# Forward-time simulations using simuPOP, an in-depth course

Bo Peng, Ph.D.

Department of Epidemiology
UT MD Anderson Cancer Center
Houston, TX

June 15th, 2007
simuPOP workshop
School of Public Health, Department of Biostatistics
University of Alabama Birmingham

# outline

# Outline



## 1 The global view

- Population
- Individual
- Operator
- Mating scheme
- Simulator
- Other utilities

# six types of simuPOP modules



1. Possible allele states:

| | |
|---|---|
| **short** | $0 \sim 2^8 - 1$ |
| **long** | $0 \sim 2^{16} - 1$ |
| **binary** | 0 and 1 |

2. Debug information

| | |
|---|---|
| **standard** | with debug information and runtime check |
| **optimized** | without debug information and runtime check |

# Loading simuPOP



## You can specify which module to load by

1. Use `simuOpt.setOptions`

```
>>> from simuOpt import setOptions
>>> setOptions(alleleType='long', optimized=False)
>>> from simuPOP import *
simuPOP : Copyright (c) 2004-2006 Bo Peng
Development Version (May 21 2007) for Python 2.3.4
[GCC 3.4.6 20060404 (Red Hat 3.4.6-3)]
Random Number Generator is set to mt19937 with random seed 0x224a5b0a187b5c00
This is the standard long allele version with 65536 maximum allelic states.
For more information, please visit http://simupop.sourceforge.net,
or email simupop-list@lists.sourceforge.net (subscription required).
>>>
```

2. Set environment variables
   - `SIMUALLELETYPE = short/long/binary`
   - `SIMUOPTIMIZED` for optimized version
3. Command line argument `--optimized`

# Important simuPOP concepts



**genotypic structure** number of chromosome, number and position of loci on each chromosome etc.

# Important simuPOP concepts



**genotypic structure** number of chromosome, number and position of loci on each chromosome etc.

**individual** building blocks of populations

# Important simuPOP concepts



**genotypic structure** number of chromosome, number and position of loci on each chromosome etc.

**individual** building blocks of populations

**population** consist of individuals of the same genotypic structure

# Important simuPOP concepts



**genotypic structure** number of chromosome, number and position of loci on each chromosome etc.

**individual** building blocks of populations

**population** consist of individuals of the same genotypic structure

**operator** objects that operate on populations

# Important simuPOP concepts



**genotypic structure** number of chromosome, number and position of loci on each chromosome etc.

**individual** building blocks of populations

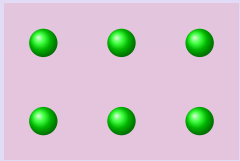
**population** consist of individuals of the same genotypic structure

**operator** objects that operate on populations

**mating scheme** controls how offspring generation is populated

# Illustration of the evolutionary process

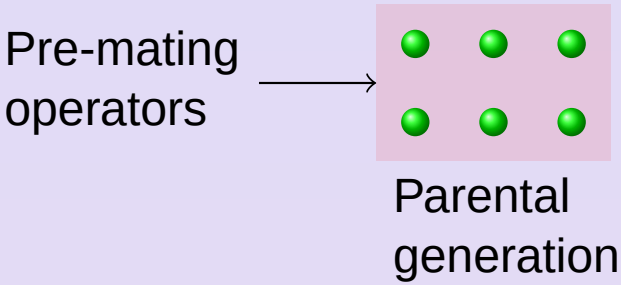




Parental generation

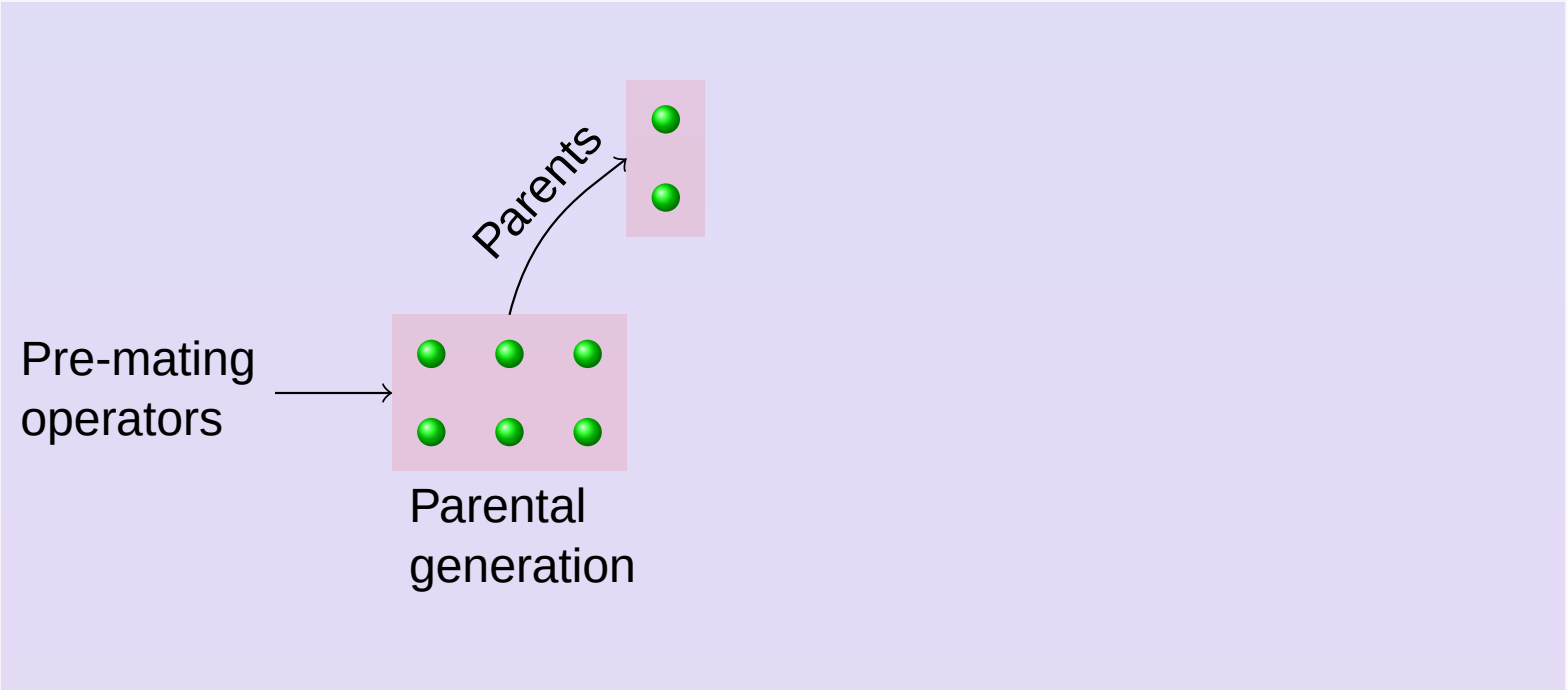
# Illustration of the evolutionary process





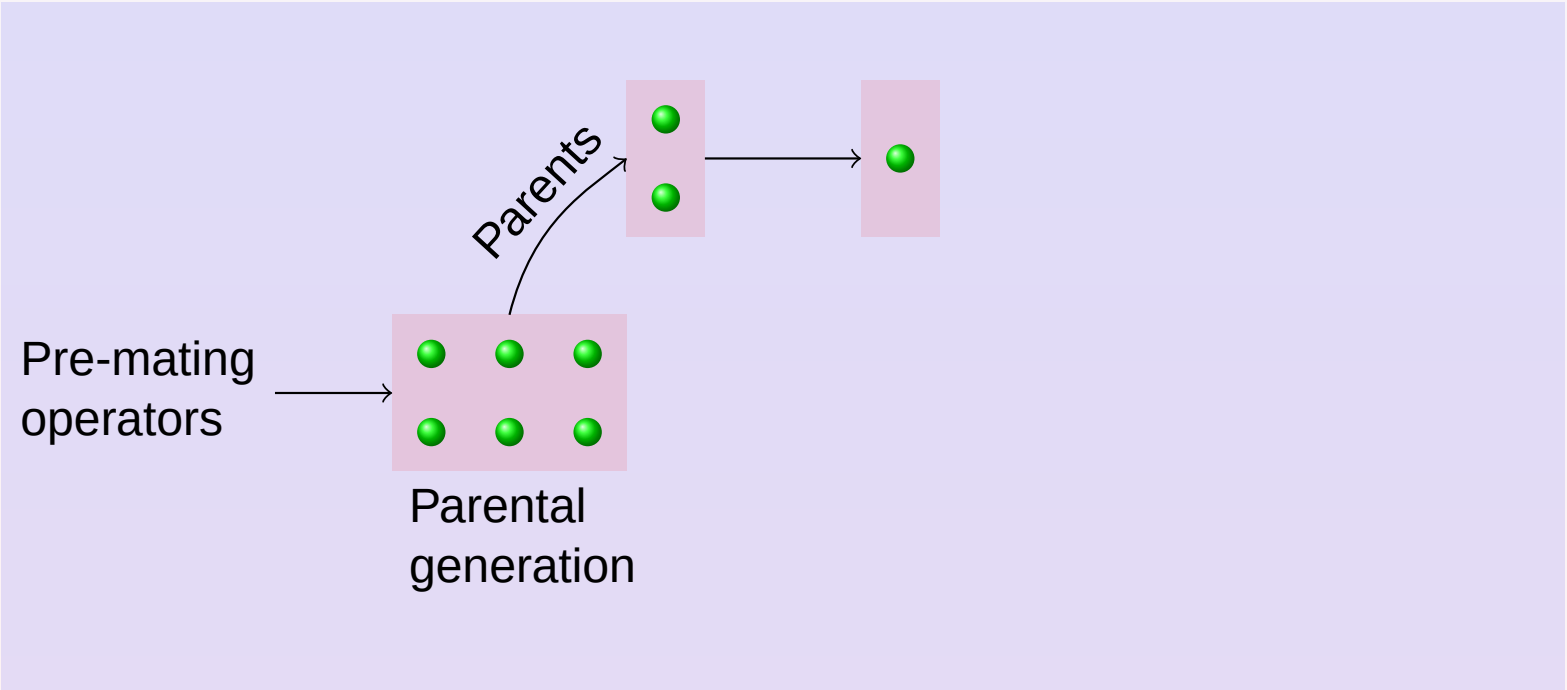
# Illustration of the evolutionary process





