ZOMBI2

Simulating the Evolution of Species, Genomes, Sequences and Traits

Adrián A. Davín

Version 0.9.0 — July 2026

Contents

Chapter 1

Introduction

1.1 Why simulate

Evolutionary biology infers the past from what survives into the present: a gene tree from an alignment, a rate of gene loss from a set of genomes, an ancestral body size from the sizes of living species. The true history is gone, so there is nothing to check the answer against.

Simulation is the way around this. You create a dataset whose history you already know: which lineages went extinct, which gene was transferred and when, what the sequence at each internal node was. Run a method on that dataset and you can measure how much of the history it recovers. This is how phylogenetic methods are tested, calibrated and compared.

1.2 What ZOMBI2 is

ZOMBI2 simulates four levels of evolution: **Species**, **Genomes**, **Sequences** and **Traits**. You can run one level, several in sequence, or let one drive another. It is a Python library and a command-line tool over the same engine, taking the same parameters, so a run can be written either way.

1.3 What it can do

Some questions ZOMBI2 is built to answer, each of which is one run and a comparison:

- **How well does a reconciliation method recover the truth?** Evolve gene families under duplication, transfer and loss, and you get every family's true gene tree with the event behind every node. Reconcile against the species tree and score what the method found against what happened.
- **Can a transfer be detected when the donor is gone?** Transfers in ZOMBI2 can come from lineages that later go extinct, so a gene arrives in a survivor from a donor that leaves no other trace. The event log names that donor, so you can ask how often a detection method finds a transfer whose source is no longer on the tree.
- **What does genome reduction look like in host-restricted bacteria?** Evolve a lifestyle trait, free-living or host-restricted, and let it drive the loss rate. Lineages that move inside a host shed genes faster, so you can measure how much of the genome-size pattern a method attributes to lifestyle rather than to shared ancestry.
- **Does a dating method survive a clock that is not strict?** Give the substitution rate a relaxed clock, so lineages evolve at different paces, and compare the dates a method infers from the alignment against the true node ages.

- **How accurate is ancestral sequence reconstruction?** A run records the sequence at every internal node, not just the tips, so a reconstruction can be compared residue by residue against the one that really sat there.
- **Is a trait correlation real, or an artefact of the tree?** Two traits evolving on the same tree look correlated at the tips whether or not either drives the other, because they share ancestry. Simulate with the correlation switched off, on the same tree, and you have the baseline any comparative method has to beat.
- **Does a trait actually drive diversification?** Let a trait set the speciation rate, so the tree and the trait grow together, and test whether a state-dependent method recovers the effect — or reports one when the trait is inert.
- **What signal survives in gene order?** At the ordered resolution, inversions, transpositions and translocations rearrange genes on chromosomes, and fissions and fusions change the karyotype itself, so synteny and rearrangement methods can be tested against the moves that were actually made.

1.4 Installing it

ZOMBI2 needs Python 3.10 or newer and depends only on NumPy, so there is no compiler and no toolchain to set up.

```
pip install zombi2
```

`zombi2 --version` confirms the install, and `zombi2 -h` lists the commands, one per level.

1.5 For the impatient

Two commands: build a species tree, then evolve gene families along it.

```
# 1. a species tree: 20 surviving species from a birth-death process
zombi2 species out/ --birth 1 --death 0.3 --n-extant 20 --seed 1

# 2. gene families along it, by duplication, transfer, loss and origination
zombi2 genomes out/ \
    --duplication 0.2 --transfer 0.1 --loss 0.25 --origination 0.5 --seed 42
```

`out/` now holds one directory per level:

```
out/species/    species_complete.nwk   the tree, extinct lineages kept
                species_extant.nwk     the survivors only
                species_events.tsv     every speciation and extinction, with its time
                species.log            what was run, and with which parameters
out/genomes/    genome_events.tsv      every duplication, transfer, loss and origination
                profiles.tsv           gene-family copy numbers per species
                genomes.log
```

Each level keeps to its own directory, run log included, and outputs that run to one file per gene family get a directory of their own inside it, so a few hundred families stay legible. `--flat` writes everything straight into `out/` instead; `--quiet` turns off the progress bar.

Here is the same run from Python:

```
from zombi2 import species, genomes

sp  = species.simulate_species_tree(birth=1.0, death=0.3, n_extant=20, seed=1)
gen = genomes.simulate_genomes_family(sp, duplication=0.2, transfer=0.1,
                                      loss=0.25, origination=0.5, seed=42)

sp.write("out/")
gen.write("out/")
```

Chapter 2

A tour of ZOMBI2

This chapter introduces the four levels ZOMBI2 simulates, the three ways they can relate, and the single shape every rate takes.

2.1 The four levels of ZOMBI2

- **Species** — the tree of lineages: a strictly bifurcating rooted tree, with branches measured in time. Every workflow has a species tree, so this is the first thing to run.
- **Genomes** — the genes that exist in each lineage. Genomes can be simulated at three **resolutions**: gene families alone, genes placed on chromosomes, or a full nucleotide genome. A genome is always simulated within a species tree.
- **Sequences** — the nucleotides or amino acids inside each gene. Sequences evolve on gene trees, so genomes must be simulated first.
- **Traits** — phenotypes evolving along a tree: body size, optimal growth temperature, the presence or absence of a flagellum.

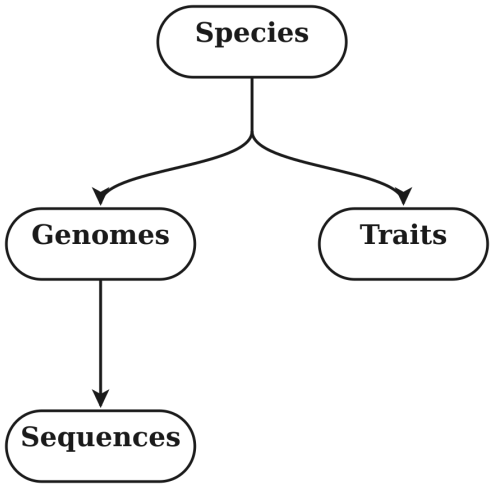

Figure 2.1. The four levels of ZOMBI2. Everything starts from the species tree, which forks: a genome evolves along it, and so does a trait. Sequences continue below genomes, because a sequence lives inside a gene.

A run in which every level is simulated:

$$P(\text{Species}) \cdot P(\text{Genomes} \mid \text{Species}) \cdot P(\text{Sequences} \mid \text{Genomes}) \cdot P(\text{Traits} \mid \text{Species})$$

You need not run them all. Skip sequences if you only want gene trees, which the genome level already produces:

$$P(\text{Species}) \cdot P(\text{Genomes} \mid \text{Species})$$

Skip genomes if you want a species tree with traits on it:

$$P(\text{Species}) \cdot P(\text{Traits} \mid \text{Species})$$

Everything depends on a species tree, so a workflow almost always begins by simulating one alone. The exception is a **joint** model, below, where the tree is an output rather than an input.

2.2 Time

ZOMBI2 is a forward simulator: evolution runs from an ancestral state at time 0 to the present.

Time is imposed by the species tree, and every rate is measured against that scale. If your tree runs from 0 at the root to 1 at the tips, the simulation lasts one unit of time. Time 0 is the origin of the founding lineage, and every time you give ZOMBI2 — the moment of a mass extinction, the breakpoints of a rate that changes through time — is measured on it.

The founding lineage lives for a while before it first splits. That stretch is the tree's **stem**, and the first split is its **crown**.

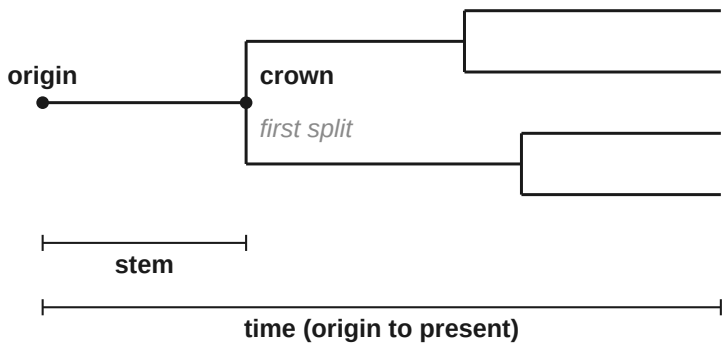

Figure 2.2. The stem. A run starts from one lineage at time 0 — the origin — which lives for a while before it first splits. Everything up to that split is the stem, and it is ordinary simulated time: genes are gained and lost along it, traits drift along it. Every tree ZOMBI2 writes therefore gives its root a branch length.

2.3 Rates

Everything is driven by events that fire over time:

- **Species** — speciations and extinctions
- **Genomes** — duplications, transfers, losses, originations, inversions, transpositions
- **Sequences** — substitutions

- **Traits** — phenotypic changes

How often an event fires is its **effective rate**:

$$\text{effective rate} = \text{scope(base)} \times \text{modifiers}$$

The **base** is the speed of a single event (how fast), in units of inverse time. The **scope** wraps it to say how many independent chances the event has: per lineage, per copy, per site. The **modifiers** are dimensionless multipliers that make a rate faster or slower depending on context — the lineage, the gene family, the total diversity present.

You rarely need to touch the default scope or the modifiers. With them you can simulate:

- an early burst that tapers off, by giving the rate a schedule that starts high and drops;
- a molecular clock that speeds and slows along the tree, with close relatives ticking alike, by letting each lineage inherit its rate from its parent and drift at every split;
- a radiation that eases off as the clade fills up, by having the speciation rate read the diversity present at each moment.

Appendix A is the full rate reference — the units, each level's default scope, the catalogue of modifiers — and the Gillespie algorithm that turns rates into events.

2.4 Going beyond: conditioning and joining levels

Some scenarios need more than the levels running in sequence:

- A trait evolves along a tree and controls how fast that tree speciates.
- Gene content decides survival: lineages that acquire a key gene diversify faster.
- Two levels feed back: a trait raises a gene family's loss rate, and carrying that family pulls on the trait.

ZOMBI2 adds two connections for these — **conditioning** and **joining**. Each has its own chapter; this is the shape of them.

Conditioning makes a parameter of one level read the state of another instead of being a fixed number. Aquatic lineages lose their olfactory genes faster: the gene-loss rate depends on the trait's value on each branch. Because the trait can be simulated first and then held fixed, this is still two ordinary runs in order — simulate the driver, write it out, feed it to the second, exactly as you already feed a species tree to a genome run.

$$P(\text{Species}) \cdot P(\text{Traits} \mid \text{Species}) \cdot P(\text{Genomes} \mid \text{Species}, \text{Traits})$$

Joining is for when neither level can go first. If a trait speeds up speciation, faster-speciating lineages leave more descendants, so the tree's shape depends on the trait — while the trait is evolving along that very tree. No order works, so both are grown together in one run, and the tree becomes an output rather than an input.

$$P(\text{Species}, \text{Traits})$$

The test is whether the influence runs one way or in a loop. One way: condition. In a loop — one level shaping another and being shaped back, directly or through the tree — join.

2.5 Using ZOMBI2 in Python

Each level is a function in its own subpackage, and they compose by feeding one level's result into the next. A run returns a *result object*; read it in the session or write it to disk with `.write()`. A whole workflow is a short script:

```
from zombi2 import species, genomes, sequences, traits
from zombi2.sequences import substitution_models as sm

sp   = species.simulate_species_tree(birth=1.0, death=0.3, n_extant=20, seed=1)
gen  = genomes.simulate_genomes_family(sp, duplication=0.2, loss=0.25, origination=0.5, seed=42)
seqs = sequences.simulate_sequences(gen, model=sm.hky85(kappa=2.0), length=300, seed=7)
bm   = traits.simulate_continuous(sp, rate=1.0, seed=1)
```

Each call takes the object it depends on — genomes and traits read the species result, sequences reads the genomes result — so the script reads top to bottom in the `P(·)` order above.

2.6 Using ZOMBI2 from the CLI

The same simulations run from the command line. Each level is a subcommand of `zombi2`, and its flags are the long-form names of the Python arguments:

```
# a dated species tree (20 extant tips)
zombi2 species out/ --birth 1 --death 0.3 --n-extant 20 --seed 1

# gene families along it
zombi2 genomes out/ --duplication 0.2 --loss 0.25 --origination 0.5 --seed 42
```

A rate flag takes a rate **written exactly as you would write it in Python** — a bare number, or a scope wrapper and modifiers composed with `*`, quoted so the shell keeps it in one piece:

```
# speciation drops to a third of its rate at time 3 (a skyline)
zombi2 species out/ --birth "1.0 * OnTime({0: 1.0, 3: 0.3})" --death 0.3 --total-time 5 --seed 1
```

Every command takes one positional argument, the **run directory**. It is both where that command writes and where it reads the level before it, so a pipeline is the same directory named once per command and nothing is passed by hand. `--from` overrides the reading half, for a tree from elsewhere or a run you would rather not write into; a `--params` TOML file can hold a whole pipeline's settings.

Because the levels share one directory, a command refuses to re-run a level in place when a later level was built from it — that would leave the later output out of step. `--force` re-runs anyway and removes the now-stale downstream. The CLI covers all four levels; the coupled models are run from Python until their commands land.

2.7 Output in ZOMBI2

Every run can be written with `result.write("out/", outputs=[...])`; with no `outputs` it writes that level's **default** set. The formats are uniform — **trees** in Newick, **tables and event logs** in TSV, **sequences** in FASTA. Branch lengths are in time everywhere except the sequence phylograms, which are in substitutions per site.

At every level the **event log** (`*_events.tsv`) is the true, ordered history the run followed: the source of truth the summaries are derived from. Appendix B lists every file, level by level.

Chapter 3

Species trees

The species tree is the backbone every other level runs on, so it is where almost every workflow begins. This chapter is about making one.

3.1 The birth–death process

A species tree grows by two kinds of event: a lineage **speciates**, splitting in two, or it **goes extinct** and stops. You give a **speciation rate** and an **extinction rate**, and every lineage alive at a given moment has the same constant chance per unit time of splitting or dying, independently of the rest.

The two rates set the tempo. Their difference fixes how fast diversity builds up. Their ratio fixes how much of the history is hidden, because a lineage that goes extinct takes its part of the tree with it. With extinction set to zero nothing is ever lost, and the tree you get is the whole tree that grew: this is the classic **Yule** (pure-birth) process. As extinction rises, the tree of survivors becomes a thinner trace of the one that actually grew. The lineages that died are kept: they are in the complete tree, and *Outputs* below says where.

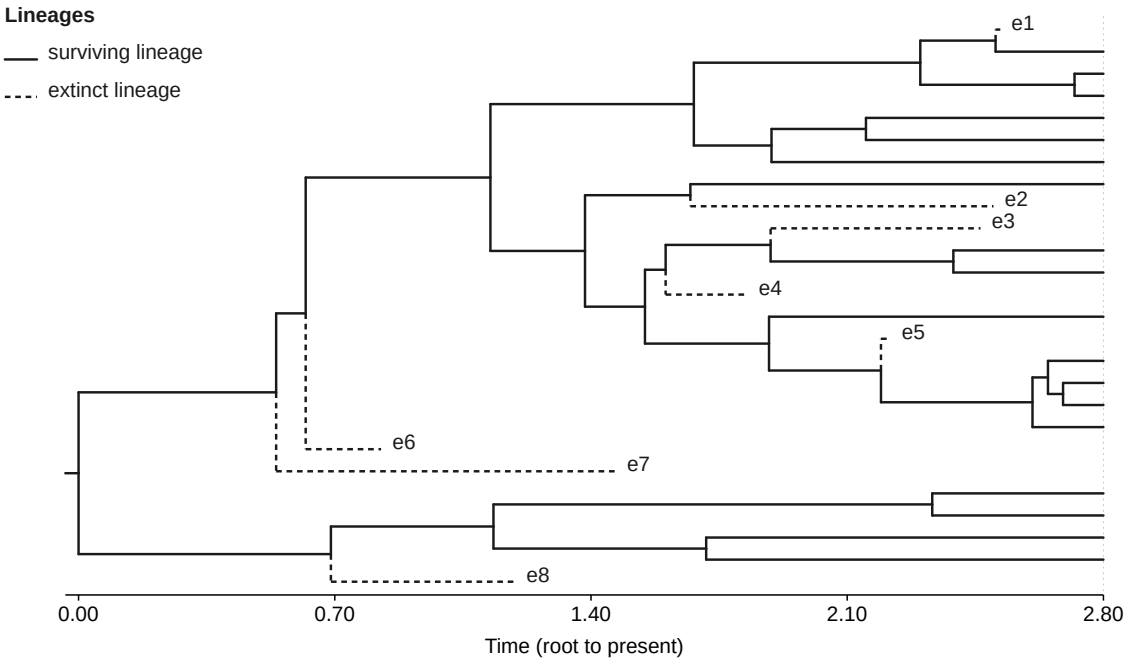

Figure 3.1. A species tree grown by the birth–death process. Every lineage alive at a given moment has the same chance per unit time of splitting or of dying. The lineages that died are drawn dashed and stop where they died; the survivors reach the present. Both are in the complete tree, and only the solid ones are in the extant tree.

You also say when to stop: grow to a fixed **total time** (`total_time`), or to a fixed **number of surviving lineages** (`n_extant`).

They differ in one practical way. `n_extant` bounds the run by construction; `total_time` does not, because standing diversity grows like $\exp((\text{birth} - \text{death}) \cdot t)$, so a rate slightly too high or a time slightly too long is the difference between a thousand lineages and ten million. A run that passes **100,000 standing lineages** therefore stops with an error rather than filling memory. Raise `max_lineages` if that is the size you want, or set it to `None` to lift the guard. It never truncates: a tree cut off at a size is no longer a sample from the process you asked for, so handing one back would be worse than not running at all.

```
from zombi2 import species
# a birth–death tree of 20 surviving lineages
result = species.simulate_species_tree(birth=1.0, death=0.3, n_extant=20, seed=1)
```

3.2 Simulating species trees with variable rates

A birth or death rate need not be constant. It can depend on **time**, on **how crowded the tree is**, or on a lineage's **ancestry**, and you express each the same way: multiply the base rate by a **modifier** naming what it depends on.

- **On time** — the rates change at set points in time. This is the skyline, or episodic, tree.

`birth = 1.0 * mod.OnTime({0: 1.0, 3: 0.3})` runs at full rate until time 3, then at a third of it.

- **On total diversity** — the rate slows as the tree fills up, so diversity levels off at a carrying capacity instead of growing without bound: `birth = 1.0 * mod.OnTotalDiversity(cap=100)`.
- **On the parent's rate** — each lineage inherits its parent's rate, nudged at every split, so rates wander across the tree and close relatives resemble each other: `birth = 1.0 * mod.FromParent(spread=0.2)`.

