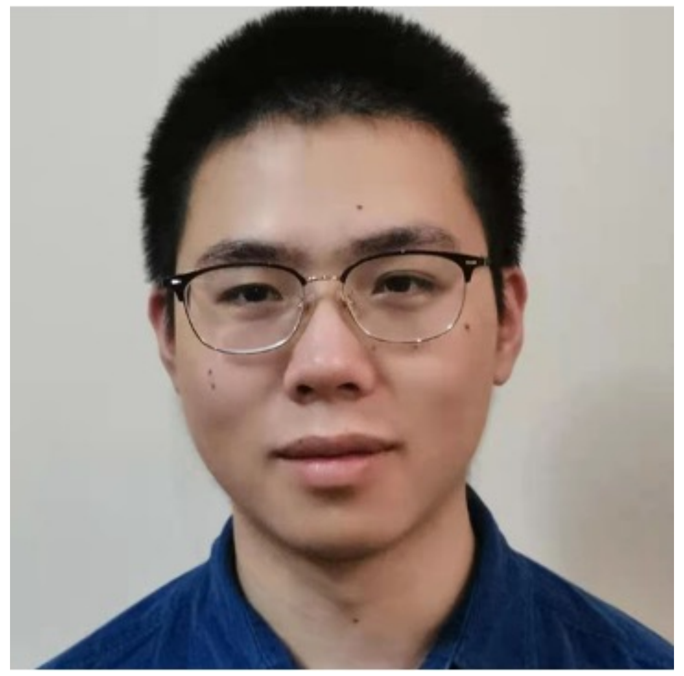
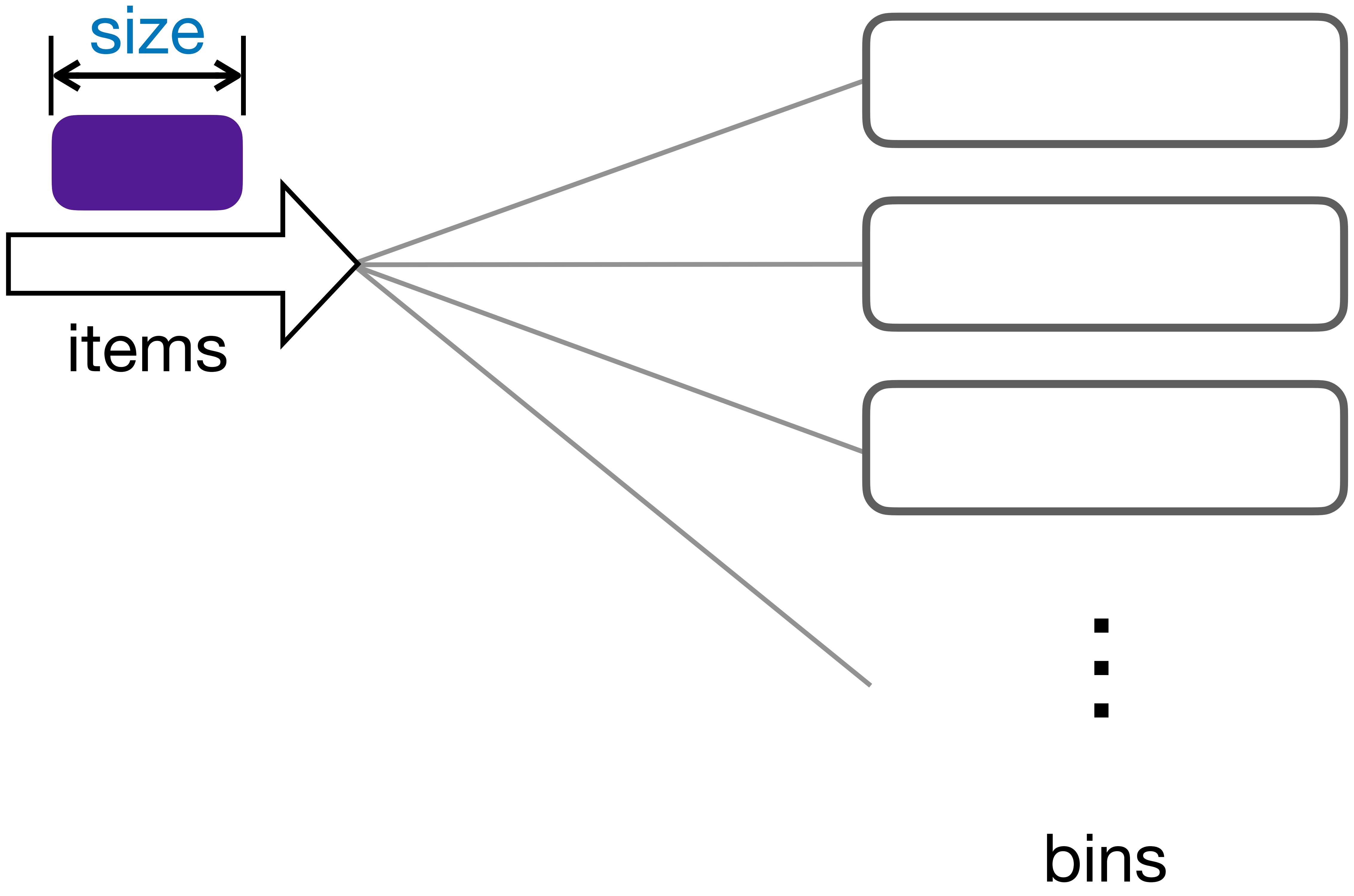
Near-Optimal Stochastic Bin-Packing in Large Service Systems with Time-Varying Item Sizes



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

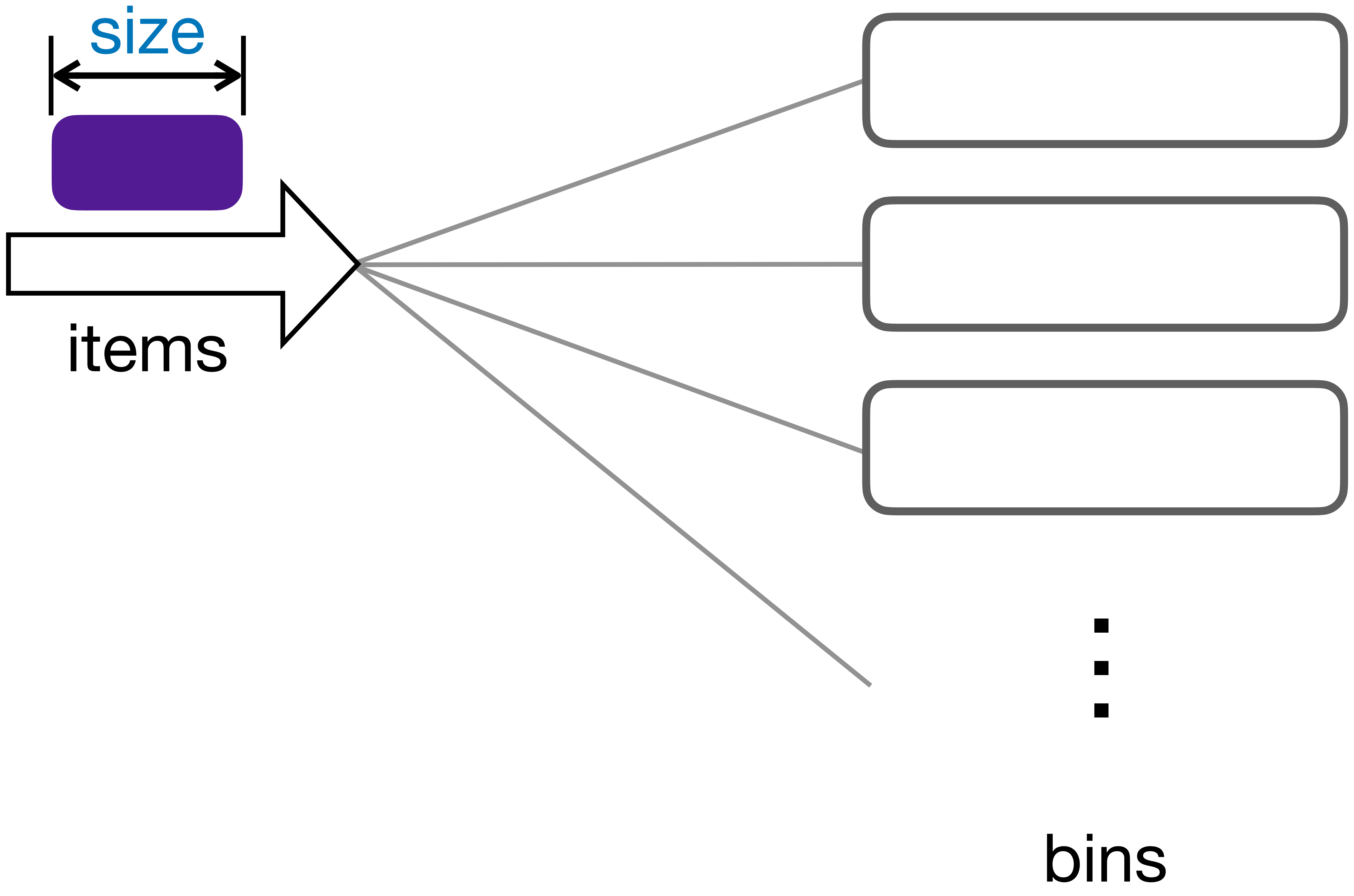
Weina Wang
Carnegie Mellon University

The problem



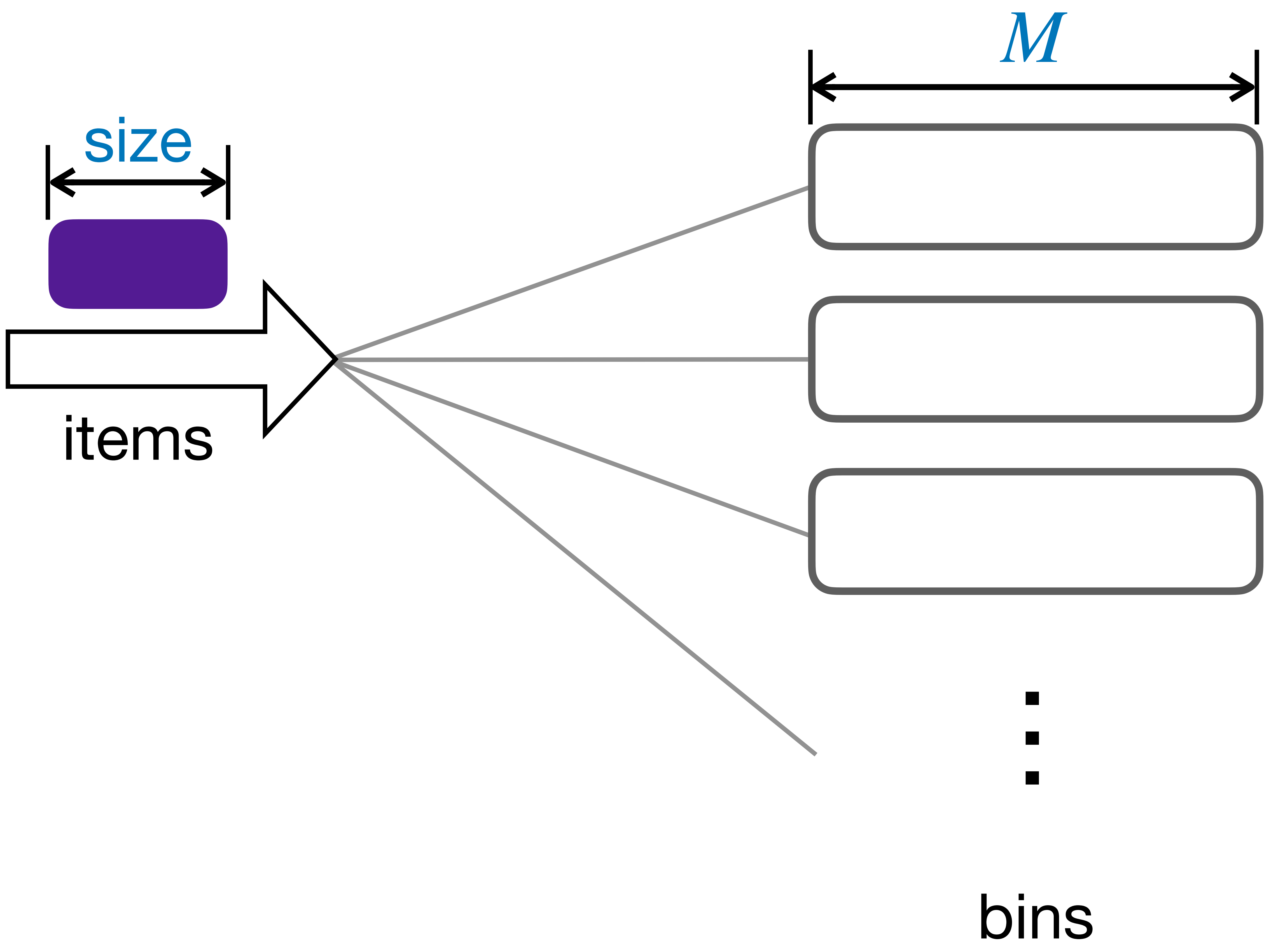
The problem

- Each arriving item needs to be assigned to a bin



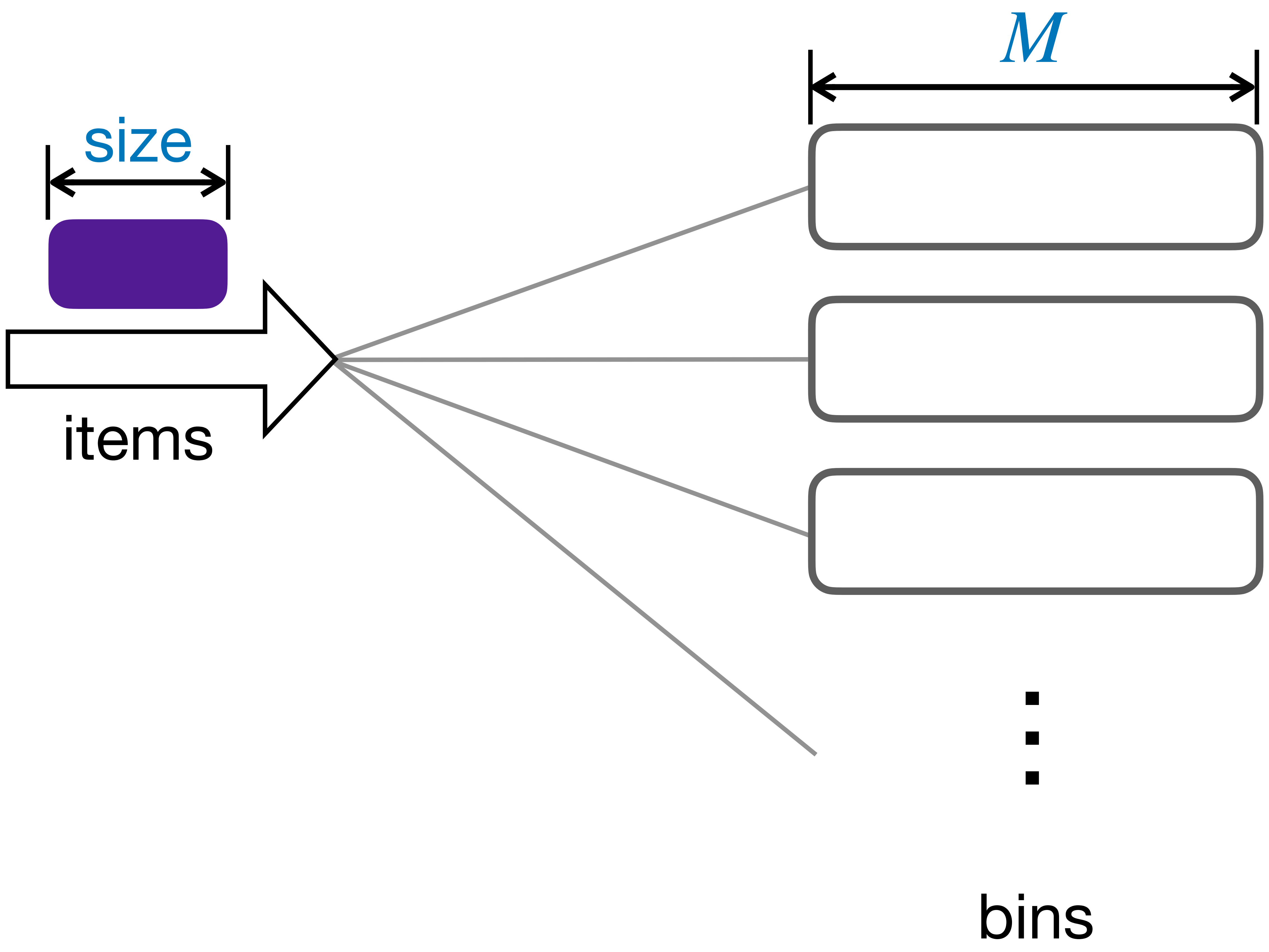
The problem

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



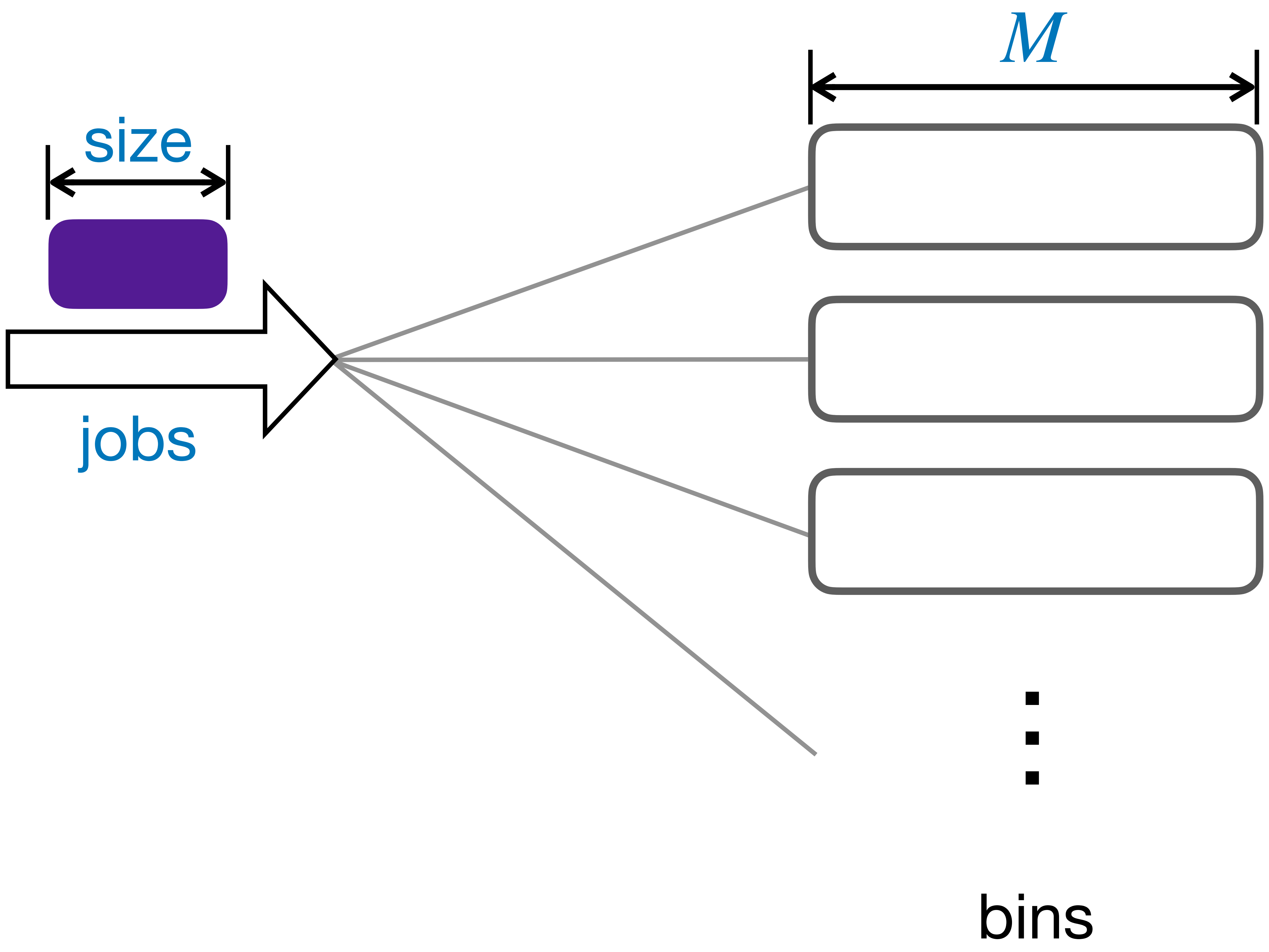
The problem

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



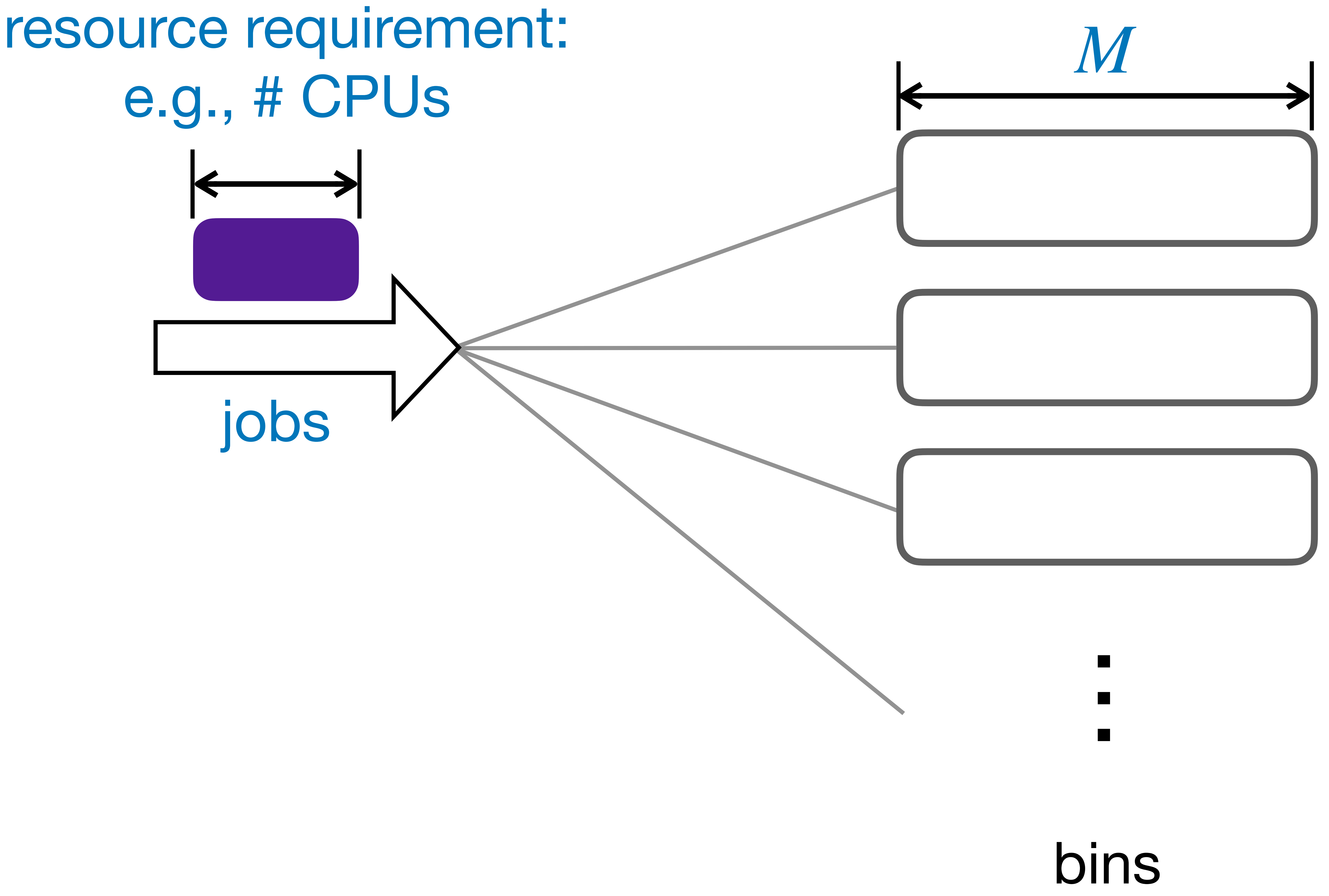
The problem

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



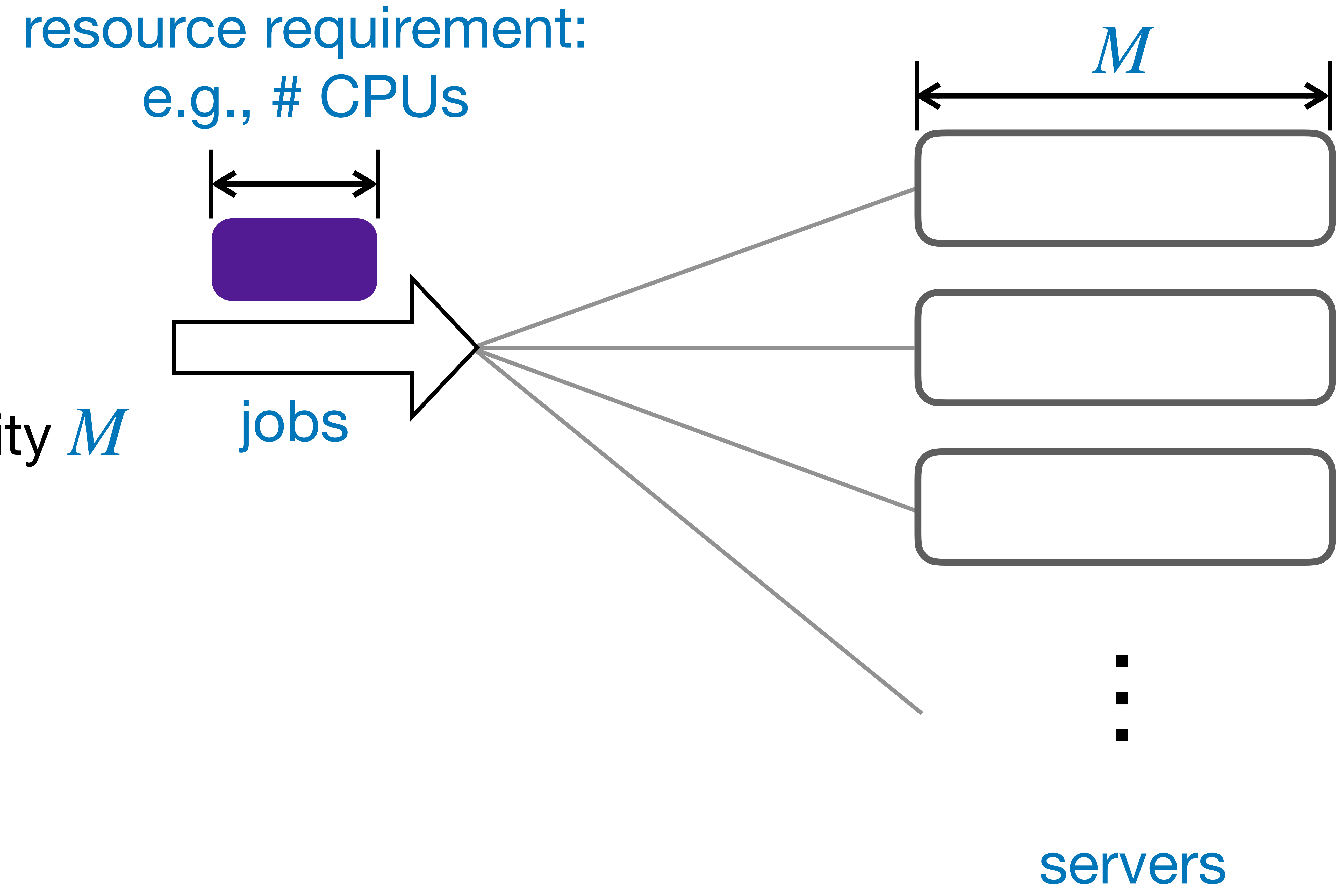
The problem

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



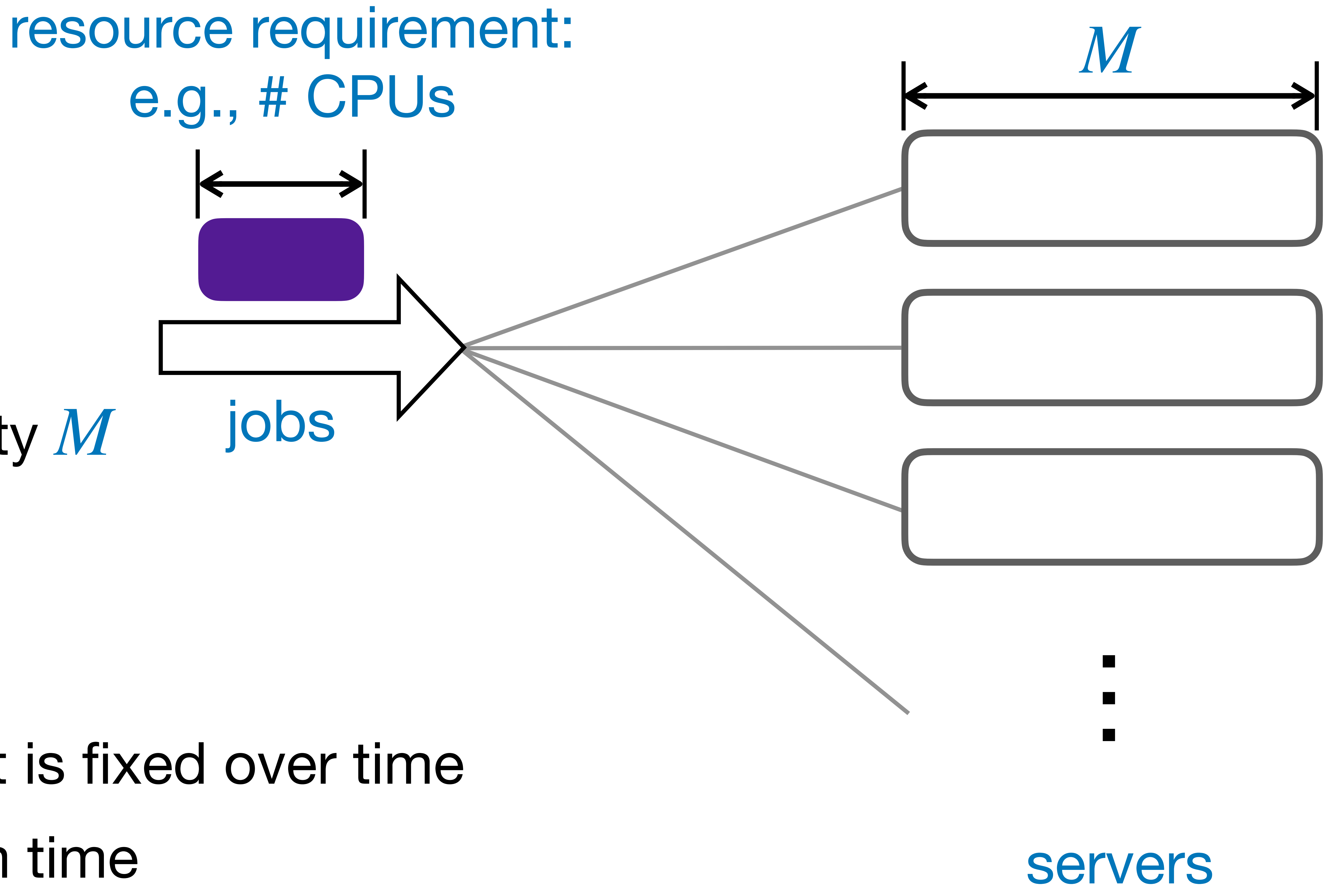
The problem

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



The problem

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

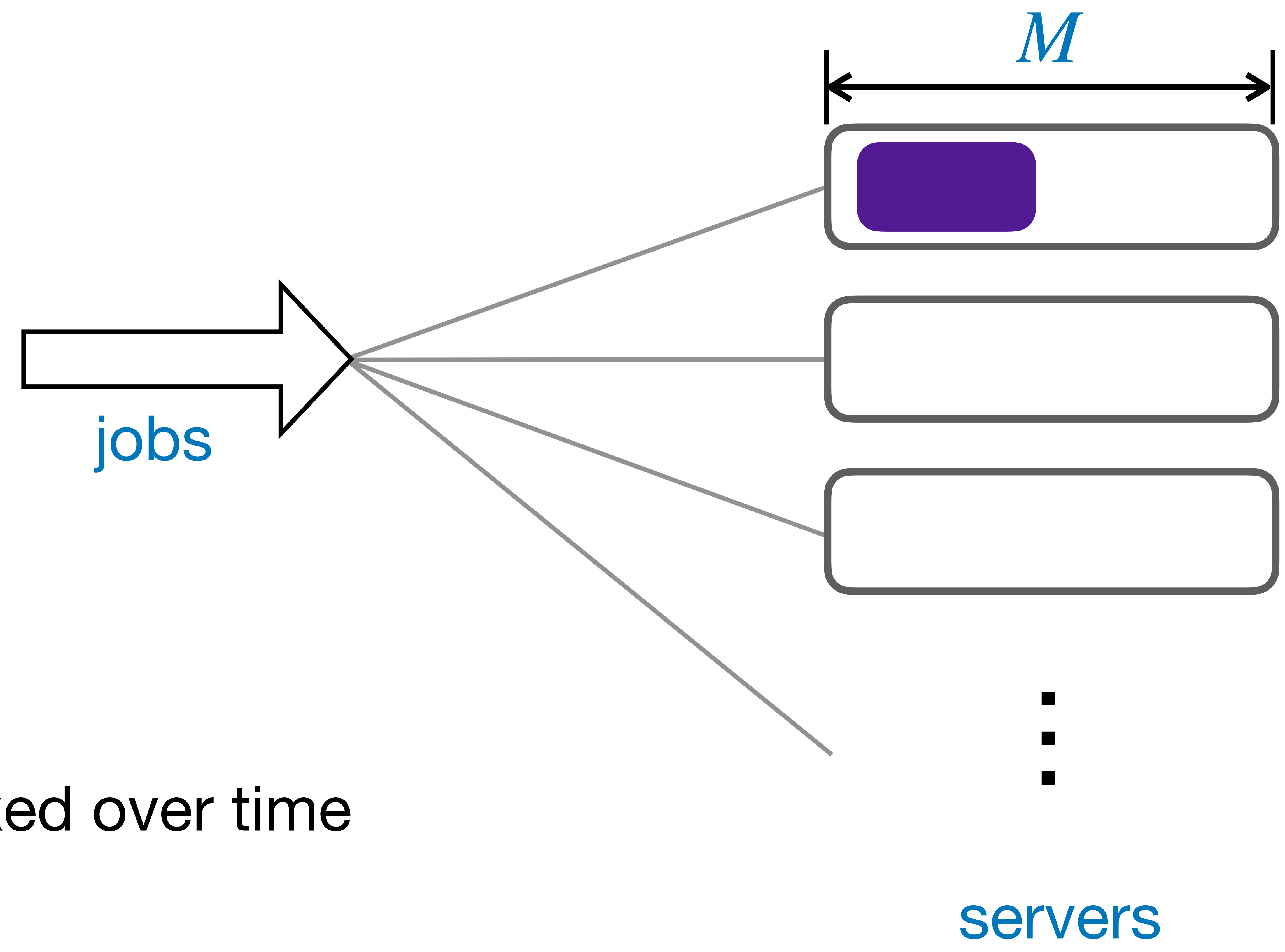


Traditional job model:

- Each job's resource requirement is fixed over time
- Each job departs after a random time

The problem

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

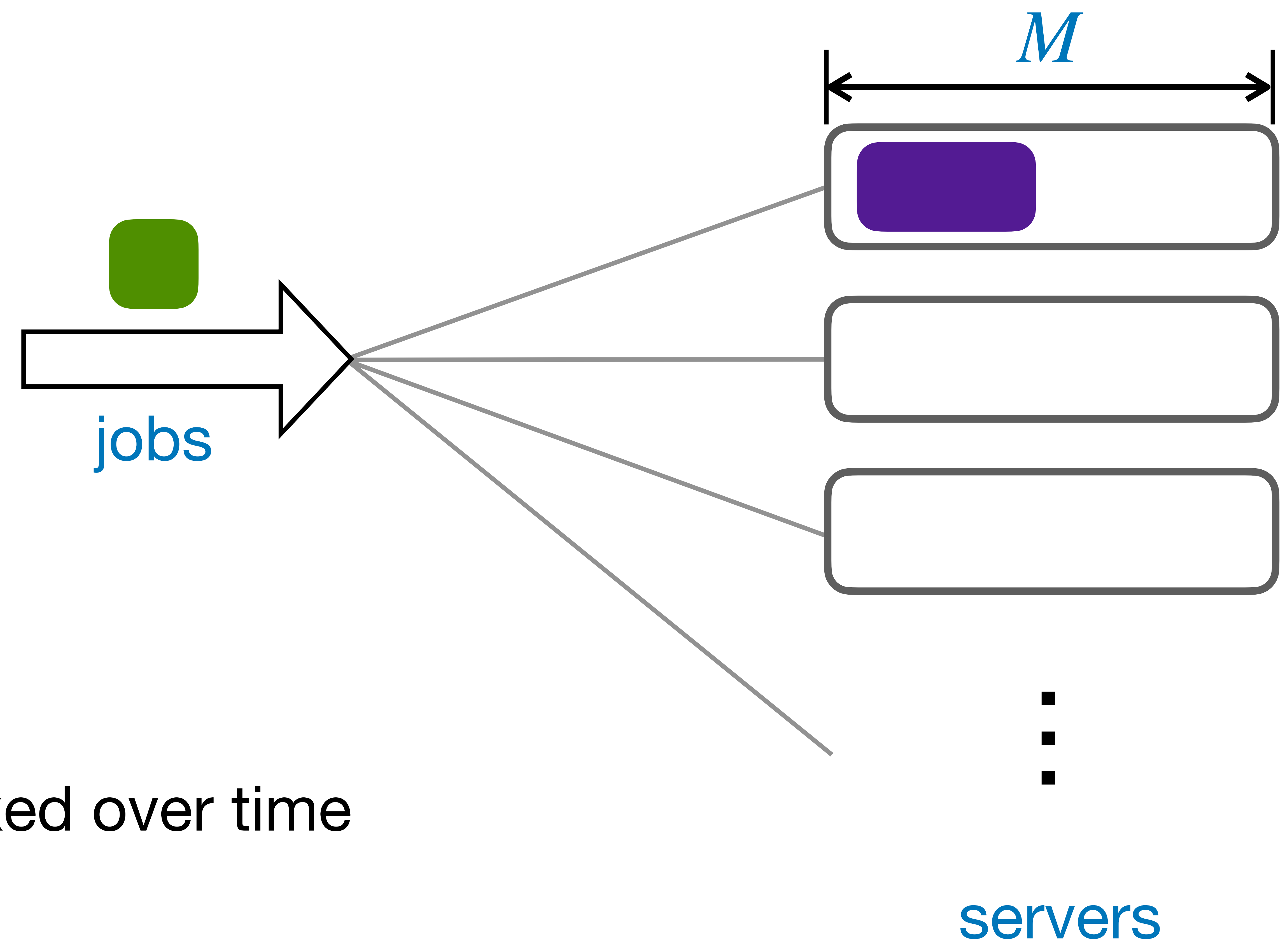


Traditional job model:

- Each job's resource requirement is fixed over time
- Each job departs after a random time

The problem

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

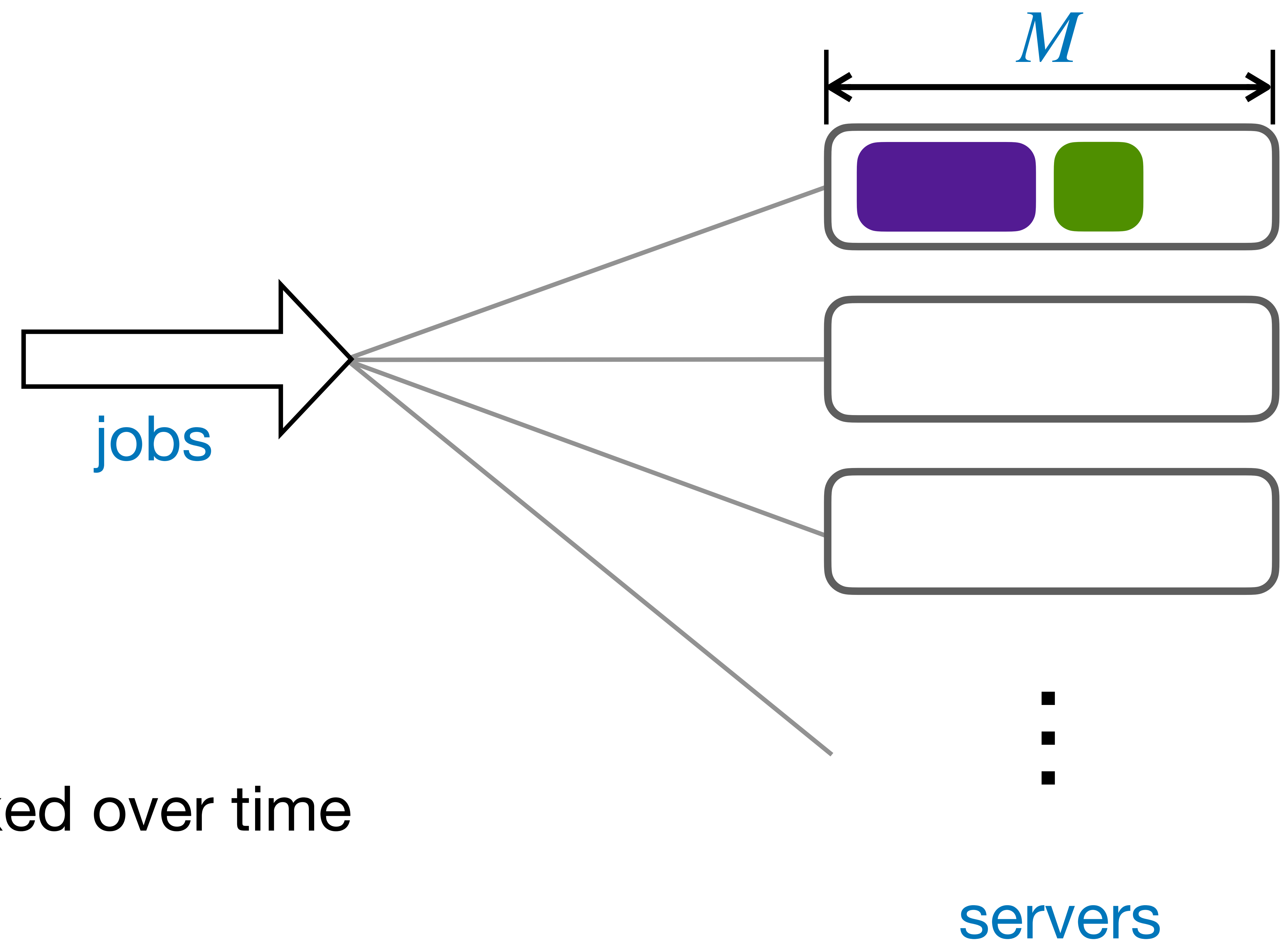


Traditional job model:

- Each job's resource requirement is fixed over time
- Each job departs after a random time

The problem

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

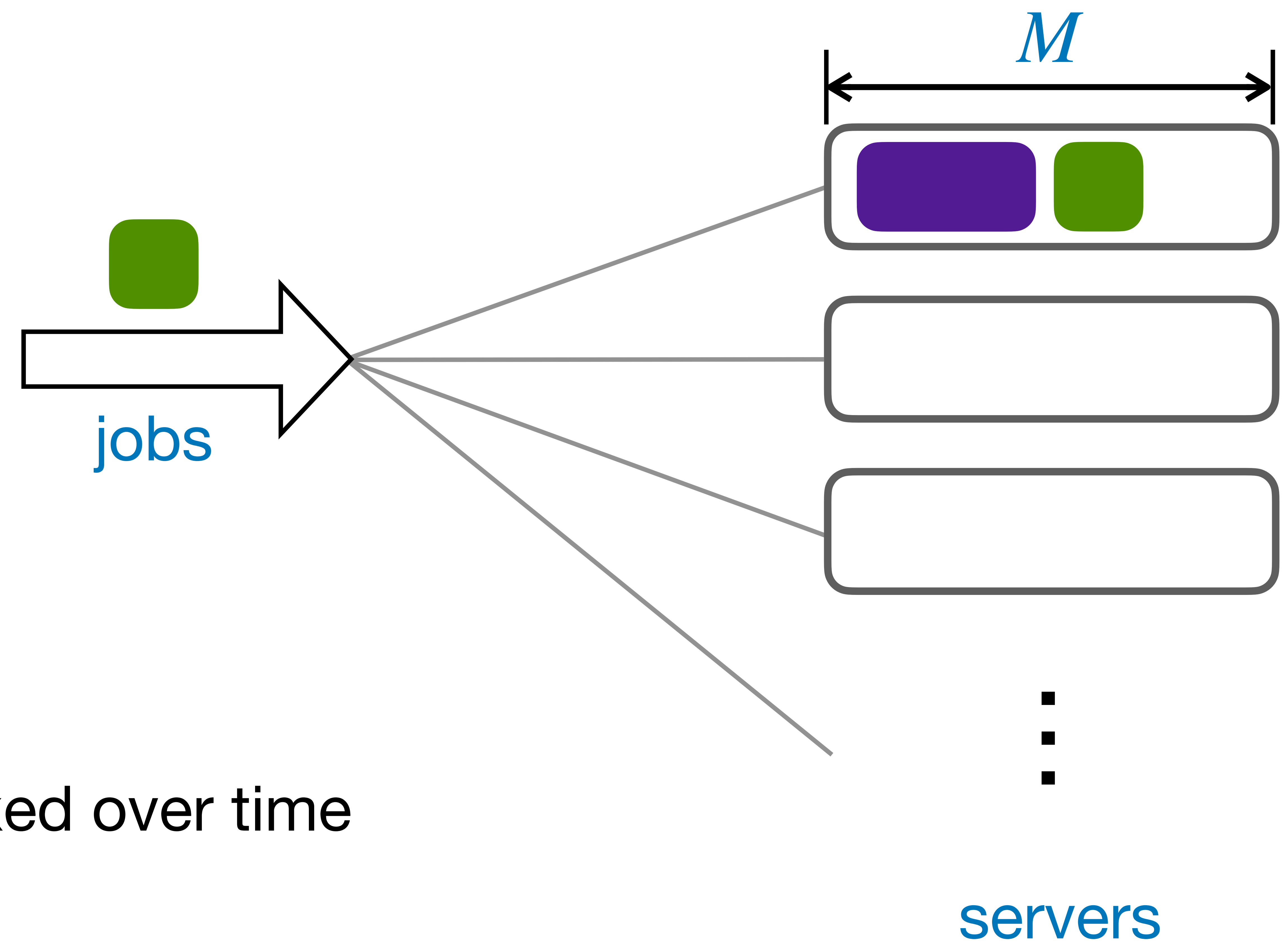


Traditional job model:

- Each job's resource requirement is fixed over time
- Each job departs after a random time

The problem

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

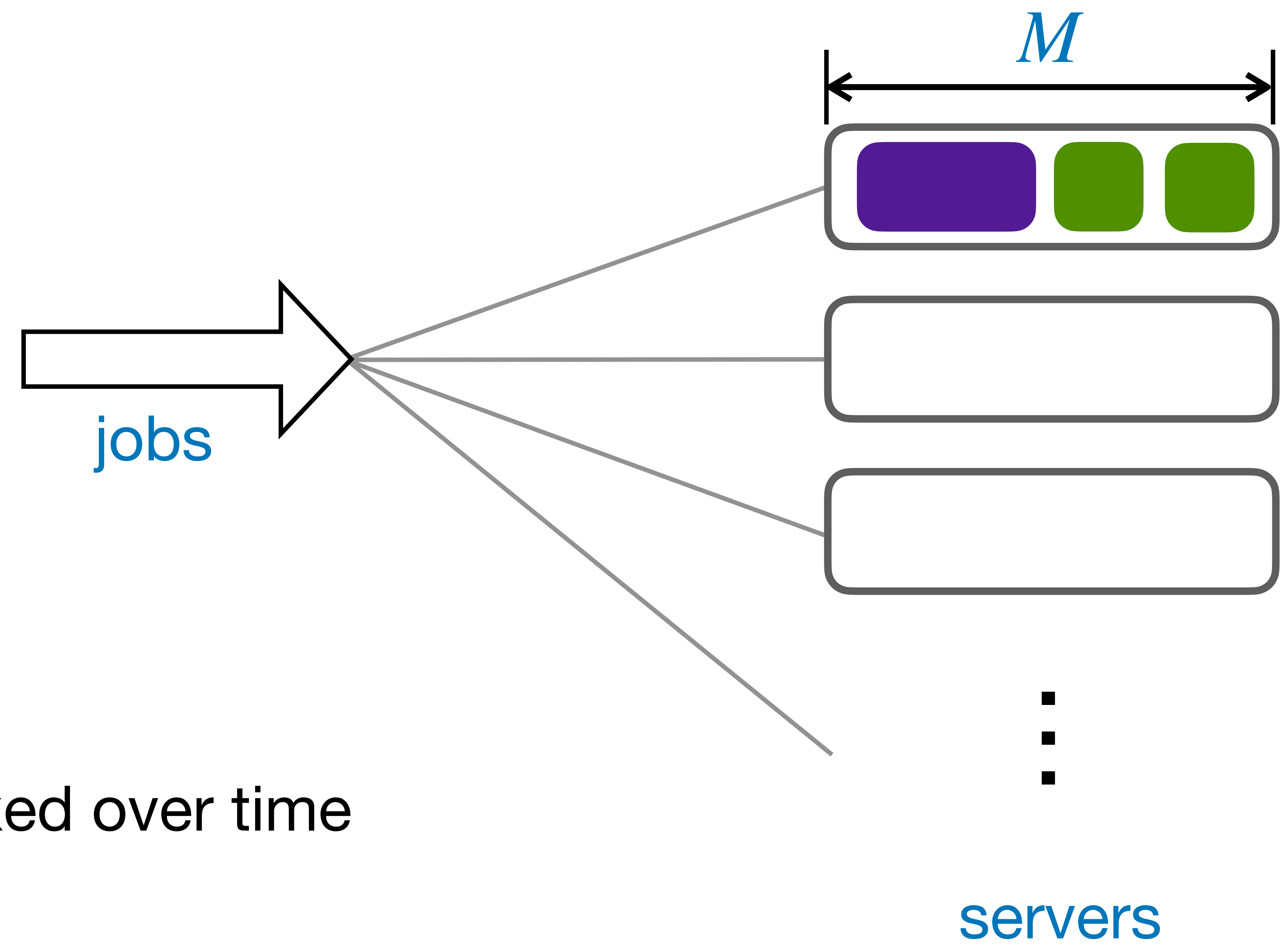


Traditional job model:

- Each job's resource requirement is fixed over time
- Each job departs after a random time

The problem

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

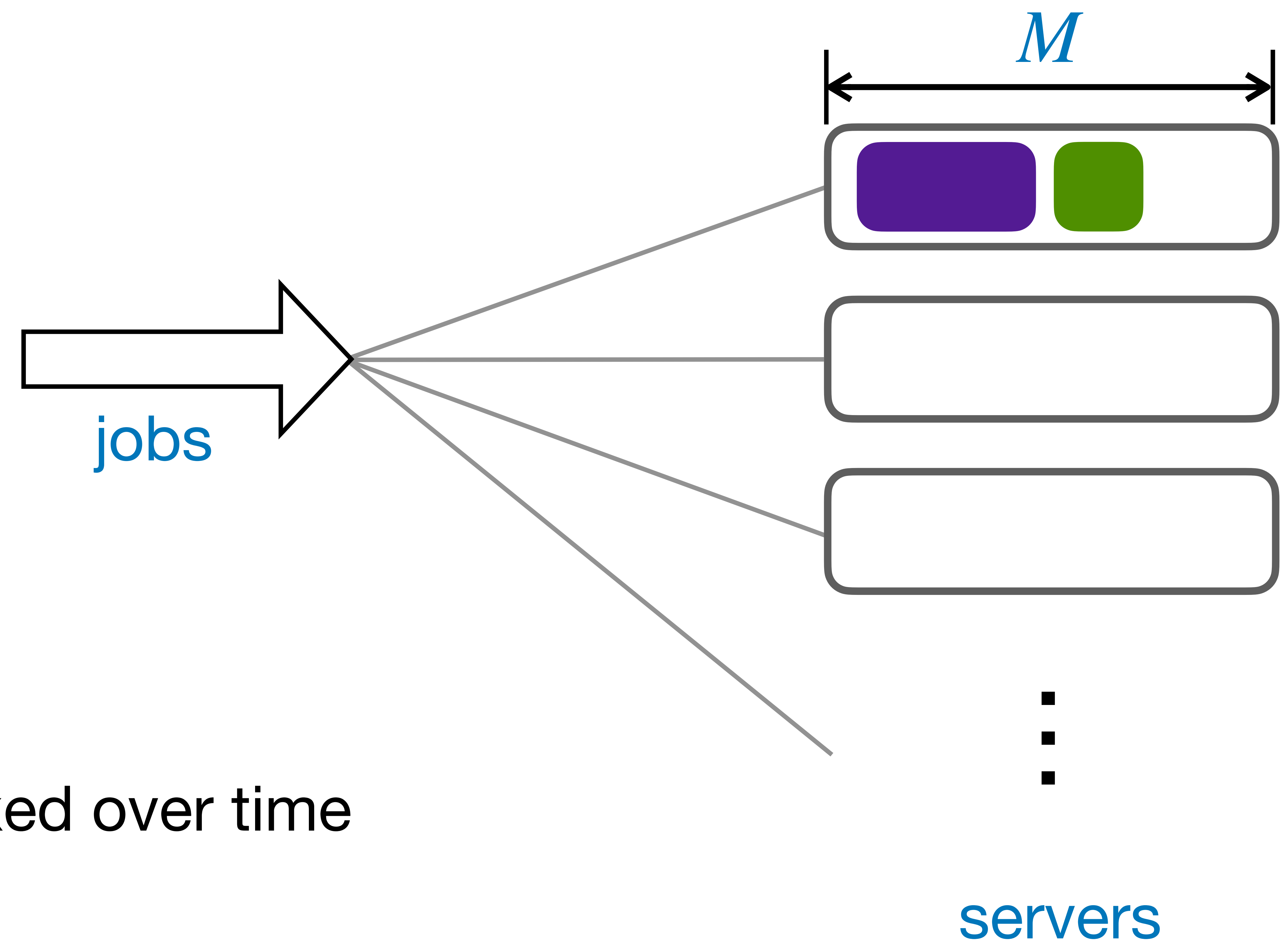


Traditional job model:

- Each job's resource requirement is fixed over time
- Each job departs after a random time

The problem

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



Traditional job model:

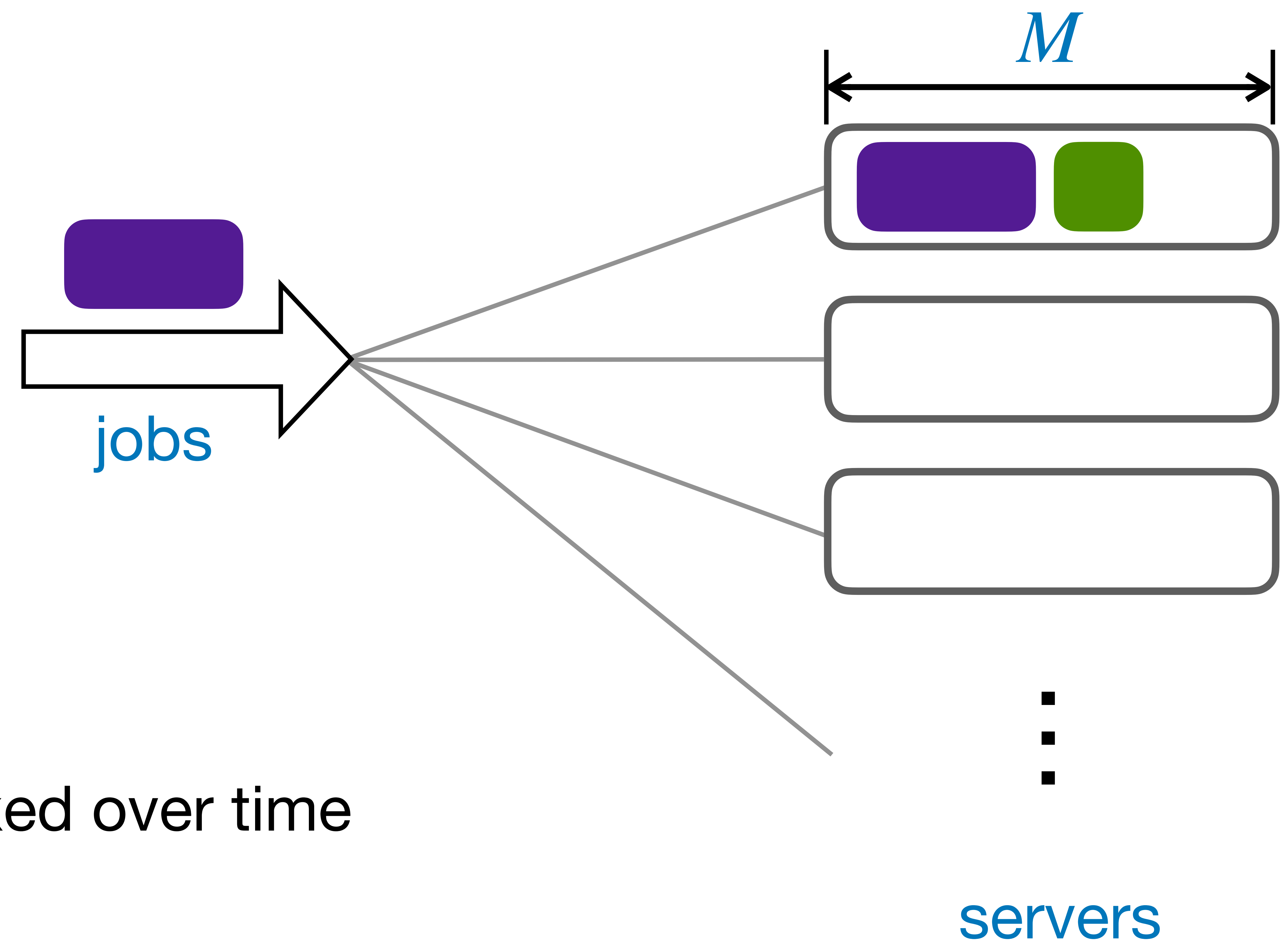
- Each job's resource requirement is fixed over time
- Each job departs after a random time

The problem

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

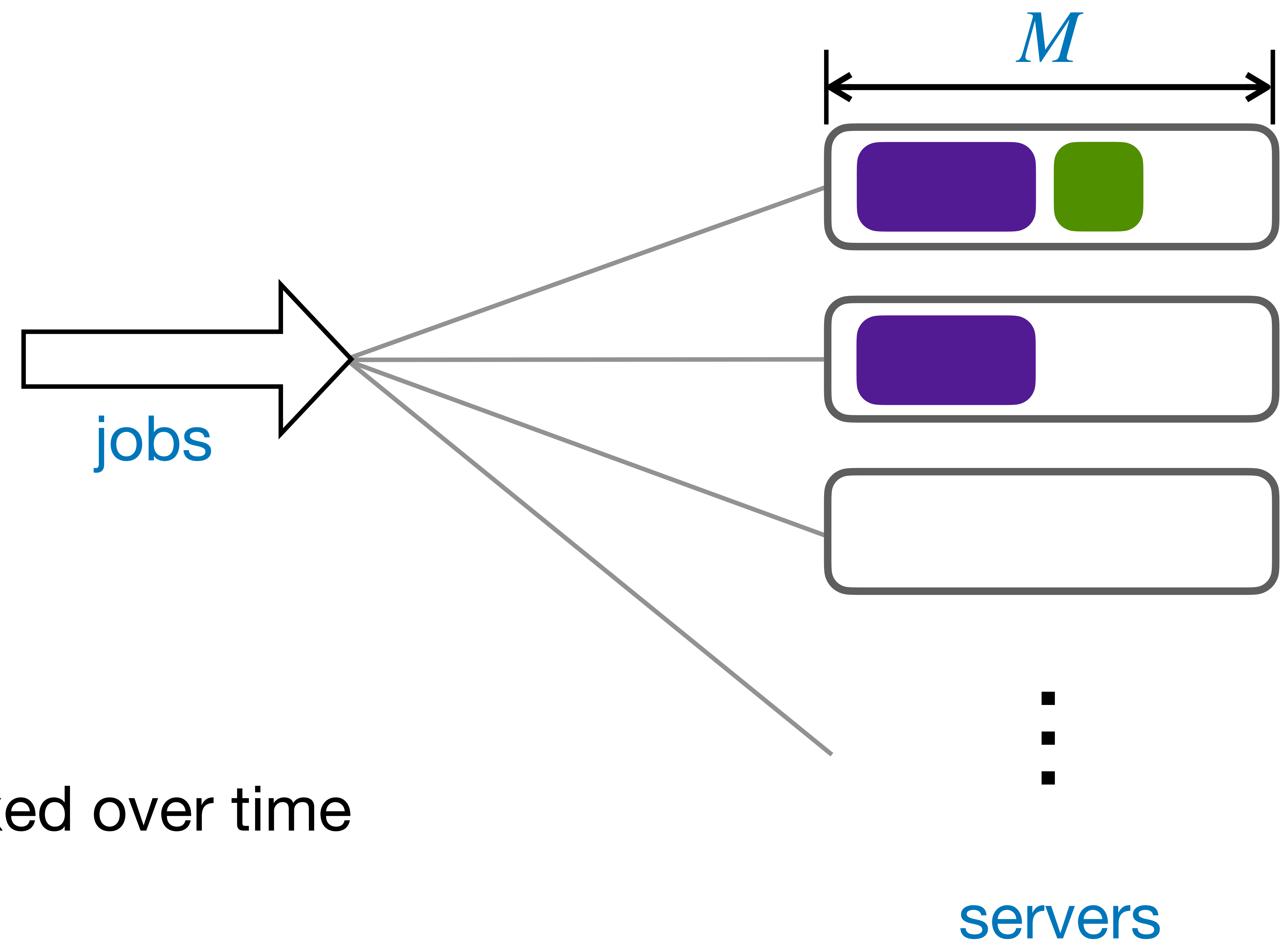
Traditional job model:

- Each job's resource requirement is fixed over time
- Each job departs after a random time



The problem

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

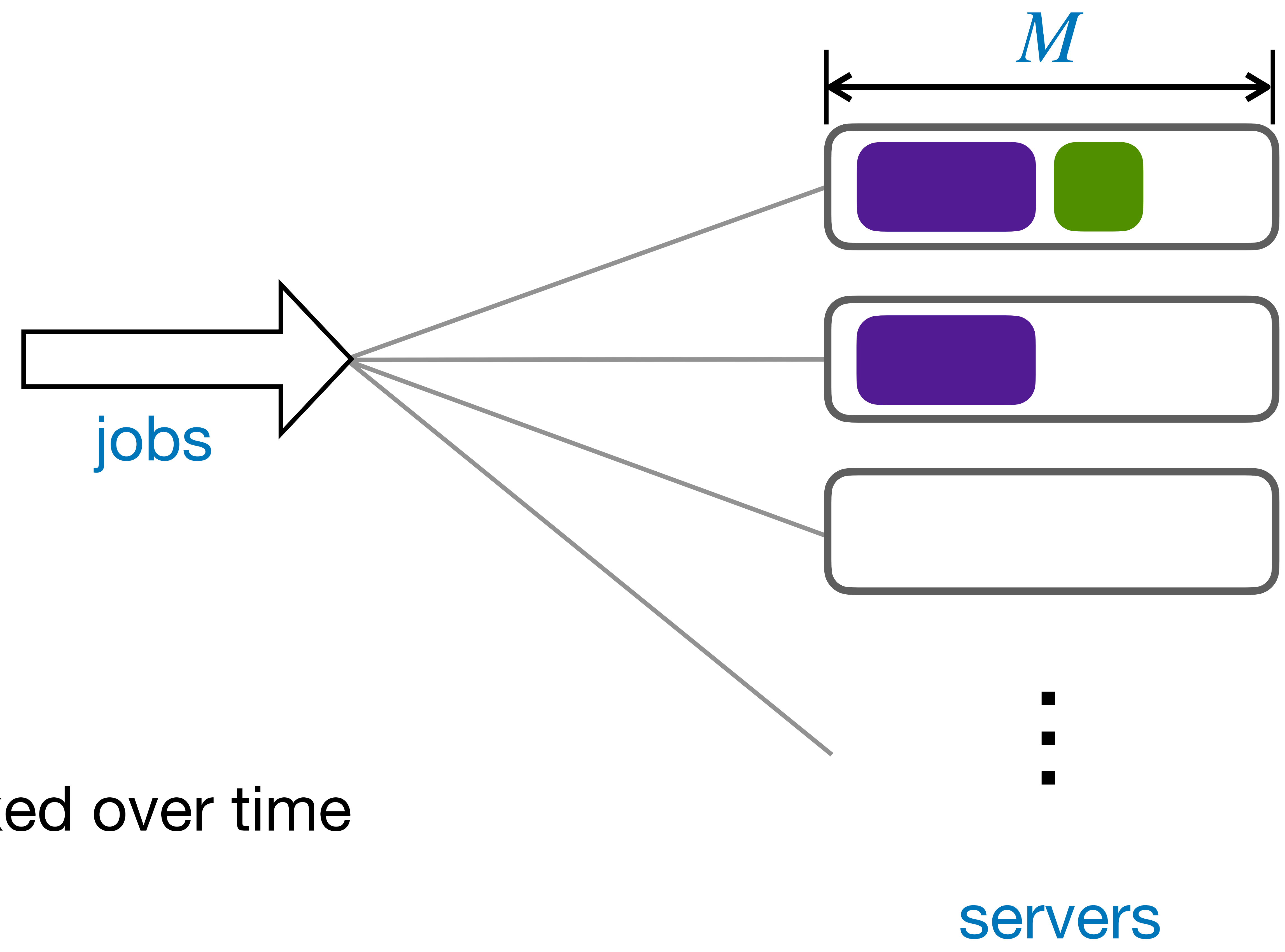


Traditional job model:

- Each job's resource requirement is fixed over time
- Each job departs after a random time

The problem

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



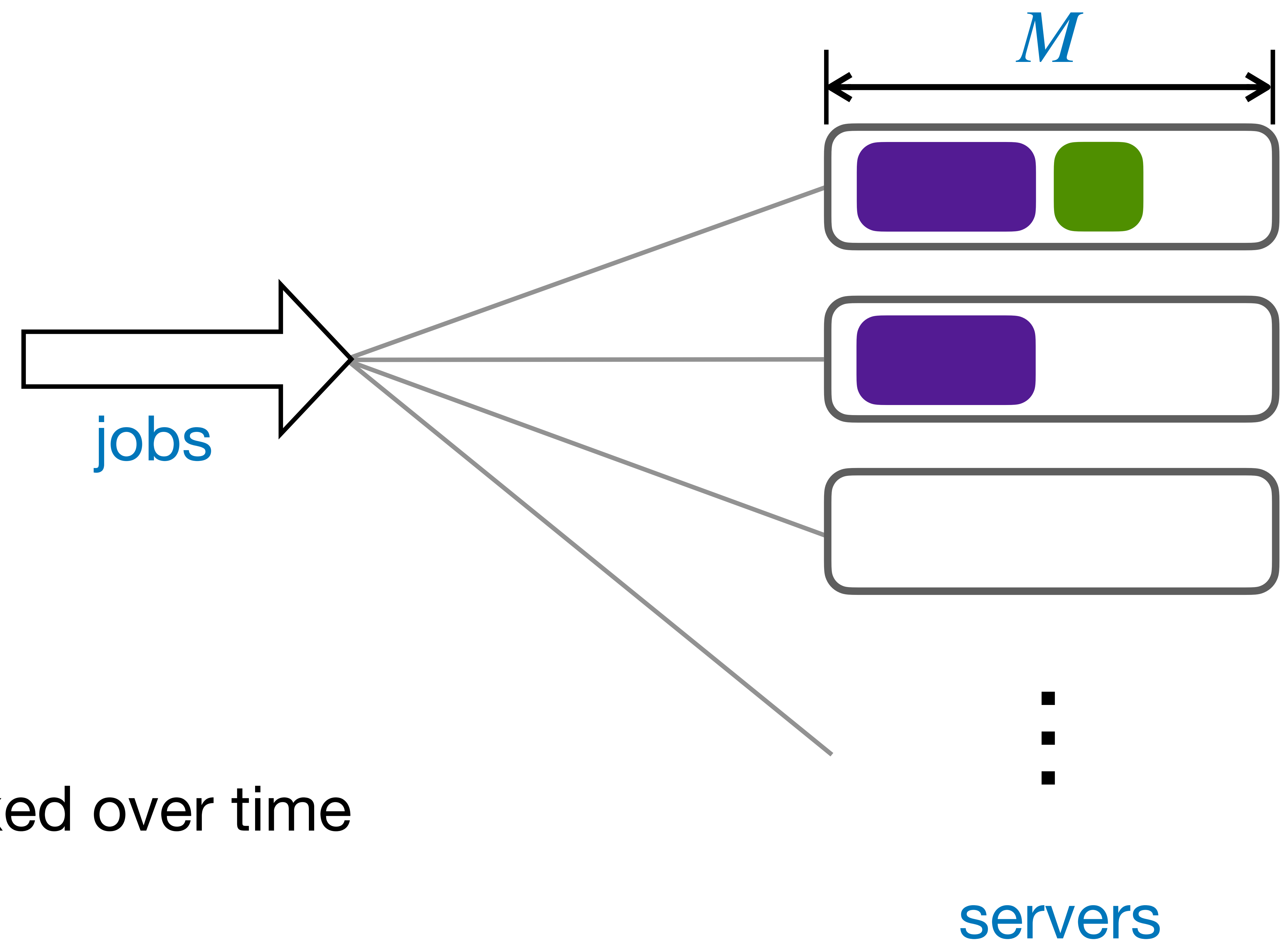
Traditional job model:

- Each job's resource requirement is fixed over time
- Each job departs after a random time

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

The problem

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



Traditional job model:

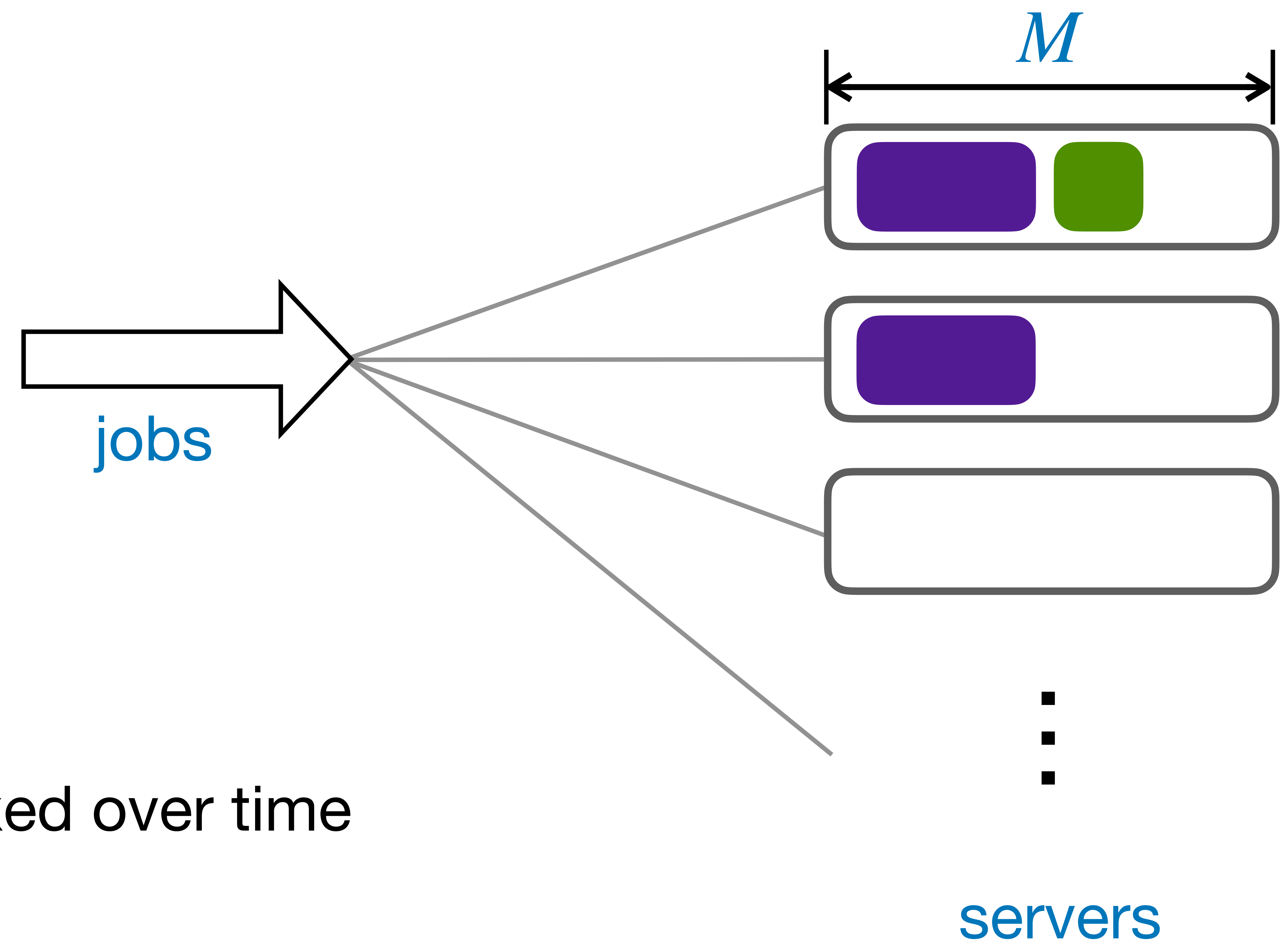
- Each job's resource requirement is fixed over time
- Each job departs after a random time

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

"Stochastic bin-packing in service systems"

The problem

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



A new job model:

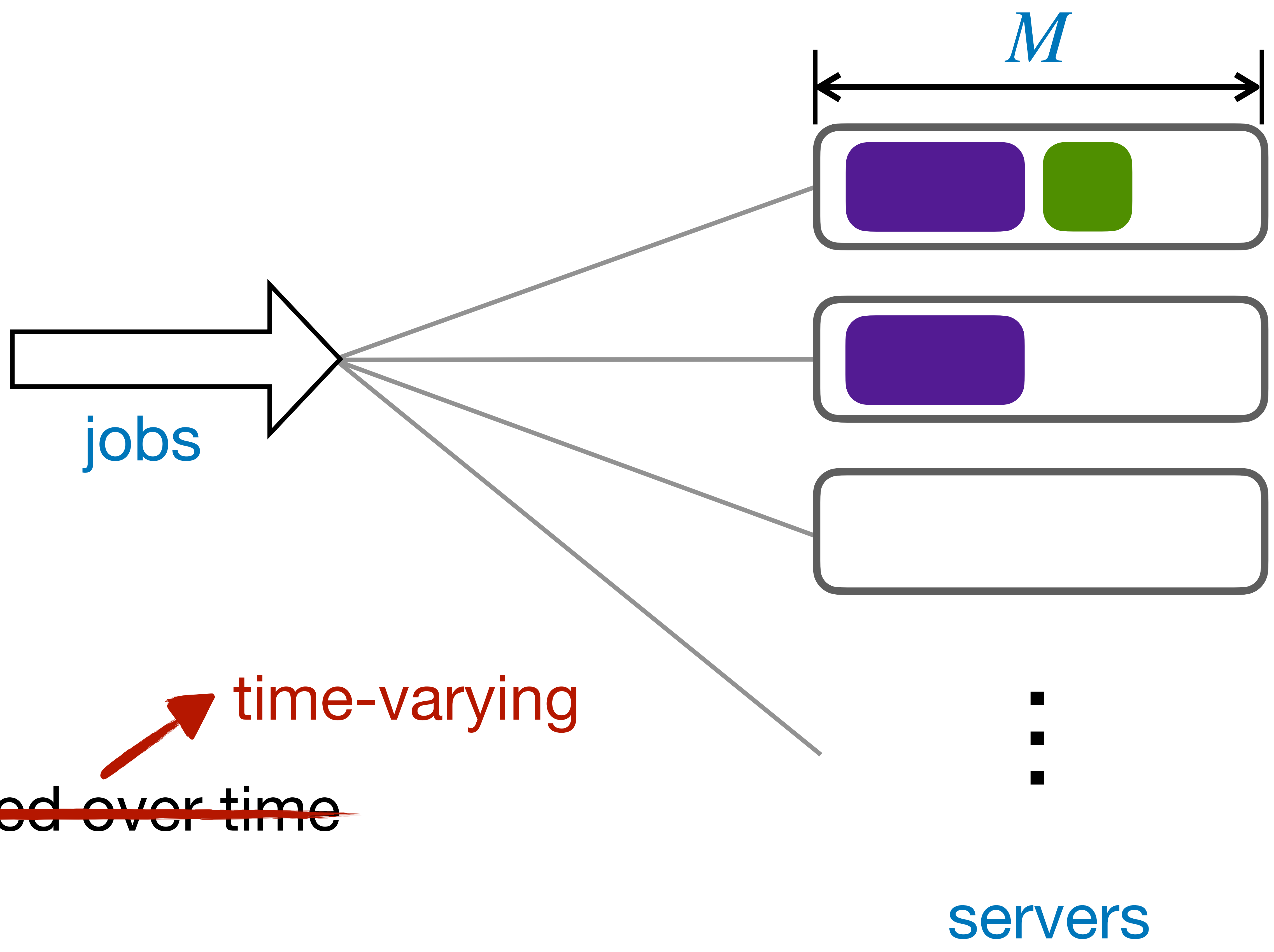
- Each job's resource requirement is fixed over time
- Each job departs after a random time

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

"Stochastic bin-packing in service systems"

The problem

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



A new job model:

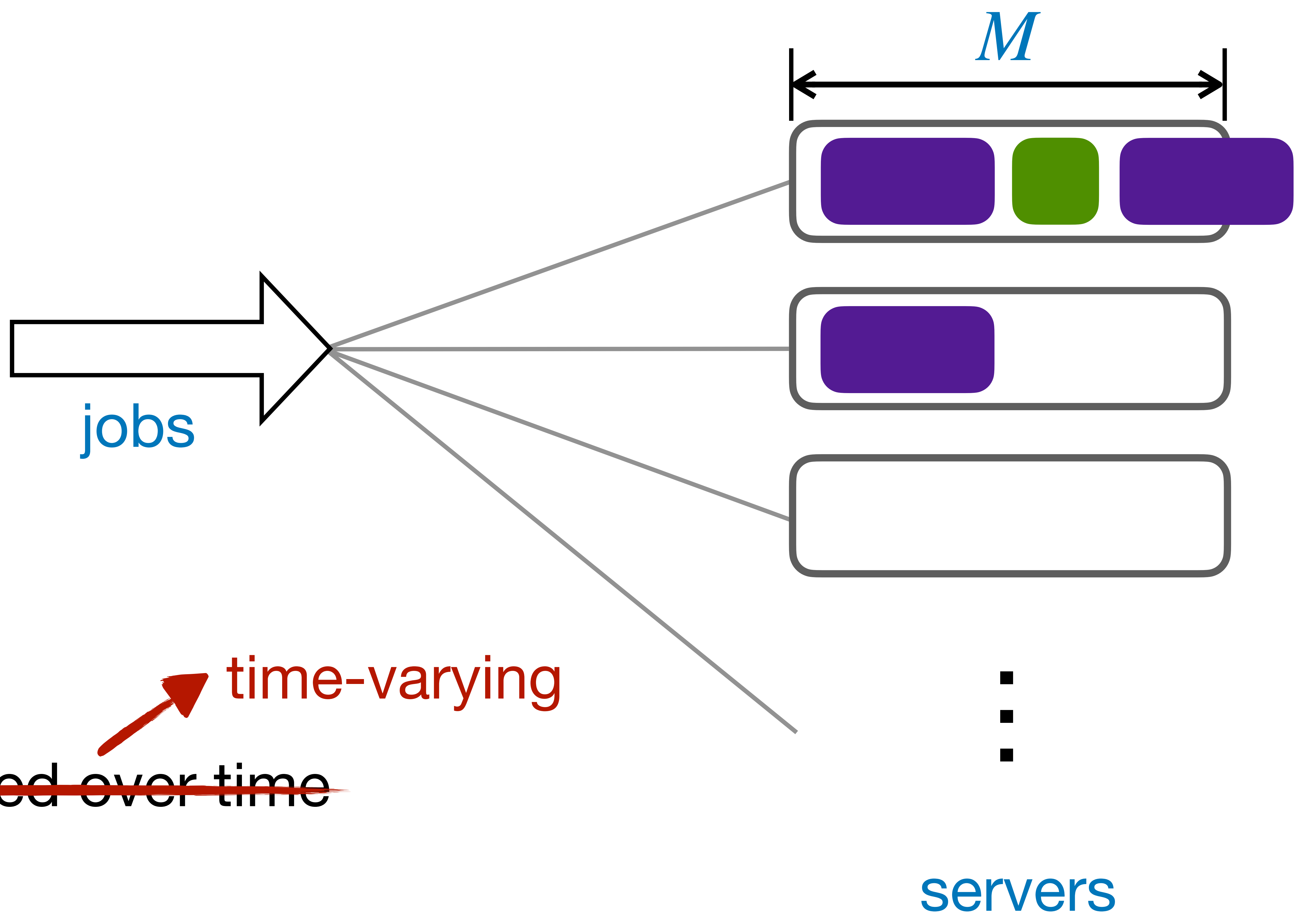
- Each job's resource requirement is ~~fixed over time~~ time-varying
- Each job departs after a random time

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

"Stochastic bin-packing in service systems"

The problem

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



A new job model:

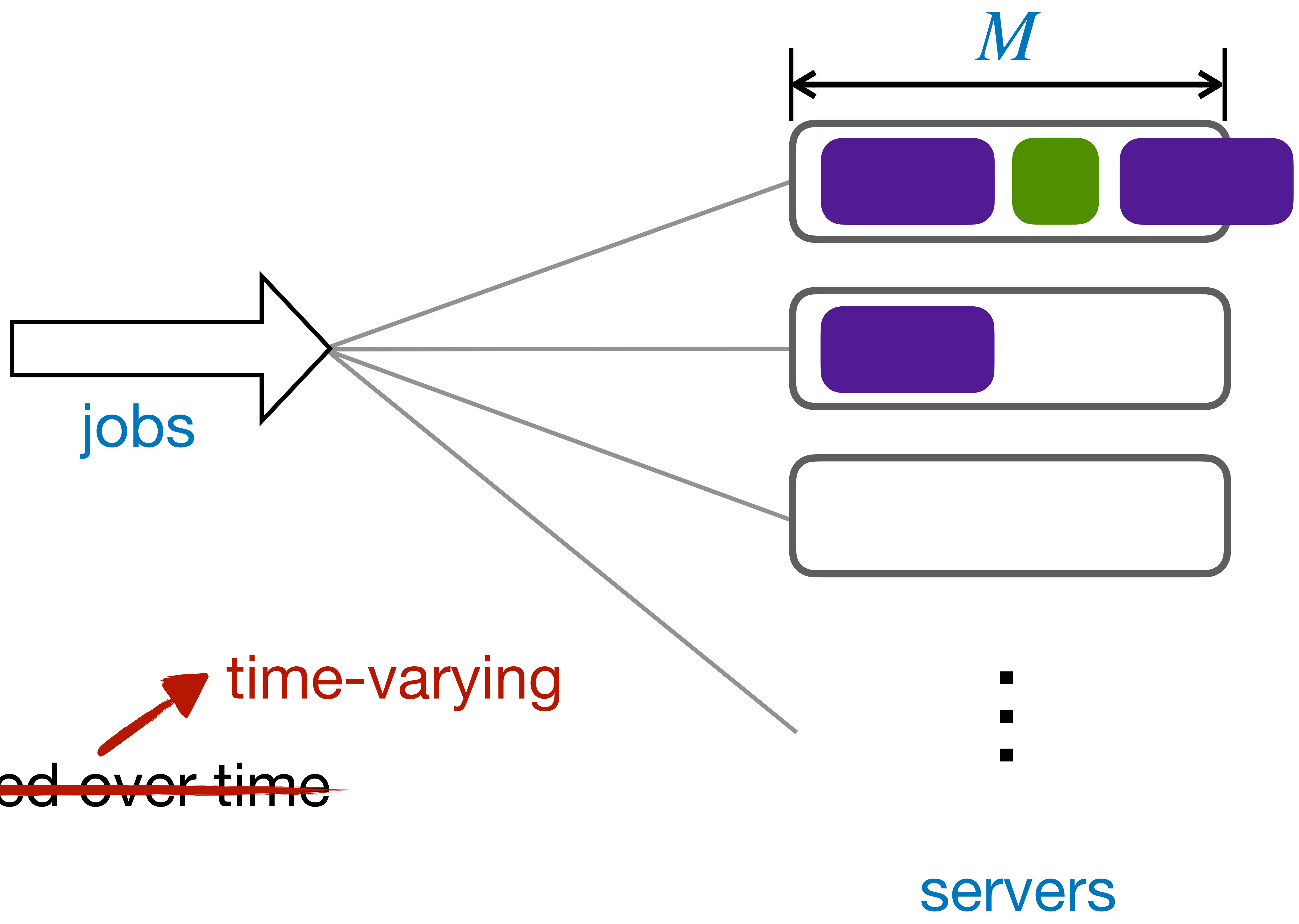
- Each job's resource requirement is ~~fixed over time~~ time-varying
- Each job departs after a random time

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

"Stochastic bin-packing in service systems"

The problem

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



A new job model:

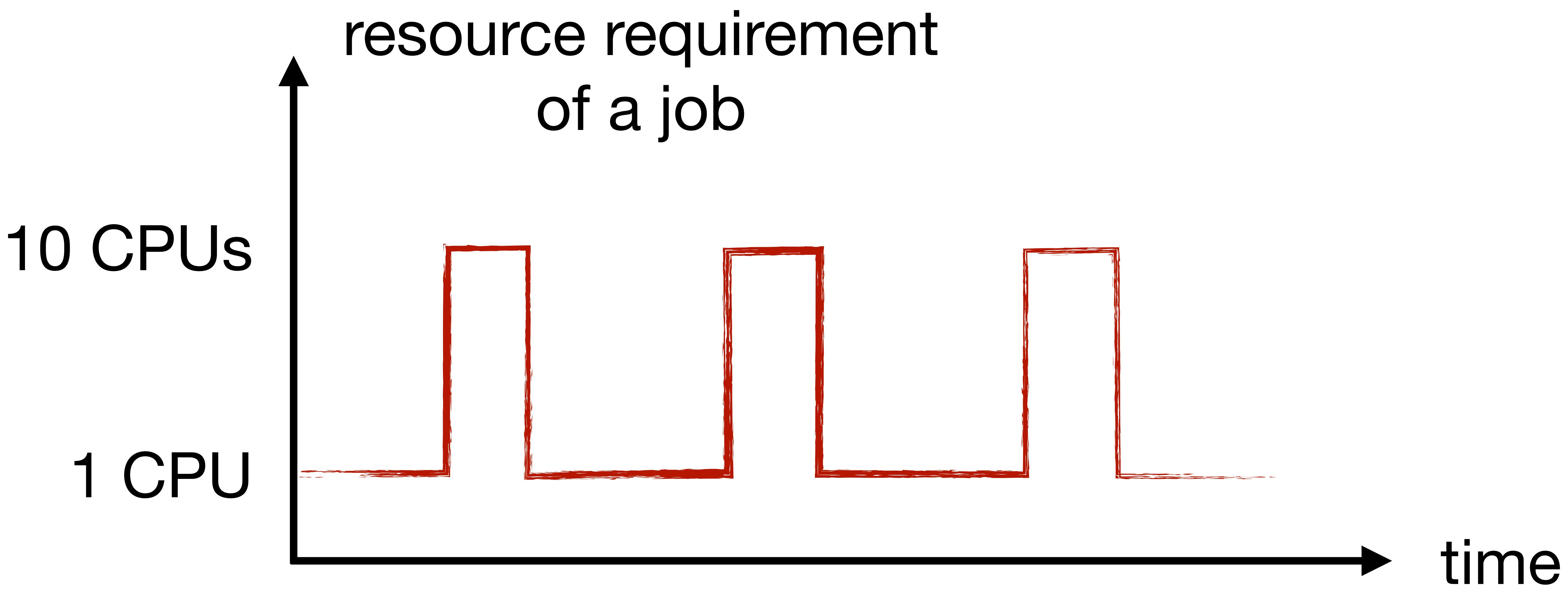
- Each job's resource requirement is ~~fixed over time~~ time-varying
- Each job departs after a random time

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

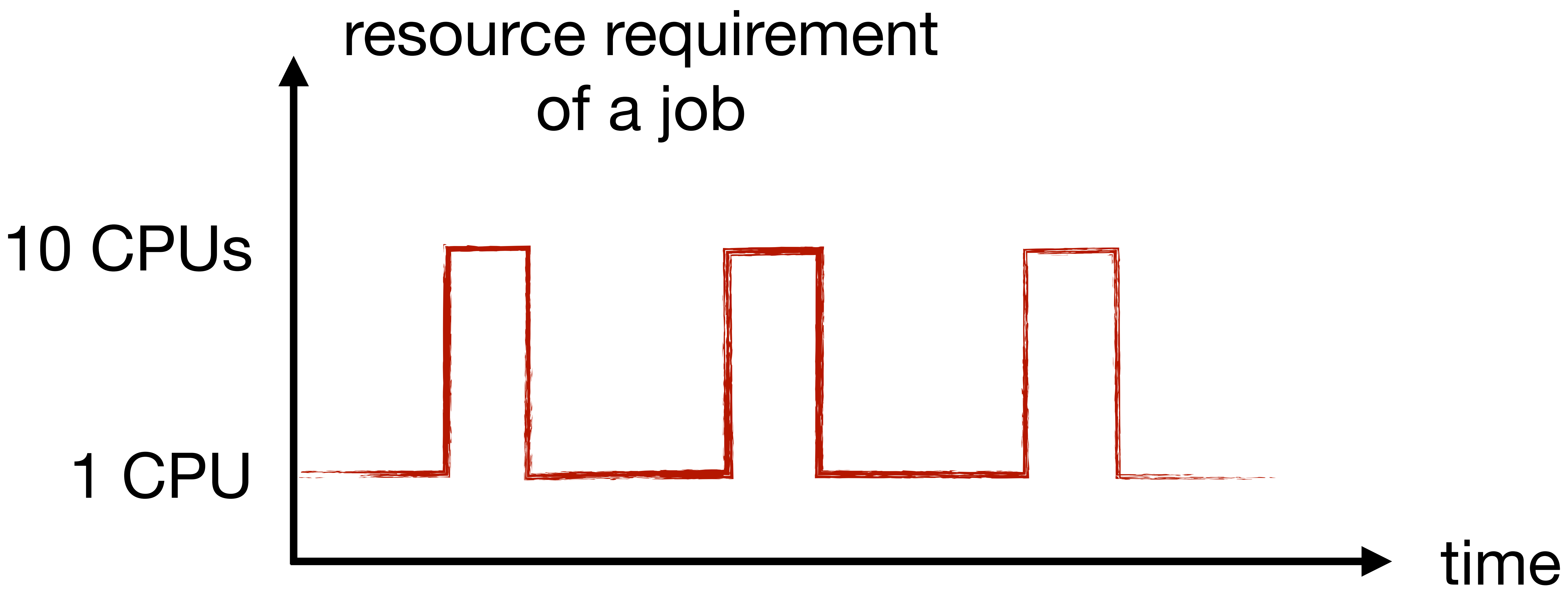
subject to **cost** (resource contention) ≤ budget

"Stochastic bin-packing in service systems"

Why time-varying?

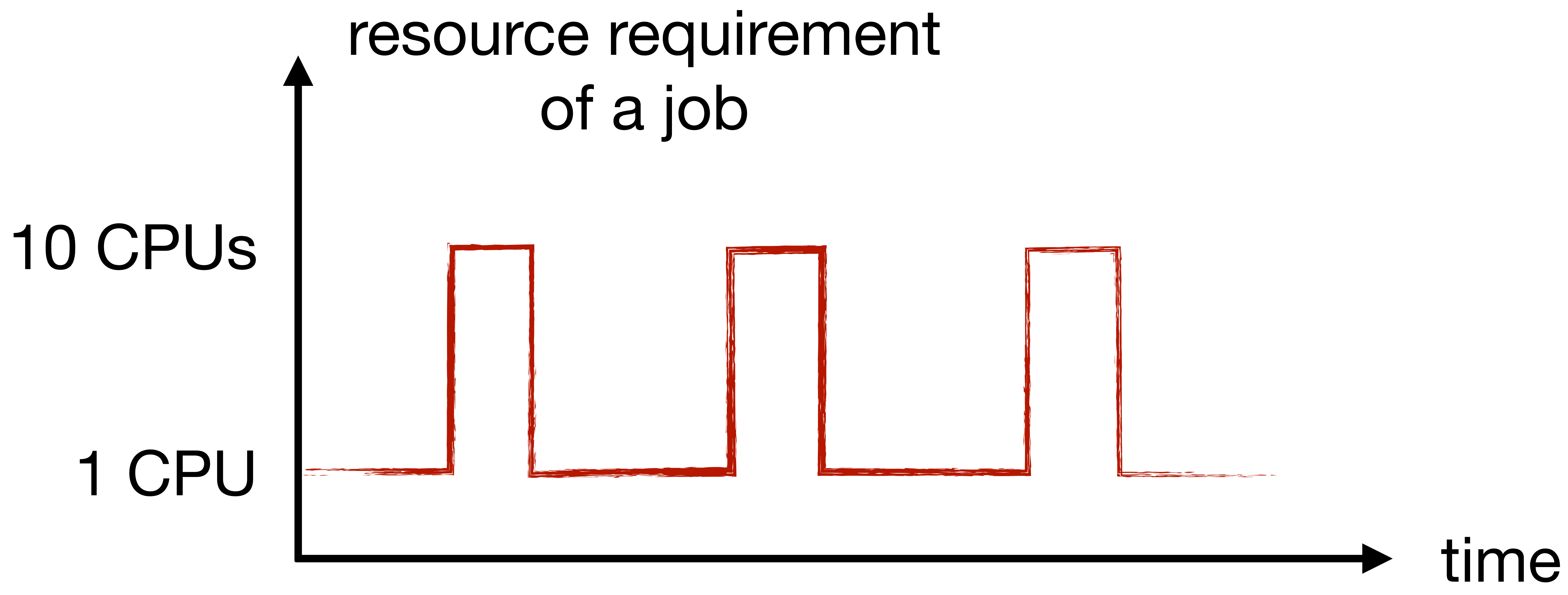


Why time-varying?



- Reserve resources based on peak requirement

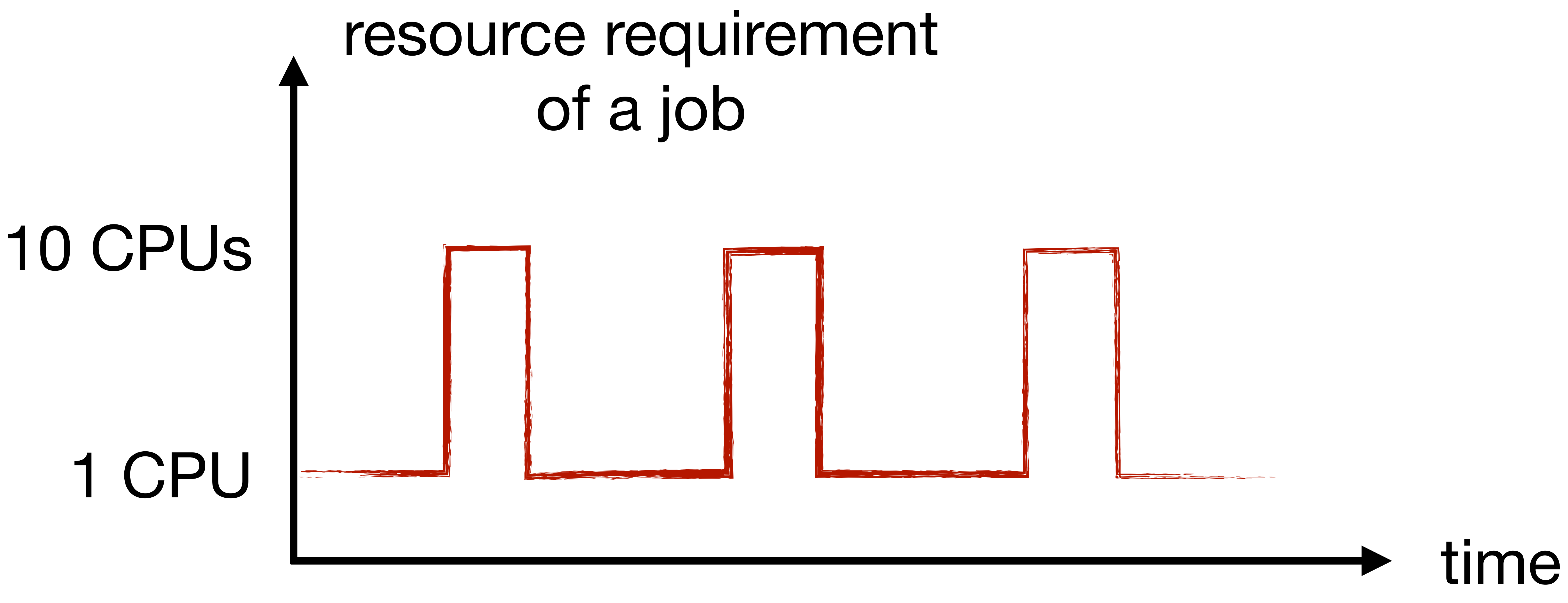
Why time-varying?



- Reserve resources based on peak requirement

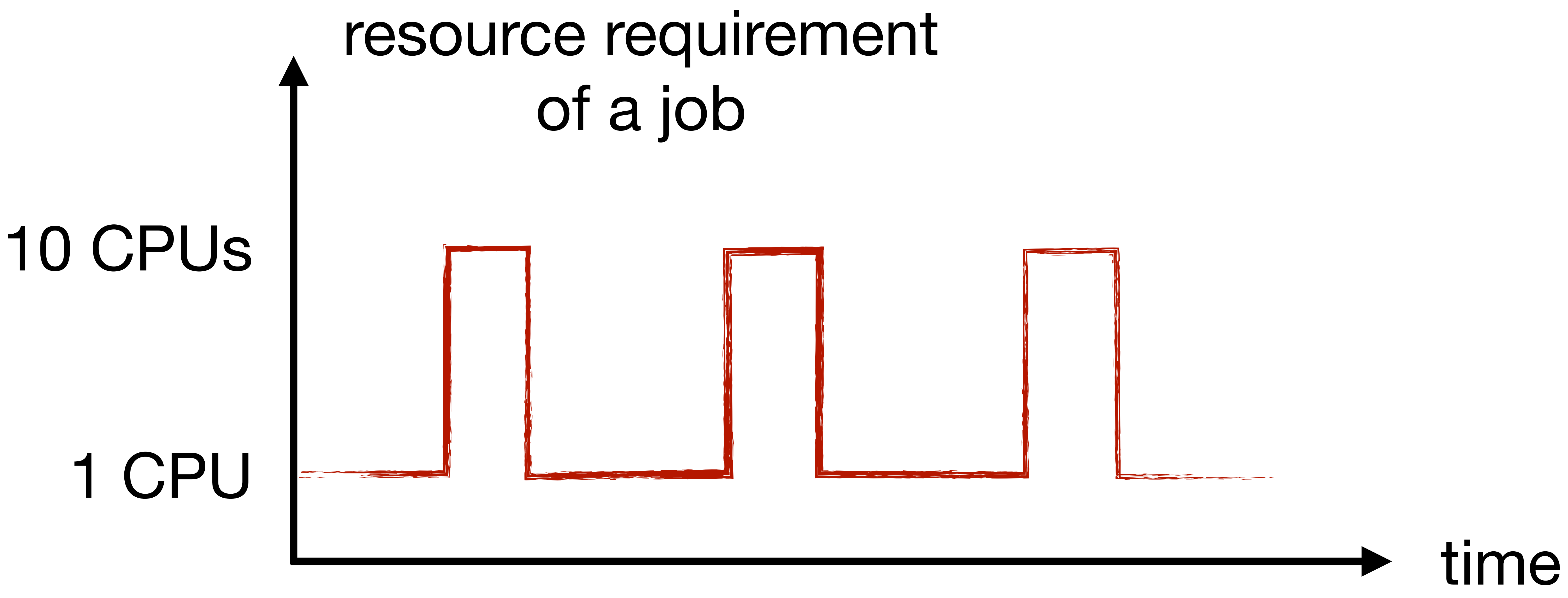
  ➡ low resource utilization on a server

Why time-varying?



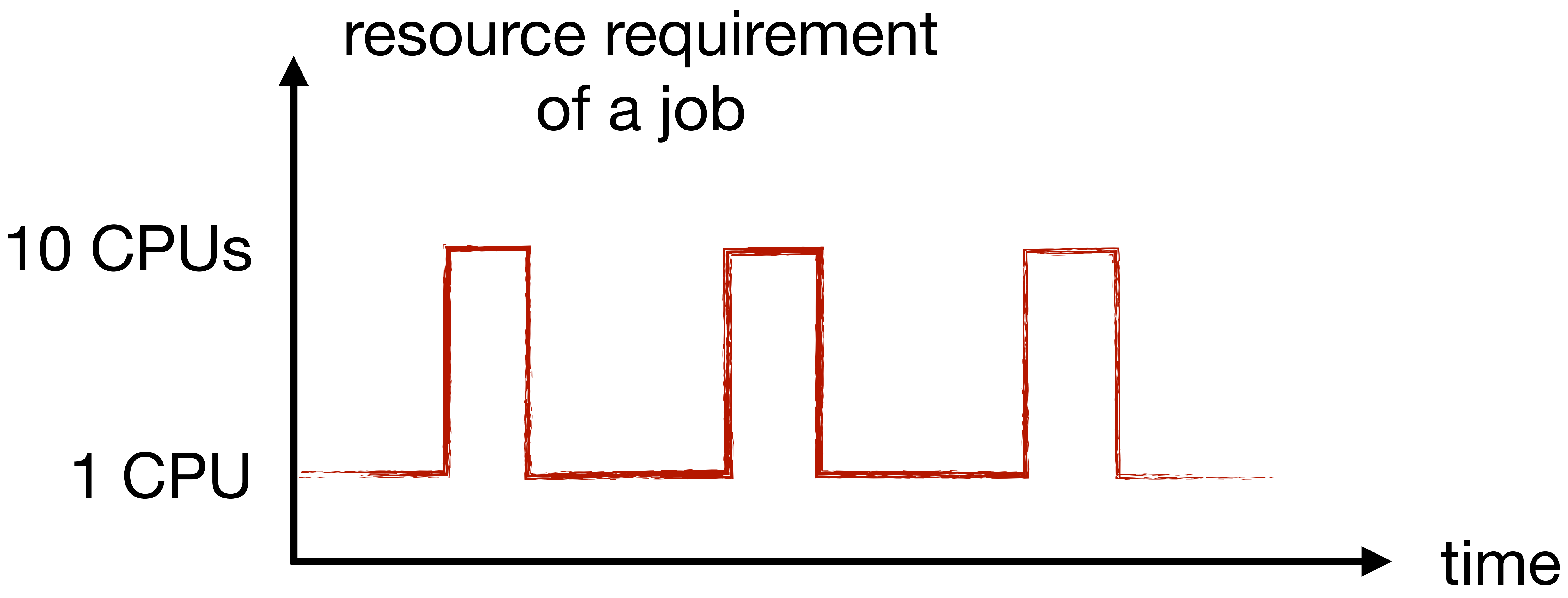
- Reserve resources based on peak requirement
  - ➡ low resource utilization on a server
  - ➡ larger # active servers

Why time-varying?



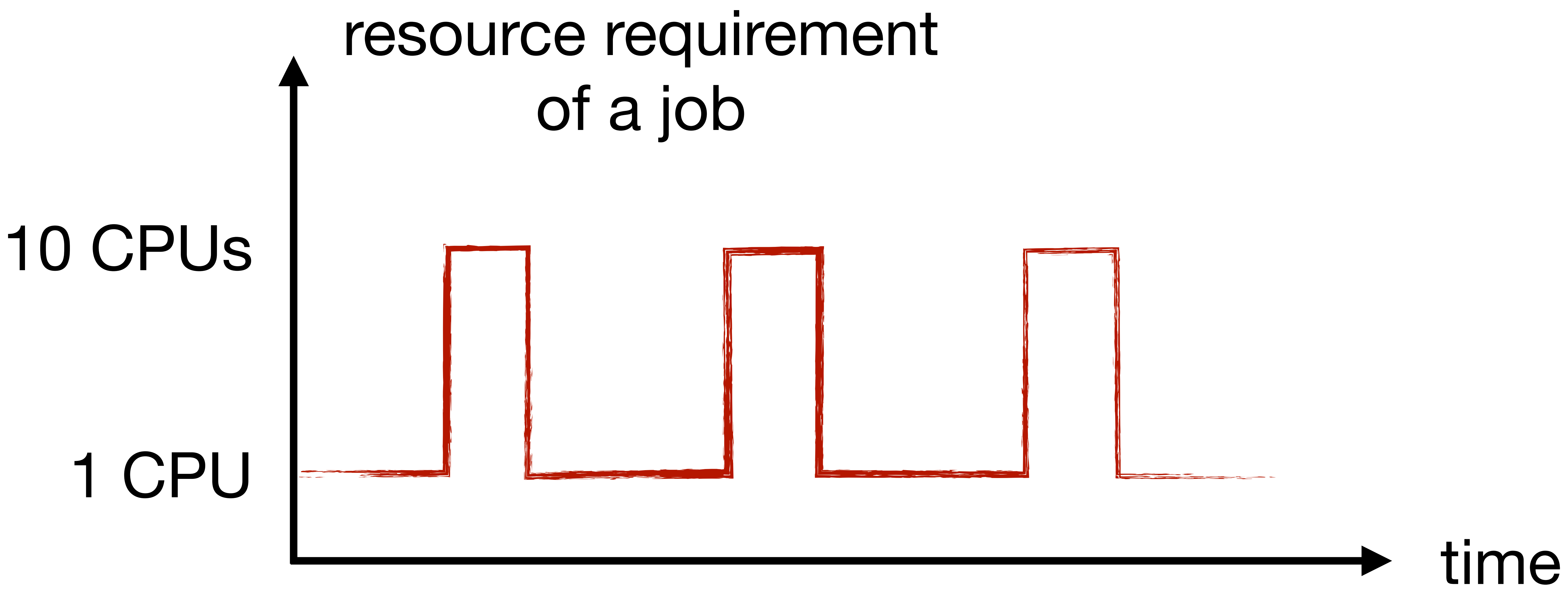
- Reserve resources based on peak requirement
  ➡ low resource utilization on a server
  ➡ larger # active servers
- Overcommit resources on a server

Why time-varying?



