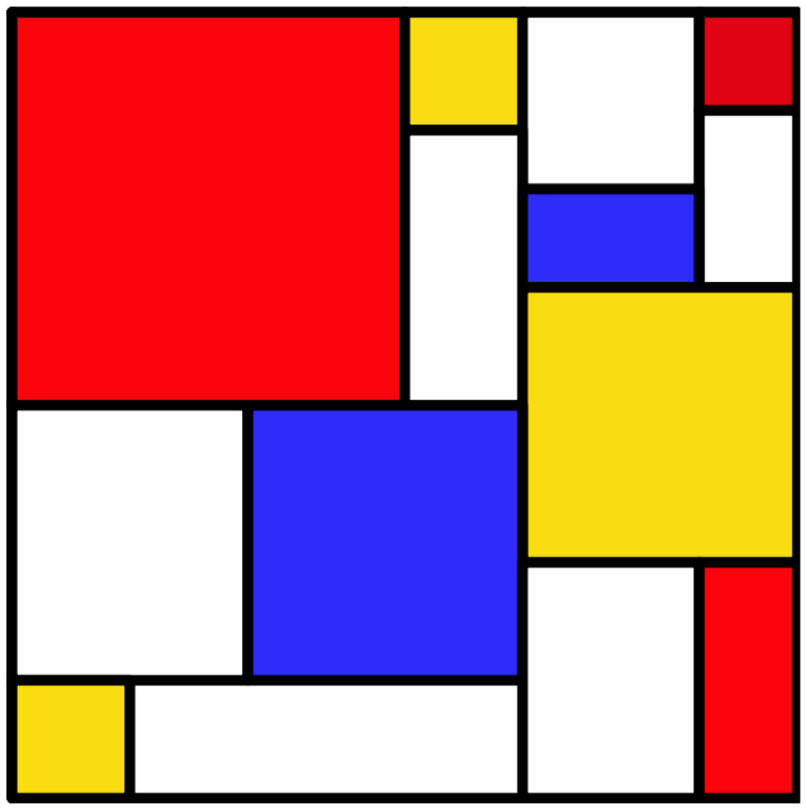
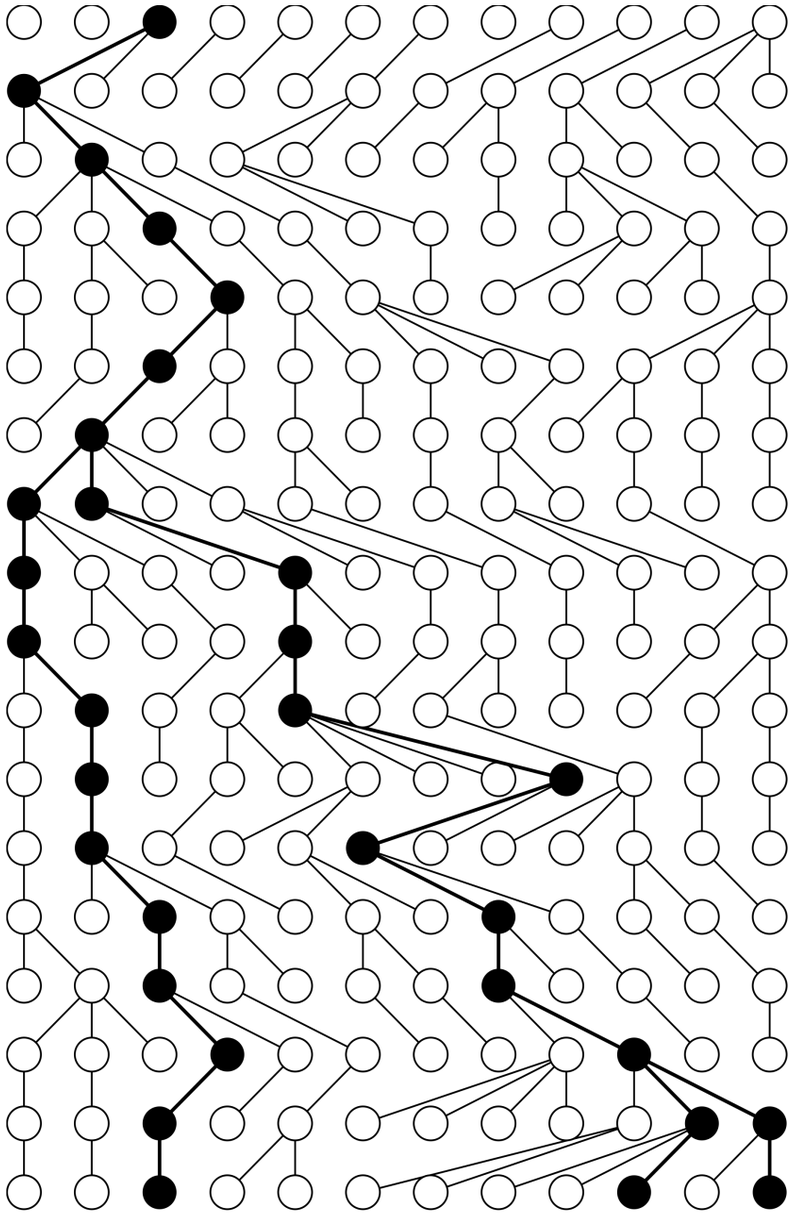
Forward-time simulation using SLiM



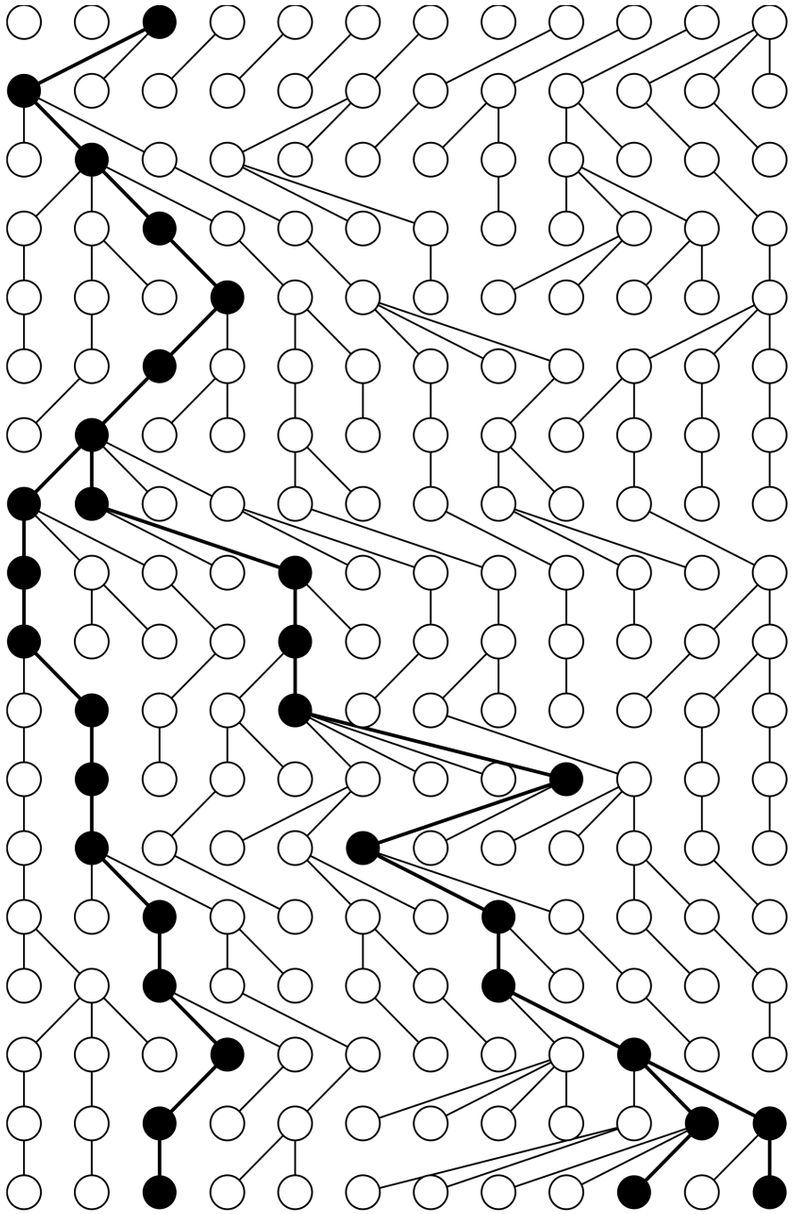
Backwards-in-time modelling :
The Coalescent