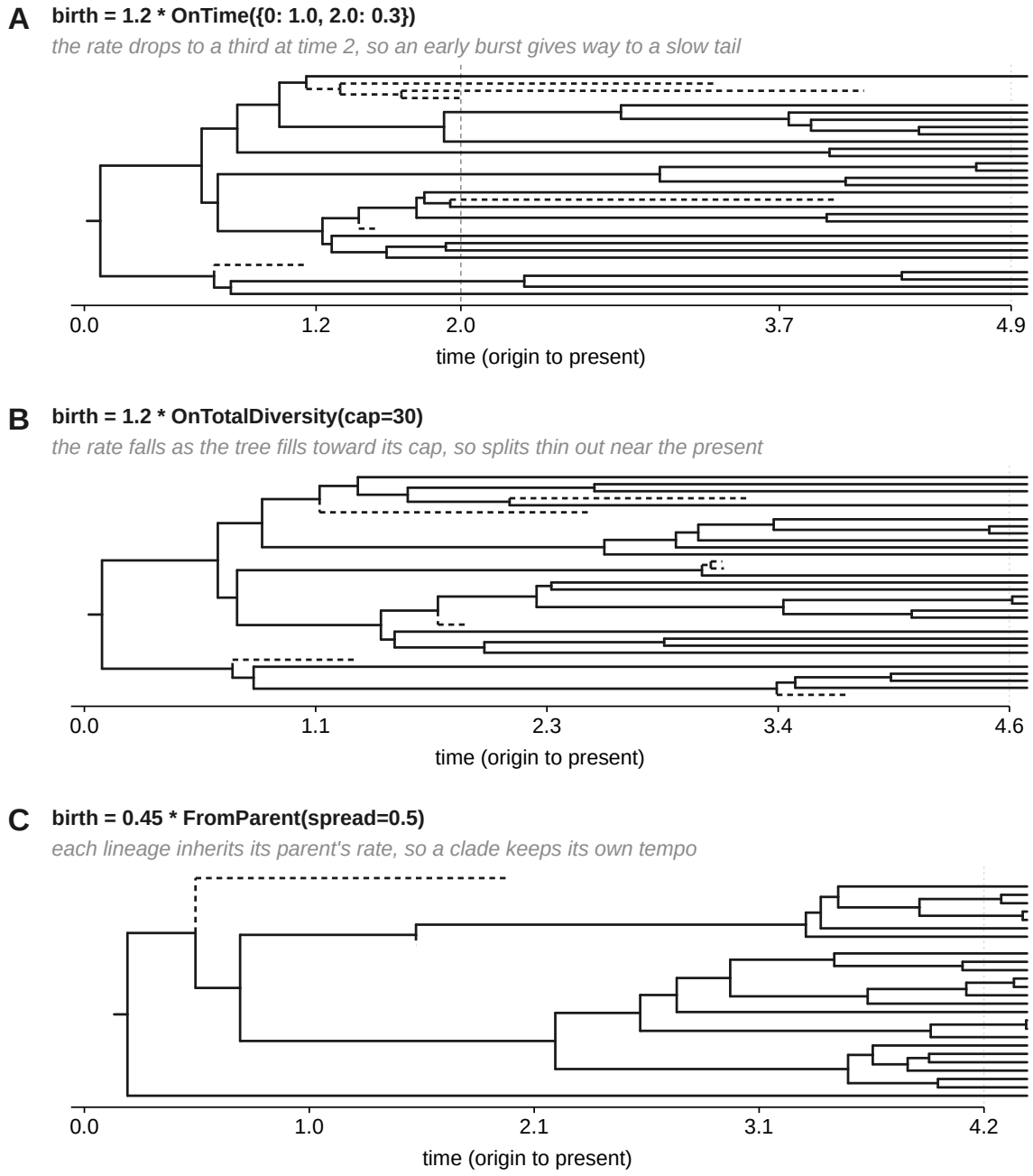

Figure 3.2. Three ways a rate can vary, one tree apiece — all three stopped at the same 25 surviving lineages, so what differs is how they got there. **A** `OnTime`: the rate drops at time 2, so an early burst gives way to a long slow tail. **B** `OnTotalDiversity`: the rate falls as the tree fills toward its cap, and splits thin out near the present. **C** `FromParent`: each lineage inherits its parent's rate, so one clade radiates late while its sister stays sparse. Solid lineages survive to the present and dashed ones died, as in the previous figure.

The modifiers live in `zombi2.rates.modifiers`. Each is a dimensionless factor on the base rate, and you can stack them with `*` to get a rate that changes in time *and* saturates.

Birth and death are modified independently. Give both a `FromParent` and each lineage draws its own speciation factor and its own extinction factor at every split, so the two rates drift without

correlation.

3.3 Other models

One model does not fit the modifier framework: a **mass extinction**, where at one instant only a fraction of the living lineages survive. `mass_extinctions=[(3.0, 0.75)]` kills three-quarters of those alive at time 3. It is a pulse rather than a steady rate, so it is its own argument.

Figure 3.3. A mass extinction as a survival pulse. The tree grows under a constant birth–death process until, at one instant, a fraction of the standing lineages die together — the cohort of dots along the vertical wall. Survivors are solid and extinct lineages dashed. The lineages-through-time curve below shares the time axis and shows the diversity crash at the pulse and the recovery after it. This tree was grown with `mass_extinctions=[(2.5, 0.75)]`.

3.4 A summary of models

What it does	Here	From the literature
rates change at set times	`1.0 * mod.OnTime({…})`	skyline / episodic birth–death
rate slows as the tree fills	`1.0 * mod.OnTotalDiversity(cap=…)`	diversity-dependent diversification
rates drift, inherited at each split	`1.0 * mod.FromParent(spread=…)`	ClaDS
a fraction culled at an instant	`mass_extinctions=[(t, f)]`	mass extinction

3.5 Sampling

Two more choices decide not how the tree grows, but how much of it you get to see.

By default you see every surviving species. **`sampling`** keeps a random fraction of the extant tips, so `sampling=0.5` gives you half. It thins a tree that has already grown, so it costs nothing.

`fossils` does the opposite: it recovers lineages from the past. Fossils are picked up along **every** branch of the complete tree at a rate you set — a surviving lineage's branch as readily as an extinct one — so `fossils=0.1` scatters dated samples through its history. They are a side output, reported alongside the trees; a fossil does not remove its lineage and does not appear in the extant tree.

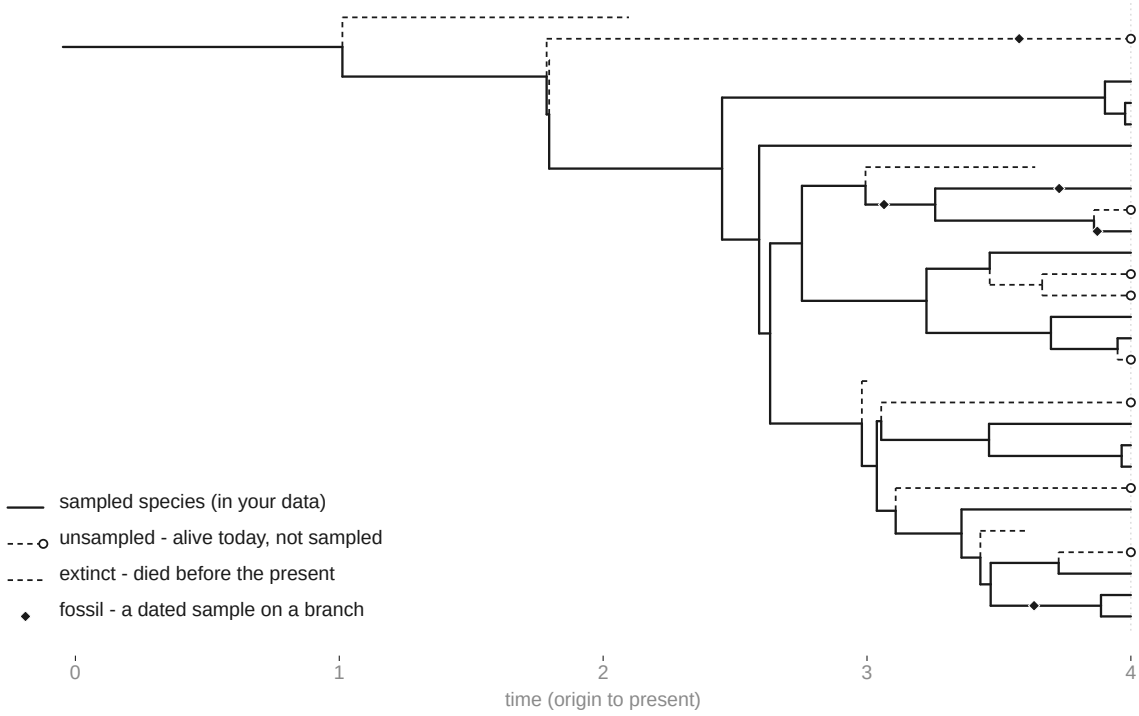

Figure 3.4. Sampling and fossils, the two ways a dataset falls short of the whole tree. A single complete tree shows every lineage's fate. Sampled species reach the present as solid lines and are the data you keep. Lineages alive today but not sampled reach the present as dashed lines ending in an open ring. Lineages that went extinct are dashed and stop where they died. Fossils are dated samples recovered along any branch of the complete tree, a surviving lineage's branch as readily as an extinct one, shown as black diamonds. The data is the solid tips together with the diamonds; the dashed lineages are never observed. This tree was grown with `sampling=0.6, fossils=0.15`.

```
# see only half the survivors
result = species.simulate_species_tree(birth=1.0, death=0.3, n_extant=20, sampling=0.5, seed=1)

# recover fossils of extinct lineages along the branches
result = species.simulate_species_tree(birth=1.0, death=0.3, total_time=6.0, fossils=0.1, seed=2)
```

3.6 The `SpeciesResult` object

`simulate_species_tree` returns a **`SpeciesResult`**, which carries:

- `.extant_tree` — the survivors' tree, dated and bifurcating; this is what you get by default and hand to the next level.
- `.complete_tree` — the whole tree that grew, with the extinct lineages still on it.
- `.fossils` — the sampled fossil lineages and their ages, present when you asked for `fossils`.
- `.events` — the event log: every speciation and extinction with its time, the source of truth the run exists to record.

As at every level it also carries `.seed` and `.write(dir, outputs=[...])`. Each tree carries its topology and dated branch lengths, and lets you ask for its tips, its internal nodes, and which are extant.

3.7 Usage from Python

The whole range is one function call:

```
from zombi2 import species
from zombi2.rates import scope, modifiers as mod   # the rate grammar: scopes + modifiers

# constant-rate birth–death (per lineage, the default)
result = species.simulate_species_tree(birth=1.0, death=0.3, n_extant=20, seed=1)

# Yule (pure birth) — death defaults to 0
result = species.simulate_species_tree(birth=1.0, n_extant=50, seed=1)

# skyline birth that also slows with diversity, with a global death rate
result = species.simulate_species_tree(
    birth = 1.0 * mod.OnTime({0: 1.0, 3: 0.5}) * mod.OnTotalDiversity(cap=100),
    death = scope.Global(0.3), total_time=8.0, seed=2)

# a mass extinction and incomplete sampling
result = species.simulate_species_tree(
    birth=1.0, death=0.3, mass_extinctions=[(3.0, 0.75)],
    sampling=0.5, total_time=5.0, seed=2)
```

3.8 Usage from the CLI

The command mirrors the Python call — the base rates, the stop condition, and the sampling and fossil knobs each have a flag:

```
# a birth–death tree of 20 surviving lineages
zombi2 species out/ --birth 1.0 --death 0.3 --n-extant 20 --seed 1

# grow to time 5, with a mass extinction at time 3 and half the survivors sampled
zombi2 species out/ --birth 1.0 --death 0.4 --total-time 5 --mass-extinction 3 0.75 --sampling 0.5 --seed
```

3.9 Outputs

A run writes two Newick trees by default: the **extant** tree of survivors (`species_extant.nwk`) and the **complete** tree, which also carries the extinct and unsampled lineages (`species_complete.nwk`). The survivors are tips of both trees; the dead and unsampled are tips of the complete tree only.

Both trees give the root a branch length, which many simulators leave off. A run begins with a single lineage and that lineage lives for a while before it first splits, so the root's branch is the **stem**: the time from the origin to the crown. It is ordinary simulated time — genes are gained and lost along it, traits drift along it — and a tree written without it would start at the crown and lose that history. In the complete tree the stem runs from the origin to the first speciation; in the extant tree it runs from the origin to the most recent common ancestor of the survivors, absorbing whatever branches were pruned away above it.

The **event log** (`species_events.tsv`) is always written: every speciation and extinction with its time. It is the ground truth the simulator exists to record. If you asked for fossils, the sampled fossil lineages are written too.

All of them land in `out/species/`; `--flat` writes them straight into `out/` instead. Appendix B lists every file.

Chapter 4

Genomes I: gene families

Genomes live inside the species tree, and they can be simulated at three **resolutions**, one per chapter: **family** here, **ordered** in Chapter 5, **nucleotide** in Chapter 6. The simplest is a gene family evolving along the tree; the most detailed tracks every nucleotide across several chromosomes.

The **family** resolution is genomes made of gene families and nothing more: no position along a chromosome, no DNA sequence. Genes are copied, lost, born from nothing, and passed sideways between lineages.

4.1 The four events

A genome at the family resolution evolves by four kinds of event, applied to every lineage as it runs down the species tree:

- **Duplication** — a gene copy is copied, so its family gains a member in that lineage.
- **Transfer** — a copy jumps from one lineage to another that is alive at the same moment. This is the only event that crosses lineages, and it gets its own section below.
- **Loss** — a gene copy is deleted; a family that loses its last copy is gone from that lineage.
- **Origination** — a brand-new family appears in a lineage, with one copy.

You give a rate for each, and the events play out along the tree from the initial genome, speciation handing a lineage's genome down to both children. Out comes the genome of *every* lineage together with the event log that produced it.

```
from zombi2 import species, genomes
from zombi2.rates import modifiers as mod

tree = species.simulate_species_tree(birth=1.0, death=0.3, n_extant=20, seed=1)
g = genomes.simulate_genomes_family(
    tree, duplication=0.2, loss=0.25, origination=0.5, initial_families=20, seed=1)
```

The root starts with `initial_families` families of one copy each, recorded as originations at the crown; from there the four rates drive everything.

4.1.1 Families that differ from one another

A bare rate is shared by every family: give `loss=0.25` and all of them lose at 0.25. Real families do not behave alike, and **`ByFamily`** is how you say so — the family twin of the sequence level's `ByLineage`. Each family draws one multiplier, mean-corrected so `E[factor] = 1`, which lets you widen the spread without moving the average family off the base rate.

Where you put it decides what varies *together*:

```
# each rate varies by family on its own — a family that loses fast is not thereby duplicating fast
g = genomes.simulate_genomes_family(
    tree,
    duplication = 0.2  * mod.ByFamily(spread=0.6),
    transfer    = 0.1  * mod.ByFamily(spread=0.6),
    loss        = 0.25 * mod.ByFamily(spread=0.6),
    initial_families = 100, seed = 42)

# one tempo per family, scaling every rate it has — a fast family is fast at everything
g = genomes.simulate_genomes_family(
    tree, duplication=0.2, transfer=0.1, loss=0.25,
    family_speed = mod.ByFamily(spread=0.5),
    initial_families = 100, seed = 42)
```

The two compose: `family_speed` for a family's overall tempo, and a `ByFamily` on one rate for extra variation particular to it. On the command line the rate keeps its written form, `--loss "0.25 * ByFamily(spread=0.6)"`.

4.1.2 How large a family may get

Growth compounds: a duplication rate above the loss rate multiplies without bound, and with `ByFamily` some families draw a rate well above the one you typed. So a family's copies **within one genome** are capped, and the cap is on by default.

```
max_family_size = 10.0     # the default: ten times the lineages in the complete tree
max_family_size = 50       # an int is that number of copies, whatever the tree
max_family_size = None     # no ceiling
```

A float scales with the run, so the same setting means the same thing on a tree of ten species and one of a thousand; an int is absolute. At the cap the family stops duplicating, and the ceiling holds for arrivals too, so a transfer cannot push it past sideways.

4.2 What the rate depends on

The rates follow the **same grammar as the species level** (`base` optionally wrapped in a scope, optionally multiplied by modifiers). The scope answers *per what*, and the default is the natural one for each event. Duplication, transfer, and loss are counted **per copy**: a family with ten copies is ten times as likely to duplicate or lose one as a family with a single copy, which is what you want — more genes, more chances. Origination is counted **per lineage**: acquiring a wholly new family is a property of the lineage, not of any gene it already has.

Rates can also depend on **time**. Multiplying a base rate by an `OnTime` modifier makes it change at set moments — the skyline, or episodic, genome: fast early and slow later, or any schedule you give.

```
# lots of new families early, then origination shuts off after time 2
g = genomes.simulate_genomes_family(tree, origination=1.0 * mod.OnTime({0: 1.0, 2: 0.0}), seed=1)
```

4.3 Lateral gene transfers

Transfer is the one event that couples lineages, and it is what makes the family resolution more than four independent birth–death processes. When a transfer fires, a copy is picked from the whole pool of live genes, and it is delivered to another lineage that is **alive at that same instant**.

Three arguments shape what a transfer does:

- **`transfer_to`** — who receives. `"uniform"` (the default) picks any other contemporaneous lineage with equal chance; `"distance"` makes closer relatives likelier, weighting recipients by how far they sit from the donor on the tree. The distance version is *scale-free*. `Clades(...)` weights recipients by **named clades of the tree** — see below.
- **`replacement`** — what happens on arrival. By default the incoming copy is **additive**: the recipient simply gains a copy. With `replacement=True` it **overwrites** a copy of the same family already present, and falls back to additive when the recipient has none.
- **`self_transfer`** — whether a lineage may donate to itself. Off by default. With additive arrival the lineage gains a copy, so the gene content changes as it would under a duplication, but the event is recorded as a transfer.

```
tree = species.simulate_species_tree(birth=1.0, death=0.4, n_extant=30, seed=7)
# horizontal transfer biased toward close relatives, overwriting resident copies
g = genomes.simulate_genomes_family(
    tree, transfer=0.5, transfer_to="distance", replacement=True,
    origination=0.4, initial_families=10, seed=3)
```

One consequence is worth stating plainly: a transfer can arrive **from a lineage that later goes extinct**. A genome run happens on the complete tree, dead branches included, so a gene can enter a survivor from a donor that leaves no other trace.

4.3.1 Transfer between named clades

`"distance"` biases transfer by relatedness, but sometimes you want to name the groups yourself — "let genes flow between these two clades, and nowhere else." `Clades` does that. You name each clade — by a few of its tips (the clade is the subtree below their MRCA) or by a node id — and give a `Between` kernel: a weight for each ordered **(donor clade, recipient clade)** pair.

```
from zombi2 import species, genomes

sp = species.simulate_species_tree(birth=1.0, death=0.3, n_extant=16, seed=1)
# genes flow only BETWEEN clade A and clade B — never within either, never to the rest
g = genomes.simulate_genomes_family(
    sp, transfer=1.0, initial_families=20, seed=2,
    transfer_to=genomes.Clades({"A": ["n27", "n28"], "B": ["n21", "n26"]},
                               genomes.Between({("A", "B"): 1.0, ("B", "A"): 1.0}, default=0.0)))
```

The kernel is the new part. Each entry is a weight, read the same way `"distance"`'s weights are: normalised over the lineages alive at the instant a transfer fires. Naming only `("A", "B")` and `("B", "A")` and setting `default=0.0` means every other pairing weighs 0 — a clade-A donor can reach clade B but not another clade-A lineage, and the rest of the tree neither sends nor receives. Drop the `default=0.0` and unlisted pairs return to weight 1 (baseline), so `Between({("A", "B"): 5.0})` *enriches* A→B fivefold while leaving everything else to happen normally. A weight of 0 means

"cannot receive", exactly as at the end of Chapter 9: when a donor's every candidate weighs 0, the transfer has nowhere to land and does not fire.

A clade here is a fact about the *tree* — which lineage sits in which subtree — so `Clades` reads the tree directly, needs no extra file, and is a sibling of `"distance"`, not a coupling. When the groups are instead an evolved property — a habitat, an ecological guild — the same donor-and-recipient steering is a coupling, written `transfer_to = DrivenBy(trait, Between({...}))`; that is Chapter 9.

