

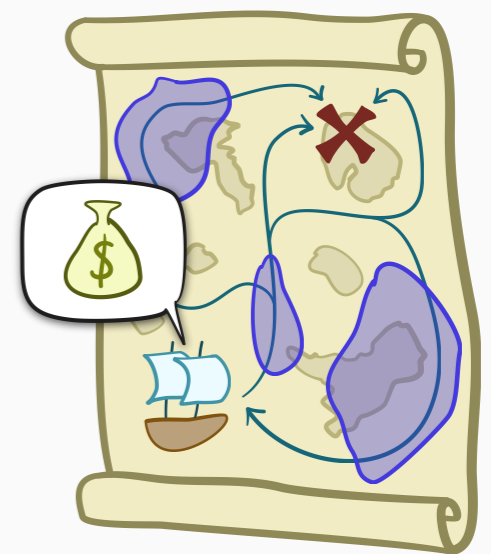
The Gittins Policy

in the $M/G/1$ Queue

Ziv Scully

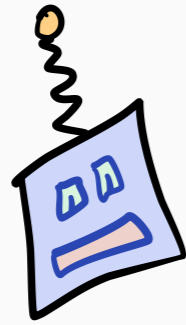
Mor Harchol-Balter

Carnegie Mellon University



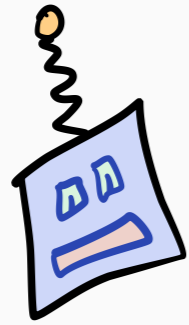
In a Nutshell

In a Nutshell



The *Gittins policy* solves a variety of queue scheduling problems

In a Nutshell

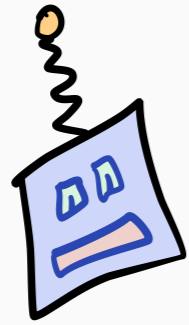


The *Gittins policy* solves a variety of queue scheduling problems



But prior results have limitations

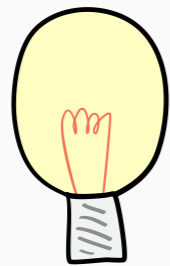
In a Nutshell



The *Gittins policy* solves a variety of queue scheduling problems

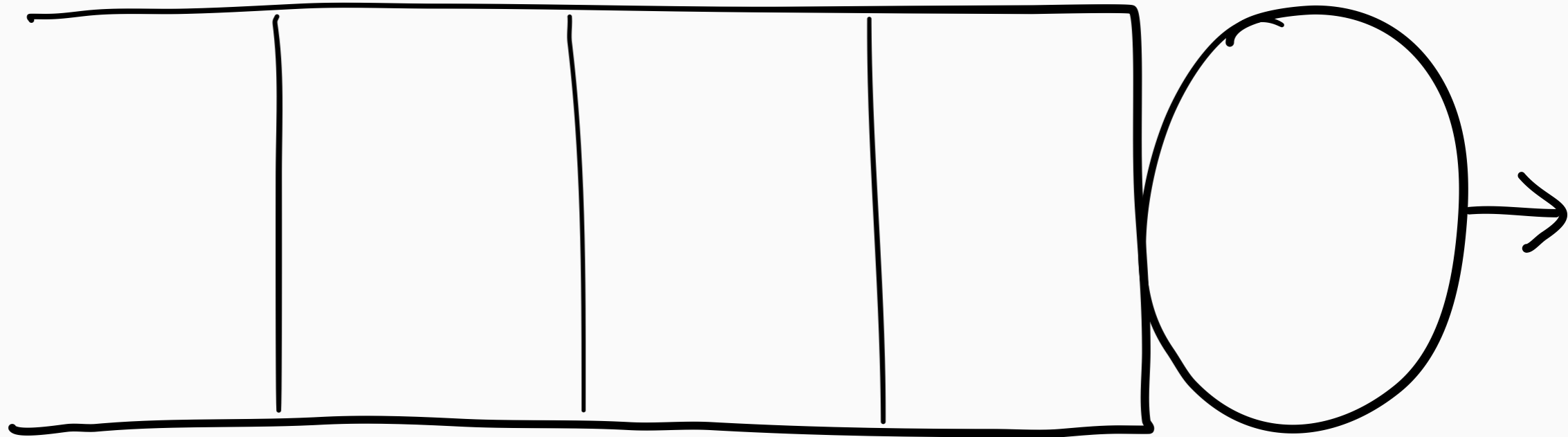


But prior results have limitations

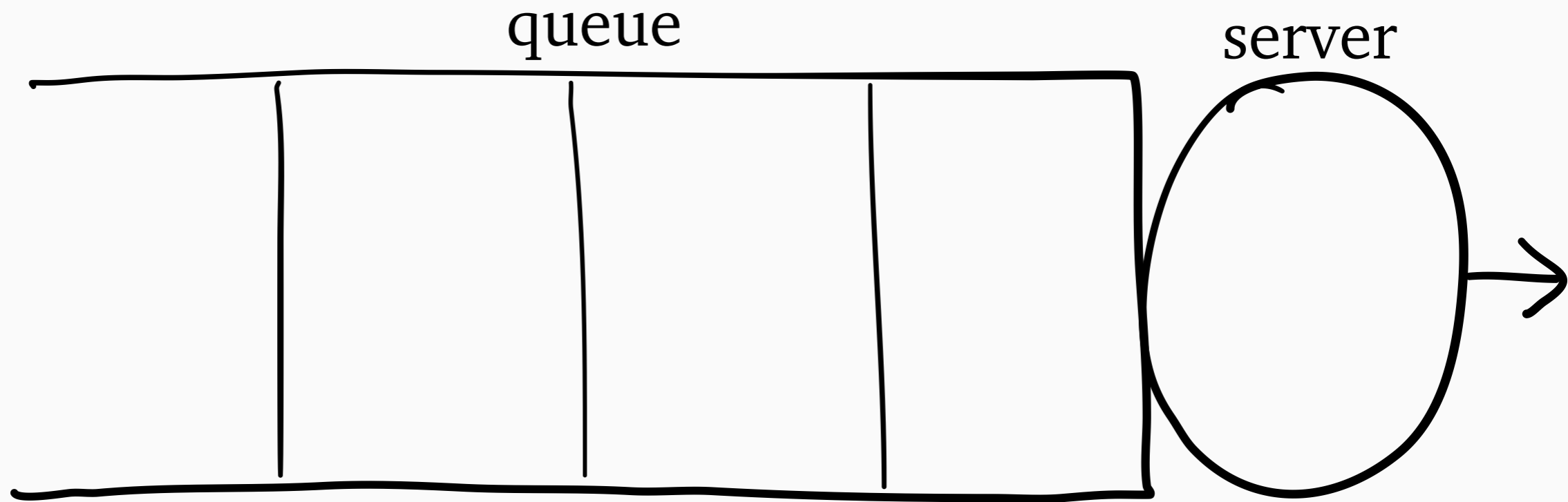


We **unify** and **generalize** all prior Gittins optimality results

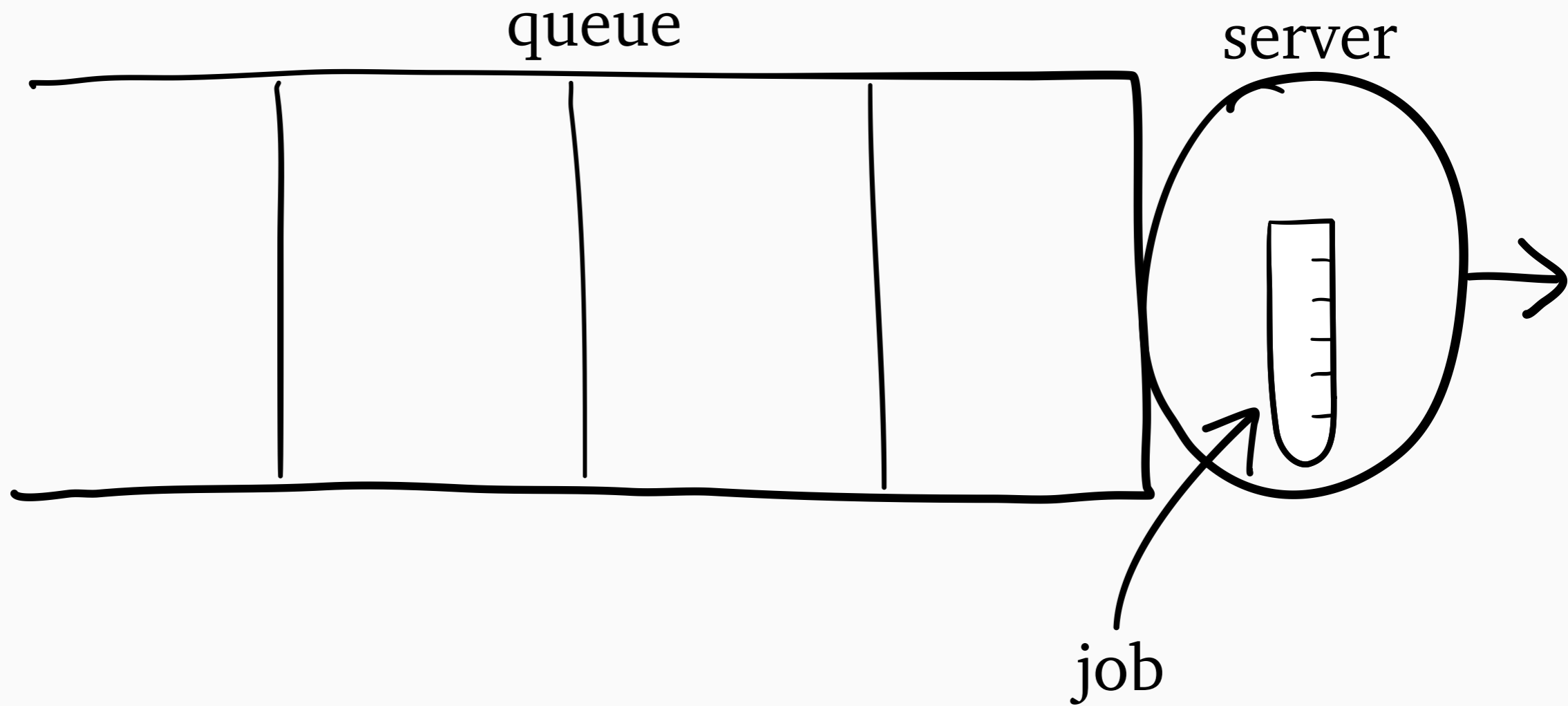
M/G/1 Queue



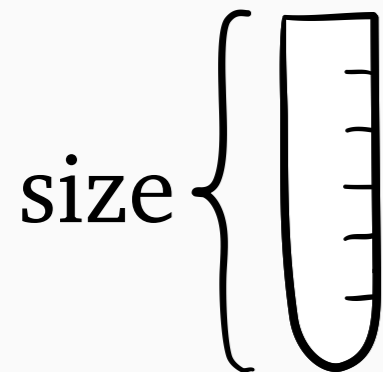
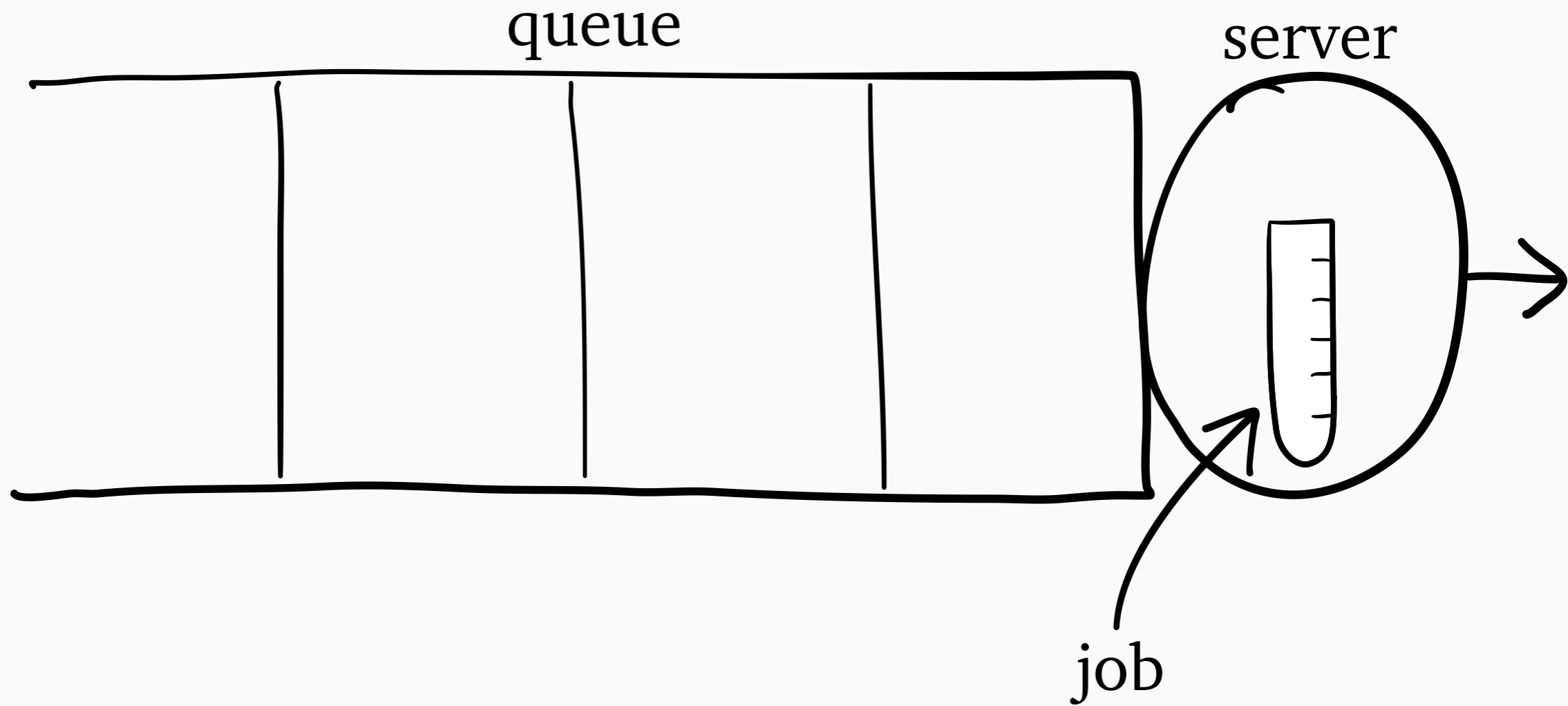
M/G/1 Queue



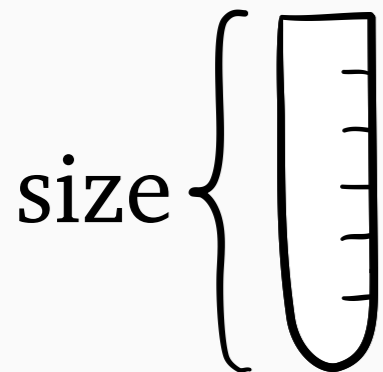
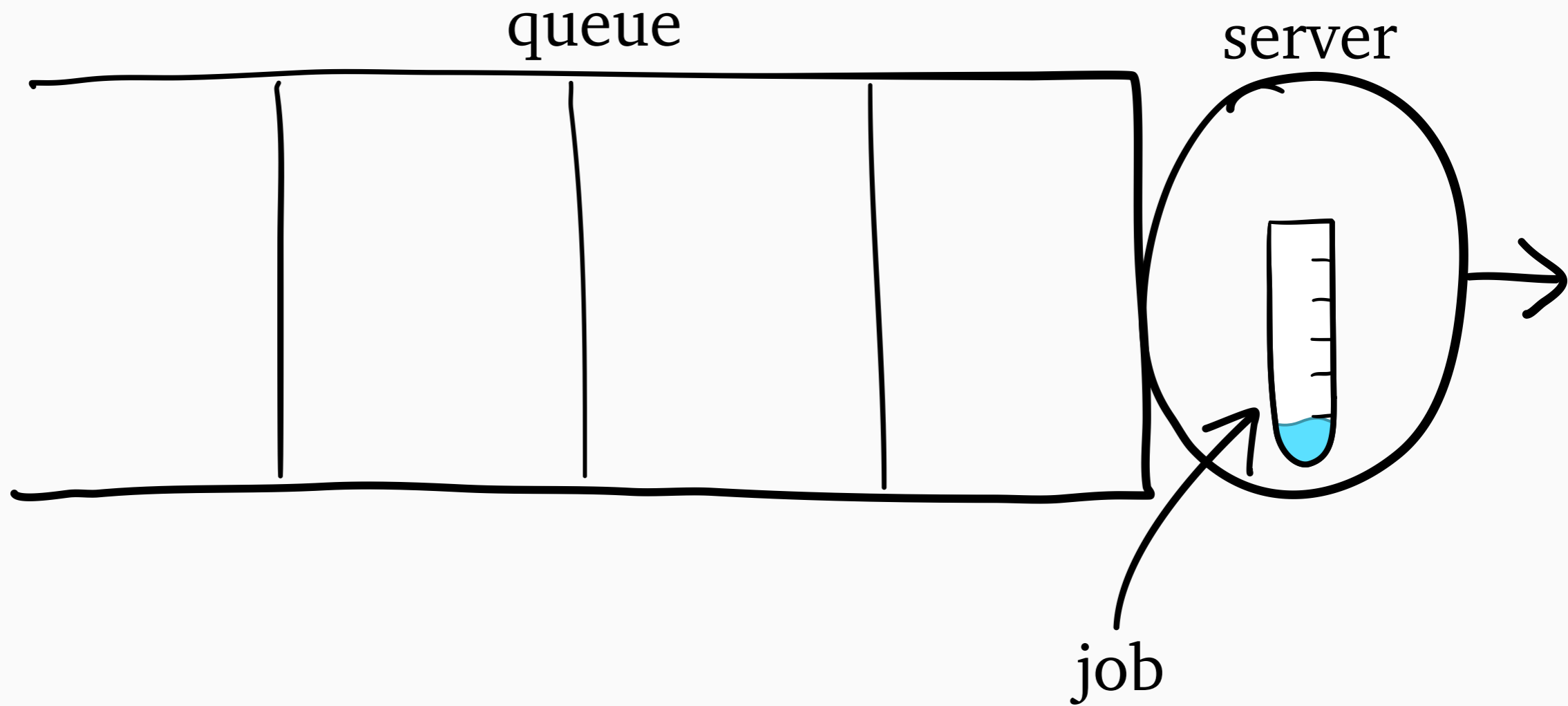
M/G/1 Queue



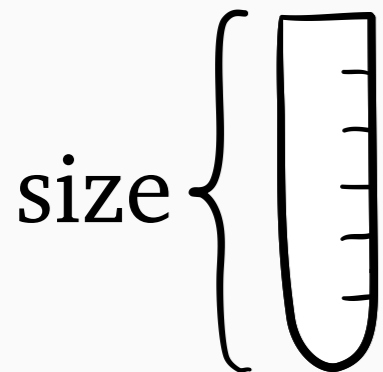
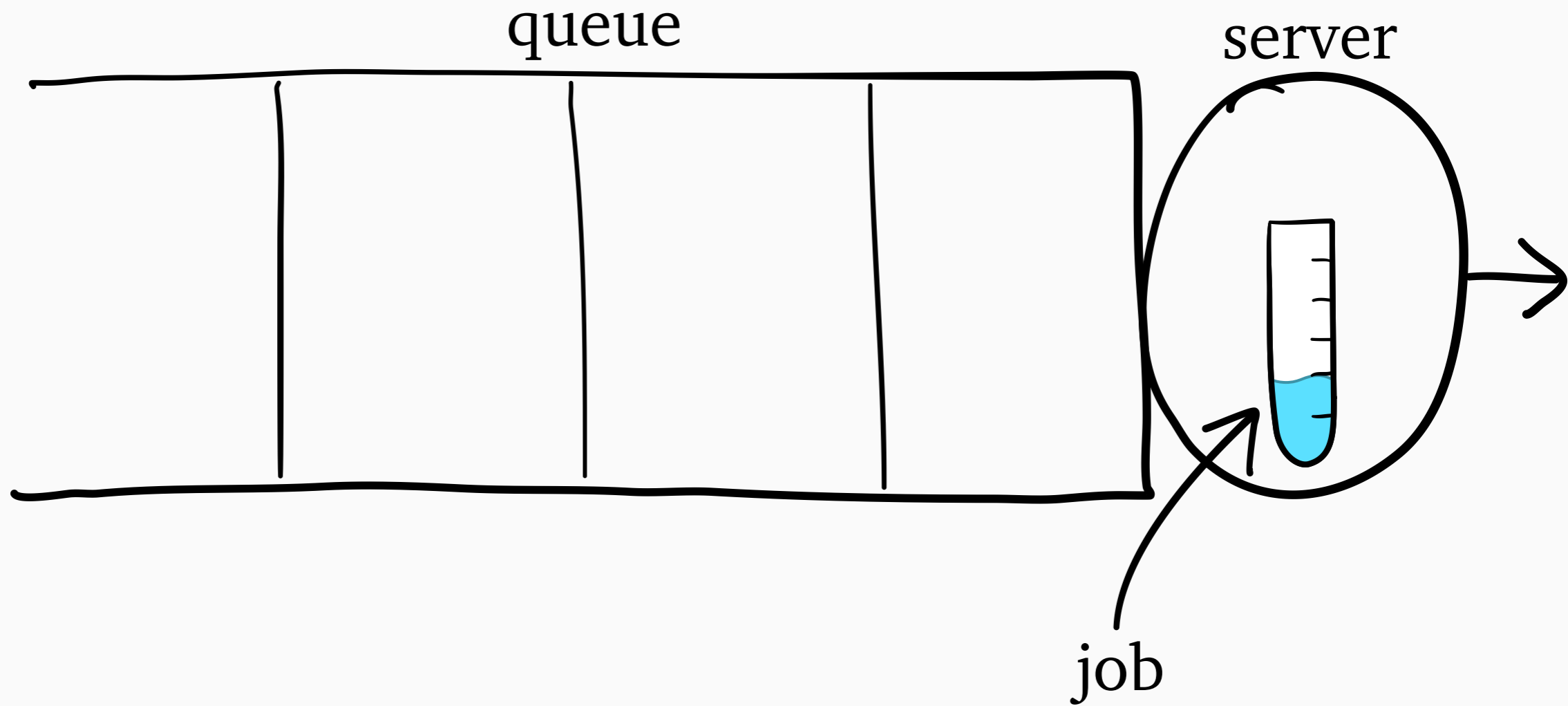
M/G/1 Queue



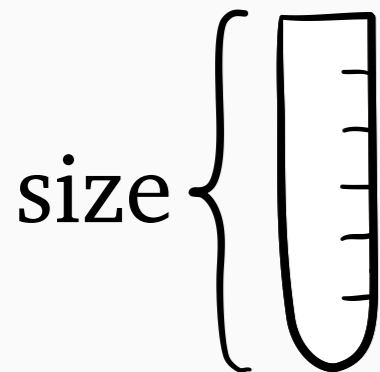
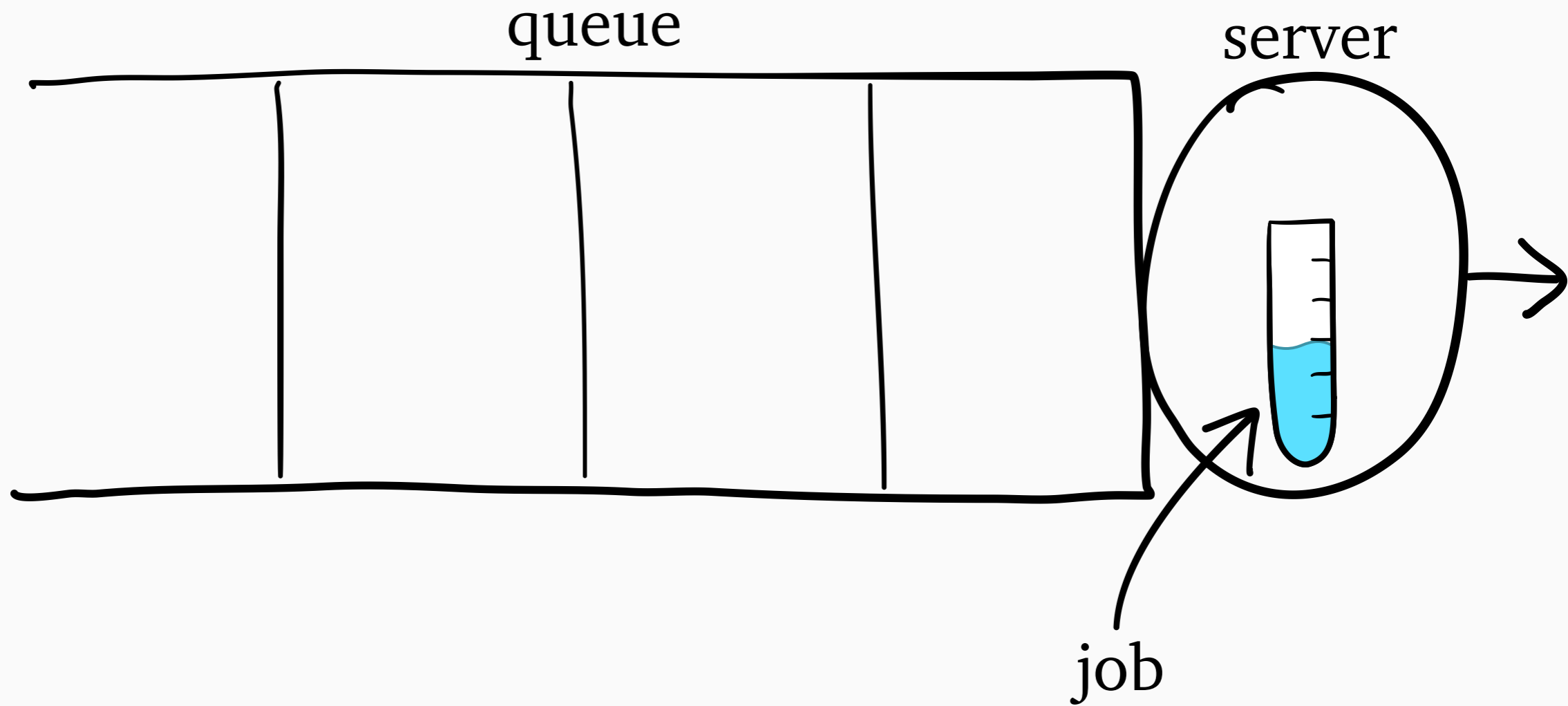
M/G/1 Queue



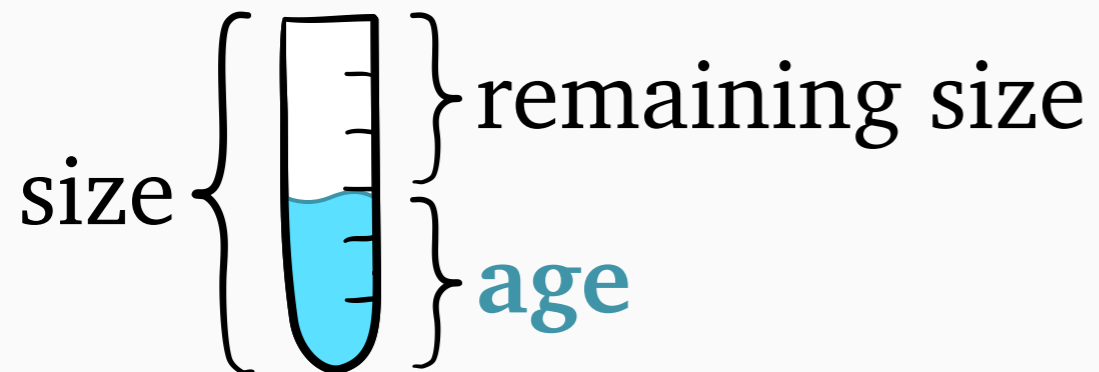
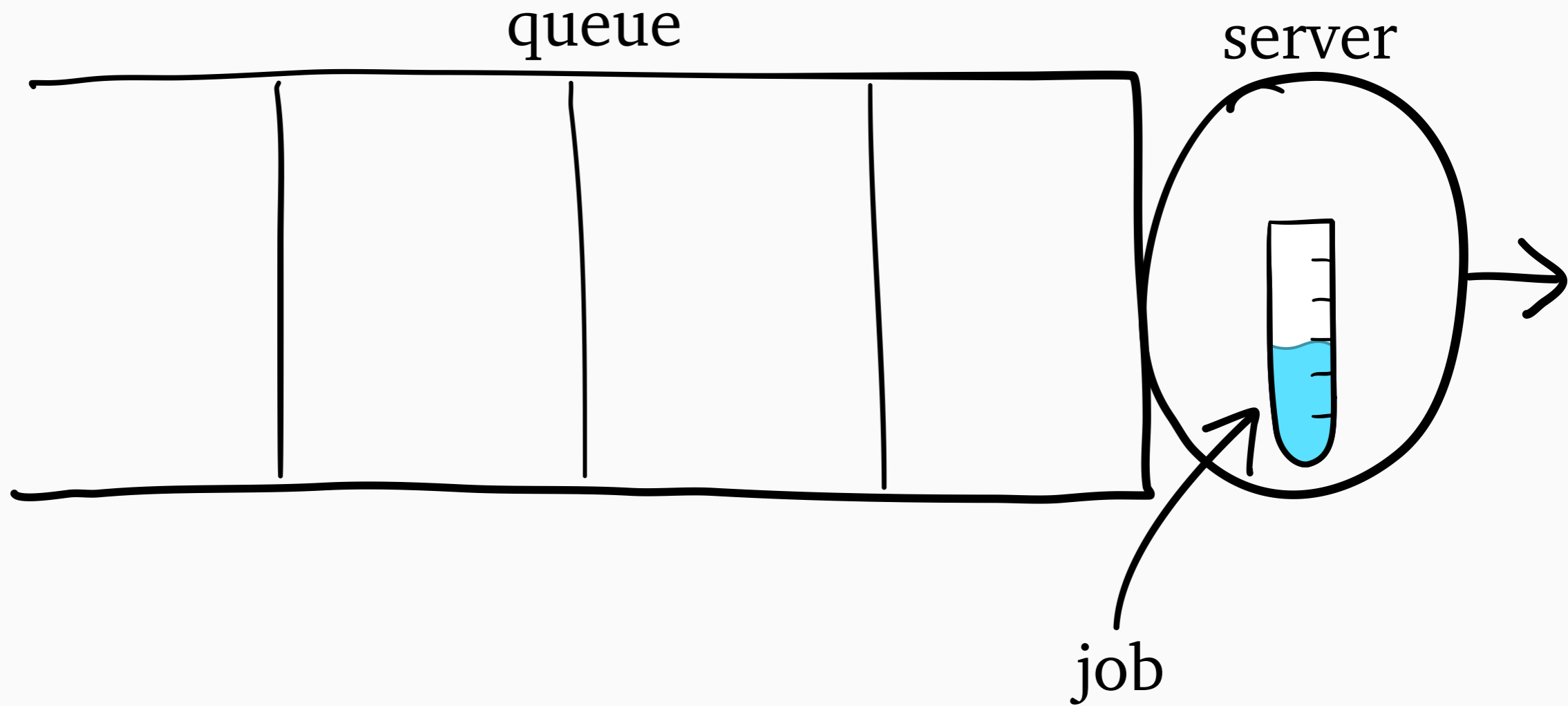
M/G/1 Queue



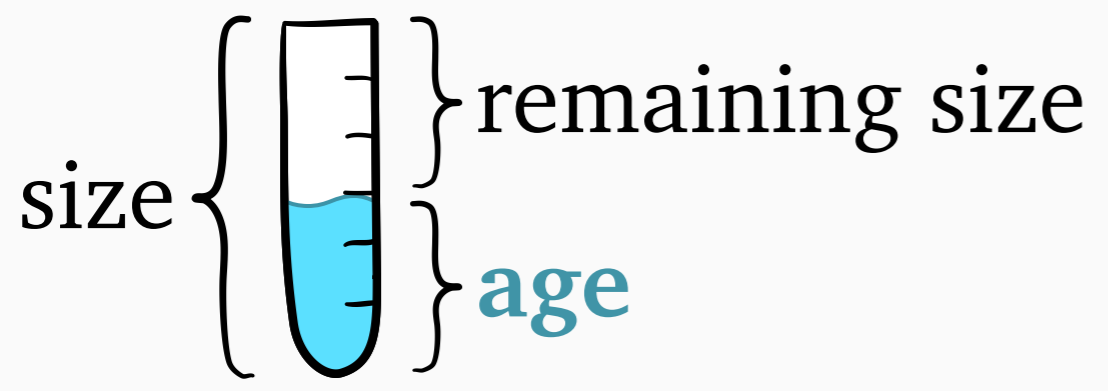
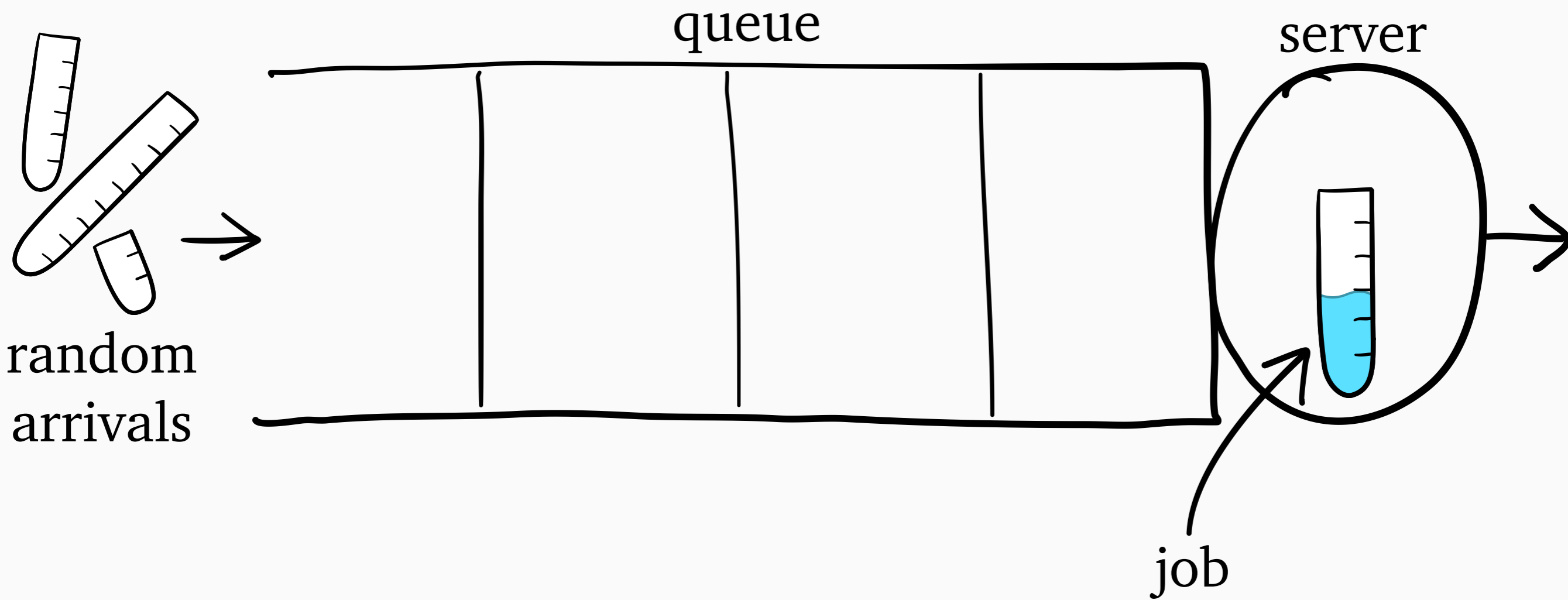
M/G/1 Queue



