Serotinous:
Stores seeds, released after fire.
Dense postfire recovery.
No reproduction without fire.

Non-serotinous
Disperses seeds annually.
Good at colonizing gaps.
Poor reseeding after fire.

Model objective: Understand the evolution of serotiny in response to changes in mean fire size

- Do we really need a spatially explicit model?
- Model structure
- Spatial model implementation
  - How to simulate fire?
  - Programming concepts—objects & references
  - Computational efficiency

Is a spatially explicit model necessary?

What patterns and processes are of interest?

How do these patterns/processes relate to space?

Patterns: both spatial and non-spatial

Non-spatial:
How do phenotype frequencies change?

Spatially implicit
What proportion of the landscape will be dominated by the **serotinous** type?

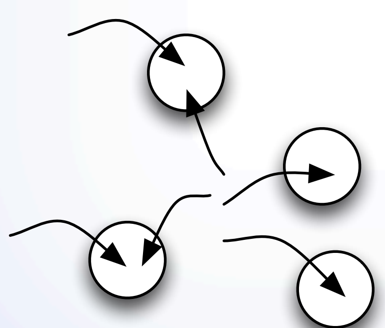
Spatially explicit
Do the two types become more or less aggregated?
Is there a critical threshold beyond which the **non-serotinous** type cannot reach parts of the landscape?

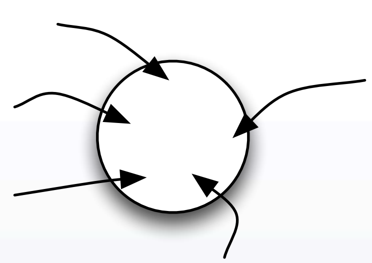
Processes: climate change will increase the average size of fires

Spatially implicit
Increase the % of landscape burned annually

Spatially explicit
Larger fires will reduce the ability of plants to colonize from unburned areas

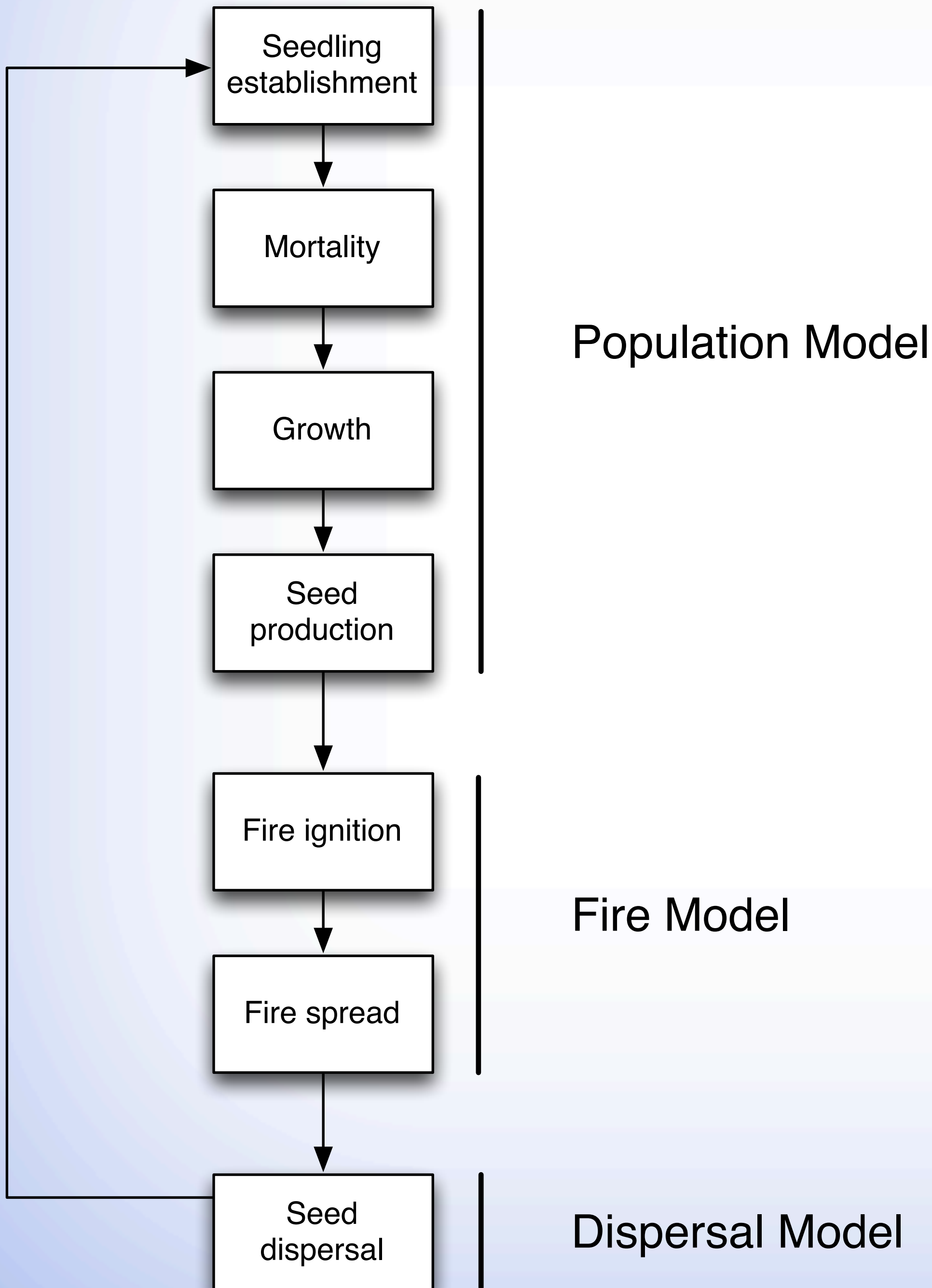
Small fires:
dispersal is easy





Large fires:
dispersal to center is difficult

Model Structure



- Lattice-based model (i.e., a grid)
- Similar to a cellular automaton
- State of a cell at time t+1 is a function only of the states of the cell and its neighbors at time t