✓Makes modelling mathematically tractable

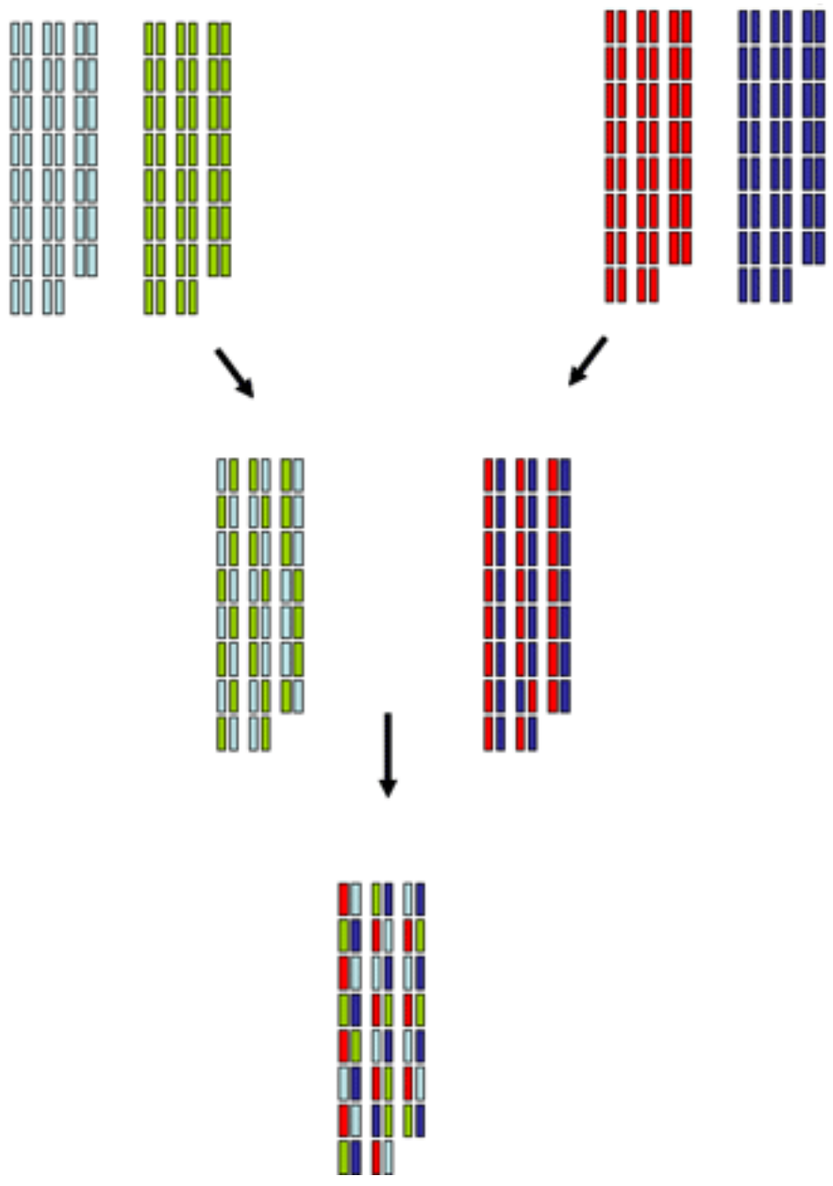
✓Huge variety of applications

Backwards-in-time modelling :
The Coalescent



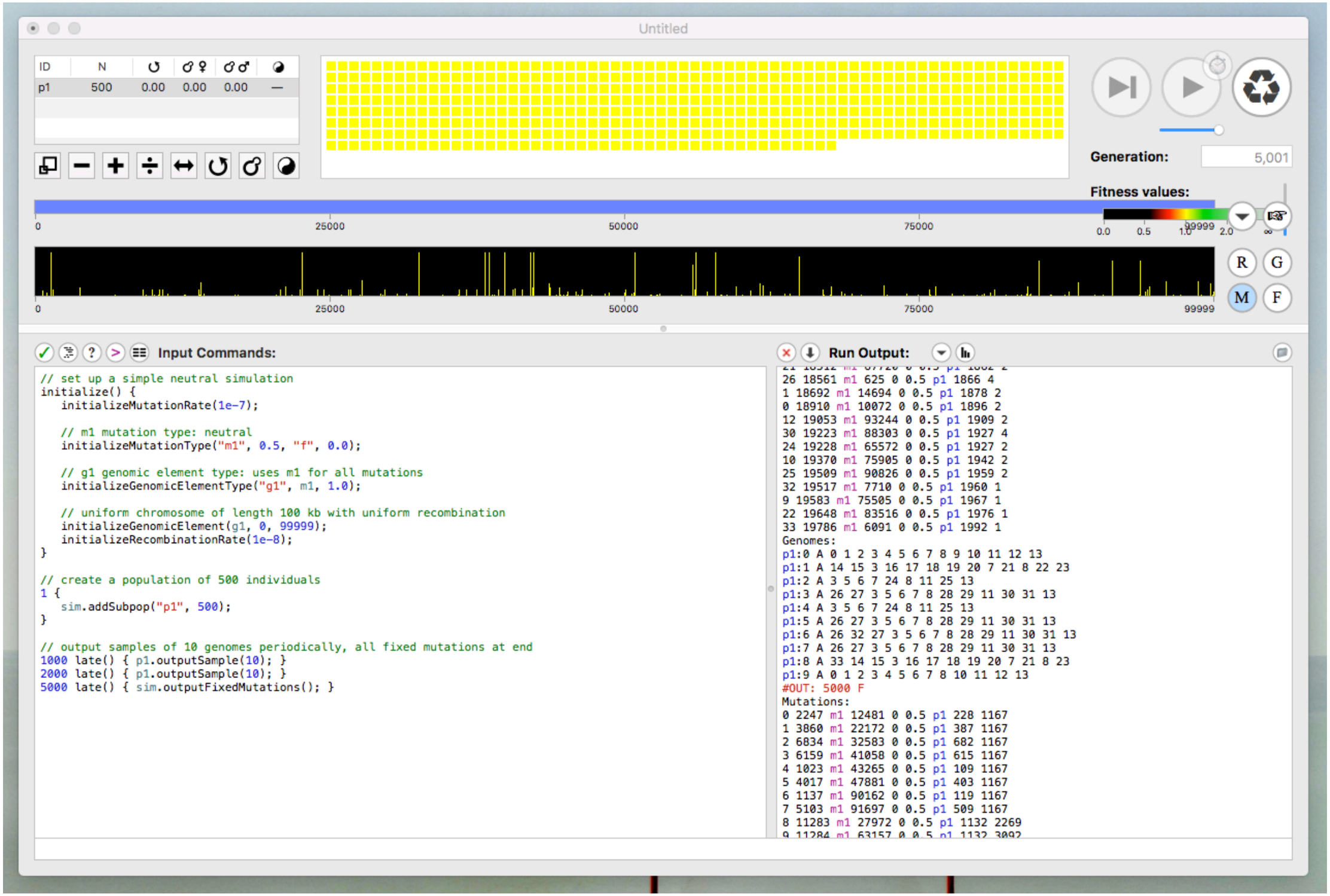
- Assumes $n << N_e$
- Is a model of a model of the actual population
- Potentially unnecessary

