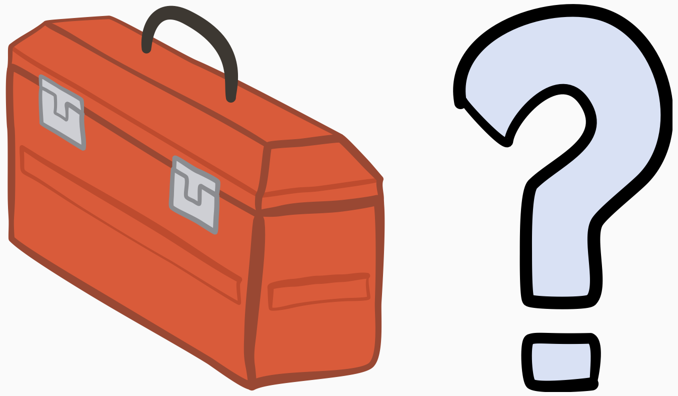
The Role of Advanced Math *in Teaching* Performance Modeling

Ziv Scully
Cornell ORIE

TeaPACS 2023



Goals of performance modeling

Goals of performance modeling

Analyzing systems

Goals of performance modeling

Analyzing systems

Designing and *optimizing* systems

Goals of performance modeling

Analyzing systems

Describing systems

Designing and *optimizing* systems

Goals of performance modeling

Analyzing systems

Describing systems

Designing and *optimizing* systems

What role does math play?

Goals of performance modeling

- exact model analysis
- guide simulation
- what to measure?

Analyzing systems

Describing systems

Designing and *optimizing* systems

What role does math play?

Goals of performance modeling

- exact model analysis
- guide simulation
- what to measure?

Analyzing systems

Describing systems

Designing and *optimizing* systems

- find optimal policies
- evaluate heuristics
- what to optimize?

What role does math play?

Goals of performance modeling

- exact model analysis
- guide simulation
- what to measure?

Analyzing systems

- stochastic modeling
- define load, stability
- what is predictable?

Describing systems

Designing and *optimizing* systems

- find optimal policies
- evaluate heuristics
- what to optimize?

What role does math play?

Goals of performance modeling

- exact model analysis
- guide simulation
- what to measure?

Analyzing systems

- stochastic modeling
- define load, stability
- what is predictable?

Describing systems

Designing and *optimizing* systems

- find optimal policies
- evaluate heuristics
- what to optimize?

How much math?

What role does math play?

Performance modeling needs advanced math

Performance modeling needs advanced math

We can teach advanced math accessibly

Part 1

Performance modeling needs advanced math

Part 2

We can teach advanced math accessibly

Part 1

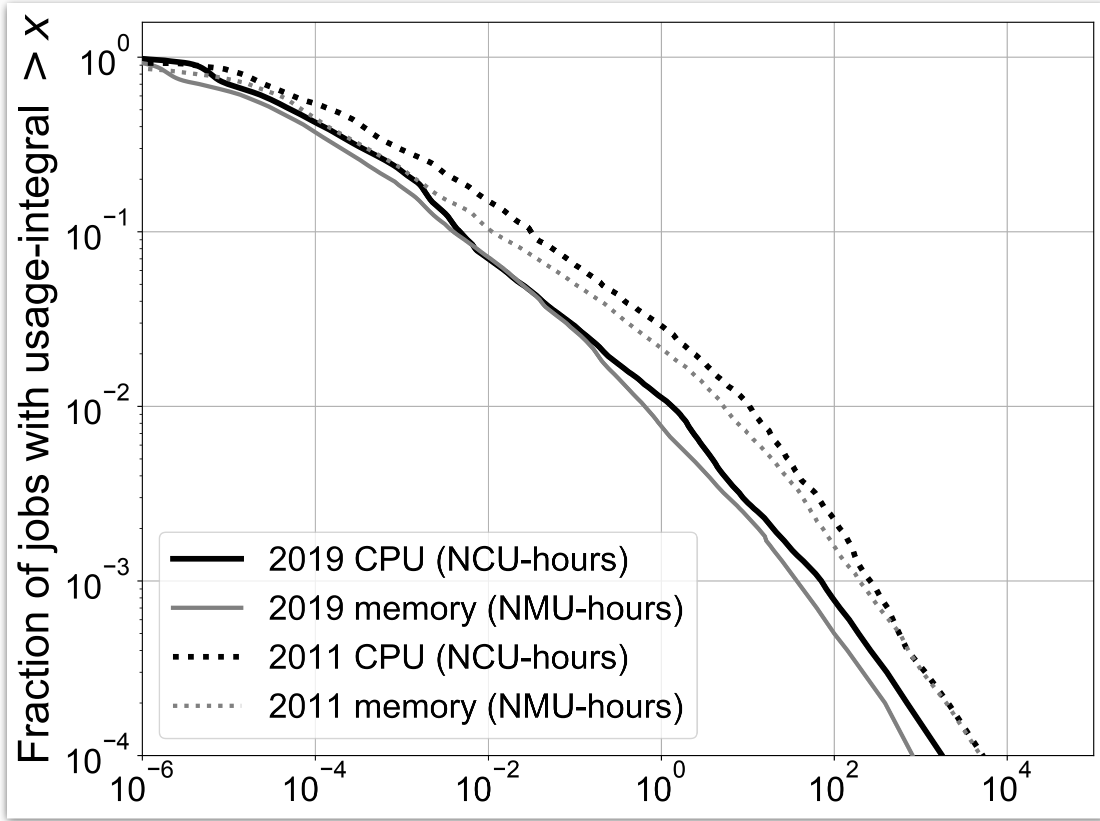
Performance modeling needs advanced math

Part 2

We can teach advanced math accessibly

What do jobs look like?

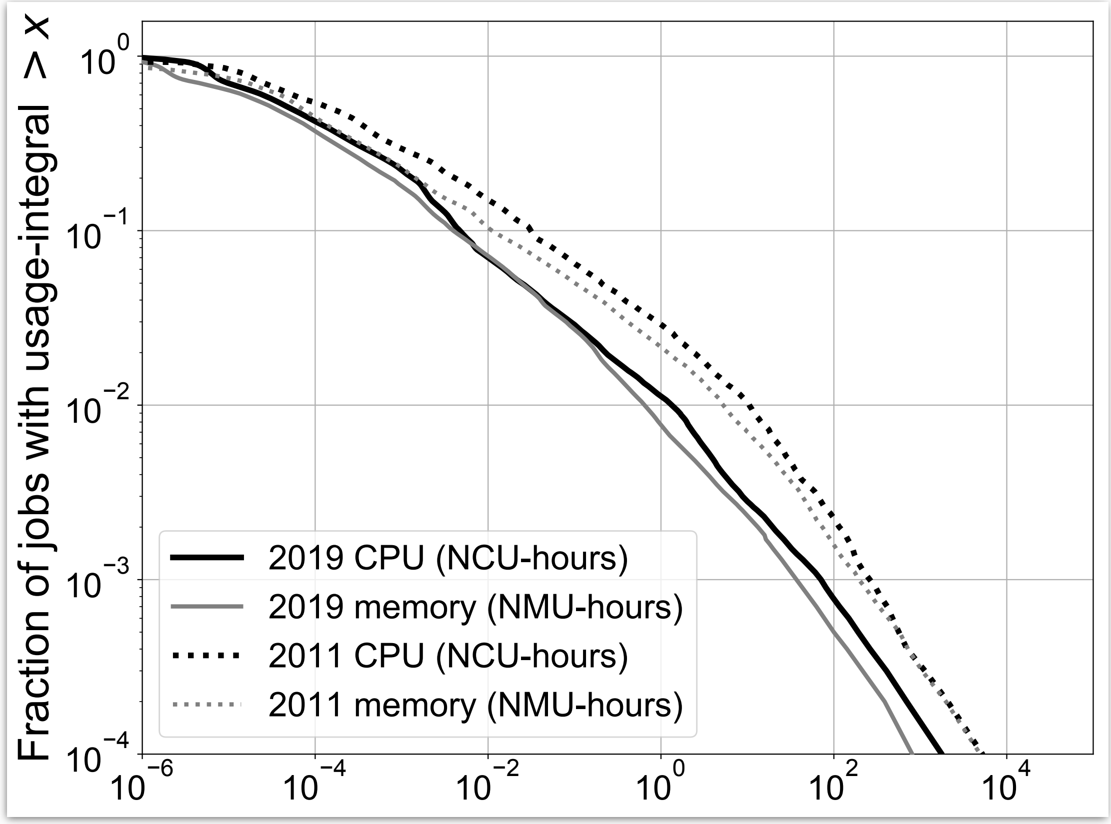
What do jobs look like?



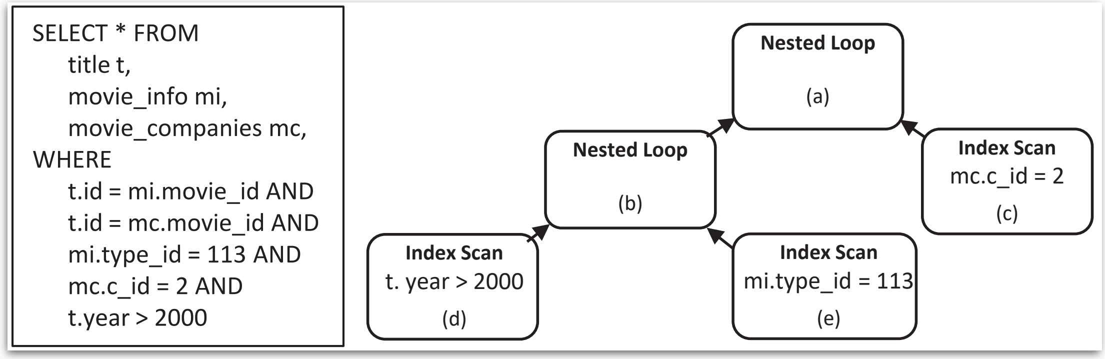
[Tirmazi et al., 2020]

Heavy tails are ubiquitous

What do jobs look like?



[Tirmazi et al., 2020]

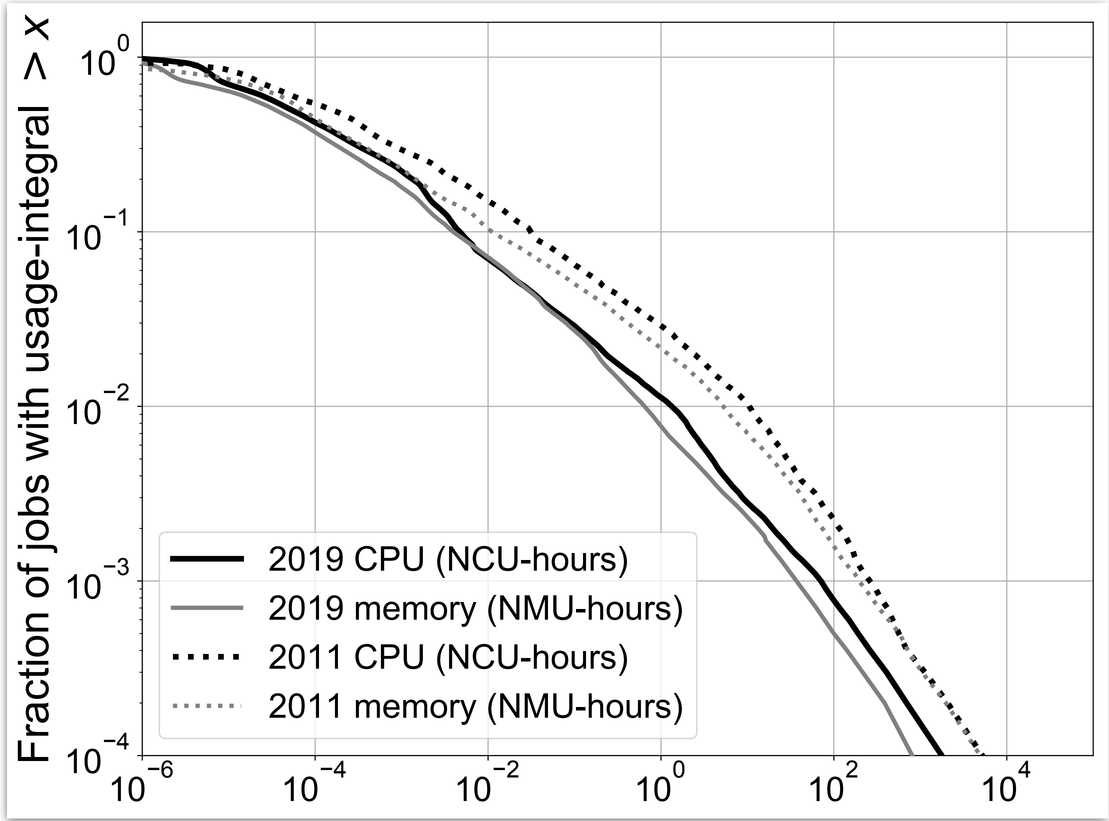


[Zhao et al., 2022]

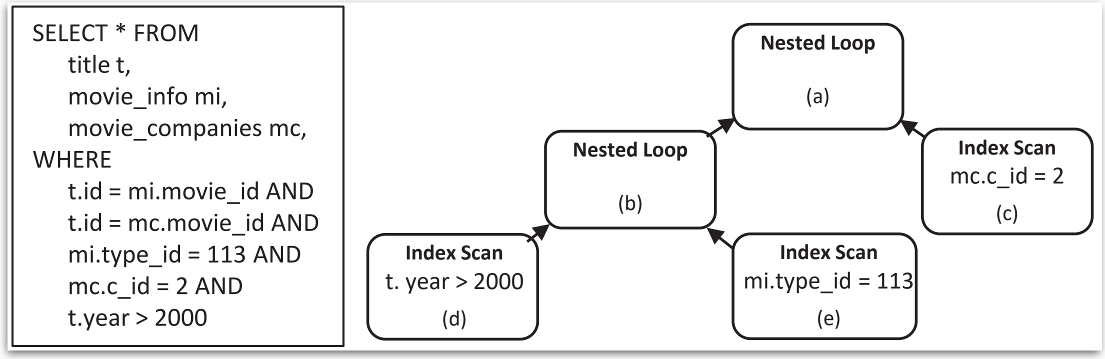
Jobs have complex structure

Heavy tails are ubiquitous

What do jobs look like?



