Delay, memory, and messaging tradeoffs in distributed service systems

John N. Tsitsiklis

(with D. Gamarnik and M. Zubeldia)

January 2016

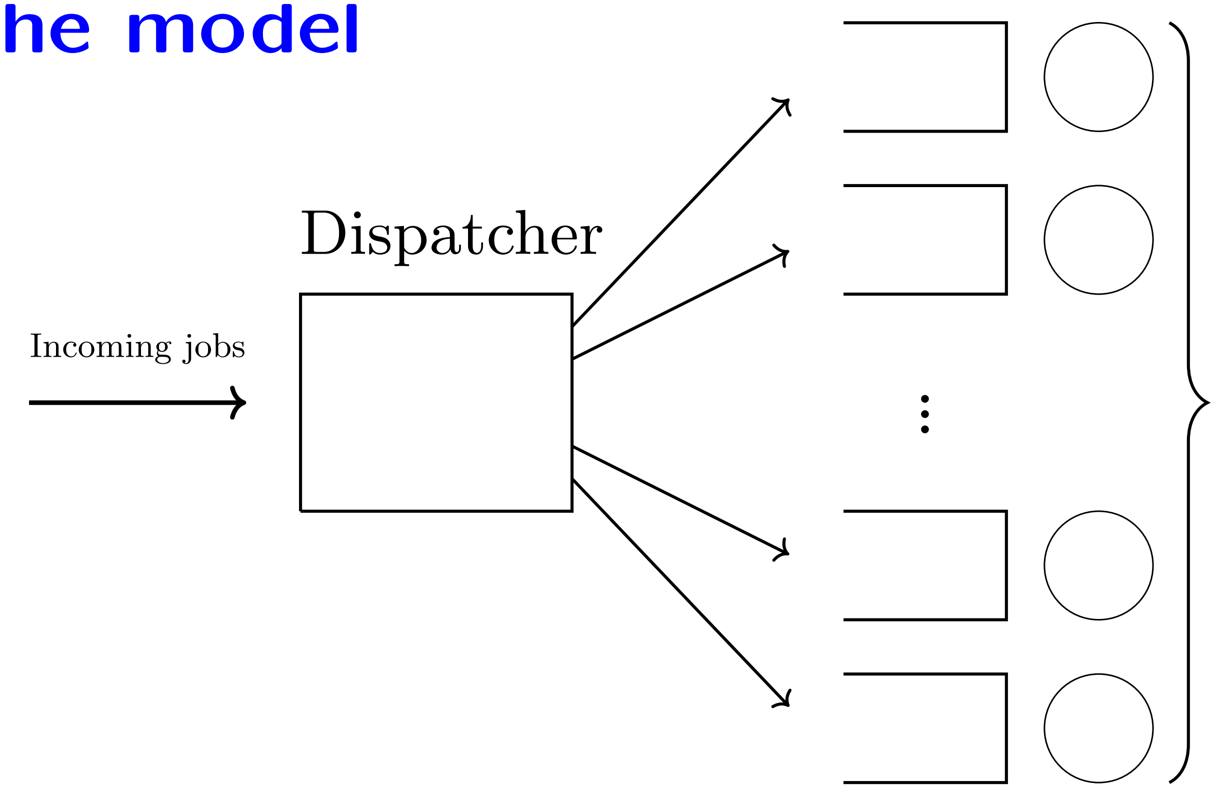
Motivation

- Many modern queueing systems are large scale
- Operating optimally requires large scale resources
- Understand the best performance
  under limited resource availability
- Our context: Dispatching policies
  with limited memory and limited information exchange
  in a many-server queueing system
  (supermarket model)

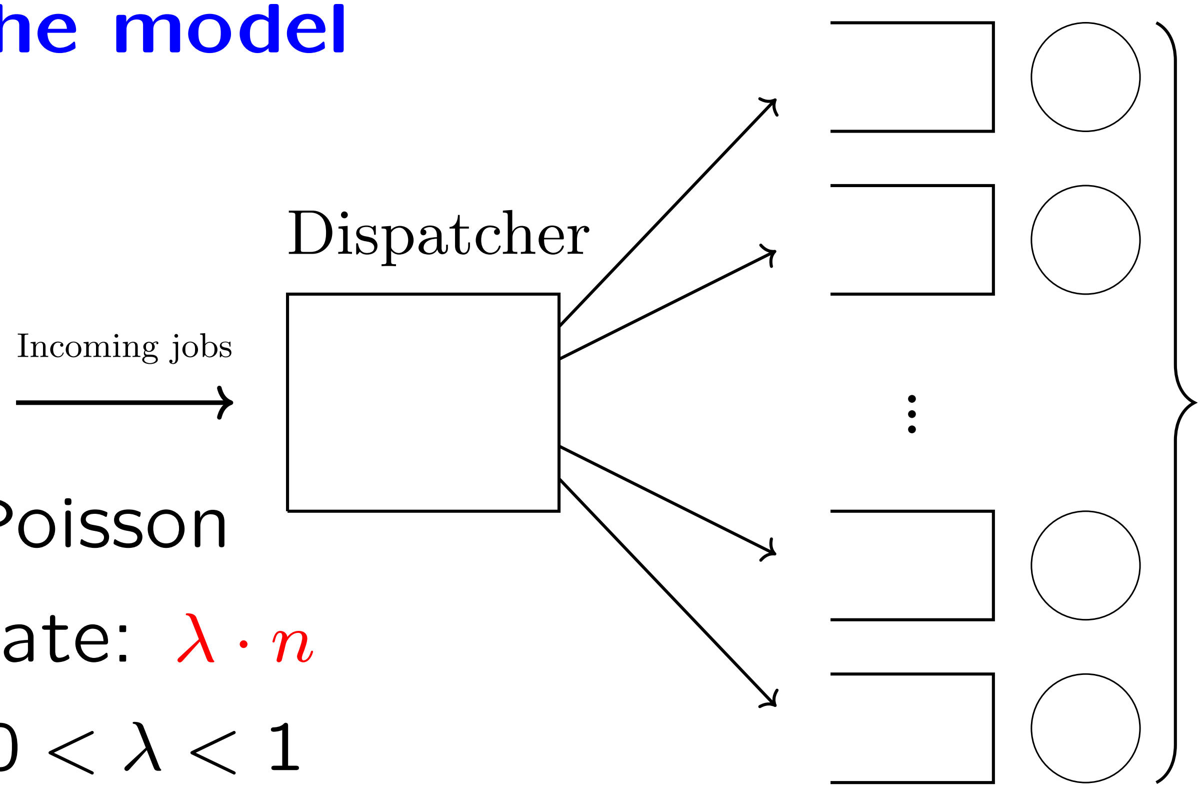
Outline

- The supermarket model
  - overview and comparison of some policies
- A (somewhat) new policy
  - performance in three regimes
- Lower bound on resources required
- Technical details
- Conclusion

The model



The model

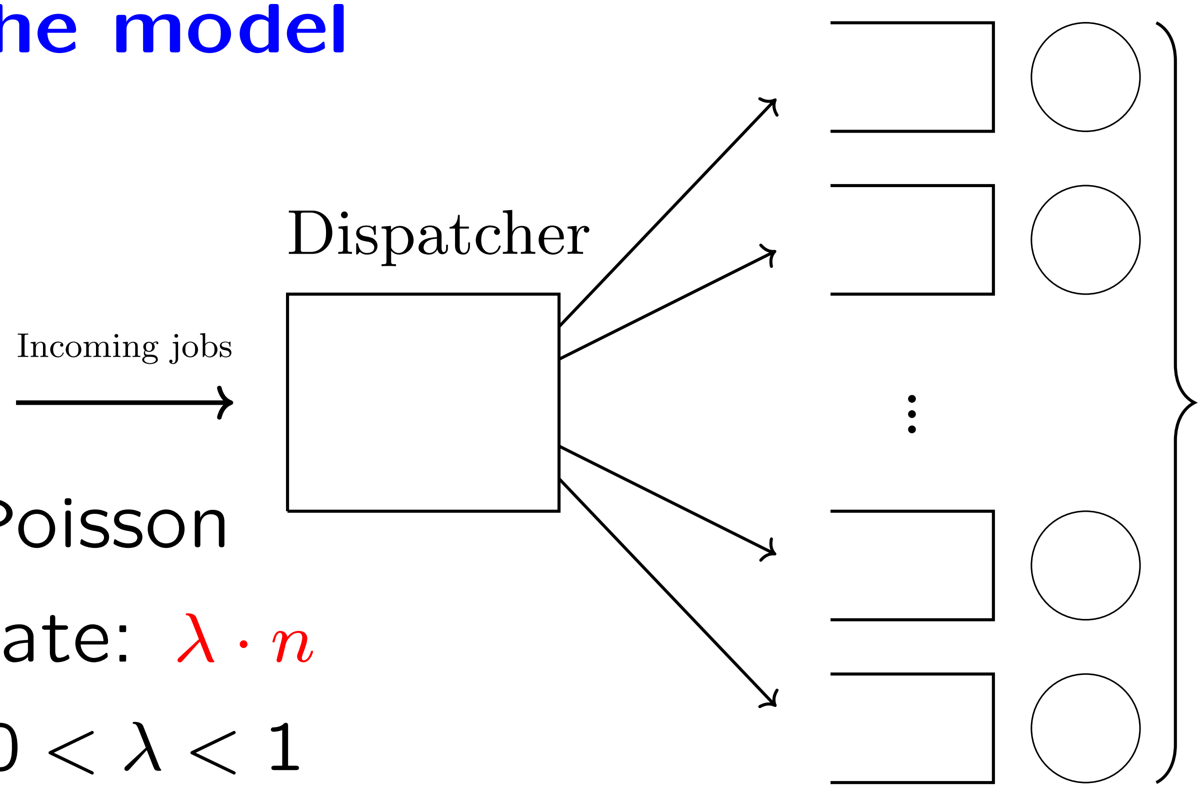


Poisson

rate: $\lambda \cdot n$

$0 < \lambda < 1$

The model



Poisson

rate: $\lambda \cdot n$

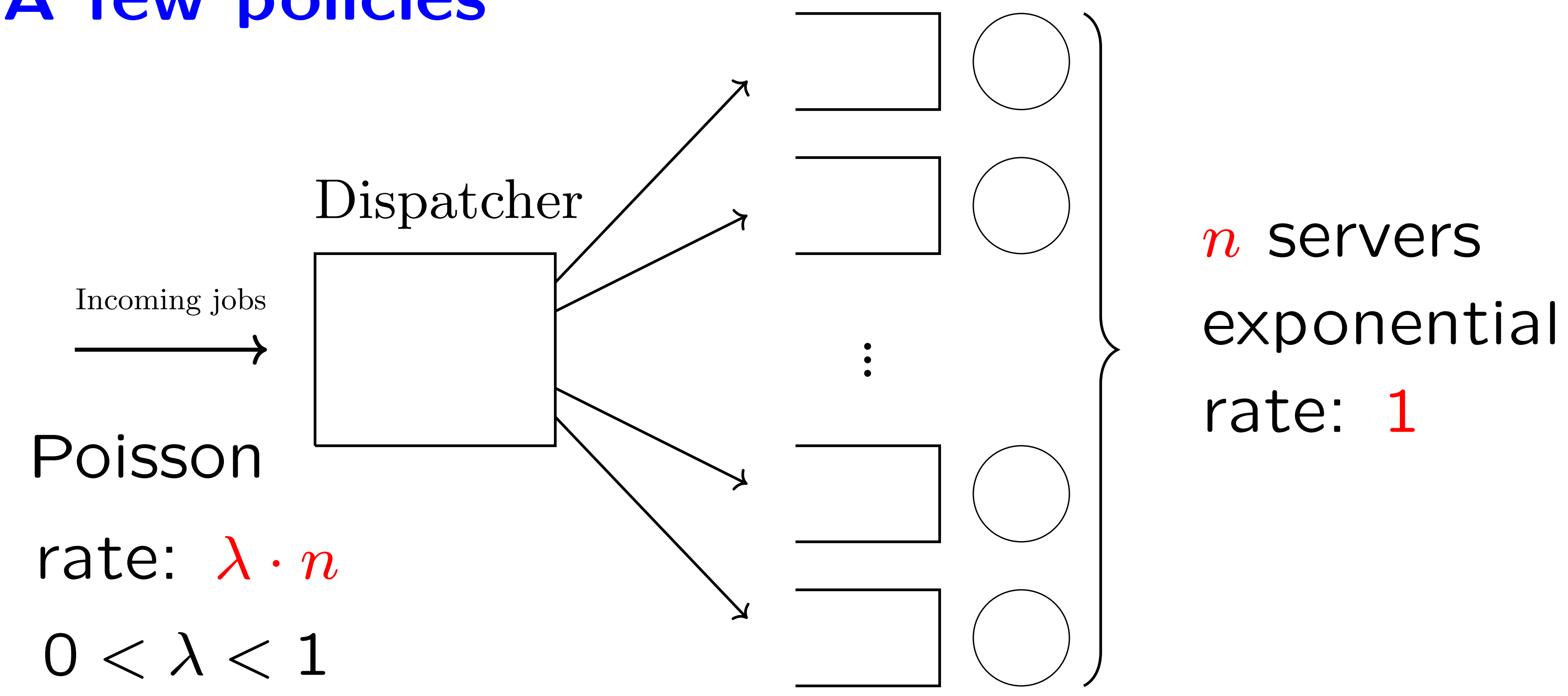
$0 < \lambda < 1$

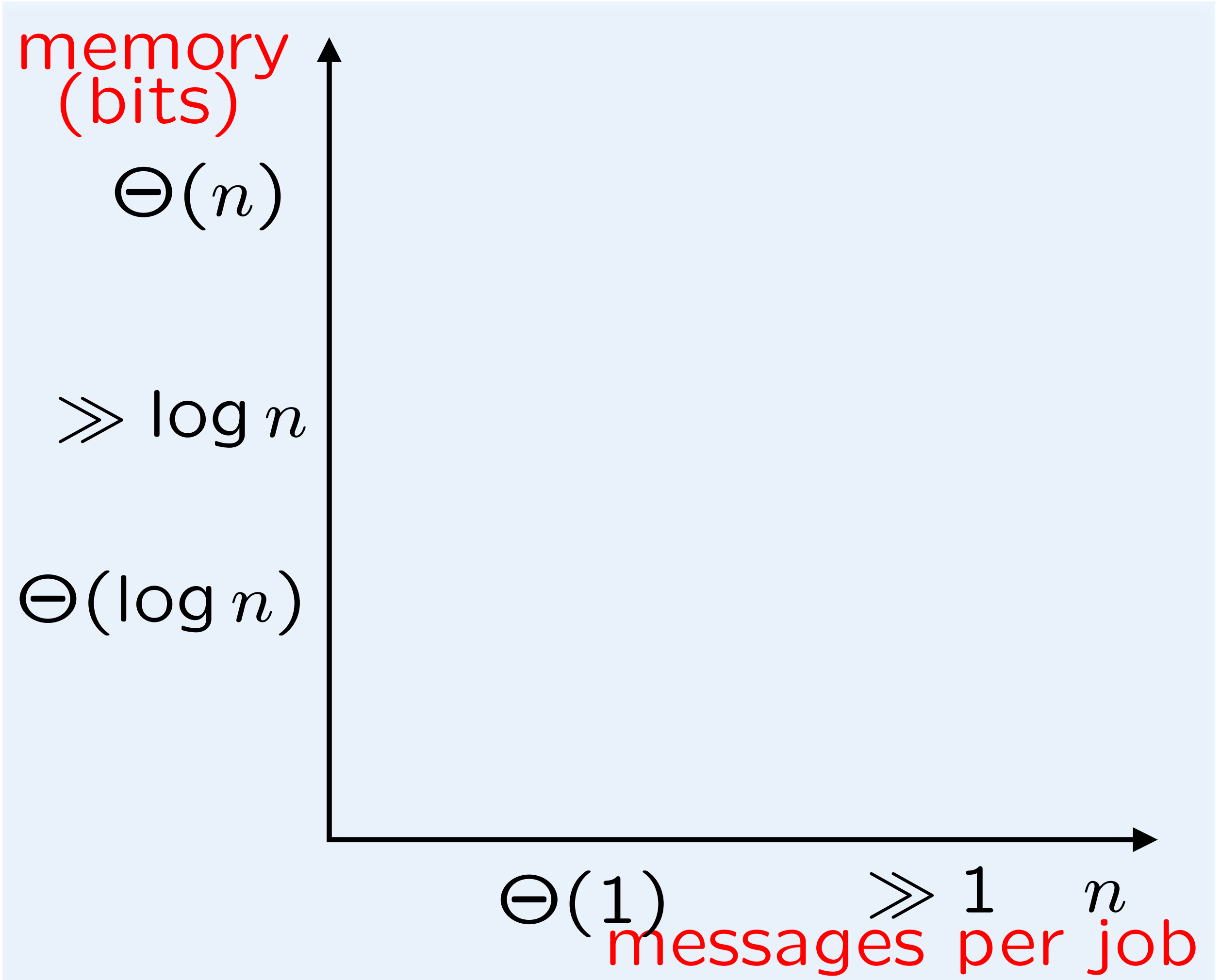
n servers

exponential

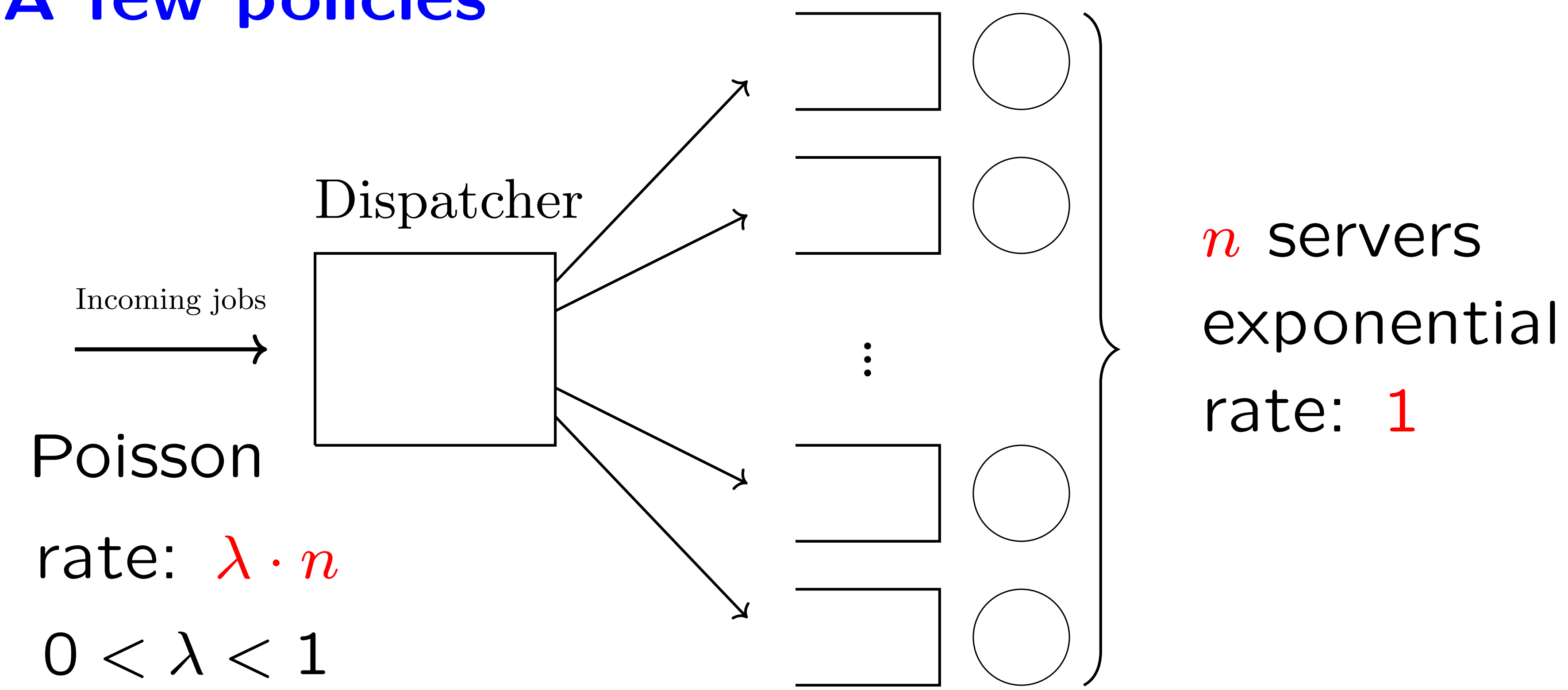
rate: 1

A few policies

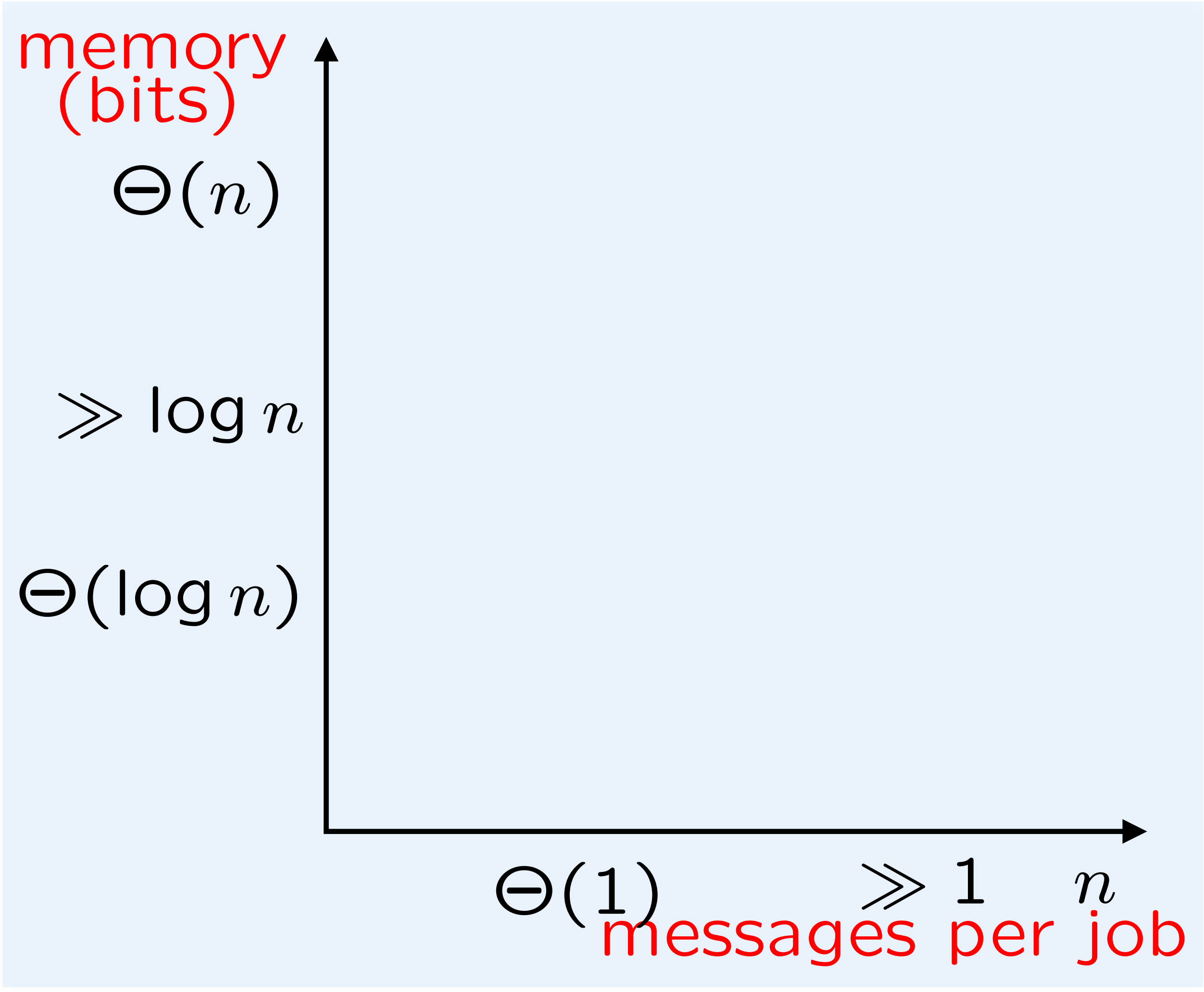




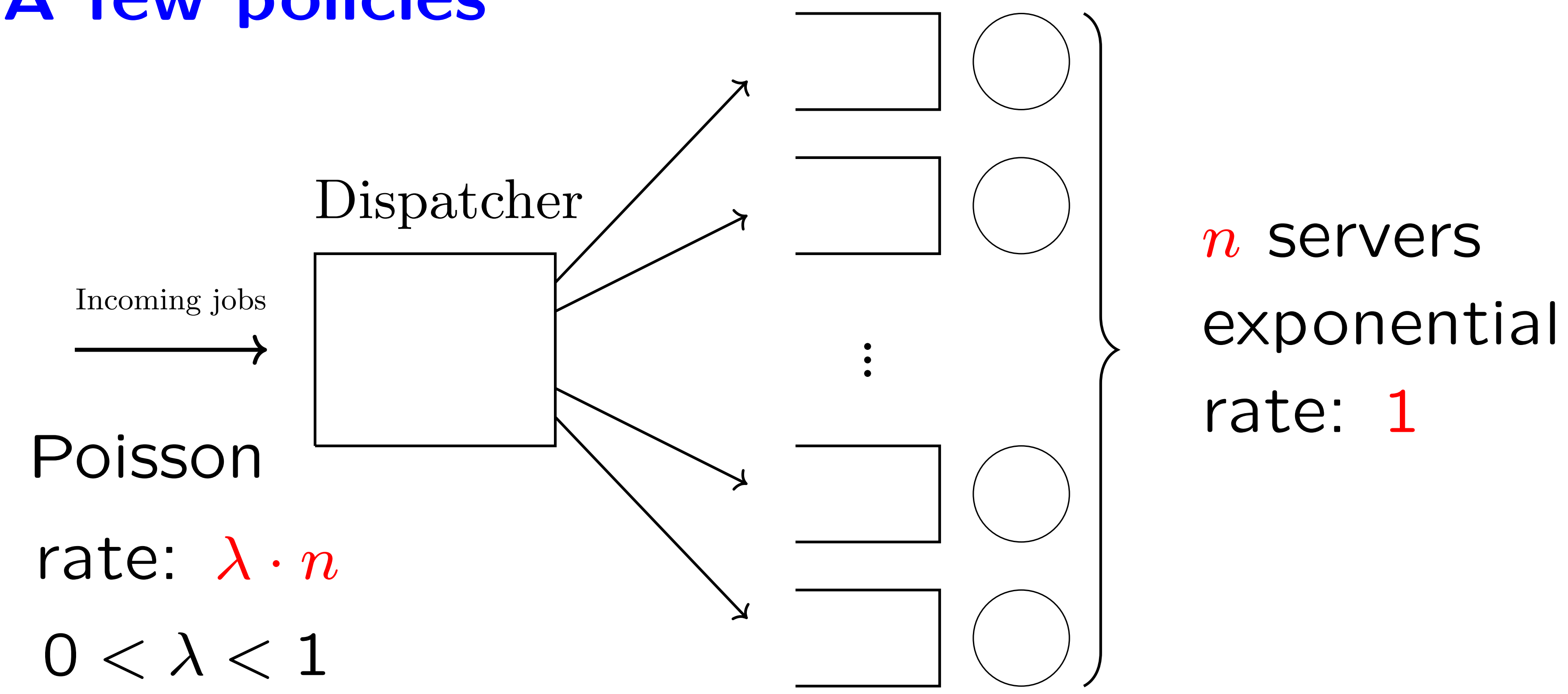
A few policies



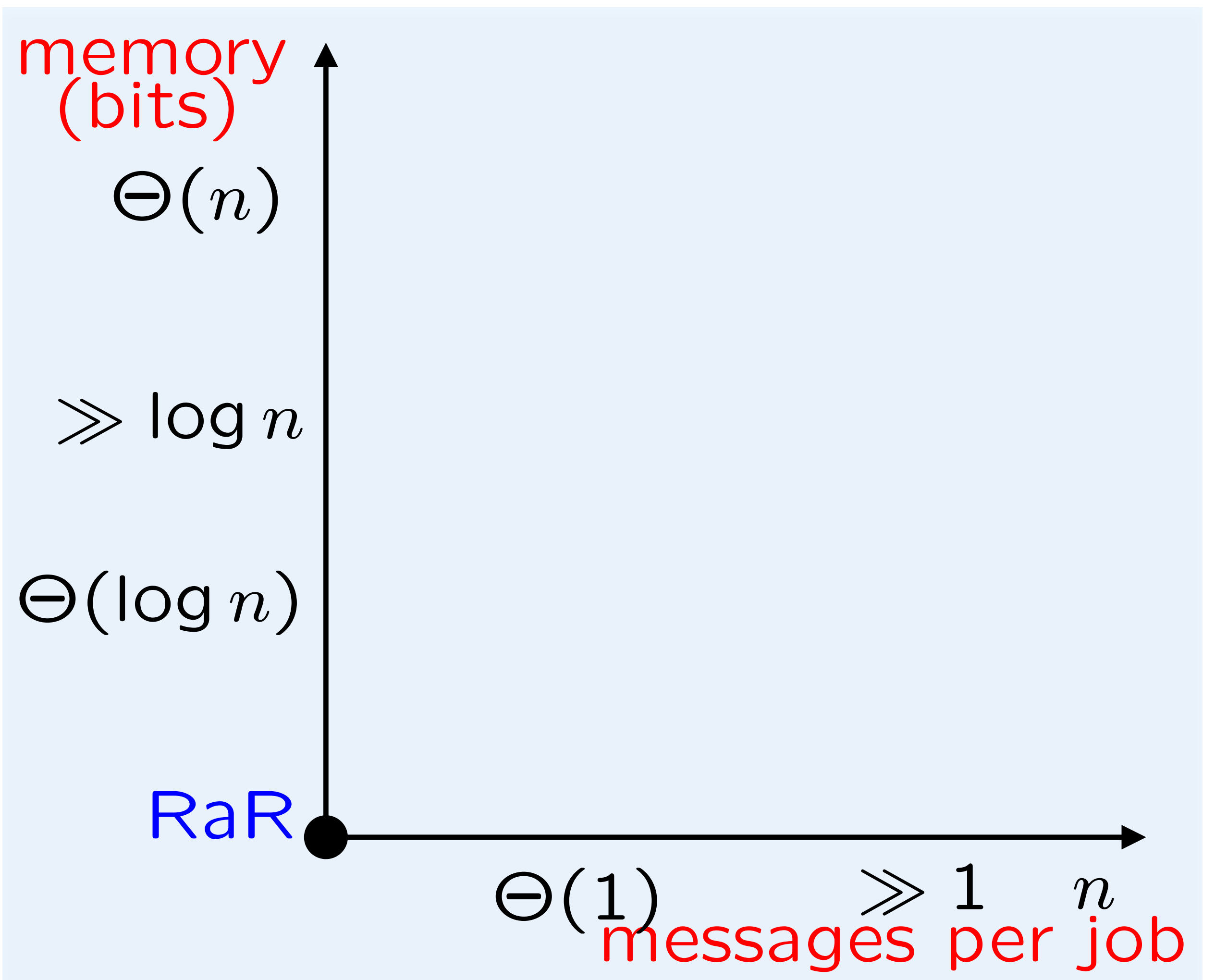
- Random routing (RaR)



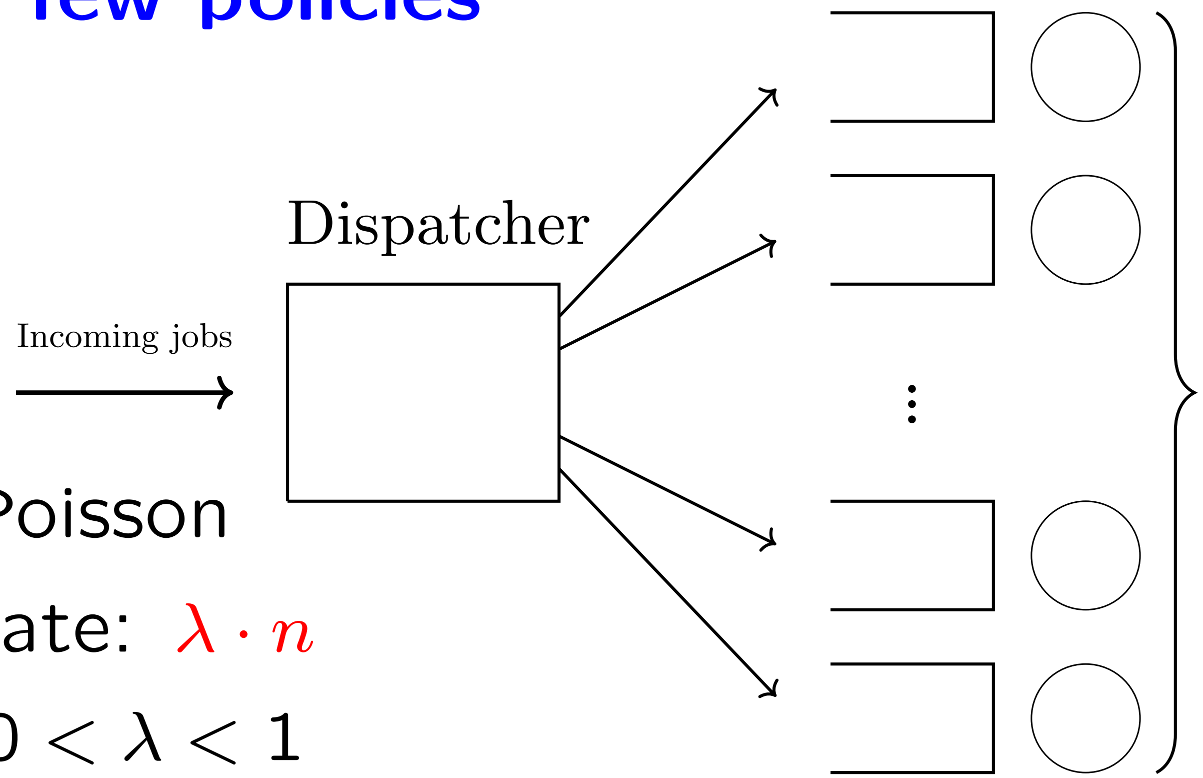
A few policies



- Random routing (RaR)



A few policies



Poisson

rate: $\lambda \cdot n$

$0 < \lambda < 1$

n servers

exponential

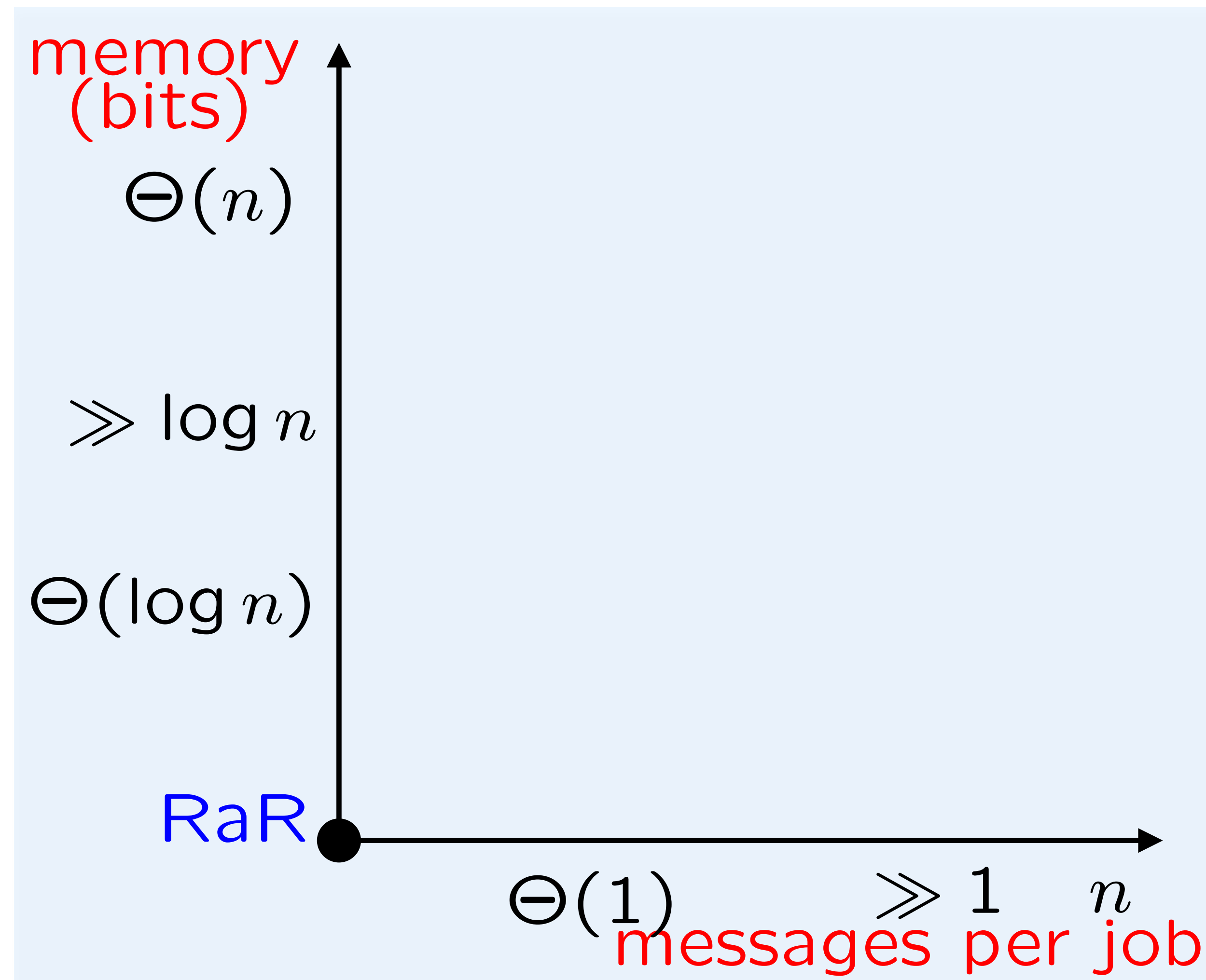
rate: 1

delay

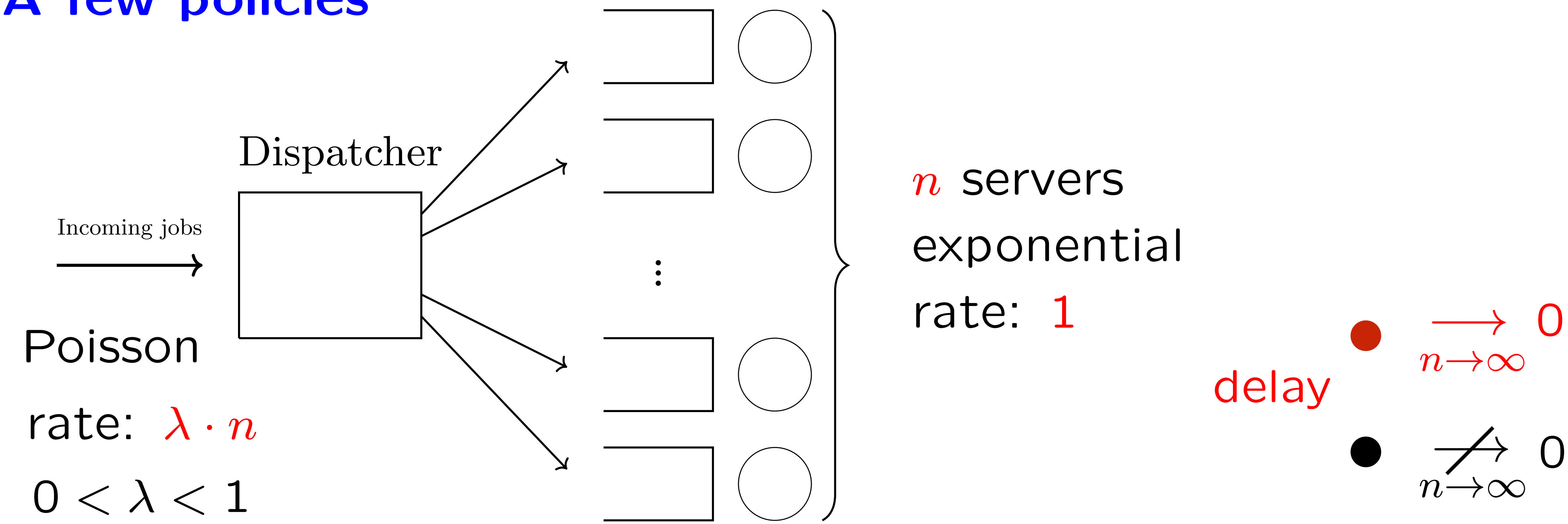
$\xrightarrow[n\to\infty]{} 0$

$\not\xrightarrow[n\to\infty]{} 0$

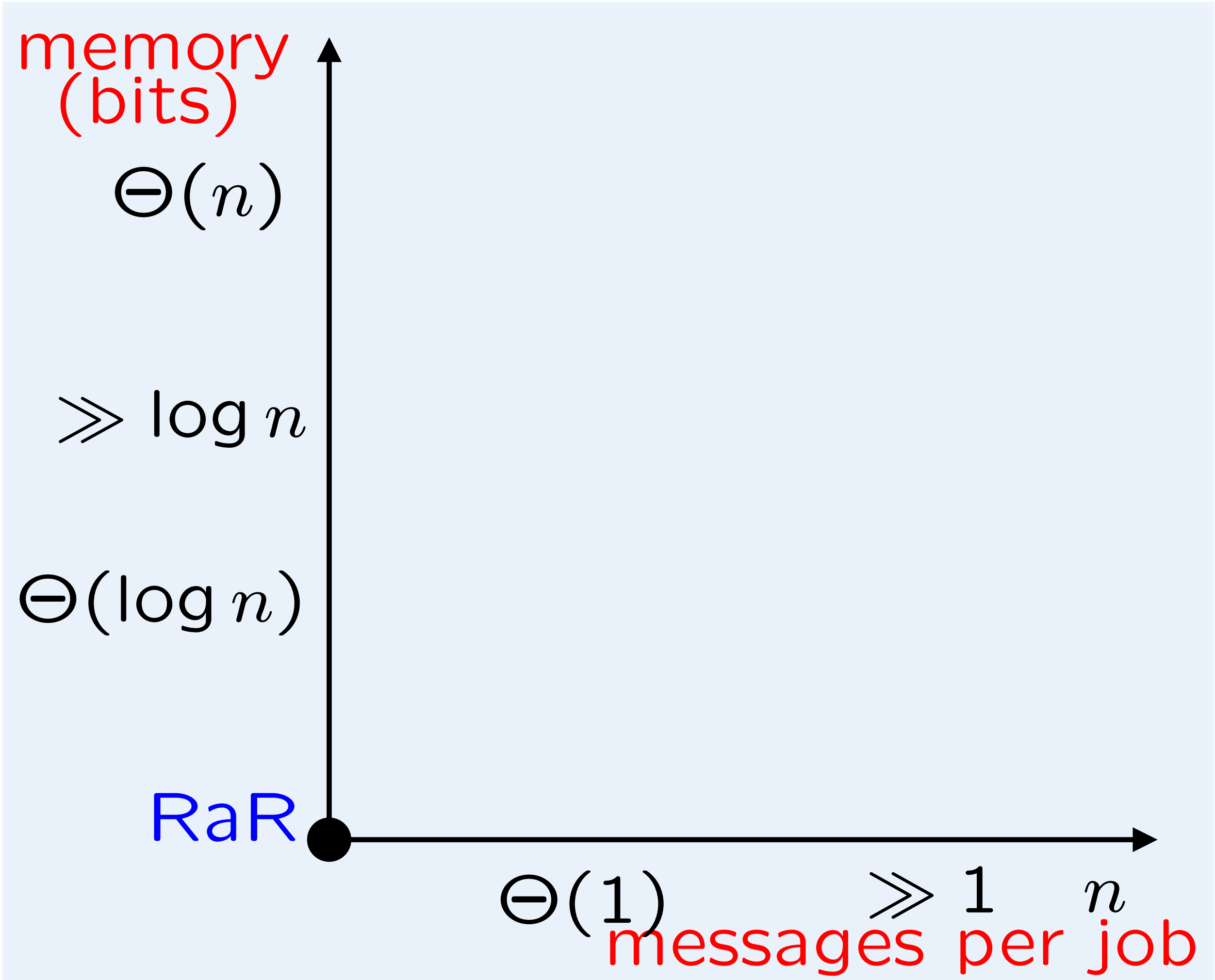
- Random routing (RaR)



