Load balancing, redundancy, and multi type job and server systems

Urtzi Ayesta

CNRS & Ikerbasque-Univ. Basque Country

Questions/Remarks welcome at : urtzi.ayesta@irit.fr

IFIP Performance 2020, 2/11/2020

Based on:

joint work with:

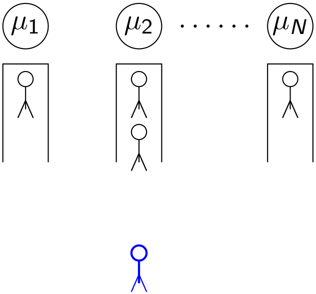
E. Anton, T. Bodas, J.L. Dorsman, M. Jonckheere, I.M. Verloop

and many other papers by:

Adan, Bonald, Busic, Comte, Gardner, Harchol-Balter, Hellemans, Hyytiä, Krzesinski, Mairesse, Moyal, Perry, Righter, Scheller Wolf, van Houdt, Visschers, Weiss

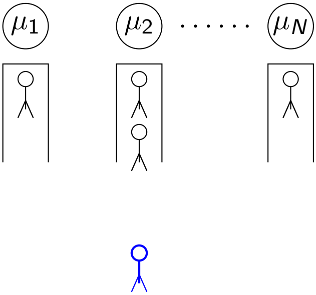
Animations by T. Bodas (Tikz !)

Load balancing



Very active research domain: JSQ, Power of d, Pull-Push based approaches, Jobs with multiple tasks, etc.

Load balancing



Very active research domain: JSQ, Power of d, Pull-Push based approaches, Jobs with multiple tasks, etc.

$\Longrightarrow$ analysis often approximate or in limiting regimes

Product forms in load balancing systems

- Markovian queues: Queues that can be modeled as a *Markov chain*

Product forms in load balancing systems

- Markovian queues: Queues that can be modeled as a *Markov chain*

  → Poisson arrival rates

  → Exponential service requirements

Product forms in load balancing systems

- Markovian queues: Queues that can be modeled as a *Markov chain*
  - → Poisson arrival rates
  - → Exponential service requirements
- Goal: Characterize *stationary distribution* of Markov chain

Product forms in load balancing systems

- Markovian queues: Queues that can be modeled as a *Markov chain*

  → Poisson arrival rates

  → Exponential service requirements

- Goal: Characterize *stationary distribution* of Markov chain

Product form

Product forms in load balancing systems

- Markovian queues: Queues that can be modeled as a *Markov chain*
  - → Poisson arrival rates
  - → Exponential service requirements
- Goal: Characterize $\underbrace{\textit{stationary distribution}}$ of Markov chain

Product form $\Rightarrow$ $\text{term}_1\ x\ \text{term}_2\ x \ldots\ \text{term}_n$

Product forms in load balancing systems

- Markovian queues: Queues that can be modeled as a *Markov chain*
  - Poisson arrival rates
  - Exponential service requirements
- Goal: Characterize $\underbrace{\textit{stationary distribution}}_{\textbf{Product form}}$ of Markov chain

Jackson's 1963 paper on product form queues was considered among Ten Most Influential Titles of Management Sciences First Fifty Years

Product forms in load balancing systems

- Markovian queues: Queues that can be modeled as a *Markov chain*

  → Poisson arrival rates

  → Exponential service requirements

- Goal: Characterize $\underbrace{\textit{stationary distribution}}_{\textbf{Product form}}$ of Markov chain

Jackson's 1963 paper on product form queues was considered among Ten Most Influential Titles of Management Sciences First Fifty Years

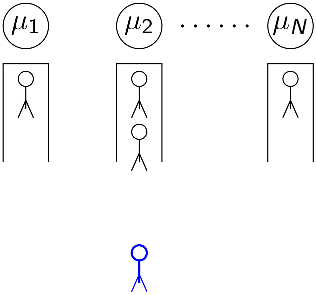
- Key Idea: Relate load balancing systems to a central queue architecture

Outline

- Redundancy
- Central Queue Architecture
- Order Independent Descriptor
  - Redundancy and cancel on complete
- Aggregated State Descriptor
  - Redundancy and cancel on start
- Generalizations:
  - Token-based framework
  - Generalized Order Independent
- Impact of assumptions: scheduling and independence

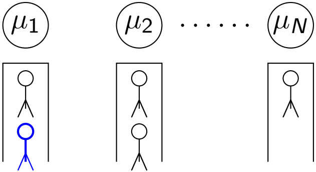
What is redundancy ?

Central idea: create several copies of the same job and use them to minimize latency !



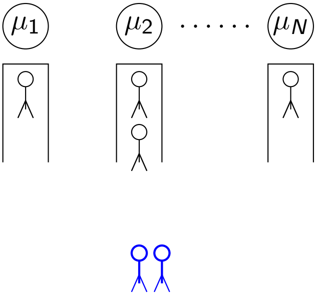
What is redundancy ?

Central idea: create several copies of the same job and use them to minimize latency !



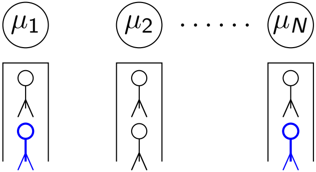
What is redundancy ?

Central idea: create several copies of the same job and use them to minimize latency !



What is redundancy ?

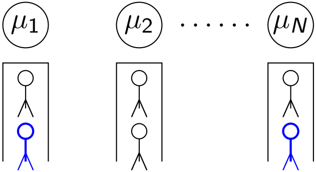
Central idea: create several copies of the same job and use them to minimize latency !



In practice deployed in DNS queries, search engines, Youtube, Mapreduce

What is redundancy ?

Central idea: create several copies of the same job and use them to minimize latency !

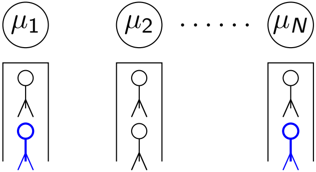


Exploit variability in the workload in different queues !

In practice deployed in DNS queries, search engines, Youtube, Mapreduce

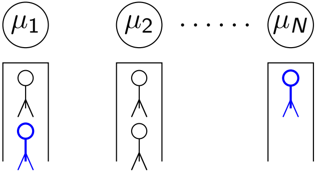
When to cancel the redundant copy ?

A supermarket example



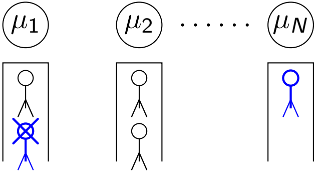
When to cancel the redundant copy ?

A supermarket example



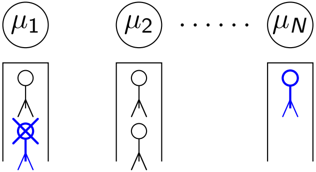
When to cancel the redundant copy ?

A supermarket example



When to cancel the redundant copy ?

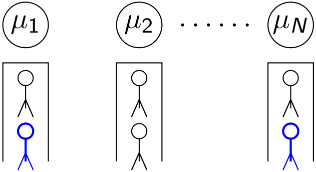
A supermarket example



Cancel on start of service ($c.o.s.$ model)

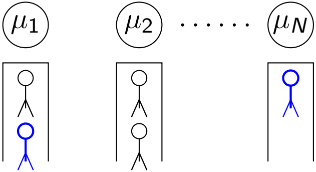
When to cancel the redundant copy ?

The cancel-on-complete variant



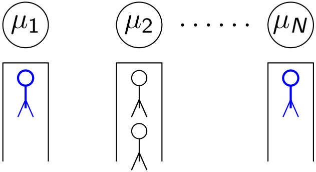
When to cancel the redundant copy ?

The cancel-on-complete variant



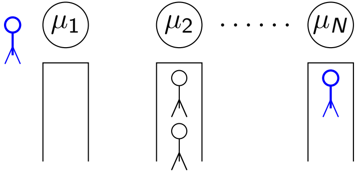
When to cancel the redundant copy ?

The cancel-on-complete variant



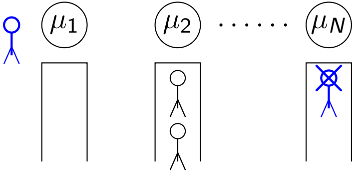
When to cancel the redundant copy ?

The cancel-on-complete variant



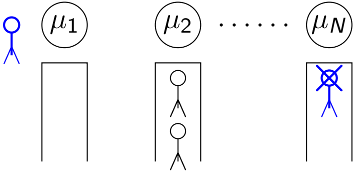
When to cancel the redundant copy ?

The cancel-on-complete variant



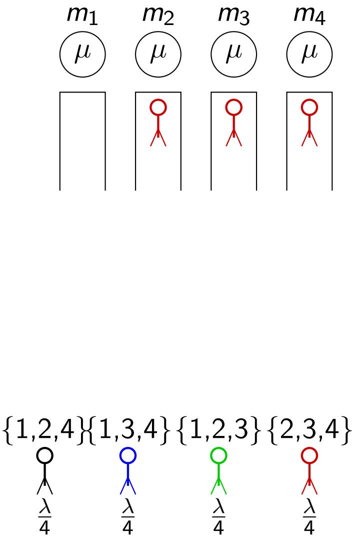
When to cancel the redundant copy ?

The cancel-on-complete variant

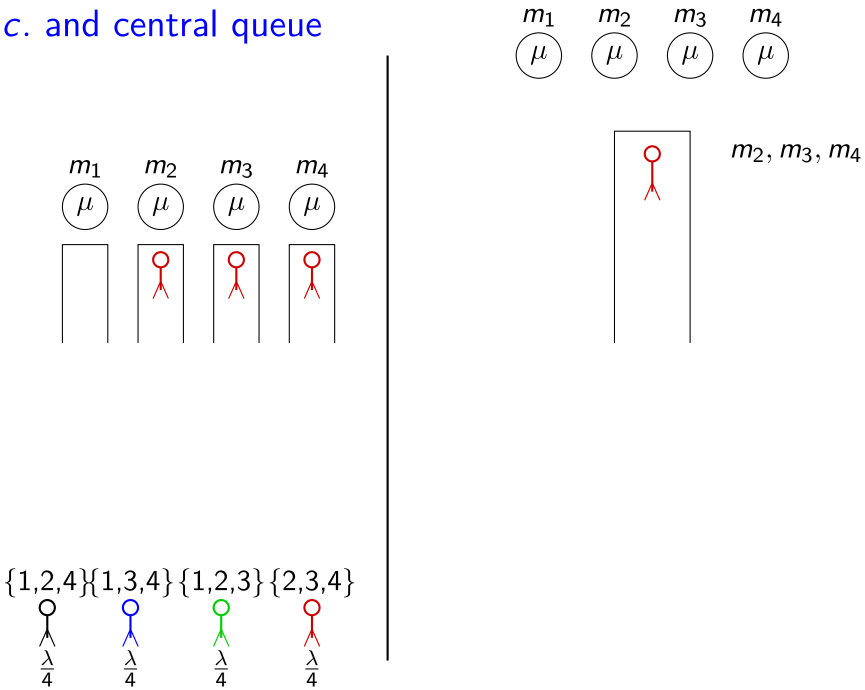


Cancel on completion of service (*c.o.c.* model)

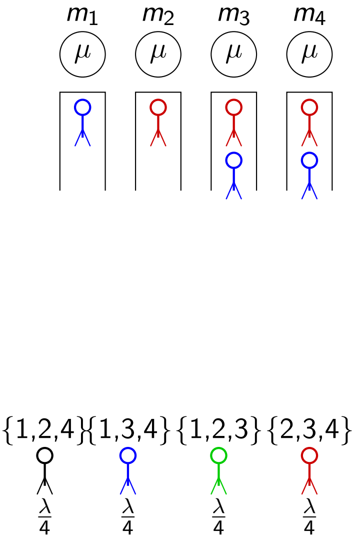
c.o.c. and central queue

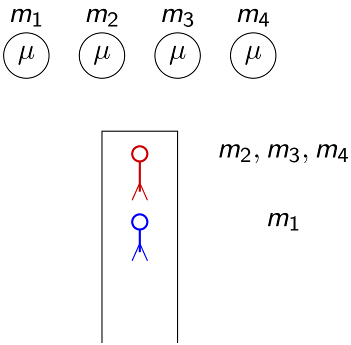


$c.o.c.$ and central queue

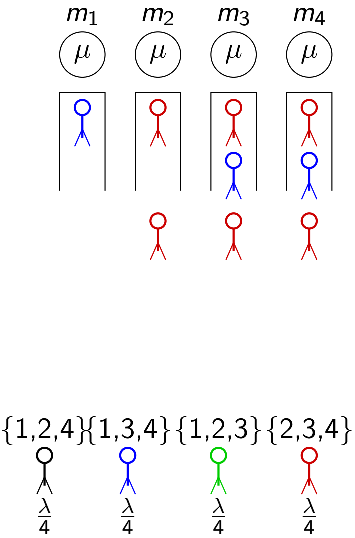


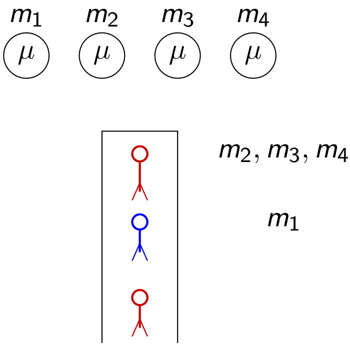
c.o.c. and central queue



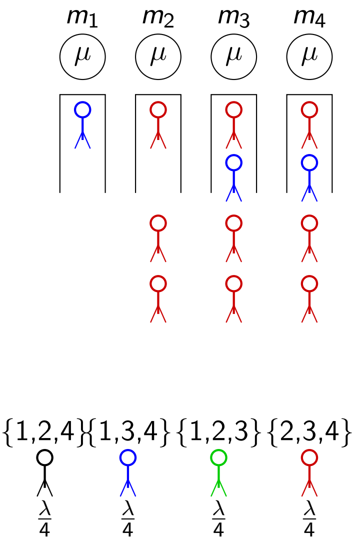


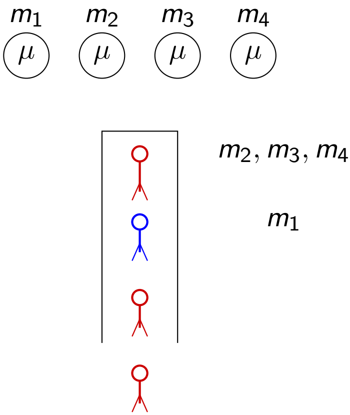
$c.o.c.$ and central queue



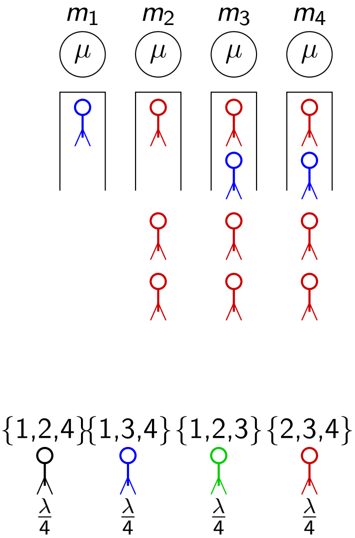


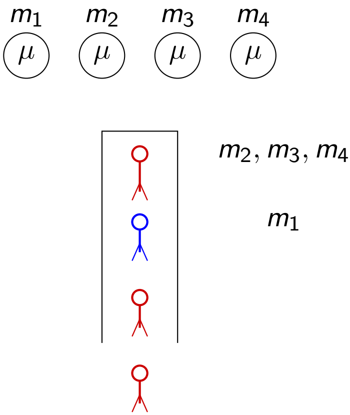
c.o.c. and central queue





$c.o.c.$ and central queue

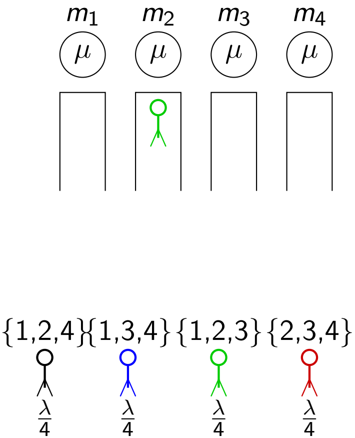




Gardner et al. QUESTA 2016

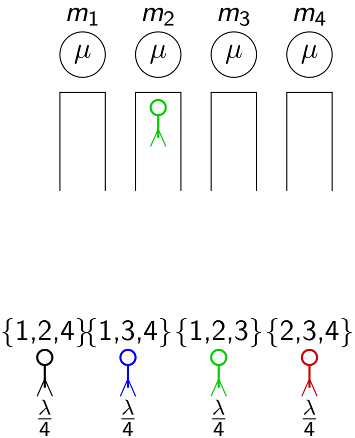
$c.o.s.$ and central queue

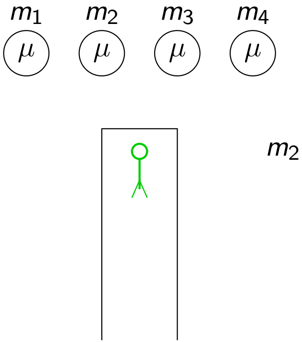
$N = 4$ and $d = 3$



c.o.s. and central queue

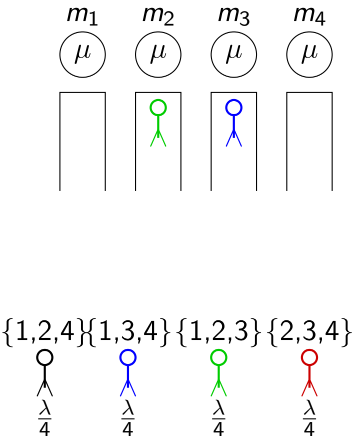
$N = 4$ *and* $d = 3$

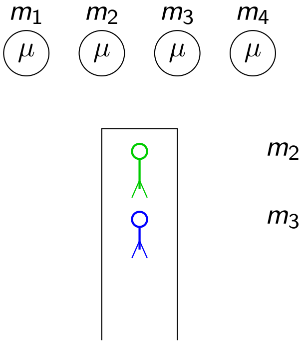




c.o.s. and central queue

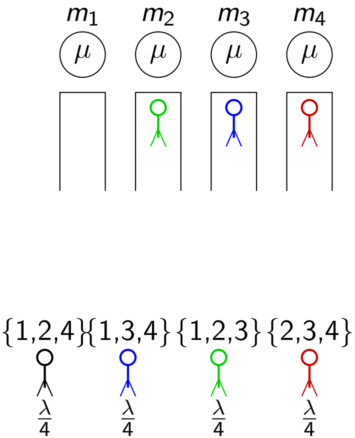
$N = 4$ *and* $d = 3$

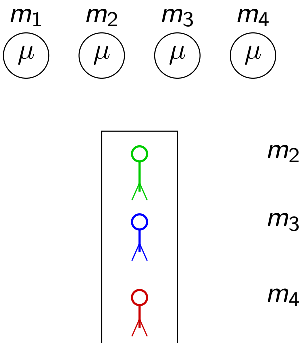




c.o.s. and central queue

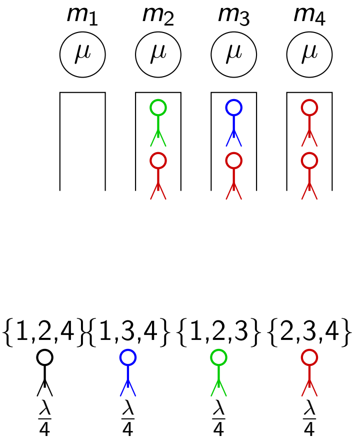
$N = 4$ *and* $d = 3$

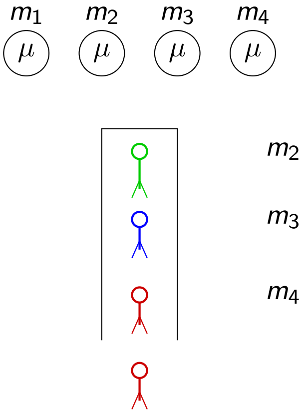




c.o.s. and central queue

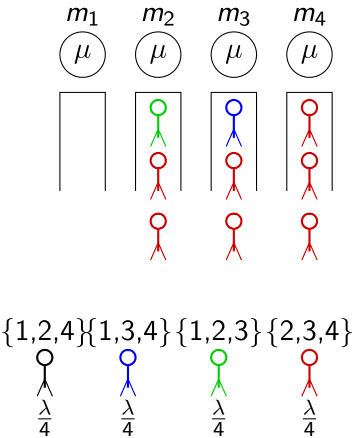
$N = 4$ *and* $d = 3$

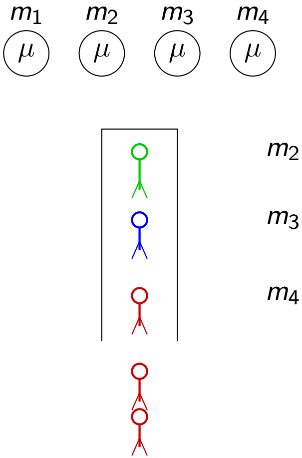




$c.o.s.$ and central queue

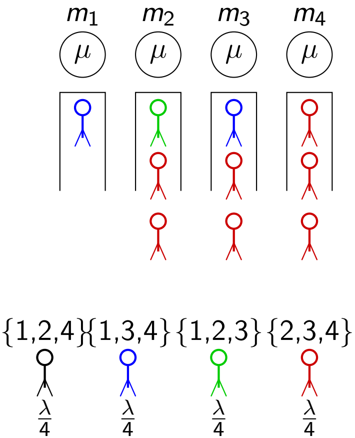
$N = 4$ *and* $d = 3$

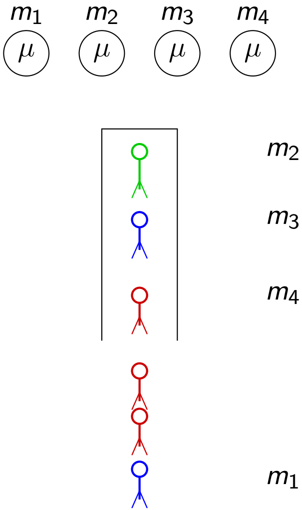




c.o.s. and central queue

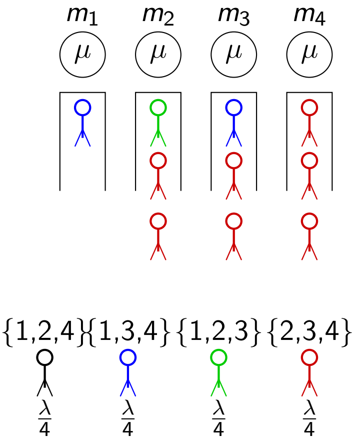
$N = 4$ *and* $d = 3$

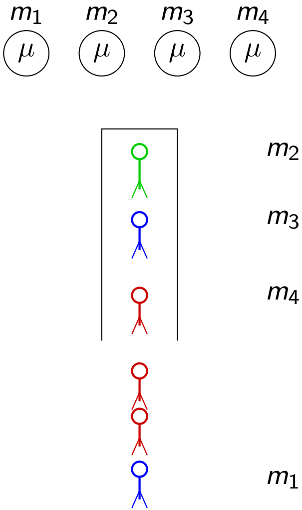




c.o.s. and central queue

$N = 4$ *and* $d = 3$

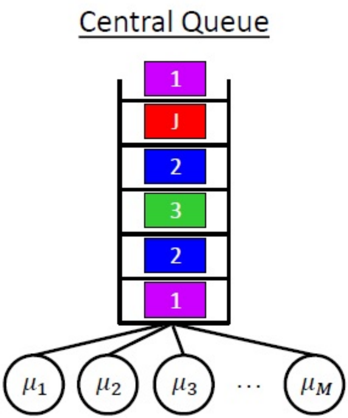




Outline

- Redundancy
- Central Queue Architecture
- Order Independent Descriptor
  - Redundancy and cancel on complete
- Aggregated State Descriptor
  - Redundancy and cancel on start
- Generalizations:
  - Token-based framework
  - Generalized Order Independent
- Impact of assumptions: scheduling and independence

Central Queue



State descriptors:

- Job's point of view: $c(n) = (c_1, \ldots, c_n)$
- Server's point of view: $s = (n_1, M_1, \ldots, n_i, M_i)$

Order Independent Queues[1]

Classes $i = 1, \ldots, N$.
$\mathcal{S}_i$ set of servers that can process class-i
Service is FCFS

State descriptor: $c(n) = (c_1, \ldots, c_n)$.

$$\mu(c(n)) = \sum_{s \in \bigcup_{k=1}^{n} \mathcal{S}_{c_k}} \mu_s$$

[1] A. Krzesinski, Order independent queues, in "Queueing Networks: a fundamental approach", Eds: R. Boucherie, N. van Dijk, 2011.