[Tirmazi et al., 2020]



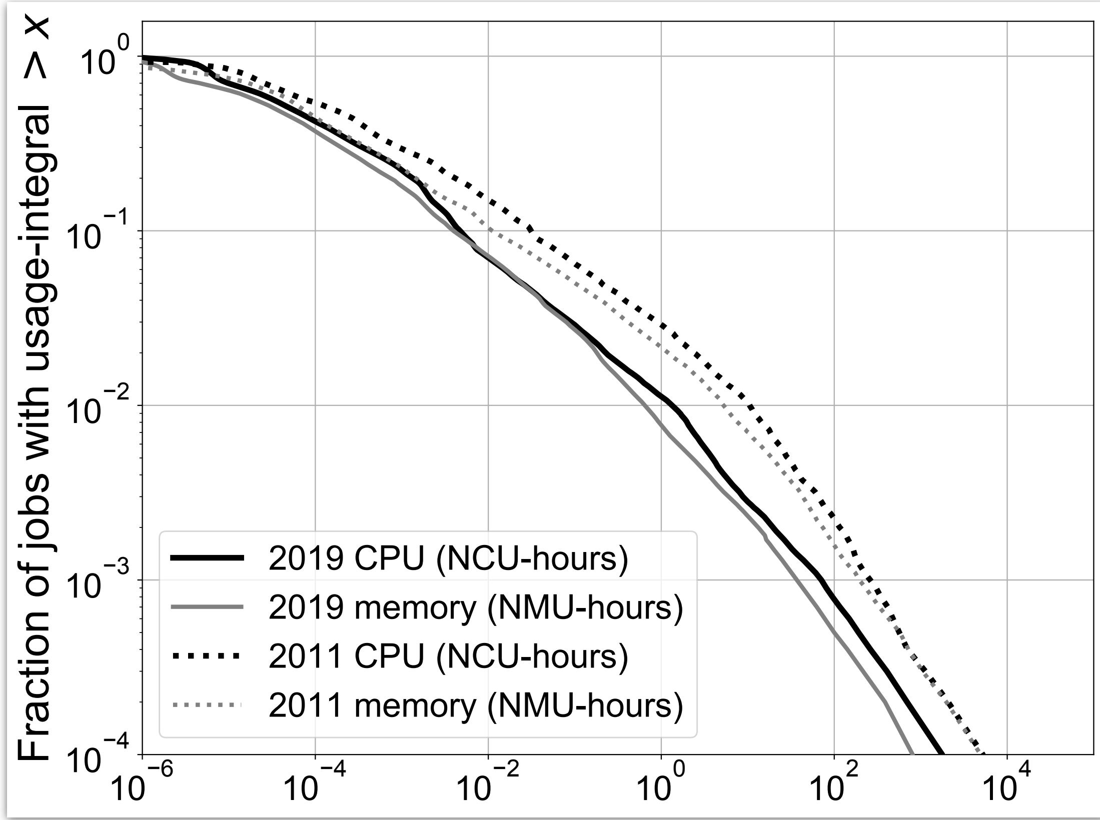
[Zhao et al., 2022]

Jobs have complex structure

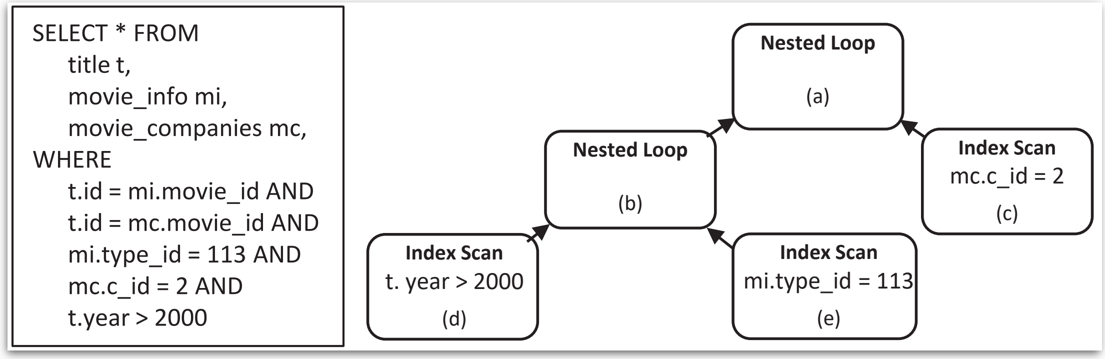
Heavy tails are ubiquitous

Finite-state Markov chains aren't enough

What do jobs look like?



[Tirmazi et al., 2020]



[Zhao et al., 2022]

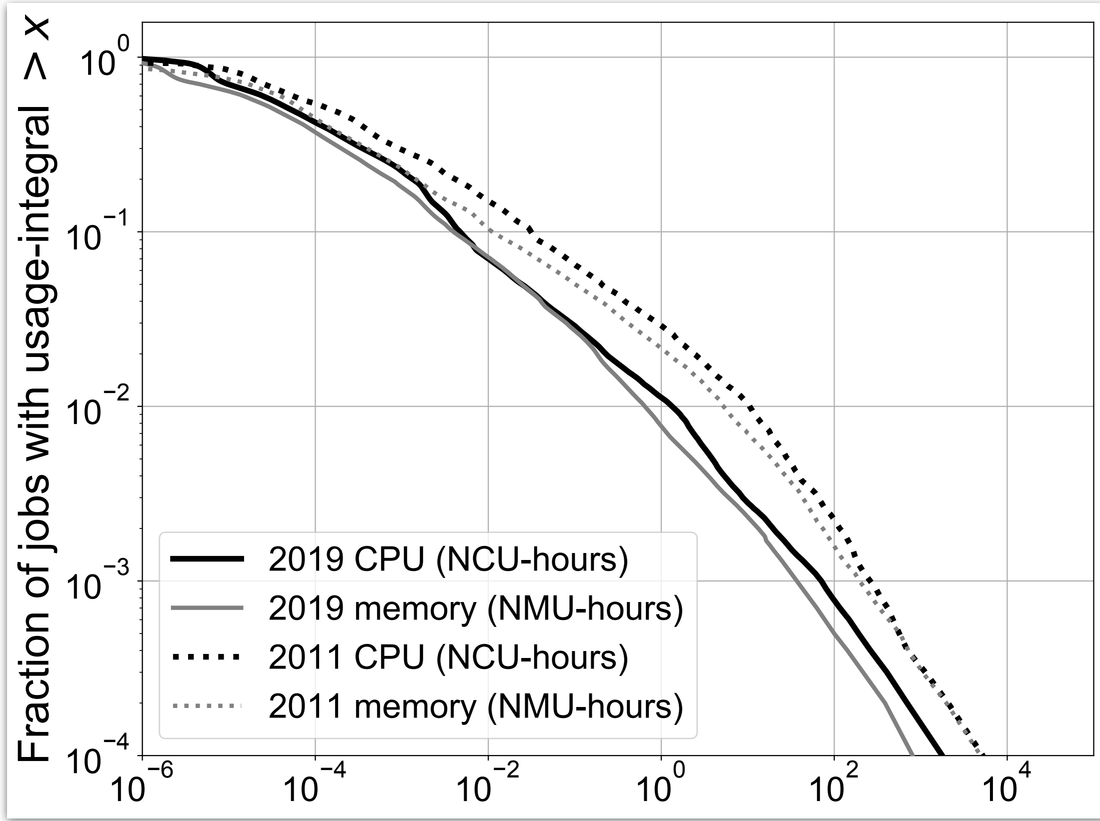
Jobs have complex structure

Age and remaining work aren't enough

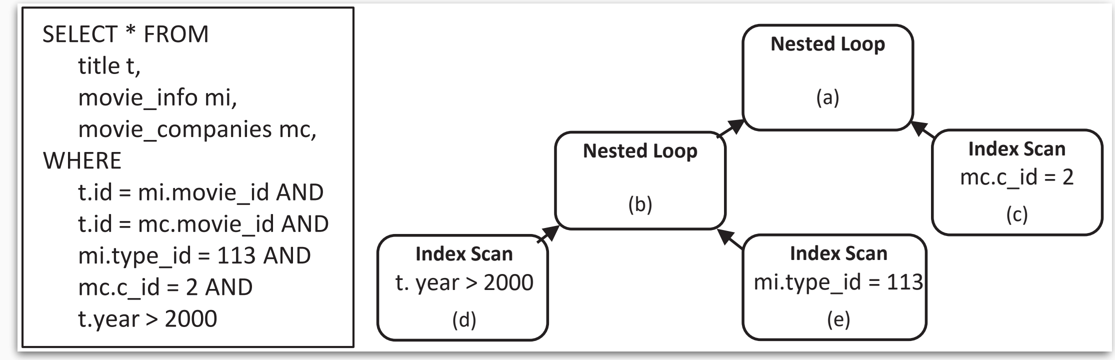
Heavy tails are ubiquitous

Finite-state Markov chains aren't enough

What do jobs look like?



[Tirmazi et al., 2020]



[Zhao et al., 2022]

Heavy tails are ubiquitous

⇓

Finite-state Markov chains aren't enough

Jobs have complex structure

⇓

Age and remaining work aren't enough

Need: general Markov processes

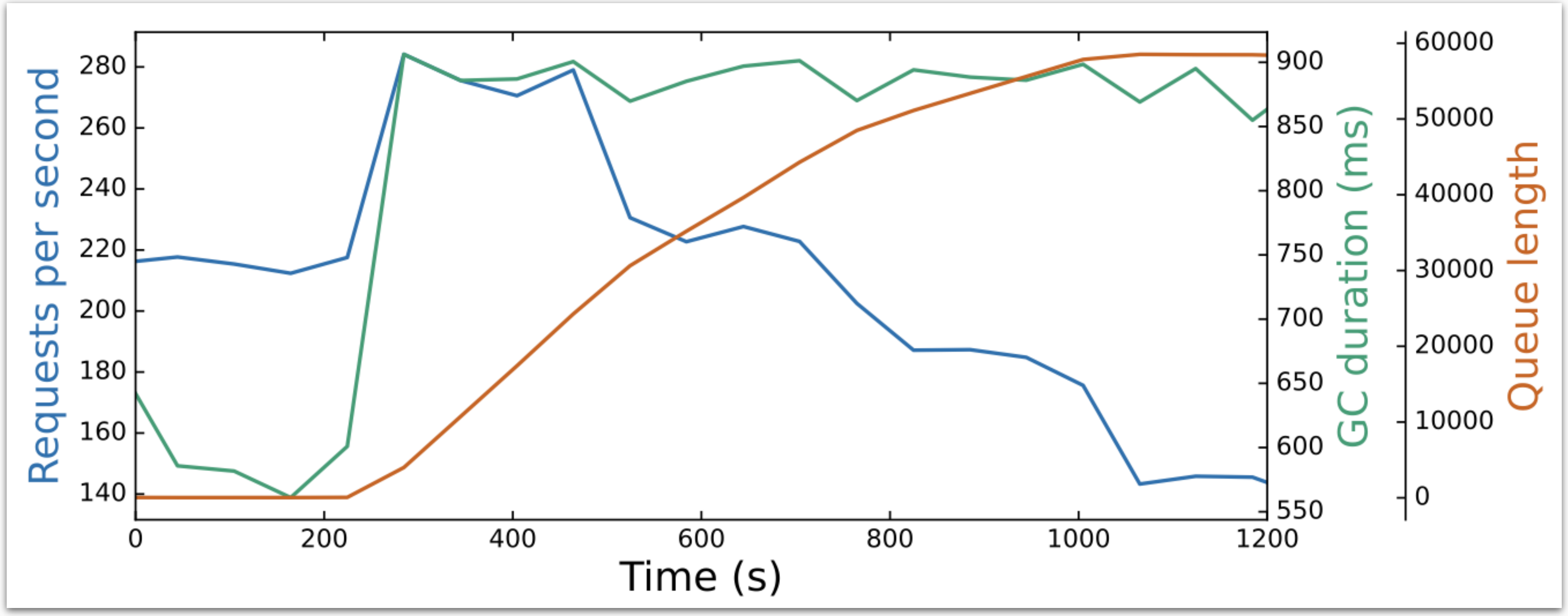
Stability in complex systems

Stability in complex systems

Metastable failures

Stability in complex systems

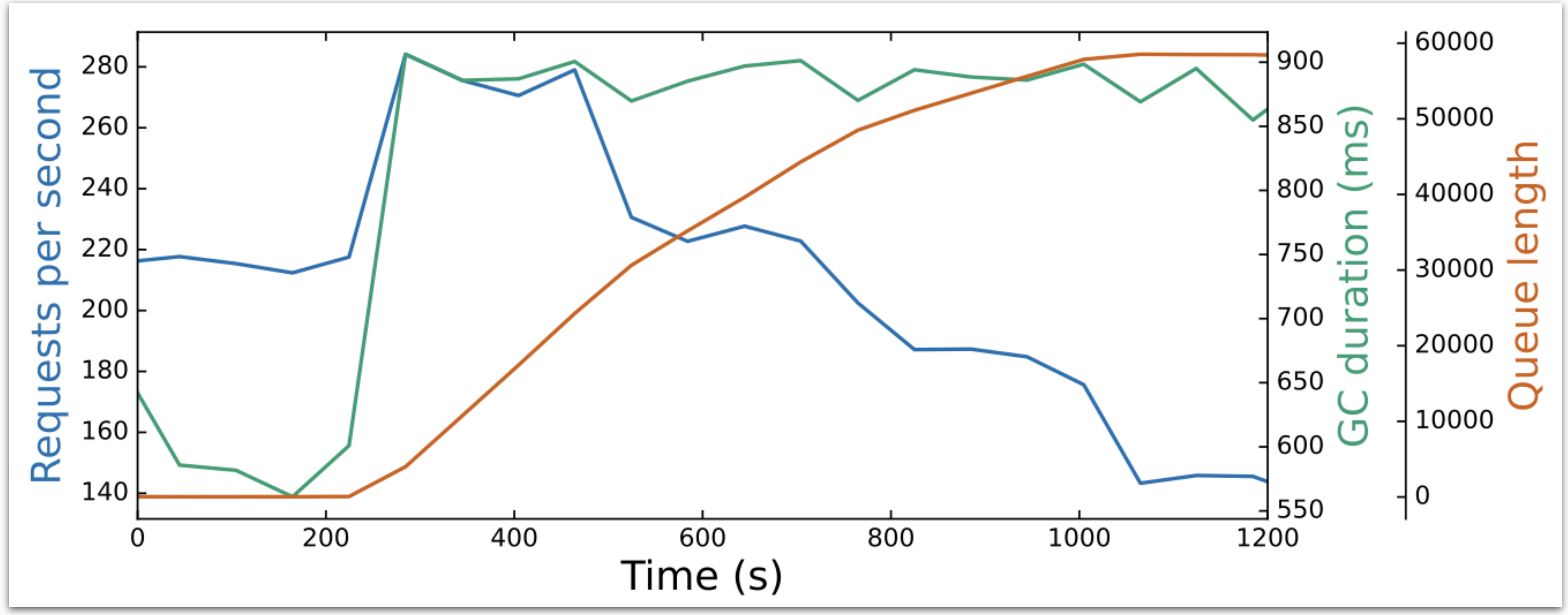
Metastable failures



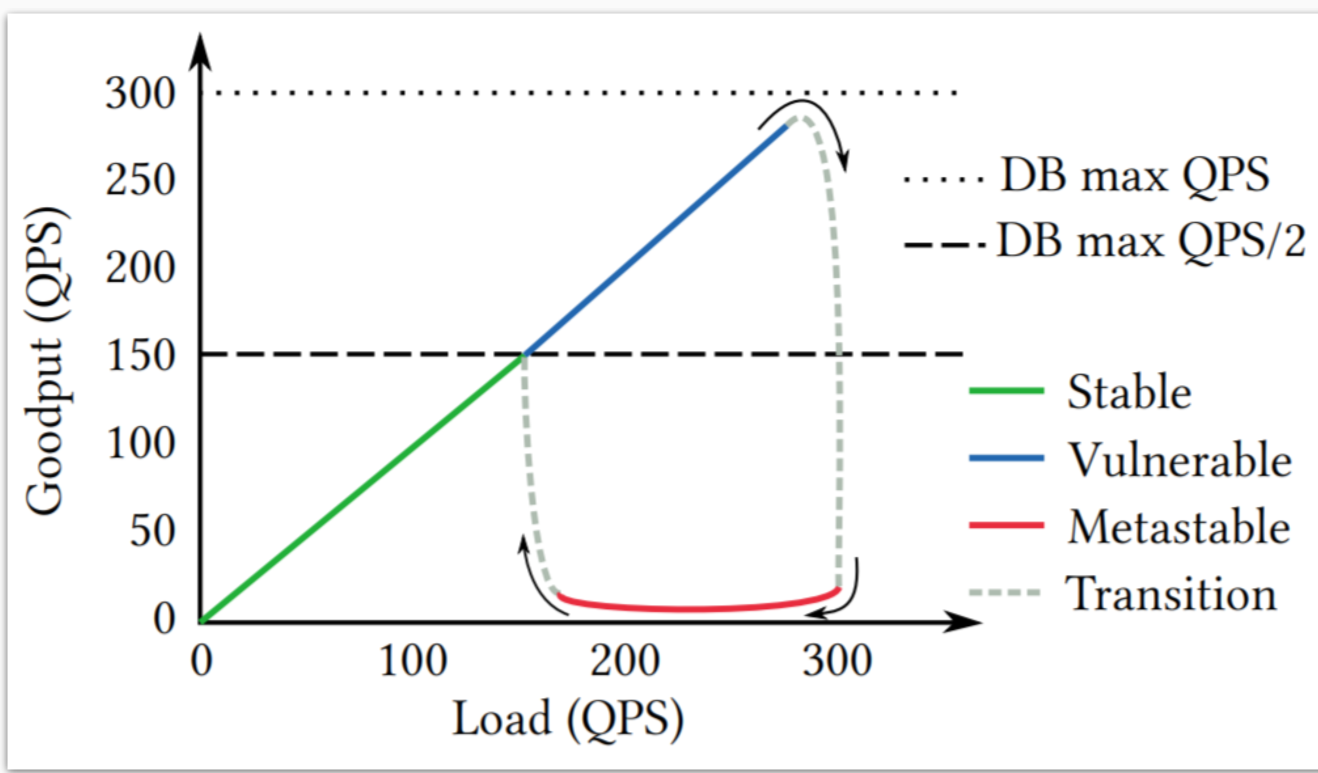
[Huang et al., 2020]

