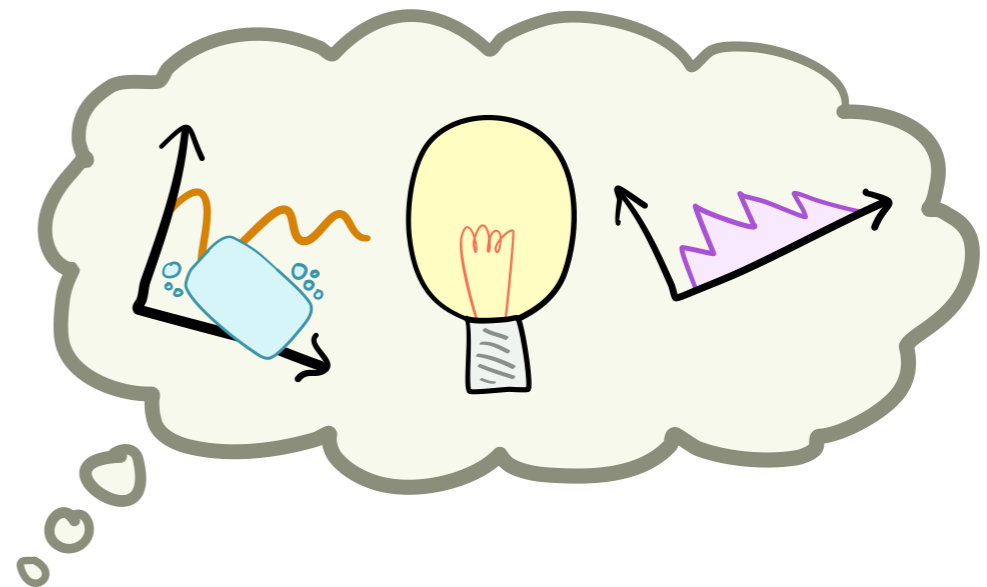


How to Schedule Near-Optimally *under* **Real-World Constraints**

Ziv Scully
Carnegie Mellon University



Collaborators



Mor Harchol-Balter (CMU)



Alan Scheller-Wolf (CMU)



Isaac Grosf (CMU)

Adam Wierman (Caltech)

Onno Boxma (TU/e)

Jan-Pieter Dorsman (UvA)

Lucas van Kreveld (UvA)

Queues in Computer Systems

Queues in Computer Systems

↖ Queueing system: *jobs* waiting for *service*

Queues in Computer Systems

 Queueing system: *jobs* waiting for *service*

- File servers
 - *Jobs*: file requests
 - *Service*: load and send contents

Queues in Computer Systems

↖ Queueing system: *jobs* waiting for *service*

- File servers
 - *Jobs*: file requests
 - *Service*: load and send contents
- Databases
 - *Jobs*: SQL queries
 - *Service*: execute and send result

Queues in Computer Systems

 Queueing system: *jobs* waiting for *service*

- File servers
 - *Jobs*: file requests
 - *Service*: load and send contents
- Databases
 - *Jobs*: SQL queries
 - *Service*: execute and send result
- Network switches
 - *Jobs*: packet flows
 - *Service*: transmit all packets

Queues in Computer Systems

 Queueing system: *jobs* waiting for *service*

- File servers
 - *Jobs*: file requests
 - *Service*: load and send contents
- Databases
 - *Jobs*: SQL queries
 - *Service*: execute and send result
- Network switches
 - *Jobs*: packet flows
 - *Service*: transmit all packets
- Operating systems
 - *Jobs*: threads
 - *Service*: run on a CPU core

Queues in Computer Systems

↖ Queueing system: *jobs* waiting for *service*

- File servers
 - *Jobs*: file requests
 - *Service*: load and send contents
- Databases
 - *Jobs*: SQL queries
 - *Service*: execute and send result
- Network switches
 - *Jobs*: packet flows
 - *Service*: transmit all packets
- Operating systems
 - *Jobs*: threads
 - *Service*: run on a CPU core

Queueing theory: studies the *mathematical essence* of queueing systems

Queues in Computer Systems

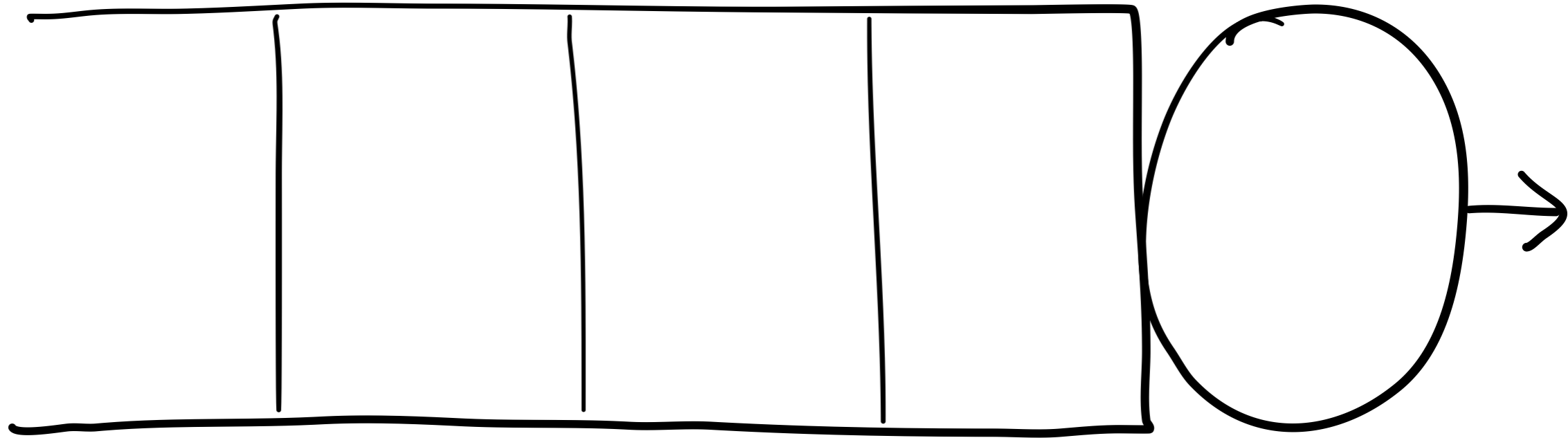
↖ Queueing system: *jobs* waiting for *service*

- File servers
 - *Jobs*: file requests
 - *Service*: load and send contents
- Databases
 - *Jobs*: SQL queries
 - *Service*: execute and send result
- Network switches
 - *Jobs*: packet flows
 - *Service*: transmit all packets
- Operating systems
 - *Jobs*: threads
 - *Service*: run on a CPU core

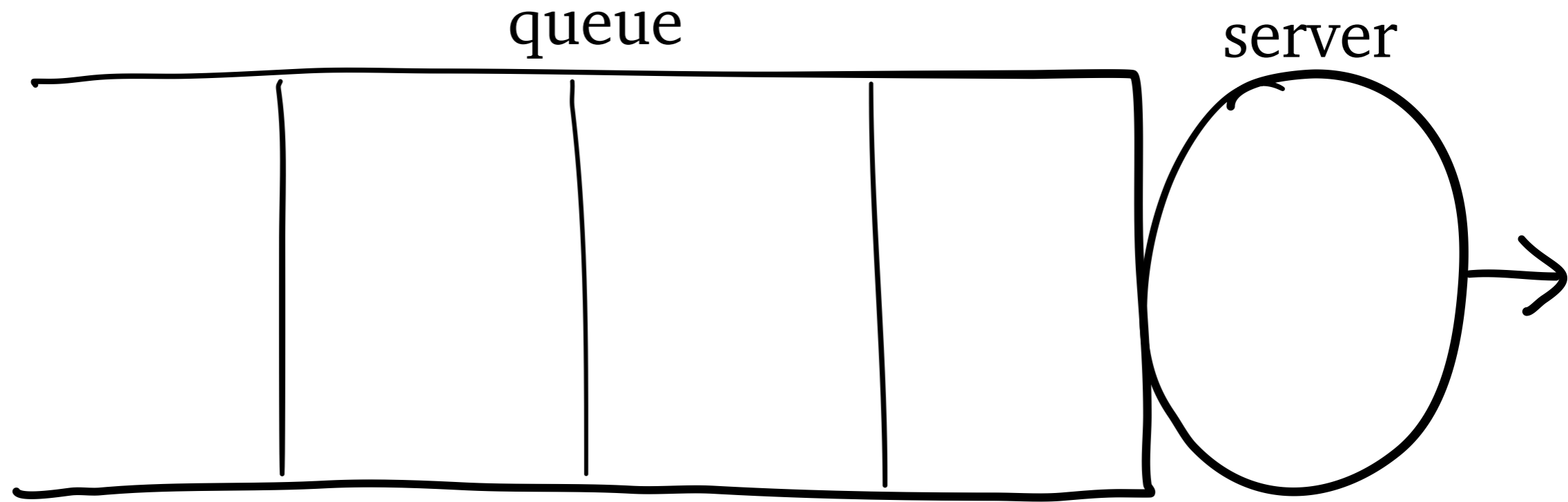
Queueing theory: studies the *mathematical essence* of queueing systems

