Analytic Methods for Optimizing Realtime Crowdsourcing

Michael Bernstein, David Karger, Rob Miller, and Joel Brandt
MIT CSAIL and Adobe Systems

CSAIL Adobe

MIT HUMAN-COMPUTER INTERACTION

Use **queueing theory** to **understand** and **optimize performance** of a paid, realtime crowdsourcing platform.

- Relationship between crowd size and response time
- Algorithm for optimizing crowd size & cost vs. response time
- Improvements to the platform: 500 millisecond feedback

Realtime Crowds

Answering visual questions for blind users

[Bigham et al. 2010]

Realtime Crowds

Answering visual questions for blind users

[Bigham et al. 2010]

Crowd-assisted photography

[Bernstein et al. 2011]

Realtime Crowds

Answering visual questions for blind users

[Bigham et al. 2010]

Crowd-assisted photography

[Bernstein et al. 2011]

Paid Crowdsourcing

Pay small amounts of money for short tasks

Amazon Mechanical Turk: Roughly five million tasks completed per year at 1-5¢ each [Ipeirotis 2010]

Label an image

Requester: Matt C.
Reward: $0.01

Transcribe short audio clip

Requester: Gordon L.
Reward: $0.04

Retainer Recruitment

Workers sign up in advance

½¢ per minute to remain on call

Alert when the task is ready

Wait at most:

5 minutes

Task:

Click on the verbs in the paragraph

He leapt the fence and dashed toward the door.

[Bernstein et al. 2011]

Retainer Recruitment

Workers sign up in advance

½¢ per minute to remain on call

Alert when the task is ready

Wait at most:

5 minutes

Task:

Click on the verbs in the paragraph

He leapt the fence and dashed toward the door.

`alert()`

Start now! OK

[Bernstein et al. 2011]

Retainer Recruitment

Workers sign up in advance
½¢ per minute to remain on call
Alert when the task is ready

50% of workers return in two seconds, and
75% of workers return in three seconds.

[Bernstein et al. 2011]

State of the Literature

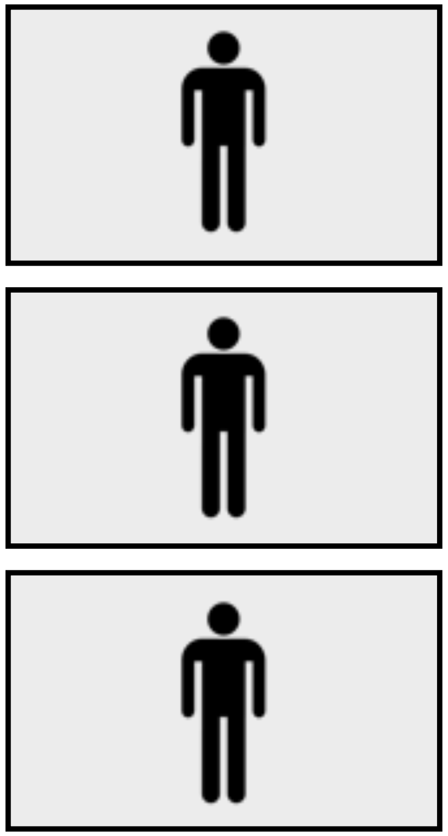
Realtime Crowds

- Recruit crowds in two seconds, execute traditional tasks (e.g., votes) in five seconds
- Maintain continuous control of remote interfaces
- Opportunities in deployable, intelligently reactive software

[Bigham et al. 2010, Bernstein et al. 2011, Lasecki et al. 2011]

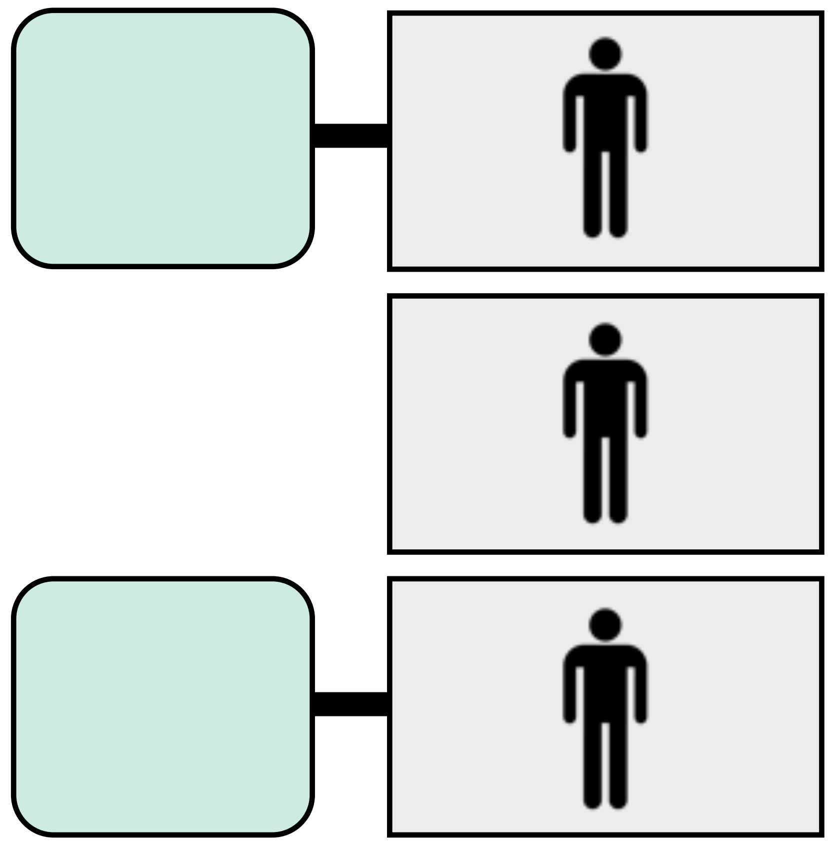
The Challenge

Running Out of Retainer Workers



The Challenge

Running Out of Retainer Workers

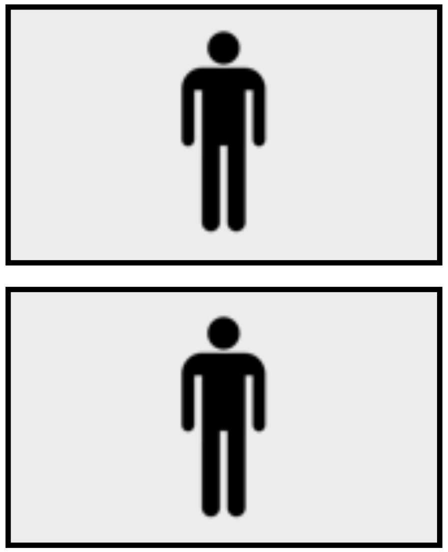


The Challenge

Running Out of Retainer Workers

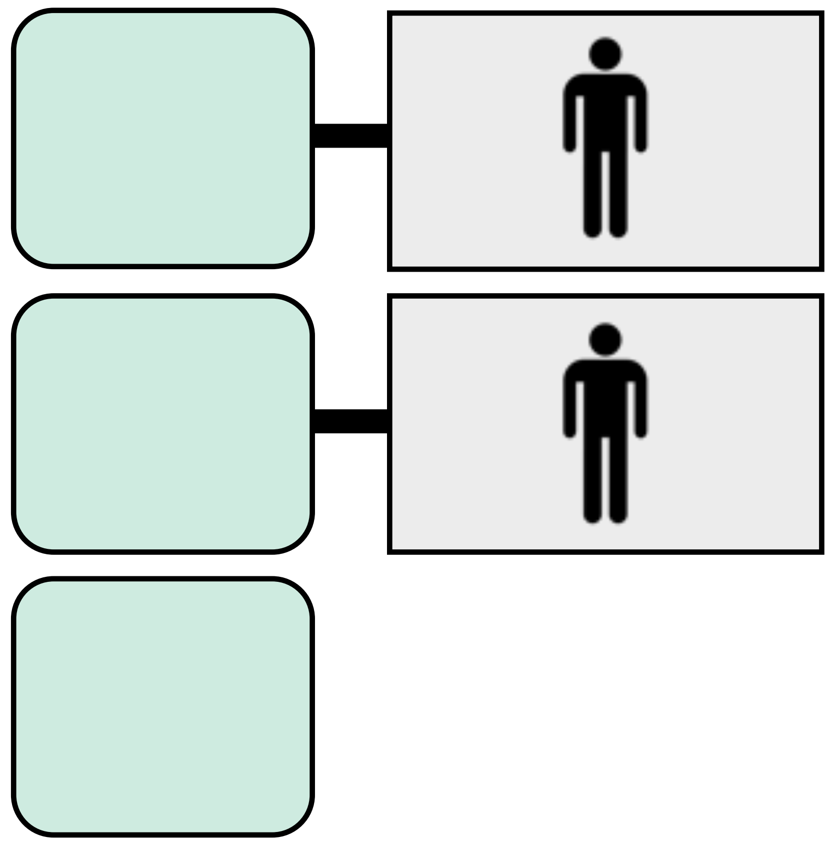
The Challenge

Running Out of Retainer Workers



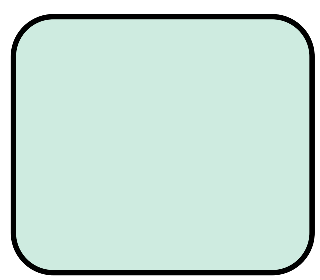
The Challenge

Running Out of Retainer Workers



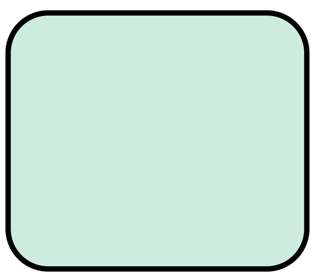
The Challenge

Running Out of Retainer Workers



The Challenge

Running Out of Retainer Workers

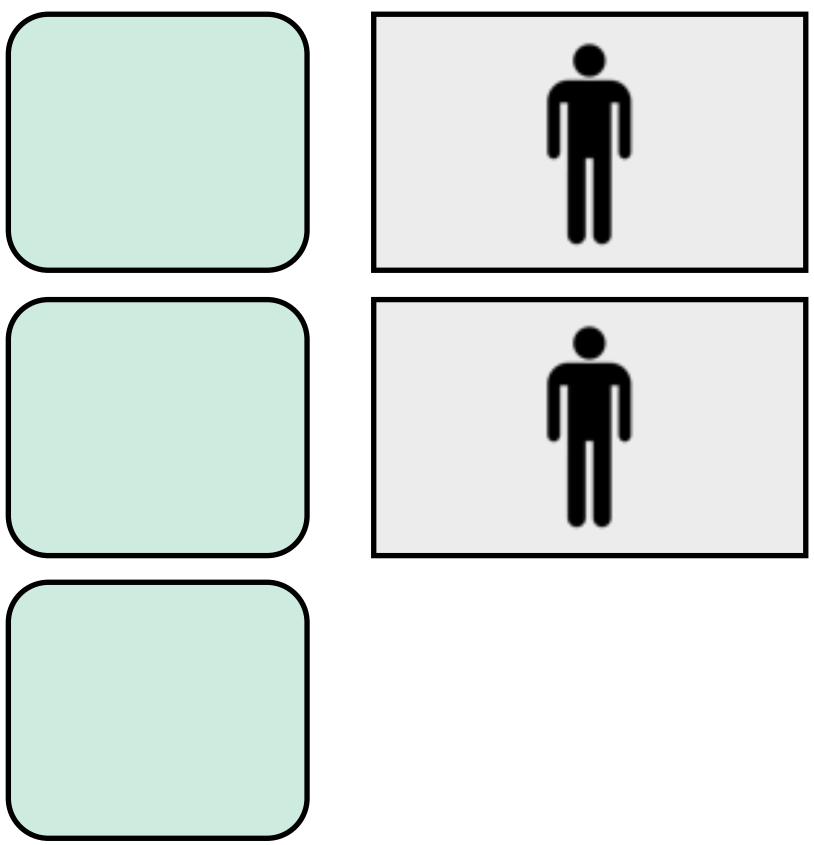


Loss

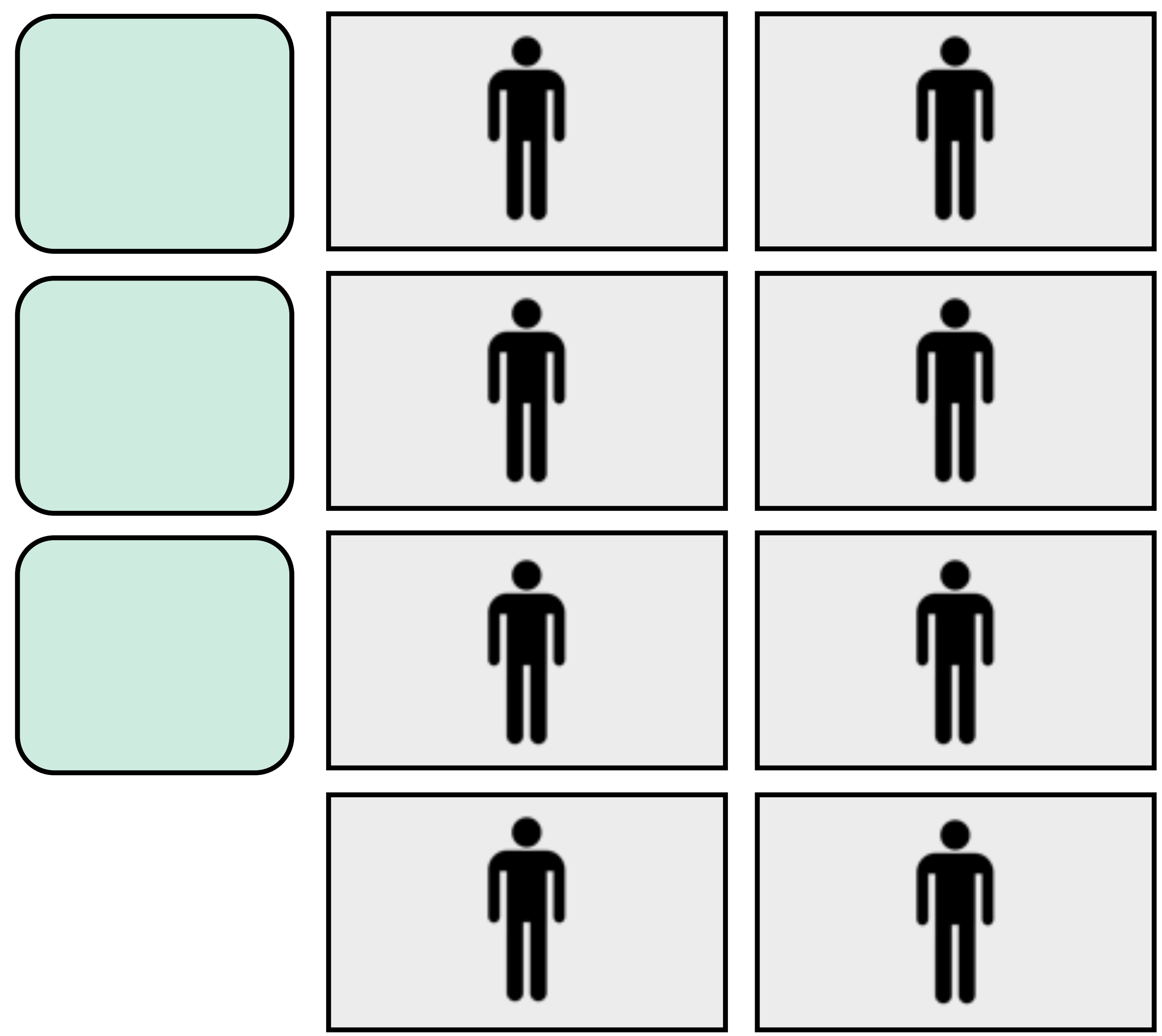
Non-realtime response

The Tradeoff

Missed tasks,
non-realtime results



Extra retainer workers,
extra cost



The Goal

Optimize the tradeoff between recruiting too many workers and dropping too many tasks.

The Goal

Optimize the tradeoff between recruiting too many workers and dropping too many tasks.

