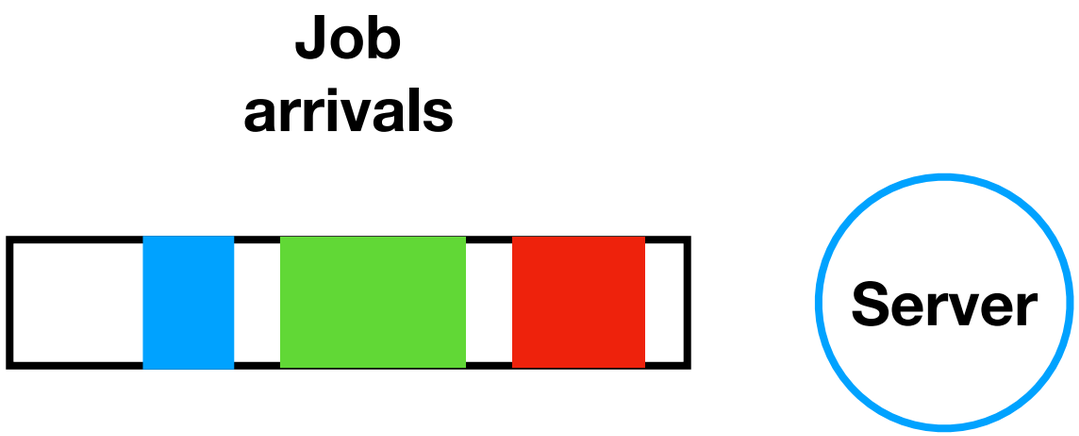
Speed Scaling with Sum-Power Constraint

Rahul Vaze

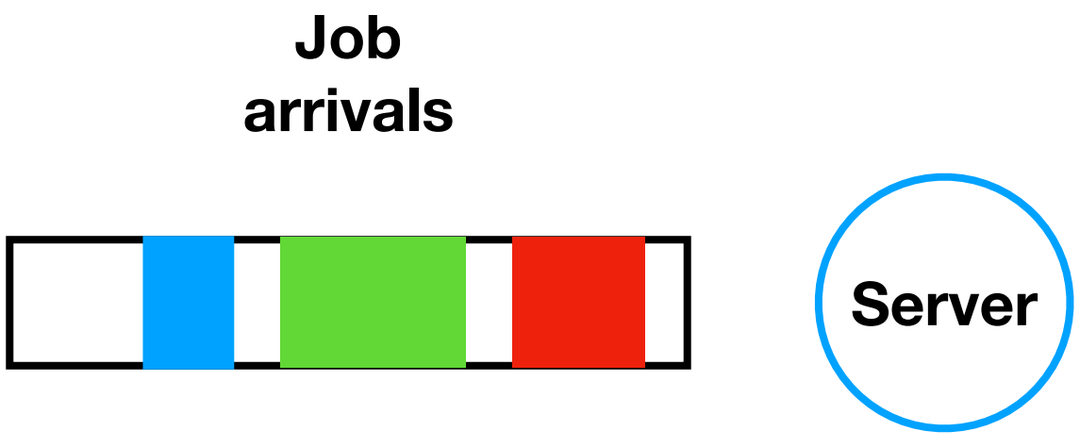
Tata Institute of Fundamental Research, Mumbai

Joint work Jayakrishnan Nair- IIT-Bombay

Scheduling 101

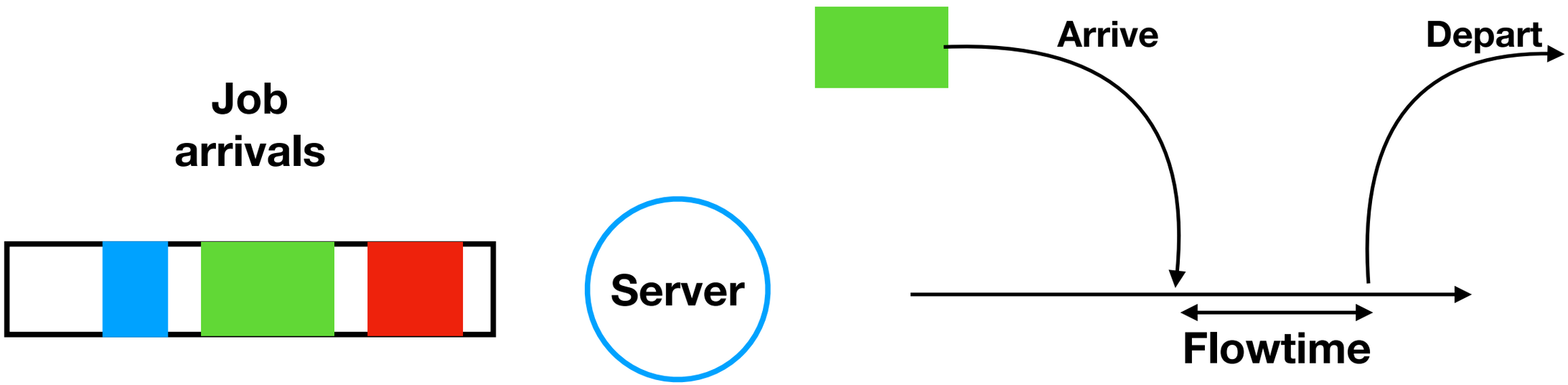


Scheduling 101



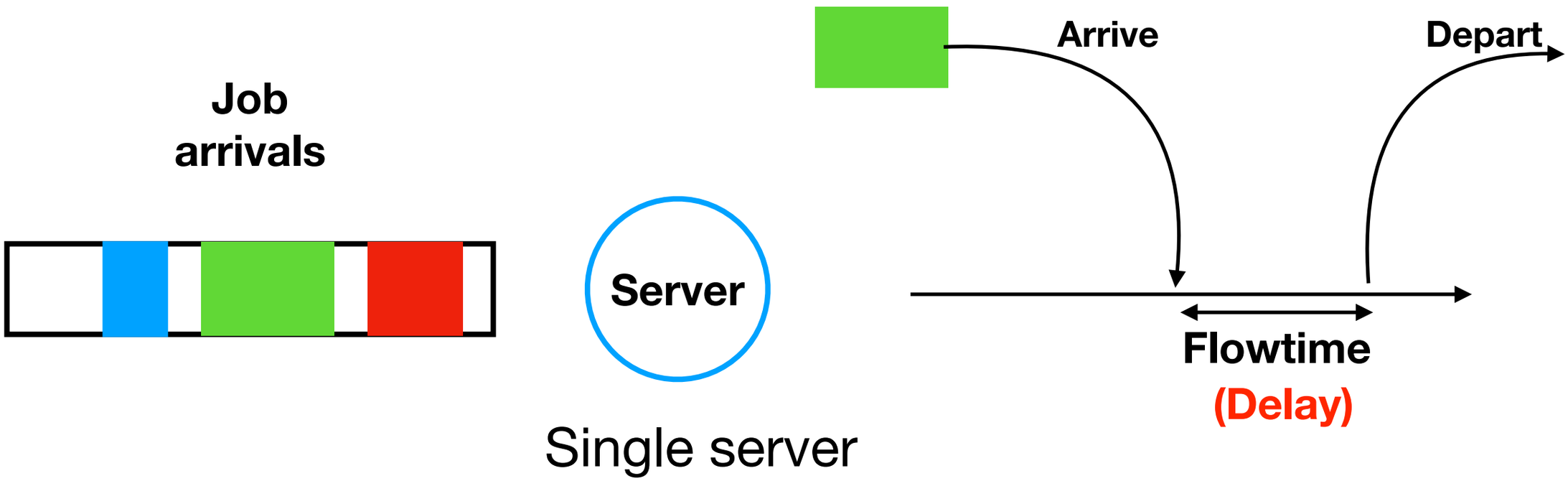
Single server

Scheduling 101



Single server

Scheduling 101

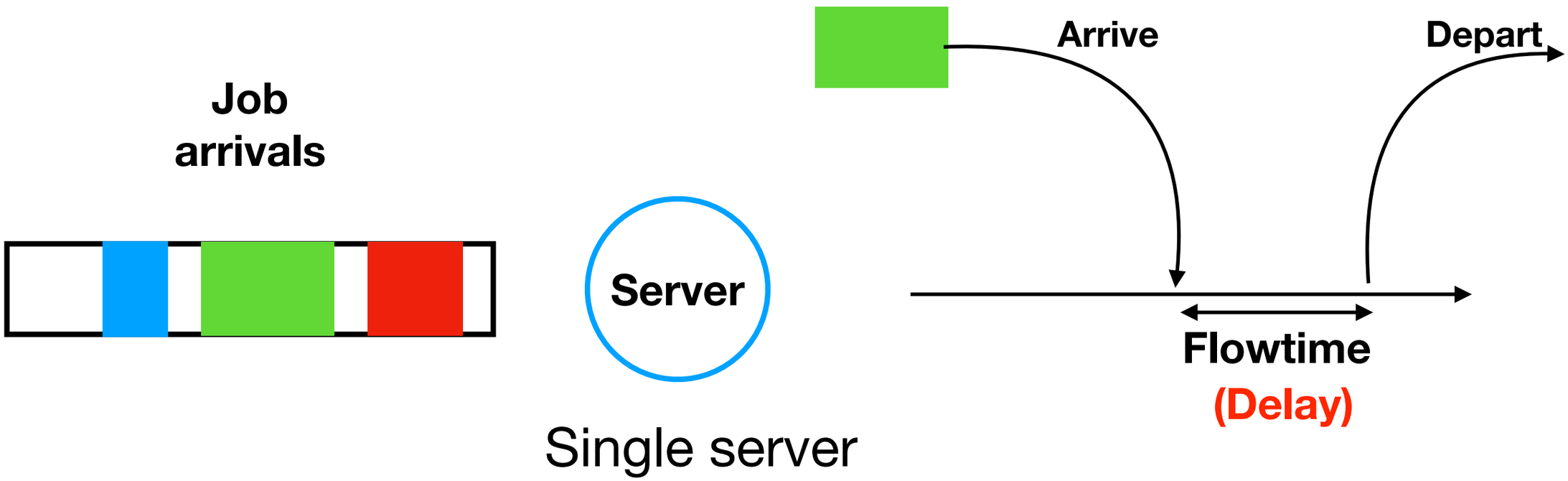


Single server

Obj: *min* **total flow time**

$$\sum_{j \in \text{Jobs}} \text{flow time}_j = \int n(t)dt$$

Scheduling 101



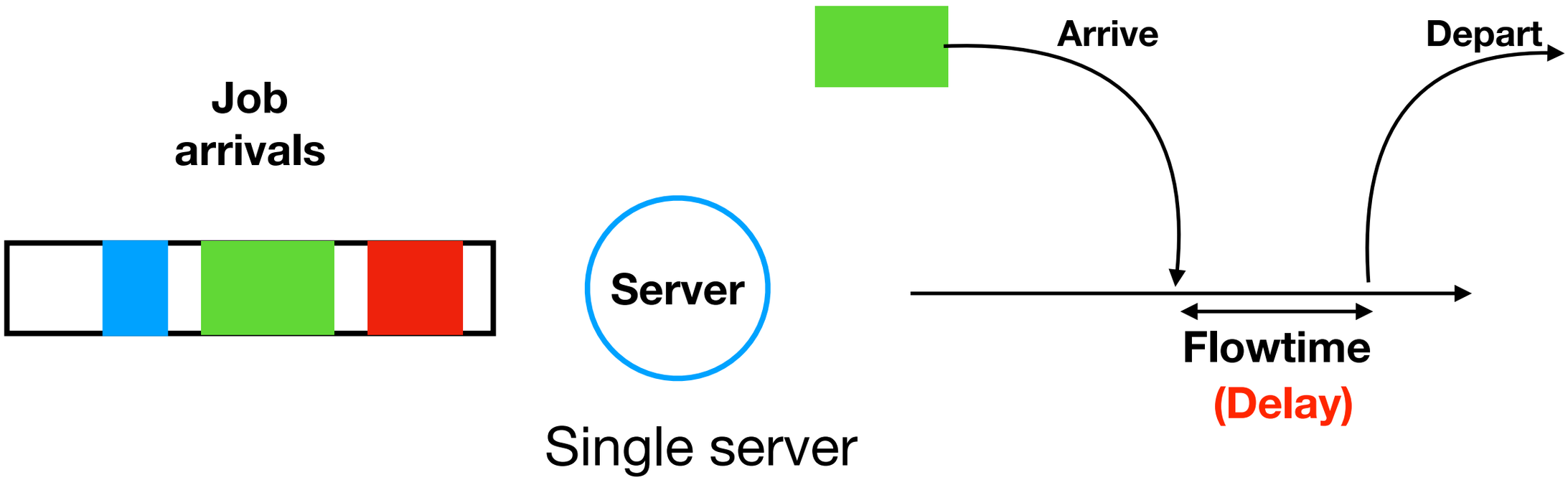
Single server

Obj: *min* **total flow time**

$$\sum_{j \in \text{Jobs}} \text{flow time}_j = \int n(t)dt$$

n(t) **number of outstanding jobs at time t**

Scheduling 101



Single server

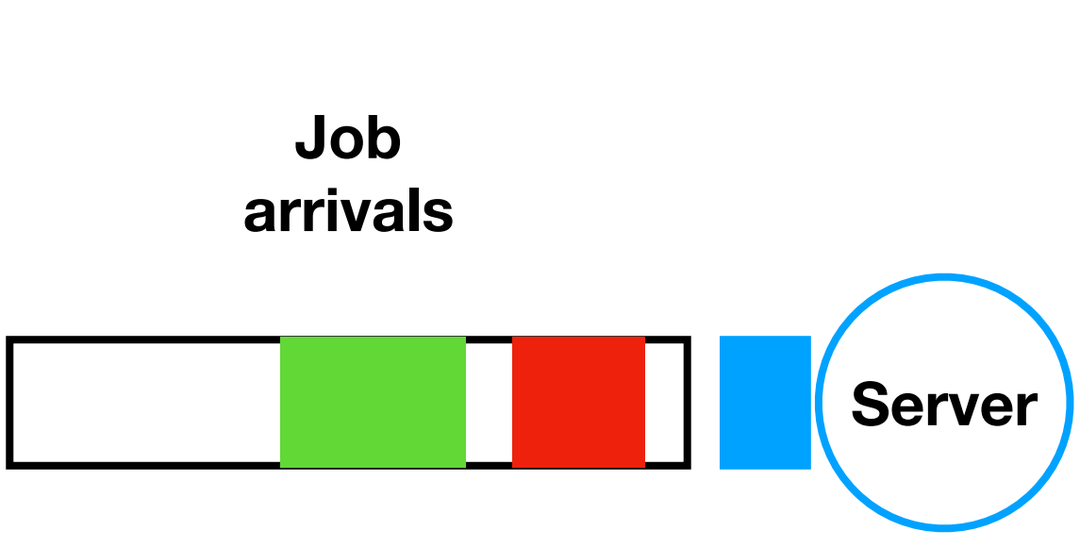
Obj: *min* **total flow time**

$$\sum_{j \in \text{Jobs}} \text{flow time}_j = \int n(t) dt$$

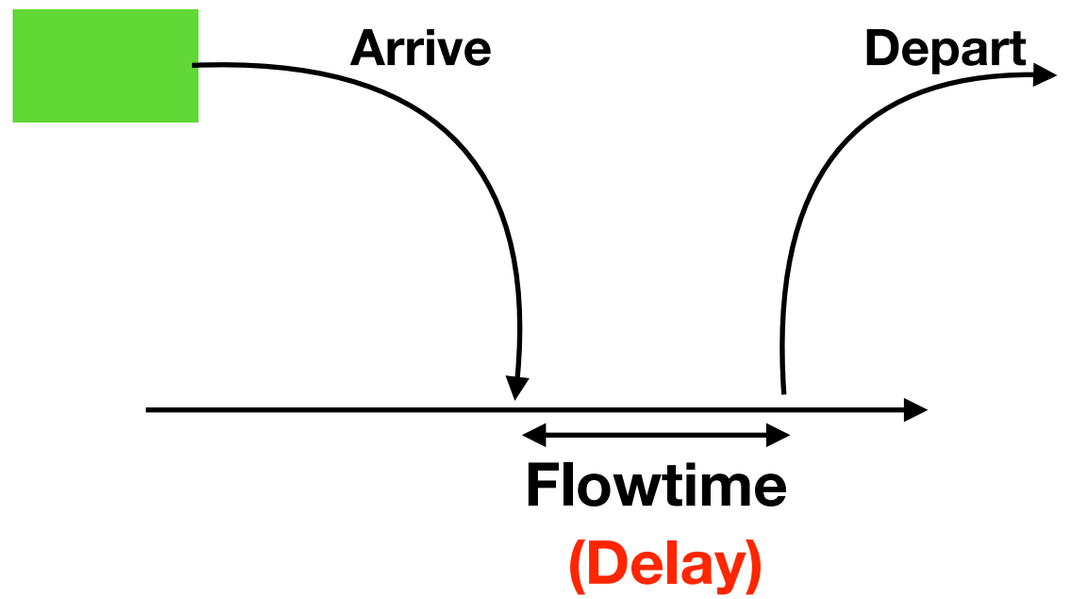
n(t) **number of outstanding jobs at time t**

SRPT is optimal

Scheduling 101



Single server



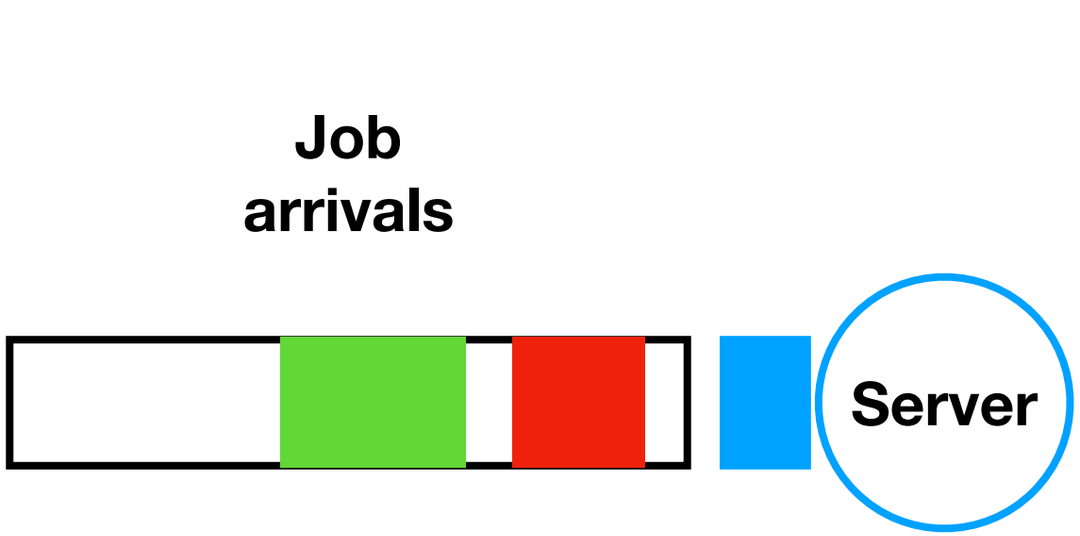
Obj: *min* **total flow time**

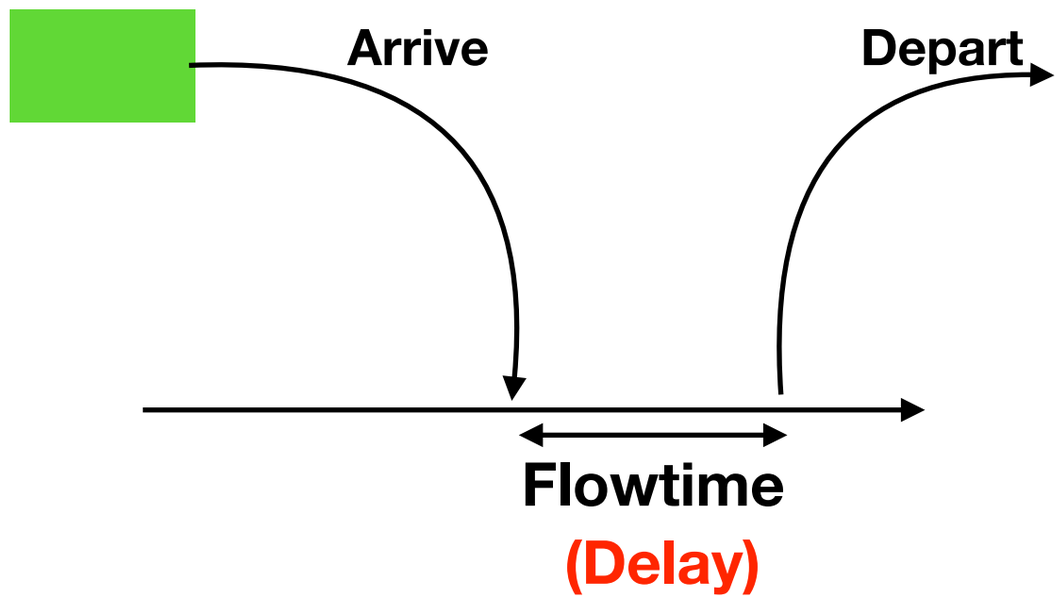
$$\sum_{j \in \text{Jobs}} \text{flow time}_j = \int n(t) dt$$

n(t) **number of outstanding jobs at time t**

SRPT is optimal

Scheduling 101





Single server

Obj: *min* **total flow time**

$$\sum_{j \in \text{Jobs}} \text{flow time}_j = \int n(t) dt$$

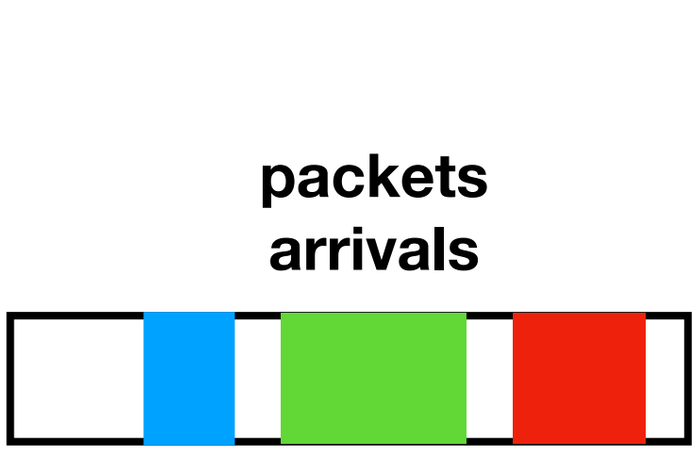
n(t) **number of outstanding jobs at time t**