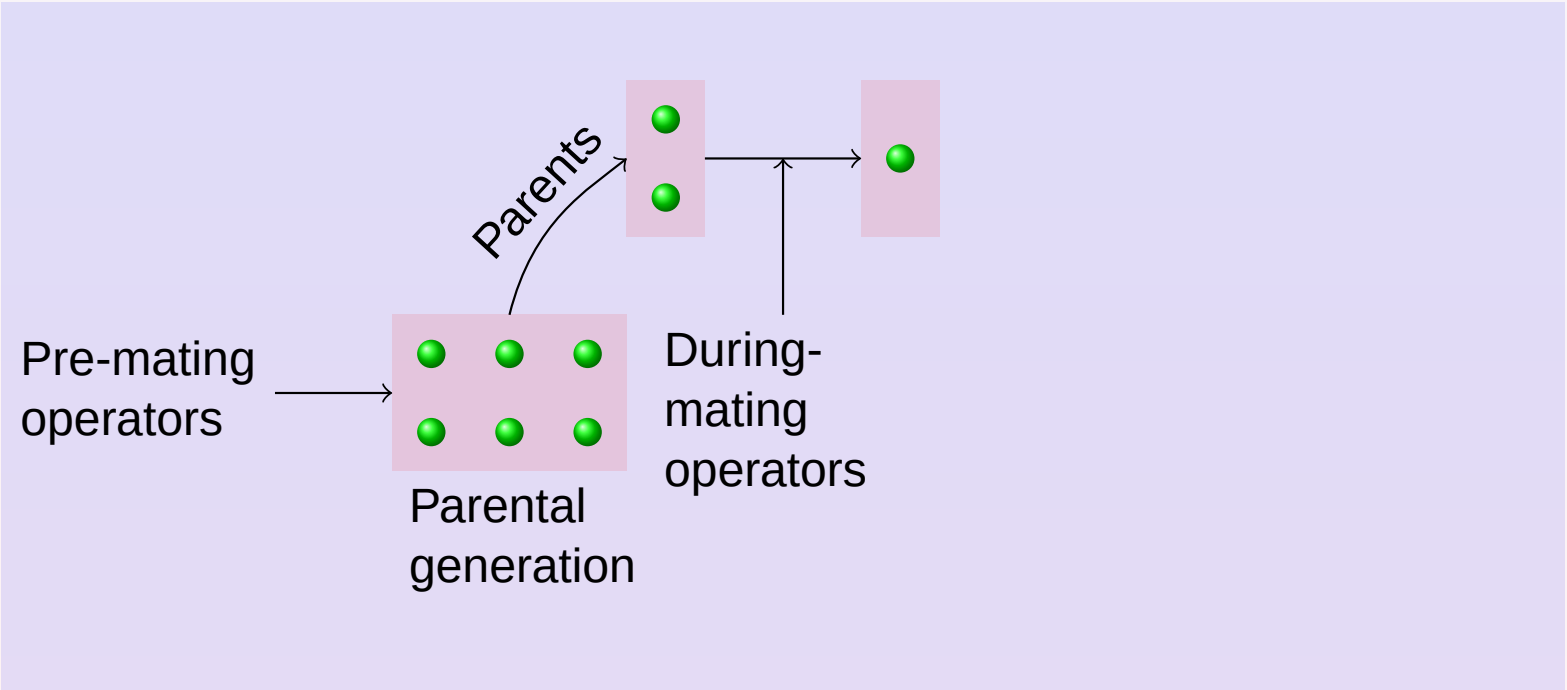
# Illustration of the evolutionary process





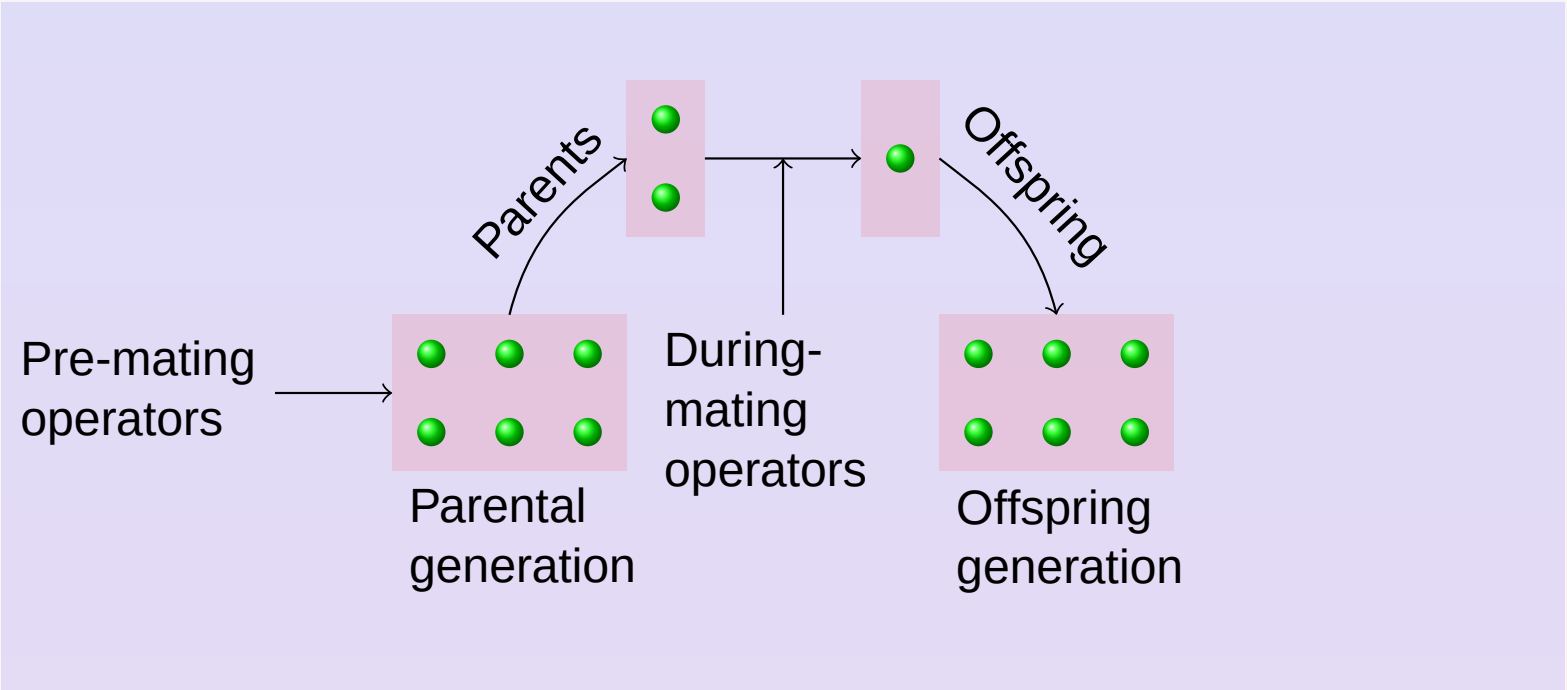
# Illustration of the evolutionary process





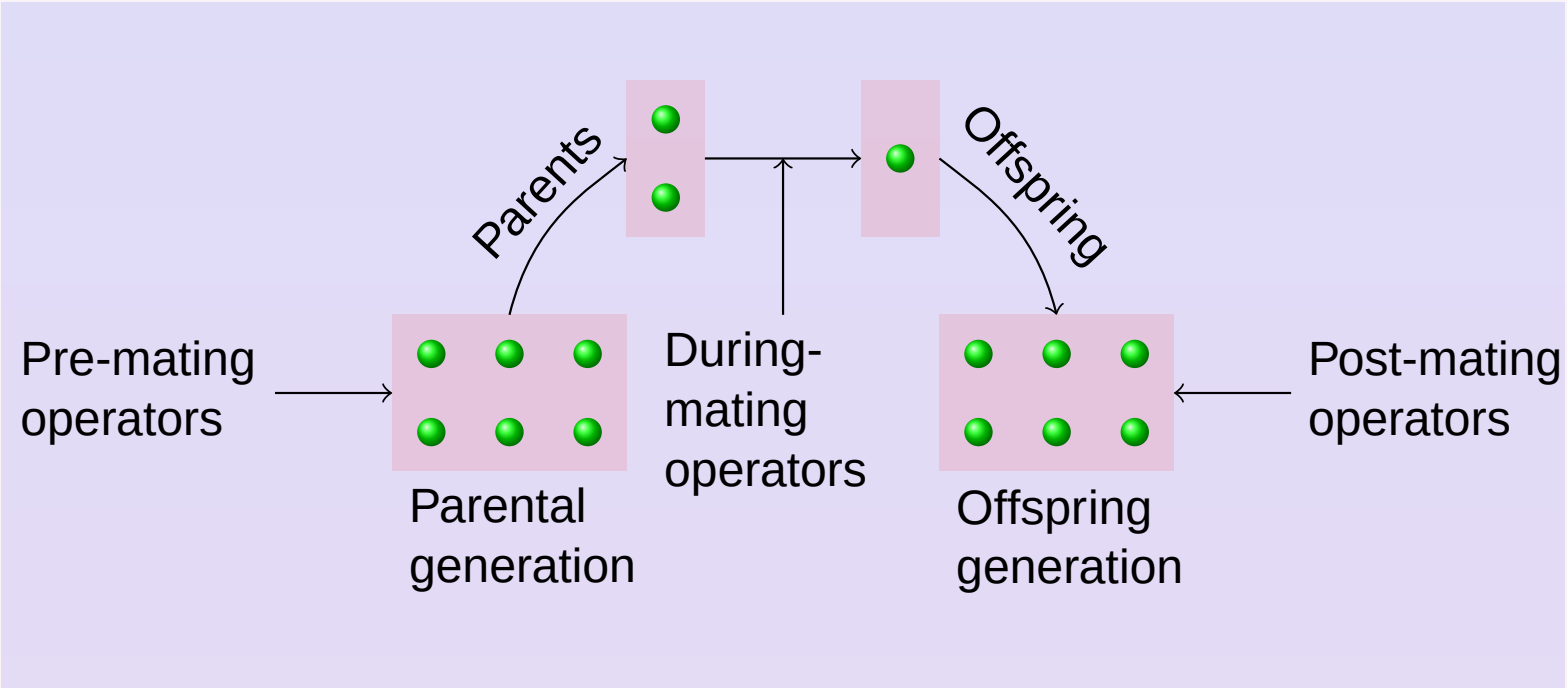
# Illustration of the evolutionary process