Budget-optimal crowdsourcing is possible in non-realtime scenarios

[Dai, Mausam and Weld 2010; Kamar, Hacker and Horvitz 2012; Karger, Oh, and Shah 2011]

Outline

1 Model

2 Optimization

3 Platform

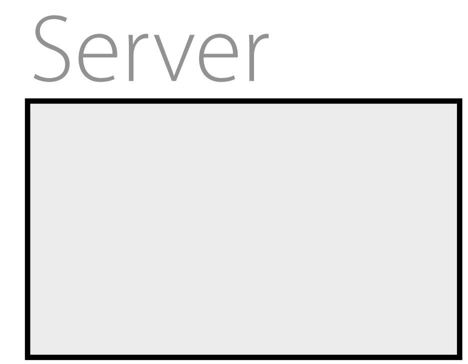
Queueing Theory

- Formal framework for stochastic arrival and service processes
- Basic idea: random task arrivals and random processing times for workers
- Quantify how long tasks will need to wait in line

Queueing theory for completion times: [Ipeirotis 2010]

Queueing Theory

M/M/1 queue



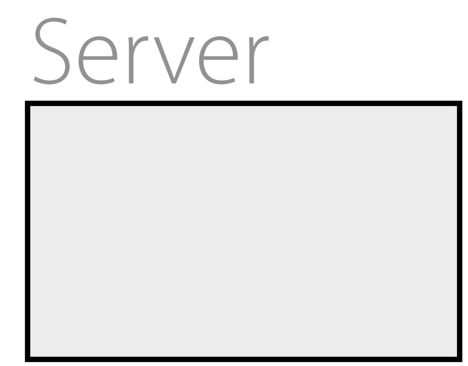
M Markovian (Poisson process) task arrivals, rate λ

M Markovian (Poisson process) server work time, rate μ

1 One server

Queueing Theory

M/M/1 queue



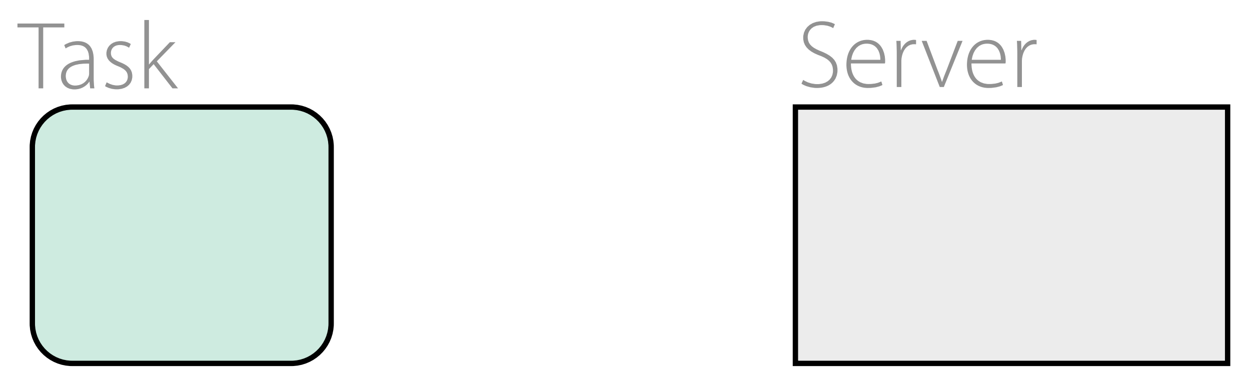
M Markovian (Poisson process) task arrivals, rate λ

M Markovian (Poisson process) server work time, rate μ

1 One server

Queueing Theory

M/M/1 queue



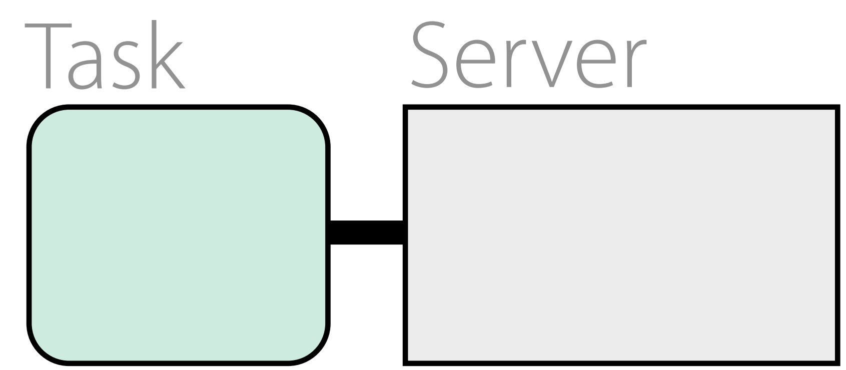
M Markovian (Poisson process) task arrivals, rate λ

M Markovian (Poisson process) server work time, rate μ

1 One server

Queueing Theory

M/M/1 queue



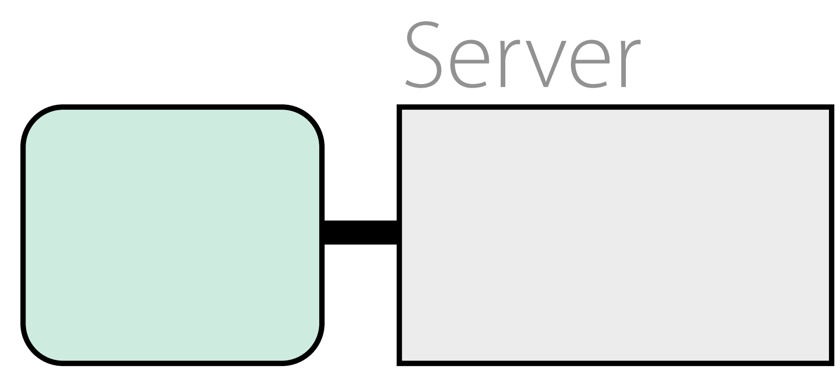
M Markovian (Poisson process) task arrivals, rate λ

M Markovian (Poisson process) server work time, rate μ

1 One server

Queueing Theory

M/M/1 queue



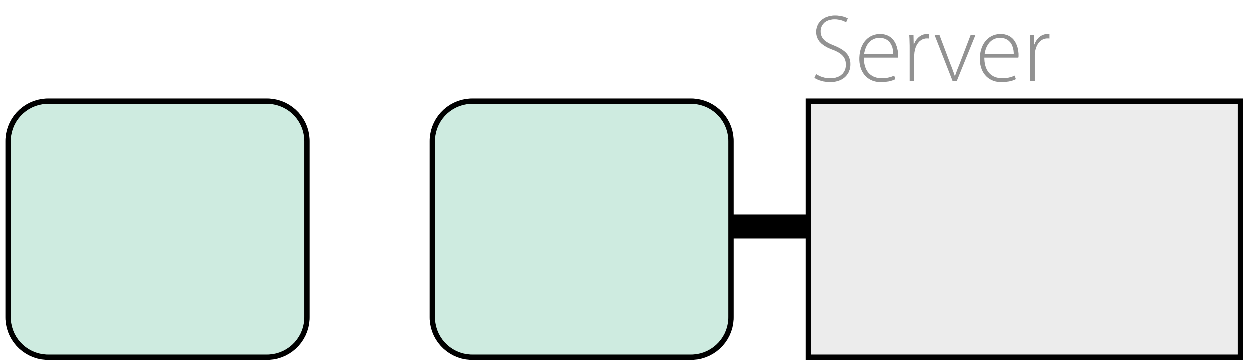
M Markovian (Poisson process) task arrivals, rate λ

M Markovian (Poisson process) server work time, rate μ

1 One server

Queueing Theory

M/M/1 queue



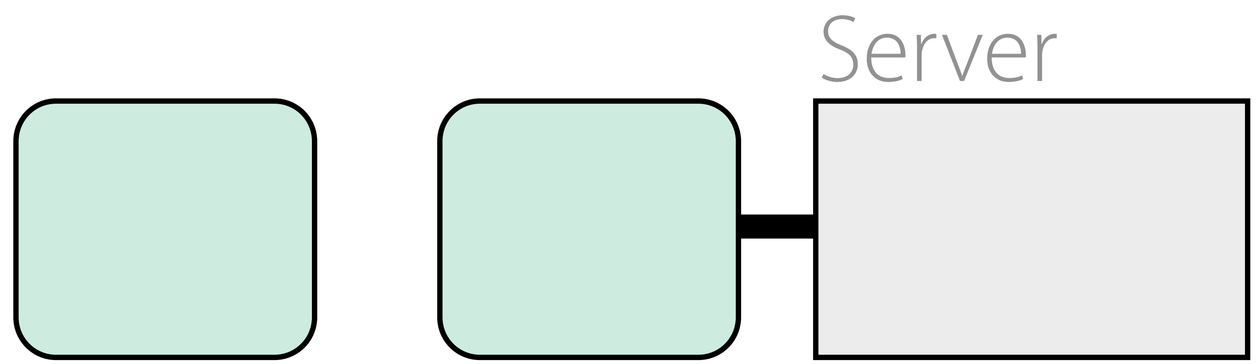
M Markovian (Poisson process) task arrivals, rate λ

M Markovian (Poisson process) server work time, rate μ

1 One server

Queueing Theory

M/M/1 queue



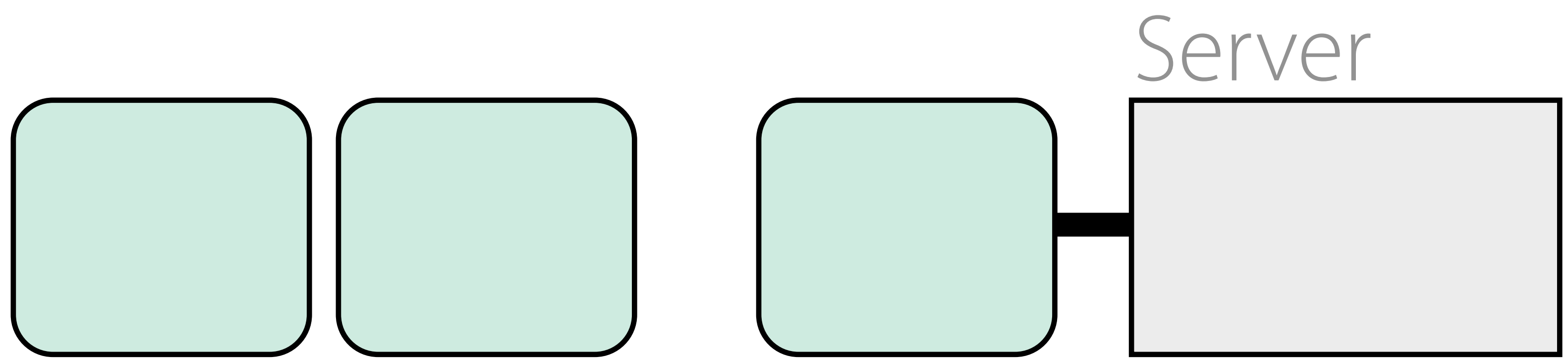
M Markovian (Poisson process) task arrivals, rate λ

M Markovian (Poisson process) server work time, rate μ

1 One server

Queueing Theory

M/M/1 queue



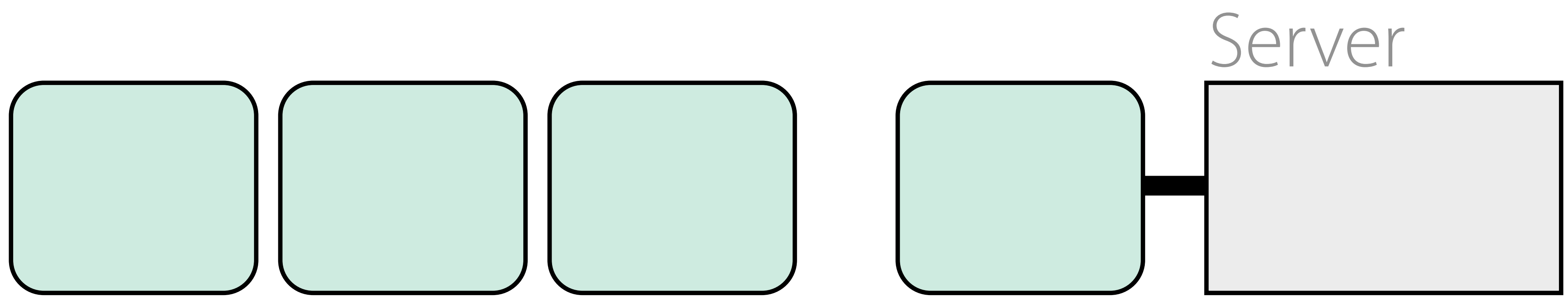
M Markovian (Poisson process) task arrivals, rate λ

M Markovian (Poisson process) server work time, rate μ

1 One server

Queueing Theory

M/M/1 queue



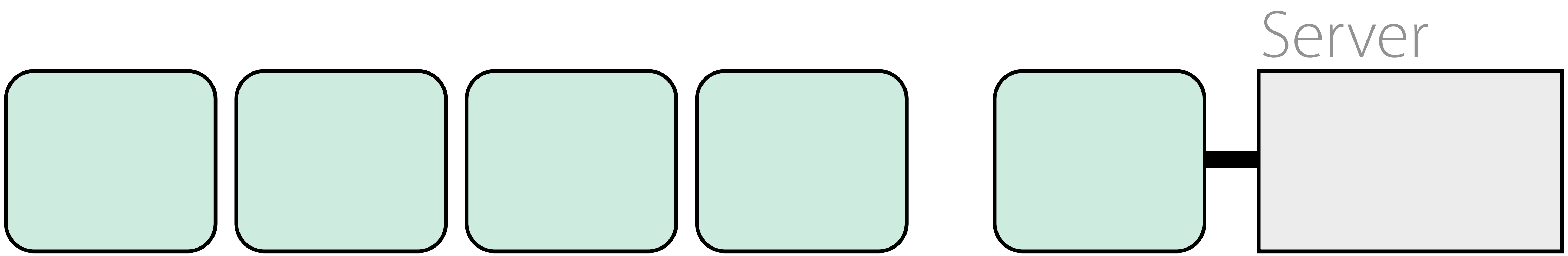
M Markovian (Poisson process) task arrivals, rate λ

M Markovian (Poisson process) server work time, rate μ

1 One server

Queueing Theory

M/M/1 queue



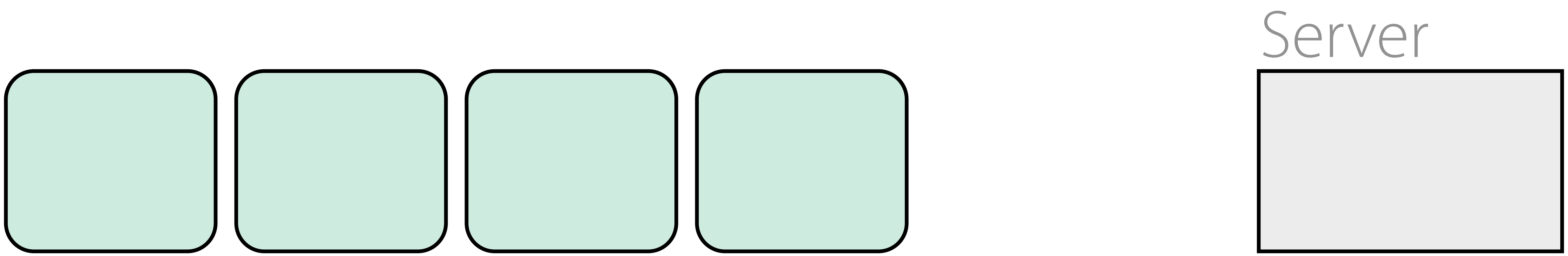
M Markovian (Poisson process) task arrivals, rate λ

