“R and SLiM walk into a bar...”

— introduction to *slendr*, plus a few tips & tricks discovered during its development

www.slendr.net

Martin Petr
mp@bodkan.net
@fleventy5

Globe Institute
University of Copenhagen

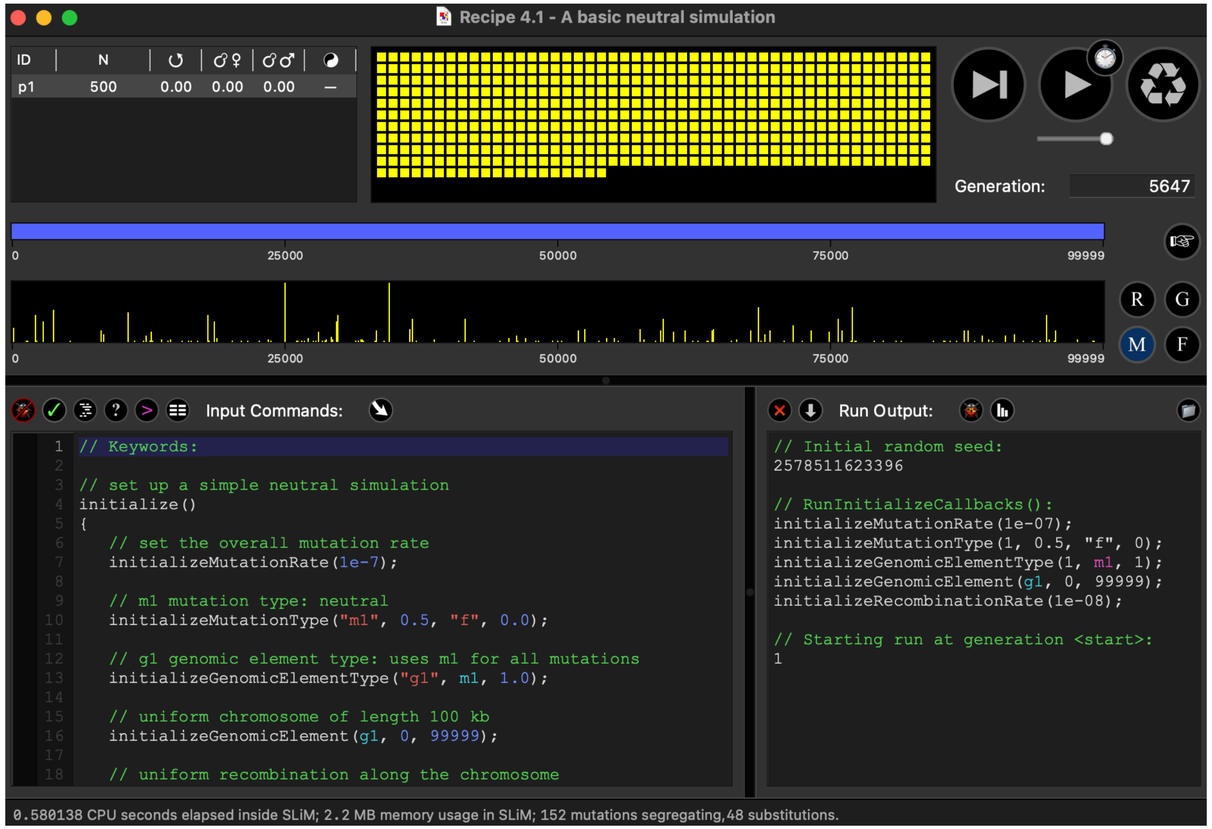
How to simulate realistic spatial data?

Haak *et al.* 2015

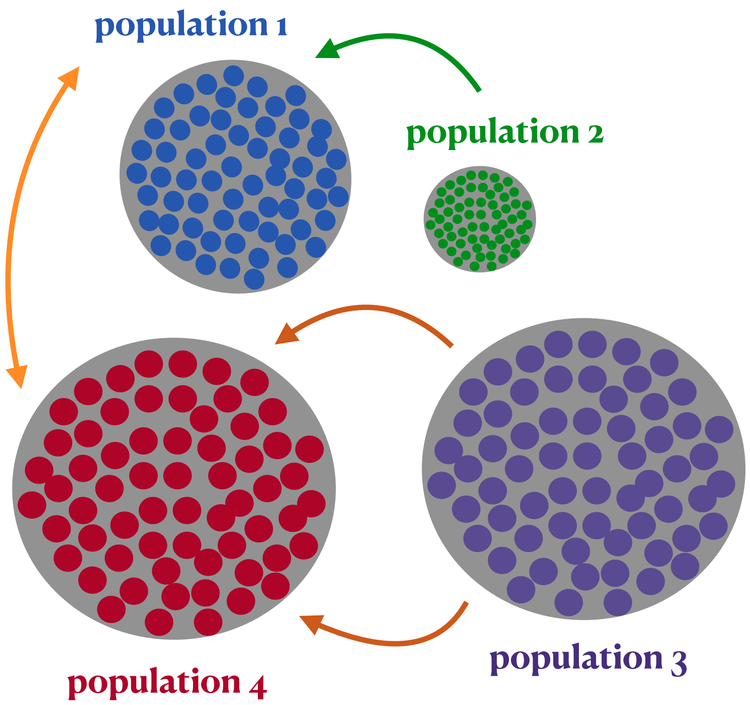
SLiM as the obvious first building block!

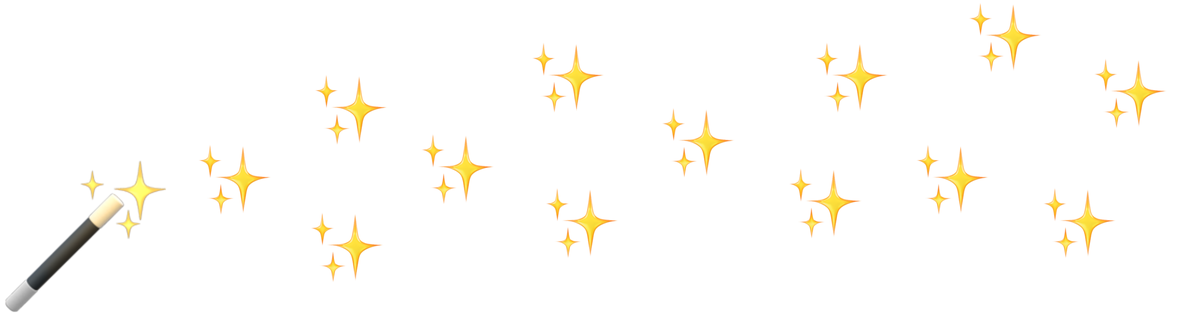
- extremely popular and powerful
- custom programming language
- complex demographic models
- custom mating schemes

... also continuous space!

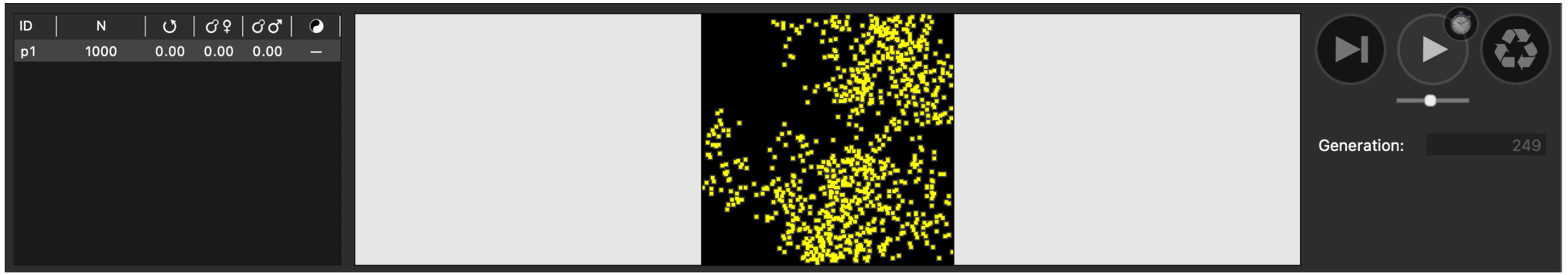


Random-mating simulation:



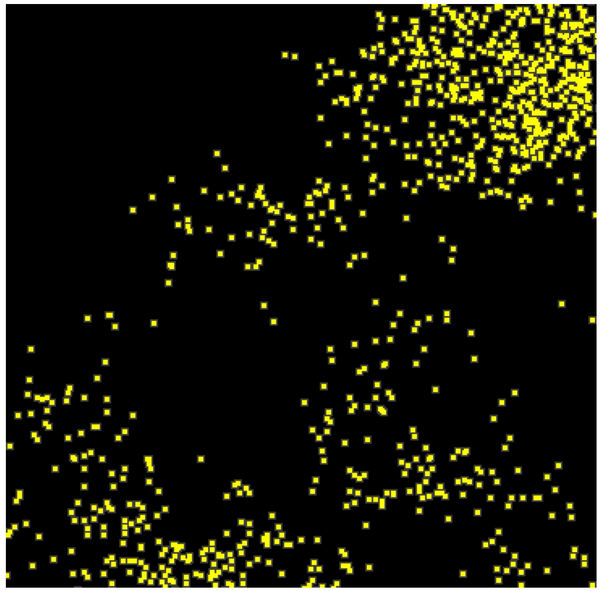


```
initializeSLiMOptions(dimensionality="xy");
```



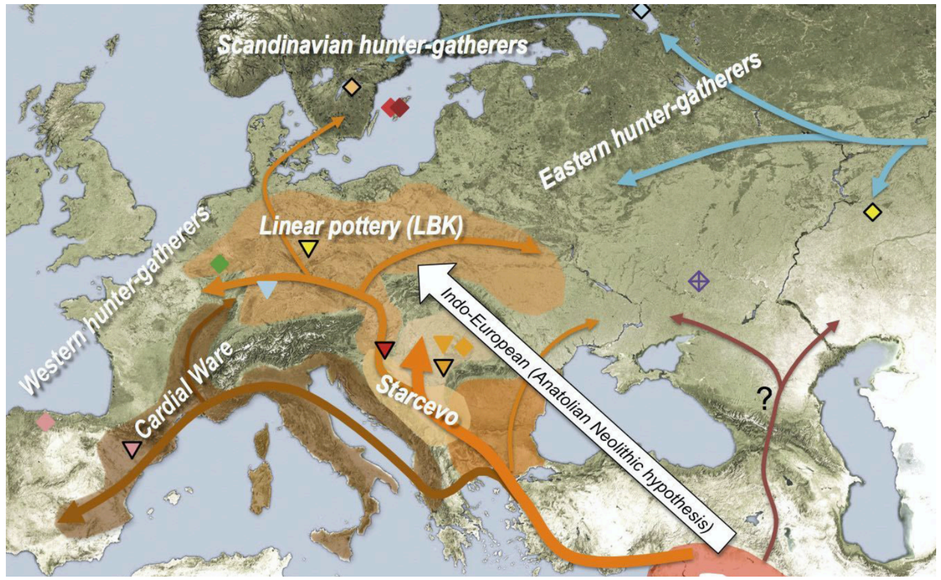
Each individual has a coordinate on a two-dimensional plane!

