2IMN20 - Real-Time Systems

Server-based scheduling

Lecturer: Dr. Mitra Nasri

Assistant professor

IRIS Cluster

m.nasri@tue.nl

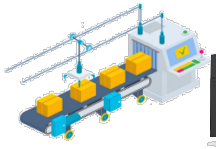
A bit about me

If you are looking for a **Master Thesis topic**, send me an email (to plan a meeting)

Designing scheduling solutions with predictable performance

Since 2009

embedded systems



Product-lines and manufacturing systems (Canon, Bosch)

Real-time systems on cloud (Philips)

Response-time analysis using formal methods

Since 2014

Dependable Robotics on ROS 2

(Robot Operating System)

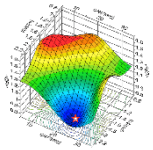
Since 2020

ROS
Robot Operating System

Real-time systems security

Attacks that can cause deadline misses, attacks that are harmful only if they are done at certain moments, vulnerabilities that come from predictability, …

Since 2019



Design-space exploration for timing parameters

How to set parameters of the tasks, how to partition the system, how to set priorities, …

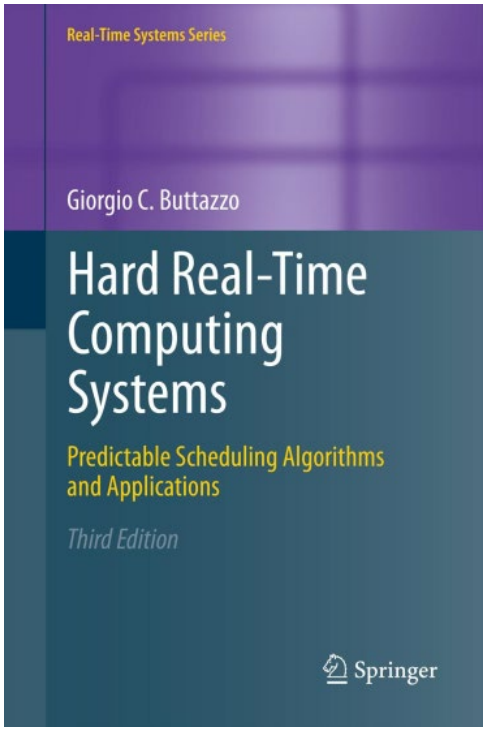
Since 2014

I collaborate with Canon, DAF, Philips, …

IRiS

Sources

Giorgio Buttazzo's book, chapter 9



Some slides have been taken from Giorgio Buttazzo's website:

http://retis.sssup.it/~giorgio/rts-MECS.html

Thank you Giorgio :)

Why should we tame the beasts? And what to do with goblins?

Tasks that may overrun

Aperiodic events

Background tasks

Image: bing.com image creator

Agenda

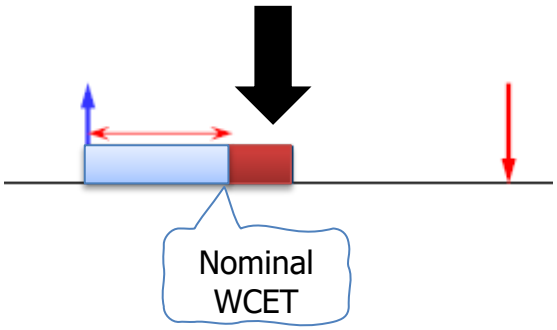
- **Server-based scheduling:** a remedy for aperiodic events, background work, and misbehaving real-time tasks

- **Types of servers**
  - Periodic servers
  - Polling servers
  - Deferrable servers
  - Constant-bandwidth servers (CBS)

The beasts:

What to do with tasks that may overrun their expected execution-time (or nominal WCET)?

Overrun: happens when a task exceeds its “expected execution time”.

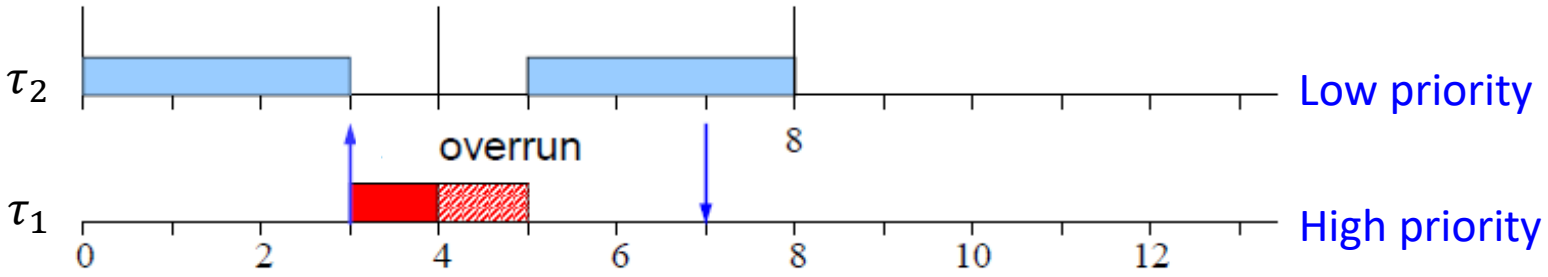


How is this possible?

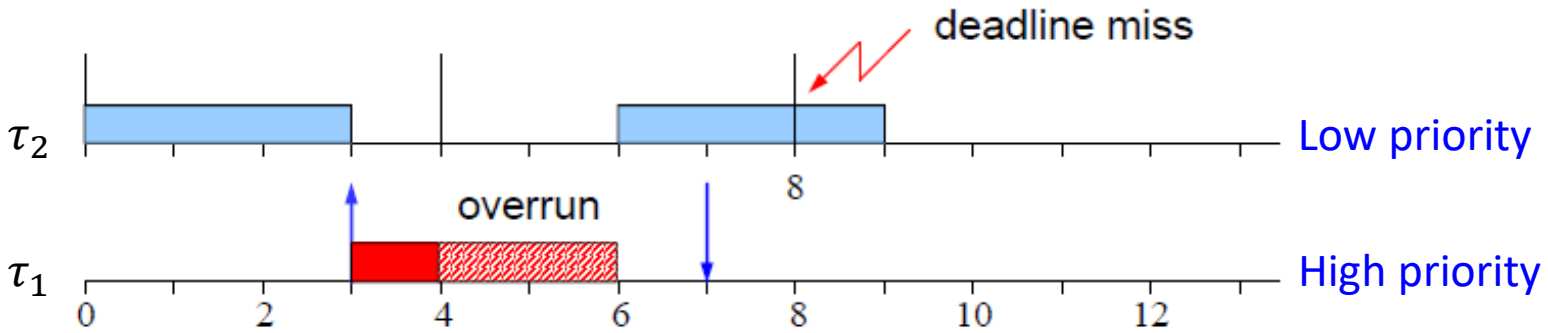
For example, it happens if the WCET has been obtained by measurement (and therefore is inaccurate)

Consequences of overruns

A task's overrun may not cause a deadline miss:



But in general, it may delay the execution of other tasks, causing a deadline miss for them:

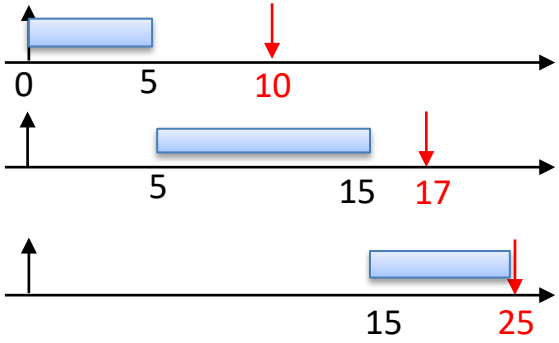


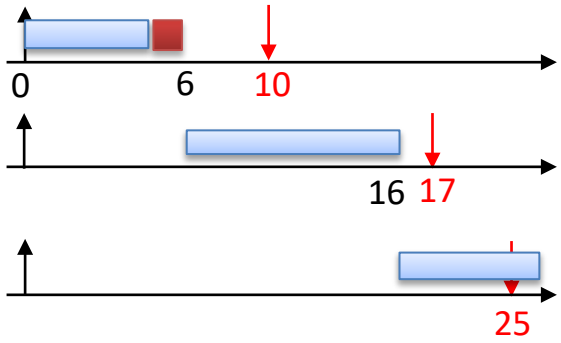
What can go wrong if overruns are not controlled?

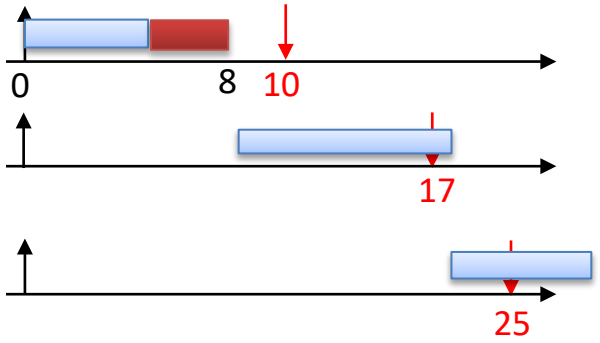
EDF may have **domino effect**:
Namely, one (or more) overrun(s) may result in a series of deadline misses

Example: Consider the following task set scheduled by EDF.

1. **How much overrun in τ_1 will cause one deadline miss for any of these three tasks?** 1 unit
2. **How much overrun in τ_1 will cause two deadline misses for any of these three tasks?** 3 units

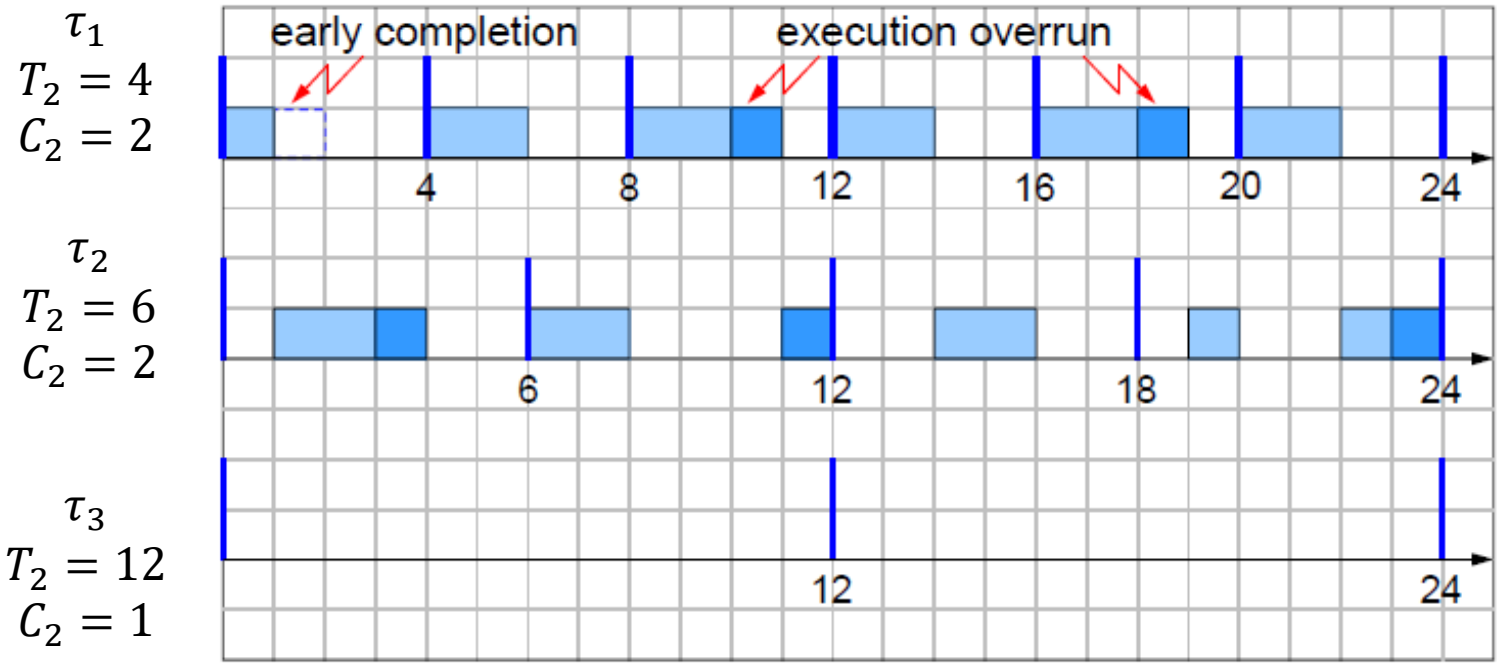






What can go wrong if overruns are not controlled?

In **FP,** low-priority tasks may **starve** due to an overrun in high-priority tasks



Assume a rate monotonic priority assignment.

Server-based scheduling

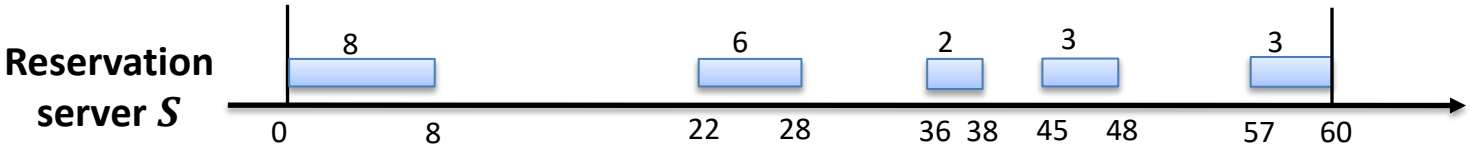
allows scheduling real-time and non-real-time tasks despite potential overruns

A reservation server determines when the processor could become available to the task(s) assigned to it

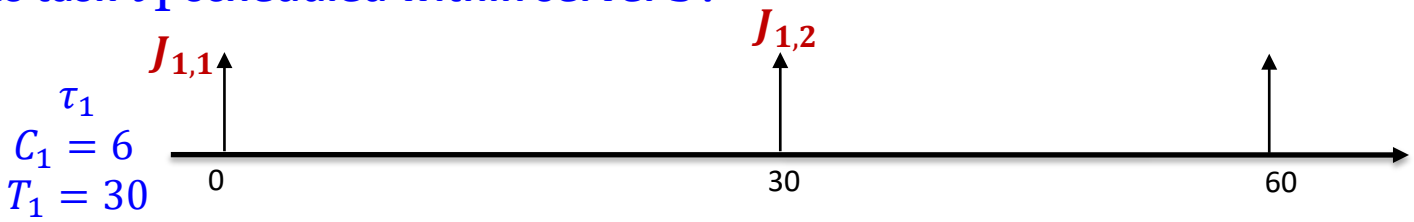
Therefore, it bound the resource consumption of tasks and limit their interference on each other

Example: server-based scheduling

Assume this is the pattern of available budget from a reservation server:



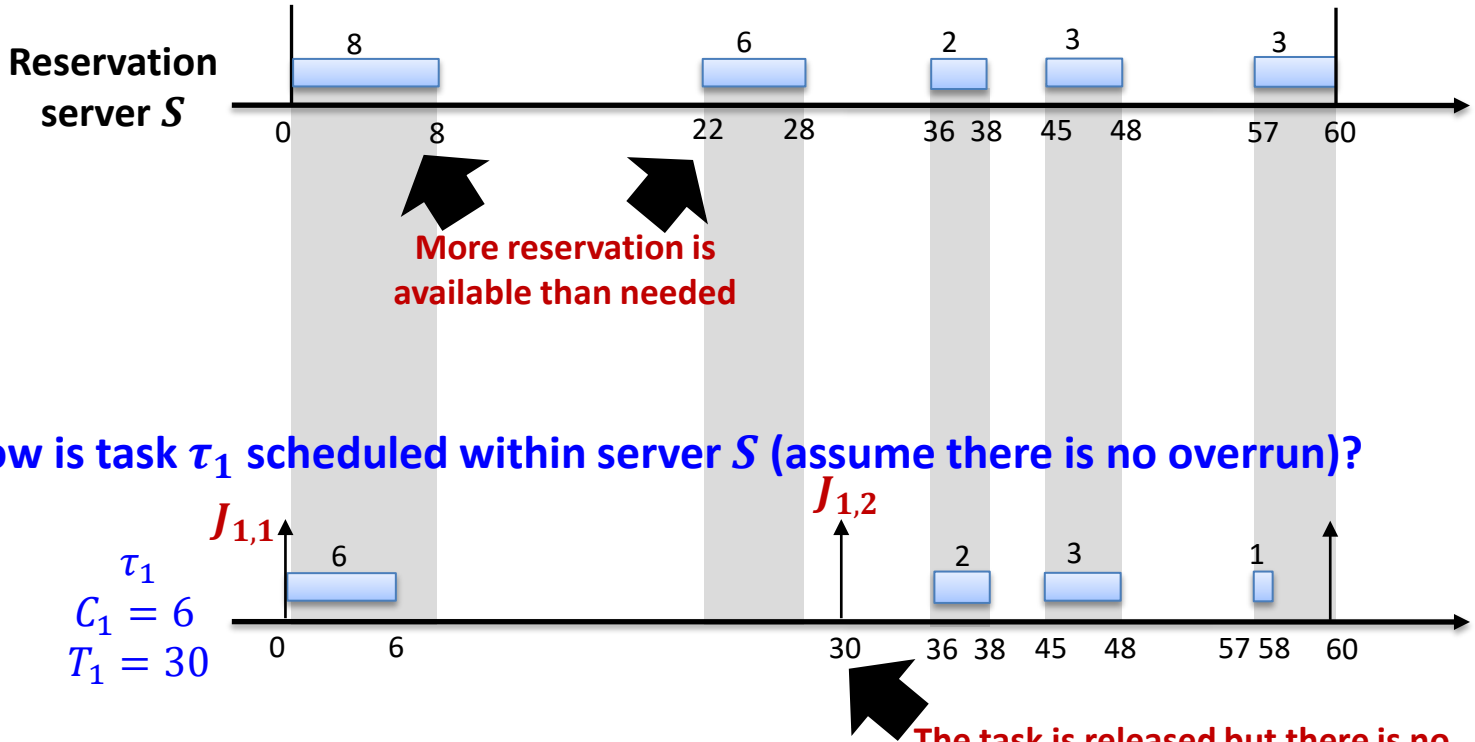
How is task τ_1 scheduled within server S?



