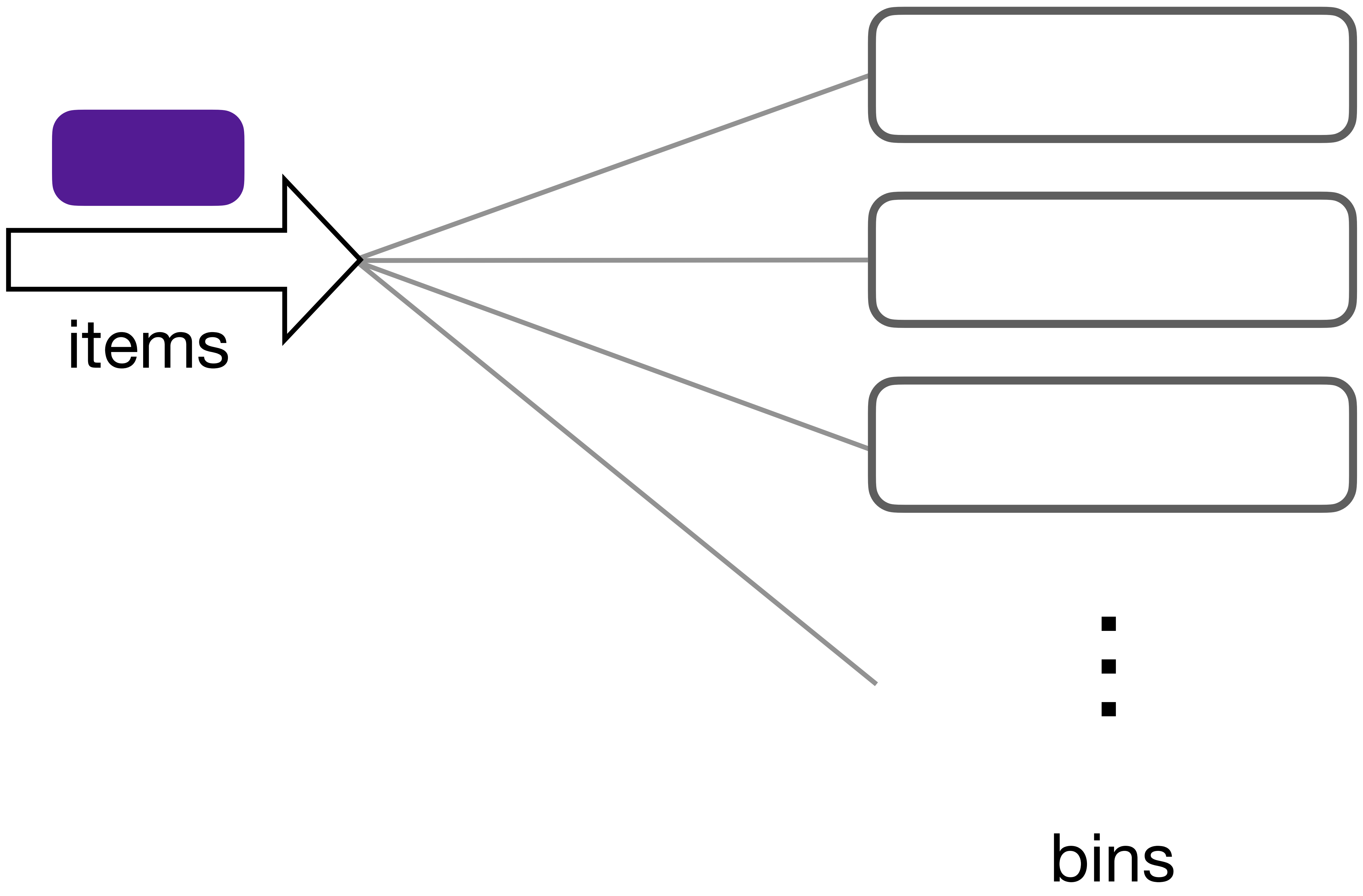
Stochastic Bin Packing with Time-Varying Item Sizes

Joint work with Yige Hong (CMU) and Qiaomin Xie (UW–Madison)

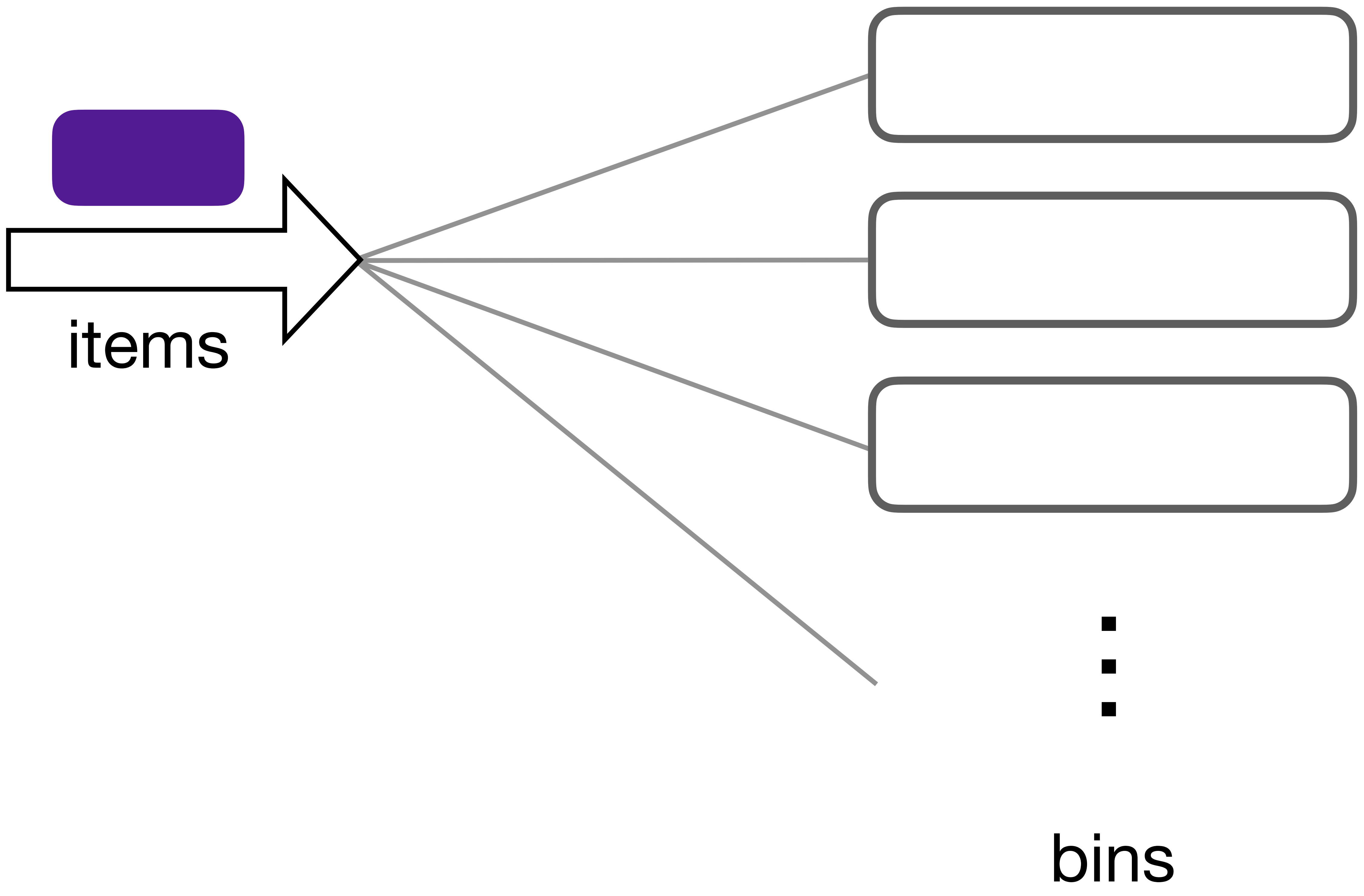
Weina Wang
Carnegie Mellon University

The problem



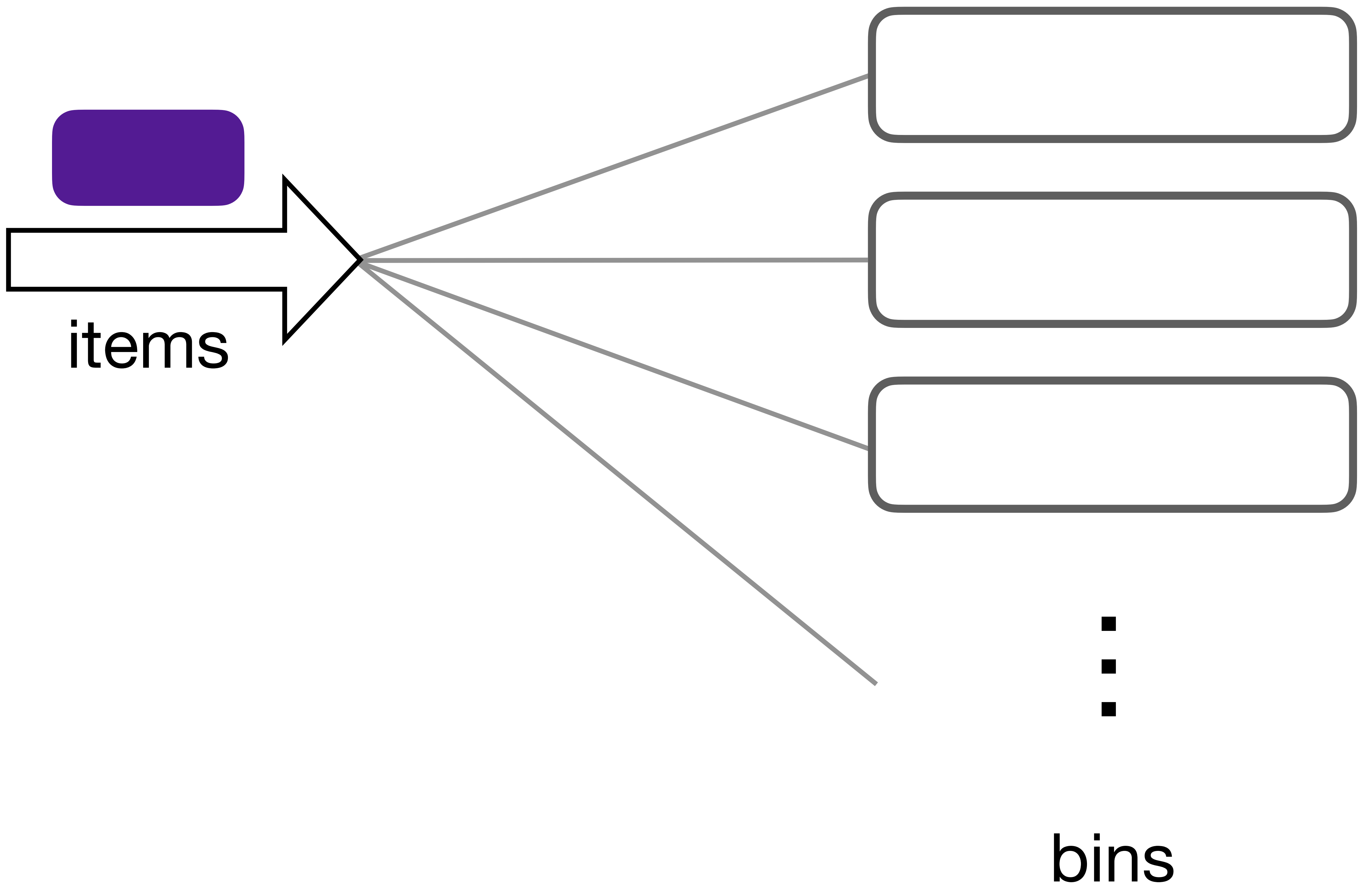
The problem

- Each arriving item needs to be assigned to a bin



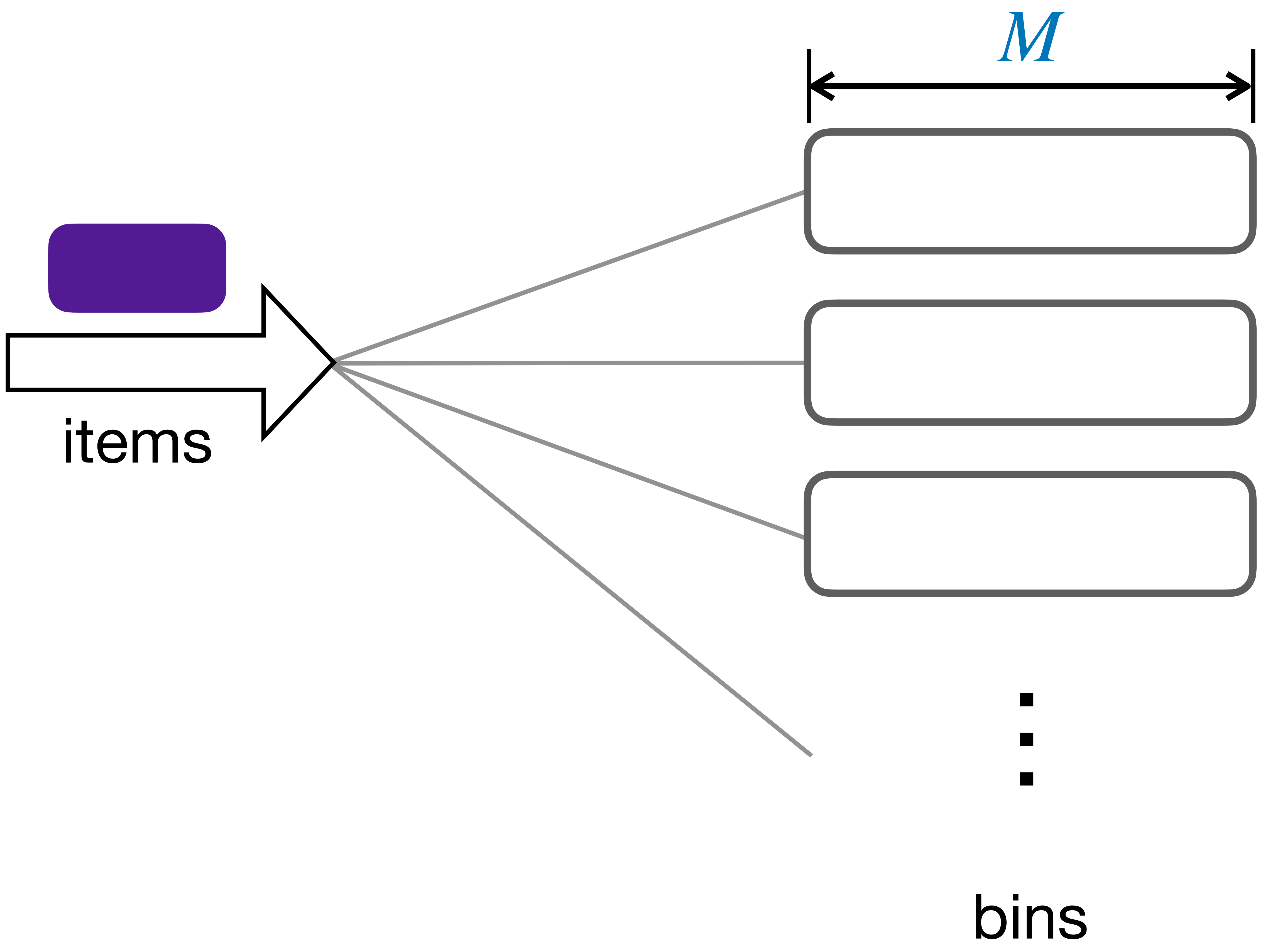
The problem

- Each arriving item needs to be assigned to a bin
- Infinite # bins



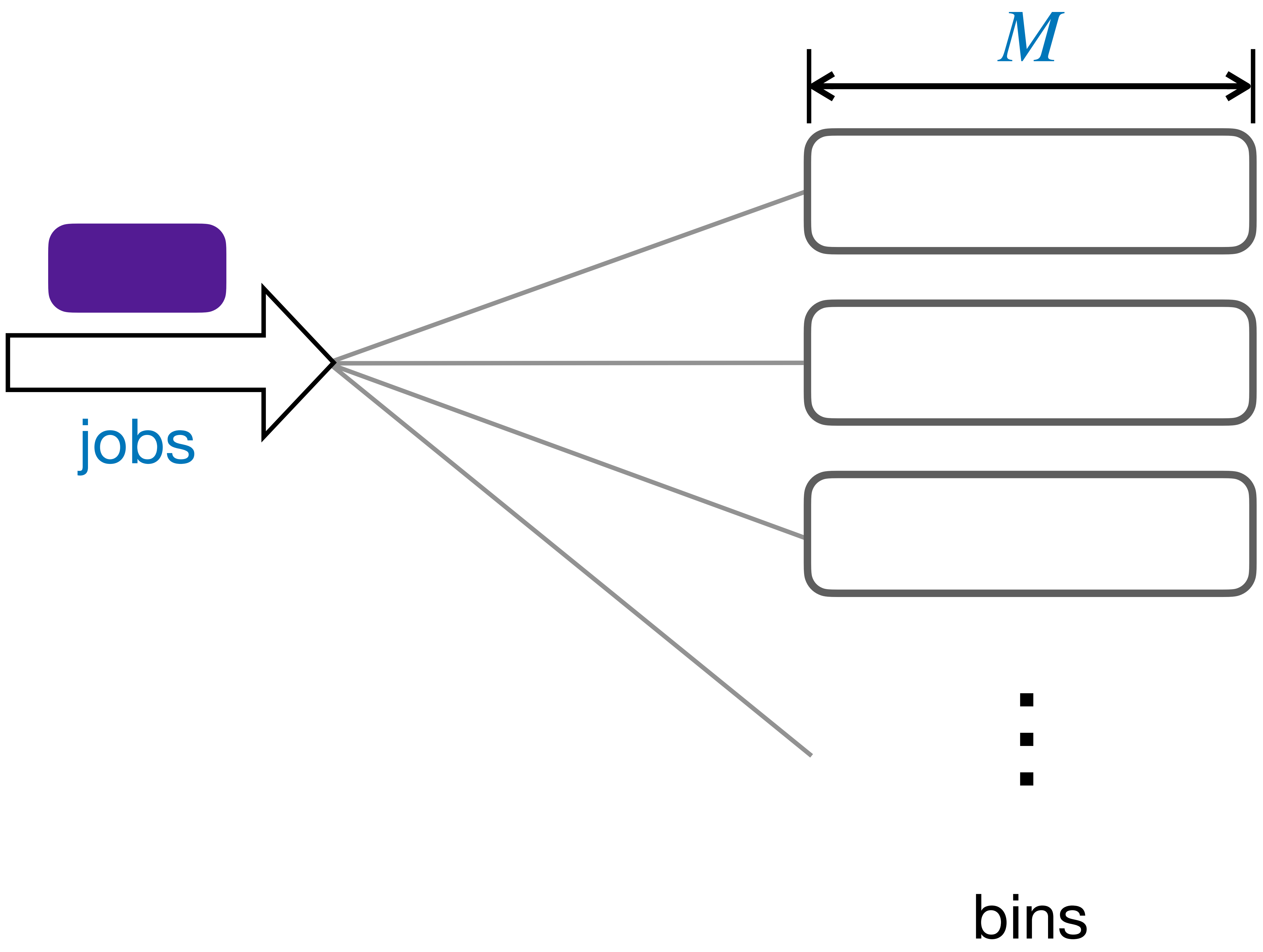
The problem

- Each arriving item needs to be assigned to a bin
- Infinite # bins
- Each bin has a capacity M



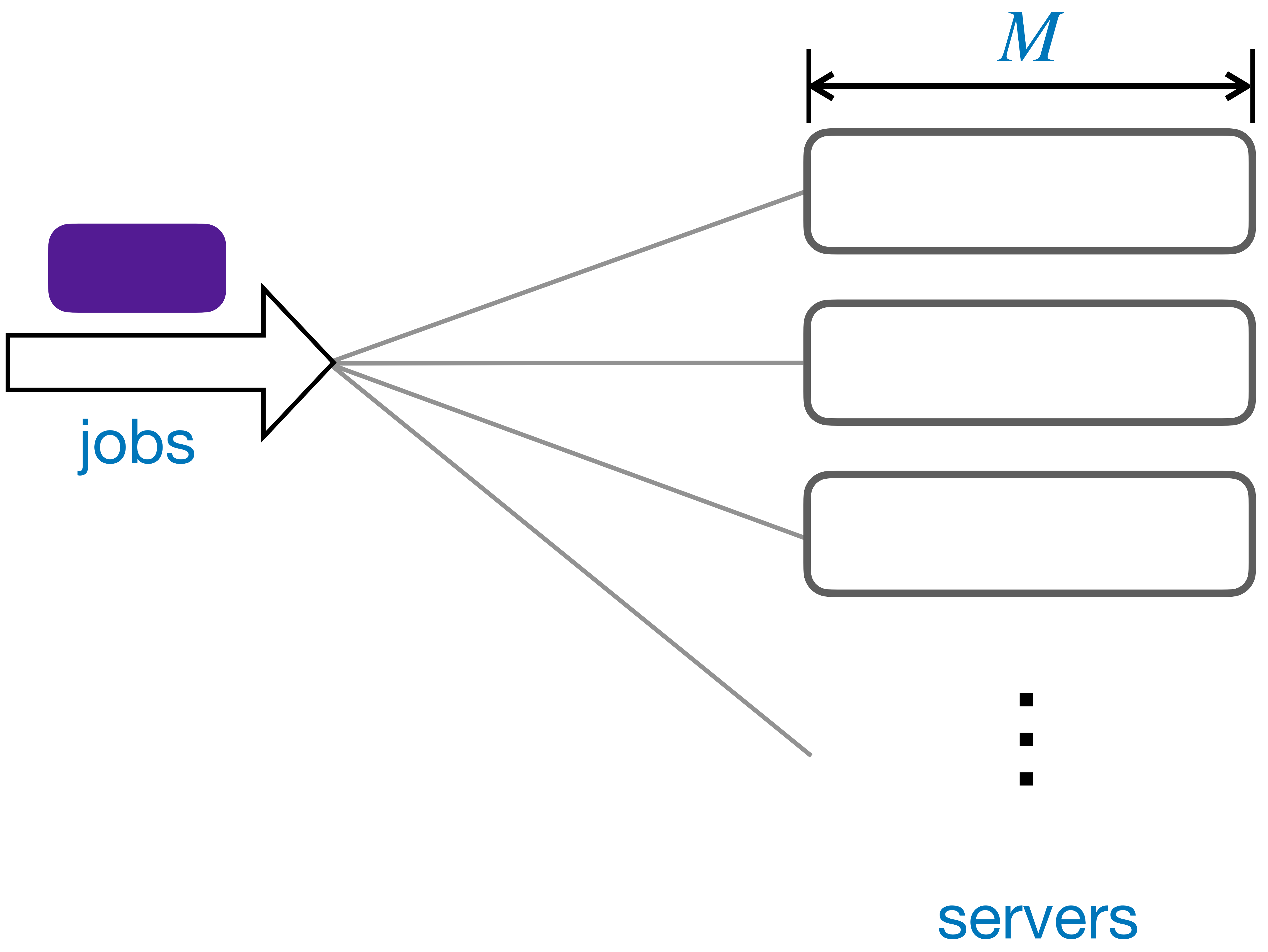
The problem

- Each arriving job needs to be assigned to a bin
- Infinite # bins
- Each bin has a capacity M



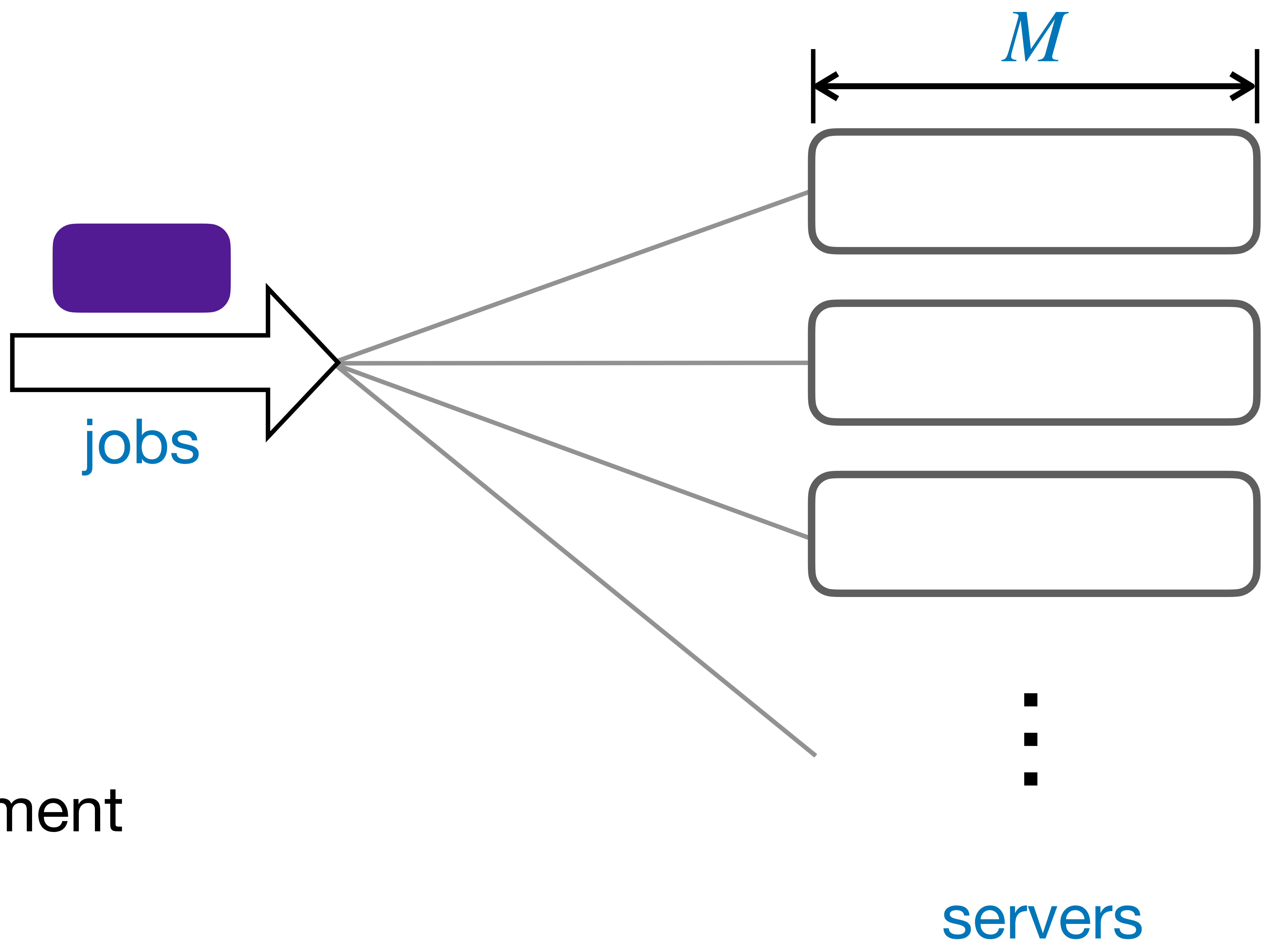
The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M



The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M

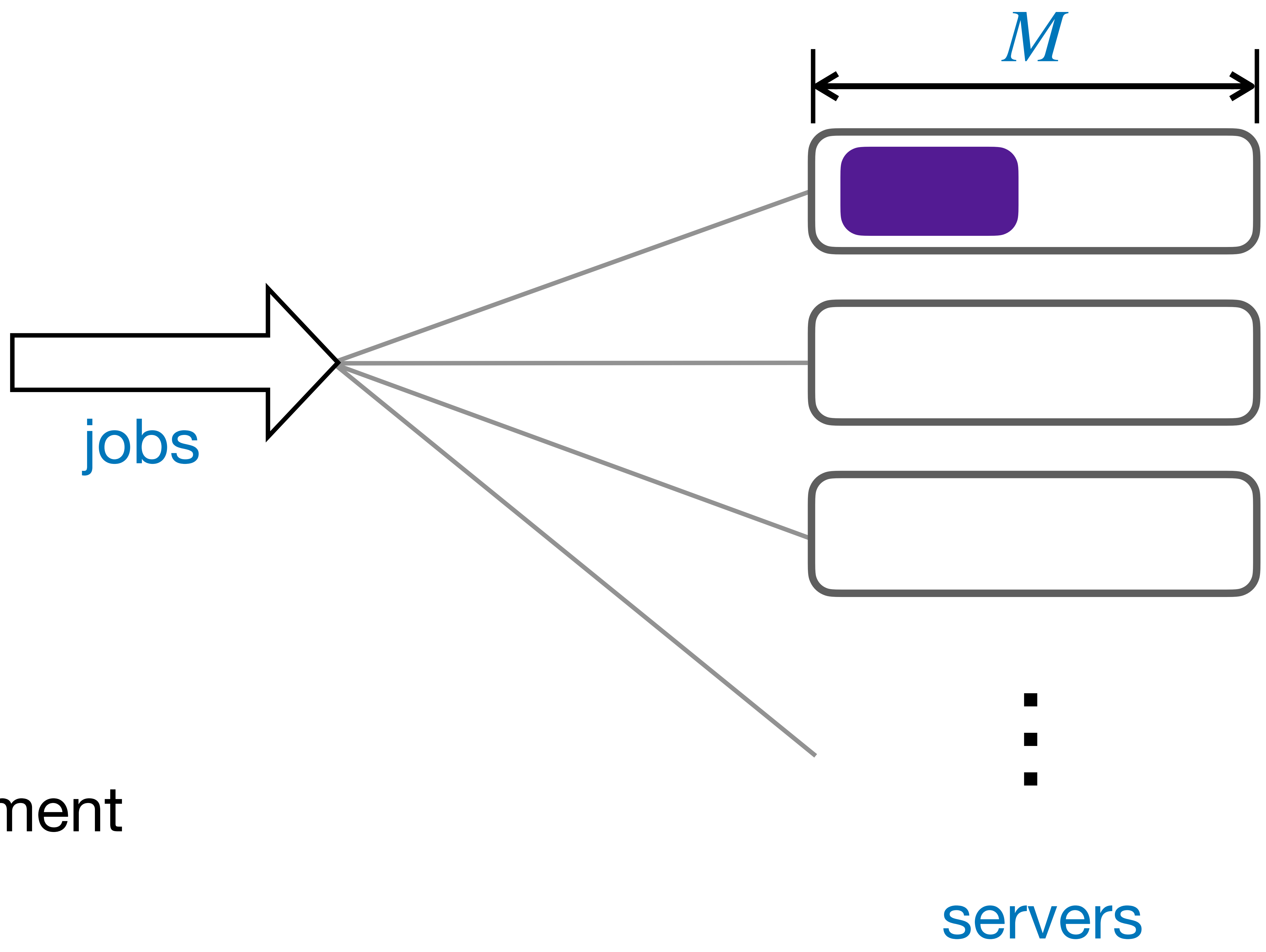


Traditional job model:

- Each job has a fixed resource requirement
- Each job departs after a random time

The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M

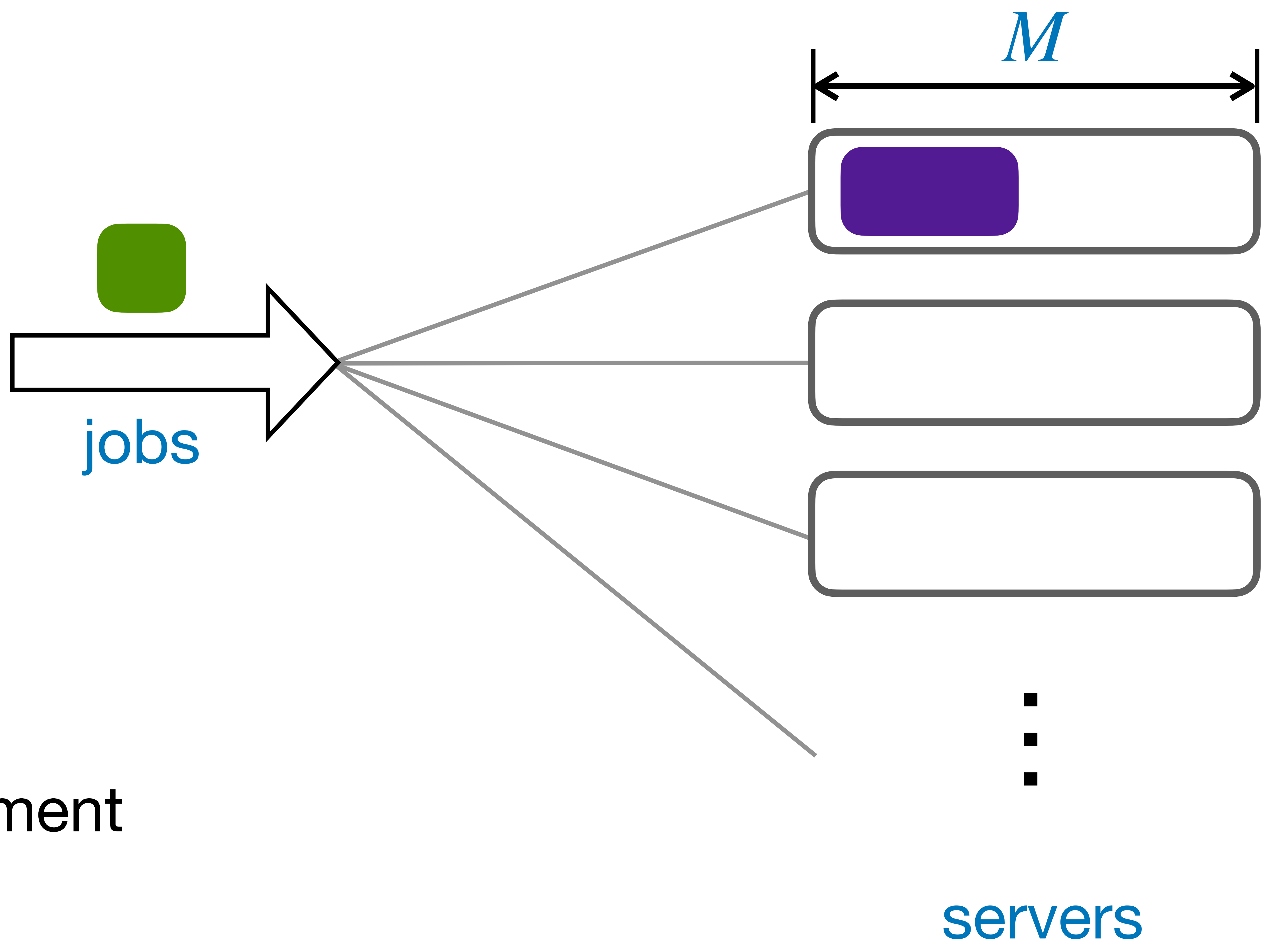


Traditional job model:

- Each job has a fixed resource requirement
- Each job departs after a random time

The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M

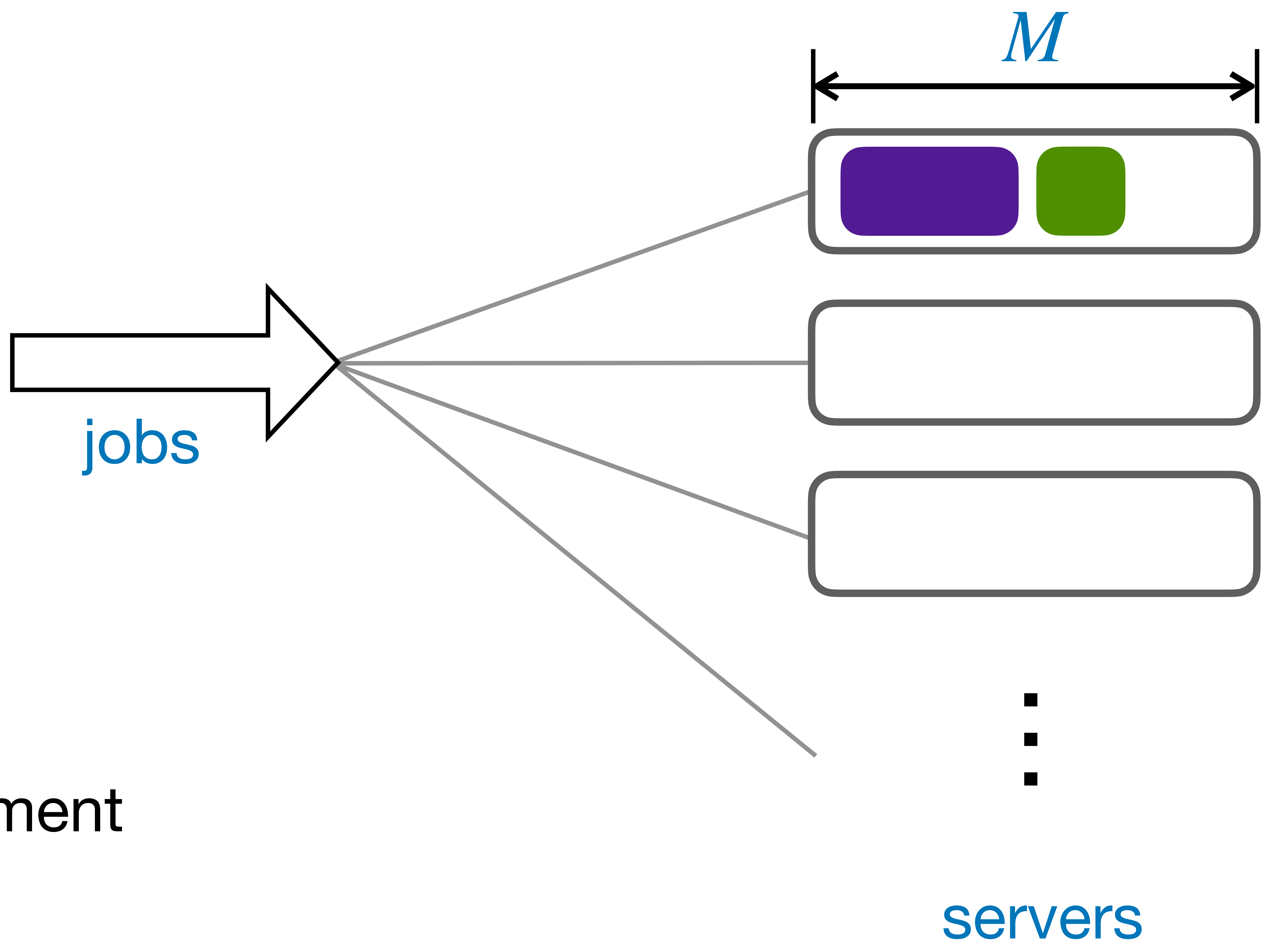


Traditional job model:

- Each job has a fixed resource requirement
- Each job departs after a random time

The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M



Traditional job model:

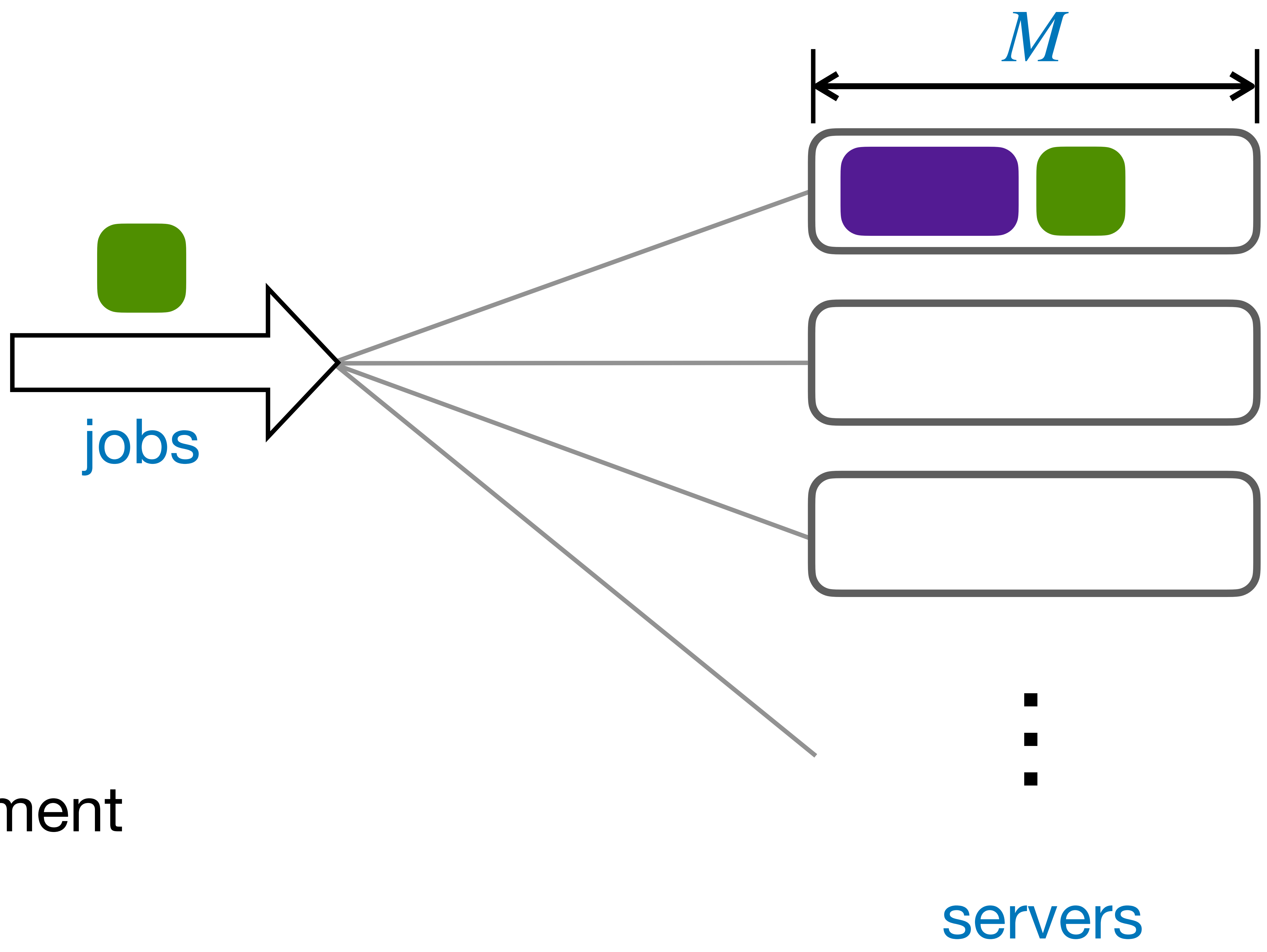
- Each job has a fixed resource requirement
- Each job departs after a random time

The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M

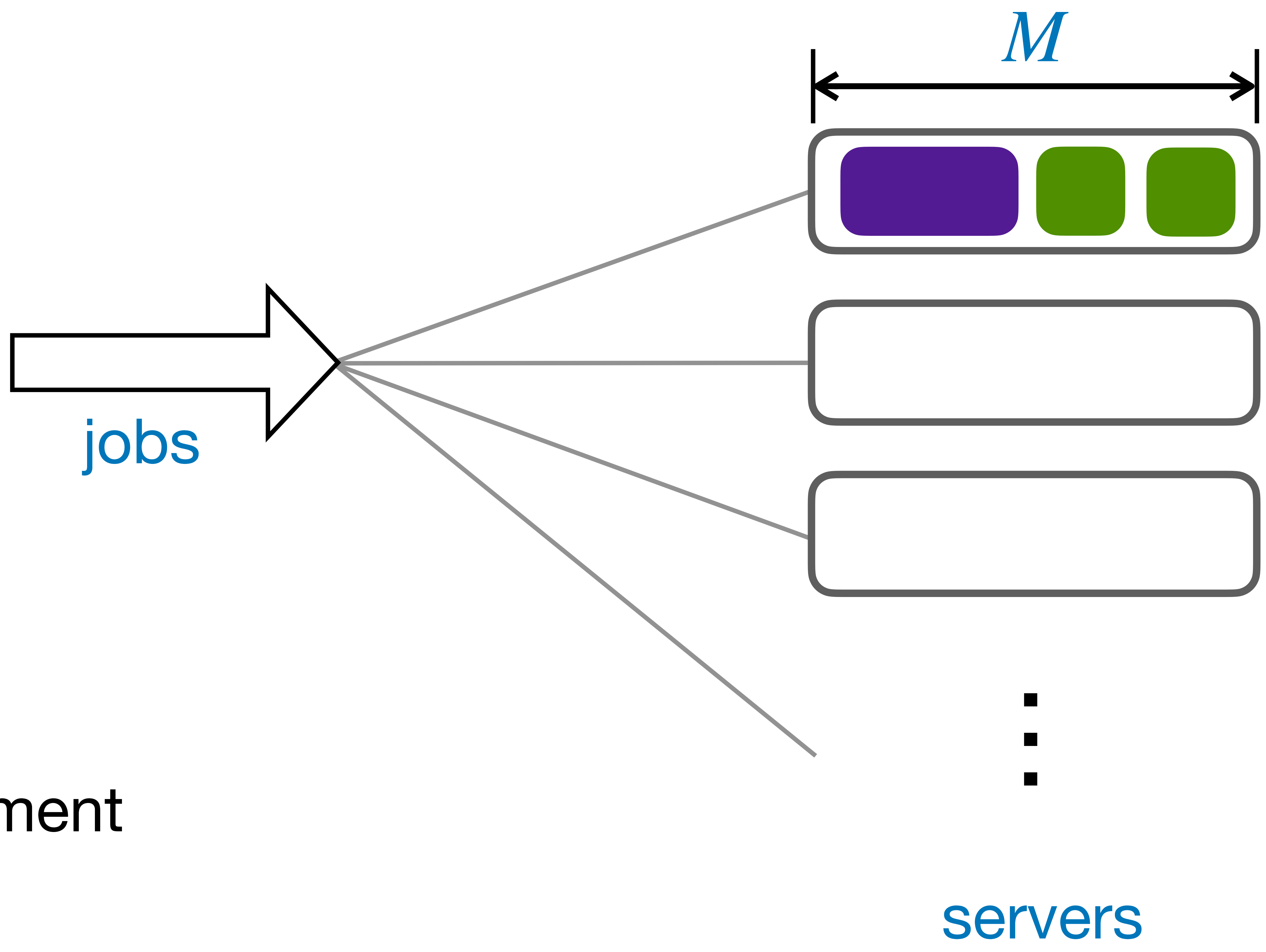
Traditional job model:

- Each job has a fixed resource requirement
- Each job departs after a random time



The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M

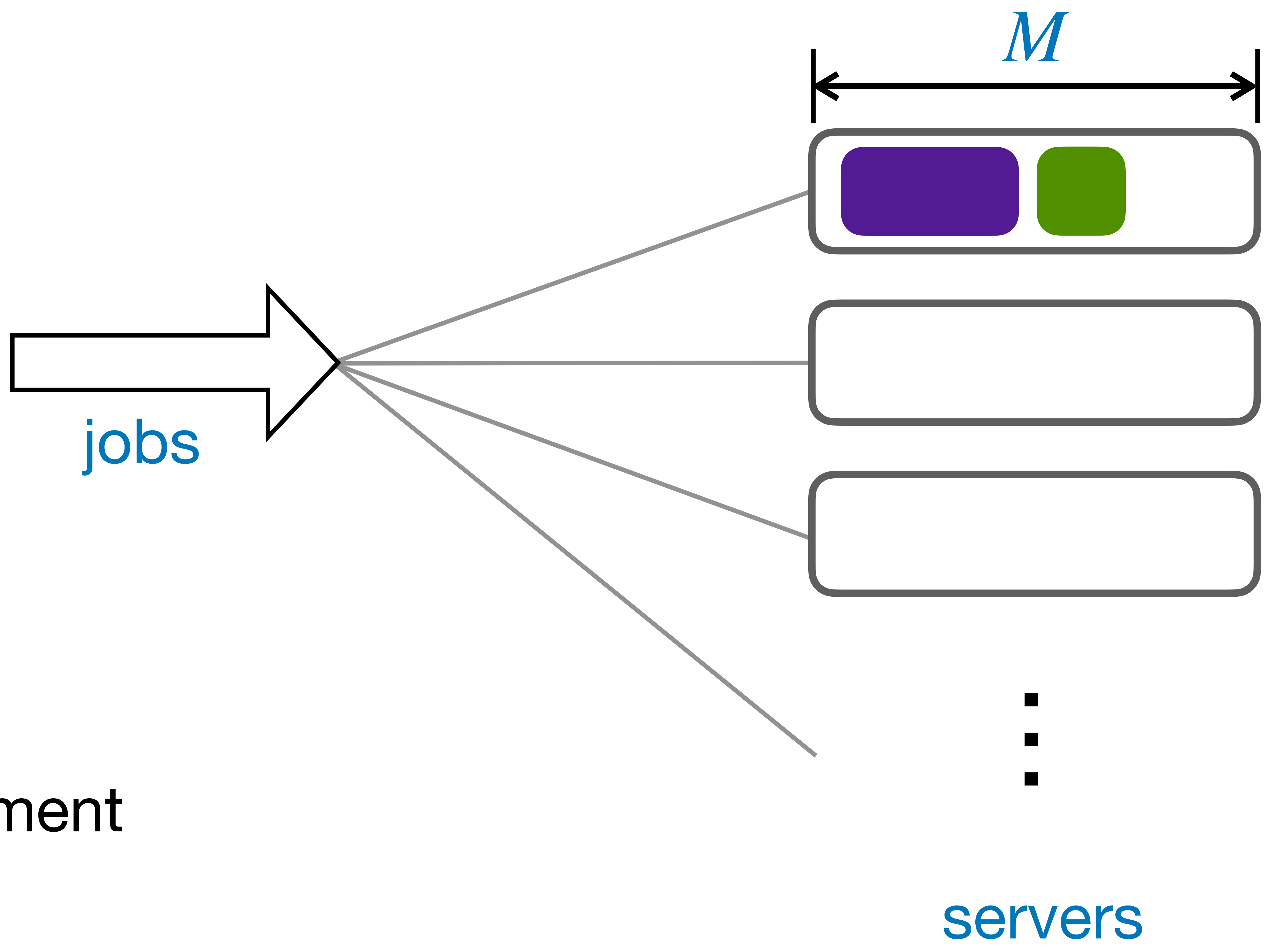


Traditional job model:

- Each job has a fixed resource requirement
- Each job departs after a random time