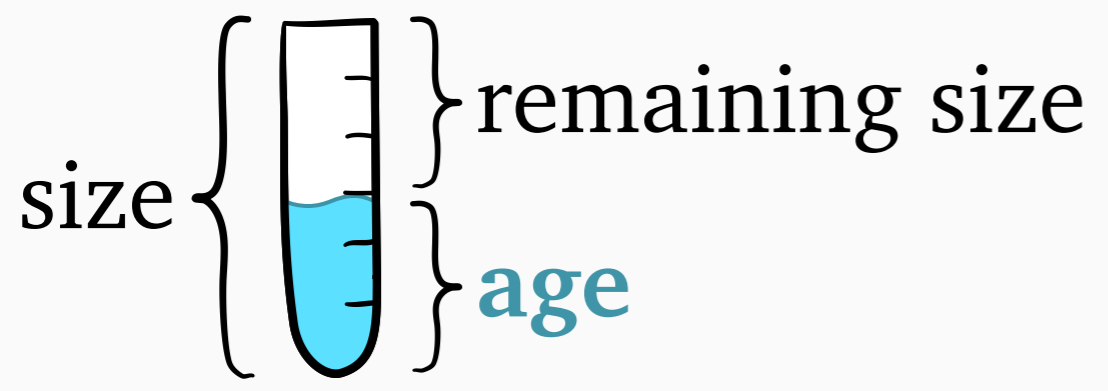
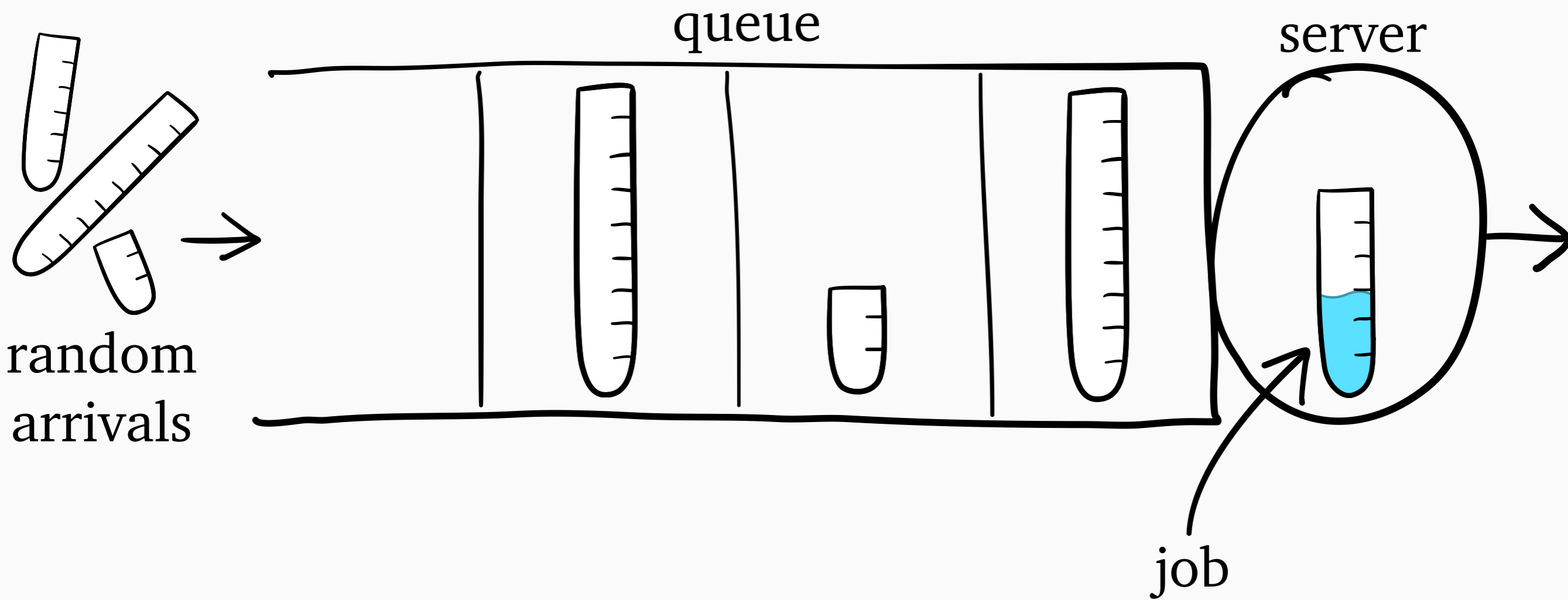
M/G/1 Queue



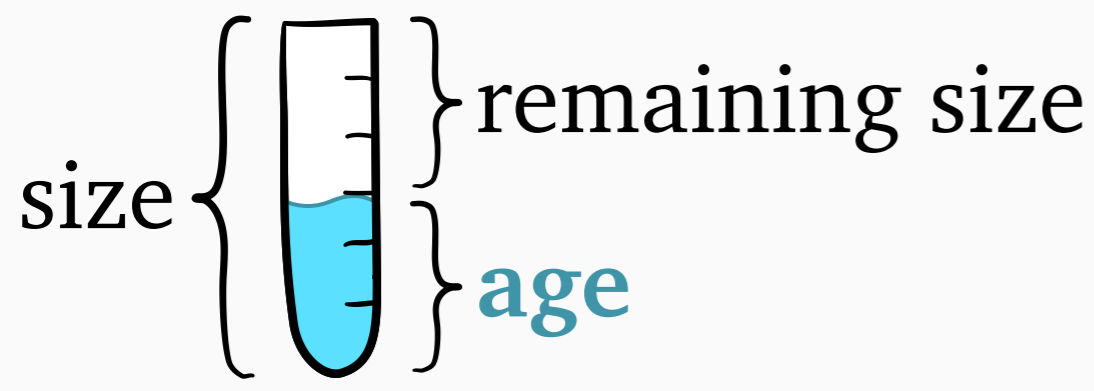
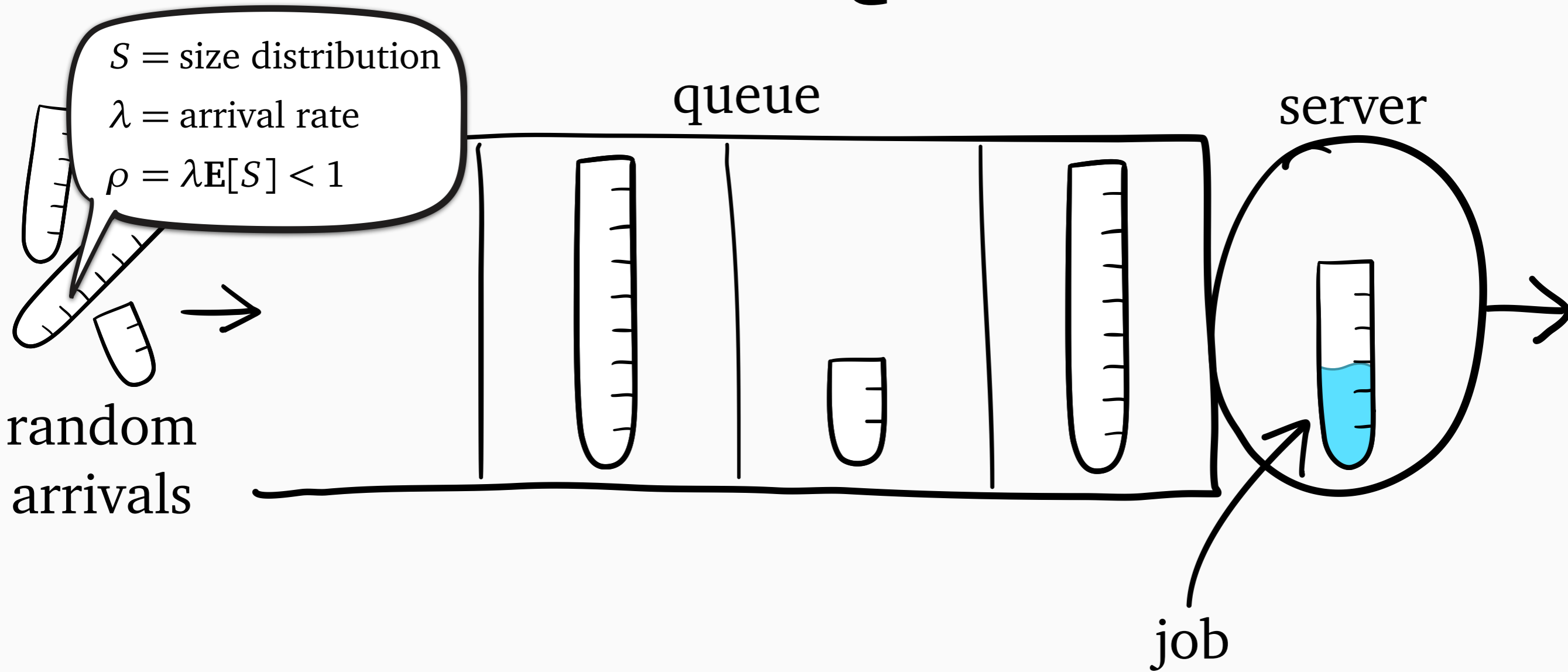
M/G/1 Queue



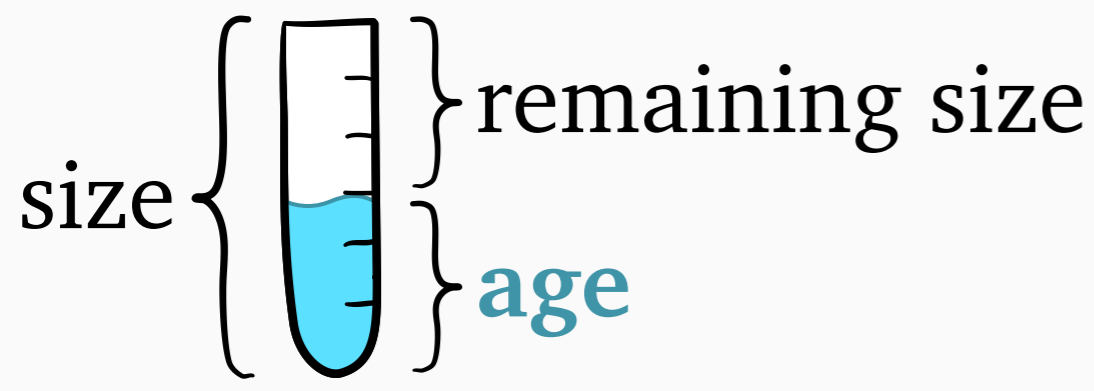
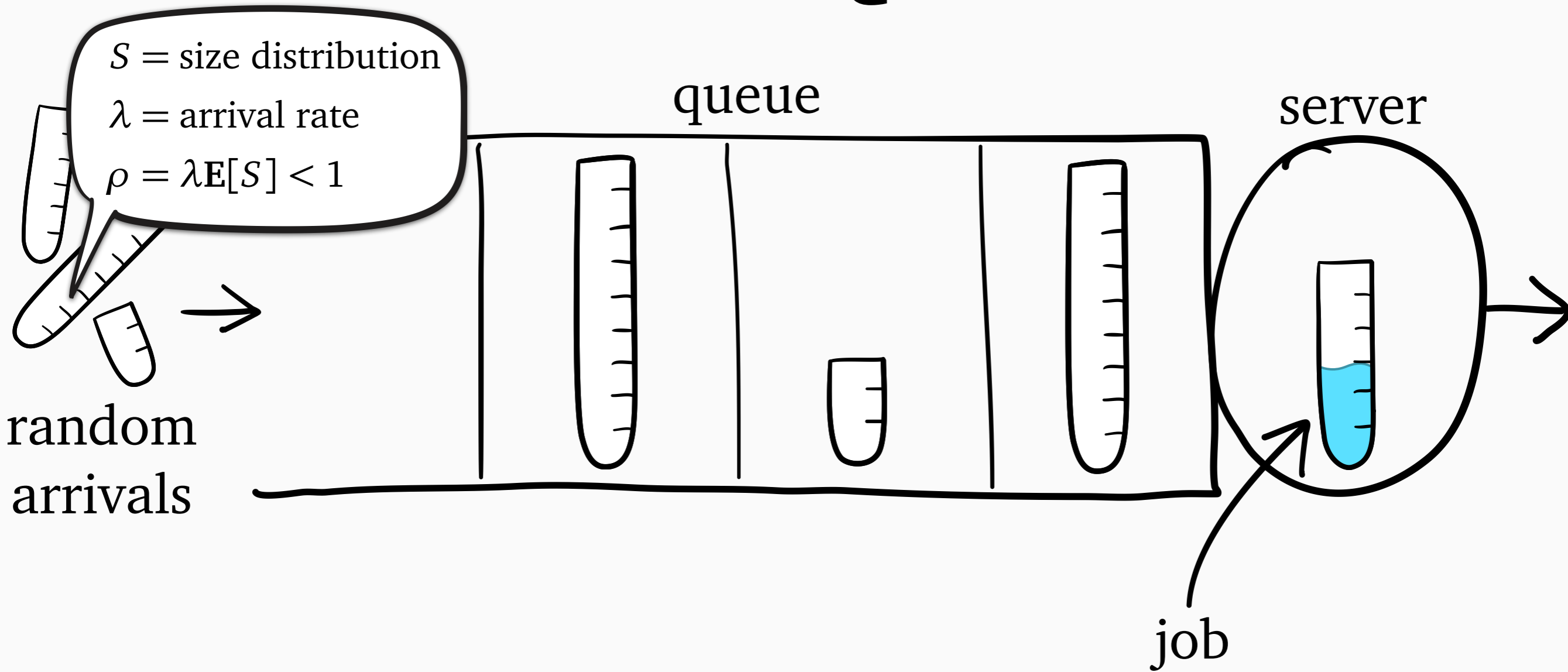
M/G/1 Queue



M/G/1 Queue

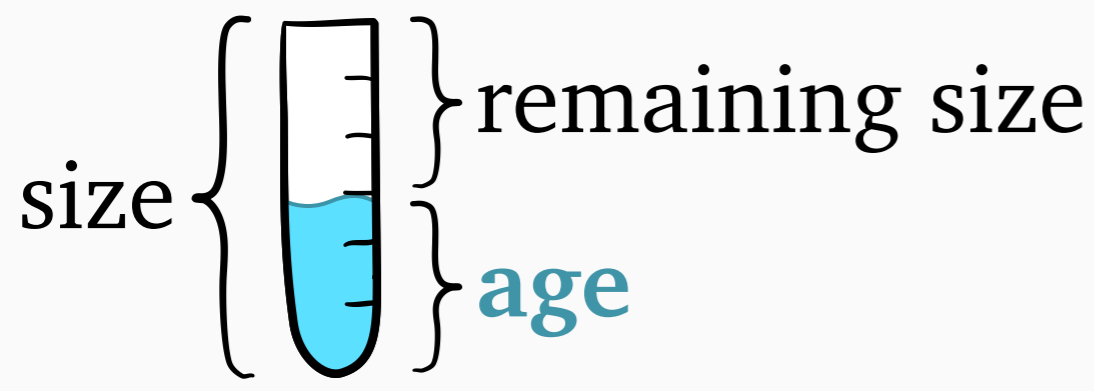
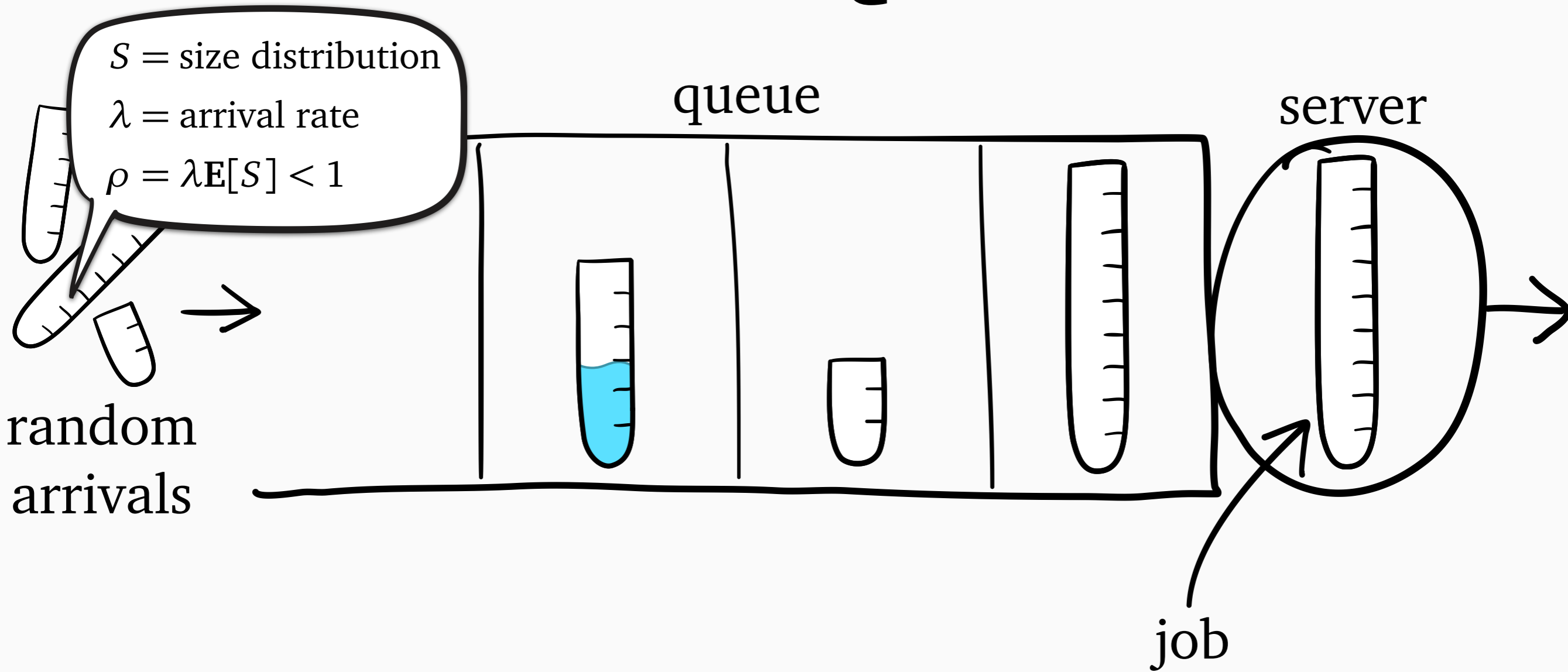


M/G/1 Queue



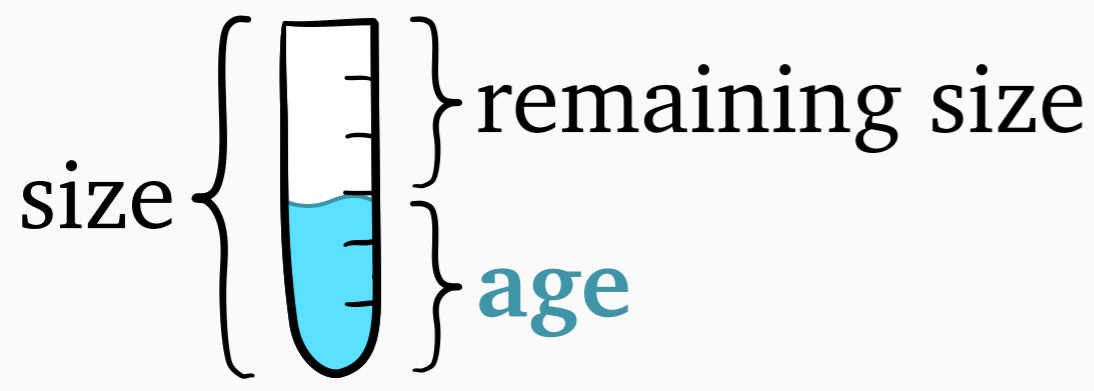
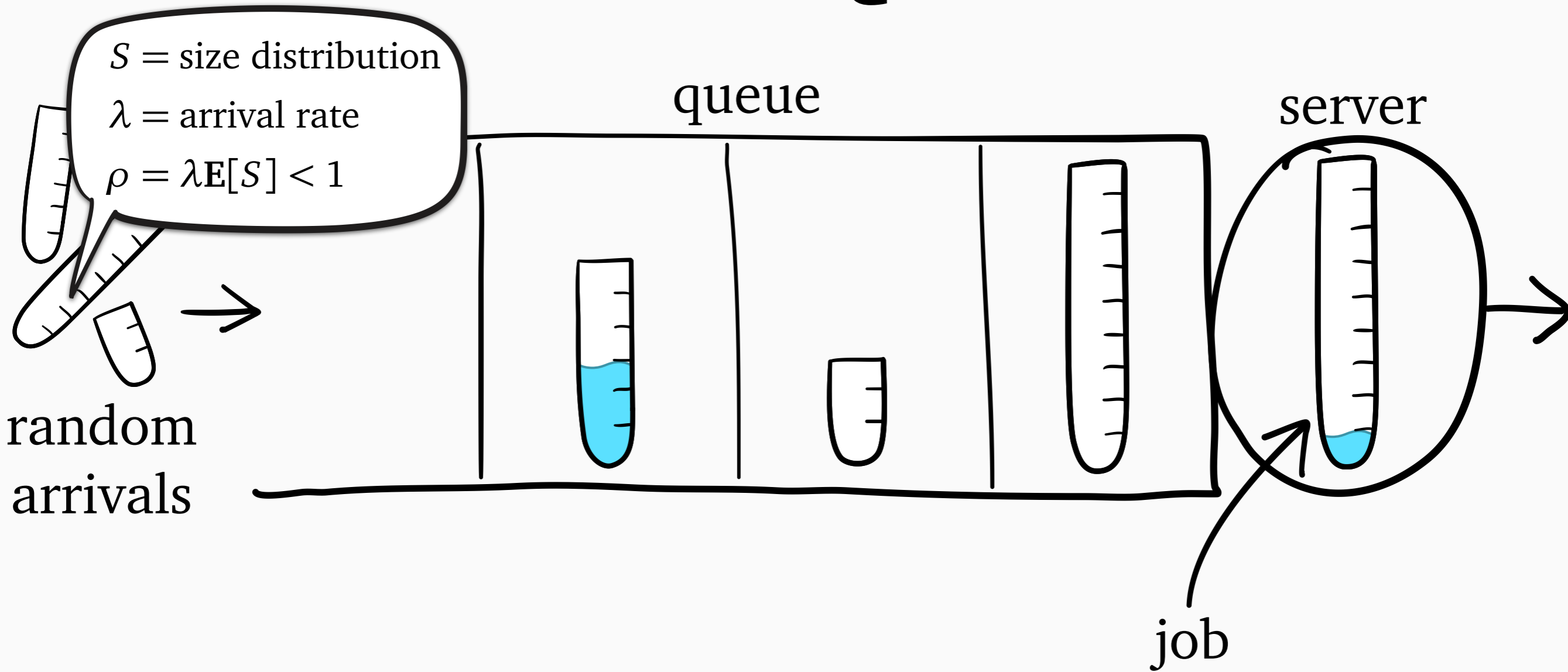
 **Scheduling policy:**
picks which job to serve

M/G/1 Queue



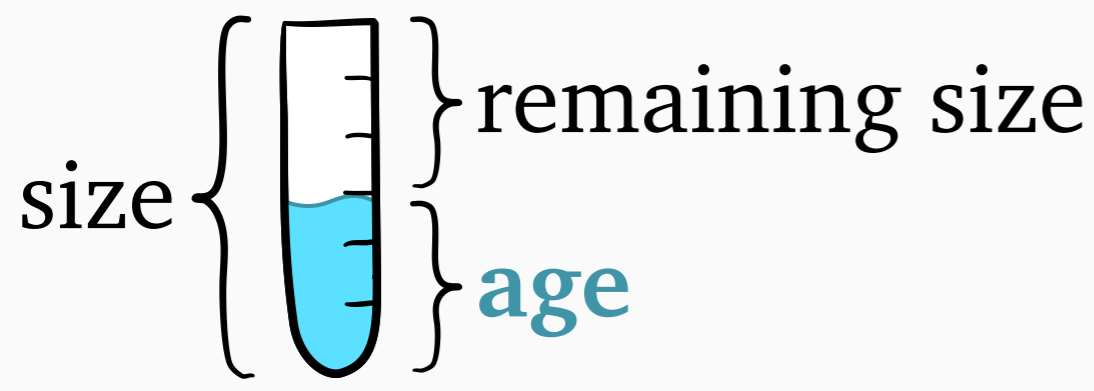
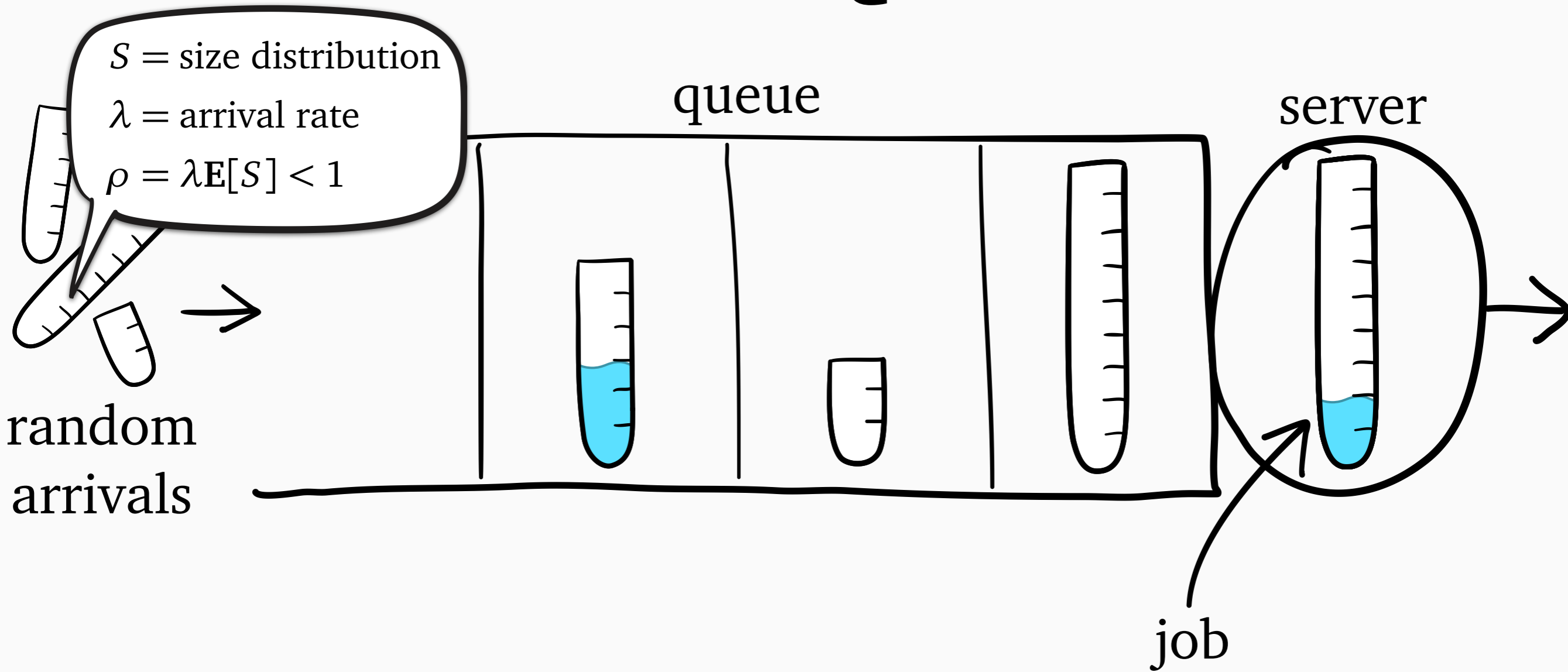
 **Scheduling policy:**
picks which job to serve

M/G/1 Queue



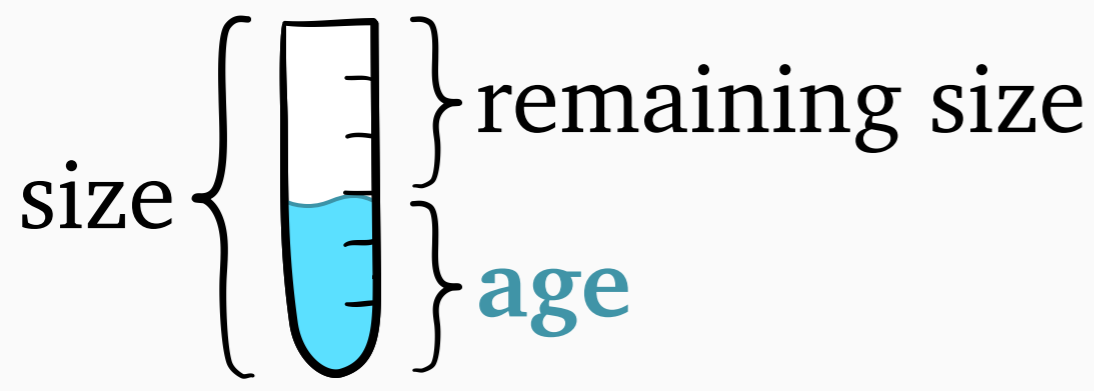
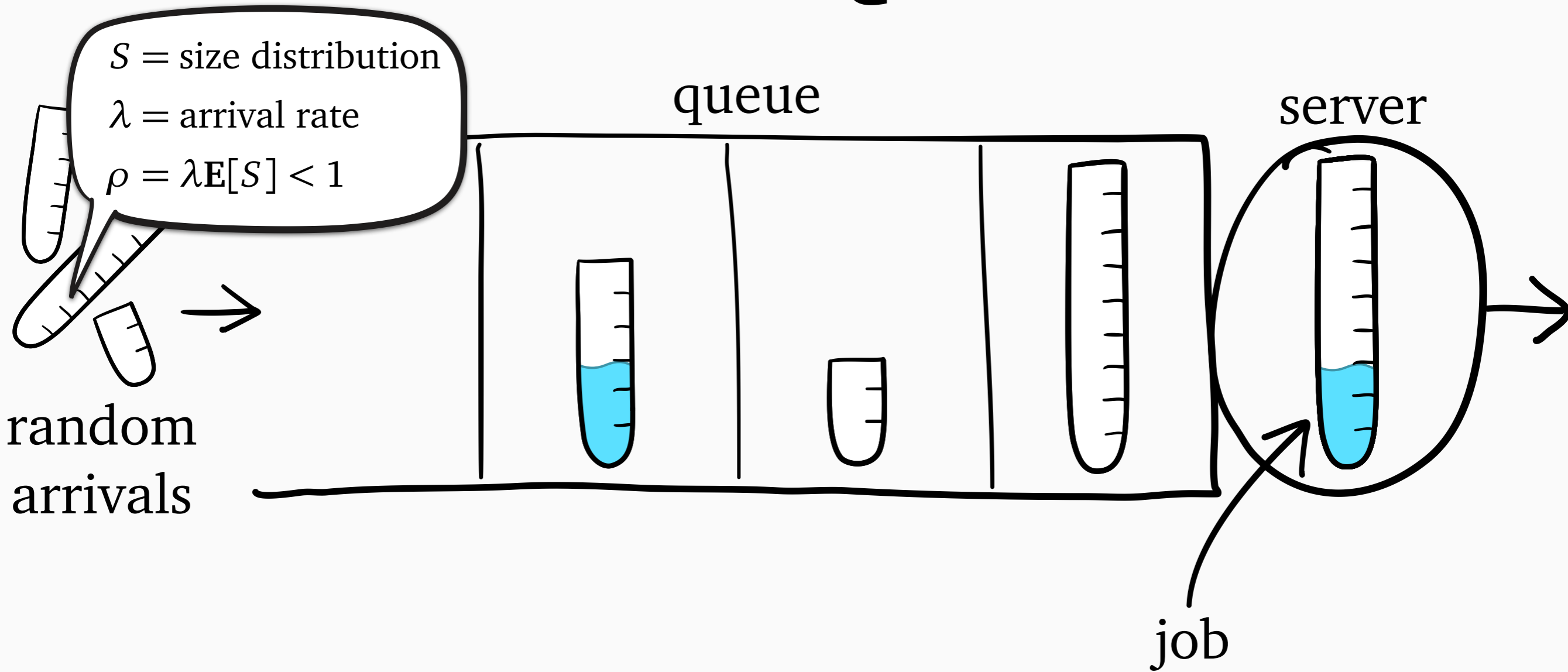
 **Scheduling policy:**
picks which job to serve

M/G/1 Queue



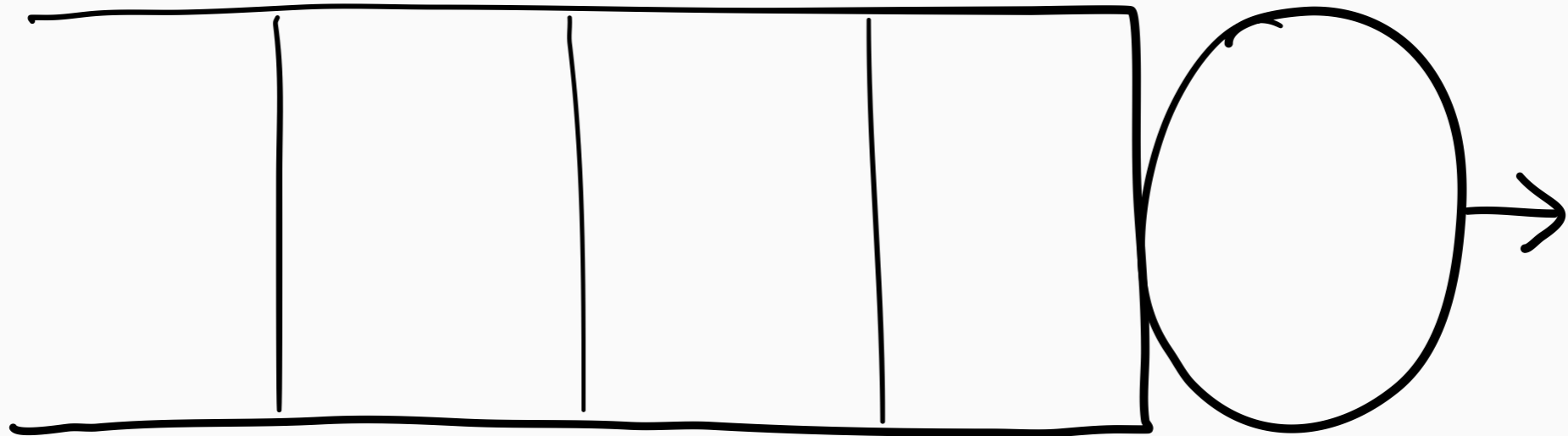
 **Scheduling policy:**
picks which job to serve