Order Independent Queues[1]

Classes $i = 1, \ldots, N$.
$\mathcal{S}_i$ set of servers that can process class-i
Service is FCFS

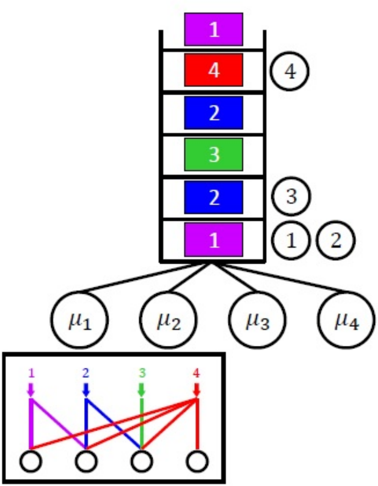
State descriptor: $c(n) = (c_1, \ldots, c_n)$.

$$\mu(c(n)) = \sum_{s \in \bigcup_{k=1}^{n} \mathcal{S}_{c_k}} \mu_s$$

$$\mu(c_1, \ldots, c_k) - \mu(c_1, \ldots, c_{k-1}) = \sum_{s \in \mathcal{S}_{c_k} \setminus \bigcup_{l=1}^{k-1} \mathcal{S}_{c_l}} \mu_s$$

[1] A. Krzesinski, Order independent queues, in "Queueing Networks: a fundamental approach", Eds: R. Boucherie, N. van Dijk, 2011.

Order Independent Queues (example)[2]

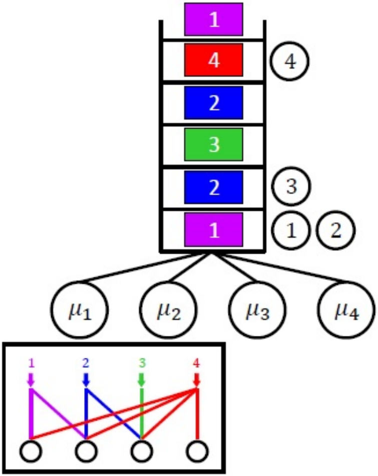


State $c(6) = (1, 2, 3, 2, 4, 1)$

$\mu(c(6)) = \mu_1 + \mu_2 + \mu_3 + \mu_4$

[2] From Gardner and Righter's APS tutorial: https://kgardner.people.amherst.edu/

Order Independent Queues (example)[2]



State $c(6) = (1, 2, 3, 2, 4, 1)$

$\mu(c(6)) = \mu_1 + \mu_2 + \mu_3 + \mu_4$

$\mu(1, 2) - \mu(1) = \mu_3$

[2] From Gardner and Righter's APS tutorial: https://kgardner.people.amherst.edu/

OI queues

Key OI properties:

- For k-th job, its service rate $\mu(c_1, \ldots, c_k) - \mu(c_1, \ldots, c_{k-1})$ can only depend on what lies ahead.
- $\mu(c_1, \ldots, c_n) = \mu(c_{\sigma(1)}, \ldots, c_{\sigma(n)})$

OI queues

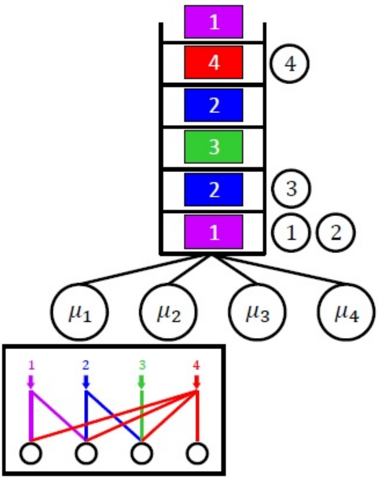
Key OI properties:

- For k-th job, its service rate $\mu(c_1, \ldots, c_k) - \mu(c_1, \ldots, c_{k-1})$ can only depend on what lies ahead.
- $\mu(c_1, \ldots, c_n) = \mu(c_{\sigma(1)}, \ldots, c_{\sigma(n)})$

: Theorem: Given OI properties, the queue is quasi-reversible and the stationary distribution is:

$$\pi(c) = \pi(\emptyset) \prod_{k=1}^{n} \frac{\lambda_{c_k}}{\mu(c_1, \ldots, c_k)}$$

Order Independent Queues (example)



State $c(6) = (1, 2, 3, 2, 4, 1)$

$$\pi(c(6)) = \pi(\emptyset)\left(\frac{\lambda_1}{\mu_{1,2}}\right)\left(\frac{\lambda_2}{\mu_{1,2,3}}\right)\left(\frac{\lambda_3}{\mu_{1,2,3}}\right)\left(\frac{\lambda_2}{\mu_{1,2,3}}\right)\left(\frac{\lambda_4}{\mu}\right)\left(\frac{\lambda_1}{\mu}\right)$$

Sketch Proof of OI

Distribution satisfies partial balance equations:

rate out of $c(n)$ due to departure $=$ rate into $c(n)$ due to arrival

rate out of $c(n)$ due to class c arrival $=$ rate into $c(n)$ due to class c departure

[3] T. Bonald, C. Comte, F Mathieu, Performance of Balanced Fairness in Resource Pools: A Recursive Approach, Sigmetrics 2017

Sketch Proof of OI

Distribution satisfies partial balance equations:

rate out of $c(n)$ due to departure $=$ rate into $c(n)$ due to arrival
rate out of $c(n)$ due to class c arrival $=$ rate into $c(n)$ due to class c departure

Quasi reversible $\Longrightarrow$ Networks of queues have product form stationary distribution

[3] T. Bonald, C. Comte, F Mathieu, Performance of Balanced Fairness in Resource Pools: A Recursive Approach, Sigmetrics 2017

Sketch Proof of OI

Distribution satisfies partial balance equations:

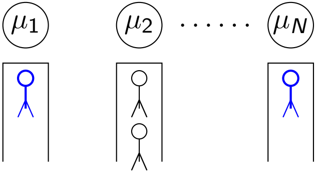
rate out of $c(n)$ due to departure $=$ rate into $c(n)$ due to arrival
rate out of $c(n)$ due to class c arrival $=$ rate into $c(n)$ due to class c departure

Quasi reversible $\Longrightarrow$ Networks of queues have product form stationary distribution

$\pi(\emptyset)$ can be computed recursively by removing a server s and all classes compatible with that server[3]

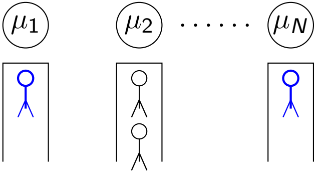
[3] T. Bonald, C. Comte, F Mathieu, Performance of Balanced Fairness in Resource Pools: A Recursive Approach, Sigmetrics 2017

Redundancy with *c.o.c.* is OI



$\mu(c_1, \dots, c_n) =$ sum of rates serving these classes

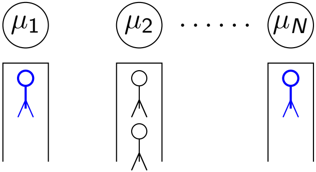
Redundancy with *c.o.c.* is OI



$\mu(c_1, \ldots, c_n)$ = sum of rates serving these classes

$\mu(c_1, \ldots, c_n) - \mu(c_1, \ldots, c_{n-1})$ = sum of rates of servers available to n class jobs

Redundancy with *c.o.c.* is OI



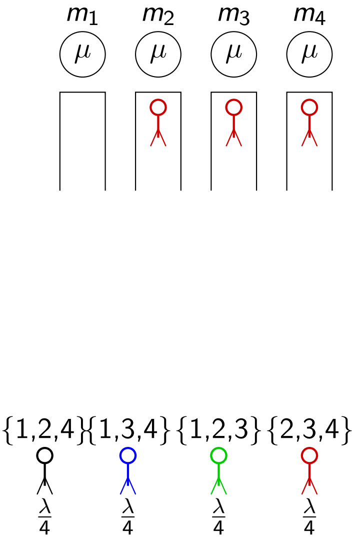
$\mu(c_1, \ldots, c_n)$ = sum of rates serving these classes

$\mu(c_1, \ldots, c_n) - \mu(c_1, \ldots, c_{n-1})$ = sum of rates of servers available to n class jobs

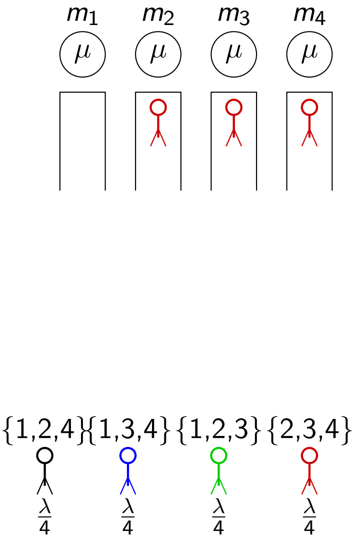
Steady-state distribution is product form!

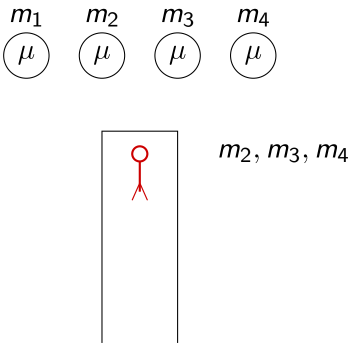
Key assumption: service time of copies independent!

c.o.c. and central queue

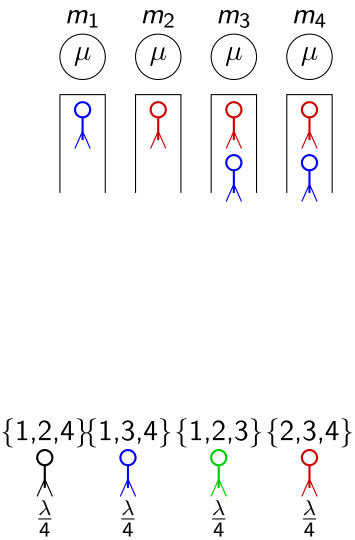


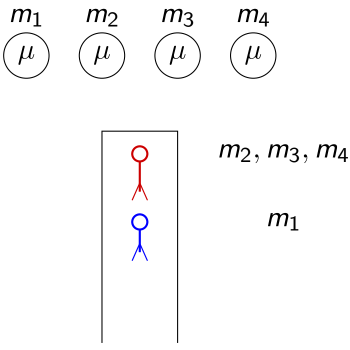
$c.o.c.$ and central queue



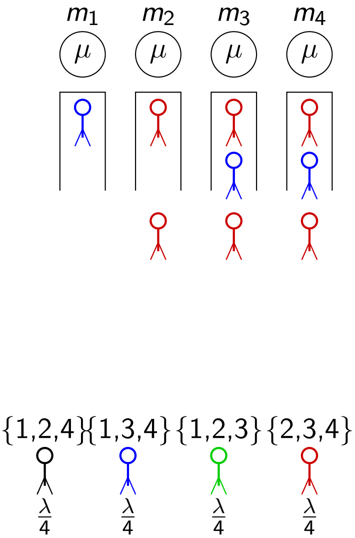


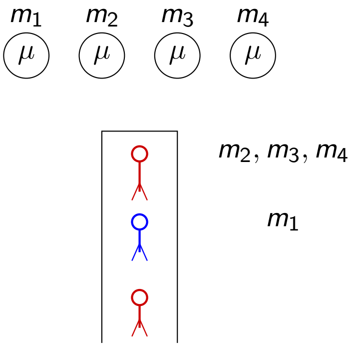
$c.o.c.$ and central queue



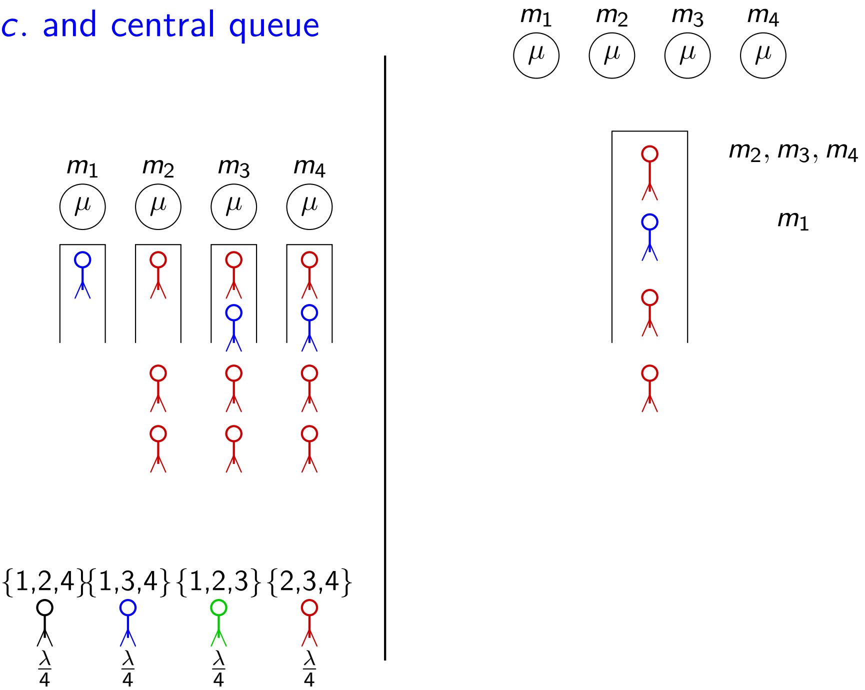


c.o.c. and central queue

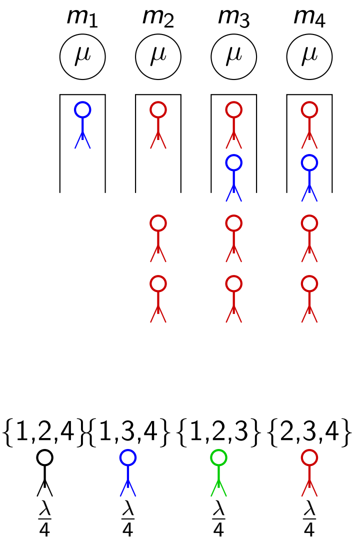


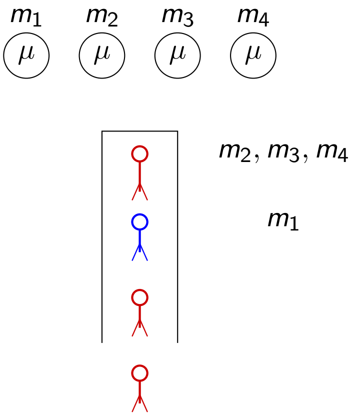


$c.o.c.$ and central queue



c.o.c. and central queue

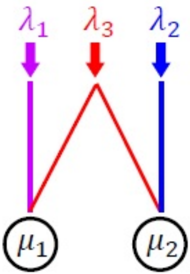




Gardner et al. QUESTA 2016

Performance Evaluation of Redundancy with *c.o.c.*

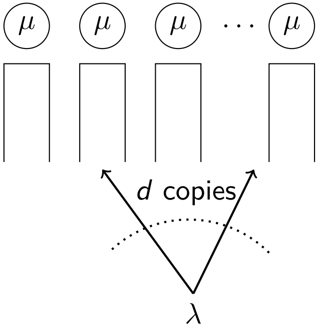
- Assume particular model: W, N etc.



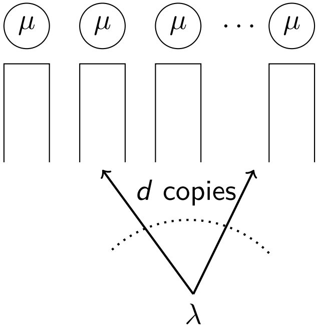
- Redundancy-d model[4]

[4] Gardner et al, Redundancy-d: The power of d choices for redundancy, OR 2017

redundancy-d with $c.o.c.$

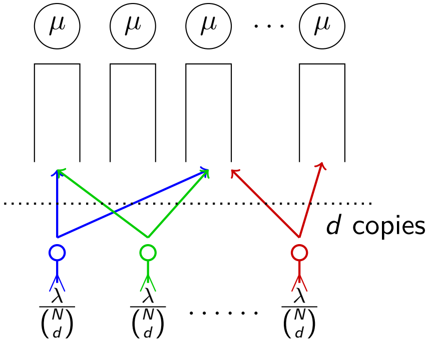


redundancy-d with $c.o.c.$



- N homogeneous servers with FIFO discipline
- Jobs arrivals are Poisson with rate λ
- Jobs have exponential service requirement
- Each arrival chooses d servers at random
- i.i.d. redundant copies for a job placed at d servers

redundancy-d with $c.o.c.$



- A special case of the generic multiclass model
- There are $\binom{N}{d}$ classes
- Each class has an arrival rate of $\frac{\lambda}{\binom{N}{d}}$

Performance Measures

Mean Delay

$$\mathbb{E}(T^{coc}) = \sum_{i=d}^{k} \frac{1}{k\mu \frac{\binom{k-1}{d-1}}{\binom{i-1}{d-1}} - k\lambda}$$

Mean-field limit[5]

$$\mathbb{P}(T^{coc} > t) = \left(\frac{1}{\rho + (1-\rho)e^{t\mu(d-1)}} \right)^{\frac{\alpha}{\alpha-1}}$$

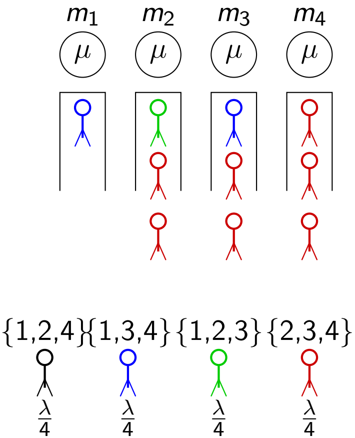
[5] M. Bramson, Yi Lu, B. Prabhakar, Randomized load balancing with general service time distributions. SIGMETRICS 2010: 275-286

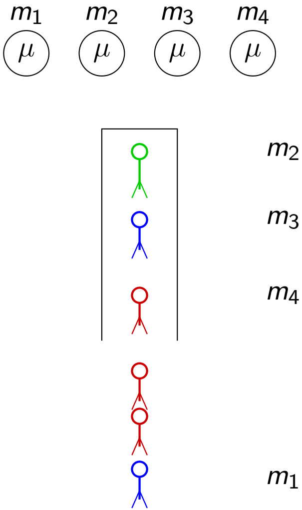
Outline

- Redundancy
- Central Queue Architecture
- Order Independent Descriptor
  - Redundancy and cancel on complete
- Aggregated State Descriptor
  - Redundancy and cancel on start
- Generalizations:
  - Token-based framework
  - Generalized Order Independent
- Impact of assumptions: scheduling and independence

Redundancy *c.o.s.* is not OI

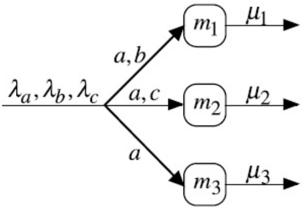
$N = 4$ *and* $d = 3$

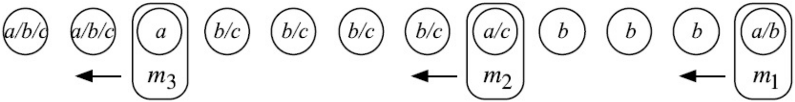




$(c_n, \ldots, c_1)$ no product form!

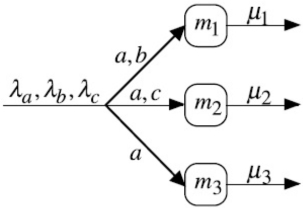
Multi-type job and multi-type servers[6]

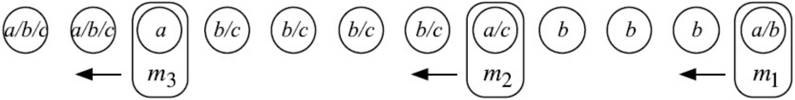




[6] Visschers, Adan, Weiss, *A product form solution to a system with multi-type jobs and multi-type servers, Queueing Systems, 2012.*

Multi-type job and multi-type servers[6]





- Markovian descriptor of (aggregated) form $s = (n_i, M_i, \ldots, n_d, M_d, \ldots, M_1)$
- If i servers are busy, then departure rate is $i\mu$
- Assignment rule determines server in case multiple compatible servers are available

[6]Visschers, Adan, Weiss, *A product form solution to a system with multi-type jobs and multi-type servers, Queueing Systems, 2012.*

Multi-type job and multi-type servers (cont.)

Key results: Existence of assignment rule, stability condition, characterization of steady-state distribution

$$\pi(\mathfrak{s}) = \alpha_i^{n_i} \cdots \alpha_1^{n_1} \frac{\Pi_\lambda(\{M_1, \ldots, M_i\})}{\Pi_\mu(M_i, \ldots, M_1)} \pi(0).$$

Caveat: No efficient way to calculate $\pi(0)$

[7] Adan, Weiss, A skill based parallel service system under FCFS-ALIS: steady state, overloads, and abandonments Stochastic Systems, INFORMS, 2014, 4, 250-299

Multi-type job and multi-type servers (cont.)

Key results: Existence of assignment rule, stability condition, characterization of steady-state distribution

$$\pi(\mathfrak{s}) = \alpha_i^{n_i} \cdots \alpha_1^{n_1} \frac{\Pi_\lambda(\{M_1, \ldots, M_i\})}{\Pi_\mu(M_i, \ldots, M_1)} \pi(0).$$

Caveat: No efficient way to calculate $\pi(0)$

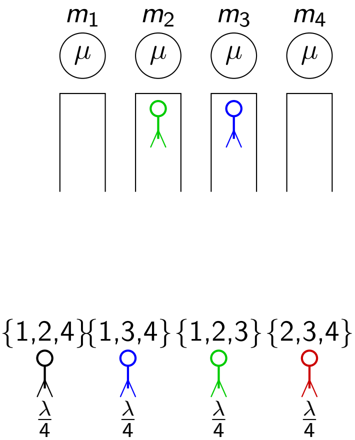
Results of similar nature with ALIS[7]

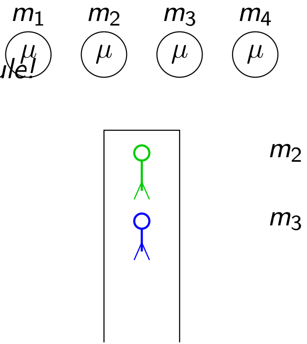
[7] Adan, Weiss, A skill based parallel service system under FCFS-ALIS: steady state, overloads, and abandonments Stochastic Systems, INFORMS, 2014, 4, 250-299

redundancy-d with $c.o.s.$ (example)

$N = 4$ and $d = 3$.

Uniform assignment satisfies assignment rule!