Stability in complex systems

Metastable failures



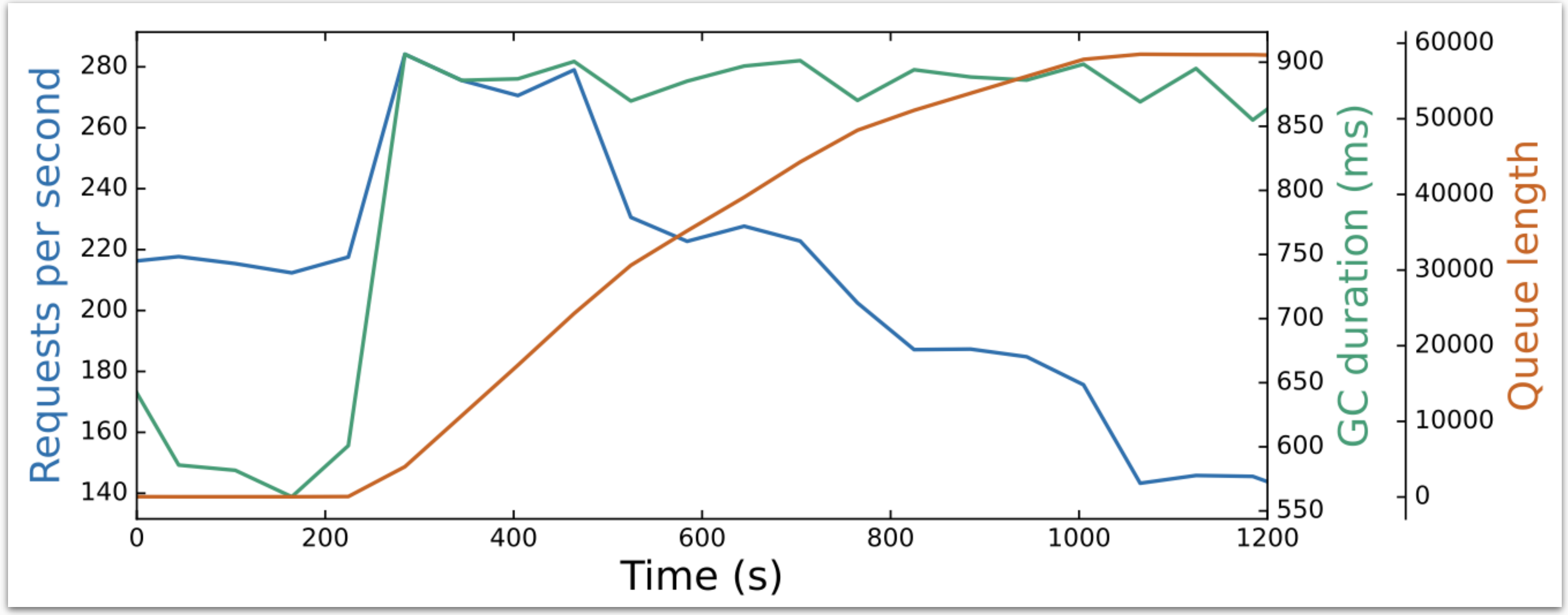
[Huang et al., 2020]



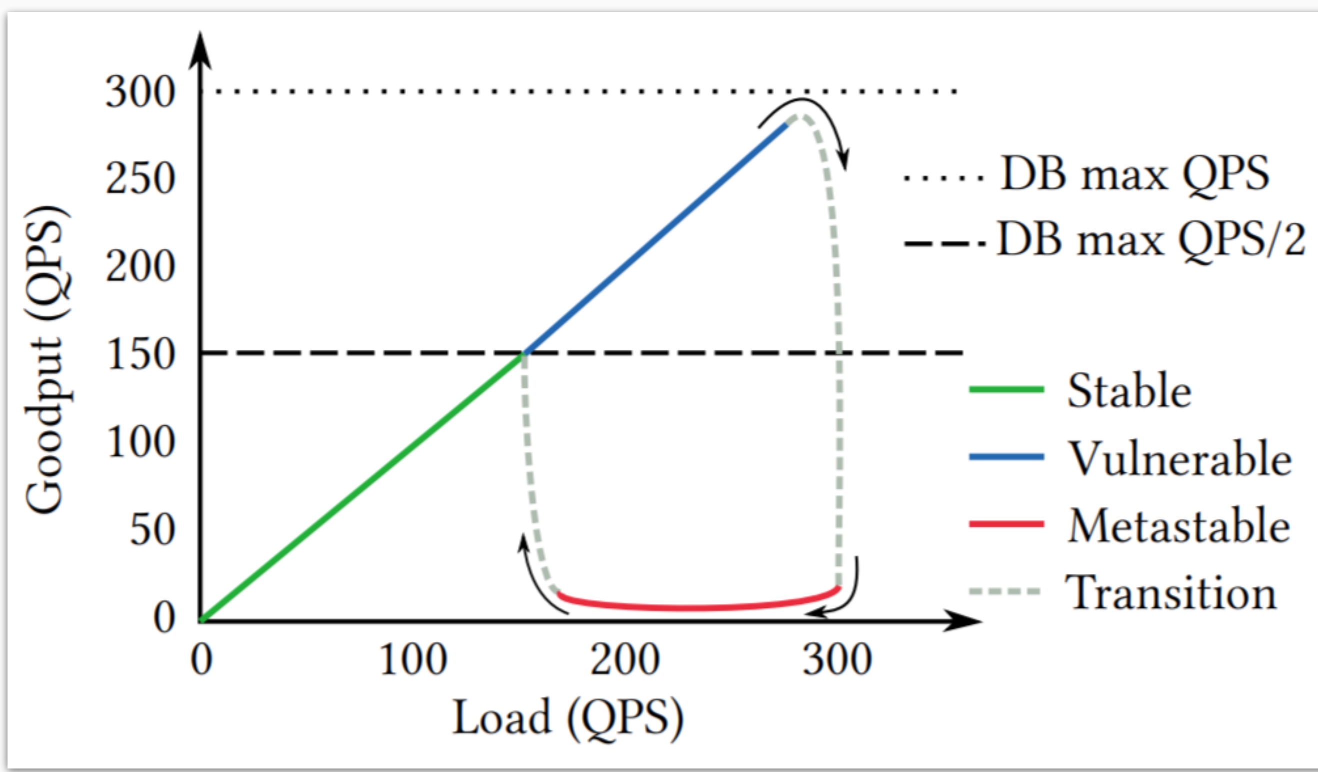
[Bronson et al., 2020]

Stability in complex systems

Metastable failures



[Huang et al., 2020]



[Bronson et al., 2020]

Need: drift methods, mean field methods

Scheduling practicalities

Scheduling practicalities

Theory: SRPT

Scheduling practicalities

SRPT in networks

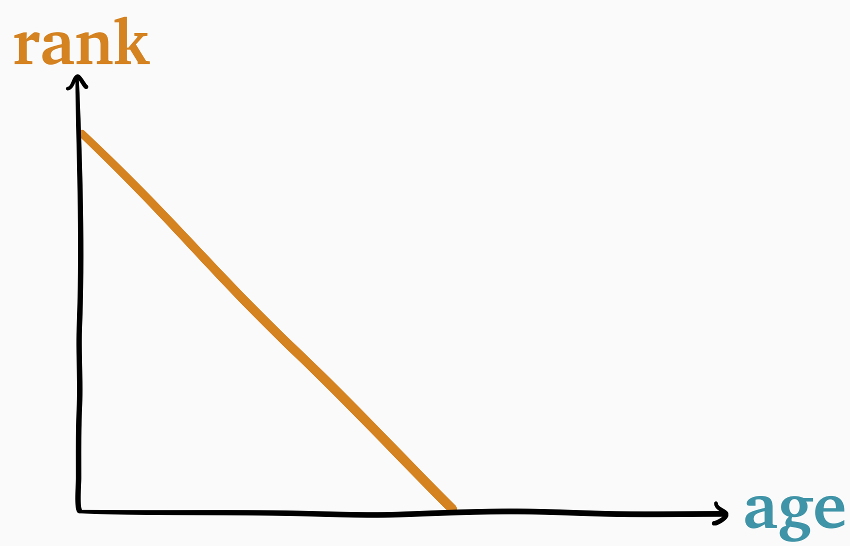
Theory: SRPT

Practice: Homa
[Montazeri et al., 2020]

Scheduling practicalities

SRPT in networks

Theory: SRPT



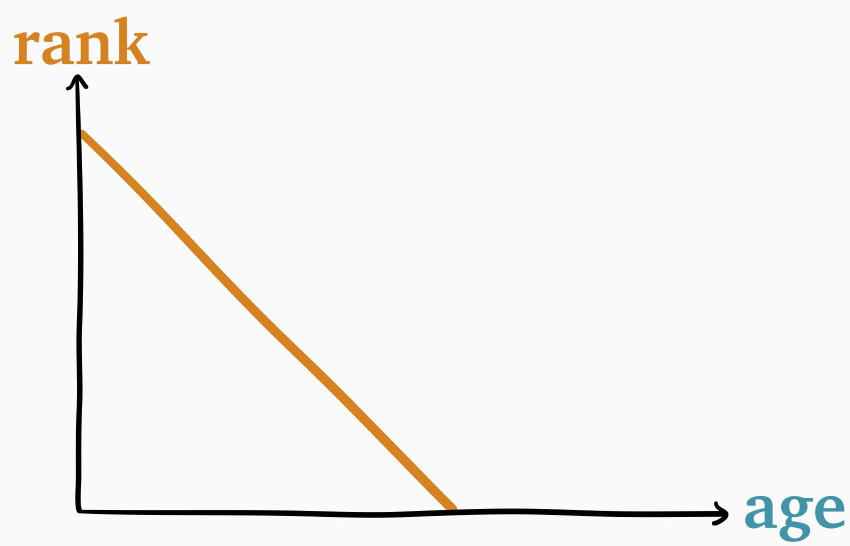
Continuous priority,
no overhead/delay

Practice: Homa
[Montazeri et al., 2020]

Scheduling practicalities

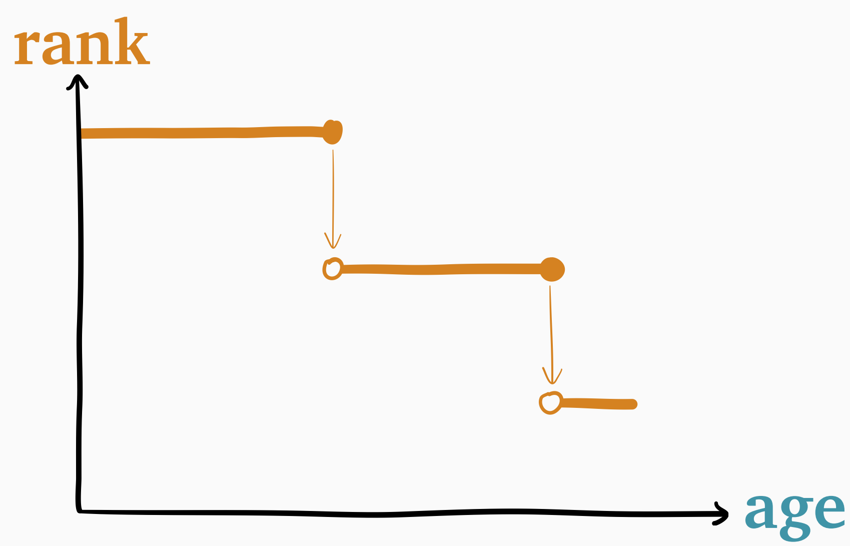
SRPT in networks

Theory: SRPT



Continuous priority,
no overhead/delay

Practice: Homa
[Montazeri et al., 2020]



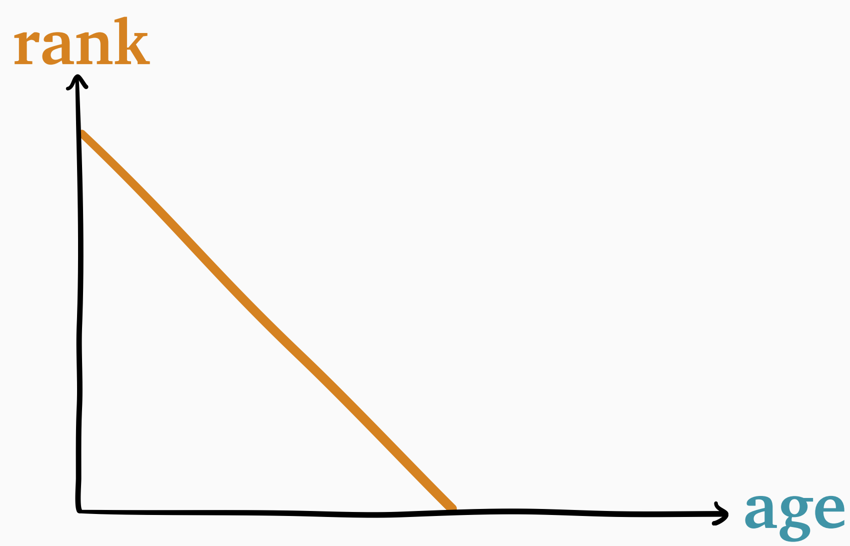
Discrete priorities,
overheads/delays

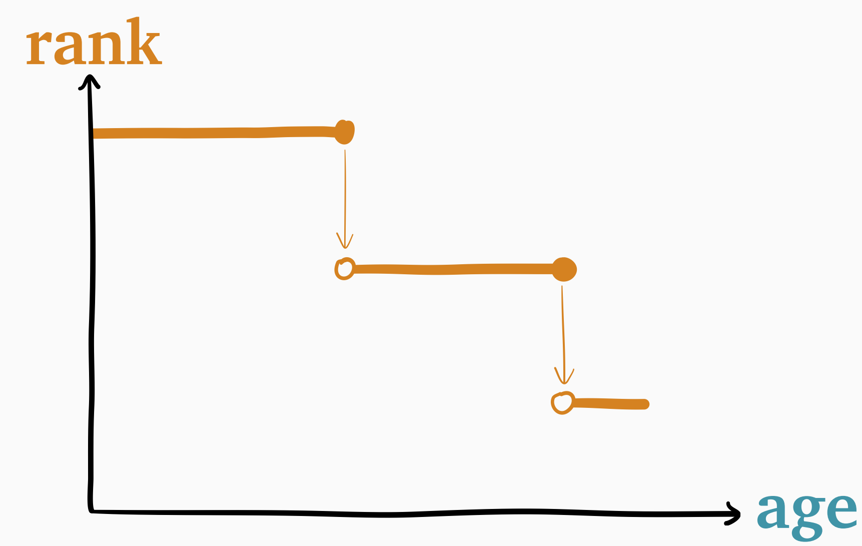
Scheduling practicalities

SRPT in networks

Theory: SRPT

Practice: Homa
[Montazeri et al., 2020]