The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M

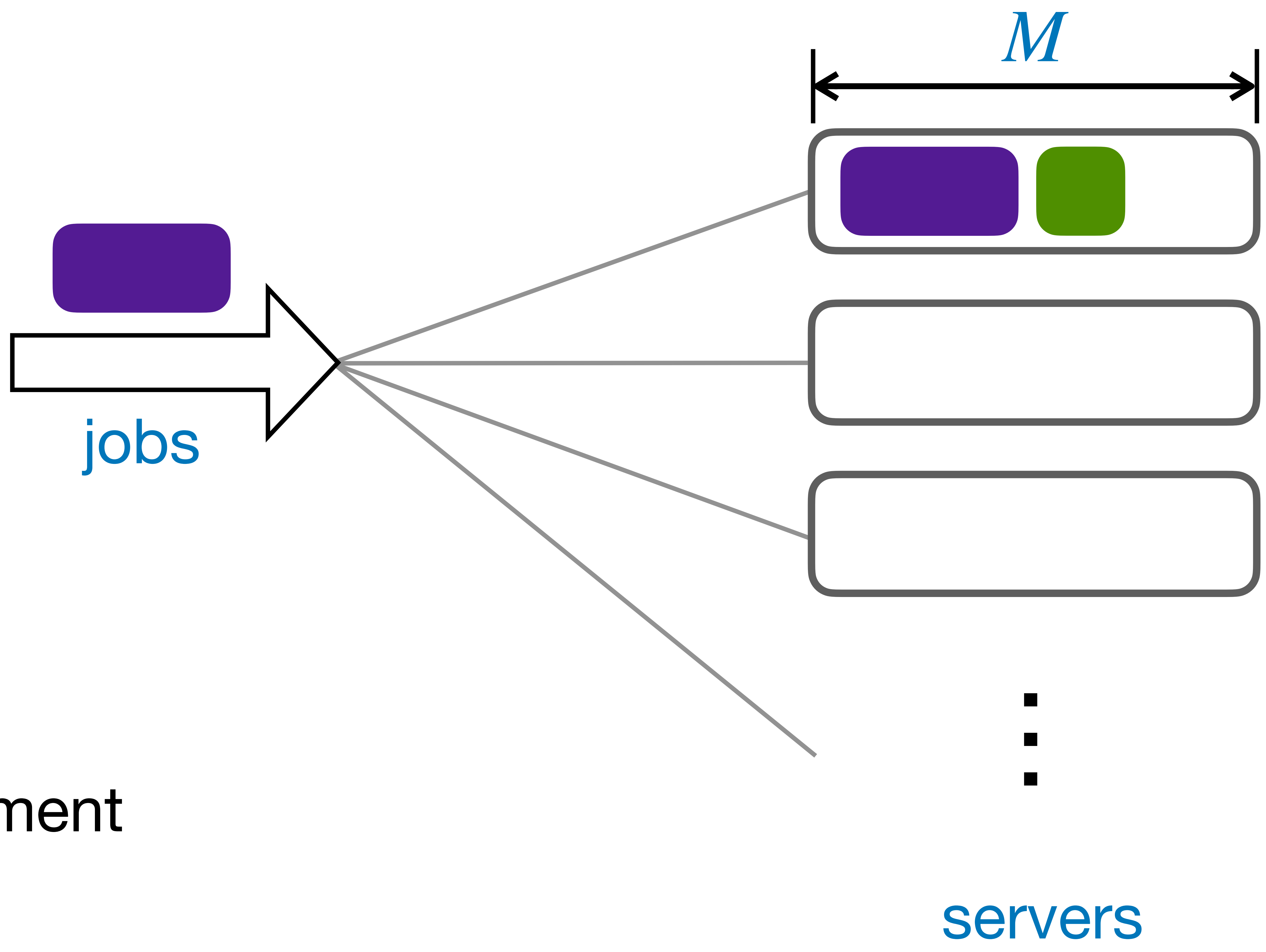


Traditional job model:

- Each job has a fixed resource requirement
- Each job departs after a random time

The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M

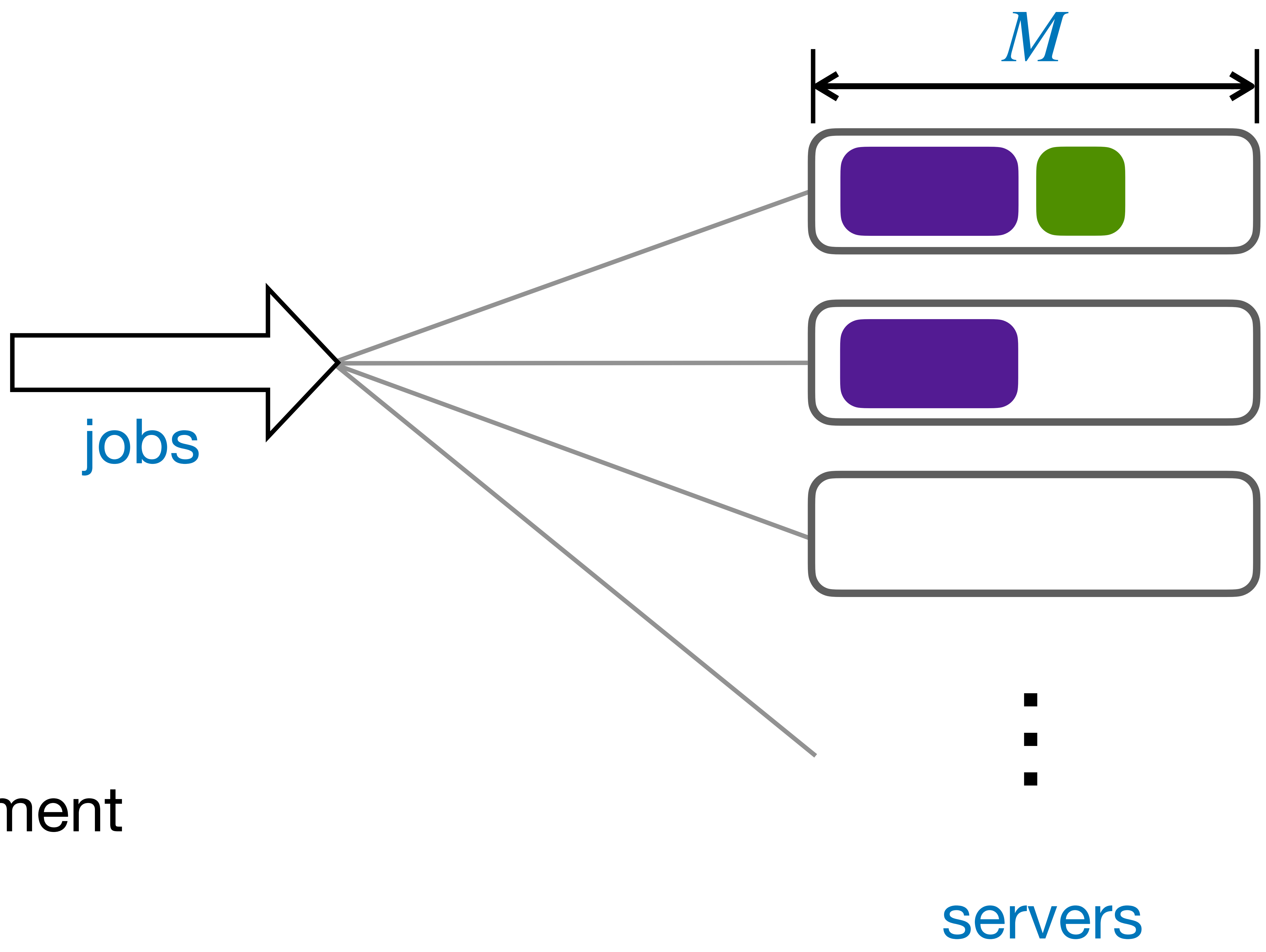


Traditional job model:

- Each job has a fixed resource requirement
- Each job departs after a random time

The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M

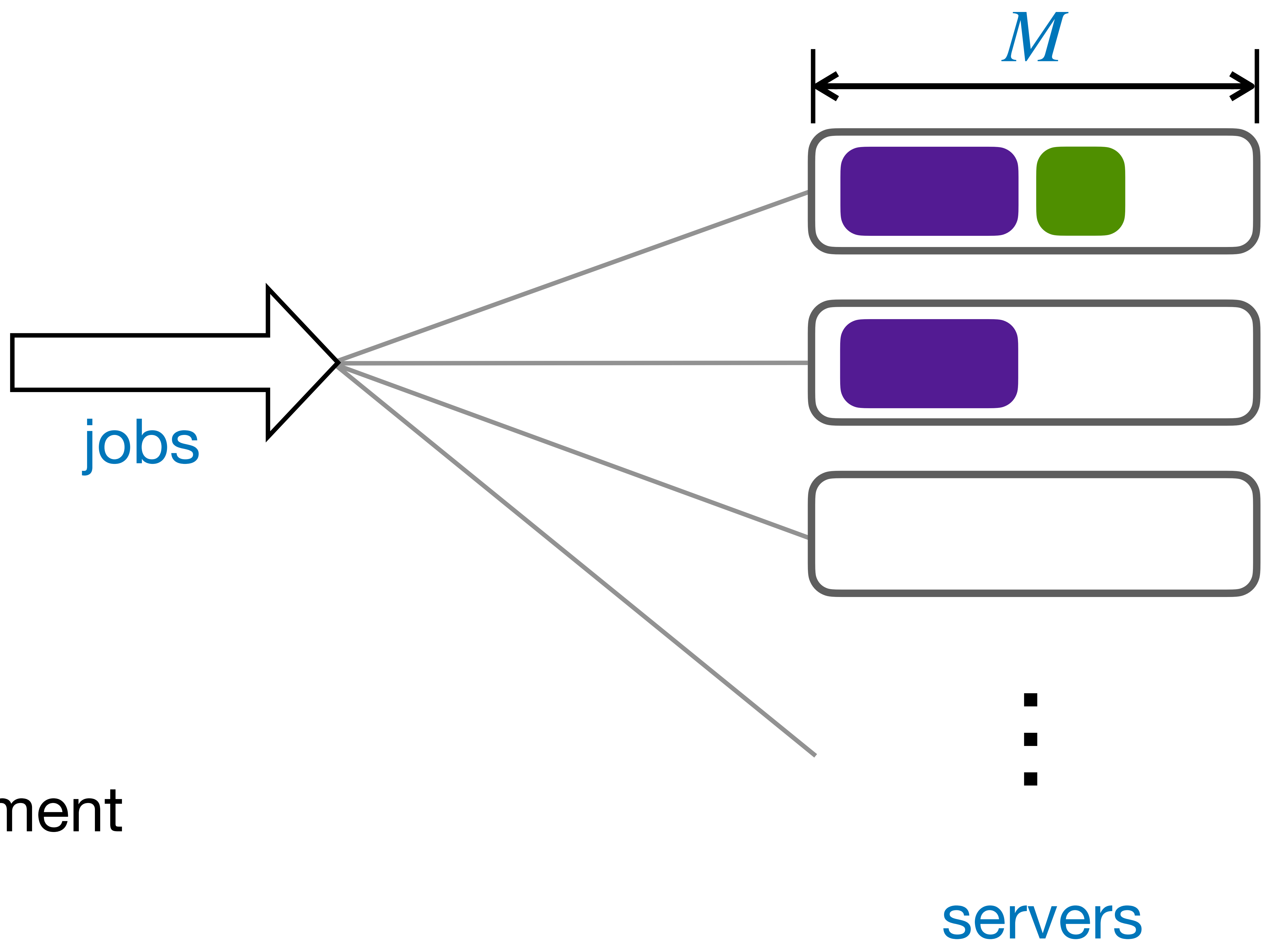


Traditional job model:

- Each job has a fixed resource requirement
- Each job departs after a random time

The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M



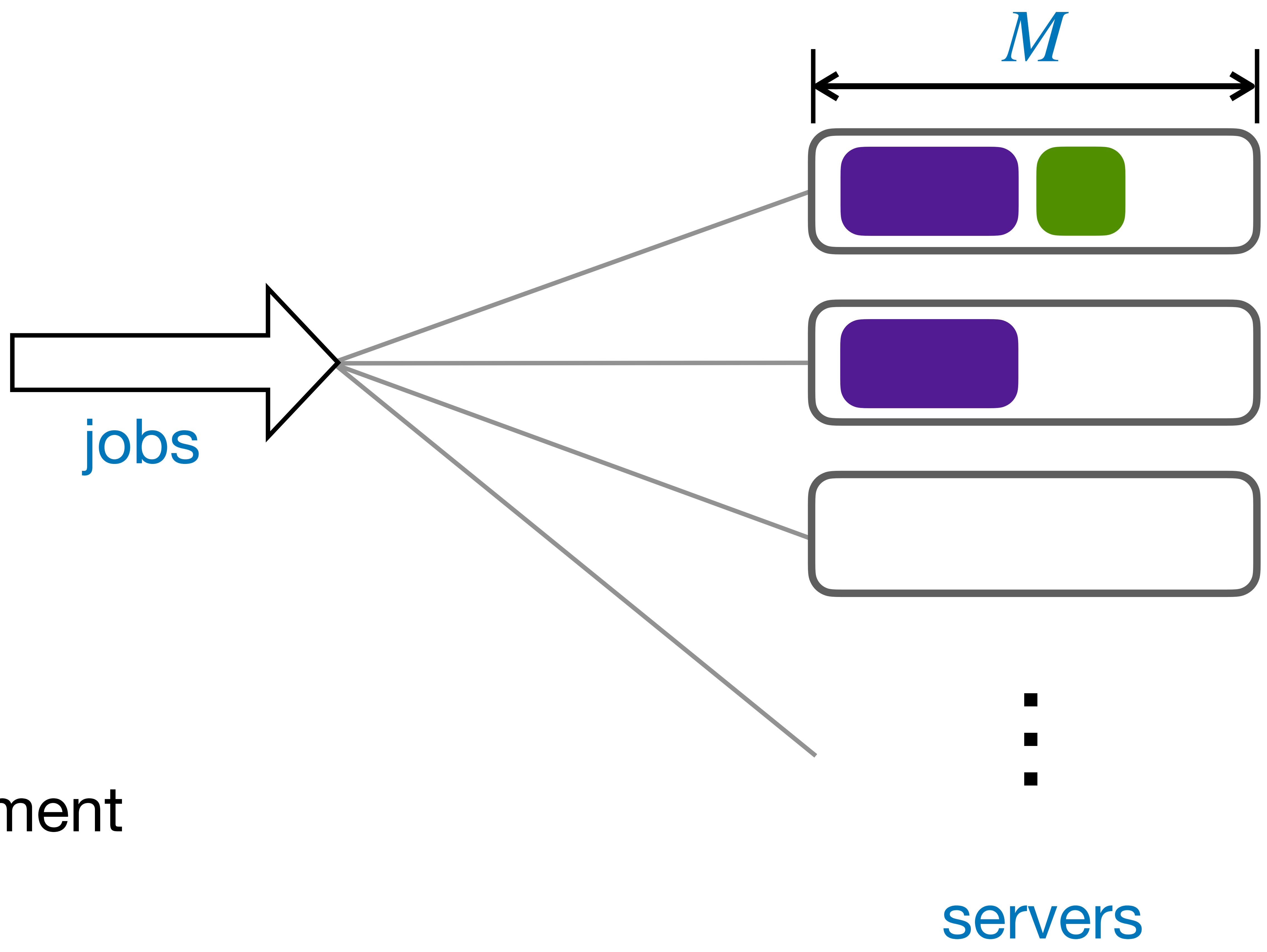
Traditional job model:

- Each job has a fixed resource requirement
- Each job departs after a random time

Goal: $\underset{\text{job assigning policy}}{\text{minimize}}$ **E** [# active servers]

The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M



Traditional job model:

- Each job has a fixed resource requirement
- Each job departs after a random time

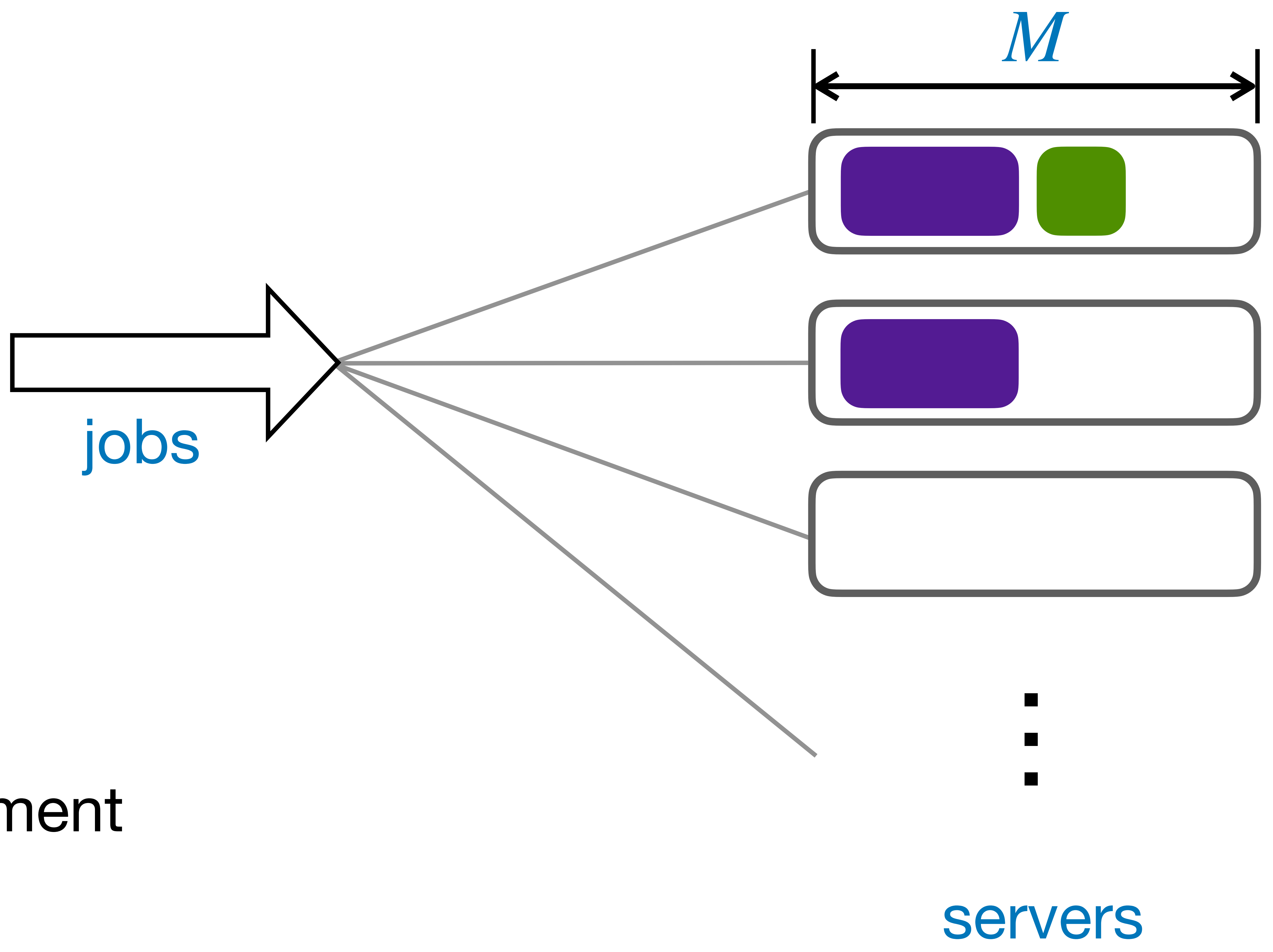
Goal: $\min_{\text{job assigning policy}}$ **E** [# active servers]

Prior work: algorithms with asymptotic optimality

[Stolyar and Zhong 2013, 2015], [Stolyar 2017], [Stolyar and Zhong 2021], …

The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M



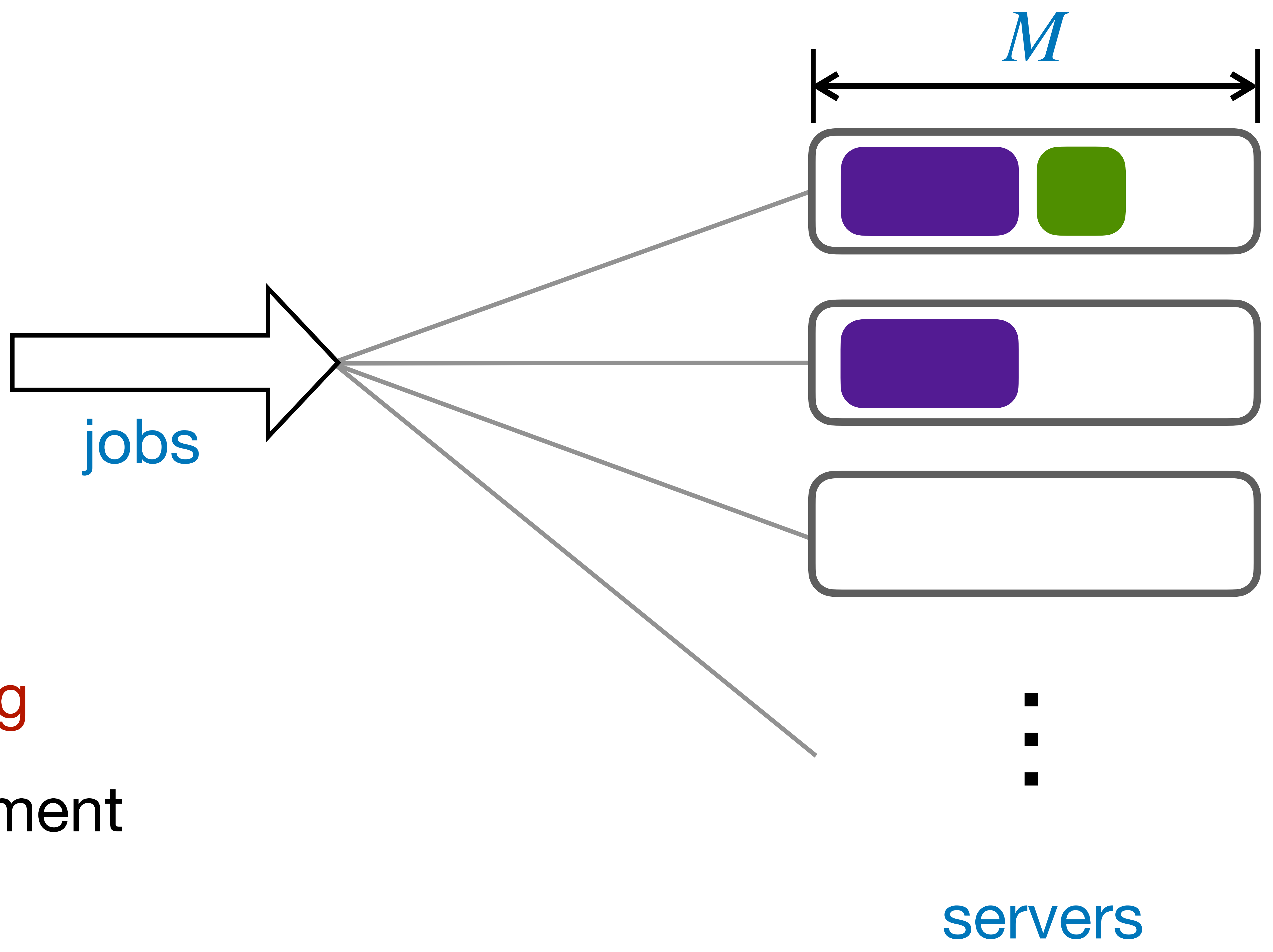
A new job model:

- Each job has a fixed resource requirement
- Each job departs after a random time

Goal: minimize E [# active servers]
job assigning policy

The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M



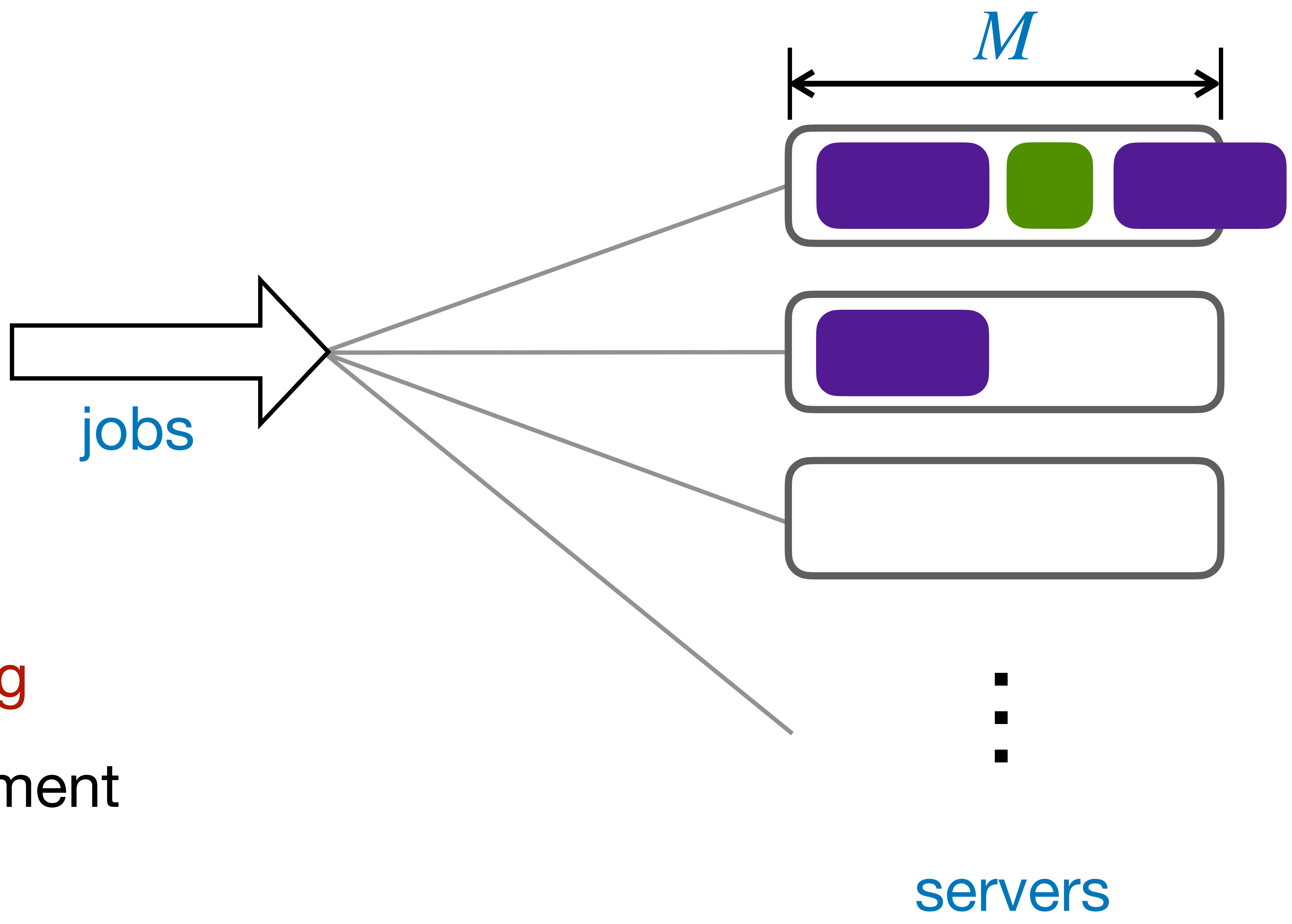
A new job model:

- Each job has a ~~fixed~~ time-varying resource requirement
- Each job departs after a random time

Goal: minimize (job assigning policy) **E** [# active servers]

The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M



A new job model:

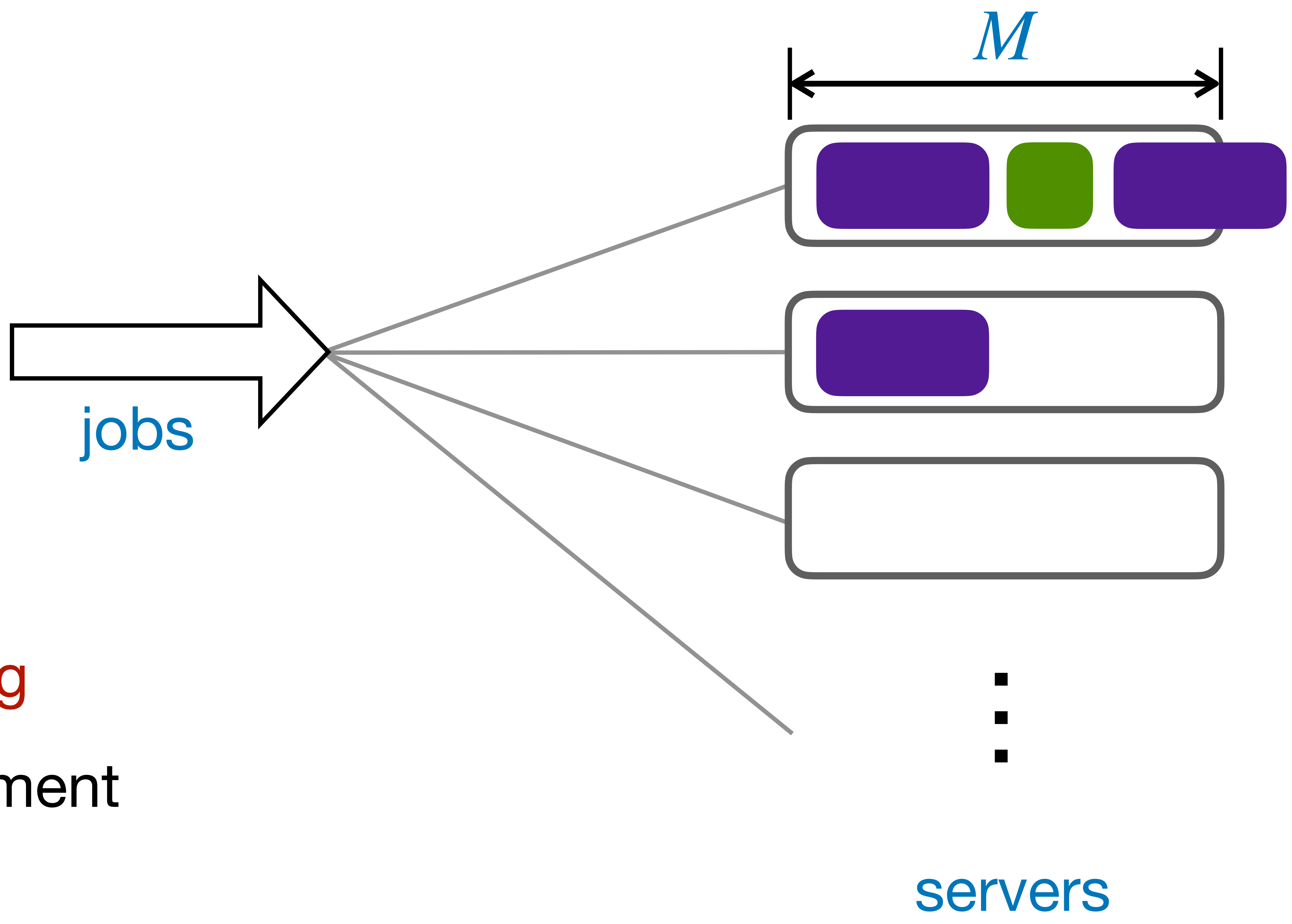
time-varying

- Each job has a ~~fixed~~ resource requirement
- Each job departs after a random time

Goal: minimize (job assigning policy) **E** [# active servers]

The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity M



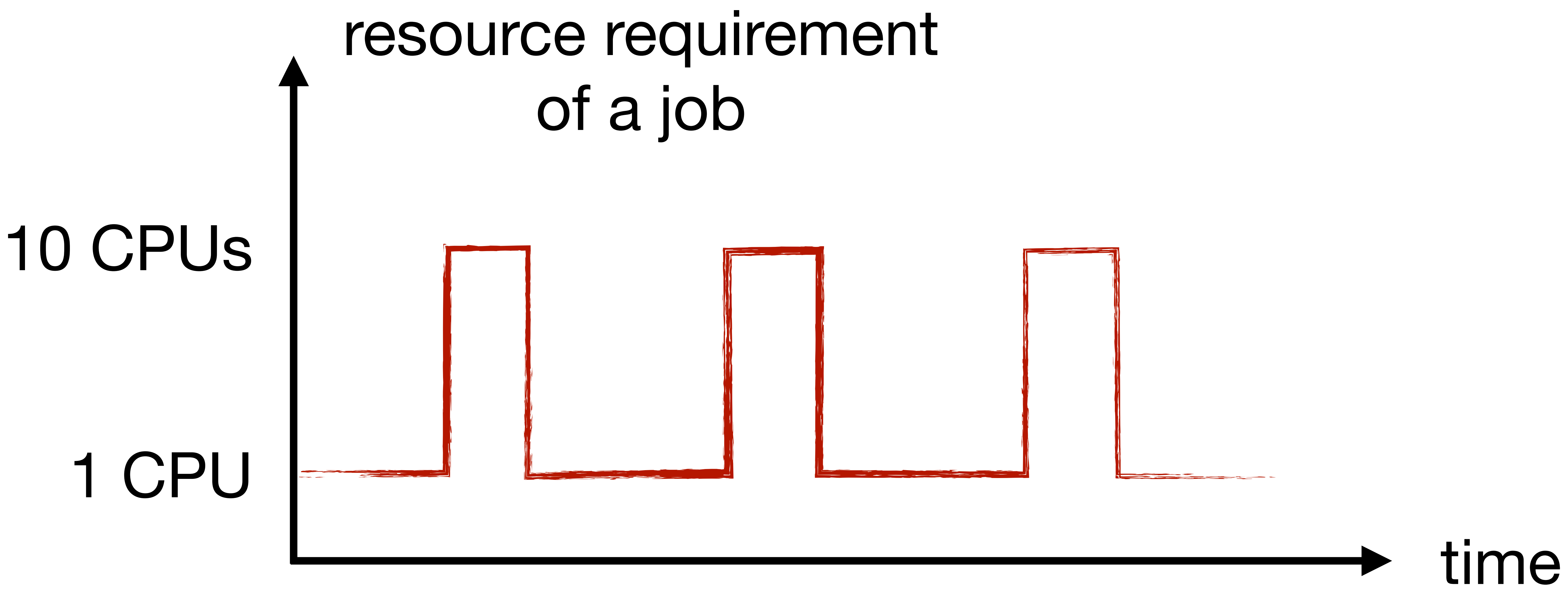
A new job model: time-varying

- Each job has a ~~fixed~~ resource requirement
- Each job departs after a random time

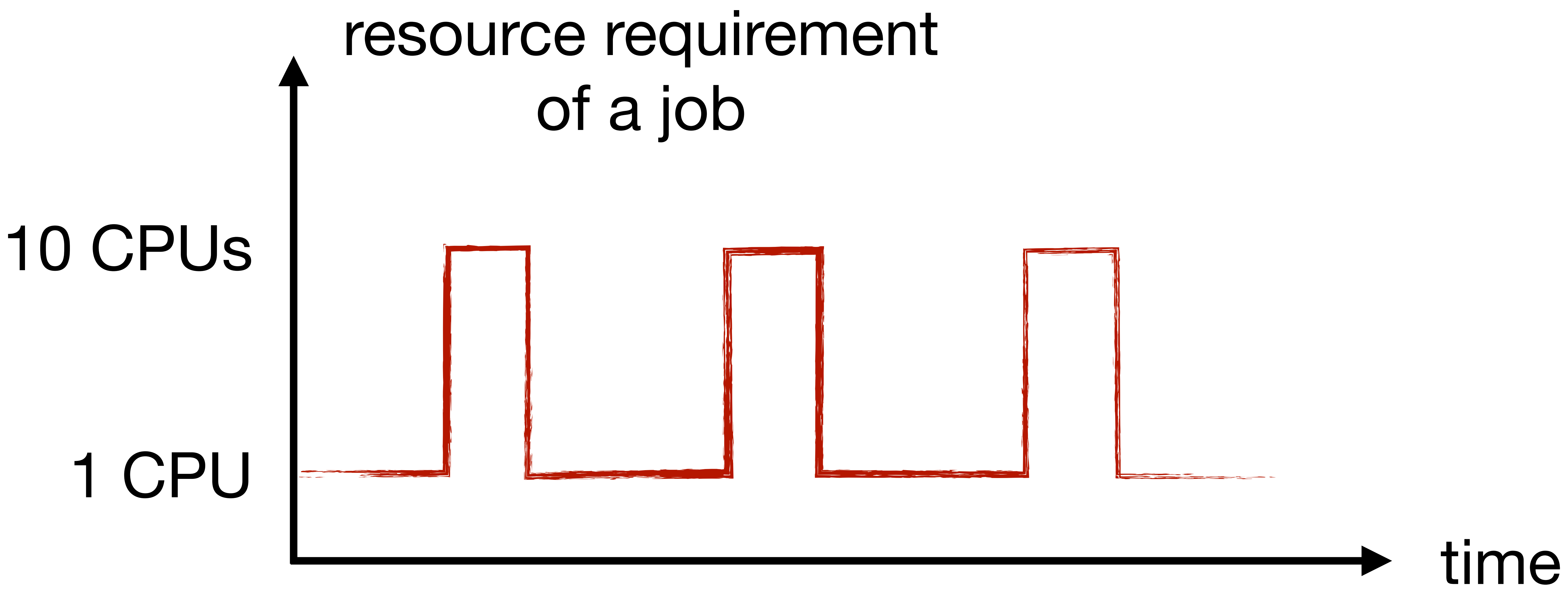
Goal: minimize (job assigning policy) **E** [# active servers]

subject to **cost** (resource contention) ≤ budget

Why does time-varying matter?

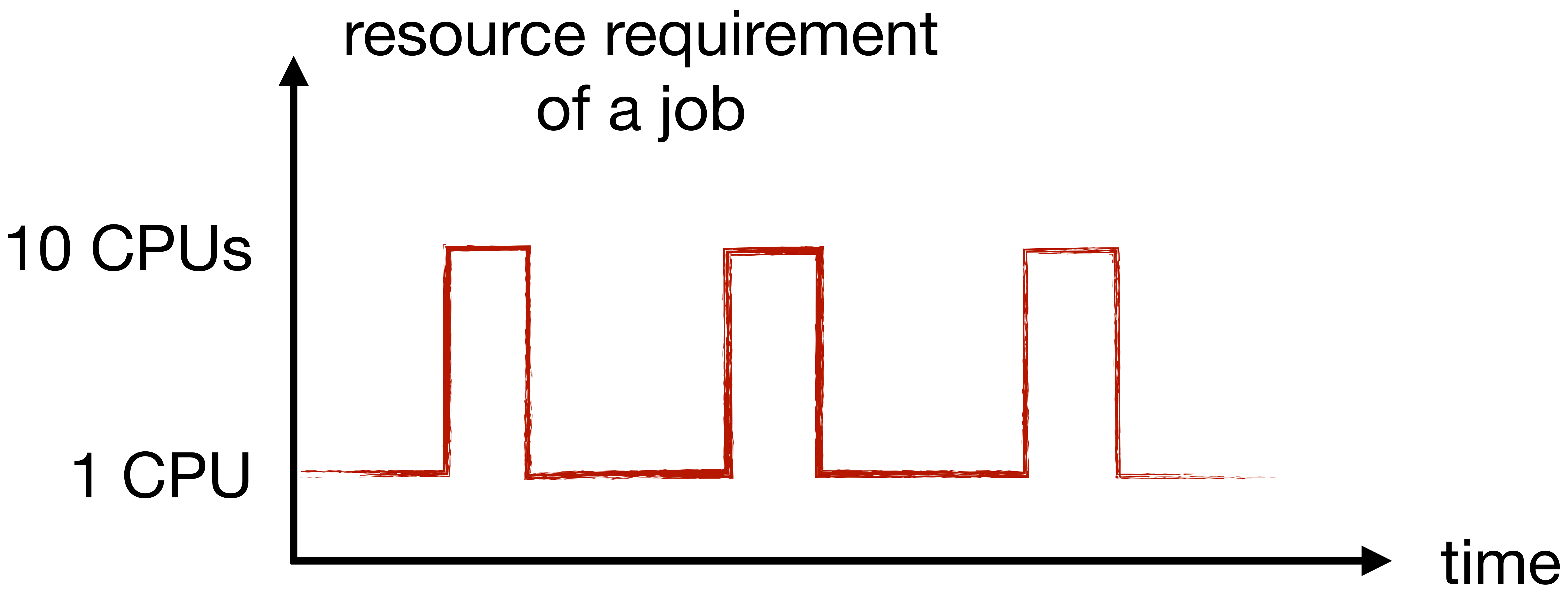


Why does time-varying matter?



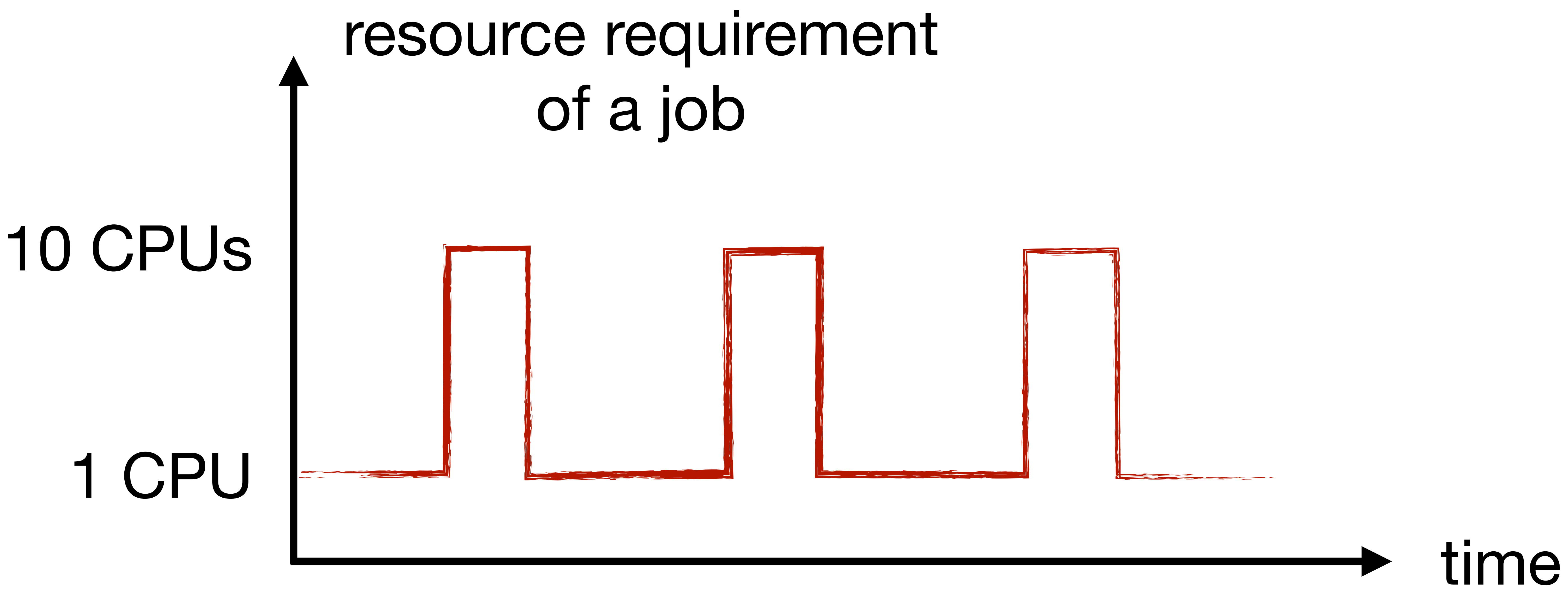
- Reserve resources based on peak requirement

Why does time-varying matter?



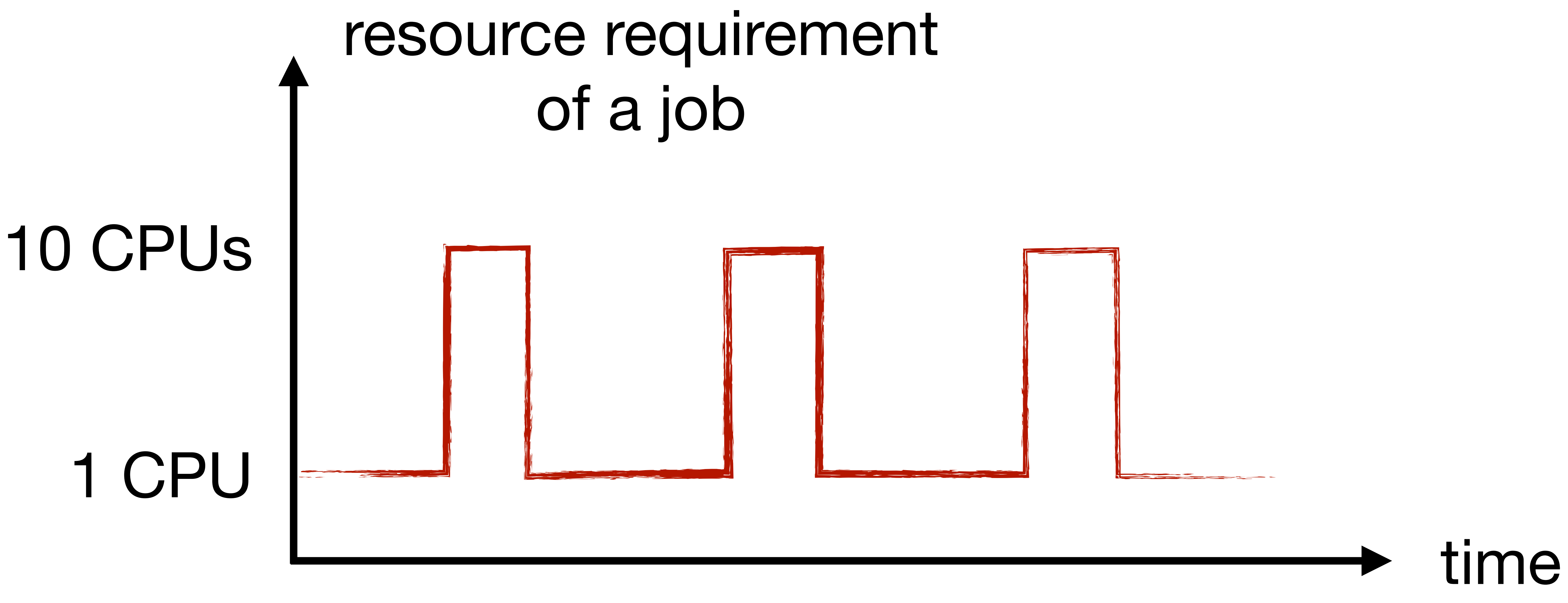
- Reserve resources based on peak requirement
  ➡ low resource utilization on a server

Why does time-varying matter?



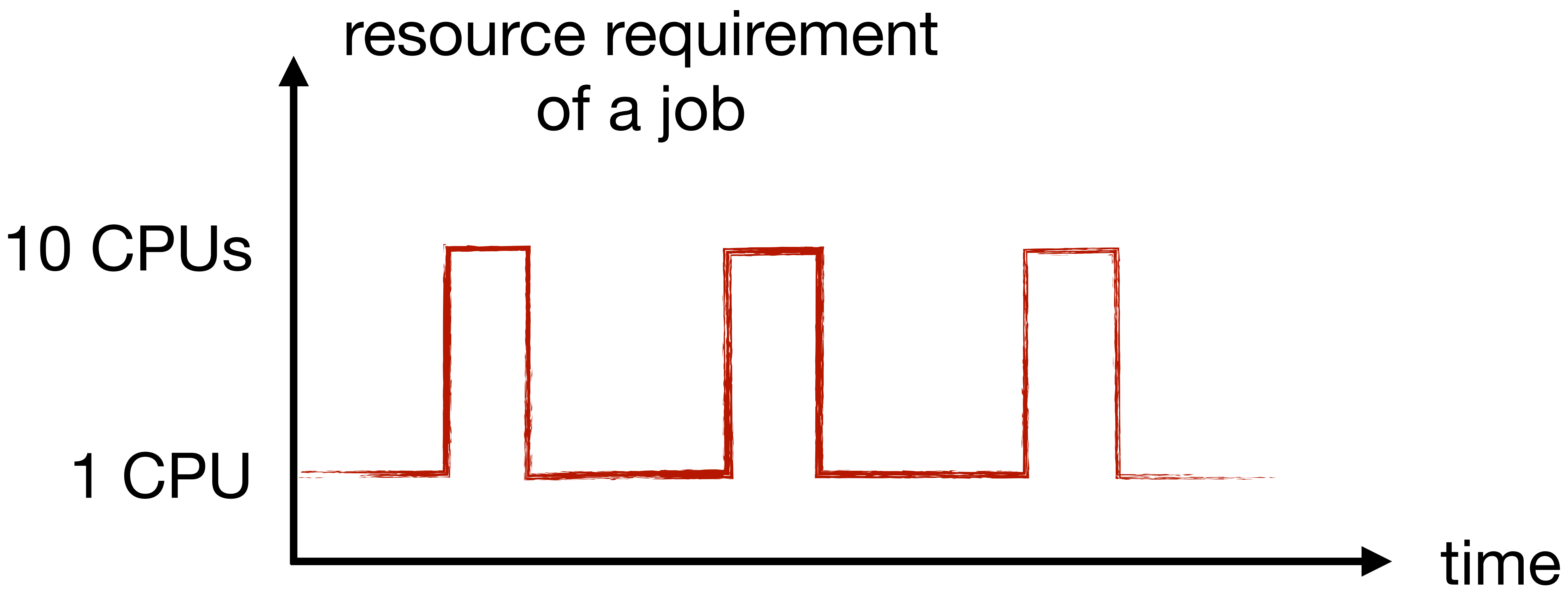
- Reserve resources based on peak requirement
  - ➡ low resource utilization on a server
  - ➡ larger # active servers

Why does time-varying matter?



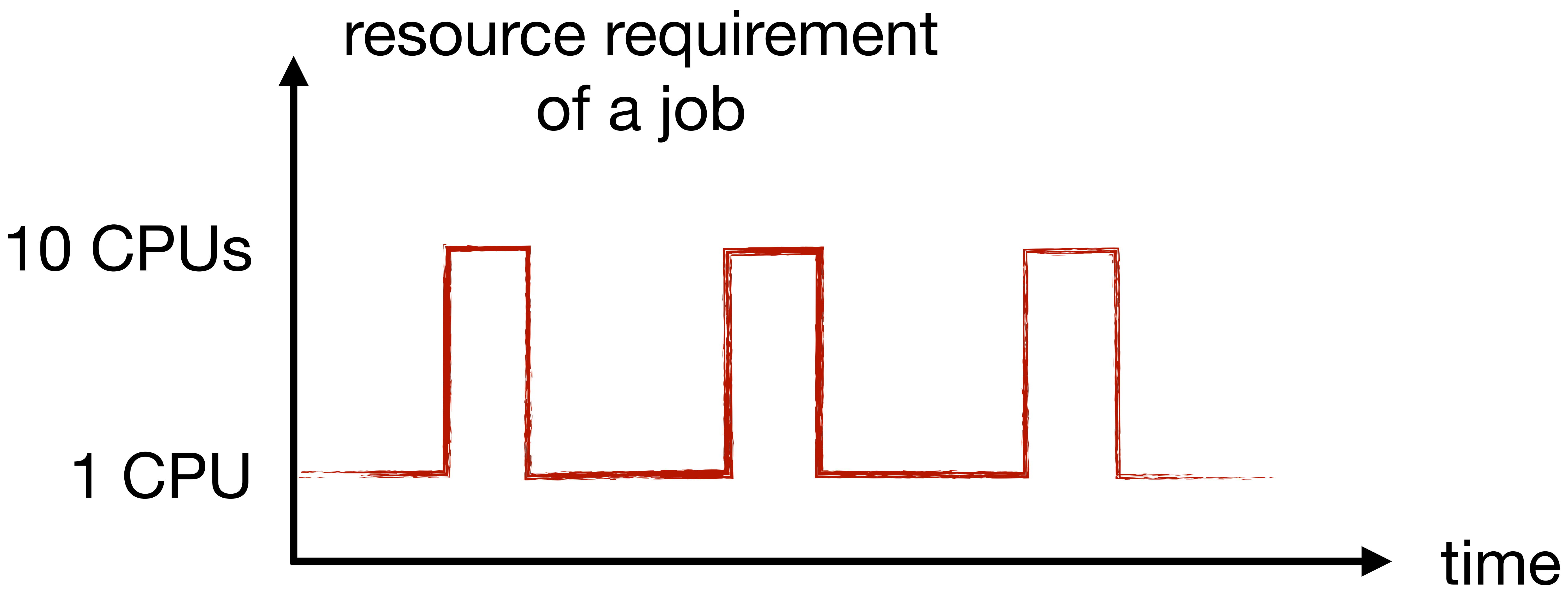
- Reserve resources based on peak requirement
  - ➡ low resource utilization on a server
  - ➡ larger # active servers
- Overcommit resources on a server

Why does time-varying matter?



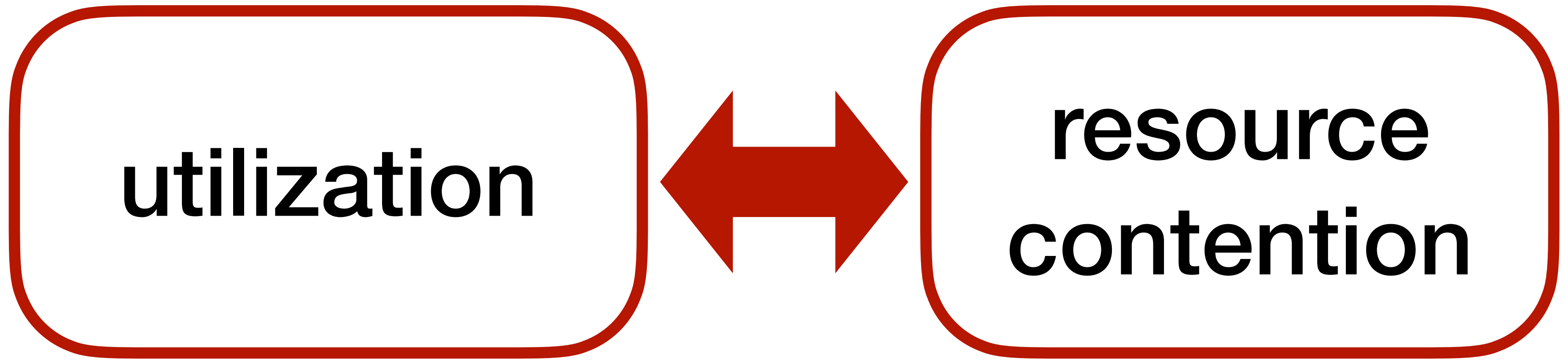
- Reserve resources based on peak requirement
  - ➡ low resource utilization on a server
  - ➡ larger # active servers
- Overcommit resources on a server
  - ➡ possible resource contention

Why does time-varying matter?