↙ ↘
jobs service

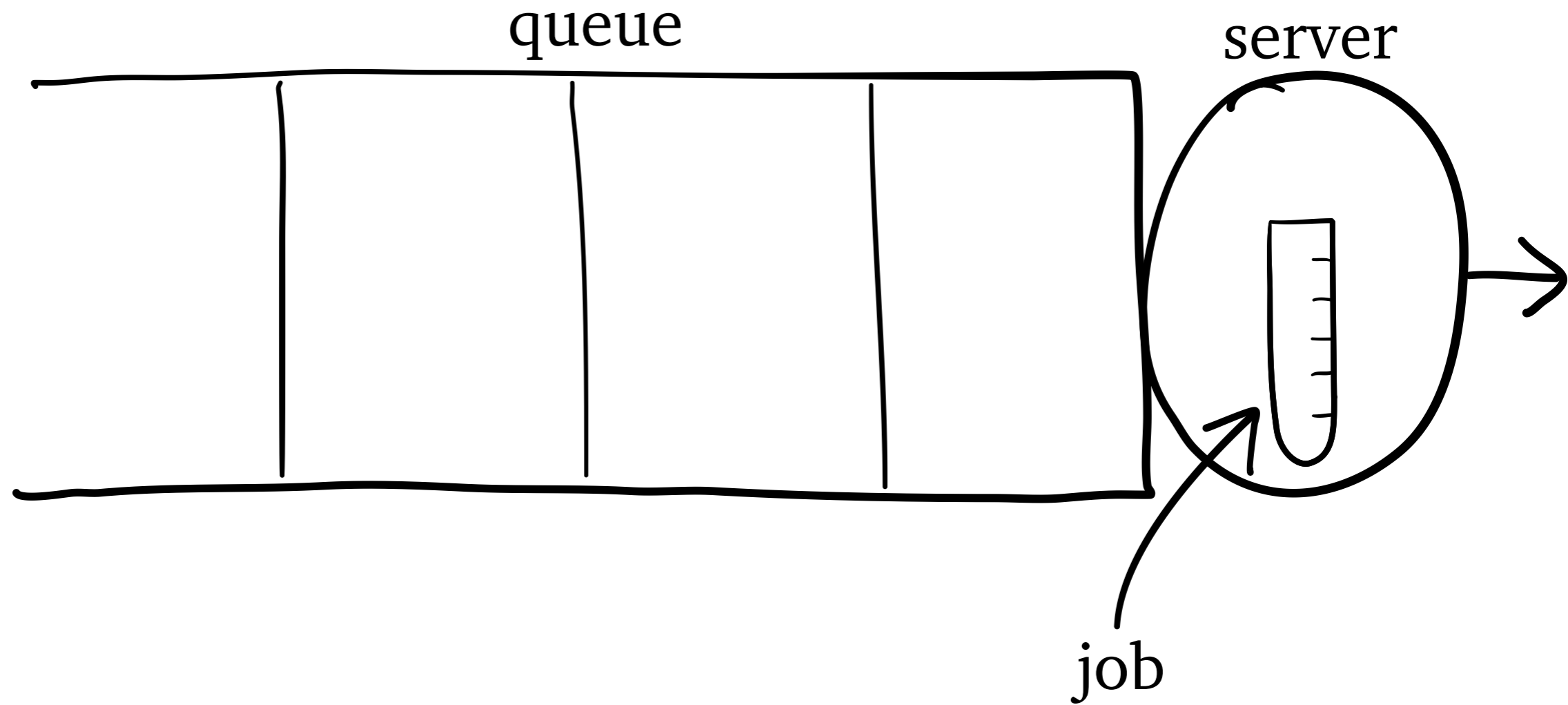
M/G/1 Queueing Model



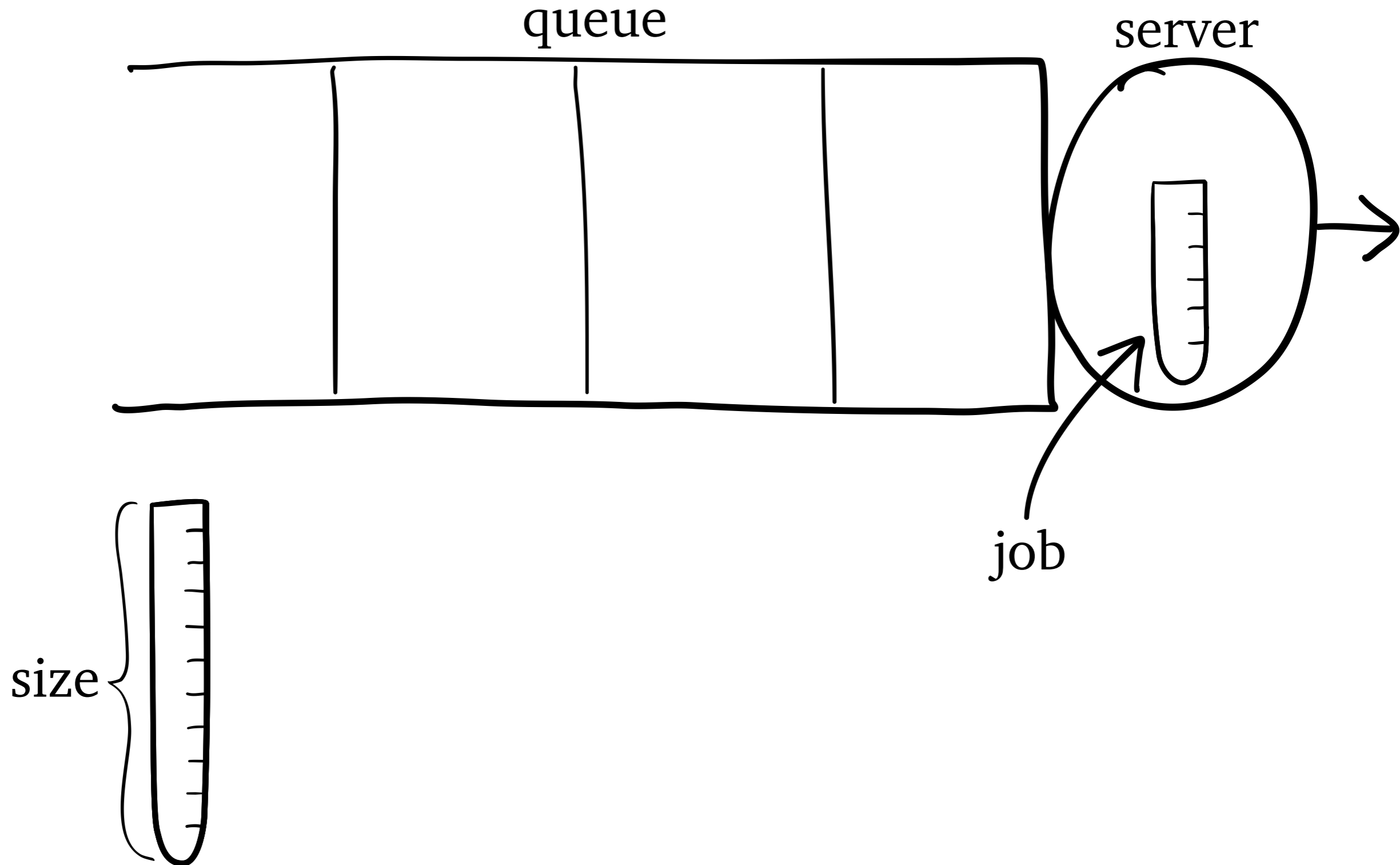
M/G/1 Queueing Model



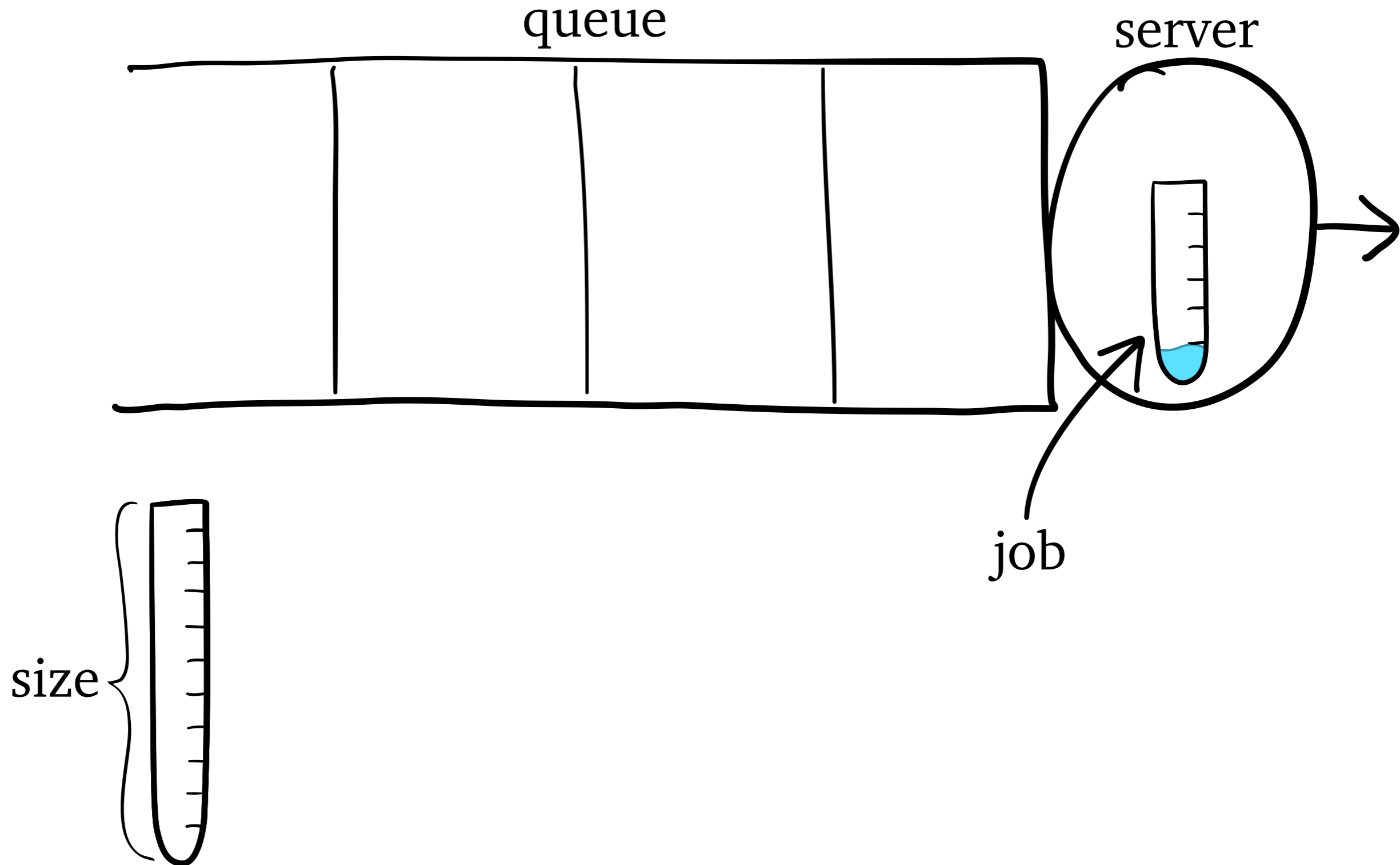
M/G/1 Queueing Model



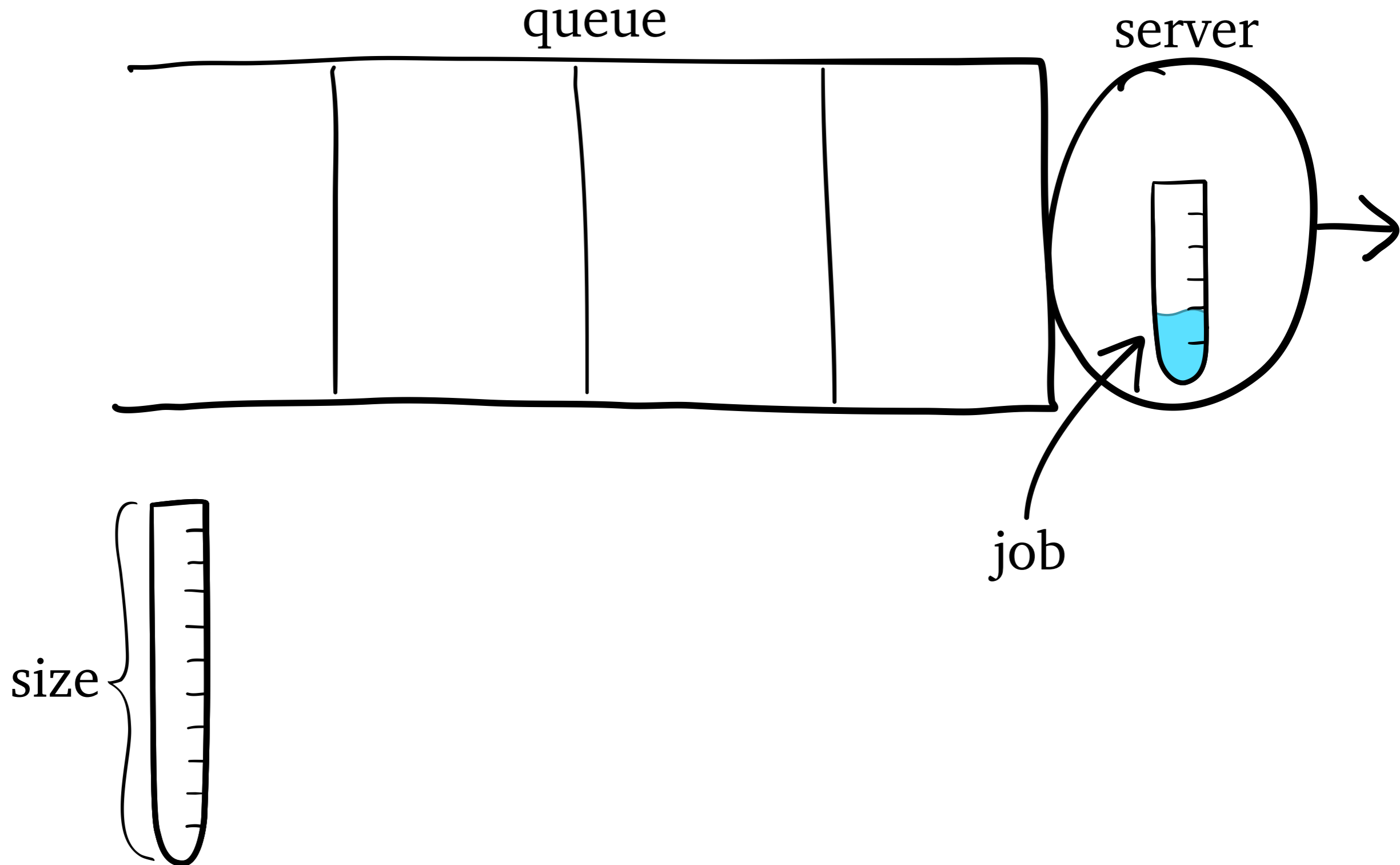
M/G/1 Queueing Model



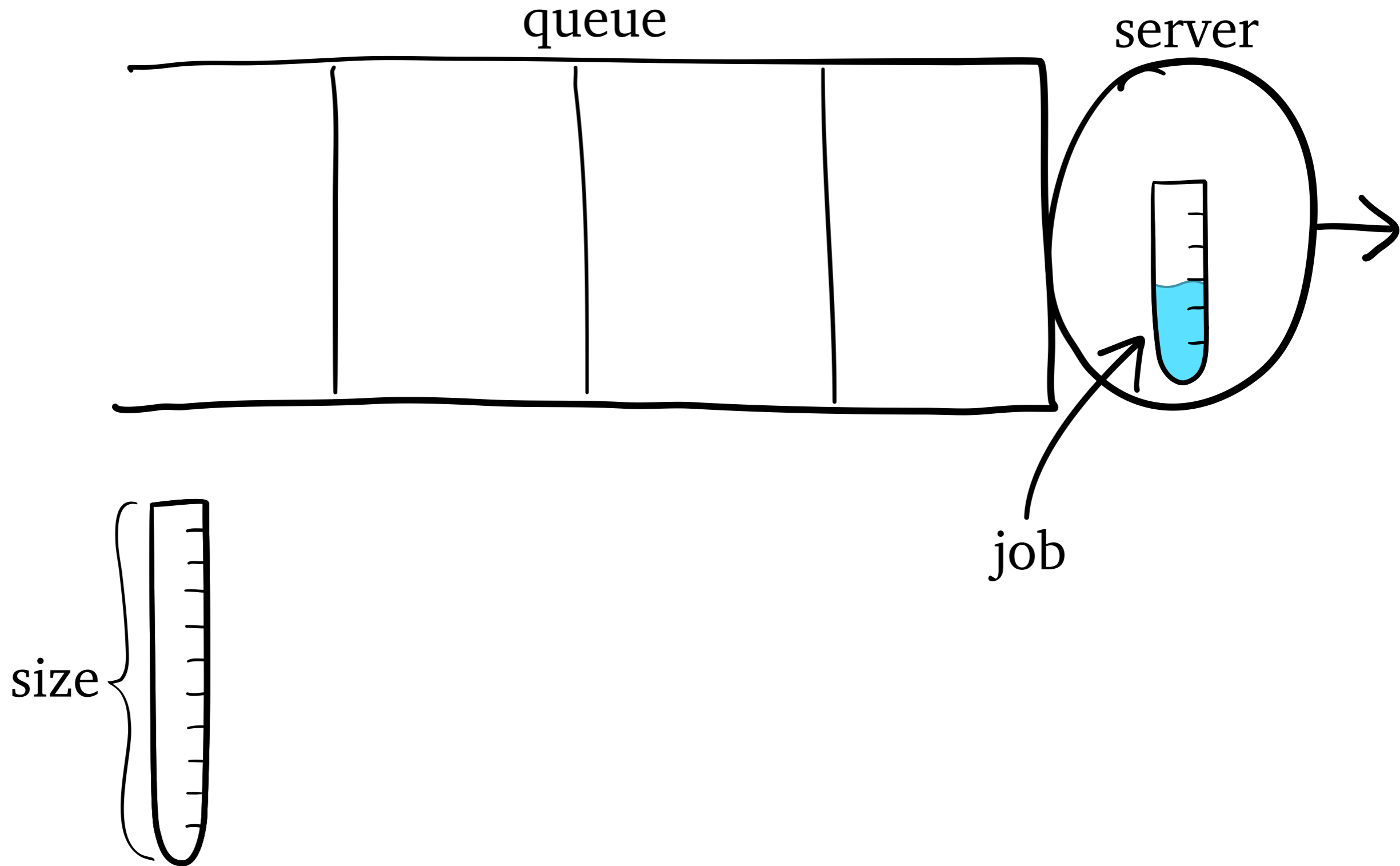
M/G/1 Queueing Model



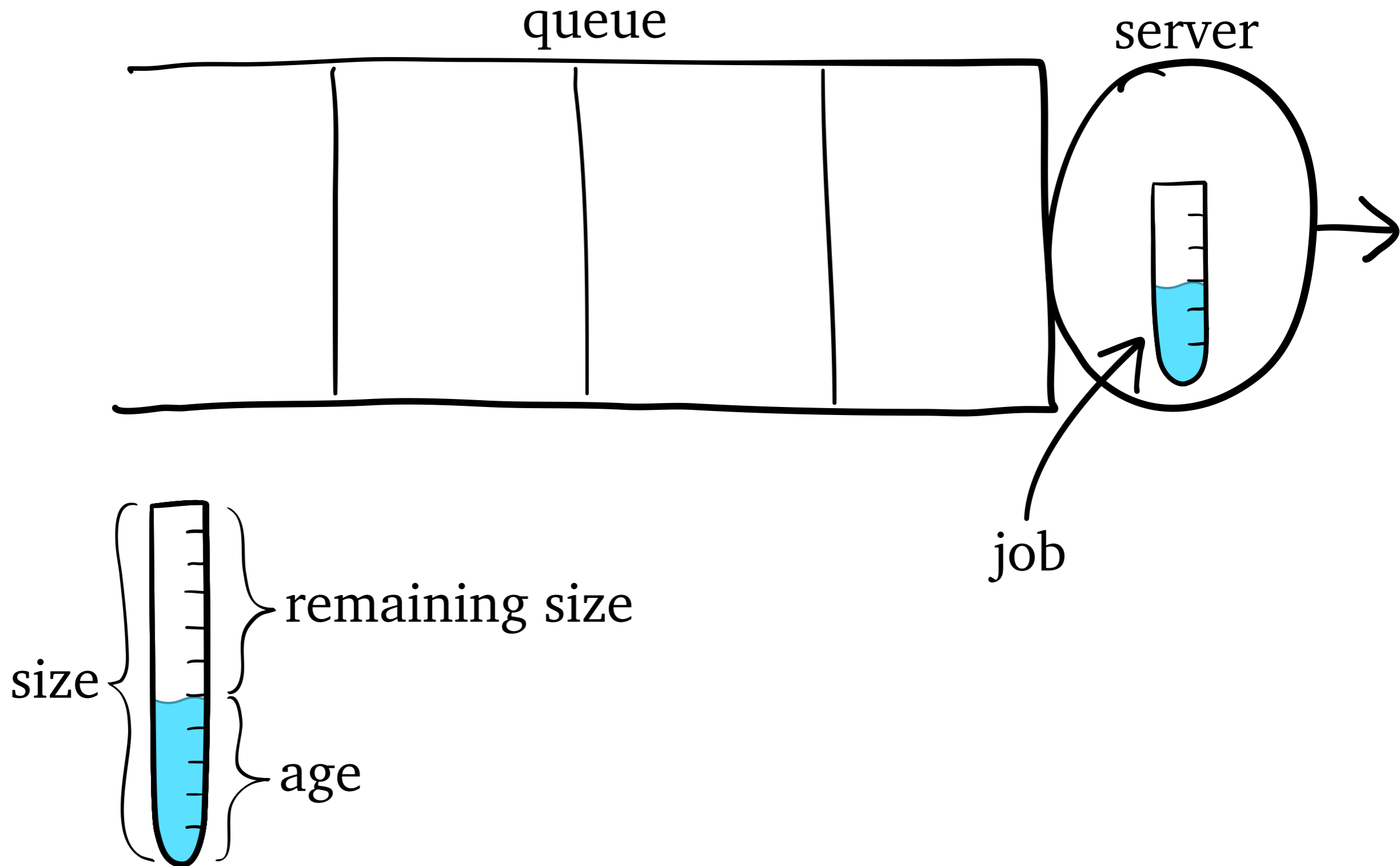
M/G/1 Queueing Model



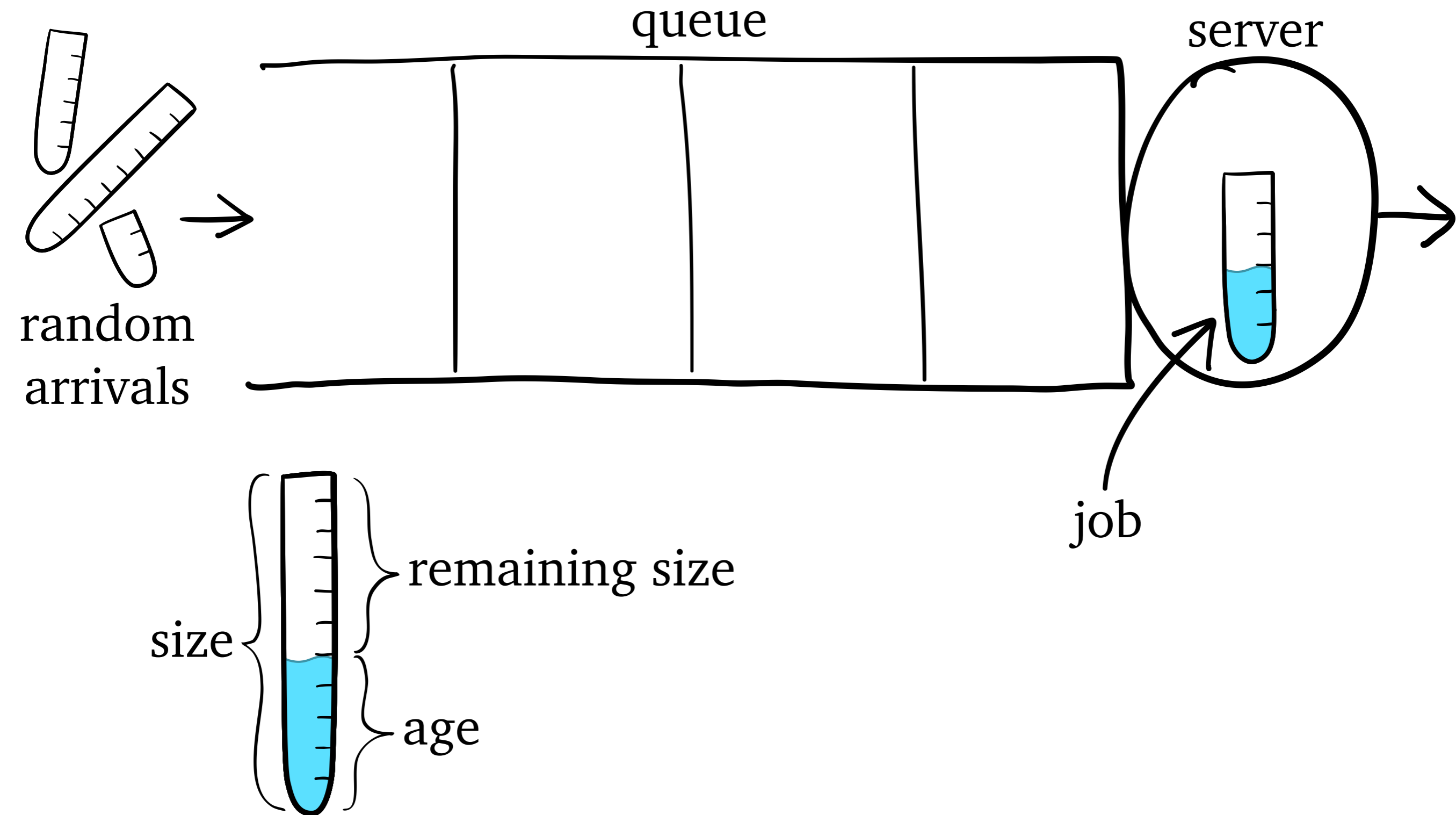
M/G/1 Queueing Model



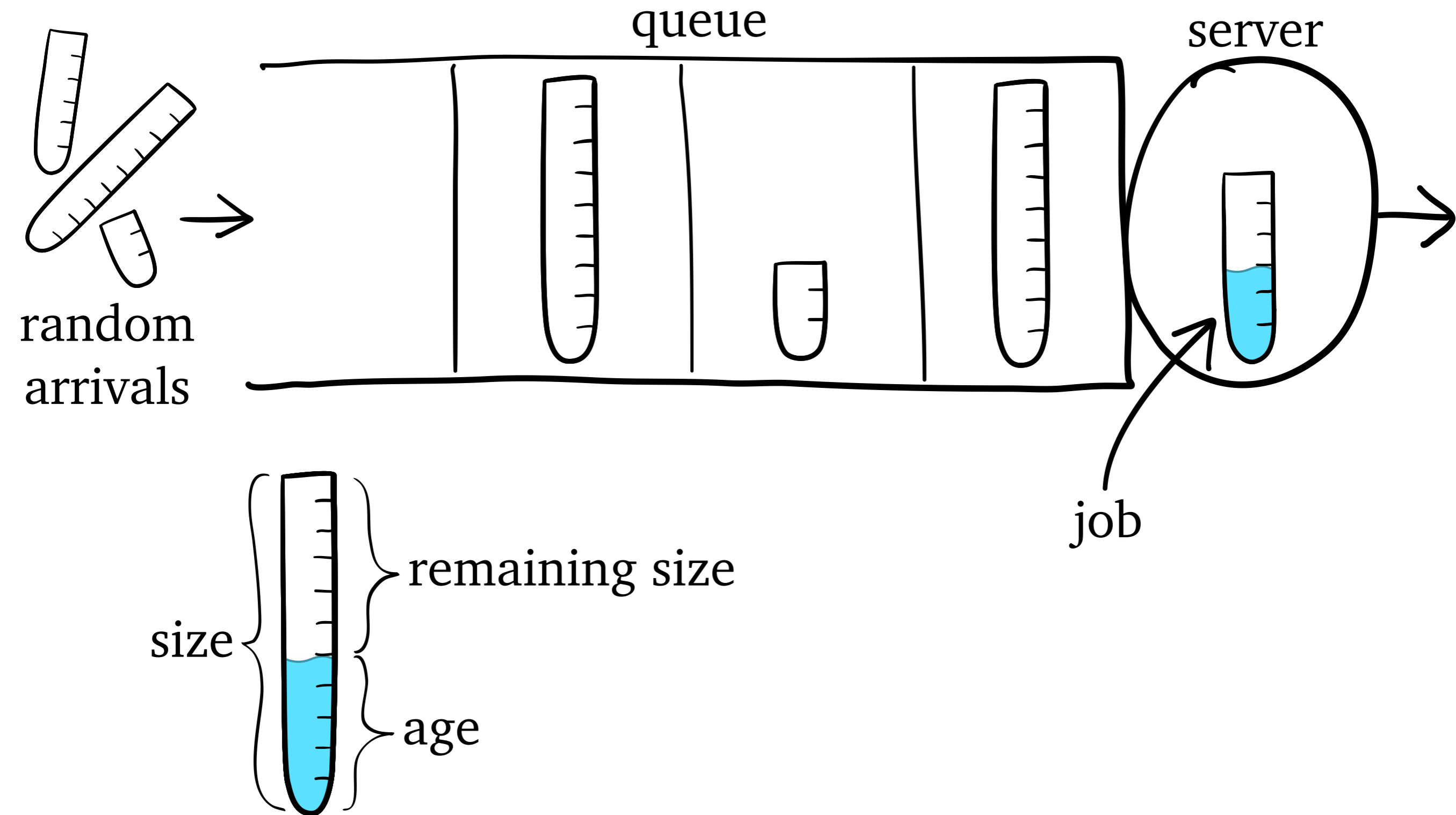
M/G/1 Queueing Model



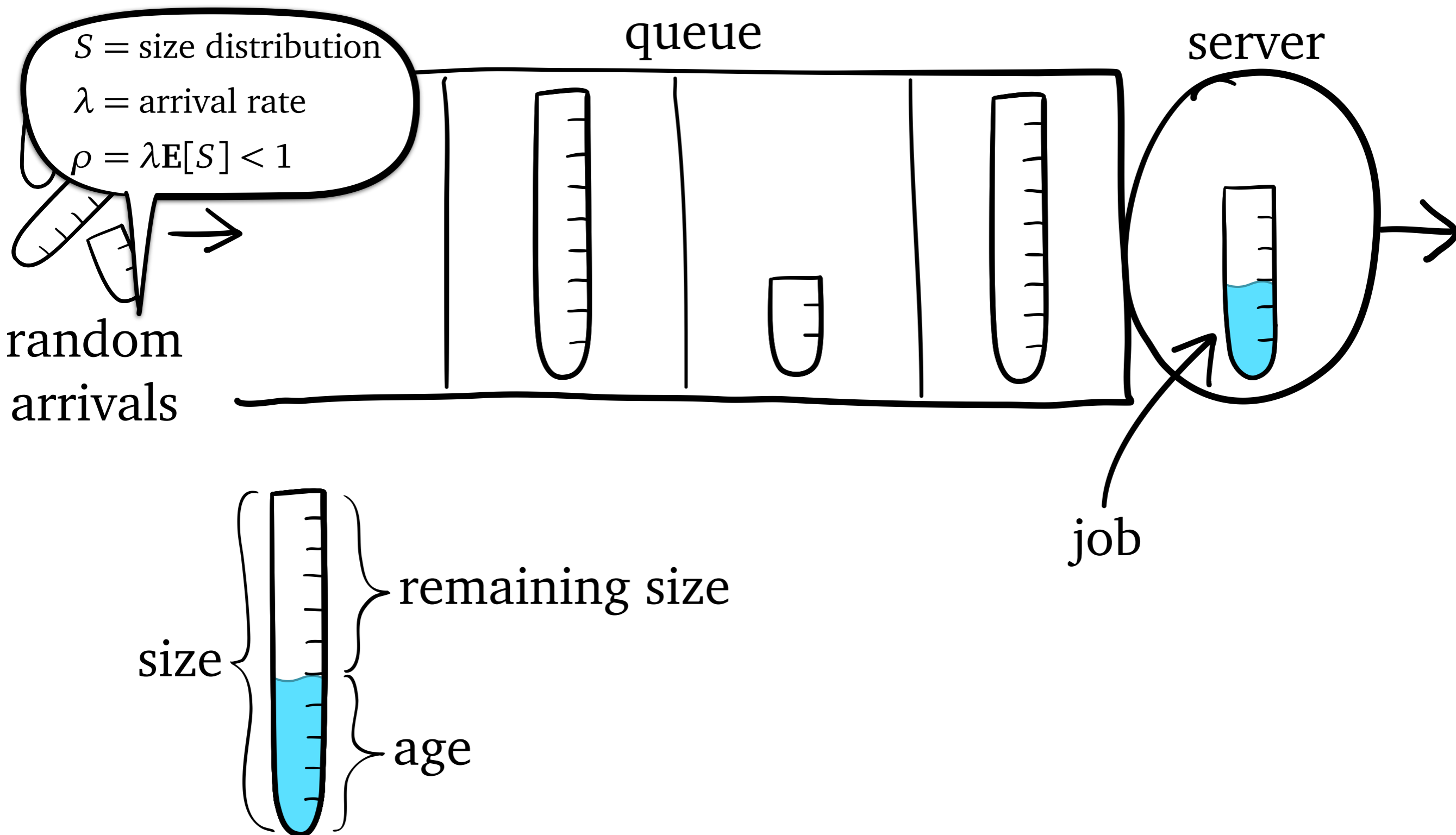
M/G/1 Queueing Model



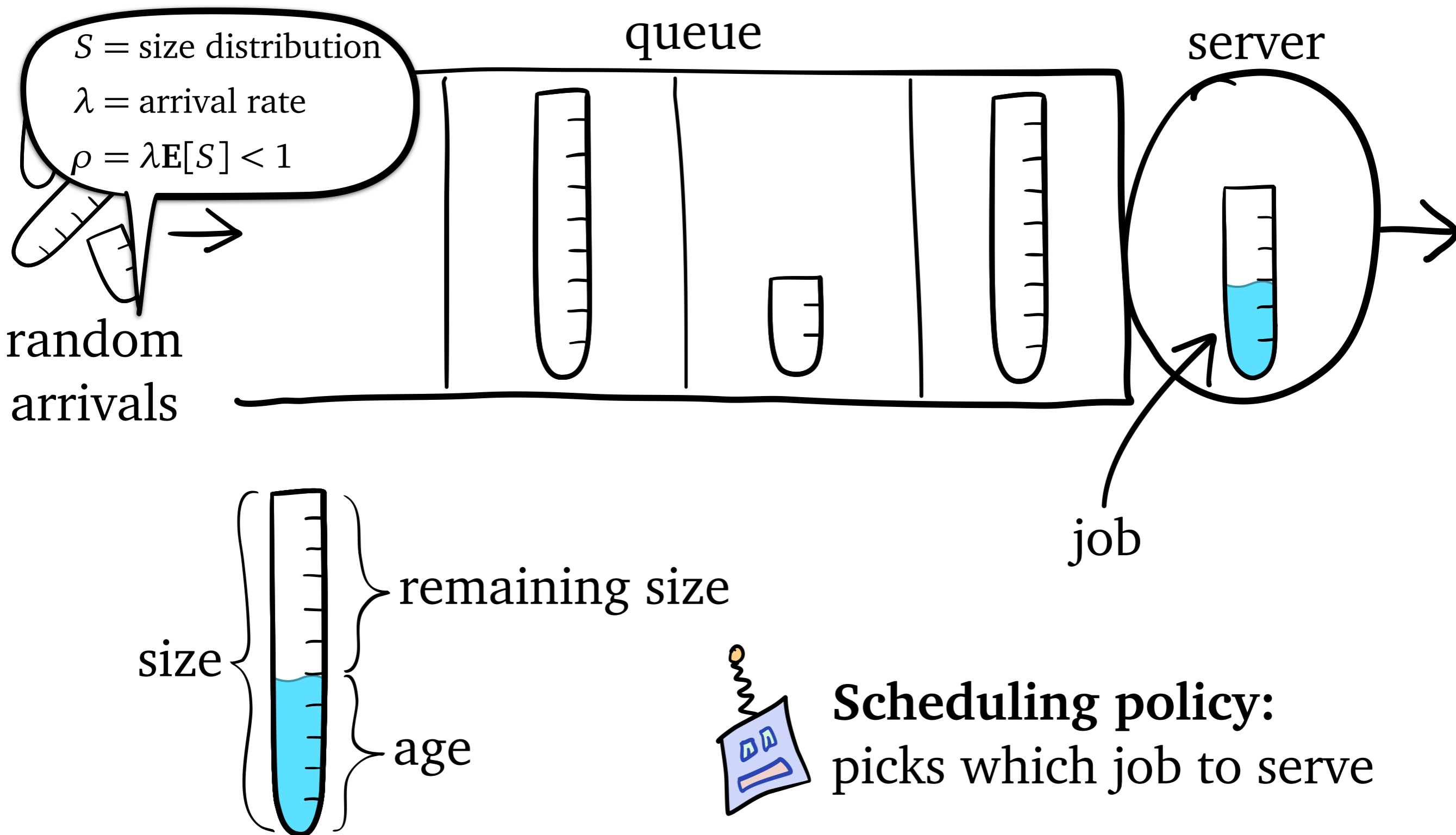
M/G/1 Queueing Model



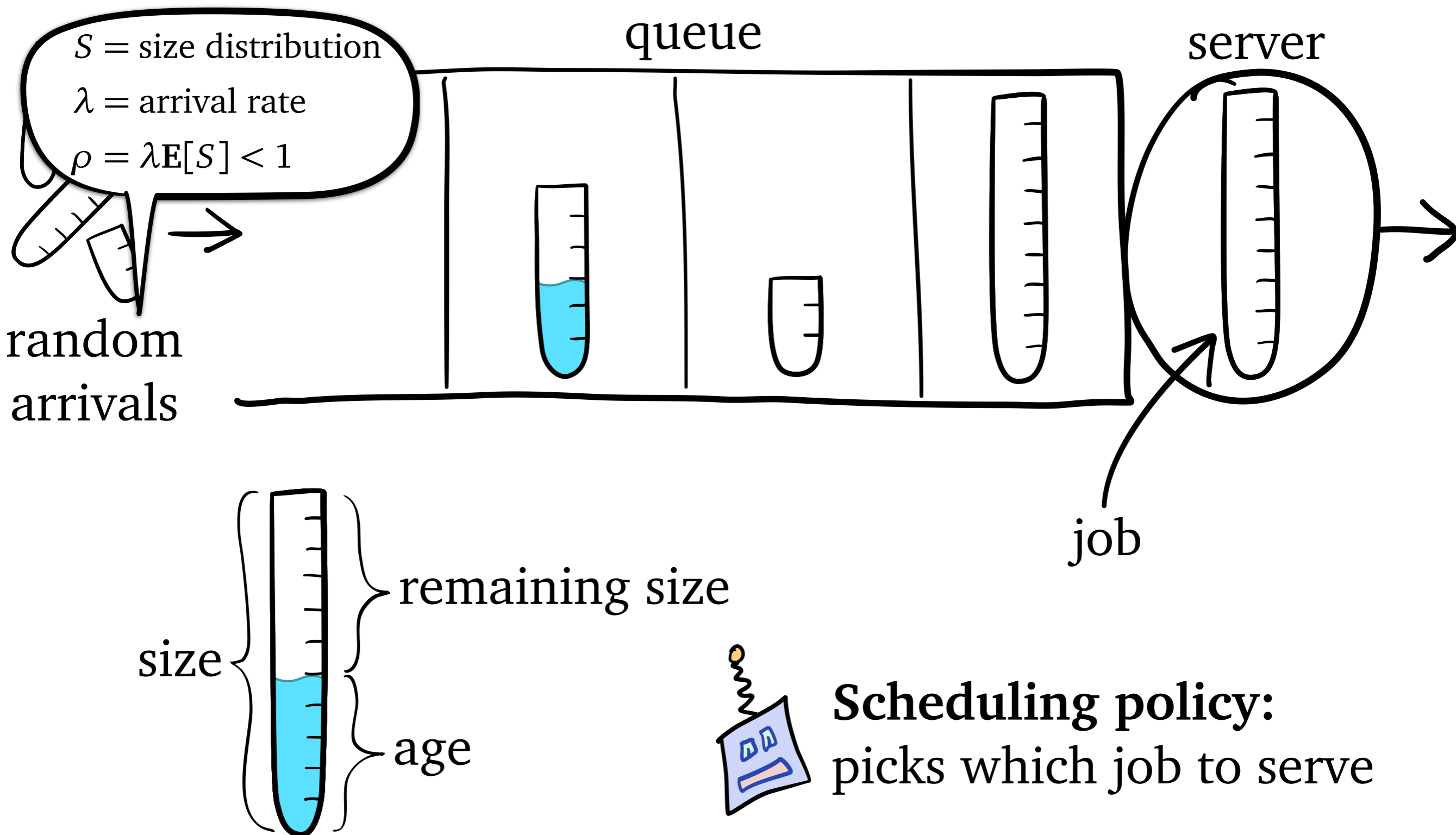
M/G/1 Queueing Model



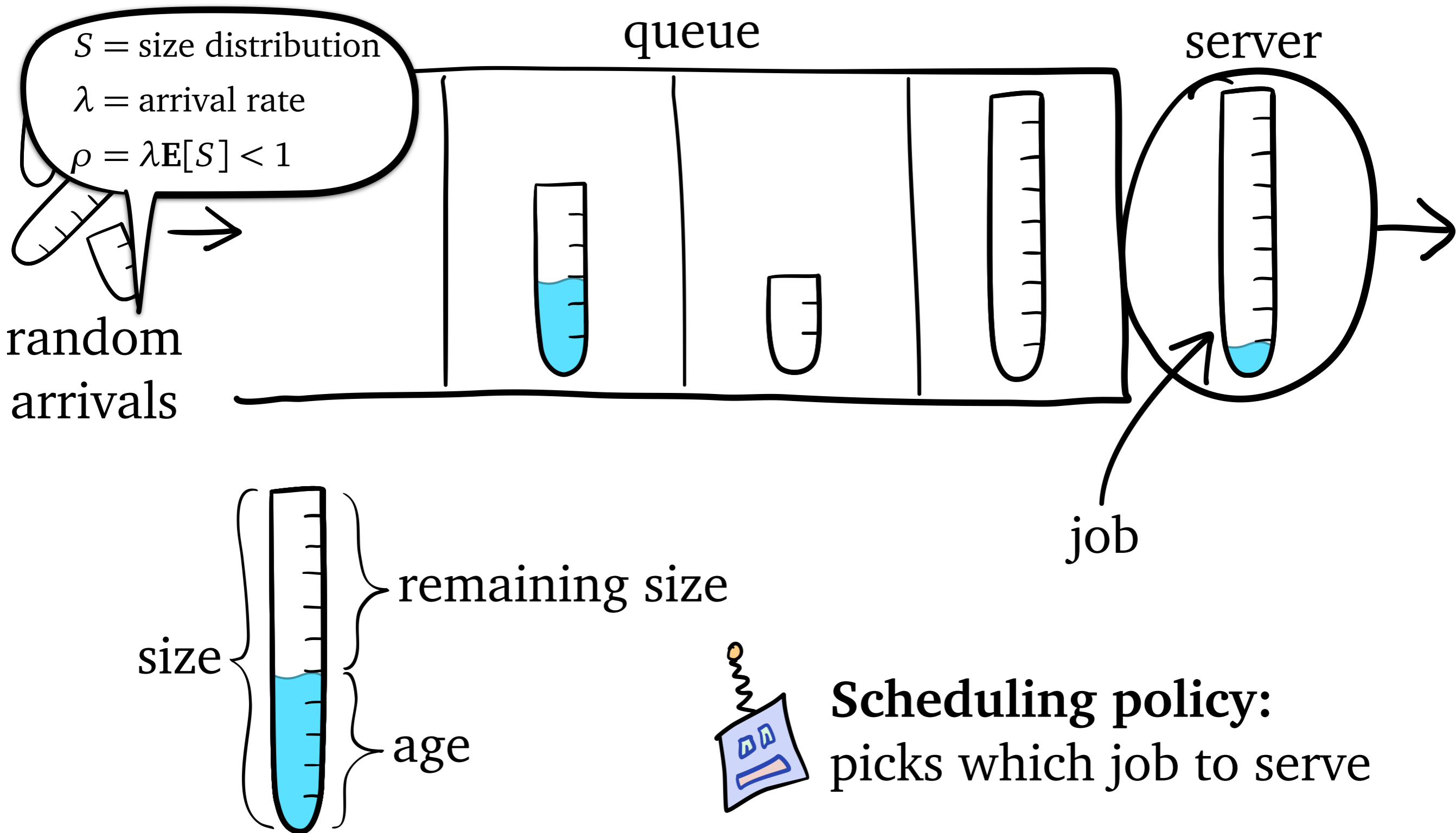
M/G/1 Queueing Model



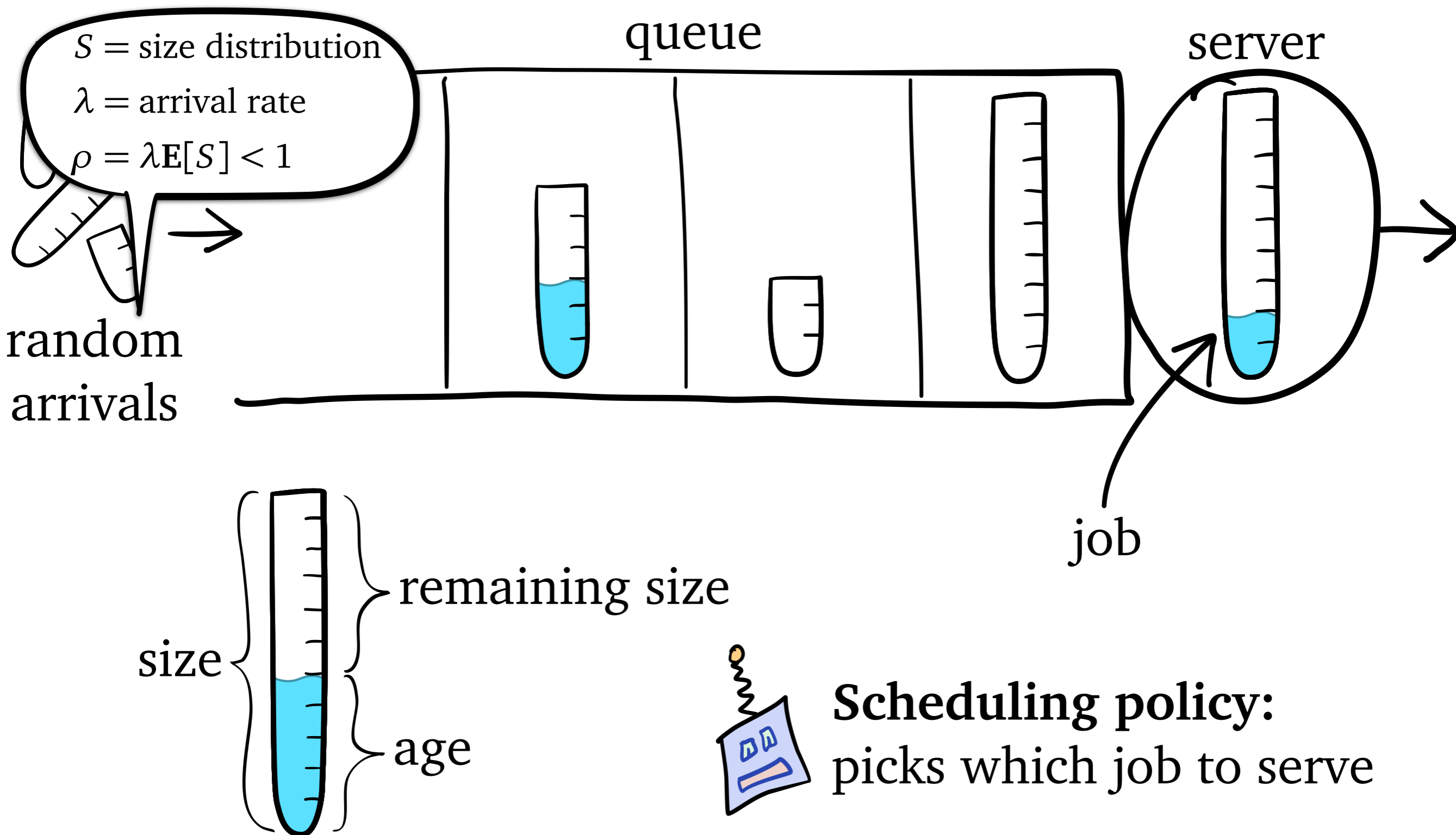
M/G/1 Queueing Model



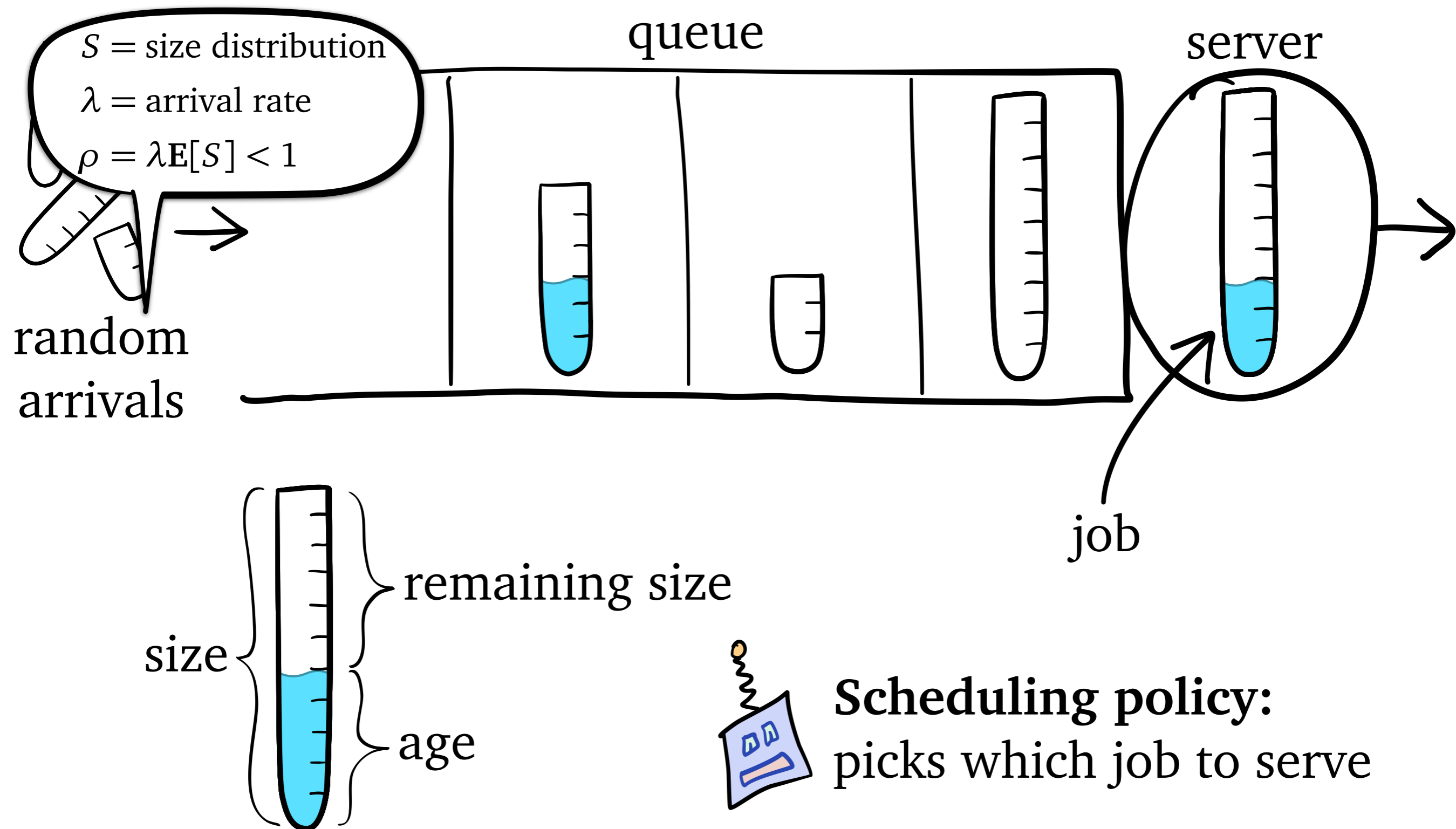
M/G/1 Queueing Model



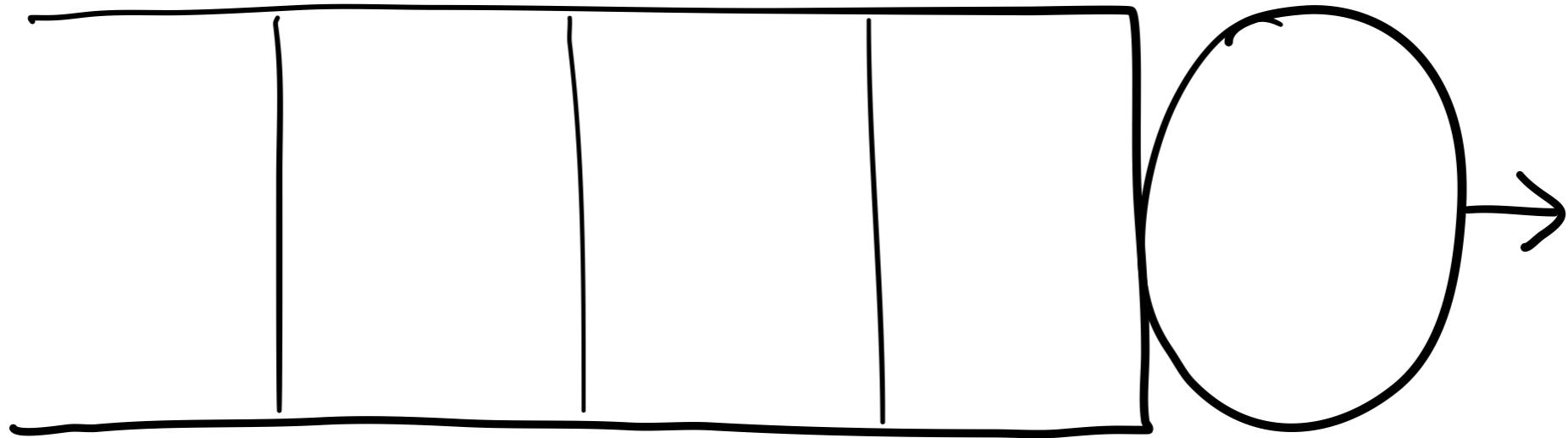
M/G/1 Queueing Model



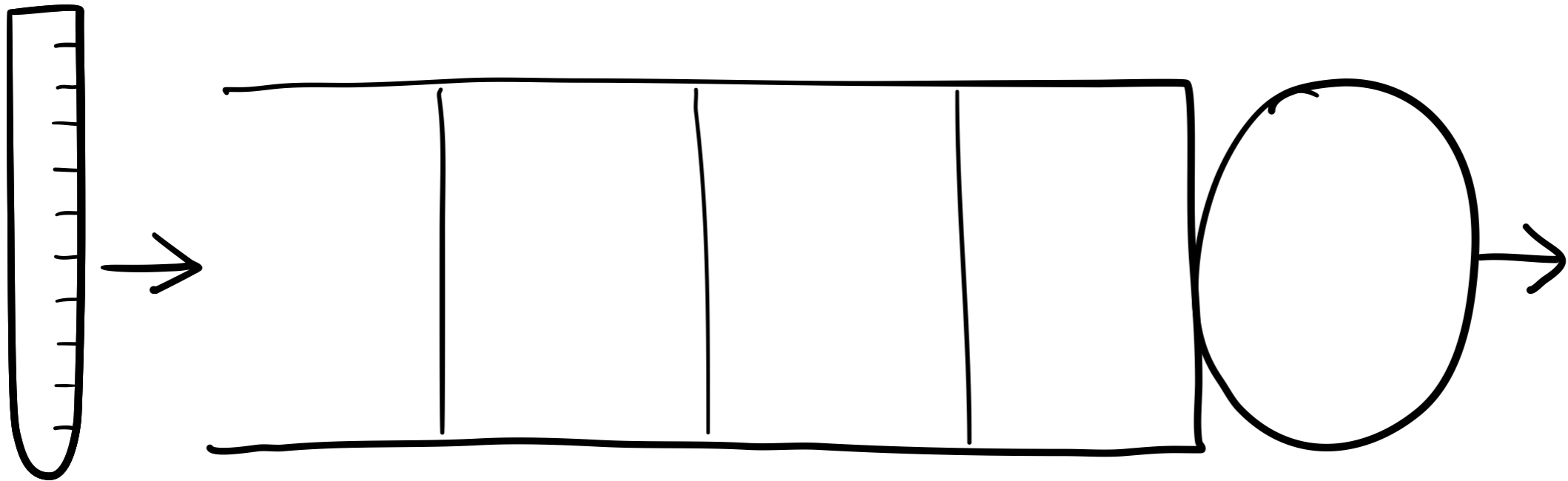
M/G/1 Queueing Model



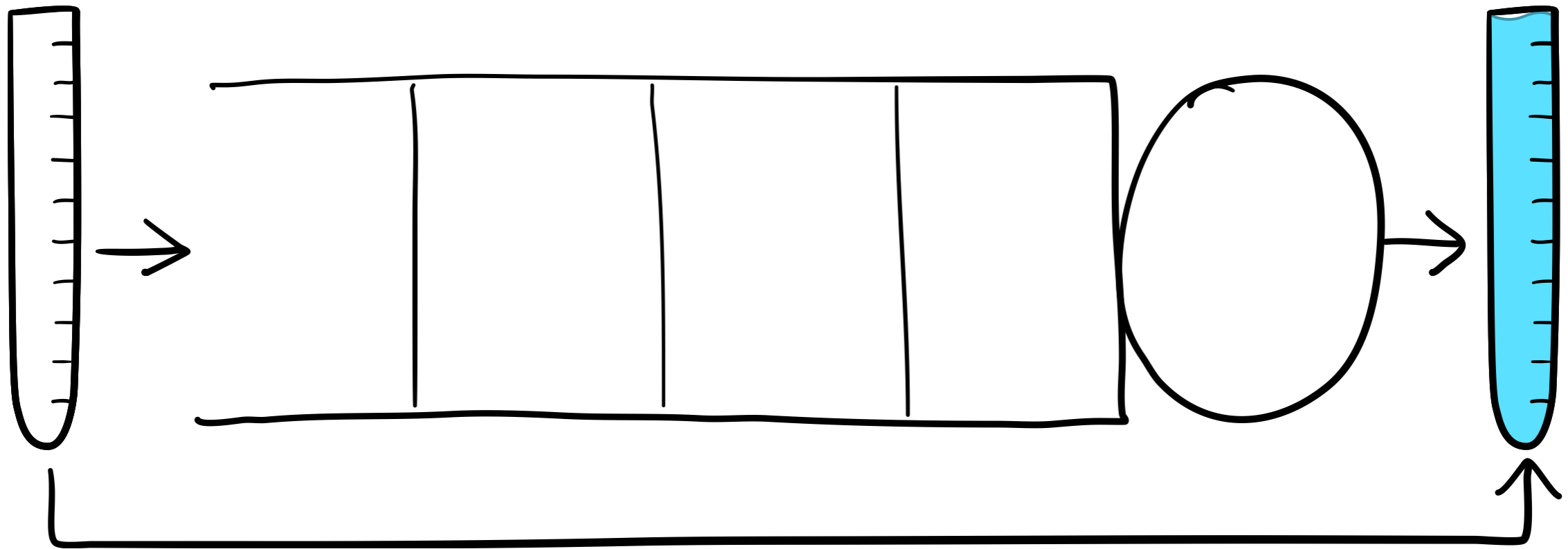
Response Time




Response Time

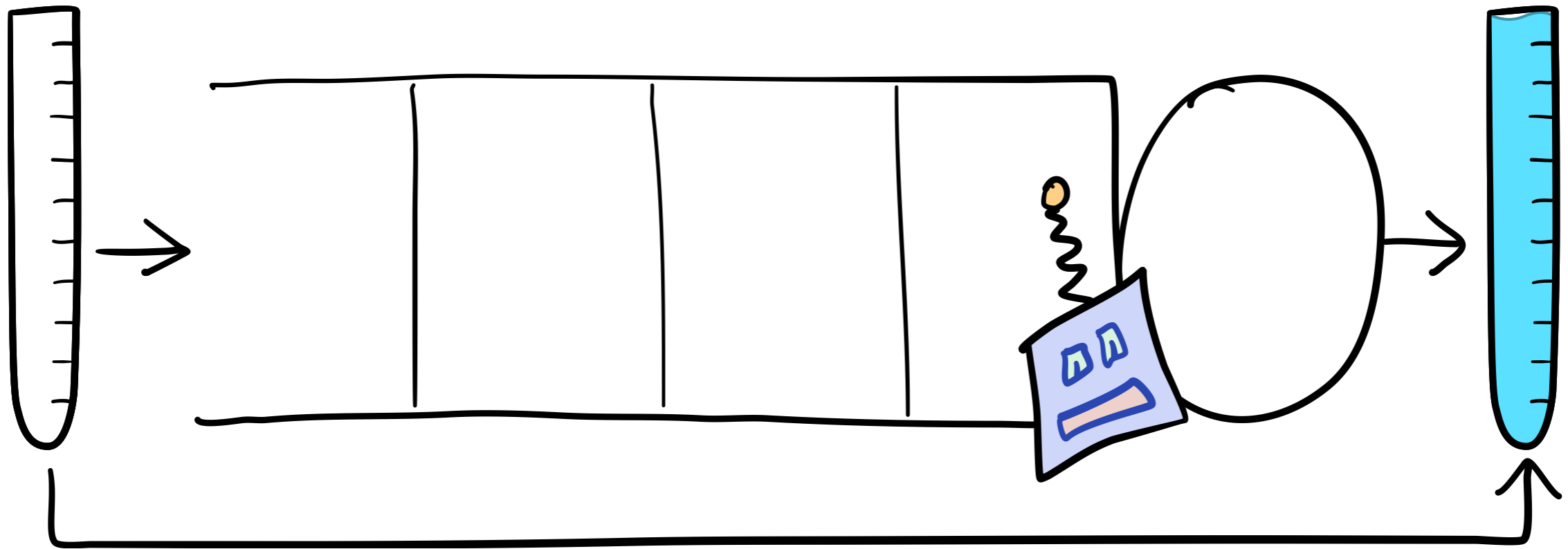



Response Time



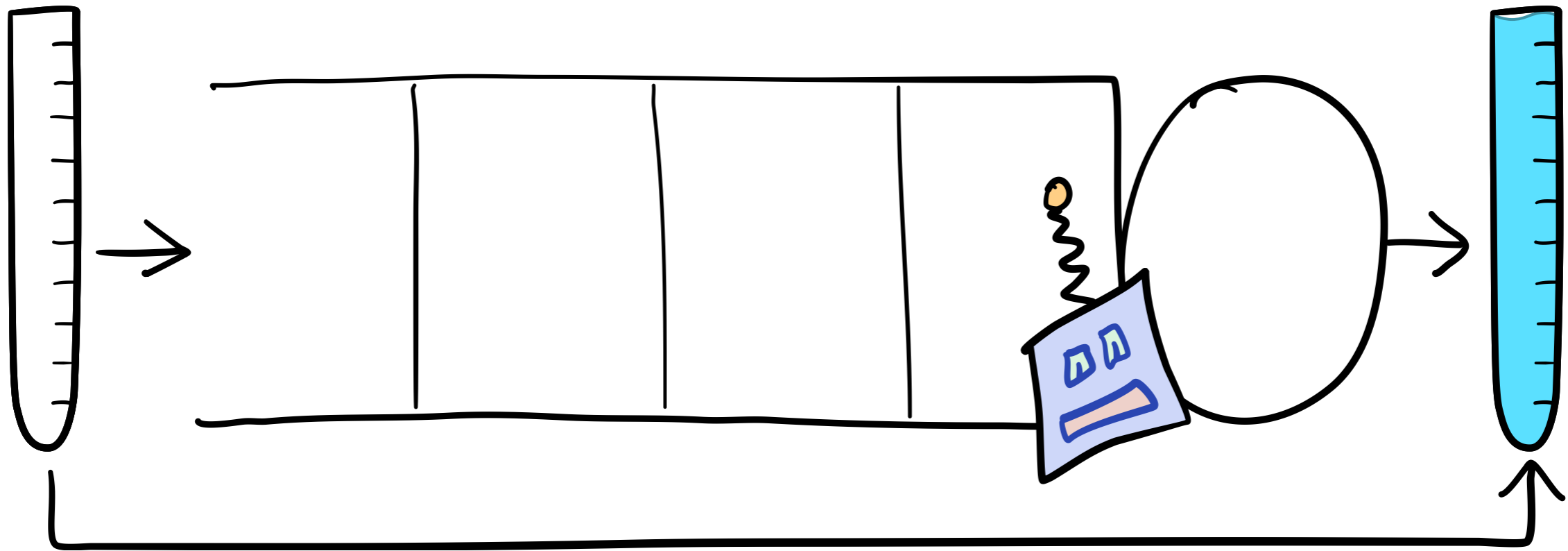
 = T = *response time*


Response Time



 = T = *response time*

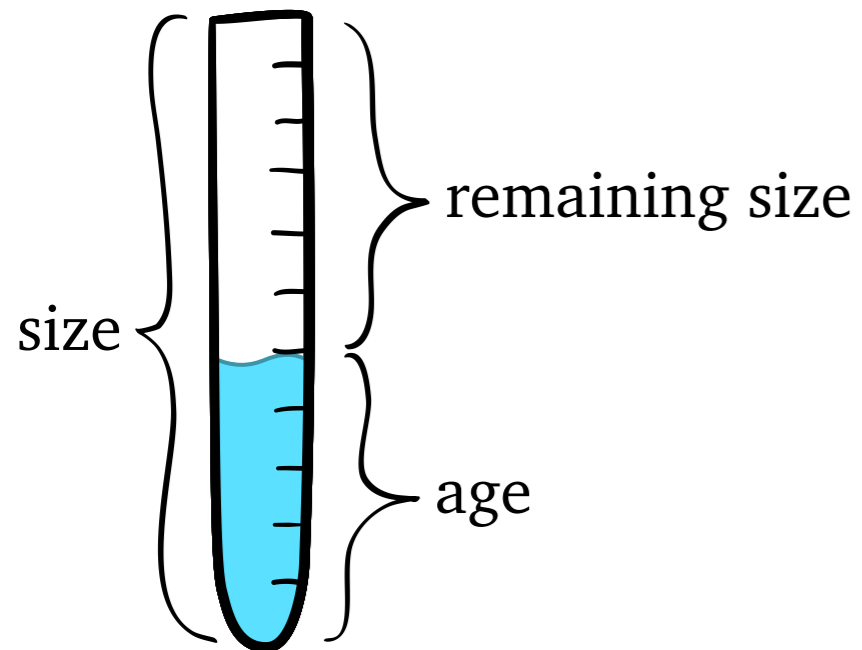
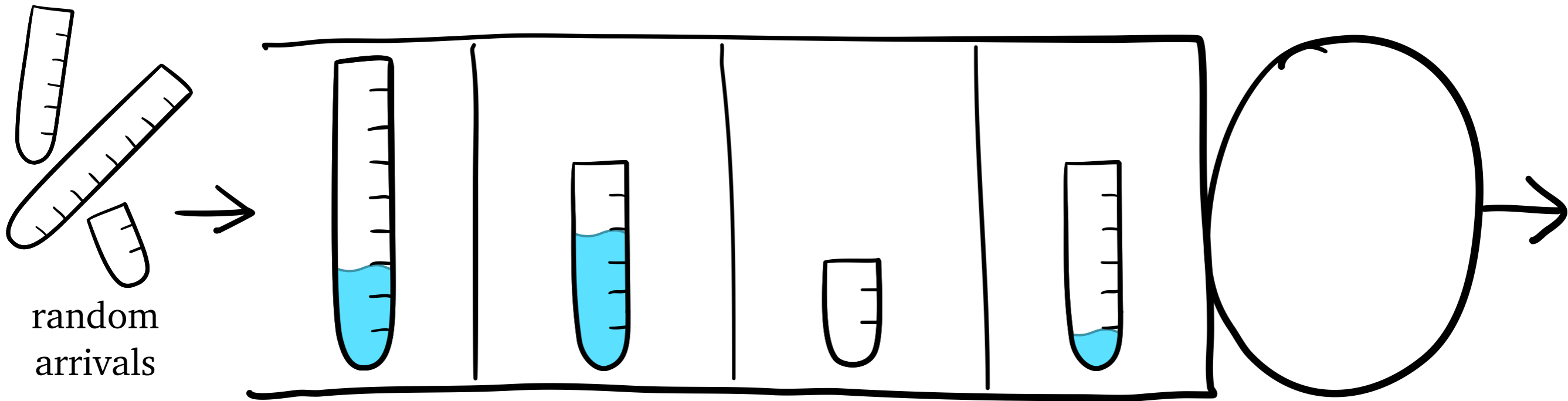
Response Time



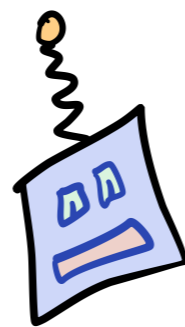
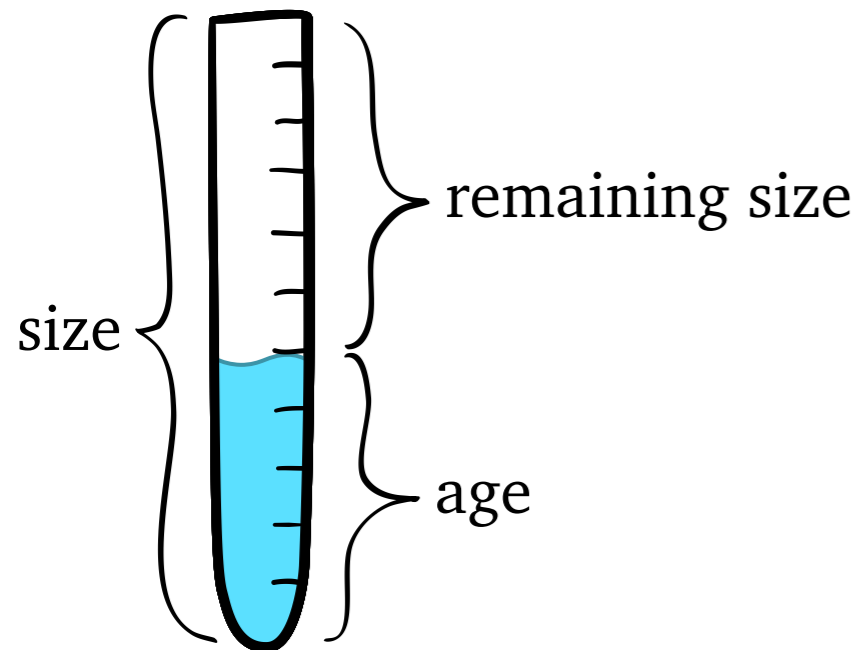
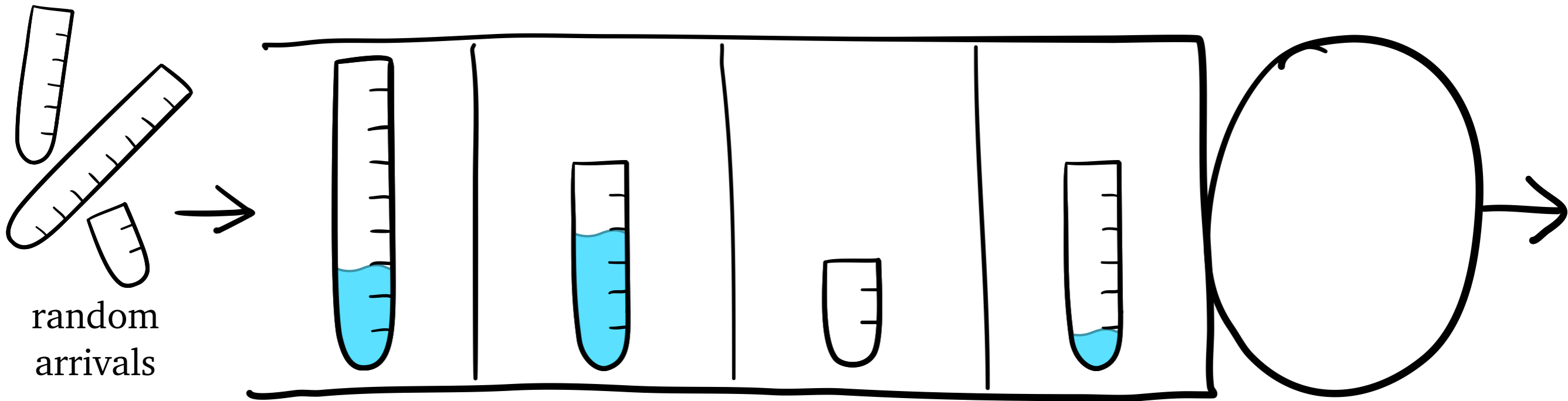
 = T = *response time*

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

How to Schedule?

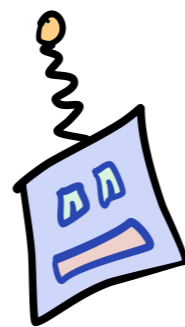
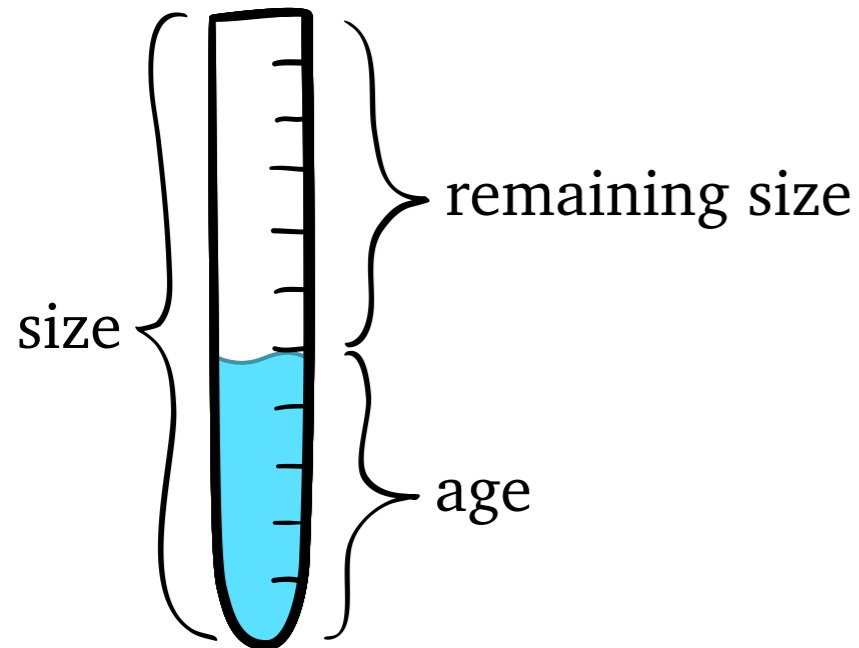
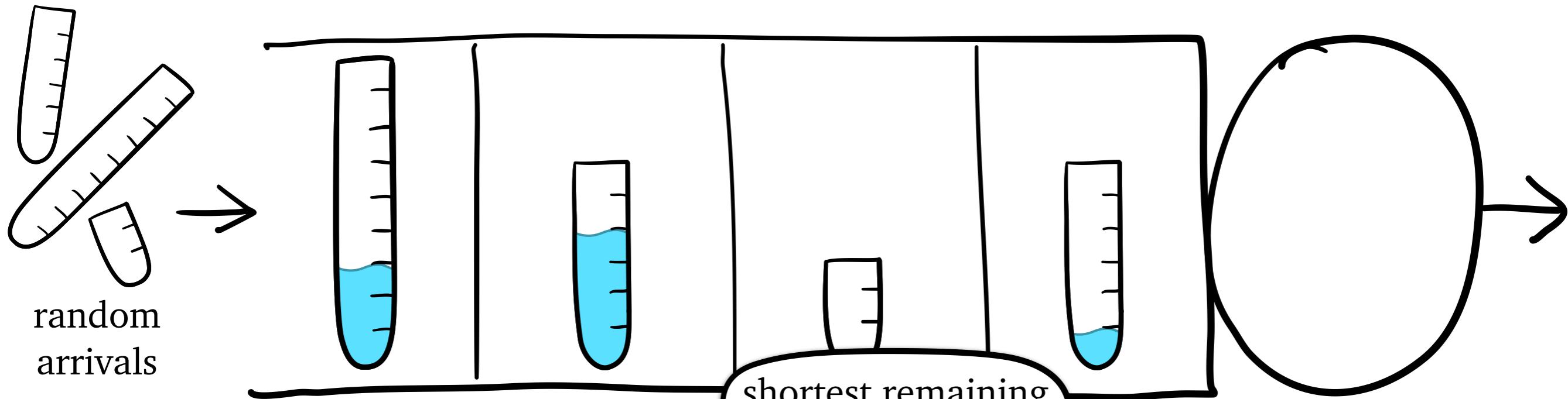


How to Schedule?



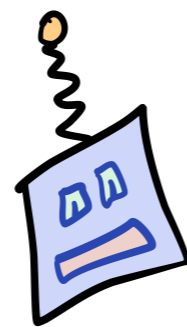
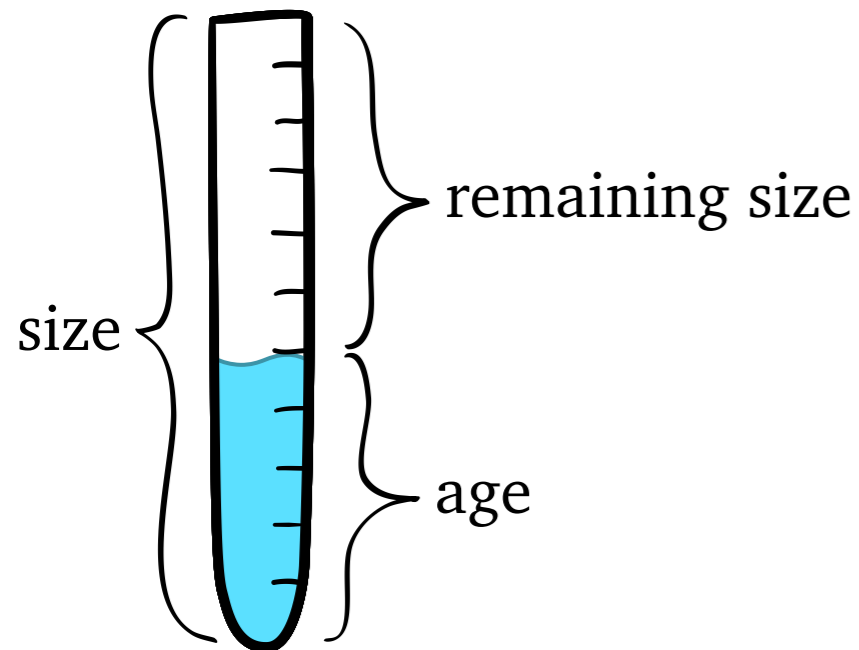
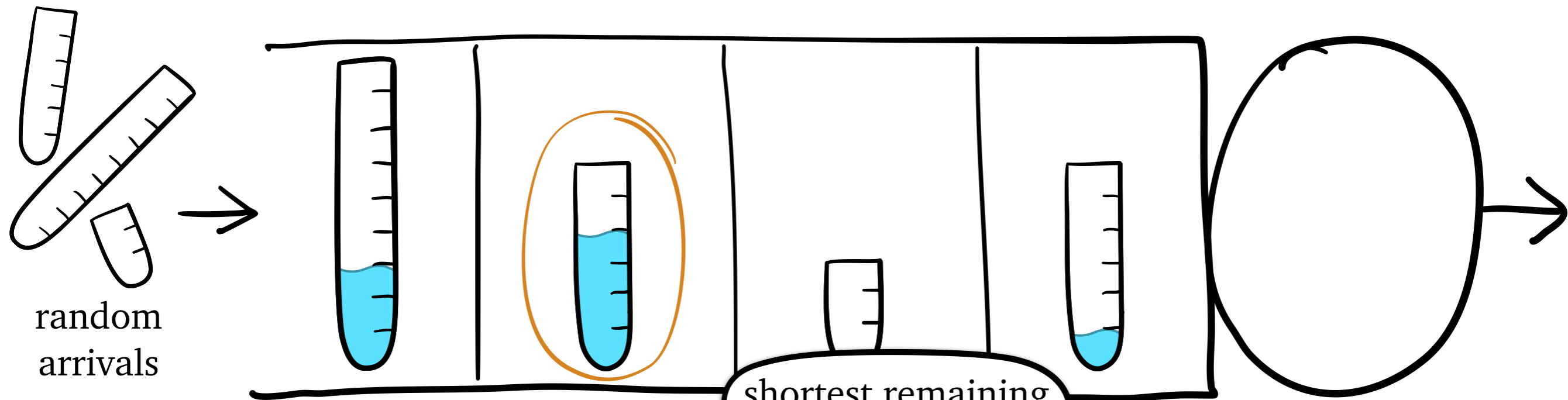
SRPT: always serve job of *least remaining size*

How to Schedule?



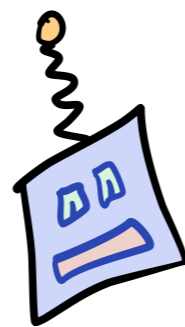
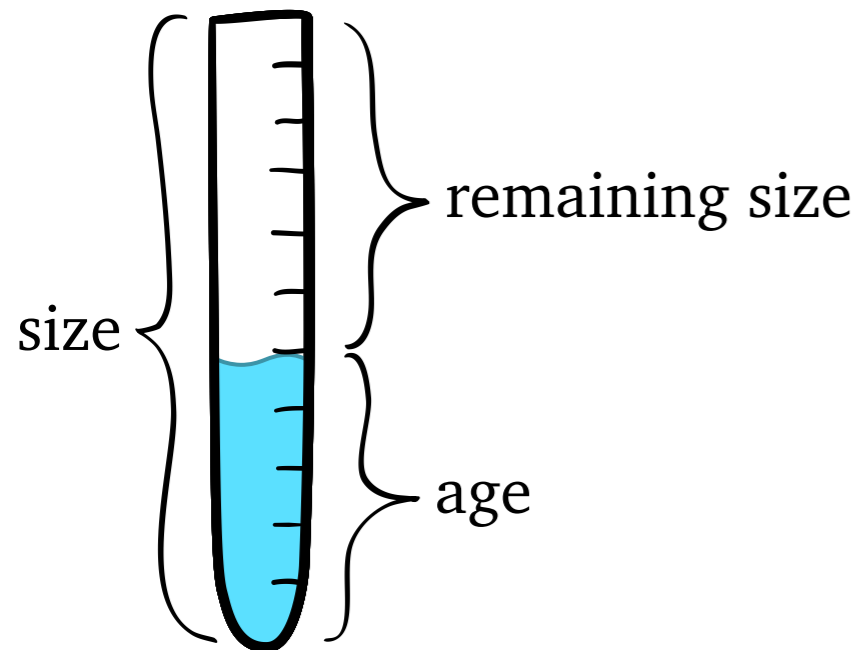
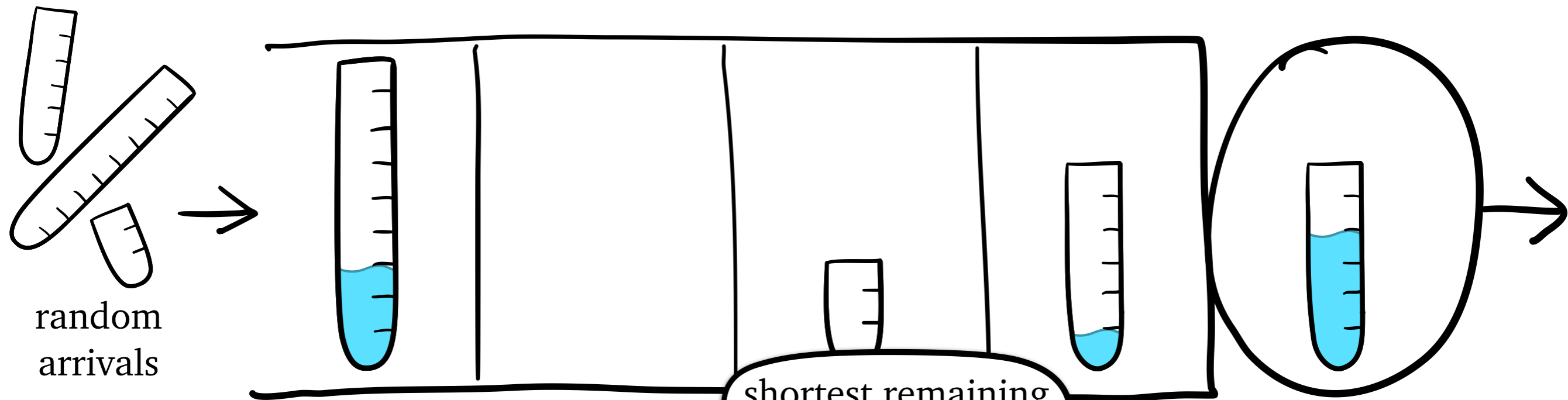
SRPT: always serve job of *least remaining size*

How to Schedule?



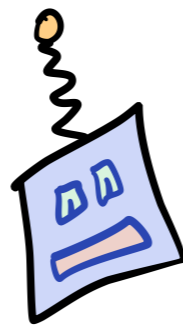
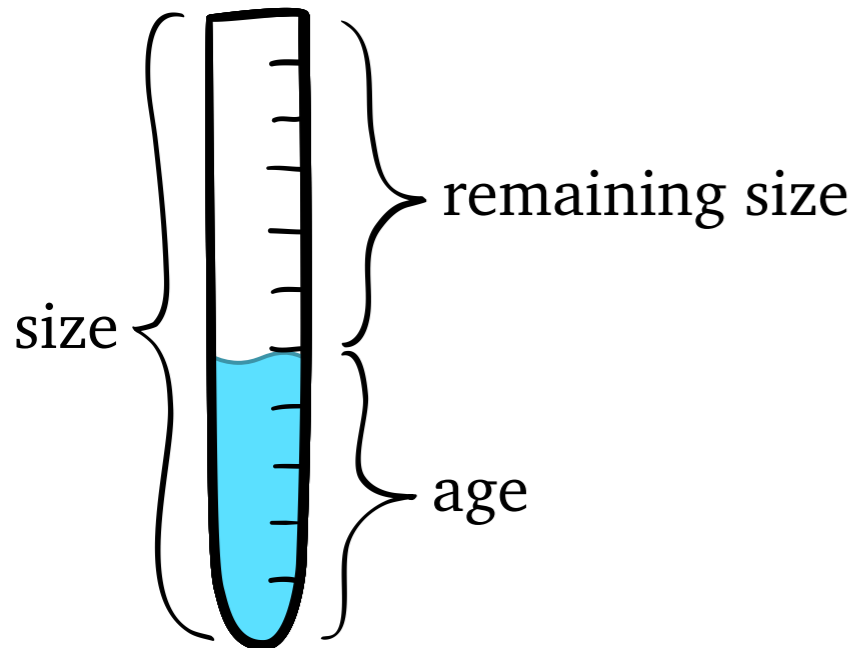
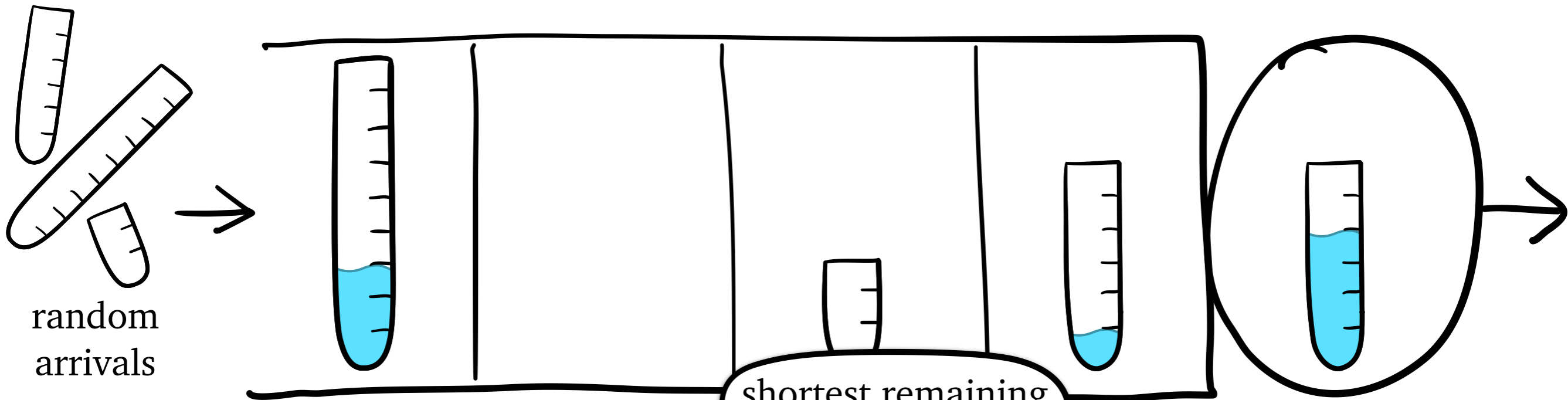
SRPT: always serve job of *least remaining size*

How to Schedule?



SRPT: always serve job of *least remaining size*

How to Schedule?

