Optimizing Resource Sharing Systems

VARUN GUPTA
Carnegie Mellon University

Based on papers co-authored with:

JIM DAI	MOR HARCHOL-BALTER	KARL SIGMAN, WARD WHITT	BERT ZWART
Georgia Tech	Carnegie Mellon	Columbia University	CWI, Netherlands

Resource sharing systems are everywhere…

I/O+CPU+Bandwidth
by Web servers

Benefits of
resource sharing?

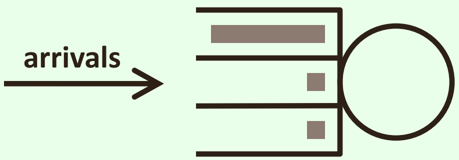
CPU cycles by
OS task scheduler

Wireless channel
by WAPs

…and you!

Why resource sharing: A queueing theory primer

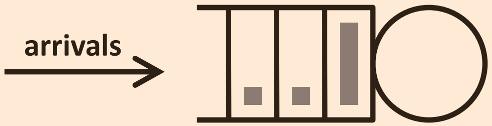
Processor Sharing (PS)



n jobs $\Rightarrow$ each job gets 1/n capacity

3+3+6=12

First-Come-First-Served (FCFS)



Earliest job to arrive is served until completed

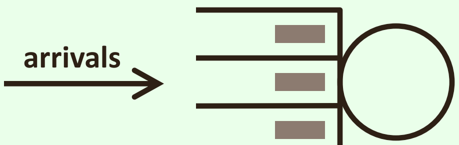
4+5+6=15

Which has smaller mean response time?

☑ PS ☐ FCFS

Why resource sharing: A queueing theory primer

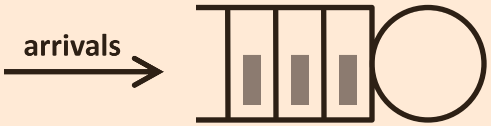
Processor Sharing (PS)



n jobs ⇒ each job gets 1/n capacity

6+6+6=18

First-Come-First-Served (FCFS)



Earliest job to arrive is served until completed

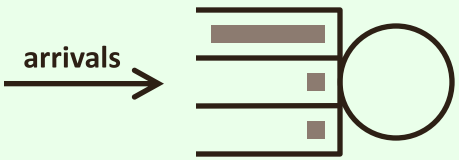
2+4+6=12

Now which has smaller mean response time?

☐ PS ☑ FCFS

Why resource sharing: A queueing theory primer

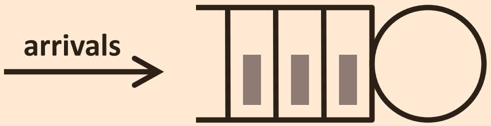
Processor Sharing (PS)



n jobs ⇒ each job gets 1/n capacity

✓ Good for <u>high</u> job-size variability

First-Come-First-Served (FCFS)

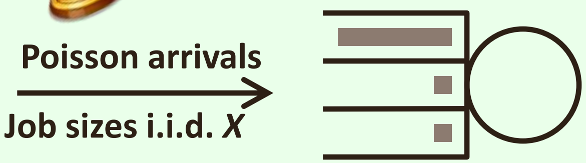


Earliest job to arrive is served until completed

✓ Good for <u>low</u> job-size variability

Why resource sharing: A queueing theory primer

Processor Sharing (M/G/1/PS)



n jobs $\Rightarrow$ each job gets 1/n capacity

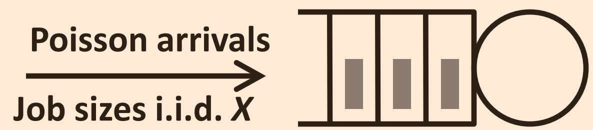
✔ Good for high job-size variability

$$E[T^{PS}] = \frac{E[X]}{1-\rho}$$

$$\rho = \text{arrival rate} \cdot E[X]$$

measure of system utilization

First-Come-First-Served (M/G/1/FCFS)



Earliest job to arrive is served until completed

✔ Good for low job-size variability

$$E[T^{FCFS}] = E[T^{PS}] \left(1 + \rho \cdot \frac{C^2 - 1}{2}\right)$$

$$C^2 = \frac{var(X)}{E[X]^2}$$

measure of job size variability

UNIX process lifetimes: $C^2 > 40$

Files transferred over Internet: $C^2 > 25$

} **Variability matters!**

Real world ≠ Ideal theoretical policies

Reality check 1: Context-switch overheads

- ! Quantum-based Round-Robin
- ? How to choose the optimal quantum size?

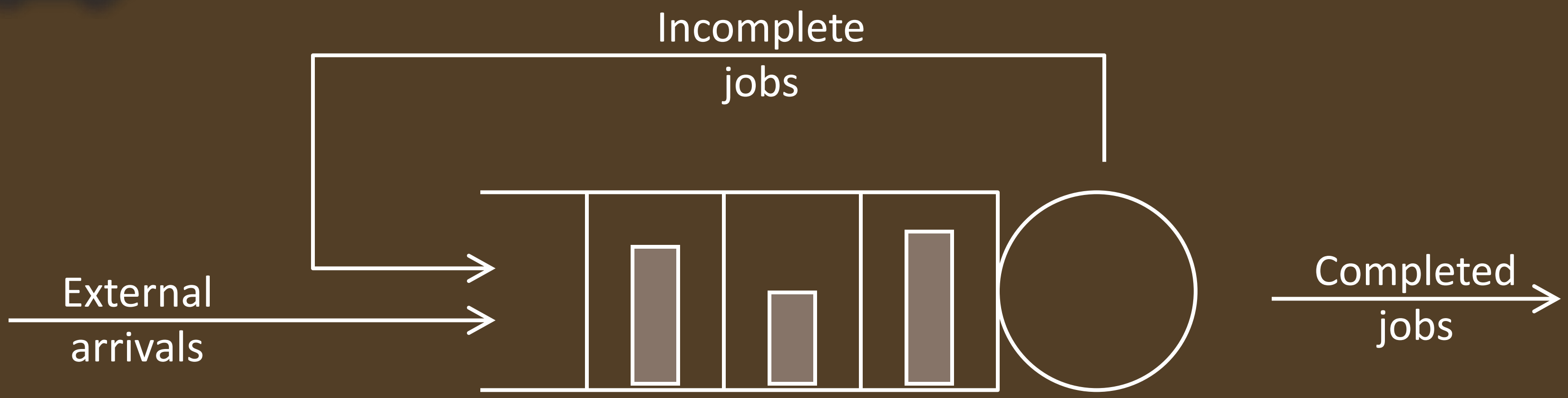
Reality check 2: Thrashing

- ! Impose a Multi-Programming-Limit (MPL)
- ? How to choose the optimal MPL?

Reality check 3: Load balancing in server farms

- ? How do load-balancing algorithms interact with servers?
- ? What are good load-balancing algorithms?

Quantum-based Round-Robin (RR)



Jobs served for *q* units at a time

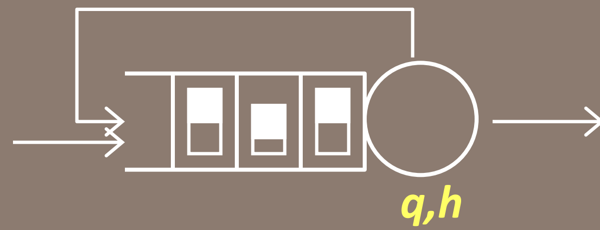
h units of context-switch overhead after every quantum

Quantum-based Round-Robin (RR)

Jobs served for *q* units at a time

h units of context-switch overhead after every quantum

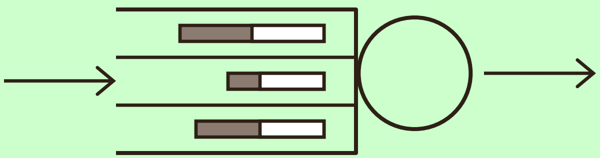
M/G/1/RR



$q \rightarrow 0$
$h = 0$

$q = \infty$
$h = 0$

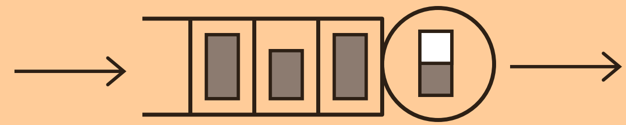
M/G/1/PS



$$E[T^{PS}] = \frac{E[X]}{1-\rho}$$

✘ **Context-switches cause overhead**

M/G/1/FCFS



$$E[T^{FCFS}] = E[T^{PS}]\left(1 + \rho\frac{C^2-1}{2}\right)$$

✘ **Variable job sizes cause long delays**

small q large q

A hammer for most occasions, …the H^* job-size distribution

$$H^* \sim \begin{cases} 0 & \text{w.p. } p \\ \mathrm{Exp}(\gamma) & \text{w.p. } 1-p \end{cases}$$

- 2 degrees of freedom
- Can match any E[X] and $C^2 \geq 1$

- $\mathrm{Exp}(\gamma) \equiv$ Exponential distribution
    - easy to analyze $\Leftarrow$ Markov chains

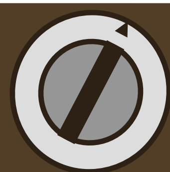
- H^* captures the key phenomenon of (frequent) small vs. (rare) big jobs

For many systems (all cases in this talk), H^* provides a good approximation for mean response time.

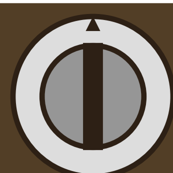
Step 1: M/G/1/RR with no overheads

$$E[T^{RR}] \approx E[T^{PS}] \left[1 + \frac{C^2-1}{C^2+1} \cdot \frac{\rho}{\frac{E[X]}{q} + \frac{2}{C^2+1}} \right]$$

$E[T^{PS}]$ ← 0 ——— ∞ → $E[T^{FCFS}] = E[T^{PS}] \left(1 + \rho \frac{C^2-1}{2}\right)$


q

$E[T^{PS}]$ ← 1 ——— ∞ → $E[T^{PS}](1 + \rho q / E[X])$


*C*²

For high C^2: $E[T^{RR}] \approx E[T^{PS}](1 + \rho q / E[X])$

Step 2: Optimizing *q*

1. System with context-switch overhead *h* → a system with no overheads
   - New quantum size = *q*+*h*
   - Stretch job sizes by a factor (1+*h*/*q*)

2. OPT quantum $q^* = \text{argmin}_q E[T^{RR}]$

*q** is a simple function of *h*, *E*[*X*] and utilization

Common case: *h* ≪ *E*[*X*]