# Illustration of the evolutionary process



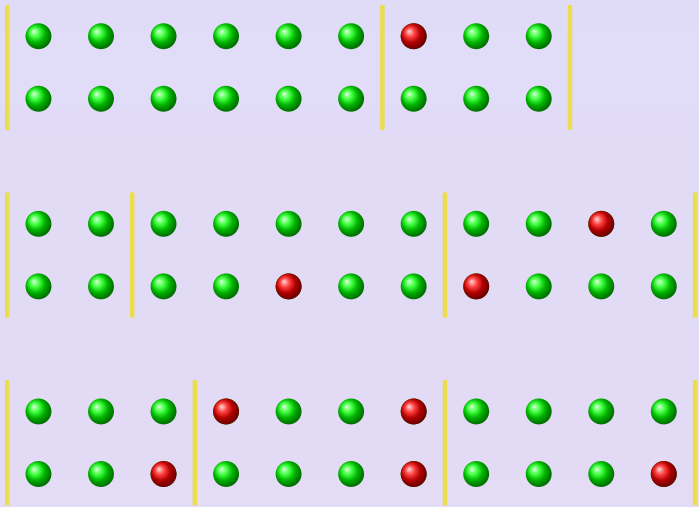


- Allele type: short, long, binary
- Standard and Optimized
- MPI (parallel) version, not ready

# Structure of a population



- Unaffected
- Affected

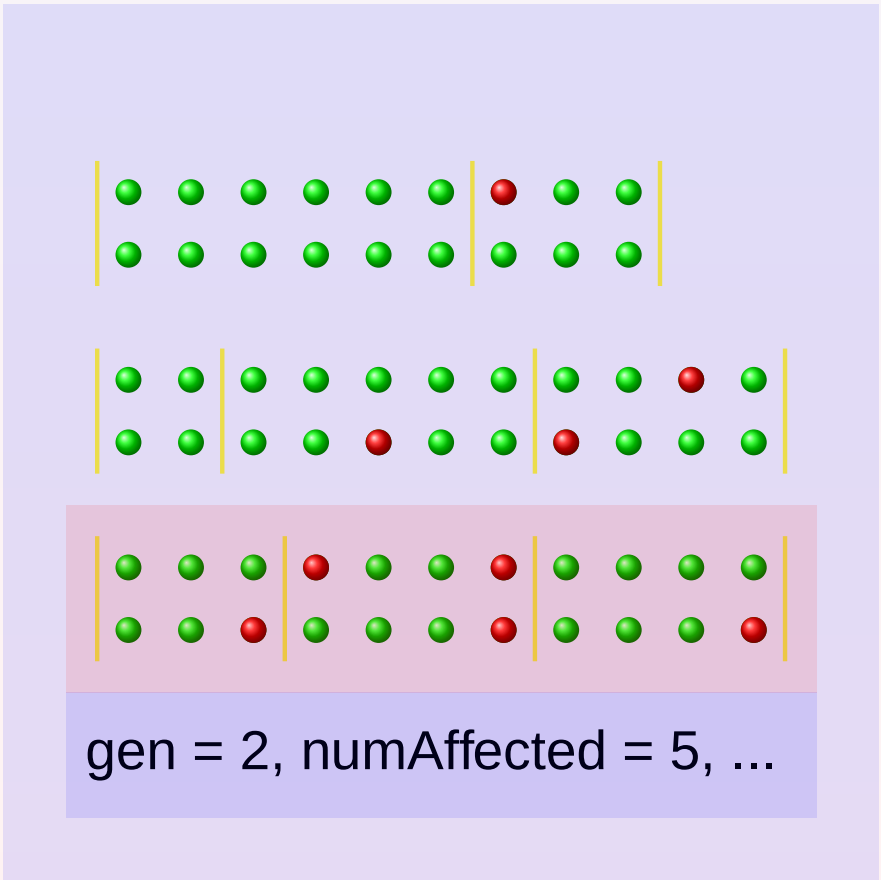


gen = 2, numAffected = 5, ...