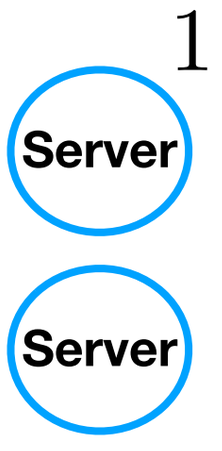
SRPT is optimal

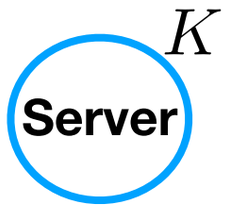
min **# of outstanding jobs**

Parallel Scheduling

Parallel Scheduling







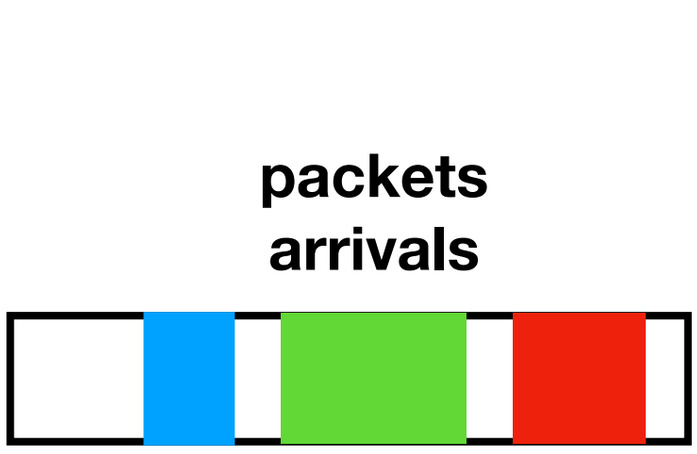
Multiple Servers

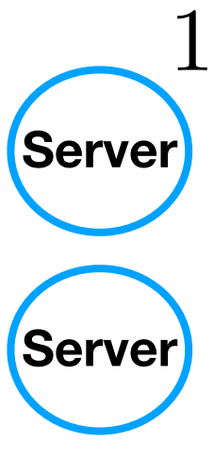
Obj: *min* total flow time

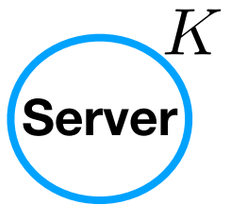
$$\int n(t)dt$$

n(t) **number of outstanding jobs at time t**

Parallel Scheduling







Multiple Servers

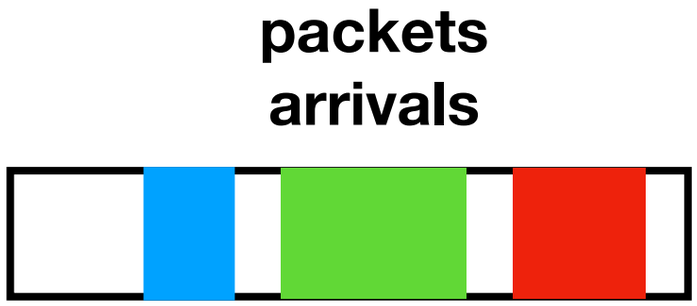
Fully Parallelizable Jobs

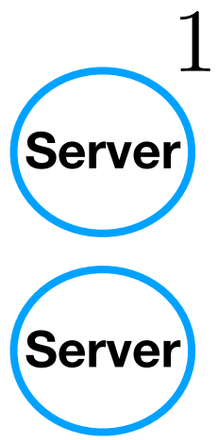
Obj: *min* total flow time

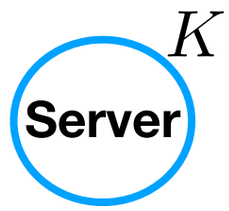
$$\int n(t)dt$$

n(t) **number of outstanding jobs at time t**

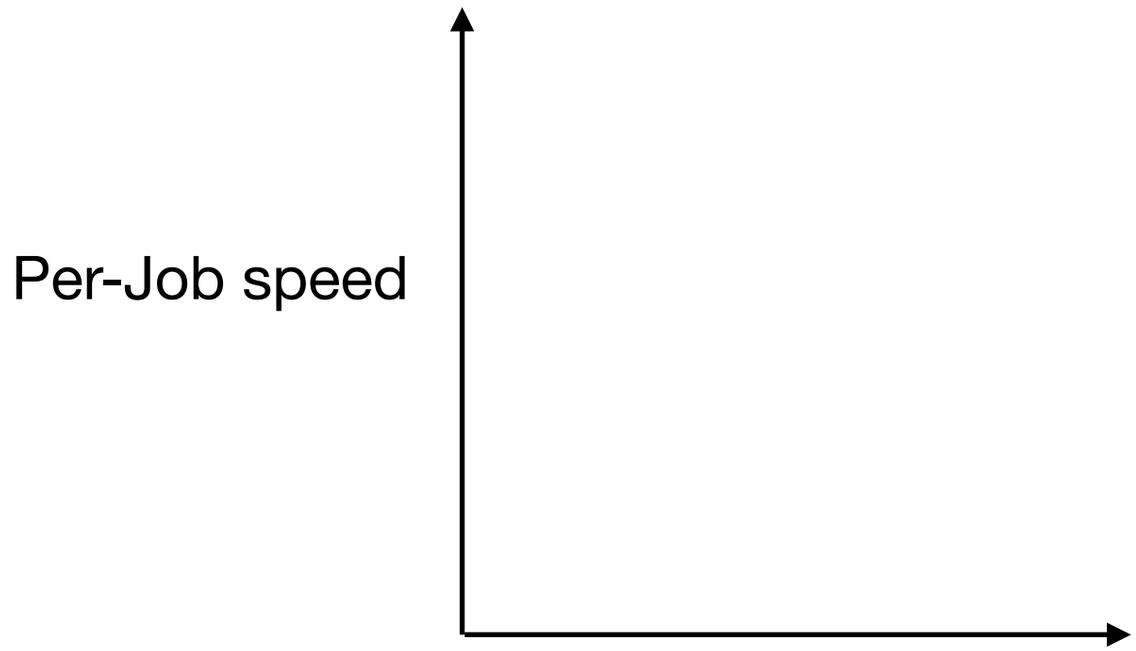
Parallel Scheduling







Multiple Servers



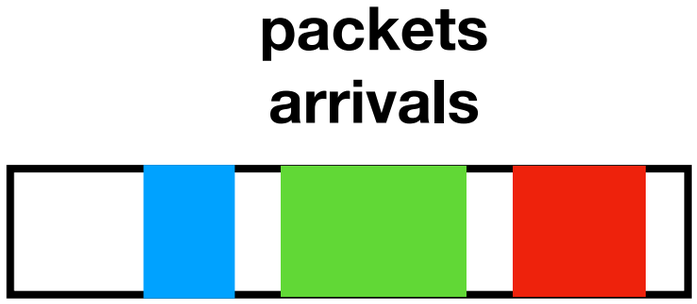
Fully Parallelizable Jobs

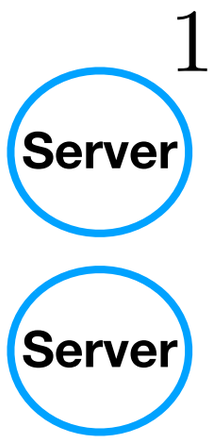
Obj: *min* total flow time

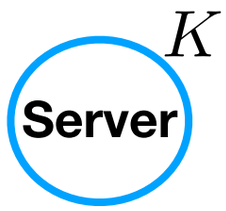
$$\int n(t)dt$$

n(t) **number of outstanding jobs at time t**

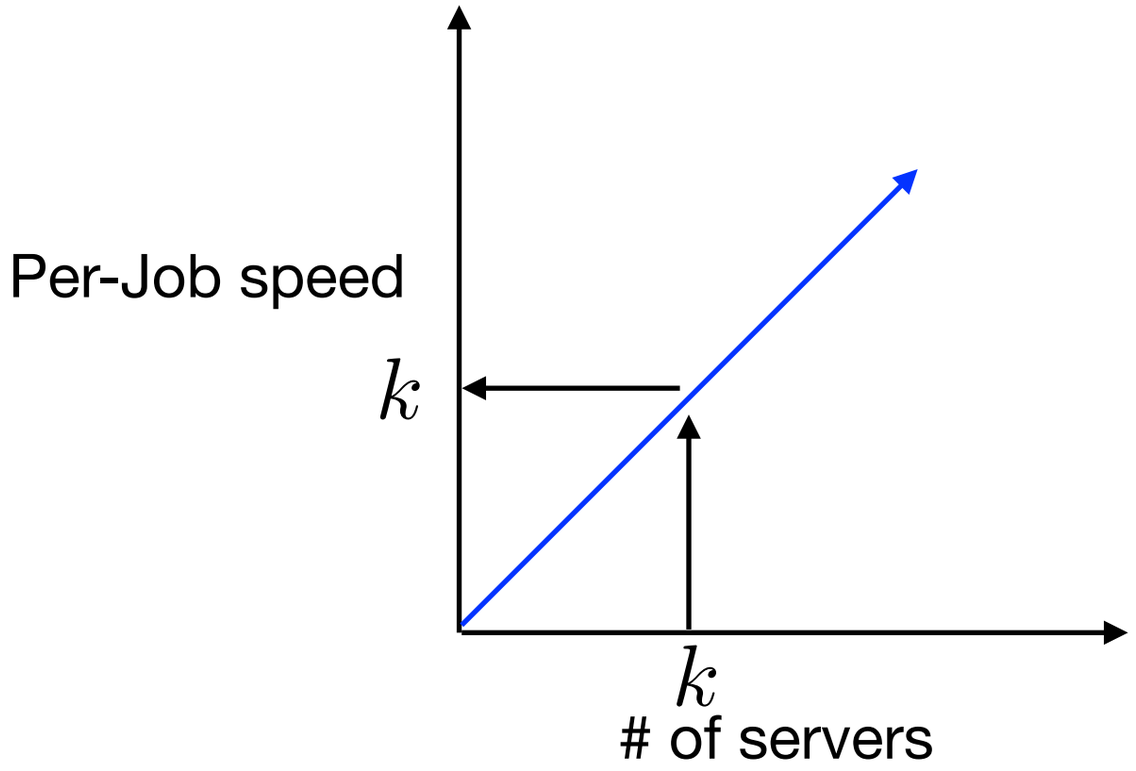
Parallel Scheduling







Multiple Servers



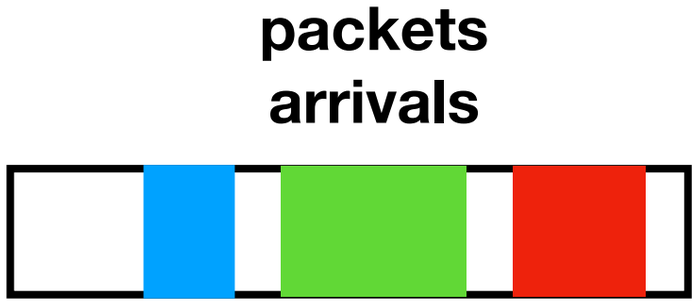
Fully Parallelizable Jobs

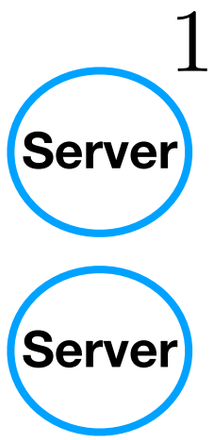
Obj: *min* total flow time

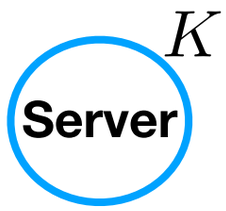
$$\int n(t)dt$$

n(t) **number of outstanding jobs at time t**

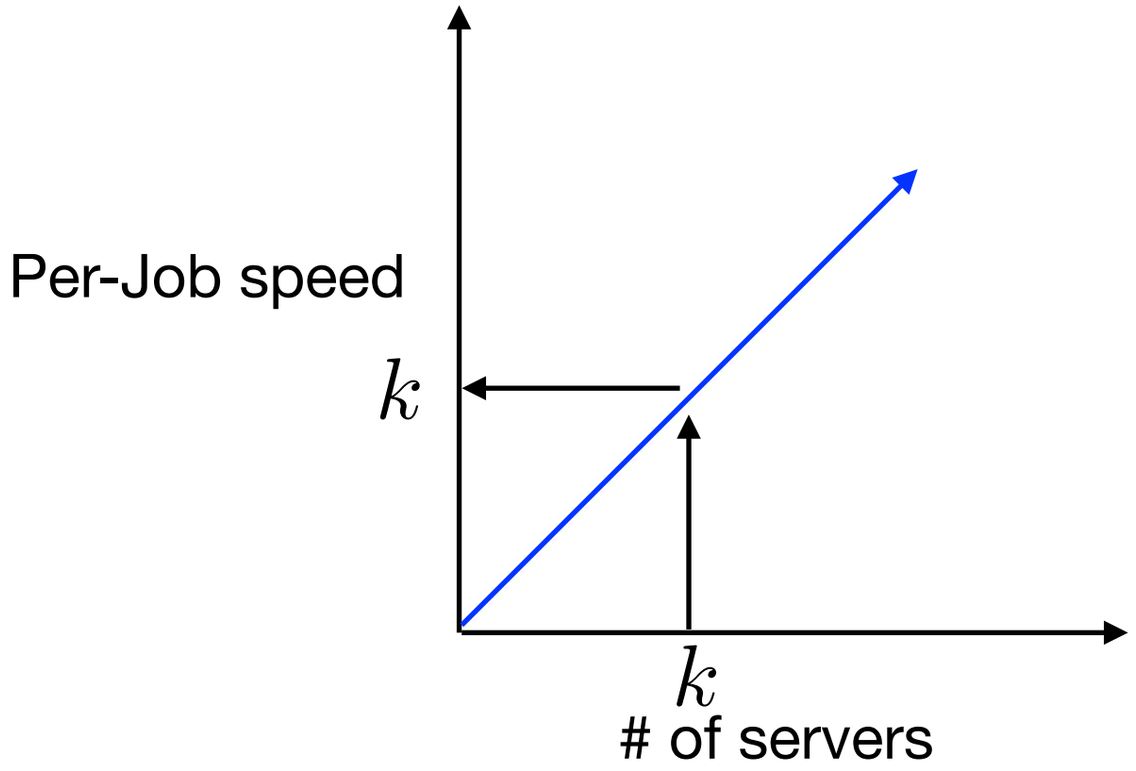
Parallel Scheduling







Multiple Servers



Fully Parallelizable Jobs

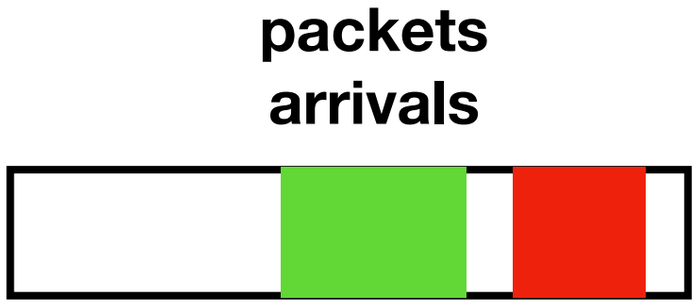
Obj: *min* total flow time

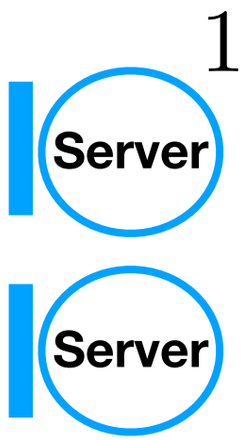
$$\int n(t)dt$$

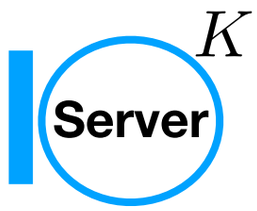
n(t) **number of outstanding jobs at time t**

SRPT is optimal

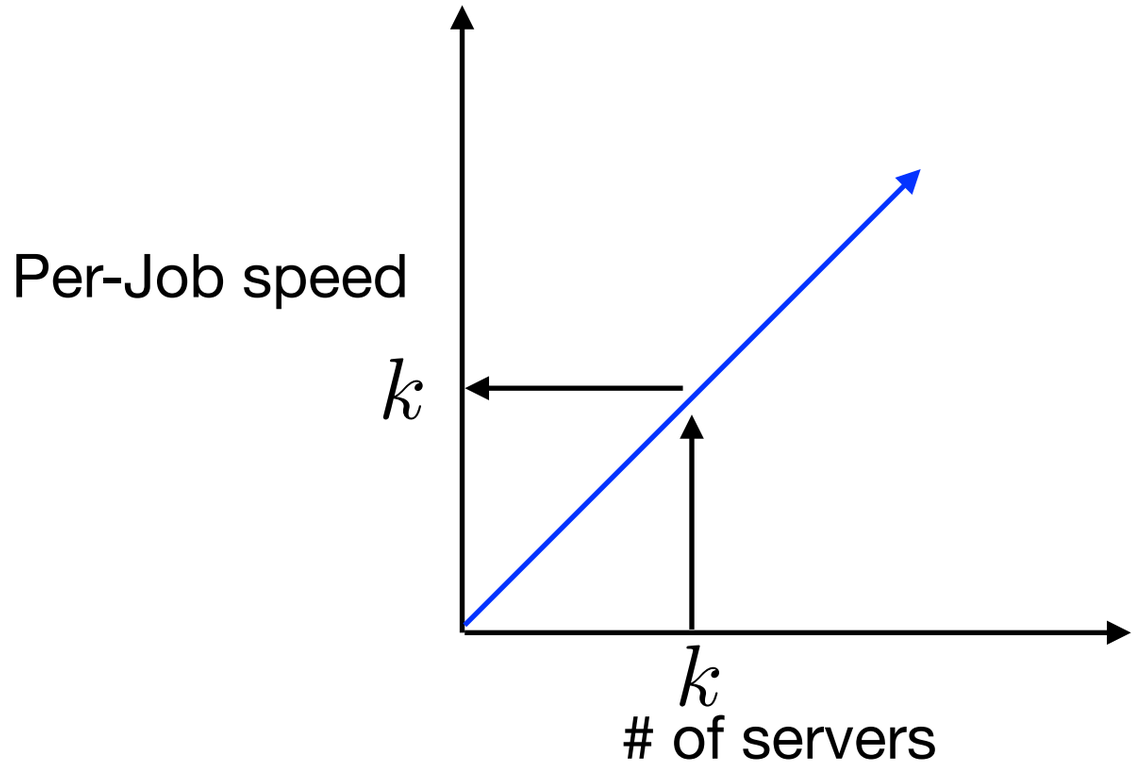
Parallel Scheduling







Multiple Servers



Fully Parallelizable Jobs

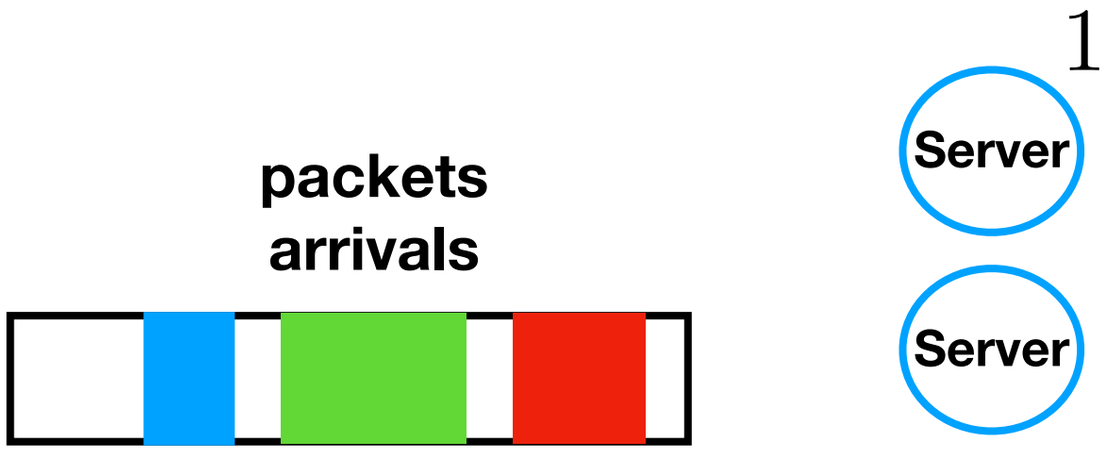
Obj: *min* total flow time

$$\int n(t)dt$$

***n(t)* number of outstanding jobs at time t**

SRPT is optimal

Parallel Scheduling- Reality



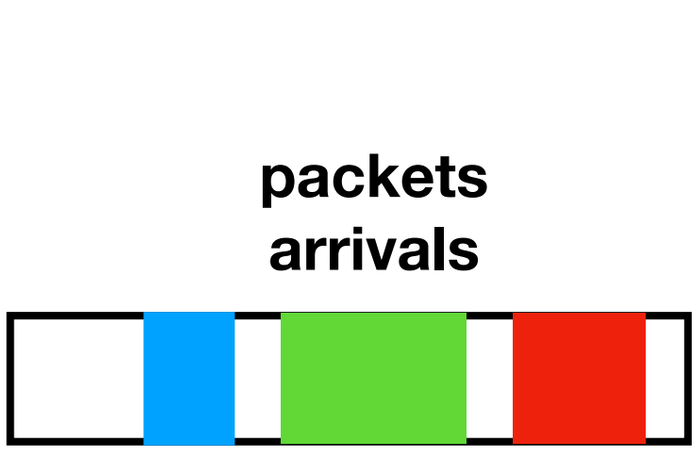
Multiple Servers

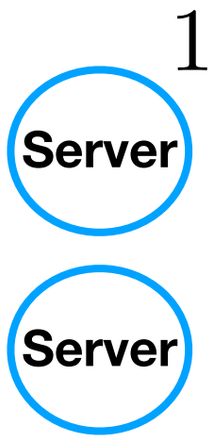
Obj: ***min*** **total flow time**

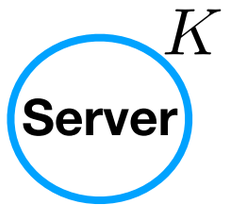
$$\int n(t)dt$$

n(t) **number of outstanding jobs at time t**

Parallel Scheduling- Reality







Multiple Servers

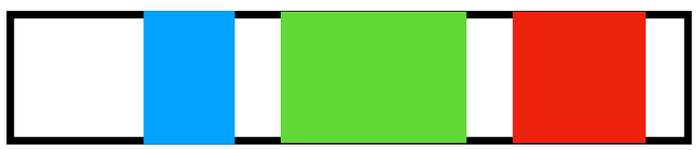
Limited Parallelizable Jobs

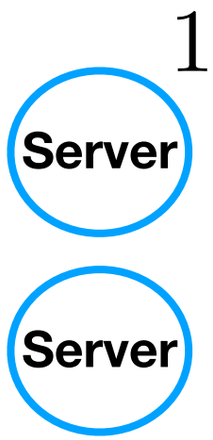
Obj: *min* total flow time

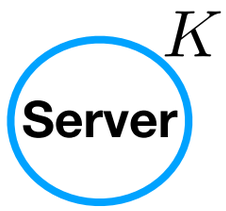
$$\int n(t)dt$$

n(t) **number of outstanding jobs at time t**

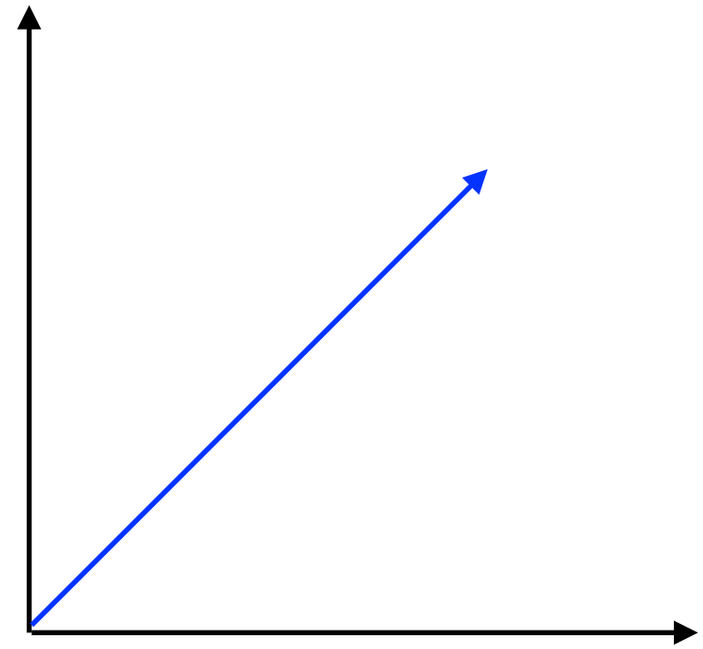
Parallel Scheduling- Reality







Multiple Servers



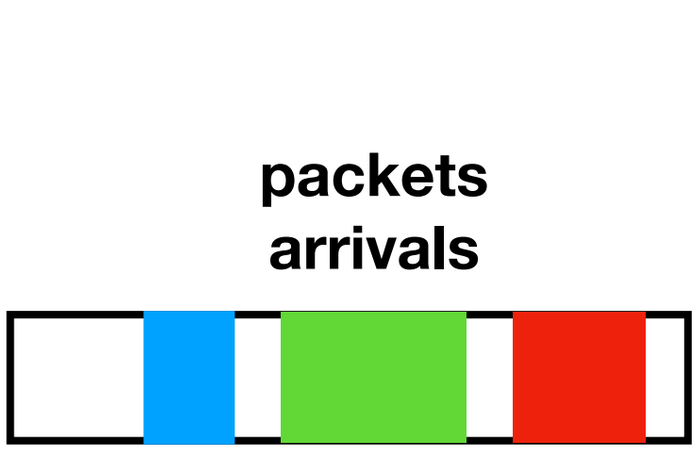
Limited
Parallelizable Jobs

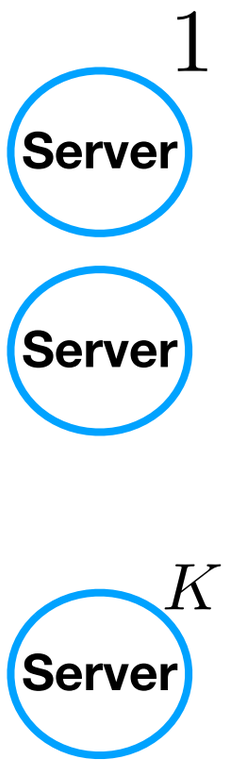
Obj: *min* total flow time

$$\int n(t)dt$$

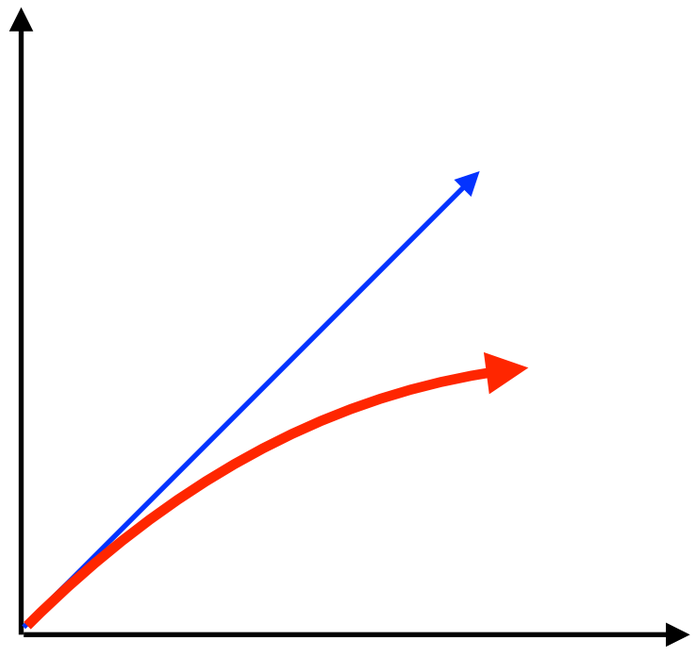
n(t) **number of outstanding jobs at time t**

Parallel Scheduling- Reality





Multiple Servers



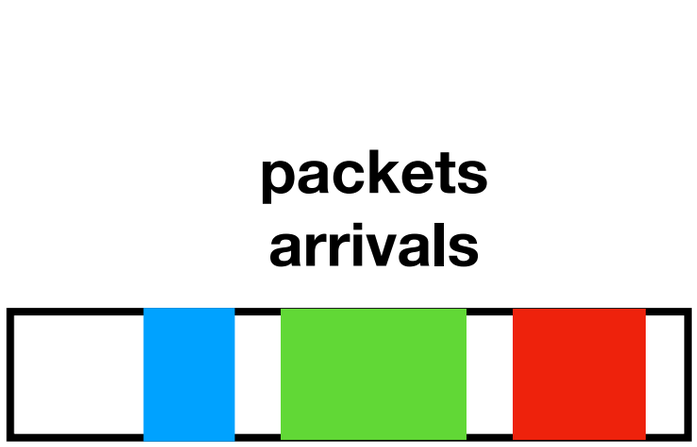
Limited Parallelizable Jobs

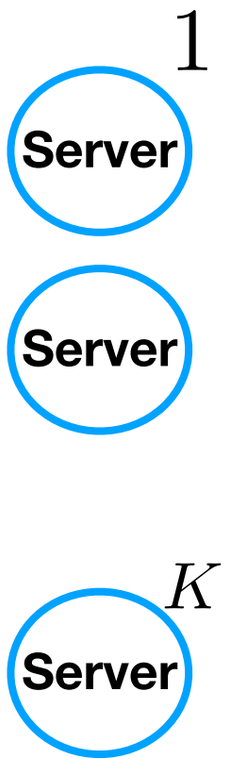
Obj: *min* **total flow time**

$$\int n(t)dt$$

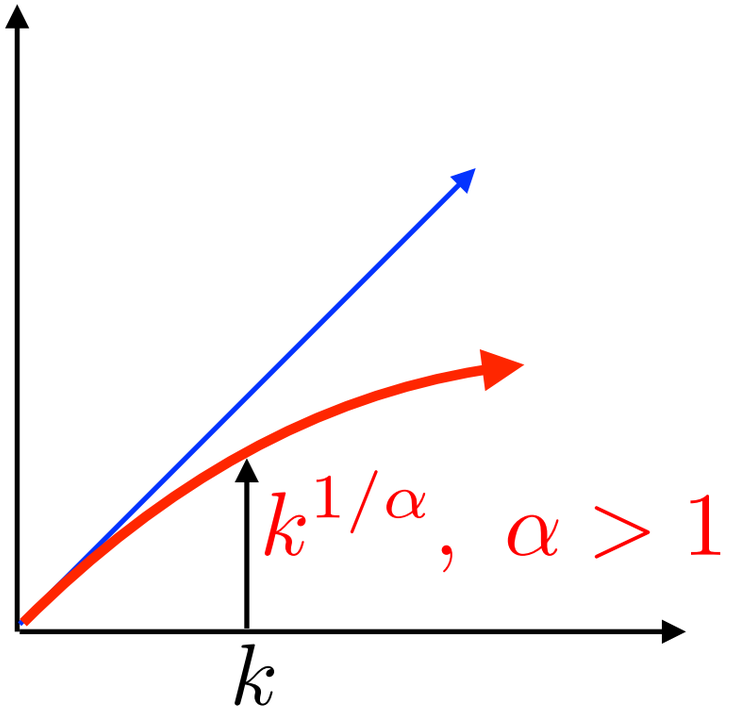
n(t) **number of outstanding jobs at time t**

Parallel Scheduling- Reality





Multiple Servers



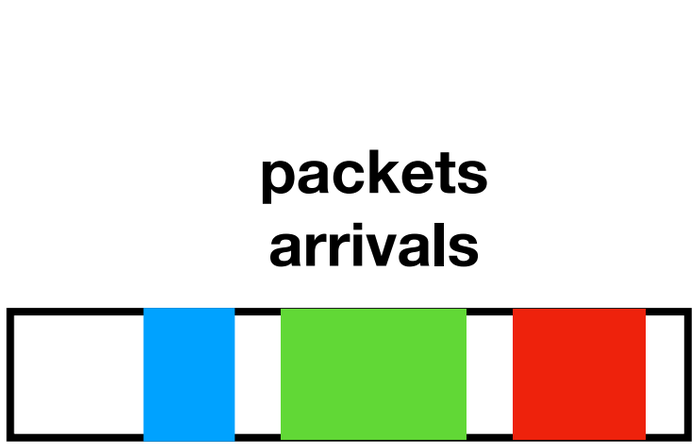
Limited Parallelizable Jobs

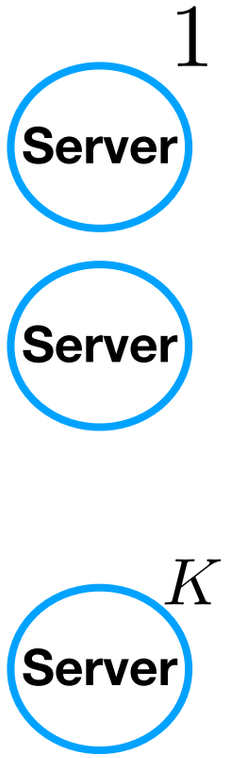
Obj: *min* total flow time

$$\int n(t)dt$$

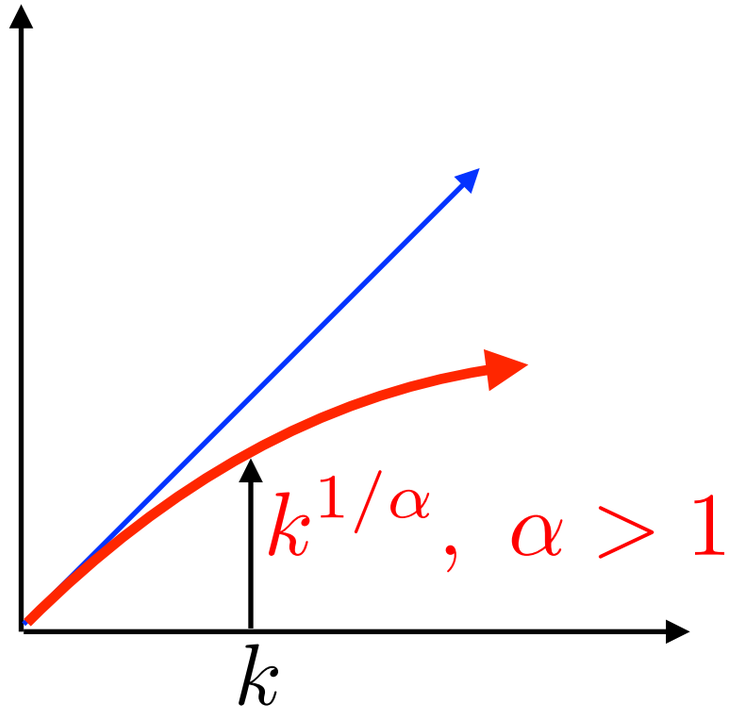
n(t) **number of outstanding jobs at time t**

Parallel Scheduling- Reality





Multiple Servers



Limited Parallelizable Jobs

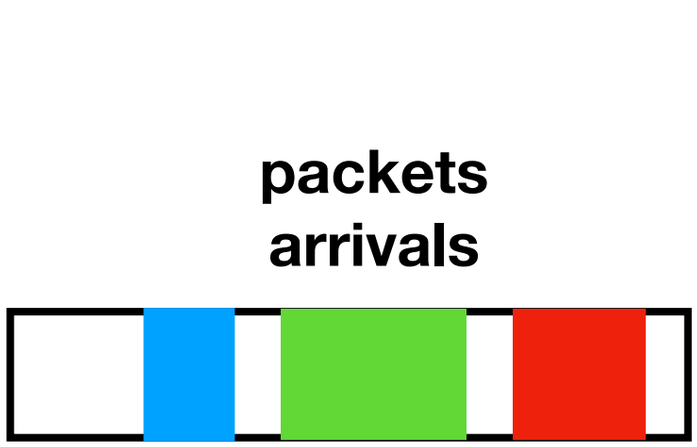
Diminishing returns by using more servers for the same job