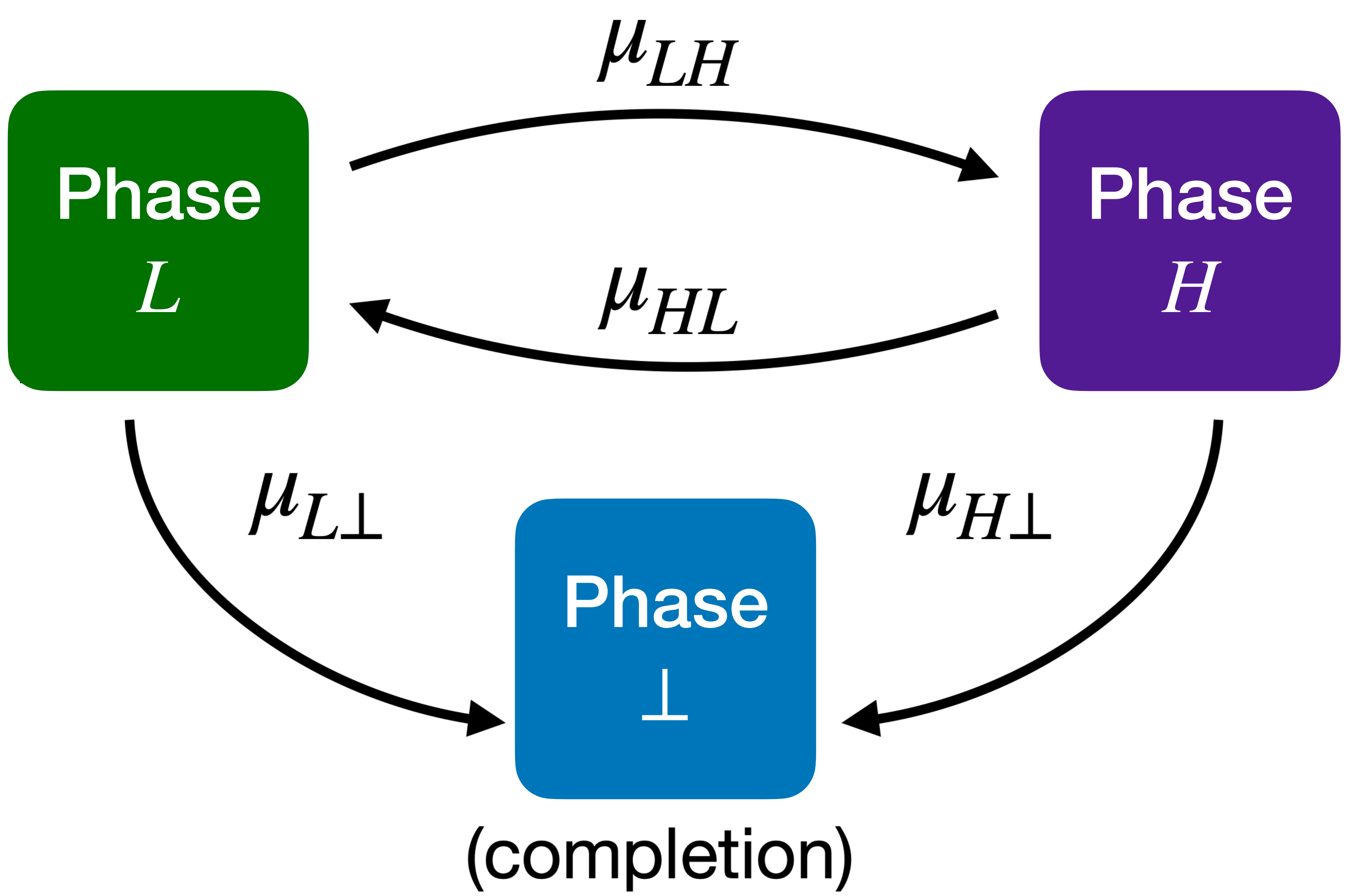


- Reserve resources based on peak requirement
  - ➡ low resource utilization on a server
  - ➡ larger # active servers
- Overcommit resources on a server
  - ➡ possible resource contention

Our formulation captures:



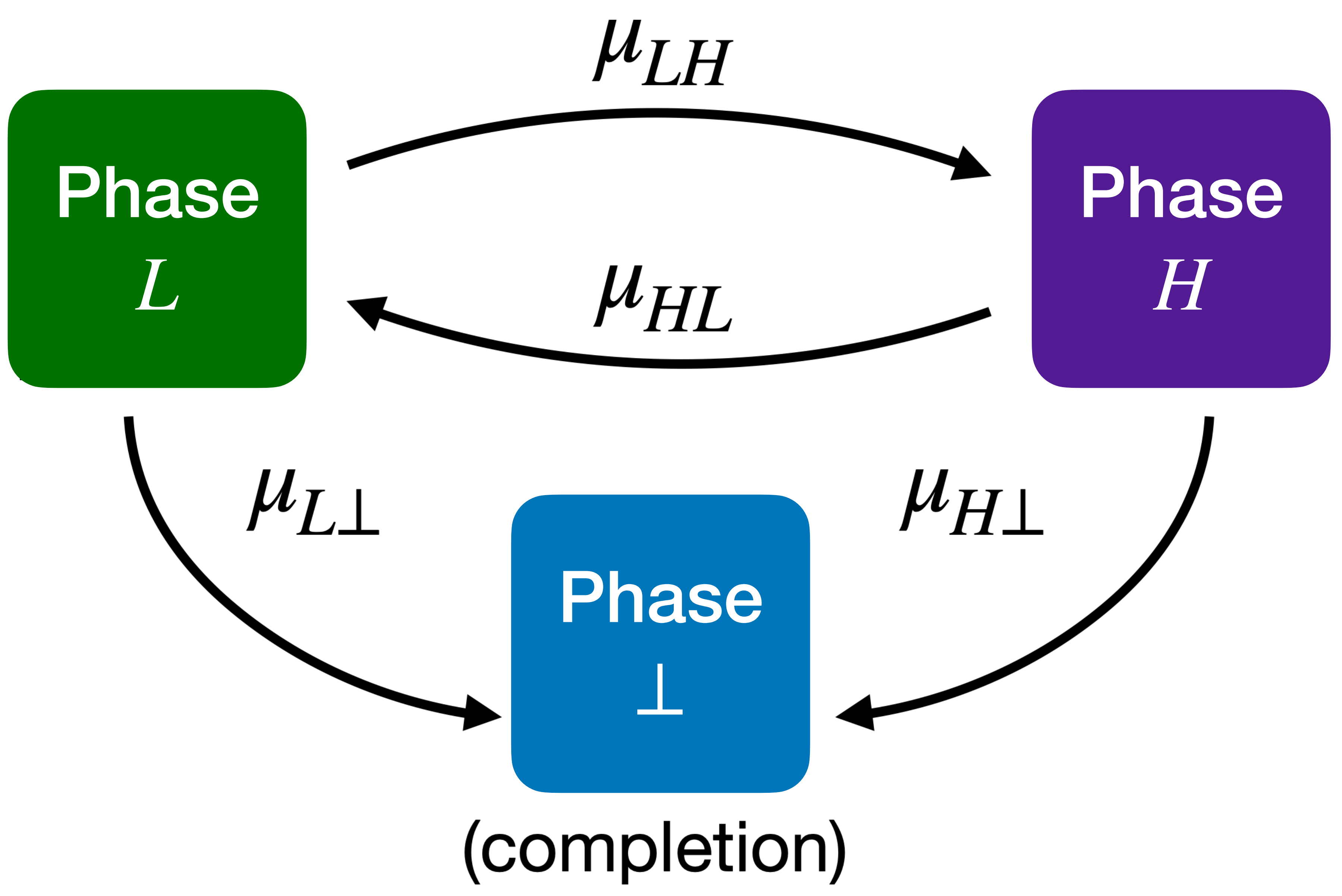
More details on the job model



Example MC

More details on the job model

- Resource requirement of a job evolves over time following a Markov chain

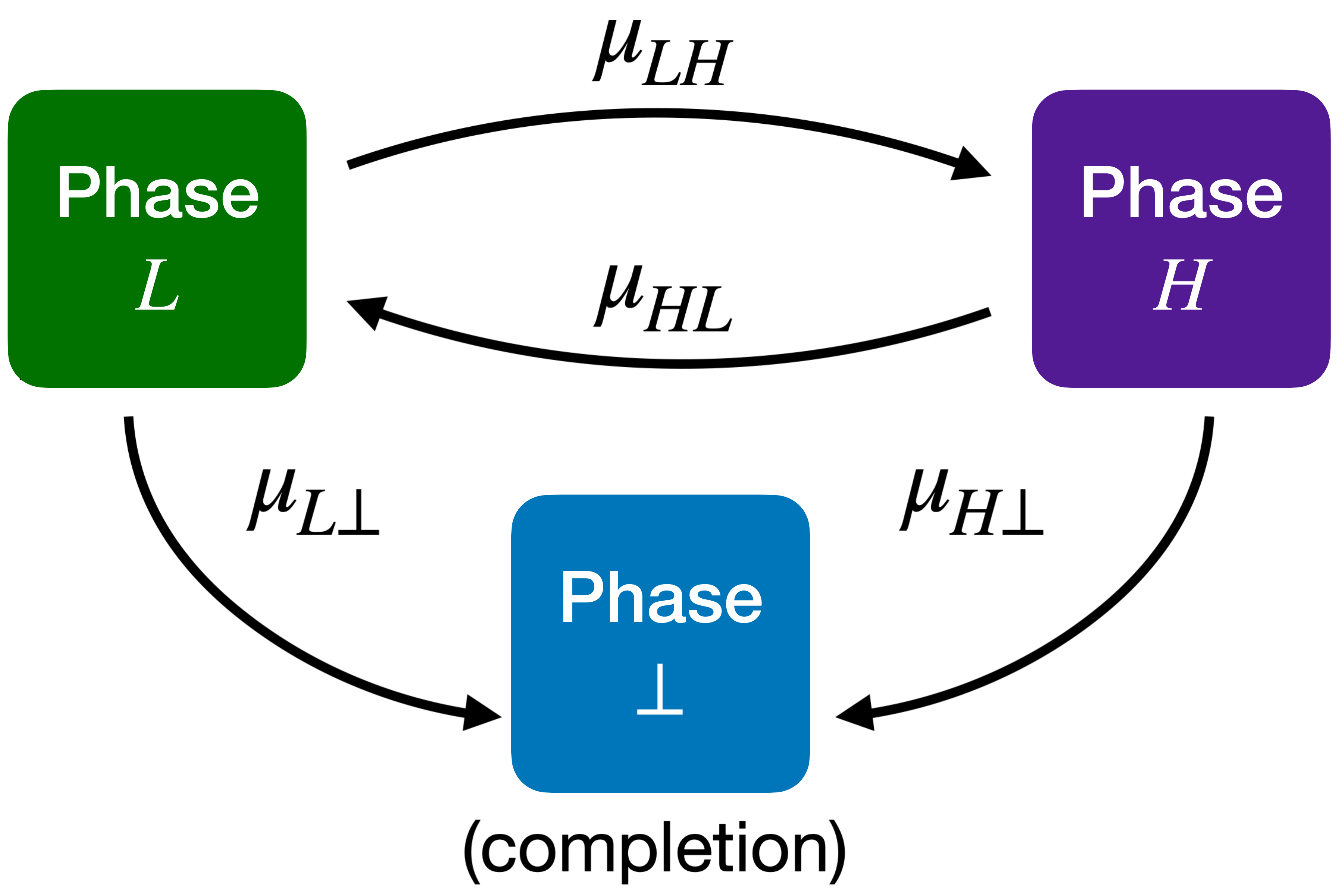


Example MC

More details on the job model

- Resource requirement of a job evolves over time following a Markov chain

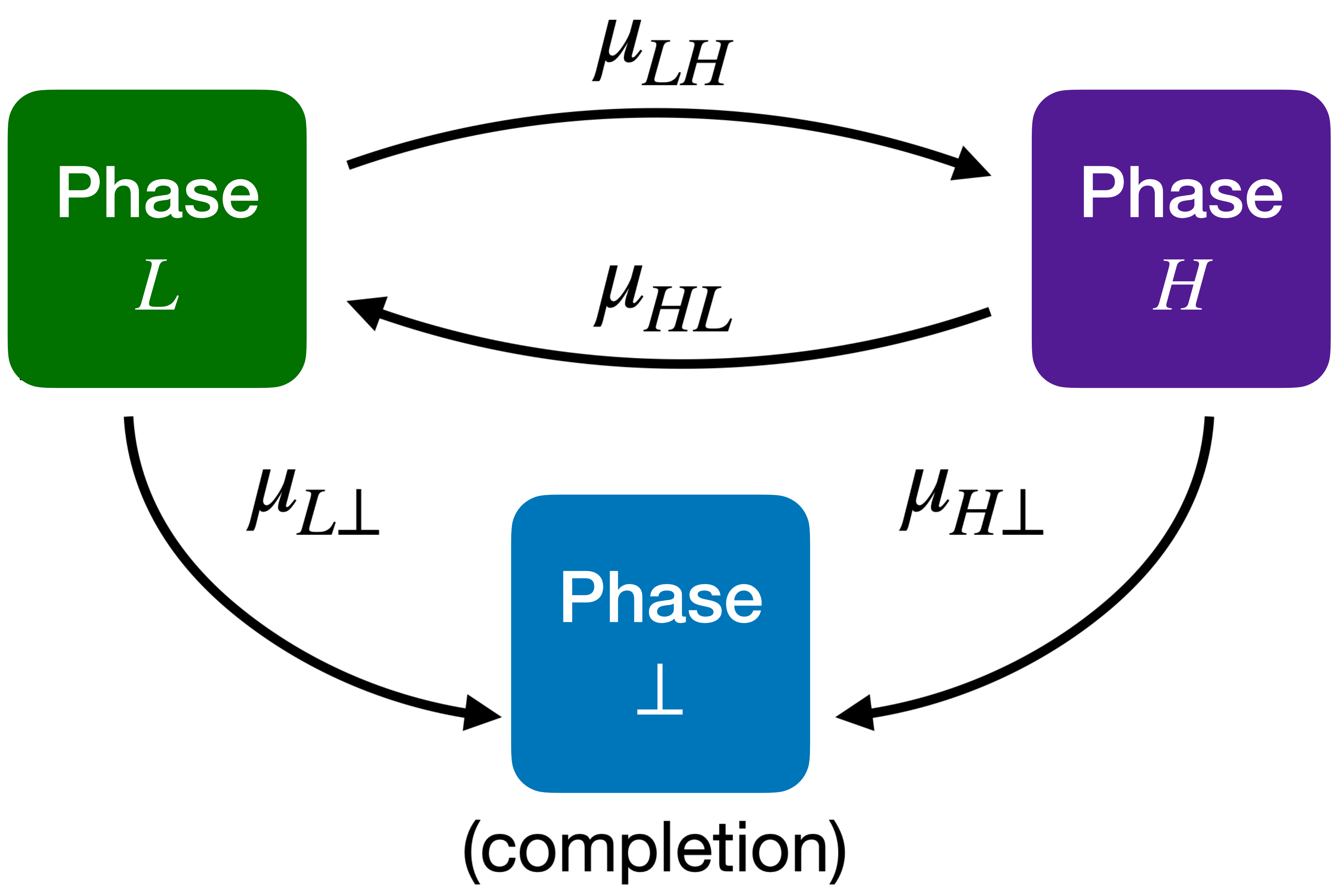
- Initial job type follows an initial distribution



Example MC

More details on the job model

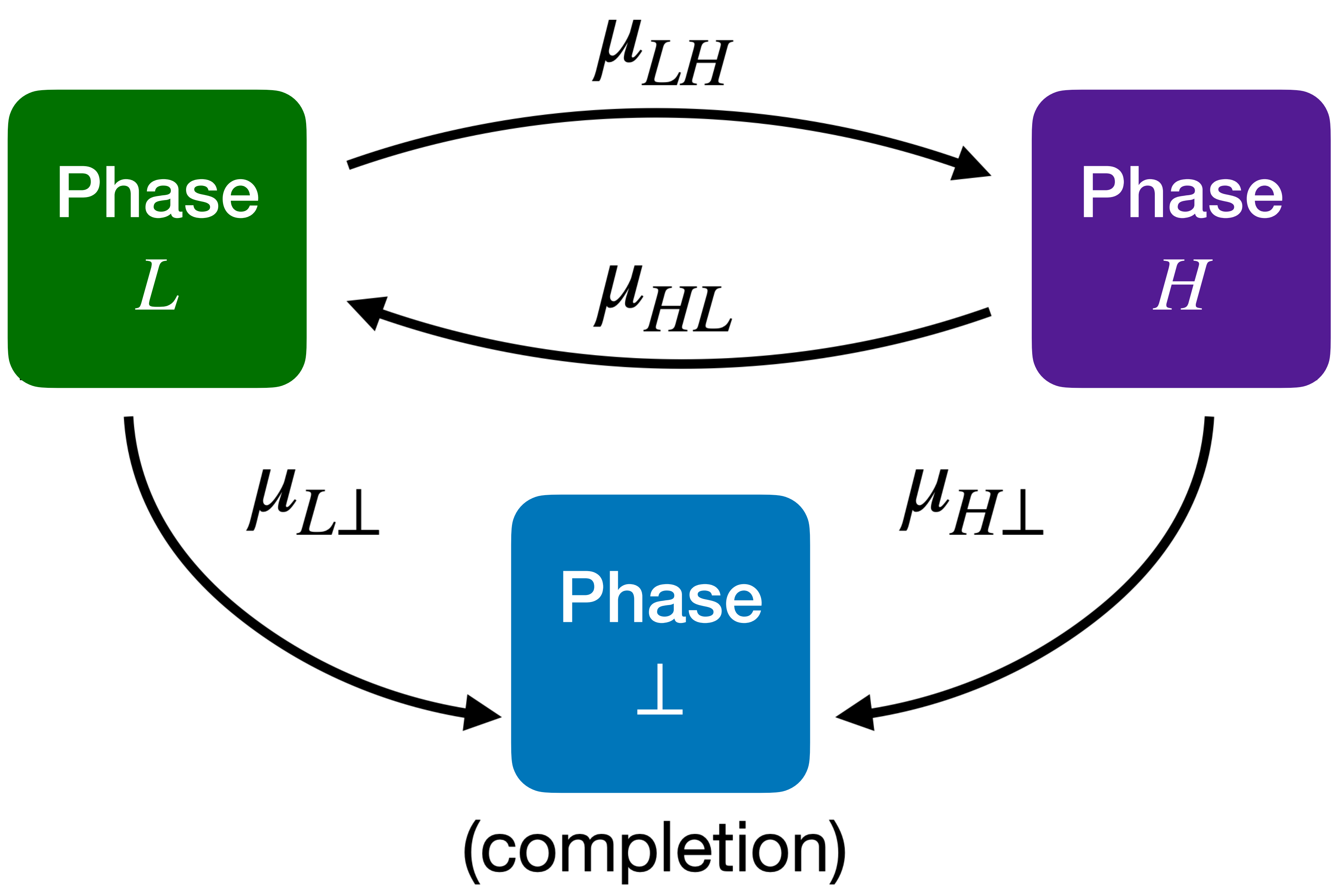
- Resource requirement of a job evolves over time following a Markov chain
- Initial job type follows an initial distribution
- MCs of jobs are independent of each other, and they are exogenous (not affected by resource contention)



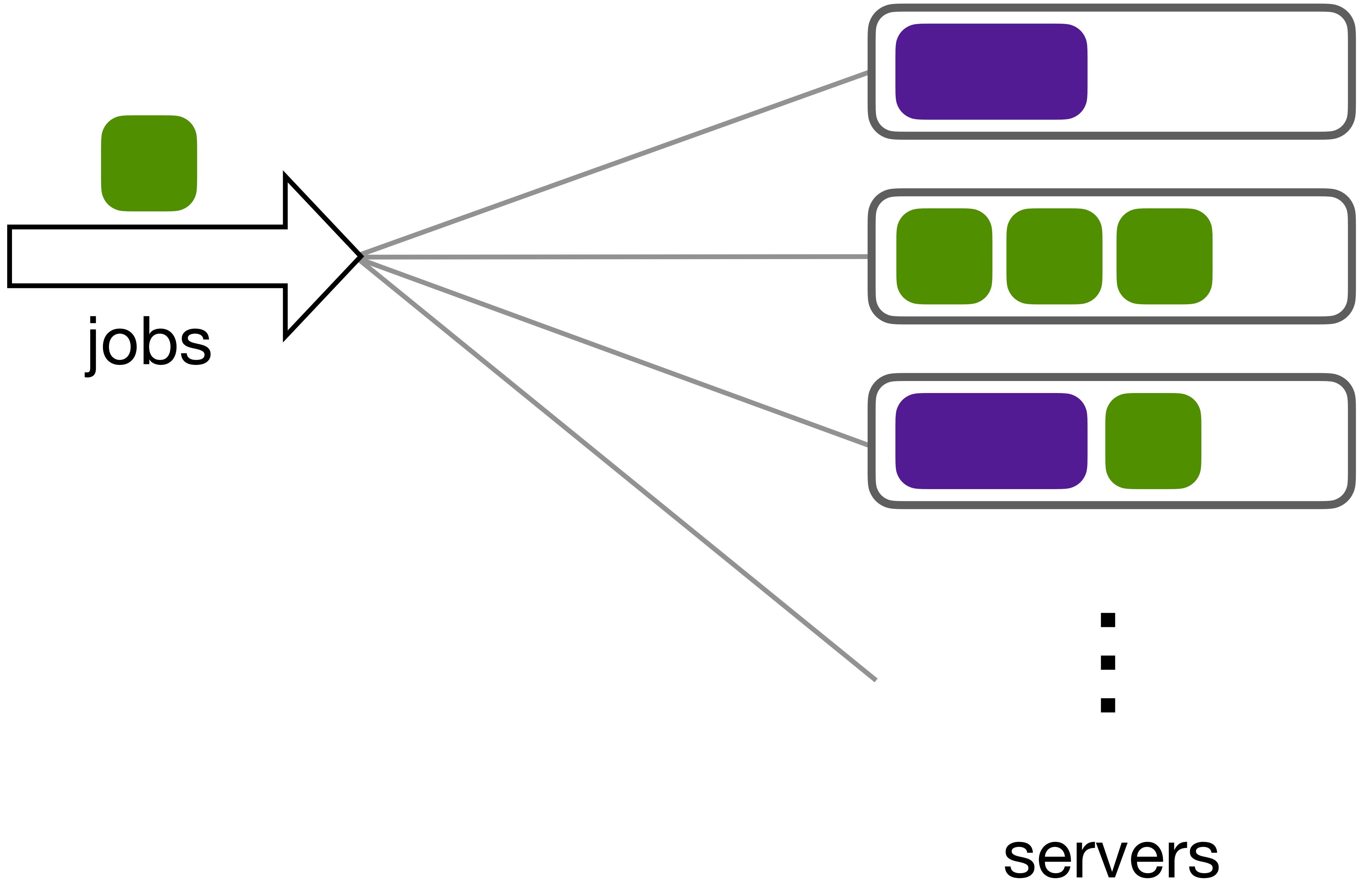
Example MC

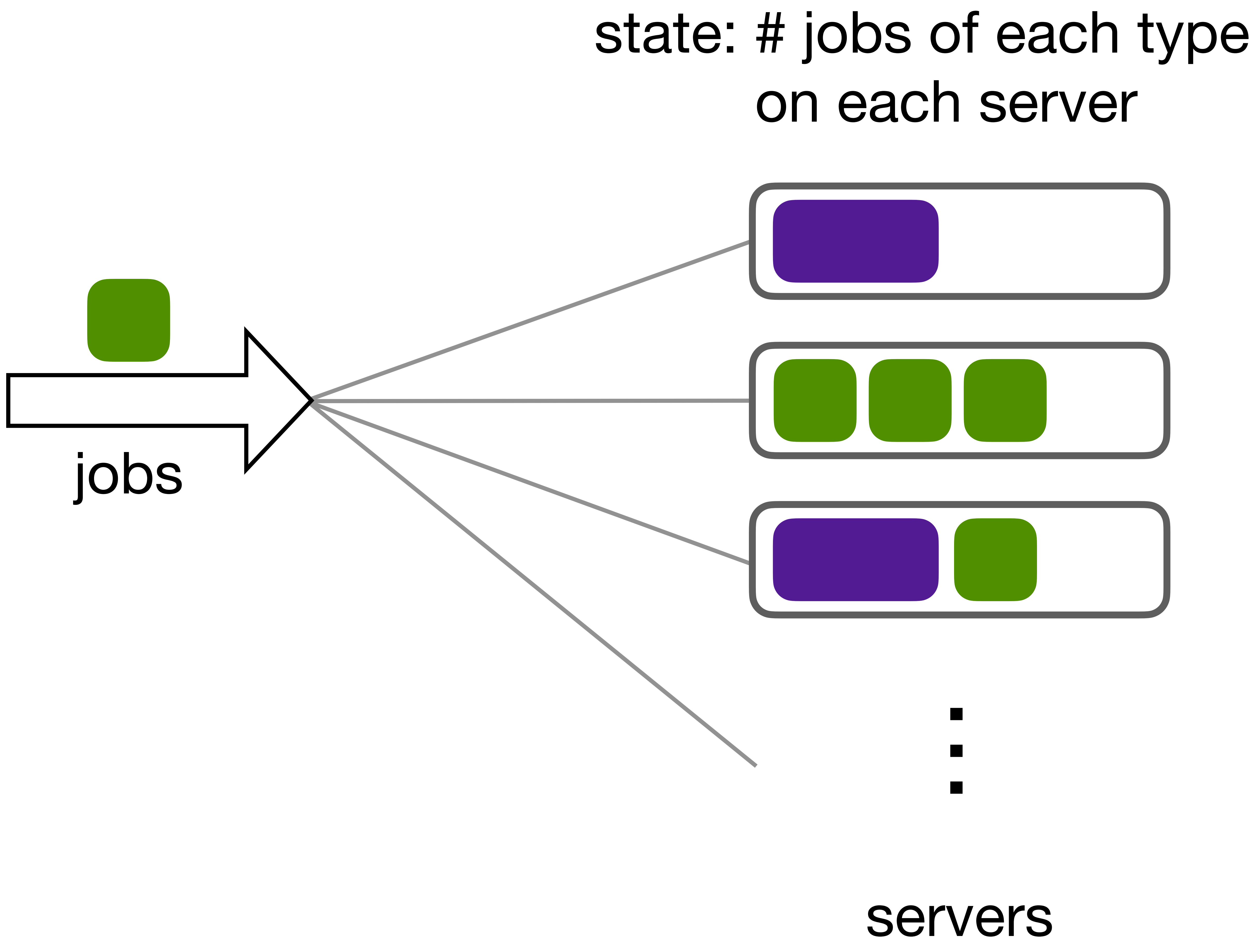
More details on the job model

- Resource requirement of a job evolves over time following a Markov chain
- Initial job type follows an initial distribution
- MCs of jobs are independent of each other, and they are exogenous (not affected by resource contention)
- Jobs arrive following a Poisson process



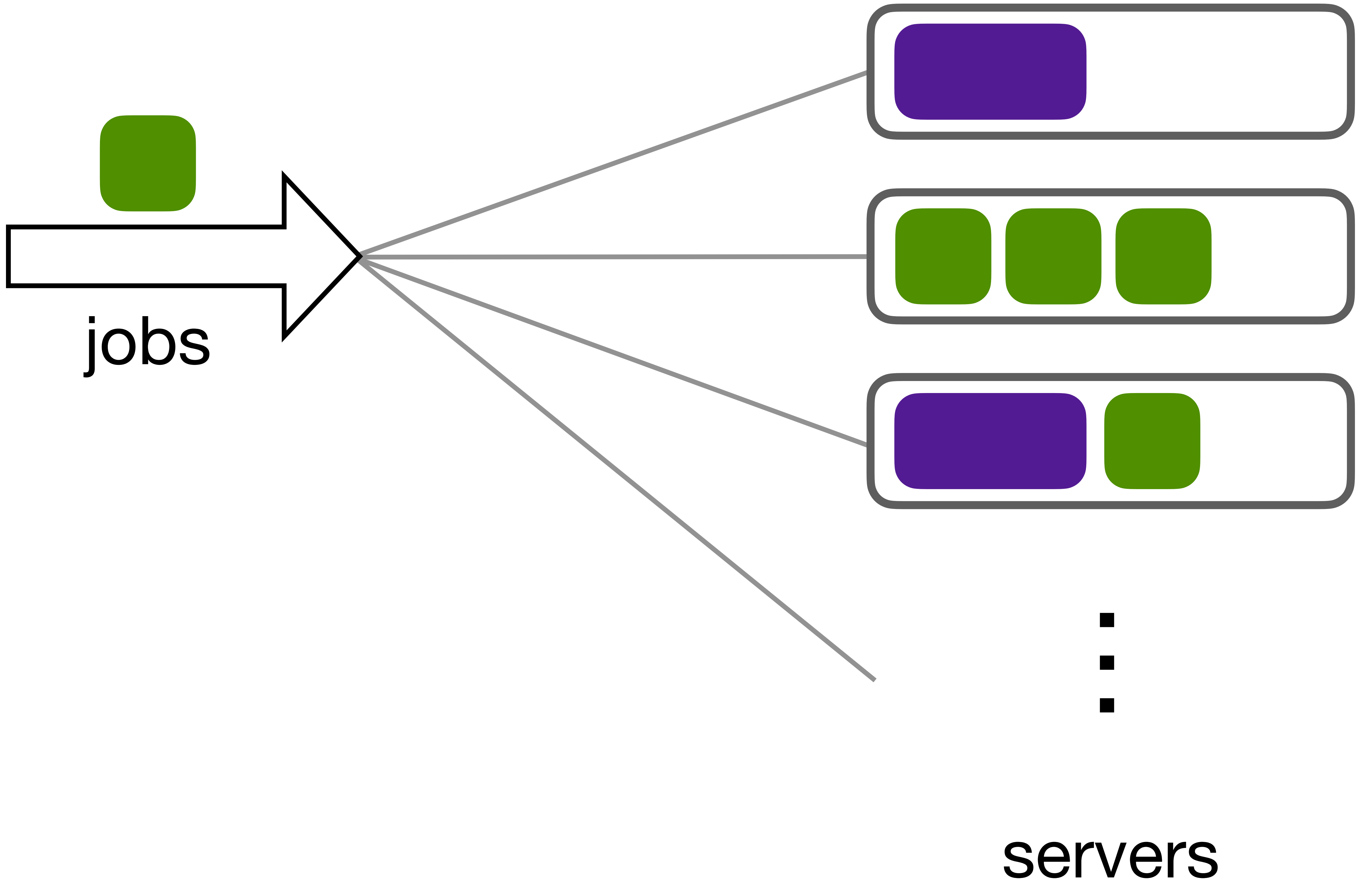
Example MC


jobs
servers


state: # jobs of each type
on each server
jobs
servers

state space is large!

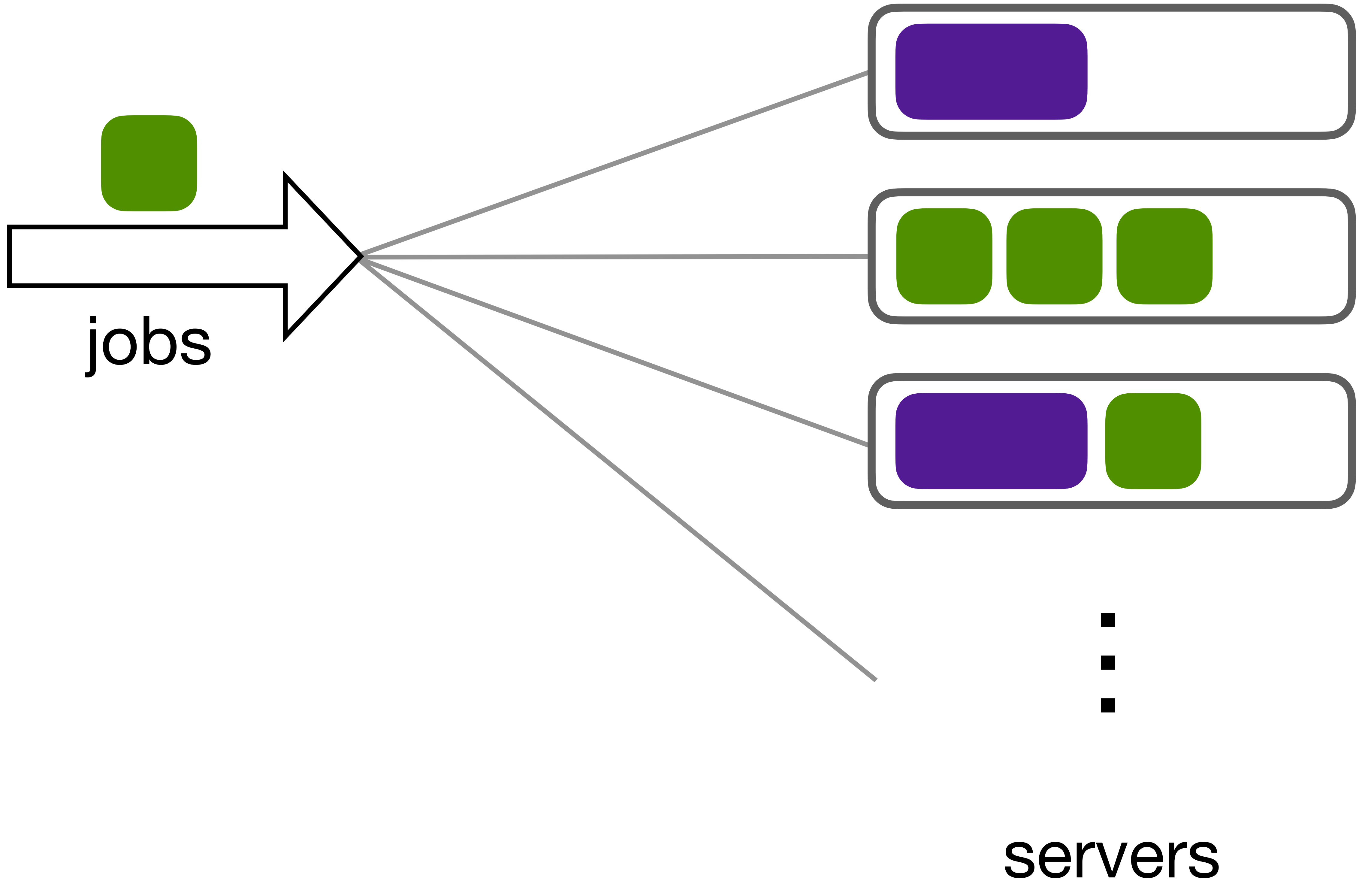
state: # jobs of each type on each server



Reducing dimensionality

state space is large!

state: # jobs of each type on each server

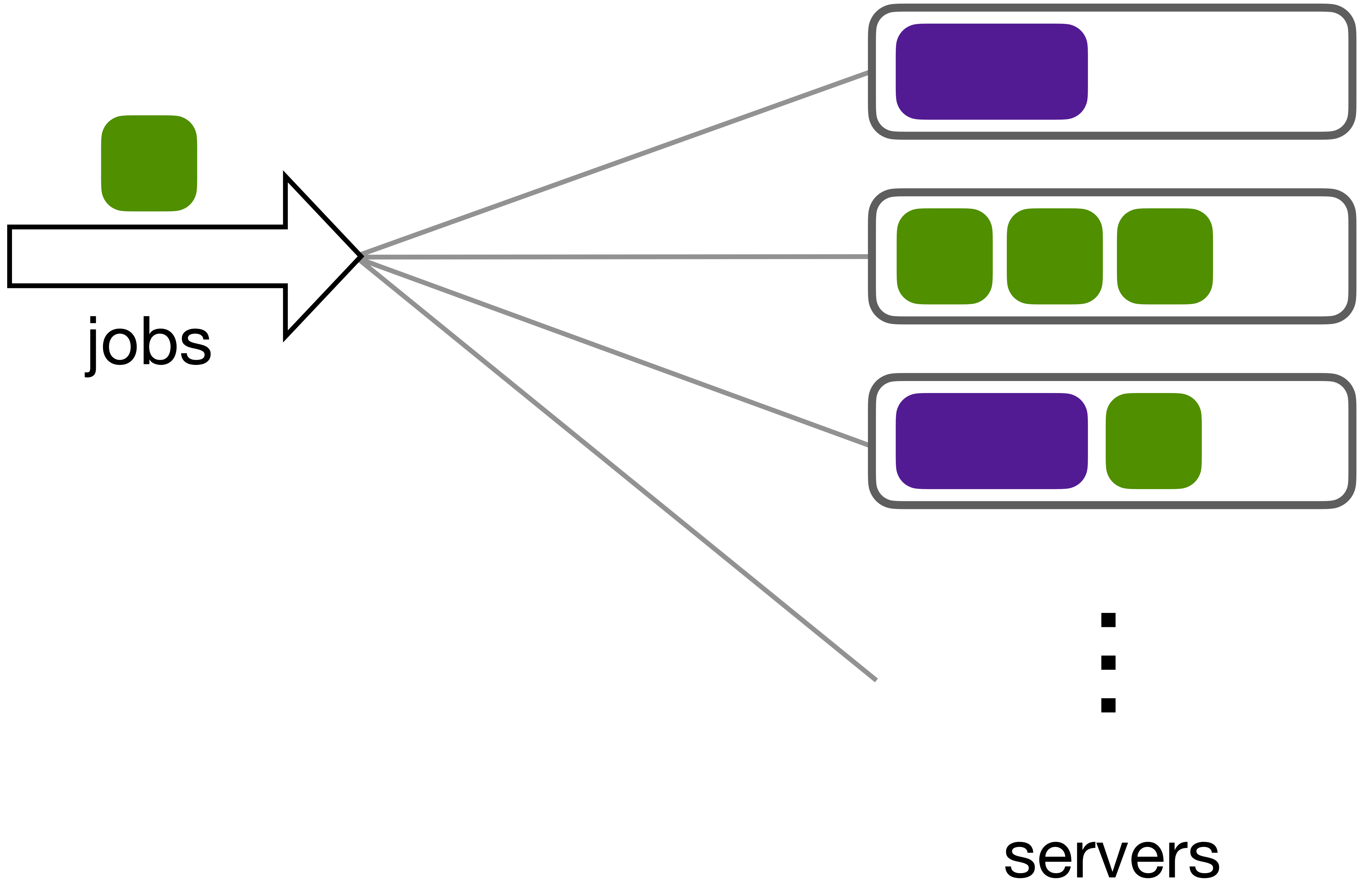


Reducing dimensionality

state space is large!

state: # jobs of each type on each server

- Server-by-server evaluation:

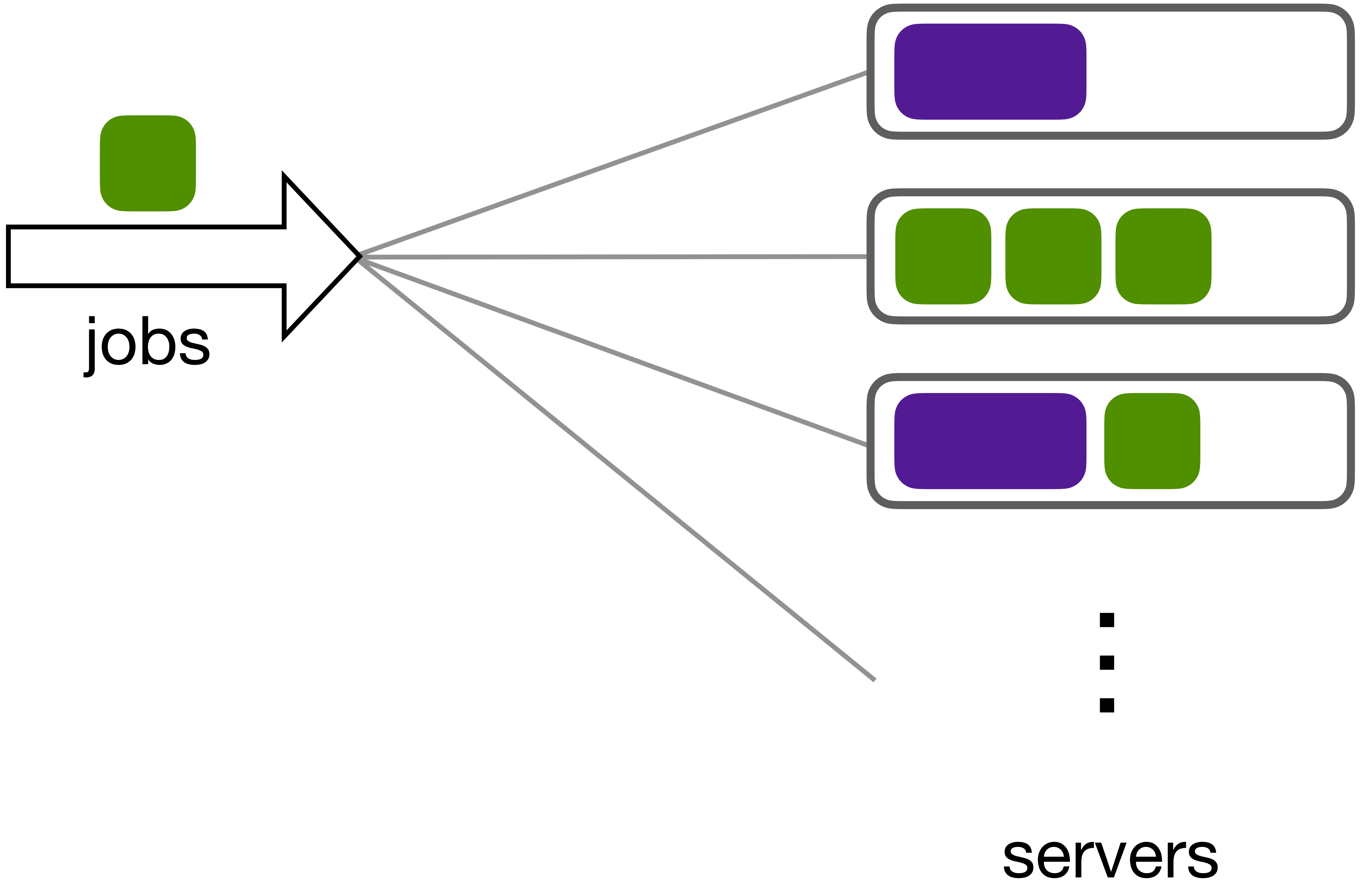


Reducing dimensionality

state space is large!

state: # jobs of each type on each server

- Server-by-server evaluation:
  - How to evaluate each server?

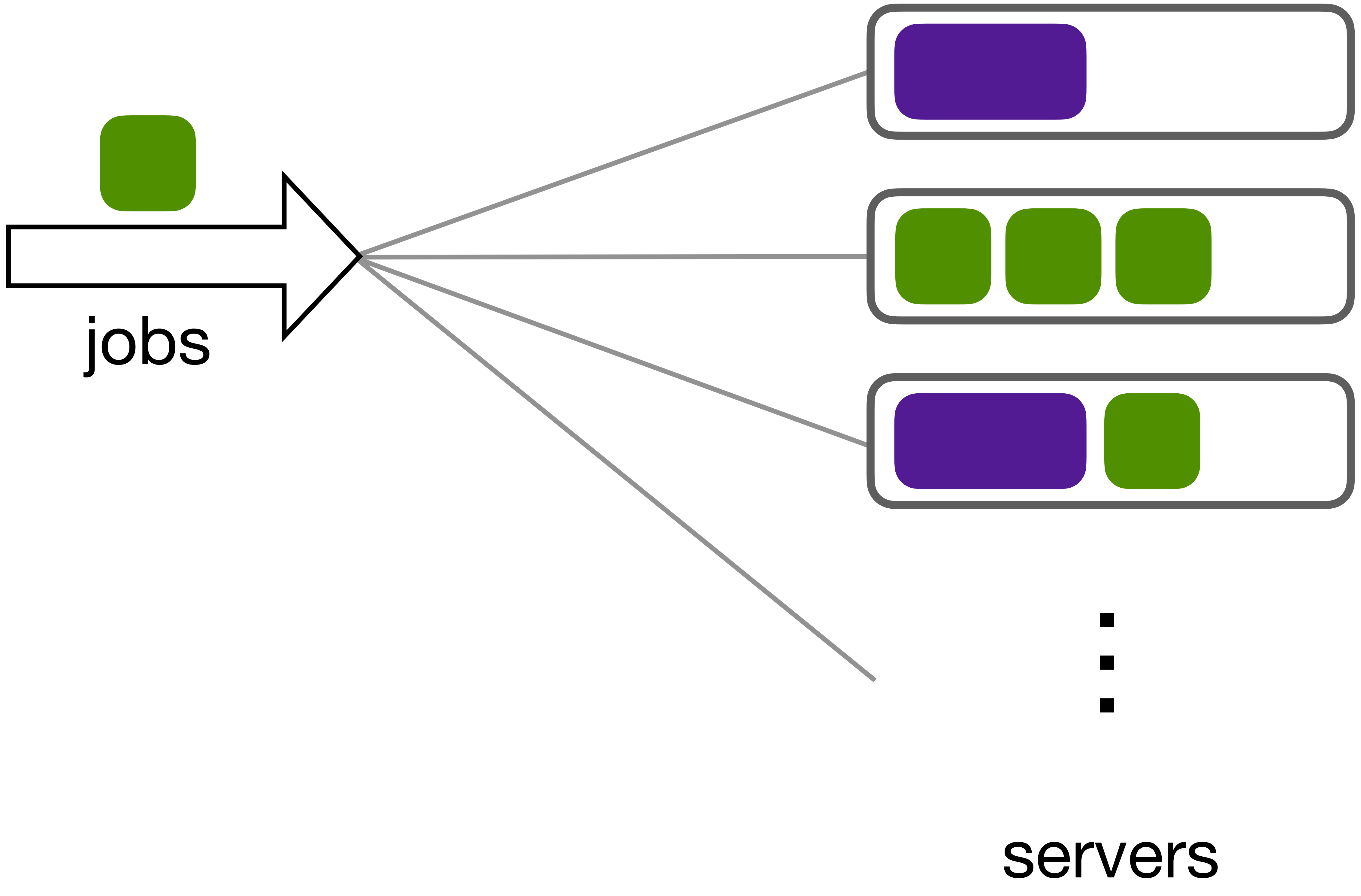


Reducing dimensionality

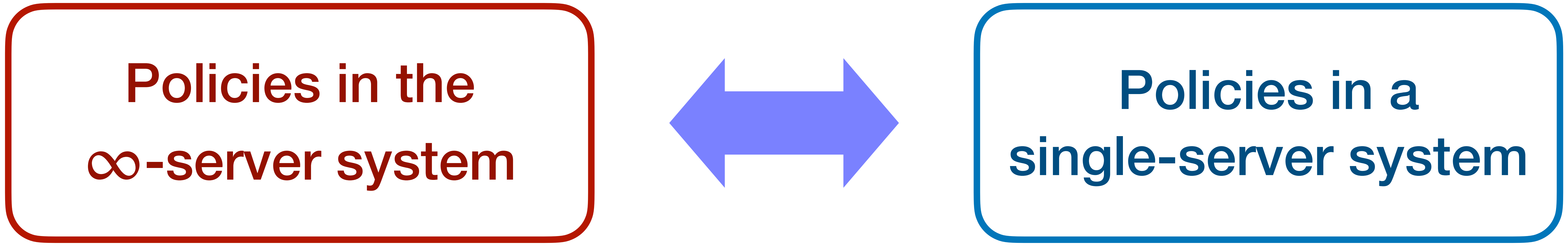
state space is large!

state: # jobs of each type
on each server

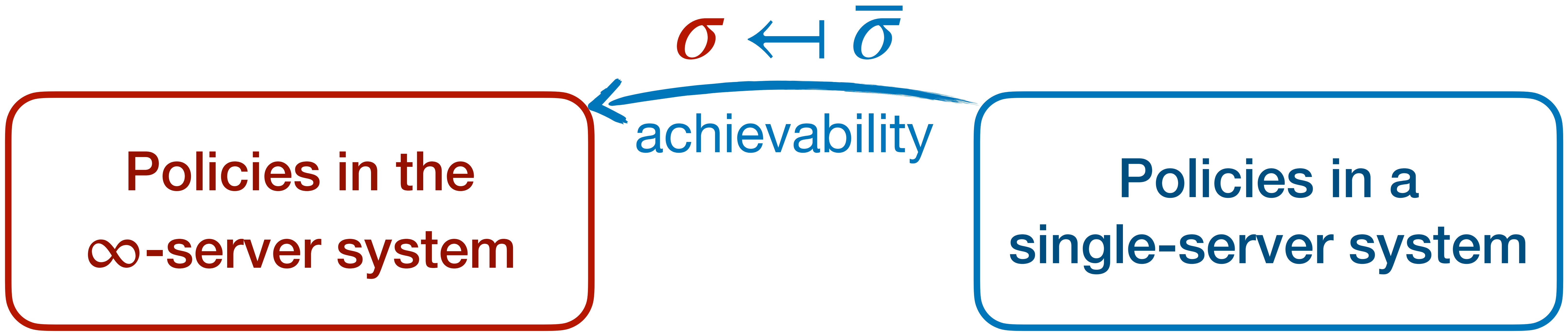
- Server-by-server evaluation:
  - How to evaluate each server?
  - How to relate to E[# active servers]?