A reservation shows when the processor can be assigned to a task

Example: server-based scheduling

Assume this is the pattern of available budget from a reservation server:



How is task τ_1 scheduled within server S (assume there is no overrun)?

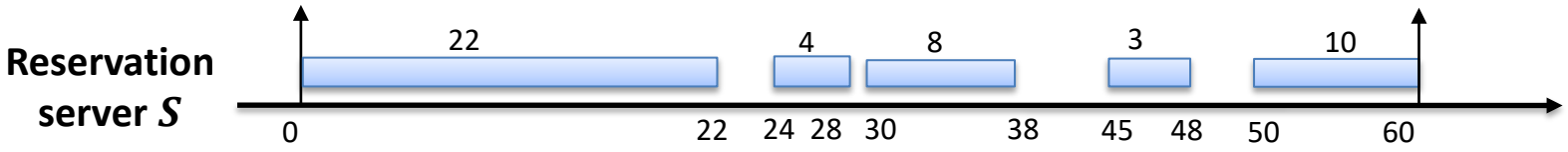
What is the response time of $J_{1,1}$? 6

What is the WCRT of τ_1? 28

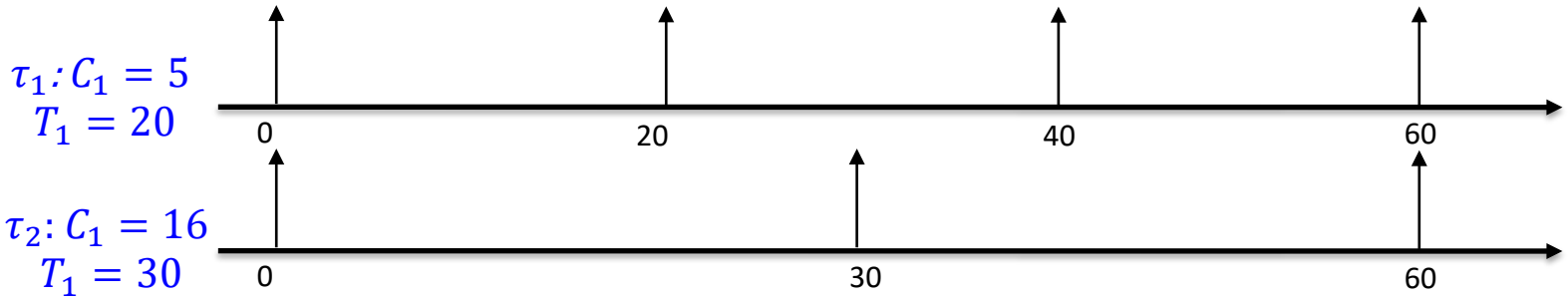
A reservation shows when the processor can be assigned to a task

Example: server-based scheduling

Assume this is the pattern of available budget from a reservation server:



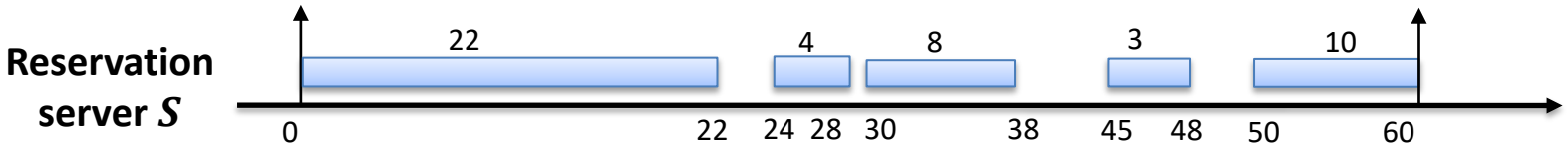
How are tasks τ_1 and τ_2 scheduled via server S? Assume that within the server, these tasks are scheduled by the Rate Monotonic policy.



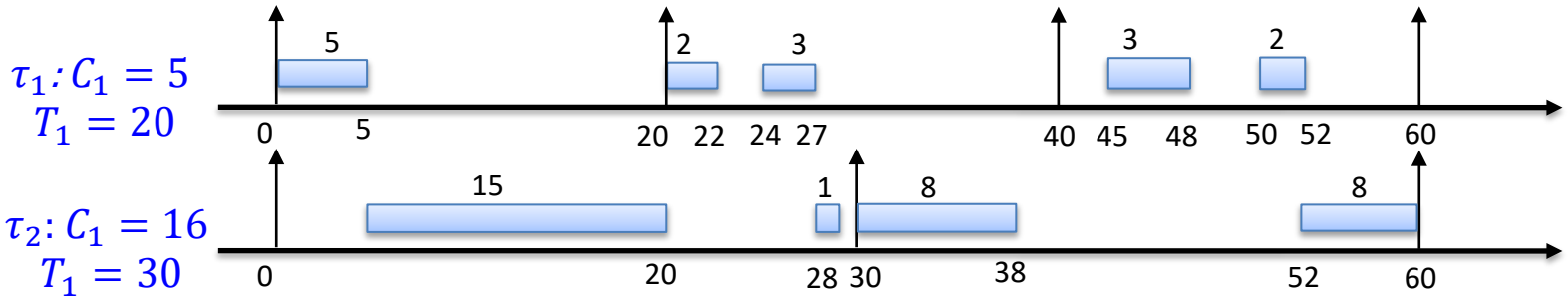
A reservation shows when the processor can be assigned to a task

Example: server-based scheduling

Assume this is the pattern of available budget from a reservation server:



How are tasks τ_1 and τ_2 scheduled via server S? Assume that within the server, these tasks are scheduled by the Rate Monotonic policy.



What is the WCRT of τ_1? **12**

What is the WCRT of τ_2? **30**

A reservation shows when the processor can be assigned to a task

Agenda

- Handling overruns via server-based scheduling
- **Handling aperiodic events and background tasks via server-based scheduling**
- Types of servers

The goblins!

What to do with aperiodic events and background tasks?

Event-driven tasks with deadline

They handle I/O events, garbage collection, etc. and typically have soft deadlines.

Aperiodic tasks are important for the responsiveness of the system for non-critical events.

Background tasks

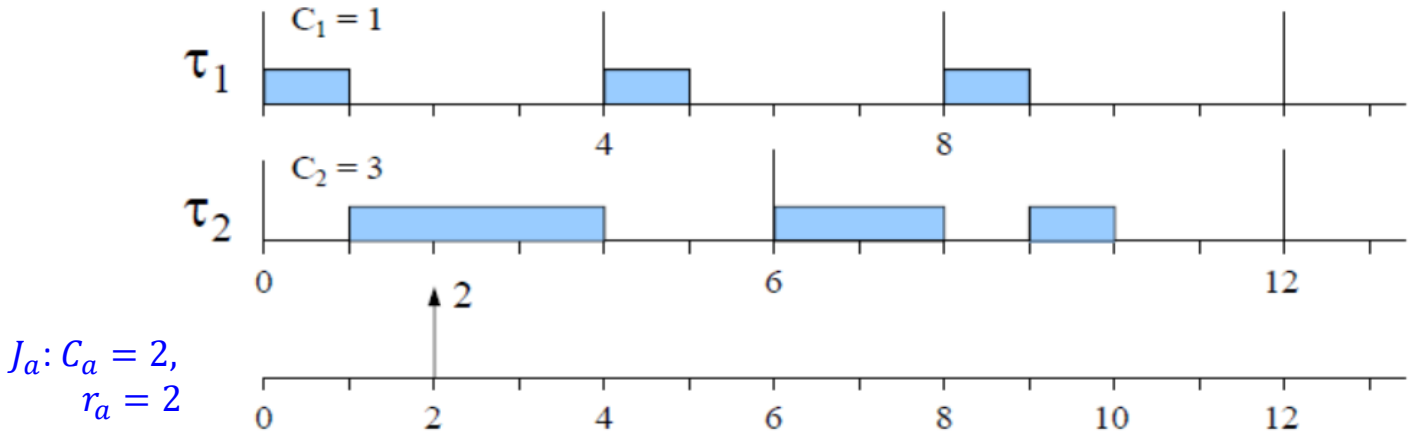
They perform non-critical functionalities like logging, etc.

Background tasks are important for the health of the system, but they do not have deadline.

Image: bing.com image creator.

Aperiodic task scheduling

Example: Two periodic tasks scheduled by RM and a single aperiodic job J_a (arrives at $r_a = 2$ and its WCET is $C_a = 2$):



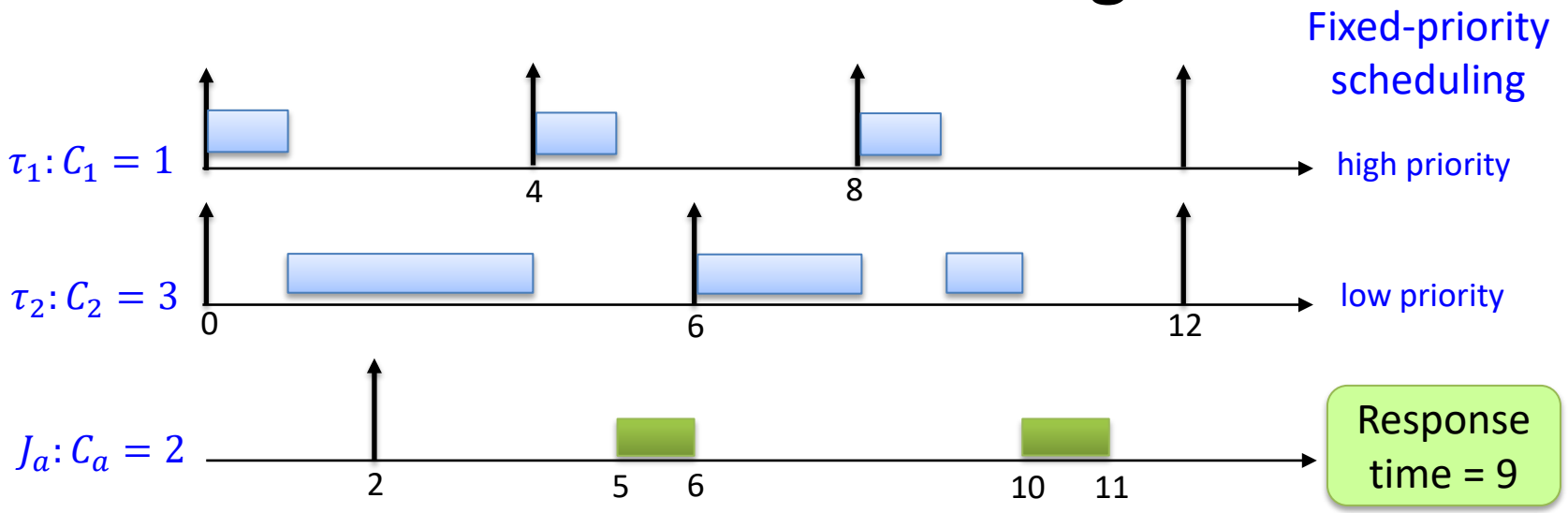
When should we schedule J_a?

Solution 1:
whenever nothing else is running (as a background task with the lowest priority)

Solution 2:
immediately

Solution 3:
Inside a reservation server

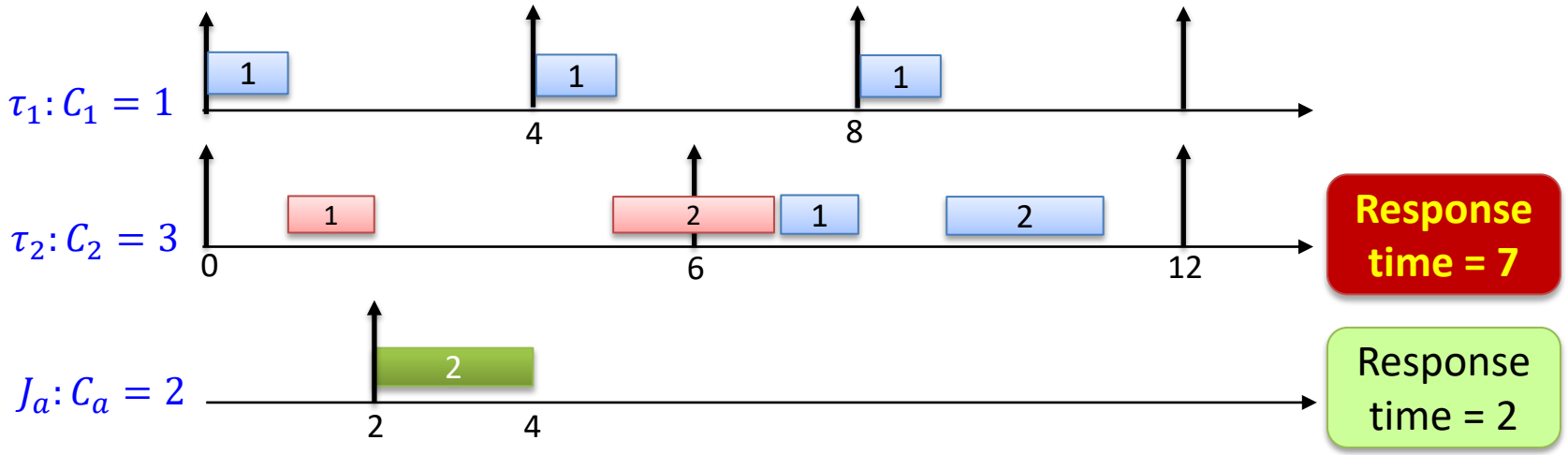
Solution 1: schedule them as a background task



Advantages? **The aperiodic workload will never cause a deadline miss for the periodic tasks**

Disadvantages? **Long response time for the aperiodic task!**

Solution 2: schedule them immediately



Advantages?

- **The aperiodic workload will have a smaller response time.**
- **The system becomes more responsive to the aperiodic events (e.g., interrupts).**

Disadvantages? **Long response time and possibly deadline miss for the periodic tasks!**

Solution 3: use server-based scheduling

Our goal:

Reduce the response time of aperiodic tasks (interrupt latency)

✚

Keep the system **schedulable**

Image: bing.com image creator.

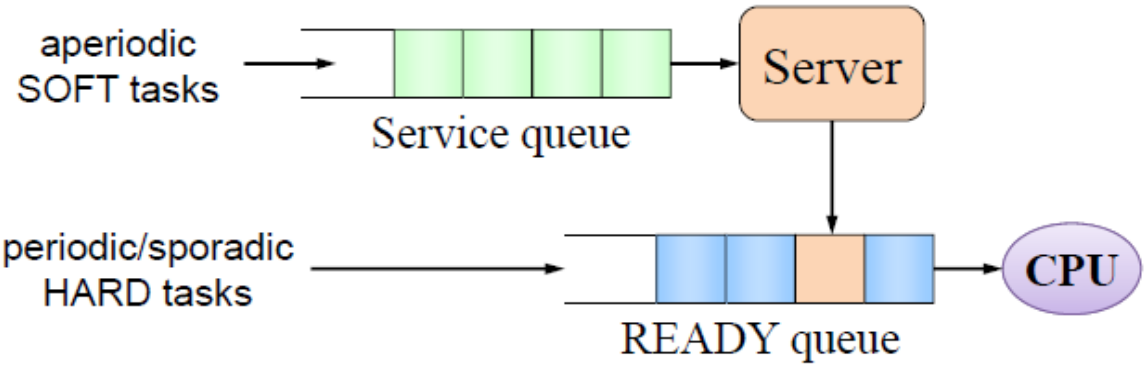
For this, we will need to learn about types of reservation servers and their properties

Type of servers

- **Fixed-priority Servers**
  - **Periodic server**
  - **Polling server (PS)**
  - **Deferrable server (DS)**
  - Sporadic server
  - Slack stealer
- **Dynamic-priority Servers (to be used with EDF scheduler)**
  - Dynamic Polling Server
  - Dynamic Sporadic Server
  - Total Bandwidth Server
  - Tunable Bandwidth Server
  - **Constant Bandwidth Server (CBS)** ← **Implemented in Linux (this is the most interesting one)**

Periodic servers

In this lecture, we assume that there is only one server that services the aperiodic tasks



The server is scheduled as **any periodic task** (it can have any priority).

- maximum budget C_s,
- period T_s and
- priority P_s

Aperiodic tasks assigned to the server can execute **only when** the server has access to the CPU.

Aperiodic tasks can be selected using any queueing discipline (e.g., FIFO).