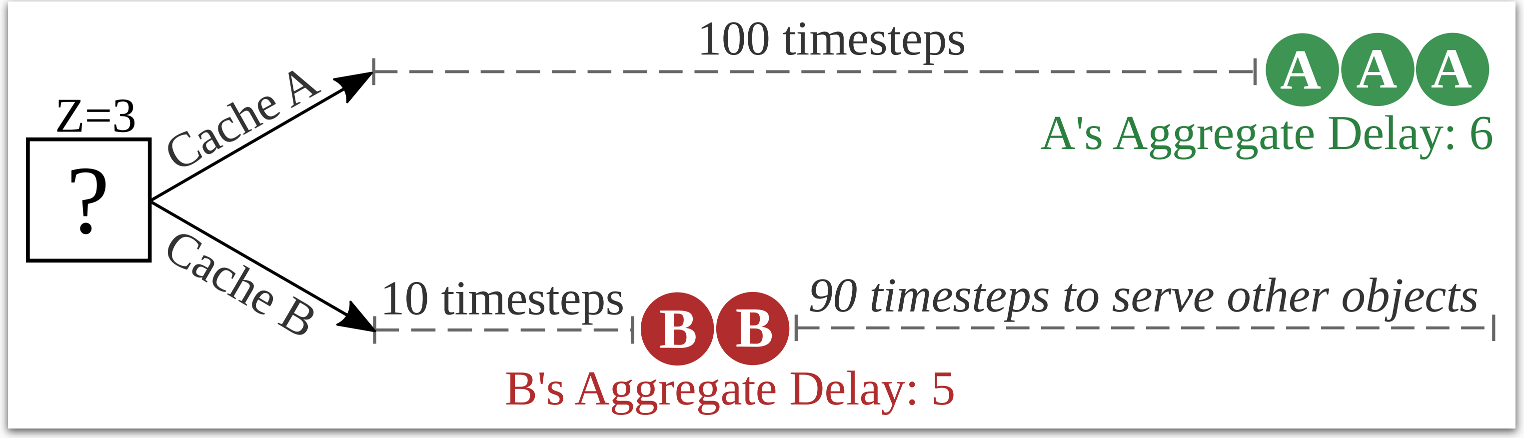
Continuous priority,
no overhead/delay

Discrete priorities,
overheads/delays

Need: analyze variety of scheduling policies

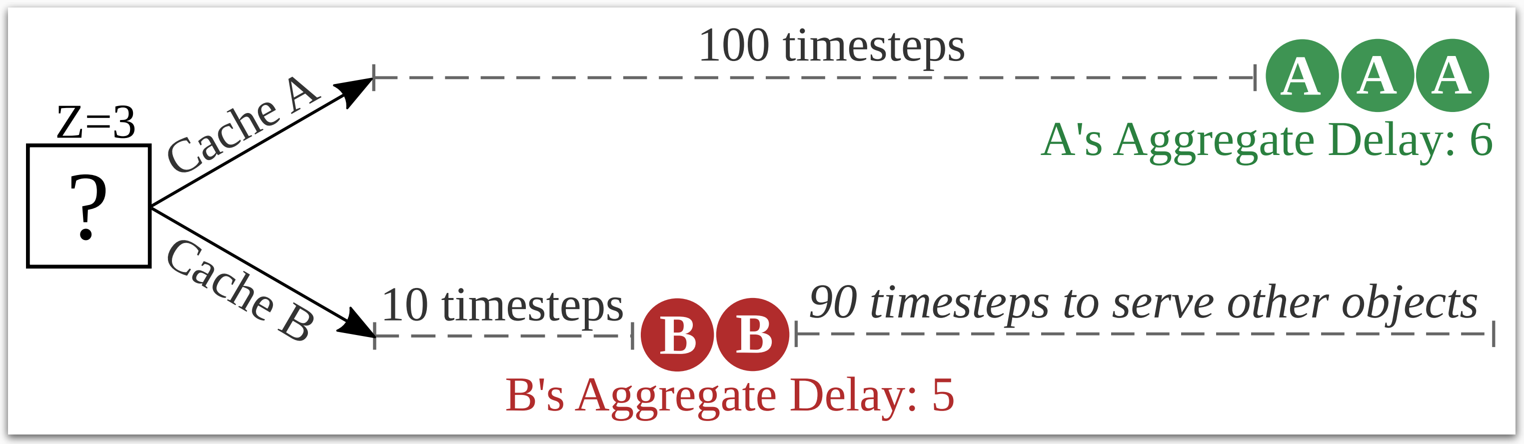
What should we measure?

What should we measure?



[Atre et al., 2020]

What should we measure?



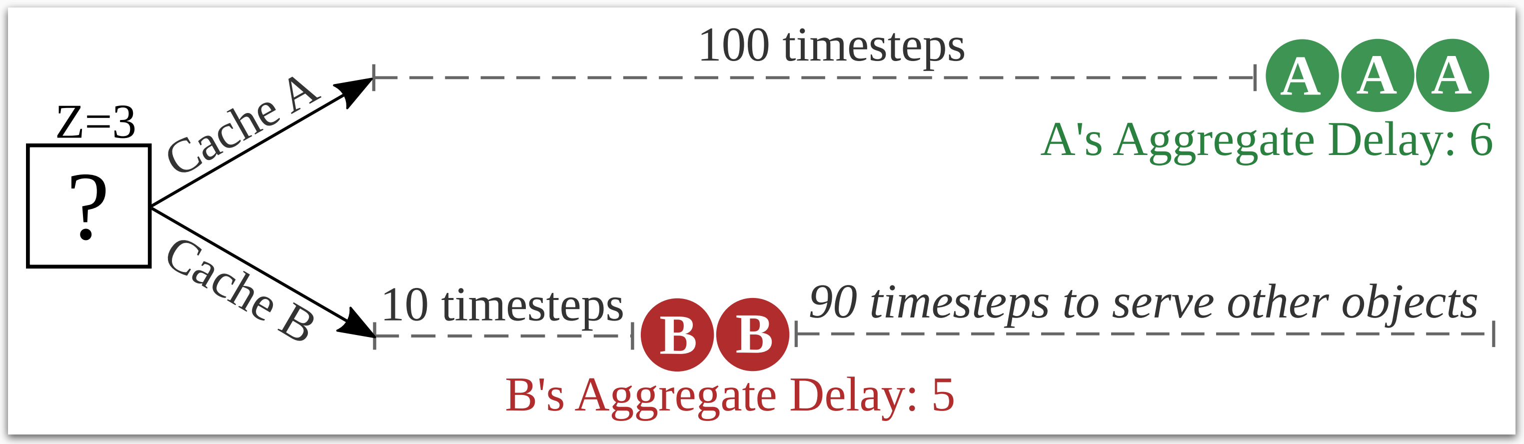
[Atre et al., 2020]

Algorithm 1 Estimating AggregateDelay

```
struct ObjectMetadata
    NumWindows = 0
    CumulativeDelay = 0
    WindowStartIdx = −∞

function EstimateAggregateDelay(X: ObjectMetadata)
    return X.CumulativeDelay / X.NumWindows
end function

function OnAccess(TimeIdx, X: ObjectMetadata)
    // Time since start of the previous miss window
    TSSW = (TimeIdx - X.WindowStartIdx)

    if TSSW ≥ Z then
        // This access commences a new miss window
        X.NumWindows += 1
        X.CumulativeDelay += Z
        X.WindowStartIdx = TimeIdx
    else
        // This access is part of the previous miss window
        X.CumulativeDelay += (Z - TSSW)
    end if
end function
```

[Atre et al., 2020]

What should we measure?



[Atre et al., 2020]

Algorithm 1 Estimating AggregateDelay

```
struct ObjectMetadata
    NumWindows = 0
    CumulativeDelay = 0
    WindowStartIdx = −∞

function EstimateAggregateDelay(X: ObjectMetadata)
    return X.CumulativeDelay / X.NumWindows
end function

function OnAccess(TimeIdx, X: ObjectMetadata)
    // Time since start of the previous miss window
    TSSW = (TimeIdx - X.WindowStartIdx)

    if TSSW ≥ Z then
        // This access commences a new miss window
        X.NumWindows += 1
        X.CumulativeDelay += Z
        X.WindowStartIdx = TimeIdx
    else
        // This access is part of the previous miss window
        X.CumulativeDelay += (Z - TSSW)
    end if
end function
```

[Atre et al., 2020]

Need: expectations from different perspectives

Part 1

Performance modeling needs advanced math

Part 2

We can teach advanced math accessibly

Part 1

Performance modeling needs advanced math

Part 2

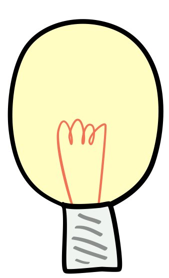
We can teach advanced math accessibly

Part 1

Performance modeling needs advanced math

Part 2

We can teach advanced math accessibly



Simplify core foundations

Part 1

Performance modeling needs advanced math

Part 2

We can teach advanced math accessibly

Simplify core foundations

Prioritize very flexible tools

⚠️ **Problem:** many students lack math background

Problem: many students lack math background

Solution: hand-wave

Problem: many students lack math background

Solution: hand-wave

Problem: how to know when to hand-wave?

Problem: many students lack math background

Solution: hand-wave

Problem: how to know when to hand-wave?

Solution: clear rules for hand-waving

Problem: many students lack math background

Solution: hand-wave

Problem: how to know when to hand-wave?

Solution: clear rules for hand-waving

- *Principles:* rules that work most of the time

Problem: many students lack math background

Solution: hand-wave

Problem: how to know when to hand-wave?

Solution: clear rules for hand-waving

- *Principles:* rules that work most of the time
- *Recipes:* common patterns for using principles

Problem: many students lack math background

Solution: hand-wave

Problem: how to know when to hand-wave?

Solution: clear rules for hand-waving

- *Principles:* rules that work most of the time
- *Recipes:* common patterns for using principles

Problem: each topic needs many principles

Problem: many students lack math background

Solution: hand-wave

Problem: how to know when to hand-wave?

Solution: clear rules for hand-waving

- *Principles:* rules that work most of the time
- *Recipes:* common patterns for using principles

Problem: each topic needs many principles

Solution: focus on a few very powerful topics

Proposed toolbox

Proposed toolbox

Description: model with *Markov processes*

Proposed toolbox

Description: model with *Markov processes*

Metrics: define using *long-run averages*

Proposed toolbox

Description: model with *Markov processes*

Metrics: define using *long-run averages*

Analysis: reduce to questions about *drift*

Description via Markov processes

Description via Markov processes

State: all info we need to describe evolution

Description via Markov processes

State: all info we need to describe evolution

current state

Description via Markov processes

State: all info we need to describe evolution

deterministic flow

current state

Description via Markov processes

State: all info we need to describe evolution

deterministic flow

current state

stochastic jumps

Description via Markov processes

State: all info we need to describe evolution

deterministic flow

current state

future states

stochastic jumps

Description via Markov processes

State: all info we need to describe evolution

current state

deterministic flow

stochastic jumps

future states

Goal: clear process definition

Description via Markov processes

State: all info we need to describe evolution

deterministic flow

current state future states

stochastic jumps

Goal: clear process definition

Non-goal (yet): tractable analysis

Description via Markov processes

State: all info we need to describe evolution

deterministic flow

current state　　　　　　　　　　future states

stochastic jumps

Goal: clear process definition

Non-goal (yet): tractable analysis

Non-goal: verifying Markov property

Example: M/G/1

Example: M/G/1

State: list with remaining work of each job

$$[r_1, \ldots, r_n]$$

Example: M/G/1

State: list with remaining work of each job

$$[r_1, \ldots, r_n]$$

Dynamics:

Example: M/G/1

State: list with remaining work of each job

$$[r_1, \ldots, r_n]$$

Dynamics:

- If list nonempty: decrease r_1 at rate 1

Example: M/G/1

State: list with remaining work of each job

$$[r_1, \ldots, r_n]$$

Dynamics:

- If list nonempty: decrease r_1 at rate 1
- When $r_1 = 0$: remove r_1 from list

Example: M/G/1

State: list with remaining work of each job

$$[r_1, \ldots, r_n]$$

Dynamics:

- If list nonempty: decrease r_1 at rate 1
- When $r_1 = 0$: remove r_1 from list
- Poisson(λ): draw from S, append it to list

Example: M/G/1

State: list with remaining work of each job

$$[r_1, \ldots, r_n]$$

Dynamics:

- If list nonempty: decrease r_1 at rate 1
- When $r_1 = 0$: remove r_1 from list
- Poisson(λ): draw from S, append it to list

$$\textit{Work:} \quad w([r_1, \ldots, r_n]) = r_1 + \cdots + r_n$$

Example: M/G/1

State: list with remaining work of each job

$$[r_1, \ldots, r_n]$$

Dynamics:

- If list nonempty: decrease r_1 at rate 1
- When $r_1 = 0$: remove r_1 from list
- Poisson(λ): draw from S, append it to list

$$\textit{Work:} \quad w([r_1, \ldots, r_n]) = r_1 + \cdots + r_n$$

$$\textit{Queue length:} \quad q([r_1, \ldots, r_n]) = (n-1)^+$$

Metrics via long-run averages

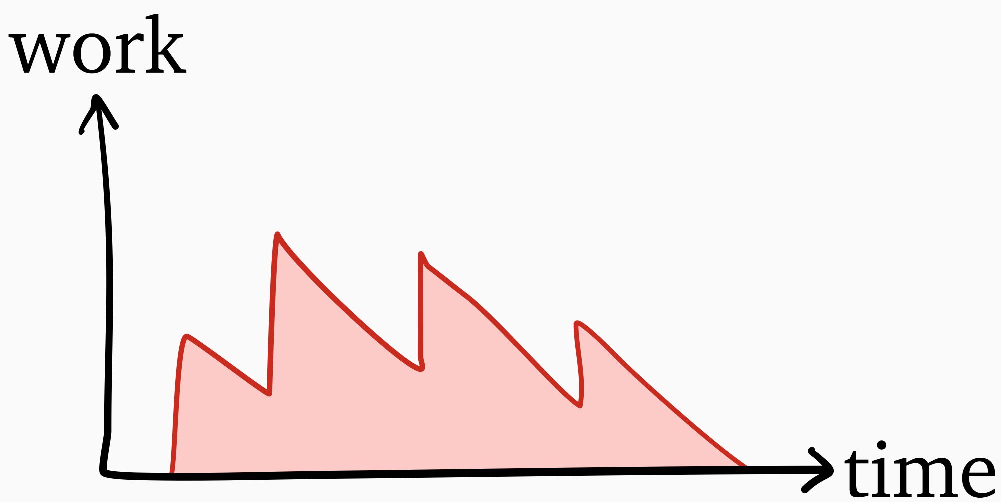
$$\begin{aligned} X(t) &= \text{state at time } t \\ \text{mean waiting time} &= \mathbf{E}_{\text{arrival}}[w(X)] \\ \text{mean number in queue} &= \mathbf{E}_{\text{time}}[q(X)] \end{aligned}$$