A policy-conversion framework

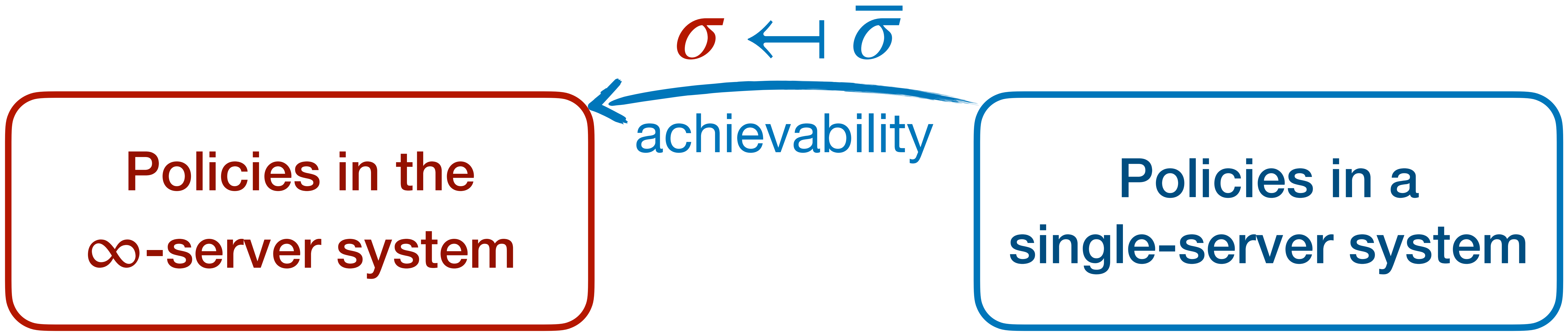


A policy-conversion framework



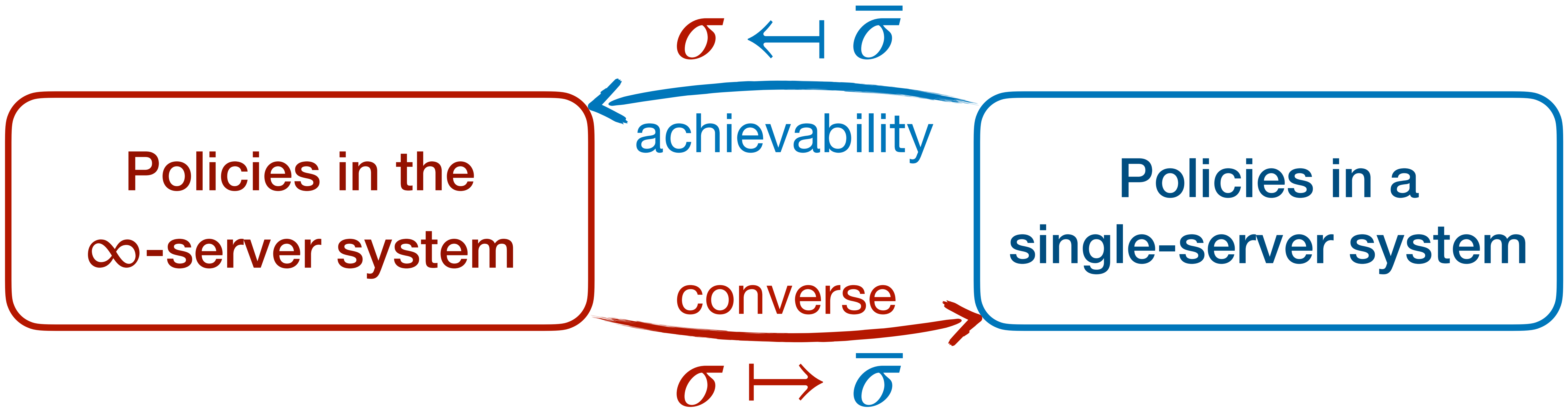
A policy-conversion framework

- Use $\overline{\sigma}$ to tell how to evaluate each server
- Performance of σ is related to properties of $\overline{\sigma}$



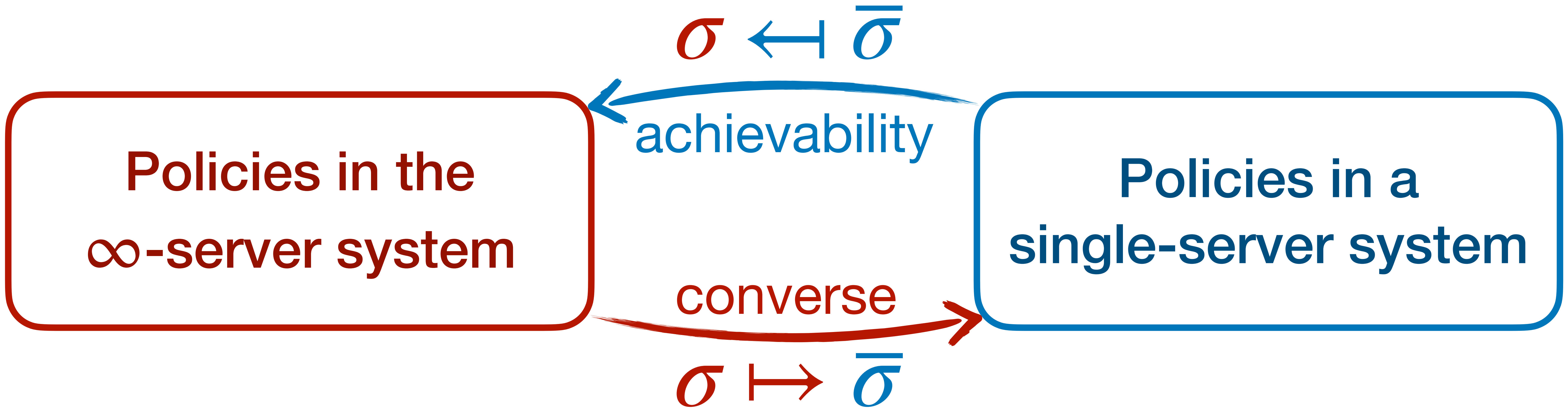
A policy-conversion framework

- Use $\overline{\sigma}$ to tell how to evaluate each server
- Performance of σ is related to properties of $\overline{\sigma}$



A policy-conversion framework

- Use $\overline{\sigma}$ to tell how to evaluate each server
- Performance of σ is related to properties of $\overline{\sigma}$



- Allows us to obtain lower bound on E[# active servers]

A policy-conversion framework

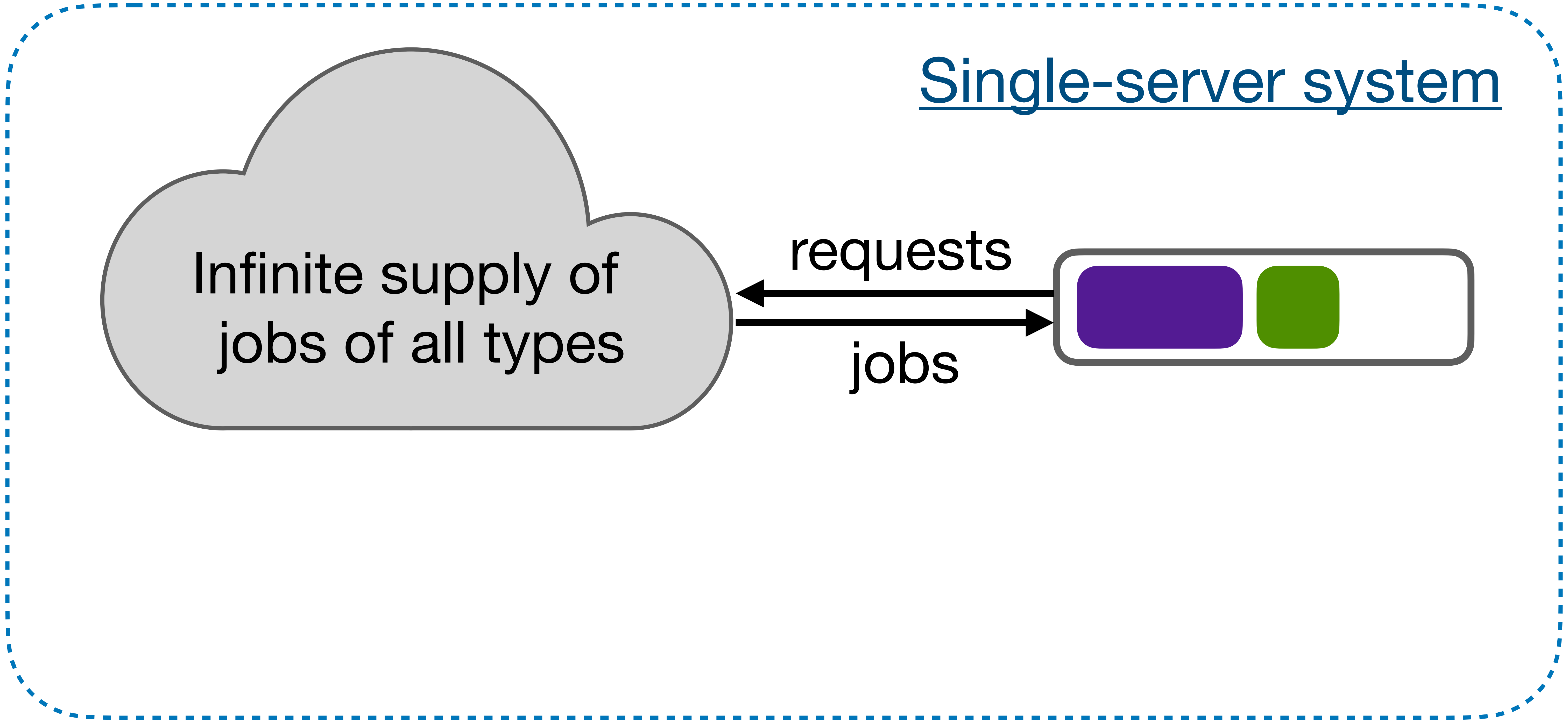
Policies in the ∞-server system ⟷ Policies in a single-server system

Single-server system

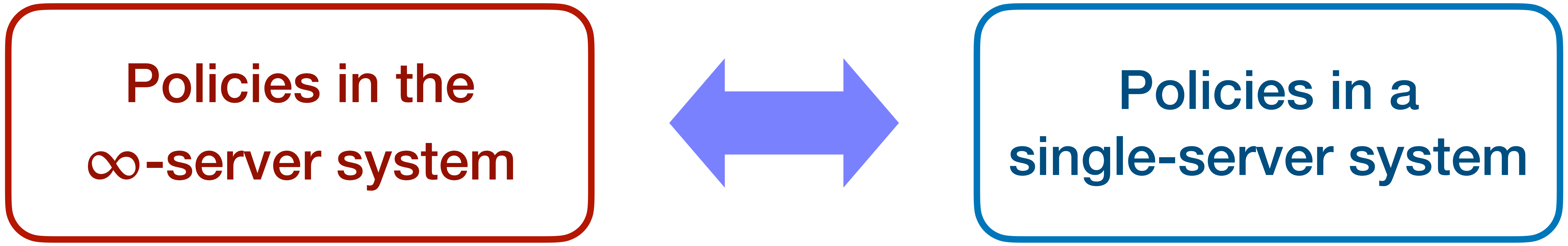
A policy-conversion framework

Policies in the ∞-server system

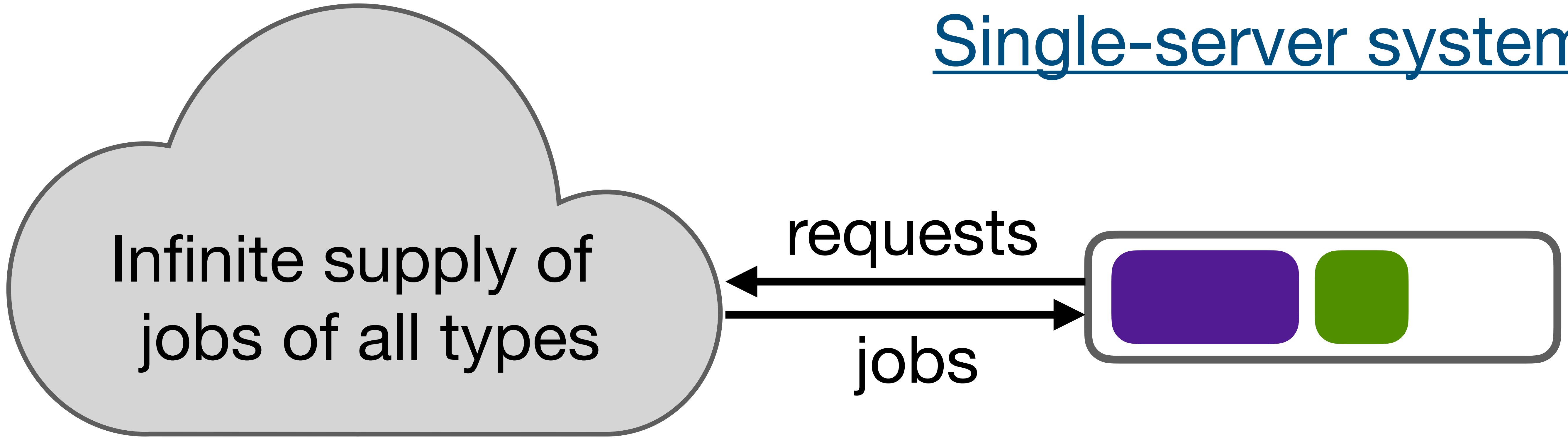
Policies in a single-server system



A policy-conversion framework



Single-server system



A policy decides when to request what types of jobs to:

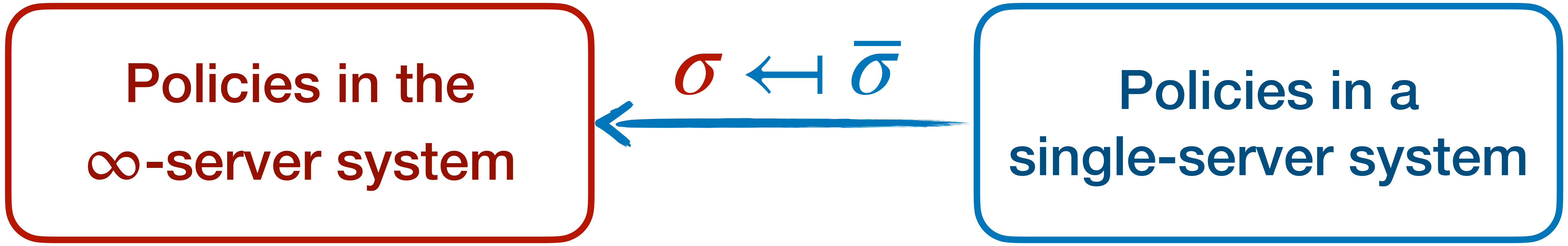
maximize throughput

subject to **cost** (resource contention) ≤ budget

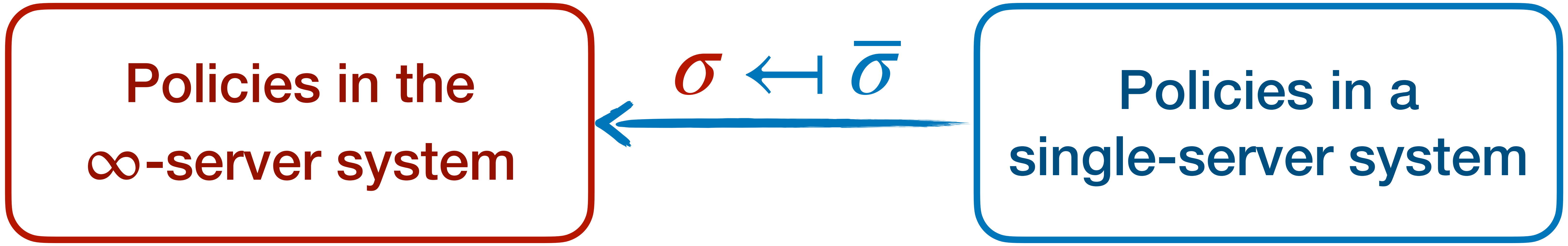
Policies in the ∞-server system

$$\sigma \leftarrow \bar{\sigma}$$

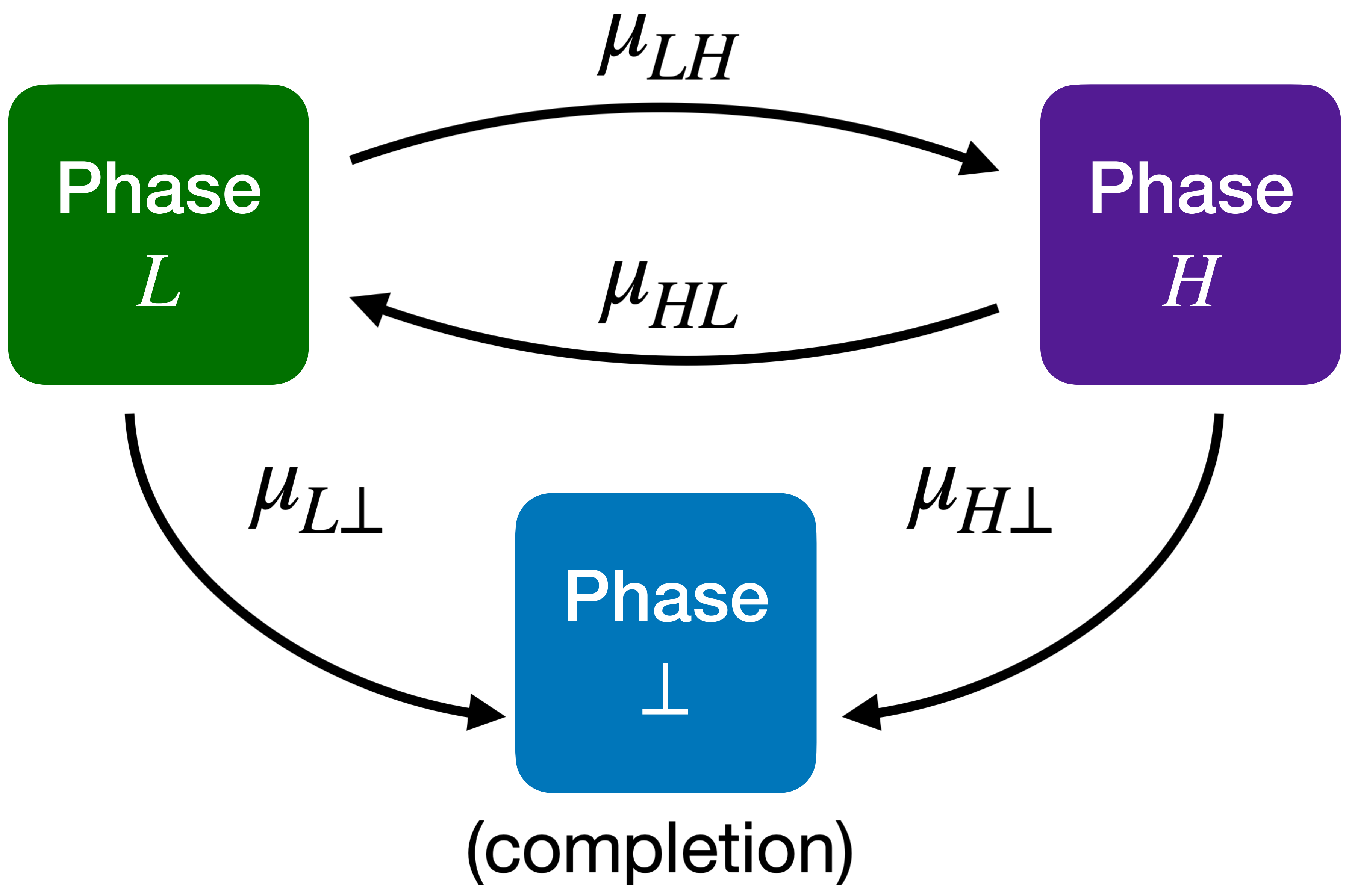
Policies in a single-server system

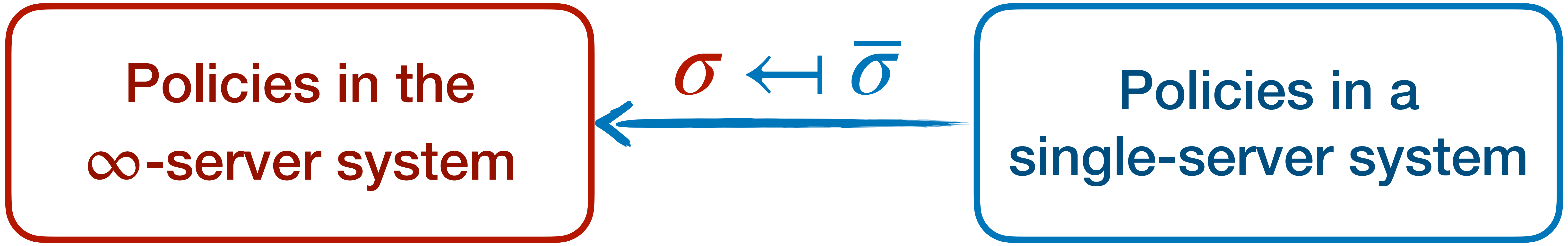


- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$

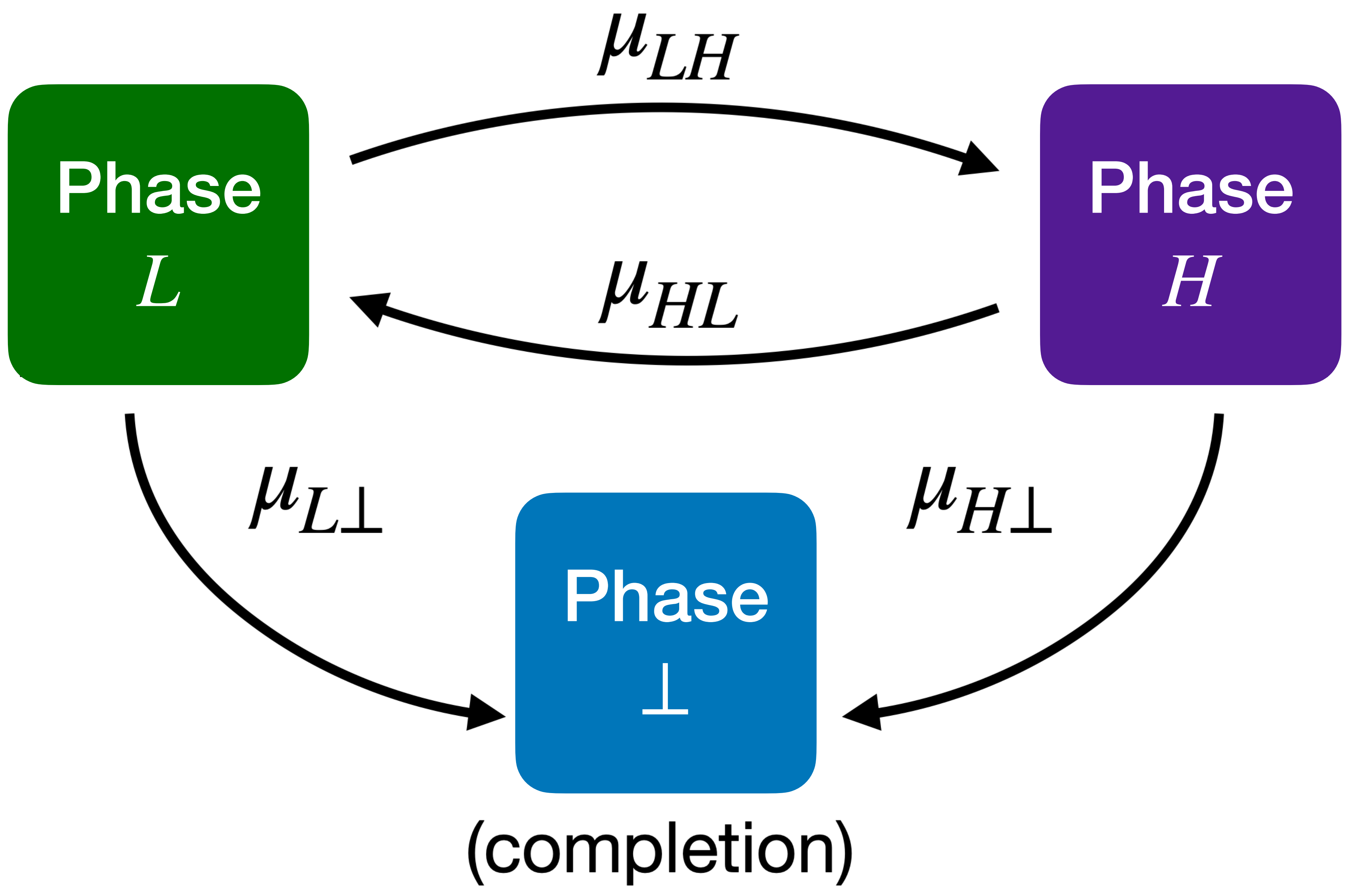


- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$



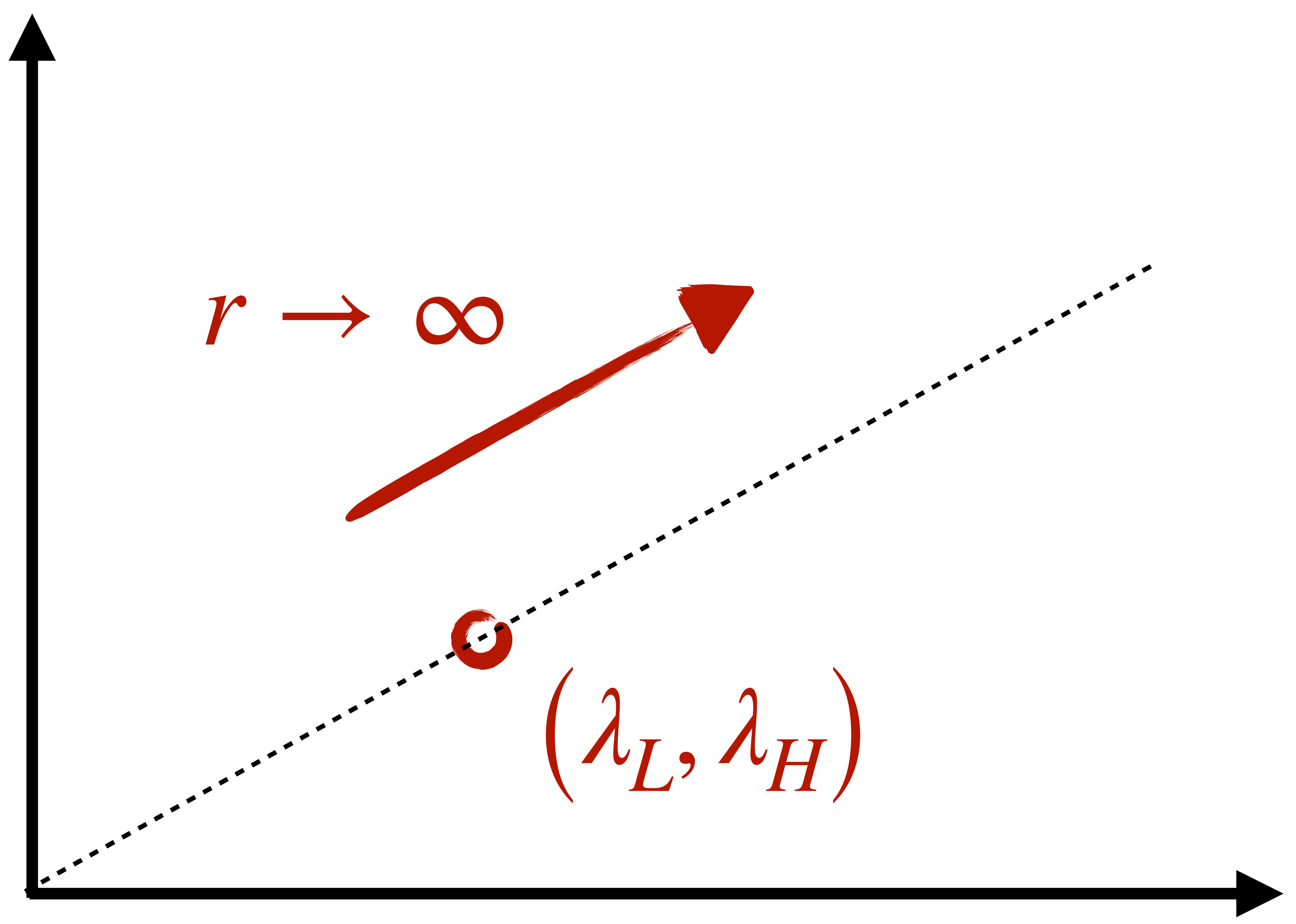


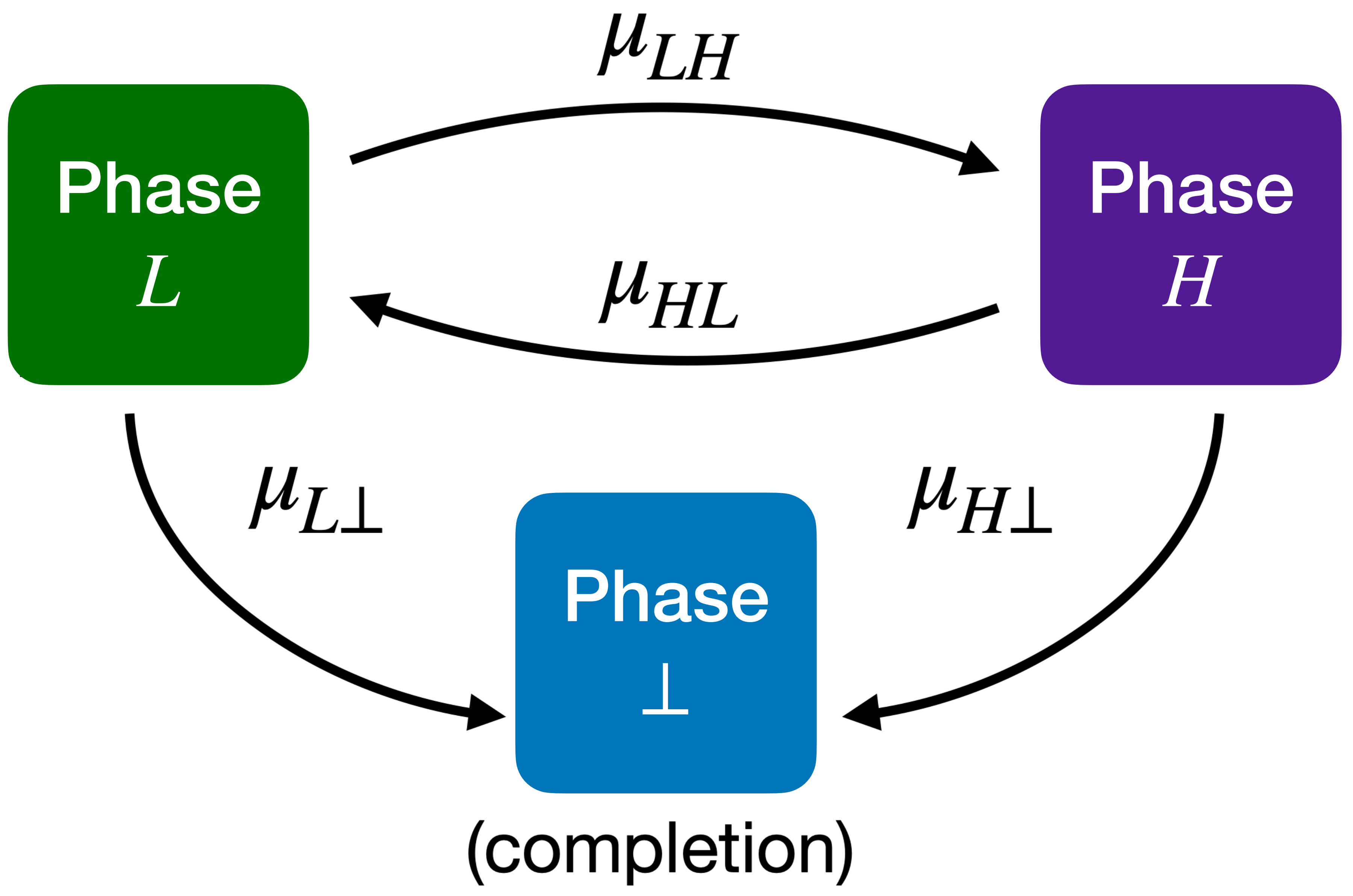
- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$
- Asymptotic regime: $r \to +\infty$



Policies in the ∞-server system $\sigma \leftarrow\!\shortmid \bar{\sigma}$ Policies in a single-server system

- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$
- Asymptotic regime: $r \to +\infty$





Policies in the ∞-server system

$\sigma \mapsfrom \bar{\sigma}$

Policies in a single-server system

- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$
- Asymptotic regime: $r \to +\infty$

Policies in the ∞-server system

$\sigma \leftarrow\!\shortmid \bar{\sigma}$

Policies in a single-server system

- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$
- Asymptotic regime: $r \to +\infty$

Policy $\bar{\sigma}$

throughput $\cdot \bar{N} = r \cdot (\lambda_L, \lambda_H)$

cost (resource contention) ≤ budget

Policies in the ∞-server system

$\sigma \leftarrow\!\!\dashv \bar{\sigma}$

Policies in a single-server system

- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$
- Asymptotic regime: $r \to +\infty$

Policy $\bar{\sigma}$

throughput $\cdot \overline{N} = r \cdot (\lambda_L, \lambda_H)$

cost (resource contention) ≤ budget

convert

Policy σ

E [# active servers] $\leq \left(1 + O\left(r^{-0.5}\right)\right) \cdot \overline{N}$

cost (resource contention) $\leq \left(1 + O\left(r^{-0.5}\right)\right) \cdot$ budget

Policies in the ∞-server system

$\sigma \leftarrow\!\dashv \bar{\sigma}$

Policies in a single-server system

- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$
- Asymptotic regime: $r \to +\infty$

Policy $\bar{\sigma}$

throughput $\cdot \, \overline{N} = r \cdot (\lambda_L, \lambda_H)$

cost (resource contention) $\leq$ budget

convert

Policy σ

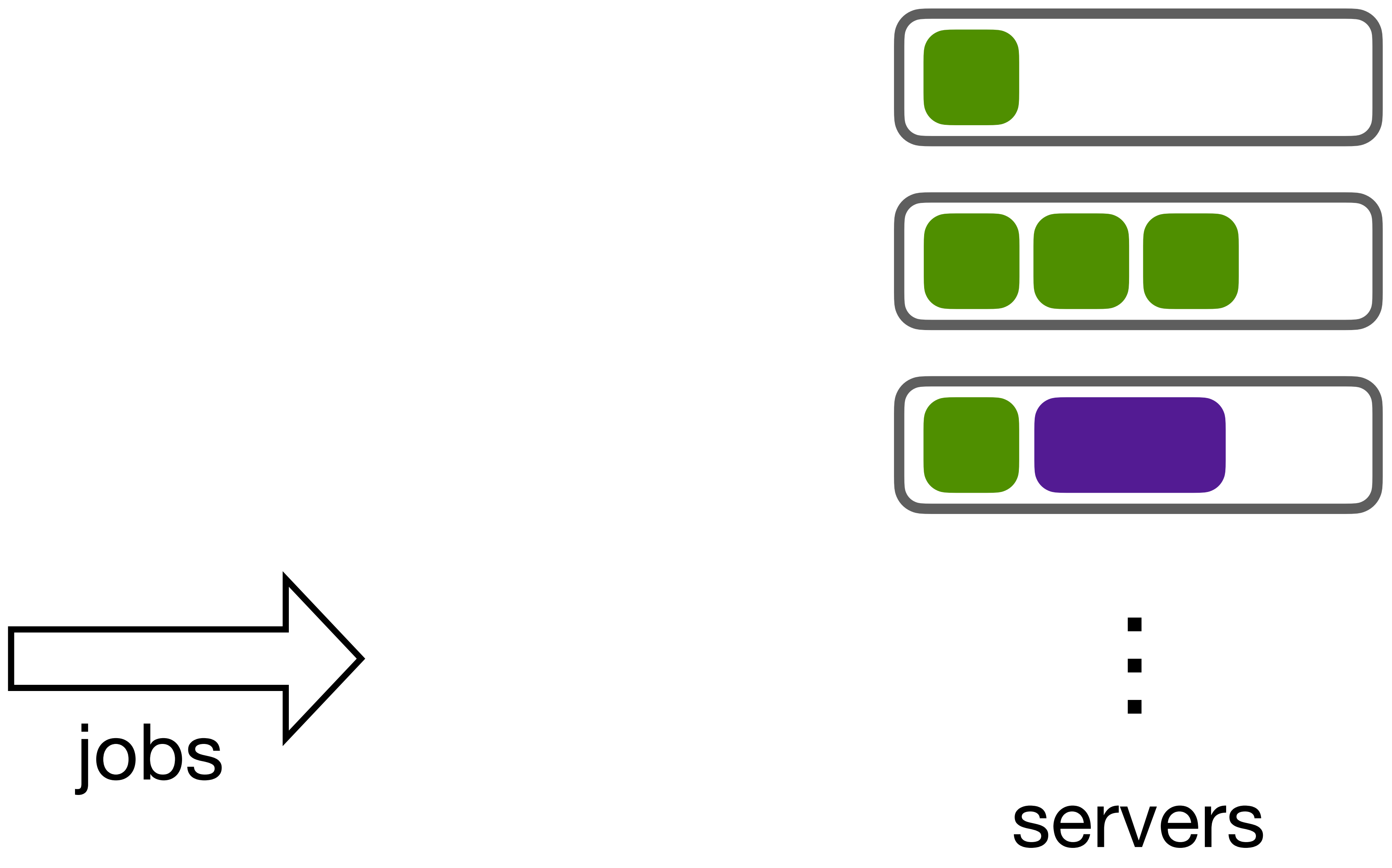
E [# active servers] $\leq \left(1 + O\left(r^{-0.5}\right)\right) \cdot \overline{N}$

cost (resource contention) $\leq \left(1 + O\left(r^{-0.5}\right)\right) \cdot$ budget

Main Result: We design a policy for the original ∞-server system that is *asymptotically optimal*

Policy conversion: single-server to ∞-server

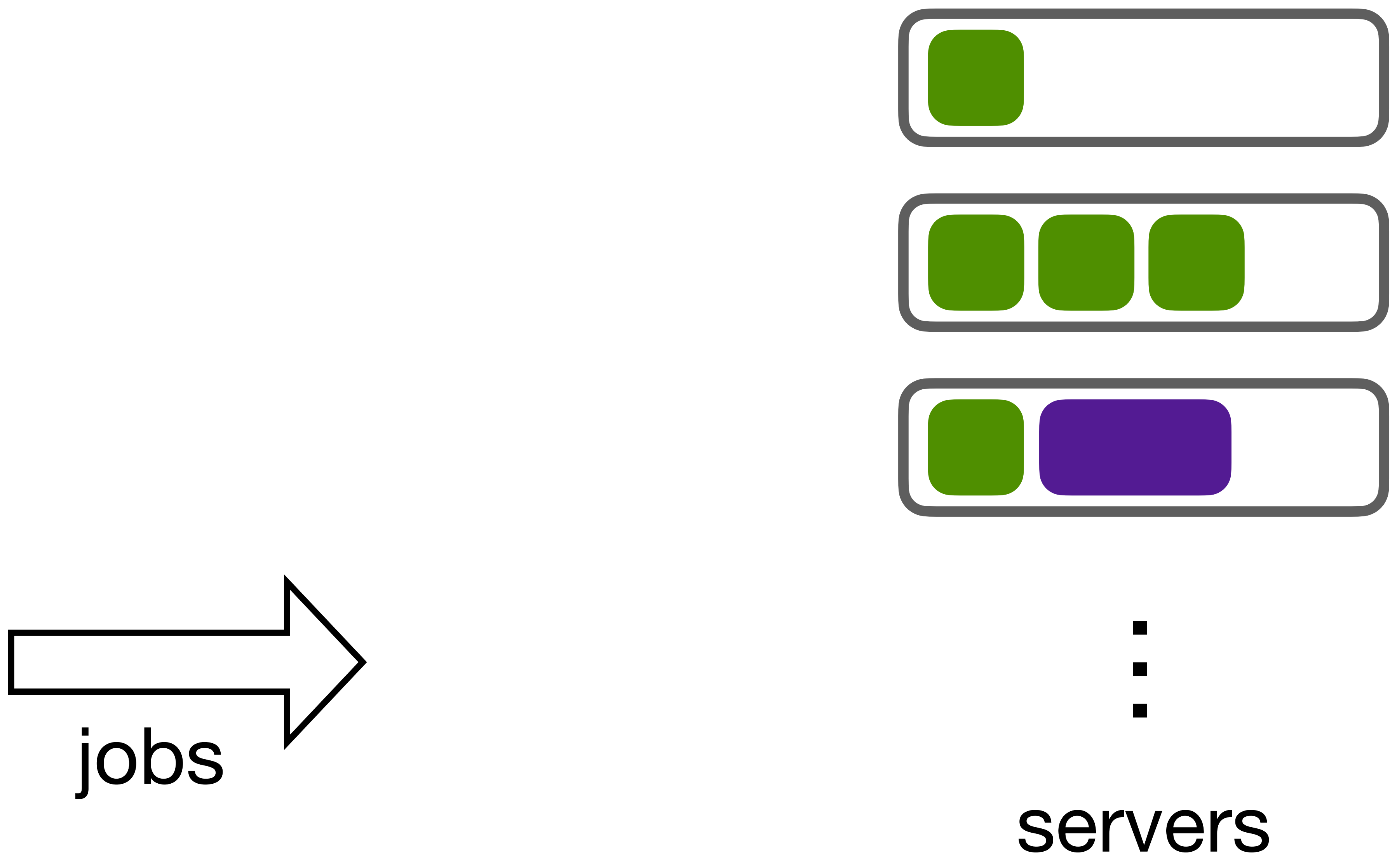
Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\overline{\sigma}$)



Policy conversion: single-server to ∞-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\overline{\sigma}$)

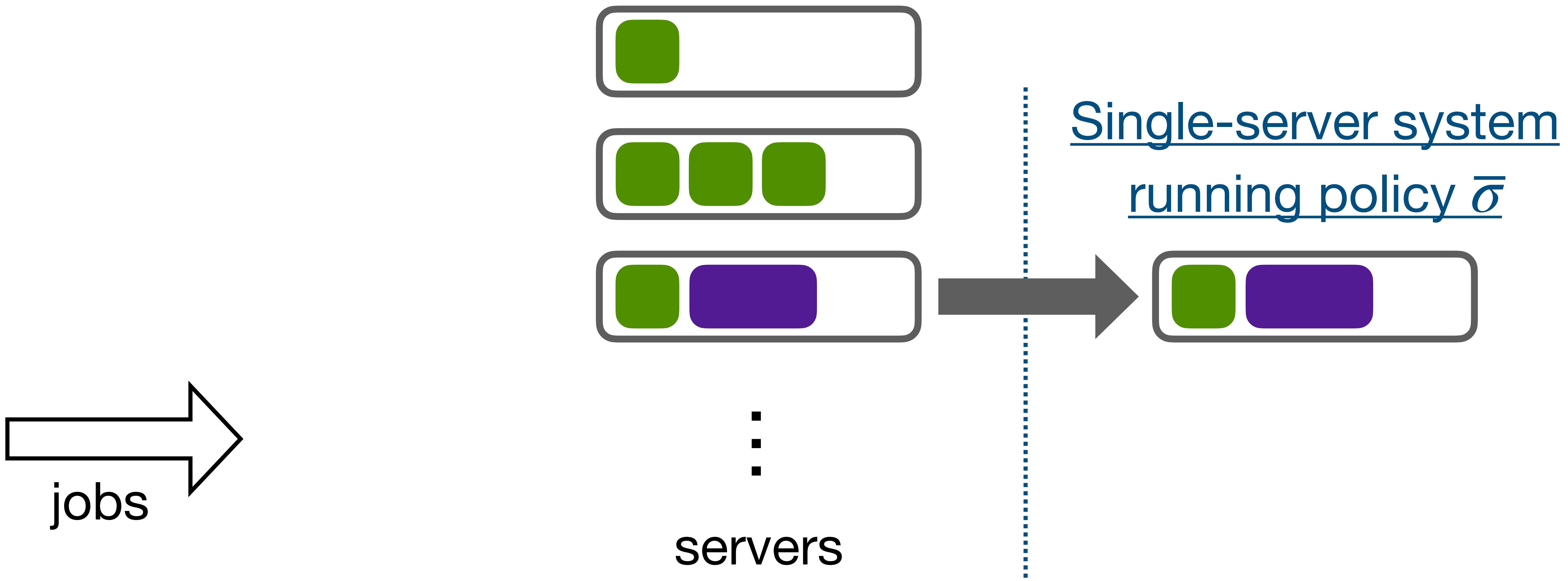
- For each server, run a single-server policy $\overline{\sigma}$



Policy conversion: single-server to ∞-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\overline{\sigma}$)

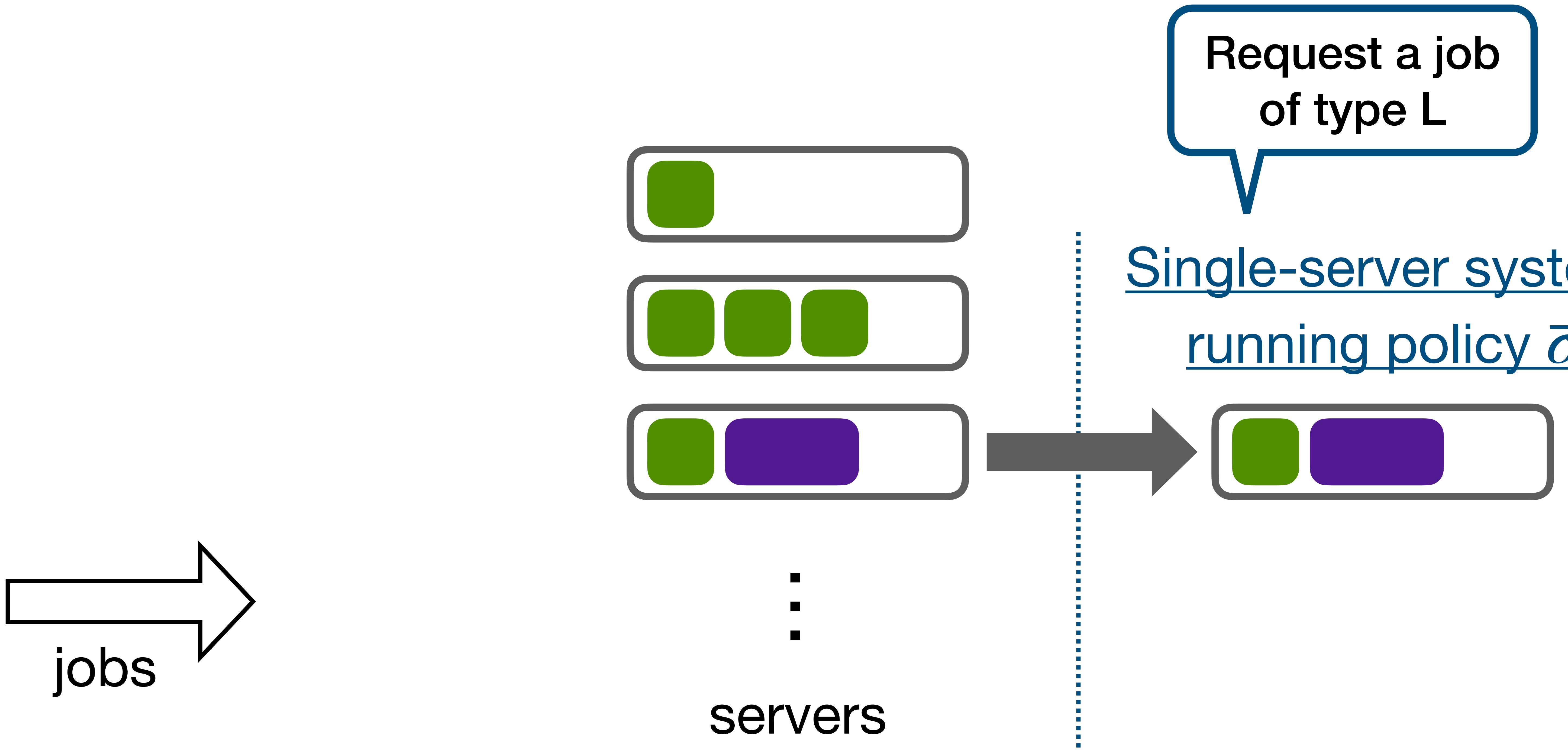
- For each server, run a single-server policy $\overline{\sigma}$



Policy conversion: single-server to ∞-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\overline{\sigma}$)

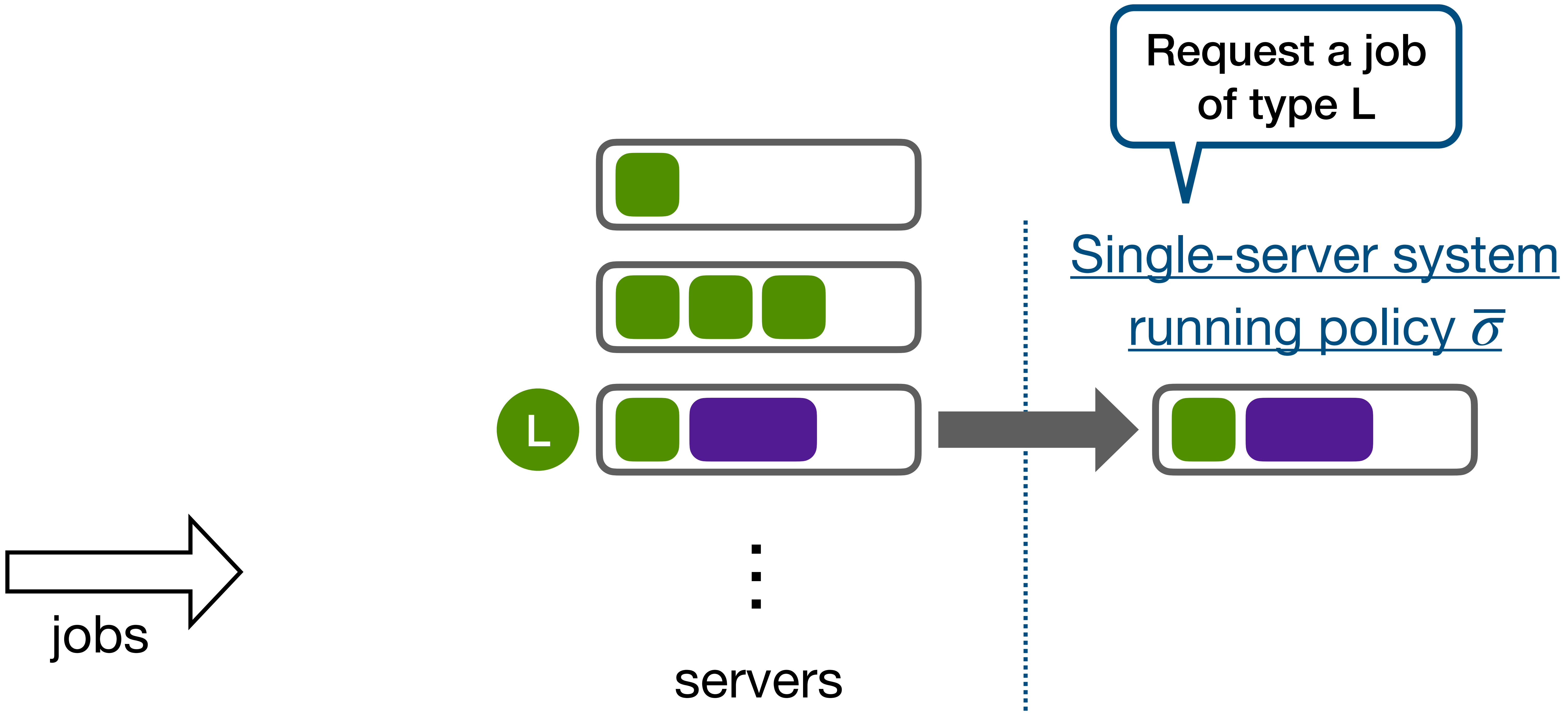
- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type i, generate a token of type i



Policy conversion: single-server to ∞-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\overline{\sigma}$)

