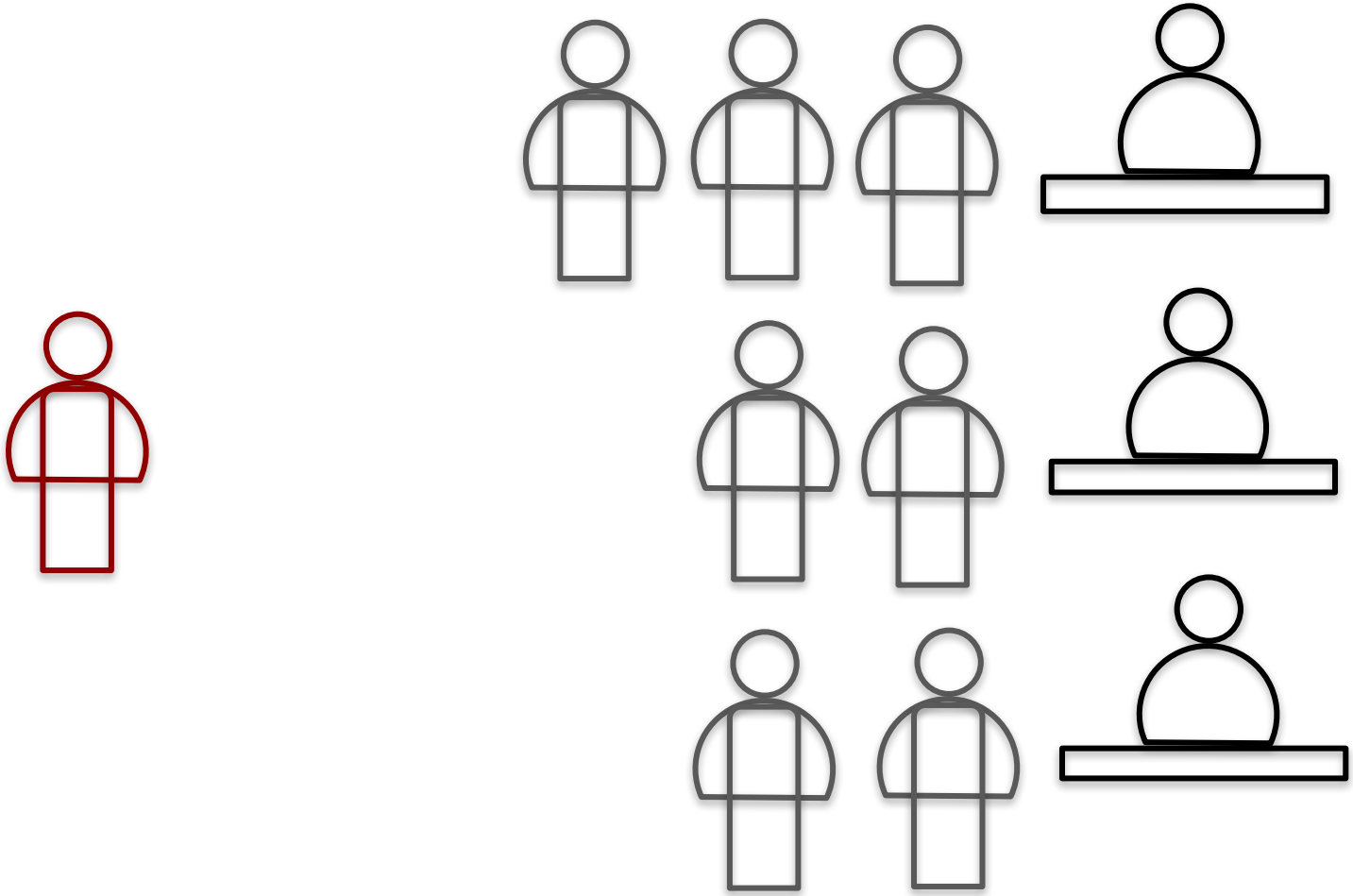
Using Efficient Redundancy to Reduce Latency in Cloud Systems

Gauri Joshi

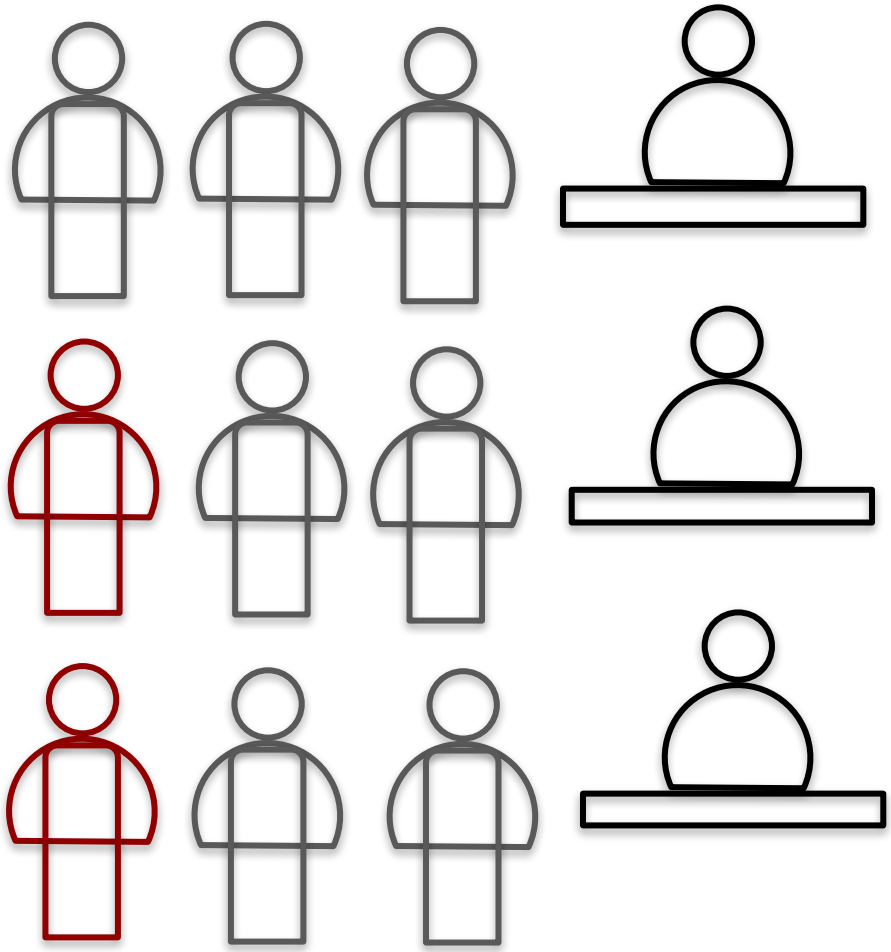
joint work with Emina Soljanin, Gregory Wornell

Ticket Counter Queues



Ticket Counter Queues

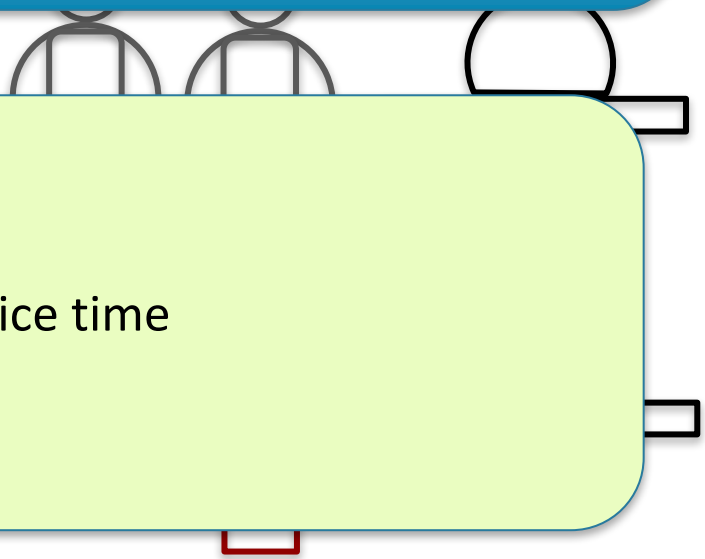
Abandon when any one reaches the head of the queue



Ticket Counter Queues

Q1: Wait for 1 person to reach the counter, or until they actually get the tickets?

Q2: How many, and which queues should we join?



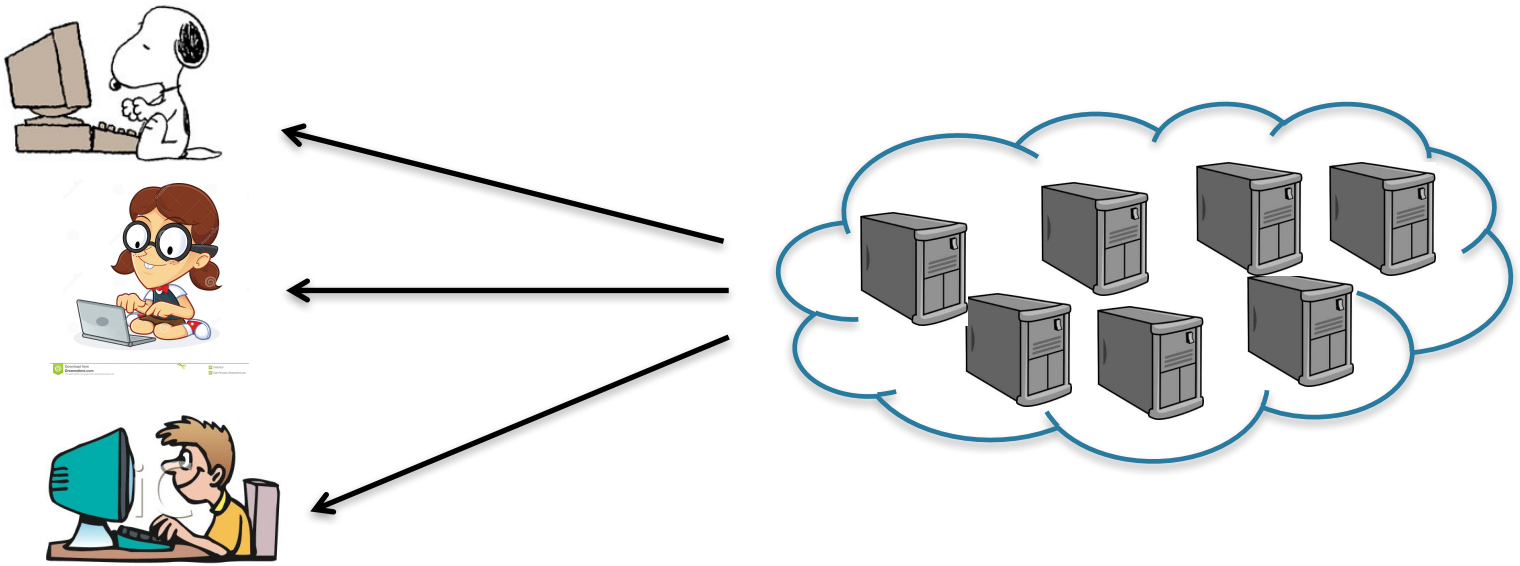
May
wait
pers
the t

Answers depend on:

- Randomness of the service time
- Customer arrival rate

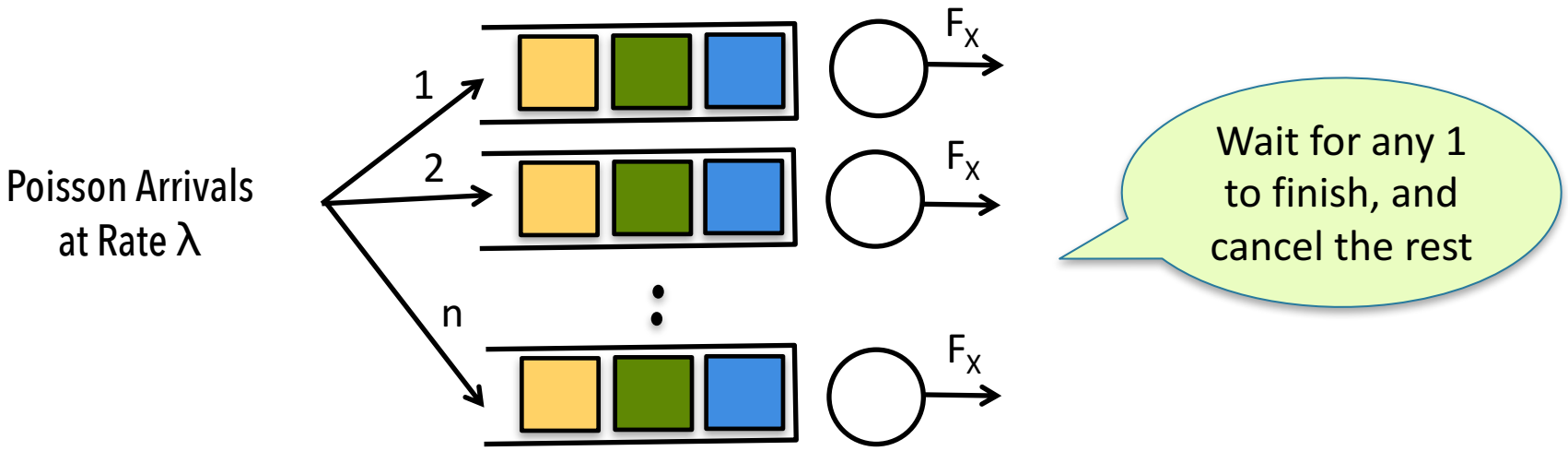
Replication to Reduce Latency in the Cloud

- Large-scale resource sharing → Variability in service time
  - Virtualization, Server outages, Network Packet Loss



The (n,1) fork-join model

- Fork a job into tasks at n servers, and wait for any one to finish
- Cancel the redundant tasks immediately
- Each task takes time X to finish, $X \sim F_X$ i.i.d. across servers



(n,k) fork-join: Erasure Coded Storage, or Approx. Computing
[Joshi-Liu-Soljanin 2012, 14]
k = n: Famously hard fork-join queue

Variants of the (n,1) fork-join system

(n, 1) fork-early-cancel system

Fork to all n servers

Cancel redundant tasks as soon as 1 task begins service

(n, r, 1) partial-fork-join system

Fork into tasks at r out of n servers.

Wait for any 1 task to finish and cancel the rest

Can reduce cost, but we lose the diversity provided by redundancy

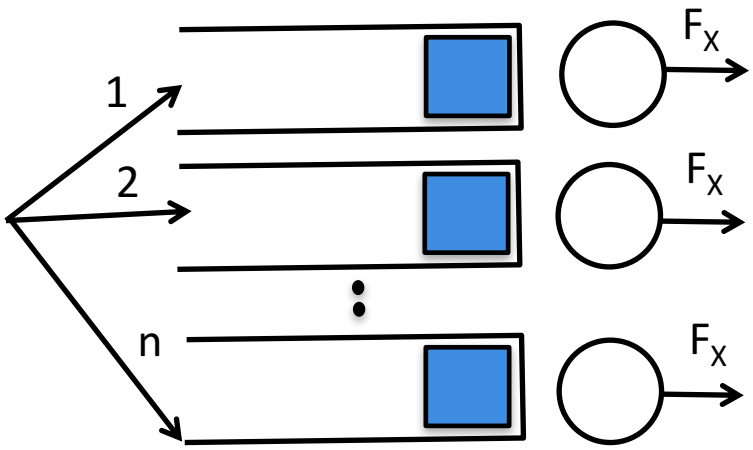
Performance Metrics

Expected Latency E[T]

Expected time from arrival until any 1 task is served
= Waiting time in queue + Service Time

Expected Computing Cost E[C]

Total expected time spent by servers per job.
Does not include waiting time in queue



No Queueing

$$\mathbb{E}[T] = \mathbb{E}[X_{1:n}]$$

$$\mathbb{E}[C] = n\mathbb{E}[X_{1:n}]$$

minimum of
n i.i.d rvs
$X_1, X_2, .. X_n$

Related Previous Work

Queues with Redundancy

- (n,k) fork-join with exponential service time [Joshi-Liu-Soljanin 2012,14]
- Exponential service time, heterogeneous jobs [Gardner et al 2015]
- When is it better to fork to all n servers?
  [Shah-Lee-Ramchandran 2013] [Koole-Righter 2008]

Contributions

- Impact of redundancy on the latency, and the computing cost
- 'Log-concavity' of the service time is a key factor
- Using E[C] to compare systems in high traffic regime

Outline

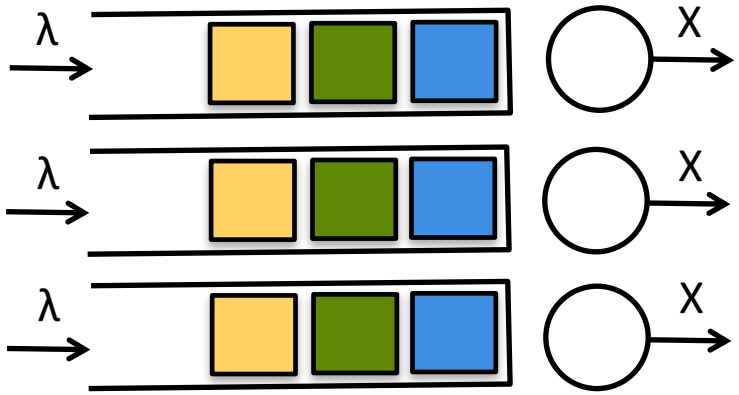
- Forking to n and waiting for 1 to finish
  - How redundancy affects E[T] and E[C]?
- Canceling redundant tasks as soon as 1 starts
  - When is early cancellation better?
- Partial forking to r out of n servers
  - Optimal choice of r
  - Which r servers to fork to?

Outline

- Forking to n and waiting for 1 to finish
  - How redundancy affects E[T] and E[C]?
- Canceling redundant tasks as soon as 1 starts
  - When is early cancellation better?
- Partial forking to r out of n servers
  - Optimal choice of r
  - Which r servers to fork to?

(n,1) Fork-join system

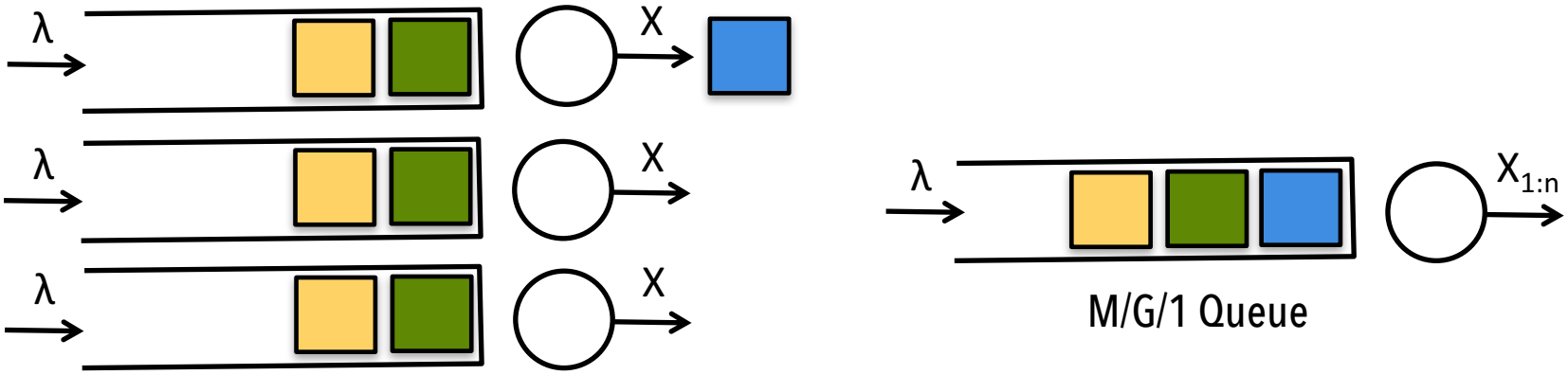
Forking to n and waiting for 1 to finish



The n tasks of each job start at the same time!

Equivalent to an M/G/1 Queue!

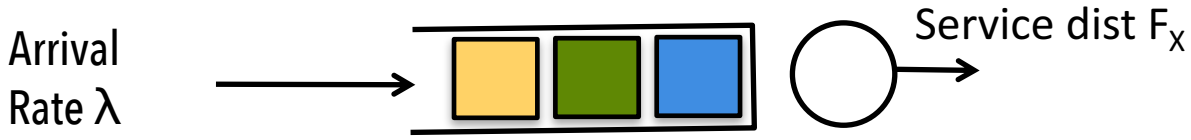
Forking to n and waiting for 1 to finish



The n tasks of each job start at the same time!