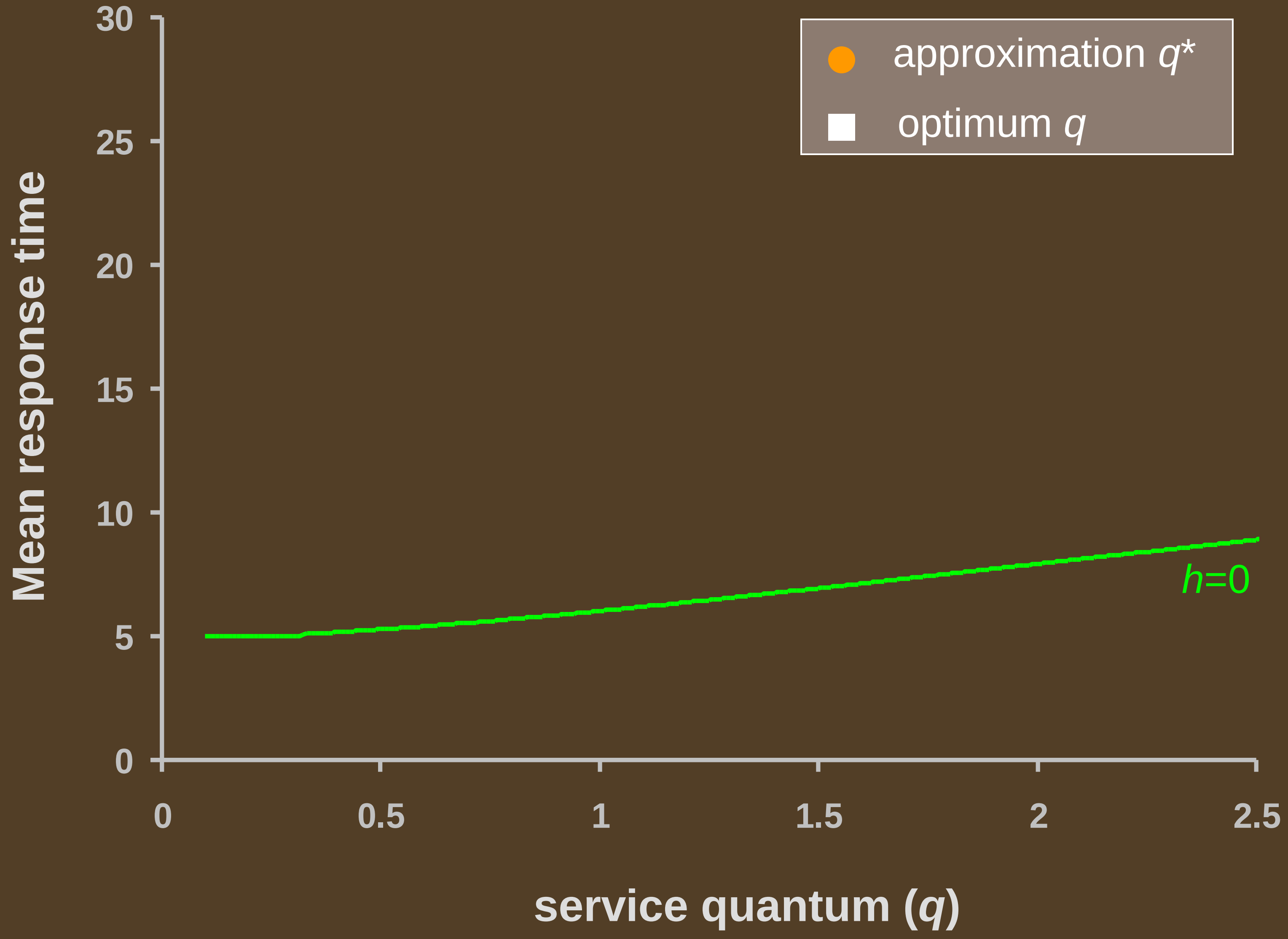
$$q^* \approx \alpha(\rho)\sqrt{hE[X]}$$

EXAMPLE: Linux context switch time ≈ 5 microseconds

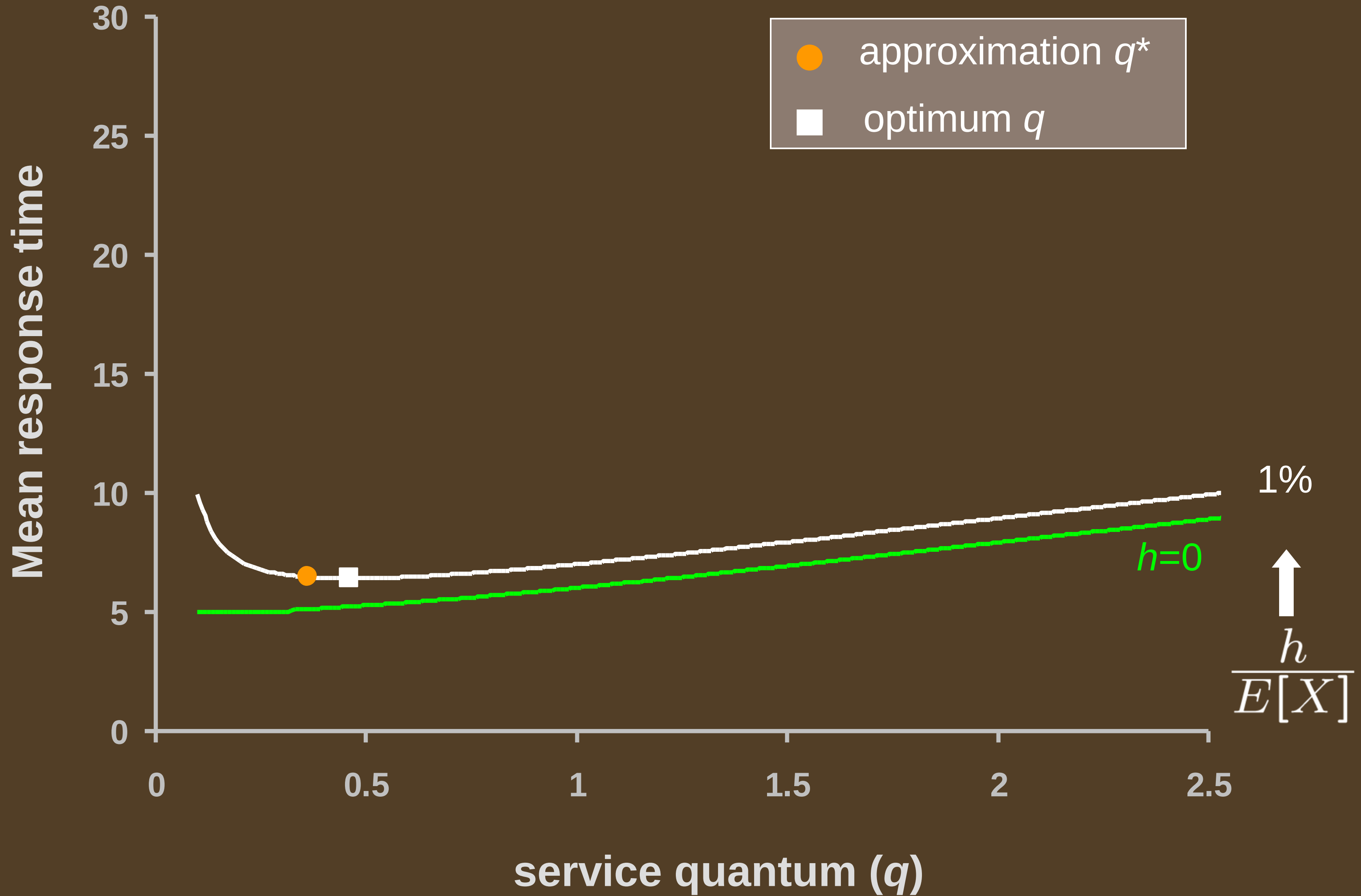
Assume: mean job size = 5 sec, 80% utilization

q* ≈ 15 msec

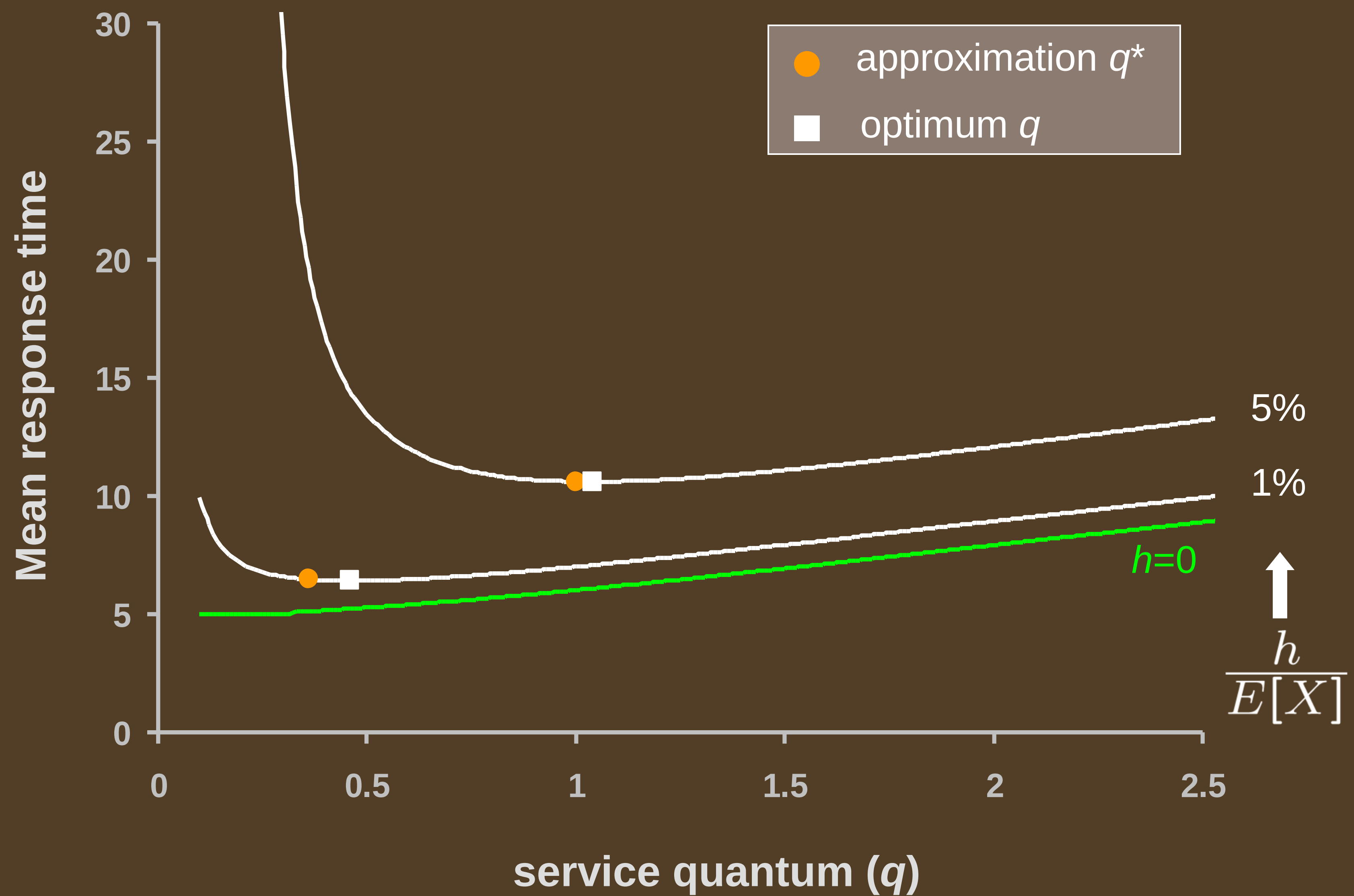
Actual Linux quantum size = between 10 and 200 msec



