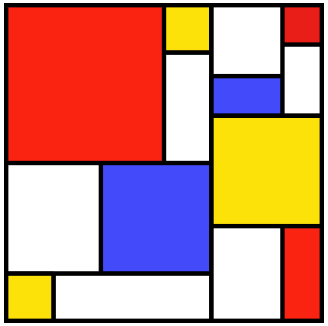
SLiM

Workshop Series



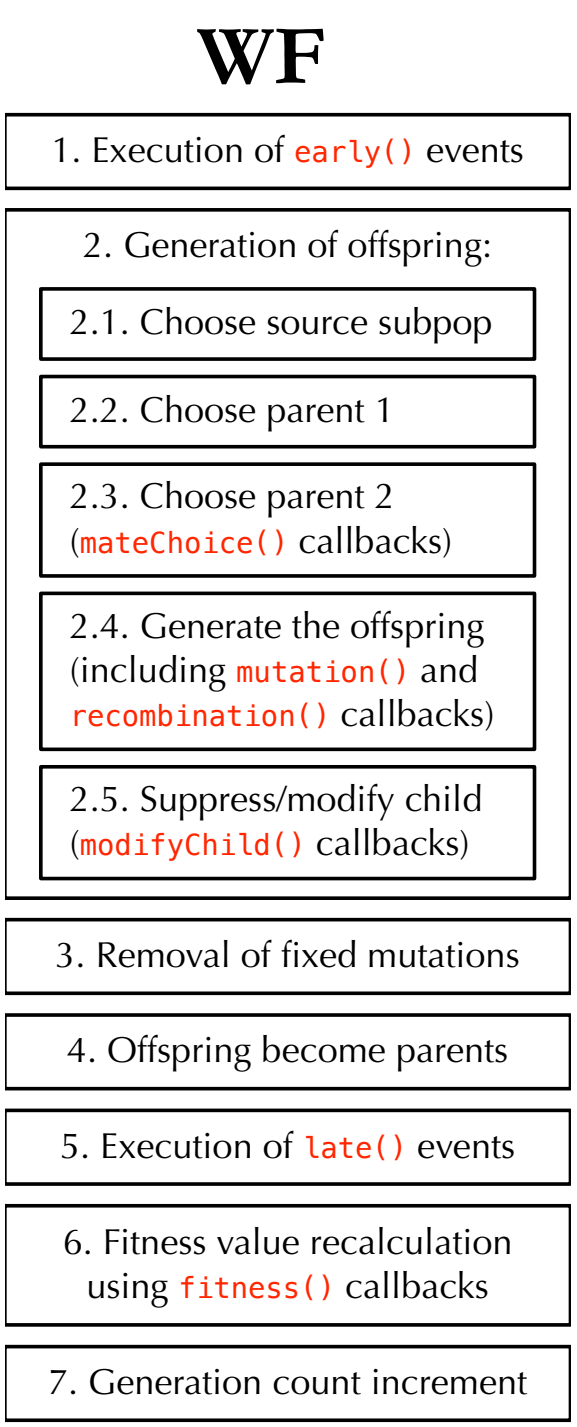
#15: non-Wright–Fisher models

Benjamin C. Haller

Messer Lab, Cornell University

WF versus nonWF

- SLiM supports two different model types:
  - **WF**: Wright–Fisher
  - **nonWF**: non-Wright–Fisher
- Differences:
  - generation cycles
  - offspring generation
  - population regulation
  - age structure
  - fitness models
  - demography
  - mate choice
  - migration