M Markovian (Poisson process) server work time, rate μ

1 One server

Queueing Theory

M/M/1 queue



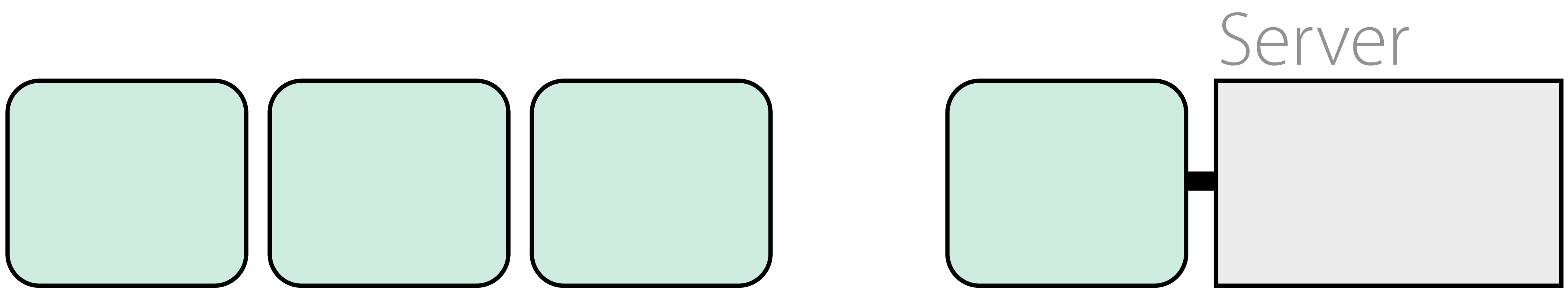
M Markovian (Poisson process) task arrivals, rate λ

M Markovian (Poisson process) server work time, rate μ

1 One server

Queueing Theory

M/M/1 queue



M Markovian (Poisson process) task arrivals, rate λ

M Markovian (Poisson process) server work time, rate μ

1 One server

Queueing Theory

M/M/c/c queue

M Markovian (Poisson process) task arrivals, rate λ

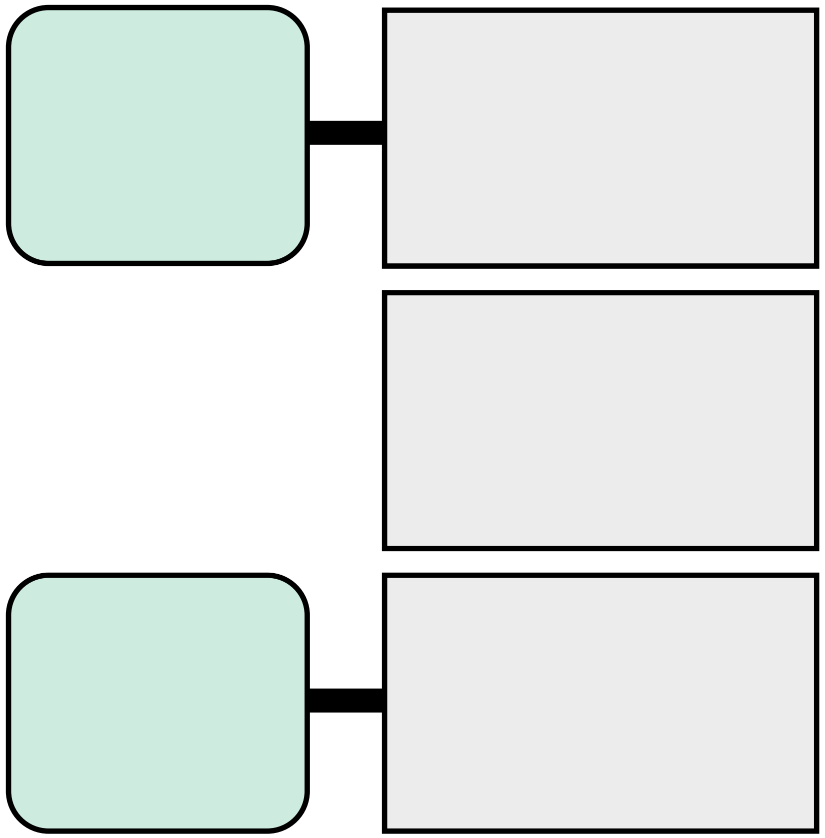
M Markovian (Poisson process) server work time, rate μ

c c servers

c c max tasks in servers and queue

Queueing Theory

M/M/c/c queue



M Markovian (Poisson process) task arrivals, rate λ

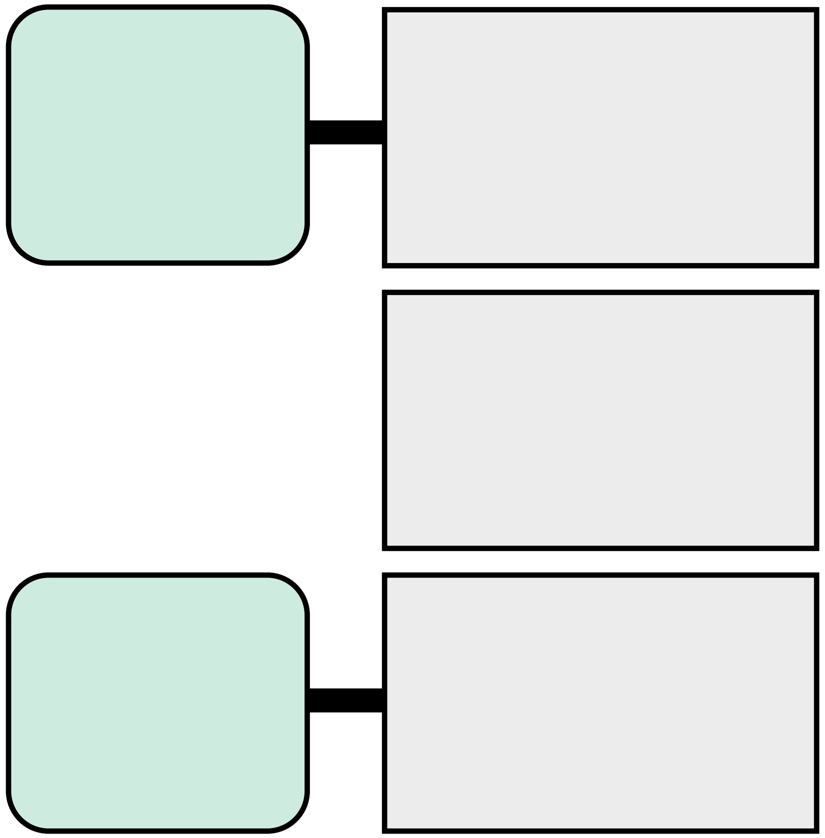
M Markovian (Poisson process) server work time, rate μ

c c servers

c c max tasks in servers and queue

Queueing Theory

M/M/c/c queue



M Markovian (Poisson process) task arrivals, rate λ

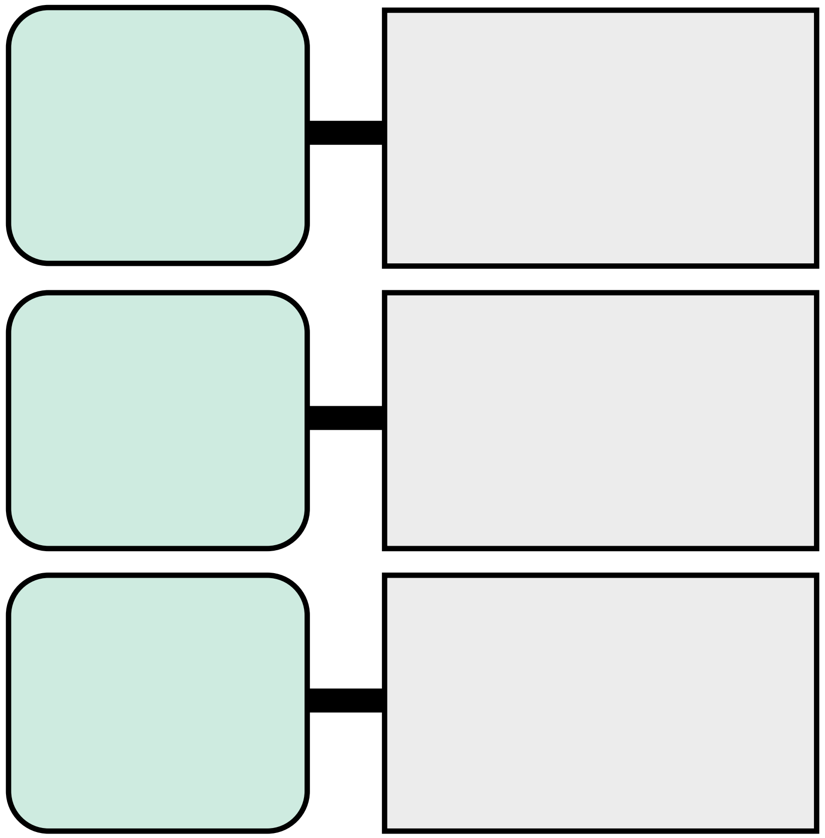
M Markovian (Poisson process) server work time, rate μ

c c servers

c c max tasks in servers and queue

Queueing Theory

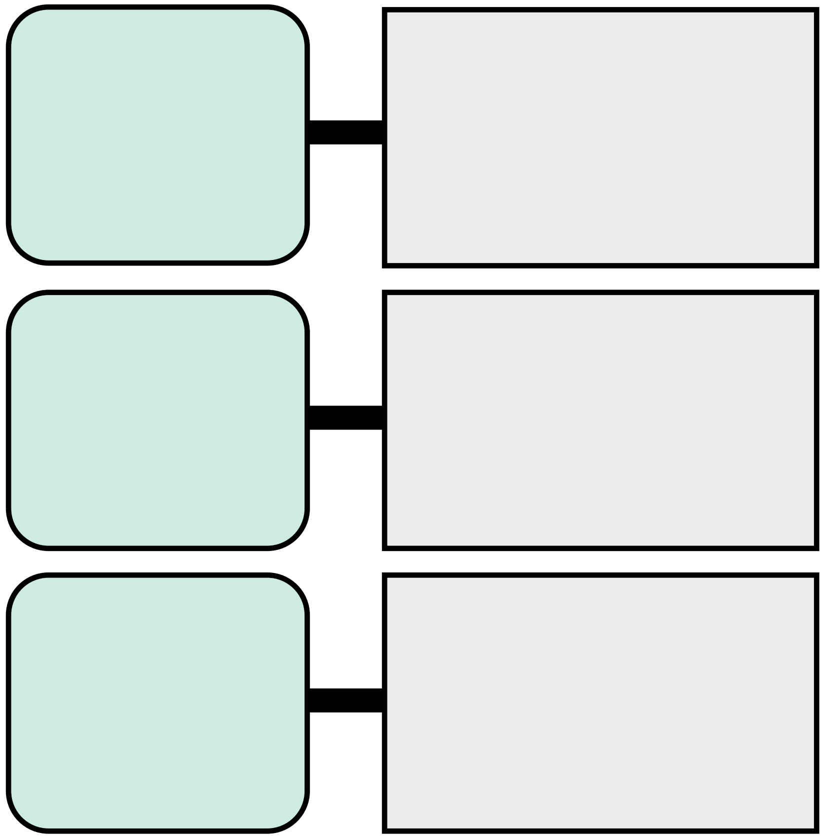
M/M/c/c queue



- **M** Markovian (Poisson process) task arrivals, rate λ
- **M** Markovian (Poisson process) server work time, rate μ
- **c** c servers
- **c** c max tasks in servers and queue

Queueing Theory

M/M/c/c queue



M Markovian (Poisson process) task arrivals, rate λ

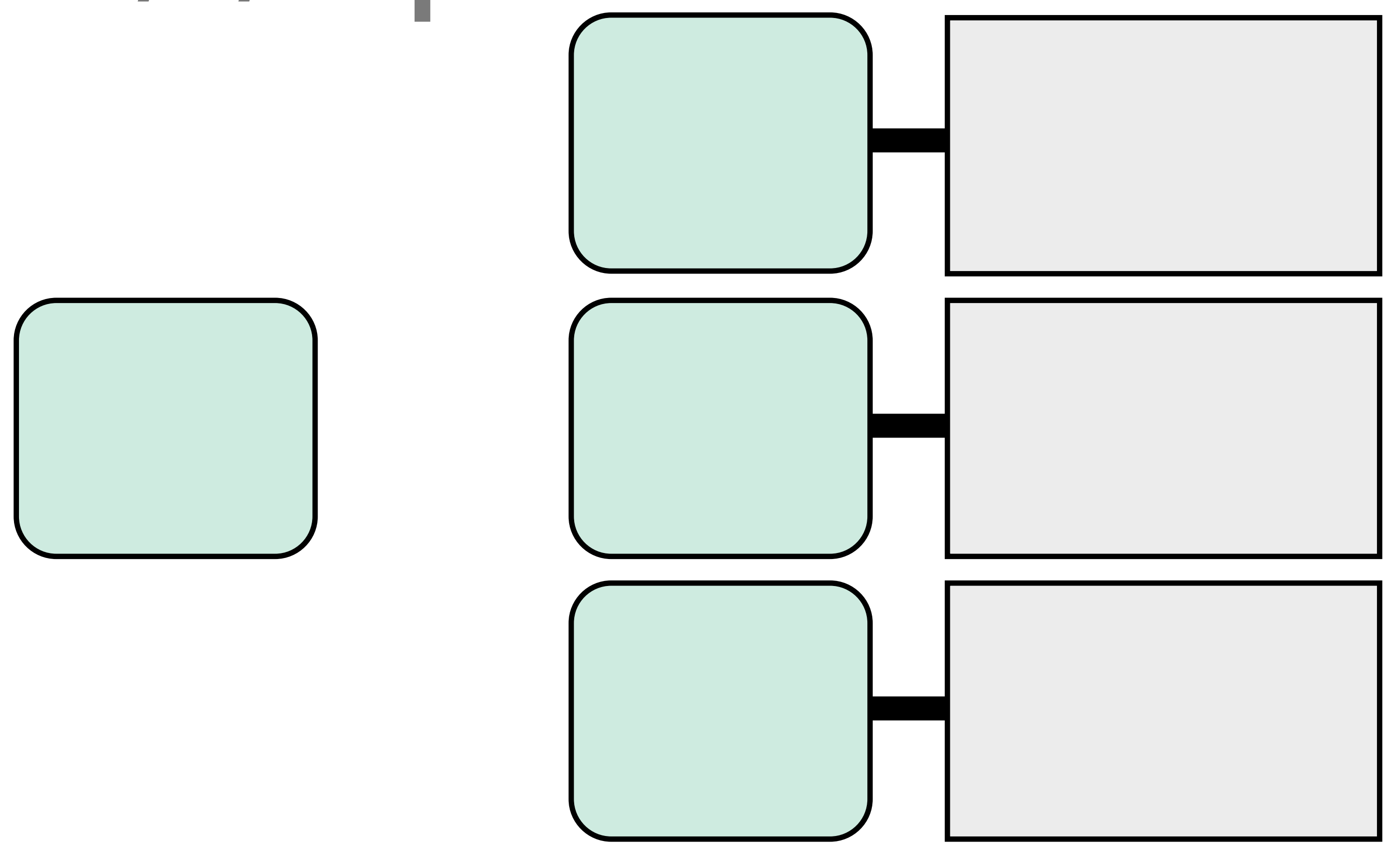
M Markovian (Poisson process) server work time, rate μ