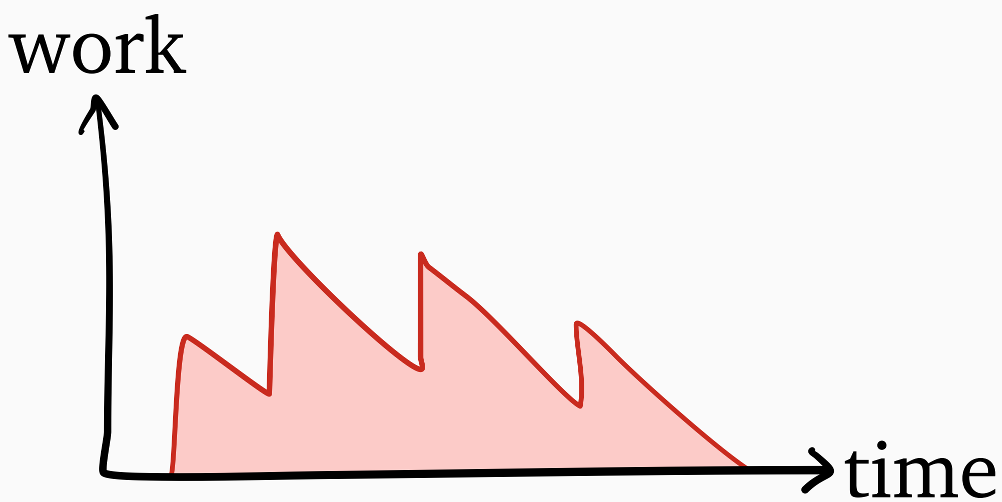
Metrics via long-run averages

$$X(t) = \text{state at time } t$$

$$\text{mean waiting time} = \mathbf{E}_{\text{arrival}}[w(X)]$$

$$\text{mean number in queue} = \mathbf{E}_{\text{time}}[q(X)]$$



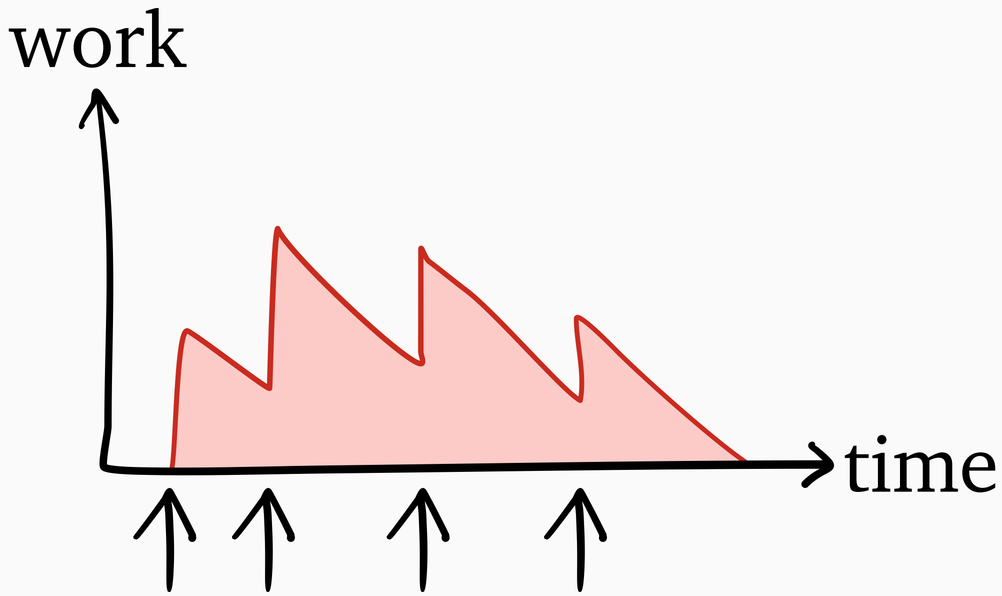


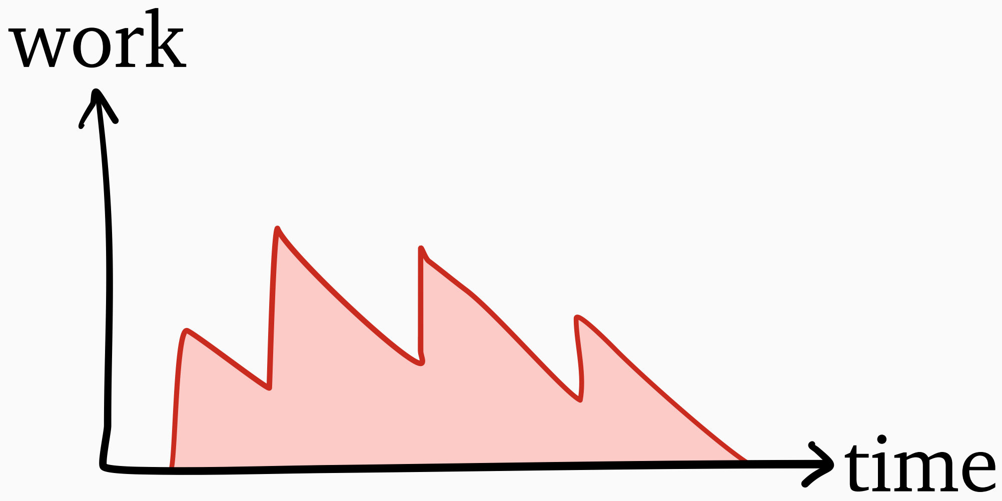
Metrics via long-run averages

$$X(t) = \text{state at time } t$$

$$\text{mean waiting time} = \mathbf{E}_{\text{arrival}}[w(X)]$$

$$\text{mean number in queue} = \mathbf{E}_{\text{time}}[q(X)]$$

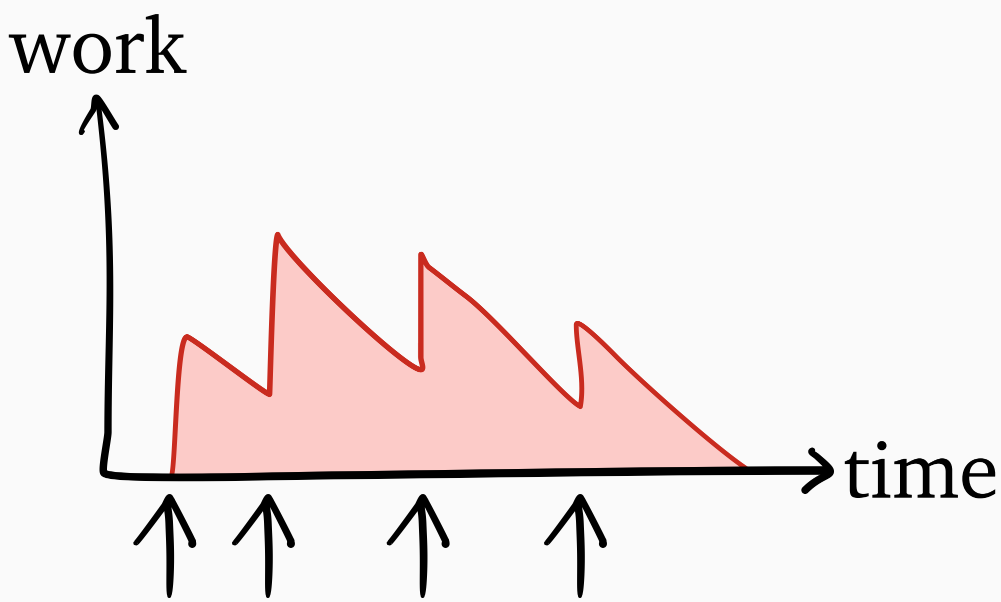


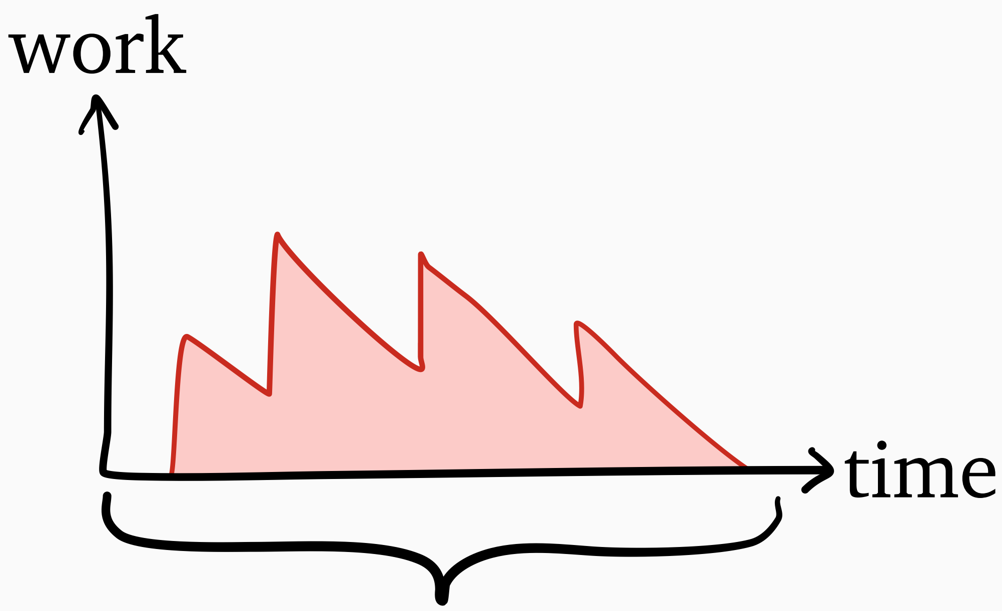


$$\mathbf{E}_{\text{arrival}}[f(X)] = \frac{\sum_{t \text{ arrival}} f(X(t))}{\#\ \text{arrivals}}$$

Metrics via long-run averages

$$X(t) = \text{state at time } t$$
$$\text{mean waiting time} = \mathbf{E}_{\text{arrival}}[w(X)]$$
$$\text{mean number in queue} = \mathbf{E}_{\text{time}}[q(X)]$$





$$\mathbf{E}_{\text{arrival}}[f(X)] = \frac{\sum_{t \text{ arrival}} f(X(t))}{\#\text{ arrivals}}$$

$$\mathbf{E}_{\text{time}}[f(X)] = \frac{\int_0^{\text{long time}} f(X(t))\,\mathrm{d}t}{\text{long time}}$$

Principles for long-run averages

Principles for long-run averages

Base principle: when averaging over entire timeline,

ignore edge effects

Principles for long-run averages

Base principle: when averaging over entire timeline,

ignore edge effects

Little's law

Principles for long-run averages

Base principle: when averaging over entire timeline,

ignore edge effects

Little's law   Renewal-reward

Principles for long-run averages

Base principle: when averaging over entire timeline,

ignore edge effects

Little's law　　Renewal-reward　　Palm inversion

Principles for long-run averages

Base principle: when averaging over entire timeline,

ignore edge effects

Little's law　　Renewal-reward　　Palm inversion

Rate conservation law:

for any f, average rate of change in $f(X)$ is 0

Principles for long-run averages

Base principle: when averaging over entire timeline,

ignore edge effects

requires stability!

Little's law　　Renewal-reward　　Palm inversion

Rate conservation law:

for any f, average rate of change in $f(X)$ is 0

Analysis via drift

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $= 0$: remove head of list
- Poisson(λ): draw from S, append it to list

Analysis via drift

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $= 0$: remove head of list
- Poisson(λ): draw from S, append it to list

$\Big\Downarrow$ **RCL**

$$
\begin{aligned}
0 = {} & \mathbf{E}_{\text{time}}\Big[\frac{\partial}{\partial r_1} f(X)\Big] \\
& + \lambda \mathbf{E}_{\text{departure}}\big[f(\text{tail}(X)) - f(X)\big] \\
& + \lambda \mathbf{E}_{\text{arrival}}\big[f(\text{join}(X, [S])) - f(X)\big]
\end{aligned}
$$

Analysis via drift

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $= 0$: remove head of list
- Poisson(λ): draw from S, append it to list

$$\Big\Downarrow \textbf{RCL}$$

$$\begin{aligned} 0 = \mathbf{E}_{\text{time}}\left[\frac{\partial}{\partial r_1} f(X)\right] \\ + \lambda \mathbf{E}_{\text{departure}}\left[f(\text{tail}(X)) - f(X)\right] \\ + \lambda \mathbf{E}_{\text{arrival}}\left[f(\text{join}(X, [S])) - f(X)\right] \end{aligned}$$

Analysis via drift

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $= 0$: remove head of list
- Poisson(λ): draw from S, append it to list

RCL

$$
\begin{aligned}
0 = \mathbf{E}_{\text{time}}&\left[\frac{\partial}{\partial r_1} f(X)\right] \\
&+ \lambda \mathbf{E}_{\text{departure}}\left[f(\text{tail}(X)) - f(X)\right] \\
&+ \lambda \mathbf{E}_{\text{arrival}}\left[f(\text{join}(X, [S])) - f(X)\right]
\end{aligned}
$$

Analysis via drift

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $= 0$: remove head of list
- Poisson(λ): draw from S, append it to list

RCL

$$
\begin{aligned}
0 = \mathbf{E}_{\text{time}}&\left[\frac{\partial}{\partial r_1} f(X)\right] \\
&+ \lambda \mathbf{E}_{\text{departure}}\left[f(\text{tail}(X)) - f(X)\right] \\
&+ \lambda \mathbf{E}_{\text{arrival}}\left[f(\text{join}(X, [S])) - f(X)\right]
\end{aligned}
$$

Analysis via drift

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $= 0$: remove head of list
- Poisson(λ): draw from S, append it to list

$$f(x) = w(x)$$

RCL

$$\begin{aligned} 0 = \mathbf{E}_{\text{time}}&\left[\frac{\partial}{\partial r_1} f(X)\right] \\ + \lambda \mathbf{E}_{\text{departure}}&\left[f(\text{tail}(X)) - f(X)\right] \\ + \lambda \mathbf{E}_{\text{arrival}}&\left[f(\text{join}(X, [S])) - f(X)\right] \end{aligned}$$

Analysis via drift

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $=0$: remove head of list
- Poisson(λ): draw from S, append it to list

RCL

$$0 = \mathbf{E}_{\text{time}}\left[\frac{\partial}{\partial r_1} f(X)\right] + \lambda \mathbf{E}_{\text{departure}}\left[f(\text{tail}(X)) - f(X)\right] + \lambda \mathbf{E}_{\text{arrival}}\left[f(\text{join}(X, [S])) - f(X)\right]$$

$$f(x) = w(x)$$

$$\mathbf{P}_{\text{time}}[X \text{ empty}] = 1 - \lambda \mathbf{E}[S]$$

Analysis via drift

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $=0$: remove head of list
- Poisson(λ): draw from S, append it to list

$$\Big\Downarrow \text{RCL}$$

$$0 = \mathbf{E}_{\text{time}}\left[\frac{\partial}{\partial r_1} f(X)\right]$$
$$+ \lambda \mathbf{E}_{\text{departure}}\left[f(\text{tail}(X)) - f(X)\right]$$
$$+ \lambda \mathbf{E}_{\text{arrival}}\left[f(\text{join}(X, [S])) - f(X)\right]$$

$$f(x) = w(x)$$

$$\Big\Downarrow$$

$$\mathbf{P}_{\text{time}}[X \text{ empty}] = 1 - \lambda \mathbf{E}[S]$$

$$f(x) = w(x)^2$$

Analysis via drift

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $= 0$: remove head of list
- Poisson(λ): draw from S, append it to list