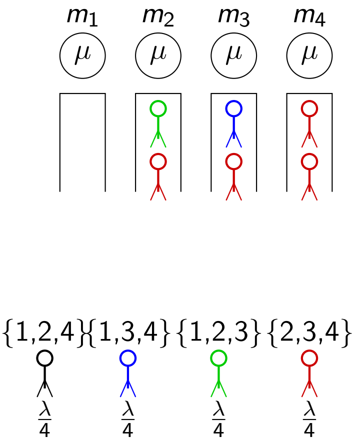


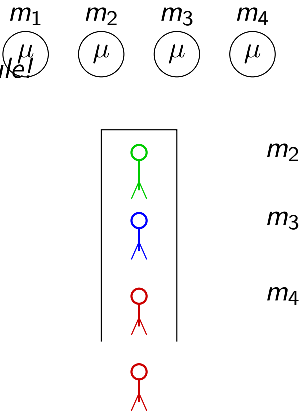


redundancy-d with *c.o.s.* (example)

$N = 4$ *and* $d = 3$.

Uniform assignment satisfies assignment rule!

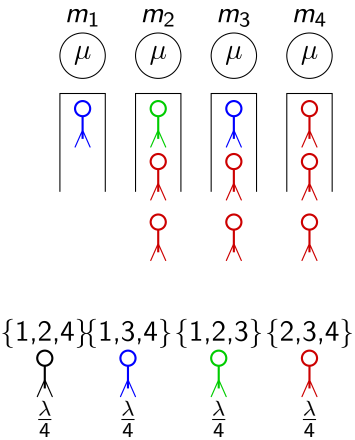


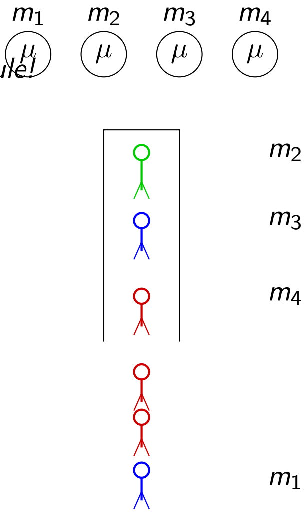


redundancy-d with $c.o.s.$ (example)

$N = 4$ and $d = 3$.

Uniform assignment satisfies assignment rule!

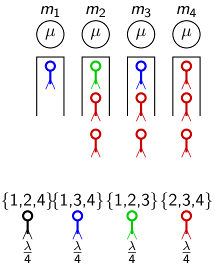


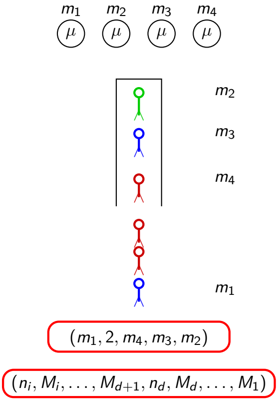


$$\boxed{(m_2, m_3, m_4, 2, m_1)}$$

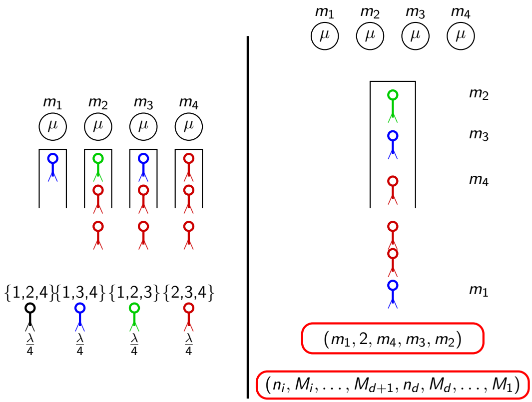
$$\boxed{(M_1, \ldots, M_d, n_d, M_{d+1}, \ldots, M_i, n_i)}$$

redundancy-d with $c.o.s.$ – results





redundancy-d with $c.o.s.$ – results



For any state $s = (n_i, M_i, \ldots, M_{d+1}, n_d, M_d, \ldots, M_1)$, we have

$$\pi(s) = {r_i}^{n_i} \ldots {r_d}^{n_d} \prod_{j=1}^{i} G_j(K, d) \frac{\pi(0)}{i!\mu^i} \rightarrow \text{product form}$$

redundancy-d with $c.o.s.$ – results[8]

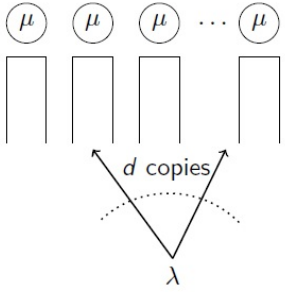
$$\pi(s) = {r_i}^{n_i} \dots {r_d}^{n_d} \prod_{j=1}^{i} G_j(K, d) \frac{\pi(0)}{i!\mu^i}$$

With a bit of algebra, and using the form of $\pi(s)$, we obtain

- $\pi(0)$, the normalizing constant
- $p(i)$, probability of i busy servers
- The P.G.F. of the number of waiting jobs in the system
- $E(N)$ the expected number of jobs in the system

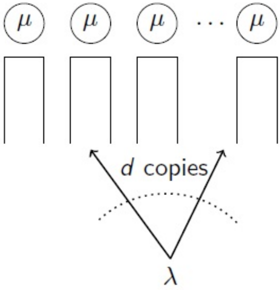
[8] A., Bodas, Verloop, On a unifying product form framework for redundancy models, Performance Evaluation, 2018

Comparing *c.o.s.* and *c.o.c.*



If $d = K$, c.o.c. is equivalent to an $M/M/1$ with rate μK
If $d = K$, c.o.s. is equivalent to an $M/M/K$

Comparing *c.o.s.* and *c.o.c.*

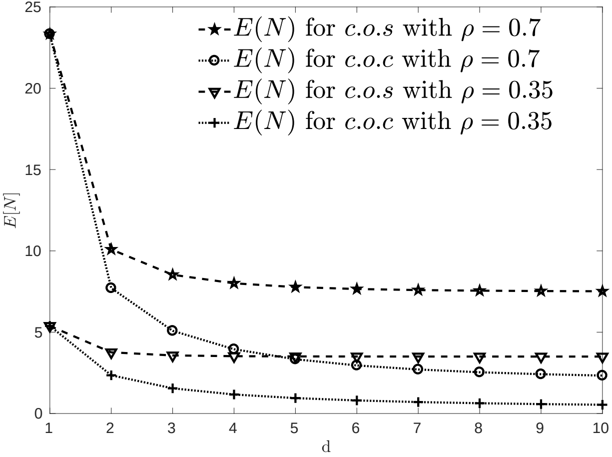


If $d = K$, c.o.c. is equivalent to an $M/M/1$ with rate μK
If $d = K$, c.o.s. is equivalent to an $M/M/K$

- What without i.i.d. assumption?

$d = K, c.o.c. \implies$ single server with rate μ

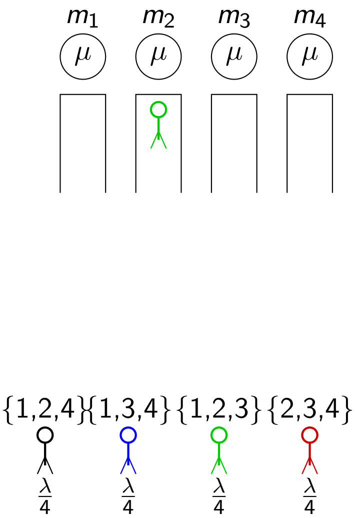
Performance comparison between *c.o.s.* and *c.o.c.*



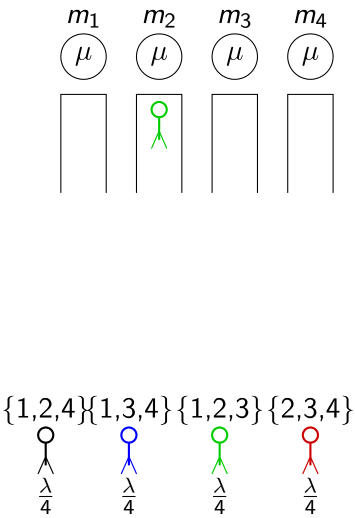
Sample path arguments

$N = 4$ *and* $d = 3$

redundancy-d with *c.o.s.*

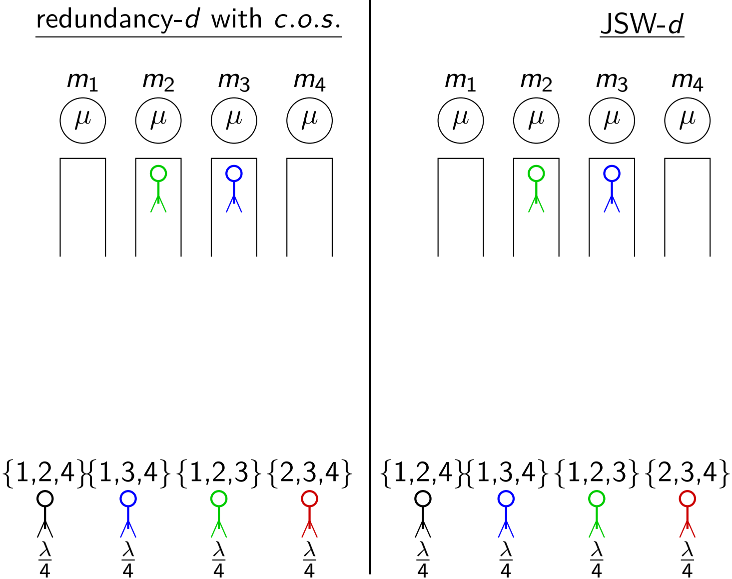


JSW-d



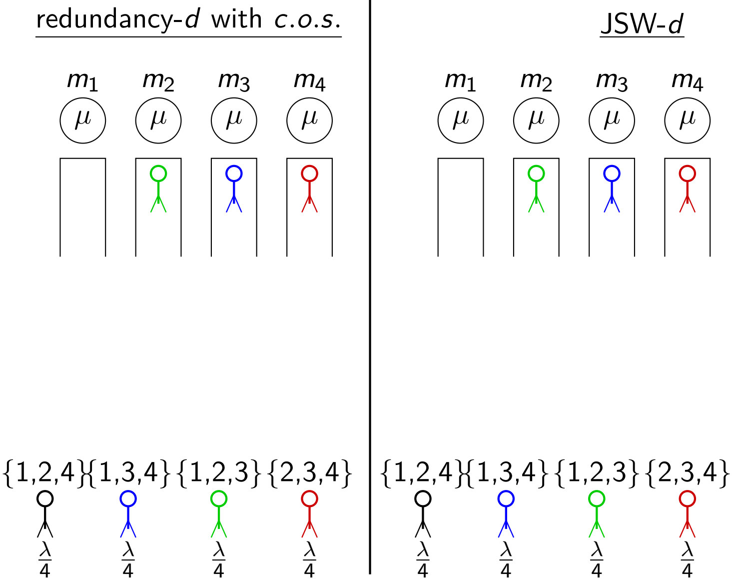
Sample path arguments

$N = 4$ *and* $d = 3$



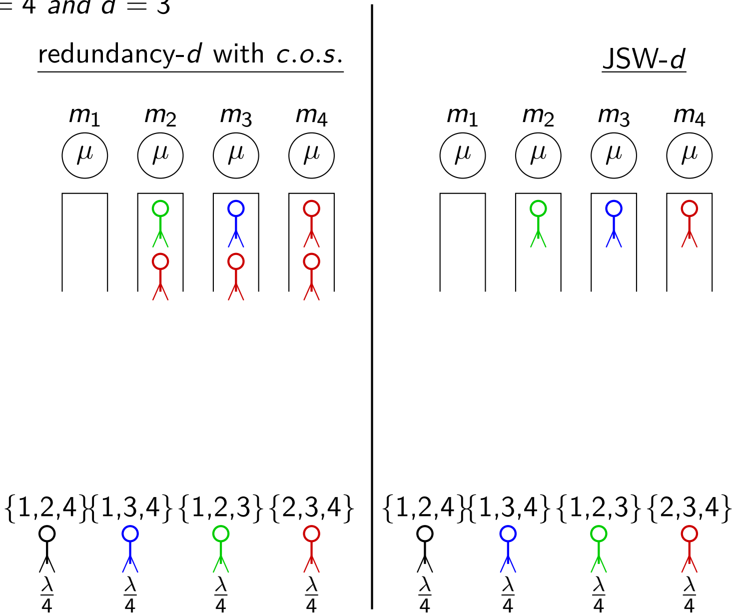
Sample path arguments

$N = 4$ *and* $d = 3$



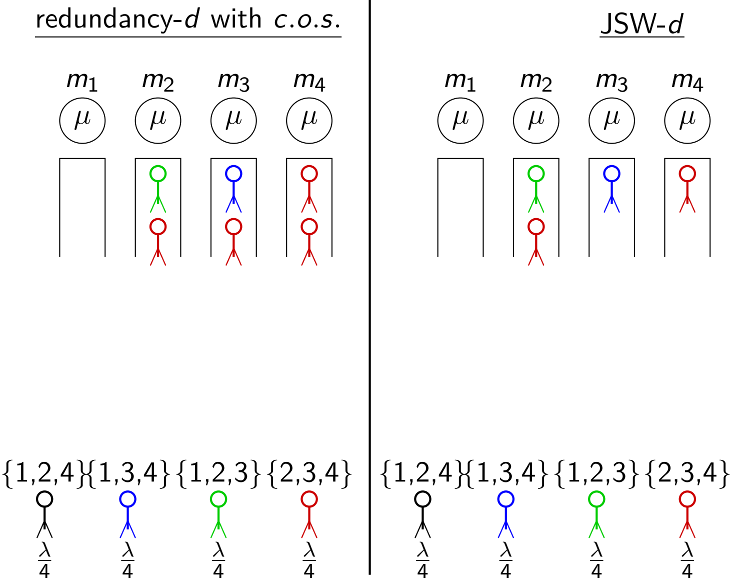
Sample path arguments

$N = 4$ *and* $d = 3$

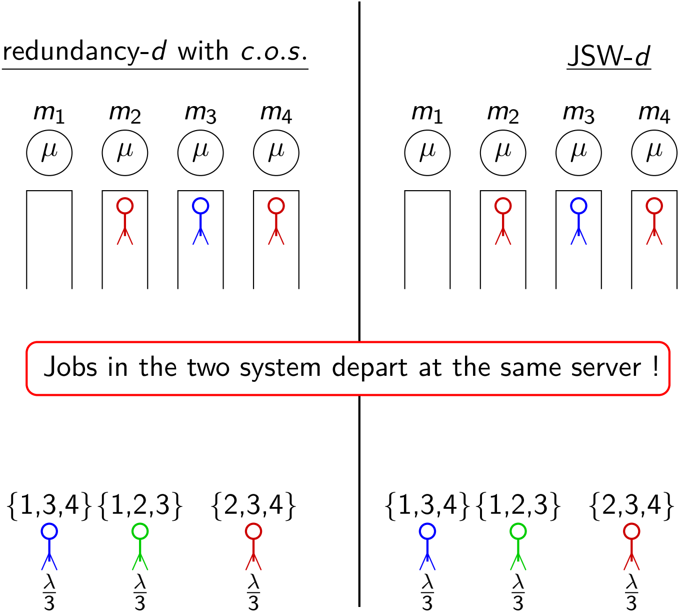


Sample path arguments

N = 4 *and d* = 3



Sample path coupling



Equivalence with JSW(d)[9]

Proposition

For any given sample-path realization, a given job will be served under both JSW(d) and redundancy-d with $c.o.s.$ in the same server.

We now know $\pi(0), p(i)$, P.G.F for the number of waiting jobs and $E(N)$ for JSW-d through the analysis of Redundancy c.o.s.

[9] A., Bodas, Verloop, On a unifying product form framework for redundancy models, Performance Evaluation, 2018

Redundancy with c.o.s. in mean-field[10]

$$\mathbb{P}(T > t) = \left(\lambda^d + (1-\lambda^d)e^{t(d-1)}\right)^{\frac{1}{d-1}}$$

$$\mathbb{E}(T^{cos}) = \sum_{n=0}^{\infty} \frac{\lambda^{dn}}{1 + n(d-1)}$$

[10] T. Hellemans, B. van Houdt, On the power-of-d-choices with least loaded server selection, Sigmetrics 2018

Redundancy with c.o.s. in mean-field[10]

$$\mathbb{P}(T > t) = \left(\lambda^d + (1 - \lambda^d)e^{t(d-1)}\right)^{\frac{1}{d-1}}$$

$$\mathbb{E}(T^{cos}) = \sum_{n=0}^{\infty} \frac{\lambda^{dn}}{1 + n(d-1)}$$

In the mean-field limit:

$$\mathbb{E}(T^{cos}) \geq \mathbb{E}(T^{coc})$$

[10] T. Hellemans, B. van Houdt, On the power-of-d-choices with least loaded server selection, Sigmetrics 2018

Outline

- Redundancy
- Central Queue Architecture
- Order Independent Descriptor
  - Redundancy and cancel on complete
- Aggregated State Descriptor
  - Redundancy and cancel on start
- Generalizations:
  - Token-based framework
  - Generalized Order Independent
- Impact of assumptions: scheduling and independence

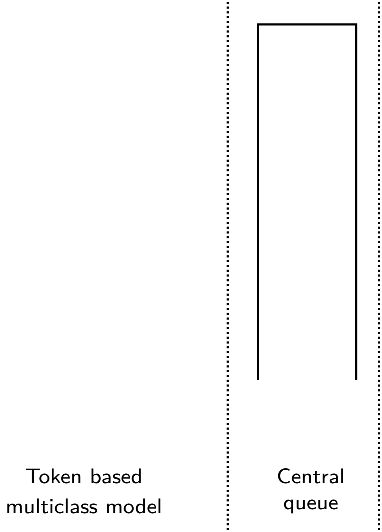
Token based framework[11]

- Multiclass jobs and tokens for service
- Product form stationary distribution
- Subsumes OI and Visschers et al., and more

[11] A., Bodas, Dorsman, Verloop, A token-based central queue with order-independent service rates , to appear in OR

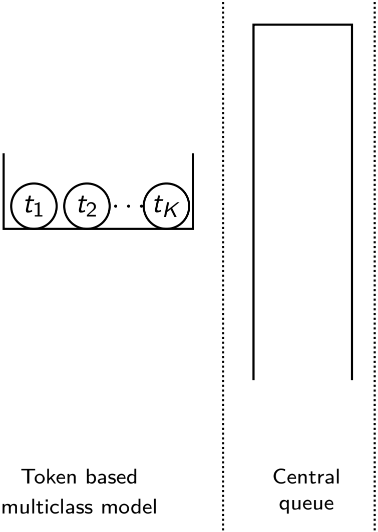
A token based central queue

A token based central queue



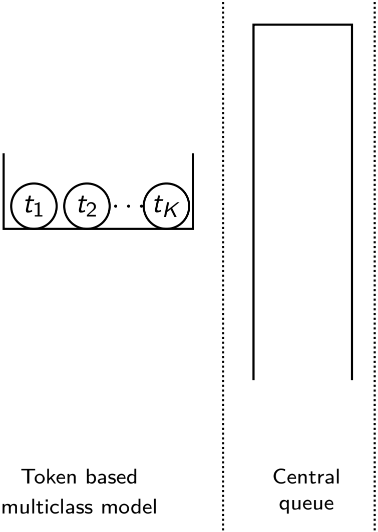
- A single central queue

A token based central queue



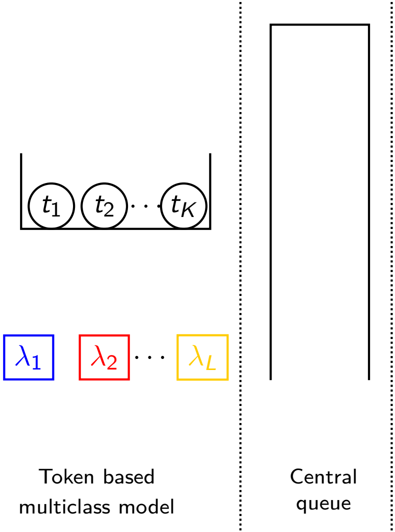
- A single central queue
- A set $\mathcal{T} = \{t_1, \ldots, t_K\}$ of K tokens

A token based central queue



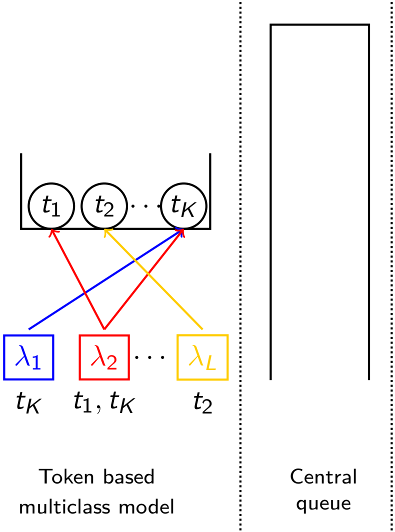
- A single central queue
- A set $\mathcal{T} = \{t_1, \ldots, t_K\}$ of K tokens
- Only jobs with tokens are served

A token based central queue



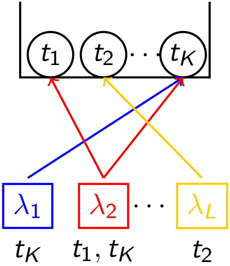
- A single central queue
- A set $\mathcal{T} = \{t_1, \ldots, t_K\}$ of K tokens
- Only jobs with tokens are served
- There are L job classes
- Poisson arrival rate λ_i for class i

A token based central queue

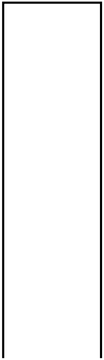


- A single central queue
- A set $\mathcal{T} = \{t_1, \ldots, t_K\}$ of K tokens
- Only jobs with tokens are served
- There are L job classes
- Poisson arrival rate λ_i for class i
- A compatibility graph (arbitrary)

A token based central queue

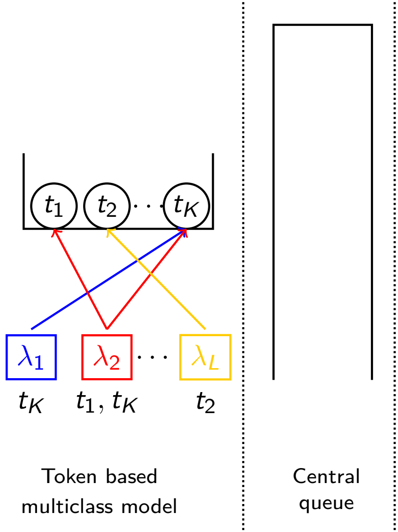


Token based multiclass model

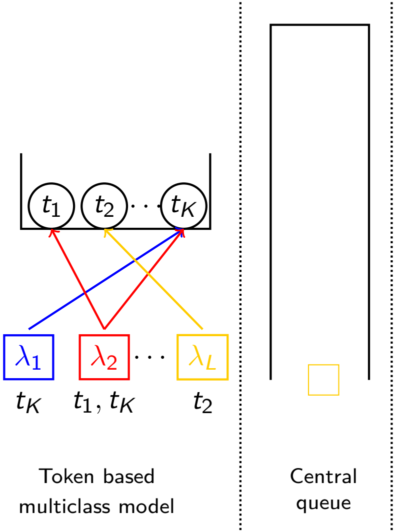


Central queue

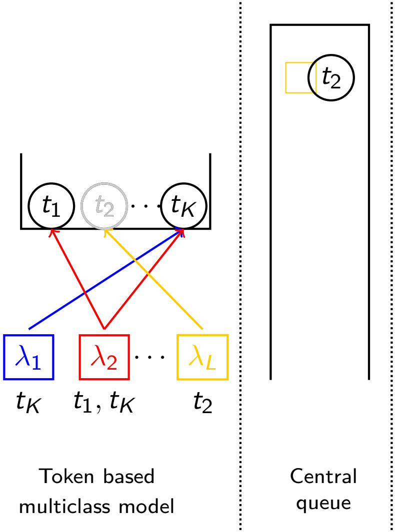
- A single central queue
- A set $\mathcal{T} = \{t_1, \ldots, t_K\}$ of K tokens
- Only jobs with tokens are served
- There are L job classes
- Poisson arrival rate λ_i for class i
- A compatibility graph (arbitrary)
- Features of token based queue
  - Token assignment
  - Releasing a token
  - Token service rate
  - State space