Forward-in-time modelling



- Highly intuitive
- One step closer to reality (a model of the empirical population)
- More tractable due to efficient data storage and improved computing power

SLiM


Untitled
ID N
p1 500 0.00 0.00 0.00 —
Generation: 5,001
Fitness values:
0 25000 50000 75000
0.0 0.5 1.0 99999 2.0 ∞
0 25000 50000 75000 99999
R G M F
Input Commands:
Run Output:
// set up a simple neutral simulation
initialize() {
initializeMutationRate(1e-7);
// m1 mutation type: neutral
initializeMutationType("m1", 0.5, "f", 0.0);
// g1 genomic element type: uses m1 for all mutations
initializeGenomicElementType("g1", m1, 1.0);
// uniform chromosome of length 100 kb with uniform recombination
initializeGenomicElement(g1, 0, 99999);
initializeRecombinationRate(1e-8);
}
// create a population of 500 individuals
1 {
sim.addSubpop("p1", 500);
}
// output samples of 10 genomes periodically, all fixed mutations at end
1000 late() { p1.outputSample(10); }
2000 late() { p1.outputSample(10); }
5000 late() { sim.outputFixedMutations(); }
26 18561 m1 625 0 0.5 p1 1866 4
1 18692 m1 14694 0 0.5 p1 1878 2
0 18910 m1 10072 0 0.5 p1 1896 2
12 19053 m1 93244 0 0.5 p1 1909 2
30 19223 m1 88303 0 0.5 p1 1927 4
24 19228 m1 65572 0 0.5 p1 1927 2
10 19370 m1 75905 0 0.5 p1 1942 2
25 19509 m1 90826 0 0.5 p1 1959 2
32 19517 m1 7710 0 0.5 p1 1960 1
9 19583 m1 75505 0 0.5 p1 1967 1
22 19648 m1 83516 0 0.5 p1 1976 1
33 19786 m1 6091 0 0.5 p1 1992 1
Genomes:
p1:0 A 0 1 2 3 4 5 6 7 8 9 10 11 12 13
p1:1 A 14 15 3 16 17 18 19 20 7 21 8 22 23
p1:2 A 3 5 6 7 24 8 11 25 13
p1:3 A 26 27 3 5 6 7 8 28 29 11 30 31 13
p1:4 A 3 5 6 7 24 8 11 25 13
p1:5 A 26 27 3 5 6 7 8 28 29 11 30 31 13
p1:6 A 26 32 27 3 5 6 7 8 28 29 11 30 31 13
p1:7 A 26 27 3 5 6 7 8 28 29 11 30 31 13
p1:8 A 33 14 15 3 16 17 18 19 20 7 21 8 23
p1:9 A 0 1 2 3 4 5 6 7 8 10 11 12 13
#OUT: 5000 F
Mutations:
0 2247 m1 12481 0 0.5 p1 228 1167
1 3860 m1 22172 0 0.5 p1 387 1167
2 6834 m1 32583 0 0.5 p1 682 1167
3 6159 m1 41058 0 0.5 p1 615 1167
4 1023 m1 43265 0 0.5 p1 109 1167
5 4017 m1 47881 0 0.5 p1 403 1167
6 1137 m1 90162 0 0.5 p1 119 1167
7 5103 m1 91697 0 0.5 p1 509 1167
8 11283 m1 27972 0 0.5 p1 1132 2269

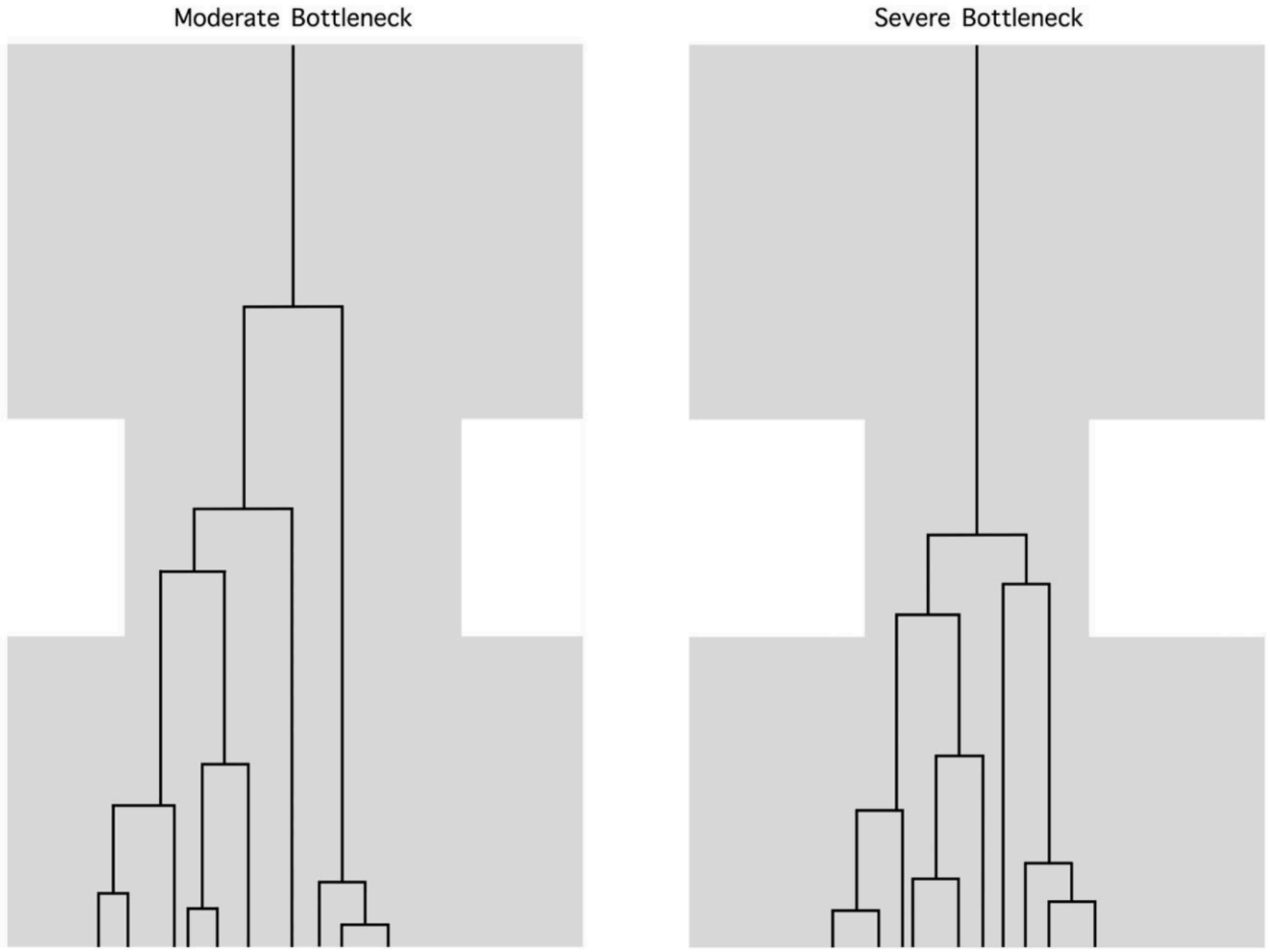
Why make simulations?