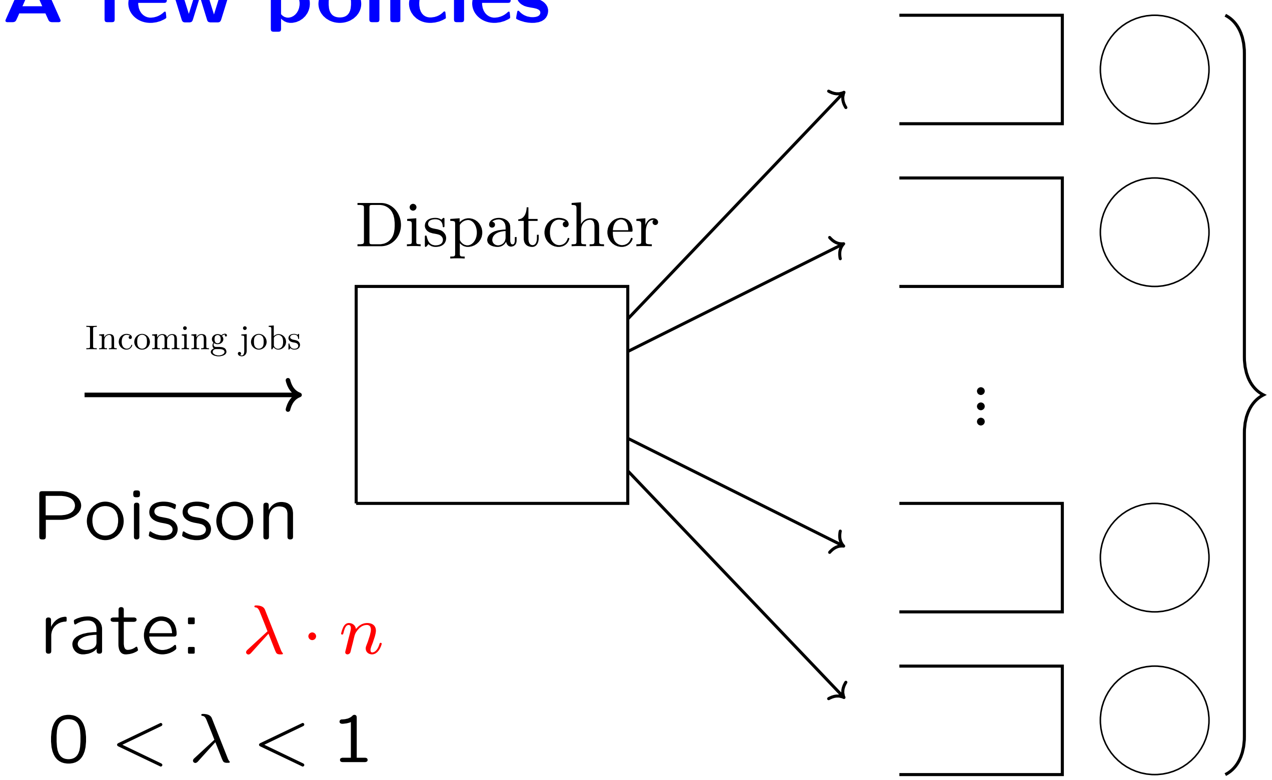
A few policies



- Random routing (RaR)
- Round robin (RR)



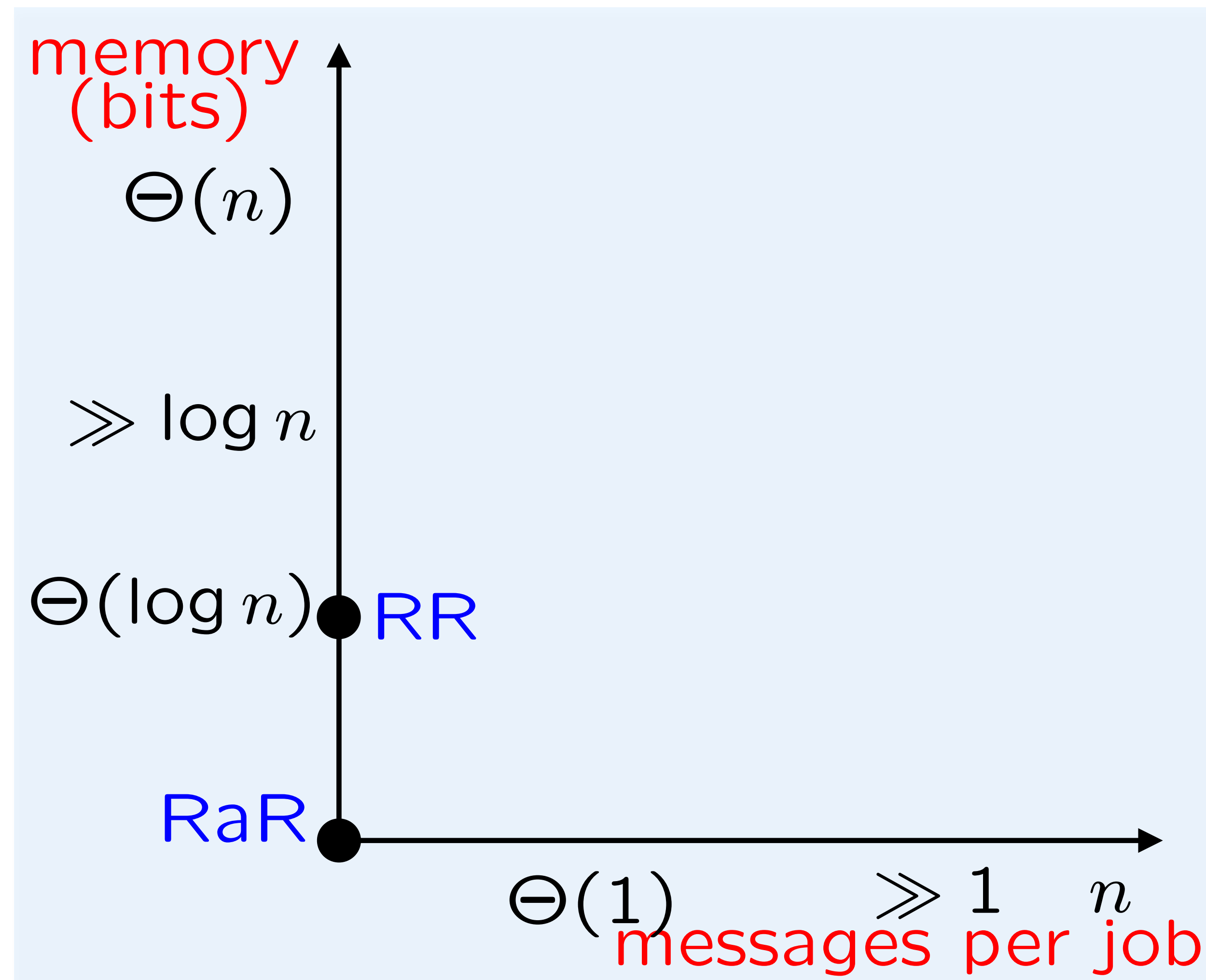
A few policies



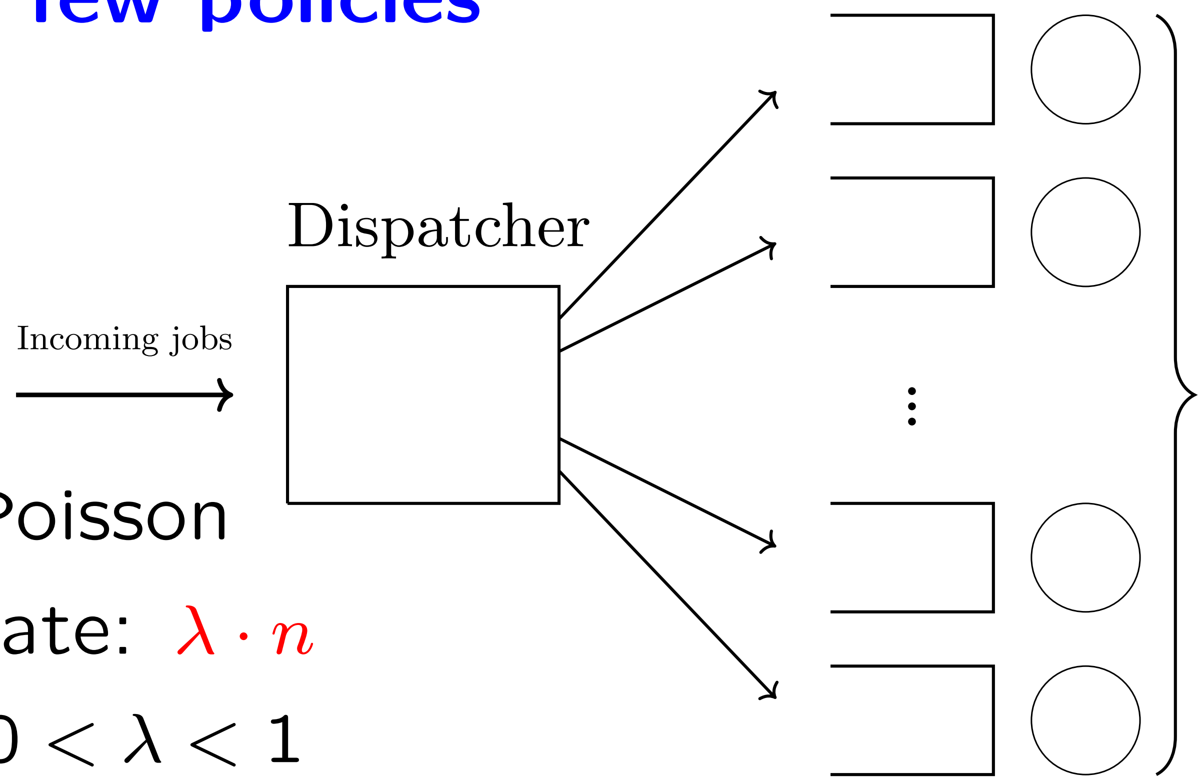
n servers
exponential
rate: 1

delay
● $\xrightarrow[n\to\infty]{} 0$
● $\not\xrightarrow[n\to\infty]{} 0$

- Random routing (RaR)
- Round robin (RR)



A few policies



Poisson

rate: $\lambda \cdot n$

$0 < \lambda < 1$

n servers

exponential

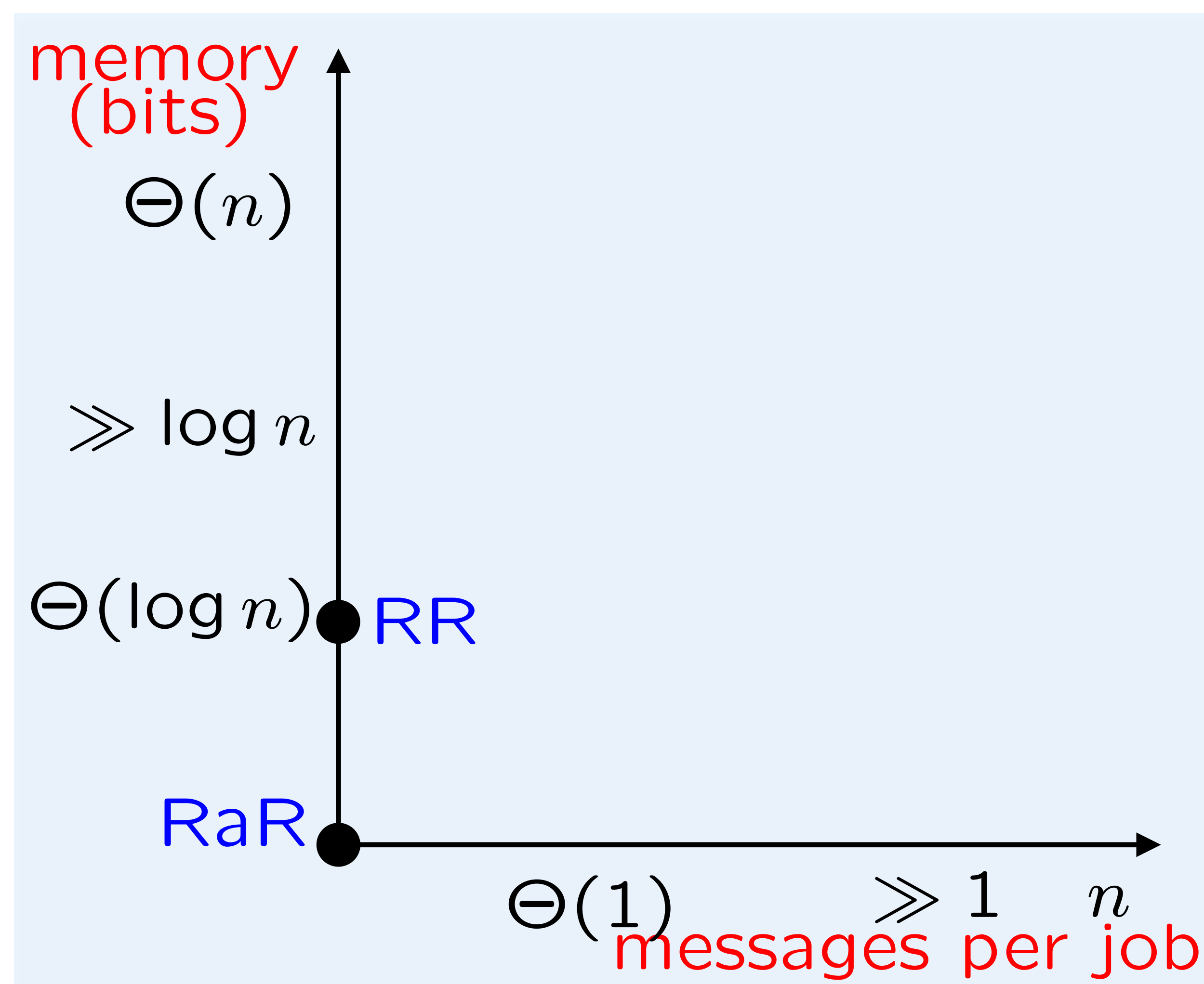
rate: 1

delay

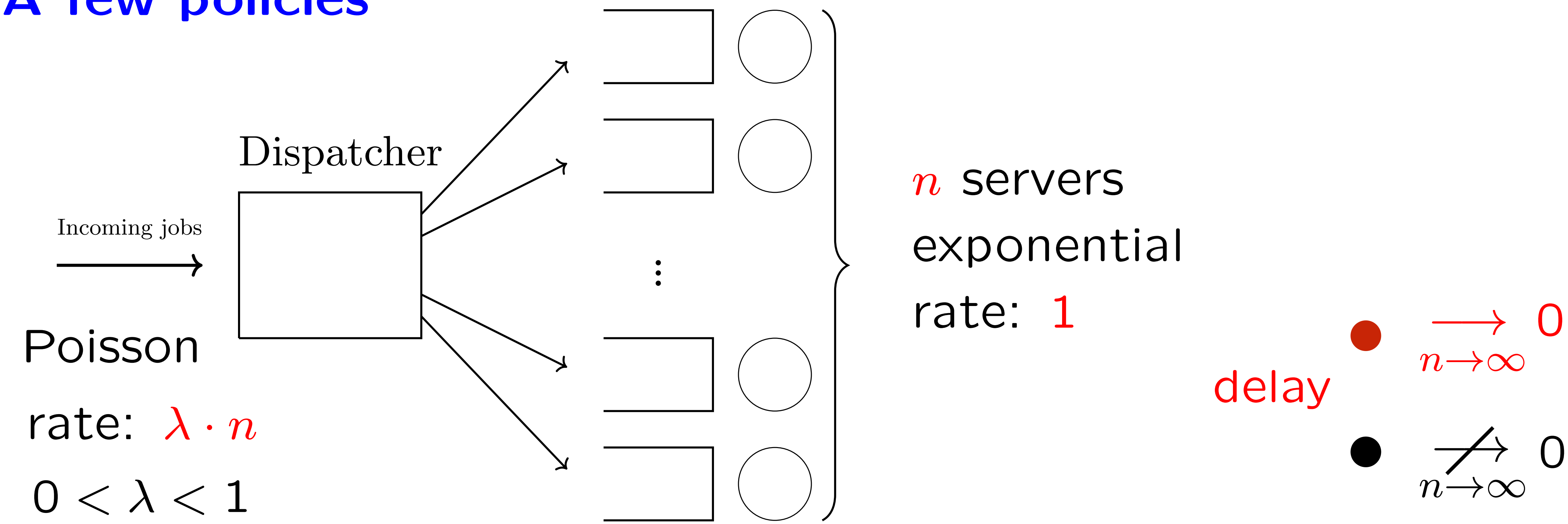
● $\xrightarrow[n\to\infty]{} 0$

● $\not\xrightarrow[n\to\infty]{} 0$

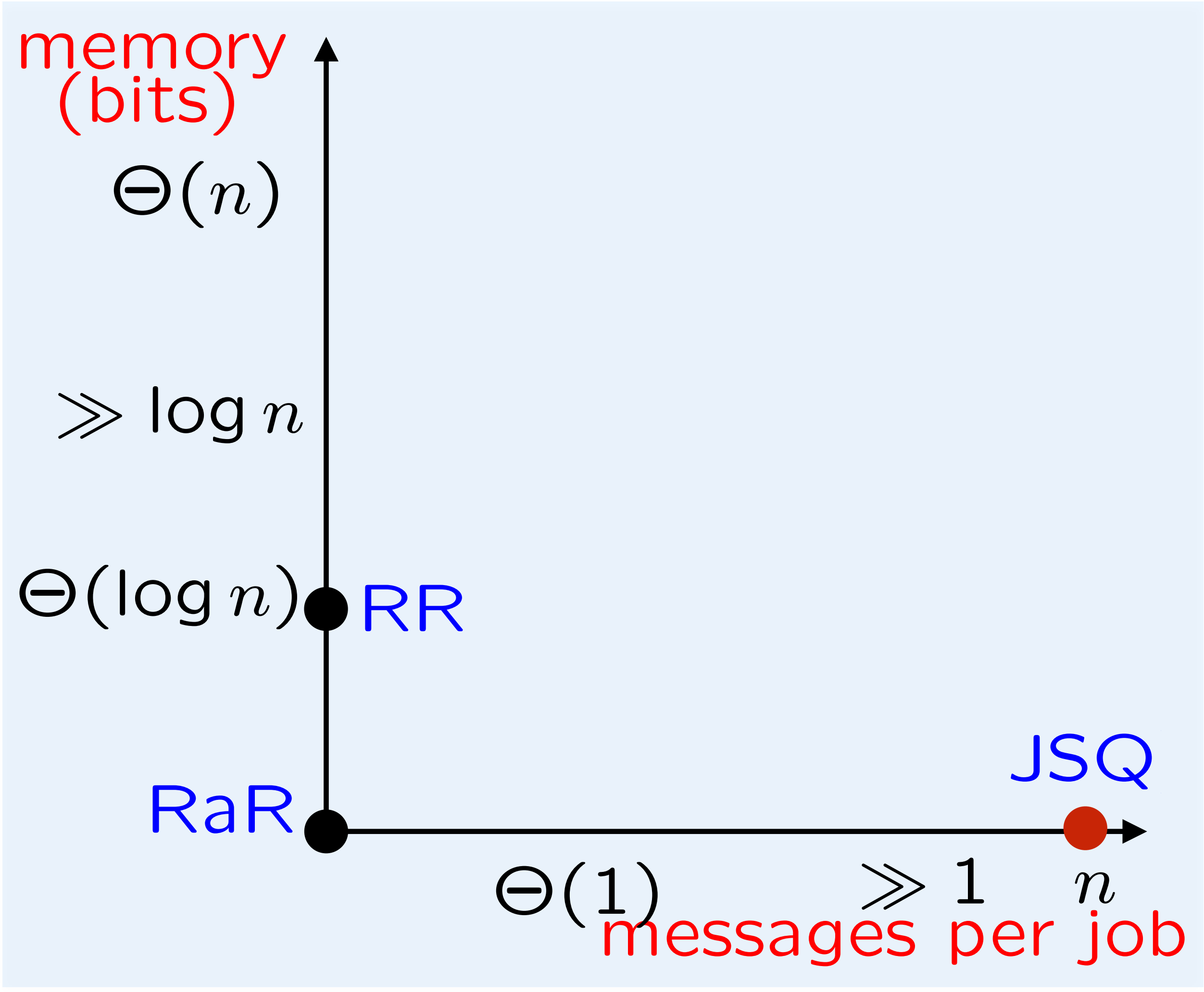
- Random routing (RaR)
- Round robin (RR)
- Join the shortest queue (JSQ)



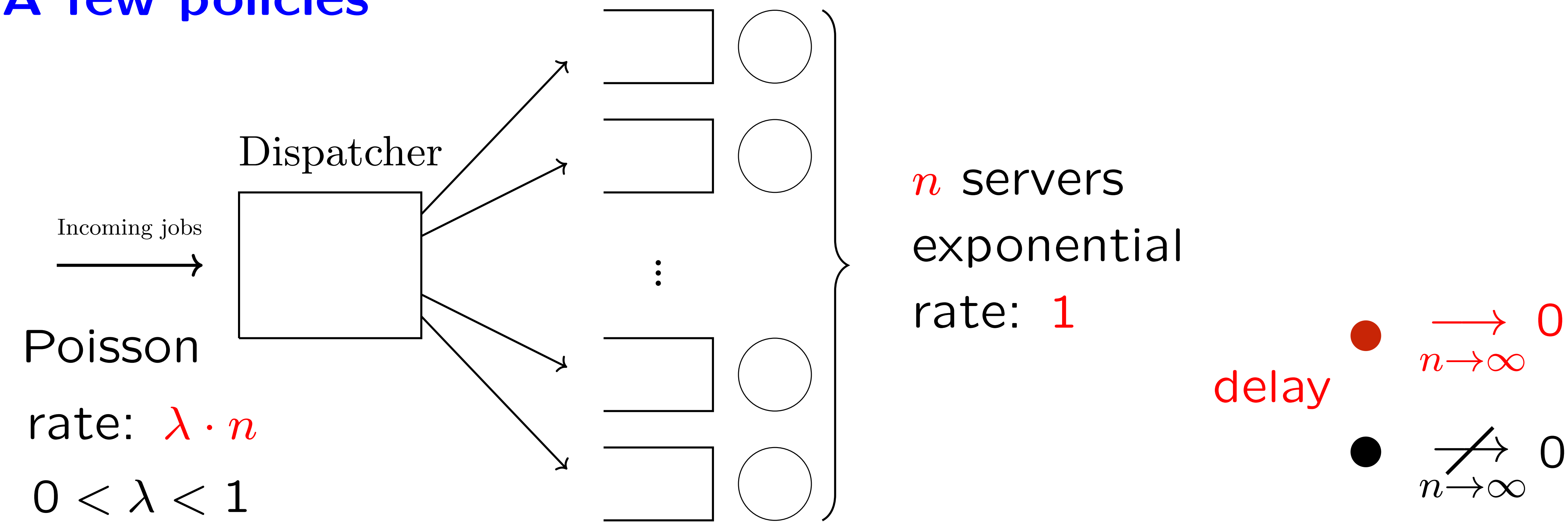
A few policies



- Random routing (RaR)
- Round robin (RR)
- Join the shortest queue (JSQ)

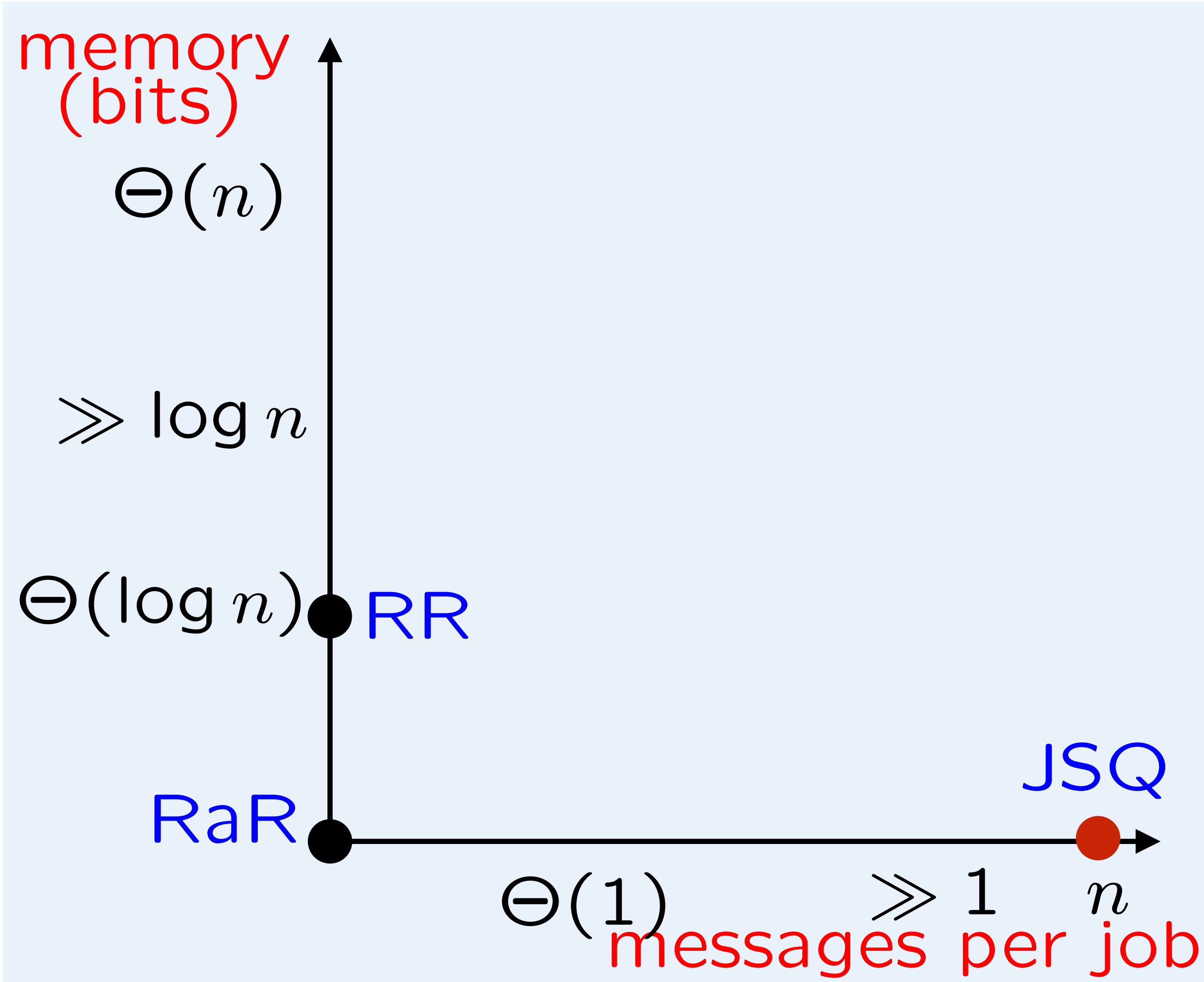


A few policies

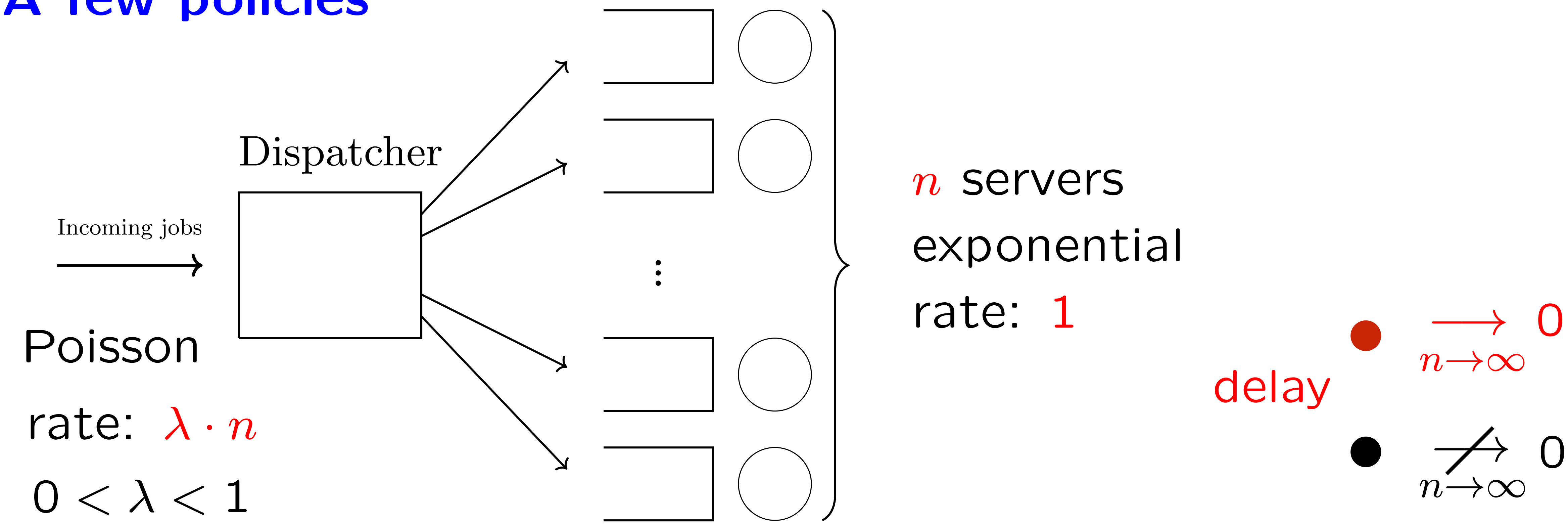


- Random routing (RaR)
- Round robin (RR)
- Join the shortest queue (JSQ)
- Join shortest of d (JSQ(d))

Vvedenskaya et al. (1996), Mitzenmacher (1996)

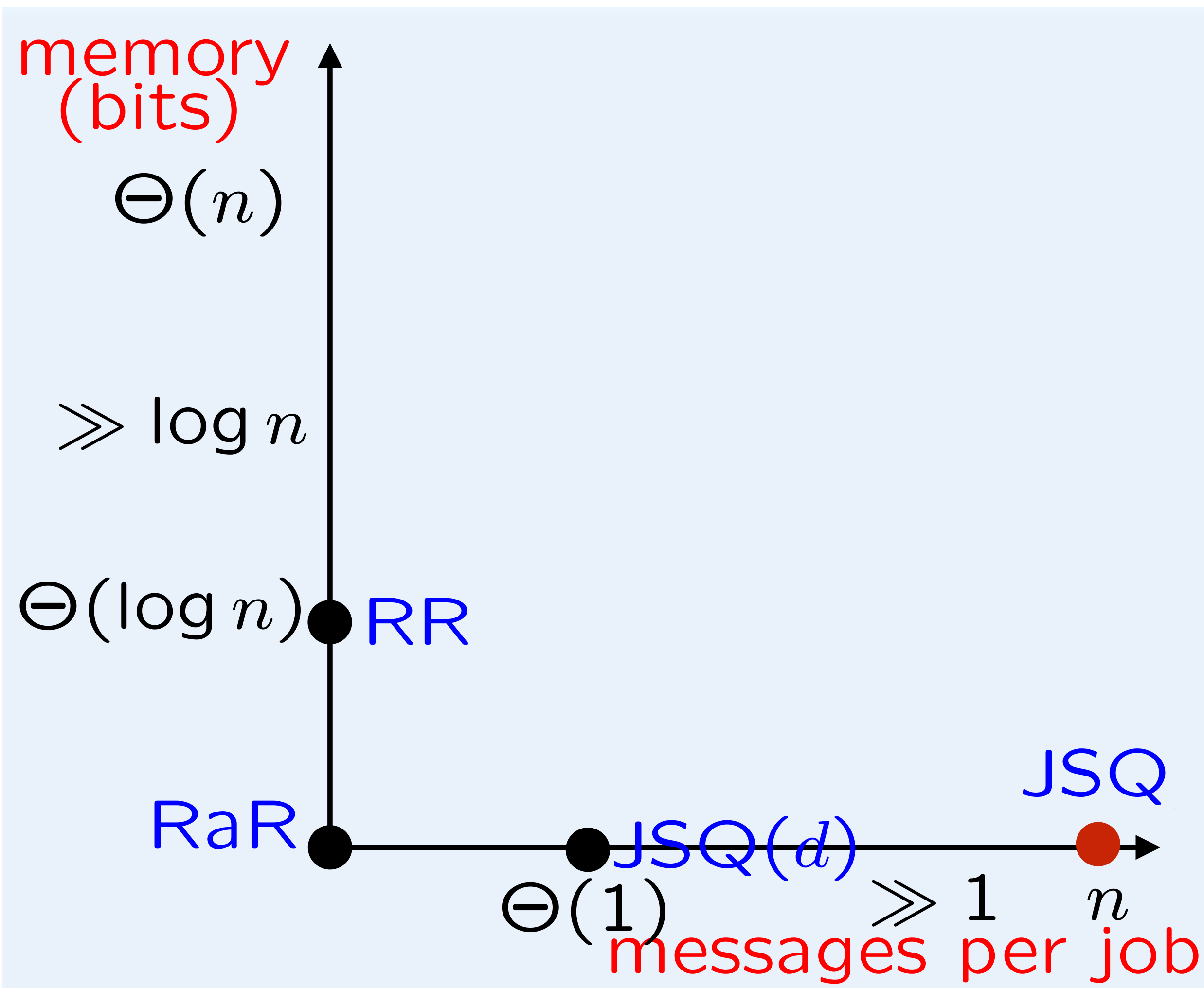


A few policies

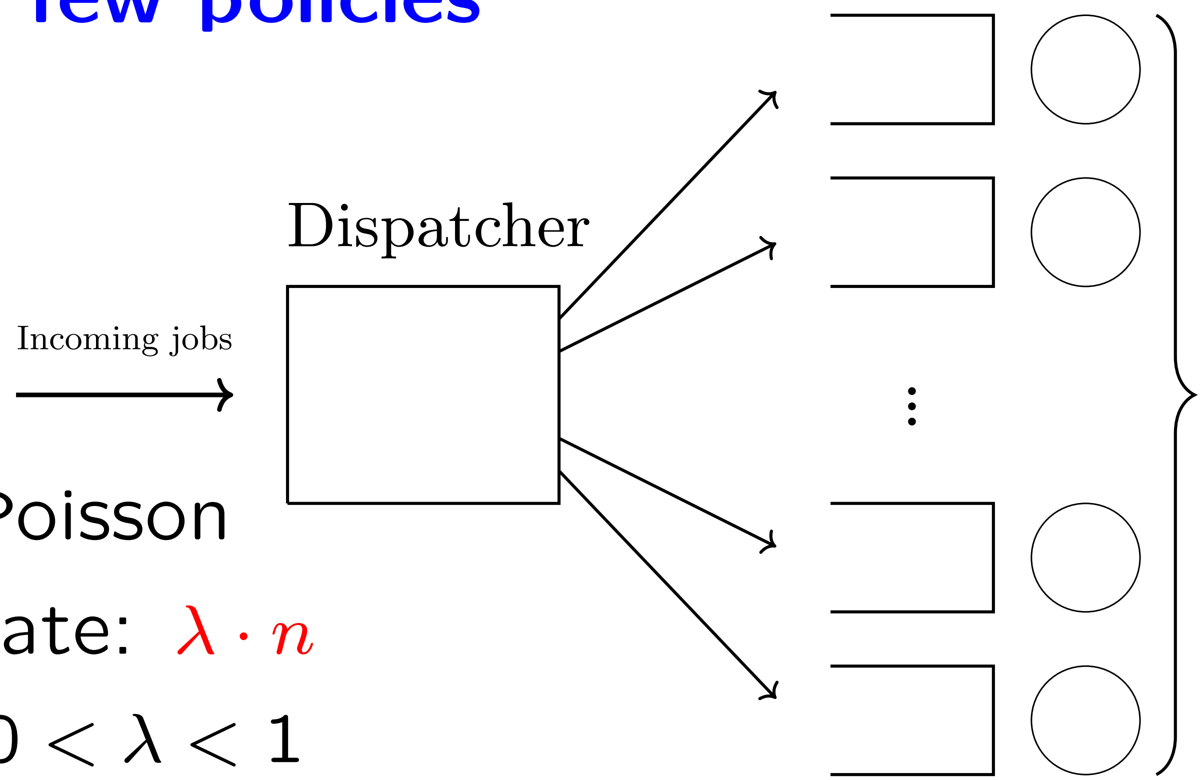


- Random routing (RaR)
- Round robin (RR)
- Join the shortest queue (JSQ)
- Join shortest of d (JSQ(d))

Vvedenskaya et al. (1996), Mitzenmacher (1996)



A few policies



Poisson

rate: $\lambda \cdot n$

$0 < \lambda < 1$

n servers

exponential

rate: 1

delay

● $\xrightarrow[n\to\infty]{} 0$

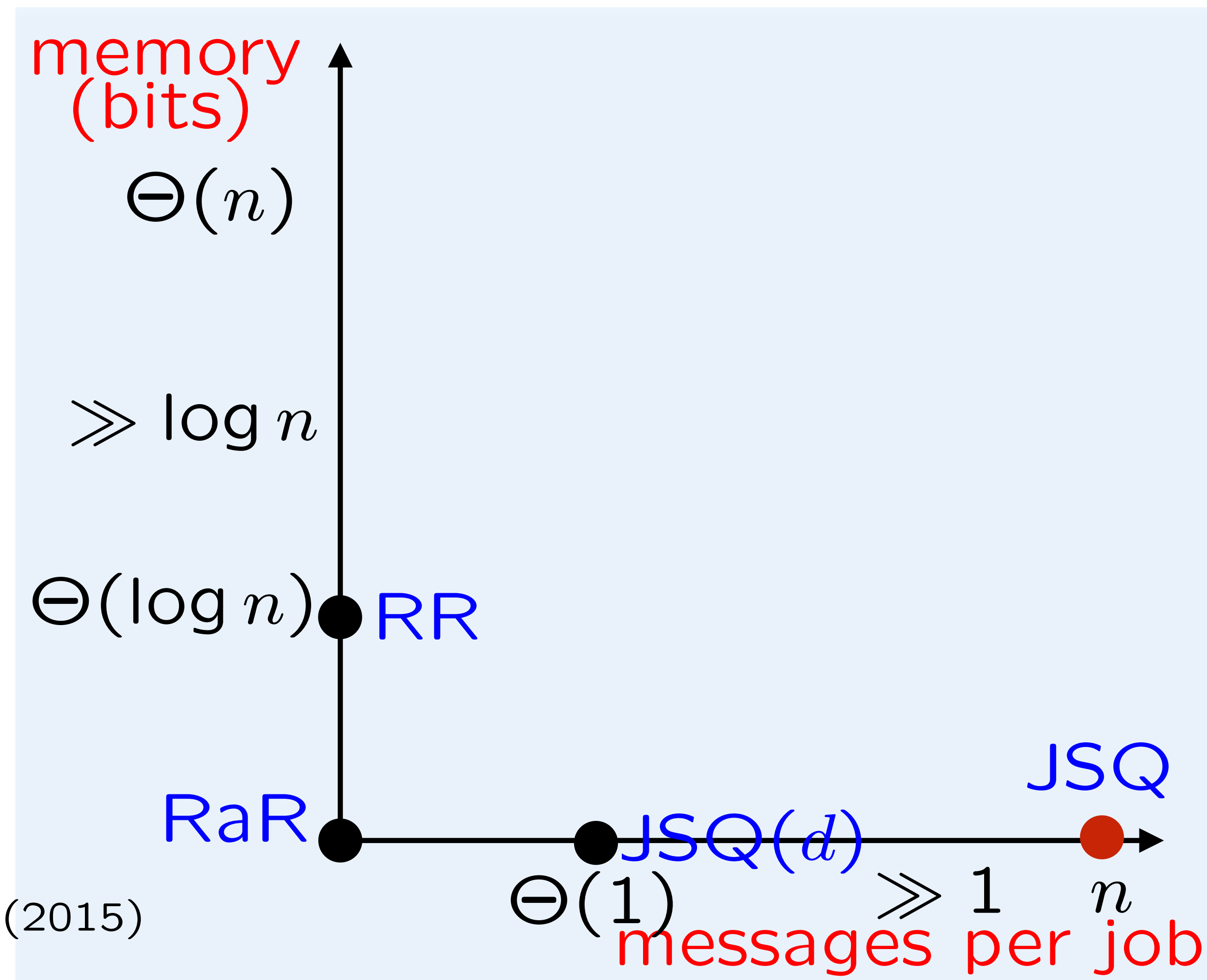
● $\not\xrightarrow[n\to\infty]{} 0$

- Random routing (RaR)
- Round robin (RR)
- Join the shortest queue (JSQ)
- Join shortest of d (JSQ(d))