M/G/1 Queue

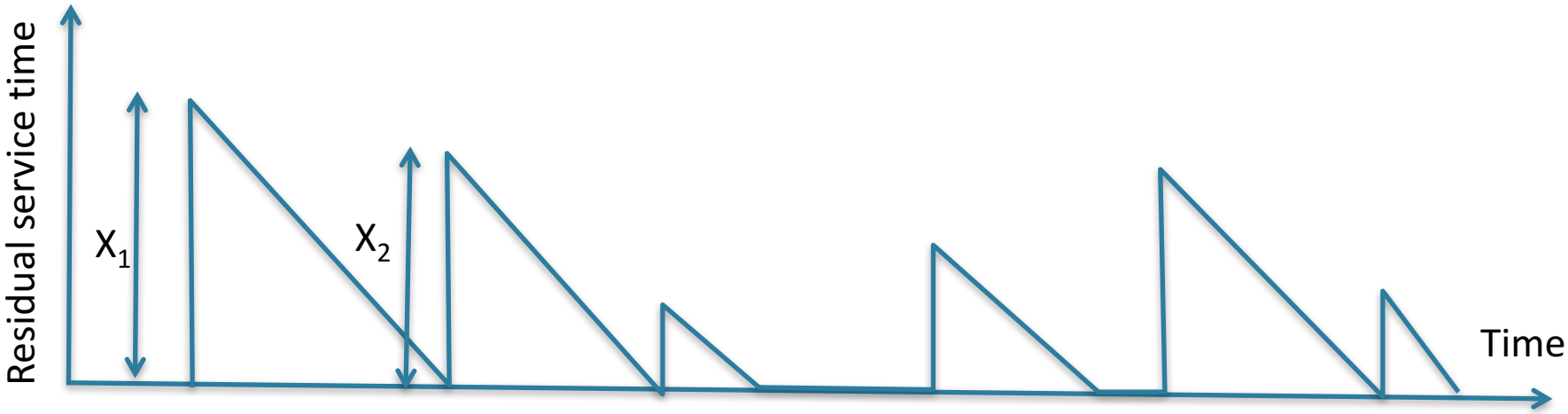
Pollaczek-Khinchine Formula



Pollaczek-Khinchine Formula

$$\mathbb{E}[T] = \mathbb{E}[X] + \frac{\mathbb{E}[X^2]}{2(1 - \lambda\mathbb{E}[X])}$$

Proof of PK formula



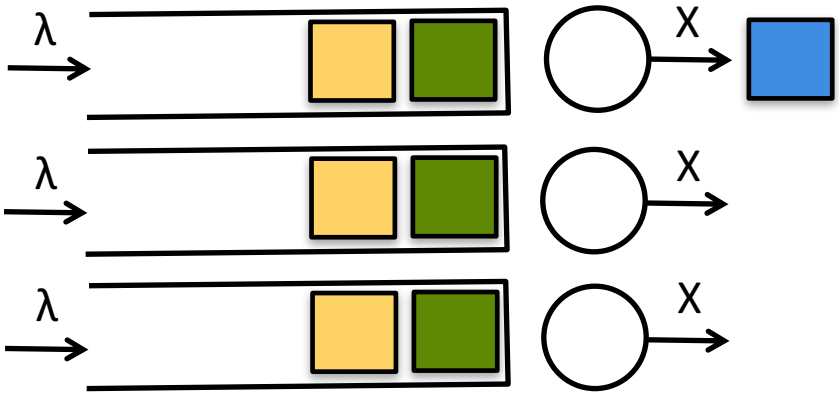
$$\mathbb{E}[T_w] = \mathbb{E}[N_w] \cdot \mathbb{E}[X] + E[R]$$

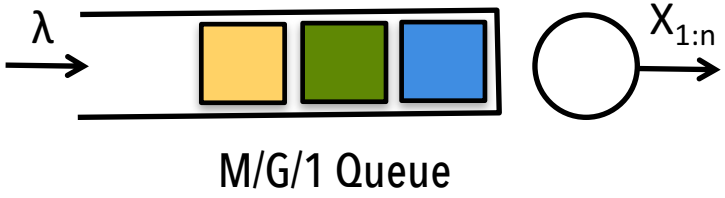
$$= \lambda \mathbb{E}[T_w] \cdot \mathbb{E}[X] + \frac{\mathbb{E}[X^2]}{2}$$

$$= \frac{\mathbb{E}[X^2]}{2(1 - \lambda \mathbb{E}[X])}$$

What are E[T] and E[C]?

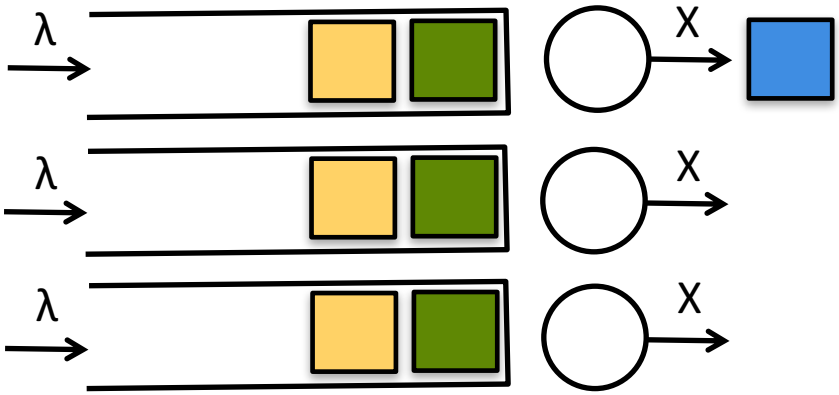
Forking to n and waiting for 1 to finish

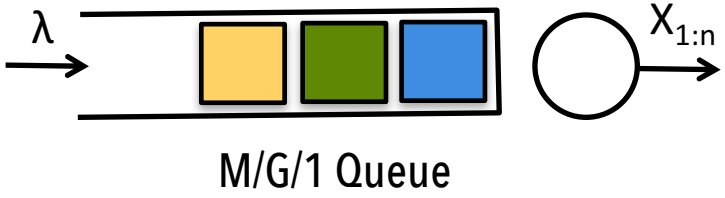




What are E[T] and E[C]?

Forking to n and waiting for 1 to finish





M/G/1 Queue

= E[X] without replication

$$\mathbb{E}[C] = n\mathbb{E}[X_{1:n}]$$

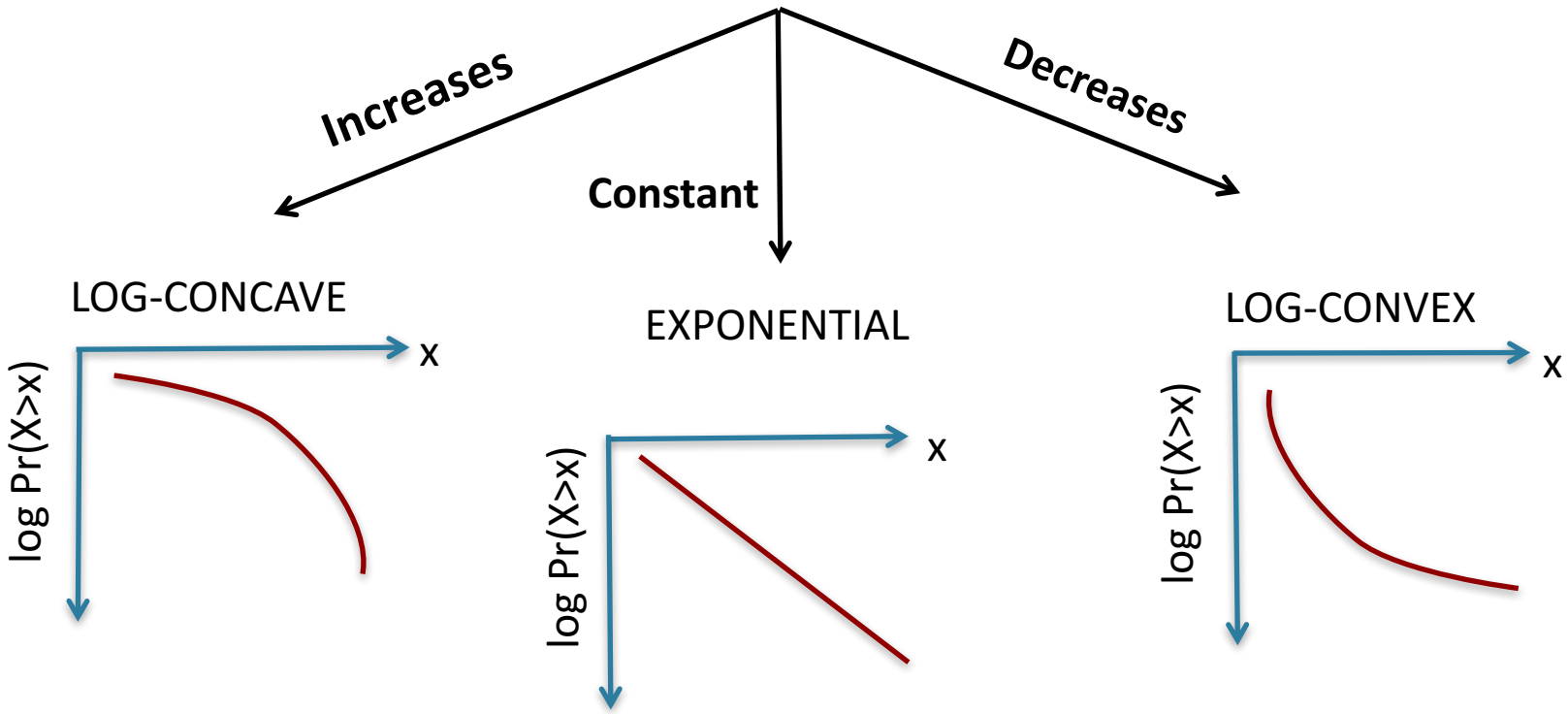
$$\mathbb{E}[T] = \mathbb{E}[X_{1:n}] + \frac{\lambda \mathbb{E}[X_{1:n}^2]}{2(1 - \lambda \mathbb{E}[X_{1:n}])}$$

Pollaczek-Khinchine Formula

How does cost E[C] vary with r?

$$\mathbb{E}[C] = r\mathbb{E}[X_{1:r}]$$

[GJ-Soljanin-Wornell Allerton 2015]



- Some distributions are neither log-concave nor log-convex
- Studied in reliability theory, economics

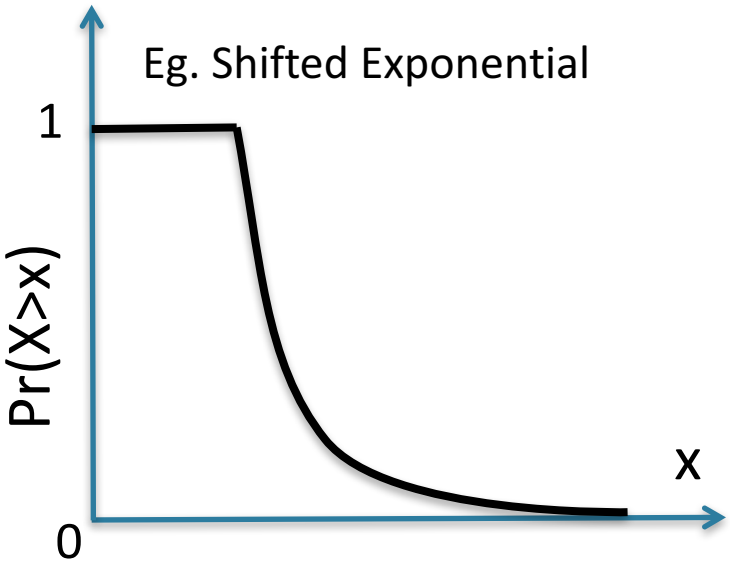
Properties and Examples

LOG-CONCAVE Pr(X>x)