4.4 The `FamilyGenomesResult` object

`simulate_genomes_family` returns a **FamilyGenomesResult** which carries:

- `.complete_tree` — the species tree the genomes ran on, extinct lineages and all.
- `.genomes` — a dict from node to that node's genome.
- `.initial_genome` — the genome the run **started** with, at the root lineage's origination. It is not `.genomes[root]`: a node sits at the **end** of its branch, and the root branch is real simulated time, so events happen along it. Written to its own `initial_genome.tsv`, with no `lineage` column, because it belongs to no node.
- `.events` — the event log: every gene event with its time and lineage — origination, duplication, transfer, loss, and the *speciations* at a split.
- `.profiles` — the family × extant-species copy-count table.
- `.gene_trees` — one `GeneTree` per family.
- `.seed` — the seed, so the run reproduces.

and two methods:

- `.family_counts(node_id)` — a `Counter` collapsing a node's genome to `family → number of copies`, when you want the multiset rather than the individual copies.
- `.write(dir)` — materialise the outputs to disk: the event log (`genome_events.tsv`) and the profiles (`profiles.tsv`).

```
n5 = g.genomes[5]                    # the gene copies in node n5
counts = g.family_counts(5)          # {family: copies} for the same node
g.write("out/")                      # genome_events.tsv + profiles.tsv
```

4.5 Profiles and gene trees

Two products are derived from the run's recorded history.

Profiles are the classic comparative-genomics view: how many copies of each gene family sit in each extant species. They are read off the observed genomes on access, so the run itself stays lean.

```
g.profiles.matrix        # families × extant-species copy counts, a NumPy array
g.profiles.presence      # the same as 0/1 presence/absence
g.profiles.to_tsv()      # the table as text
```

Gene trees are the deeper output. Every family has two trees: the `.complete` tree with every gene lineage, and the `.extant` tree pruned to the genes that survive.

```
gt = g.gene_trees[7]                 # the gene tree of family 7
gt.to_newick("extant")               # the surviving copies as Newick ...
```

```
gt.to_newick("complete")              # ... or the whole genealogy
gt.origination                        # when the family began
```

The root of a gene tree carries a branch length, as the species tree's does. A family starts at its origination, and the founding gene lives a while before its first duplication, transfer or speciation; that wait is the root's branch, the family's stem. A gene that originated and never split at all is a one-node tree, written as its own lifespan — `g55:0.263097;`.

4.6 Usage from Python

The whole range is one function call:

```
from zombi2 import species, genomes
from zombi2.rates import modifiers as mod

tree = species.simulate_species_tree(birth=1.0, death=0.3, n_extant=30, seed=1)

# a plain duplication–loss–origination run
g = genomes.simulate_genomes_family(
    tree, duplication=0.2, loss=0.25, origination=0.5, initial_families=20, seed=1)

# origination only — every family stays a single copy, none is ever duplicated
g = genomes.simulate_genomes_family(tree, origination=0.6, seed=1)

# a skyline: new families pour in early, then origination shuts off after time 2
g = genomes.simulate_genomes_family(tree, origination=1.0 * mod.OnTime({0: 1.0, 2: 0.0}), seed=1)

# horizontal transfer, biased toward close relatives, overwriting resident copies
g = genomes.simulate_genomes_family(
    tree, transfer=0.5, transfer_to="distance", replacement=True,
    origination=0.4, initial_families=10, seed=3)

# the genomes you observe are the extant tips
observed = {n.id: g.genomes[n.id] for n in g.complete_tree.extant()}

# and the outputs, derived from that history
g.profiles.matrix                                 # family × extant-species copy counts
some_family = next(iter(g.gene_trees))
g.gene_trees[some_family].to_newick("extant")     # that family's surviving gene tree
g.write("out/")                                   # the event log + profiles, on disk
```

4.7 Usage from the CLI

`zombi2 genomes` evolves gene families along a species tree read from a Newick file. The family resolution is the default, and each rate is a plain number:

```
# duplication–loss–origination along a species tree
zombi2 genomes out/ --duplication 0.2 --loss 0.25 --origination 0.5 --seed 1
```

```
# horizontal transfer biased toward close relatives, overwriting resident copies
zombi2 genomes out/ --transfer 0.5 --transfer-to distance --replacement --origination 0.4 --seed 3
```

4.8 Outputs

A run writes the event log and the profiles:

```
out/genome_events.tsv    the gene genealogy (the source of truth)
out/profiles.tsv         family × extant-species copy counts
```

Two more come with them, one row per gene copy and one Newick per family:

```
out/genomes.tsv                    every node's genes, ancestors included
out/gene_tree_fam<f>_*.nwk         each family's genealogy, complete and extant
```

`genomes.tsv` is one row per gene copy, so a lineage carrying six genes has six rows:

```
lineage  family  copy
n0       0       g0
n0       1       g1
n0       2       g2
n1       0       g3
n1       0       g8
```

Read a list of rows sharing a `lineage` and you have that lineage's genome. `family` says which gene family a copy belongs to, so the two `n1` rows above are two copies of family `0`, one of them a duplicate. `copy` is the individual gene, written `g<id>` — the same token the gene-tree leaf and the alignment header carry, so a row joins straight onto a tree or a sequence — and it is the same identifier the event log uses, so any gene here can be traced back to the event that made it. `profiles.tsv` is this same information counted rather than listed, and only for the extant tips; `genomes.tsv` keeps the ancestors, the root included, which is what you want if you are scoring a reconstruction of ancestral gene content.

4.9 Evolving families in parallel

Because families are independent — a transfer moves a copy between lineages, but no event ever mixes two families — a run can evolve them **concurrently**, one family per worker process. It is off by default; `parallel` turns it on.

```
g = genomes.simulate_genomes_family(tree, duplication=0.2, loss=0.25, origination=0.5,
                                    initial_families=1000, seed=1, parallel=8)   # 8 workers
```

`parallel=True` uses every core and an integer sets the worker count; on the command line it is `--parallel` for all cores or `--parallel 8` for eight. It is a **separate engine**, not a faster path through the default one: each family draws from its own random stream, so the result is identical for any worker count, but it differs from a serial run of the same seed — both are valid draws of the same process. A driven rate or `transfer_to` (Chapter 9) is not handled yet; a run that uses one says so and falls back to serial.

The gain is modest and a few workers is the sweet spot: the simulation splits across cores, but stitching the per-family logs back into one run stays serial, so past a handful there is little more to win. It pays off on a large run — many families, or high rates — and costs on a small one. From a script, because it starts worker processes, guard the entry point with `if __name__ == "__main__":`; the `zombi2` command already does.

When the families themselves are the scale — hundreds of thousands, a million — even the parallel run's *result* stops fitting in memory. `stream_to` writes each family to a directory as it finishes and hands back a light path handle instead of a `FamilyGenomesResult`, so memory stays flat however many families you run: 2 GB held in memory streams in about 40 MB. Pick the files with `outputs=`, exactly as `.write` takes them; the disk is then the handoff to the sequence level. On the command line it is `--stream`.

```
run = genomes.simulate_genomes_family(tree, origination=2.0, initial_families=5000, seed=1,
                                      parallel=8, stream_to="out/", outputs=("events", "profiles"))
run.path("events")            # out/genome_events.tsv — the log, ready to replay
```

Chapter 5

Genomes II: ordered

The previous chapter put genes on the tree as a *bag of families* — how many copies of each, and nothing more. This chapter gives them **structure**. A genome becomes one or more **chromosomes**, each an ordered run of genes, and each gene knows which way it points. This is the **ordered** resolution.

```
from zombi2 import species, genomes

tree = species.simulate_species_tree(birth=1.0, death=0.1, n_extant=4, seed=2)
g = genomes.simulate_genomes_ordered(
    tree, duplication=0.3, loss=0.2, origination=0.15, inversion=0.5,
    chromosomes=1, initial_families=5, seed=2)
```

Reading one extant leaf:

```
leaf n2, chromosome 2 (circular):  [ 0+ 1+ 3+ 3+ 4- ]
```

Each gene is written as its family with the strand as `+` or `-` (the strand is the integer `+1` or `-1`). This leaf has one chromosome of five genes, in which family `3` sits in a run of two tandem copies and family `4` points backwards, left that way by an inversion. The gene tree of a family is unchanged from Chapter 4 — the true genealogy, read off the same event log:

```
g.gene_trees[0].to_newick("extant")
# (((g24:0.0396561,g28:0.0396561)speciation_n3:0.405992,g19:0.445648)speciation_n1:0.2334,g9:0.679048)spe
```

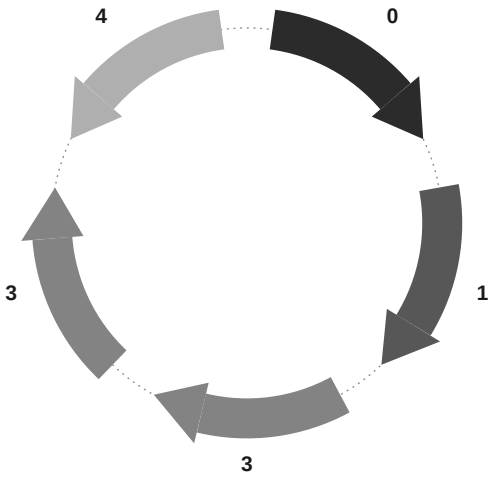

Figure 5.1. Leaf `n2`, the same chromosome `[ 0+ 1+ 3+ 3+ 4− ]` drawn as the ring it is. Each gene is an arrow that points the way its strand reads, and its shade marks its family. The two copies of family `3` are a tandem duplication — one shade, side by side; family `4`, left backward by an inversion, is the one arrow pointing against the flow.

5.1 The karyotype

A genome has a **karyotype**: `chromosomes=N` chromosomes, each with a `topology` — `"circular"` (the default) or `"linear"`, or a per-chromosome list like `["circular", "linear"]` for a mixed set. The founding `initial_families` genes are dealt round-robin across them. Topology is not just a label: it decides where a segmental event stops, which the section on segments below takes up.

5.2 Chromosomes split, merge, appear and die

On top of the karyotype, four events change the **number** of chromosomes:

- **`fission`** *(per chromosome)* — a chromosome splits in two.
- **`fusion`** *(per chromosome)* — two chromosomes of a genome merge into one.
- **`chromosome_origination`** *(per lineage)* — a de-novo replicon appears: a new chromosome, empty and circular, a plasmid.
- **`chromosome_loss`** *(per chromosome)* — a whole chromosome dies, and every gene on it is recorded as a loss. A lineage never loses its *last* chromosome this way.

```
tree = species.simulate_species_tree(birth=1.0, death=0.1, n_extant=3, seed=42)
g = genomes.simulate_genomes_ordered(
    tree, duplication=0.15, loss=0.1, origination=0.25,
    chromosomes=2, fission=0.25, fusion=0.25,
    chromosome_origination=0.03, chromosome_loss=0.03,
    initial_families=5, seed=42)
```

5.3 The chromosome network

Chromosomes are tracked. A chromosome id is re-minted at every event that reshapes it — a speciation, a fission, a fusion — and each of those edges is recorded. So the run leaves behind not just the chromosomes at the tips but the *genealogy* that connects them: the **chromosome network**. It is the middle of three tiers that nest, the species tree containing the chromosome network, which contains the gene trees:

```
species tree  ⊃  chromosome network  ⊃  gene trees
```

It is a **network** and not a tree because of one event: **fusion joins two chromosome lineages into one**, two parents and one child. Fission and speciation are ordinary splits (one parent, two children); origination is a root; loss is a leaf. The whole thing is a directed graph, and it is recorded the way graphs are, as an **edge list** — `chromosome_events`, one row per event. The run above gives:

```
  time   kind          parents -> children
  0.00   origination         -  -> 0          an initial chromosome
  0.00   origination         -  -> 1          an initial chromosome
  0.97   loss                1  -> -          chromosome 1 (and its genes) dies
  2.19   speciation          0  -> 2, 3
  3.27   speciation          2  -> 4, 5
  3.35   fission             4  -> 6, 7        a bifurcation
  3.67   fusion           6, 7  -> 8           a reticulation (two parents)
```

5.4 Events act on segments

Once genes have neighbours, **a gene-level event acts on a segment** — a run of consecutive genes — not on one gene. A duplication copies a segment, a loss removes one, a transfer sends one sideways. That produces the signature of real genome evolution: neighbouring genes sharing a history because they were copied, moved or lost *together*.

How much does an event take? That is its **extent**, set per event type as `<event>_extent`. A bare number is the **mean**, so `duplication_extent=3` copies about three adjacent genes — often two or three, sometimes one, occasionally many more. Write `Fixed(3)` for exactly three every time, or name any other distribution. The default is a single gene, so out of the box every event touches one gene and you recover the simplest behaviour.

```
from zombi2.rates.distributions import Geometric

tree = species.simulate_species_tree(birth=1.0, death=0.1, n_extant=3, seed=27)
g = genomes.simulate_genomes_ordered(
    tree, duplication=0.35, loss=0.3,
    duplication_extent=Geometric(mean=3),      # duplications copy ~3 adjacent genes at once
    chromosomes=1, initial_families=5, seed=27)
```

```
leaf n2:  [ 4+ 1+ 4+ 1+ 2+ 3+ ]
            └────┘ └────┘

            the segment 4 1, duplicated as a unit and landed in tandem
```

The segment `4 1` appears twice: a single segmental duplication copied those two adjacent genes together. (Family `0` is absent — it was lost earlier, which is what left `4` and `1` next to each other.) A duplication puts its copy **in tandem**, immediately after the original run; a transferred segment arrives together on the recipient. **Origination is the exception**: a family is born once, as a single new gene, so it has no extent.

5.4.1 How often, where, how much

A segmental event answers three questions, and it takes two numbers to describe one:

- **how often** it starts — the rate;
- **where** it starts — a gene drawn from the genome;
- **how much** it takes — the extent.

In Chapter 4 the third answer was always "one gene", so the rate was the whole story. Here it is not, and the two numbers **multiply**.

One consequence catches people out: **a rate counts starts, not hits.** `duplication=0.2` with `duplication_extent=3` does *not* mean each gene is duplicated 0.2 times per unit time. A gene is copied whenever any event begins on a run that covers it — which is about three times as often, so roughly `0.2 × 3 = 0.6`. If you want genes duplicated at a known rate, divide that rate by the mean extent.

The same reading holds at the nucleotide resolution of Chapter 6, where the extent is in base pairs rather than genes.

5.4.2 Families that differ, once events cover several at once

`ByFamily` and `family_speed` work here as they do in Chapter 4, but with one difference that matters. A run covers several families at once, so the weight is applied to **the run, averaged over the genes it covers** — not to the gene the run happened to start on.

Weighting the starting gene is the obvious implementation and the wrong one: a fast family's rate would then apply to whatever sat beside it, so you would be describing the *neighbourhood* of a fast family rather than the family — and the neighbourhood is reshuffled by every inversion and translocation, so the parameter would not name a stable thing. Averaging over the run keeps what you wrote true: a run of heavily-weighted genes is favoured, a mixed one sits between, an ordinary one is unweighted.

With no weights set every run averages to one, so a run using neither knob is unchanged.

5.5 Rearrangements: inversion, transposition, translocation

Three more events reshape the order without creating or destroying genes:

- **Inversion** — reverse a segment in place, flipping every gene's strand: `+2 +3 +4` becomes `-4 -3 -2`. On a circular chromosome the segment may span the origin.
- **Transposition** — cut a segment out and reinsert it **elsewhere on the same chromosome**.
- **Translocation** — move a segment to a **different chromosome** of the same genome. A no-op if the genome has only one chromosome.

A moved segment — transposed or translocated — lands **inverted** with probability `inversion_probability` (default `0`, so it keeps its orientation).

```
tree = species.simulate_species_tree(birth=1.0, death=0.1, n_extant=3, seed=0)
g = genomes.simulate_genomes_ordered(
    tree, duplication=0.15, loss=0.15, origination=0.1,
    inversion=0.3, transposition=0.25, translocation=0.2,
    transposition_extent=Geometric(mean=2), inversion_probability=0.5,
    chromosomes=2, initial_families=6, seed=0)
```

5.6 The `OrderedGenomesResult` object

`simulate_genomes_ordered` returns an **`OrderedGenomesResult`** — the `FamilyGenomesResult` spine, with the structured extras:

- `.complete_tree` — the species tree the genomes ran on, extinct lineages included.
- `.genomes` — a dict from node id to that node's genome, now a tuple of **`Chromosome`** objects. Each `Chromosome` has an `id`, a `topology`, and an ordered list of **`Gene`** objects (`id`, `family`, `strand`).
- `.initial_genome` — the genome the run **started** with, at the root lineage's origination. It is not `.genomes[root]`: a node sits at the **end** of its branch, and the root branch is real simulated time, so events happen along it. Written to its own `initial_genome.tsv`, with no `lineage` column, because it belongs to no node.
- `.events` — the gene-genealogy log, exactly as in Chapter 4, from which `.gene_trees` and `.profiles` are derived unchanged. Position and orientation are *not* here; they live in the genomes and the two logs below.
- `.rearrangements` — the inversion / transposition / translocation log.
- `.chromosome_events` — the chromosome network, as an edge list.
- `.gene_trees`, `.profiles`, `.seed` — as before.

with the methods `.family_counts(node_id)` (the multiset view), `.gene_order(node_id)` (the layout — `(chromosome, position, strand, family, gene id)` per gene), and `.write(dir, outputs=[...])`.

```
g.genomes[2]                    # the chromosomes of node n2
g.gene_order(2)                 # its layout, gene by gene
g.chromosome_events             # the chromosome network, as an edge list
g.gene_trees[0].to_newick()     # a family's gene tree — unchanged from the family resolution
```

5.7 Usage from Python

```
from zombi2 import species, genomes
from zombi2.rates.distributions import Geometric
from zombi2.rates import modifiers as mod

tree = species.simulate_species_tree(birth=1.0, death=0.2, n_extant=30, seed=1)

# several chromosomes that split, merge, and gain a plasmid
g = genomes.simulate_genomes_ordered(
    tree, chromosomes=6, topology="linear",
```

```
    fission=0.05, fusion=0.05, chromosome_origination=0.02, chromosome_loss=0.02,
    origination=0.4, initial_families=20, seed=1)

# ordered genome: the Chapter 4 events, plus inversions, on a single chromosome
g = genomes.simulate_genomes_ordered(
    tree, duplication=0.2, loss=0.2, origination=0.3, inversion=0.3,
    chromosomes=1, initial_families=20, seed=1)

# segmental everything: duplications, losses and inversions act on segments of genes
g = genomes.simulate_genomes_ordered(
    tree, duplication=0.2, loss=0.25, inversion=0.3,
    duplication_extent=Geometric(mean=4), loss_extent=Geometric(mean=3),
    inversion_extent=Geometric(mean=5), initial_families=15, seed=1)

# rearrangements: relocate and move segments between chromosomes, sometimes inverting them
g = genomes.simulate_genomes_ordered(
    tree, duplication=0.2, transposition=0.2, translocation=0.2,
    inversion_probability=0.5, chromosomes=3, initial_families=15, seed=1)

# rates can still depend on time (the skyline), as at every level
g = genomes.simulate_genomes_ordered(
    tree, inversion=1.0 * mod.OnTime({0: 1.0, 2: 0.2}), initial_families=10, seed=1)

# the outputs
g.genomes                              # every node's chromosomes
g.gene_order(next(iter(g.genomes)))    # a node's layout
g.chromosome_events                    # the chromosome network
g.rearrangements                       # the inversion/transposition/translocation log
g.gene_trees                           # one gene tree per family, as in Chapter 4
```

5.8 Usage from the CLI

The ordered resolution is `--resolution ordered`. It adds the chromosome and segment flags to the Chapter 4 events, each still a plain number:

```
# chromosomes split and merge along the tree
zombi2 genomes out/ --resolution ordered \
    --origination 0.5 --fission 0.05 --fusion 0.05 --chromosomes 2 --seed 1

# segmental duplications, losses and inversions on three chromosomes
zombi2 genomes out/ --resolution ordered \
    --duplication 0.2 --loss 0.2 --origination 0.5 --inversion 0.3 --chromosomes 3 --seed 1

# segments relocate and move between chromosomes, sometimes inverting
zombi2 genomes out/ --resolution ordered \
    --origination 0.5 --transposition 0.2 --translocation 0.1 \
    --inversion-probability 0.5 --chromosomes 2 --seed 1
```

5.9 Outputs

`.write(dir, outputs=[...])` materialises the chosen products to disk:

```
g.write("out/", outputs=("events", "profiles", "gene_order",
                         "rearrangements", "chromosome_events"))
```

```
out/genome_events.tsv        the whole history: the genealogy, where each event
                             happened, and the rearrangements — in time order
out/profiles.tsv             family × extant-species copy counts
out/gene_order.tsv           every node's layout, one row per gene
out/chromosome_events.tsv    the chromosome network (edge list)
```

chromosome_events.tsvis the network's ground truth, its columns the edge list above —time · kind · lineage · parents · children‘, one row per event.