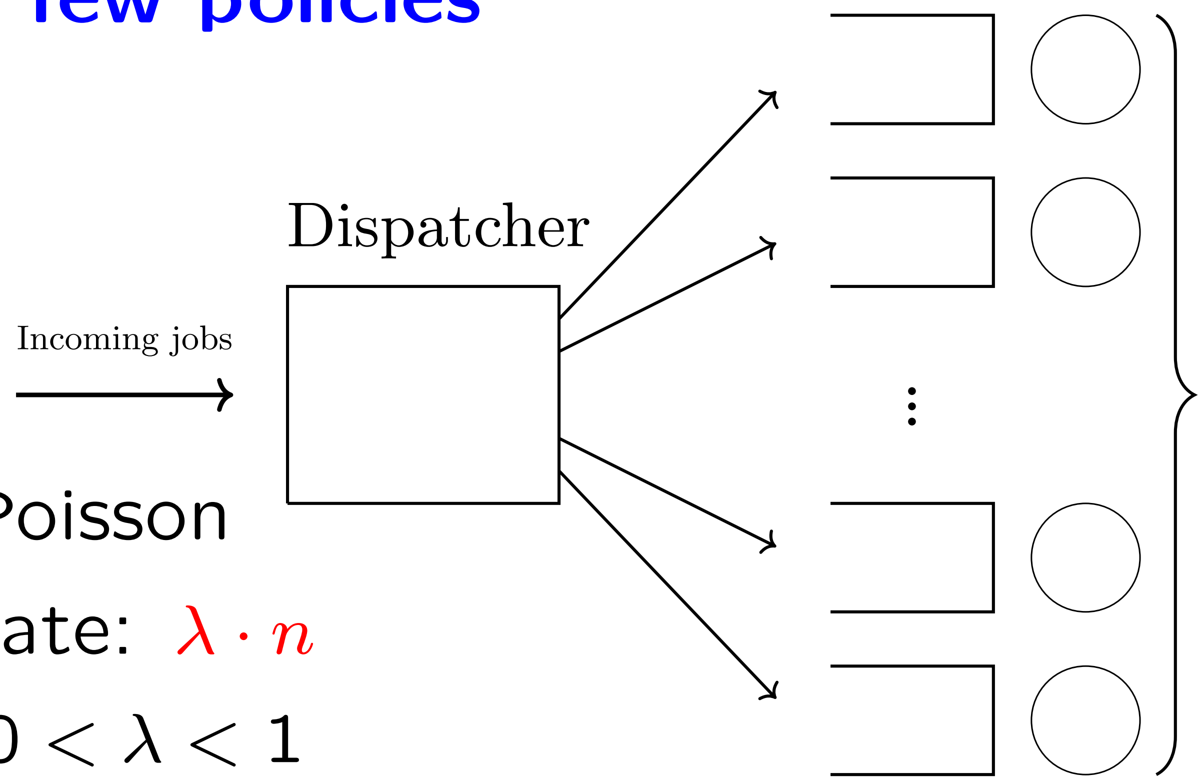
Vvedenskaya et al. (1996), Mitzenmacher (1996)

- Idle servers pull jobs (Pull)

Badonnel & Burgess (2008), Y. Lu et al. (2011), Stolyar (2015)



A few policies



Poisson

rate: $\lambda \cdot n$

$0 < \lambda < 1$

n servers

exponential

rate: 1

delay $\xrightarrow[n\to\infty]{} 0$

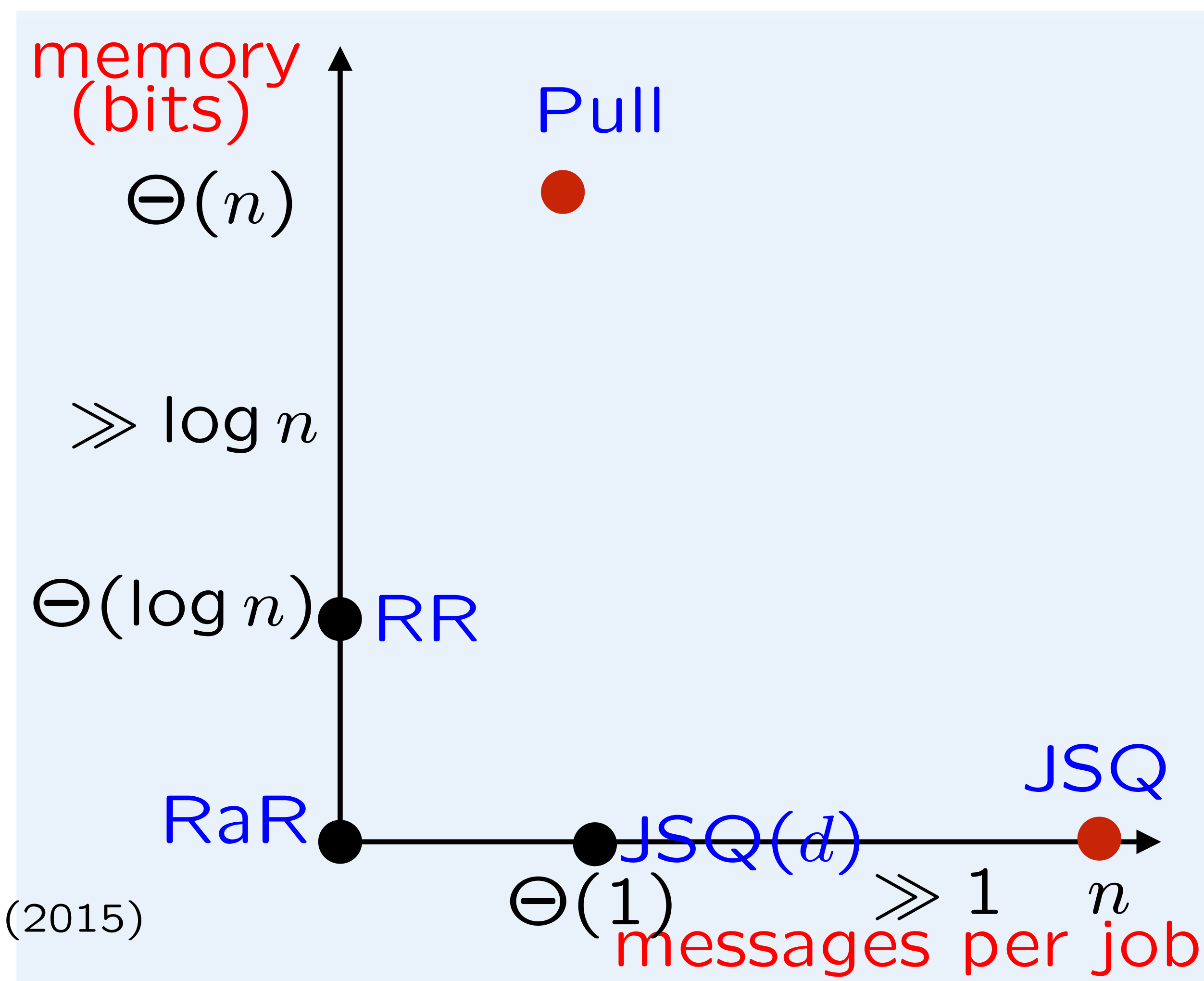
$\not\xrightarrow[n\to\infty]{} 0$

- Random routing (RaR)
- Round robin (RR)
- Join the shortest queue (JSQ)
- Join shortest of d (JSQ(d))

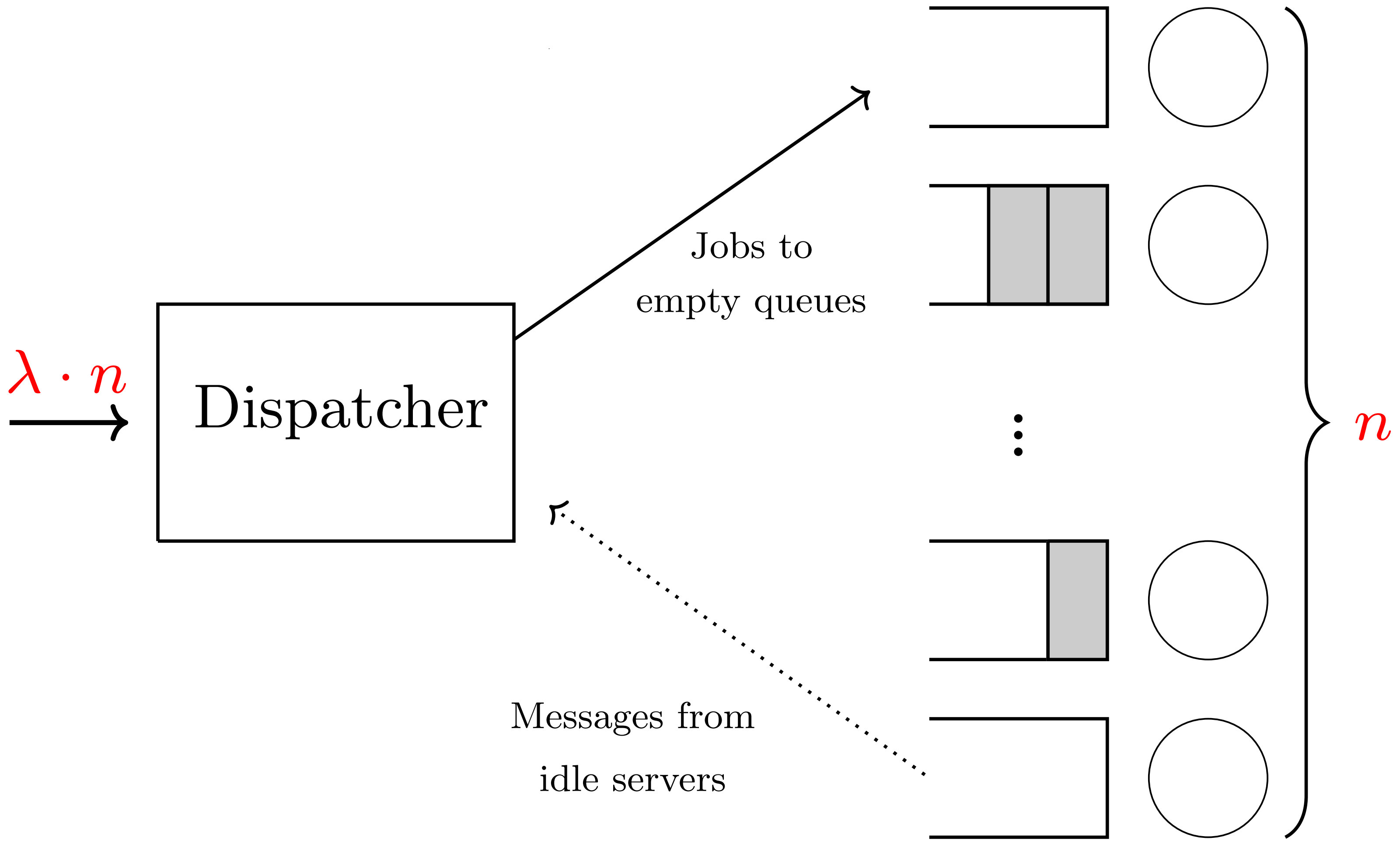
Vvedenskaya et al. (1996), Mitzenmacher (1996)

- Idle servers pull jobs (Pull)

Badonnel & Burgess (2008), Y. Lu et al. (2011), Stolyar (2015)

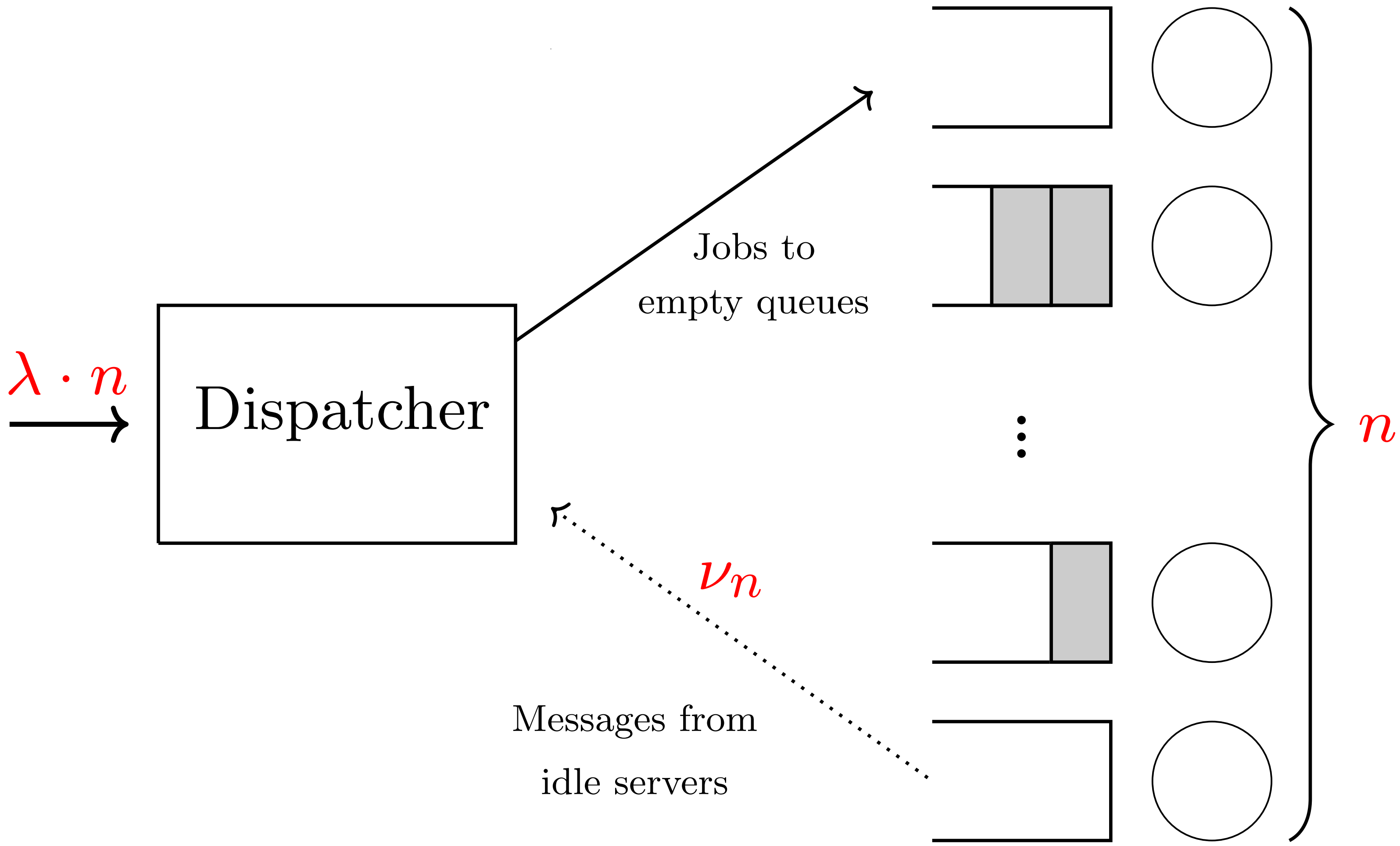


Our policy



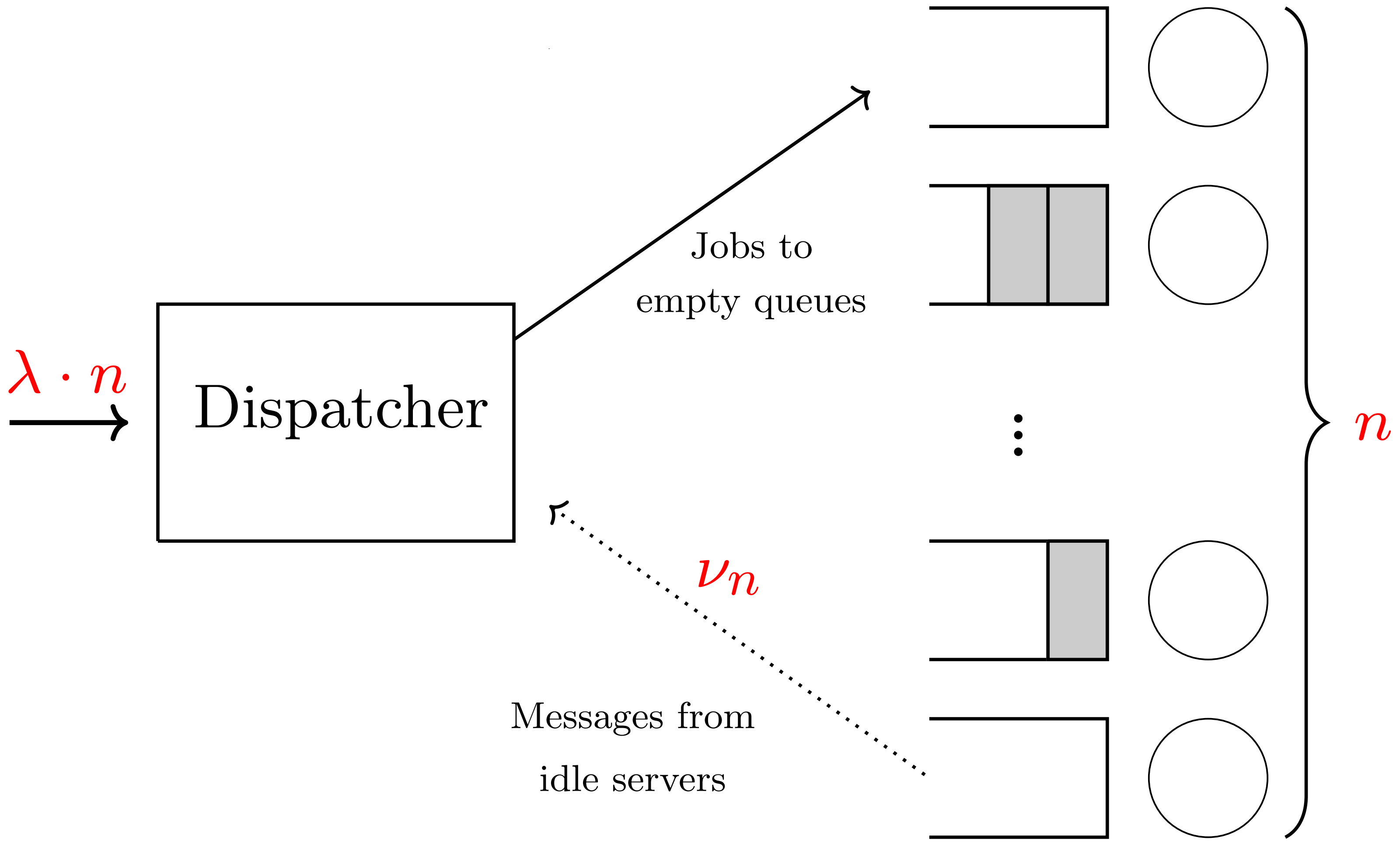
$\lambda \cdot n$
Dispatcher
Jobs to
empty queues
Messages from
idle servers
n

Our policy



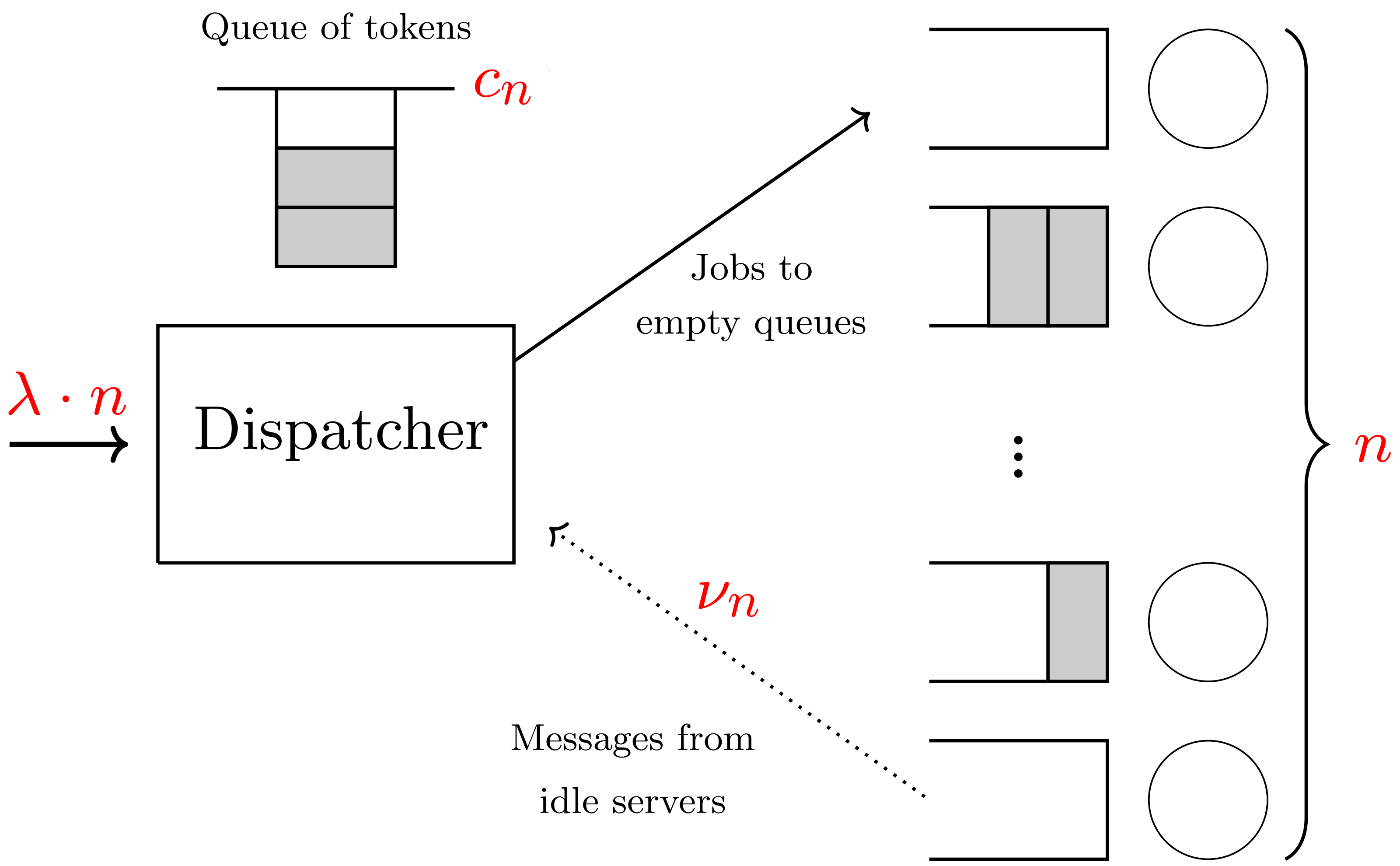
- idle processors send messages at rate ν_n

Our policy



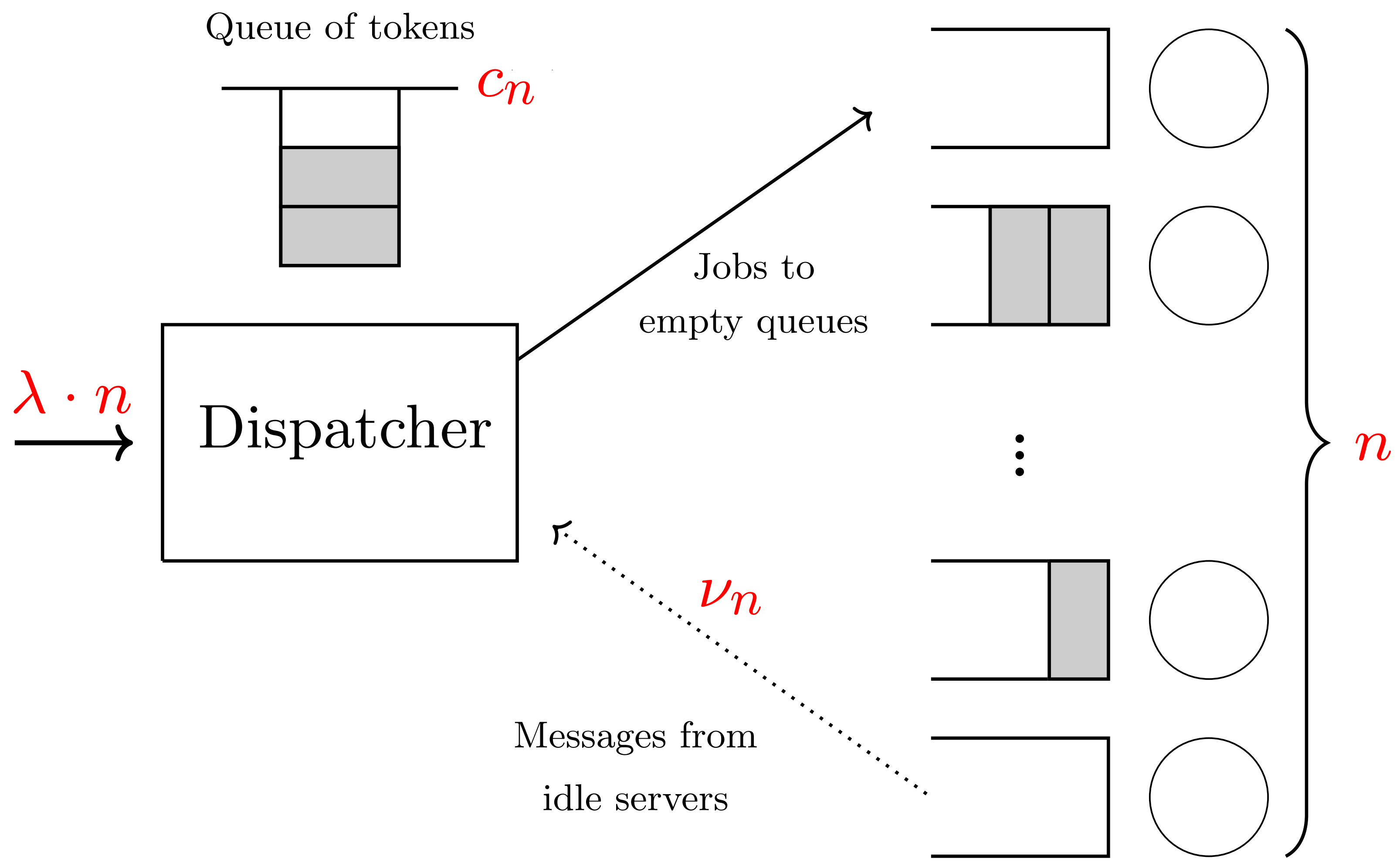
- idle processors send messages at rate ν_n
  - message rate per job $\approx \nu_n$

Our policy



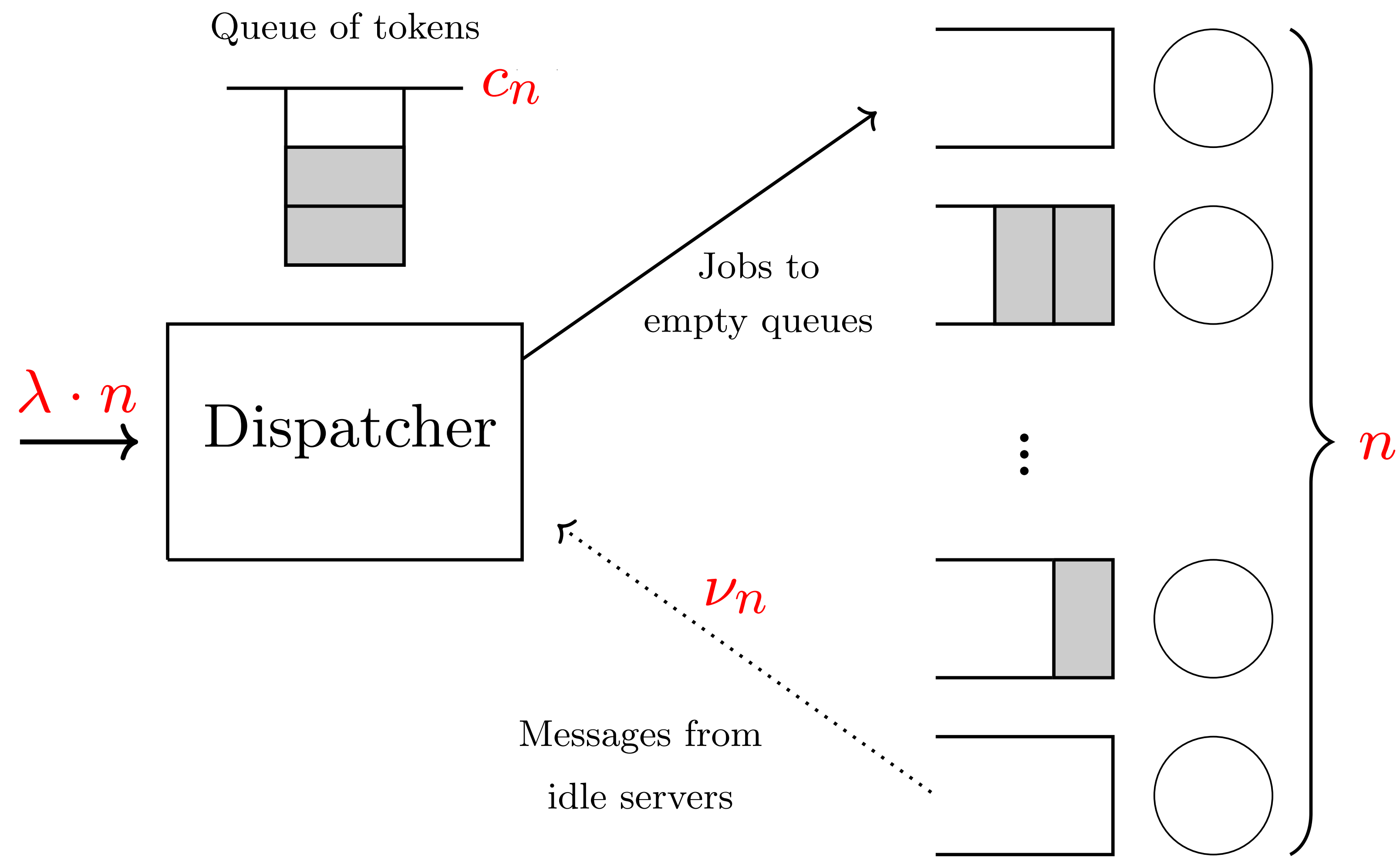
- idle processors send messages at rate ν_n
  - message rate per job $\approx \nu_n$

Our policy



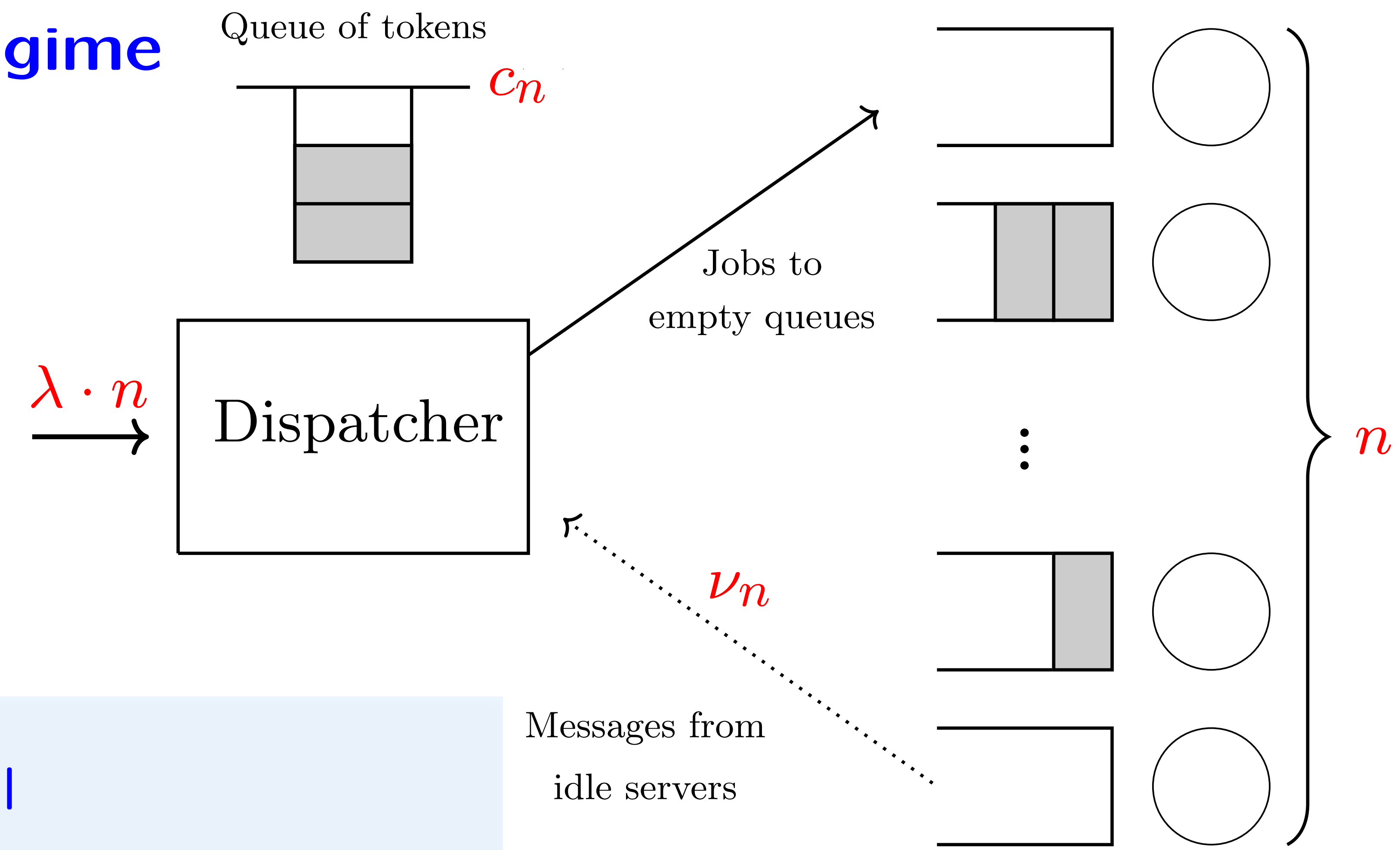
- idle processors send messages at rate ν_n
  - message rate per job $\approx \nu_n$
- make an entry in the memory, if there is room [c_n]
  - memory size $c_n \log n$

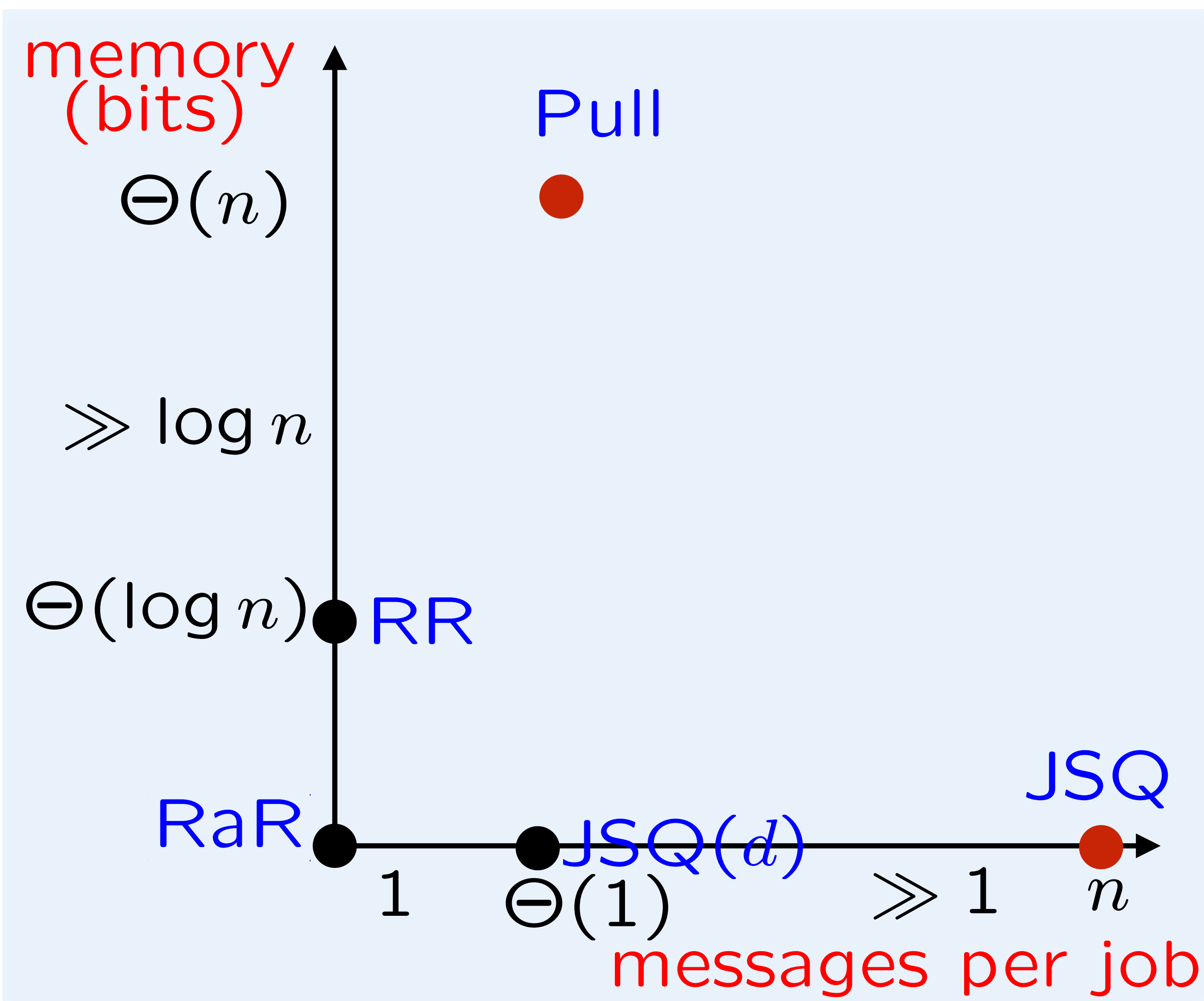
Our policy



- idle processors send messages at rate ν_n
  - message rate per job $\approx \nu_n$
- make an entry in the memory, if there is room [c_n]
  - memory size $c_n \log n$
- when job arrives:
  - send to server in memory
  - if empty memory, send to random server