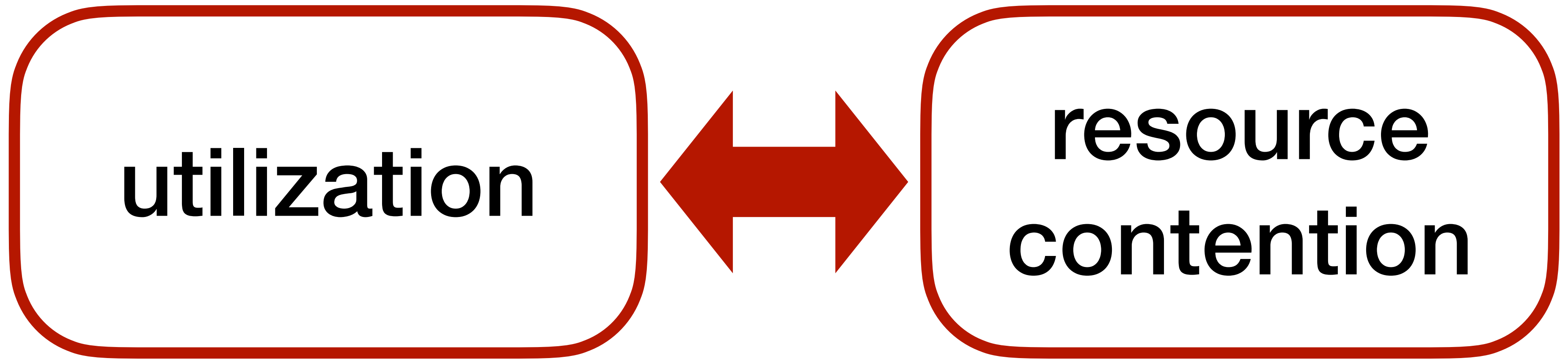
- Reserve resources based on peak requirement
  - ➡ low resource utilization on a server
  - ➡ larger # active servers
- Overcommit resources on a server
  - ➡ possible resource contention

Why time-varying?



- 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:



What's known?

What's known?

- **Stochastic bin-packing in service systems**

What's known?

- **Stochastic bin-packing in service systems**
  - Traditional job model: Asymptotic optimality, no convergence rate

    [Stolyar 2013], [Stolyar and Zhong 2013, 2015, 2021], [Ghaderi, Zhong, and Srikant 2014], [Stolyar 2017]

What’s known?

- **Stochastic bin-packing in service systems**
  - Traditional job model: Asymptotic optimality, no convergence rate

    [Stolyar 2013], [Stolyar and Zhong 2013, 2015, 2021], [Ghaderi, Zhong, and Srikant 2014], [Stolyar 2017]
  - Finite-server model: Maximizing throughput/reward, heavy-traffic optimality, loss model

    [Maguluri, Srikant, and Ying 2012], [Maguluri and Srikant 2013], [Xie et al. 2015], [Ghaderi 2016], [Psychas and Ghaderi 2017, 2018, 2019, 2021, 2021], …

What's known?

- **Stochastic bin-packing in service systems**
  - Traditional job model: Asymptotic optimality, no convergence rate

    [Stolyar 2013], [Stolyar and Zhong 2013, 2015, 2021], [Ghaderi, Zhong, and Srikant 2014], [Stolyar 2017]
  - Finite-server model: Maximizing throughput/reward, heavy-traffic optimality, loss model

    [Maguluri, Srikant, and Ying 2012], [Maguluri and Srikant 2013], [Xie et al. 2015], [Ghaderi 2016], [Psychas and Ghaderi 2017, 2018, 2019, 2021, 2021], …
- **Stochastic bin-packing without job departures**

  [Courcoubetis and Weber 1986, 1990], [Csirik et al. 2006], [Freund and Banerjee 2019], [Gupta and Radovanović 2020], …

What's known?

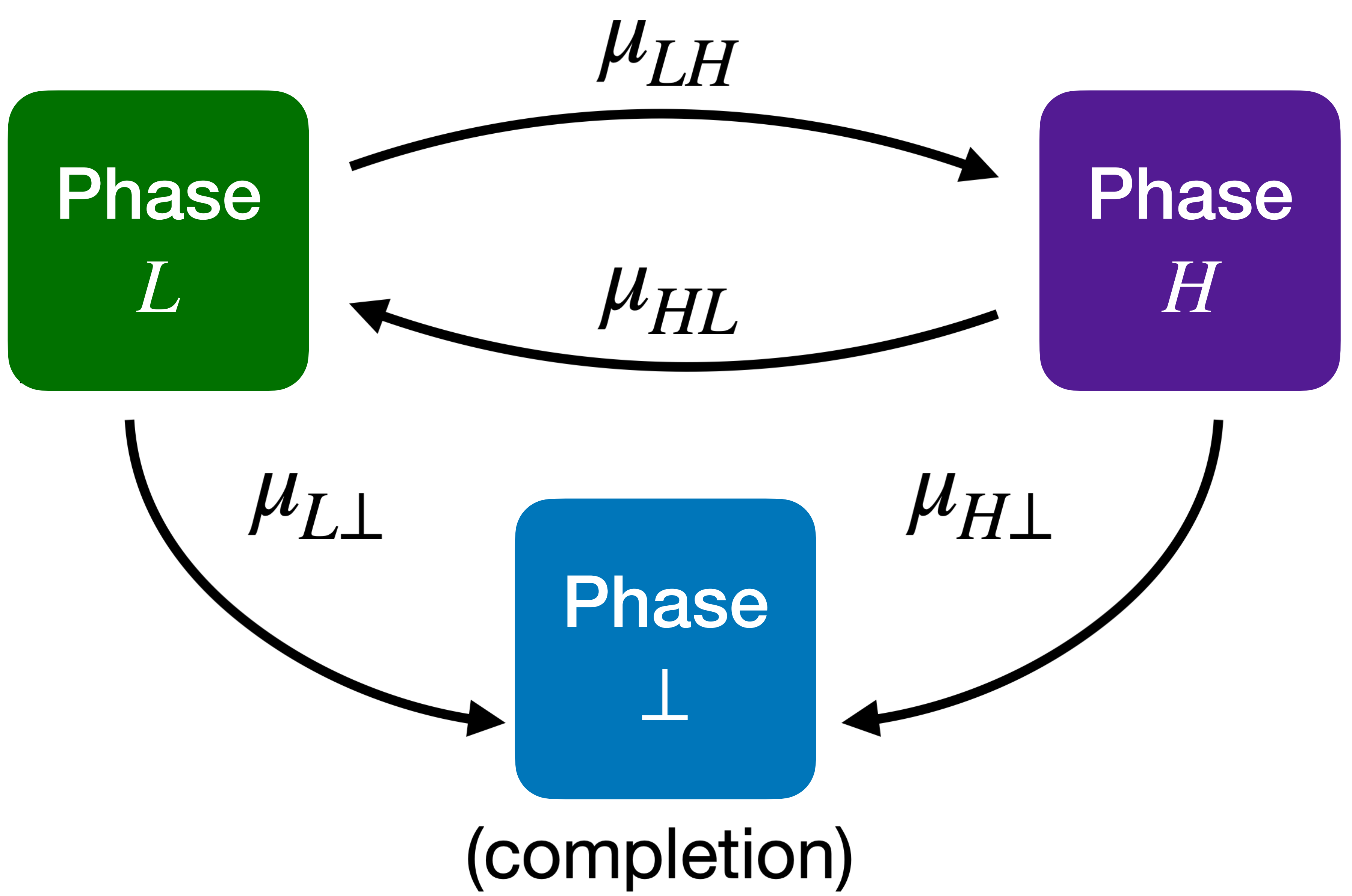
- **Stochastic bin-packing in service systems**
  - Traditional job model: Asymptotic optimality, no convergence rate

    [Stolyar 2013], [Stolyar and Zhong 2013, 2015, 2021], [Ghaderi, Zhong, and Srikant 2014], [Stolyar 2017]
  - Finite-server model: Maximizing throughput/reward, heavy-traffic optimality, loss model

    [Maguluri, Srikant, and Ying 2012], [Maguluri and Srikant 2013], [Xie et al. 2015], [Ghaderi 2016], [Psychas and Ghaderi 2017, 2018, 2019, 2021, 2021], …
- **Stochastic bin-packing without job departures**

  [Courcoubetis and Weber 1986, 1990], [Csirik et al. 2006], [Freund and Banerjee 2019], [Gupta and Radovanović 2020], …
- **Classical bin-packing: Vast literature**

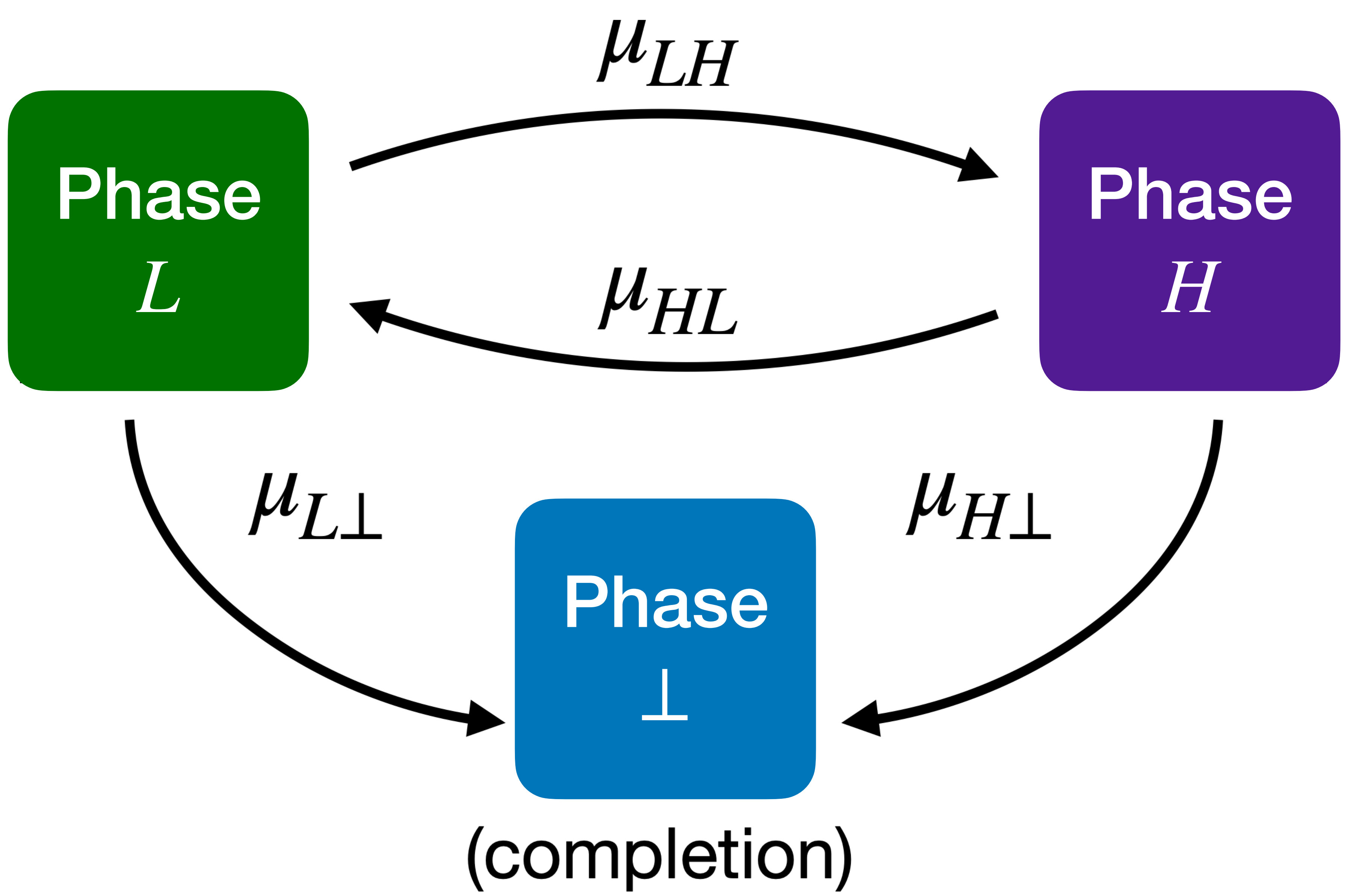
More details on the job model



Example MC

More details on the job model

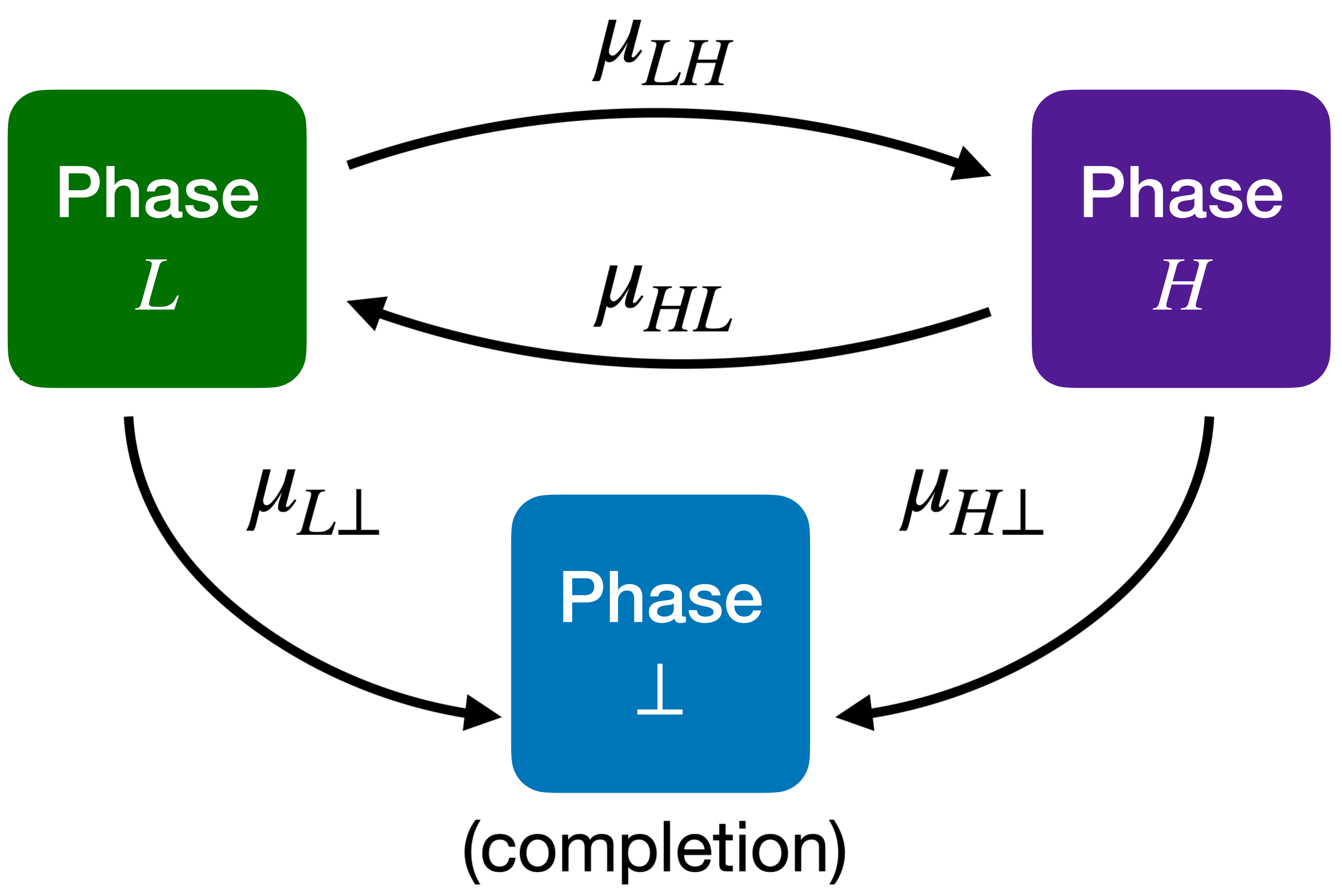
- Resource requirement of a job evolves over time following a Markov chain



Example MC

More details on the job model

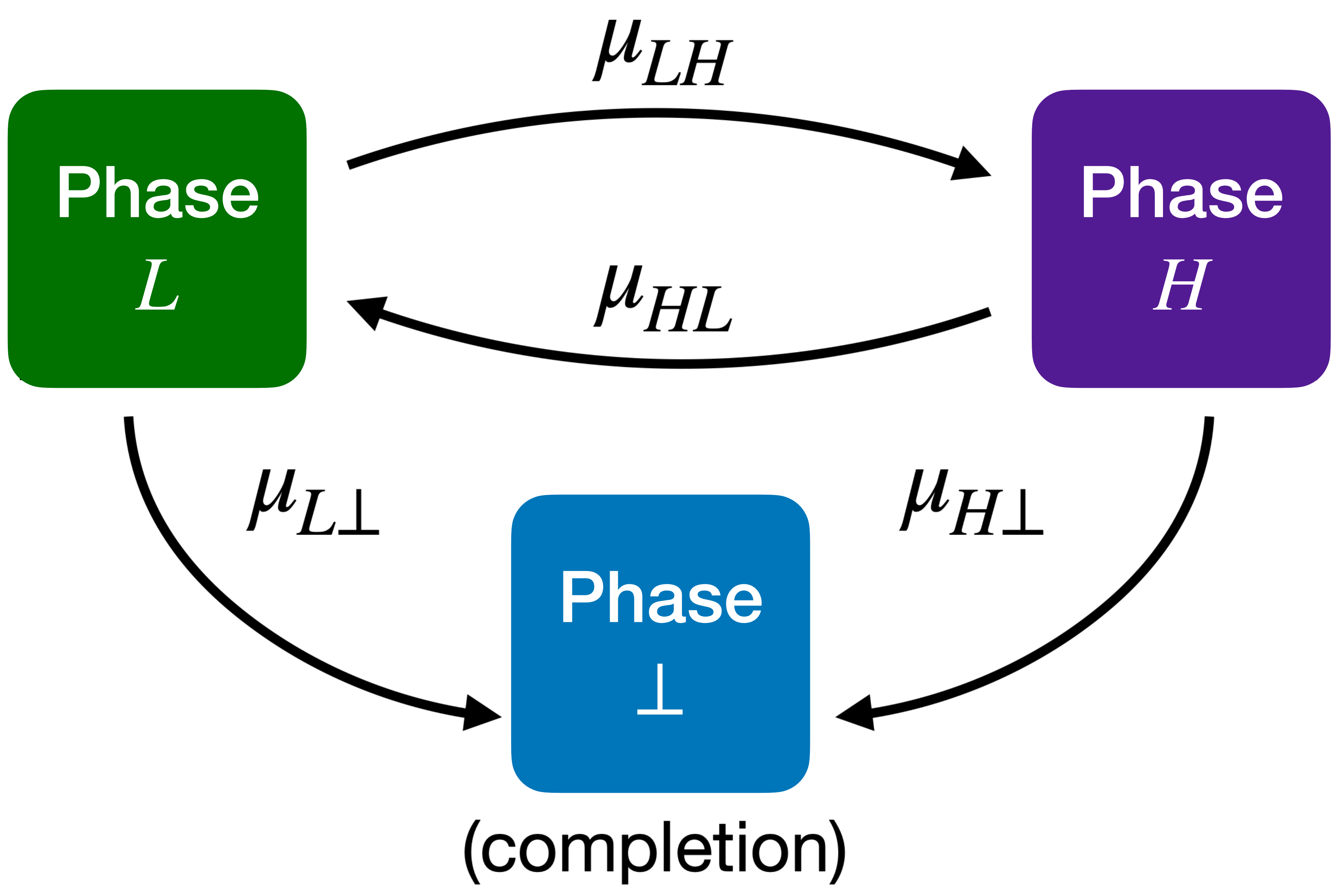
- Resource requirement of a job evolves over time following a Markov chain
- Initial job type follows an initial distribution



Example MC

More details on the job model

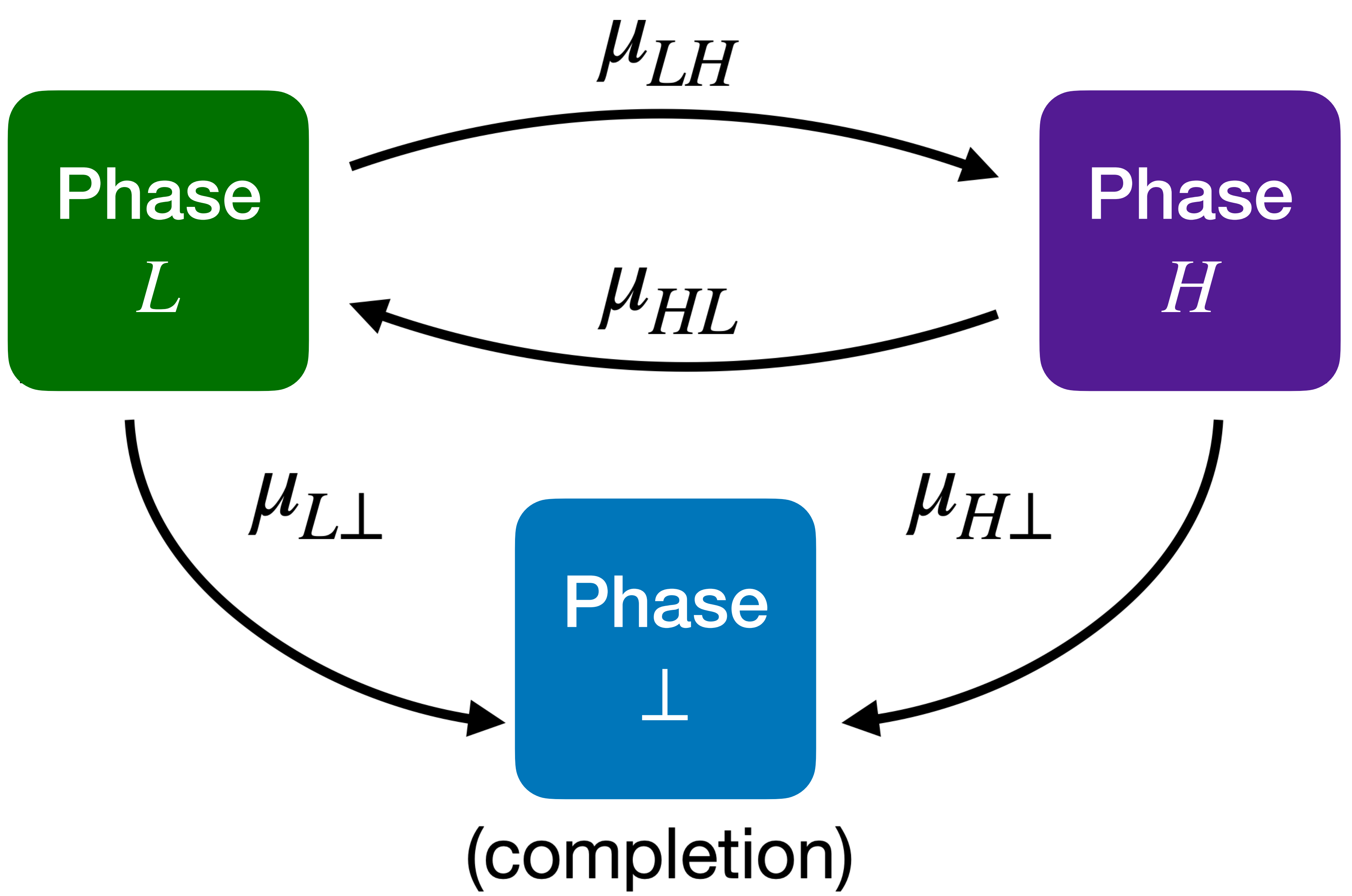
- Resource requirement of a job evolves over time following a Markov chain
- Initial job type follows an initial distribution
- MCs of jobs are independent of each other, and they are exogenous (not affected by resource contention)



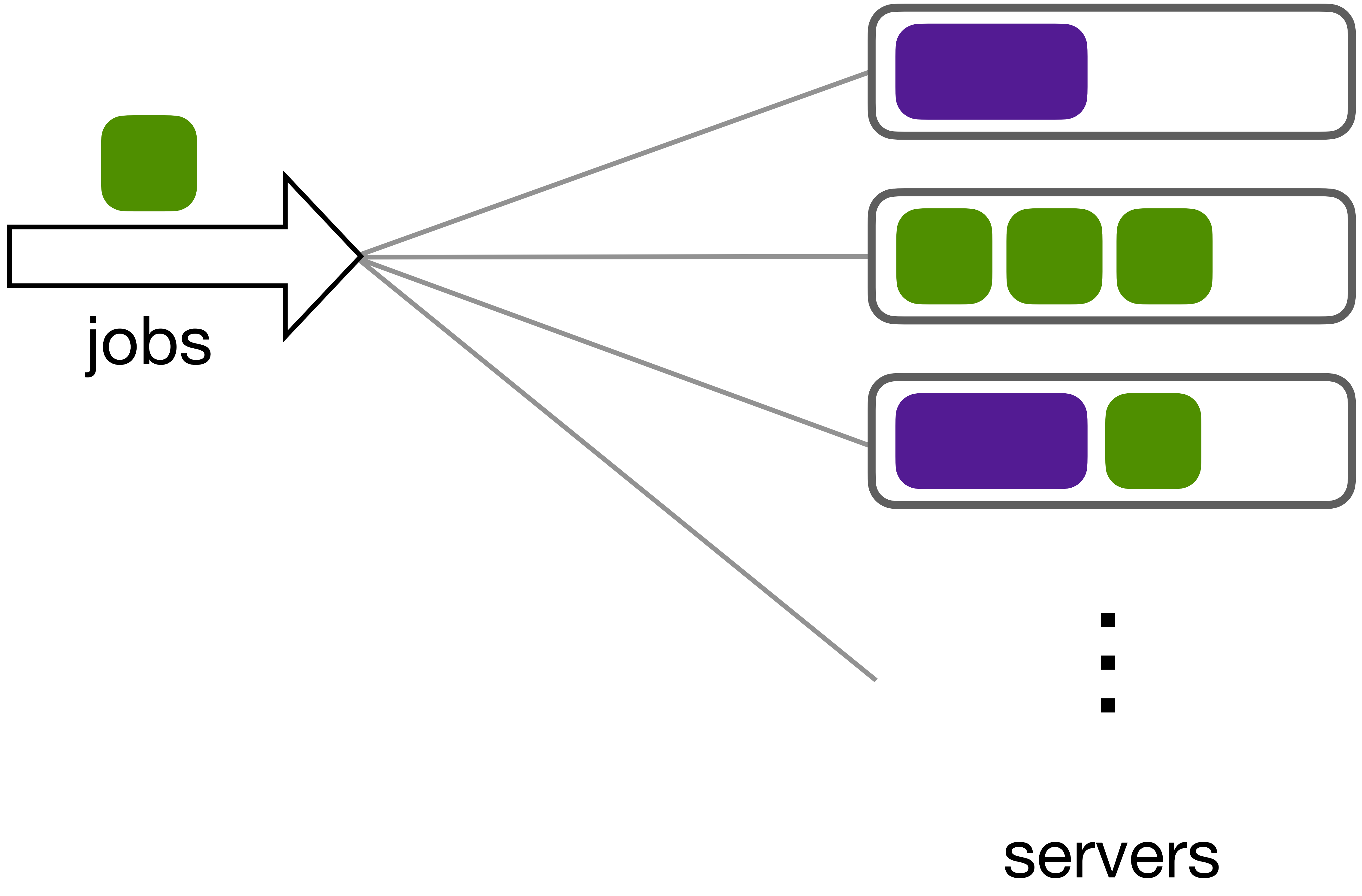
Example MC

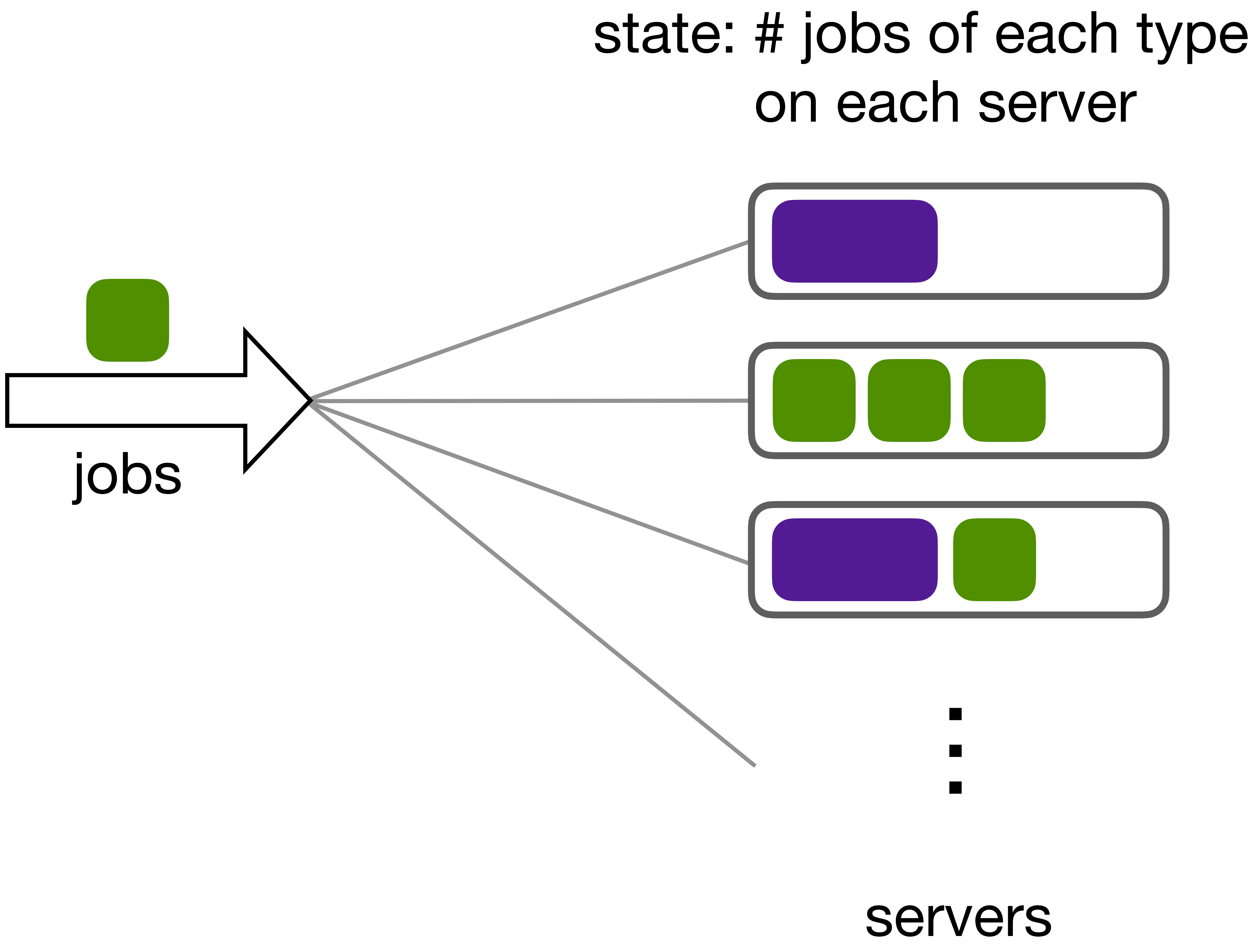
More details on the job model

- 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



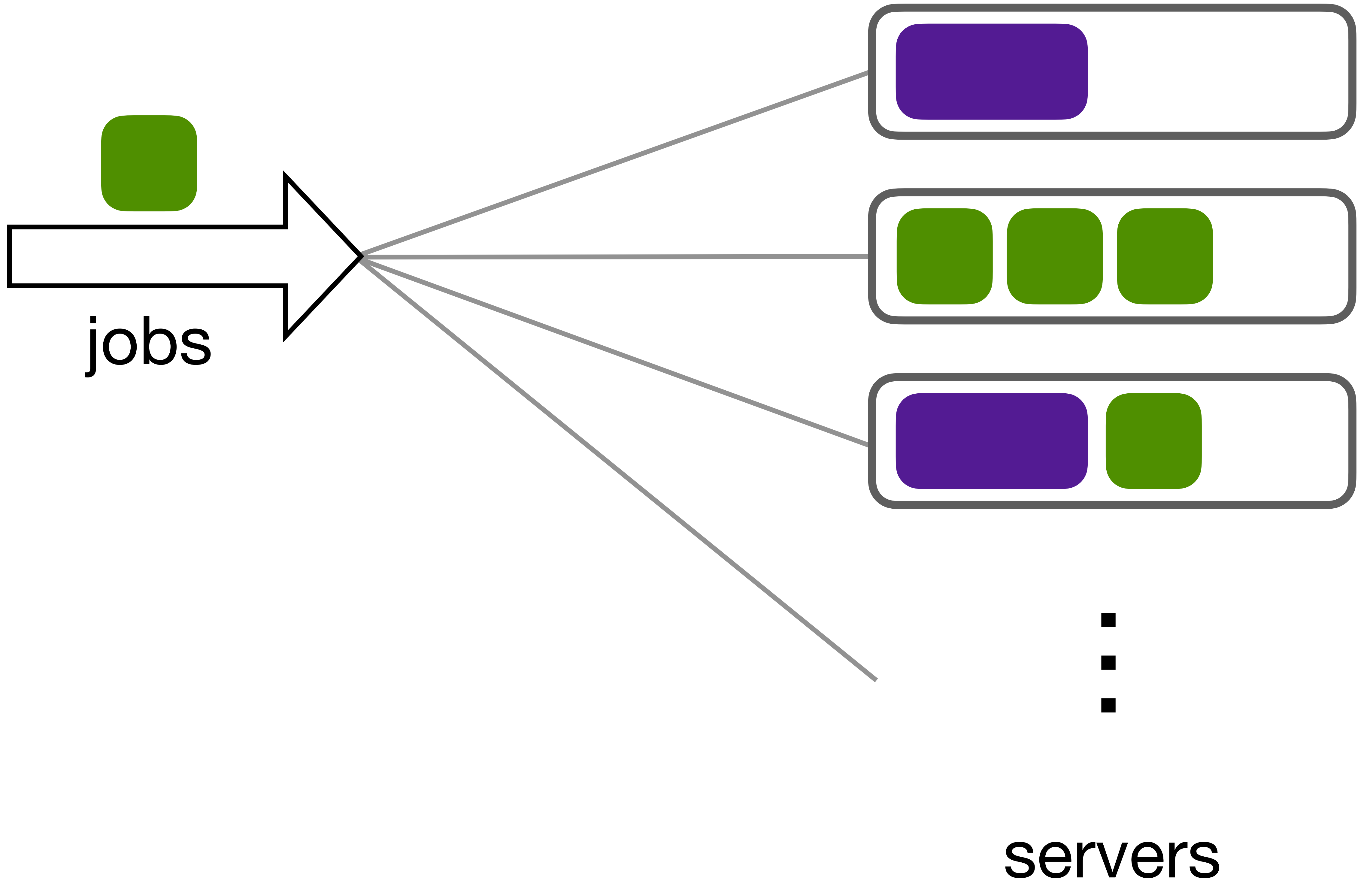
Example MC


jobs
servers


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

state space is large!

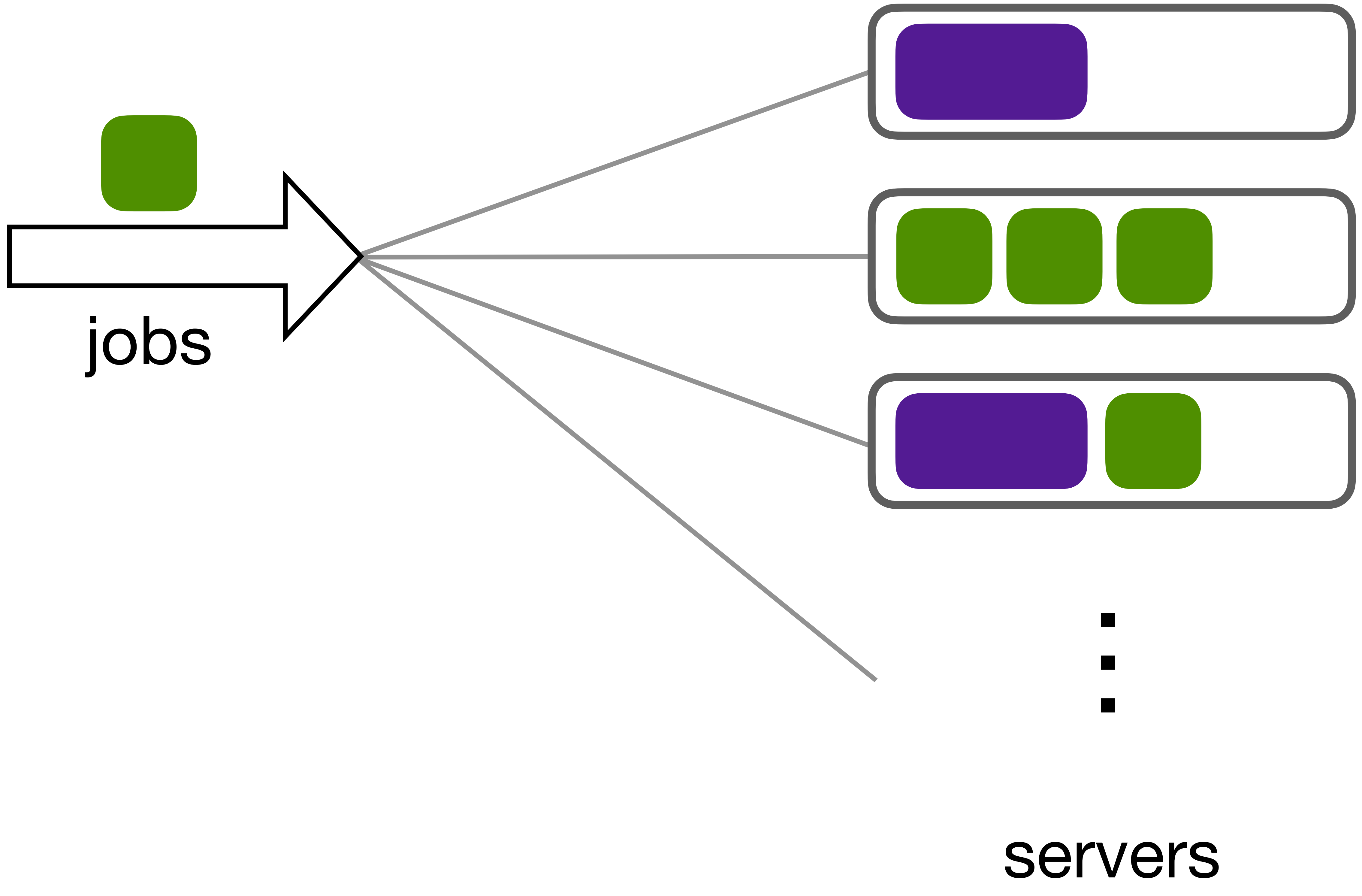
state: # jobs of each type on each server



Reducing dimensionality

state space is large!

state: # jobs of each type on each server

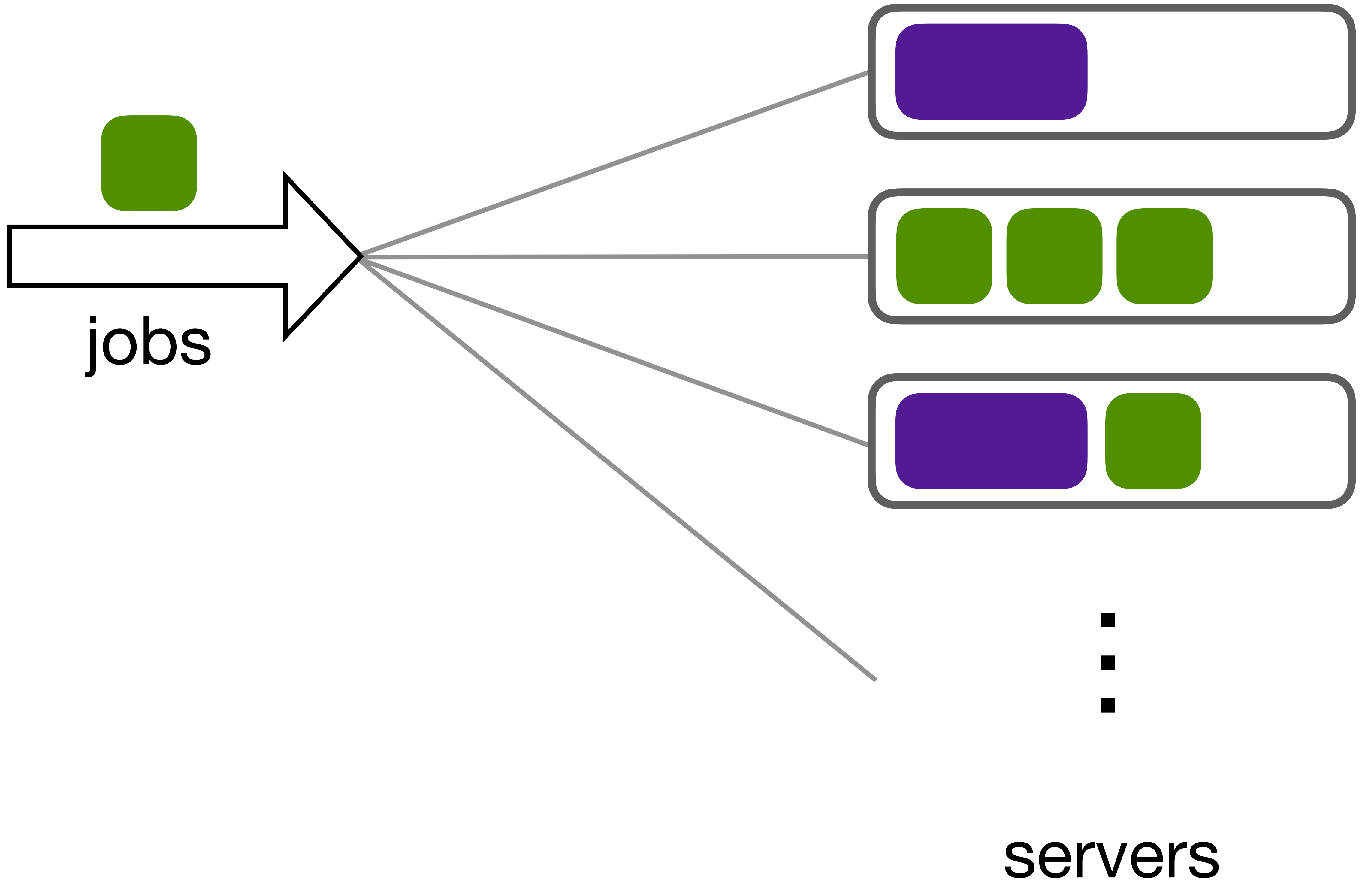


Reducing dimensionality

state space is large!

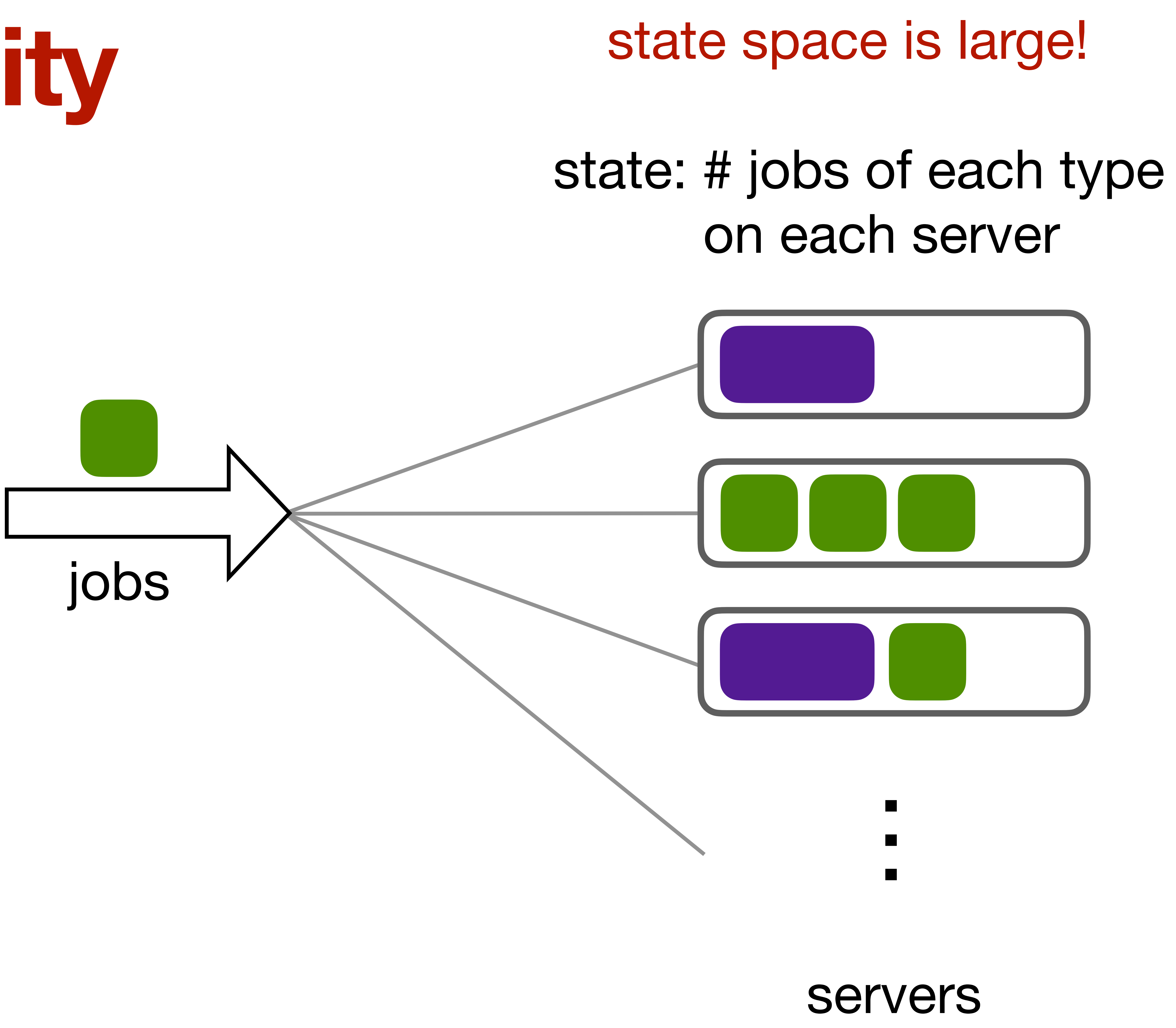
state: # jobs of each type on each server

- Server-by-server evaluation:



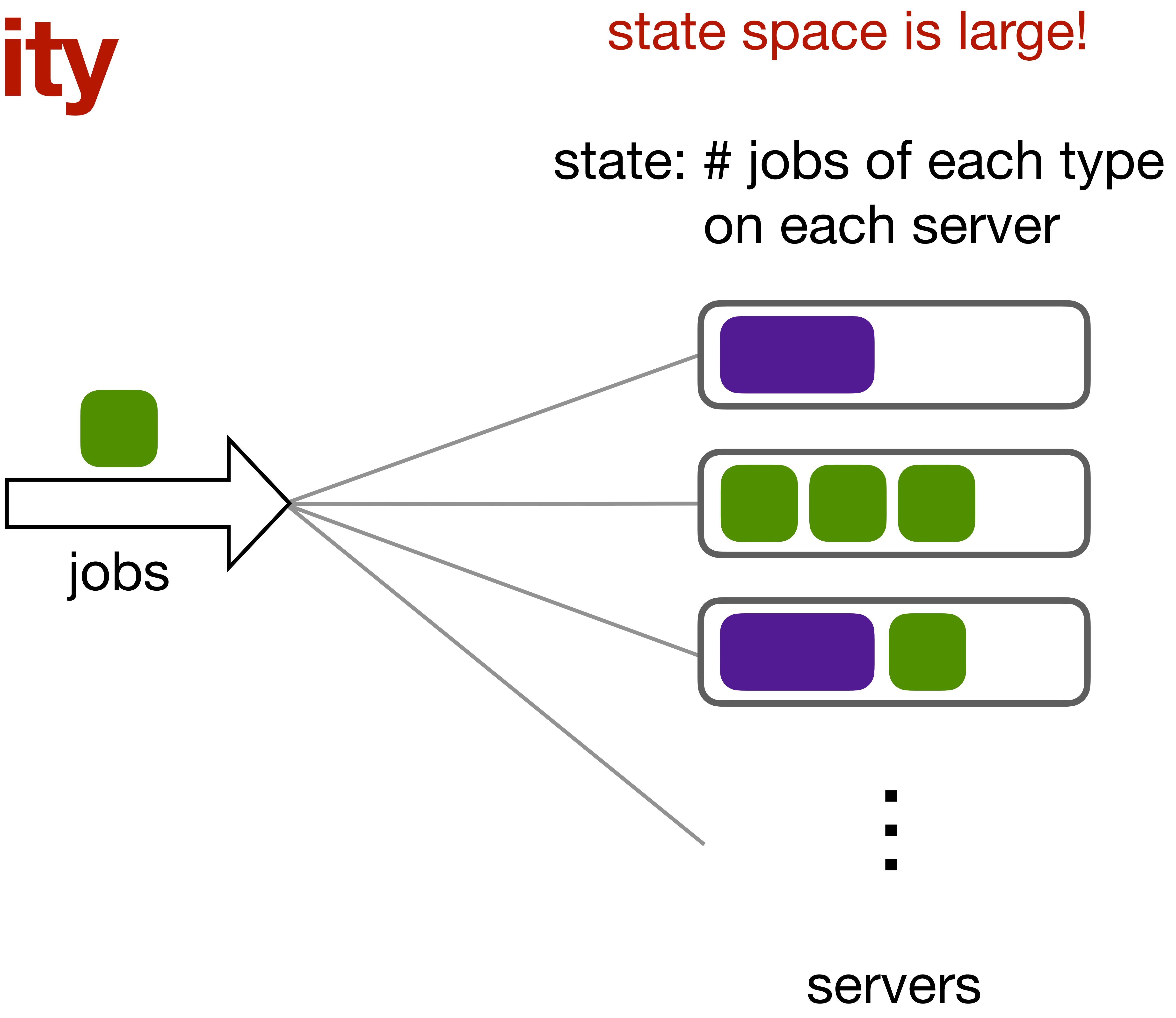
Reducing dimensionality

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



Reducing dimensionality

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



A policy-conversion framework

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

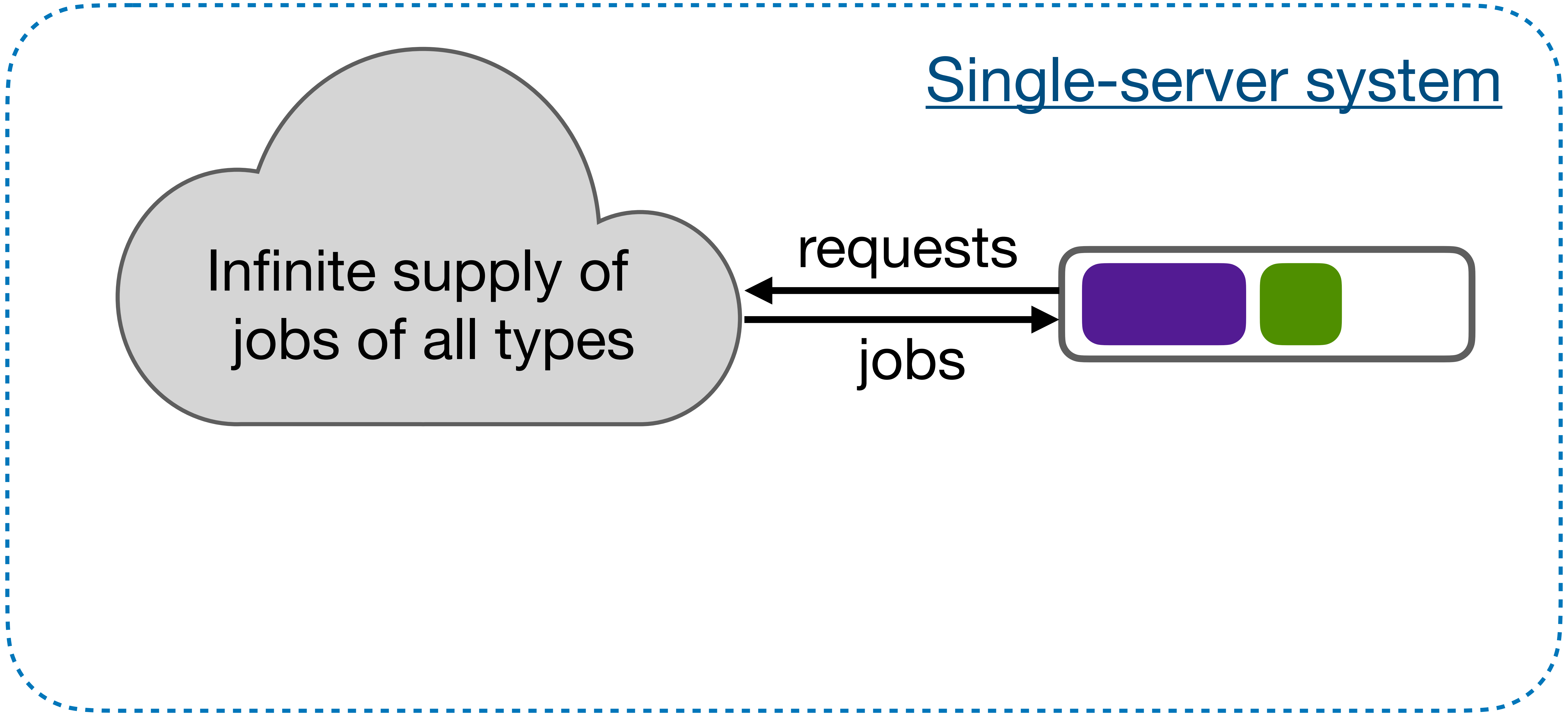
A policy-conversion framework

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

Single-server system

A policy-conversion framework

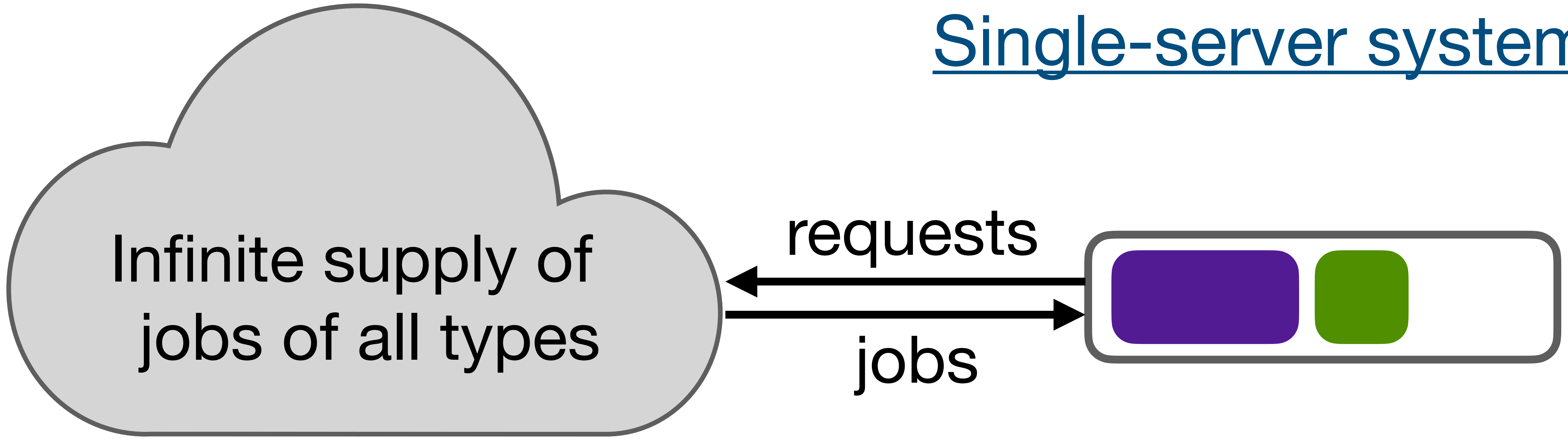
Policies in the ∞-server system ⟷ Policies in a single-server system