M/G/1 Queue

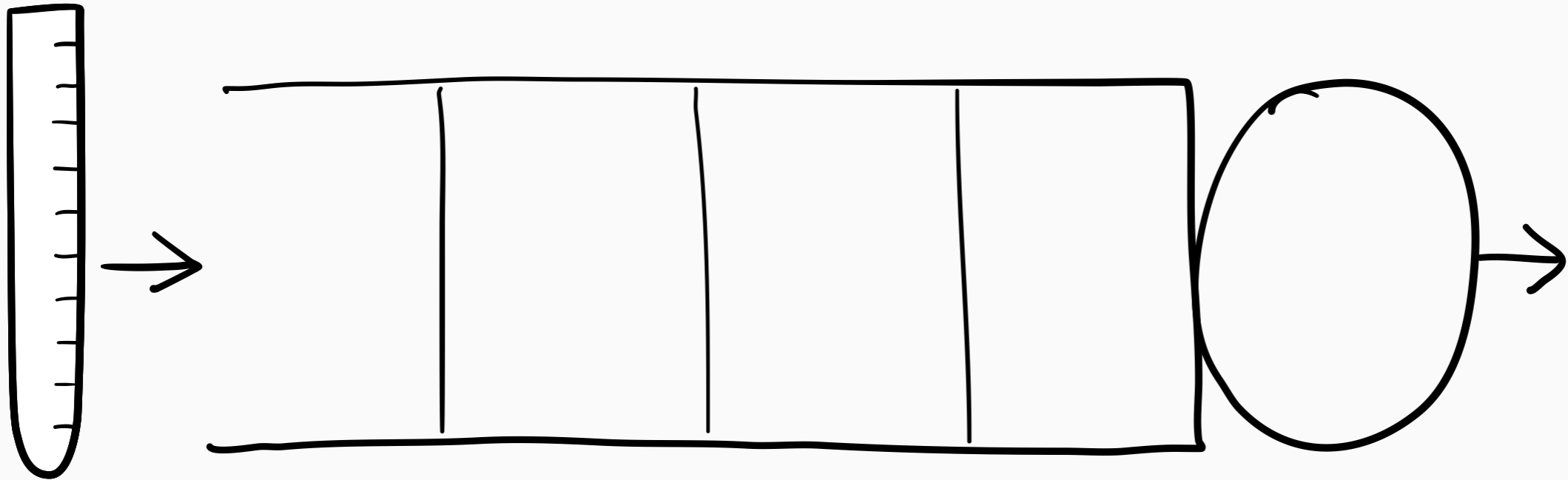


 **Scheduling policy:**
picks which job to serve

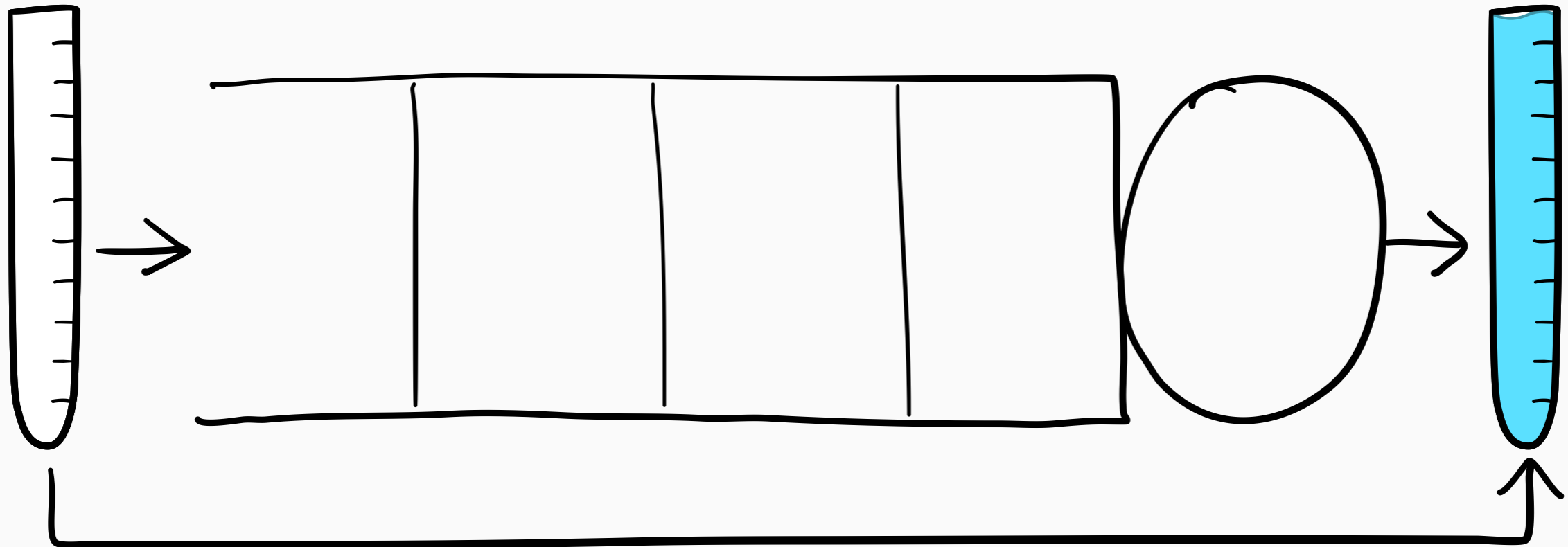
Response Time




Response Time

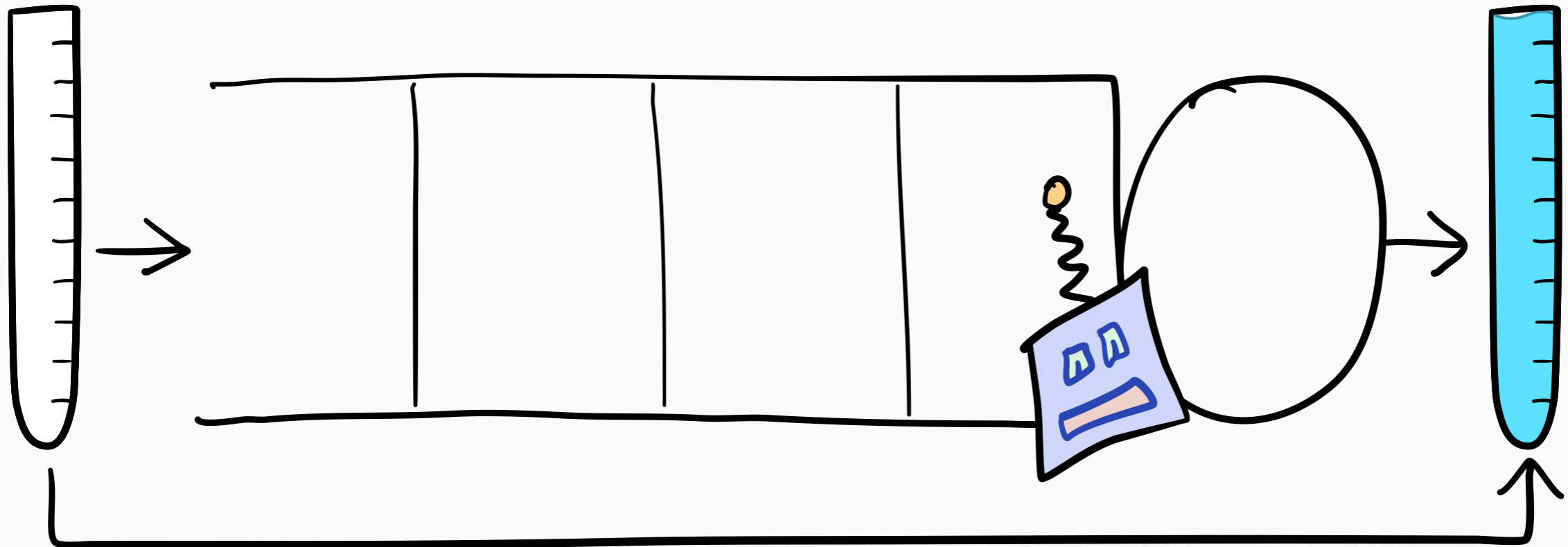



Response Time



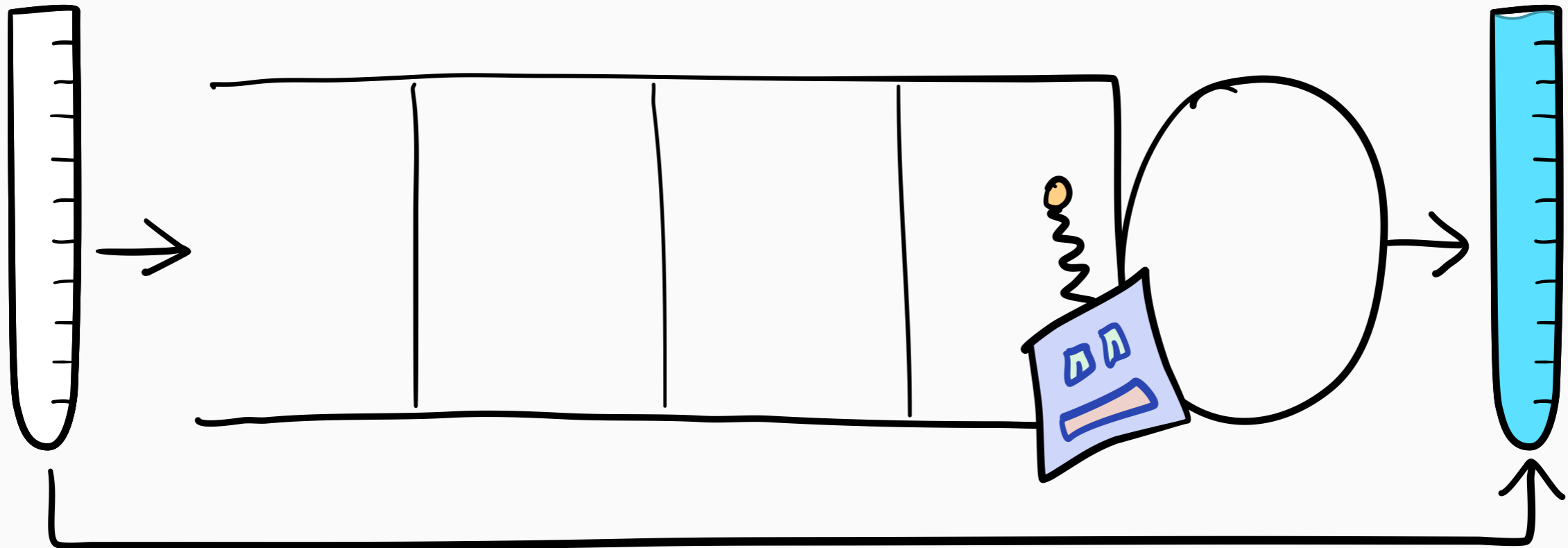
 = T = response time


Response Time



 = T = response time

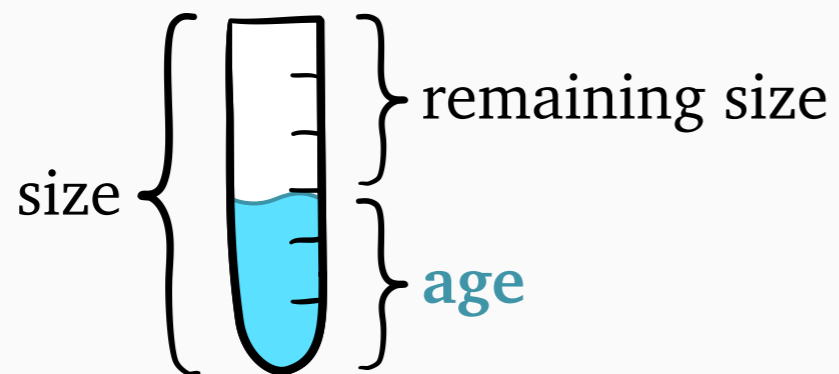
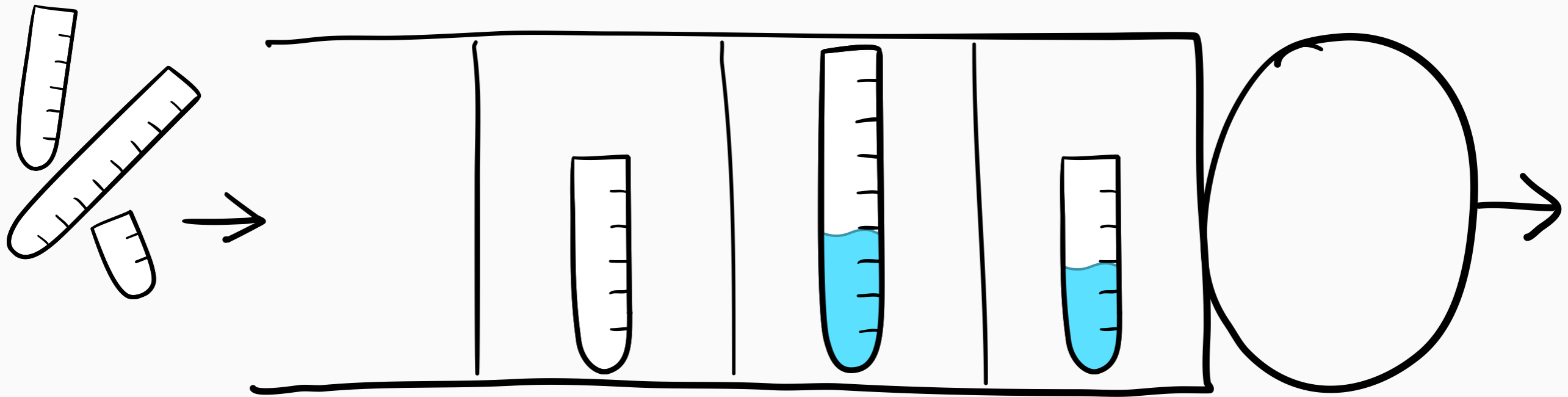
Response Time



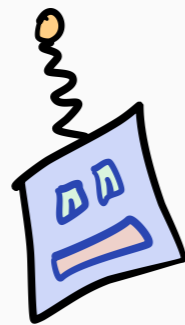
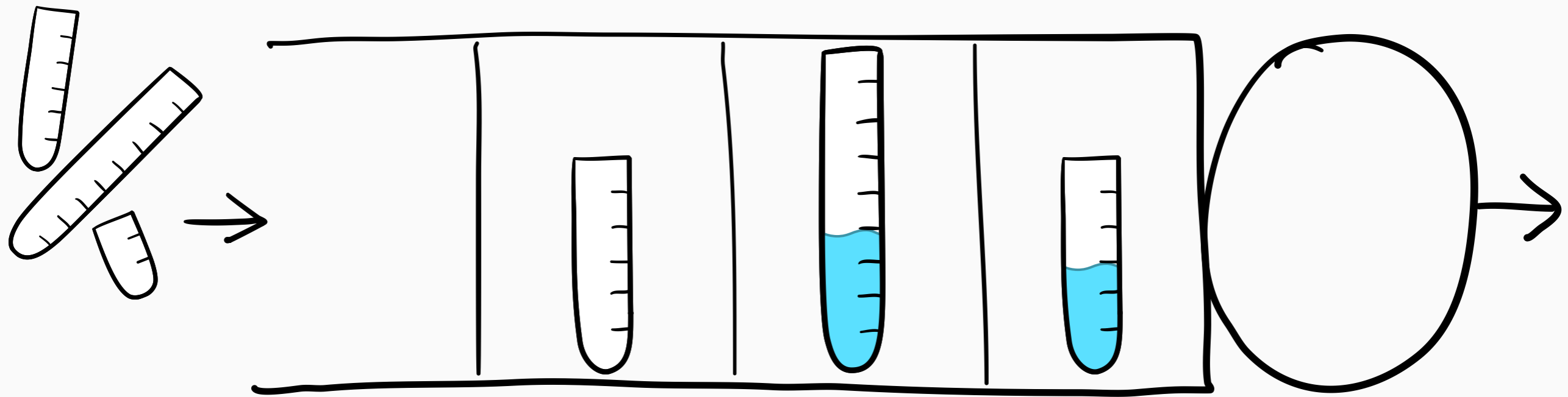
 = T = response time

Goal: schedule to minimize metrics like *mean response time* $E[T]$

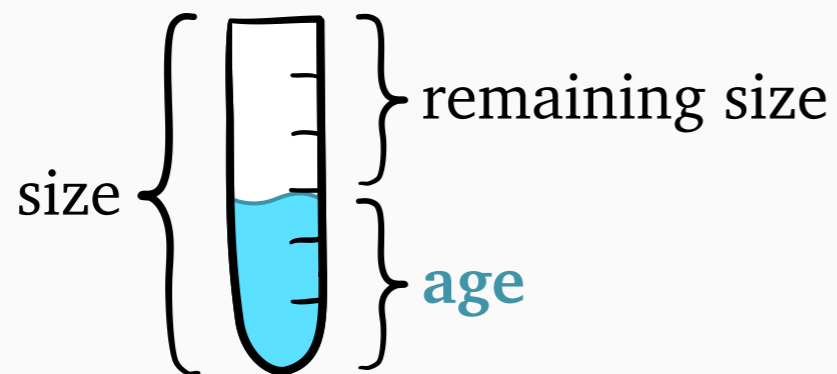
How to Schedule?



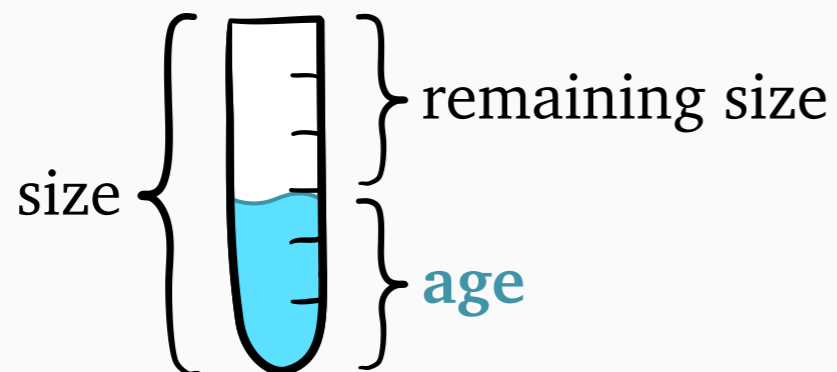
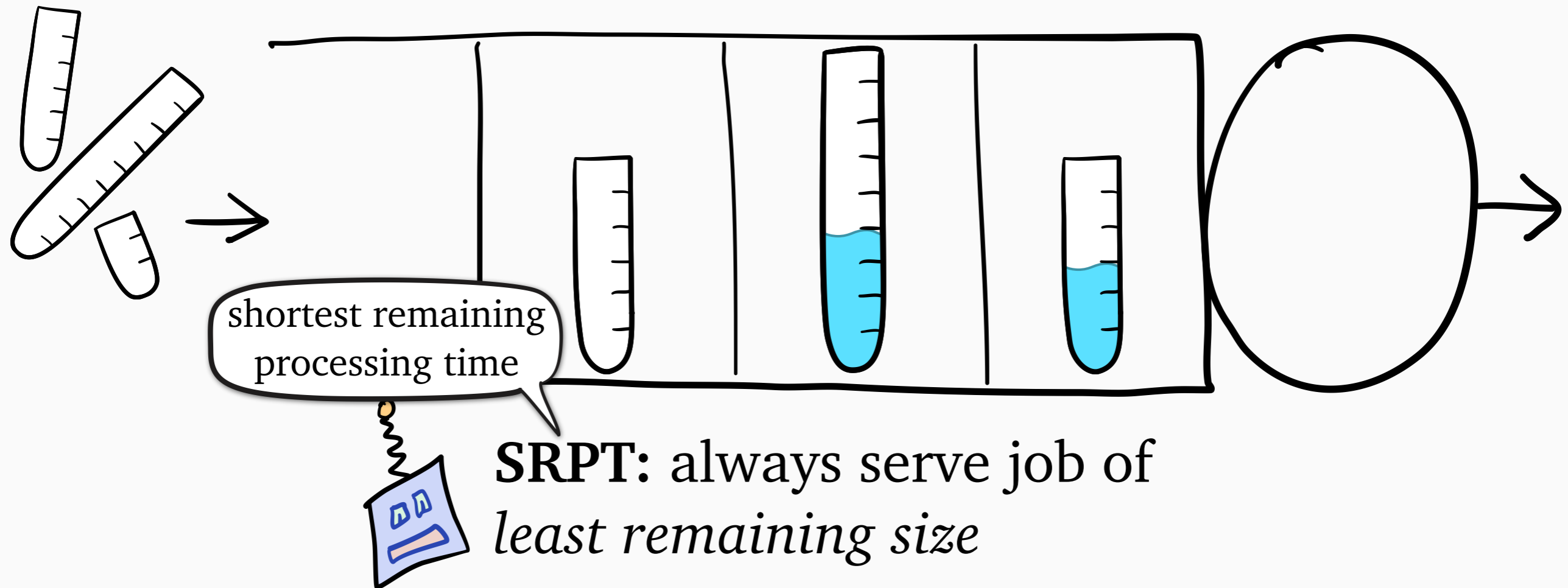
How to Schedule?



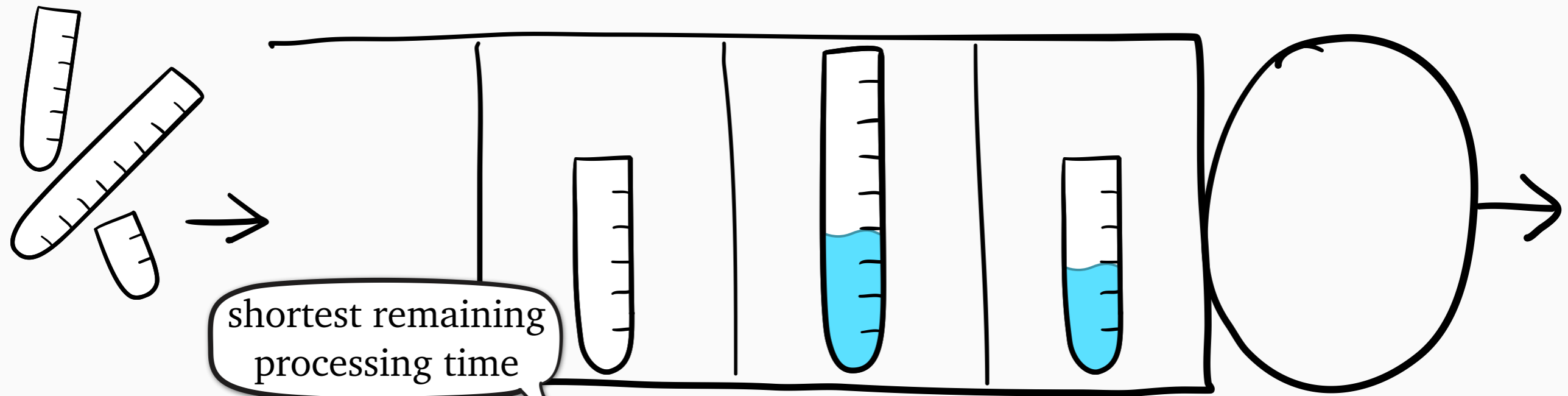
SRPT: always serve job of *least remaining size*



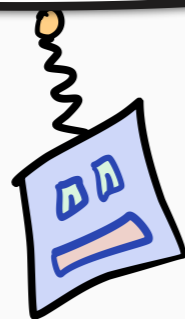
How to Schedule?



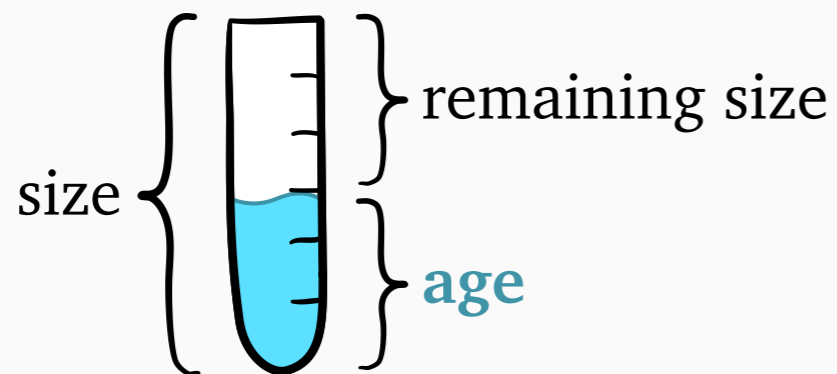
How to Schedule?



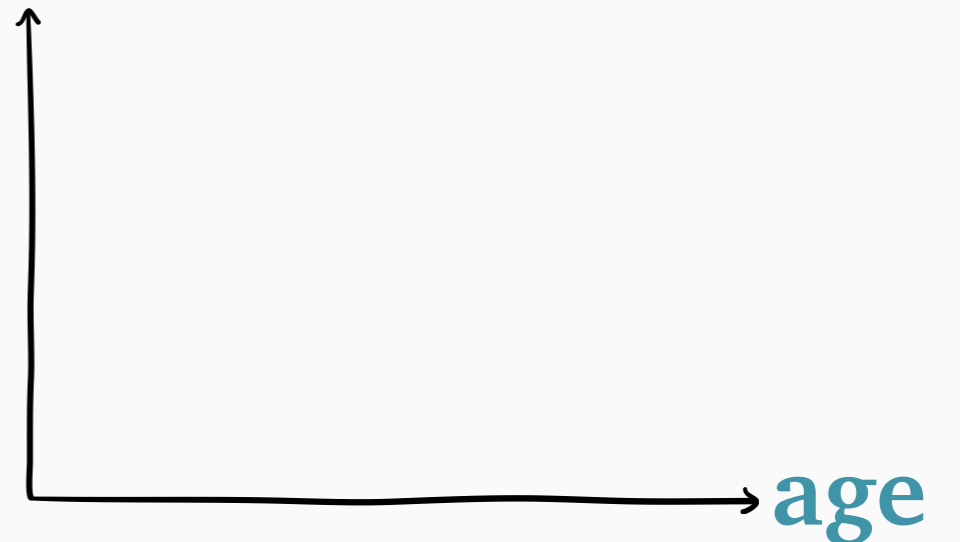
shortest remaining processing time



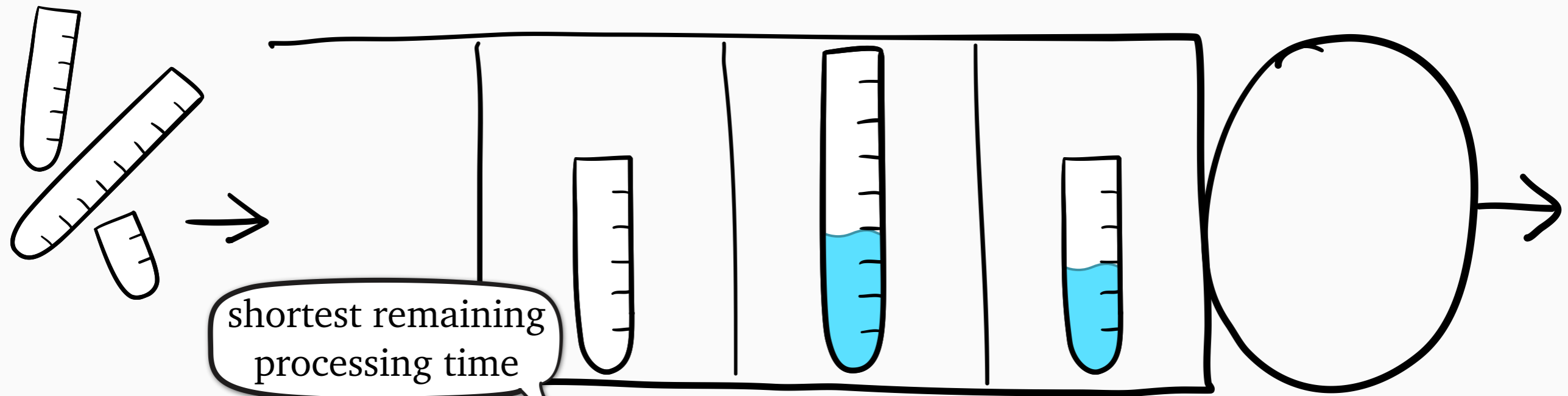
SRPT: always serve job of *least remaining size*



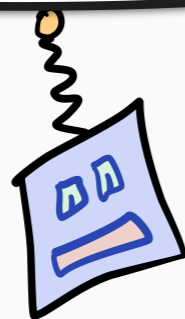
rank



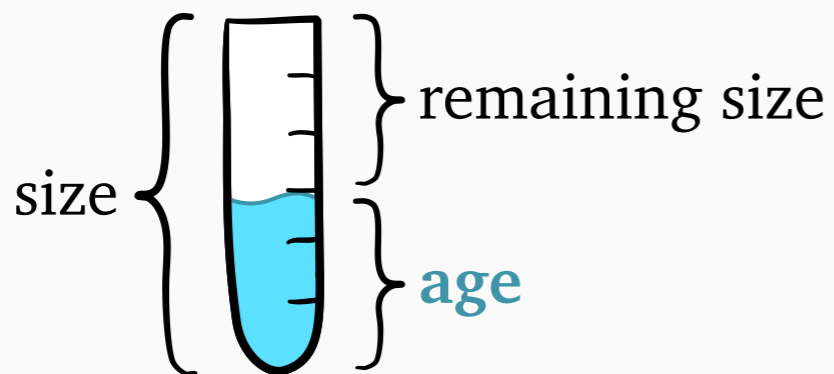
How to Schedule?



shortest remaining processing time



SRPT: always serve job of *least remaining size*

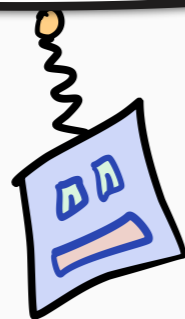
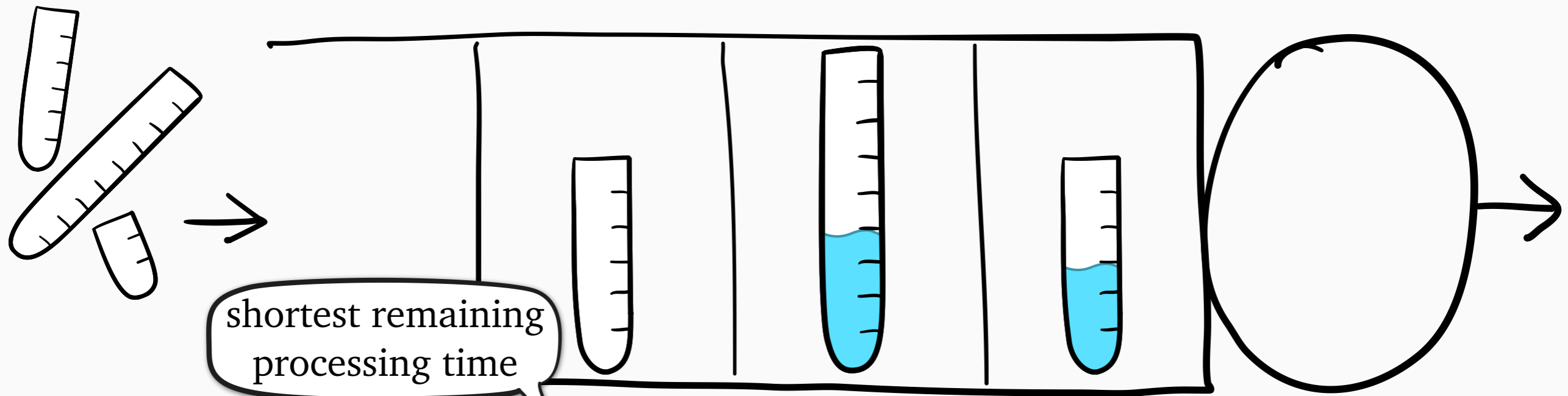


rank

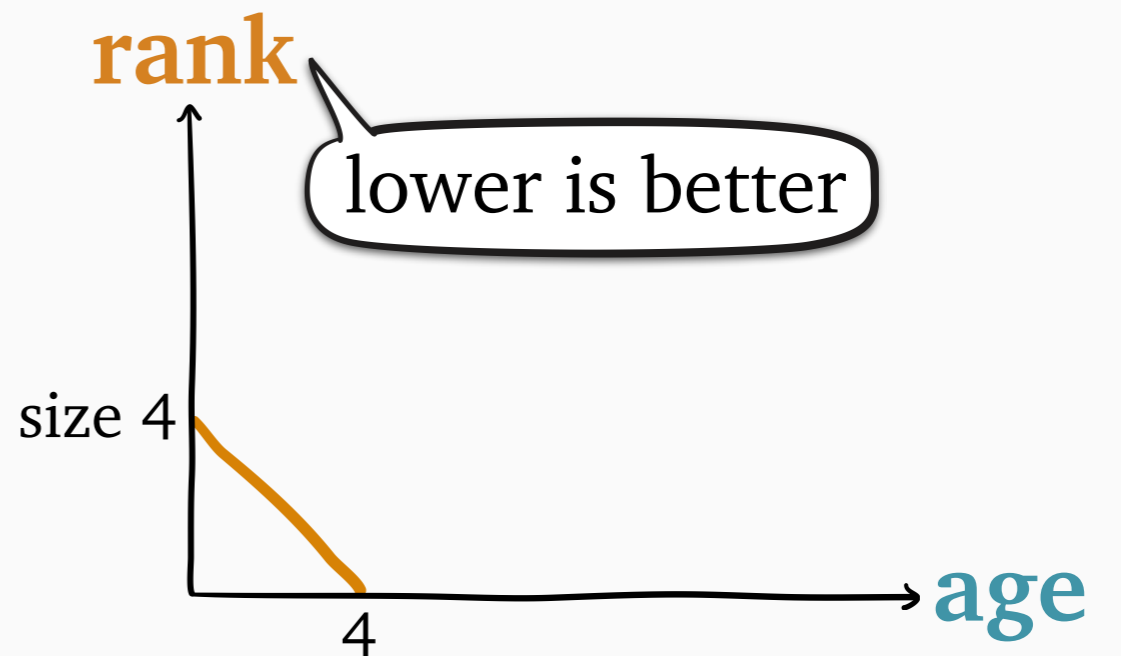
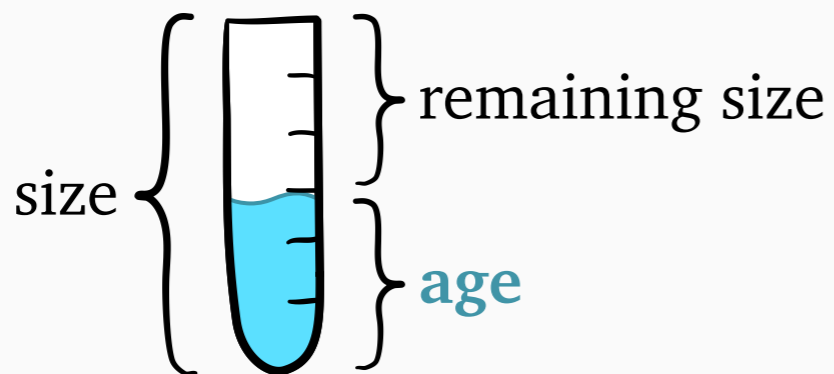
lower is better

age

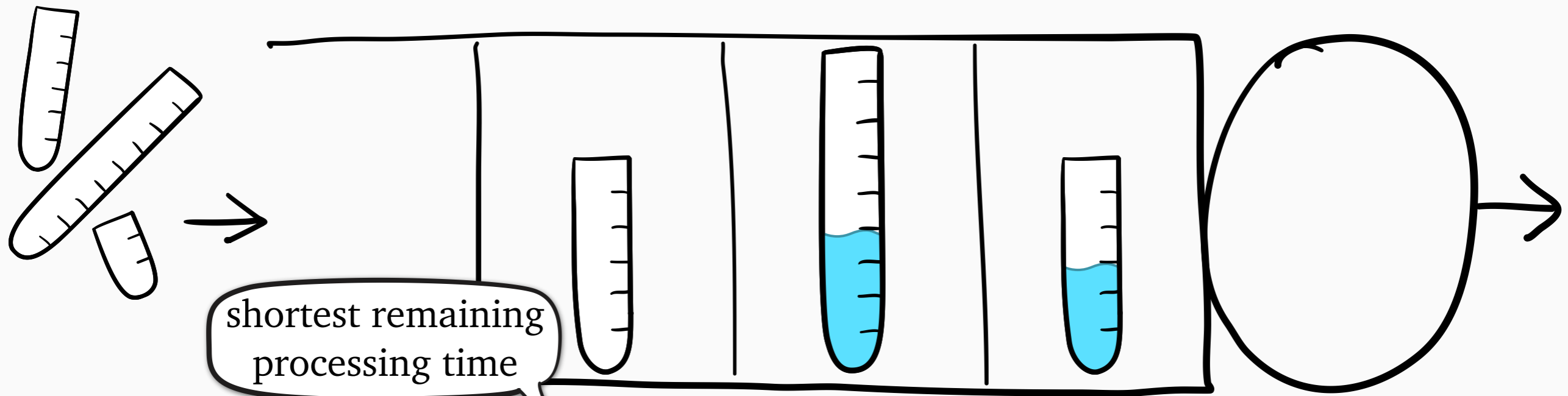
How to Schedule?