Population model:

- local dynamics only
- simple logistic model

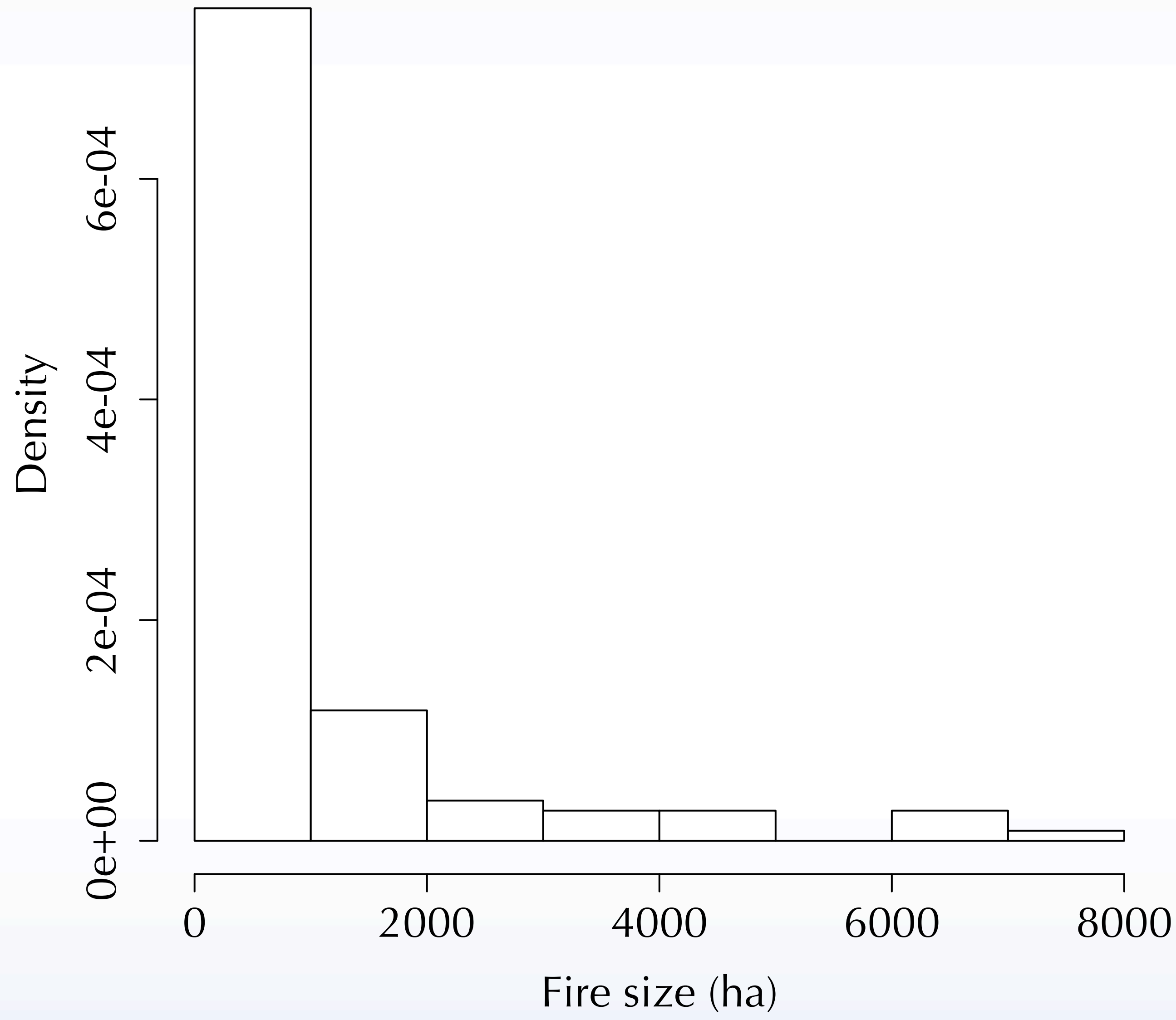
Dispersal:

- Seeds disperse to immediate neighbors

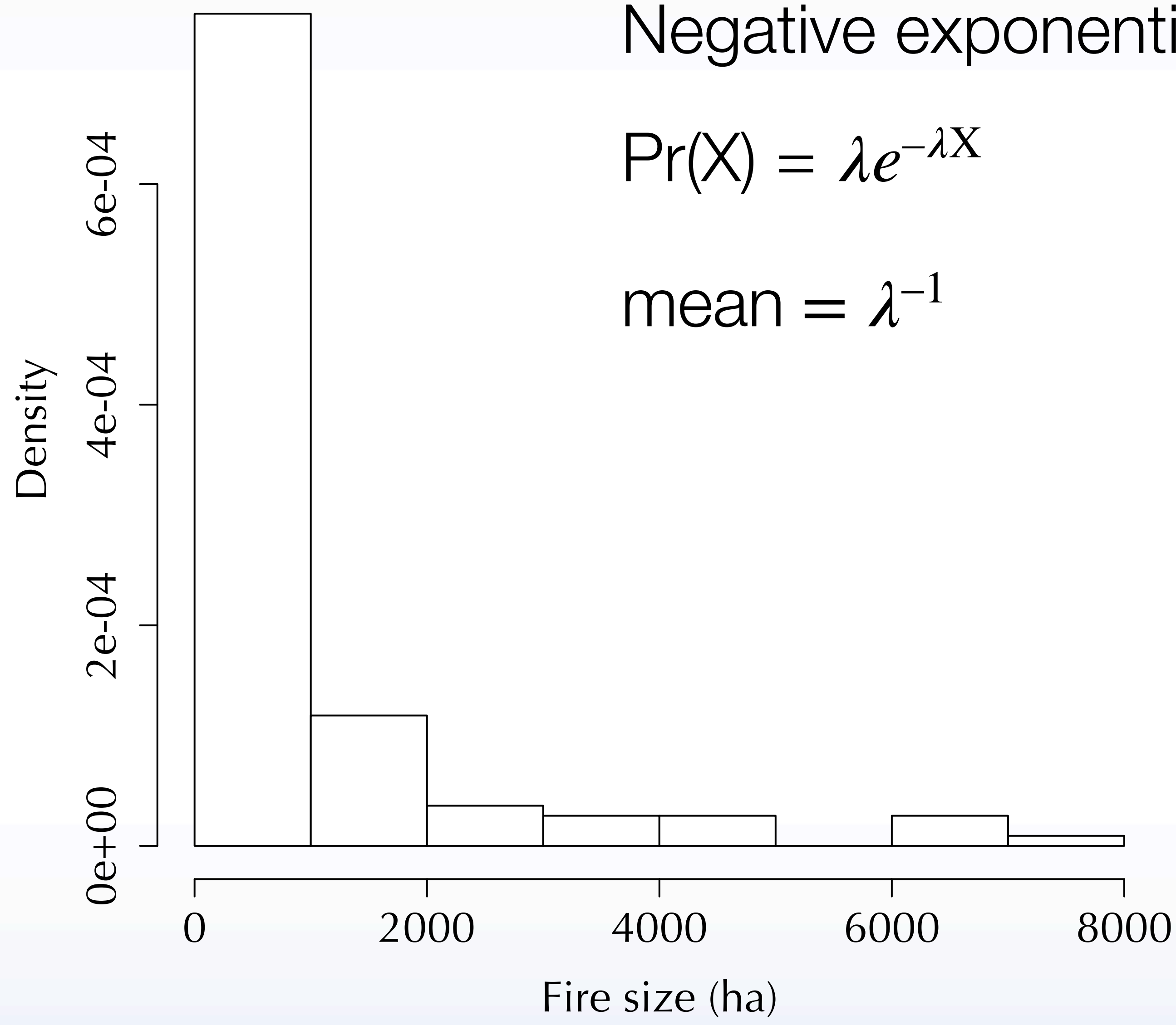
Fire Model

1. How many fires occur per year?
2. How large are the fires?
3. Where does each fire start?
4. Where does each fire spread?
5. Implementation must be efficient

How large are the fires?



How large are the fires?