RCL

$$0 = \mathbf{E}_{\text{time}}\left[\frac{\partial}{\partial r_1} f(X)\right] + \lambda \mathbf{E}_{\text{departure}}\left[f(\text{tail}(X)) - f(X)\right] + \lambda \mathbf{E}_{\text{arrival}}\left[f(\text{join}(X,[S])) - f(X)\right]$$

$$f(x) = w(x)$$

$$\mathbf{P}_{\text{time}}[X \text{ empty}] = 1 - \lambda \mathbf{E}[S]$$

$$f(x) = w(x)^2$$

$$\mathbf{E}_{\text{time}}[w(X)] = \frac{\frac{\lambda}{2}\mathbf{E}[S^2]}{1 - \lambda \mathbf{E}[S]}$$

Analysis via drift

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $=0$: remove head of list
- Poisson(λ): draw from S, append it to list

RCL

$$0 = \mathbf{E}_{\text{time}}\left[\frac{\partial}{\partial r_1} f(X)\right] + \lambda \mathbf{E}_{\text{departure}}\left[f(\text{tail}(X)) - f(X)\right] + \lambda \mathbf{E}_{\text{arrival}}\left[f(\text{join}(X,[S])) - f(X)\right]$$

$$f(x) = w(x)$$

$$\mathbf{P}_{\text{time}}[X \text{ empty}] = 1 - \lambda \mathbf{E}[S]$$

$$f(x) = w(x)^2$$

$$\mathbf{E}_{\text{time}}[w(X)] = \frac{\frac{\lambda}{2}\mathbf{E}[S^2]}{1 - \lambda \mathbf{E}[S]}$$

Principle: PASTA
$\mathbf{E}_{\text{arrival}}[\cdot] = \mathbf{E}_{\text{time}}[\cdot]$

Analysis via drift

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $= 0$: remove head of list
- Poisson(λ): draw from S, append it to list

RCL

$$
\begin{aligned}
0 = {} & \mathbf{E}_{\text{time}}\left[\frac{\partial}{\partial r_1} f(X)\right] \\
& + \lambda \mathbf{E}_{\text{departure}}\left[f(\text{tail}(X)) - f(X)\right] \\
& + \lambda \mathbf{E}_{\text{arrival}}\left[f(\text{join}(X, [S])) - f(X)\right]
\end{aligned}
$$

Recipe: to get nth-order info, use $(n+1)$th-order function f

$$f(x) = w(x)$$

$$\mathbf{P}_{\text{time}}[X \text{ empty}] = 1 - \lambda \mathbf{E}[S]$$

$$f(x) = w(x)^2$$

$$\mathbf{E}_{\text{time}}[w(X)] = \frac{\frac{\lambda}{2}\mathbf{E}[S^2]}{1 - \lambda \mathbf{E}[S]}$$

Principle: PASTA
$\mathbf{E}_{\text{arrival}}[\cdot] = \mathbf{E}_{\text{time}}[\cdot]$

Beyond the M/G/1

Beyond the M/G/1

Work decomposition law: under M/G arrivals,

$$\mathbf{E}_{\text{time}}[w(X)] = \frac{\frac{\lambda}{2}\mathbf{E}[S^2] + \mathbf{E}_{\text{time}}[u(X)\,w(X)]}{1 - \lambda\mathbf{E}[S]}$$

Beyond the M/G/1

Work decomposition law: under M/G arrivals,

$$\mathbf{E}_{\text{time}}[w(X)] = \frac{\frac{\lambda}{2}\mathbf{E}[S^2] + \mathbf{E}_{\text{time}}[u(X)\,w(X)]}{1 - \lambda\mathbf{E}[S]}$$

unused
service

Beyond the M/G/1

Work decomposition law: under M/G arrivals,

$$\mathbf{E}_{\text{time}}[w(X)] = \frac{\frac{\lambda}{2}\mathbf{E}[S^2] + \mathbf{E}_{\text{time}}[u(X)\,w(X)]}{1 - \lambda\mathbf{E}[S]}$$

unused
service

M/G/1

$$\mathbf{E}_{\text{time}}[u(X)\,w(X)] = 0$$

Beyond the M/G/1

Work decomposition law: under M/G arrivals,

$$\mathbf{E}_{\text{time}}[w(X)] = \frac{\frac{\lambda}{2}\mathbf{E}[S^2] + \mathbf{E}_{\text{time}}[u(X)\,w(X)]}{1 - \lambda\mathbf{E}[S]}$$

unused service

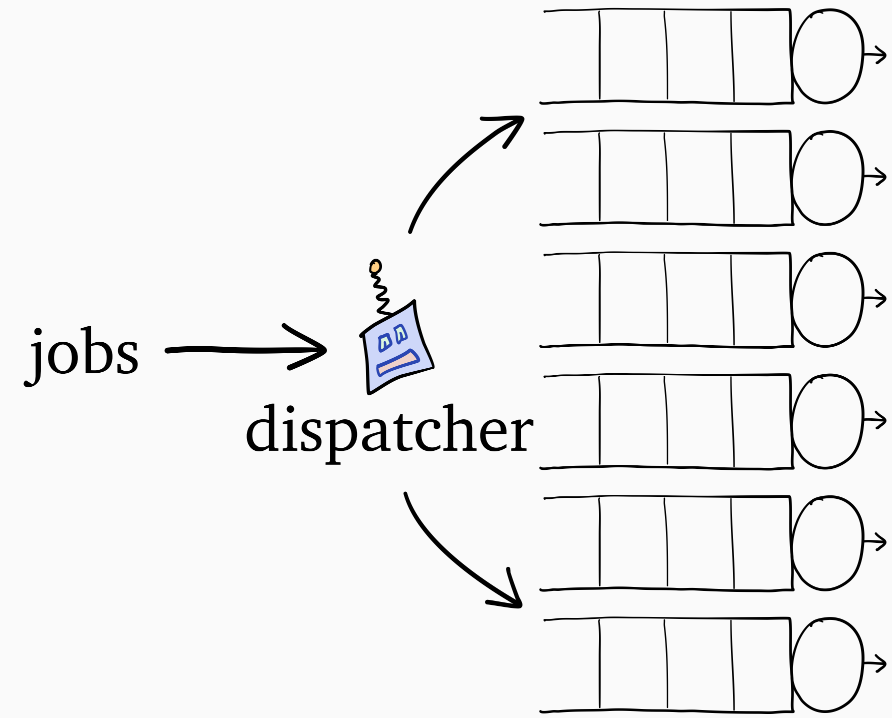
M/G/1

$$\mathbf{E}_{\text{time}}[u(X)\,w(X)] = 0$$

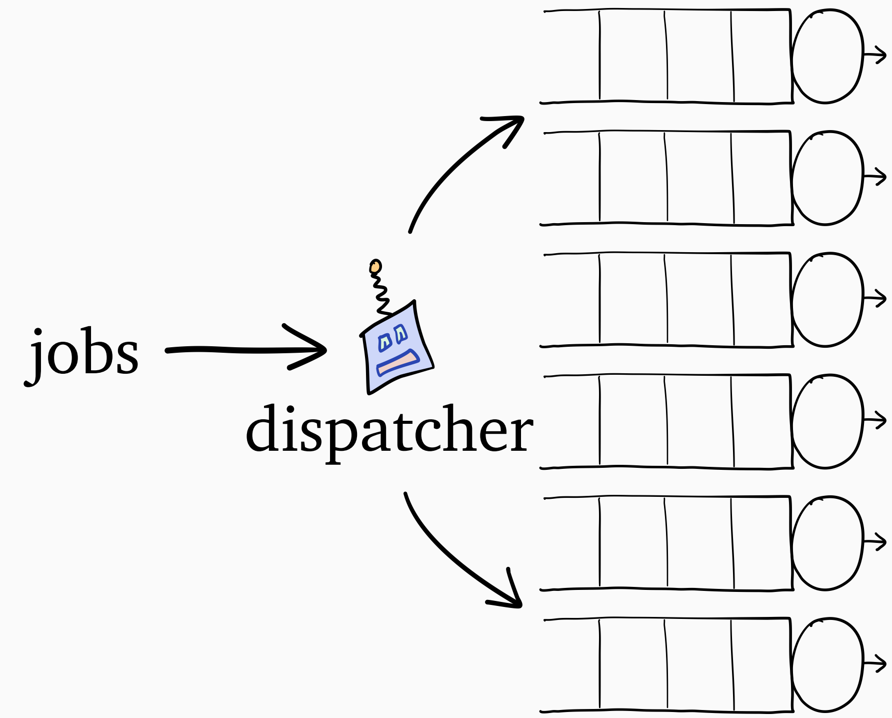
M/G/*k*

$\dfrac{\mathbf{E}_{\text{time}}[u(X)\,w(X)]}{1 - \lambda\mathbf{E}[S]}$ is work of $\leq k-1$ jobs