How can we get from here...



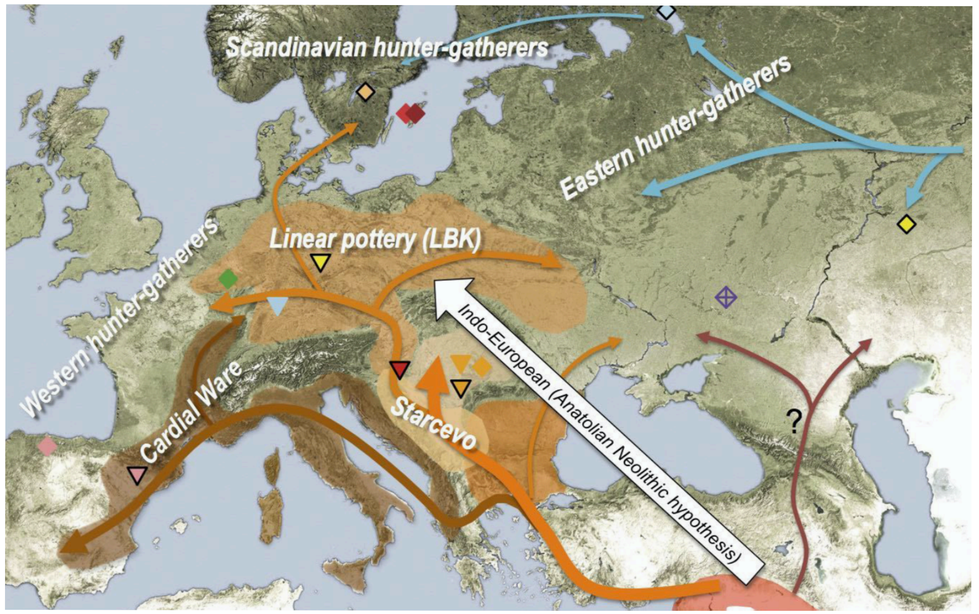
?

... to here?



Design goal: a programmable framework for spatial popgen

- create a "mini-language" for formal definition of
  - population ranges, movement, expansion, ...
  - no geospatial know-how required!
- use SLiM as a “low-level” engine
- implement in
- build a research tool, but also a means of exploration



R package ***slendr***

www.slendr.net

Model definition & simulation

Model components

`population()`	Define a population and its original spatial range
`world()`	Define a world map for all spatial operations
`geneflow()`	Define a geneflow event
`region()`	Define a geographic region

Population dynamics

`move()`	Move the population to a new location in a given amount of time
`expand()`	Expand the population range
`shrink()`	Shrink the population range
`boundary()`	Update the population range
`resize()`	Resize the population size
`dispersal()`	Change dispersal parameters

Manipulation of spatial objects

`join()`	Merge two spatial `slendr` objects into one
`overlap()`	Generate the overlap of two `slendr` objects
`subtract()`	Generate the difference between two `slendr` objects
`reproject()`	Reproject coordinates between coordinate systems
`distance()`	Calculate the distance between a pair of spatial boundaries
`area()`	Calculate the area covered by the given slendr object
`dimensions()`	Return the dimensions of the world map

Model visualization and diagnostics

`plot(<slendr>)`	Plot `slendr` geographic features on a map
`plot_graph()`	Plot geneflow graph based on given model configuration
`plot_ancestry()`	Plot simulated ancestry proportions
`animate()`	Animate the simulated population dynamics
`explore()`	Open an interactive browser of the spatial model

Compiling and running spatial models

`compile()`	Compile the spatial demographic model
`sampling()`	Define sampling events at specified times for a given set of populations
`slim()`	Run a slendr model as a SLiM script
`read()`	Read a previously serialized model configuration
`script()`	Substitute variables in a template SLiM script

Tree-sequence analysis

Tree sequence loading and processing

`ts_load()`	Load a tree sequence file produced by a given model
`ts_recapitate()`	Recapitate the tree sequence
`ts_simplify()`	Simplify the tree sequence down to a given set of individuals
`ts_mutate()`	Add mutations to the given tree sequence
`ts_coalesced()`	Check that all trees in the tree sequence are fully coalesced

Extracting genotypes from tree sequences

`ts_genotypes()`	Extract genotype table from the tree sequence
`ts_eigenstrat()`	Extract genotypes from the tree sequence in the EIGENSTRAT format
`ts_vcf()`	Save genotypes from the tree sequenceas a VCF file

Accessing tree sequence components

`ts_data()`	Extract combined annotated table of individuals and nodes
`ts_ancestors()`	Infer spatio-temporal ancestral history for given nodes/individuals
`ts_individuals()` `ts_edges()` `ts_nodes()`	Get the table of individuals/nodes/edges from the tree sequence
`ts_samples()`	Extract names and times of individuals scheduled for sampling
`ts_tree()`	Get a tree from a given tree sequence
`ts_draw()`	Plot a graphical representation of a single tree

Tree sequence statistics

`ts_f2()` `ts_f3()` `ts_f4()` `ts_f4ratio()`	Calculate the f2, f3, f4, and f4-ratio statistics
`ts_afs()`	Compute the allele frequency spectrum (AFS)
`ts_fst()` `ts_divergence()`	Calculate pairwise statistics between sets of individuals
`ts_diversity()`	Calculate diversity in given sets of individuals
`ts_tajima()`	Calculate Tajima's D for given sets of individuals
`ts_segregating()`	Calculate the density of segregating sites for the given sets of individuals

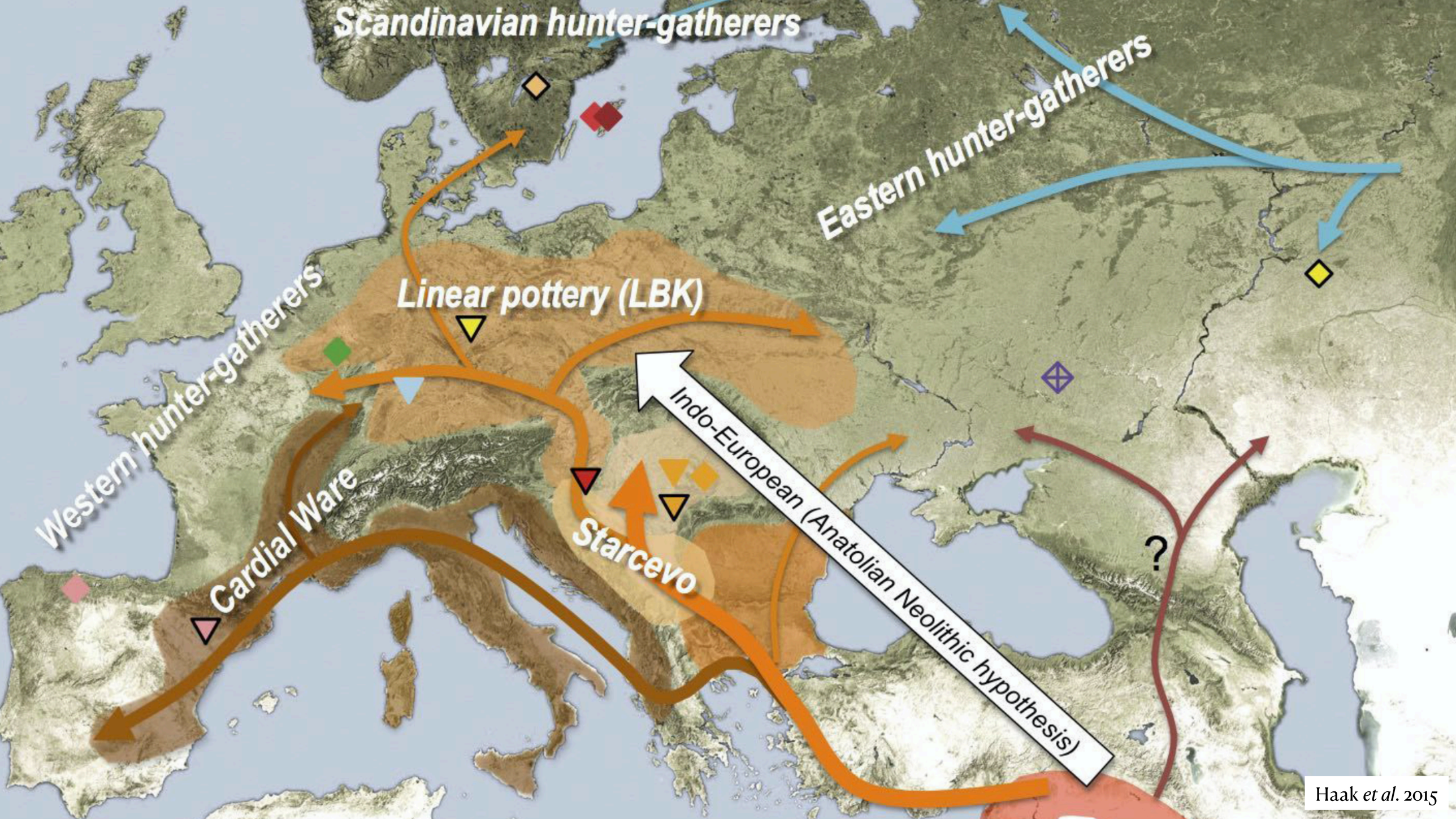
slendr workflow demo

interactive RStudio session in the browser:

`github.com/bodkan/slendr`

click on launch binder

(no installation needed!)