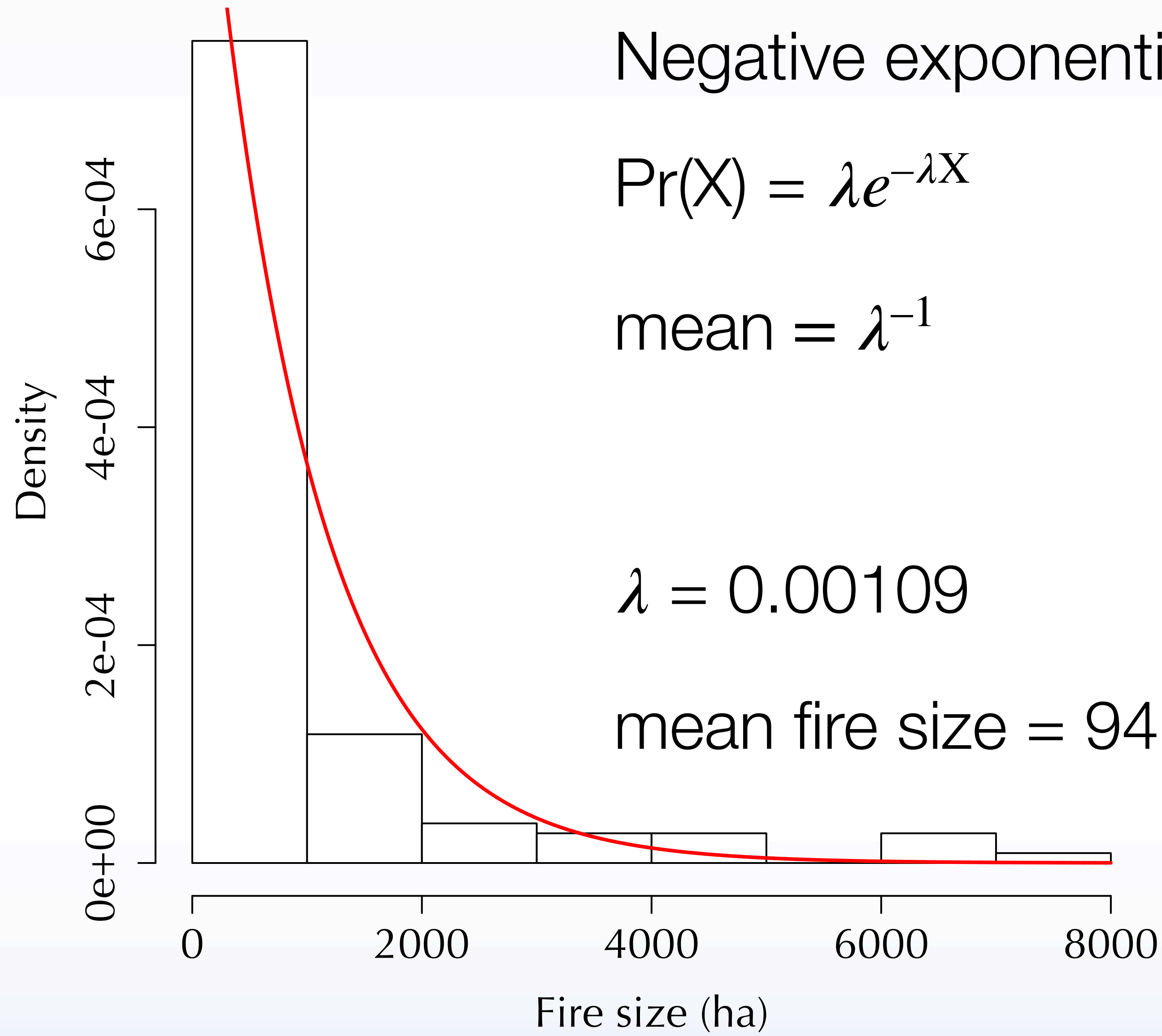


Negative exponential distribution:

$\Pr(X) = \lambda e^{-\lambda X}$

mean $= \lambda^{-1}$

How large are the fires?



Negative exponential distribution:

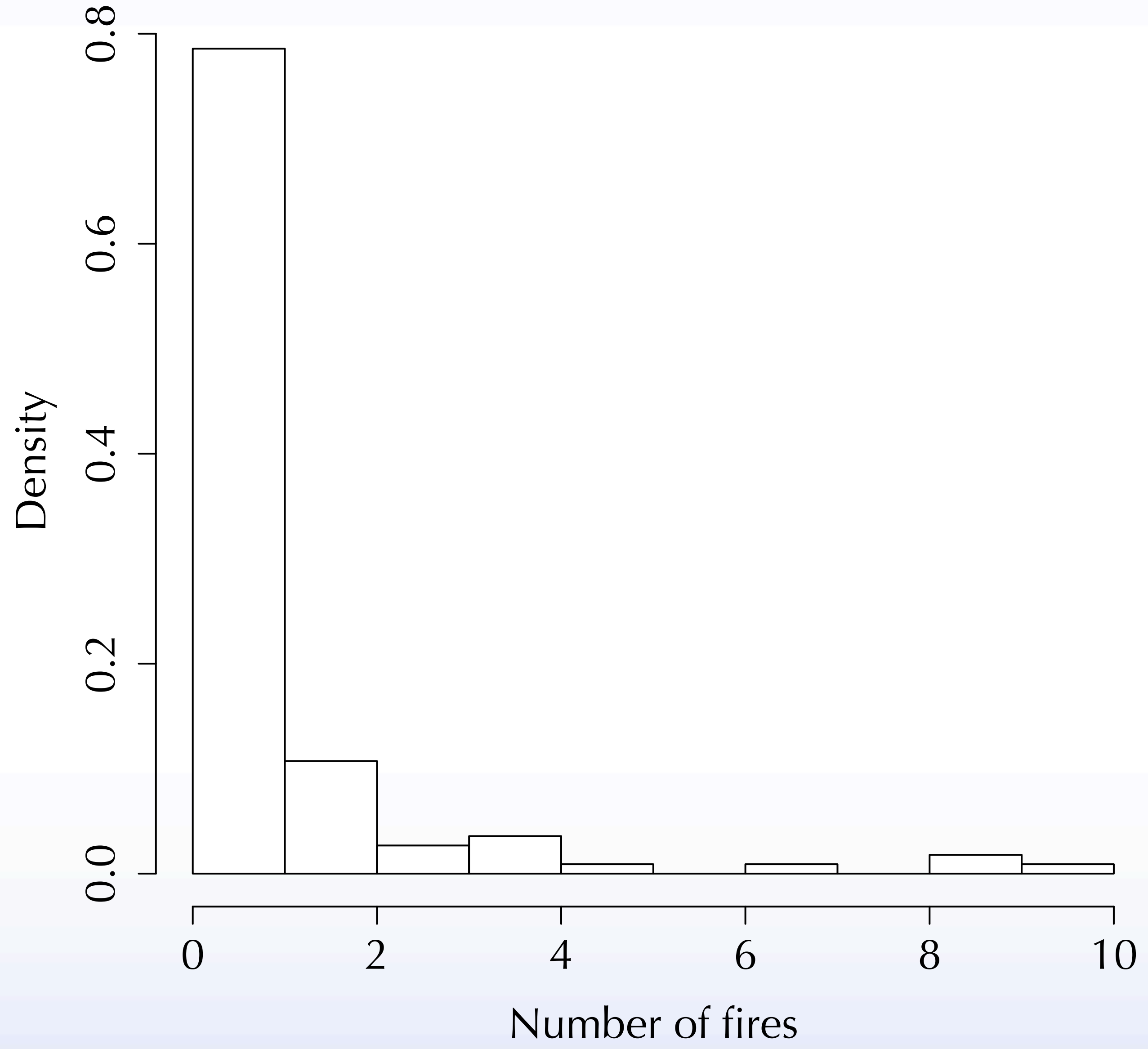
$\Pr(X) = \lambda e^{-\lambda X}$

$\text{mean} = \lambda^{-1}$

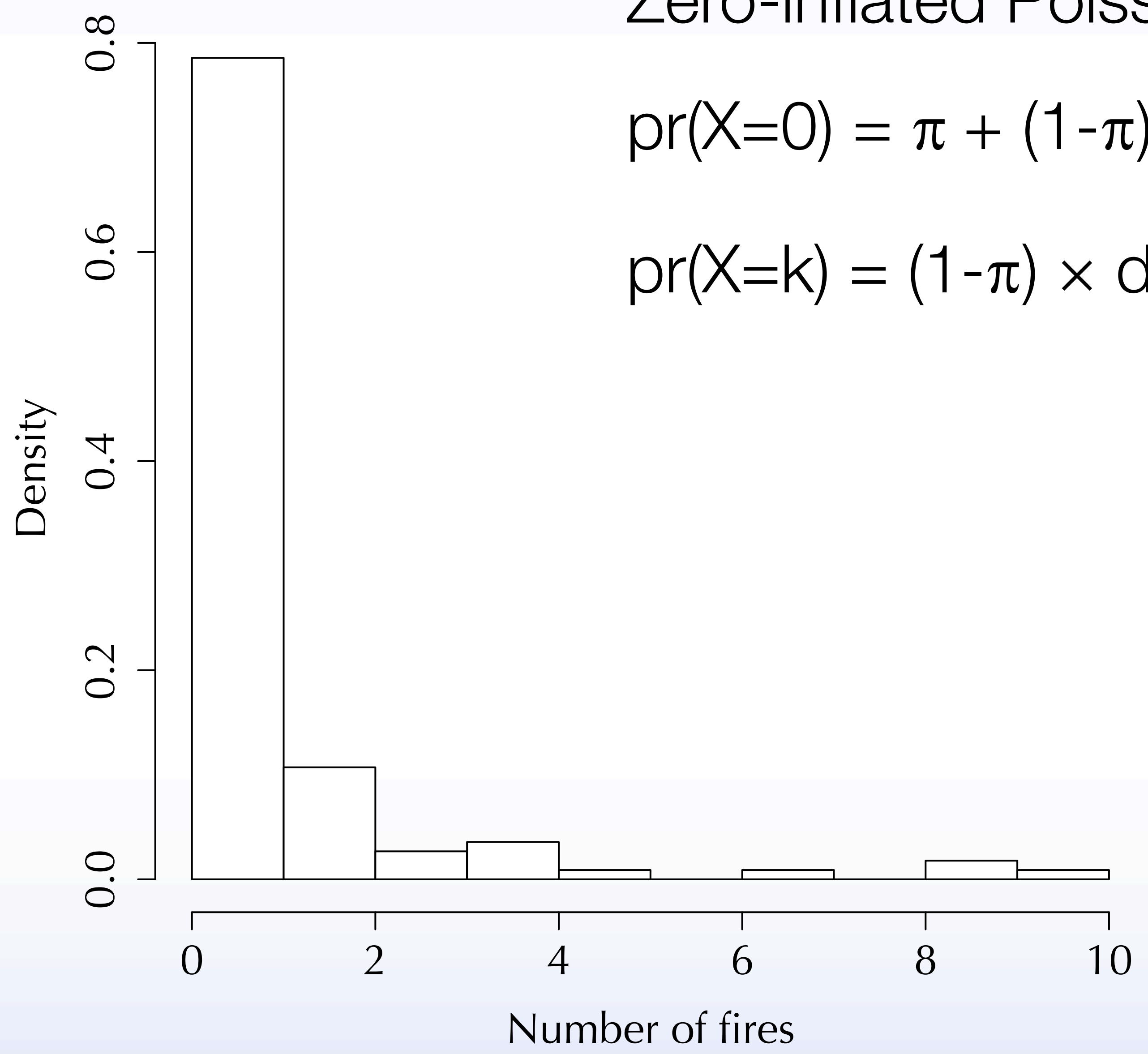
$\lambda = 0.00109$

mean fire size = 941.9 ha

How Many Fires Occur?



How Many Fires Occur?