Dispatching systems

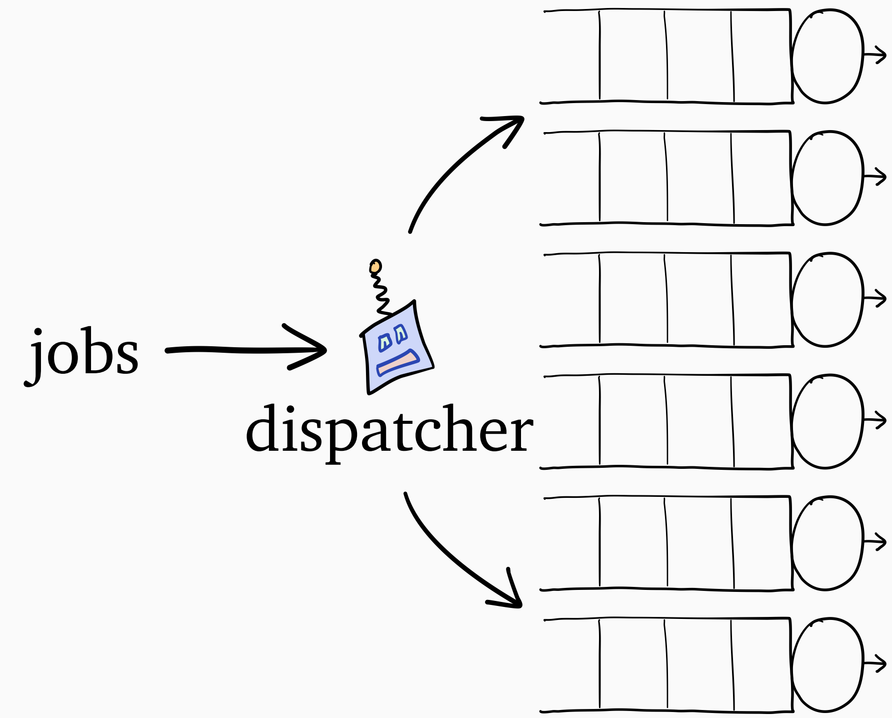


Dispatching systems

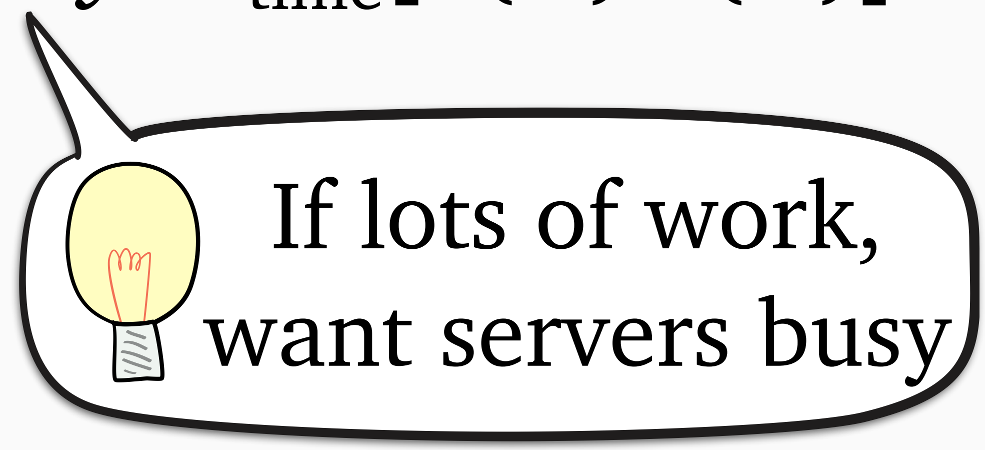


Key: $\mathbf{E}_{\text{time}}[u(X)\,w(X)]$

Dispatching systems

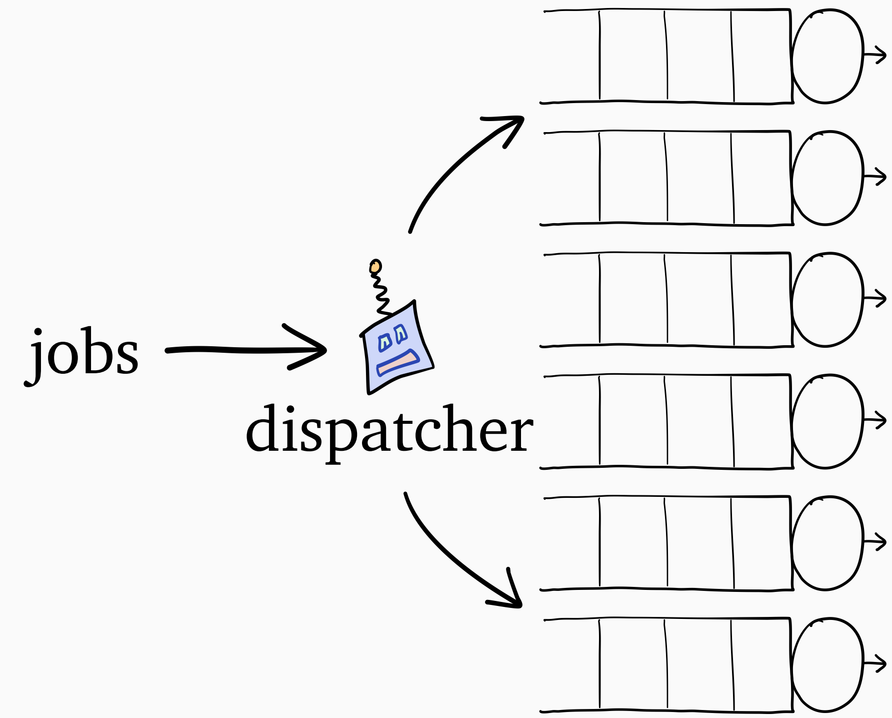


Key: $\mathbf{E}_{\text{time}}[u(X)\,w(X)]$



Dispatching systems

Possible policy:
dispatch to server
with less work

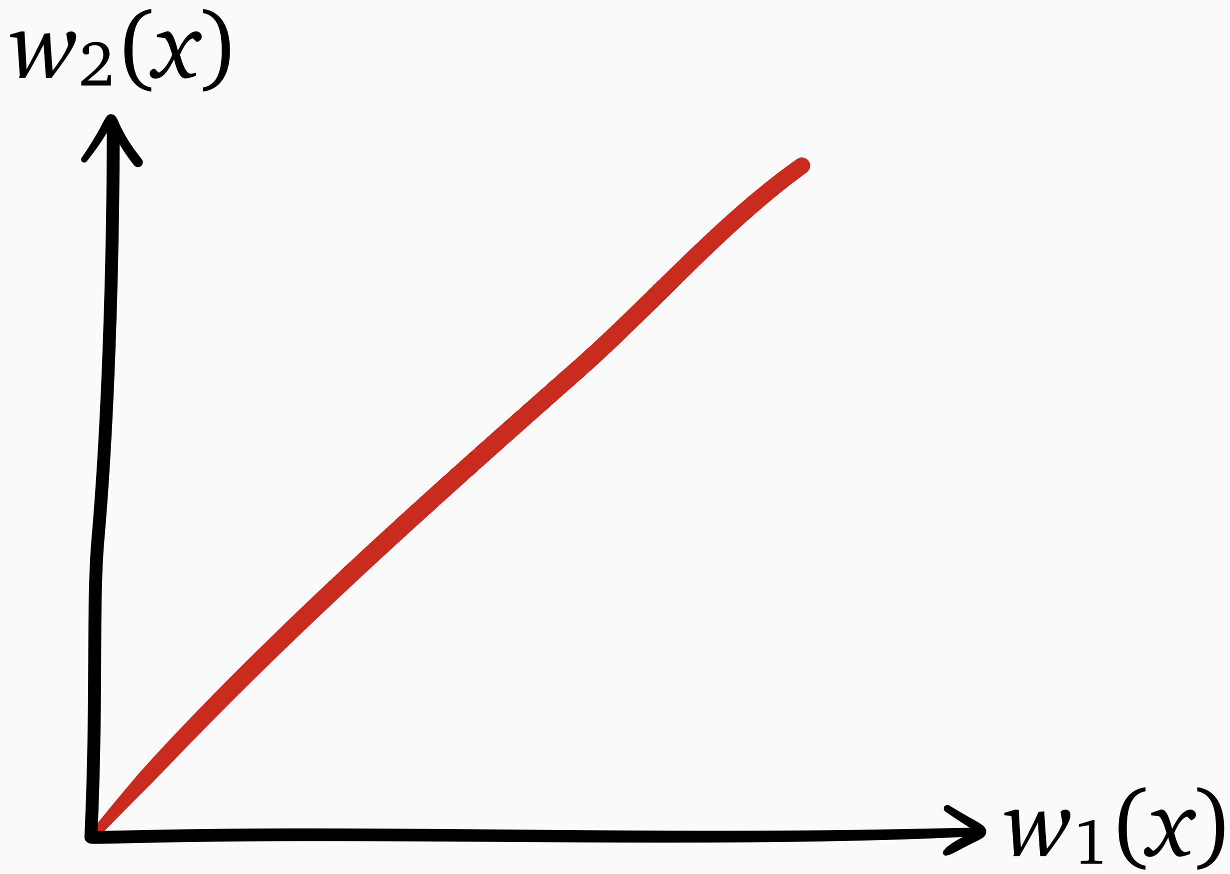


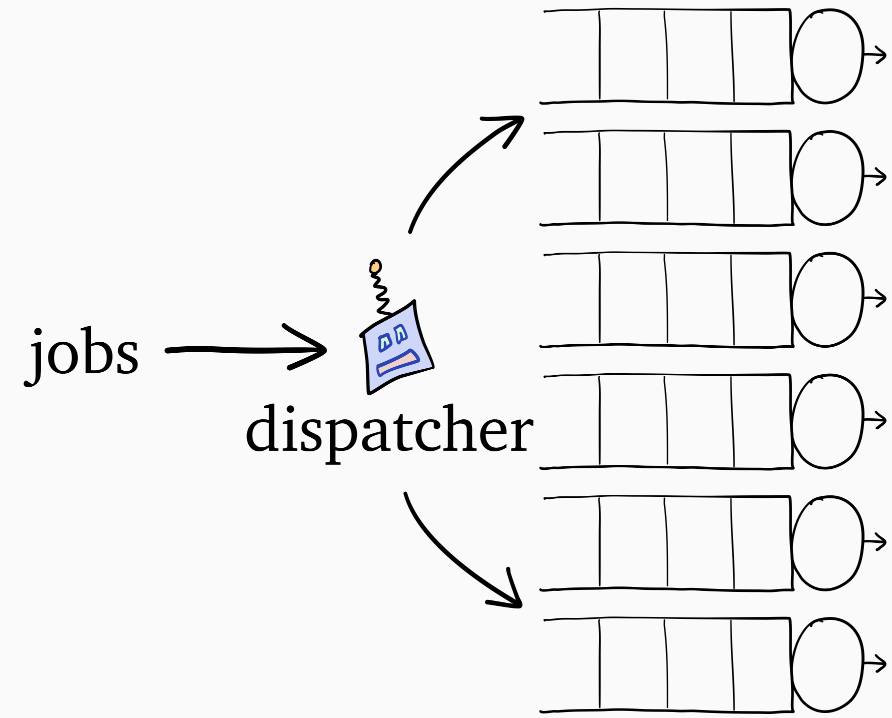
Key: $\mathbf{E}_{\text{time}}[u(X)\,w(X)]$

If lots of work,
want servers busy

Dispatching systems

Possible policy:
dispatch to server
with less work



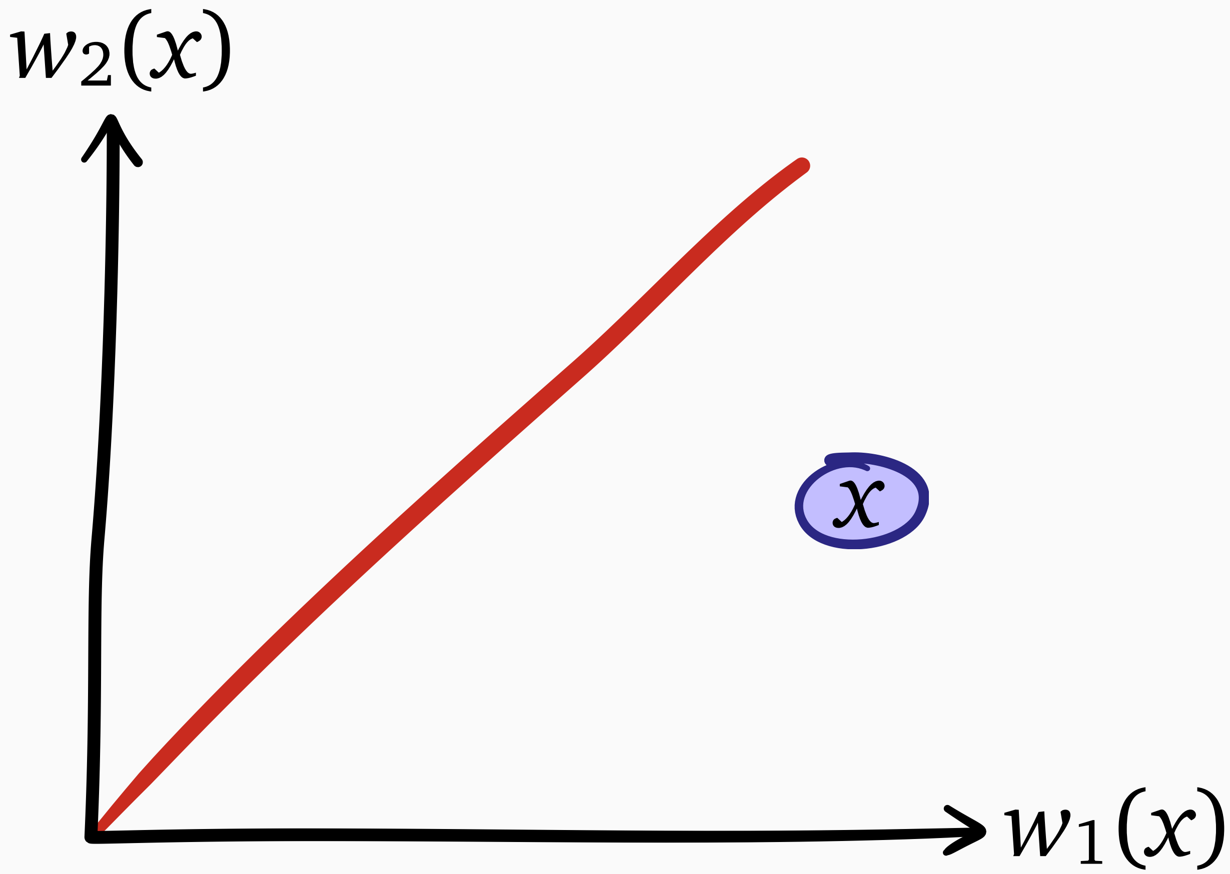


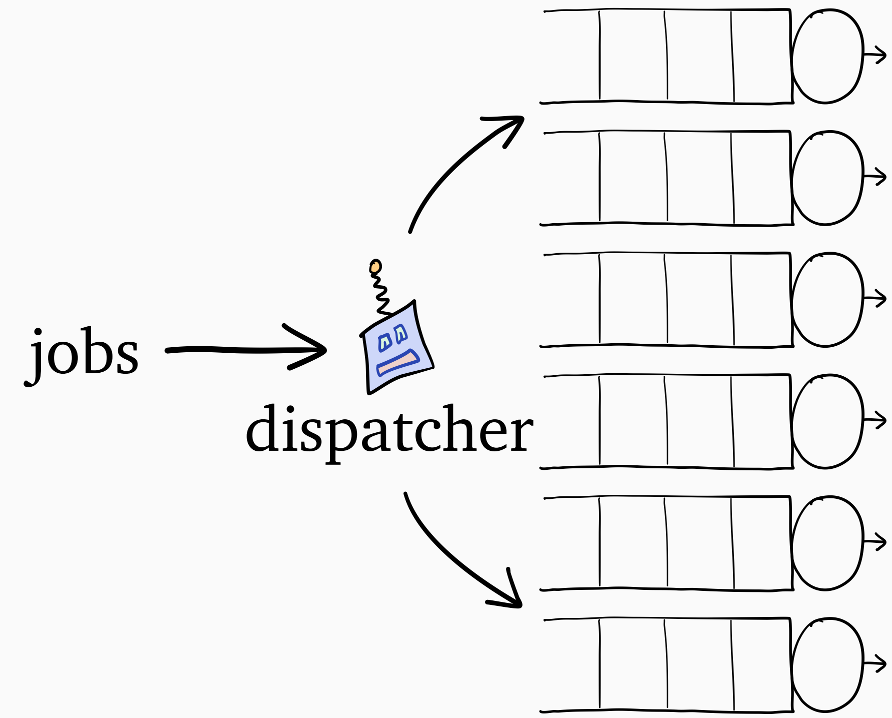
Key: $\mathbf{E}_{\text{time}}[u(X)\,w(X)]$

If lots of work,
want servers busy

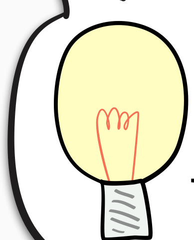
Dispatching systems

Possible policy:
dispatch to server
with less work





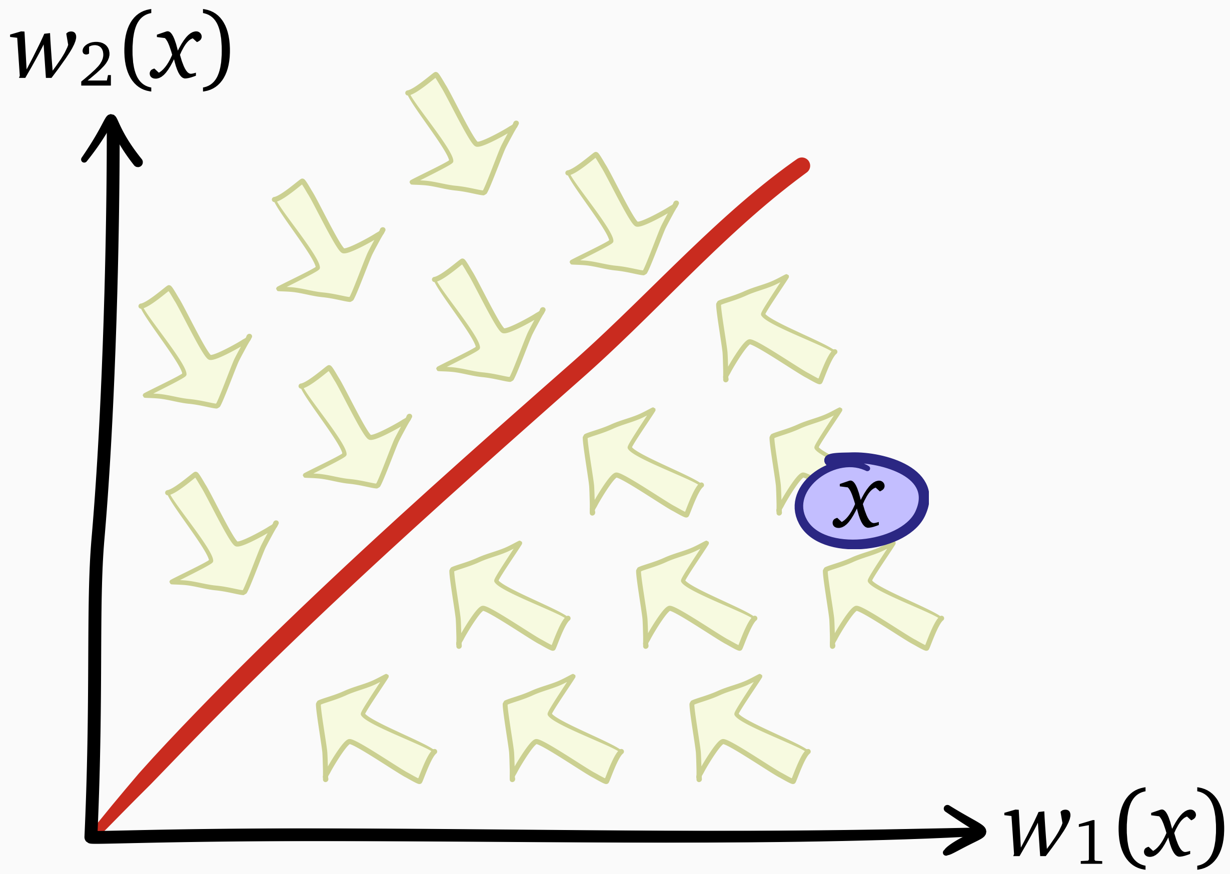
Key: $\mathbf{E}_{\text{time}}[u(X)\,w(X)]$