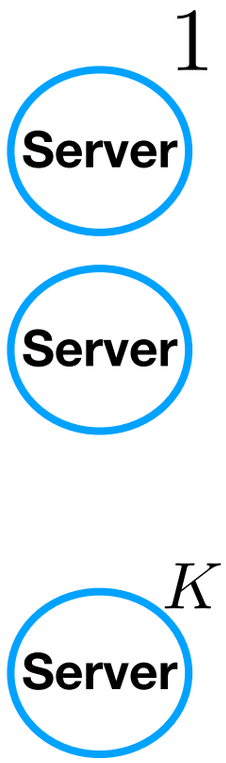
Obj: *min* total flow time

$$\int n(t)dt$$

n(t) **number of outstanding jobs at time t**

Parallel Scheduling- Reality



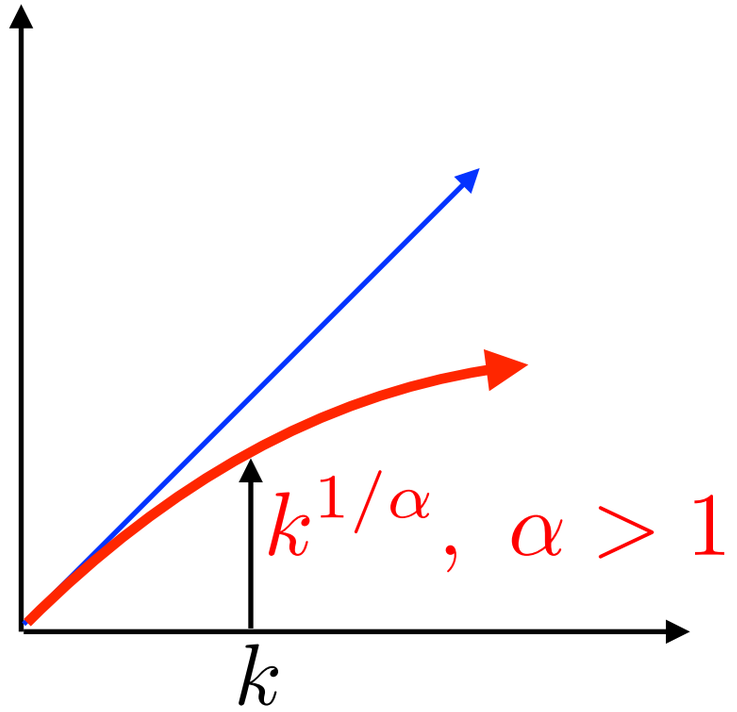


Multiple Servers

Obj: ***min*** **total flow time**

$$\int n(t)dt$$

n(t) **number of outstanding jobs at time t**

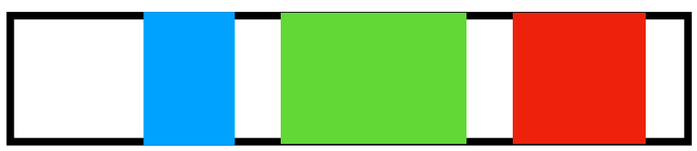


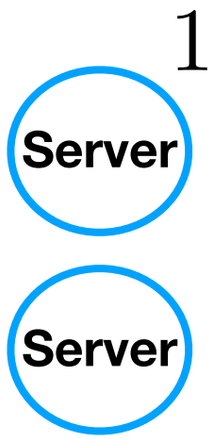
Limited
Parallelizable Jobs

Diminishing returns by using more servers for the same job

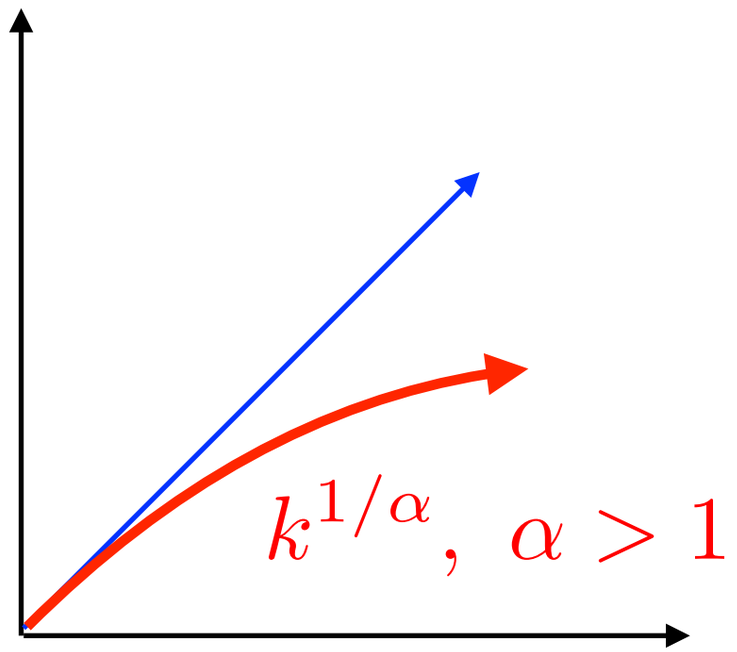
Open Question:
Optimal Scheduling

Prior Work



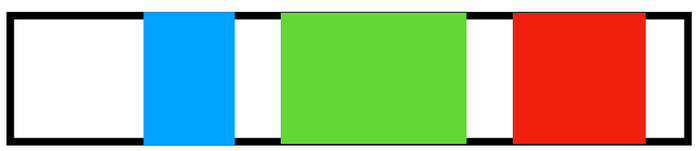


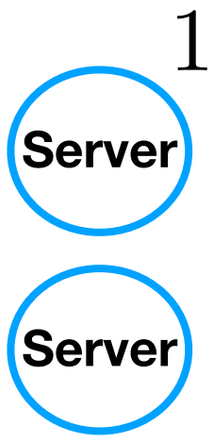
Multiple Servers



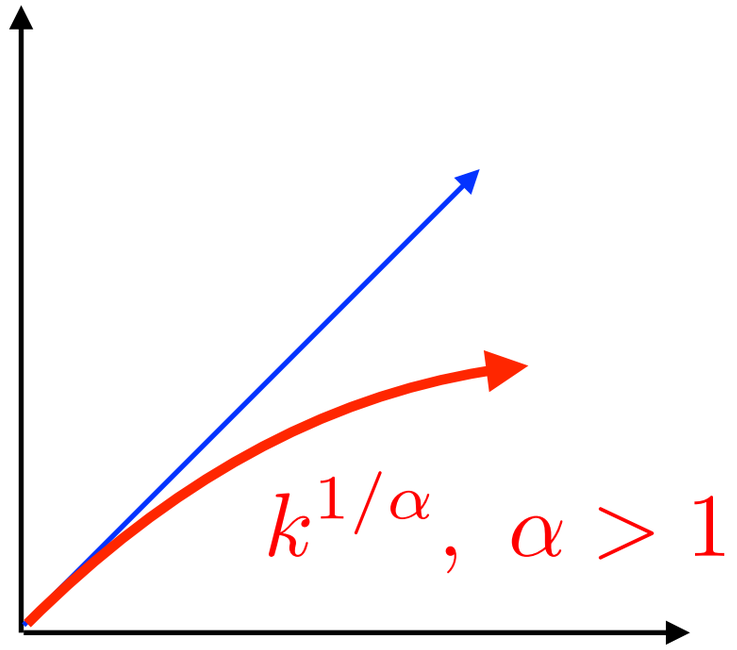
**Limited
Parallelizable Jobs**

Prior Work





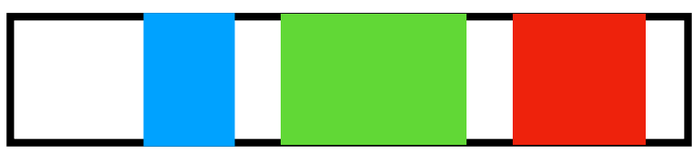
Multiple Servers

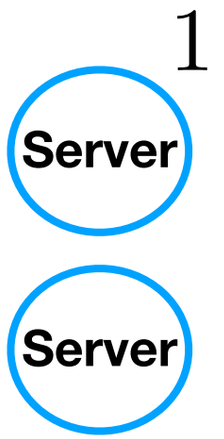


Limited
Parallelizable Jobs

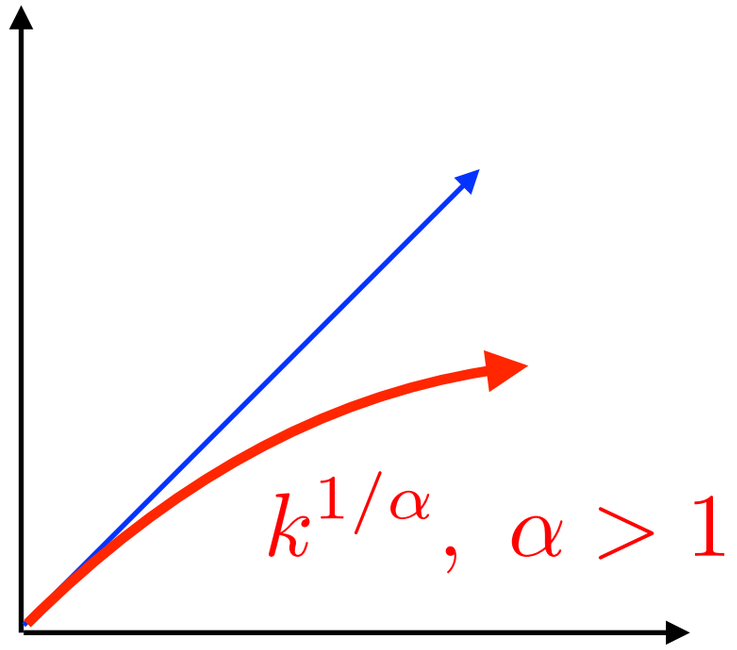
All jobs available at time 0,

Prior Work





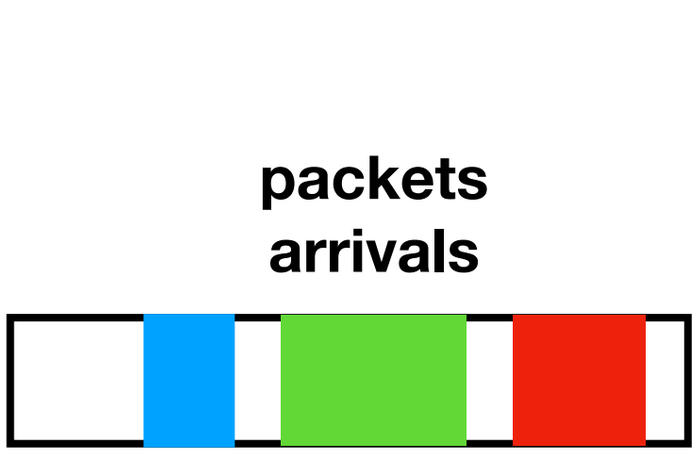
Multiple Servers

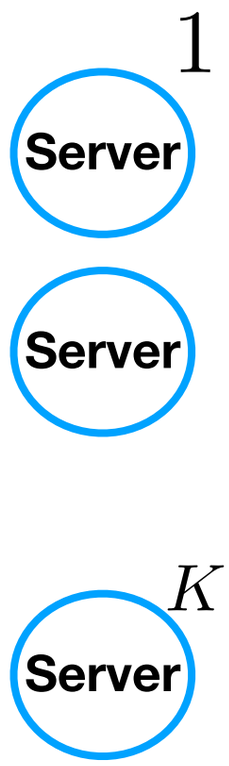


Limited Parallelizable Jobs

All jobs available at time 0, Optimal Scheduling : heSRPT **[Berg et al' 20]**

Prior Work

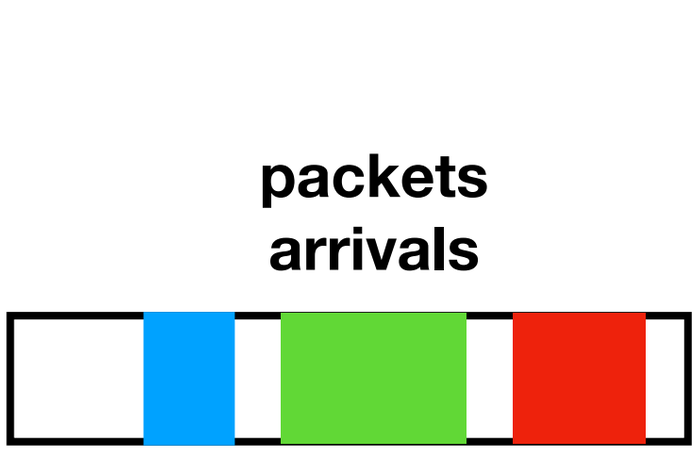


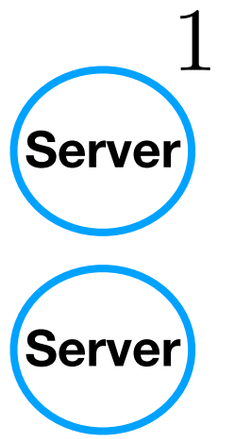


Multiple Servers

All jobs available at time 0, Optimal Scheduling : heSRPT **[Berg et al' 20]**

Prior Work



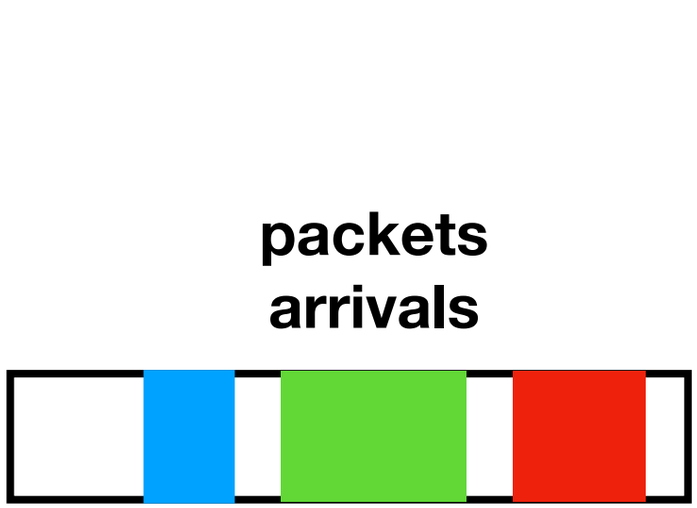


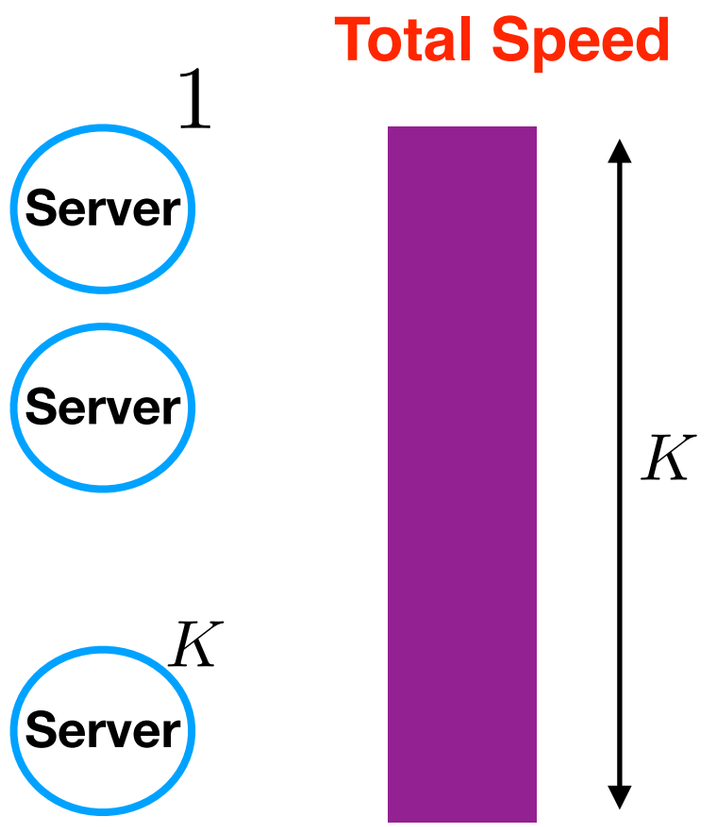
Multiple Servers

All jobs available at time 0, Optimal Scheduling : heSRPT **[Berg et al' 20]**

All jobs get non-zero speed, while shorter jobs get more speed

Prior Work



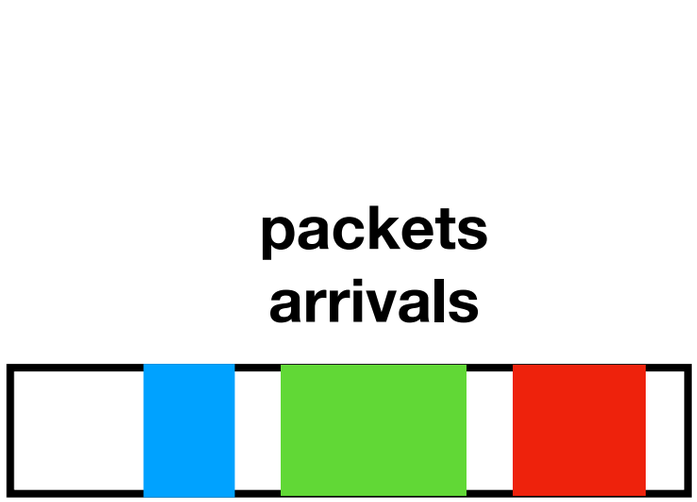


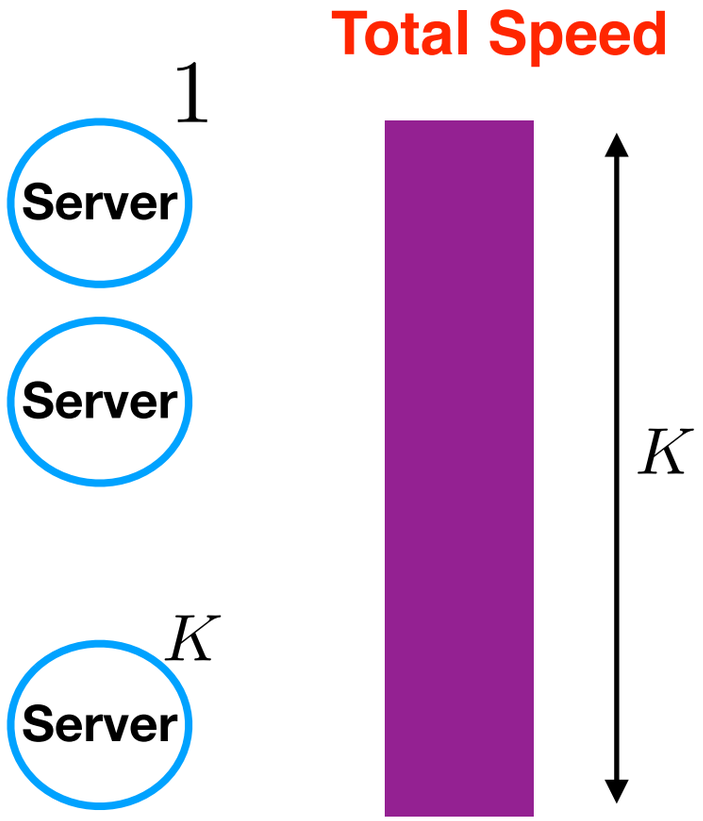
Multiple Servers

All jobs available at time 0, Optimal Scheduling : `heSRPT` **[Berg et al' 20]**

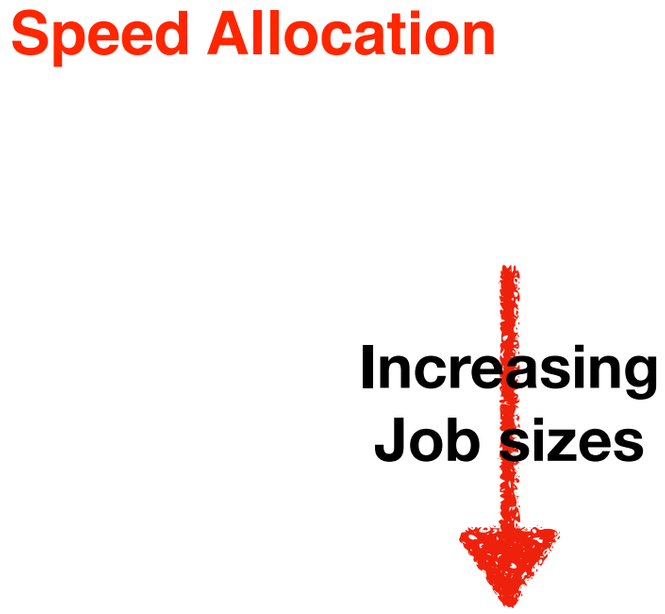
All jobs get non-zero speed, while shorter jobs get more speed

Prior Work





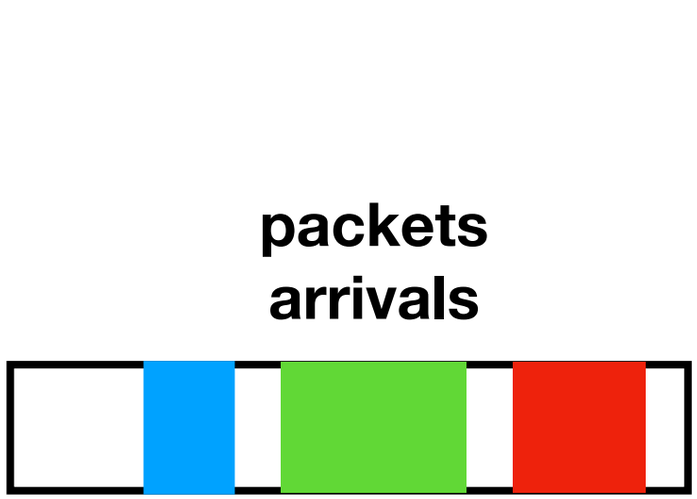
Multiple Servers

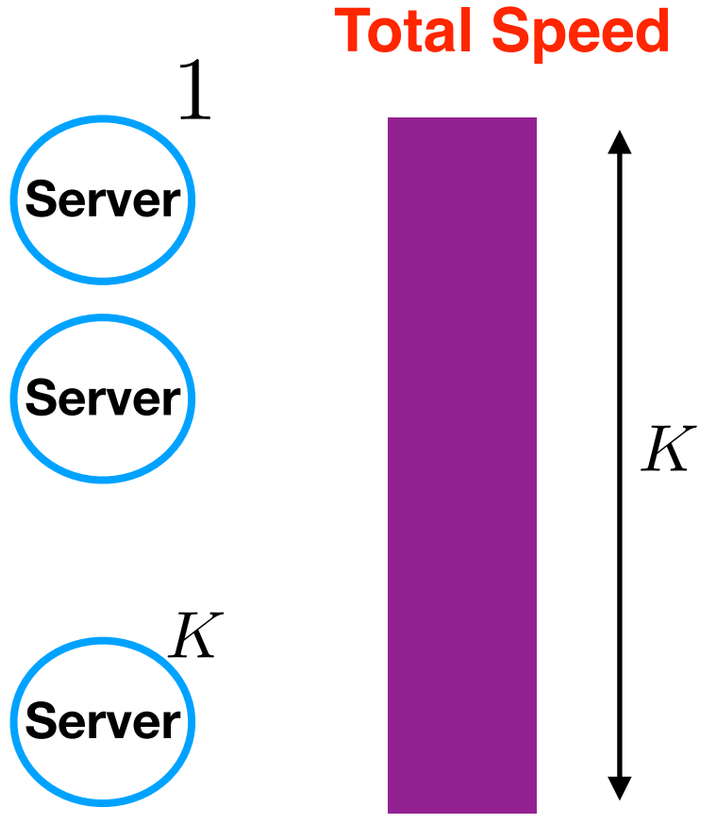


All jobs available at time 0, Optimal Scheduling : `heSRPT` **[Berg et al' 20]**

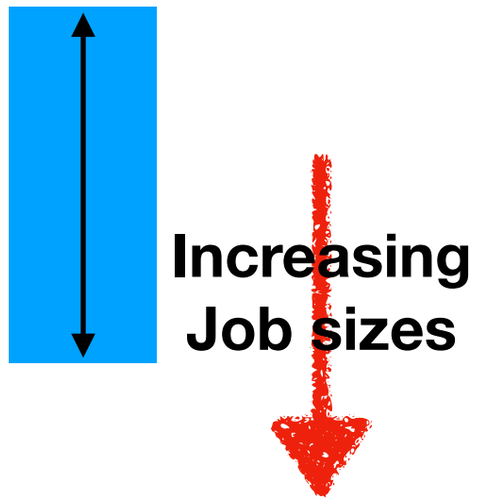
All jobs get non-zero speed, while shorter jobs get more speed

Prior Work





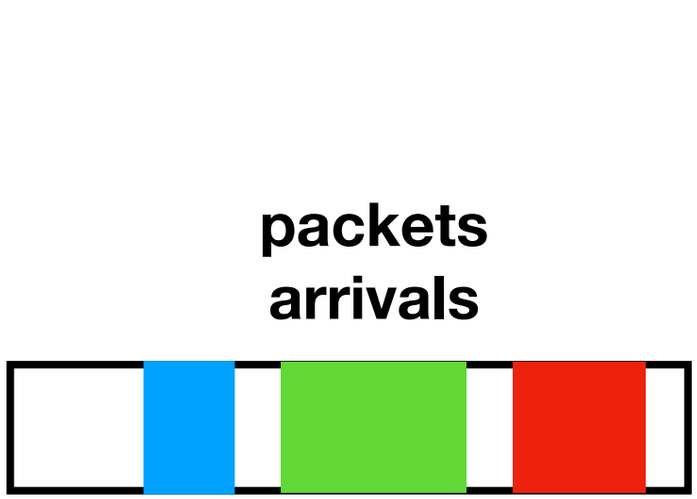
Multiple Servers

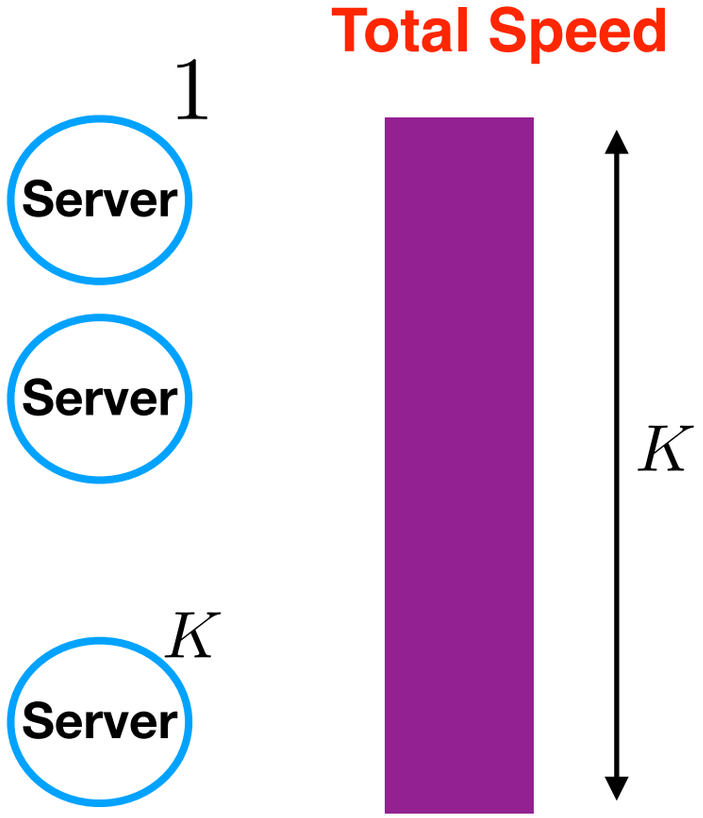


All jobs available at time 0, Optimal Scheduling : heSRPT **[Berg et al' 20]**

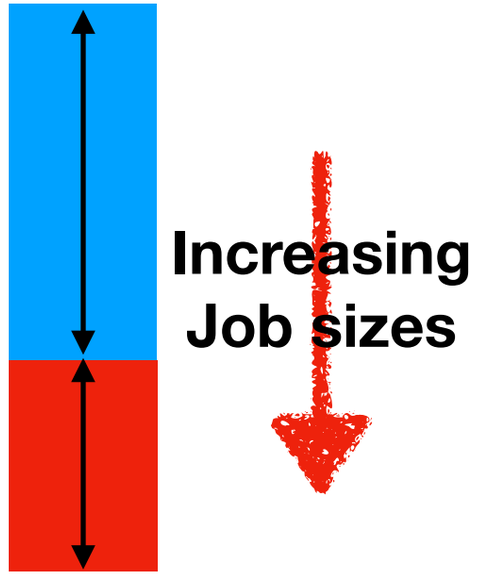
All jobs get non-zero speed, while shorter jobs get more speed

Prior Work





Multiple Servers



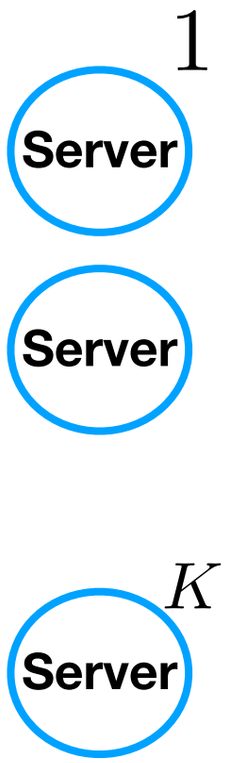
All jobs available at time 0, Optimal Scheduling : heSRPT **[Berg et al' 20]**

All jobs get non-zero speed, while shorter jobs get more speed

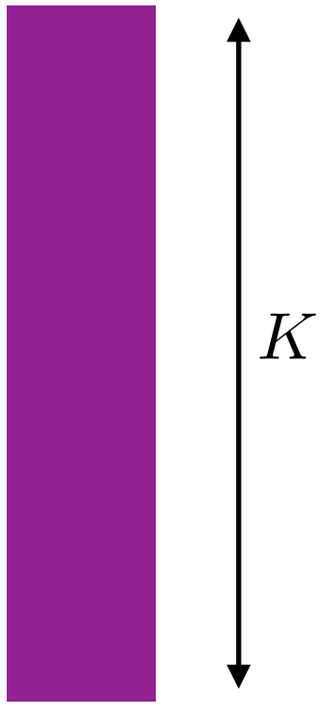
Prior Work

packets arrivals

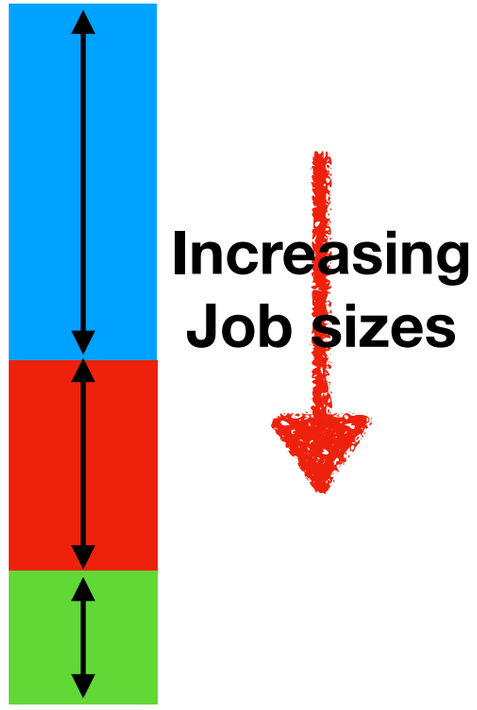
Multiple Servers



Total Speed



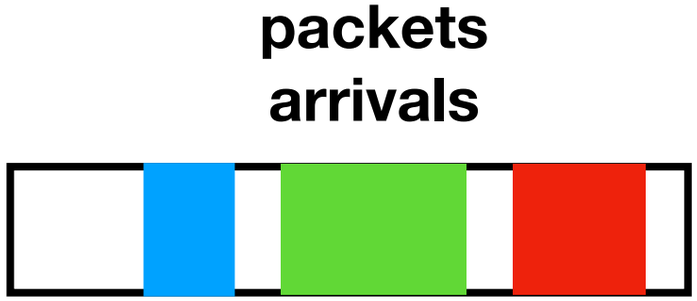
Speed Allocation

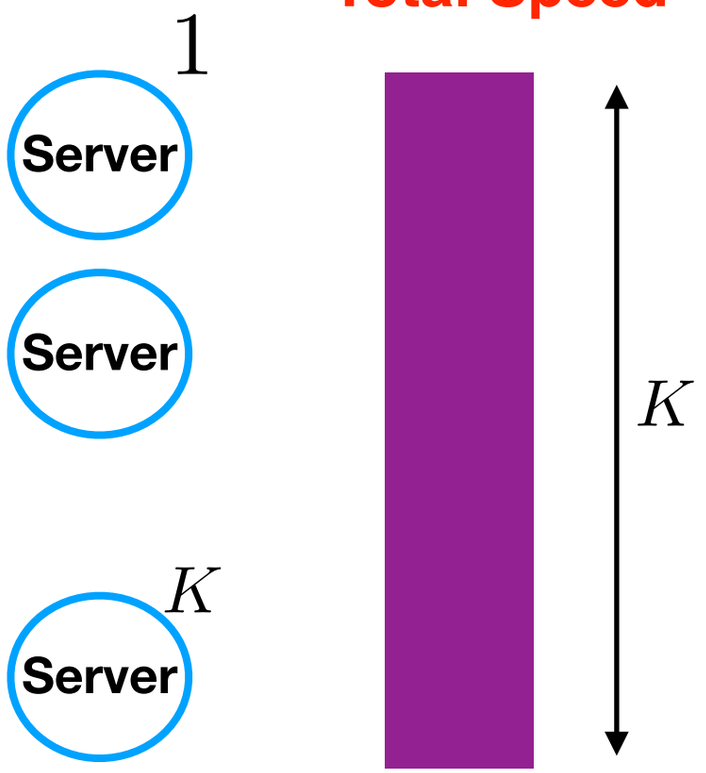


All jobs available at time 0, Optimal Scheduling : heSRPT **[Berg et al' 20]**

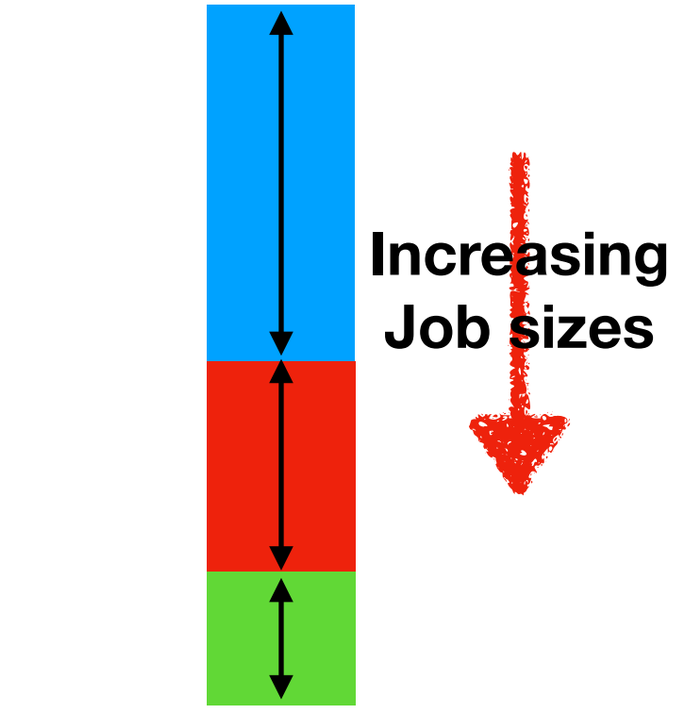
All jobs get non-zero speed, while shorter jobs get more speed