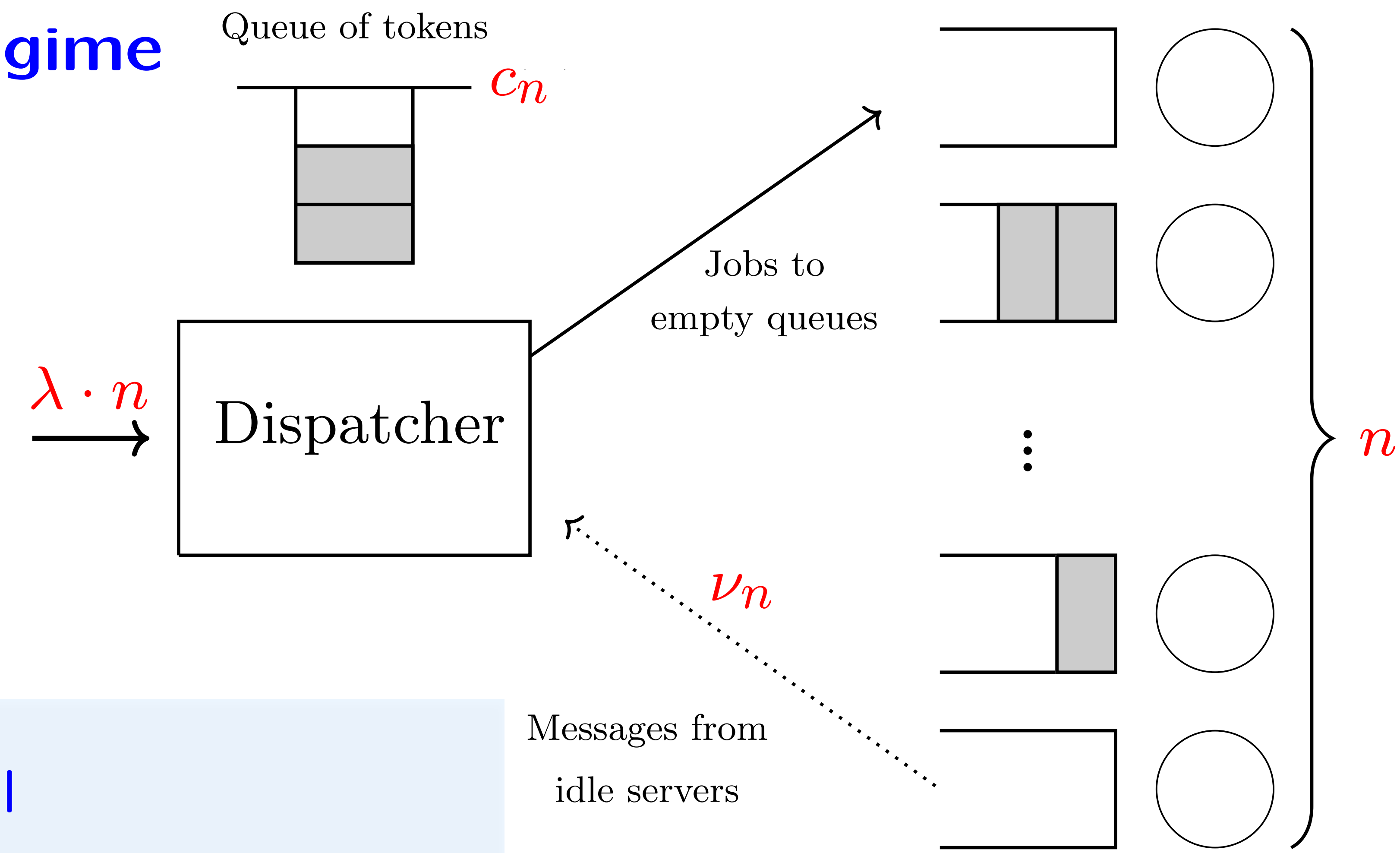
High message rate regime

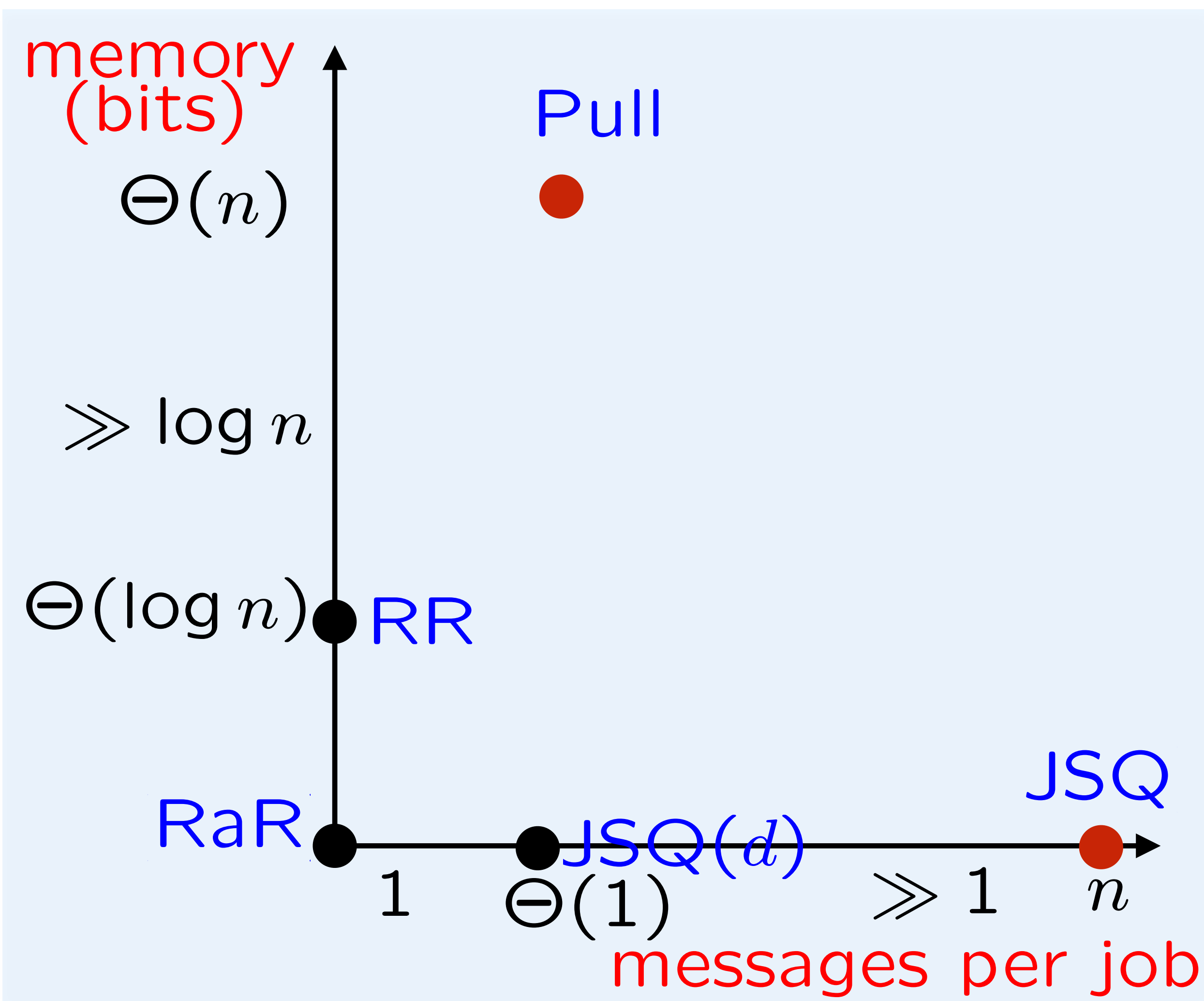




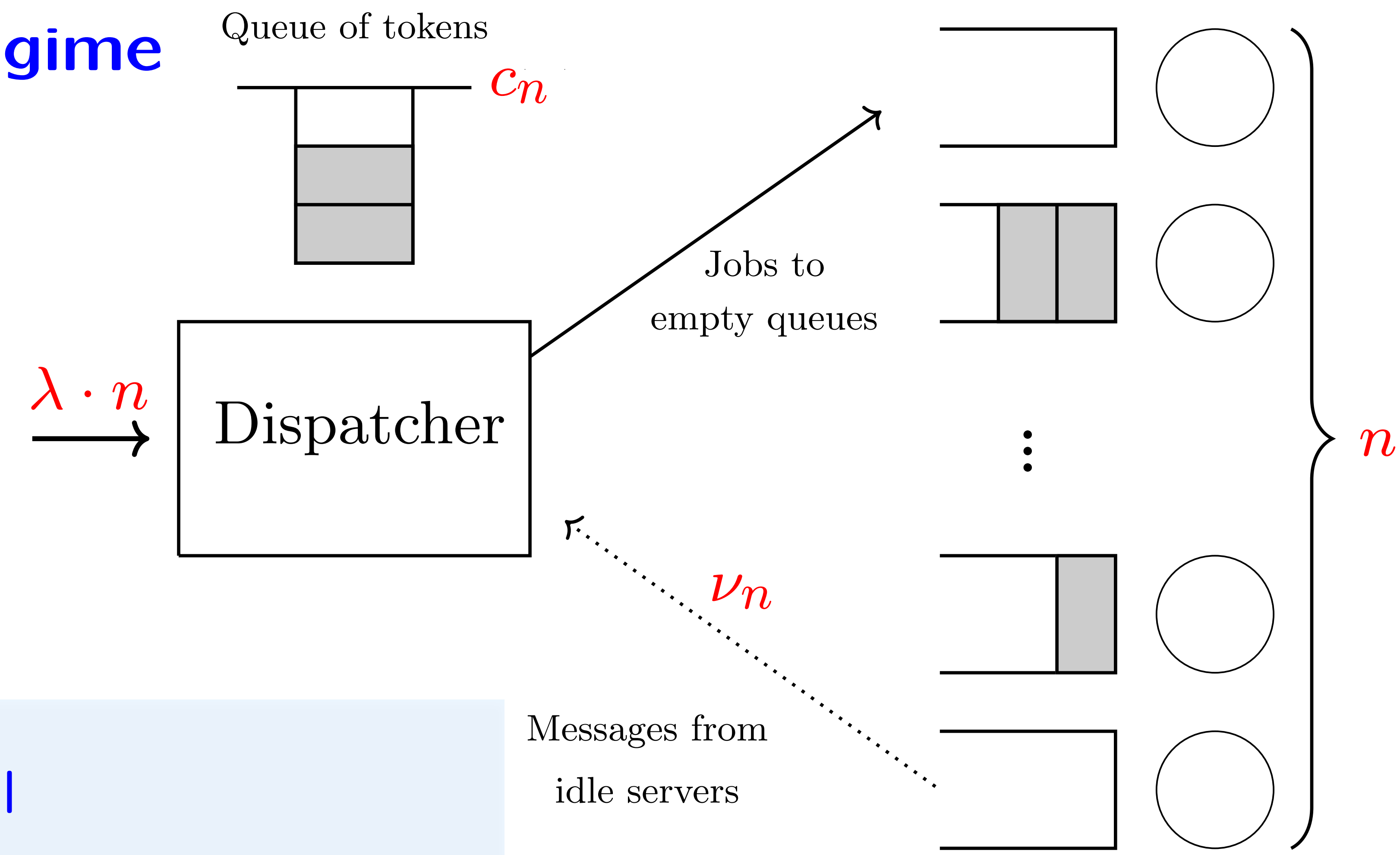
High message rate regime

- $\nu_n \to \infty$
- $c_n =$ constant ≥ 1

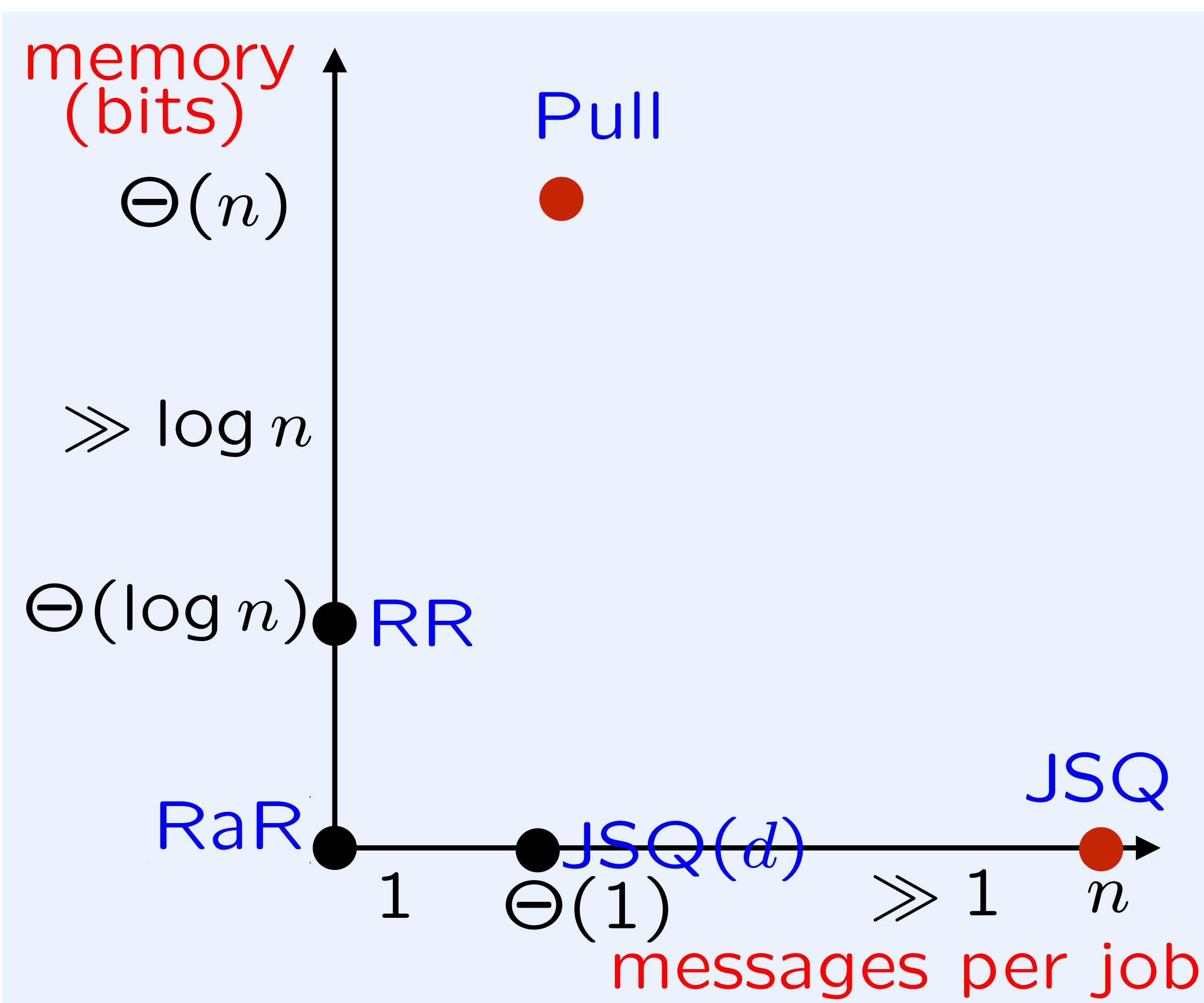




High message rate regime

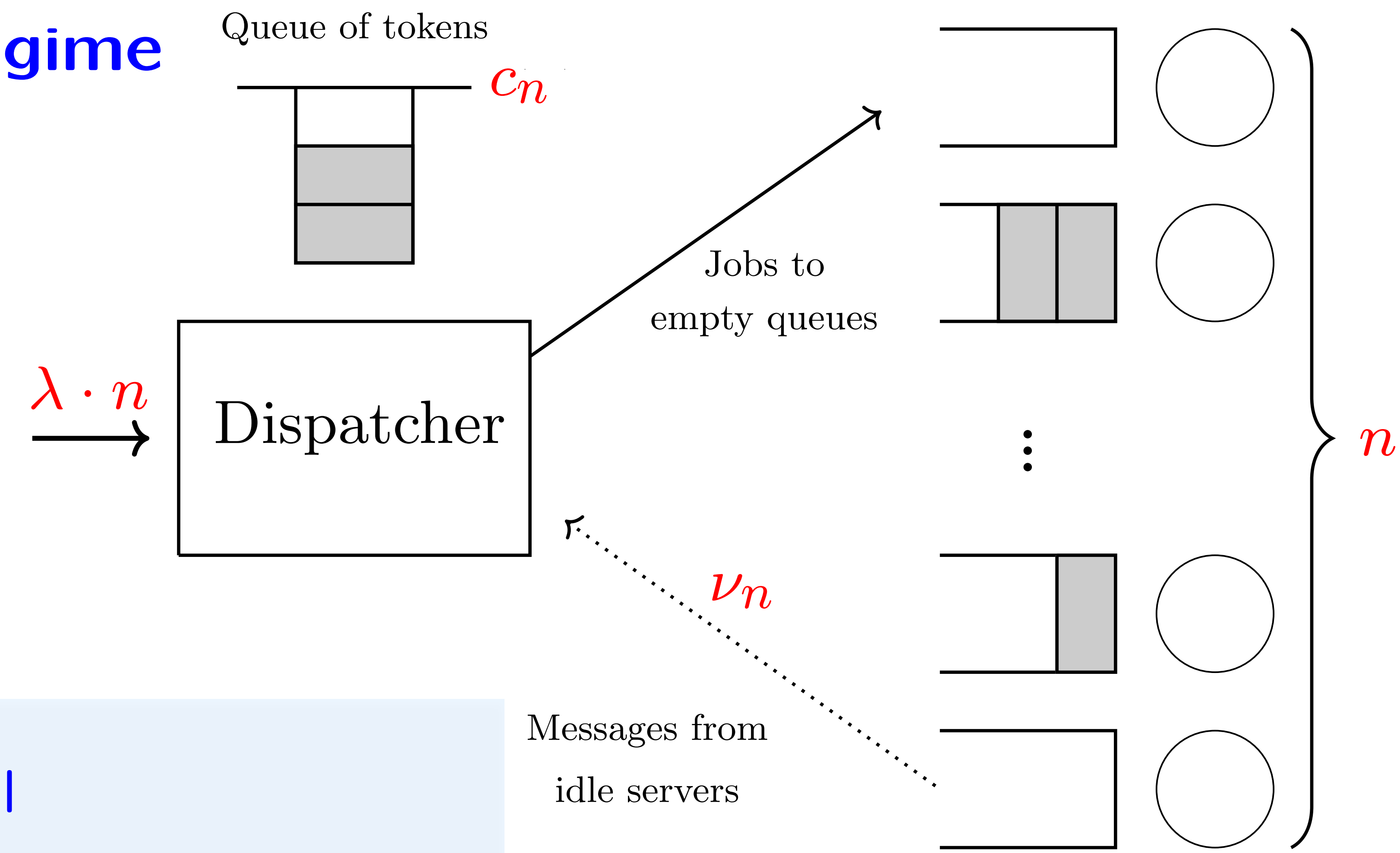


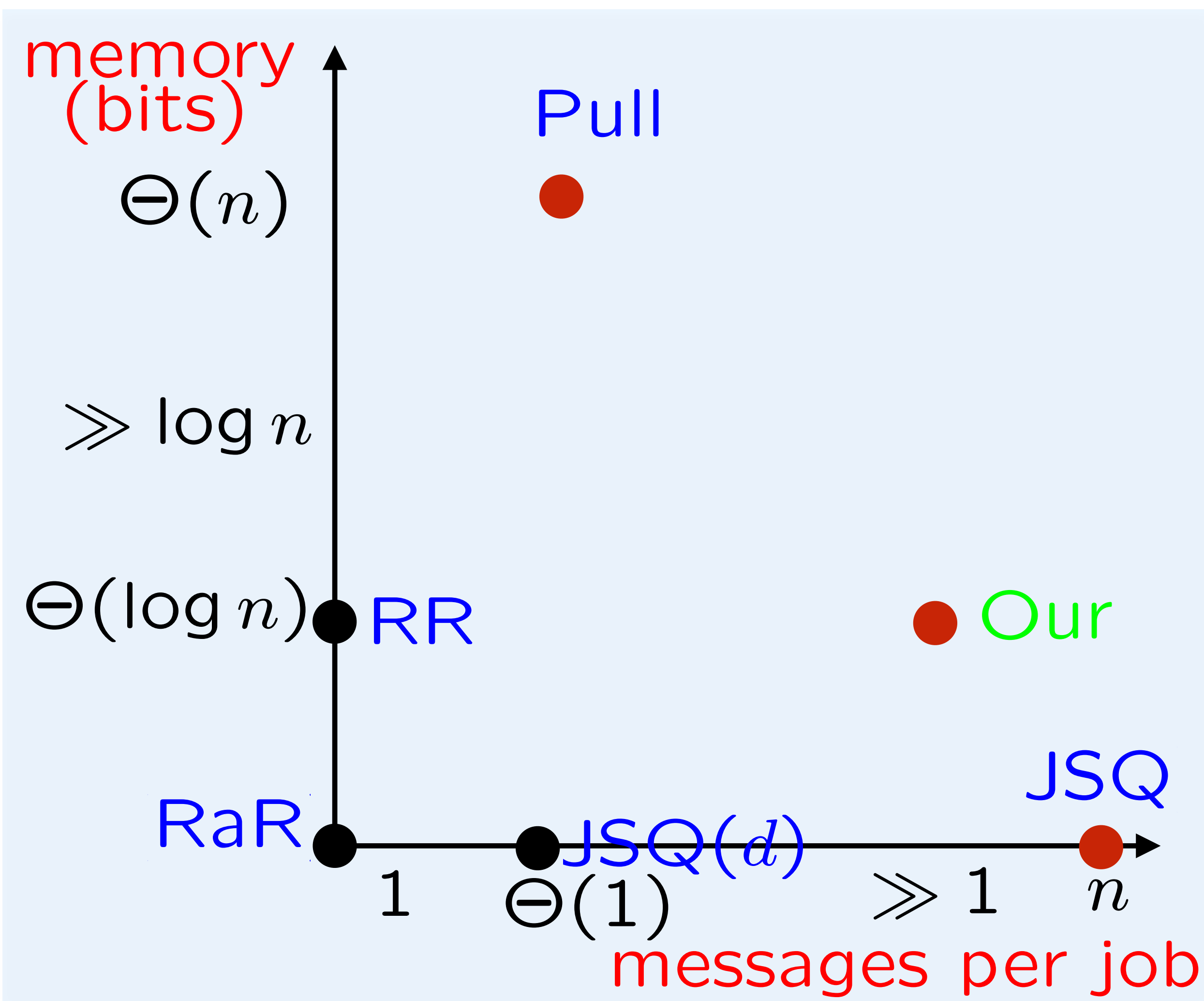
- $\nu_n \to \infty$
- $c_n =$ constant ≥ 1
- queueing delay $\to 0$



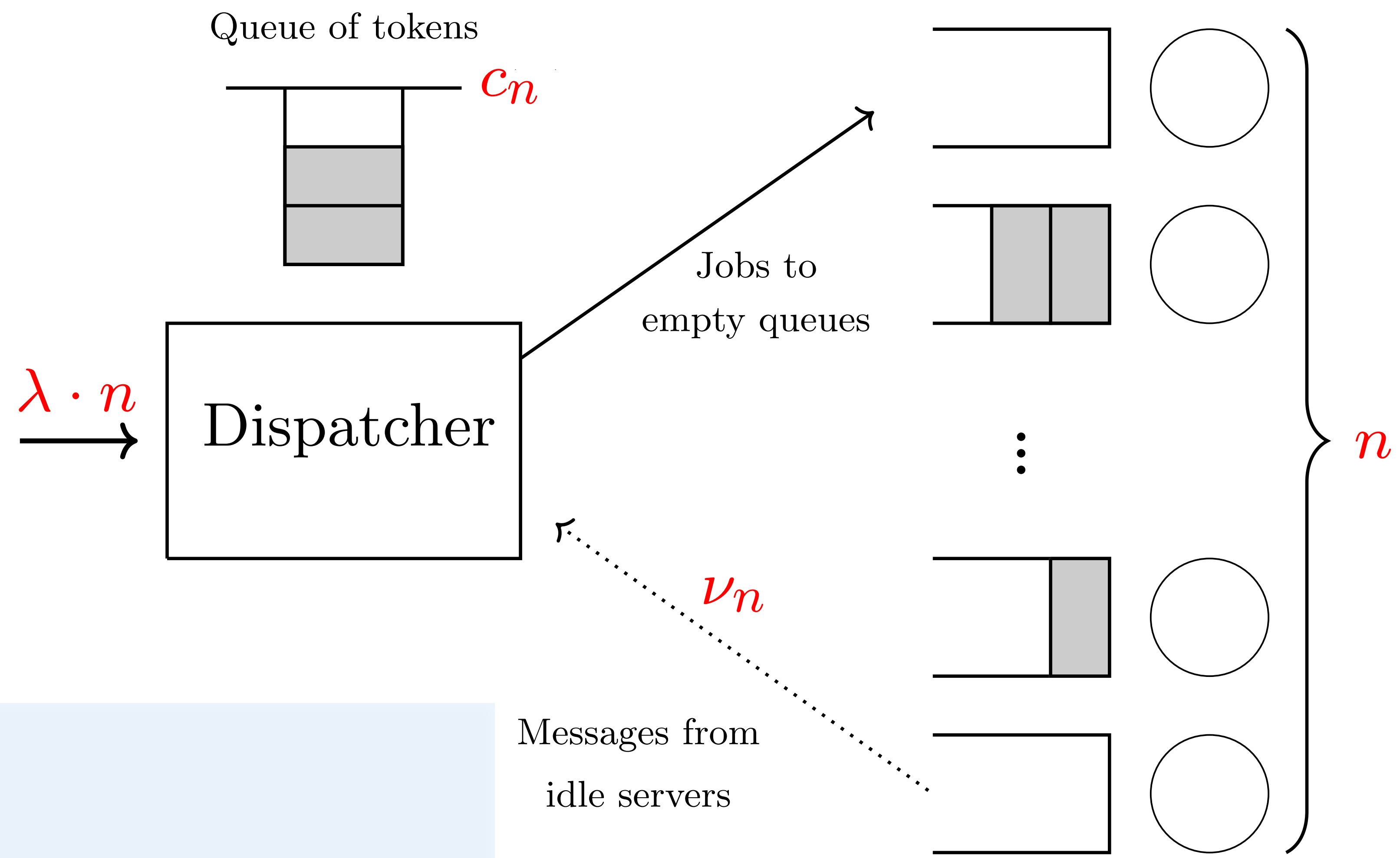
High message rate regime

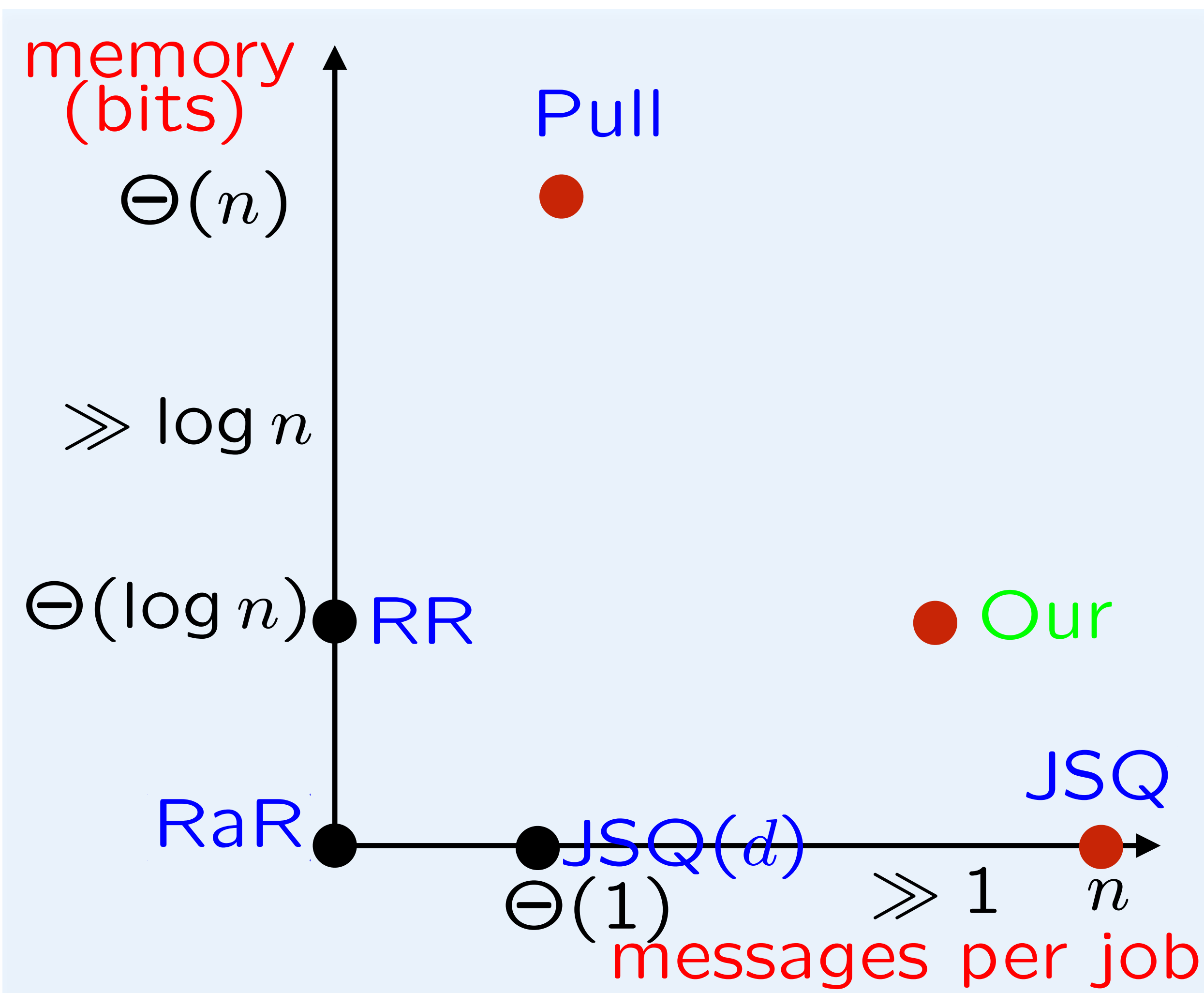
- $\nu_n \to \infty$
- $c_n =$ constant ≥ 1
- queueing delay $\to 0$





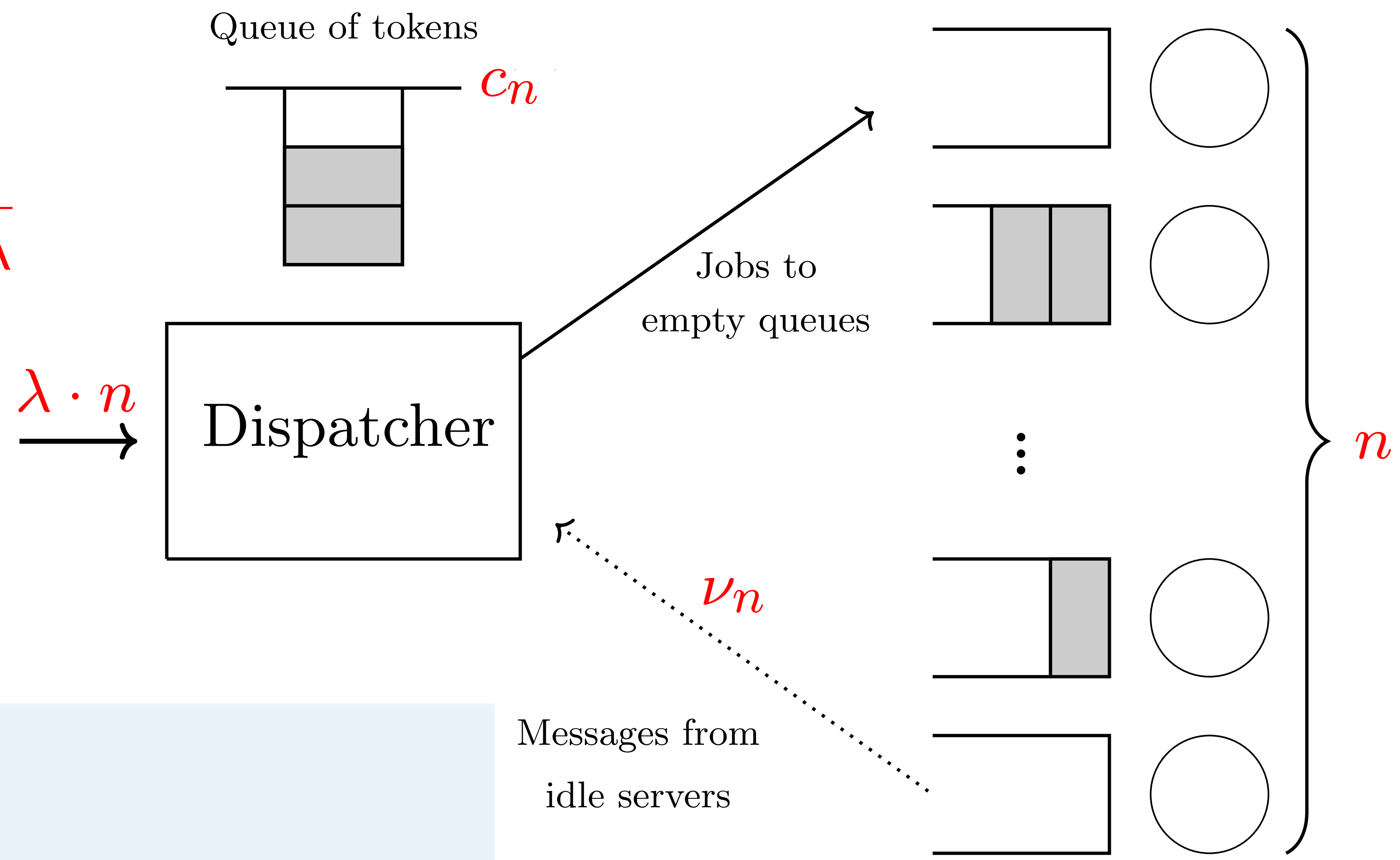
High memory regime

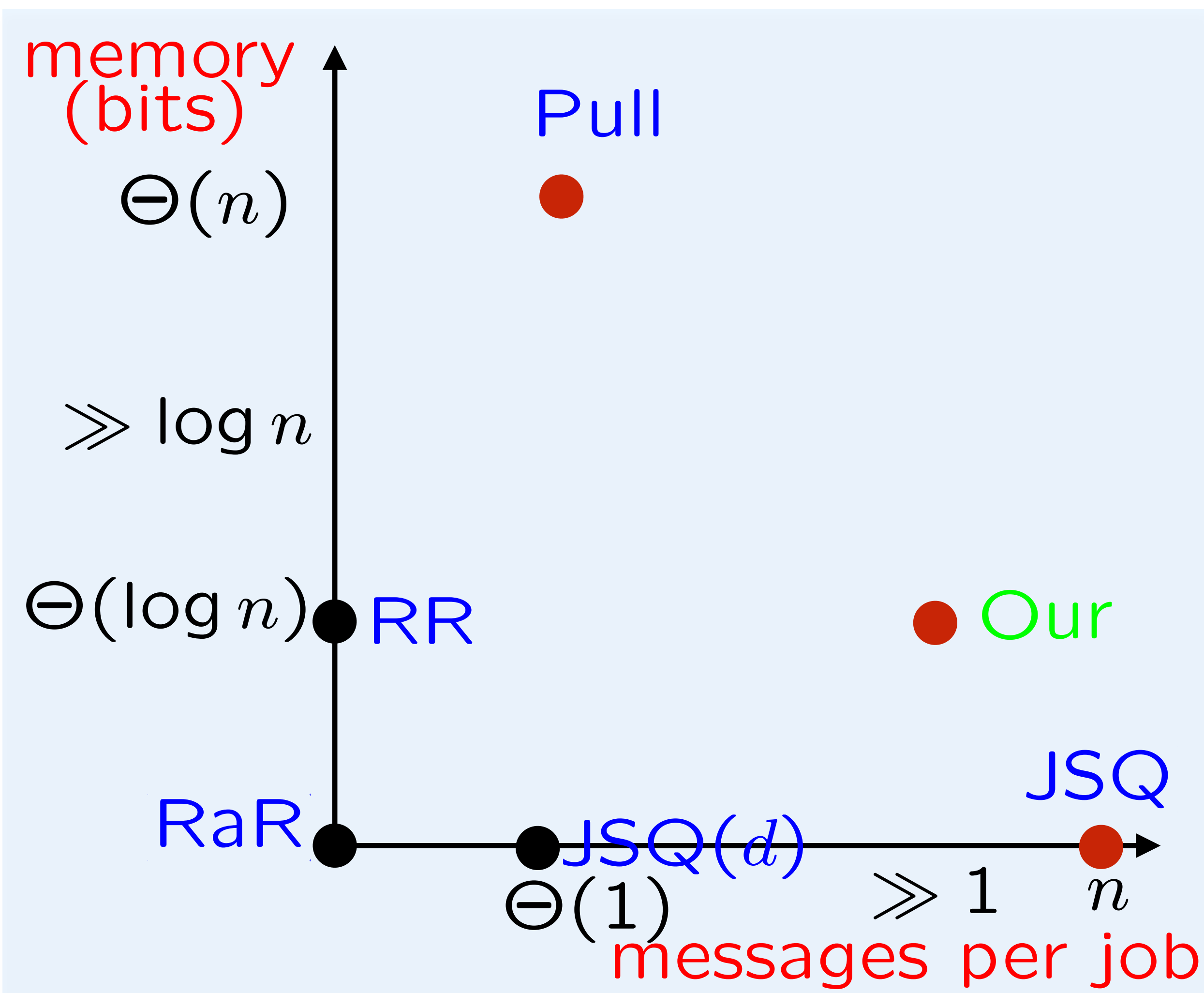




High memory regime

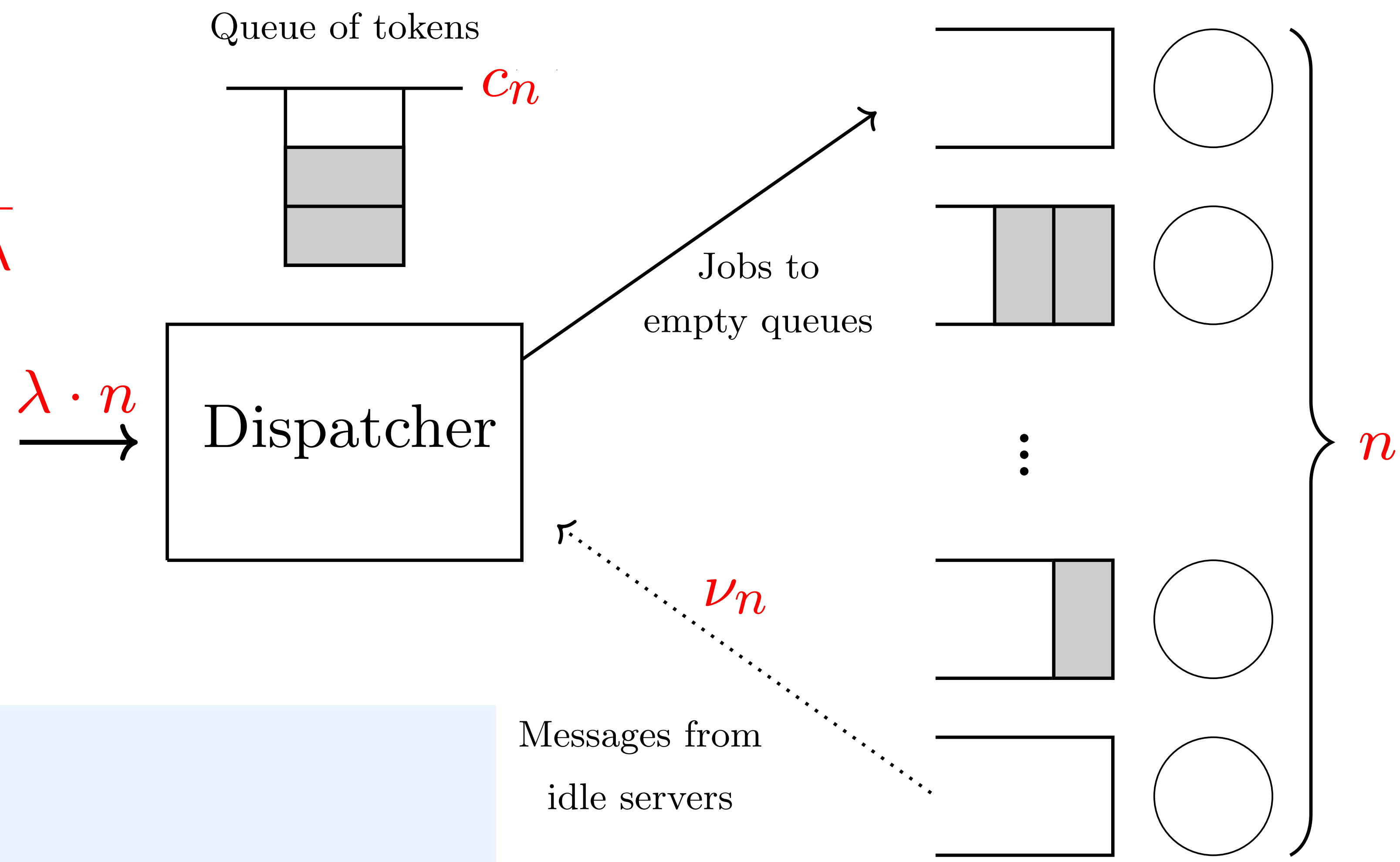
- $\nu_n = \text{constant} \geq \dfrac{\lambda}{1-\lambda}$