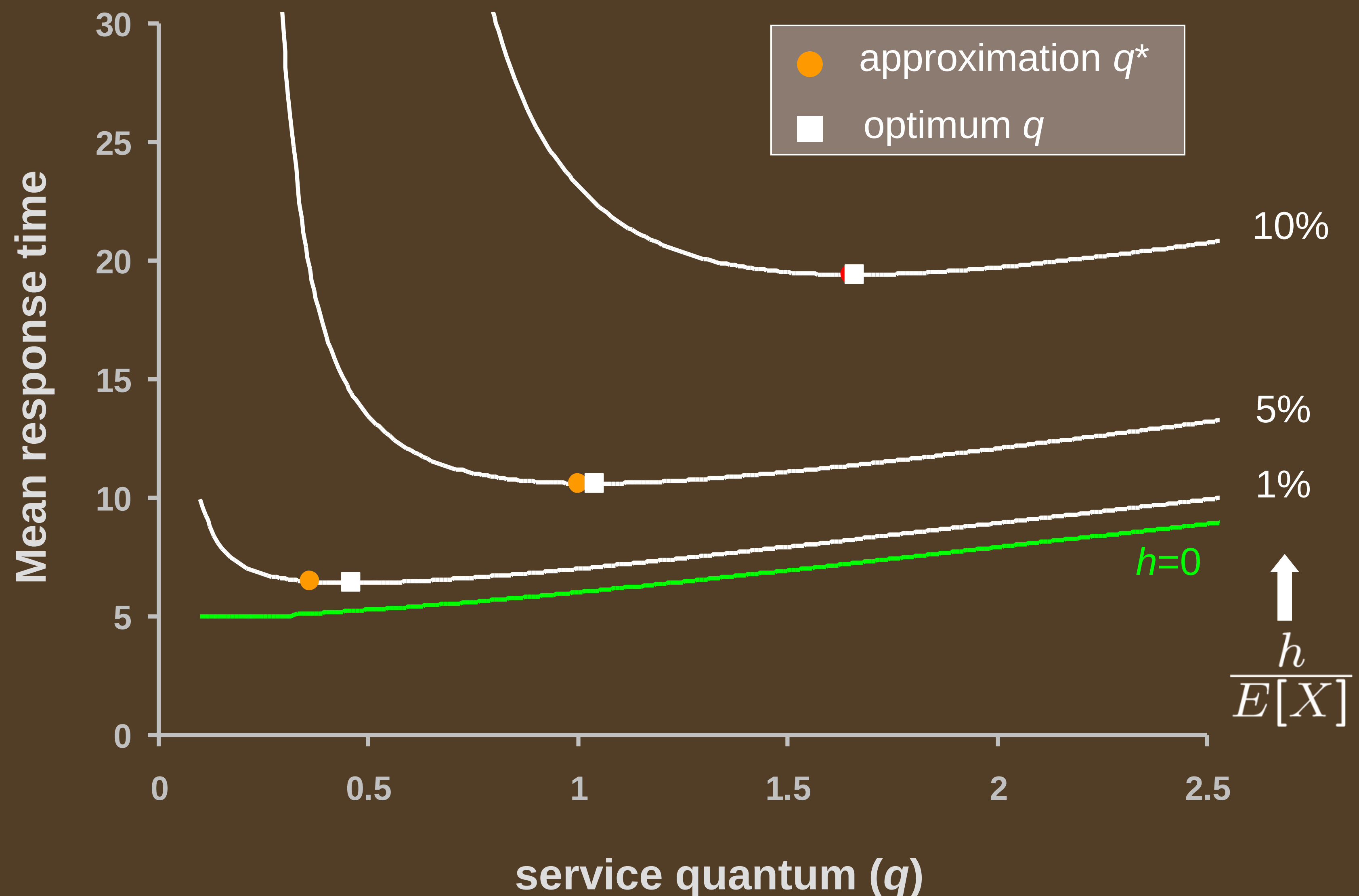
$E[X] = 1$, $C^2 = 19$, $\rho = 0.8$



$E[X] = 1$, $C^2 = 19$, $\rho = 0.8$



$E[X] = 1$, $C^2 = 19$, $\rho = 0.8$



1. Effect of context-switch overheads can be significant
   - performance quite far from ideal PS
2. Choosing too small a q is very bad, OK to err towards larger q
3. Performance of q^* close to OPT

Real world ≠ Ideal theoretical policies

Reality check 1: Context-switch overheads

(!) Quantum-based Round-Robin

(?) How to choose the optimal quantum size?

Reality check 2: Thrashing

(!) Impose a Multi-Programming-Limit (MPL)

(?) How to choose the optimal MPL?

Reality check 3: Load balancing in server farms

(?) How do load-balancing algorithms interact with servers?

(?) What are good load-balancing algorithms?

Real world ≠ Ideal theoretical policies

Reality check 1: Context-switch overheads

! Quantum-based Round-Robin

? How to choose the optimal quantum size?

Reality check 2: Thrashing

! Impose a Multi-Programming-Limit (MPL)

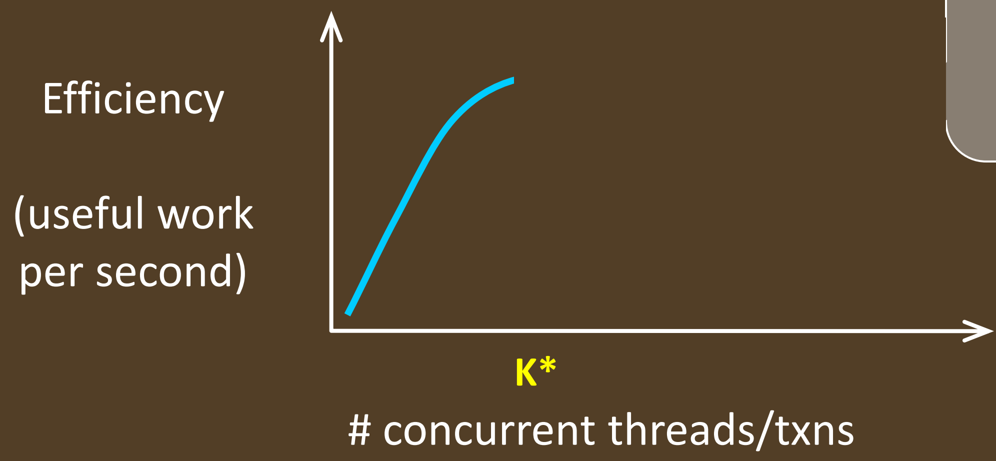
? How to choose the optimal MPL?

Reality check 3: Load balancing in server farms

? How do load-balancing algorithms interact with servers?

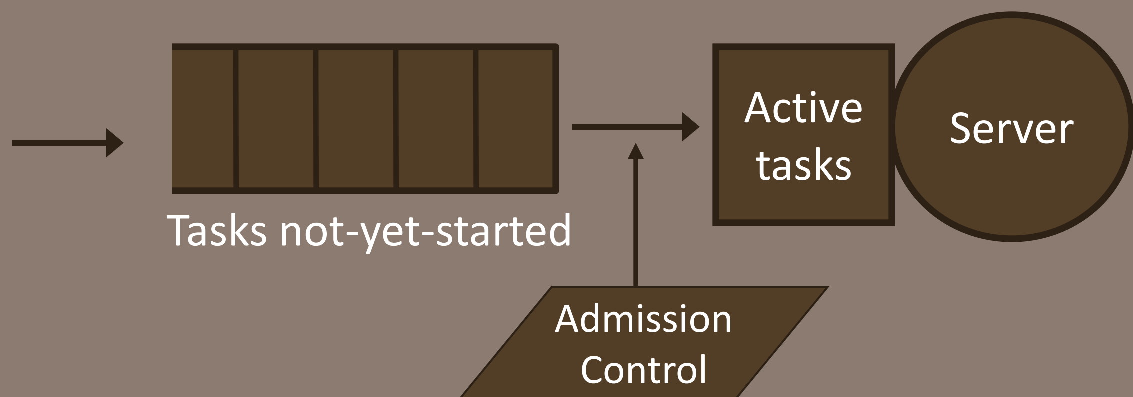
? What are good load-balancing algorithms?

Tale of a typical server

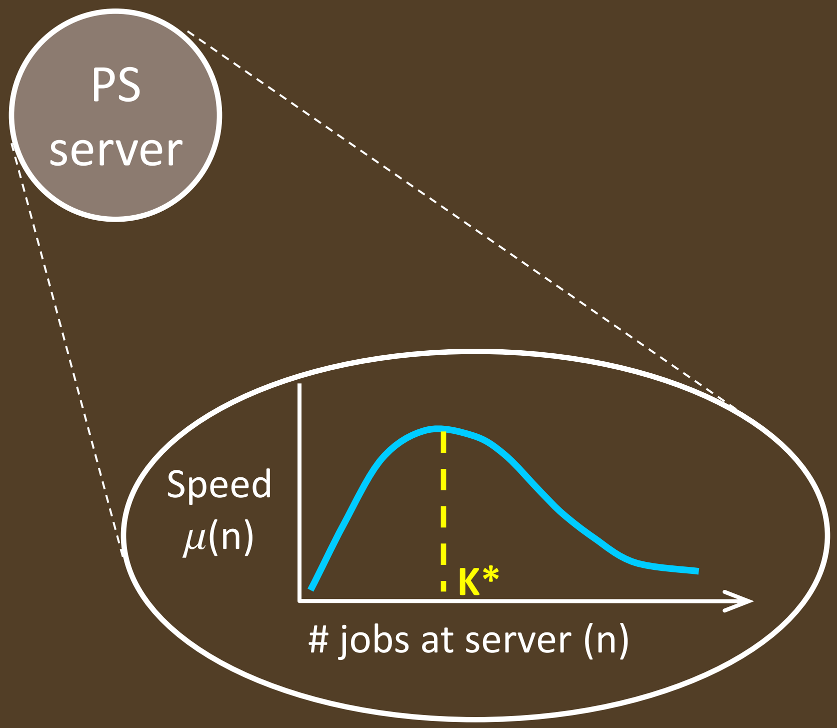


Q: Max number of tasks allowed to share server?