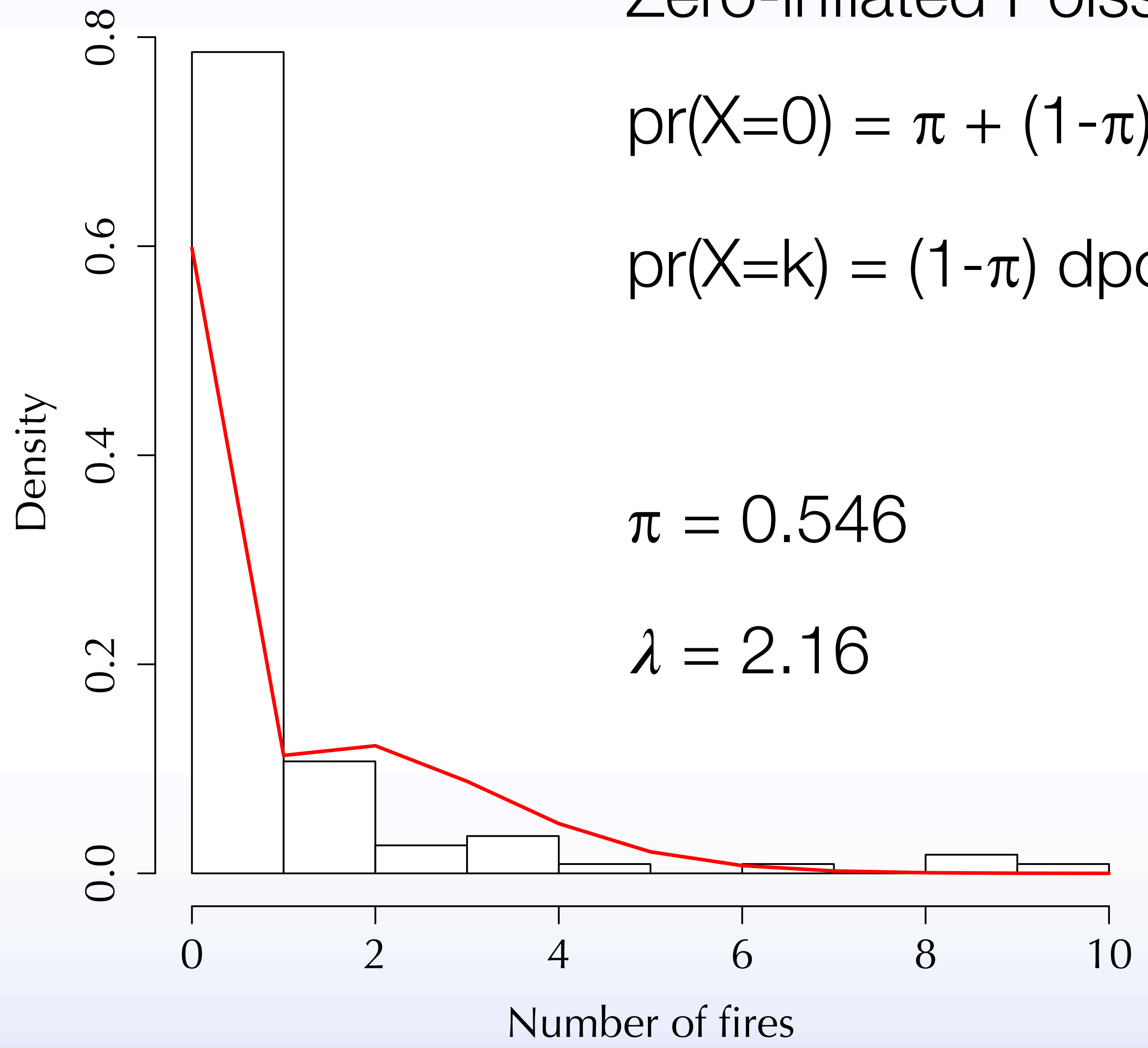


Zero-inflated Poisson:

pr(X=0) = $\pi + (1-\pi) \times$ dpois(0, λ)

pr(X=k) = $(1-\pi) \times$ dpois(k, λ)

How Many Fires Occur?



Zero-inflated Poisson:

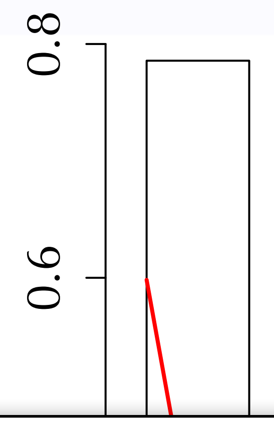
pr(X=0) = $\pi + (1-\pi) \times$ dpois(0, λ)

pr(X=k) = $(1-\pi)$ dpois(k, λ)

$\pi = 0.546$

$\lambda = 2.16$

How Many Fires Occur?



Zero-inflated Poisson:

pr(X=0) = $\pi + (1-\pi) \times$ dpois(0, λ)

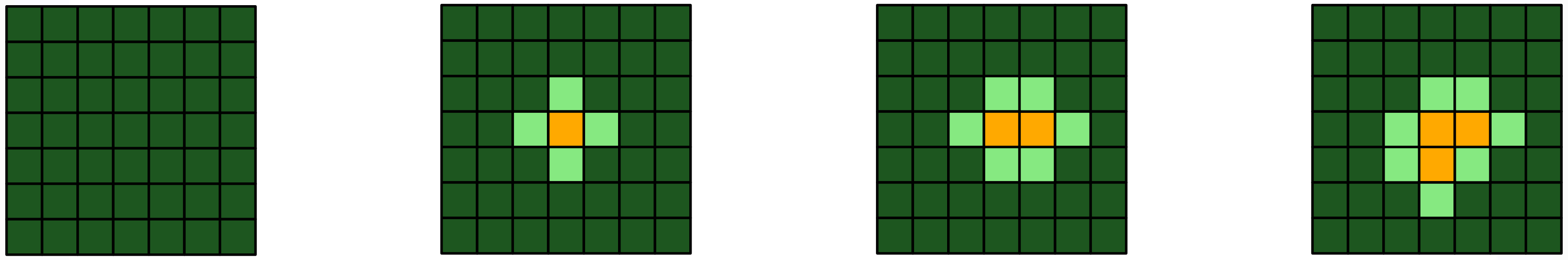
pr(X=k) = $(1-\pi)$ dpois(k, λ)

```
dzip <- function(x, pi, lambda) {
  pr <- log(1-pi) + dpois(x, lambda, log=T)
  pr[x == 0] <- log(exp(pr[x==0]) + pi)
  return(pr)
}

n_log_lhood <- function(pars) {
  return( -sum(dzip(count_data, pars[1], pars[2])))
}

optim(par=c(0.5,2), fn=n_log_lhood)
```

Where do fires start and spread?