# Structure of a population



- Unaffected
- Affected

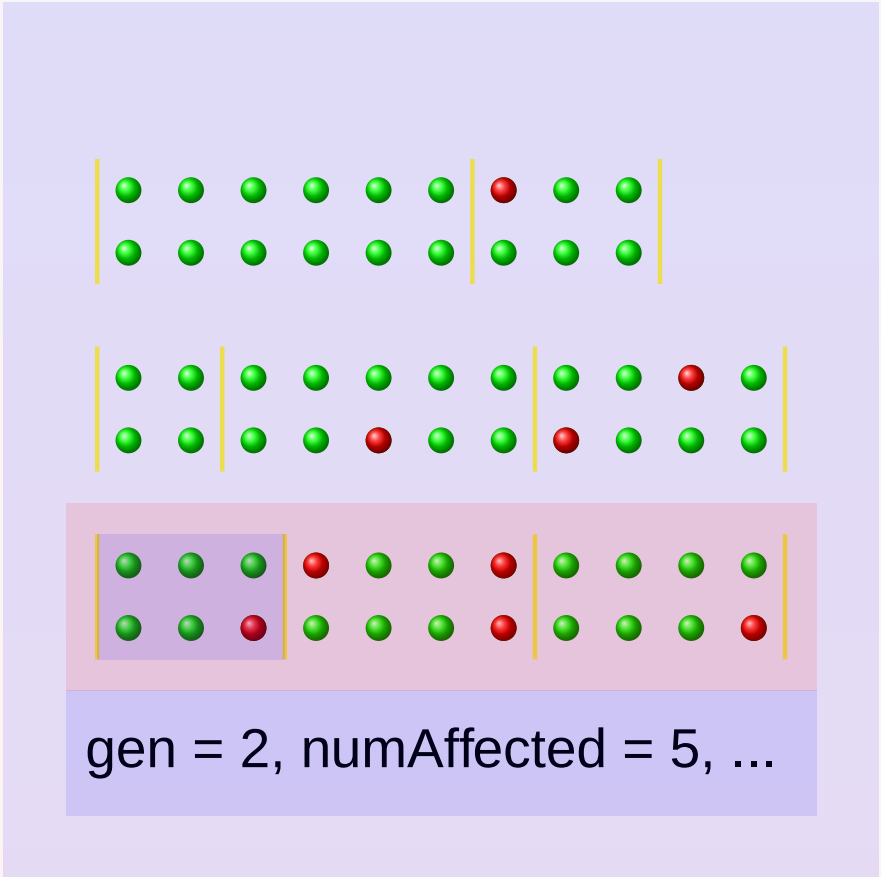



Current generation

# Structure of a population



- Unaffected
- Affected

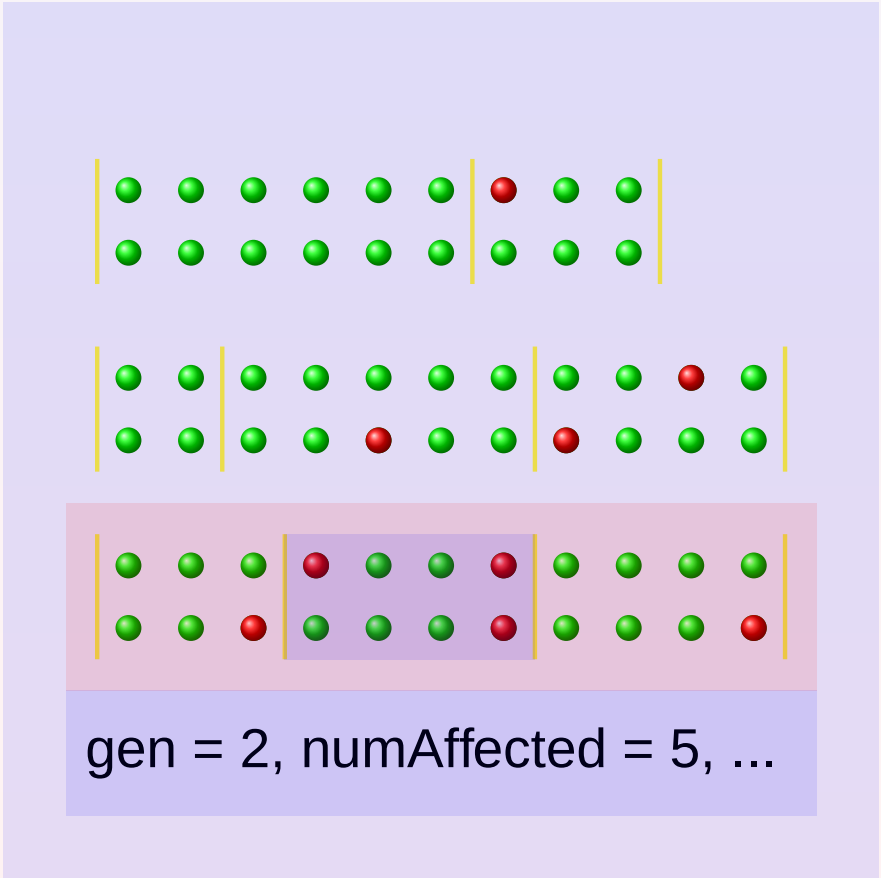



Current generation

# Structure of a population



- Unaffected
- Affected

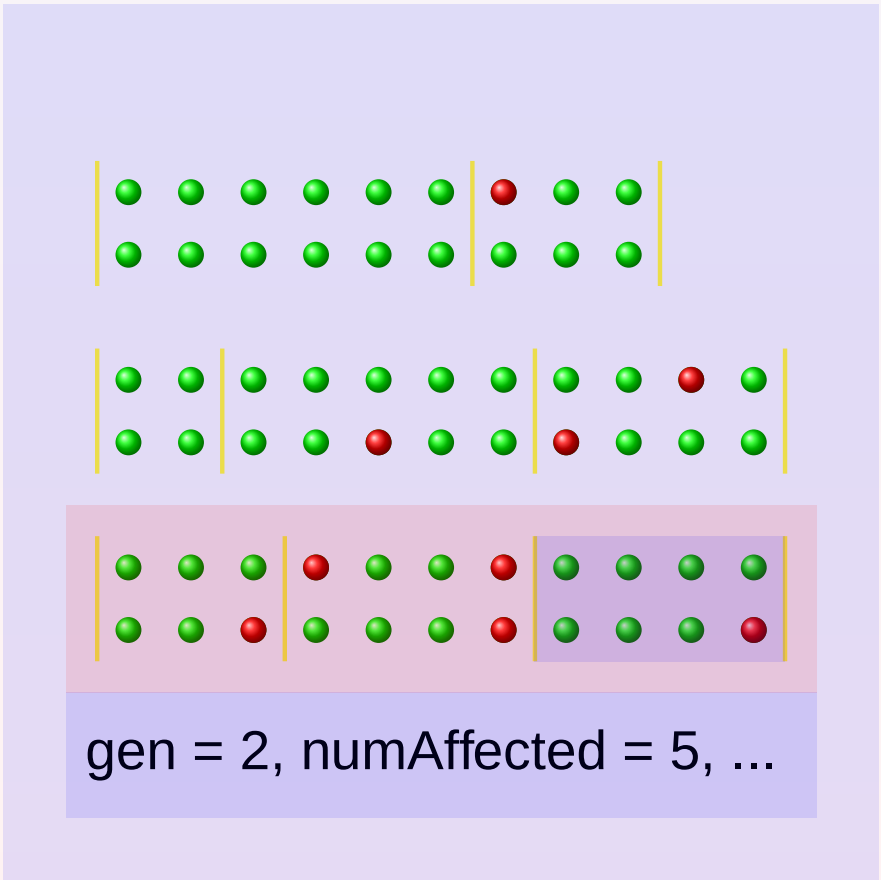



Current generation

# Structure of a population



- Unaffected
- Affected

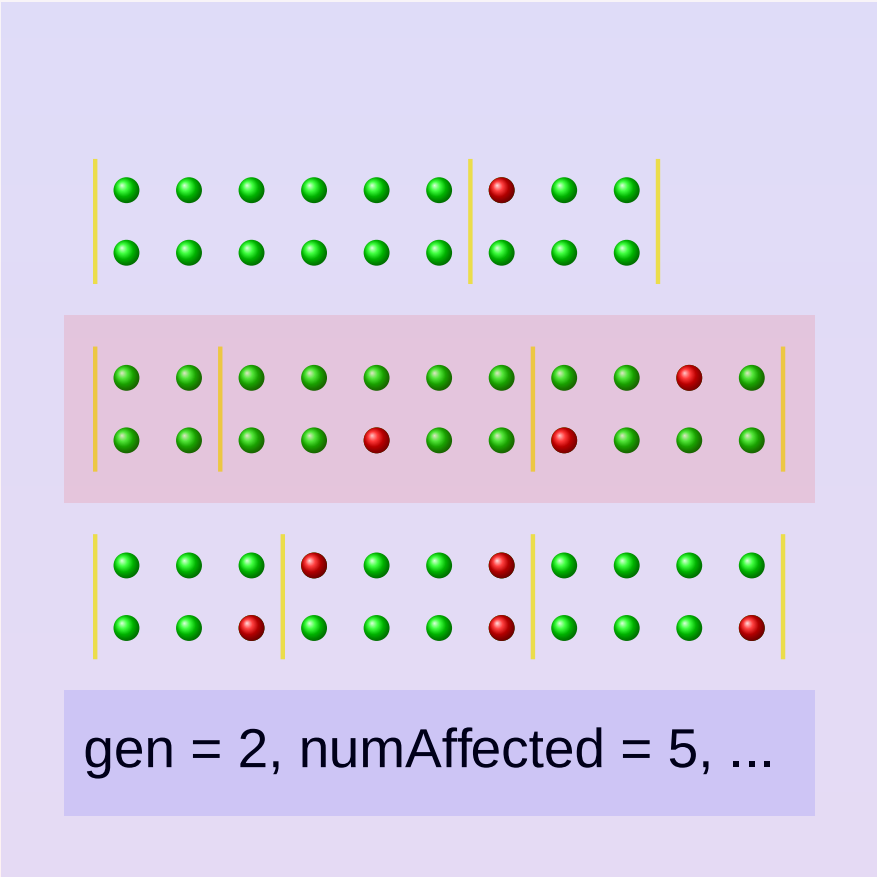



Current generation

# Structure of a population



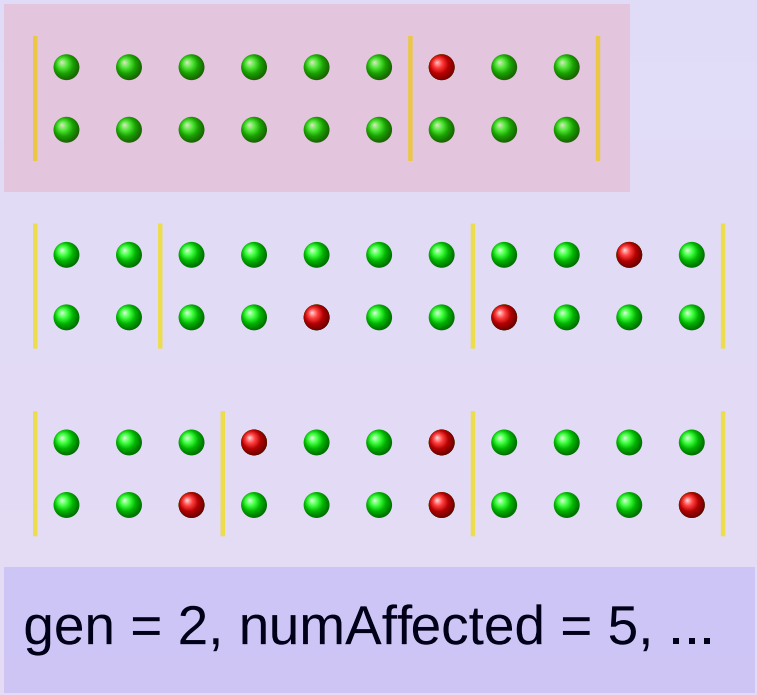
- Unaffected
- Affected



# Structure of a population



- Unaffected
- Affected



Ancestral generation 2

Ancestral generation 1

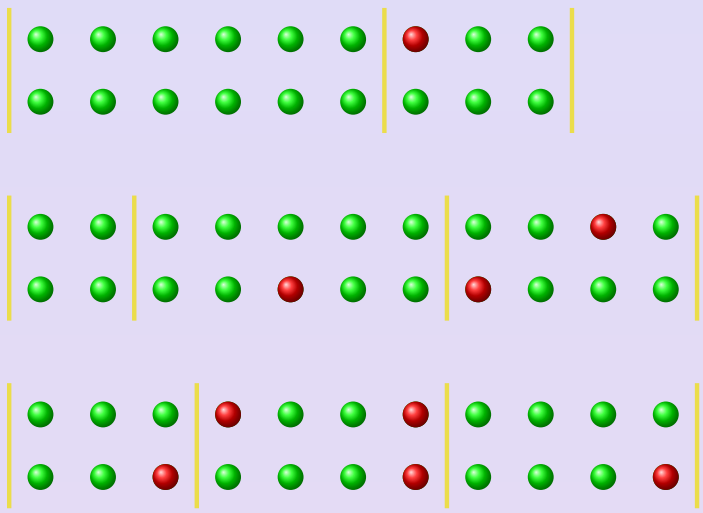
Current generation

gen = 2, numAffected = 5, ...

# Structure of a population



- Unaffected
- Affected



Ancestral generation 2

Ancestral generation 1

Current generation

gen = 2, numAffected = 5, ...

