APSys 2024 – Kyoto, Japan – 4th September

Developing Process Scheduling Policies in User Space with Common OS Features

Kenichi Yasukata
IIJ Research Laboratory

Kenta Ishiguro
Keio University

Process Scheduling

- Process scheduling is one of the keys to multiprogramming where a CPU core runs multiple programs concurrently

Process Scheduling

- Process scheduling is one of the keys to multiprogramming where a CPU core runs multiple programs concurrently

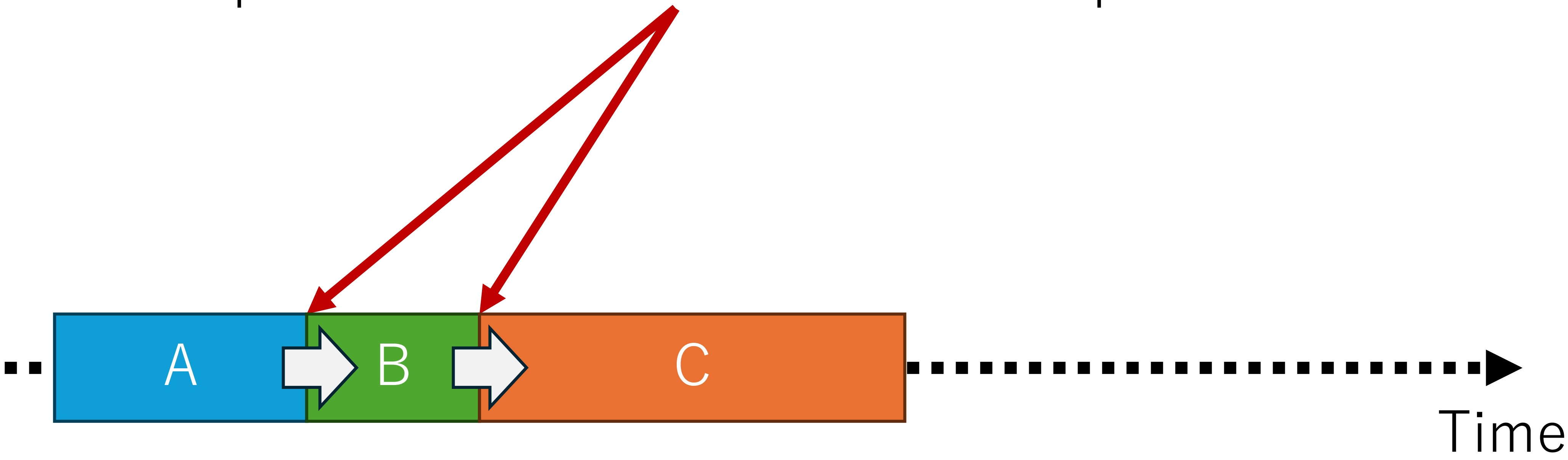
Time

Process Scheduling

- Process scheduling is one of the keys to multiprogramming where a CPU core runs multiple programs concurrently

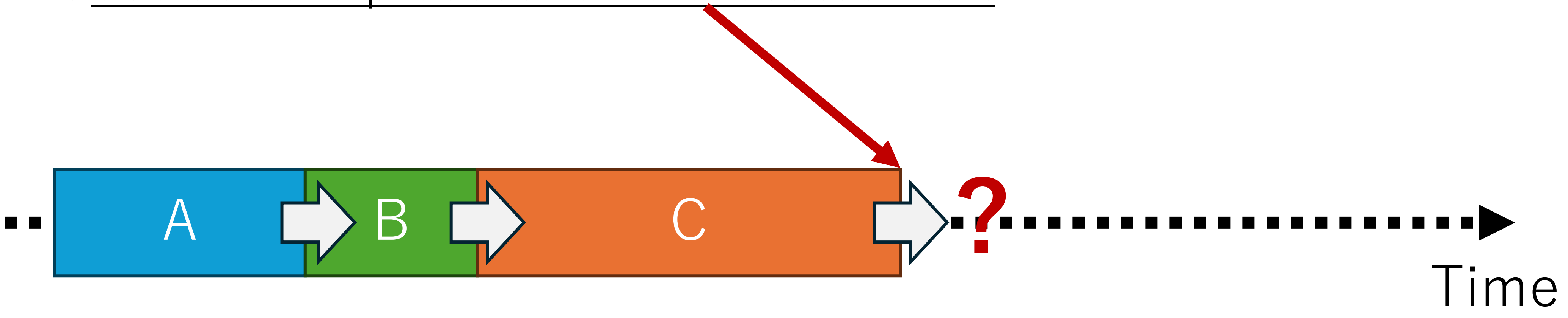
Process Scheduling

- Process scheduling is one of the keys to multiprogramming where a CPU core runs multiple programs concurrently
- Executed processes are switched at some point



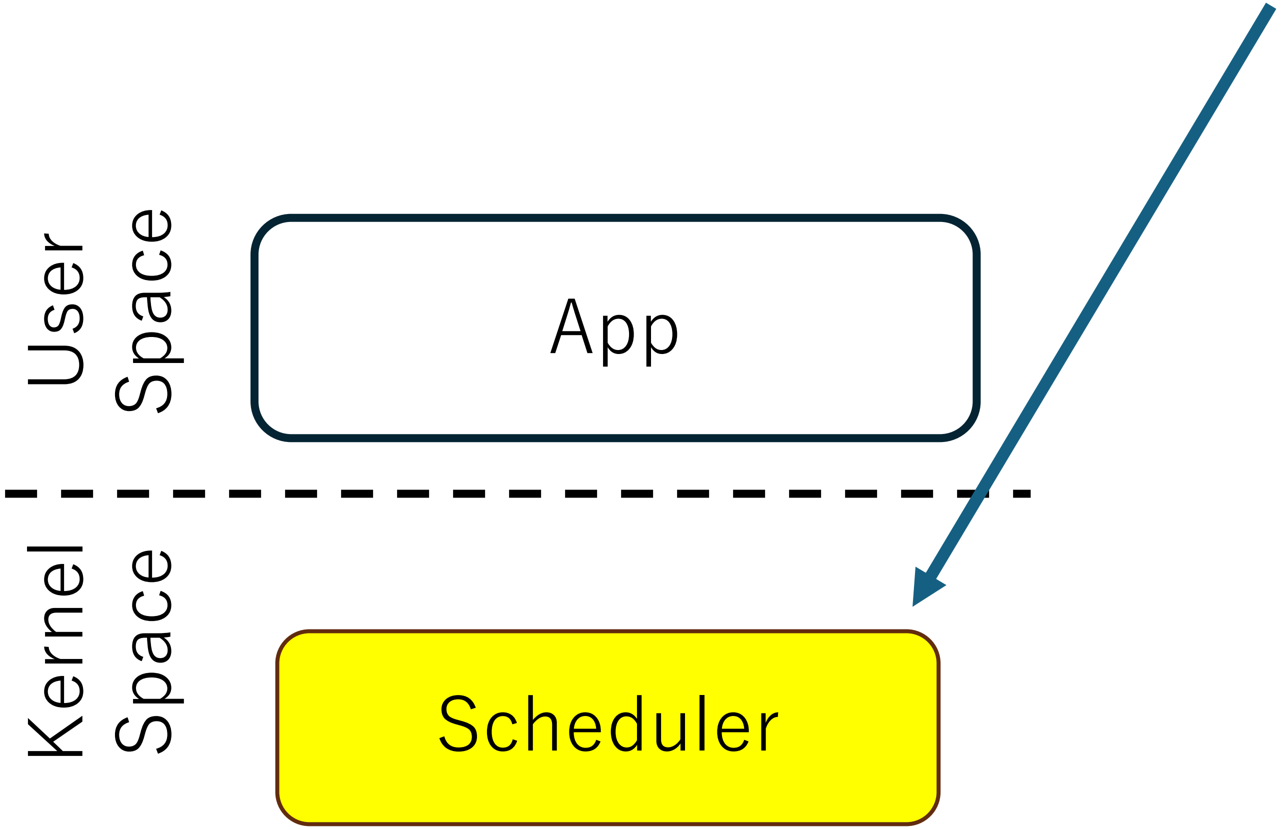
Process Scheduling

- Process scheduling is one of the keys to multiprogramming where a CPU core runs multiple programs concurrently
- Executed processes are switched at some point
- A process scheduler makes a scheduling decision
  - It decides the process to be executed next



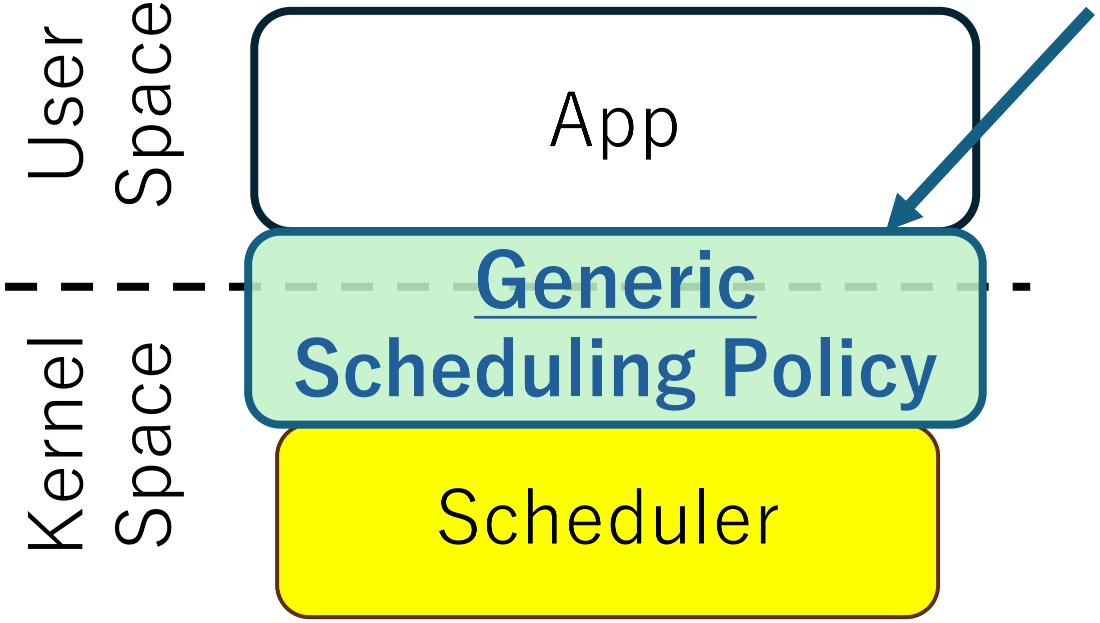
Process Scheduling

- Process schedulers and their scheduling policies have been typically implemented as part of OS kernels



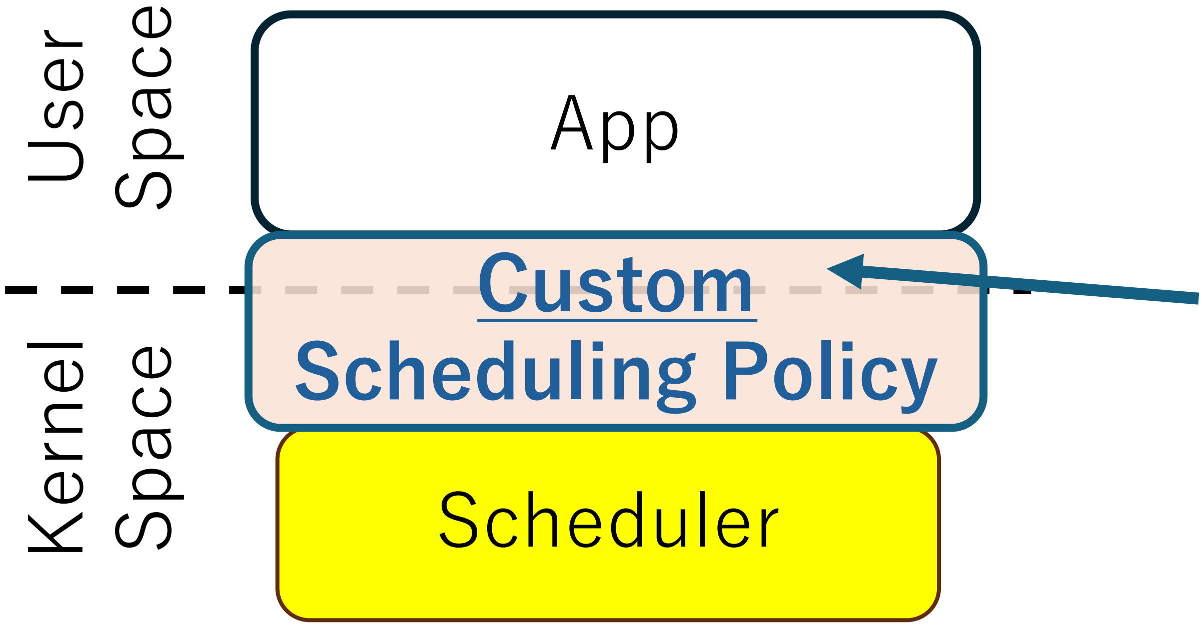
Process Scheduling

- Process schedulers and their scheduling policies have been typically implemented as part of OS kernels
- Their goal is generality enabling a wide range of applications to achieve not the best but good enough performance



Process Scheduling

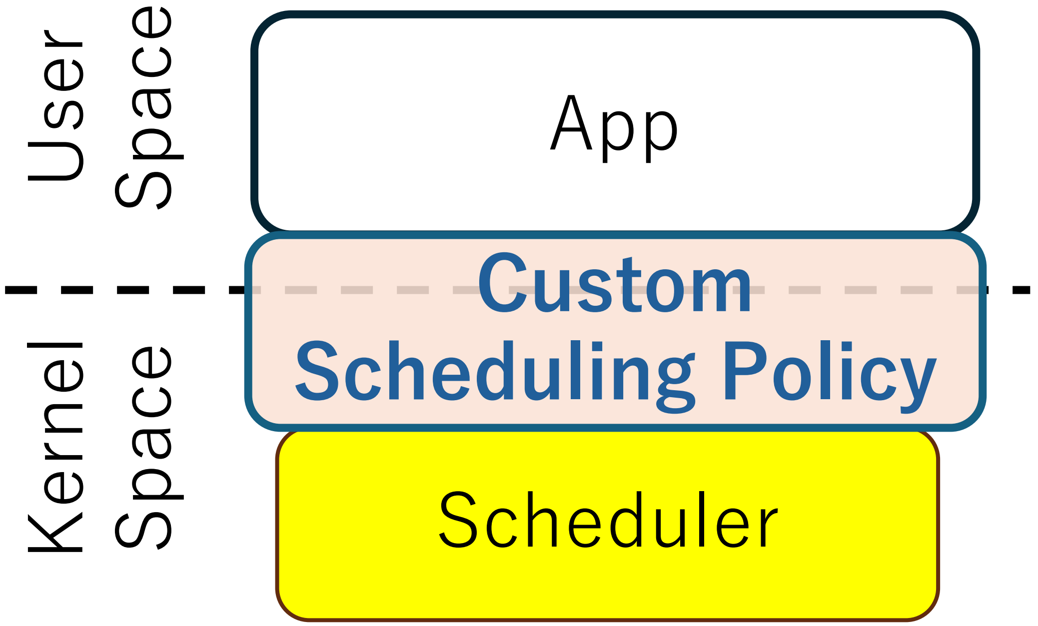
- Process schedulers and their scheduling policies have been typically implemented as part of OS kernels
- Their goal is generality enabling a wide range of applications to achieve not the best but good enough performance



On the other hand, previous studies showed that custom scheduling policies contribute to application performance

Problem

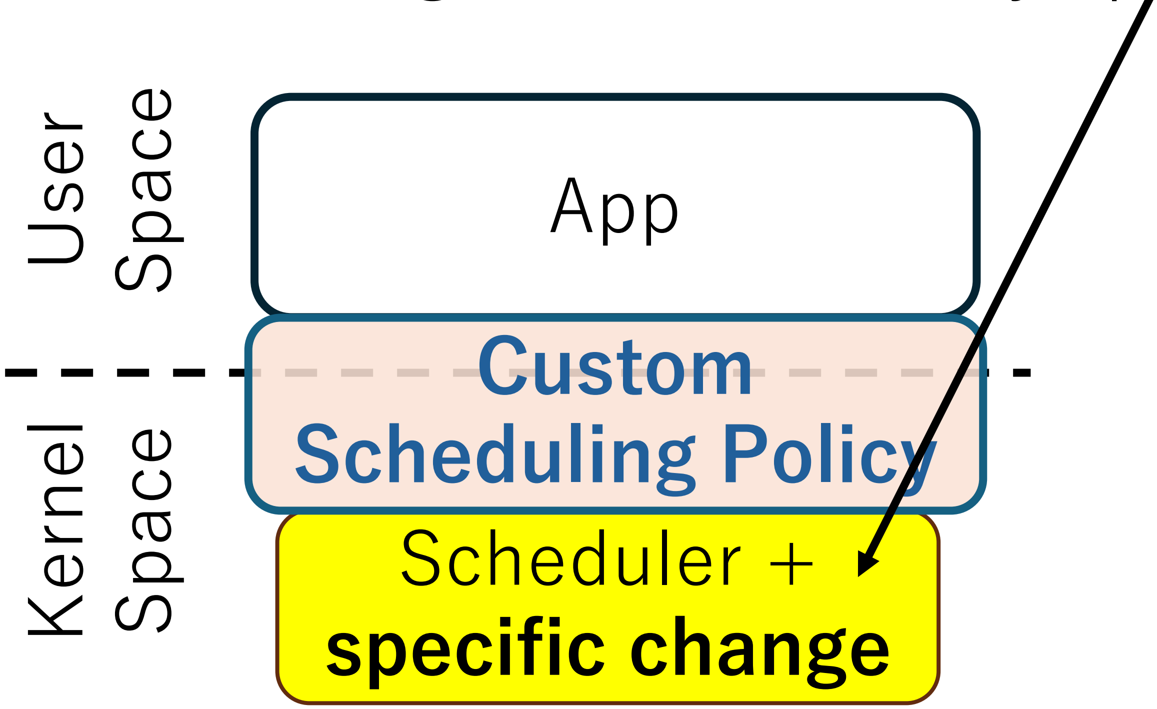
- Despite their benefits, it is hard to develop and deploy custom process scheduling policies



Related Work (1/4)

- Despite their benefits, it is hard to develop and deploy custom process scheduling policies

Scheduling enhancement by specific kernel/hypervisor extensions

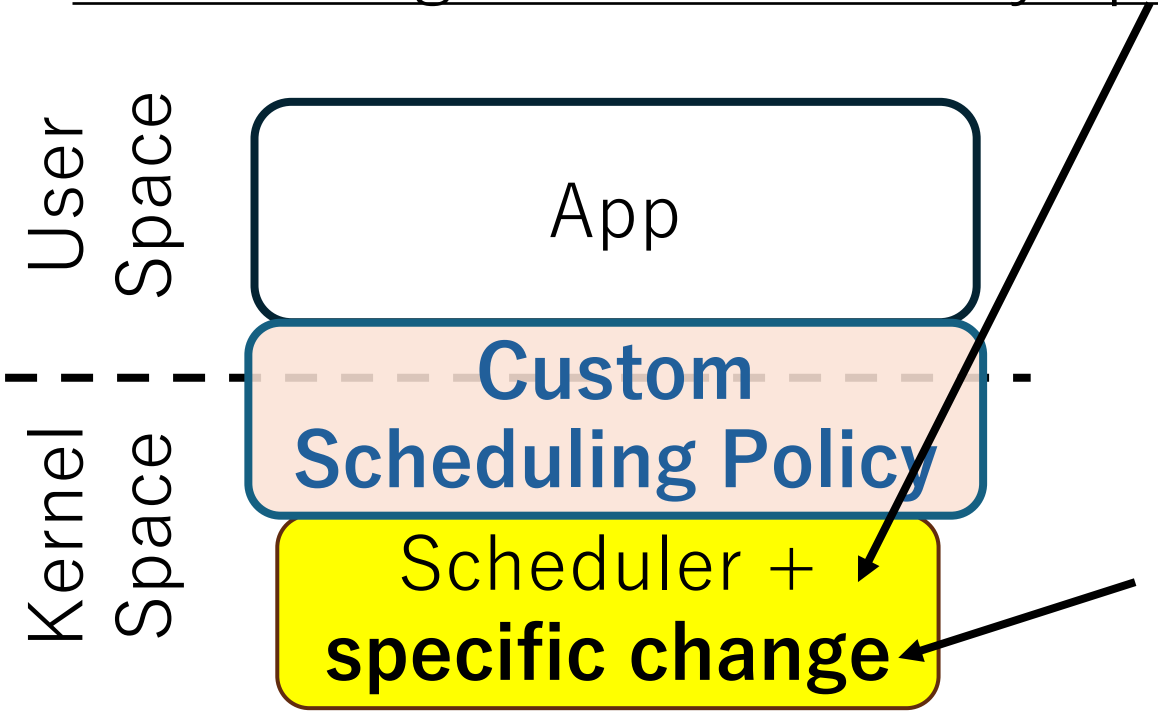


e.g., vTurbo (ATC'13), Tableau (EuroSys'18), Shinjuku (NSDI'19), Caladan (OSDI'20)

Related Work (1/4)

- Despite their benefits, it is hard to develop and deploy custom process scheduling policies

Scheduling enhancement by specific kernel/hypervisor extensions



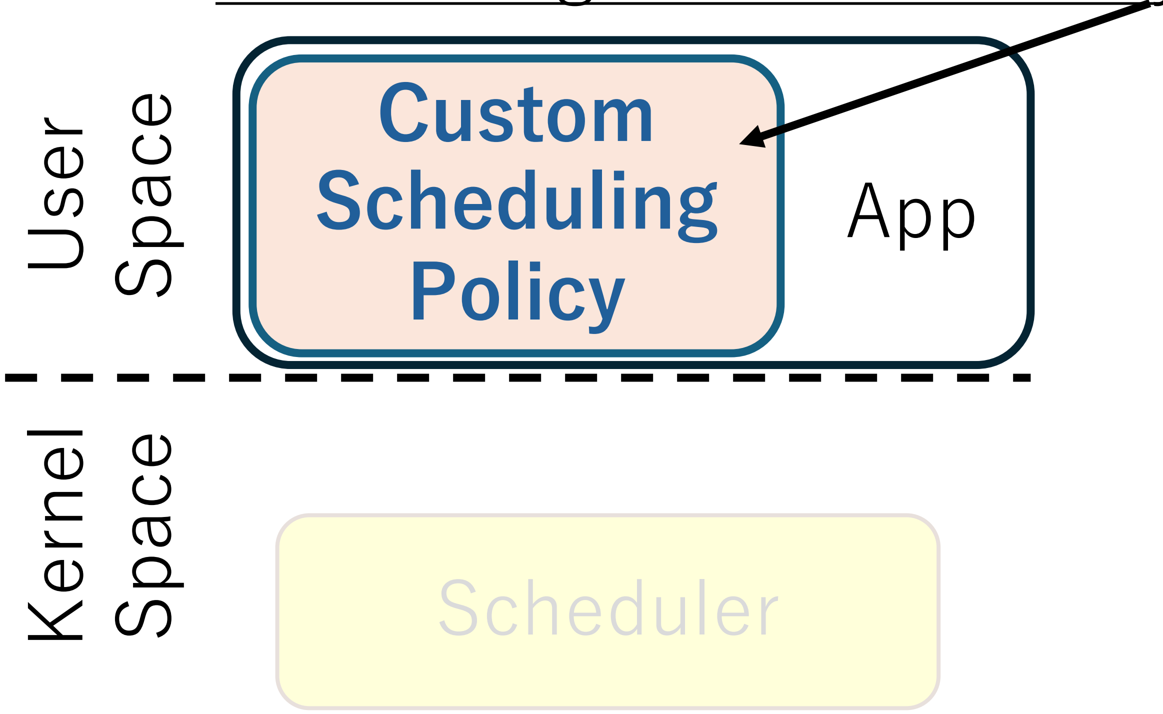
e.g., vTurbo (ATC'13), Tableau (EuroSys'18), Shinjuku (NSDI'19), Caladan (OSDI'20)

It is hard for users to deploy them because of concerns for security, stability, and future maintenance

Related Work (2/4)

- Despite their benefits, it is hard to develop and deploy custom process scheduling policies

Scheduling enhancement by specific user-space runtimes

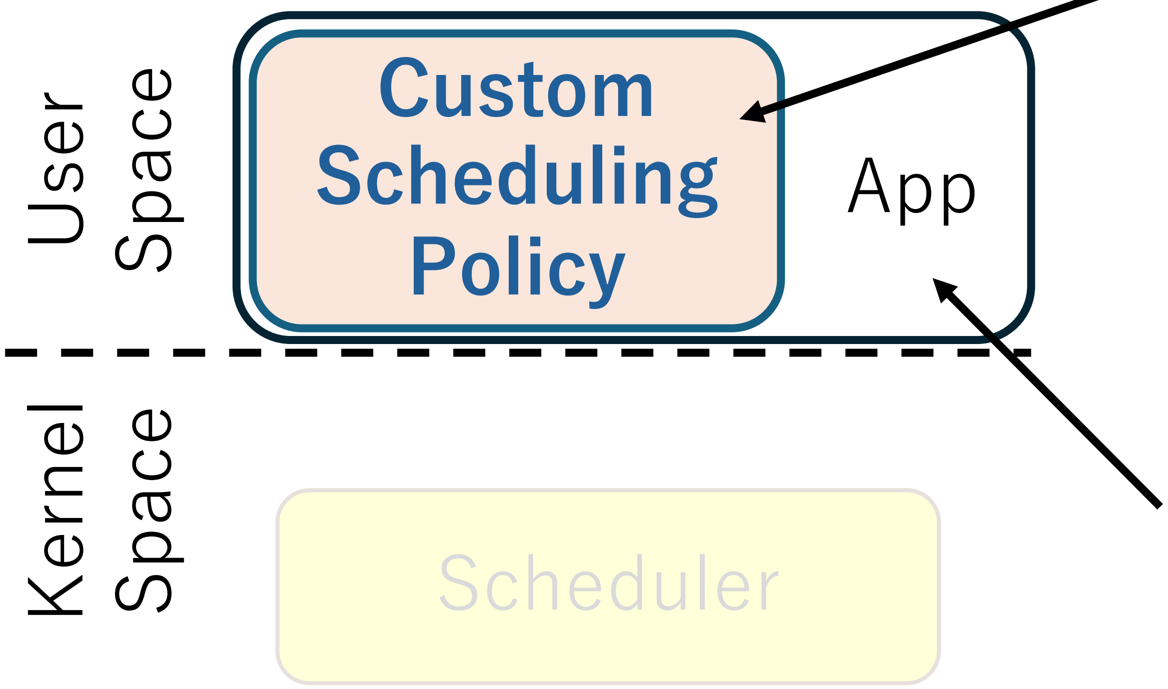


e.g., Arachne (OSDI'18), Shenango (NSDI'19), Concord (SOSP'23)