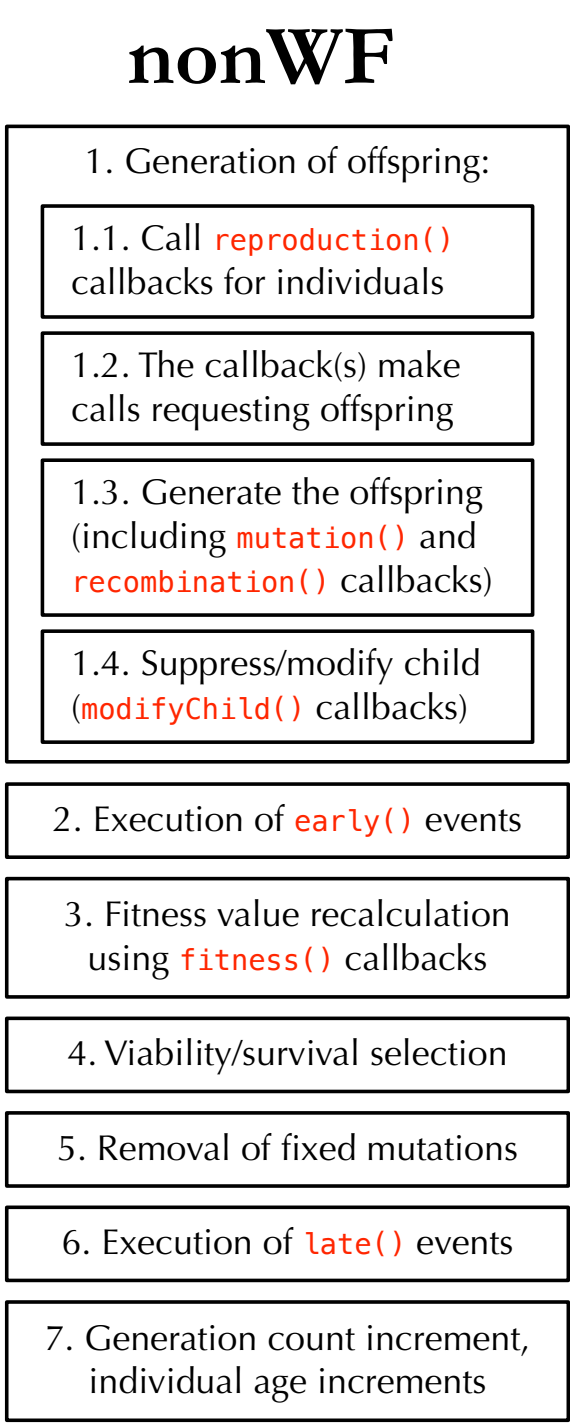


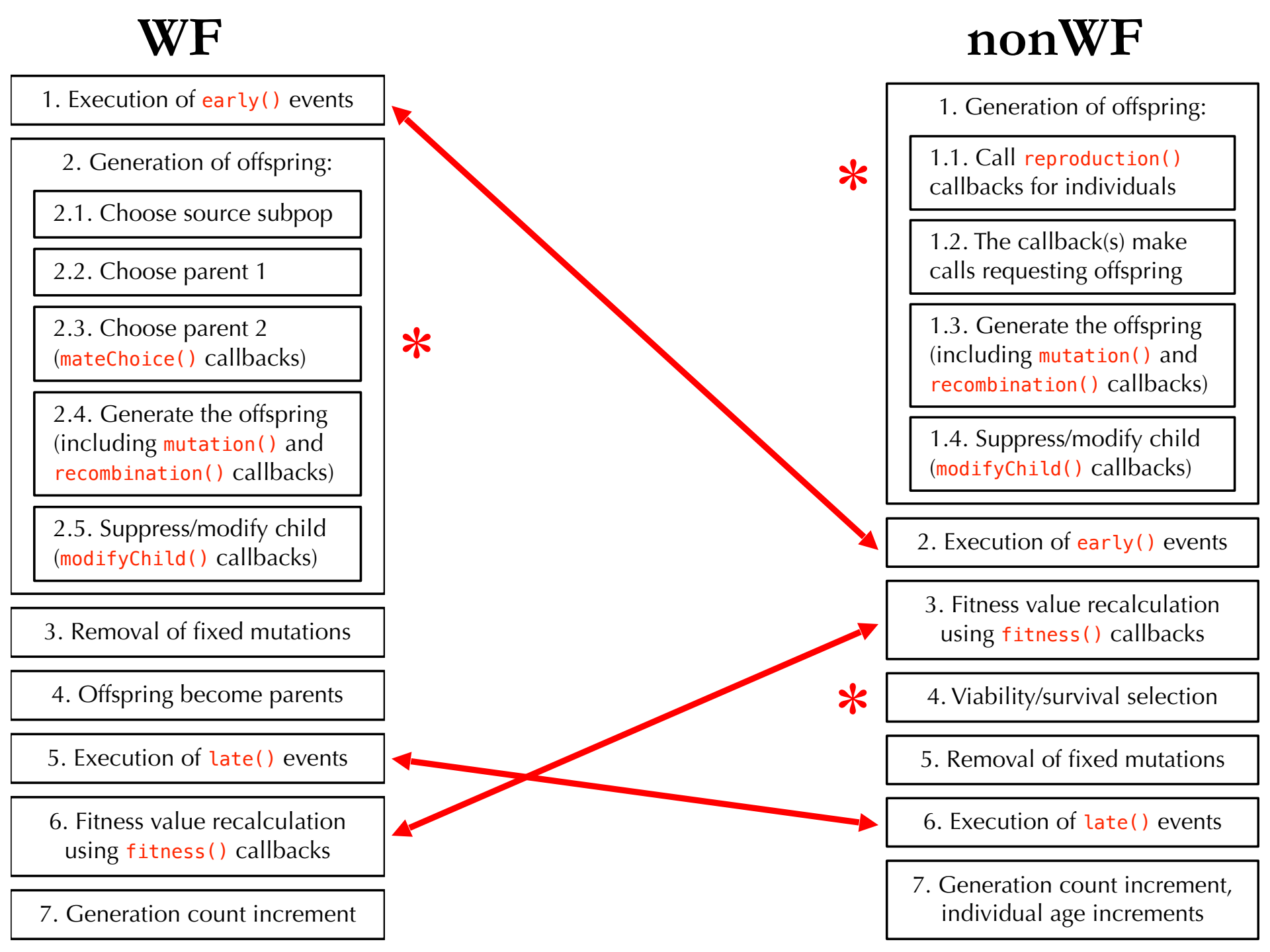
WF Models

- Population size is a parameter
- Population regulation is automatic
- Fitness affects mating probability
- Selection is soft (relative fitness)
- Non-overlapping generations
- No age structure
- Migration is due to parameters

nonWF Models

- Population size is emergent
- Population regulation is scripted
- Fitness affects viability/survival
- Selection is hard (absolute fitness)
- Generations can overlap
- Age structure is emergent
- Migration is scripted




WF
1. Execution of early() events
2. Generation of offspring:
2.1. Choose source subpop
2.2. Choose parent 1
2.3. Choose parent 2 (mateChoice() callbacks)
2.4. Generate the offspring (including mutation() and recombination() callbacks)
2.5. Suppress/modify child (modifyChild() callbacks)
3. Removal of fixed mutations
4. Offspring become parents
5. Execution of late() events
6. Fitness value recalculation using fitness() callbacks
7. Generation count increment
nonWF
1. Generation of offspring:
1.1. Call reproduction() callbacks for individuals
1.2. The callback(s) make calls requesting offspring
1.3. Generate the offspring (including mutation() and recombination() callbacks)
1.4. Suppress/modify child (modifyChild() callbacks)