Prior Work





Multiple Servers



All jobs available at time 0, Optimal Scheduling : heSRPT **[Berg et al' 20]**

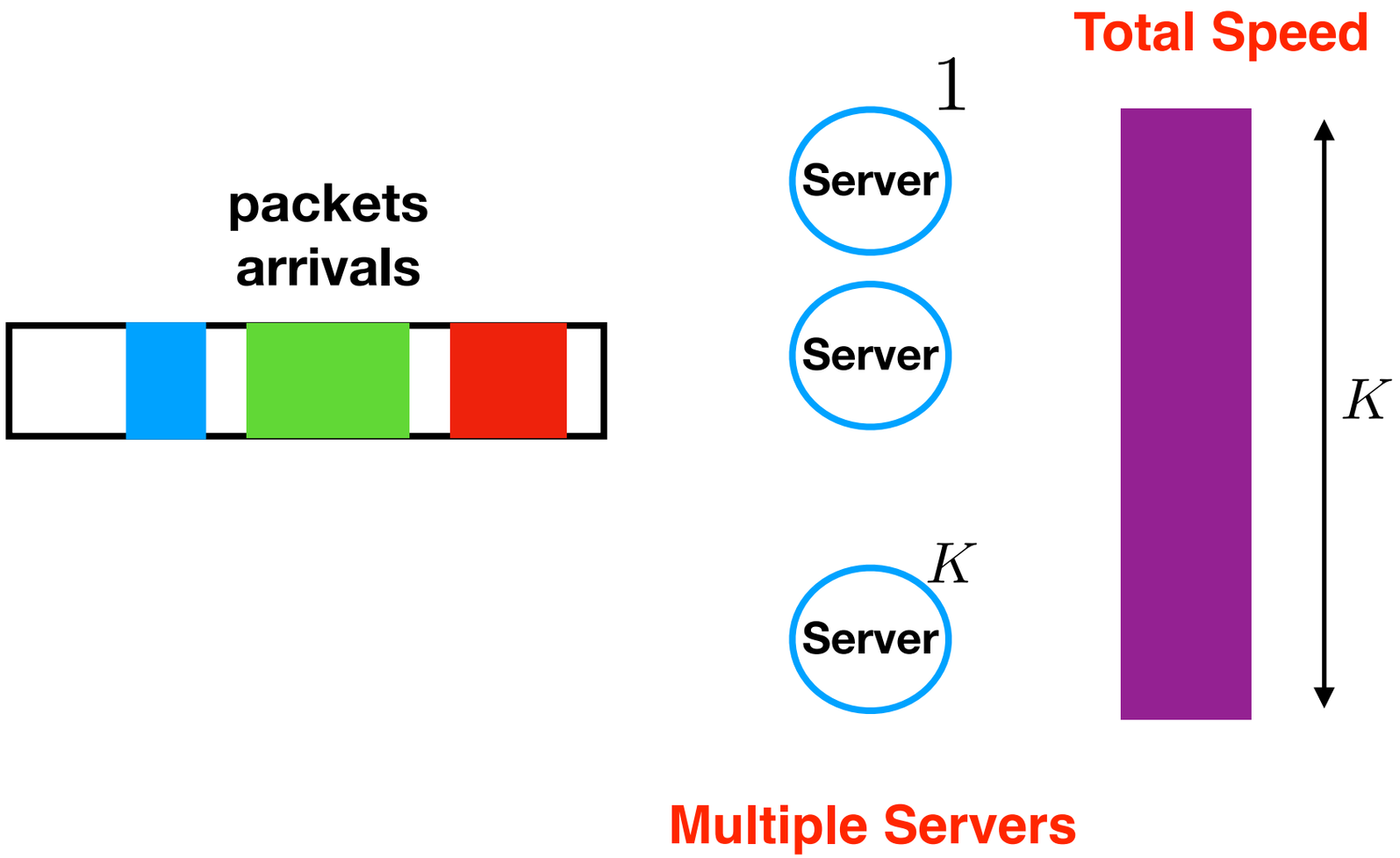
All jobs get non-zero speed, while shorter jobs get more speed

Among n(t) outstanding jobs

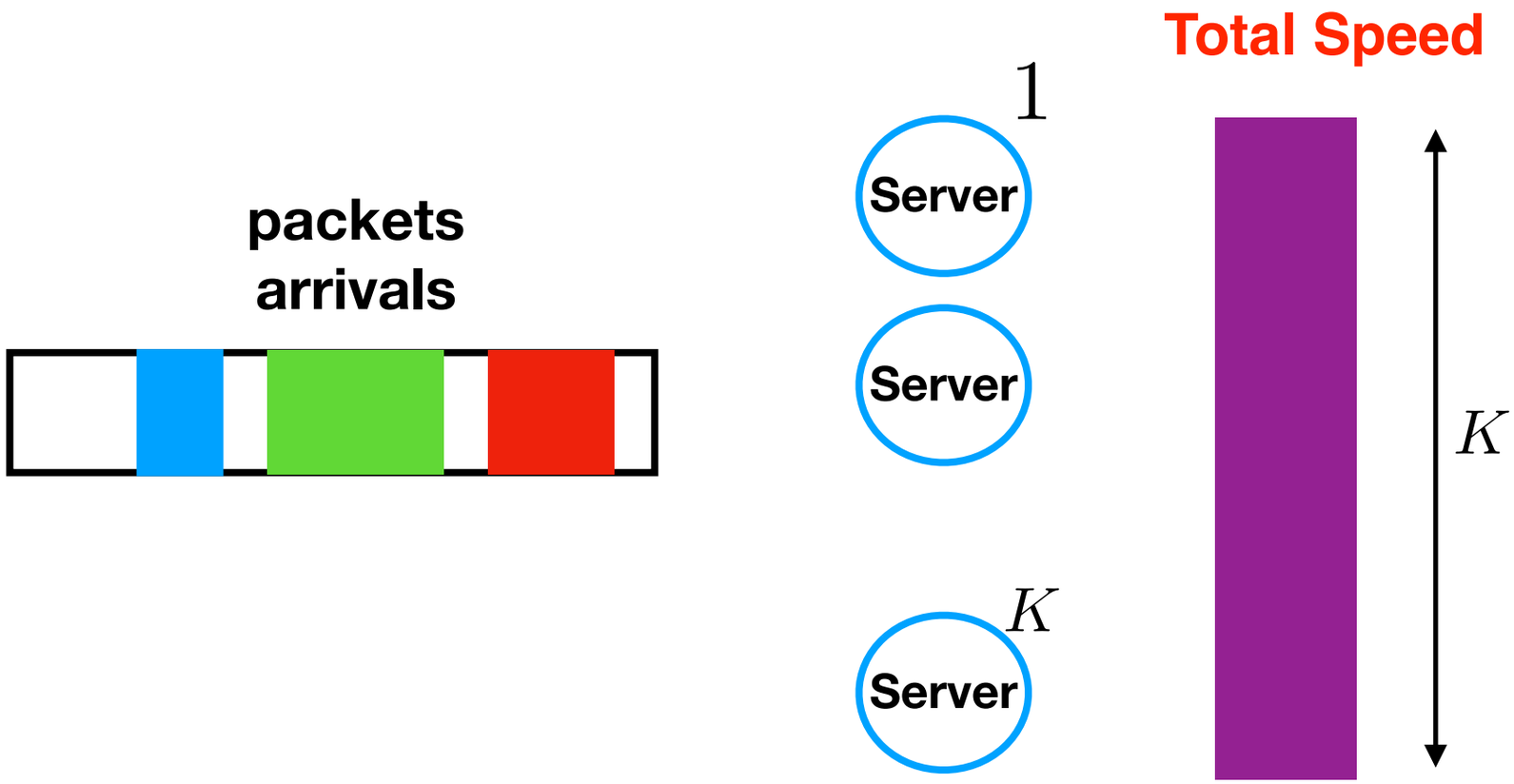
ith shortest job's speed

$$s_i(t) = K\left(\left(\frac{i}{n(t)}\right)^{\left(\frac{1}{1-1/\alpha}\right)} - \left(\frac{i-1}{n(t)}\right)^{\left(\frac{1}{1-1/\alpha}\right)}\right)$$

Simple Algorithm EQUI



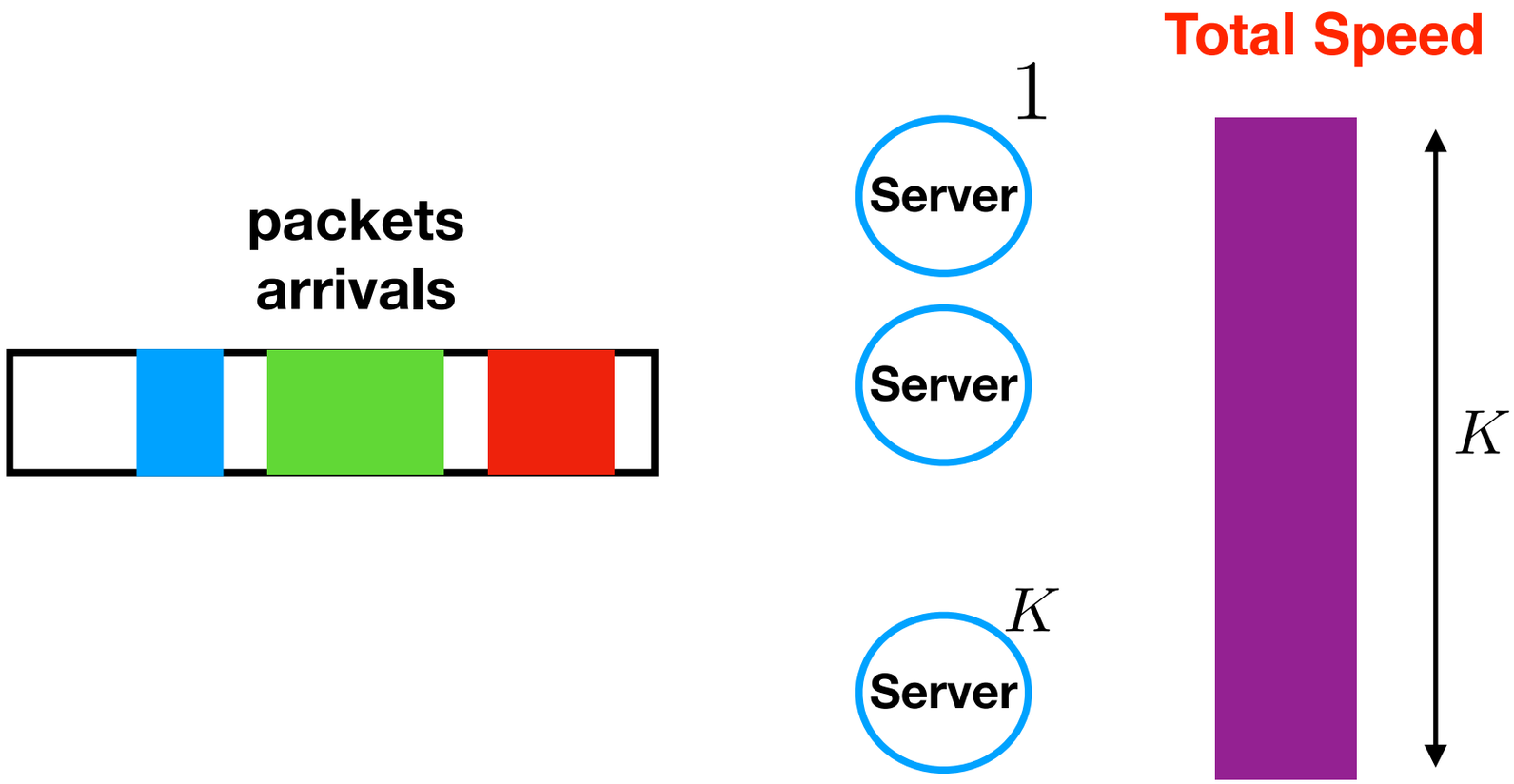
Simple Algorithm EQUI



Multiple Servers

All jobs available at time 0,

Simple Algorithm EQUI

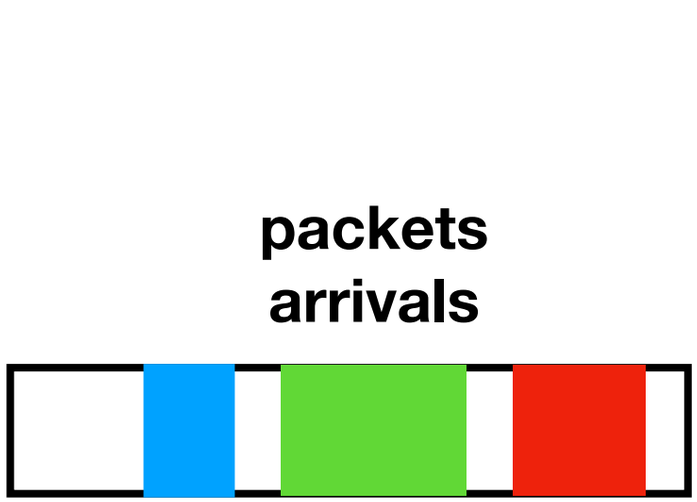


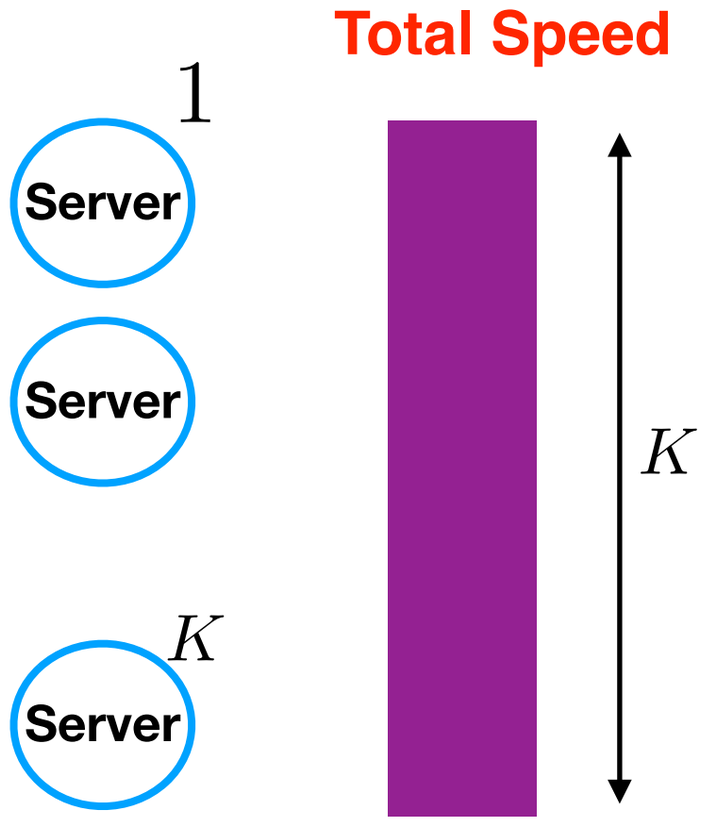
Multiple Servers

All jobs available at time 0,

Assign equal speed to all jobs

Simple Algorithm EQUI



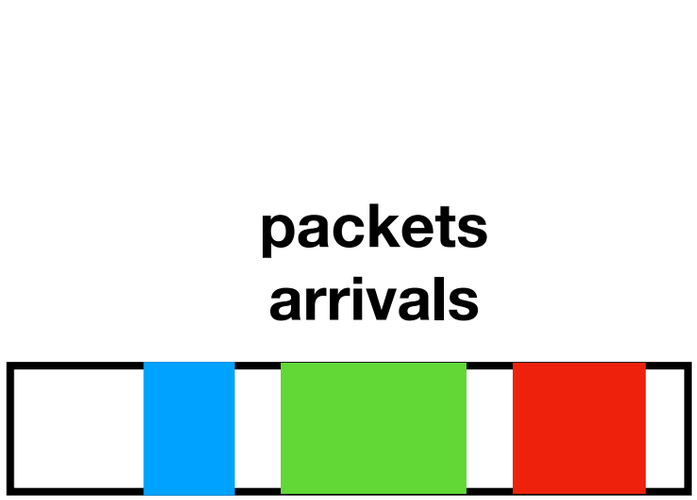


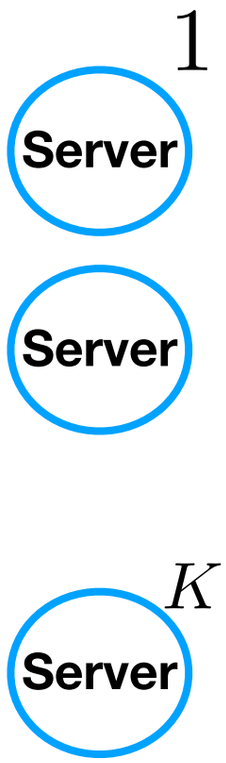
Multiple Servers

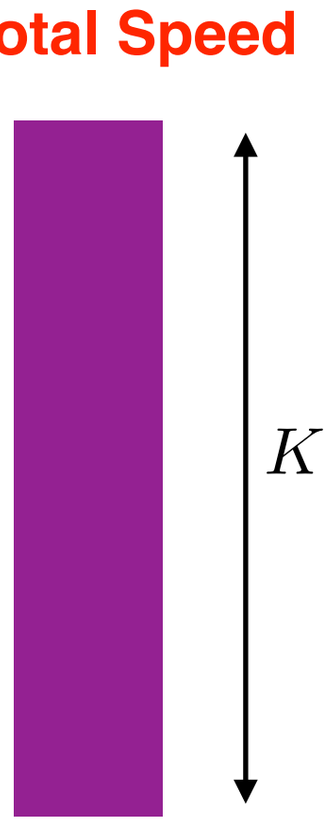
All jobs available at time 0,

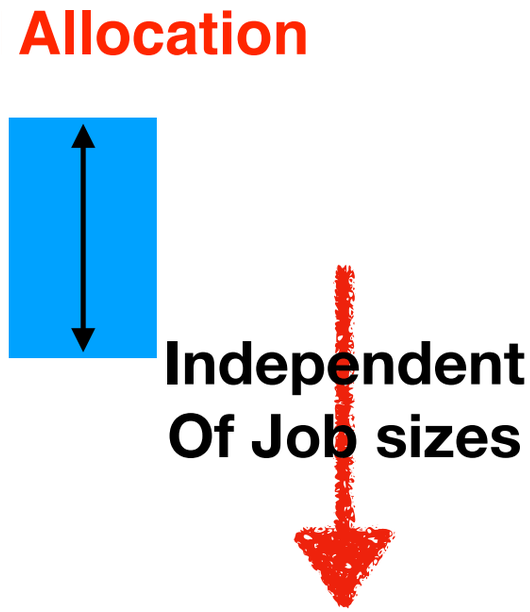
Assign equal speed to all jobs

Simple Algorithm EQUI







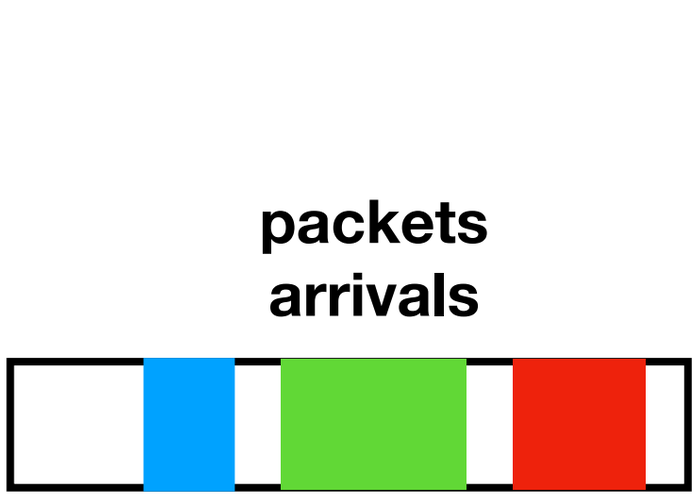


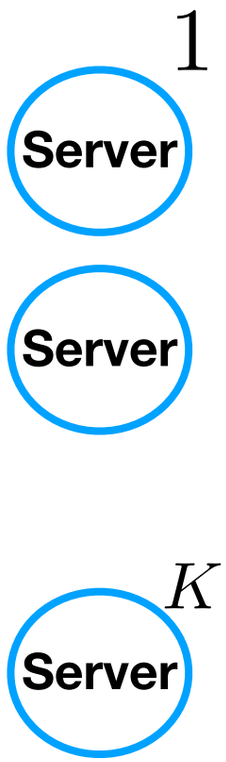
Multiple Servers

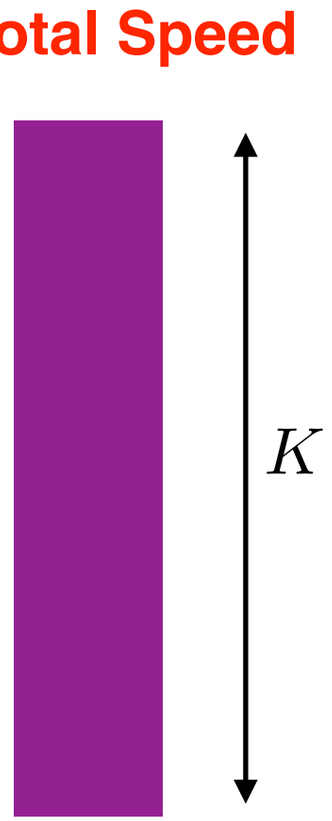
All jobs available at time 0,

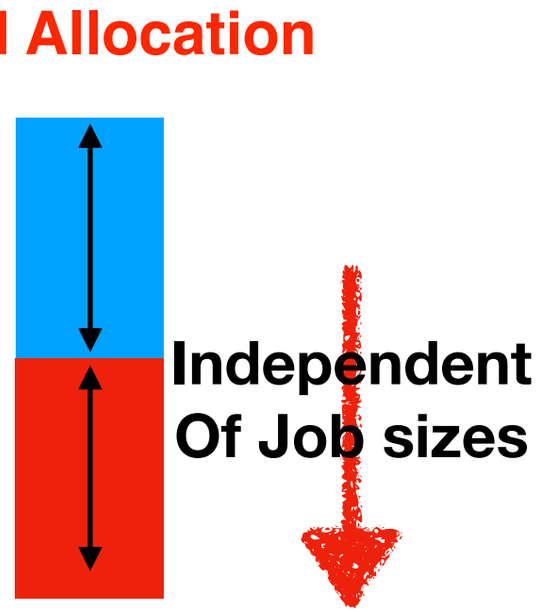
Assign equal speed to all jobs

Simple Algorithm EQUI







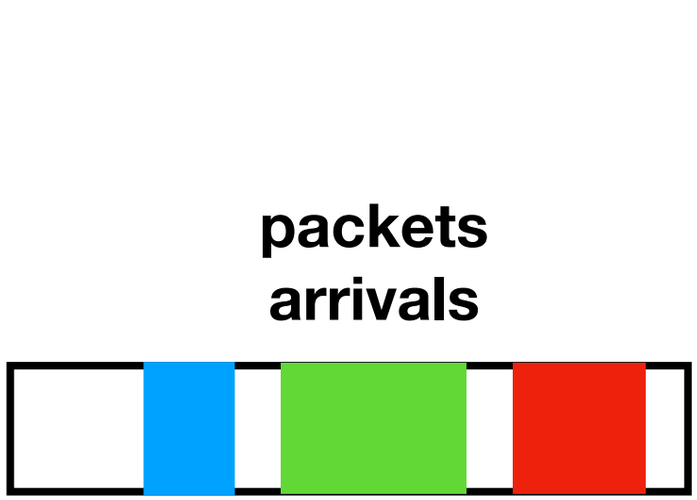


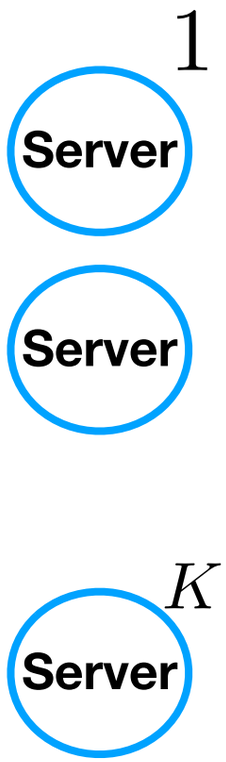
Multiple Servers

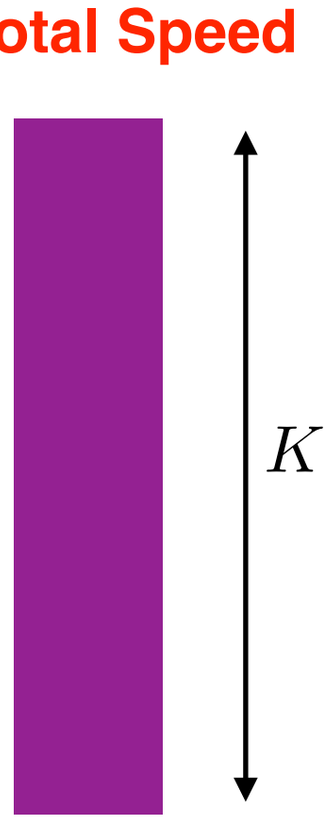
All jobs available at time 0,

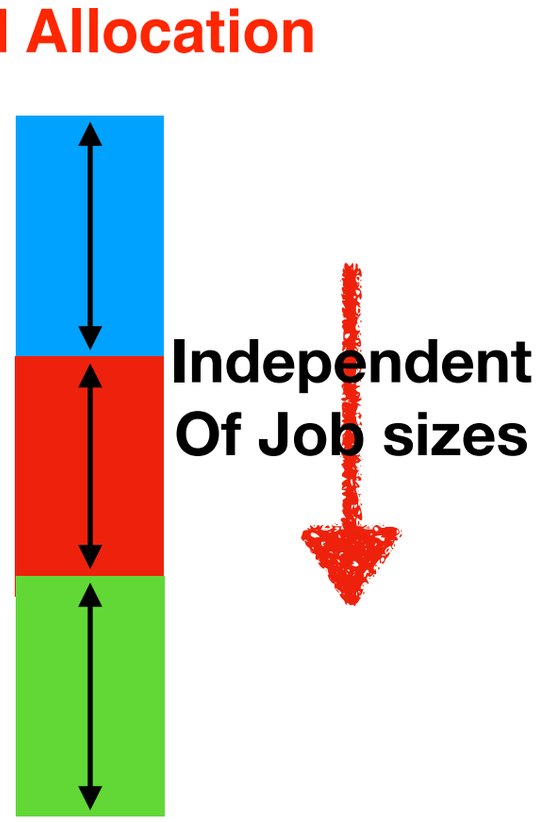
Assign equal speed to all jobs

Simple Algorithm EQUI









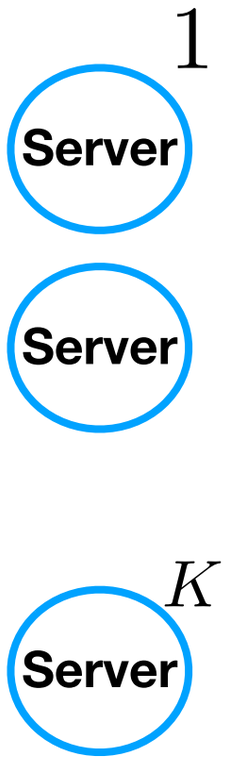
Multiple Servers

All jobs available at time 0,

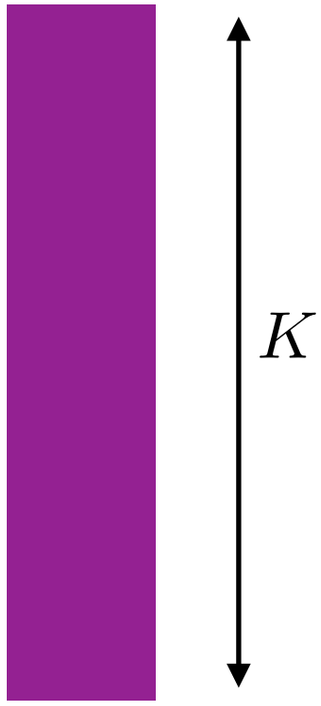
Assign equal speed to all jobs

Simple Algorithm EQUI

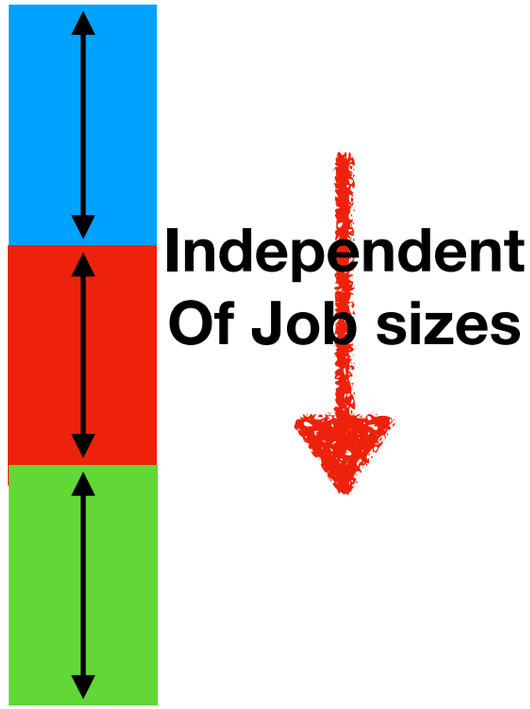
packets arrivals



Total Speed

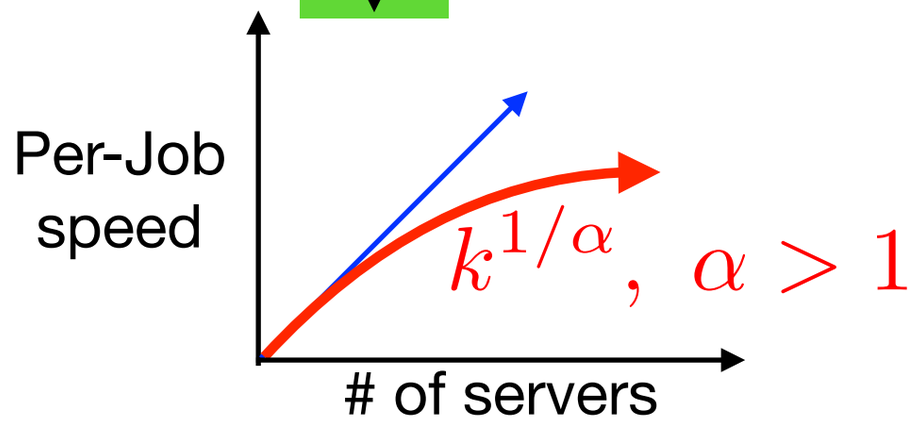


Speed Allocation



Multiple Servers

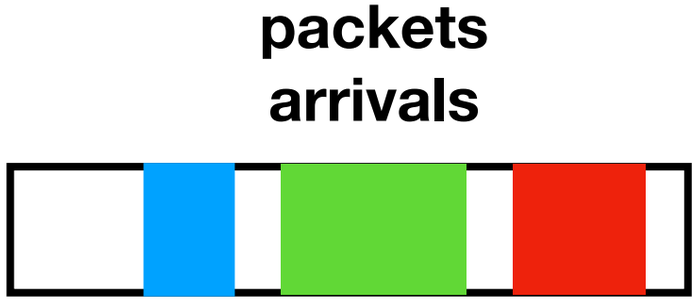
All jobs available at time 0,

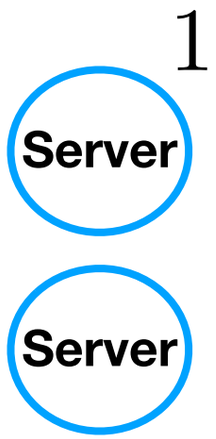


Assign equal speed to all jobs

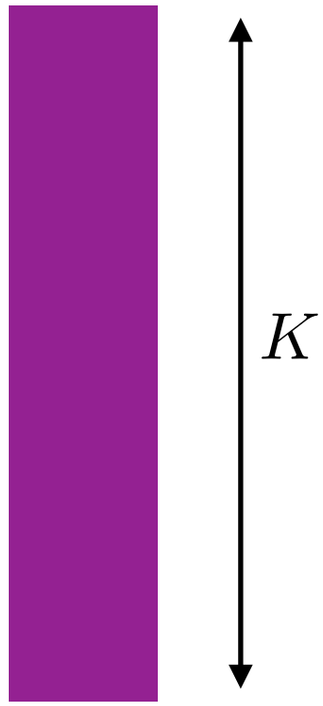
We show that Approximation Ratio of EQUI $\left(2 - \frac{1}{\alpha}\right) \frac{1}{\left(1 - \left(\frac{1}{\alpha}\right)\right)}$