Related Work (2/4)

- Despite their benefits, it is hard to develop and deploy custom process scheduling policies

Scheduling enhancement by specific user-space runtimes

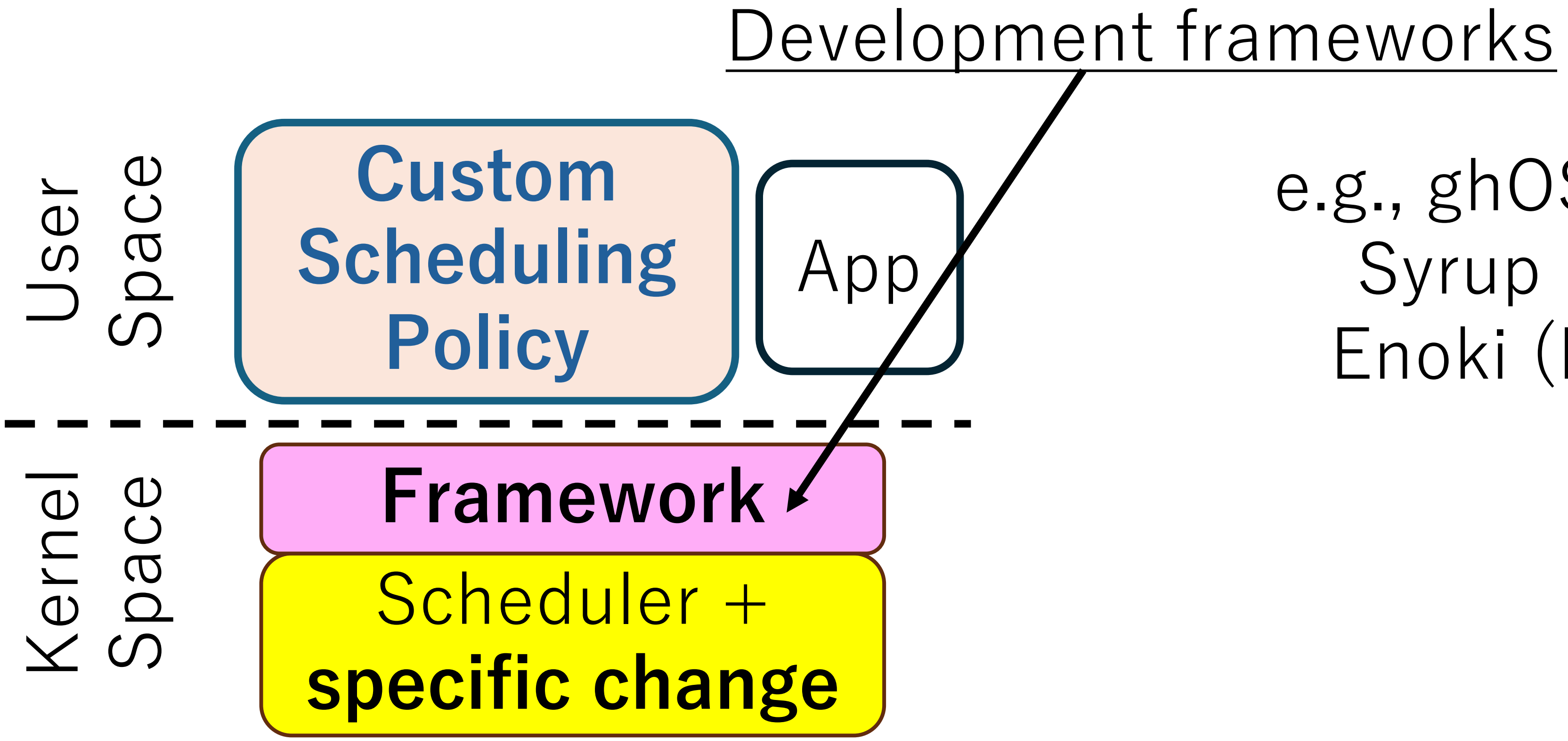


e.g., Arachne (OSDI'18), Shenango (NSDI'19), Concord (SOSP'23)

It is hard to employ them because applications need to directly involve the specific user-space runtimes

Related Work (3/4)

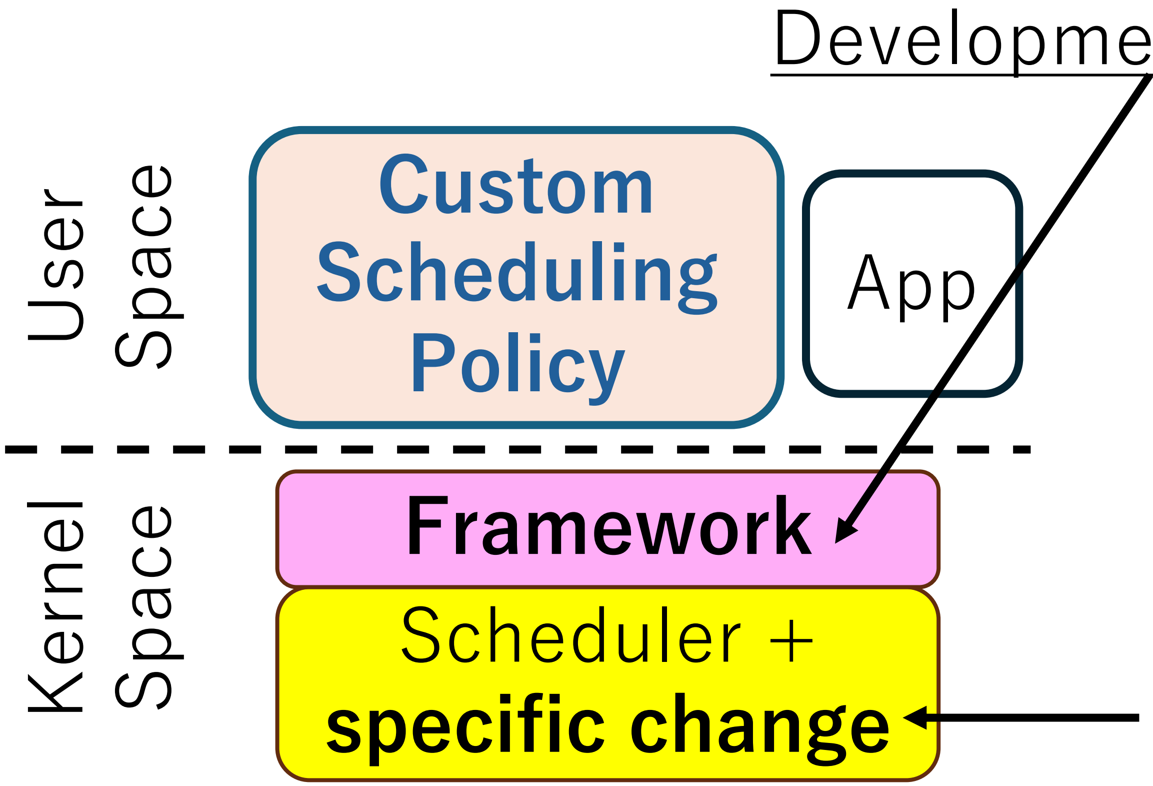
- Despite their benefits, it is hard to develop and deploy custom process scheduling policies



e.g., ghOSt (SOSP'21),
Syrup (SOSP'21),
Enoki (EuroSys'24)

Related Work (3/4)

- Despite their benefits, it is hard to develop and deploy custom process scheduling policies



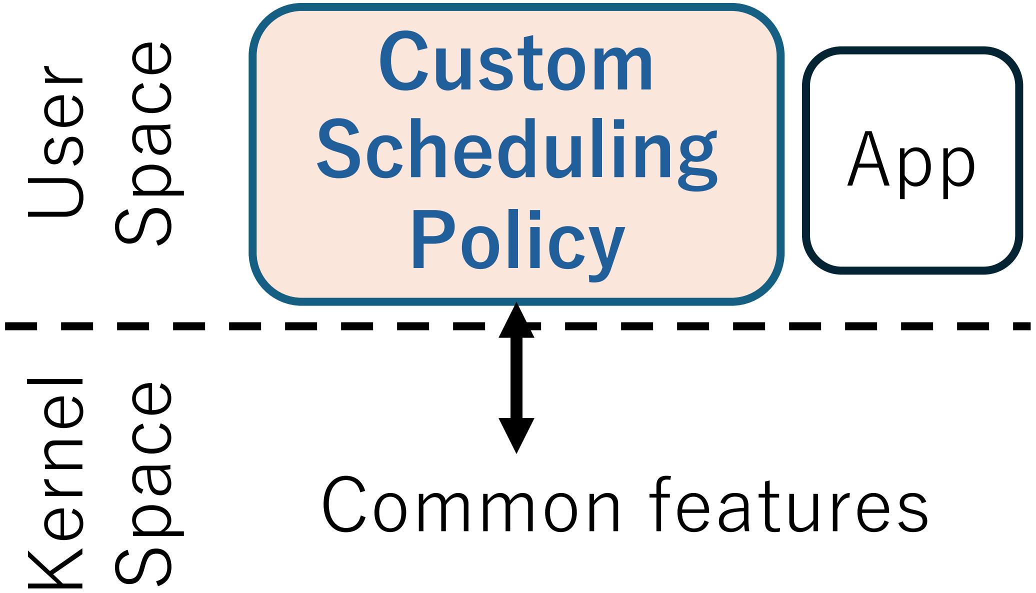
e.g., ghOSt (SOSP'21), Syrup (SOSP'21), Enoki (EuroSys'24)

It is hard to deploy systems made on them because the frameworks rely on specific kernel extensions

Related Work (4/4)

- Despite their benefits, it is hard to develop and deploy custom process scheduling policies

<u>Using common OS features for scheduling policy development</u>

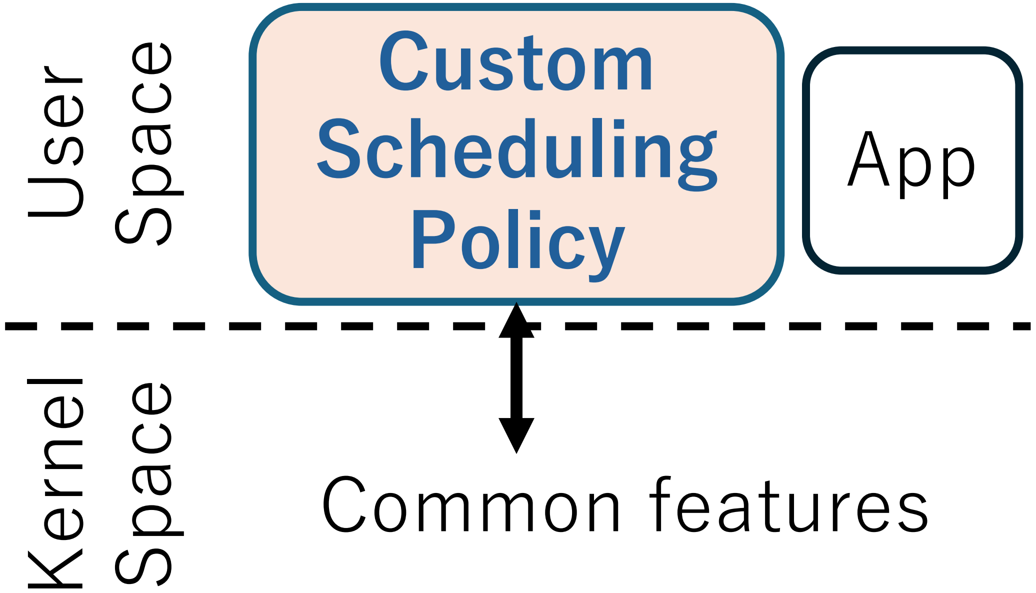


e.g., Lachesis (Middleware'21), SFS (SC'22)

Related Work (4/4)

- Despite their benefits, it is hard to develop and deploy custom process scheduling policies

<u>Using common OS features for scheduling policy development</u>



e.g., Lachesis (Middleware'21), SFS (SC'22)

They are for stream processing and serverless computing platforms, and not flexible enough to implement complicated scheduling policies

This Work

- We present a mechanism called the priority elevation trick

This Work

- We present a mechanism called the priority elevation trick

The priority elevation trick

This Work

- We present a mechanism called the priority elevation trick

The priority elevation trick

- enables flexible scheduling policy development in user space

This Work

- We present a mechanism called the priority elevation trick

The priority elevation trick

- enables flexible scheduling policy development in user space
- by only using common OS features

This Work

- We present a mechanism called the priority elevation trick

The priority elevation trick

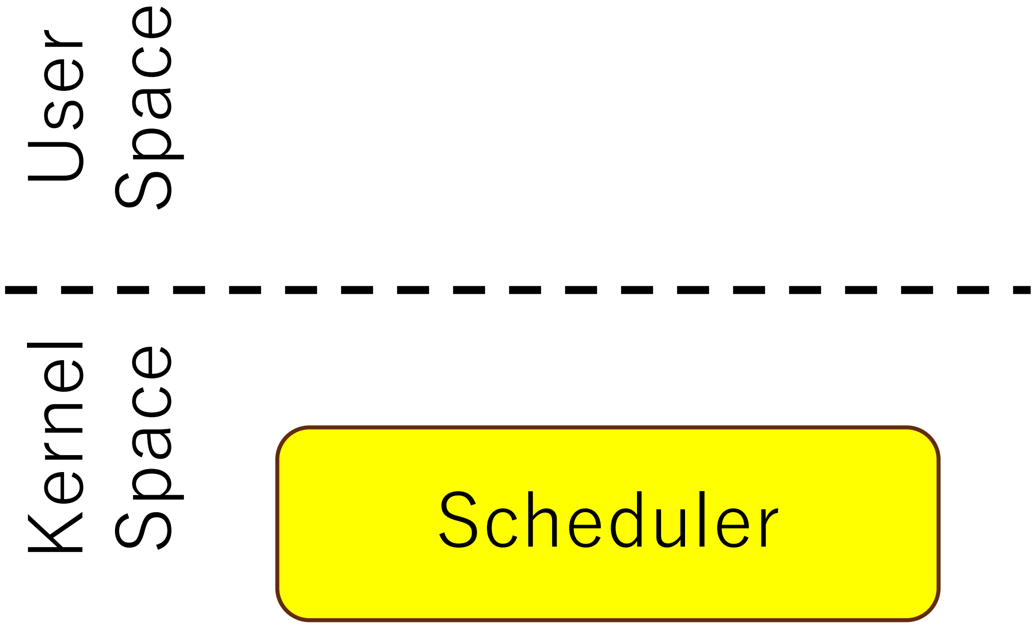
- enables flexible scheduling policy development in user space
- by only using common OS features
- without necessarily relying on a specific user-space runtime

Key Idea

- A kernel-space process scheduler normally gives a longer execution time to a process having a higher priority

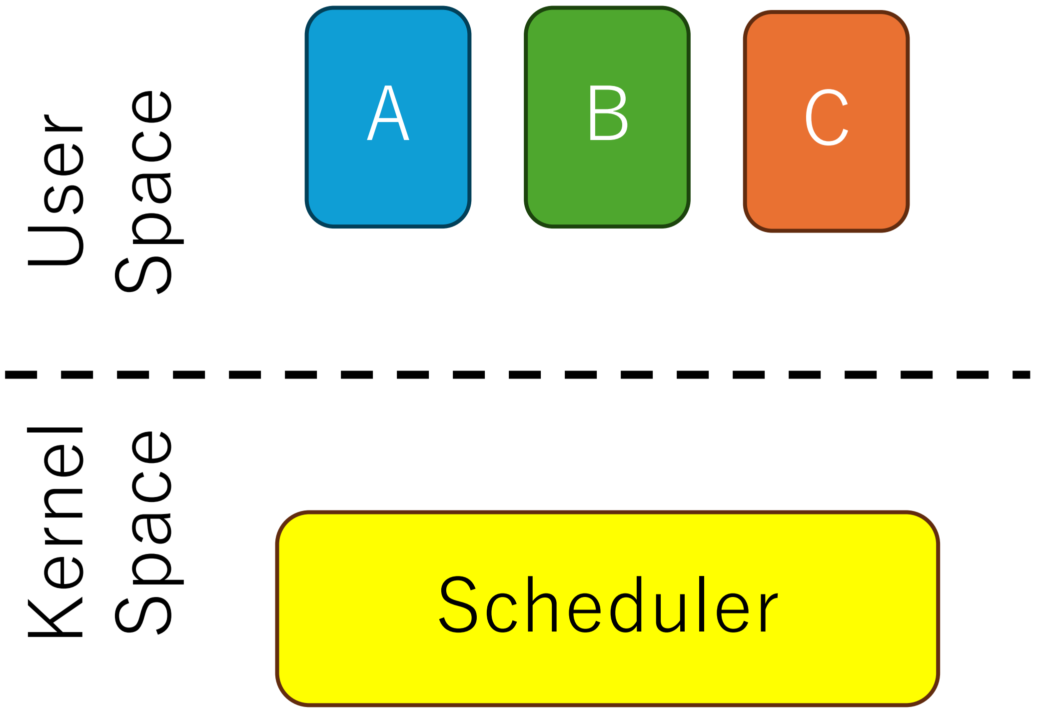
Key Idea

- A kernel-space process scheduler normally gives a longer execution time to a process having a higher priority



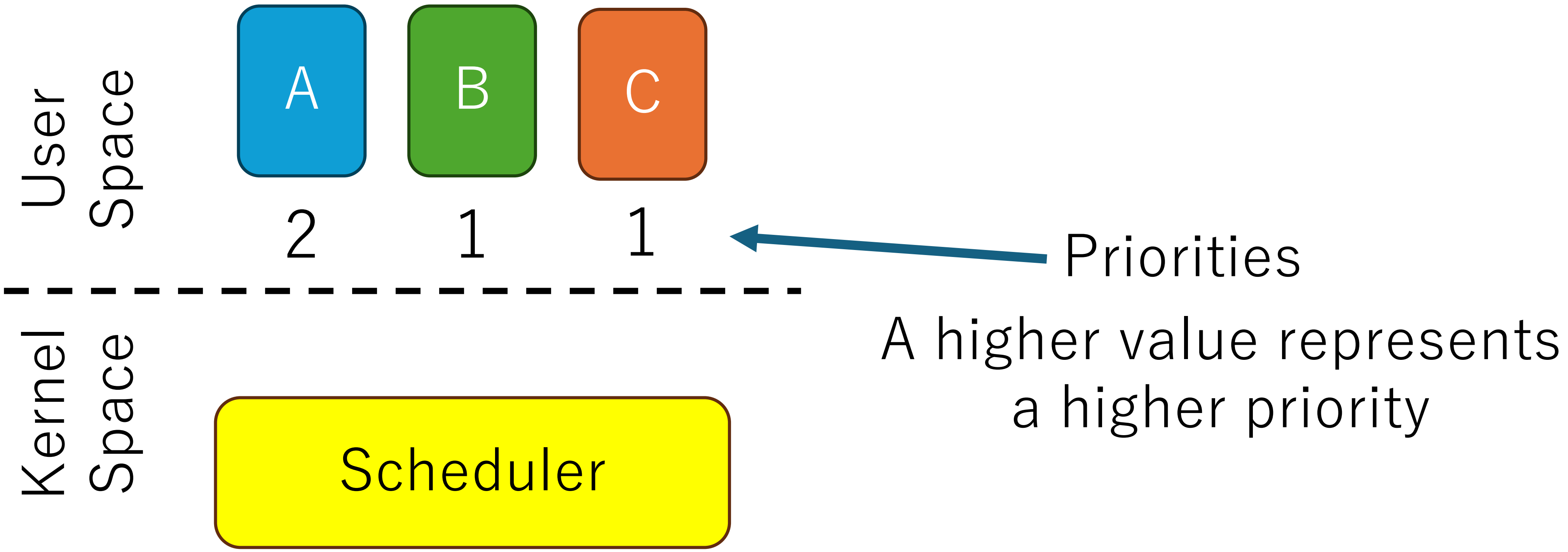
Key Idea

- A kernel-space process scheduler normally gives a longer execution time to a process having a higher priority



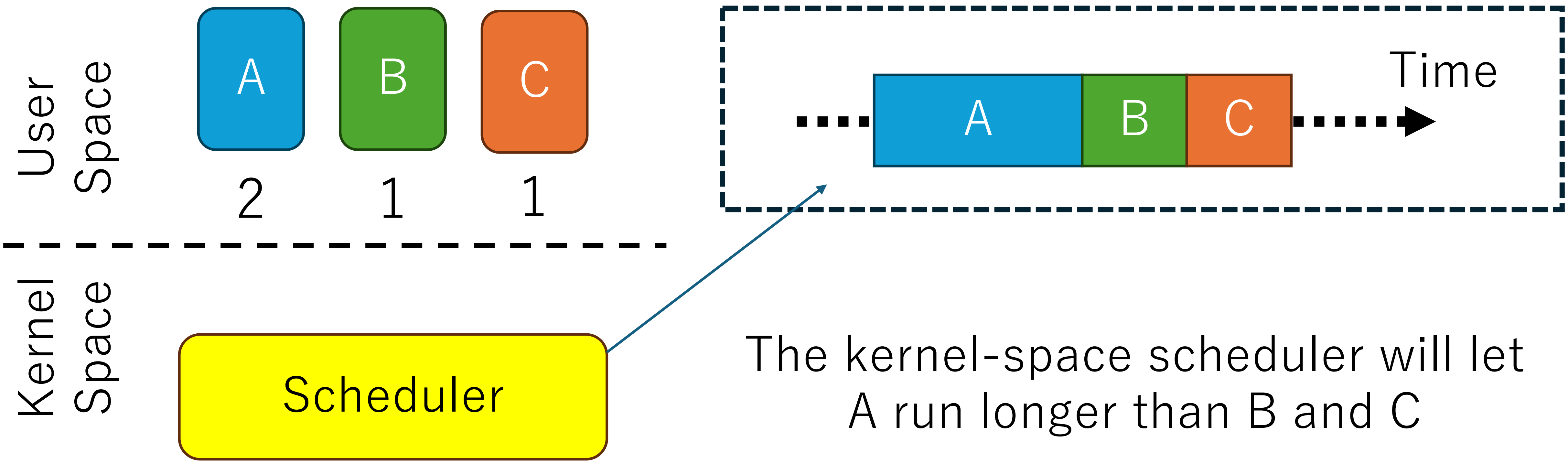
Key Idea

- A kernel-space process scheduler normally gives a longer execution time to a process having a higher priority



Key Idea

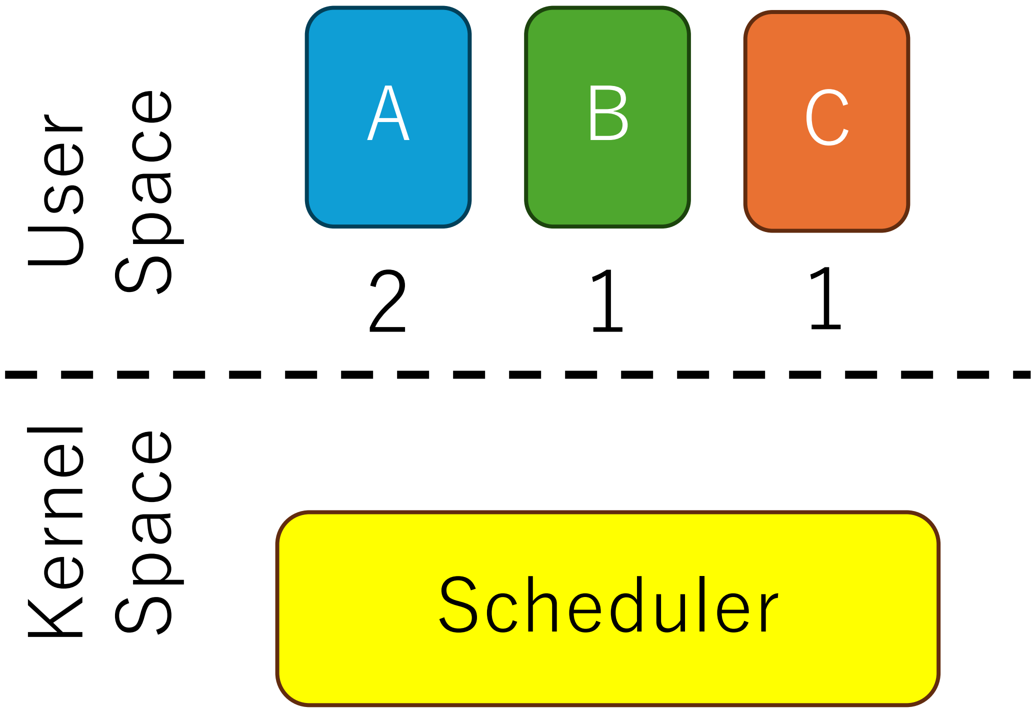
- A kernel-space process scheduler normally gives a longer execution time to a process having a higher priority