c c servers

c c max tasks in servers and queue

Queueing Theory

M/M/c/c queue



M Markovian (Poisson process) task arrivals, rate λ

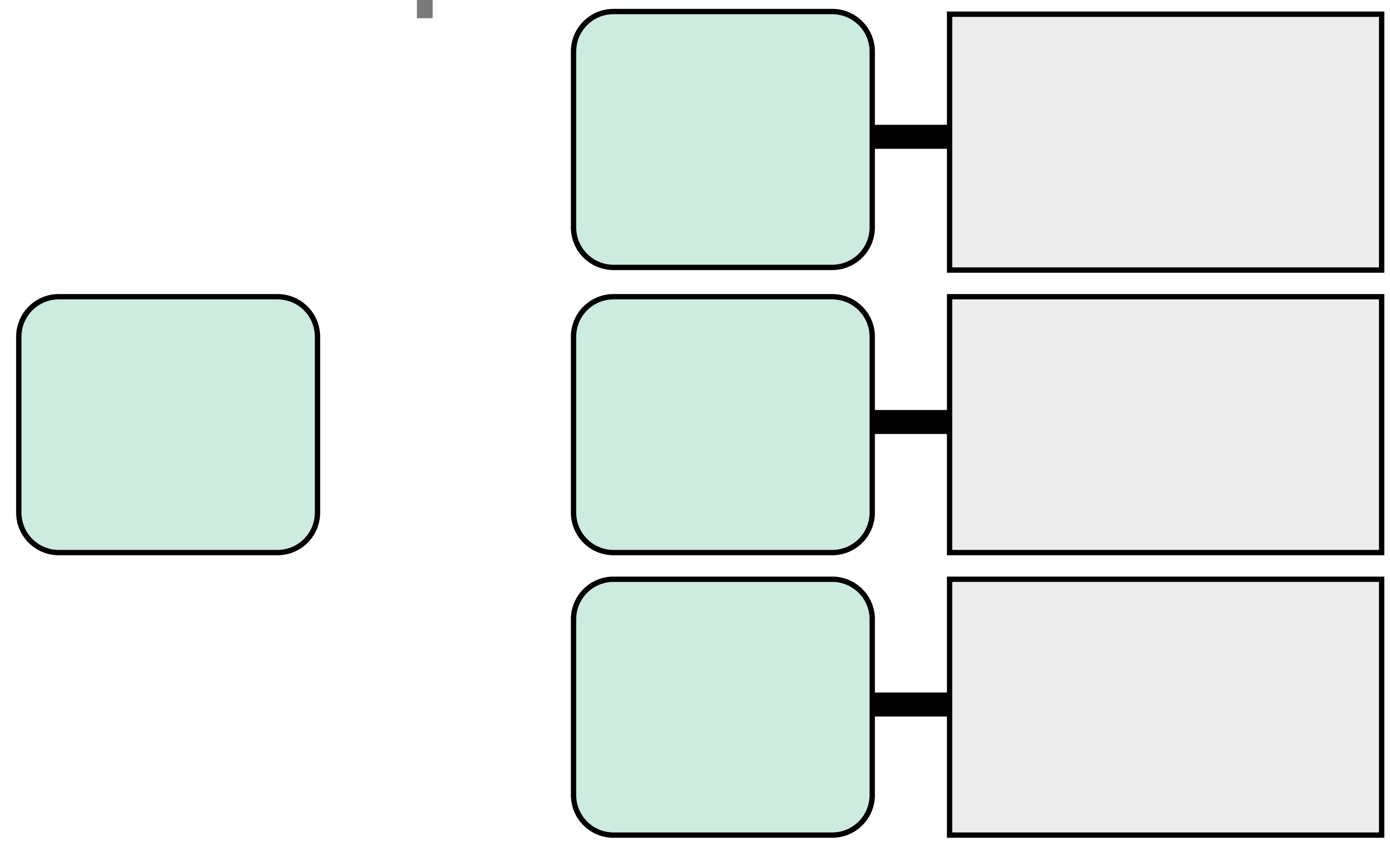
M Markovian (Poisson process) server work time, rate μ

c c servers

c c max tasks in servers and queue

Queueing Theory

M/M/c/c queue



M Markovian (Poisson process) task arrivals, rate λ

M Markovian (Poisson process) server work time, rate μ

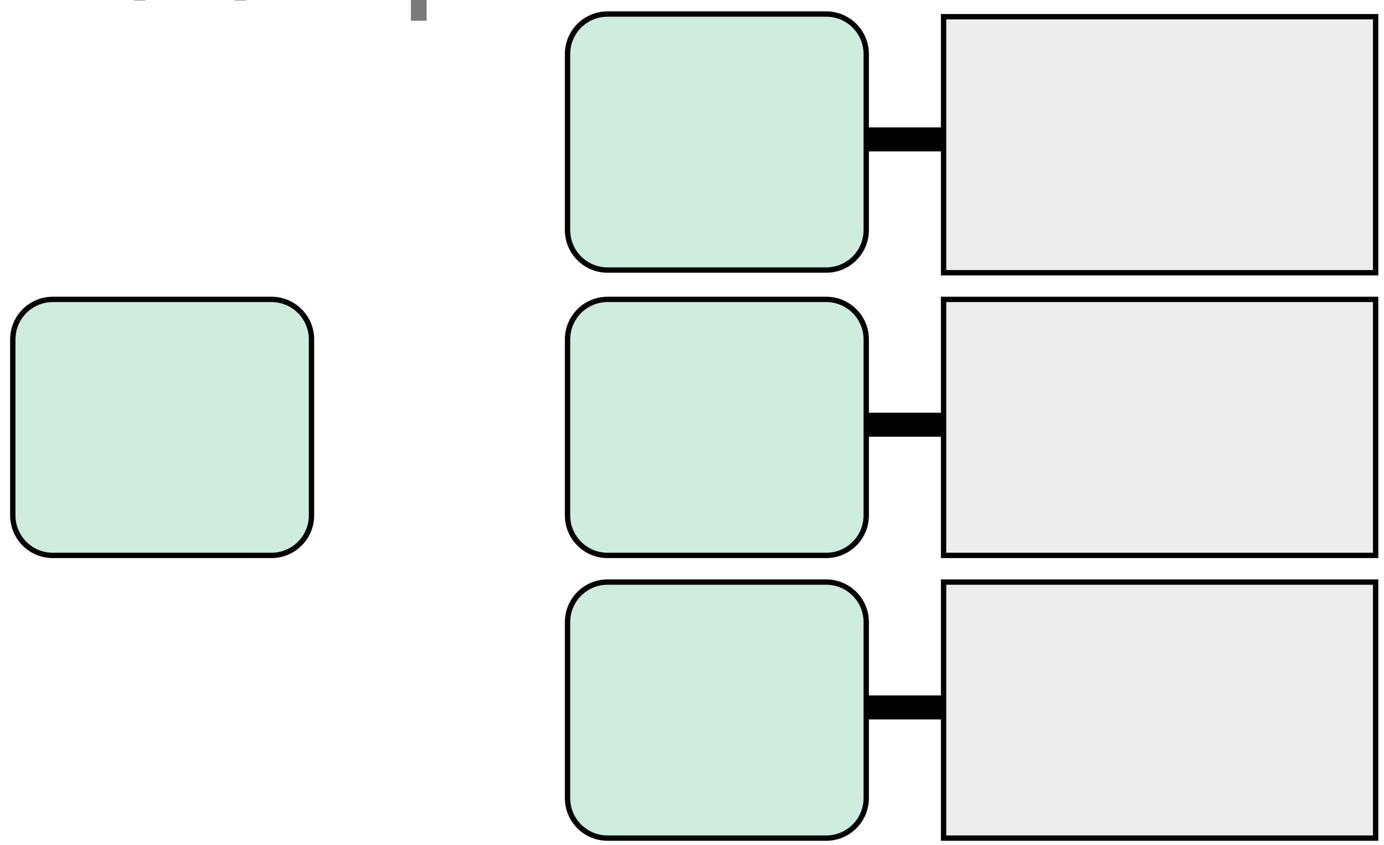
c c servers

c c max tasks in servers and queue

Queueing Theory

M/M/c/c queue

All servers busy



M Markovian (Poisson process) task arrivals, rate λ

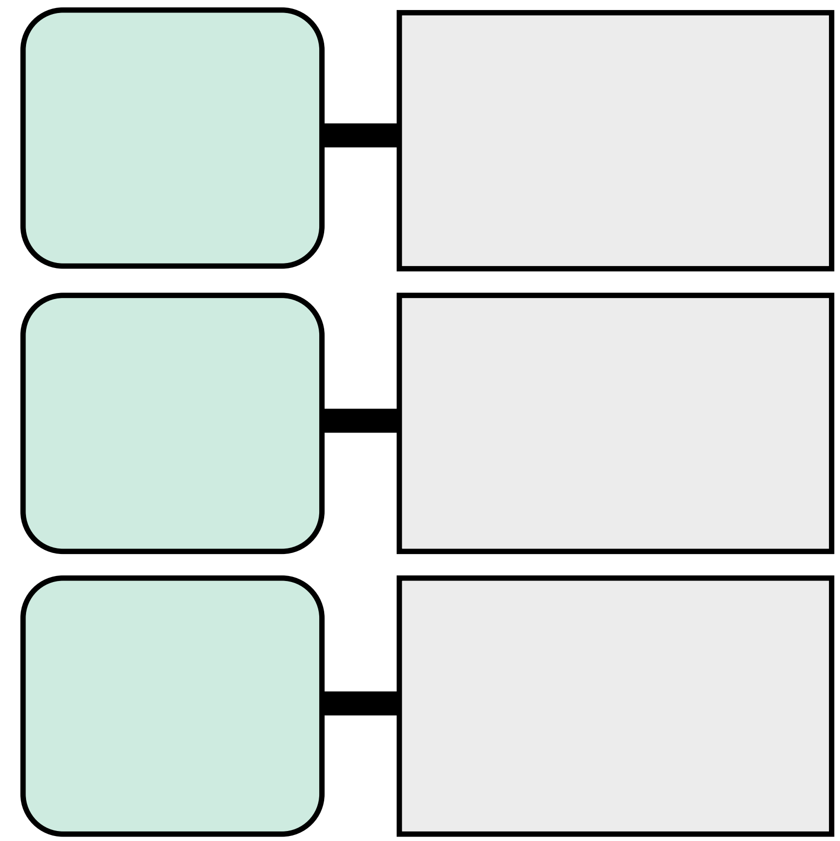
M Markovian (Poisson process) server work time, rate μ

c c servers

c c max tasks in servers and queue

Queueing Theory

M/M/c/c queue

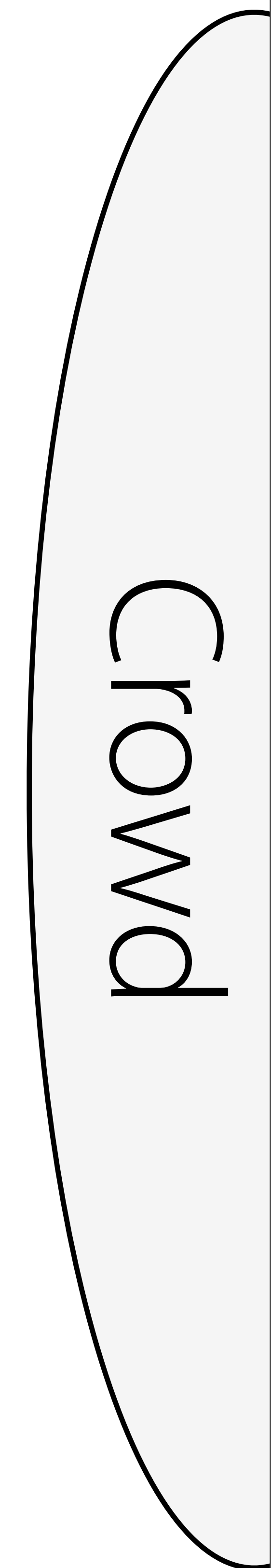


M Markovian (Poisson process) task arrivals, rate λ
M Markovian (Poisson process) server work time, rate μ
c c servers
c c max tasks in servers and queue