- Demography (or recombination rate, or mutation rate, or sampling effort…) can have confounding effects on many of the summary statistics that we use to infer the action of selection or gene flow

- We should not rely on intuition

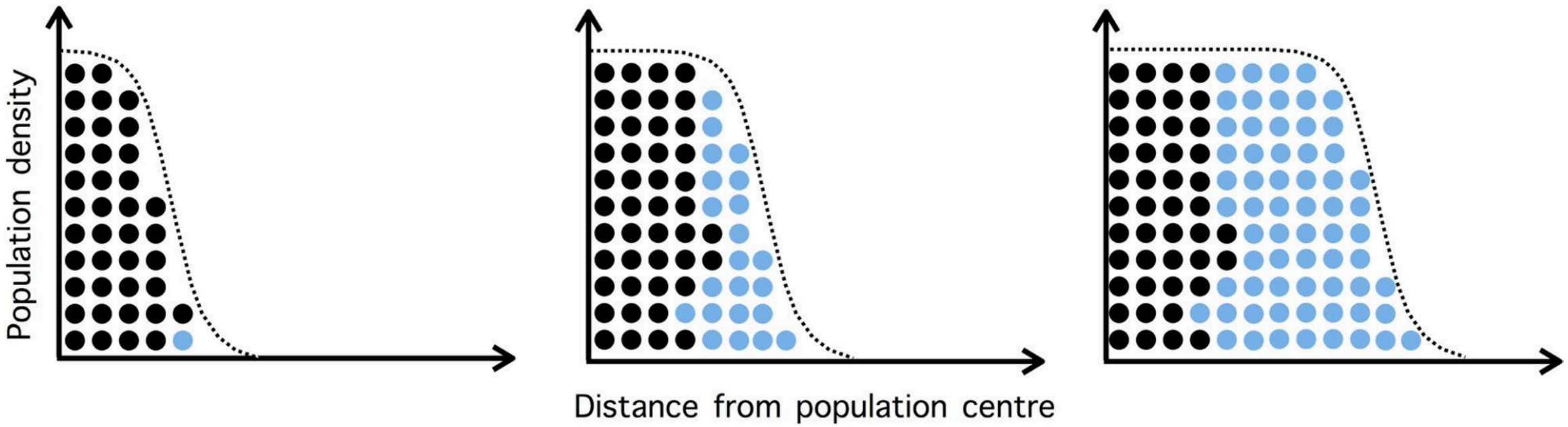
- We should do our best to convince ourselves (or reviewers, or readers) that our results are robust to assumptions about the underlying model

Temporal effects of demography on genetic diversity



Population bottleneck

Spatial effects of demography on genetic diversity



Allele surfing

Tajima's D: compare two estimates of the same parameter

Nucleotide diversity

(average number of pairwise differences between sequences)