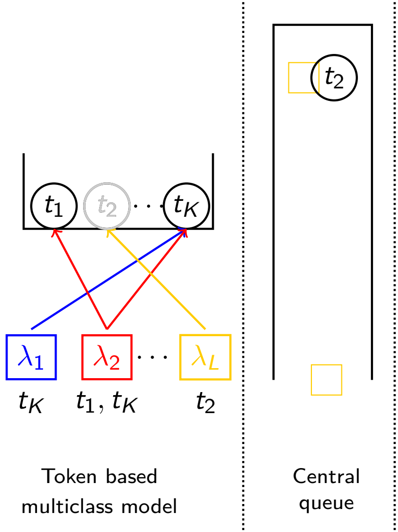
- An arriving job must pick a compatible token if available



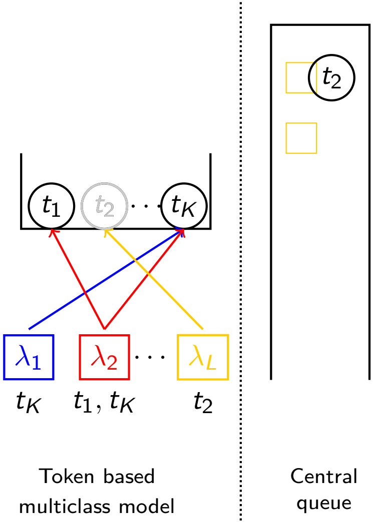
- An arriving job must pick a compatible token if available



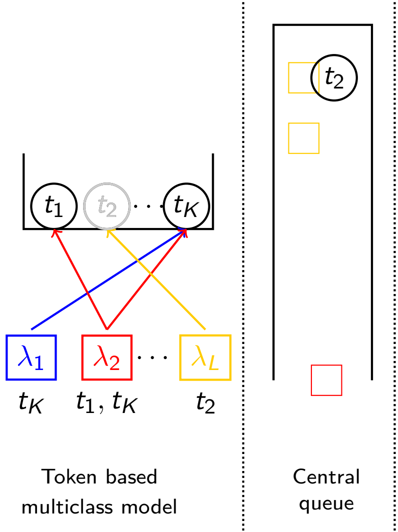
- An arriving job must pick a compatible token if available



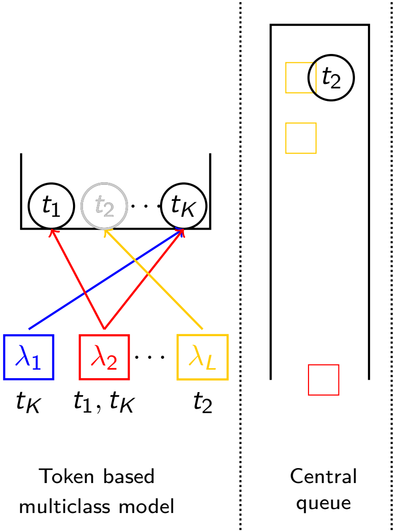
- An arriving job must pick a compatible token if available



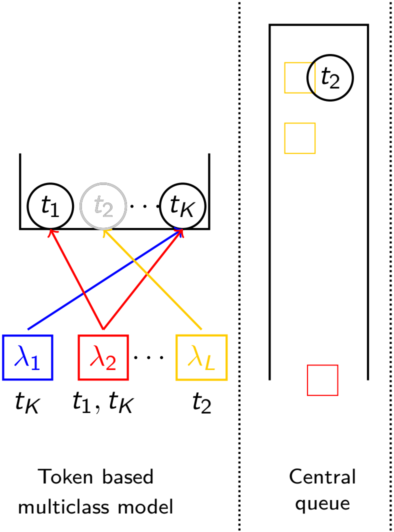
- An arriving job must pick a compatible token if available
- Job with no feasible token will wait in the queue



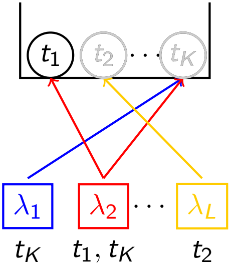
- An arriving job must pick a compatible token if available
- Job with no feasible token will wait in the queue



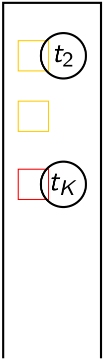
- An arriving job must pick a compatible token if available
- Job with no feasible token will wait in the queue
- A job can claim only one compatible token



- An arriving job must pick a compatible token if available
- Job with no feasible token will wait in the queue
- A job can claim only one compatible token
- An *assignment rule* specifies the tie-breaking rule

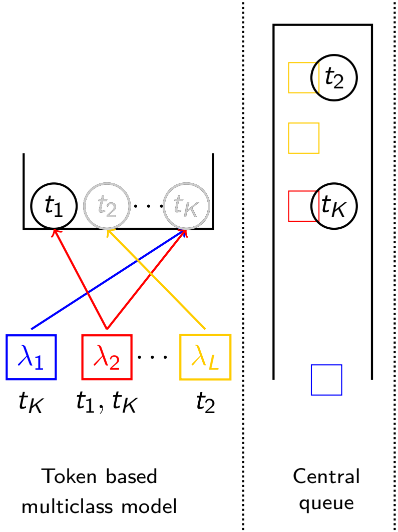


Token based multiclass model

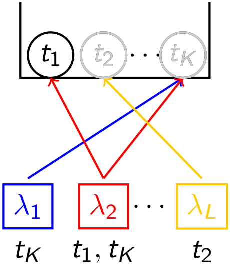


Central queue

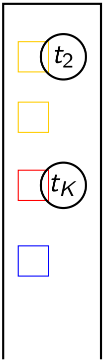
- An arriving job must pick a compatible token if available
- Job with no feasible token will wait in the queue
- A job can claim only one compatible token
- An *assignment rule* specifies the tie-breaking rule



- An arriving job must pick a compatible token if available
- Job with no feasible token will wait in the queue
- A job can claim only one compatible token
- An *assignment rule* specifies the tie-breaking rule

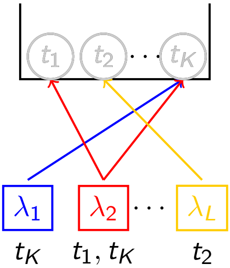


Token based multiclass model

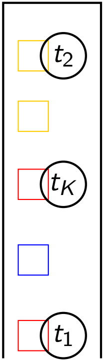


Central queue

- An arriving job must pick a compatible token if available
- Job with no feasible token will wait in the queue
- A job can claim only one compatible token
- An *assignment rule* specifies the tie-breaking rule

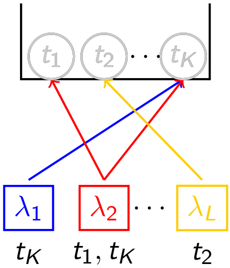


Token based
multiclass model

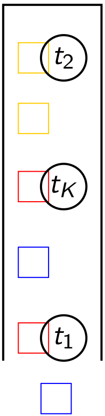


Central
queue

- An arriving job must pick a compatible token if available
- Job with no feasible token will wait in the queue
- A job can claim only one compatible token
- An *assignment rule* specifies the tie-breaking rule

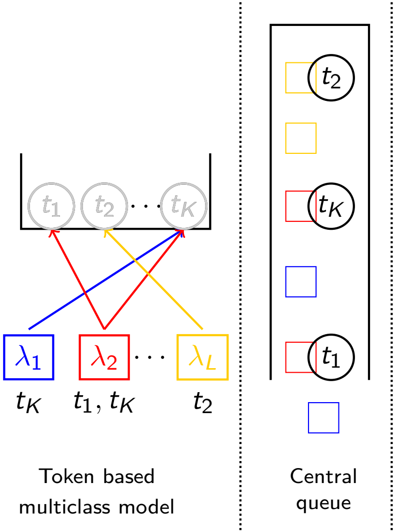


Token based multiclass model

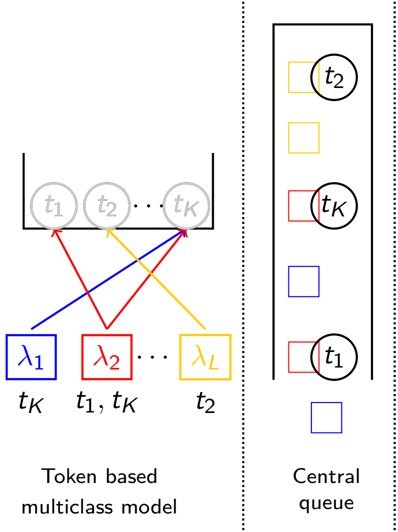


Central queue

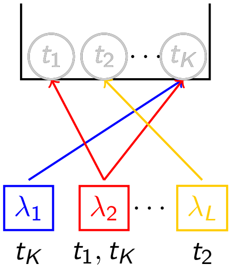
- An arriving job must pick a compatible token if available
- Job with no feasible token will wait in the queue
- A job can claim only one compatible token
- An *assignment rule* specifies the tie-breaking rule

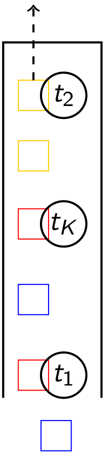


- Jobs have an exponential service requirement with unit mean
- Only jobs with tokens will receive a non-negative service rate

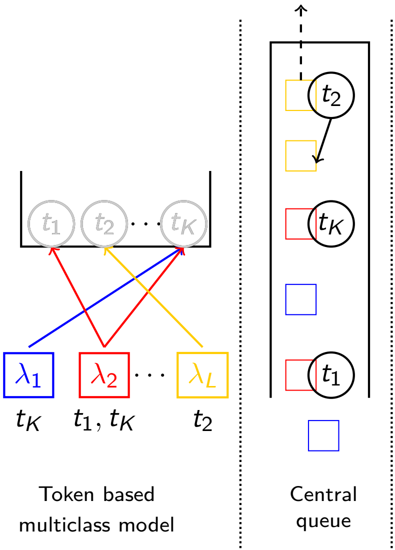


- Jobs have an exponential service requirement with unit mean
- Only jobs with tokens will receive a non-negative service rate
- When a job departs, the released token picks the next compatible waiting job

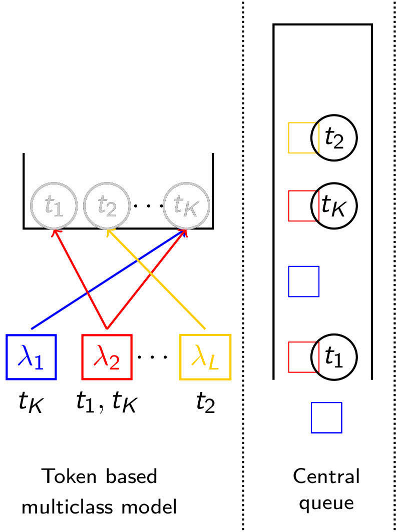




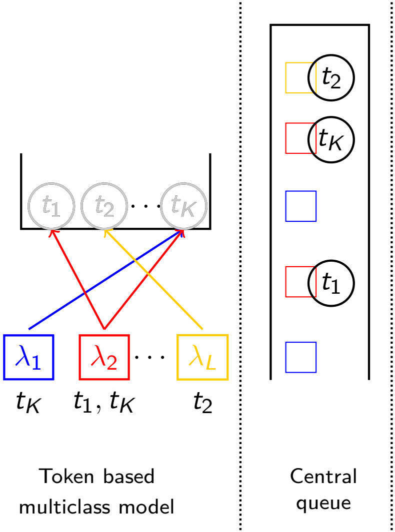
- Jobs have an exponential service requirement with unit mean
- Only jobs with tokens will receive a non-negative service rate
- When a job departs, the released token picks the next compatible waiting job



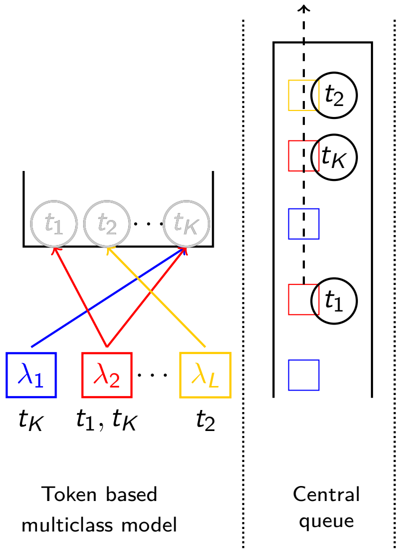
- Jobs have an exponential service requirement with unit mean
- Only jobs with tokens will receive a non-negative service rate
- When a job departs, the released token picks the next compatible waiting job



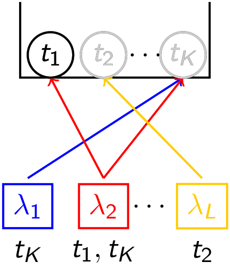
- Jobs have an exponential service requirement with unit mean
- Only jobs with tokens will receive a non-negative service rate
- When a job departs, the released token picks the next compatible waiting job



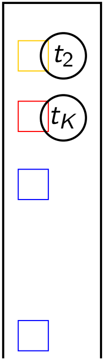
- Jobs have an exponential service requirement with unit mean
- Only jobs with tokens will receive a non-negative service rate
- When a job departs, the released token picks the next compatible waiting job



- Jobs have an exponential service requirement with unit mean
- Only jobs with tokens will receive a non-negative service rate
- When a job departs, the released token picks the next compatible waiting job

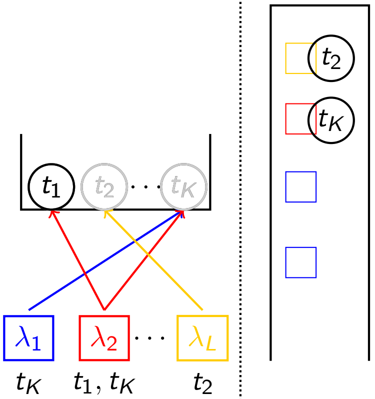


Token based multiclass model



Central queue

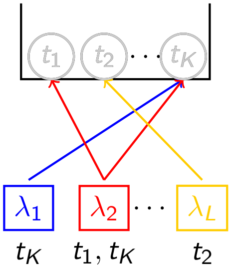
- Jobs have an exponential service requirement with unit mean
- Only jobs with tokens will receive a non-negative service rate
- When a job departs, the released token picks the next compatible waiting job
- If no waiting compatible job present, token added back to the token set



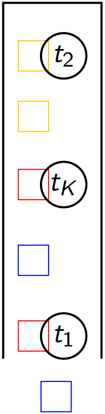
Token based multiclass model

Central queue

- Jobs have an exponential service requirement with unit mean
- Only jobs with tokens will receive a non-negative service rate
- When a job departs, the released token picks the next compatible waiting job
- If no waiting compatible job present, token added back to the token set



Token based multiclass model

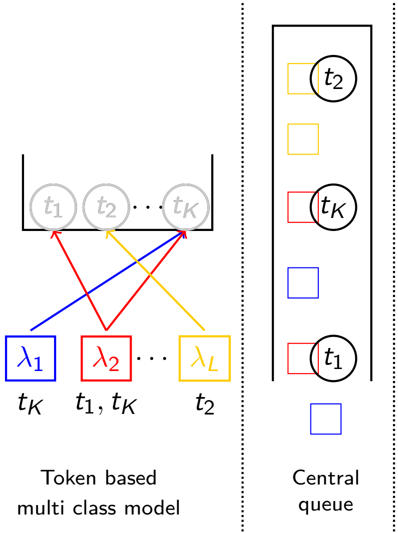


Central queue

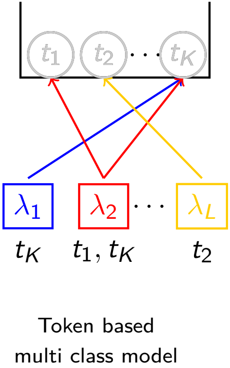
- Markovian descriptor for token based model
- Anonymize the class information
- $(t_2, 1, t_k, 1, t_1, 1)$
- $(T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$

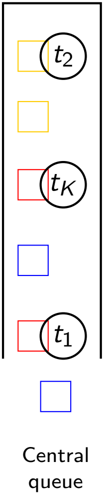
oldest job

jobs that are waiting for their compatible tokens

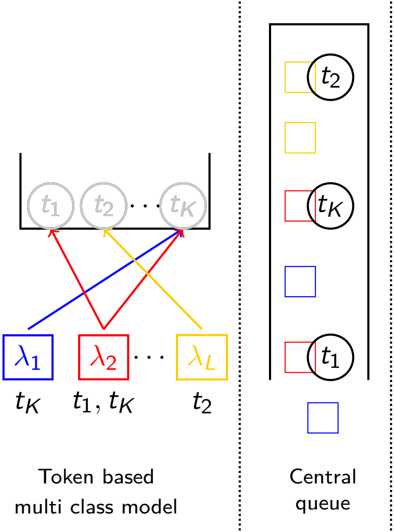


t_1 t_2 … t_K
λ_1 λ_2 … λ_L
t_K t_1, t_K t_2
Token based multi class model
t_2
t_K
t_1
Central queue

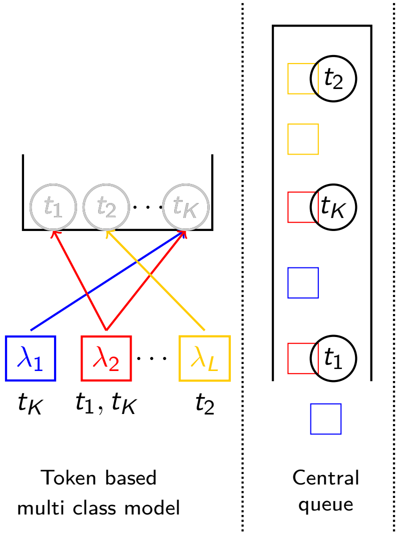




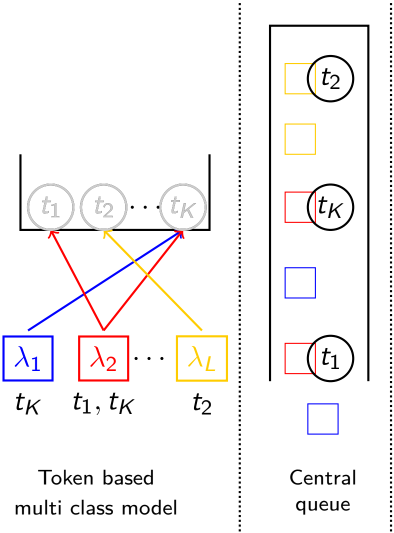
- Consider a generic state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$



- Consider a generic state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$
- $\mu_{T_j}(x)$ denote the departure rate of the job with token T_j



- Consider a generic state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$
- $\mu_{T_j}(x)$ denote the departure rate of the job with token T_j
- $\mu(x) = \sum_{j=1}^{i} \mu_{T_j}(x)$



- Consider a generic state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$
- $\mu_{T_j}(x)$ denote the departure rate of the job with token T_j
- $\mu(x) = \sum_{j=1}^{i} \mu_{T_j}(x)$

Token service rate

Objects of departures T_j

$$s = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$$

[12] A. Krzesinski, Order independent queues, in "Queueing Networks: a fundamental approach", Eds: R. Boucherie, N. van Dijk, 2011

Token service rate

Objects of departures T_j

$$s = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$$

Require the departure function $\hat{\mu}(\cdot)$ satisfying the following

1. $\hat{\mu}(T_1 \ldots T_j) - \hat{\mu}(T_1 \ldots T_{j-1}) \geq 0$
2. $\hat{\mu}(T_1 \ldots T_j) = \hat{\mu}(T_{\sigma(1)} \ldots O_{\sigma(j-1)})$

Order Independent rates[12] $\Longrightarrow$ Sufficient condition for product-form

[12] A. Krzesinski, Order independent queues, in "Queueing Networks: a fundamental approach", Eds: R. Boucherie, N. van Dijk, 2011

Token service rate

Objects of departures T_j

$$s = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$$

Require the departure function $\hat{\mu}(\cdot)$ *satisfying the following*

1. $\hat{\mu}(T_1 \ldots T_j) - \hat{\mu}(T_1 \ldots T_{j-1}) \geq 0$
2. $\hat{\mu}(T_1 \ldots T_j) = \hat{\mu}(T_{\sigma(1)} \ldots O_{\sigma(j-1)})$

Order Independent rates[12] $\Longrightarrow$ Sufficient condition for product-form

For redundancy-d c.o.s., the T_j *correspond to server* M_j
For redundancy-d c.o.c., the T_j *correspond to first customer of a class* c_j

[12] A. Krzesinski, Order independent queues, in "Queueing Networks: a fundamental approach", Eds: R. Boucherie, N. van Dijk, 2011

State transitions

$$(T_1, n_1, .., T_j, n_j, .., T_i, n_i)$$

State transitions

$$\underbrace{\boxed{(T_1, n_1, .., T_j, n_j, .., T_i, n_i)}}_{x}$$

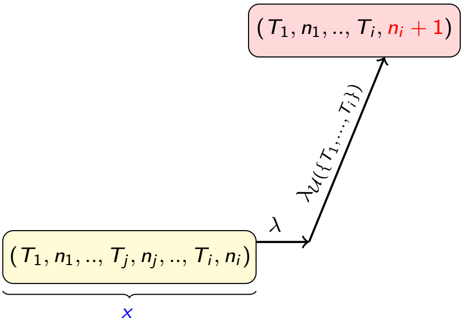
State transitions

$$\underbrace{\boxed{(T_1, n_1, .., T_j, n_j, .., T_i, n_i)}}_{x} \xrightarrow{\lambda}$$

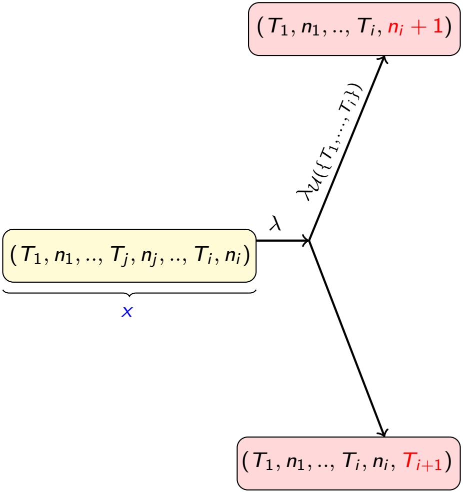
State transitions

$$\begin{tikzcd} & (T_1, n_1, .., T_i, n_i + 1) \\ \underbrace{(T_1, n_1, .., T_j, n_j, .., T_i, n_i)}_{x} \arrow[ur, "\lambda"] & \end{tikzcd}$$