Modeling Retainer Recruitment

Retainer Queue

M/M/c/c queue



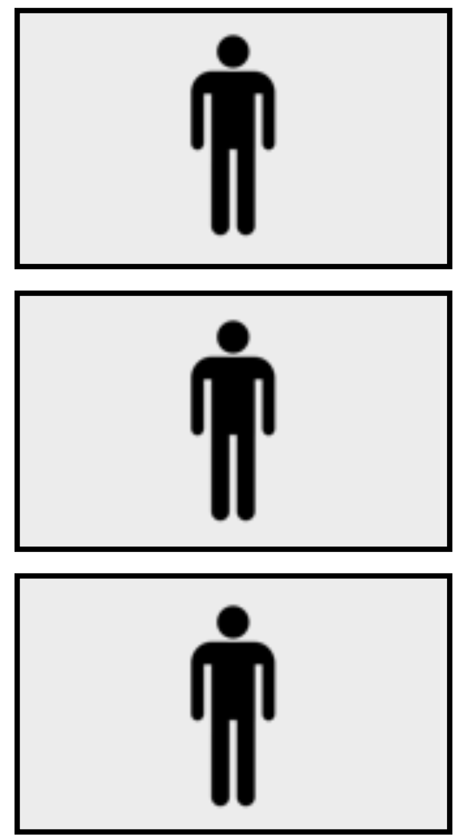
c workers, no waiting queue

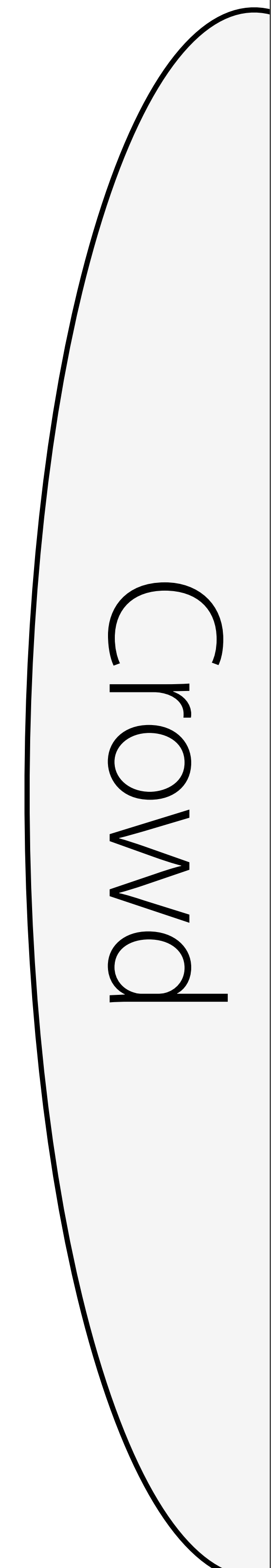
Task arrivals: Poisson process, rate λ

Worker recruitment time: Poisson process, rate μ

Retainer Queue

M/M/c/c queue





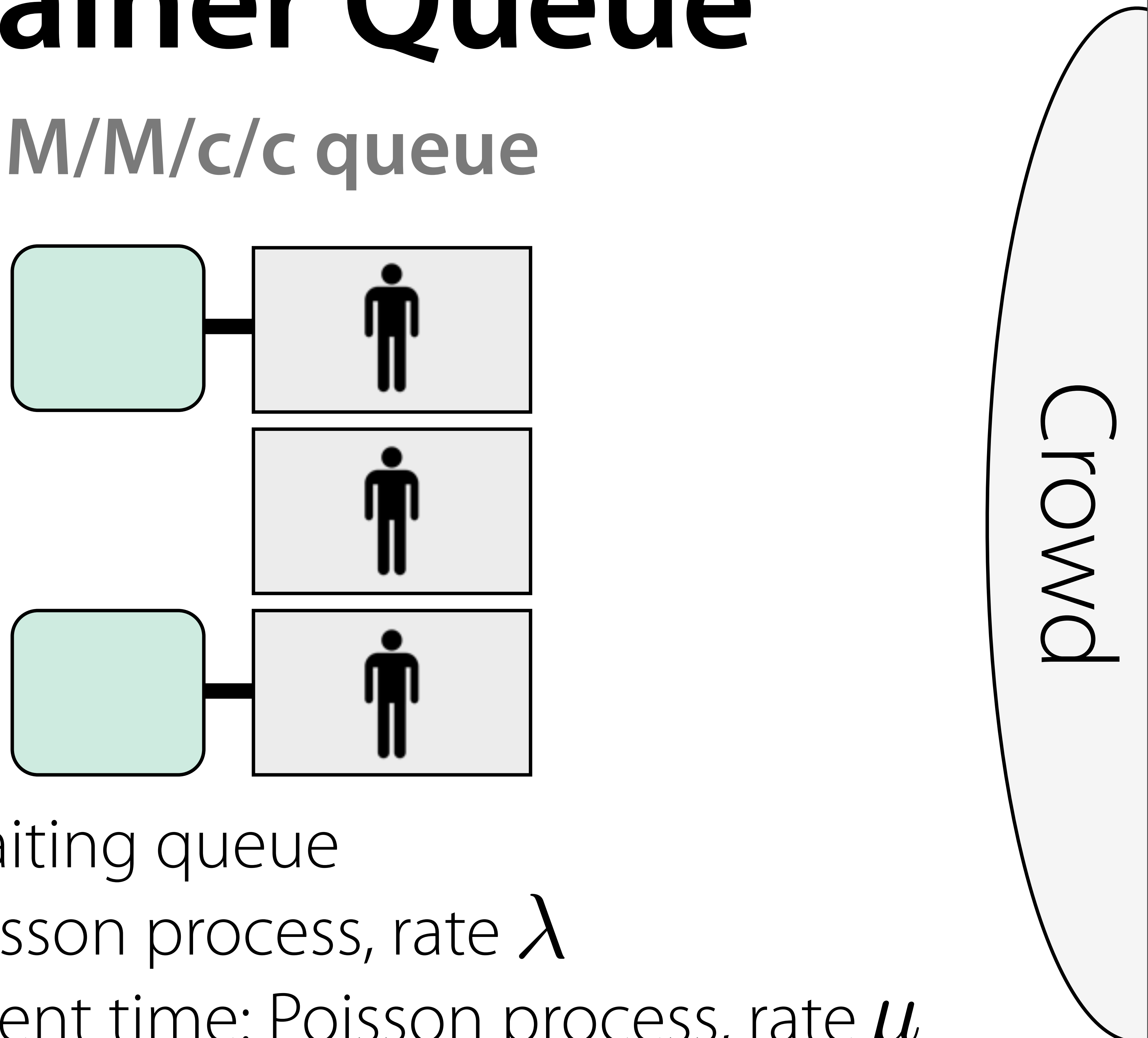
c workers, no waiting queue

Task arrivals: Poisson process, rate λ

Worker recruitment time: Poisson process, rate μ

Retainer Queue

M/M/c/c queue



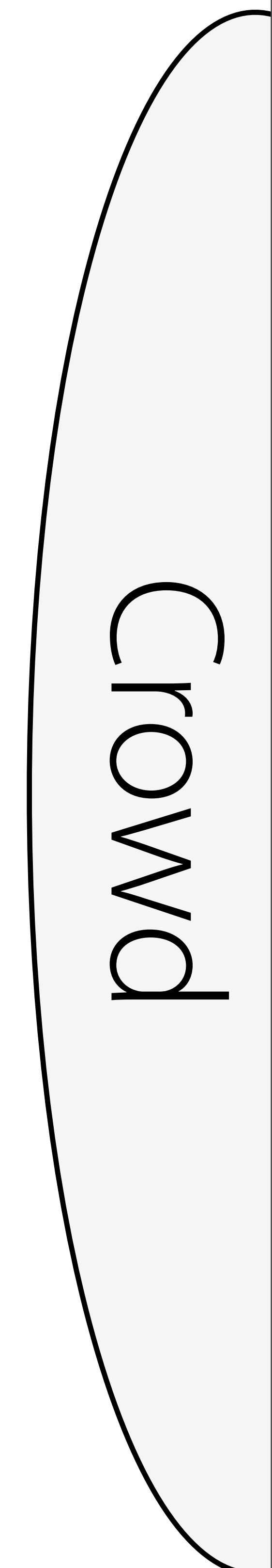
c workers, no waiting queue

Task arrivals: Poisson process, rate λ

Worker recruitment time: Poisson process, rate μ

Retainer Queue

M/M/c/c queue



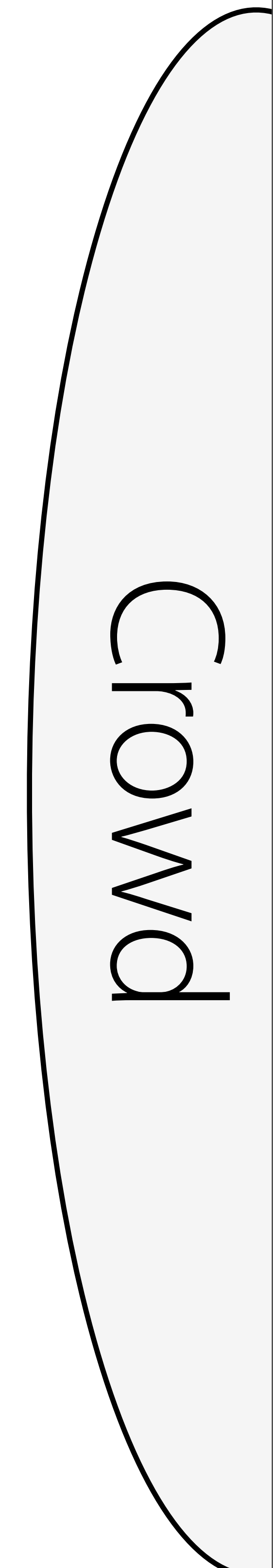
c workers, no waiting queue

Task arrivals: Poisson process, rate λ

Worker recruitment time: Poisson process, rate μ

Retainer Queue

M/M/c/c queue



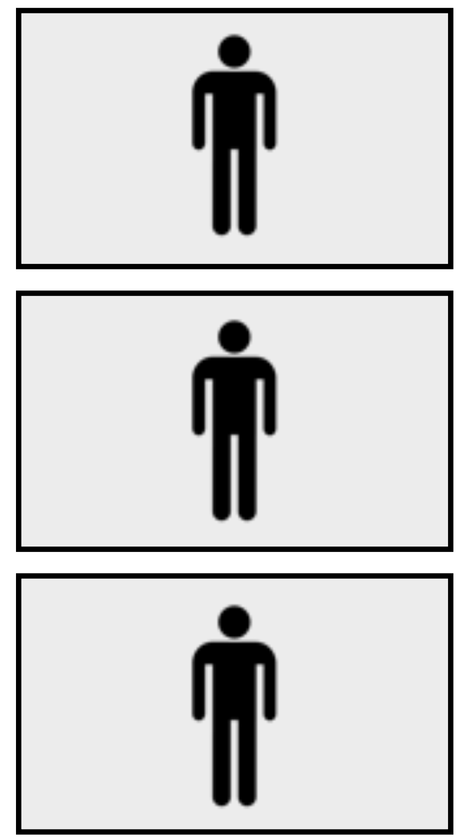
c workers, no waiting queue

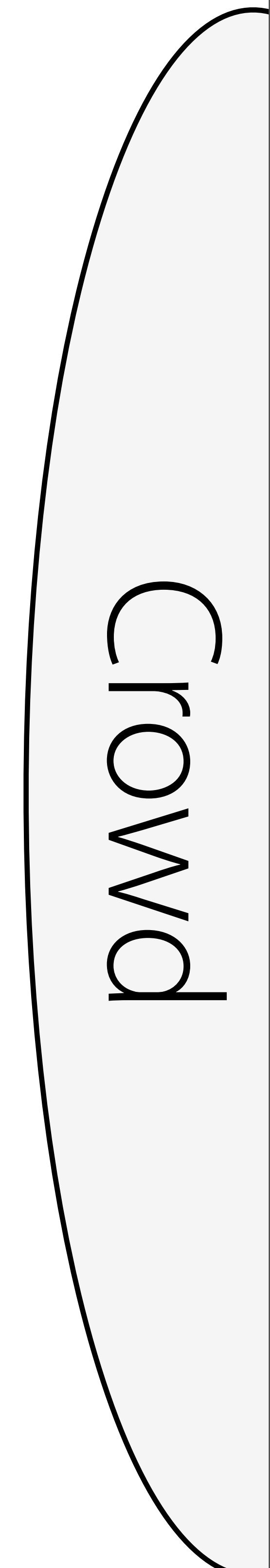
Task arrivals: Poisson process, rate λ

Worker recruitment time: Poisson process, rate μ

Retainer Queue

M/M/c/c queue





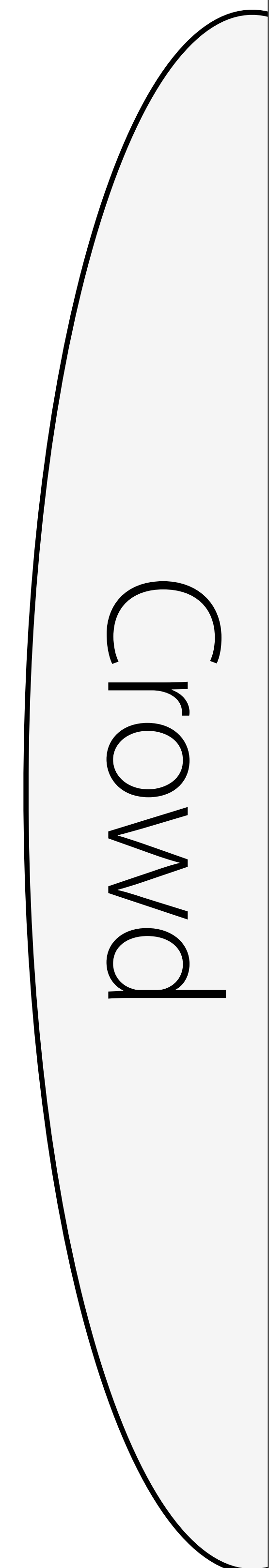
c workers, no waiting queue

Task arrivals: Poisson process, rate λ

Worker recruitment time: Poisson process, rate μ

Retainer Queue

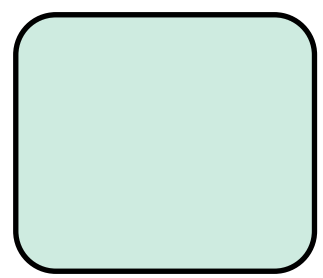
Loss

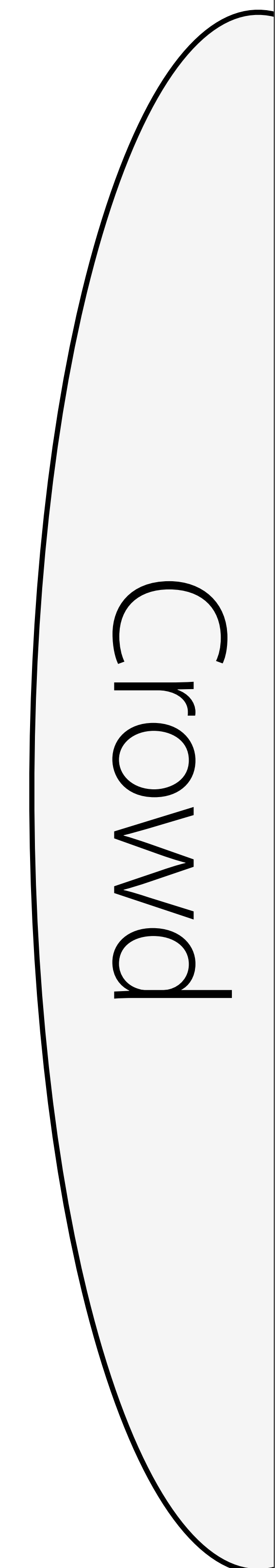


c workers, no waiting queue
Task arrivals: Poisson process, rate λ
Worker recruitment time: Poisson process, rate μ

Retainer Queue

Loss





c workers, no waiting queue

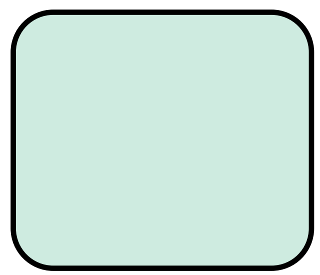
Task arrivals: Poisson process, rate λ

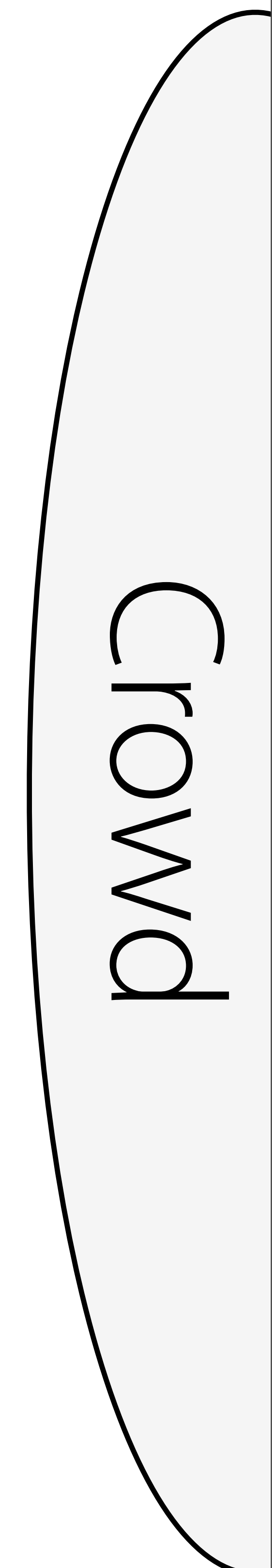
Worker recruitment time: Poisson process, rate μ