The full list of files lives in Appendix B.

Chapter 6

Genomes III: nucleotide

At the **nucleotide** resolution a chromosome is a **coordinate axis of DNA** rather than a list of gene tokens, and an event is an arc on it: an inversion reverses 600 bp, a loss deletes 900 bp, a duplication copies 2 kb in tandem. Genes still exist and still get gene trees, but they are stretches of that axis with a start and an end, and the DNA between them is simulated too.

```
from zombi2 import species, genomes

tree = species.simulate_species_tree(birth=1.0, death=0.1, n_extant=4, seed=2)
g = genomes.simulate_genomes_nucleotide(
    tree, root_length=3000, genes=3, gene_length=400,
    inversion=1.0, inversion_extent=600,
    duplication=0.3, duplication_extent=300, loss=0.3, loss_extent=300, seed=2)
```

This starts the run from a 3000 bp circular chromosome carrying three 400 bp genes, evenly spaced, and evolves it down the tree.

6.1 Blocks: genes and intergenes

A chromosome is stored as an ordered list of **blocks**. A block is a run of DNA with one unbroken ancestry: the interval `[start, end)` on some ancestral **source**, read forward (+) or reverse-complemented (−). Events only ever *split* blocks, never merge them, so the block boundaries you see at a leaf are the accumulated breakpoints of its whole history.

Blocks come in two kinds:

- a **gene** is a *declared, indivisible* block — one family, one id, **never split**. It carries a gene tree.
- an **intergene** is the spacer between genes. It fragments freely into as many blocks as events dictate.

A genome is therefore an alternating chain of intergenes and genes, and either extreme is legal. Declare no genes and the chromosome is one big intergene — the uniform-sequence model. Fill the replicon with genes and there is no spacer at all; that evolves too, because events break at the joins *between* genes (see *Genes are never split*, below).

Reading two leaves of the run above shows what the events did:

```
leaf n2, chromosome 2 (circular), 3000 bp
  [   0,  600) +   600 bp  intergene
  [ 600, 1000) +   400 bp  gene 1
  [1000, 1144) +   144 bp  intergene
  [1144, 1374) −   230 bp  intergene
```

```
leaf n5, chromosome 5 (circular), 3000 bp
  [   0,  599) +   599 bp  intergene
  [1174, 1374) +   200 bp  intergene
  [1000, 1144) −   144 bp  intergene
  [ 600, 1000) −   400 bp  gene 1
```

```
[1374, 1404) +    30 bp  intergene
[2000, 2262) -   262 bp  intergene
[1600, 2000) -   400 bp  gene 2
[1404, 1600) -   196 bp  intergene
[2262, 2600) +   338 bp  intergene
[2600, 3000) +   400 bp  gene 3
```

```
[ 599,  600) -     1 bp  intergene
[1144, 1174) -    30 bp  intergene
[1374, 1600) +   226 bp  intergene
[1600, 2000) +   400 bp  gene 2
[2000, 2021) +    21 bp  intergene
[2021, 2319) -   298 bp  intergene
[2319, 2600) +   281 bp  intergene
[2600, 3000) +   400 bp  gene 3
```

Both leaves still carry all three genes. In `n2` an inversion covered gene 2, which now reads on the `-` strand with the spacer around it reversed; in `n5` a different inversion covered gene 1 instead. The coordinates are what make this readable: every block still names where it came from in the root, so `[600, 1000)` is gene 1 wherever it turns up and whichever way it points.

6.2 Genes are never split

An event either **engulfs a gene whole** or leaves it alone; a breakpoint never falls strictly inside one. So an event does not pick an arc and then clean up afterwards. Both of its ends are drawn **directly from the positions where a breakpoint is legal**. A genome can therefore be **all gene, with no spacer at all**: ten 100 bp genes in 1000 bp is a legal genome, and it evolves. Its breakpoints simply all fall at the joins between genes, so genes are inverted, moved, duplicated and lost whole. Genes may sit flush; they are not required to leave a gap.

6.3 Extents

There is one important consequence: **the realised extent is not always the extent you asked for.** It is quantised to the legal breakpoints, and on a gene-dense genome the difference is large. Take thirty-one 3000 bp genes in a 100 kb genome — 93% genic, so the spacers are about 200 bp — and ask for an inversion of:

asked	realised
500 bp	59
1 500 bp	1 000
3 000 bp	2 916
10 000 bp	10 315

A 500 bp event has nowhere to go but inside a spacer, so it comes out at 59 bp. Long events land near what you asked for, because they can span whole genes.

The correction runs both ways. When every legal end lies *further* out than you asked — a long gene sitting just past the start — the arc snaps to the nearest legal breakpoint, which is **longer** than the extent you set. That is the `10 000 → 10 315` row above. When the genome cannot give what you asked at all, the arc is capped by the replicon and comes out shorter. Either way the event still fires: an extent is a request, and the genome answers it.

The one case that yields no event is degenerate — a replicon with no legal end within reach at all, such as one under 2 bp — where the event is skipped rather than forced.

6.4 A note on rates

Every rate here is per lineage. The rate sets how often a lineage does the event; the extent (above) sets how much DNA it touches. Keeping the rate per lineage means the number you type reads the same whatever the genome's size: a rate counted per base pair would rise as the genome grew, so `inversion=5.0` would mean one thing at 10 kb and another at 1 Mb. Per lineage, the event count stays flat as the genome grows — the same tree at `inversion=5.0`, with the genome a hundred times longer each row, gives:

```
   10 000 bp  ->  77 inversions
  100 000 bp  ->  77 inversions
1 000 000 bp  ->  77 inversions
```

The chromosome tier below is the exception: `fission`, `fusion` and `chromosome_loss` are counted **per chromosome**, and `chromosome_origination` per lineage.

Rates here take the same written form as everywhere else — `scope(base) × modifiers` — and the scopes above are the defaults, so a bare number stays a bare number. The **skyline** works: `inversion = 5.0 * OnTime({0: 1.0, 3: 0.2})` drops the inversion rate fivefold at time 3, and the run re-reads its rates at each step rather than racing past it.

So does **conditioning**. Every rate here takes a `DrivenBy`, so a trait can drive how much DNA a lineage sheds — genome reduction as it is usually meant — and can drive the rearrangements too:

```
from zombi2 import traits
from zombi2.rates import modifiers as mod

habitat = traits.simulate_discrete(tree, states=["host", "free"], switch=0.8, seed=2)
loss = 0.8 * mod.DrivenBy(habitat, {"host": 20.0, "free": 0.5})
```

The extent takes the same modifiers, and that is a different statement:

```
loss        = 0.8 * mod.DrivenBy(habitat, {"host": 20.0, "free": 0.5})   # deletes more often
loss_extent = 150 * mod.DrivenBy(habitat, {"host": 6.0,  "free": 1.0})   # deletes in bigger chunks
```

The first raises how often a host-restricted lineage deletes, the second how much each deletion takes. Set both and they multiply: the DNA shed per unit time goes up by the product, not the sum.

A modifier on an *extent* is read when an event fires, so unlike the same modifier on a rate it adds no step to the run's clock. Chapter 9 covers what a driver is and how to grow one; anything not wired raises rather than being quietly ignored.

6.5 The initial genome

The **initial genome** — the genome the run starts from, at time 0, before any event — is declared in one of two ways. (It is not the same as the genome at the root *node*: the root branch is real simulated time, so that one already carries stem events. See `.initial_genome` below.)

Evenly spaced genes — `genes=N, gene_length=L` lays down `N` genes of `L` bp on each replicon, spreading the leftover DNA evenly between them. Good for controlled experiments, since gene density is then a number you set.

A GFF file — `gff="genome.gff"` takes exact coordinates from a real annotation. `##sequence-region` gives each replicon's extent, `gene` features give coordinates, strand and name, and other feature types are ignored. Names land in `result.gene_names`, so you can follow a named gene through the run. `gff=` and `genes=` are mutually exclusive; a GFF already declares the genes.

6.6 The `NucleotideGenomesResult` object

`simulate_genomes_nucleotide` returns a **`NucleotideGenomesResult`**:

- `.complete_tree` — the species tree the genomes ran on, extinct lineages included.
- `.genomes` — a dict from node id to that node's `NucleotideGenome`, a list of `Chromosome`s, each a list of `Block`s.
- `.initial_genome` — the genome the run **started** with, at the root lineage's origination. It is not `.genomes[root]`: a node sits at the **end** of its branch, and the root branch is real simulated time, so events happen along it. Written to its own `initial_genome.tsv`, with no `lineage` column, because it belongs to no node. It votes on the root partition like every other genome, so `.initial_assembly()` rebuilds it too.
- `.events` — the copy-lineage genealogy: origination, loss, duplication, transfer, speciation.
- `.rearrangements` — the ancestry-neutral log: inversion, transposition, translocation.
- `.chromosome_events` — the chromosome network, as in Chapter 5.
- `.gene_spans` — `{family: (source, start, end)}`, where each declared gene sits in initial coordinates.
- `.gene_names`, `.gene_strands` — a named gene's family id, and its coding strand, from a GFF.
- `.gene_trees` — one recovered gene tree per gene family, for the families that survive in at least one extant leaf.
- `.root_blocks` — the recovered root partition: the maximal never-cut intervals that some node still carries. Cut at **every** node's breakpoints, not just the survivors', which is what lets any node's genome be rebuilt.
- `.block_trees` — a recovered tree for **every** root block, spacer as well as gene, keyed by its index in `.root_blocks`.
- `.initial_assembly()` — the same for `.initial_genome`, as `(block, strand)` pairs. No gene id: the initial genome predates every event, so each block has exactly one sequence there.
- `.block_of(family)` — the block index a declared gene family occupies: the join between the two numbering schemes here, since `.gene_spans` and `.gene_trees` are keyed by family id while `.root_blocks` and `.block_trees` are keyed by block index. Both are plain integers, so mixing them up is silent; this is how you avoid it.
- `.seed`.

and four ways to read one node's genome, at four grains:

```
g.mosaic(2)        # per block:      {chromosome: [(source, start, end, strand), ...]}
g.trace_back(2)    # per nucleotide: {chromosome: [(source, position, strand), ...]}
g.ancestry(2)      # the multiset of ancestral (source, position) it still carries
g.assembly(2)      # per piece:      {chromosome: [(block, gene, start, end, strand), ...]}
```

`ancestry` is the invariant worth knowing: rearrangements conserve it exactly, so an inversion-only run leaves every leaf holding a permutation of the initial sequence. Loss makes it a subset, and duplication, transfer and origination add to it.

6.7 Usage from Python

```
from zombi2 import species, genomes

tree = species.simulate_species_tree(birth=1.0, death=0.1, n_extant=6, seed=4)

# rearrangement only — every leaf is a permutation of the initial sequence
g = genomes.simulate_genomes_nucleotide(
    tree, root_length=100_000, inversion=5.0, inversion_extent=1000, seed=4)

# a genic genome with real turnover: genes duplicate, are lost, transfer, and arise
g = genomes.simulate_genomes_nucleotide(
    tree, root_length=6000, genes=6, gene_length=400,
    duplication=2.0, duplication_extent=900, loss=2.0, loss_extent=900,
    transfer=1.0, transfer_extent=900, transfer_to="distance",
    origination=0.5, origination_extent=400, seed=4)

# a karyotype that splits and merges, with a plasmid
g = genomes.simulate_genomes_nucleotide(
    tree, chromosomes=3, root_length=4000, genes=6, gene_length=200,
    fission=0.2, fusion=0.2, chromosome_origination=0.05, chromosome_loss=0.05,
    inversion=1.0, seed=5)

# a real annotation as the initial genome
g = genomes.simulate_genomes_nucleotide(
    tree, gff="ecoli.gff", inversion=2.0, inversion_extent=5000,
    loss=1.0, loss_extent=3000, seed=1)

# the outputs
g.gene_spans                            # where each gene sits, in initial coordinates
family = min(g.gene_trees)              # gene_trees holds only the families that survive
g.gene_trees[family].to_newick("extant")
leaf = next(n.id for n in g.complete_tree.extant())
g.mosaic(leaf)                          # that leaf's genome, block by block
g.chromosome_events                     # the chromosome network
```

A family lost in **every** extant lineage still gets a complete tree — it is still history — but `.gene_trees[fam].extant` is `None` and no `_extant.nwk` is written. Only a family deleted from every node has no tree at all, and it still appears in `.gene_spans` because it was declared. Ask `.gene_trees` what it holds rather than assuming a declared family is in it.

6.8 Usage from the CLI

`--resolution nucleotide` takes the same event rates as Chapter 5 and adds two things: how to set up the initial genome, and how long an event is in base pairs.

```
# an evenly spaced initial genome: 5 kb, six genes of 300 bp, with inversions averaging 400 bp
zombi2 genomes out/ --resolution nucleotide \
```

```
  --root-length 5000 --genes 6 --gene-length 300 \
  --inversion 1.0 --inversion-extent 400 --duplication 0.3 --loss 0.3 --seed 1

# or start from a real genome: the GFF declares the replicons and the genes,
# and a paired FASTA supplies the actual DNA those coordinates hold
zombi2 genomes out/ --resolution nucleotide \
  --gff ecoli.gff --fasta ecoli.fasta --inversion 0.5 --loss 0.4 --loss-extent 900 --seed 1
```

Every event kind has its own `--<event>-extent`, the mean of a geometric draw in base pairs: `--inversion-extent`, `--loss-extent`, `--duplication-extent`, `--transfer-extent`, `--transposition-extent`, `--translocation-extent`, `--origination-extent`.

6.9 Outputs

```
zombi2 genomes out/ --resolution nucleotide --root-length 5000
```

```
out/genome_events.tsv    the whole history: the copy-lineage genealogy and the
                         rearrangements — in time order
out/genes.tsv            where each gene sits in the root, and on which strand
out/blocks.tsv           every node's genome as its block mosaic
out/initial_genome.tsv   the genome the run started with
out/gene_trees/          one Newick per family, complete and extant
out/genome_<lineage>.gff · .bed   every genome's genes and blocks
out/chromosome_events.tsv  the chromosome network's edges
```

Everything is written by default.

Chapter 7

Sequence evolution

The sequence level does two main things:

- It rescales the gene trees and the species tree from time into substitutions per site (**phylograms**).
- It evolves the residues that sit inside every gene, so each family ends with an alignment.

The sequence level always follows a genome run, and takes that run's result directly:

```
from zombi2 import species, genomes, sequences
from zombi2.sequences.substitution_models import hky85
from zombi2.rates import modifiers as mod

tree = species.simulate_species_tree(birth=1.0, death=0.3, n_extant=20, seed=1)
my_genomes = genomes.simulate_genomes_family(tree, duplication=0.2, transfer=0.1,
                                             loss=0.25, origination=0.5, seed=1)
result = sequences.simulate_sequences(my_genomes, model=hky85(kappa=2.0),
                                      length=1000, seed=1)
```

7.1 Creating phylograms

A gene tree in ZOMBI2 is by default a **chronogram**, its branch lengths measure time. What a sequence actually accumulates along a branch is not time but a number of *substitutions per site*, and that is time multiplied by an evolutionary rate. Turning one into the other is the whole job of the sequence level. Applying a rate to every branch rescales the tree from time into expected substitutions and yields a phylogram.

Two things therefore have to be chosen: *what* changes (the substitution model, the chemistry of which residue turns into which) and *how fast* it changes along each branch, which is the clock.

7.2 The substitution models

ZOMBI2 implements different standard models of sequence evolution:

```
from zombi2.sequences.substitution_models import (jc69, k80, hky85, gtr,
                                                  poisson, dayhoff, jtt, wag, lg)

# --- nucleotide models (4 states, ACGT) ---
model = jc69()                    # equal rates, equal base frequencies — no free parameters
model = k80(kappa=2.0)            # a transition/transversion bias
model = hky85(kappa=2.0, freqs=(0.3, 0.2, 0.2, 0.3))           # bias + unequal frequencies
model = gtr(rates=(1,2,1,1,2,1), freqs=(0.25,0.25,0.25,0.25))   # six exchangeabilities + freqs
```

```
# --- protein models (20 states, amino acids) ---
model = poisson()                 # equal rates, equal frequencies — the JC69 of proteins
model = jtt()                     # Jones, Taylor & Thornton 1992
model = dayhoff()                 # Dayhoff, Schwartz & Orcutt 1978
model = wag()                     # Whelan & Goldman 2001
model = lg()                      # Le & Gascuel 2008
```

The model decides the alphabet, and `length` counts whatever that alphabet holds: bases for a nucleotide model, residues for a protein one. The nucleotide models are four different rate matrices, not one model with four settings, but they nest in the order written — `jc69` is `k80` with `kappa=1`, `k80` is `hky85` with equal base frequencies — so each step adds free parameters. The protein matrices are **empirical**, each estimated once from a large set of real alignments and then used as a fixed table, which is why they take no parameters.

7.3 Relaxed molecular clocks

The rate itself is `substitution`, and it is counted **per site**: a gene-tree branch of Δt time accrues `substitution · Δt` substitutions at every site. Leave it alone and it is `1.0` everywhere — the **strict clock**, one tempo for the whole tree.

A substitution rate that changes from lineage to lineage is what the field calls a **relaxed clock**. It is not a new kind of object here: you multiply the rate by a modifier, exactly as at every other level.

```
# strict clock — one rate everywhere; the default, so write nothing
substitution = 1.0

# relaxed — each lineage draws its own rate, independently of its neighbours
substitution = 1.0 * mod.ByLineage(spread=0.3)                # lognormal (the default)
substitution = 1.0 * mod.ByLineage(spread=0.3, dist="gamma")  # or gamma
```

`ByLineage` has *no memory*: each lineage is an independent draw, so a lineage's rate tells you nothing about its neighbours'. The distribution it draws from (`dist="lognormal"` or `"gamma"`) is a parameter of the modifier.

One important point: **the clock belongs to the species tree, not to the gene trees.**

ZOMBI2 draws one rate for each species branch. Every gene that passes through that branch then evolves at that rate. Each gene-tree branch looks up the species branch it sits inside, which the genome run already recorded. The consequence is that if a species evolves quickly, all of its genes evolve quickly together.