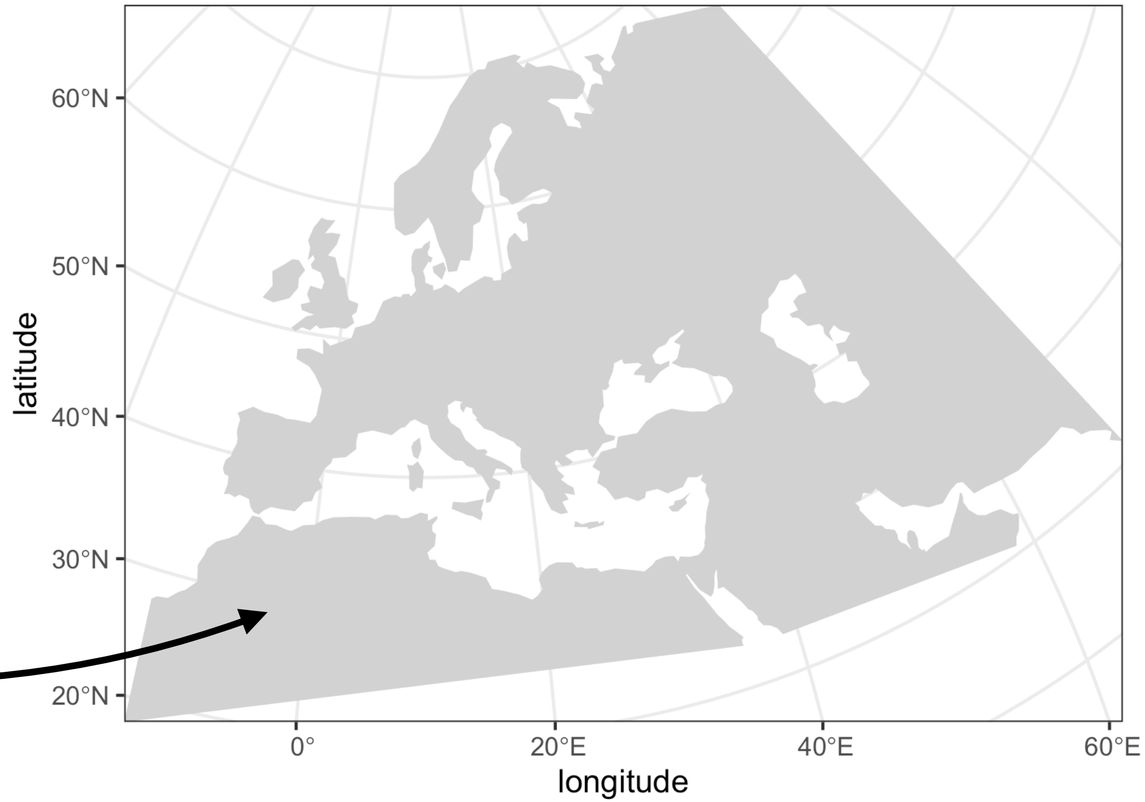


Haak *et al*. 2015

1. Define a spatial context ("real `world()`")

(section of the world which will be occupied by populations)

```
map <- world(
  xrange = c(-13, 70),          # min-max longitude
  yrange = c(18, 65),           # min-max latitude
  landscape = "naturalearth", # source of data
  crs = "EPSG:3035"             # coordinate reference system (CRS)
)
```

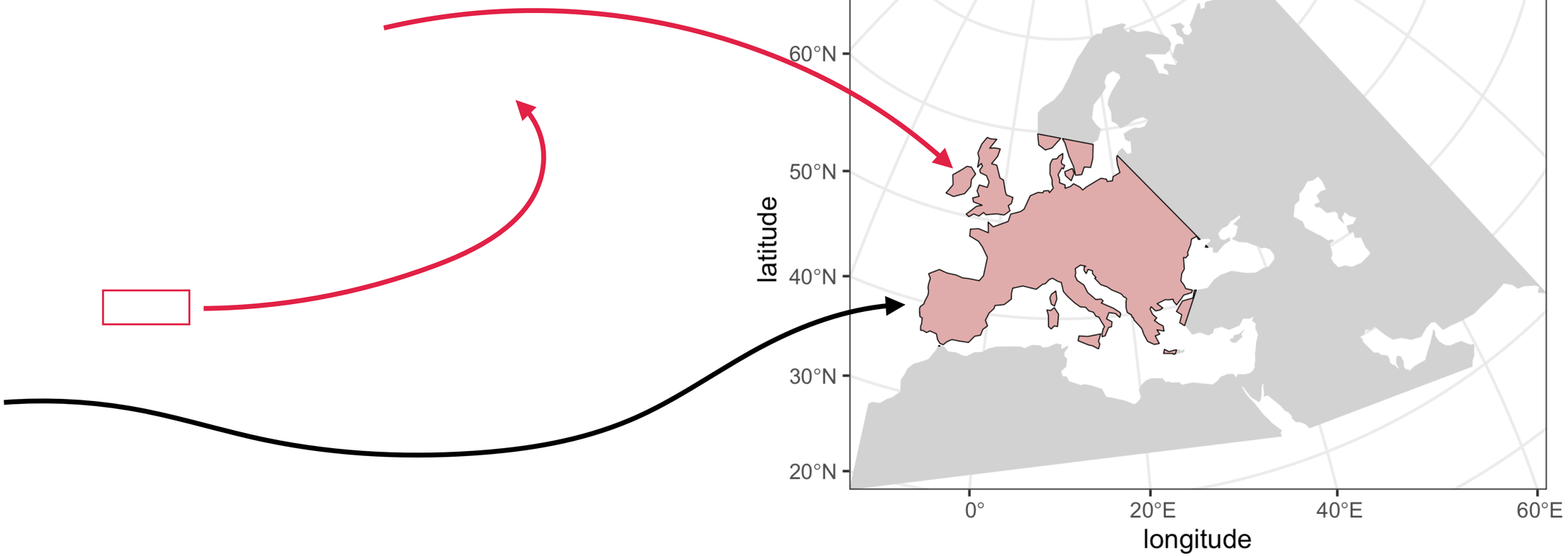


(using freely available *Natural Earth* cartographic data)

2. `population()` (and its boundaries)

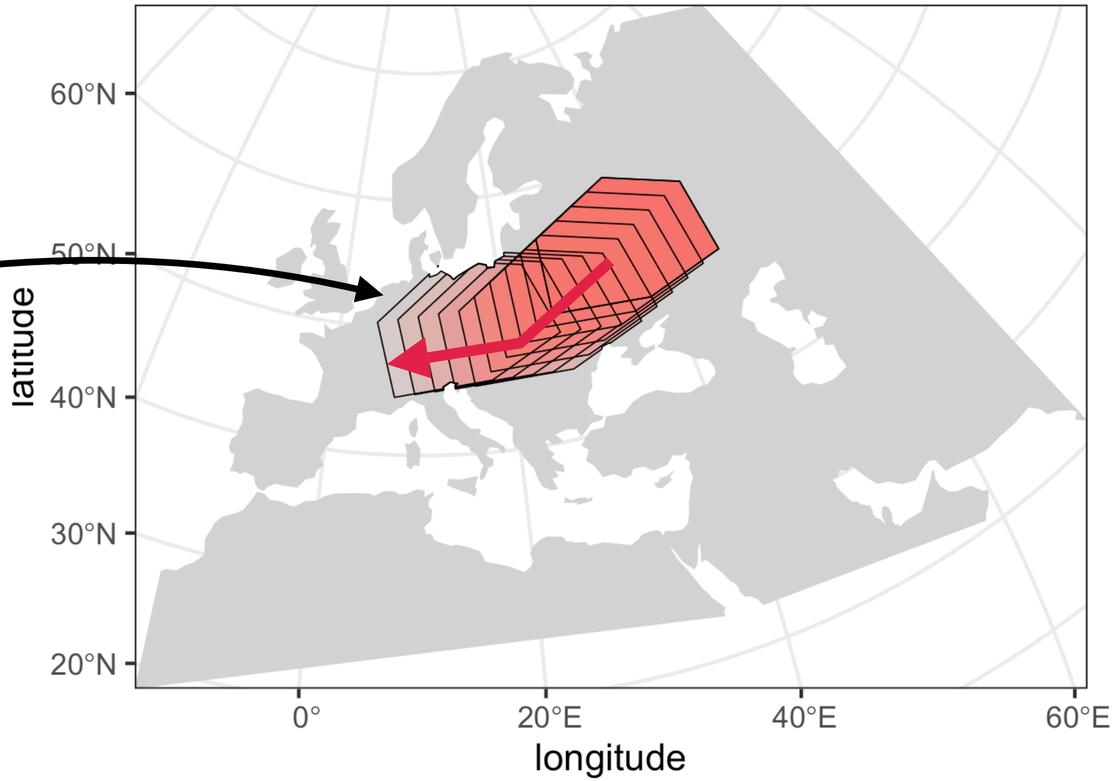
Everything is stored in a vectorised form!

```
> eur$geometry[[1]]
POLYGON ((2676851 1520500, 2568327 3290814, 3489173 4188384, 4910944 3919000,
6104864 2710417, 5695390 1461722, 5240847 1387400, 4321000 1657812, 2965317 15
70227, 2676851 1520500))
```