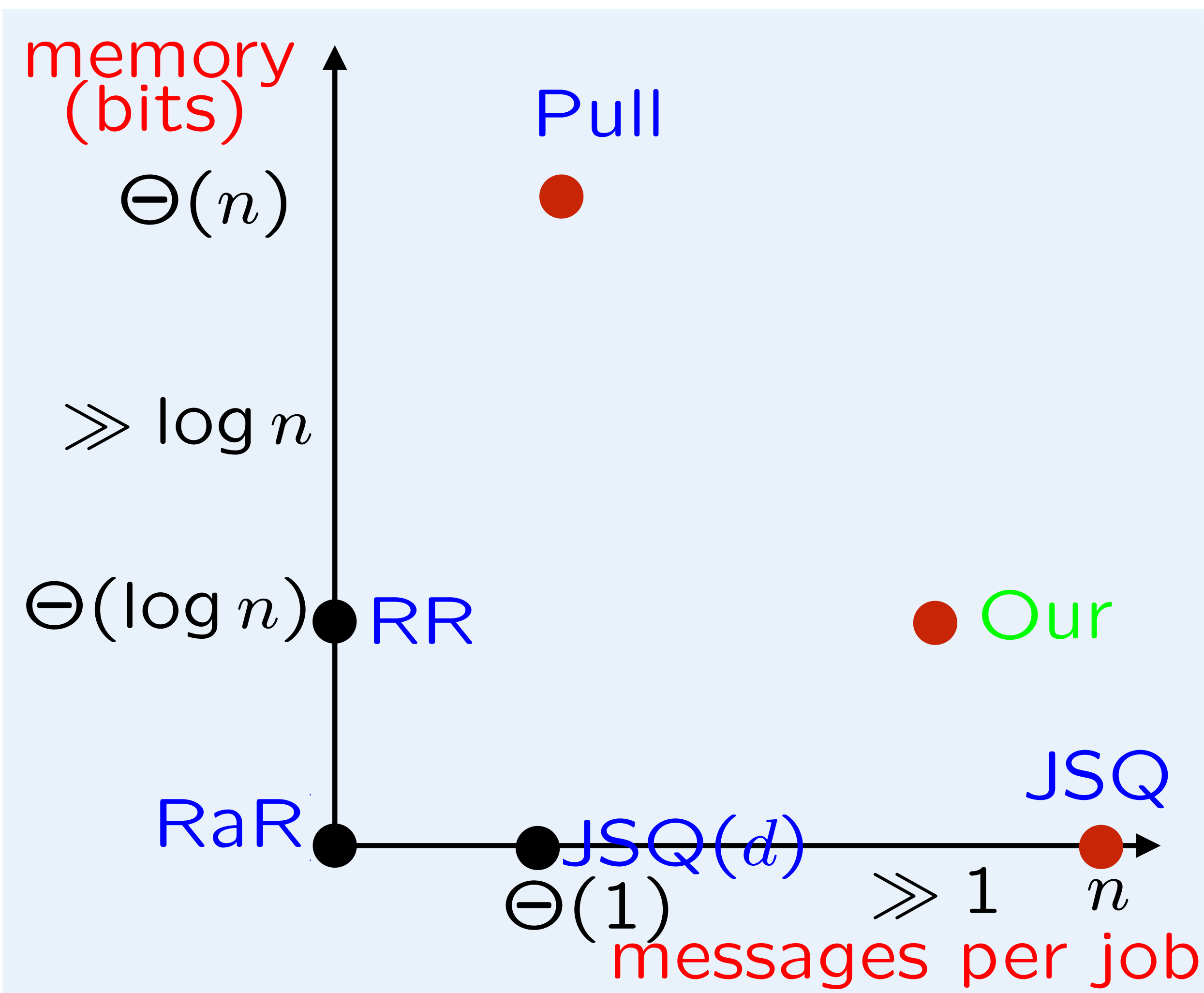




High memory regime

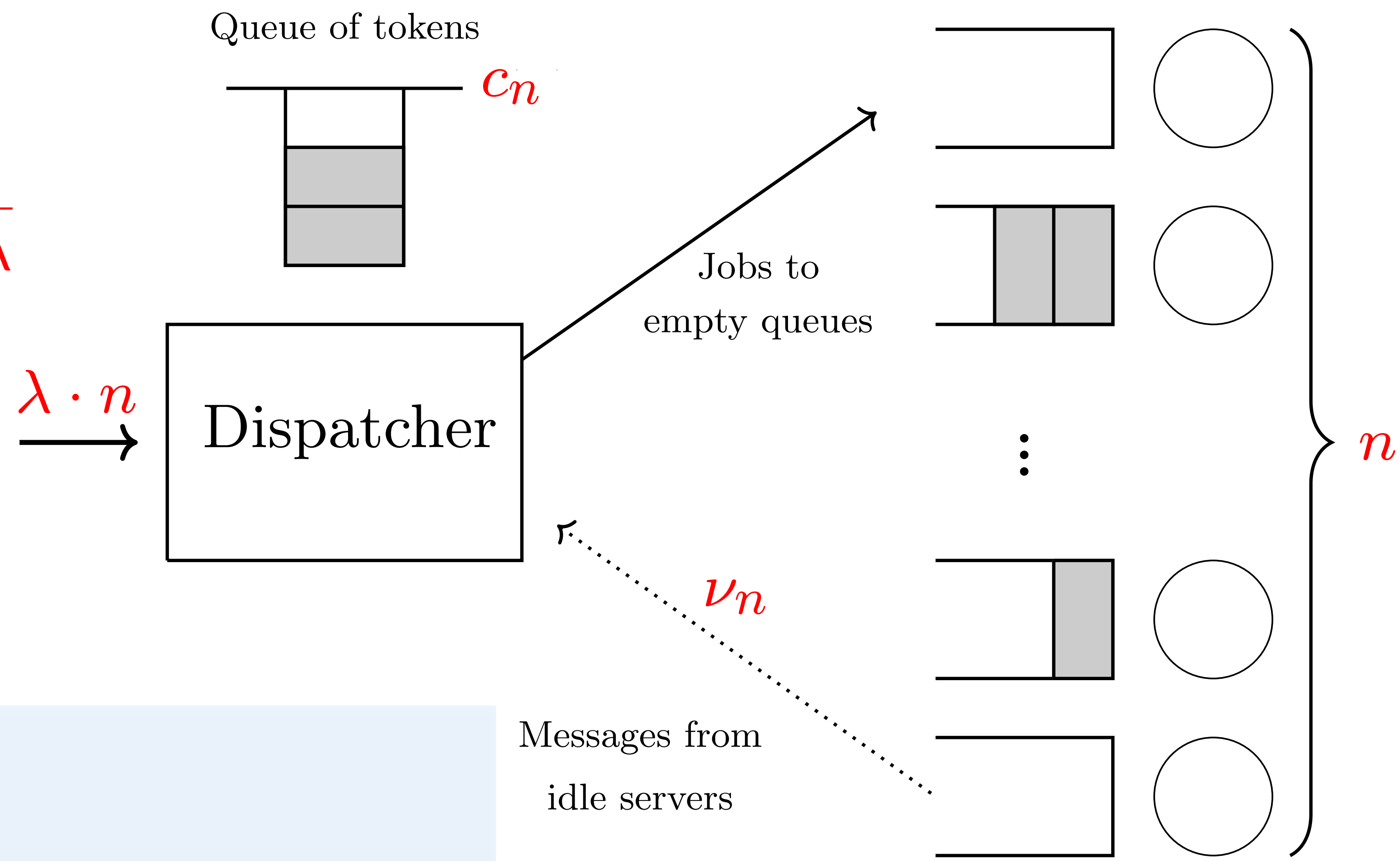
- $\nu_n = \text{constant} \geq \dfrac{\lambda}{1-\lambda}$
- $c_n \to \infty$

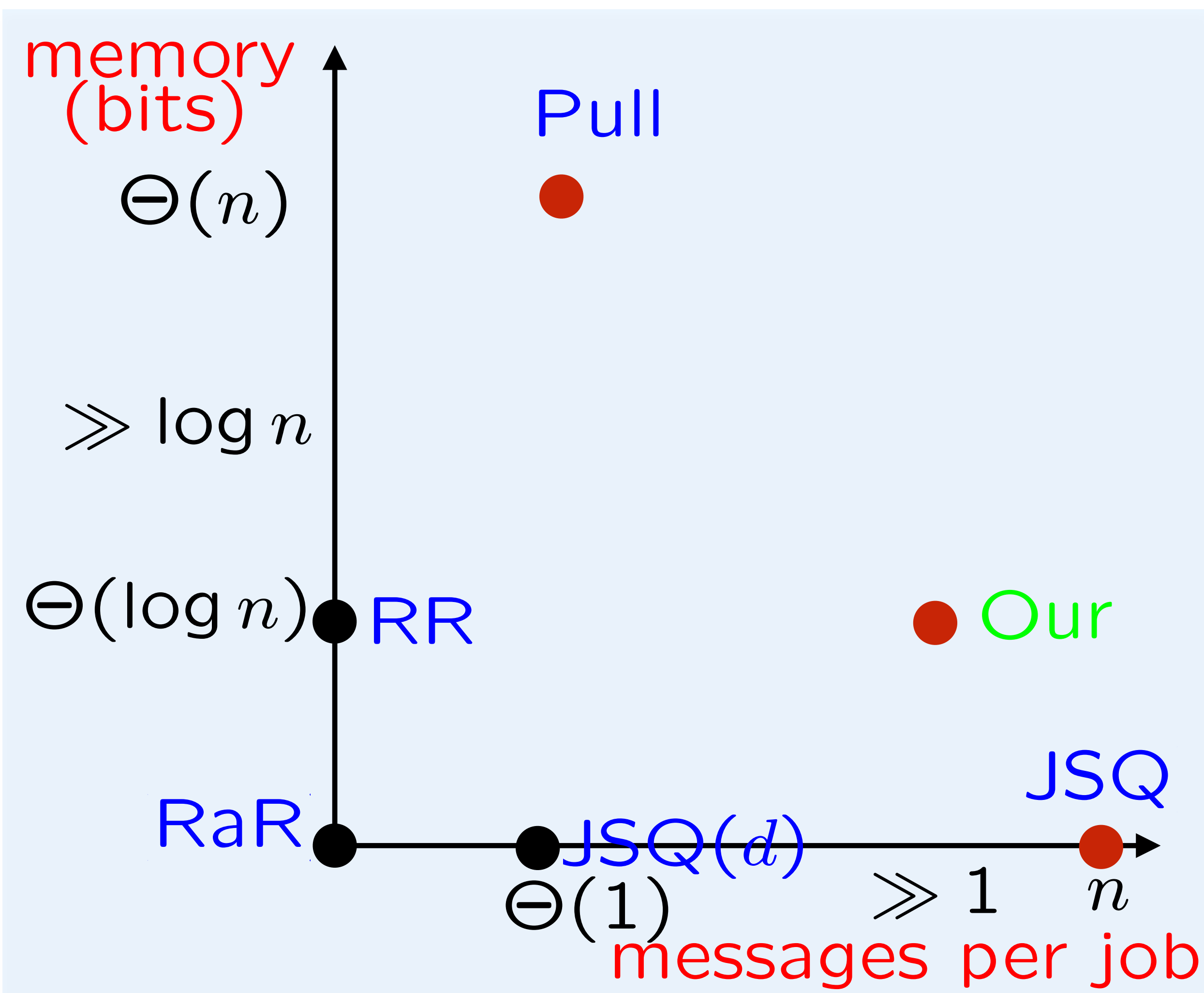




High memory regime

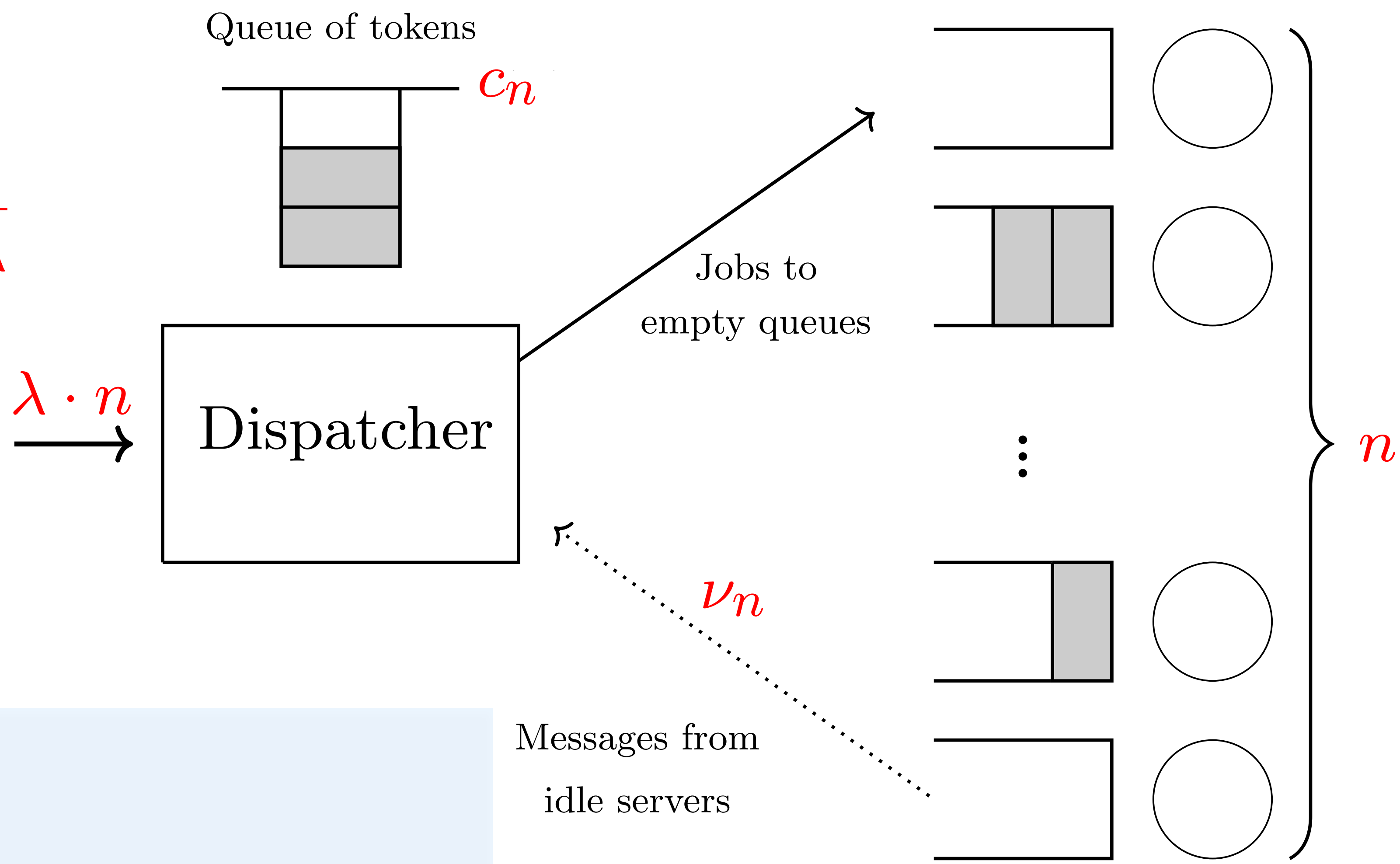
- $\nu_n = \text{constant} \geq \dfrac{\lambda}{1-\lambda}$
- $c_n \to \infty$
- queueing delay $\to 0$

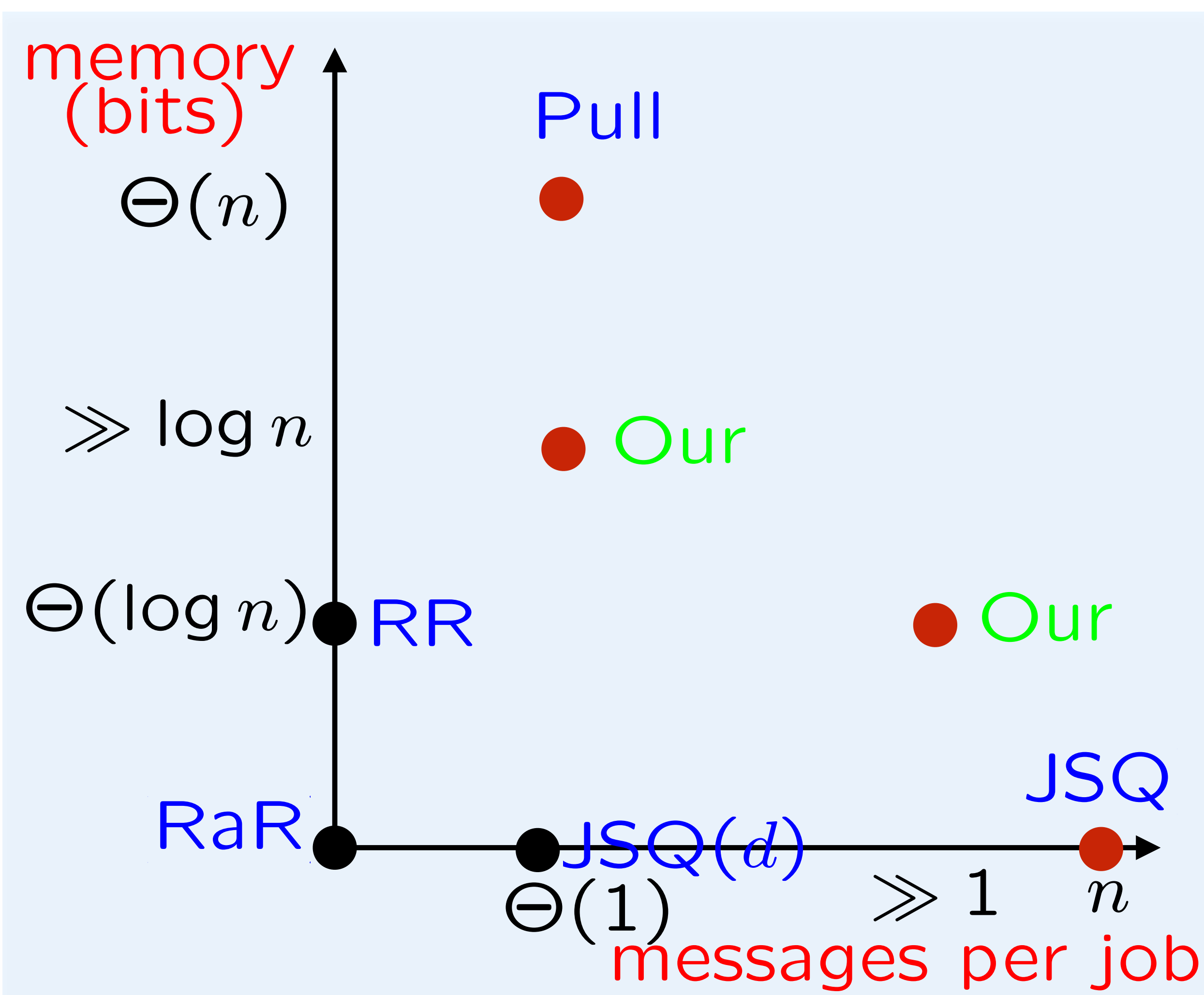




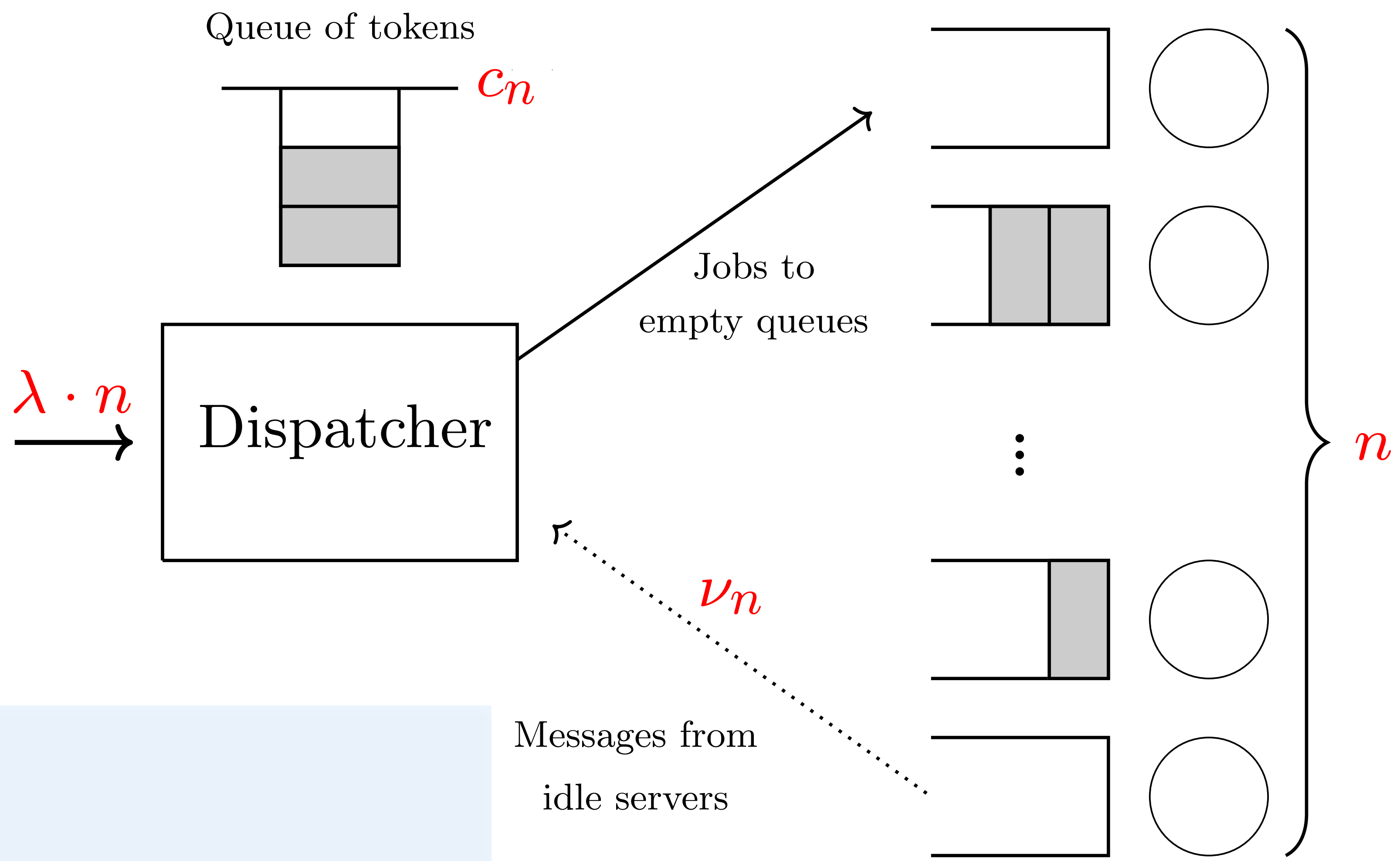
High memory regime

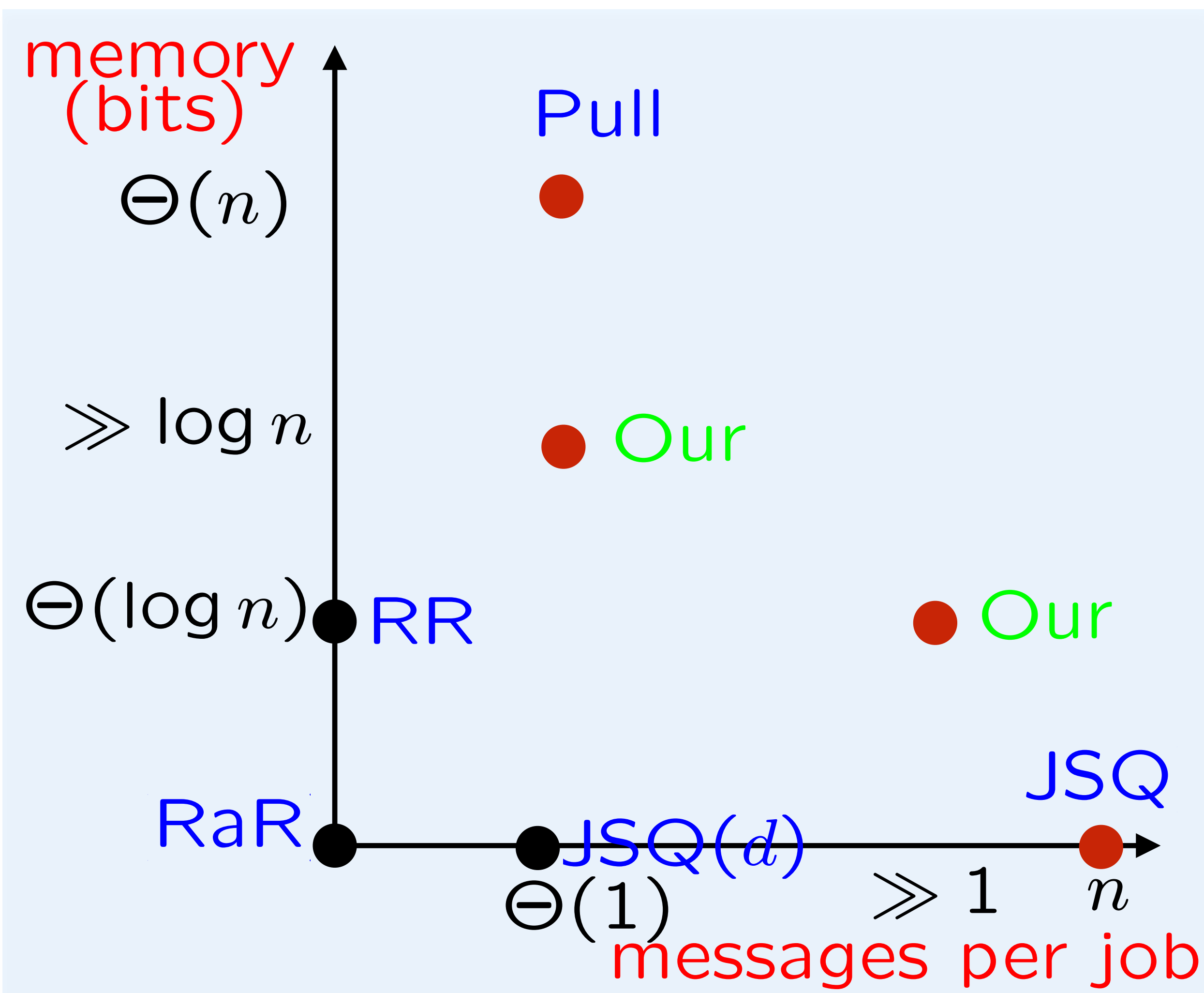
- $\nu_n = \text{constant} \geq \dfrac{\lambda}{1-\lambda}$
- $c_n \to \infty$
- queueing delay $\to 0$





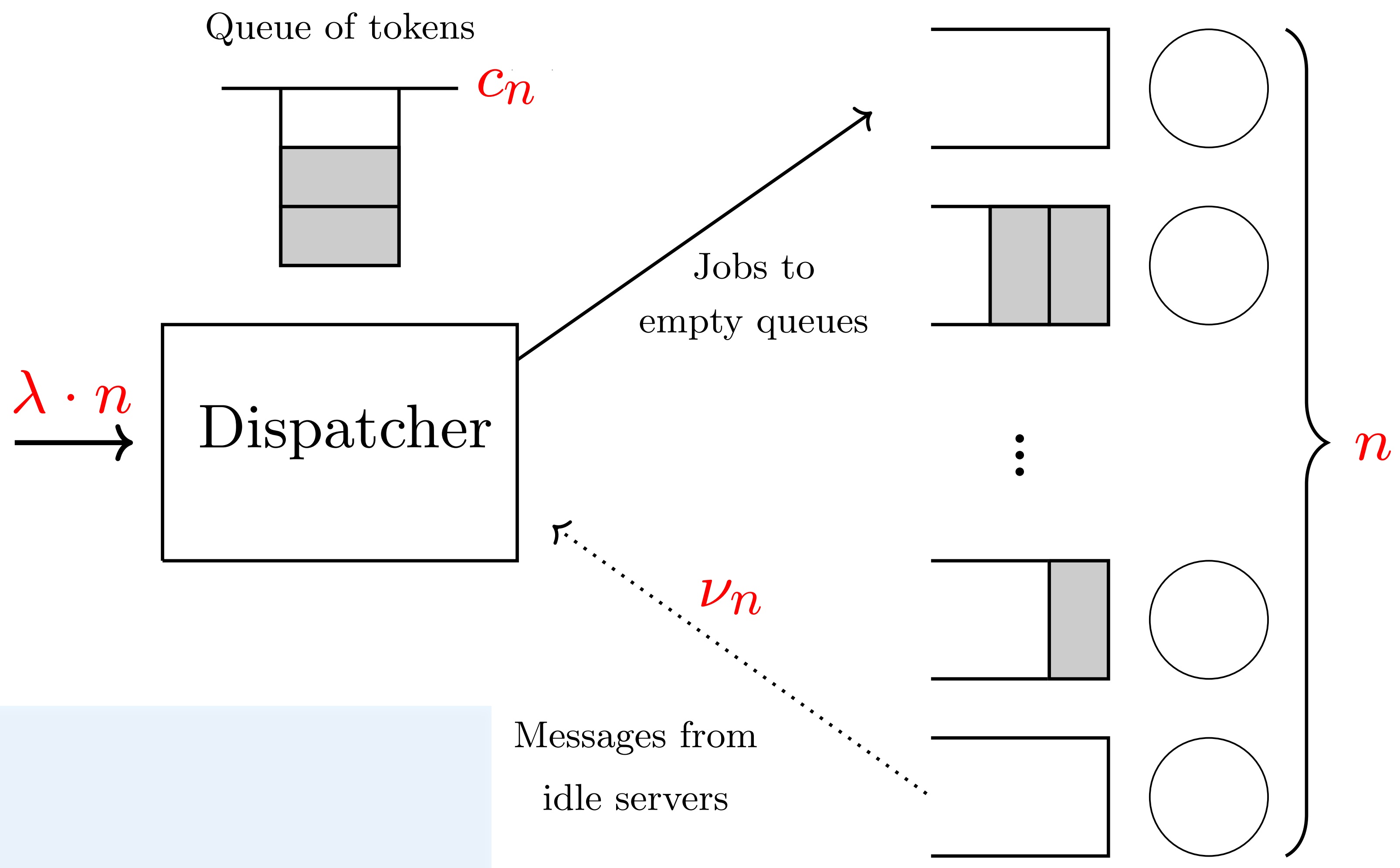
Parsimonious regime

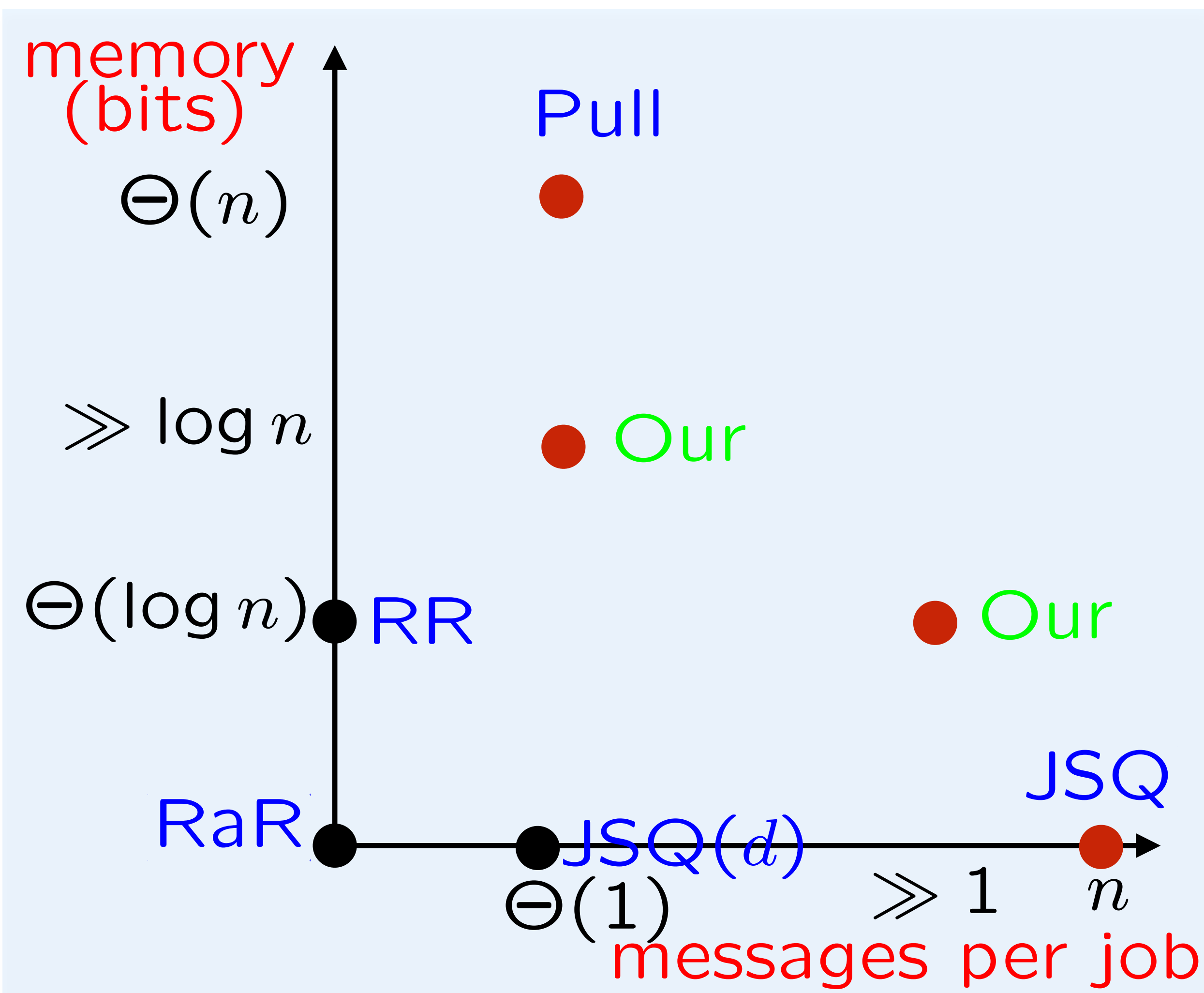




Parsimonious regime

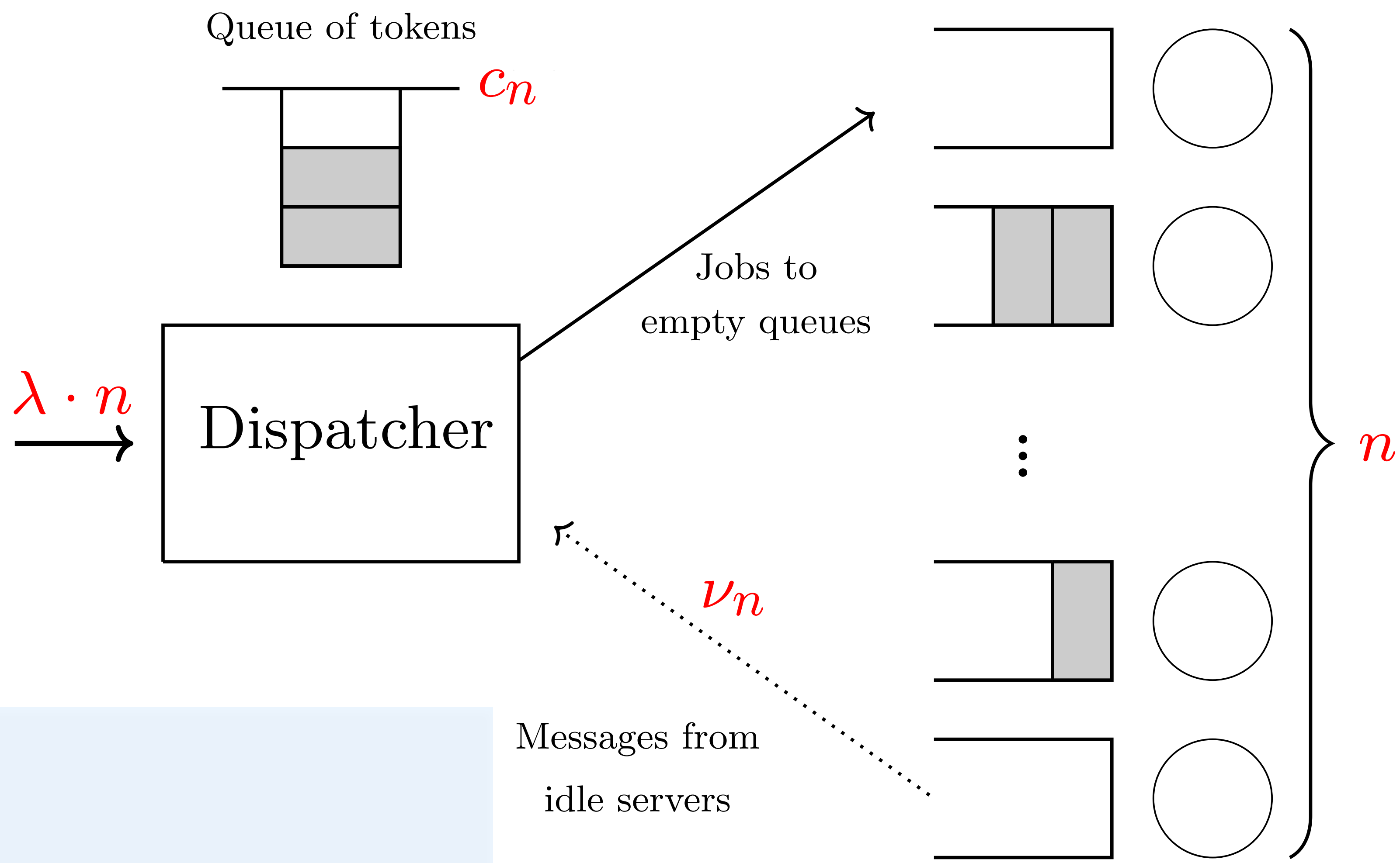
- $\nu_n =$ constant
- $c_n =$ constant

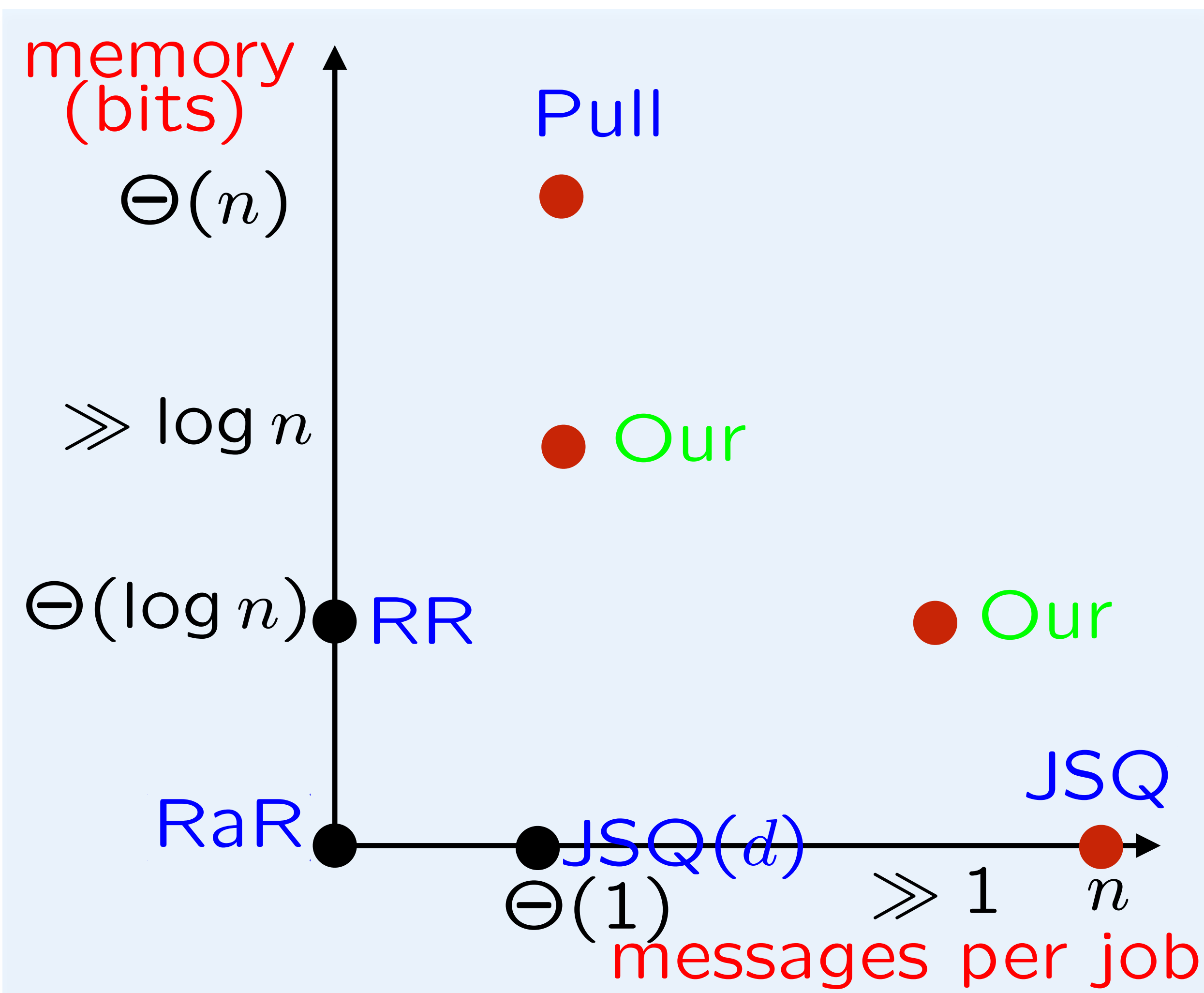




Parsimonious regime

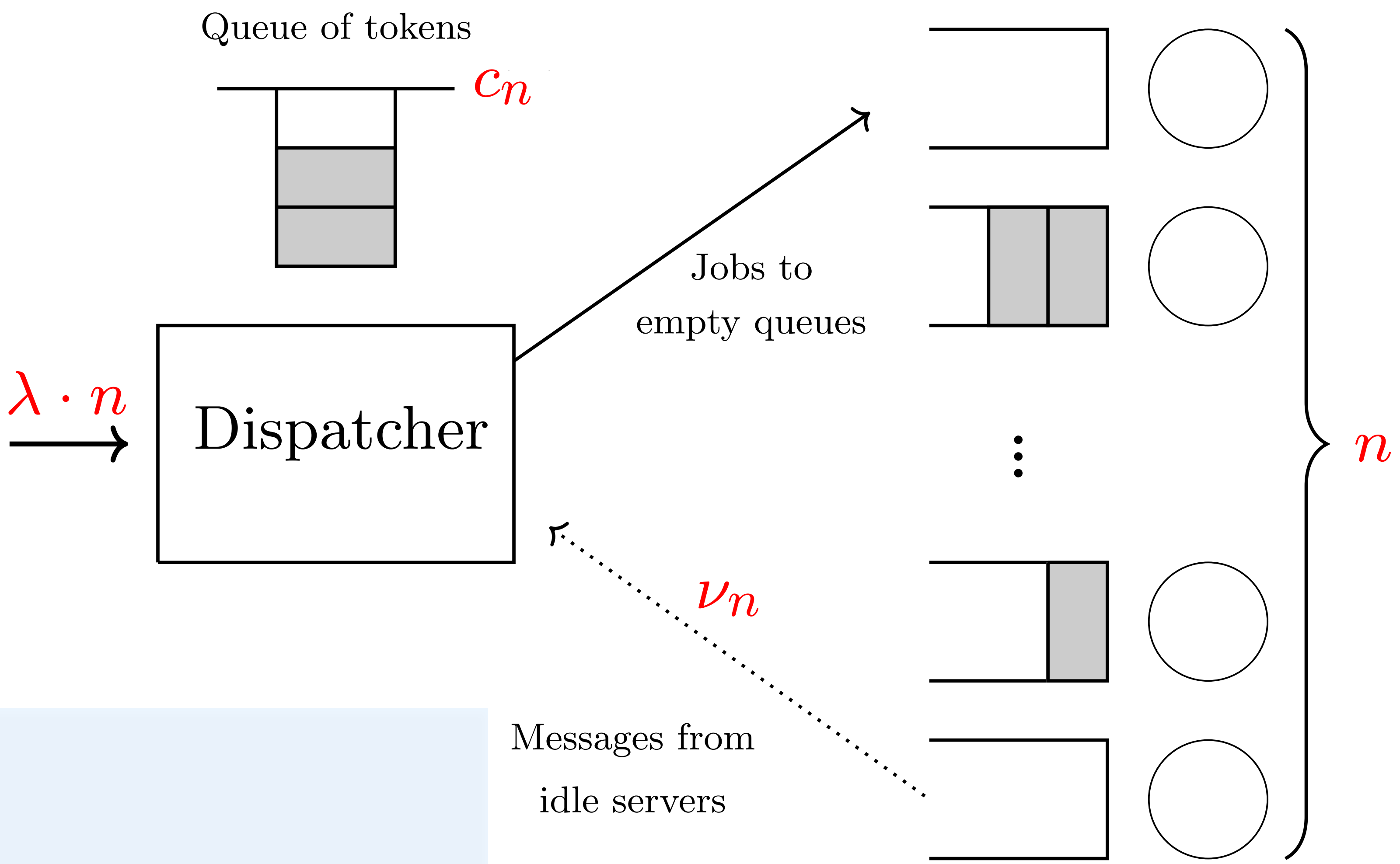
- $\nu_n =$ constant
- $c_n =$ constant
- queueing delay $\not\to 0$