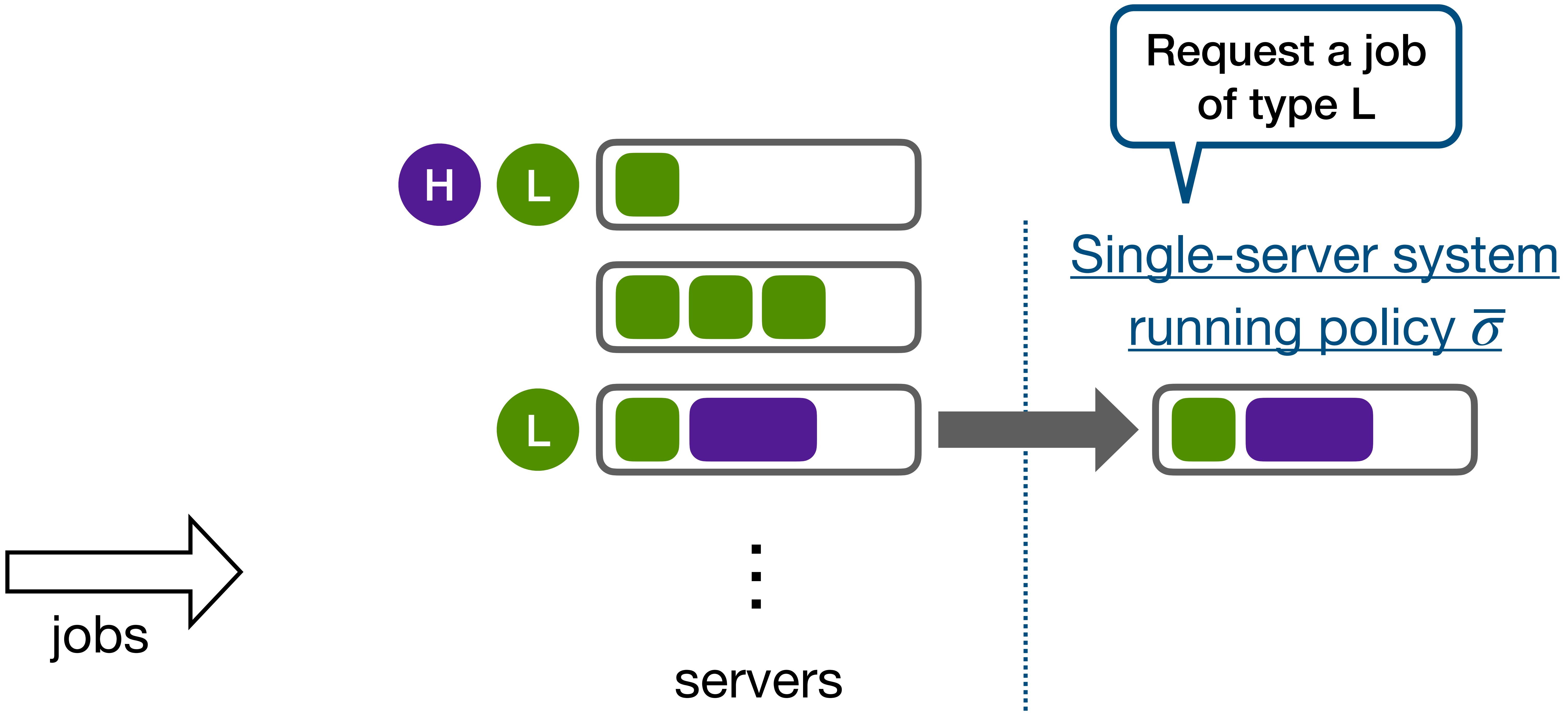
- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type i, generate a token of type i



Policy conversion: single-server to ∞-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\overline{\sigma}$)

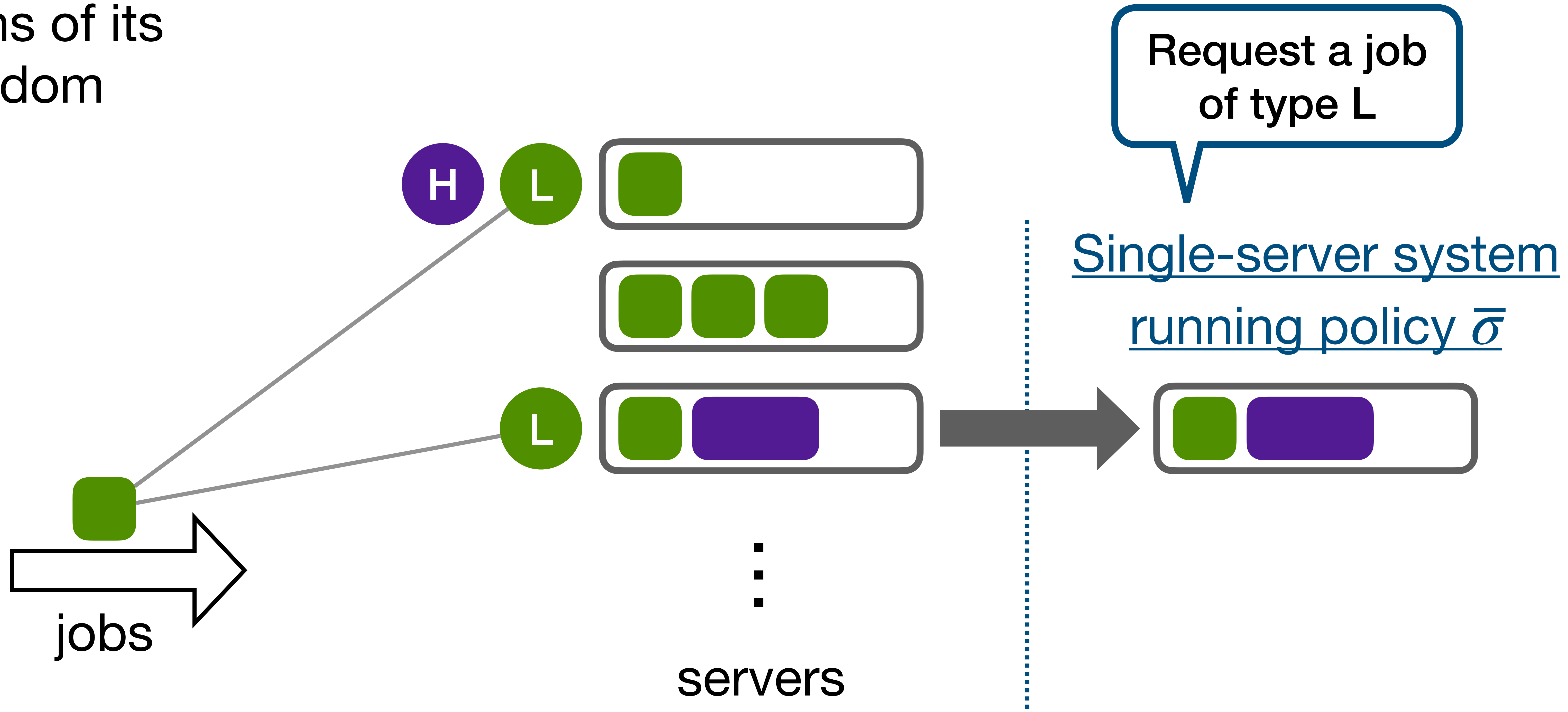
- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type i, generate a token of type i



Policy conversion: single-server to ∞-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\bar{\sigma}$)

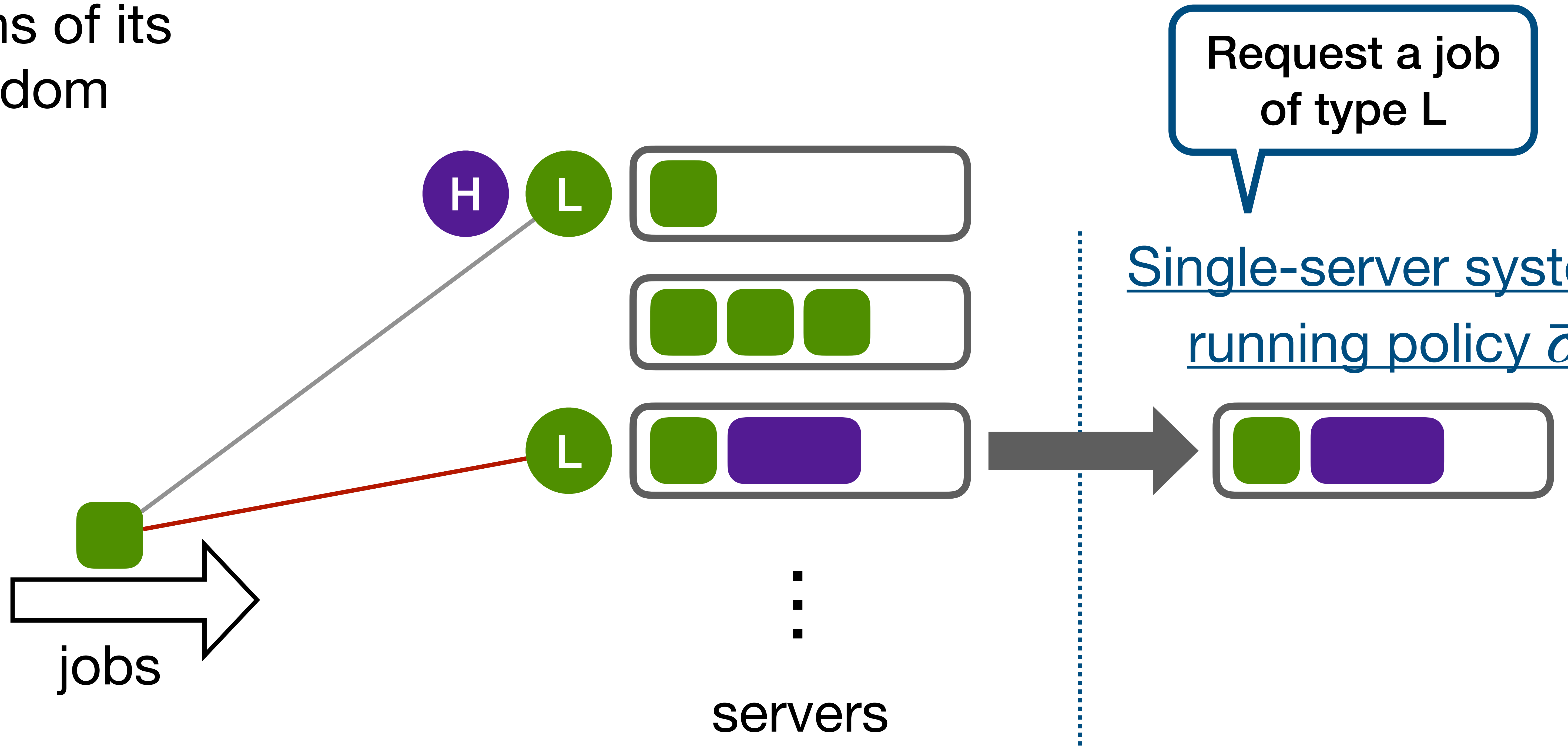
- For each server, run a single-server policy $\bar{\sigma}$
- If $\bar{\sigma}$ requests a job of type i, generate a token of type i
- When a job arrives, it checks tokens of its type and joins one uniformly at random



Policy conversion: single-server to ∞-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\overline{\sigma}$)

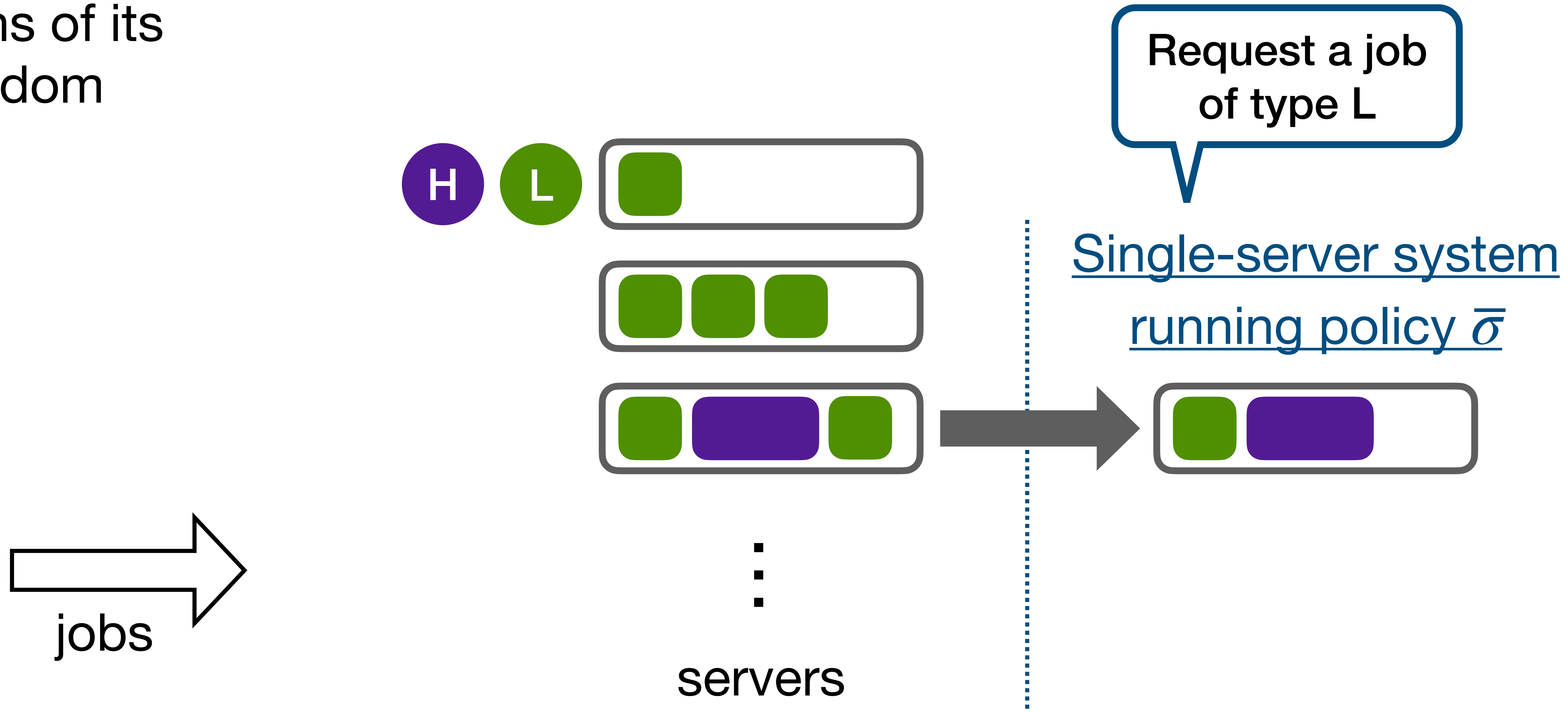
- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type i, generate a token of type i
- When a job arrives, it checks tokens of its type and joins one uniformly at random



Policy conversion: single-server to ∞-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\overline{\sigma}$)

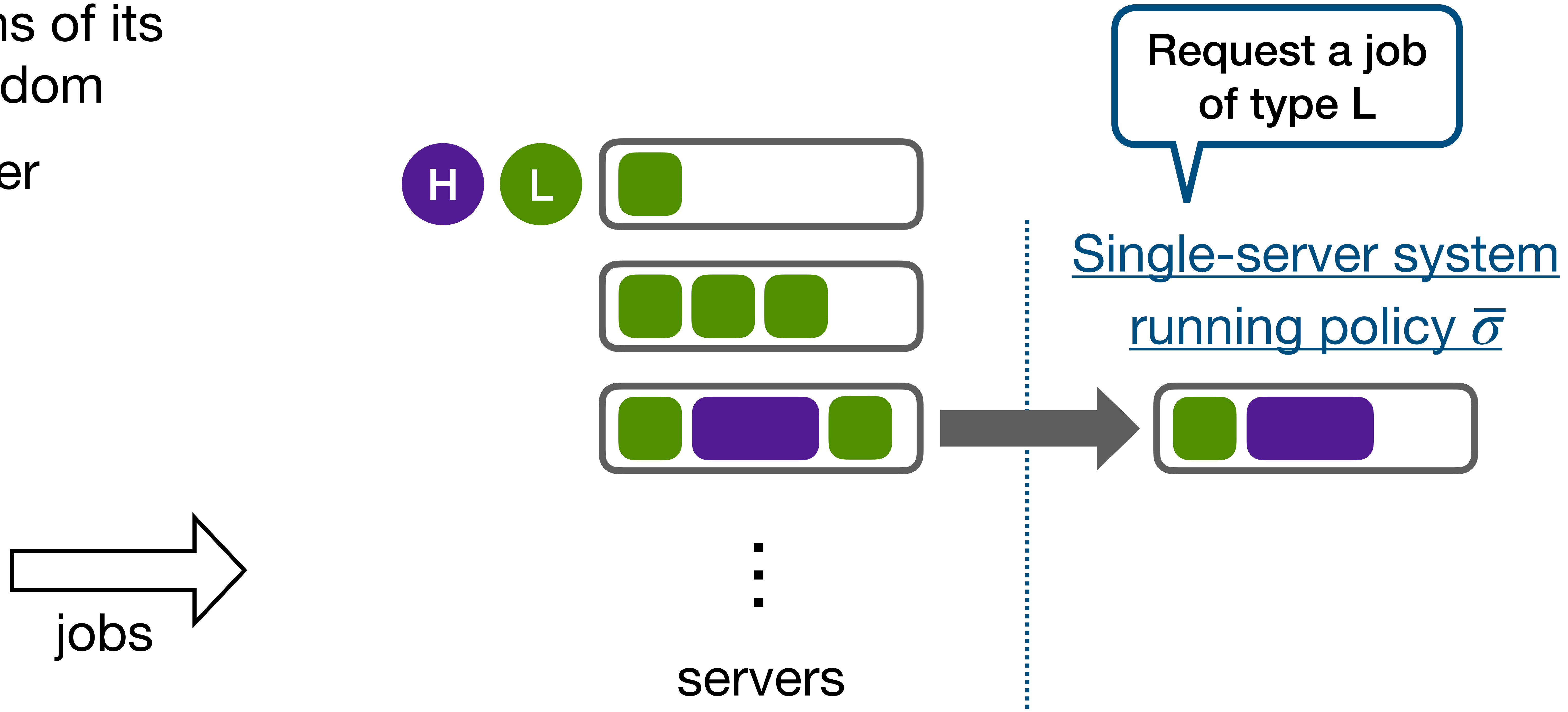
- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type i, generate a token of type i
- When a job arrives, it checks tokens of its type and joins one uniformly at random



Policy conversion: single-server to ∞-server

Meta-algorithm: Join-the-Recently-Requesting-Server ($\overline{\sigma}$)

- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type i, generate a token of type i
- When a job arrives, it checks tokens of its type and joins one uniformly at random
- If no tokens, go to an inactive server

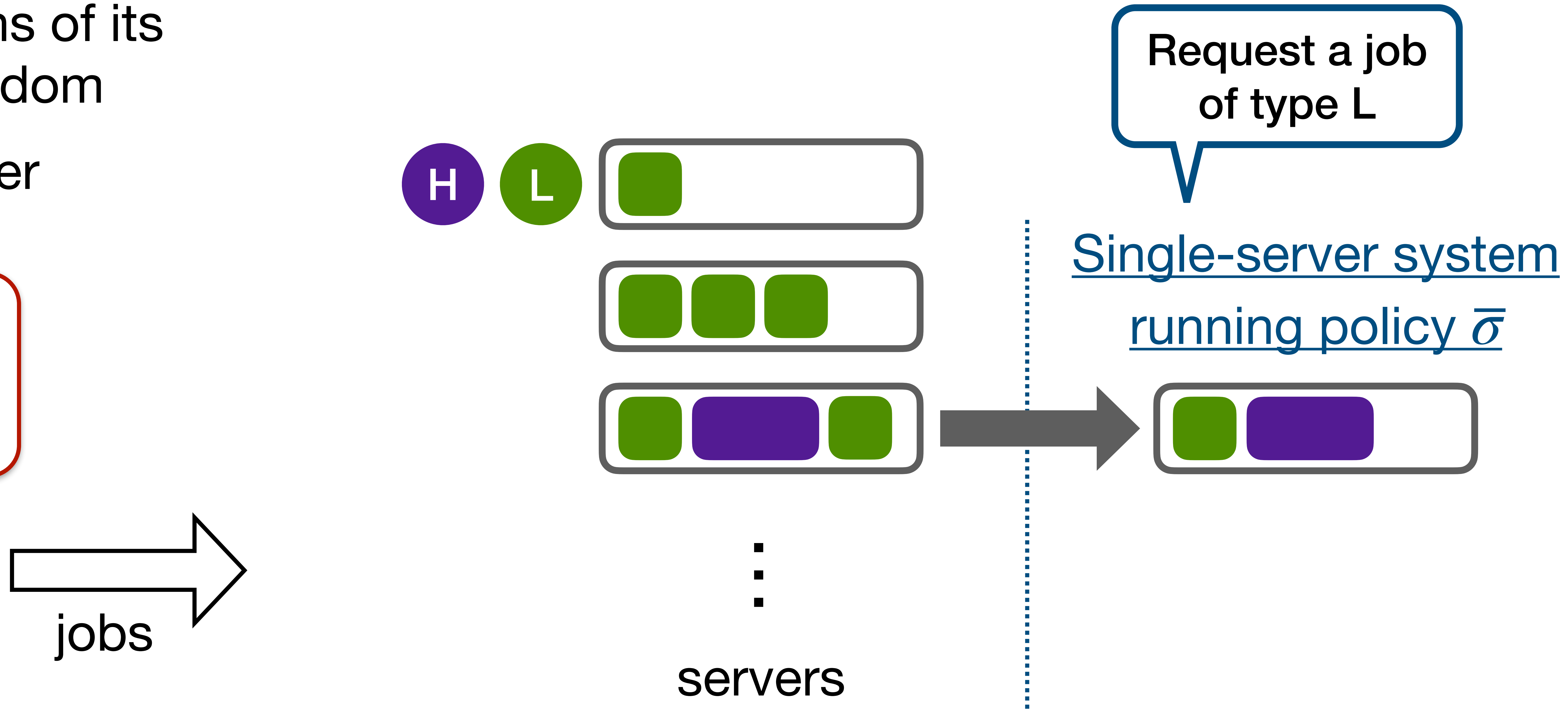


Policy conversion: single-server to ∞-server

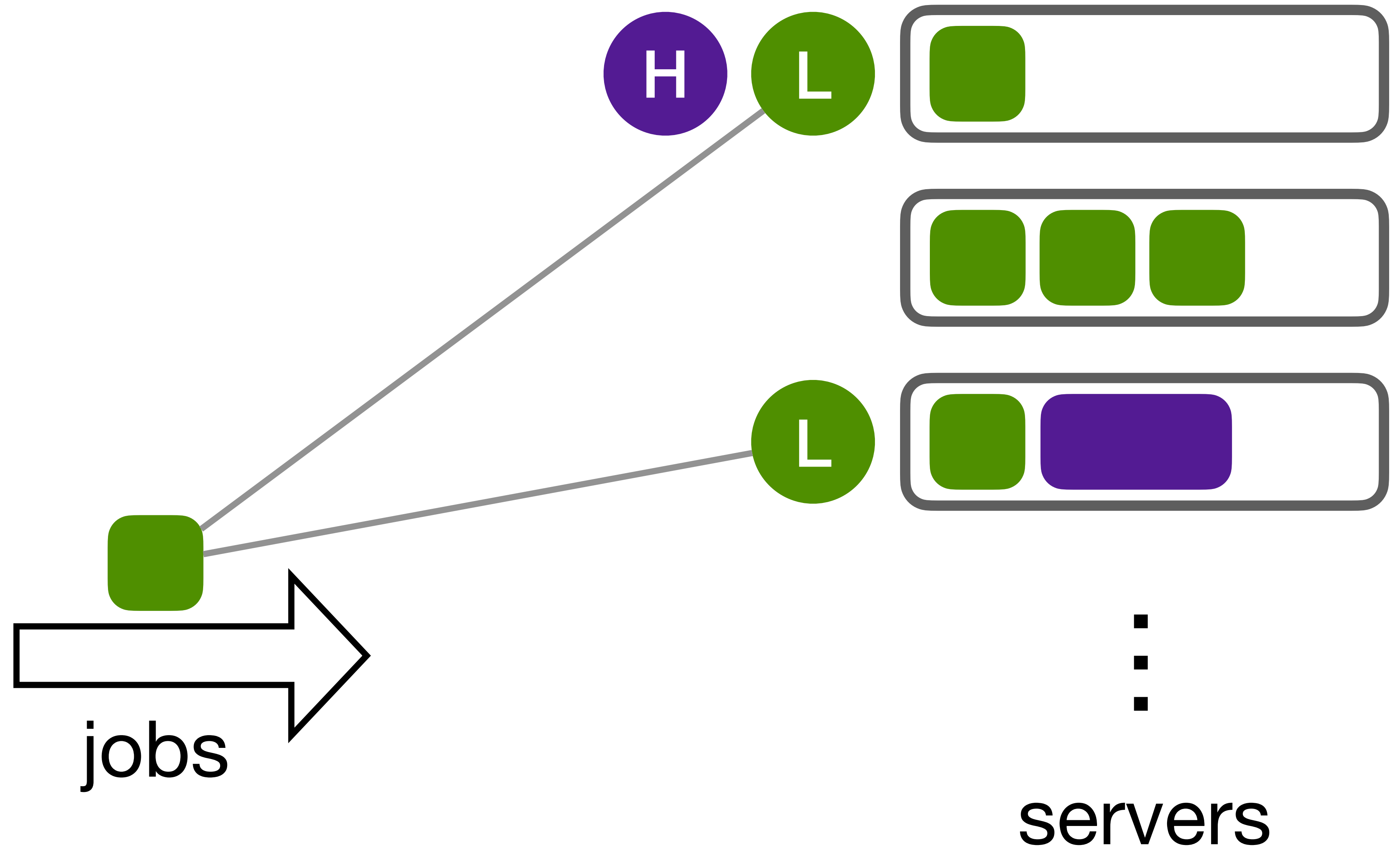
Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\overline{\sigma}$)

- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type i, generate a token of type i
- When a job arrives, it checks tokens of its type and joins one uniformly at random
- If no tokens, go to an inactive server

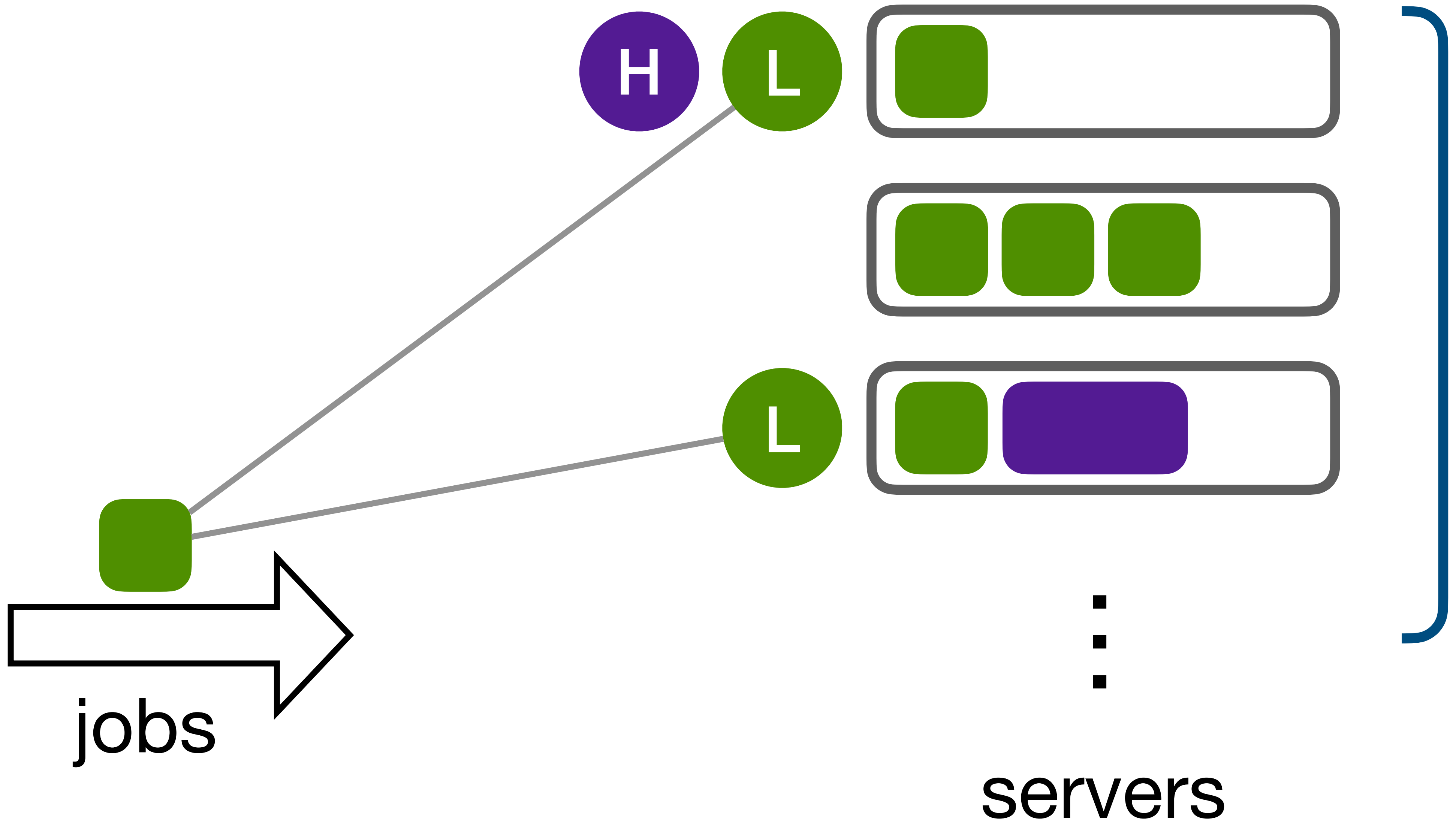
How is the throughput related to # active servers via tokens?



Policy conversion: more details



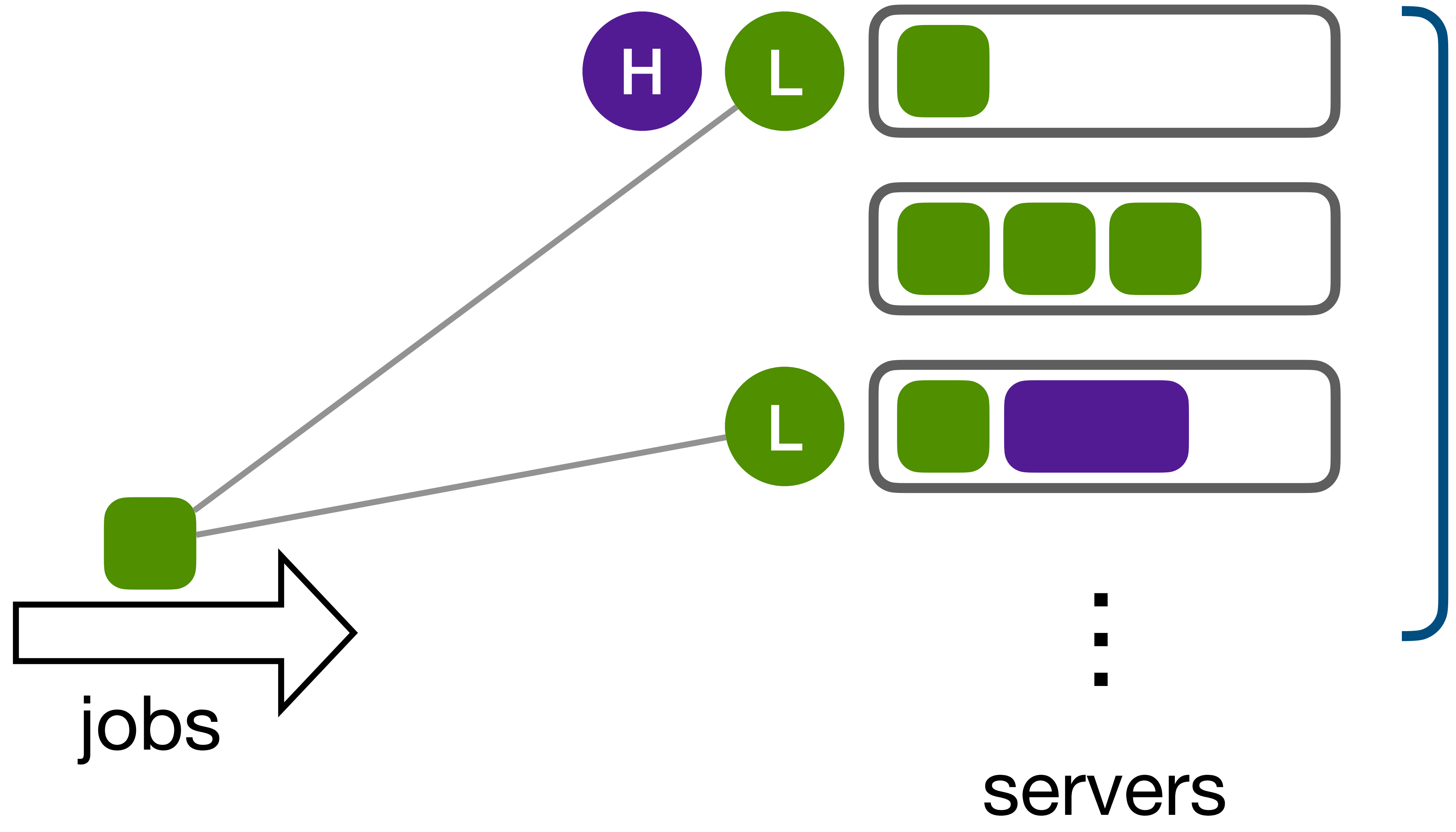
Policy conversion: more details



Run single-server policy $\overline{\sigma}$ for only

$$\overline{N} = \frac{\text{arrival rate}}{\text{throughtput}(\overline{\sigma})} \text{ servers}$$

Policy conversion: more details



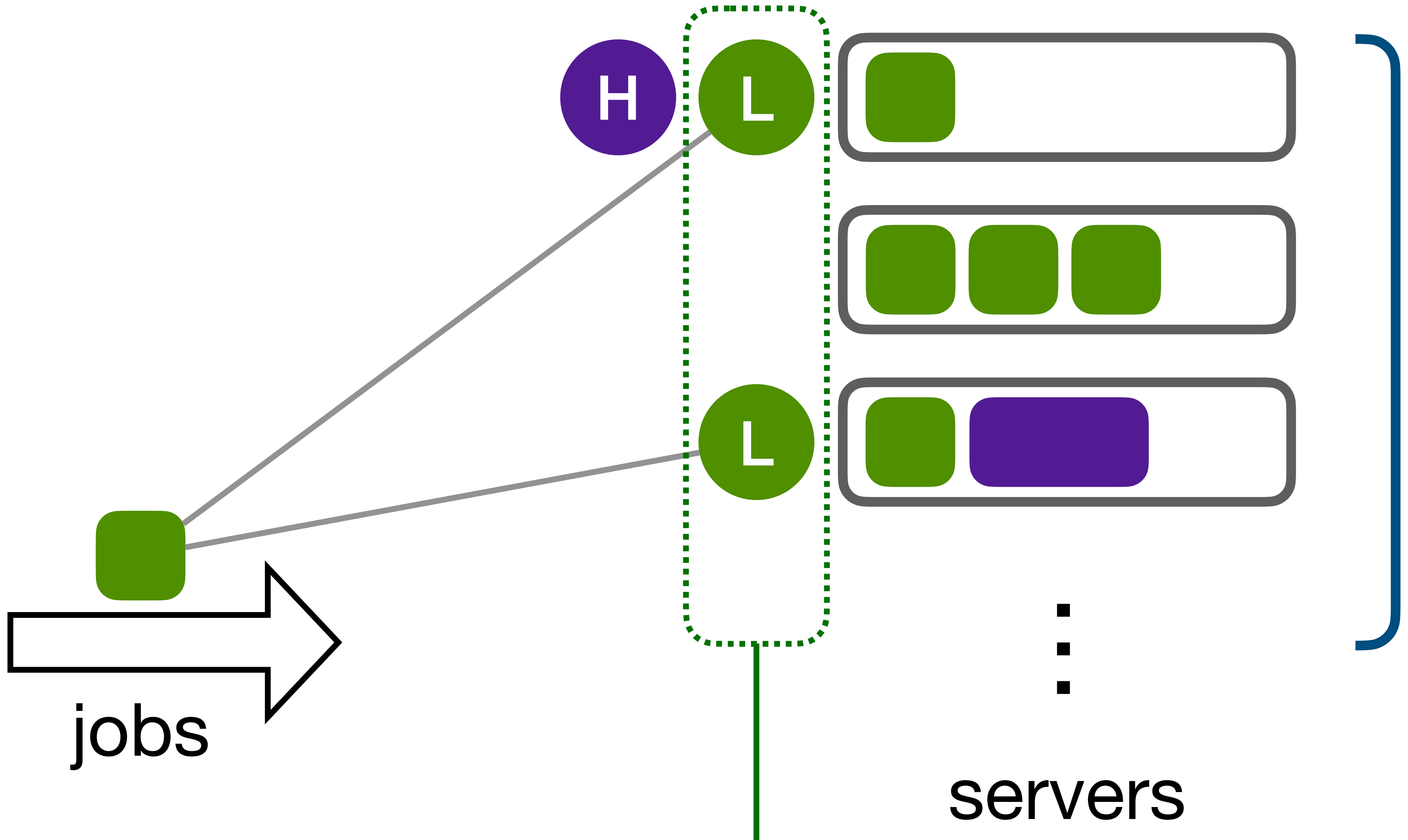
Run single-server policy $\overline{\sigma}$ for only

$$\overline{N} = \frac{\text{arrival rate}}{\text{throughtput}(\overline{\sigma})} \text{ servers}$$

Recall that we aim to show

$$\mathbf{E}\left[\#\text{ active servers}\right] \leq \left(1 + O\left(r^{-0.5}\right)\right) \cdot \overline{N}$$

Policy conversion: more details



When the # tokens of a type $> \sqrt{r}$, remove the overflow tokens and generate virtual jobs

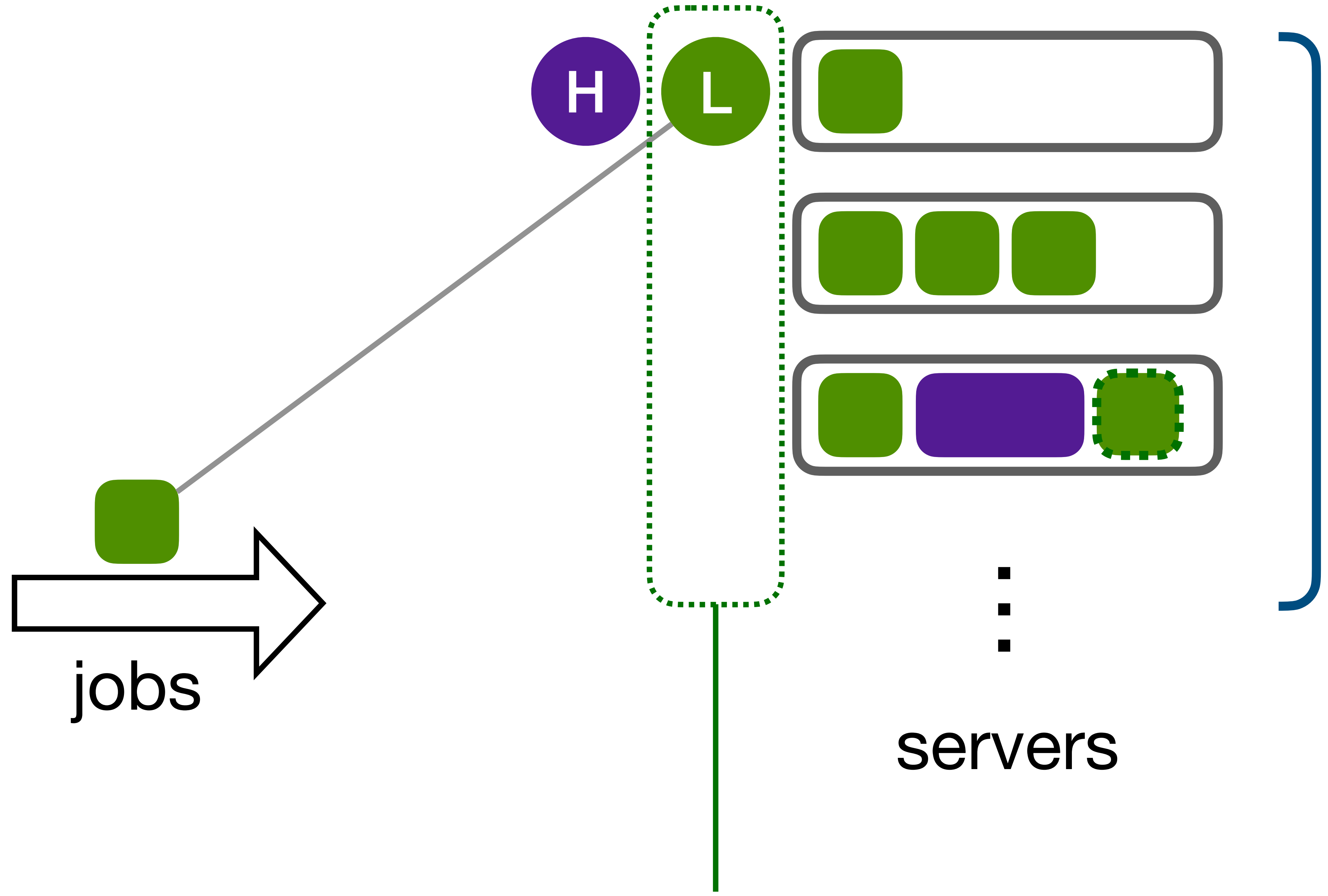
Run single-server policy $\overline{\sigma}$ for only

$$\overline{N} = \frac{\text{arrival rate}}{\text{throughtput}(\overline{\sigma})} \text{ servers}$$

Recall that we aim to show

$$\mathbf{E}\left[\#\text{ active servers}\right] \leq \left(1 + O\left(r^{-0.5}\right)\right) \cdot \overline{N}$$

Policy conversion: more details



When the # tokens of a type $> \sqrt{r}$, remove the overflow tokens and generate virtual jobs

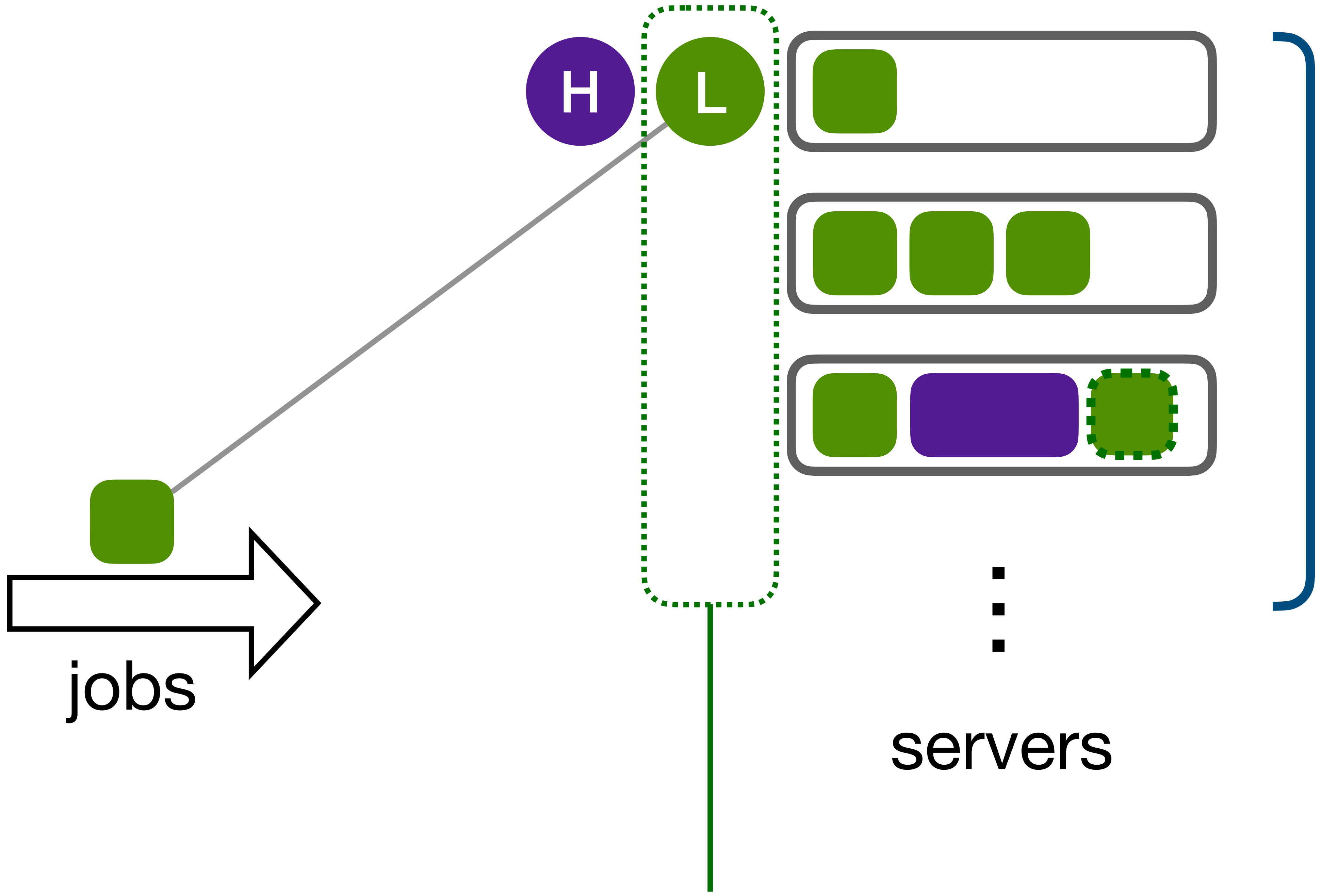
Run single-server policy $\overline{\sigma}$ for only

$$\overline{N} = \frac{\text{arrival rate}}{\text{throughtput}(\overline{\sigma})} \text{ servers}$$

Recall that we aim to show

$$\mathbf{E}\,[\text{\# active servers}] \leq \left(1 + O\left(r^{-0.5}\right)\right) \cdot \overline{N}$$

Policy conversion: more details



When the # tokens of a type $> \sqrt{r}$, remove the overflow tokens and generate virtual jobs

Run single-server policy $\bar{\sigma}$ for only

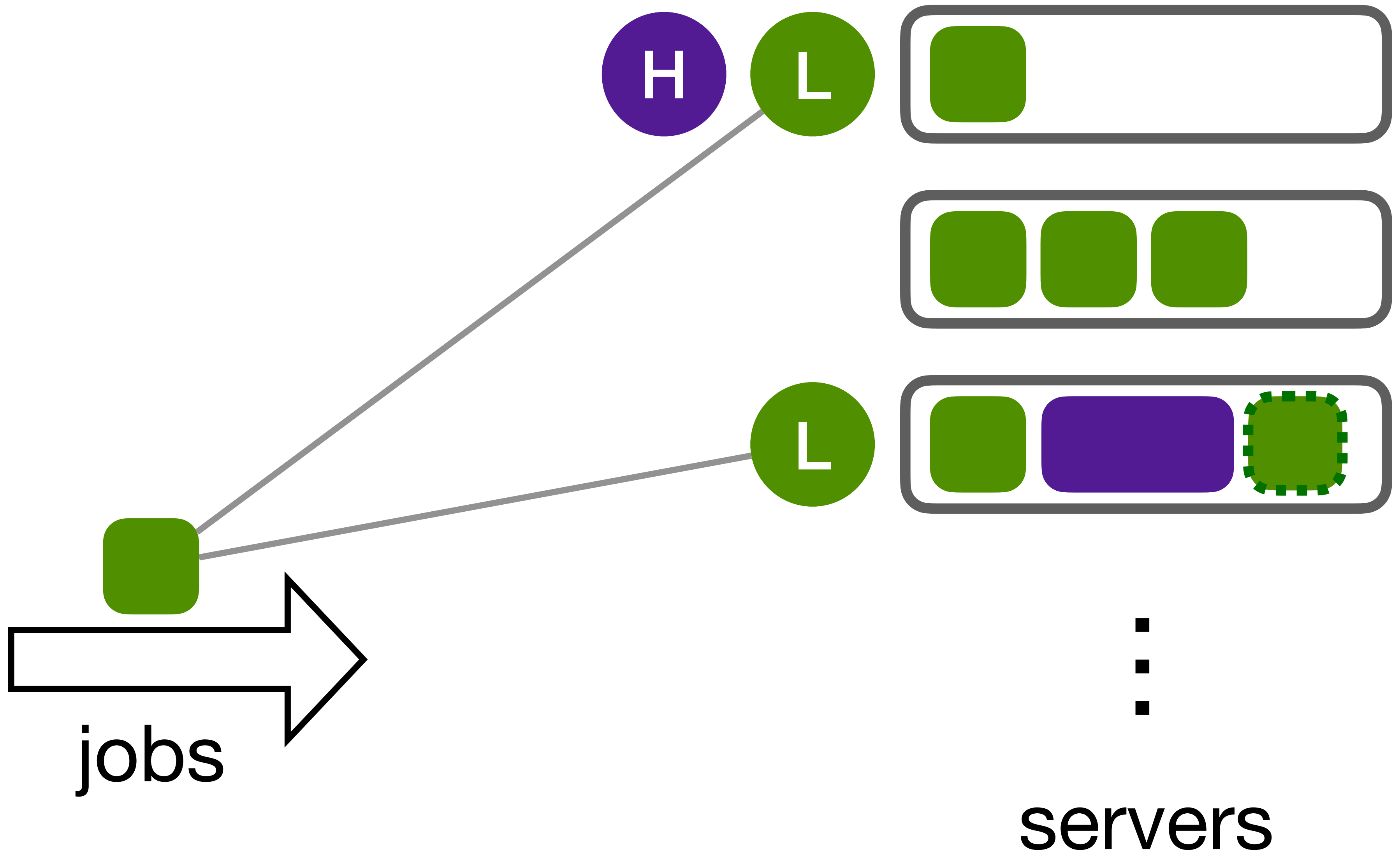
$$\bar{N} = \frac{\text{arrival rate}}{\text{throughtput}(\bar{\sigma})} \text{ servers}$$

Recall that we aim to show

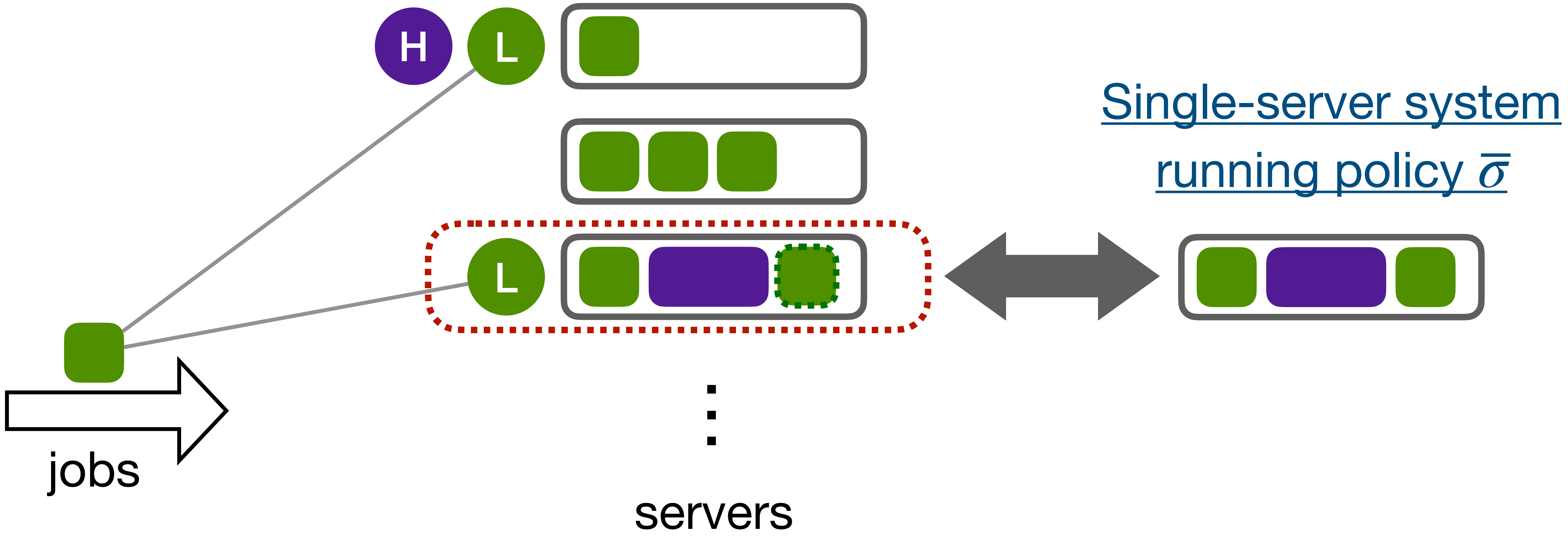
$$\mathbf{E}\,[\text{\# active servers}] \leq \left(1 + O\left(r^{-0.5}\right)\right) \cdot \bar{N}$$