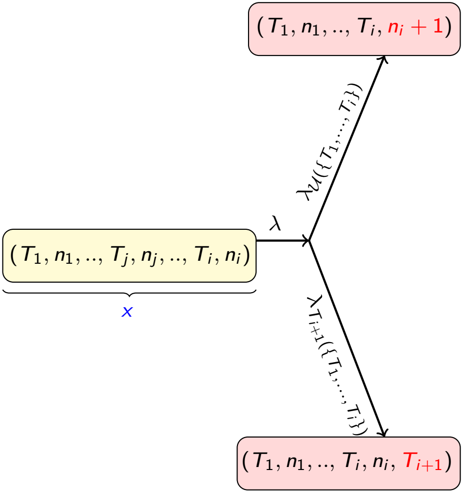
State transitions



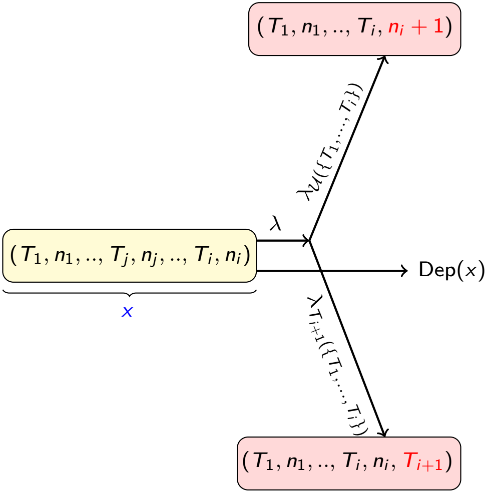
State transitions



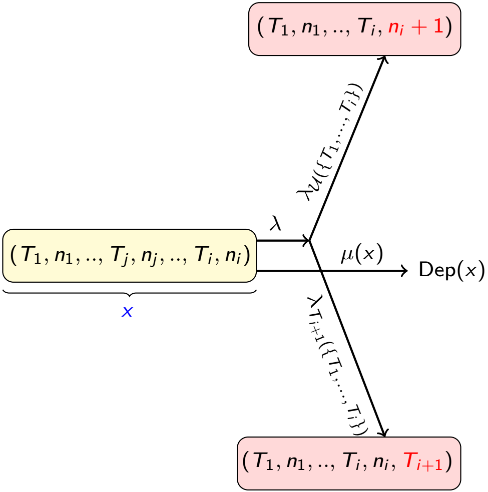
State transitions



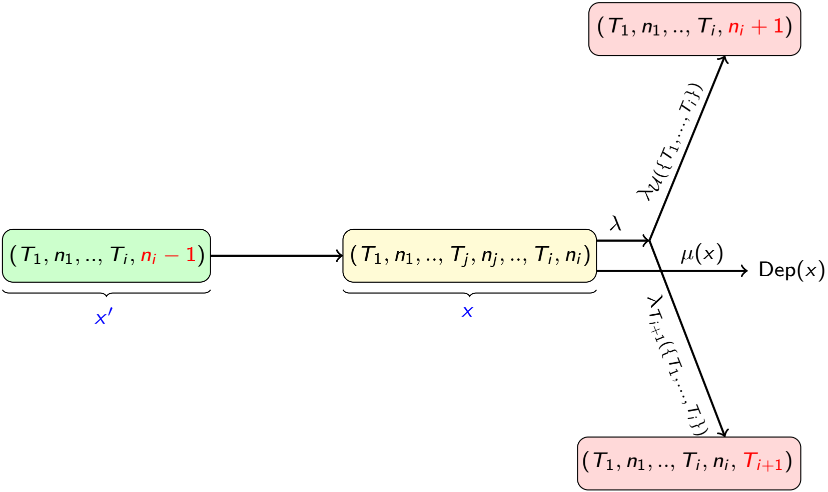
State transitions



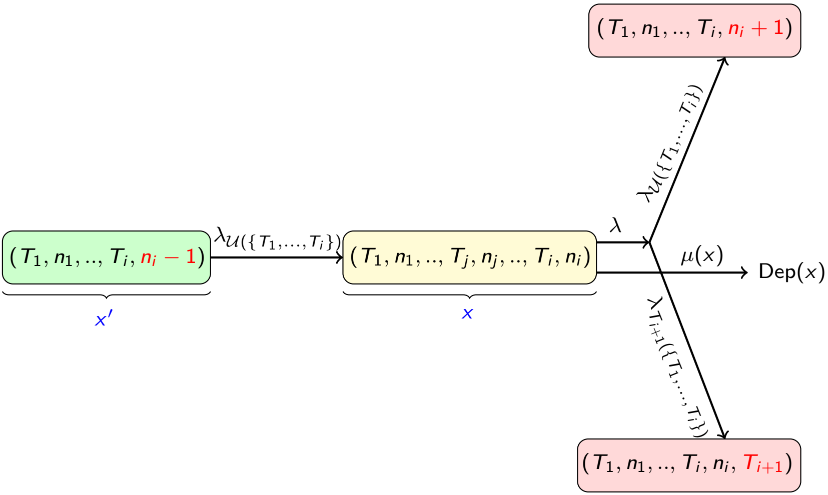
State transitions



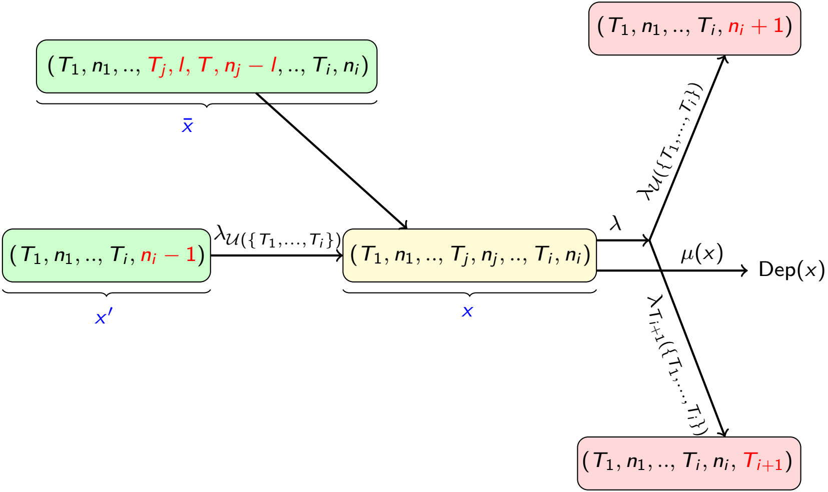
State transitions



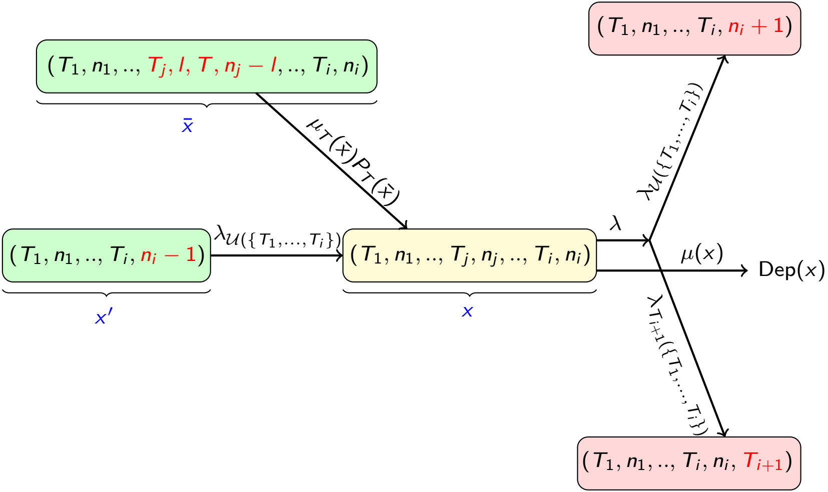
State transitions



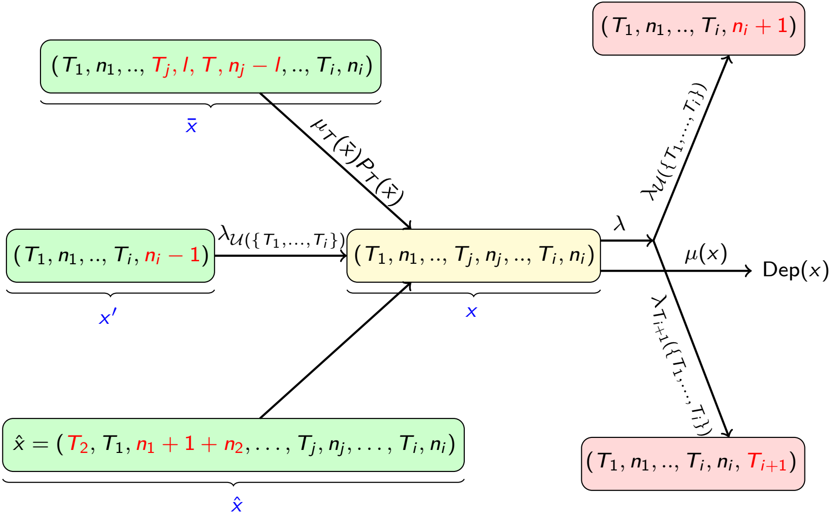
State transitions



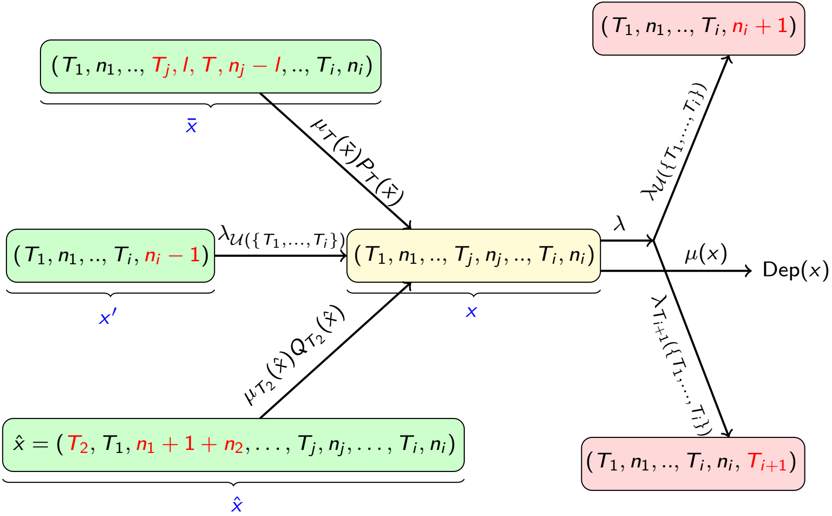
State transitions



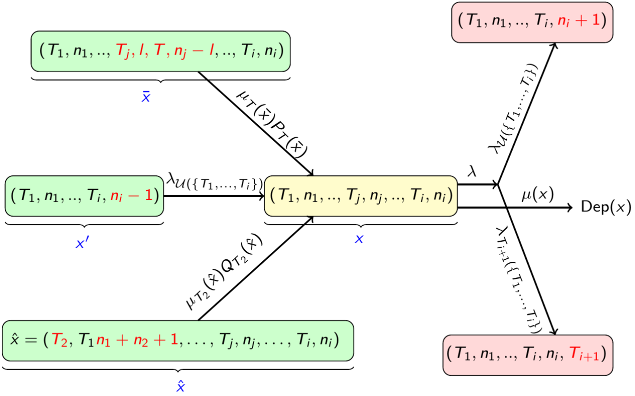
State transitions



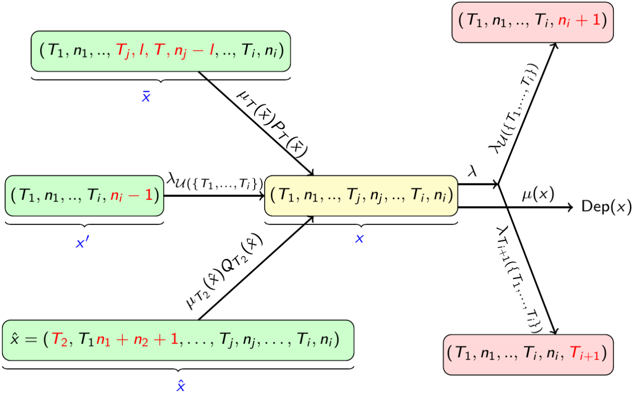
State transitions



Global balance equations (GBE) for stationary distribution

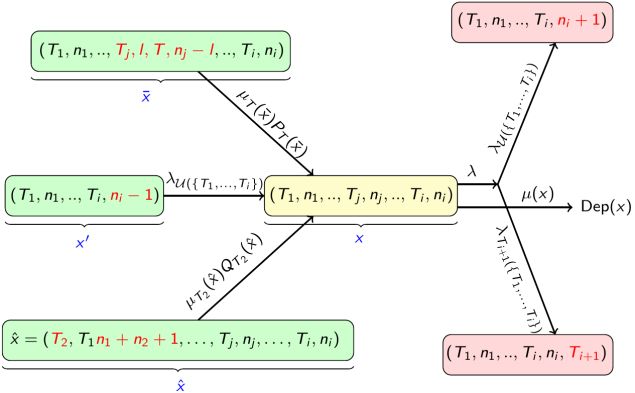


Global balance equations (GBE) for stationary distribution



- GBE: Average rate of leaving state x = Average rate of entering state x

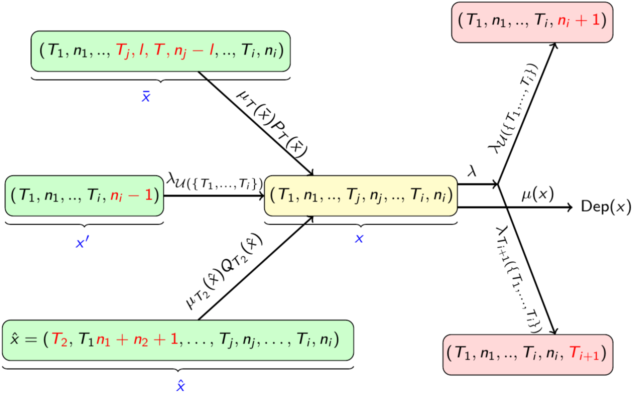
Global balance equations (GBE) for stationary distribution



- GBE: Average rate of leaving state x = Average rate of entering state x

$$\pi(x)\left[\lambda_{\mathcal{U}(\{\cdot\})} + \sum_{T_{i+1}} \lambda_{T_{i+1}}(\cdot) + \mu(x)\right]$$

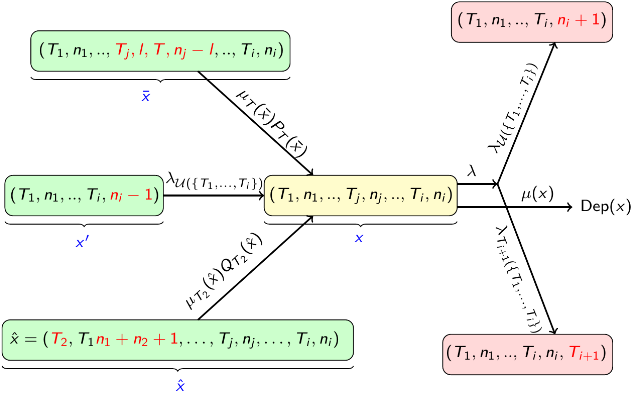
Global balance equations (GBE) for stationary distribution



- GBE: Average rate of leaving state x = Average rate of entering state x

$$\pi(x)\left[\lambda_{\mathcal{U}(\{\cdot\})} + \sum\nolimits_{T_{i+1}} \lambda_{T_{i+1}}(\cdot) + \mu(x)\right] = \pi(x')\lambda_{\mathcal{U}(\cdot)} + \sum \pi(\bar{x})\mu_T(\bar{x})P_T(\bar{x})$$
$$+ \sum \pi(\hat{x})\mu_{T_2}(\hat{x})Q_{T_2}(\hat{x})$$

Global balance equations (GBE) for stationary distribution



- GBE: Average rate of leaving state x = Average rate of entering state x

$$\pi(x)\left[\lambda_{\mathcal{U}(\{\cdot\})} + \sum_{T_{i+1}} \lambda_{T_{i+1}}(\cdot) + \mu(x)\right] = \pi(x')\lambda_{\mathcal{U}(\cdot)} + \sum \pi(\bar{x})\mu_T(\bar{x})P_T(\bar{x})$$
$$+ \sum \pi(\hat{x})\mu_{T_2}(\hat{x})Q_{T_2}(\hat{x})$$

- We solve this using partial balance equations (PBE's).

Main Result

Theorem

The steady state distribution for the token based model in state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$ is given by

Main Result

Theorem

The steady state distribution for the token based model in state $x = (T_1, n_1, \dots, T_j, n_j, \dots, T_i, n_i)$ is given by

$$\pi(x) = \pi(0) \ \prod_{j=1}^{i} \left(\frac{\lambda_{T_j}(\{T_1, \dots, T_{j-1}\})}{\sum_{l=1}^{j} \mu_{T_l}(x)} \right) \ \prod_{j=1}^{i} \left(\frac{\lambda_{\mathcal{U}(\{T_1, \dots, T_j\})}}{\sum_{l=1}^{j} \mu_{T_l}(x)} \right)^{n_j}$$

Main Result

Theorem

The steady state distribution for the token based model in state $x = (T_1, n_1, \dots, T_j, n_j, \dots, T_i, n_i)$ is given by

$$\pi(x) = \pi(0) \ \prod_{j=1}^{i} \left(\frac{\lambda_{T_j}(\{T_1,\dots,T_{j-1}\})}{\sum_{l=1}^{j} \mu_{T_l}(x)} \right) \ \prod_{j=1}^{i} \left(\frac{\lambda_{\mathcal{U}(\{T_1,\dots,T_j\})}}{\sum_{l=1}^{j} \mu_{T_l}(x)} \right)^{n_j}$$

normalising
constant

Main Result

Theorem

The steady state distribution for the token based model in state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$ is given by

$$\pi(x) = \underbrace{\pi(0)}_{\text{normalising constant}} \prod_{j=1}^{i} \underbrace{\left(\frac{\lambda_{T_j}(\{T_1, \ldots, T_{j-1}\})}{\sum_{l=1}^{j} \mu_{T_l}(x)} \right)}_{\text{term corresponding to to token } T_j} \prod_{j=1}^{i} \left(\frac{\lambda_{\mathcal{U}(\{T_1, \ldots, T_j\})}}{\sum_{l=1}^{j} \mu_{T_l}(x)} \right)^{n_j}$$

Main Result

Theorem

The steady state distribution for the token based model in state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$ is given by

$$\pi(x) = \underset{\text{normalising constant}}{\pi(0)} \; \prod_{j=1}^{i} \underbrace{\left(\frac{\lambda_{T_j}(\{T_1, \ldots, T_{j-1}\})}{\sum_{l=1}^{j} \mu_{T_l}(x)} \right)}_{\text{term corresponding to to token } T_j} \; \prod_{j=1}^{i} \underbrace{\left(\frac{\lambda_{\mathcal{U}(\{T_1, \ldots, T_j\})}}{\sum_{l=1}^{j} \mu_{T_l}(x)} \right)}_{\text{term corresponding to each of the } n_j \text{ waiting jobs}}^{n_j}$$

Main Result

Theorem

The steady state distribution for the token based model in state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$ is given by

$$\pi(x) = \underset{\substack{\downarrow\\ \text{normalising}\\ \text{constant}}}{\pi(0)} \; \prod_{j=1}^{i} \underbrace{\left(\frac{\lambda_{T_j}(\{T_1,\ldots,T_{j-1}\})}{\sum_{l=1}^{j} \mu_{T_l}(x)} \right)}_{\text{term corresponding to to token } T_j} \; \prod_{j=1}^{i} \underbrace{\left(\frac{\lambda_{\mathcal{U}(\{T_1,\ldots,T_j\})}}{\sum_{l=1}^{j} \mu_{T_l}(x)} \right)^{n_j}}_{\text{term corresponding to each of the } n_j \text{ waiting jobs}}$$

- PGF for the number of waiting jobs and the total number of jobs

Main Result

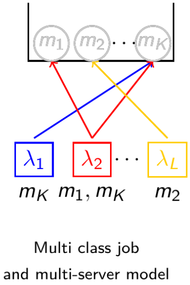
Theorem

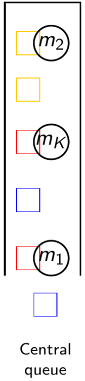
The steady state distribution for the token based model in state $x = (T_1, n_1, \dots, T_j, n_j, \dots, T_i, n_i)$ is given by

$$\pi(x) = \underbrace{\pi(0)}_{\text{normalising constant}} \prod_{j=1}^{i} \underbrace{\left(\frac{\lambda_{T_j}(\{T_1,\dots,T_{j-1}\})}{\sum_{l=1}^{j}\mu_{T_l}(x)}\right)}_{\text{term corresponding to to token } T_j} \prod_{j=1}^{i} \underbrace{\left(\frac{\lambda_{\mathcal{U}(\{T_1,\dots,T_j\})}}{\sum_{l=1}^{j}\mu_{T_l}(x)}\right)^{n_j}}_{\text{term corresponding to each of the } n_j \text{ waiting jobs}}$$

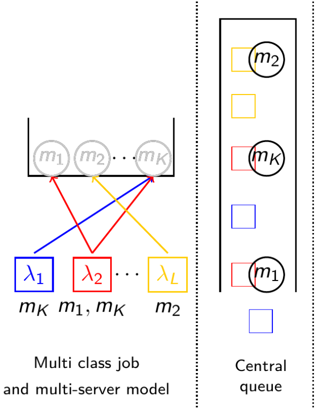
- PGF for the number of waiting jobs and the total number of jobs
- PGF for the waiting time and sojourn time for a job

Visschers and order-independent (OI) models



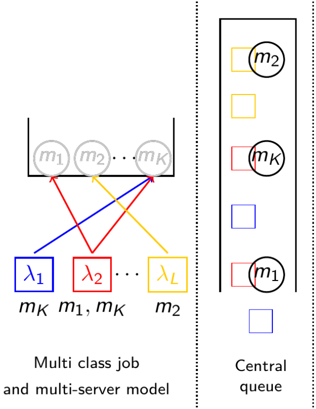


Visschers and order-independent (OI) models



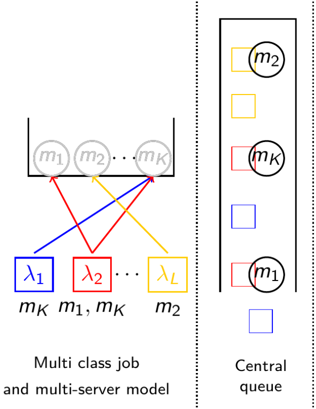
- Tokens replaced by servers

Visschers and order-independent (OI) models



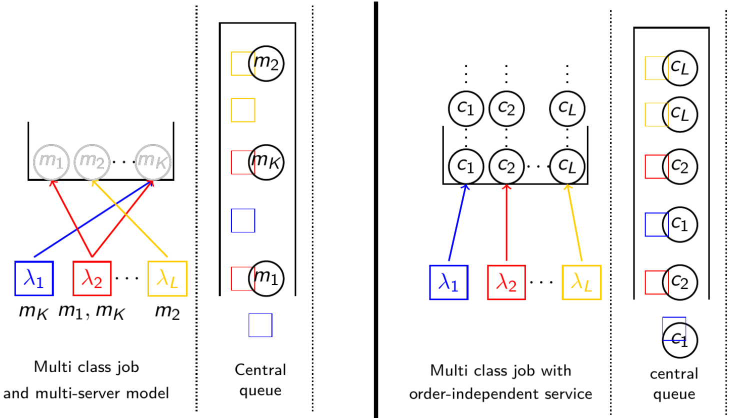
- Tokens replaced by servers
- Constant service rate