A reference table that can be handy to people who want to implement a specific model from the literature:

What it does	ZOMBI2	From the literature
one rate everywhere	`substitution = 1.0` (default)	Strict / global clock
each lineage i.i.d. lognormal	`1.0 * mod.ByLineage(spread=…)`	Uncorrelated lognormal (UCLN)
each lineage i.i.d. gamma	`1.0 * mod.ByLineage(spread=…, dist="gamma")`	Uncorrelated gamma (UGAM)

What it does	ZOMBI2	From the literature
each lineage i.i.d. — that *is* white-noise	`1.0 * mod.ByLineage(spread=…)`	White-noise clock

7.4 The objects

`simulate_sequences` returns a **`SequencesResult`**, which carries:

- `.alignments` — the observable data: for each family, the sequence at every **extant** gene copy. This is the alignment a phylogenetic method would be handed.
- `.ancestral` — the sequence at every node that is **not** an extant tip: internal nodes, and the tips where a copy was lost or its species died. The run wrote a sequence at each node as it went, so these are the exact ancestors, not estimates. With `.alignments` it accounts for every node of the tree exactly once, so every label in a complete phylogram names a sequence.
- `.founding` — for each family, the sequence it began with, at its origination.
- `.phylograms` — for each family, its gene tree with branch lengths converted from time into substitutions per site: the tree the sequences were drawn along.
- `.species_phylogram` — the same conversion applied to the species tree, so the clock is visible as branch lengths.
- `.genomes`, `.initial_genome` — the assembled genome of every node, and of the run's starting point, present only when the run came from a **nucleotide** genome. See below.

As with every level, the bundle also carries `.seed` and `.write(directory, outputs=[...])` to put the chosen outputs on disk.

7.4.1 Where a sequence starts

A family does not begin at the first branching of its gene tree. It begins when it originates, and the founding gene then lives for a while — its **stem** — before anything splits it. So that is where the sequence starts: one draw from the model's stationary frequencies at the origination, which then evolves across the stem in the ordinary way and arrives at the root gene as a sequence that has already changed. `.founding` is that first draw; `.ancestral` holds what the root gene ended up with, and the two differ by however much the stem allowed.

7.5 Usage from Python

An end-to-end run, from a species tree through genomes to alignments:

```
from zombi2 import species, genomes, sequences
from zombi2.rates import modifiers as mod
from zombi2.sequences.substitution_models import hky85, gtr, lg

tree = species.simulate_species_tree(birth=1.0, death=0.3, n_extant=20, seed=1).complete_tree
my_genomes = genomes.simulate_genomes_family(tree, duplication=0.2, transfer=0.1,
                                             loss=0.25, origination=0.5, seed=1)

# the common case: DNA under HKY85, a strict clock
result = sequences.simulate_sequences(my_genomes, model=hky85(kappa=2.0),
```

```
                                    length=1000, seed=1)
result.alignments          # {family: {gene copy: sequence}} — the observable data
result.ancestral           # the same, at every node
result.species_phylogram   # the species tree in substitutions per site

# GTR with unequal base frequencies, under a relaxed (uncorrelated) clock
result = sequences.simulate_sequences(my_genomes,
    model=gtr(rates=(1, 2, 1, 1, 2, 1), freqs=(0.3, 0.2, 0.2, 0.3)),
    substitution=1.0 * mod.ByLineage(spread=0.3),   # the relaxed clock
    length=500, seed=1)

# proteins under LG — 300 residues per gene
result = sequences.simulate_sequences(my_genomes, model=lg(), length=300, seed=1)
```

7.6 Running on a nucleotide genome

Hand it a **nucleotide** genome run instead you get the full fasta genomes. Genes and spacer get their own models. `model` evolves the genes; `intergene_model` evolves the spacer, at `intergene_speed` times the rate — 3× by default, and `jc69` by default, which is flat and has no free parameters.

```
from zombi2 import species, genomes, sequences
from zombi2.sequences.substitution_models import hky85

tree = species.simulate_species_tree(birth=1.0, death=0.2, n_extant=5, seed=1).complete_tree
my_genomes = genomes.simulate_genomes_nucleotide(
    tree, gff="ecoli.gff", inversion=1.0, inversion_extent=5000,
    duplication=0.3, loss=0.3, seed=1)

result = sequences.simulate_sequences(my_genomes, model=hky85(kappa=3.0),
                                      intergene_speed=3.0, substitution=0.05, seed=1)

result.genomes["n5"]             # {chromosome: sequence} — a whole assembled genome
result.genomes["n0"]             # the same at an ancestor — reconstructed, not estimated
result.initial_genome            # the genome the run started with, before anything happened
```

From the command line it is the same two commands as any other run:

```
zombi2 genomes out/ --resolution nucleotide --gff ecoli.gff --trim-overlaps \
  --inversion 5.0 --inversion-extent 50000 --loss 2.0 --loss-extent 8000 --seed 7

zombi2 sequences out/ --model hky85 --kappa 3.0 --substitution 0.02 \
  --intergene-speed 3.0 --seed 7
```

7.6.1 Starting from a real sequence

So far the founding sequence of each block is *drawn* — from the model's frequencies, random ACGT. Hand the genomes run a **FASTA** alongside the GFF and it starts from the sequence you supply instead:

```
my_genomes = genomes.simulate_genomes_nucleotide(
    tree, gff="ecoli.gff", fasta="ecoli.fasta",     # layout AND letters
    inversion=1.0, loss=0.3, seed=1)
result = sequences.simulate_sequences(my_genomes, model=hky85(kappa=3.0), substitution=0.05, seed=1)
```

The FASTA has one `>seqid` record per GFF `##sequence-region`, each exactly its declared length. Every block is then founded from the real DNA at its own initial coordinates, so an assembled genome descends from exactly what you gave. A gene that origination invents mid-run has no supplied DNA (it did not exist initially), so its block still draws from the model.

7.7 Usage from the CLI

On the command line the genome run is handed over as a **directory** — the run directory itself, which by then holds the genomes. `zombi2 sequences out/` reads that run's species tree and event log and replays the gene genealogy from them, so the two commands chain without anything else passing between them. A nucleotide run given a `--fasta` also hands its initial DNA across (in `initial_sequence.fasta`), so the sequences descend from your real sequence without you naming it twice. Point `--from` at another run to read one and write somewhere else.

```
# 1. genomes along a species tree (from the previous chapters)
zombi2 genomes out/ \
    --duplication 0.2 --transfer 0.1 --loss 0.25 --origination 0.5 --seed 1

# 2. HKY85, 1000 sites, strict clock
zombi2 sequences seqs/ --from out/ --model hky85 --kappa 2.0 \
    --length 1000 --seed 1

# GTR with unequal frequencies under a relaxed clock, also writing the ancestral sequences
zombi2 sequences seqs/ --from out/ --model gtr \
    --frequencies 0.3 0.2 0.2 0.3 \
    --substitution "1.0 * ByLineage(spread=0.3)" \
    --seed 1 --write alignments phylograms ancestral species_phylogram
```

A protein model is the same command with a different `--model`:

```
# proteins under LG, 300 residues per gene
zombi2 sequences seqs/ --from out/ --model lg --length 300 --seed 1
```

Because a protein model has no parameters, passing one is an error rather than a flag that gets quietly ignored: `--model lg --kappa 2.0` stops with *"these options don't apply to –model lg: –kappa"*.

7.8 Outputs

A run writes, by default, one **alignment** per gene family in FASTA, with the extant gene copies as the aligned rows, and the **phylograms** those sequences were drawn along — each family's gene tree in Newick, with branch lengths in substitutions per site rather than time, so the ground-truth tree behind every alignment is kept beside it.

The **clock species tree** (`clock_species_tree_complete.nwk`, and `…_extant.nwk`) comes with them:

the species tree under the same conversion, and where the clock becomes visible — a fast-evolving lineage has a longer branch there than its age alone would give it. One output is written only on request: the **ancestral sequences** (`--write ancestral`) give the sequence at every internal node, which is what you need to score an ancestral-reconstruction method against the truth.

Chapter 8

Trait evolution

The trait level evolves **phenotypes**: a body size, a habitat, the presence or absence of a structure. A trait evolves along the species tree like everything else here. There are two kinds, continuous and discrete:

```
from zombi2 import species, traits
from zombi2.rates import modifiers as mod

tree = species.simulate_species_tree(birth=1.0, death=0.3, n_extant=20, seed=1)
result = traits.simulate_continuous(tree, start=0.0, rate=1.0, seed=1)   # a real value
result = traits.simulate_discrete(tree, states=["marine", "terrestrial"],
                                  switch=0.1, seed=1)                    # a discrete state
```

8.1 Continuous traits

A continuous trait does **Brownian motion**. Give it a starting value and a diffusion rate and it wanders down every branch, its variance growing in proportion to elapsed time:

```
# BM — a body size diffusing from 0 at variance-rate σ² = 1.0
traits.simulate_continuous(tree, start=0.0, rate=1.0, seed=1)
```

Here `rate` is the Brownian variance-rate σ^2, the trait level's reading of "how fast", and like every rate in ZOMBI2 it accepts modifiers. For example, two variations on this are:

```
# OU — the same diffusion, pulled toward an optimum value
traits.simulate_continuous(tree, start=0.0, rate=1.0,
                           reverts_to=2.0, pull=0.5, seed=1)

# early burst — the diffusion rate itself decays through time
traits.simulate_continuous(tree, start=0.0,
                           rate=1.0 * mod.OnTime({0: 1.0, 5: 0.2}), seed=1)
```

The **Ornstein–Uhlenbeck** process is Brownian motion with a rubber band: `reverts_to` is the optimum it is pulled back toward, and `pull` is how hard. **Early burst** (or ACDC) is a diffusion rate that decays as the tree ages, so most of the divergence happens near the root; it is written with the same `mod.OnTime` that gives the species tree its skyline.

The rest of the modifier vocabulary applies to `rate` unchanged, each with a name in the comparative-methods literature: `mod.FromParent(spread=…)` makes σ^2 drift from parent to daughter (variable-rates BM, the trait twin of ClaDS), `mod.OnTotalDiversity(cap=…)` slows σ^2 as the clade fills. Two knobs sit alongside `rate`: `regimes=` paints a multi-optimum OU, where clades pull toward different optima (a discrete trait supplies the painting and `reverts_to` becomes one optimum per regime),

and `at_speciation=` adds a jump *at* each split rather than along the branches.

8.2 Discrete traits

A discrete trait takes a finite set of states and switches between them along the branches — a continuous-time Markov chain, the field's **Mk model**:

```
# Mk — habitat flips between two states at rate 0.1
traits.simulate_discrete(tree, states=["marine", "terrestrial"],
                         switch=0.1, start="marine", seed=1)
```

When the flips are not symmetric, replace the single rate with a small matrix of directed rates:

```
# asymmetric — gains are commoner than losses
traits.simulate_discrete(tree, states=["absent", "present"],
                         switch={"absent->present": 0.2, "present->absent": 0.05},
                         seed=1)
```

A **threshold** trait is the third case, and it is a bridge back to the continuous world. An observed discrete state can be driven by an underlying continuous **liability** that itself does Brownian motion; the state you see is which side of a threshold the liability currently sits on:

```
# threshold — a discrete state read off an underlying continuous liability
traits.simulate_discrete(tree, states=["absent", "present"],
                         liability=1.0, threshold=0.0, seed=1)
```

8.3 Correlated traits

Two traits that evolve independently are two separate calls, in either order. Two traits that drift *together* cannot be simulated one before the other, because each is entangled with the other as it unfolds. Correlation is specified as per-trait rates plus a correlation overlay:

```
traits.simulate_continuous(tree,
    start={"size": 0.0, "limb": 0.0},
    rate={"size": 1.0, "limb": 0.8},        # one variance-rate per trait
    correlation={("size", "limb"): 0.6},    # the overlay, ∈ [−1, 1]
    seed=1)
```

The overlay is a dimensionless number in `[−1, 1]`, not a covariance matrix. Under `correlation=` the per-trait rates are plain numbers.

The same overlay handles *discrete* correlation with no extra machinery, through the threshold model: give each trait a liability, correlate the liabilities, and put the thresholds on top. Correlated presence/absence characters, the setting Pagel's method was built for, are then one call:

```
traits.simulate_discrete(tree, states=["absent", "present"],
    liability={"wings": 1.0, "flight": 1.0},
    correlation={("wings", "flight"): 0.7}, threshold=0.0, seed=1)
```

8.4 Models from the literature

Trait models arrive under a thicket of names, and a reader who wants "an OU model" or "a threshold model" should be able to find it. The names live here, in one table, and organise nothing else in the chapter.

What it does	ZOMBI2	From the literature
a value diffusing	`simulate_continuous(rate=…)`	Brownian motion (BM)
diffusion pulled to an optimum	`simulate_continuous(rate=…, reverts_to=…, pull=…)`	Ornstein–Uhlenbeck (OU)
diffusion rate decays through time	`simulate_continuous(rate=1.0 * mod.OnTime({…}))`	Early burst (EB / ACDC)
diffusion rate drifts between lineages	`simulate_continuous(rate=1.0 * mod.FromParent(spread=…))`	Variable-rates BM
diffusion rate slows as the clade fills	`simulate_continuous(rate=1.0 * mod.OnTotalDiversity(cap=…))`	Diversity-dependent / ecological limits
the optimum differs between painted clades	`simulate_continuous(regimes=…, reverts_to={…}, pull=…)`	Multi-optimum OU (OUM)
the value jumps at each split	`at_speciation=…` (either kind)	Cladogenetic / punctuational change
traits evolving together	one `simulate_continuous(rate={…}, correlation={…})` call	Multivariate BM
a discrete state switching	`simulate_discrete(states=…, switch=…)`	Mk (k-state Markov)
discrete driven by continuous liability	`simulate_discrete(liability=…, threshold=…)`	Threshold / liability (Wright–Felsenstein)
discrete traits evolving together	`simulate_discrete(liability={…}, correlation={…})`	Correlated binary / Pagel

8.5 The objects

A run returns a **`TraitsResult`** bundle:

- `.values` — the observable vector: the trait's value at each **extant tip**. This is the comparative-data matrix a method would be handed.
- `.node_values` — the value at **every** node (extant, extinct, and internal alike), the true ancestors at each split, from the same process that produced the tips.
- `.events` — the timestamped event log, the same shape as the genome level's: each entry is a change on a lineage at a time, from one state to another, and its `kind` is `on_branch` (a switch along a branch) or `on_speciation` (a jump at a split). For a discrete trait this log is the source of truth. A continuous trait diffuses with no along-branch events, so its log holds only the `at_speciation` jumps and is empty without them.
- `.history` — for a **discrete** trait, the per-branch stochastic character map derived from that log: the ordered list of `(state, duration)` segments each branch passed through. It is `None` for a continuous trait, which has no map, and for a threshold trait, whose liability crossings are un-timed.

For discrete traits the stored values are the state labels you gave (not integer indices), so `.values` and `.node_values` already read back in your own vocabulary.

8.6 Usage from Python

```
from zombi2 import species, traits
from zombi2.rates import modifiers as mod

# a species tree from the previous chapters, then a trait riding along it
# (the complete tree — a trait evolves on extinct lineages too)
tree = species.simulate_species_tree(birth=1.0, death=0.3, n_extant=30, seed=1).complete_tree

# continuous: body size under Brownian motion
size = traits.simulate_continuous(tree, start=0.0, rate=1.0, seed=1)
size.values                 # {extant tip: float}

# continuous: an Ornstein–Uhlenbeck trait pulled toward an optimum of 2
temp = traits.simulate_continuous(tree, start=0.0, rate=1.0,
                                  reverts_to=2.0, pull=0.5, seed=1)

# discrete: habitat flipping between two states
habitat = traits.simulate_discrete(tree, states=["marine", "terrestrial"],
                                   switch=0.1, start="marine", seed=1)
habitat.values              # {extant tip: "marine" | "terrestrial"}
habitat.events              # the realized flips, in time order

# two continuous traits that drift together — one joint call
bodyplan = traits.simulate_continuous(tree,
    start={"size": 0.0, "limb": 0.0},
    rate={"size": 1.0, "limb": 0.8},
    correlation={("size", "limb"): 0.6}, seed=1)
```

8.7 Usage from the CLI

The state space is `--kind`, and it is required. It decides which of the other flags apply — `--rate` and the OU knobs belong to a continuous trait, `--states` and `--switch` to a discrete one — so there is no default that would not silently pick a model for you:

```
# a continuous (Brownian) trait along a species tree
zombi2 traits out/ --kind continuous \
    --start 0.0 --rate 1.0 --seed 1

# a discrete two-state trait
zombi2 traits out/ --kind discrete \
    --states marine,terrestrial --switch 0.1 --seed 1

# the same, also writing the driver file a conditioned genome run reads (Chapter 9)
zombi2 traits out/ --kind discrete \
    --states cave,surface --switch 0.1 --seed 1 \
    --write values events tree
```

The trait evolves on the **complete** tree, extinct lineages included, so `species_complete.nwk` is the file to hand it. An external tree works too; if it is not ultrametric you must declare each tip's fate with `--tip-fates`, because ZOMBI will not guess which early-ending tips are extinct.

8.8 Outputs

A run writes the **trait values** at the extant tips (`trait_values.tsv`, the observable comparative-data vector) and the **trait tree** (`trait_tree.nwk`, the complete tree with every node annotated `[&trait=…]`, which opens in FigTree or iTOL). Because the value at every node comes from the same process that produced the tips, that annotated tree carries the *exact* ancestral states, not a reconstruction.

For a discrete trait the command also writes the **event log** (`trait_events.tsv`): a `root` row giving the state at t=0, then every realized transition with its time — the ground truth against which an ancestral-state or stochastic-mapping method would be scored. That one file is *also* what a conditioned genome run reads to let a trait drive its rates: given the shared tree, the root state plus the switches reconstruct the trait on every lineage at every instant, so no separate driver file is needed. The full list of files lives in Appendix B.

Chapter 9

Conditioning and joining

The book has so far run the levels one at a time: species tree, then genomes, then sequences. They already depend on the tree, but sometimes you want more than that background dependency. There are two ways: conditioning and joining.

Conditioning takes a value of one level and makes it drive a rate in another. Three examples:

- **Cave fish lose their eyes.** A habitat trait, cave or surface, has already evolved down the species tree. Wherever a lineage sits in the dark, its genes are lost four times faster than on the surface.
- **Endosymbionts shed their genomes.** A lifestyle trait, free-living or host-restricted, drives gene loss across the board, so the lineages that moved inside a host are the ones that end up with the small genomes.
- **Competent bacteria pick genes up.** A trait for natural competence raises the rate at which new gene families appear in a lineage, so the trait leaves its mark on gene *gain* rather than on loss.

In each case one level's value is read by another level's rate. The value read is the **driver**; the rate reading it is the **target**. They are not interchangeable, and the asymmetry is the point: a driver is a *value* that already varies from lineage to lineage — a habitat state, a gene count — while a target is a *rate*, a "how often", multiplied by a factor the driver's value picks out. The arrow runs one way, and nothing flows back.

That one-wayness is what makes conditioning cheap. The habitat is unaffected by how many genes a lineage has, so the trait can be grown on its own and written to a file before the genome run that reads it starts. Two ordinary commands, in order.

Joining simulates two levels **at once**, because that ordering is no longer available. Suppose a trait drives **speciation** itself: large-bodied lineages split twice as fast as small ones. You cannot grow the trait first, because a trait is grown *along a tree* and the tree is what this trait shapes. Nor the tree first, because its branching rate needs a trait value that does not exist yet. Neither can be finished before the other starts, so neither can be a file handed on. They are grown together, in one run whose Gillespie races speciation, extinction and trait change against one another, each event reading the other level's current state.