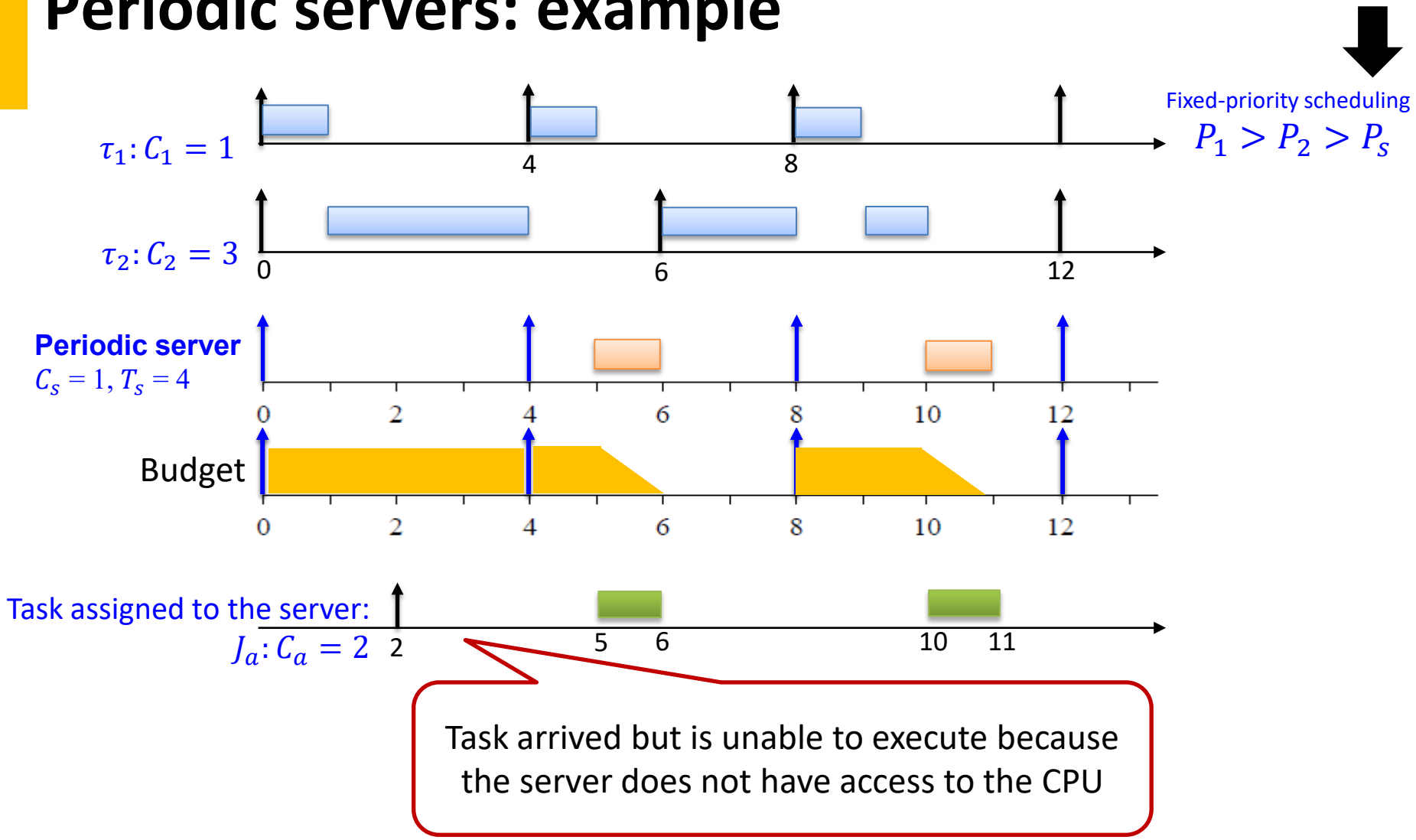
The server **releases a job** with period T_s

The server competes for the CPU with other tasks based on its **priority** P_s

Priority ties are broken in **favor** of the server.

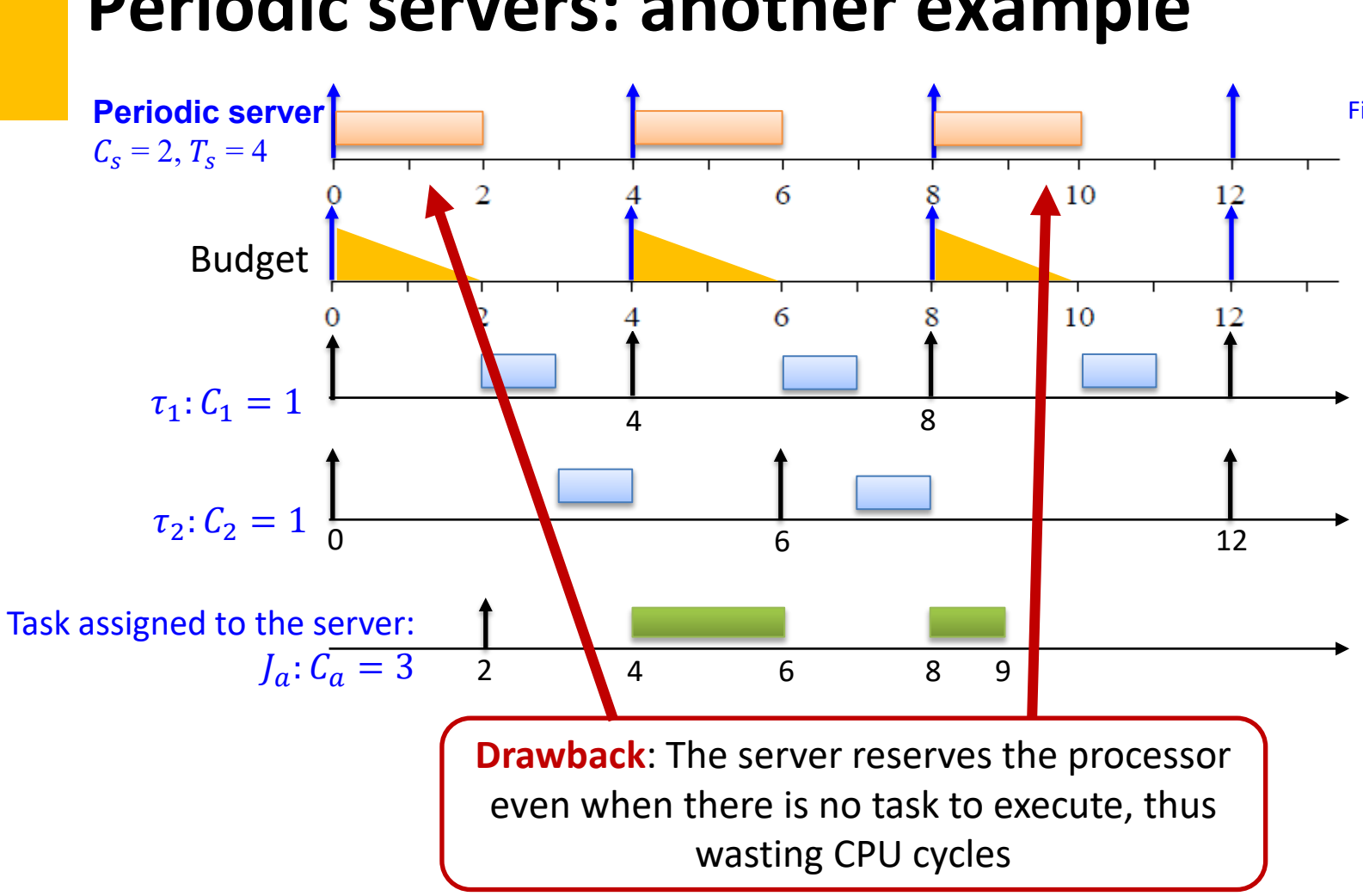
The budget is **replenished** to C_s at each **arrival** of a **new job** of the server

Periodic servers: example



Note: the server in this example may miss its deadline (no schedulability test was done)

Periodic servers: another example



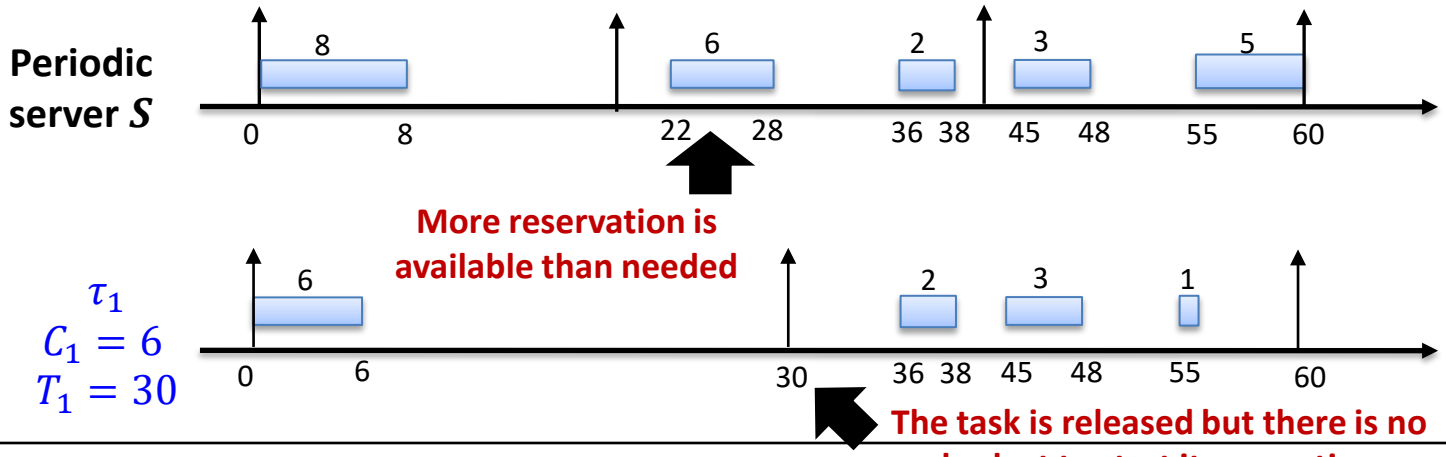
What is bad about periodic servers?

It wastes the budget if there is nothing in the queue to execute

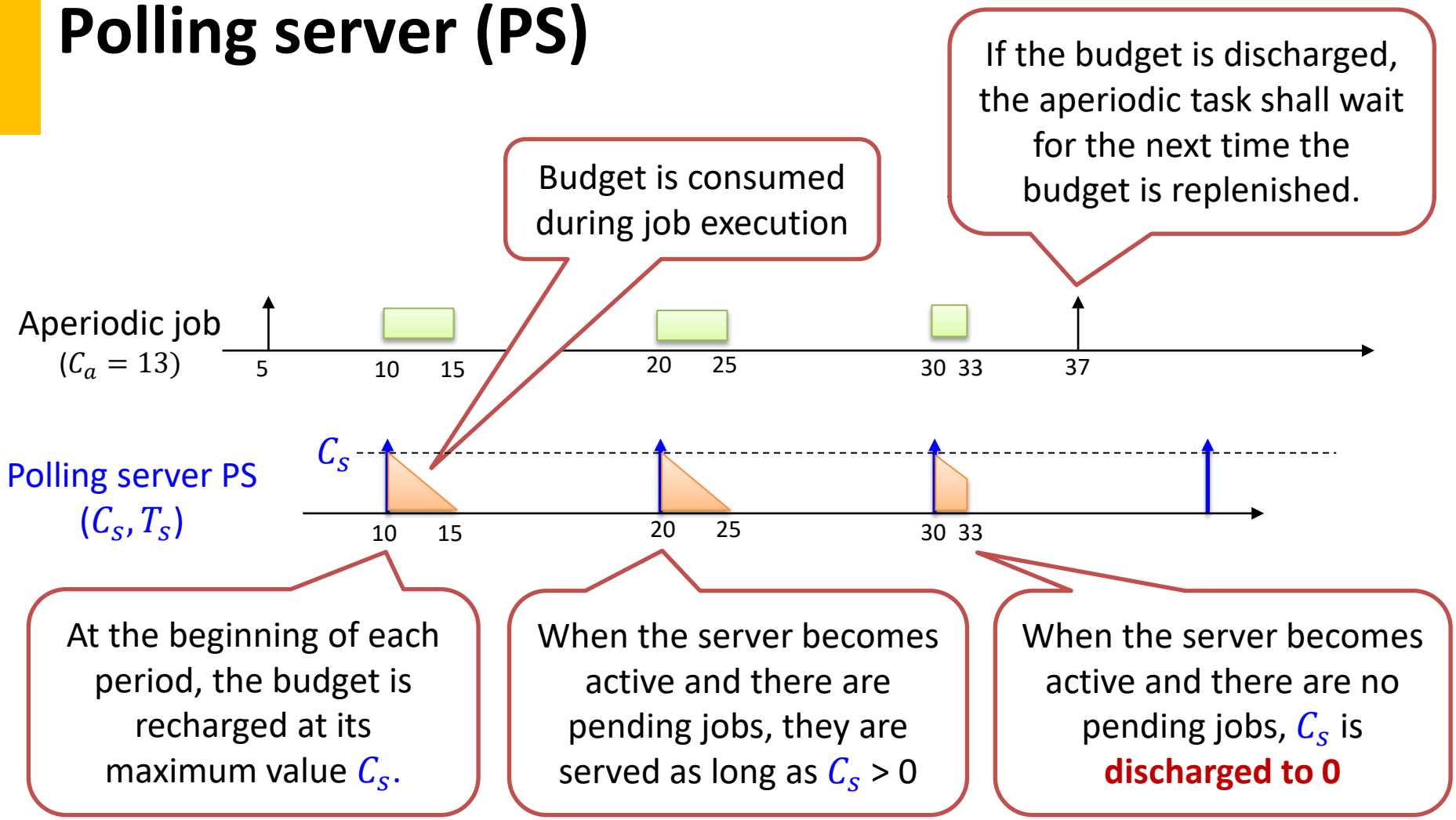
It increases the response time of the tasks running within the server

Solution: discharge the budget to zero if there is nothing in the queue!

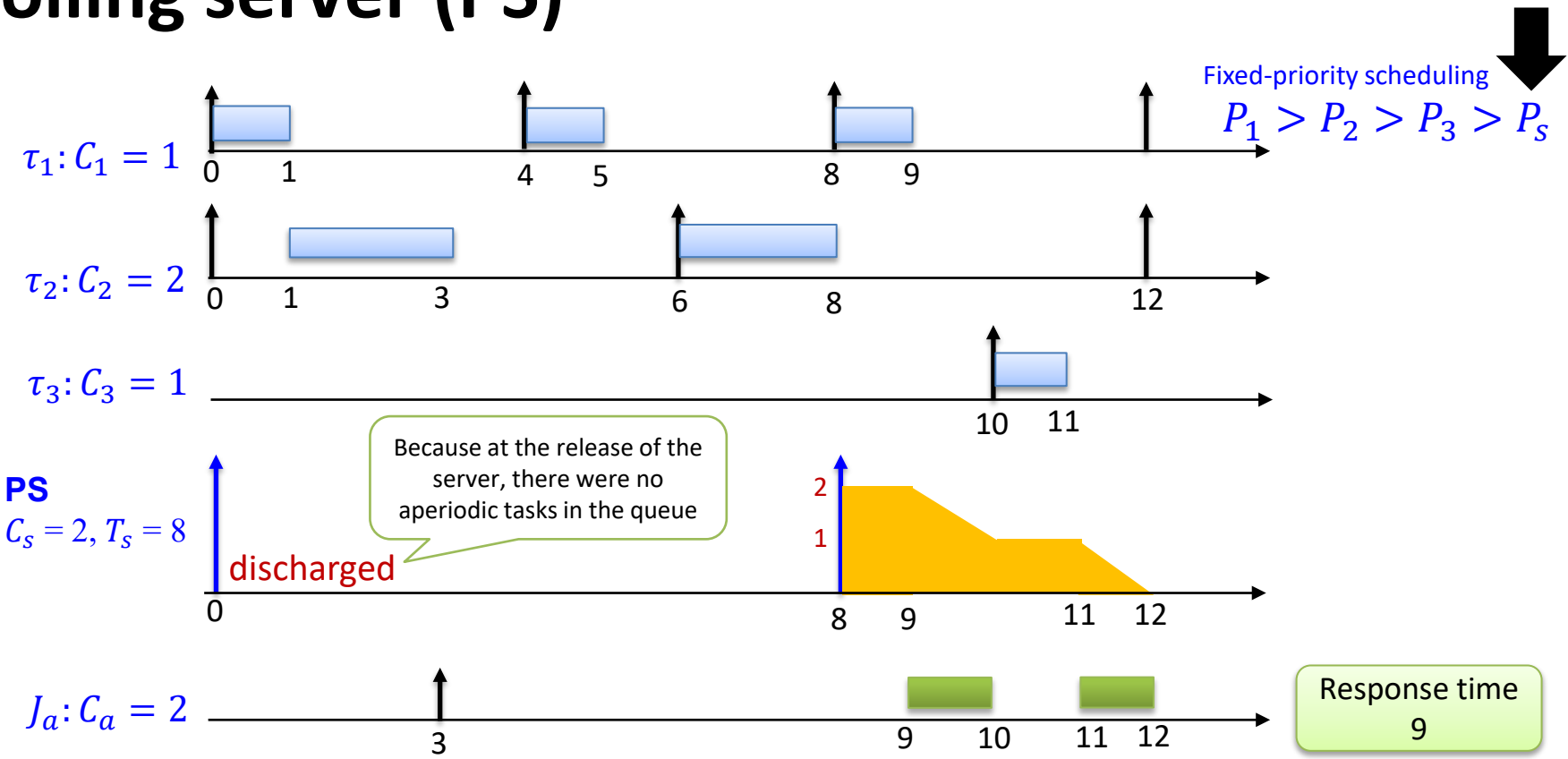
This is called Polling Server (PS)!
(Invented in 1987)



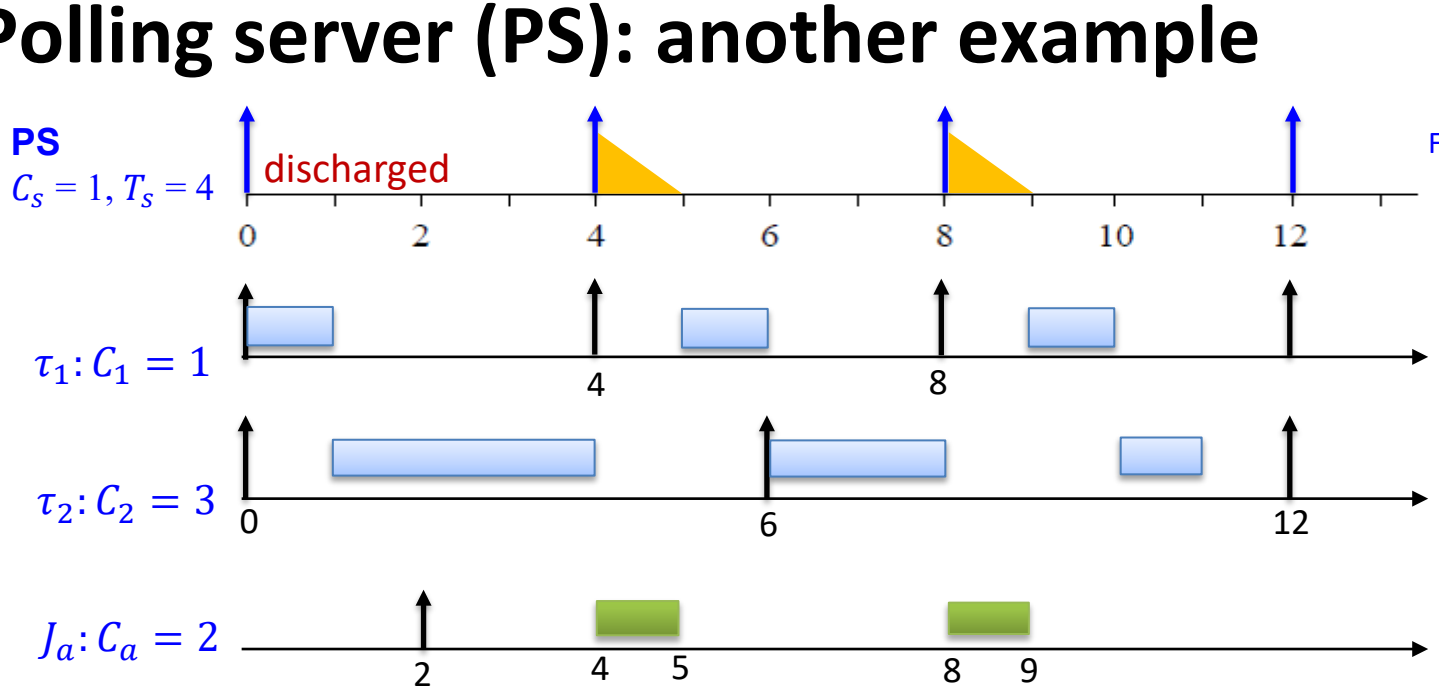
Polling server (PS)