The kernel-space scheduler will let A run longer than B and C

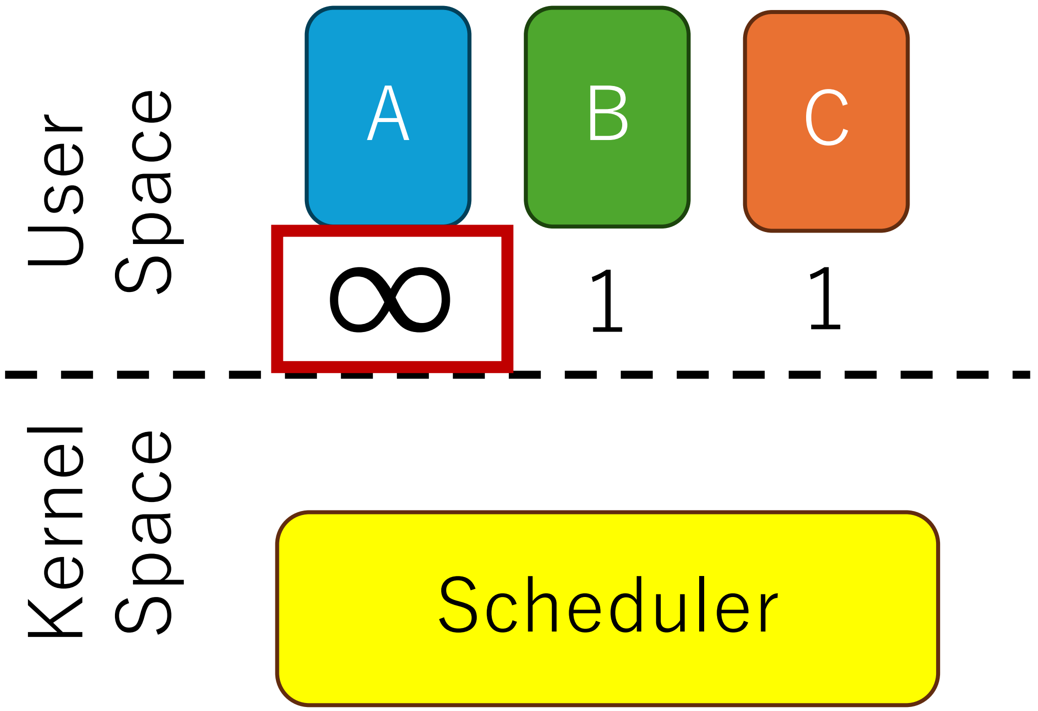
Key Idea

- An extreme case: a process (A) has a very high priority compared to the other processes (B and C)



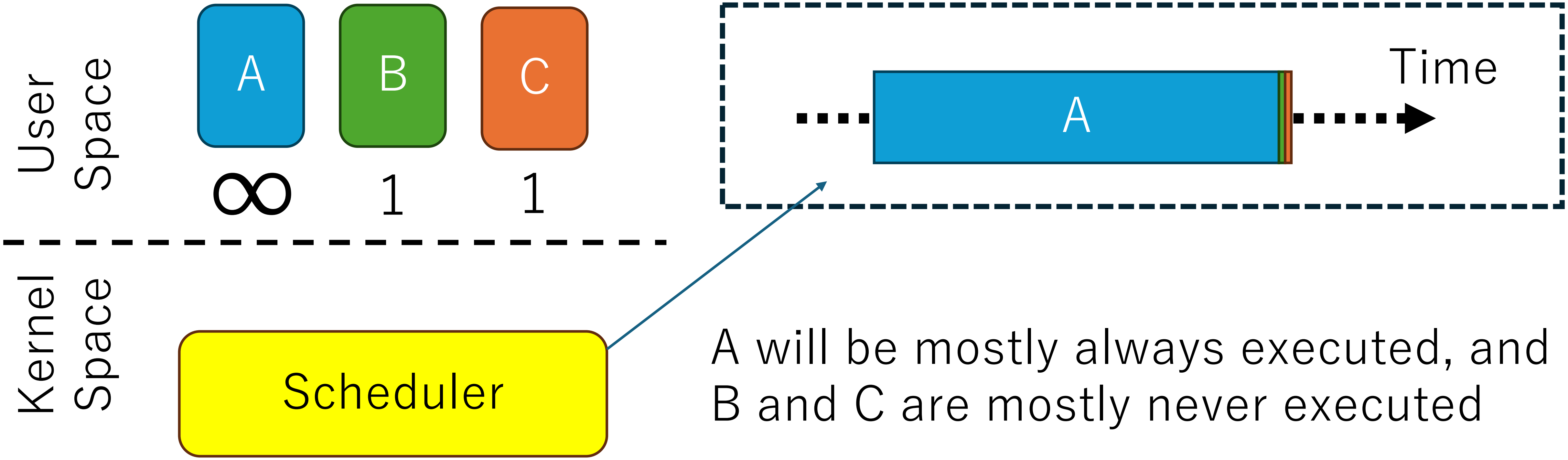
Key Idea

- An extreme case: a process (A) has a very high priority compared to the other processes (B and C)



Key Idea

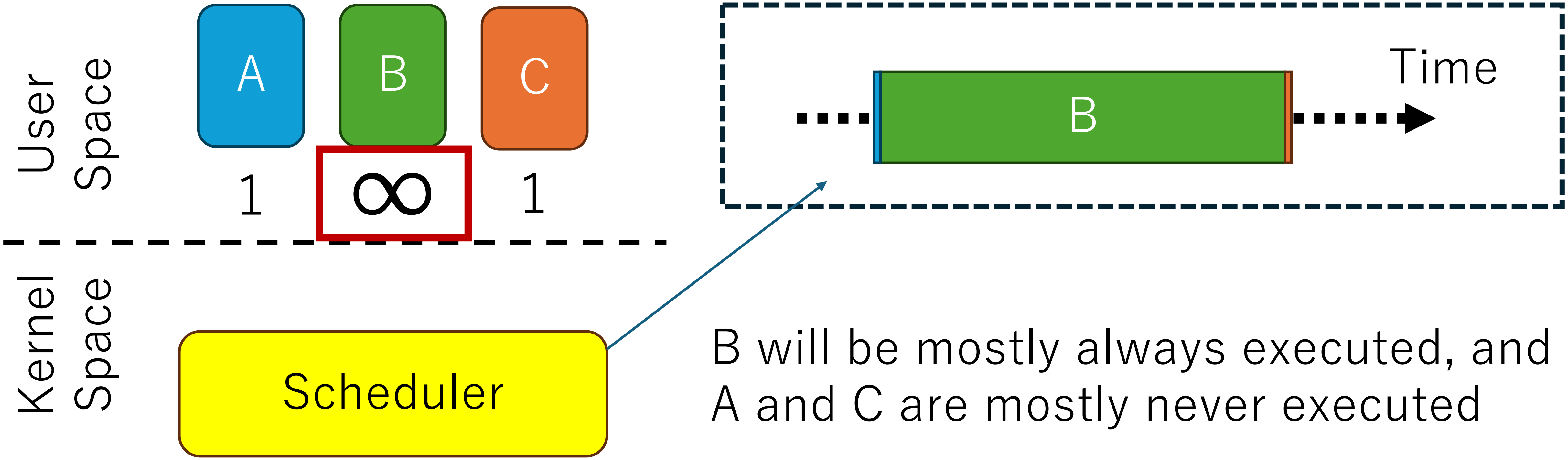
- An extreme case: a process (A) has a very high priority compared to the other processes (B and C)



A will be mostly always executed, and B and C are mostly never executed

Key Idea

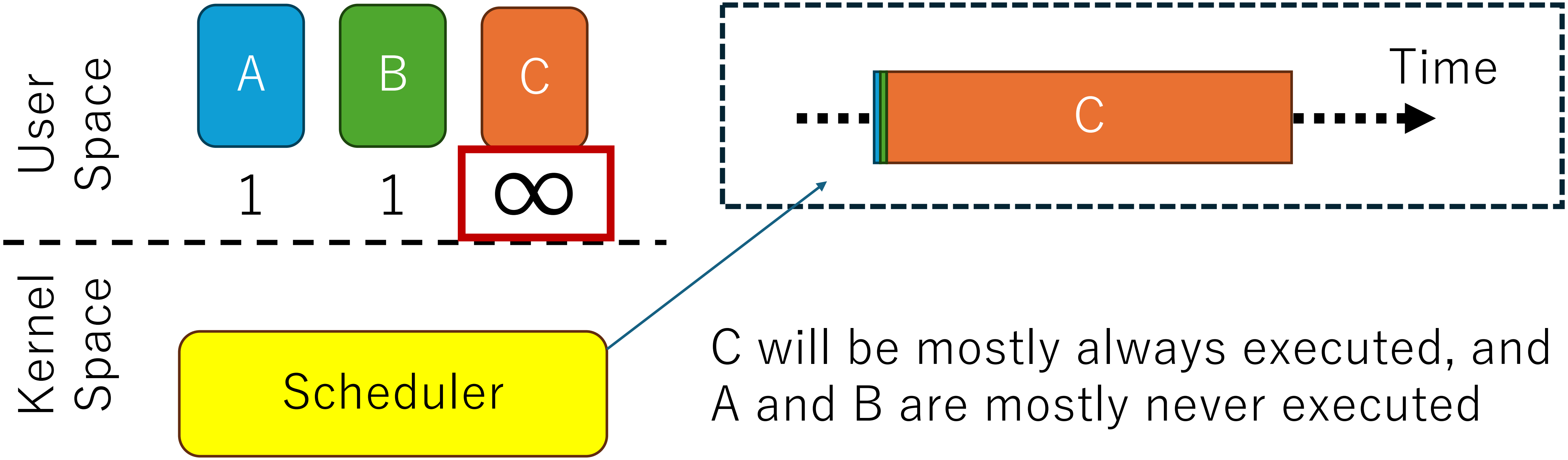
- An extreme case: a process (B) has a very high priority compared to the other processes (A and C)



B will be mostly always executed, and A and C are mostly never executed

Key Idea

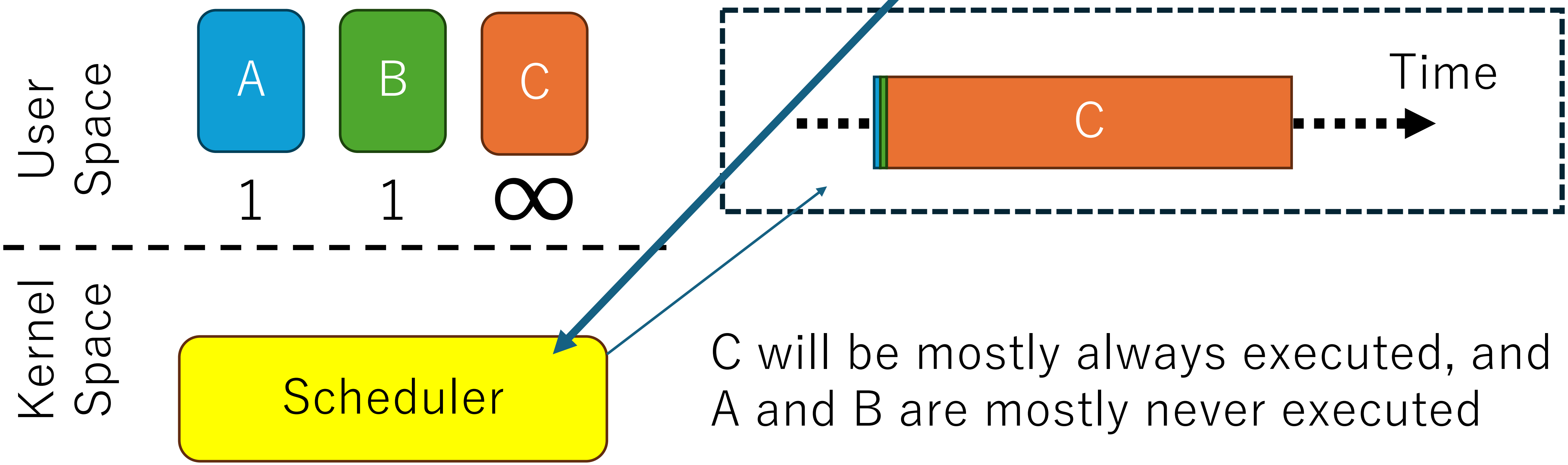
- An extreme case: a process (C) has a very high priority compared to the other processes (A and B)



C will be mostly always executed, and A and B are mostly never executed

Key Idea

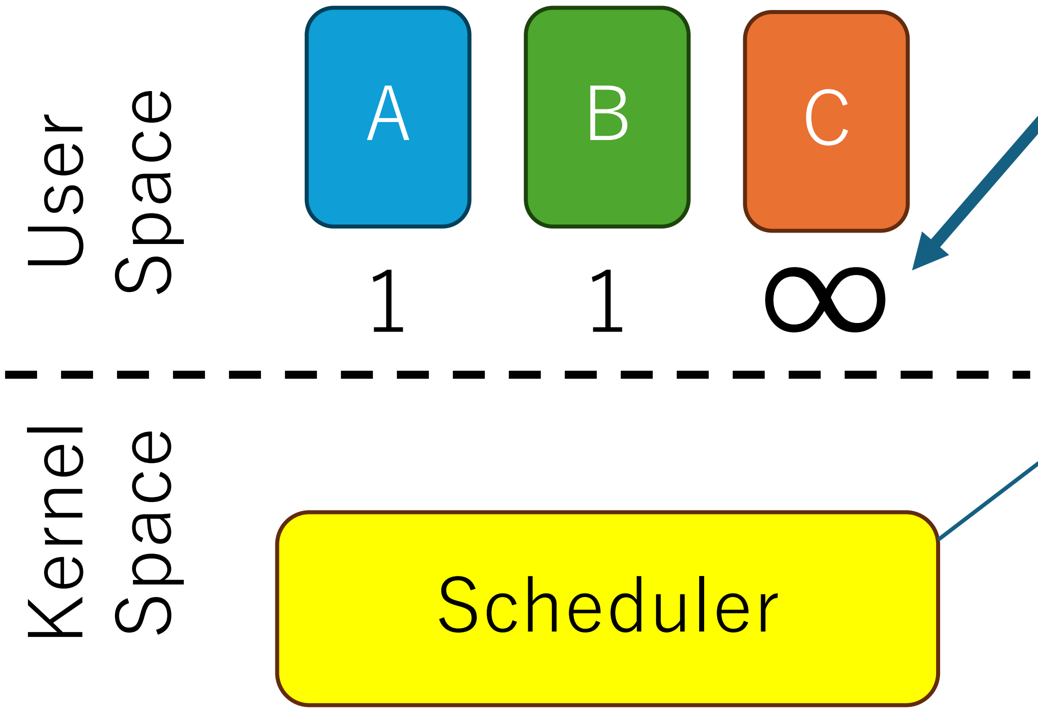
We can indirectly control <u>the kernel-space process scheduler</u>



C will be mostly always executed, and
A and B are mostly never executed

Key Idea

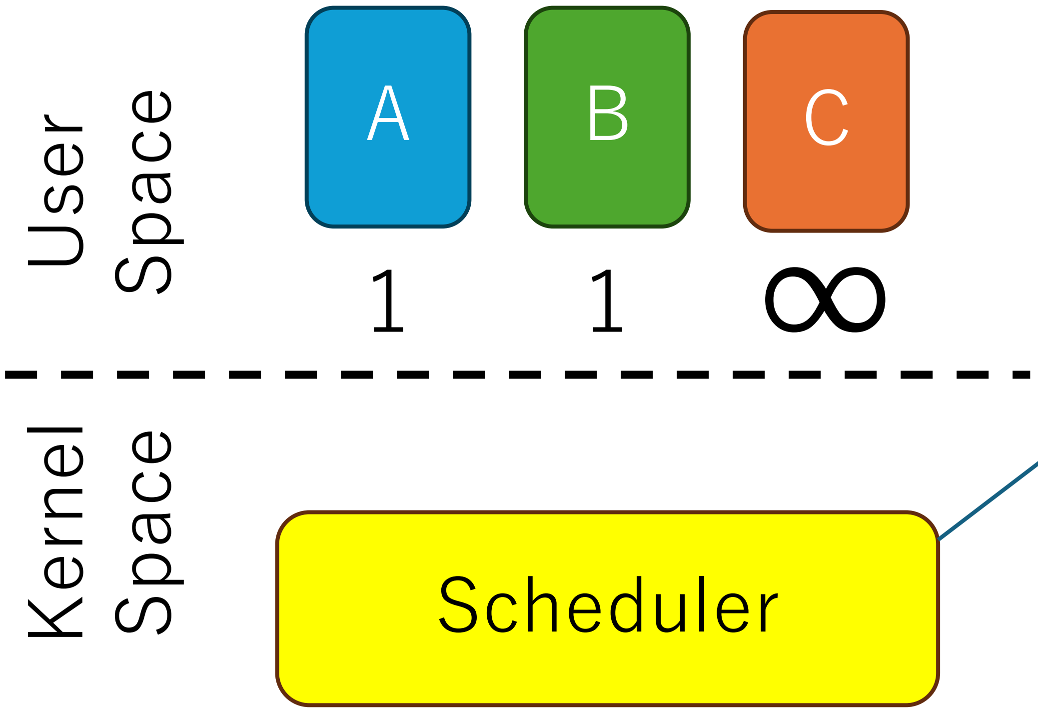
We can indirectly control the kernel-space process scheduler by making sufficiently vast priority gaps among processes;



C will be mostly always executed, and A and B are mostly never executed

Key Idea

We can indirectly control the kernel-space process scheduler by making sufficiently vast priority gaps among processes; we can do this by the kernel-provided priority-setting facility



C will be mostly always executed, and
A and B are mostly never executed

Control Knobs for Prototypes on Linux

- `sched_setscheduler` system call: priority setting

- `sched_setaffinity` system call: CPU core affinity setting

Control Knobs for Prototypes on Linux

- `sched_setscheduler` system call: priority setting
  - argument 1: process ID (pid) of a process
  - argument 2: scheduling policy
  - argument 3: parameter (e.g., priority value)

- `sched_setaffinity` system call: CPU core affinity setting
  - argument 1: process ID (pid) of a process
  - argument 2 and 3: CPU core affinity specification

Control Knobs for Prototypes on Linux

- **`sched_setscheduler`** system call: priority setting
  - argument 1: process ID (pid) of a process
  - argument 2: scheduling policy
  - argument 3: parameter (e.g., priority value)

- **`sched_setaffinity`** system call: CPU core affinity setting
  - argument 1: process ID (pid) of a process
  - argument 2 and 3: CPU core affinity specification

The priority elevation trick can be applied for scheduling entities having pids (i.e., **processes, pthreads, vCPUs** backed by QEMU)

Control Knobs for Prototypes on Linux

- `sched_setscheduler` system call: priority setting
  - argument 1: process ID (pid) of a process
  - argument 2: scheduling policy
  - argument 3: parameter (e.g., priority value)

- `sched_setaffinity` system call: CPU core affinity setting
  - argument 1: process ID (pid) of a process
  - argument 2 and 3: CPU core affinity specification

We use the **SCHED_FIFO** policy of the Linux scheduler and its prioritization scheme to configure extreme priority gaps

Control Knobs for Prototypes on Linux

- `sched_setscheduler` system call: priority setting
  - argument 1: process ID (pid) of a process
  - argument 2: scheduling policy
  - argument 3: parameter (e.g., priority value)

- `sched_setaffinity` sy
  - argument 1: process ID (p
  - argument 2 and 3: CPU co

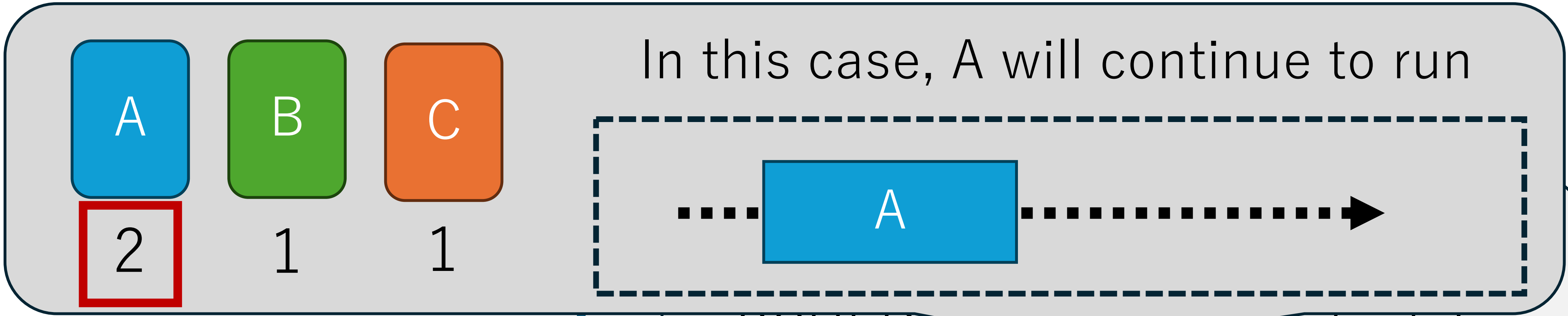
Point

SCHED_FIFO **always** schedules the highest priority process

We use the **SCHED_FIFO** policy of the Linux scheduler and its prioritization scheme to configure extreme priority gaps

Control Knobs for Prototypes on Linux

- **sched_setscheduler** system call: priority setting



  - argument 1: process ID (p
  - argument 2 and 3: CPU co

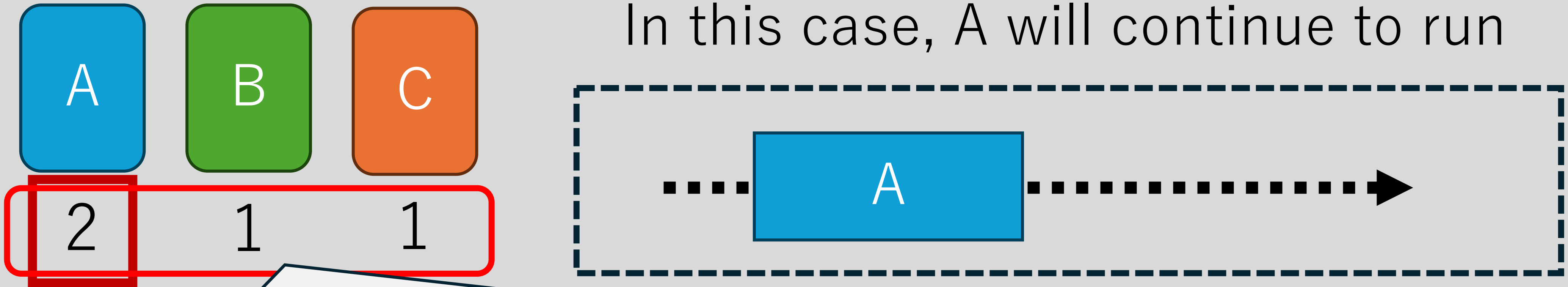
SCHED_FIFO **always** schedules the highest priority process

We use the **SCHED_FIFO** policy of the Linux scheduler and its prioritization scheme to configure extreme priority gaps

Control Knobs for Prototypes on Linux

- **`sched_setscheduler`** system call: priority setting

In this case, A will continue to run

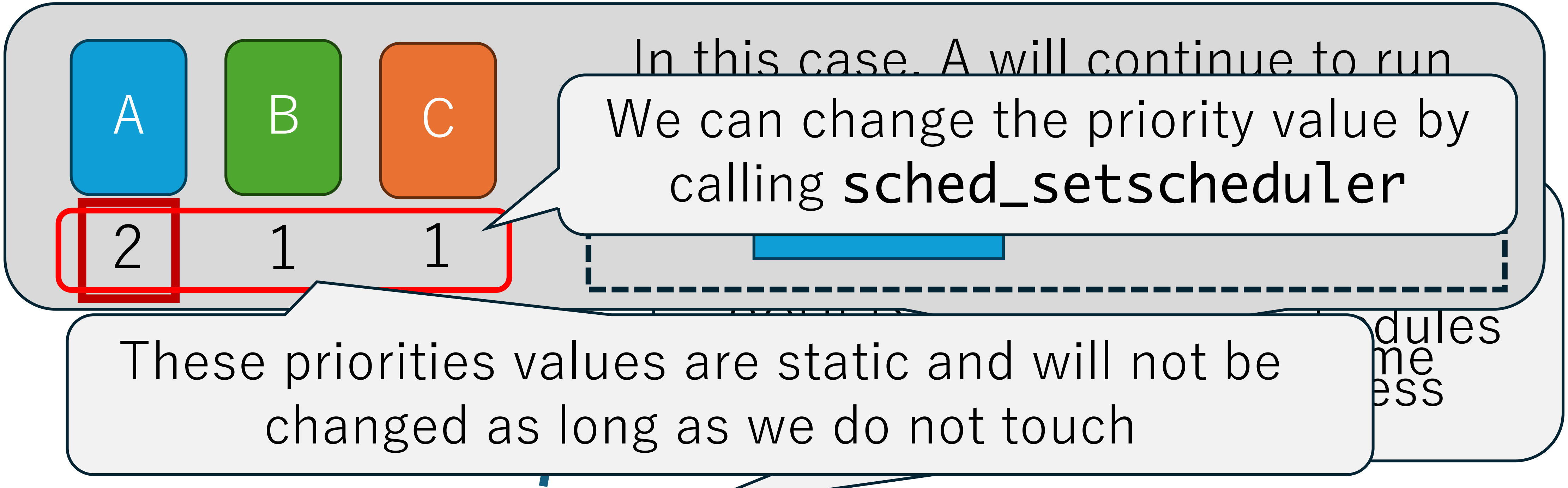


These priorities values are static and will not be changed as long as we do not touch