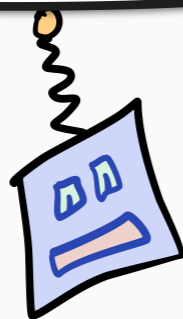
SRPT: always serve job of *least remaining size*



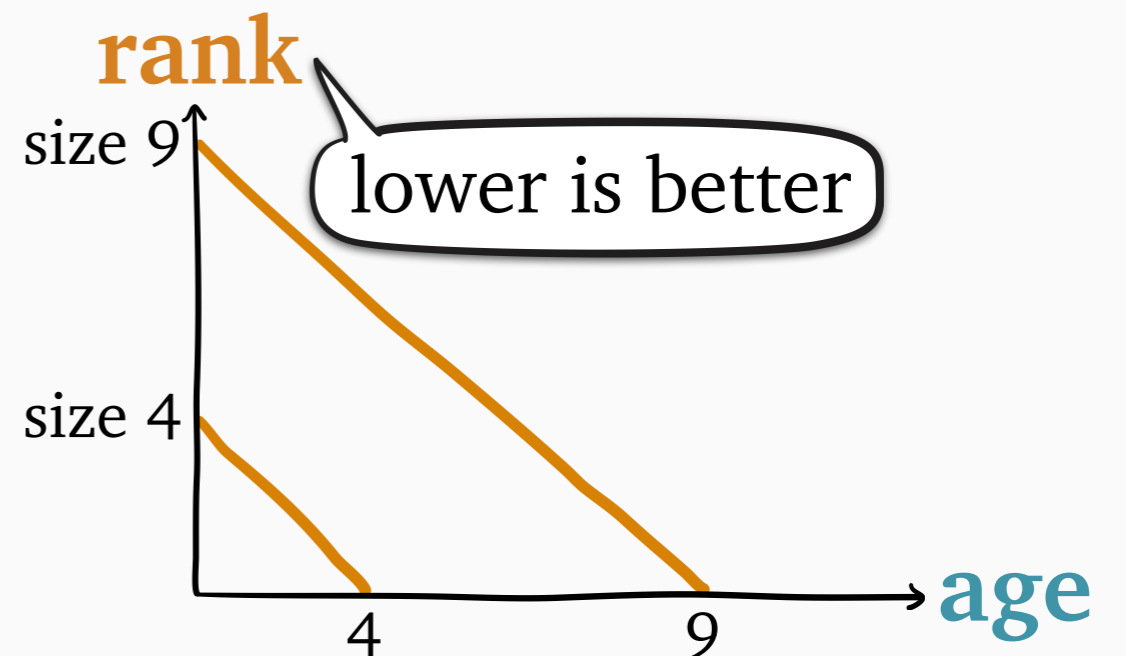
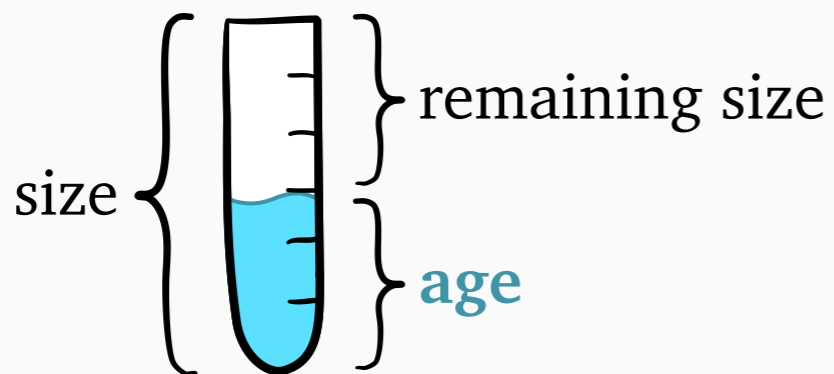
How to Schedule?



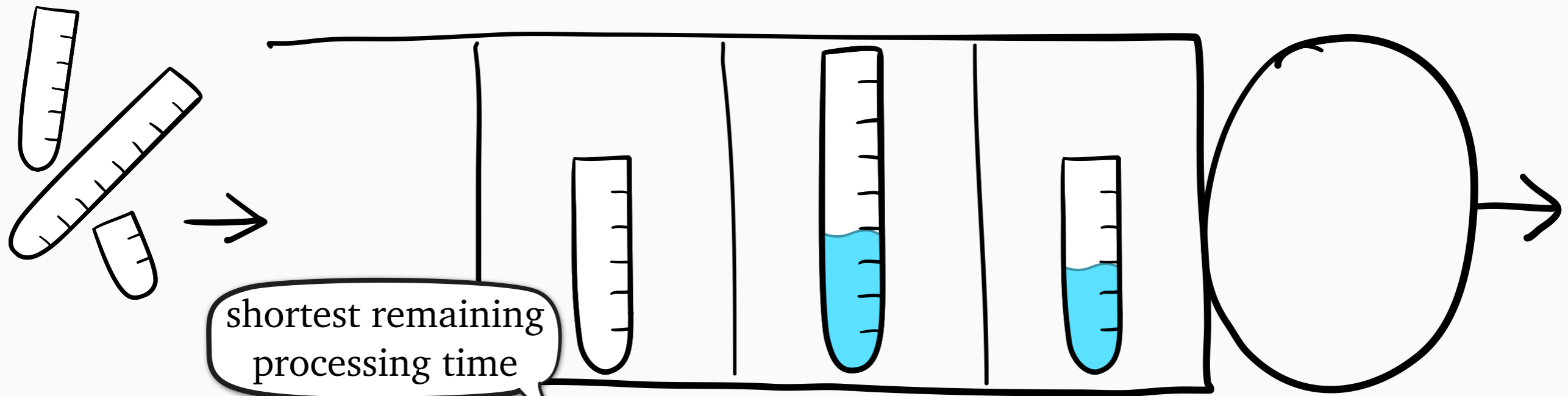
shortest remaining processing time



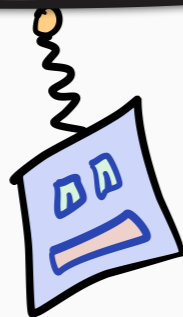
SRPT: always serve job of *least remaining size*



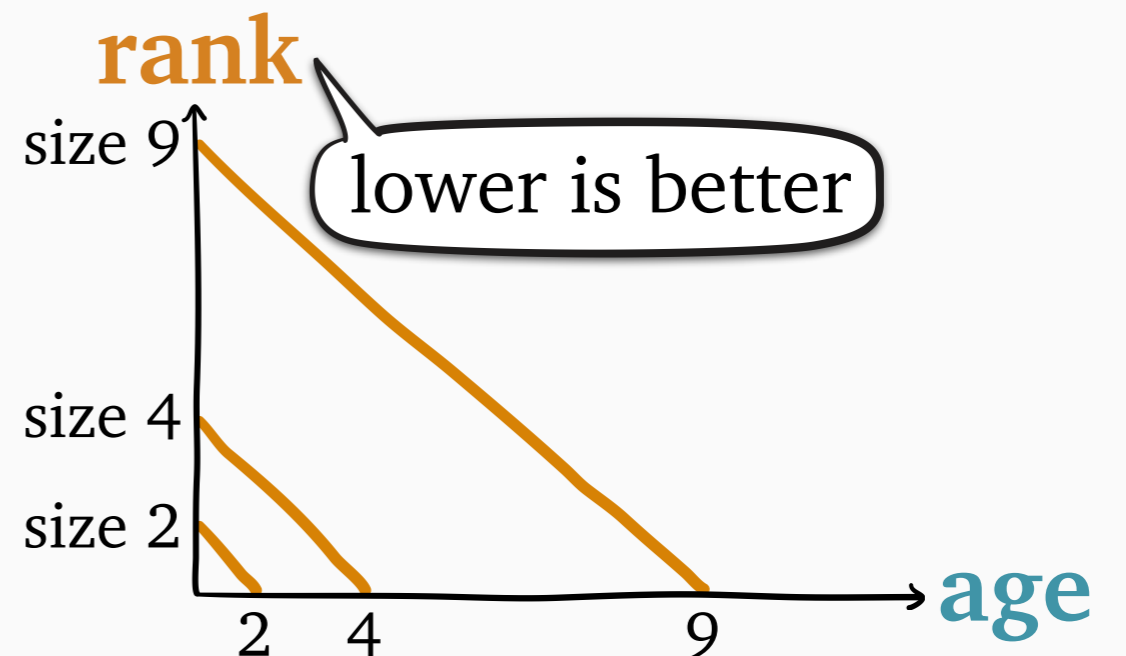
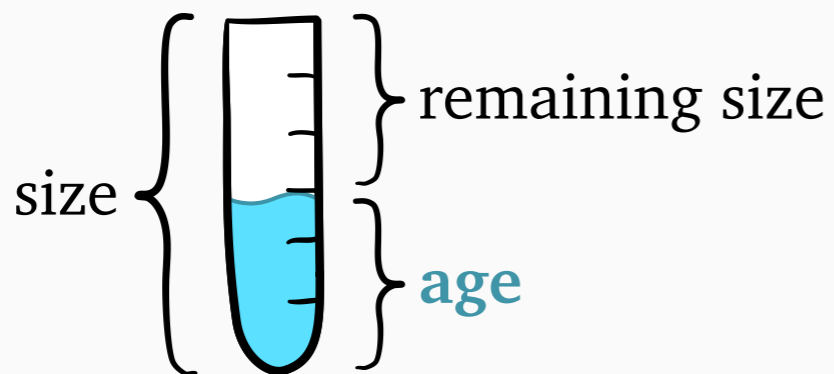
How to Schedule?



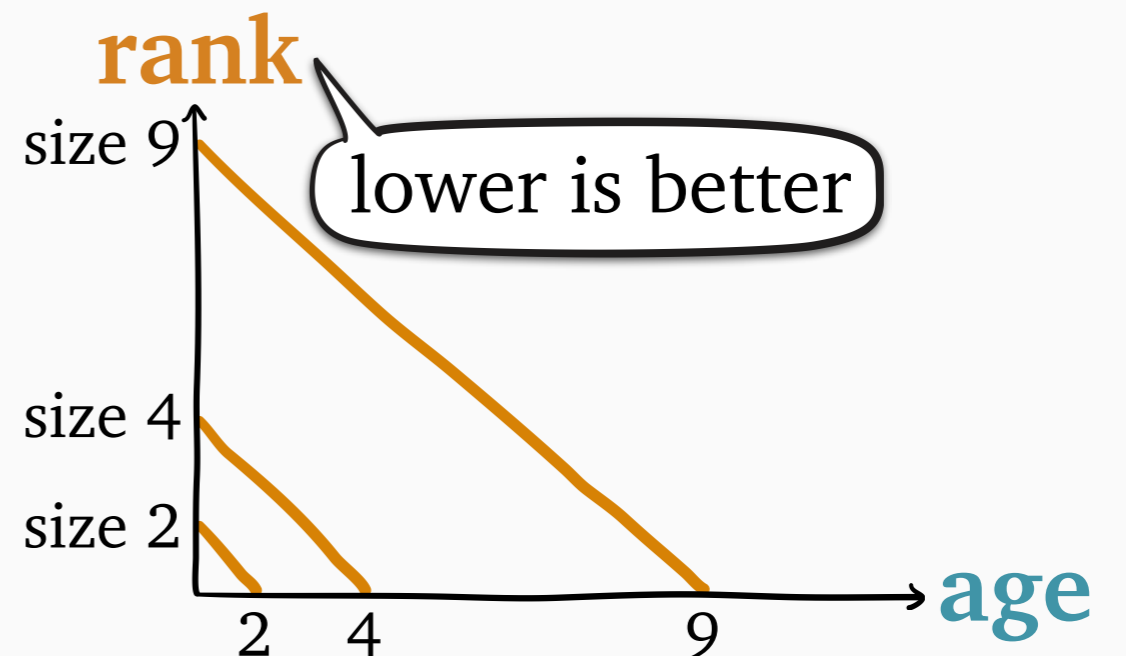
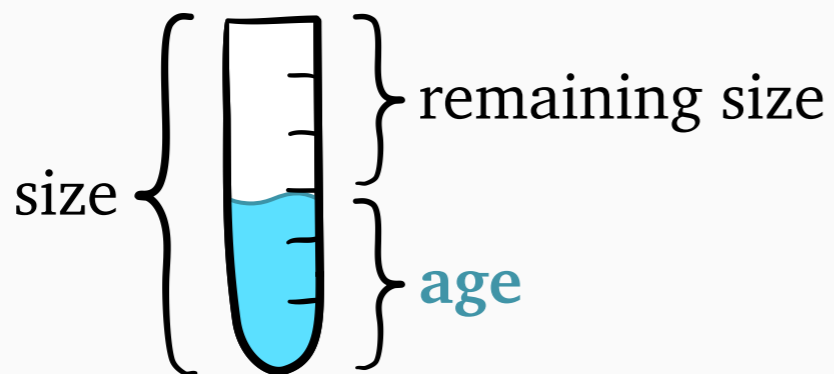
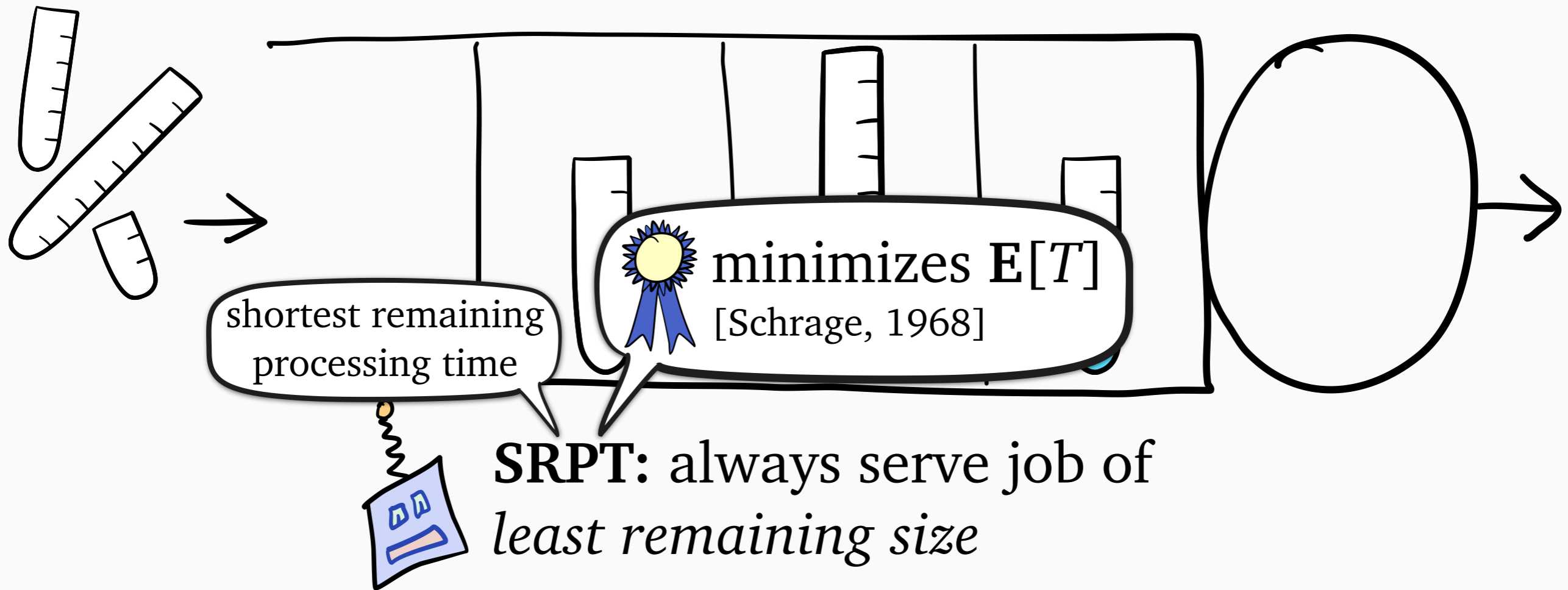
shortest remaining processing time



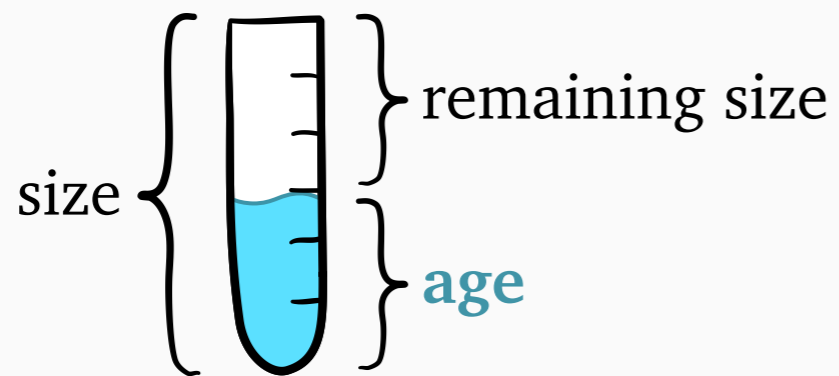
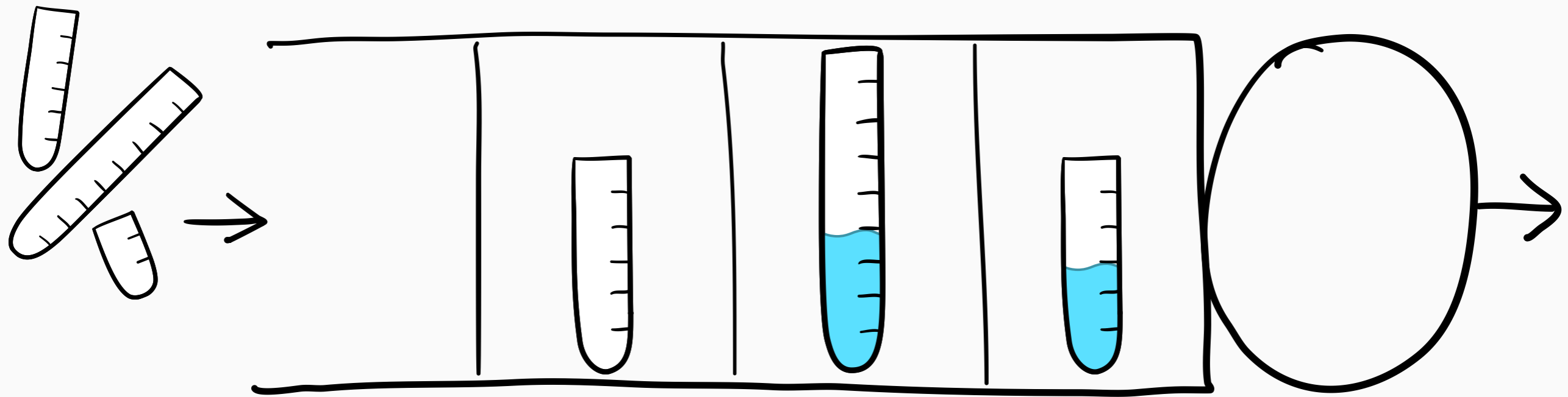
SRPT: always serve job of *least remaining size*



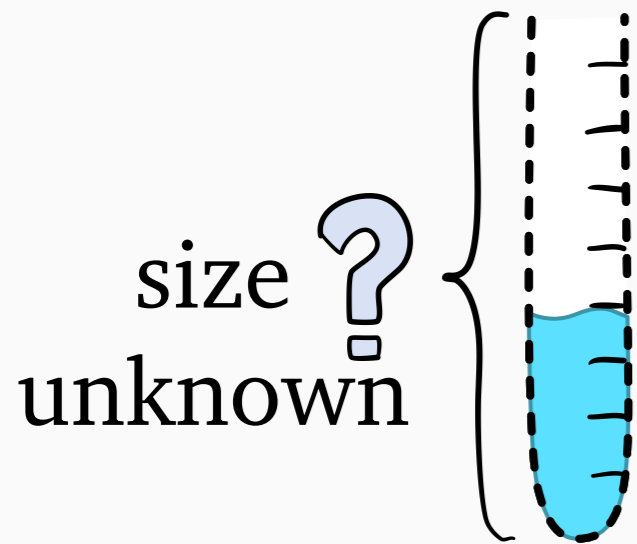
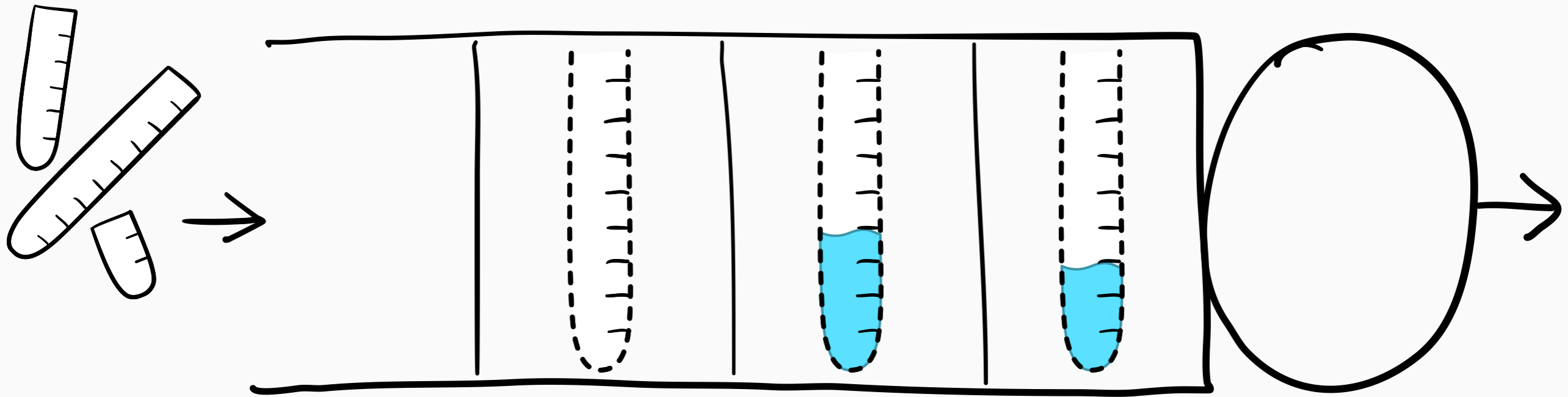
How to Schedule?



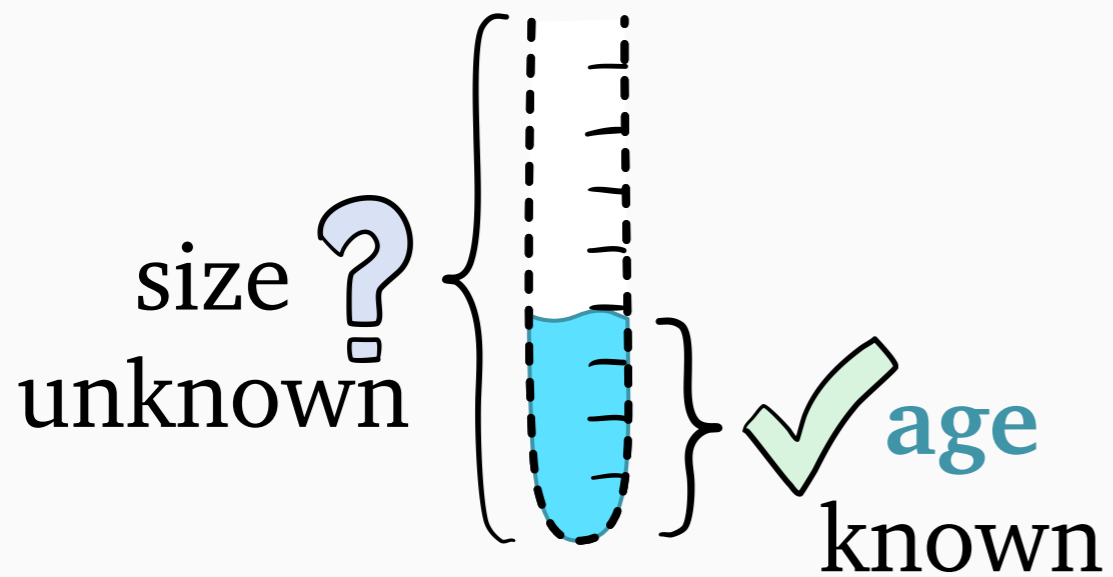
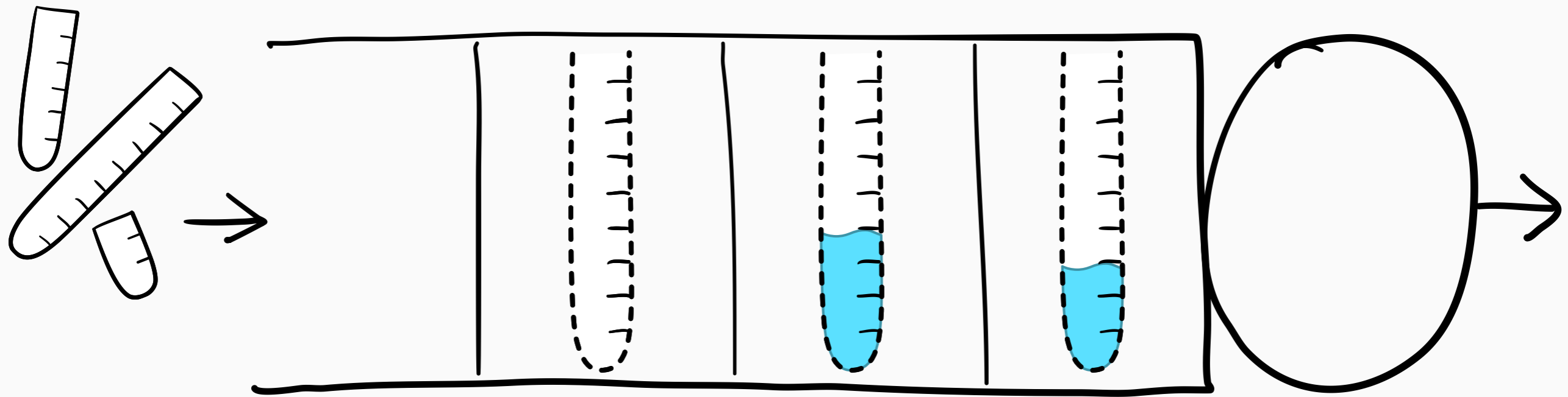
Unknown Job Sizes



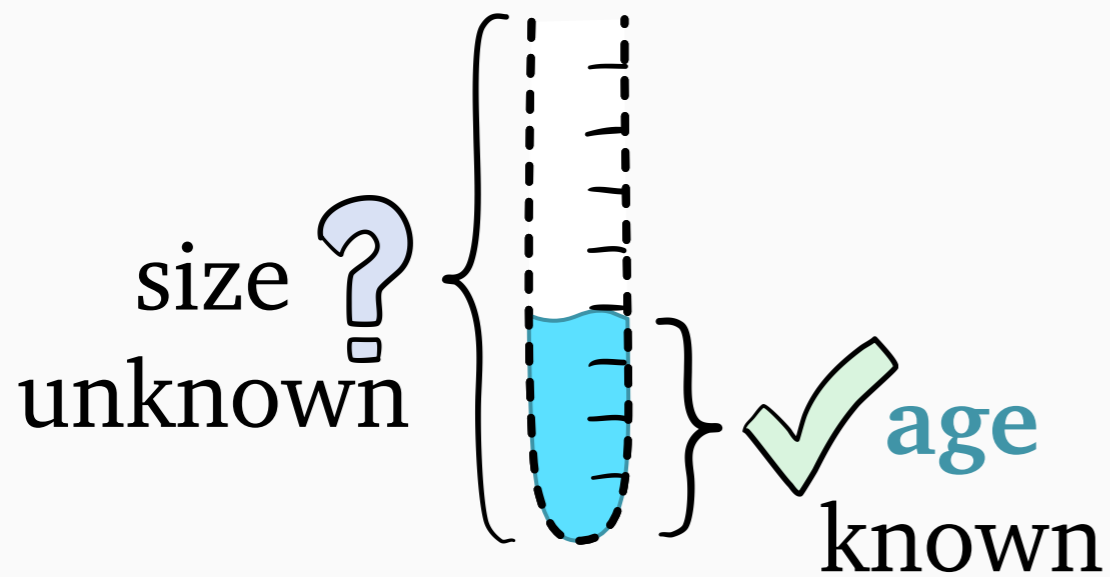
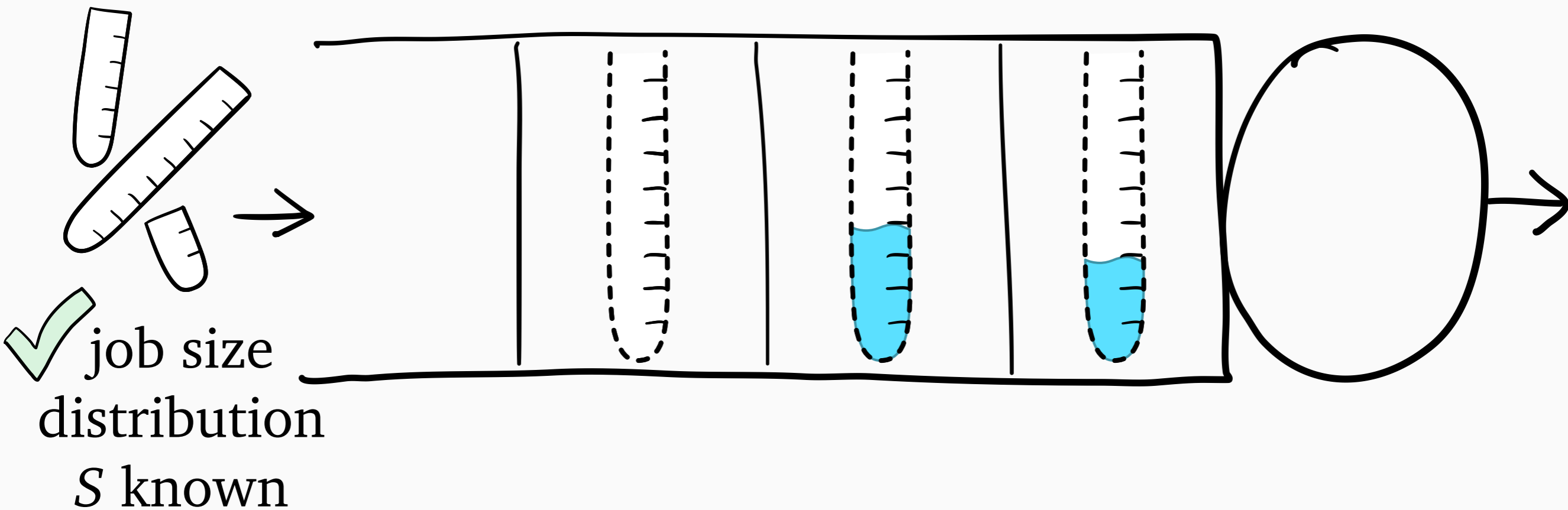
Unknown Job Sizes