A policy-conversion framework

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

Single-server system

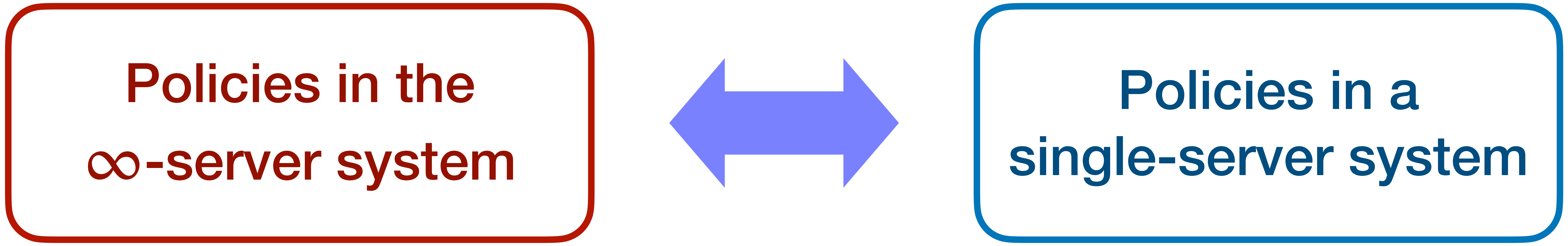


A policy $\overline{\sigma}$ decides when to request what types of jobs to:

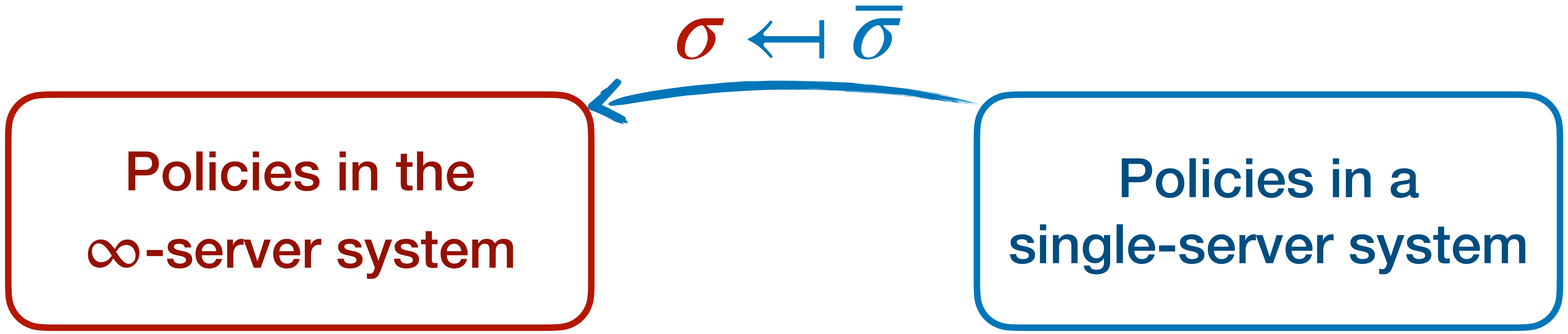
maximize throughput

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

A policy-conversion framework

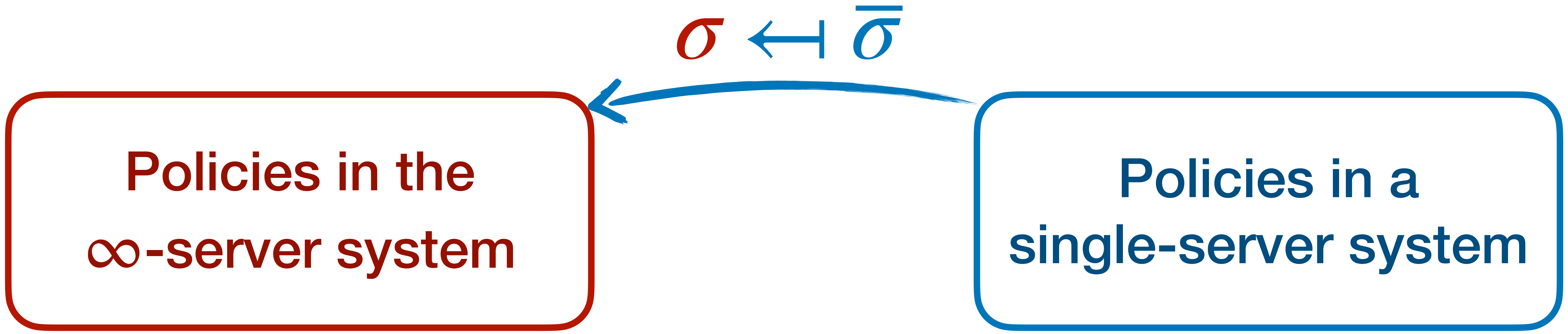


A policy-conversion framework



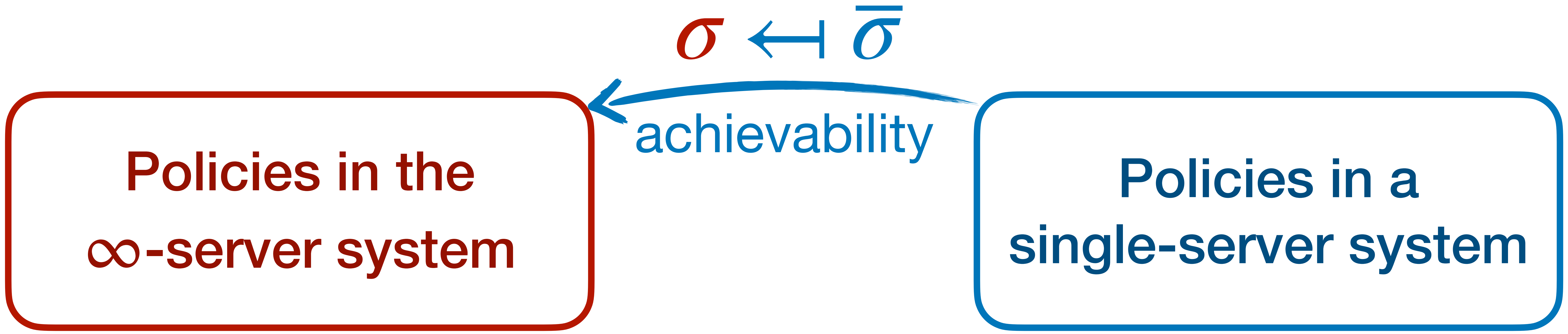
A policy-conversion framework

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



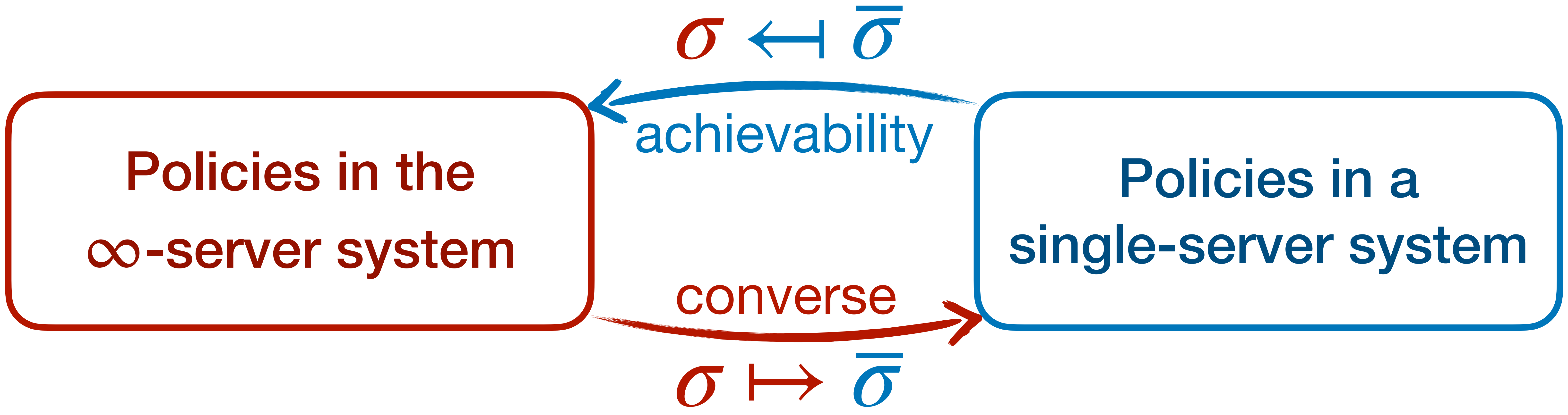
A policy-conversion framework

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



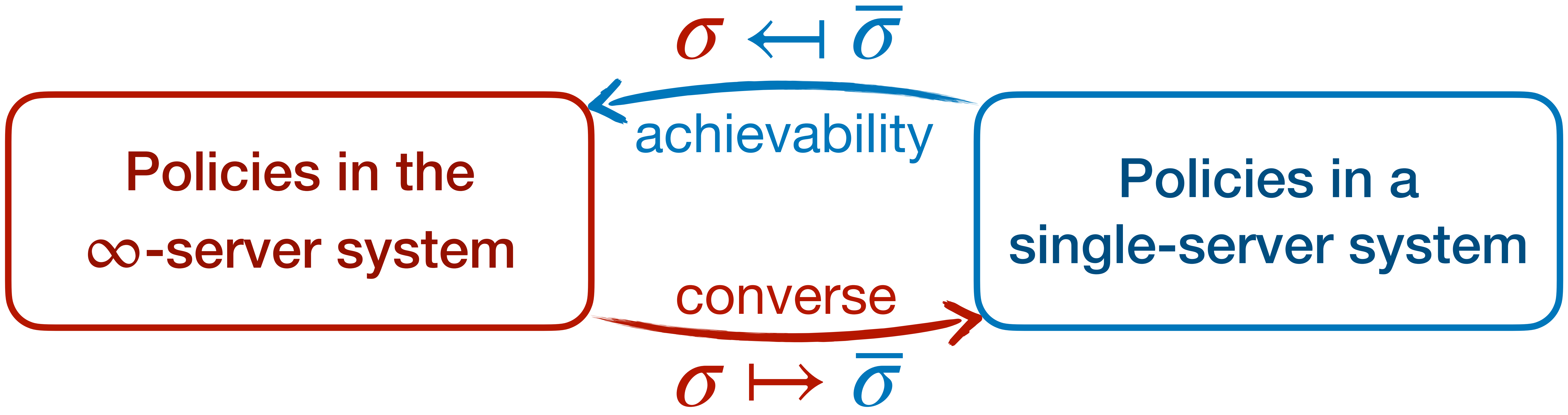
A policy-conversion framework

- 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]

Policies in the ∞-server system

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

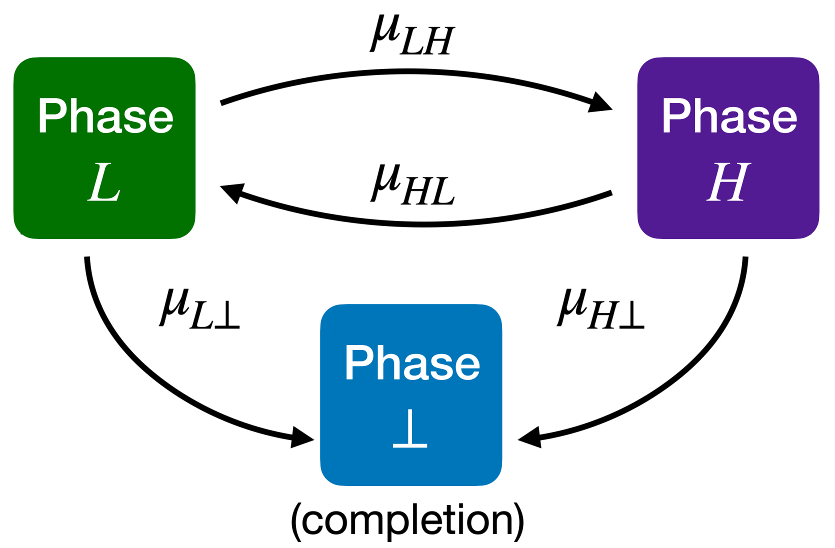
Policies in a single-server system

Policies in the ∞-server system

$\sigma \hookleftarrow \overline{\sigma}$

Policies in a single-server system

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

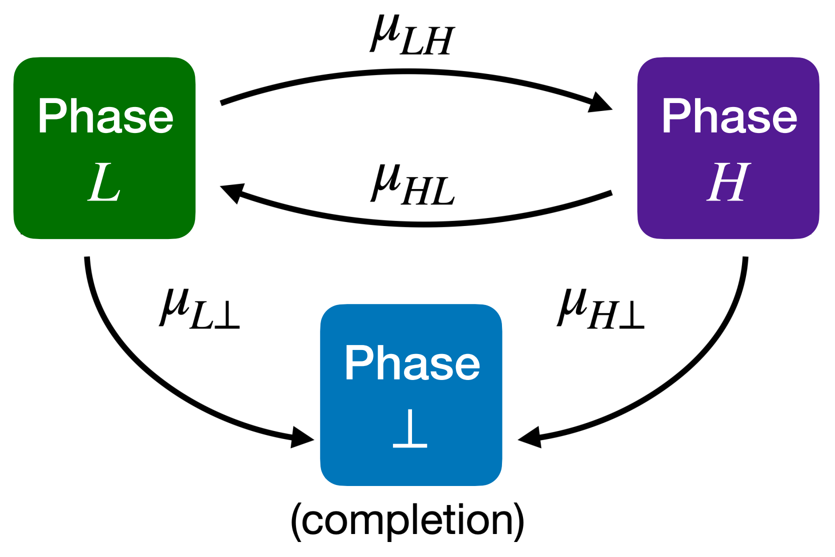


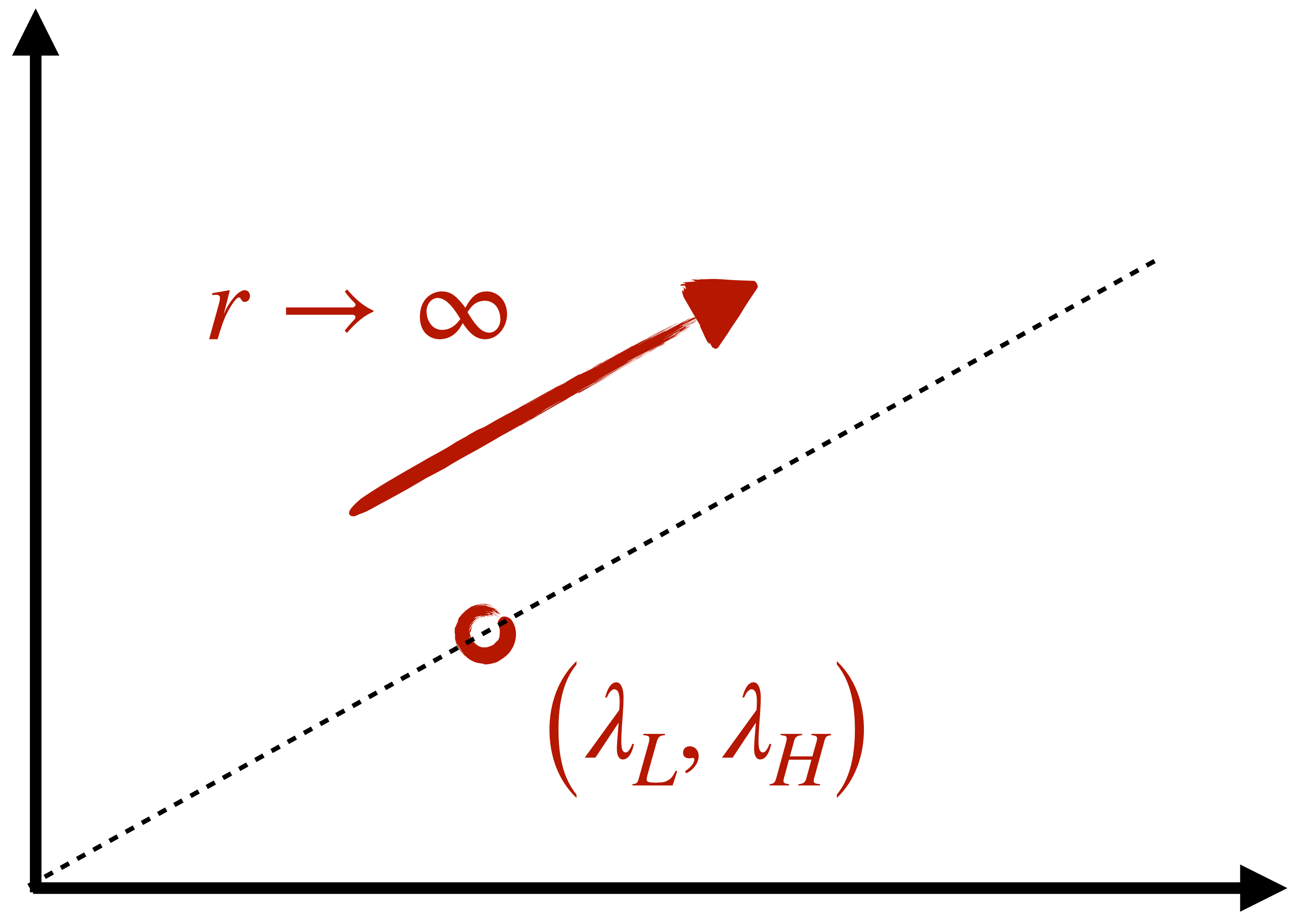
Policies in the ∞-server system

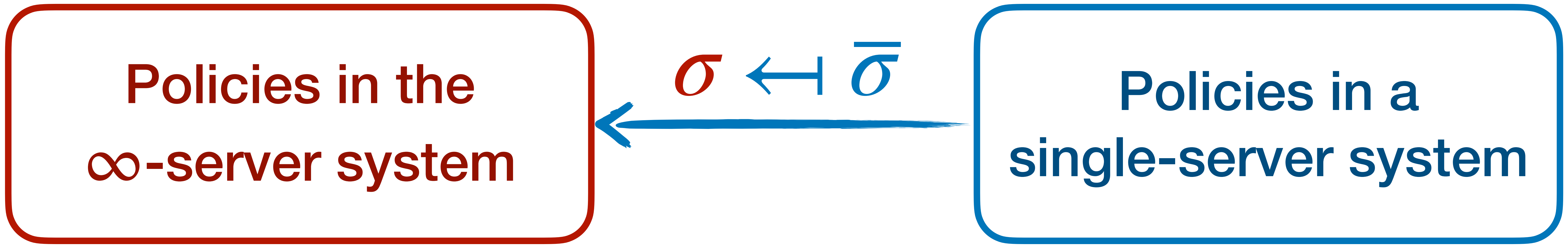
$\sigma \leftarrow\!\dashv \bar{\sigma}$

Policies in a single-server system

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







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

Policies in the ∞-server system

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

Policies in a single-server system

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

Policy $\bar{\sigma}$

$\bar{N} \cdot \left(\textbf{throughput}_L, \textbf{throughput}_R\right) = r \cdot (\lambda_L, \lambda_H)$

cost (resource contention) ≤ budget

Policies in the ∞-server system

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

Policies in a single-server system

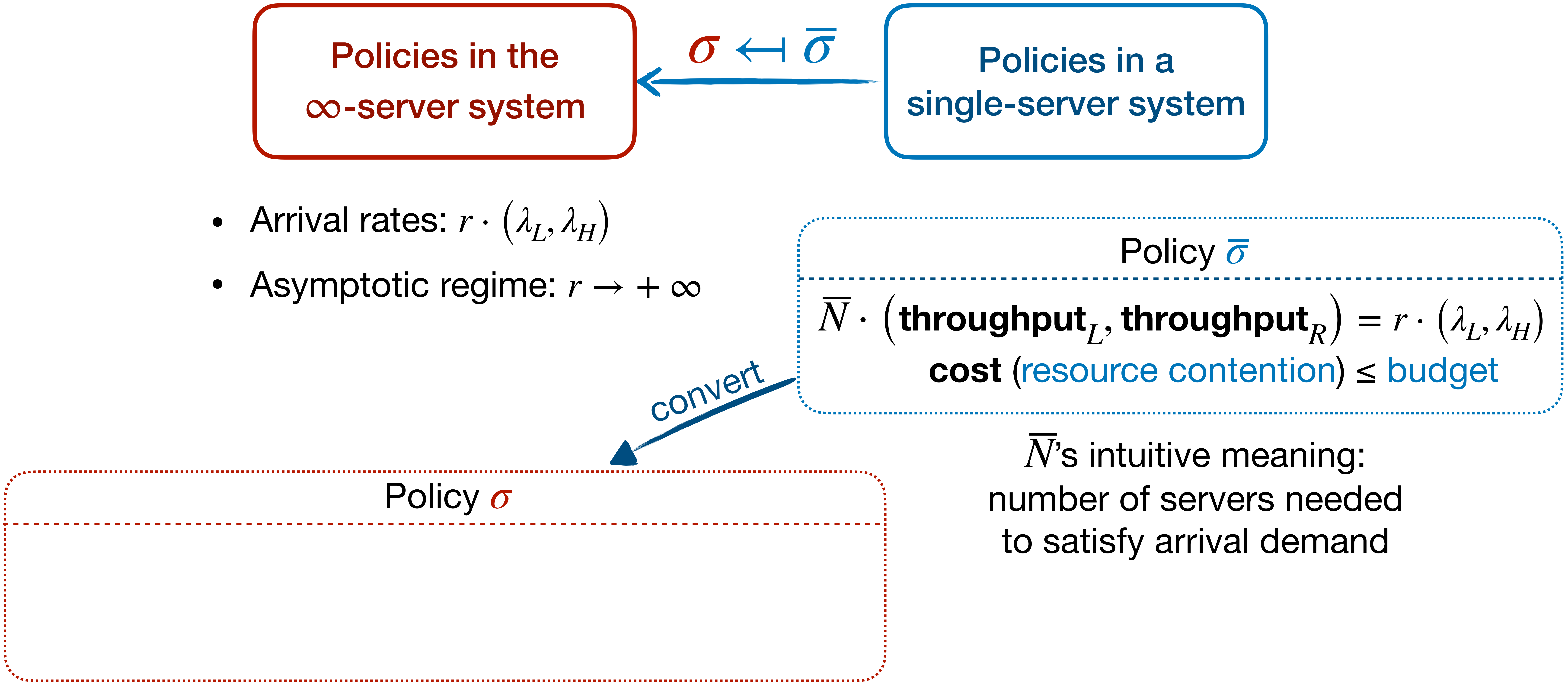
- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$
- Asymptotic regime: $r \to +\infty$

Policy $\bar{\sigma}$

$\bar{N} \cdot \left(\textbf{throughput}_L, \textbf{throughput}_R\right) = r \cdot (\lambda_L, \lambda_H)$

cost (resource contention) ≤ budget

$\bar{N}$'s intuitive meaning:
number of servers needed
to satisfy arrival demand



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

$\bar{N}$'s intuitive meaning:
number of servers needed
to satisfy arrival demand

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 $\bar{\sigma}$

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

cost (resource contention) ≤ budget

convert

Policy σ

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

$\bar{N}$'s intuitive meaning: number of servers needed to satisfy arrival demand

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 $\bar{\sigma}$

$\bar{N} \cdot \left(\textbf{throughput}_L, \textbf{throughput}_R\right) = r \cdot (\lambda_L, \lambda_H)$

cost (resource contention) ≤ budget

convert

Policy σ

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

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

$\bar{N}$'s intuitive meaning:
number of servers needed
to satisfy arrival demand

Policies in the ∞-server system $\xleftarrow{\sigma \leftarrow \overline{\sigma}}$ Policies in a single-server system

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

Policy $\overline{\sigma}^*$

$\overline{N}^* \cdot \left(\textbf{throughput}_L, \textbf{throughput}_R\right) = r \cdot (\lambda_L, \lambda_H)$

cost (resource contention) ≤ budget

convert

Policy σ^*

$\mathbf{E}$ [# 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

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

Policies in a single-server system

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

Policy $\overline{\sigma}^*$

$\overline{N}^* \cdot \left(\textbf{throughput}_L, \textbf{throughput}_R\right) = r \cdot (\lambda_L, \lambda_H)$

cost (resource contention) ≤ budget

$\overline{\sigma}^*$ maximizes the throughput and thus achieves the minimum $\overline{N}^*$

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

$\sigma \leftarrow \overline{\sigma}$

Policies in a
single-server system

Main Results:

Policy $\overline{\sigma}^*$

$\overline{N}^* \cdot \left(\textbf{throughput}_L, \textbf{throughput}_R\right) = r \cdot (\lambda_L, \lambda_H)$

cost (resource contention) ≤ budget

$\overline{\sigma}^*$ maximizes the throughput and thus achieves the minimum $\overline{N}^*$

Policy σ^*

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

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

Policies in the ∞-server system

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

Policies in a single-server system

Main Results:

- E[# active servers] $\geq \overline{N}^*$ under any policy

Policy $\overline{\sigma}^*$

$$\overline{N}^* \cdot \left(\textbf{throughput}_L, \textbf{throughput}_R\right) = r \cdot (\lambda_L, \lambda_H)$$

cost (resource contention) ≤ budget

$\overline{\sigma}^*$ maximizes the throughput and thus achieves the minimum $\overline{N}^*$

Policy σ^*

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

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

Policies in the ∞-server system

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

Policies in a single-server system

Main Results:

- E[# active servers] $\geq \overline{N}^*$ under any policy
- The policy σ^* converted from $\overline{\sigma}^*$ is *asymptotically optimal*

Policy $\overline{\sigma}^*$

$$\overline{N}^* \cdot \left(\textbf{throughput}_L, \textbf{throughput}_R\right) = r \cdot (\lambda_L, \lambda_H)$$

cost (resource contention) ≤ budget

$\overline{\sigma}^*$ maximizes the throughput and thus achieves the minimum $\overline{N}^*$

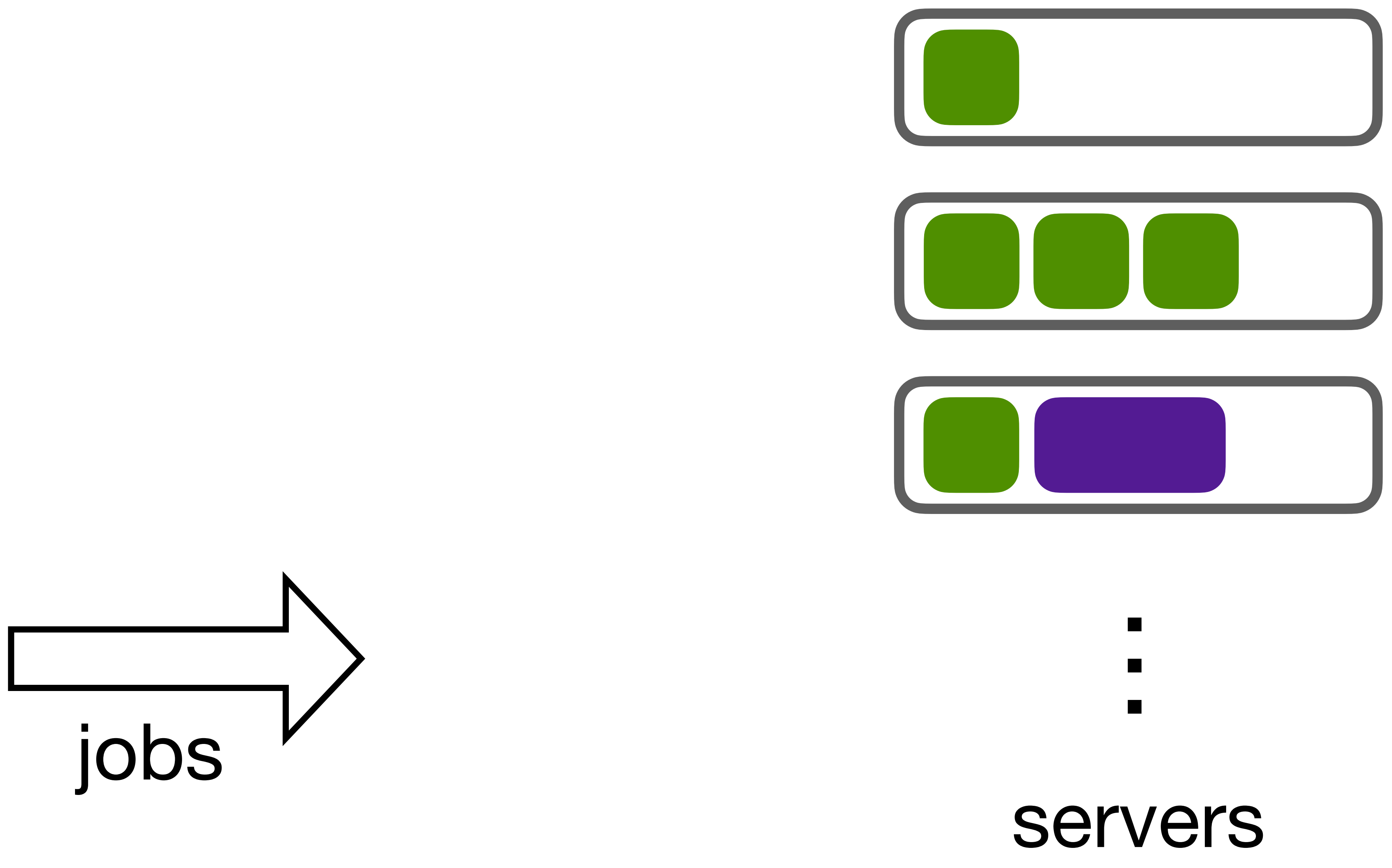
Policy σ^*

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

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

Policy conversion: single-server to ∞-server

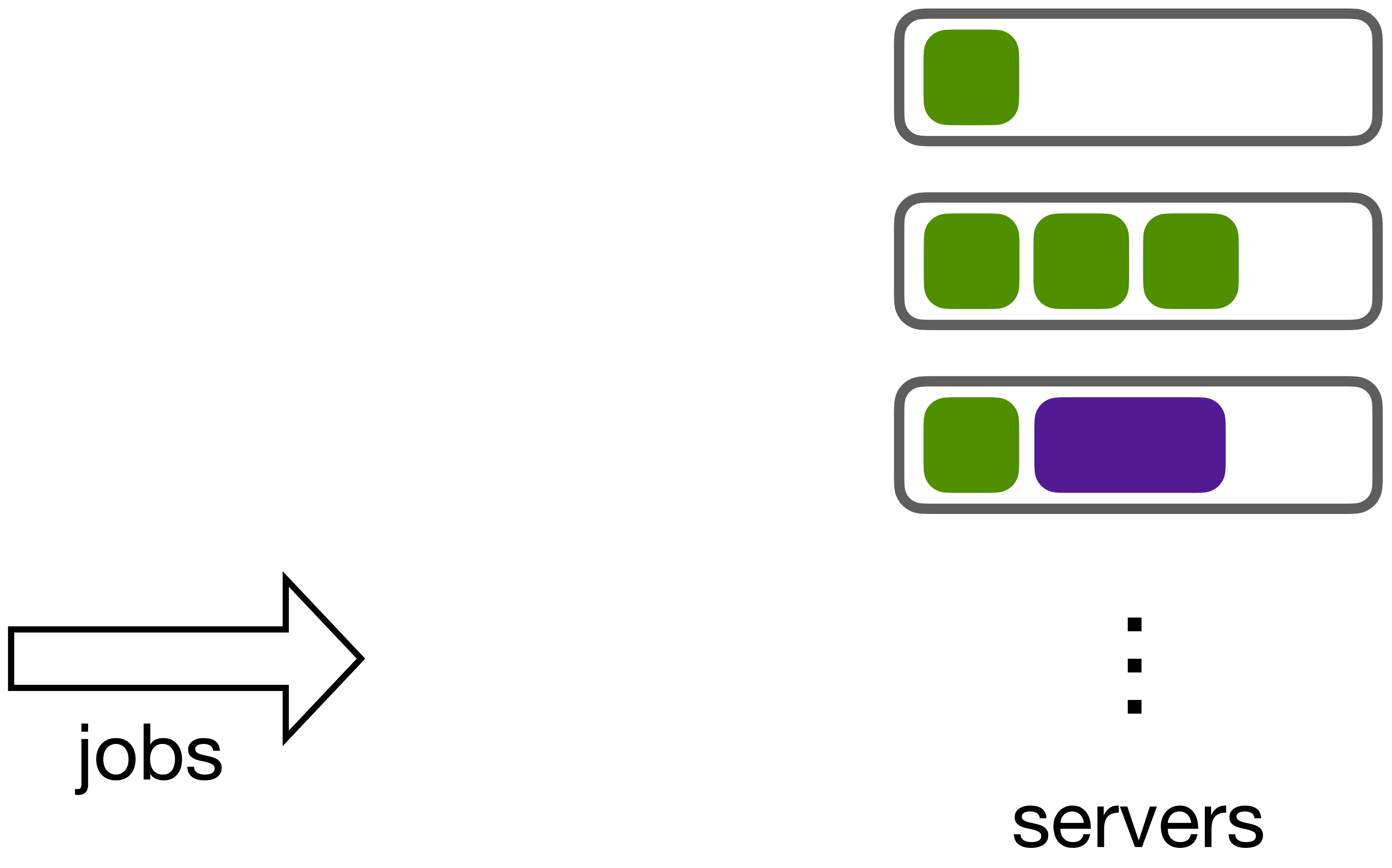
Meta-algorithm: JOIN-REQUESTING-SERVER ($\vec{\sigma}$)



Policy conversion: single-server to ∞-server

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

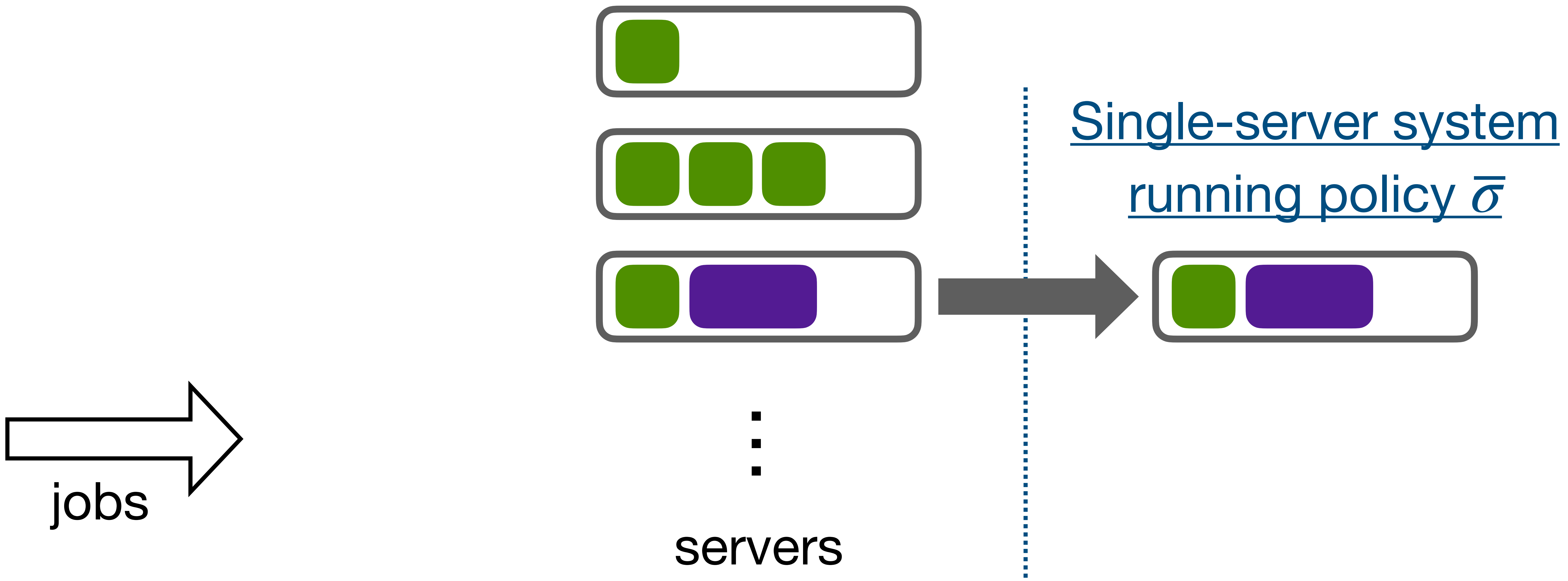
- For each server, run a single-server policy $\overline{\sigma}$



Policy conversion: single-server to ∞-server

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

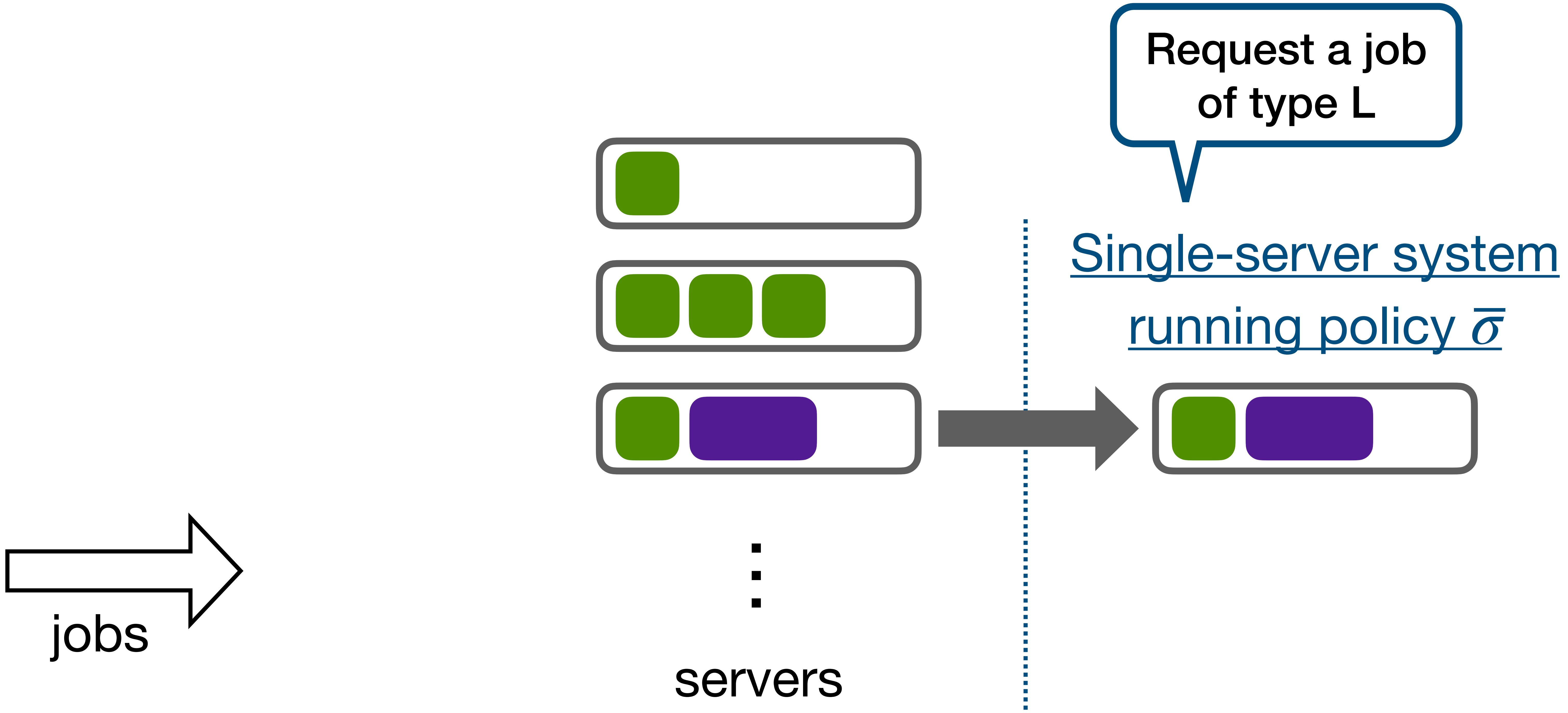
- For each server, run a single-server policy $\overline{\sigma}$



Policy conversion: single-server to ∞-server

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

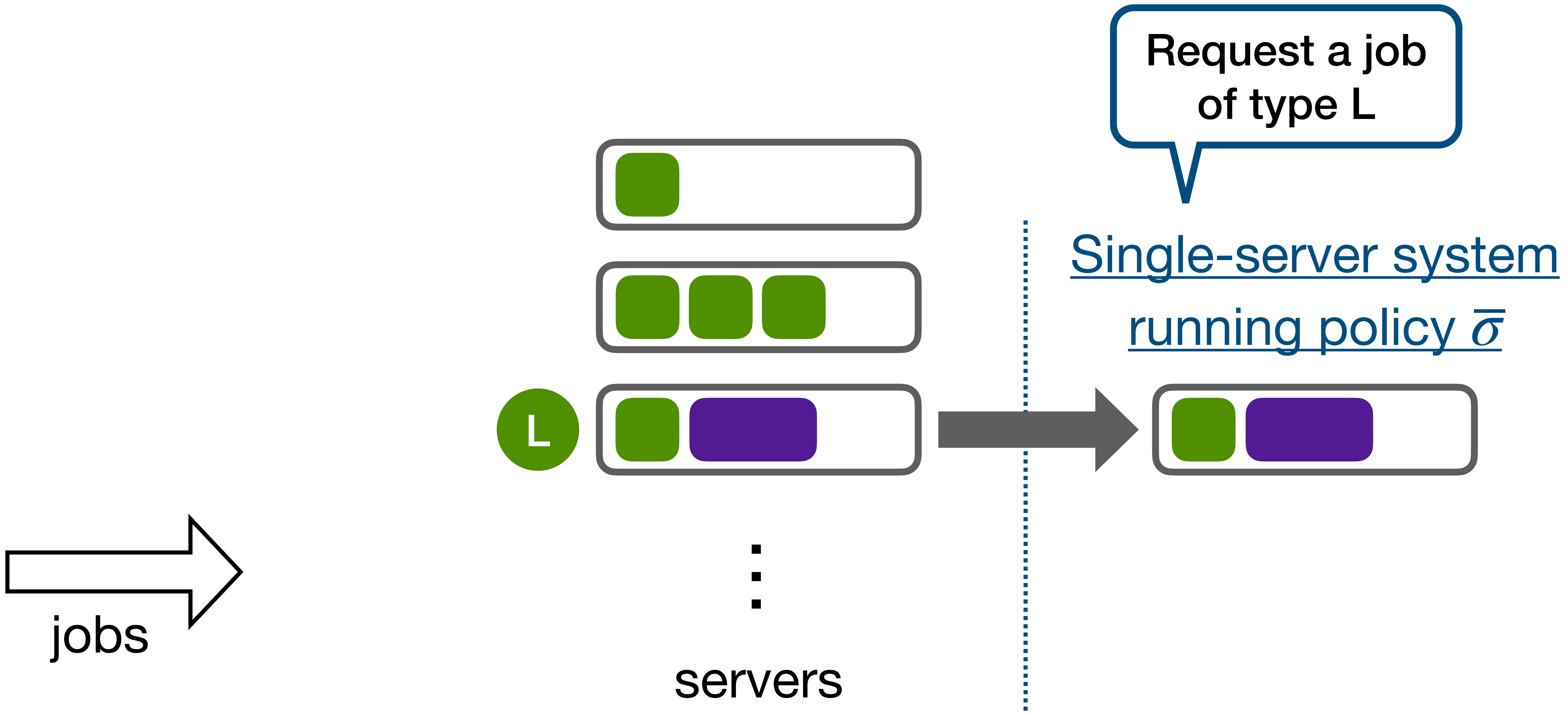
- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type i, generate a token of type i



Policy conversion: single-server to ∞-server

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

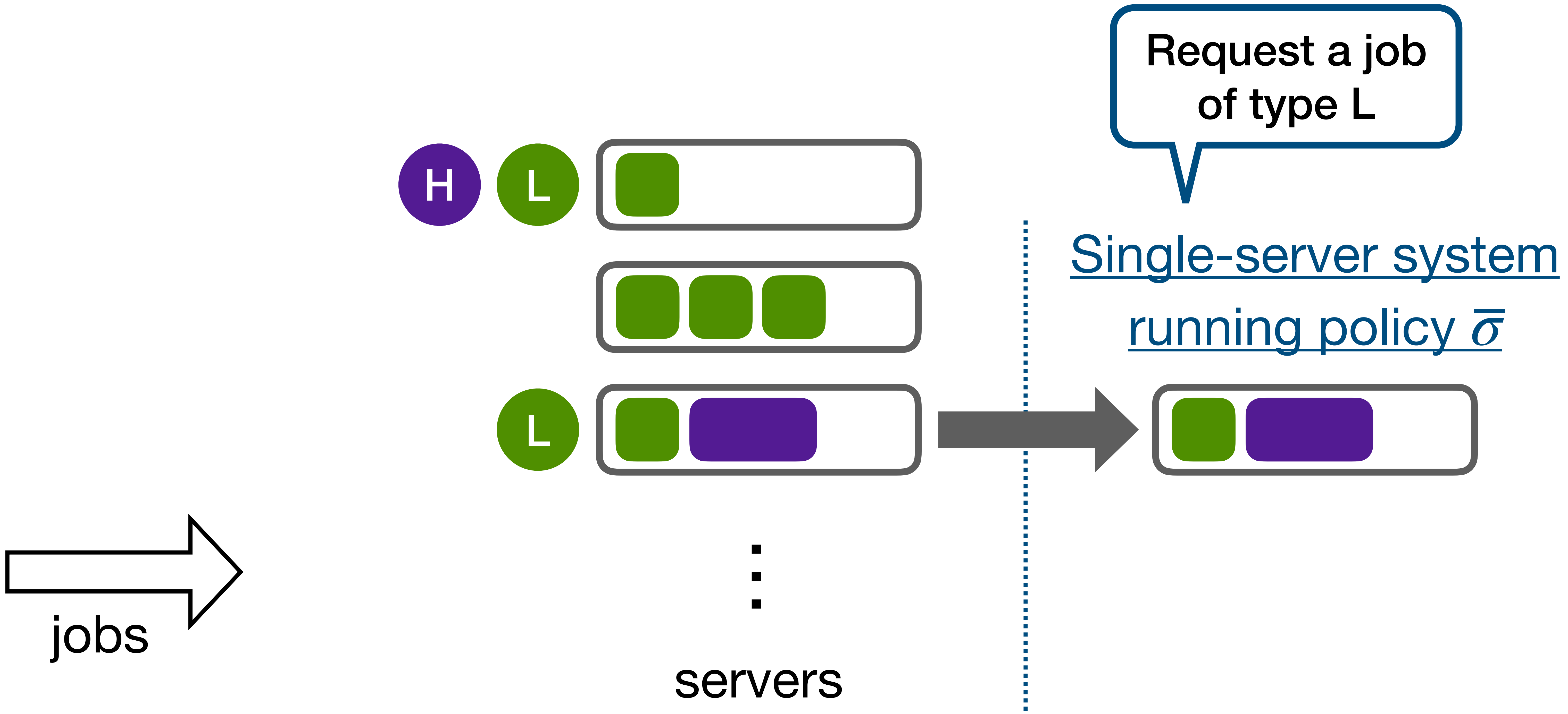
- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type i, generate a token of type i



Policy conversion: single-server to ∞-server

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

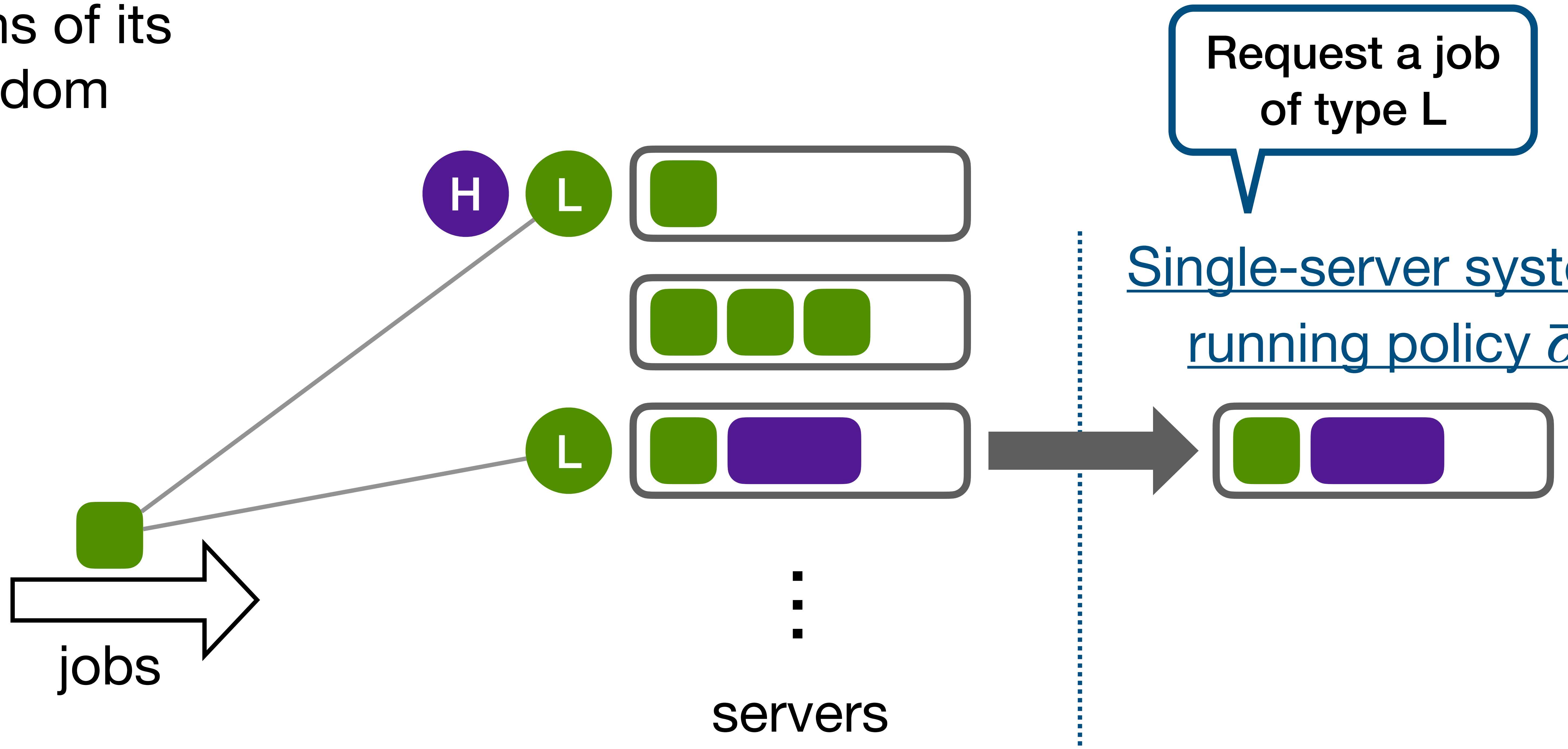
- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type i, generate a token of type i