Simple Algorithm EQUI

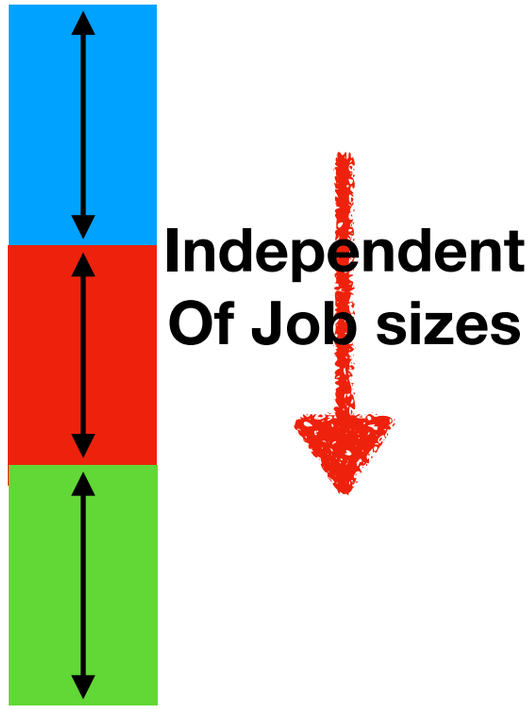




Total Speed

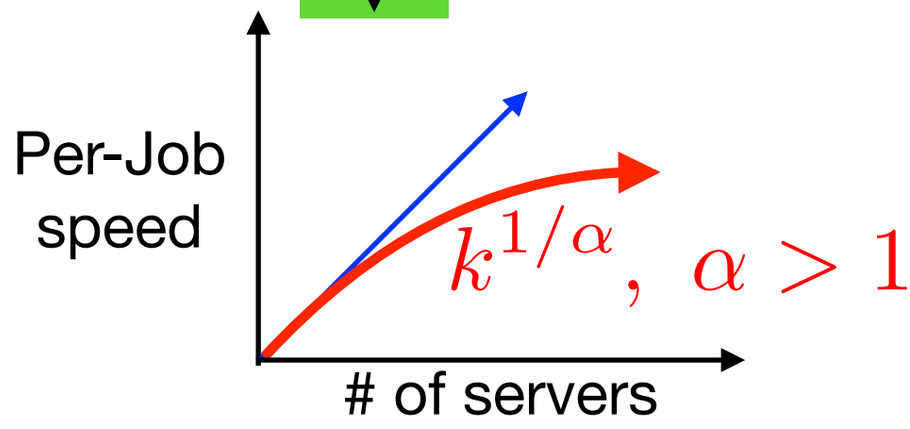


Speed Allocation



Multiple Servers

All jobs available at time 0,



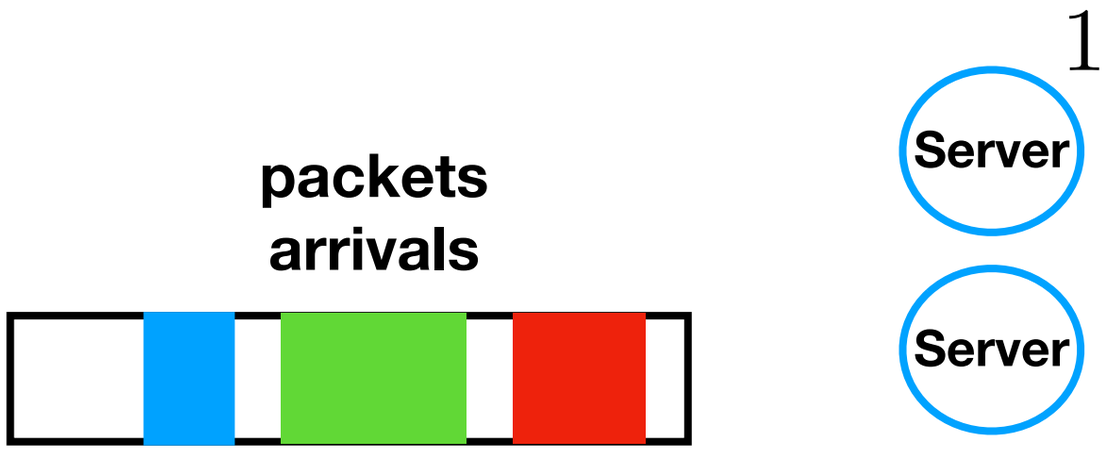
Assign equal speed to all jobs

We show that Approximation Ratio of EQUI $\left(2 - \frac{1}{\alpha}\right) \frac{1}{\left(1 - \left(\frac{1}{\alpha}\right)\right)}$

EQUI is simpler than heSRPT to implement

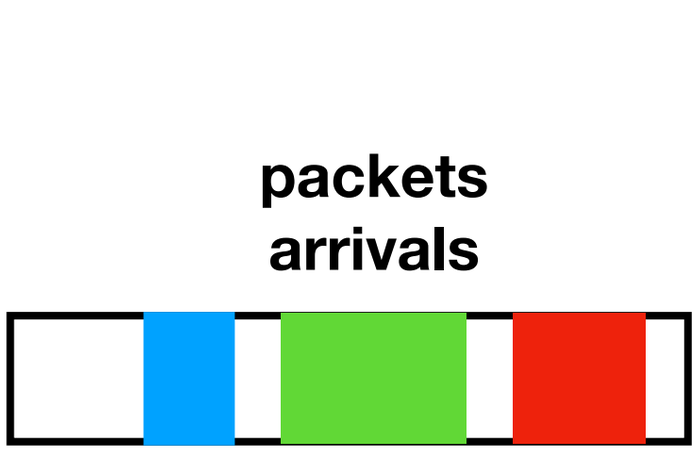
Real Problem: Online

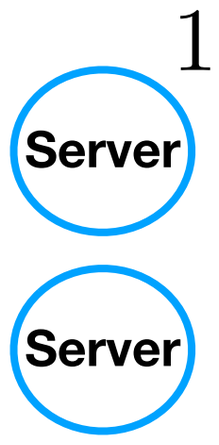
Real Problem: Online

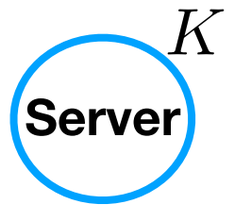


Multiple Servers

Real Problem: Online







Multiple Servers

Jobs arrive over time, arbitrary time and sizes

Metric for Online Algorithms

Metric for Online Algorithms

Competitive ratio

Metric for Online Algorithms

Competitive ratio ratio of the cost of an **online** and the **offline Opt** algorithm

$$r_{\mathsf{ON}} = \max_{\sigma} \frac{v_{\mathsf{ON}}(\sigma)}{v_{\mathsf{OPT}}(\sigma)}$$

Worst Case Input

Metric for Online Algorithms

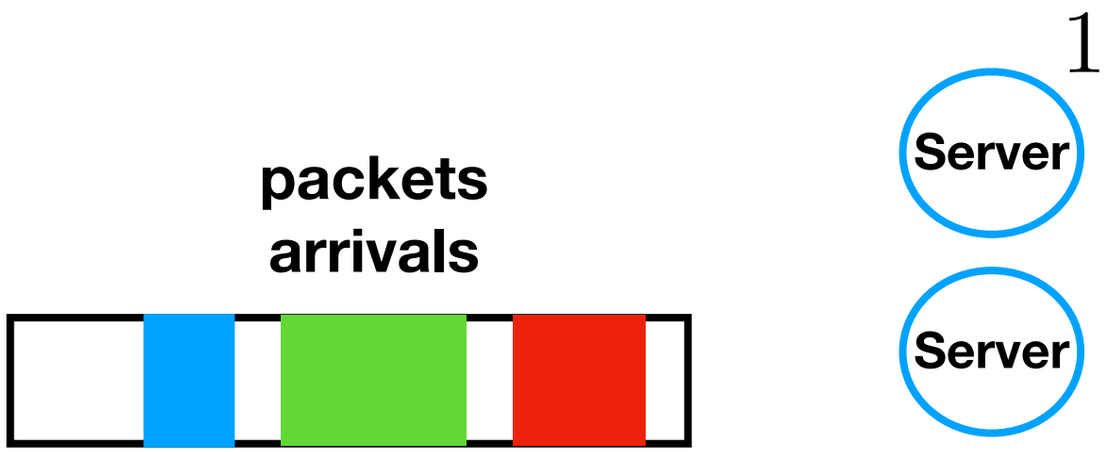
Competitive ratio ratio of the cost of an **online** and the **offline Opt** algorithm

$$r_{\mathsf{ON}} = \max_{\sigma} \frac{v_{\mathsf{ON}}(\sigma)}{v_{\mathsf{OPT}}(\sigma)}$$

Worst Case Input

Goal online algorithm with least **CR**

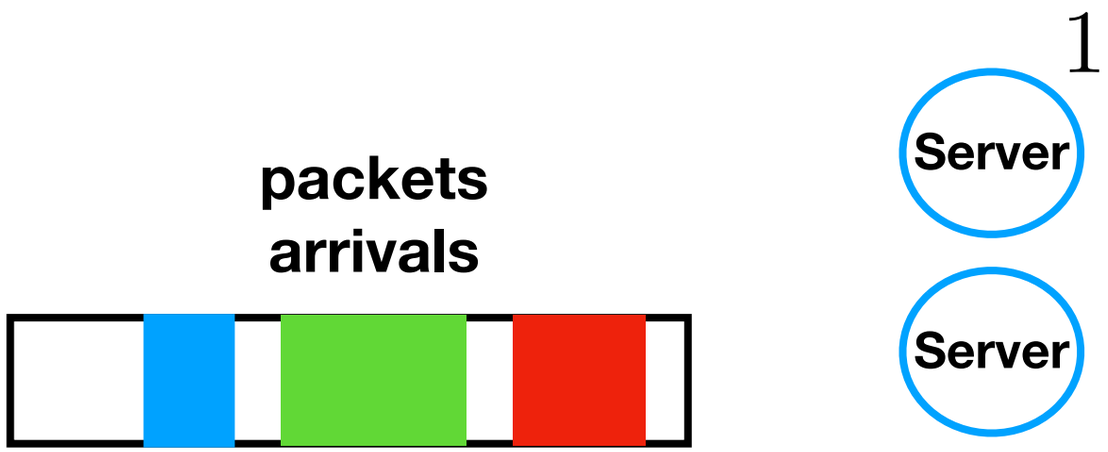
Prior Work : Online Case



Multiple Servers

Jobs arrive over time, arbitrary time and sizes

Prior Work : Online Case

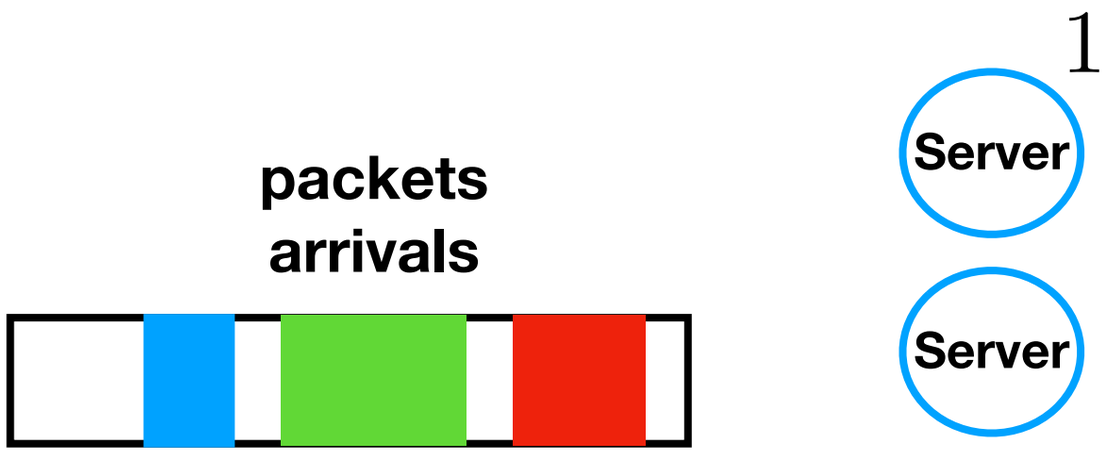


Multiple Servers

Jobs arrive over time, arbitrary time and sizes

Not much is known, unless resource augmentation is provided

Prior Work : Online Case



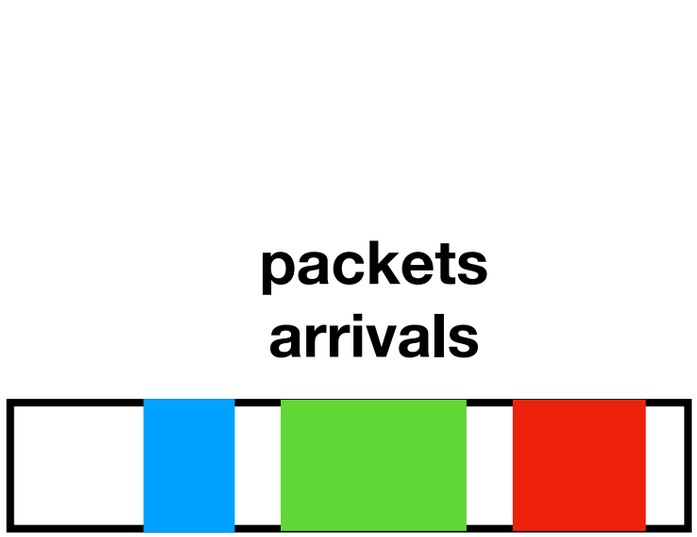
Multiple Servers

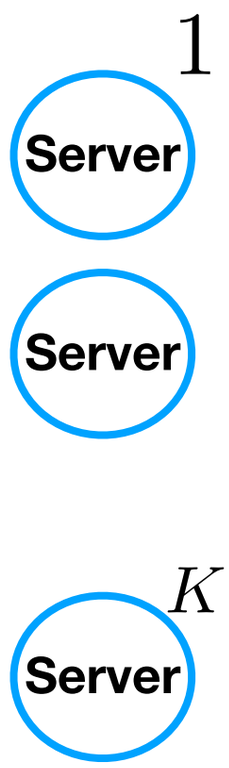
Jobs arrive over time, arbitrary time and sizes

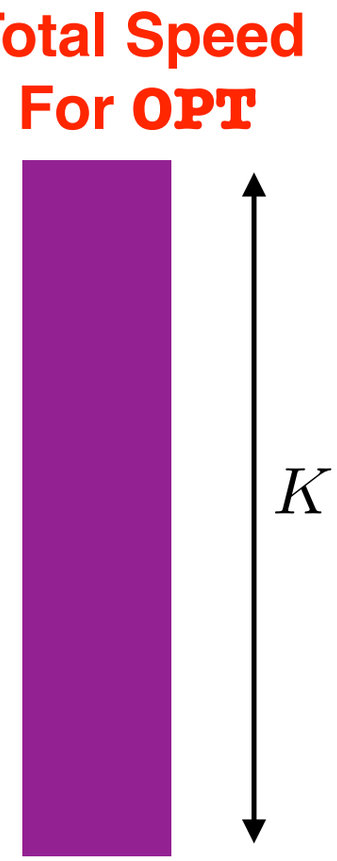
Not much is known, unless resource augmentation is provided

With resource augmentation an algorithm has more resources than OPT

Prior Work : Online Case







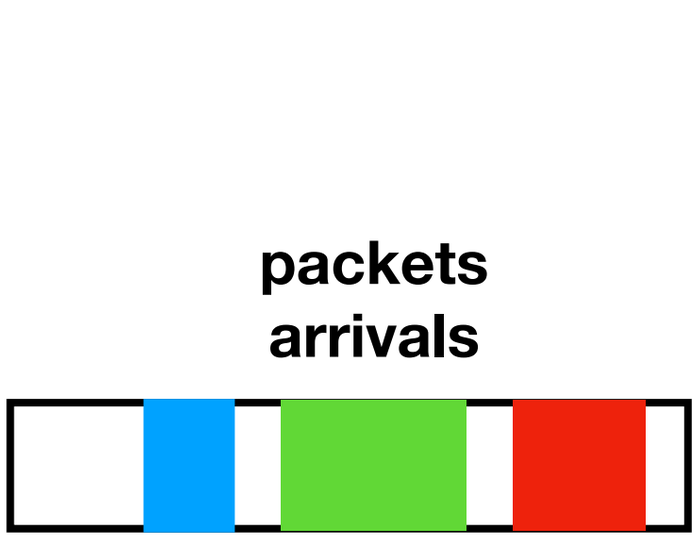
Multiple Servers

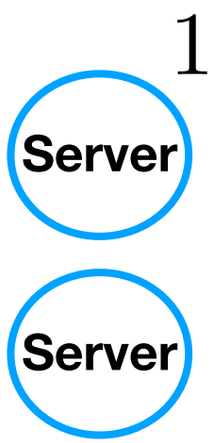
Jobs arrive over time, arbitrary time and sizes

Not much is known, unless resource augmentation is provided

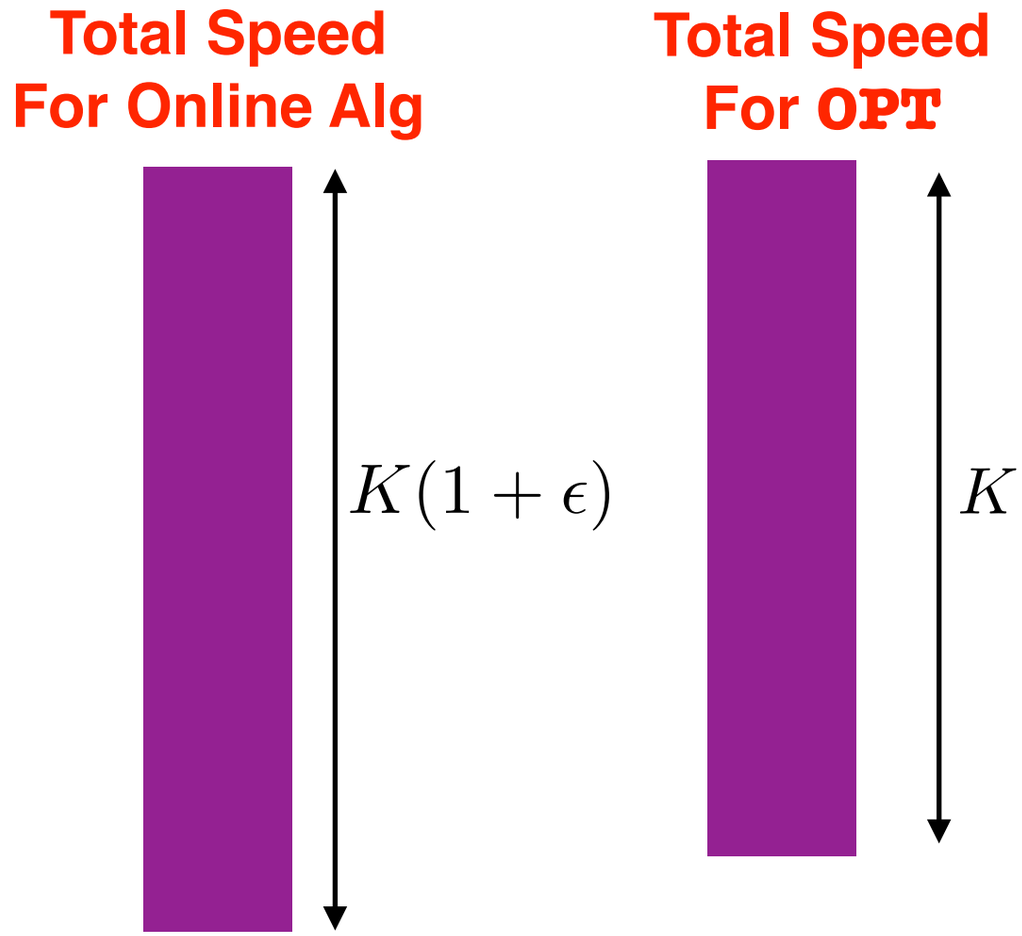
With resource augmentation an algorithm has more resources than OPT

Prior Work : Online Case





Multiple Servers

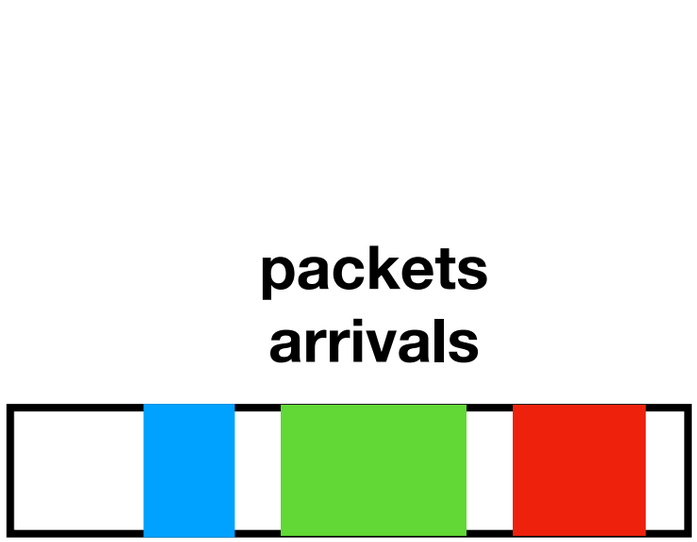


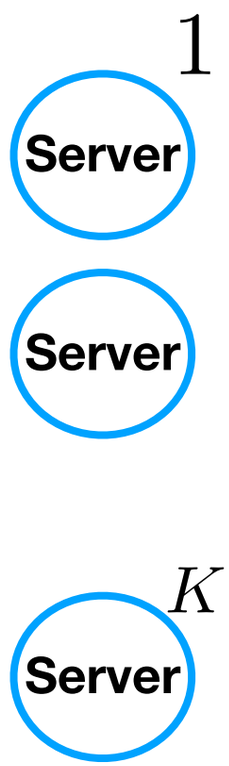
Jobs arrive over time, arbitrary time and sizes

Not much is known, unless resource augmentation is provided

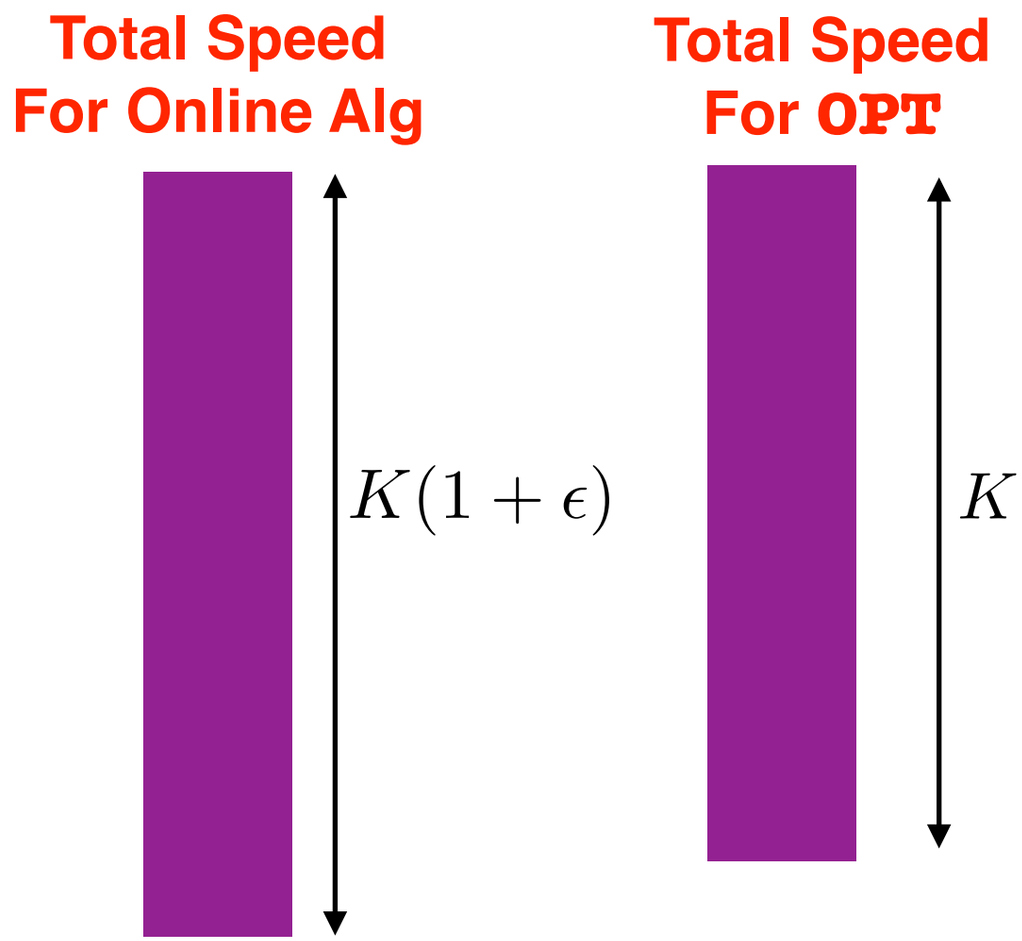
With resource augmentation an algorithm has more resources than OPT

Prior Work : Online Case





Multiple Servers



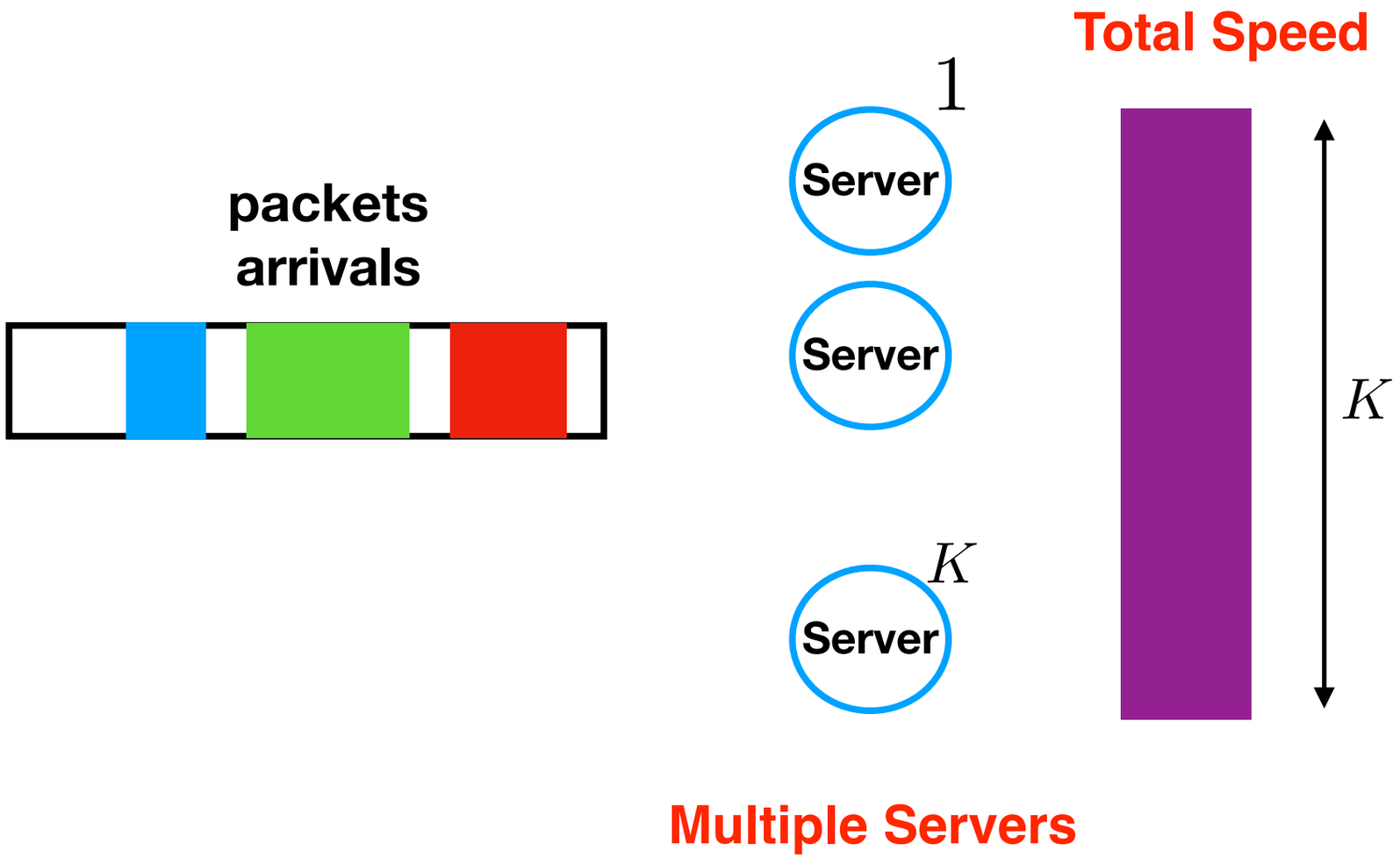
Jobs arrive over time, arbitrary time and sizes