Optimistic Memory

$$\Pr(X > x + t | X > t) \leq \Pr(X > x)$$

The more you wait, the time remaining is shorter

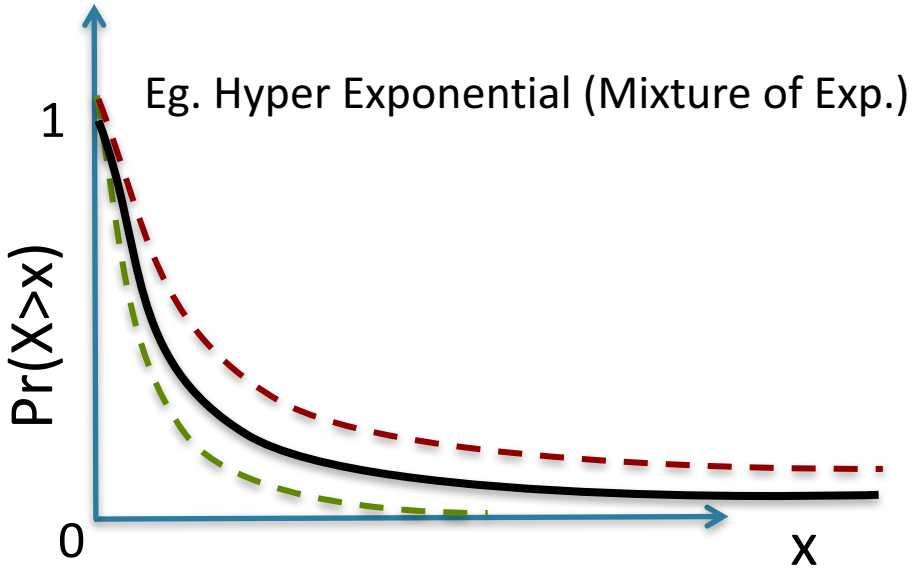


LOG-CONVEX Pr(X>x)

Pessimistic Memory

$$\Pr(X > x + t | X > t) \geq \Pr(X > x)$$

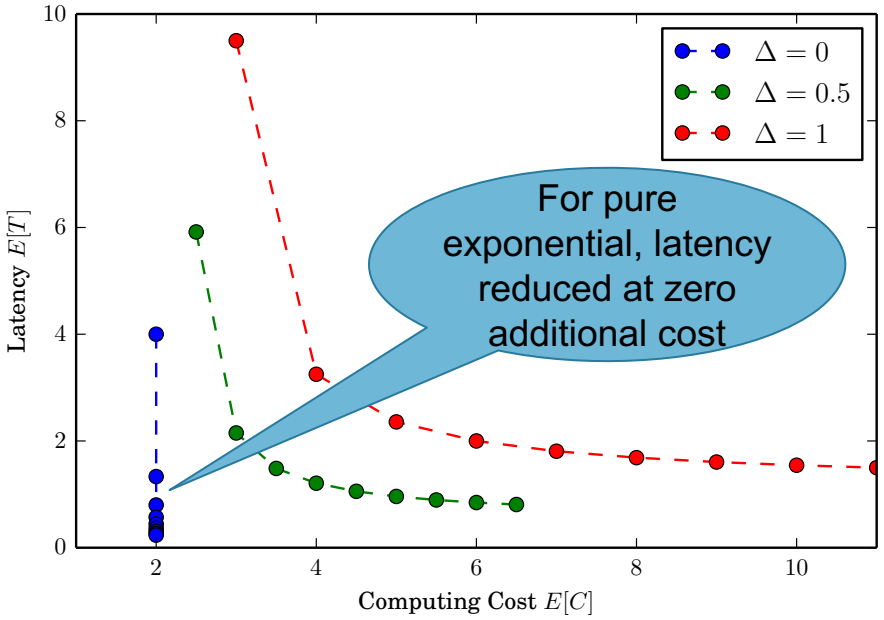
The more you wait, the time remaining is longer



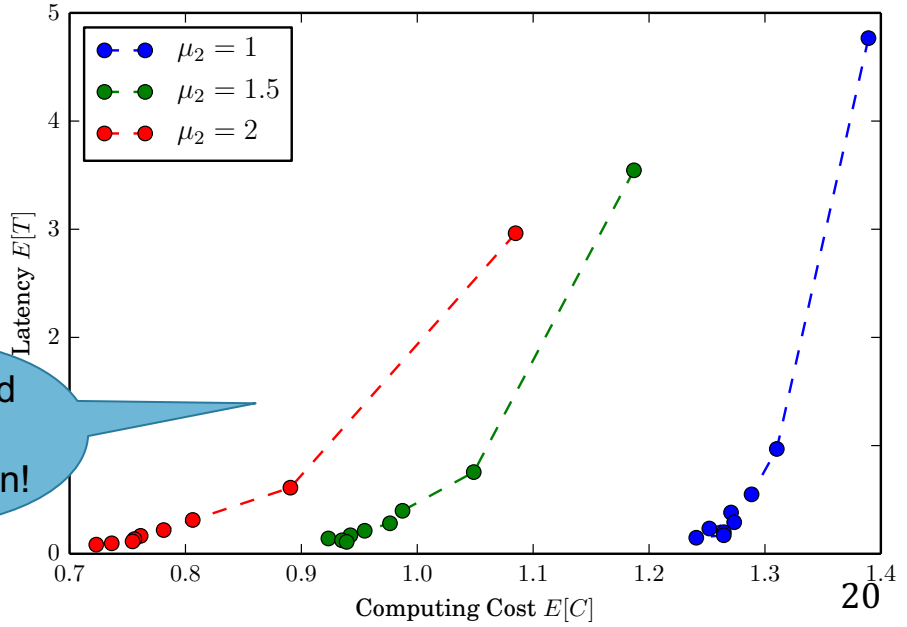
Latency vs. Cost as n varies

Forking to n and waiting for 1 to finish

λ = 0.25, Log-concave X ~ Δ + Exp(0.5), varying n



Log-convex X ~ HyperExp (p =0.4, 0.5, diff μ_2)

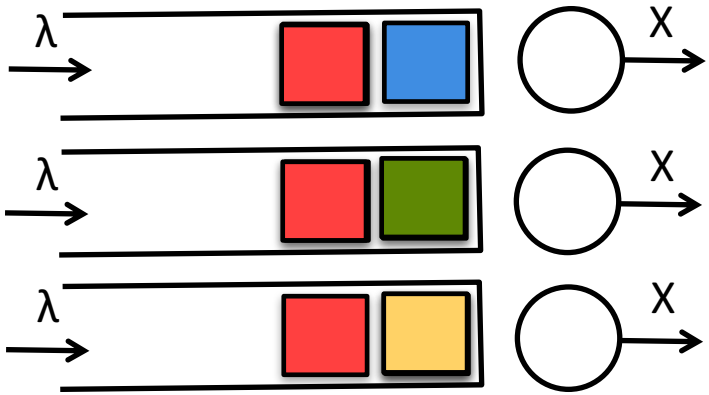


Outline

- Forking to n and waiting for 1 to finish
  - When does redundancy reduce both E[T] and E[C]?
- Canceling redundant tasks as soon as 1 starts
  - When is early cancellation better?
- Partial forking to r out of n servers
  - Optimal choice of r
  - Which r servers to fork to?

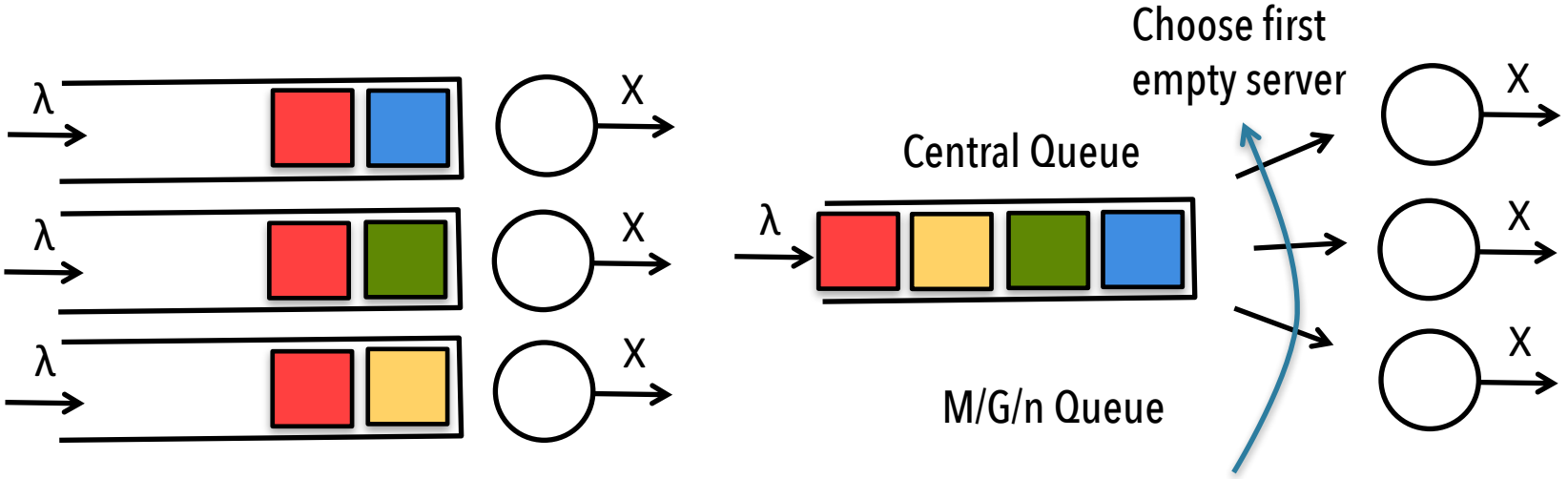
(n,1) Fork-early-cancel System

Canceling redundant tasks when any 1 task starts service



Equivalent to an M/G/n queue!

Canceling redundant tasks when 1 task starts service



$$\mathbb{E}[T] \approx \mathbb{E}[X] + \frac{\mathbb{E}[X^2]}{2\mathbb{E}[X]} \cdot \mathbb{E}[W^{M/M/n}]$$

[Lee-Loughton 1959]