We can prove that $\mathbf{E}\,[\text{\# virtual jobs}] = O\left(r^{0.5}\right)$

Key proof idea 1

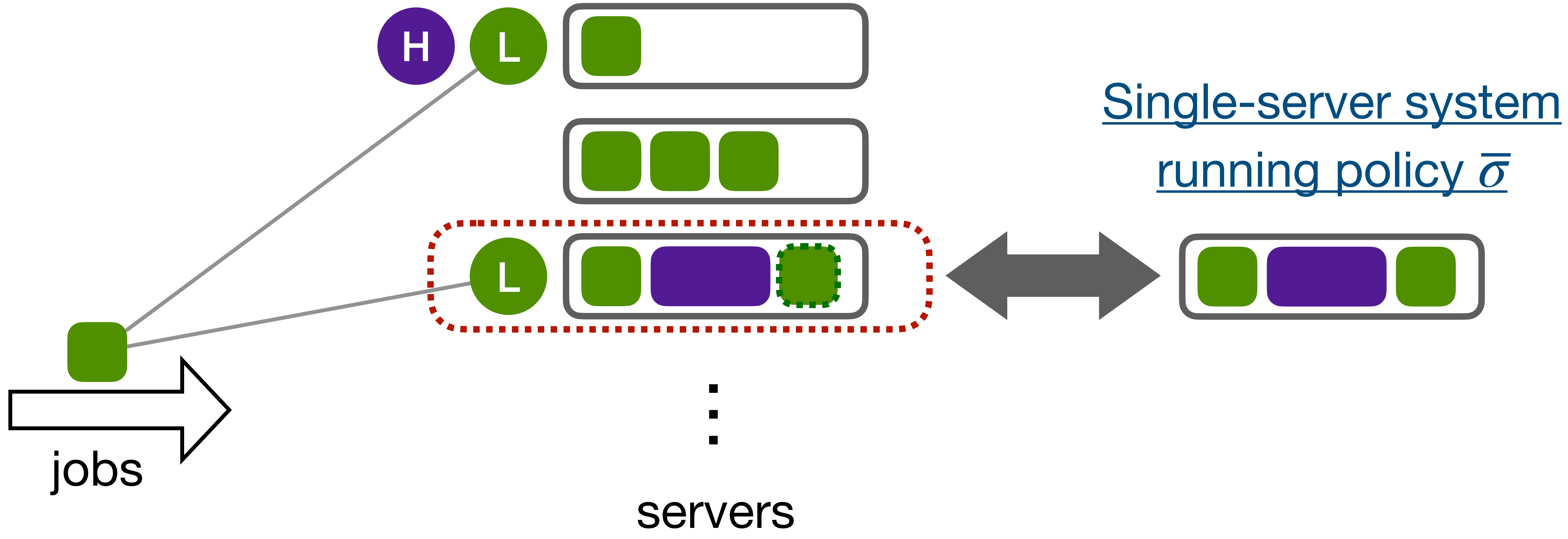


Key proof idea 1



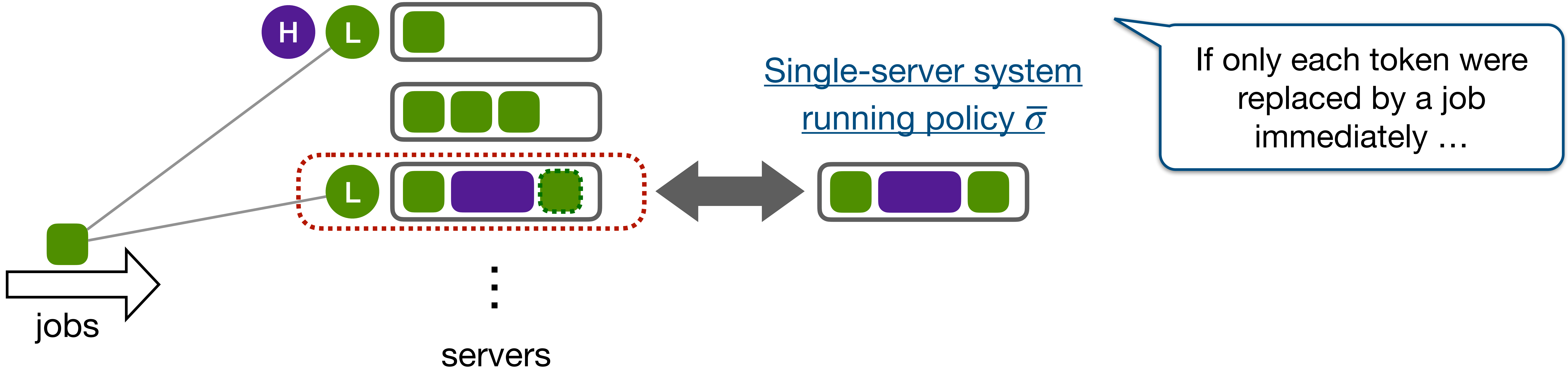
Key proof idea 1

Will show that each server in the original system
≈ an independent single-server system



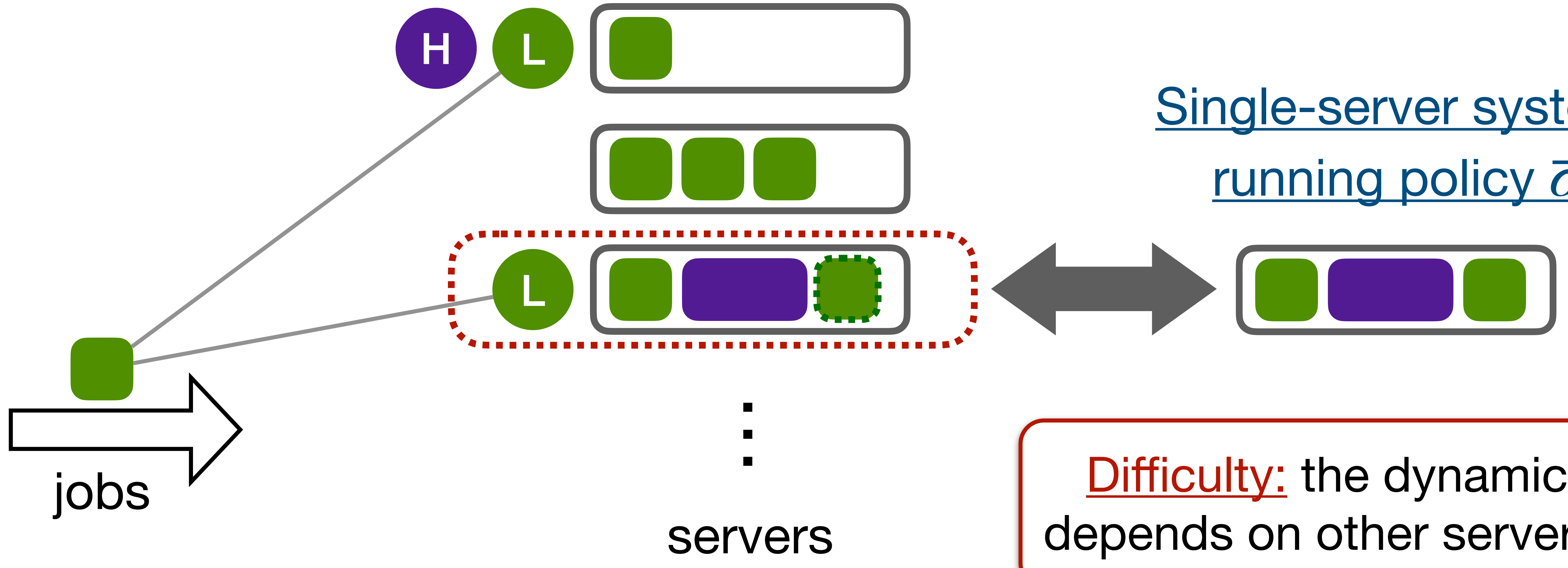
Key proof idea 1

Will show that each server in the original system
≈ an independent single-server system



Key proof idea 1

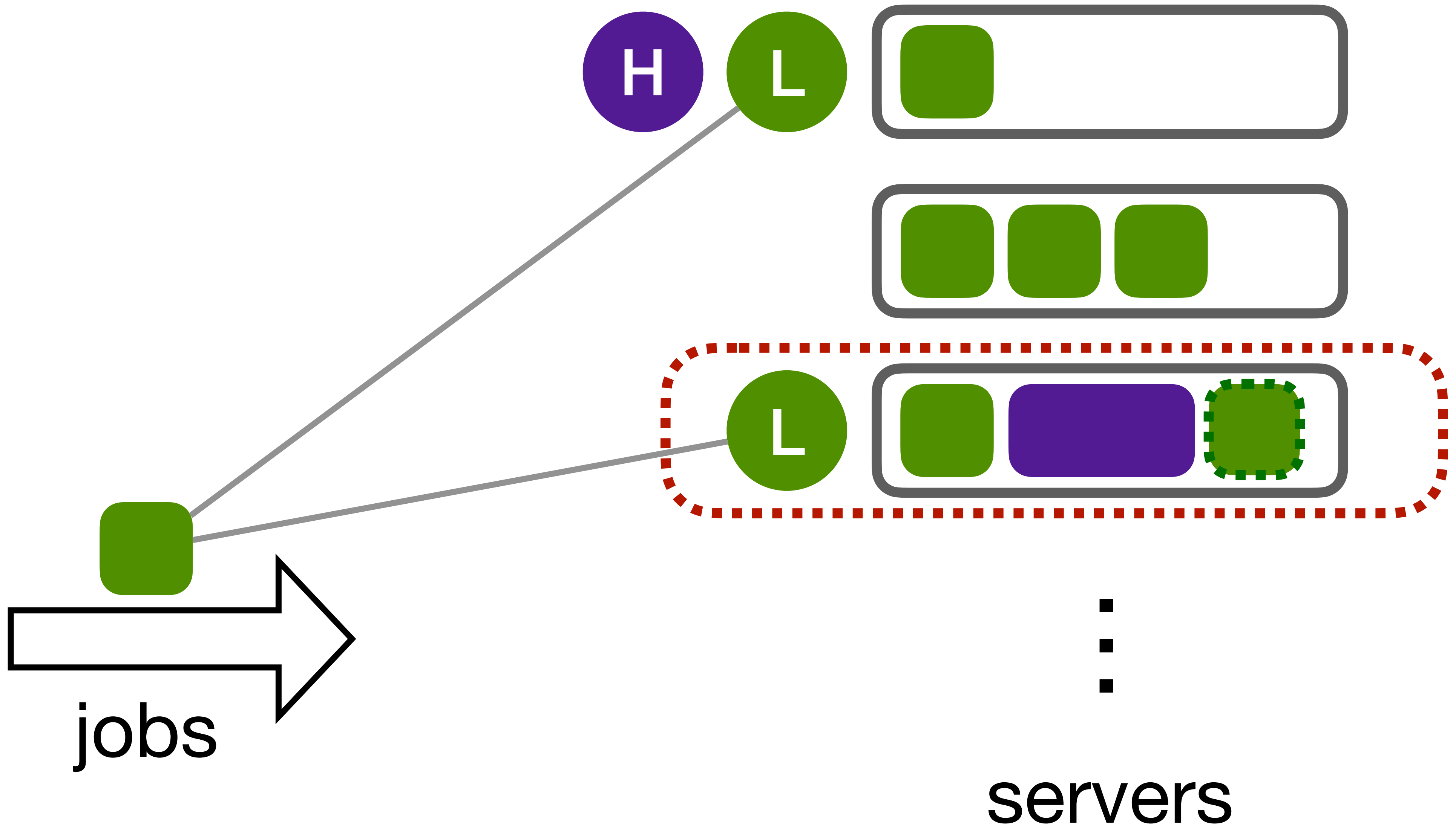
Will show that each server in the original system
≈ an independent single-server system



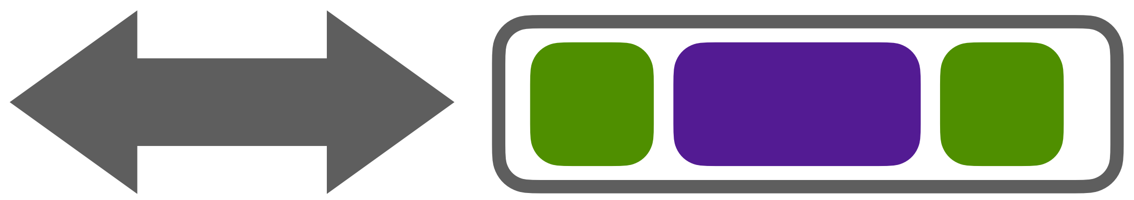
Difficulty: the dynamics of a server in the original system depends on other servers through arrivals & token overflows

Key proof idea 1

Will show that each server in the original system
≈ an independent single-server system



Single-server system
running policy $\bar{\sigma}$



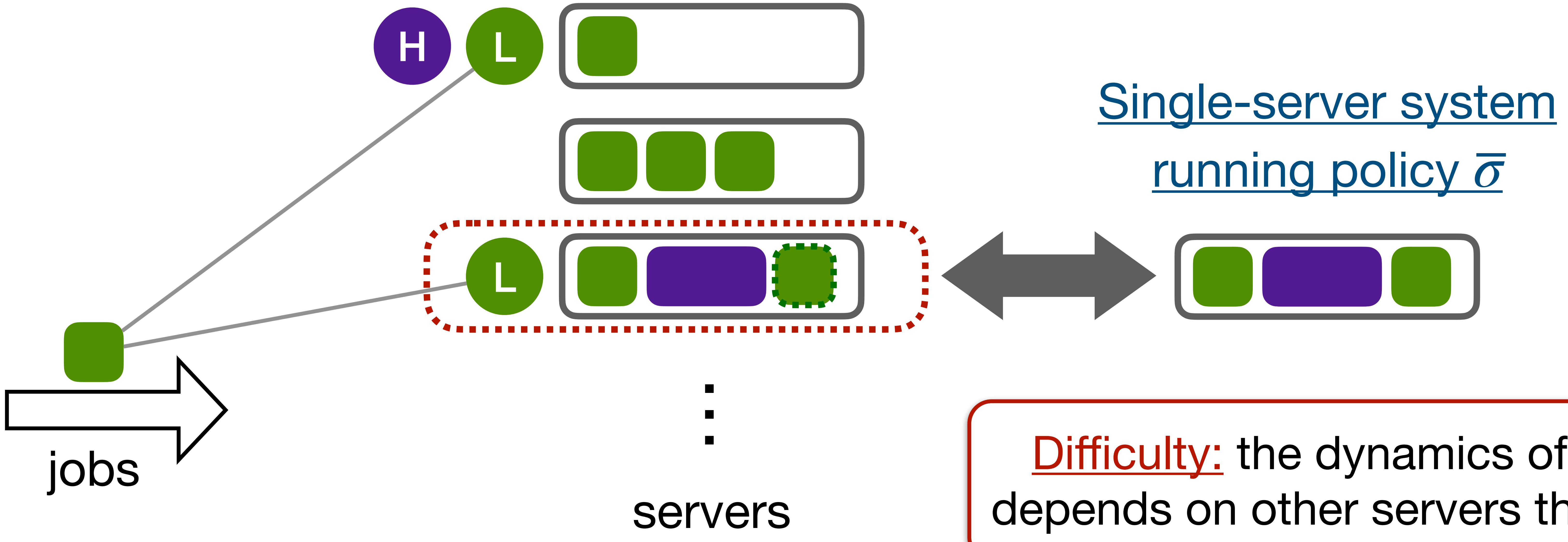
Difficulty: the dynamics of a server in the original system depends on other servers through arrivals & token overflows

Idea: for each type i, consider

$\widetilde{K}_i$ = # jobs + # virtual jobs + # tokens

Key proof idea 1

Will show that each server in the original system
$\approx$ an independent single-server system



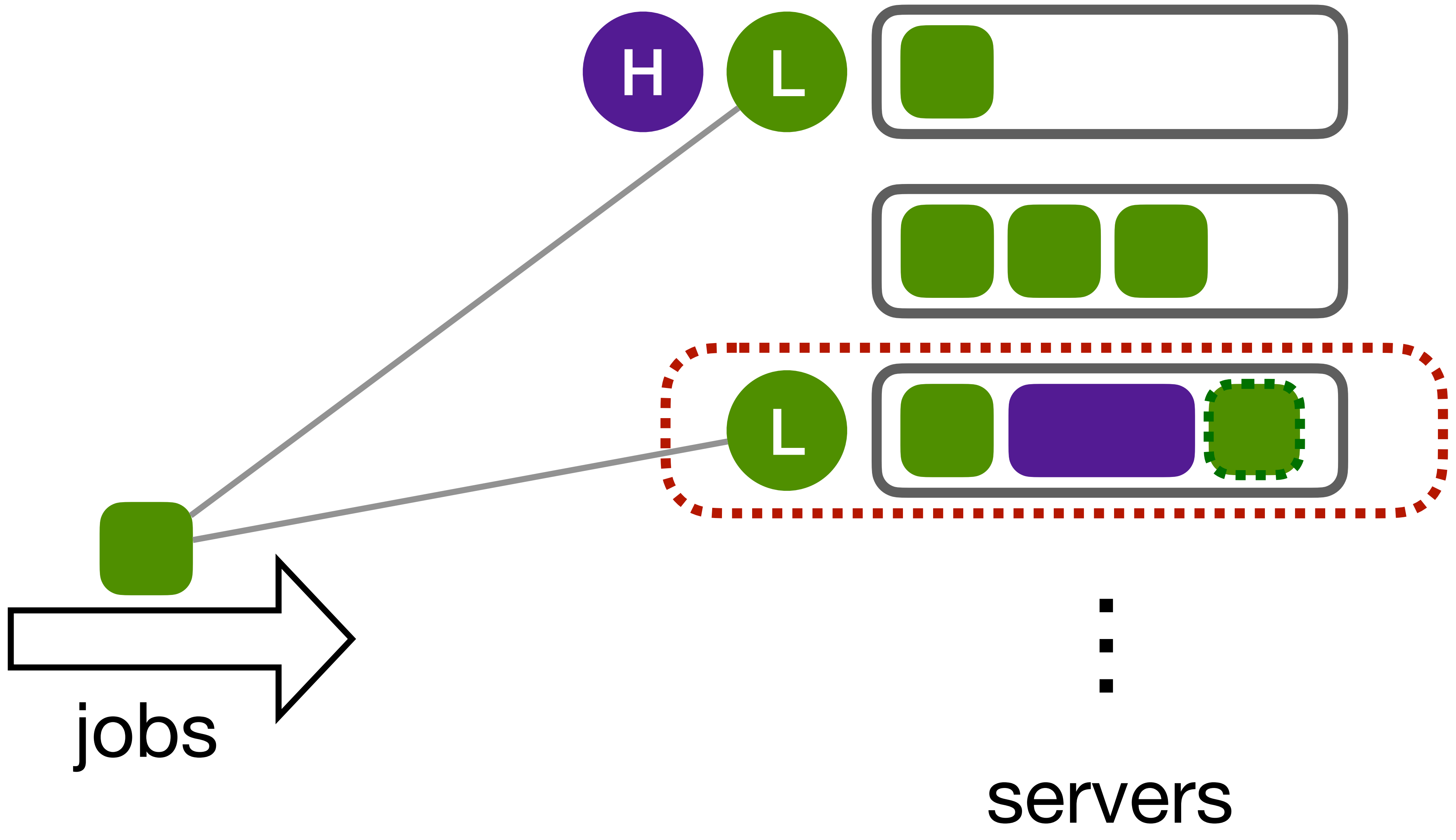
Difficulty: the dynamics of a server in the original system depends on other servers through arrivals & token overflows

Idea: for each type i, consider
$\widetilde{K}_i$ = # jobs + # virtual jobs + # tokens

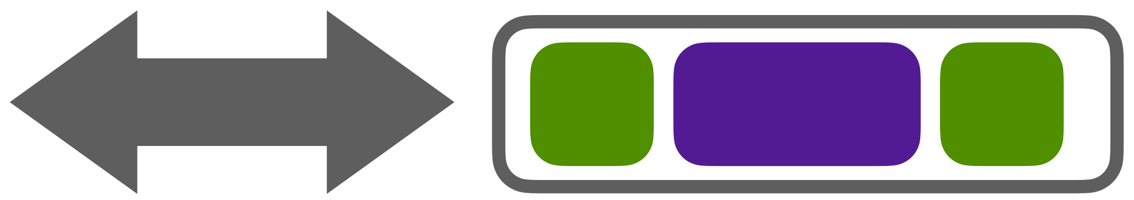
Why does considering $\widetilde{K}_i$ help decouple servers?

Key proof idea 1

Will show that each server in the original system
$\approx$ an independent single-server system



Single-server system
running policy $\bar{\sigma}$



Difficulty: the dynamics of a server in the original system depends on other servers through arrivals & token overflows

Idea: for each type i, consider

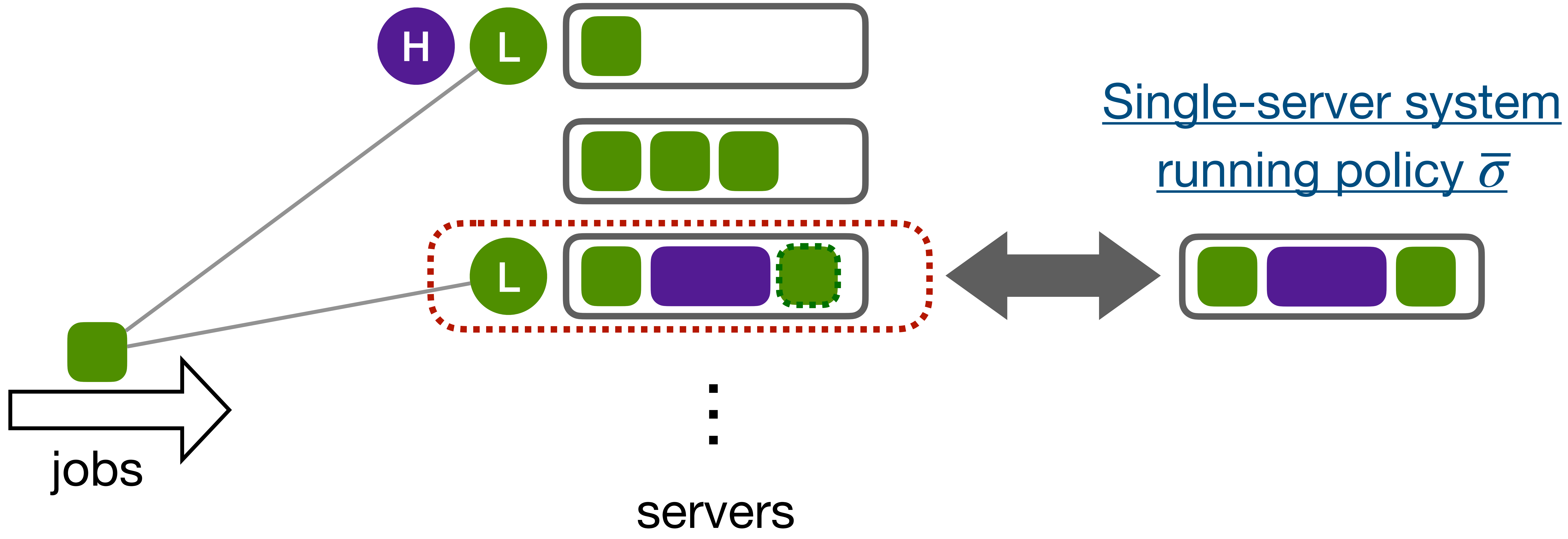
$\widetilde{K}_i$ = # jobs + # virtual jobs + # tokens

- Arrivals & token overflows do not affect $\widetilde{K}_i$

Why does considering $\widetilde{K}_i$ help decouple servers?

Key proof idea 1

Will show that each server in the original system $\approx$ an independent single-server system



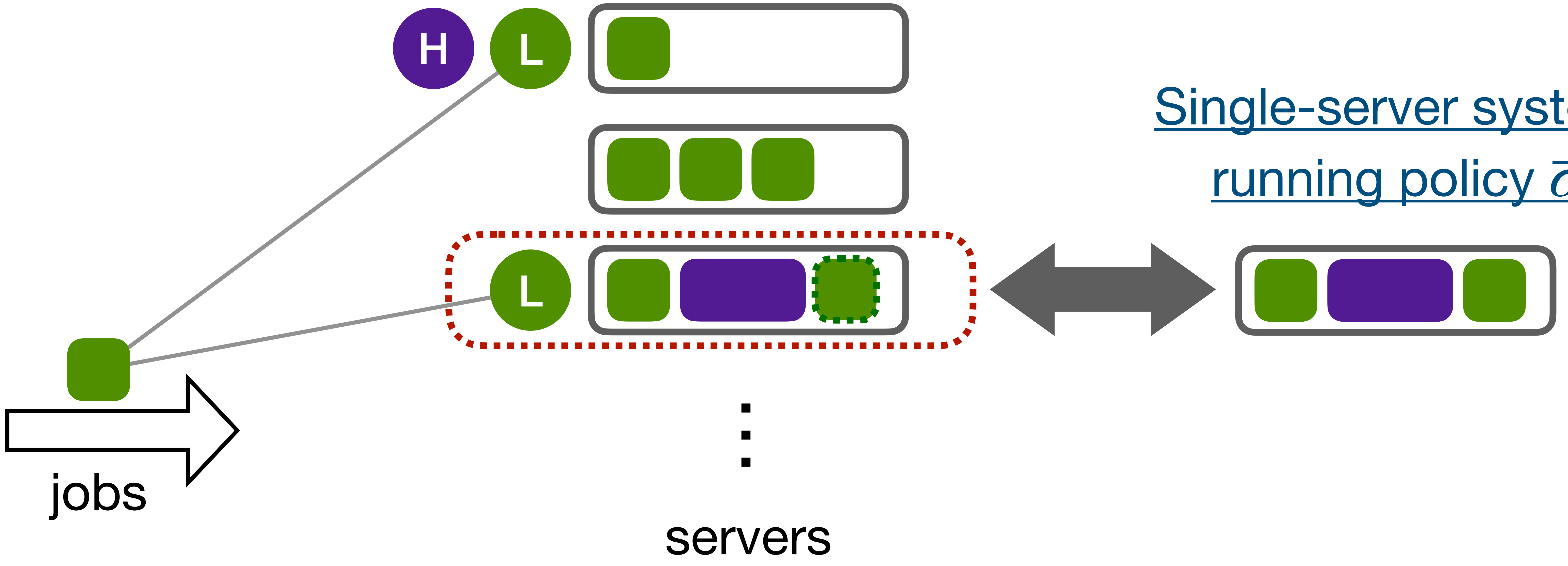
Idea: for each type i, consider

$\widetilde{K}_i$ = # jobs + # virtual jobs + # tokens v.s. $\bar{K}_i$ = # jobs of type i

- Arrivals & token overflows do not affect $\widetilde{K}_i$

Key proof idea 1

Will show that each server in the original system
≈ an independent single-server system



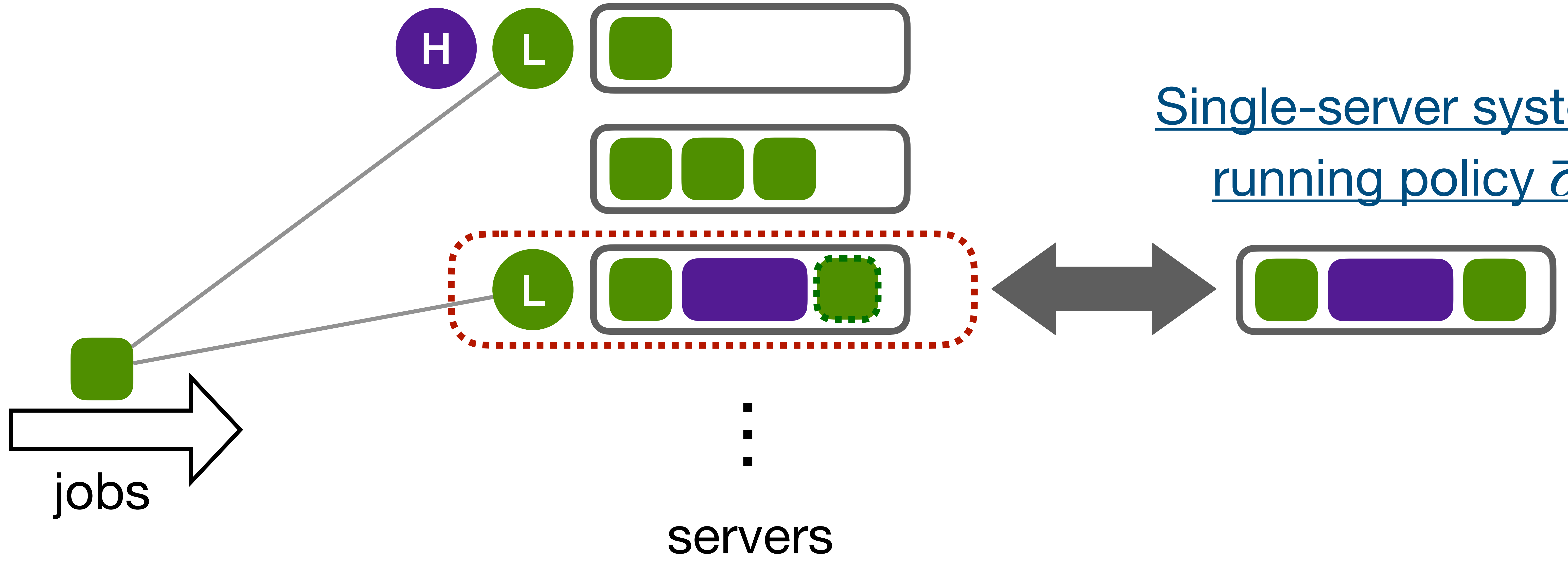
Idea: for each type i, consider

$\widetilde{K}_i$ = # jobs + # virtual jobs + # tokens v.s. $\overline{K}_i$ = # jobs of type i

- Arrivals & token overflows do not affect $\widetilde{K}_i$
- Requests by $\bar{\sigma}$ change $\widetilde{K}_i$ and $\overline{K}_i$ in the same way, difference bounded by # tokens

Key proof idea 1

Will show that each server in the original system
$\approx$ an independent single-server system



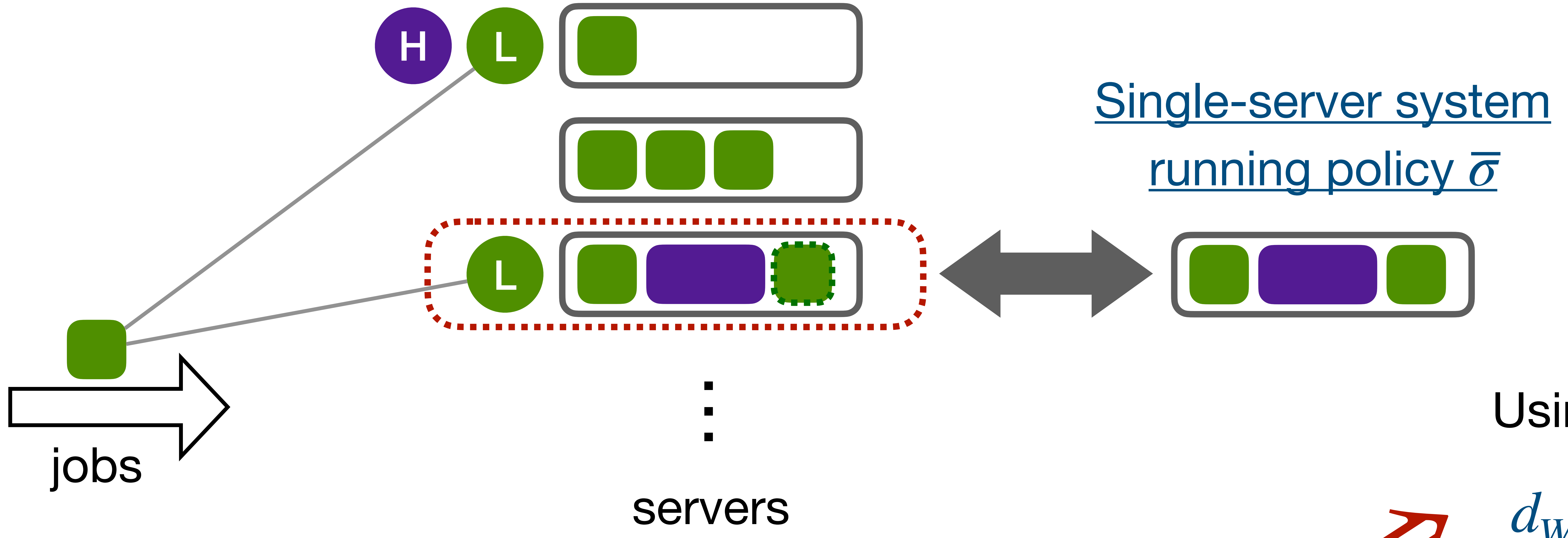
Idea: for each type i, consider

$\widetilde{K}_i$ = # jobs + # virtual jobs + # tokens v.s. $\overline{K}_i$ = # jobs of type i

- Arrivals & token overflows do not affect $\widetilde{K}_i$
- Requests by $\overline{\sigma}$ change $\widetilde{K}_i$ and $\overline{K}_i$ in the same way, difference bounded by # tokens
- Job phase transitions in $\widetilde{K}_i$ and $\overline{K}_i$ differ by # tokens

Key proof idea 1

Will show that each server in the original system
$\approx$ an independent single-server system



Using Stein's method, we show

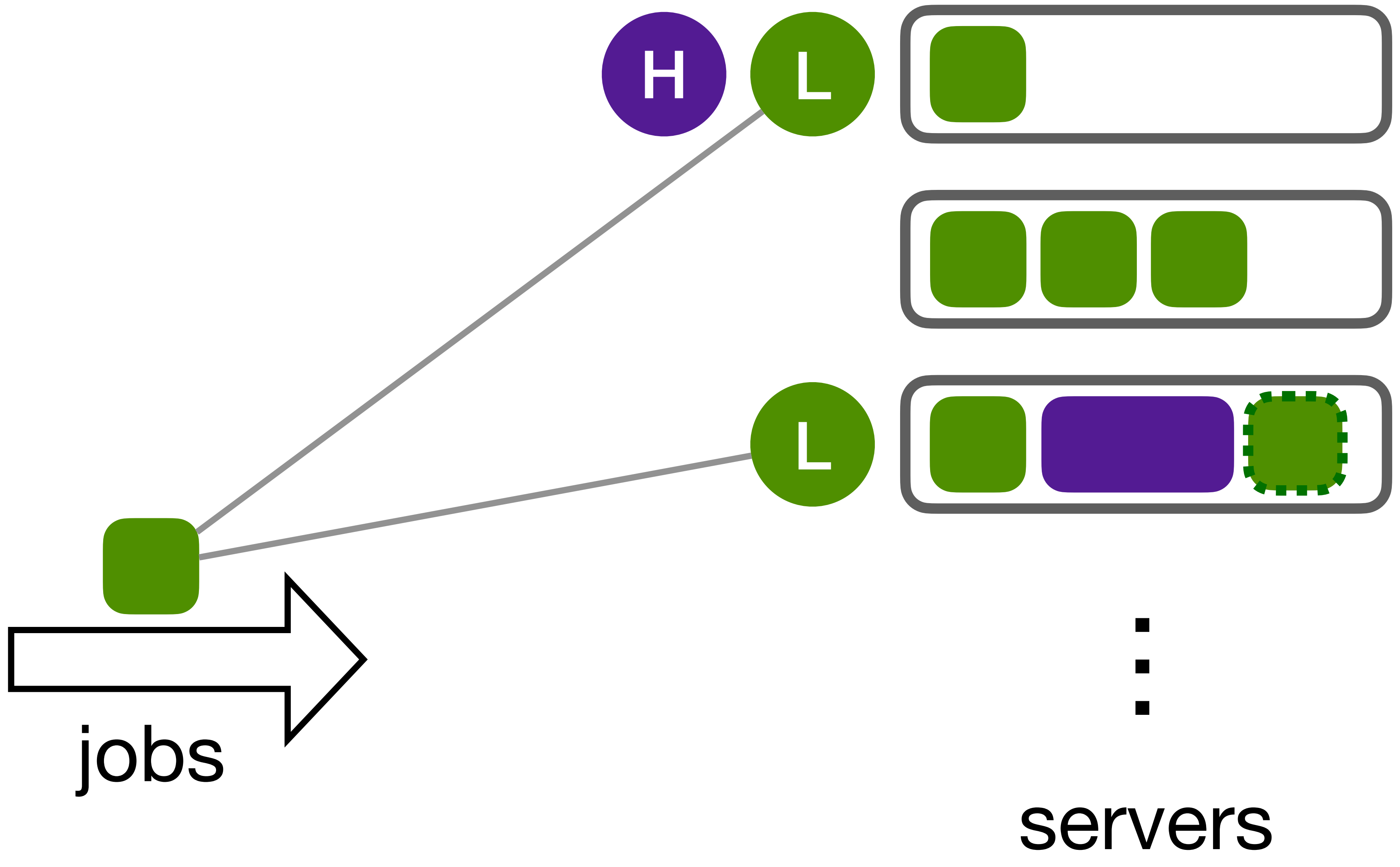
$$d_W\left(\widetilde{K}^{1:\bar{N}}, \bar{K}^{1:\bar{N}}\right) = O\left(r^{0.5}\right)$$

Idea: for each type i, consider

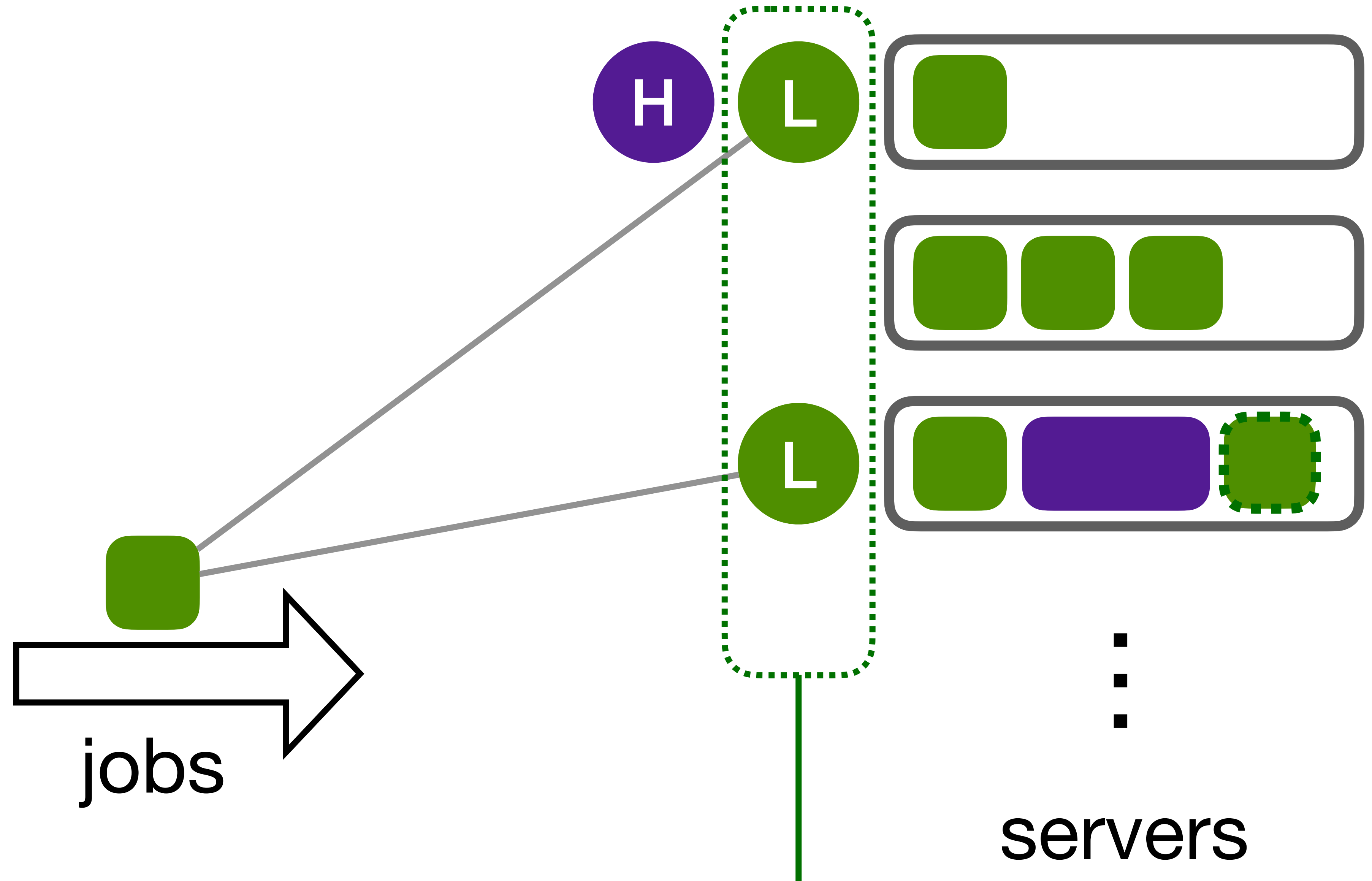
$\widetilde{K}_i$ = # jobs + # virtual jobs + # tokens v.s. $\bar{K}_i$ = # jobs of type i

- Arrivals & token overflows do not affect $\widetilde{K}_i$
- Requests by $\bar{\sigma}$ change $\widetilde{K}_i$ and $\bar{K}_i$ in the same way, difference bounded by # tokens
- Job phase transitions in $\widetilde{K}_i$ and $\bar{K}_i$ differ by # tokens

Key proof idea 2

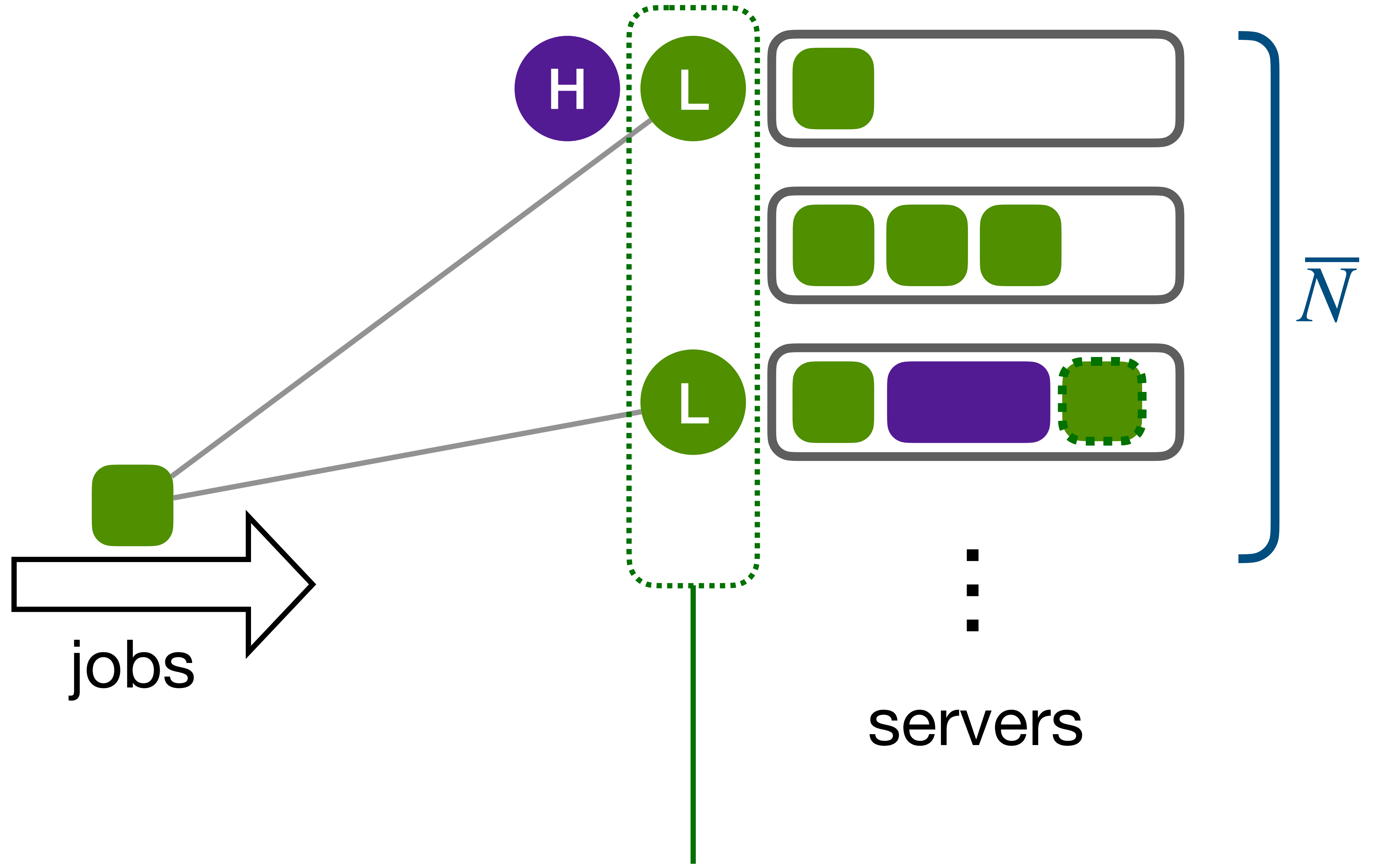


Key proof idea 2



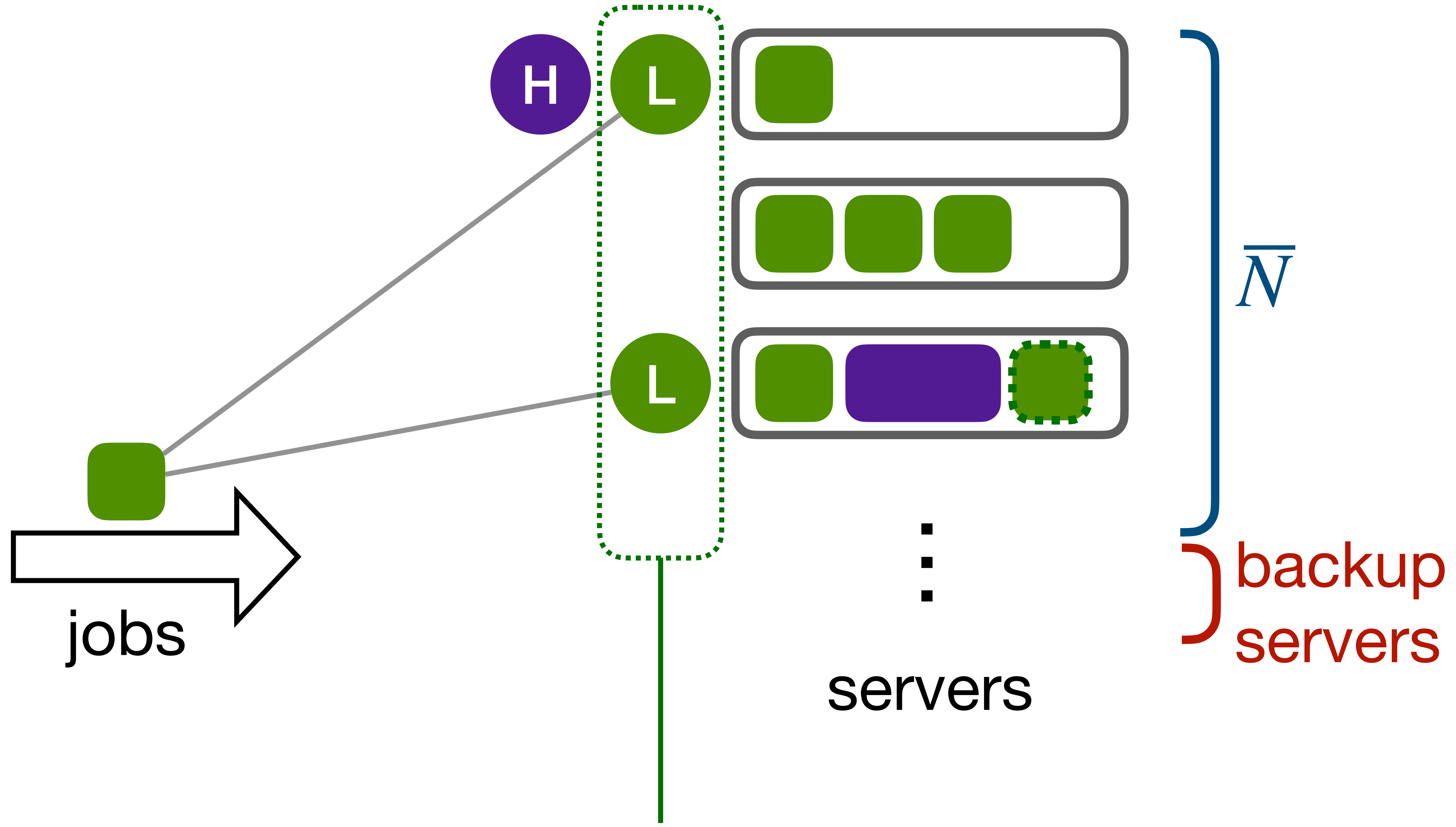
When the # tokens of a type $> \sqrt{r}$, remove the overflow tokens and generate virtual jobs

Key proof idea 2



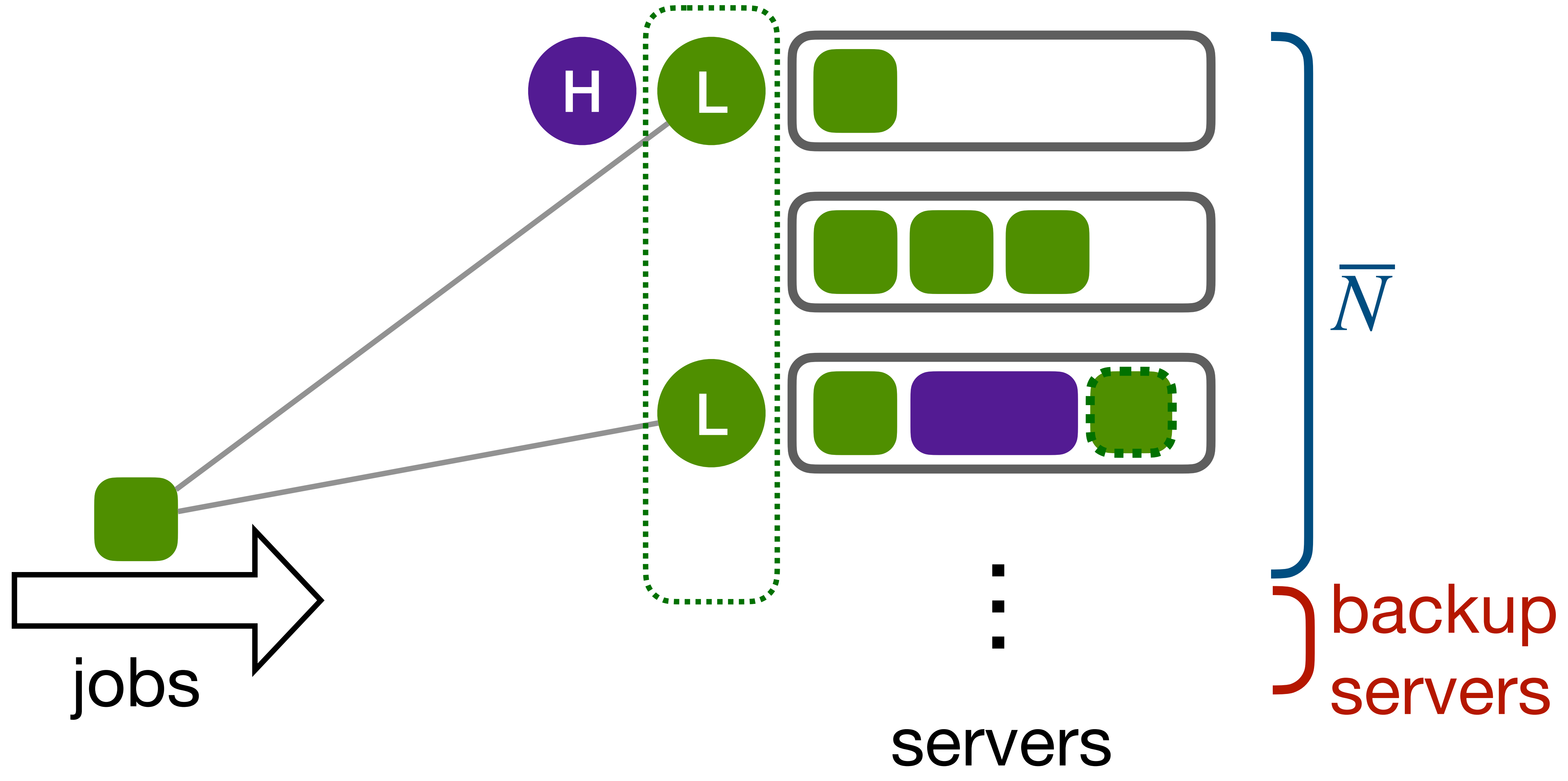
When the # tokens of a type $> \sqrt{r}$, remove the overflow tokens and generate virtual jobs