Polling server (PS)



Polling server (PS): another example



Fixed-priority scheduling

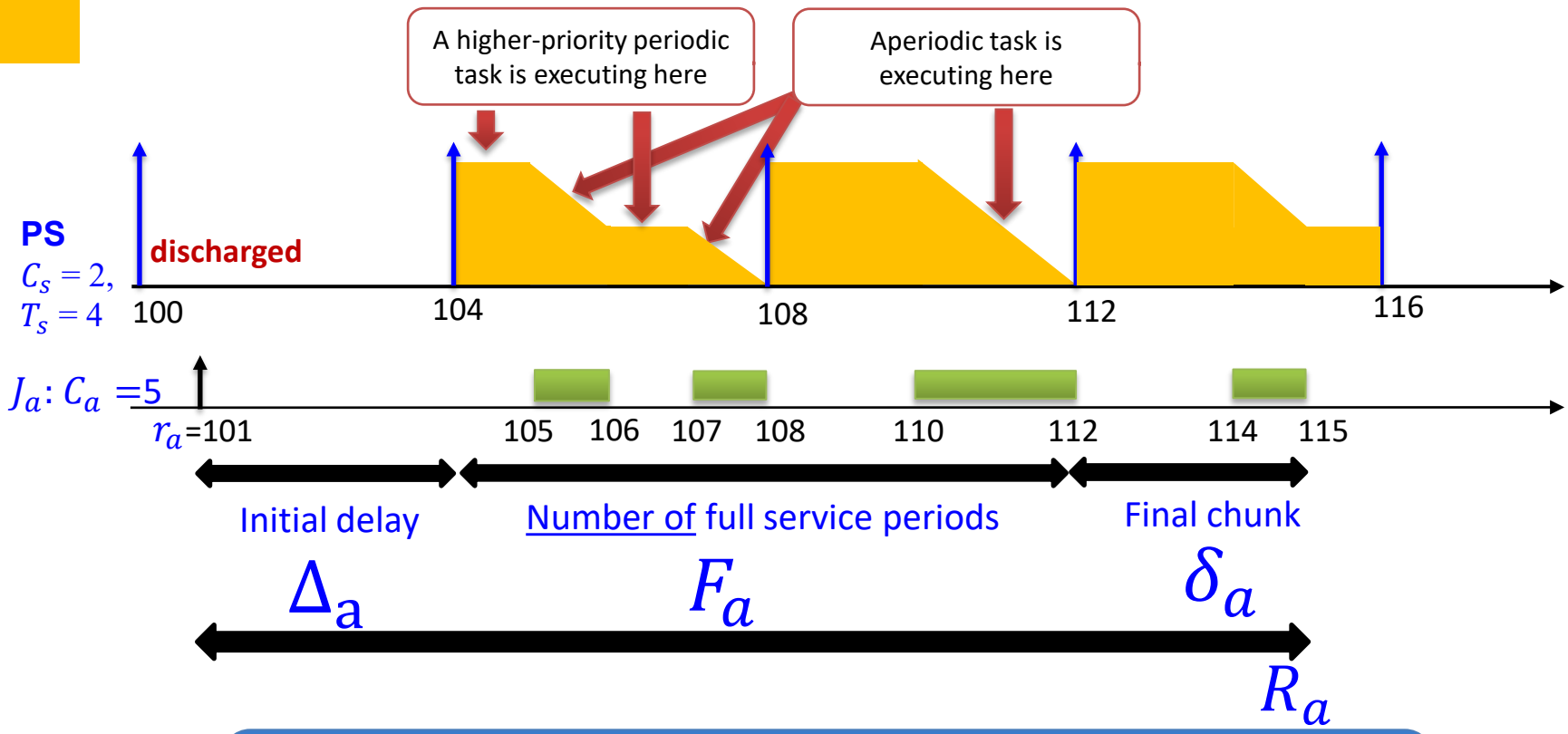
$P_s > P_1 > P_2$

Response time
7

If the budget is discharged, the aperiodic task shall wait for the next time the budget is replenished.

Next, we try to formulate the WCRT of an aperiodic task that is released at time r_a and has execution time of C_a as a function of parameters of the PS (i.e., C_s, T_s)

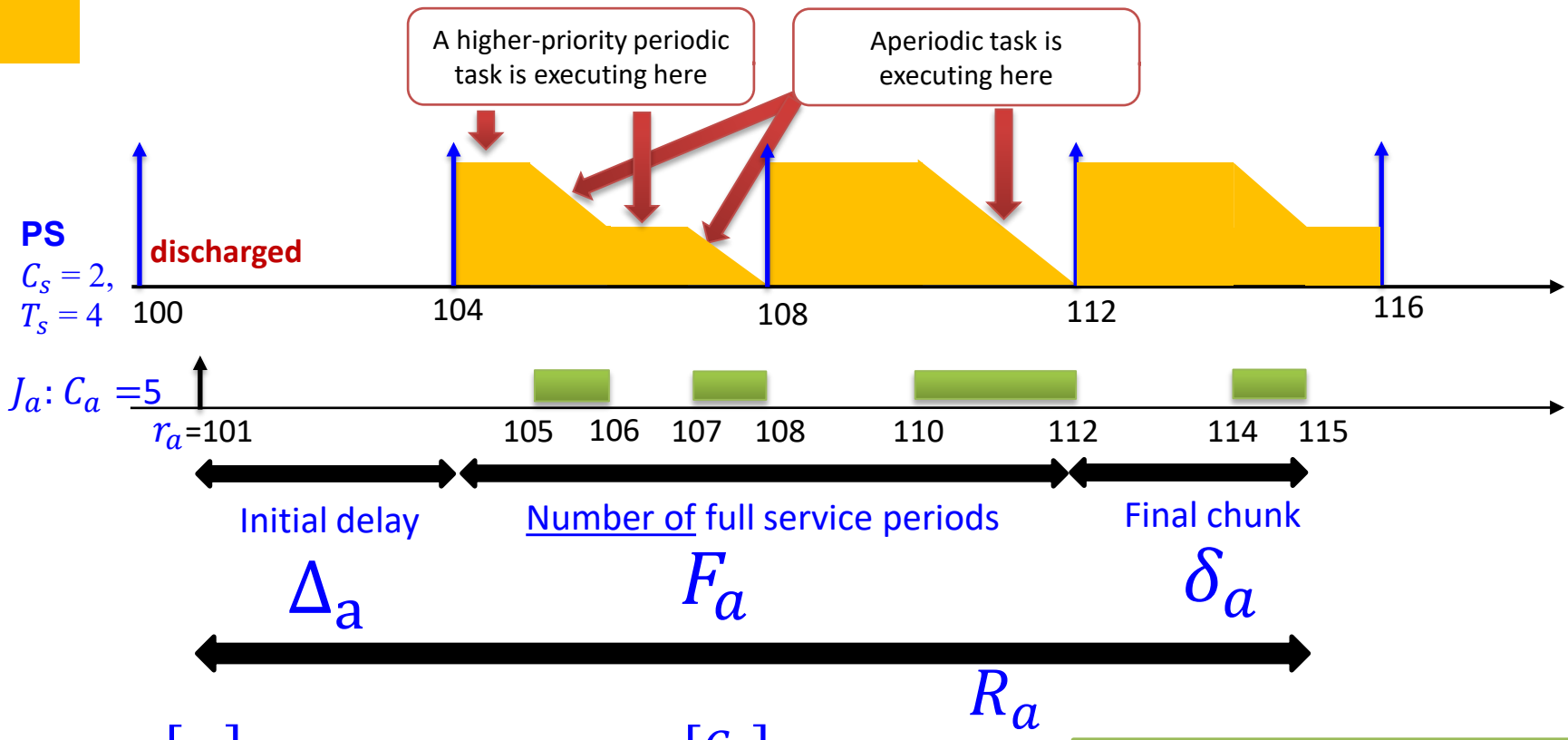
WCRT of an aperiodic task under PS



Does the schedule of "full-service period" impact the worst-case response time of the aperiodic job?
(assume that the server meets its deadlines)

No, because in each release of the server, we can only execute up to C_s units of the execution time of J_a. The WCRT is affected by the "final chunk"!

WCRT of an aperiodic task under PS



$$\Delta_a = \left\lceil \frac{r_a}{T_s} \right\rceil \cdot T_s - r_a$$

A bit pessimistic

$$F_a = \left\lceil \frac{C_a}{C_s} \right\rceil - 1$$

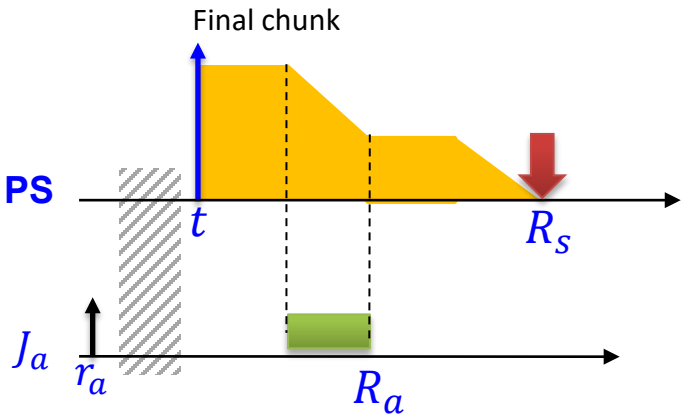
⇒ Duration of full service periods: $F_a \cdot T_s$

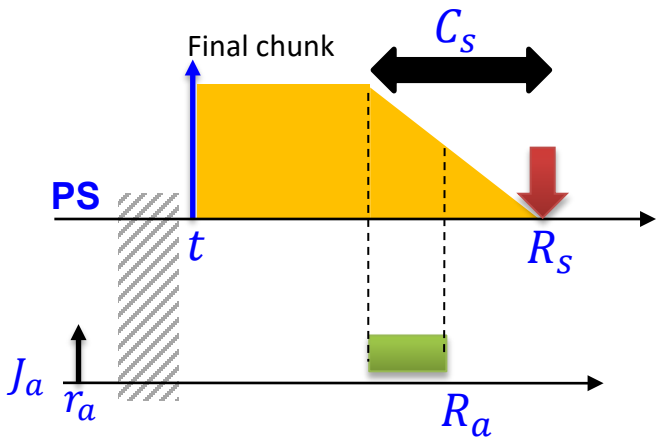
$$R_a = \Delta_a + F_a \cdot T_s + \delta_a$$

WCRT of an aperiodic task under PS

Finding δ_a

We can use the WCRT analysis (RTA) to obtain the **WCRT of the server**, denoted by R_s, (assuming that it consumes all its C_s units of budget) because a server is like a periodic task in the task set.

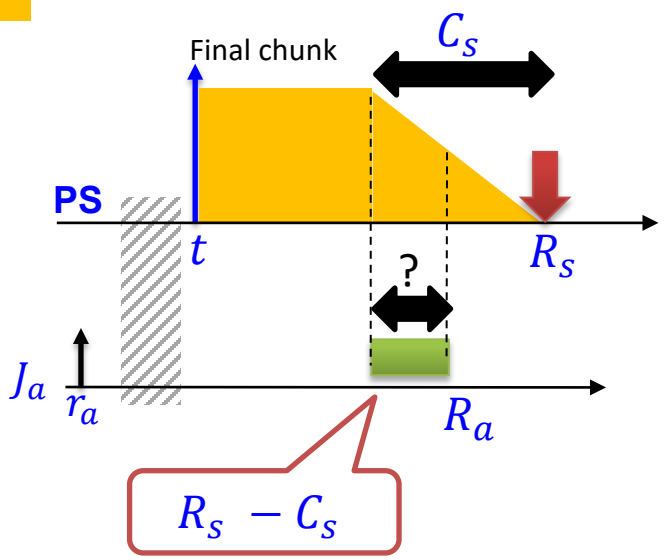




Now, given that we know the value of R_s, which scenario creates a larger R_a?

WCRT of an aperiodic task under PS

Finding δ_a



The latest start time of the aperiodic job in the final chunk

$$\delta_a \leq (R_s - C_s) + (C_a - F_a C_s)$$

Remained execution time of the aperiodic task in the last chunk

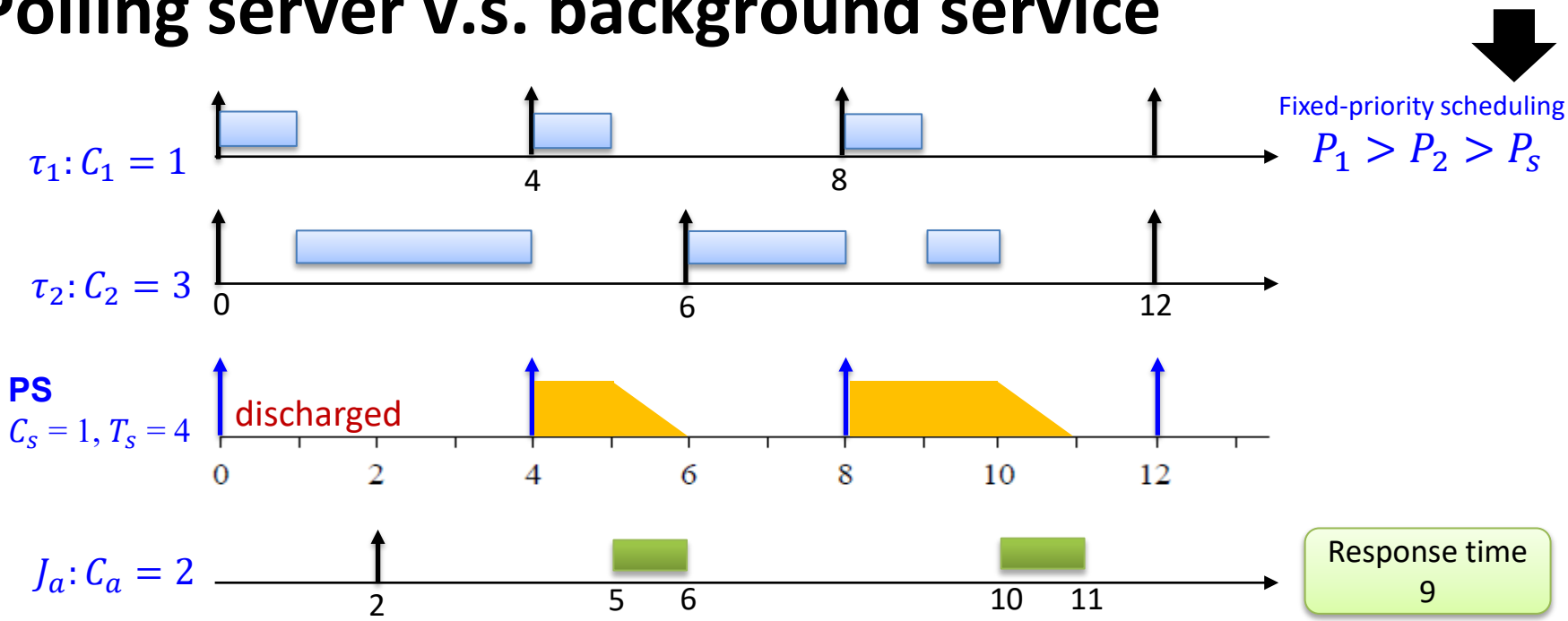
$$R_a = \left\lceil \frac{r_a}{T_s} \right\rceil \cdot T_s - r_a + \left(\left\lceil \frac{C_a}{C_s} \right\rceil - 1 \right) \cdot T_s + (R_s - C_s) + C_a - F_a C_s$$

polling server

v.s.

a background service

Polling server v.s. background service



If the polling server has the lowest priority, can we say it is like if the aperiodic tasks are executed as a background service?

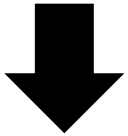
No (because the budget might be "discharged" just before the arrival of the aperiodic task)

How to improve PS?

If aperiodic tasks come later than the release time of the server, they have to wait for the next release of the server!

How can we fix this?

Keep the budget even if there is no aperiodic task in the queue.