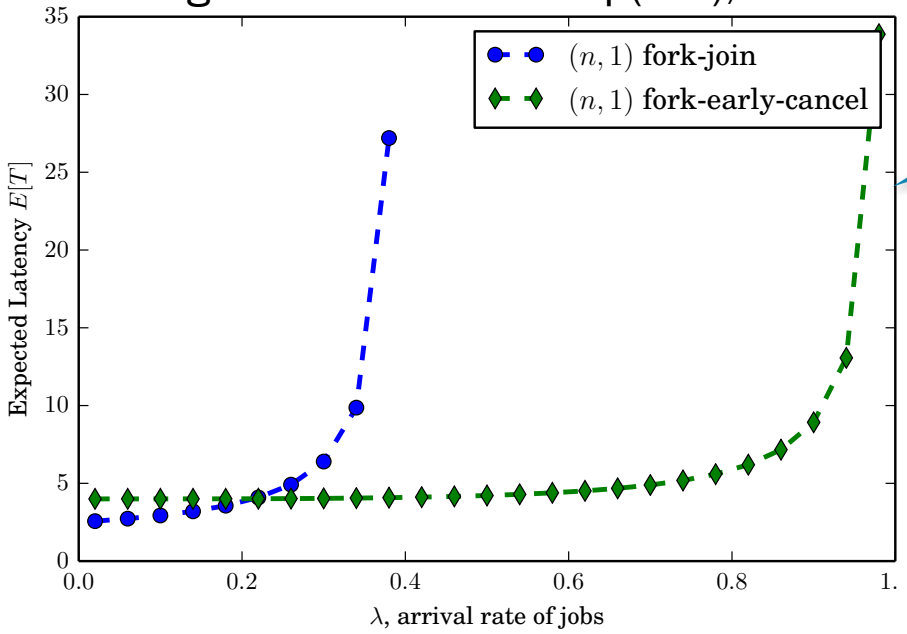
$$\mathbb{E}[C] = \mathbb{E}[X]$$

Waiting time of M/M/n queue

< $nE[X_{1:n}]$ for log-concave X

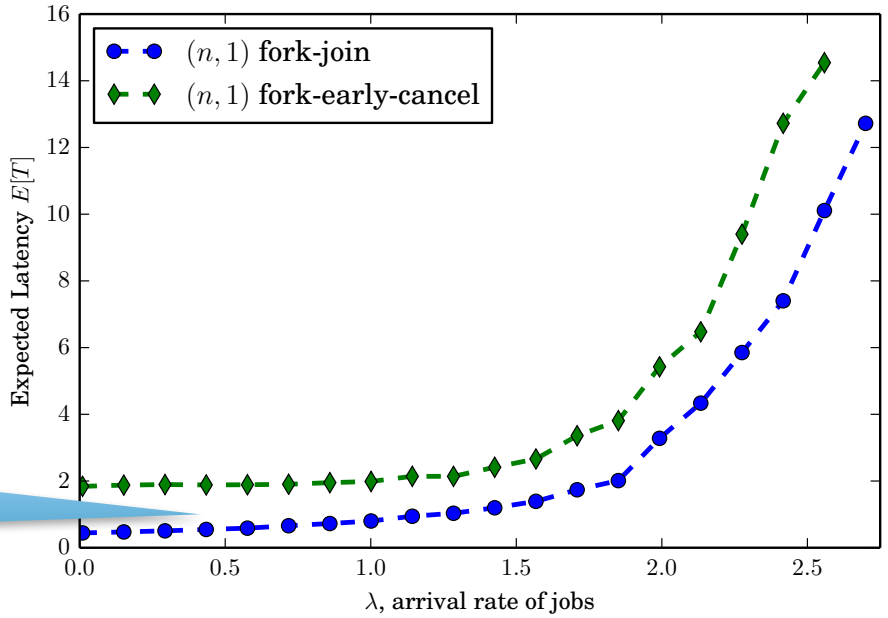
Cancel Early or Keep Redundancy?

Log concave X ~ 2 + Exp(0.5), n = 4



Early cancel gives lower E[C]
→ lower E[T] in high load

Log-convex X ~ HyperExp(p=0.1, 1.5, 0.5)



Keeping Redundancy better
for both low and high loads

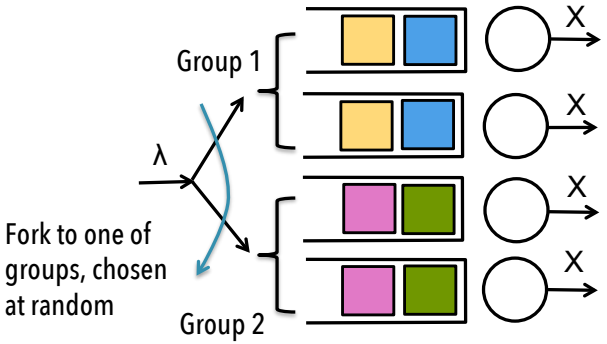
Outline

- Forking to n and waiting for 1 to finish
  - How redundancy affects E[T] and E[C]?
- Canceling redundant tasks as soon as 1 starts
  - When is early cancellation better?
- Partial forking to r out of n servers
  - Optimal choice of r
  - Which r servers to fork to?

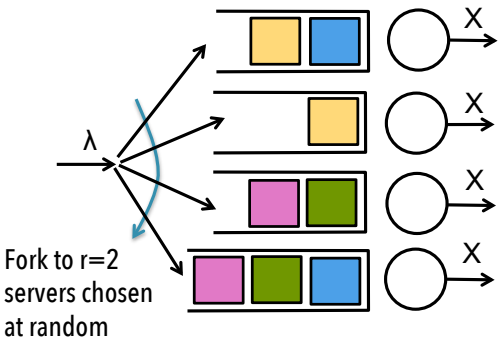
Partial forking to r out of n servers

Questions:

- How many servers to fork to? (Optimal r)
- Which r servers to fork to? (Optimal Scheduling Policy)

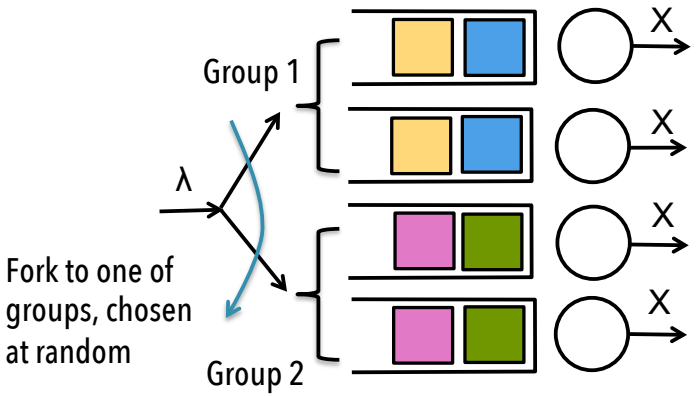


Group-based random assgnt



Uniform random assgnt

Group-based Random Policy



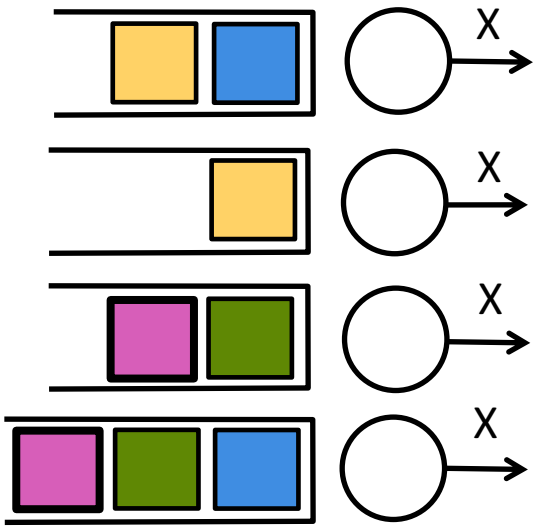
- The r tasks of each job start at the same time
- Each group behaves as an independent (r,1) fork-join system with arrival rate λr/ n

$$\mathbb{E}[C] = r\mathbb{E}[X_{1:r}]$$

$$\mathbb{E}[T] = \mathbb{E}[X_{1:r}] + \frac{\lambda r \mathbb{E}[X_{1:r}^2]}{2(n - \lambda r \mathbb{E}[X_{1:r}])}$$

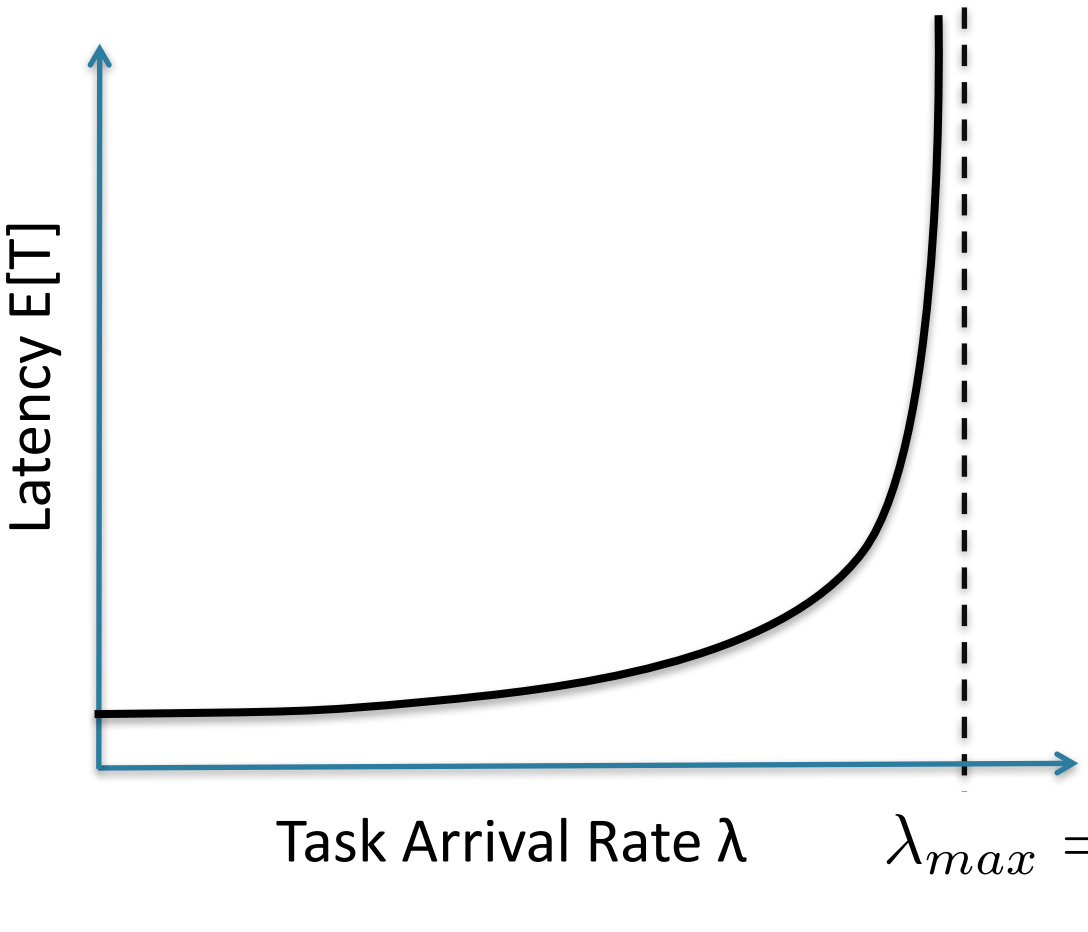
Evaluating E[T] is hard for other policies

- Depends on how many tasks are in front of each replica
- Also, the locations of the replicas of the tasks in front



λ_{max} in terms of E[C]