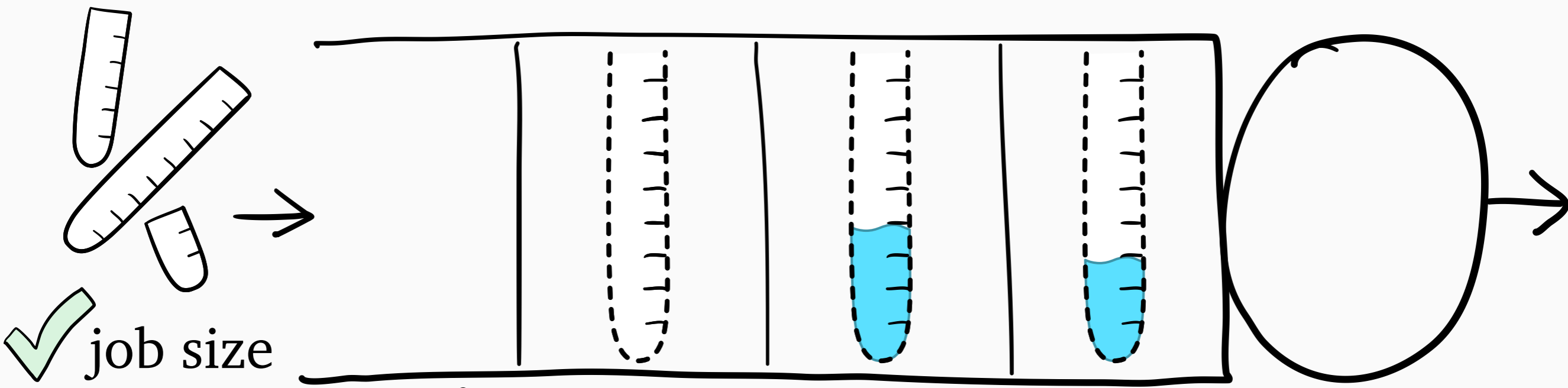
Unknown Job Sizes



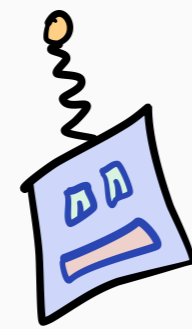
Unknown Job Sizes



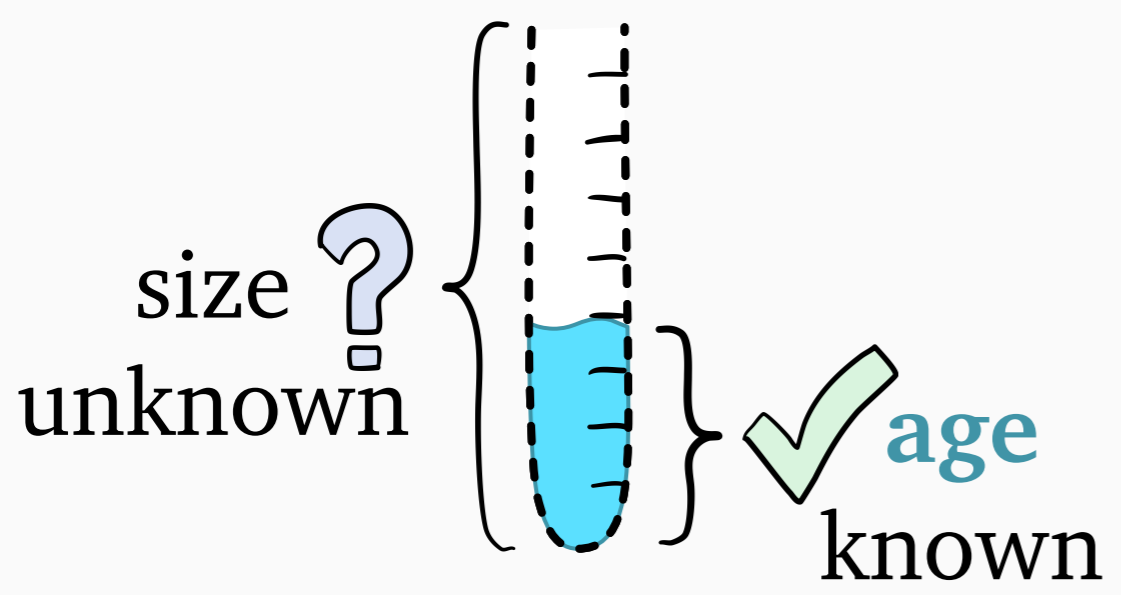
Unknown Job Sizes



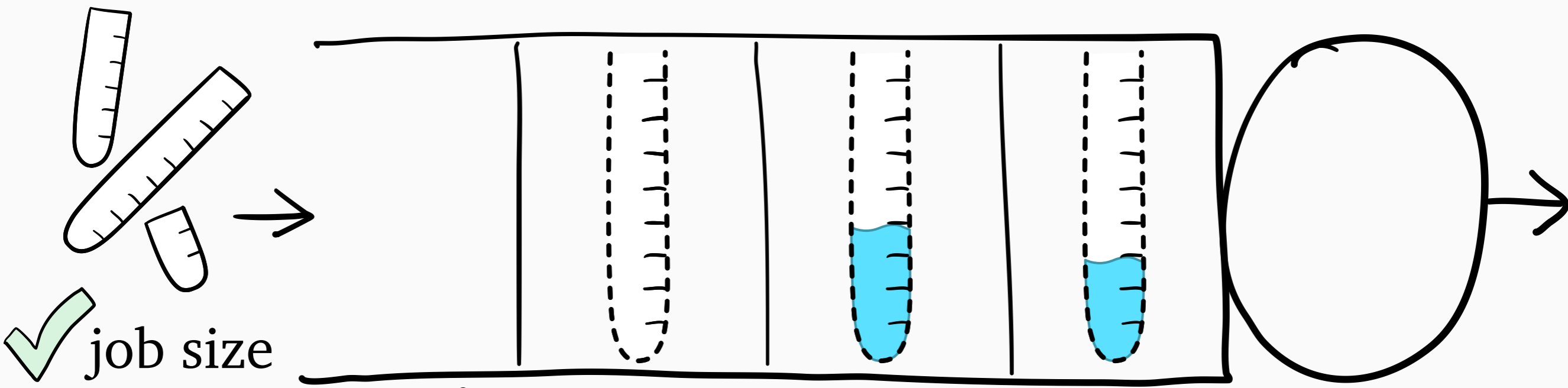
job size distribution S known



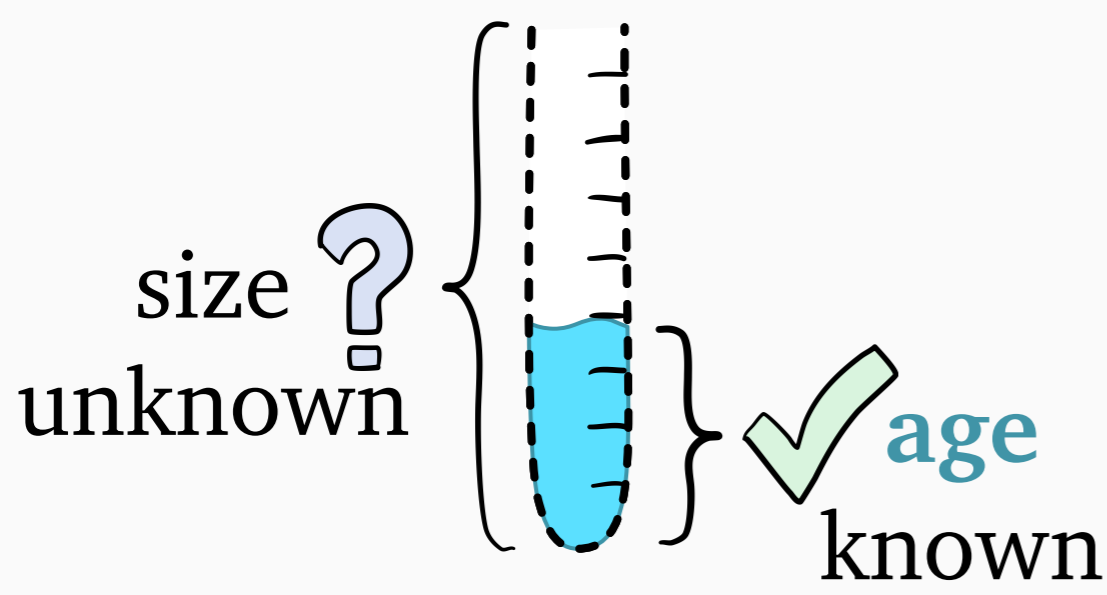
Gittins: computes job's rank using age and S



Unknown Job Sizes



Gittins: computes job's **rank** using **age** and S

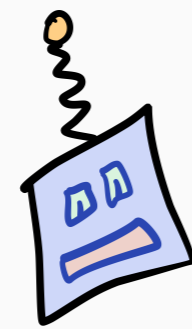
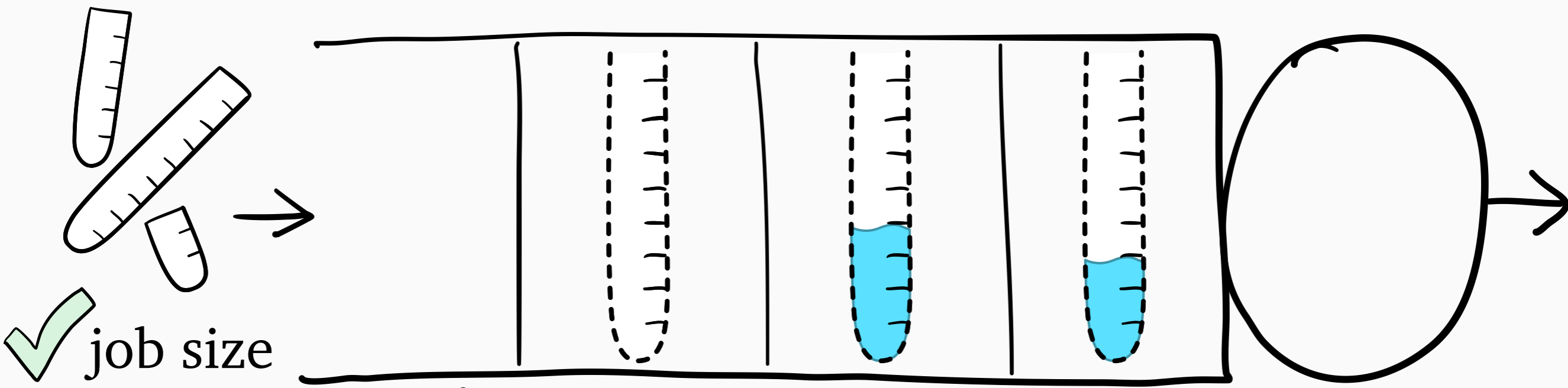


rank

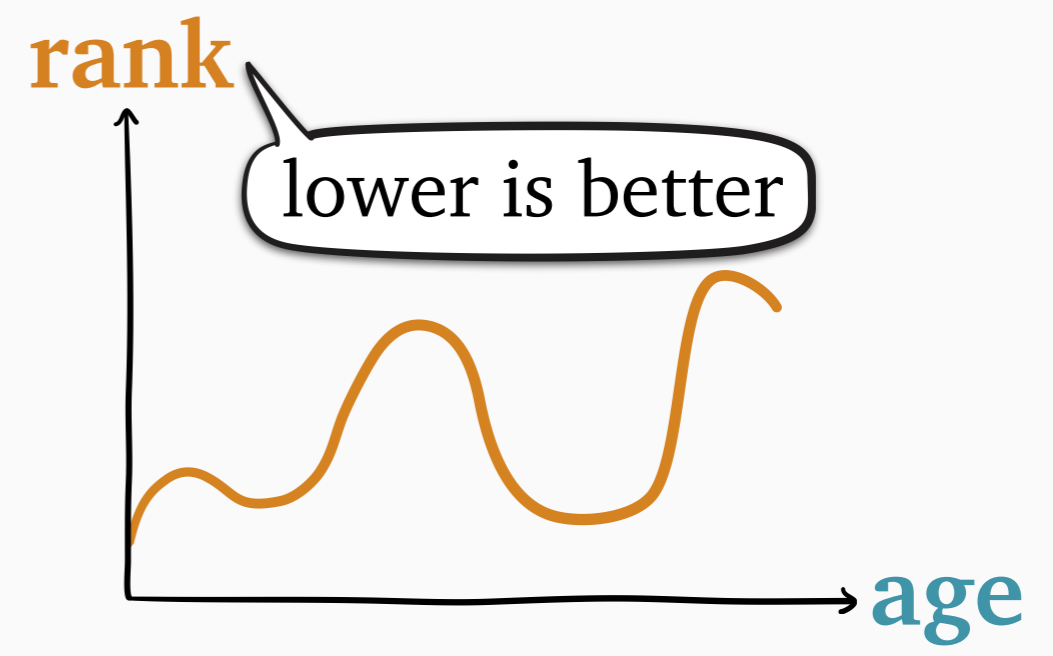
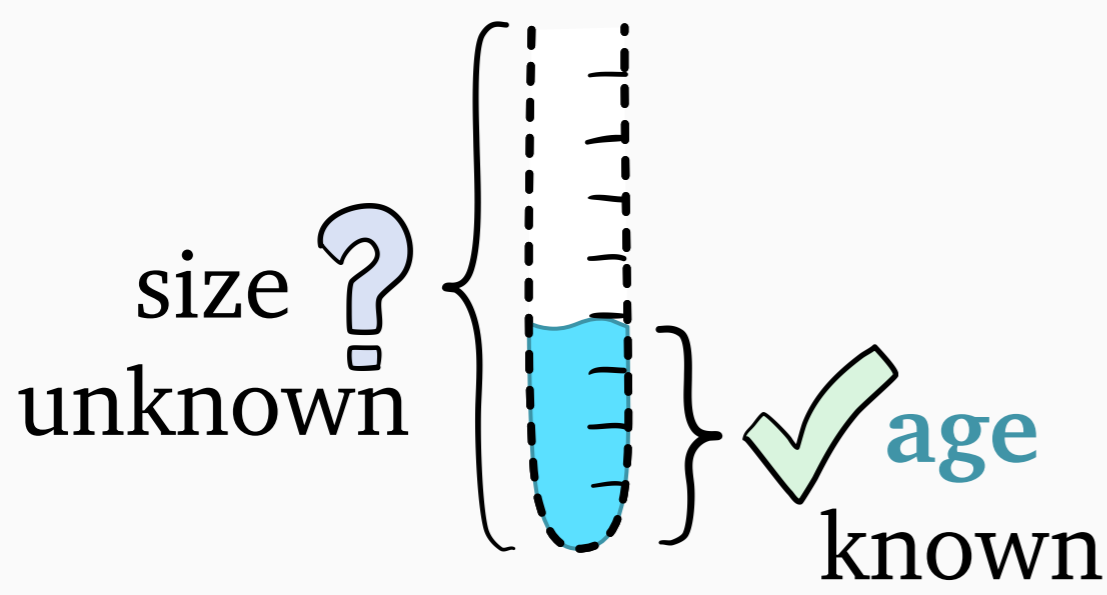
lower is better

age

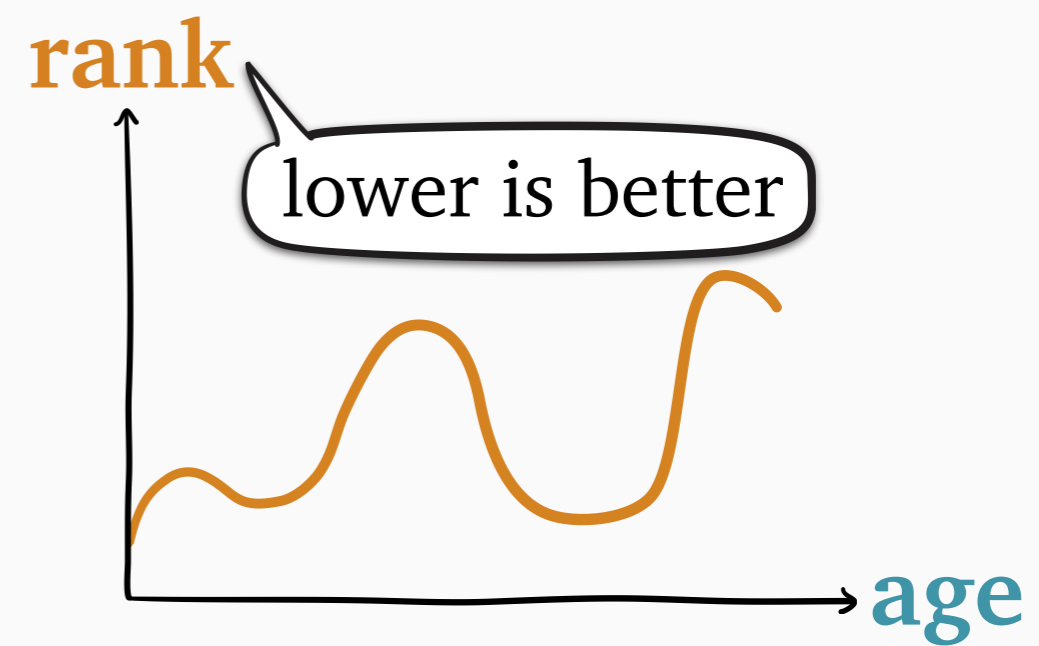
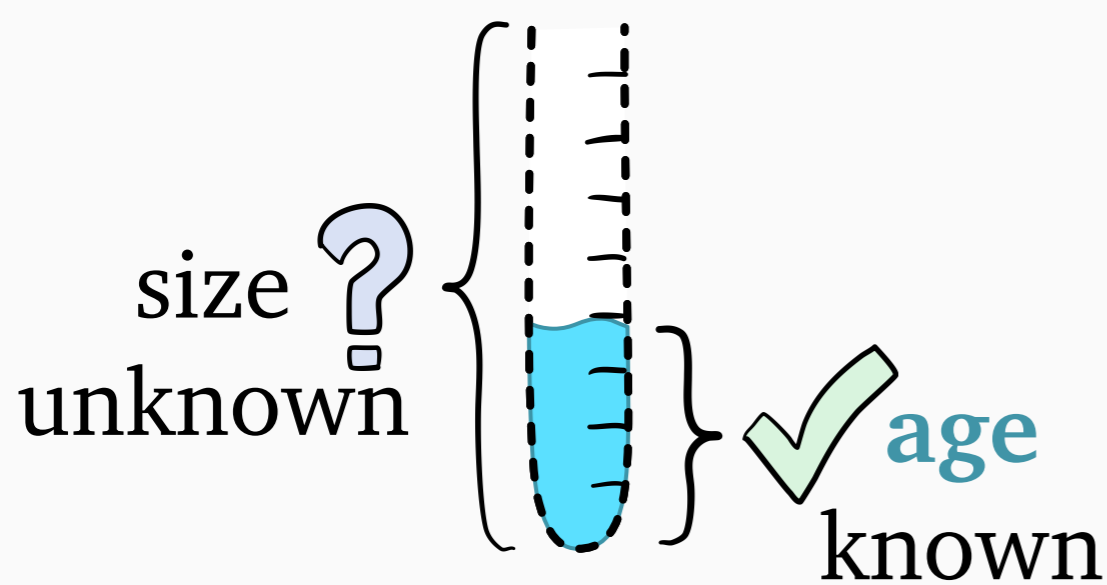
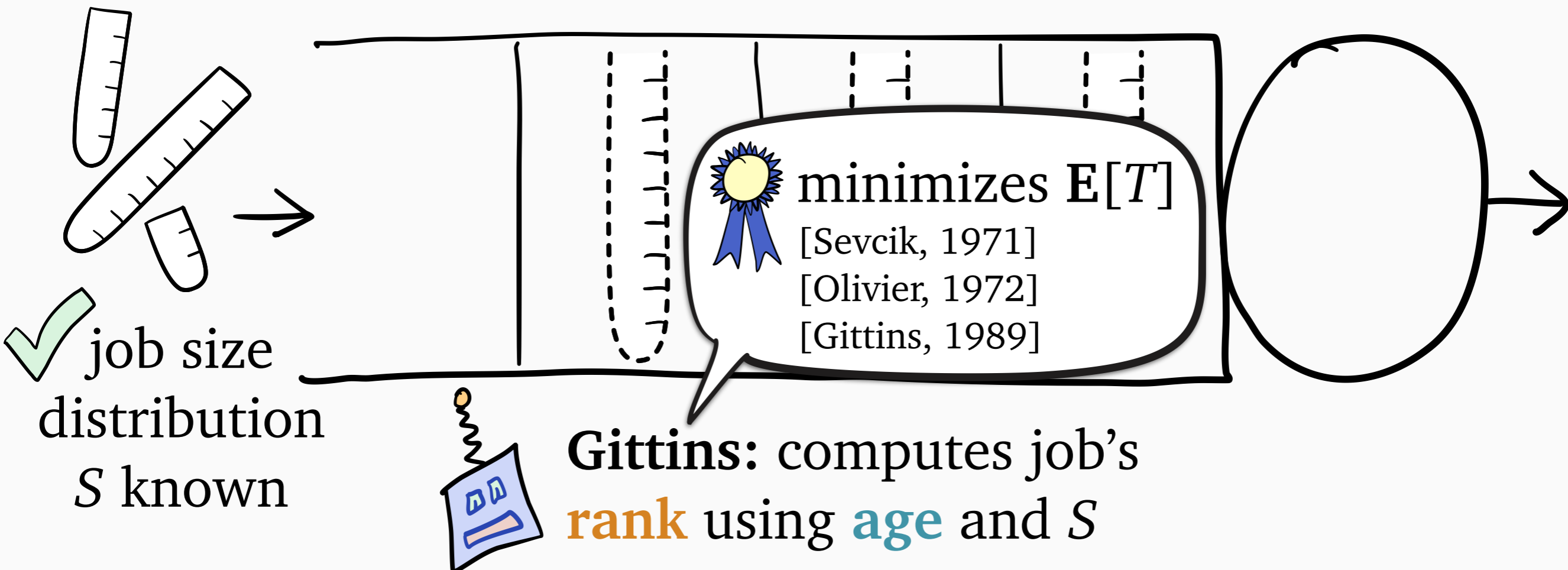
Unknown Job Sizes



Gittins: computes job's **rank** using **age** and S



Unknown Job Sizes



Gittins Optimality Results

minimizes $E[T]$

Size info:

unknown

Size distribution:

general

[Sevcik, 1971]

[Olivier, 1972]

Service:

preemptive

[Gittins, 1989]

Gittins Optimality Results

Size info:

unknown

Size distribution:

general

[Sevcik, 1971]

[Olivier, 1972]

Service:

preemptive

[Gittins, 1989]

Job weights:

equal, class-based

Job types:

all same, multiple classes

Gittins Optimality Results

Size info:

unknown

Size distribution:

general

[Sevcik, 1971]

[Olivier, 1972]

Service:

preemptive

[Gittins, 1989]

Job weights:

equal, class-based

Job types:

all same, multiple classes

Gittins Optimality Results

Size info:

unknown, known

Size distribution:

general

[Sevcik, 1971]

[Olivier, 1972]

Service:

preemptive

[Gittins, 1989]

Job weights:

equal, class-based

Job types:

all same, multiple classes

Gittins Optimality Results

Size info:

unknown, known

Size distribution:

general

[Sevcik, 1971]

[Olivier, 1972]

Service:

preemptive

[Gittins, 1989]

Job weights:

equal, class-based, size-based

Job types:

all same, multiple classes

Gittins Optimality Results

Size info:

unknown, known

cμ rule

[Cox and Smith, 1961]

SRPT

[Schrage, 1968]

Size distribution:

general

[Sevcik, 1971]

[Olivier, 1972]

Service:

preemptive

[Gittins, 1989]

Job weights:

equal, class-based, size-based

Job types:

all same, multiple classes

Gittins Optimality Results

Size info:

unknown, known

cμ rule

[Cox and Smith, 1961]

SRPT

[Schrage, 1968]

Size distribution:

general

[Sevcik, 1971]

[Olivier, 1972]

Service:

preemptive

[Gittins, 1989]

Job weights:

equal, class-based, size-based

Job types:

all same, multiple classes

Gittins Optimality Results

Size info:

unknown, known

Size distribution:

general, exponential

Service:

preemptive

Job weights:

equal, class-based, size-based

Job types:

all same, multiple classes

cμ rule

[Cox and Smith, 1961]

SRPT

[Schrage, 1968]

[Sevcik, 1971]

[Olivier, 1972]

[Gittins, 1989]

Gittins Optimality Results

Size info:

unknown, known

Size distribution:

general, exponential

Service:

preemptive, nonpreemptive

Job weights:

equal, class-based, size-based

Job types:

all same, multiple classes

cμ rule

[Cox and Smith, 1961]

SRPT

[Schrage, 1968]

[Sevcik, 1971]

[Olivier, 1972]

[Gittins, 1989]

Gittins Optimality Results

Size info:

unknown, known

cμ rule

[Cox and Smith, 1961]

SRPT

[Schrage, 1968]

Size distribution:

general, exponential

[Sevcik, 1971]

[Olivier, 1972]

Service:

preemptive, nonpreemptive

[Klimov, 1975]

[Gittins, 1989]

Job weights:

equal, class-based, size-based

[Lai and Ying, 1988]

Job types:

all same, multiple classes, multistage

Gittins Optimality Results

Size info:

unknown, known

Size distribution:

general, exponential

Service:

preemptive, nonpreemptive

Job weights:

equal, class-based, size-based

Job types:

all same, multiple classes, multistage

cμ rule

[Cox and Smith, 1961]

SRPT

[Schrage, 1968]

[Sevcik, 1971]

[Olivier, 1972]

[Klimov, 1975]

[Gittins, 1989]

[Lai and Ying, 1988]

Gittins Optimality Results

Size info:

unknown, known

Size distribution:

general, exponential

Service:

preemptive, nonpreemptive

Job weights:

equal, class-based, size-based

Job types:

all same, multiple classes, multistage

cμ rule

[Cox and Smith, 1961]

SRPT

[Schrage, 1968]

[Sevcik, 1971]

[Olivier, 1972]

[Klimov, 1975]

[Gittins, 1989]

[Lai and Ying, 1988]

Gittins Optimality Results

Size info:

unknown, known

cμ rule

[Cox and Smith, 1961]

SRPT

[Schrage, 1968]

Size distribution:

general, exponential

[Sevcik, 1971]

[Olivier, 1972]

Service:

preemptive, nonpreemptive

[Klimov, 1975]

[Gittins, 1989]

Job weights:

equal, class-based, size-based

[Lai and Ying, 1988]

[Bertsimas, 1995]

Job types:

all same, multiple classes, multistage

[Dacre et al., 1999]

[Whittle, 2007]

Gittins Optimality Results

Size info:

unknown, known, **noisily estimated?**

cμ rule

[Cox and Smith, 1961]

SRPT

[Schrage, 1968]

Size distribution:

general, exponential

[Sevcik, 1971]

[Olivier, 1972]

Service:

preemptive, nonpreemptive

[Klimov, 1975]

[Gittins, 1989]

Job weights:

equal, class-based, size-based

[Lai and Ying, 1988]

[Bertsimas, 1995]

Job types:

all same, multiple classes, multistage

[Dacre et al., 1999]

[Whittle, 2007]

Gittins Optimality Results

Size info:

unknown, known, **noisily estimated?**

cμ rule

[Cox and Smith, 1961]

SRPT

[Schrage, 1968]

Size distribution:

general, exponential

[Sevcik, 1971]

[Olivier, 1972]

Service:

preemptive, nonpreemptive, **partly preemptive?**

[Klimov, 1975]

[Gittins, 1989]

Job weights:

equal, class-based, size-based

[Lai and Ying, 1988]

[Bertsimas, 1995]

Job types:

all same, multiple classes, multistage

[Dacre et al., 1999]

[Whittle, 2007]

Gittins Optimality Results

Size info:

unknown, known, **noisily estimated?**

cμ rule

[Cox and Smith, 1961]

SRPT

[Schrage, 1968]

Size distribution:

general, exponential

[Sevcik, 1971]

[Olivier, 1972]

Service:

preemptive, nonpreemptive, **partly preemptive?**

[Klimov, 1975]

[Gittins, 1989]

Job weights:

equal, class-based, size-based, **non-constant?**

[Lai and Ying, 1988]

[Bertsimas, 1995]

Job types:

all same, multiple classes, multistage

[Dacre et al., 1999]

[Whittle, 2007]

Gittins Optimality Results

Size info:

unknown, known, **noisily estimated?**

cμ rule

[Cox and Smith, 1961]

SRPT

[Schrage, 1968]

Size distribution:

general, exponential

[Sevcik, 1971]

[Olivier, 1972]

Service:

preemptive, nonpreemptive, **partly preemptive?**

[Klimov, 1975]

[Gittins, 1989]

Job weights:

assuming finitely μ -based, **non-constant?**

[Lai and Ying, 1988]

[Bertsimas, 1995]

many classes

[Dacre et al., 1999]

Job types:

all same, multiple classes, multistage

[Whittle, 2007]

Gittins Optimality Results

Size info:

unknown, known, **noisily estimated?**

cμ rule

[Cox and Smith, 1961]

SRPT

[Schrage, 1968]

Size distribution:

general, exponential

[Sevcik, 1971]

assuming upper

bound on job sizes, **partly preemptive?**

[Olivier, 1972]

[Klimov, 1975]

[Gittins, 1989]

Job weights:

assuming finitely e-based, **non-constant?**

[Lai and Ying, 1988]

[Bertsimas, 1995]

[Dacre et al., 1999]

Job types:

all same, multiple classes, multistage

[Whittle, 2007]

We **unify** and **generalize**
all prior Gittins optimality results

Gittins Optimality Results

Size info:

unknown, known, **noisily estimated?**

[Cox and Smith, 1961]

[Schrage, 1968]

Size distribution:

general, exponential

[Sevcik, 1971]

[Olivier, 1972]

Service:

preemptive, nonpreemptive, **partly preemptive?**

[Klimov, 1975]

[Gittins, 1989]

Job weights:

equal, class-based, size-based, **non-constant?**

[Lai and Ying, 1988]

[Bertsimas, 1995]

Job types:

all same, multiple classes, multistage

[Dacre et al., 1999]

[Whittle, 2007]

Gittins Optimality Results

Size info:

unknown, known, **noisily estimated?**

Size distribution:

general, exponential

Service:

preemptive, nonpreemptive, **partly preemptive?**

Job weights:

equal, class-based, size-based, **non-constant?**

Job types:

all same, multiple classes, multistage

our result

[Smith, 1961]

[Schrage, 1968]

[Sevcik, 1971]

[Olivier, 1972]

[Klimov, 1975]

[Gittins, 1989]

[Lai and Ying, 1988]

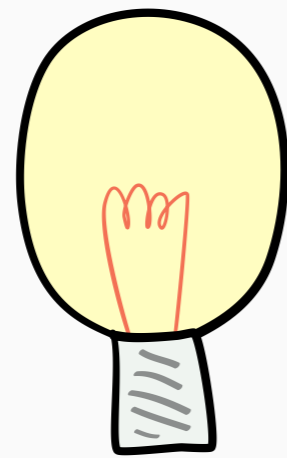
[Bertsimas, 1995]

[Dacre et al., 1999]

[Whittle, 2007]

We **unify** and **generalize**
all prior Gittins optimality results

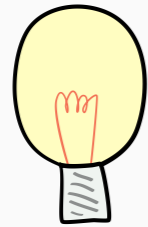
We **unify** and **generalize**
all prior Gittins optimality results



Key idea: very general
definition of “job”

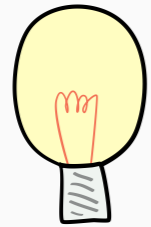
General Job Model

General Job Model



Key idea: a job is a *Markov process*

General Job Model



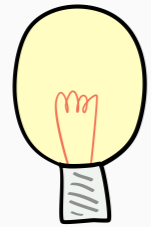
Key idea: a job is a *Markov process*



general *state space*



General Job Model



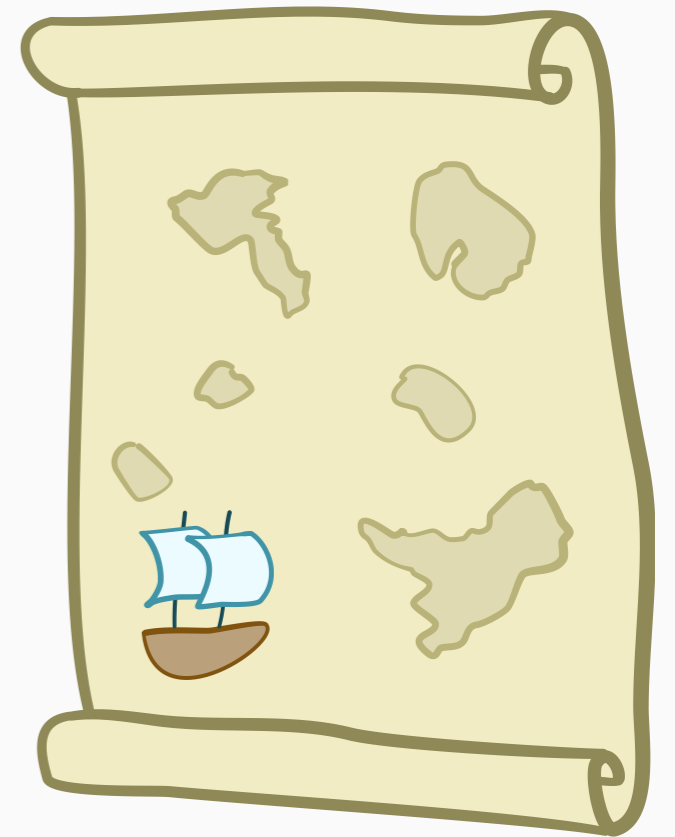
Key idea: a job is a *Markov process*



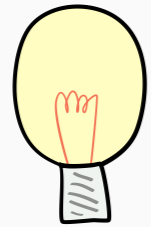
general *state space*



job's *state* encodes all known info



General Job Model



Key idea: a job is a *Markov process*



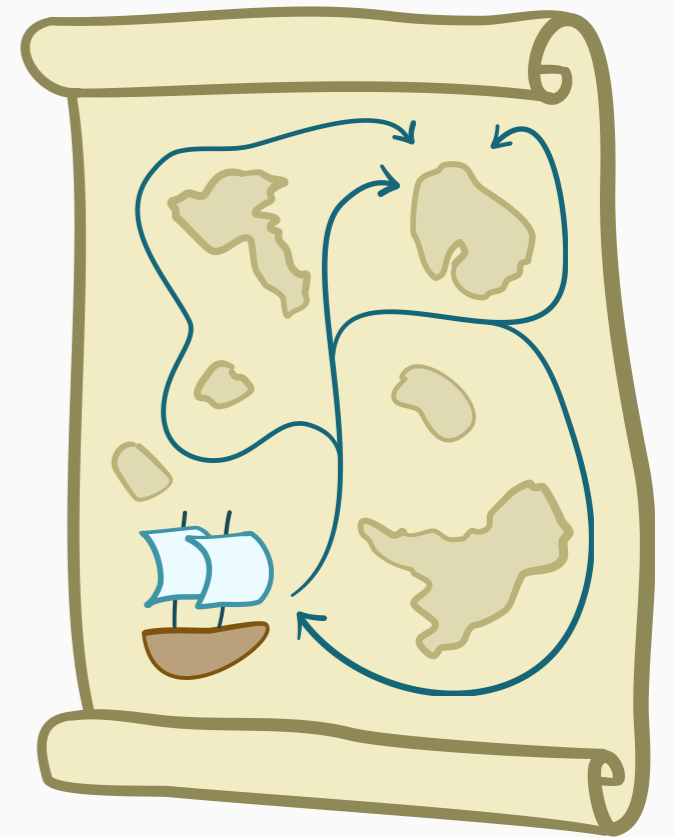
general *state space*



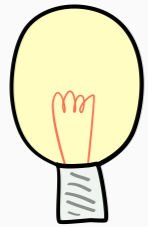
job's **state** encodes all known info



state *stochastically evolves* with service



General Job Model



Key idea: a job is a *Markov process*



general *state space*



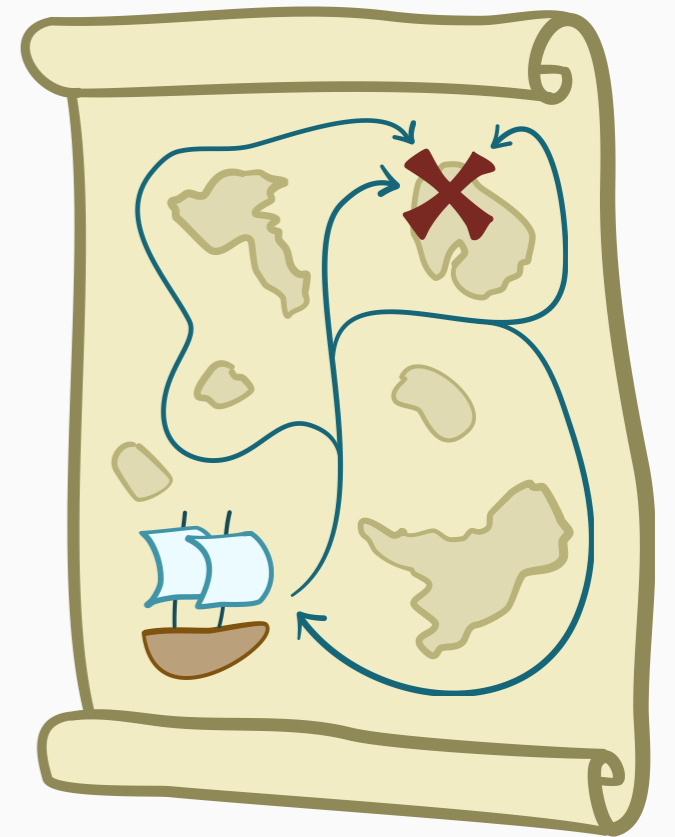
job's *state* encodes all known info



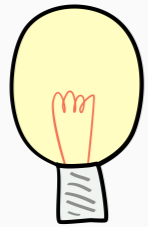
state *stochastically evolves* with service



completes upon entering *goal state*



General Job Model



Key idea: a job is a *Markov process*



general *state space*



job's *state* encodes all known info



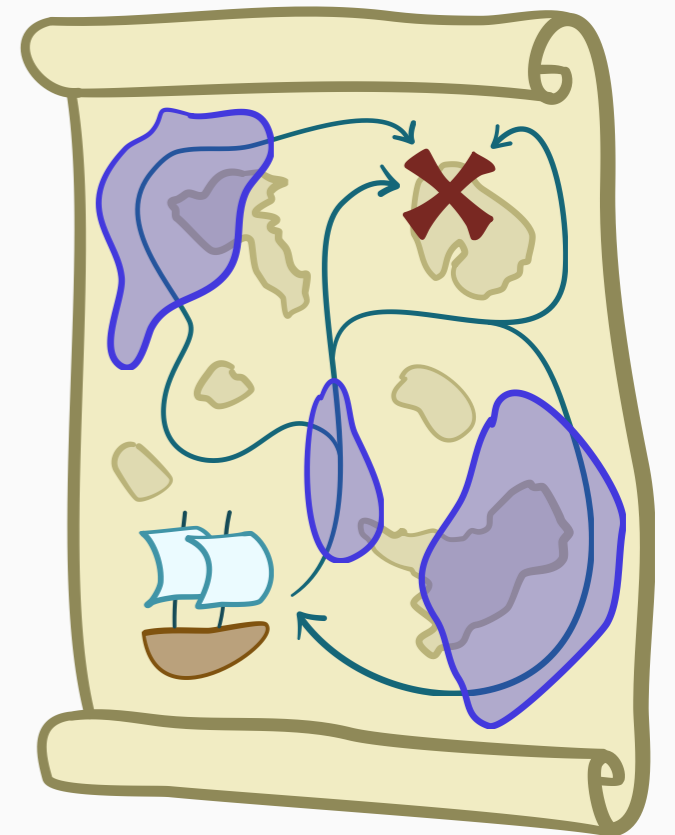
state *stochastically evolves* with service