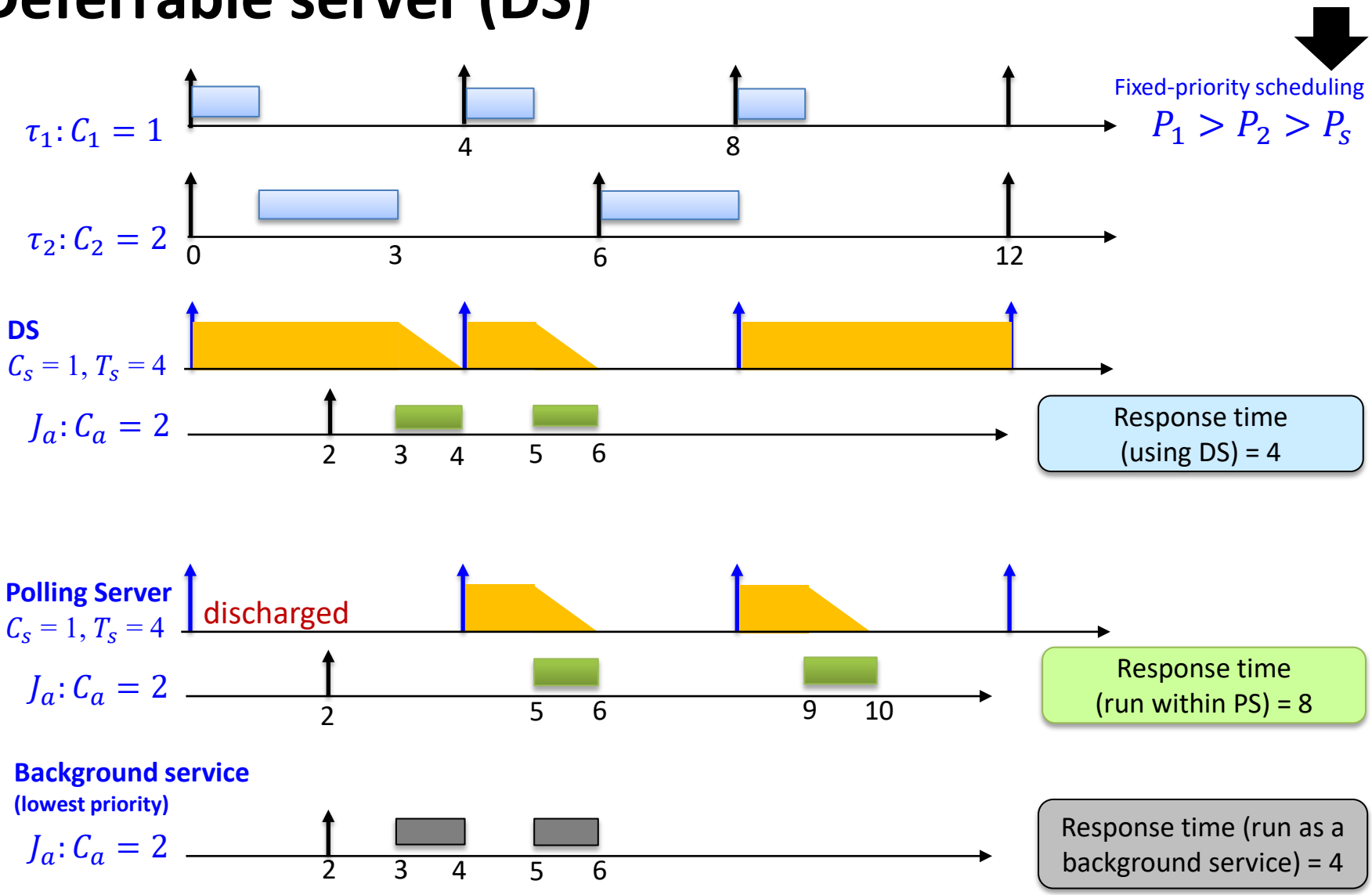


This is called Deferrable Server (DS)!

(Invented in 1987 and improved in 1995)

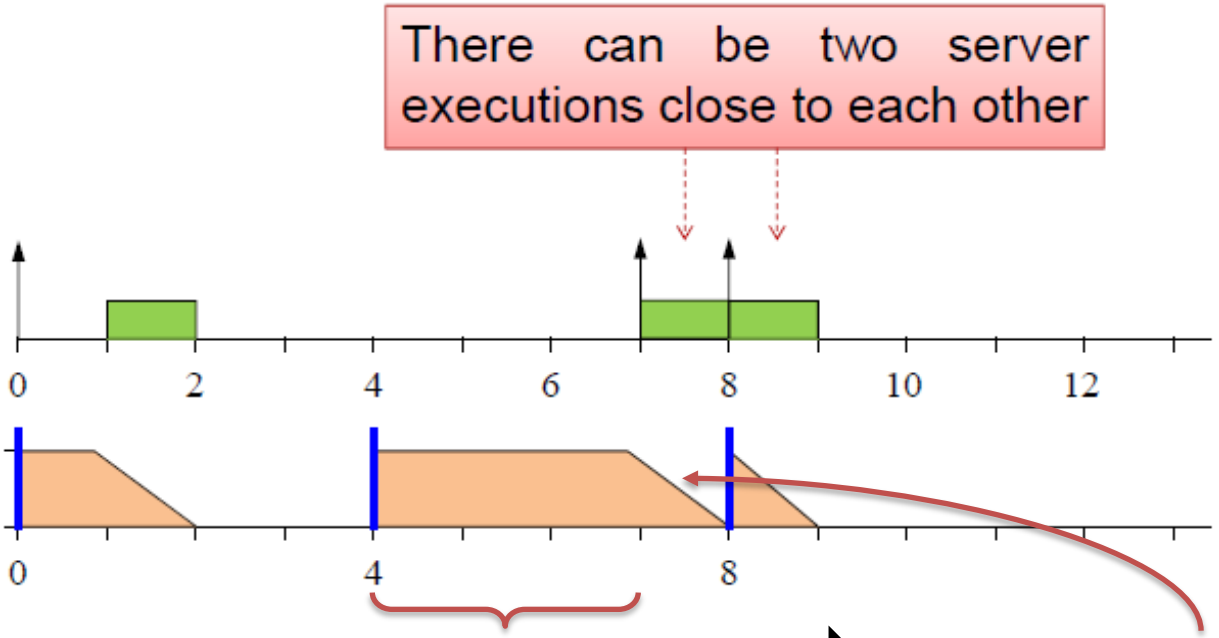
Deferrable server (DS)

DS: Keep the budget even if there is no aperiodic task in the queue.



Issue with DS

- DS does not behave like a periodic task. It is more invasive than PS.
- Keeping the budget decreases the schedulable utilization bound of fixed-priority scheduling.



DS is waiting for an aperiodic task to arrive

Hence, if it has higher priority than some of the periodic tasks, it can "preempt" them unexpectedly

WCRT of periodic tasks in the presence of a DS

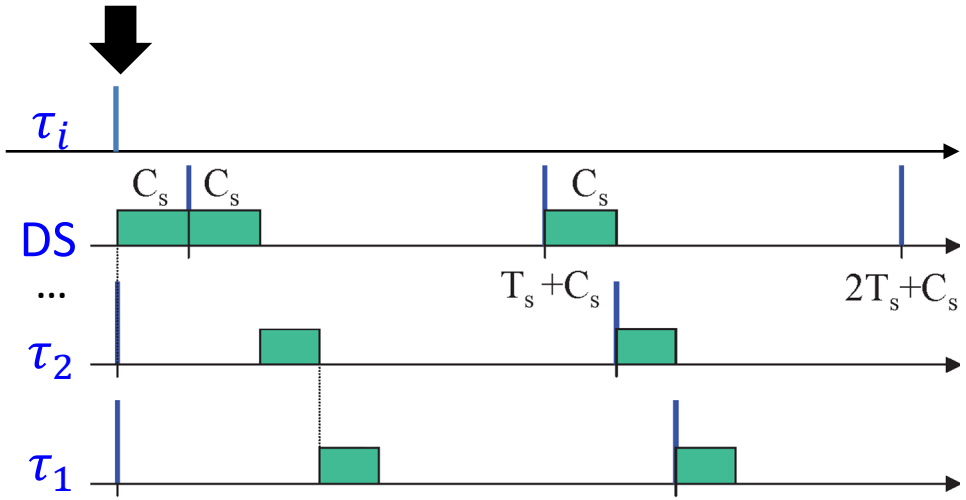
Assume we use FP with the following priorities: $P_1 > P_2 > \cdots P_{i-1} > P_s > P_i > \cdots > P_n$

Can DS impact the schedule of ANY higher-priority task?

No! DS never preempts a task with a higher priority

Can DS impact the schedule of any low-priority task?

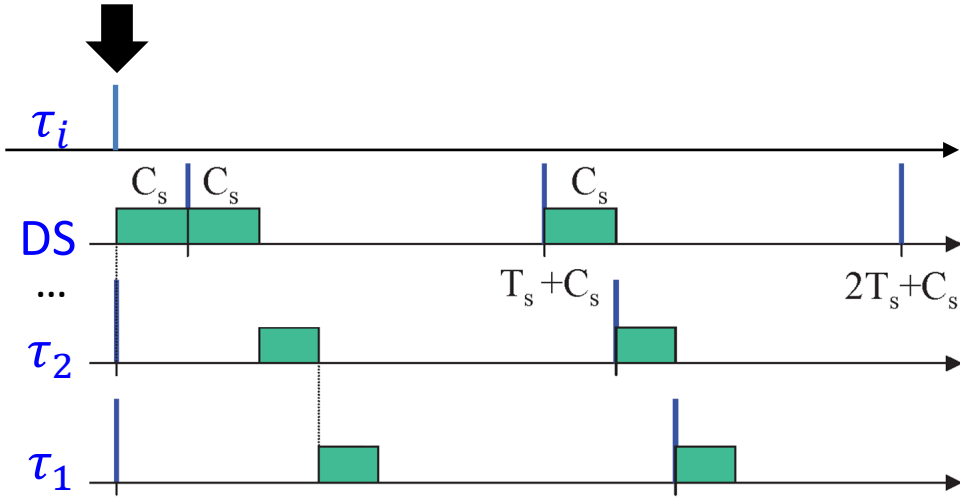
Yes!



How is it different from a normal fixed-priority schedule?

WCRT of periodic tasks in the presence of a DS

Assume we use FP with the following priorities: $P_1 > P_2 > \cdots P_{i-1} > P_s > P_i > \cdots > P_n$



- The critical instance in FP is when all higher-priority tasks are **released together**.
- However, here we need to account for the previous job of DS that has not been scheduled until just before the release of τ_i

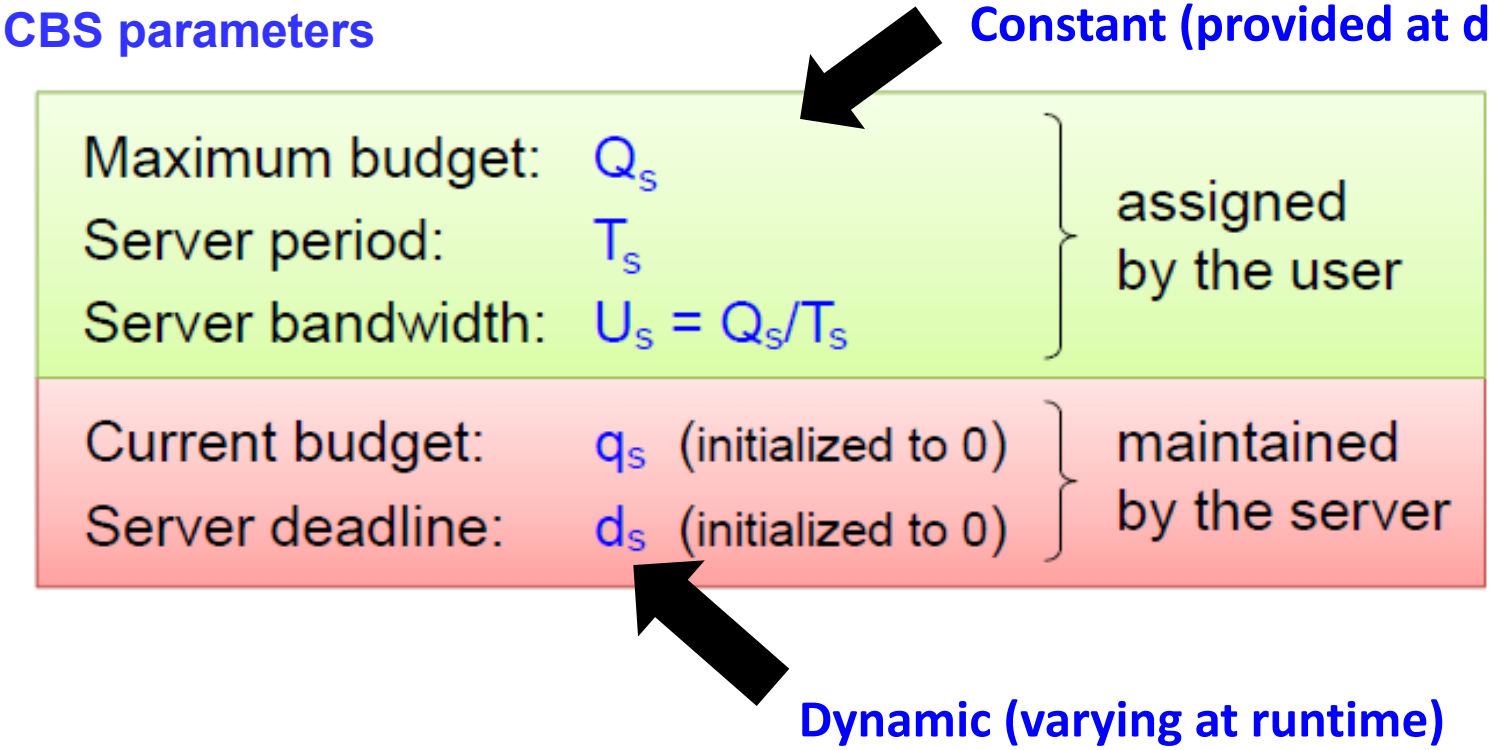
Type of servers

- Fixed-priority Servers
  - Periodic server (previous lecture)
  - Polling server
  - Deferrable server
  - Sporadic server
  - Slack stealer
- Dynamic-priority Servers (to be used with EDF scheduler)
  - Dynamic Polling Server
  - Dynamic Sporadic Server
  - Total Bandwidth Server
  - Tunable Bandwidth Server
  - **Constant Bandwidth Server**

Implemented in Linux
(this is the most interesting one)

Constant bandwidth server (CBS)

- It is a server designed to work with the **EDF scheduling policy**
- It keeps the "bandwidth = utilization" constant.



CBS idea

Replenish the budget as soon as it is finished!

→ **Allows using the resource more efficiently**

However, each time you do so, increase the current deadline by the value of T_s

→ **Why do we do this?**

Always make sure that the server does not have a utilization larger than U_s

It smoothly reduces the priority of the current job of the server among other jobs in the system
while keeping the utilization of the server constant!

Note: in EDF, the priority comes from the absolute deadline.

Recall: the absolute deadline determines a job's priority in EDF

CBS idea

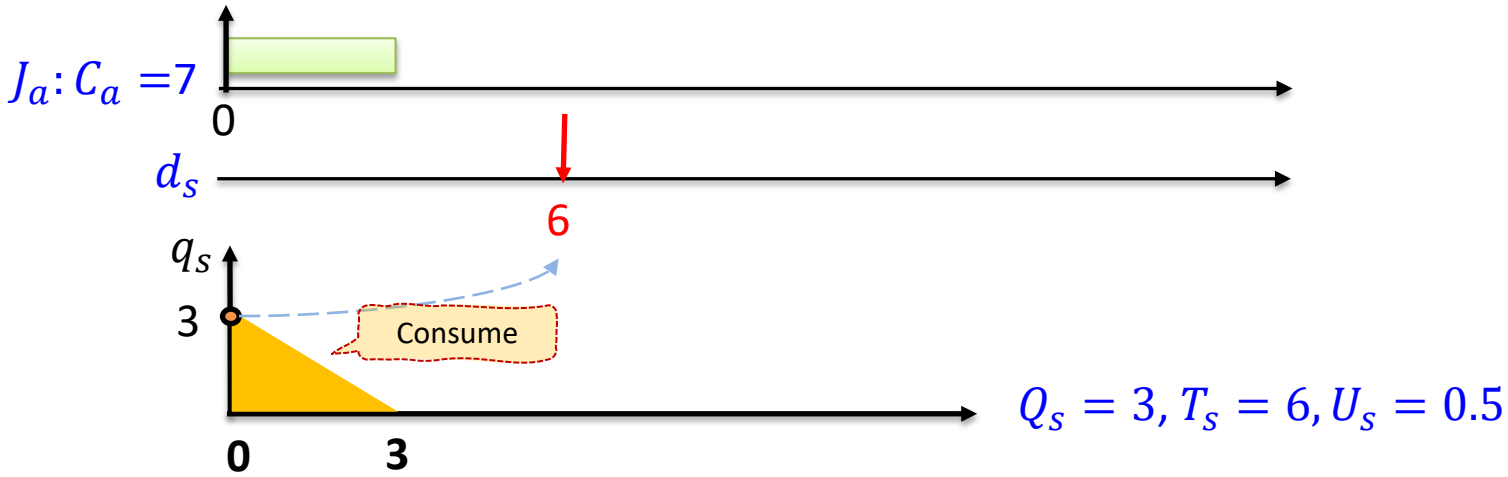
Replenish the budget as soon as it is finished!

However, each time you do so, increase the current deadline by the value of T_s

Always make sure that the server does not have a utilization larger than U_s

What is the benefit?

It smoothly reduces the priority of the current job of the server among other jobs in the system
while keeping the utilization of the server constant!