2. Execution of early() events
3. Fitness value recalculation using fitness() callbacks
4. Viability/survival selection
5. Removal of fixed mutations
6. Execution of late() events
7. Generation count increment, individual age increments

nonWF Models

- Need to declare the model as nonWF
  - `initializeSLiMModelType("nonWF")`
- Need to consider substitution explicitly
  - `convertToSubstitution` defaults to `F`
- Need a `reproduction()` callback
  - generates offspring for one *focal individual*
  - does so by calling `addCrossed()` etc.
- Need to impose population regulation
  - typically density-dependence in `early()`

The `initialize()` callback

```
// set up a simple neutral nonWF simulation
initialize() {
    initializeSLiMModelType("nonWF");
    defineConstant("K", 500); // carrying capacity

    // neutral mutations, which are allowed to fix
    initializeMutationType("m1", 0.5, "f", 0.0);
    m1.convertToSubstitution = T;

    initializeGenomicElementType("g1", m1, 1.0);
    initializeGenomicElement(g1, 0, 99999);
    initializeMutationRate(1e-7);
    initializeRecombinationRate(1e-8);
}
```

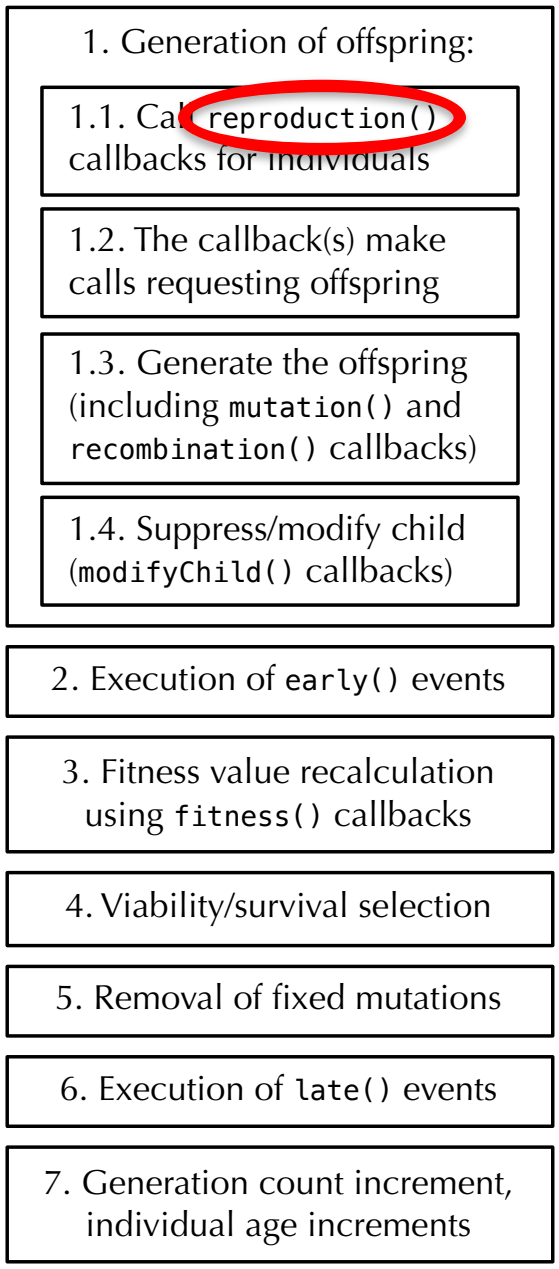
- Calls `initializeSLiMModelType("nonWF")`
- Defines a carrying capacity, `K`
- Sets `convertToSubstitution=T` for `m1`