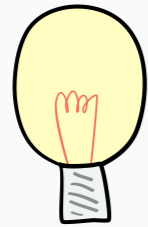
completes upon entering *goal state*



set of *nonpreemptible states*



General Job Model



Key idea: a job is a *Markov process*



general *state space*



job's *state* encodes all known info



state *stochastically evolves* with service



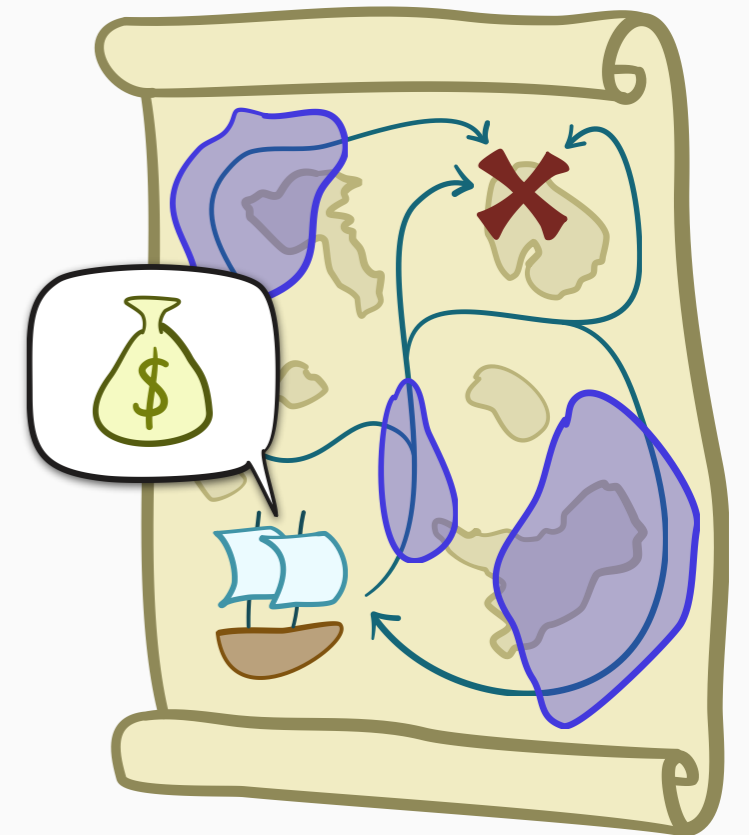
completes upon entering *goal state*







set of *nonpreemptible states*



every *state* has a *holding cost*



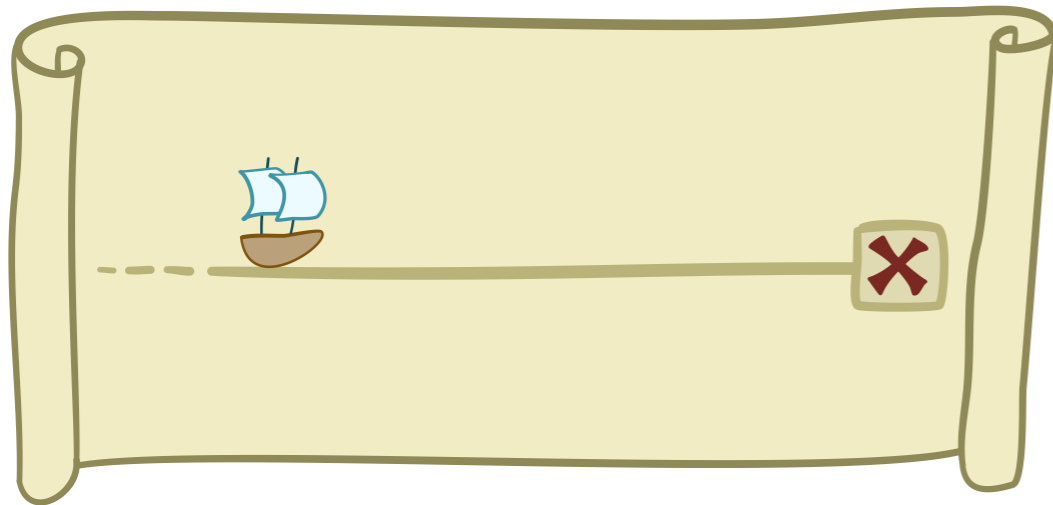
Job Examples





-  general *state space*
-  job's *state* encodes all known info
-  *state* *stochastically evolves* with service
-  completes upon entering *goal state*

Job Examples

Known Size

state = remaining size

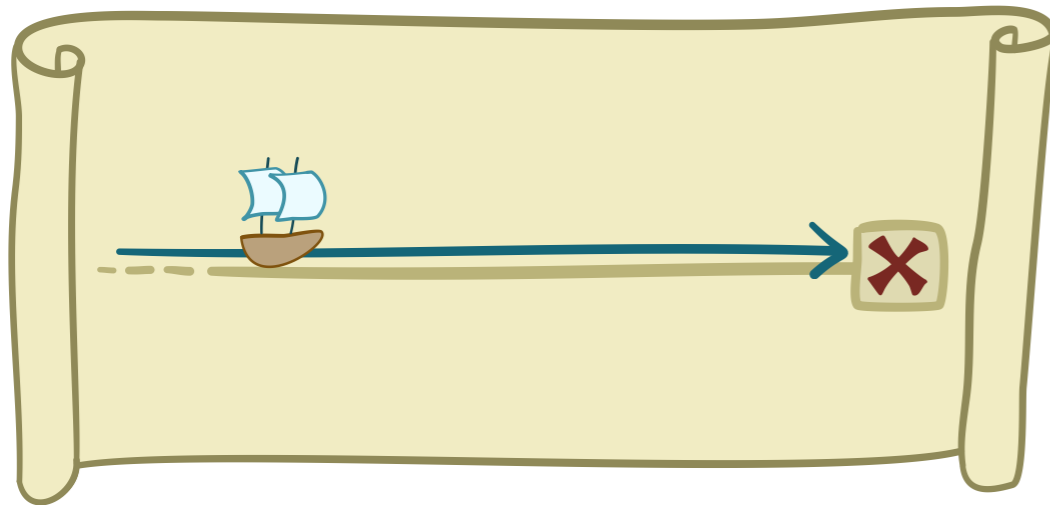






-  general *state space*
-  job's **state** encodes all known info
-  **state** *stochastically evolves* with service
-  completes upon entering **goal state**

Job Examples

Known Size

state = remaining size

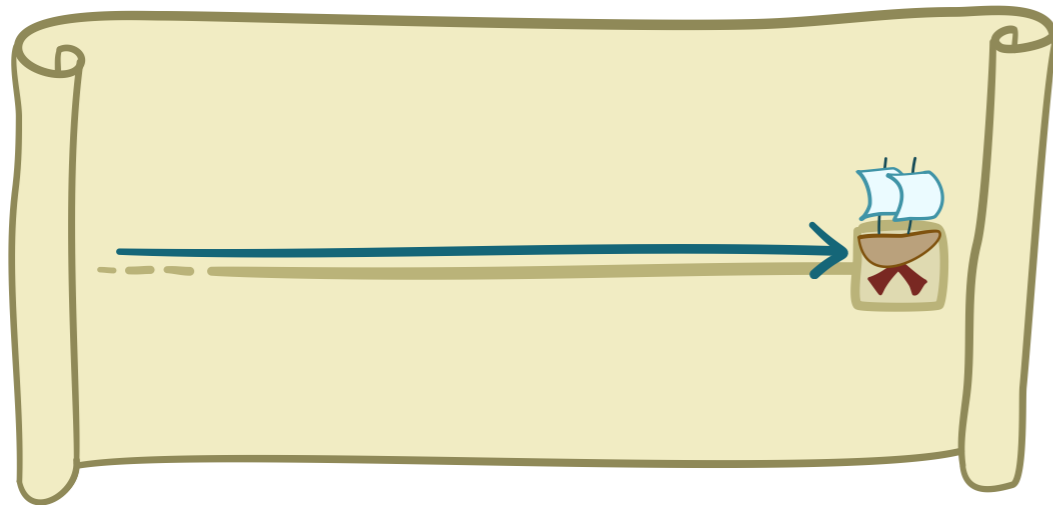






-  general *state space*
-  job's *state* encodes all known info
-  *state* *stochastically evolves* with service
-  completes upon entering *goal state*

Job Examples

Known Size

state = remaining size

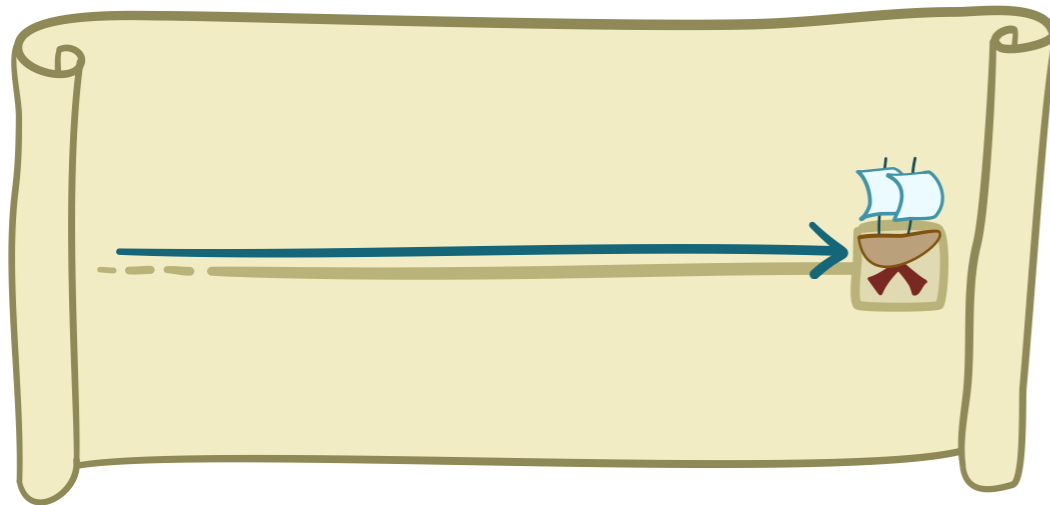


-  general *state space*
-  job's **state** encodes all known info
-  **state** *stochastically evolves* with service
-  completes upon entering **goal state**

Job Examples

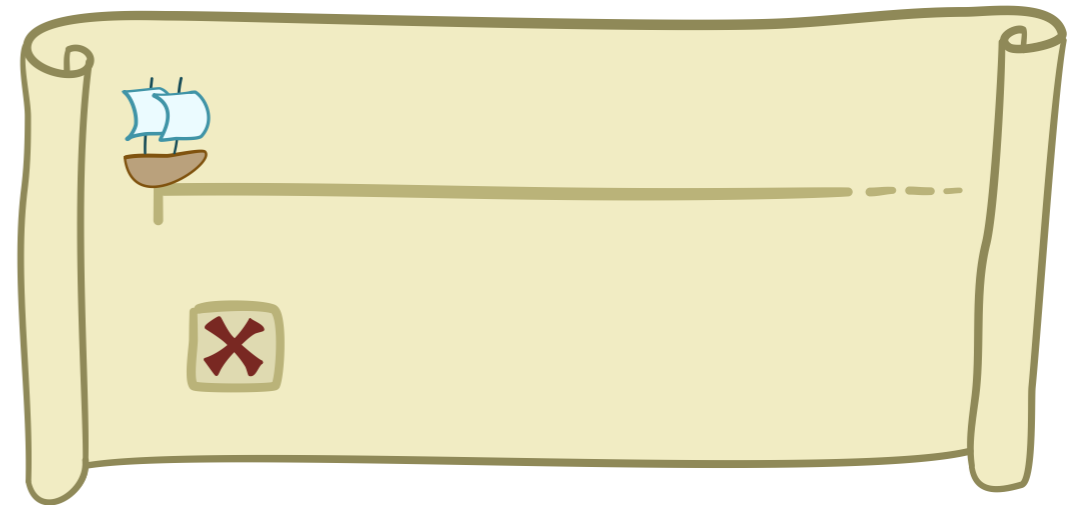
Known Size





state = remaining size



Unknown Size

state = age

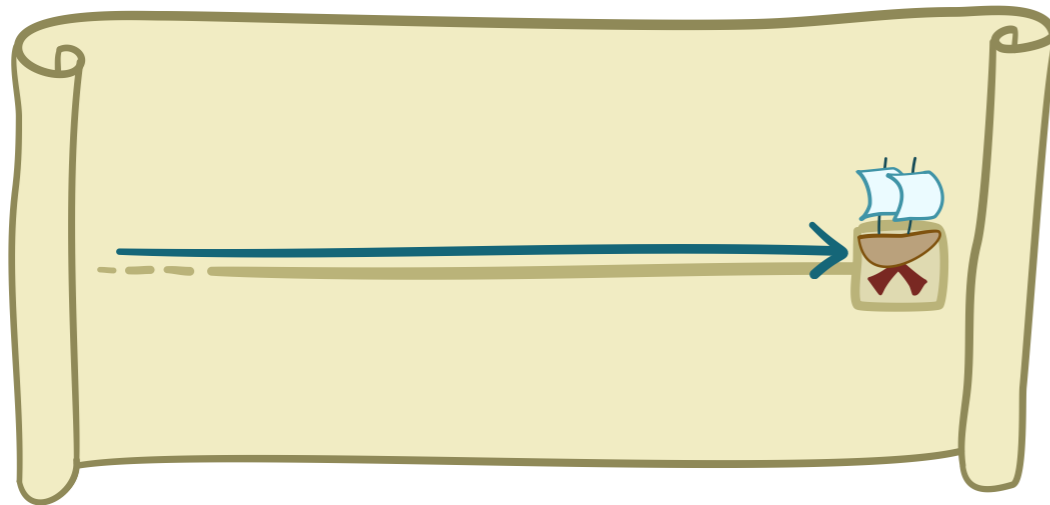


-  general *state space*
-  job's **state** encodes all known info
-  **state** *stochastically evolves* with service
-  completes upon entering **goal state**

Job Examples

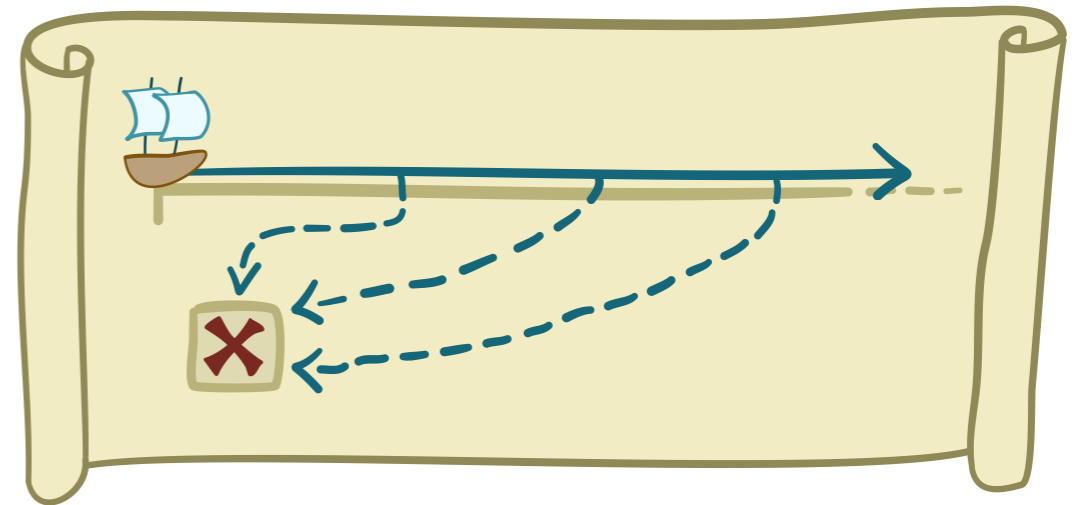
Known Size





state = remaining size



Unknown Size

state = age

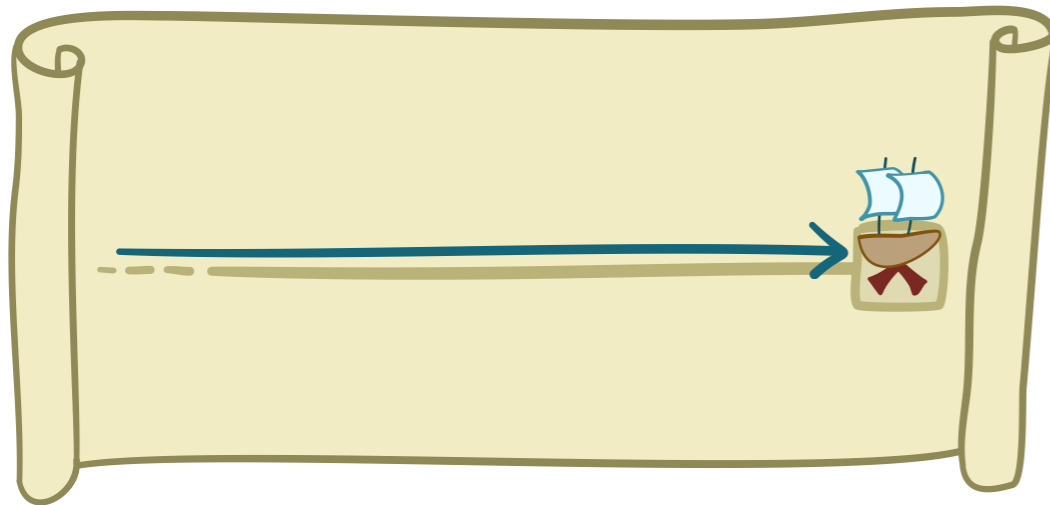


-  general *state space*
-  job's **state** encodes all known info
-  **state** *stochastically evolves* with service
-  completes upon entering **goal state**

Job Examples

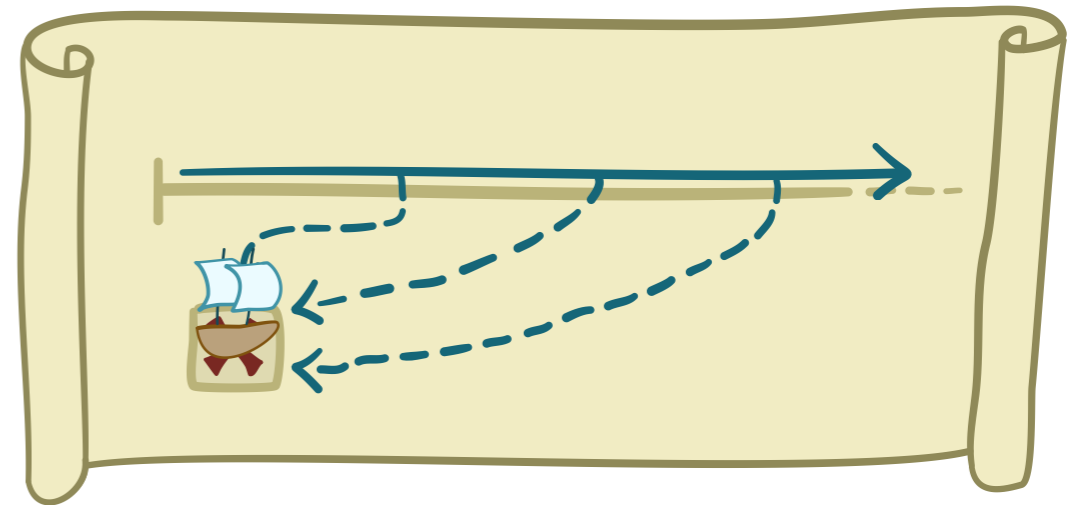
Known Size





state = remaining size



Unknown Size

state = age

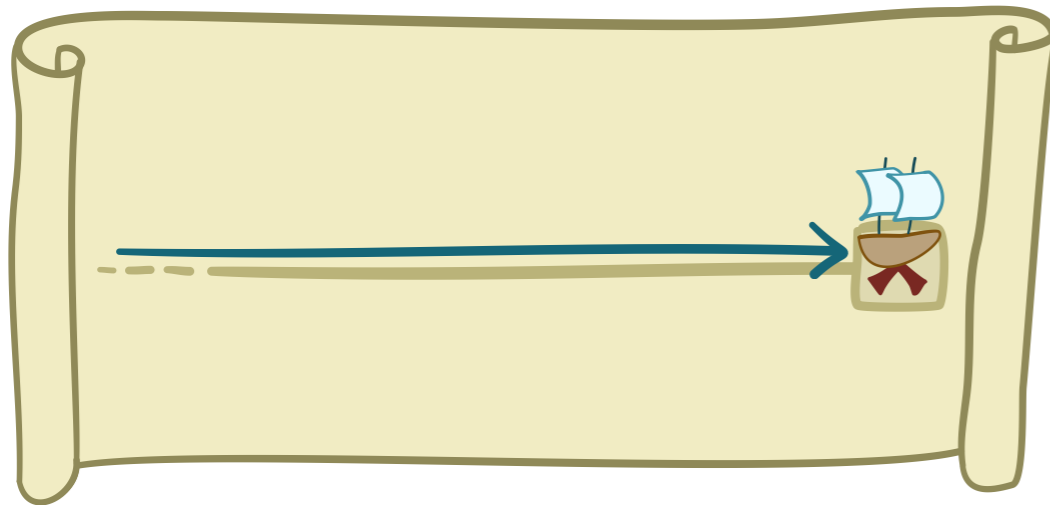


-  general *state space*
-  job's **state** encodes all known info
-  **state** *stochastically evolves* with service
-  completes upon entering **goal state**

Job Examples

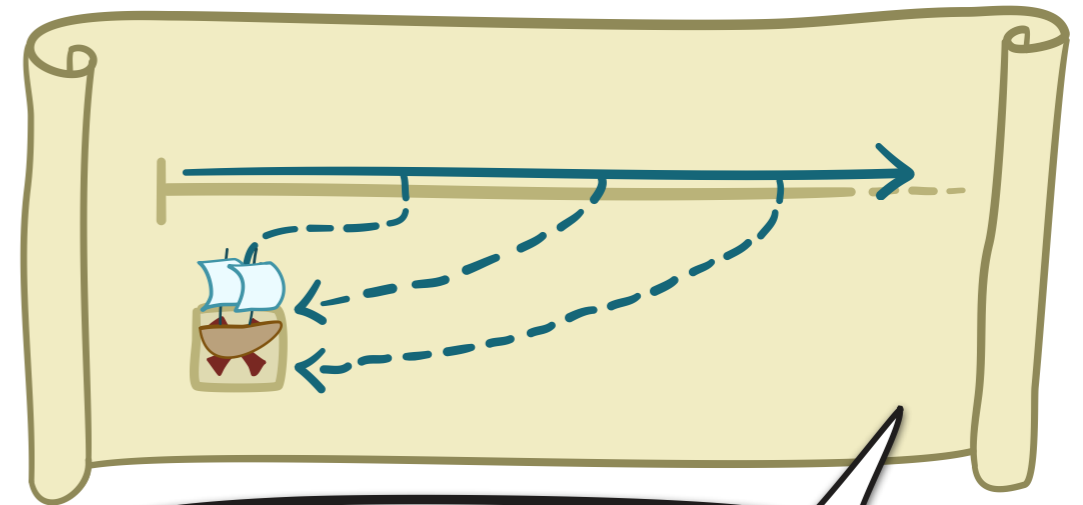
Known Size

state = remaining size







Unknown Size

state = age



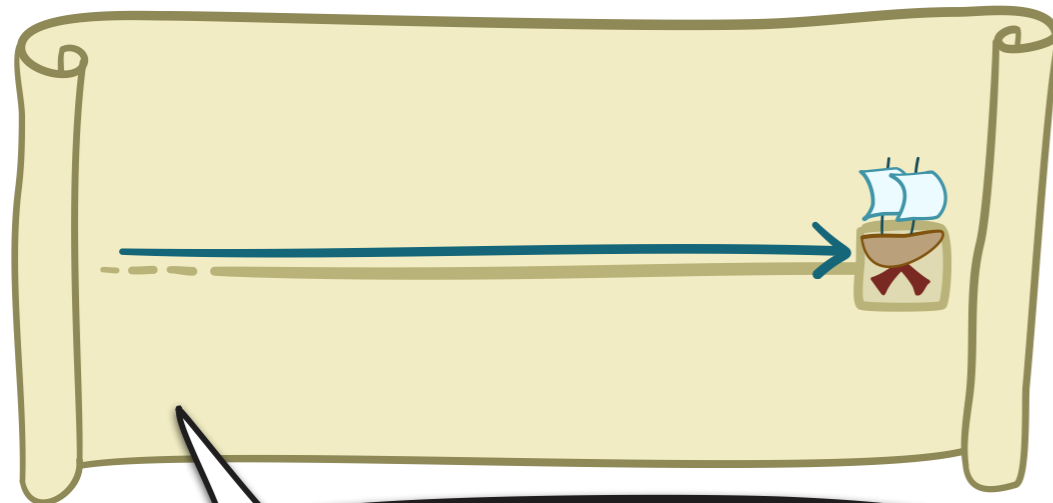
S determines
jump probabilities

-  general *state space*
-  job's **state** encodes all known info
-  **state** *stochastically evolves* with service
-  completes upon entering **goal state**

Job Examples

Known Size

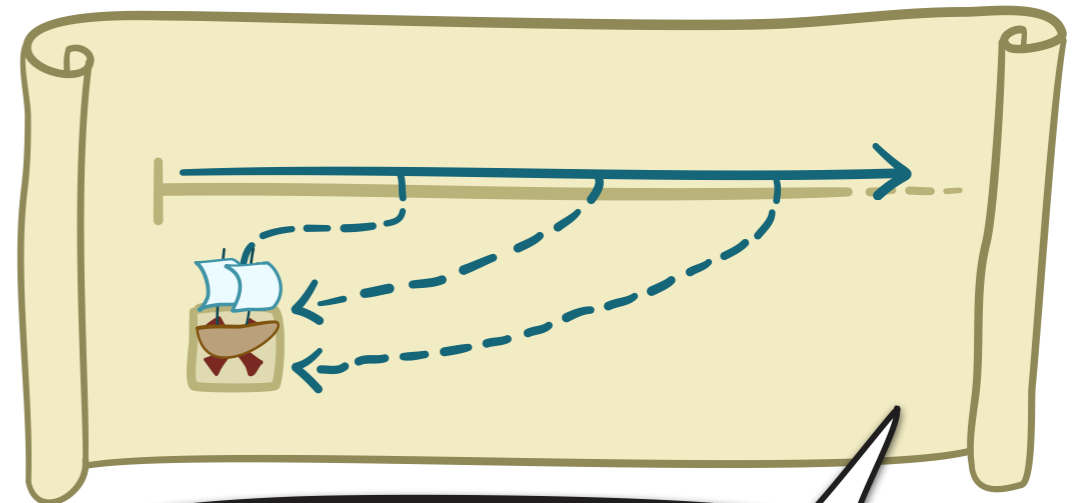
state = remaining size







S determines random initial state

Unknown Size

state = age

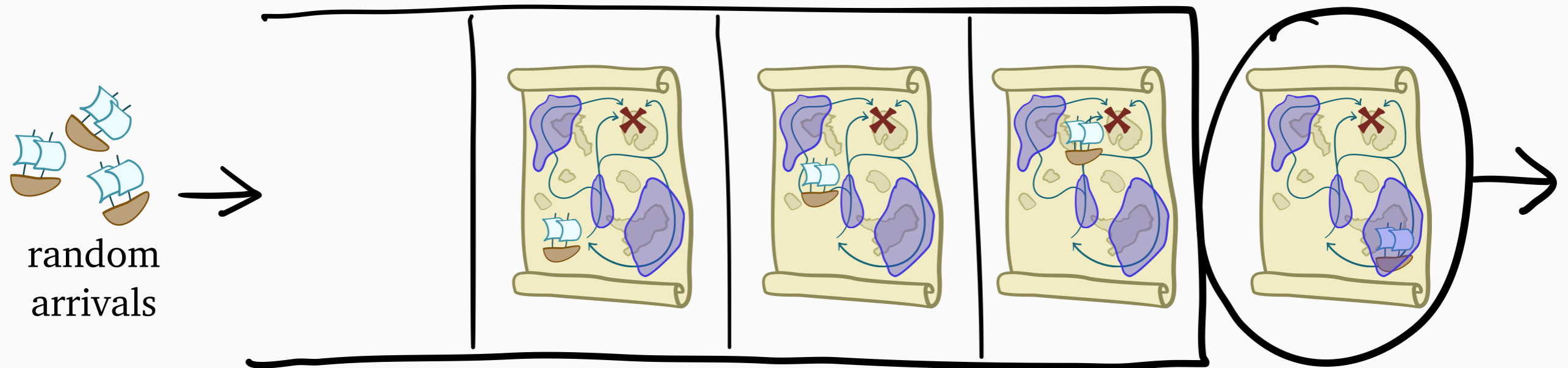


S determines jump probabilities

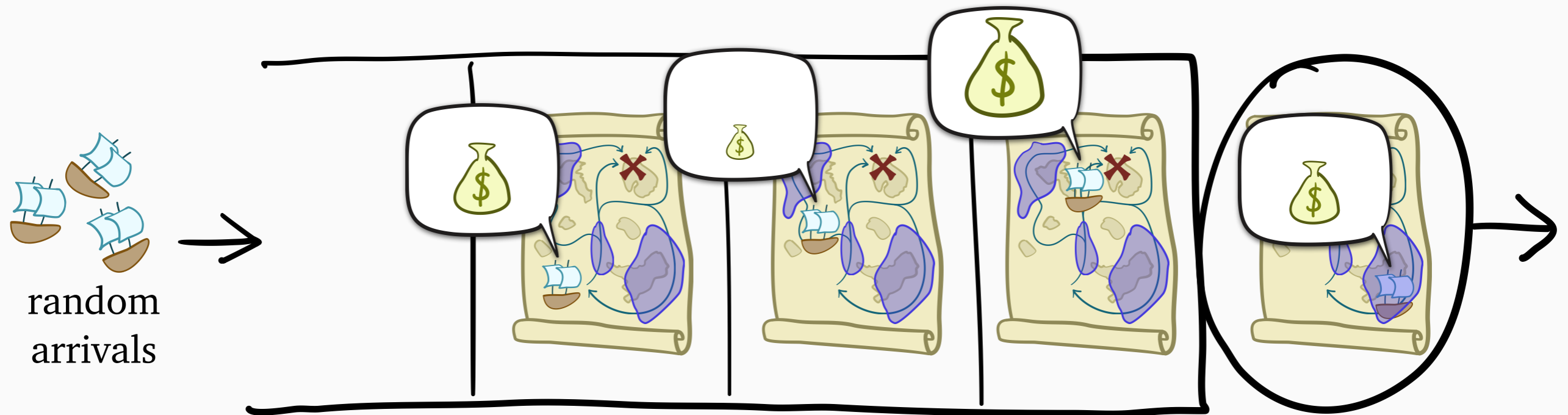
-  general *state space*
-  job's **state** encodes all known info
-  **state** *stochastically evolves* with service
-  completes upon entering **goal state**

Theorem: Gittins minimizes
mean **holding cost** in M/G/1

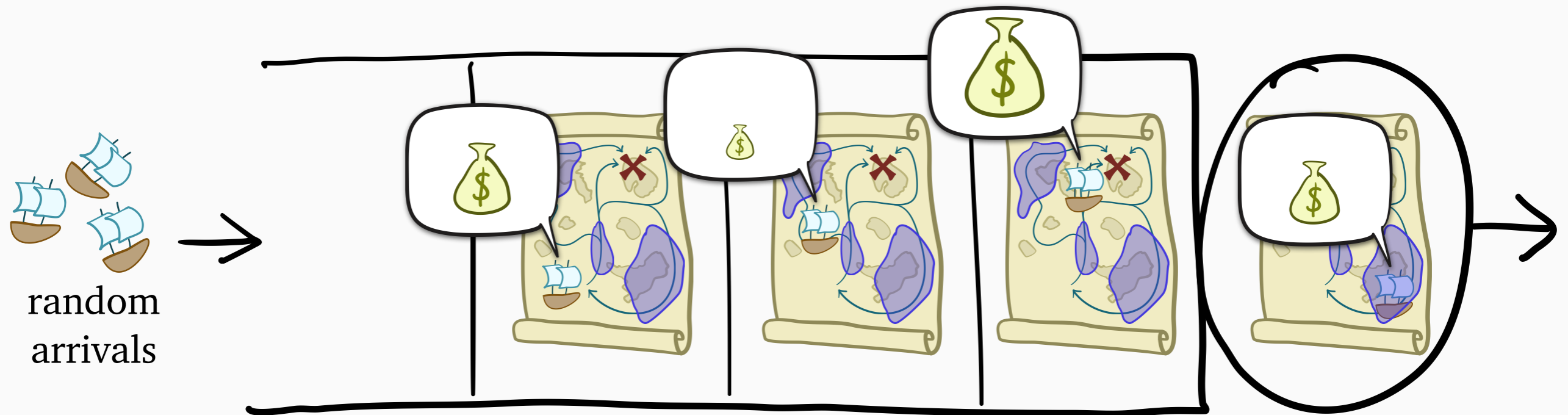
Theorem: Gittins minimizes mean **holding cost** in M/G/1