Recall: the absolute deadline determines a job's priority in EDF

CBS idea

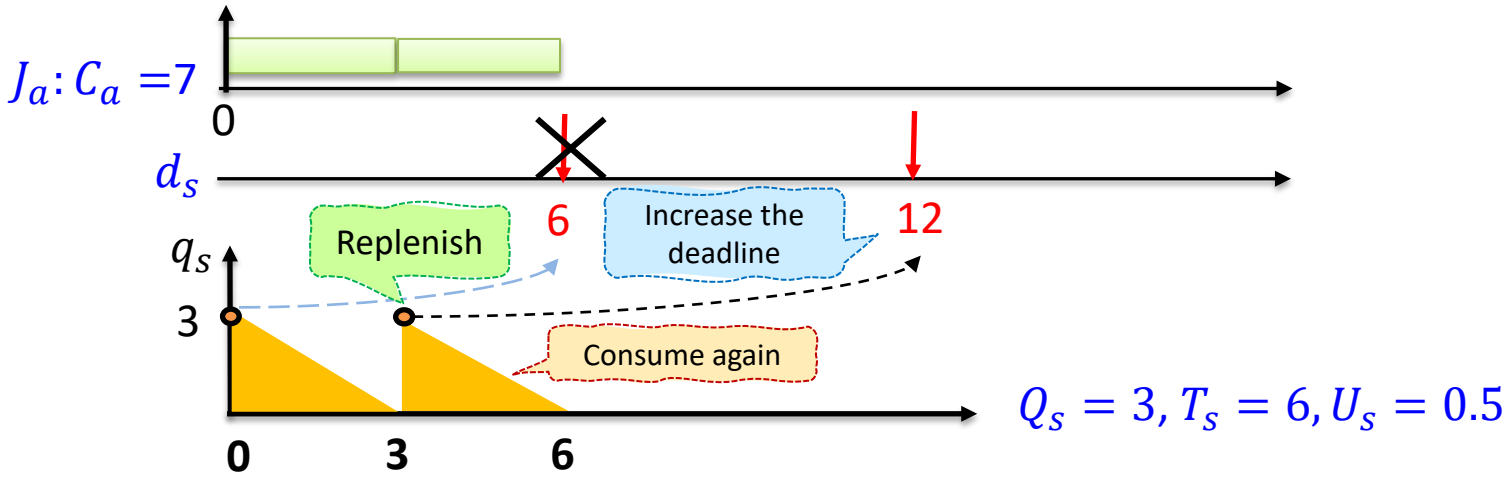
Replenish the budget as soon as it is finished!

However, each time you do so, increase the current deadline by the value of T_s

Always make sure that the server does not have a utilization larger than U_s

What is the benefit?

It smoothly reduces the priority of the current job of the server among other jobs in the system
while keeping the utilization of the server constant!



Recall: the absolute deadline determines a job's priority in EDF

CBS idea

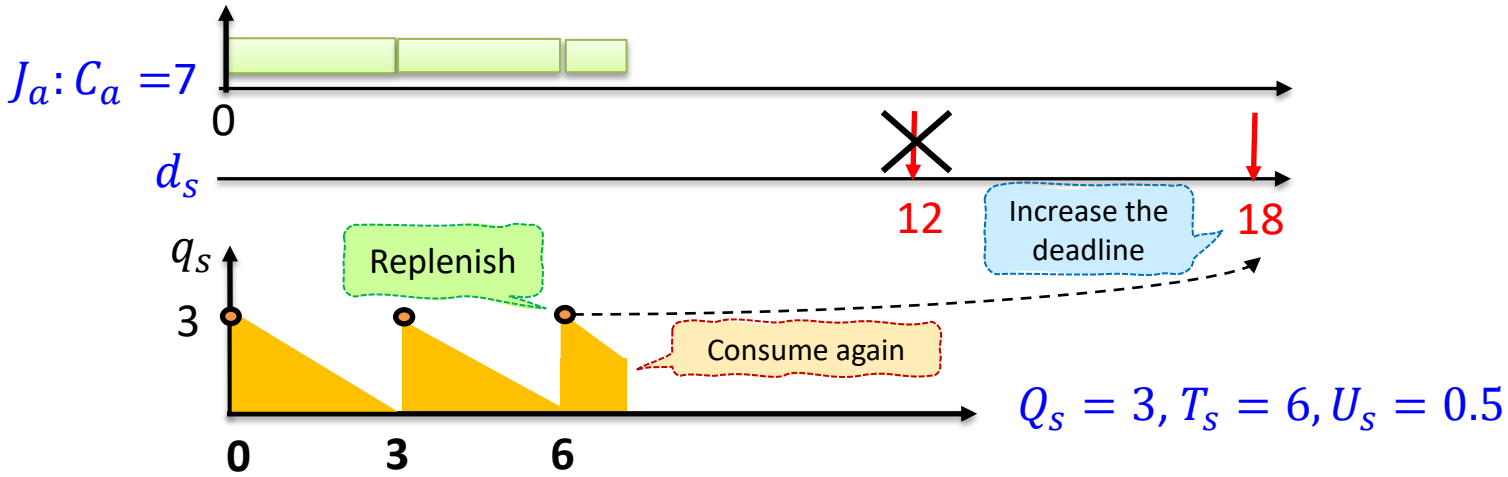
Replenish the budget as soon as it is finished!

However, each time you do so, increase the current deadline by the value of T_s

Always make sure that the server does not have a utilization larger than U_s

What is the benefit?

It smoothly reduces the priority of the current job of the server among other jobs in the system!
While keeping the utilization of the server constant!



Recall: the absolute deadline determines a job's priority in EDF

Basic CBS rules

Arrival of job J_k at time r_k $\Rightarrow$ assign a new d_s

If ($\exists$ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
  $q_s \leftarrow Q_s$
  $d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}

Case 1: Some jobs already waiting? Just put the new job in the queue (do nothing else)

Case 2: Nothing in the queue but with the current budget, we won't respect the **constant-bandwidth rule** in the interval $[r_k, d_s]$ (namely, the remaining budget divided by the length of the interval results in a larger utilization than U_s)?

Then set a new budget (replenish) and set a new deadline (reset the server)

Case 3: Nothing is in the queue, and we respect the constant-bandwidth rule?

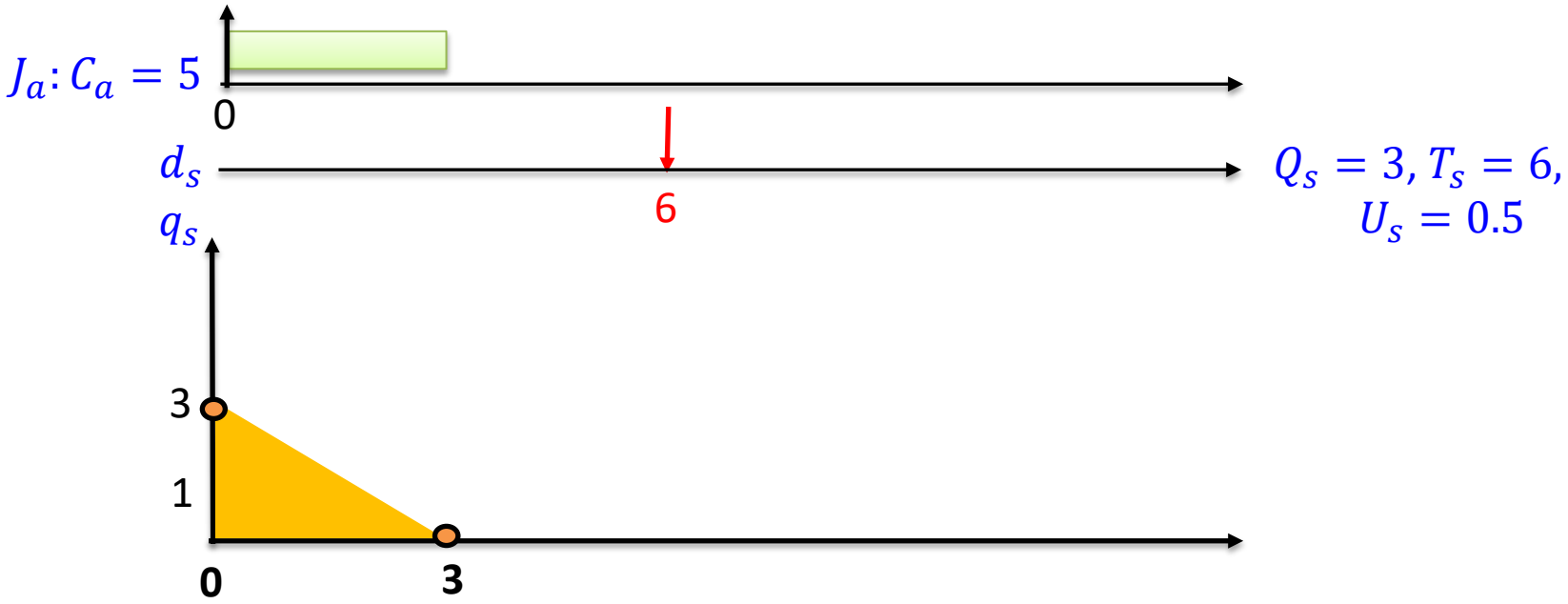
Then continue with the current budget and deadline.

Budget exhausted $\Rightarrow$
replenish and set a new deadline

$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$

CBS budget exhaustion

$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$



At time 0:

What is d_s? 6

What is q_s? 3

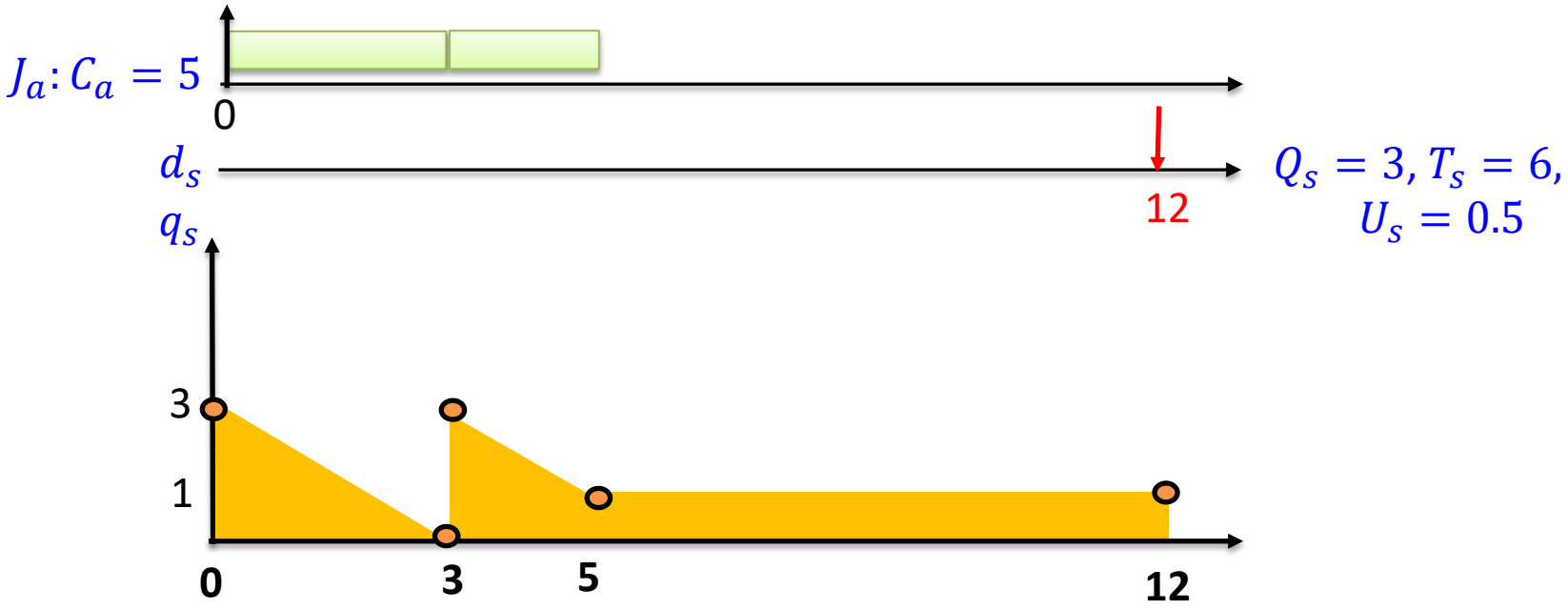
At time 3 after the budget exhausts:

What is d_s? 12

What is q_s? 3

CBS budget exhaustion

$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$



$Q_s = 3, T_s = 6,$
$U_s = 0.5$

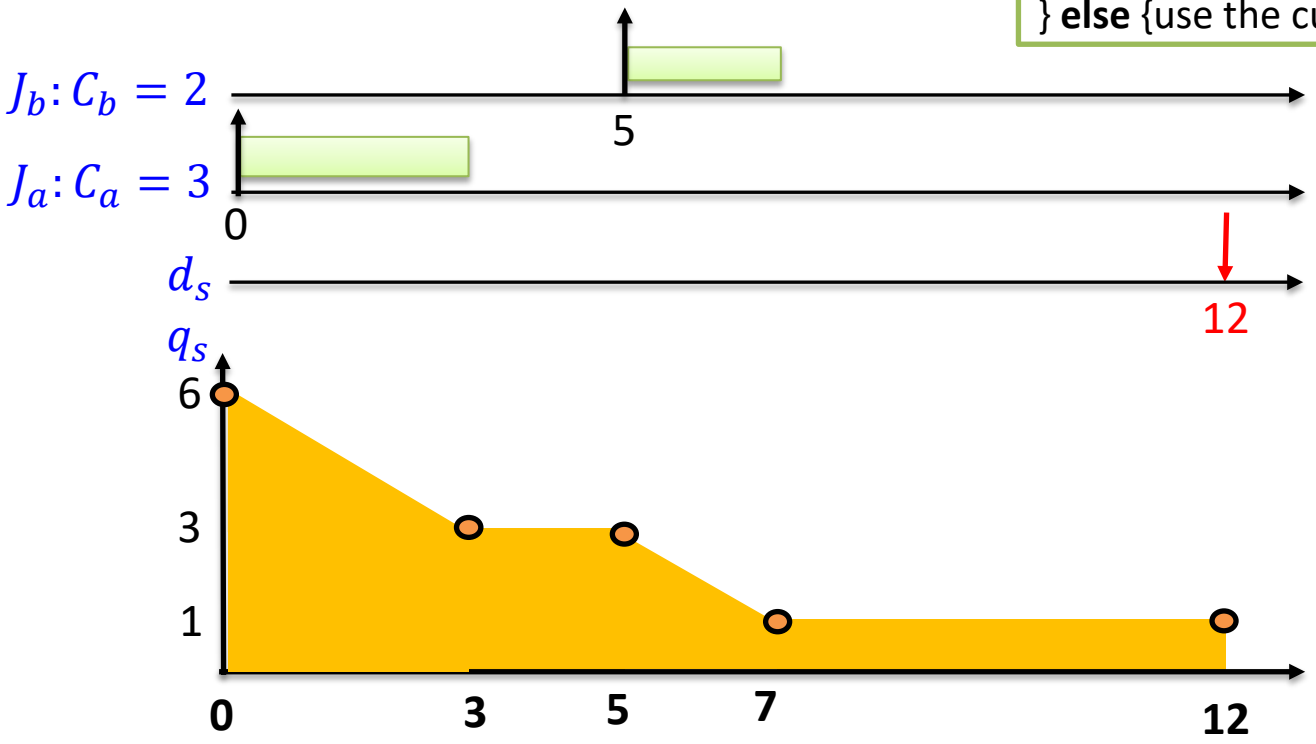
At time 3 after the budget exhausts:

What is d_s? **12**

What is q_s? **3**

CBS deadline assignment on job arrival

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}



$Q_s = 6, T_s = 12,$
$U_s = 0.5$

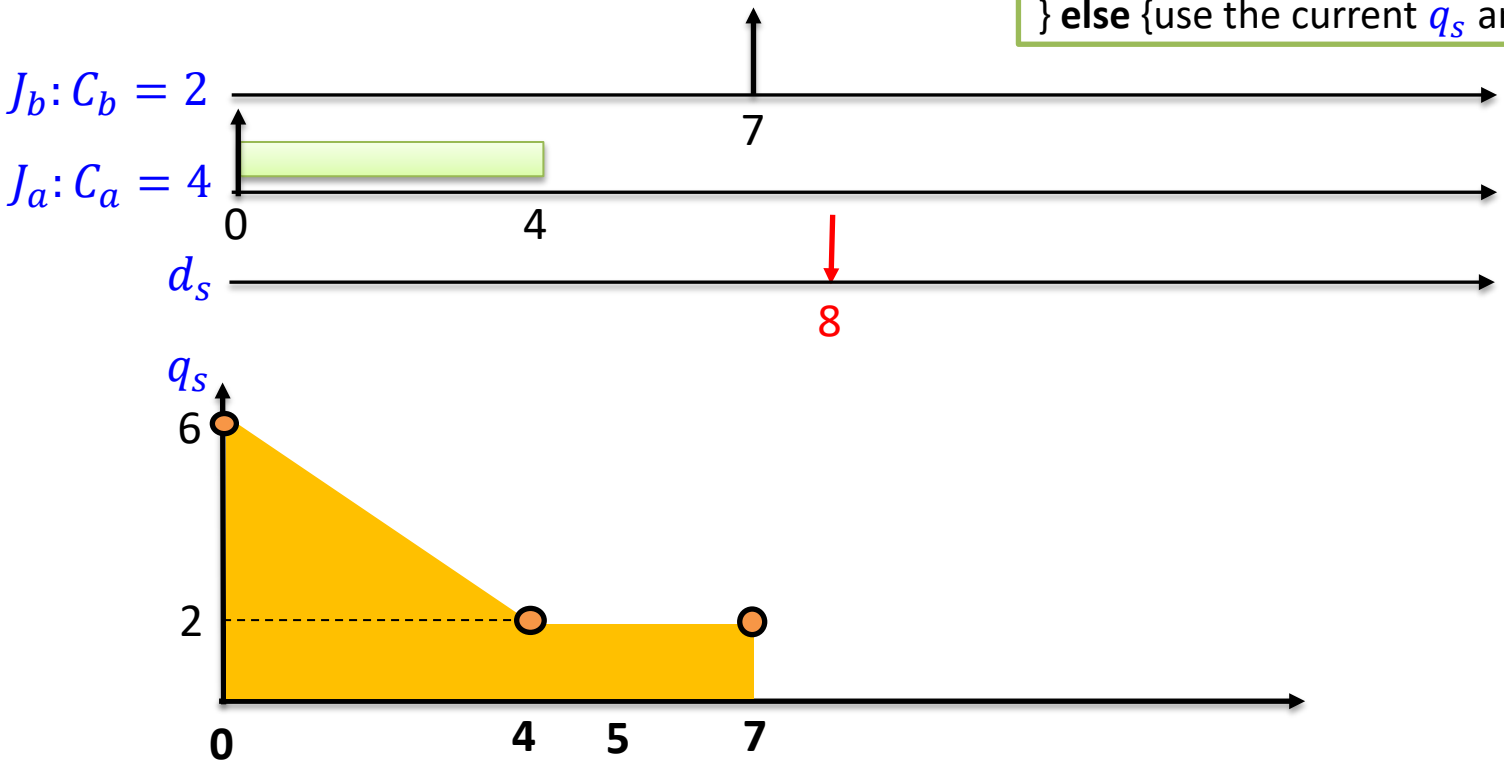
At time 0:
What is d_s? **12**
What is q_s? **6**