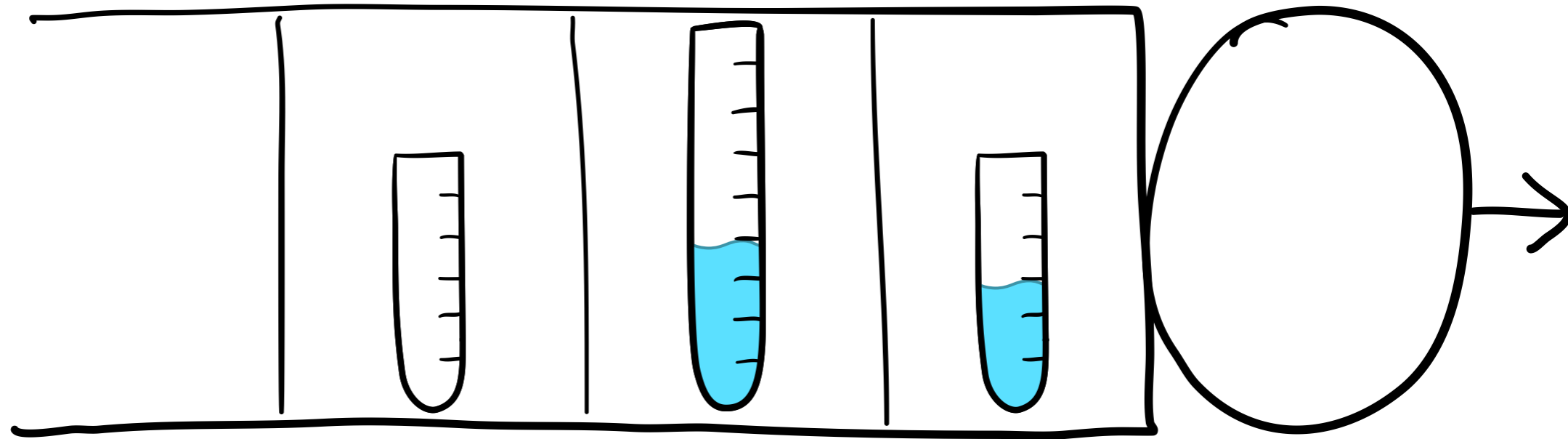


SRPT: always serve job of *least remaining size*

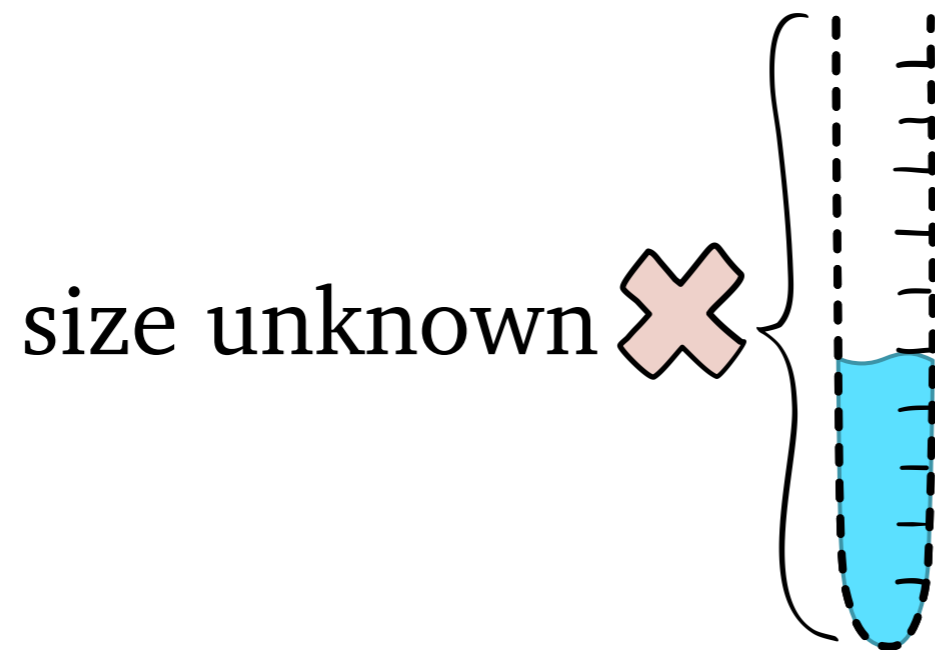
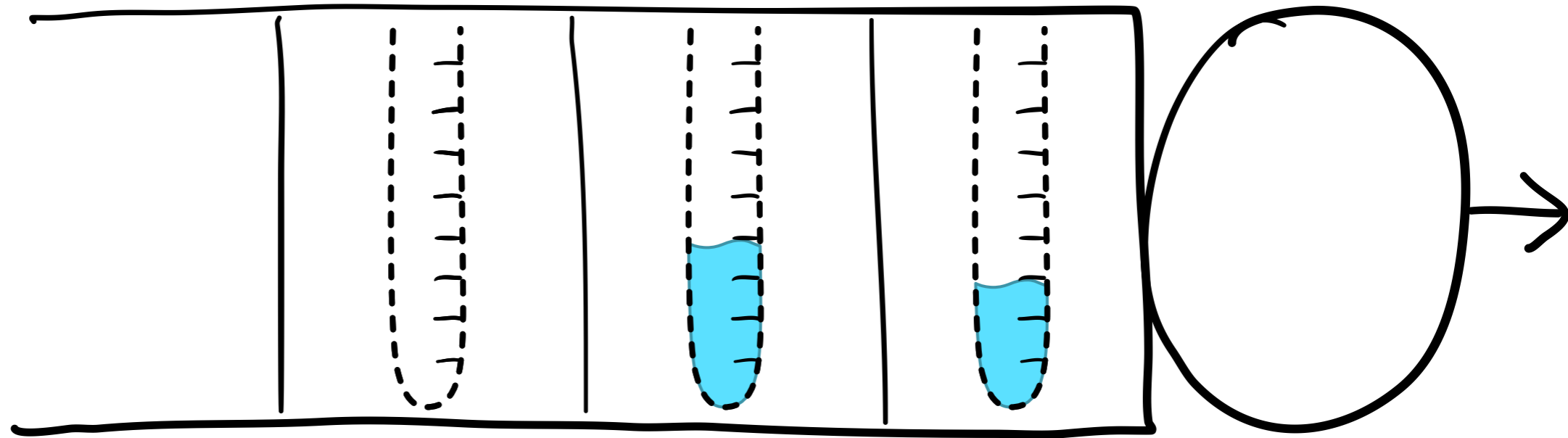


SRPT minimizes $E[T]$

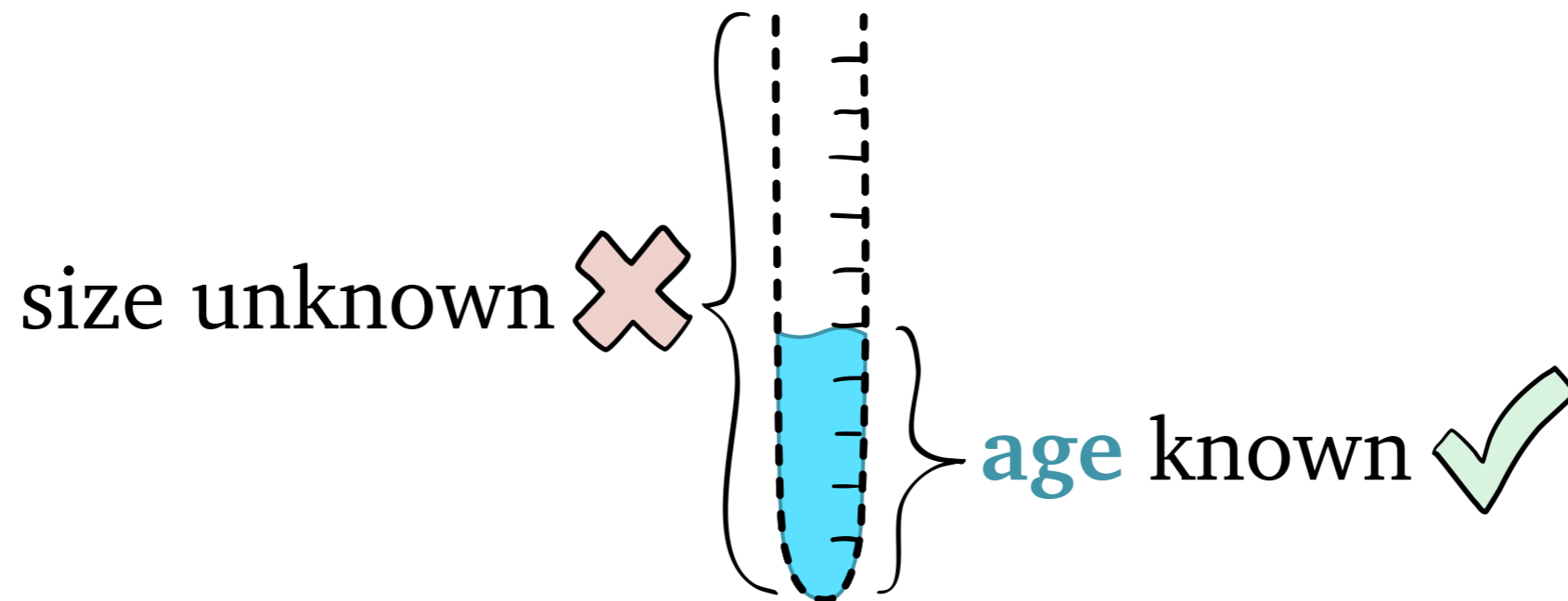
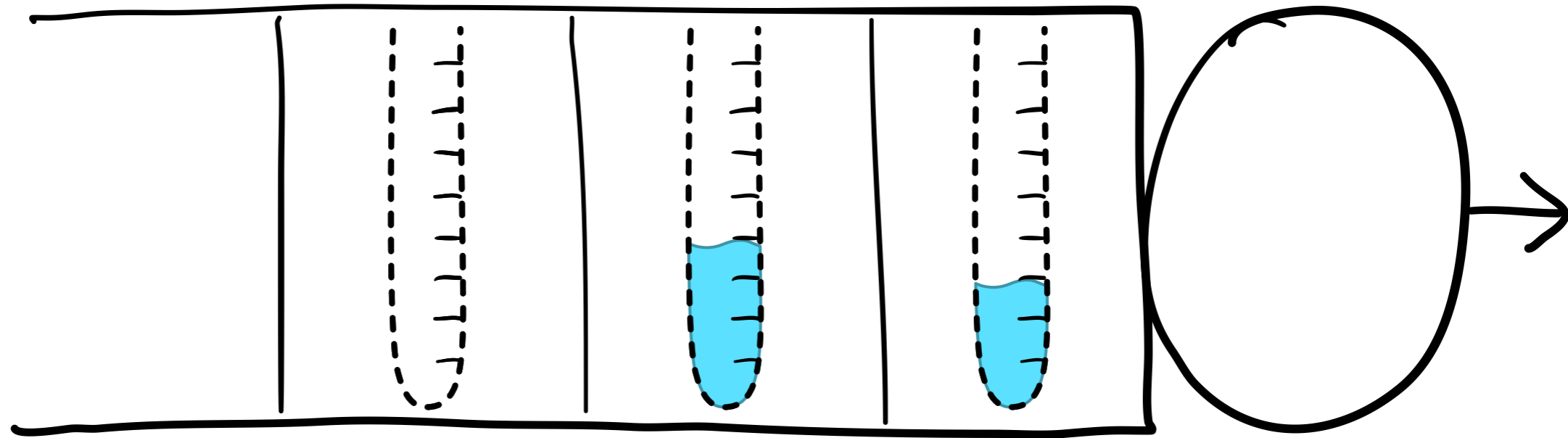
Unknown Job Sizes



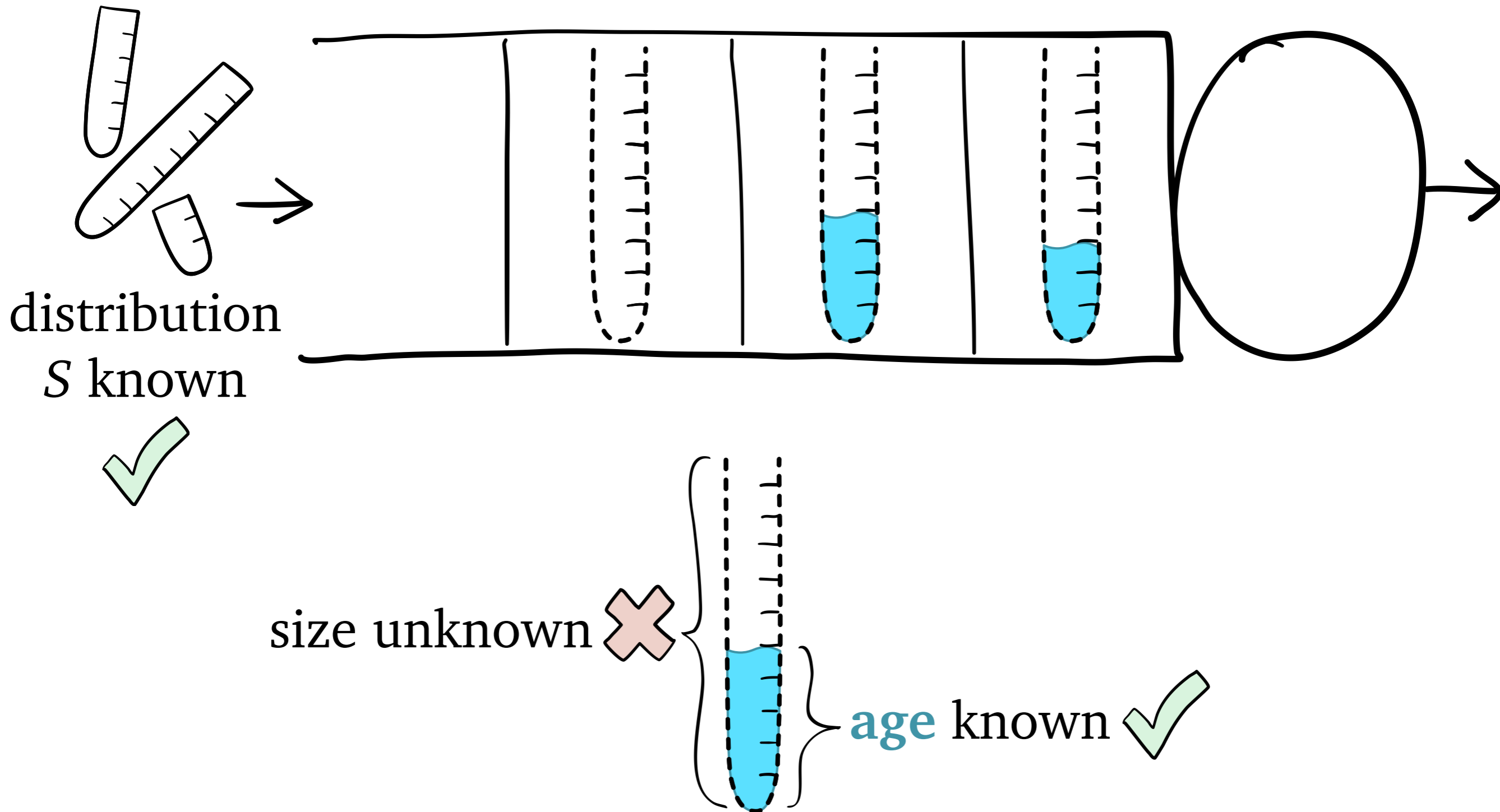
Unknown Job Sizes



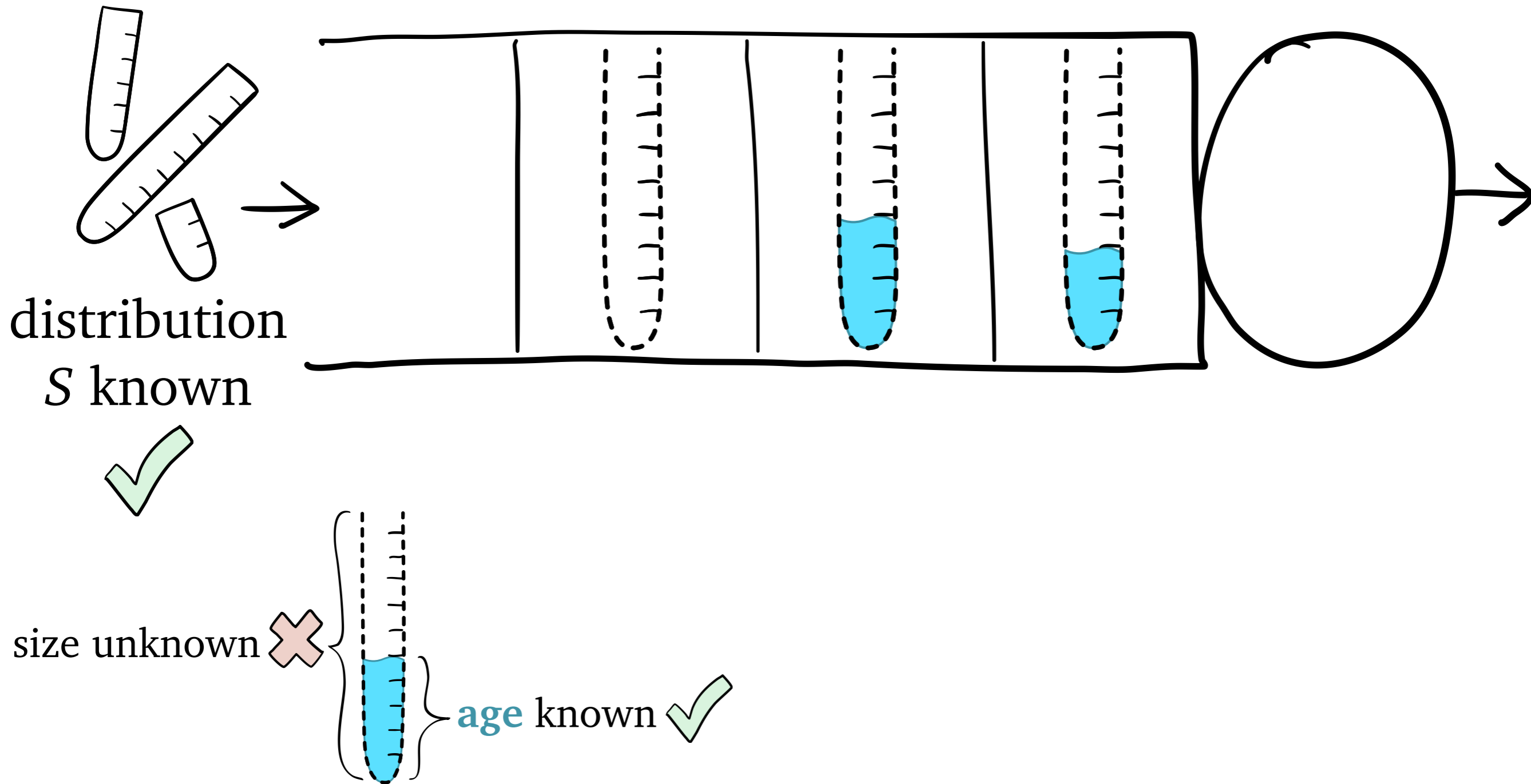
Unknown Job Sizes



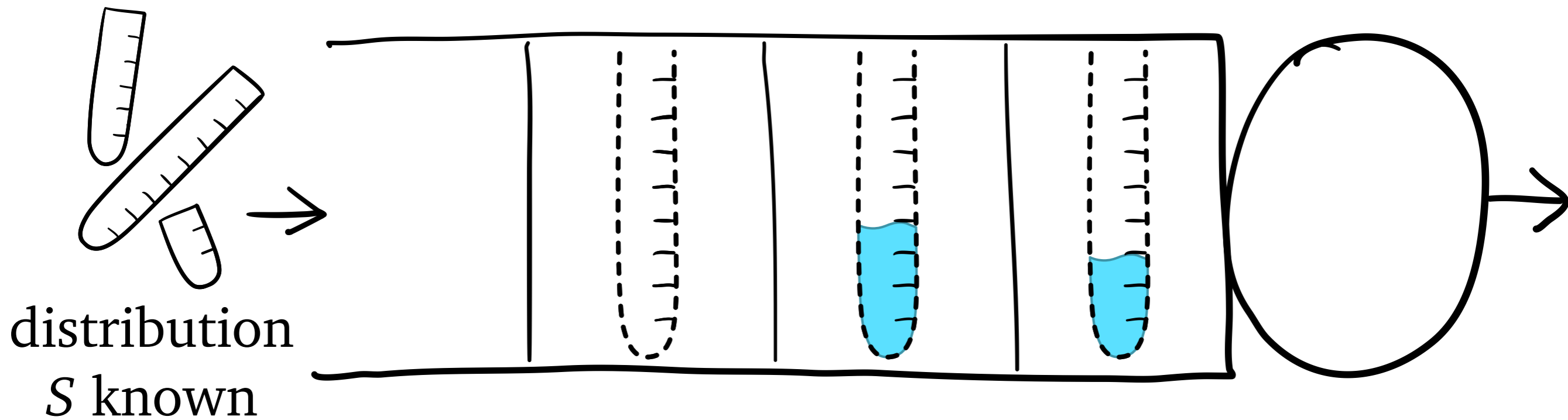
Unknown Job Sizes



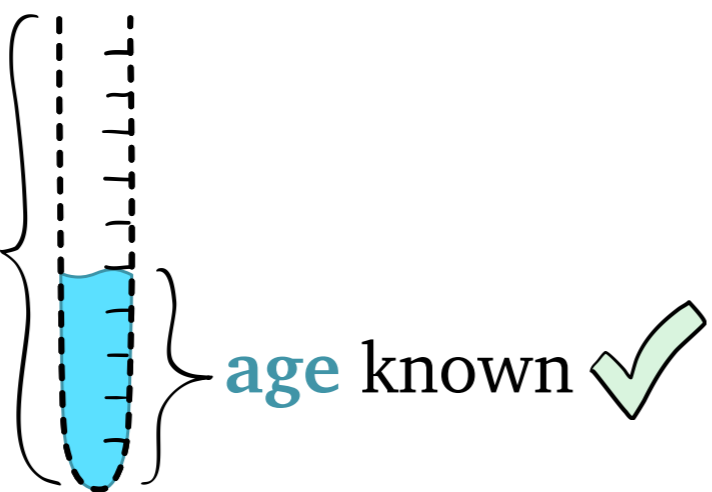
Unknown Job Sizes



Unknown Job Sizes

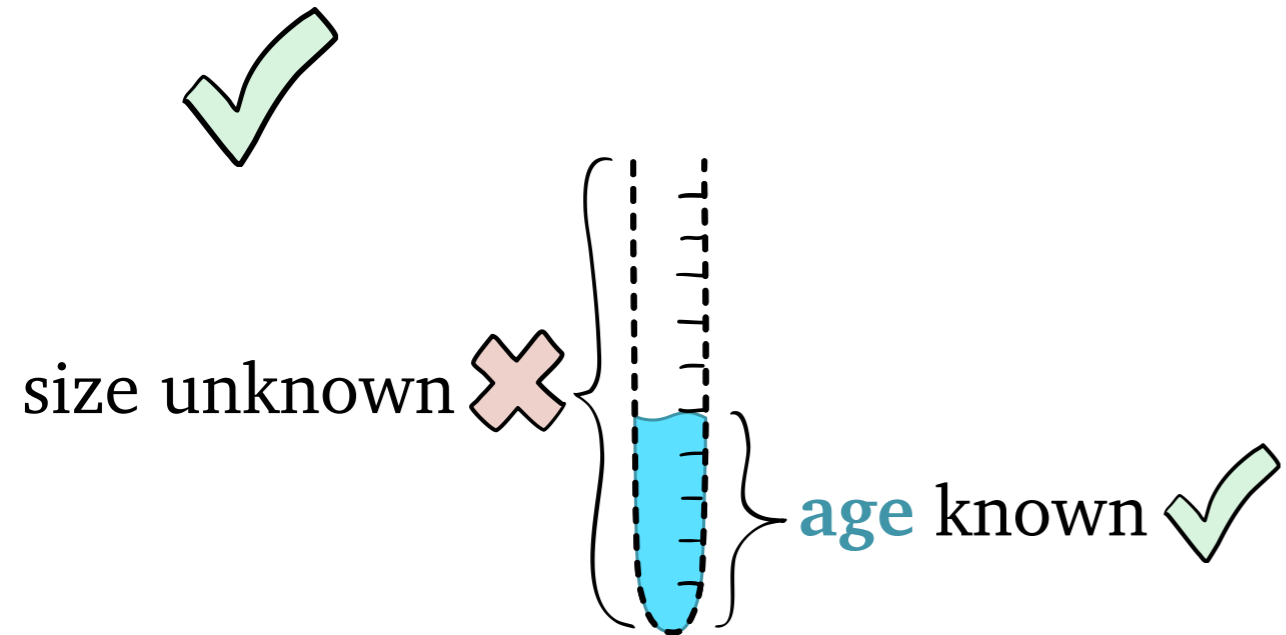
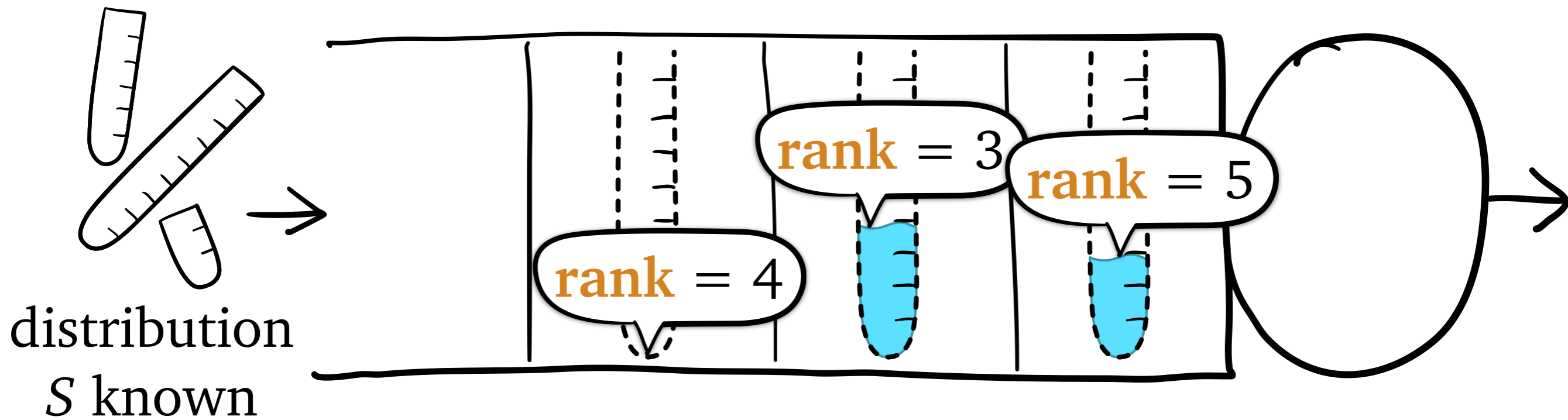



size unknown ✗



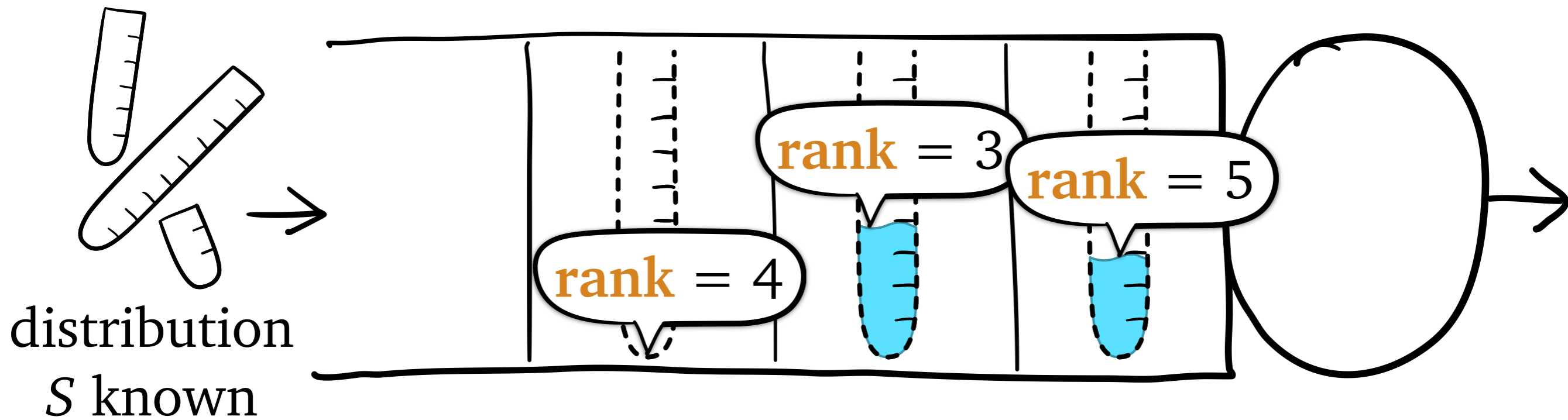
Gittins: assign each job a **rank** based on **age** and S
(lower is better)

Unknown Job Sizes



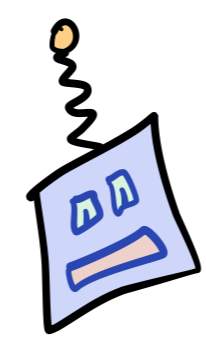
 **Gittins:** assign each job a **rank** based on **age** and S (lower is better)

Unknown Job Sizes



distribution S known
✓

size unknown ✗
age known ✓



Gittins: assign each job a **rank** based on **age** and S (lower is better)



Gittins minimizes $E[T]$

SRPT and Gittins
minimize $E[T]$

known job sizes

SRPT and Gittins
minimize $E[T]$

known job sizes

sizes unknown,
partially known, known
(subsumes SRPT), ...

SRPT and Gittins
minimize $E[T]$

known job sizes

sizes unknown,
partially known, known
(subsumes SRPT), ...

SRPT and Gittins
minimize $E[T]$

Why *not* use Gittins?

Gittins Assumption

Computer System Reality

Gittins Assumption

Computer System Reality

Single server

Gittins Assumption

Single server

Computer System Reality

Multiple servers

Gittins Assumption

Computer System Reality

Single server

Multiple servers

*Complicated implementation
not a problem*

Gittins Assumption

Single server

Complicated implementation
not a problem

Computer System Reality

Multiple servers

Simple implementation
preferred

Gittins Assumption

Single server

Complicated implementation
not a problem

Preemption *unrestricted*
with *no cost*

Computer System Reality

Multiple servers

Simple implementation
preferred

Gittins Assumption

Single server

Complicated implementation
not a problem

Preemption *unrestricted*
with *no cost*

Computer System Reality

Multiple servers

Simple implementation
preferred

Preemption *restricted*
and/or *costly*

Gittins Assumption

Single server

Complicated implementation
not a problem

Preemption *unrestricted*
with *no cost*

Arbitrarily many
priority levels

Computer System Reality

Multiple servers

Simple implementation
preferred

Preemption *restricted*
and/or *costly*

Gittins Assumption

Single server

Complicated implementation
not a problem

Preemption *unrestricted*
with *no cost*

Arbitrarily many
priority levels

Computer System Reality

Multiple servers

Simple implementation
preferred

Preemption *restricted*
and/or *costly*

Limited number
of priority levels

Gittins Assumption

Single server

Complicated implementation
not a problem

Preemption *unrestricted*
with *no cost*

Arbitrarily many
priority levels

Goal is minimizing
mean response time

Computer System Reality

Multiple servers

Simple implementation
preferred

Preemption *restricted*
and/or *costly*

Limited number
of priority levels

Gittins Assumption

Single server

Complicated implementation
not a problem

Preemption *unrestricted*
with *no cost*

Arbitrarily many
priority levels

Goal is minimizing
mean response time

Computer System Reality

Multiple servers

Simple implementation
preferred

Preemption *restricted*
and/or *costly*

Limited number
of priority levels

Want to optimize *other*
response time metrics

Gittins Assumption

Single server

*Complicated implementation
not a problem*

*Preemption unrestricted
with no cost*

*Arbitrarily many
priority levels*

*Goal is minimizing
mean response time*

Computer System Reality

Multiple servers

*Simple implementation
preferred*

*Preemption restricted
and/or costly*

*Limited number
of priority levels*

*Want to optimize other
response time metrics*

Gittins Assumption

Single server

*Complicated implementation
not a problem*

*Preemption unrestricted
with no cost*

*Arbitrarily many
priority levels*

*Goal is minimizing
mean response time*

Easy

Computer System Reality

Multiple servers

*Simple implementation
preferred*

*Preemption restricted
and/or costly*

*Limited number
of priority levels*

*Want to optimize other
response time metrics*

Gittins Assumption

Single server

*Complicated implementation
not a problem*

*Preemption unrestricted
with no cost*

*Arbitrarily many
priority levels*

*Goal is minimizing
mean response time*

Easy

Computer System Reality

Multiple servers

*Simple implementation
preferred*

*Preemption restricted
and/or costly*

*Limited number
of priority levels*

*Want to optimize other
response time metrics*

Hard!

I work on inventing
new queueing-theoretic tools
for solving
practical scheduling problems

Overview

Goals

Multiple servers

Simple implementation preferred

Preemption restricted and/or costly

Limited number of priority levels

Want to optimize other response time metrics

Overview

New Tools

Goals

Multiple servers

Simple implementation preferred

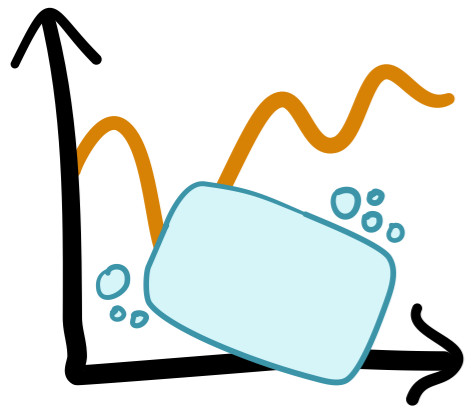
Preemption restricted and/or costly

Limited number of priority levels

Want to optimize other response time metrics

Overview

New Tools



SOAP

*analyzes a huge variety
of scheduling heuristics*

Goals

Multiple servers

*Simple implementation
preferred*

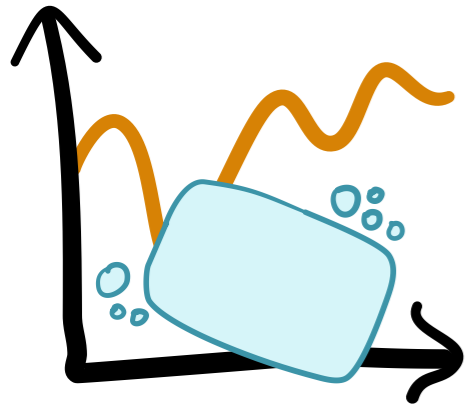
*Preemption restricted
and/or costly*

*Limited number
of priority levels*

*Want to optimize other
response time metrics*

Overview

New Tools



SOAP

analyzes a huge variety of scheduling heuristics

r-Work

provides a new, deeper understanding of Gittins

Goals

Multiple servers

Simple implementation preferred

Preemption restricted and/or costly

Limited number of priority levels

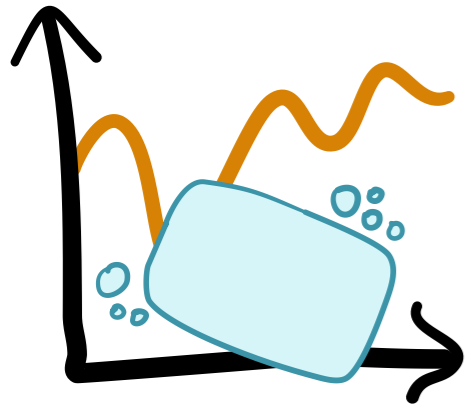
Want to optimize other response time metrics

Overview

New Tools

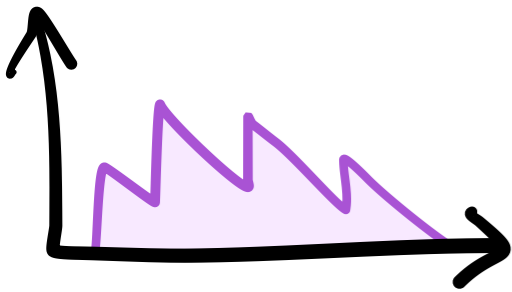
SOAP

analyzes a huge variety of scheduling heuristics



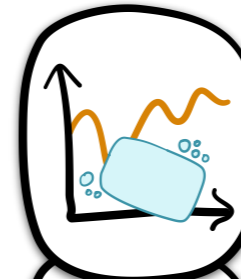
r-Work

provides a new, deeper understanding of Gittins

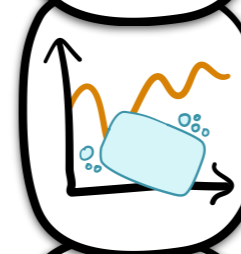


Goals

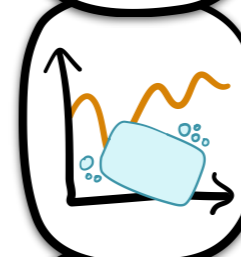
Multiple servers



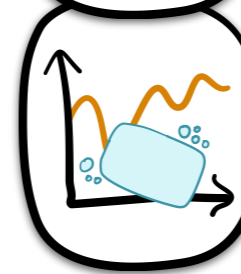
Simple implementation preferred



Preemption restricted and/or costly



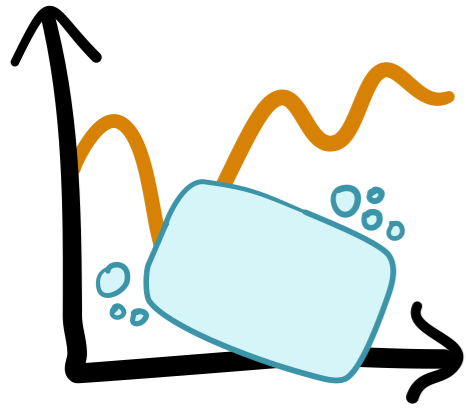
Limited number of priority levels



Want to optimize other response time metrics

Overview

New Tools



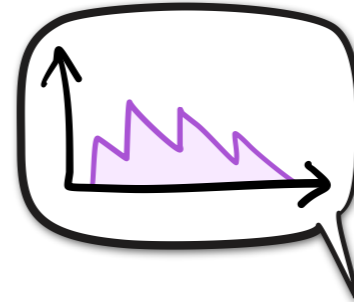
SOAP

analyzes a huge variety of scheduling heuristics

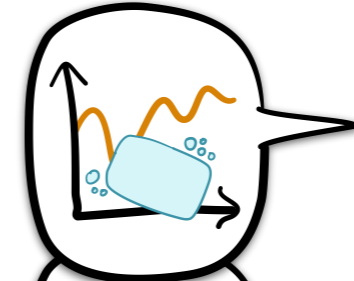
r-Work

provides a new, deeper understanding of Gittins

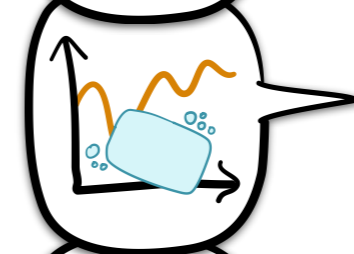
Goals



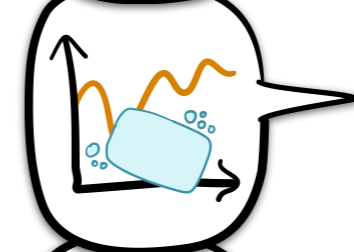
Multiple servers



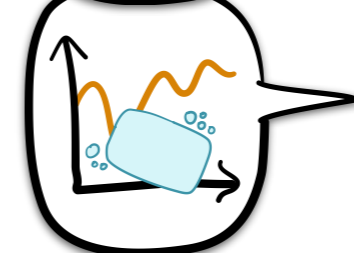
Simple implementation preferred



Preemption restricted and/or costly



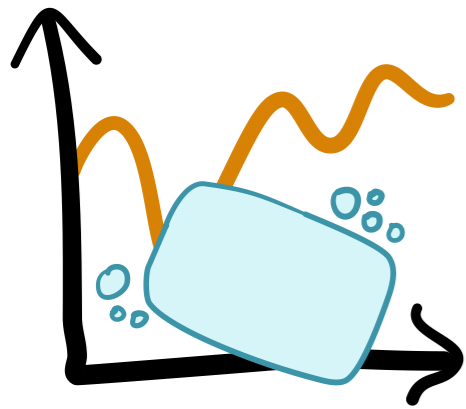
Limited number of priority levels



Want to optimize other response time metrics

Overview

New Tools



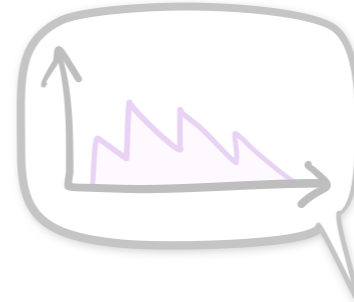
SOAP

analyzes a huge variety of scheduling heuristics

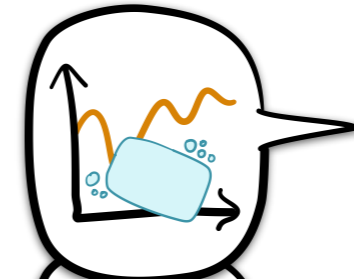
r-Work

provides a new, deeper understanding of Gittins

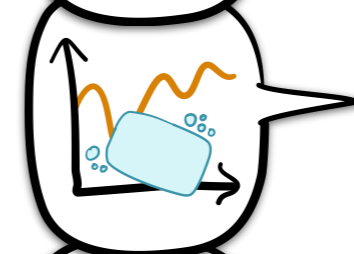
Goals



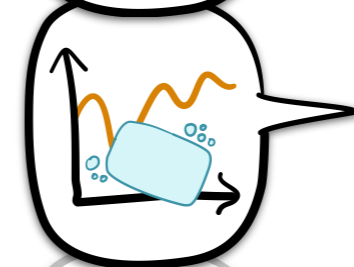
Multiple servers



Simple implementation preferred



Preemption restricted and/or costly



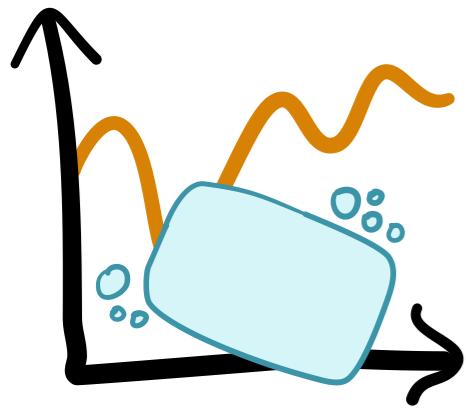
Limited number of priority levels



Want to optimize other response time metrics

Overview

New Tools

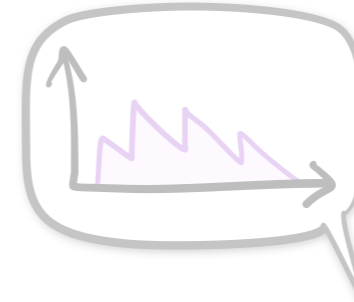


SOAP

analyzes a huge variety of scheduling heuristics

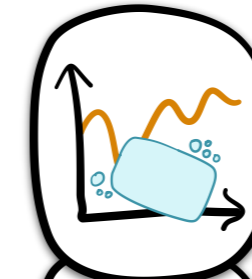
r-Work

provides a new, deeper understanding of Gittins

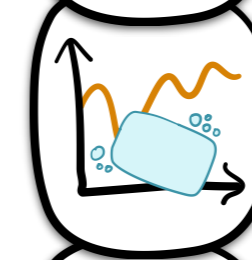


PERFORMANCE 2018,
SIGMETRICS 2019,
SIGMETRICS 2021

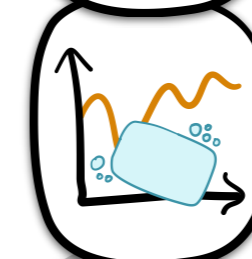
Multiple servers



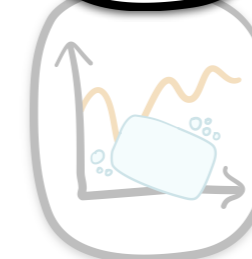
Simple implementation preferred



Preemption restricted and/or costly



Limited number of priority levels

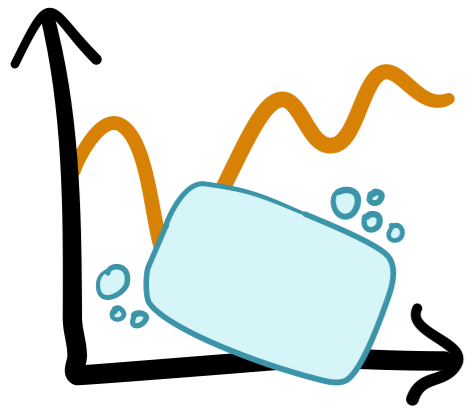


Want to optimize other response time metrics

SIGMETRICS 2020,
SIGMETRICS 2021

Overview

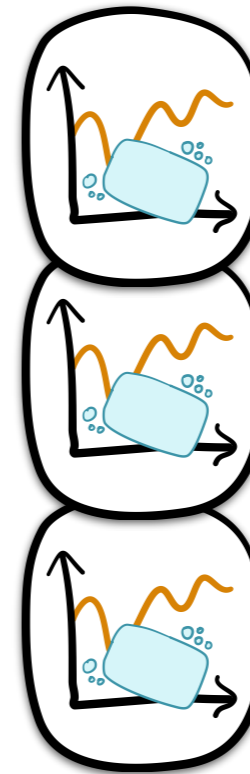
New Tools



SOAP

analyzes a huge variety of scheduling heuristics

Goals



Simple implementation preferred

Preemption restricted and/or costly

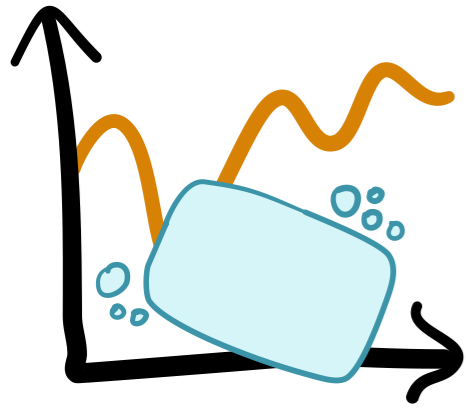
Limited number of priority levels

Overview

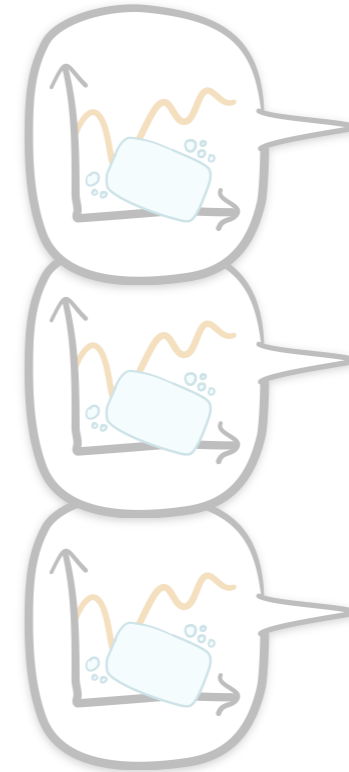
New Tools

SOAP

*analyzes a huge variety
of scheduling heuristics*



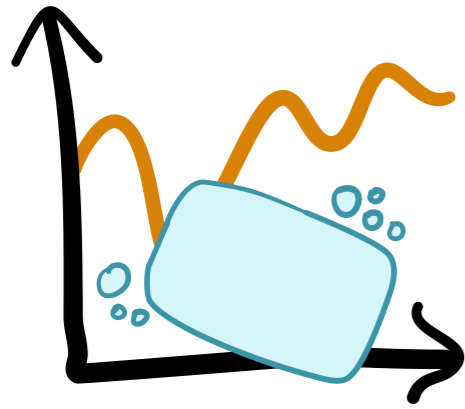
Goals



*Simple implementation
preferred*

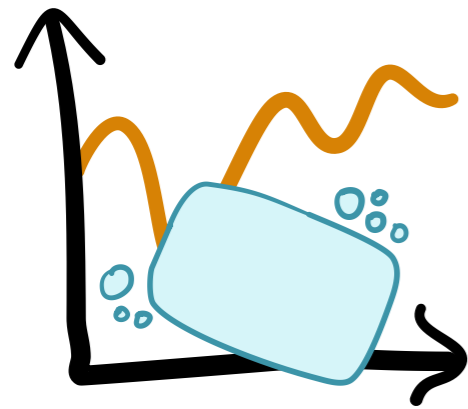
*Preemption restricted
and/or costly*

*Limited number
of priority levels*



SOAP

Schedule Ordered by Age-based Priority

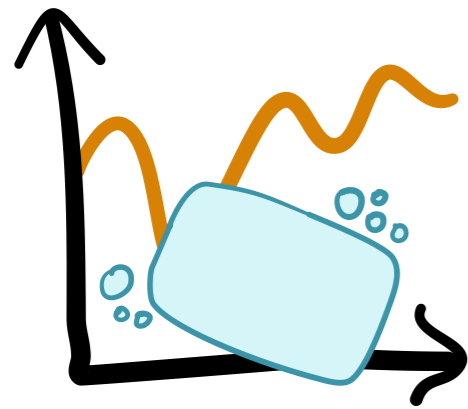


SOAP

Schedule Ordered by Age-based Priority

SOAP policies:

broad class of scheduling policies



SOAP

Schedule Ordered by Age-based Priority

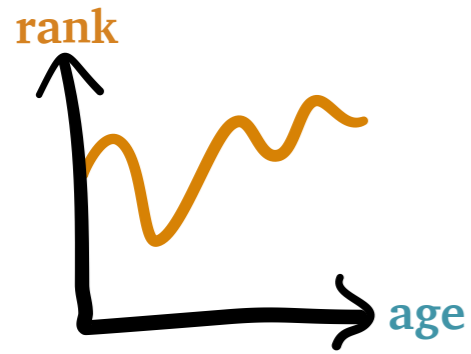
SOAP policies:

broad class of scheduling policies

SOAP analysis:

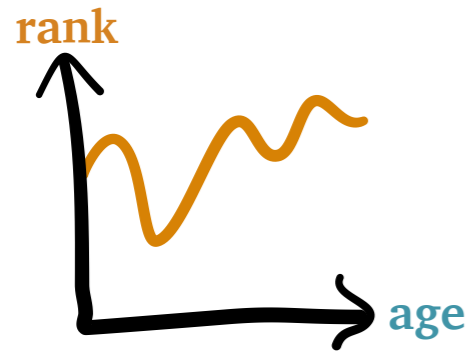
analyze response time of *any* **SOAP** policy

SOAP Policies



SOAP policy: any scheduling policy where a job's **rank** is a function of its **age**

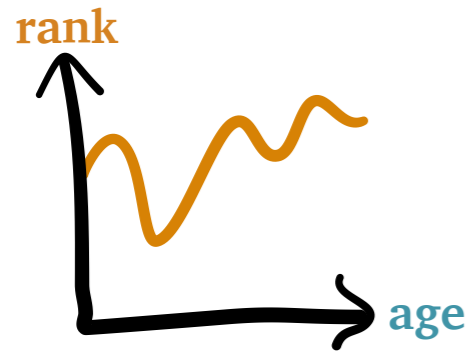
SOAP Policies



SOAP policy: any scheduling policy where a job's **rank** is a function of its **age**

priority, lower is better

SOAP Policies

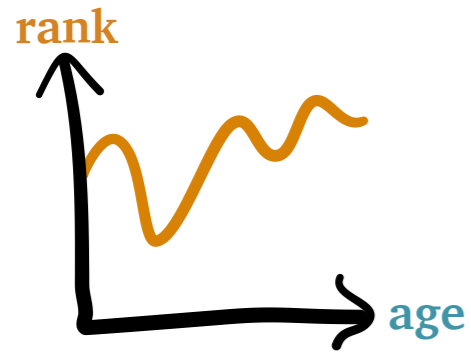


SOAP policy: any scheduling policy where a job's **rank** is a function of its **age**

priority, lower is better

service so far

SOAP Policies

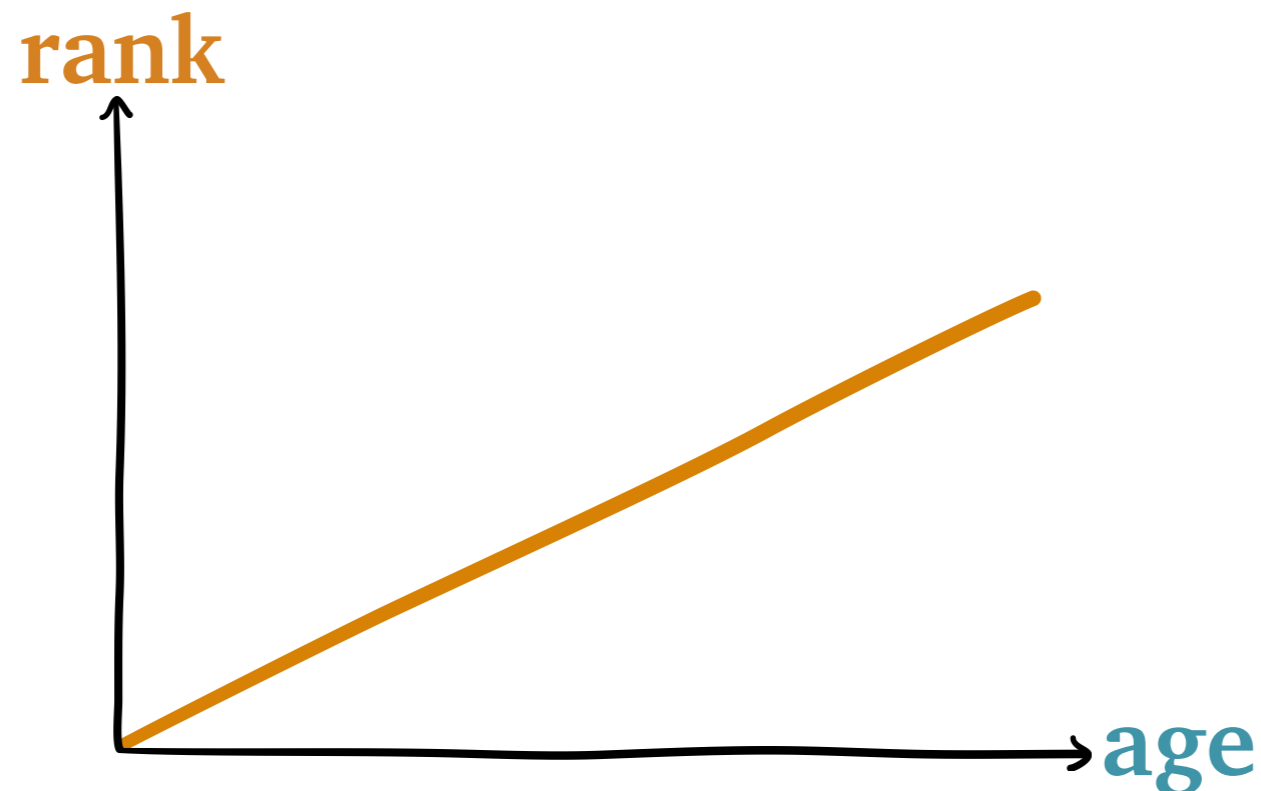


SOAP policy: any scheduling policy where a job's **rank** is a function of its **age**