Population variables

# Genotypic Structure

# Common properties of all individuals



All individuals have the same genotypic structure, which refers to

- Ploidy (diploid, haploid, triploid, ...)
- Number of chromosomes
- Number of loci on each chromosome
- Name and position of loci
- Name of information fields

And less importantly

- Allele names
- Existence of sex chromosome

# Chromosome structure



```
>>> pop = population(size=10, loci=[2, 4, 5])
>>> print pop.numLoci()
(2, 4, 5)
>>> # index starts at zero!
>>> print pop.numLoci(1)
4
>>> print pop.ploidy()
2
>>> print pop.ploidyName()
diploid
>>> print pop.chromBegin(1)
2
>>> print pop.locusPos(3)
2.0
>>> print pop.locusName(4)
loc2-3
>>>
```

# Loci position and names



```
>>> pop = population(size=10, loci=[2, 4], maxAllele=3,
...     lociPos=[[1.5, 2.5], [1, 2, 5, 10]],
...     lociNames=['loc%x' % x for x in range(6)],
...     alleleNames=['A', 'T', 'C', 'G'])
>>> print pop.locusPos(3)
2.0
>>> print pop.locusName(4)
loc4
>>> print pop.alleleName(1)
T
>>>
```

# Create and manipulate populations



```
>>> pop = population(size=10, loci=[2, 3])
>>> Dump(pop)
Ploidy:                    2
Number of chrom:           2
Number of loci:            2 3
Maximum allele state:      65535
Loci positions:
                  1 2
                  1 2 3
Loci names:
                  loc1-1 loc1-2
                  loc2-1 loc2-2 loc2-3
population size:           10
Number of subPop:          1
Subpop sizes:              10
Number of ancestral populations:          0
individual info:
sub population 0:
   0: MU    0  0    0  0  0 |    0  0    0  0  0
   1: MU    0  0    0  0  0 |    0  0    0  0  0
   2: MU    0  0    0  0  0 |    0  0    0  0  0
   3: MU    0  0    0  0  0 |    0  0    0  0  0
```

# Genotypic structure



```
>>> pop = population(subPop=[200, 300], loci=[3, 2],
...     maxAllele=3, ploidy=3,
...     lociPos=[[1, 3, 5], [2.5, 4]],
...     alleleNames=['A', 'C', 'T', 'G'])
>>> pop.numLoci(0)
3
>>> pop.totNumLoci()
5
>>> pop.locusPos(4)
4.0
>>> pop.subPopSize(1)
300
>>> pop.popSize()
500
>>> pop.ploidyName()
'triploid'
>>> pop.individual(1).allele(1, 2)
0
>>>
```

# Population manipulation



```
>>> # make a copy of pop
>>> pop1 = pop.clone()
>>> # remove loci 2, 3, 4
>>> pop.removeLoci(keep=[0, 1])
>>> # pop2 will have 3 chromosomes, with loci 2, 3, 2
>>> pop2 = MergePopulationsByLoci(pops=[pop, pop1])
>>> # randomly assign alleles using given allele frequencies
>>> InitByFreq(pop2, [0.8, .2])
>>> # calculate population allele frequency
>>> Stat(pop2, alleleFreq=range(pop2.totNumLoci()))
>>> # print allele frequency
>>> print pop2.dvars().alleleFreq
[[0.8006666666666664, 0.1993333333333333], [0.7933333333333333
>>> # assign affection status using a penetrance model
>>> MapPenetrance(pop2, locus=1,
...     penetrance={'0-0': 0.05, '0-1': 0.2, '1-1': 0.8})
>>> # draw case control sample
>>> (sample,) = CaseControlSample(pop2, cases=5, controls=5)
>>> # save sample in Merlin QTDT format
>>> from simuUtil import SaveQTDT
>>> SaveQTDT(sample, output='sample', affectionCode=['U', 'A'],
...     fields=['affection'])
```

# Population manipulation (cont.)



```
>>> # have a look at the sample in Merlin-QTDT Format
>>> print open('sample.map').read()
CHROMOSOME MARKER POSITION
1        loc1-1  1.000000
1        loc1-2  3.000000
2        loc1-1_1        1.000000
2        loc1-2_1        3.000000
2        loc1-3  5.000000
3        loc2-1  2.500000
3        loc2-2  4.000000

>>> print open('sample.dat').read()
A        affection
M        loc1-1
M        loc1-2
M        loc1-1_1
M        loc1-2_1
M        loc1-3
M        loc2-1
M        loc2-2
```

# Population manipulation (cont.)



```
>>> print open('sample.ped').read()
1 1 0 0 2 A 1 2 2 1 1 1 1 1 1 1 2 1 1 2
2 1 0 0 2 A 1 2 2 2 1 1 1 1 2 2 1 1 1 2
3 1 0 0 1 A 1 2 1 2 1 1 1 2 1 1 1 1 2 1
4 1 0 0 2 A 1 1 2 2 1 1 1 1 2 1 1 1 1 1
5 1 0 0 2 A 1 1 1 1 1 2 1 1 2 1 2 1 1 2
6 1 0 0 1 U 1 1 2 1 2 2 1 1 1 1 1 1 1 1
7 1 0 0 1 U 1 1 1 1 2 1 1 1 1 1 1 1 2 1
8 1 0 0 1 U 1 2 1 1 2 1 1 1 1 2 2 2 1 1
9 1 0 0 1 U 1 1 1 1 1 1 1 1 2 1 1 1 1 1
10 1 0 0 1 U 2 1 2 1 1 1 1 1 2 2 1 2 2 2

>>>
```

# Population variables



```
>>> pop = population(subPop=[5, 10], loci=[5])
>>> InitByFreq(pop, [.6, .3, .1])
>>> Stat(pop, alleleFreq=[1], genoFreq=[2])
>>> print pop.dvars().alleleFreq[1][0]
0.533333333333
>>> from simuUtil import ListVars
>>> ListVars(pop.dvars(), useWxPython=False)
 grp :  -1
 rep :  -1
 alleleNum :
   [1]
     [0]        16
     [1]        13
     [2]        1
 genoFreq :
   [2]
     [0]
       0 :      0.4
       1 :      0.333333333333
       2 :      0.2
     [1]
       2 :      0.0666666666667
 genoNum :
   [2]
     [0]
       0 :      6.0
       1 :      5.0
       2 :      3.0
     [1]
       2 :      1.0
 alleleFreq :
```

# Population variables (cont.)



```
subPop
  [0]
    alleleNum :
      [1]
        [0]    6
        [1]    4
    genoNum :
      [2]
        [0]
          0 :  4.0
          1 :  1.0
    genoFreq :
      [2]
        [0]
          0 :  0.8
          1 :  0.2
    alleleFreq :
      [1]
        [0]    0.6
        [1]    0.4
  [1]
    alleleNum :
      [1]
        [0]    10
        [1]    9
        [2]    1
```

# Structure of Individuals



Assume ploidy = 2, maxAllele = 1

| 0 | 1 | 2 | | 3 | 4 | | | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | | 1 | 0 | | | 0 | 1 |
| 0 | 0 | 1 | | 1 | 1 | | | 0 | 1 |

| 0 | | | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| 0 | | | 1 | 0 | 0 | 0 | 1 |
| 1 | | | 0 | 1 | 1 | 0 | 0 |

Male

- Affected

| fitness | father_id | ··· |
|---|---|---|

# Structure of Individuals



Assume ploidy = 2, maxAllele = 1

| 0 | 1 | 2 | | 3 | 4 | | | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | | 1 | 0 | | | 0 | 1 |
| 0 | 0 | 1 | | 1 | 1 | | | 0 | 1 |

| 0 | | | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| 0 | | | 1 | 0 | 0 | 0 | 1 |
| 1 | | | 0 | 1 | 1 | 0 | 0 |

Chromosome 0

Male

- Affected

| fitness | father_id | ··· |
|---|---|---|

# Structure of Individuals



Assume ploidy = 2, maxAllele = 1

| 0 | 1 | 2 | | 3 | 4 | | | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | | 1 | 0 | | | 0 | 1 |
| 0 | 0 | 1 | | 1 | 1 | | | 0 | 1 |

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 |

Chromosome 0

Chromosome 1

Male

- Affected

| fitness | father_id | ··· |
|---|---|---|

# Structure of Individuals



Assume ploidy = 2, maxAllele = 1

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |

Chromosome 0

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 |

Chromosome 1

Male

Sex

- Affected

| fitness | father_id | ··· |
|---|---|---|

# Structure of Individuals



Assume ploidy = 2, maxAllele = 1

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 |

Chromosome 0

Chromosome 1

Male

Sex

- Affected

Affection status

| fitness | father_id | ··· |
|---|---|---|

# Structure of Individuals



Assume ploidy = 2, maxAllele = 1

| 0 | 1 | 2 | | 3 | 4 | | | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | | 1 | 0 | | | 0 | 1 |
| 0 | 0 | 1 | | 1 | 1 | | | 0 | 1 |

| 0 | | | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| 0 | | | 1 | 0 | 0 | 0 | 1 |
| 1 | | | 0 | 1 | 1 | 0 | 0 |

Male

- Affected

| fitness | father_id | ... |
|---|---|---|

Chromosome 0

Chromosome 1

Sex

Affection status

Information fields

# Individuals



```
>>> pop = population(subPop=[5, 8], loci=[5],
...     infoFields=['penetrance'])
>>> InitByFreq(pop, [.6, .3, .1])
>>> MaPenetrance(pop, locus=2, penetrance=[0.05, 0.2, 0.5],
...     wildtype=[0], infoFields=['penetrance'])
>>> # iterate through all inviduals in subPop 1
>>> for ind in pop.individuals(1):
...     print 'Aff: %d Fit: %.3f Geno: %d %d' % \
...         (ind.affected(), ind.info('penetrance'), \
...         ind.allele(2, 0), ind.allele(2, 1))
...
Aff: 0 Fit: 0.500 Geno: 2 1
Aff: 0 Fit: 0.200 Geno: 0 1
Aff: 0 Fit: 0.050 Geno: 0 0
Aff: 0 Fit: 0.050 Geno: 0 0
Aff: 0 Fit: 0.200 Geno: 1 0
Aff: 0 Fit: 0.200 Geno: 2 0
Aff: 0 Fit: 0.200 Geno: 0 1
Aff: 0 Fit: 0.200 Geno: 0 1
>>>
```