Policy conversion: single-server to ∞-server

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

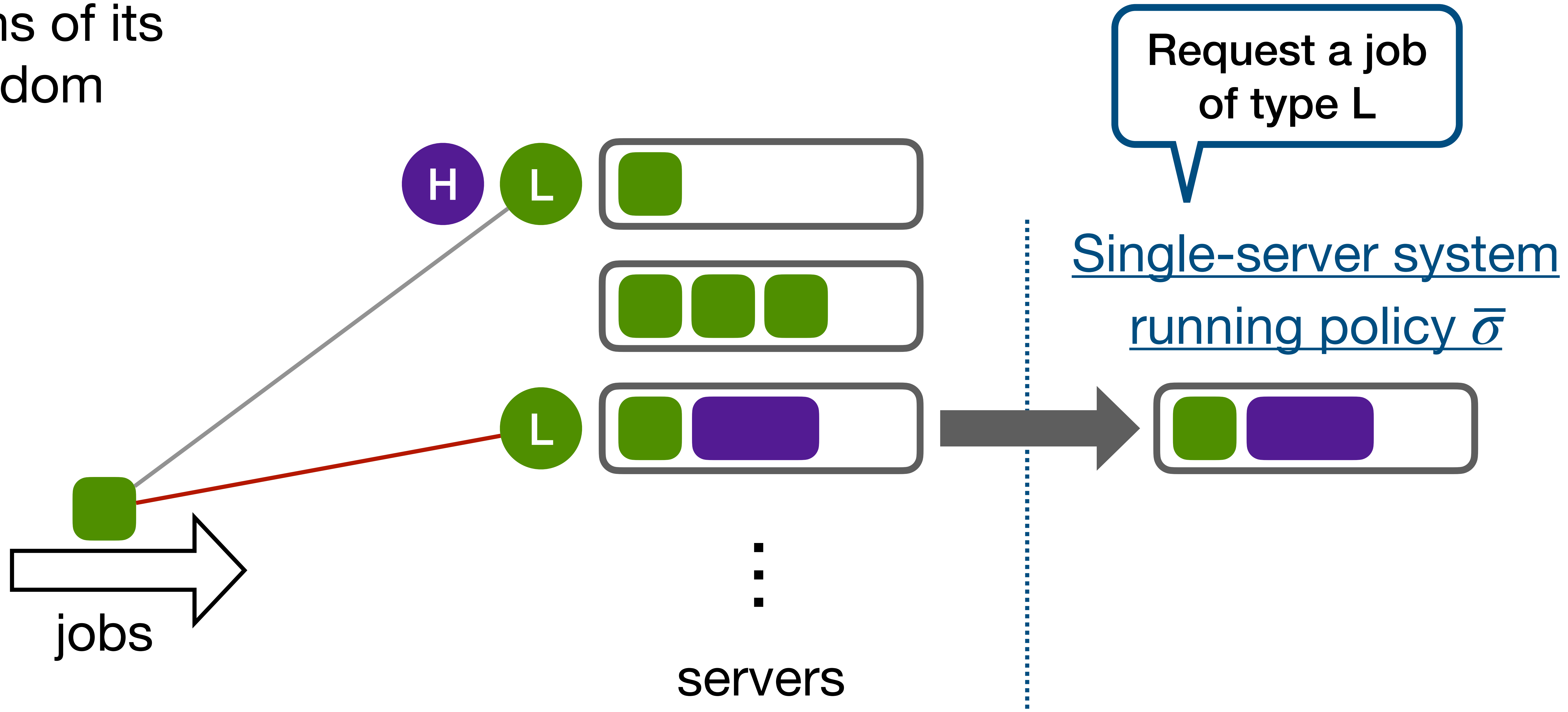
- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type i, generate a token of type i
- When a job arrives, it checks tokens of its type and joins one uniformly at random



Policy conversion: single-server to ∞-server

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

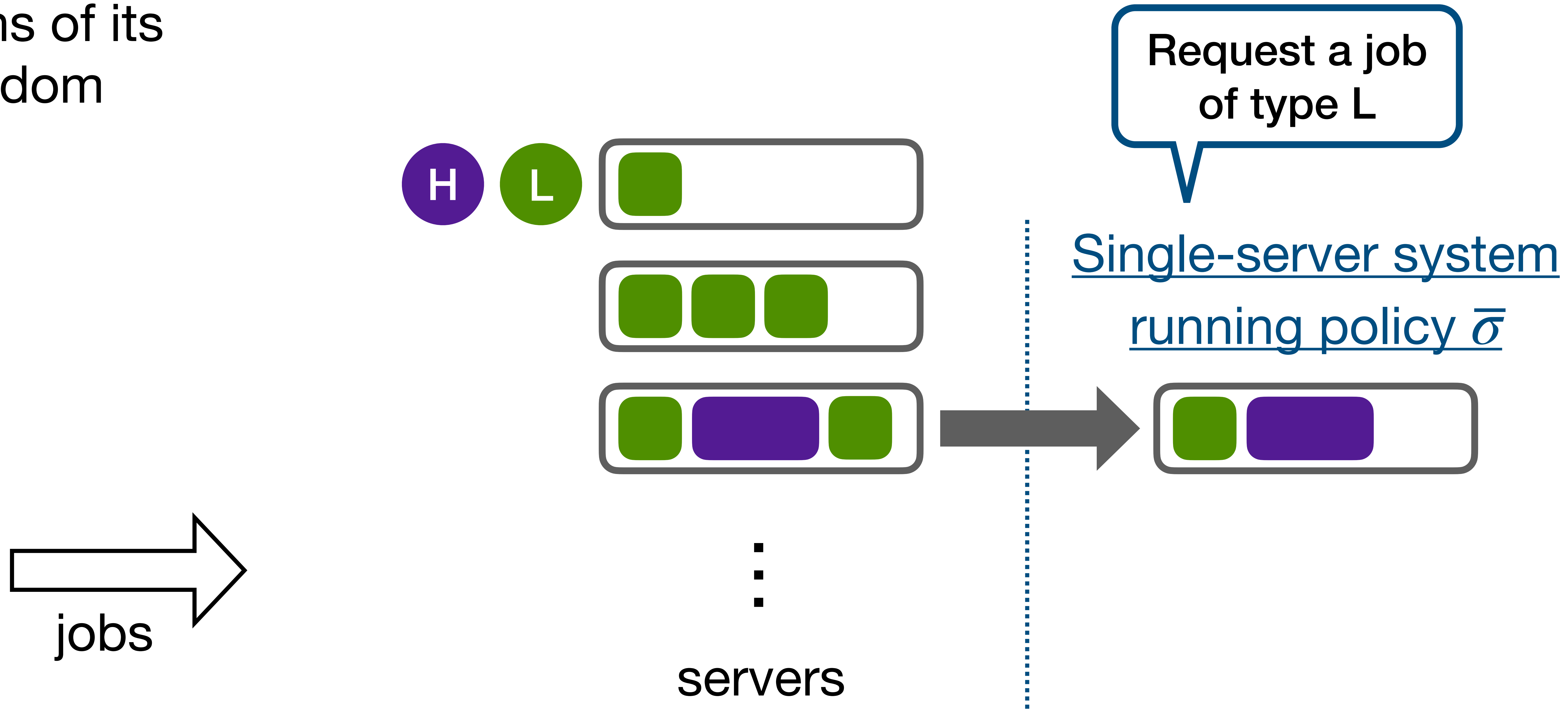
- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type i, generate a token of type i
- When a job arrives, it checks tokens of its type and joins one uniformly at random



Policy conversion: single-server to ∞-server

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

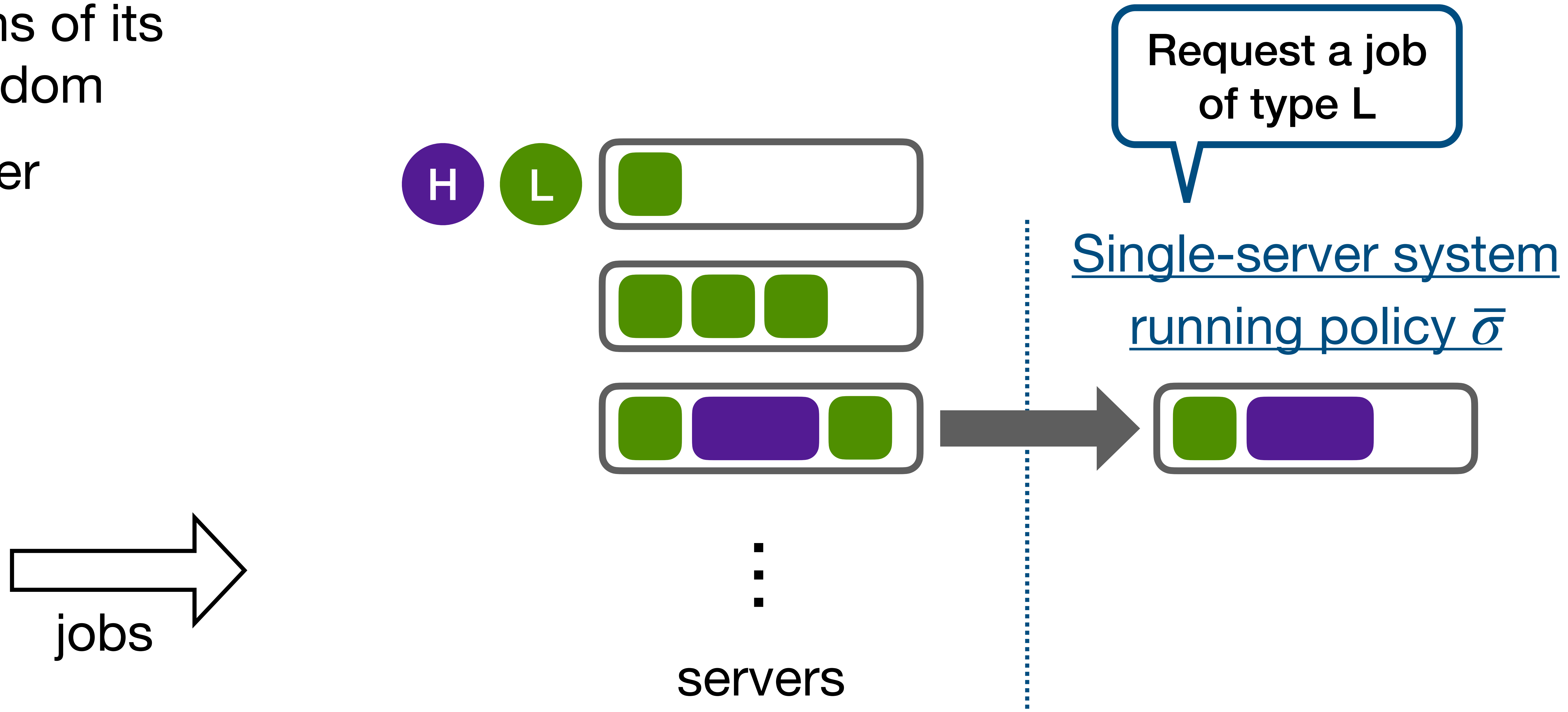
- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type i, generate a token of type i
- When a job arrives, it checks tokens of its type and joins one uniformly at random



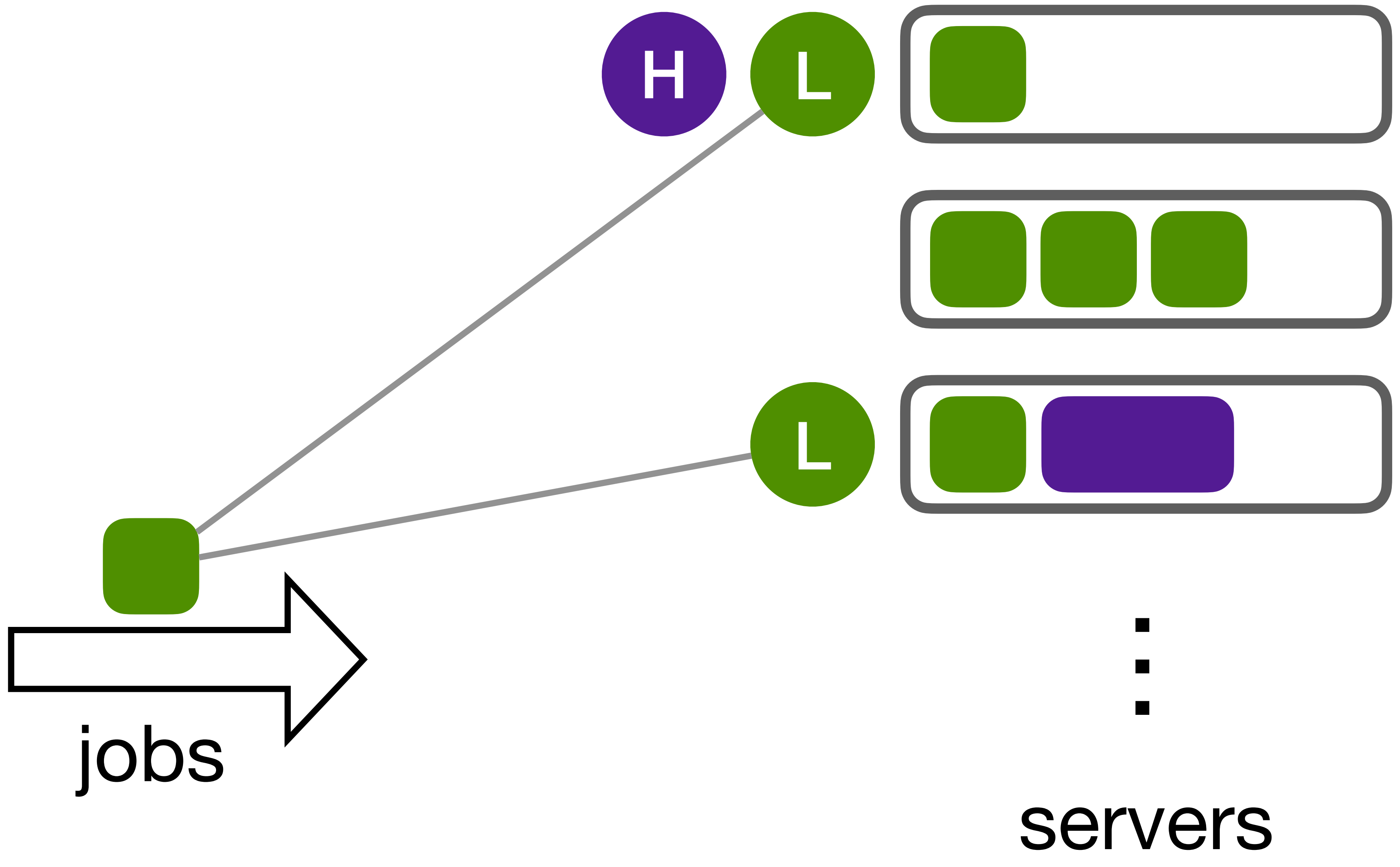
Policy conversion: single-server to ∞-server

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

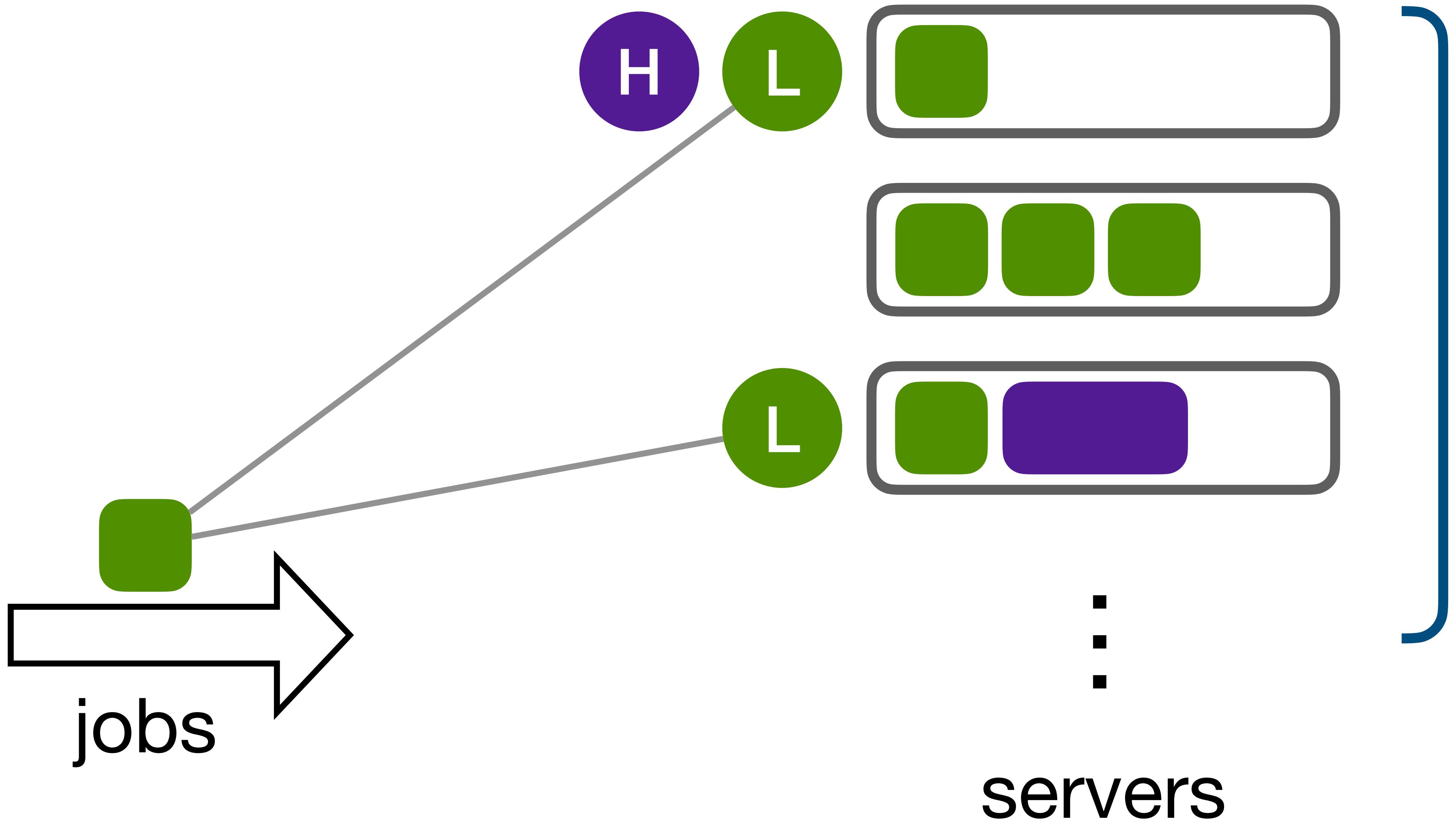
- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type i, generate a token of type i
- When a job arrives, it checks tokens of its type and joins one uniformly at random
- If no tokens, go to an inactive server



Policy conversion: more details



Policy conversion: more details

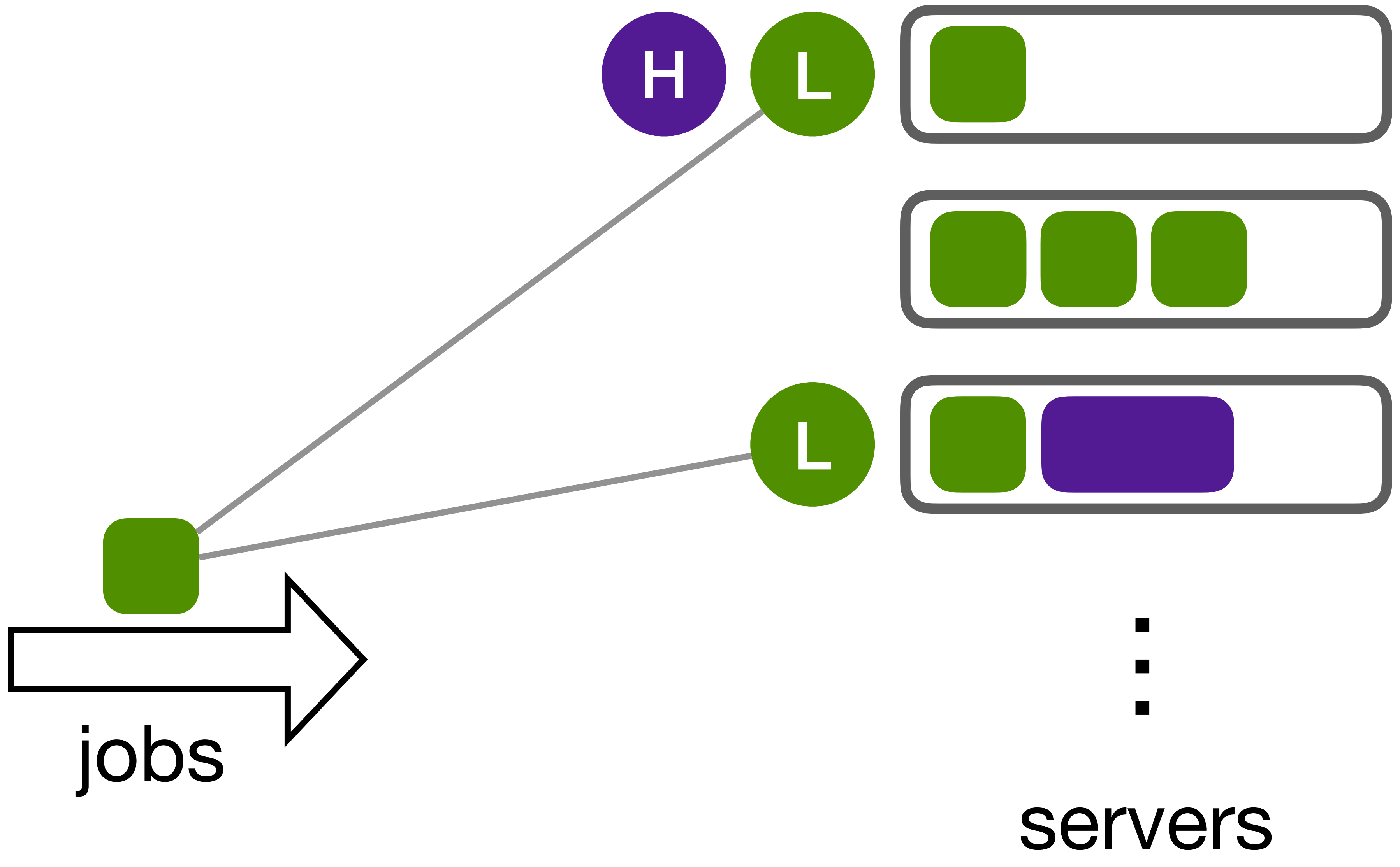


Run single-server policy $\overline{\sigma}$ for only

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

(regular servers)

Policy conversion: more details



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

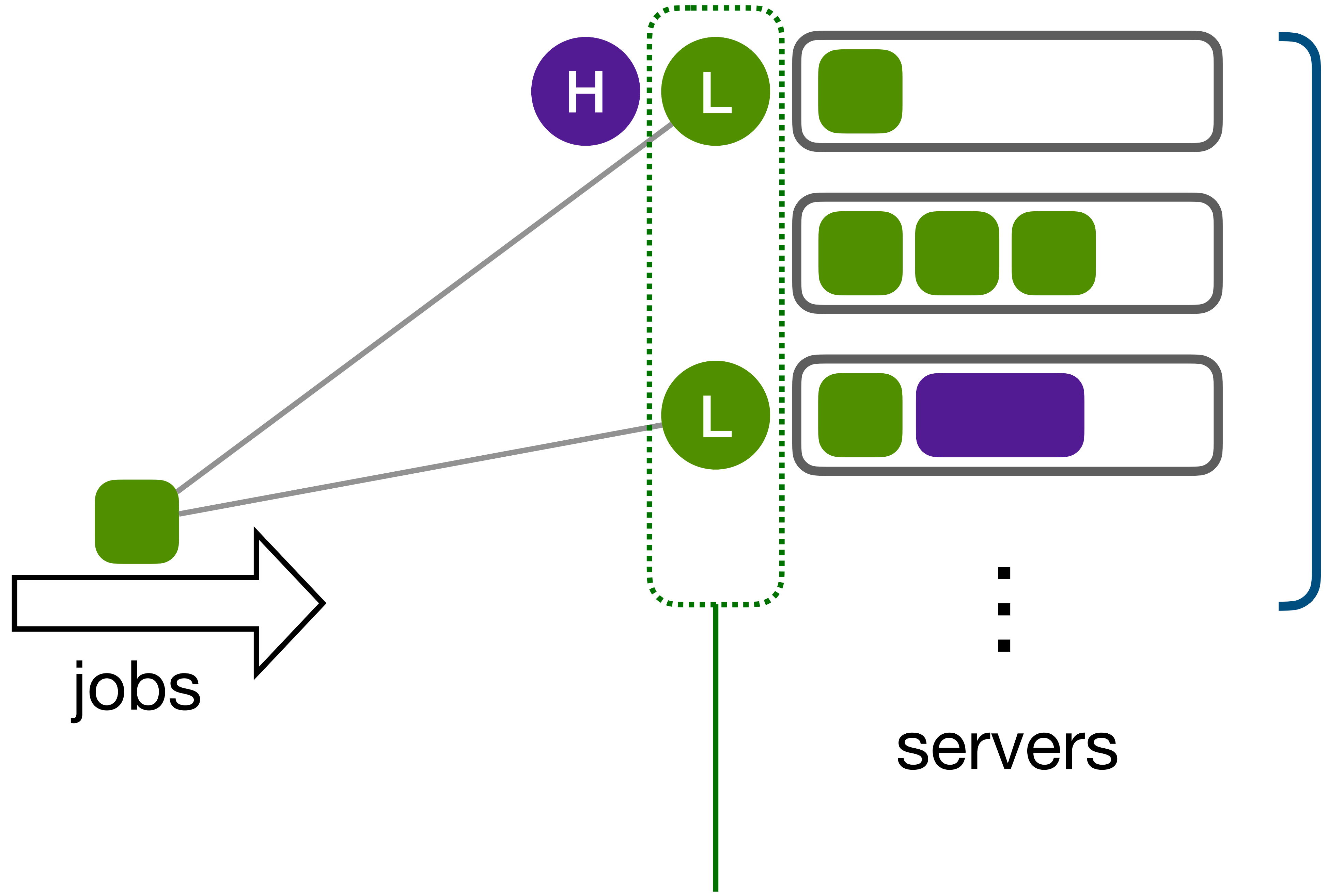
$$\overline{N} = \frac{\text{arrival rate}}{\text{throughtput}(\overline{\sigma})} \text{ servers}$$

(regular 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



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

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

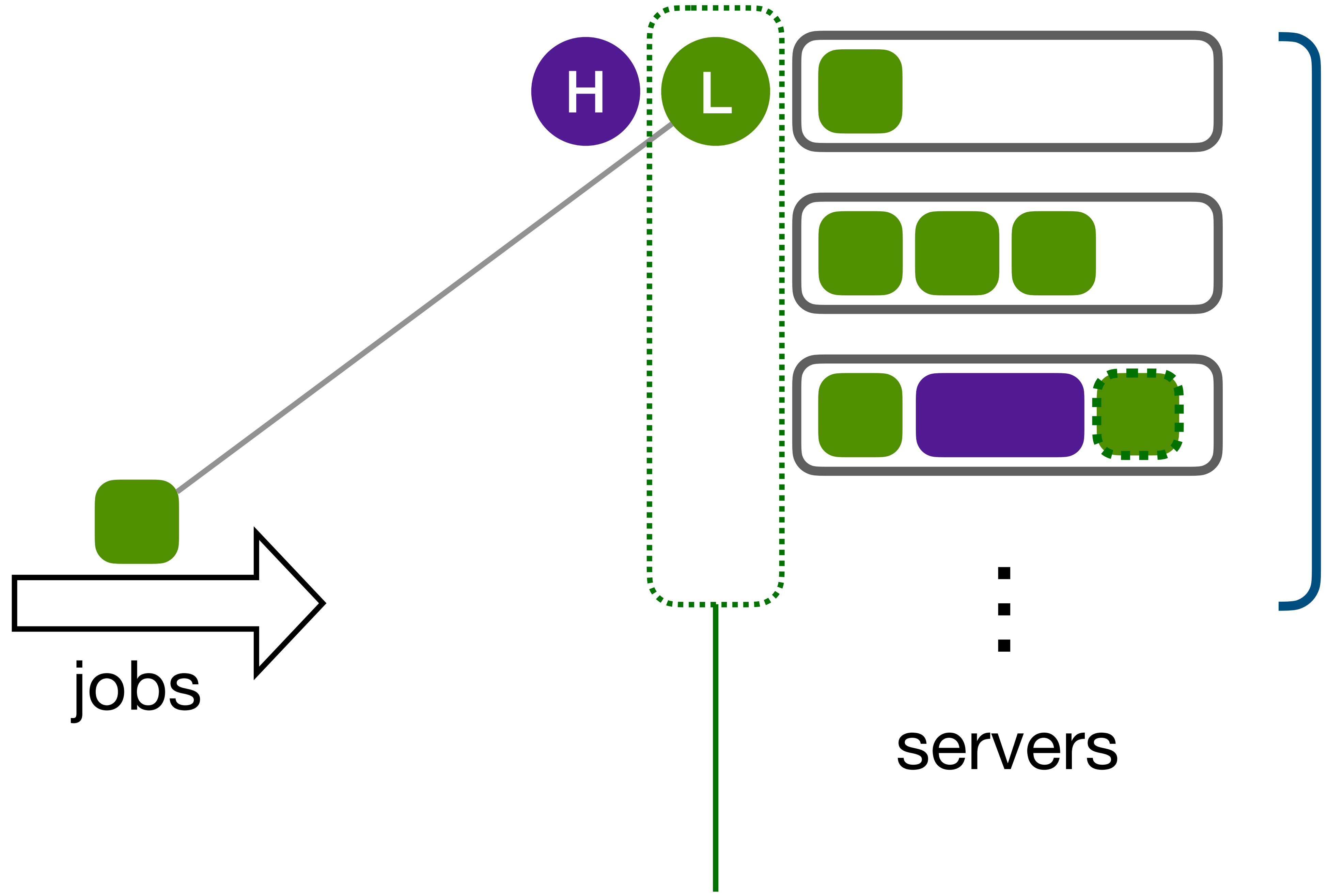
(regular servers)

Recall that we aim to show

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

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

Policy conversion: more details



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

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

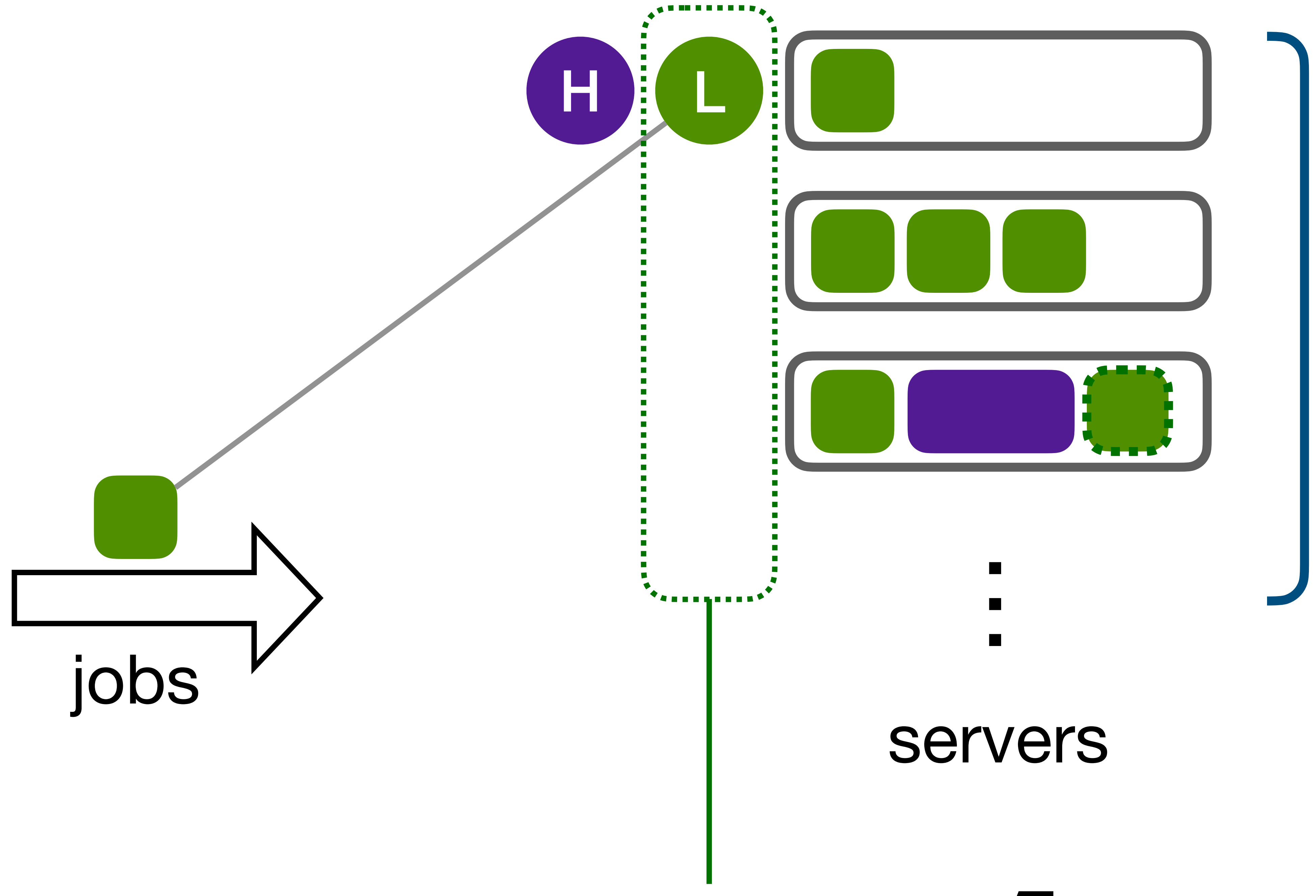
(regular servers)

Recall that we aim to show

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

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

Policy conversion: more details



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

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

(regular servers)

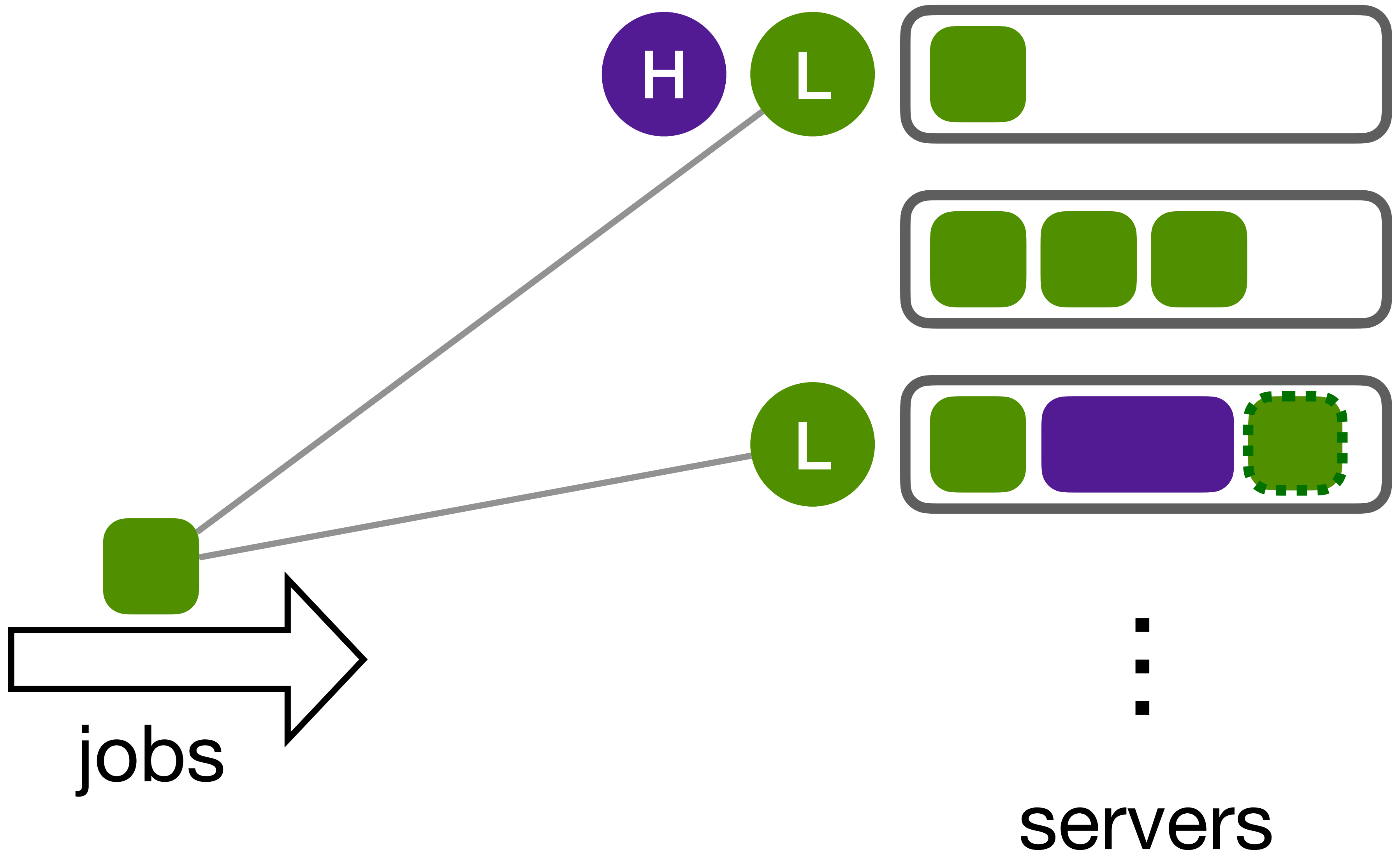
Recall that we aim to show

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

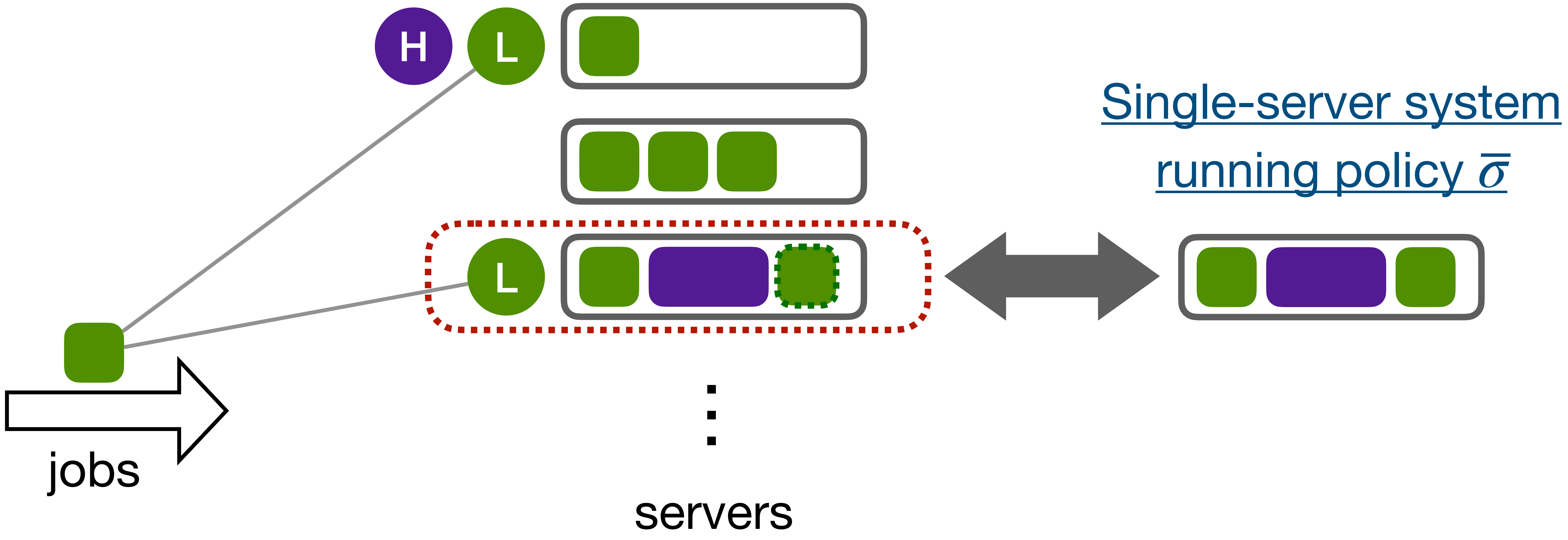
When the # tokens of a type $> \sqrt{r}$, remove the overflow tokens and generate virtual jobs

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

Key proof idea 1

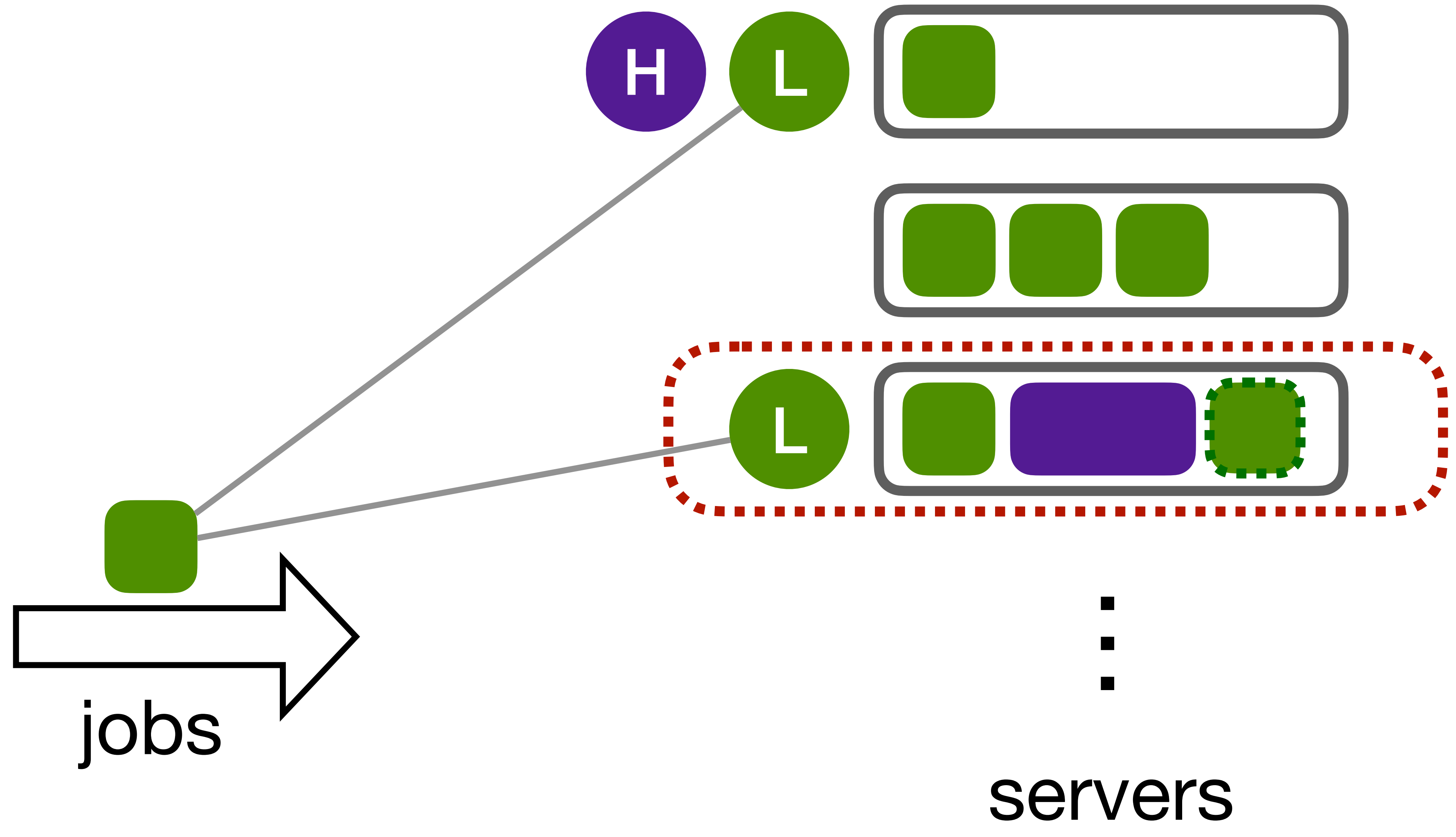


Key proof idea 1

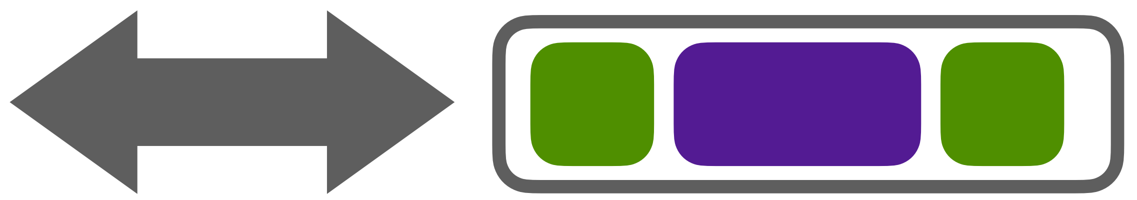


Key proof idea 1

Will show that regular servers in the original system $\approx \overline{N}$ independent single-server systems

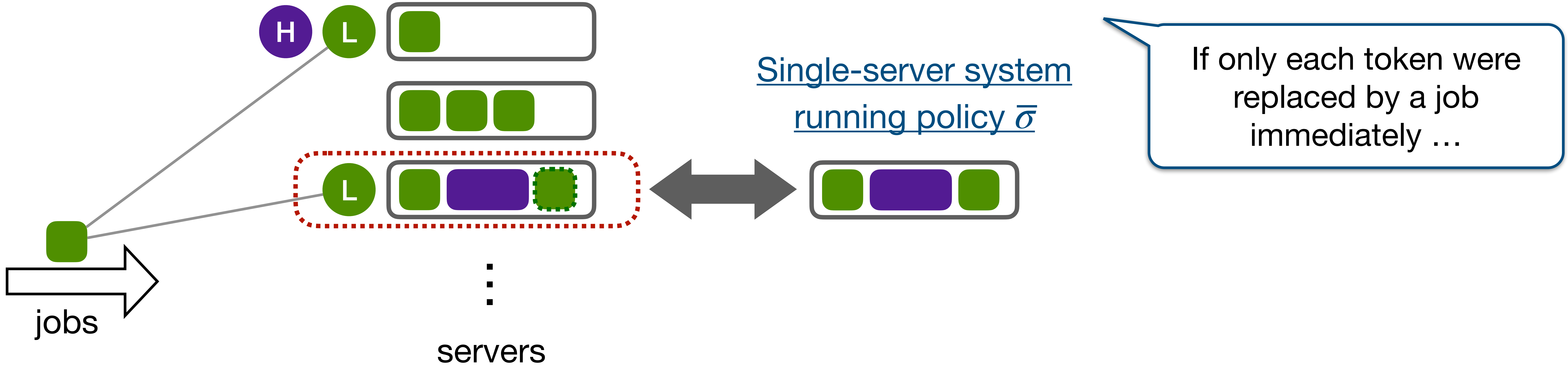


Single-server system running policy $\overline{\sigma}$



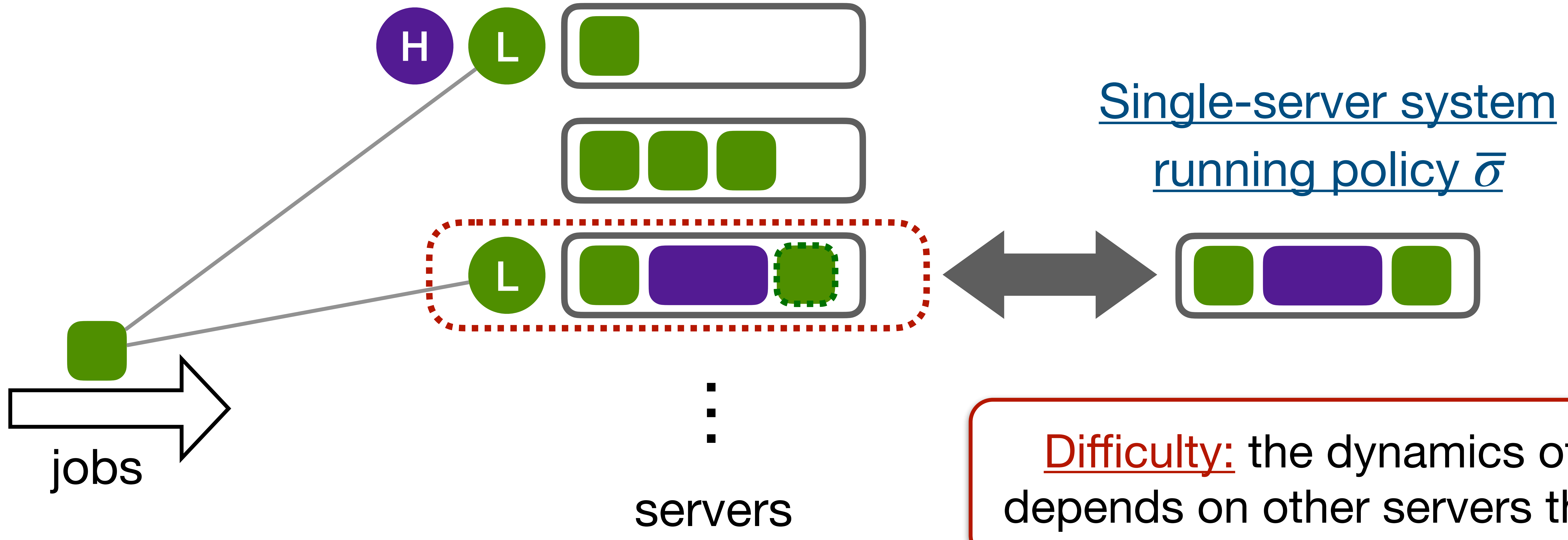
Key proof idea 1

Will show that regular servers in the original system $\approx \bar{N}$ independent single-server systems



Key proof idea 1

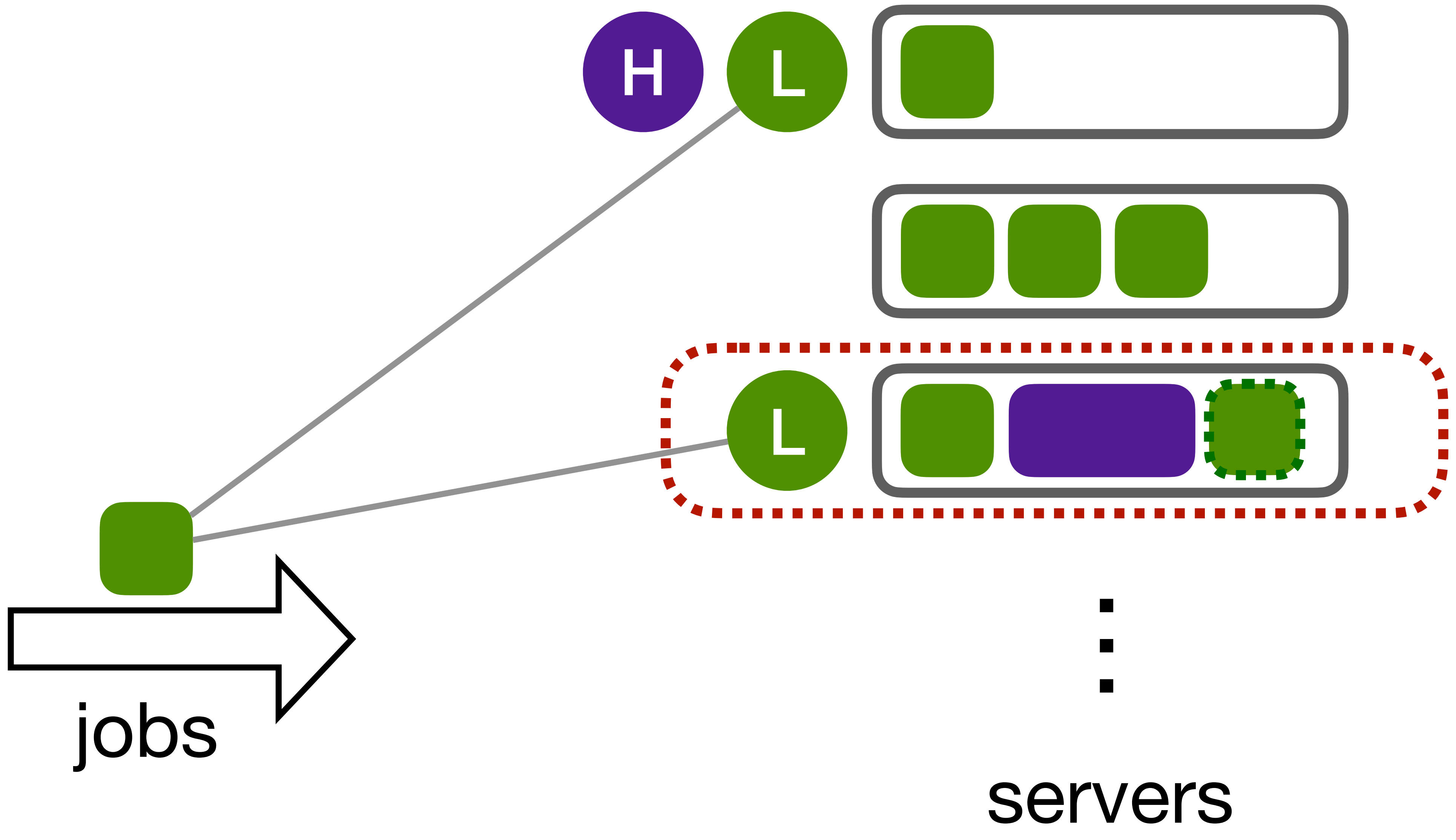
Will show that regular servers in the original system $\approx \overline{N}$ independent single-server systems