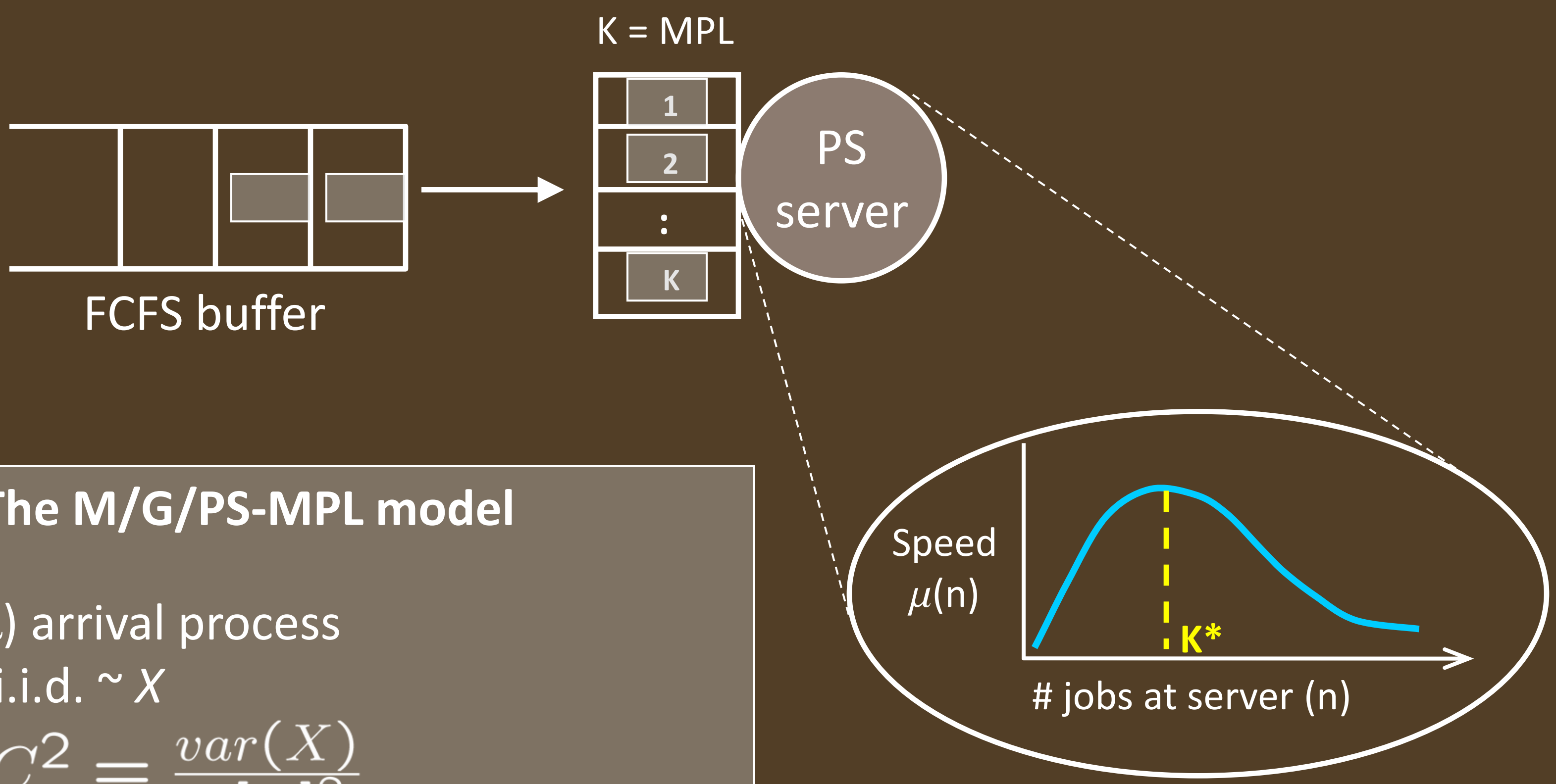
Common solution: K*



A Queueing-theoretic model



A Queueing-theoretic model



The M/G/PS-MPL model

- Poisson(λ) arrival process
- Job sizes i.i.d. ~ X

$$C^2 = \frac{var(X)}{E[X]^2}$$

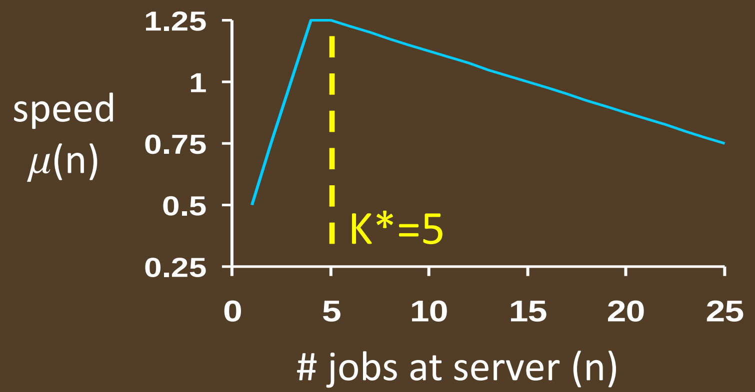
- Sizes unknown, distribution of X known

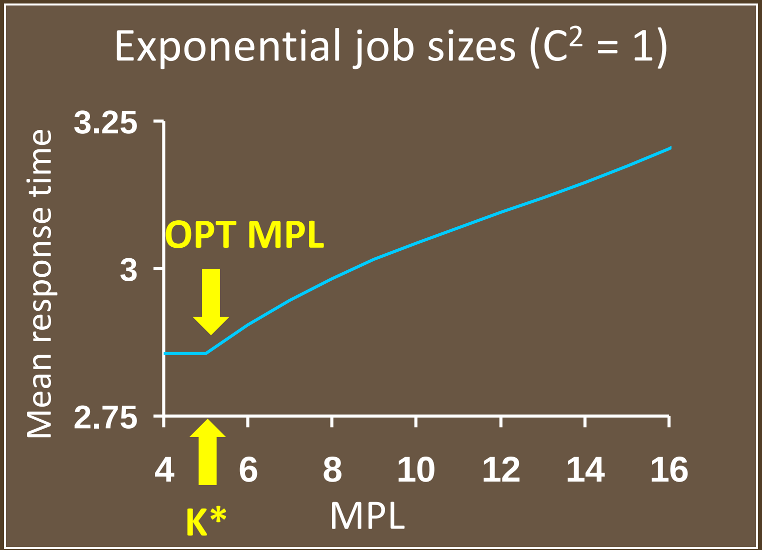
GOAL: Find MPL (i.e. K) to minimize mean response time

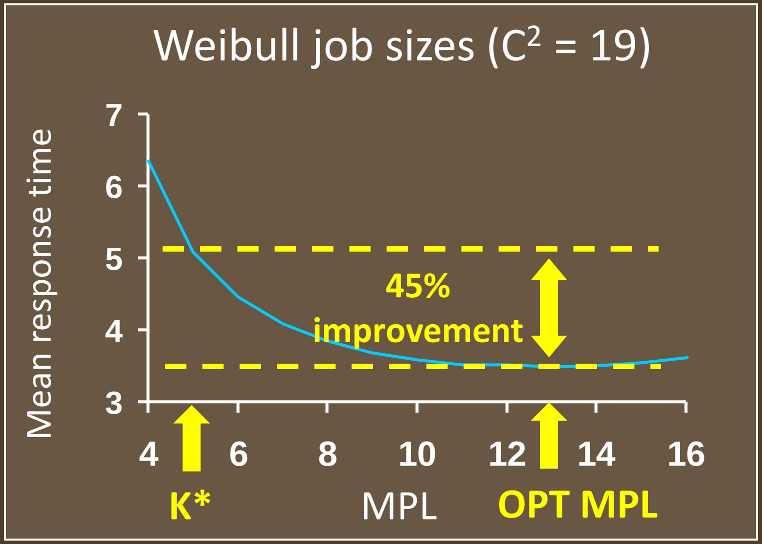
Optimal MPL= K* ?

Example

Poisson(0.8) arrival process



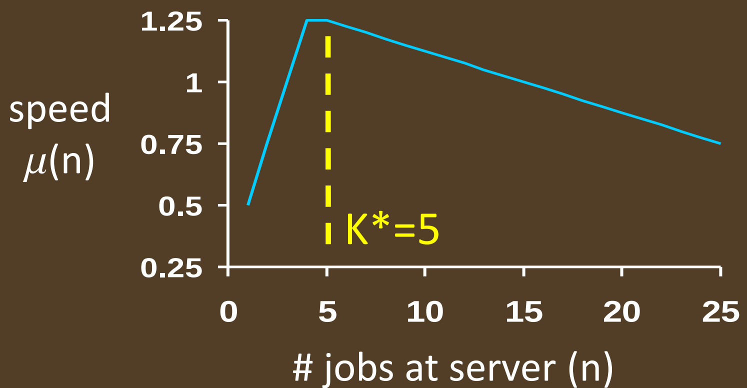


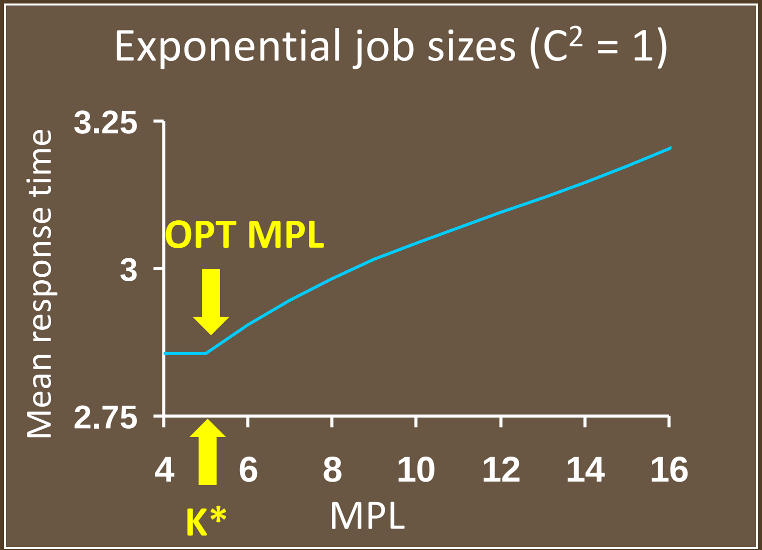


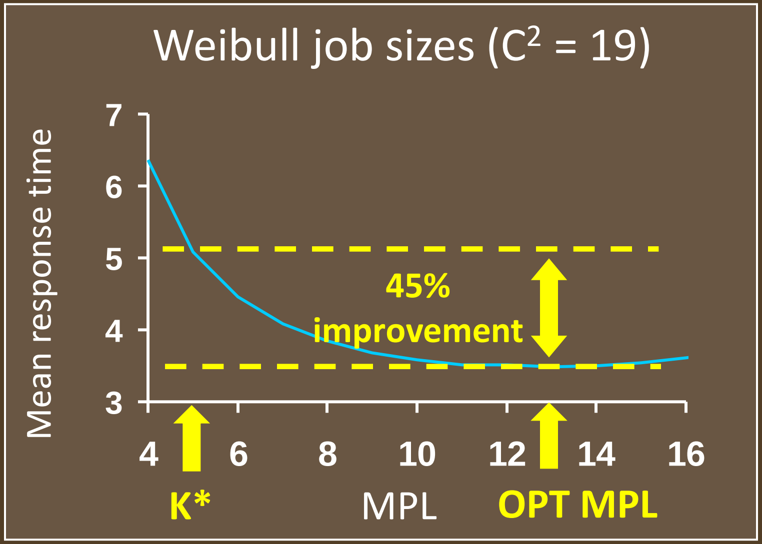
Optimal MPL= K* ? **DEPENDS!**

Example

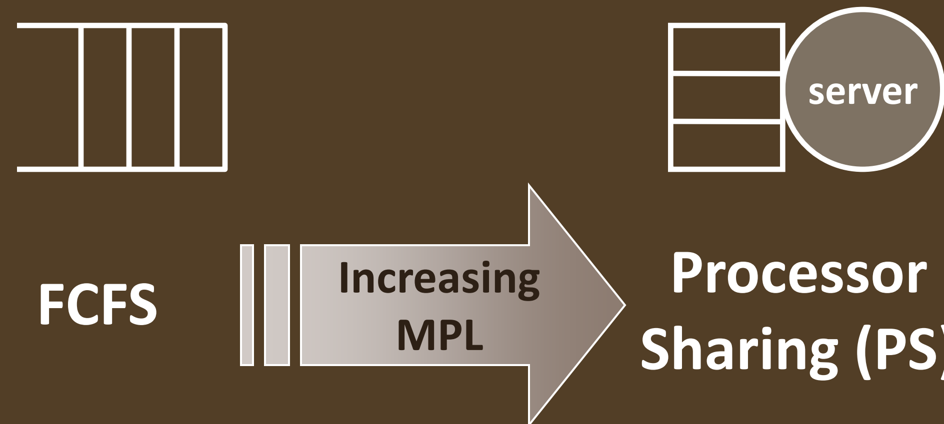
Poisson(0.8) arrival process



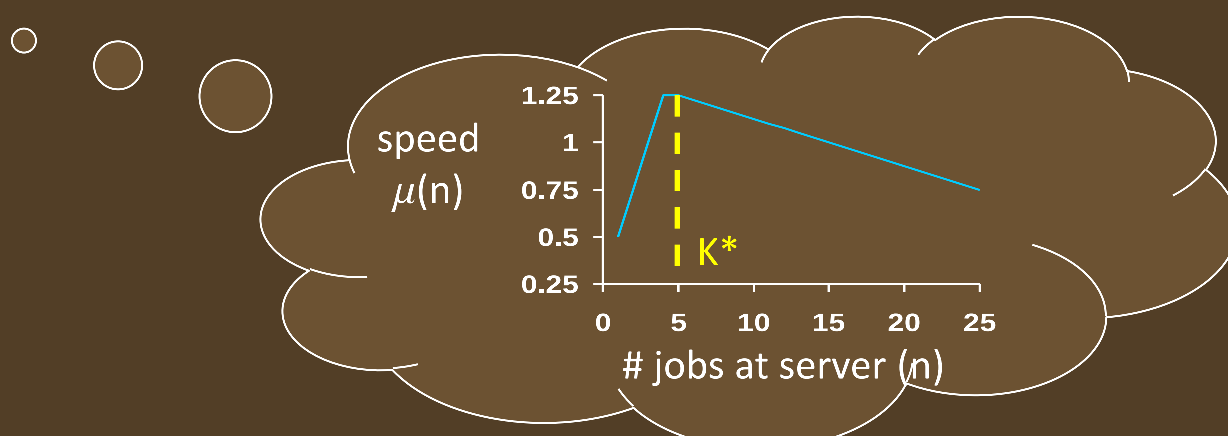




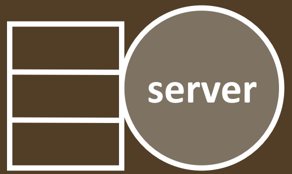
Intuition for the effect of MPL



	FCFS	Processor Sharing (PS)
High job-size variability (C^2)	✗	✓
High arrival rate (λ)	✓	✗