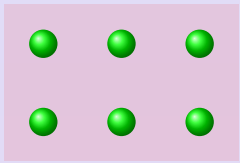
# Information fields



```
>>> pop = population(100, loci=[5, 8],
...     infoFields=['father_idx', 'mother_idx'])
>>> simu = simulator(pop, randomMating(numOffspring=2))
>>> simu.evolve(ops=[parentsTagger()], end=5)
True
>>> ind = simu.population(0).individual(0)
>>> ind1 = simu.population(0).individual(1)
>>> print ind.info('father_idx'), ind.info('mother_idx')
34.0 76.0
>>> print ind1.info('father_idx'), ind1.info('mother_idx')
34.0 76.0
>>>
>>>
```

# Life cycle of a generation



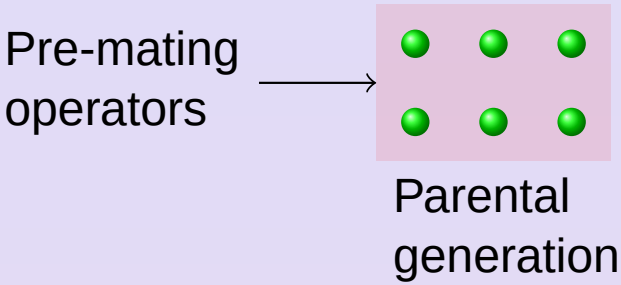


Parental generation

Every operator has a default stage, and a `stage` parameter to change it.

# Life cycle of a generation

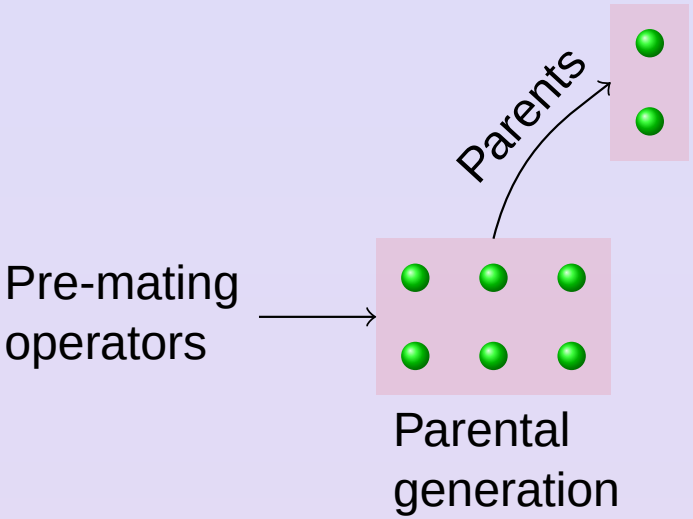





Every operator has a default stage, and a `stage` parameter to change it.

# Life cycle of a generation

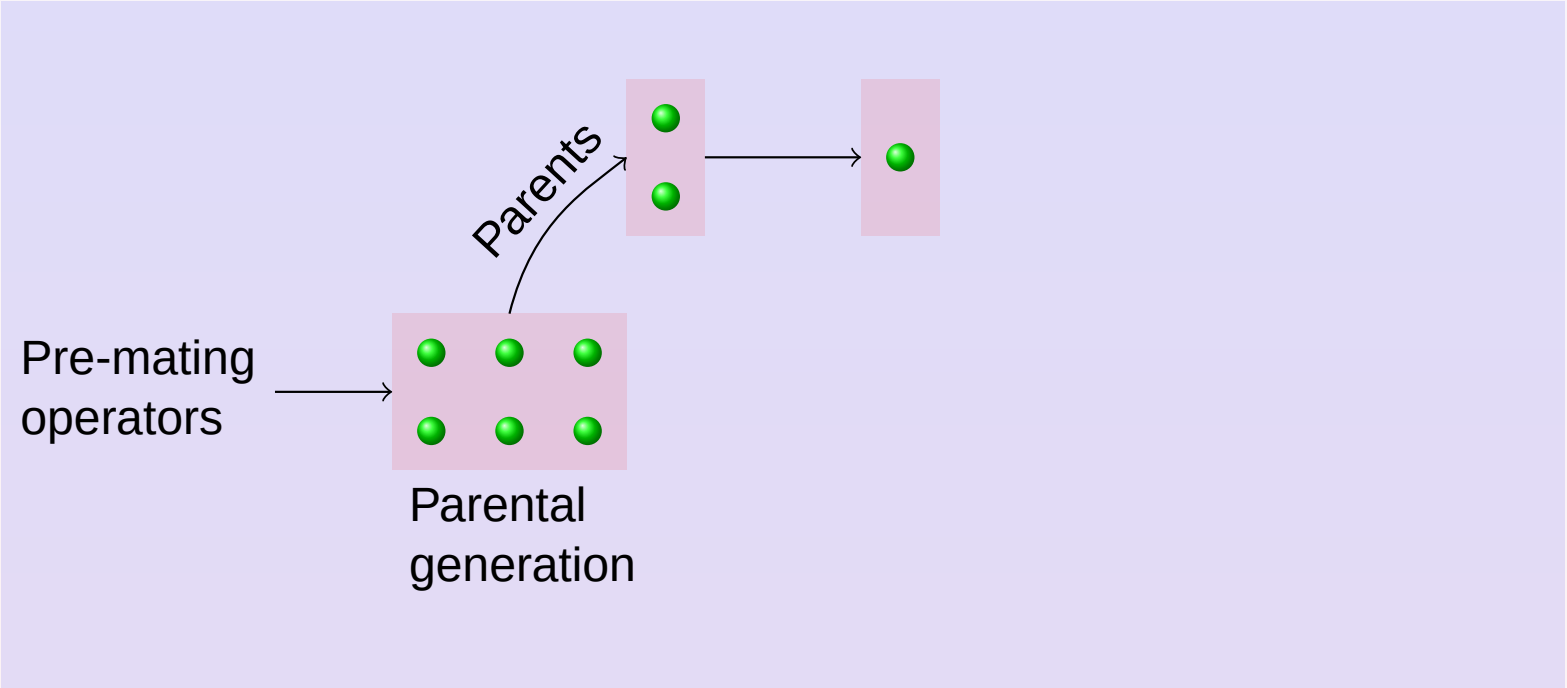





Every operator has a default stage, and a `stage` parameter to change it.

# Life cycle of a generation

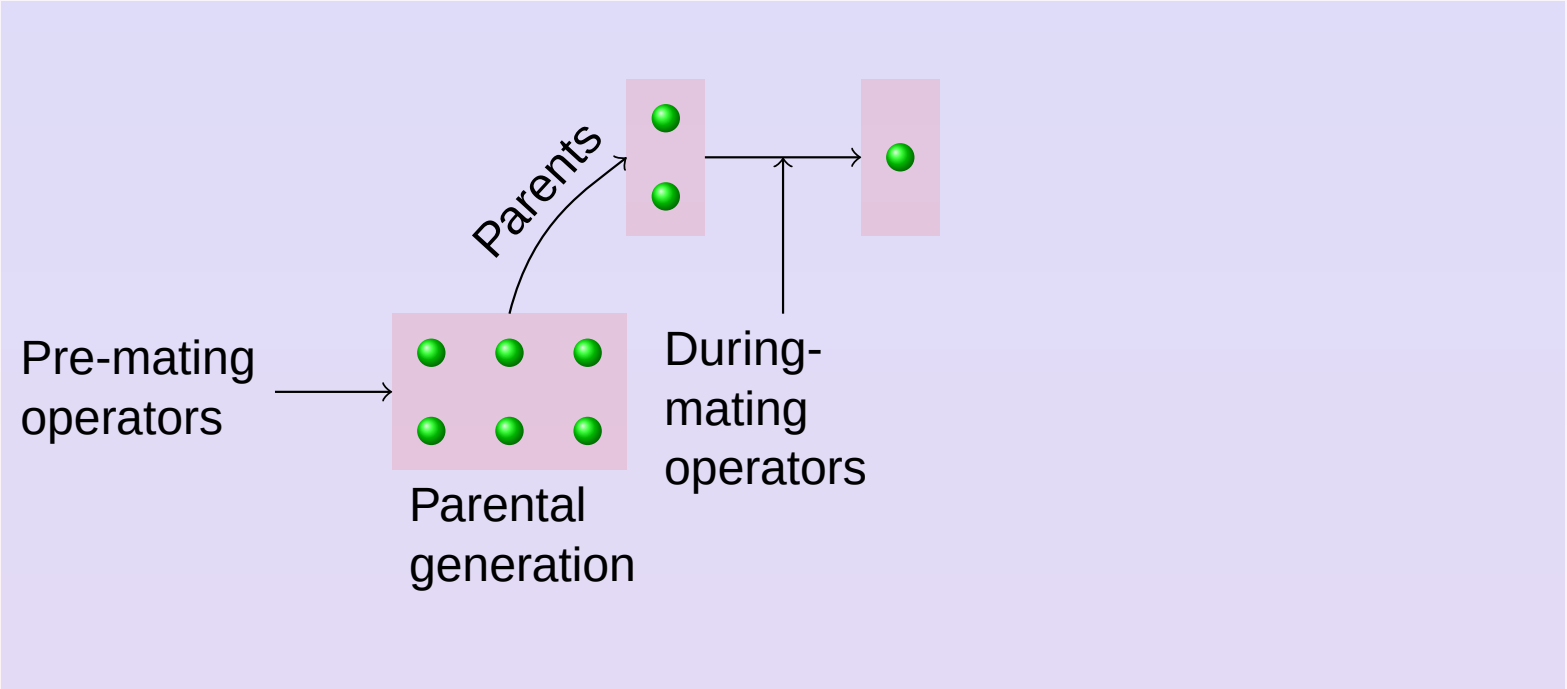





Every operator has a default stage, and a `stage` parameter to change it.