Key proof idea 2



When the # tokens of a type $> \sqrt{r}$, remove the overflow tokens and generate virtual jobs

Key proof idea 2



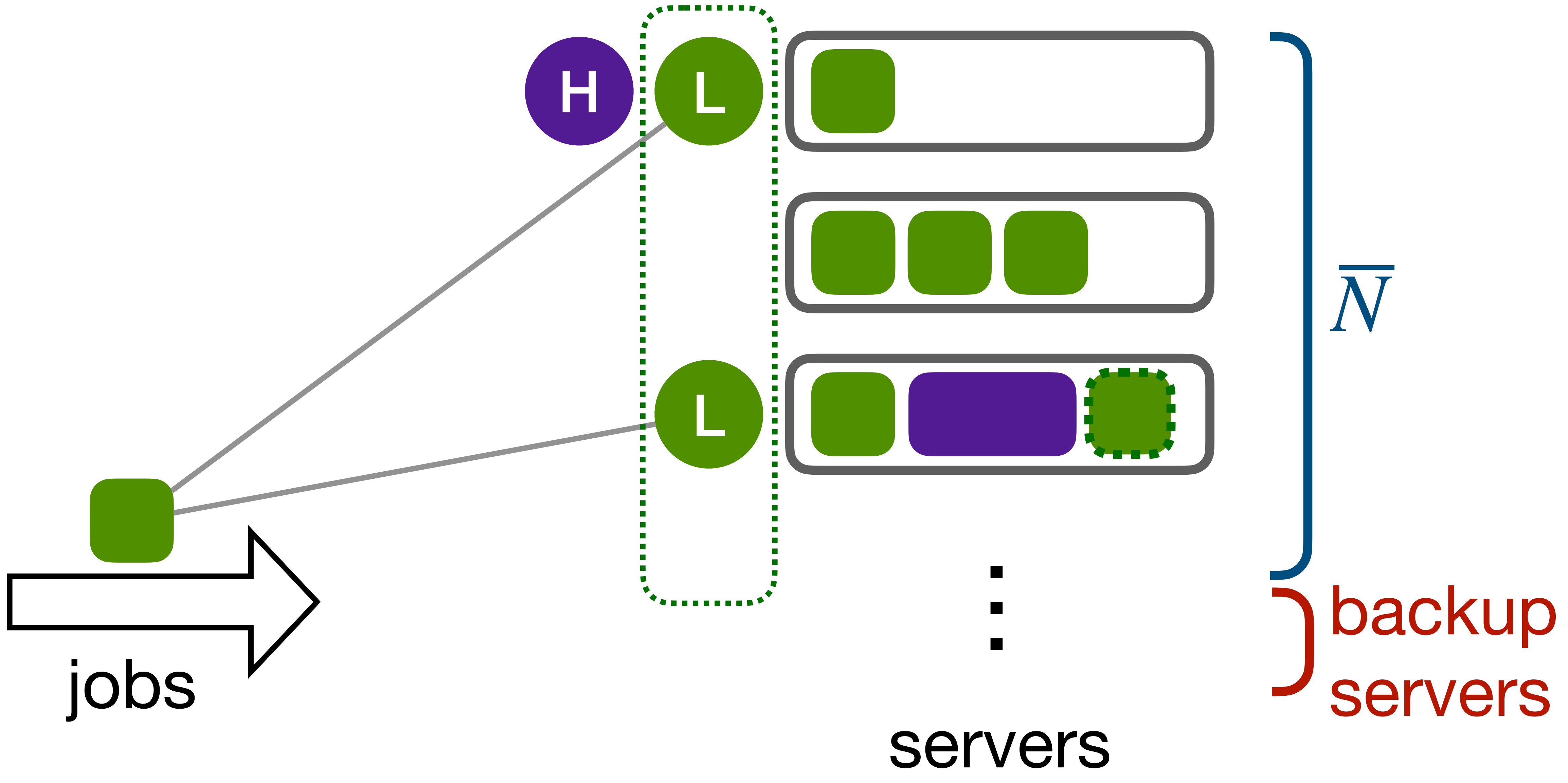
Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

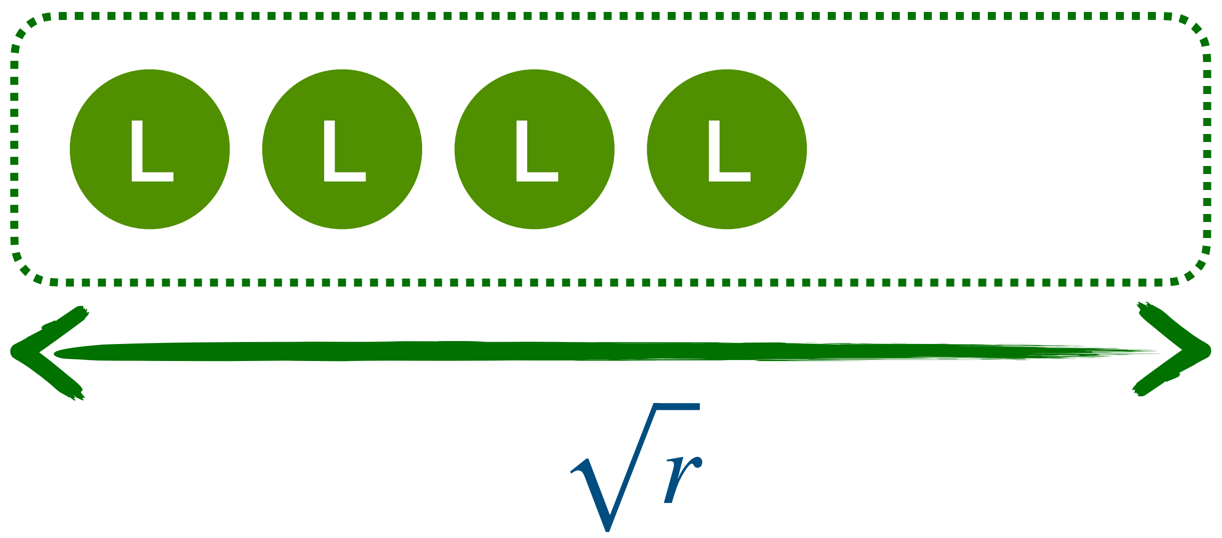
and # backup servers $= O\left(\sqrt{r}\right)$

Key proof idea 2

Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

and # backup servers $= O\left(\sqrt{r}\right)$

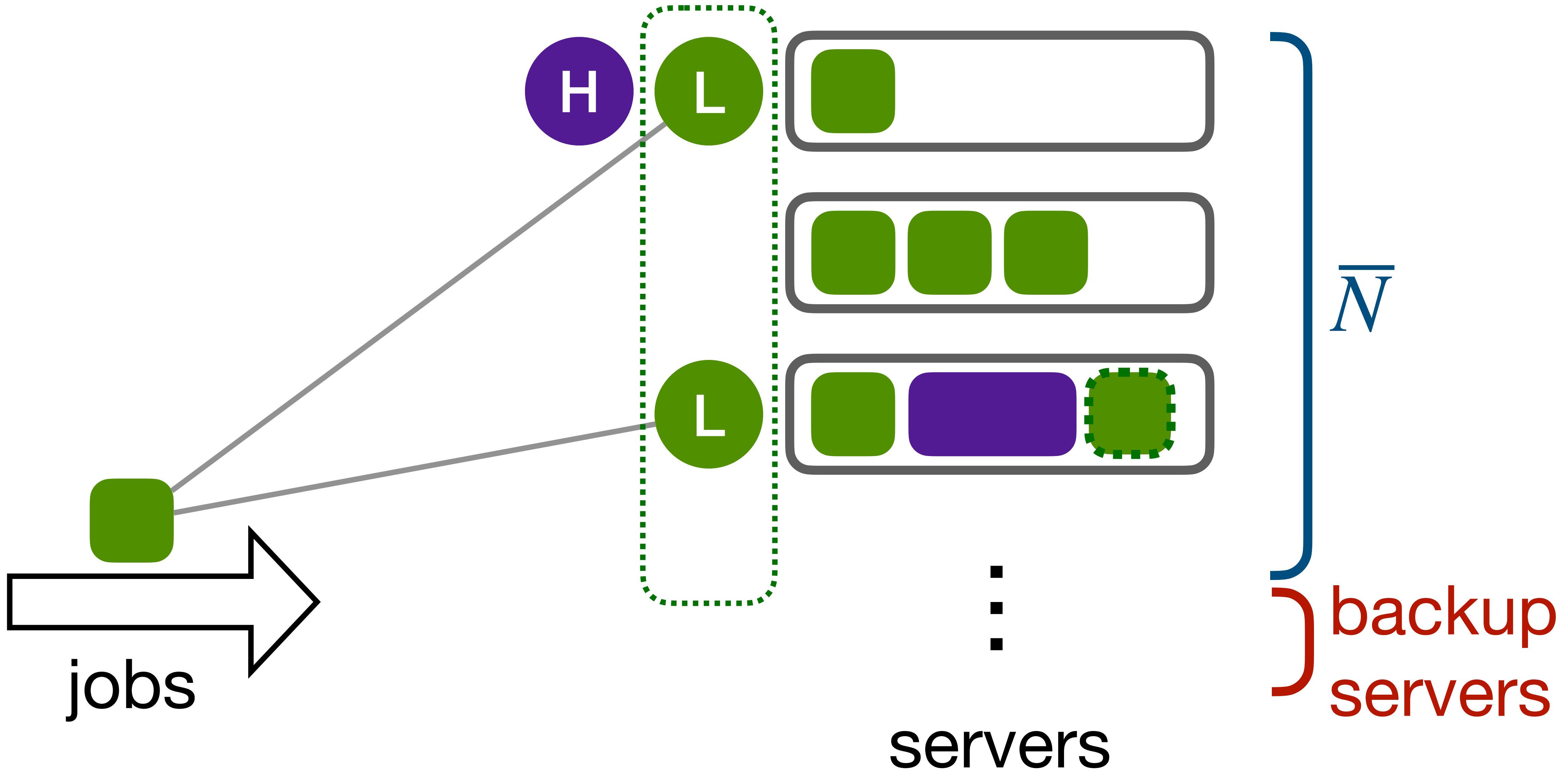


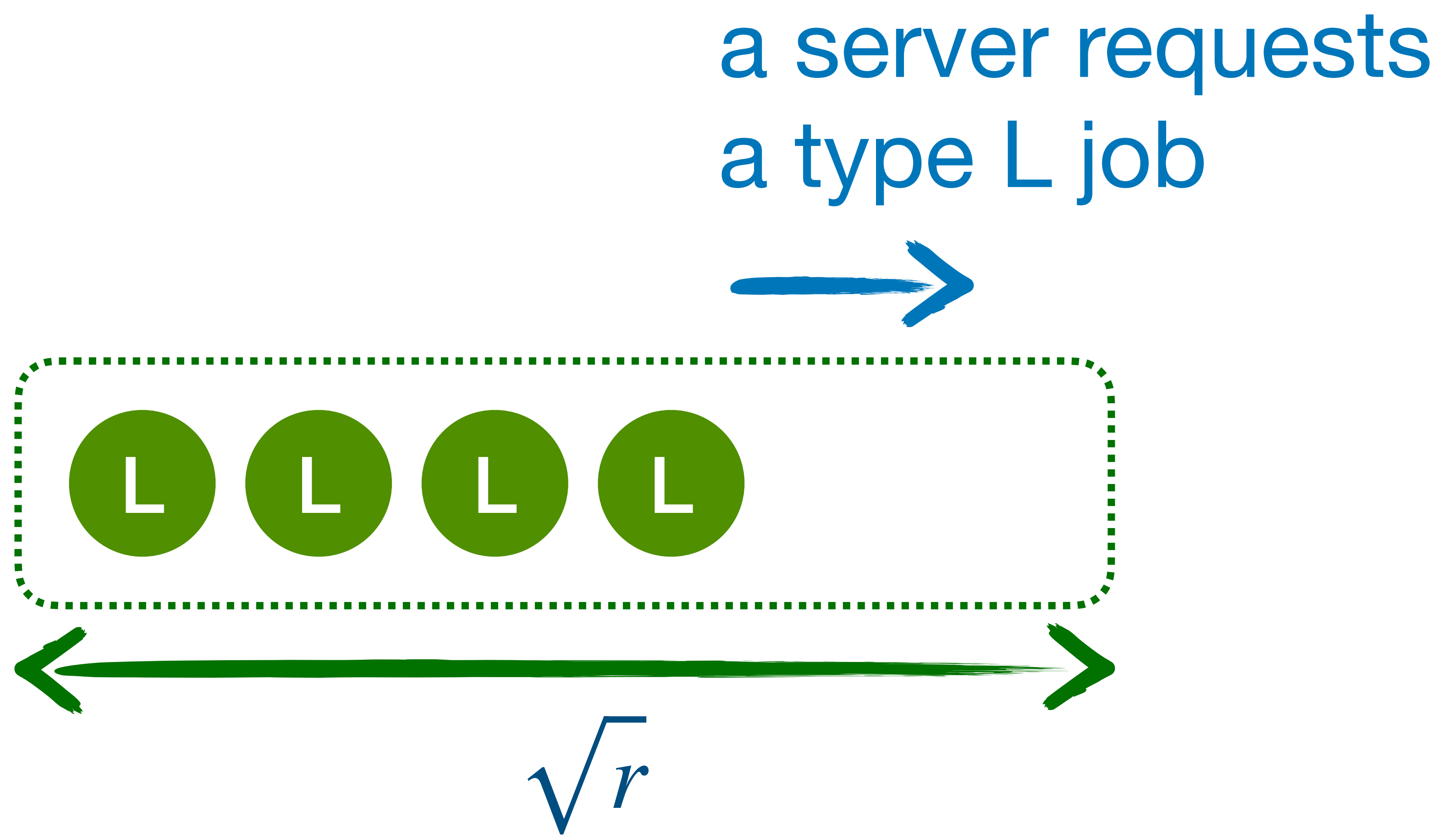


Key proof idea 2

Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

and # backup servers $= O\left(\sqrt{r}\right)$

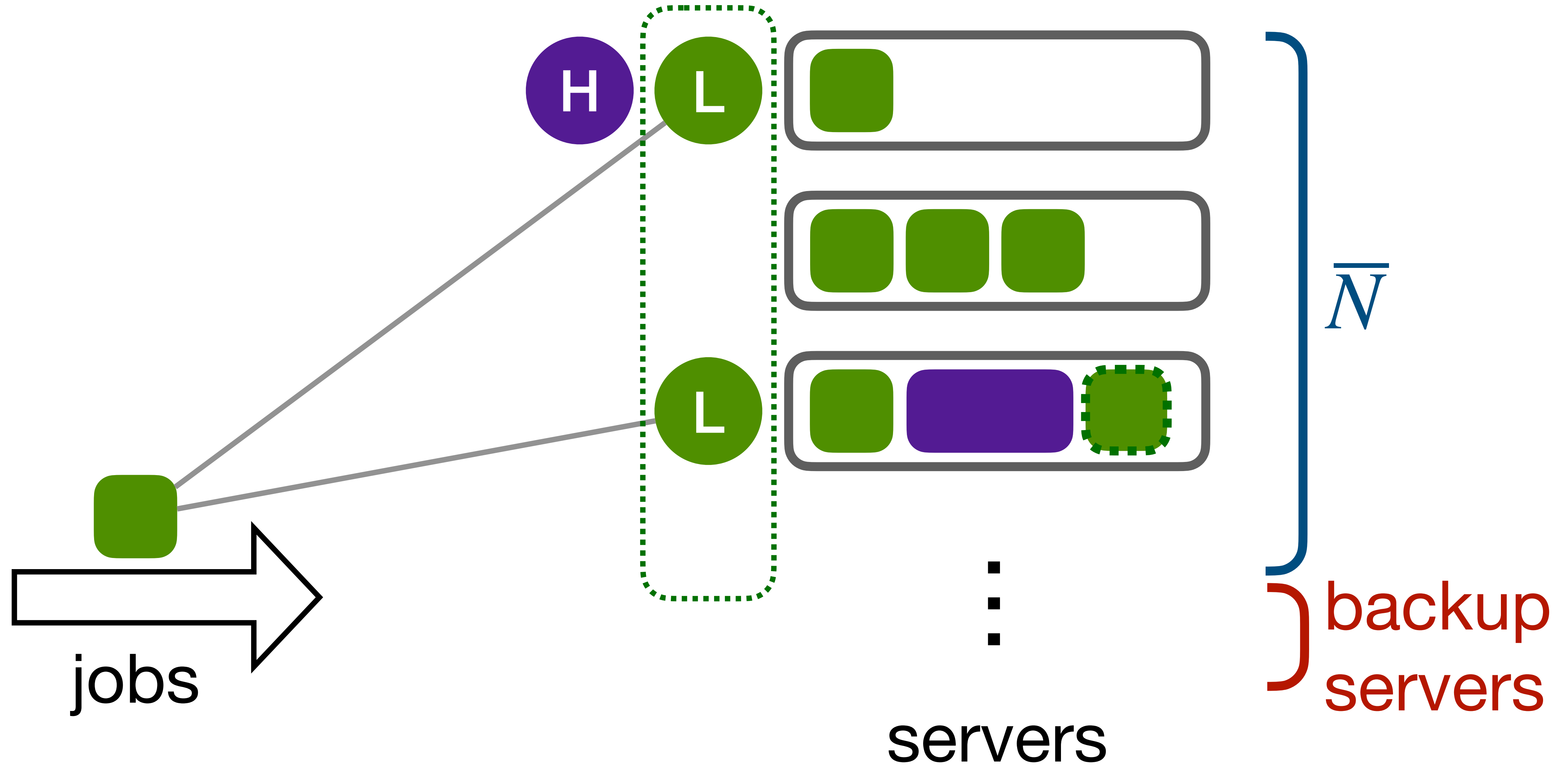


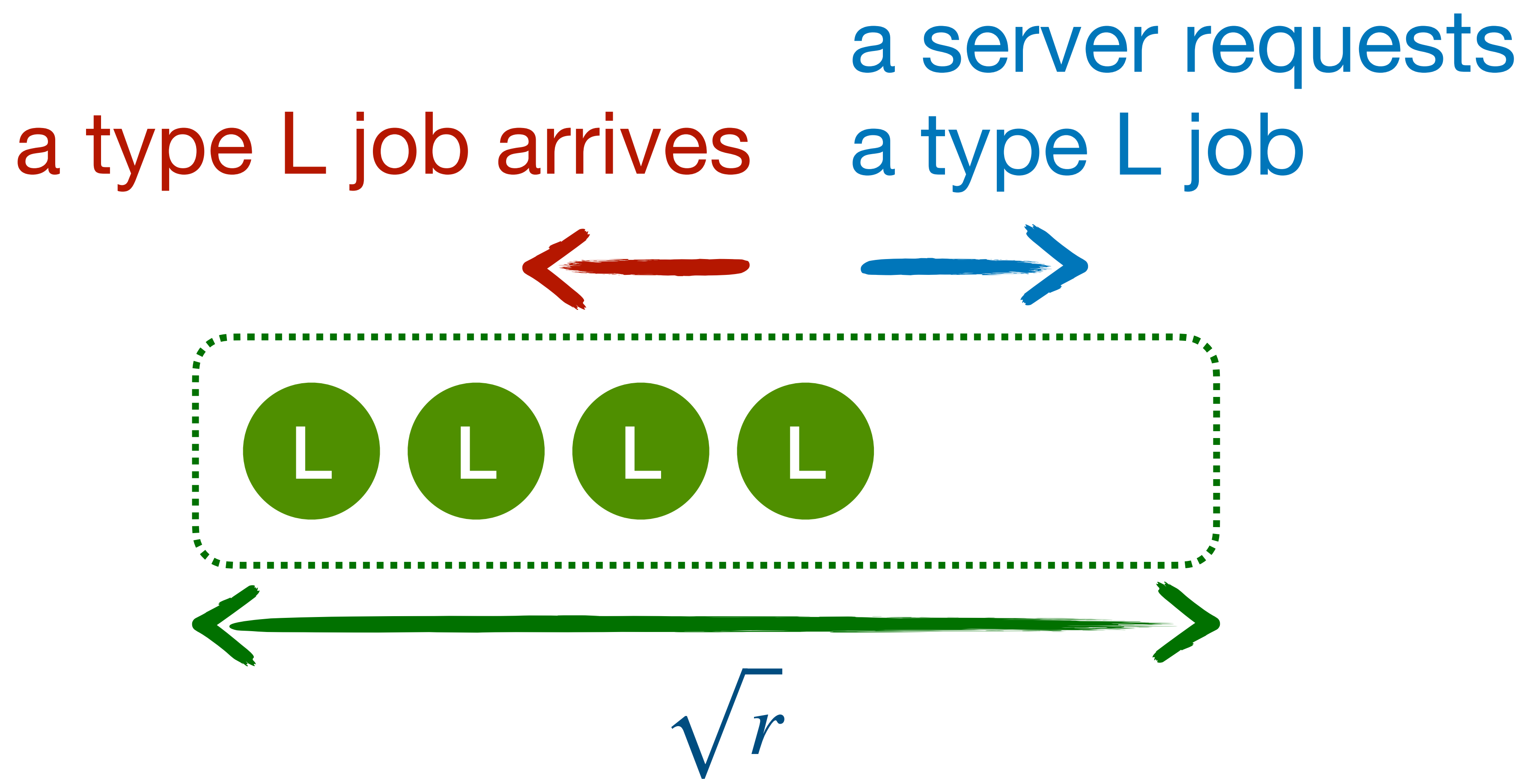


Key proof idea 2

Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

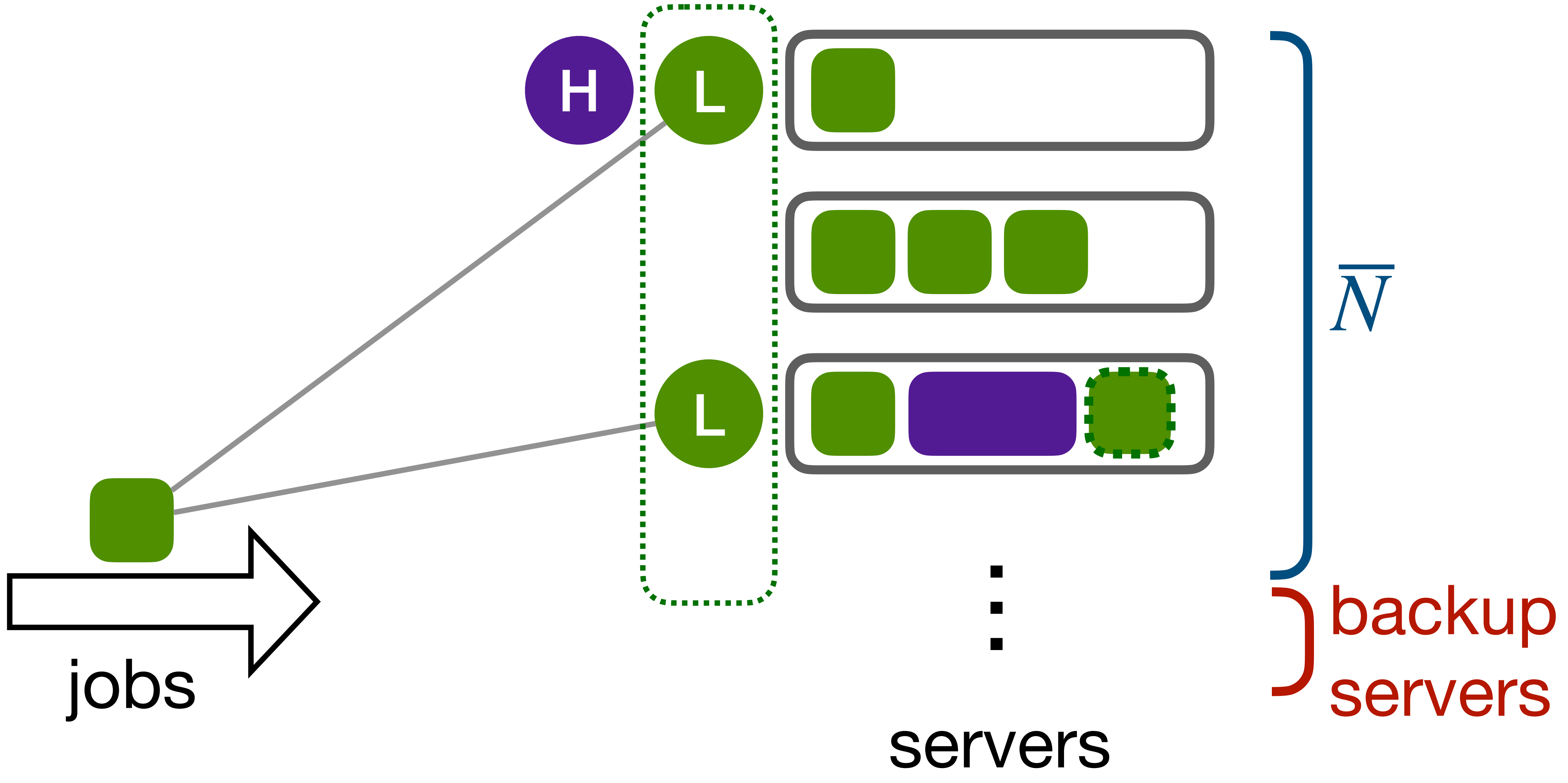
and # backup servers $= O\left(\sqrt{r}\right)$

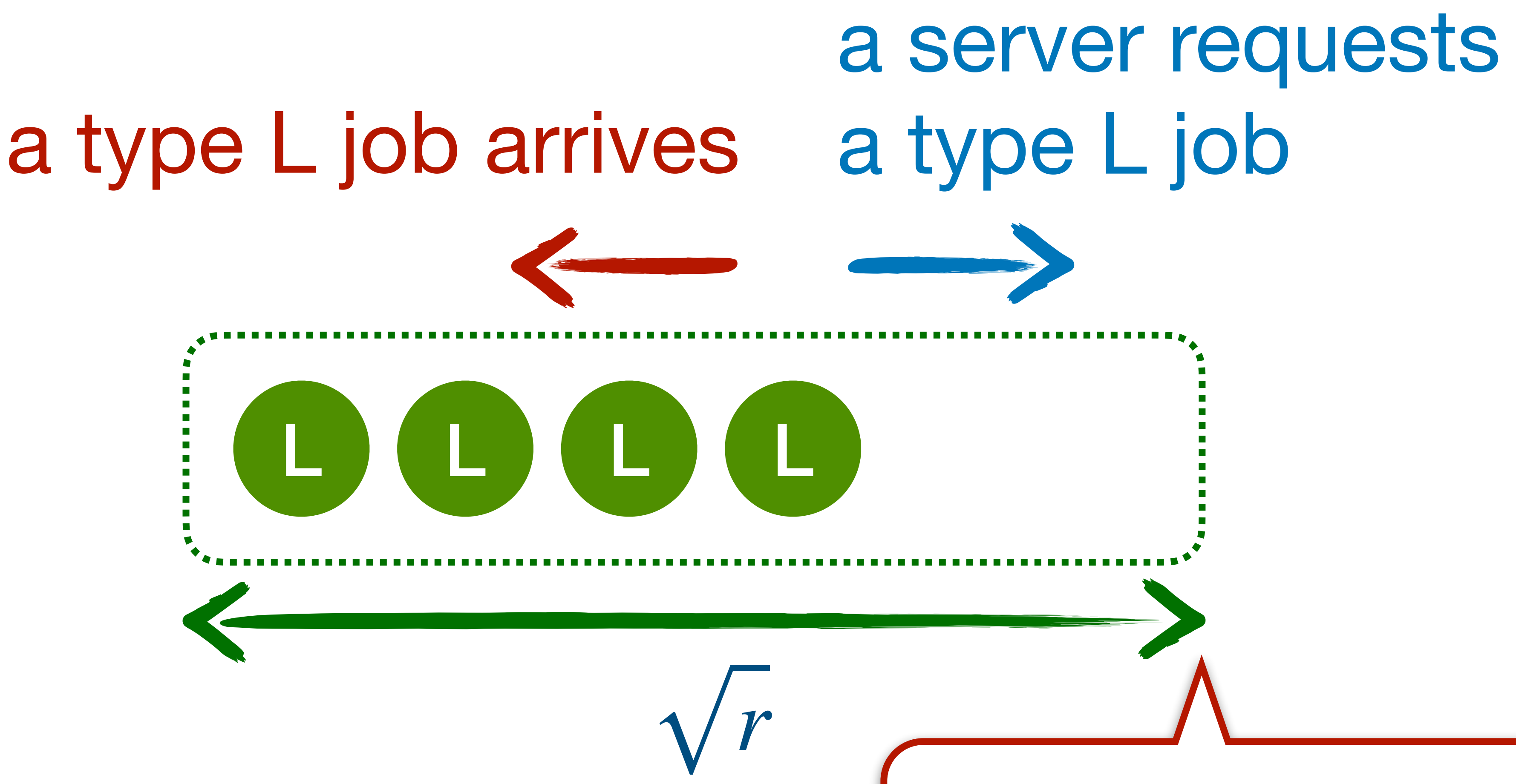




Key proof idea 2

Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,
and # backup servers $= O\left(\sqrt{r}\right)$



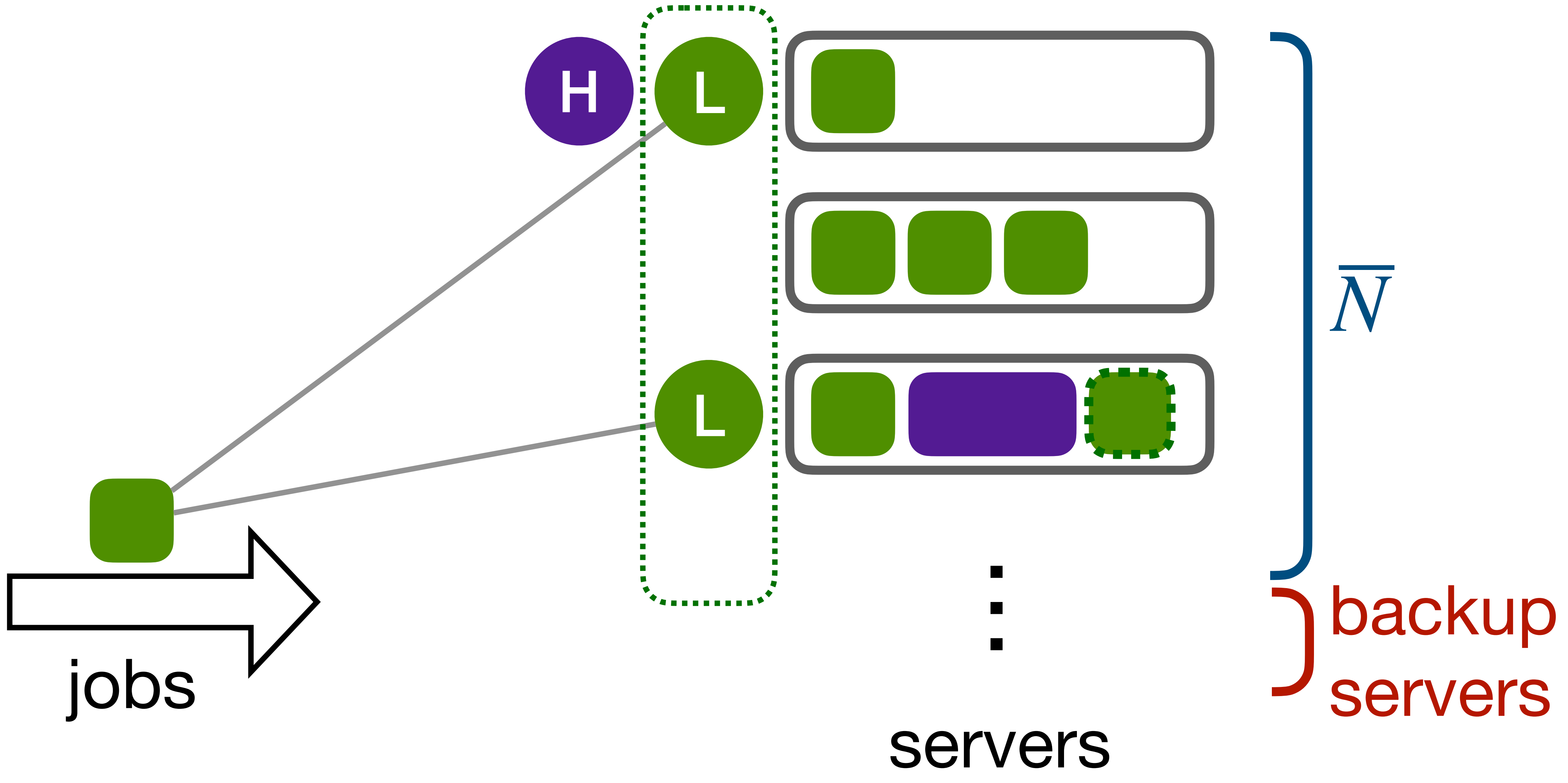


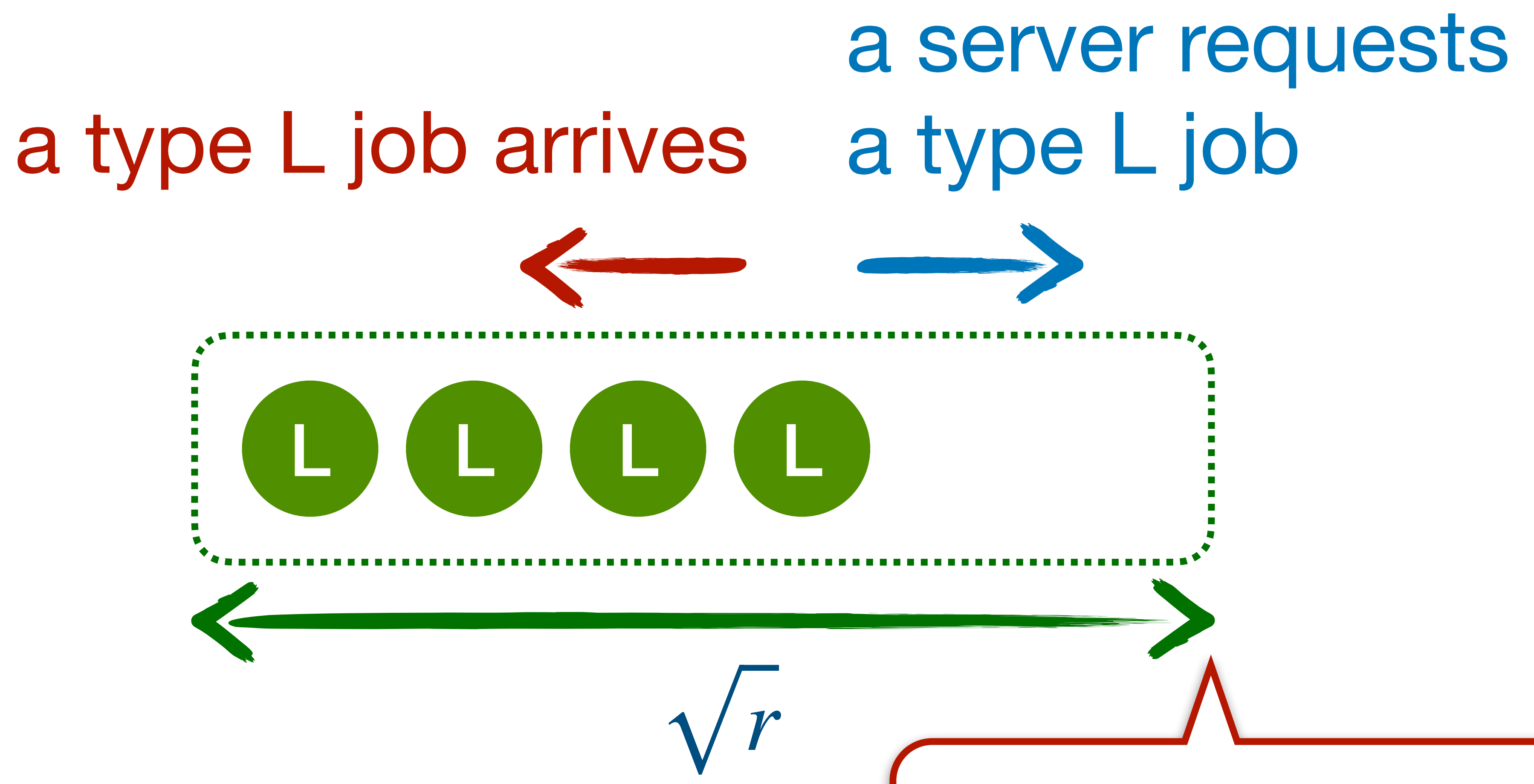
What happens when # tokens hits $\sqrt{r}$?

Key proof idea 2

Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

and # backup servers $= O\left(\sqrt{r}\right)$





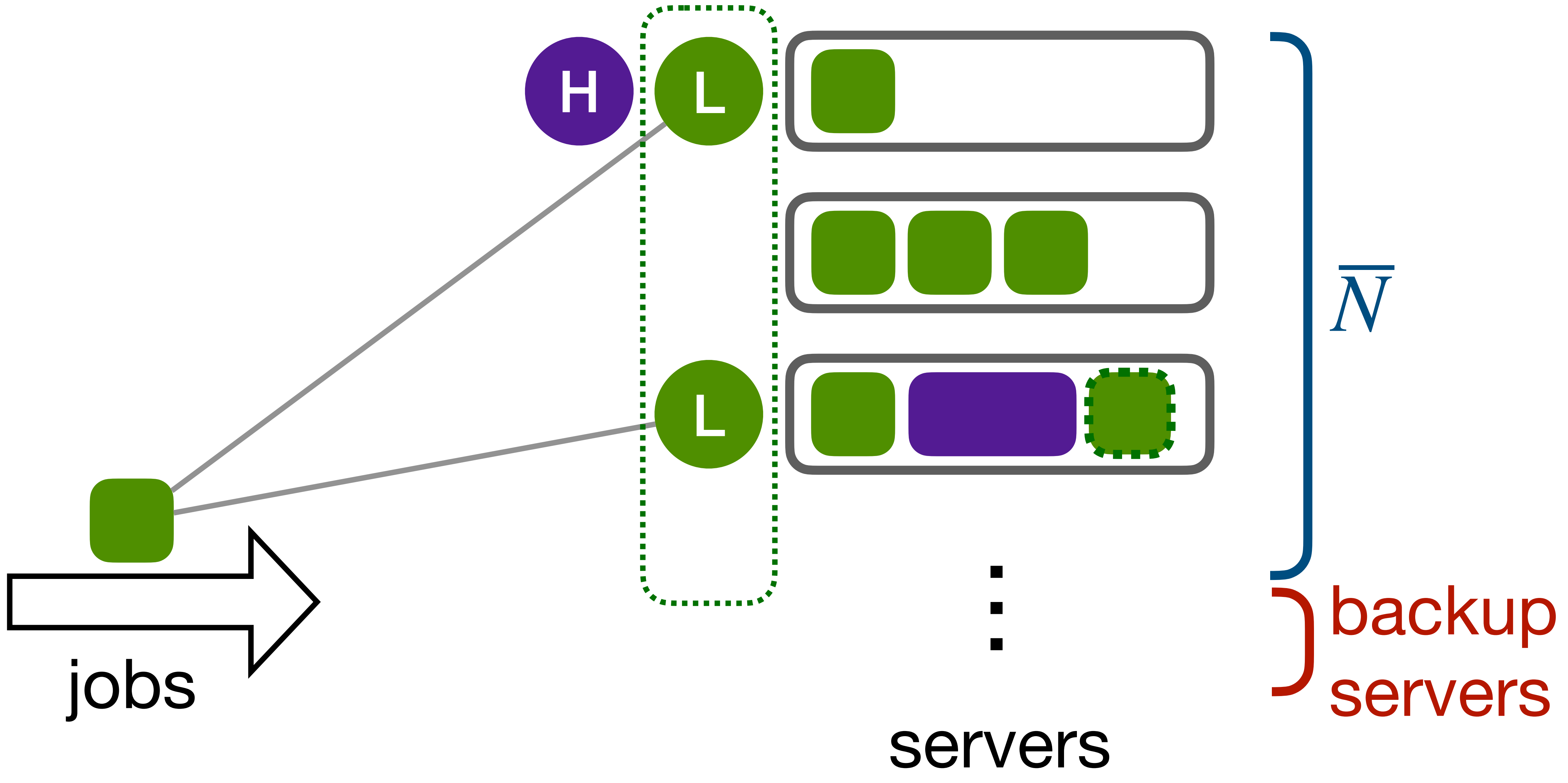
What happens when # tokens hits $\sqrt{r}$?

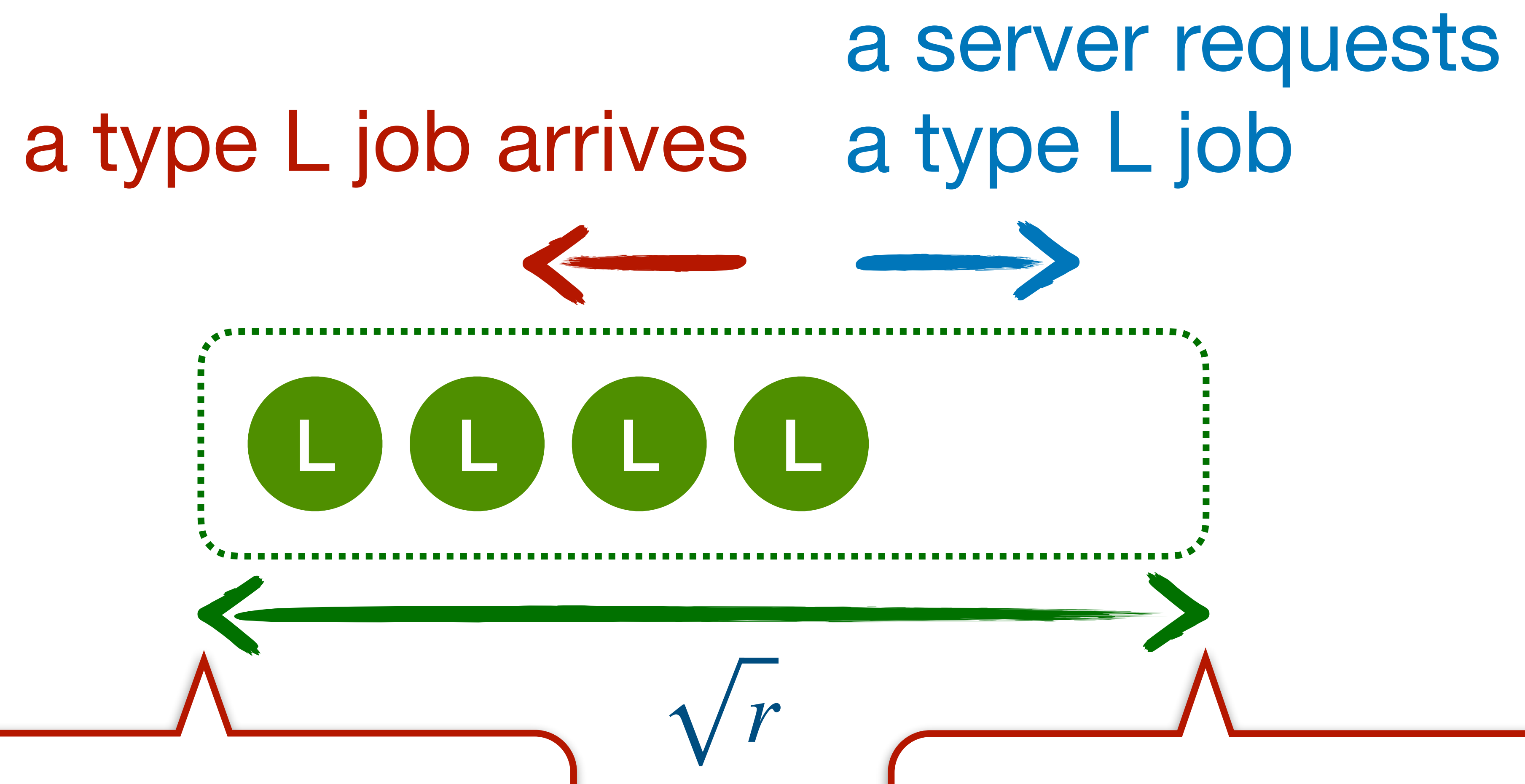
Generate a virtual job

Key proof idea 2

Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

and # backup servers $= O\left(\sqrt{r}\right)$



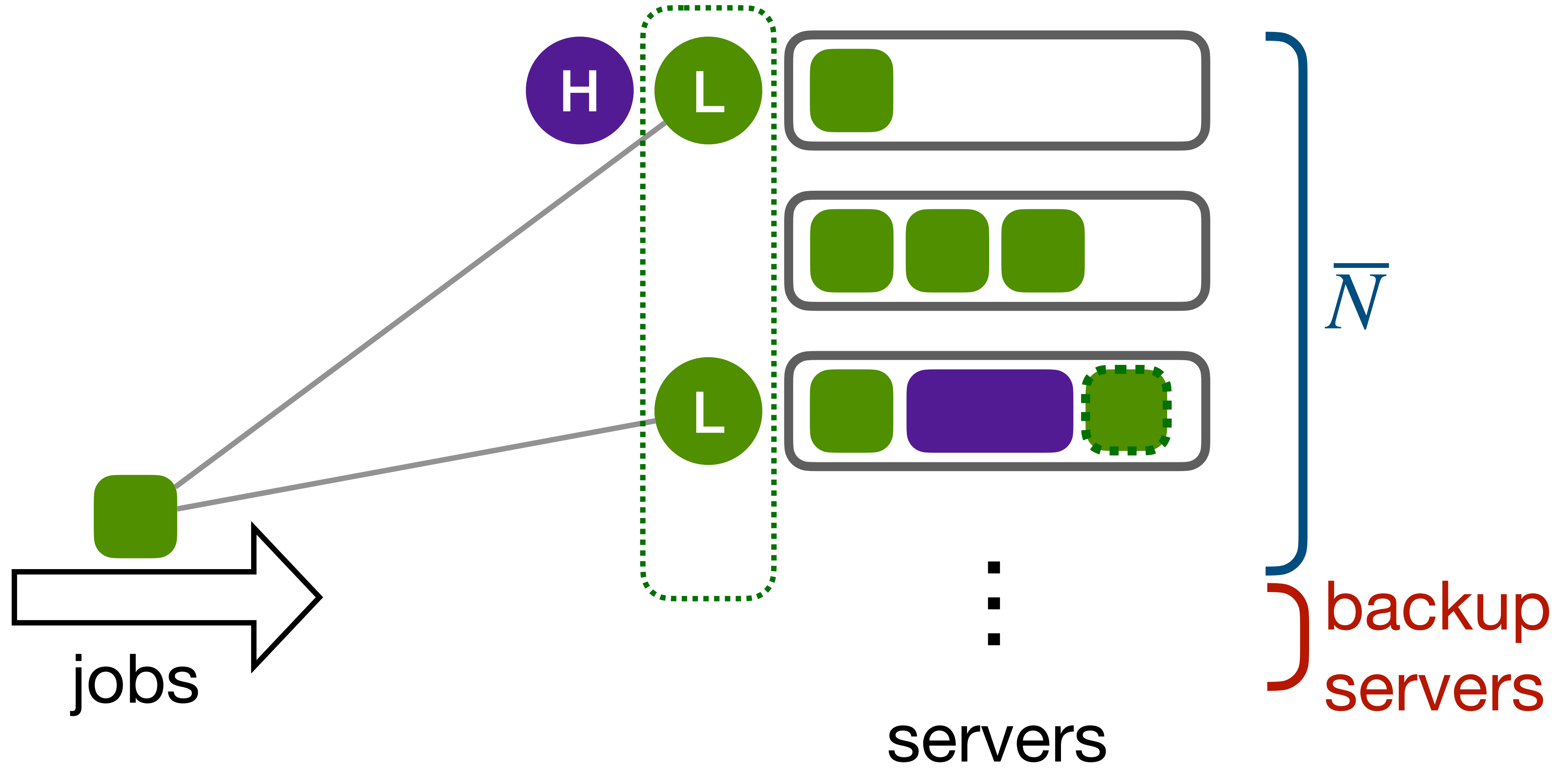


What happens when # tokens hits 0?

What happens when # tokens hits $\sqrt{r}$?

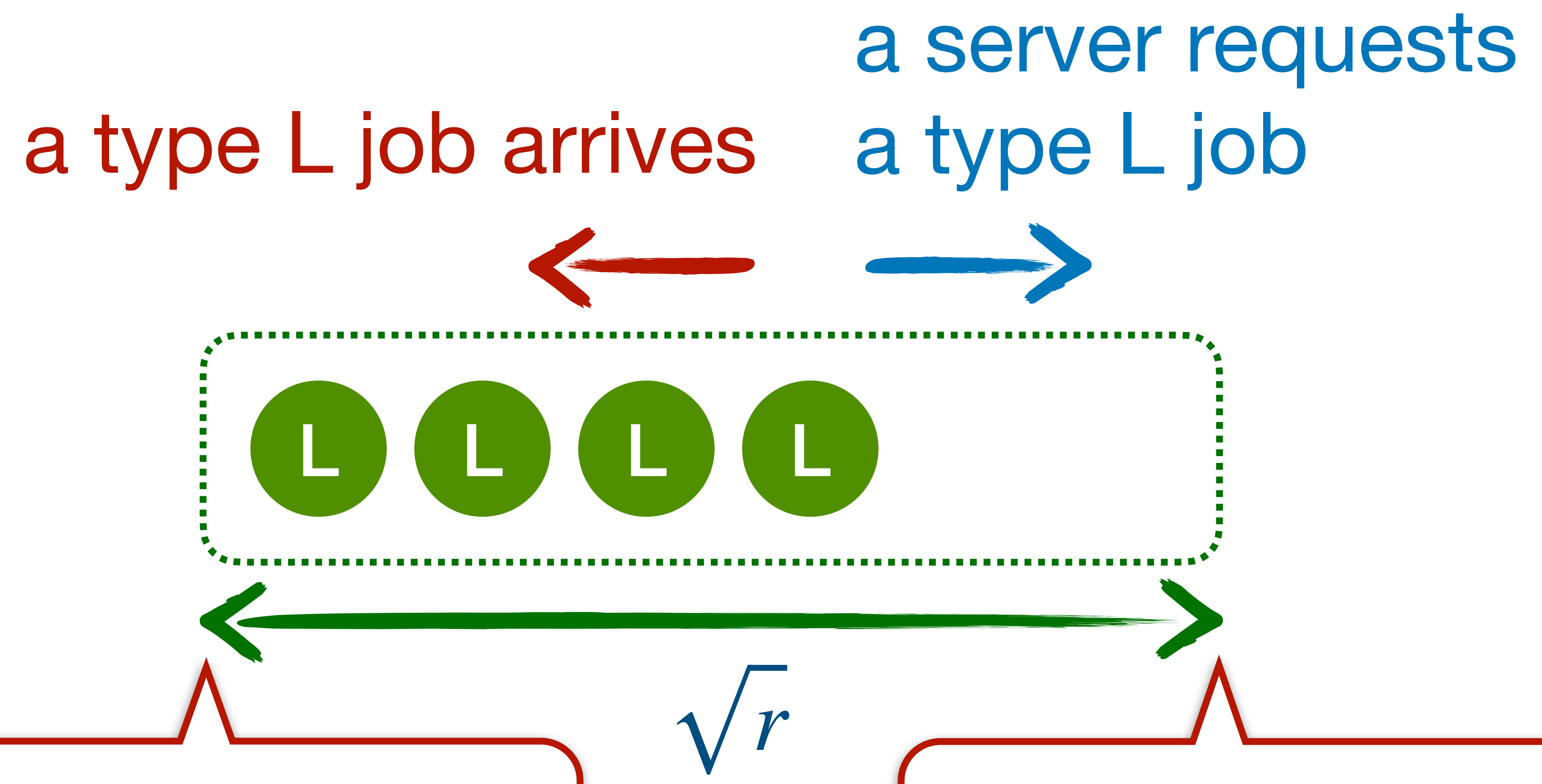
Generate a virtual job

Key proof idea 2



Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

and # backup servers $= O\left(\sqrt{r}\right)$



What happens when
tokens hits 0?