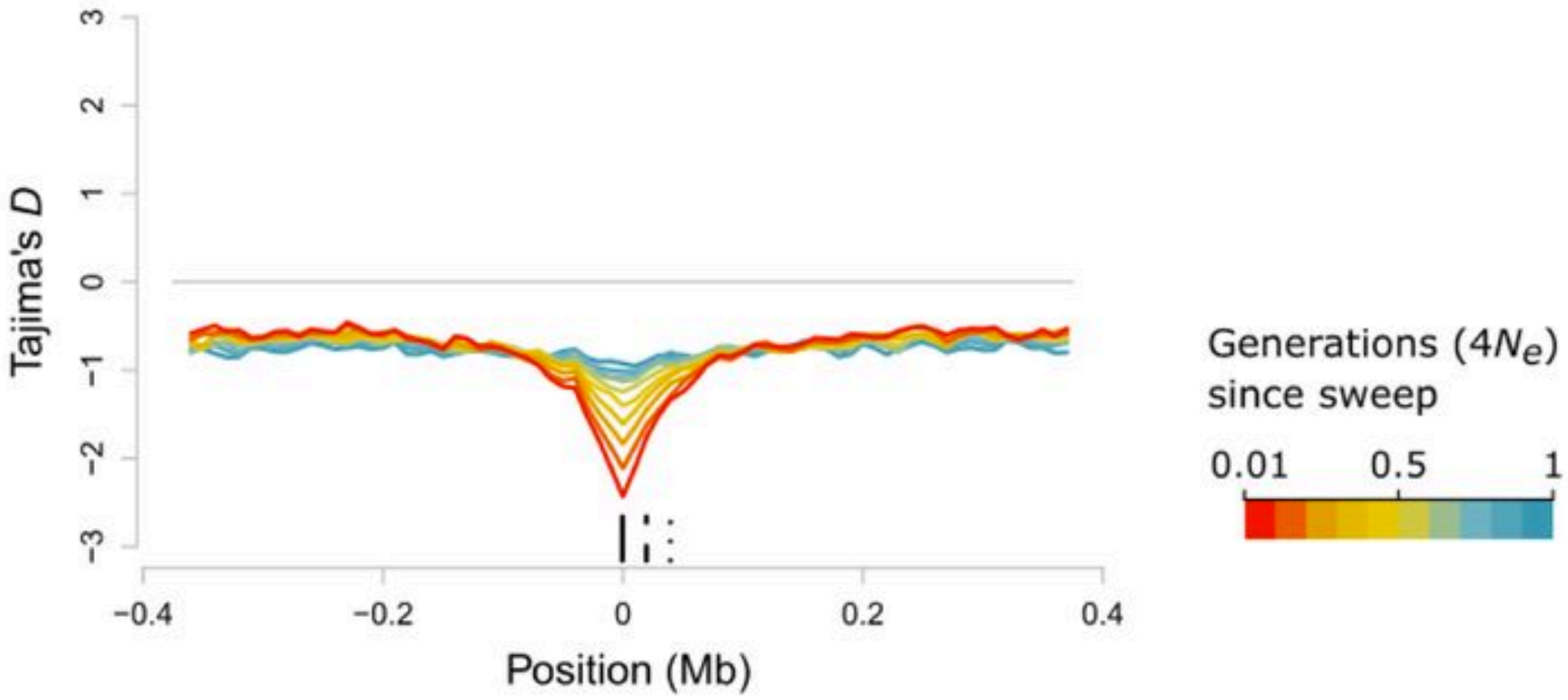
Proportion of segregating sites

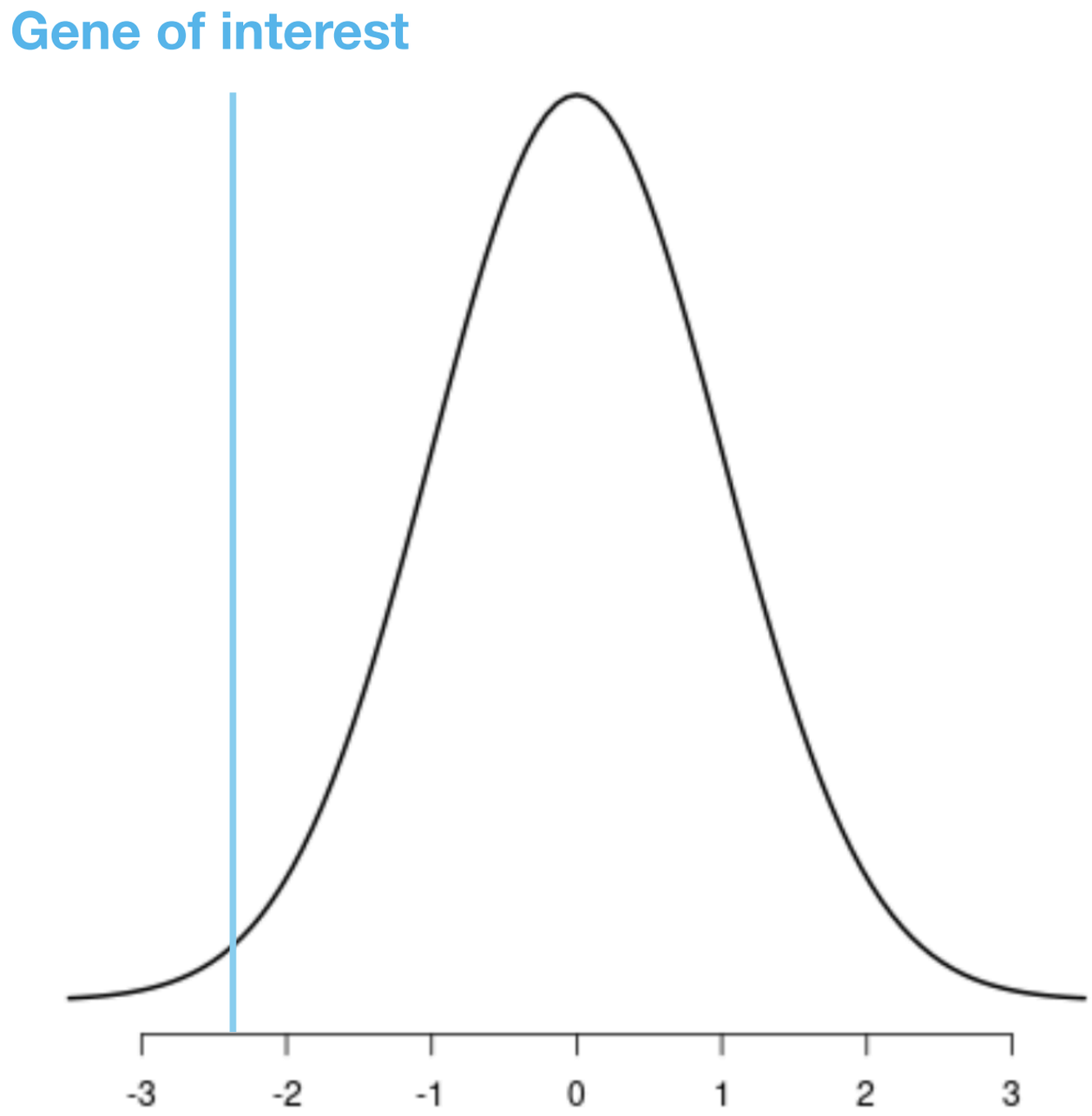
$$\theta_S = \frac{S}{a_1}$$

$$d = \theta_\pi - \theta_S$$

$$D = \frac{d}{\sqrt{V(d)}}$$

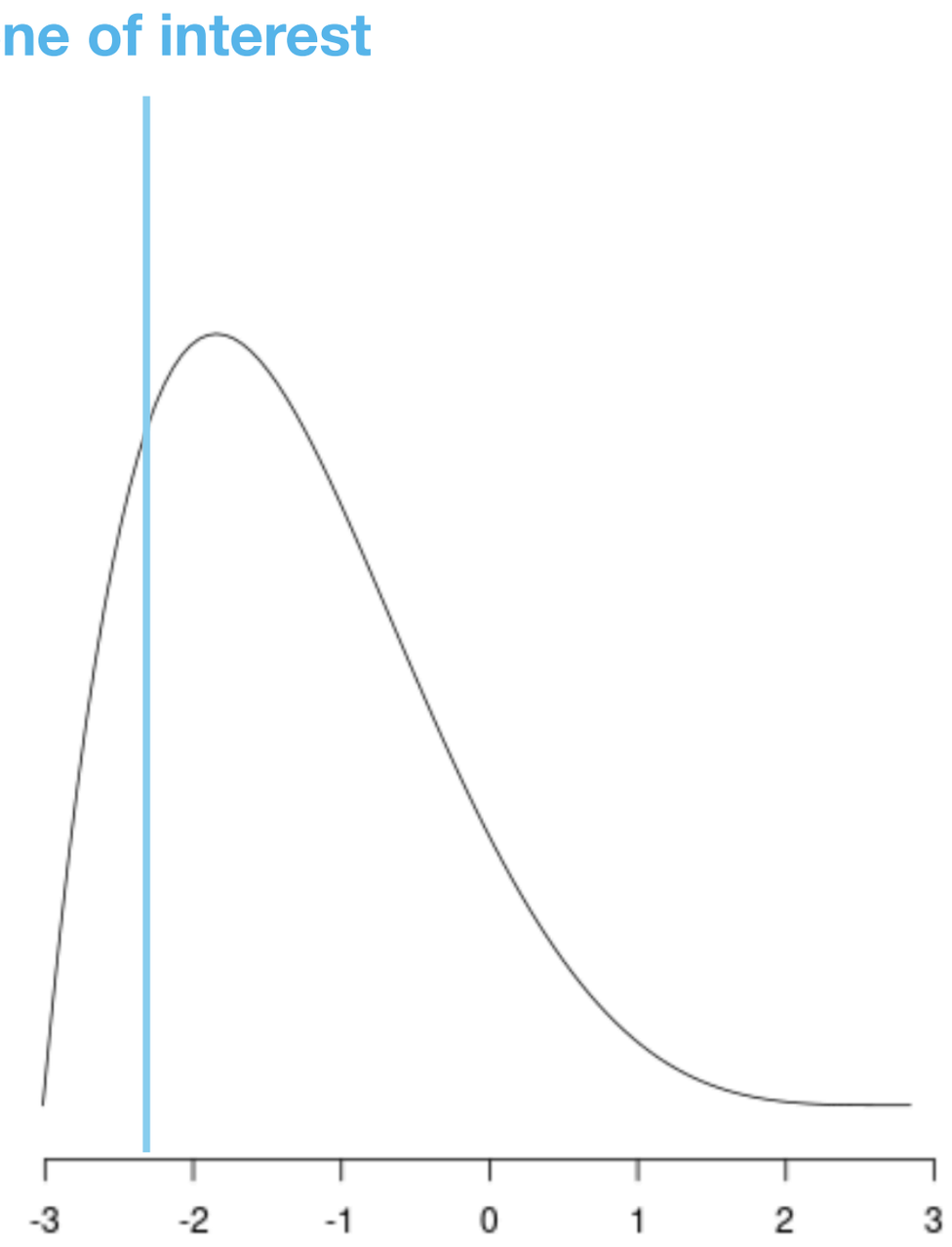
D > |2| is highly unlikely under a neutral model