Intuition for the effect of MPL



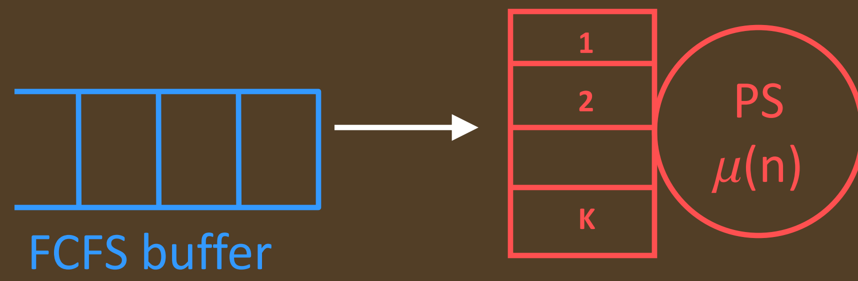
FCFS — Increasing MPL → **Processor Sharing (PS)**

	FCFS	Processor Sharing (PS)
High job-size variability (C^2)	✗	✓
High arrival rate (λ)	✓	✗

$C^2 \uparrow \quad \Longrightarrow \quad$ **Optimal MPL ↑**

Arrival rate ↑ $\quad \Longrightarrow \quad$ **Optimal MPL → K***

Step 1: M/G/PS-MPL approximation



$$E\left[T_X\right] = E\left[T_X^Q(K)\right] + E\left[T_X^S(K)\right]$$

Approximation assumption:

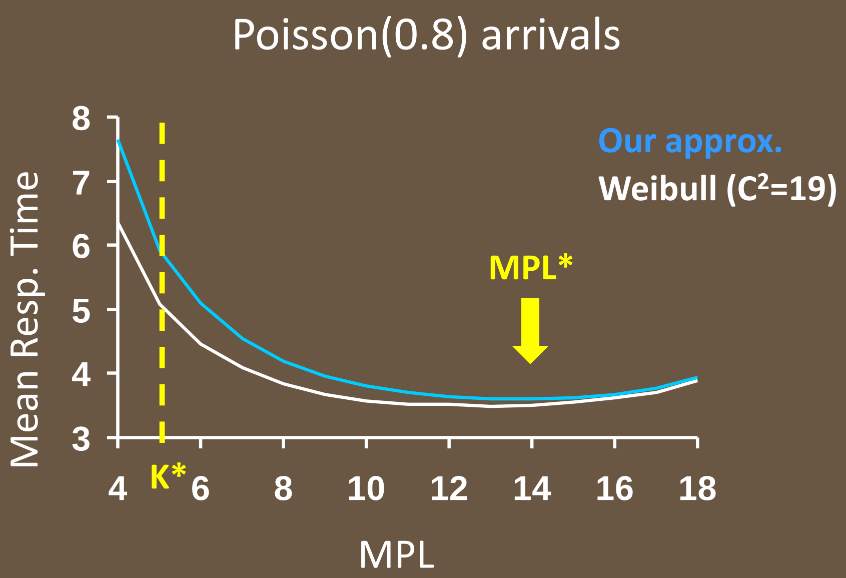
Job size distribution ~ *H**

$$E[T_X] \approx \frac{C^2+1}{2} E[T_{Exp}^Q] + E[T_{Exp}^S]$$

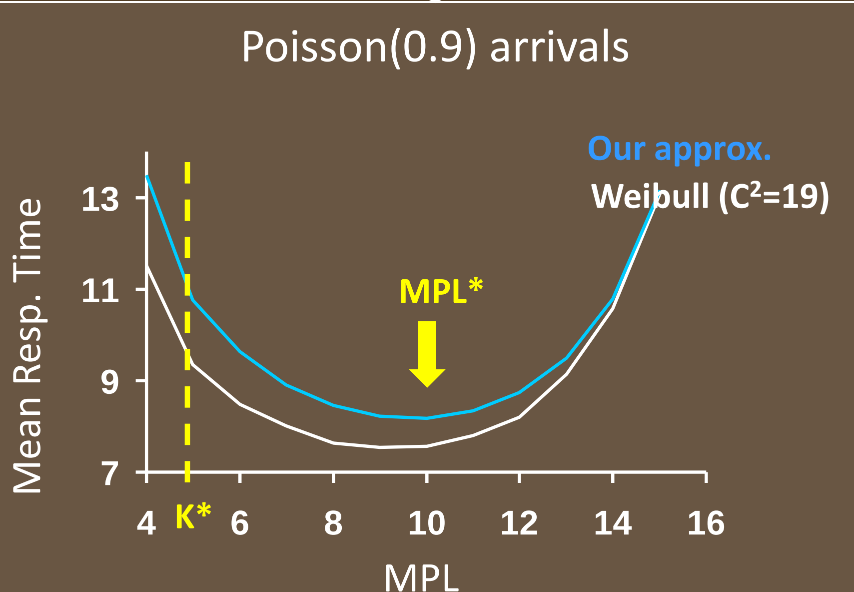
Step 2: Optimizing MPL

Set MPL = MPL*, where:

$$MPL^* = \underset{K}{\operatorname{argmin}} \left\{ \frac{C^2+1}{2} E\left[T^Q_{Exp}(K)\right] + E\left[T^S_{Exp}(K)\right] \right\}$$



K* gives 45% worse performance than MPL*



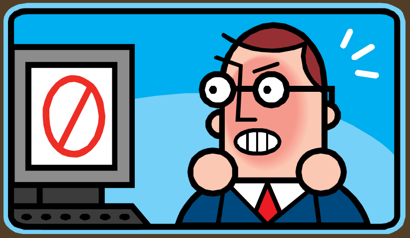
K* gives 25% worse performance than MPL*

- Our approx accurately predicts the *behavior* of the curve, and hence the correct MPL
- Higher arrival rate $\Rightarrow$ MPL* decreases

Going even further…

I don't know the arrival rate!!

My arrivals are not Poisson!!

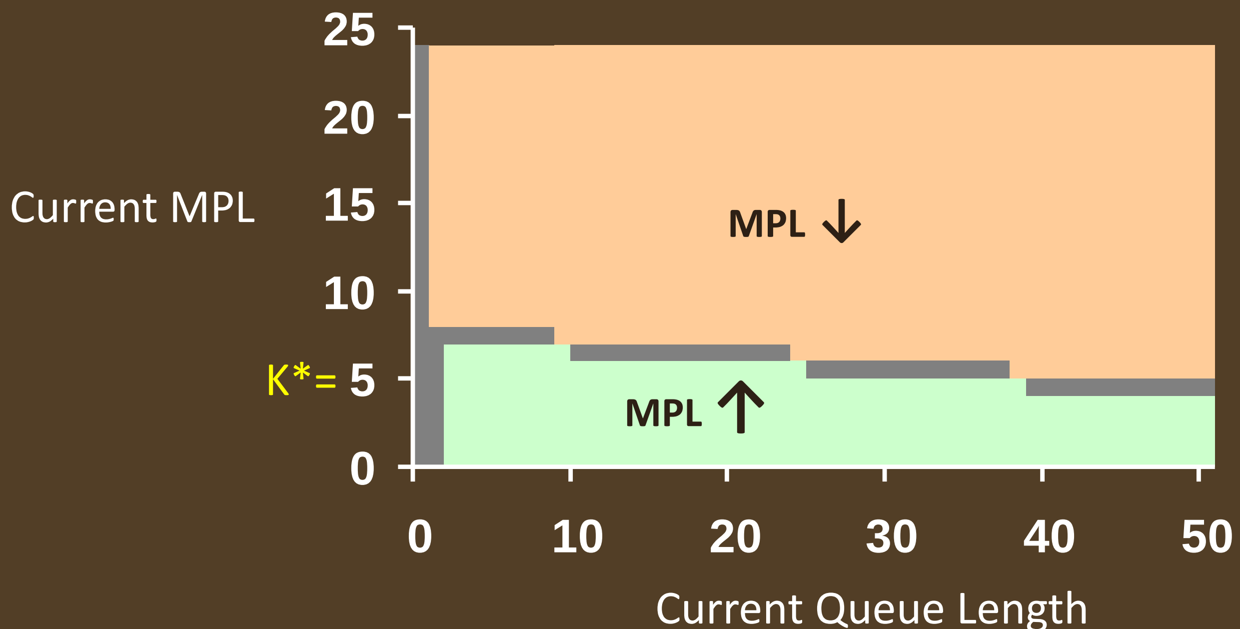


- Straw man proposal 1: Choose a "robust" static MPL
  - Must choose MPL=K* : but suboptimal in light/moderate traffic
- Straw man proposal 2: Learn the arrival rate
  - Can't adapt to changes on small scale/correlations

We Demonstrate: A Dynamic MPL control policy which is

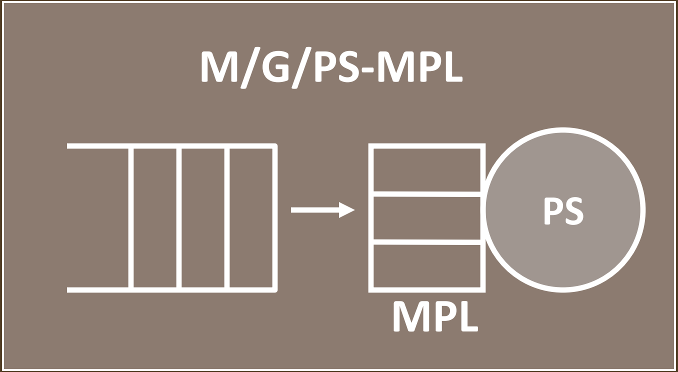
1. **Traffic-oblivious:** self-adapts to variations in the arrival process
2. **Light-weight:** makes decisions based *only* on current queue length, Q(t), and current MPL, K(t)

Structure of our dynamic policy

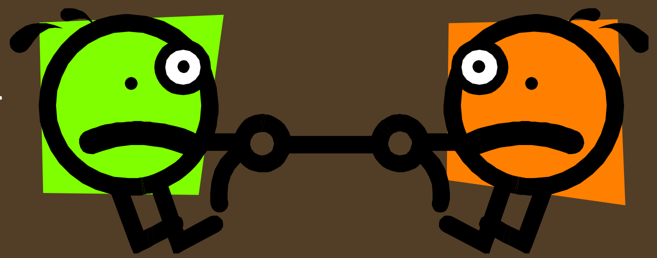


- obtained by combining policy iteration with some new tricks (happy to discuss offline)
- robust to unknown and non-Poisson arrival processes
  - 20% performance loss in the worst case (compared to the *optimal traffic-aware* MPL)
  - MPL=K* becomes worse under non-Poisson arrivals

What we’ve learnt…



large MPL (PS)



small MPL (FCFS)

Job-size variability

Loss in efficiency at high MPL

- Running the system at maximum efficiency is not optimal for mean response time
  - At moderate arrival rate: MPL > K* can result in more than 45% smaller mean response time
- If don’t know arrival process: a dynamic policy can self-adapt while only knowing current queue length and MPL

Real world ≠ Ideal theoretical policies

Reality check 1: Context-switch overheads

(!) Quantum-based Round-Robin

(?) How to choose the optimal quantum size?

Reality check 2: Thrashing

(!) Impose a Multi-Programming-Limit (MPL)

(?) How to choose the optimal MPL?

Reality check 3: Load balancing in server farms

(?) How do load-balancing algorithms interact with servers?

(?) What are good load-balancing algorithms?

Real world ≠ Ideal theoretical policies

Reality check 1: Context-switch overheads

(!) Quantum-based Round-Robin

(?) How to choose the optimal quantum size?