Theorem: Gittins minimizes mean **holding cost** in M/G/1

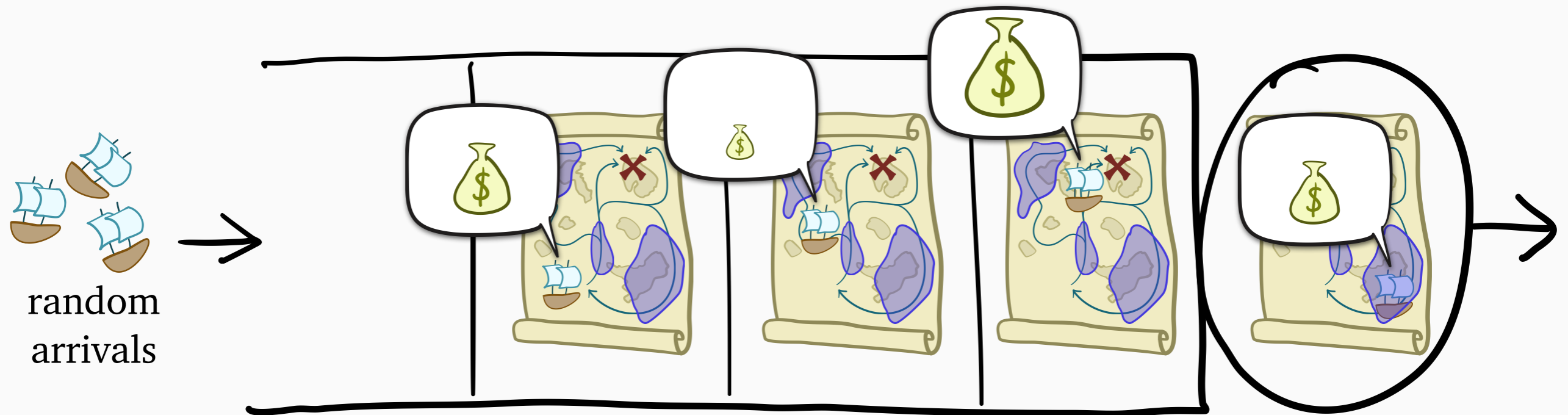


Theorem: Gittins minimizes mean **holding cost** in M/G/1



In paper:

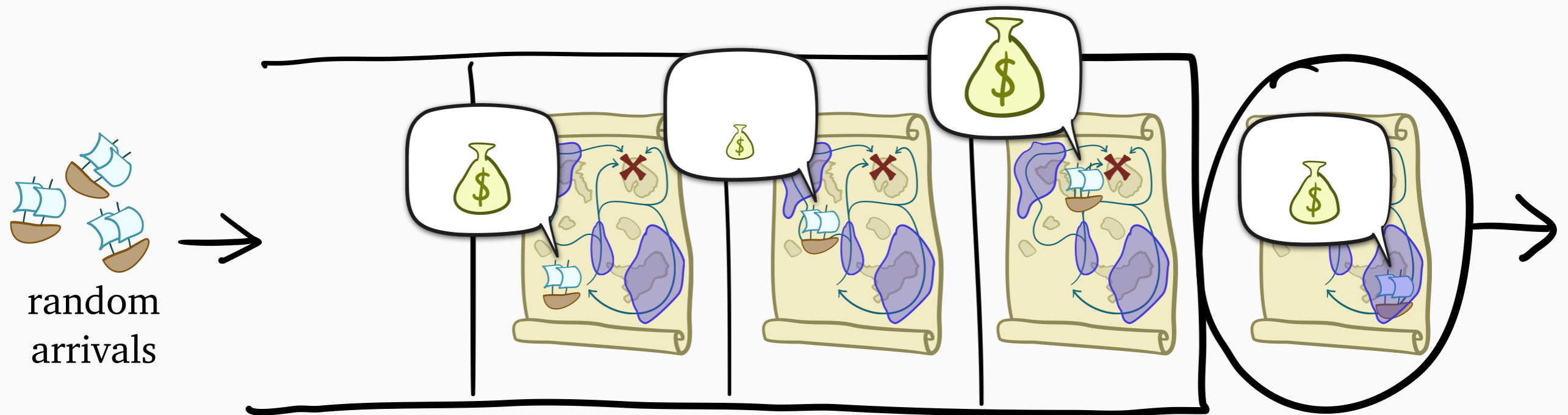
Theorem: Gittins minimizes mean **holding cost** in M/G/1



In paper:

- How to define the **rank** of each **state**

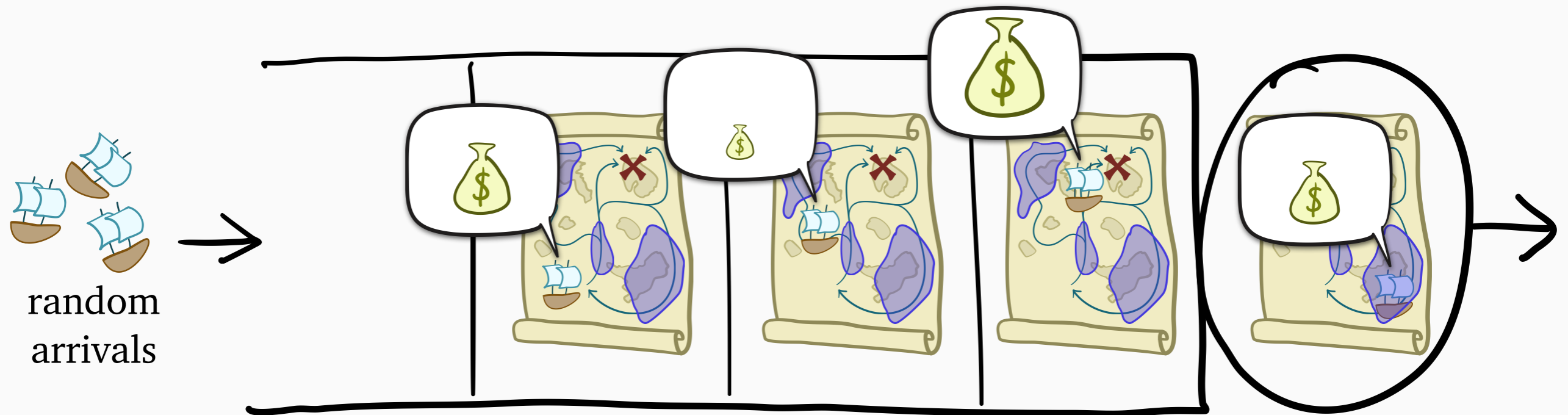
Theorem: Gittins minimizes mean **holding cost** in M/G/1



In paper:

- How to define the **rank** of each **state**
- Review of 10+ prior proofs

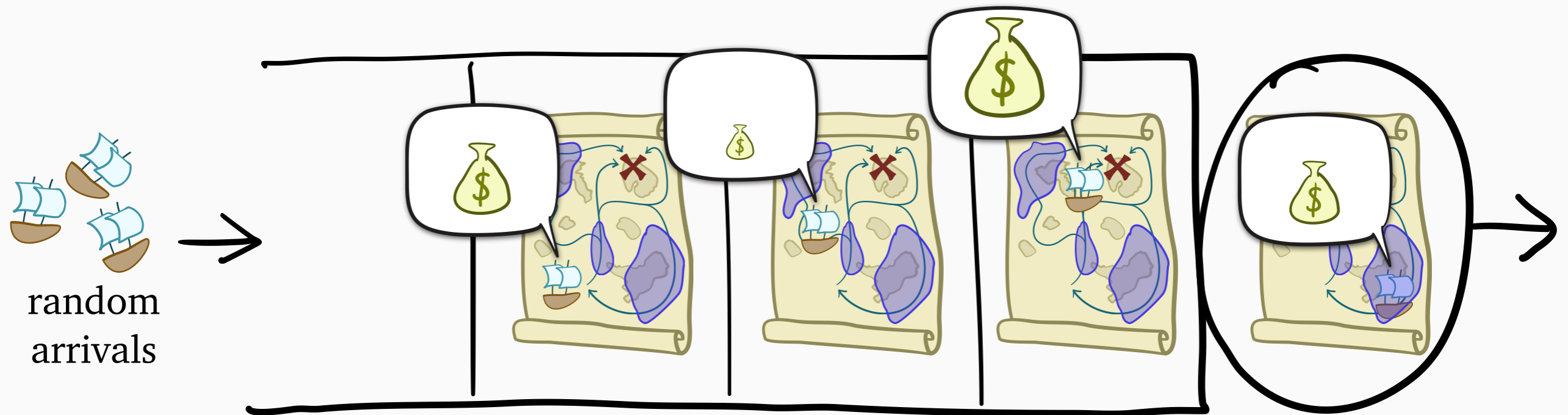
Theorem: Gittins minimizes mean **holding cost** in M/G/1



In paper:

- How to define the **rank** of each **state**
- Review of 10+ prior proofs
- **New proof** that handles general job model

Theorem: Gittins minimizes mean **holding cost** in M/G/1



In paper:

Today: re-prove to define the **rank** of each **state**
review of 10+ prior proofs

- **New proof** that handles general job model



WINE

Work Integral Number Equality



WINE

Work Integral Number Equality

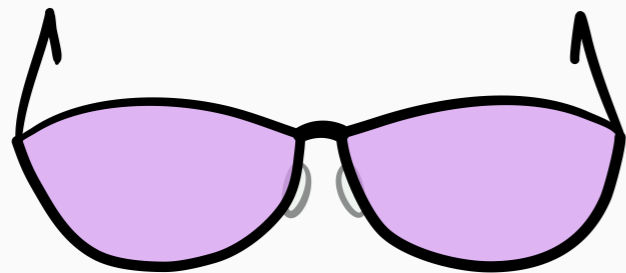


holding cost

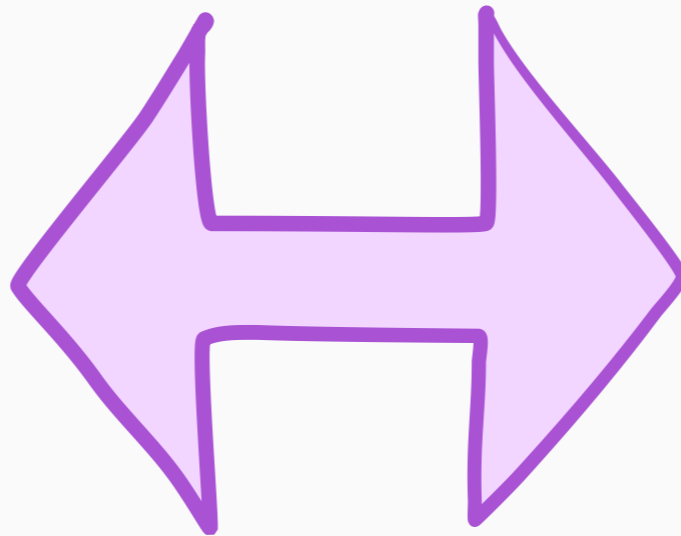


WINE

Work Integral Number Equality



r -work $W(r)$

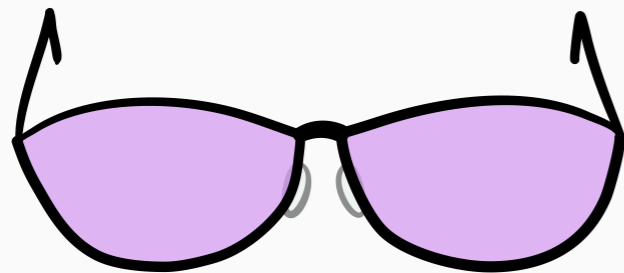


holding cost

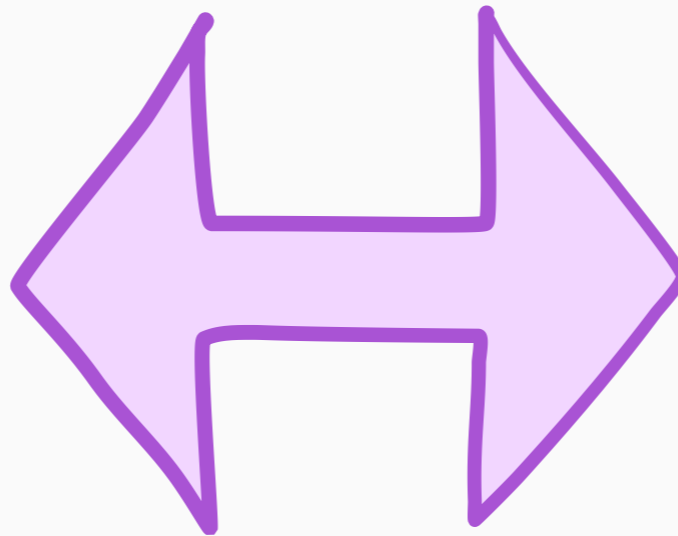


WINE

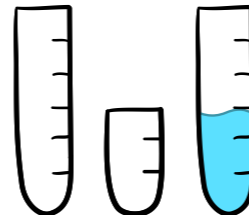
Work Integral Number Equality



r -work $W(r)$



holding cost



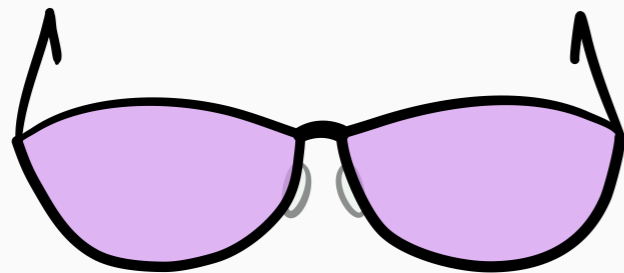
Special case:
number of jobs N



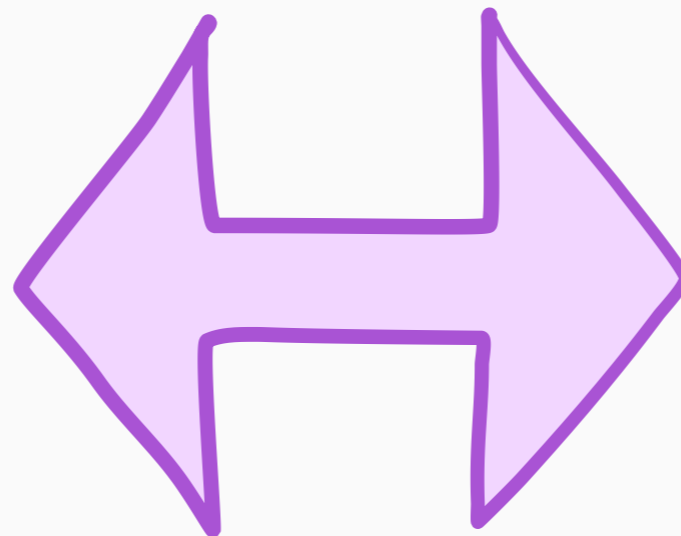
WINE

Work Integral Number Equality

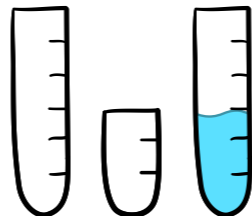
or "Nuisance"



r -work $W(r)$



holding cost

 **Special case:**
number of jobs N

Defining r -work

for SRPT

$W(r)$ = work relevant to rank r

Defining r -work

rank = priority
(lower is better)

for SRPT

$W(r)$ = work relevant to **rank** r

Defining r -work

rank = priority
(lower is better)

for SRPT

$W(r)$ = work relevant to **rank** r

$w_x(r)$ = r -work of *single job* of rem. size x = {



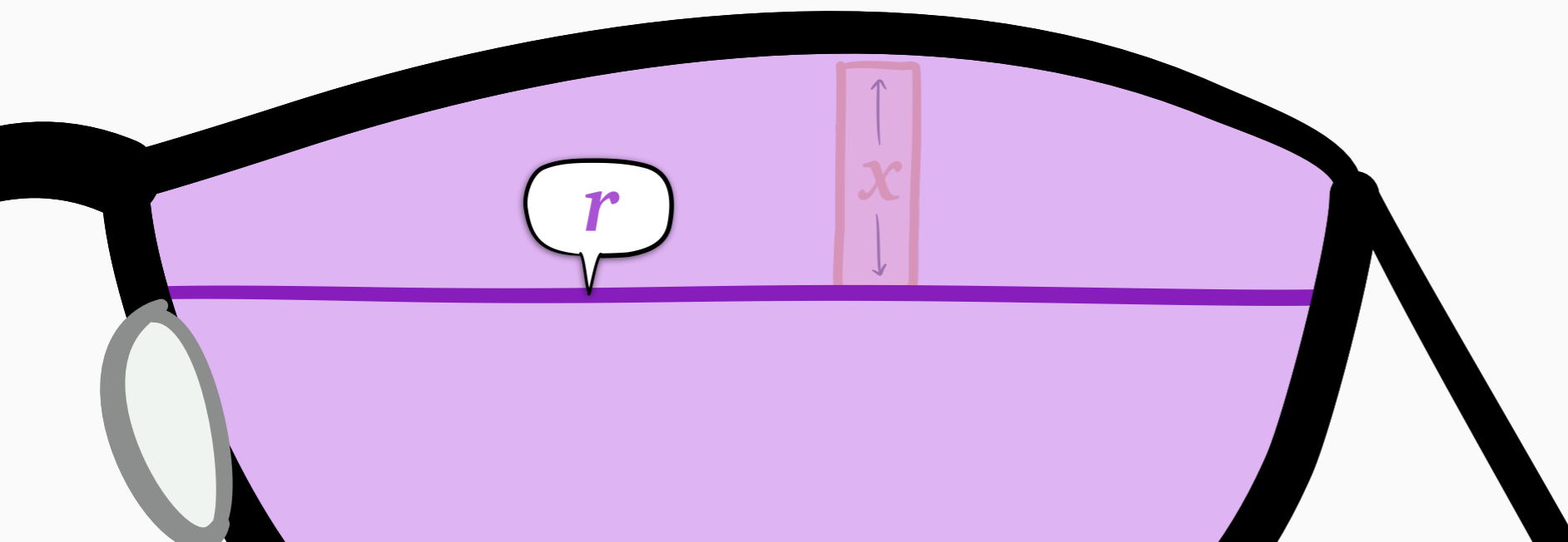
Defining r -work

rank = priority
(lower is better)

for SRPT

$W(r)$ = work relevant to **rank** r

$w_x(r)$ = r -work of *single job* of rem. size x = {



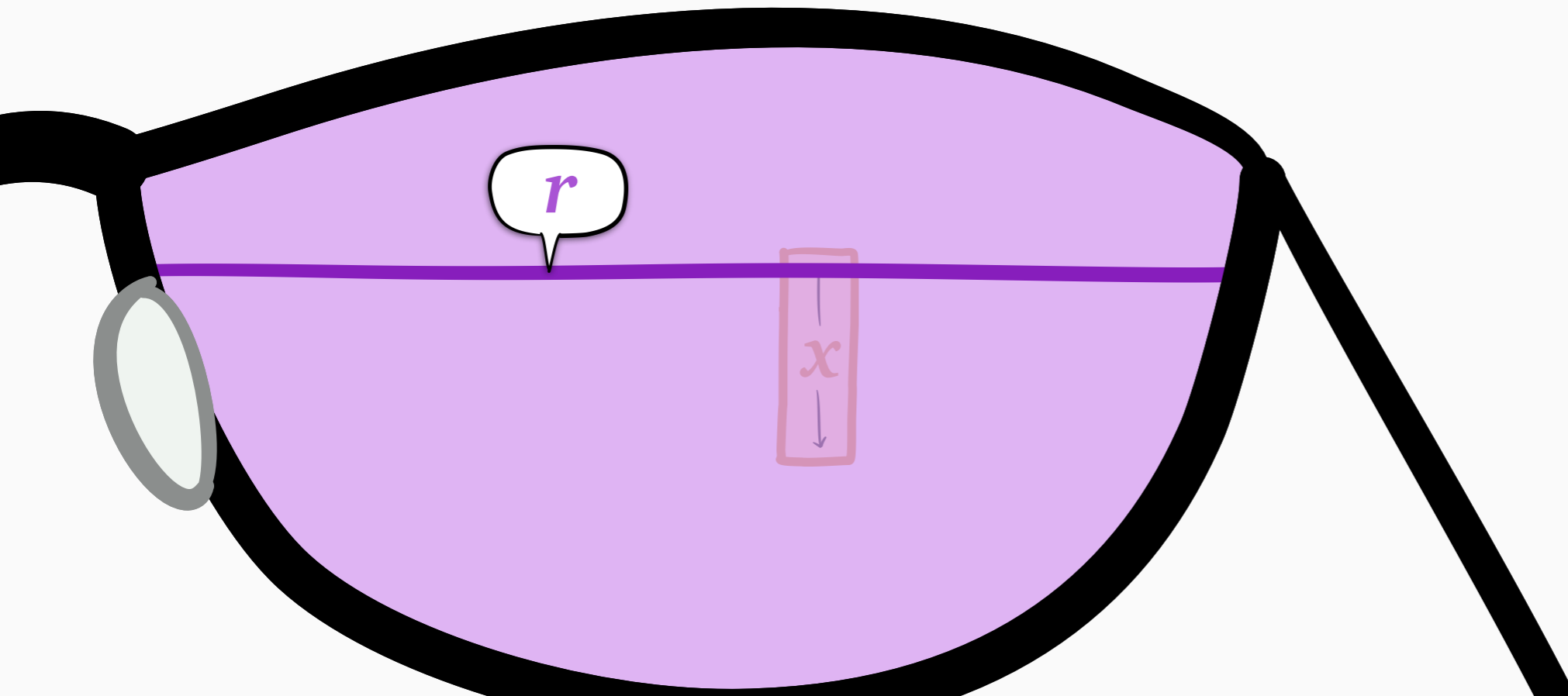
Defining r -work

rank = priority
(lower is better)

for SRPT

$W(r)$ = work relevant to **rank** r

$w_x(r)$ = r -work of *single job* of rem. size x = {



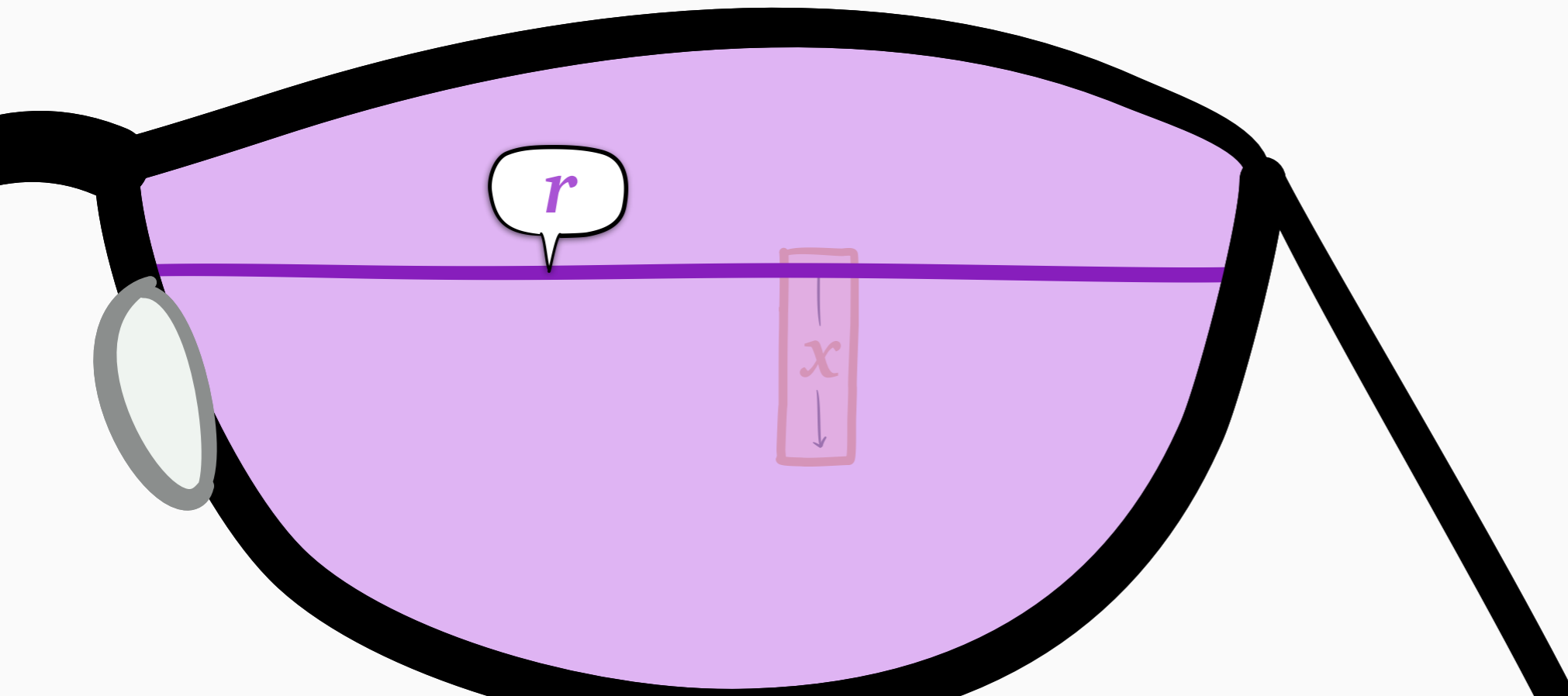
Defining r -work

rank = priority
(lower is better)

for SRPT

$W(r)$ = work relevant to **rank** r

$w_x(r)$ = r -work of *single job* of rem. size x = $\begin{cases} 0 & \text{if } r < x \end{cases}$



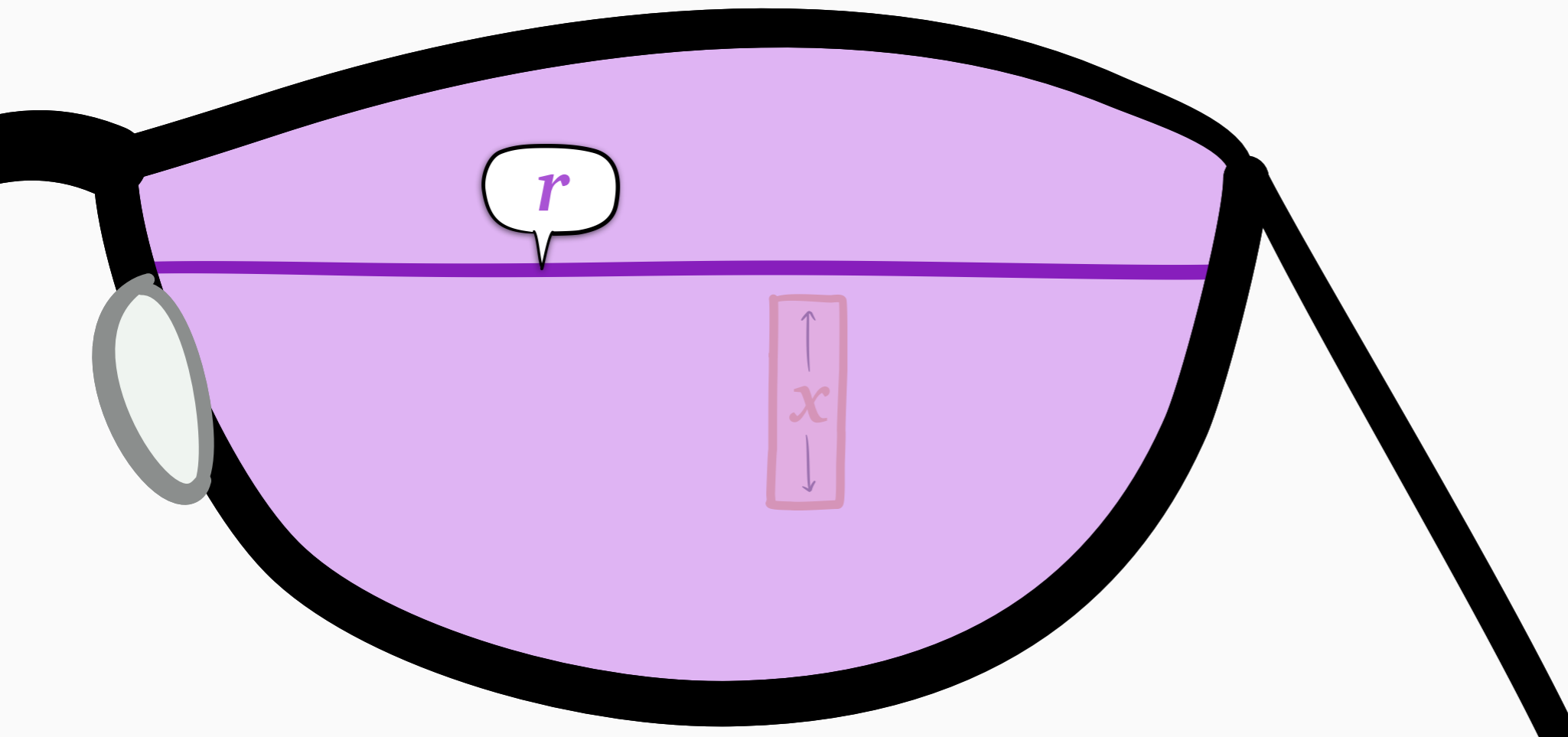
Defining r -work

rank = priority
(lower is better)

for SRPT

$W(r)$ = work relevant to **rank** r

$w_x(r)$ = r -work of *single job* of rem. size x = $\begin{cases} 0 & \text{if } r < x \end{cases}$



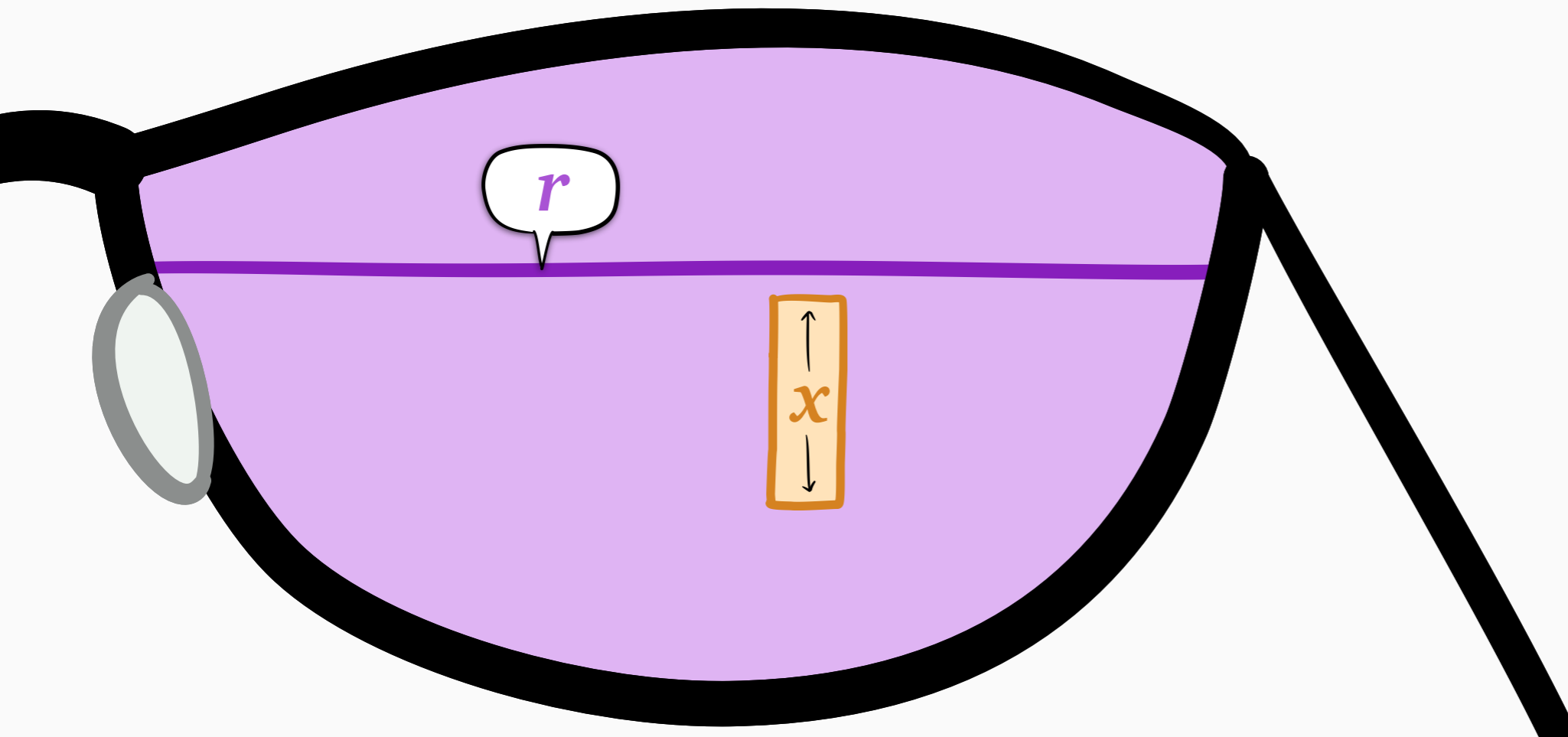
Defining r -work

rank = priority
(lower is better)

for SRPT

$W(r)$ = work relevant to **rank** r

$w_x(r)$ = r -work of *single job* of rem. size x = $\begin{cases} 0 & \text{if } r < x \end{cases}$