We use the **SCHED_FIFO** policy of the Linux scheduler and its prioritization scheme to configure extreme priority gaps

Control Knobs for Prototypes on Linux

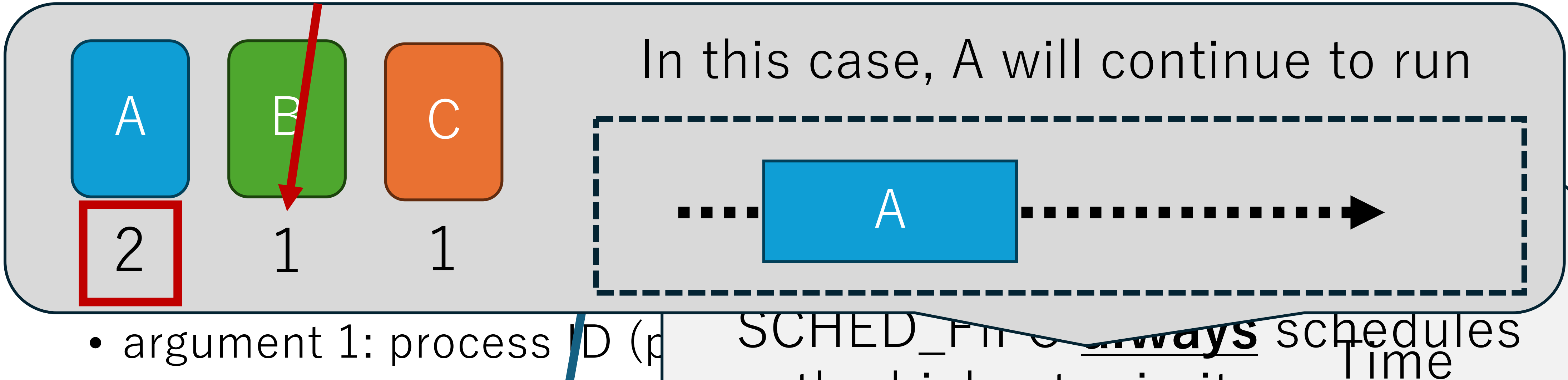
- **sched_setscheduler** system call: priority setting



We use the **SCHED_FIFO** policy of the Linux scheduler and its prioritization scheme to configure extreme priority gaps

Control Knobs for Prototypes on Linux

- `sched_setscheduler` system call: priority setting



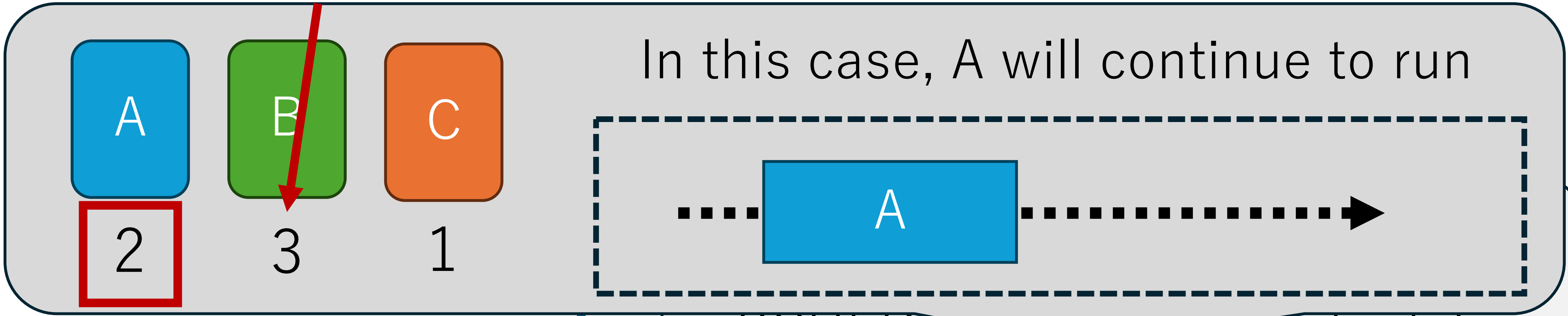
  - argument 1: process ID (p
  - argument 2 and 3: CPU c

SCHED_FIFO **always** schedules the highest priority process

We use the **SCHED_FIFO** policy of the Linux scheduler and its prioritization scheme to configure extreme priority gaps

Control Knobs for Prototypes on Linux

- `sched_setscheduler` system call: priority setting



  - argument 1: process ID (p
  - argument 2 and 3: CPU co

SCHED_FIFO **always** schedules the highest priority process

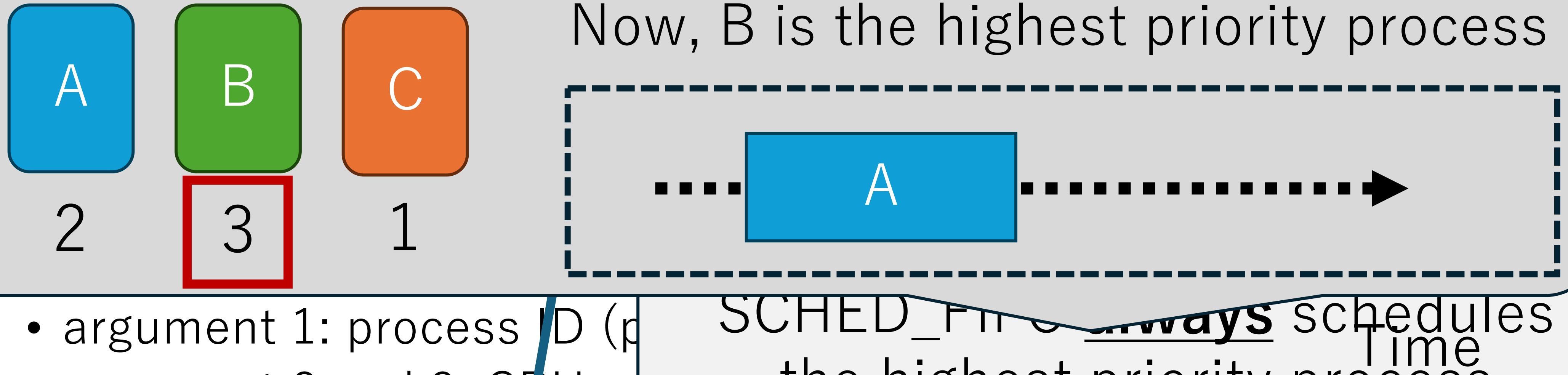
Time

We use the **SCHED_FIFO** policy of the Linux scheduler and its prioritization scheme to configure extreme priority gaps

Control Knobs for Prototypes on Linux

- `sched_setscheduler` system call: priority setting

Now, B is the highest priority process



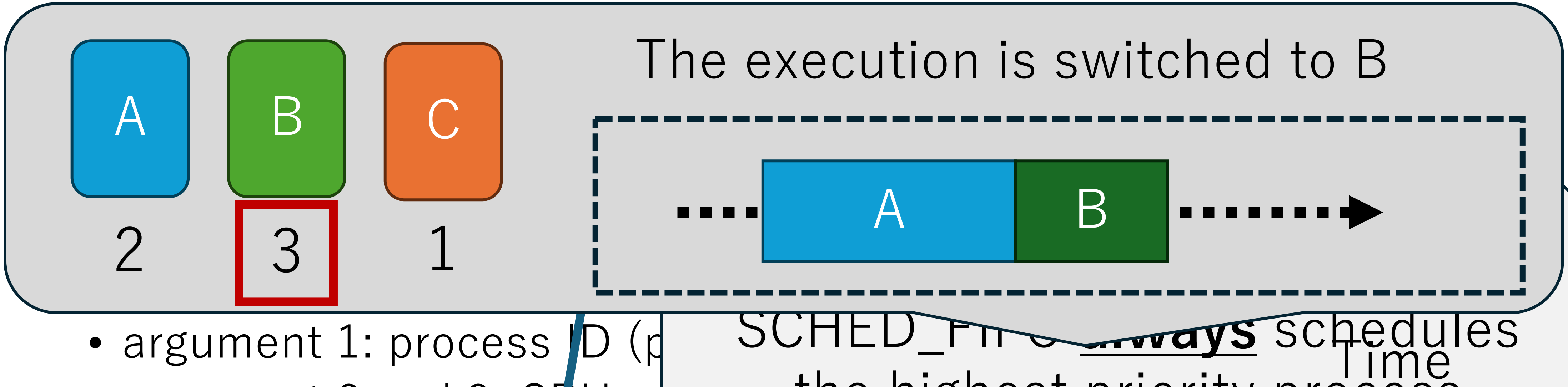
  - argument 1: process ID (p
  - argument 2 and 3: CPU co

SCHED_FIFO **always** schedules the highest priority process

We use the **SCHED_FIFO** policy of the Linux scheduler and its prioritization scheme to configure extreme priority gaps

Control Knobs for Prototypes on Linux

- **`sched_setscheduler`** system call: priority setting
  - argument 1: process ID (p
  - argument 2 and 3: CPU co

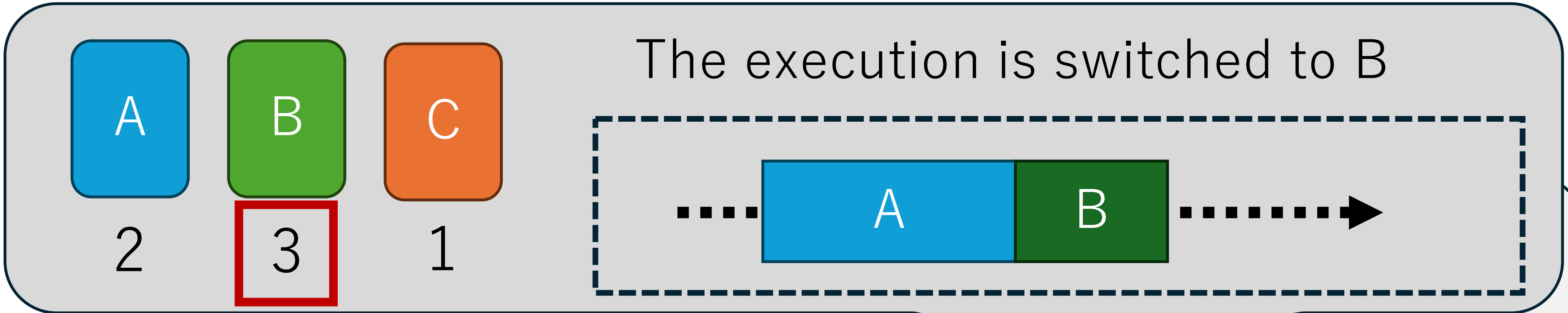


SCHED_FIFO **always** schedules the highest priority process

We use the **SCHED_FIFO** policy of the Linux scheduler and its prioritization scheme to configure extreme priority gaps

Control Knobs for Prototypes on Linux

- `sched_setscheduler` system call: priority setting

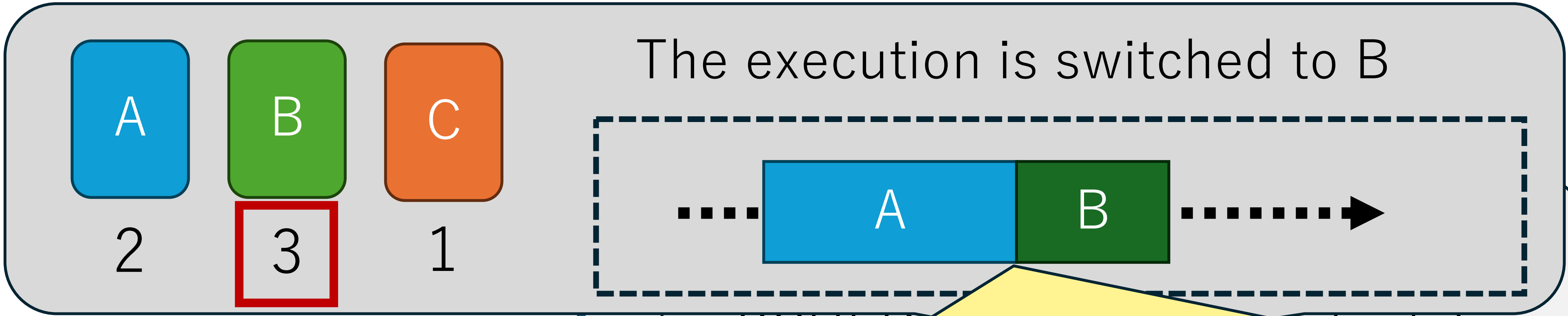


This is the behavior of SCHED_FIFO-applied processes

We use the **SCHED_FIFO** policy of the Linux scheduler and its prioritization scheme to configure extreme priority gaps

Control Knobs for Prototypes on Linux

- `sched_setscheduler` system call: priority setting



The priority elevation trick leverages this behavior to indirectly control the kernel-space process scheduler

We use the **SCHED_FIFO** policy of the Linux scheduler and its prioritization scheme to configure extreme priority gaps

Process Types Considered in the Trick

Process Types Considered in the Trick

- A normal process is normal and does not perform scheduling

A B C : normal process

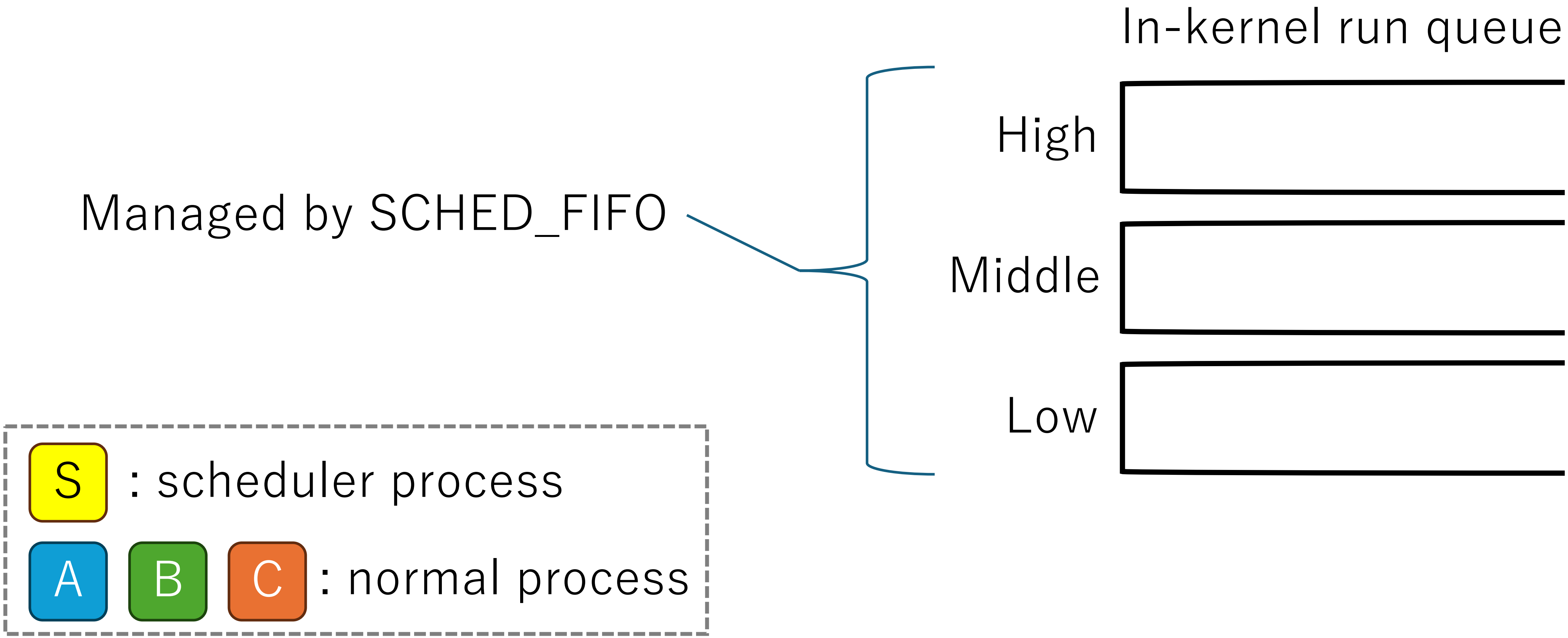
Process Types Considered in the Trick

- A normal process is normal and does not perform scheduling
- A scheduler process schedules normal processes

S : scheduler process

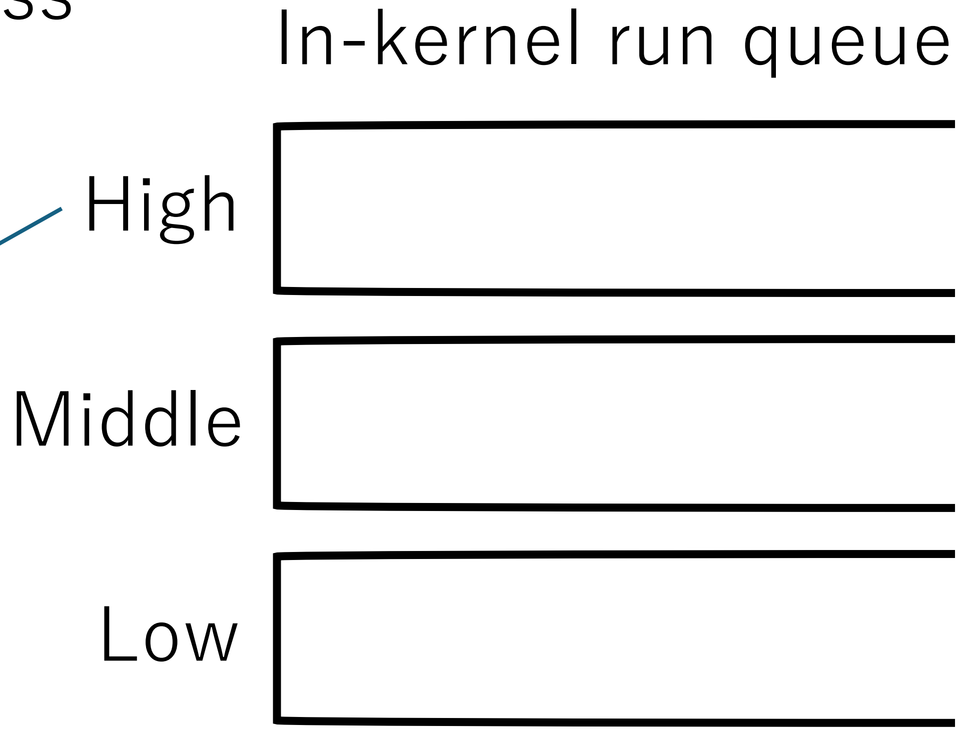
A B C : normal process

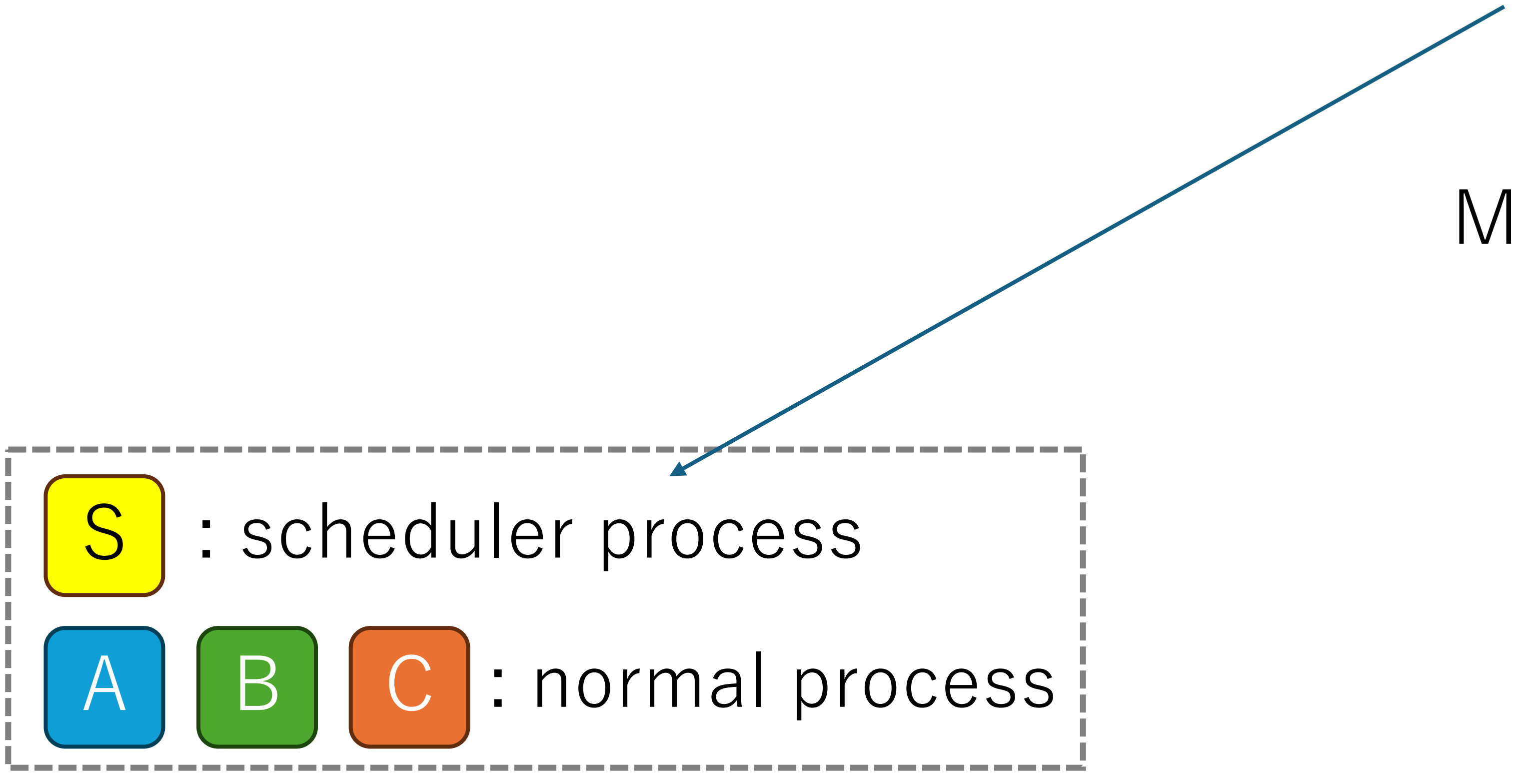
Three Priority Values



Three Priority Values

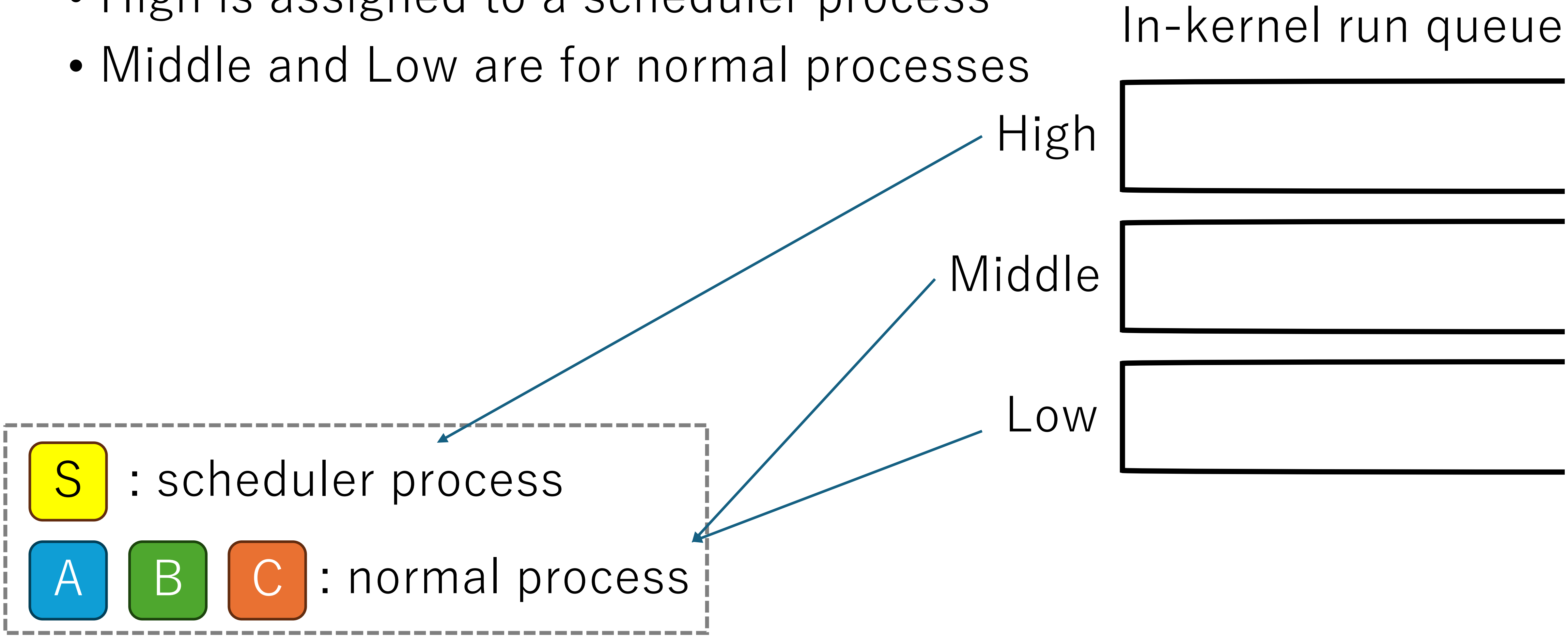
- High is assigned to a scheduler process