# Life cycle of a generation

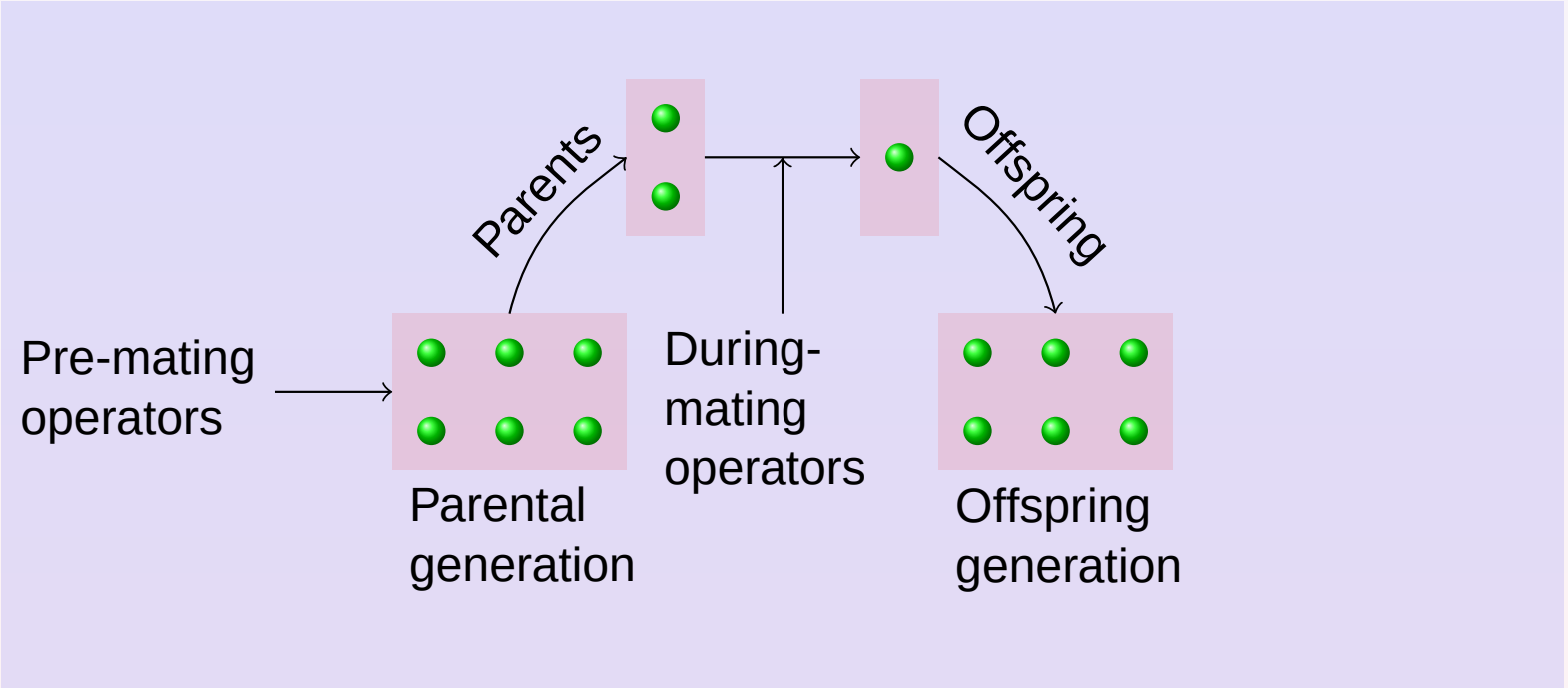





Every operator has a default stage, and a `stage` parameter to change it.

# Life cycle of a generation

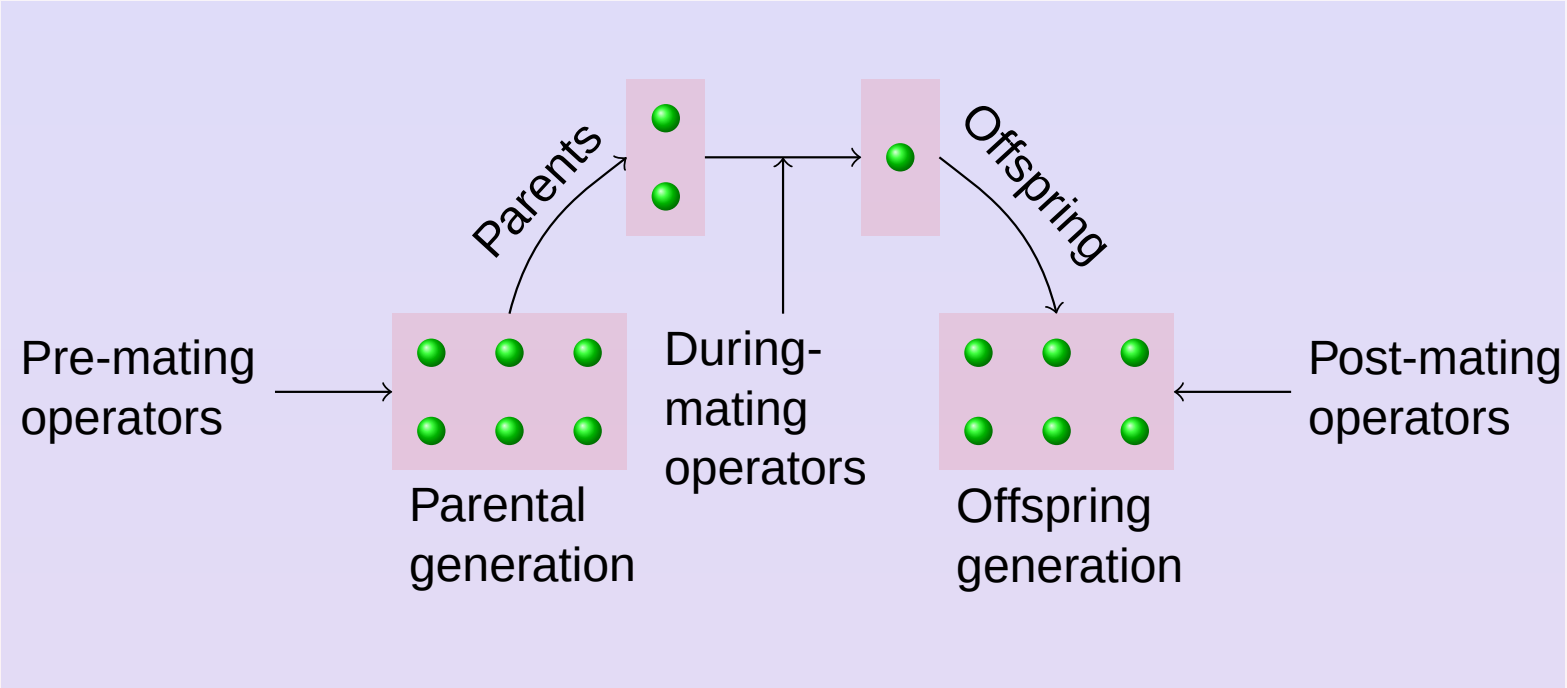





Every operator has a default stage, and a `stage` parameter to change it.

# Life cycle of a generation






Every operator has a default stage, and a `stage` parameter to change it.

# Pre-, During- and PostMating operators



```
>>> simu = simulator(
...     population(subPop=[20, 80], loci=[3]),
...     randomMating())
>>> simu.evolve(
...     preOps = [initByFreq([0.2, 0.8])],
...     ops = [
...         kamMutator(maxAllele=10, rate=0.00005, atLoci=[0,2]),
...         recombinator(rate=0.001),
...         dumper(stage=PrePostMating),
...         stat(alleleFreq=[1]),
...     ],
...     dryrun=True
... )
Dryrun mode: display calling sequence
Apply pre-evolution operators
  Replicate 0
    Apply pre-mating ops
      - <simuPOP::initByFreq> end at 1
Start evolution:
  Replicate 0
    Pre-mating operators:
      - <simuPOP::dumper> at all generations
    Start mating.
      - <simuPOP::recombination> at all generations
    Apply post-mating operators
      - <simuPOP::k-allele model mutator K=10> at all generations
      - <simuPOP::dumper> at all generations
      - <simuPOP::statistics> at all generations
True
>>>
```

# Applicable generations



```
>>> simu = simulator(
...     population(10000, loci=[3]),
...     randomMating())
>>> eval1 = r"'Gen: %3d  Freq: %f\n' % (gen, alleleFreq[1][0])"
>>> eval2 = r"'Last Gen: %3d  Freq: %s\n' % (gen, alleleFreq[1])"
>>> simu.evolve(
...     preOps = [initByFreq([0.3, 0.7])],
...     ops = [
...         recombinator(rate=0.01, begin=10, end=30),
...         stat(alleleFreq=[1], step=10),
...         pyEval(eval1, step=10),
...         pyEval(eval2, at=[-1])
...     ],
...     end = 50
... )
Gen:   0  Freq: 0.300050
Gen:  10  Freq: 0.310100
Gen:  20  Freq: 0.294950
Gen:  30  Freq: 0.297350
Gen:  40  Freq: 0.292450
Gen:  50  Freq: 0.292550
Last Gen:  50  Freq: [0.29254999999999998, 0.70745000000000002]
True
>>>
```