Visschers and order-independent (OI) models



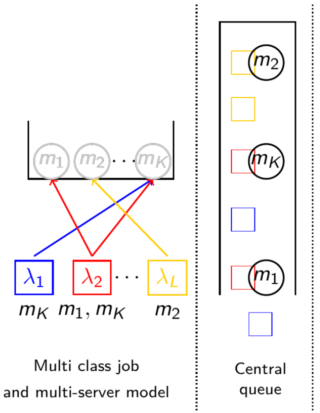
- Tokens replaced by servers
- Constant service rate
- *Assignment rule*

Visschers and order-independent (OI) models

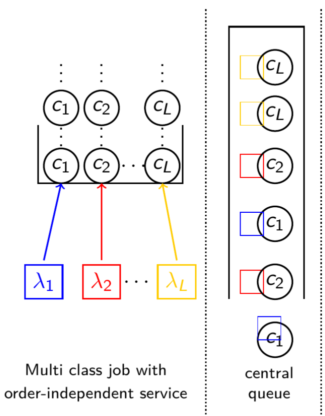


- Tokens replaced by servers
- Constant service rate
- *Assignment rule*

Visschers and order-independent (OI) models

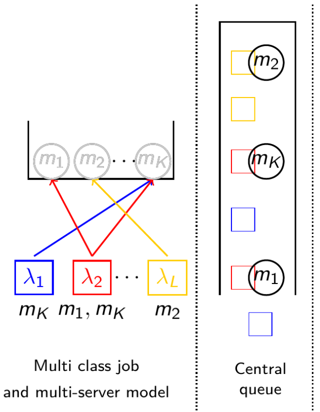


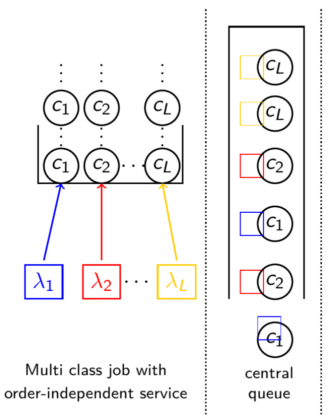
- Tokens replaced by servers
- Constant service rate
- *Assignment rule*



- Each job of class i gets a token c_i

Visschers and order-independent (OI) models

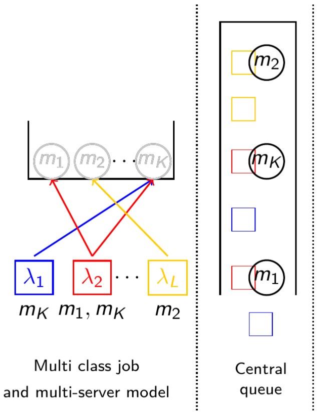


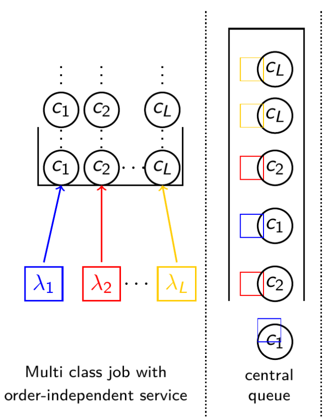


- Tokens replaced by servers
- Constant service rate
- *Assignment rule*

- Each job of class i gets a token c_i
- Order-independent (OI) service rate

Visschers and order-independent (OI) models

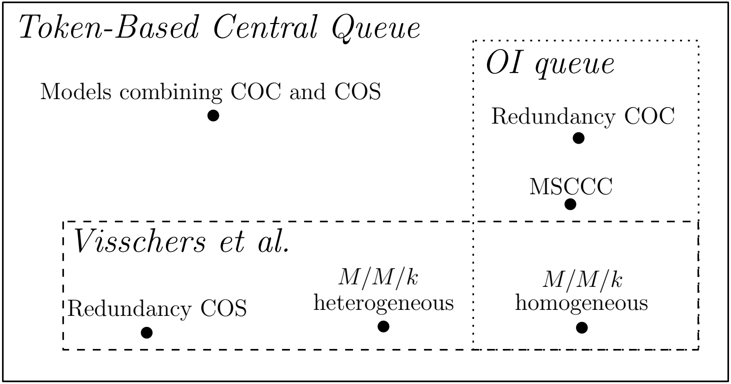




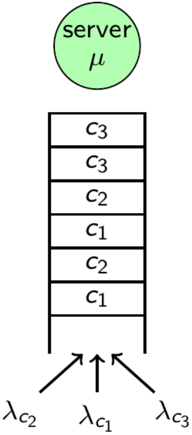
- Tokens replaced by servers
- Constant service rate
- *Assignment rule*

- Each job of class i gets a token c_i
- Order-independent (OI) service rate
- No *assignment rule*

A classification of token-based central queues

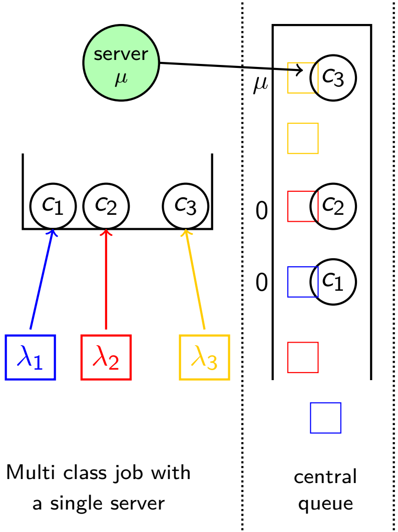


Example 1: Multiclass $M/M/1$ queue



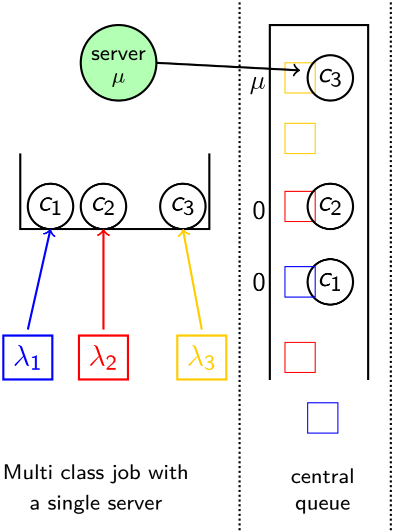
- single server with multiple classes
- exponential service requirement
- state space of the form $(c_3, c_3, c_2, c_1, c_2, c_1)$
- $\pi(c_3, c_3, c_2, c_1, c_2, c_1)$
  $= (1-\rho)\rho_{c_3}^2 \rho_{c_2} \rho_{c_1} \rho_{c_2} \rho_{c_1}$
  where $\rho_i = \frac{\lambda_i}{\mu}$

Example 1: Multiclass $M/M/1$ queue (token view)



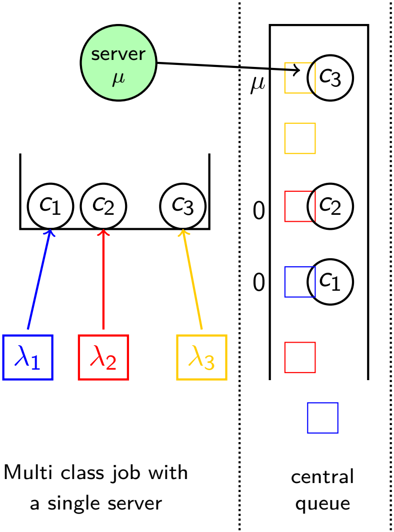
- We associate a unique token per class
- state space $(c_3, 1, c_2, c_1, 2)$
- Service rate μ for the first token
  service rate 0 for the rest

Example 1: Multiclass $M/M/1$ queue (token view)



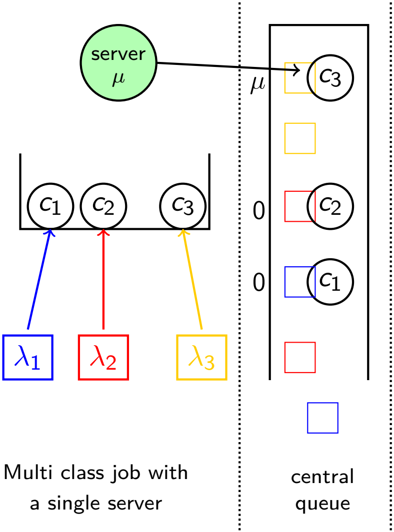
- We associate a unique token per class
- state space $(c_3, 1, c_2, c_1, 2)$
- Service rate μ for the first token
  service rate 0 for the rest

Example 1: Multiclass $M/M/1$ queue (token view)



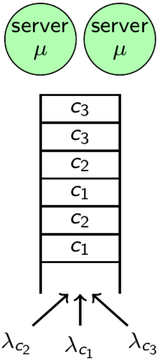
- We associate a unique token per class
- state space $(c_3, 1, c_2, c_1, 2)$
- Service rate μ for the first token
  service rate 0 for the rest
- $\pi(c_3, 1, c_2, c_1, 2) =$
  $(1-\rho)\left(\frac{\lambda_{c_3}}{\mu}\right)^2\left(\frac{\lambda_{c_2}}{\mu}\right)\left(\frac{\lambda_{c_1}}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^2$

Example 1: Multiclass $M/M/1$ queue (token view)



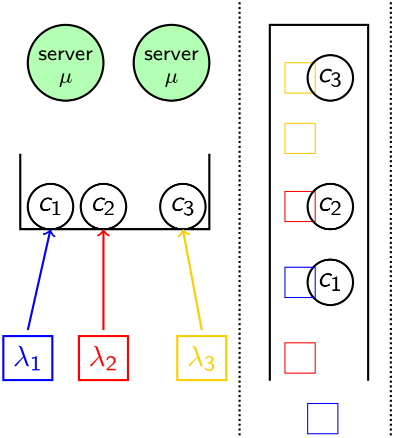
- We associate a unique token per class
- state space $(c_3, 1, c_2, c_1, 2)$
- Service rate μ for the first token
  service rate 0 for the rest
- $\pi(c_3, 1, c_2, c_1, 2) =$
  $(1-\rho)\left(\frac{\lambda_{c_3}}{\mu}\right)^2\left(\frac{\lambda_{c_2}}{\mu}\right)\left(\frac{\lambda_{c_1}}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^2$

Example 2: Multi-server station with concurrent control of customers (MSCCC)



- 2 identical servers and 3 classes
- Servers are compatible with all classes
- Both servers cannot serve the same class at any time (concurrent control)

Example 2: Multi-server station with concurrent control of customers (MSCCC)

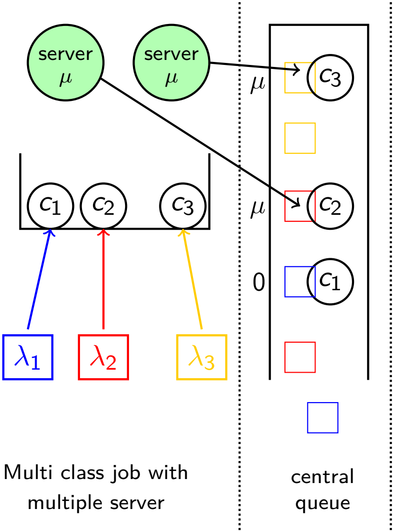


Multi class job with multiple server

central queue

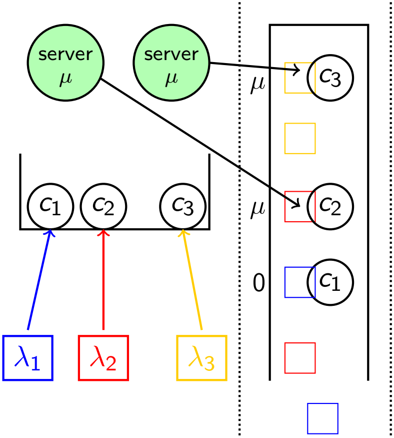
- One token per class as earlier
- At most 1 job per class is served (concurrent control)
- Two server, three tokens
- Associate the two servers with the first 2 active tokens

Example 2: Multi-server station with concurrent control of customers (MSCCC)



- One token per class as earlier
- At most 1 job per class is served (concurrent control)
- Two server, three tokens
- Associate the two servers with the first 2 active tokens
- Order-independent service rate where only the first 2 token have a service rate μ

Example 2: Multi-server station with concurrent control of customers (MSCCC)

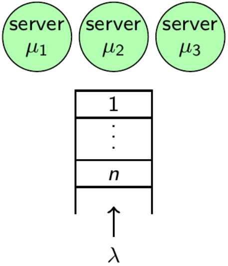


Multi class job with multiple server

central queue

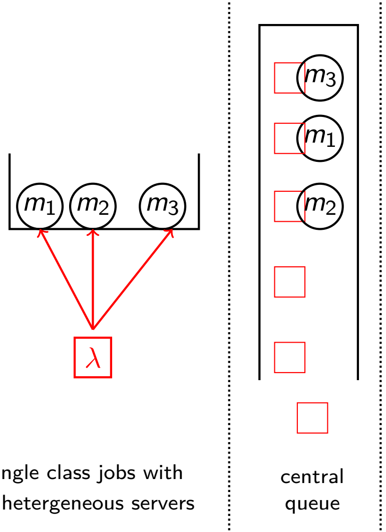
- One token per class as earlier
- At most 1 job per class is served (concurrent control)
- Two server, three tokens
- Associate the two servers with the first 2 active tokens
- Order-independent service rate where only the first 2 token have a service rate μ
- $\pi(c_3, 1, c_2, c_1, 2) = (1-\rho)\left(\frac{\lambda_{c_3}}{\mu}\right)^2 \left(\frac{\lambda_{c_2}}{2\mu}\right) \left(\frac{\lambda_{c_1}}{2\mu}\right) \left(\frac{\lambda}{2\mu}\right)^2$

Example 3: $M/M/K$ queue



- One class of jobs and K possibly heterogeneous servers
- At most 3 jobs can be served at a time
- Also known as *Erlang-K* model
- When servers are identical, state space denoted by number of jobs in the system
- $\pi(n) = \pi(0)\frac{\rho^n K^K}{K!}$ where $\rho = \frac{\lambda}{K\mu}$

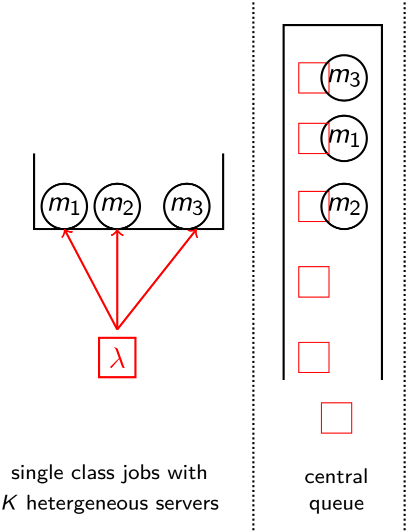
Example 3: $M/M/K$ queue



single class jobs with
K hetergeneous servers

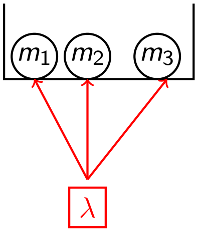
central
queue

Example 3: $M/M/K$ queue

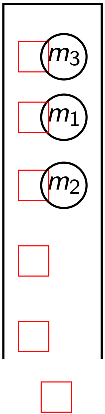


- Single class & token m_i for server i

Example 3: $M/M/K$ queue



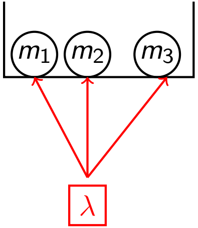
single class jobs with
K hetergeneous servers



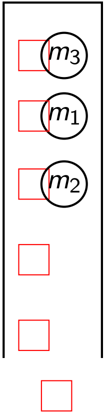
central
queue

- Single class & token m_i for server i
- Arriving jobs are compatible with all tokens

Example 3: $M/M/K$ queue



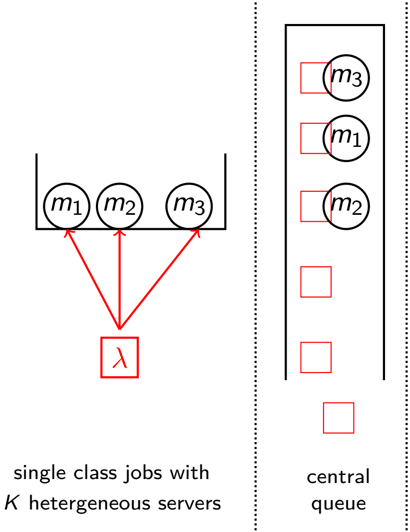
single class jobs with
K hetergeneous servers



central
queue

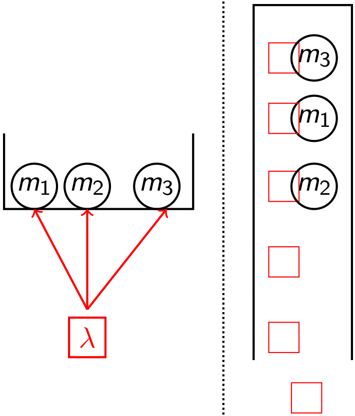
- Single class & token m_i for server i
- Arriving jobs are compatible with all tokens
- *Assignment rule* is *uniform*

Example 3: $M/M/K$ queue



- Single class & token m_i for server i
- Arriving jobs are compatible with all tokens
- *Assignment rule* is *uniform*
- $(m_3, m_1, m_2, 3)$

Example 3: $M/M/K$ queue



single class jobs with
K hetergeneous servers

central
queue

- Single class & token m_i for server i
- Arriving jobs are compatible with all tokens
- *Assignment rule* is *uniform*
- $(m_3, m_1, m_2, 3)$
- $\pi(m_3, m_1, m_2, 3) =$

$$\pi(0)\frac{\lambda}{3\mu_3}\frac{\lambda}{2(\mu_3+\mu_1)}\frac{\lambda}{(\mu_3+\mu_1+\mu_2)}\left(\frac{\lambda}{(\mu_3+\mu_1+\mu_2)}\right)^3$$

Outline

- Redundancy
- Central Queue Architecture
- Order Independent Descriptor
  - Redundancy and cancel on complete
- Aggregated State Descriptor
  - Redundancy and cancel on start
- Generalizations:
  - Token-based framework
  - Generalized Order Independent
- Impact of assumptions: scheduling and independence

Generalized OI queue[13]

Detailed state descriptor $\vec{z}_m$, where m is number of jobs

- It includes jobs in service and in the queue

service $\Longrightarrow$ track server
queue $\Longrightarrow$ track class

[13] K. Gardner, R. Righter, Product Forms for FCFS Queueing Models with Arbitrary Server-Job Compatibilities: An Overview, to appear in QUESTA

Generalized OI queue[13]

Detailed state descriptor $\vec{z}_m$, where m is number of jobs

- It includes jobs in service and in the queue

service $\Longrightarrow$ track server
queue $\Longrightarrow$ track class

- $\mu(\vec{z}_m)$ satisfies the OI Properties

Theorem: The steady-state distribution of the Generalized OI queue is product form

$\Longrightarrow$ Unifying framework for product-form distributions

[13] K. Gardner, R. Righter, Product Forms for FCFS Queueing Models with Arbitrary Server-Job Compatibilities: An Overview, to appear in QUESTA

Outline

- Redundancy
- Central Queue Architecture
- Order Independent Descriptor
  - Redundancy and cancel on complete
- Aggregated State Descriptor
  - Redundancy and cancel on start
- Generalizations:
  - Token-based framework
  - Generalized Order Independent
- Impact of assumptions: scheduling and independence

Stability: Impact of independence assumption

- With i.i.d. copies and FCFS, redundancy does not impact stability.
  $\Longrightarrow$ papers by Gardner et al., Bonald et al.

$$d = K \quad \Longrightarrow \textit{single server with rate } \mu K$$
$$d = 1 \quad \Longrightarrow \quad K \textit{ indep. single servers with rate } \mu$$

Stability: Impact of independence assumption

- With i.i.d. copies and FCFS, redundancy does not impact stability.
  $\Longrightarrow$ papers by Gardner et al., Bonald et al.

$$\begin{array}{lll} d = K & \Longrightarrow & \textit{single server with rate } \mu K \\ d = 1 & \Longrightarrow & K \textit{ indep. single servers with rate } \mu \end{array}$$

- What without i.i.d. assumption?

$$\begin{array}{lll} d = K & \Longrightarrow & \textit{single server with rate } \mu \\ d = 1 & \Longrightarrow & K \textit{ indep. single servers with rate } \mu \end{array}$$

Stability of redundancy: Impact of assumptions

Most of existing literature is with i.i.d. copies and FCFS
$\Longrightarrow$ stability not reduced
yield results that are qualitatively misleading.
Need of better models: $S\&X$[14]

[14] Gardner, Harchol-Balter, Scheller-Wolf, Van Houdt, A Better Model for Job Redundancy: Decoupling Server Slowdown and Job Size, IEEE/ACM ToN, 2017

Stability of redundancy: Impact of assumptions

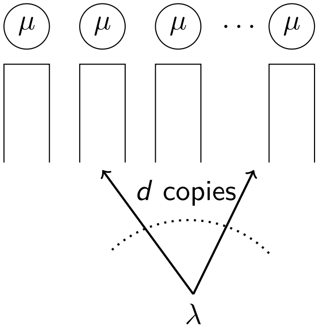
Most of existing literature is with i.i.d. copies and FCFS
$\Longrightarrow$ stability not reduced
yield results that are qualitatively misleading.
Need of better models: $S\&X$[14]

Main questions:

- How does redundancy with identical copies impact stability?
- Does stability depend on the scheduling discipline?

[14] Gardner, Harchol-Balter, Scheller-Wolf, Van Houdt, A Better Model for Job Redundancy: Decoupling Server Slowdown and Job Size, IEEE/ACM ToN, 2017

Redundancy d

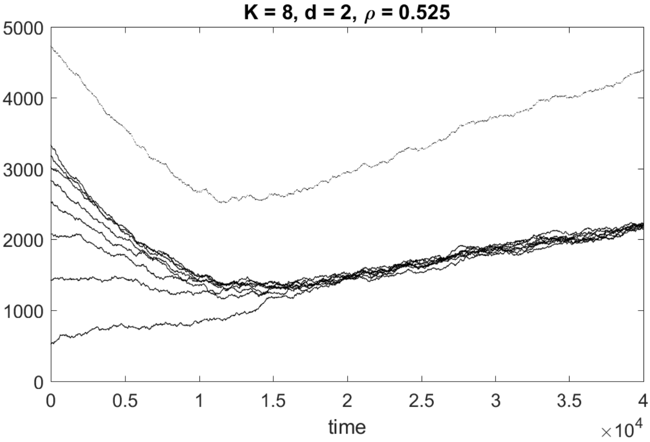


We study[15]:

- *c.o.c.* with identical copies
- scheduling discipline in servers can be PS, FCFS, or ROS.

[15] E. Anton, U. Ayesta, M. Jonckheere, I.M. Verloop. On the stability of redundancy models, to appear in OR

Instability with identical copies



Neither total number nor minimum are Lyapunov functions

Scheduling disciplines

- FCFS.
- Processor-Sharing (PS). The capacity is shared equally among all copies
- Random Order of Service (ROS)

[16] Baccelli, Foss, On the Saturation Rule for the Stability of Queues, JAP, 1995

Scheduling disciplines

- FCFS.
- Processor-Sharing (PS). The capacity is shared equally among all copies
- Random Order of Service (ROS)

FCFS. Stability through saturated system[16]

[16] Baccelli, Foss, On the Saturation Rule for the Stability of Queues, JAP, 1995

Scheduling disciplines

- FCFS.
- Processor-Sharing (PS). The capacity is shared equally among all copies
- Random Order of Service (ROS)

FCFS. Stability through saturated system[16]
For PS.