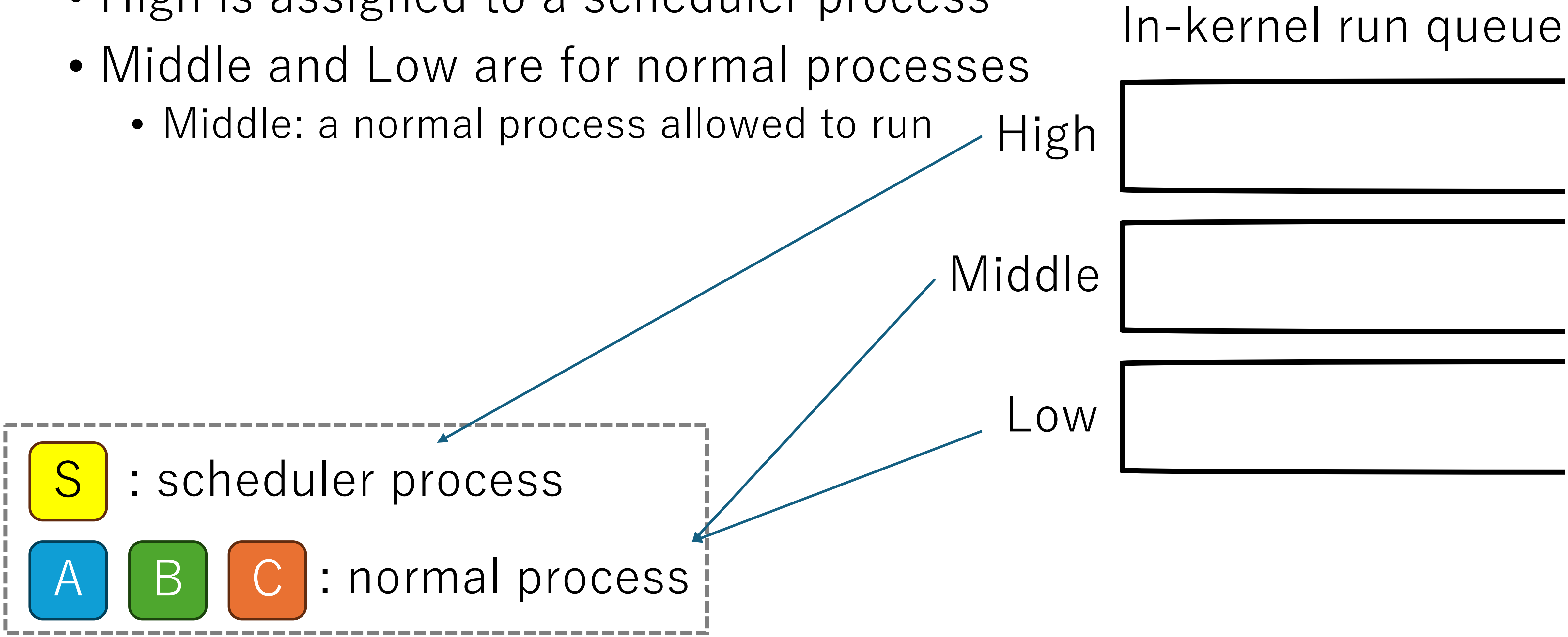
Three Priority Values

- High is assigned to a scheduler process
- Middle and Low are for normal processes



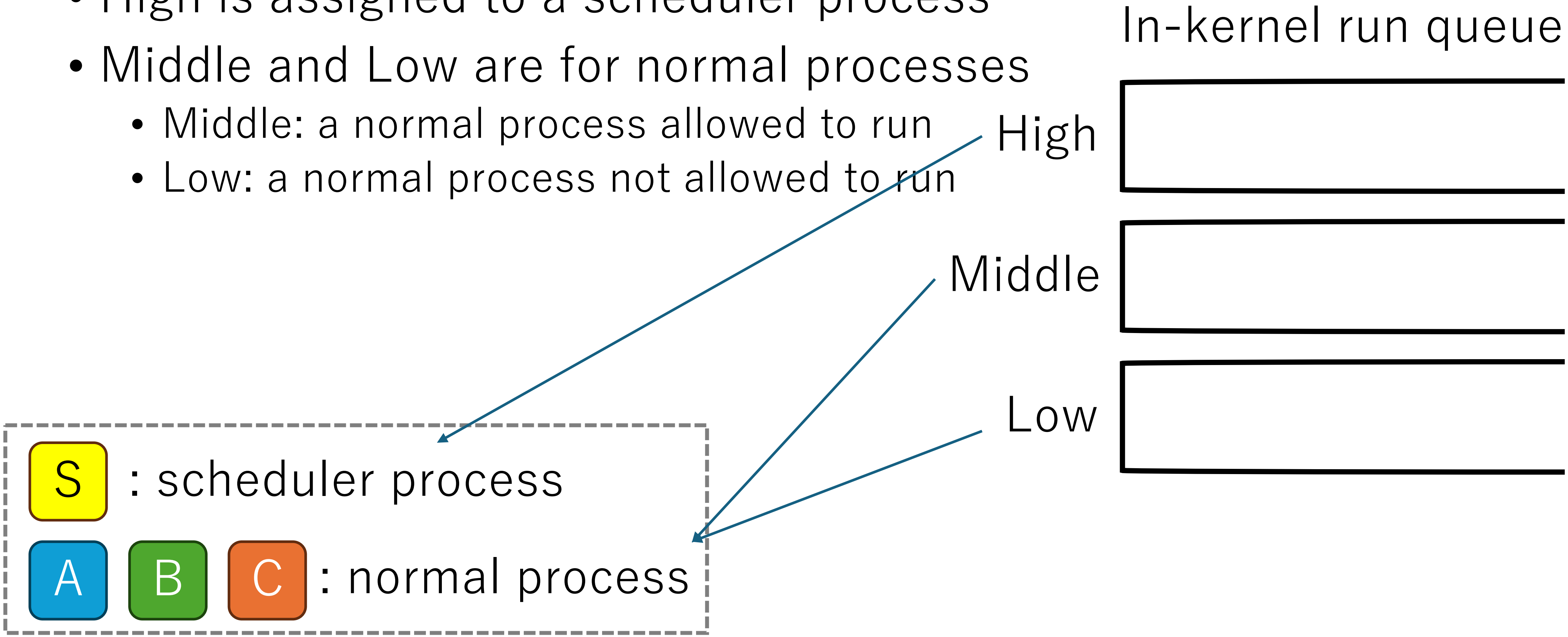
Three Priority Values

- High is assigned to a scheduler process
- Middle and Low are for normal processes
  - Middle: a normal process allowed to run



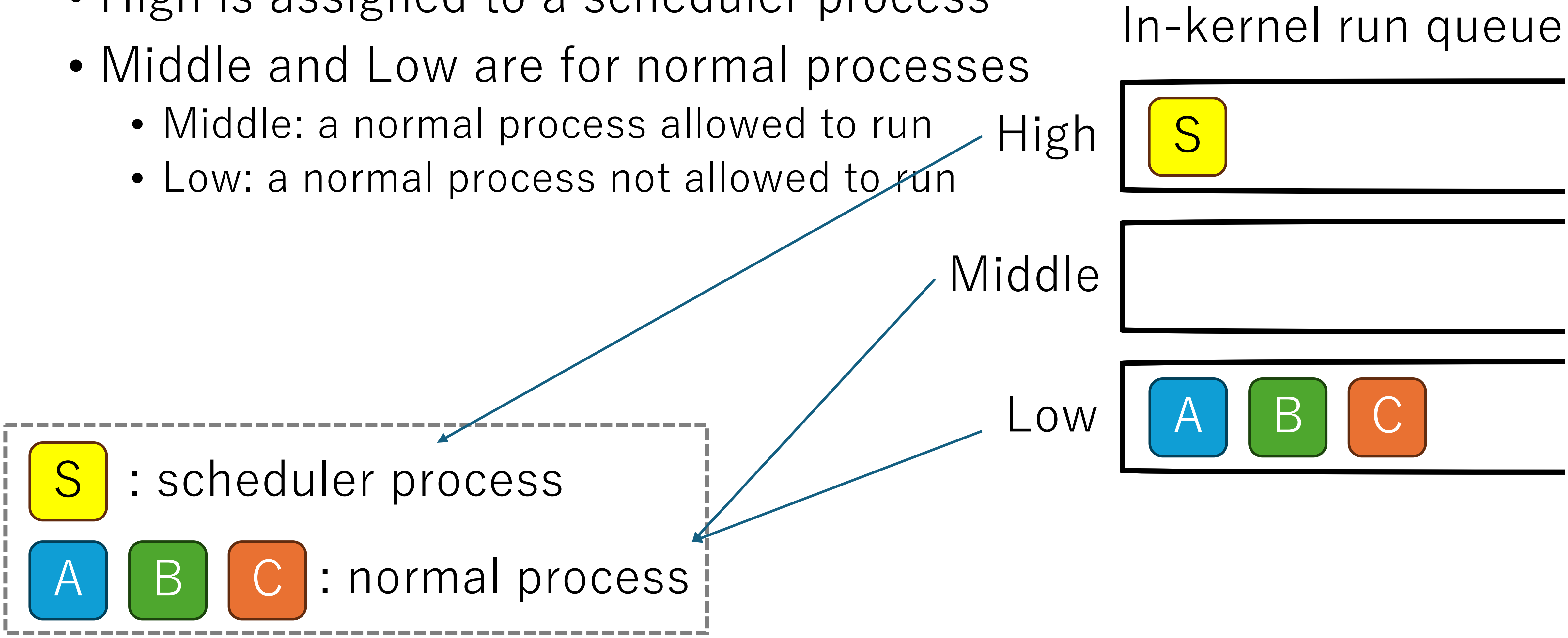
Three Priority Values

- High is assigned to a scheduler process
- Middle and Low are for normal processes
  - Middle: a normal process allowed to run
  - Low: a normal process not allowed to run



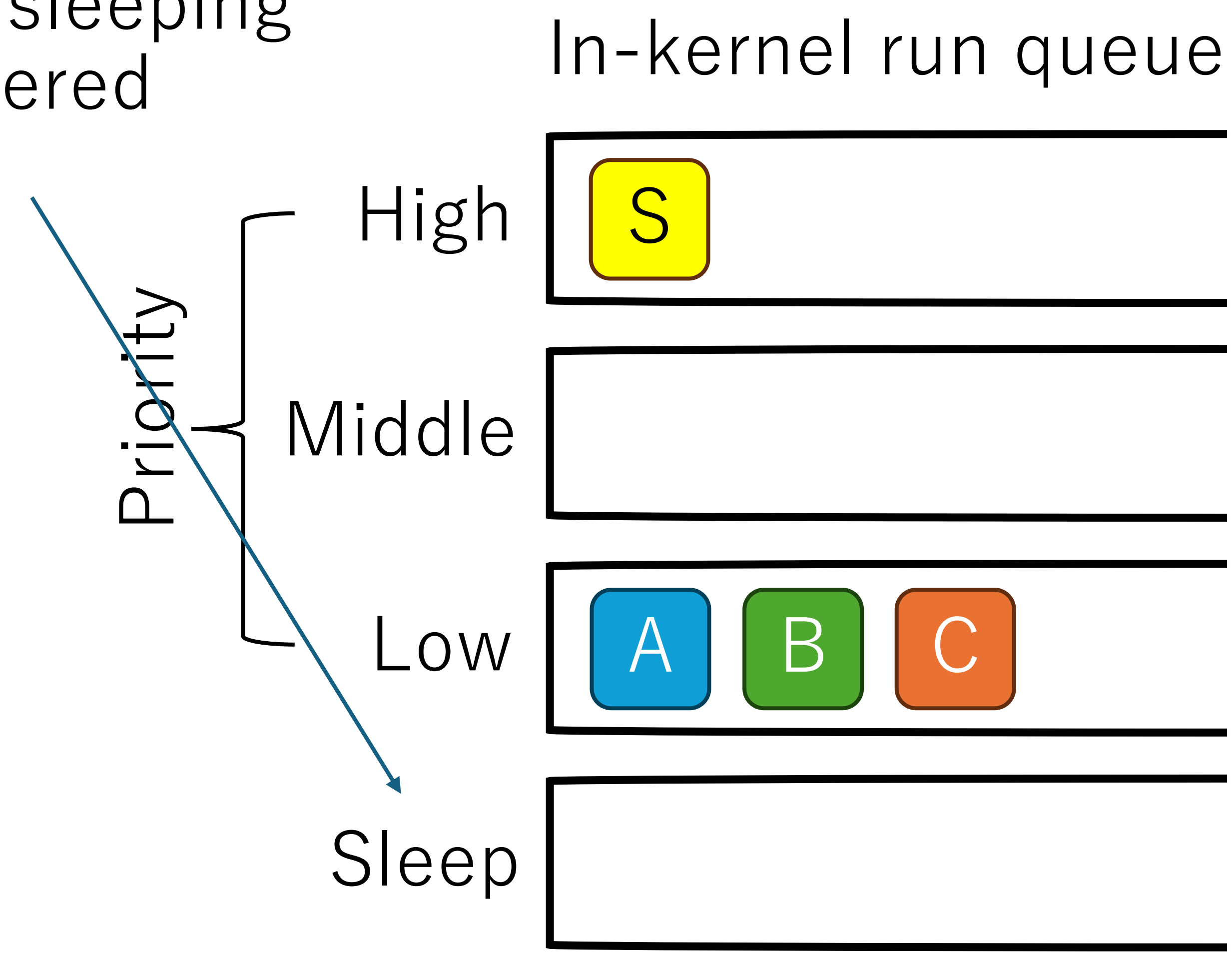
Initial Priority Setting

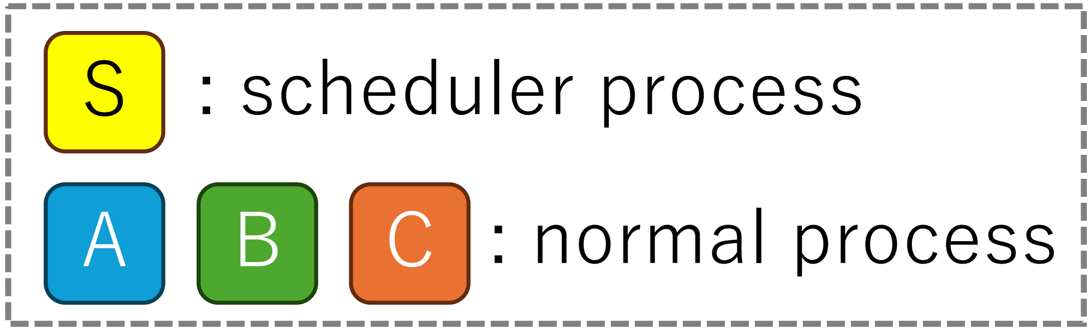
- High is assigned to a scheduler process
- Middle and Low are for normal processes
  - Middle: a normal process allowed to run
  - Low: a normal process not allowed to run



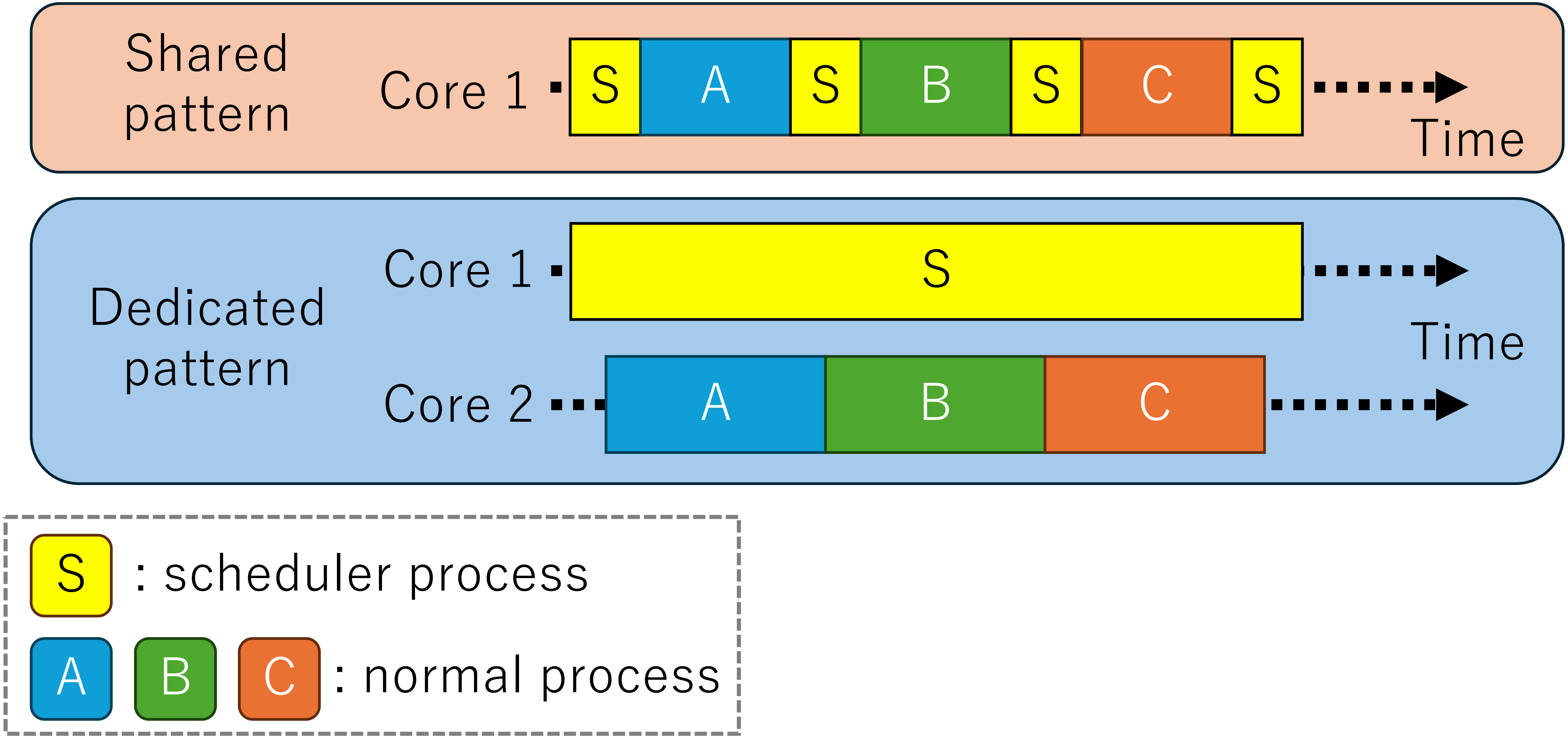
Initial Priority Setting

- The kernel maintains a list of sleeping processes that are not considered candidates for the scheduling