3. Population displacement

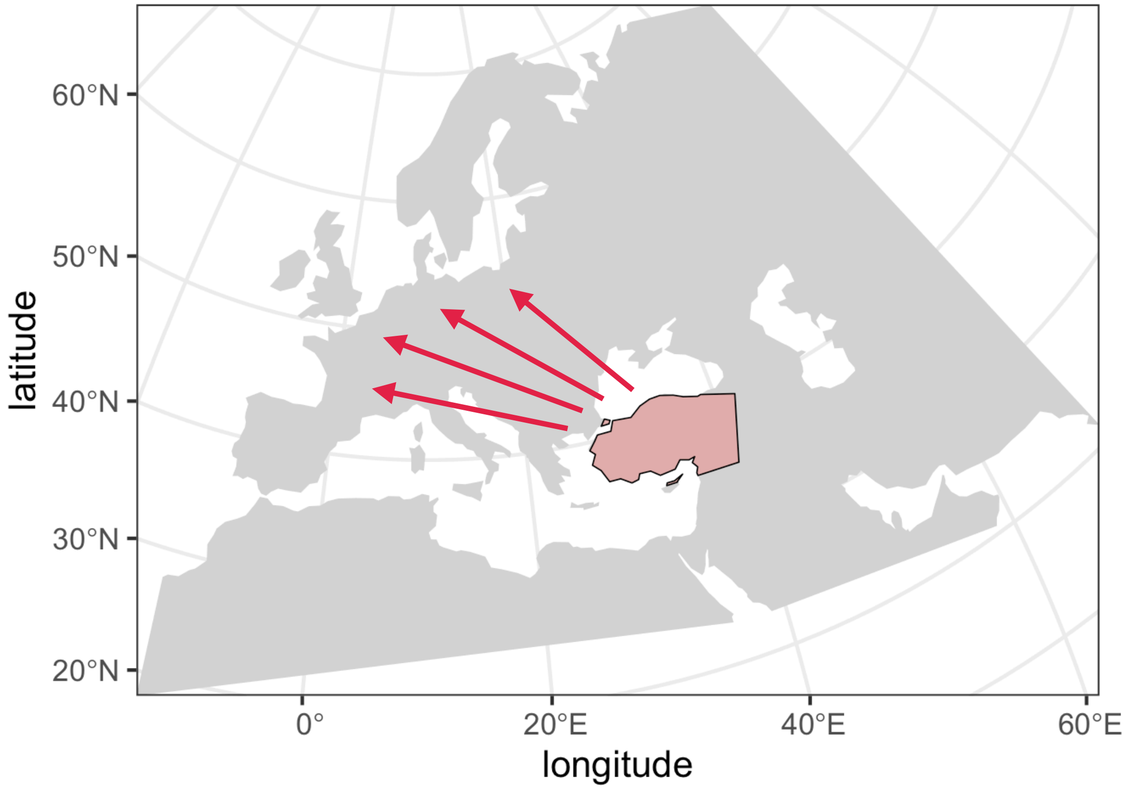
```
yam <- population( # Yamnaya steppe population
  name = "YAM", time = 7000, N = 500, remove = 2500, map = map,
  polygon = list(c(26, 50), c(38, 49), c(48, 50),
                 c(48, 56), c(38, 59), c(26, 56))
)
```




```
> yam
slendr 'population' object
--------------------------
name: YAM
habitat: terrestrial

number of spatial maps: 11
map: internal coordinate reference system EPSG 3035
scheduled removal at time  2500

population history overview:
  - time 7000: created as an ancestral population (N = 500)
  - time 5000-3000: movement across a landscape
```

4. Population range expansion

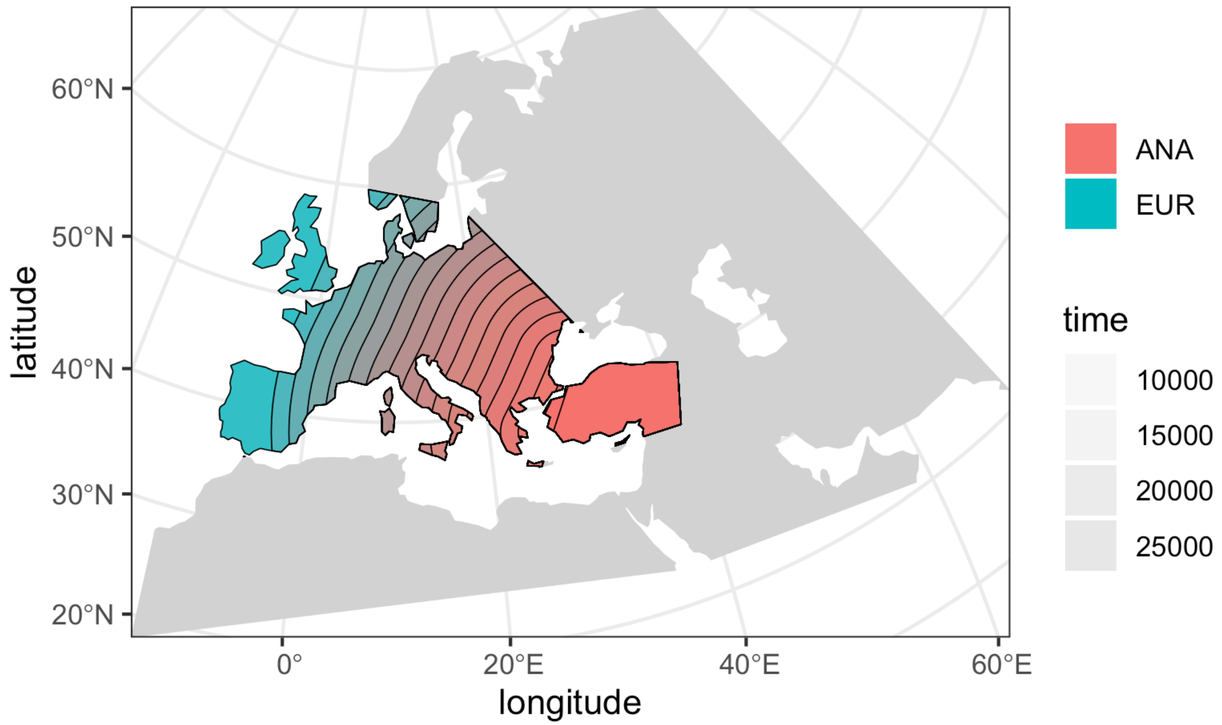
```
ana <- population( # Anatolian farmers
  name = "ANA", time = 28000, N = 3000, map = map,
  center = c(34, 38), radius = 500e3, polygon = anatolia
)
```



5. `gene_flow()` events

```
eur <- population(
  name = "EUR", time = 25000, N = 2000,
  map = map, polygon = europe
)
```

```
ana <- population( # Anatolian farmers
  name = "ANA", time = 28000, N = 3000, map = map,
  center = c(34, 38), radius = 500e3, polygon = anatolia
) %>%
  expand_range( # expand the range by 2.500 km
    by = 2500e3, start = 10000, end = 7000,
    polygon = join(europe, anatolia),
  )
```




```
gf <- gene_flow(from = ana, to = eur,
                rate = 0.5, start = 8000, end = 6000)
```

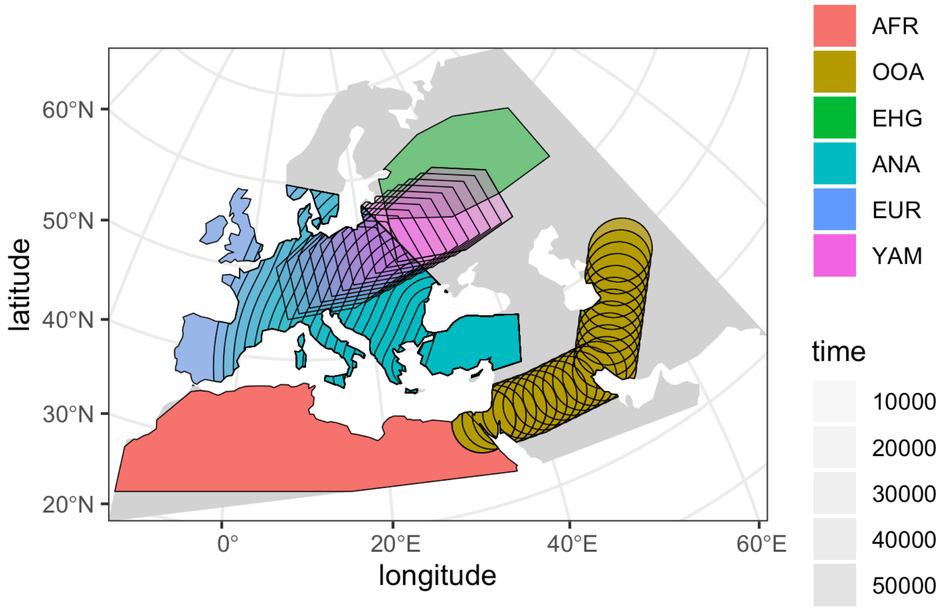
(`gene_flow()` performs sanity checks to make sure the event makes sense as programmed)

6. Model compilation

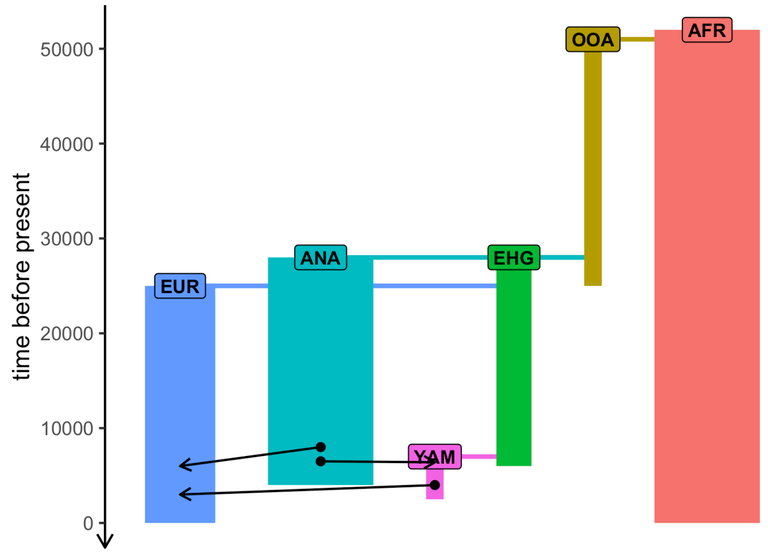
we created these R objects earlier

```
model <- compile_model(
  populations = list(afr, ooa, ehg, eur, ana, yam),
  gene_flow = gf,
  generation_time = 30,
  resolution = 10e3,                    # resolution in meters per pixel
  competition = 130e3, mating = 100e3, # spatial interaction parameters
  dispersal = 70e3                      # how far can offspring 'migrate'
)
```

`plot_map(model)` — "compressed" spatial dynamics



`plot_model(model)` — embedded demographic model



How is a *slendr* `model` represented in the background?