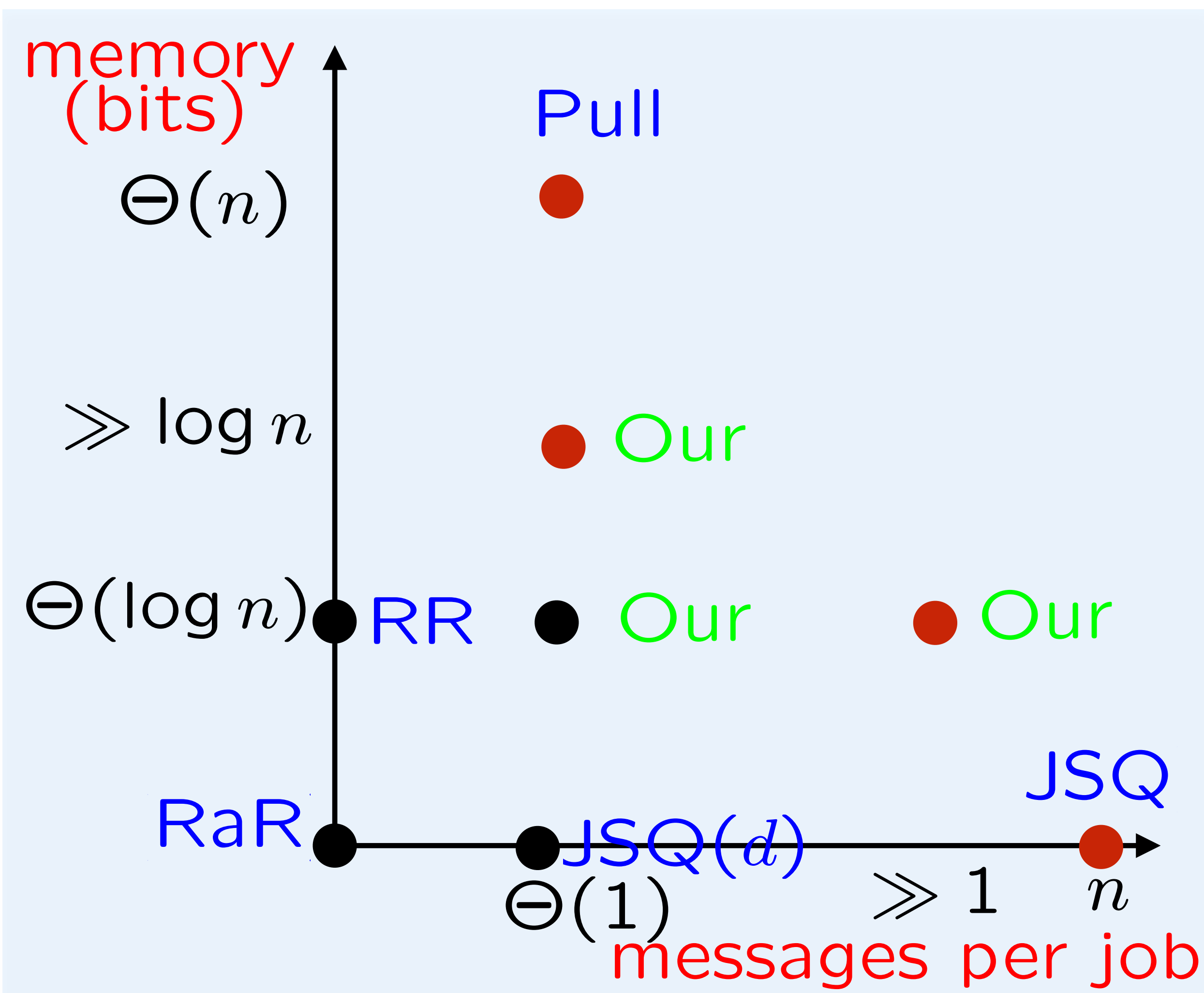




Parsimonious regime

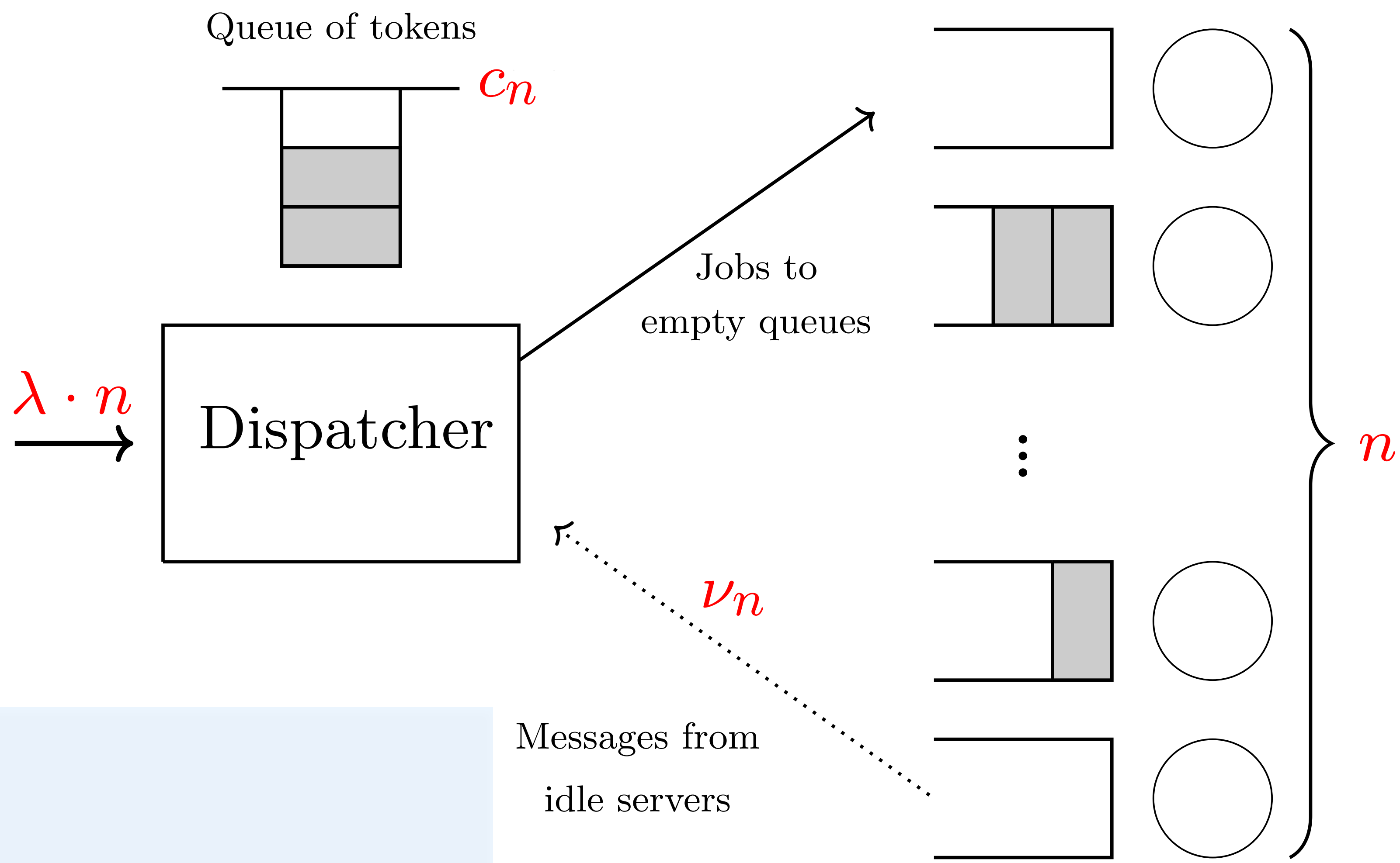
- ν_n = constant
- c_n = constant
- queueing delay $\not\to 0$



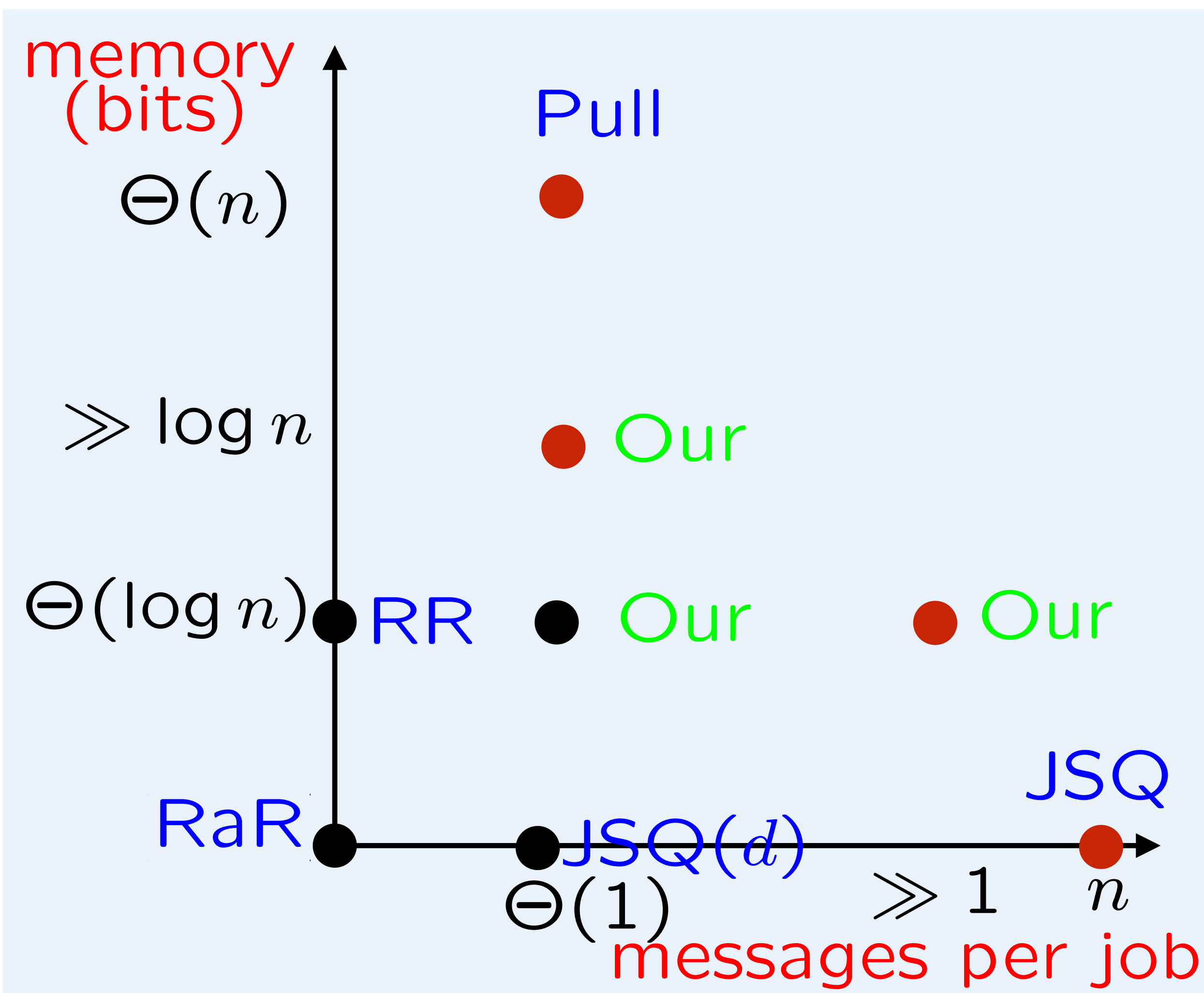


Parsimonious regime

- $\nu_n =$ constant
- $c_n =$ constant
- queueing delay $\not\to 0$

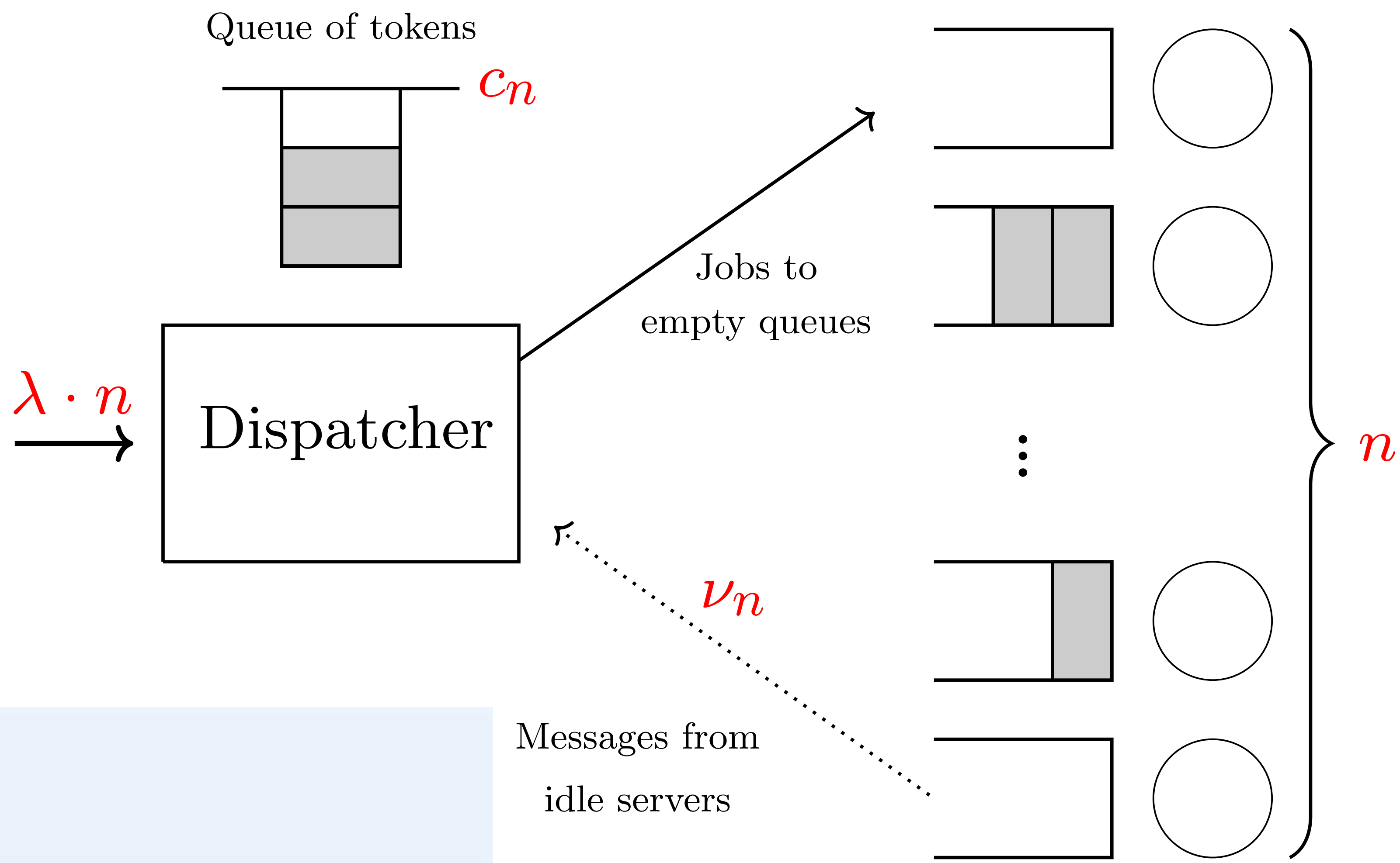


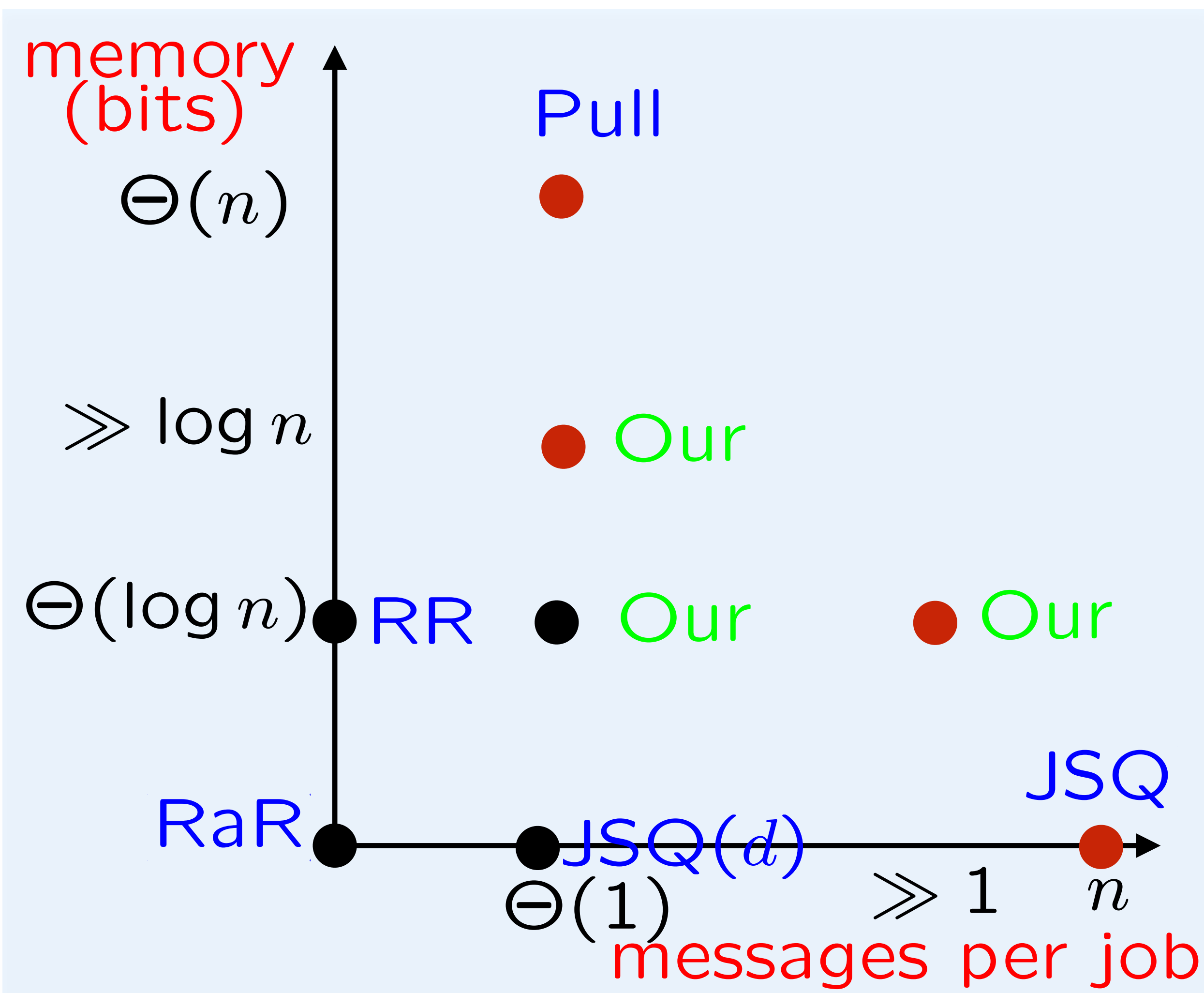
- Expressions for delay via fluid models



Parsimonious regime

- $\nu_n =$ constant
- $c_n =$ constant
- queueing delay $\not\to 0$

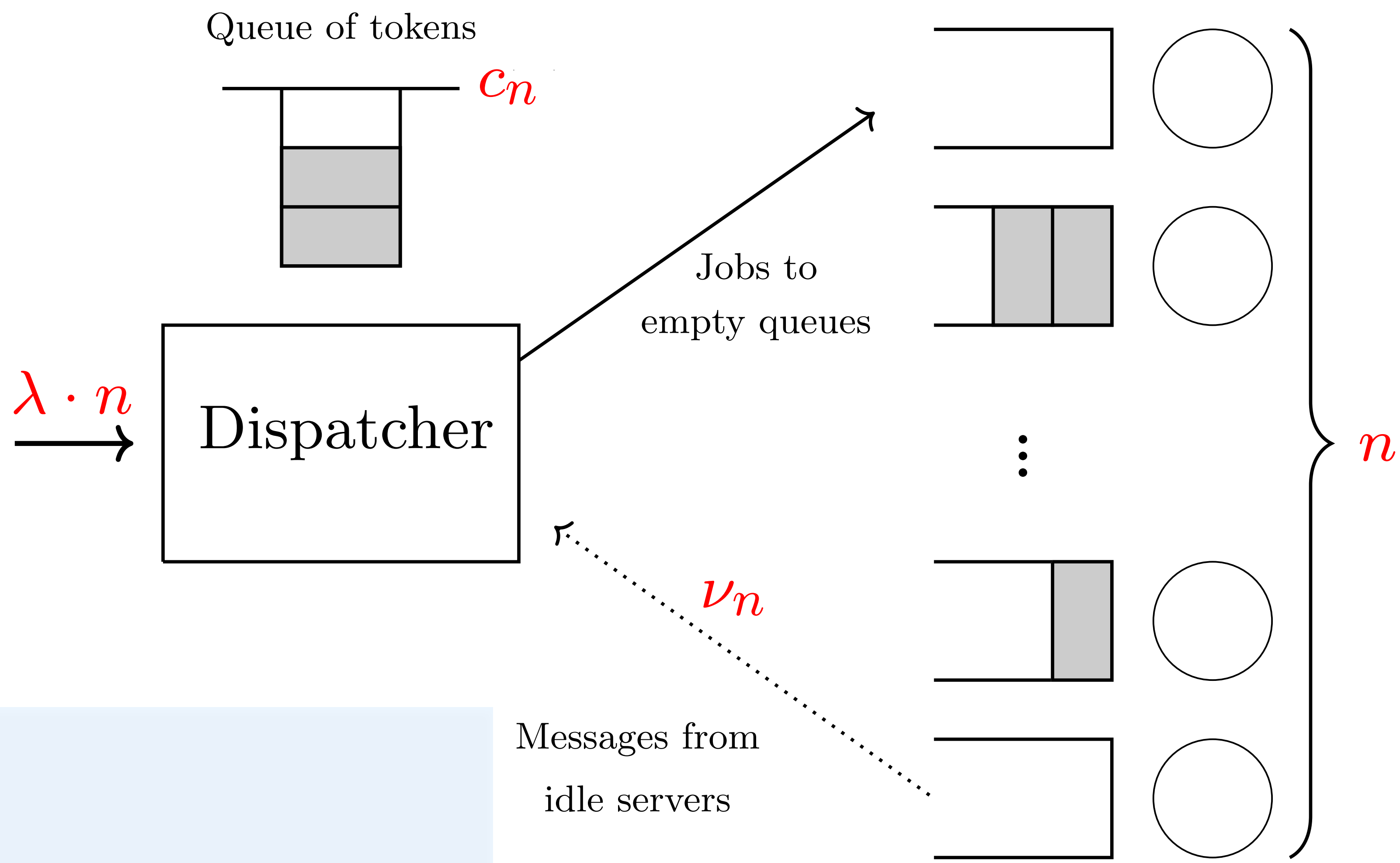


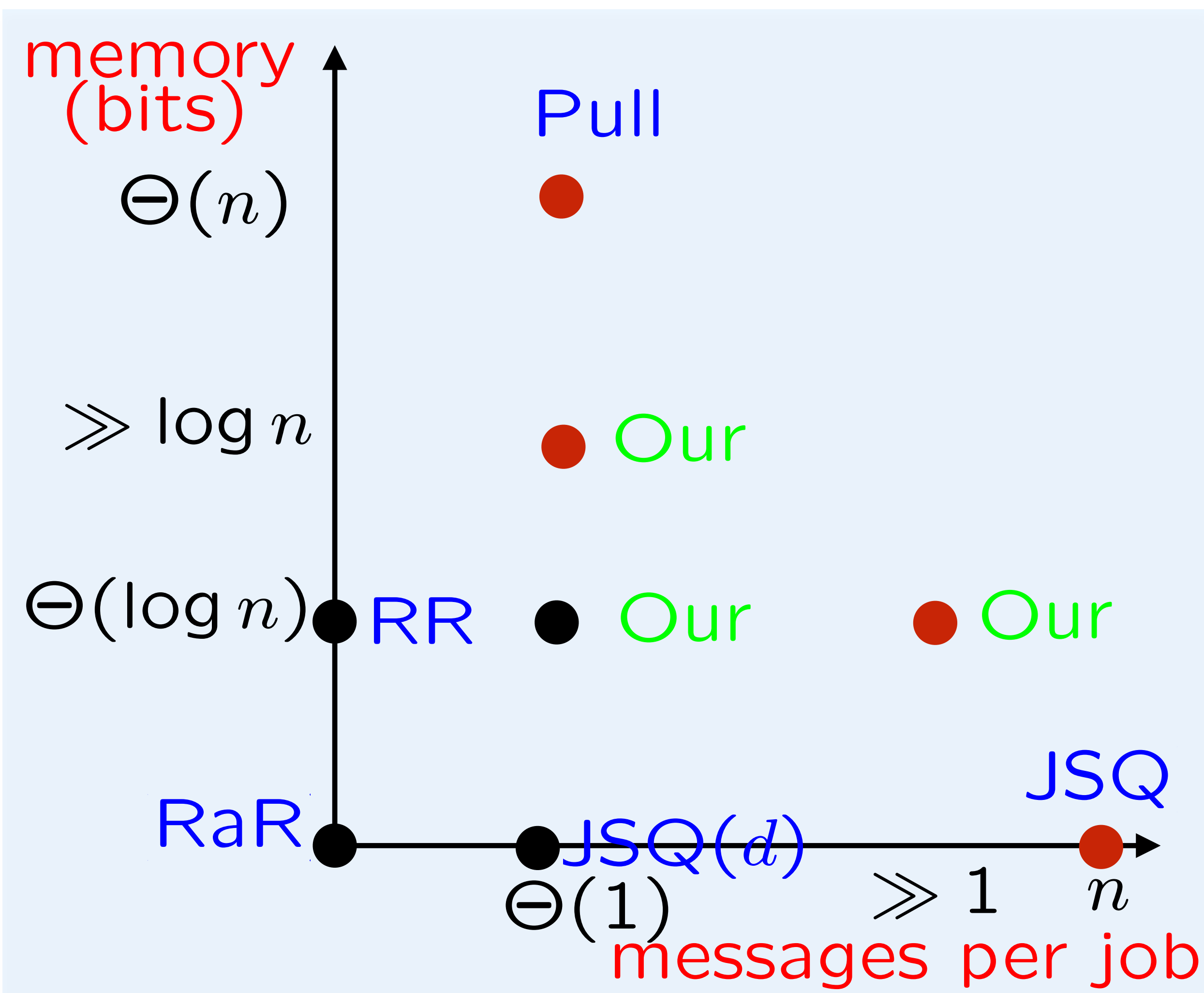


- Expressions for delay via fluid models
- Bounded delay as $\lambda \to 1$

Parsimonious regime

- $\nu_n =$ constant
- $c_n =$ constant
- queueing delay $\not\to 0$

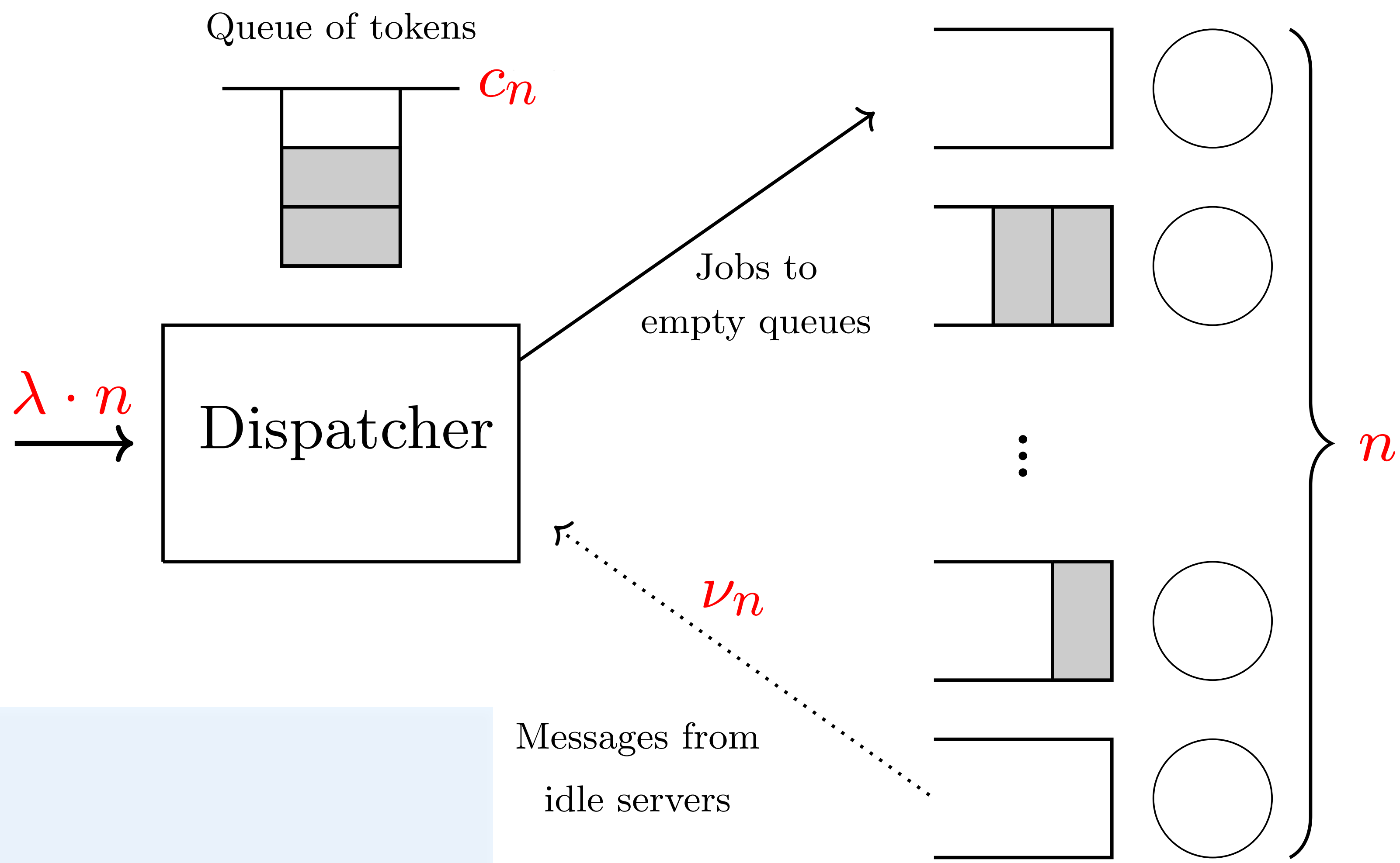


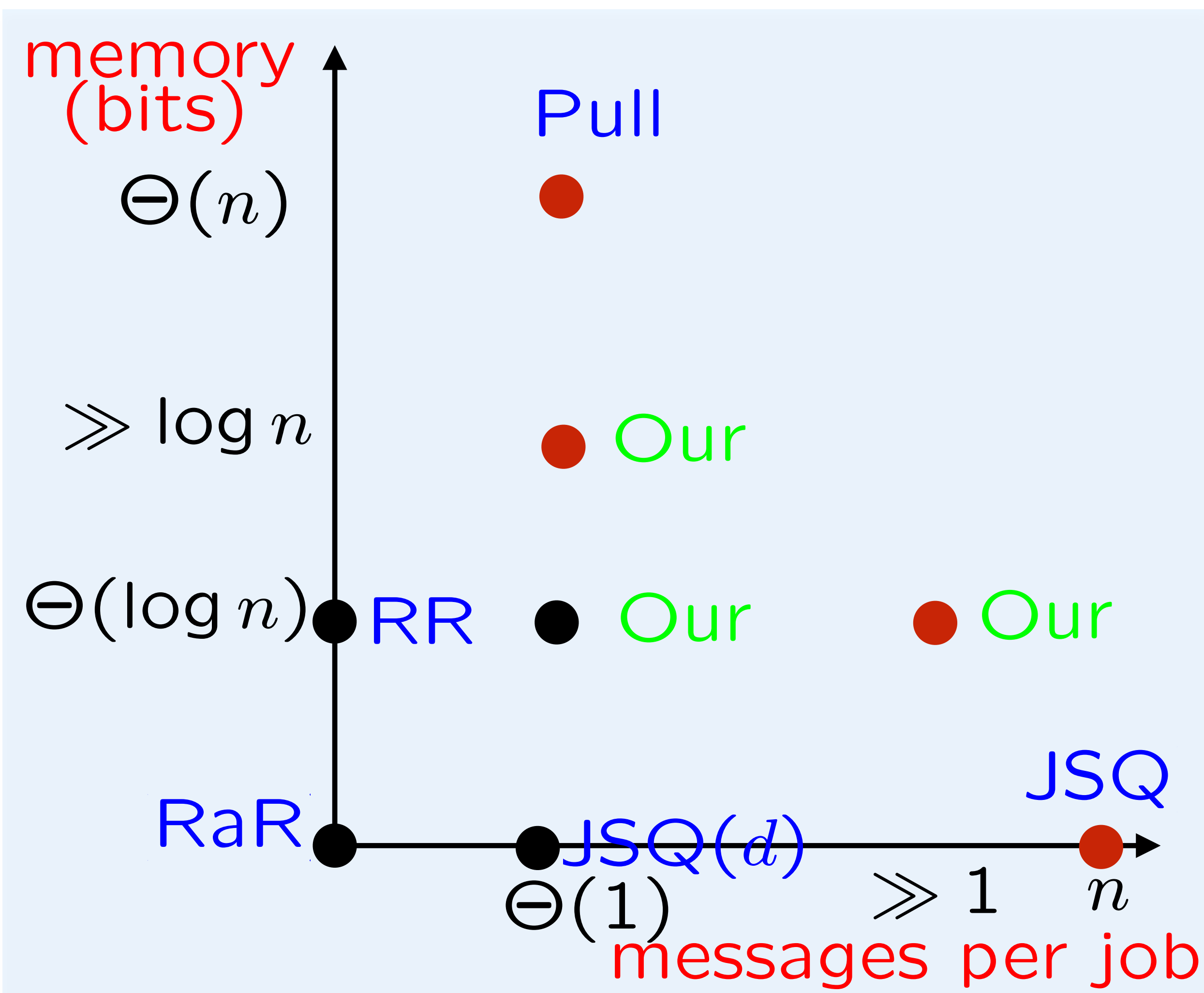


- Expressions for delay via fluid models
- Bounded delay as $\lambda \to 1$
- Better than JSQ(d)

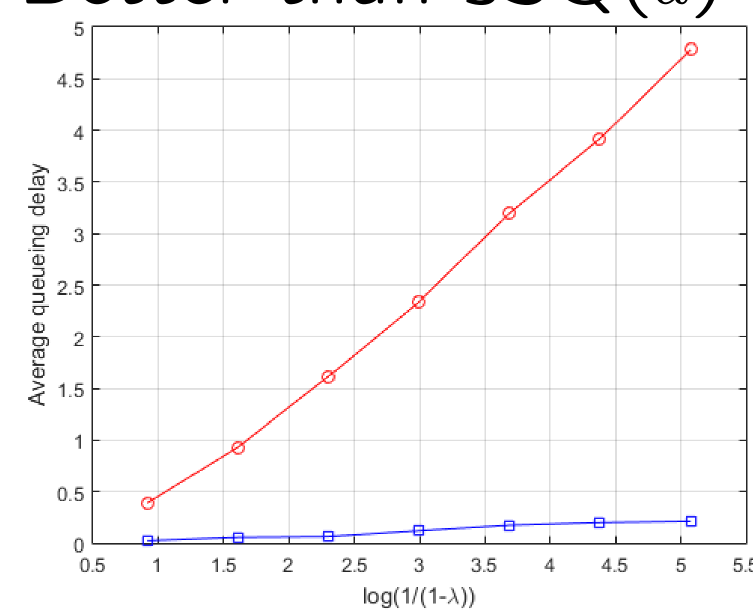
Parsimonious regime

- $\nu_n =$ constant
- $c_n =$ constant
- queueing delay $\not\to 0$

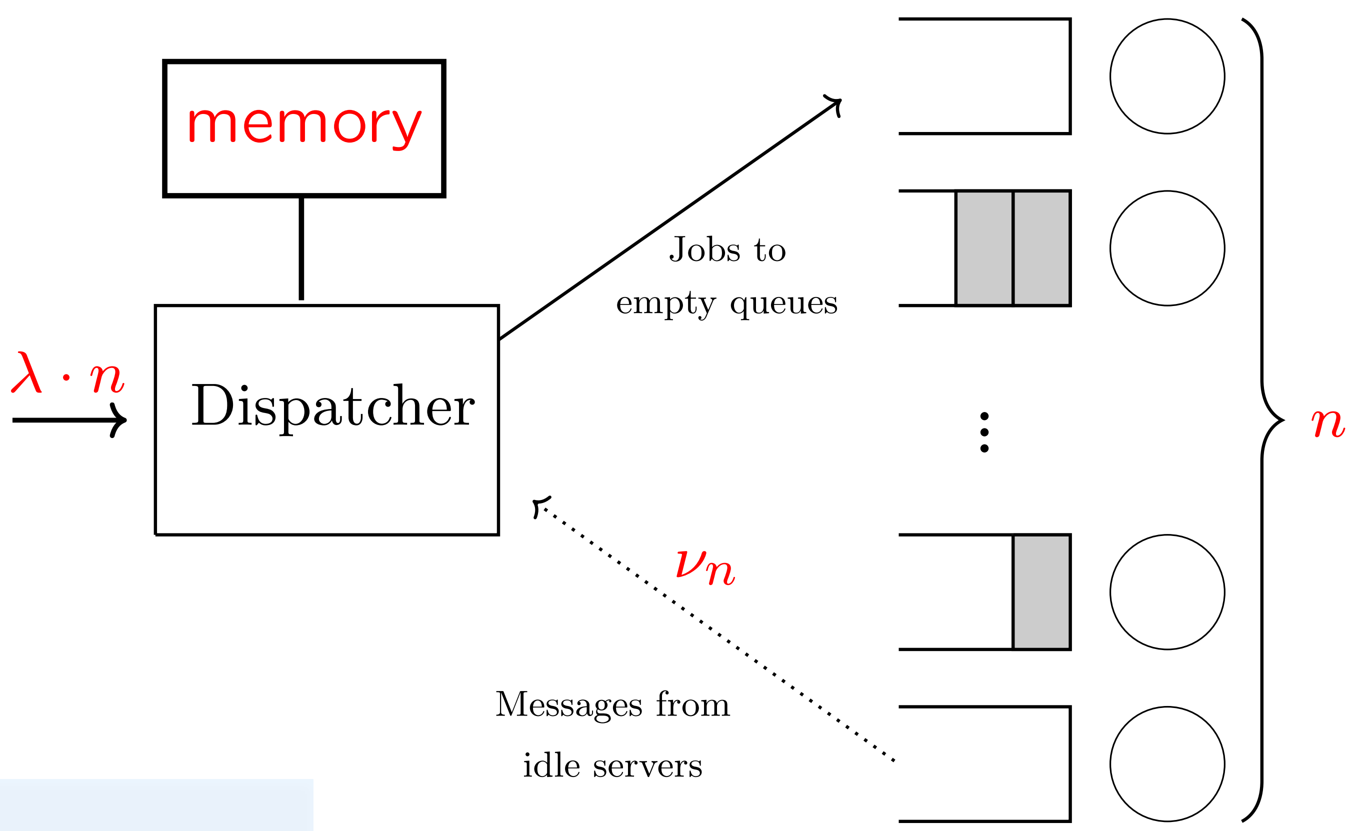


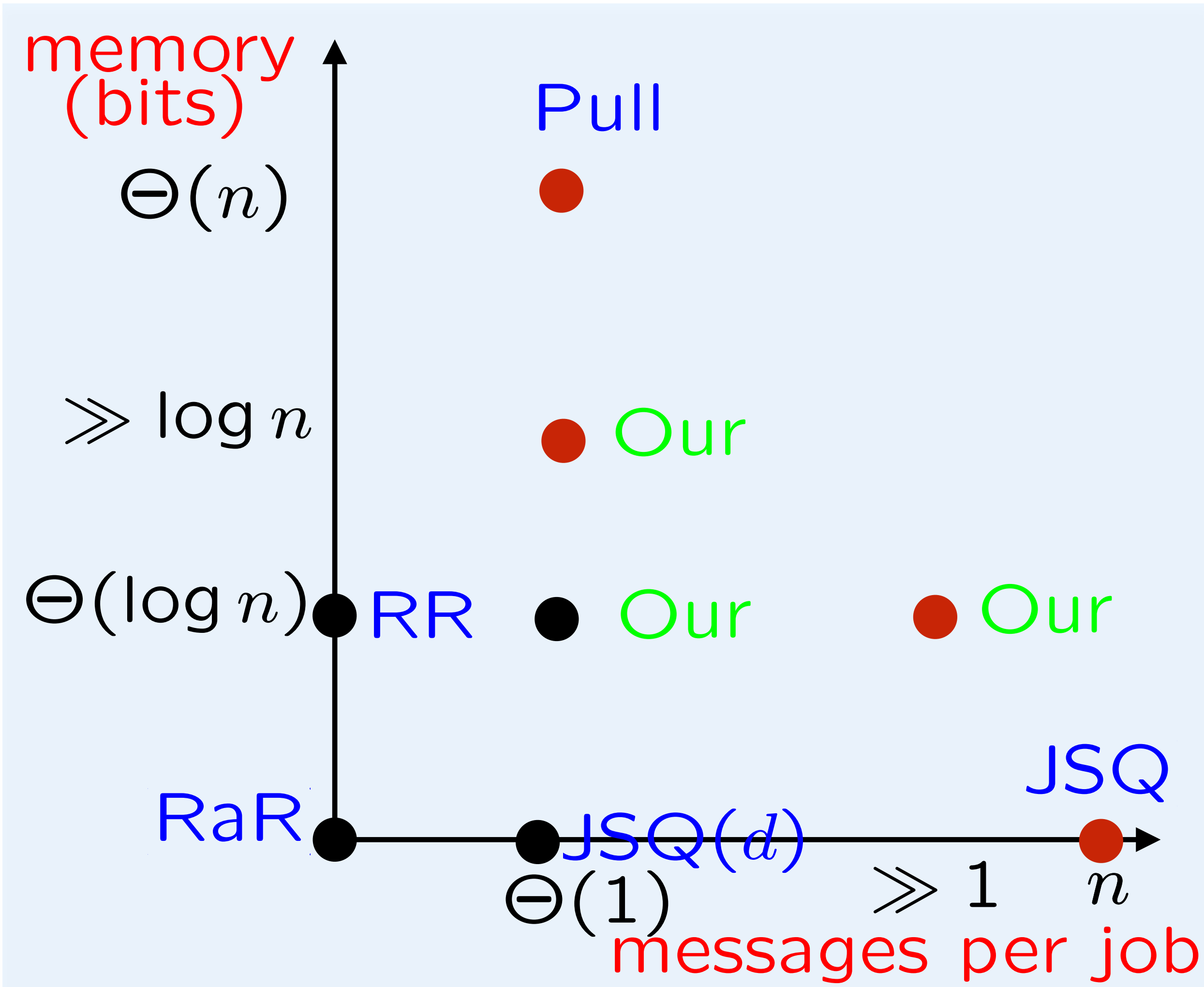


- Expressions for delay via fluid models
- Bounded delay as $\lambda \to 1$
- Better than JSQ(d)