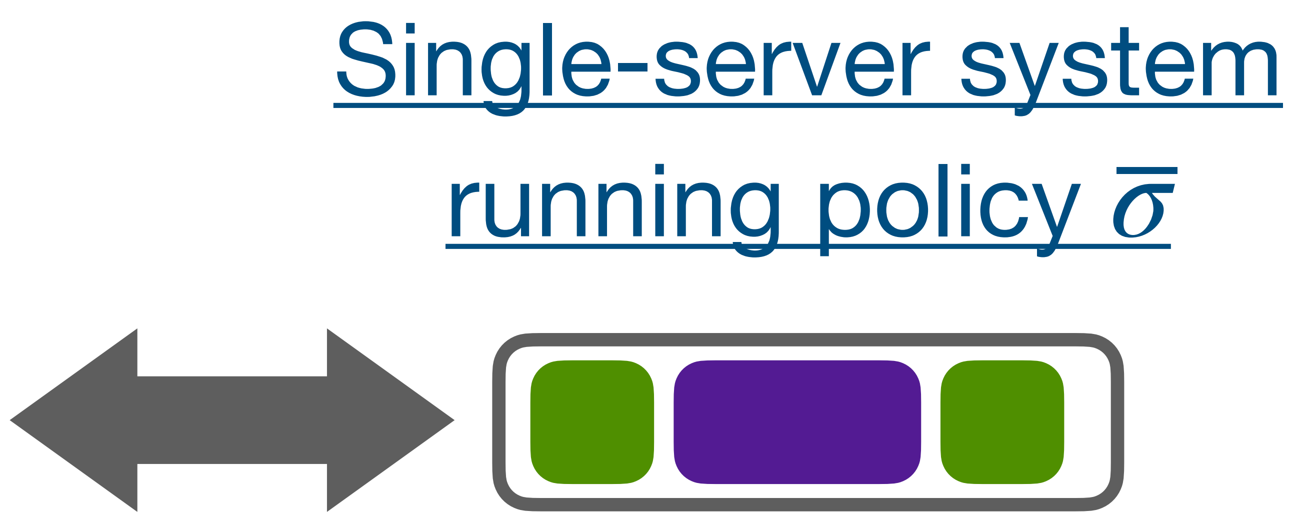


Difficulty: the dynamics of a server in the original system depends on other servers through arrivals & token overflows

Key proof idea 1

Will show that regular servers in the original system $\approx \overline{N}$ independent single-server systems





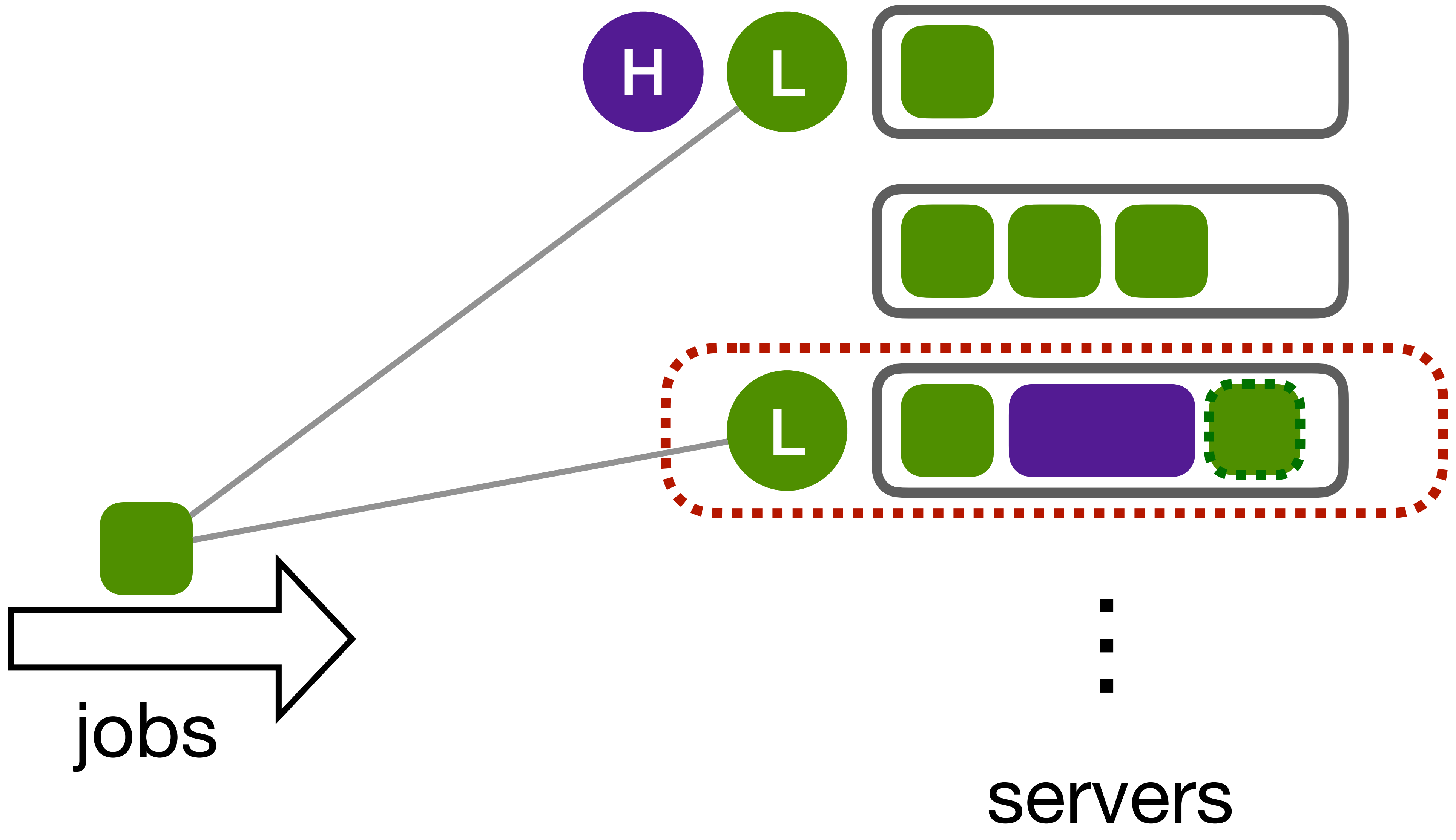
Difficulty: the dynamics of a server in the original system depends on other servers through arrivals & token overflows

Idea: for each type i, consider

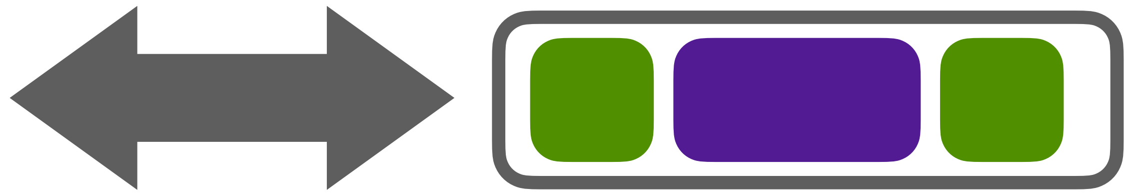
$\widetilde{K}_i =$ # jobs + # virtual jobs + # tokens

Key proof idea 1

Will show that regular servers in the original system $\approx \overline{N}$ independent single-server systems



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



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

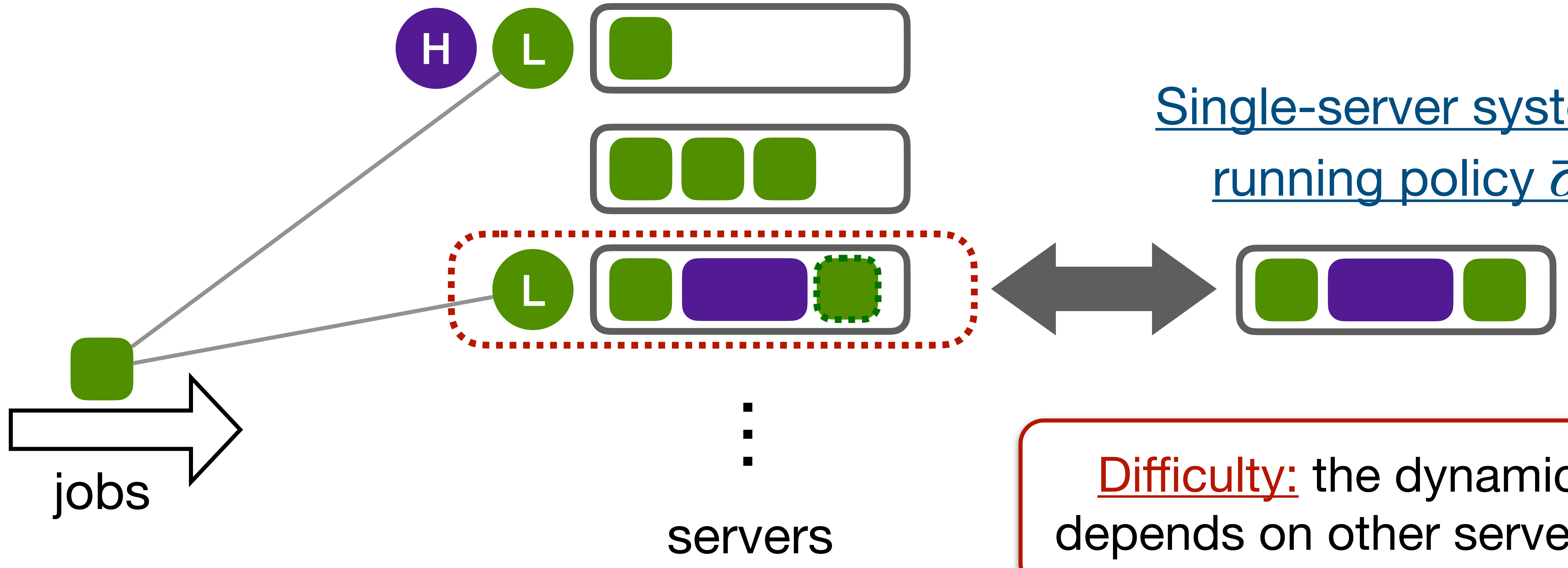
Idea: for each type i, consider

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

Do job arrivals affect $\widetilde{K}_i$?

Key proof idea 1

Will show that regular servers in the original system $\approx \overline{N}$ independent single-server systems



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

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

Idea: for each type i, consider

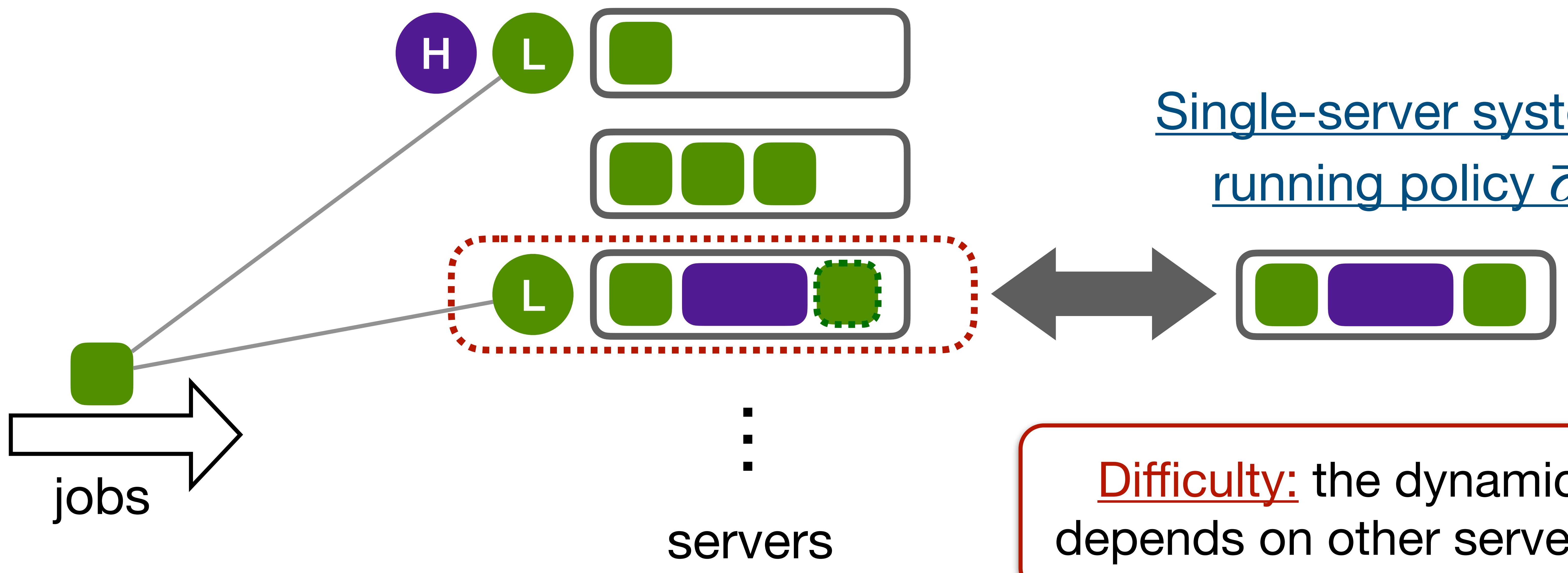
$\widetilde{K}_i$ = # jobs + # virtual jobs + # tokens

Do job arrivals affect $\widetilde{K}_i$?

Do token overflows affect $\widetilde{K}_i$?

Key proof idea 1

Will show that regular servers in the original system $\approx \bar{N}$ independent single-server systems



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

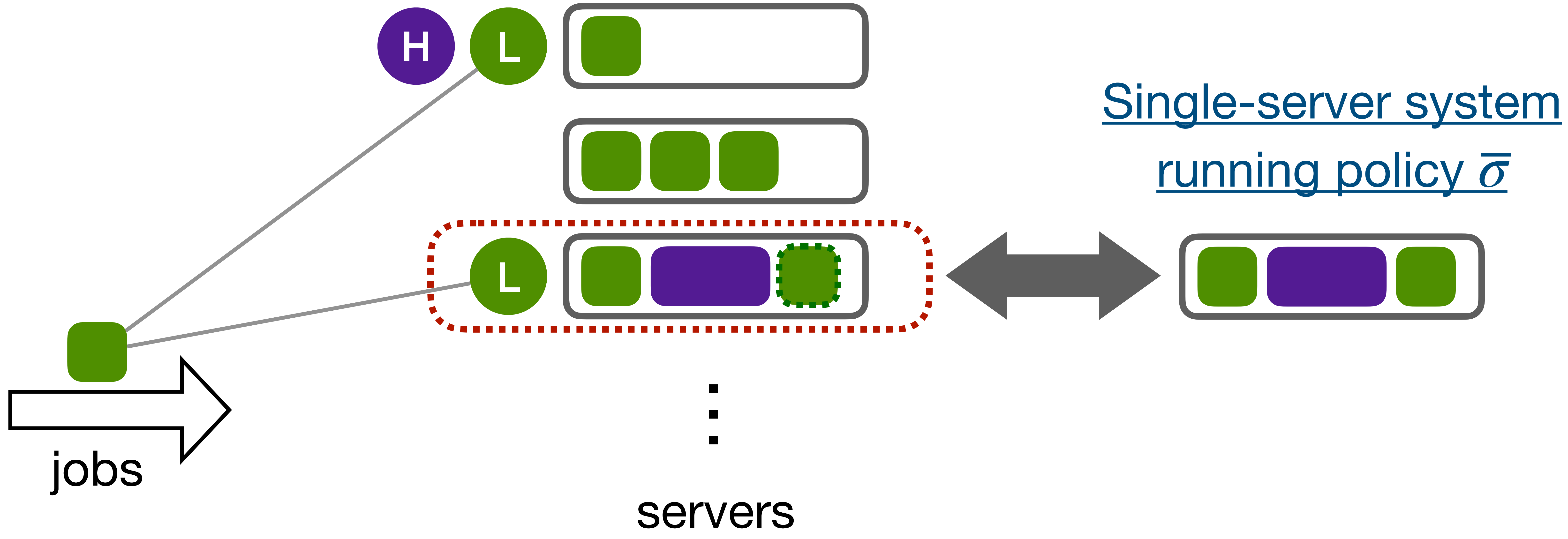
Do job arrivals affect $\widetilde{K}_i$?

Do token overflows affect $\widetilde{K}_i$?

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

Key proof idea 1

Will show that regular servers in the original system $\approx \overline{N}$ independent single-server systems



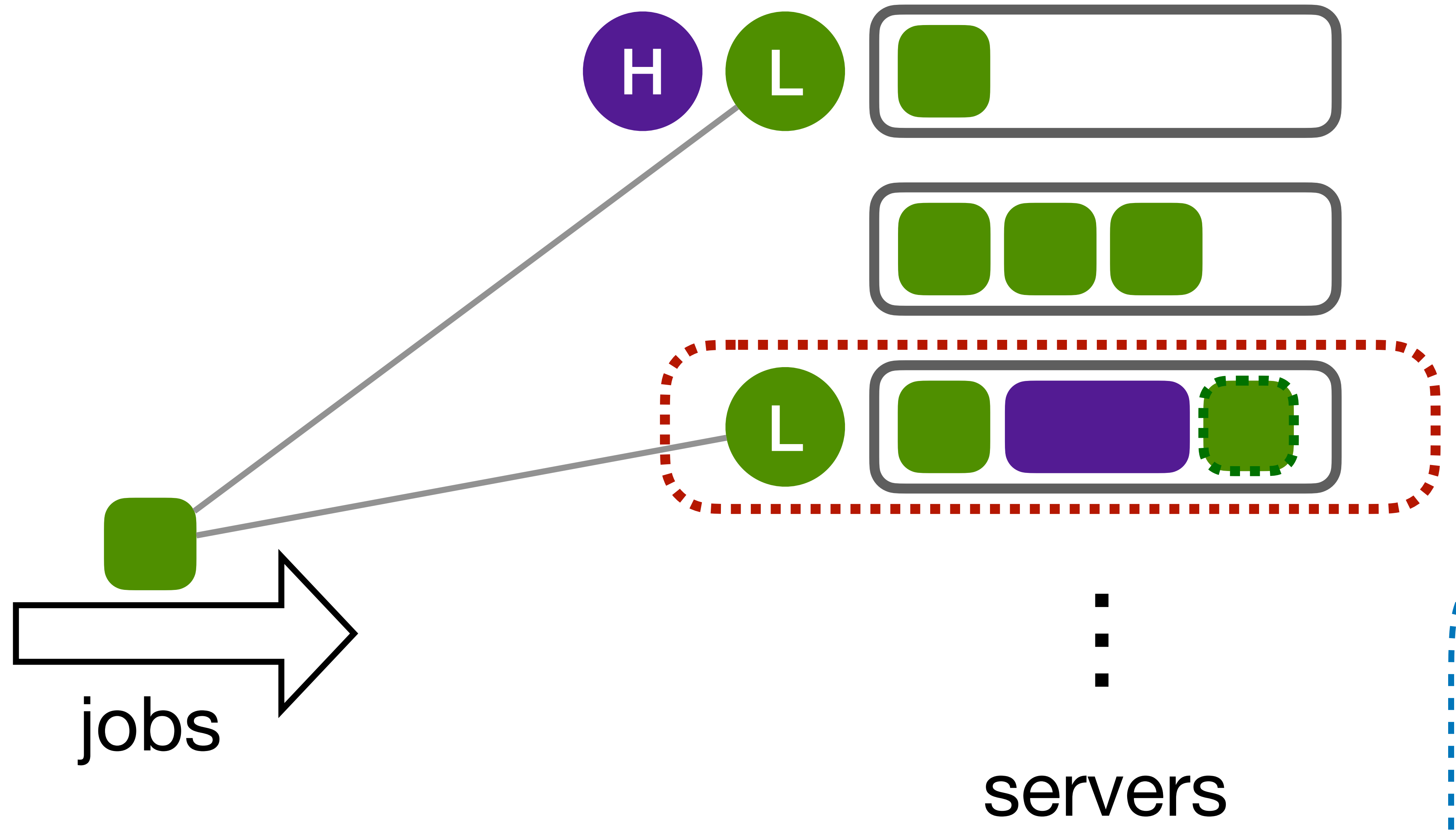
Idea: for each type i, consider

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

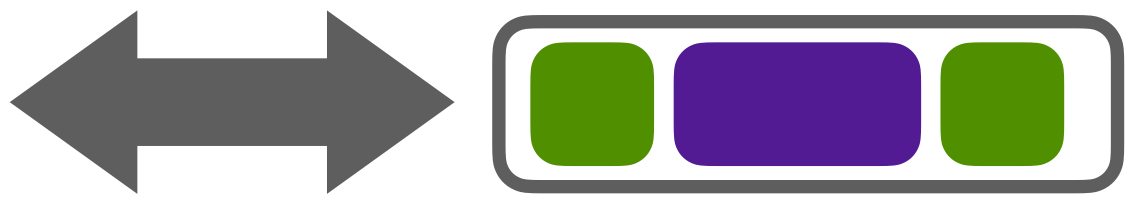
Arrivals & token overflows do not affect $\widetilde{K}_i$

Key proof idea 1

Will show that regular servers in the original system
$\approx \overline{N}$ independent single-server systems



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



Dynamics in the original system v.s.
Dynamics in $\overline{N}$ independent single-server systems

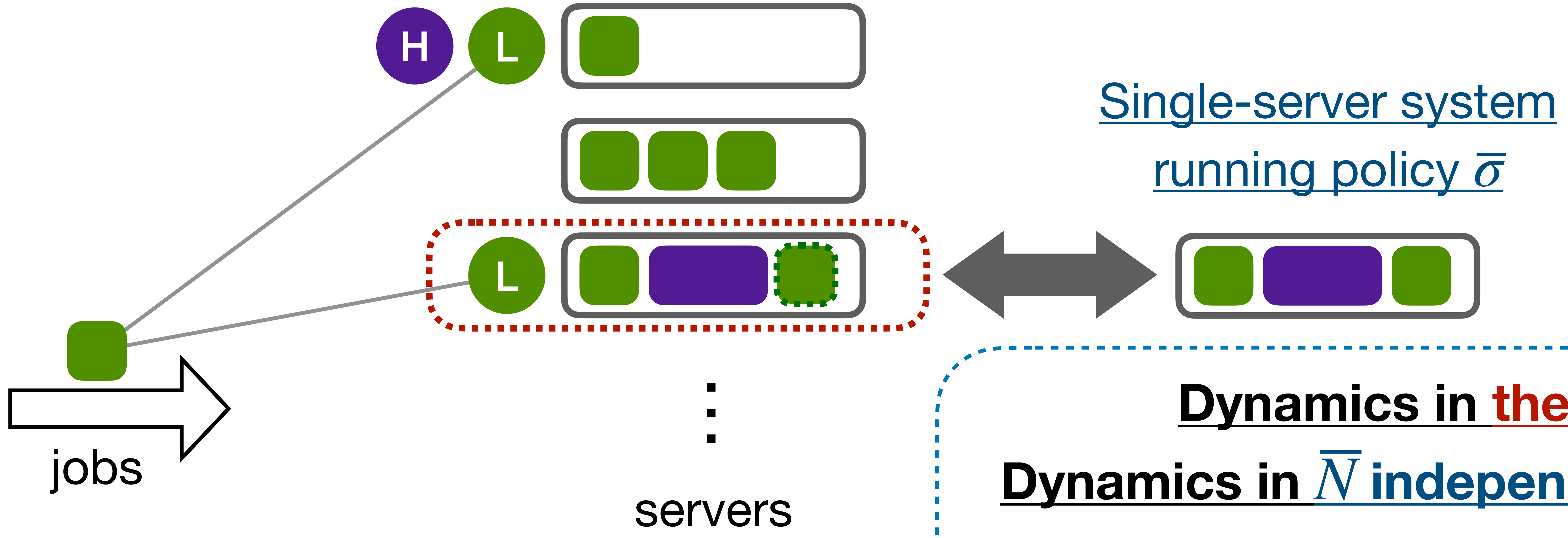
Idea: for each type i, consider

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

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

Key proof idea 1

Will show that regular servers in the original system $\approx \overline{N}$ independent single-server systems



Idea: for each type i, consider

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

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

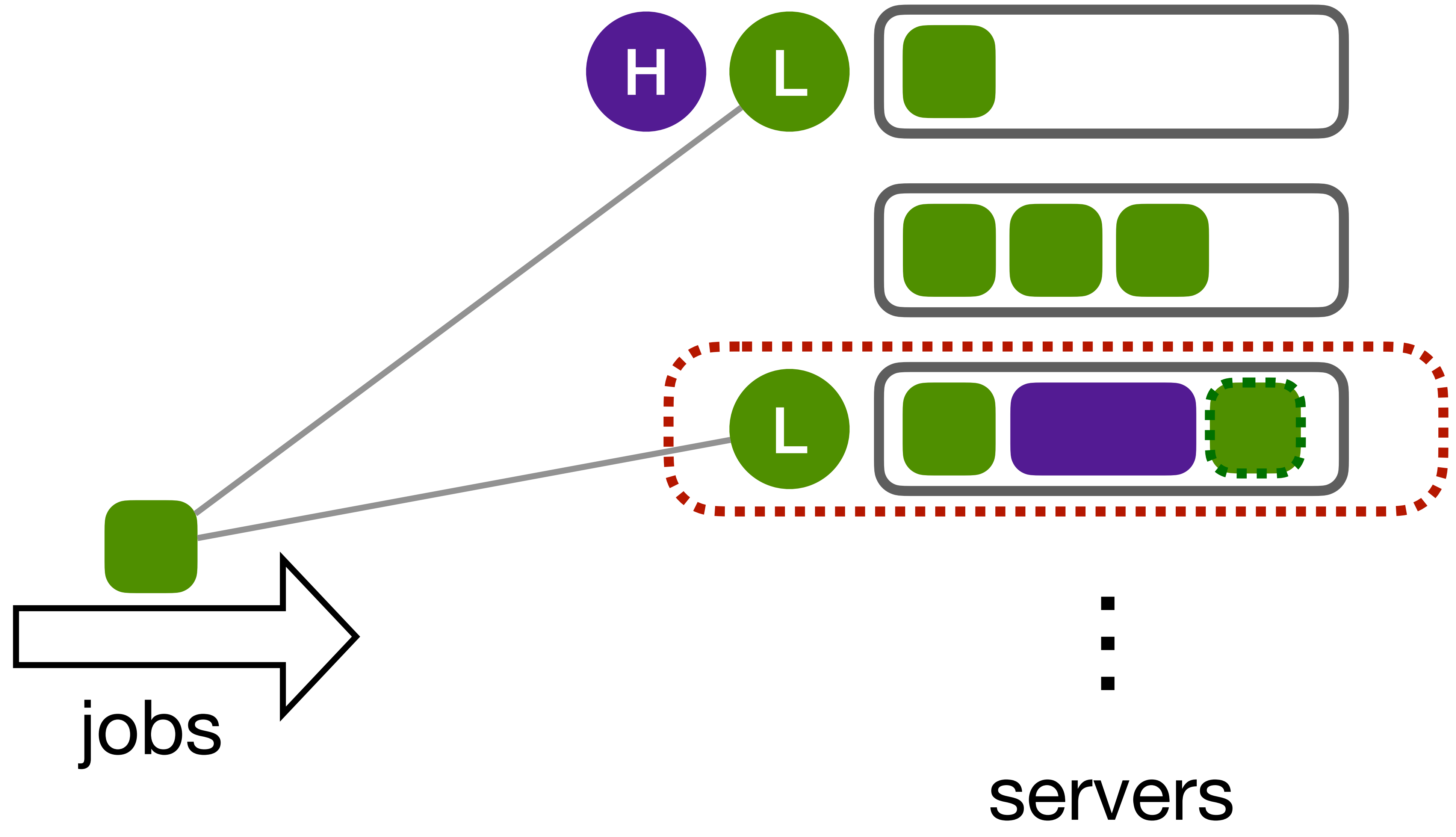
Dynamics in the original system v.s.

Dynamics in $\overline{N}$ independent single-server systems

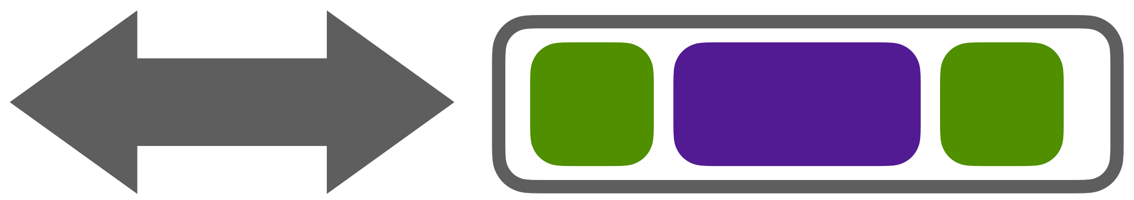
- A server without tokens in the original system generates tokens and has job transitions in the same way as a single-server system

Key proof idea 1

Will show that regular servers in the original system $\approx \overline{N}$ independent single-server systems



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



Idea: for each type i, consider

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

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

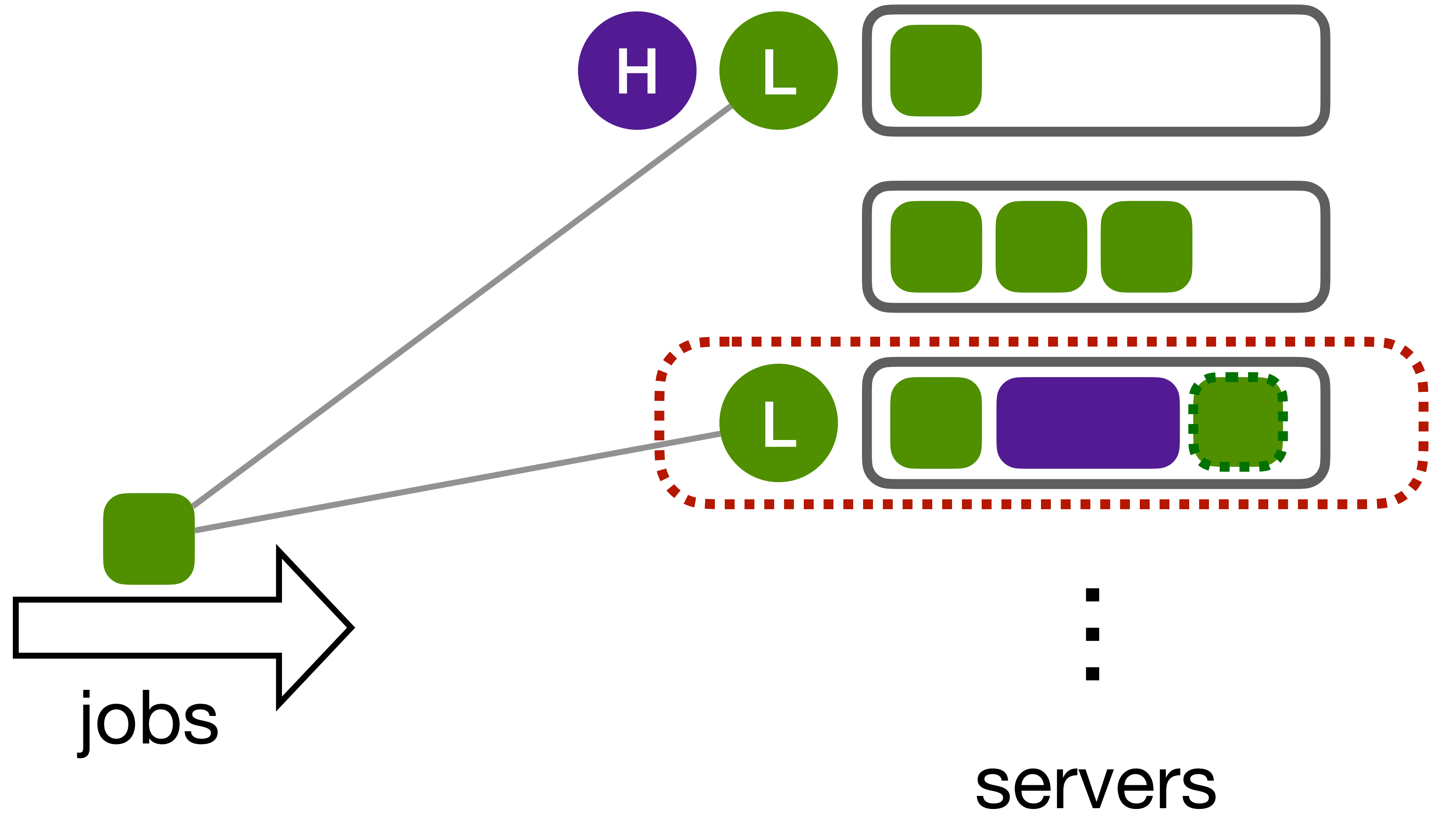
Dynamics in the original system v.s.

Dynamics in $\overline{N}$ independent single-server systems

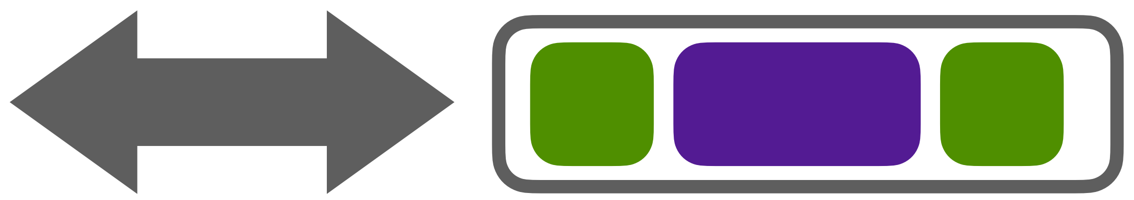
- A server without tokens in the original system generates tokens and has job transitions in the same way as a single-server system
- Difference between two systems is bounded by # tokens

Key proof idea 1

Will show that regular servers in the original system $\approx \overline{N}$ independent single-server systems



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



Idea: for each type i, consider

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

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

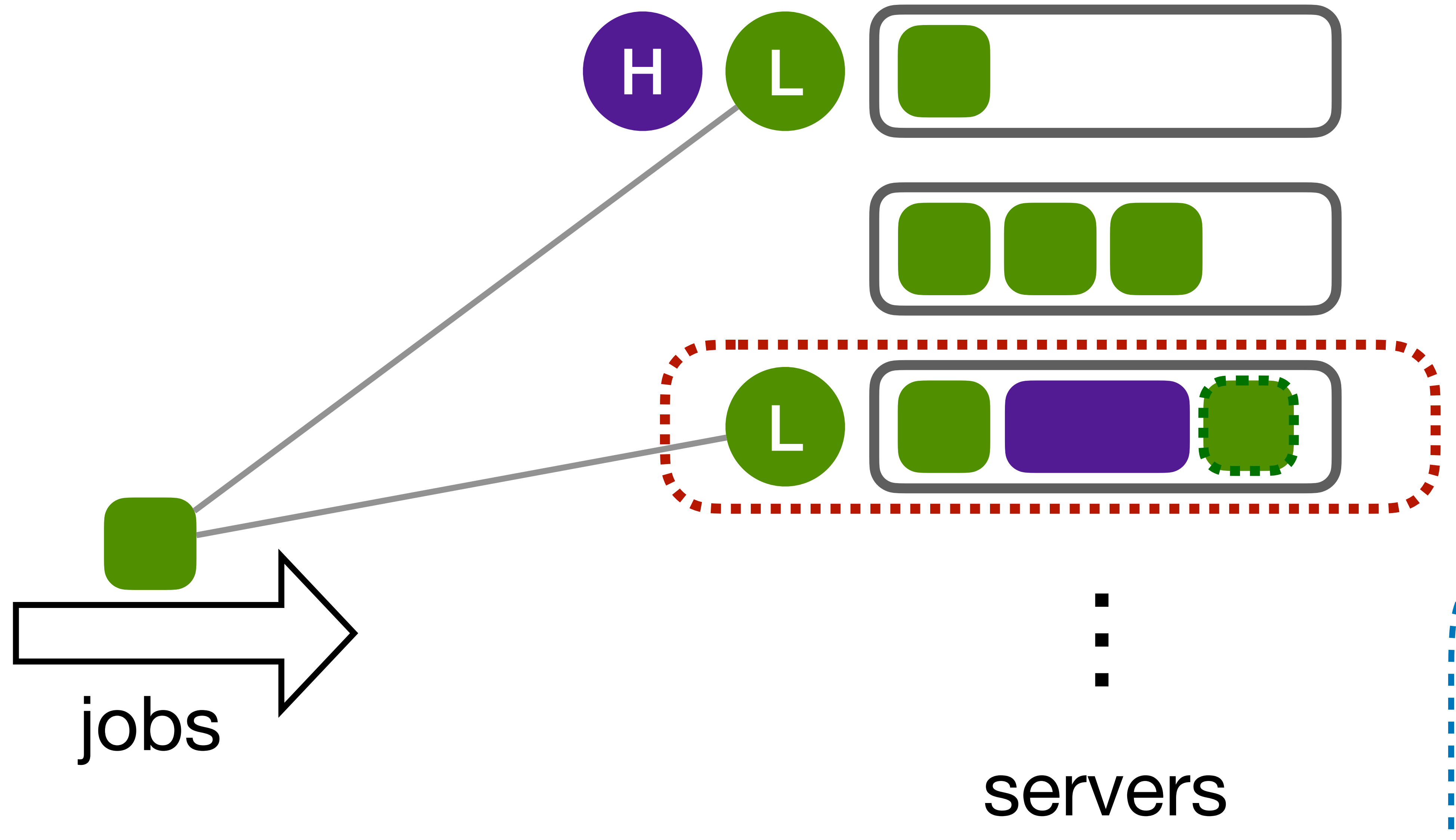
Dynamics in the original system v.s.

Dynamics in $\overline{N}$ independent single-server systems

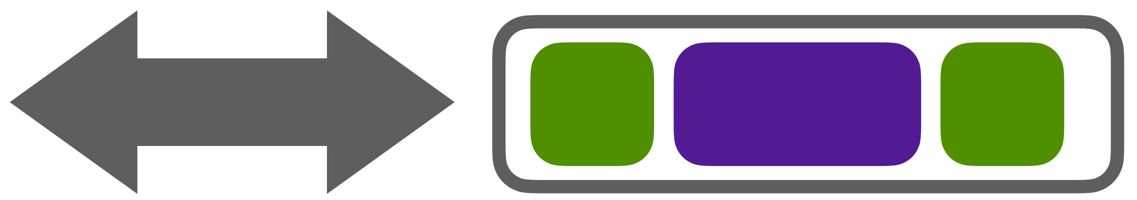
- A server without tokens in the original system generates tokens and has job transitions in the same way as a single-server system
- Difference between two systems is bounded by # tokens
- # tokens $\leq \sqrt{r}$

Key proof idea 1

Will show that regular servers in the original system $\approx \overline{N}$ independent single-server systems



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



Idea: for each type i, consider

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

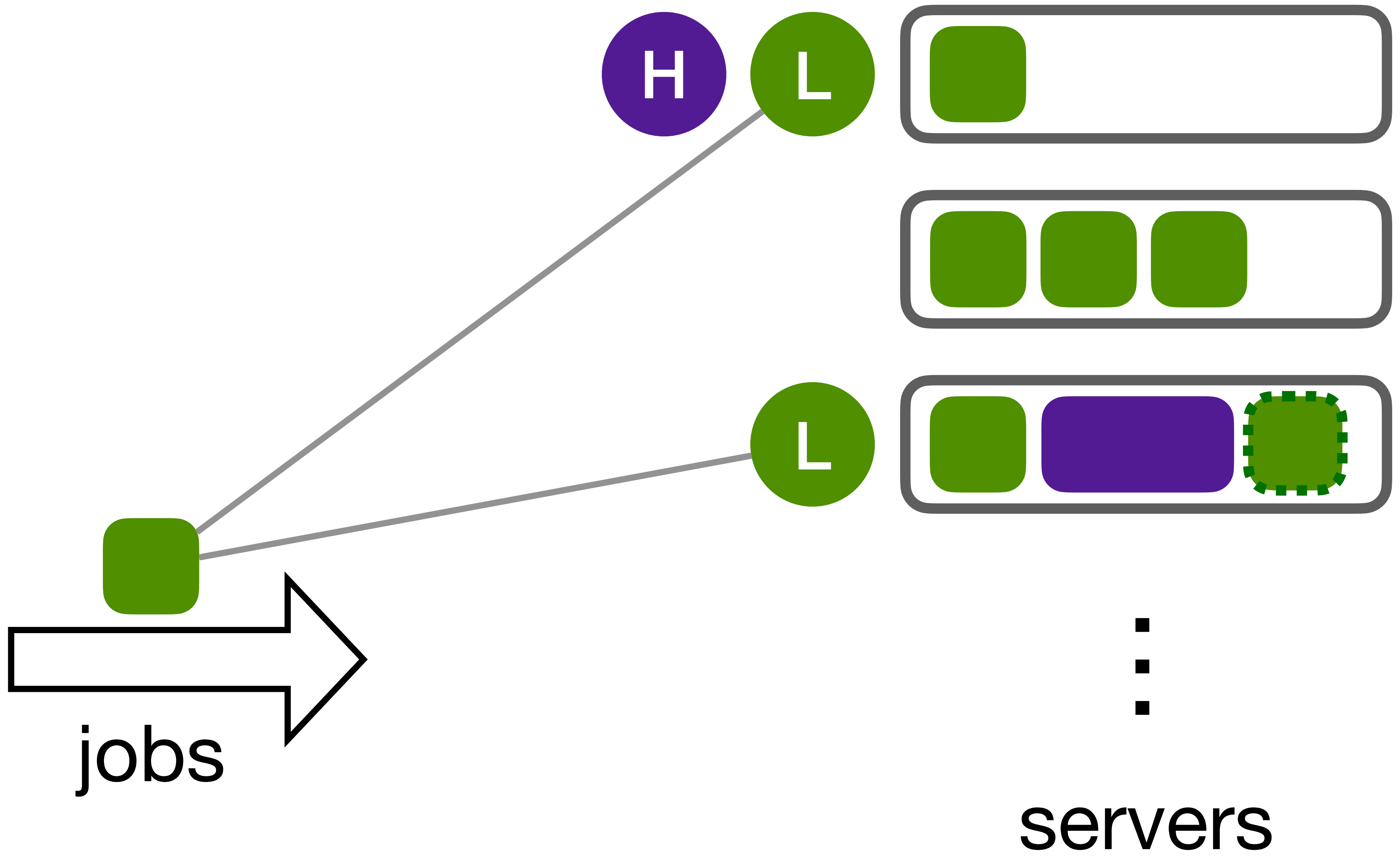
Arrivals & token overflows do not affect $\widetilde{K}_i$

Dynamics in the original system v.s. Dynamics in $\overline{N}$ independent single-server systems

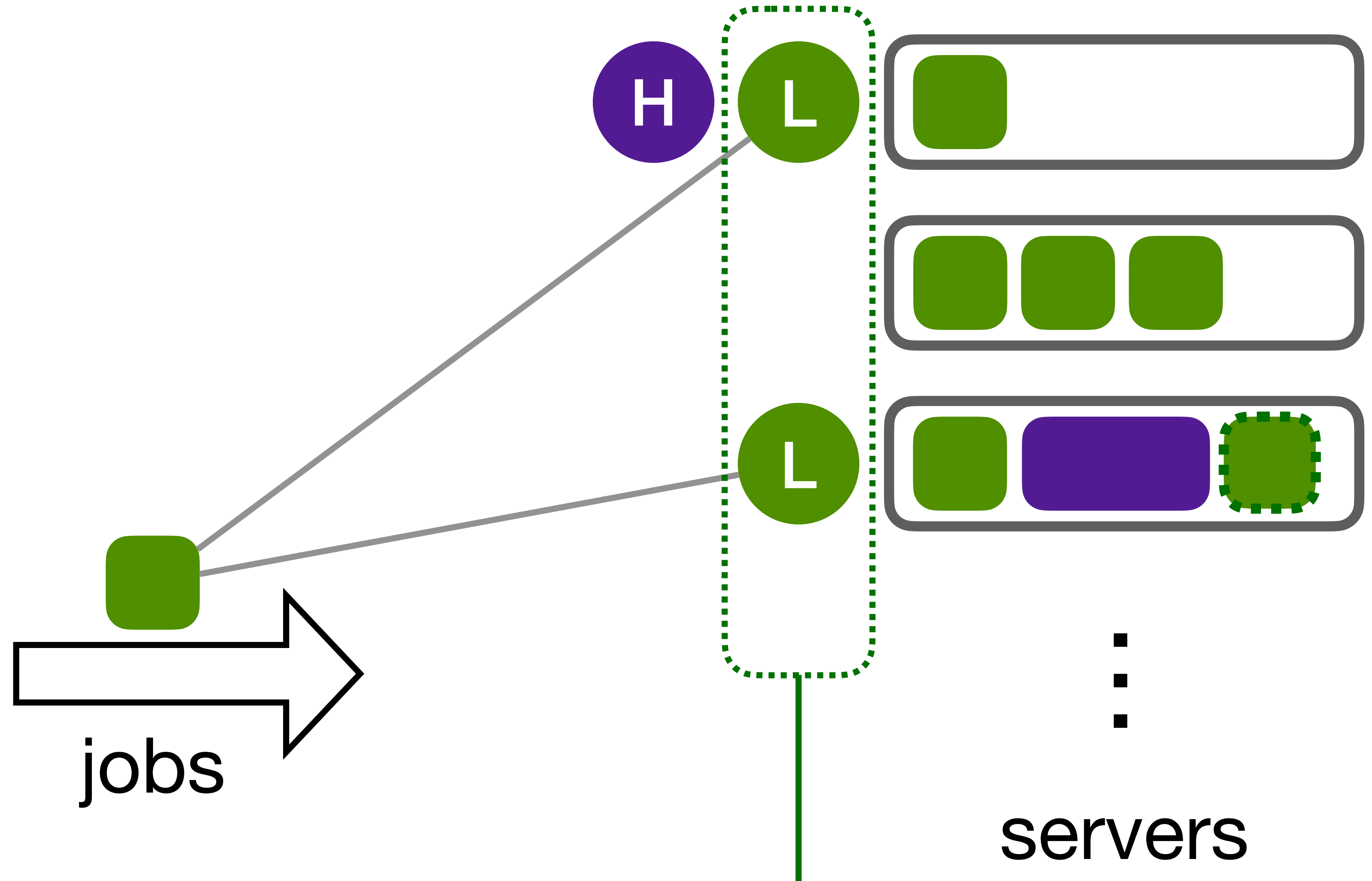
- A server without tokens in the original system generates tokens and has job transitions in the same way as a single-server system
- Difference between two systems is bounded by # tokens
- # tokens $\leq \sqrt{r}$

$\Rightarrow$ Using Stein's method, $d_W\left(\widetilde{K}^{1:\overline{N}}, \overline{K}^{1:\overline{N}}\right) = O\left(r^{0.5}\right)$

Key proof idea 2

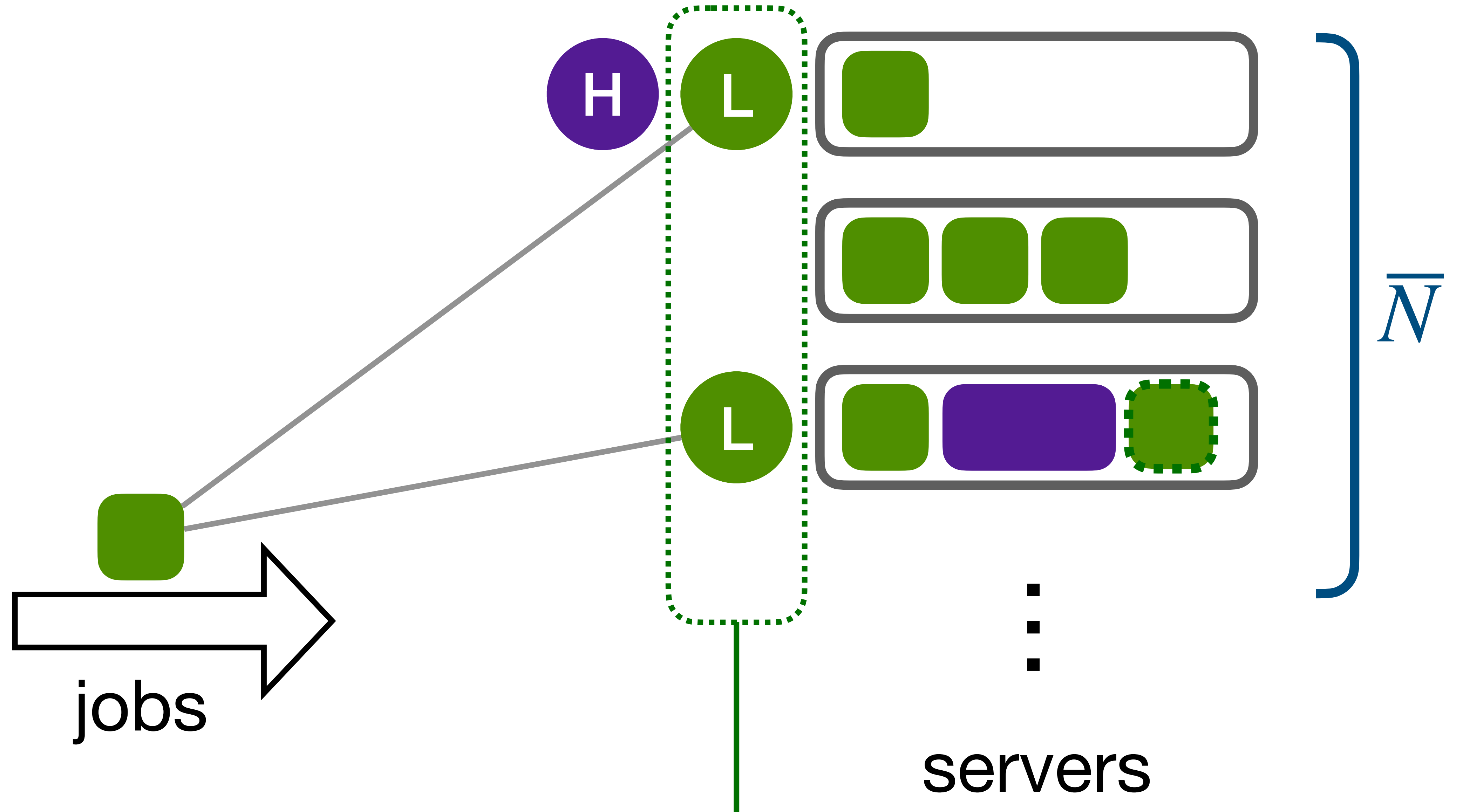


Key proof idea 2



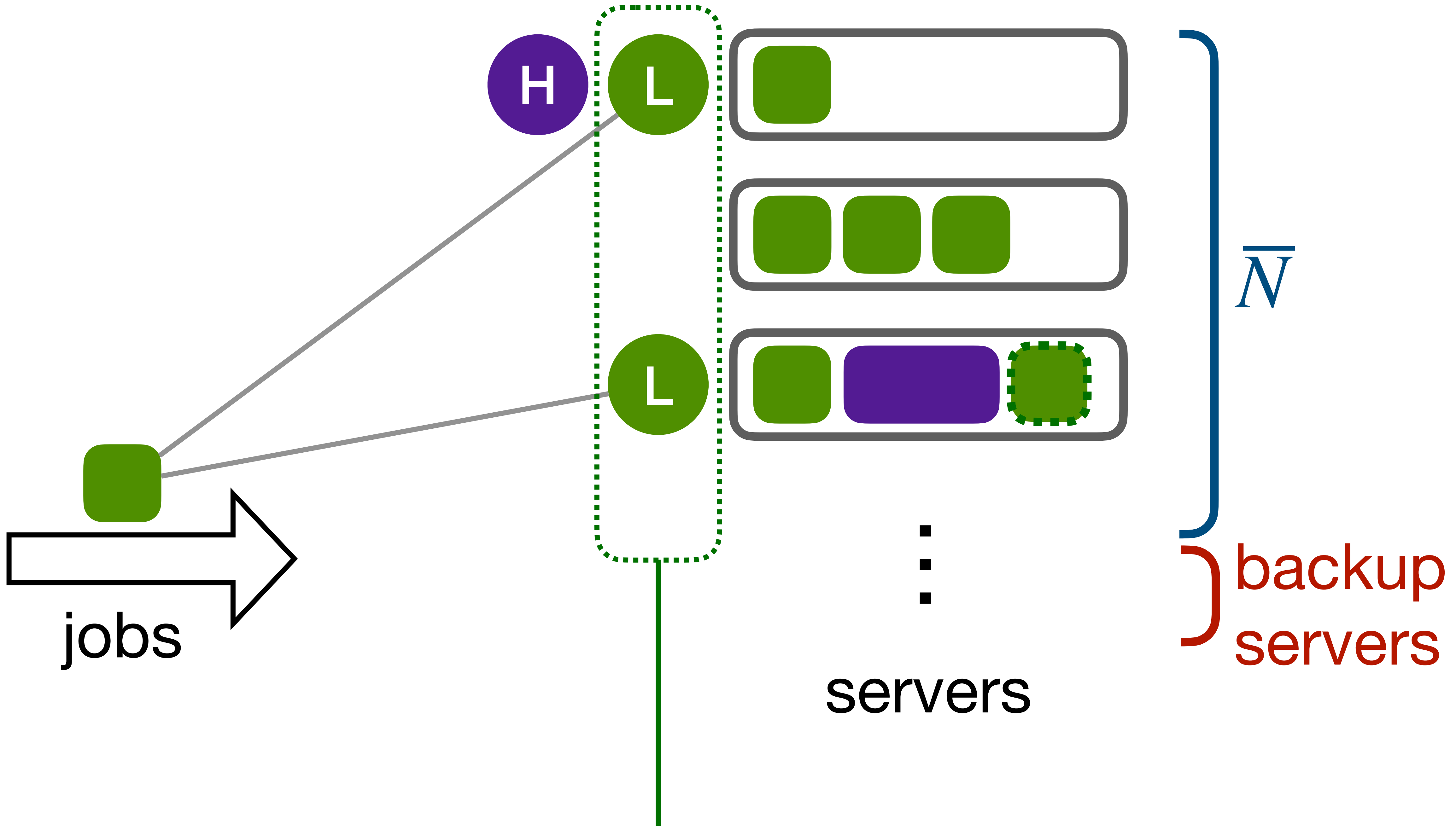
When the # tokens of a type $> \sqrt{r}$, remove the overflow tokens and generate virtual jobs