At time 5 after J_2 arrives:
What is d_s? **12**
What is q_s? **3**

0.5 > 3/(12-5) = 0.375

CBS deadline assignment on job arrival

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}



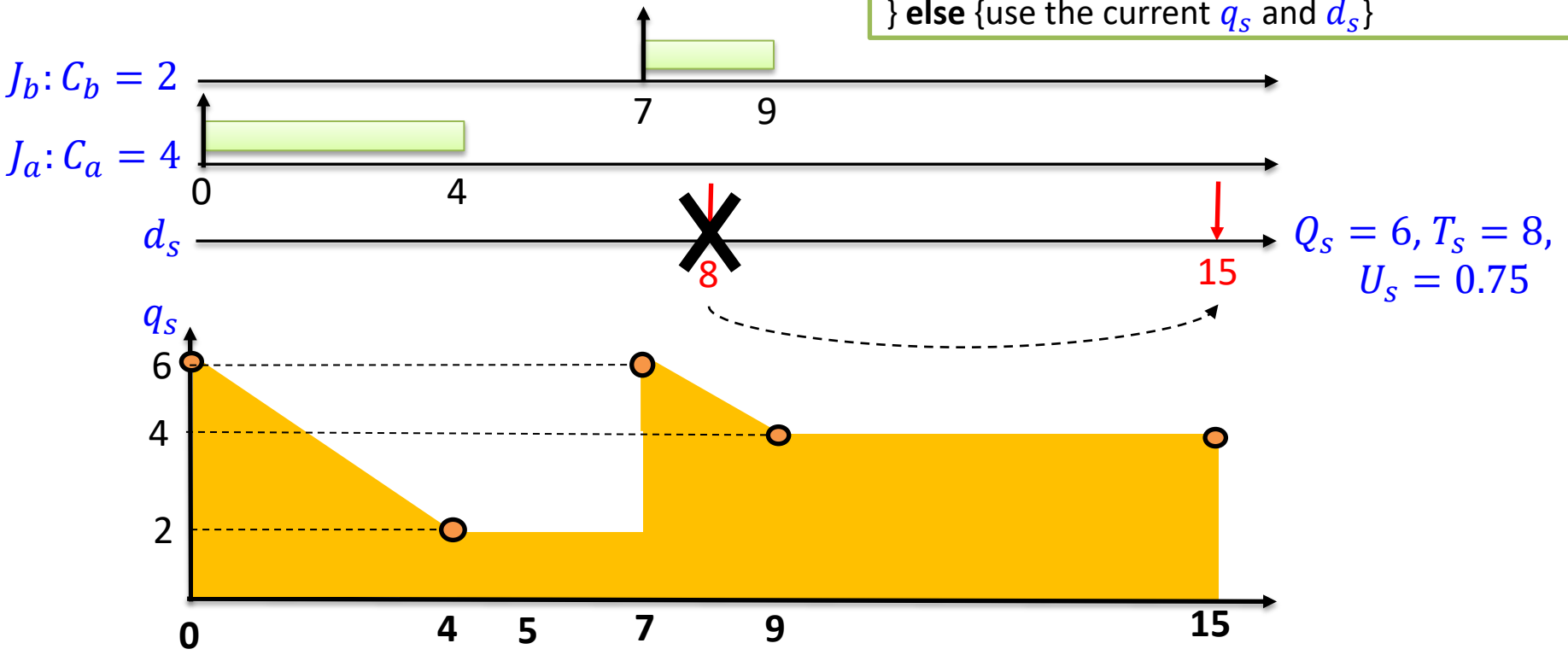
At time 0:
What is d_s? 8
What is q_s? 6

At time 7 after J_2 arrives:
What is d_s? 15 = 7 + 8
What is q_s? 6

$0.75 < 2/(8-7)$? Yes. So we need to replenish the budget

CBS deadline assignment on job arrival

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}



At time 0:
What is d_s? 8
What is q_s? 6

At time 7 after J_2 arrives:
What is d_s? 15 = 7 + 8
What is q_s? 6

Example: CBS

Aperiodic jobs:

$J_a: (r_a = 2, C_a = 3), J_b: (r_b = 12, C_b = 3)$

Server:

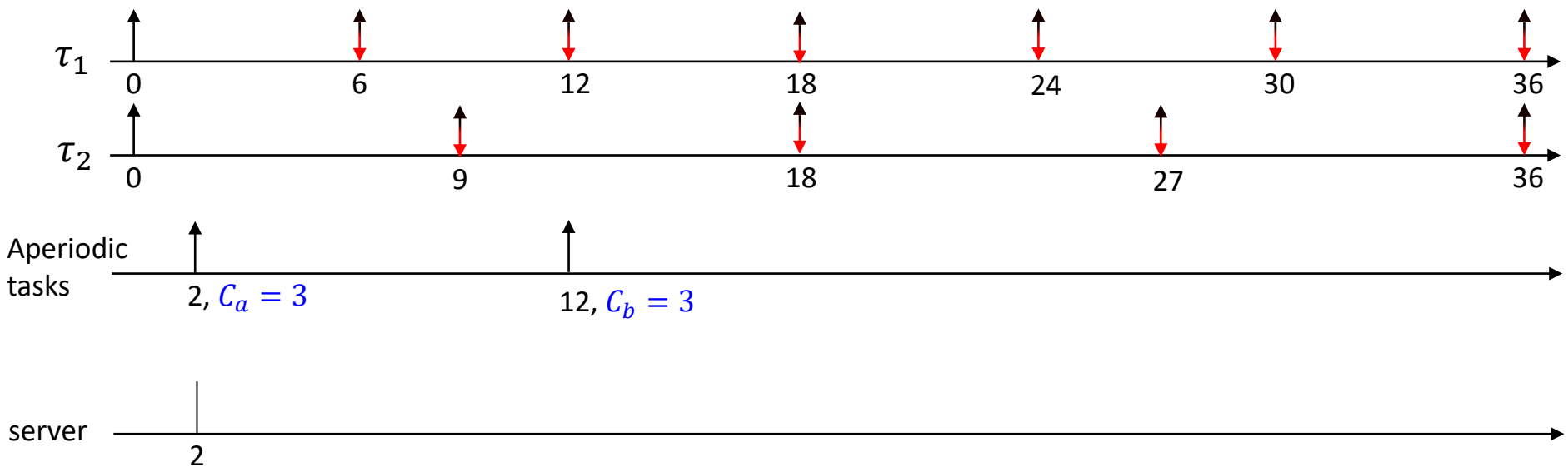
$Q_s = 2$ and $T_s = 6$

Periodic tasks:

$\tau_1: (C_1 = 2, T_1 = 6)$ and $\tau_2: (C_2 = 3, T_2 = 9)$

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}

Exhaustion:
$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$

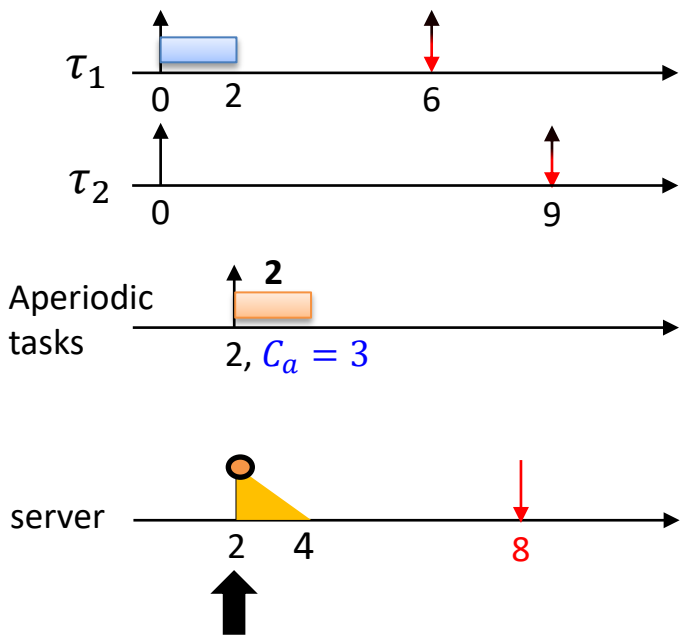


Example: CBS

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}

Exhaustion:
$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$

Aperiodic jobs:
$J_a: (r_a = 2, C_a = 3), J_b: (r_b = 12, C_b = 3)$
Server:
$Q_s = 2$ and $T_s = 6$
Periodic tasks:
$\tau_1: (C_1 = 2, T_1 = 6)$ and $\tau_2: (C_2 = 3, T_2 = 9)$



What are d_s and q_s at time 2 right after the arrival of J_a?
$d_s = 2 + 6 = 8$
$q_s = 2$

Which task will be scheduled at time 2?
J_a because its deadline is smaller than τ_2's deadline.

What happens at time 4?
The budget has exhausted. Server must be updated.

Example: CBS

Aperiodic jobs:

$J_a: (r_a = 2, C_a = 3), J_b: (r_b = 12, C_b = 3)$

Server:

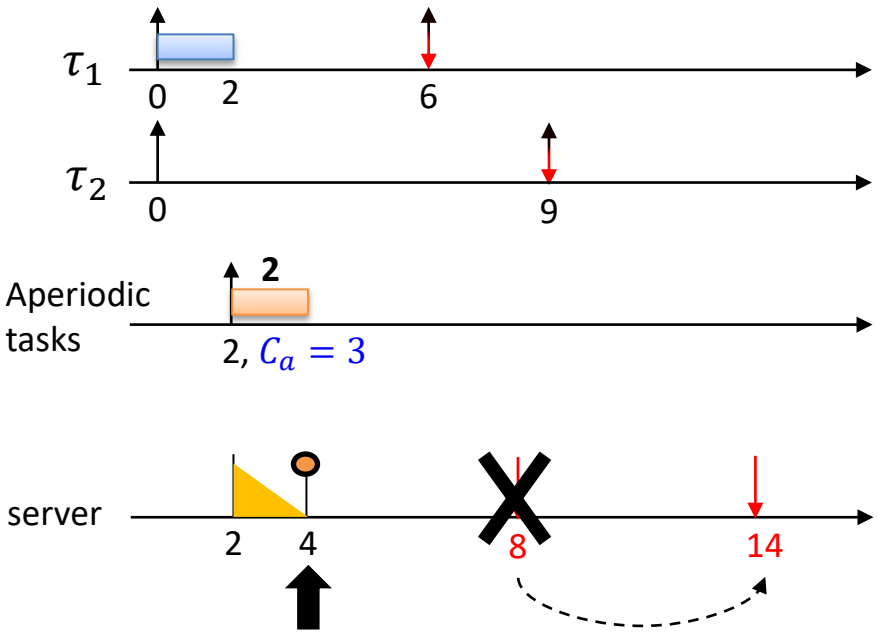
$Q_s = 2$ and $T_s = 6$

Periodic tasks:

$\tau_1: (C_1 = 2, T_1 = 6)$ and $\tau_2: (C_2 = 3, T_2 = 9)$

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}

Exhaustion:
$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$



Which task will be scheduled at time 4?

τ_2 because it has an earlier deadline than the server (i.e., $d_s > d_{2,1}$).

Example: CBS

Aperiodic jobs:

$J_a: (r_a = 2, C_a = 3), J_b: (r_b = 12, C_b = 3)$

Server:

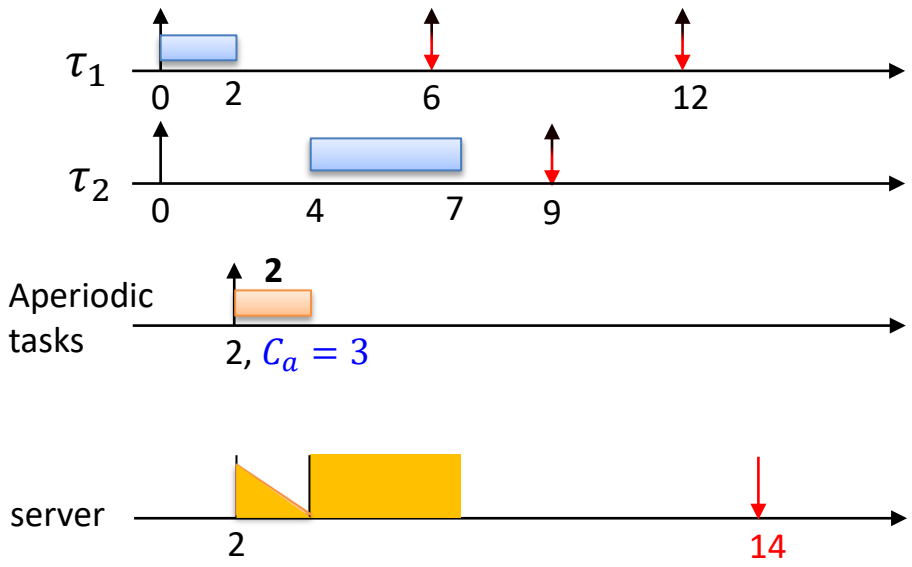
$Q_s = 2$ and $T_s = 6$

Periodic tasks:

$\tau_1: (C_1 = 2, T_1 = 6)$ and $\tau_2: (C_2 = 3, T_2 = 9)$

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}

Exhaustion:
$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$



Which task will be scheduled at time 7?

τ_1 because it has an earlier deadline than the server (i.e., $d_s > d_{1,2}$).

Example: CBS

Aperiodic jobs:

$J_a: (r_a = 2, C_a = 3), J_b: (r_b = 12, C_b = 3)$

Server:

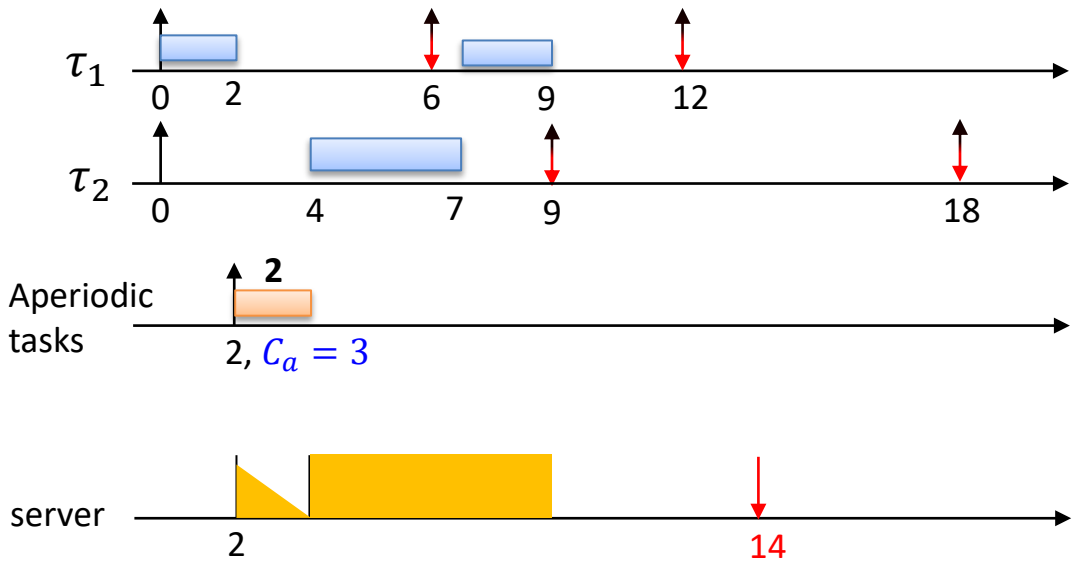
$Q_s = 2$ and $T_s = 6$

Periodic tasks:

$\tau_1: (C_1 = 2, T_1 = 6)$ and $\tau_2: (C_2 = 3, T_2 = 9)$

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}

Exhaustion:
$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$



Which task will be scheduled at time 9?

J_a because it has an earlier deadline than τ_2 (i.e., $d_s < d_{2,2}$).

Until when J_a will be scheduled using the current server parameters?

Until time 10 because it has only 1 units of remained execution time.