The `reproduction()` callback

```
// each individual reproduces itself once
reproduction() {
    subpop.addCrossed(individual, p1.sampleIndividuals(1));
}
```

- Required in nonWF models
- Called during offspring generation
- Called once for each *focal individual*
- Does all reproduction for that individual
- Replaces the `mateChoice()` callback
- Partly replaces `modifyChild()`

The sequence of events within one generation in nonWF models.



`reproduction()` callbacks

```
reproduction([<subpopID>[, <sex>]])
{
   ...
}
```

- apply to *one or all* subpops
- apply to *one or all* sexes
- **pseudo-parameters:**
  - `individual`
  - `genome1`
  - `genome2`
  - `subpop`
- no return value

Population regulation

```
// provide density-dependent selection
early() {
    p1.fitnessScaling = K / p1.individualCount;
}
```

- There is no set population size
- Population size is regulated by **mortality**
- Mortality depends upon **fitness**
- Various mechanisms can operate:
  - resource availability
  - predation
  - territorial behavior
  - natural disasters

A complete nonWF model

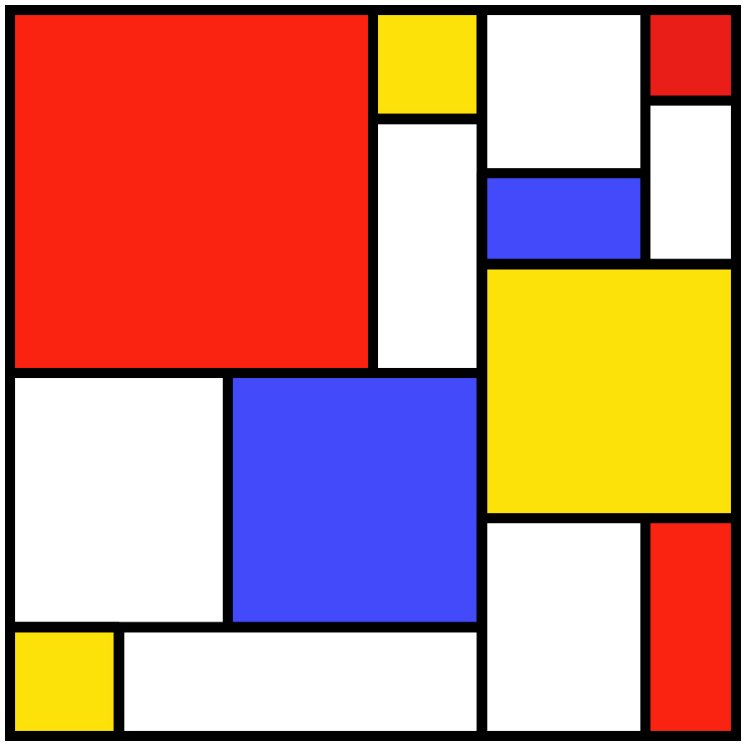
```
// set up a simple neutral nonWF simulation
initialize() {
	initializeSLiMModelType("nonWF");
	defineConstant("K", 500);  // carrying capacity

	// neutral mutations, which are allowed to fix
	initializeMutationType("m1", 0.5, "f", 0.0);
	m1.convertToSubstitution = T;

	initializeGenomicElementType("g1", m1, 1.0);
	initializeGenomicElement(g1, 0, 99999);
	initializeMutationRate(1e-7);
	initializeRecombinationRate(1e-8);
}

// each individual reproduces itself once
reproduction() {
	subpop.addCrossed(individual, p1.sampleIndividuals(1));
}

// create an initial population of 10 individuals
1 early() {
	sim.addSubpop("p1", 10);
}

// provide density-dependent selection
early() {
	p1.fitnessScaling = K / p1.individualCount;
}

2000 late() { sim.outputFixedMutations(); }
```

Fitness in nonWF models

- In WF models:
  - fitness is *relative*, selection is *soft*
  - fitness influences **mating success**

Fitness in nonWF models

- In WF models:
  - fitness is *relative*, selection is *soft*
  - fitness influences **mating success**
- In nonWF models:
  - fitness is *absolute*, selection is *hard*
  - fitness influences **mortality**
  - survival probability is capped at 100%
  - beneficial mutations behave unexpectedly!



SLiM Workshop Exercise #15