`compile_model()` **generates these files on disk:**

1.png	PNG image
2.png	PNG image
3.png	PNG image
4.png	PNG image
5.png	PNG image
[...]	
57.png	PNG image
58.png	PNG image
59.png	PNG image
60.png	PNG image
checksums.tsv	Plain Text
description.txt	Plain Text
direction.txt	Plain Text
dispersals.tsv	Plain Text
geneflow.tsv	Plain Text
generation_time.txt	Plain Text
length.txt	Plain Text
maps.tsv	Plain Text
orig_length.txt	Plain Text
populations.tsv	Plain Text
ranges.rds	R Data File
resolution.txt	Plain Text
script.py	Python Source
script.slim	SLiM model

a bunch of image files

a bunch of text files

a Python script

a SLiM script

```
model <- compile_model(
  populations = list(afr, ooa, ehg, eur, ana, yam),
  gene_flow = gf,
  generation_time = 30,
  resolution = 10e3,                    # resolution in meters per pixel
  competition = 130e3, mating = 100e3, # spatial interaction parameters
  dispersal = 70e3                      # how far can offspring 'migrate'
)
```

```
> model
slendr 'model' object
---------------------
populations: AFR, OOA, EHG, ANA, EUR, YAM
geneflow events: 3
generation time: 30
time direction: backward
total running length: 52000 model time units
model type: spatial
  - number of spatial maps: 60
  - resolution: 10000 distance units per pixel

configuration files in: /private/var/folders/70/b_q2zdh116b9pfg29p
03sx600000gn/T/RtmpZwjanI/filed3d867bb9977_slendr_model
```

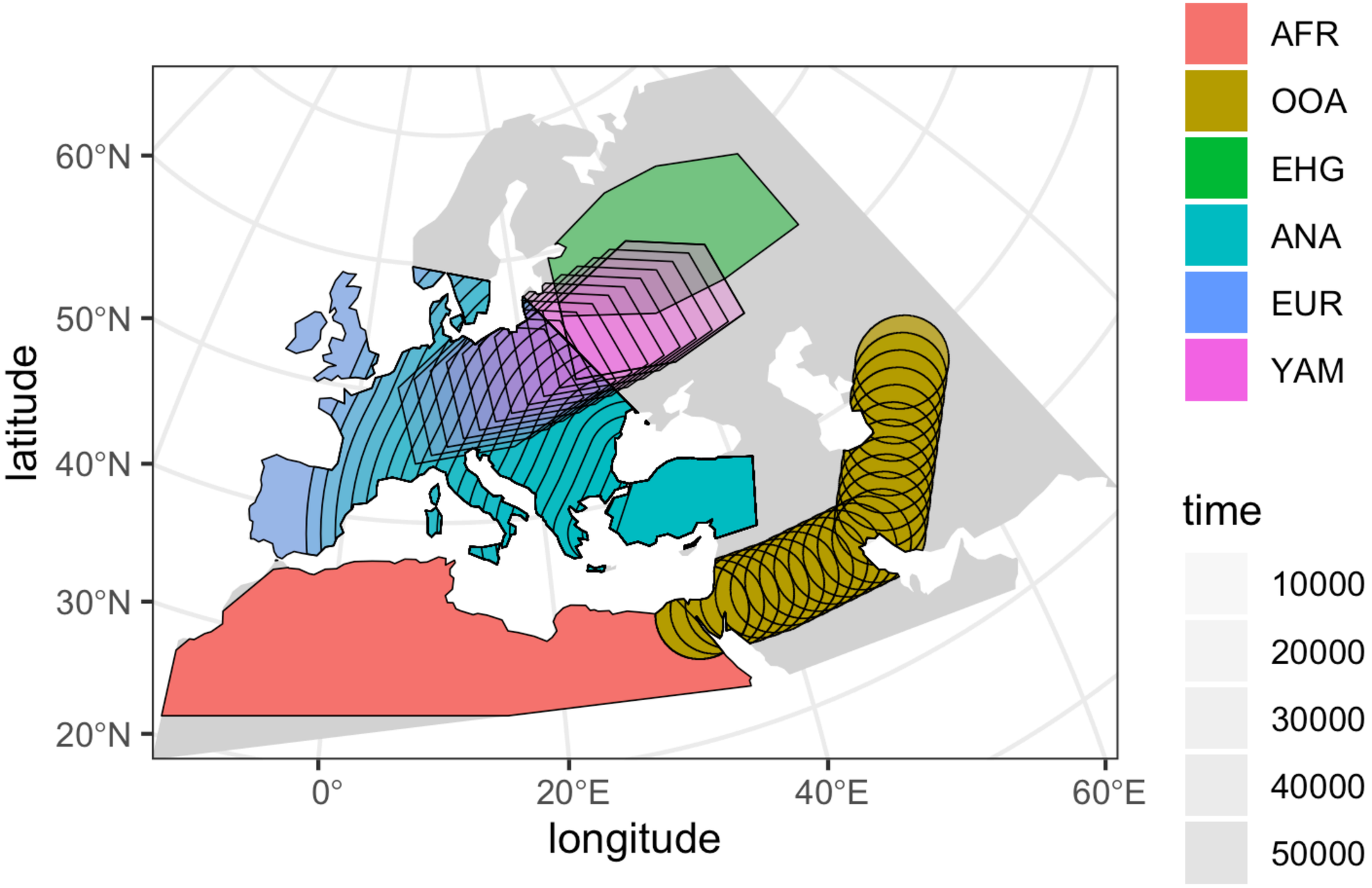
7. Simulation of the model

[slim() or msprime()]

```
ts <- slim(
  model, # model object compiled earlier
  sequence_length = 10e6,     # simulate 10Mb of sequence ...
  recombination_rate = 1e-8  # ... with this uniform recombination rate
)
```

The `slim()` function executes a built-in SLiM script designed for *slendr* demographic models.

The result `ts` is a tree sequence.

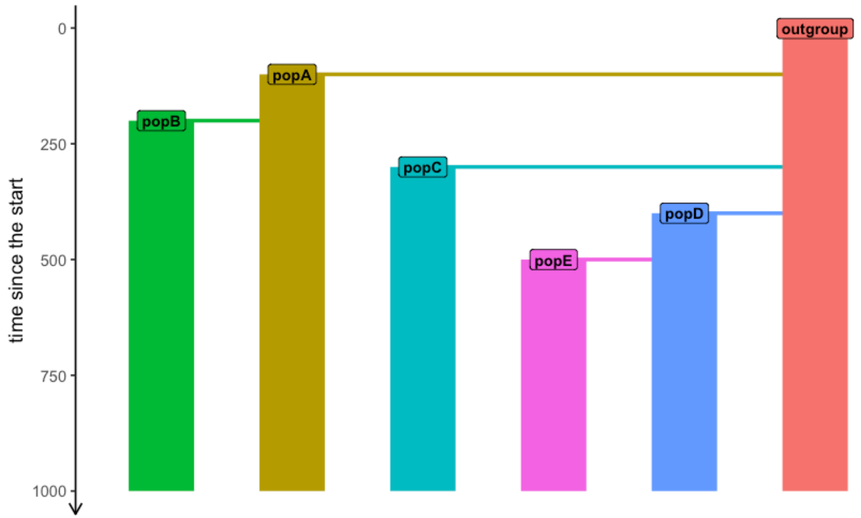

60°N
50°N
40°N
30°N
20°N
latitude
0°
20°E
40°E
60°E
longitude
AFR
OOA
EHG
ANA
EUR
YAM
time
10000
20000
30000
40000
50000

We know the genome and exact location & time of everyone

Generated by a **formally defined spatio-temporal model**

How is a *slendr* `model` executed in the background?

```
outgroup <- population("outgroup", time = 1, N = 1000)
popA <- population("popA", time = 100, N = 1000, parent = outgroup)
popB <- population("popB", time = 200, N = 1000, parent = popA)
popC <- population("popC", time = 300, N = 1000, parent = outgroup)
popD <- population("popD", time = 400, N = 1000, parent = outgroup)
popE <- population("popE", time = 500, N = 1000, parent = popD)

model <- compile_model(
  populations = list(outgroup, popA, popB, popC, popD, popE),
  generation_time = 1, simulation_length = 1000
)
```



(we will focus on non-spatial *slendr* models now, for simplicity)

The most basic SLiM equivalent of this model

```
initialize() {
   initializeMutationRate(0);
   initializeMutationType("m0", 0.5, "f", 0.0);
   initializeGenomicElementType("g1", m0, 1.0);
   initializeGenomicElement(g1, 0, 999999);
   initializeRecombinationRate(1e-8);
}

1 late() {
  sim.addSubpop("p0", 1000);
  p0.name = "outgroup";
}

100 late() {
  sim.addSubpopSplit("p1", 1000, p0);
  p1.name = "popA";
}

200 late() {
  sim.addSubpopSplit("p2", 1000, p1);
  p2.name = "popB";
}

300 late() {
  sim.addSubpopSplit("p3", 1000, p0);
  p3.name = "popC";
}

400 late() {
  sim.addSubpopSplit("p4", 1000, p0);
  p3.name = "popD";
}

500 late() {
  sim.addSubpopSplit("p5", 1000, p4);
  p3.name = "popE";
}

1000 late() {
  sim.simulationFinished();
}
```

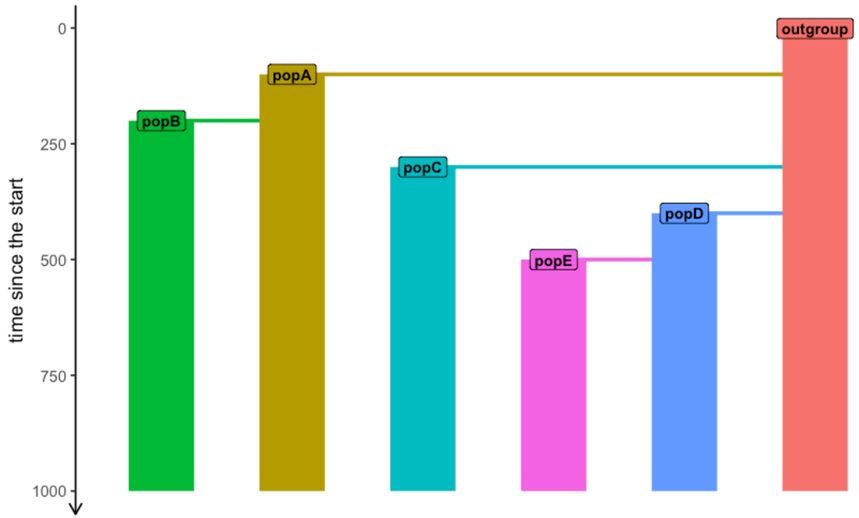
Each demographic event
(here just population splits)
has its own script block

But there are many many other demographic events we might want to simulate:

- population resizes
- gene-flow events
- spatial dynamics (expansions, contractions, displacements, ...)

Each of them would likely need a dedicated script block.

We might even want to parametrize these events.