Retainer Queue

Loss

All servers busy





c workers, no waiting queue

Task arrivals: Poisson process, rate λ

Worker recruitment time: Poisson process, rate μ

Retainer Queue

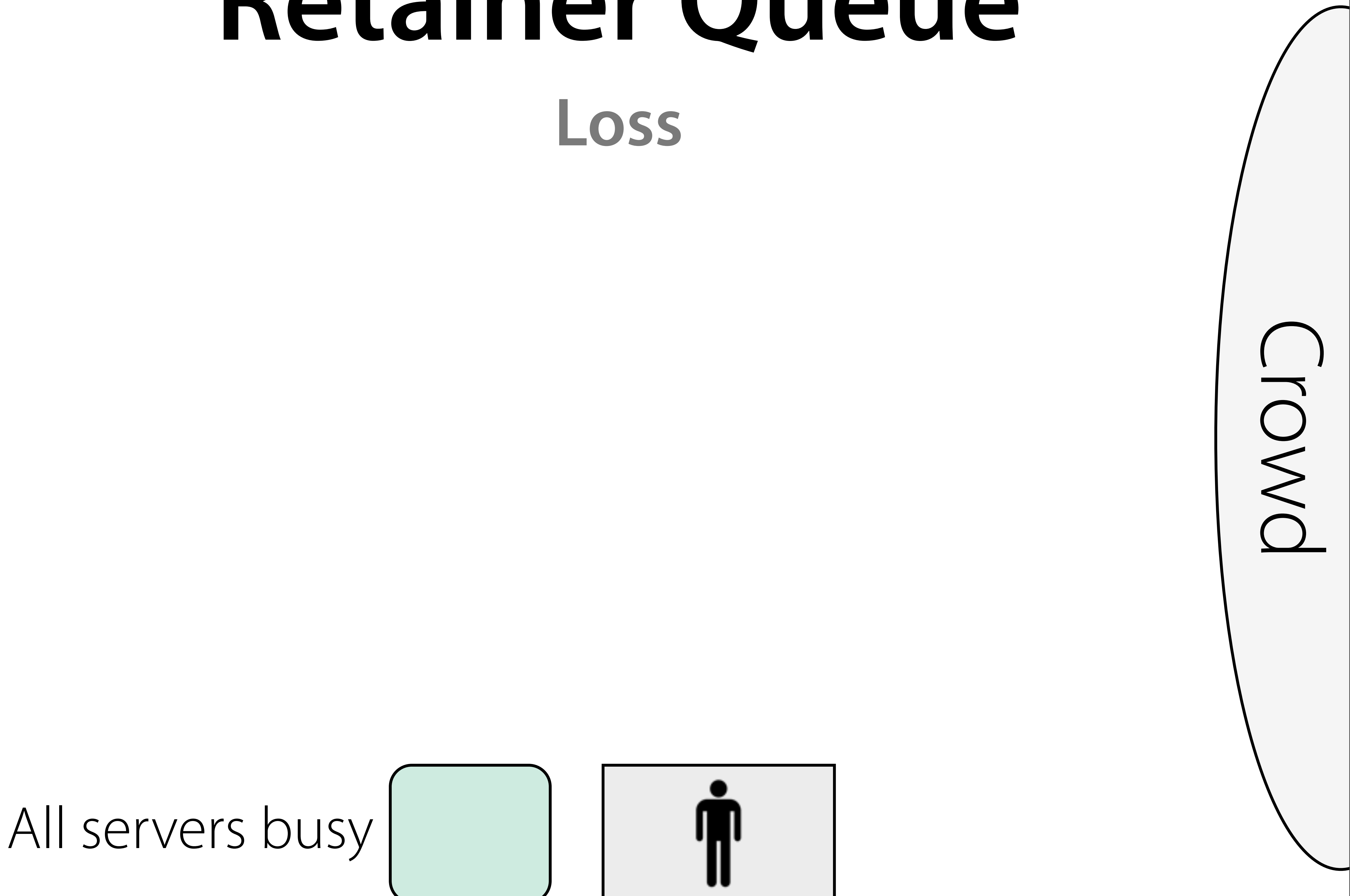
Loss

Crowd

All servers busy

Retainer Queue

Loss

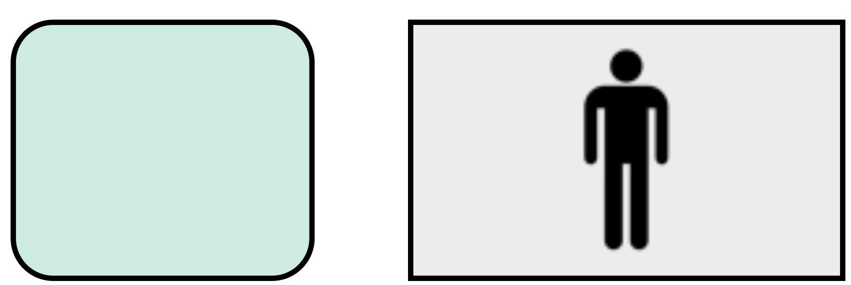


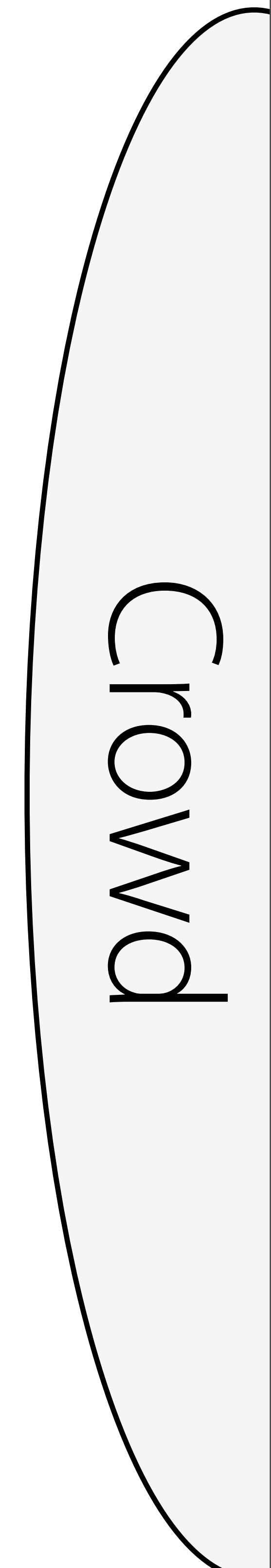
Retainer Queue

Loss

$$P(i \text{ servers busy})$$

All servers busy

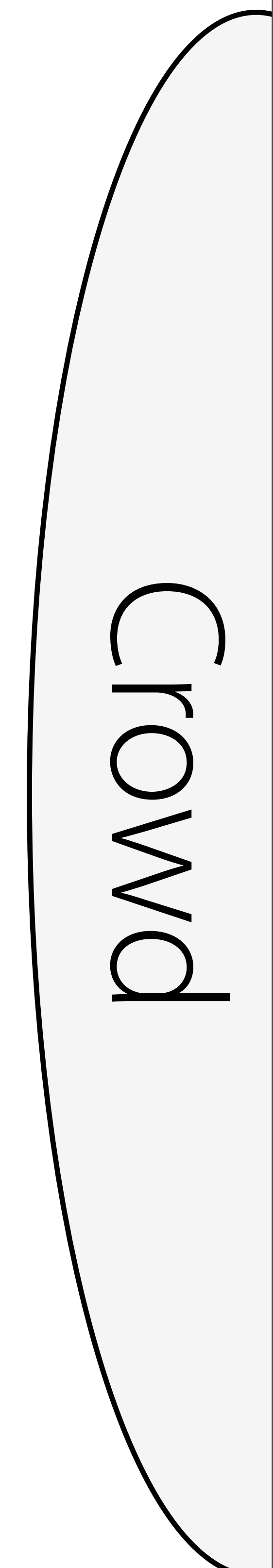




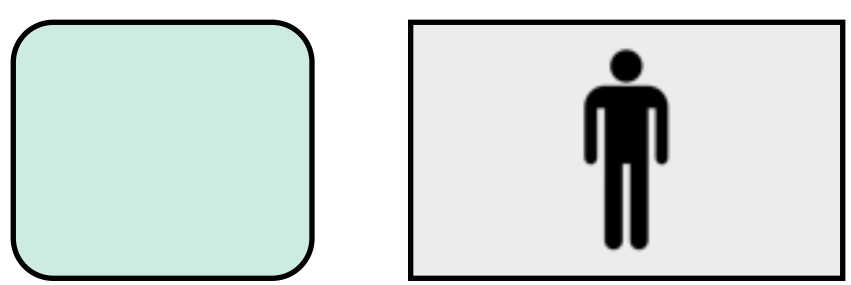
Retainer Queue

Loss

$$P(i \text{ servers busy}) = \pi(i)$$



All servers busy



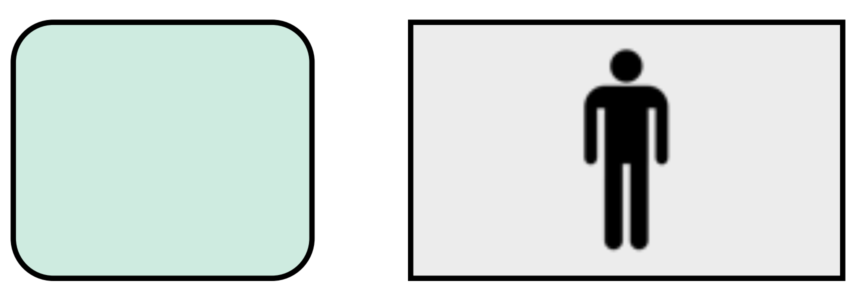
Retainer Queue

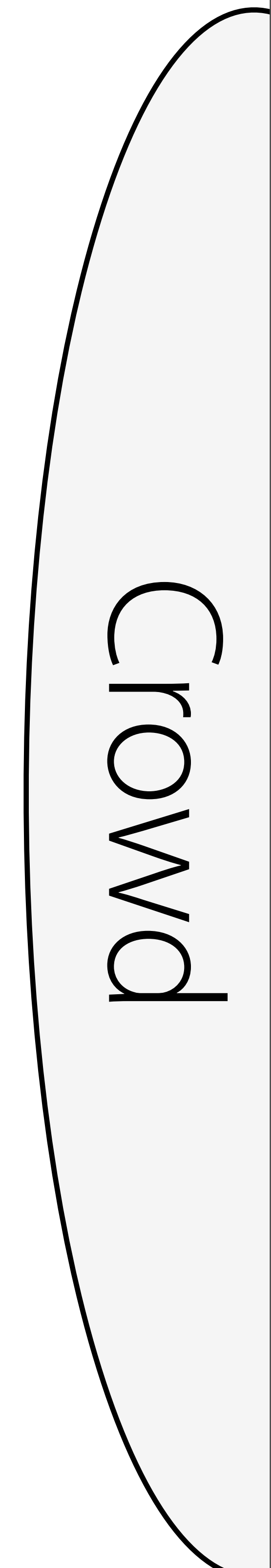
Loss

$$P(i \text{ servers busy}) = \pi(i)$$

$$P(\text{all servers busy})$$

All servers busy



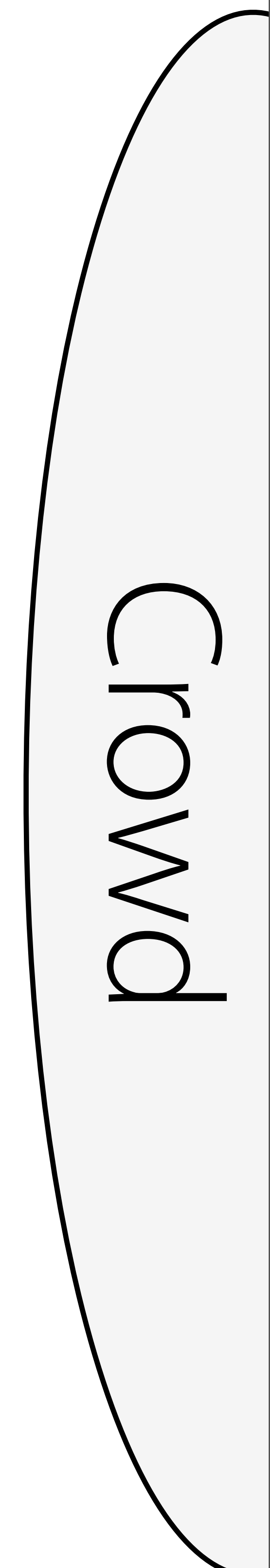


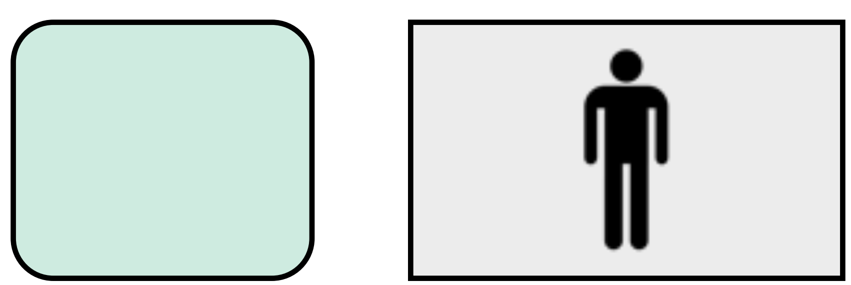
Retainer Queue

Loss

$$P(i \text{ servers busy}) = \pi(i)$$

$$P(\text{all servers busy}) = \pi(c)$$



All servers busy 

Model Predictions

1. Probability that all workers are busy: $\pi(c)$
   → the task has to wait for expected time $1/\mu$
2. Cost of keeping a retainer pool of size *c*
   → cost depends on number of *idle* servers

Probability of Loss

- Draw on Erlang's Loss Formula from queueing theory: probability of a rejected request in an M/M/c/c queue
- Let ρ be the traffic intensity:
  $\rho = \lambda/\mu$
  (roughly, the number of new tasks that will arrive in the time it takes to recruit a worker)

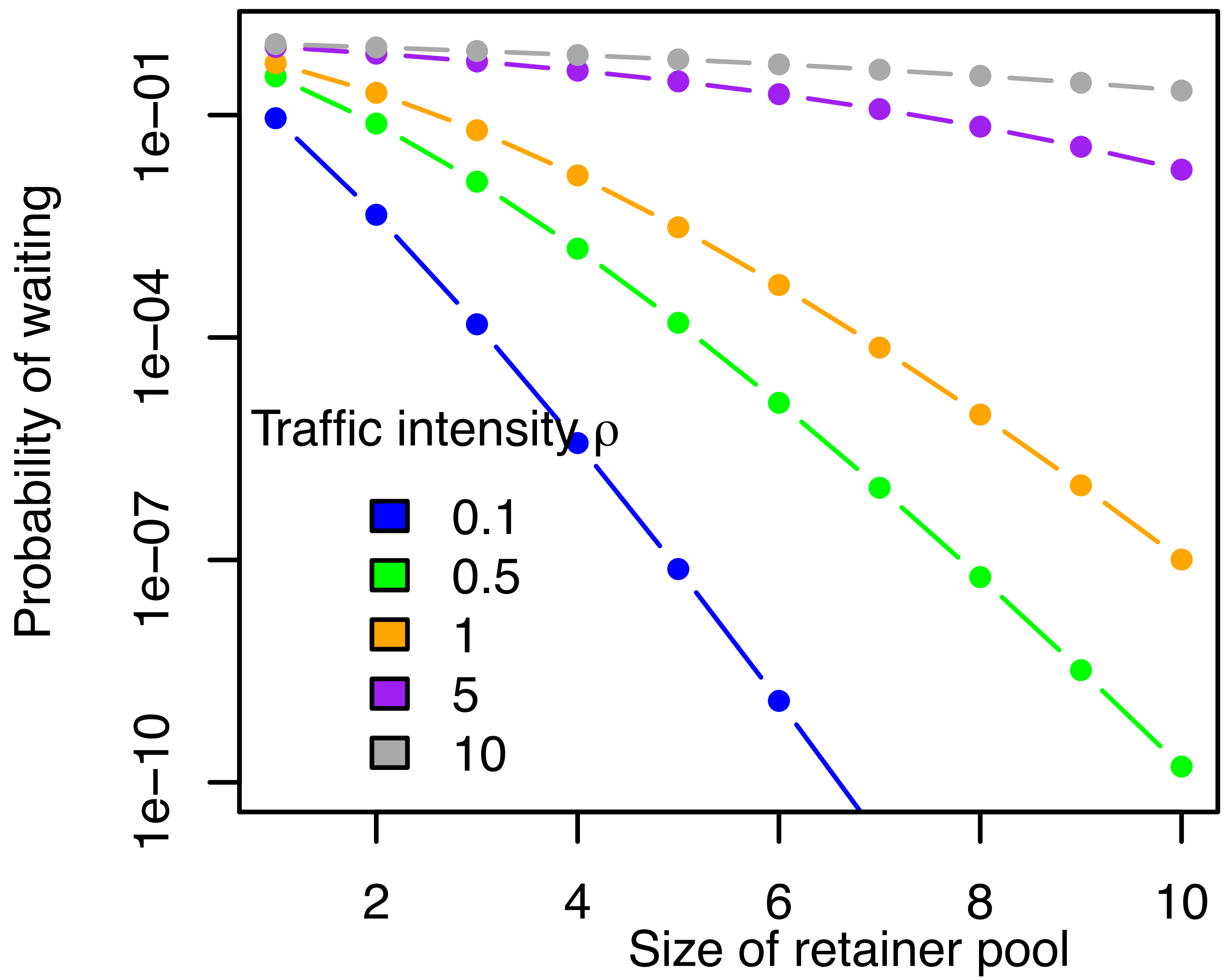
Probability of Loss

Erlang's Loss Formula says:

$$\pi(c) = P(c \text{ servers busy})$$
$$= \frac{\rho^c / c!}{\sum_{i=0}^{c} \rho^i / i!}$$

Remarkably, this result makes no assumptions about the arrival distribution.

Probability of Loss



Expected Waiting Time

$$P(c\ servers\ busy) \times (expected\ recruitment\ time)$$

$$= \pi(c)\frac{1}{\mu}$$

$$= \frac{\rho^c/c!}{\sum_{i=0}^{c} \rho^i/i!}\frac{1}{\mu}$$

Expected Cost

How much do we pay in steady-state?

Depends on how many workers are usually waiting on retainer.