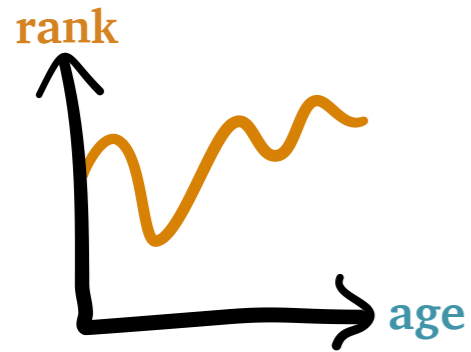
priority, lower is better

service so far

Foreground-Background (FB)



SOAP Policies

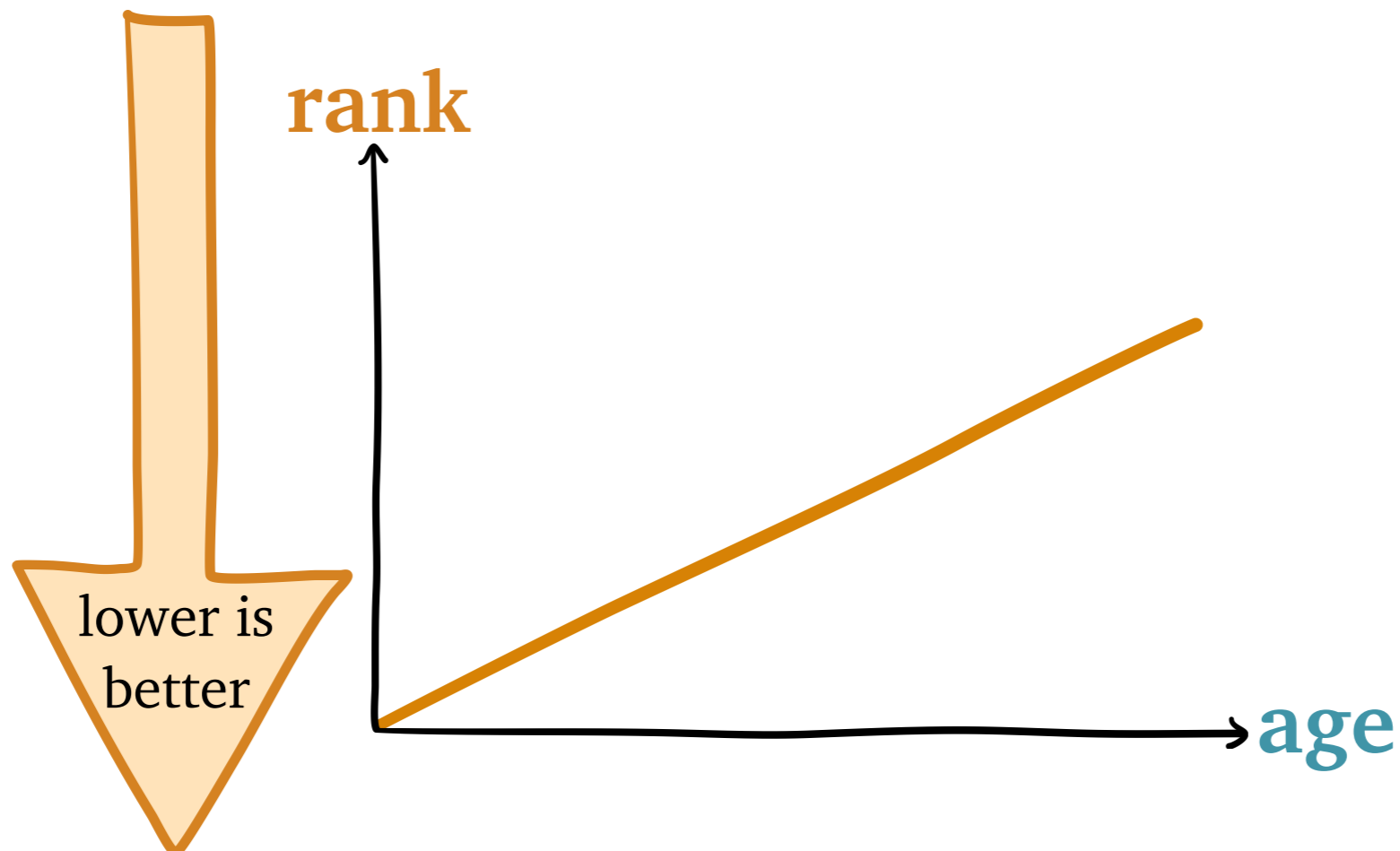


SOAP policy: any scheduling policy where a job's **rank** is a function of its **age**

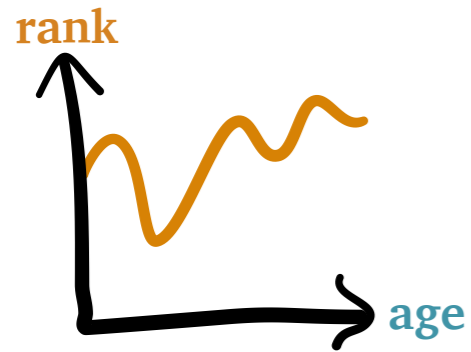
priority, lower is better

service so far

Foreground-Background (FB)



SOAP Policies

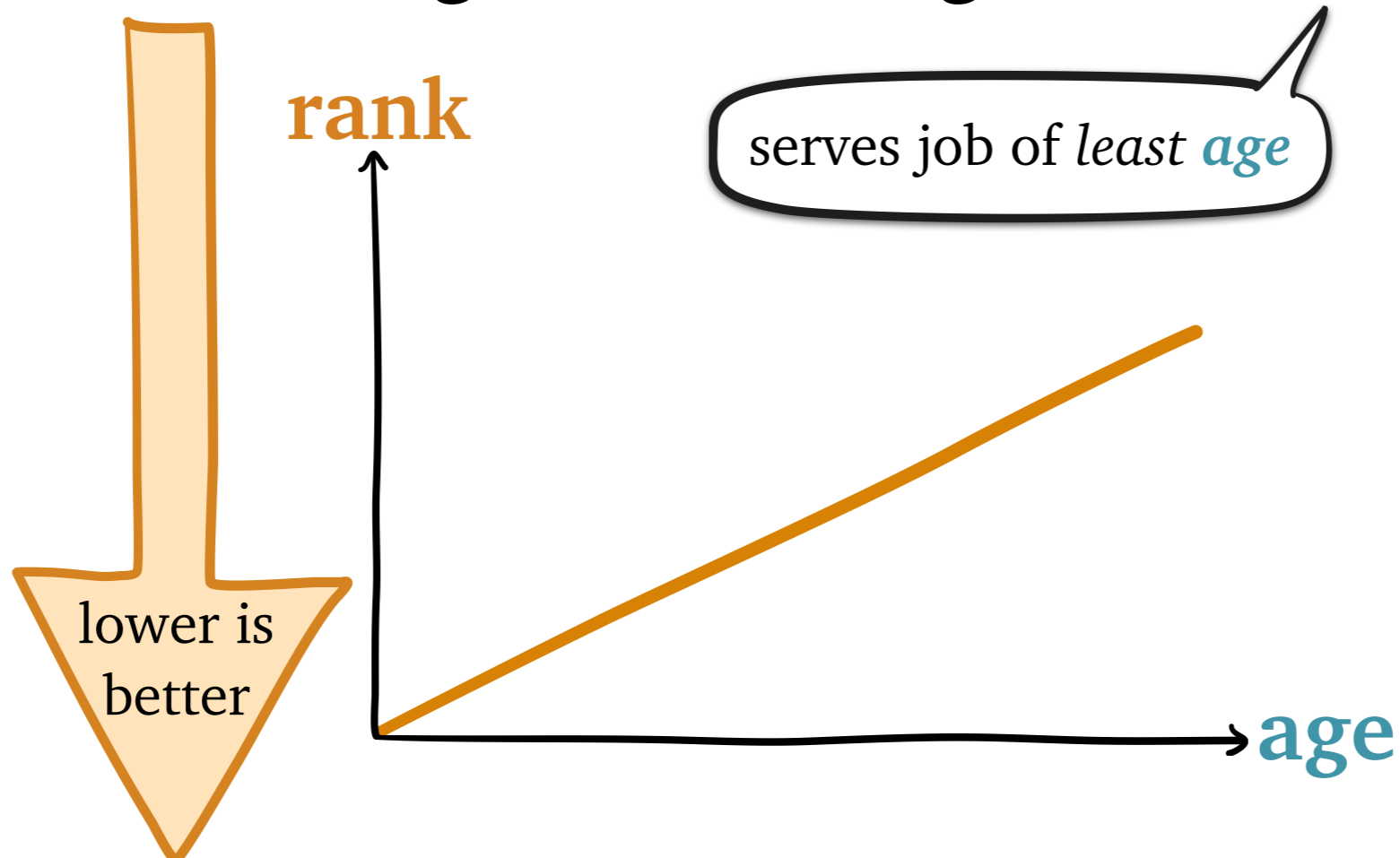


SOAP policy: any scheduling policy where a job's **rank** is a function of its **age**

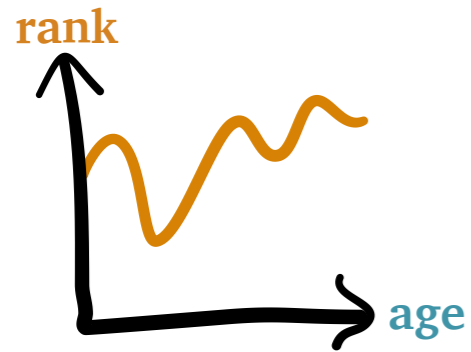
priority, lower is better

service so far

Foreground-Background (FB)



SOAP Policies

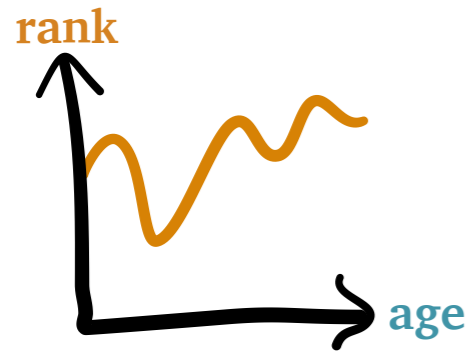


SOAP policy: any scheduling policy where a job's **rank** is a function of its **age**

priority, lower is better

service so far

SOAP Policies



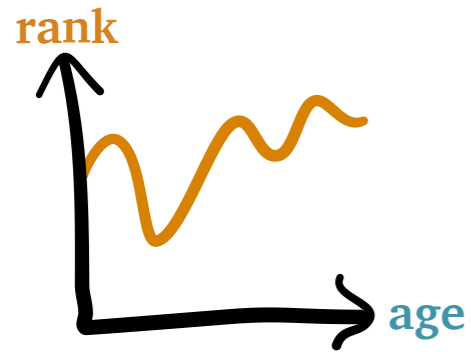
SOAP policy: any scheduling policy where a job's **rank** is a function of its **age**

priority, lower is better

and other "static" info

service so far

SOAP Policies



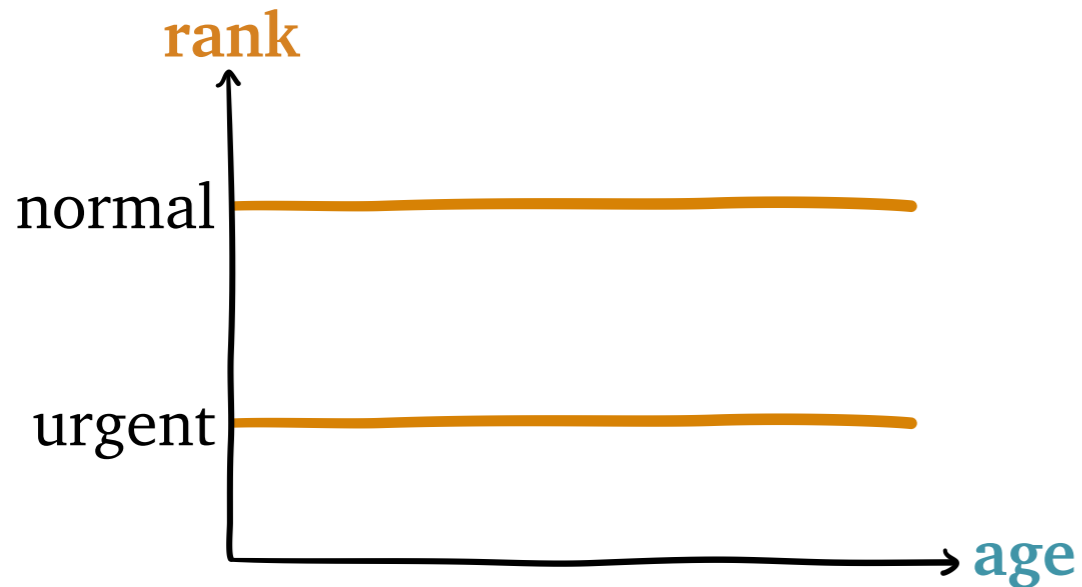
SOAP policy: any scheduling policy where a job's **rank** is a function of its **age**

priority, lower is better

and other "static" info

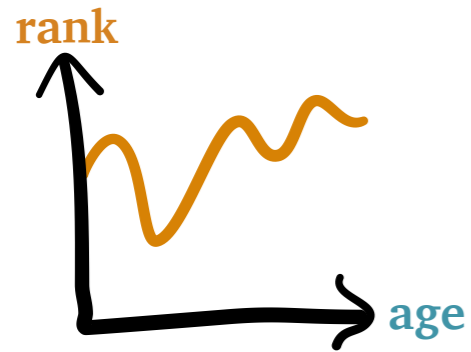
service so far

Preemptive Priority



lower is better

SOAP Policies



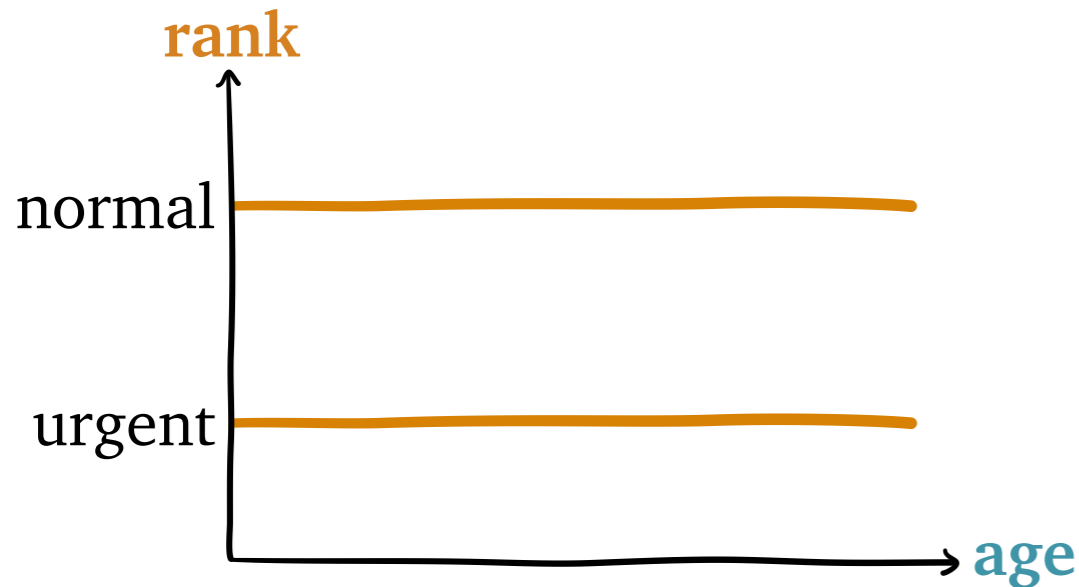
SOAP policy: any scheduling policy where a job's **rank** is a function of its **age**

priority, lower is better

and other "static" info

service so far

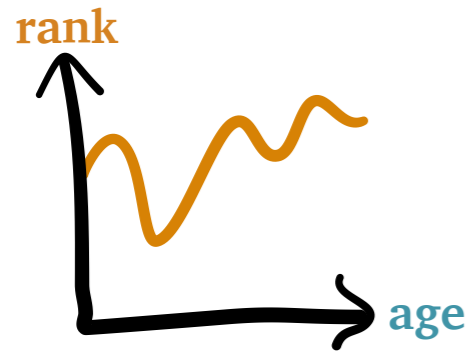
Preemptive Priority



break ties FCFS

lower is better

SOAP Policies



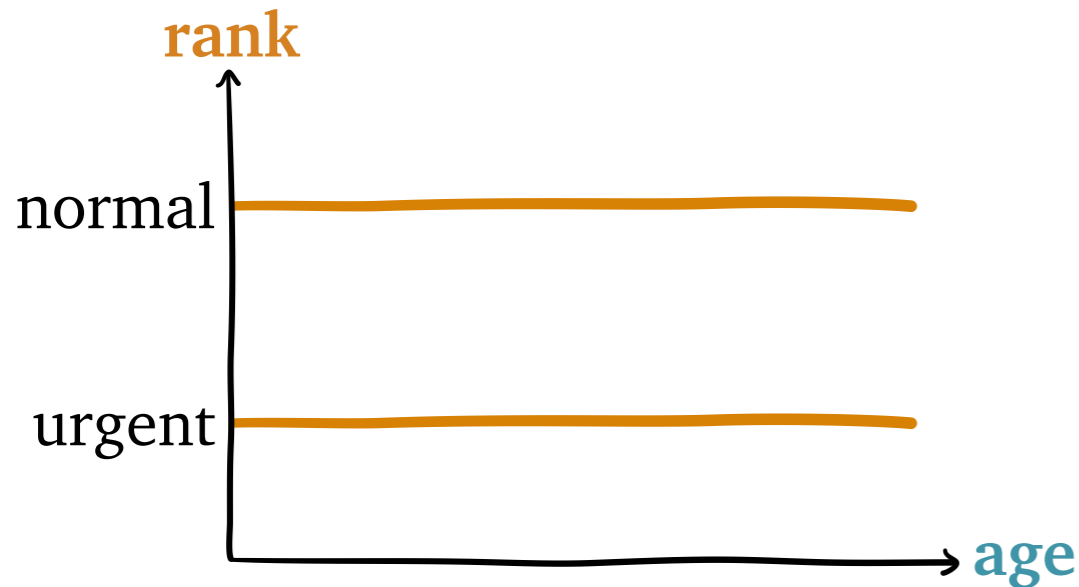
SOAP policy: any scheduling policy where a job's **rank** is a function of its **age**

priority, lower is better

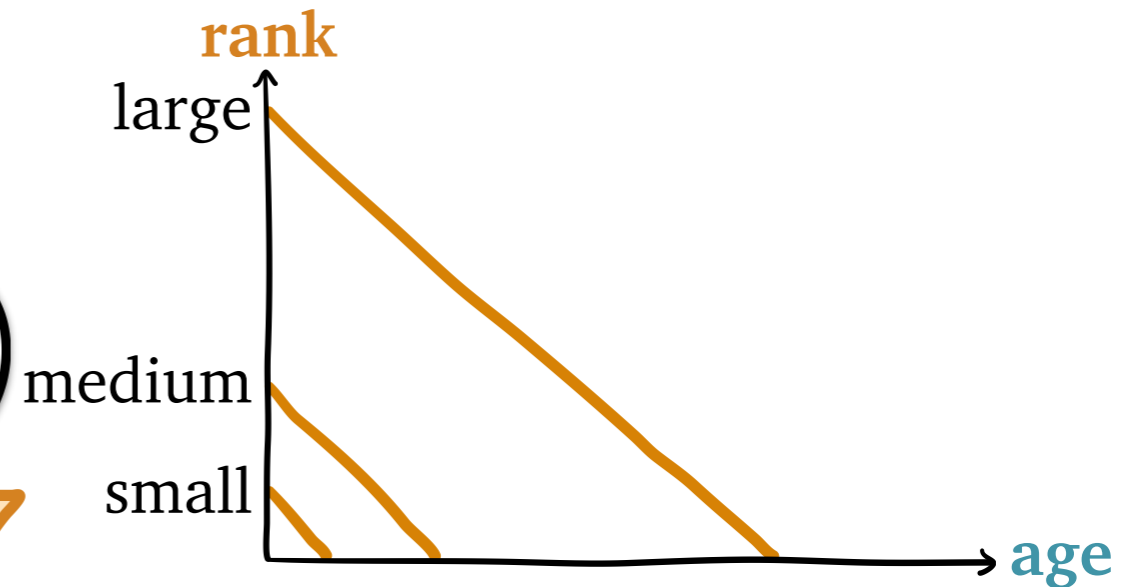
and other "static" info

service so far

Preemptive Priority



SRPT

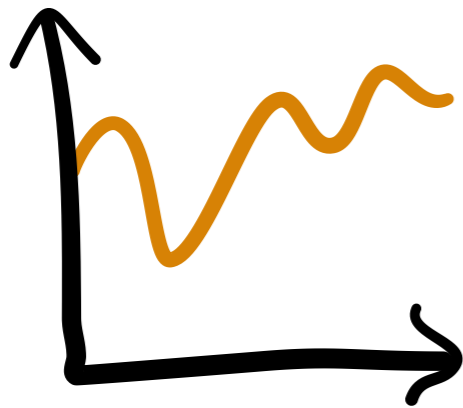


break ties FCFS

lower is better

SOAP Analysis

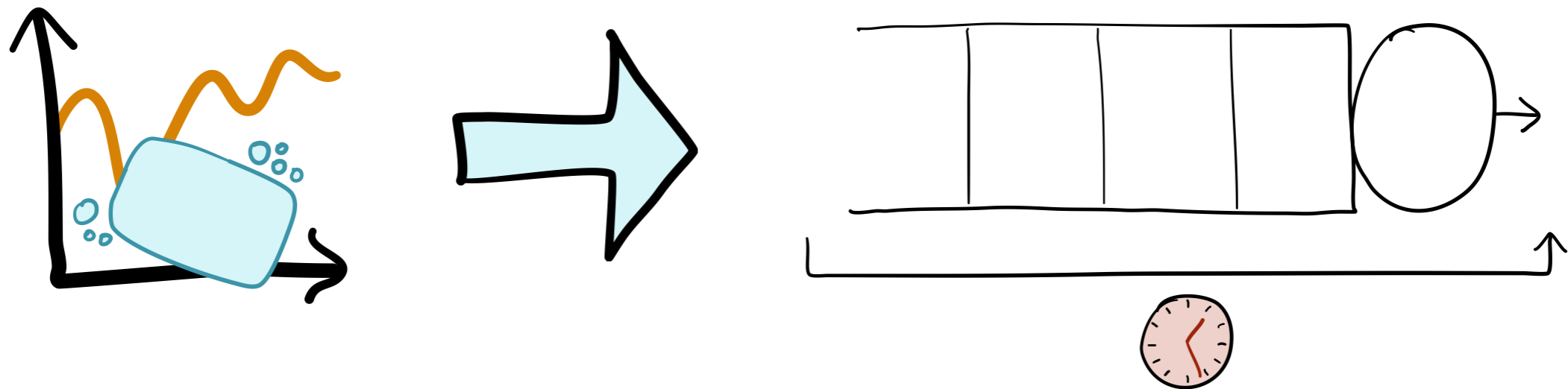
Given *any* **rank** function...



SOAP Analysis

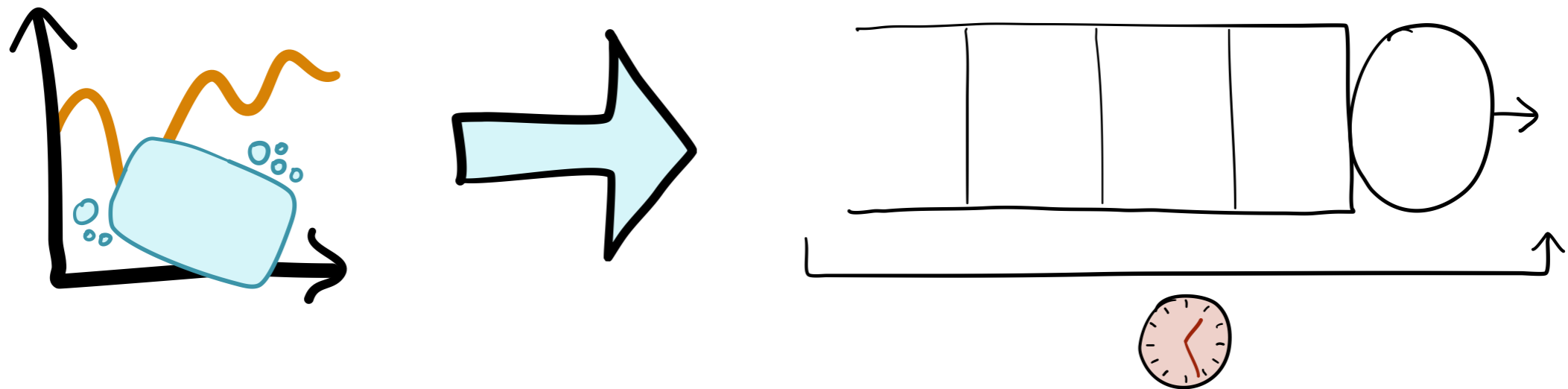
Given *any* **rank** function...

... **SOAP** analyzes its response time



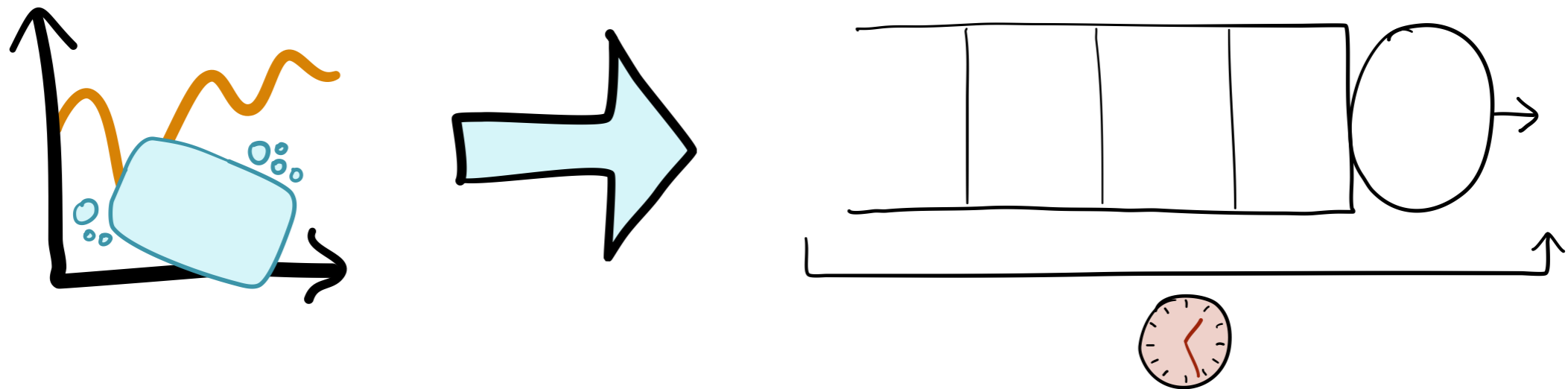
SOAP Analysis

Given *any* **rank** function... exact formula!
... **SOAP** analyzes its response time



SOAP Analysis

Given *any* **rank** function... exact formula!
... **SOAP** analyzes its response time



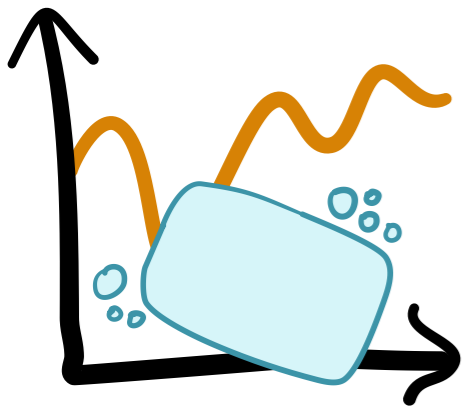
[Scully, Harchol-Balter, & Scheller-Wolf, SIGMETRICS 2018]

Overview

New Tools

SOAP

*analyzes a huge variety
of scheduling heuristics*



Goals

*Simple implementation
preferred*

*Preemption restricted
and/or costly*

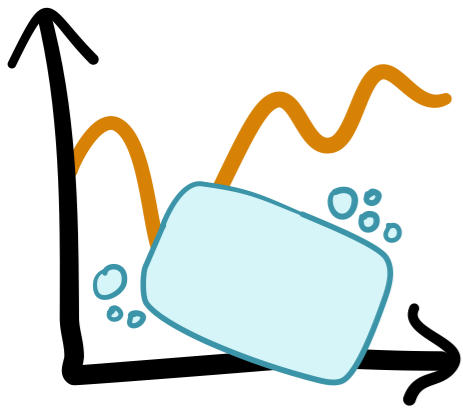
*Limited number
of priority levels*

Overview

New Tools

SOAP

*analyzes a huge variety
of scheduling heuristics*



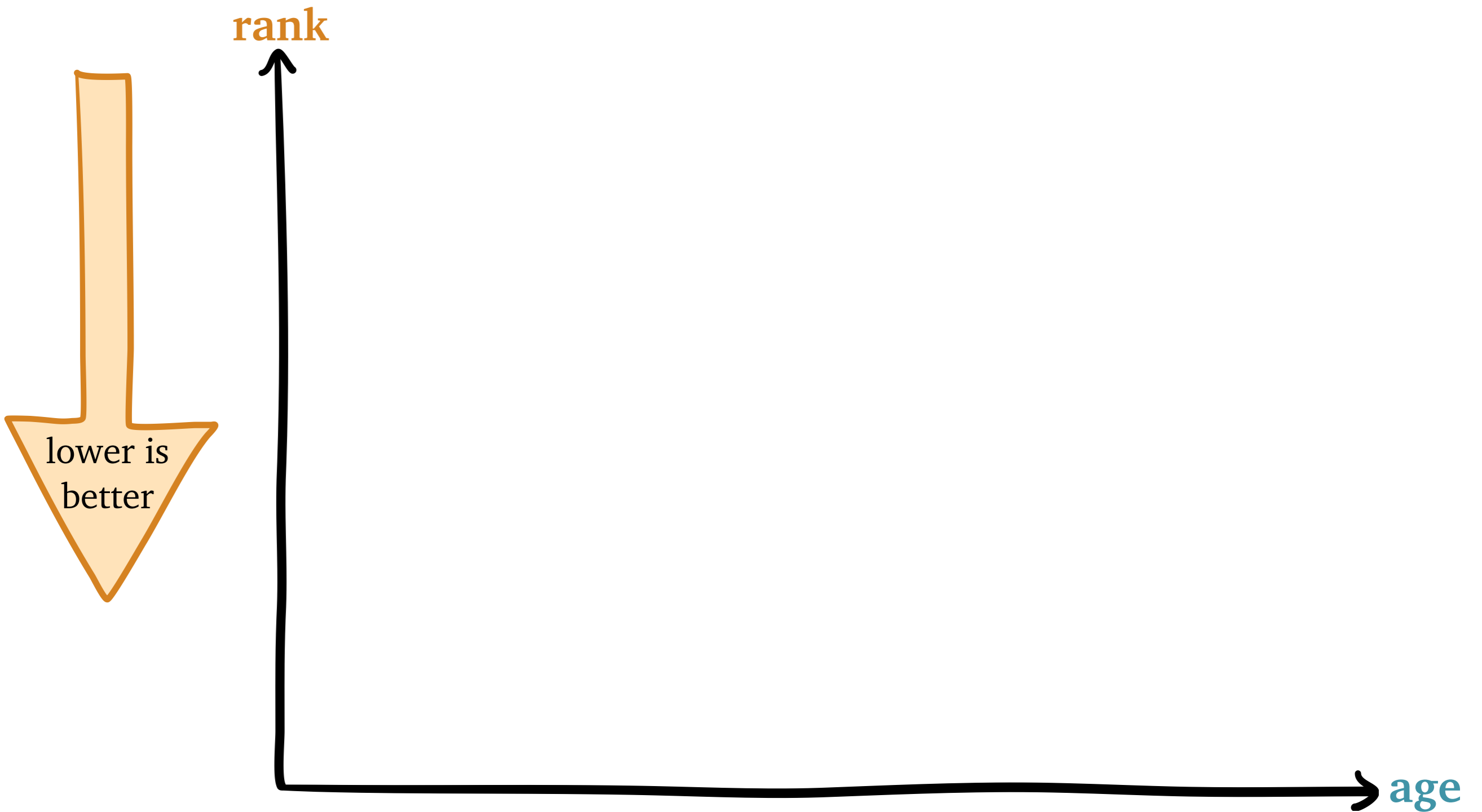
Goals

*Simple implementation
preferred*

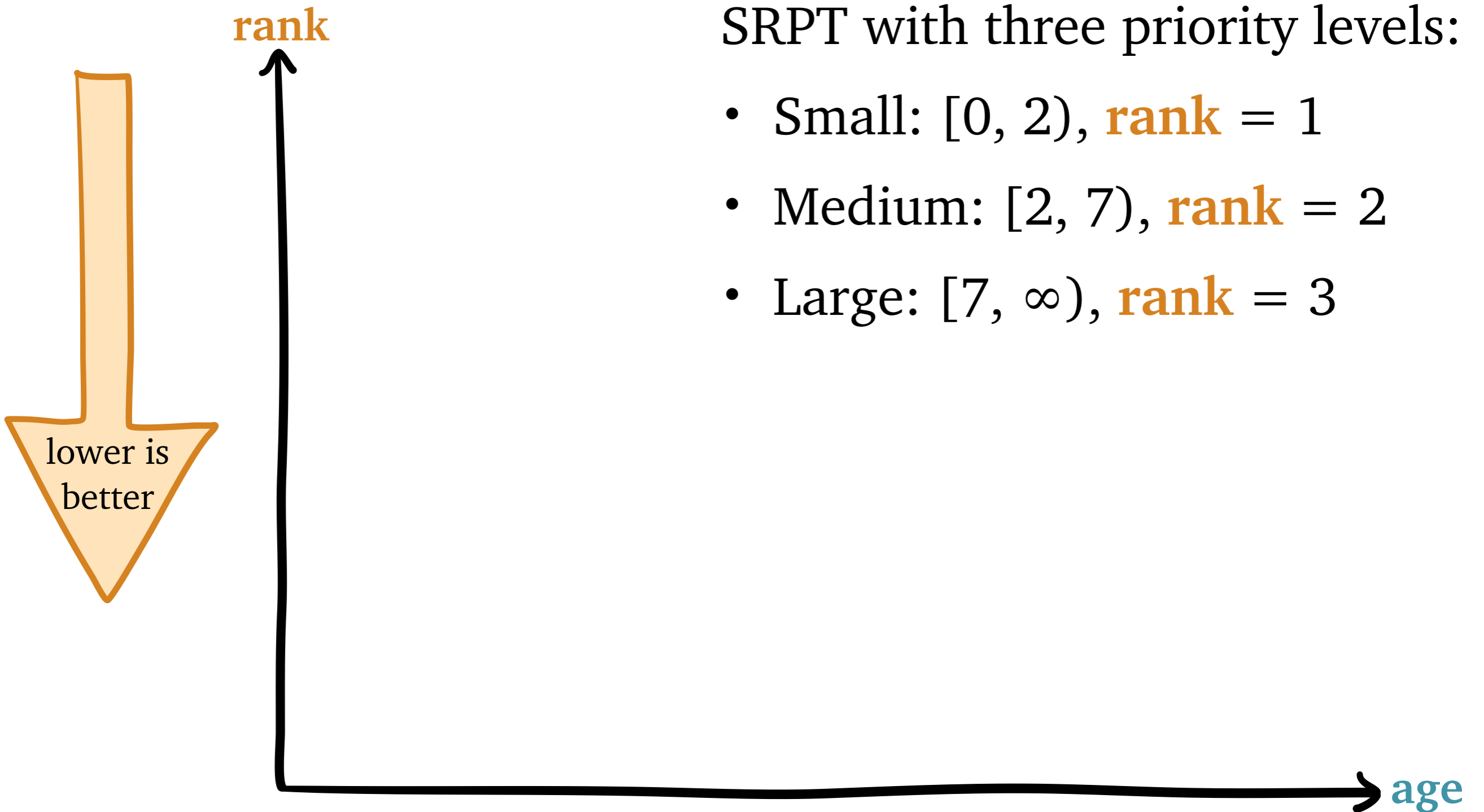
*Preemption restricted
and/or costly*

*Limited number
of priority levels*

Limited Priority Levels



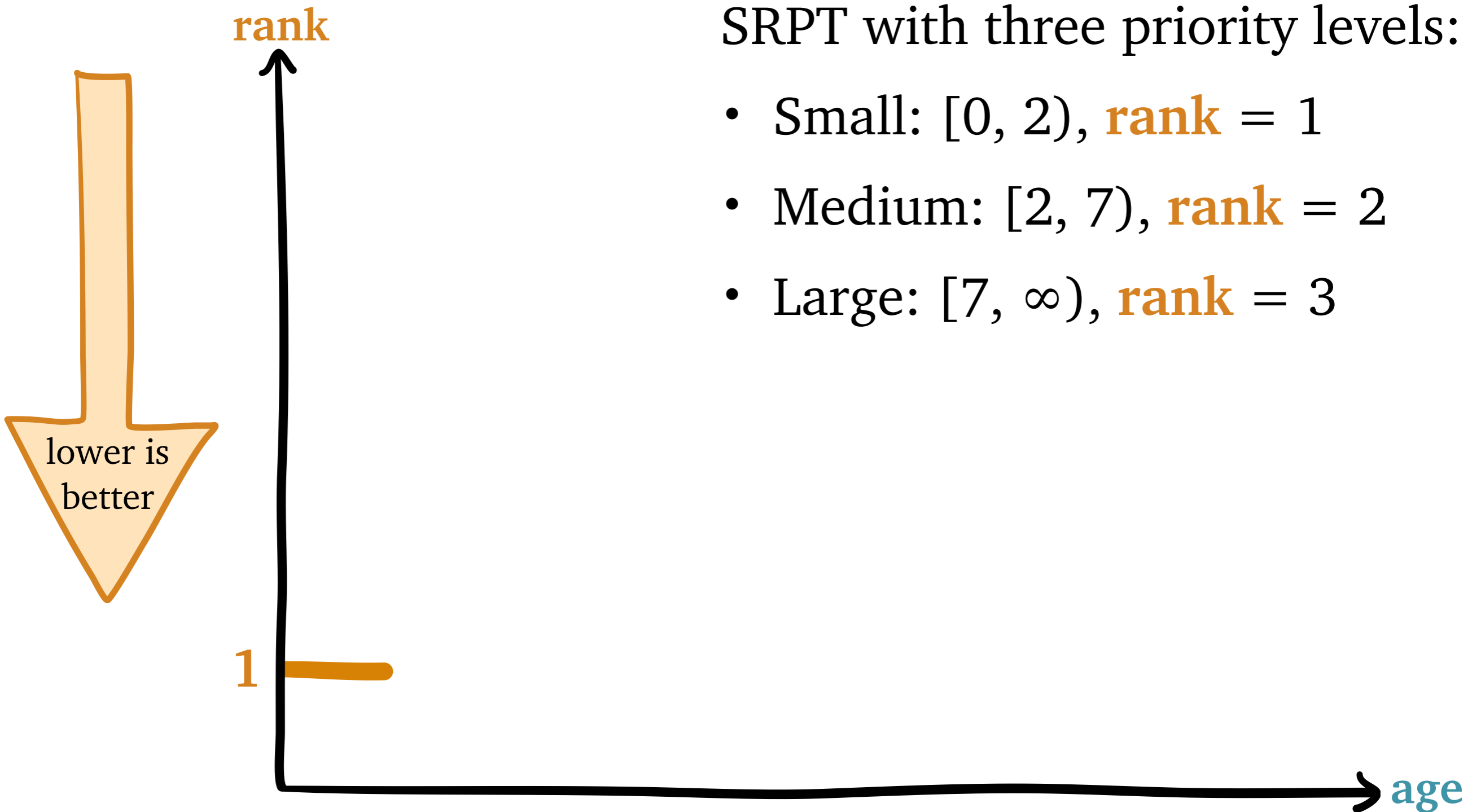
Limited Priority Levels



Limited Priority Levels

SRPT with three priority levels:

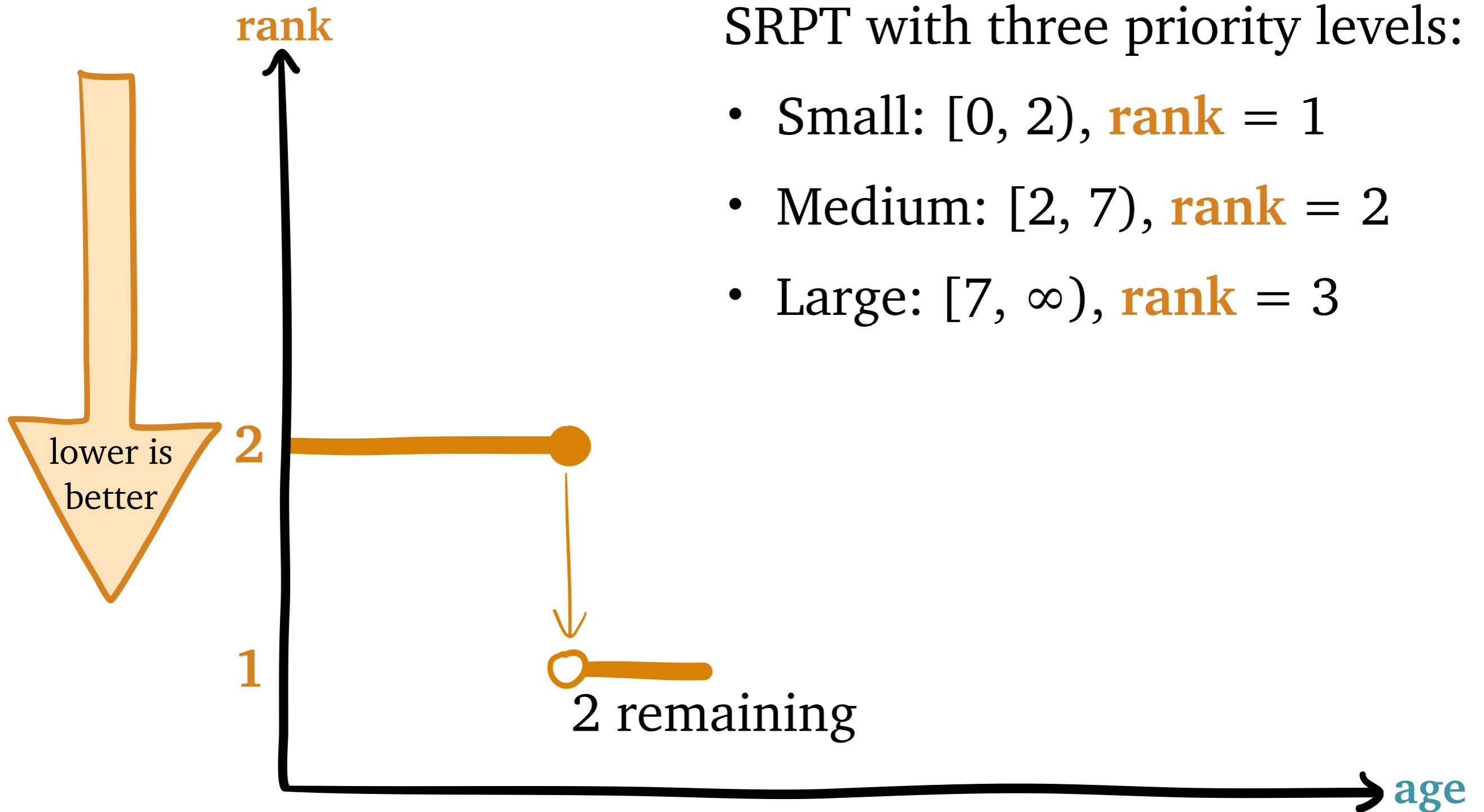
- Small: $[0, 2)$, **rank** = 1
- Medium: $[2, 7)$, **rank** = 2
- Large: $[7, \infty)$, **rank** = 3



Limited Priority Levels

SRPT with three priority levels:

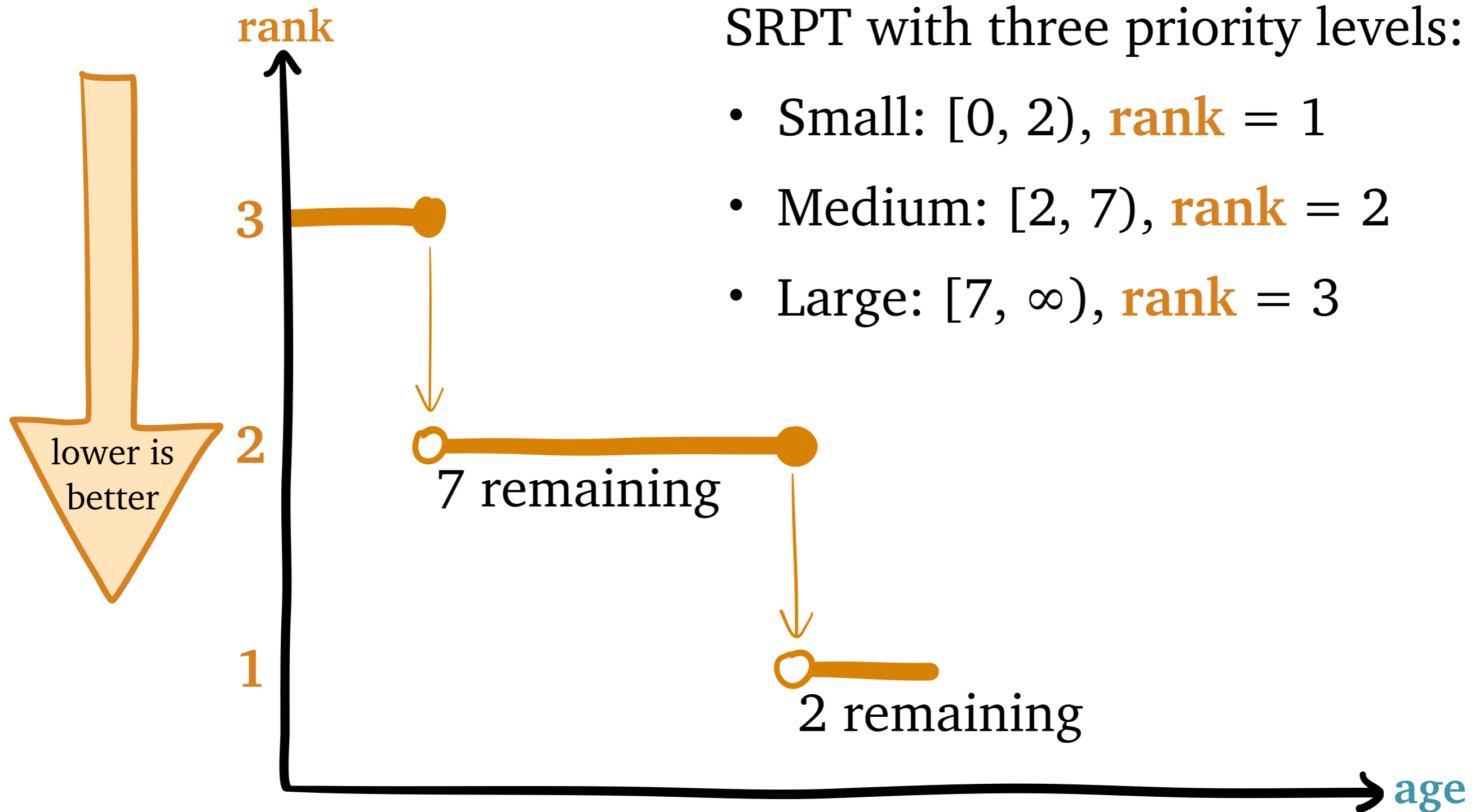
- Small: $[0, 2)$, **rank** = 1
- Medium: $[2, 7)$, **rank** = 2
- Large: $[7, \infty)$, **rank** = 3



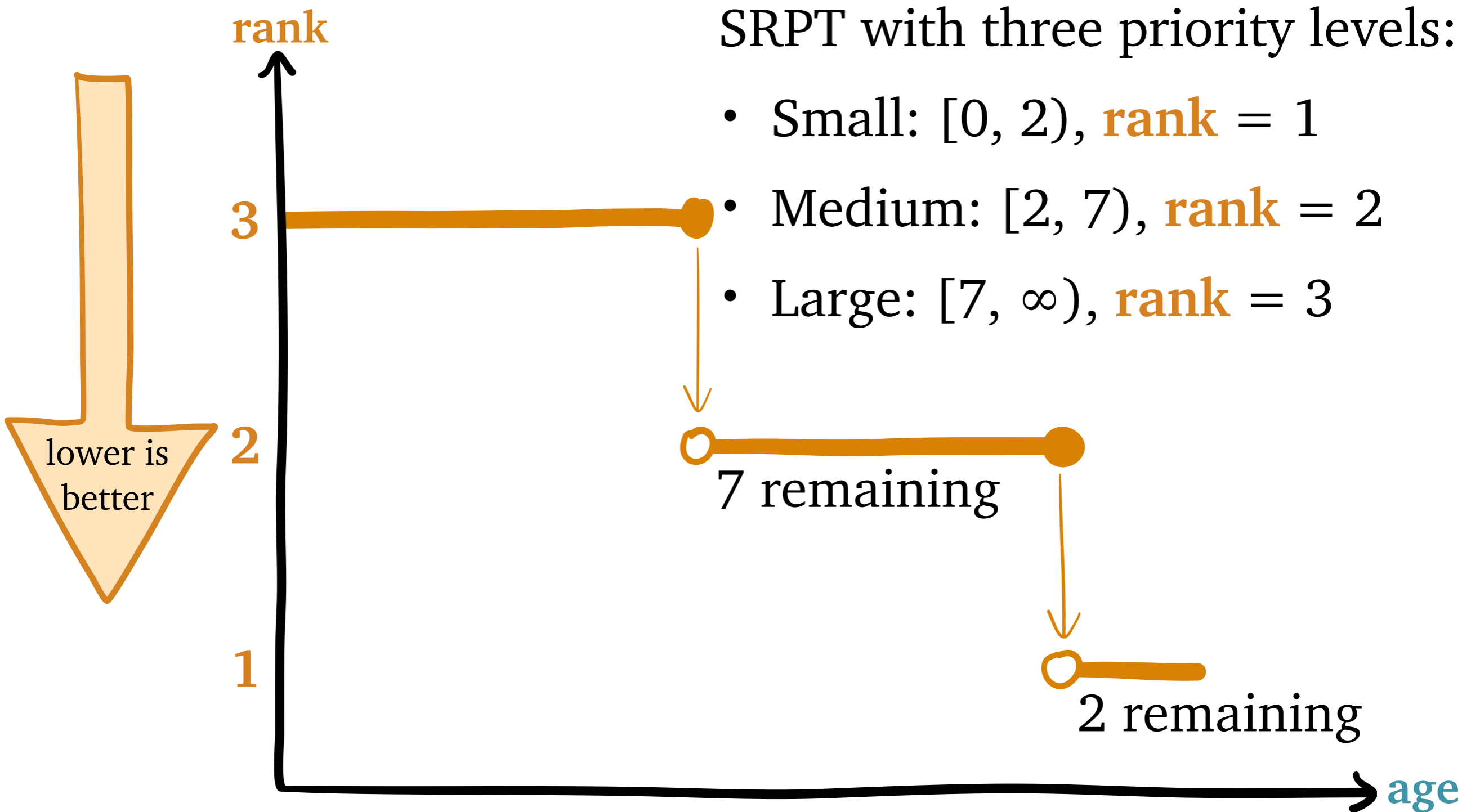
Limited Priority Levels

SRPT with three priority levels:

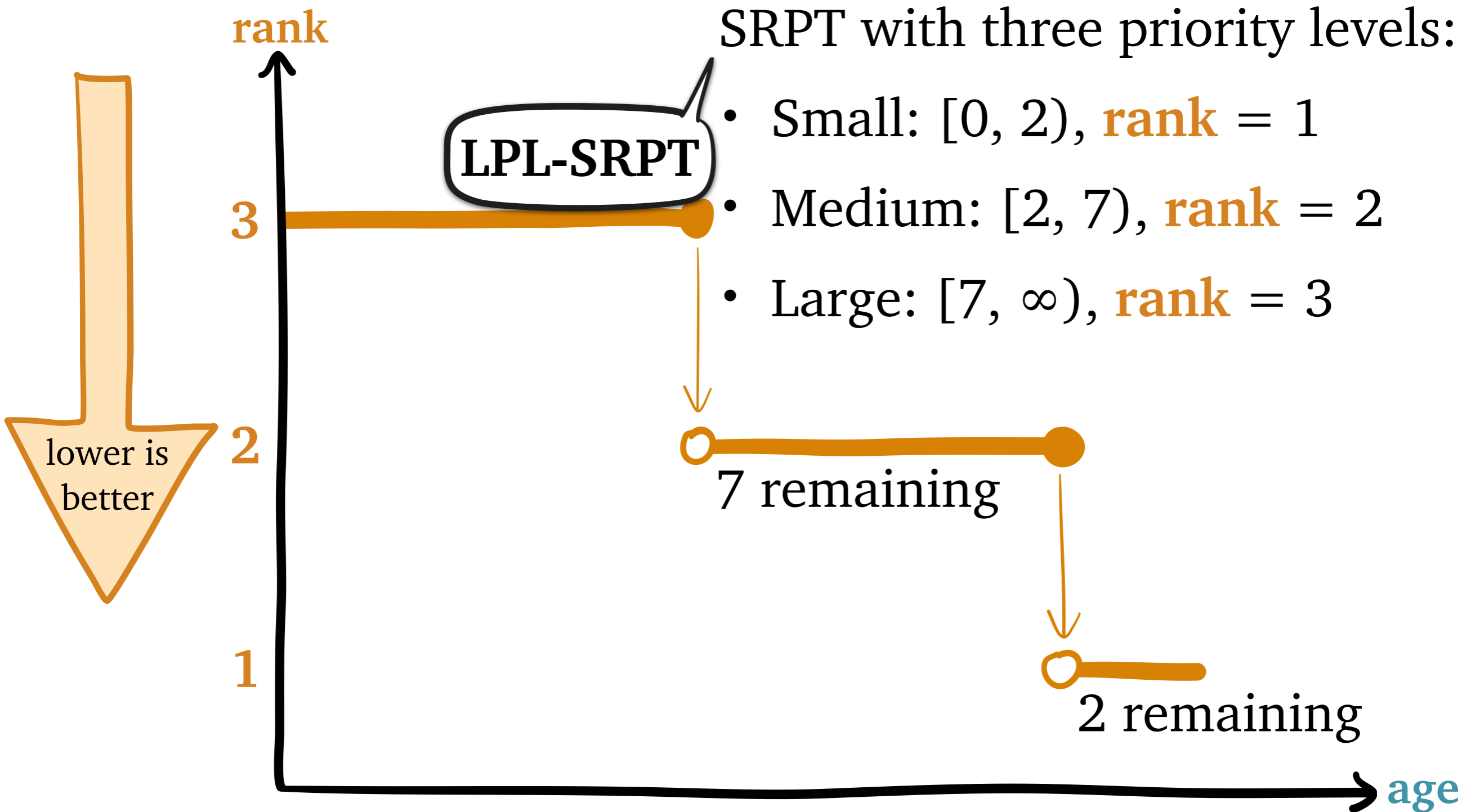
- Small: $[0, 2)$, **rank** = 1
- Medium: $[2, 7)$, **rank** = 2
- Large: $[7, \infty)$, **rank** = 3



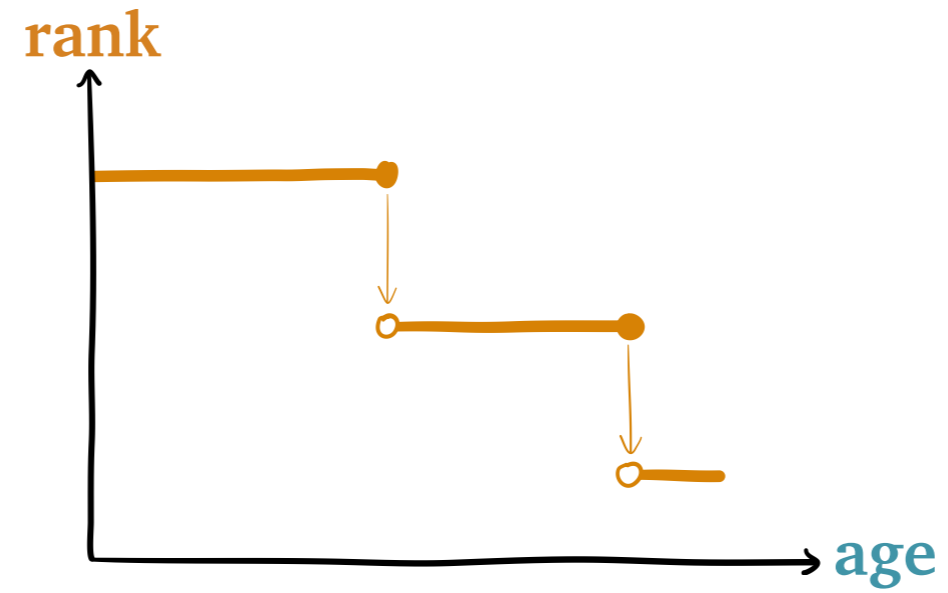
Limited Priority Levels



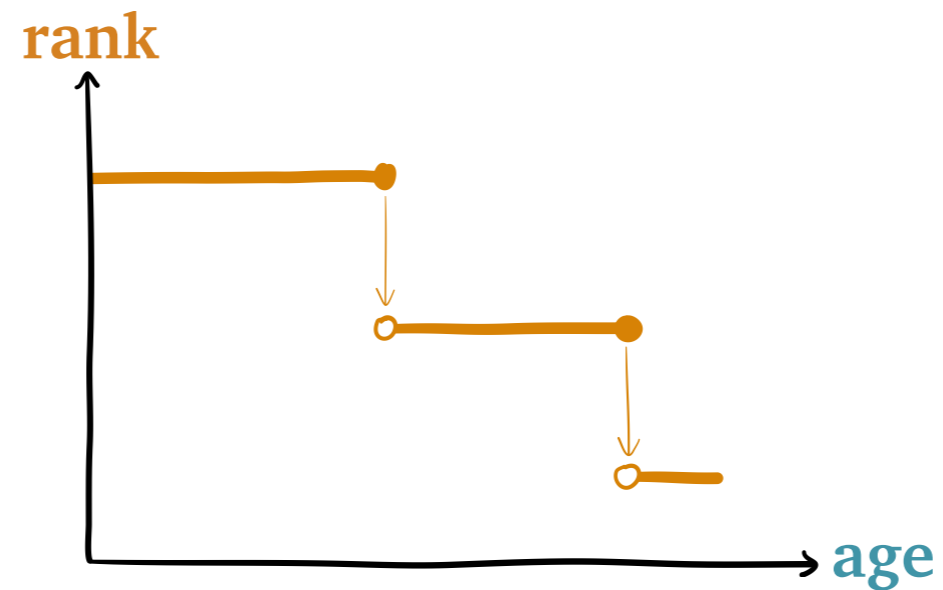
Limited Priority Levels



LPL-SRPT Questions

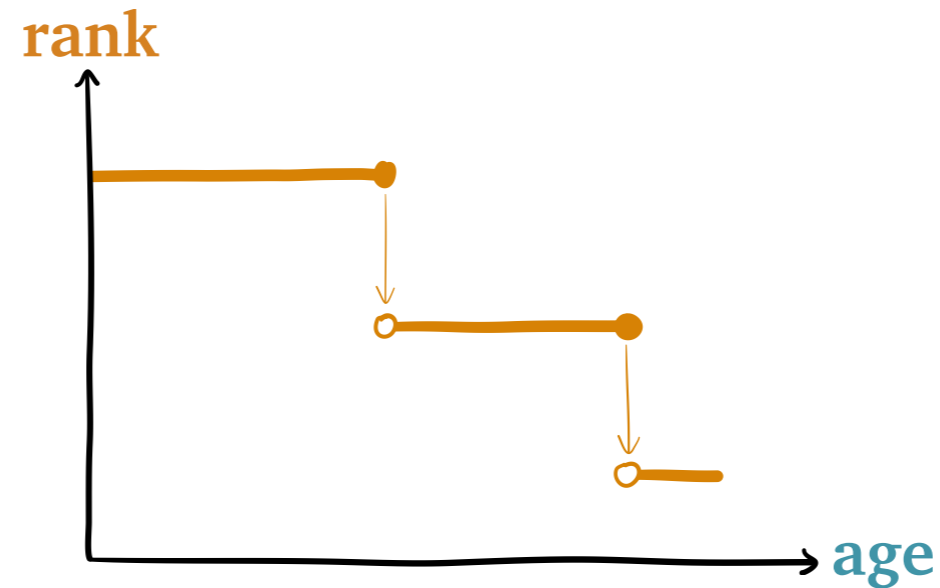


LPL-SRPT Questions



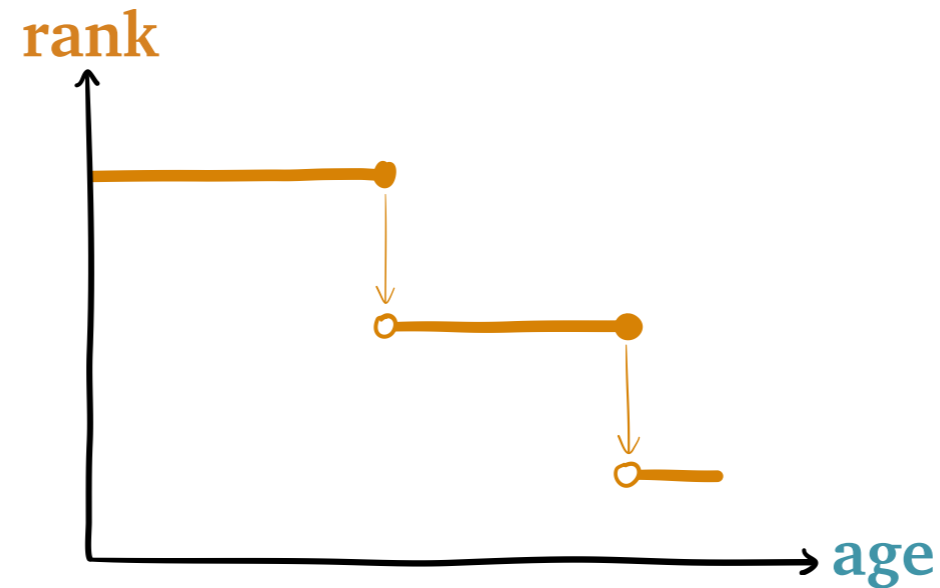
- How many levels do we need?

LPL-SRPT Questions



- How many levels do we need?
- How do we choose size cutoffs?

LPL-SRPT Questions



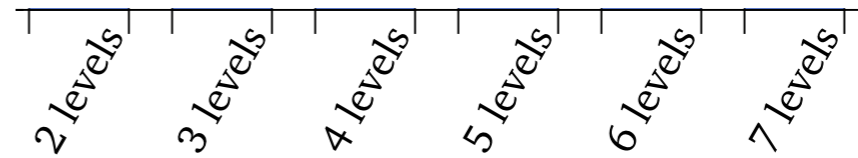
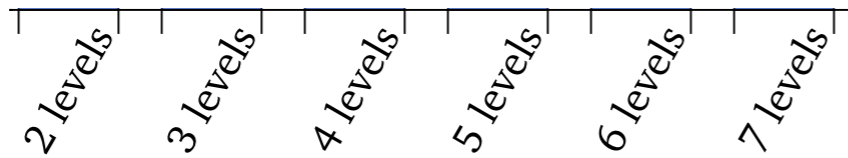
- How many levels do we need?
- How do we choose size cutoffs?
- Can we do better than LPL-SRPT?

How Many Levels?

How Many Levels?

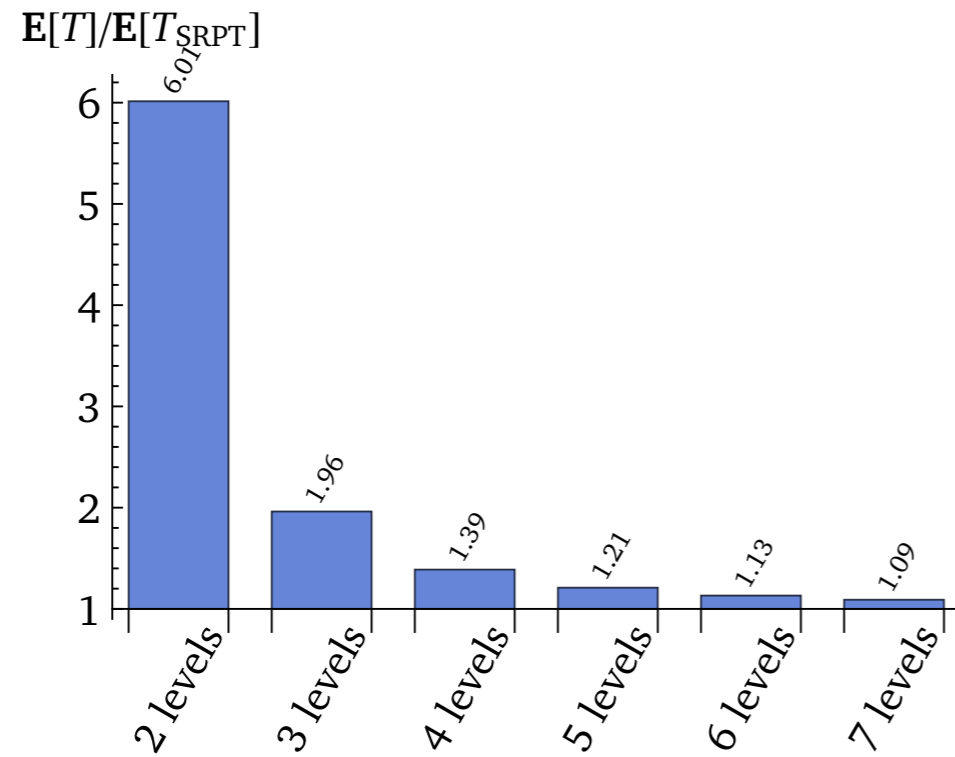
BOUNDED PARETO, $\rho = 0.8$

WEIBULL, $\rho = 0.8$

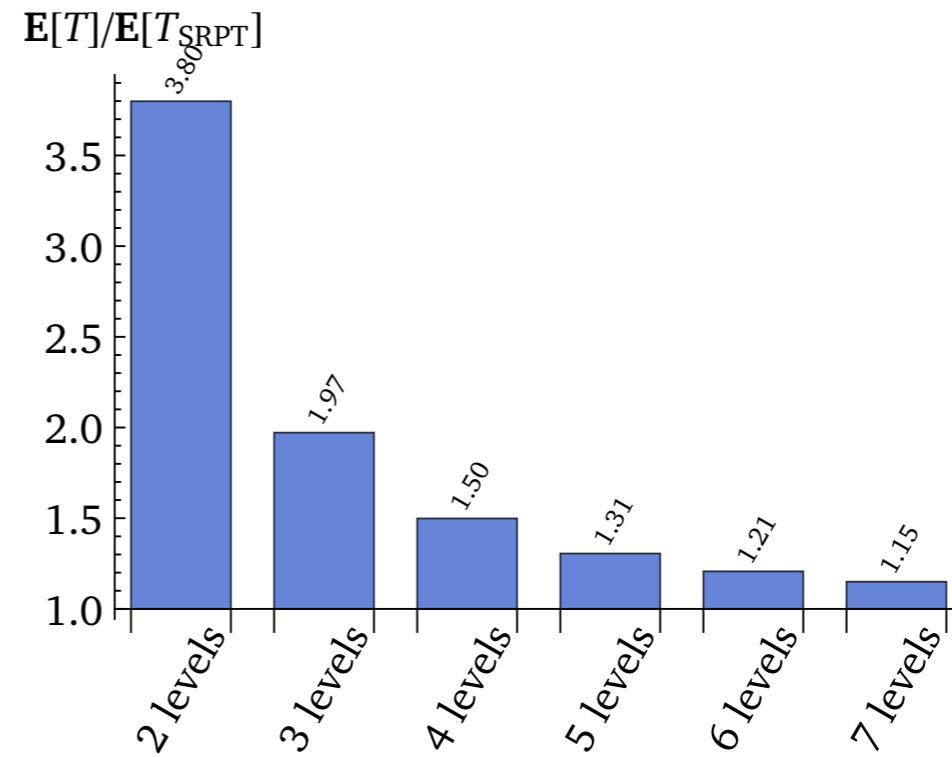


How Many Levels?

BOUNDED PARETO, $\rho = 0.8$

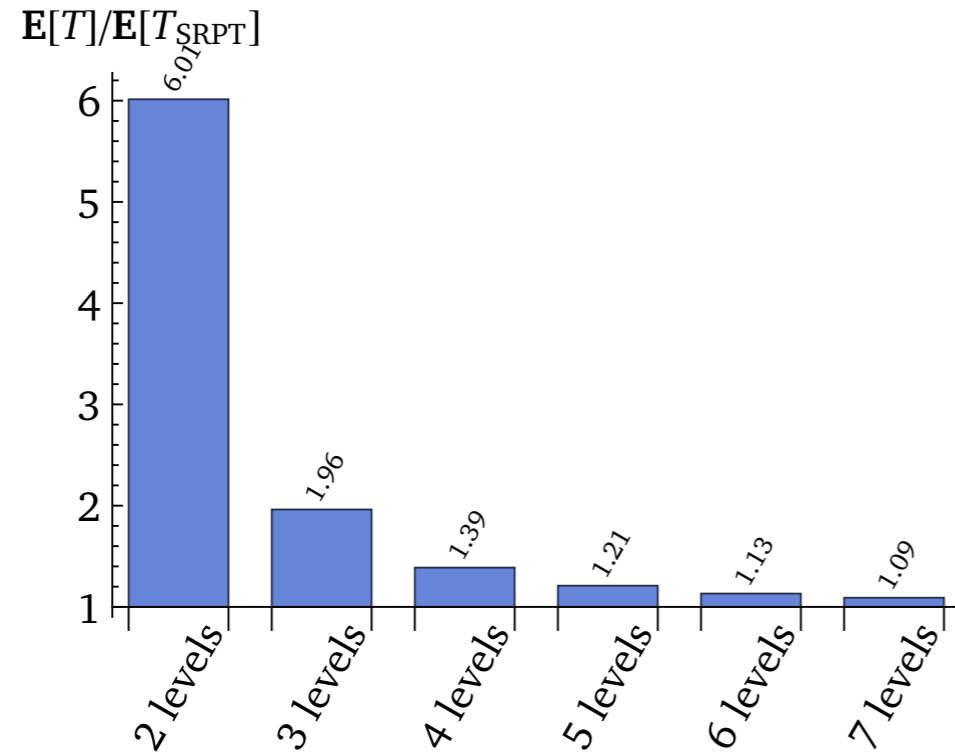


WEIBULL, $\rho = 0.8$

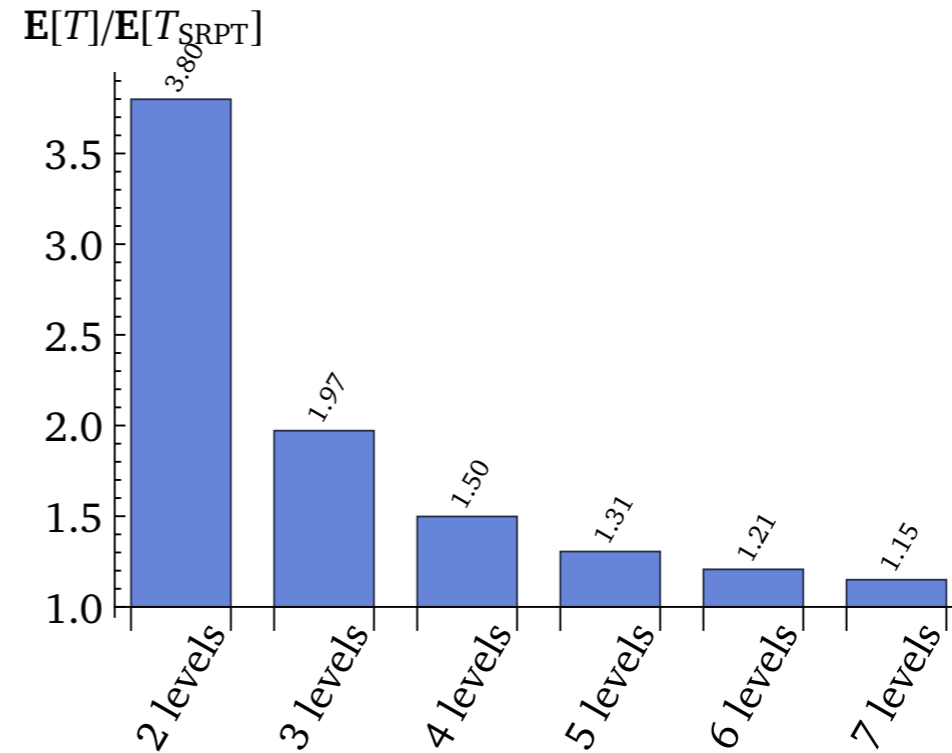


How Many Levels?

BOUNDED PARETO, $\rho = 0.8$



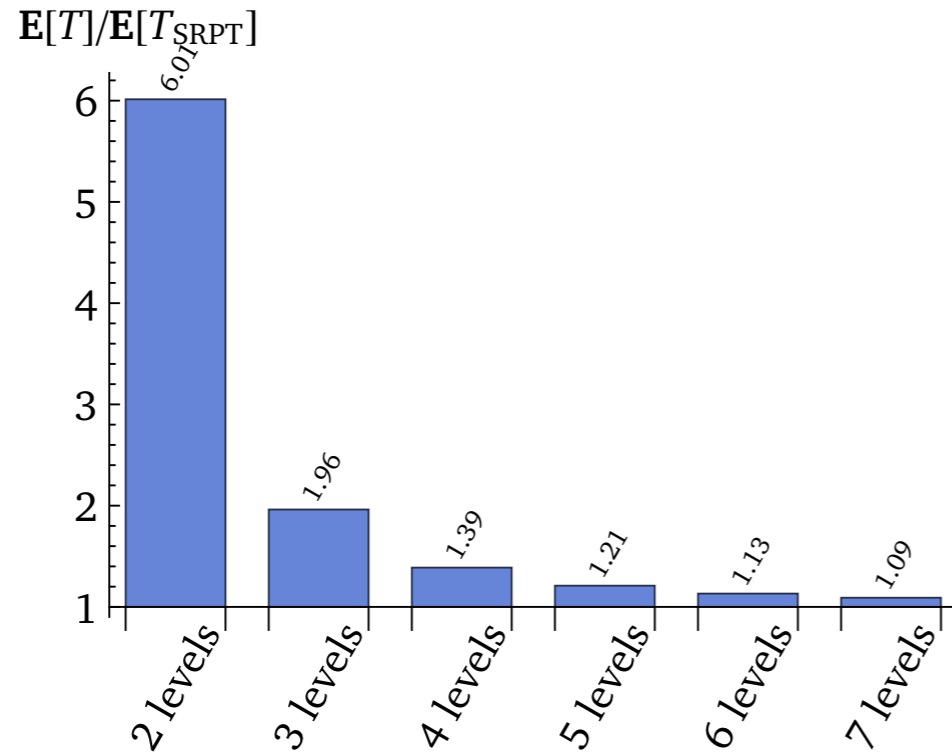
WEIBULL, $\rho = 0.8$



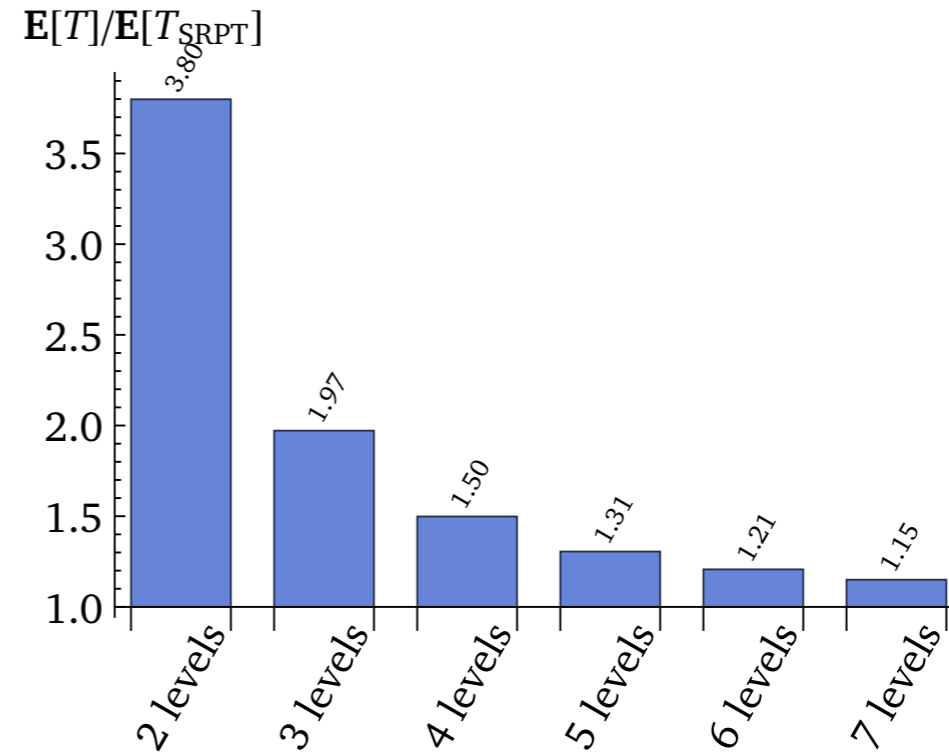
High variance: 5-ish levels suffice

How Many Levels?

BOUNDED PARETO, $\rho = 0.8$



WEIBULL, $\rho = 0.8$



High variance: 5-ish levels suffice

Low variance: 2-ish levels suffice

How to Choose Cutoffs?

How to Choose Cutoffs?

BOUNDED PARETO, $\rho = 0.8$

WEIBULL, $\rho = 0.8$



How to Choose Cutoffs?

BOUNDED PARETO, $\rho = 0.8$

WEIBULL, $\rho = 0.8$

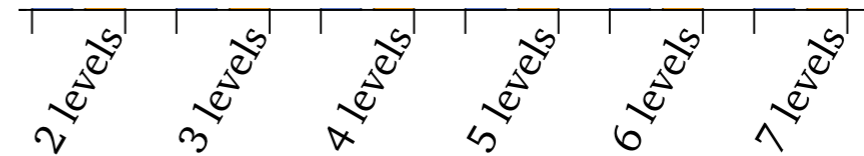
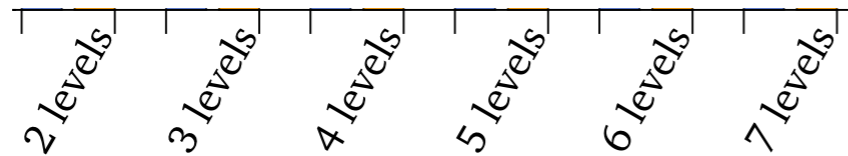


balance load arriving to each size bucket

How to Choose Cutoffs?

BOUNDED PARETO, $\rho = 0.8$

WEIBULL, $\rho = 0.8$



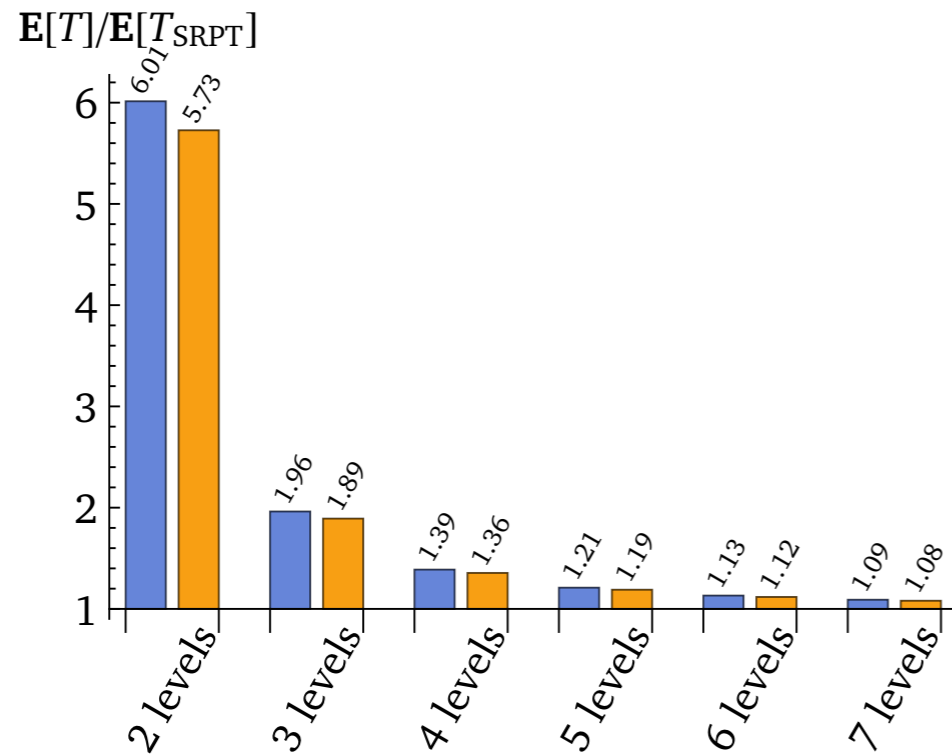
■ LPL-SRPT, heuristic cutoffs ■ LPL-SRPT, optimal cutoffs

balance load arriving to each size bucket

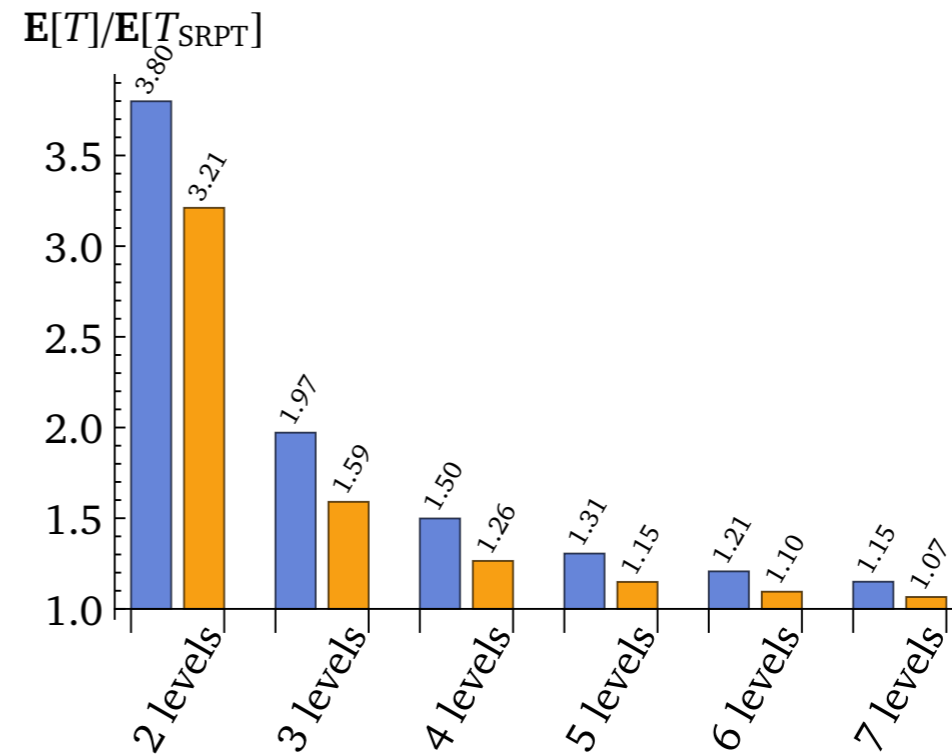
numerical optimization

How to Choose Cutoffs?

BOUNDED PARETO, $\rho = 0.8$



WEIBULL, $\rho = 0.8$



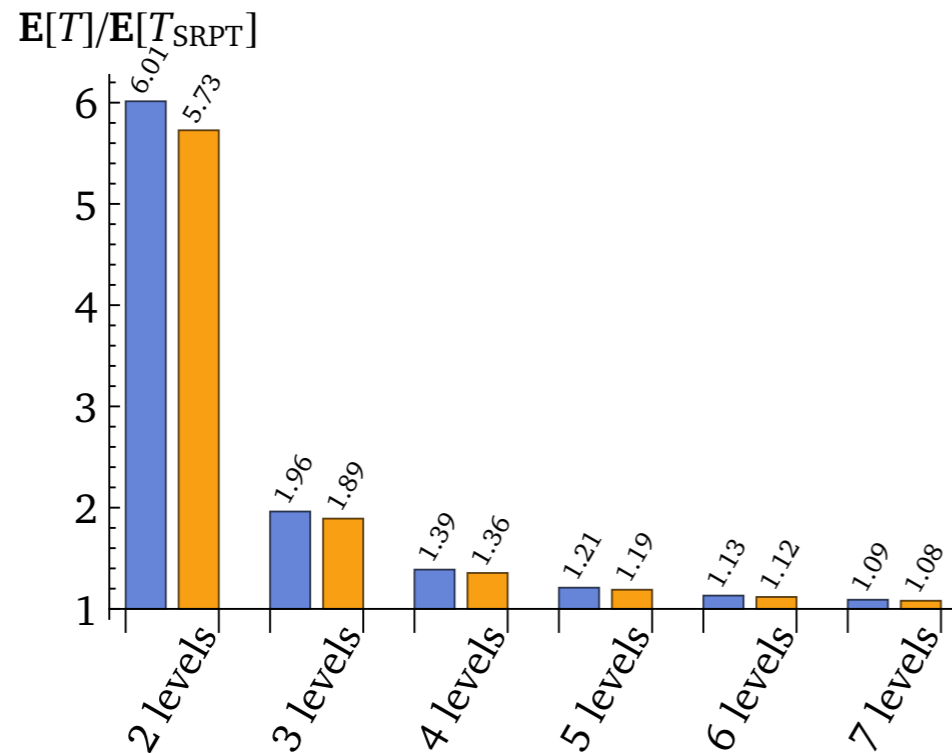
■ LPL-SRPT, heuristic cutoffs ■ LPL-SRPT, optimal cutoffs

balance load arriving to each size bucket

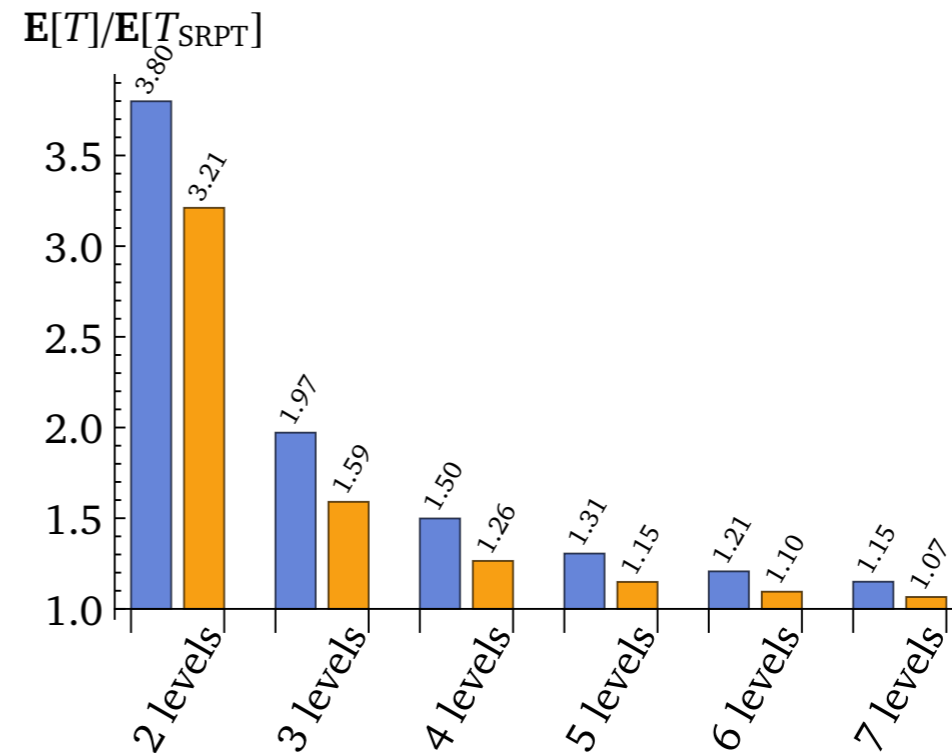
numerical optimization

How to Choose Cutoffs?

BOUNDED PARETO, $\rho = 0.8$



WEIBULL, $\rho = 0.8$



■ LPL-SRPT, heuristic cutoffs ■ LPL-SRPT, optimal cutoffs

balance load arriving to each size bucket

numerical optimization

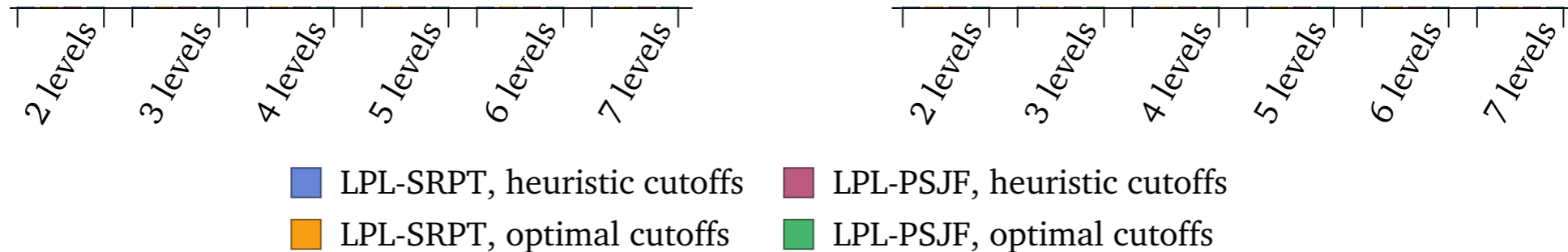
Load-balancing heuristic is pretty good

Can We Improve LPL-SRPT?

Can We Improve LPL-SRPT?

BOUNDED PARETO, $\rho = 0.8$

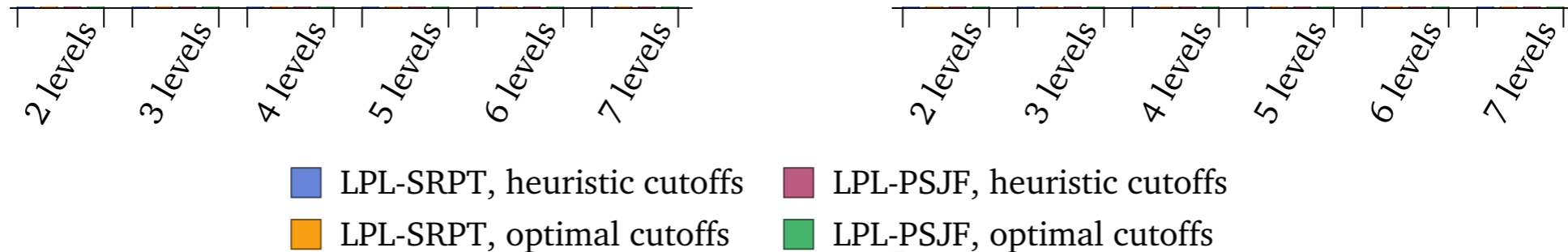
WEIBULL, $\rho = 0.8$



Can We Improve LPL-SRPT?

BOUNDED PARETO, $\rho = 0.8$

WEIBULL, $\rho = 0.8$



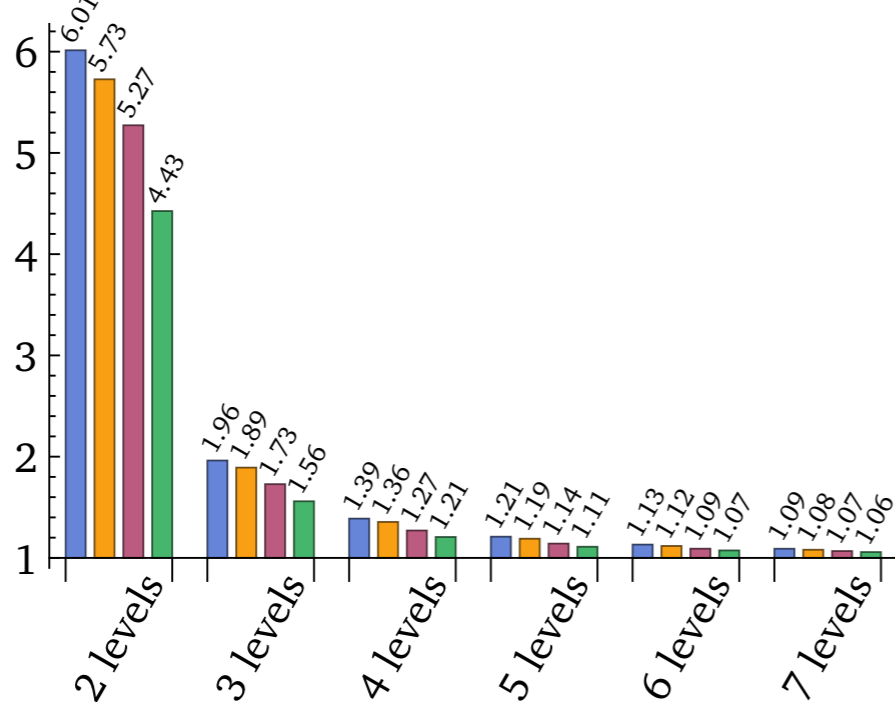
LPL-PSJF: uses *original* size instead of remaining size

Can We Improve LPL-SRPT?

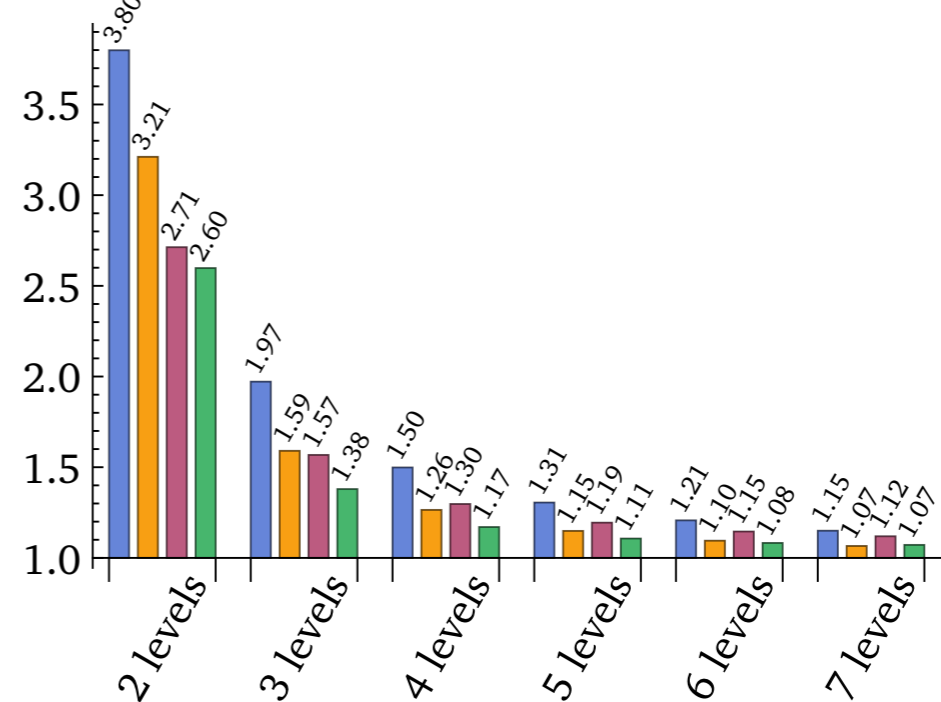
BOUNDED PARETO, $\rho = 0.8$

WEIBULL, $\rho = 0.8$

$E[T]/E[T_{SRPT}]$



$E[T]/E[T_{SRPT}]$



- LPL-SRPT, heuristic cutoffs
- LPL-SRPT, optimal cutoffs
- LPL-PSJF, heuristic cutoffs
- LPL-PSJF, optimal cutoffs

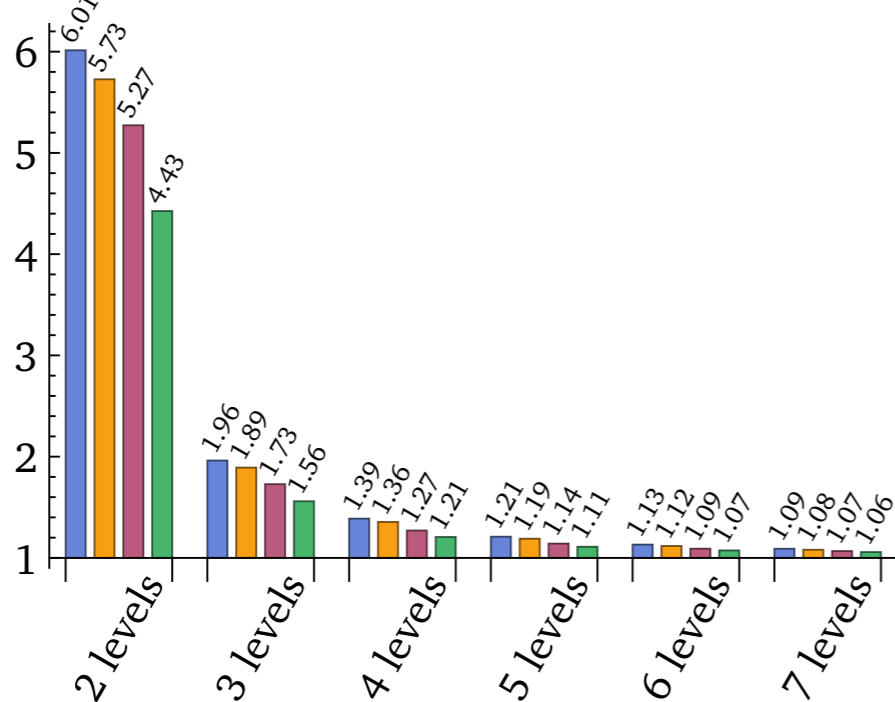
LPL-PSJF: uses *original size* instead of remaining size

Can We Improve LPL-SRPT?

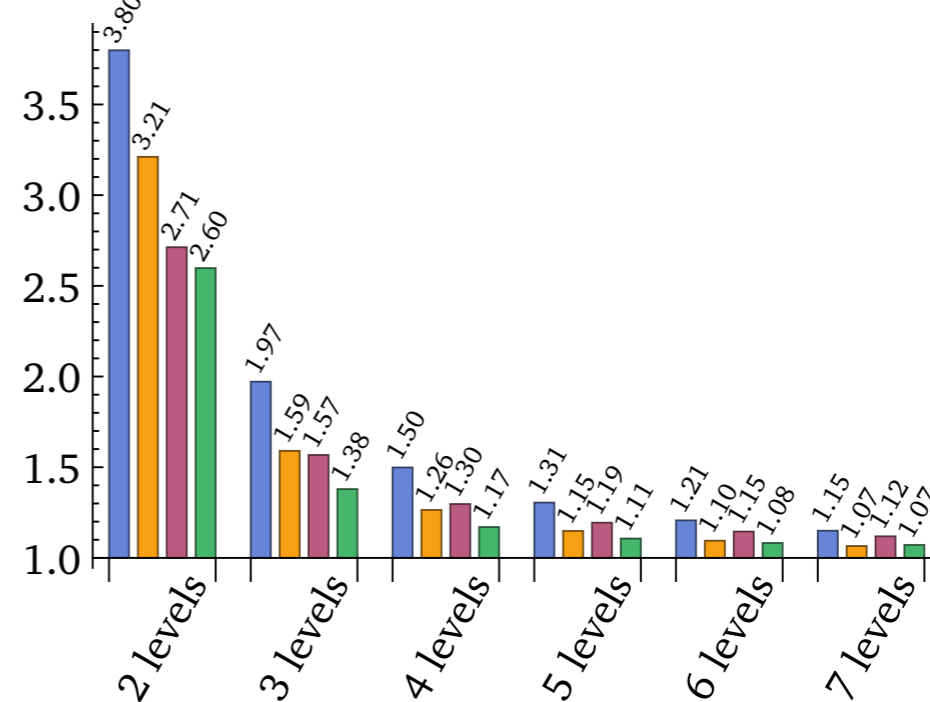
BOUNDED PARETO, $\rho = 0.8$

WEIBULL, $\rho = 0.8$

$E[T]/E[T_{SRPT}]$



$E[T]/E[T_{SRPT}]$

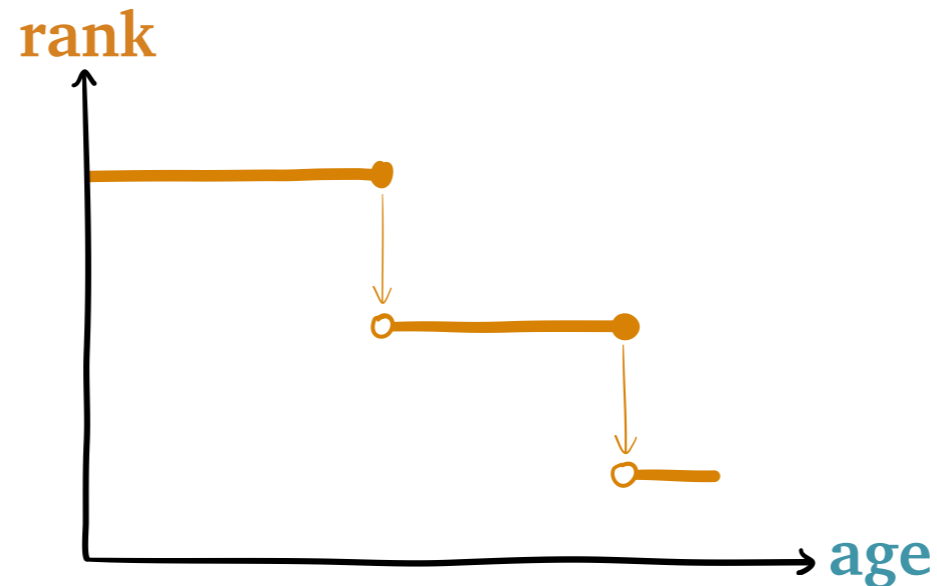


■ LPL-SRPT, heuristic cutoffs ■ LPL-PSJF, heuristic cutoffs
■ LPL-SRPT, optimal cutoffs ■ LPL-PSJF, optimal cutoffs

LPL-PSJF: uses *original size* instead of remaining size

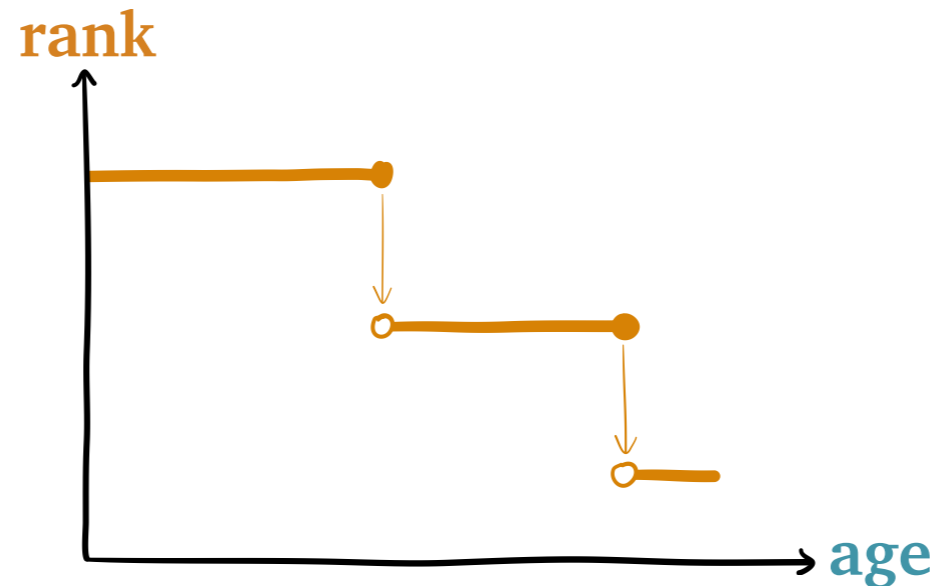
LPL-PSJF beats LPL-SRPT!