This can make for a long and complex SLiM script, which is OK if we know exactly what sort of model to simulate — but *slendr* doesn't "know" this.

slendr solves this by turning scripts/models "inside out"

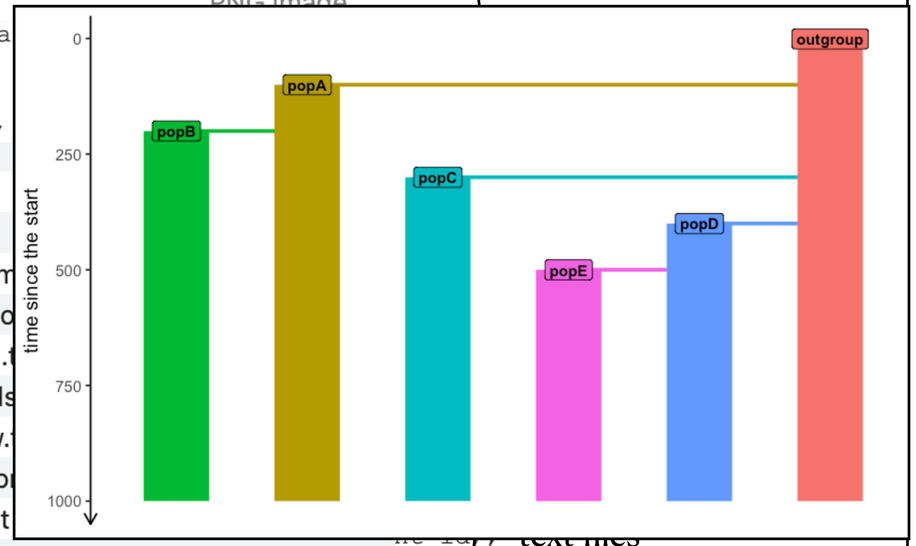
```
initialize() {
   initializeMutationRate(0);
   initializeMutationType("m0", 0.5, "f", 0.0);
   initializeGenomicElementType("g1", m0, 1.0);
   initializeGenomicElement(g1, 0, 999999);
   initializeRecombinationRate(1e-8);
}

1 late() {
  sim.addSubpop("p0", 1000);
  p0.name = "outgroup";
}

100 late() {
  sim.addSubpopSplit("p1", 1000, p0);
  p1.name = "popA";
}

200 late() {
  sim.addSubpopSplit("p2", 1000, p1);
  p2.name = "popB";
}

300 late() {
  sim.addSubpopSplit("p3", 1000, p0);
  p3.name = "popC";
}

400 late() {
  sim.addSubpopSplit("p4", 1000, p0);
  p3.name = "popD";
}

500 late() {
  sim.addSubpopSplit("p5", 1000, p4);
  p3.name = "popE";
}

1000 late() {
  sim.simulationFinished();
}
```

`compile_model()` generates these files on disk:

1.png PNG image
2.png PNG image
3.png PNG image
4.png PNG image
5.png PNG image
58.png
59.png
60.png
checksum
descriptio
direction.t
dispersals
geneflow.t
generatio
length.txt
maps.tsv Plain Text
orig_length.txt Plain Text
populations.tsv Plain Text
ranges.rds File
resolution.txt Plain Text
script.py Python Source
script.slim SLiM model

A simplified equivalent code inside ***slendr***:

```
ir
defineConsta
}
c(1, 100, 200,
}
1(
}
```




```
, 0.0);
, 1.0);
9);
id).name = name;
```

maps.tsv

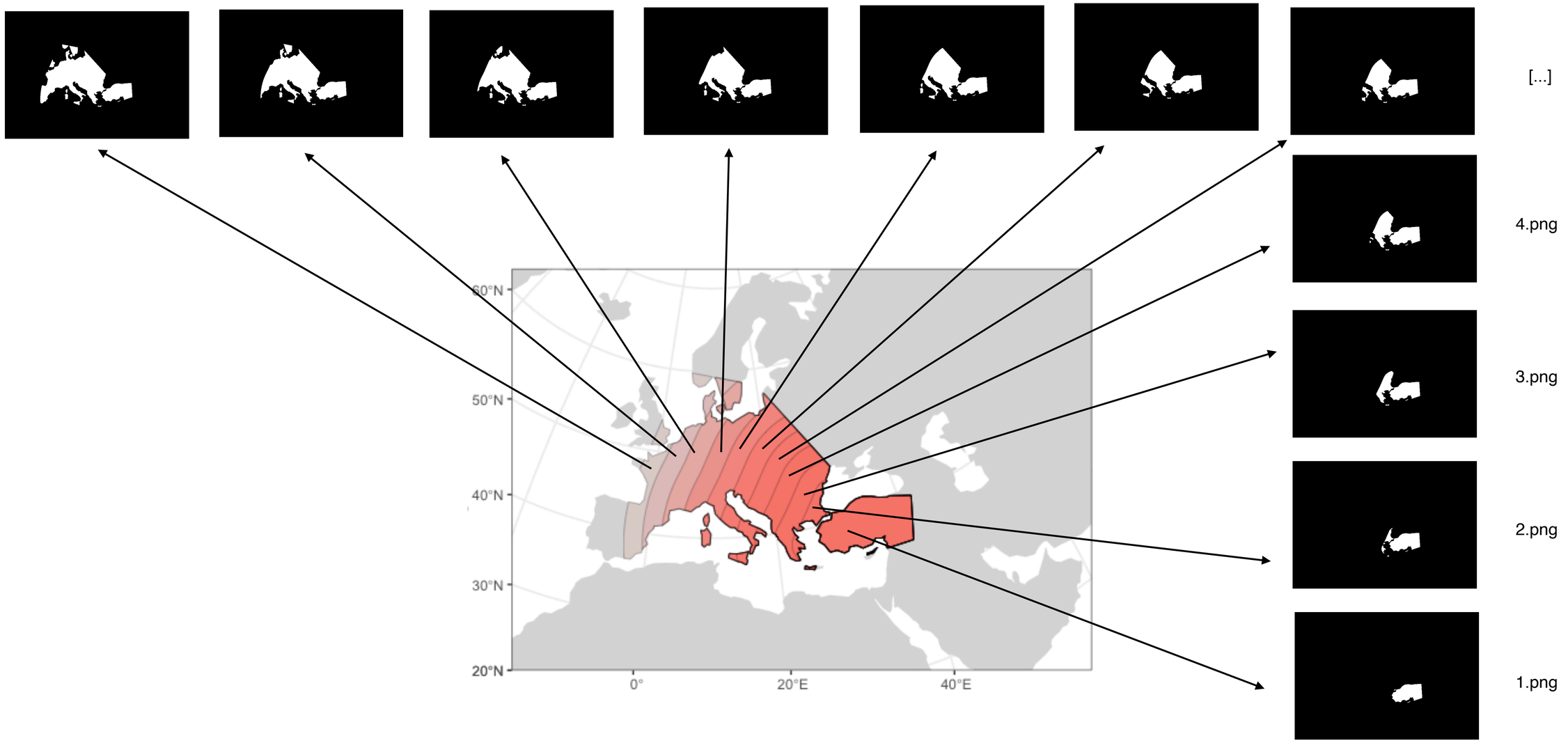
pop	pop_id	time	path
AFR	0	1	1.png
OOA	1	34	2.png
OOA	1	68	3.png
OOA	1	83	4.png
OOA	1	98	5.png
		[...]	
YAM	5	1621	58.png
YAM	5	1628	59.png
YAM	5	1634	60.png

name
outgroup

from	t
3	5
3	4
5	4
<...>	

<subpop>.**defineSpatialMap**("world", "xy", **<XYZ.png>**,

Vector → raster conversion during compilation



■ pixel→ 🛑 □ pixel→ ✓

[...]