Key proof idea 2



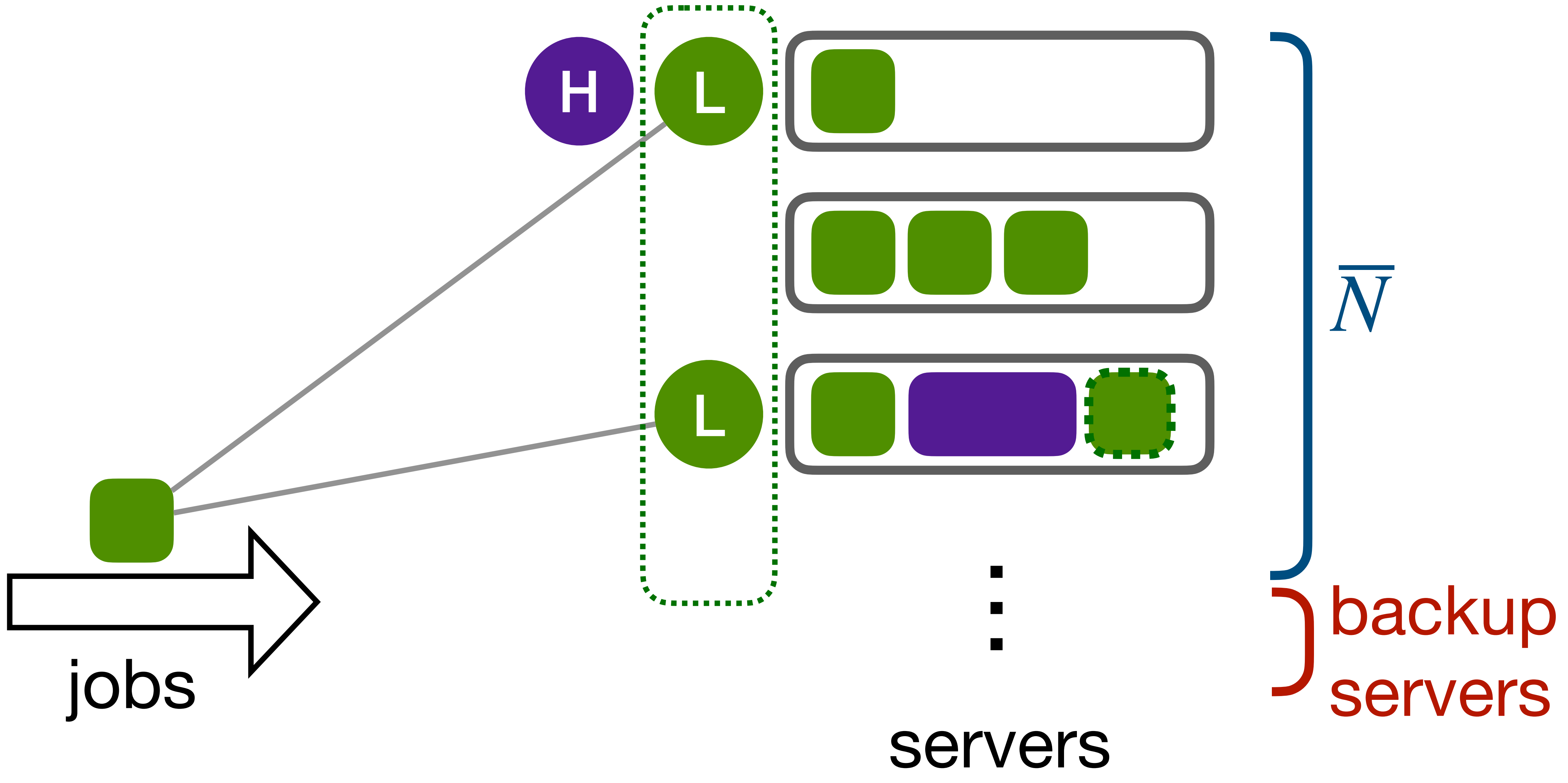
When the # tokens of a type $> \sqrt{r}$, remove the overflow tokens and generate virtual jobs

Key proof idea 2



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

Key proof idea 2



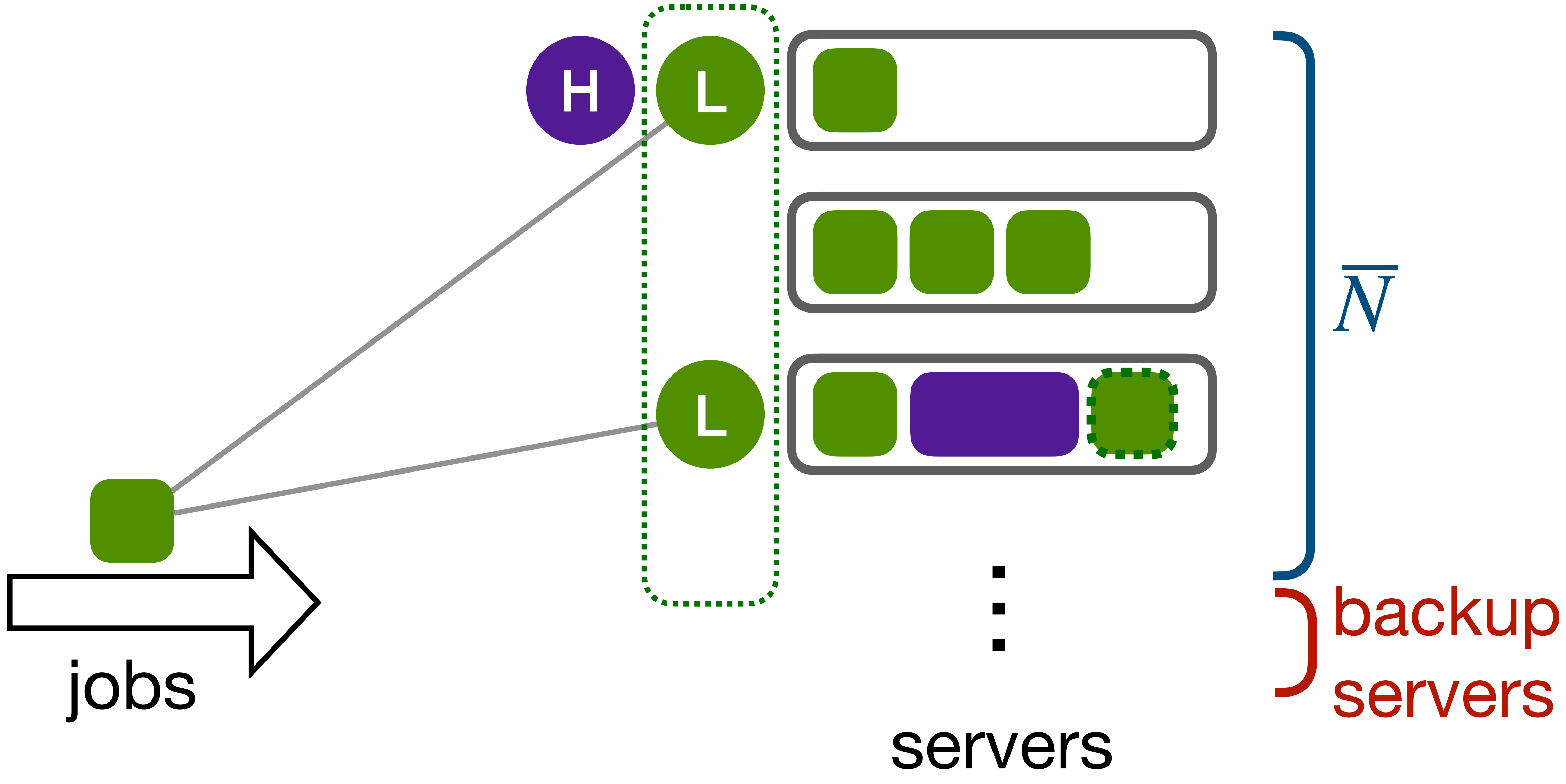
Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

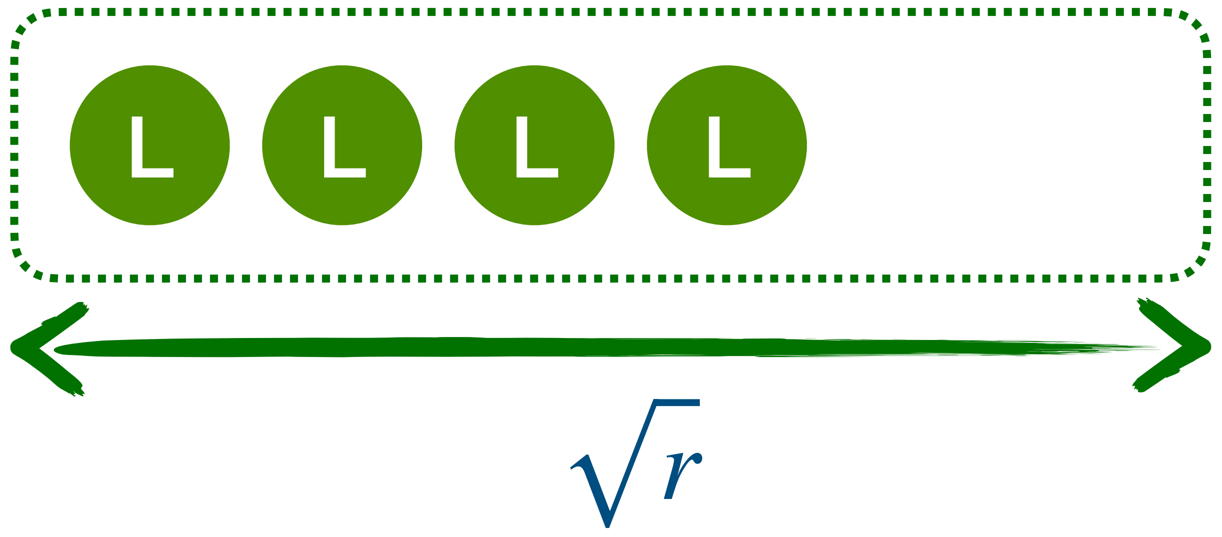
and # backup servers $= O\left(\sqrt{r}\right)$

Key proof idea 2

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

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

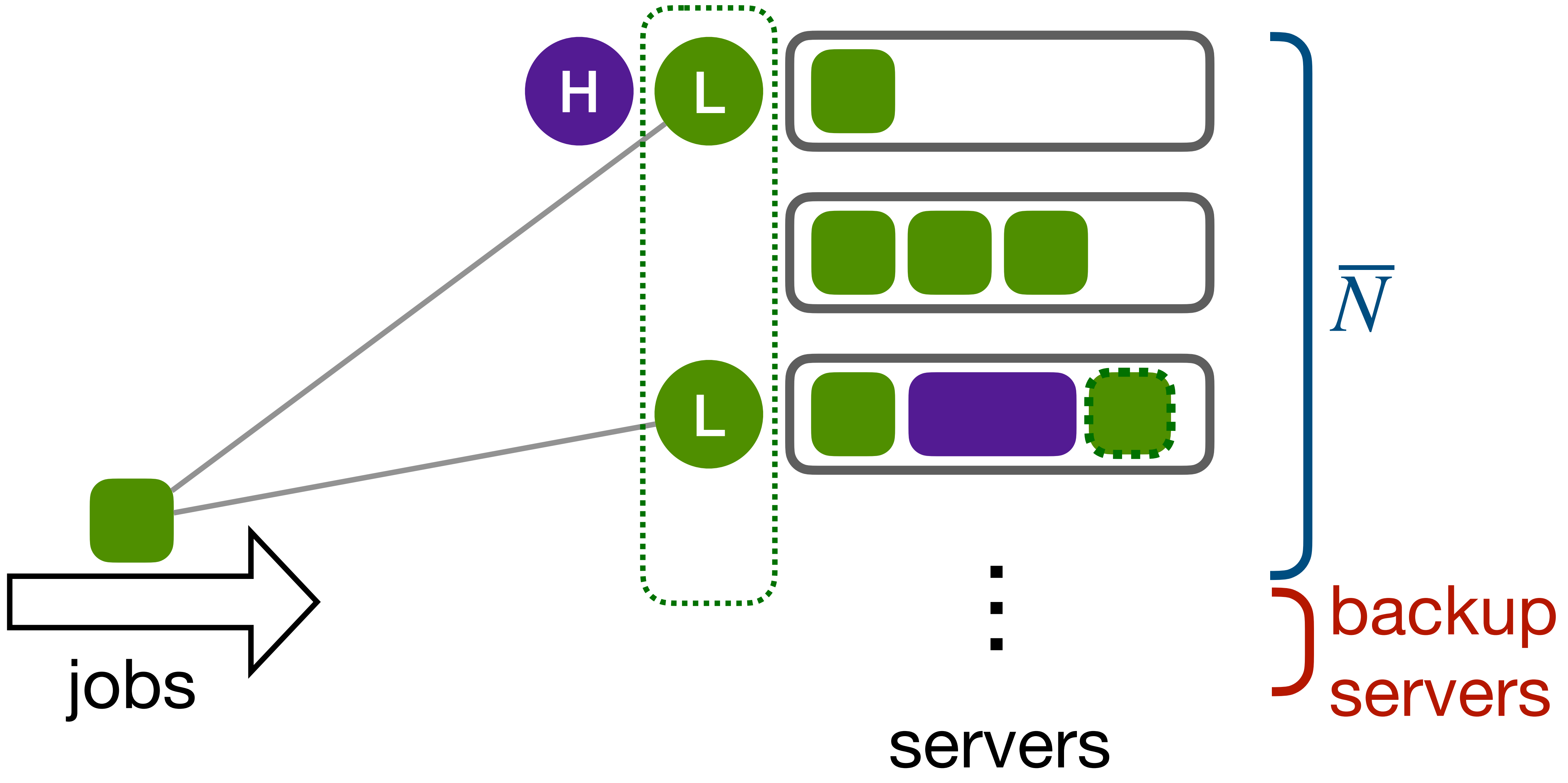


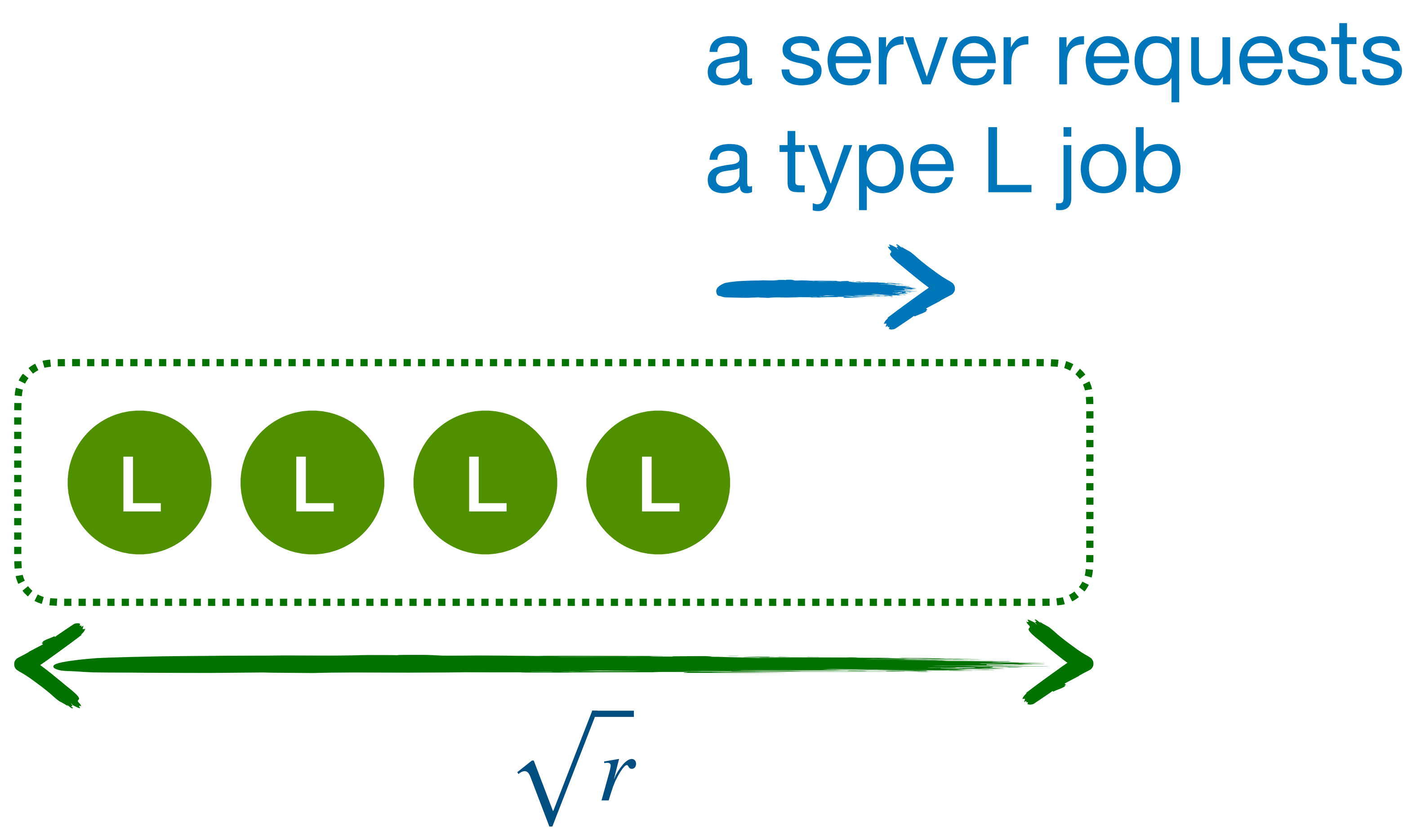


Key proof idea 2

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

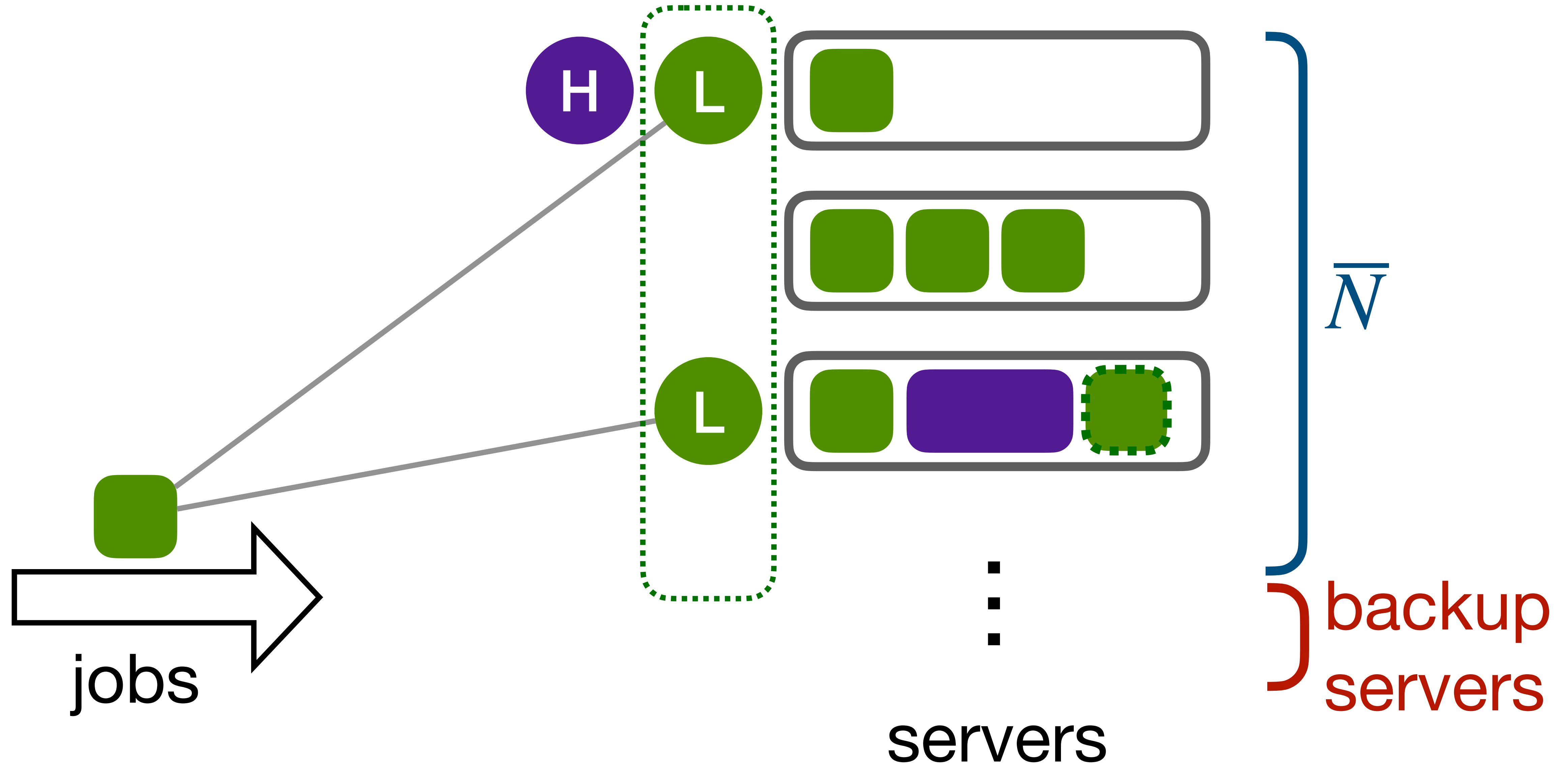
and # backup servers $= O\left(\sqrt{r}\right)$

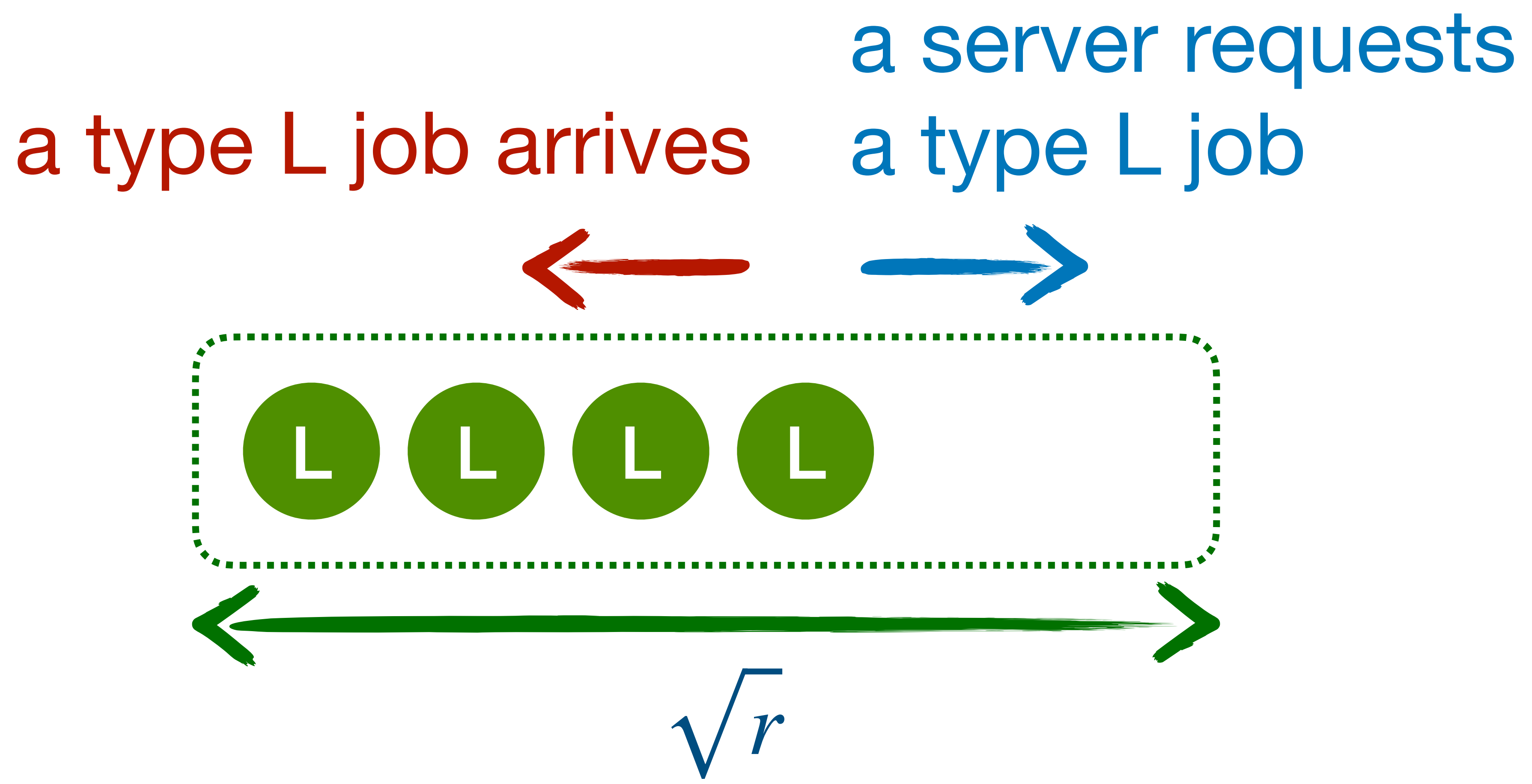




Key proof idea 2

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

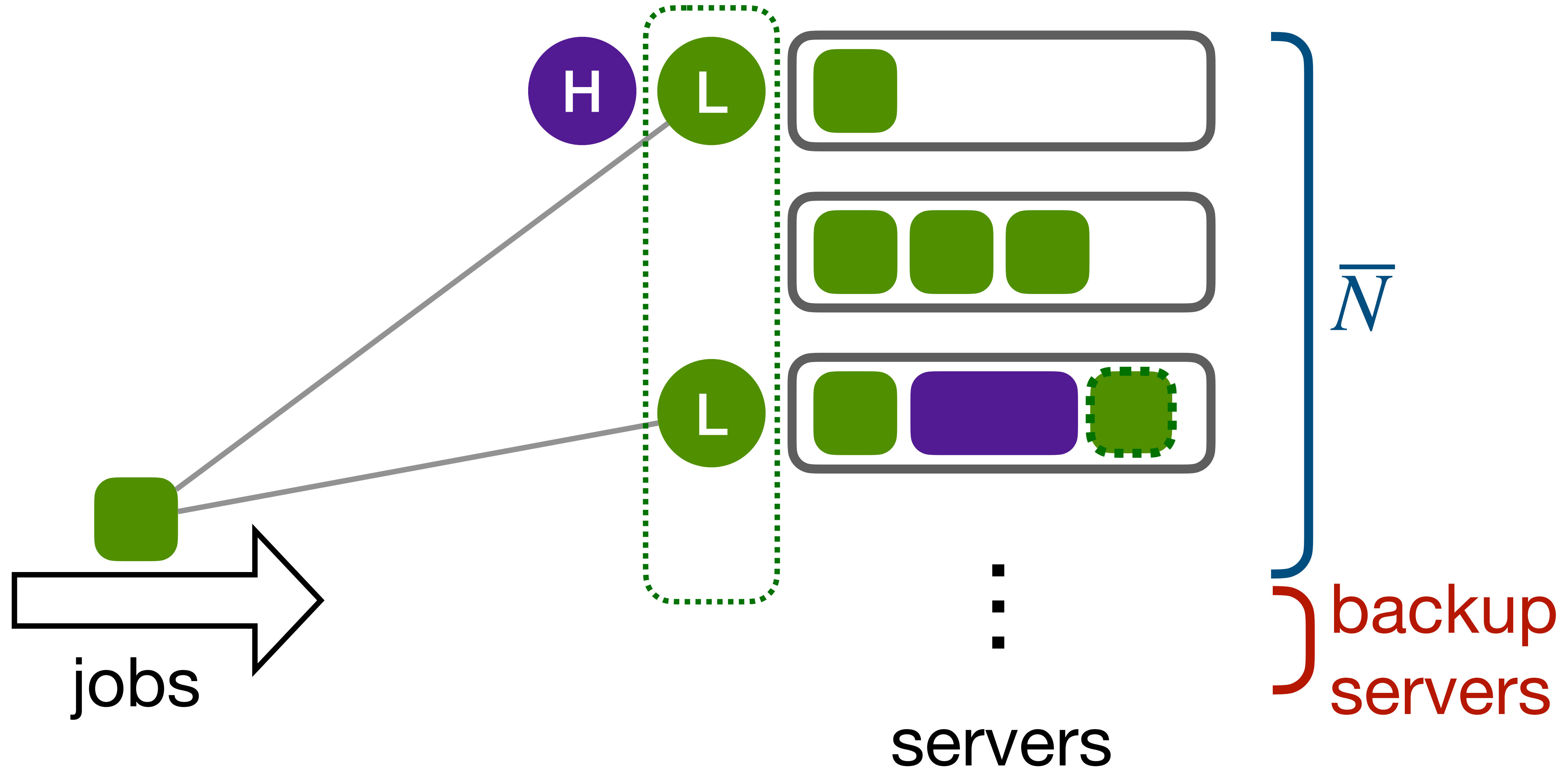


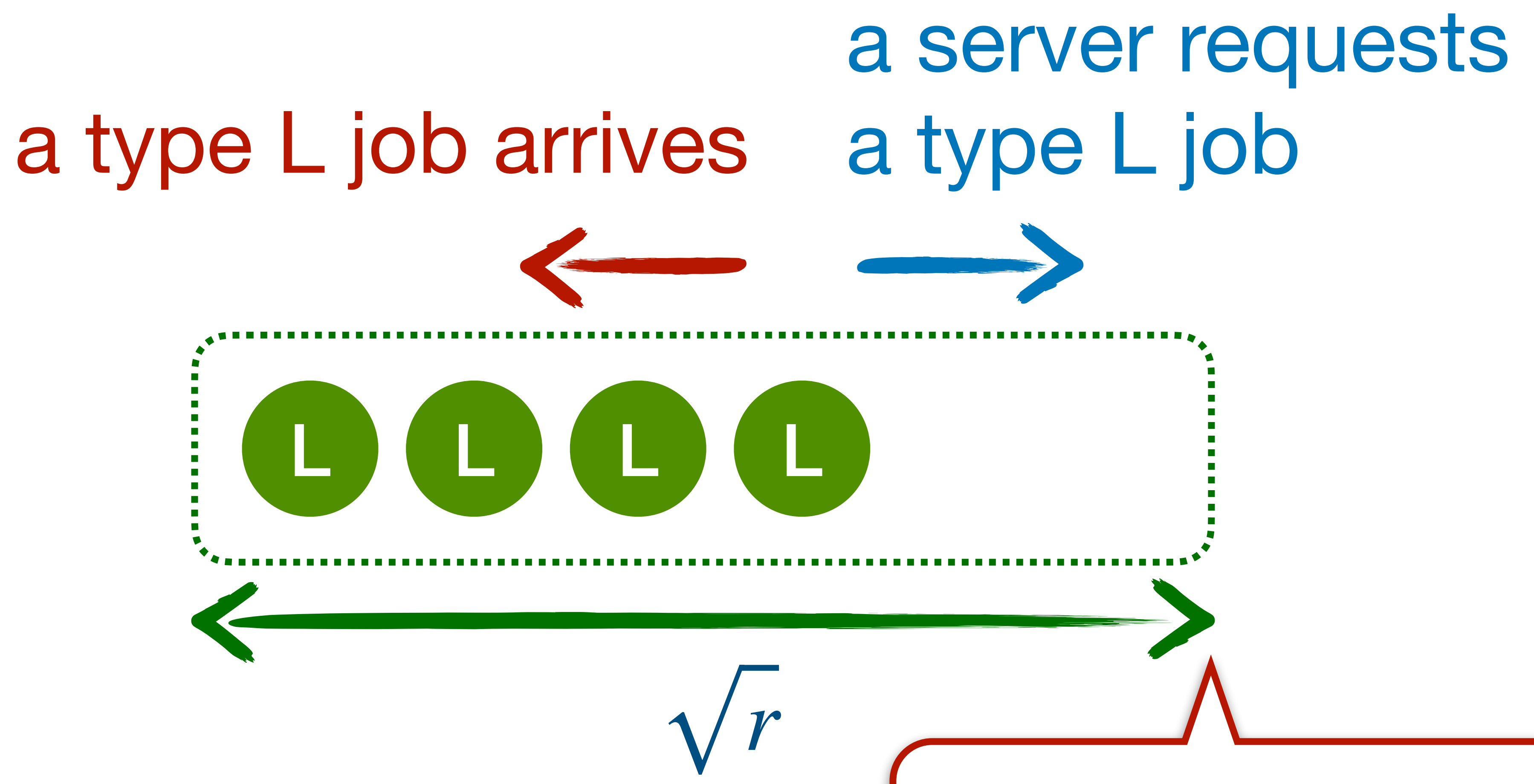


Key proof idea 2

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

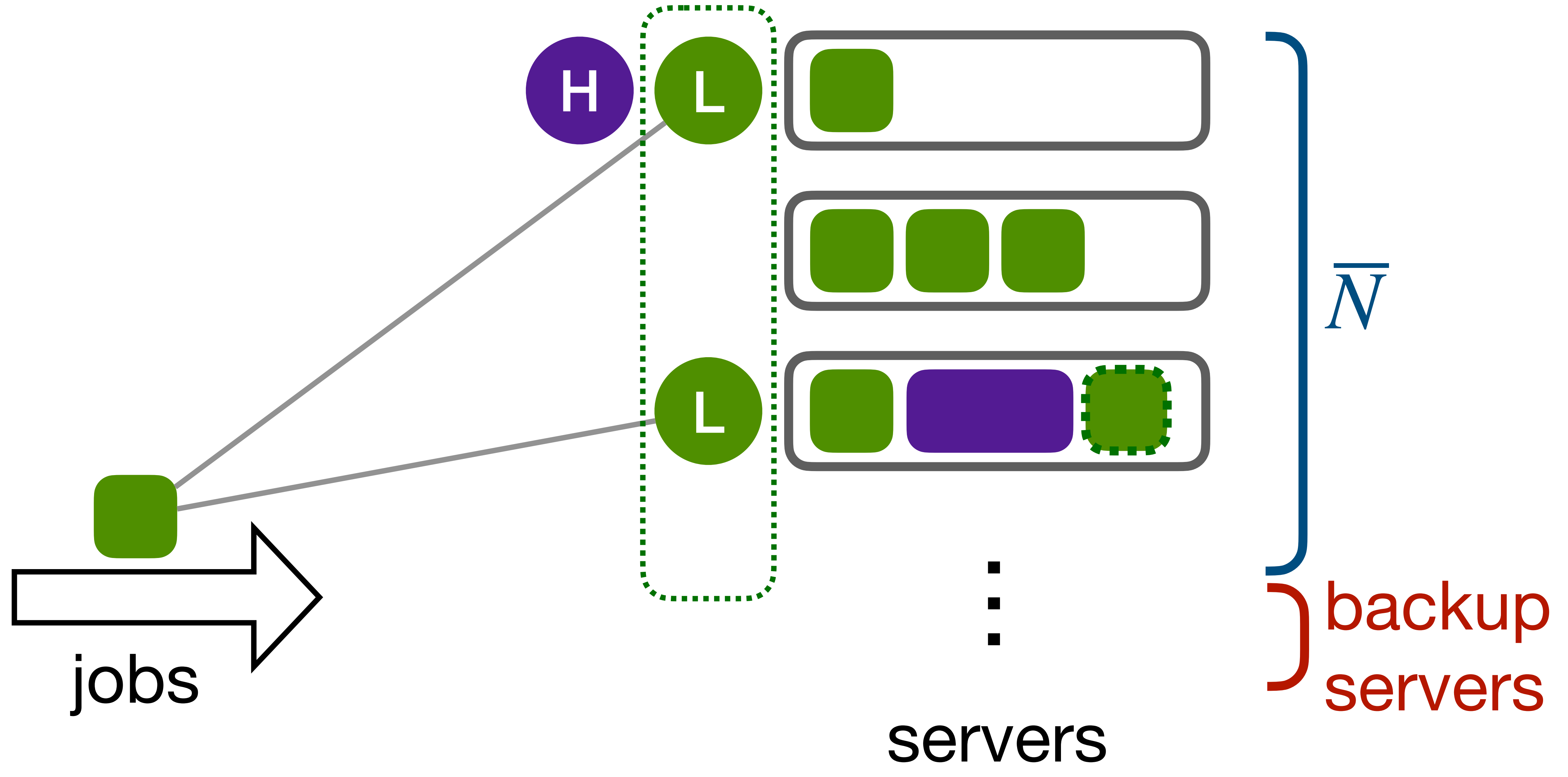
and # backup servers $= O\left(\sqrt{r}\right)$





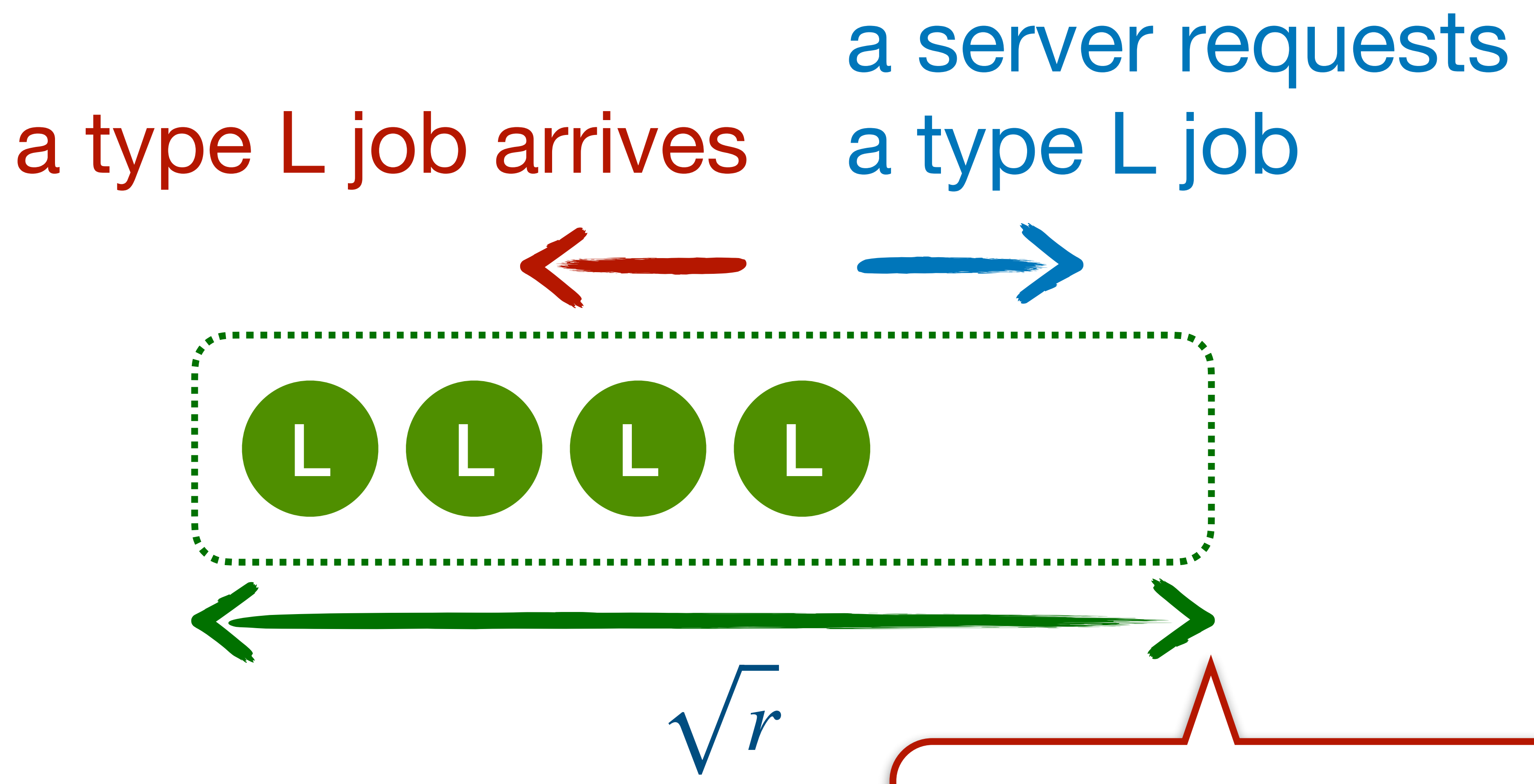
What happens when # tokens hits $\sqrt{r}$?

Key proof idea 2



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

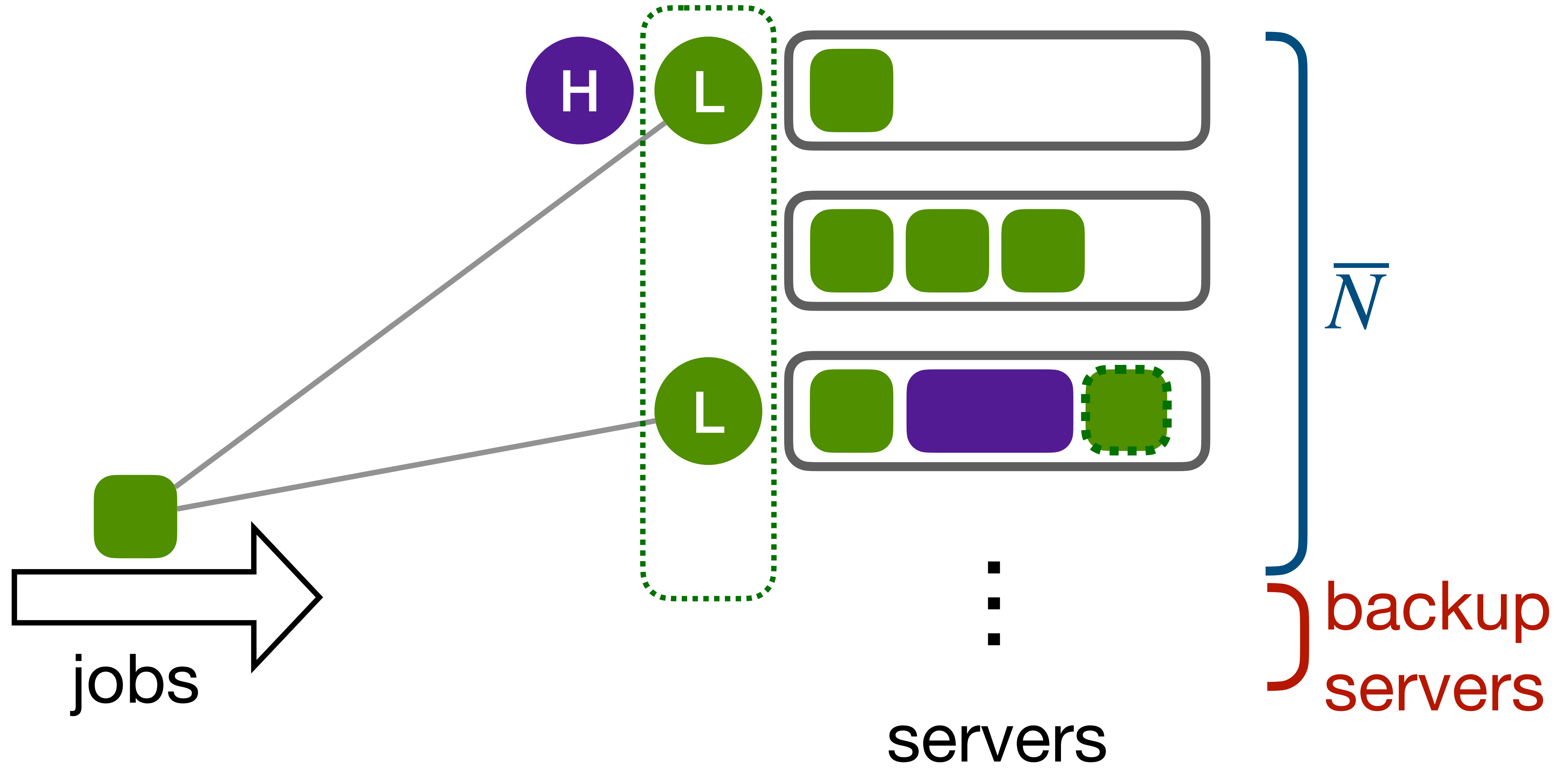
and # backup servers $= O\left(\sqrt{r}\right)$



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

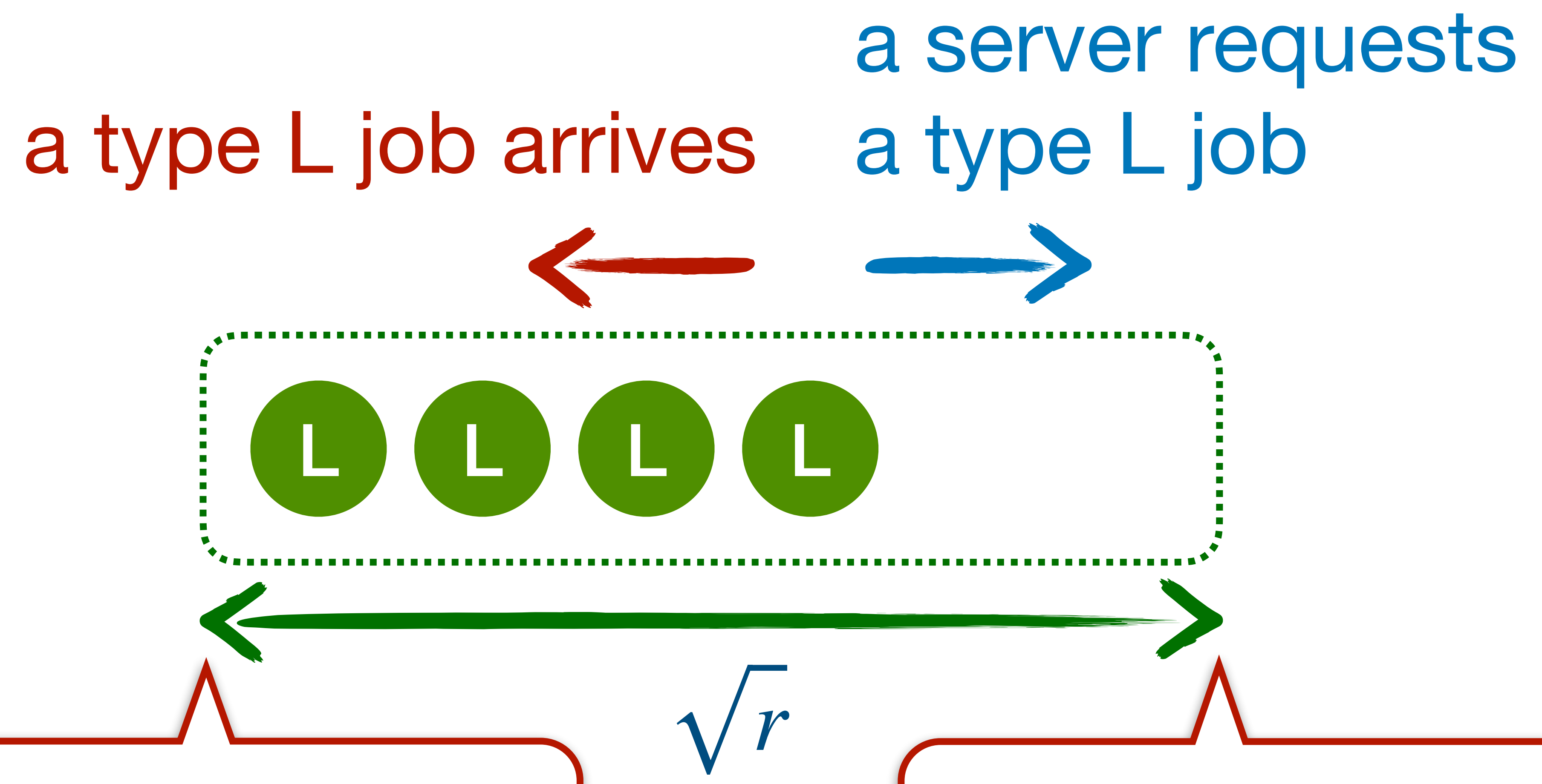
Generate a virtual job

Key proof idea 2



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

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



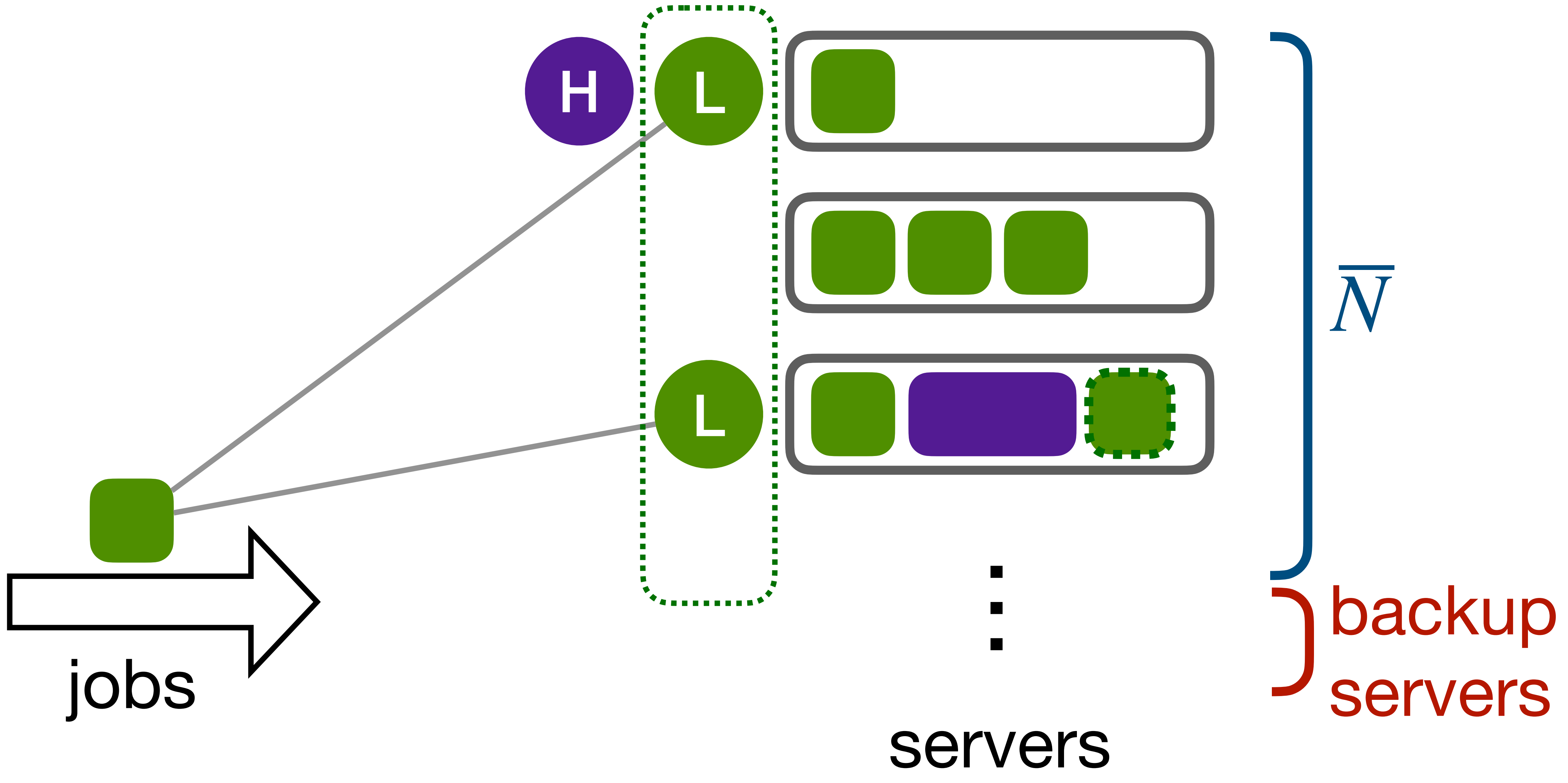
What happens when # tokens hits 0?