LPL-SRPT Questions



- How many levels do we need?
- How do we choose size cutoffs?
- Can we do better than LPL-SRPT?

LPL-SRPT Questions

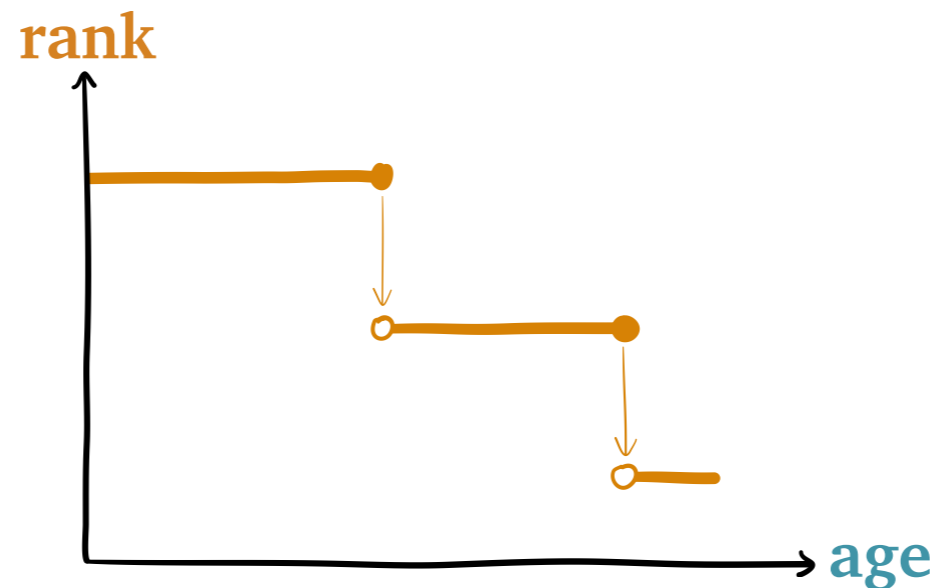


High var: 5-ish

Low var: 2-ish

- How many levels do we need?
- How do we choose size cutoffs?
- Can we do better than LPL-SRPT?

LPL-SRPT Questions



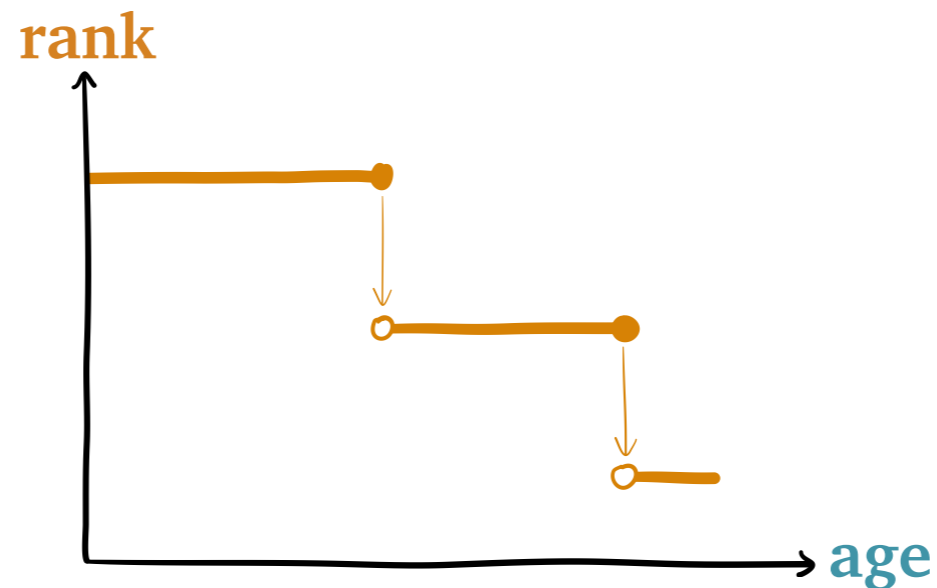
High var: 5-ish

Low var: 2-ish

Balancing load is a good heuristic

- How many levels do we need?
- How do we choose size cutoffs?
- Can we do better than LPL-SRPT?

LPL-SRPT Questions



High var: 5-ish

Low var: 2-ish

Balancing load is a good heuristic

- How many levels do we need?
- How do we choose size cutoffs?
- Can we do better than LPL-SRPT?

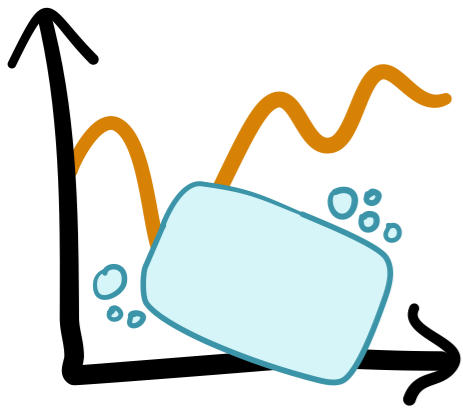
Yes! LPL-PSJF often better

Overview

New Tools

SOAP

*analyzes a huge variety
of scheduling heuristics*



Goals

*Simple implementation
preferred*

*Preemption restricted
and/or costly*

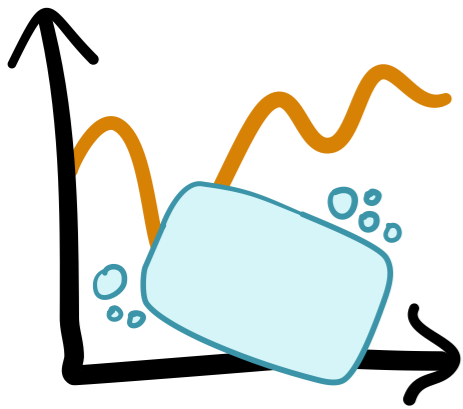
*Limited number
of priority levels*

Overview

New Tools

SOAP

*analyzes a huge variety
of scheduling heuristics*



Goals

*Simple implementation
preferred*

*Preemption restricted
and/or costly*



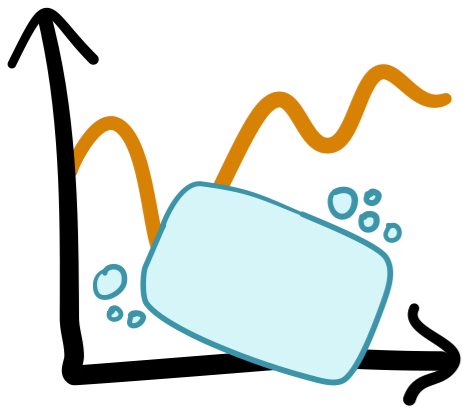
*Limited number
of priority levels*

Overview

New Tools

SOAP

*analyzes a huge variety
of scheduling heuristics*



Goals

*Simple implementation
preferred*

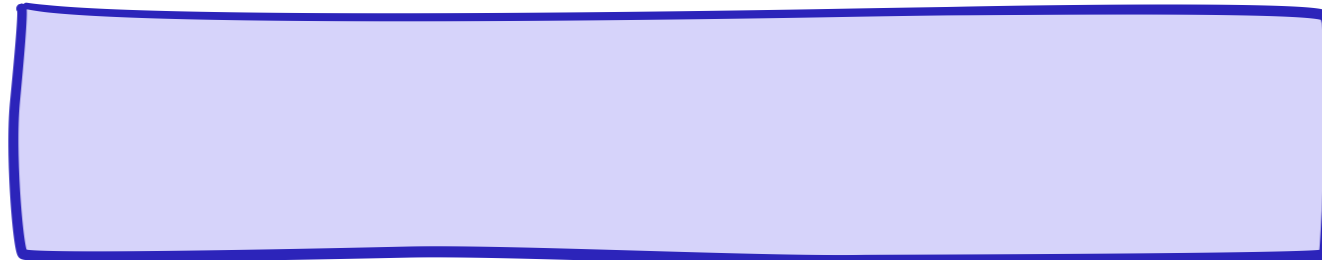
*Preemption restricted
and/or costly*



*Limited number
of priority levels*

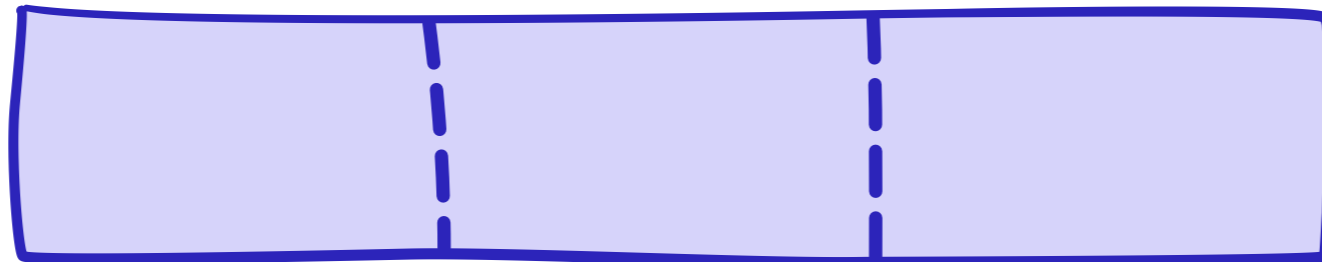
Scheduling Packet Flows

Job is sequence of *packets*



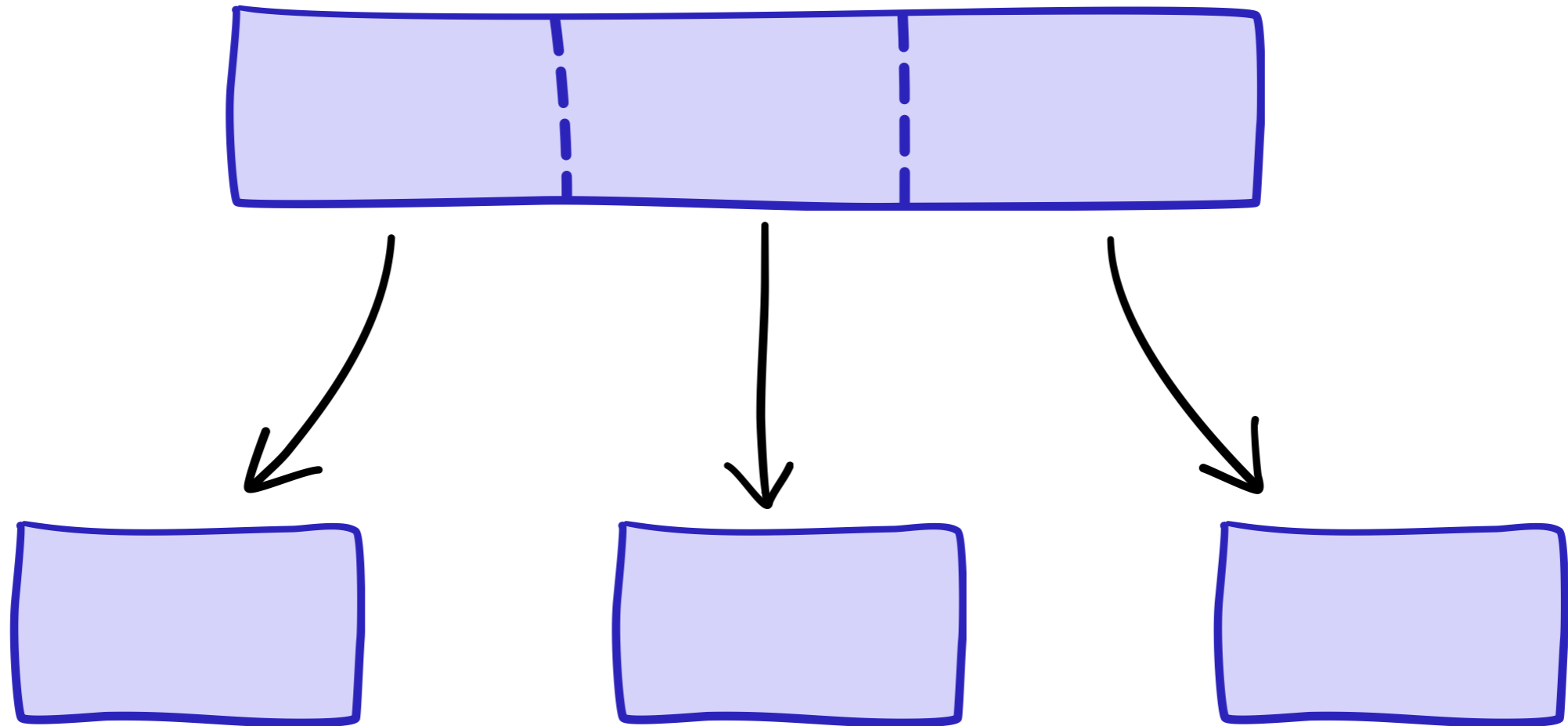
Scheduling Packet Flows

Job is sequence of *packets*



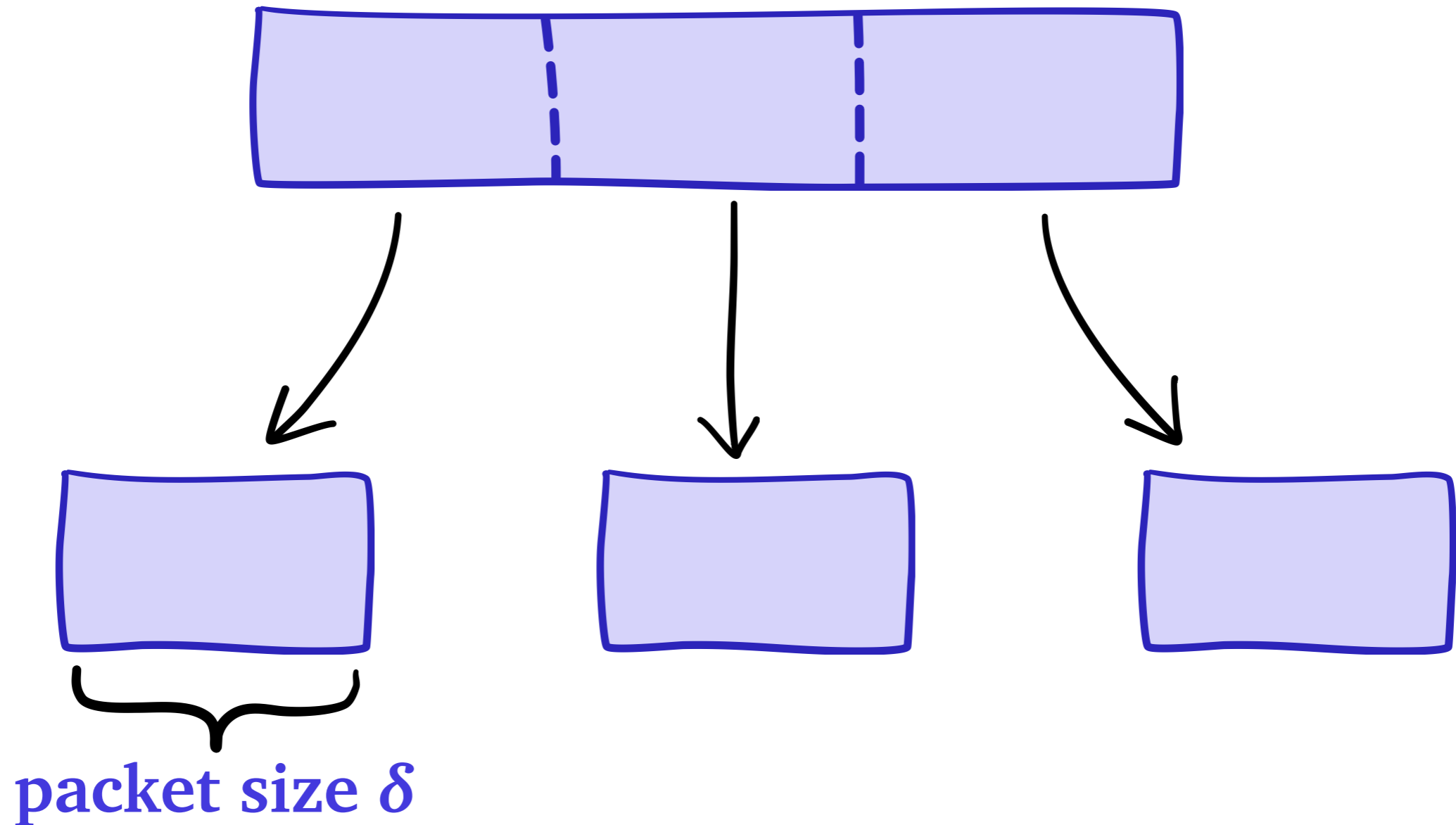
Scheduling Packet Flows

Job is sequence of *packets*

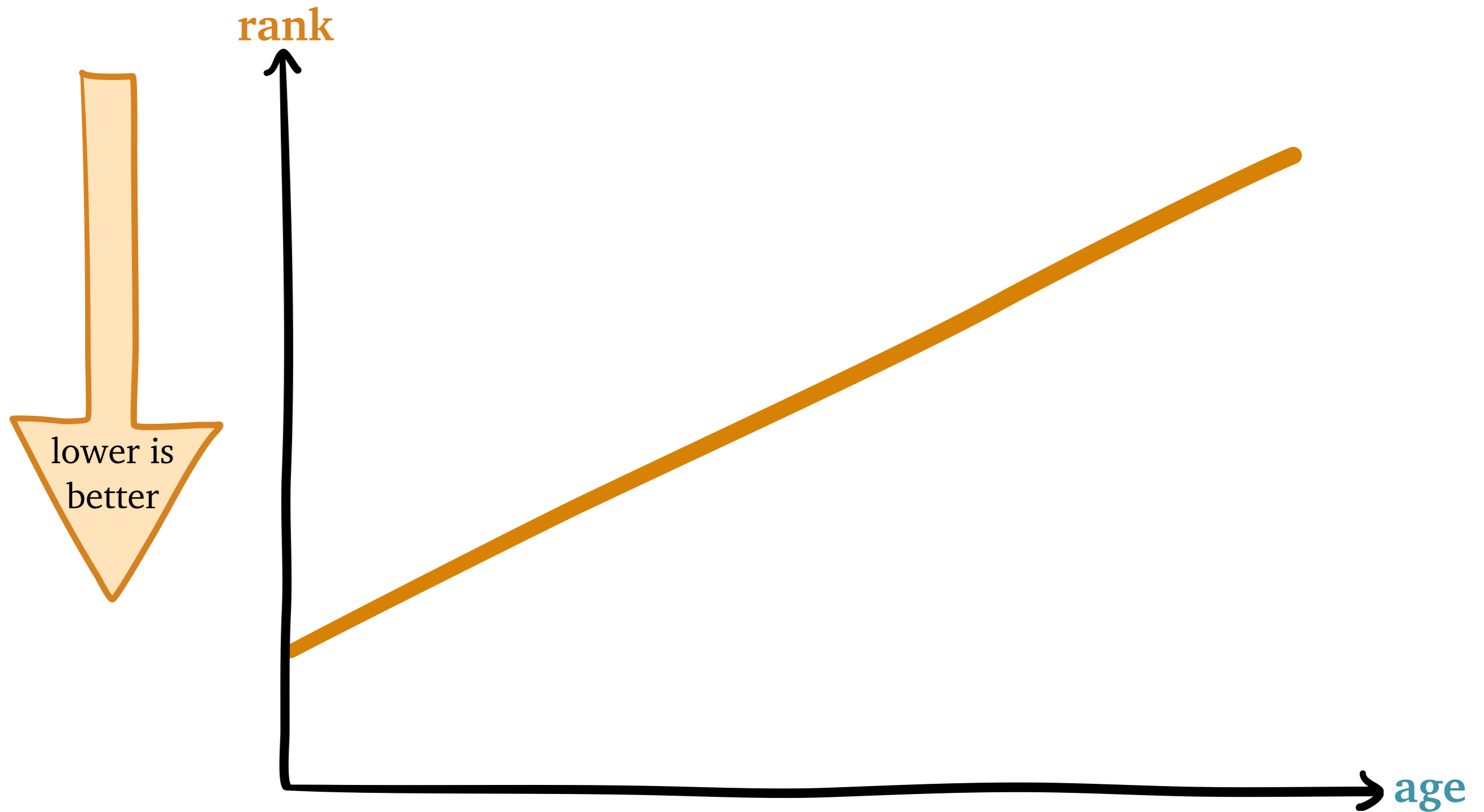


Scheduling Packet Flows

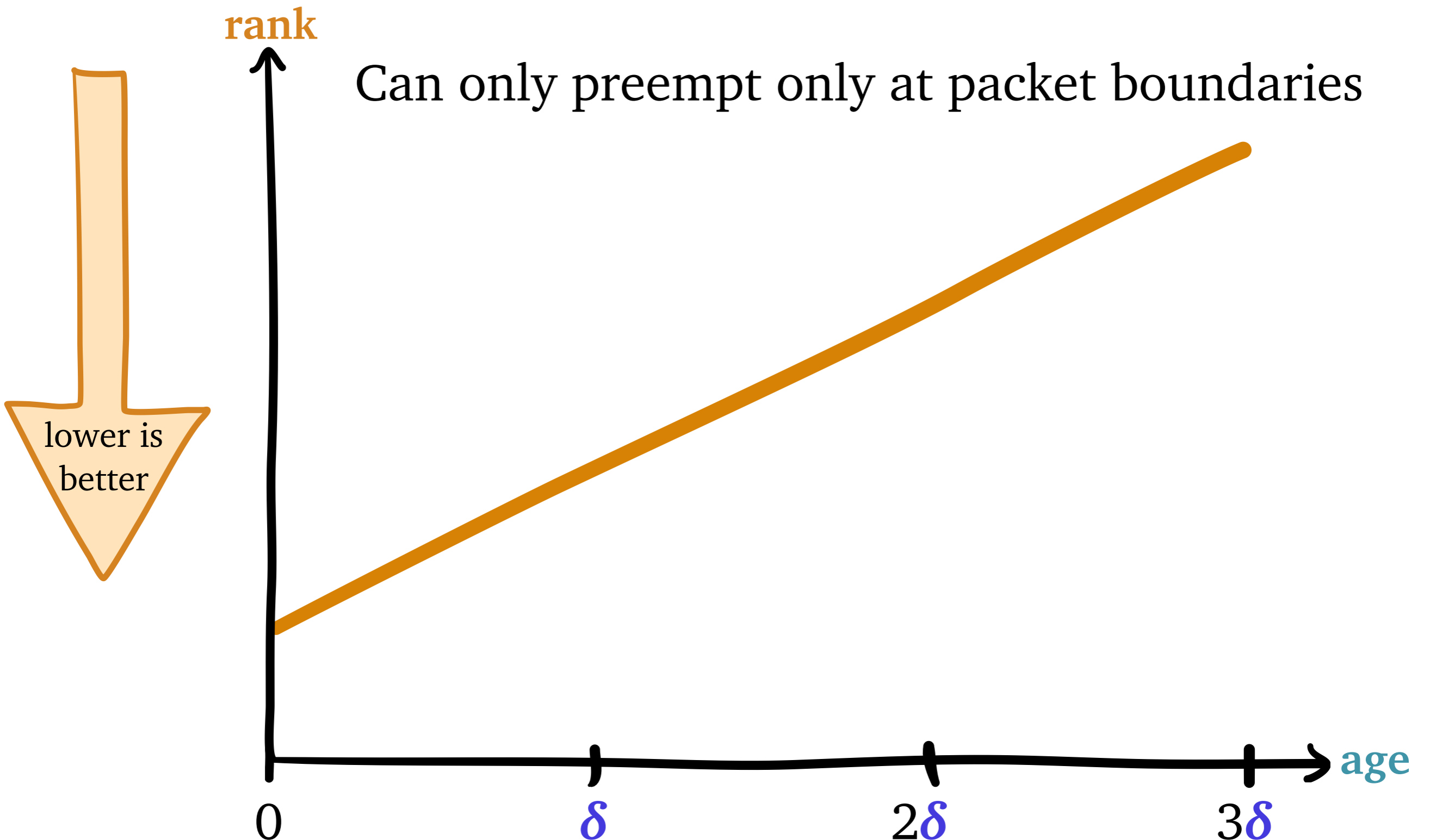
Job is sequence of *packets*



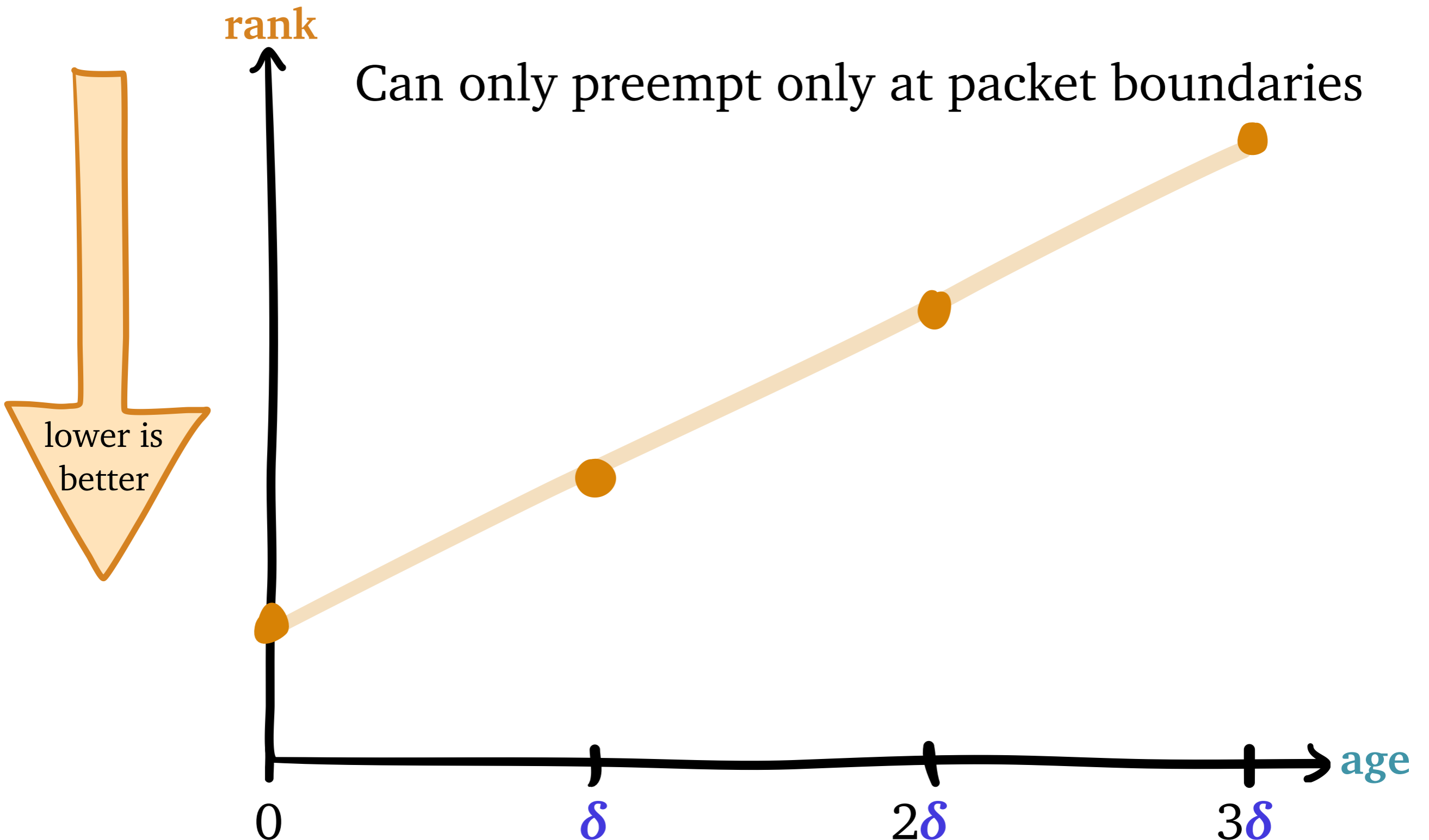
Packets as a Rank Function



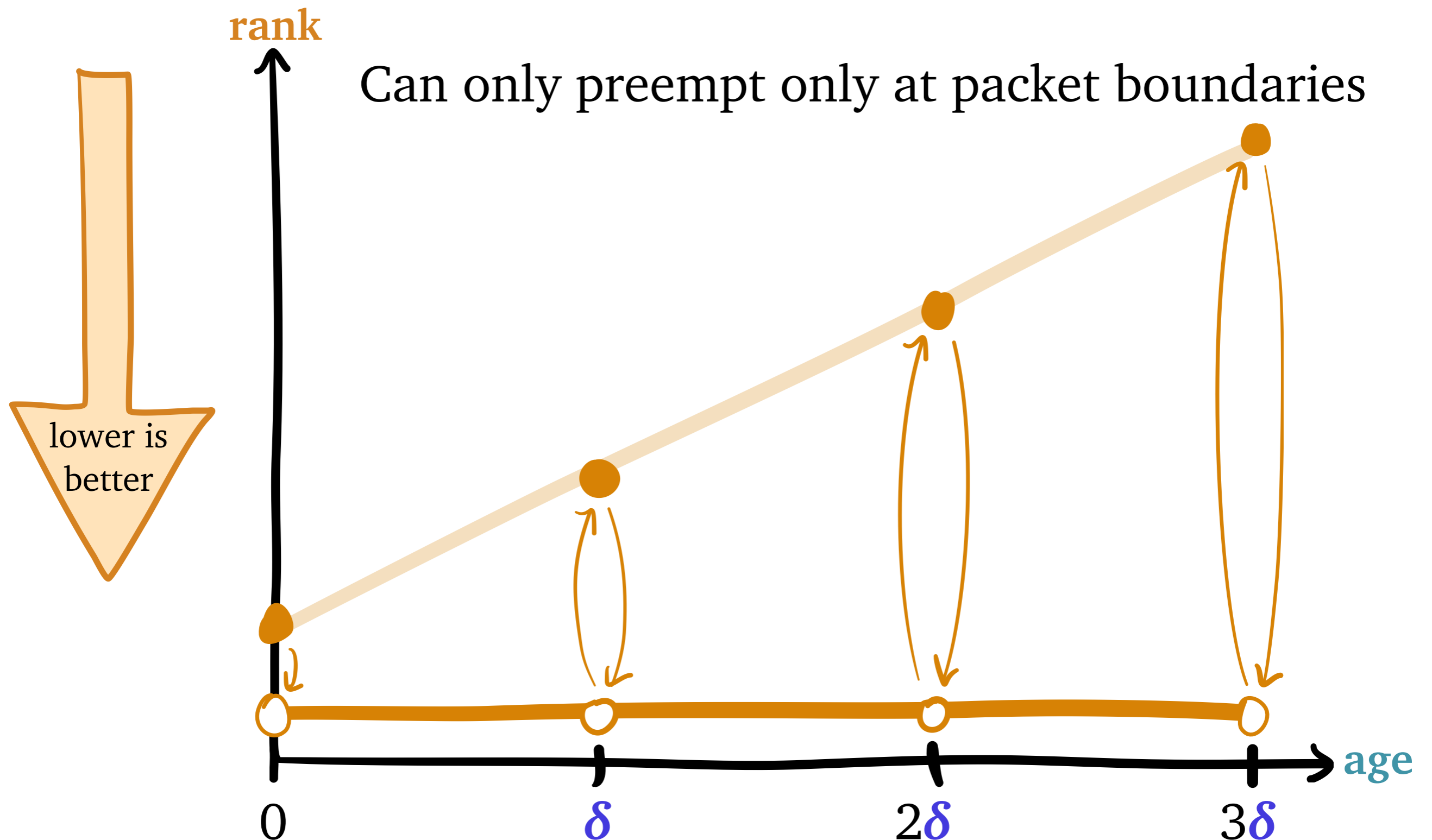
Packets as a Rank Function



Packets as a Rank Function

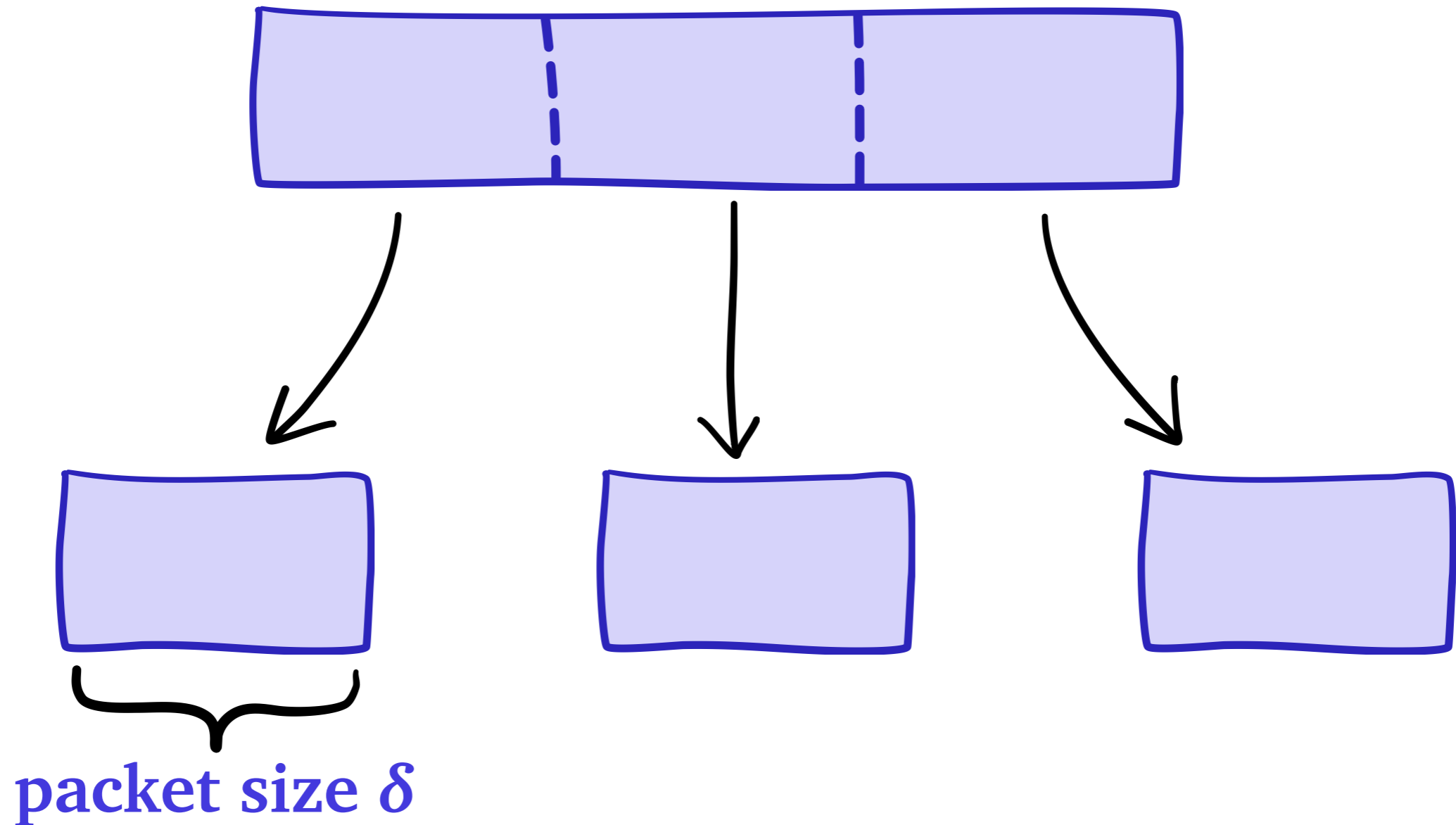


Packets as a Rank Function



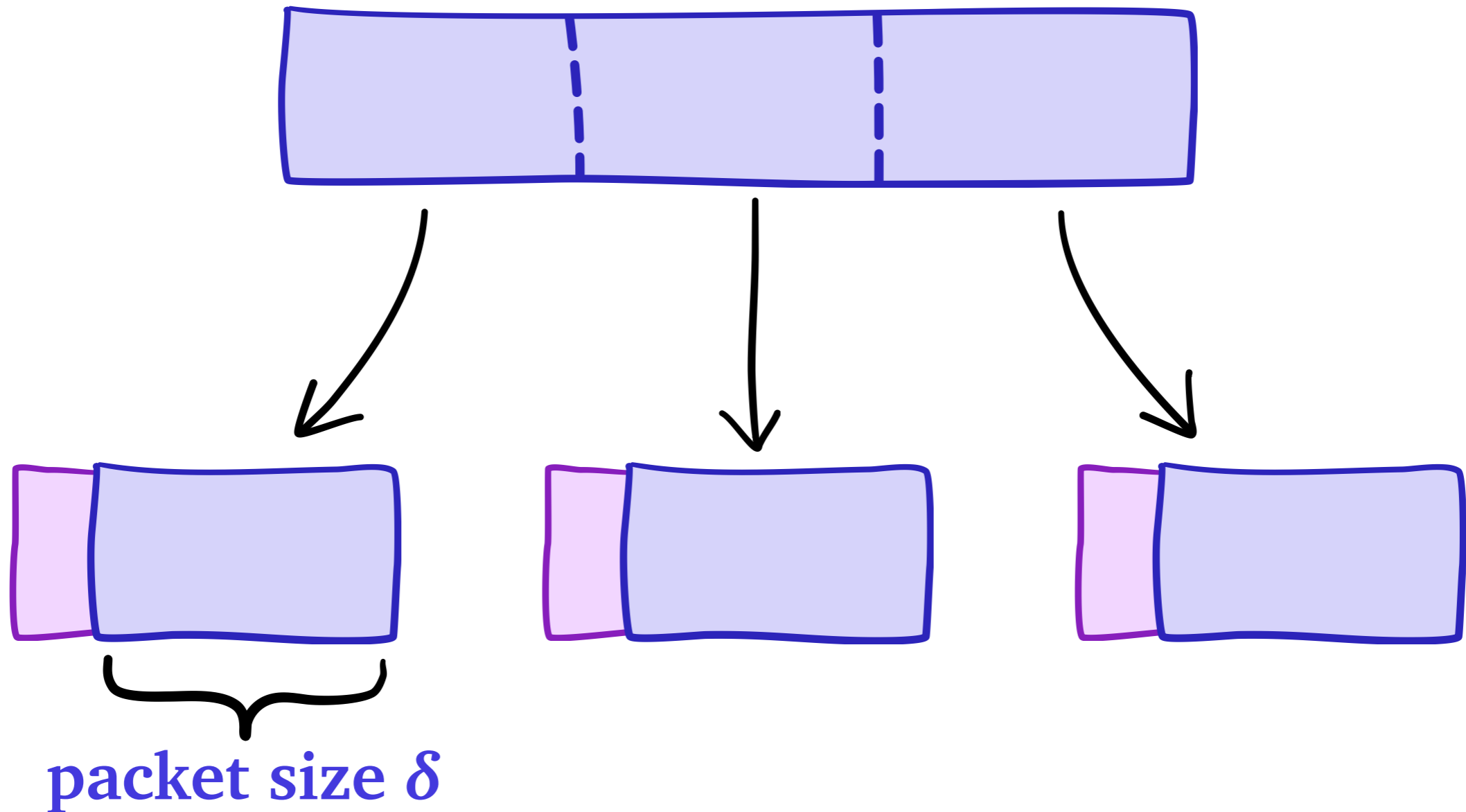
Scheduling Packet Flows

Job is sequence of *packets*



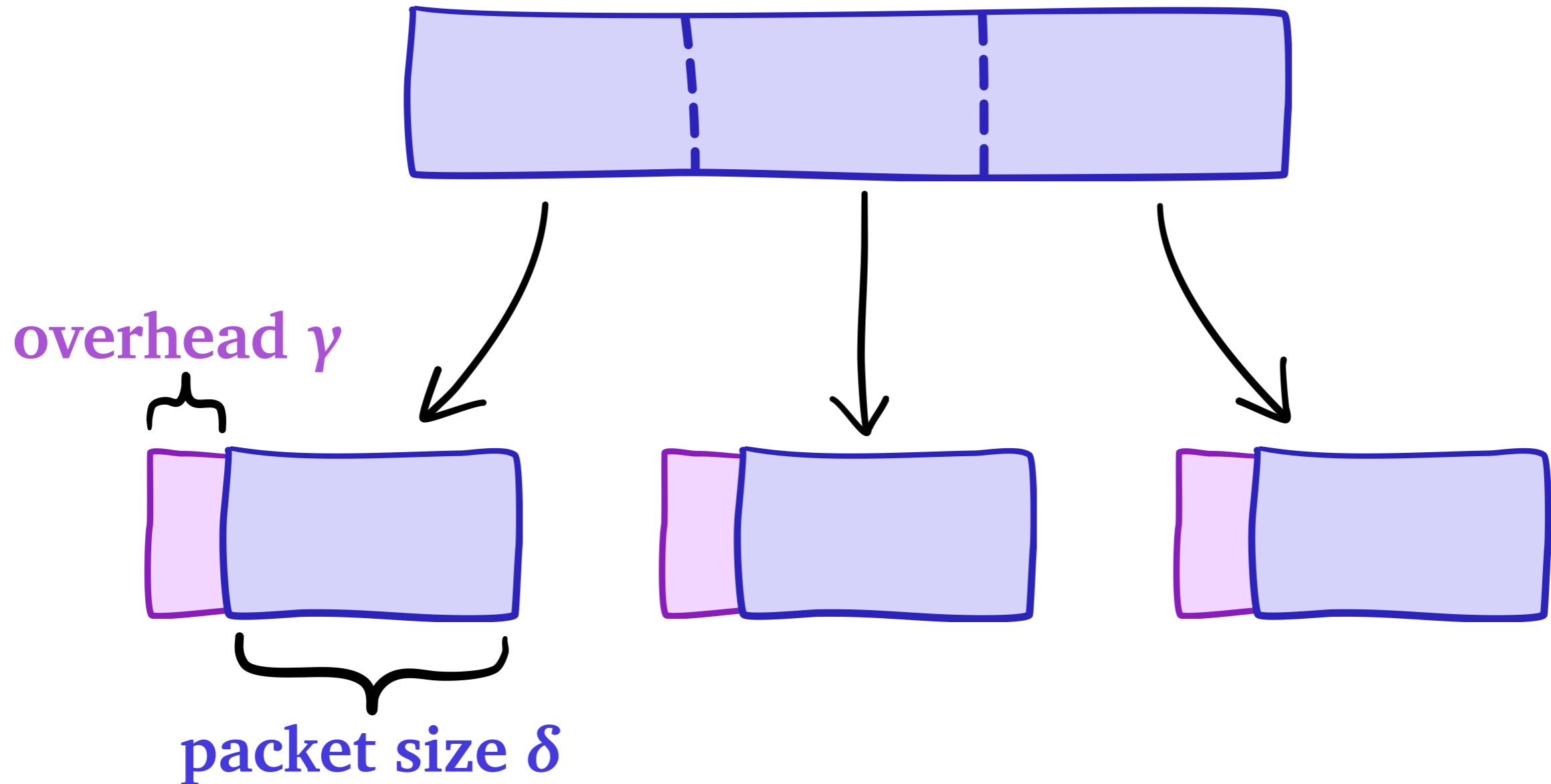
Scheduling Packet Flows

Job is sequence of *packets*



Scheduling Packet Flows

Job is sequence of *packets*



Best Packet Size?

Given job size distribution S , load ρ , and **overhead γ** ,
what is the optimal **packet size δ** ?

- large δ : less overhead

Best Packet Size?

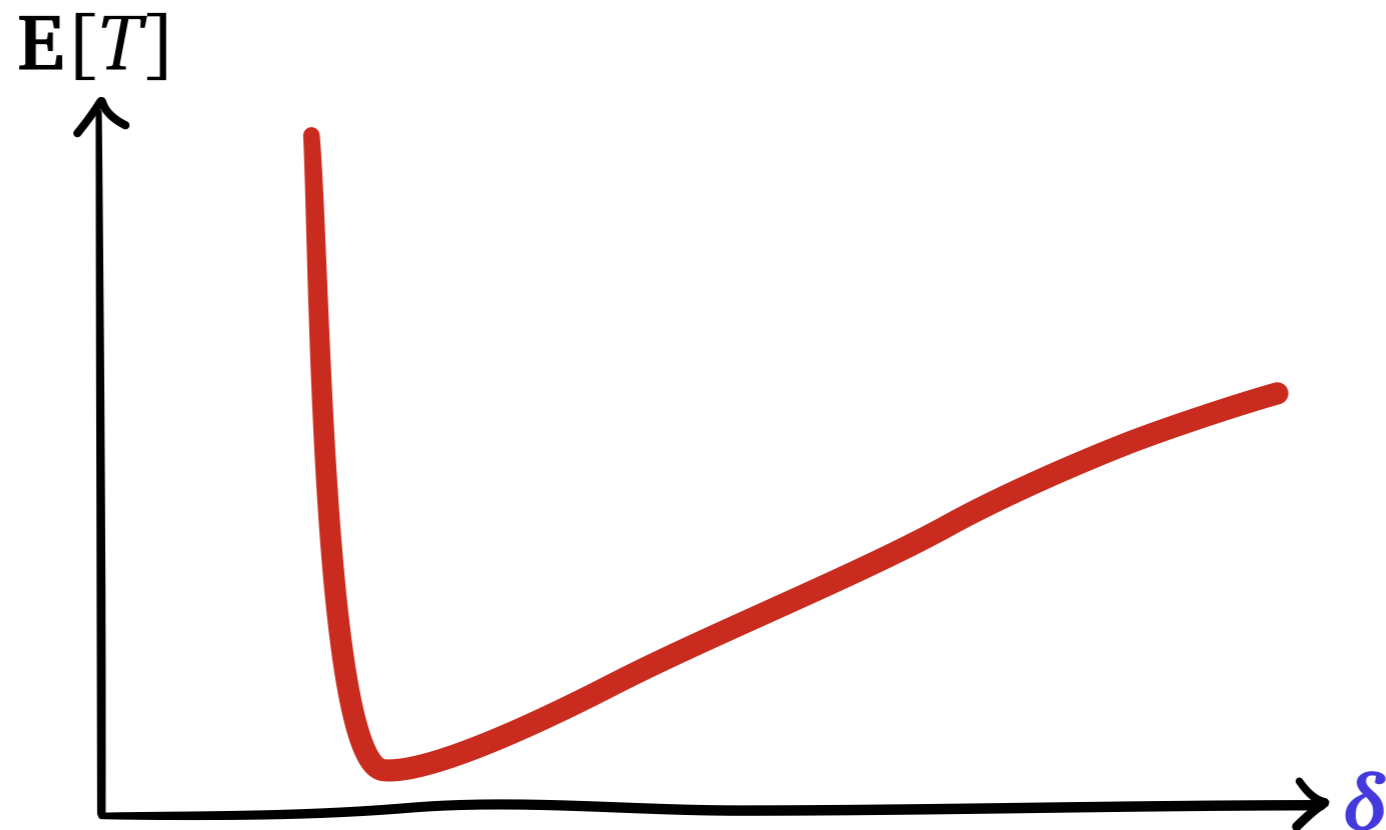
Given job size distribution S , load ρ , and **overhead γ** , what is the optimal **packet size δ** ?

- large δ : less overhead
- small δ : better scheduling

Best Packet Size?

Given job size distribution S , load ρ , and **overhead** γ , what is the optimal **packet size** δ ?

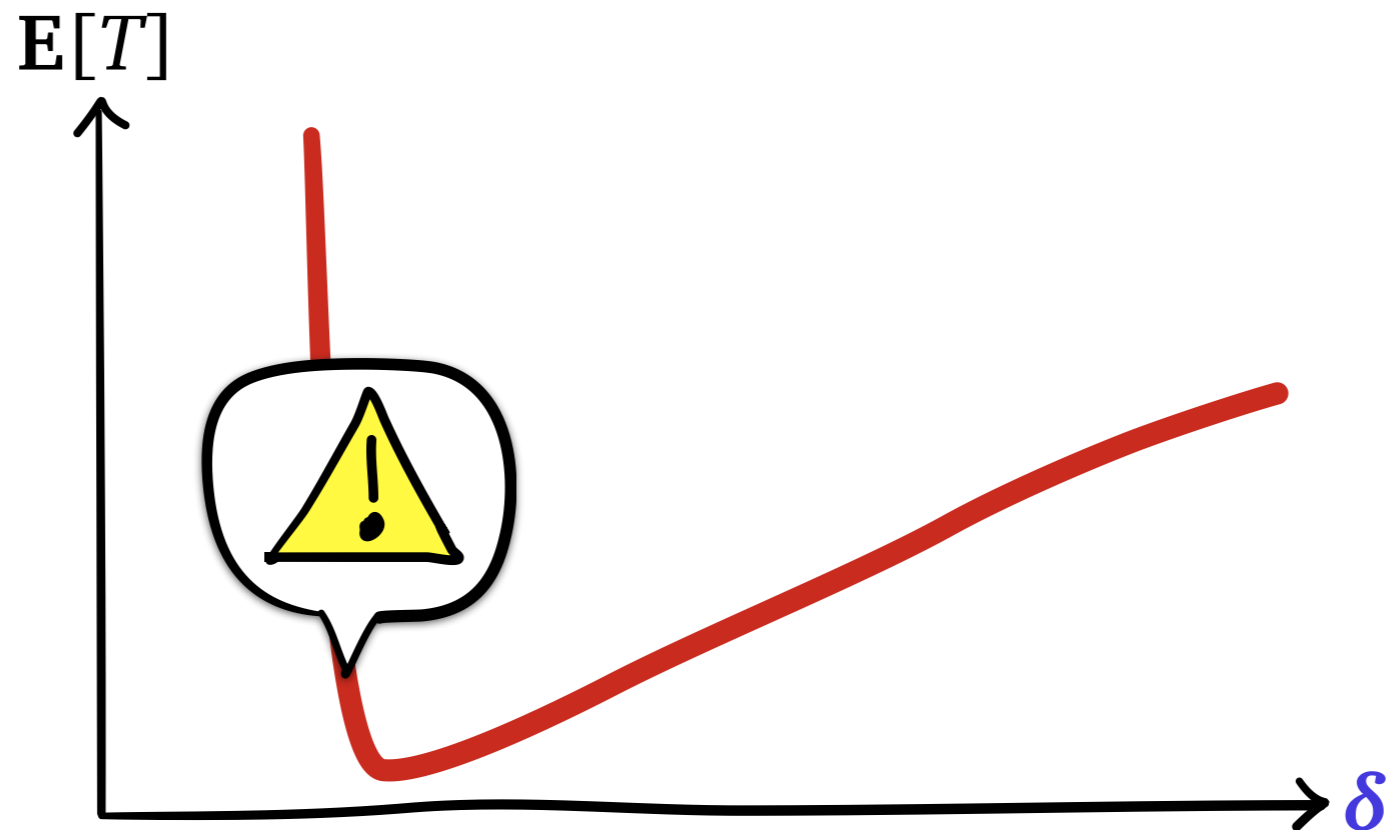
- large δ : less overhead
- small δ : better scheduling



Best Packet Size?