Expected Cost

Probability of *i* busy servers in an M/M/c/c queue is a more general version of Erlang's Loss Formula:

$$\pi(i) = \frac{\rho^i / i!}{\sum_{i=0}^{c} \rho^i / i!}$$

Derive the expected number of busy workers:

$$E[i] = \rho[1 - \pi(c)]$$

Expected Cost

Probability of *i* busy servers in an M/M/c/c queue is a more general version of Erlang's Loss Formula:

$$\pi(i) = \frac{\rho^i / i!}{\sum_{i=0}^{c} \rho^i / i!}$$

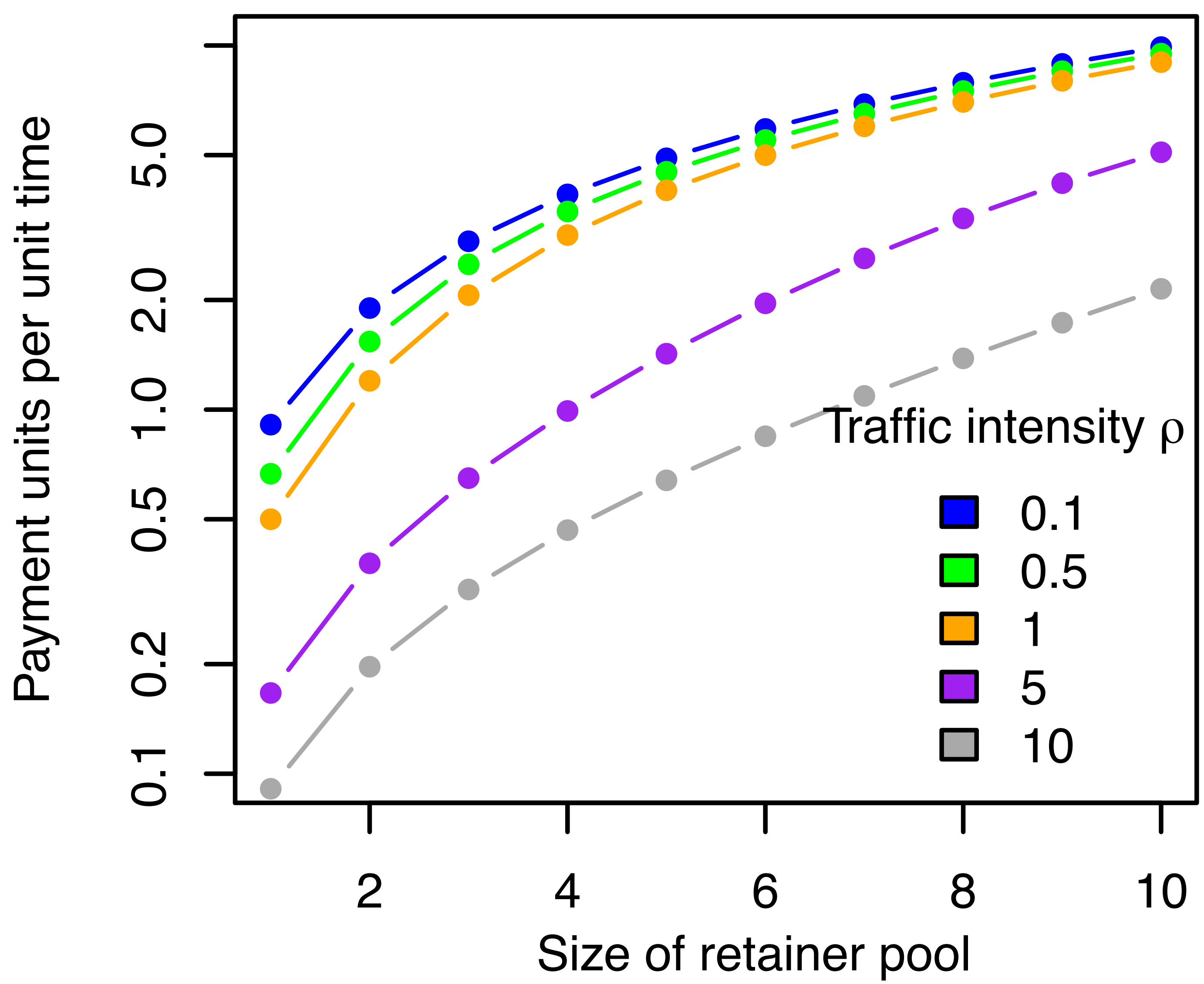
Derive the expected number of busy workers:

$$E[i] = \rho[1 - \pi(c)]$$

Total cost is the number of *idle* workers:

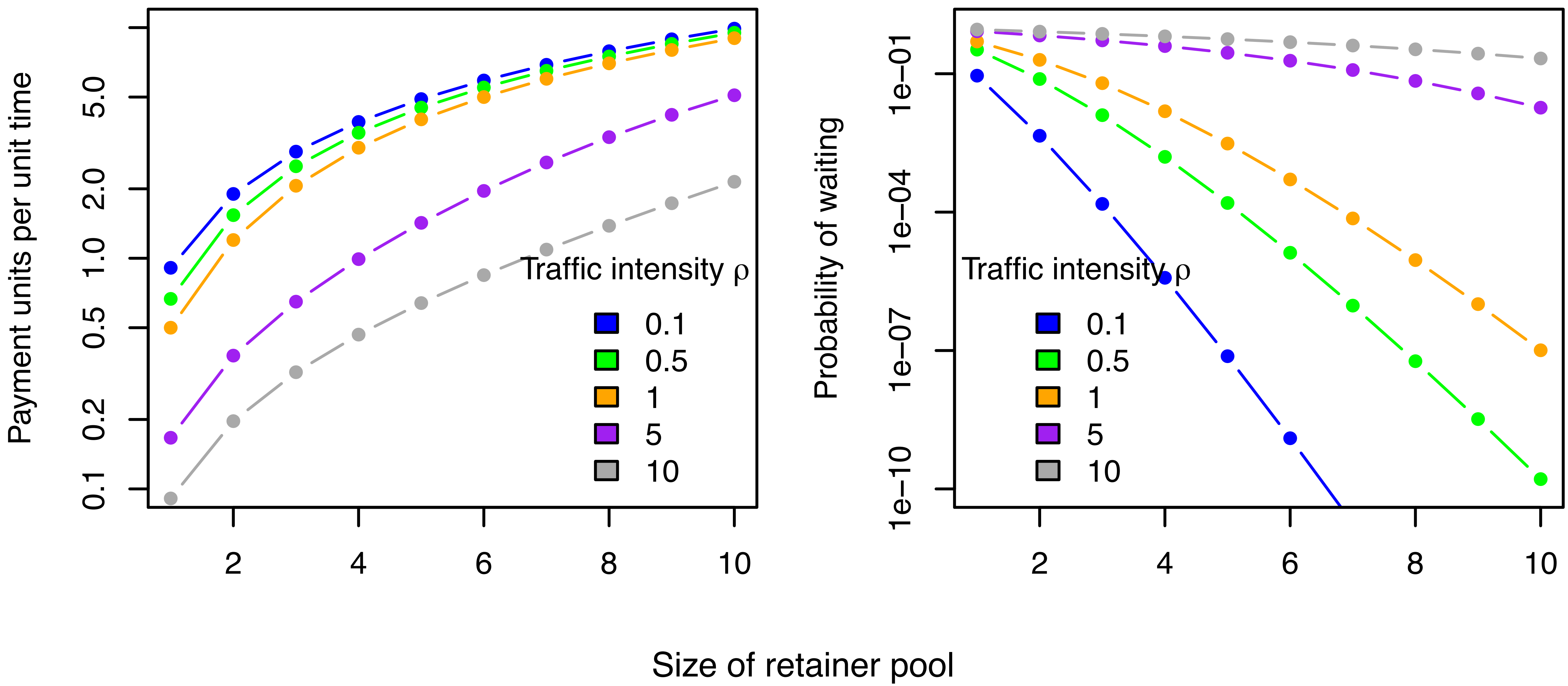
$$c - \rho[1 - \pi(c)]$$

Expected Cost



Cost goes down when $c < \rho$,
but performance suffers.

Expected Cost



Cost goes down when $c < \rho$,

but performance suffers.

Optimal Retainer Size

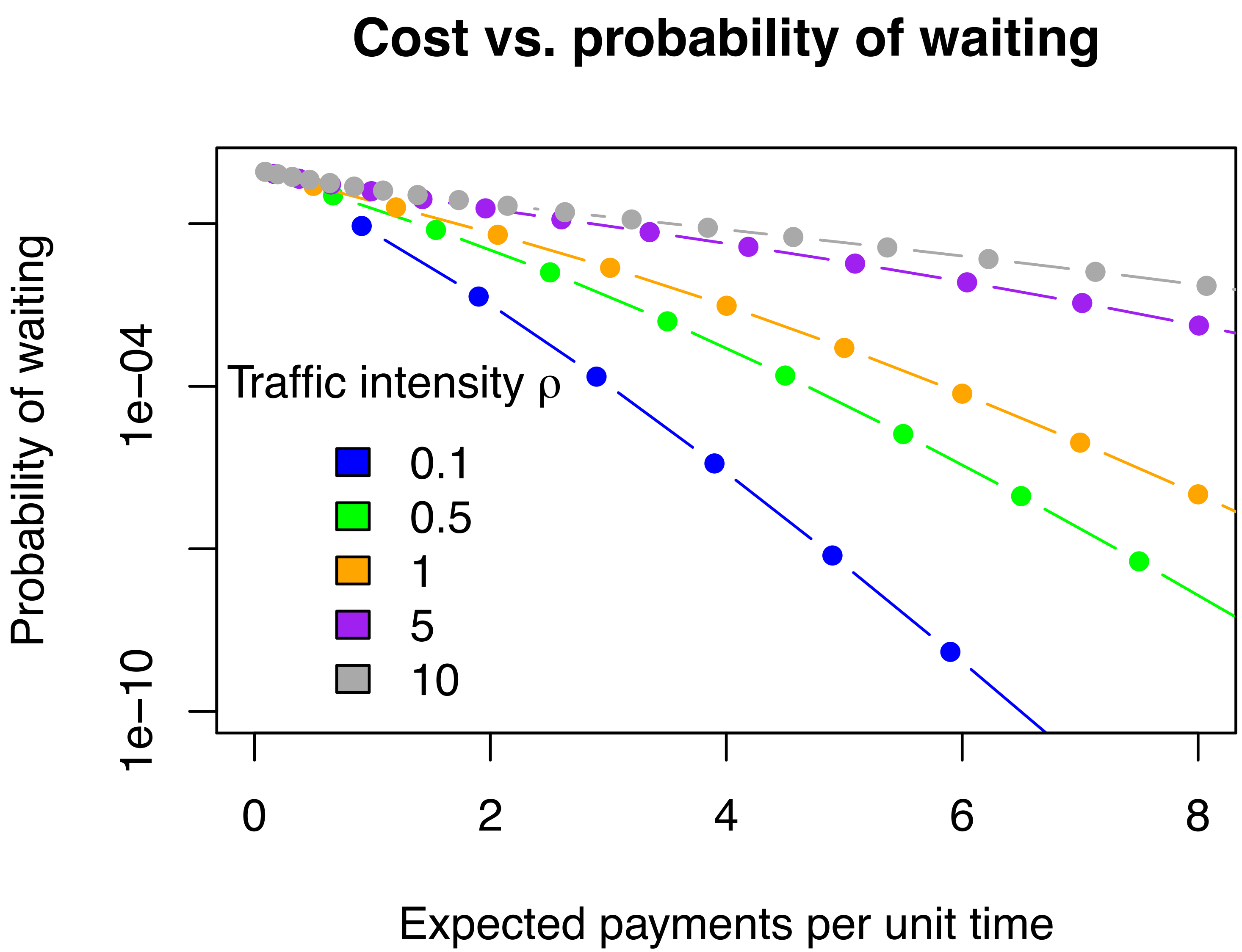
- Size of retainer pool is typically the only value that requesters can manipulate
- Minimize costs by keeping the retainer pool small while keeping $\pi(c)$ low

Optimal Retainer Size

Based on Maximum Miss Probability

Given a maximum desired probability of a miss p_{max}:

Minimize c subject to $\pi(c) \leq p_{max}$

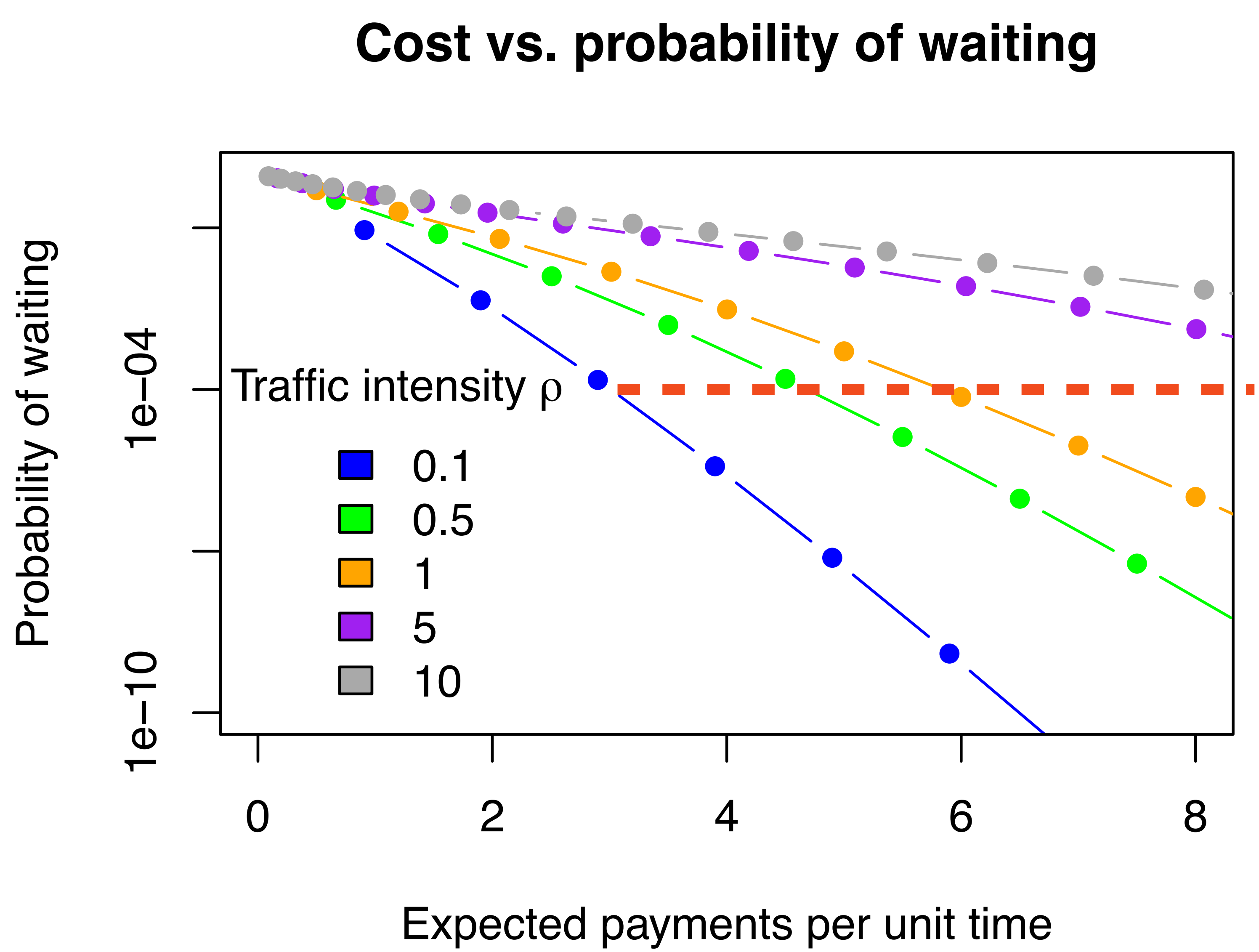


Optimal Retainer Size

Based on Maximum Miss Probability

Given a maximum desired probability of a miss p_{max}:

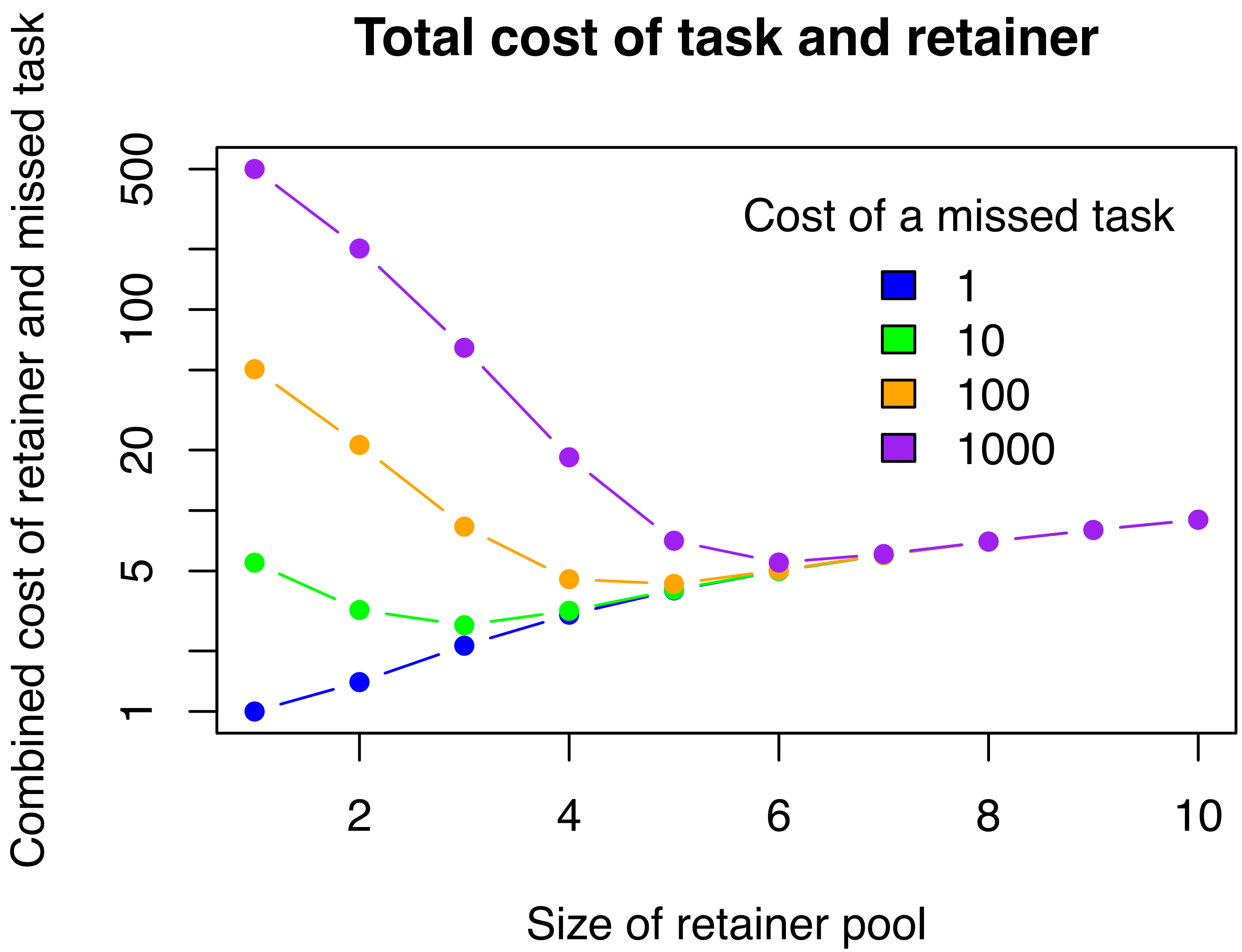
Minimize c subject to $\pi(c) \leq p_{max}$