Not much is known, unless resource augmentation is provided

With resource augmentation an algorithm has more resources than OPT

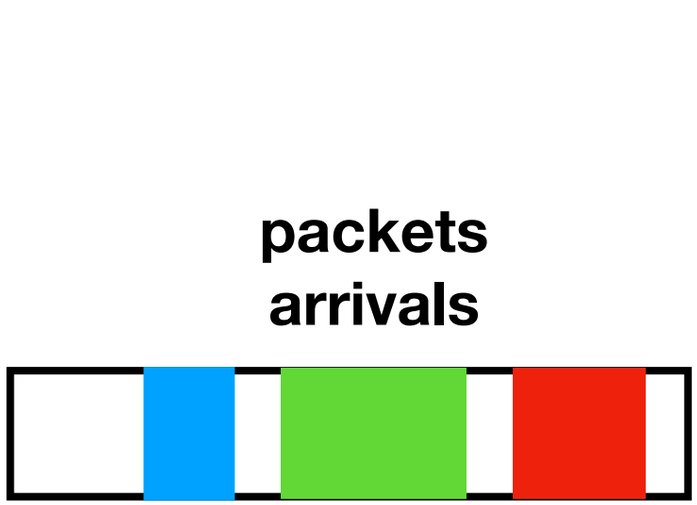
With resource augmentation constant competitive algorithms are known [*Edmonds* et al]

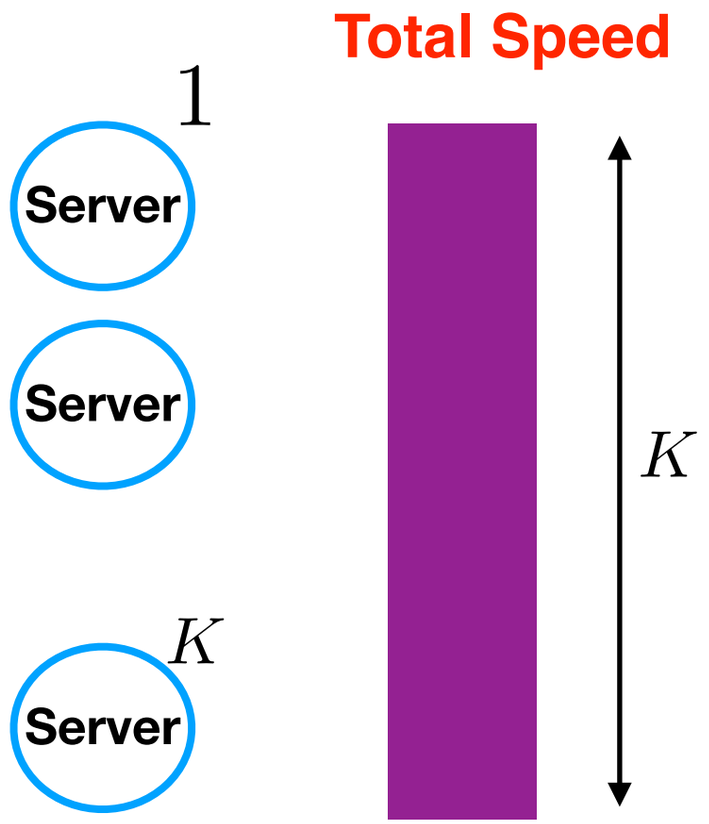
Our Algorithm



Multiple Servers

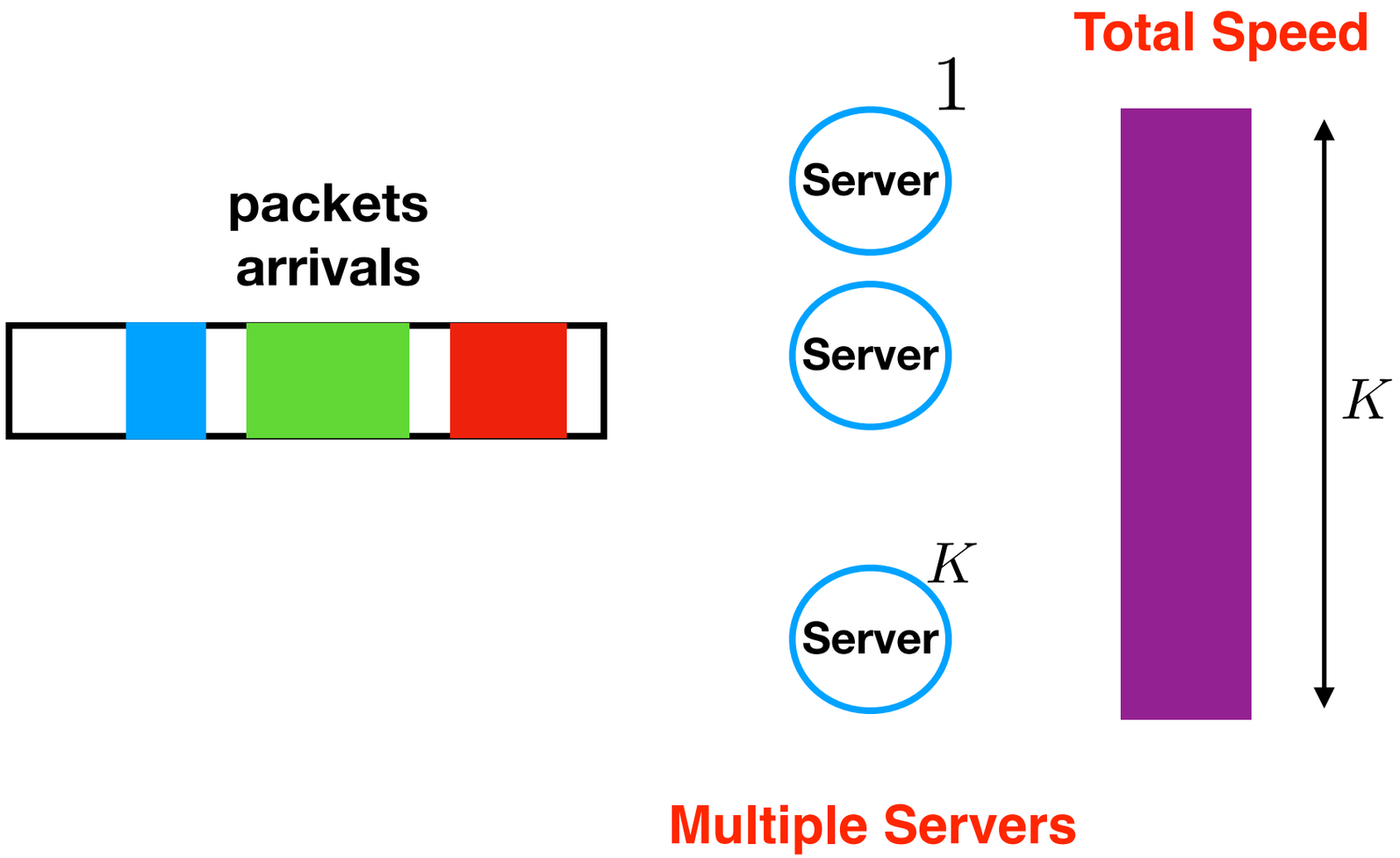
Our Algorithm LCFS-EQUI





Multiple Servers

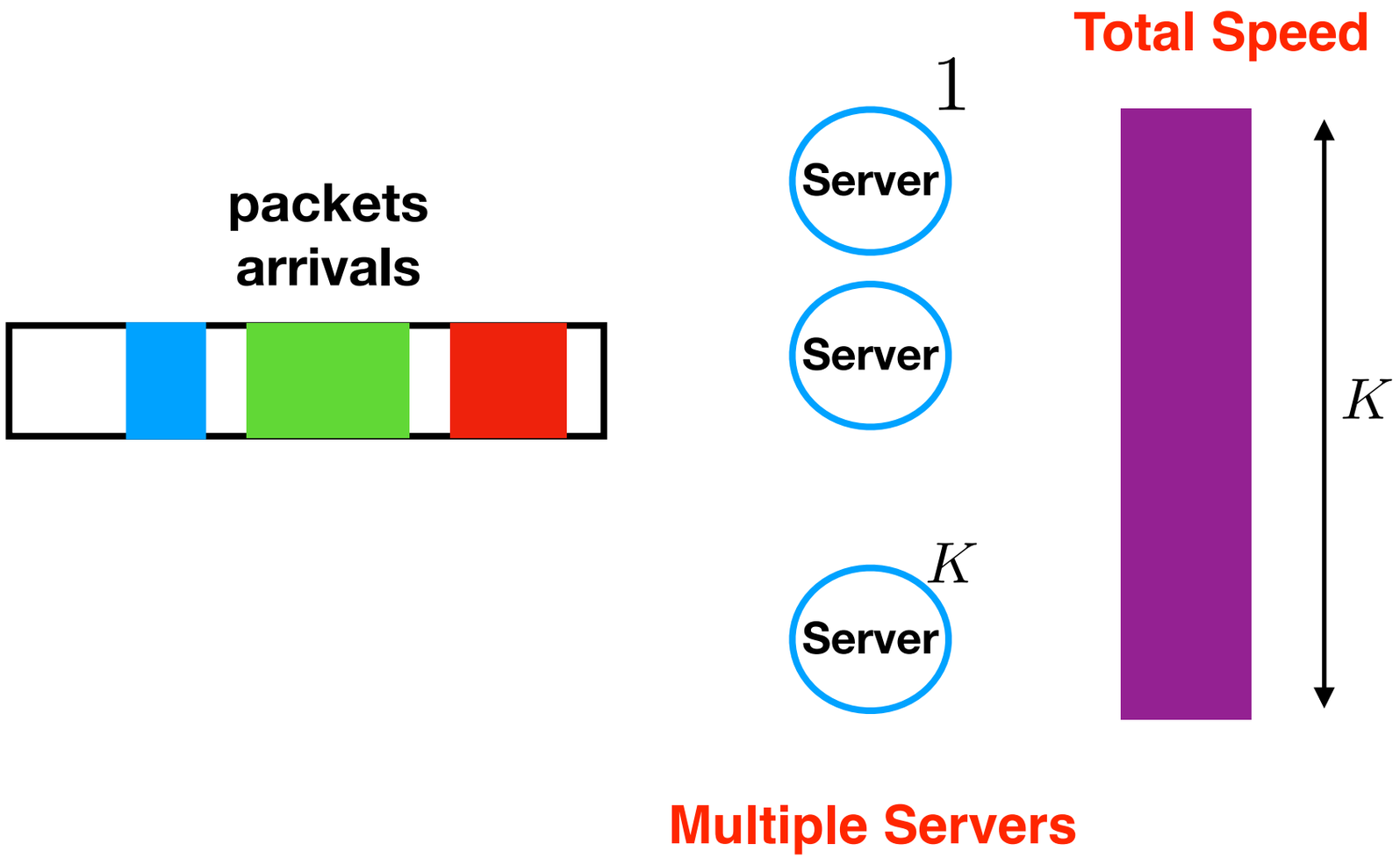
Our Algorithm LCFS-EQUI



Multiple Servers

Process the β-**fraction** of the **most recently arrived** jobs

Our Algorithm LCFS-EQUI

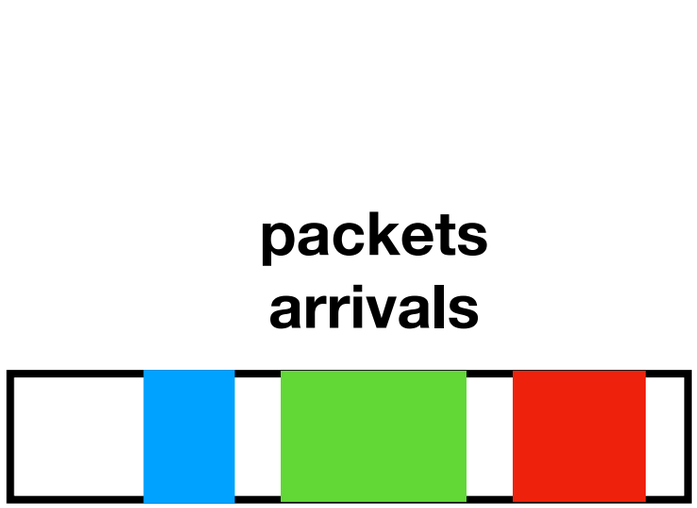


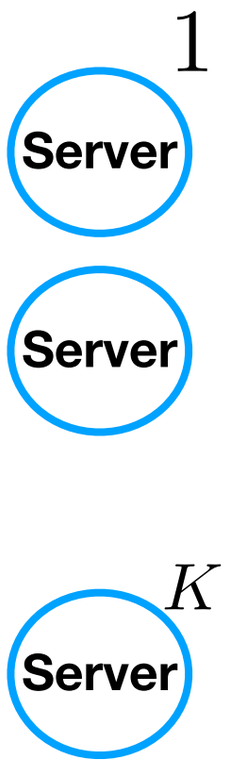
Multiple Servers

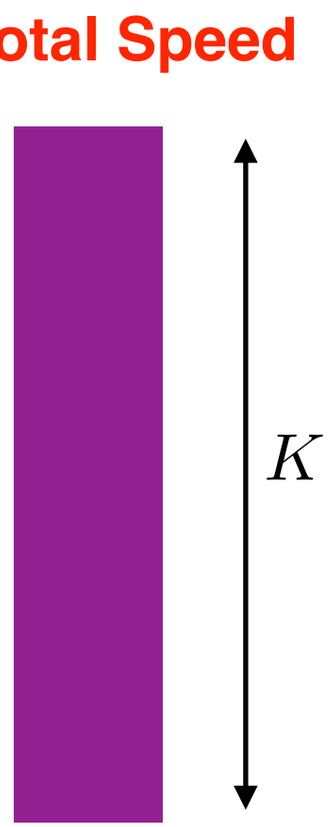
Process the β-**fraction** of the **most recently arrived** jobs

Assign **equal speed** to all jobs being processed

Our Algorithm LCFS-EQUI





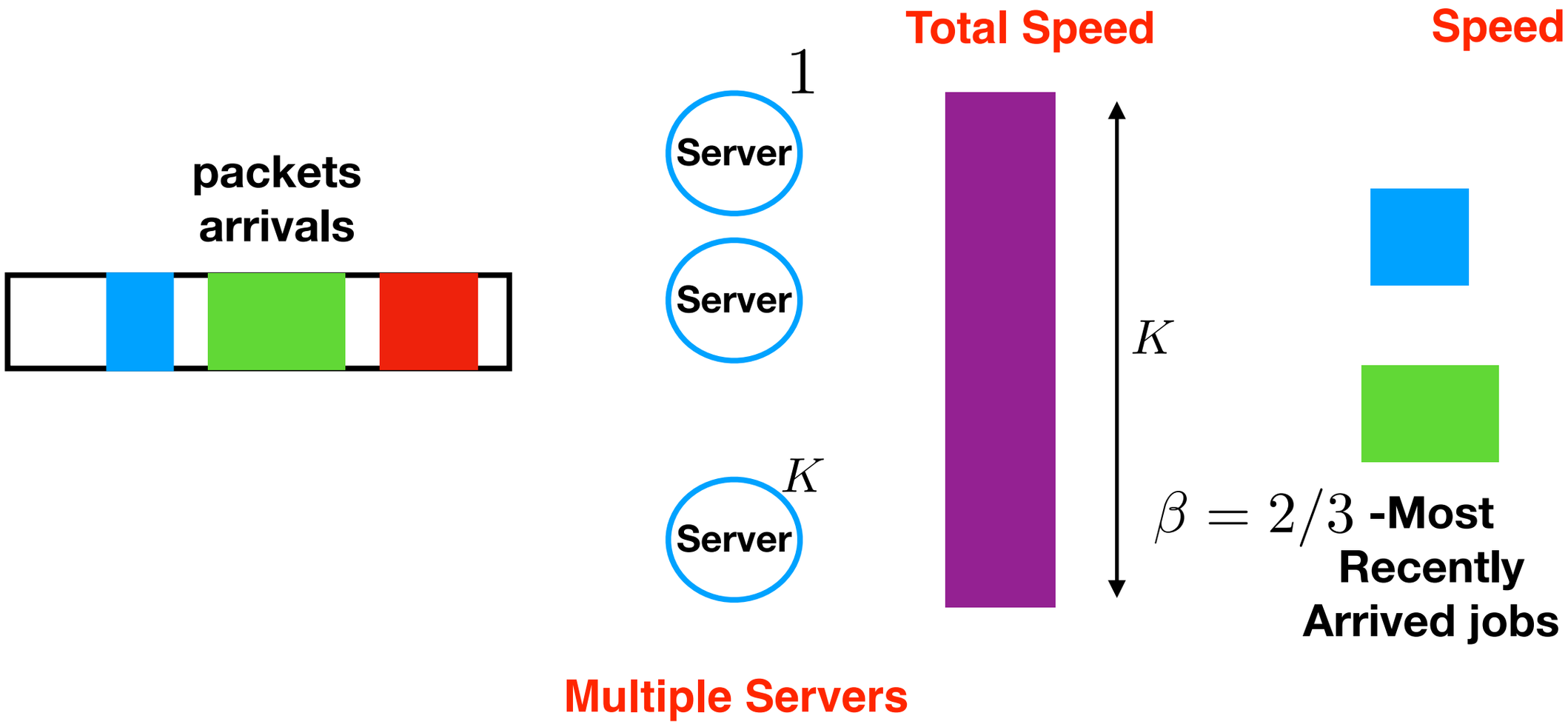


Multiple Servers

Process the β -**fraction** of the **most recently arrived** jobs

Assign **equal speed** to all jobs being processed

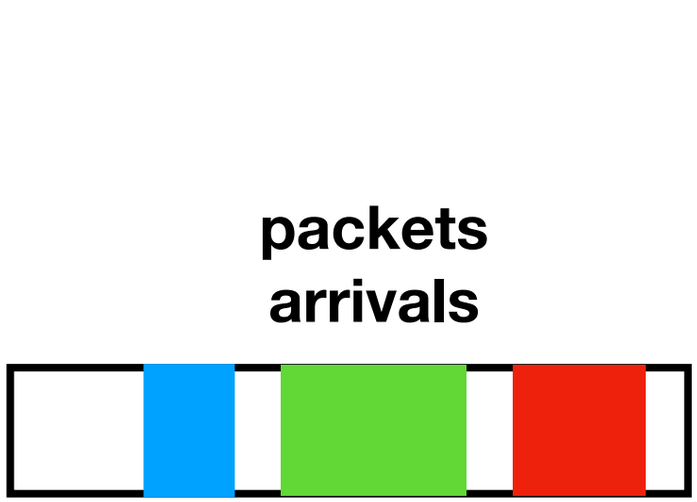
Our Algorithm LCFS-EQUI

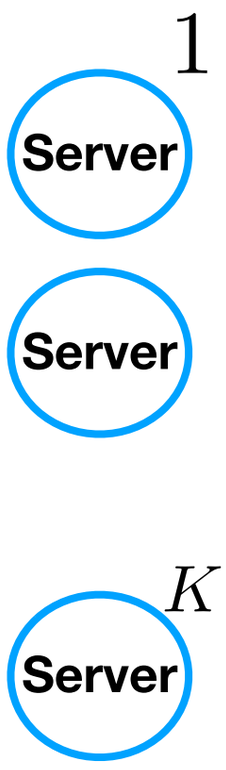


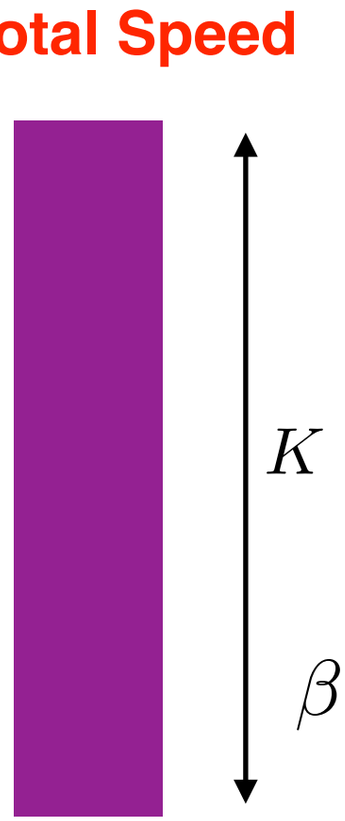
Process the β -**fraction** of the **most recently arrived** jobs

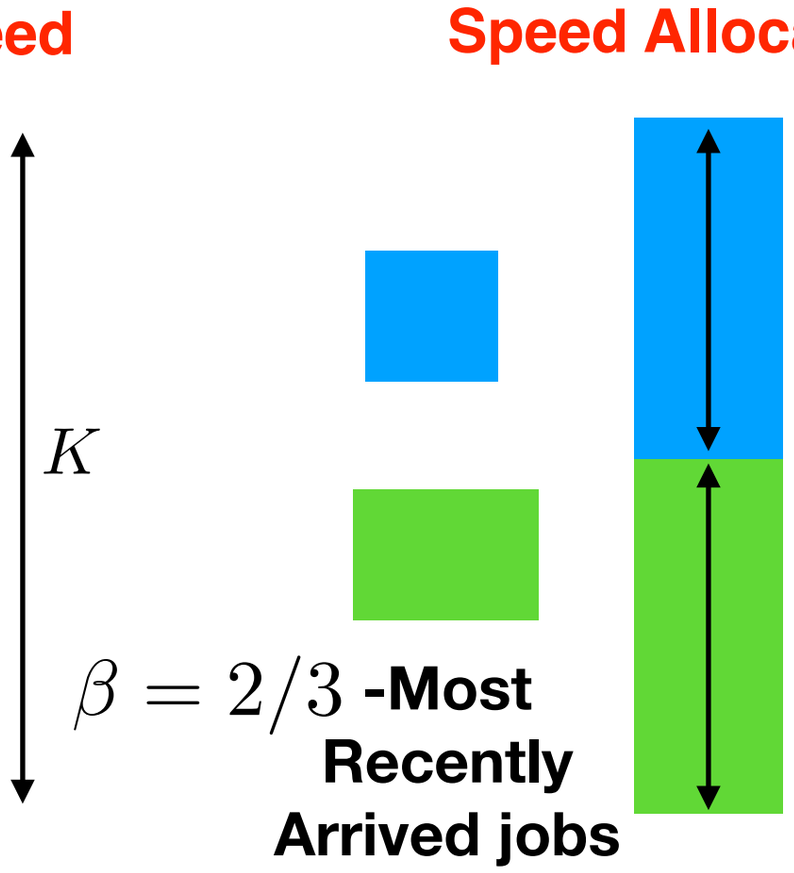
Assign **equal speed** to all jobs being processed

Our Algorithm LCFS-EQUI







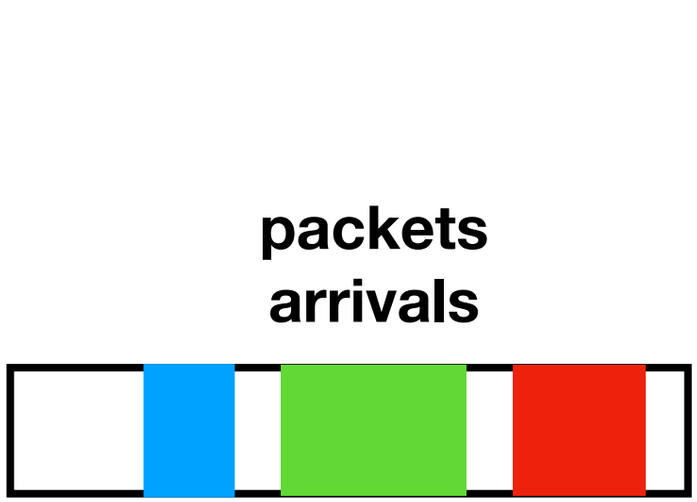


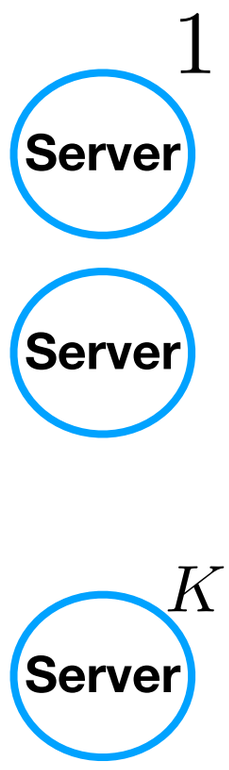
Multiple Servers

Process the β -**fraction** of the **most recently arrived** jobs

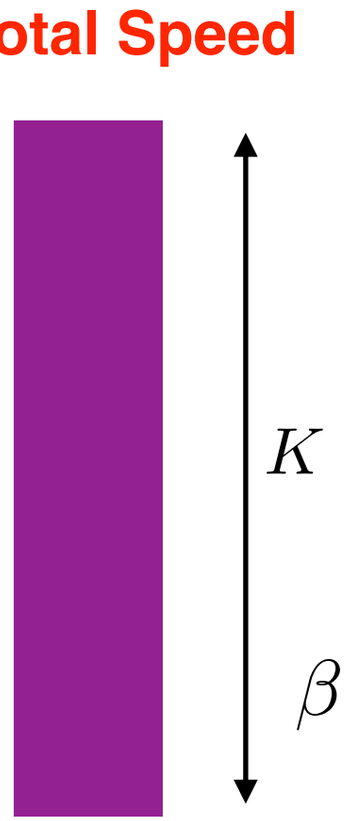
Assign **equal speed** to all jobs being processed

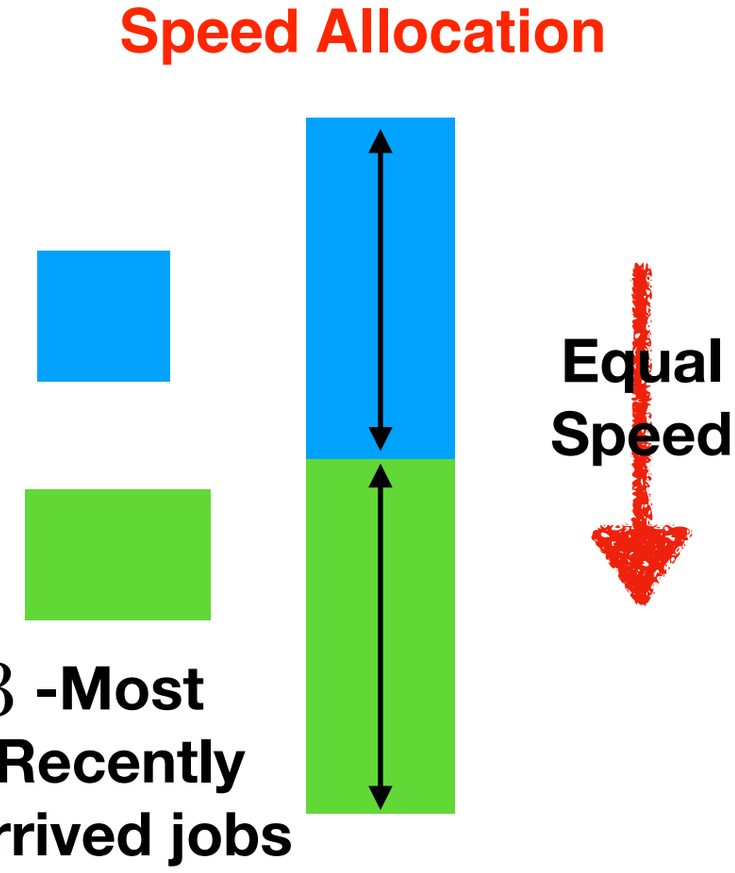
Our Algorithm LCFS-EQUI





Multiple Servers



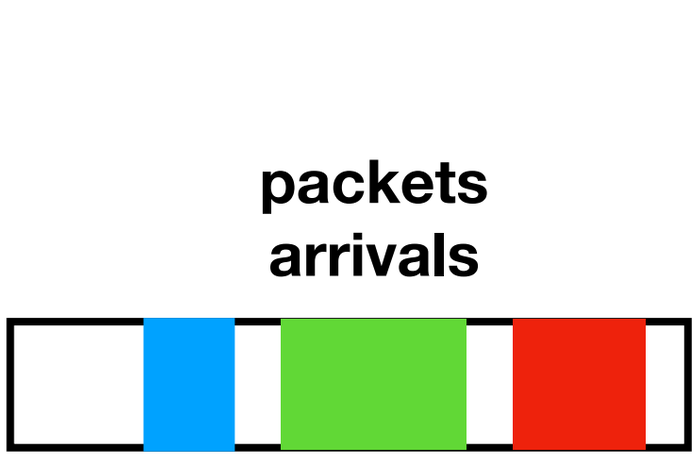


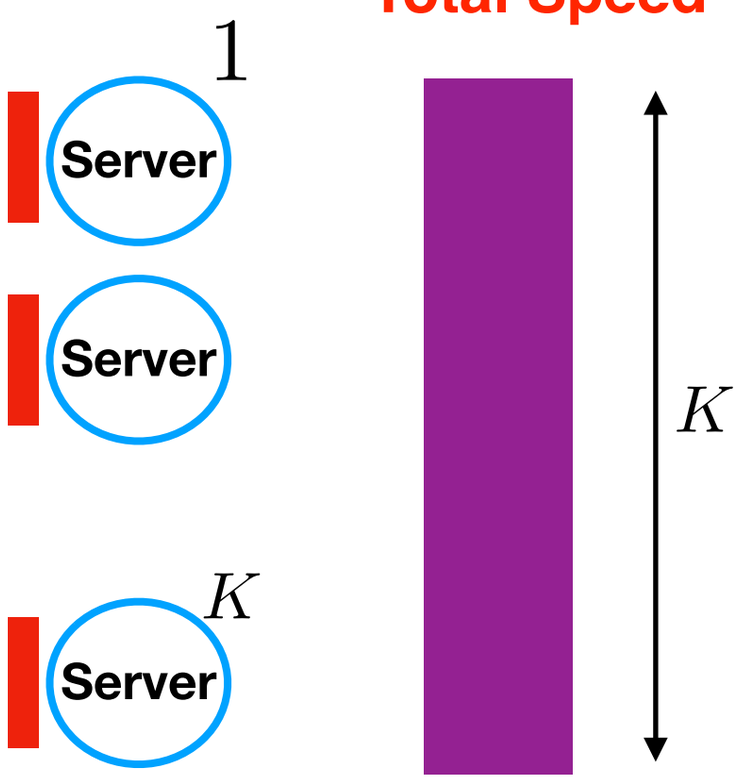
Process the β -**fraction** of the **most recently arrived** jobs