# Applicable replicates



```
>>> simu = simulator(
...     population(100, loci=[3]),
...     randomMating(),
...     rep=5, grp=[1,1,2,2,2])
>>> simu.evolve(
...     preOps = [initByFreq([0.5, 0.5])],
...     ops = [
...         stat(alleleFreq=[1]),
...         recombinator(rate=0.01, grp=1),
...         recombinator(rate=0.01, grp=2),
...         pyEval(r"'%.2f ' % alleleFreq[1][0]", grp=1),
...         pyEval(r"'\n'", rep=REP_LAST),
...     ],
...     end=5
... )
Warning: More than one during mating operators. Make sure they a
0.48 0.47
0.52 0.46
0.45 0.42
0.46 0.47
0.45 0.48
0.50 0.47
True
```

# Output



```
>>> simu = simulator(
...     population(100, loci=[3]),
...     randomMating(),
...     rep=5, grp=[1,1,2,2,2])
>>> simu.evolve(
...     preOps = [initByFreq([0.5, 0.5])],
...     ops = [
...         stat(alleleFreq=[1]),
...         pyEval(r"'%.2f ' % alleleFreq[1][0]",
...             output='>>out'),
...         pyEval(r"'\n'", rep=REP_LAST, output='>>out'),
...         pyEval(r"'%.2f ' % alleleFreq[1][0]",
...             outputExpr="'>>out%d' % grp"),
...     ],
...     end=2
... )
True
>>> print open('out').read()
0.55 0.47 0.47 0.51 0.47
0.50 0.42 0.52 0.55 0.44
0.51 0.47 0.56 0.52 0.43

>>> print open('out1').read()
0.55 0.47 0.50 0.42 0.51 0.47
>>> print open('out2').read()
0.47 0.51 0.47 0.52 0.55 0.44 0.56 0.52 0.43
>>>
```

# Mating schemes



## Mating schemes

- Population offspring subpopulation from corresponding parental subpopulation
- Can change subpopulation size
- Select parents according to their `fitness` value (information field)
- Can produce more than one offspring

# Demographic model



```
>>> def lin_inc(gen, oldsize=[]):
...     return [10+gen]*5
...
>>> simu = simulator(
...     population(subPop=lin_inc(1), loci=[1]),
...     randomMating(newSubPopSizeFunc=lin_inc)
... )
>>> simu.evolve(
...     ops = [
...         stat(popSize=True),
...         pyEval(r'"%d %d\n"%(gen, subPop[0]["popSize"])'),
...     ],
...     end=5
... )
0 10
1 11
2 12
3 13
4 14
5 15
True
>>>
```

# Number of offspring



```
>>> simu = simulator(
...     population(size=10000, loci=[1]),
...     randomMating(),
... )
>>> simu.evolve(
...     preOps = [initByFreq([0.1, 0.9])],
...     ops = [ ], end=100
... )
True
>>> simu.setMatingScheme(randomMating(numOffspring=2))
>>> simu.addInfoFields(['father_idx', 'mother_idx'])
>>> simu.setAncestralDepth(1)
>>> simu.step(ops=[parentsTagger()])
True
>>> pop = simu.getPopulation(0)
>>> MaPenetrance(pop, locus=0, penetrance=[0.05, 0.1, 0.5])
>>> AffectedSibpairSample(pop, size=100)
[<simuPOP::population of size 200>]
>>>
```

# Simulator



## A simulator manages

- Replicates of a population
- A mating scheme
- Many operators

and evolve the populations.

# Utility modules and scripts



**simuOpt.py** provides an easy way to handle parameters.

**simuUtil.py** provides functions to save/load in many formats, gene mapping functions, list variables etc

**simuCluster.py** a control script to send jobs to cluster systems

**simuLDDecay.py** a simple script to demonstrate the decay of linkage disequilibrium under recombination

**simuForward.py** implements a traditional forward-time simulation scenario

**simuComplexDisease.py** implements a new forward-time simulation method (PLoS Genetics, 2007)

**simuCDCV.py** demonstrate the evolution of allelic spectrum

...

# Outline



## 2 Populations

- Overview
- Population structure
- Information fields

```
>>> Dump(pop)
Ploidy:                    2
Number of chrom:           2
Number of loci:            2 3
Maximum allele state:   65535
Loci positions:
                  1 2
                  1 2 3
Loci names:
                  loc1-1 loc1-2
                  loc2-1 loc2-2 loc2-3
population size:           10
Number of subPop:          1
Subpop sizes:              10
Number of ancestral populations:         0
individual info:
sub population 0:
   0: MU   0  0   0  0  0 |   0  0   0  0  0
   1: MU   0  0   0  0  0 |   0  0   0  0  0
   2: MU   0  0   0  0  0 |   0  0   0  0  0
   3: MU   0  0   0  0  0 |   0  0   0  0  0
   4: MU   0  0   0  0  0 |   0  0   0  0  0
   5: MU   0  0   0  0  0 |   0  0   0  0  0
   6: MU   0  0   0  0  0 |   0  0   0  0  0
   7: MU   0  0   0  0  0 |   0  0   0  0  0
   8: MU   0  0   0  0  0 |   0  0   0  0  0
   9: MU   0  0   0  0  0 |   0  0   0  0  0
End of individual info.
```

```
>>> InitByFreq(pop, alleleFreq=[0.3, 0.7])
>>> Dump(pop)
Ploidy:                    2
Number of chrom:           2
Number of loci:            2 3
Maximum allele state:   65535
Loci positions:
                  1 2
                  1 2 3
Loci names:
                  loc1-1 loc1-2
                  loc2-1 loc2-2 loc2-3
population size:           10
Number of subPop:          1
Subpop sizes:              10
Number of ancestral populations:         0
individual info:
sub population 0:
   0: FU   0  1   0  0  1 |   1  1   1  0  0
   1: MU   1  1   1  0  1 |   1  1   1  1  1
   2: MU   1  1   1  1  1 |   1  0   1  0  1
   3: MU   0  1   1  1  1 |   1  1   1  1  0
   4: MU   0  1   1  0  1 |   1  1   1  0  1
   5: MU   1  1   1  1  0 |   1  1   1  1  1
   6: FU   1  1   1  1  1 |   1  0   1  0  0
   7: MU   1  0   1  1  1 |   0  1   0  1  0
   8: MU   0  0   1  1  1 |   1  1   1  0  1
   9: MU   0  0   1  1  0 |   0  1   1  1  1
```

```
>>> pop = population(subPop=[2, 5, 6], loci=[2])
>>> print pop.popSize()
13
>>> print pop.subPopSizes()
(2, 5, 6)
>>> print pop.subPopSize(1)
5
>>> Dump(pop, infoOnly=True)
Ploidy:                    2
Number of chrom:           1
Number of loci:            2
Maximum allele state:   65535
Loci positions:
                  1 2
Loci names:
                  loc1-1 loc1-2
population size:           13
Number of subPop:          3
Subpop sizes:              2 5 6
Number of ancestral populations:         0
>>>
```

# Mating is within subpopulation only



```
>>> pop = population(subPop=[5, 6], loci=[2])
>>> simu = simulator(pop, randomMating())
>>> simu.evolve(
...     preOps = [
...         initByFreq(alleleFreq=[0.2, 0.8], subPop=[0]),
...         initByFreq([0, 0, 0, 0.5, 0.5], subPop=[1])
...         ],
...     ops = [
...         dumper(alleleOnly=True, indRange=[[0, 3], [5, 7]]),
...         recombinator(rate=0.1) ],
...     end = 1
... )
```

# Mating is within subpopulation only – continue



```
individual info:
sub population 0:
   0: MU   1  1 |   0  1
   1: FU   1  0 |   0  1
   2: FU   1  1 |   1  1
sub population 1:
   5: FU   3  4 |   3  4
   6: FU   3  4 |   3  4
End of individual info.


No ancenstral population recorded.
individual info:
sub population 0:
   0: MU   1  1 |   0  1
   1: MU   0  1 |   1  1
   2: FU   1  1 |   1  1
sub population 1:
   5: MU   3  4 |   4  3
   6: MU   4  3 |   3  4
End of individual info.
```

# Subpopulation manipulations

# Information fields



Pieces of information that can be attached to each individual, e.g.

- `fitness:` fitness of each individual, calculated by selectors
- `father_idx, mother_idx`: index of parents in the parental generation
- `old_index:` index of an individual in the population where it is sampled

Or, self-defined

- birthday
- geographic location
- ...

# Information fields – an example

# Outline



## 3 Operators

- Python Operators

The most flexible operators that can perform any operation, but are less efficient.

The idea: user provide a function with specified input and output, simuPOP calls this function during evolution.

# Python operator



```
func(pop [, param])
```

# Python Individual operator



`func(ind [, genotype] [, param]), return True/False or an array`

- `ind`: individual
- `genotype`: if parameter loci is given, genotype at these loci are passed to the function
- `param`: if parameter param is given, param passed from simuPOP
- `return`: if parameter infoFields is given, assign return values to these information fields

# An example of `pyIndOperator`

# Specialized Python operators: pyPenetrance

# pySelector

# Outline



## 4 Simulator

# simulator

# mating scheme

# evolve!

# Exercise time!



simuLDDecay.py

# Outline



## 5 simuPOP components

- Population object
- Operators
- Mating scheme, Simulator and forward-time simulation

# Create a population

# Genotypic structure

# Individuals

# Population strcuture

# Information fields

# Variables

# Stages

# Stages, an example

# Output

# Table-like output

# Mating schemes

# Simulator

# Evolve?!

# Outline



## 6 A real example

- Handling of HapMap data

# Outline



## 7 A quick Python tutorial

# Outline



## 8 Some more examples

- Example 1
- Example 2

# Manipulating of HapMap data

# Outline



## 9 Advanced topics