Optimal Retainer Size

Based on Joint Cost

If the "pizza delivery" property holds: we can quantify the cost of loss



Improving the Retainer Model

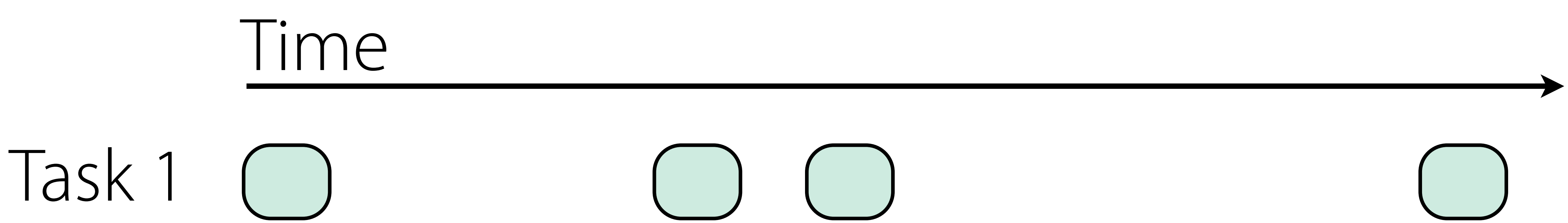
1. Subscriptions
2. Shared Pools
3. Predictive Recruitment

Retainer Subscriptions

- Proposal: increase μ by allowing workers to **subscribe** to realtime tasks
- Instead of posting to the global task list, the platform sends a message to subscribers
- Change crowdsourcing from a *pull* model to a *push* model

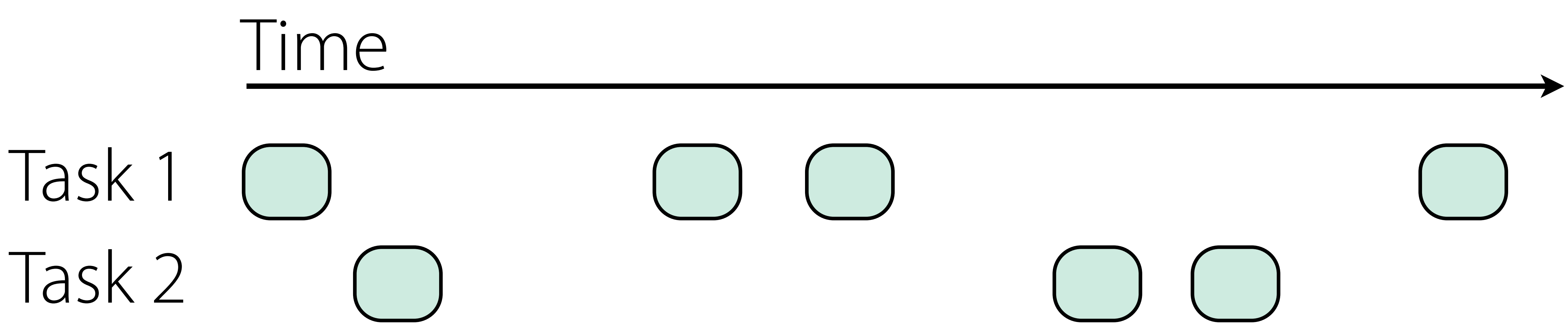
Global Retainer Pools

- Sharing one global retainer pool across requesters improves performance
- Intuition: Most workers are padding for unlikely runs of arrivals)



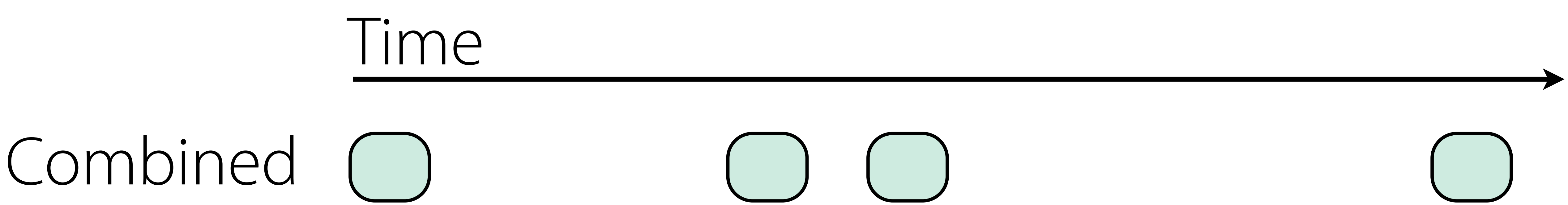
Global Retainer Pools

- Sharing one global retainer pool across requesters improves performance
- Intuition: Most workers are padding for unlikely runs of arrivals)



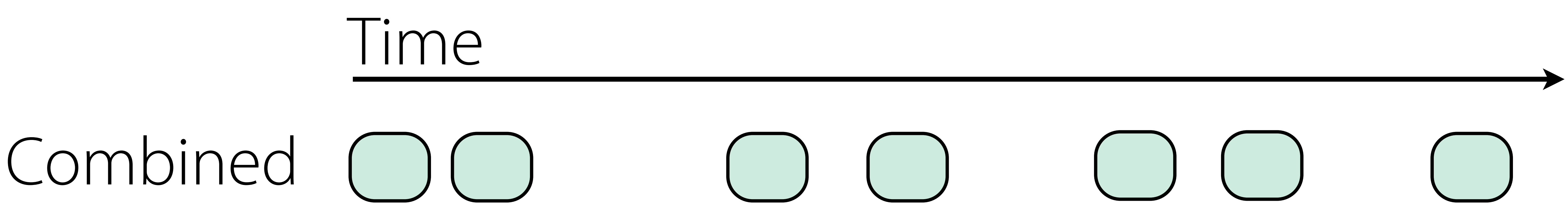
Global Retainer Pools

- Sharing one global retainer pool across requesters improves performance
- Intuition: Most workers are padding for unlikely runs of arrivals)



Global Retainer Pools

- Sharing one global retainer pool across requesters improves performance
- Intuition: Most workers are padding for unlikely runs of arrivals)



Global Retainer Pools

- Through approximation, individual pools:

$$\pi(c) \approx \sqrt{2\pi c}\left(e^{-\rho}(e\rho/c)^{c}\right)$$

- Shared pools across *k* requesters:

$$\pi(c) \approx \sqrt{2\pi kc}\left(e^{-\rho}(e\rho/c)^{c}\right)^{k}$$

- Loss rate declines exponentially with the number of bundled retainer pools

Global Retainer Pools

- Through approximation, individual pools:

$$\pi(c) \approx \sqrt{2\pi c}\left(e^{-\rho}(e\rho/c)^{c}\right)$$

- Shared pools across *k* requesters:

$$\pi(c) \approx \sqrt{2\pi k c}\left(e^{-\rho}(e\rho/c)^{c}\right)^{k}$$

- Loss rate declines exponentially with the number of bundled retainer pools

Global Retainer Pools

Cost dramatically decreases
as you combine retainers:
k dollars to log(k) dollars

Global Retainer Routing

- Not every worker in a global retainer pool is good at every task
- If we assigned each worker to any task they could do, some tasks would starve

Global Retainer Routing

- We want to maintain a buffer of workers to respond to all kinds of tasks
- A linear programming technique can balance the traffic intensities across all tasks

Precruitment

- Predictive Recruitment: notify workers **before** the task arrives
- Recall workers in expectation of having a task by the time they arrive 2–3 seconds later

Precruitment

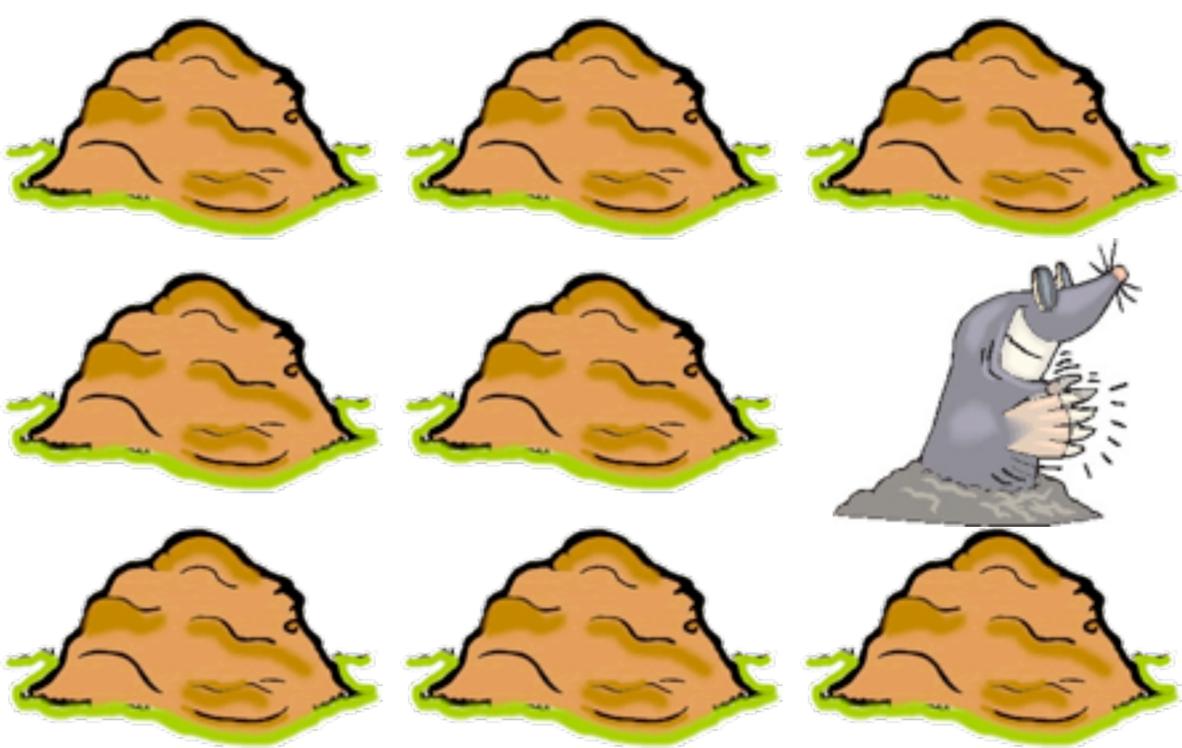
Formative Study, N=373 tasks

- 3¢ for 3-minute retainer task: whack-a-mole
- 'Loading...' screen for randomly-selected time [0, 20] seconds after worker returns
- Click on randomly-placed mole

Precruitment

Formative Study, N=373 tasks

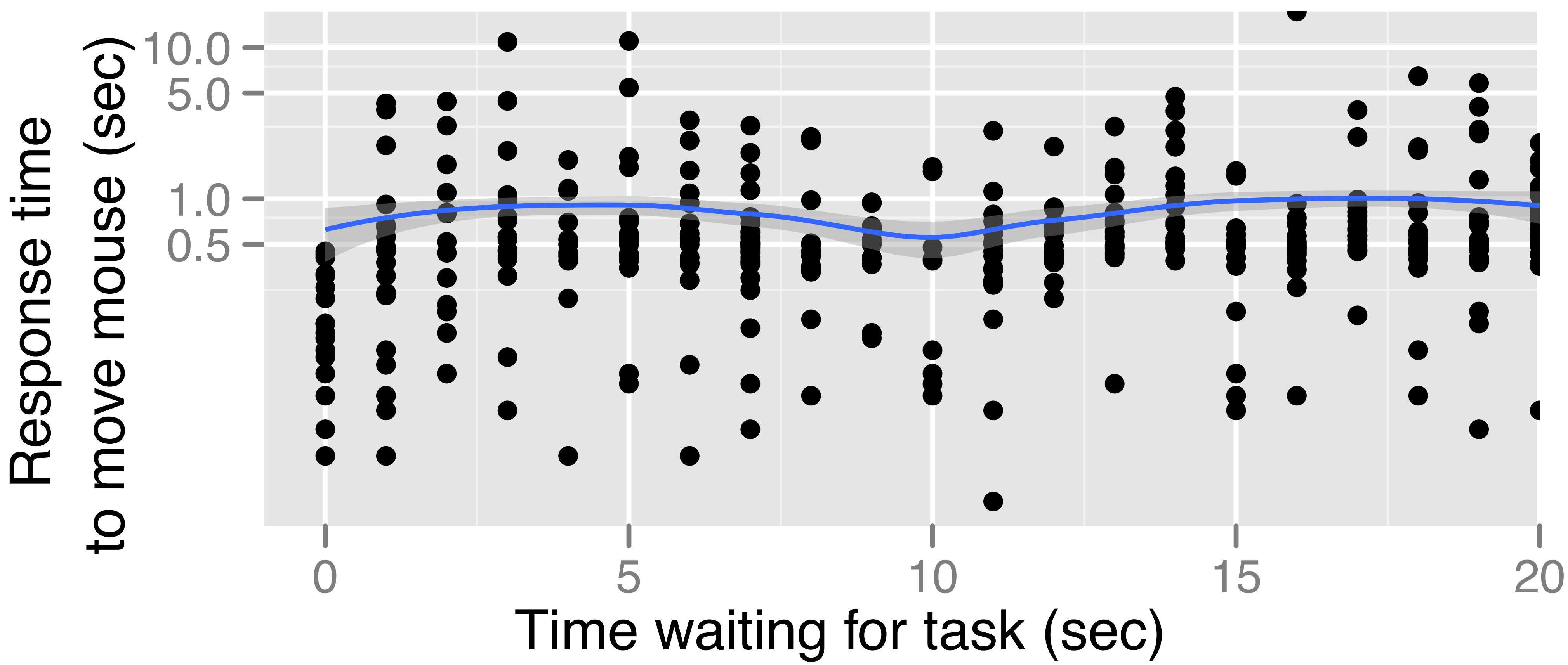
- 3¢ for 3-minute retainer task: whack-a-mole
- 'Loading...' screen for randomly-selected time [0, 20] seconds after worker returns
- Click on randomly-placed mole



Precruitment

Results

- Median time to mouse move: 0.50 seconds



- Standard retainer model (start timer @ alert): median mouse move in 1.36 seconds

Discussion

- Empirics: Can deployed crowdsourcing platforms support lots of realtime tasks?
- Theory: Crowds as queueing systems
- Reputation: median response time, overall response rate

Use **queueing theory** to **understand** and **optimize performance** of a paid, realtime crowdsourcing platform.

- Relationship between crowd size and response time
- Algorithm for optimizing crowd size vs. response time
- Improvements to the platform: 500 millisecond feedback

Analytic Methods for Optimizing Realtime Crowdsourcing

Michael Bernstein, David Karger, Rob Miller, and Joel Brandt

MIT CSAIL and Adobe Systems

CSAIL Adobe

MIT HUMAN-COMPUTER INTERACTION