Assign **equal speed** to all jobs being processed

Algorithm is non-clairvoyant : no information needed about remaining job size

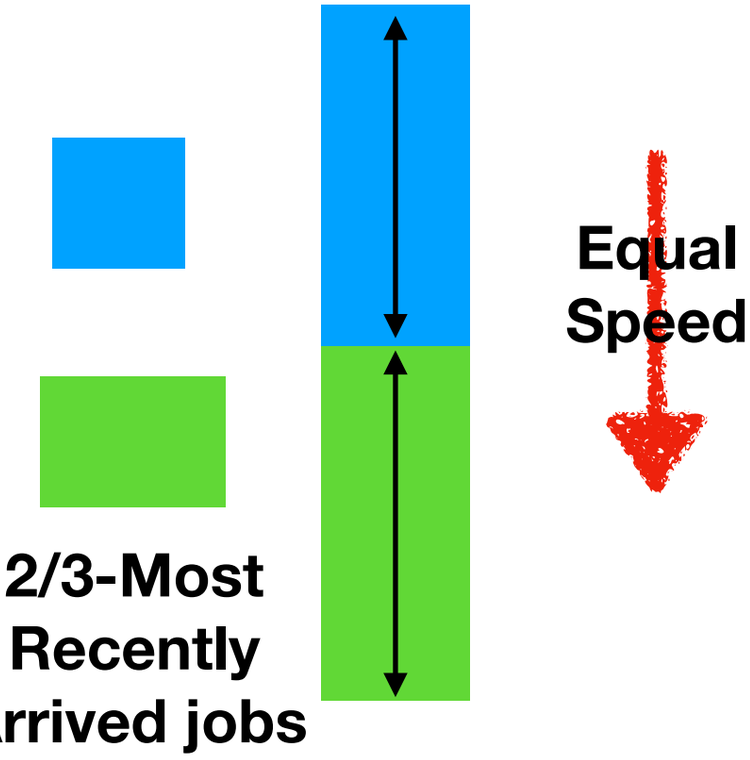
Intuition LCFS-EQUI





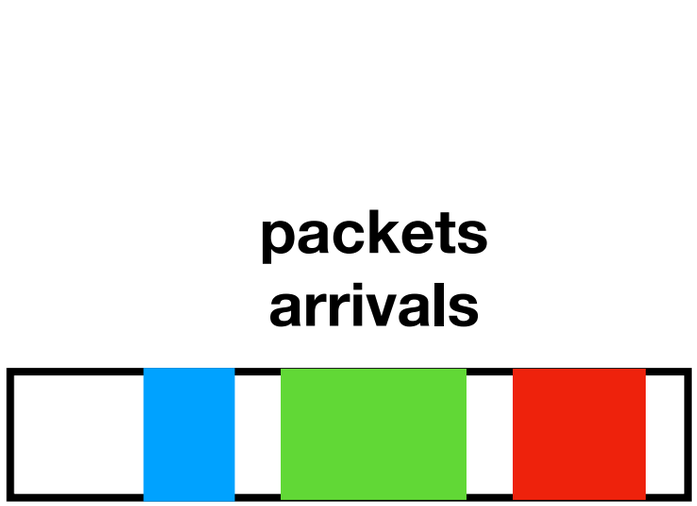
Multiple Servers

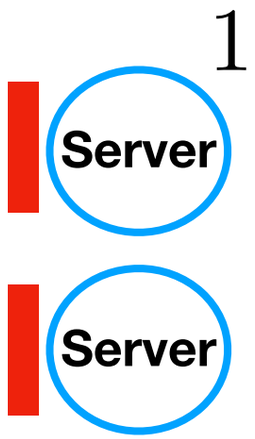
Total Speed

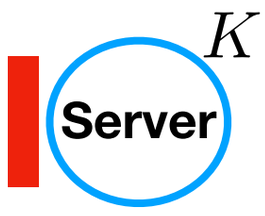


Speed Allocation

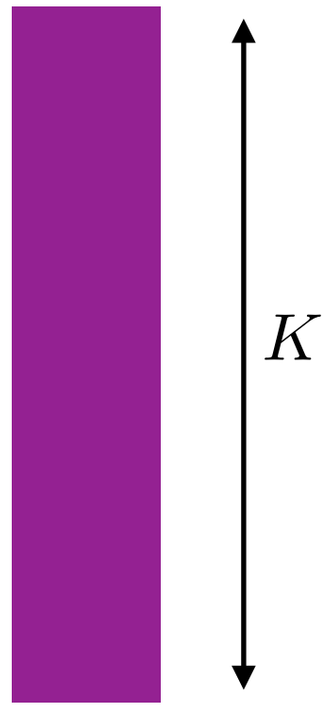
Intuition LCFS-EQUI

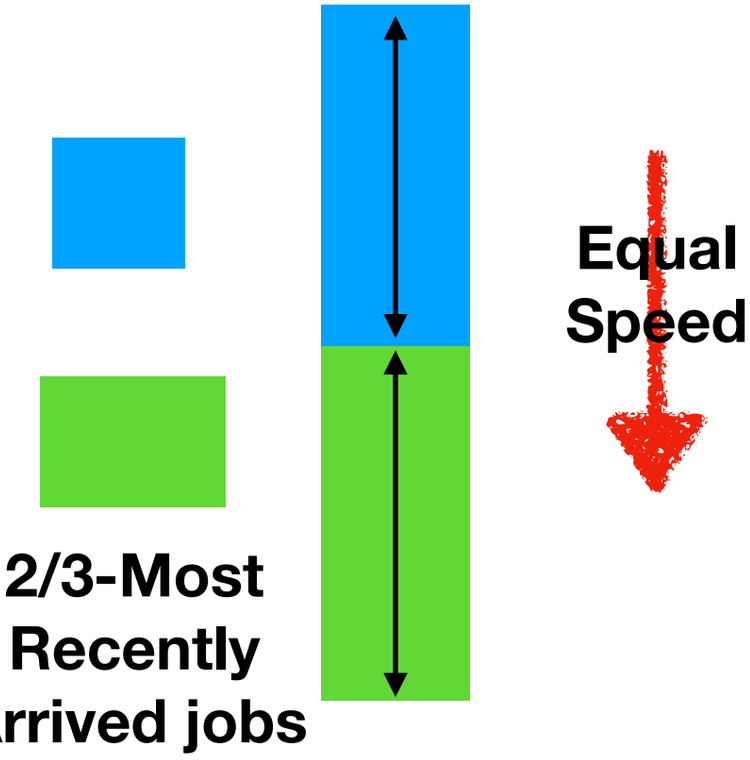






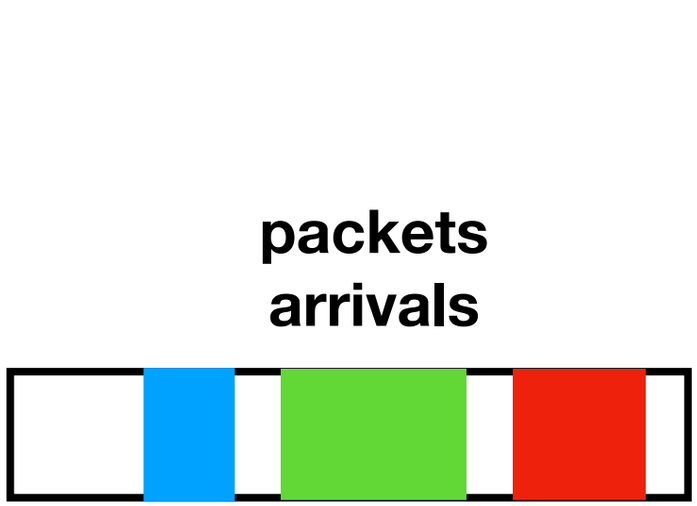
Multiple Servers

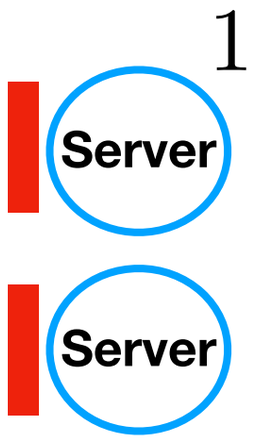


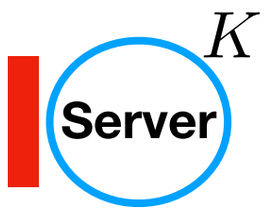


SRPT- short jobs remain in system for short time

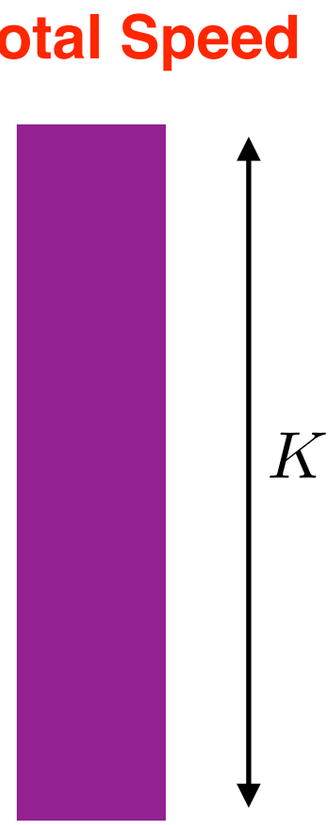
Intuition LCFS-EQUI

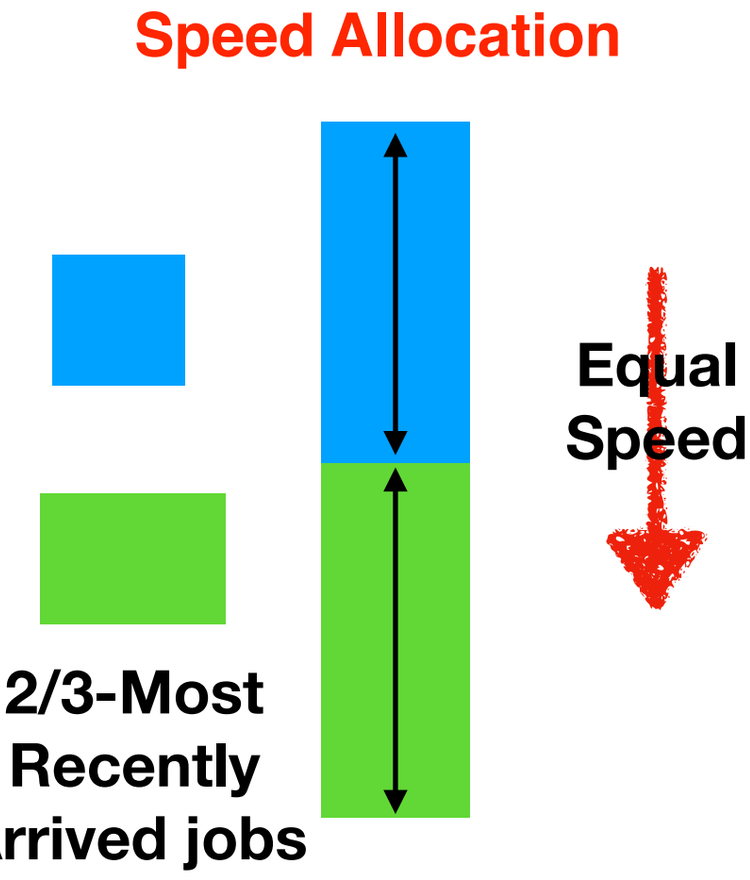






Multiple Servers

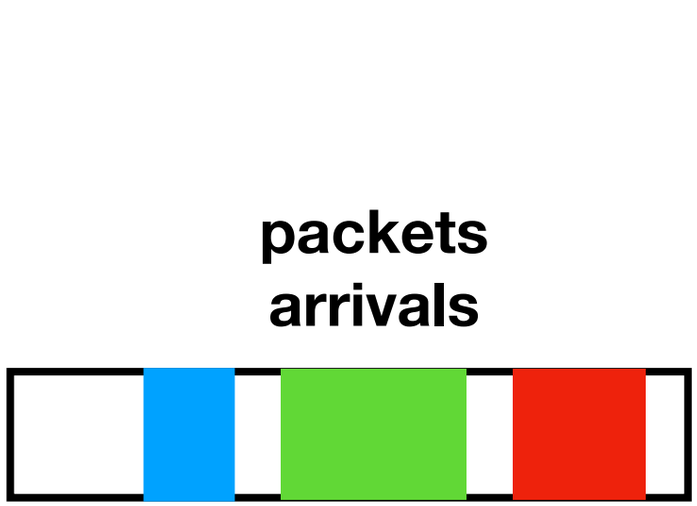


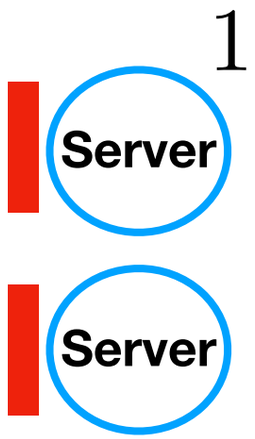


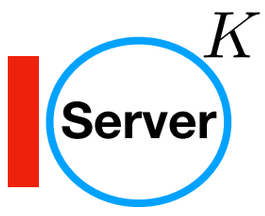
SRPT- short jobs remain in system for short time

LCFS- w/o job size information, long jobs remain in system for long

Intuition LCFS-EQUI

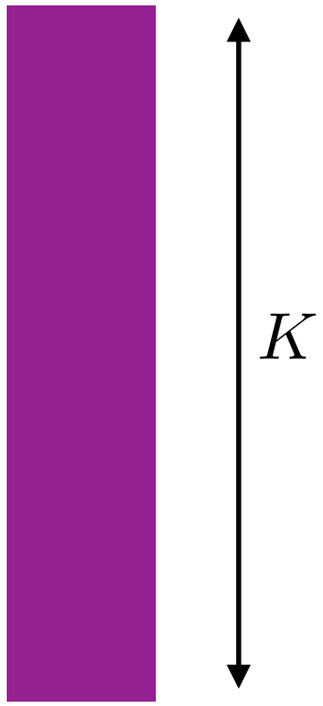




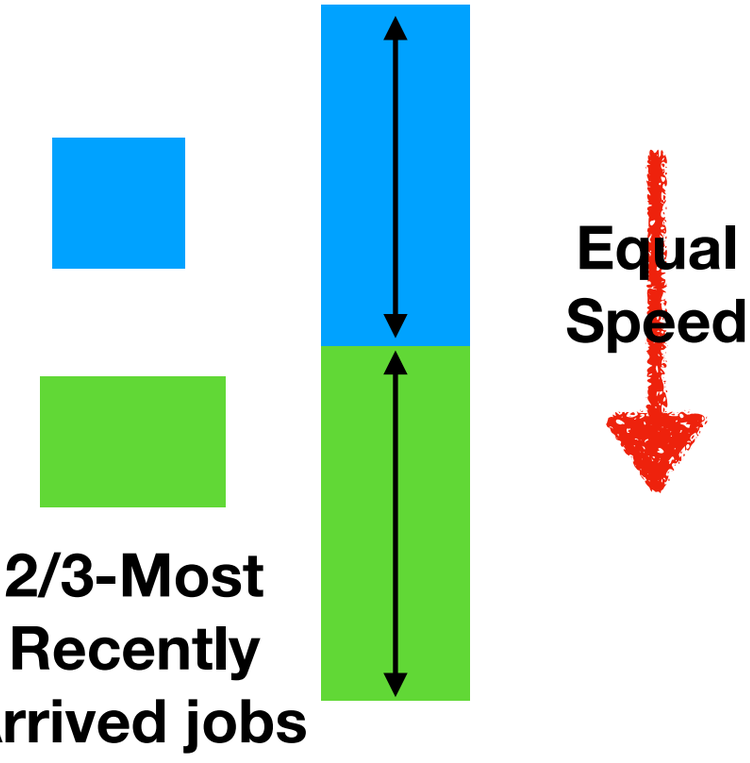


Multiple Servers

Total Speed



Speed Allocation

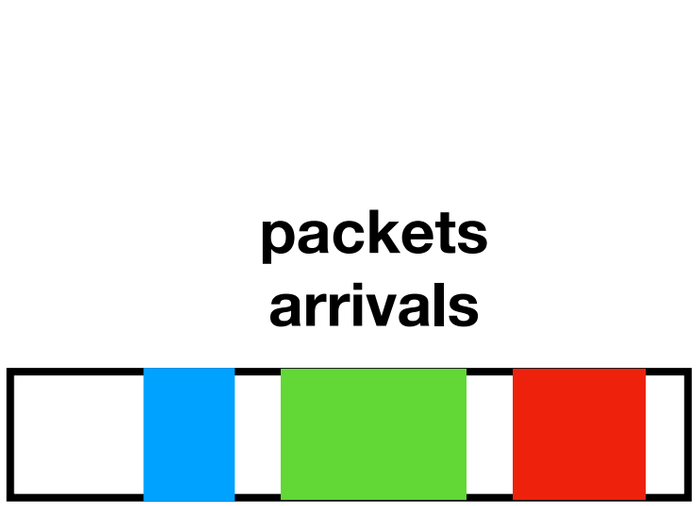


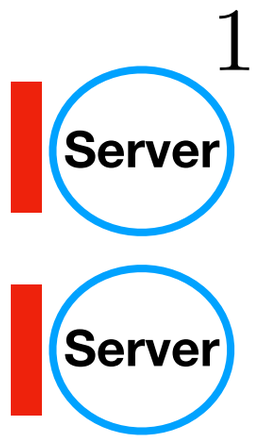
SRPT- short jobs remain in system for short time

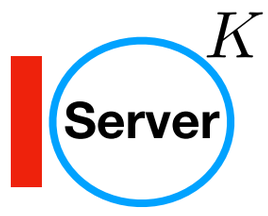
LCFS- w/o job size information, long jobs remain in system for long

Technical - construction of potential (Lyapunov) function is easy

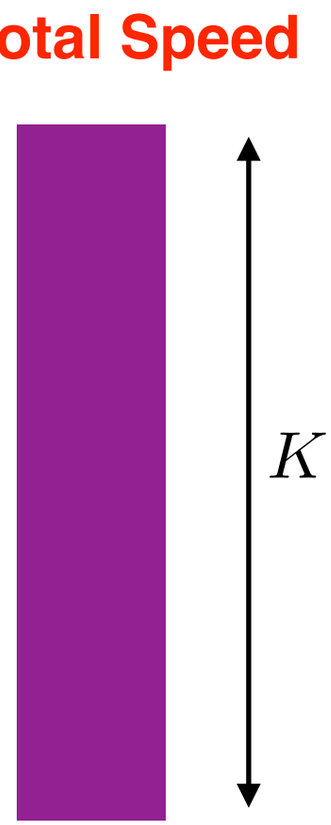
Intuition LCFS-EQUI

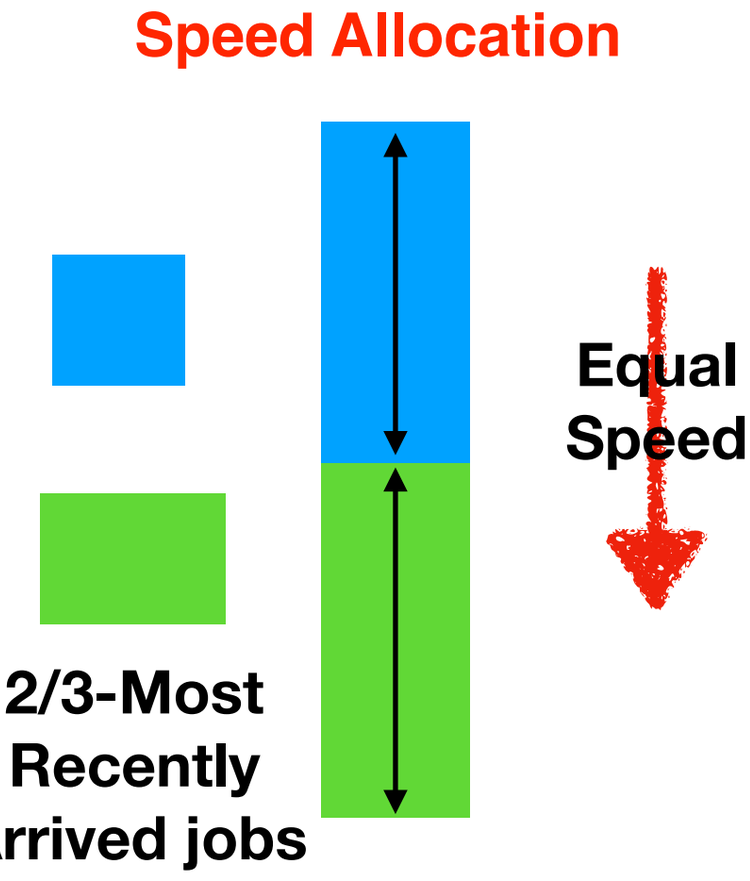






Multiple Servers





SRPT- short jobs remain in system for short time

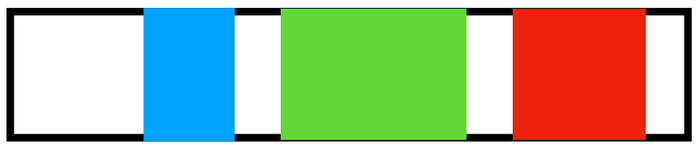
LCFS- w/o job size information, long jobs remain in system for long

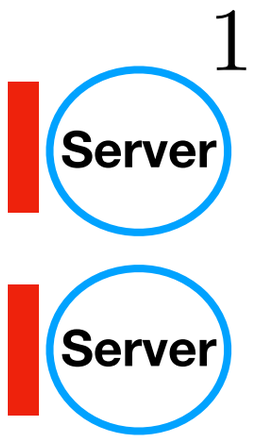
Technical - construction of potential (Lyapunov) function is easy