Reality check 2: Thrashing

(!) Impose a Multi-Programming-Limit (MPL)

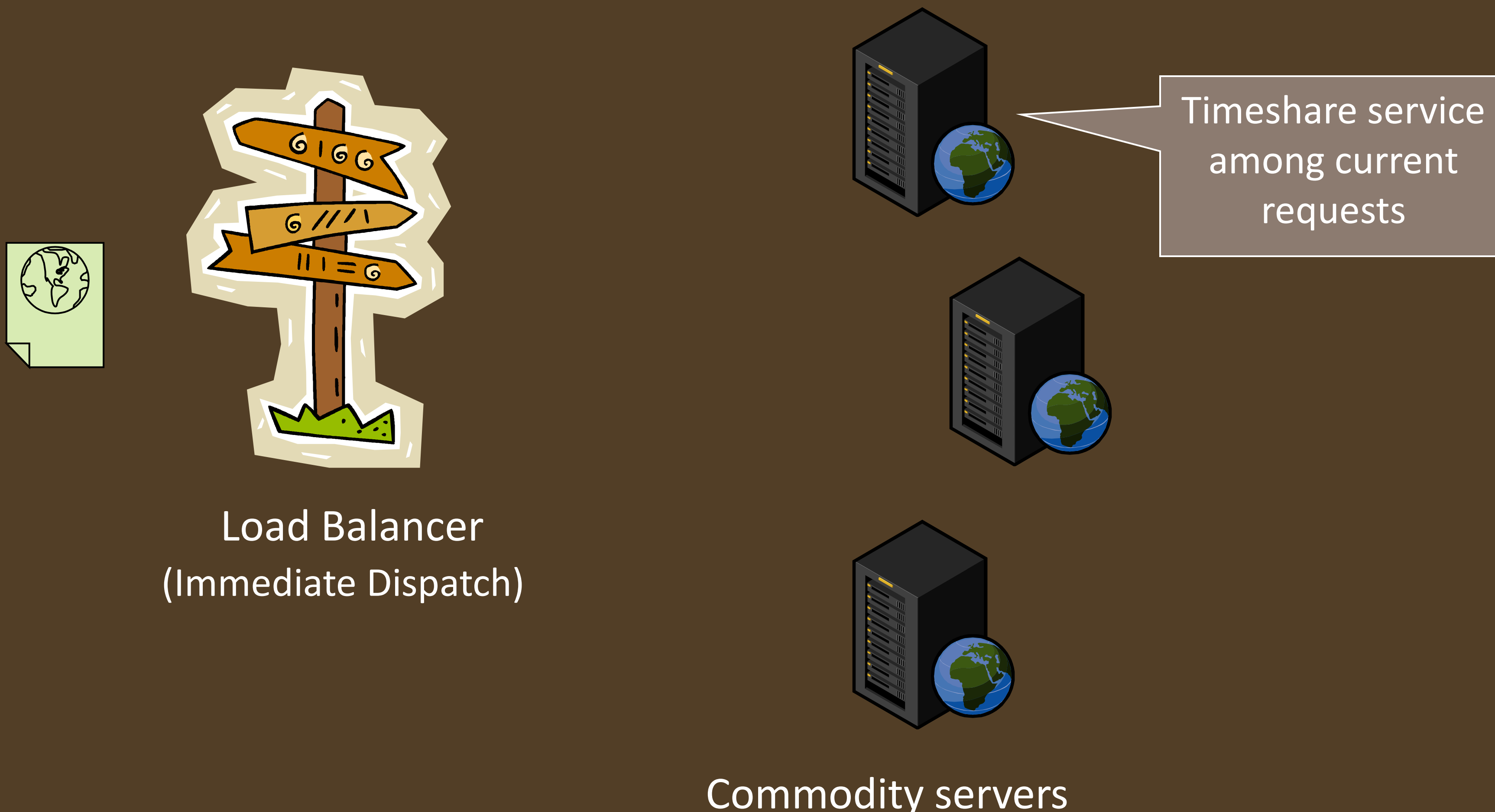
(?) How to choose the optimal MPL?

Reality check 3: Load balancing in server farms

(?) How do load-balancing algorithms interact with servers?

(?) What are good load-balancing algorithms?

A typical Web server farm



Model: PS server farm

Load Balancer
(Immediate Dispatch)

Timeshare service among current requests

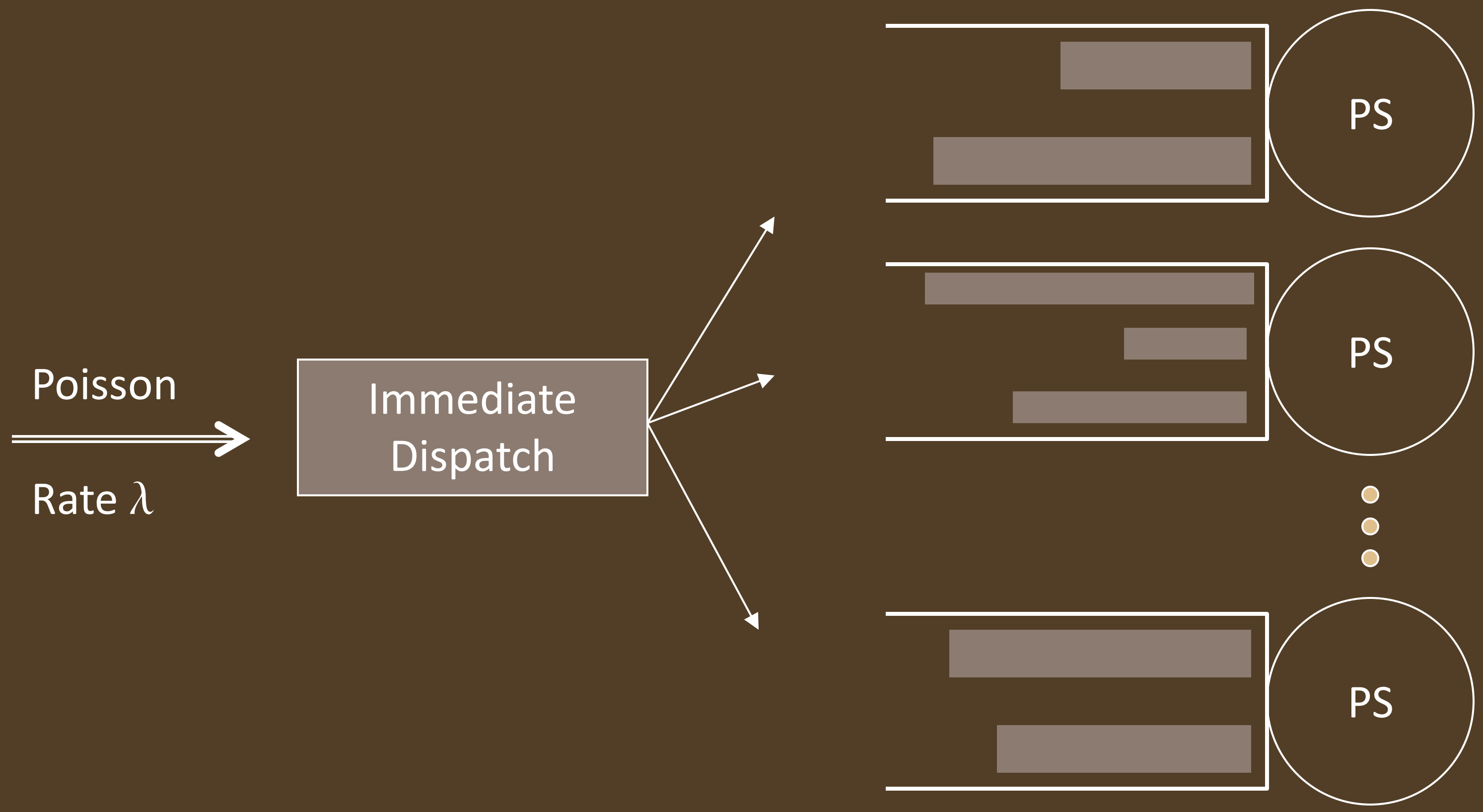
Commodity servers

Model: PS server farm

Load Balancer
(Immediate Dispatch)