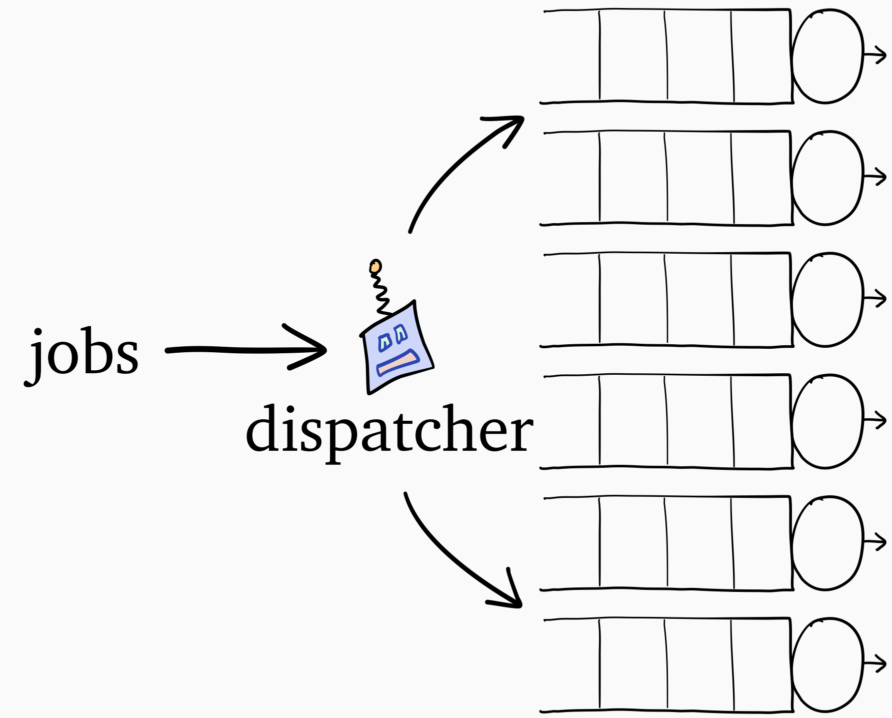


If lots of work,
want servers busy

Dispatching systems

Possible policy:
dispatch to server
with less work



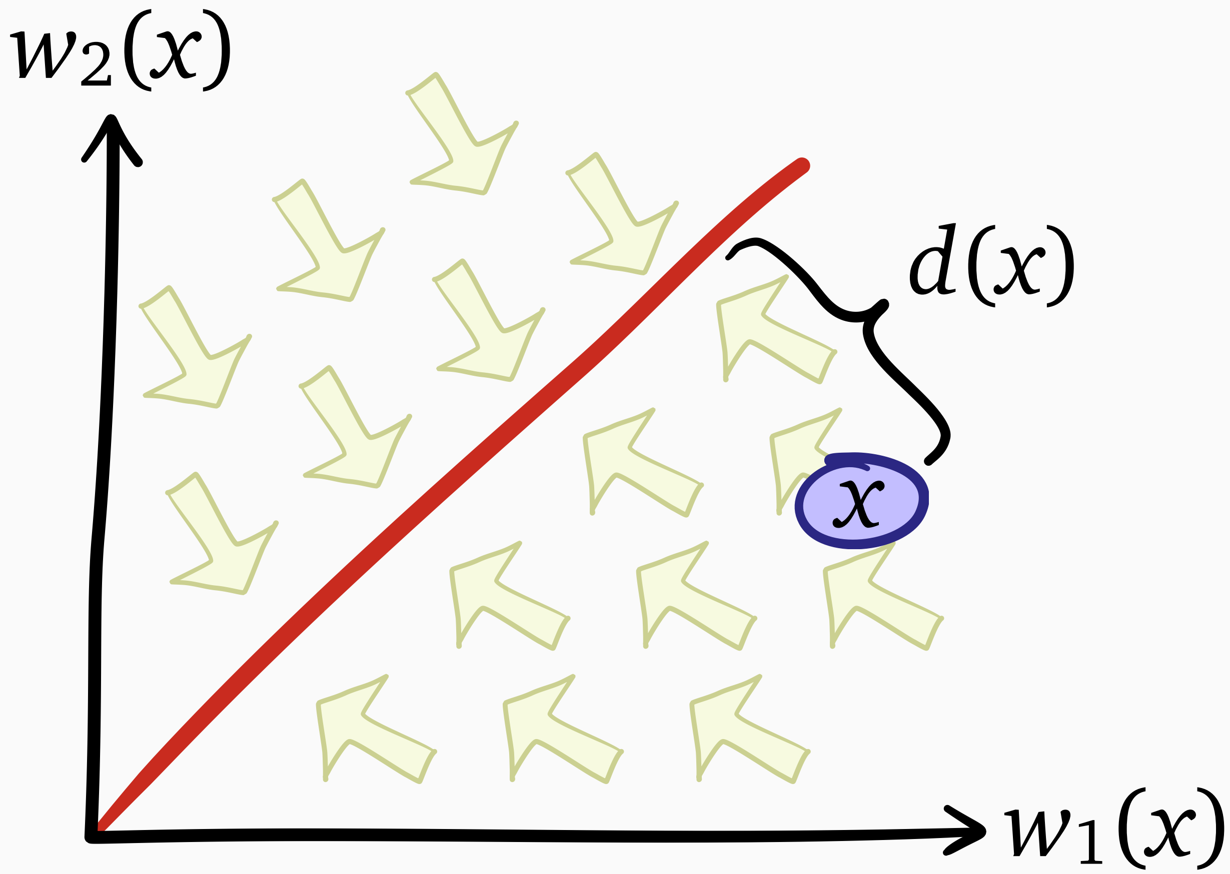


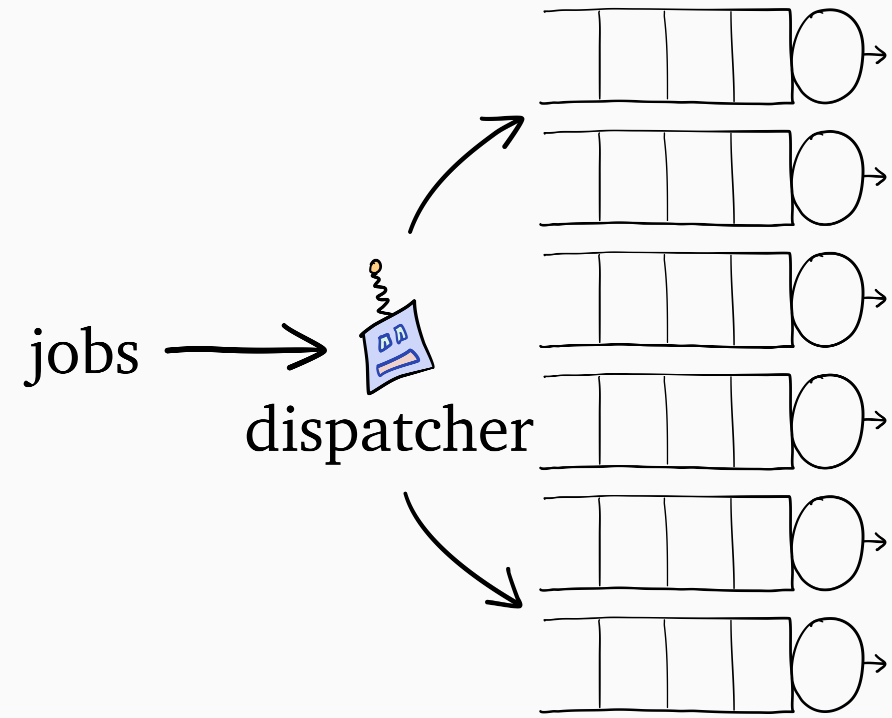
Key: $\mathbf{E}_{\text{time}}[u(X)\,w(X)]$

If lots of work,
want servers busy

Dispatching systems

Possible policy:
dispatch to server
with less work



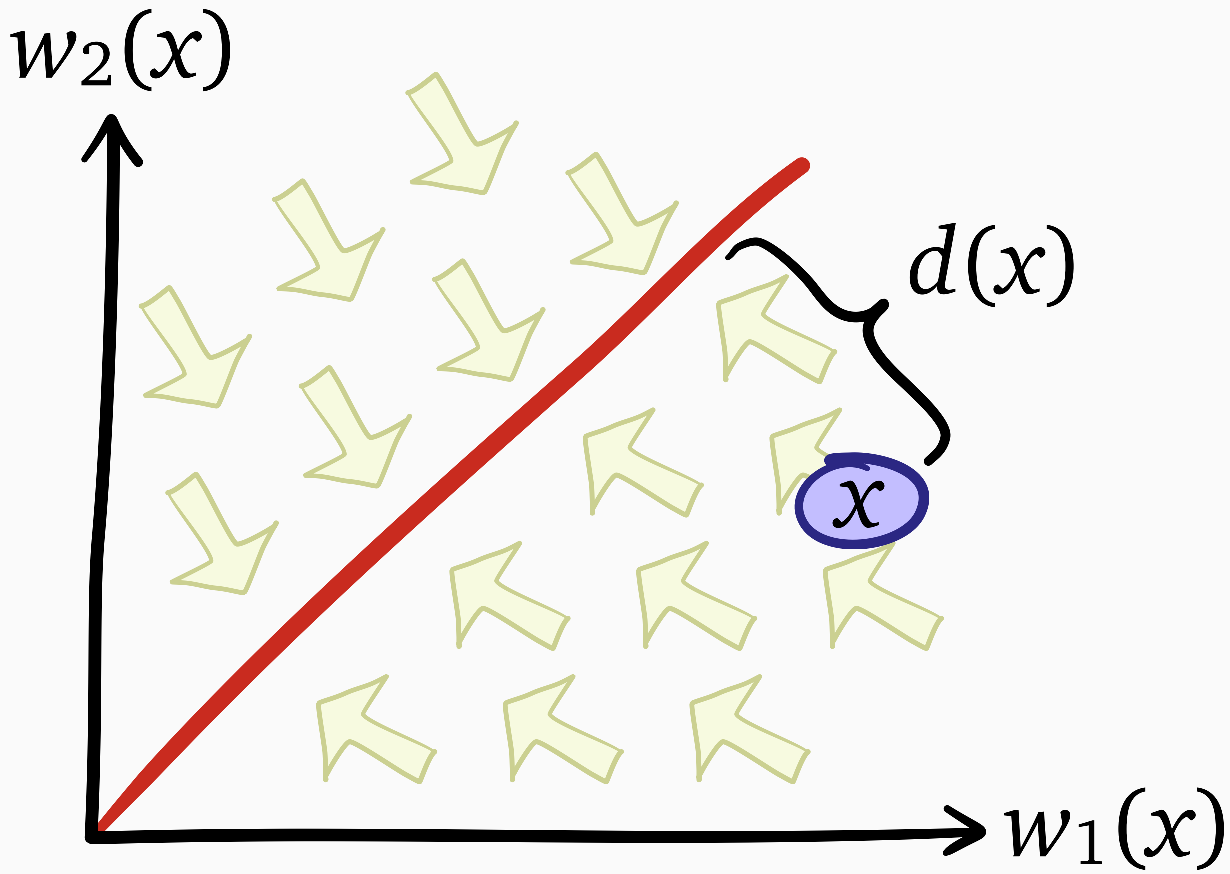


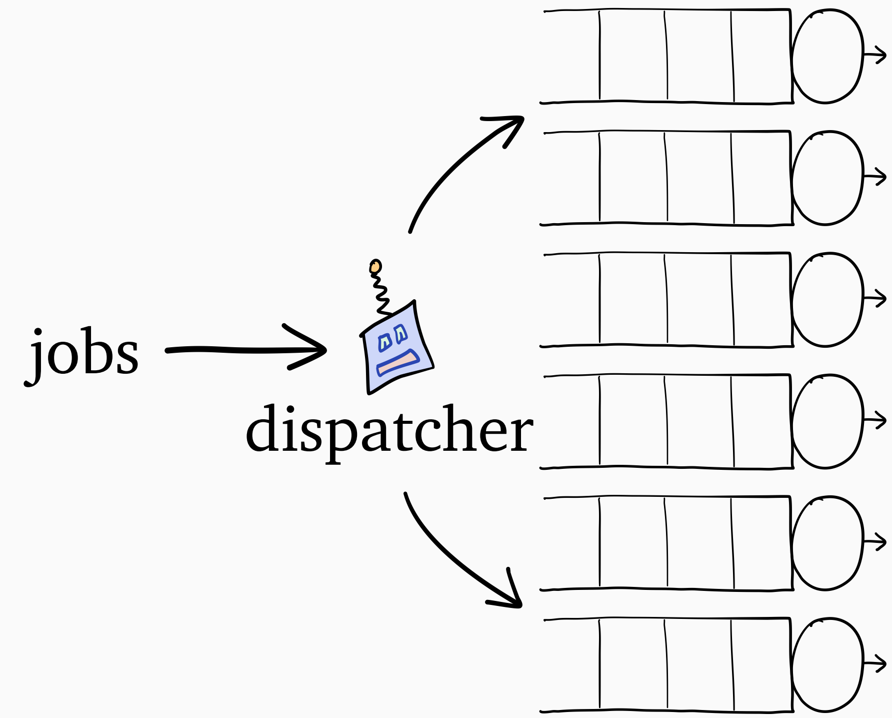
Key: $\mathbf{E}_{\text{time}}[u(X)\,w(X)]$

If lots of work,
want servers busy

Dispatching systems

Possible policy:
dispatch to server
with less work





Key: $\mathbf{E}_{\text{time}}[u(X)\,w(X)]$

If lots of work,
want servers busy

$f(x) = \exp(\theta d(x))$ **RCL** ⇒ state space collapse

What principles do we need?

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $= 0$: remove head of list
- Poisson(λ): draw from S, append it to list

$\Big\Downarrow$ **RCL**

$$
\begin{aligned}
0 = \mathbf{E}_{\text{time}}&\left[\frac{\partial}{\partial r_1} f(X)\right] \\
&+ \lambda \mathbf{E}_{\text{departure}}\left[f(\text{tail}(X)) - f(X)\right] \\
&+ \lambda \mathbf{E}_{\text{arrival}}\left[f(\text{join}(X, [S])) - f(X)\right]
\end{aligned}
$$

What principles do we need?

Dynamics:

- If list nonempty: decrease head at rate 1
- When head = 0: remove head of list
- Poisson(λ): draw from S, append it to list

RCL

$$
\begin{aligned}
0 = \mathbf{E}_{\text{time}}&\left[\frac{\partial}{\partial r_1} f(X)\right] \\
&+ \lambda \mathbf{E}_{\text{departure}}\left[f(\text{tail}(X)) - f(X)\right] \\
&+ \lambda \mathbf{E}_{\text{arrival}}\left[f(\text{join}(X, [S])) - f(X)\right]
\end{aligned}
$$

Principle: translating dynamics to mean rate

What principles do we need?

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $= 0$: remove head of list
- Poisson(λ): draw from S, append it to list

RCL

$$
\begin{aligned}
0 = \mathbf{E}_{\text{time}}&\left[\frac{\partial}{\partial r_1} f(X)\right] \\
&+ \lambda \mathbf{E}_{\text{departure}}\left[f(\text{tail}(X)) - f(X)\right] \\
&+ \lambda \mathbf{E}_{\text{arrival}}\left[f(\text{join}(X, [S])) - f(X)\right]
\end{aligned}
$$

Principle: translating dynamics to mean rate

Principle for stability?

What principles do we need?

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $= 0$: remove head of list
- Poisson(λ): draw from S, append it to list

RCL

$$
\begin{aligned}
0 = \mathbf{E}_{\text{time}}&\left[\frac{\partial}{\partial r_1} f(X)\right] \\
&+ \lambda \mathbf{E}_{\text{departure}}\left[f(\text{tail}(X)) - f(X)\right] \\
&+ \lambda \mathbf{E}_{\text{arrival}}\left[f(\text{join}(X, [S])) - f(X)\right]
\end{aligned}
$$

Principle: translating dynamics to mean rate

Principle for stability?

Principle for mean field?

What principles do we need?

Dynamics:

- If list nonempty: decrease head at rate 1
- When head $= 0$: remove head of list
- Poisson(λ): draw from S, append it to list

RCL

$$
\begin{aligned}
0 = \mathbf{E}_{\text{time}}&\left[\frac{\partial}{\partial r_1} f(X)\right] \\
+ \lambda \mathbf{E}_{\text{departure}}&\left[f(\text{tail}(X)) - f(X)\right] \\
+ \lambda \mathbf{E}_{\text{arrival}}&\left[f(\text{join}(X, [S])) - f(X)\right]
\end{aligned}
$$

Principle: translating dynamics to mean rate

? Principle for stability?

? Principle for mean field?

? Principles for composition?

Part 1

Performance modeling needs advanced math

Part 2

We can teach advanced math accessibly

Simplify core foundations

Prioritize very flexible tools