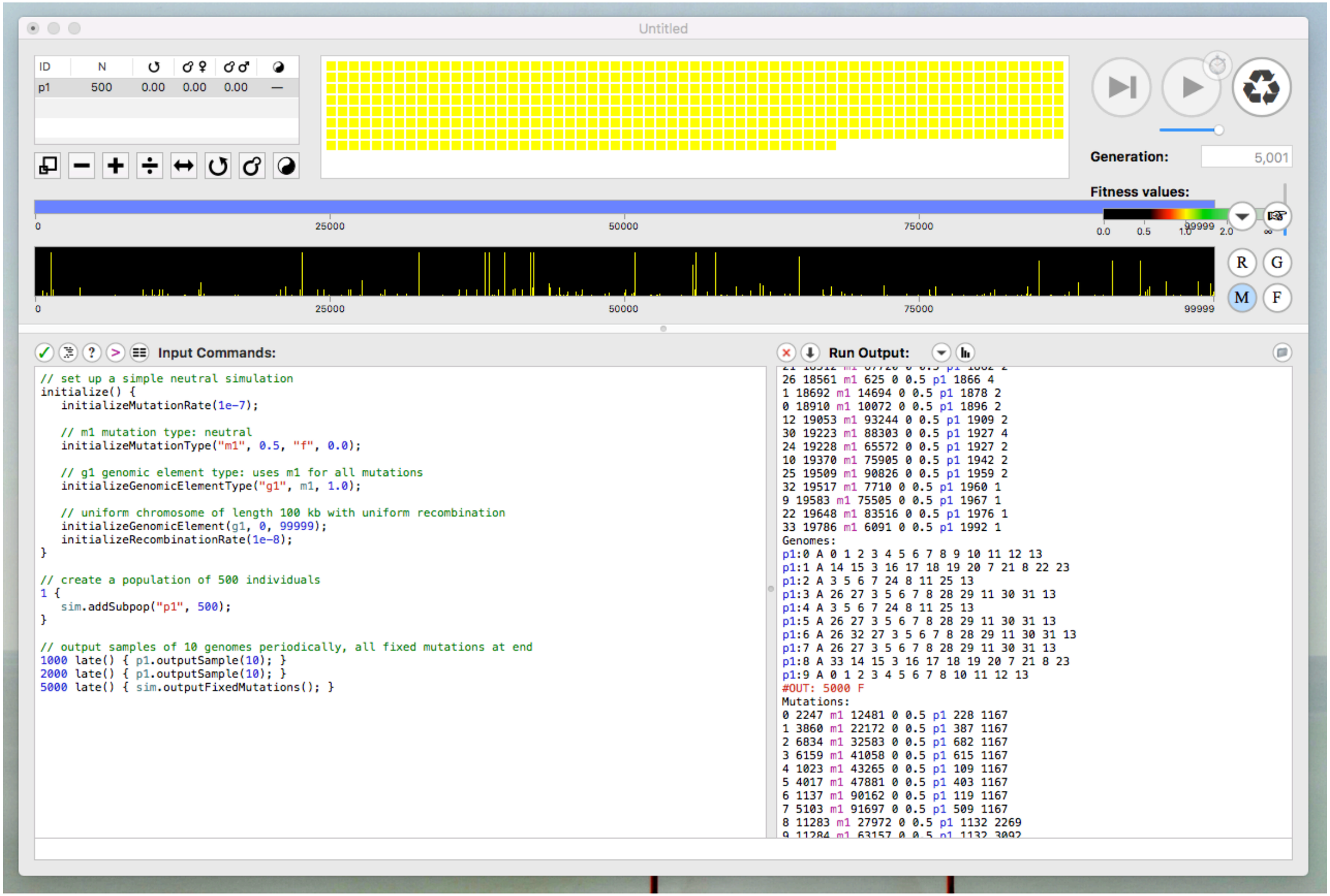
Tajima’s D

A population expansion will lead to an excess of rare variants, skewing the genome-wide distribution of Tajima's D



Given the population size (or μ, or sample size etc.) change the null distribution

SLiM


Untitled
ID N
p1 500 0.00 0.00 0.00 —
Generation: 5,001
Fitness values:
0 25000 50000 75000
0.0 0.5 1.0 99999 2.0 ∞
0 25000 50000 75000 99999
R G M F
Input Commands:
Run Output:
// set up a simple neutral simulation
initialize() {
initializeMutationRate(1e-7);
// m1 mutation type: neutral
initializeMutationType("m1", 0.5, "f", 0.0);
// g1 genomic element type: uses m1 for all mutations
initializeGenomicElementType("g1", m1, 1.0);
// uniform chromosome of length 100 kb with uniform recombination
initializeGenomicElement(g1, 0, 99999);
initializeRecombinationRate(1e-8);
}
// create a population of 500 individuals
1 {
sim.addSubpop("p1", 500);
}
// output samples of 10 genomes periodically, all fixed mutations at end
1000 late() { p1.outputSample(10); }
2000 late() { p1.outputSample(10); }
5000 late() { sim.outputFixedMutations(); }
26 18561 m1 625 0 0.5 p1 1866 4
1 18692 m1 14694 0 0.5 p1 1878 2
0 18910 m1 10072 0 0.5 p1 1896 2
12 19053 m1 93244 0 0.5 p1 1909 2
30 19223 m1 88303 0 0.5 p1 1927 4
24 19228 m1 65572 0 0.5 p1 1927 2
10 19370 m1 75905 0 0.5 p1 1942 2
25 19509 m1 90826 0 0.5 p1 1959 2
32 19517 m1 7710 0 0.5 p1 1960 1
9 19583 m1 75505 0 0.5 p1 1967 1
22 19648 m1 83516 0 0.5 p1 1976 1
33 19786 m1 6091 0 0.5 p1 1992 1
Genomes:
p1:0 A 0 1 2 3 4 5 6 7 8 9 10 11 12 13
p1:1 A 14 15 3 16 17 18 19 20 7 21 8 22 23
p1:2 A 3 5 6 7 24 8 11 25 13
p1:3 A 26 27 3 5 6 7 8 28 29 11 30 31 13
p1:4 A 3 5 6 7 24 8 11 25 13
p1:5 A 26 27 3 5 6 7 8 28 29 11 30 31 13
p1:6 A 26 32 27 3 5 6 7 8 28 29 11 30 31 13
p1:7 A 26 27 3 5 6 7 8 28 29 11 30 31 13
p1:8 A 33 14 15 3 16 17 18 19 20 7 21 8 23
p1:9 A 0 1 2 3 4 5 6 7 8 10 11 12 13
#OUT: 5000 F
Mutations:
0 2247 m1 12481 0 0.5 p1 228 1167
1 3860 m1 22172 0 0.5 p1 387 1167
2 6834 m1 32583 0 0.5 p1 682 1167
3 6159 m1 41058 0 0.5 p1 615 1167
4 1023 m1 43265 0 0.5 p1 109 1167
5 4017 m1 47881 0 0.5 p1 403 1167
6 1137 m1 90162 0 0.5 p1 119 1167
7 5103 m1 91697 0 0.5 p1 509 1167
8 11283 m1 27972 0 0.5 p1 1132 2269