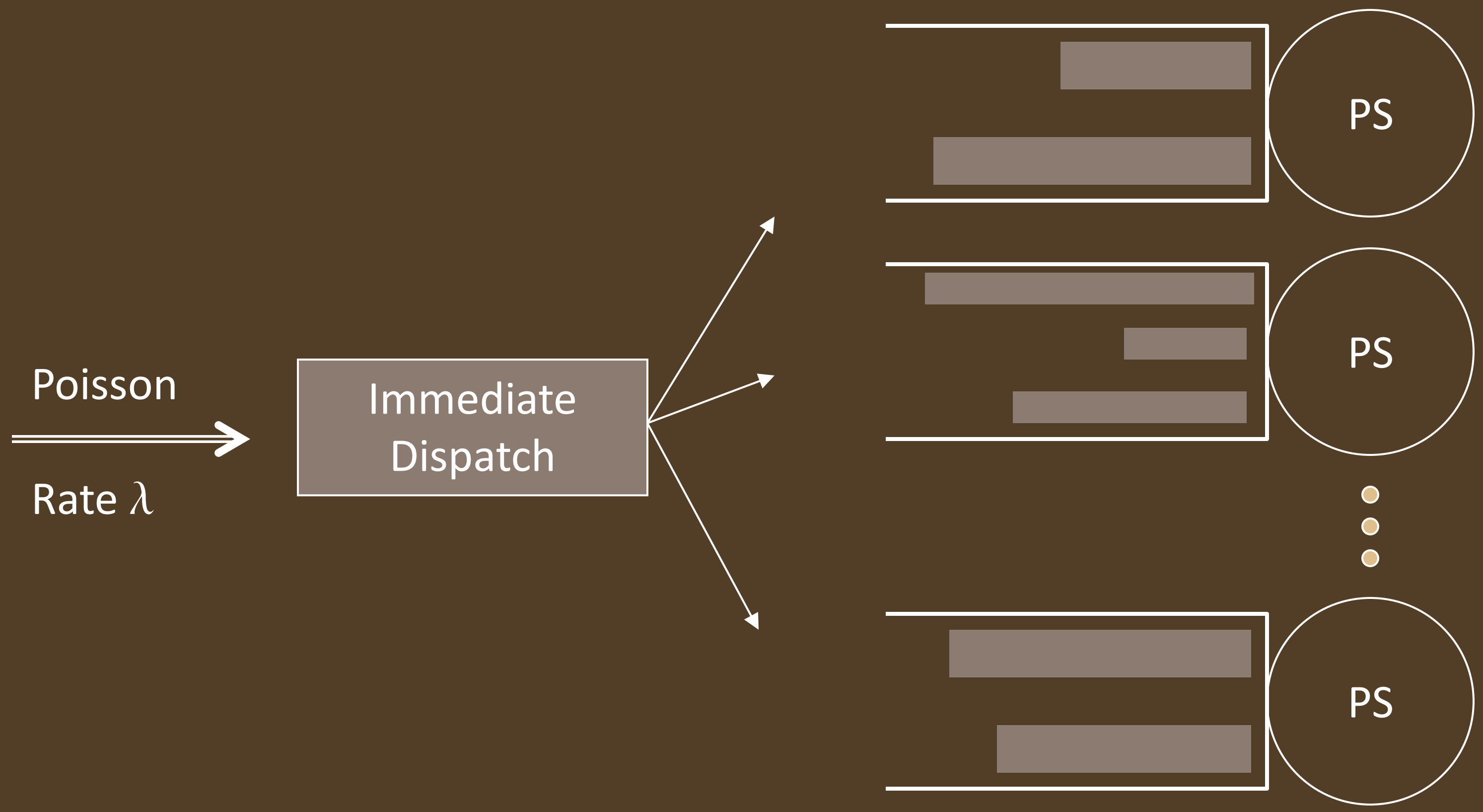
- K homogeneous, PS servers

Model: PS server farm



- K homogeneous, PS servers
- Poisson arrivals
- Job sizes i.i.d. ~ X

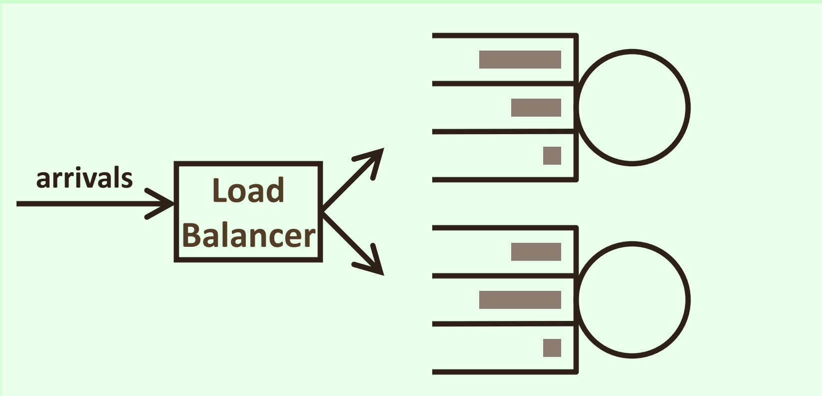
Model: PS server farm

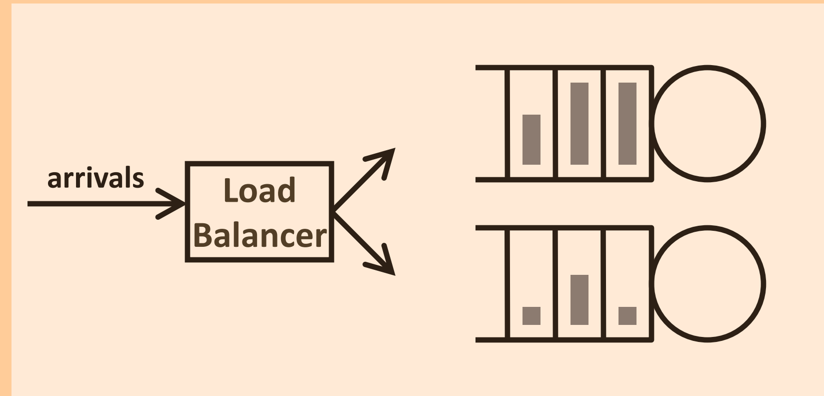


GOAL

Good Load balancing algorithms for PS server farms

PS server farms vs. FCFS server farms

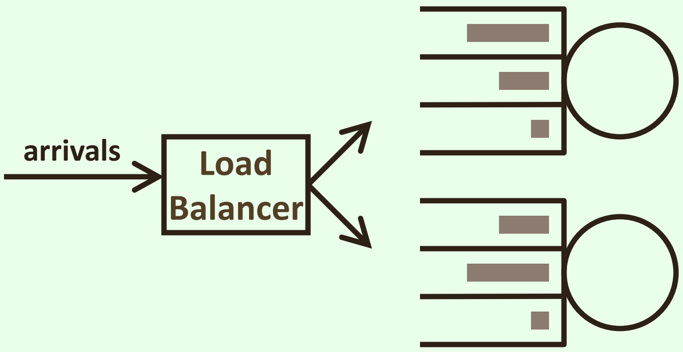




Which is a good FCFS load balancer?
(Hint: your local supermarket)

- ☐ Random
- ☐ Round-Robin
- ☐ Least-Work-Left
- ☐ Size-based-splitting
- ☐ Shortest Queue

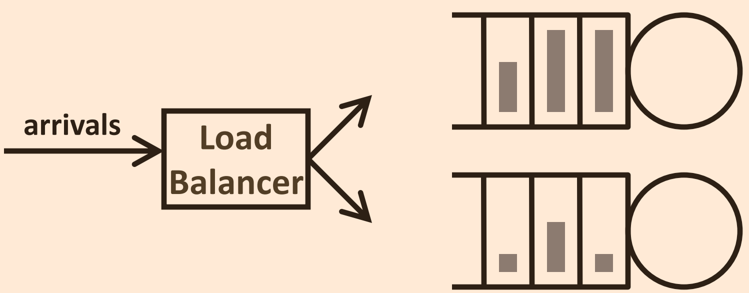
PS server farms vs. FCFS server farms



Which is a good PS load balancer?

☐ Random
☐ Round-Robin
☐ Least-Work-Left
☐ Size-based-splitting
☐ Shortest Queue ⇐ greedy!

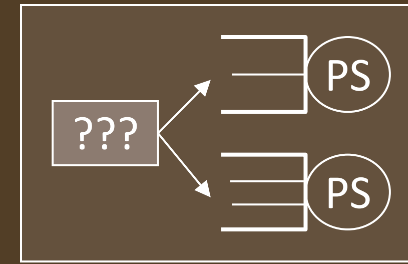
same perf.

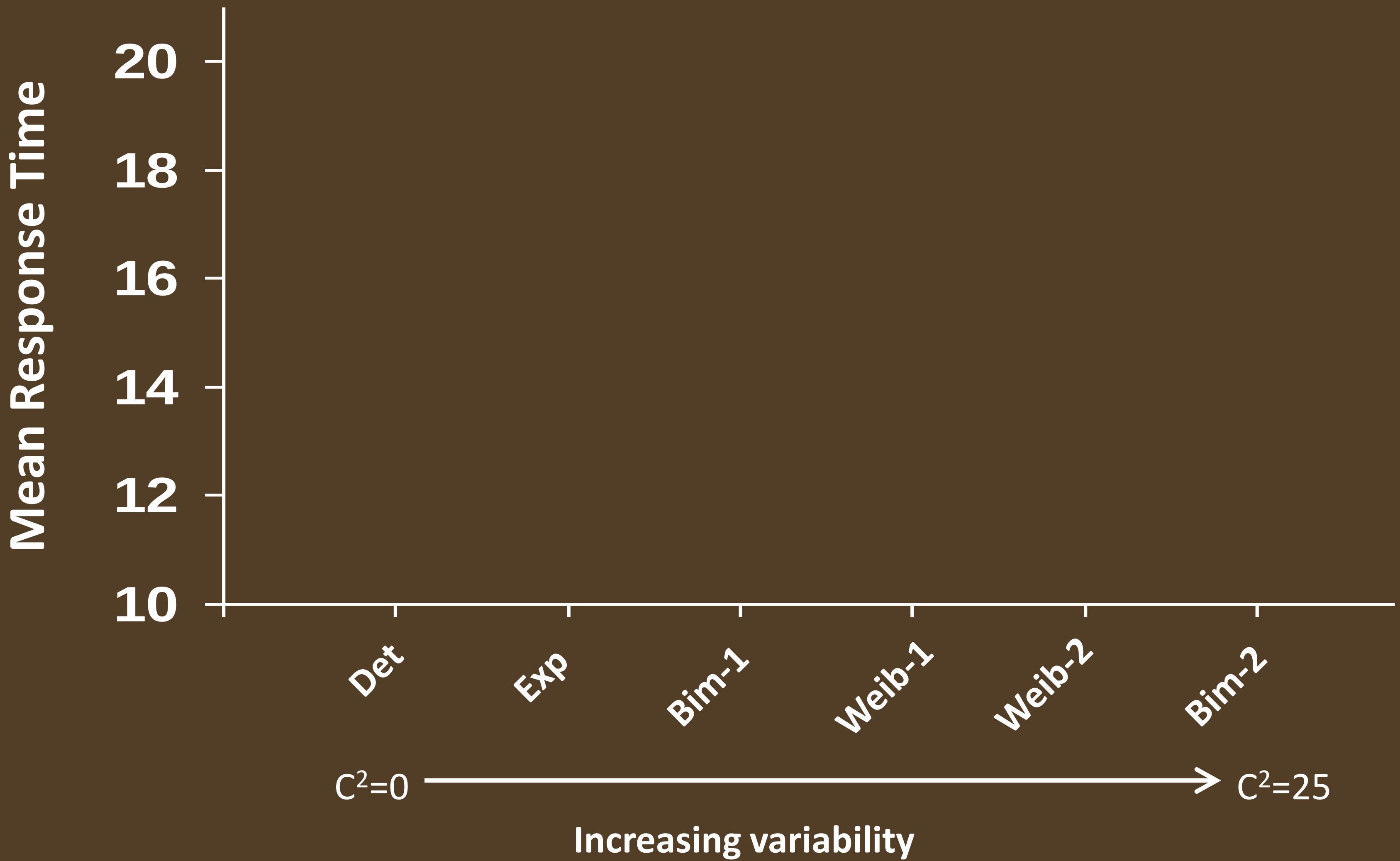


Which is a good FCFS load balancer?
(Hint: your local supermarket)

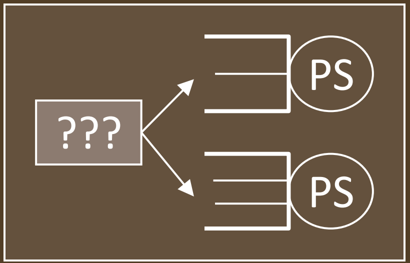
☐ Random
☐ Round-Robin
☑ Least-Work-Left ⇐ greedy!
☑ Size-based-splitting ⇐ reduces C^2
☐ Shortest Queue

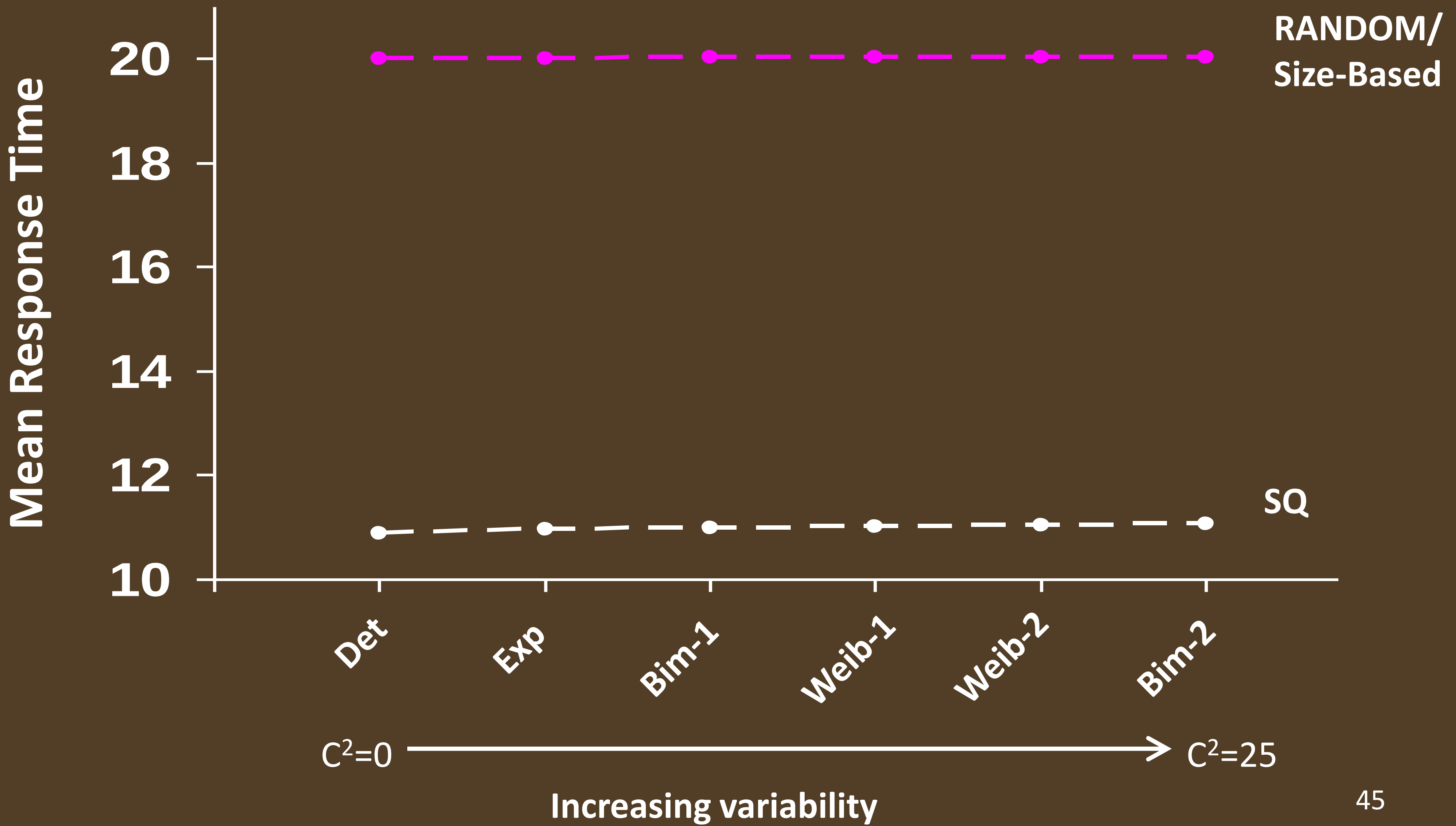
Why?