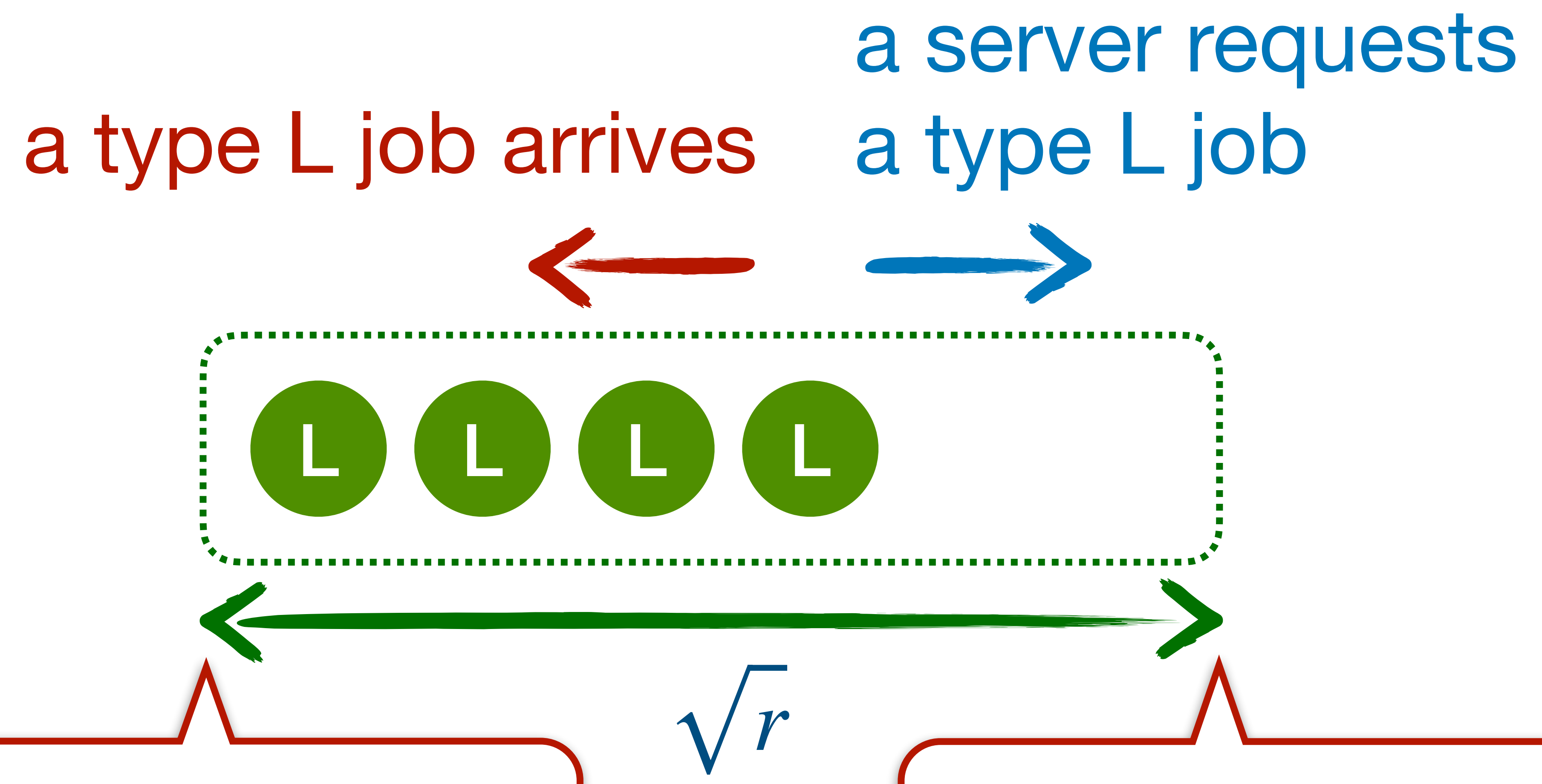
What happens when # tokens hits $\sqrt{r}$?

Generate a virtual job

Key proof idea 2

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





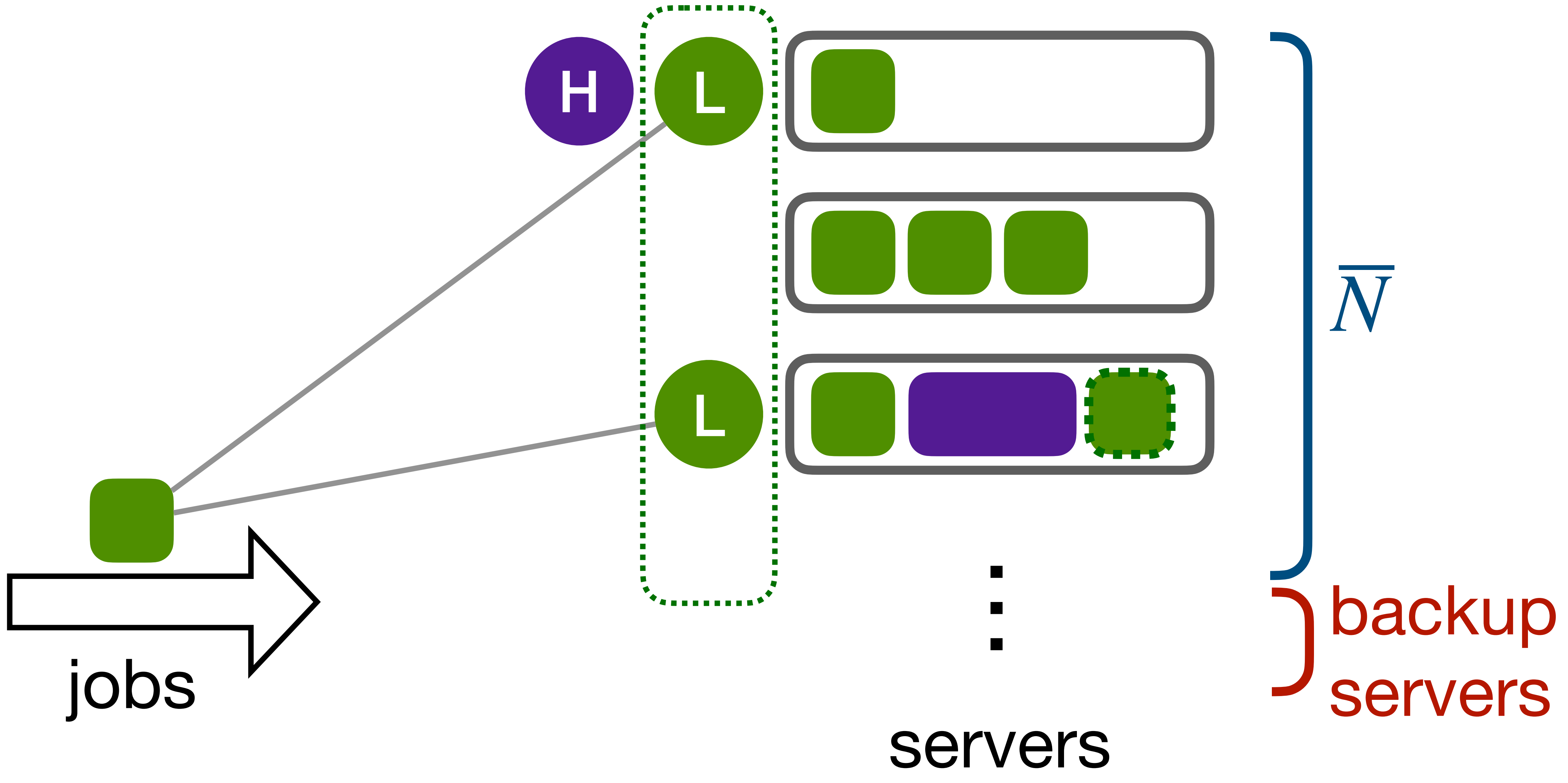
What happens when # tokens hits 0?

Generate a job to backup servers

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

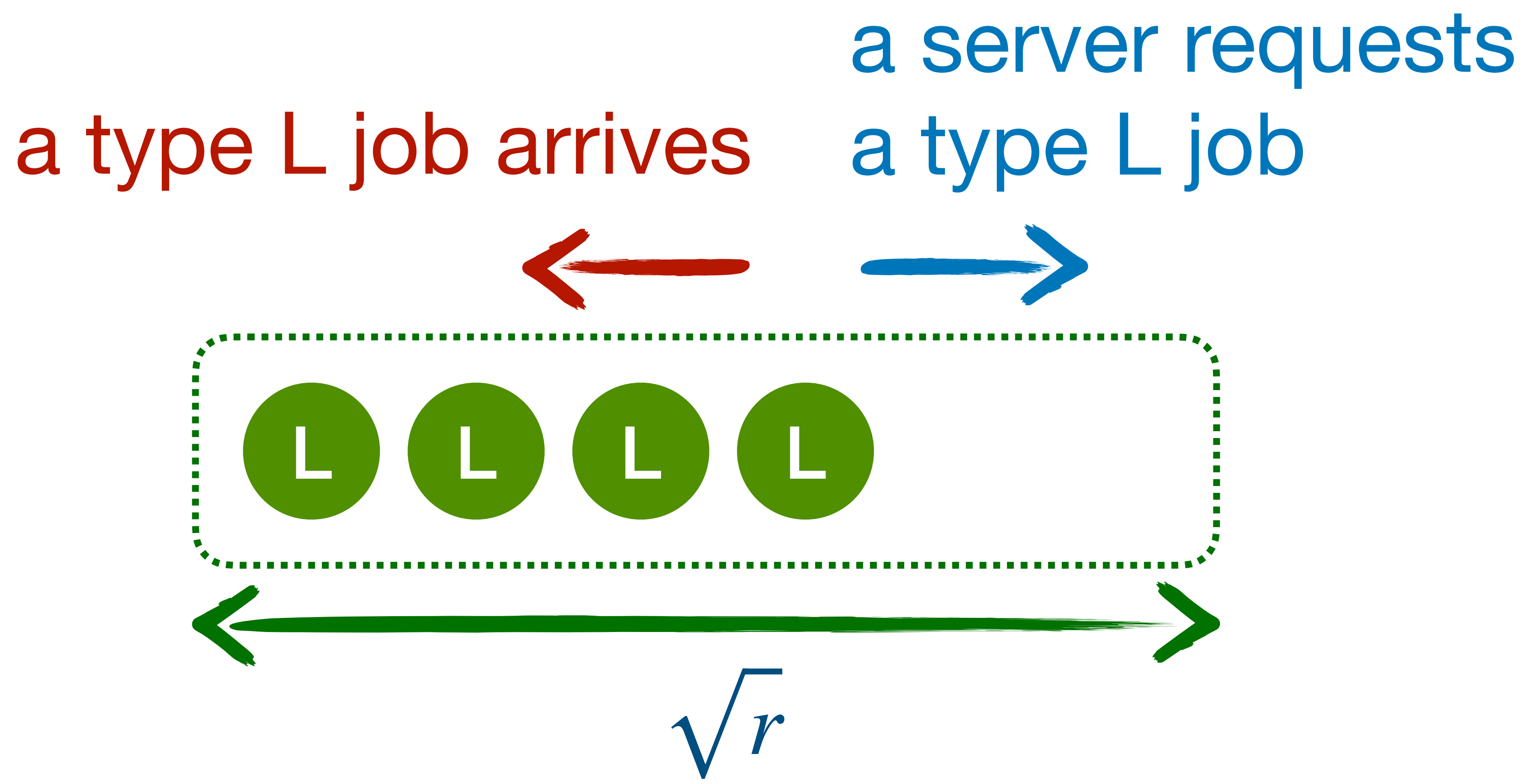
Generate a virtual job

Key proof idea 2



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

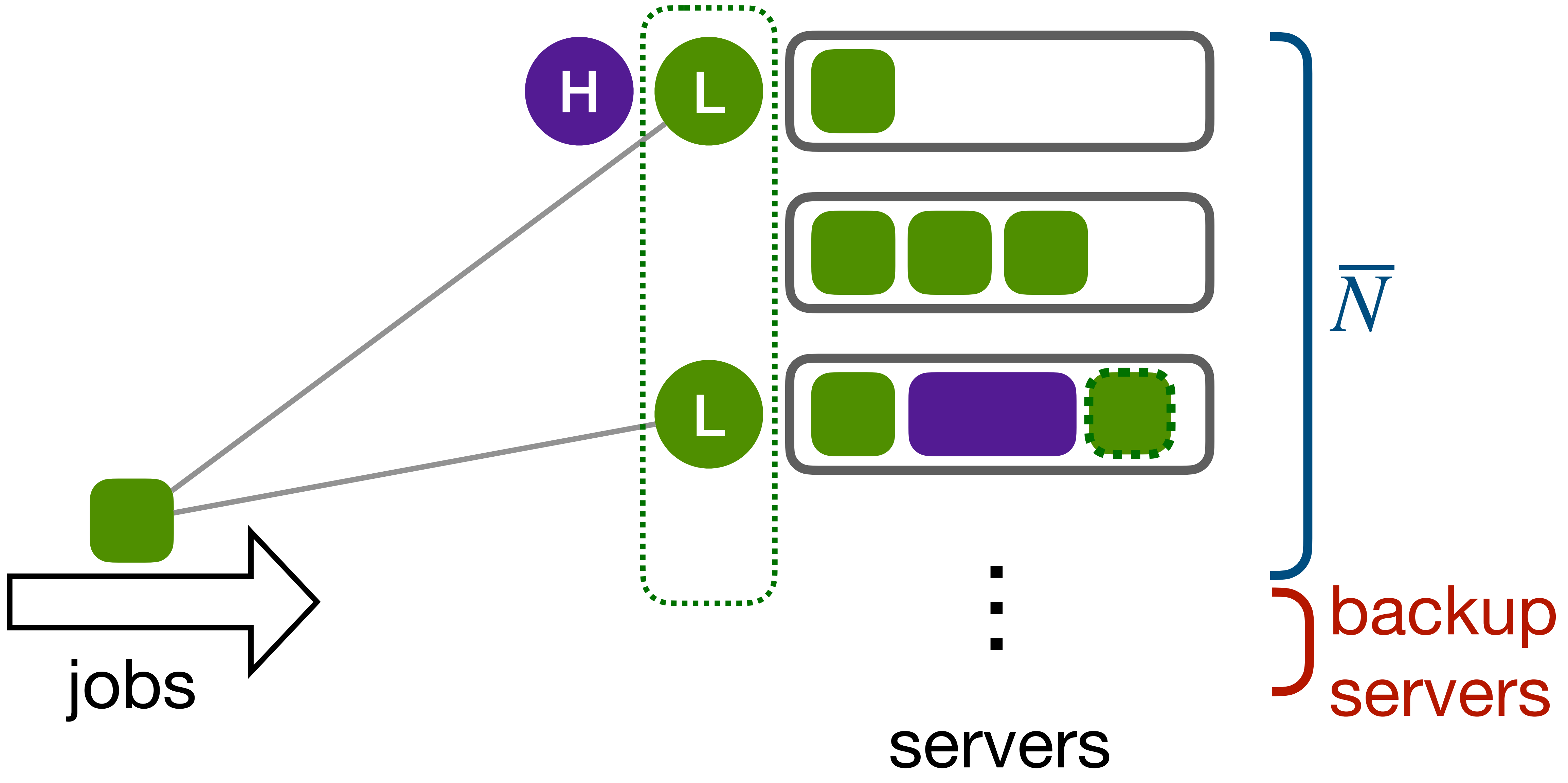
and # backup servers $= O\left(\sqrt{r}\right)$

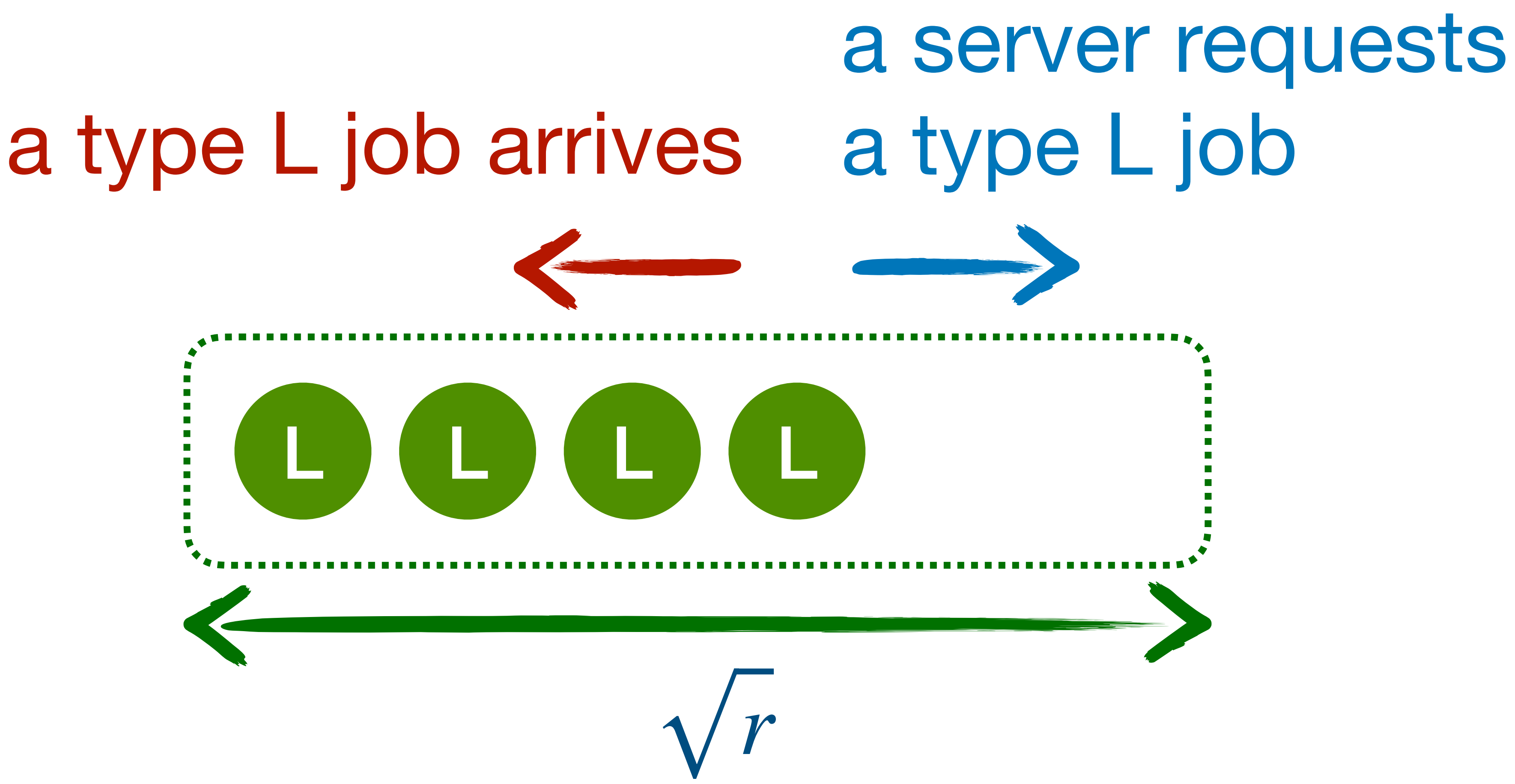


Key proof idea 2

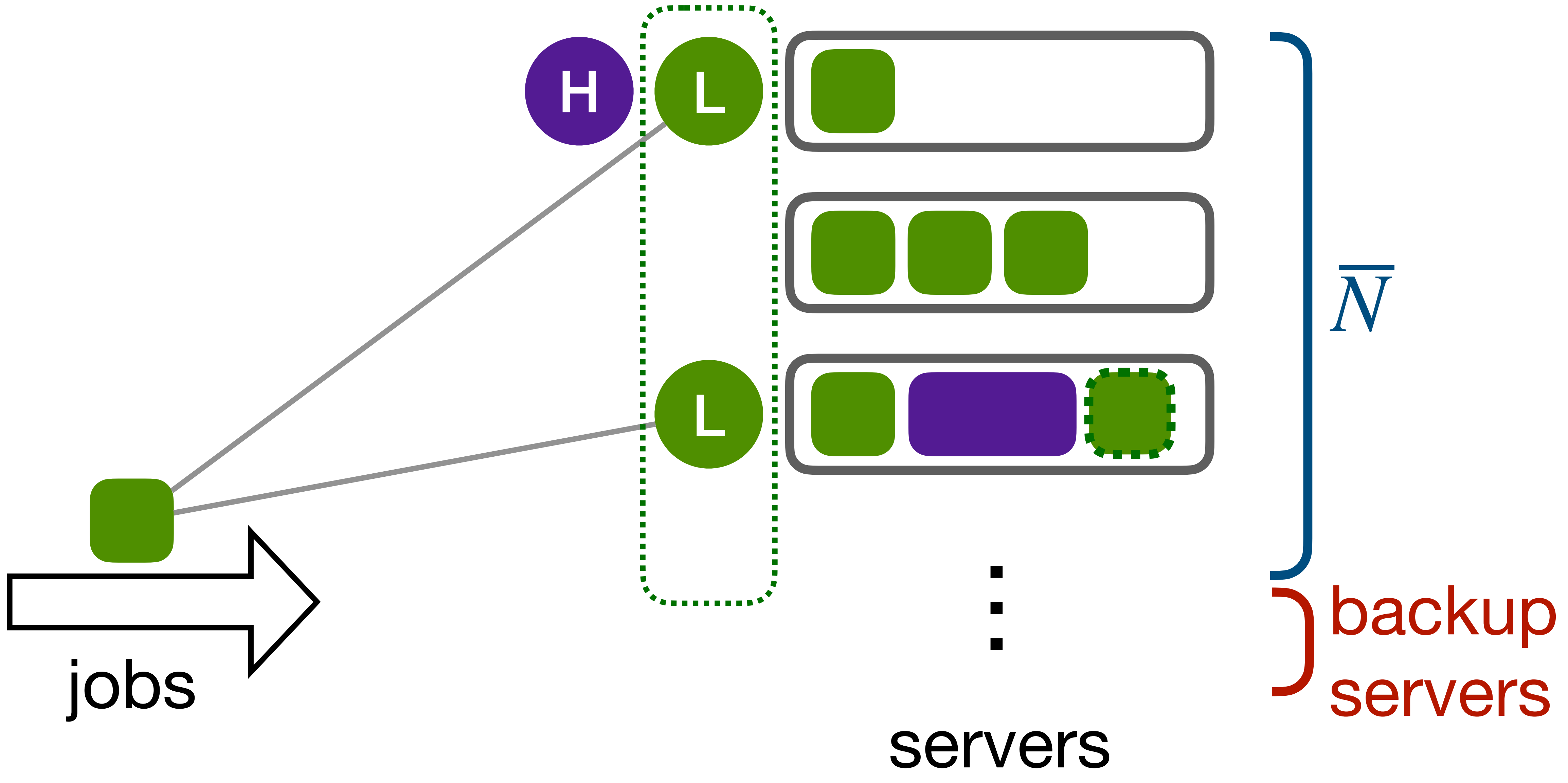
Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

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

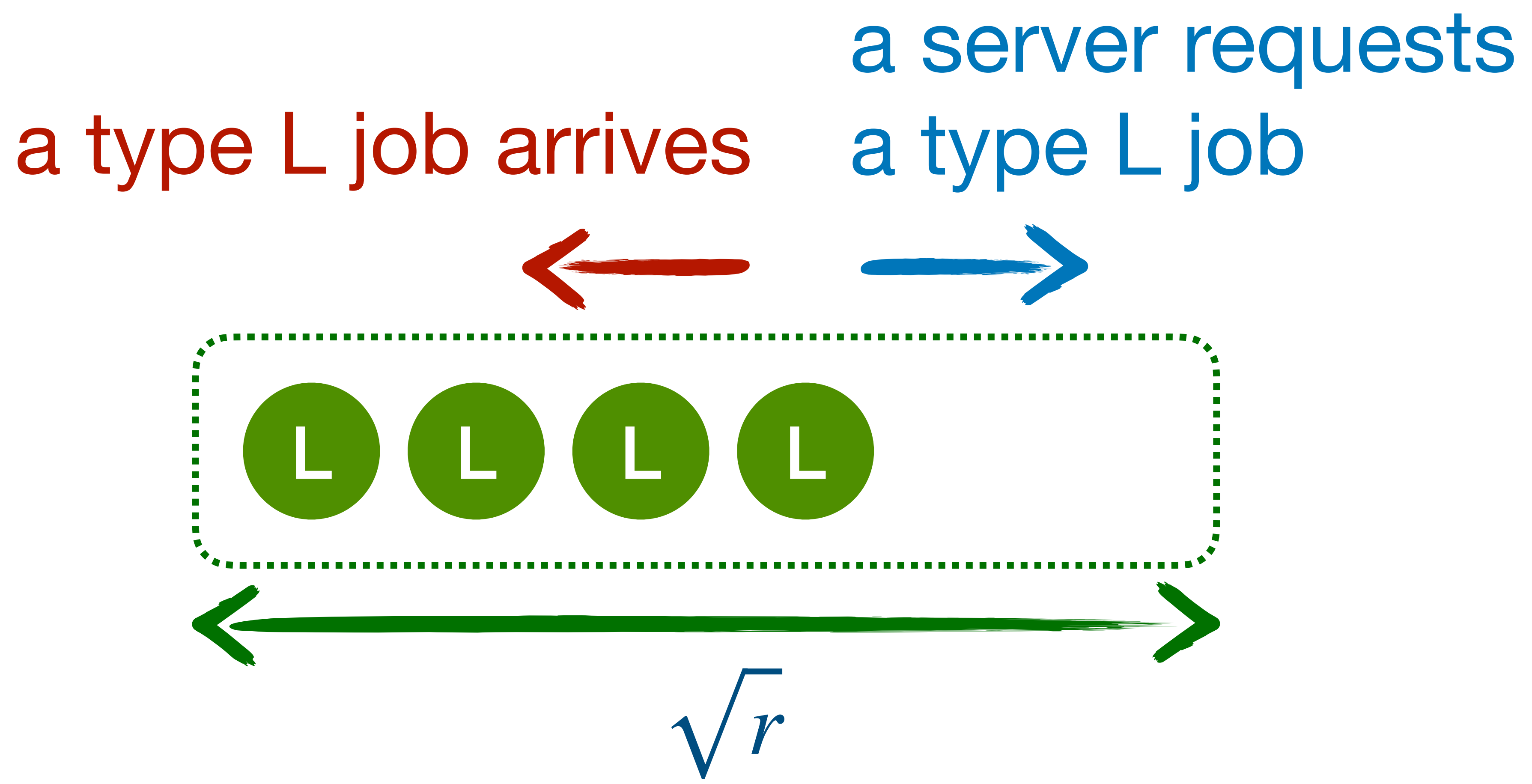




Key proof idea 2

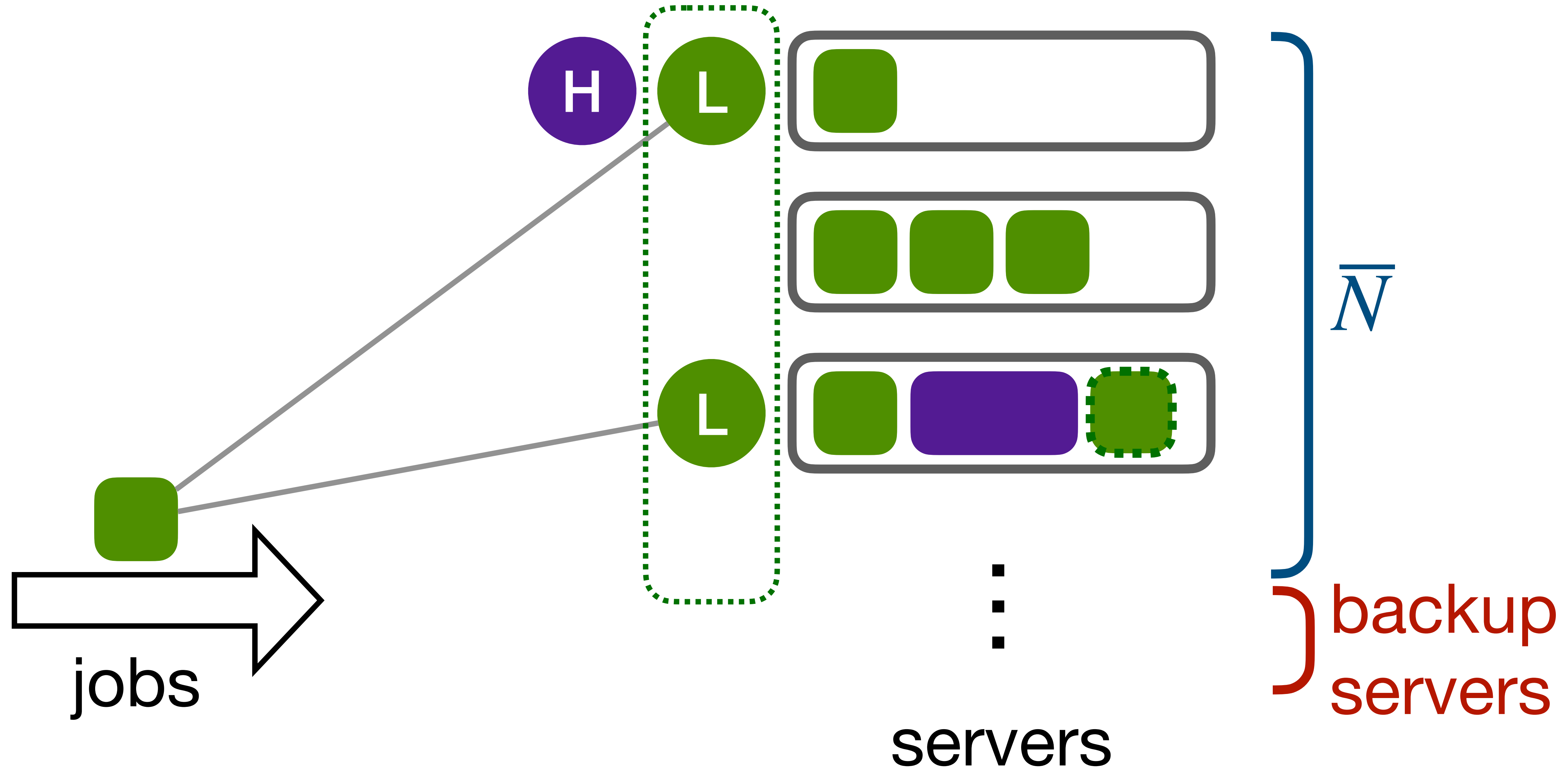


Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,
and # backup servers $= O\left(\sqrt{r}\right)$



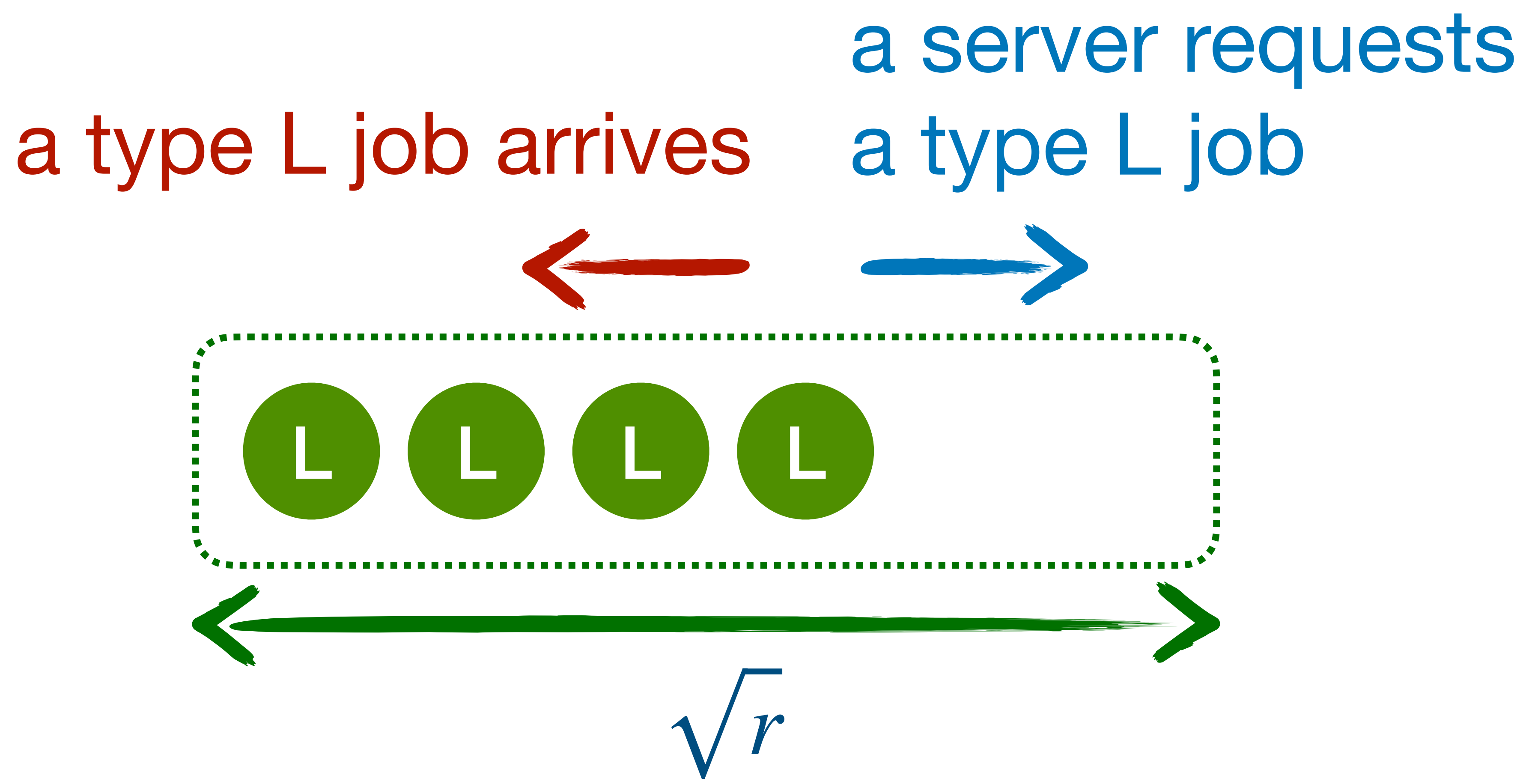
- An almost balanced random walk

Key proof idea 2



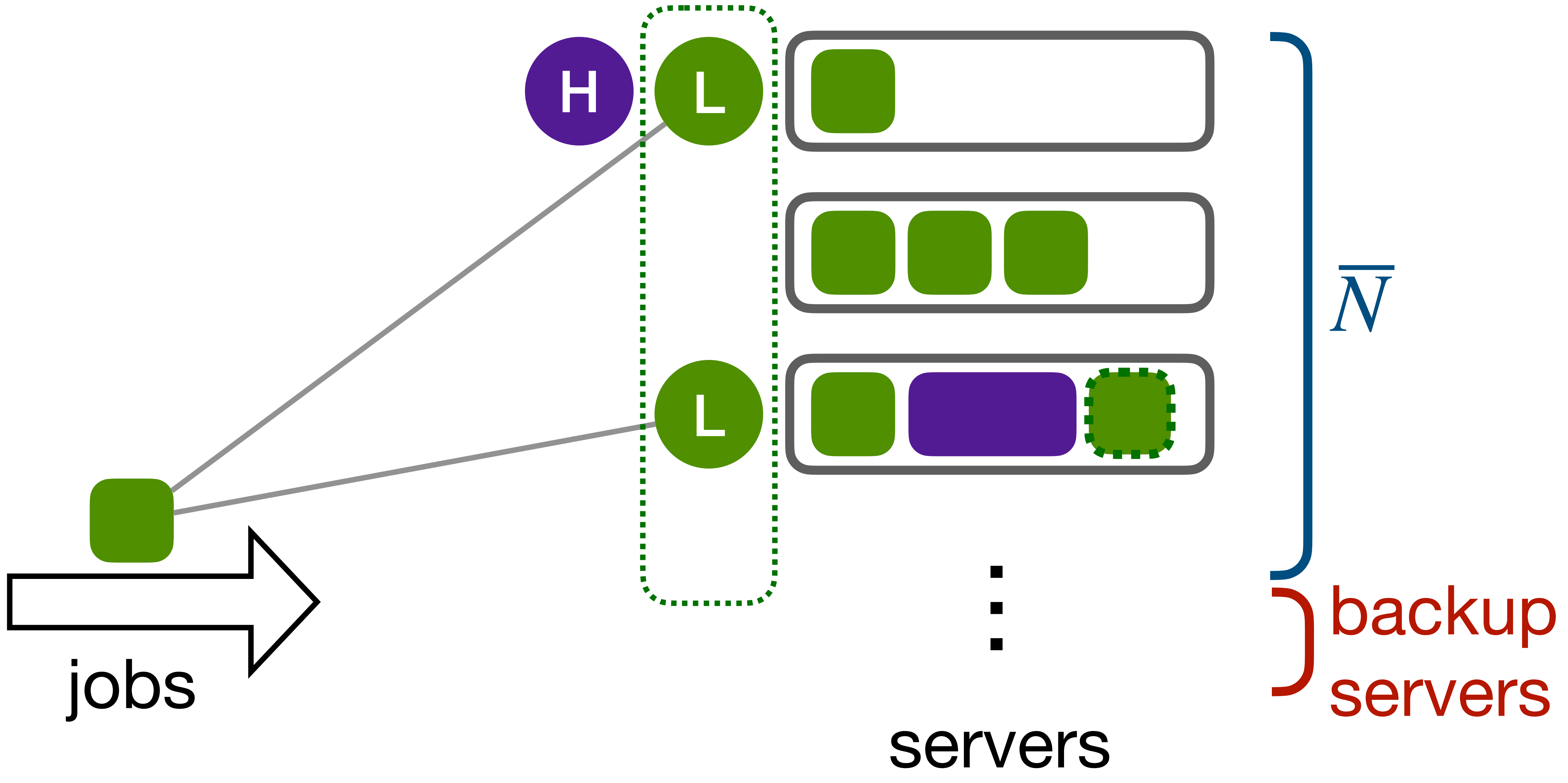
Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

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



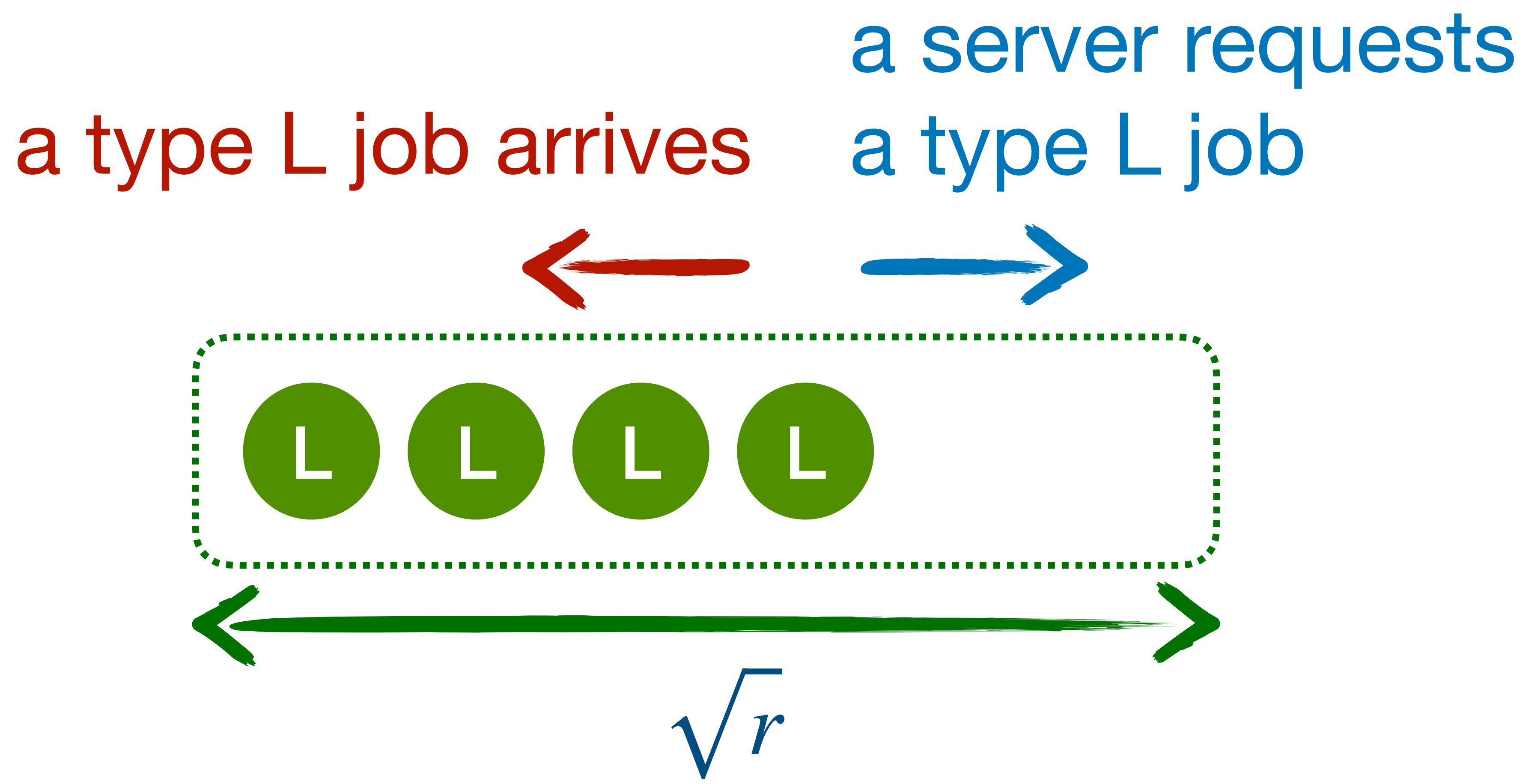
- An almost balanced random walk
- Stationary distribution ≈ uniform on $\{0, 1, \ldots, \sqrt{r}\}$

Key proof idea 2



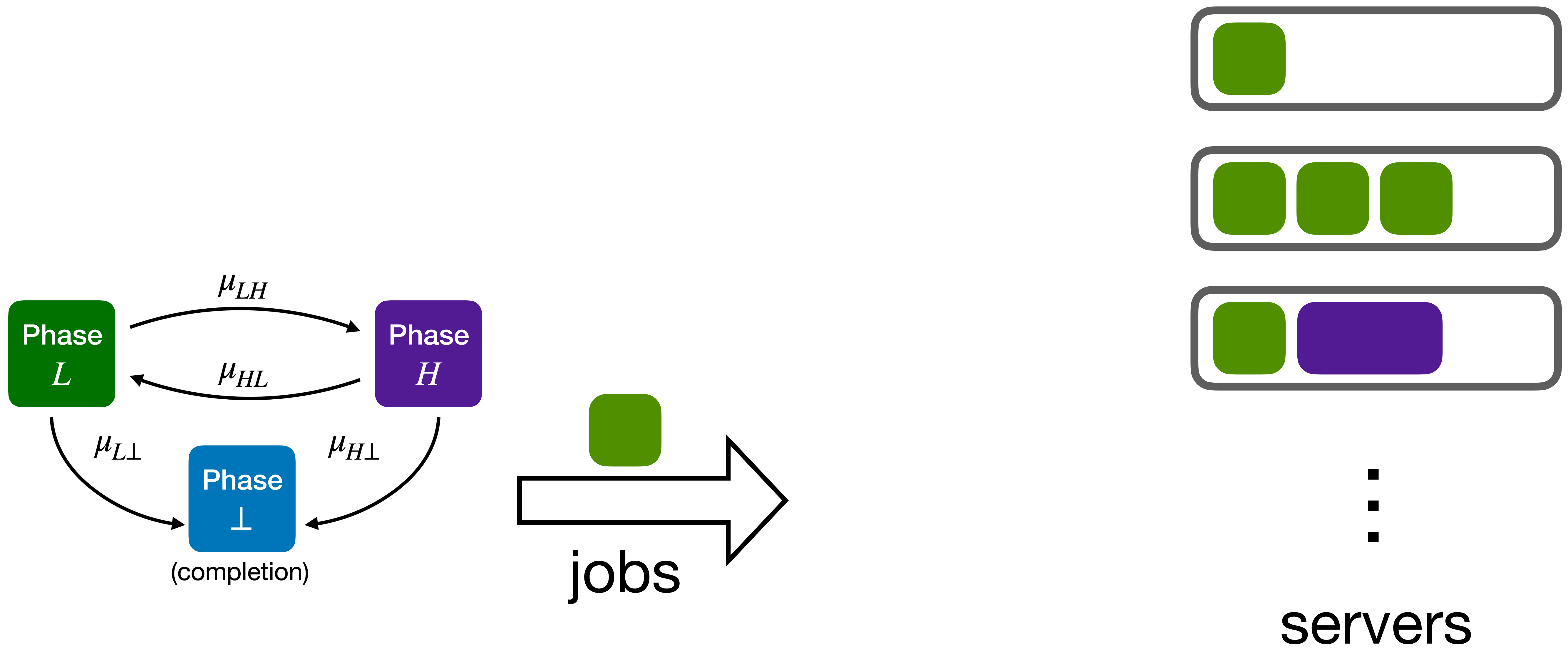
Will show that # virtual jobs $= O\left(\sqrt{r}\right)$,

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



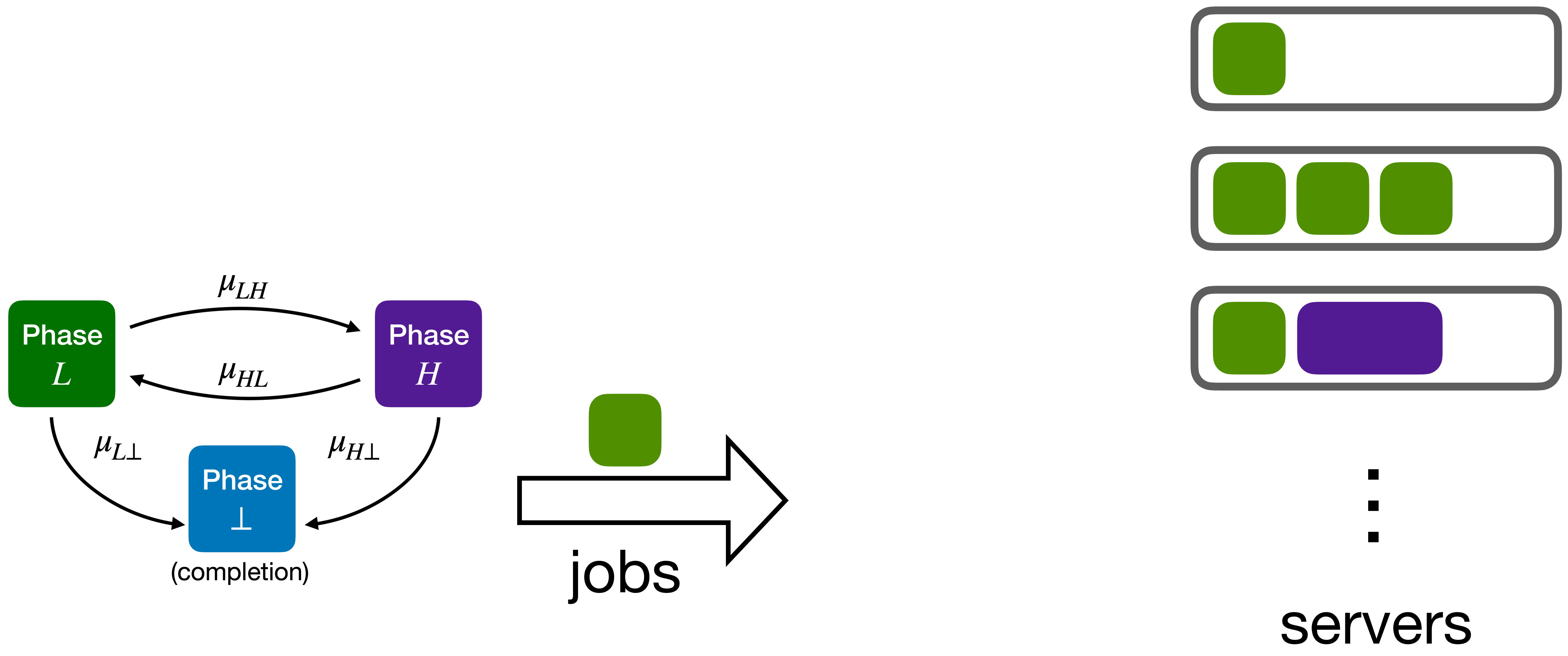
- An almost balanced random walk
- Stationary distribution ≈ uniform on {0, 1, …, $\sqrt{r}$}
- Rate of generating virtual jobs
  ≈ rate of sending jobs to backup servers
  ≈ arrival rate$/\sqrt{r} = O\left(\sqrt{r}\right)$

Summary


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

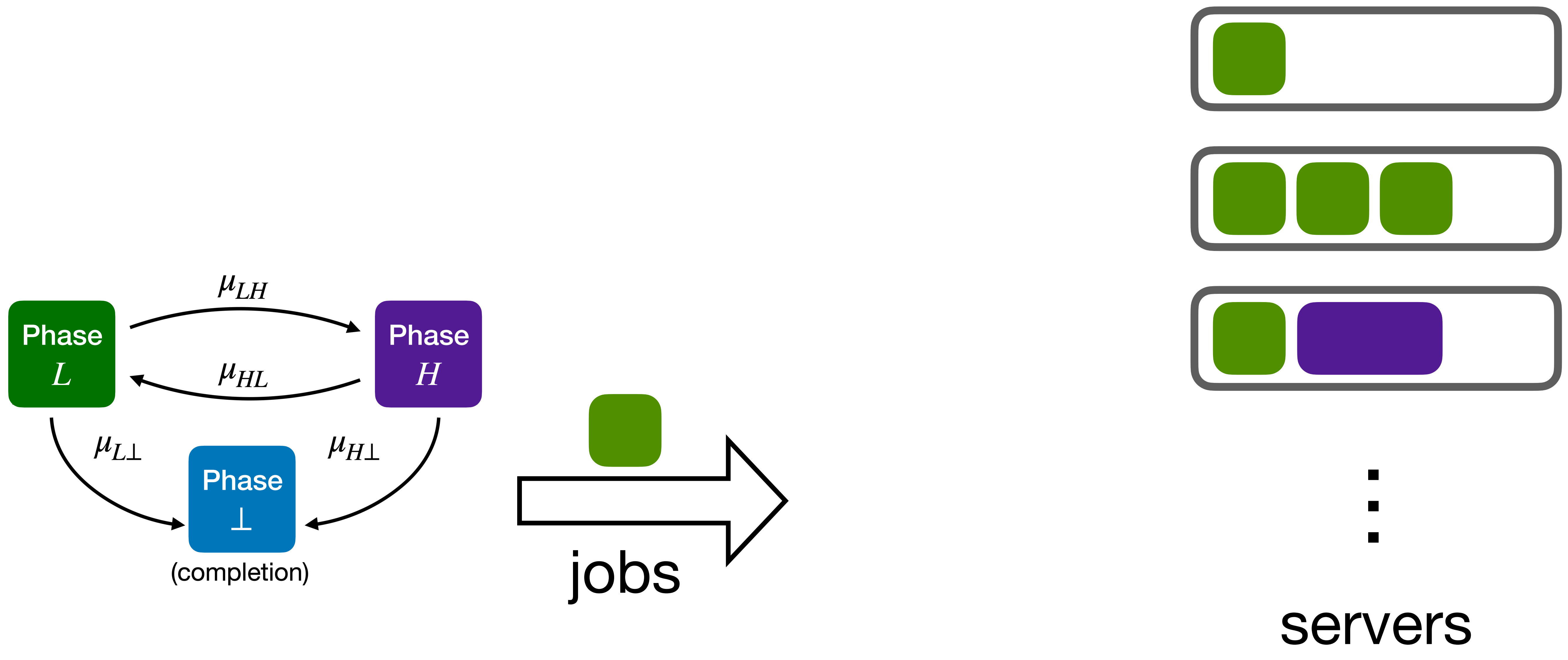
Summary

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



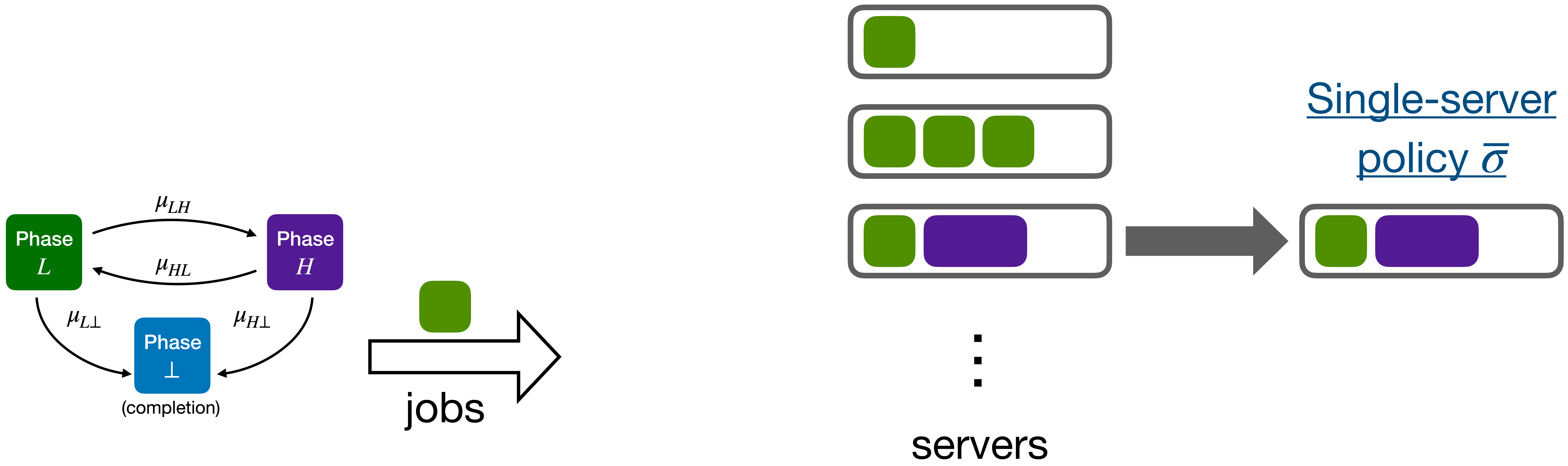
Summary

- 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-REQUESTING-SERVER (JRS), that converts a single-server policy to a policy in the original system