So the whole chapter turns on one question:

> **Can the driver be grown first, on its own, and handed over?**

If yes, it is **conditioning**: two runs, and the coupling's `source` is a file. If no, it is **joining**: one run, and the `source` is the name of a level growing beside it. Underneath, both are the same single mechanism — a modifier, `mod.DrivenBy` — and only the `source` differs.

```
from zombi2.rates import modifiers as mod

loss  = 0.25 * mod.DrivenBy("trait_events.tsv", {"cave": 4.0, "surface": 1.0})   # conditioned
birth = 1.0  * mod.DrivenBy("trait",            {"small": 1.0, "large": 2.0})    # joint
```

9.1 Conditioning

`mod.DrivenBy` takes two things: a `source` and a `mapping`. The `source` we have just split into file versus live level. The `mapping` is the other half, and it answers a separate question: once you know the driver's value on a lineage, what factor does the rate get multiplied by? It comes in three shapes:

- **Table** — a discrete driver becomes a dict, one factor per state: `{"cave": 4.0, "surface": 1.0}`. Any state you leave out keeps its rate unchanged, and states are matched by their written form, so an integer-labelled trait still finds its entry.
- **Curve** — a numeric driver becomes a function: `lambda n: math.exp(0.05 * n)`, a rate that rises smoothly with a lineage's gene count. Pass `bound=` to cap the factor when the driver has no ceiling of its own.
- **Scalar** — a single log-link coefficient, `exp(strength · value)`, the natural response when the driver is already a 0/1 indicator or one continuous covariate. `Scalar(0.0)` is the null: factor 1 everywhere.

The `source` says where the driver lives; the `mapping` says how its value is read. Whatever the shape, the factor it returns is dimensionless and non-negative, because it is going to multiply a rate.

Conditioning goes in exactly one direction today: **a trait drives gene gain or loss** (Traits → Genomes). The cave example is the canonical one — lineages in the dark lose genes faster than lineages in the light — and it takes two runs:

```
from zombi2 import species, traits, genomes

tree = species.simulate_species_tree(birth=1.0, death=0.3, n_extant=20, seed=1)
# 1. grow the driver: a habitat trait down the species tree
habitat = traits.simulate_discrete(tree, states=["cave", "surface"], switch=0.1, seed=1)

# 2. grow the genomes, with loss reading the habitat on each lineage
genomes.simulate_genomes_family(tree,
    loss = 0.25 * mod.DrivenBy(habitat, {"cave": 4.0, "surface": 1.0}),
    duplication=0.2, origination=0.5, seed=2)
```

The `source` here is the grown `TraitsResult` itself. That is the in-memory shortcut for the file: it is still conditioning, still two runs in order, but with no `write` and re-read in between. Hand it a filename instead and nothing else changes:

```
habitat.write("out/", outputs=("events",))          # writes out/trait_events.tsv (a bare
                                                    # write() puts it where you point it)
genomes.simulate_genomes_family(tree,
    loss = 0.25 * mod.DrivenBy("out/trait_events.tsv", {"cave": 4.0, "surface": 1.0}),
    duplication=0.2, origination=0.5, seed=2)
```

That is the whole of conditioning today, and it fits in four rows:

The driver	What it drives	Written like this	Mapping
a discrete trait	`loss`, `duplication`, `origination`, `transfer` — the gene-family rates, at the **family** resolution	`loss = 0.25 * mod.DrivenBy(source, {…})`	Table
a discrete trait	**every rate** of a **nucleotide** run — those four, plus `inversion`, `transposition`, `translocation` and the chromosome tier	`inversion = 1.5 * mod.DrivenBy(source, {…})`	Table
a discrete trait	**every extent** of a **nucleotide** run — how much DNA an event takes, rather than how often one happens	`loss_extent = 150 * mod.DrivenBy(source, {…})`	Table
a discrete trait	`transfer_to` — which lineage a transfer lands on (**family** resolution only)	`transfer_to = mod.DrivenBy(source, {…})`	Table, or Between (reads the donor too)

The middle two rows are the pair worth holding apart. Driving a **rate** makes a lineage delete more often; driving an **extent** makes each deletion bigger. Both are ways of saying "this lineage sheds more DNA", they are different processes, and set together they multiply.

`source` in both rows is the grown `TraitsResult`, or the path to the `trait_events.tsv` it wrote — the trait event log, which a driven run replays against the shared tree.

The driver file is the trait event log from Chapter 8: a `root` row for the state at t=0, then every switch with its time. Replayed against the shared species tree, that rebuilds each branch's constant stretches, which is what lets the genome engine step its Gillespie at every switch — so a lineage that changes habitat halfway down a branch loses genes at one rate before the switch and another after it. The coupling is exact, not a per-branch average.

9.1.1 Two ways a trait can drive transfer

A transfer joins two lineages, a donor and a recipient. A trait can drive either end, and the two are different models. The second row of the table is the recipient end.

Driving the **rate** drives the donor. `transfer = 0.1 * mod.DrivenBy(competence, {"competent": 3.0, "normal": 1.0})` makes a competent lineage donate three times as often as a normal one. That changes how much horizontal transfer happens in the run.

Driving `transfer_to` drives the recipient. `transfer_to = mod.DrivenBy(competence, {"competent": 3.0, "normal": 1.0})` makes a competent lineage three times likelier than a normal one to be the lineage a transfer lands on. That changes no rate at all. The same transfers happen; they go somewhere else.

The two expressions look alike, but their numbers mean different things. In a rate, the number is a multiplier: it multiplies the rate of the lineage it is read on. In `transfer_to` it is a **weight**:

the engine reads it on every lineage alive at that instant, and the recipient is drawn in proportion. Five candidates at weight 1 and five at weight 2 send two thirds of the transfers to the weight-2 group, because ten of the fifteen units of weight are theirs. Weights are normalised, so doubling all of them changes nothing.

That is why `transfer_to` takes the modifier on its own, with no number in front of it. A rate has a base, `0.1` per copy per unit time; a weight does not. Writing `transfer_to = 1.0 * mod.DrivenBy(...)` is an error.

A weight of 0 means the lineage cannot receive, which is often the point: only a competent lineage takes DNA up. That has one consequence worth stating plainly. If at some instant every candidate weighs 0, the transfer has nowhere to land, so it does not happen. While no eligible recipient is alive, the run's transfer rate is 0.

The two couplings are independent, and a run may use either or both:

```
competence = traits.simulate_discrete(tree, states=["competent", "normal"],
                                      switch=0.3, seed=1)

genomes.simulate_genomes_family(tree,
    transfer    = 0.1 * mod.DrivenBy(competence, {"competent": 3.0, "normal": 1.0}),
    transfer_to =       mod.DrivenBy(competence, {"competent": 3.0, "normal": 1.0}),
    initial_families=10, seed=2)
```

9.1.1.1 The recipient weight can read the donor too

The `transfer_to` above reads the driver on the *recipient* only. So it can say "competent lineages take DNA up more often", but not "genes move between two states and not within them" — the weight on a candidate does not know who is donating. A **`Between`** kernel closes that gap. In place of one factor per recipient state, it gives a weight per ordered **(donor state, recipient state)** pair, and the engine reads the driver on the donor as well as the candidate:

```
from zombi2.rates.mapping import Between

habitat = traits.simulate_discrete(tree, states=["marine", "soil"], switch=0.3, seed=1)

genomes.simulate_genomes_family(tree,
    transfer    = 0.5,
    transfer_to = mod.DrivenBy(habitat, Between({("marine", "marine"): 1.0,
                                                 ("soil",   "soil"):   1.0}, default=0.0)),
    initial_families=10, seed=2)
```

Every transfer now stays within one habitat: a marine donor can only reach a marine recipient, a soil donor only a soil one, because every cross-habitat pair weighs 0. The numbers are weights, read exactly as before — normalised over the candidates, so they redistribute transfers without changing how many happen. `default` (1.0) is the weight of any pair you leave out, so `Between({("marine", "soil"): 3.0})` *enriches* one direction against a baseline rather than forbidding the rest; `default=0.0` is the "only the flows I name" idiom. A `Between` is a recipient weight, never a rate: driving a rate with one is refused, because a rate has no donor to condition on.

This is the trait-driven twin of Chapter 4's `Clades`. There the groups are clades — a fact about

the tree — so no coupling is involved; here they are an evolved trait, so it is. Same kernel, same steering; only where the groups come from differs.

Combining a driven `transfer_to` with the `"distance"` rule of Chapter 4 is not supported: `transfer_to` takes one rule.

Everything outside those two rows raises an error. The driver has to be a discrete trait: only a discrete trait carries the character map that cuts a branch into constant segments, so a continuous trait is refused as a driver. That is the discipline everywhere in ZOMBI2 — a modifier a level cannot honour raises an error rather than being silently dropped.

9.1.2 What can be conditioned, and what cannot yet

Conditioning is wired at two of the three genome resolutions. At the **family** resolution it covers the four gene-family rates in the table above, plus `transfer_to`. At the **nucleotide** resolution it covers *every* rate the engine has — the rearrangements included — so a trait can speed up inversion, transposition and translocation, and can drive how much DNA a lineage sheds. That last one is genome reduction as it is usually meant: a lifestyle trait raising `loss`, on a genome measured in base pairs rather than in family tokens.

What is not wired yet:

- **The ordered resolution does not take a `DrivenBy` on its rates.** It wires `OnTime` (a skyline that varies in *time*) and `ByFamily` (per-family heterogeneity, applied to the segment an event covers), but not a driver. If you want a trait to drive rearrangement, use the nucleotide resolution, where an event is an arc of DNA rather than a run of gene tokens.
- **`transfer_to` — where a transfer lands — is family-resolution only.** A nucleotide transfer's *rate* can be driven; its recipient rule cannot.
- **Sequence evolution and trait runs** take no `DrivenBy` on their own rates.

These are limits of the implementation, not of the model — the coupling grammar (`SPEC §5`) is the same everywhere, so wiring `DrivenBy` into the ordered engine is a pure addition when it comes. Until then, a driven rate an engine cannot honour raises rather than being silently dropped.

Notice too that the coupling **folds into the target level's own command**. There is no separate coupling step and no coupling object to build; you grow the driver, then make an ordinary genome run whose `loss` happens to be `DrivenBy` instead of a bare number. That holds on the command line as well, where the rate keeps its written form:

```
# 1. a species tree
zombi2 species out/ --birth 1 --death 0.3 --n-extant 20 --seed 1

# 2. the driver: a habitat trait, writing its event log
zombi2 traits out/ --kind discrete \
    --states cave,surface --switch 0.1 --seed 1 --write values tree events

# 3. the target: genomes whose loss reads that trait
zombi2 genomes out/ \
    --loss "0.25 * DrivenBy('out/traits/trait_events.tsv', {'cave': 4.0, 'surface': 1.0})" \
    --duplication 0.2 --origination 0.5 --seed 2
```

Both halves of transfer take that same text: the rate with a base number in front of it, the recipient weight without one.

```
# the driver: a competence trait, into its own directory
zombi2 traits comp/ --kind discrete --from out/ \
    --states competent,normal --switch 0.3 --seed 1 --write events

zombi2 genomes comp_genomes/ --from out/ --initial-families 10 \
    --transfer    "0.1 * DrivenBy('comp/traits/trait_events.tsv', {'competent': 3.0, 'normal': 1.0})" \
    --transfer-to "DrivenBy('comp/traits/trait_events.tsv', {'competent': 3.0, 'normal': 1.0})" \
    --seed 2
```

9.2 Joining

When the driver cannot be grown first, there is nothing to write to a file, because the file would have to be written onto a tree that does not exist yet. Instead both levels are grown in one Gillespie that races every kind of event against every other: a speciation event reads the current trait state to set its rate, a trait-change event evolves the trait on the tree as it has grown so far, and a speciation hands the parent's state down to both daughters. Out come both levels at once. Because these drivers only change *at* events, the rate is constant between them and the race is exact — no thinning, no approximation.

Version 1 ships two joint pairs, and in both a level reaches back into the species tree, so the tree itself is an output rather than an input:

A trait drives speciation — `P(Species, Traits)`. A body-size state makes large lineages speciate twice as fast as small ones. The trait enters as a **process spec**, `traits.discrete(...)`, rather than as a finished run: it is a description of a process to be grown with the tree, not a result:

```
from zombi2 import joint, traits, genomes
from zombi2.rates import modifiers as mod

joint.simulate_joint(
    birth = 1.0 * mod.DrivenBy("trait", {"small": 1.0, "large": 2.0}),
    death = 0.2,
    trait = traits.discrete(states=["small", "large"], switch=0.1),
    n_extant = 100, seed = 1)
```

Driving `death` with the same modifier makes extinction state-dependent too, and a birth *and* death that both read the trait is the model the literature calls BiSSE — with more than two states, MuSSE.

Gene content drives speciation — `P(Species, Genomes)`. The presence of a key gene, a toxin or a transporter, lifts a lineage's speciation rate, and the genome and the tree grow together. The genome enters as a process spec in the same way, and the family whose presence does the driving has to be declared in the initial genome:

```
joint.simulate_joint(
    birth  = 1.0 * mod.DrivenBy("genomes:toxin", {"present": 1.8, "absent": 1.0}),
    death  = 0.2,
    genome = genomes.family(duplication=0.2, loss=0.25, origination=0.5,
```

```
                              family_names=["toxin"]),
    n_extant = 100, seed = 1)
```

The live `source` names either the family, `"genomes:toxin"`, or the lineage's total gene count, `"genomes:count"` — and a count is a number, so that is where a **Curve** earns its place: `mod.DrivenBy("genomes:count", lambda n: math.exp(0.02 * n))` makes gene-rich lineages speciate faster along a smooth response rather than a lookup table.

The three live sources in full:

The driver	The rates it can drive	`source`	Mapping
a discrete trait	`birth`, `death`	`"trait"`	Table
a named family's presence	`birth`, `death`	`"genomes:<family>"`	Table on `present`/`absent`
a lineage's gene count	`birth`, `death`	`"genomes:count"`	Curve, or Scalar

Give one driver per run, `trait=` or `genome=`, not both, and drive `birth`, `death`, or both with it.

Stop the run at a size with `n_extant=` or at an age with `total_time=`, exactly as in Chapter 3, and give one or the other. What comes back is a `JointResult` carrying **both** grown levels: `.species` always, and then either `.trait` or `.genome`, the same result objects the standalone commands return. They share one `complete_tree`, because there was only ever one tree — the one they grew between them.

9.2.1 Usage from the CLI

Conditioning folds into the target level's own command, as above. Joining cannot: there is no level to run first, so it has its own command, `zombi2 joint`. The driver is named in the rate exactly as in Python — `DrivenBy('trait', …)` rather than a path, because it names a level being grown rather than a file already written — and the flags that build the driver are the ones you would pass to `zombi2 traits` or `zombi2 genomes`.

```
# BiSSE: 'large' lineages speciate three times as fast as 'small' ones
zombi2 joint out/ --birth "1.0 * DrivenBy('trait', {'small': 1.0, 'large': 3.0})" --death 0.2 \
    --states small,large --switch 0.3 --n-extant 100 --seed 1

# state-dependent extinction too, by driving --death as well
zombi2 joint out/ --birth "1.0 * DrivenBy('trait', {'small': 1.0, 'large': 3.0})" \
    --death "0.2 * DrivenBy('trait', {'small': 2.0, 'large': 1.0})" \
    --states small,large --switch 0.3 --n-extant 100 --seed 1

# gene content drives it: carrying the 'toxin' family triples the speciation rate
zombi2 joint out/ --birth "1.0 * DrivenBy('genomes:toxin', {'present': 3.0, 'absent': 1.0})" \
    --origination 0.2 --loss 0.1 --families toxin --n-extant 60 --seed 1
```

One driver per run. `--states` builds the trait driver; the gene-content flags build the genome one; giving flags from both is an error rather than a silent choice between them.

9.2.2 Not everything that looks like a connection is one

One distinction keeps this chapter from swallowing material that belongs elsewhere. A trait that jumps at a speciation event, or a genome that changes only at splits, looks like a coupling to the species level, but it is not one. It is the level reading the tree it *already lives on*, which every level does for free. A cladogenetic trait shift and a punctuational burst of gene change are options of the trait's and the genome's own models, and they stay in Chapters 8 and 6 respectively. A coupling, in this chapter's sense, always reads a *different* level. When in doubt, ask whether the rate reads a value that some *other* level produced; if it only reads the tree, it is not here.

9.3 Literature

The state-dependent models arrive under a wall of acronyms, and a reader who wants "a BiSSE model" should be able to find the door. The names live here, in one table, and organise nothing else in the chapter.

What it does	ZOMBI2	From the literature
a binary trait drives speciation (and extinction)	`simulate_joint: birth, death = … * mod.DrivenBy("trait", {…})`	BiSSE
a multi-state trait drives speciation	the same, with more states in the Table	MuSSE

9.4 Outputs

A conditioned run writes what any ordinary level run writes — its genome or trait output — plus the trait **event log** that fed it (`trait_events.tsv`), so the pairing that produced the pattern is kept on disk alongside the result. A joint run writes **both** levels from one call: the grown species tree (`species_complete.nwk`, `species_extant.nwk`, `species_events.tsv`) together with the trait it grew (`trait_values.tsv`, `trait_events.tsv`, `trait_tree.nwk`) or the genomes it grew (`genome_events.tsv`, `profiles.tsv`), each in the format it would have had from its own command. Because a joint run grows the tree, the tree it writes is a *complete* tree in the sense of Chapter 3, with the extinct lineages that shaped the trait or gene distribution still in place — which matters here more than anywhere, since those are exactly the lineages whose fate the coupling decided. The full list of files lives in Appendix B.

Appendix A

Rates in detail, and the Gillespie algorithm

Chapter 2 introduced the shape every rate takes, `effective rate = scope(base) × modifiers`. This appendix is the full reference: how a rate's units work, the default scope at each level, the catalogue of modifiers and which levels accept them, and the Gillespie algorithm that turns rates into events.

A.1 How a rate is counted: the scope

A rate always has units of time^{-1}, on the scale imposed by the species tree. In a phylogenetic context, though, a single global rate rarely makes sense for most events. A substitution happens at a **site**, so a mutation rate is counted per site (mutations × time^{-1} × per site): each site is an independent chance to mutate. A speciation happens to a **lineage**, so the speciation rate is counted per lineage (speciations × time^{-1} × per lineage): each branch alive is an independent chance for the tree to split. And a gene is lost one gene copy at a time, so gene loss is counted per copy (loss × time^{-1} × per gene-copy). The unit a rate is counted in — per lineage, per copy, per site — is what we call its **scope**.

By default, this is the scope ZOMBI2 uses at each level:

Level	Counted per	The rates it applies to
Species	lineage	`birth`, `death`
Genomes	copy	`duplication`, `transfer`, `loss`
Genomes	lineage	`origination`
Genomes, ordered	chromosome	`inversion`, `transposition`, `fission`, `fusion`, `chromosome_loss`
Sequences	site	`substitution` (times a clock)
Traits	lineage	`rate` (continuous), `switch` (discrete)

A bare number takes the default. To count a rate some other way, wrap it in the scope you want:

```
from zombi2 import species
from zombi2.rates import scope

# a death rate applied to the whole tree at once, not once per lineage
species.simulate_species_tree(birth=1.0, death=scope.Global(0.3), total_time=8.0, seed=2)
```

The wrappers are `Global`, `PerLineage`, `PerCopy`, `PerSite` and `PerChromosome`. `Global(x)` is the one that most changes behaviour: it detaches the rate from the amount of material present, so a `Global` death rate does not grow as the tree does.