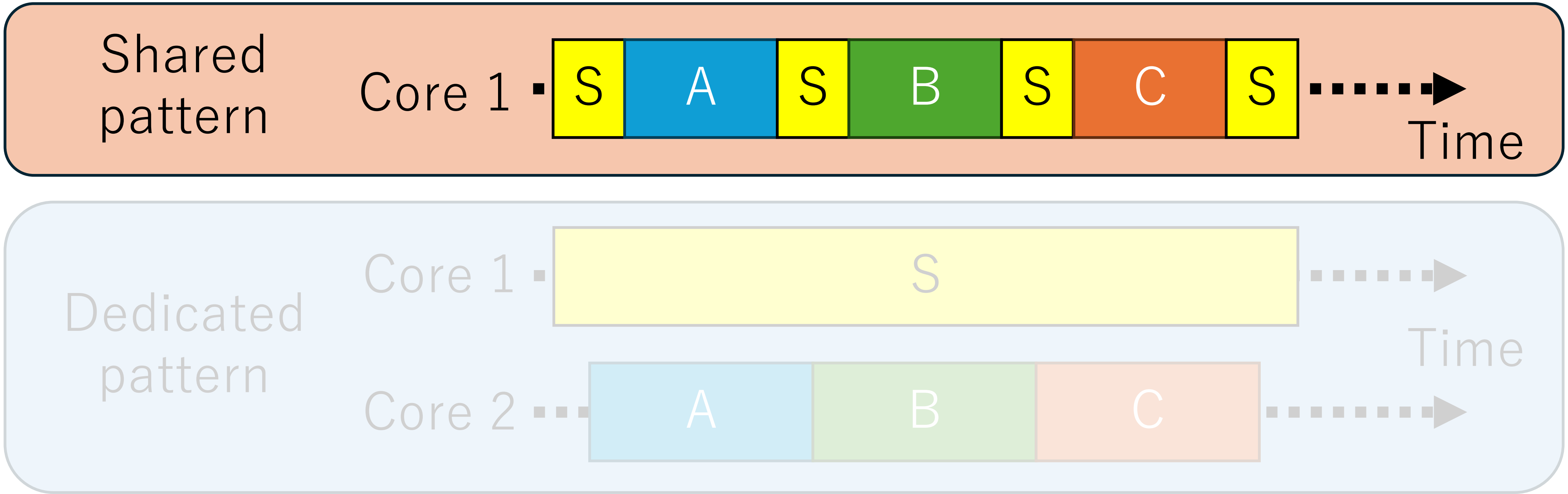


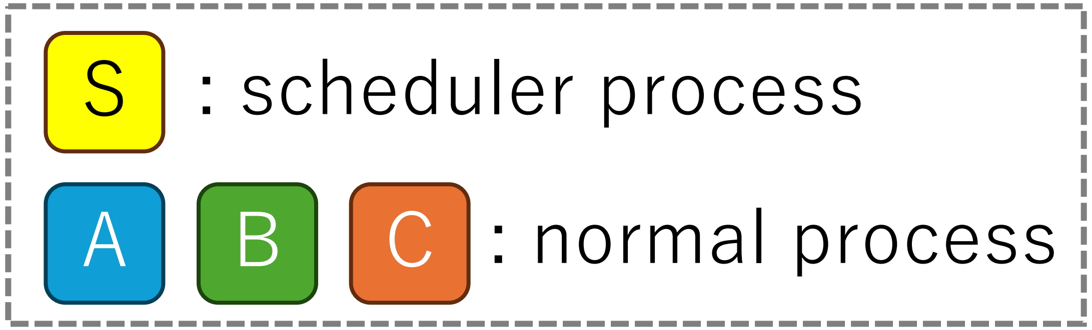


Two CPU Core Assignment Patterns



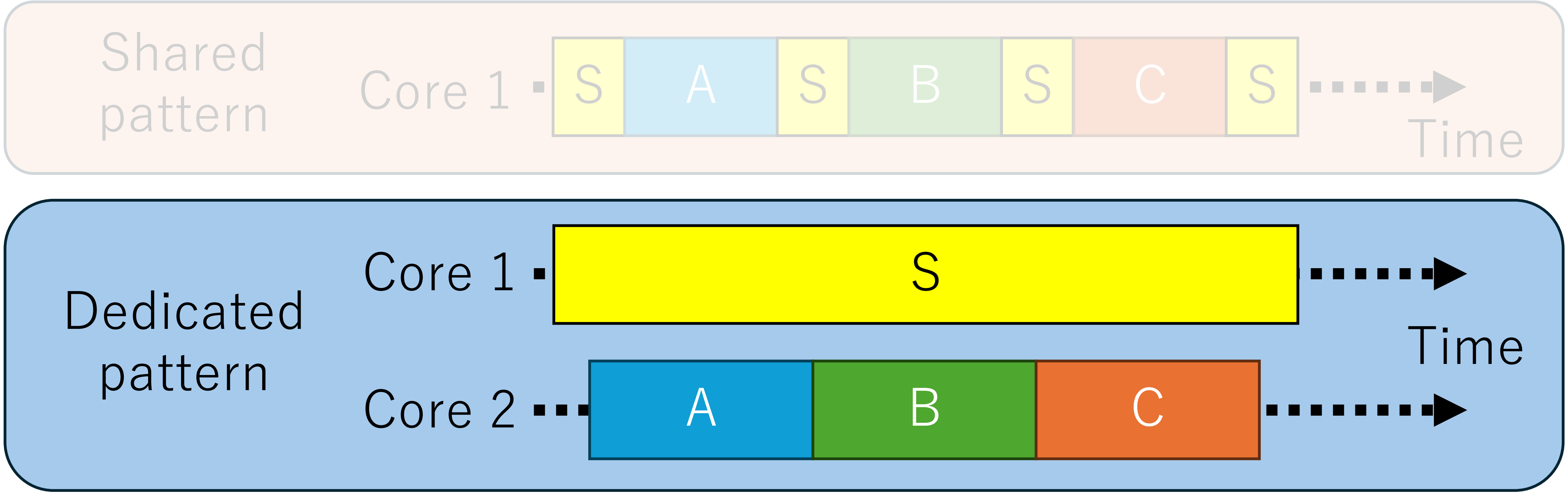
Two CPU Core Assignment Patterns

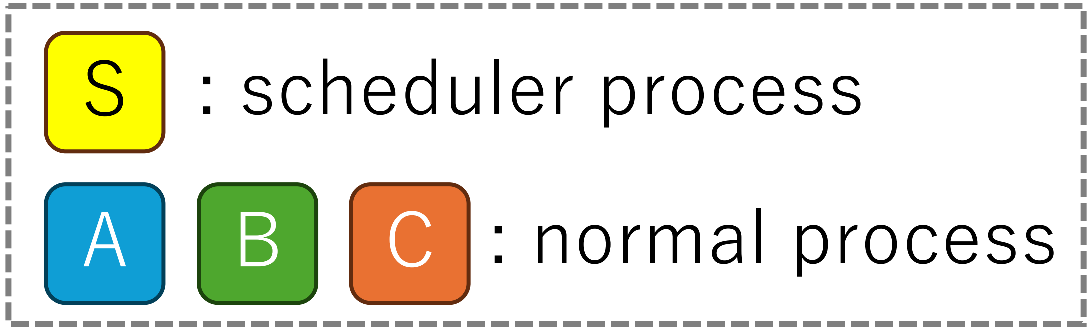




The shared pattern runs scheduler and normal processes on the **same** CPU core

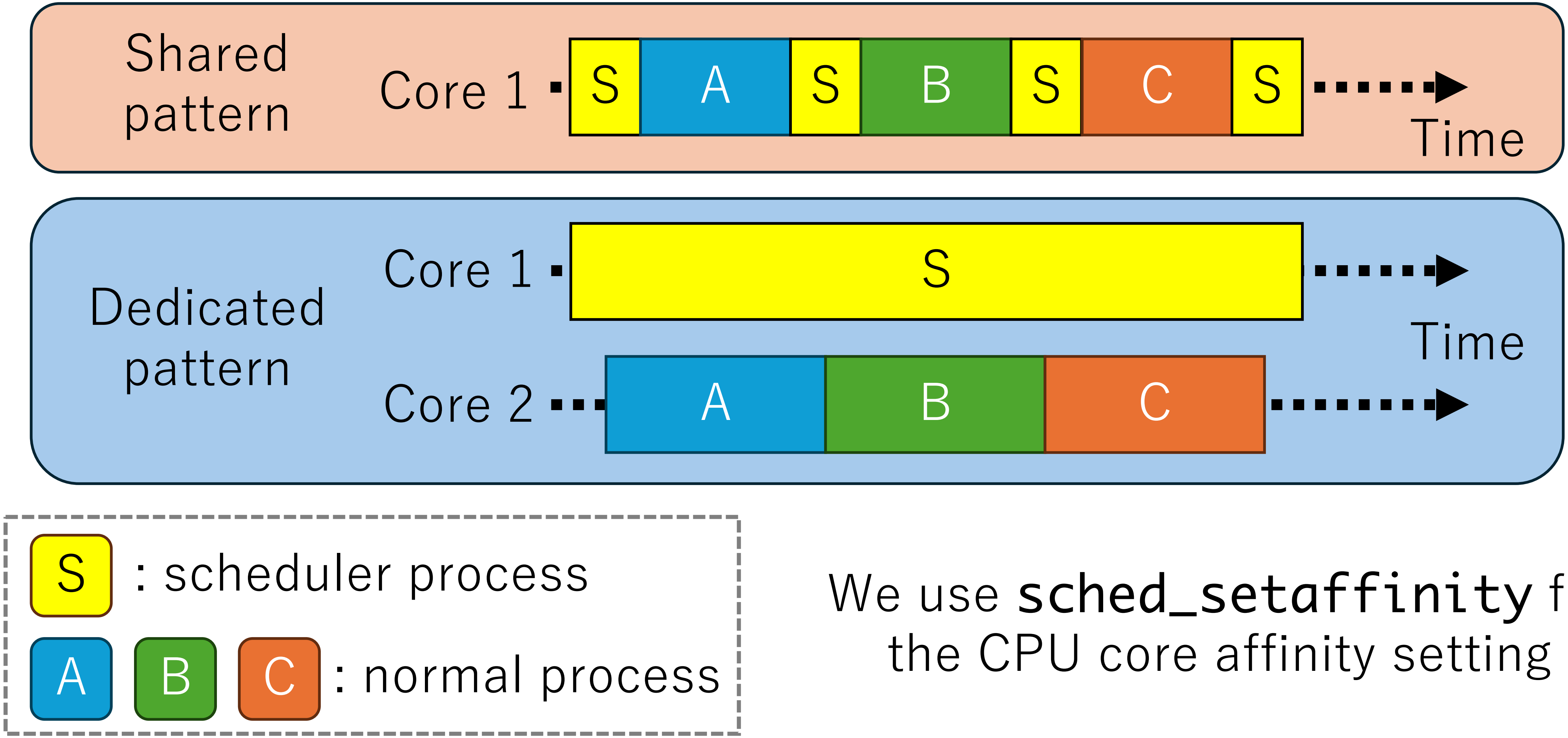
Two CPU Core Assignment Patterns





The dedicated pattern runs scheduler and normal processes on **different** CPU cores

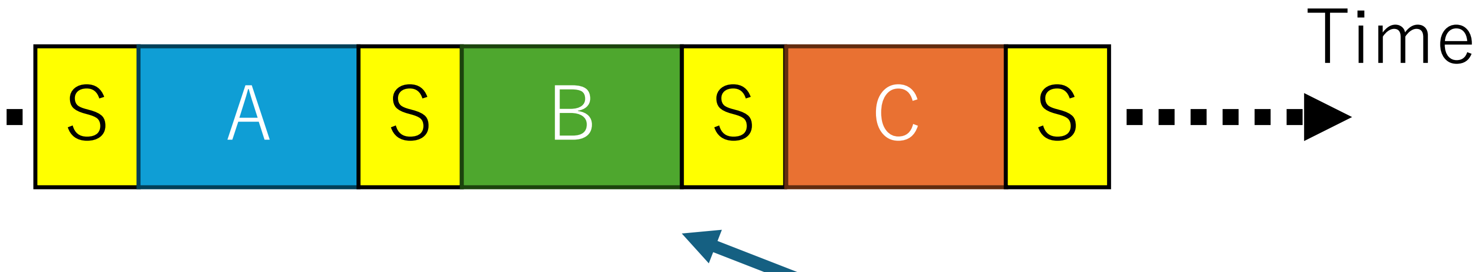
Two CPU Core Assignment Patterns



We use **sched_setaffinity** for the CPU core affinity setting

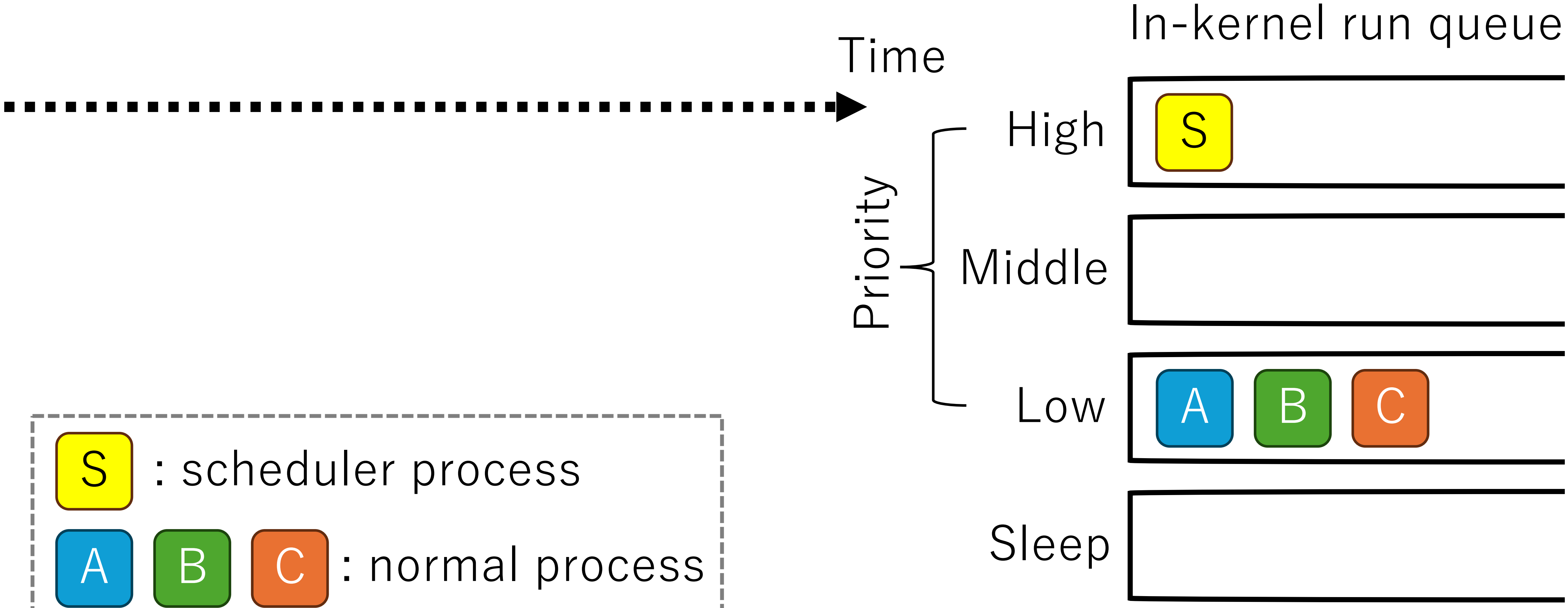
Priority Elevation (Shared Pattern)

Round-robin scheduling policy



From here, we look through how a round-robin scheduling policy can be implemented with the shared pattern

Priority Elevation (Shared Pattern)

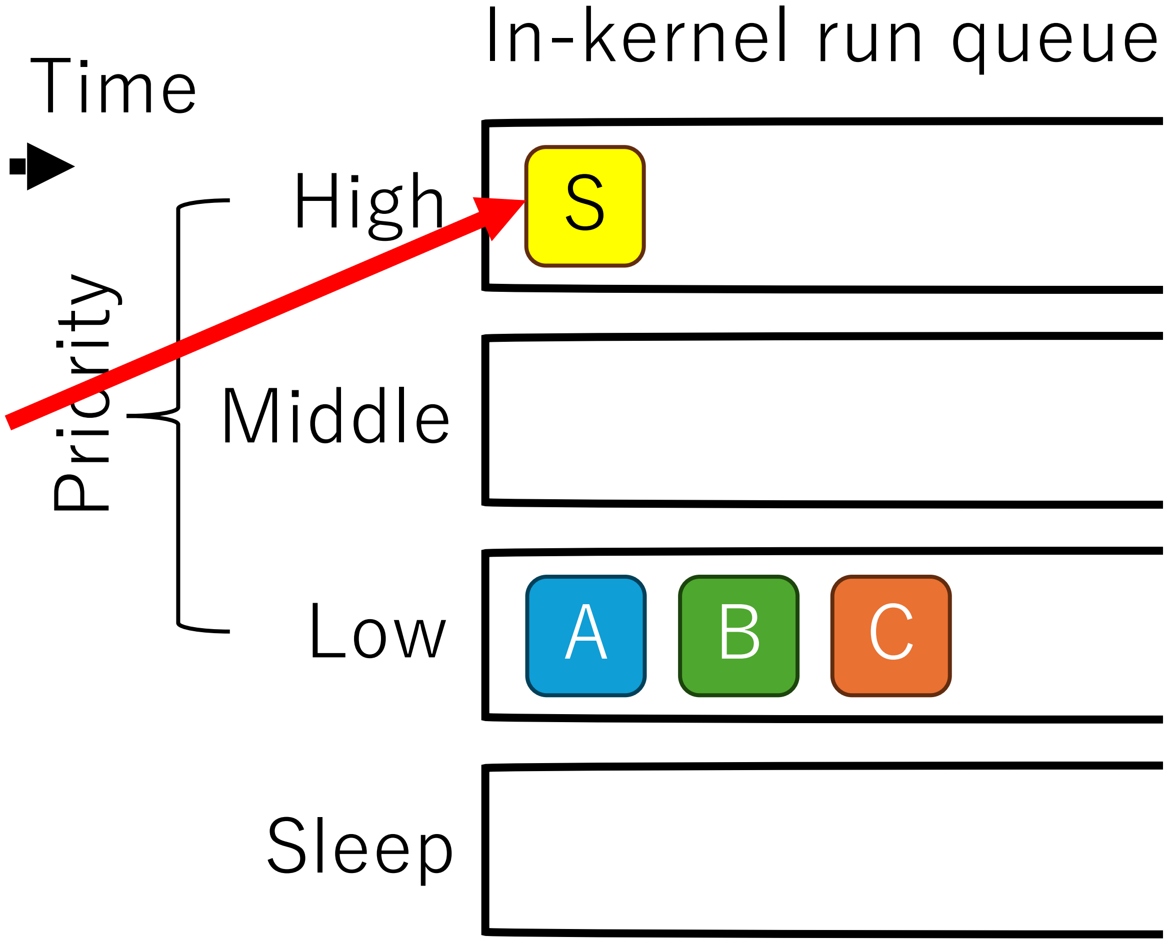


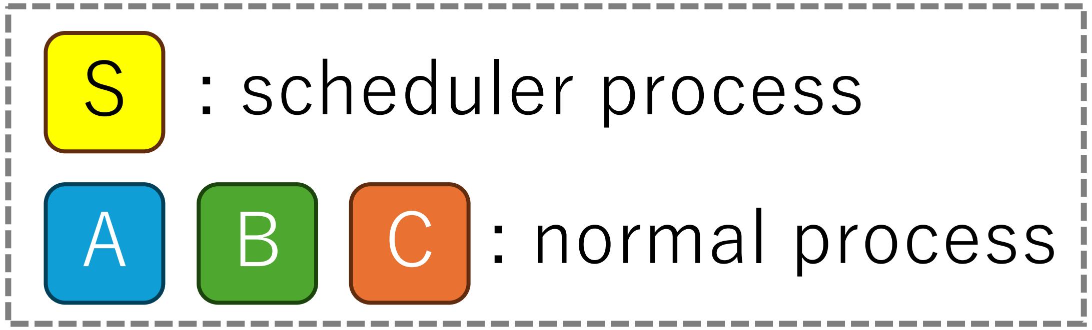
S : scheduler process

A B C : normal process

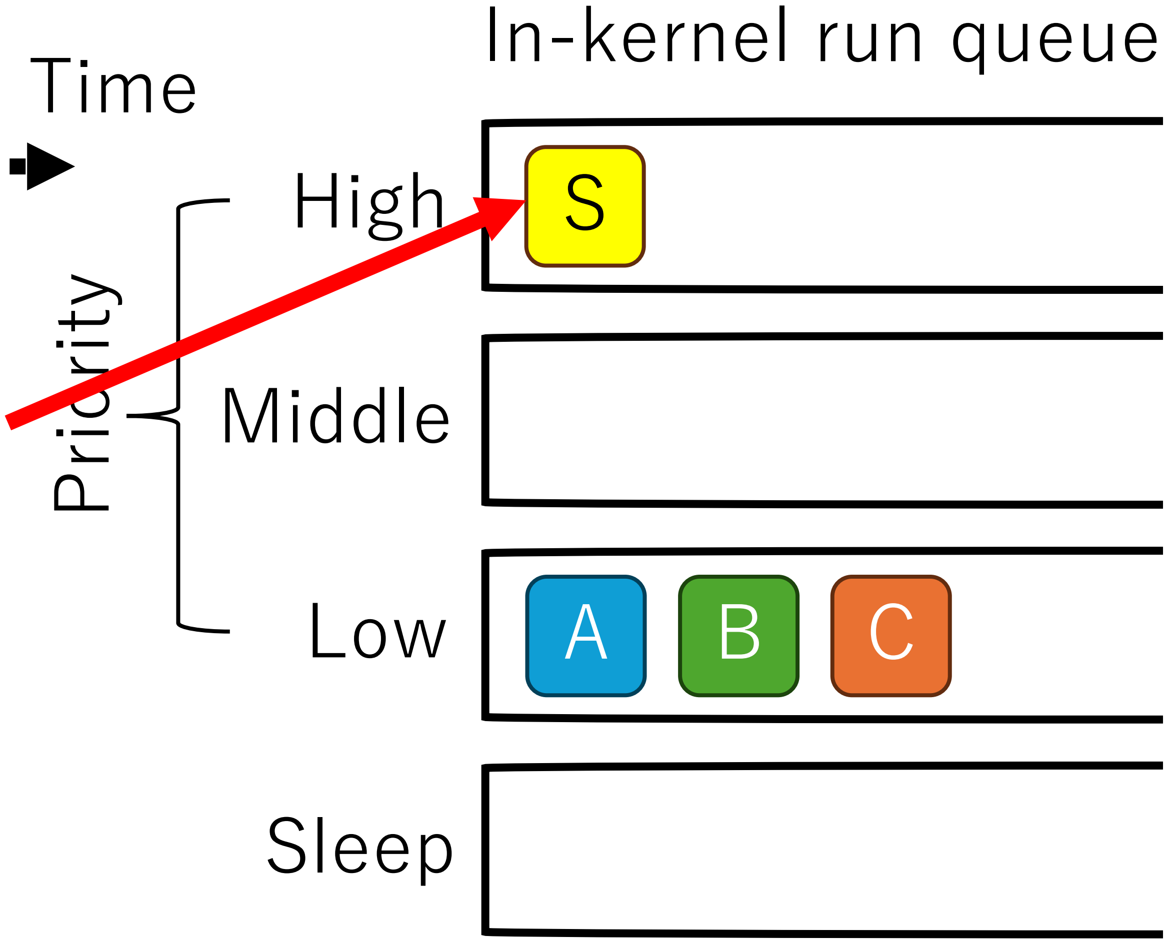
Priority Elevation（Shared Pattern)

The kernel-space process scheduler picks up the highest priority process



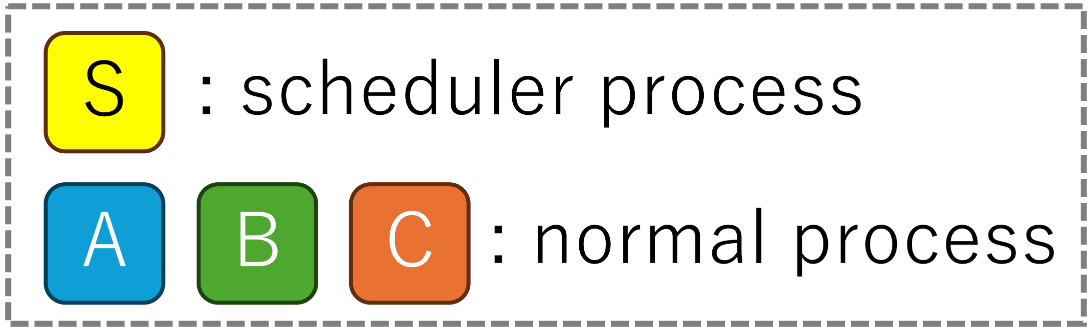


Priority Elevation (Shared Pattern)

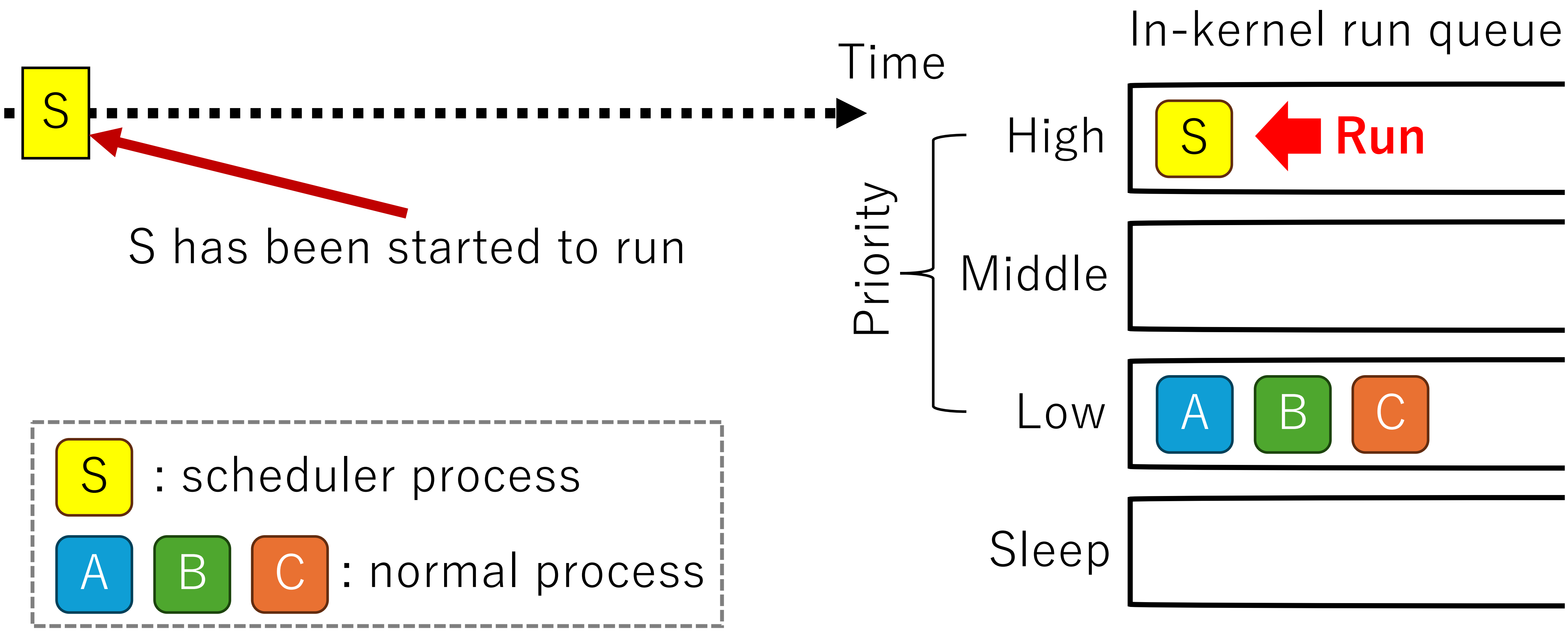


The kernel-space process scheduler picks up the highest priority process

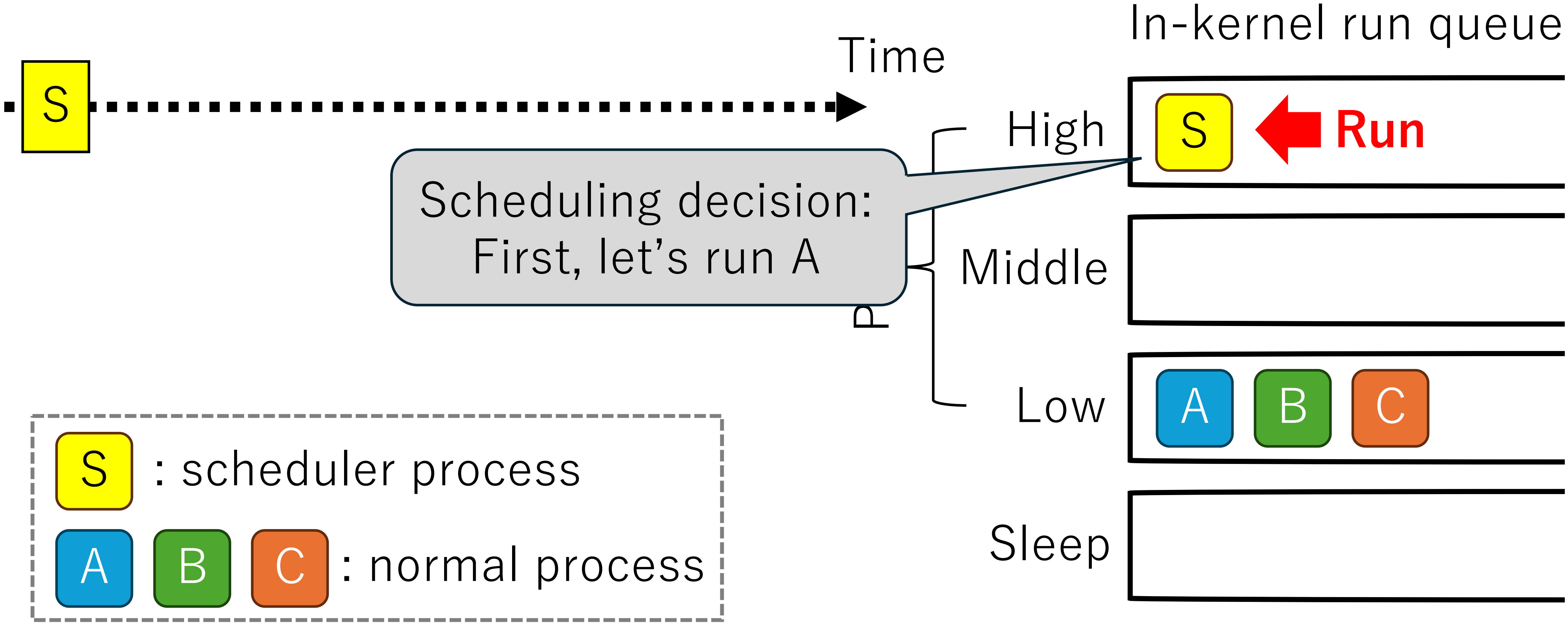
The nature of SCHED_FIFO



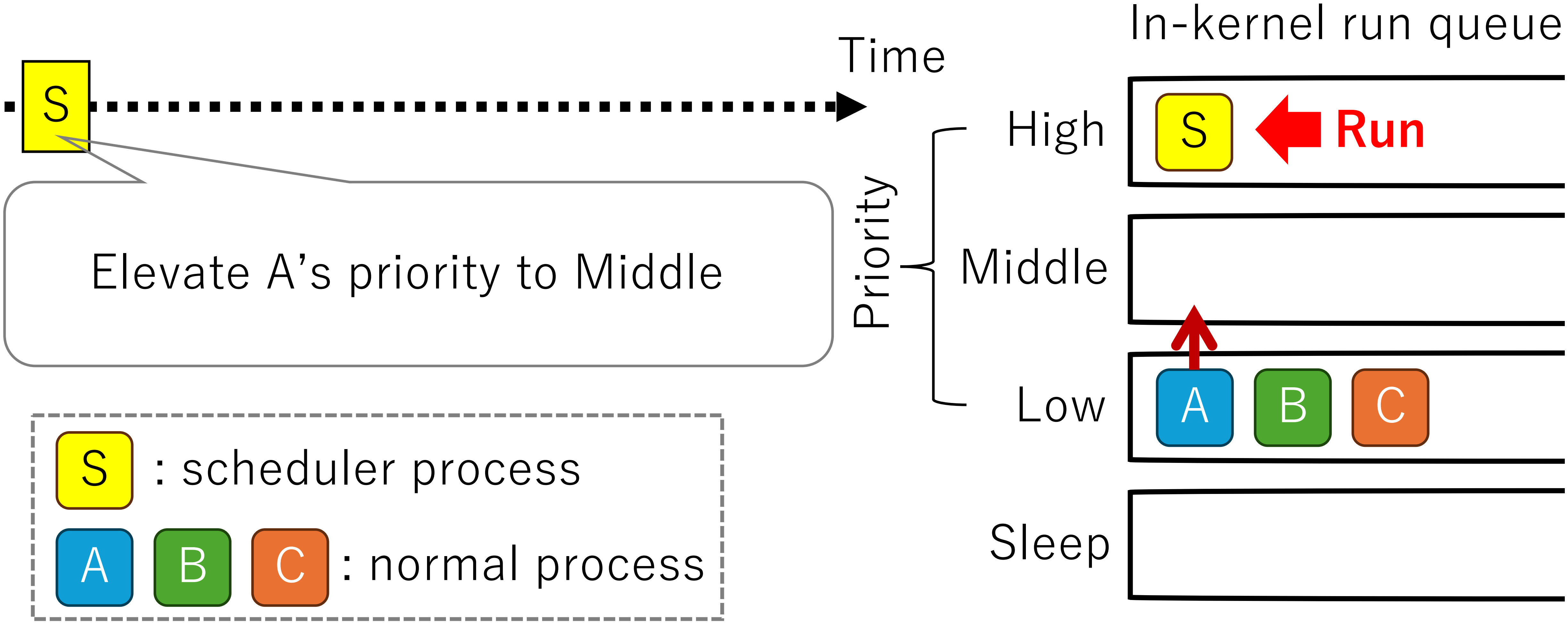
Priority Elevation（Shared Pattern）



Priority Elevation (Shared Pattern)