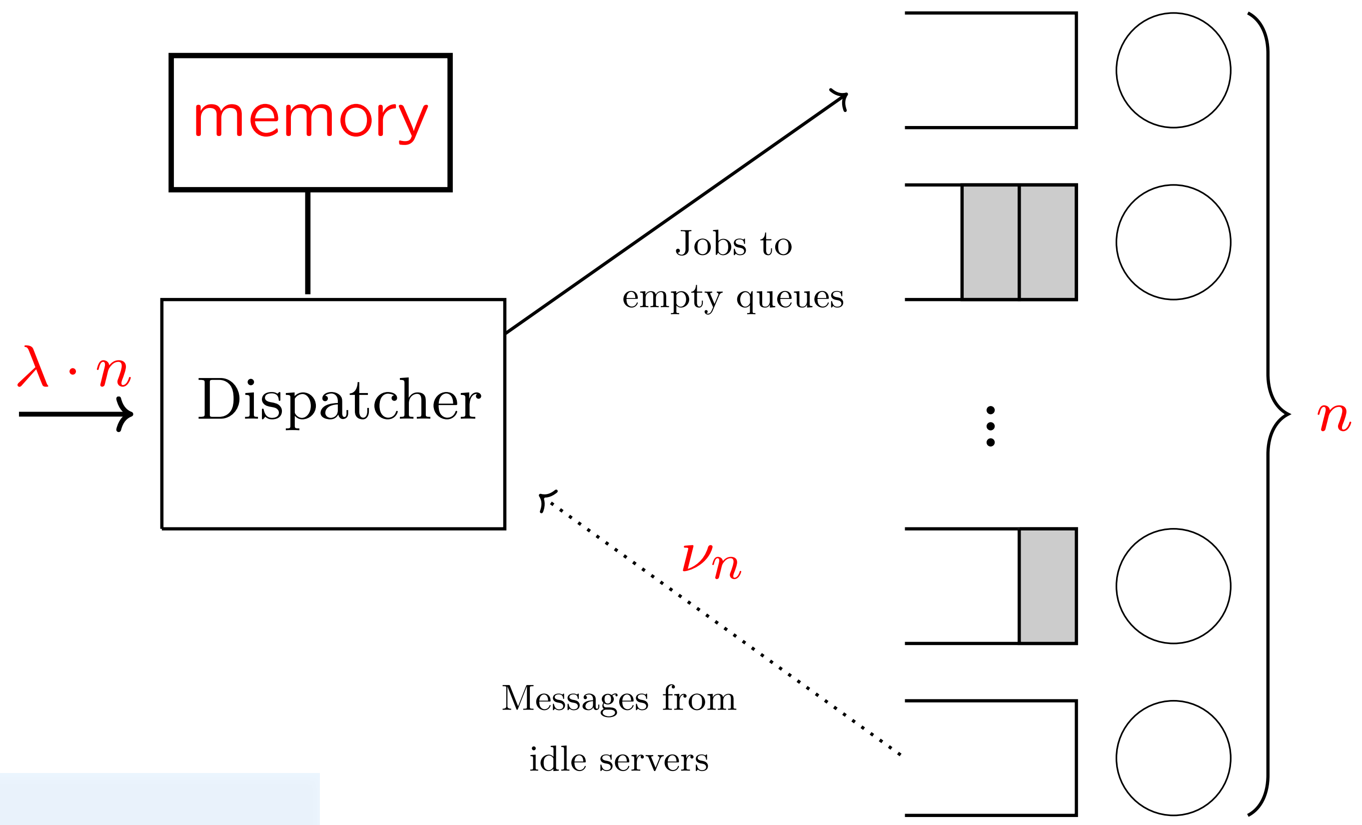
Cannot do better

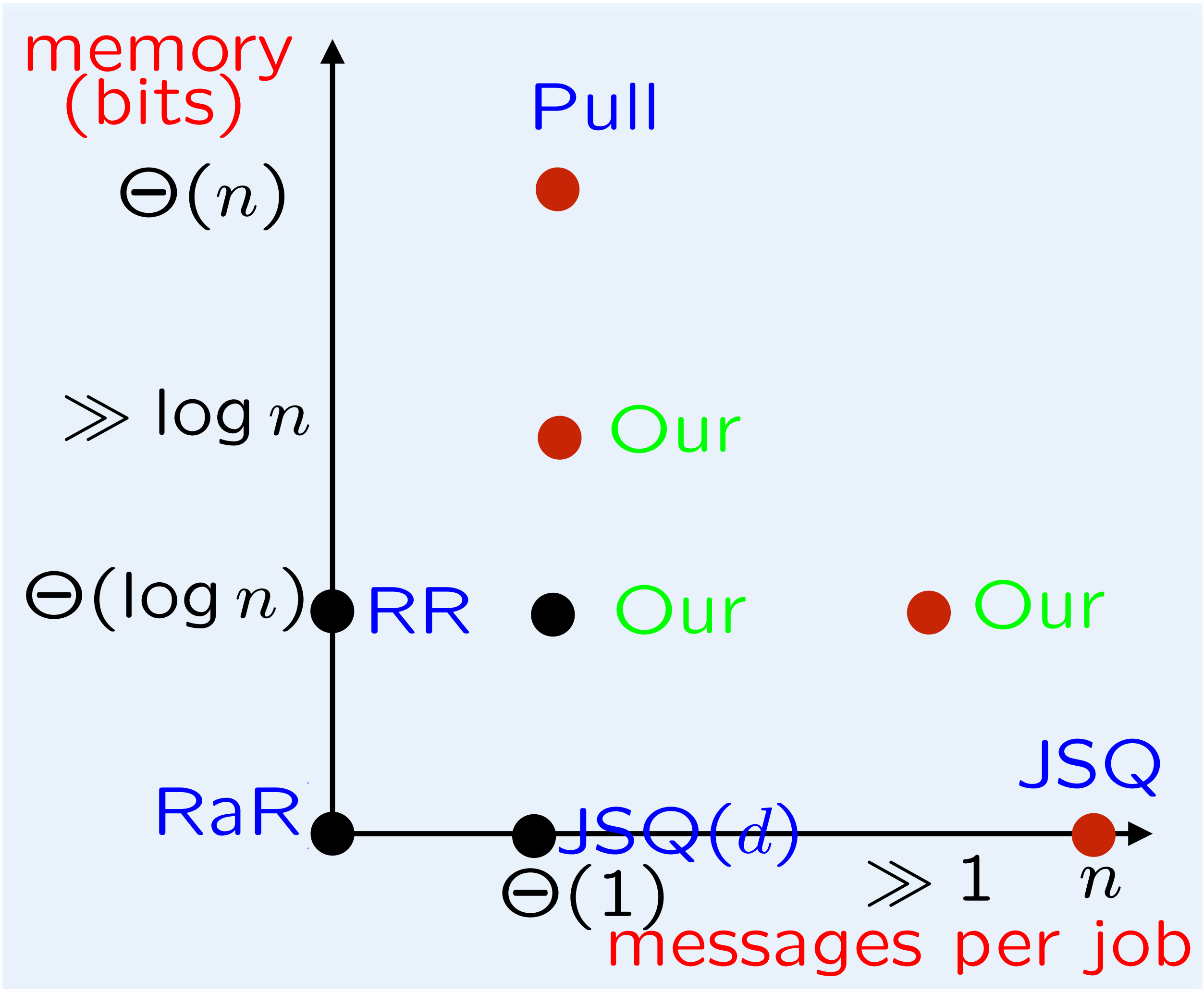




Cannot do better

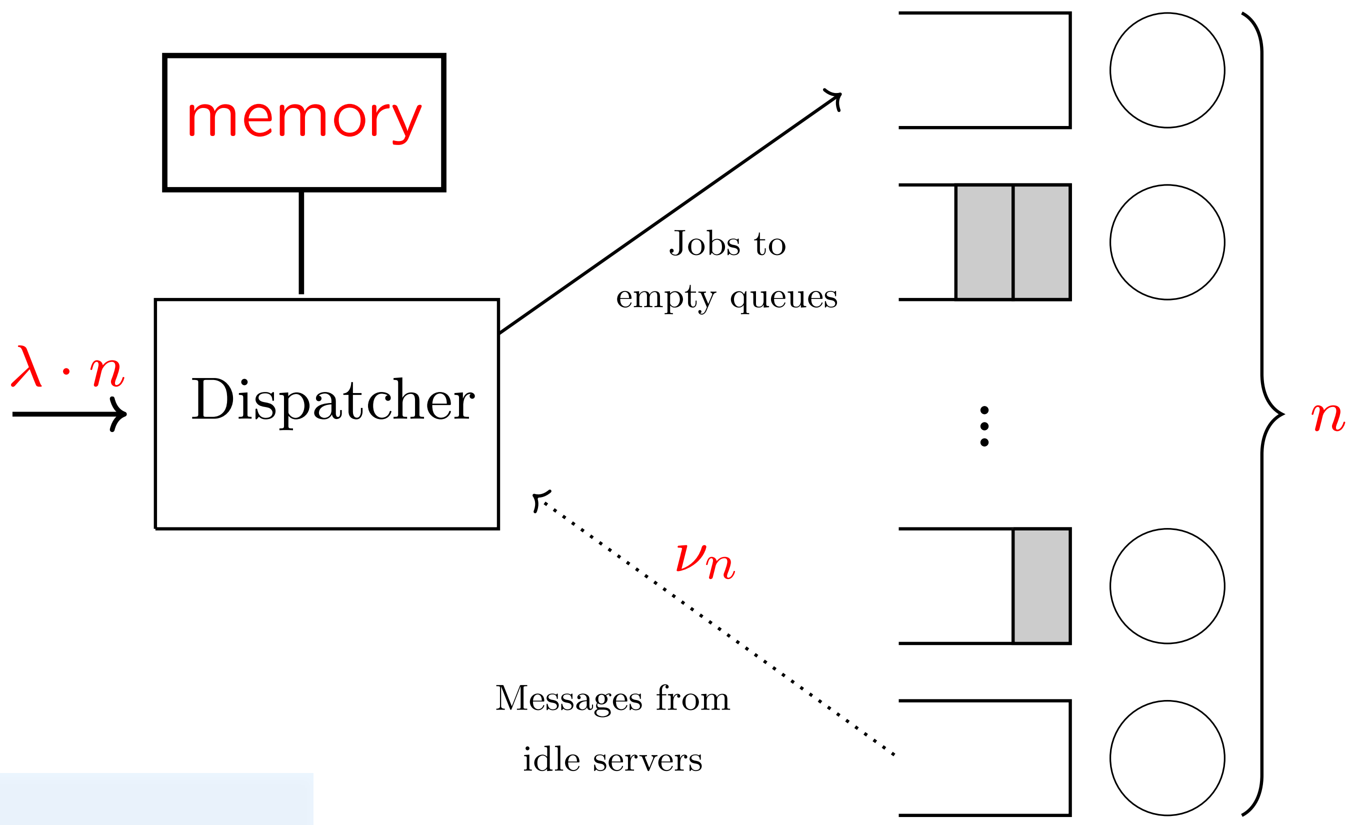
- $\nu_n =$ constant
- $\Theta(\log n)$ memory

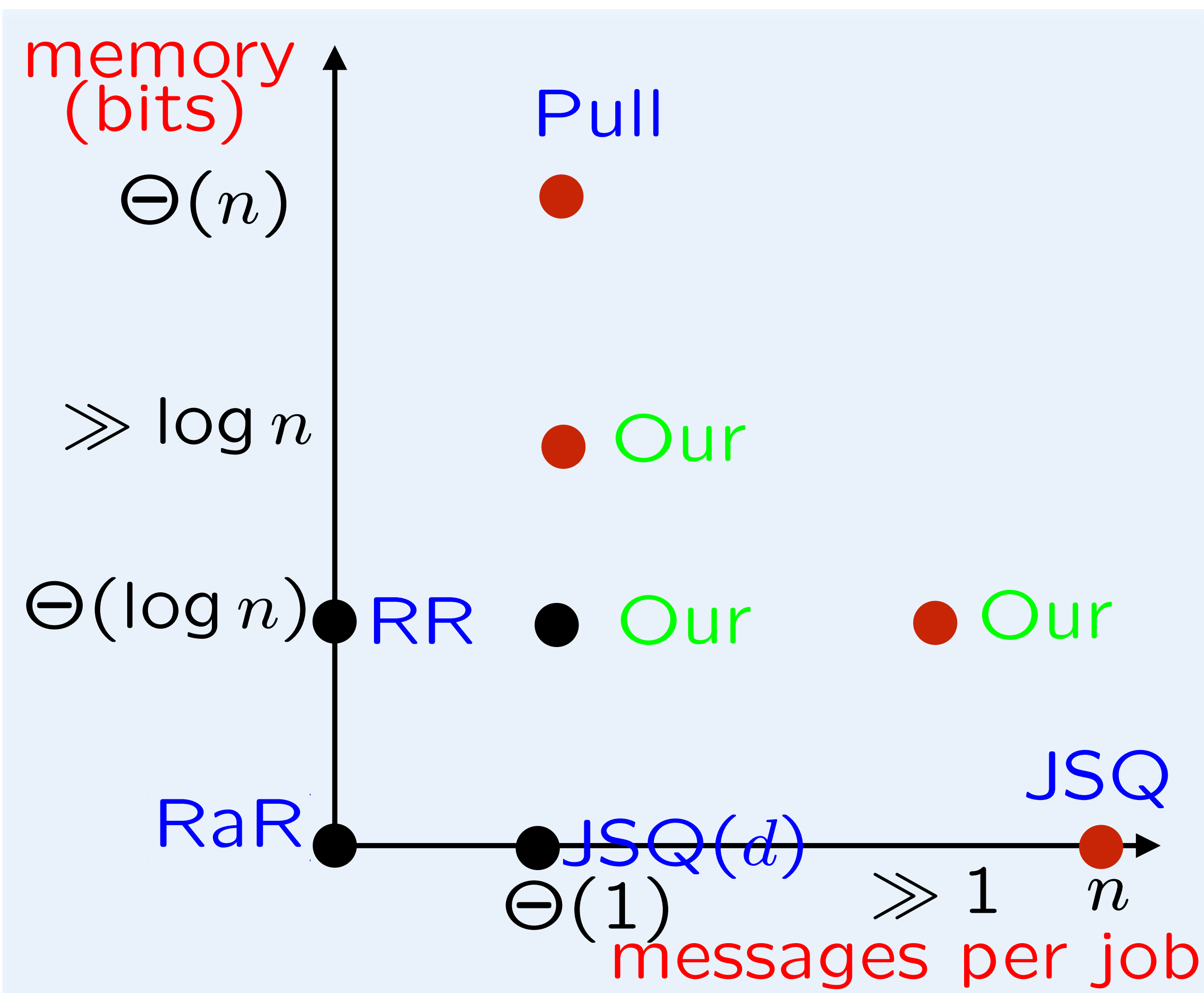




Cannot do better

- $\nu_n =$ constant
- $\Theta(\log n)$ memory

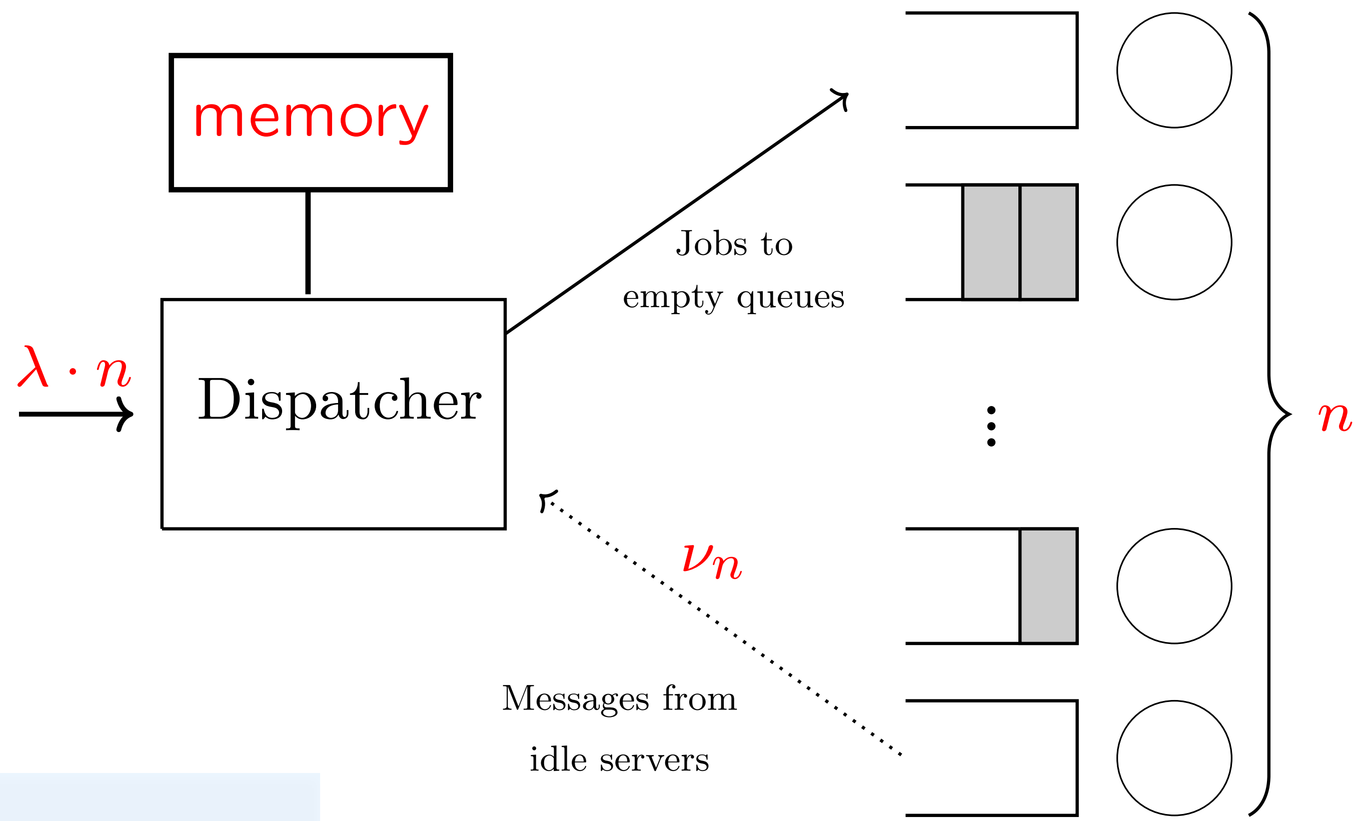


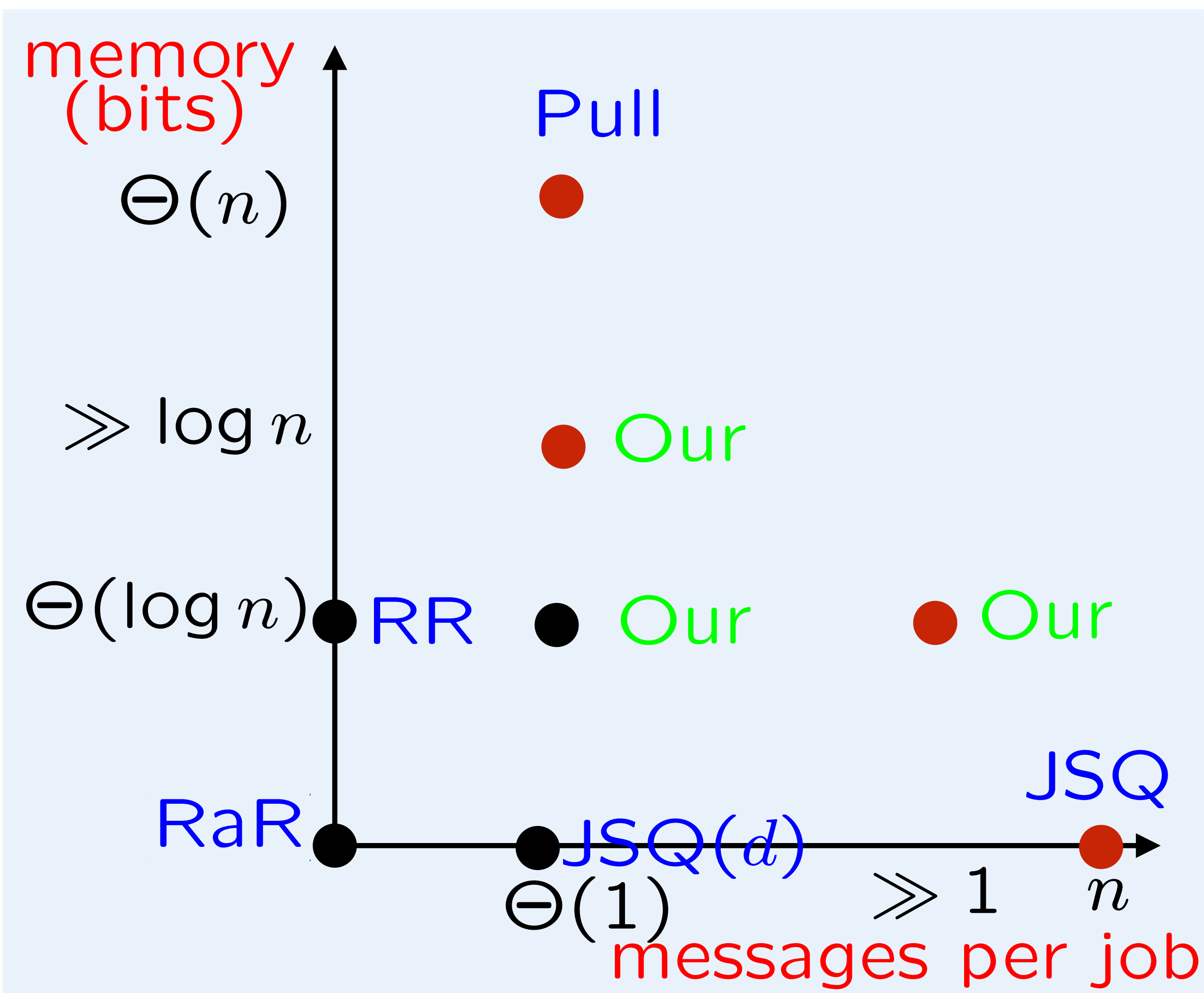


- **Thm:** queueing delay $\not\to 0$

Cannot do better

- $\nu_n =$ constant
- $\Theta(\log n)$ memory

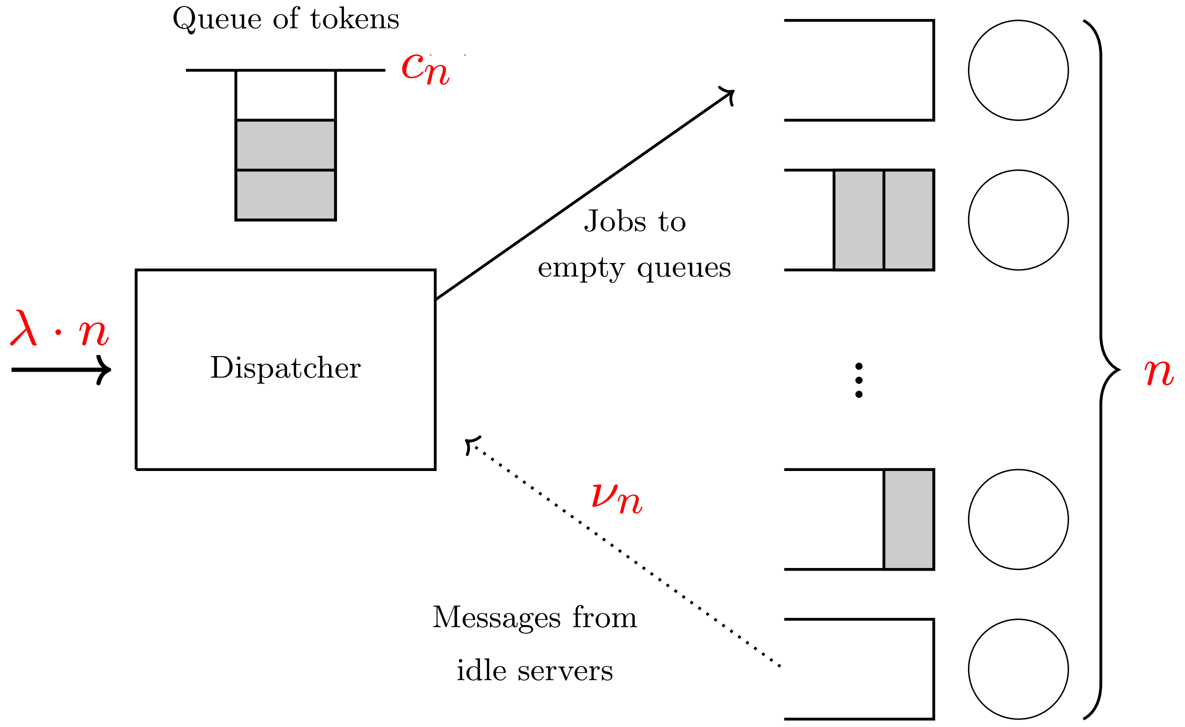




- **Thm:** queueing delay $\not\to 0$
- Assumptions:
  - no queueing at dispatcher
  - "symmetric" policy
  - not too many back-and-forths in too little time

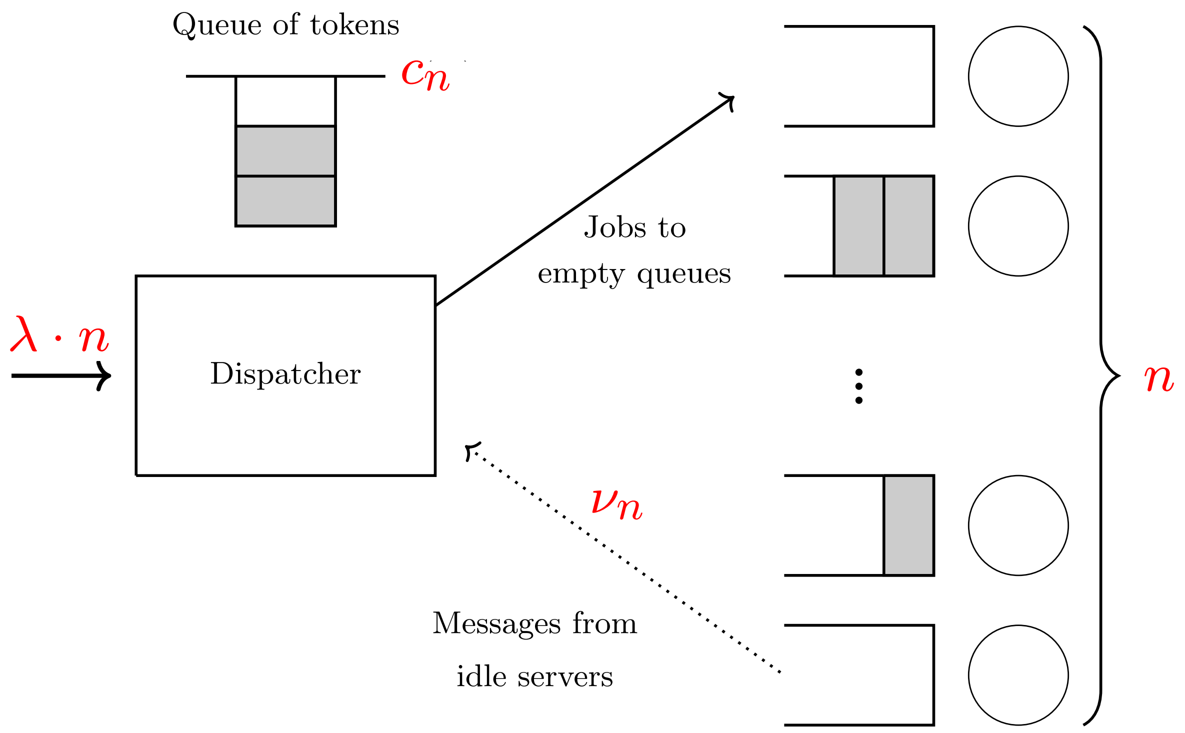
The technical side

Two time scale dynamics



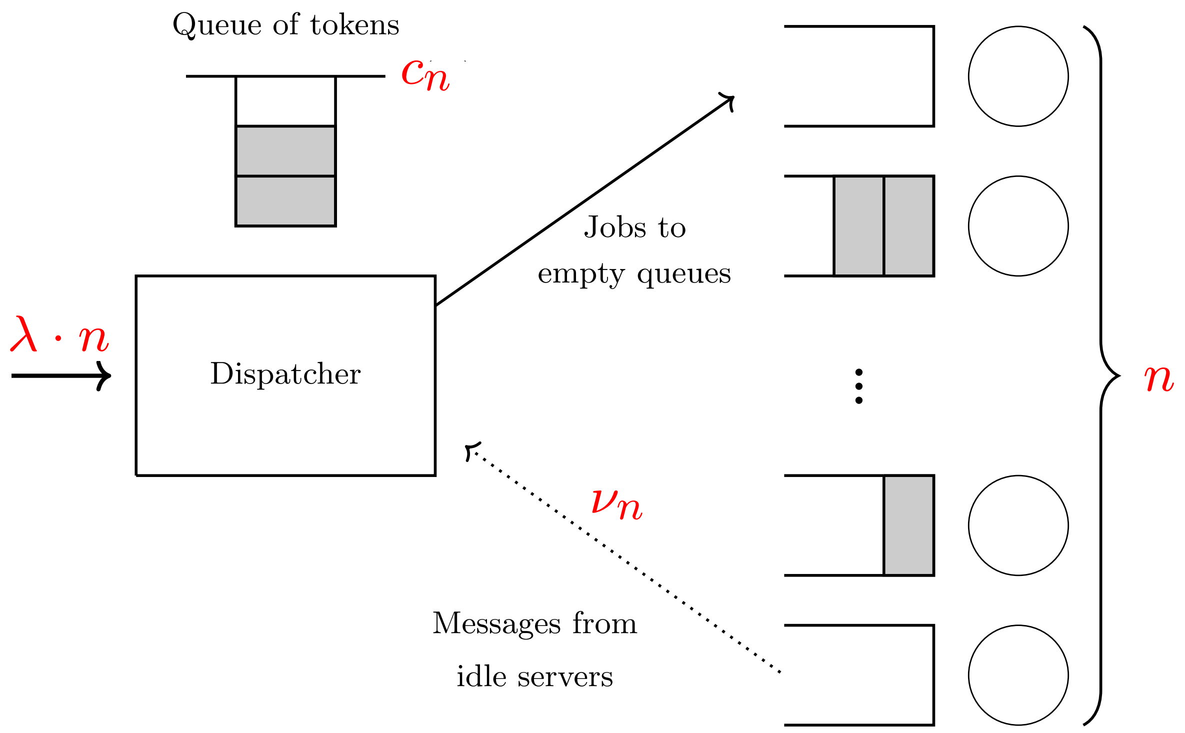
Two time scale dynamics

- $S_i^n(t)$: fraction of servers with at least i jobs (in queue or in service)



Two time scale dynamics

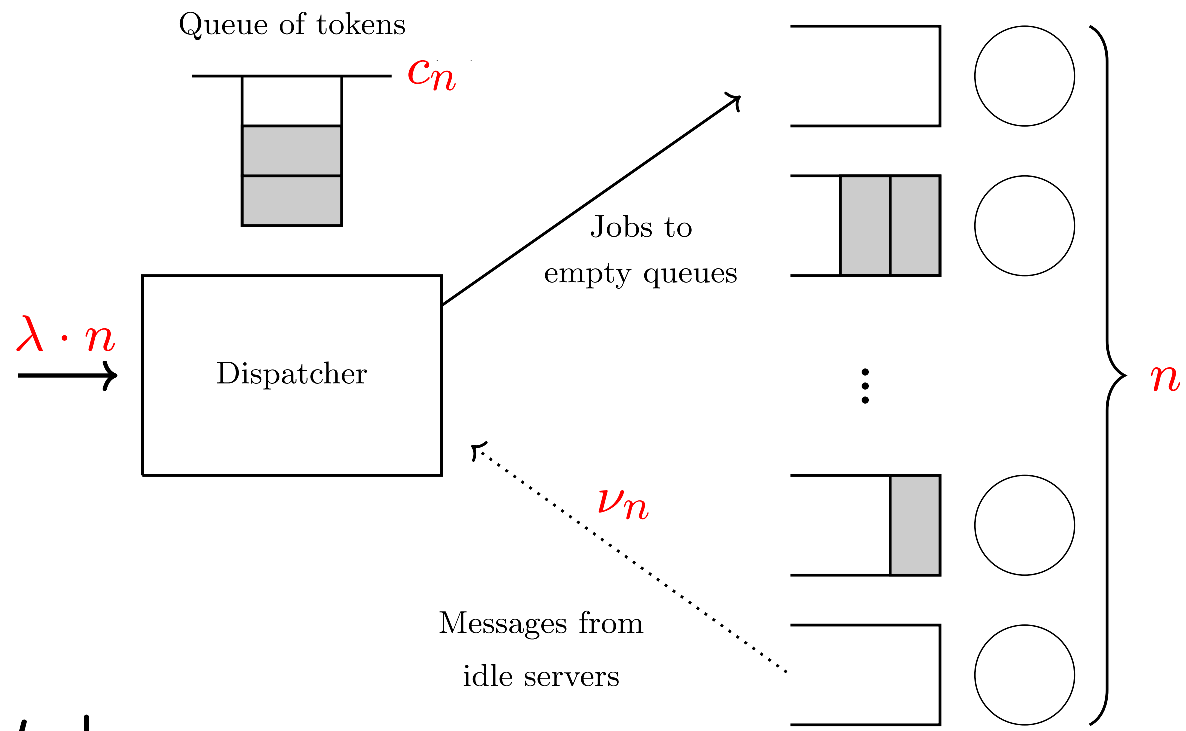
- $S_i^n(t)$: fraction of servers with at least i jobs (in queue or in service)
  - (∞-dimensional) state: $S^n(t)$



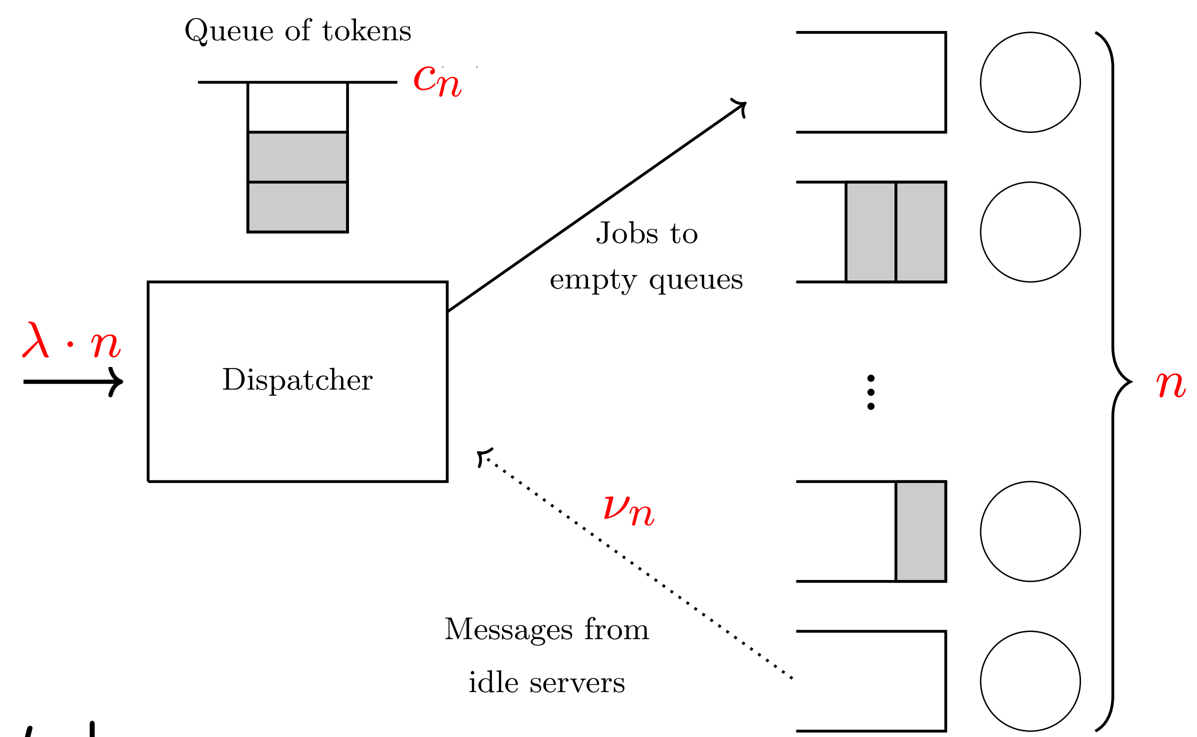
Two time scale dynamics

- $S_i^n(t)$: fraction of servers with at least i jobs (in queue or in service)
  - (∞-dimensional) state: $S^n(t)$

- $S^n(t) = s \ \Rightarrow \ S^n(\tau) \approx s$ for $t \leq \tau \leq t + \epsilon$



Two time scale dynamics

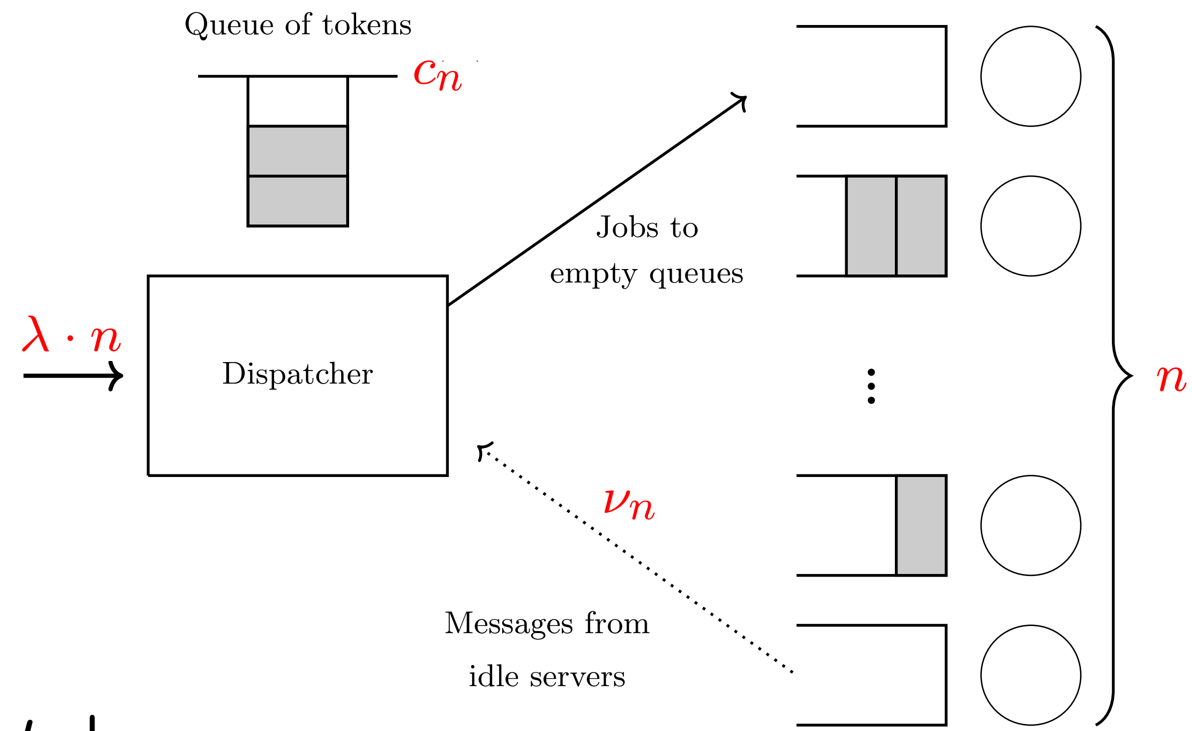


- $S_i^n(t)$: fraction of servers with at least i jobs (in queue or in service)
  - (∞-dimensional) state: $S^n(t)$

- $S^n(t) = s \;\Rightarrow\; S^n(\tau) \approx s$ for $t \leq \tau \leq t + \epsilon$