A.2 Bending a rate: modifiers

A **modifier** alters a rate in context. You might give a gene family a constant loss rate across the tree except in one clade known to shed genes — a symbiotic bacterium — by multiplying the rate there. Or let families evolve at different speeds: an antimicrobial-resistance family prone to transfer, a ribosomal-protein family the opposite.

The modifiers are:

Modifier	What it does to the rate
`OnTime`	Follows a **time schedule**: one factor up to a breakpoint, another after it.
`OnTotalDiversity`	**Slows as the tree fills up**: the factor falls from 1 toward 0 as the number of lineages approaches a carrying capacity, and stays there.
`FromParent`	Is **inherited from the parent lineage and nudged at each split**, so the rate drifts gradually down the tree and close relatives keep similar rates.
`ByLineage`	Is an **independent draw for each lineage**, with no memory of its parent, so nearby branches are no more alike than distant ones.
`DrivenBy`	**Reads another level**: the factor is looked up from a driver's state, which is how one level conditions another (Chapter 9).

The first two are **deterministic**: `OnTime` and `OnTotalDiversity` are fixed functions of the state of the world, so every lineage that meets the same time, or the same diversity, gets the same factor. The next two are **random and vary from lineage to lineage**, and they differ in *memory*: `FromParent` is passed down and drifts, so the rate is autocorrelated along the tree — a slowly wandering clock, or a clade that inherits a fast tempo — whereas `ByLineage` is drawn afresh on every branch, so the variation is scattered, an uncorrelated ("relaxed") clock. The random modifiers are **mean-corrected**, meaning their factors average to 1, so switching on heterogeneity spreads a rate around without secretly speeding the whole tree up. `DrivenBy` is neither: its factor is whatever the driver's state says it is.

Modifiers **stack by multiplication**, so they combine: `1.0 * mod.OnTime({0: 1, 5: 0.3}) * mod.FromParent(spread=0.3)` is a rate that both follows a schedule and drifts between lineages.

A.2.1 Which level accepts which

A modifier only makes sense where the level can act on it, and a level **rejects** a modifier it does not wire rather than silently ignoring it. This is what each accepts today:

Level	The modifiers it accepts
Species	`OnTime` · `OnTotalDiversity` · `FromParent`
Genomes	`OnTime` · `DrivenBy`
Sequences	`ByLineage`
Traits	`OnTime` · `OnTotalDiversity` · `FromParent`

The gaps are the rebuild, not the design: a level gains a modifier when its engine learns to read it. Pass one that a level does not wire and you get an error naming the modifiers that level accepts, so the table above is always recoverable from the tool itself.

A.3 The Gillespie algorithm

Almost every simulation here comes from one small engine, run over and over: the same loop grows a birth–death tree, duplicates and loses genes, and switches a discrete trait between states, given a different list of events each time.

That engine is the **Gillespie algorithm** (Gillespie 1976, 1977), an exact, event-by-event recipe for a continuous-time process defined by rates. This section builds it from scratch — what a rate is, why waiting times are exponential, how competing events race, and how those assemble into the loop — assuming no prior exposure to continuous-time Markov chains. **Rates in, a timed history out.**

A.3.1 From a rate to a waiting time

A rate says *how fast* something tends to happen: the expected number of events per unit of time. If a gene copy is lost at rate $\mu = 0.25$, then, left alone, it is lost on average once every $1/0.25 = 4$ time units.

More precisely, a rate λ is defined by what happens over a very short slice of time Δt. The chance that one event fires during that slice is proportional to its length,

$$P(\text{an event in the next } \Delta t) \approx \lambda\, \Delta t,$$

and the shorter the slice, the better the approximation. Notice this says nothing about a clock ticking down to the next event: in any instant the chance of firing is the same regardless of how long we have already been waiting. A rate has no memory. That single fact is what makes the whole algorithm work.

It also means a rate is a statement about probability, not a fixed schedule. A loss rate of one per unit of time does not deliver exactly one loss in every unit: over any given unit the number of events that fire is random. That count follows the **Poisson distribution**, with mean λT over an interval of length T (Figure A.1).

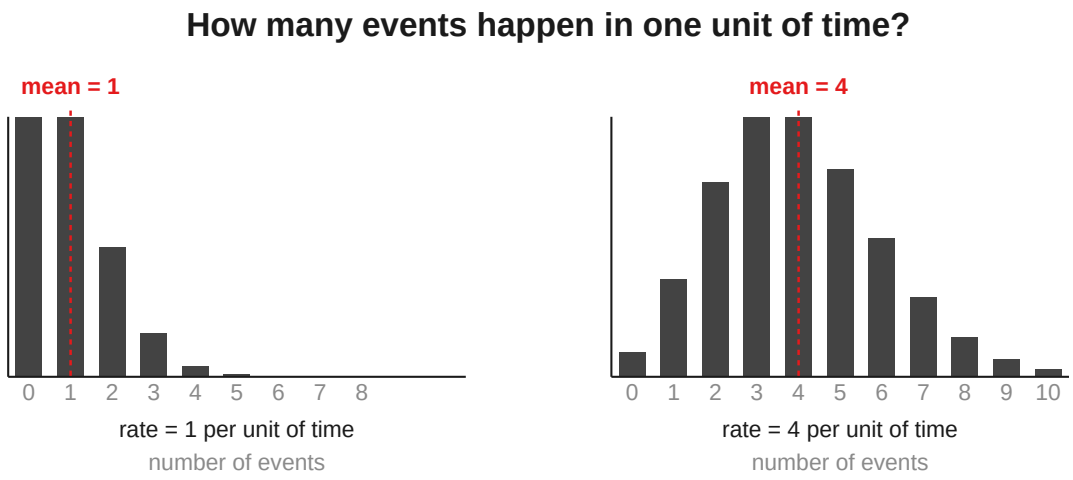

Figure A.1. The count of events in a fixed window is random, not fixed. With rate λ, the number of events in one unit of time is Poisson-distributed with mean λ (dashed line). At a low rate (left) most windows see zero or one event and a few see more; at a higher rate (right) the count spreads out around the mean. The rate fixes only the average.

Now fix a single event with a constant rate λ and ask: starting now, how long until it fires? Call that waiting time W. Because the chance of firing in each little slice is $\lambda\,\Delta t$ and slices are independent, the chance of surviving without an event up to time t decays to an exponential:

$$P(W > t) = e^{-\lambda t}.$$

W follows an **exponential distribution** with rate λ, whose mean is $1/\lambda$. Short waits are the most common. Because the exponential is memoryless, we never have to simulate the empty time *between* events tick by tick — we can draw the waiting time in one shot and jump to the next event. Drawing it is a single line: given a uniform random number u on $(0, 1)$,

$$\Delta t = -\frac{\ln u}{\lambda},$$

which is what `rng.exponential(1 / lambda)` returns, its argument being the mean. Every waiting time in ZOMBI2 is drawn this way.

A.3.2 When several things can happen: the race

A real simulation never has just one possible event. A genome with many gene families can duplicate any of them, transfer any of them, or lose any of them; a species tree with many lineages can speciate or go extinct on any branch. At a given moment there is a whole menu of possible events, event i with its own rate r_i.

Treat every possible event as an independent alarm clock, each set to go off after its own exponential waiting time. They all start together and race; the first alarm to ring is the event that happens. Two facts govern that race, and together they *are* the Gillespie algorithm.

When does the first event fire? The minimum of independent exponential waiting times is itself exponential, with a rate equal to the sum of the individual rates. So with a **total rate**

$$R = \sum_i r_i,$$

the time to the next event — whichever it turns out to be — is a single exponential draw with rate R. More possible events, or faster ones, means a larger R and therefore shorter waits. This is why we need only one waiting-time draw per step, however long the menu.

Which event fires? The winner is event i with probability equal to its share of the total rate,

$$P(\text{event } i \text{ fires}) = \frac{r_i}{R},$$

and which event wins is independent of when it happens. So the two are decided separately: draw the time from the total rate, then pick the event on a weighted roulette wheel, each slice sized to a rate.

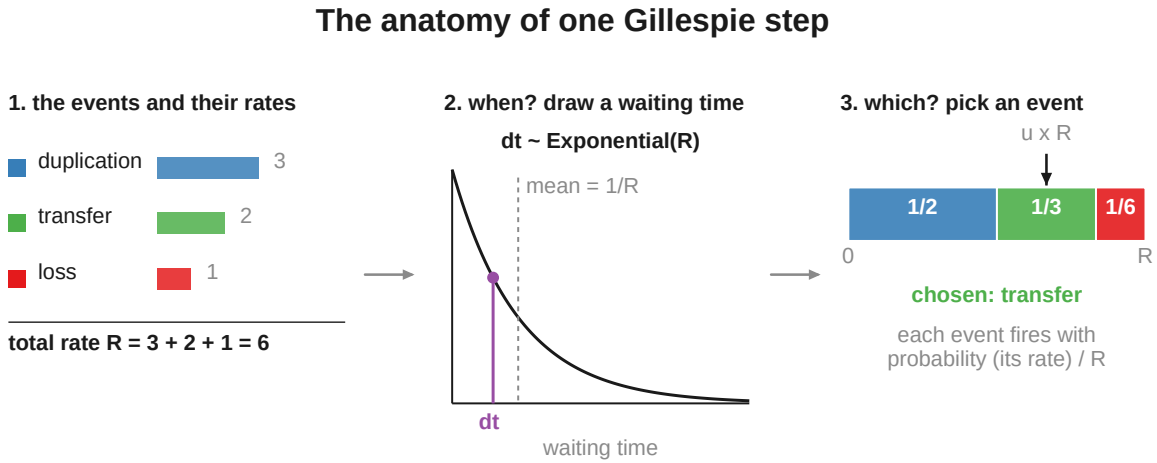

Figure A.2. The two draws that make up one Gillespie step. **(1)** Each possible event has a rate; here duplication, transfer and loss have rates 3, 2 and 1, summing to a total rate $R = 6$. **(2)** The waiting time to the *next* event is a single exponential draw with rate R; larger total rates give shorter waits, with mean $1/R$. **(3)** Which event fires is a second, independent draw: event i wins with probability r_i/R — the rates laid end to end as a roulette wheel, here landing on transfer. The step then advances the clock by Δt, applies the chosen event to the state, and repeats.

In code the roulette wheel is a running sum: lay the rates end to end, draw a point uniformly along their combined length R, and see which segment it lands in.

A.3.3 The loop

Assembling the pieces gives the loop below. Starting from an initial state at time $t = 0$, repeat: read off the current rates and their total R; draw a waiting time and advance the clock; stop if the clock has run past the target time, or the process has died out; otherwise pick one event in proportion to its rate, apply it to the state, record it, and go round again.

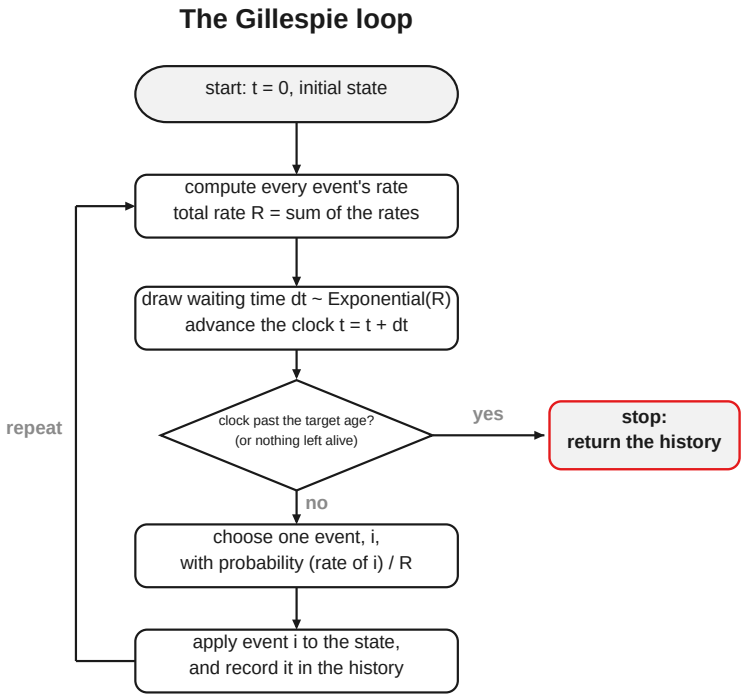

Figure A.3. The Gillespie loop. Each pass computes the current total rate, draws one exponential waiting time, and — unless the clock has passed the target age — fires a single event chosen in proportion to its rate, updates the state, and repeats. The output is a list of events with the exact times at which they occurred: a timed history.

As pseudocode, the whole engine is short:

```
t = 0.0
state = initial_state
history = []
while t < total_time:
    rates = event_rates(state)      # every possible event's rate, given the state
    R = sum(rates)                  # the total rate
    if R == 0:                      # nothing can happen; the process is frozen
        break
    t += rng.exponential(1 / R)     # WHEN: draw the waiting time, advance the clock
    if t >= total_time:
        break                       # the next event would fall past the horizon
    i = choose(rates, R, rng)       # WHAT: pick an event with probability r_i / R
    state = apply(state, i)         # update the state
    history.append((t, i))          # record the timed event
```

The result is not a snapshot but a **timed history**: the exact sequence of events and the exact real-valued times at which they happened. That is what a phylogenetic simulator needs — a species tree *is* the history of its speciation and extinction events, a gene family *is* the history of its duplications, transfers and losses. Because the times are drawn from continuous exponentials rather than stepped through a fixed grid, the histories are exact: no time-step to tune, and no discretisation error.

Note

The `rng` is a seeded `numpy` random generator. Because the waiting times and event choices are its only source of randomness, the same seed reproduces the same history exactly. The clean core runs this loop in plain Python.

A.3.4 When the rate changes with the clock

The loop above assumes the rates hold still *between* events, so that a single Exponential(R) draw lands exactly on the next event. That holds whenever the rates depend only on the current state, which changes only when an event fires. But some rates move with the clock itself, even while nothing is happening: an `OnTime` schedule steps at fixed breakpoints, a scheduled mass extinction arrives at a set time, and under `DrivenBy` the driving level changes state on its own timetable. Now R is a moving target, and a draw at today's R would be wrong.

ZOMBI2 keeps the draw exact by never letting it cross a change. Every rate can report the next time it changes on its own, and the engine takes the earliest such time — together with the next scheduled pulse and the end of the run — as a **horizon**:

1. Compute R from the rates as they stand, and the horizon.
2. Draw $\Delta t \sim$ Exponential(R).
3. If the event lands **before** the horizon, fire it: the rates really were constant over that stretch, so the draw is exact.
4. If it lands **after**, discard it, advance the clock to the horizon, and start again with the rates as they are there.

Discarding is sound for the same reason the loop works at all: the exponential is memoryless, so a partial wait carries no information into the next stretch. The rates are piecewise constant, each piece gets its own exact draw, and no integral or rejection step is ever needed.

A.3.5 It's all Gillespie

Each level supplies its own events and rates, and the same loop realises all of them:

Level	The events	Their rates
Species	speciation, extinction	`birth`, `death`, per lineage
Genomes	duplication, transfer, loss, origination	one rate per event
Traits (discrete)	switches between character states	the entries of the switch matrix

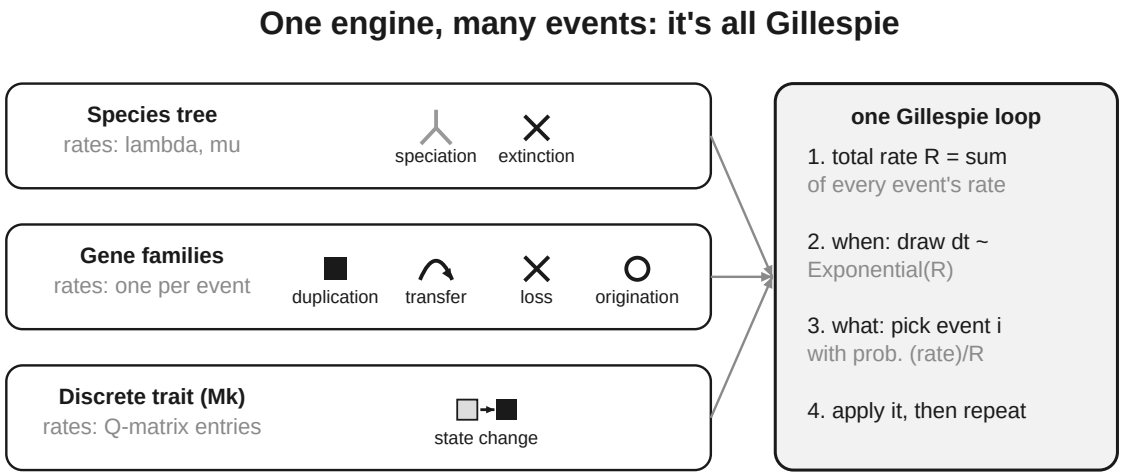

Figure A.4. One engine, many events. Each level supplies its own events and rates, but all are realised by the identical loop on the right: total rate, exponential waiting time, an event chosen in proportion to its rate, apply, repeat.

Swapping levels swaps the list of events and how their rates are computed; the timing machinery — total rate, exponential wait, proportional choice — never changes.

A.3.6 ...except when it isn't

ZOMBI2 steps outside the event-by-event loop in two places, for the same reason both times: when only the endpoints are needed, not the whole timed history, an exact shortcut beats simulating events you would throw away.

The first is **sequence substitution along a branch**. Once a gene tree and its branch lengths are settled, evolving a sequence down a branch does not require the individual substitution events, only the state at each end. The probability of ending in each state after a branch of length t is given exactly by the matrix exponential $P(t) = e^{Qt}$, so ZOMBI2 draws each site's descendant state straight from $P(t)$ in one step (Sequence evolution). Running Gillespie here would generate, and then discard, thousands of intermediate substitutions.

The second is a **continuous trait**. Brownian motion has no events to fire — it moves at every instant — so there is nothing for the loop to enumerate. A constant-rate run is drawn in closed form instead, as one multivariate normal over the whole tree, with variance $\sigma^2\times$ root-to-tip depth and covariance $\sigma^2\times$ shared path length (Trait evolution). When the rate varies along the tree, each branch takes the exact integral of σ^2 over it.

The rule of thumb is the same each time. Reach for Gillespie when you need the whole history — every branching, gain and loss at its exact time; reach for a shortcut when the endpoints are all you need. For the trees and genomes that are ZOMBI2's real subject the history *is* the result, so the loop is the norm and these two shortcuts are the exceptions.

Appendix B

Output files

Every `simulate_*` returns a result; `result.write("out/", outputs=[...])` writes the files, and omitting `outputs` writes the **default** set. Trees are Newick, tables and logs are TSV, sequences are FASTA. Tree branch lengths are **time** everywhere except the sequence phylograms, whose lengths are in **substitutions per site**. The **Default** column says whether a file is written with no arguments (**yes**), only when you name its token (**no**), or is available in Python but has no file yet (**Python**).

A species-tree node is written `n<id>` everywhere it appears, and the column holding one is always called `lineage` (or `donor` / `recipient` where a row names two), so a node reads the same in any file of a run. A gene copy is written `g<id>` the same way — the same token the gene-tree Newick leaves and the alignment FASTA headers use, so a copy joins across files without translation — and the column holding one is always `copy` (or `parent`, the source copy of a duplication or transfer). In `blocks.tsv` alone, `gene` means something else — the genic classification of a block, `0` for spacer or the family id for a declared gene.