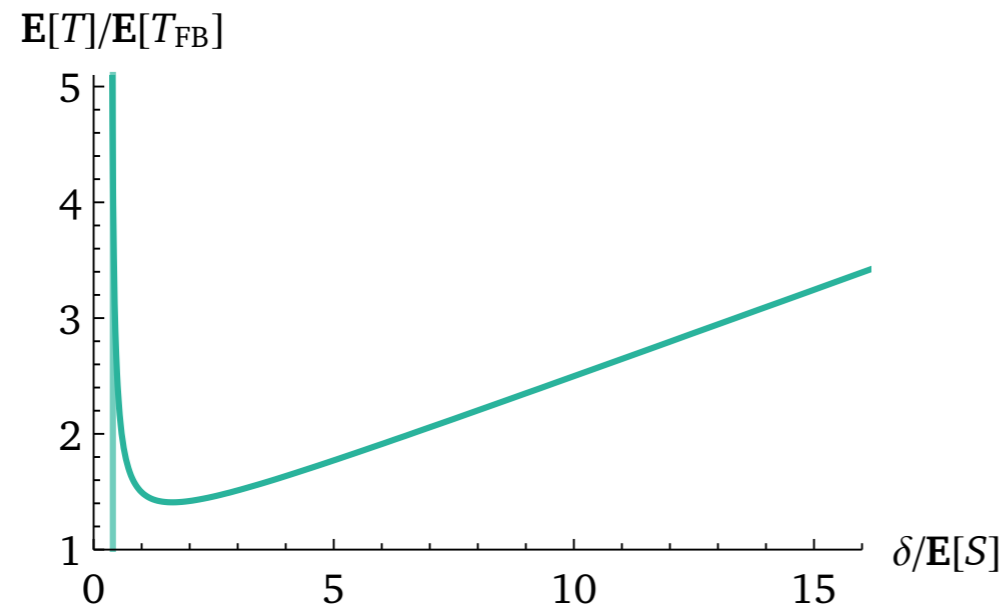
Given job size distribution S , load ρ , and **overhead** γ , what is the optimal **packet size** δ ?

- large δ : less overhead
- small δ : better scheduling

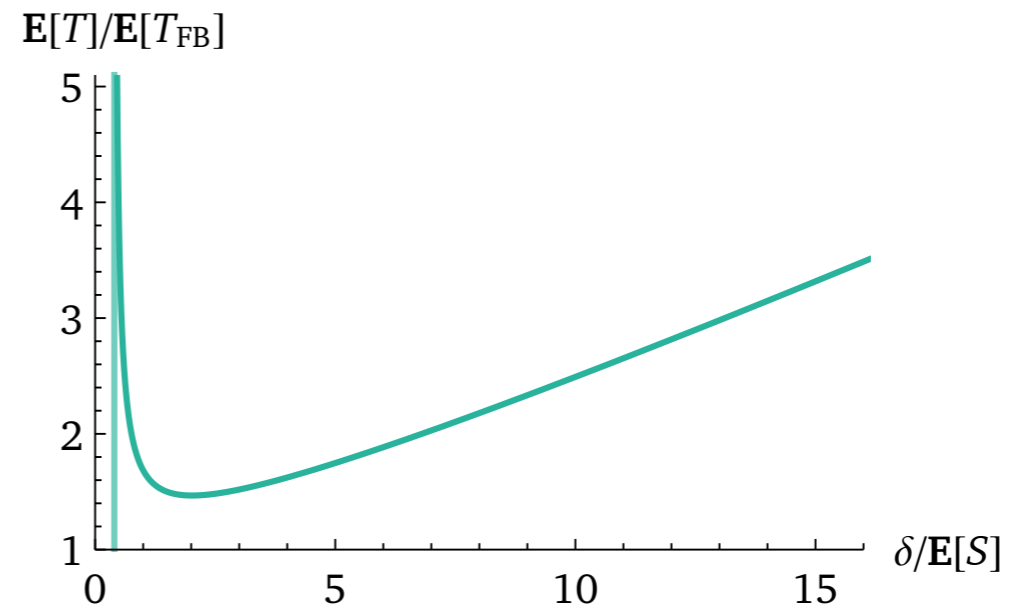


Ensuring Stability

BOUNDED PARETO, $\gamma = 0.1 \mathbf{E}[S]$, $\rho = 0.8$

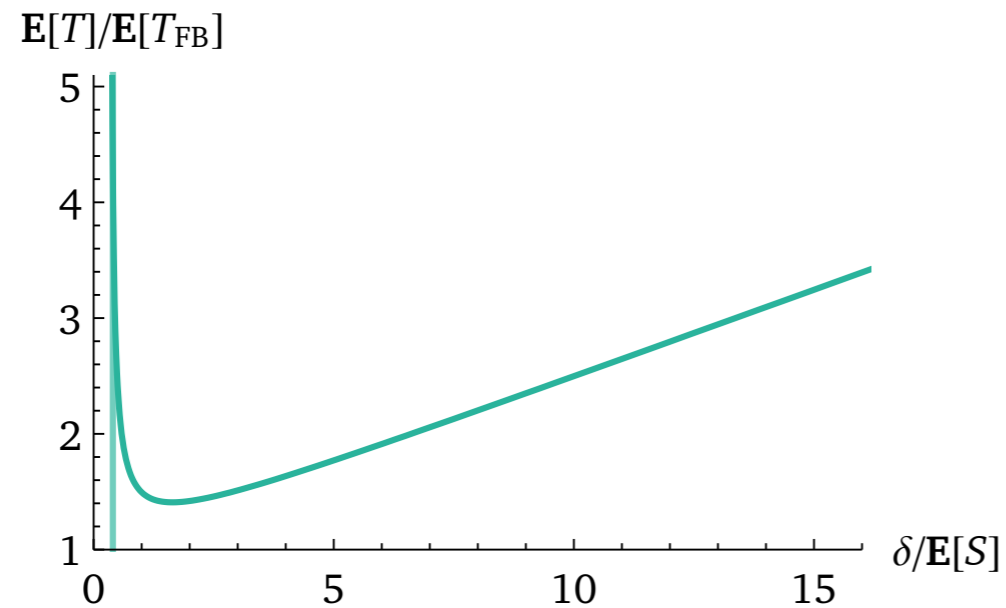


WEIBULL, $\gamma = 0.1 \mathbf{E}[S]$, $\rho = 0.8$

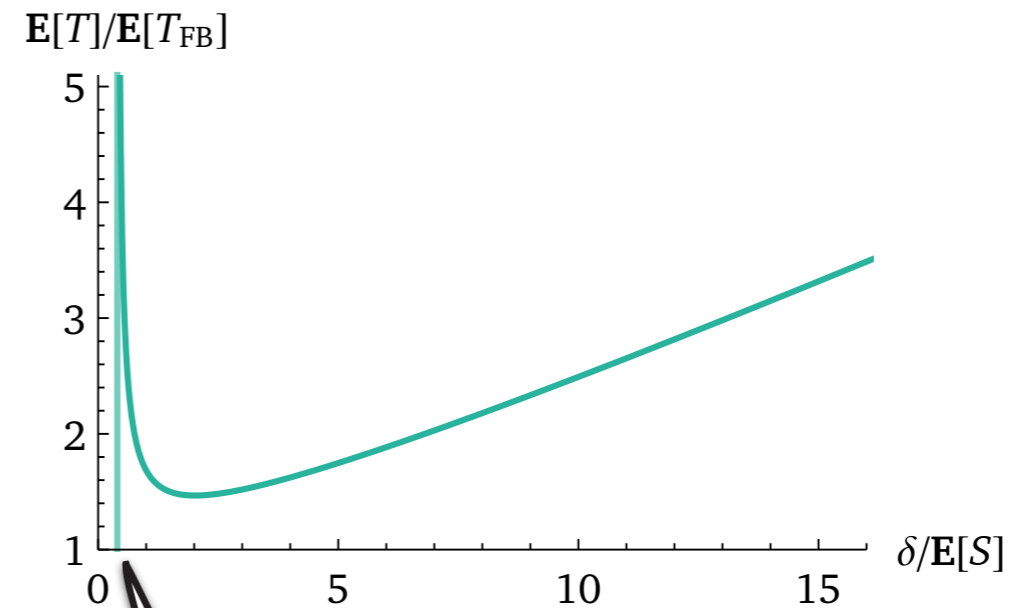


Ensuring Stability

BOUNDED PARETO, $\gamma = 0.1 \mathbf{E}[S]$, $\rho = 0.8$

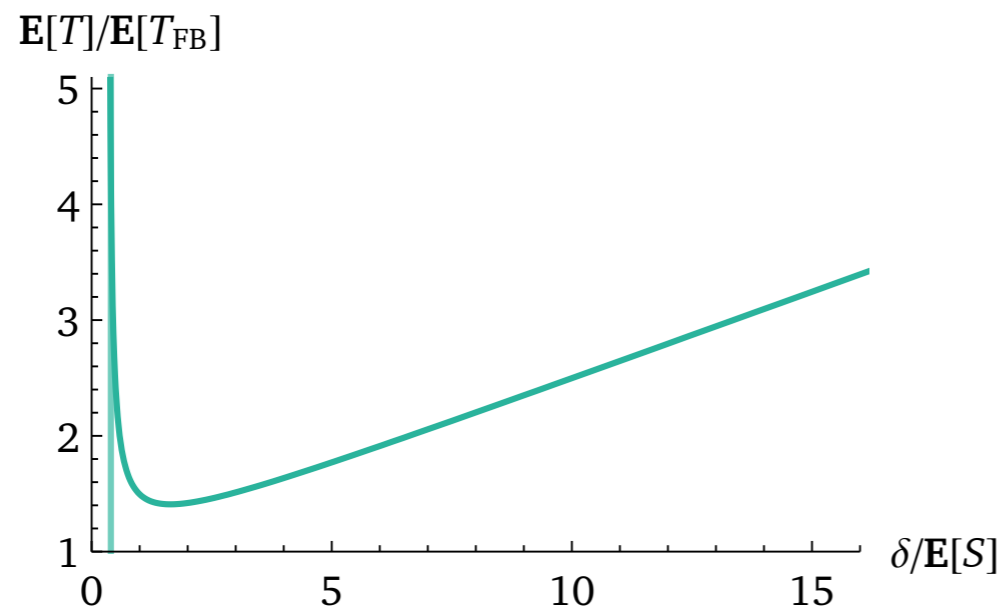


WEIBULL, $\gamma = 0.1 \mathbf{E}[S]$, $\rho = 0.8$

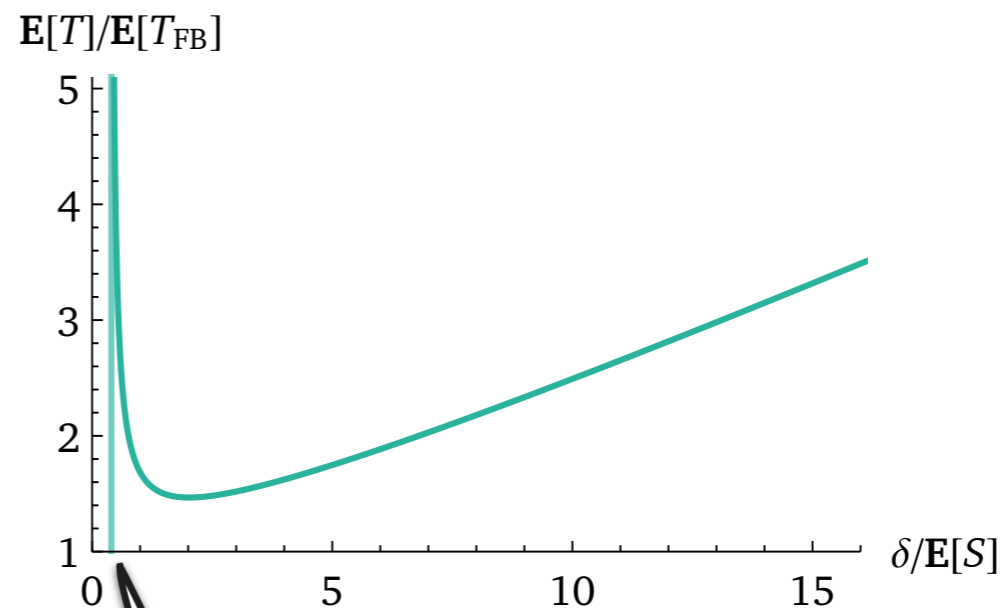


Ensuring Stability

BOUNDED PARETO, $\gamma = 0.1 \mathbf{E}[S]$, $\rho = 0.8$



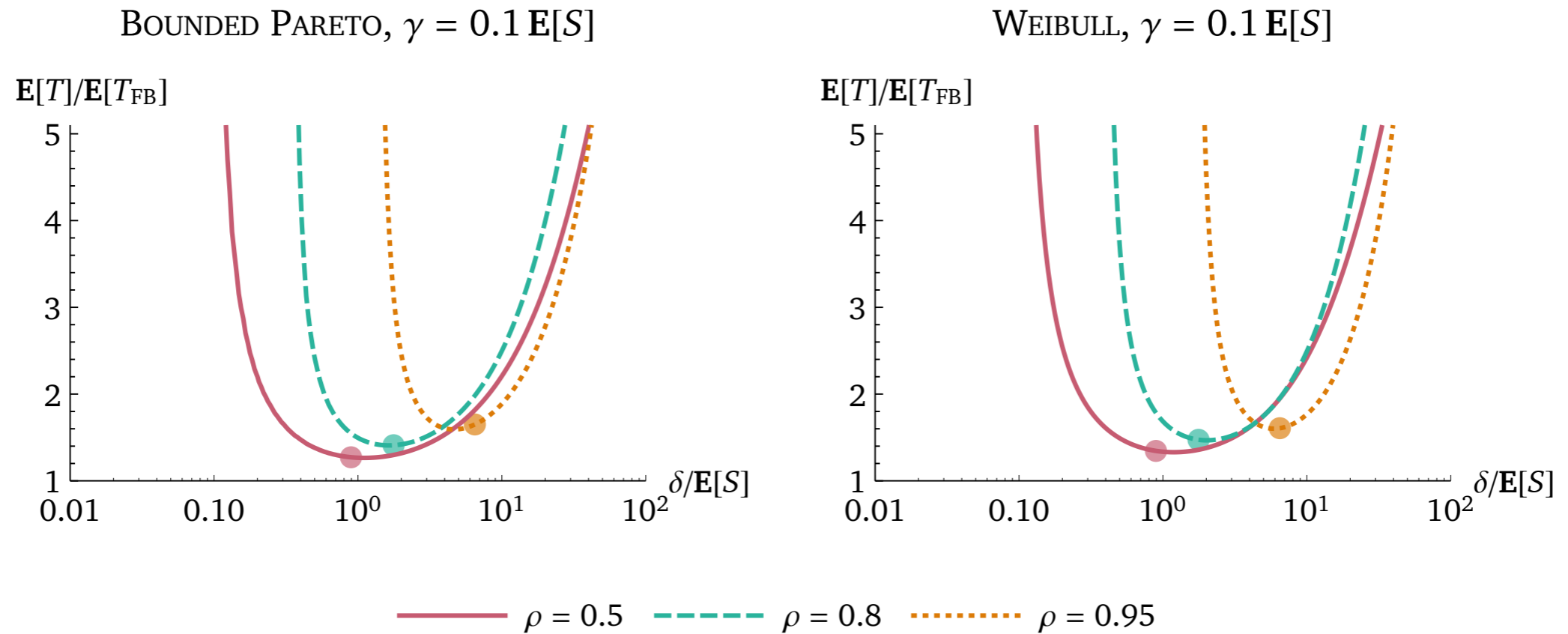
WEIBULL, $\gamma = 0.1 \mathbf{E}[S]$, $\rho = 0.8$



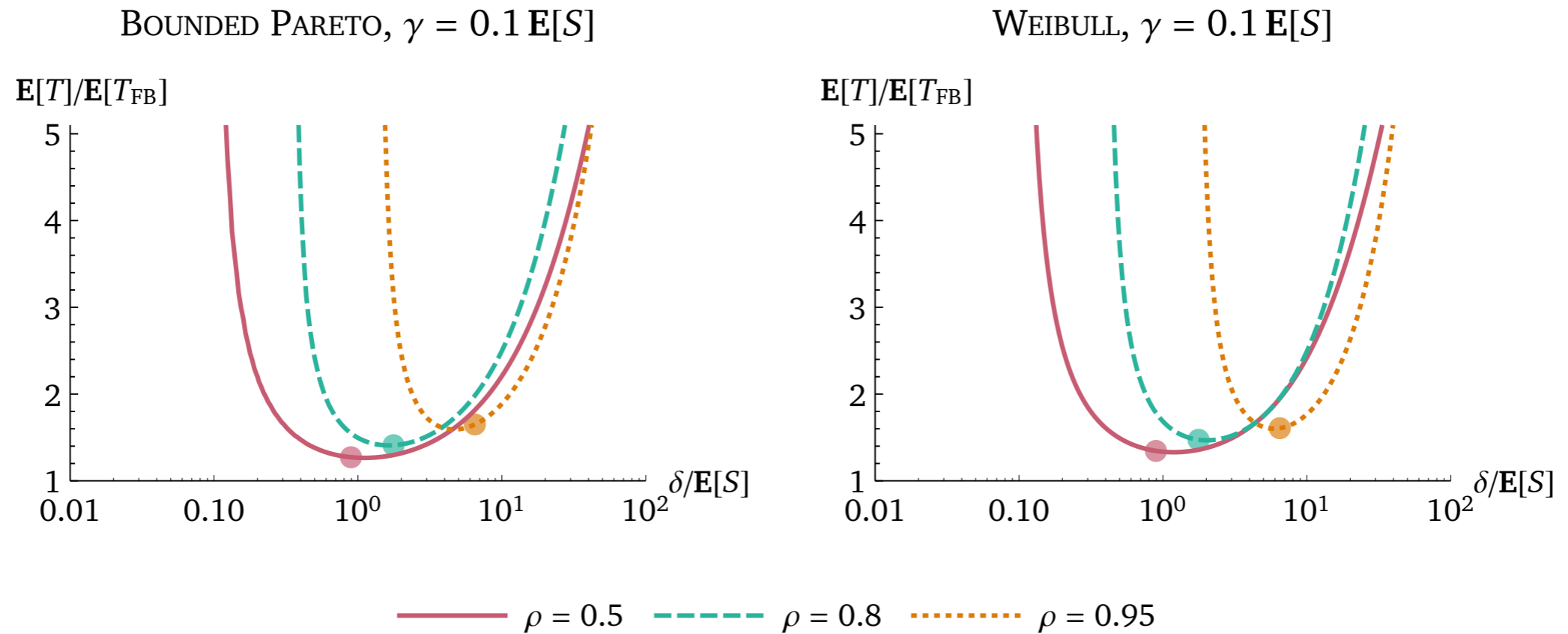
minimum safe
packet size

$$\delta_{\text{safe}} = \frac{1}{1 - \rho} \gamma \rho$$

Packet Size Heuristic



Packet Size Heuristic



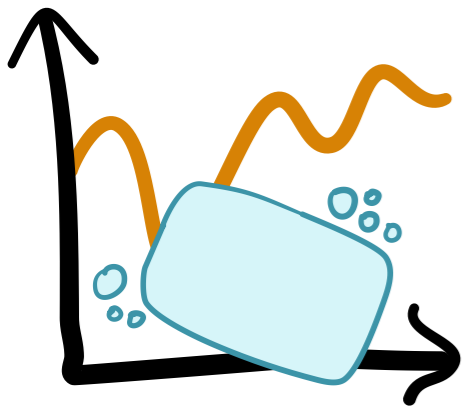
$$\delta_{\text{heuristic}} = \frac{1}{1 - \rho} \sqrt{\frac{\gamma \mathbf{E}[S]}{\rho}}$$

Overview

New Tools

SOAP

*analyzes a huge variety
of scheduling heuristics*



Goals

*Simple implementation
preferred*

*Preemption restricted
and/or costly*



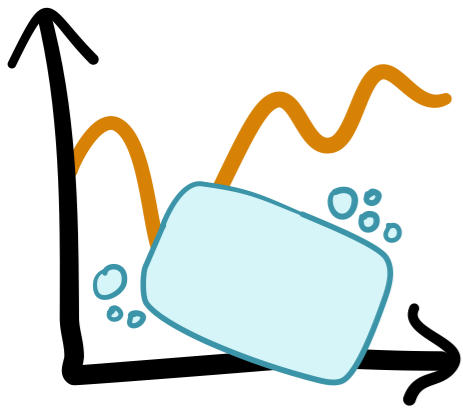
*Limited number
of priority levels*

Overview

New Tools

SOAP

*analyzes a huge variety
of scheduling heuristics*



Goals

*Simple implementation
preferred*

✓ *Preemption restricted
and/or costly*

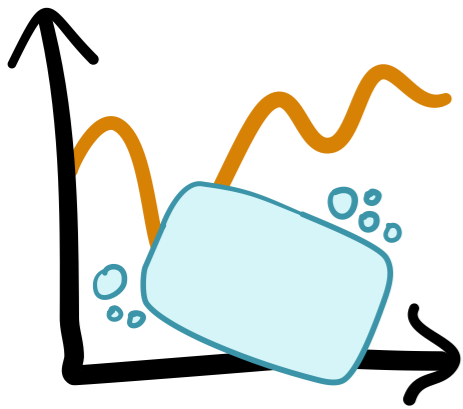
✓ *Limited number
of priority levels*

Overview

New Tools

SOAP

*analyzes a huge variety
of scheduling heuristics*



Goals

*Simple implementation
preferred*

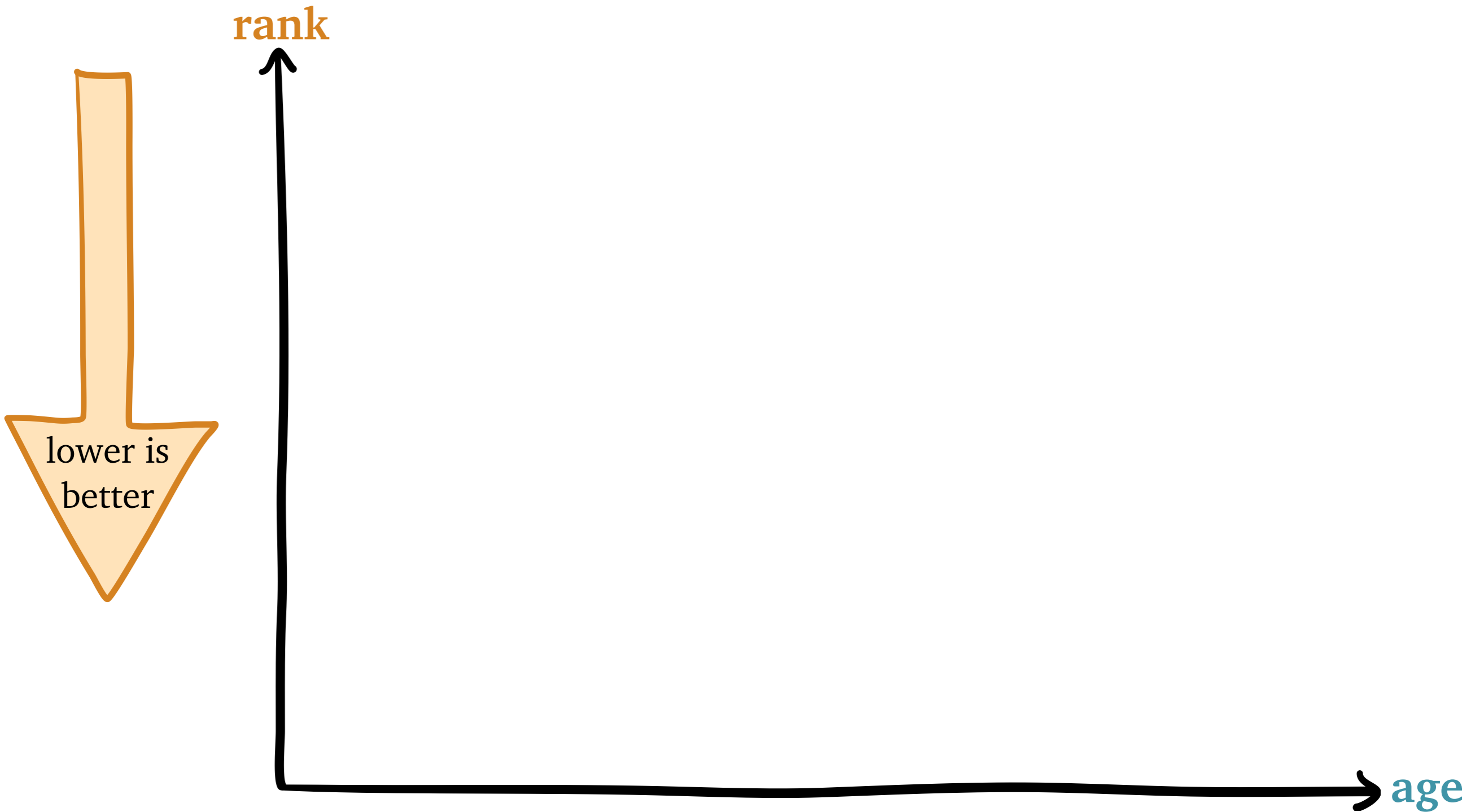


*Preemption restricted
and/or costly*

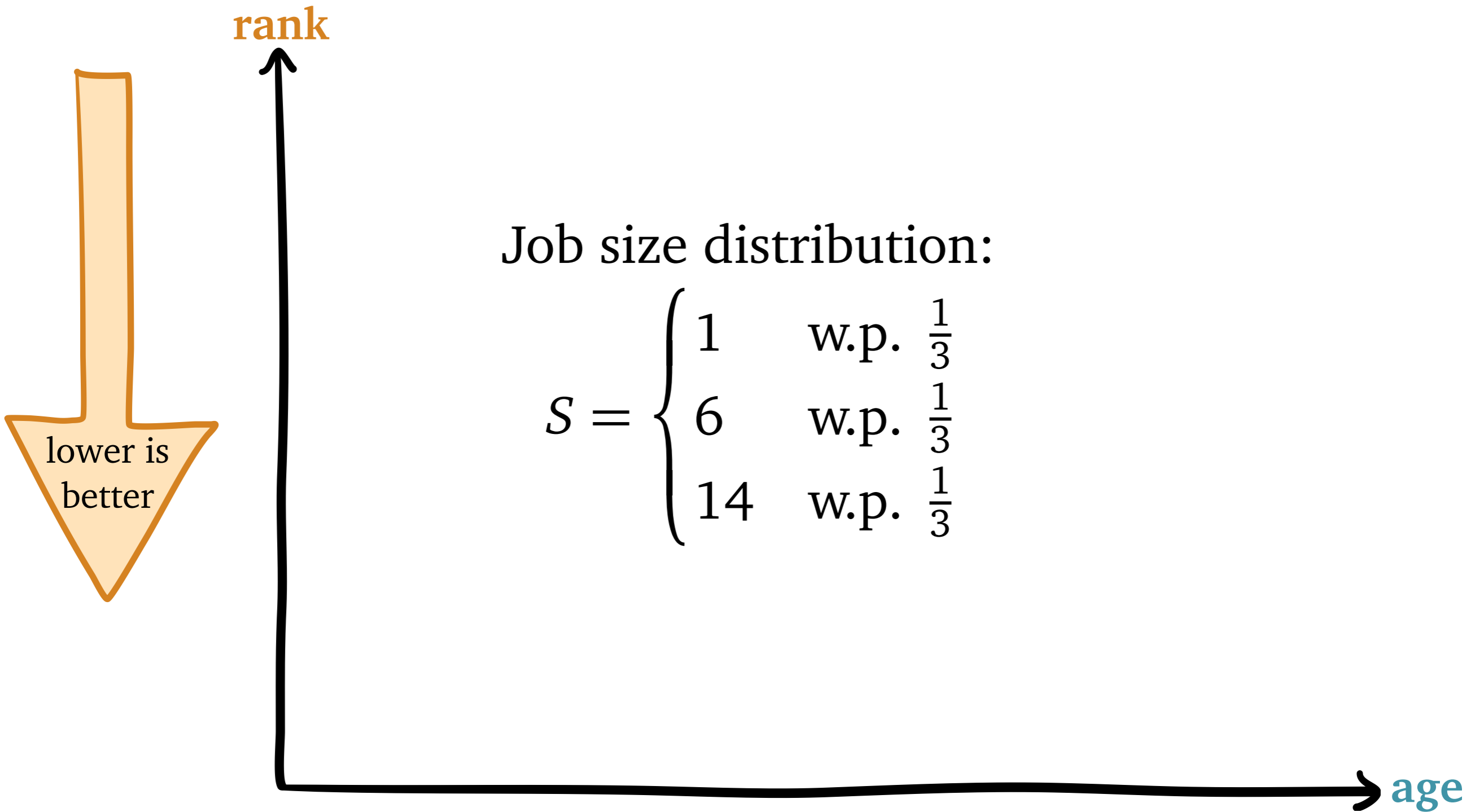


*Limited number
of priority levels*

Gittins: Optimal but Complex



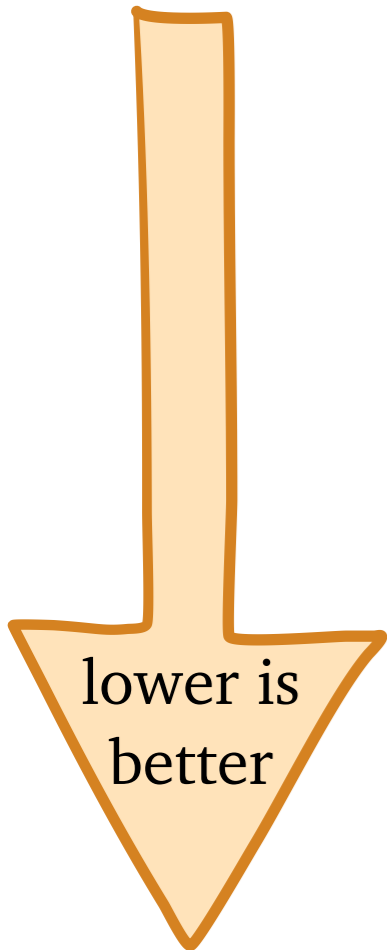
Gittins: Optimal but Complex



Gittins: Optimal but Complex

$$r_{\text{Gittins}}(a) = \inf_{b>a} \frac{\mathbf{E}[\min\{S - a, b\} \mid S > a]}{\mathbf{P}[S \leq b \mid S > a]}$$

rank



Job size distribution:

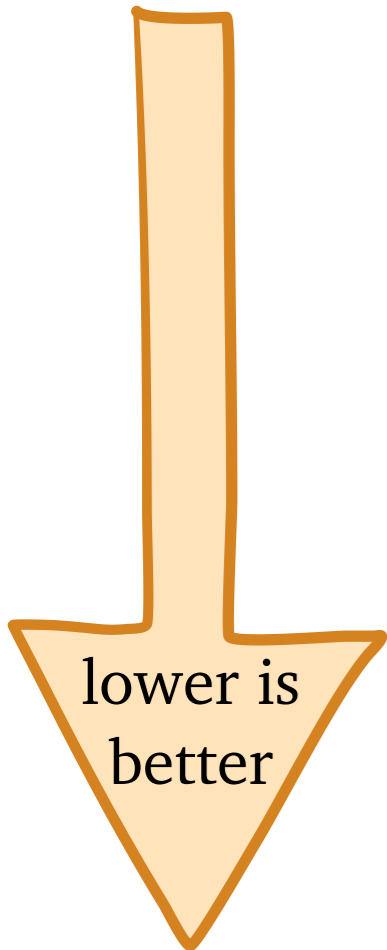
$$S = \begin{cases} 1 & \text{w.p. } \frac{1}{3} \\ 6 & \text{w.p. } \frac{1}{3} \\ 14 & \text{w.p. } \frac{1}{3} \end{cases}$$

age

Gittins: Optimal but Complex

$$r_{\text{Gittins}}(a) = \inf_{b>a} \frac{\mathbf{E}[\min\{S - a, b\} \mid S > a]}{\mathbf{P}[S \leq b \mid S > a]}$$

rank



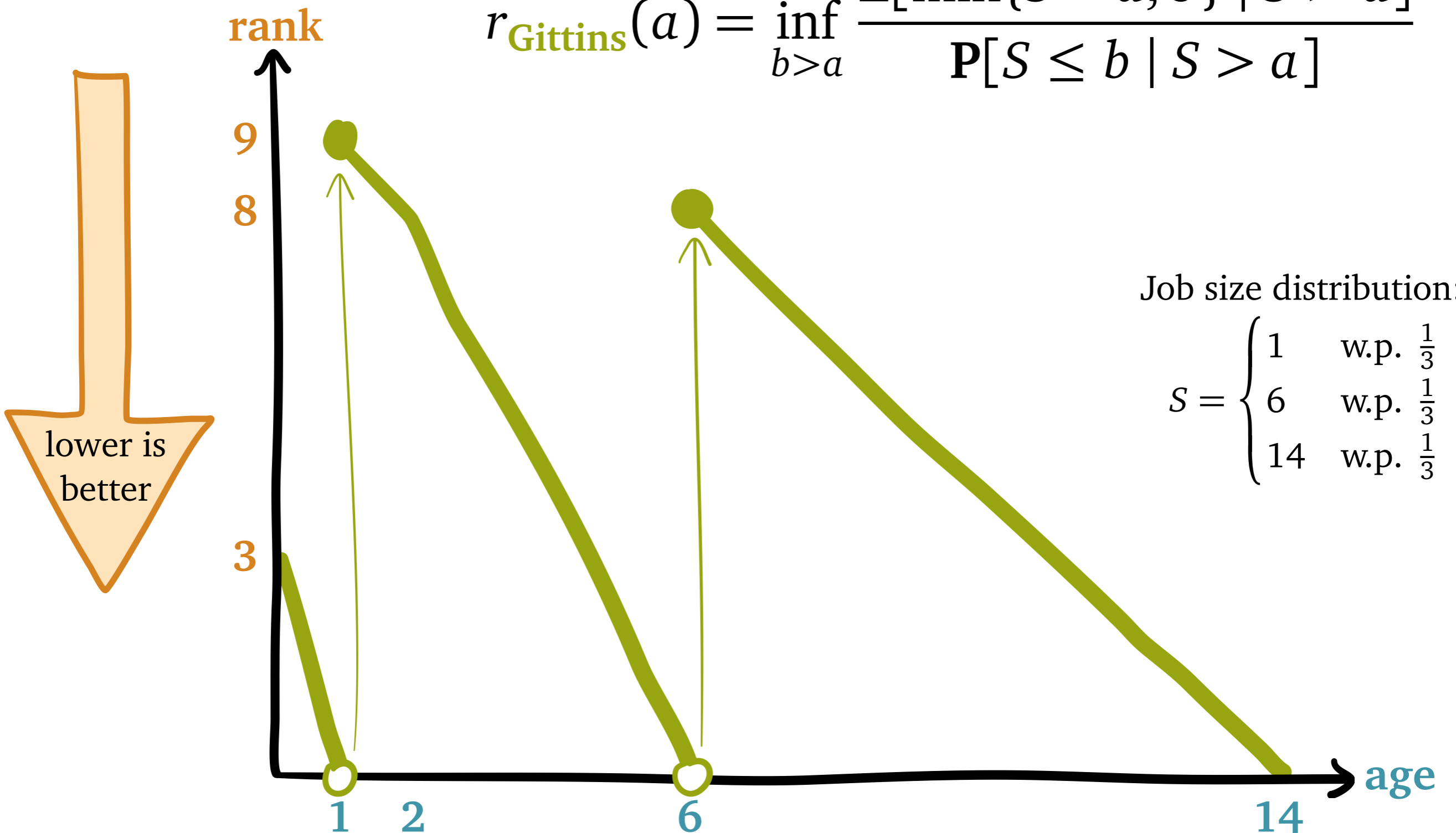
Job size distribution:

$$S = \begin{cases} 1 & \text{w.p. } \frac{1}{3} \\ 6 & \text{w.p. } \frac{1}{3} \\ 14 & \text{w.p. } \frac{1}{3} \end{cases}$$

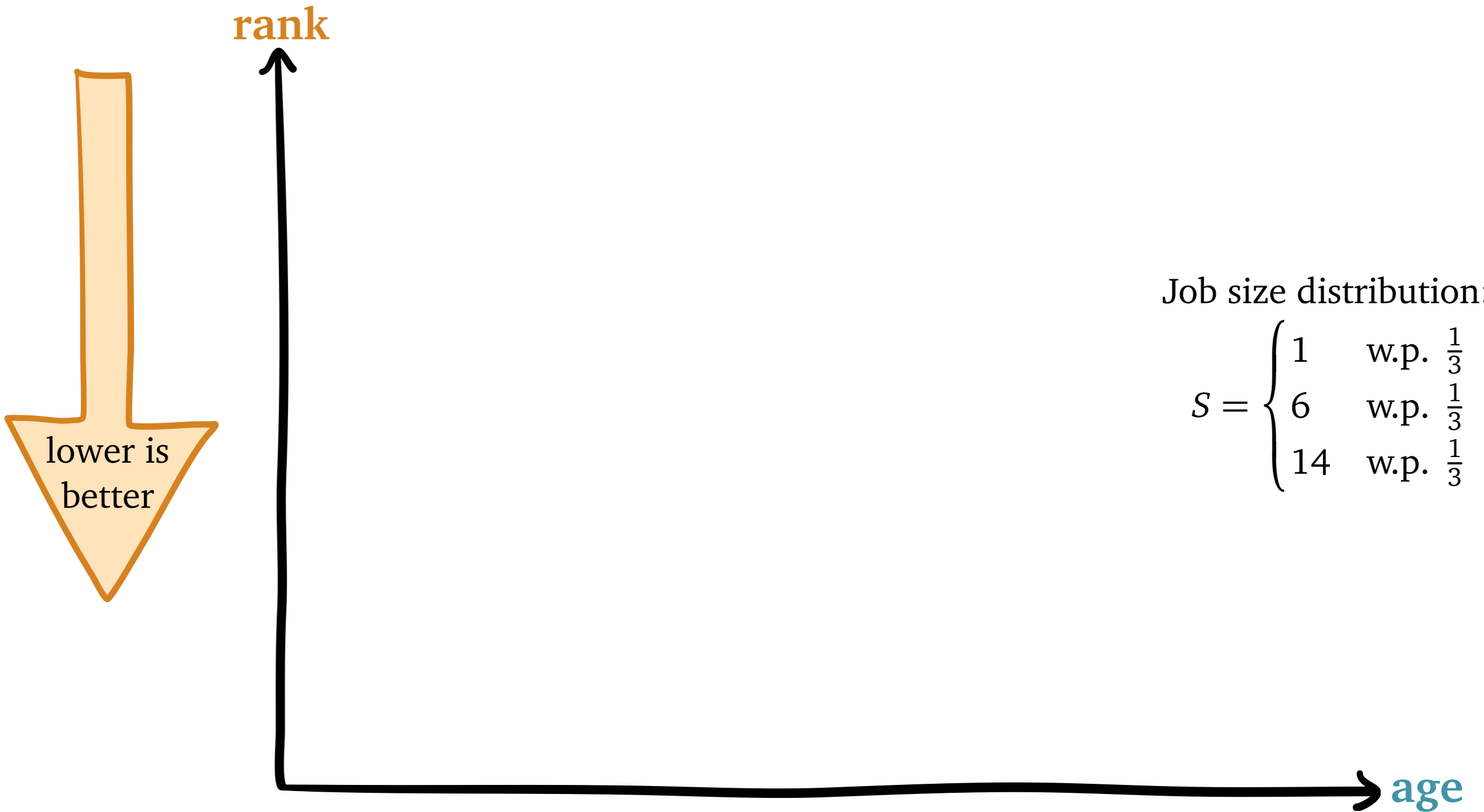
age

Gittins: Optimal but Complex

$$r_{\text{Gittins}}(a) = \inf_{b>a} \frac{\mathbb{E}[\min\{S - a, b\} \mid S > a]}{\mathbb{P}[S \leq b \mid S > a]}$$



SERPT: Simple Heuristic



SERPT: Simple Heuristic

shortest *expected* remaining
processing time

lower is
better

Job size distribution:

$$S = \begin{cases} 1 & \text{w.p. } \frac{1}{3} \\ 6 & \text{w.p. } \frac{1}{3} \\ 14 & \text{w.p. } \frac{1}{3} \end{cases}$$

age

SERPT: Simple Heuristic

shortest *expected* remaining processing time

$$r_{\text{SERPT}}(a) = \mathbf{E}[S - a \mid S > a]$$

lower is better

Job size distribution:

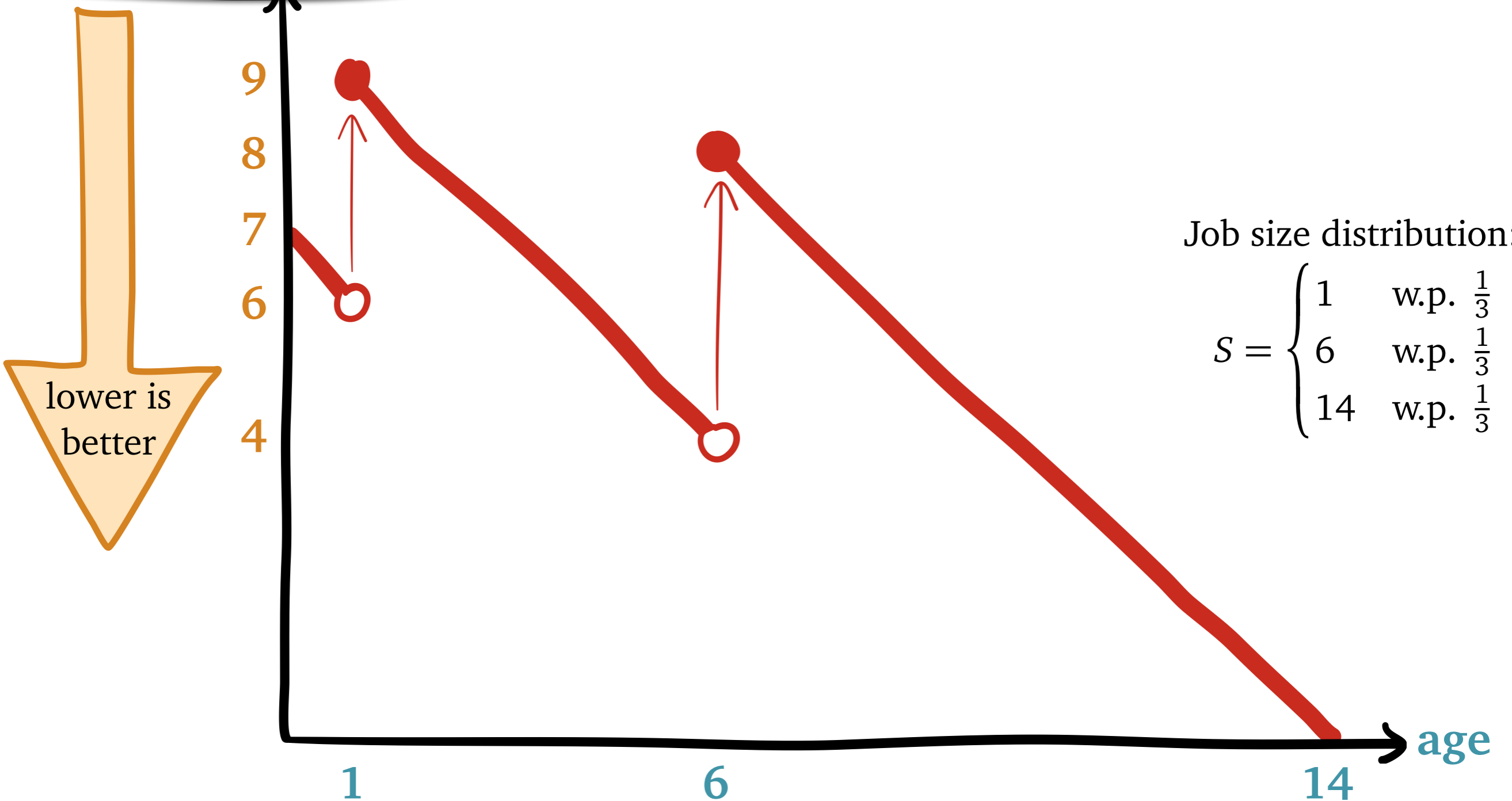
$$S = \begin{cases} 1 & \text{w.p. } \frac{1}{3} \\ 6 & \text{w.p. } \frac{1}{3} \\ 14 & \text{w.p. } \frac{1}{3} \end{cases}$$

age

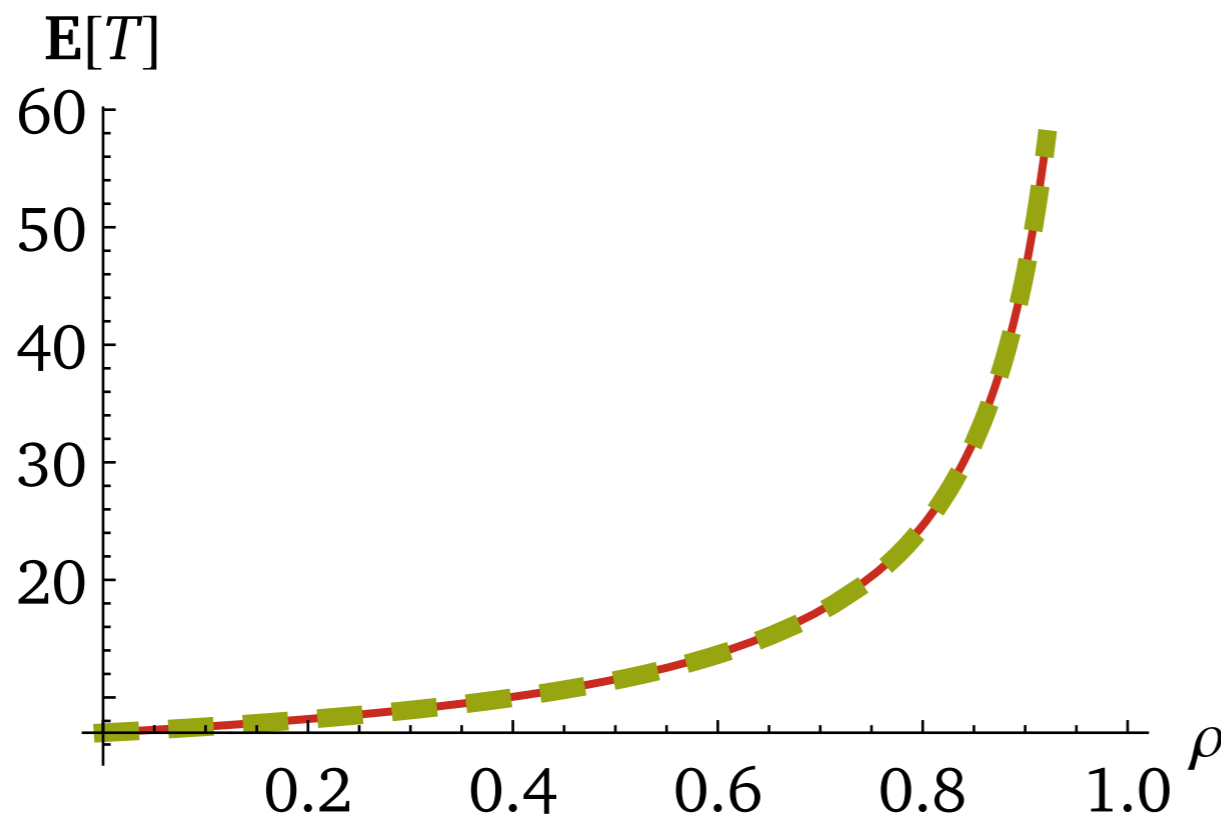
SERPT: Simple Heuristic

shortest *expected* remaining processing time

$$r_{\text{SERPT}}(a) = \mathbf{E}[S - a \mid S > a]$$



Can **SERPT** Replace **Gittins**?