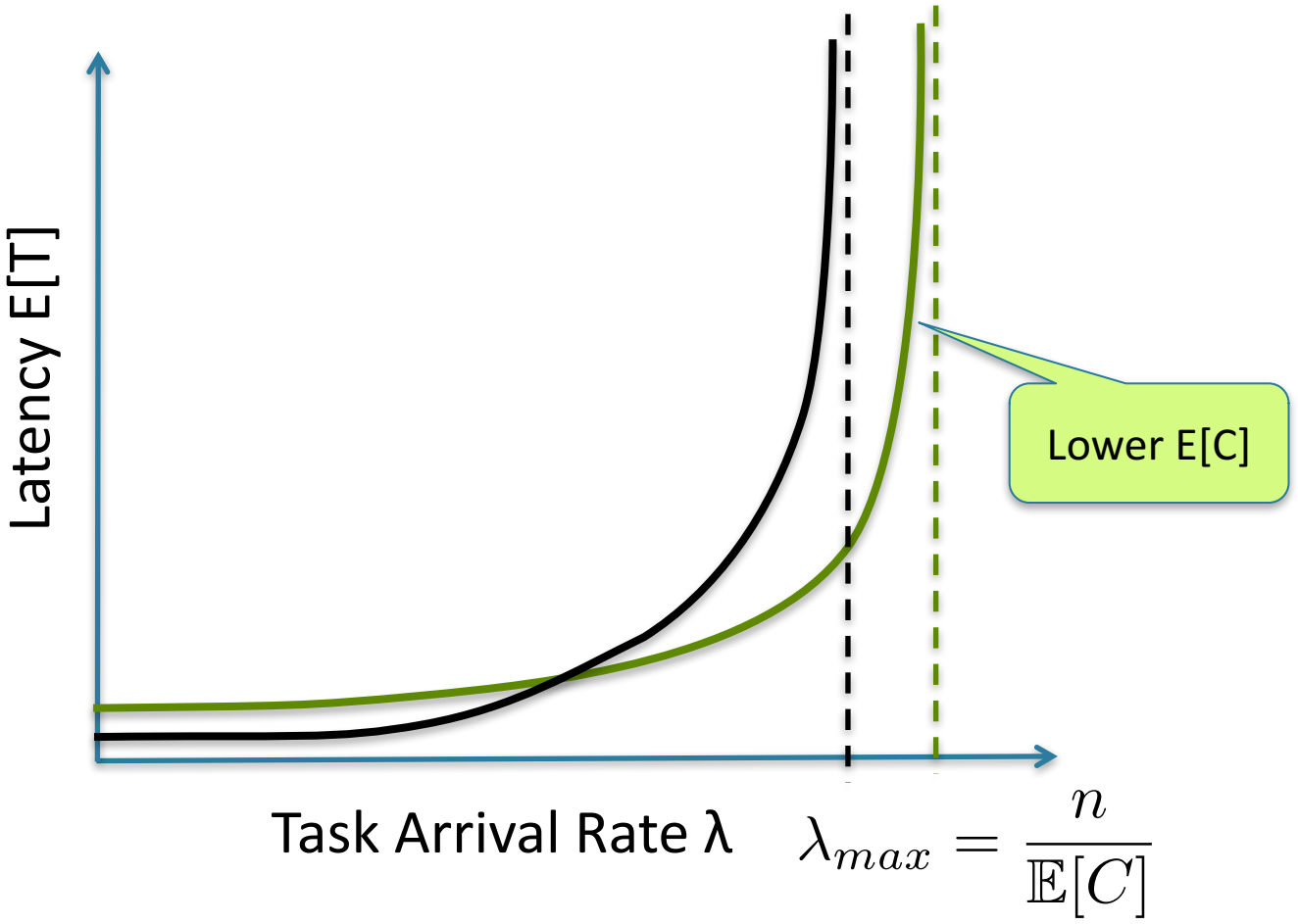
[GJ-Soljanin-Wornell MAMA 2015]



$$\lambda_{max} = \frac{n}{\mathbb{E}[C]}$$

for any symmetric policy

Lower E[C] → Higher λ_{max}
→ Lower E[T] in high traffic



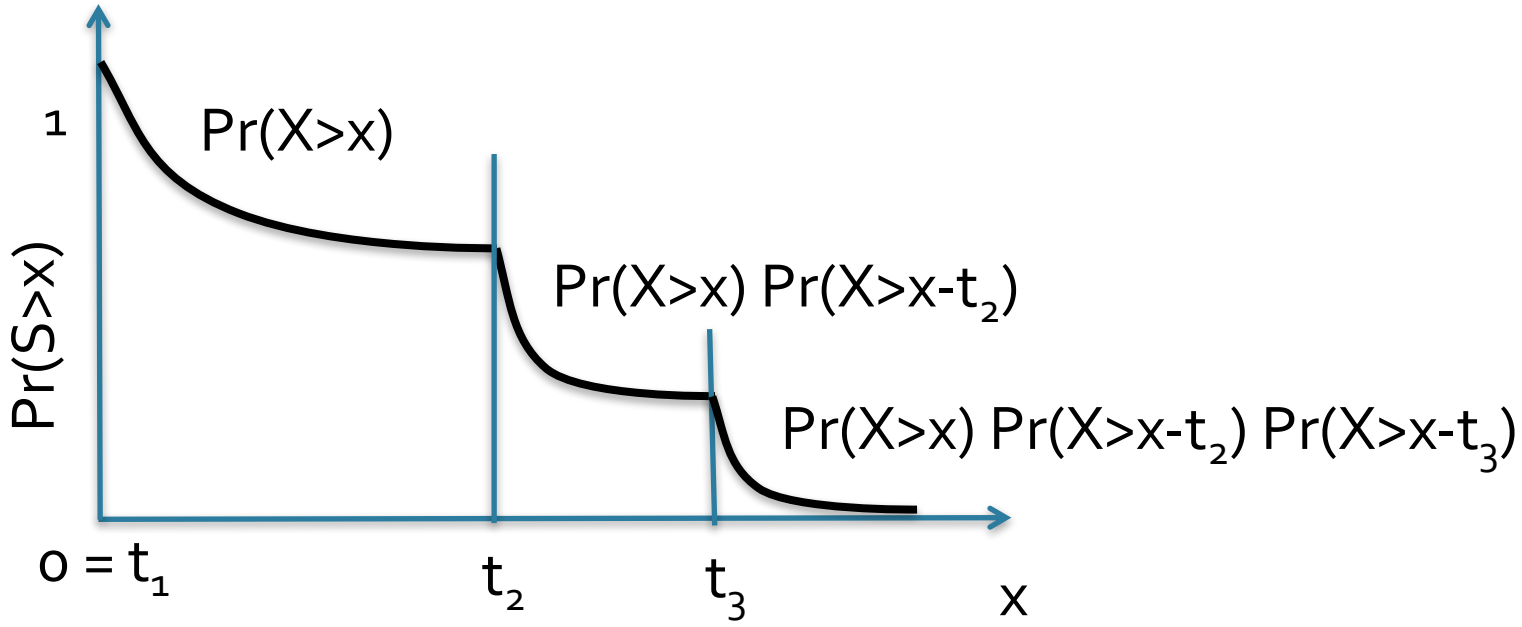
Problem reduces to finding the strategy that minimizes E[C]

E[C] in terms of relative task start times

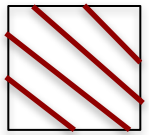
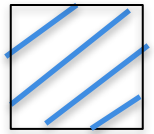
For relative task start times $0 = t_1 \le t_2 \le t_3 \ldots \le t_r$,

$$C = S + (S - t_2)^+ + (S - t_3)^+ .. + (S - t_r)^+$$

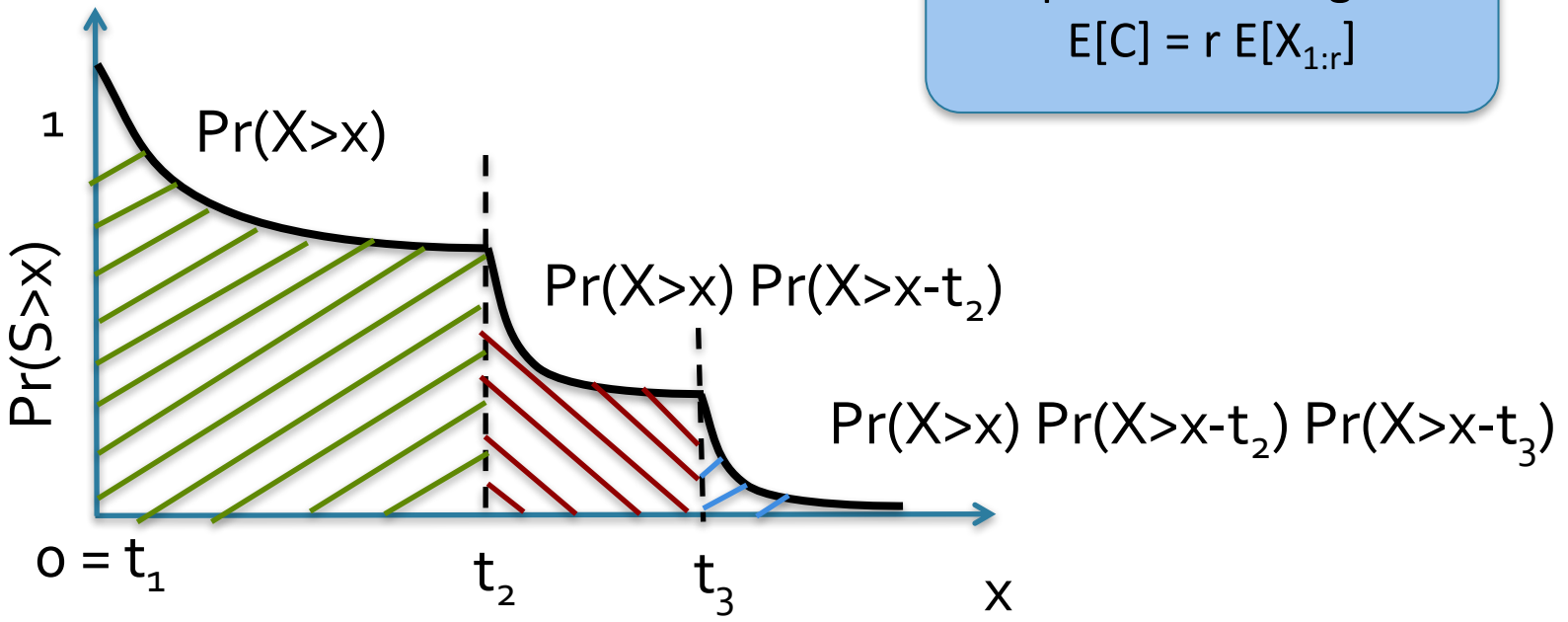
where $S = \min(X_1, X_2 + t_2, \ldots X_r + t_r)$, the time when earliest task starts, until any 1 task finishes



E[C] in terms of relative task start times

E[C] =  + 2 ×  + 3 ×  + ..