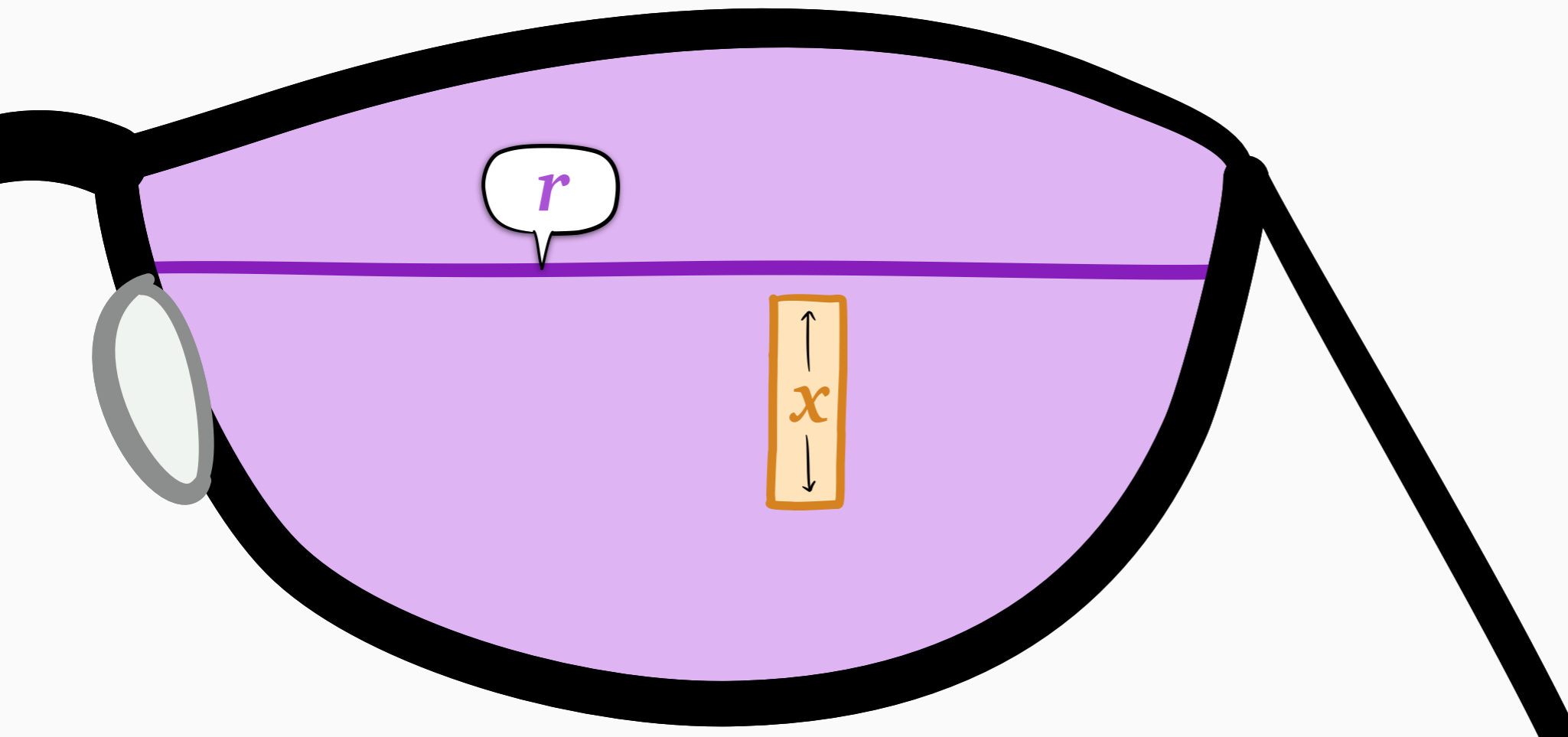
Defining r -work

rank = priority
(lower is better)

for SRPT

$W(r)$ = work relevant to **rank** r

$w_x(r)$ = r -work of *single job* of rem. size x = $\begin{cases} 0 & \text{if } r < x \\ x & \text{if } r \geq x \end{cases}$



Defining r -work

rank = priority
(lower is better)

for SRPT

$W(r)$ = work relevant to **rank** r

$w_x(r)$ = r -work of *single job* of rem. size x = $\begin{cases} 0 & \text{if } r < x \\ x & \text{if } r \geq x \end{cases}$



Defining r -work

rank = priority
(lower is better)

for SRPT

$W(r)$ = work relevant to **rank** r
= total r -work of all jobs

$w_x(r)$ = r -work of *single job* of rem. size x = $\begin{cases} 0 & \text{if } r < x \\ x & \text{if } r \geq x \end{cases}$



From r -work to number of jobs N

From r -work to number of jobs N

Goal: integral = N

$W(r)$



From r -work to number of jobs N

Goal: integral = N

$W(r)$



Suffices: integral = 1

$w_x(r)$

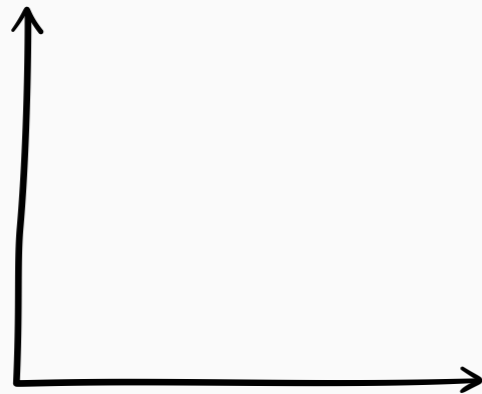


$$w_x(r) = r\text{-work of job of rem. size } x = \begin{cases} 0 & \text{if } r < x \\ x & \text{if } r \geq x \end{cases}$$

From r -work to number of jobs N

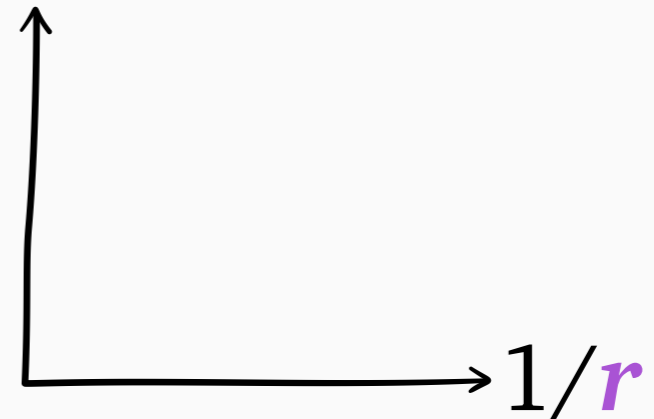
Goal: integral = N

$W(r)$



Suffices: integral = 1

$w_x(r)$

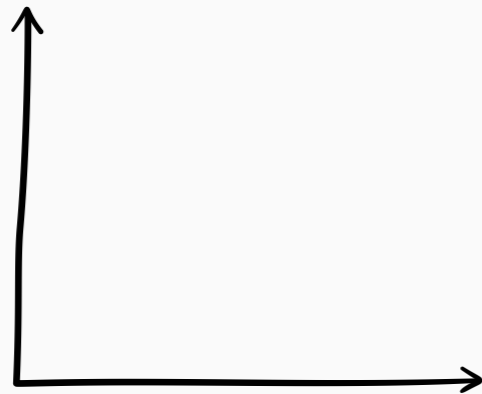


$$w_x(r) = r\text{-work of job of rem. size } x = \begin{cases} 0 & \text{if } r < x \\ x & \text{if } r \geq x \end{cases}$$

From r -work to number of jobs N

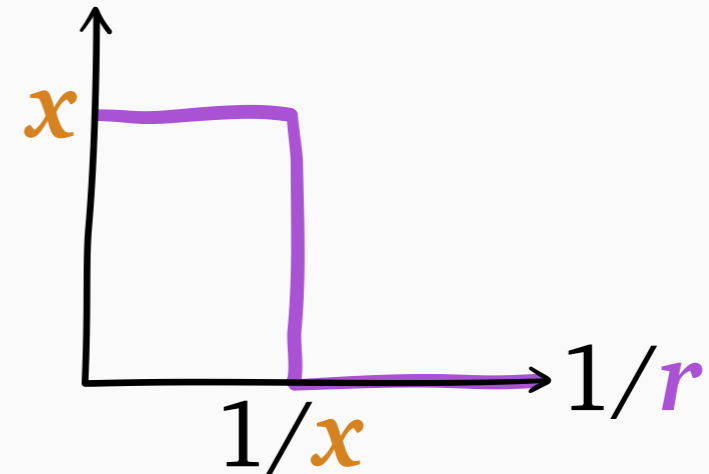
Goal: integral = N

$W(r)$



Suffices: integral = 1

$w_x(r)$

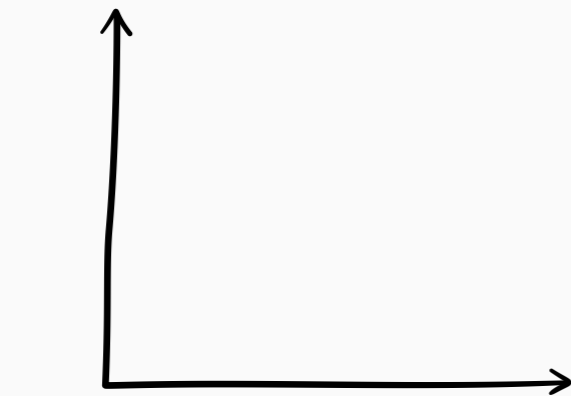


$$w_x(r) = r\text{-work of job of rem. size } x = \begin{cases} 0 & \text{if } r < x \\ x & \text{if } r \geq x \end{cases}$$

From r -work to number of jobs N

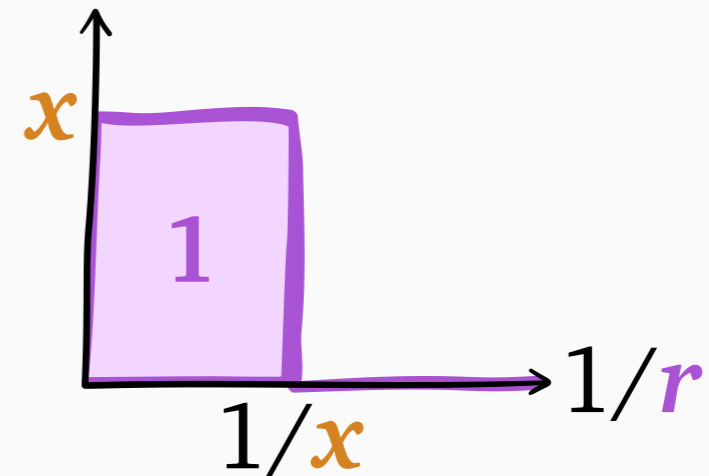
Goal: integral = N

$W(r)$



Suffices: integral = 1

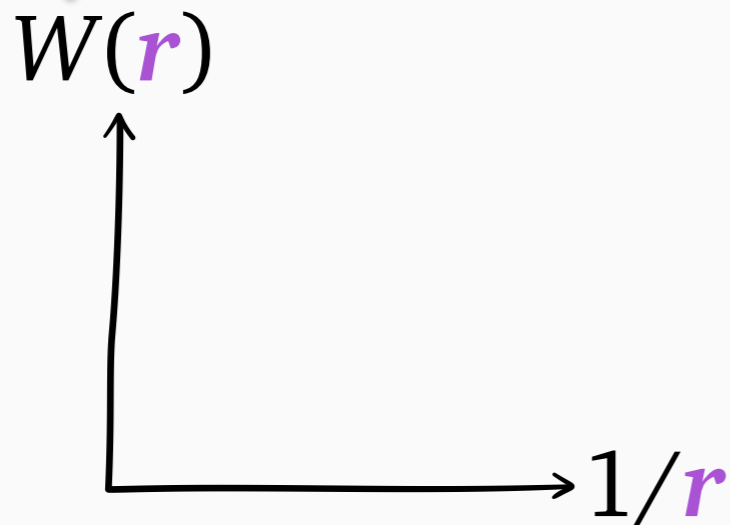
$w_x(r)$



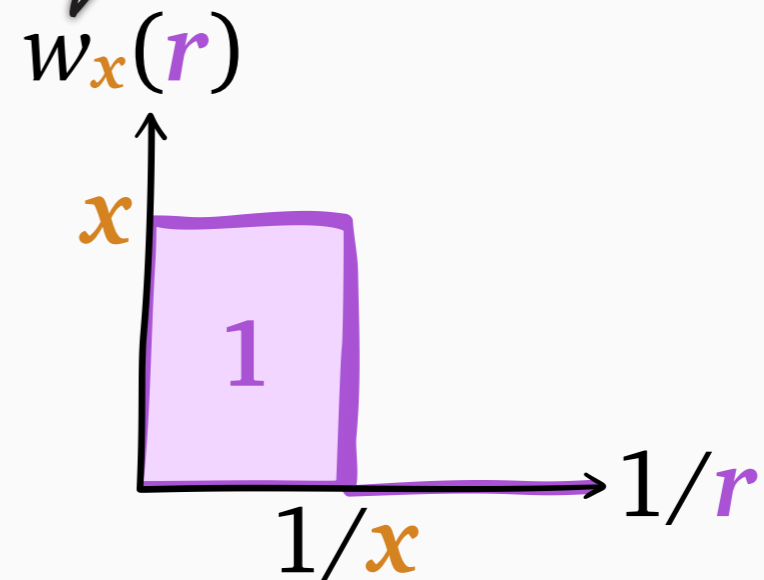
$$w_x(r) = r\text{-work of job of rem. size } x = \begin{cases} 0 & \text{if } r < x \\ x & \text{if } r \geq x \end{cases}$$

From r -work to number of jobs N

Goal: integral = N



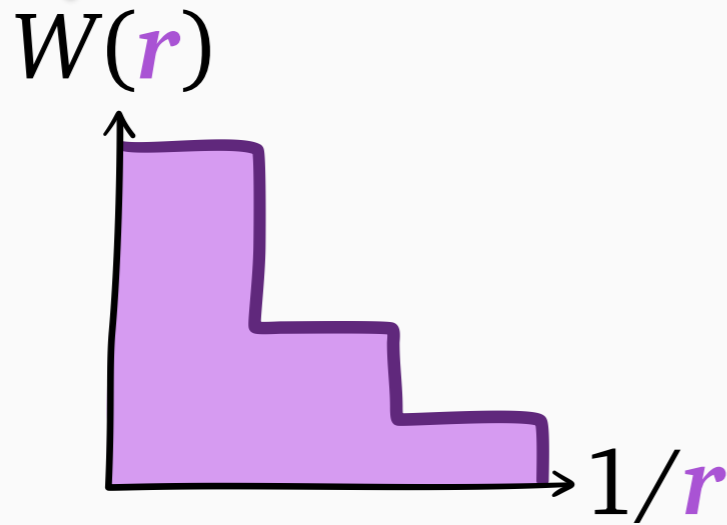
Suffices: integral = 1



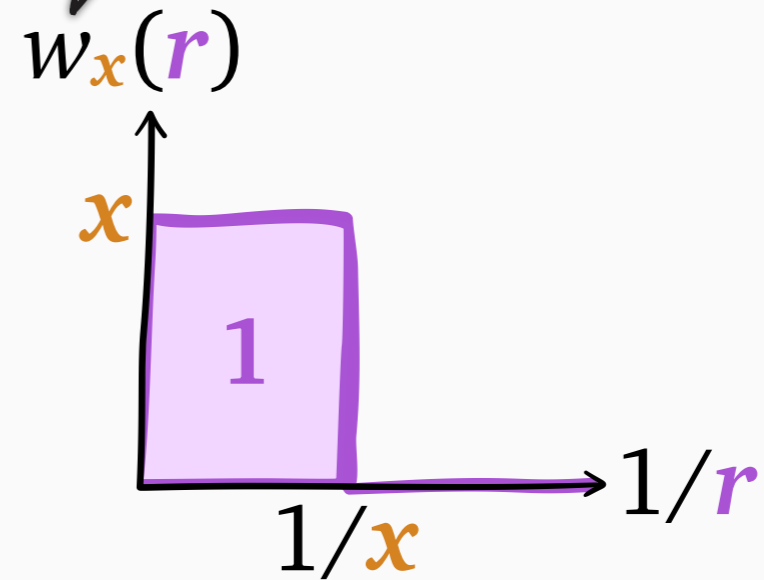
$$w_x(r) = r\text{-work of job of rem. size } x = \begin{cases} 0 & \text{if } r < x \\ x & \text{if } r \geq x \end{cases}$$

From r -work to number of jobs N

Goal: integral = N



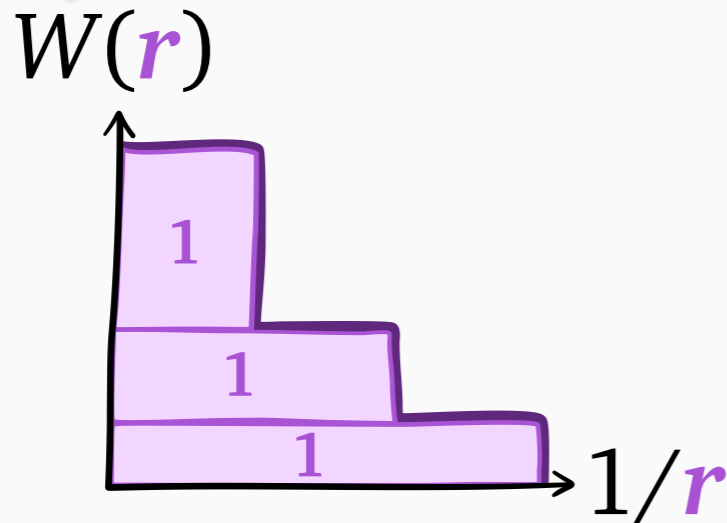
Suffices: integral = 1



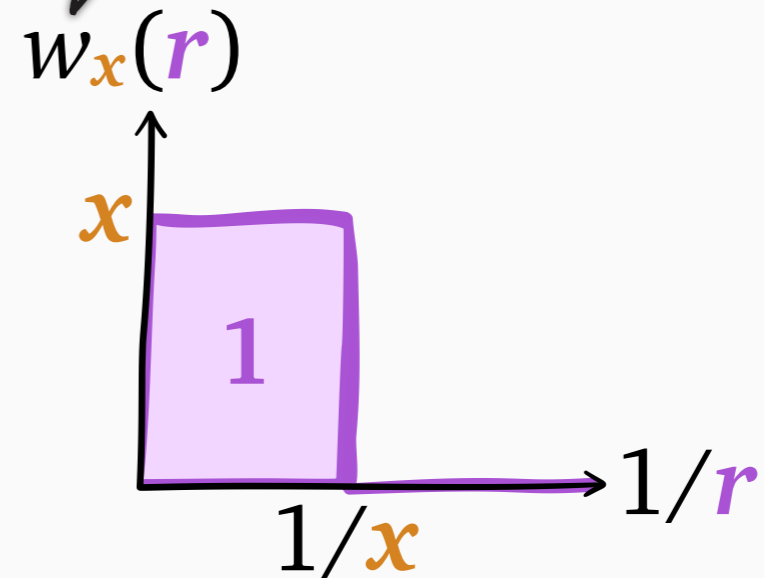
$$w_x(r) = r\text{-work of job of rem. size } x = \begin{cases} 0 & \text{if } r < x \\ x & \text{if } r \geq x \end{cases}$$

From r -work to number of jobs N

Goal: integral = N



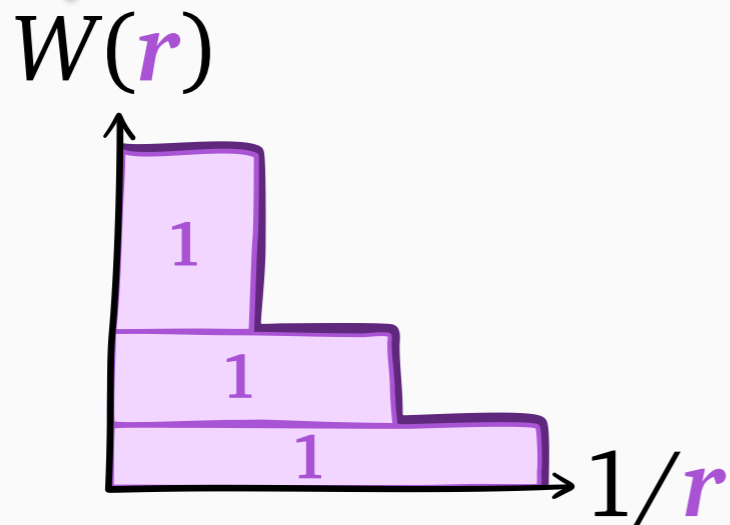
Suffices: integral = 1



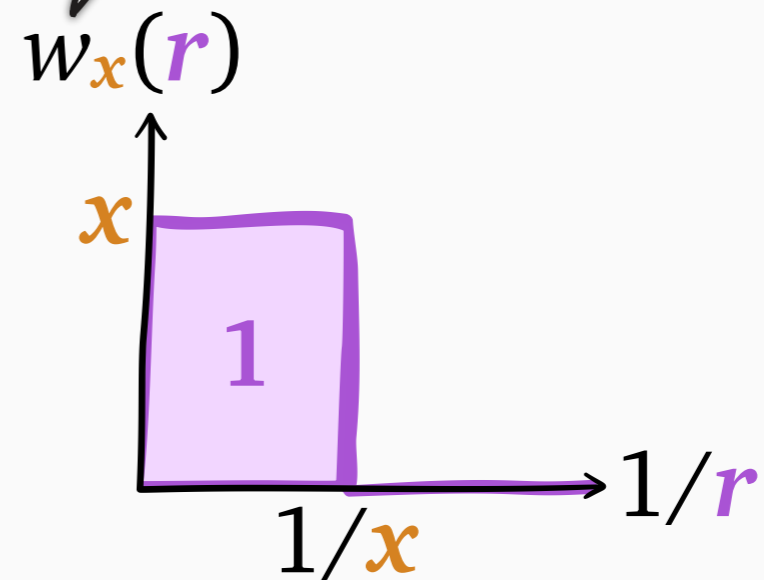
$$w_x(r) = r\text{-work of job of rem. size } x = \begin{cases} 0 & \text{if } r < x \\ x & \text{if } r \geq x \end{cases}$$

From r -work to number of jobs N

Goal: integral = N



Suffices: integral = 1

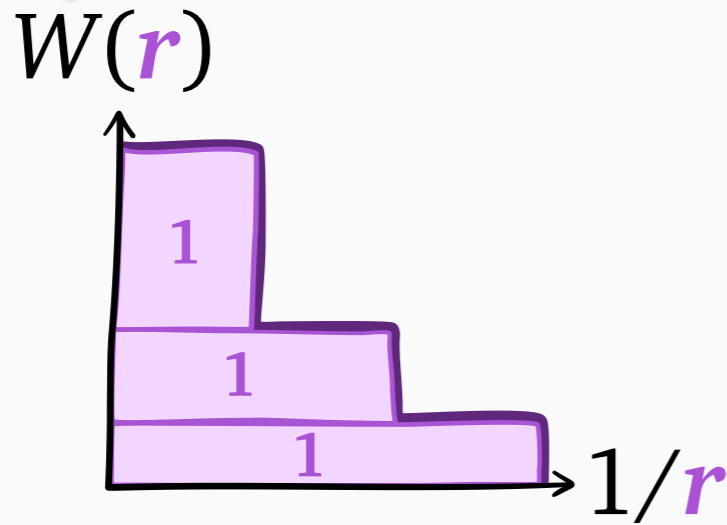


Theorem:

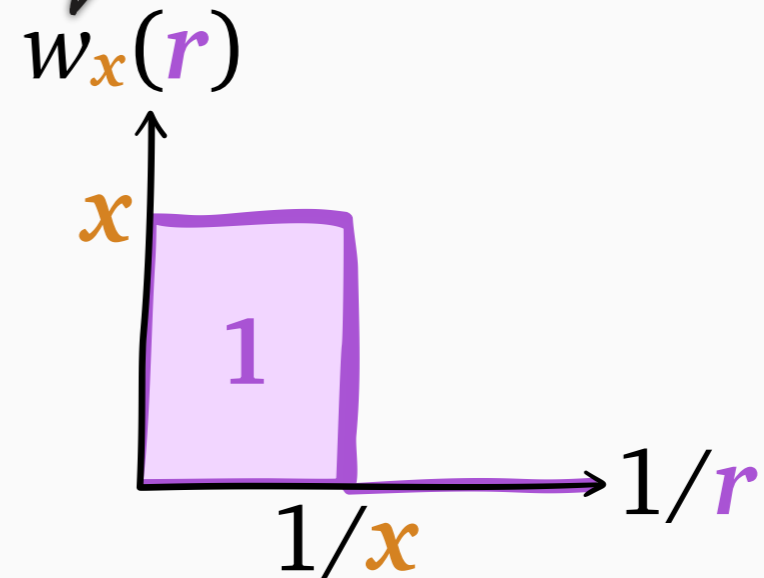
$$N = \int_0^{\infty} \frac{W(r)}{r^2} dr$$

From r -work to number of jobs N

Goal: integral = N



Suffices: integral = 1



Theorem:

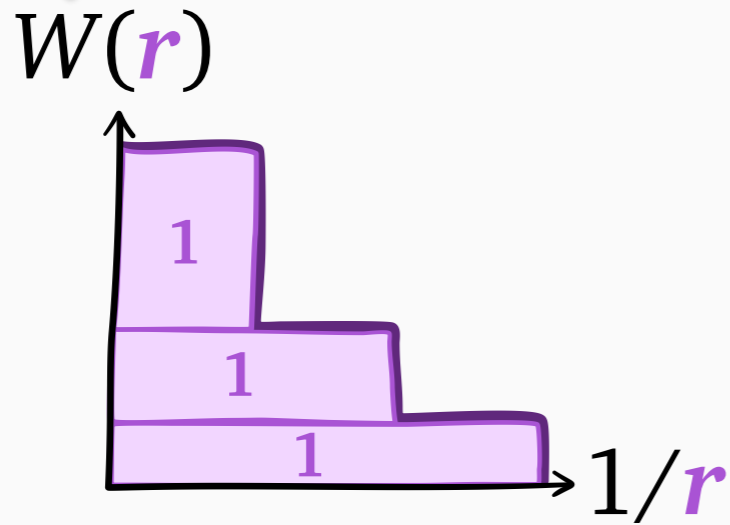
$$N = \int_0^{\infty} \frac{W(r)}{r^2} dr$$



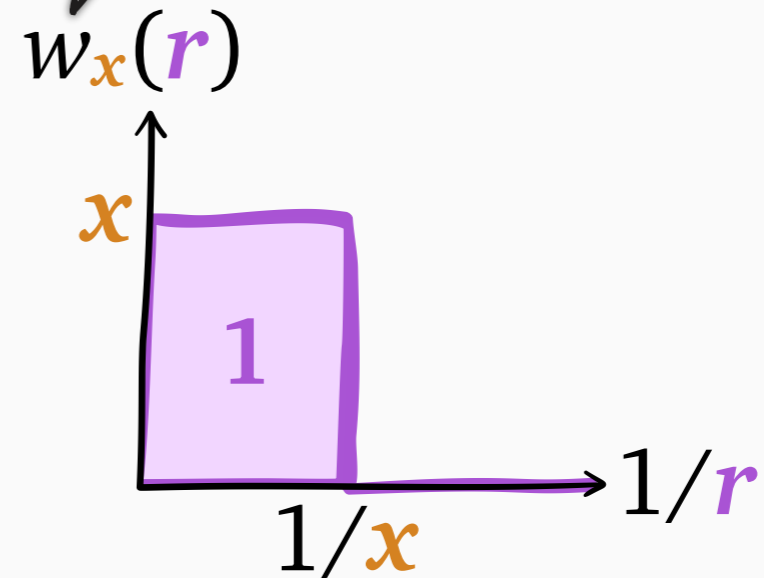
WINE

From r -work to number of jobs N

Goal: integral = N



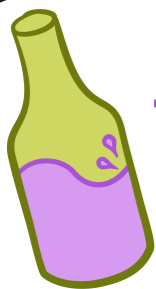
Suffices: integral = 1



Theorem:

$$N = \int_0^{\infty} \frac{W(r)}{r^2} dr$$

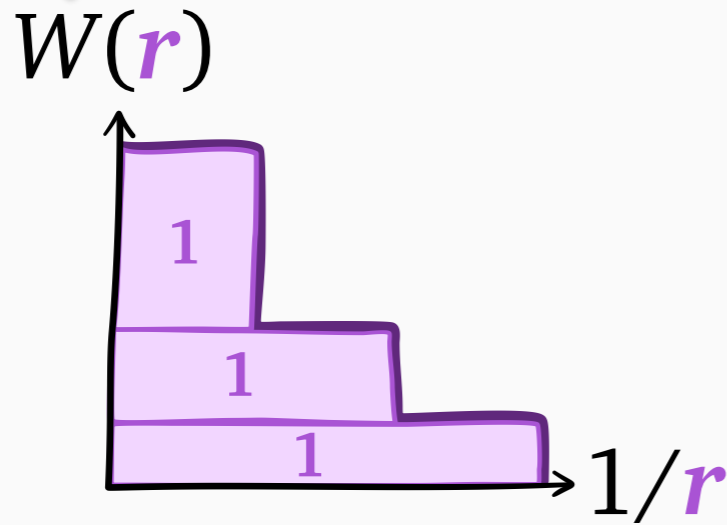
uses rank = rem. size



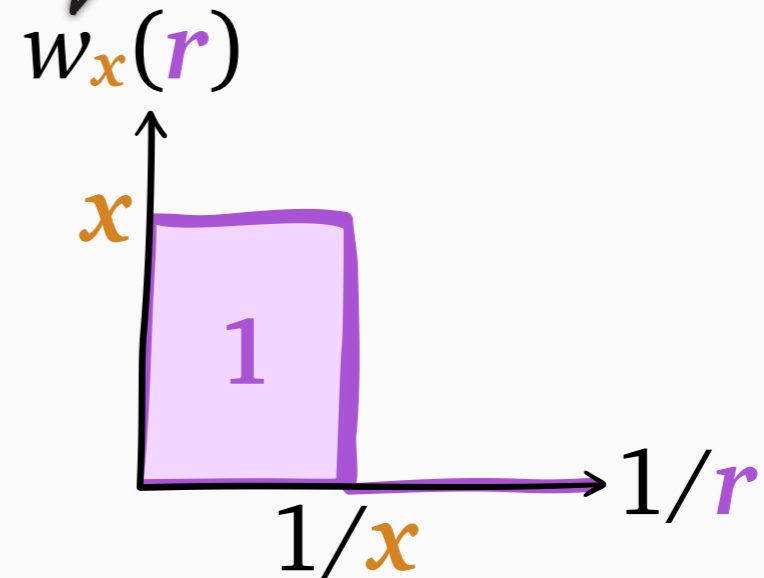
WINE

From r -work to number of jobs N

Goal: integral = N



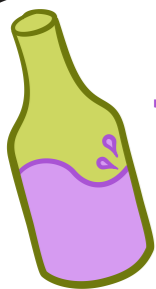
Suffices: integral = 1



Theorem: under *any* policy,

$$N = \int_0^{\infty} \frac{W(r)}{r^2} dr$$

uses **rank** = rem. size



WINE

Optimality of SRPT

Theorem: under *any* policy,



$$N = \int_0^{\infty} \frac{W(r)}{r^2} dr$$

Optimality of SRPT

Theorem: under *any* policy,



$$N = \int_0^{\infty} \frac{W(r)}{r^2} dr$$

Why does SRPT minimize $E[N]$ in M/G/1?

Optimality of SRPT

Theorem: under *any* policy,

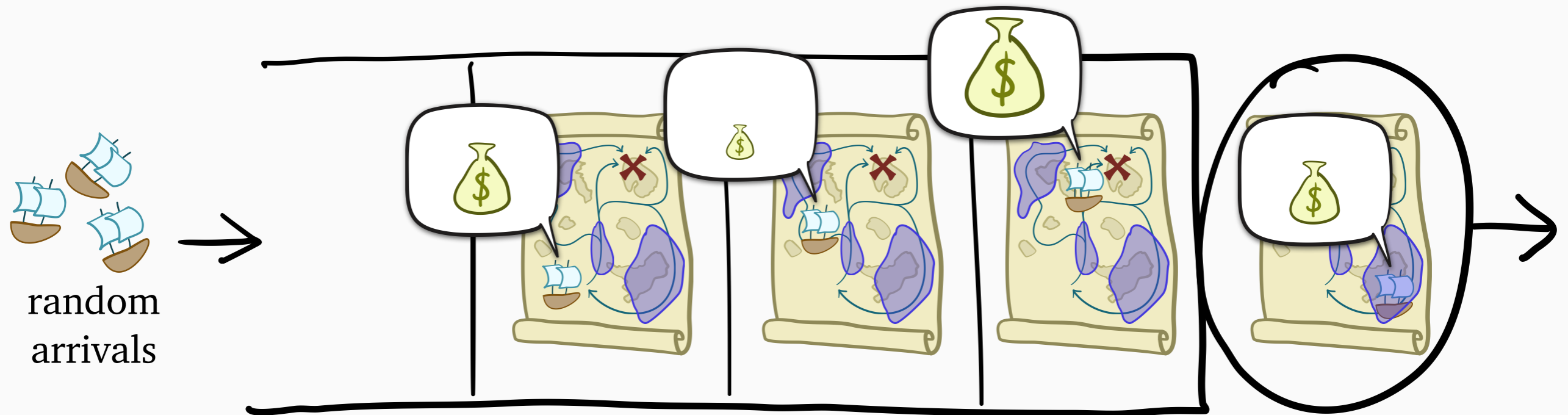


$$N = \int_0^{\infty} \frac{W(r)}{r^2} dr$$

Why does SRPT minimize $E[N]$ in M/G/1?

Because it minimizes $E[W(r)]$ for all r !

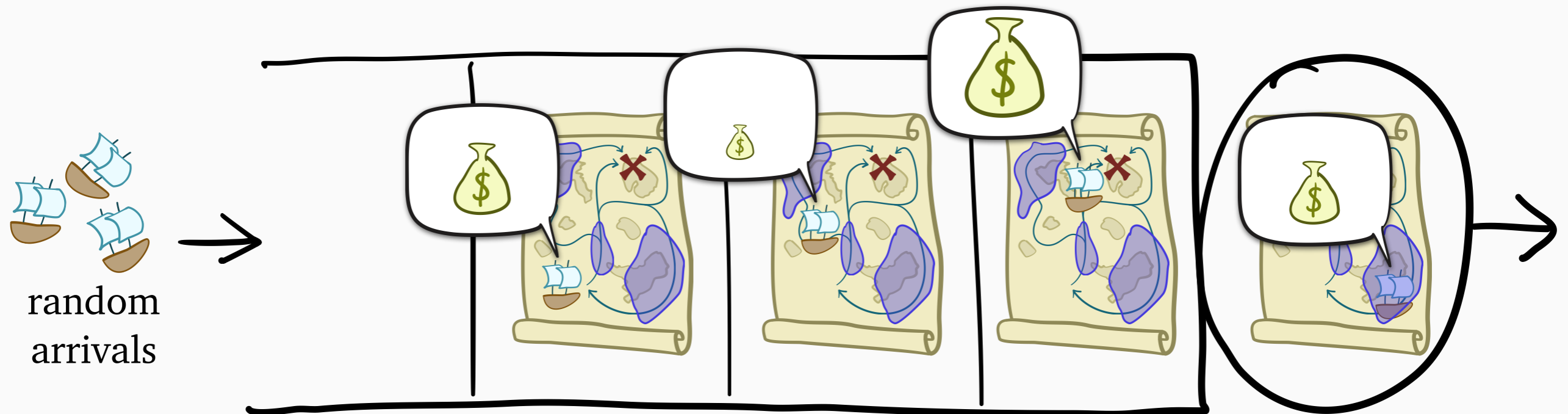
Theorem: Gittins minimizes mean **holding cost** in M/G/1



In paper:

- How to define the **rank** of each **state**
- Review of 10+ prior proofs
- **New proof** that handles general job model

Theorem: Gittins minimizes mean **holding cost** in M/G/1



In paper:

- How to define the **rank** of each **state**
- Review of 10+ prior proofs
- **New proof** that handles general job model