???
PS
PS

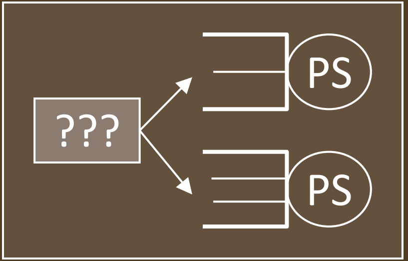

Mean Response Time
20
18
16
14
12
10
Det
Exp
Bim-1
Weib-1
Weib-2
Bim-2
C2=0
C2=25
Increasing variability

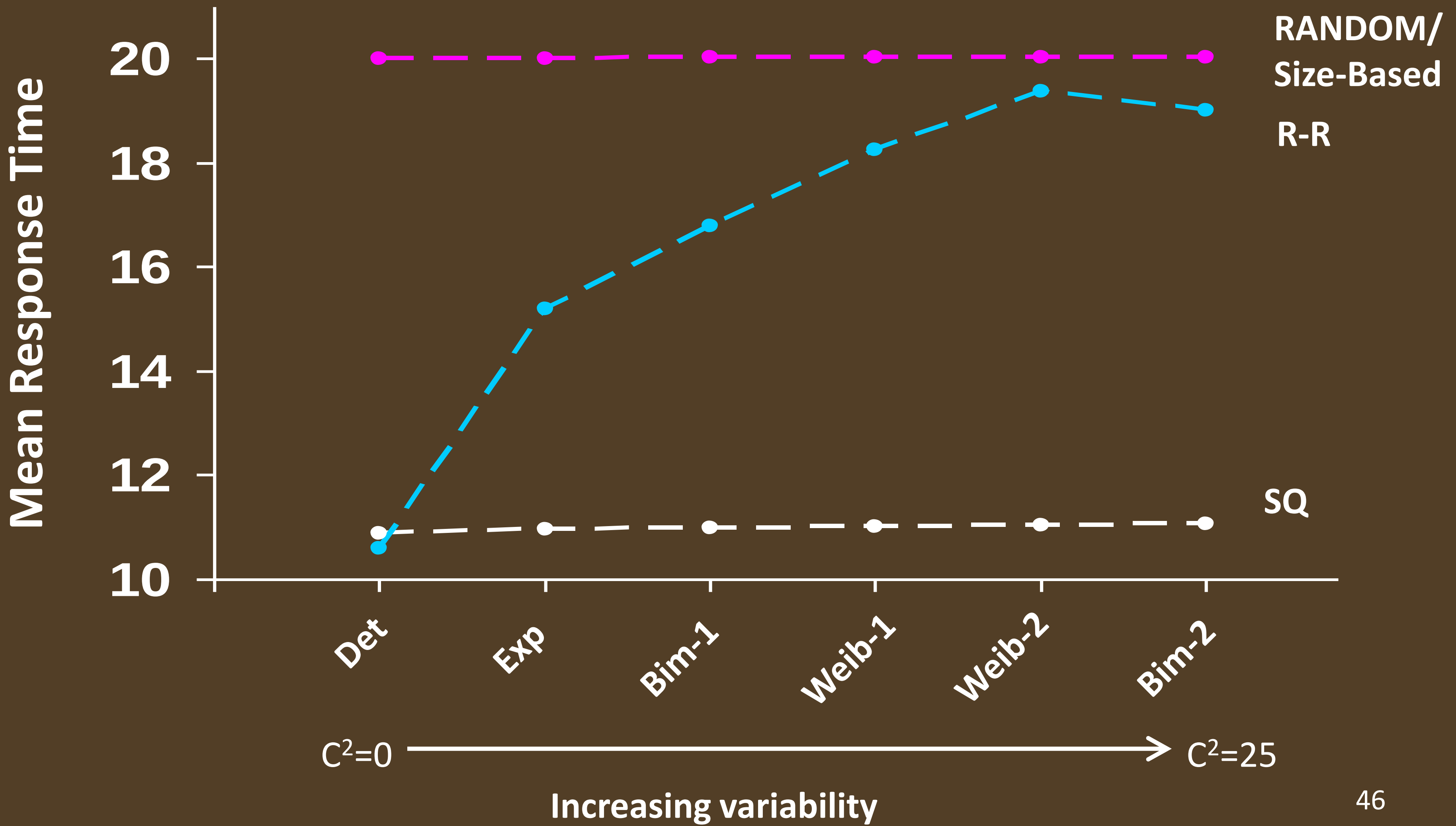
E[T] under SQ/PS is **"nearly insensitive"** to the variability of job size distribution



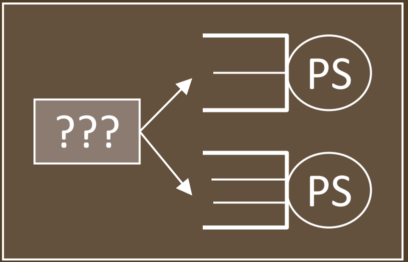


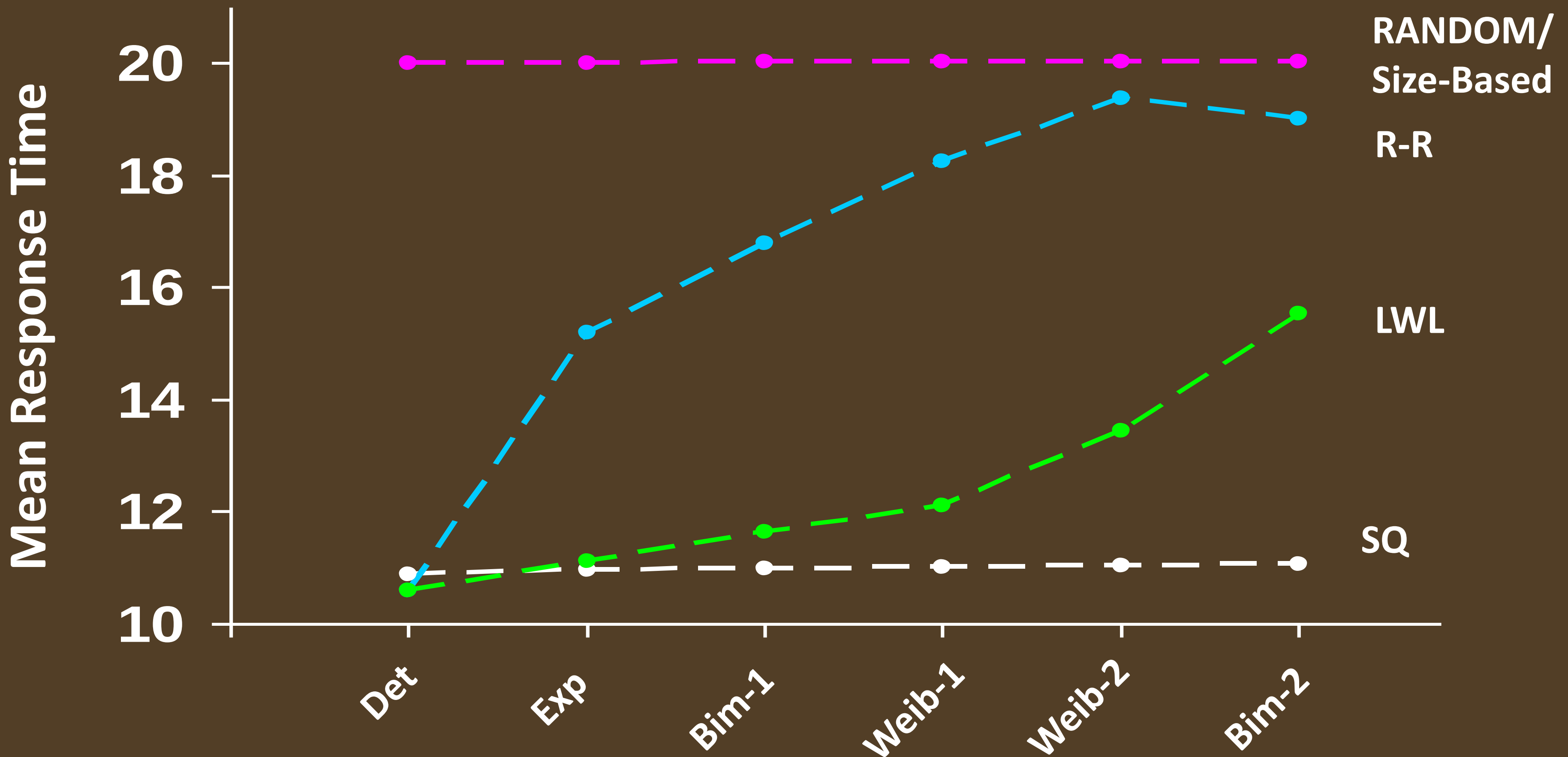
E[T] under SQ/PS is "nearly insensitive" to the variability of job size distribution





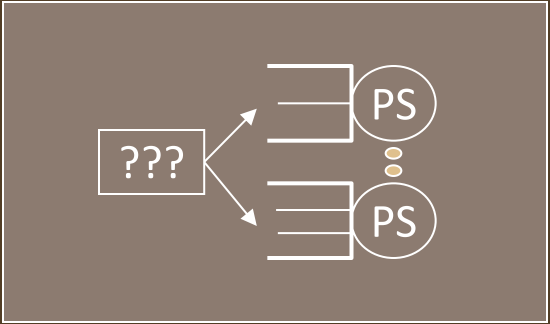
E[T] under SQ/PS is **"nearly insensitive"** to the variability of job size distribution





CONJECTURE: SQ load balancer is **"nearly optimal"** for PS servers

What we’ve learnt…



- Good load balancers for FCFS and PS servers are different!
  - Least-Work-Left and Size-based-splitting are bad for PS !
- Shortest Queue (SQ) load balancing is ‘near-optimal’ for PS servers
  - *Independent* of job size distribution

- Shortest Queue (SQ) load balancing ‘preserves’ insensitivity of PS to job-size variability

Bridging the gap between practice and theory

1: Quantum-based Round-Robin

- Overheads matter – Ideal PS a bad model
- Right quantum size is important
- We give expression for OPT quantum

2: Systems with thrashing

- Running system at max efficiency not always optimal
- We find OPT MPL
- Dynamic policies can self-adapt to unknown arrival processes

3: Load balancing for PS server farms

- Scheduling policy of backend servers is integral for choosing load balancer
- Shortest Queue (SQ) is near optimal for PS servers – *independent of job size distribution*