Example: CBS

Aperiodic jobs:

$J_a: (r_a = 2, C_a = 3), J_b: (r_b = 12, C_b = 3)$

Server:

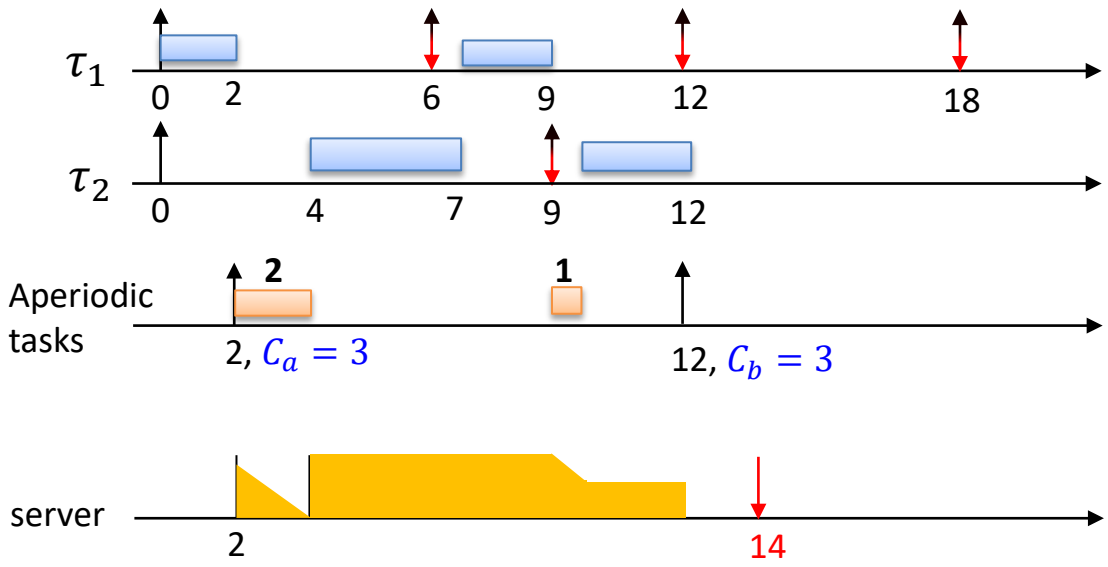
$Q_s = 2$ and $T_s = 6$

Periodic tasks:

$\tau_1: (C_1 = 2, T_1 = 6)$ and $\tau_2: (C_2 = 3, T_2 = 9)$

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}

Exhaustion:
$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$



Example: CBS

Aperiodic jobs:

$J_a: (r_a = 2, C_a = 3), J_b: (r_b = 12, C_b = 3)$

Server:

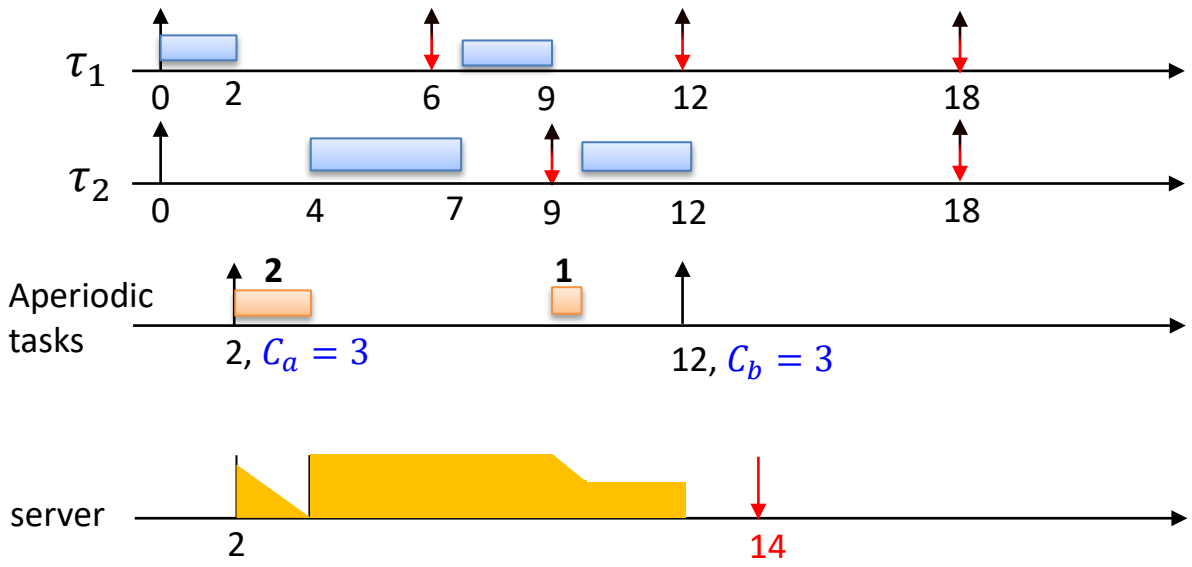
$Q_s = 2$ and $T_s = 6$

Periodic tasks:

$\tau_1: (C_1 = 2, T_1 = 6)$ and $\tau_2: (C_2 = 3, T_2 = 9)$

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}

Exhaustion:
$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$



What are the parameters of the server after the arrival of J_b?

$d_s = 12 + 6 = 18$
$q_s = 2$
Because at time 12, we have 1/(14-12) = 0.5 > 0.33
Hence, we need to replenish the server.

Example: CBS

Aperiodic jobs:

$J_a: (r_a = 2, C_a = 3), J_b: (r_b = 12, C_b = 3)$

Server:

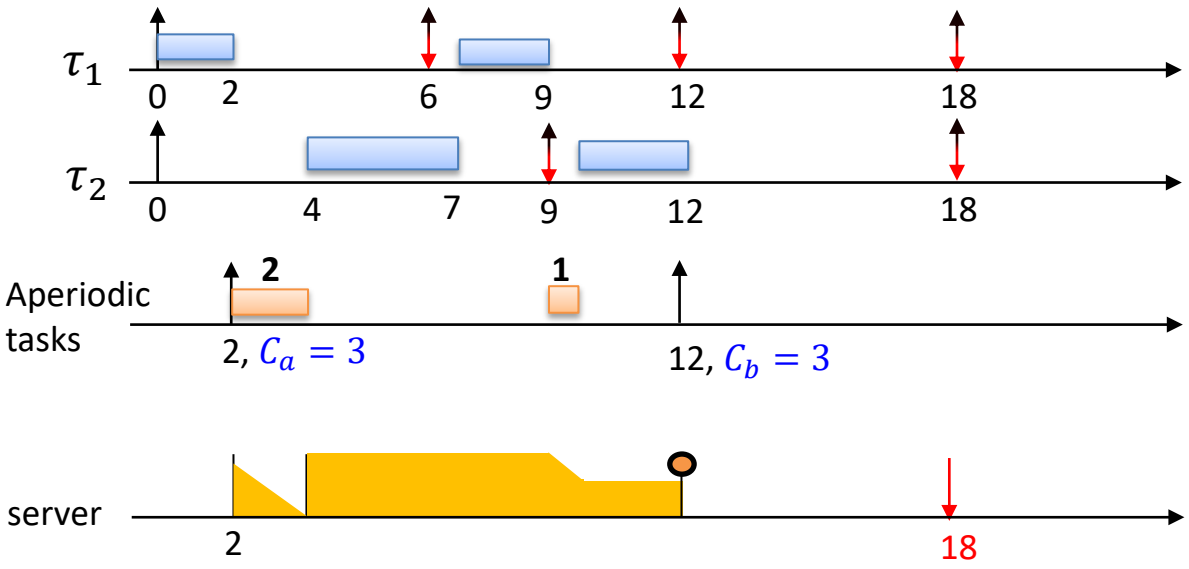
$Q_s = 2$ and $T_s = 6$

Periodic tasks:

$\tau_1: (C_1 = 2, T_1 = 6)$ and $\tau_2: (C_2 = 3, T_2 = 9)$

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}

Exhaustion:
$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$



Which job will be scheduled at time 12?

J_b because the server wins all ties on deadlines

Example: CBS

Aperiodic jobs:

$J_a: (r_a = 2, C_a = 3), J_b: (r_b = 12, C_b = 3)$

Server:

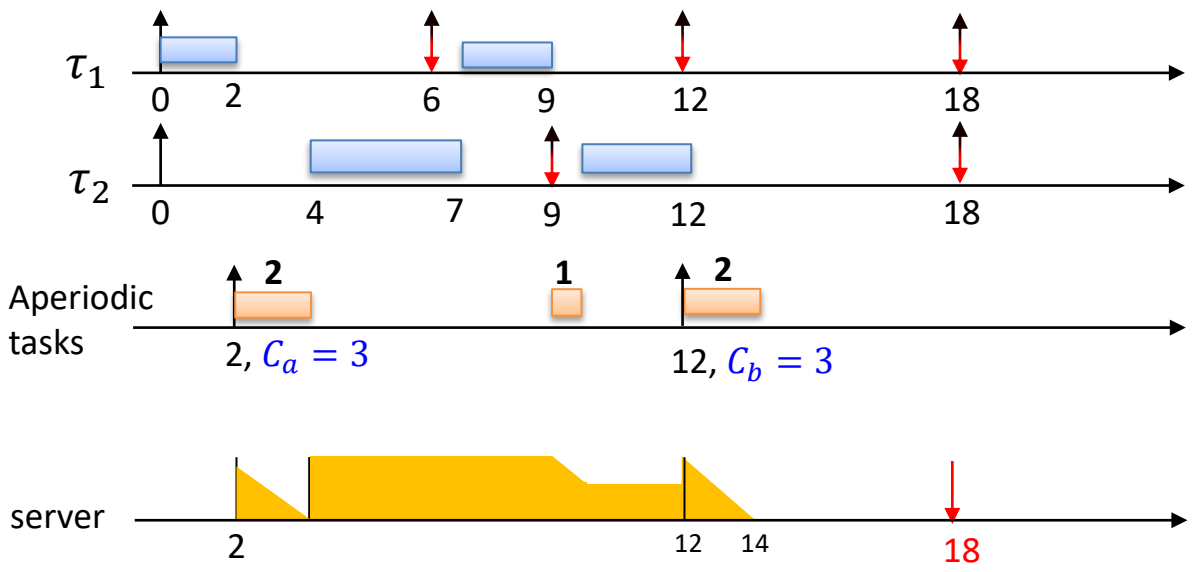
$Q_s = 2$ and $T_s = 6$

Periodic tasks:

$\tau_1: (C_1 = 2, T_1 = 6)$ and $\tau_2: (C_2 = 3, T_2 = 9)$

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}

Exhaustion:
$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$



What are the parameters of the server after being exhausted at time 14?

$d_s = 18 + 6 = 24$
$q_s = 2$

Example: CBS

Aperiodic jobs:

$J_a: (r_a = 2, C_a = 3), J_b: (r_b = 12, C_b = 3)$

Server:

$Q_s = 2$ and $T_s = 6$

Periodic tasks:

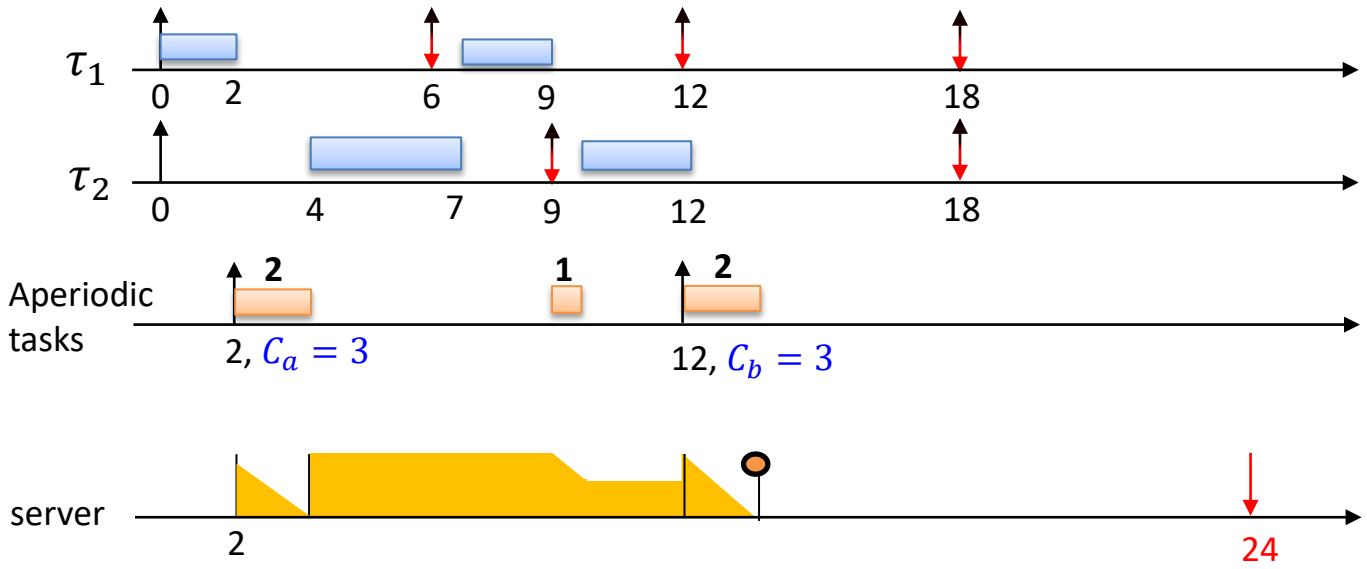
$\tau_1: (C_1 = 2, T_1 = 6)$ and $\tau_2: (C_2 = 3, T_2 = 9)$

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}

Exhaustion:
$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$

Which job will be scheduled at time 14?

$J_{1,3}$ because it has an earlier deadline and then $J_{2,2}$

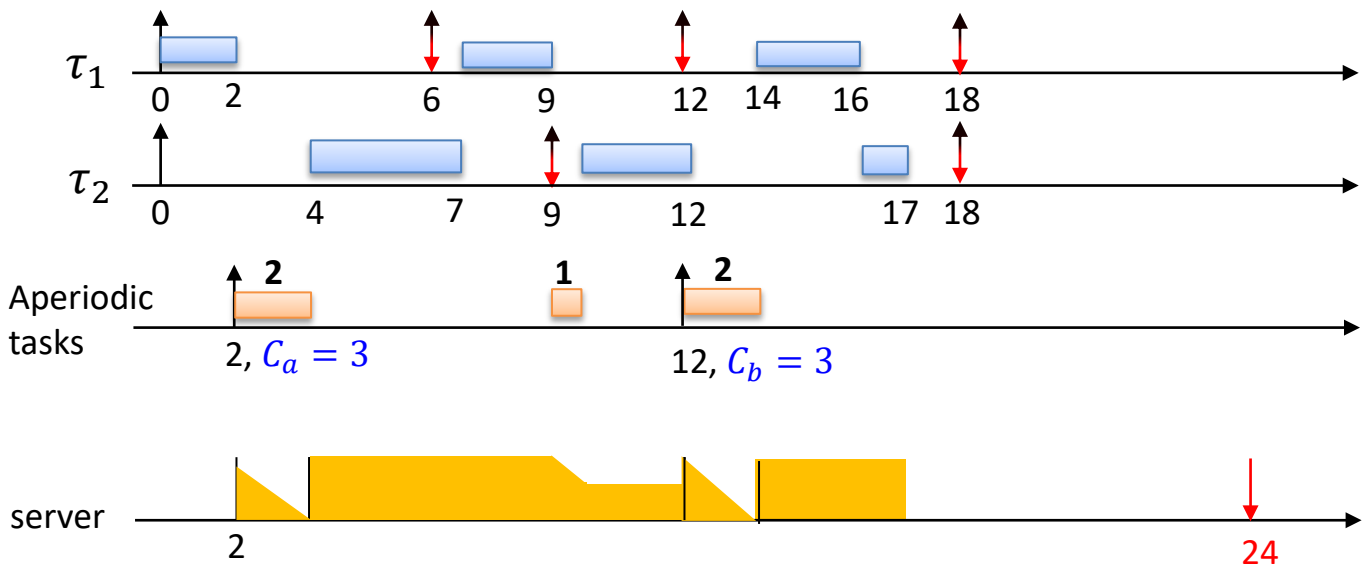


Example: CBS

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}

Exhaustion:
$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$

Aperiodic jobs:
$J_a: (r_a = 2, C_a = 3), J_b: (r_b = 12, C_b = 3)$
Server:
$Q_s = 2$ and $T_s = 6$
Periodic tasks:
$\tau_1: (C_1 = 2, T_1 = 6)$ and $\tau_2: (C_2 = 3, T_2 = 9)$



Example: CBS

Aperiodic jobs:

$J_a: (r_a = 2, C_a = 3), J_b: (r_b = 12, C_b = 3)$

Server:

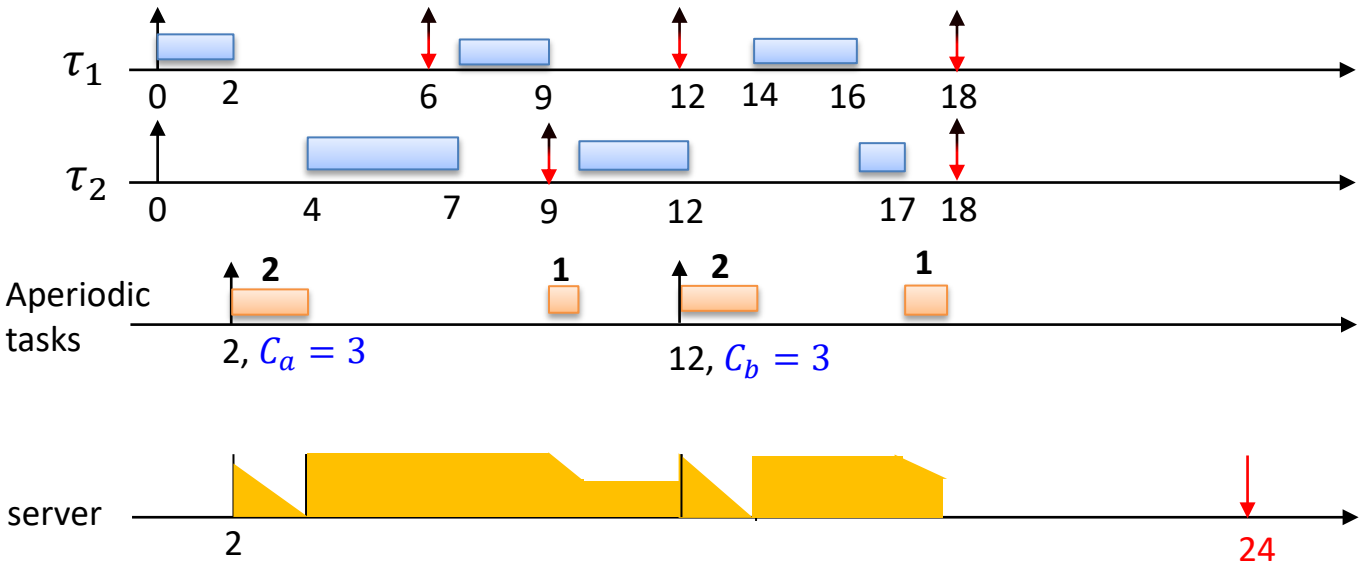
$Q_s = 2$ and $T_s = 6$

Periodic tasks:

$\tau_1: (C_1 = 2, T_1 = 6)$ and $\tau_2: (C_2 = 3, T_2 = 9)$

If (∃ a pending aperiodic job) **then** <enqueue J_k>
else if ($U_s < \frac{q_s}{d_s - r_k}$) **then** {
$q_s \leftarrow Q_s$
$d_s \leftarrow r_k + T_s$
} **else** {use the current q_s and d_s}

Exhaustion:
$q_s \leftarrow Q_s$
$d_s \leftarrow d_s + T_s$



Summary

- **Server-based scheduling:** a remedy for aperiodic events, background work, and misbehaving real-time tasks

- **Types of servers**
  - Periodic servers
  - Polling servers
  - Deferrable servers
  - Constant-bandwidth servers (CBS)