Lower Bound: *Every instant, put the copy with highest attained service, in the server with smallest number of copies.*
Upper Bound: *All copies need to be served for the job to be completed.*

[16] Baccelli, Foss, On the Saturation Rule for the Stability of Queues, JAP, 1995

Summary of stability conditions

	PS	FCFS	ROS	Priority policy
i.i.d	$\lambda < \mu K$	$\lambda < \mu K$	$\lambda < \mu K$	$\lambda << \mu K$
i.c.	$\lambda < \mu \frac{K}{d}$	$\lambda < \bar{\ell}\mu$	$\lambda < \mu K$	–

[17] Y. Raaijmakers, S. Borst, O. Boxma: Delta probing policies for redundancy. Perform. Eval (2018)

Summary of stability conditions

	PS	FCFS	ROS	Priority policy
i.i.d	$\lambda < \mu K$	$\lambda < \mu K$	$\lambda < \mu K$	$\lambda << \mu K$
i.c.	$\lambda < \mu \frac{K}{d}$	$\lambda < \bar{\ell}\mu$	$\lambda < \mu K$	–

- With PS, stability condition is the same as if all copies had to be served.
- The stability region is larger for FCFS than for PS.
- ROS. No stability reduction.

[17] Y. Raaijmakers, S. Borst, O. Boxma: Delta probing policies for redundancy. Perform. Eval (2018)

Summary of stability conditions

	PS	FCFS	ROS	Priority policy
i.i.d	$\lambda < \mu K$	$\lambda < \mu K$	$\lambda < \mu K$	$\lambda << \mu K$
i.c.	$\lambda < \mu \frac{K}{d}$	$\lambda < \bar{\ell}\mu$	$\lambda < \mu K$	–

- With PS, stability condition is the same as if all copies had to be served.
- The stability region is larger for FCFS than for PS.
- ROS. No stability reduction.
- Stability depends on scheduling discipline.

[17] Y. Raaijmakers, S. Borst, O. Boxma: Delta probing policies for redundancy. Perform. Eval (2018)

Summary of stability conditions

	PS	FCFS	ROS	Priority policy
i.i.d	$\lambda < \mu K$	$\lambda < \mu K$	$\lambda < \mu K$	$\lambda << \mu K$
i.c.	$\lambda < \mu \frac{K}{d}$	$\lambda < \bar{\ell}\mu$	$\lambda < \mu K$	–

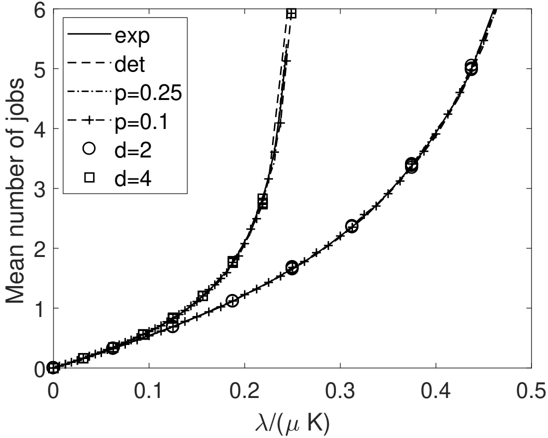
- With PS, stability condition is the same as if all copies had to be served.
- The stability region is larger for FCFS than for PS.
- ROS. No stability reduction.
- Stability depends on scheduling discipline.

Need to develop strategies that preserve stability[17]

[17] Y. Raaijmakers, S. Borst, O. Boxma: Delta probing policies for redundancy. Perform. Eval (2018)

Non-exponential service requirements: PS[18]

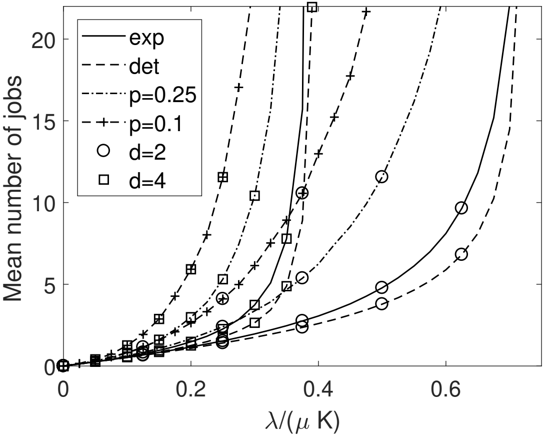
Mean number of jobs with identical copies and exponential, deterministic and degenerate hyperexponential service requirements.



[18] PG Taylor, Insensitivity in Stochastic Models, in "Queueing Networks: a fundamental approach", Eds: R. Boucherie, N. van Dijk, 2011.

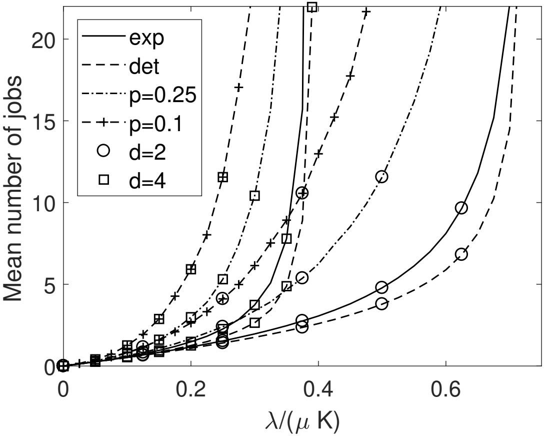
Non-exponential service requirements: FCFS

Mean number of jobs with identical copies and exponential, deterministic and degenerate hyperexponential service requirements.



Non-exponential service requirements: FCFS

Mean number of jobs with identical copies and exponential, deterministic and degenerate hyperexponential service requirements.



Approximations developed by: I Adan, M. Boon, G. Weiss, work in progress

What about heterogeneous systems? [19]

Assume $\mu_1 < \ldots < \mu_K$, redundancy-d.

- With redundancy, the stability condition is $\lambda^R = \min_{i=d,\ldots,K}\left\{\mu_i \frac{\binom{K}{d}}{\binom{i-1}{d-1}}\right\}$.

[19] E. Anton, U. Ayesta, M. Jonckheere, I. M.Verloop. Improving the Performance of Heterogeneous Data Centers through Redundancy, to appear in ACM SIGMETRICS 2021

What about heterogeneous systems? [19]

Assume $\mu_1 < \ldots < \mu_K$, redundancy-d.

- With redundancy, the stability condition is $\lambda^R = \min_{i=d,\ldots,K} \left\{ \mu_i \frac{\binom{K}{d}}{\binom{i-1}{d-1}} \right\}$.
- With Bernouilli routing, the stability condition is $\lambda^B = K\mu_1$.

[19] E. Anton, U. Ayesta, M. Jonckheere, I. M.Verloop. Improving the Performance of Heterogeneous Data Centers through Redundancy, to appear in ACM SIGMETRICS 2021

What about heterogeneous systems? [19]

Assume $\mu_1 < \ldots < \mu_K$, redundancy-d.

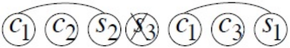
- With redundancy, the stability condition is $\lambda^R = \min_{i=d,\ldots,K} \left\{ \mu_i \frac{\binom{K}{d}}{\binom{i-1}{d-1}} \right\}$.
- With Bernouilli routing, the stability condition is $\lambda^B = K\mu_1$.

Redundancy-d has larger stability region than Bernoulli if $\mu_1 d < \mu_d$.

[19] E. Anton, U. Ayesta, M. Jonckheere, I. M.Verloop. Improving the Performance of Heterogeneous Data Centers through Redundancy, to appear in ACM SIGMETRICS 2021

Relation with matching models

Model A:[20] :Arriving jobs wait in the queue for a compatible server, arriving servers match the first compatible job and leave the system (even if unmatched)

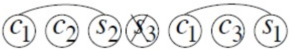


[20] I. Adan and R. Righter and G. Weiss, FCFS Parallel Service Systems and Matching Models, Valuetools 2017

[21] I. Adan, A. Busic, J. Mairesse, and G. Weiss. 2015. Reversibility and further properties of FCFS infinite bipartite matching.Math. Oper. Res.

Relation with matching models

Model A:[20] :Arriving jobs wait in the queue for a compatible server, arriving servers match the first compatible job and leave the system (even if unmatched)



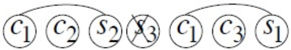
Same distribution as OI queues

[20] I. Adan and R. Righter and G. Weiss, FCFS Parallel Service Systems and Matching Models, Valuetools 2017

[21] I. Adan, A. Busic, J. Mairesse, and G. Weiss. 2015. Reversibility and further properties of FCFS infinite bipartite matching.Math. Oper. Res.

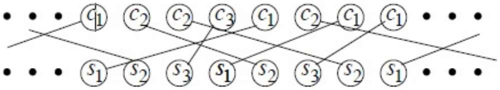
Relation with matching models

Model A:[20] :Arriving jobs wait in the queue for a compatible server, arriving servers match the first compatible job and leave the system (even if unmatched)



Same distribution as OI queues

Model B:[21] (FCFS infinite bipartite matching): arrivals are job server pairs, both of which can queue; arriving jobs (servers) match the first compatible server (job) if any, otherwise join queue



[20] I. Adan and R. Righter and G. Weiss, FCFS Parallel Service Systems and Matching Models, Valuetools 2017

[21] I. Adan, A. Busic, J. Mairesse, and G. Weiss. 2015. Reversibility and further properties of FCFS infinite bipartite matching.Math. Oper. Res.

Conclusions and Perspectives

- Establish relation between multi-server systems with central queue: redundancy, JSW, etc

Conclusions and Perspectives

- Establish relation between multi-server systems with central queue: redundancy, JSW, etc
- Token based and Gen-OI provide unifying framework to cover OI and "multi-job multi-server" models

Conclusions and Perspectives

- Establish relation between multi-server systems with central queue: redundancy, JSW, etc
- Token based and Gen-OI provide unifying framework to cover OI and "multi-job multi-server" models
- Product form distribution does not directly imply computability
  $\Longrightarrow$ computation of $\pi(0)$.

Conclusions and Perspectives

- Establish relation between multi-server systems with central queue: redundancy, JSW, etc
- Token based and Gen-OI provide unifying framework to cover OI and "multi-job multi-server" models
- Product form distribution does not directly imply computability
  $\Longrightarrow$ computation of $\pi(0)$.
- Not final word on product form distributions yet...
  $\Longrightarrow$ Pass & Swap queues (Comte& Dorsman 2020) have product form, yet not included in Token or Gen-OI

Conclusions and Perspectives

- Most of the analysis on redundancy assumes FCFS
  - Need to consider other classical disciplines
  - Develop new redundancy-aware scheduling disciplines

Conclusions and Perspectives

- Most of the analysis on redundancy assumes FCFS
  - Need to consider other classical disciplines
  - Develop new redundancy-aware scheduling disciplines
- Much less known for non exponential service times

Conclusions and Perspectives

- Most of the analysis on redundancy assumes FCFS
  - Need to consider other classical disciplines
  - Develop new redundancy-aware scheduling disciplines
- Much less known for non exponential service times
- Exploit steady-state distribution to characterize heavy-traffic, mean-field results, asymptotic optimality etc.
  $\Longrightarrow$ E. Cardinaels, S. Borst, J.H. van Leeuwaarden, Redundancy Scheduling with Locally Stable Compatibility Graphs, arxiv 2020

Conclusions and Perspectives

- Most of the analysis on redundancy assumes FCFS
  - Need to consider other classical disciplines
  - Develop new redundancy-aware scheduling disciplines
- Much less known for non exponential service times
- Exploit steady-state distribution to characterize heavy-traffic, mean-field results, asymptotic optimality etc.
  $\Longrightarrow$ E. Cardinaels, S. Borst, J.H. van Leeuwaarden, Redundancy Scheduling with Locally Stable Compatibility Graphs, arxiv 2020
- Relation between JSQ(d) and Redundancy
  $\Longrightarrow$ Coming up SNAPP talk by S. Borst

(Partial) Bibliography

Redundancy:
K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf. 2016. Queueing with redundant requests: exact analysis. Queueing Systems 83, 3-4 (2016), 227-259.
K. Gardner, M. Harchol-Balter, A. Scheller-Wolf, M. Velednitsky, S. Zbarsky, Redundancy-d: The power of d choices for redundancy, Operations Research, 2017

Multi-type job and servers:
Visschers, Adan, Weiss, *A product form solution to a system with multi-type jobs and multi-type servers, Queueing Systems, 2012.*
Adan, Weiss, A skill based parallel service system under FCFS-ALIS: steady state, overloads, and abandonments Stochastic Systems, INFORMS, 2014, 4, 250-299

***OI** Queues:*
A. Krzesinski, Order independent queues, in "Queueing Networks: a fundamental approach", eds: R. Boucherie, N. van Dijk, 2011.

(Partial) Bibliography

Redundancy and product form
Ayesta, Bodas, Verloop, On a unifying product form framework for redundancy models, Performance Evaluation, 2018
T. Bonald, C. Comte, F Mathieu, Performance of Balanced Fairness in Resource Pools: A Recursive Approach, Sigmetrics 2017
K. Gardner and R. Righter, Product (Re)forms, Tutorial APS 2019, available at: `https://kgardner.people.amherst.edu/`

Mean-Field and Heavy-Traffic:
M. Bramson, Yi Lu, B. Prabhakar, Randomized load balancing with general service time distributions. SIGMETRICS 2010: 275-286
T. Hellemans, B. Van Houdt, On the Power-of-d-choices with Least Loaded Server Selection, SIGMETRICS 2018
E. Cardinaels, S. Borst, J.H. van Leeuwaarden, Redundancy Scheduling with Locally Stable Compatibility Graphs, arxiv 2020

Generalizations:
Ayesta, Bodas, Dorsman, Verloop, A token-based central queue with order-independent service rates , to appear in OR
K. Gardner, R. Righter, Product Forms for FCFS Queueing Models with Arbitrary Server-Job Compatibilities: An Overview, to appear in QUESTA

(Partial) Bibliography

Stability:
E. Anton, U. Ayesta, M. Jonckheere, I. M.Verloop. On the stability of redundancy models, to appear in OR
Y. Raaijmakers, S. Borst, O. Boxma, Delta probing policies for redundancy. Perform. Eval (2018)
E. Anton, U. Ayesta, M. Jonckheere, I. M.Verloop. Improving the Performance of Heterogeneous Data Centers through Redundancy, to appear in ACM SIGMETRICS 2021

Scheduling and efficiency in redundancy:
K. Gardner, M. Harchol-Balter, E. Hyytiä, R. Righter: Scheduling for efficiency and fairness in systems with redundancy. Perform. Evaluation 116: 1-25 (2017)
Y. Raaijmakers, S. Borst, O. Boxma, Delta probing policies for redundancy. Perform. Eval (2018)

Matching:
I. Adan and R. Righter and G. Weiss, FCFS Parallel Service Systems and Matching Models, Valuetools 2017
I. Adan, A. Busic, J. Mairesse, and G. Weiss. 2015. Reversibility and further properties of FCFS infinite bipartite matching.Math. Oper. Res. 43(2): 598-621 (2018)
J. Mairesse, P. Moyal: Stability of the stochastic matching model. J. Appl. Probab. 2016