SLiM uses eidos

Types (in promotion order):

- `NULL`: no explicit value
- `logical`: true/false values
- `integer`: whole numbers
- `float`: real numbers
- `string`: characters
- `object`: Context objects, such as SLiM objects

Constants:

- `E`: e (2.7182...) (float)
- `PI`: π (3.1415...) (float)
- `F`: false (logical)
- `T`: true (logical)
- `INF`: infinity (float)
- `NAN`: not a number (float)
- `NULL`: a `NULL`-type value

Operators (precedence order):

`[]`, `()`, `.`	subset, call, member
`+`, `−`, `!`	unary plus/minus, logical not
`^`	exponentiation
`:`	sequence construction
`*`, `/`, `%`	multiplication, division, modulo
`+`, `−`	addition and subtraction
`<`, `>`, `<=`, `>=`	less-than, greater-than, etc.
`==`, `!=`	equality and inequality
`&`	logical (Boolean) and
`|`	logical (Boolean) or
`?else`	ternary conditional
`=`	assignment

Empty statement: `;`
Compound statement: `{ ... }`
Single-line comment: `// ...`
Block comment: `/* ... */`

SLiM uses eidos

Special Statements:

`if (condition) statement [else statement]`	conditional statement with optional alternative statement
`while (condition) statement`	loop while `T`, with a condition test at the loop top
`do statement while (condition)`	loop while `T`, with a condition test at the loop bottom
`for (identifier in vector) statement`	iterate through the values in a vector, executing statement
`next`	skip the remainder of this iteration of the enclosing loop
`break`	break out of the enclosing loop entirely
`return [return-value]`	exit a script block, returning a value if one is given
`function (return)name(params) { ... }`	create a user-defined function (only at the top level)

SLiM uses eidos

Math:

(numeric)abs(numeric x): absolute value of x
(float)acos(numeric x): arc cosine of x
(float)asin(numeric x): arc sine of x
(float)atan(numeric x): arc tangent of x
(float)atan2(numeric x, numeric y): arc tangent of y/x, inferring the correct quadrant
(float)ceil(float x): ceiling (rounding toward +∞) of x
(float)cos(numeric x): cosine of x
(numeric)cumProduct(numeric x): cumulative product along x
(numeric)cumSum(numeric x): cumulative summation along x
(float)exp(numeric x): base-*e* exponential of x, e^x
(float)floor(float x): floor (rounding toward −∞) of x
(integer)integerDiv(integer x, integer y): integer division of x by y
(integer)integerMod(integer x, integer y): integer modulo of x by y (the remainder after integer division)
(logical)isFinite(float x): T or F for each element of x; "finite" means not INF, −INF, or NAN
(logical)isInfinite(float x): T or F for each element of x; "infinite" means INF and −INF only
(logical)isNAN(float x): T or F for each element of x; "infinite" means NAN only
(float)log(numeric x): base-*e* logarithm of x
(float)log10(numeric x): base-10 logarithm of x
(float)log2(numeric x): base-2 logarithm of x
(numeric$)product(numeric x): product of the elements of x, Πx
(float)round(float x): round x to the nearest values; half-way cases round away from 0
(*)setDifference(* x, * y): set-theoretic difference, x ∖ y
(*)setIntersection(* x, * y): set-theoretic intersection, x ∩ y
(*)setSymmetricDifference(* x, * y): set-theoretic symmetric difference x Δ y
(*)setUnion(* x, * y): set-theoretic union, x ∪ y
(float)sin(numeric x): sine of x
(float)sqrt(numeric x): square root of x
(numeric$)sum(lif x): summation of the elements of x, Σx
(float$)sumExact(float x): exact summation of x without roundoff error, to the limit of floating-point precision
(float)tan(numeric x): tangent of x

Anatomy of a basic simulation (5000 generations)

Initialise (callback function)

```
// Instantaneous population expansion from 1000 to 7500 at generation 4000 with no selection
initialize()
{
   // set the overall mutation rate
   initializeMutationRate(1e-5);

   // m1 mutation type: neutral
   initializeMutationType("m1", 0.5, "f", 0.0);

   // g1 genomic element type: uses m1 for all mutations
   initializeGenomicElementType("g1", m1, 1.0);

   // Chromosome of length 10 kb, homogenous (all of type g1)
   initializeGenomicElement(g1, 0, 9999);

   // uniform recombination along the chromosome
   initializeRecombinationRate(1e-8);
}

// create a population of 1000 individuals
1
{
   sim.addSubpop("p1", 1000);
}
// at generation 4000 increase the population size to 7500 immediately

4000 { p1.setSubpopulationSize(7500); }
```

Changes that occur during the course of the simulation

late

```
// run to generation 5000
5000 late() {

// randomly subsample 20 individuals for output
allIndividuals = sim.subpopulations.individuals;
 sampledIndividuals = sample(allIndividuals, 20);
 sampledIndividuals.genomes.outputVCF();

}
```

Anatomy of a basic simulation (5000 generations)

Initialise (callback function)

```
// set the overall mutation rate
initializeMutationRate(1e-5);
```

Changes that occur during the course of the simulation

late

Anatomy of a basic simulation (5000 generations)

Initialise (callback function)

Changes that occur during the course of the simulation

late

Fitness effects are 1+s and 1+hs. Here s is fixed at 0.

```
// m1 mutation type: neutral
initializeMutationType("m1", 0.5, "f", 0.0);
```

Call it “m1”

Dominance coef.

“Fixed” DFE (from which selection coefficient is drawn)

Anatomy of a basic simulation (5000 generations)

Initialise (callback function)

100% of mutations in “g1” are drawn from g1

```
// g1 genomic element type: uses m1 for all mutations
initializeGenomicElementType("g1", m1, 1.0);

// Chromosome of length 10 kb, homogenous (all of type g1)
initializeGenomicElement(g1, 0, 9999);
```

Changes that occur during the course of the simulation

late

Anatomy of a basic simulation (5000 generations)

Initialise (callback function)

```
// uniform recombination along the chromosome
initializeRecombinationRate(1e-8);
```

Changes that occur during the course of the simulation

late

Anatomy of a basic simulation (5000 generations)

Initialise (callback function)

```
// uniform recombination along the chromosome
initializeRecombinationRate(1e-8);
```

Changes that occur during the course of the simulation

late

Anatomy of a basic simulation (5000 generations)

Initialise (callback function)

```
// create a population of 1000 individuals
1
{
   sim.addSubpop("p1", 1000);
}
```

Changes that occur during the course of the simulation

late

Anatomy of a basic simulation (5000 generations)