Important to start and spread randomly (i.e., without bias)

It is difficult to accomplish this efficiently



1. Limit looping across large arrays

2. Avoid “visiting” cells unnecessarily

Objects and references

The landscape grid is a matrix
Instead of a number, each entry in the matrix contains a **reference** to an **object** (i.e., a cell)

Object: grouping of related members (data and functions).
Members define the object's state and behavior.

```
Class Cell:
  int x,y
  bool burned
```

In R, use the $ operator to access named members:

```
model <- lm(rnorm(100)~rnorm(100))
model$coefficients
```

In Python, use the . operator:

```
a = list()
a.append(5)
```

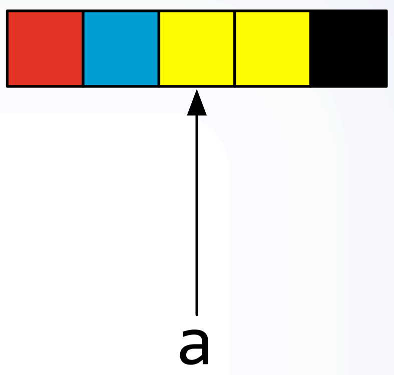
Objects and references

A **reference** to an object tells the computer where in memory the object is located

If the variable *a* is a reference to some object, accessing the object is called **dereferencing** a.

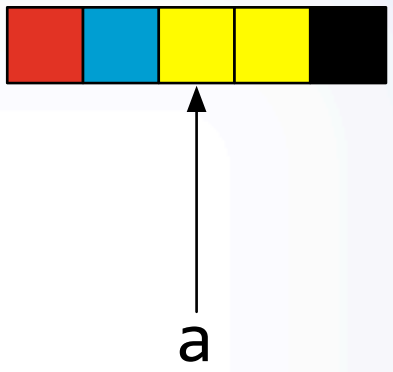
References

```
a = ['red', 'blue', 'yellow', 'yellow', 'black']
```



References

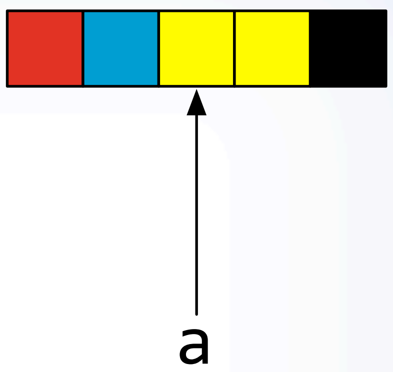
```
a = ['red', 'blue', 'yellow', 'yellow', 'black']
```




```
b = a
```

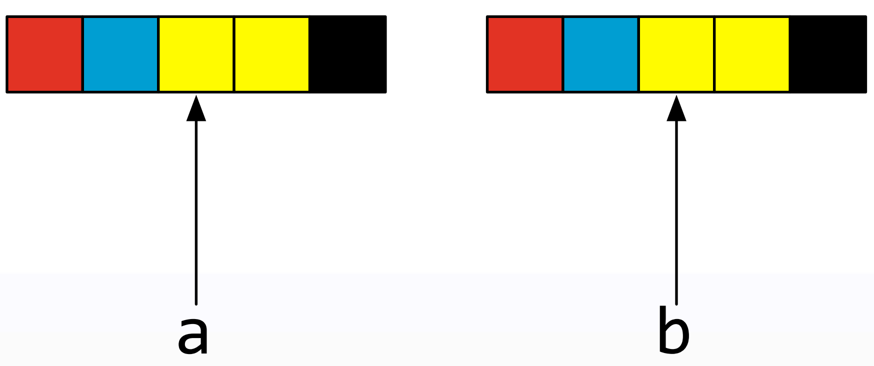
References

```
a = ['red', 'blue', 'yellow', 'yellow', 'black']
```



copy the **object**

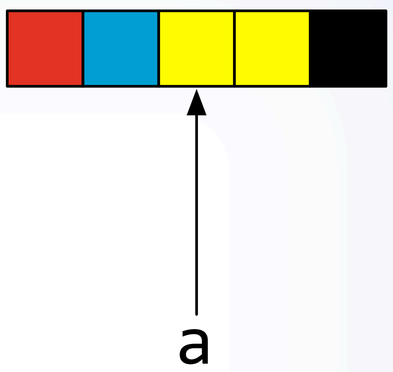
```
b = a
```



a and b are references
to different objects

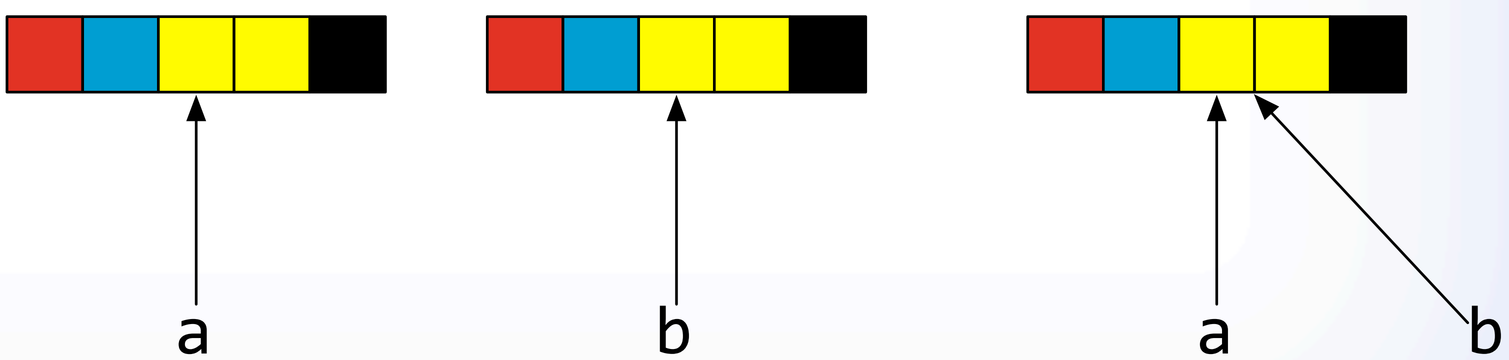
References

```
a = ['red', 'blue', 'yellow', 'yellow', 'black']
```