EQUI : with equal speed, analysis is easy

Guarantee LCFS-EQUI

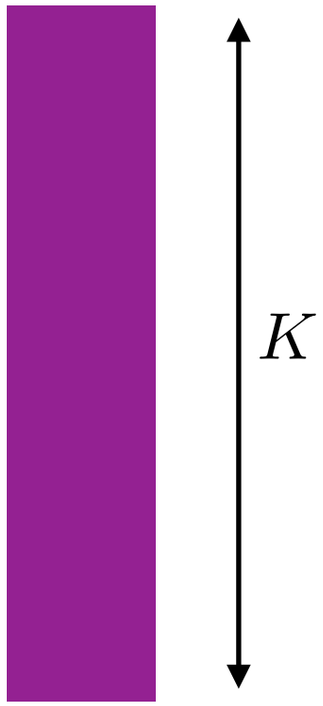
packets arrivals



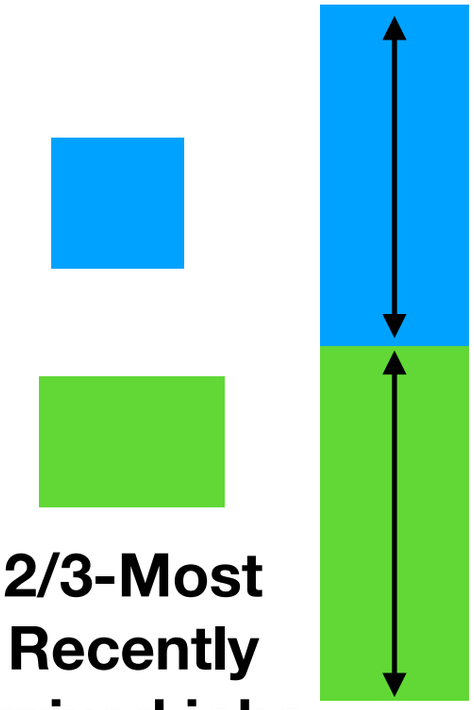


Multiple Servers

Total Speed



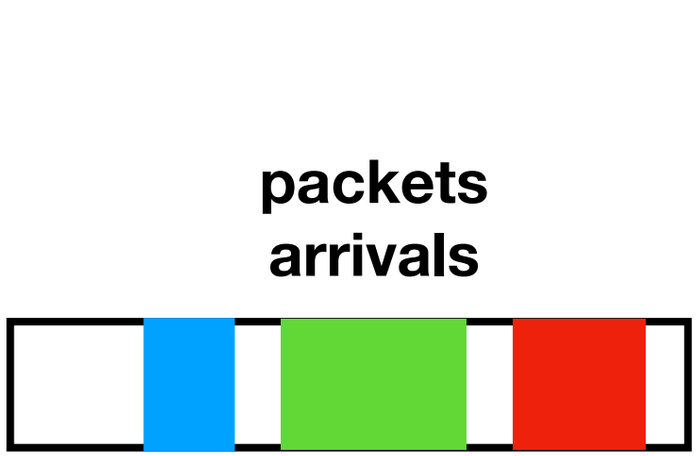
Speed Allocation

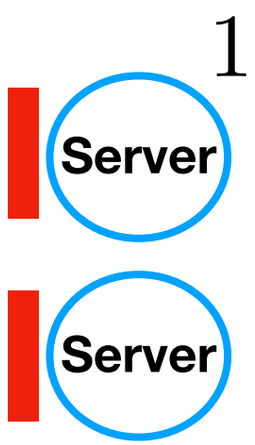


2/3-Most Recently Arrived jobs

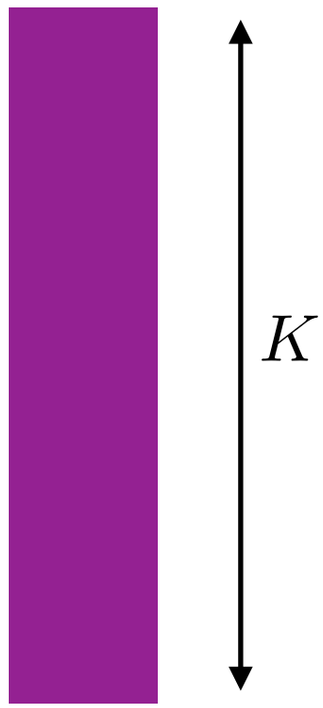
Equal Speed

Guarantee LCFS-EQUI

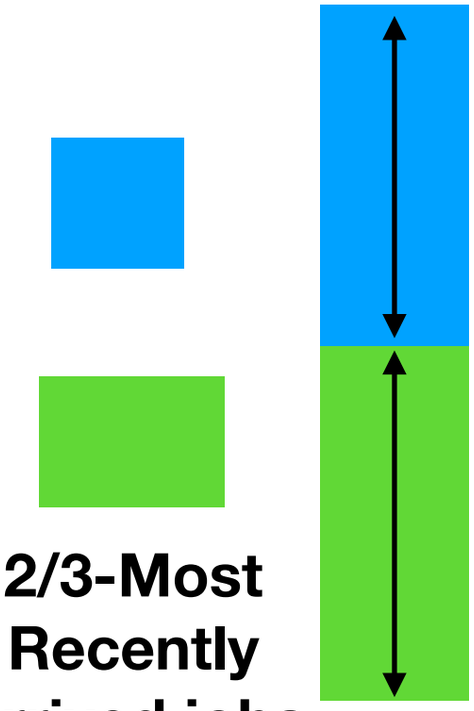




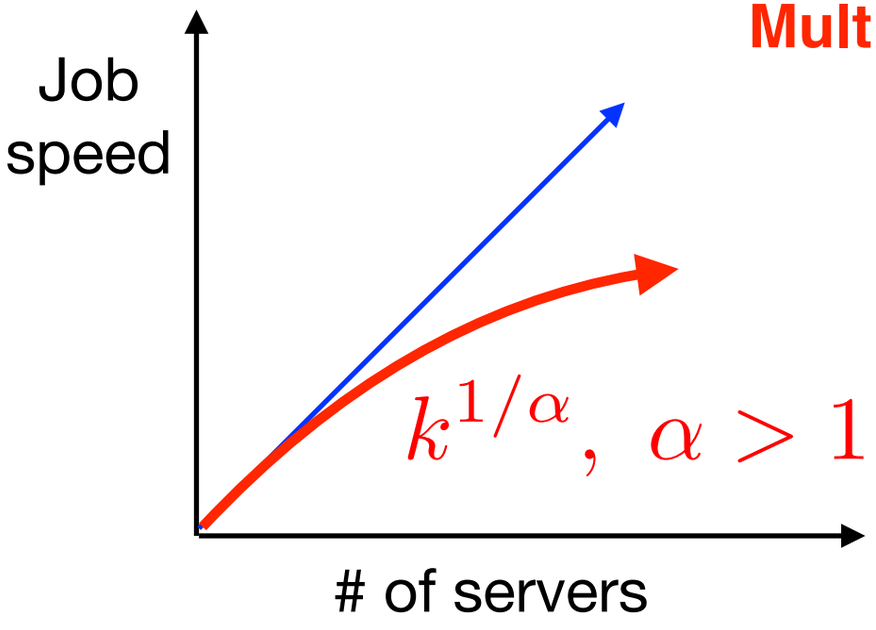
Total Speed



Speed Allocation

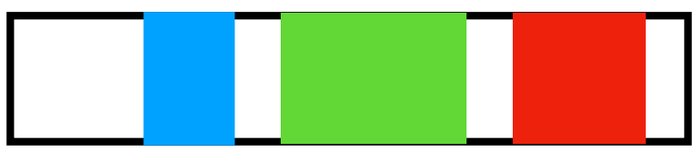


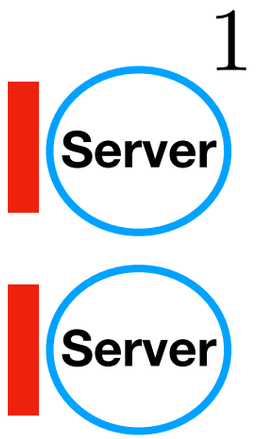
Multiple Servers



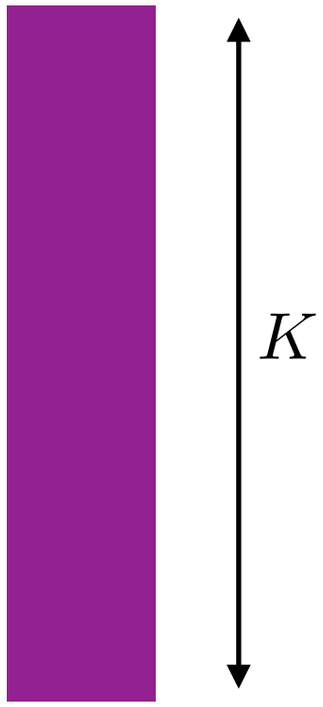
Guarantee LCFS-EQUI

packets arrivals

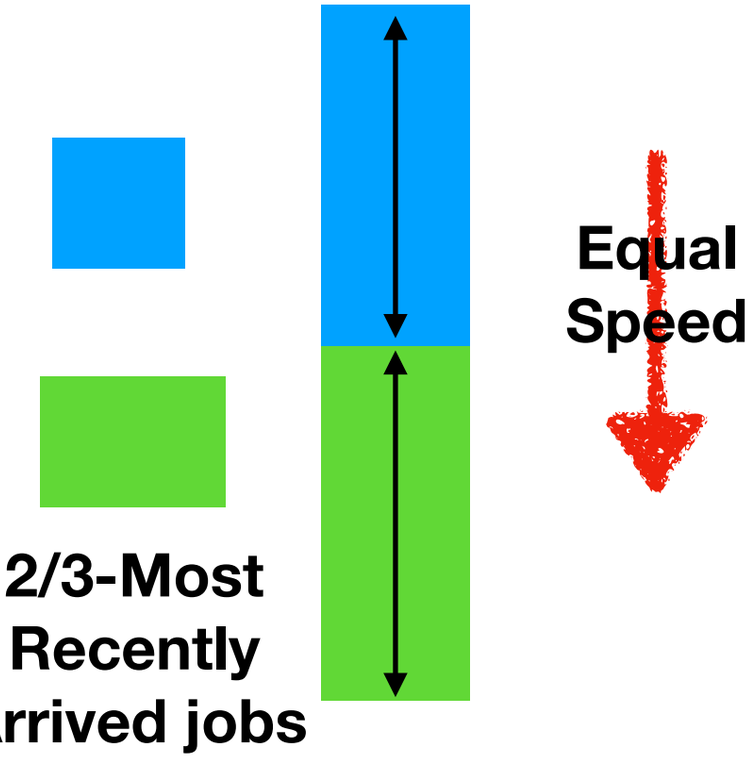




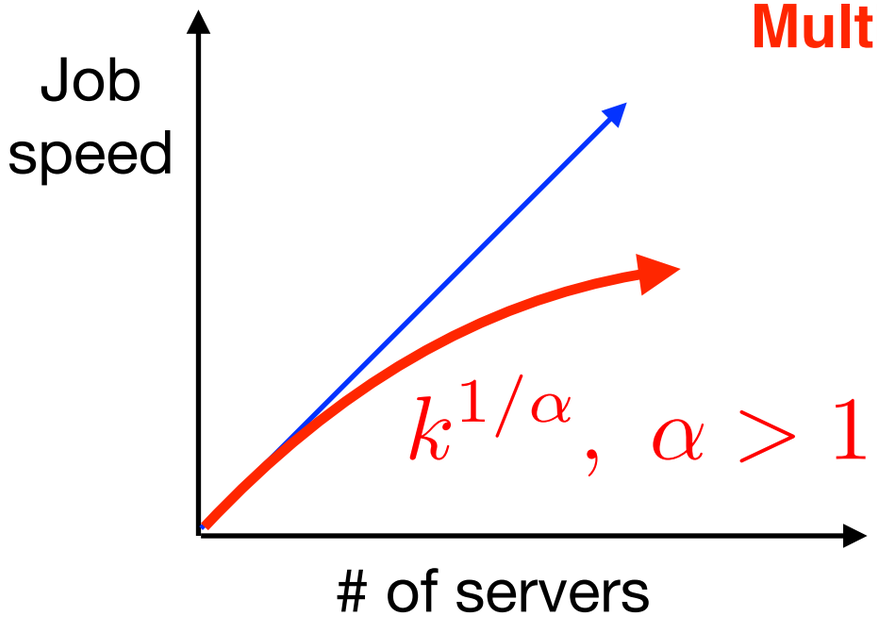
Total Speed



Speed Allocation



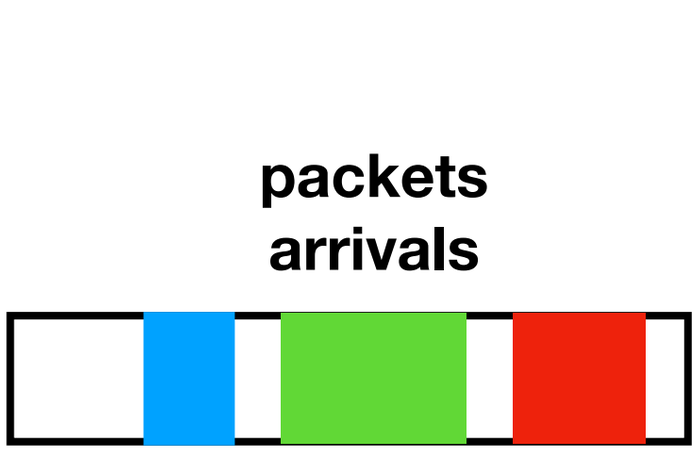
Multiple Servers

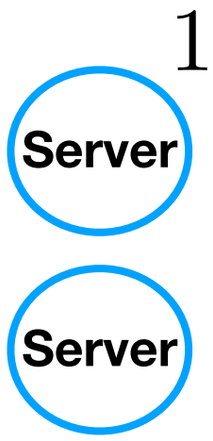


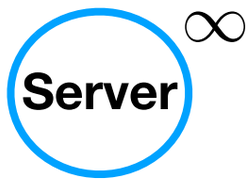
For a suitable choice of fraction β

Competitive ratio is a constant that only depends on exponent α

General Problem

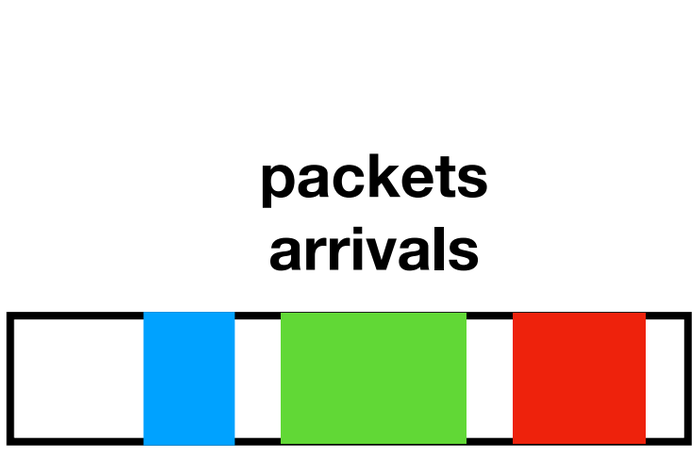


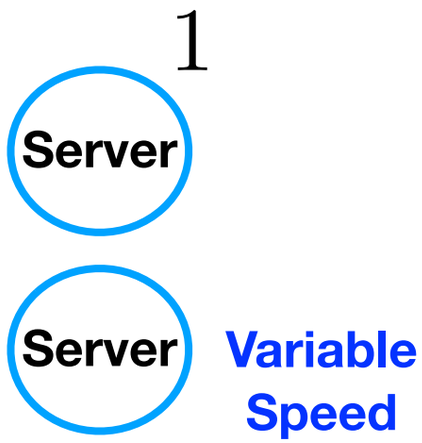


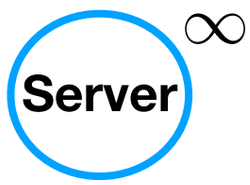


Unlimited # of Servers

General Problem

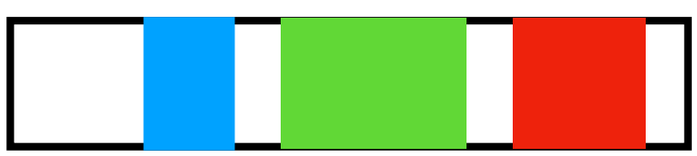


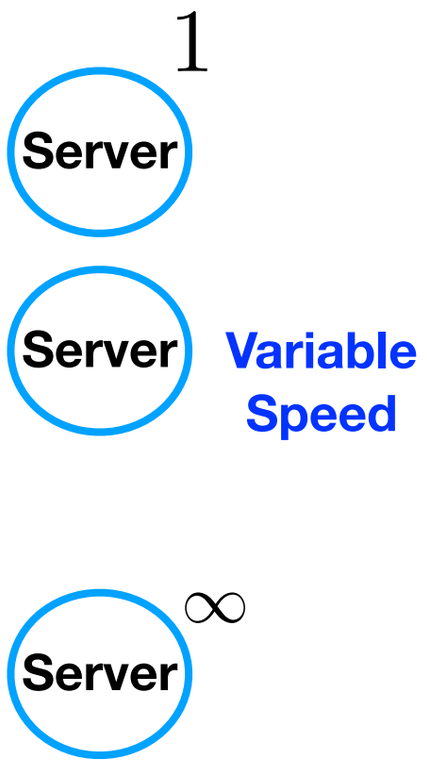




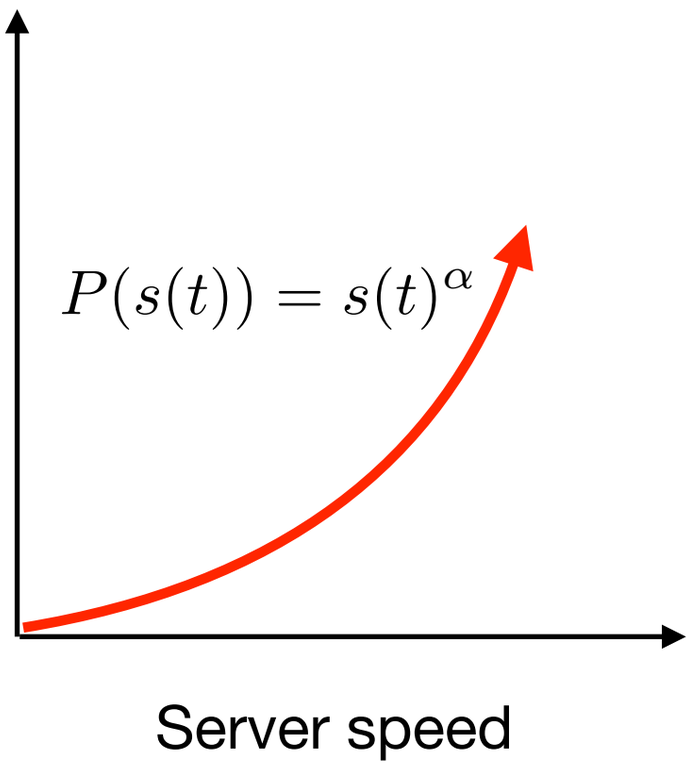
Unlimited # of Servers

General Problem

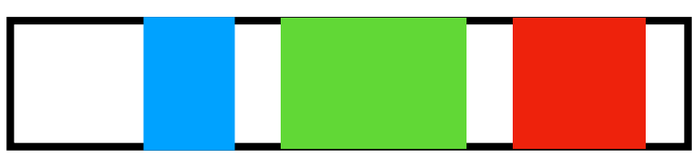


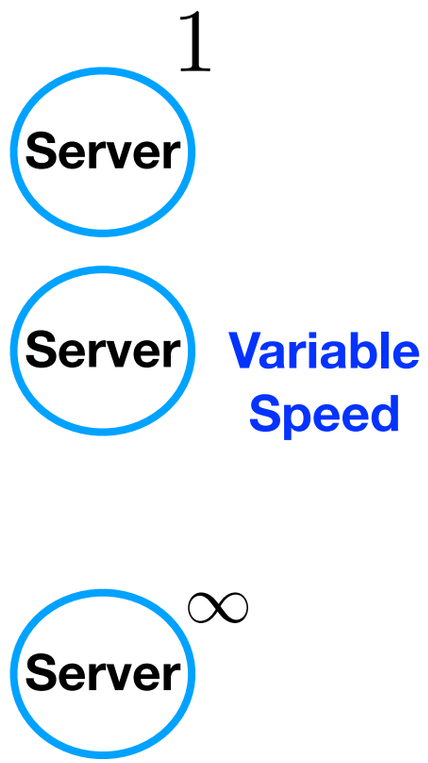


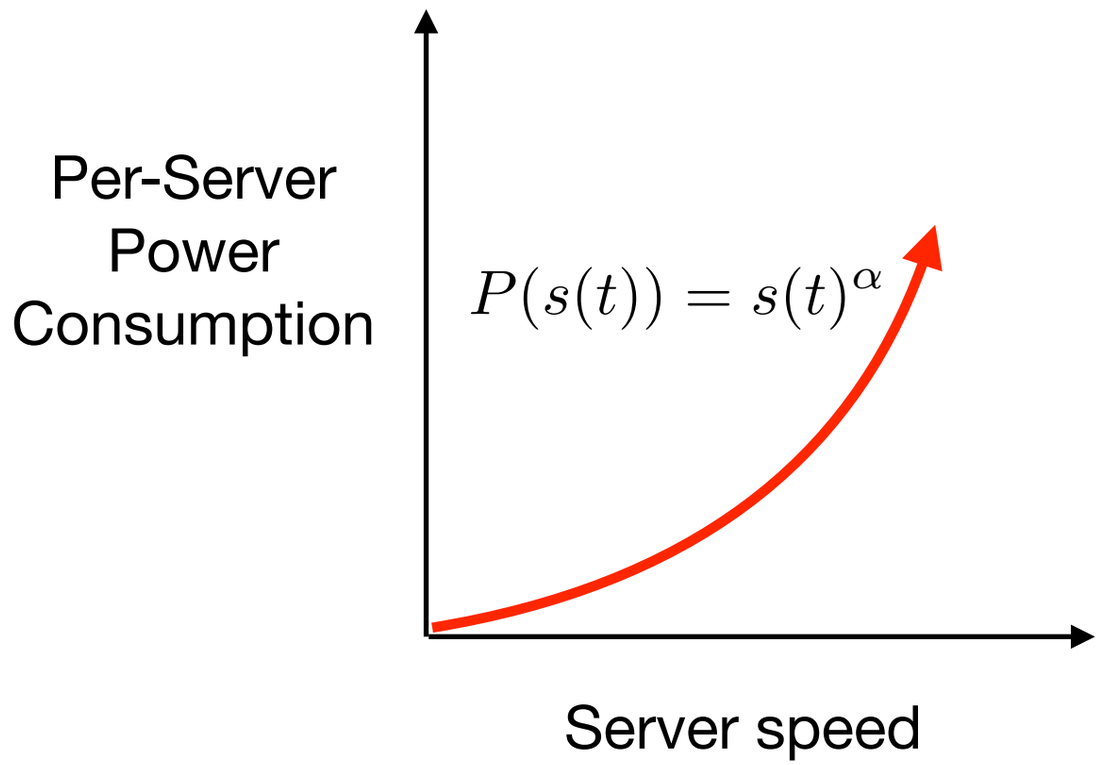
Unlimited # of Servers



General Problem







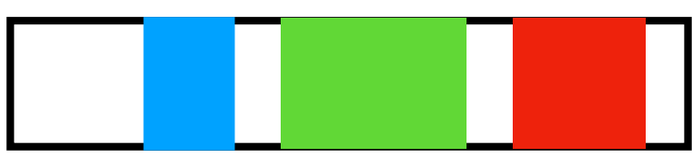
Unlimited # of Servers

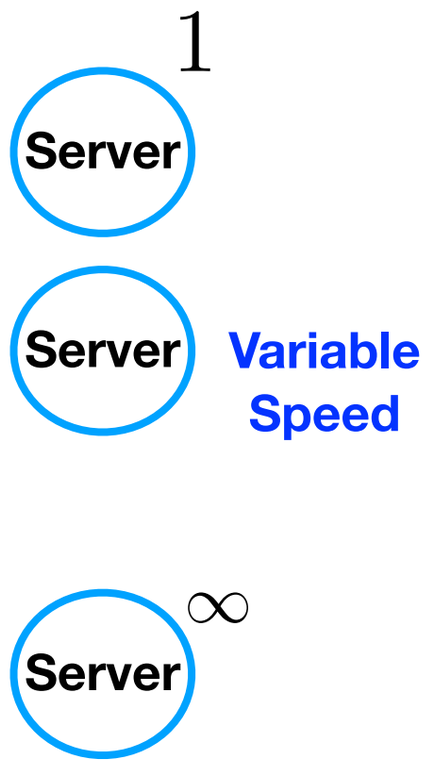
Obj: ***min*** **total flow time**

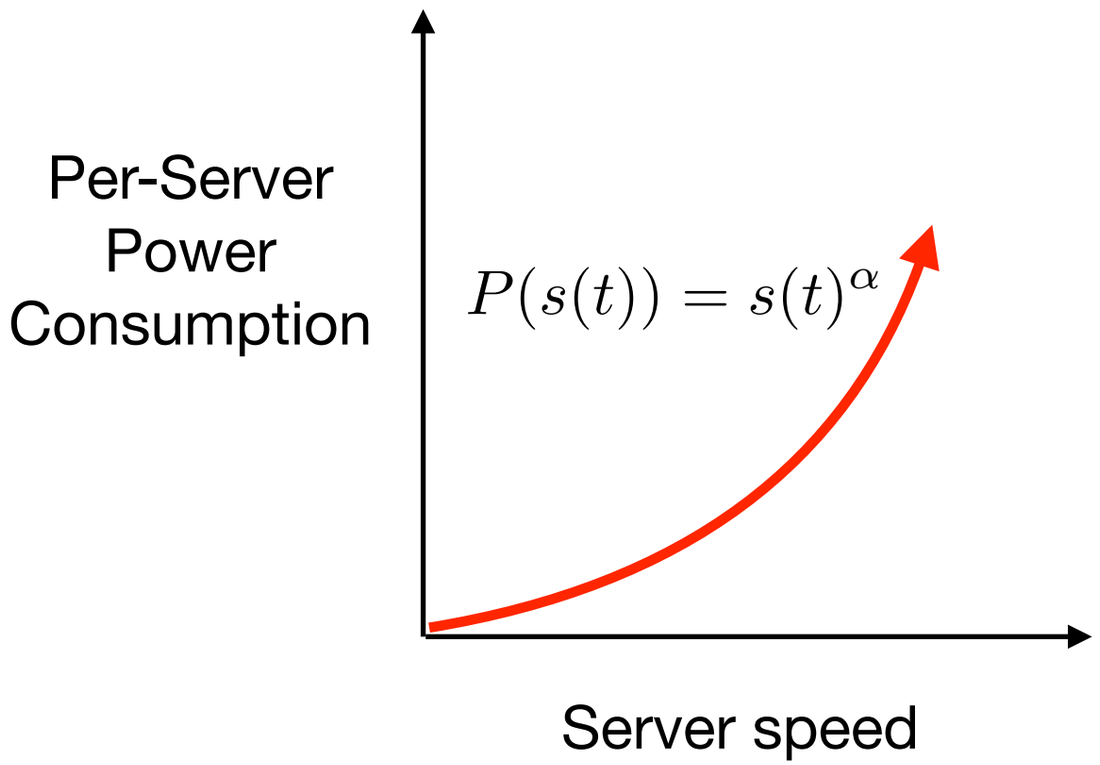
$$\int n(t)dt$$

n(t) **number of outstanding jobs at time t**

General Problem







Unlimited # of Servers

Obj: ***min*** **total flow time**

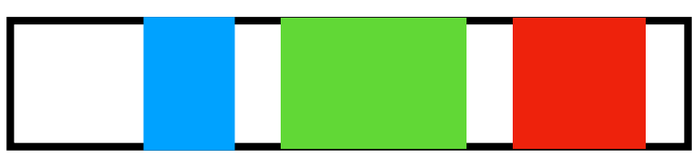
$$\int n(t)dt$$

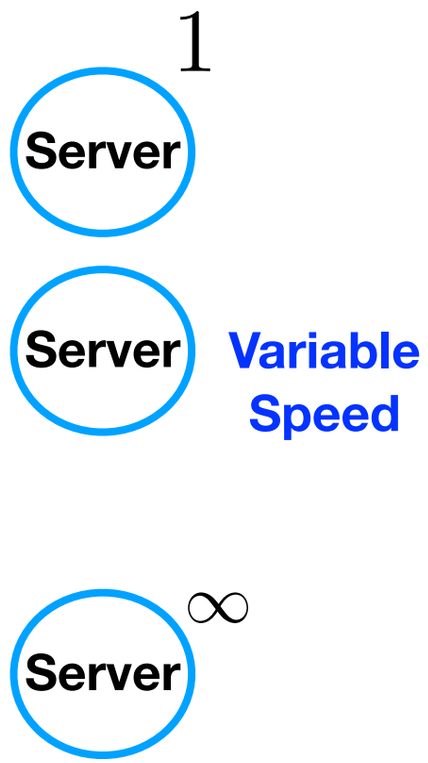
n(t) **number of outstanding jobs at time t**

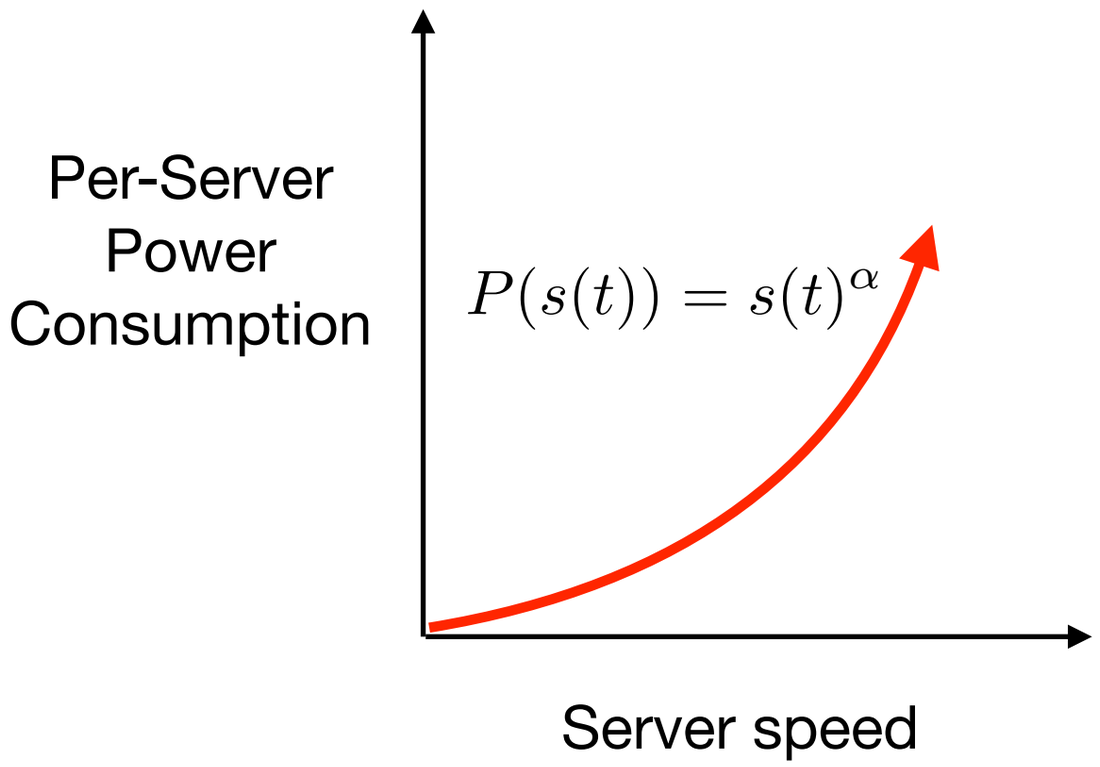
subj: sum-power constraint across all servers

$\mathcal{P}$

General Problem







Unlimited # of Servers

Obj: ***min*** **total flow time**

$$\int n(t)dt$$

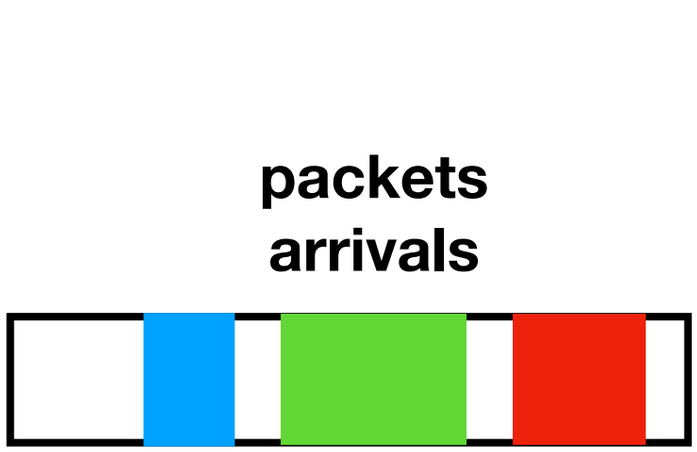
n(t) **number of outstanding jobs at time t**

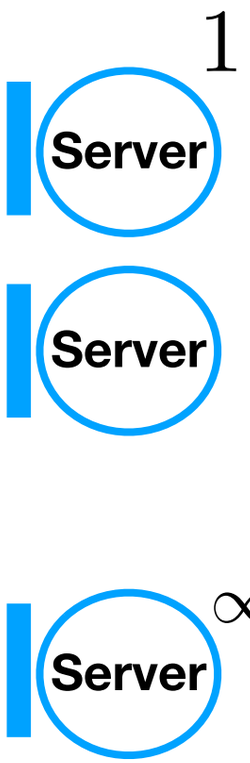
subj: sum-power constraint across all servers

$\mathcal{P}$

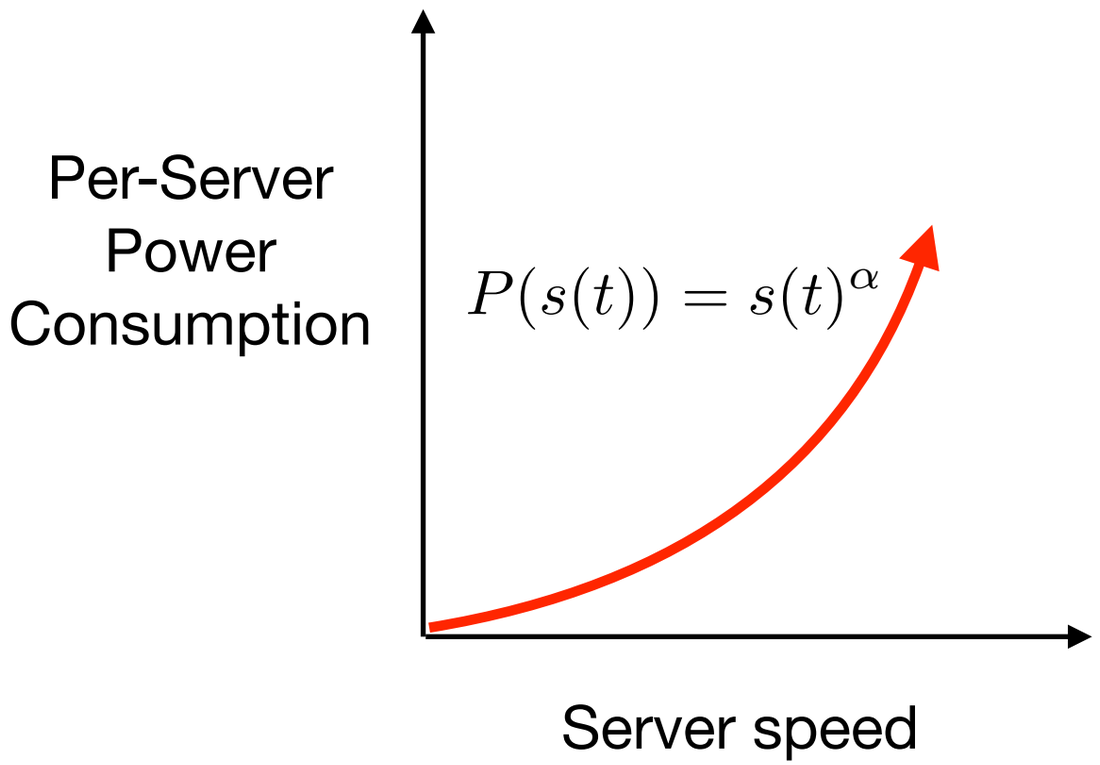
Decision : speed of each job/server

Result - General Problem

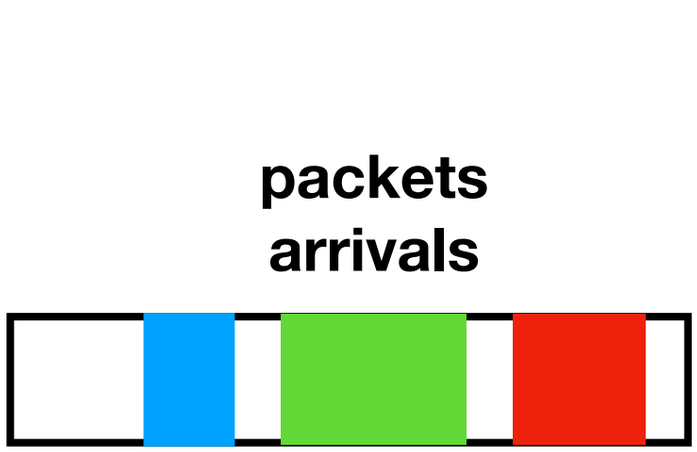


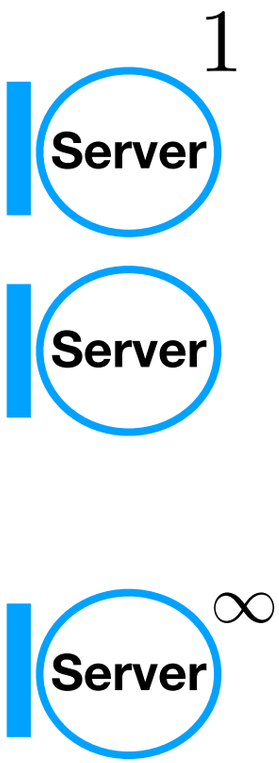


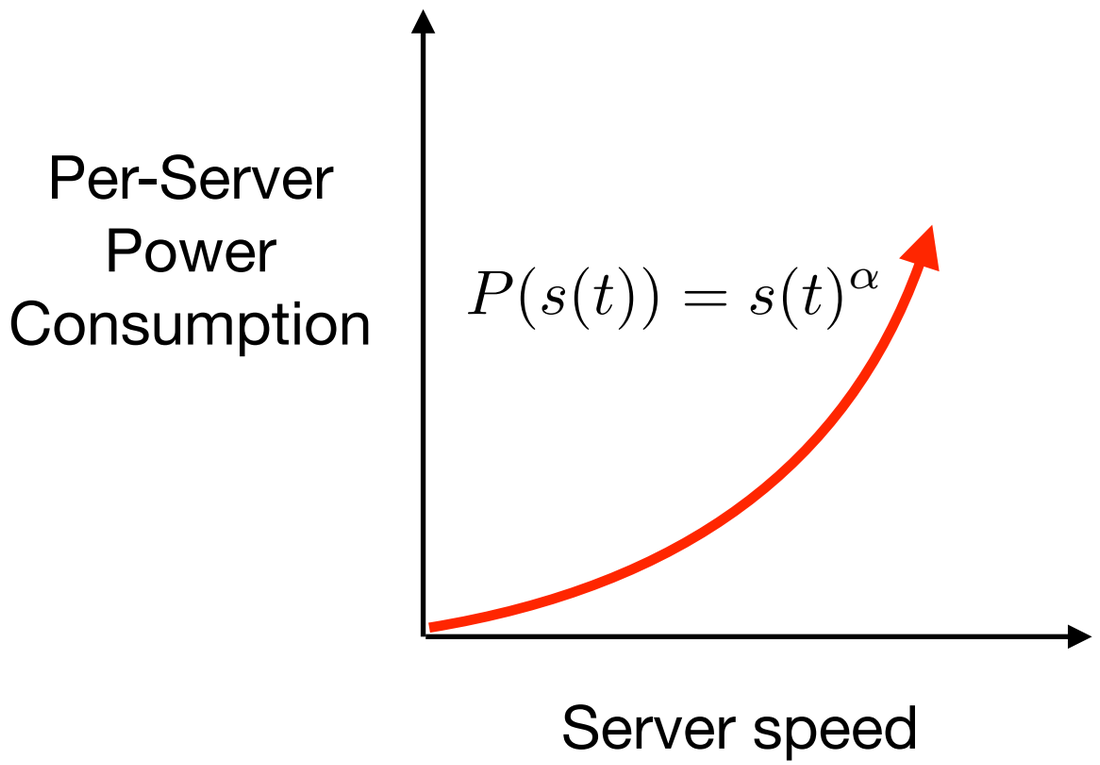
Unlimited # of Servers



Result - General Problem







Unlimited # of Servers

Competitive ratio is a constant that only depends on exponent α

Conclusions

Conclusions

An online algorithm for an important job-scheduling problem

Conclusions

An online algorithm for an important job-scheduling problem

Fractional LCFS+Equal Speed algorithm is constant competitive

Conclusions

An online algorithm for an important job-scheduling problem

Fractional LCFS+Equal Speed algorithm is constant competitive

Past approaches needed resource augmentation

Conclusions

An online algorithm for an important job-scheduling problem

Fractional LCFS+Equal Speed algorithm is constant competitive

Past approaches needed resource augmentation

Open Questions

Conclusions

An online algorithm for an important job-scheduling problem

Fractional LCFS+Equal Speed algorithm is constant competitive

Past approaches needed resource augmentation

Open Questions

Competitive ratio of heSRPT algorithm (that is locally optimal) ?

Conclusions

An online algorithm for an important job-scheduling problem

Fractional LCFS+Equal Speed algorithm is constant competitive

Past approaches needed resource augmentation

Open Questions

Competitive ratio of heSRPT algorithm (that is locally optimal) ?

Lower Bound on the competitive ratio of any online algorithm?