Initialise (callback function)

```
// Instantaneous population expansion from 1000 to 7500 at generation 4000 with no selection
initialize()
{
   // set the overall mutation rate
   initializeMutationRate(1e-5);

   // m1 mutation type: neutral
   initializeMutationType("m1", 0.5, "f", 0.0);

   // g1 genomic element type: uses m1 for all mutations
   initializeGenomicElementType("g1", m1, 1.0);

   // Chromosome of length 10 kb, homogenous (all of type g1)
   initializeGenomicElement(g1, 0, 9999);

   // uniform recombination along the chromosome
   initializeRecombinationRate(1e-8);
}

// create a population of 1000 individuals
1
{
   sim.addSubpop("p1", 1000);
}
```

Changes that occur during the course of the simulation

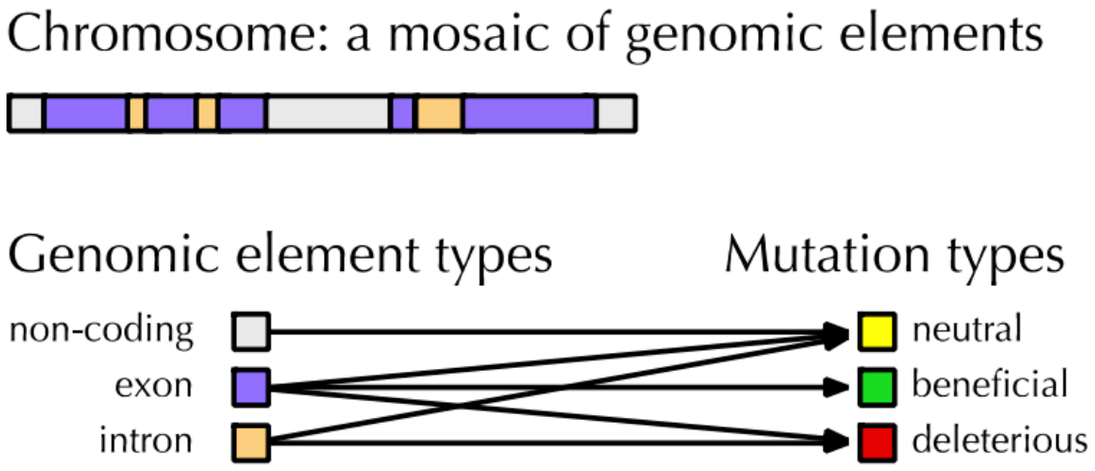
late

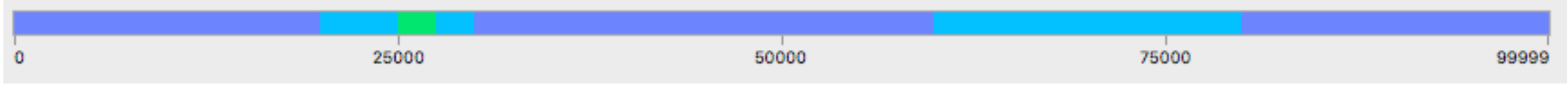
Other mutation types

```
// Mutation types
initializeMutationType("m1", 0.5, "f", 0.0); // neutral
initializeMutationType("m2", 0.1, "e", 0.1); // ben.
initializeMutationType("m3", 0.2, "g", -0.03,0.2); // del.

// Genomic element types
initializeGenomicElementType("g1", m1, 1.0); // nongenic
initializeGenomicElementType("g2", c(m1,m2,m3), c(100,1,10)); //exon
initializeGenomicElementType("g3", c(m1,m3), c(90,10)); //intron

// Genomic elements
initializeGenomicElement(g1, 0, 19999);
initializeGenomicElement(g2, 20000, 24999);
initializeGenomicElement(g3, 25000, 27499);
initializeGenomicElement(g2, 27500, 29999);
initializeGenomicElement(g1, 30000, 59999);
initializeGenomicElement(g2, 60000, 79999);
initializeGenomicElement(g1, 80000, 99999);
initializeRecombinationRate(1e-8);
```





Anatomy of a basic simulation (5000 generations)

Initialise (callback function)

```
// Instantaneous population expansion from 1000 to 7500 at generation 4000 with no selection
initialize()
{
   // set the overall mutation rate
   initializeMutationRate(1e-5);

   // m1 mutation type: neutral
   initializeMutationType("m1", 0.5, "f", 0.0);

   // g1 genomic element type: uses m1 for all mutations
   initializeGenomicElementType("g1", m1, 1.0);

   // Chromosome of length 10 kb, homogenous (all of type g1)
   initializeGenomicElement(g1, 0, 9999);

   // uniform recombination along the chromosome
   initializeRecombinationRate(1e-8);
}

// create a population of 1000 individuals
1
{
   sim.addSubpop("p1", 1000);
}
```

Changes that occur during the course of the simulation

late

Anatomy of a basic simulation (5000 generations)

Initialise (callback function)

```
// Instantaneous population expansion from 1000 to 7500 at generation 4000 with no selection
initialize()
{
   // set the overall mutation rate
   initializeMutationRate(1e-5);

   // m1 mutation type: neutral
   initializeMutationType("m1", 0.5, "f", 0.0);

   // g1 genomic element type: uses m1 for all mutations
   initializeGenomicElementType("g1", m1, 1.0);

   // Chromosome of length 10 kb, homogenous (all of type g1)
   initializeGenomicElement(g1, 0, 9999);

   // uniform recombination along the chromosome
   initializeRecombinationRate(1e-8);
}

// create a population of 1000 individuals
1
{
   sim.addSubpop("p1", 1000);
}
// at generation 4000 increase the population size to 7500 immediately

4000 { p1.setSubpopulationSize(7500); }
```

Changes that occur during the course of the simulation

late

Anatomy of a basic simulation (5000 generations)

Initialise (callback function)

```
// Instantaneous population expansion from 1000 to 7500 at generation 4000 with no selection
initialize()
{
   // set the overall mutation rate
   initializeMutationRate(1e-5);

   // m1 mutation type: neutral
   initializeMutationType("m1", 0.5, "f", 0.0);

   // g1 genomic element type: uses m1 for all mutations
   initializeGenomicElementType("g1", m1, 1.0);

   // Chromosome of length 10 kb, homogenous (all of type g1)
   initializeGenomicElement(g1, 0, 9999);

   // uniform recombination along the chromosome
   initializeRecombinationRate(1e-8);
}

// create a population of 1000 individuals
1
{
   sim.addSubpop("p1", 1000);
}
// at generation 4000 increase the population size to 7500 immediately

4000 { p1.setSubpopulationSize(7500); }
```

Changes that occur during the course of the simulation

late

```
// run to generation 5000
5000 late() {

// randomly subsample 20 individuals for output
allIndividuals = sim.subpopulations.individuals;
 sampledIndividuals = sample(allIndividuals, 20);
 sampledIndividuals.genomes.outputVCF();

}
```

What now?

- Directly calculate summary statistics from the .vcf files just as you would with “real” VCFs

- Run each simulation many times (100s-1000s) to obtain a null distribution of whatever summary statistic you are interested in

Negative FST?

- Possible if FST is calculated using Wier and Cockeham's (1984) method. It means there is more variation within than between populations and can result from uneven sampling