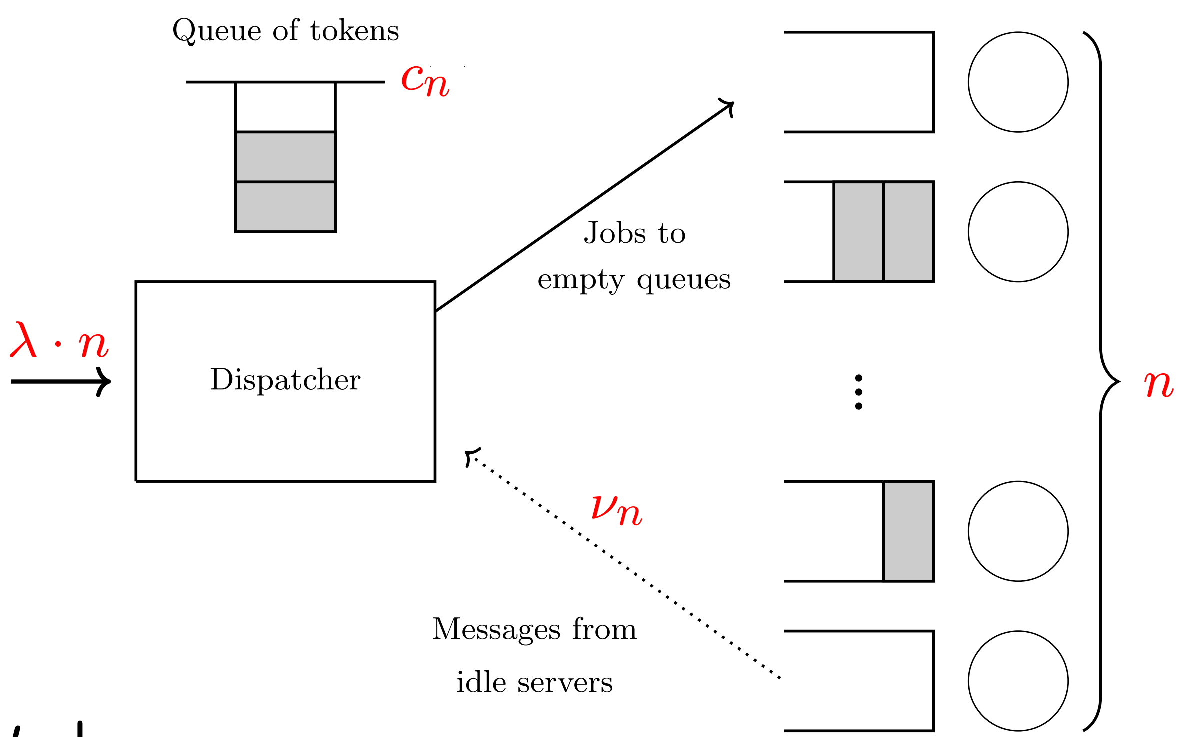
- During ϵ time interval have $\Omega(n)$ arrivals/departures

Two time scale dynamics



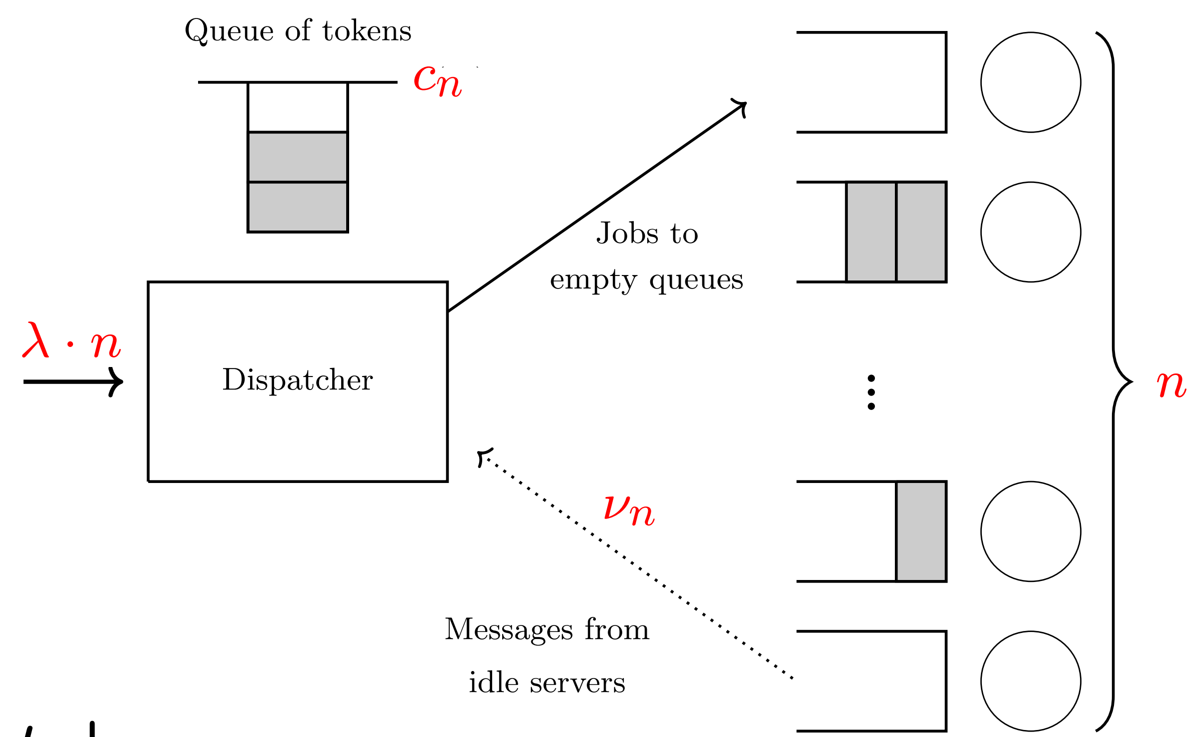
- $S_i^n(t)$: fraction of servers with at least i jobs (in queue or in service)
  - (∞-dimensional) state: $S^n(t)$
- $S^n(t) = s \;\Rightarrow\; S^n(\tau) \approx s$ for $t \leq \tau \leq t + \epsilon$
- During ϵ time interval have $\Omega(n)$ arrivals/departures
- With bounded token queue, enough time for it to reach steady state

Two time scale dynamics



- $S_i^n(t)$: fraction of servers with at least i jobs (in queue or in service)
  - (∞-dimensional) state: $S^n(t)$
- $S^n(t) = s \;\Rightarrow\; S^n(\tau) \approx s$ for $t \leq \tau \leq t + \epsilon$
- During ϵ time interval have $\Omega(n)$ arrivals/departures
- With bounded token queue, enough time for it to reach steady state
- $P_0(s)$: steady-state prob(empty)

Two time scale dynamics



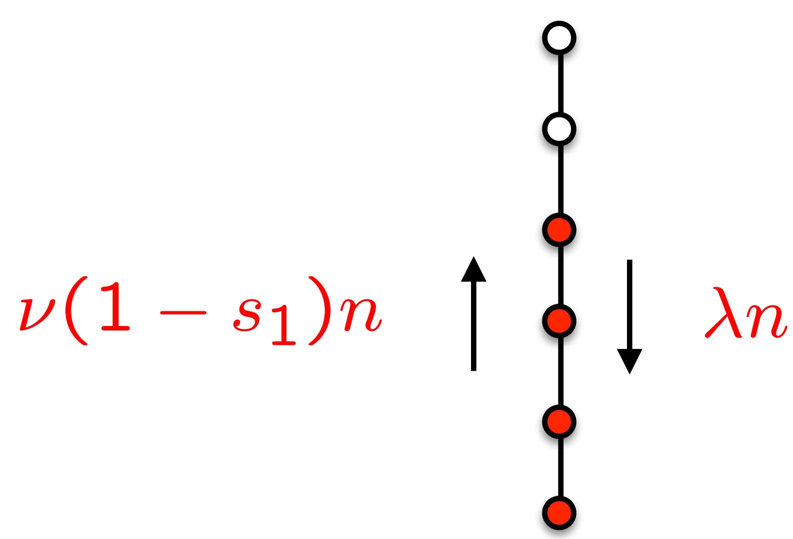
- $S_i^n(t)$: fraction of servers with at least i jobs (in queue or in service)
  - (∞-dimensional) state: $S^n(t)$

- $S^n(t) = s \;\Rightarrow\; S^n(\tau) \approx s$ for $t \leq \tau \leq t + \epsilon$

- During ϵ time interval have $\Omega(n)$ arrivals/departures

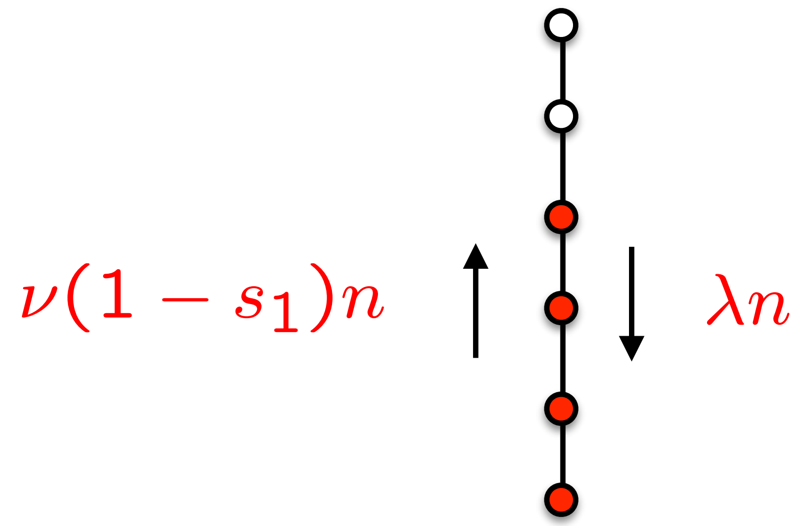
- With bounded token queue, enough time for it to reach steady state

- $P_0(s)$: steady-state prob(empty)



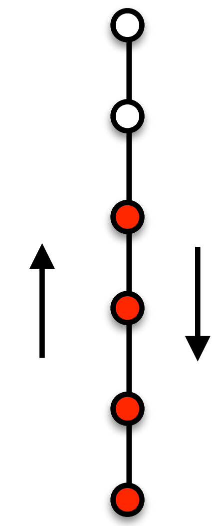
The “easy” cases

- $P_0(s)$: steady-state prob(empty)



The “easy” cases

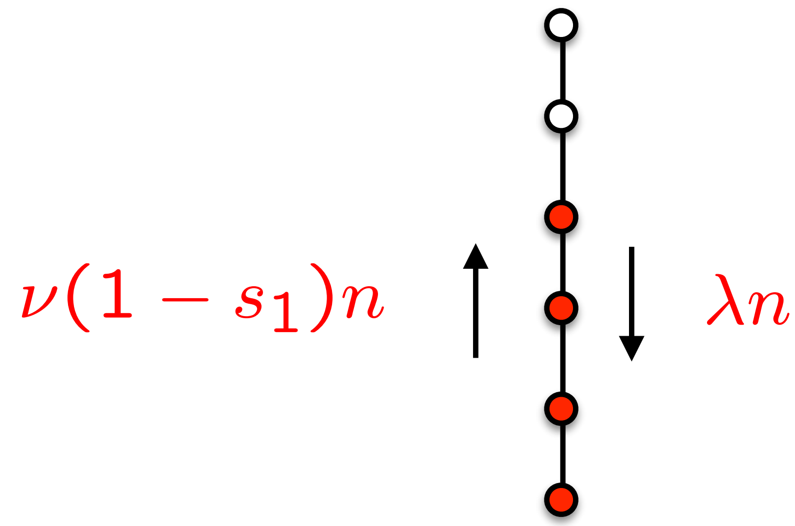
- $P_0(s)$: steady-state prob(empty)



- High message rate regime: $\nu \to \infty$: $P_0(s) \to 0$
  $\Rightarrow$ delay $\to 0$

The “easy” cases

- $P_0(s)$: steady-state prob(empty)

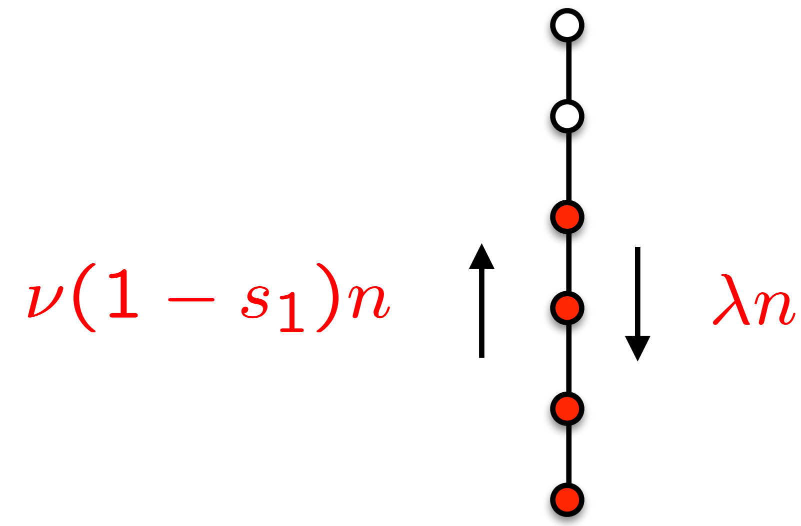


- High message rate regime: $\nu \to \infty$: $\quad P_0(s) \to 0$
  $\Rightarrow$ delay $\to 0$

- High memory regime: memory $\to \infty$

The "easy" cases

- $P_0(s)$: steady-state prob(empty)

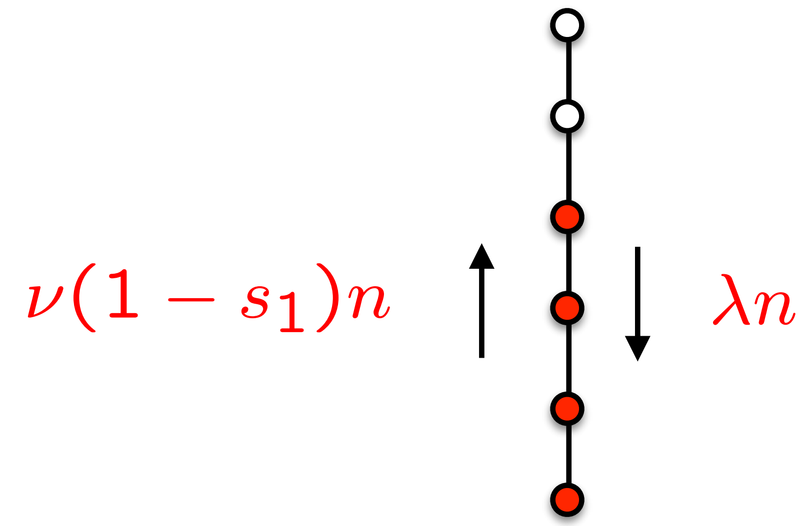


- High message rate regime: $\nu \to \infty$: $P_0(s) \to 0$
  $\Rightarrow$ delay $\to 0$

- High memory regime: memory $\to \infty$
  - $P_0(s) \to 0$, as long as drift is upwards

The "easy" cases

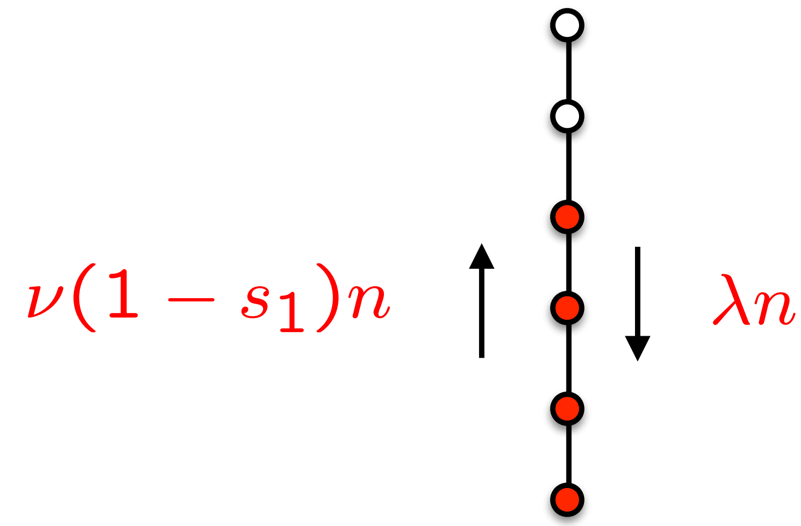
- $P_0(s)$: steady-state prob(empty)



- High message rate regime: $\nu \to \infty$: $P_0(s) \to 0$
  $\Rightarrow$ delay $\to 0$

- High memory regime: memory $\to \infty$
  - $P_0(s) \to 0$, as long as drift is upwards
  - In steady state: $s_1 = \lambda$ (Little's law)

The “easy” cases



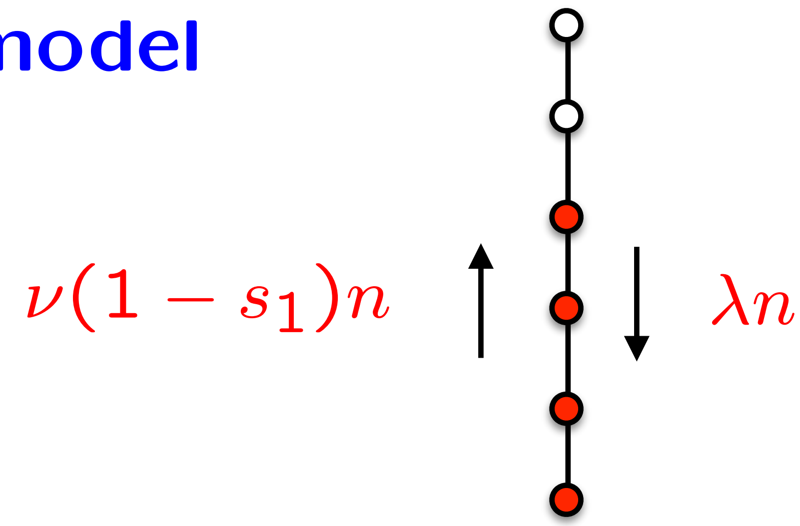
- $P_0(s)$: steady-state prob(empty)

- High message rate regime: $\nu \to \infty$: $\quad P_0(s) \to 0$
  $\Rightarrow$ delay $\to 0$

- High memory regime: memory $\to \infty$
  - $P_0(s) \to 0$, as long as drift is upwards
  - In steady state: $s_1 = \lambda$ (Little’s law)
  - delay $\to 0$ iff $\nu(1 - \lambda) \geq \lambda$

The resource constrained case - fluid model

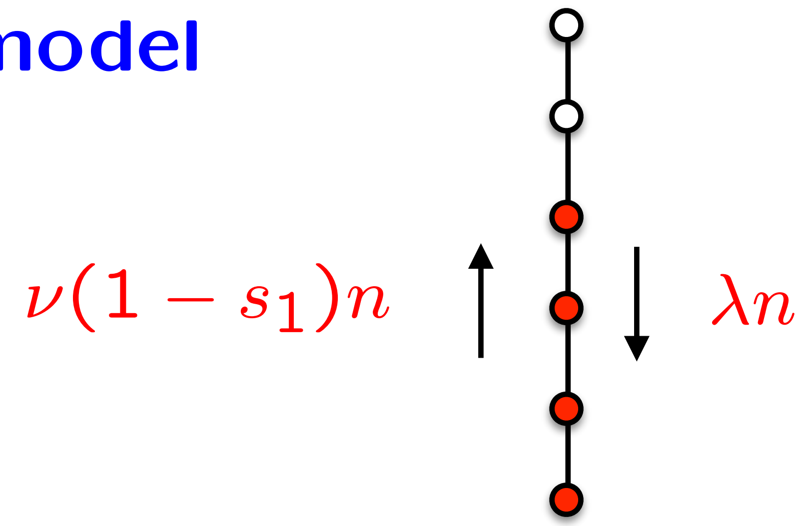
$$P_0(s) = \left[\sum_{k=0}^{C} \left(\frac{\nu(1 - s_1)}{\lambda} \right)^k \right]^{-1}$$



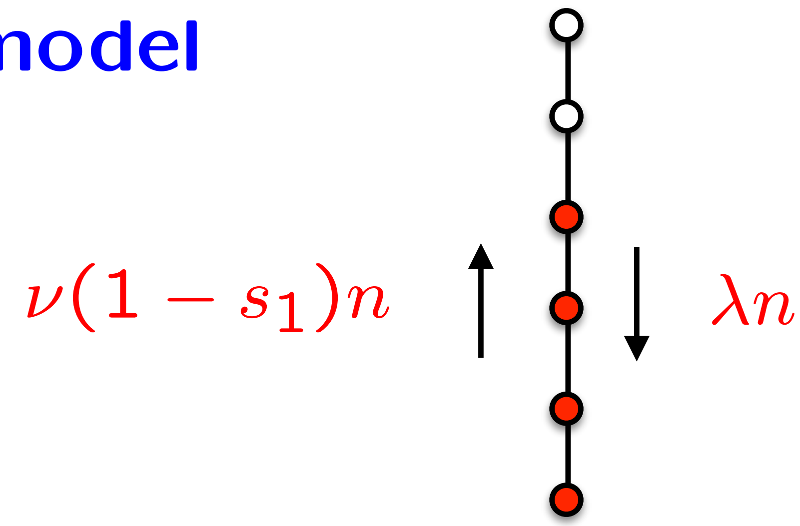
The resource constrained case - fluid model

$$P_0(s) = \left[\sum_{k=0}^{C} \left(\frac{\nu(1 - s_1)}{\lambda} \right)^k \right]^{-1}$$

Lipschitz continuous in s



The resource constrained case - fluid model

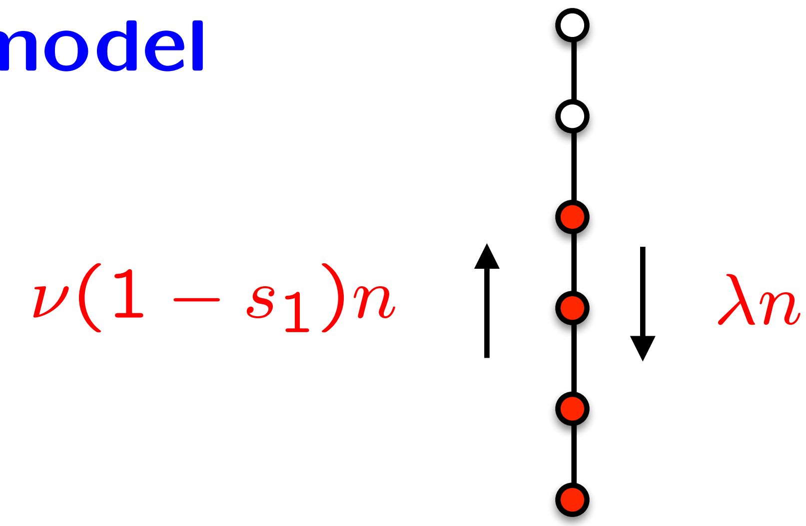


$$P_0(s) = \left[\sum_{k=0}^{C} \left(\frac{\nu(1-s_1)}{\lambda} \right)^k \right]^{-1}$$

Lipschitz continuous in s

- $S^n(t)$ evolves in a slower time scale

The resource constrained case - fluid model

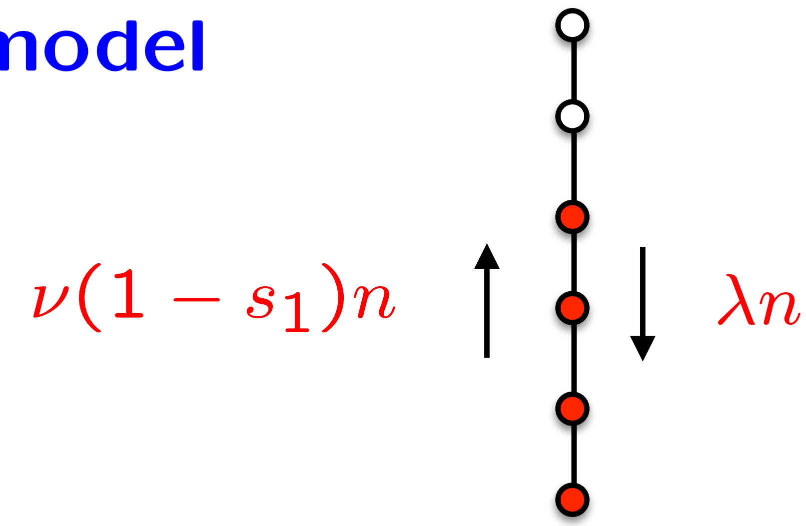


$$P_0(s) = \left[\sum_{k=0}^{C} \left(\frac{\nu(1-s_1)}{\lambda}\right)^k\right]^{-1}$$

Lipschitz continuous in s

- $S^n(t)$ evolves in a slower time scale
- during $[t, t+\epsilon]$, fraction $P_0(s)$ of arriving jobs get routed randomly

The resource constrained case - fluid model



$$P_0(s) = \left[\sum_{k=0}^{C} \left(\frac{\nu(1-s_1)}{\lambda} \right)^k \right]^{-1}$$

Lipschitz continuous in s

- $S^n(t)$ evolves in a slower time scale
- during $[t, t+\epsilon]$, fraction $P_0(s)$ of arriving jobs get routed randomly
- $\frac{ds_1}{dt}(t) = \lambda[1 - P_0(t)] + \lambda[1 - s_1(t)]P_0(t) - [s_1(t) - s_2(t)]$
- $\frac{ds_i}{dt}(t) = \lambda[s_{i-1}(t) - s_i(t)]P_0(t) - [s_i(t) - s_{i+1}(t)], \quad i \geq 2$

The theorems

The theorems

- The fluid equations have a solution

  (indirect proof: stochastic system trajectories have limit points, and these satisfy fluid equations)

The theorems

- The fluid equations have a solution

  (indirect proof: stochastic system trajectories have limit points, and these satisfy fluid equations)

- Unique solution

  (resource constrained case: from Lipschitz continuity of r.h.s.
  other cases: more delicate, because $P_0(s)$ is discontinuous)

The theorems

- The fluid equations have a solution

  (indirect proof: stochastic system trajectories have limit points, and these satisfy fluid equations)

- Unique solution

  (resource constrained case: from Lipschitz continuity of r.h.s. other cases: more delicate, because $P_0(s)$ is discontinuous)

- Unique equilibrium point s^* (algebra) which is asymptotically stable for all (interesting) initial conditions (sandwich between tractable solutions)

The theorems (ctd.)

$$
\begin{tikzcd}
S^n(t) \arrow[r, "n \to \infty"] \arrow[d, "t \to \infty"'] & s(t) \arrow[d, "t \to \infty"] \\
\pi^n \arrow[r, "n \to \infty"'] & s^*
\end{tikzcd}
$$

- standard “proof technology”

The theorems (ctd.)

$$
\begin{tikzcd}
S^n(t) \arrow[r, "n \to \infty"] \arrow[d, "t \to \infty"'] & s(t) \arrow[d, "t \to \infty"] \\
\pi^n \arrow[r, "n \to \infty"'] & s^*
\end{tikzcd}
$$

- standard “proof technology”
- $\mathbf{E}_{\pi_n}$[delay] $\to \mathbf{E}_{s^*}$[delay] $= \sum_{i=1}^{\infty} s_i^*$

Delay analysis (resource constrained case)

Delay analysis (resource constrained case)

- $s_1^* = \lambda$

Delay analysis (resource constrained case)

- $s_1^* = \lambda$
- $P_0^* = P_0(s^*) = \left[\sum_{k=0}^{C} \left(\frac{\nu(1-\lambda)}{\lambda} \right)^k \right]^{-1}$

Delay analysis (resource constrained case)

- $s_1^* = \lambda$
- $P_0^* = P_0(s^*) = \left[\sum_{k=0}^{C} \left(\frac{\nu(1-\lambda)}{\lambda} \right)^k \right]^{-1}$
   - $\frac{ds_1}{dt}(t) = \lambda[1 - P_0(t)] + \lambda[1 - s_1(t)]P_0(t) - [s_1(t) - s_2(t)]$
   - $\frac{ds_i}{dt}(t) = \lambda[s_{i-1}(t) - s_i(t)]P_0(t) - [s_i(t) - s_{i+1}(t)], \quad i \geq 2$

Delay analysis (resource constrained case)

- $s_1^* = \lambda$
- $P_0^* = P_0(s^*) = \left[\sum_{k=0}^{C} \left(\frac{\nu(1-\lambda)}{\lambda}\right)^k\right]^{-1}$
  - $\frac{ds_1}{dt}(t) = \lambda[1 - P_0(t)] + \lambda[1 - s_1(t)]P_0(t) - [s_1(t) - s_2(t)]$
  - $\frac{ds_i}{dt}(t) = \lambda[s_{i-1}(t) - s_i(t)]P_0(t) - [s_i(t) - s_{i+1}(t)], \quad i \geq 2$
- $s_i^* = \lambda(\lambda P_0^*)^{i-1}$
- $\mathbf{E}[\text{delay}] = \frac{\lambda P_0^*}{1 - \lambda P_0^*}$

Delay analysis (resource constrained case)

- $s_1^* = \lambda$
- $P_0^* = P_0(s^*) = \left[\sum_{k=0}^{C} \left(\frac{\nu(1-\lambda)}{\lambda}\right)^k\right]^{-1}$
  - $\frac{ds_1}{dt}(t) = \lambda[1 - P_0(t)] + \lambda[1 - s_1(t)]P_0(t) - [s_1(t) - s_2(t)]$
  - $\frac{ds_i}{dt}(t) = \lambda[s_{i-1}(t) - s_i(t)]P_0(t) - [s_i(t) - s_{i+1}(t)], \quad i \geq 2$
- $s_i^* = \lambda(\lambda P_0^*)^{i-1}$
- $\mathbf{E}[\text{delay}] = \frac{\lambda P_0^*}{1 - \lambda P_0^*}$

Heavy-traffic analysis, $\lambda \uparrow 1$

Delay analysis (resource constrained case)

- $s_1^* = \lambda$
- $P_0^* = P_0(s^*) = \left[\sum_{k=0}^{C} \left(\frac{\nu(1-\lambda)}{\lambda}\right)^k\right]^{-1}$
  - $\frac{ds_1}{dt}(t) = \lambda[1 - P_0(t)] + \lambda[1 - s_1(t)]P_0(t) - [s_1(t) - s_2(t)]$
  - $\frac{ds_i}{dt}(t) = \lambda[s_{i-1}(t) - s_i(t)]P_0(t) - [s_i(t) - s_{i+1}(t)], \quad i \geq 2$
- $s_i^* = \lambda(\lambda P_0^*)^{i-1}$
- $\mathbf{E}[\text{delay}] = \frac{\lambda P_0^*}{1 - \lambda P_0^*}$

Heavy-traffic analysis, $\lambda \uparrow 1$

- Choose ν so that $\nu(1-\lambda) = \lambda d$

Delay analysis (resource constrained case)

- $s_1^* = \lambda$
- $P_0^* = P_0(s^*) = \left[\sum_{k=0}^{C} \left(\frac{\nu(1-\lambda)}{\lambda}\right)^k\right]^{-1}$
  - $\frac{ds_1}{dt}(t) = \lambda[1 - P_0(t)] + \lambda[1 - s_1(t)]P_0(t) - [s_1(t) - s_2(t)]$
  - $\frac{ds_i}{dt}(t) = \lambda[s_{i-1}(t) - s_i(t)]P_0(t) - [s_i(t) - s_{i+1}(t)], \quad i \geq 2$
- $s_i^* = \lambda(\lambda P_0^*)^{i-1}$
- $\mathbf{E}[\text{delay}] = \frac{\lambda P_0^*}{1 - \lambda P_0^*}$

Heavy-traffic analysis, $\lambda \uparrow 1$

- Choose ν so that $\nu(1-\lambda) = \lambda d$
  - Messages per unit time: λdn

Delay analysis (resource constrained case)

- $s_1^* = \lambda$
- $P_0^* = P_0(s^*) = \left[\sum_{k=0}^{C} \left(\frac{\nu(1-\lambda)}{\lambda}\right)^k\right]^{-1}$
  - $\frac{ds_1}{dt}(t) = \lambda[1 - P_0(t)] + \lambda[1 - s_1(t)]P_0(t) - [s_1(t) - s_2(t)]$
  - $\frac{ds_i}{dt}(t) = \lambda[s_{i-1}(t) - s_i(t)]P_0(t) - [s_i(t) - s_{i+1}(t)], \quad i \geq 2$
- $s_i^* = \lambda(\lambda P_0^*)^{i-1}$
- $\mathbf{E}[\text{delay}] = \frac{\lambda P_0^*}{1 - \lambda P_0^*}$

Heavy-traffic analysis, $\lambda \uparrow 1$

- Choose ν so that $\nu(1-\lambda) = \lambda d$
  - Messages per unit time: $\lambda d n$
  - P_0^* (and therefore, delay): independent of λ

Delay analysis (resource constrained case)

- $s_1^* = \lambda$
- $P_0^* = P_0(s^*) = \left[\sum_{k=0}^{C} \left(\frac{\nu(1-\lambda)}{\lambda}\right)^k\right]^{-1}$
  - $\frac{ds_1}{dt}(t) = \lambda[1 - P_0(t)] + \lambda[1 - s_1(t)]P_0(t) - [s_1(t) - s_2(t)]$
  - $\frac{ds_i}{dt}(t) = \lambda[s_{i-1}(t) - s_i(t)]P_0(t) - [s_i(t) - s_{i+1}(t)], \quad i \geq 2$
- $s_i^* = \lambda(\lambda P_0^*)^{i-1}$
- $\mathbf{E}[\text{delay}] = \frac{\lambda P_0^*}{1 - \lambda P_0^*}$

Heavy-traffic analysis, $\lambda \uparrow 1$

- Choose ν so that $\nu(1-\lambda) = \lambda d$
  - Messages per unit time: λdn
  - P_0^* (and therefore, delay): independent of λ
- Send to shortest of d sampled queues
  - Messages per unit time: λdn

Delay analysis (resource constrained case)

- $s_1^* = \lambda$
- $P_0^* = P_0(s^*) = \left[\sum_{k=0}^{C} \left(\frac{\nu(1-\lambda)}{\lambda}\right)^k\right]^{-1}$
  - $\frac{ds_1}{dt}(t) = \lambda[1 - P_0(t)] + \lambda[1 - s_1(t)]P_0(t) - [s_1(t) - s_2(t)]$
  - $\frac{ds_i}{dt}(t) = \lambda[s_{i-1}(t) - s_i(t)]P_0(t) - [s_i(t) - s_{i+1}(t)], \quad i \geq 2$
- $s_i^* = \lambda(\lambda P_0^*)^{i-1}$
- $\mathbf{E}[\text{delay}] = \dfrac{\lambda P_0^*}{1 - \lambda P_0^*}$

Heavy-traffic analysis, $\lambda \uparrow 1$

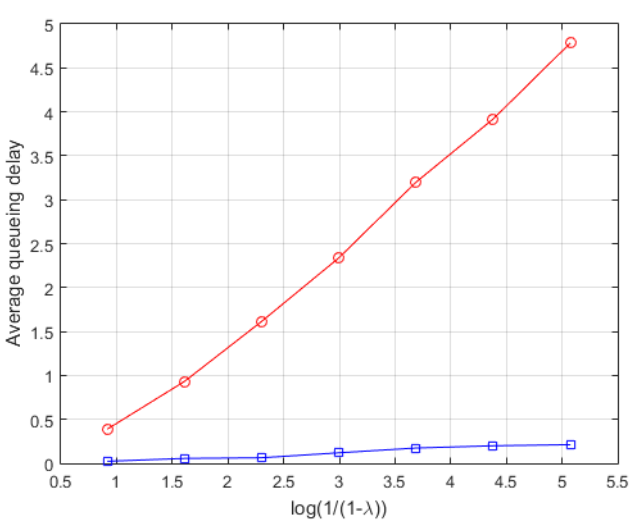
- Choose ν so that $\nu(1-\lambda) = \lambda d$
  - Messages per unit time: $\lambda d n$
  - P_0^* (and therefore, delay): independent of λ
- Send to shortest of d sampled queues
  - Messages per unit time: $\lambda d n$
  - Prob(find an empty queue) $= 1 - \lambda^d \to 0$

Delay analysis (resource constrained case)

- $s_1^* = \lambda$
- $P_0^* = P_0(s^*) = \left[\sum_{k=0}^{C}\left(\frac{\nu(1-\lambda)}{\lambda}\right)^k\right]^{-1}$
  - $\frac{ds_1}{dt}(t) = \lambda[1 - P_0(t)] + \lambda[1 - s_1(t)]P_0(t) - [s_1(t) - s_2(t)]$
  - $\frac{ds_i}{dt}(t) = \lambda[s_{i-1}(t) - s_i(t)]P_0(t) - [s_i(t) - s_{i+1}(t)], \quad i \geq 2$
- $s_i^* = \lambda(\lambda P_0^*)^{i-1}$
- $\mathbf{E}[\text{delay}] = \frac{\lambda P_0^*}{1 - \lambda P_0^*}$

Heavy-traffic analysis, $\lambda \uparrow 1$

- Choose ν so that $\nu(1-\lambda) = \lambda d$
  - Messages per unit time: $\lambda d n$
  - P_0^* (and therefore, delay): independent of λ
- Send to shortest of d sampled queues
  - Messages per unit time: $\lambda d n$
  - Prob(find an empty queue) $= 1 - \lambda^d \to 0$
  - delay $\to \infty$



Negative result

- **Thm:** Under symmetric policies:
$\Big($Memory size $\leq C \log n$ bits, message rate $\leq \alpha n\Big) \Rightarrow$ delay $\geq f(C, \alpha) > 0$

Negative result

- **Thm:** Under symmetric policies:
$\Big($Memory size $\leq C \log n$ bits, message rate $\leq \alpha n\Big) \Rightarrow$ delay $\geq f(C, \alpha) > 0$

- Symmetric policies: "If some memory state m leads to certain events, any given permutation of these events occurs
under some other memory state m' "

Negative result

- **Thm:** Under symmetric policies:
  $\Big(\text{Memory size} \le C \log n \text{ bits, message rate} \le \alpha n\Big) \Rightarrow \text{delay} \ge f(C, \alpha) > 0$

- Symmetric policies: "If some memory state m leads to certain events, any given permutation of these events occurs under some other memory state m' "

  - **Example:** Under m, I query queues $1, 3, 4$
    $\Rightarrow \exists m'$, under which I query $2, 4, 6$

Negative result

- **Thm:** Under symmetric policies:
  $\Big($Memory size $\leq C \log n$ bits, message rate $\leq \alpha n\Big) \Rightarrow$ delay $\geq f(C, \alpha) > 0$

- Symmetric policies: "If some memory state m leads to certain events, any given permutation of these events occurs under some other memory state m' "

  - **Example:** Under m, I query queues $1, 3, 4$
    $\Rightarrow \exists m'$, under which I query $2, 4, 6$

- Some possible actions:
  - Under m, I query $1, \ldots, C$

Negative result

- **Thm:** Under symmetric policies:
$\Big($Memory size $\leq C \log n$ bits, message rate $\leq \alpha n\Big) \Rightarrow$ delay $\geq f(C, \alpha) > 0$

- Symmetric policies: "If some memory state m leads to certain events, any given permutation of these events occurs under some other memory state m' "

  - **Example:** Under m, I query queues $1, 3, 4$
    $\Rightarrow \exists m'$, under which I query $2, 4, 6$

- Some possible actions:
  - Under m, I query $1, \ldots, C$
  - can also query other queues with equal probabilities

Negative result

- **Thm:** Under symmetric policies:
$\Big($Memory size $\leq C \log n$ bits, message rate $\leq \alpha n\Big) \Rightarrow$ delay $\geq f(C, \alpha) > 0$

- Symmetric policies: "If some memory state m leads to certain events, any given permutation of these events occurs under some other memory state m' "

  - **Example:** Under m, I query queues $1, 3, 4$
  $\Rightarrow \exists m'$, under which I query $2, 4, 6$

- Some possible actions:
  - Under m, I query $1, \ldots, C$
  - can also query other queues with equal probabilities

- Impossible: Under m, I query $1, \ldots, C + 1$

Negative result

- **Thm:** Under symmetric policies:
$\Big($Memory size $\leq C \log n$ bits, message rate $\leq \alpha n\Big) \Rightarrow$ delay $\geq f(C, \alpha) > 0$

- Symmetric policies: "If some memory state m leads to certain events, any given permutation of these events occurs under some other memory state m' "

  - **Example:** Under m, I query queues $1, 3, 4$
  $\Rightarrow \exists m'$, under which I query $2, 4, 6$

- Some possible actions:
  - Under m, I query $1, \ldots, C$
  - can also query other queues with equal probabilities

- Impossible: Under m, I query $1, \ldots, C+1$

- Loosely speaking: only C nodes can be treated in a "special" manner

Negative result

- **Thm:** Under symmetric policies:
$\Big($Memory size $\le C \log n$ bits, message rate $\le \alpha n\Big) \Rightarrow$ delay $\ge f(C,\alpha) > 0$

- Symmetric policies: "If some memory state m leads to certain events, any given permutation of these events occurs under some other memory state m' "

  - **Example:** Under m, I query queues $1, 3, 4$
    $\Rightarrow \exists m'$, under which I query $2, 4, 6$

- Some possible actions:
  - Under m, I query $1, \ldots, C$
  - can also query other queues with equal probabilities

- Impossible: Under m, I query $1, \ldots, C+1$

- Loosely speaking: only C nodes can be treated in a "special" manner

- If we get $C+1$ arrivals in a row, and no messages from idle servers, at least one job will be sent to a "random" server

Extensions, variations

Extensions, variations

- Allow queue at the dispatcher, queue capacity $\to \infty$

Extensions, variations

- Allow queue at the dispatcher, queue capacity $\to \infty$
  - can get vanishing delay with constant message rate and zero memory (essentially M/M/n queue)
  - but this is like the large memory case

Extensions, variations

- Allow queue at the dispatcher, queue capacity $\to \infty$
  - can get vanishing delay with constant message rate and zero memory (essentially M/M/n queue)
  - but this is like the large memory case

- Allow finite capacity queue at the dispatcher
  - same negative result

Extensions, variations

- Allow queue at the dispatcher, queue capacity $\to \infty$
  - can get vanishing delay with constant message rate and zero memory (essentially M/M/n queue)
  - but this is like the large memory case

- Allow finite capacity queue at the dispatcher
  - same negative result

- Conjecture: the impossibility result holds for arbitrary (non-symmetric) policies