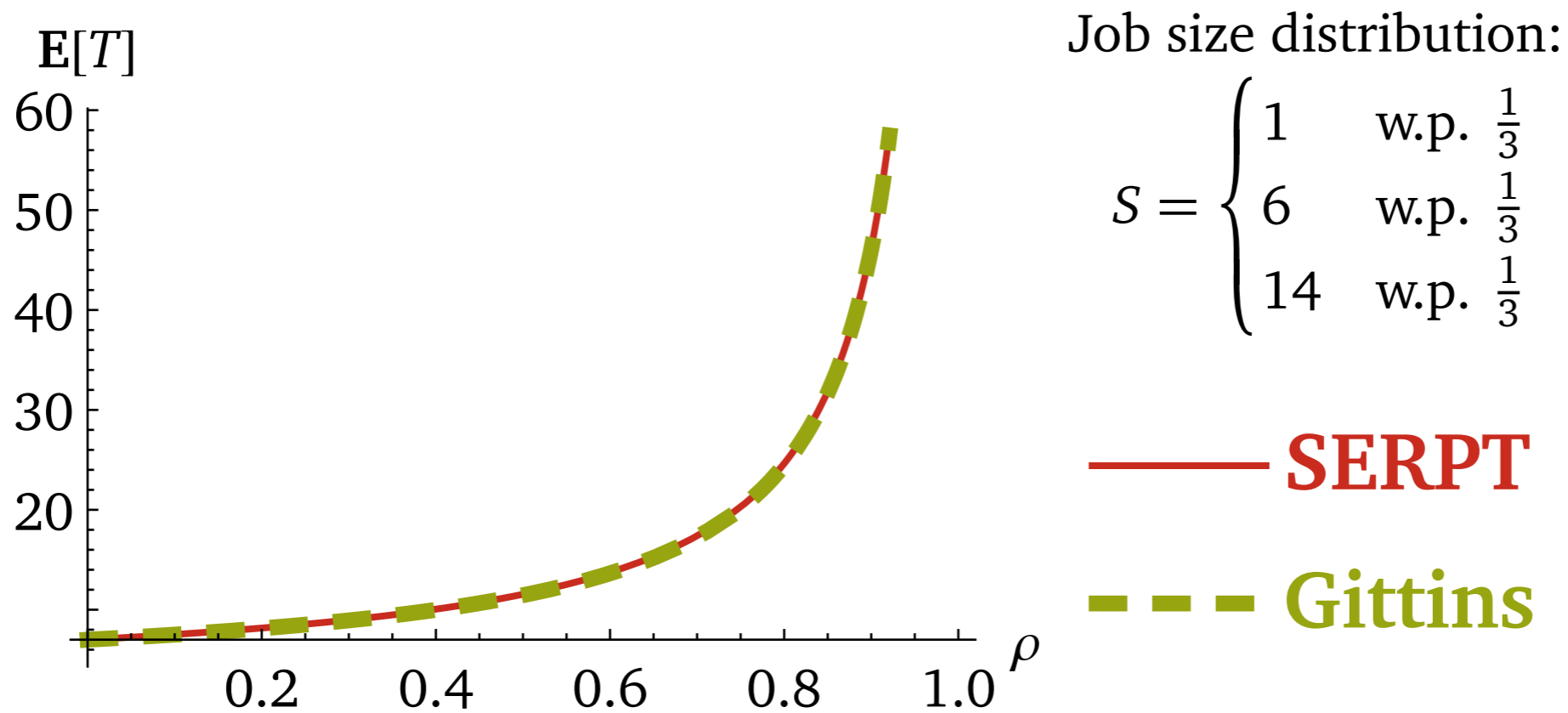
Job size distribution:

$$S = \begin{cases} 1 & \text{w.p. } \frac{1}{3} \\ 6 & \text{w.p. } \frac{1}{3} \\ 14 & \text{w.p. } \frac{1}{3} \end{cases}$$

— **SERPT**

- - - **Gittins**

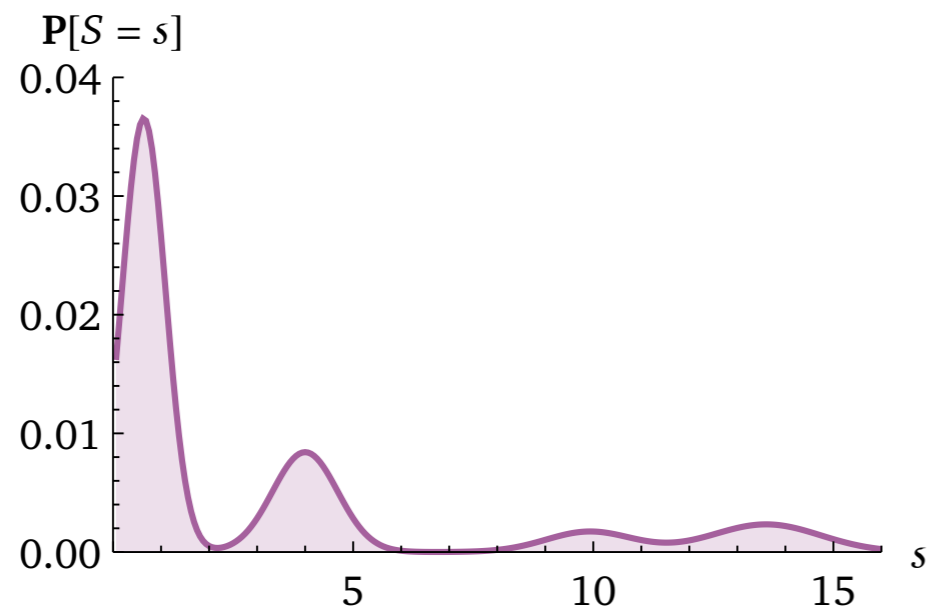
Can **SERPT** Replace **Gittins**?



- **Gittins** is hard to compute
- **SERPT** has no $E[T]$ guarantee

Simplifying Gittins: Practice

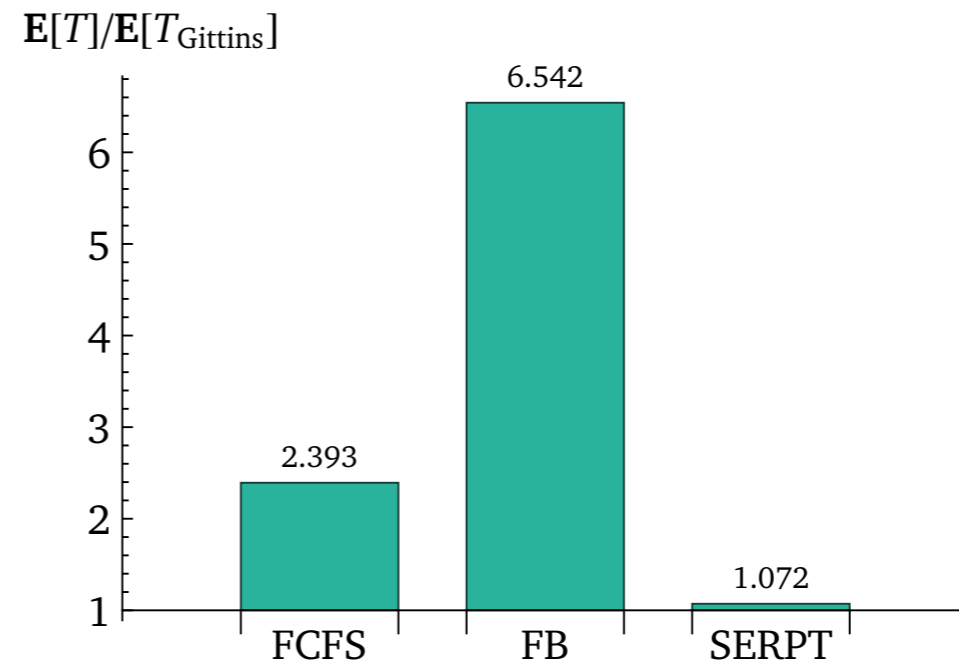
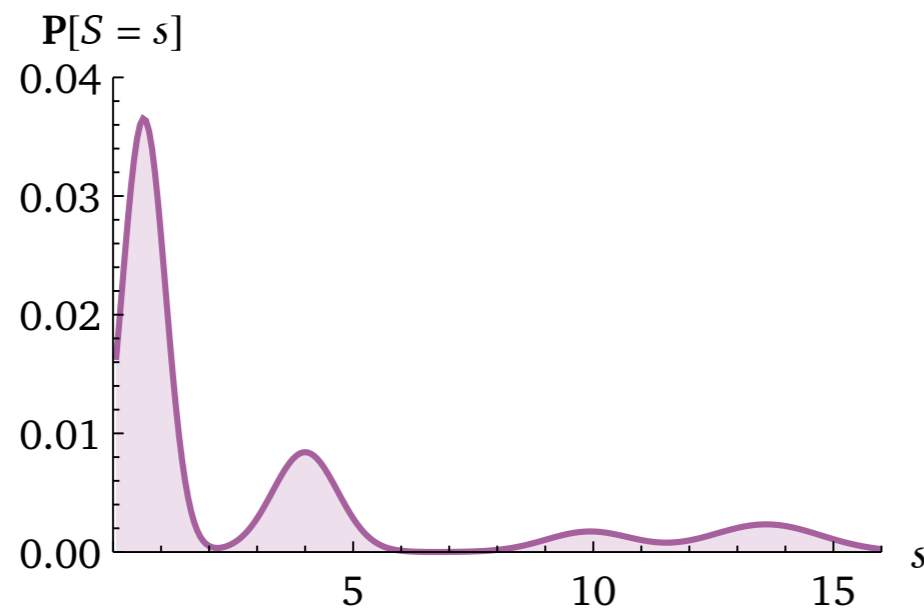
Worst-case performance on 100 distributions:



Simplifying Gittins: Practice

Worst-case performance on 100 distributions:

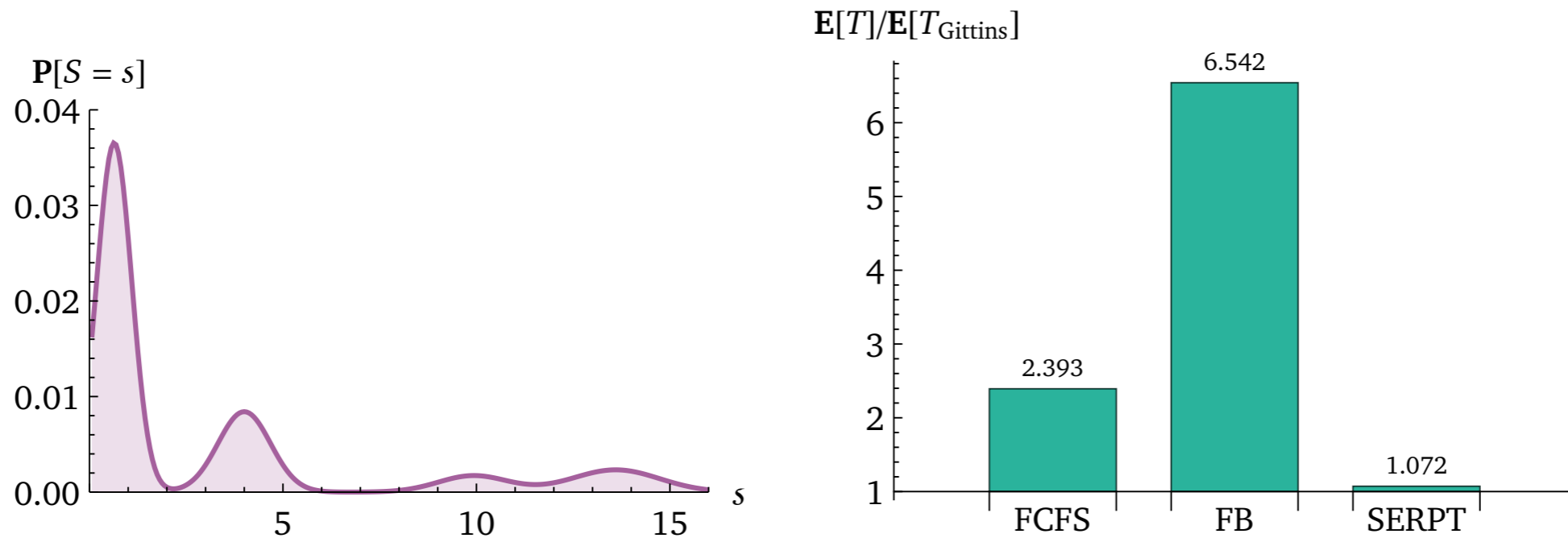
WORST-CASE SIZE-OBLIVIOUS, $\rho = 0.95$



Simplifying **Gittins**: Practice

Worst-case performance on 100 distributions:

WORST-CASE SIZE-OBLIVIOUS, $\rho = 0.95$



... just use **SERPT**

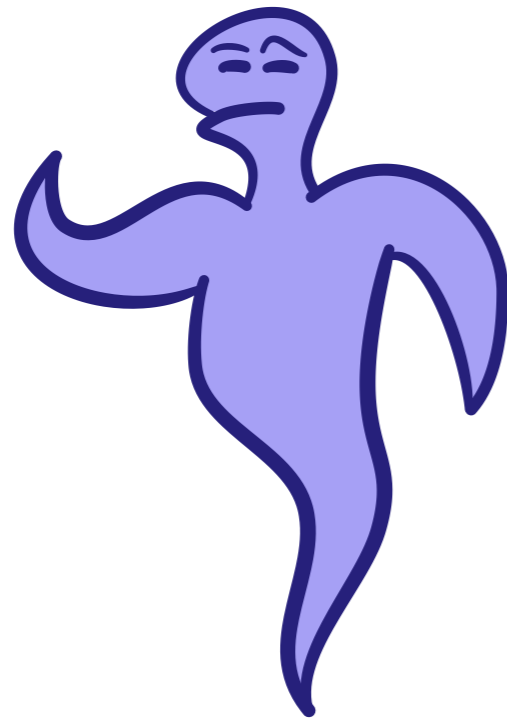
Simplifying **Gittins**: Theory



I wish for a policy with...

- *simple* definition like **SERPT**
- *provable* guarantee on $E[T]$ like **Gittins**

Simplifying **Gittins**: Theory



M-SERPT

I wish for a policy with...

- *simple* definition like **SERPT**
- *provable* guarantee on $E[T]$ like **Gittins**

Introducing **M-SERPT**

rank

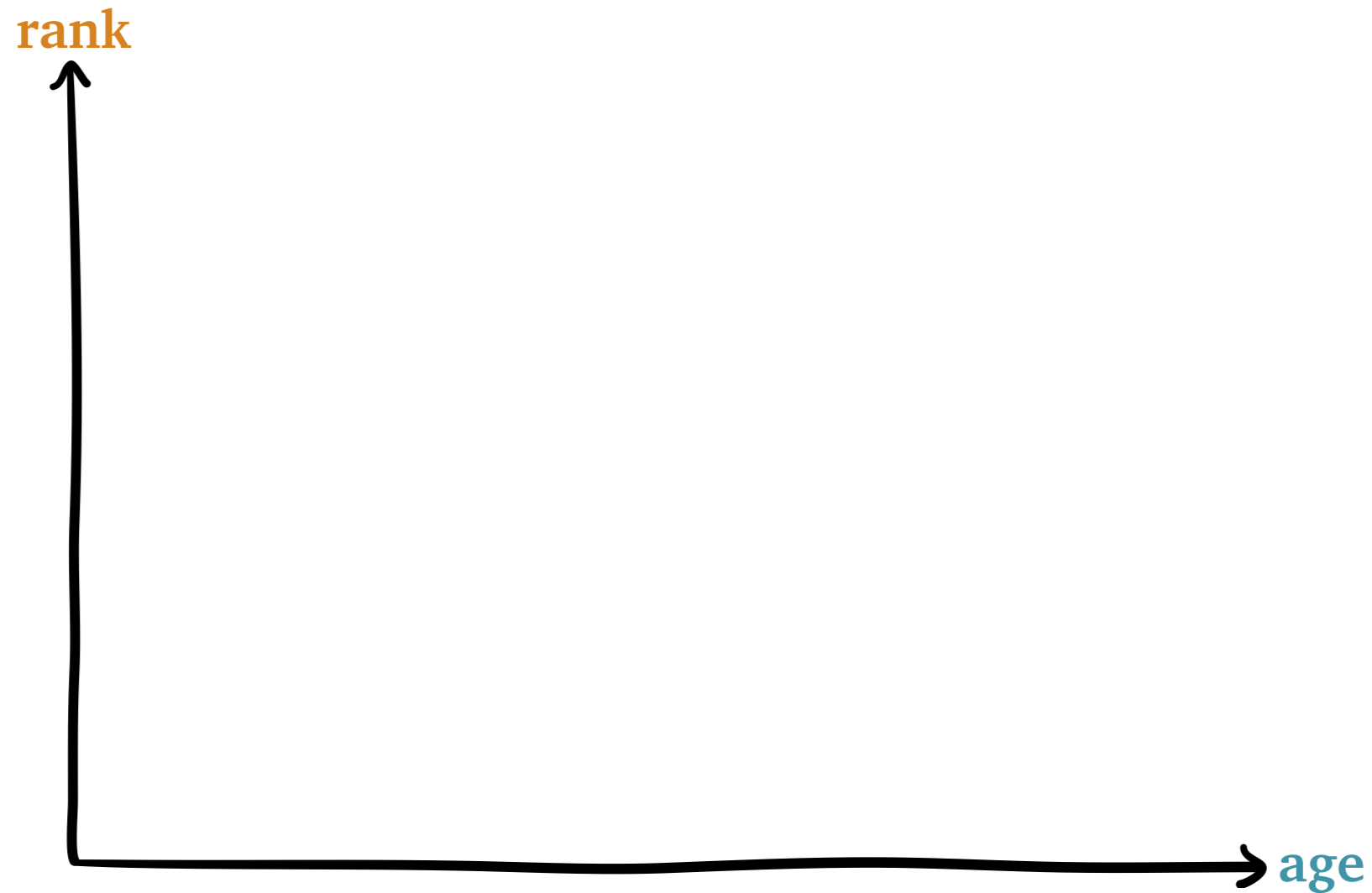


age



Introducing **M-SERPT**

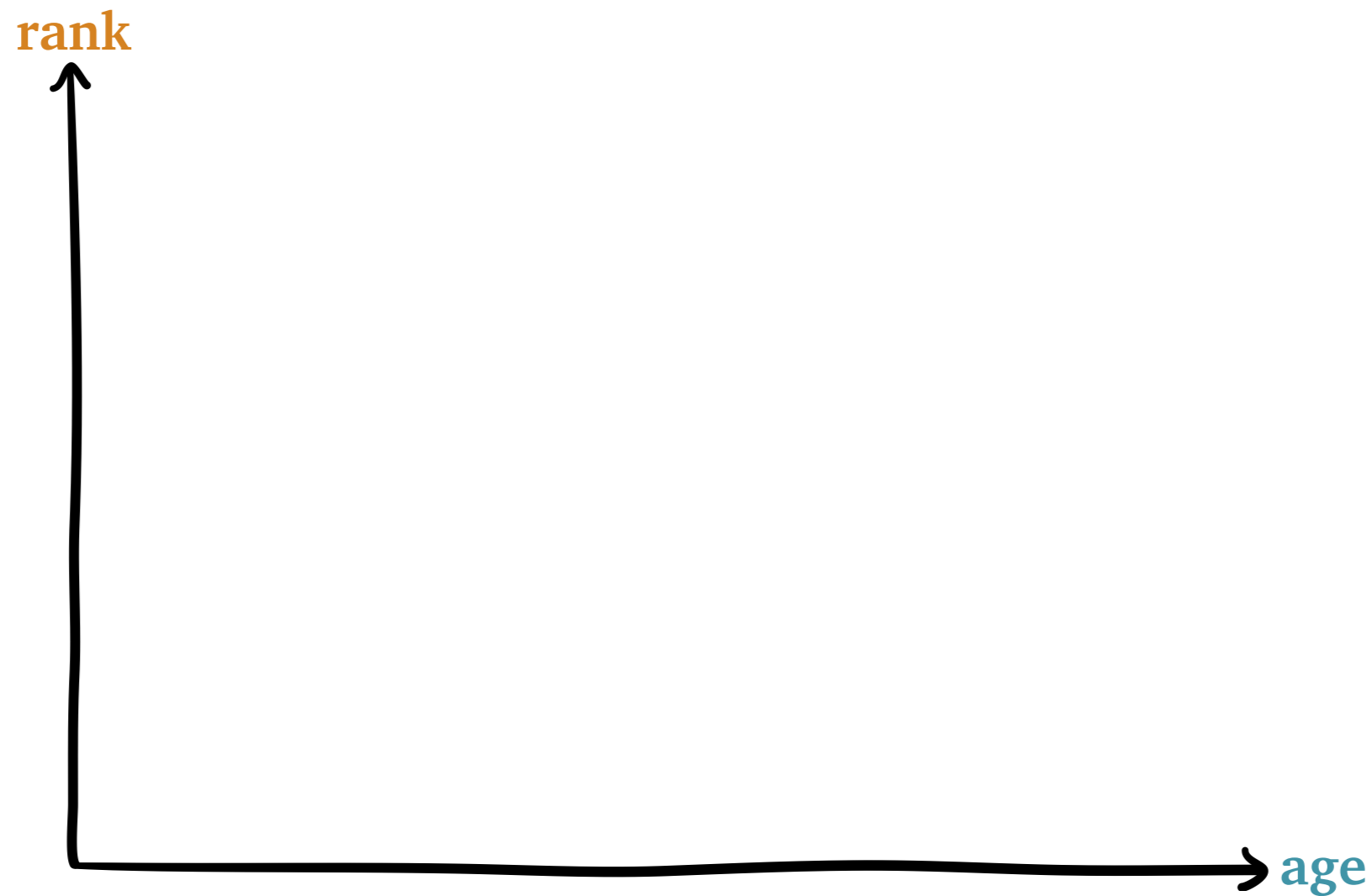
$$r_{\text{M-SERPT}}(a) = \max_{0 \leq b \leq a} r_{\text{SERPT}}(b)$$



Introducing **M-SERPT**

monotonic

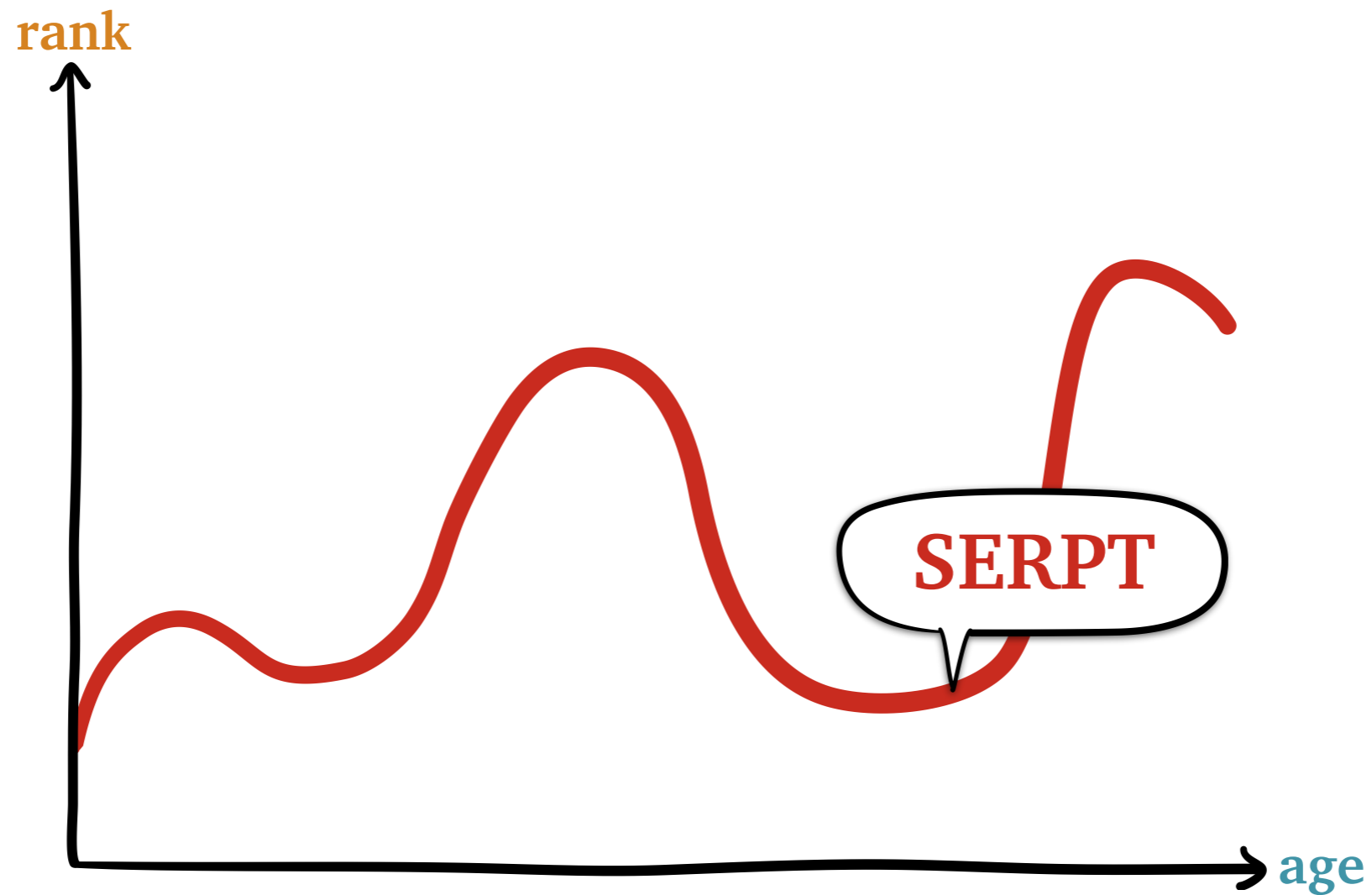
$$r_{\text{M-SERPT}}(a) = \max_{0 \leq b \leq a} r_{\text{SERPT}}(b)$$



Introducing **M-SERPT**

monotonic

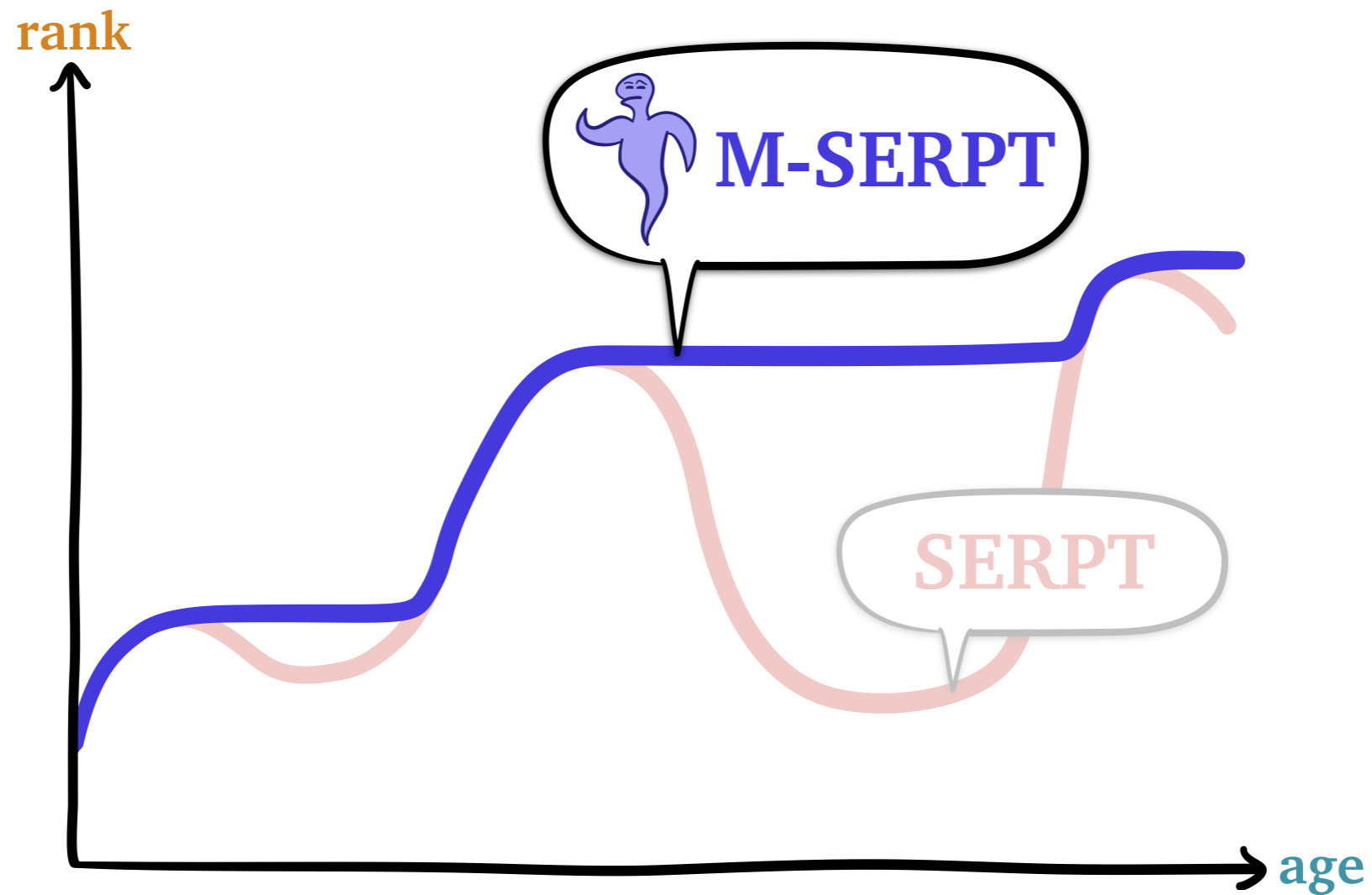
$$r_{\text{M-SERPT}}(a) = \max_{0 \leq b \leq a} r_{\text{SERPT}}(b)$$



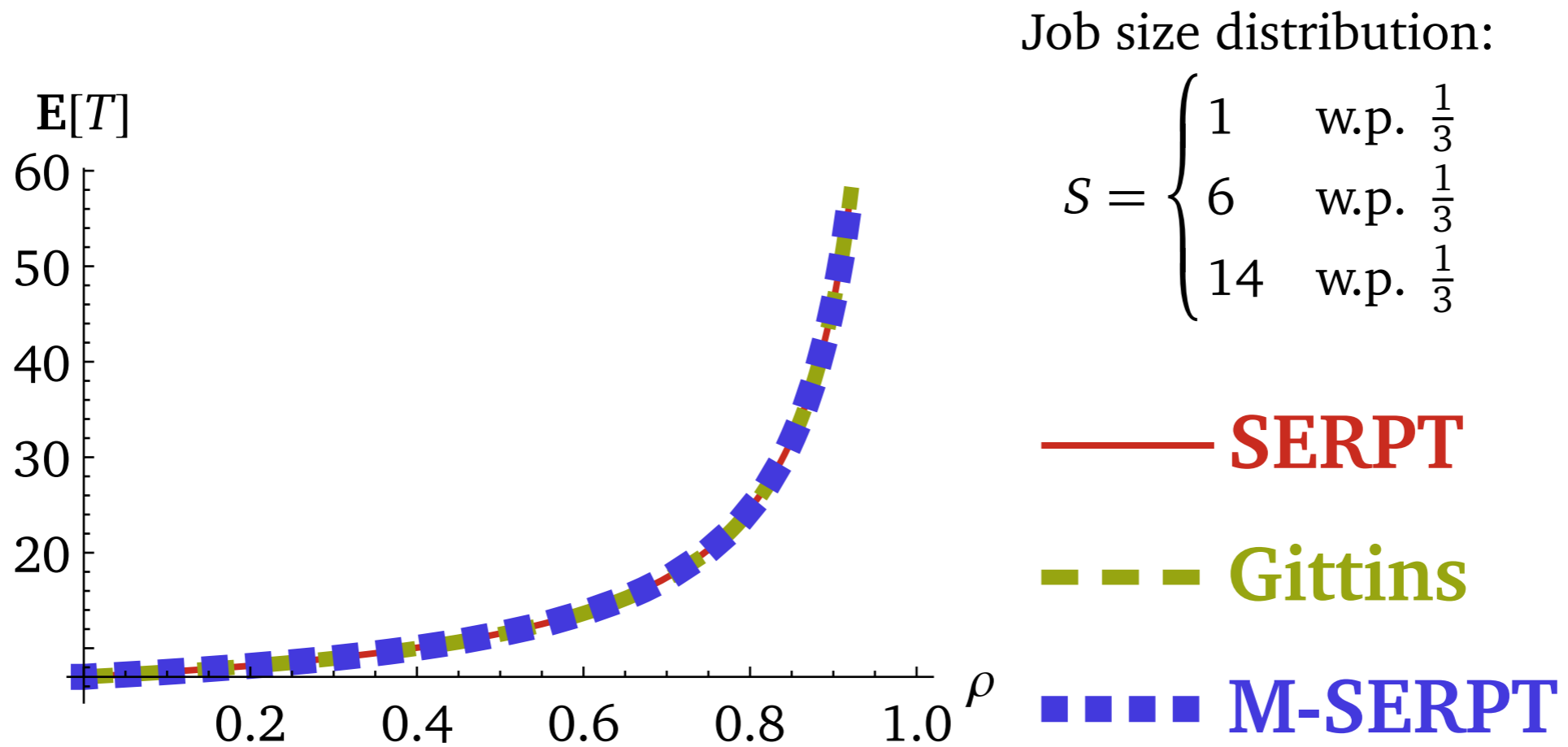
Introducing **M-SERPT**

monotonic

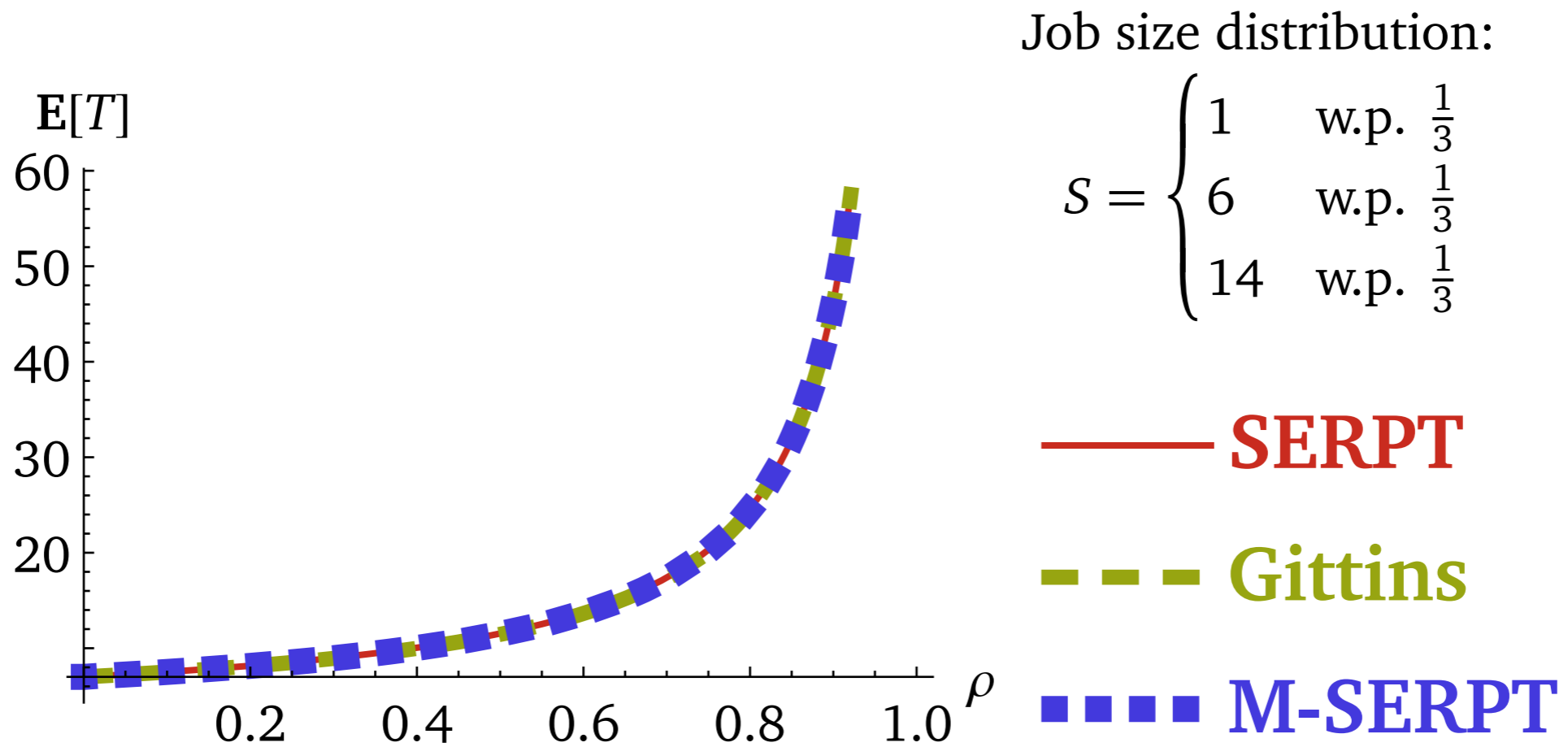
$$r_{\text{M-SERPT}}(a) = \max_{0 \leq b \leq a} r_{\text{SERPT}}(b)$$



M-SERPT Performance



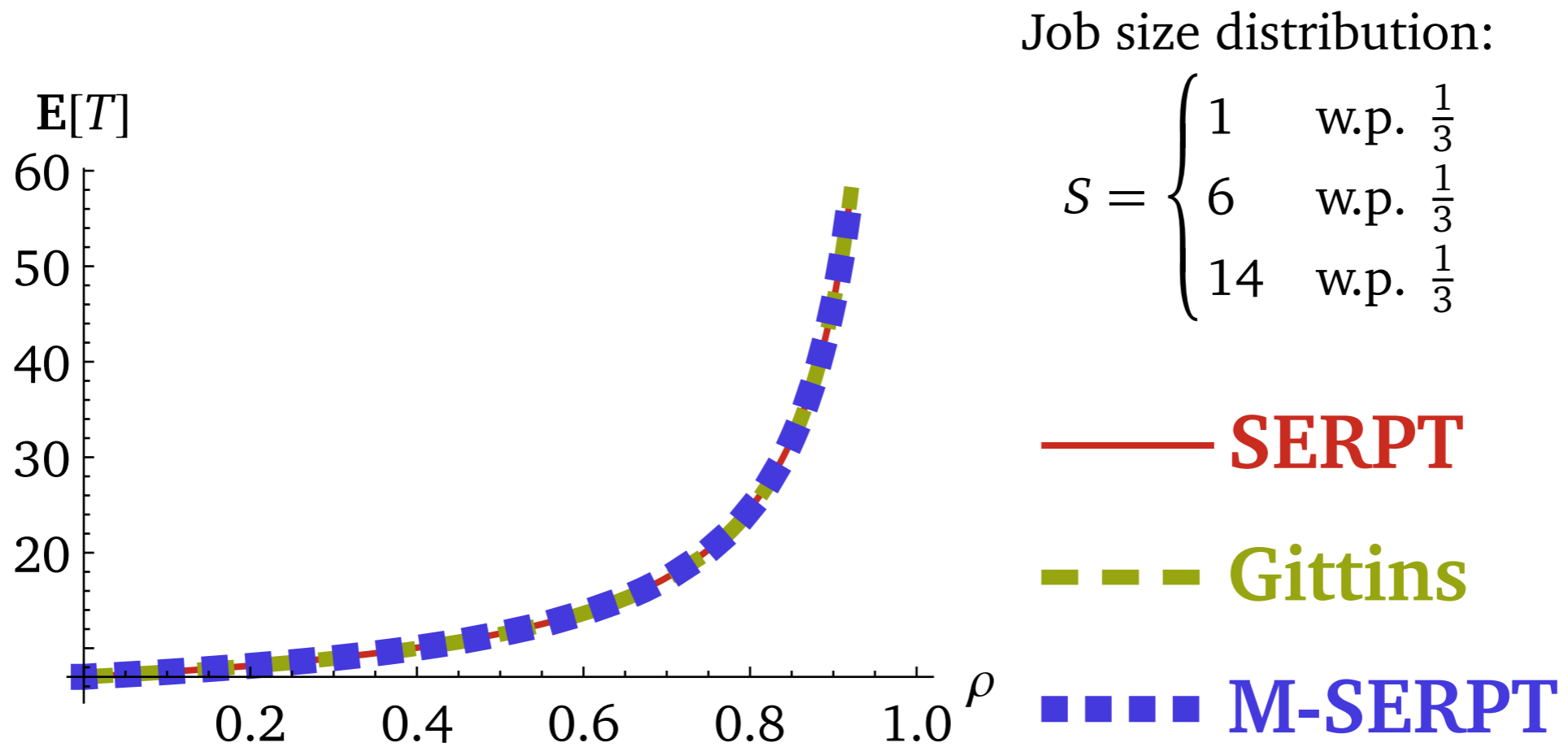
M-SERPT Performance



Theorem:

$$\frac{E[T_{\text{M-SERPT}}]}{E[T_{\text{Gittins}}]} \leq 5$$

M-SERPT Performance

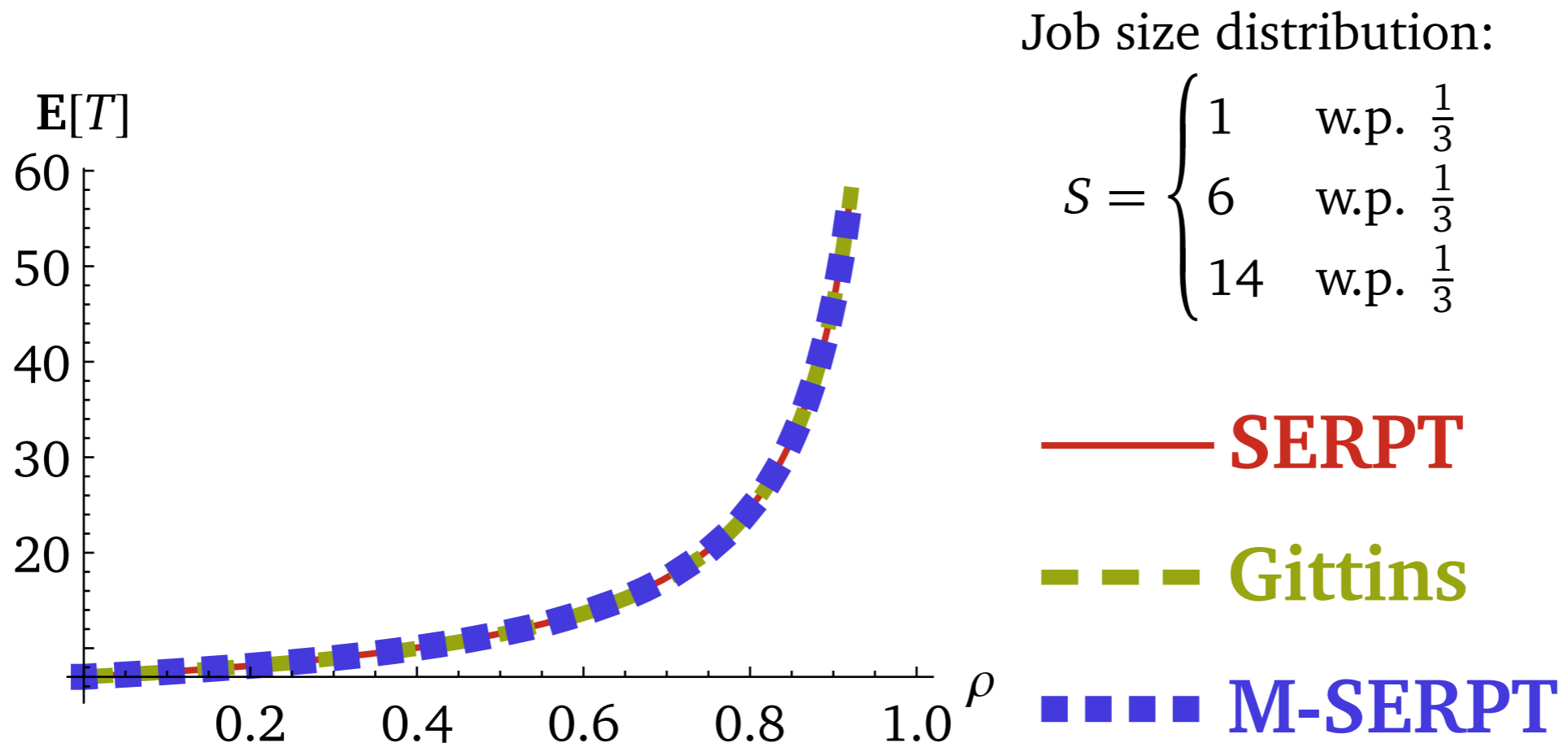


Theorem:

$$\frac{\mathbf{E}[T_{\text{M-SERPT}}]}{\mathbf{E}[T_{\text{Gittins}}]} \leq 5$$

smaller at low load
 first constant ratio

M-SERPT Performance



Theorem:

$$\frac{\mathbf{E}[T_{\mathbf{M-SERPT}}]}{\mathbf{E}[T_{\mathbf{Gittins}}]} \leq 5$$

smaller at low load
 first constant ratio

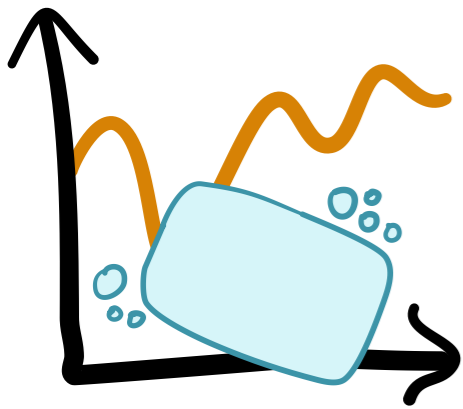
[Scully, Harchol-Balter, & Scheller-Wolf, SIGMETRICS 2020]

Overview

New Tools

SOAP

*analyzes a huge variety
of scheduling heuristics*



Goals

*Simple implementation
preferred*



*Preemption restricted
and/or costly*



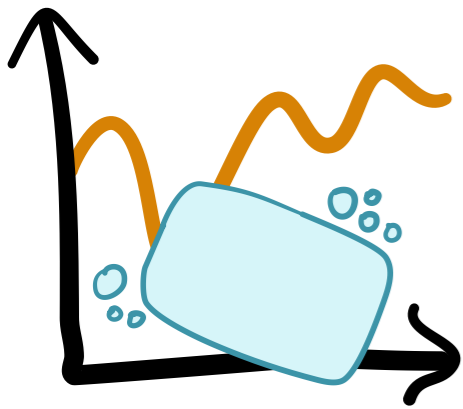
*Limited number
of priority levels*

Overview

New Tools

SOAP

*analyzes a huge variety
of scheduling heuristics*



Goals

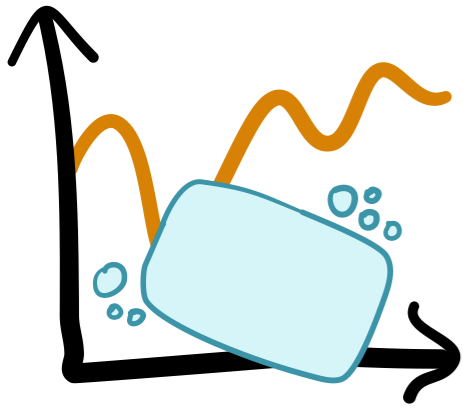
- ✓ *Simple implementation preferred*
- ✓ *Preemption restricted and/or costly*
- ✓ *Limited number of priority levels*

Overview

New Tools

SOAP

*analyzes a huge variety
of scheduling heuristics*



Goals

- ✓ *Simple implementation preferred*
- ✓ *Preemption restricted and/or costly*
- ✓ *Limited number of priority levels*

Future Directions

Future Directions

- Scheduling jobs that *occupy multiple servers*
(e.g. data centers and other clusters)

Future Directions

- Scheduling jobs that *occupy multiple servers*
(e.g. data centers and other clusters)
- Scheduling with *noisy size estimates*
(e.g. transmitting a file of known size over a noisy network)

Future Directions

- Scheduling jobs that *occupy multiple servers*
(e.g. data centers and other clusters)
- Scheduling with *noisy size estimates*
(e.g. transmitting a file of known size over a noisy network)
- Scheduling jobs that must be *served in batches*
(e.g. GPUs and other pipelined systems)

Future Directions

- Scheduling jobs that *occupy multiple servers*
(e.g. data centers and other clusters)
- Scheduling with *noisy size estimates*
(e.g. transmitting a file of known size over a noisy network)
- Scheduling jobs that must be *served in batches*
(e.g. GPUs and other pipelined systems)
- Scheduling when *complete/incomplete is not binary*
(e.g. training machine-learning models)

Future Directions

- Scheduling jobs that *occupy multiple servers*
(e.g. data centers and other clusters)
- Scheduling with *noisy size estimates*
(e.g. transmitting a file of known size over a noisy network)
- Scheduling jobs that must be *served in batches*
(e.g. GPUs and other pipelined systems)
- Scheduling when *complete/incomplete is not binary*
(e.g. training machine-learning models)
- *Your problem here!*

References

Ziv Scully, Mor Harchol-Balter, and Alan Scheller-Wolf.

SOAP: One Clean Analysis of All Age-Based Scheduling Policies.

POMACS, 2018. Presented at SIGMETRICS 2018.

Ziv Scully and Mor Harchol-Balter.

SOAP Bubbles: Robust Scheduling under Adversarial Noise.

Allerton Conference, 2018.

Isaac Grosof, Ziv Scully, and Mor Harchol-Balter.

SRPT for Multiserver Systems.

PEVA, 2018. Presented at PERFORMANCE 2018.

Ziv Scully, Mor Harchol-Balter, and Alan Scheller-Wolf.

Simple Near-Optimal Scheduling for the M/G/1.

POMACS, 2020. Presented at SIGMETRICS 2020.

Ziv Scully, Lucas van Kreveld, Onno J. Boxma, Jan-Pieter Dorsman, and Adam Wierman.

Characterizing Policies with Optimal Response Time Tails under Heavy-Tailed Job Sizes.

POMACS, 2020. Presented at SIGMETRICS 2020.

Ziv Scully, Isaac Grosof, and Mor Harchol-Balter.

Optimal Multiserver Scheduling with Unknown Job Sizes in Heavy Traffic.

PEVA, 2020. Presented at PERFORMANCE 2020.

Ziv Scully, Isaac Grosof, and Mor Harchol-Balter.

The Gittins Policy is Nearly Optimal in the M/G/k under Extremely General Conditions.

POMACS, 2020. Presented at SIGMETRICS 2021.

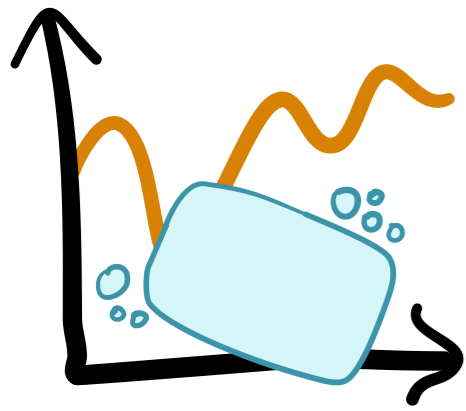
Isaac Grosof, Kunhe Yang, Ziv Scully, and Mor Harchol-Balter.

Nudge: Stochastically Improving upon FCFS.

POMACS, 2021. Presented at SIGMETRICS 2021.

Overview

New Tools



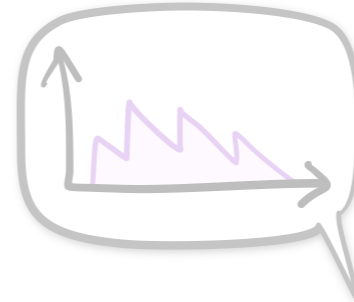
SOAP

analyzes a huge variety of scheduling heuristics

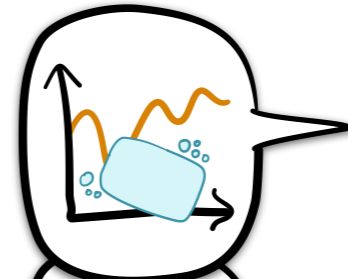
r-Work

provides a new, deeper understanding of Gittins

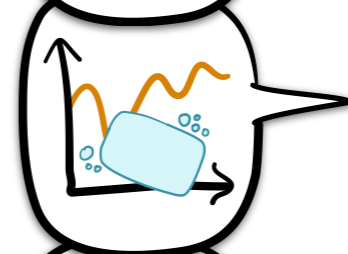
Goals



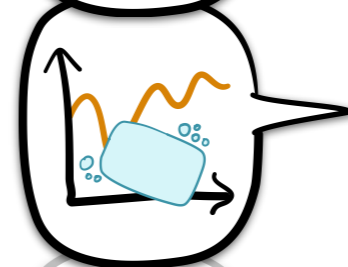
Multiple servers



Simple implementation preferred



Preemption restricted and/or costly



Limited number of priority levels



Want to optimize other response time metrics