Load balancing, redundancy, and multi type job and server systems

Urtzi Ayesta

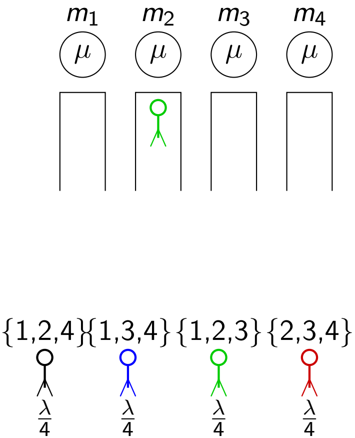
CNRS & Ikerbasque-Univ. Basque Country

Questions/Remarks welcome at : urtzi.ayesta@irit.fr

IFIP Performance 2020, 2/11/2020

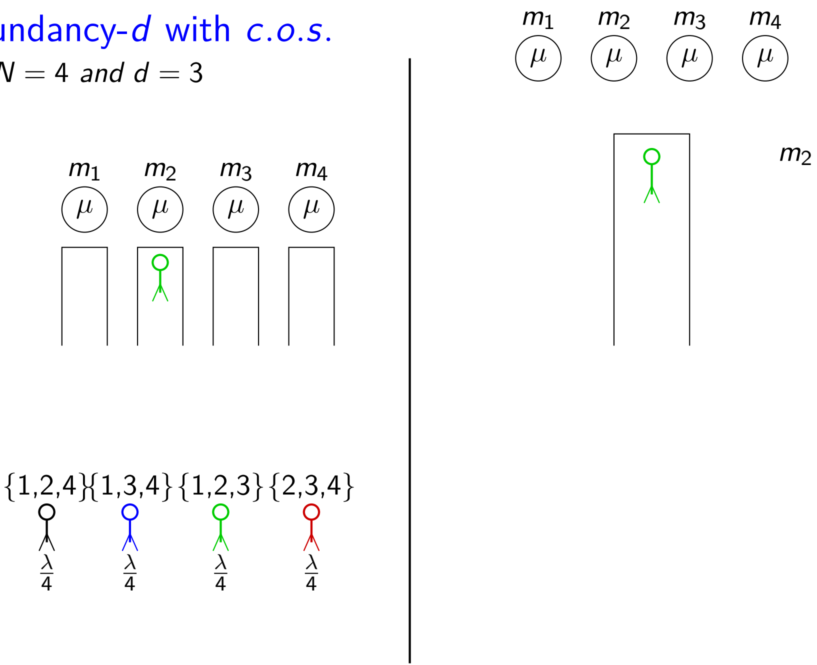
redundancy-d with $c.o.s.$

$N = 4$ *and* $d = 3$



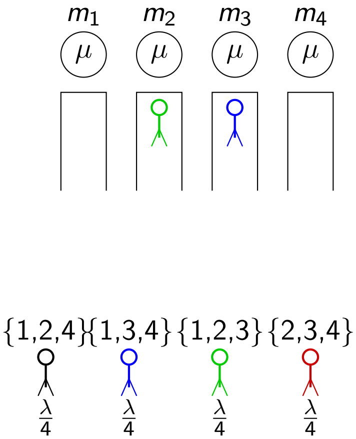
redundancy-d with $c.o.s.$

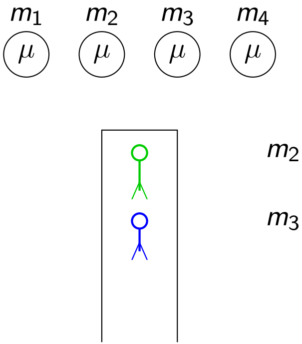
$N = 4$ *and* $d = 3$



redundancy-d with $c.o.s.$

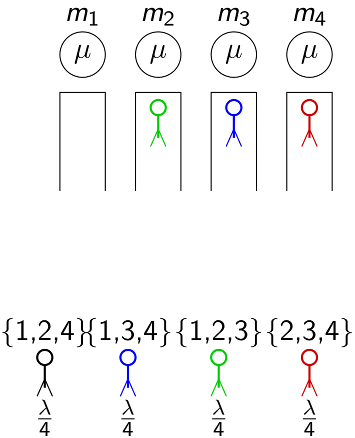
$N = 4$ and $d = 3$

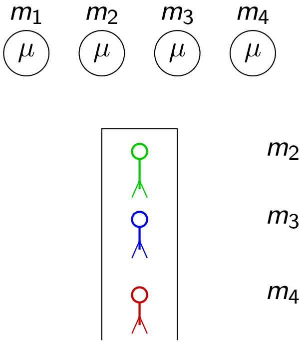




redundancy-d with $c.o.s.$

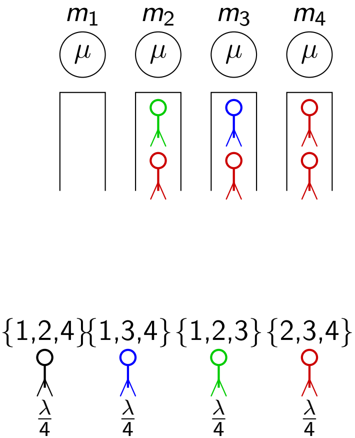
$N = 4$ *and* $d = 3$

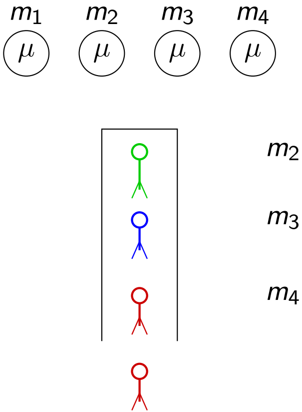




redundancy-d with $c.o.s.$

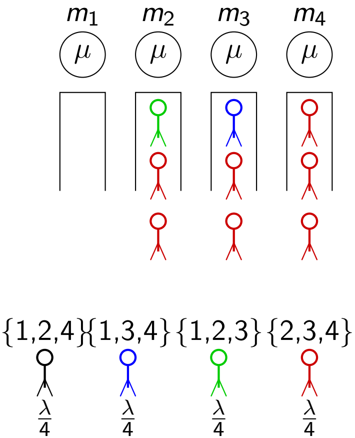
$N = 4$ *and* $d = 3$

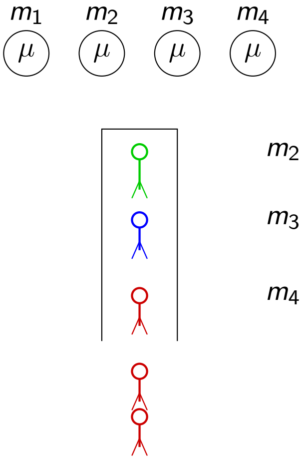




redundancy-d with $c.o.s.$

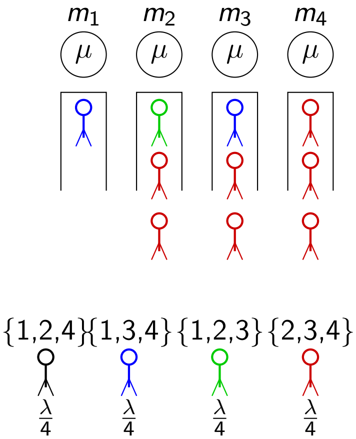
$N = 4$ *and* $d = 3$

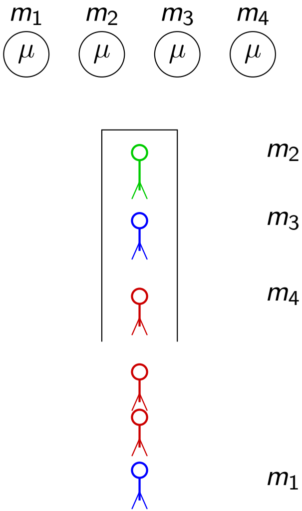




redundancy-d with $c.o.s.$

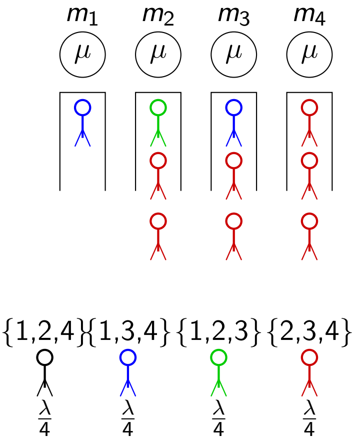
$N = 4$ *and* $d = 3$

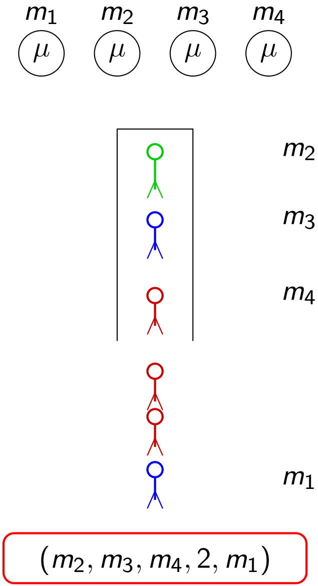




redundancy-d with $c.o.s.$

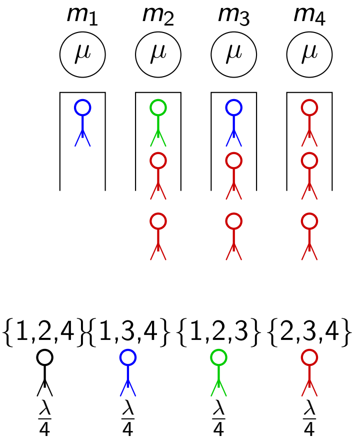
$N = 4$ *and* $d = 3$

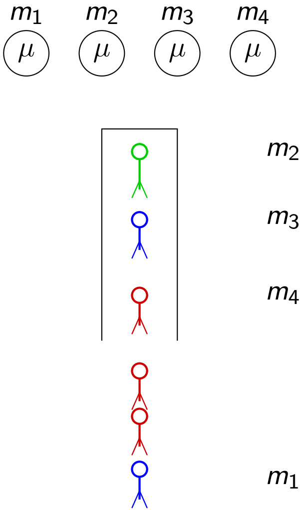




redundancy-d with $c.o.s.$

$N = 4$ *and* $d = 3$

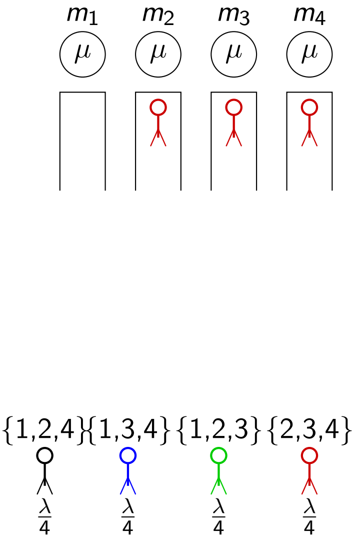




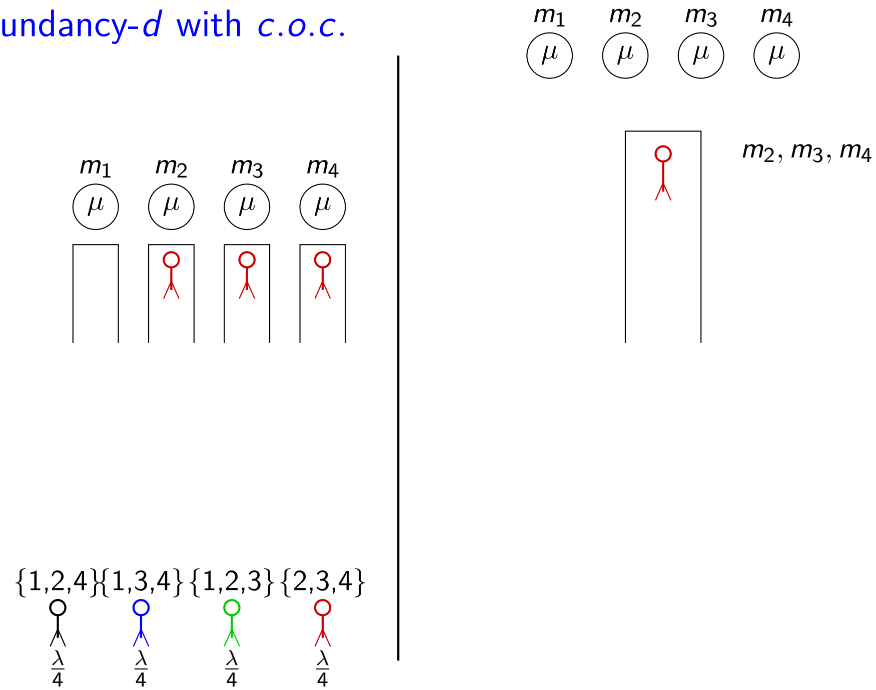
$$\boxed{(m_2, m_3, m_4, 2, m_1)}$$

$$\boxed{(M_1, \ldots, M_d, n_d, M_{d+1}, \ldots, n_i, M_i)}$$

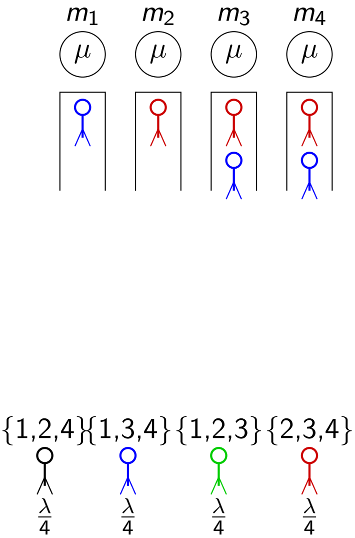
redundancy-d with $c.o.c.$

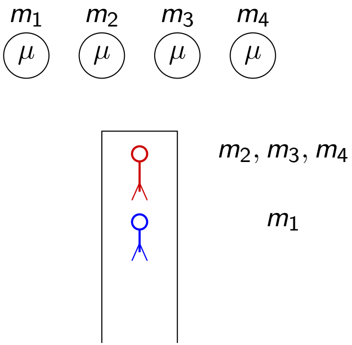


redundancy-d with $c.o.c.$

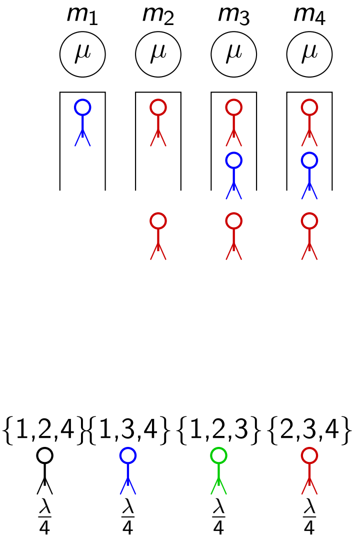


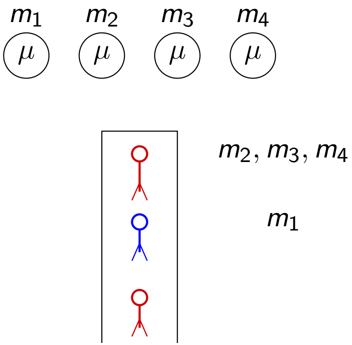
redundancy-d with $c.o.c.$



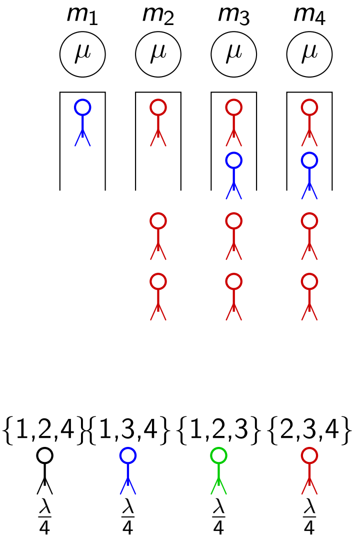


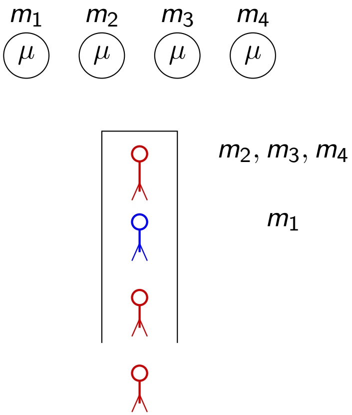
redundancy-d with $c.o.c.$



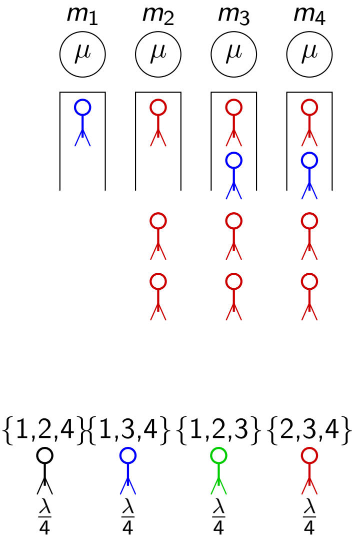


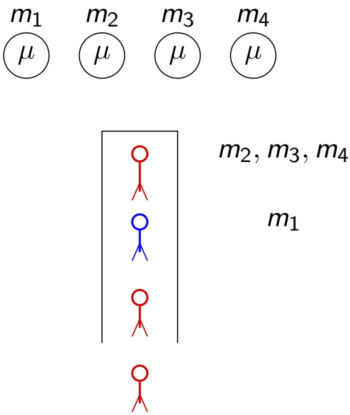
redundancy-d with $c.o.c.$





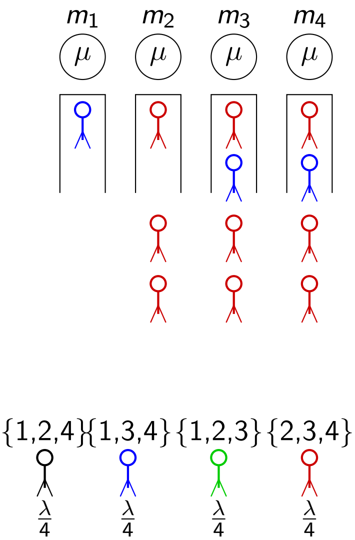
redundancy-d with $c.o.c.$

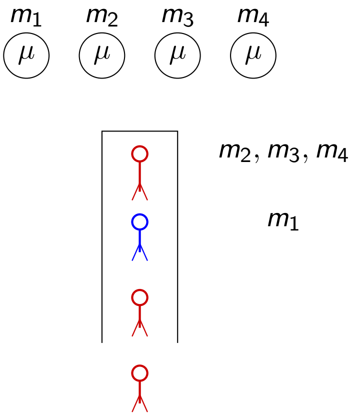




$(F_{red}, F_{blue}, 2)$

redundancy-d with $c.o.c.$

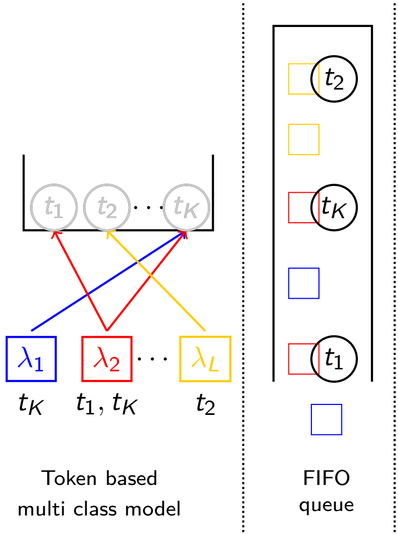




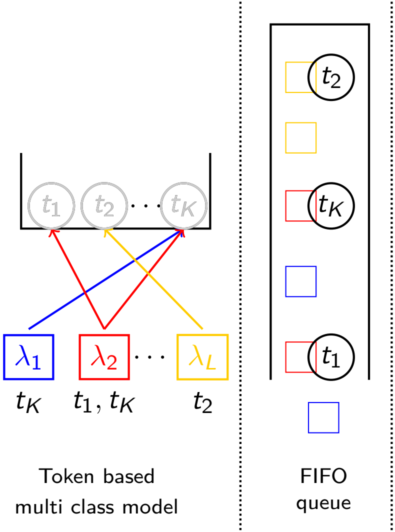
$$\boxed{(F_{red}, F_{blue}, 2)}$$

$$\boxed{F_{c_1}, n_1, F_{c_2}, n_2, \ldots, F_{c_i}, n_i}$$

Classes that wait

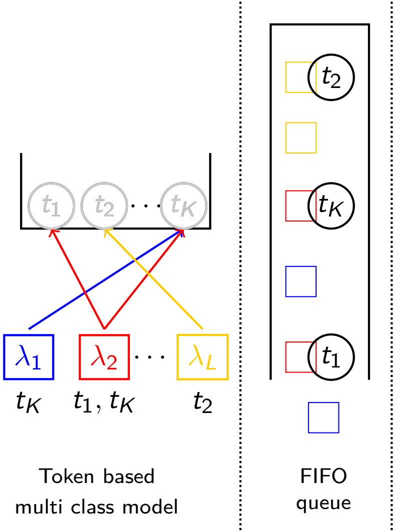


Classes that wait



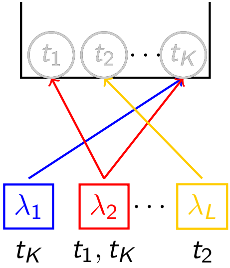
- Consider a generic state
  $x = (T_1, n_1, \dots, T_j, n_j, \dots, T_i, n_i)$

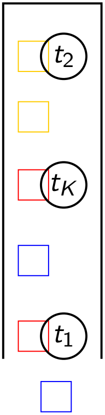
Classes that wait



- Consider a generic state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$
- What are the job classes that constitute the n_j jobs ?

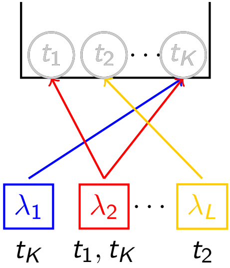
Classes that wait



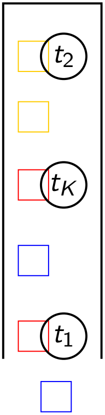


- Consider a generic state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$
- What are the job classes that constitute the n_j jobs ?
- $\mathcal{U}(\{T_1, T_2, \ldots, T_j\})$ denotes the set of job classes who have to wait because tokens $\{T_1, T_2, \ldots, T_j\}$ are busy

Classes that wait



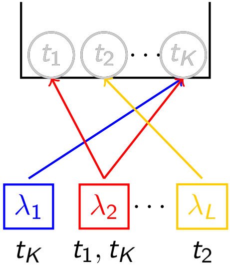
Token based
multi class model



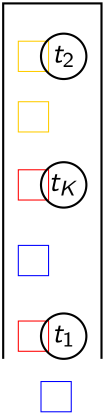
FIFO
queue

- Consider a generic state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$
- What are the job classes that constitute the n_j jobs ?
- $\mathcal{U}(\{T_1, T_2, \ldots, T_j\})$ denotes the set of job classes who have to wait because tokens $\{T_1, T_2, \ldots, T_j\}$ are busy
- $\mathcal{U}(\{t_2\}) = \{L\}, \mathcal{U}(\{t_2, t_K\}) = \{1, L\}$ and $\mathcal{U}(\{t_2, t_K, t_1\}) = \{1, 2, L\}$

Classes that wait



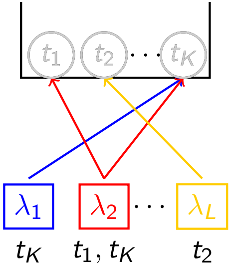
Token based
multi class model



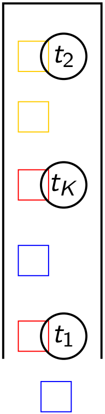
FIFO
queue

- Consider a generic state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$
- What are the job classes that constitute the n_j jobs ?
- $\mathcal{U}(\{T_1, T_2, \ldots, T_j\})$ denotes the set of job classes who have to wait because tokens $\{T_1, T_2, \ldots, T_j\}$ are busy
- $\mathcal{U}(\{t_2\}) = \{L\}, \mathcal{U}(\{t_2, t_K\}) = \{1, L\}$ and $\mathcal{U}(\{t_2, t_K, t_1\}) = \{1, 2, L\}$
- $\lambda_{\mathcal{U}(\{t_2, t_K, t_1\})} = \lambda_1 + \lambda_2 + \lambda_L$

Classes that wait



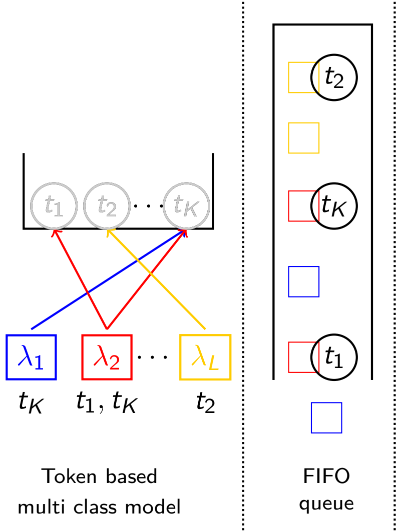
Token based
multi class model



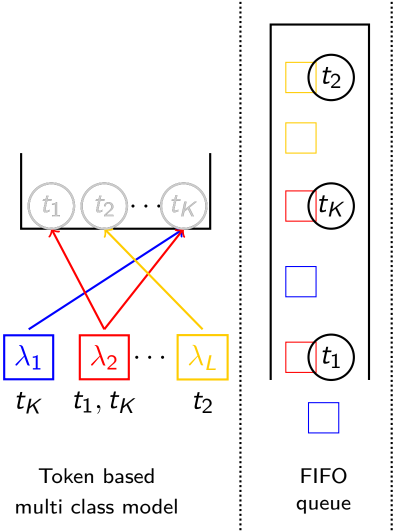
FIFO
queue

- Consider a generic state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$
- What are the job classes that constitute the n_j jobs ?
- $\mathcal{U}(\{T_1, T_2, \ldots, T_j\})$ denotes the set of job classes who have to wait because tokens $\{T_1, T_2, \ldots, T_j\}$ are busy
- $\mathcal{U}(\{t_2\}) = \{L\}, \mathcal{U}(\{t_2, t_K\}) = \{1, L\}$ and $\mathcal{U}(\{t_2, t_K, t_1\}) = \{1, 2, L\}$
- $\lambda_{\mathcal{U}(\{t_2, t_K, t_1\})} = \lambda_1 + \lambda_2 + \lambda_L$
- $\lambda_{\mathcal{U}(\{T_1, T_2, \ldots, T_j\})}$

Activating a token

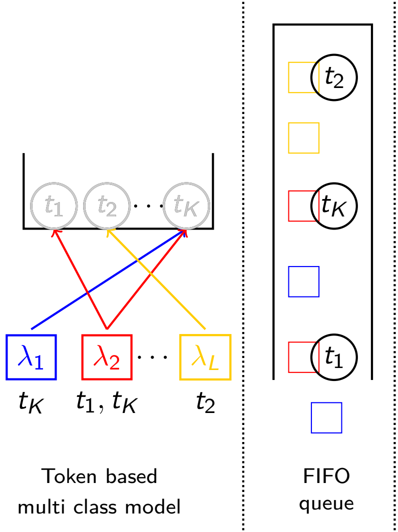


Activating a token



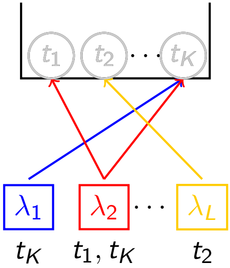
- Consider a generic state
  $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$

Activating a token

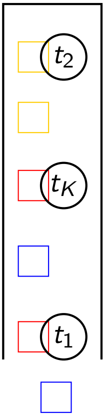


- Consider a generic state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$
- Given that tokens $\{T_1, T_2, \ldots, T_i\}$ are busy, at what rate does a free token T_{i+1} become busy ?

Activating a token



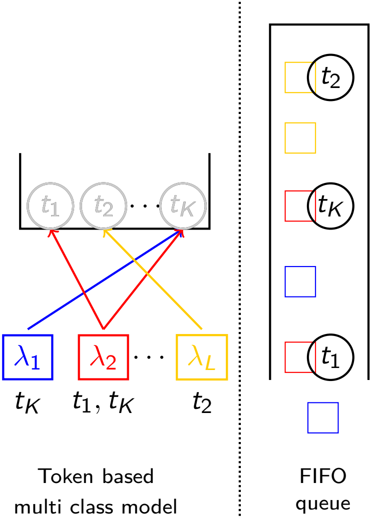
Token based
multi class model



FIFO
queue

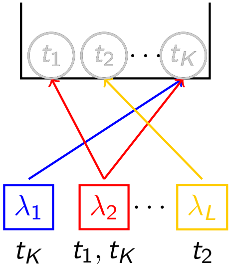
- Consider a generic state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$
- Given that tokens $\{T_1, T_2, \ldots, T_i\}$ are busy, at what rate does a free token T_{i+1} become busy ?
- Denoted by $\lambda_{T_{i+1}}(\{T_1, T_2, \ldots, T_i\})$

Activating a token

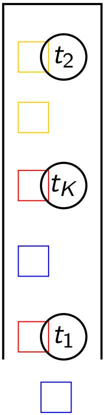


- Consider a generic state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$
- Given that tokens $\{T_1, T_2, \ldots, T_i\}$ are busy, at what rate does a free token T_{i+1} become busy ?
- Denoted by $\lambda_{T_{i+1}}(\{T_1, T_2, \ldots, T_i\})$
- $\lambda_{t_2}(\{\phi\}) = \lambda_L$

Activating a token



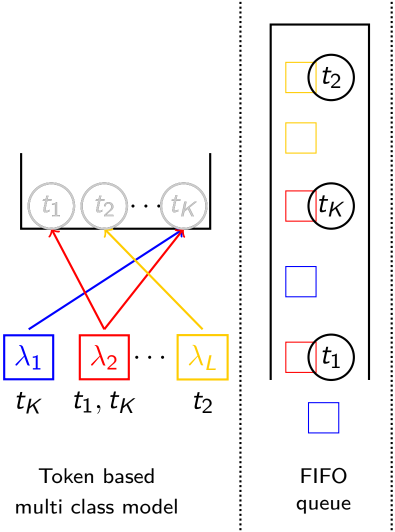
Token based
multi class model



FIFO
queue

- Consider a generic state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$
- Given that tokens $\{T_1, T_2, \ldots, T_i\}$ are busy, at what rate does a free token T_{i+1} become busy ?
- Denoted by $\lambda_{T_{i+1}}(\{T_1, T_2, \ldots, T_i\})$
- $\lambda_{t_2}(\{\phi\}) = \lambda_L$
- $\lambda_{t_K}(\{t_2\}) = \lambda_1 + p\lambda_2$

Activating a token



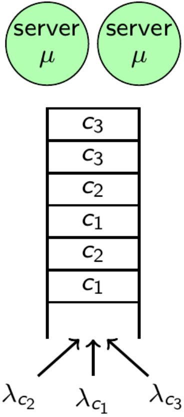
- Consider a generic state $x = (T_1, n_1, \ldots, T_j, n_j, \ldots, T_i, n_i)$
- Given that tokens $\{T_1, T_2, \ldots, T_i\}$ are busy, at what rate does a free token T_{i+1} become busy ?
- Denoted by $\lambda_{T_{i+1}}(\{T_1, T_2, \ldots, T_i\})$
- $\lambda_{t_2}(\{\phi\}) = \lambda_L$
- $\lambda_{t_K}(\{t_2\}) = \lambda_1 + p\lambda_2$
- $\lambda_{t_1}(\{t_2, t_K\}) = \lambda_2$

Applications of OI

- Multi-class M/M/1 queue

Applications of OI

- Multi-class M/M/1 queue
- MSCCC - Multi-server with concurrent customers

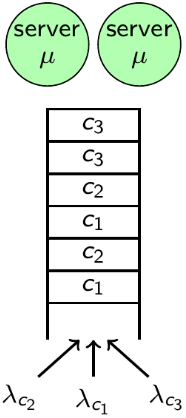


2 identical servers, 3 classes, servers compatible with all classes

Concurrent control: Both servers cannot serve the same class simultaneously

Applications of OI

- Multi-class M/M/1 queue
- MSCCC - Multi-server with concurrent customers



2 identical servers, 3 classes, servers compatible with all classes

Concurrent control: Both servers cannot serve the same class simultaneously

- Processor-Sharing systems: $\mu(c_1, \ldots, c_n)$ might depend on a scalar function $\phi(n)$