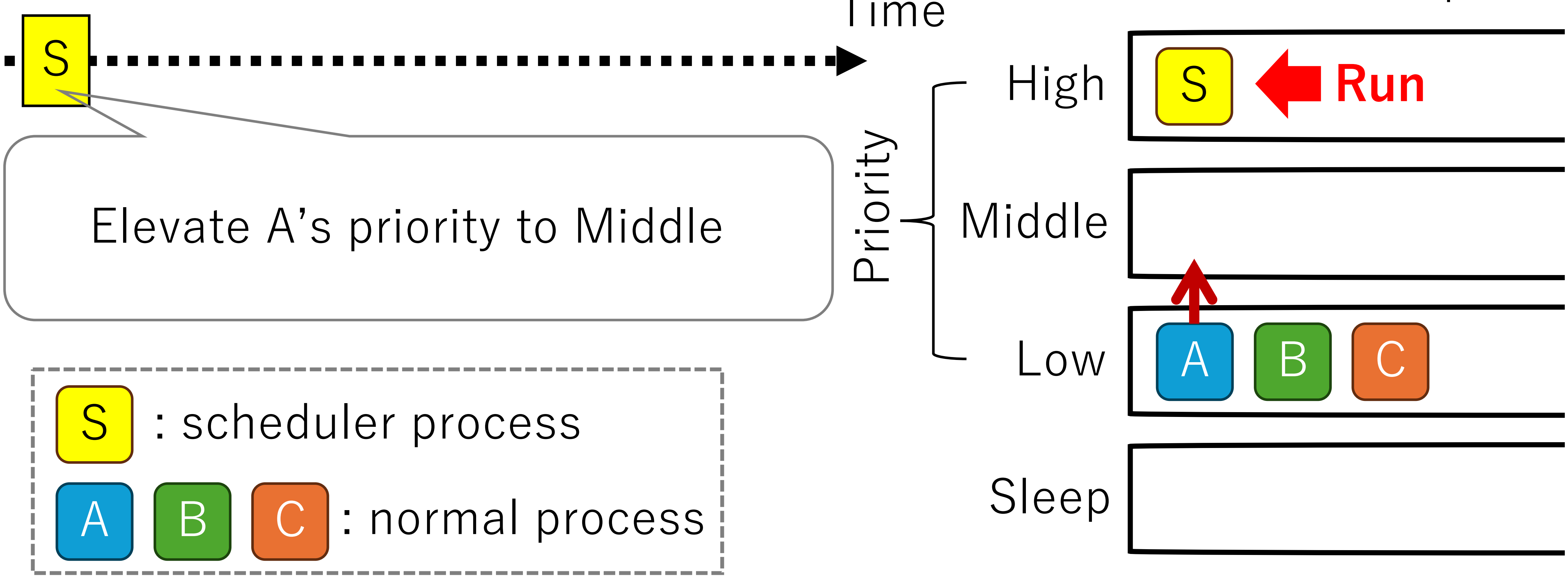
Priority Elevation (Shared Pattern)



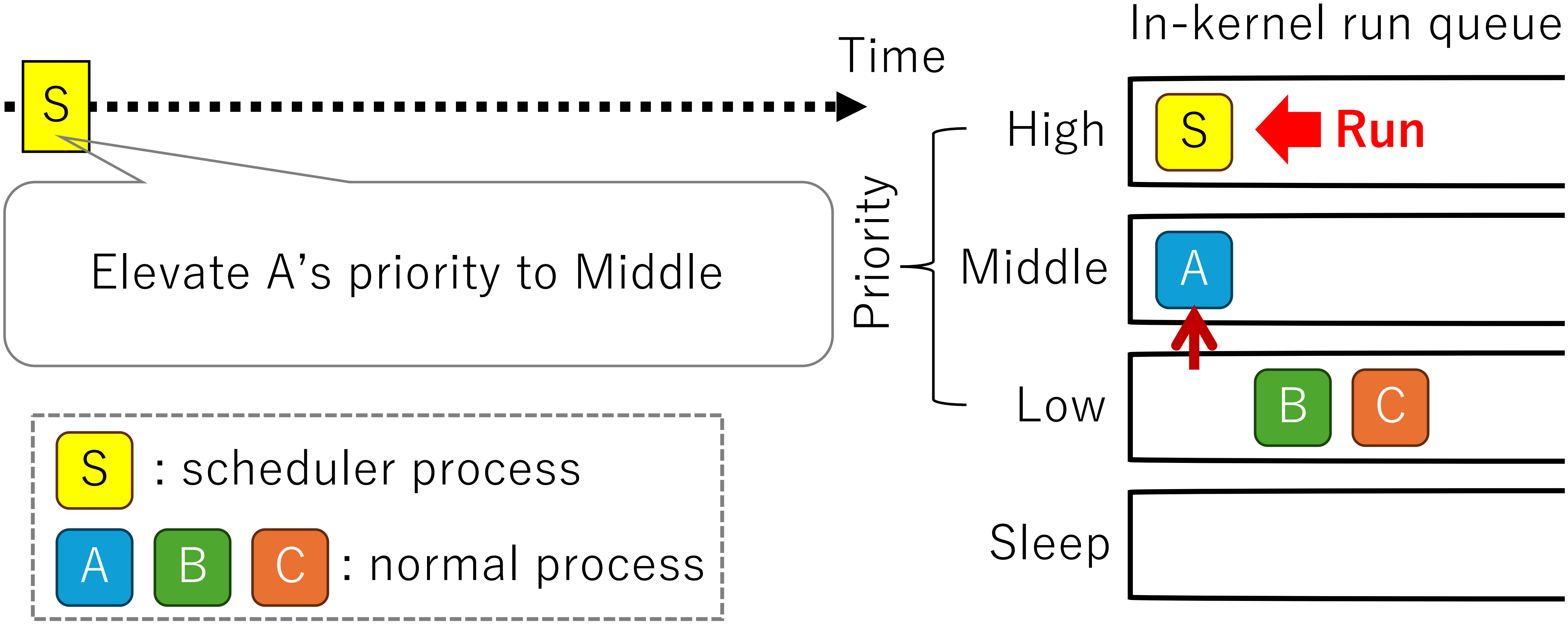
Priority Elevation (Shared Pattern)

We use **sched_setscheduler** for this priority manipulation

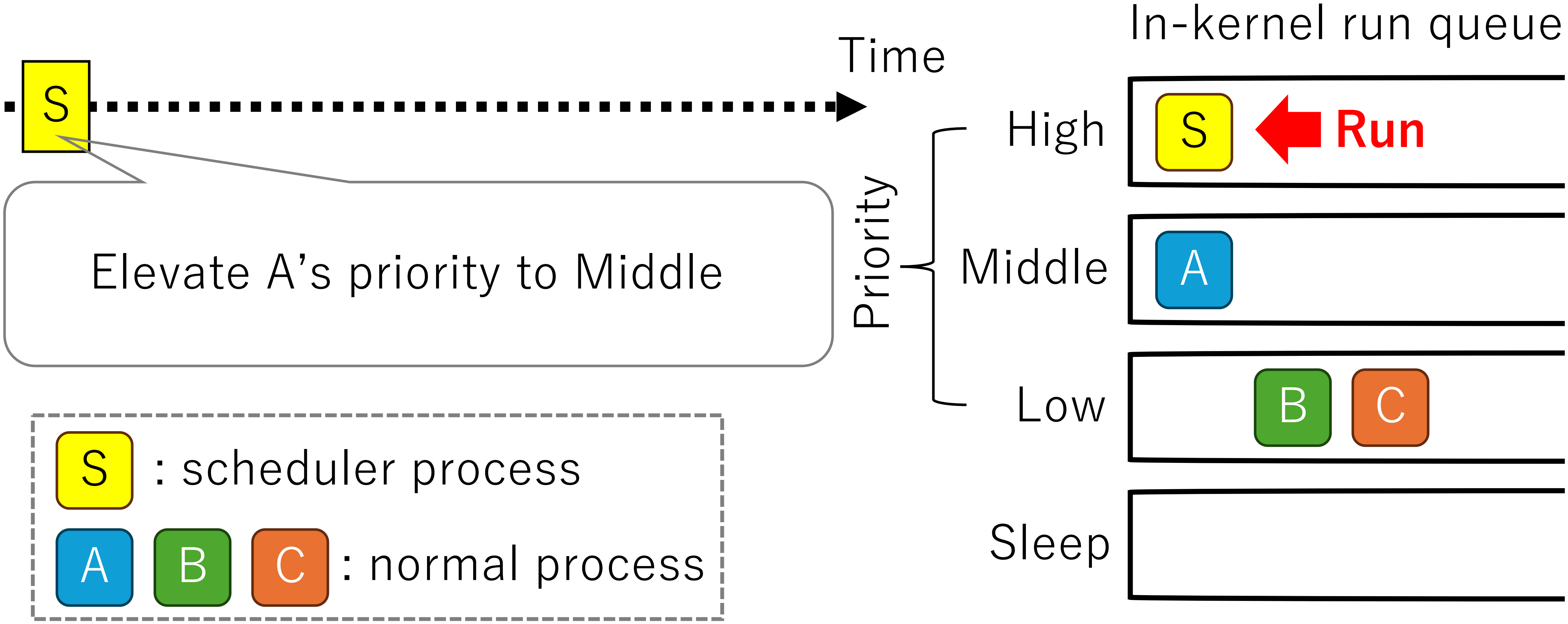
In-kernel run queue



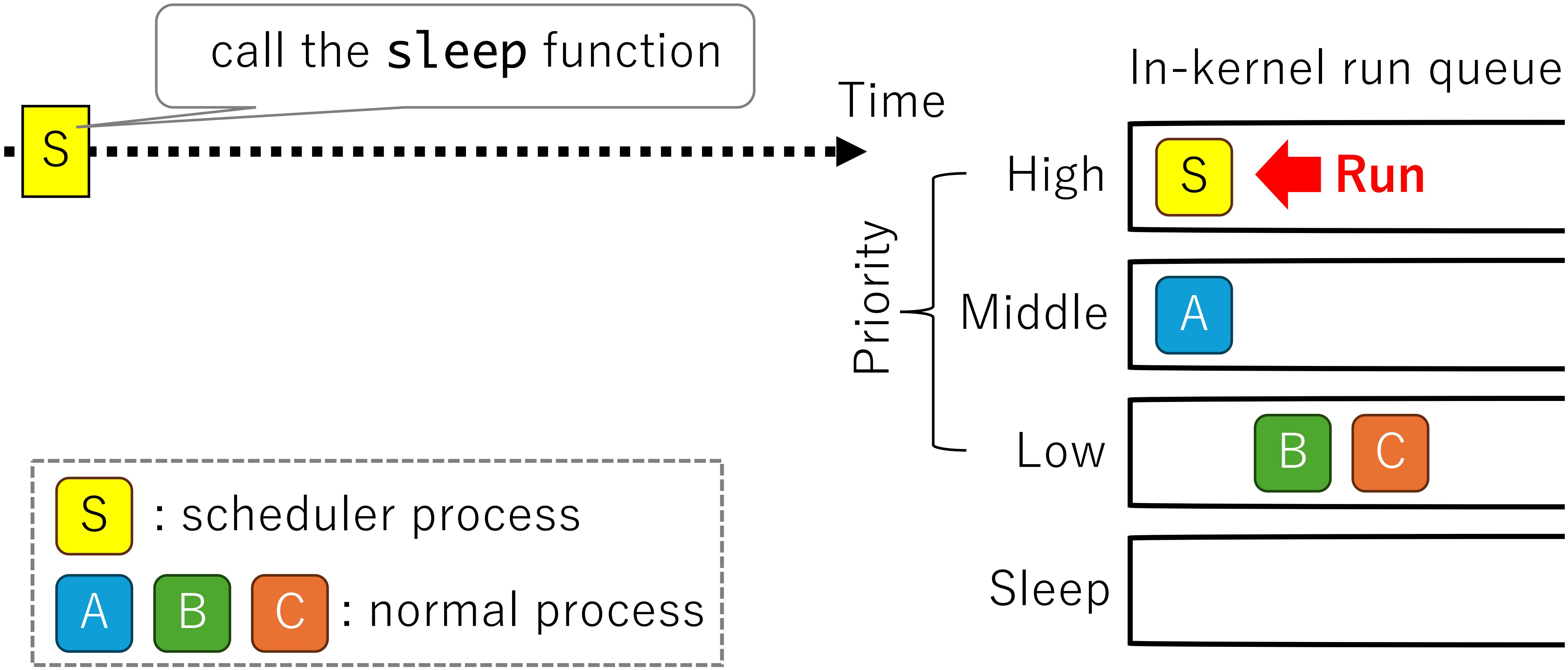
Priority Elevation (Shared Pattern)



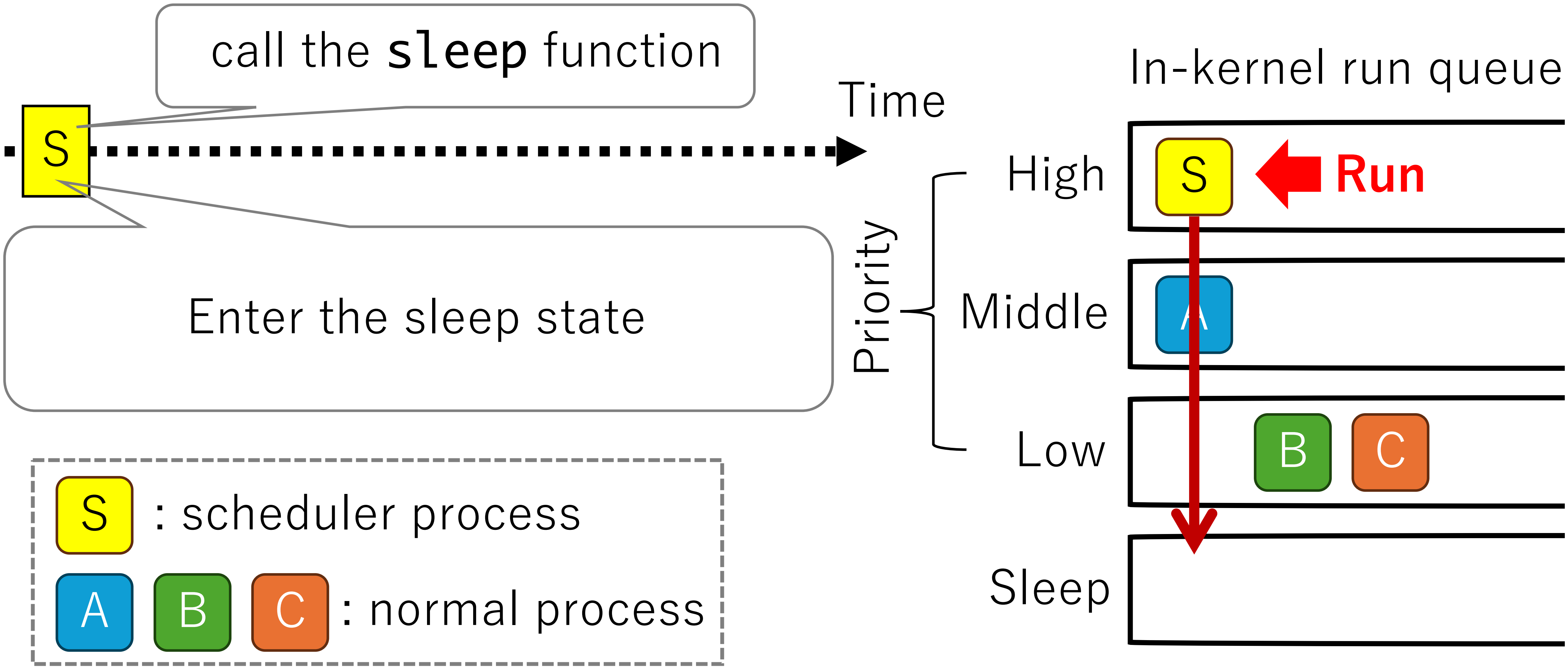
Priority Elevation (Shared Pattern)



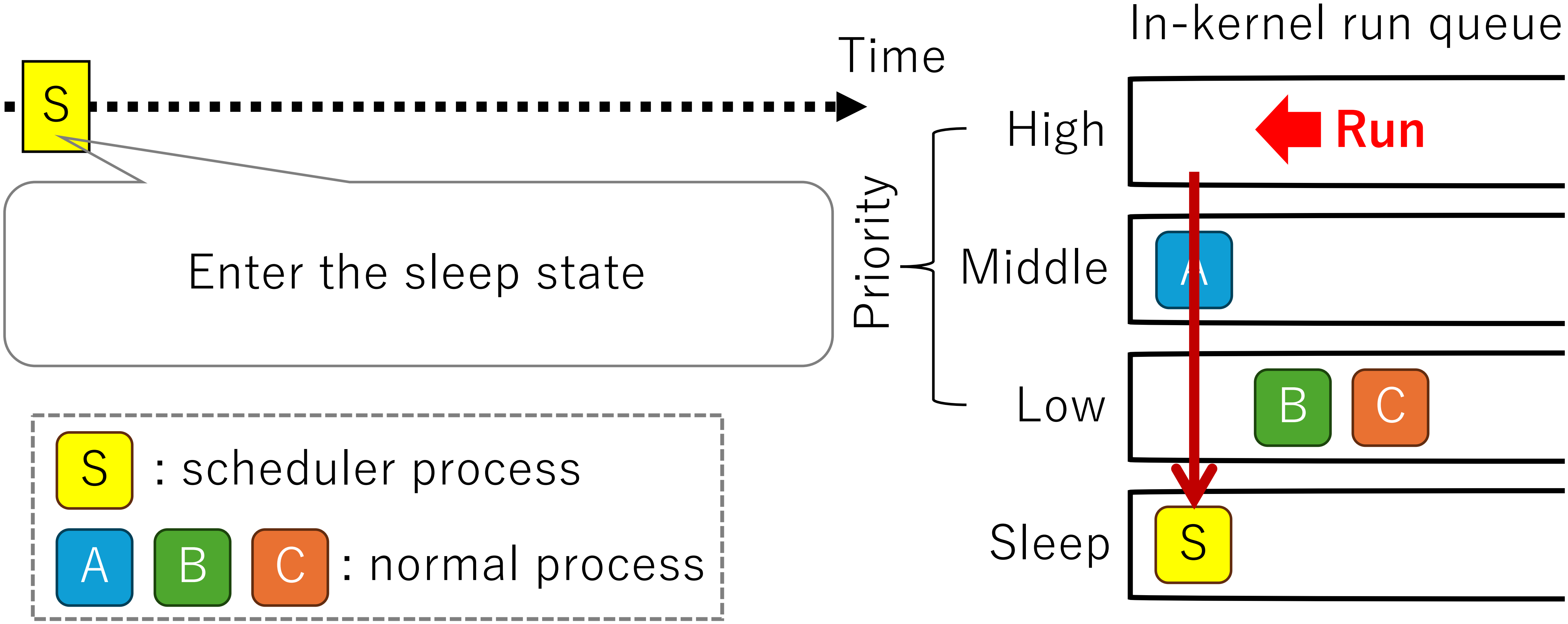
Priority Elevation（Shared Pattern）



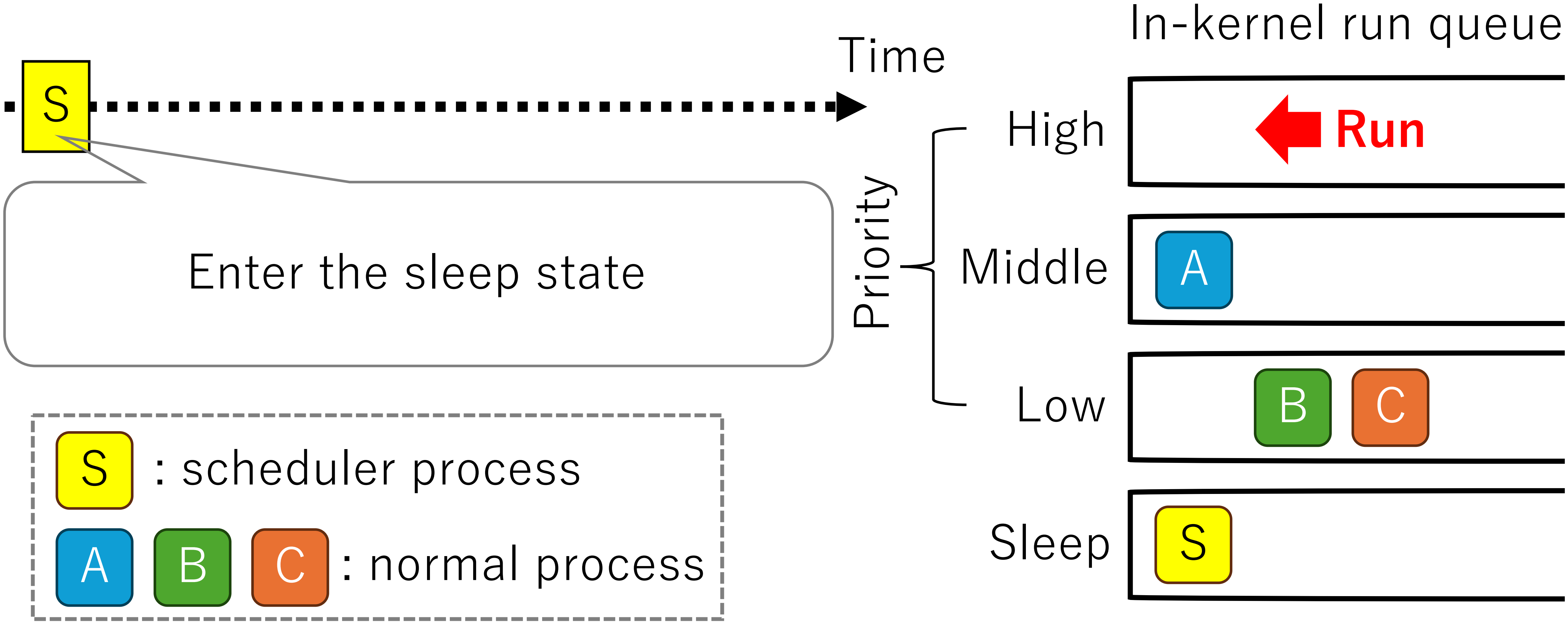
Priority Elevation (Shared Pattern)



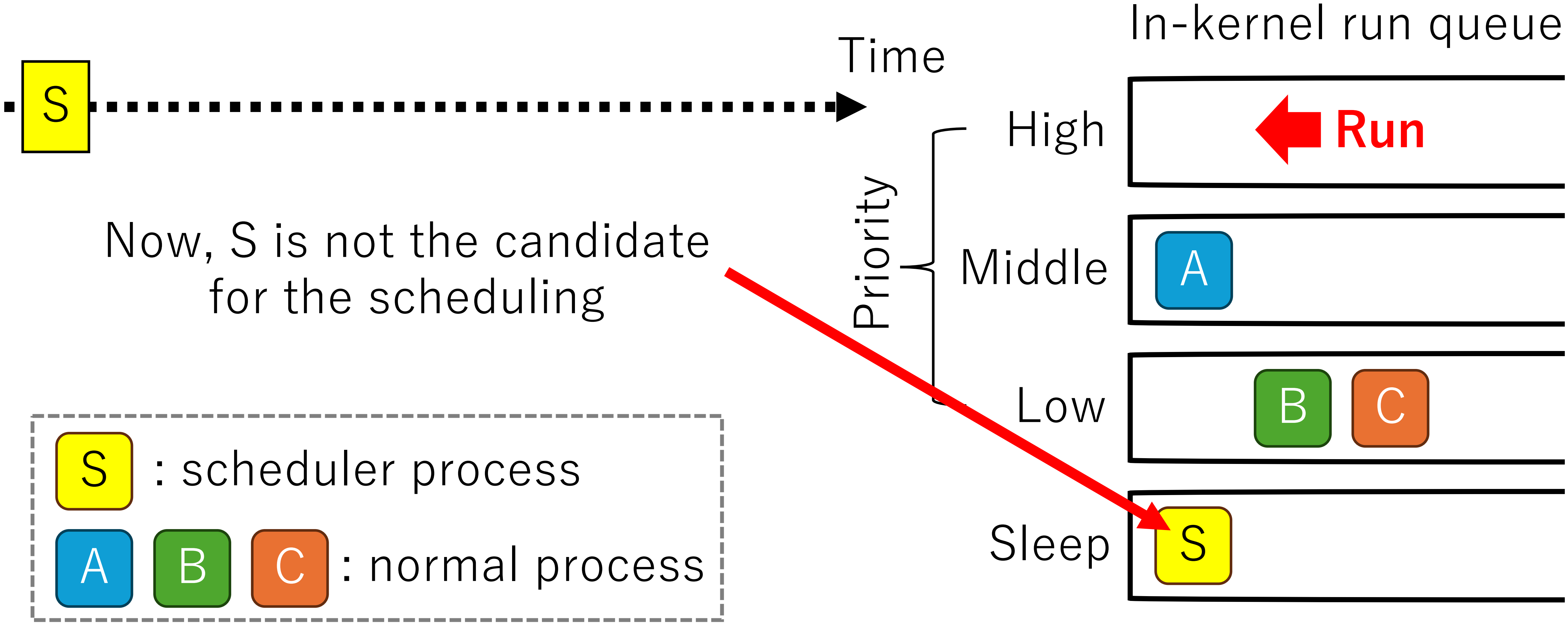
Priority Elevation (Shared Pattern)