4.png

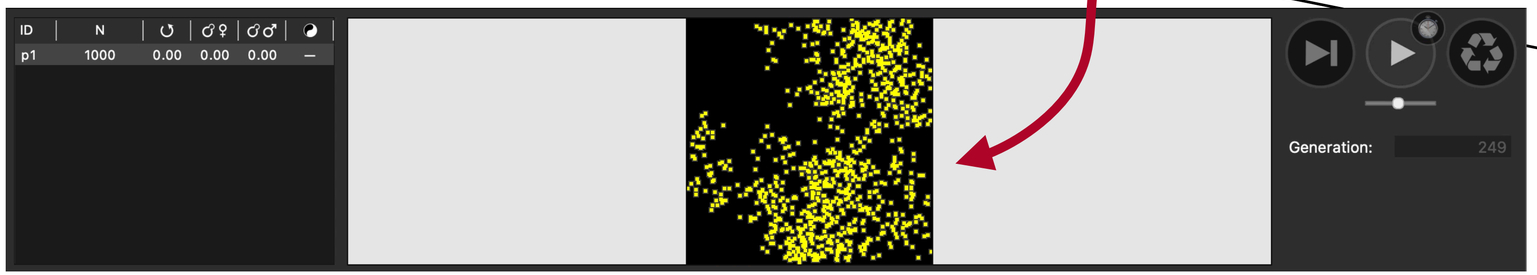
3.png

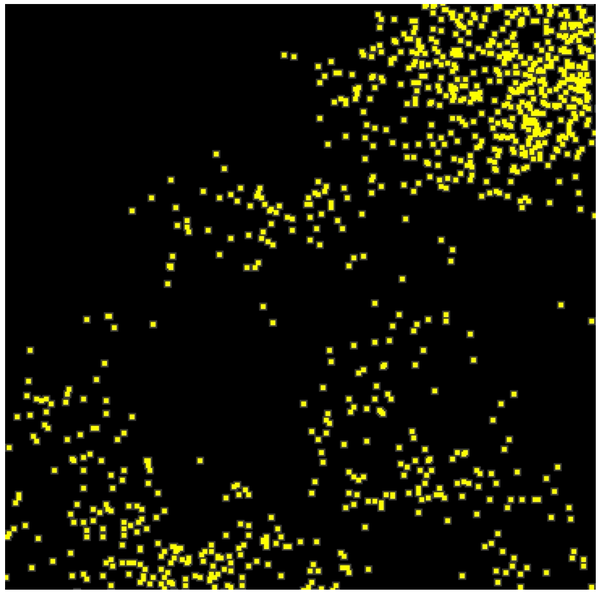
2.png

1.png

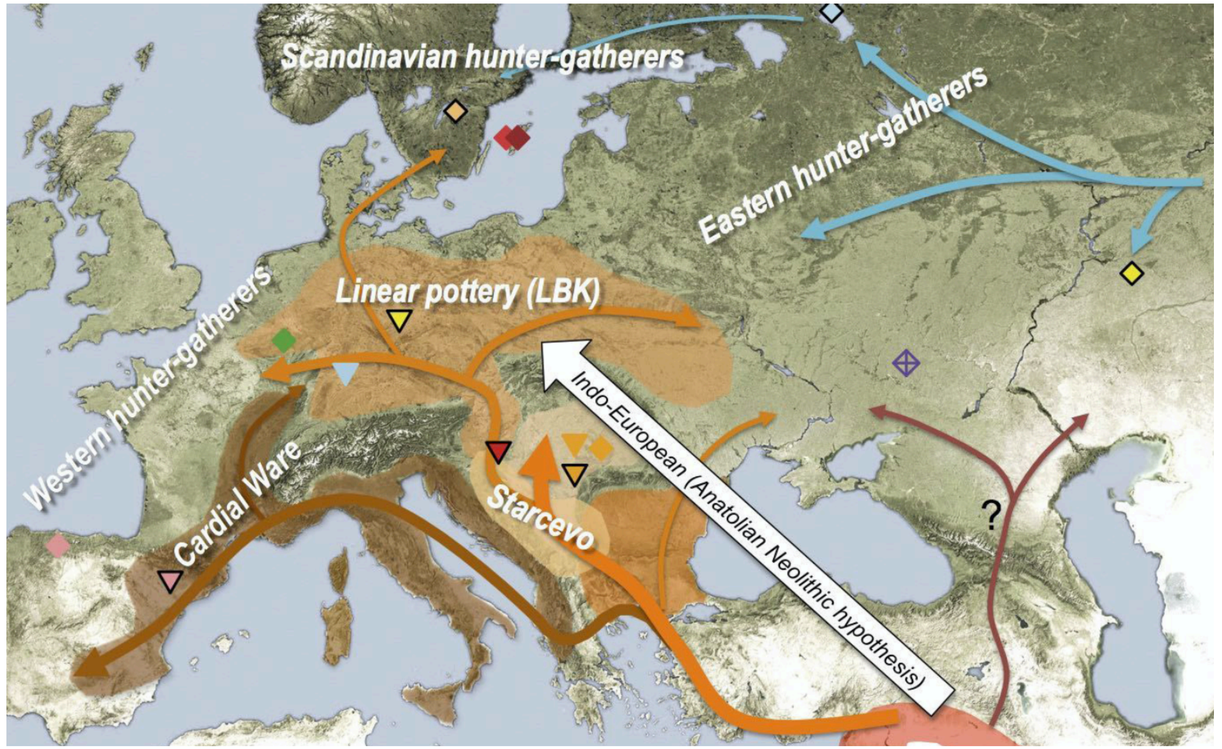
Somewhere deep in the *slendr* / SLiM codebase...

```
<subpop>.defineSpatialMap("world", "xy", <matrix>, <...>)
```



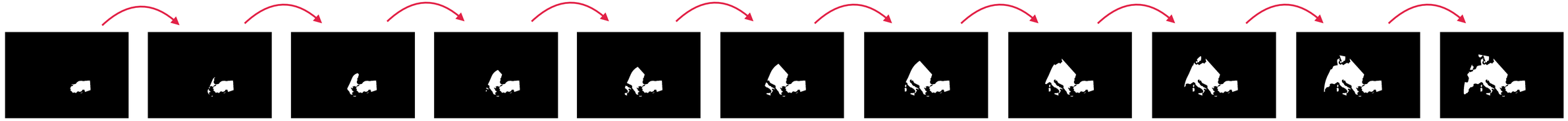


?



Somewhere deep in the *slendr* / SLiM codebase...

```
<subpop>.defineSpatialMap("world", "xy", <matrix>, <...>)
```



Model definition & simulation

Model components

`population()`	Define a population and its original spatial range
`world()`	Define a world map for all spatial operations
`geneflow()`	Define a geneflow event
`region()`	Define a geographic region

Population dynamics

`move()`	Move the population to a new location in a given amount of time
`expand()`	Expand the population range
`shrink()`	Shrink the population range
`boundary()`	Update the population range
`resize()`	Resize the population size
`dispersal()`	Change dispersal parameters

Manipulation of spatial objects

`join()`	Merge two spatial `slendr` objects into one
`overlap()`	Generate the overlap of two `slendr` objects
`subtract()`	Generate the difference between two `slendr` objects
`reproject()`	Reproject coordinates between coordinate systems
`distance()`	Calculate the distance between a pair of spatial boundaries
`area()`	Calculate the area covered by the given slendr object
`dimensions()`	Return the dimensions of the world map

Model visualization and diagnostics

`plot(<slendr>)`	Plot `slendr` geographic features on a map
`plot_graph()`	Plot geneflow graph based on given model configuration
`plot_ancestry()`	Plot simulated ancestry proportions
`animate()`	Animate the simulated population dynamics
`explore()`	Open an interactive browser of the spatial model

Compiling and running spatial models

`compile()`	Compile the spatial demographic model
`sampling()`	Define sampling events at specified times for a given set of populations
`slim()`	Run a slendr model as a SLiM script
`read()`	Read a previously serialized model configuration
`script()`	Substitute variables in a template SLiM script

Tree-sequence analysis

Tree sequence loading and processing

`ts_load()`	Load a tree sequence file produced by a given model
`ts_recapitate()`	Recapitate the tree sequence
`ts_simplify()`	Simplify the tree sequence down to a given set of individuals
`ts_mutate()`	Add mutations to the given tree sequence
`ts_coalesced()`	Check that all trees in the tree sequence are fully coalesced

Extracting genotypes from tree sequences

`ts_genotypes()`	Extract genotype table from the tree sequence
`ts_eigenstrat()`	Extract genotypes from the tree sequence in the EIGENSTRAT format
`ts_vcf()`	Save genotypes from the tree sequenceas a VCF file

Accessing tree sequence components

`ts_data()`	Extract combined annotated table of individuals and nodes
`ts_ancestors()`	Infer spatio-temporal ancestral history for given nodes/individuals
`ts_individuals()` `ts_edges()` `ts_nodes()`	Get the table of individuals/nodes/edges from the tree sequence
`ts_samples()`	Extract names and times of individuals scheduled for sampling
`ts_tree()`	Get a tree from a given tree sequence
`ts_draw()`	Plot a graphical representation of a single tree

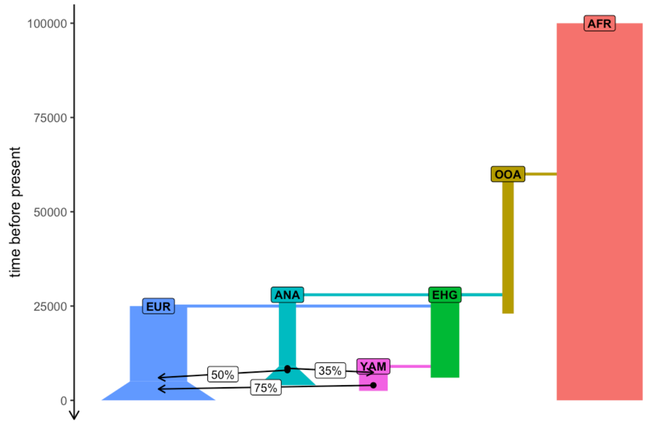
Tree sequence statistics

`ts_f2()` `ts_f3()` `ts_f4()` `ts_f4ratio()`	Calculate the f2, f3, f4, and f4-ratio statistics
`ts_afs()`	Compute the allele frequency spectrum (AFS)
`ts_fst()` `ts_divergence()`	Calculate pairwise statistics between sets of individuals
`ts_diversity()`	Calculate diversity in given sets of individuals
`ts_tajima()`	Calculate Tajima's D for given sets of individuals
`ts_segregating()`	Calculate the density of segregating sites for the given sets of individuals

slendr's tree-sequence analysis library

(all internally backed by ***tskit*** and ***pyslim***)

```
library(slendr); init_env()

afr <- population("AFR", time = 100000, N = 15000)
ooa <- population("OOA", parent = afr, time = 60000, N = 2000)
ehg <- population("EHG", parent = ooa, time = 28000, N = 5000)
eur <- population("EUR", parent = ehg, time = 25000, N = 10000) %>%
  resize(N = 20000, how = "exponential", time = 5000, end = 0)
ana <- population("ANA", time = 28000, N = 3000, parent = ooa) %>%
  resize(N = 10000, how = "exponential", time = 9000, end = 4000)
yam <- population("YAM", time = 9000, N = 5000, parent = ehg)

gf <- list(
  gene_flow(from = ana, to = yam, rate = 0.35, start = 8500, end = 7400),
  gene_flow(from = ana, to = eur, rate = 0.5, start = 8000, end = 6000),
  gene_flow(from = yam, to = eur, rate = 0.75, start = 4000, end = 3000)
)

model <- compile_model(
  populations = list(afr, ooa, ehg, eur, ana, yam),
  gene_flow = gf, generation_time = 30
)

ts <- slim(model, sequence_length = 50e6, recombination_rate = 1e-8)
```



Recapitate the simulated tree sequence and overlay mutations:

```
ts <-
  ts %>%
  ts_recapitate(Ne = 15000, recombination_rate = 1e-8) %>%
  ts_mutate(mutation_rate = 1e-8)
```

... do the same but also simplify to only a few defined individuals:

```
ts <-
  ts %>%
  ts_recapitate(Ne = 15000, recombination_rate = 1e-8) %>%
  ts_mutate(mutation_rate = 1e-8) %>%
  ts_simplify(c("AFR_1", "OOA_1", "EHG_1", "ANA_1", "EUR_1", "YAM_1"))
```

Compute an *f3* admixture test:

```
> ts_f3(ts, A = "EUR_1", B = "EHG_1", C = "ANA_1")
# A tibble: 1 × 4
  A     B     C              f3
  <chr> <chr> <chr>       <dbl>
1 EUR_1 EHG_1 ANA_1 -0.0000104
```

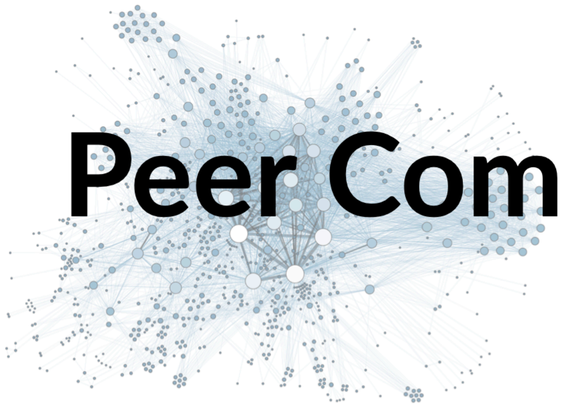
Compute allele-frequency spectrum across all individuals:

```
> ts_afs(ts)
 [1]     0 25718 10869  8127  6638  5529  4678  4070  3749  3356  3335  3519
```

F_{st} between AFR and EUR populations:

```
> ts_fst(ts, list(afr = "AFR_1",
+                 eur = c("EHG_1", "ANA_1", "EUR_1")))
# A tibble: 1 × 3
  x     y         Fst
  <chr> <chr>   <dbl>
1 afr   eur   0.0764
```

www.slendr.net



Peer Community Journal

Section: Evolutionary Biology

RESEARCH ARTICLE

2023

slendr: a framework for spatio-temporal population genomic simulations on geographic landscapes