If replicas start together
$E[C] = r\, E[X_{1:r}]$



Bounds on E[C], independent of the relative task start times

For relative task start times $0 = t_1 \leq t_2 \leq t_3 \ldots \leq t_r$,

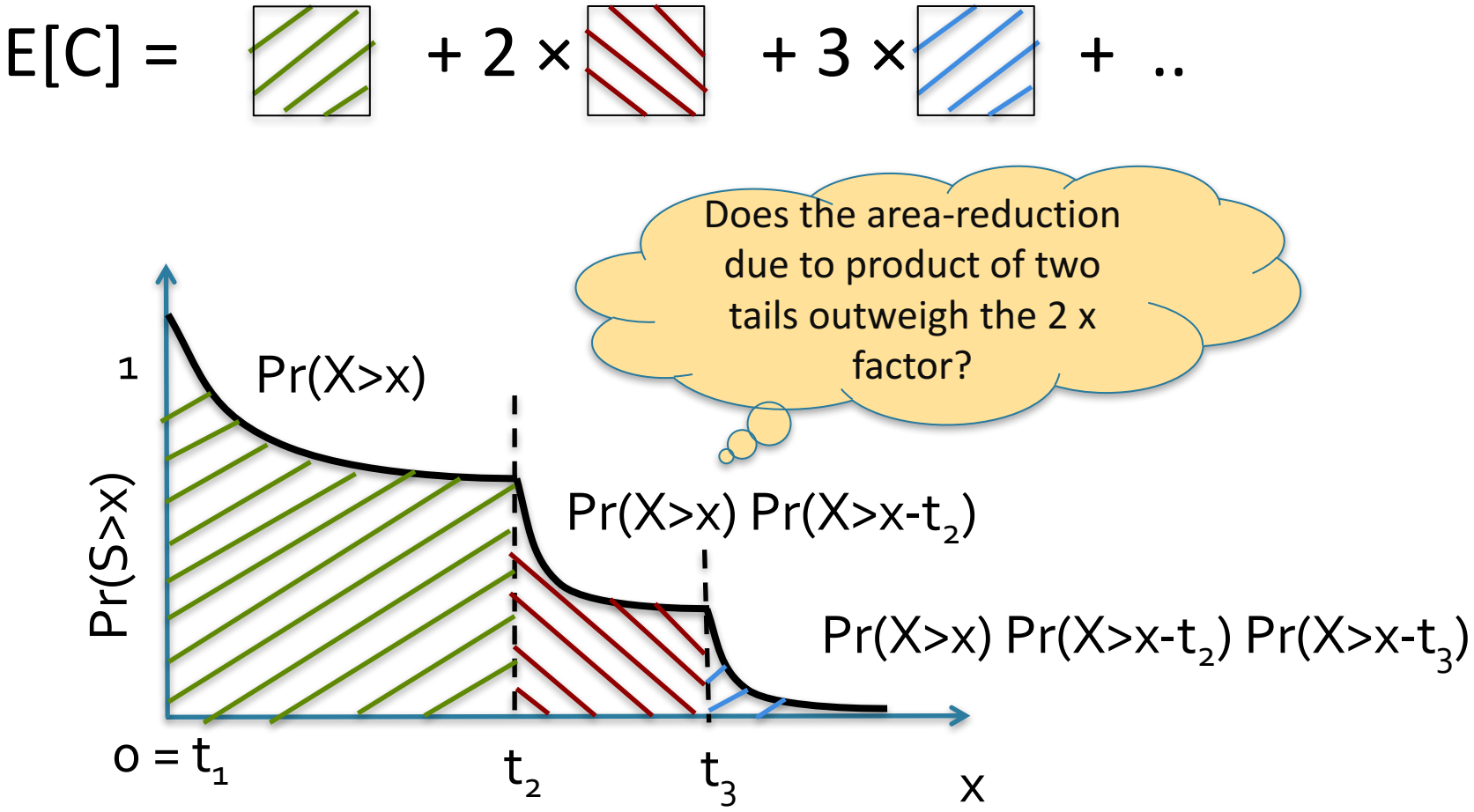
$$C = S + (S - t_2)^+ + (S - t_3)^+ .. + (S - t_r)^+$$

where $S = \min(X_1, X_2 + t_2, \ldots X_r + t_r)$, the time when earliest task starts, until any 1 task finishes

THEOREM [GJ-Soljanin-Wornell Allerton 2015]: For any relative task start times $0 = t_1 \leq t_2 \leq t_3 \ldots \leq t_r$,

$$\mathbb{E}[X] \leq \mathbb{E}[C] \leq r\mathbb{E}[X_{1:r}], \text{ if } \Pr(X > x) \text{ is log-concave}$$
$$\mathbb{E}[X] \geq \mathbb{E}[C] \geq r\mathbb{E}[X_{1:r}], \text{ if } \Pr(X > x) \text{ is log-convex}$$

Proof Idea


E[C] =
+ 2 ×
+ 3 ×
+ ..
Does the area-reduction due to product of two tails outweigh the 2 x factor?
1
Pr(X>x)
Pr(S>x)
Pr(X>x) Pr(X>x-t2)
Pr(X>x) Pr(X>x-t2) Pr(X>x-t3)
o = t1
t2
t3
x

Using E[C] bounds to determine optimal r

THEOREM [GJ-Soljanin-Wornell Allerton 2015]: For a symmetric policy resulting in any relative task start times,

$$\mathbb{E}[X] \leq \mathbb{E}[C] \leq r\mathbb{E}[X_{1:r}], \text{ if } \bar{F}_X \text{ is log-concave}$$
$$\mathbb{E}[X] \geq \mathbb{E}[C] \geq r\mathbb{E}[X_{1:r}], \text{ if } \bar{F}_X \text{ is log-convex}$$

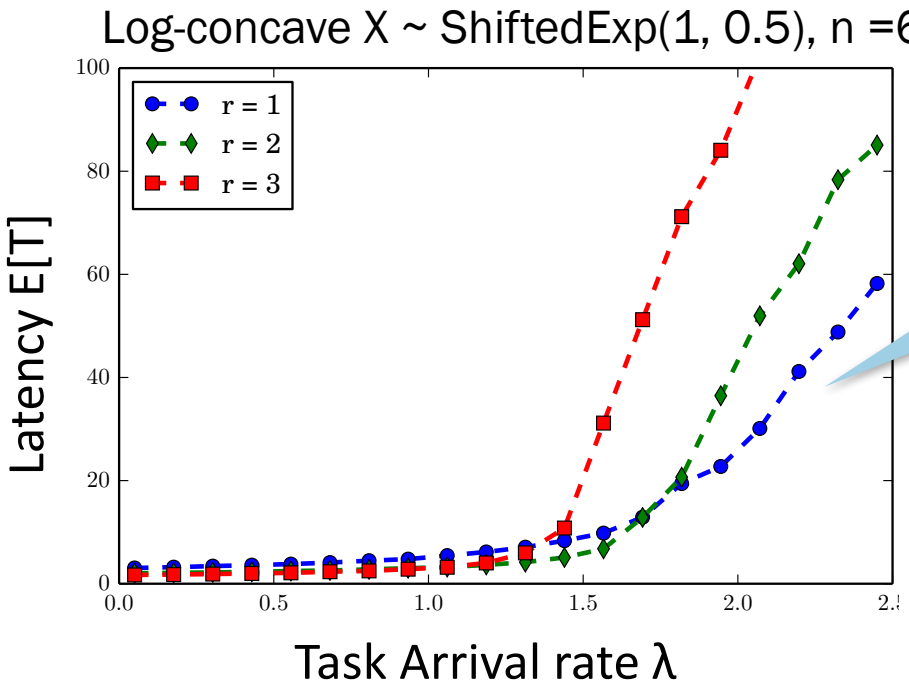
For any X, if $r = 1, \mathbb{E}[C] = \mathbb{E}[X]$ and if $r = n, \mathbb{E}[C] = n\mathbb{E}[X_{1:n}]$

COROLLARY: The value of r that minimizes E[C] (and hence minimizes E[T] in high load regime) is

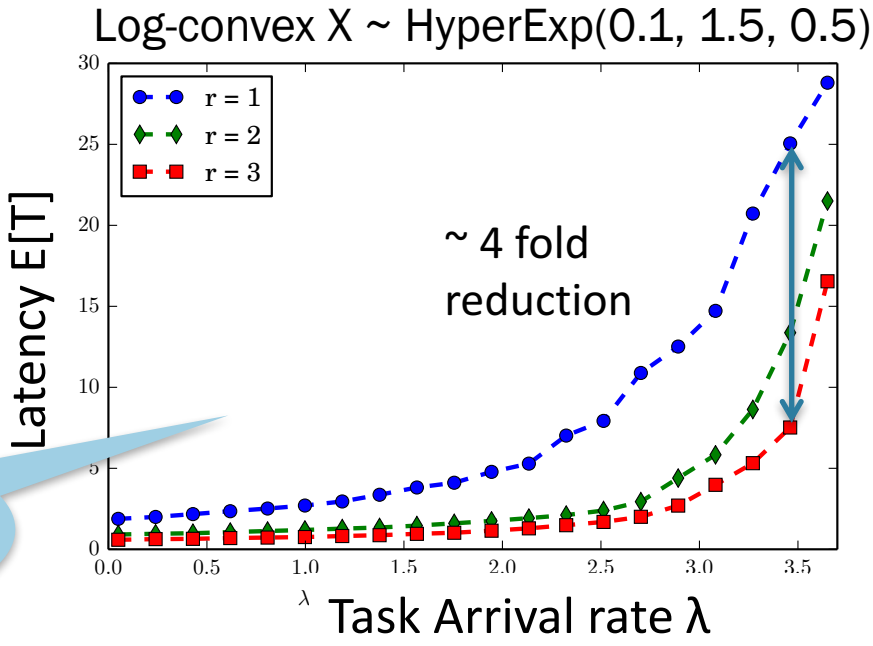
$$r = 1, \text{ if } \bar{F}_X \text{ is log-concave}$$
$$r = n, \text{ if } \bar{F}_X \text{ is log-convex}$$

Latency versus λ for different r

Uniform Random Scheduling



Low traffic: r = n
High traffic r = 1



More replicas
always better

Given r, using E[C] bounds to determine the best scheduling policy

THEOREM [GJ-Soljanin-Wornell Allerton 2015]: For a symmetric policy resulting in any relative task start times,

$$\mathbb{E}[X] \leq \mathbb{E}[C] \leq r\mathbb{E}[X_{1:r}], \text{ if } \bar{F}_X \text{ is log-concave}$$
$$\mathbb{E}[X] \geq \mathbb{E}[C] \geq r\mathbb{E}[X_{1:r}], \text{ if } \bar{F}_X \text{ is log-convex}$$

For any X, if $r = 1, \mathbb{E}[C] = \mathbb{E}[X]$ and if $r = n, \mathbb{E}[C] = n\mathbb{E}[X_{1:n}]$

Equality when $0 = t_1 = t_2 = t_3 .. = t_r$ (simultaneous task start times)

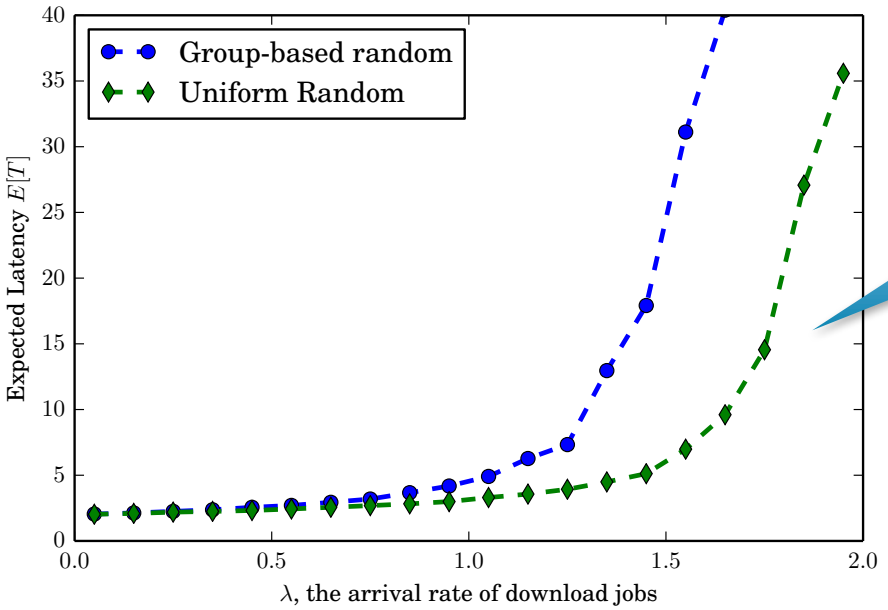
COROLLARY: In the high traffic regime,

- Log-convex: Group-based policy gives lowest E[T]
- Log-concave: Better to stagger the relative task start times

Which servers to fork to?