What happens when
tokens hits $\sqrt{r}$?

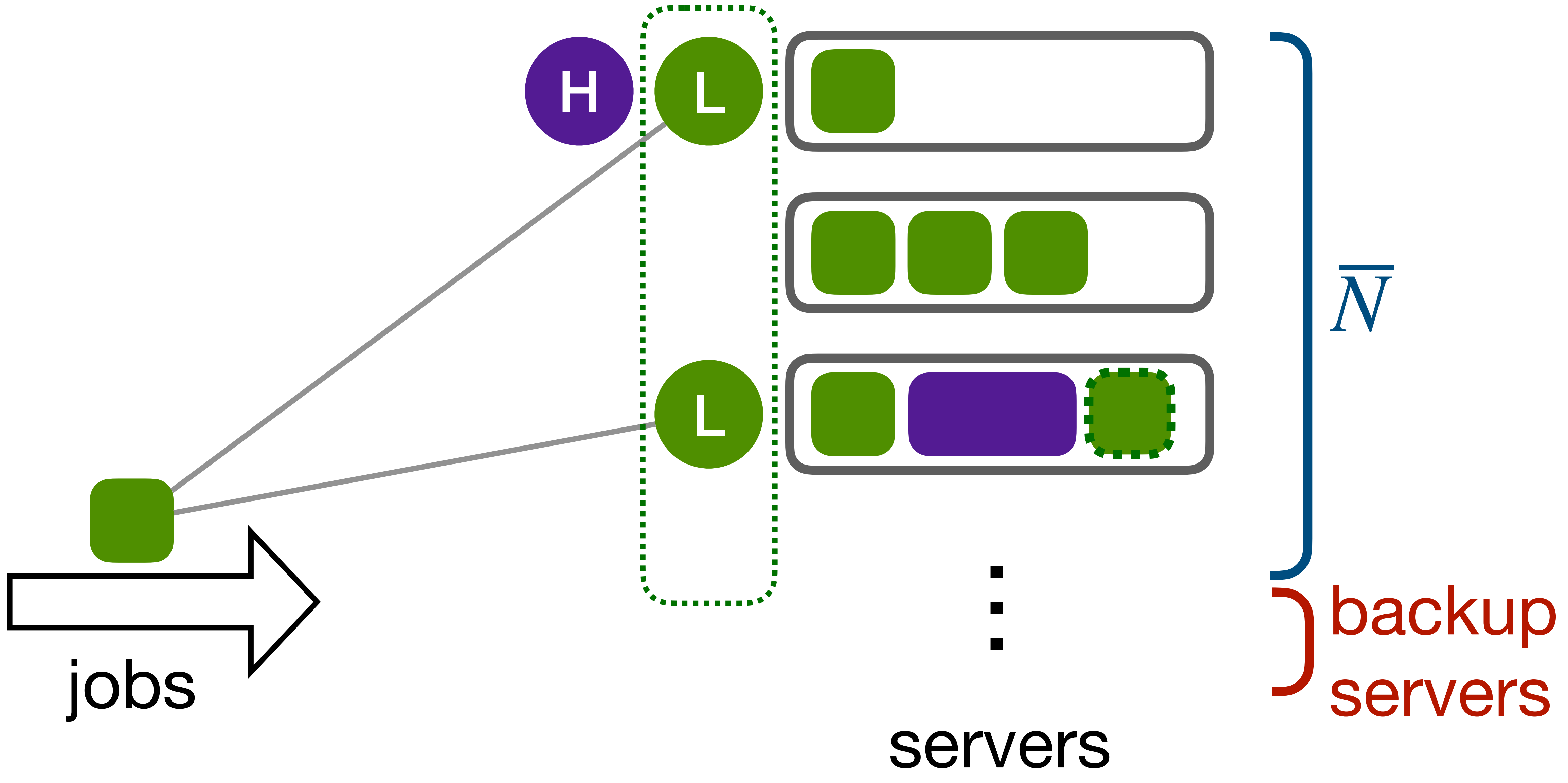
Generate a job to backup servers

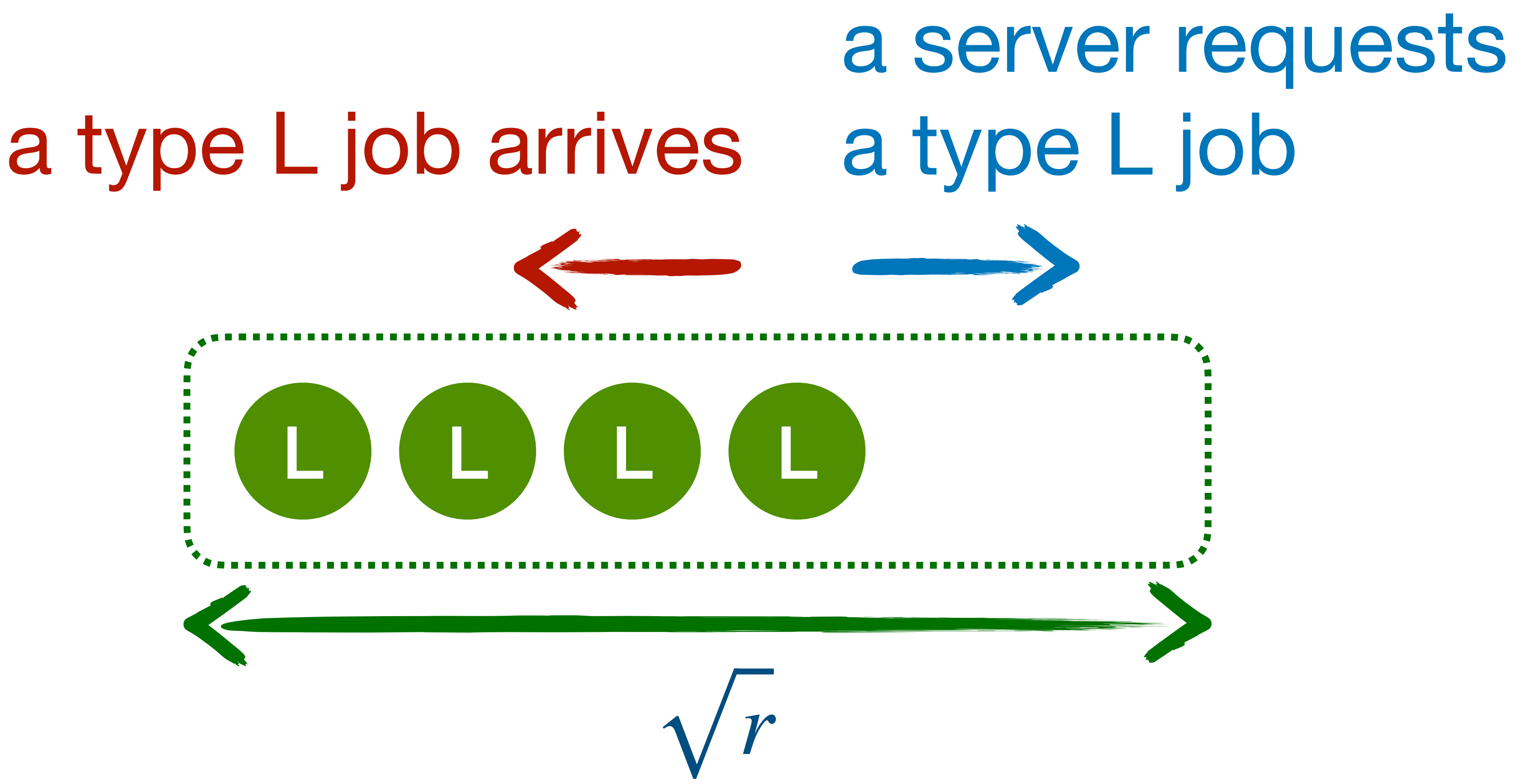
Generate a virtual job

Key proof idea 2

Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

and # backup servers $= O\left(\sqrt{r}\right)$

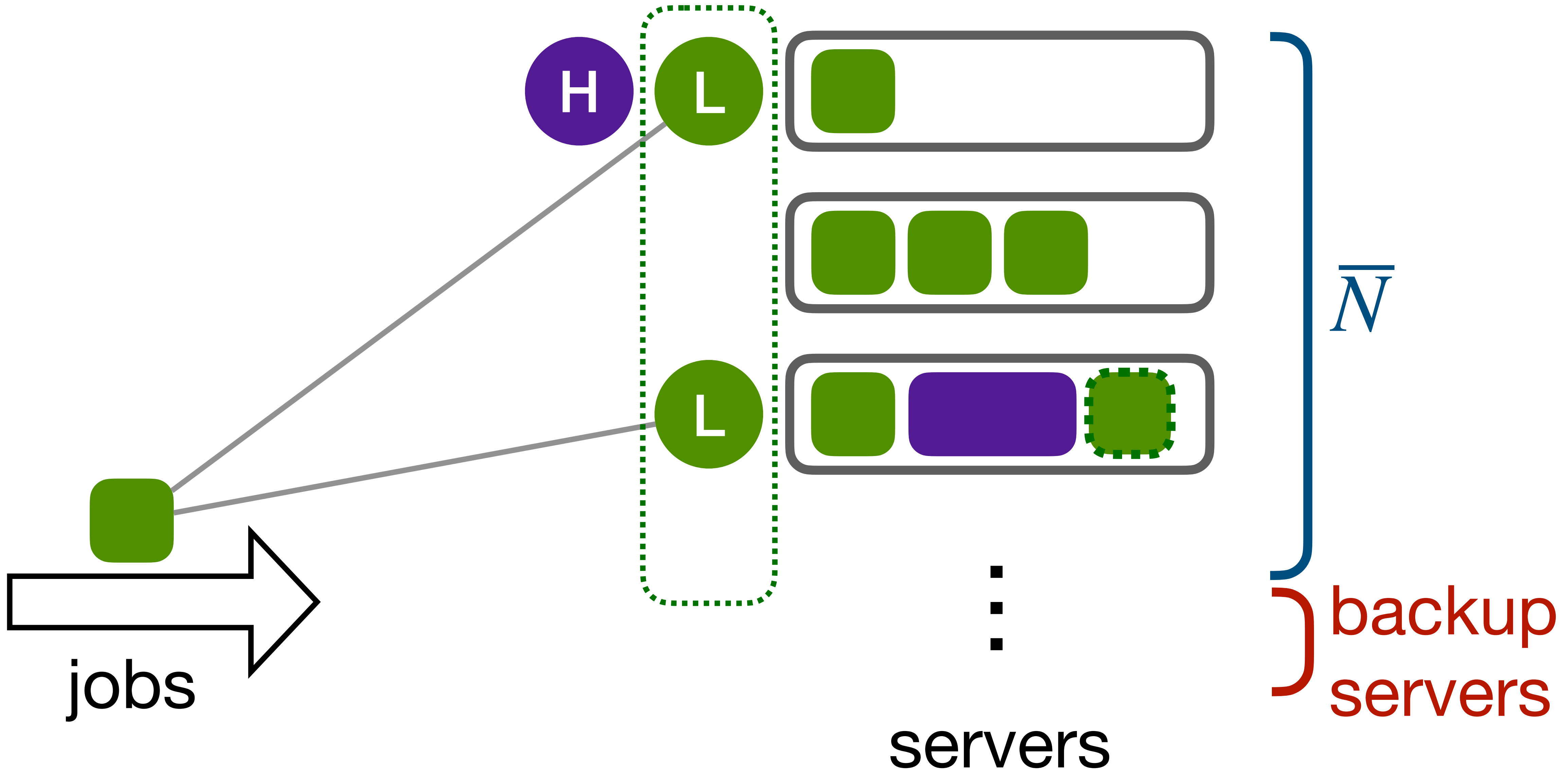


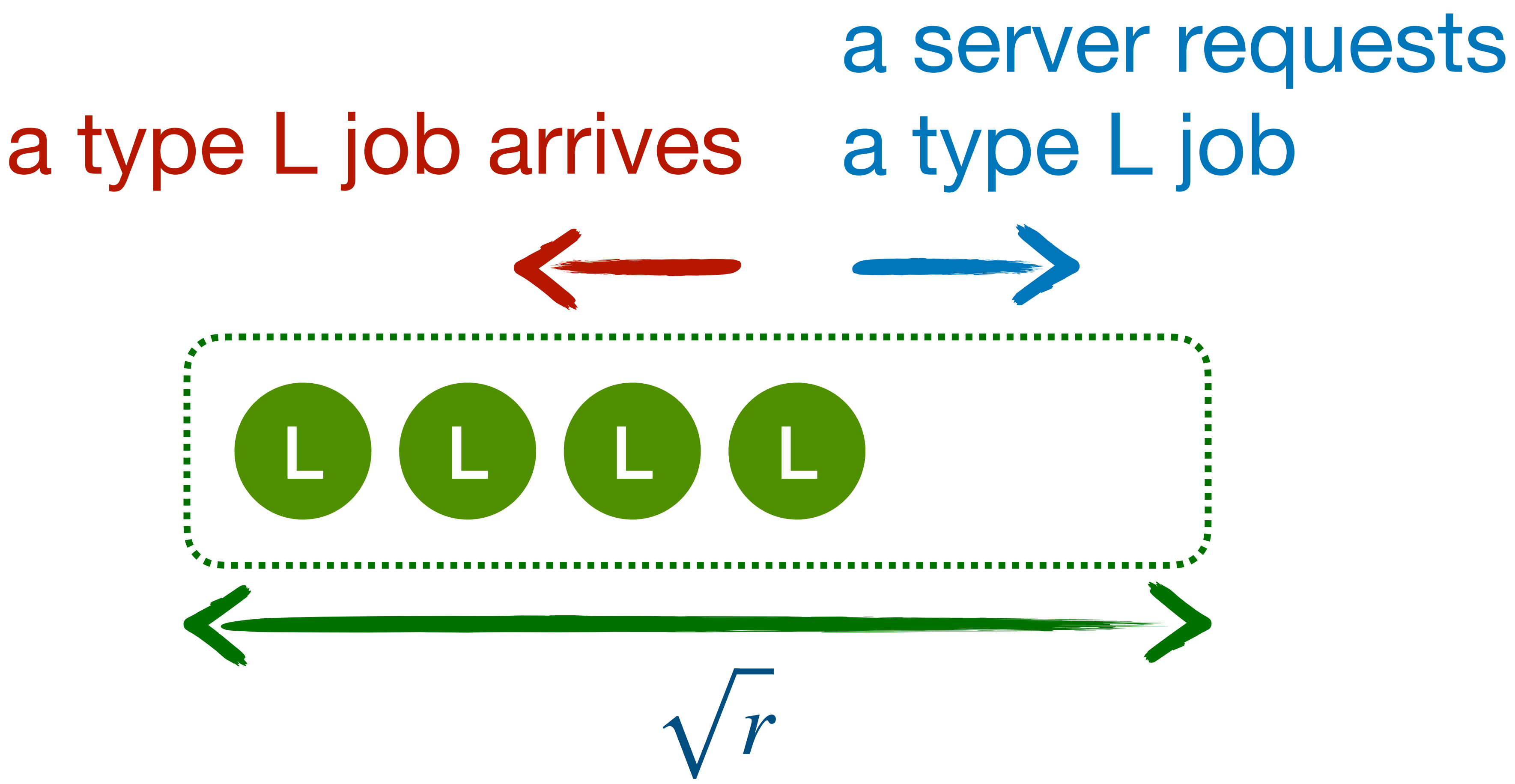


Key proof idea 2

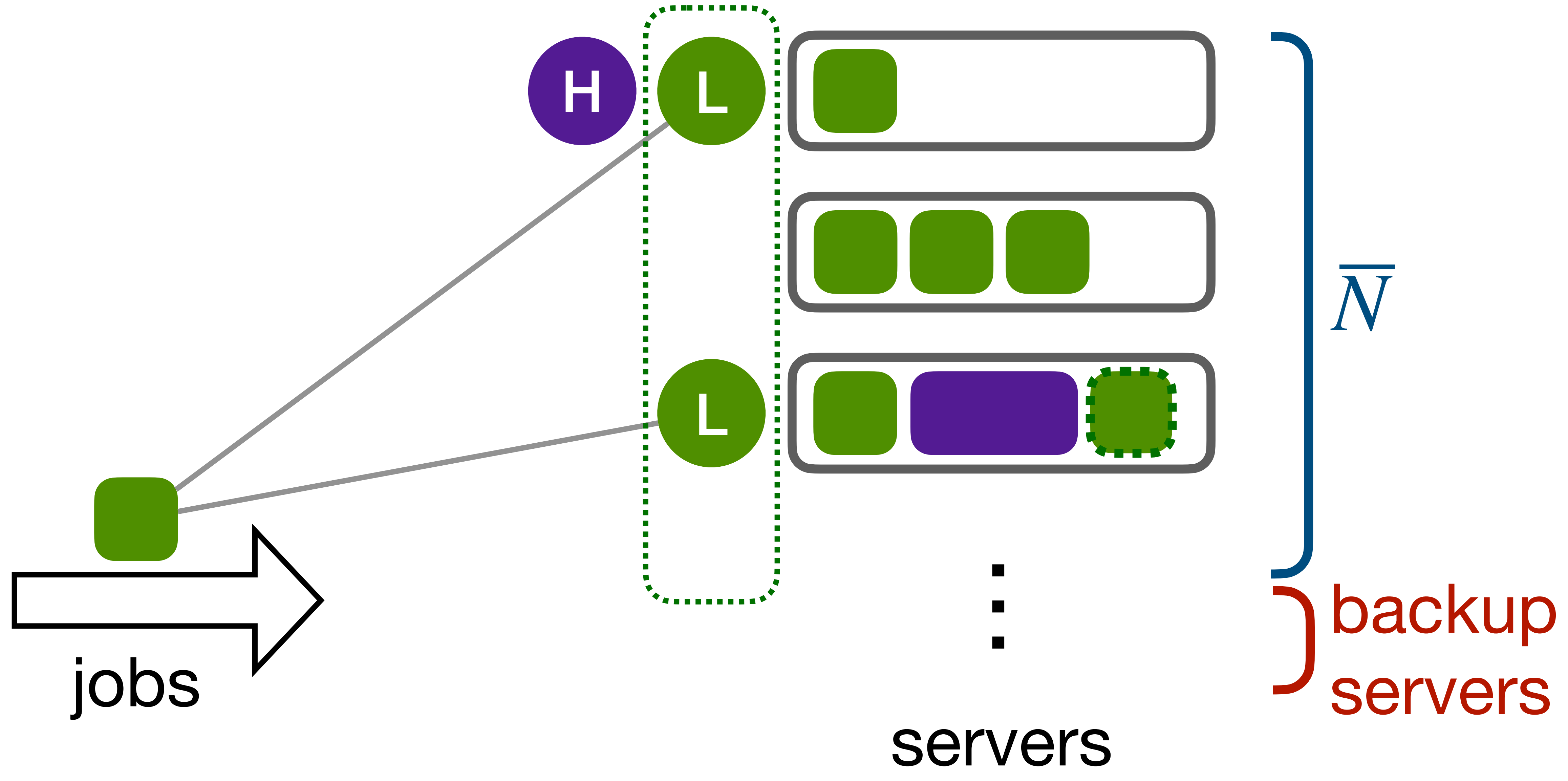
Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

and # backup servers $= O\left(\sqrt{r}\right)$



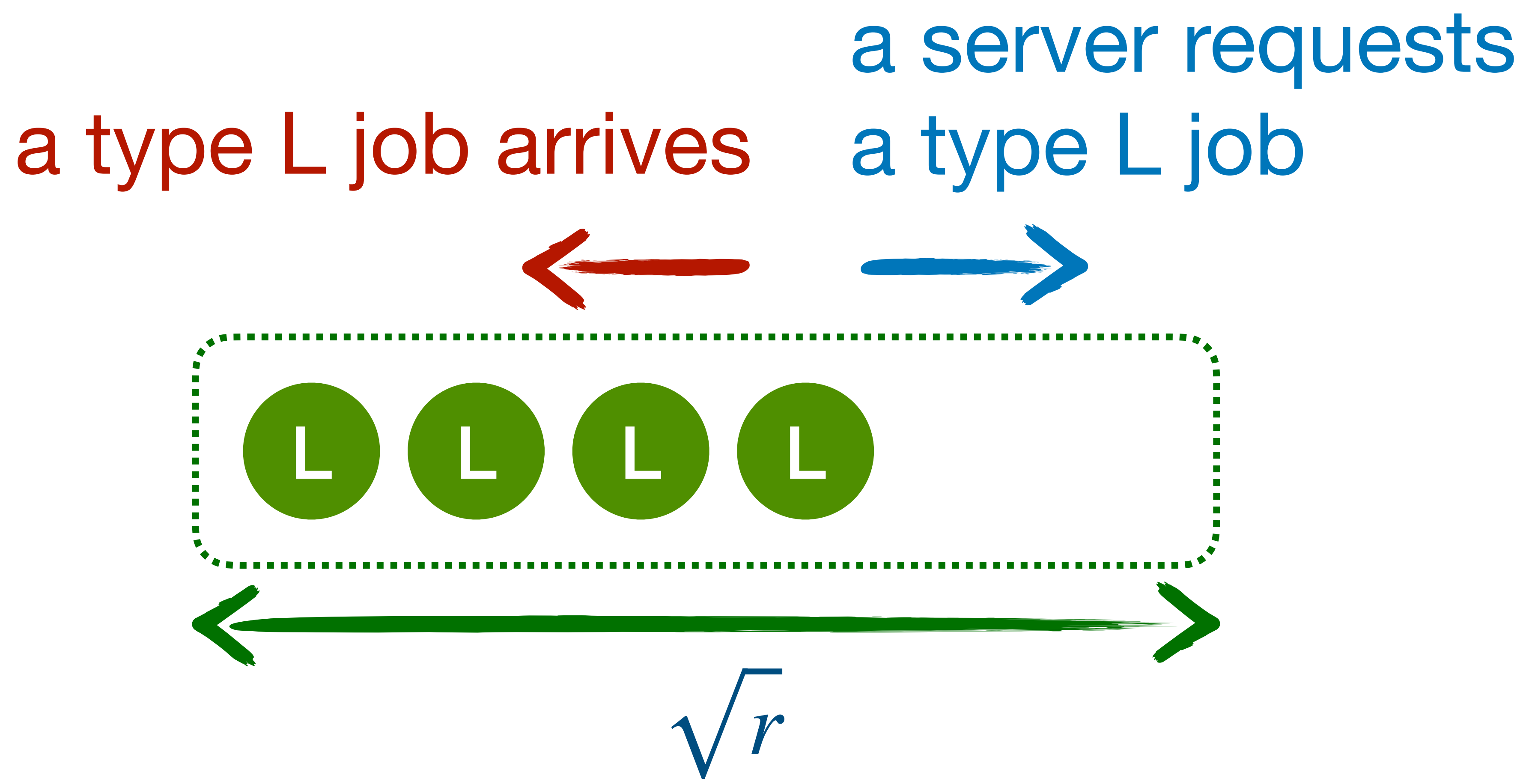


Key proof idea 2



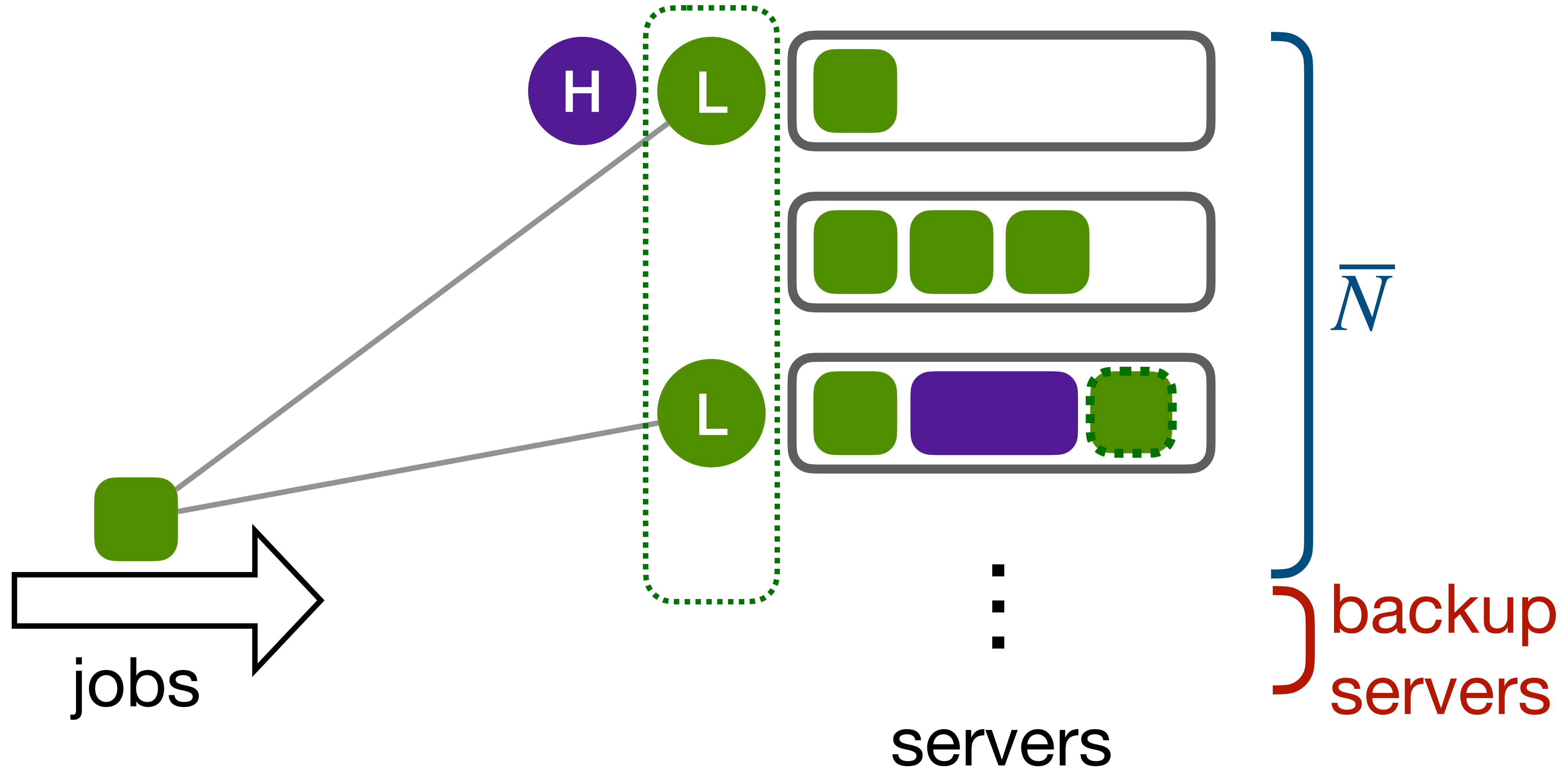
Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

and # backup servers $= O\left(\sqrt{r}\right)$

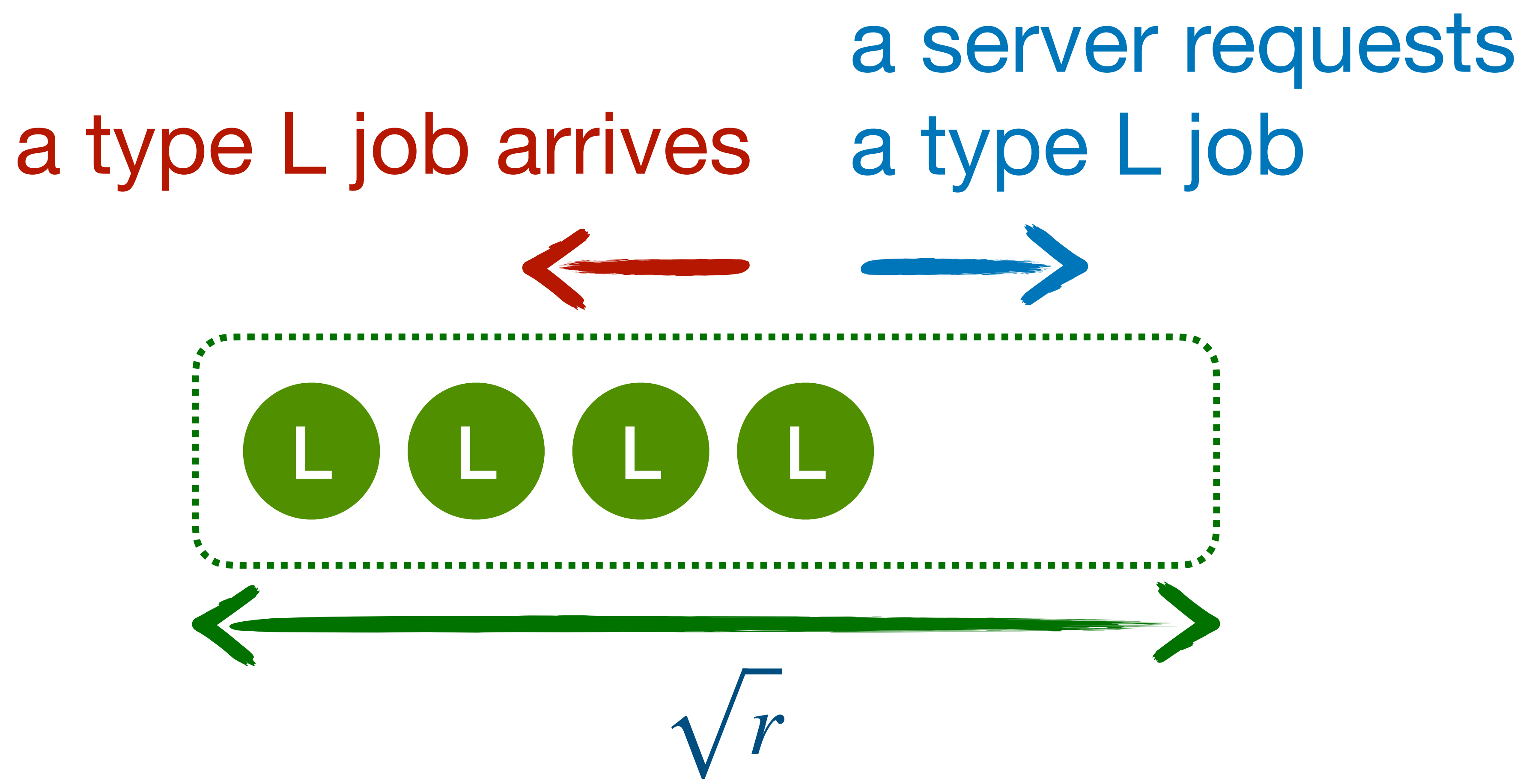


- An almost balanced random walk

Key proof idea 2

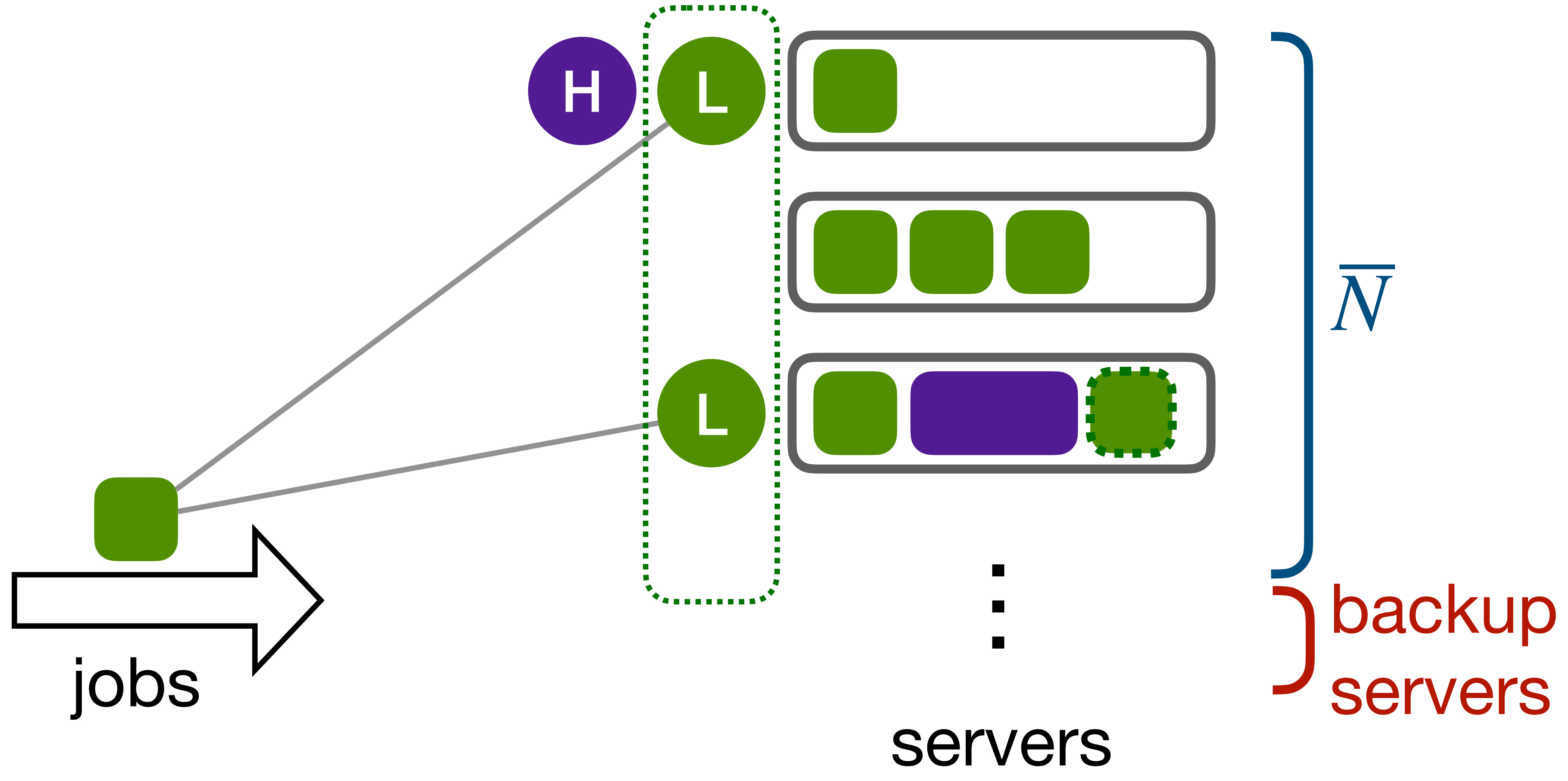


Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,
and # backup servers $= O\left(\sqrt{r}\right)$



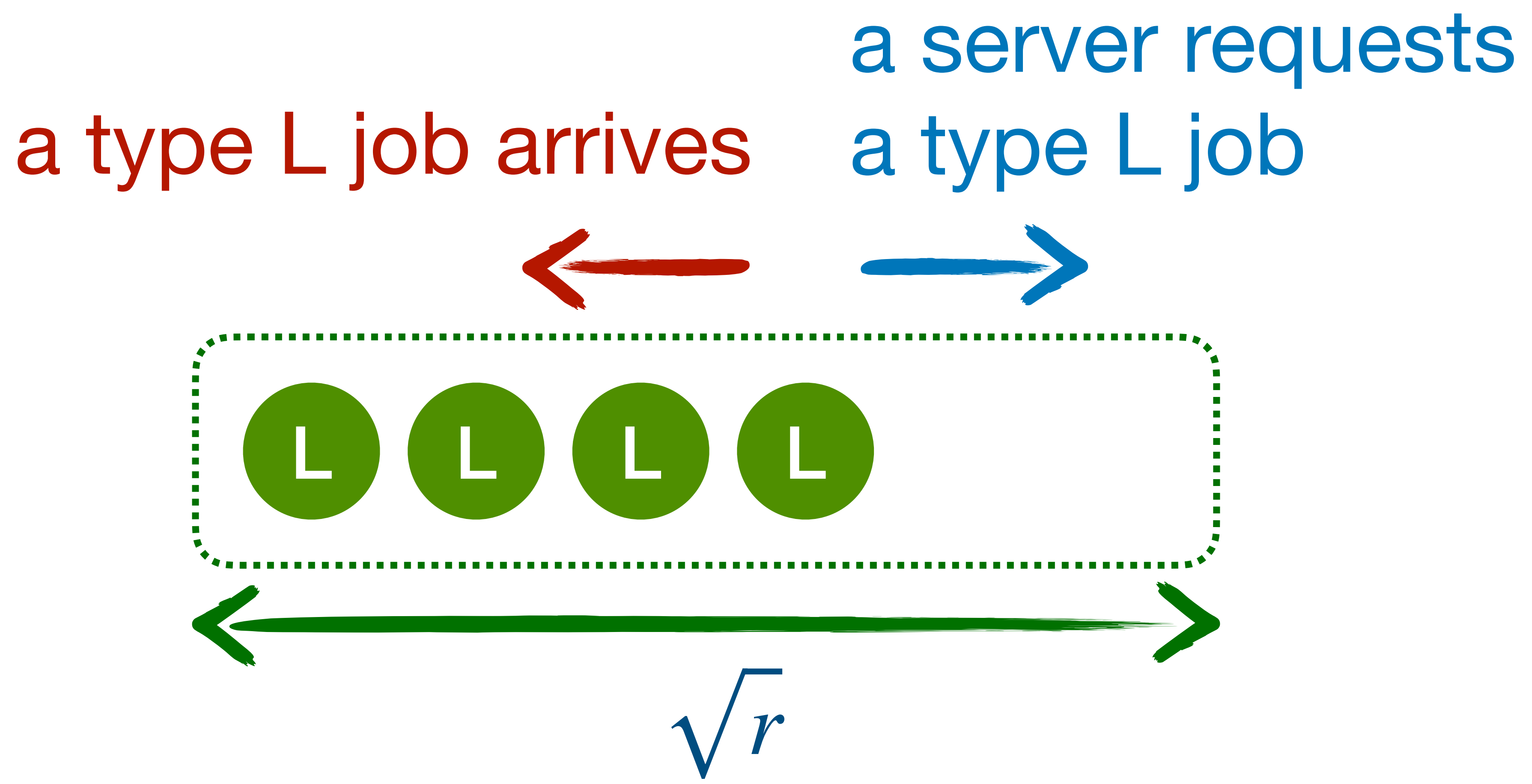
- An almost balanced random walk
- Stationary distribution ≈ uniform on $\{0, 1, \ldots, \sqrt{r}\}$

Key proof idea 2



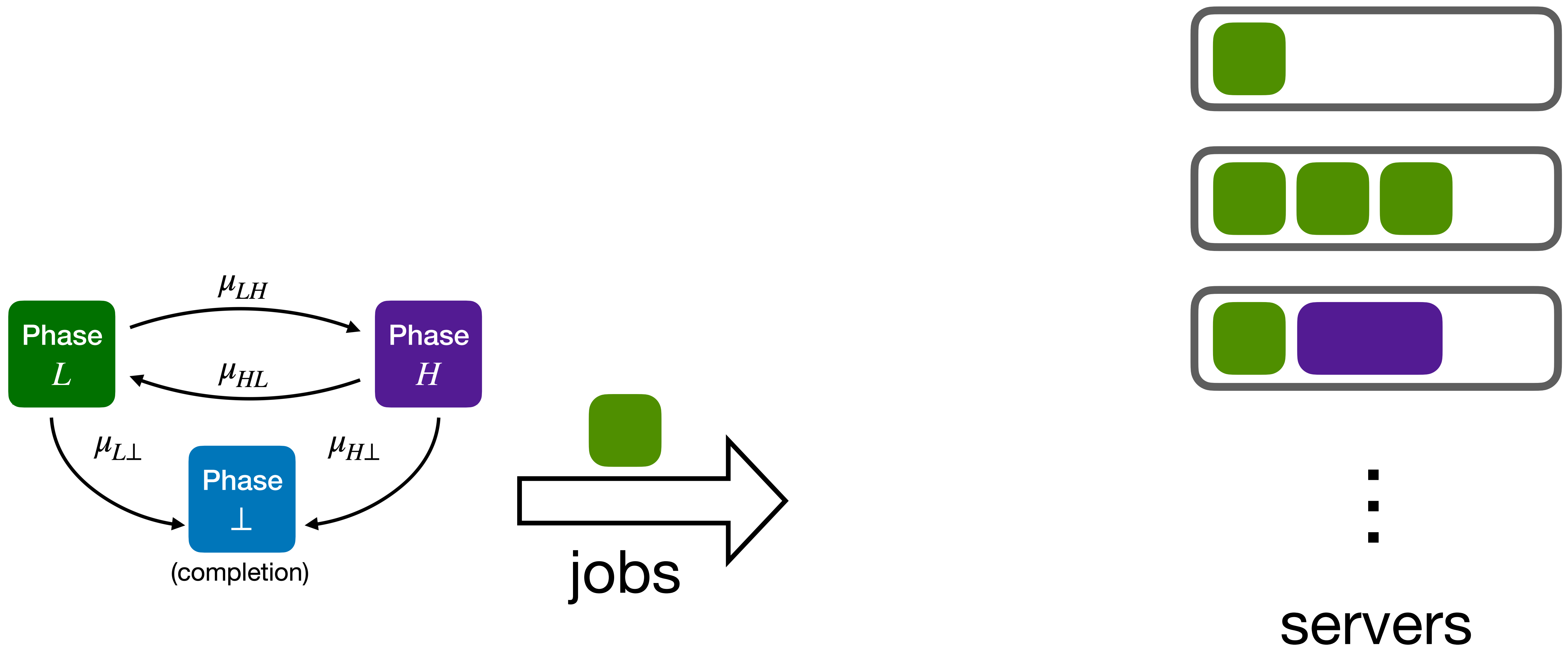
Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

and # backup servers $= O\left(\sqrt{r}\right)$



- An almost balanced random walk
- Stationary distribution ≈ uniform on $\{0, 1, \ldots, \sqrt{r}\}$
- Rate of generating virtual jobs
  ≈ rate of sending jobs to backup servers
  ≈ arrival rate$/\sqrt{r} = O\left(\sqrt{r}\right)$

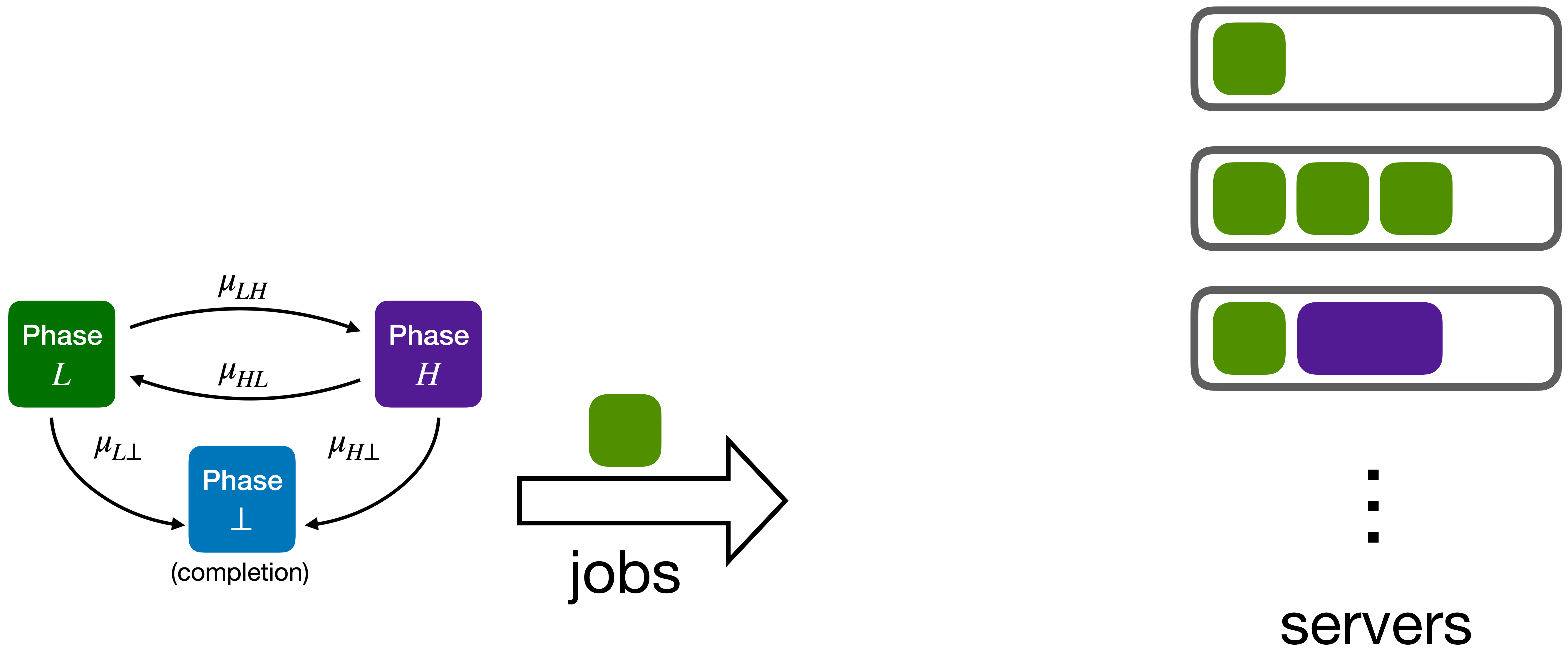
Summary



Phase L
Phase H
Phase ⊥
(completion)
μ_{LH}
μ_{HL}
$\mu_{L\perp}$
$\mu_{H\perp}$
jobs
servers

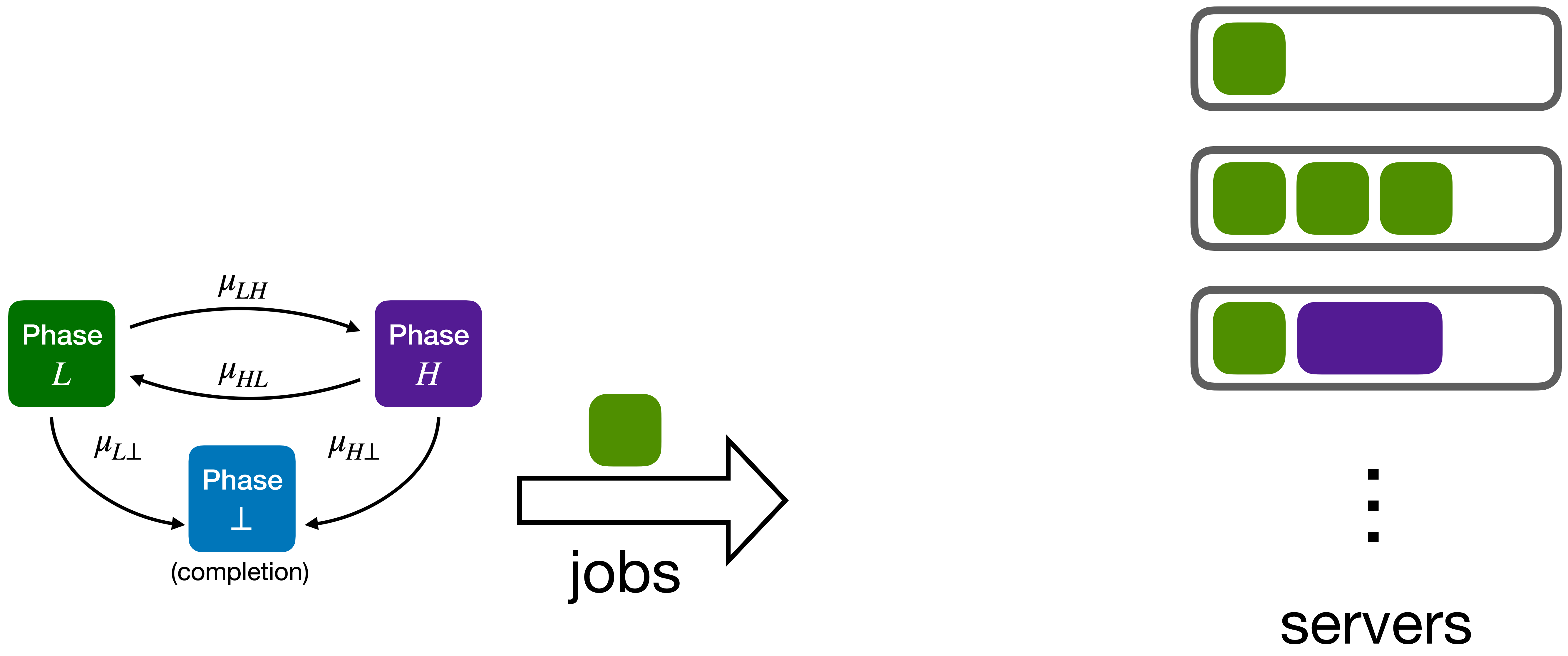
Summary

- We considered the problem of assigning jobs to servers when jobs have time-varying resource requirements



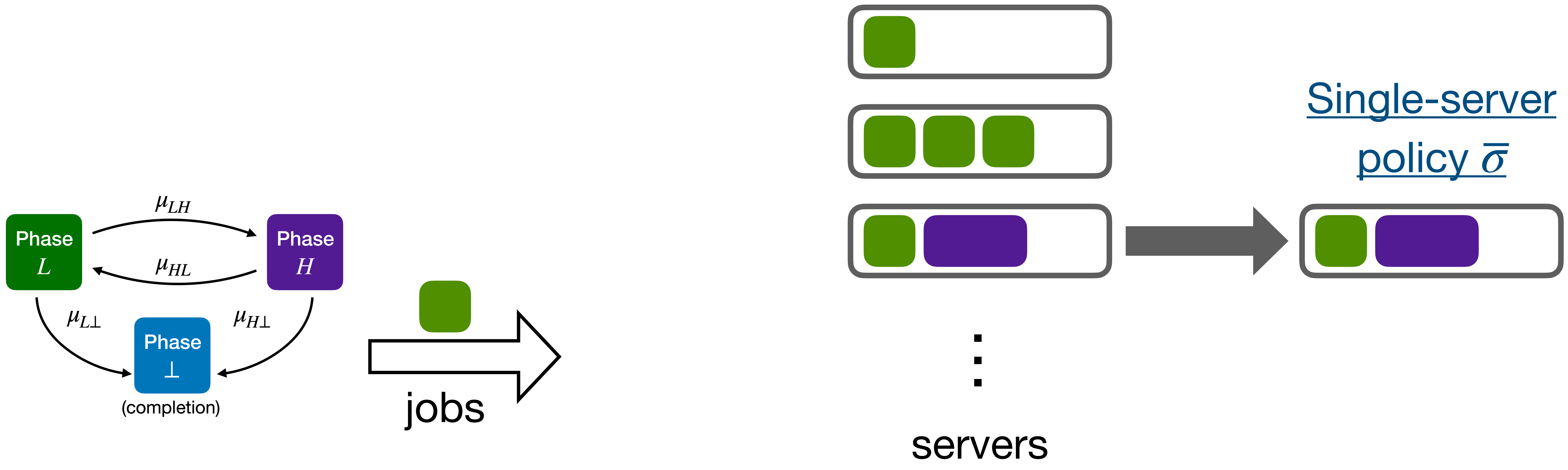
Summary

- We considered the problem of assigning jobs to servers when jobs have time-varying resource requirements
- We designed an asymptotically optimal policy



Summary

- We considered the problem of assigning jobs to servers when jobs have time-varying resource requirements
- We designed an asymptotically optimal policy
- We proposed a policy-conversion framework that allows us to reduce the policy-design problem to that in a single-server system



Summary

- We considered the problem of assigning jobs to servers when jobs have time-varying resource requirements
- We designed an asymptotically optimal policy
- We proposed a policy-conversion framework that allows us to reduce the policy-design problem to that in a single-server system
- A highlight of the framework is the meta-algorithm, JOIN-THE-RECENTLY-REQUESTING-SERVER (JRSS), that converts a single-server policy to a policy in the original system