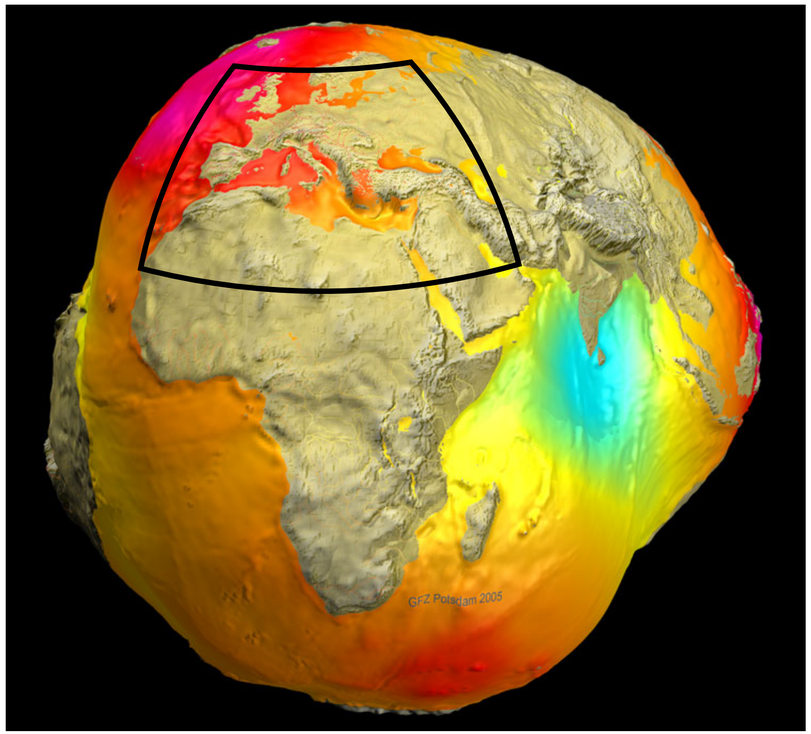
Martin Petr[1,2], Benjamin C. Haller[3], Peter L. Ralph[4], and Fernando Racimo[1,2]

Contact: mp@bodkan.net

Slides will be at: `https://bodkan.net/talks`

Why an explicit *Coordinate Reference System* (CRS)?

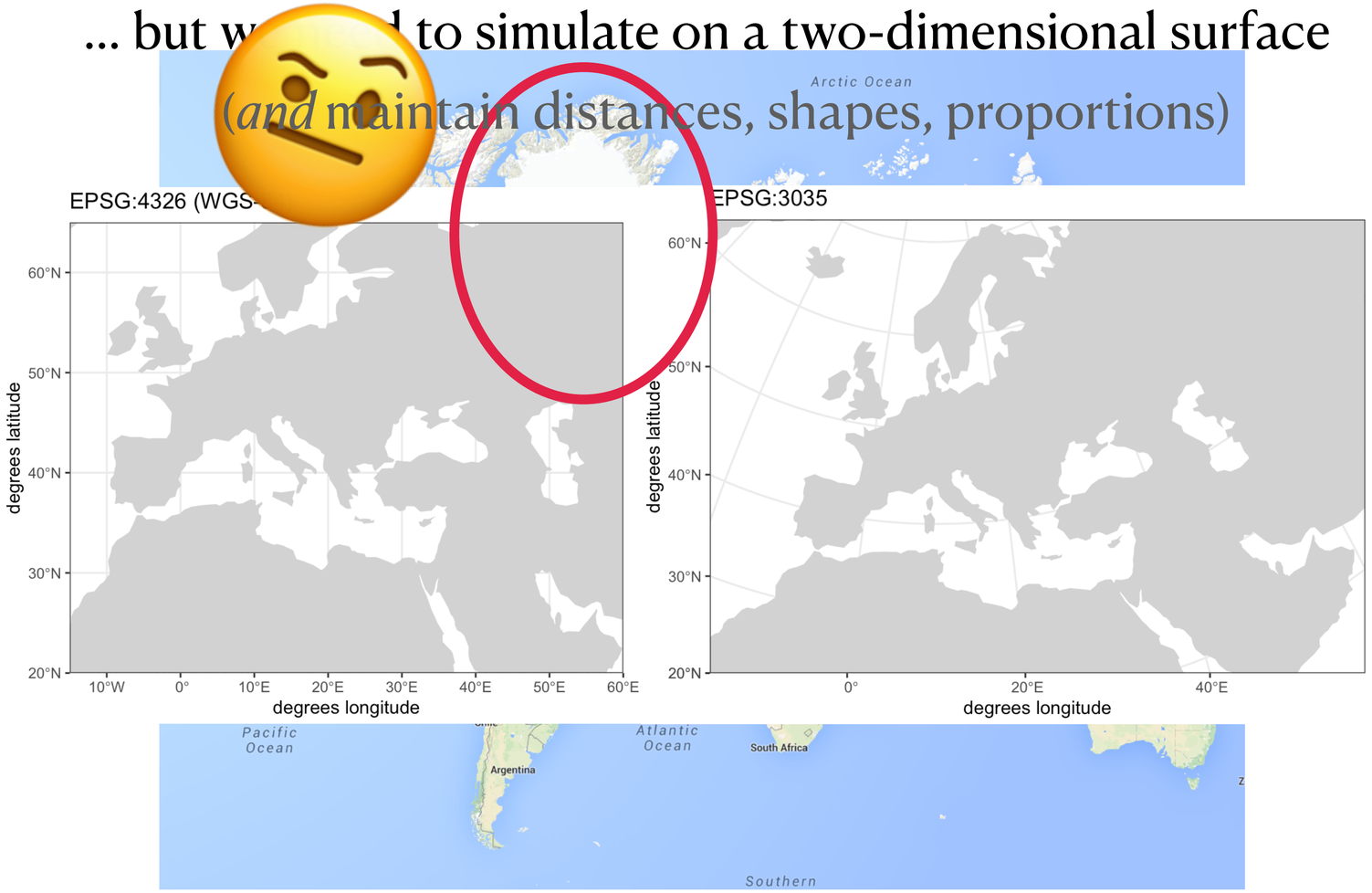
Earth is a ~~potato~~ ~~sphere~~ ellipsoid...



Astronomy Picture of the Day: 2014 December 15

... but w[illegible]d to simulate on a two-dimensional surface

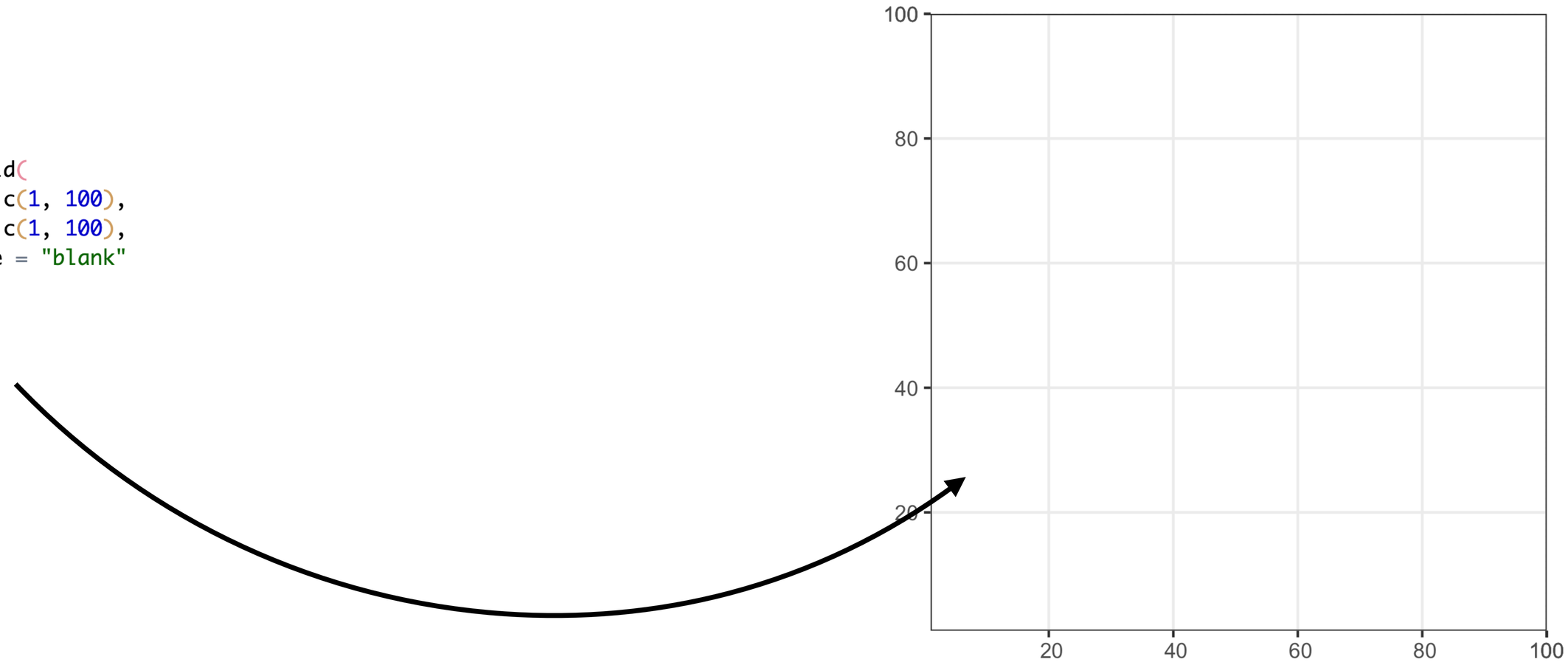
(*and* maintain distances, shapes, proportions)



1. Define a spatial context ("empty `world()`")

(section of the world which will be occupied by populations)

```
map <- world(
  xrange = c(1, 100),
  yrange = c(1, 100),
  landscape = "blank"
)
```



1. Define a spatial context ("custom `world()`")

(section of the world which will be occupied by populations)

```
# three island regions...
island1 <- region(center = c(70, 50), radius = 15)
island2 <- region(center = c(40, 75), radius = 10)
island3 <- region(polygon = list(c(20, 20), c(40, 15), c(50, 20),
                                 c(50, 35), c(40, 40), c(20, 35)))
# ... combined in a single map
map <- world(
  xrange = c(1, 100),
  yrange = c(1, 100),
  landscape = do.call(rbind, list(island1, island2, island3))
)
```