copy the **object** | copy the **reference**

```
b = a
```

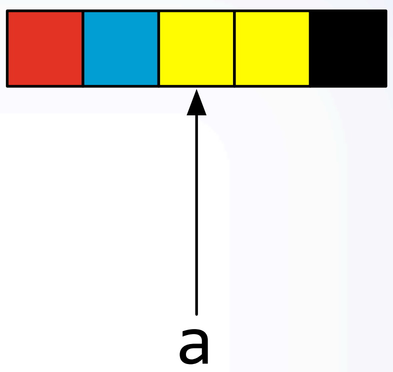


a and b are references to different objects

a and b refer to the same object

References

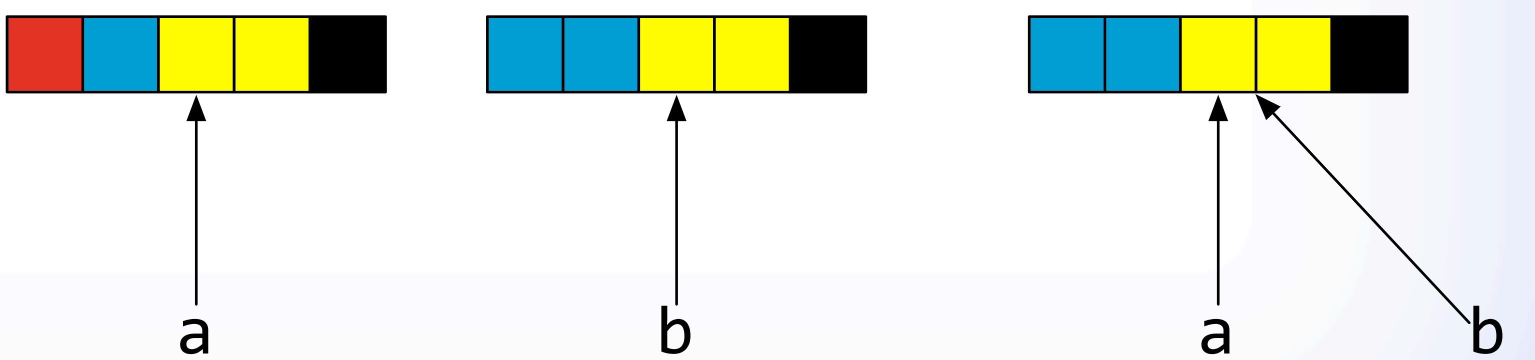
```
a = ['red', 'blue', 'yellow', 'yellow', 'black']
```



copy the **object**

copy the **reference**

```
b[0] = 'blue'
```

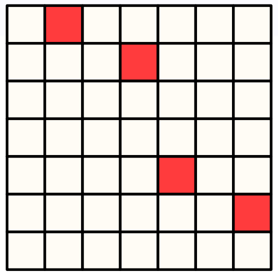


changing b has no effect on a

changing b also changes a

References

```
a = matrix(Cell(), nrow=7, ncol=7)

b = [ a[0,1], a[1,3], a[4,4], a[5,6] ]
```



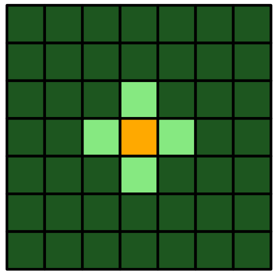
To avoid looping, use functions like `sample()` to choose references to the objects we want

Use lists of references to keep track of the cells of interest

Can modify the original cells via these shorter lists

Identify neighbors

```
function find_neighbors(list all_cells, int xmax, int ymax):
  for x in 0:xmax:
    for y in 0:ymax:
      focal_cell = all_cells[x][y]    # reference, not a copy!
      nblist = list()

      # copy cell references into the neighbor list
      if x > 0:     nblist.append(all_cells[x-1][y])
      if x < xmax:  nblist.append(all_cells[x+1][y])
      if y > 0:     nblist.append(all_cells[x][y-1])
      if y < ymax:  nblist.append(all_cells[x][y+1])

      focal_cell.neighbors = nblist
```



- do this once at initialization
- every cell has its own list of references to neighboring cells
- only loop through all cells once

Choose fire parameters

```
# choose number of fires from zero-inflated Poisson
num_fires = rzip(1, pi, lambda)

# choose fire sizes from exponential distribution
fire_sizes = rexp(num_fires, 1.0 / mean_fire_size)

# select starting cells
starting_cells = sample(num_fires, all_cells)

# run the fire model for each starting cell
for i in 0:num_fires:
  Fire(all_cells, starting_cells[i], fire_sizes[i])
```

$$\mathrm{pr}(X=0) = \pi + (1-\pi) \times \mathrm{dpois}(0, \lambda)$$

$$\mathrm{pr}(X=k) = (1-\pi)\ \mathrm{dpois}(k, \lambda)$$

Spread randomly without directional bias

```
function Fire(starting_cell, target_size):
  burning = True
  fire_neighbors = list()          # list of all cells that are next to the fire
  current_size = 0
  current_cell = starting_cell

  while burning:
    current_cell.burned = True
    current_size = current_size + 1

    # find new neighbors to burn
    for potential_neighbor in current_cell.neighbors:
      if not potential_neighbor.burned:
        fire_neighbors.append(potential_neighbor)

    if current_size >= target_size or fire_neighbors is empty:
      # fire burns out when it gets to the target size or runs out of neighbors
      burning = False
    else:
      # otherwise fire spreads to a new cell
      current_cell = sample(1, neighbor_cells)    # using sample() avoids spatial bias
```