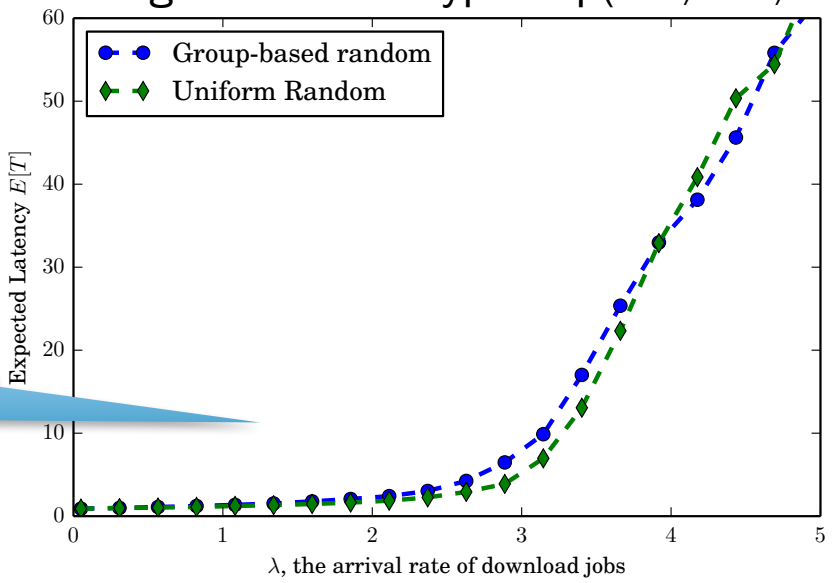
Choice of Scheduling Policy

Log-concave X ~ ShiftedExp(1, 0.5), n =6



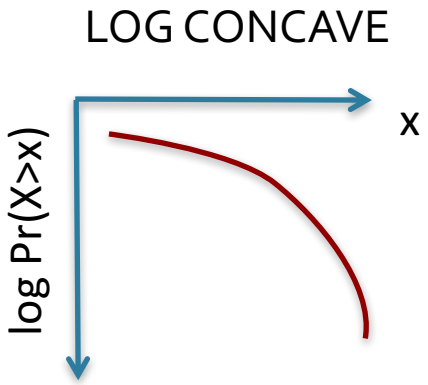
Uniform random always does better

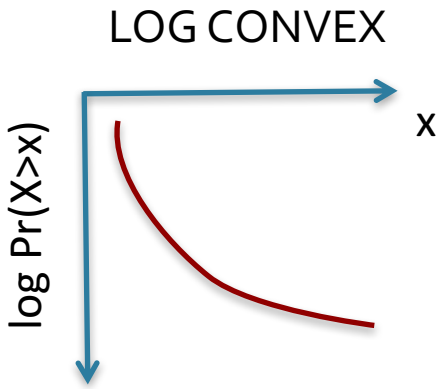
Log-convex X ~ HyperExp(0.1, 1.5, 0.5)



Group-based random better in high load

Main Takeaways





Retaining less replicas better in high traffic

More replication always better

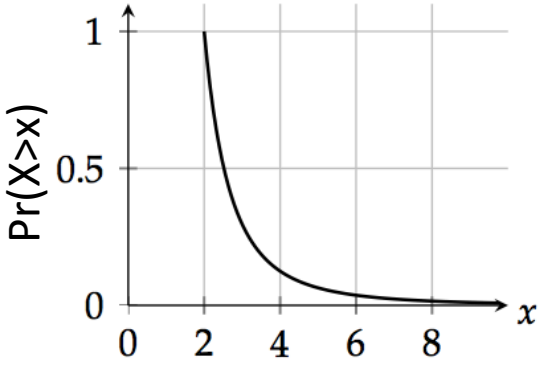
Lower E[C] → Higher service capacity $\lambda_{max} = \frac{n}{\mathbb{E}[C]}$

→ Lower E[T] in high traffic

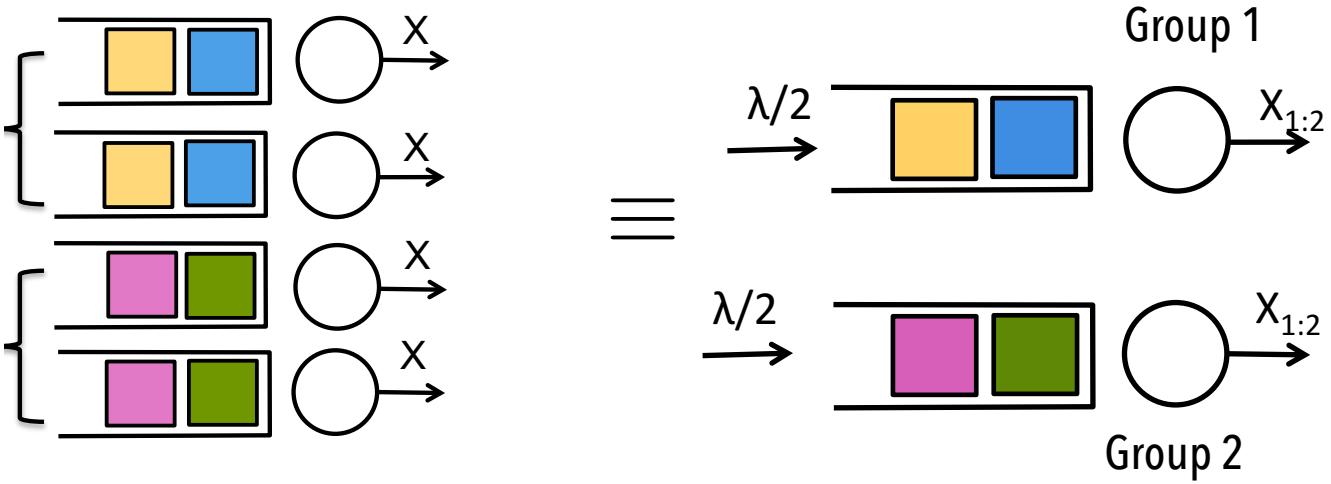
Arbitrary service dist. F_X and arrival rate λ

- Neither log-concave nor log-convex dists.,

  eg. Pareto (polynomial tail decay)

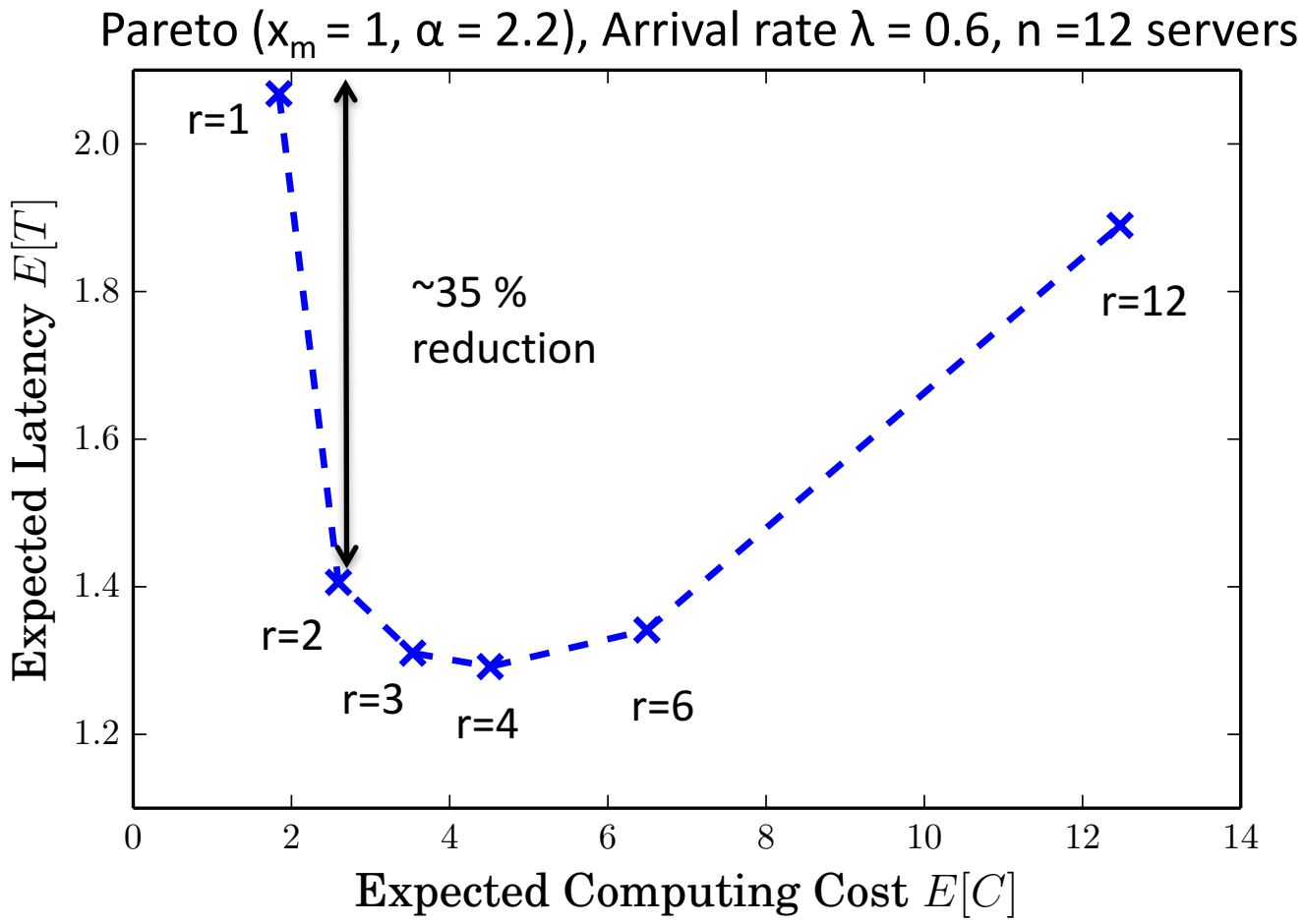


- Latency-cost analysis tractable for group-based policy



Latency versus Cost as r varies

Group-based policy



Choose suitable r based on cost sensitivity