B.1 Where the files go

A `zombi2` command groups what it writes, one directory per level, and gives every many-files-per-run output — one per gene family, or one per node — a directory of its own, so the level directory holds only its handful of tables and logs:

```
out/species/           species_complete.nwk · species_extant.nwk · species_events.tsv · species_fates.tsv
out/genomes/                genome_events.tsv · profiles.tsv · genomes.tsv · genomes.log
out/genomes/gene_trees/     gene_tree_fam<f>_complete.nwk · …_extant.nwk
out/genomes/homology/      homology_fam<f>.tsv                        (zombi2 tools format)
out/sequences/              clock_species_tree_complete.nwk · sequences.log
out/sequences/alignments/   fam<f>.fasta
out/sequences/phylograms/   phylogram_fam<f>_*.nwk
out/traits/               trait_values.tsv · trait_tree.nwk · trait_events.tsv · traits.log
```

Filenames keep their prefix inside their directory — `species/species_complete.nwk` — so a file still names itself once it has been moved or copied somewhere else.

`--flat` on any command writes its files straight into the output directory instead, for a tool that expects one directory. The same files are written either way; only the directories differ. A run's files being grouped is a CLI matter, not the library's: `result.write(dir)` writes into whatever directory you hand it.

Every command that reads a prior level takes the **run directory**, and finds the file itself, in either layout: `zombi2 genomes out/` and `zombi2 traits out/` pick up that run's complete species tree, and

`zombi2 sequences out/` picks up its handoff files. `--from` reads from somewhere else — a Newick file, or another run — which is also how you write a run separate from the one you read.

Every `zombi2` **command** also writes a run log (`species.log`, `genomes.log`, `sequences.log`, `traits.log`): the version, the timestamp, the command line, and every resolved parameter. Rates are recorded in their **written form** — `birth<TAB>1.0 * OnTime({0: 1, 3: 0.3})` — so a line pastes straight back into the flag or a `--params` file. It is a CLI artifact, not a `result.write()` output, so it has no row in the tables below.

B.2 Species trees — `simulate_species_tree`

Output	File	Format	Default	Contents
Complete tree	`species_complete.nwk`	Newick	yes	every lineage, including extinct and unsampled
Extant tree	`species_extant.nwk`	Newick	yes	only the sampled survivors
Event log	`species_events.tsv`	TSV	yes	every speciation/extinction — `time` · `kind` · `lineage` · `children`
Tip fates	`species_fates.tsv`	TSV	yes	each tip's resolved fate — `lineage` · `fate` (`extant` / `extinct` / `unsampled`)
Fossils	`species_fossils.tsv`	TSV	yes[1]	sampled fossil lineages — `lineage` · `time`

[1] written only if fossil sampling recovered any.

B.3 Genomes, family — `simulate_genomes_family`

Output	File	Format	Default	Contents
Event log	`genome_events.tsv`	TSV	yes	the source of truth — `time` · `kind` · `lineage` · `family` · `copy` · `parent` · `recipient` · `donor`. A transfer writes two rows sharing a `parent`, and **both name `donor`**, the branch the material left, so either row gives the whole edge without pairing them. On the arriving row `lineage` and `recipient` are the same branch, so `donor` is what makes a self-transfer (`donor` == `recipient`) visible at all
Profiles	`profiles.tsv`	TSV	yes	family × extant-species copy counts

Output	File	Format	Default	Contents
Genomes	`genomes.tsv`	TSV	yes	every node's gene content, **ancestors included** — `lineage` · `family` · `copy`. **One row per gene copy**, so a lineage holding six genes has six rows; two rows sharing a `family` are two copies of it. `copy` is the same identifier the event log uses, so a gene can be followed from the genome it sits in back to the event that made it. `profiles.tsv` is the same information counted, and only for the extant tips
Initial genome	`initial_genome.tsv`	TSV	yes	the genome the run **started** with, at the start of the root branch — `family` · `copy`. Its own file, with no `lineage` column, because it belongs to no node: every `lineage` elsewhere is a node, and a node sits at the *end* of its branch
Conditioning	`conditioned_on`	text	conditioned	written **only when a rate was conditioned**: the run's levels this run read via `DrivenBy` (one per line, e.g. `traits`). It records the dependency so re-running a driver level (say the trait) refuses to leave this run silently stale, or clears it under `--force`. A run with no driven rate writes no such file
Gene trees	`gene_tree_fam<f>_complete.nwk` · `…_extant.nwk`	Newick	yes	each family's true genealogy, in `genomes/gene_trees/`. A family with no surviving copy writes no `_extant` file
Family origination	`.gene_trees[f].origination`	float	Python	when the family was founded — where its gene tree's root branch begins

B.4 Genomes, ordered — `simulate_genomes_ordered`

Output	File	Format	Default	Contents
Event log	`genome_events.tsv`	TSV	yes	**the run's whole history, in one time-ordered table.** The gene genealogy as at the family resolution, plus **where** each event happened and the ancestry-neutral rearrangements as kinds of their own — `time` · `kind` · `lineage` · `family` · `copy` · `parent` · `recipient` · `donor` · `dest_lineage` · `chromosome` · `position` · `length` · `dest_chromosome` · `dest_position` · `flipped`[1]. `position` / `length` are coordinates in that branch's own genome just before the event, as `gene_order` numbers it, and are filled **once per event** — on its first row — because the arc belongs to the event, not to each copy it touched. Filtering on a non-empty `position` therefore gives one row per event that moved genes, which is what a replay walks. A transfer writes a row on each branch and each names the whole edge: `donor` is on both, and the departing row adds `dest_lineage` for where the material went
Profiles	`profiles.tsv`	TSV	yes	family × extant-species copy counts
Gene order	`gene_order.tsv`	TSV	yes	signed gene order of **every node**, ancestors included — `lineage` · `chromosome` · `position` · `strand` · `family` · `copy`
Initial genome	`initial_genome.tsv`	TSV	yes	the genome the run **started** with, at the start of the root branch — `chromosome` · `position` · `strand` · `family` · `copy`. Its own file, with no `lineage` column, because it belongs to no node: every `lineage` elsewhere is a node, and a node sits at the *end* of its branch
Chromosome events	`chromosome_events.tsv`	TSV	yes	chromosome-network edges — `time` · `kind` · `lineage` · `parents` · `children`
Gene trees	`gene_tree_fam<f>_complete.nwk` · `…_extant.nwk`	Newick	yes	as at the family resolution — position is orthogonal to genealogy

[1] a run is named by `start` (its first position, in the chromosome's frame just before the event) and `length` (how many genes it covered), counted rightwards from `start` and **wrapping past position 0 on a circular chromosome** — so `start + length` greater than the chromosome's gene count means the run crossed the origin. `dest_position` is an index into what was left after the run was

excised.

B.5 Genomes, nucleotide — `simulate_genomes_nucleotide`

From `zombi2 genomes --resolution nucleotide` or `result.write(dir, outputs=[...])`.

Output	File	Format	Default	Contents
Event log	`genome_events.tsv`	TSV	yes	**the run's whole history, in one time-ordered table**: the copy-lineage genealogy and the ancestry-neutral rearrangements — `time` · `kind` · `lineage` · `chromosome` · `copy` · `parent` · `recipient` · `source` · `start` · `end` · `position` · `length` · `dest_chromosome` · `dest_position` · `flipped`. One row per **ancestral interval** an event touched, so an event spanning several blocks writes several rows. `kind initial` marks the initial genome being laid down at time 0 (one per replicon) — what the run *starts* with, kept distinct from `origination`, the de-novo births the `--origination` rate makes, so counting one never counts the other. `source`/`start`/`end` are **ancestral** coordinates — which stretch of which source — while `position`/`length` are **physical** ones along the chromosome, as `blocks.tsv` numbers it: different frames, so different columns
Blocks	`blocks.tsv`	TSV	yes	every node's genome as its block mosaic, ancestors included — `lineage` · `chromosome` · `position` · `source` · `start` · `end` · `strand` · `copy` · `gene`. The rows of one chromosome tile it end to end from 0. The largest file this level writes: blocks are not kept maximal during a run, so it grows with their number × every node
Genes	`genes.tsv`	TSV	yes	the declared genes in initial coordinates — `family` · `name` · `source` · `start` · `end` · `strand` (the **coding** strand). Header-only when none were declared

Output	File	Format	Default	Contents
Initial sequence	`initial_sequence.fasta`	FASTA	yes[1]	the initial DNA the run was given (`--fasta`), one `>source<n>` record per replicon. Written only when a FASTA was supplied; it is what lets a separate `zombi2 sequences` run found its blocks from the real sequence
Initial genome	`initial_genome.tsv`	TSV	yes	the genome the run **started** with, at the start of the root branch — `chromosome` · `position` · `source` · `start` · `end` · `strand` · `copy` · `gene`. Its own file, with no `lineage` column, because it belongs to no node: every `lineage` elsewhere is a node, and a node sits at the *end* of its branch
Chromosome events	`chromosome_events.tsv`	TSV	yes	chromosome-network edges — same format as ordered
Gene trees	`gene_tree_fam<f>_complete.nwk` · `…_extant.nwk`	Newick	yes	one tree per declared gene (else per recovered root-block)

The nucleotide log needs no separate positions file: its events carry ancestral coordinates already.

The events index against the species tree canonicalised so its `n<id>` labels match the `lineage` column, so a genomes run needs that exact tree to be replayable. A run grown by `zombi2 species` already has it; a run whose tree came from `--from` gets a copy written to its own `species/species_complete.nwk`, rather than a second file under another name. Either way `zombi2 sequences` can replay the gene genealogy from the run directory alone. Like `names.tsv` (external input trees) and the `.log`, that copy is a CLI artifact, not a `result.write()` output.

B.6 Sequences — `simulate_sequences`

The `zombi2 sequences` command replays a prior `zombi2 genomes` run — its own run directory, or `--from` another — its species tree and its `genome_events.tsv`. Gene outputs are written **one file per gene family** (`<f>` = family number); a family with no surviving copy writes none. Every node is labelled `g<copy>`, so a phylogram's tips pair with its alignment and its internal nodes with the ancestral sequences.

Output	File	Format	Default	Contents
Alignments	`fam<f>.fasta`	FASTA	yes	one row per extant gene copy — nucleotides or amino acids, following the model. The command puts these in `sequences/alignments/`, which is what lets the name be this short
Phylograms	`phylogram_fam<f>_complete.nwk` · `…_extant.nwk`	Newick (subs/site)	yes	the gene tree each family's sequences were drawn along, in `sequences/phylograms/`

Output	File	Format	Default	Contents
Ancestral	`sequences_ancestral_fam<f>.fasta`	FASTA	no	the sequence at every node that is not an extant tip: internal nodes, and the tips where a copy was lost or its species died
Founding	`sequences_founding.fasta`	FASTA	no	one record `fam<f>` per family — the sequence it originated with, where its phylogram's root branch begins
Clock species tree	`clock_species_tree_complete.nwk` · `…_extant.nwk`	Newick (subs/site)	yes	the species tree with its branches in substitutions/site — the molecular clock made visible
Genomes	`genome_<lineage>.fasta`	FASTA	yes	one file per **node** of the complete tree — extant, extinct and ancestral alike — one record `<lineage>_chr<c>` per chromosome: the assembled genome, its blocks concatenated in physical order. **Nucleotide genome runs only**: a family or ordered run has gene families, not coordinates, so there is nothing to lay out. The biggest thing this level writes — a whole genome times every node
Initial genome	`genome_initial.fasta`	FASTA	yes	the genome the run **started** with, as sequence — the state the stem leads *from*, which is not any node's. Nucleotide runs only

On a **nucleotide** genome run every block evolves, spacer as well as gene, so a genome of b blocks writes b alignments and b phylograms — that is what makes the genomes assemblable. The number in those filenames is then a **root block index**, not a gene family id, and the files say so: `block6.fasta` and `phylogram_block6_complete.nwk` in place of `fam6.…`. The two numbering schemes are different, so go from a gene to its block with `genomes.block_of(family)` (Ch7). `zombi2 sequences` reads a nucleotide handoff too — it recognises one by its `blocks.tsv` — so all of this is reachable from the command line.

B.7 Joint — `simulate_joint` / `zombi2 joint`

A joint run grows two levels at once, so it writes both, each in the format its own command would give it: the species files, and then the driver's — the trait's or the genomes'. There is no output of its own beyond the run log.

Output	File	Format	Default	Contents
Species tree	`species/species_complete.nwk` · `…_extant.nwk` · `species_events.tsv`	Newick, TSV	yes	the grown tree — complete, so the extinct lineages the coupling decided the fate of are kept

Output	File	Format	Default	Contents
The trait it grew	`traits/trait_values.tsv` · `trait_events.tsv` · `trait_tree.nwk`	TSV, Newick	yes[1]	as the traits level writes them
The genomes it grew	`genomes/genome_events.tsv` · `profiles.tsv` · `genomes.tsv` · `gene_trees/`	TSV, Newick	yes[1]	as the genomes level writes them
Run log	`species/joint.log`	TSV	yes	the resolved parameters, as every command writes

[1] whichever driver the run used — one per run, never both.

B.8 Traits — `simulate_continuous` / `simulate_discrete`

Output	File	Format	Default	Contents
Values	`trait_values.tsv`	TSV	yes	value at each extant tip — `node` · `trait`
Events	`trait_events.tsv`	TSV	yes (discrete)	the trait's whole history — a `root` row giving the state at t=0, then every switch: `time` · `kind` · `lineage` · `from` · `to`, where `kind` is `root` · `on_branch` · `on_speciation`. Times are full precision (they drive a conditioned run's Gillespie). **This is also the conditioning file**: a genome/sequence run drives a rate with `mod.DrivenBy("trait_events.tsv", …)`, replaying it against the shared tree. A continuous trait carries only the `root` row and any `at_speciation` jumps (a diffusion can't be rebuilt from events)
Trait tree	`trait_tree.nwk`	Newick	no	tree with every node annotated `[&trait=…]` (opens in FigTree / iTOL)

B.9 Coupling — no new files

Coupling adds no formats. A **conditioned** run writes the target level's files plus the **driver file** it read (above), keeping the pairing on disk; a **joint** run writes **both** levels, each in its own format.

The `zombi2 tools` commands write their own files — the homology matrix and the reconciliation/scoring outputs — catalogued in Appendix C.

Appendix C

Tools

`zombi2 tools` runs read-back analyses. Where the level commands simulate, the tools re-express what already exists: they read files and derive a new view of them. Each tool is a sub-subcommand — `zombi2 tools <tool>`. `format` reads a whole genomes run and writes its tables beside the run. `tree` and `treedist` work on Newick trees instead: they read one or two `.nwk` files and write their result to stdout by default, or to a file with `-o`.

C.1 `format` — analysis-ready tables

`zombi2 tools format DIR` reads a genomes run and writes tables derived from its gene trees, one `--format` at a time, into a directory under `genomes/`. It works for every resolution: family and ordered runs rebuild their gene trees from the event log; a nucleotide run recovers them from the genome, and there one table is written per **declared gene** — the intergenic spacer is not a gene, so it gets none. `--from PATH` reads a run that lives elsewhere; `--flat` writes the tables straight into the output directory.

Output	File	Format	Default	Contents
Homology matrix	`homology_fam<f>.tsv`	TSV	`--format homology`	one **n**×**n** table per family (n the extant leaves), in `genomes/homology/`. Row and column headers are the leaves `n<species>|g<copy>`; each off-diagonal cell is the relation of that pair — `O` ortholog (their MRCA is a speciation), `P` paralog (a duplication), `X` xenolog (a transfer) — and the diagonal is `-`. Symmetric. A family with no surviving copy writes no table

The homology matrix is exact, not inferred. ZOMBI simulated each gene tree's embedding in the species tree, so the event at a leaf pair's most-recent common ancestor is **recorded** on the tree rather than reconstructed from it: a speciation there makes the pair orthologs, a duplication paralogs, a transfer xenologs. That is what makes these tables a ground-truth reference to score an orthology-inference method against.

C.2 `tree` — one transform on a Newick tree

`zombi2 tools tree TREE` applies a single transform to a Newick tree and writes the result — Newick to stdout, or to a file with `-o`. Exactly one action runs per call. `TREE` is a tree file, or `-` to read the

tree from stdin.

The actions split by whether they need each tip's fate. `--prune` does: it drops the dead and unsampled lineages to leave the extant tree, so it reads the fates a ZOMBI complete tree carries (an ultrametric tree, whose tips are all contemporaneous, counts as all-extant). A plain non-ultrametric tree with no fates is refused. The rest — the geometric transforms and `--red` — ignore fates and load any tree, so an inferred phylogram or a rounding-noisy dated tree goes through them unchanged.

Action	What it writes
`--prune`	the extant tree: the dead and unsampled lineages dropped, and the unifurcations they leave behind suppressed so the tree stays bifurcating
`--round`	the tree snapped to exactly ultrametric, by extending the terminal branches to a common depth. `--tol` is the tolerance as a fraction of tree height (default `1e-3`); a tip-depth spread wider than that raises, because it is real tip-date signal — extinct lineages or serial samples — not rounding
`--stem LEN` / `--stem-add LEN`	the branch above the crown set to `LEN`, or extended by `LEN`; nothing below the crown moves
`--rescale-height H` / `--rescale-factor F`	every branch length scaled — so the root-to-tip height becomes `H`, or by a raw multiplier `F`
`--red`	the RED-rescaled tree: node depths become their Relative Evolutionary Divergence (Parks et al. 2018), ultrametric on `[0, 1]` with the root at 0 and every tip at 1. `--red --values` writes a two-column `node<TAB>RED` table instead of a tree

```
# drop the extinct lineages, extant tree to stdout
zombi2 tools tree out/species/species_complete.nwk --prune

# snap a rounding-noisy dated tree to ultrametric, to a file
zombi2 tools tree dated.nwk --round -o dated_ultrametric.nwk

# the RED of every node, as a table
zombi2 tools tree out/species/species_extant.nwk --red --values
```

C.3 `treedist` — distance between two trees

`zombi2 tools treedist TREE_A TREE_B` reports how far apart two rooted trees are over their shared tips, printed as `<metric><TAB><value>` to stdout (or a file with `-o`). Pick the metric with `--metric`:

`--metric`	Distance
`rf` (default)	Robinson–Foulds — the number of clades present in one tree but not the other
`rf-normalized`	that count over the total number of non-trivial clades, so it lands in `[0, 1]`
`branch-score`	Kuhner–Felsenstein — the square root of the summed squared branch-length differences over every clade, terminal branches included; unlike RF it moves even when only the branch lengths differ
`all`	every metric above, one per line

The tips are matched by **label** — the tip name for an external tree, or `n<id>` for a ZOMBI tree — so a true tree and an inferred tree line up by taxon, whatever order their files list the tips in. The two trees must carry the same tip set: a differing leaf set is an error, not a partial score.

```
# Robinson–Foulds between a true tree and an inferred one over the same tips
zombi2 tools treedist true.nwk inferred.nwk --metric rf

# every metric at once
zombi2 tools treedist true.nwk inferred.nwk --metric all
```

C.4 The rest

The remaining scoring and reconciliation commands (reconciliation-accuracy, the undated simulator, the ALE likelihood) are quarantined during the clean-core rebuild; their files are documented here as each returns.

References

Gillespie, Daniel T. 1976. "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions." *Journal of Computational Physics* 22 (4): 403–34. https://doi.org/10.1016/0021-9991(76)90041-3.

Gillespie, Daniel T. 1977. "Exact stochastic simulation of coupled chemical reactions." *Journal of Physical Chemistry* 81 (25): 2340–61. https://doi.org/10.1021/j100540a008.