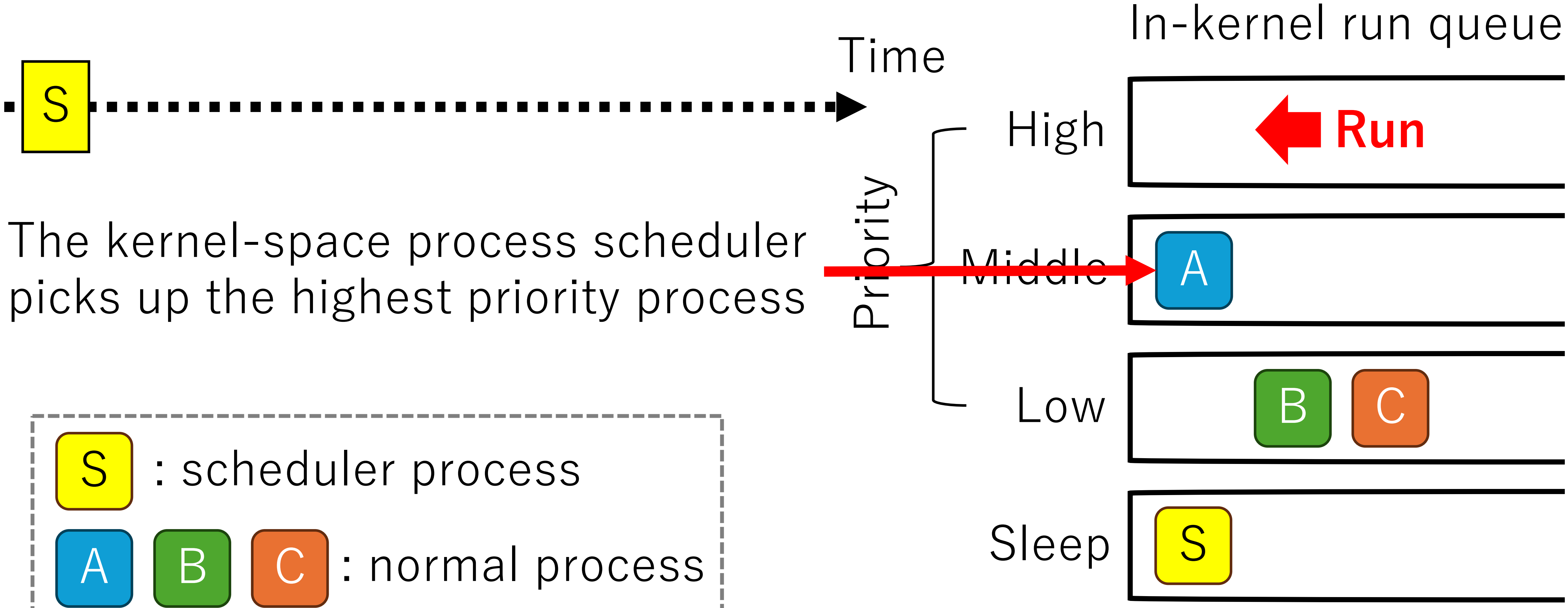
Priority Elevation（Shared Pattern)



Priority Elevation（Shared Pattern)



Priority Elevation（Shared Pattern)

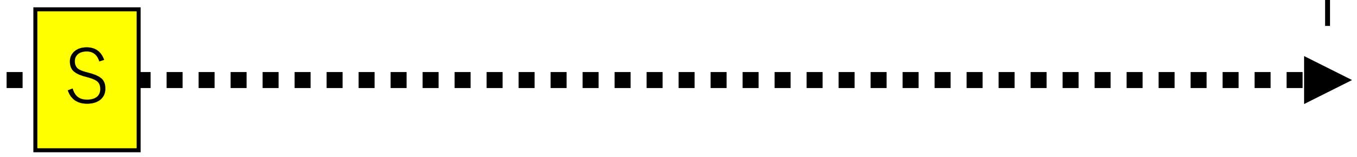


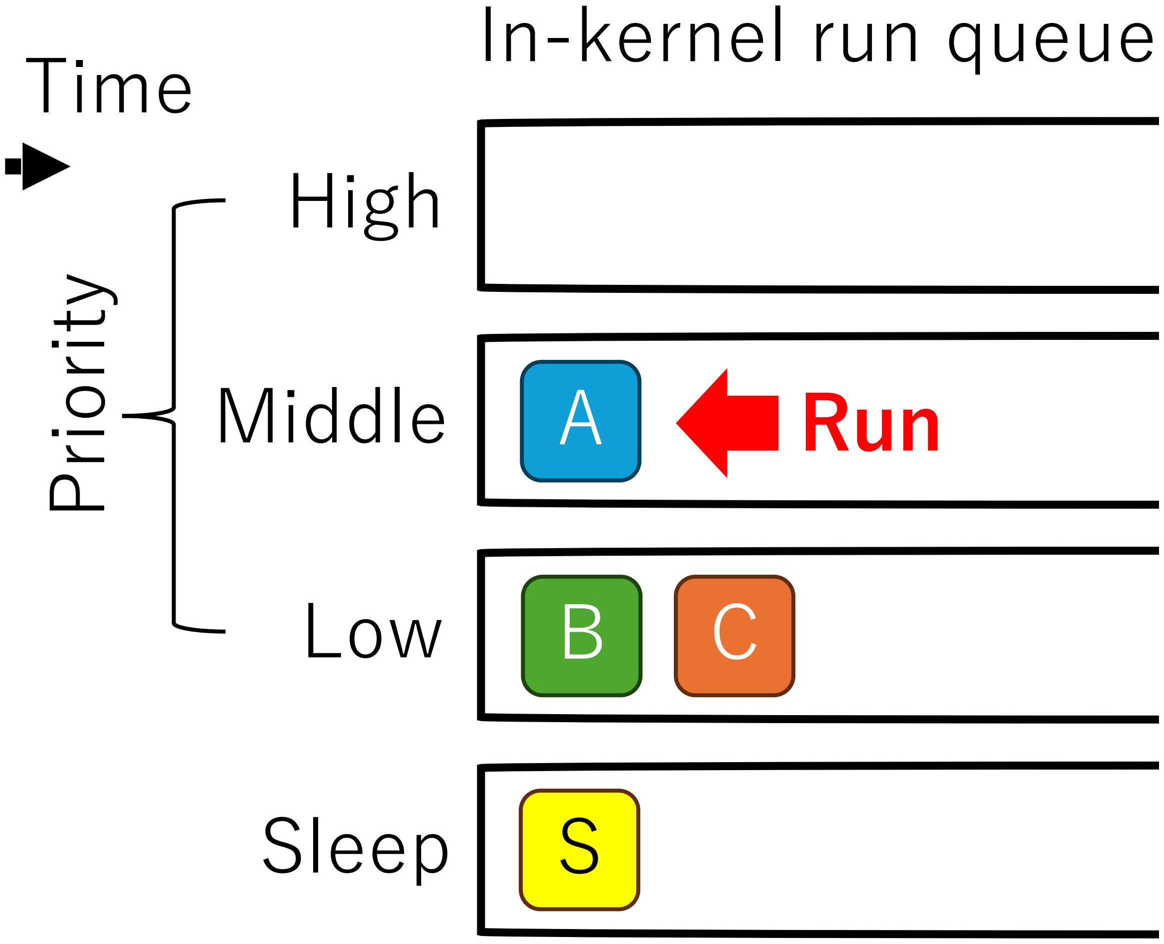
The kernel-space process scheduler picks up the highest priority process

S : scheduler process

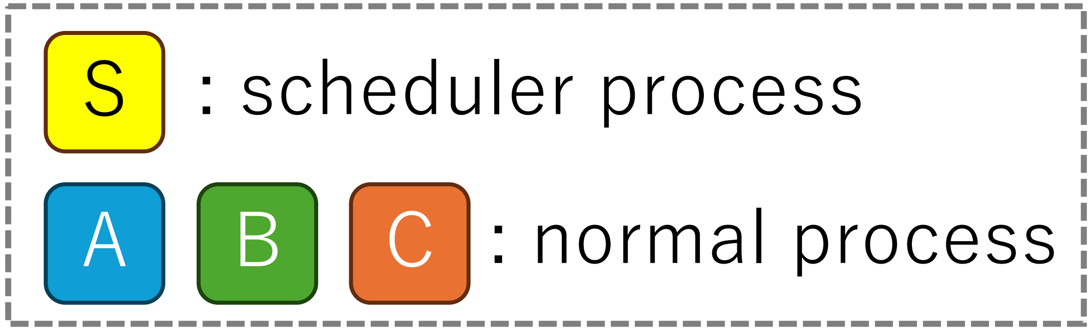
A B C : normal process

Priority Elevation (Shared Pattern)

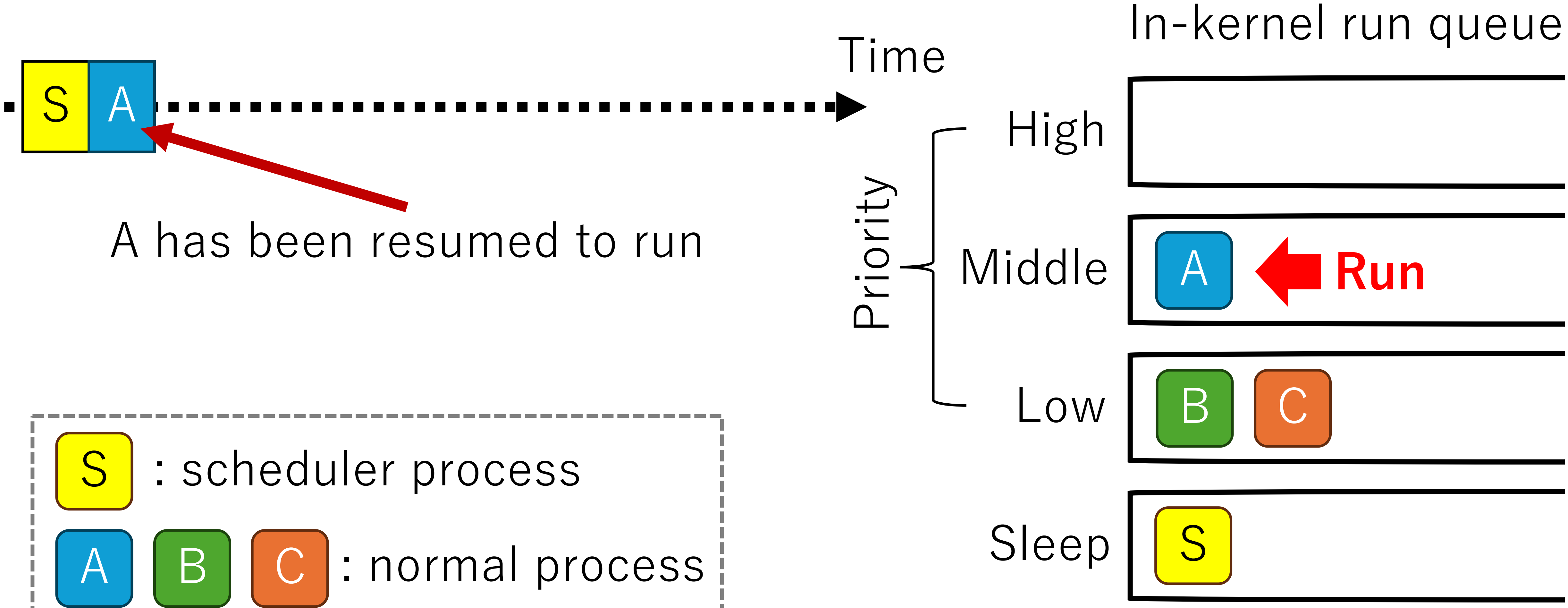




Execution is switched from S to A



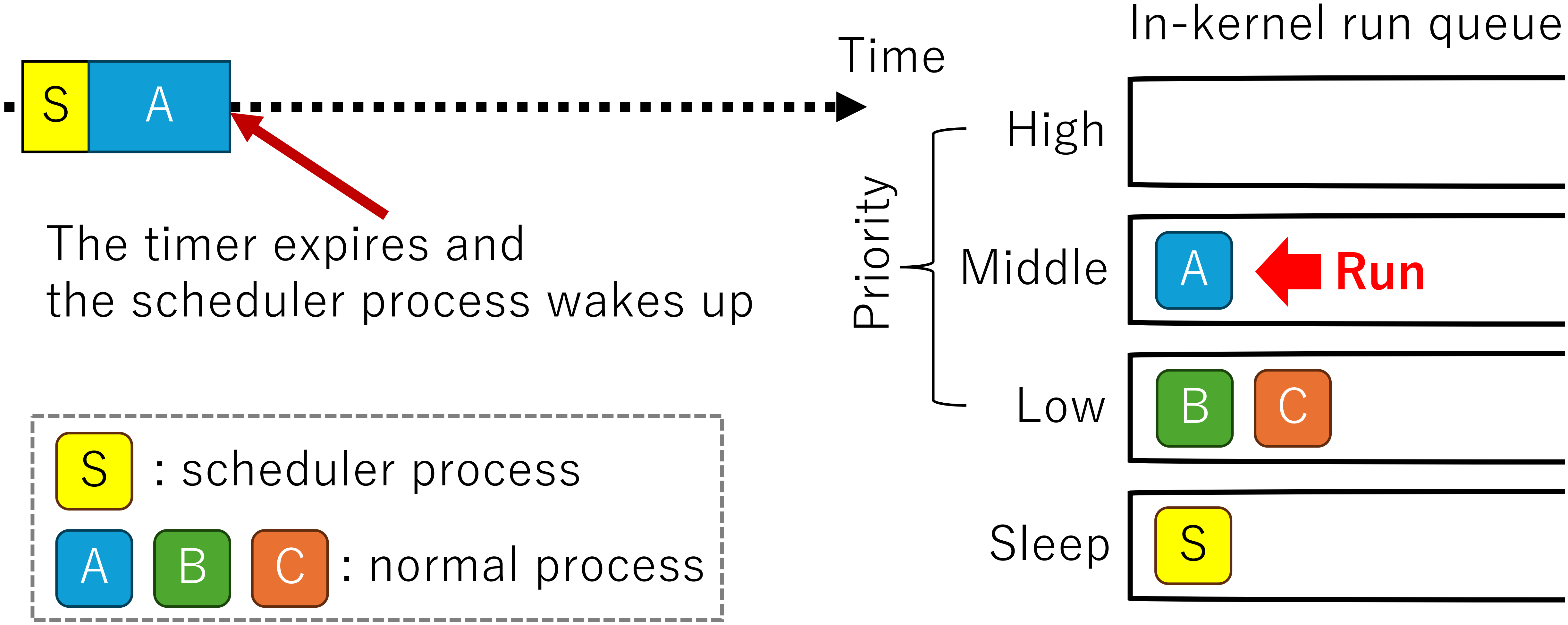
Priority Elevation（Shared Pattern）



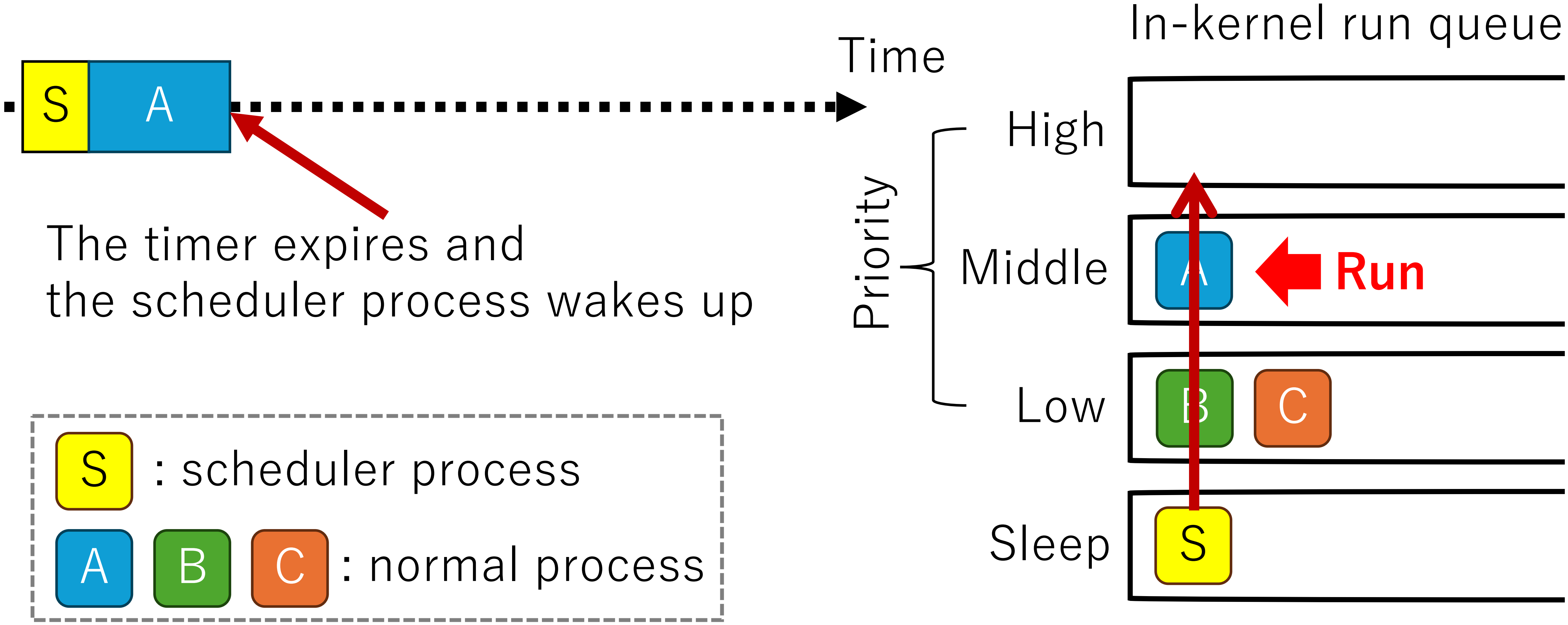
S : scheduler process

A B C : normal process

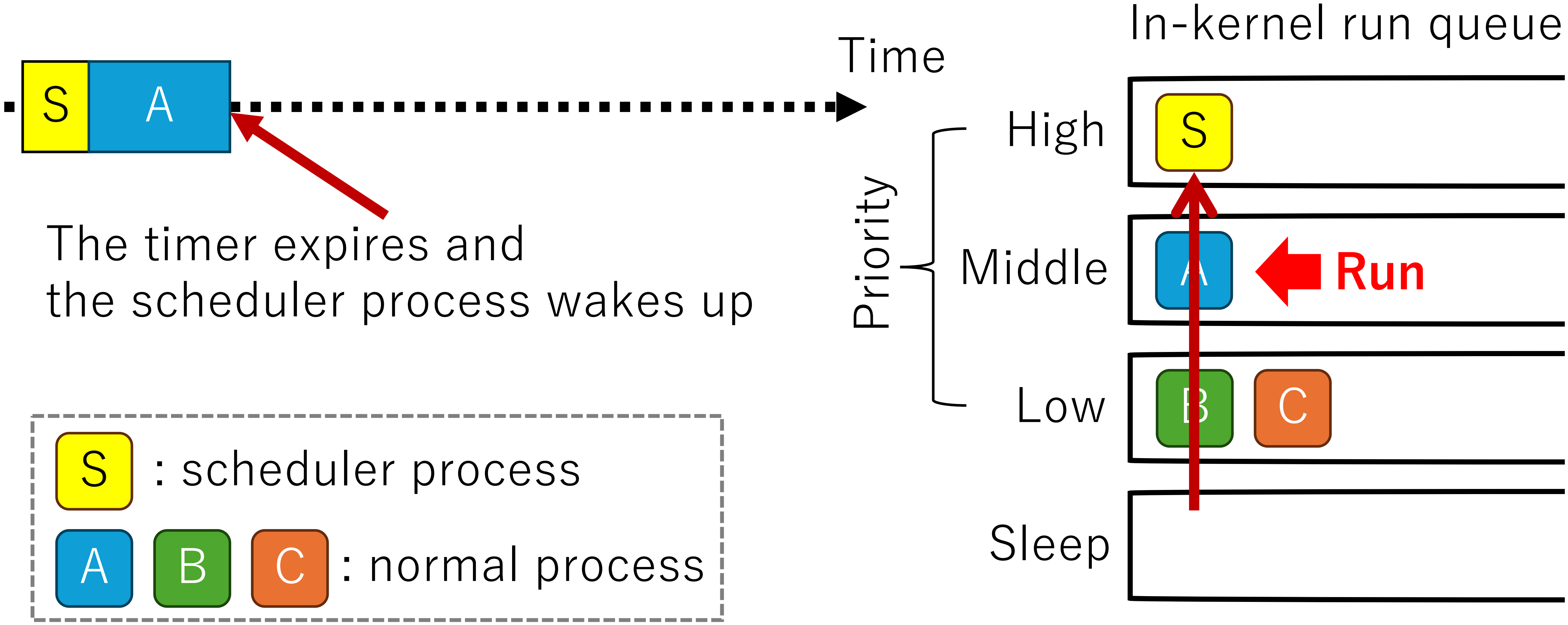
Priority Elevation (Shared Pattern)



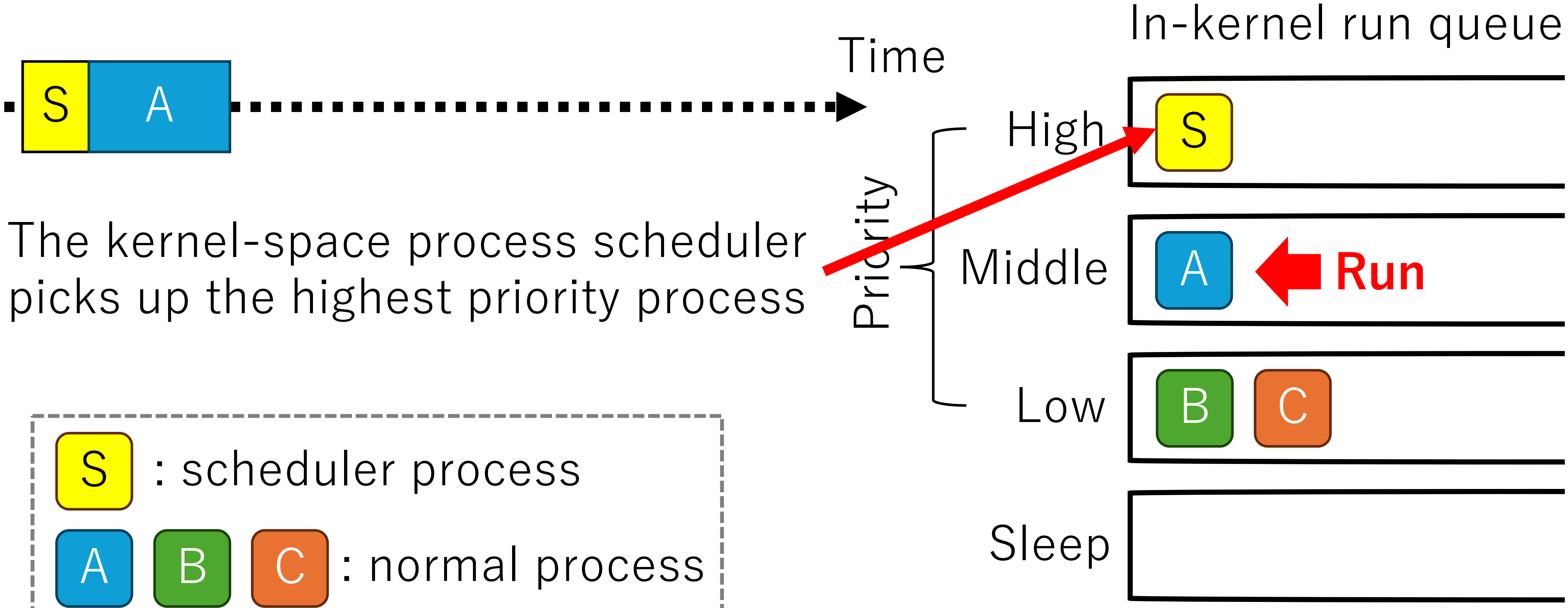
Priority Elevation（Shared Pattern）



Priority Elevation（Shared Pattern）



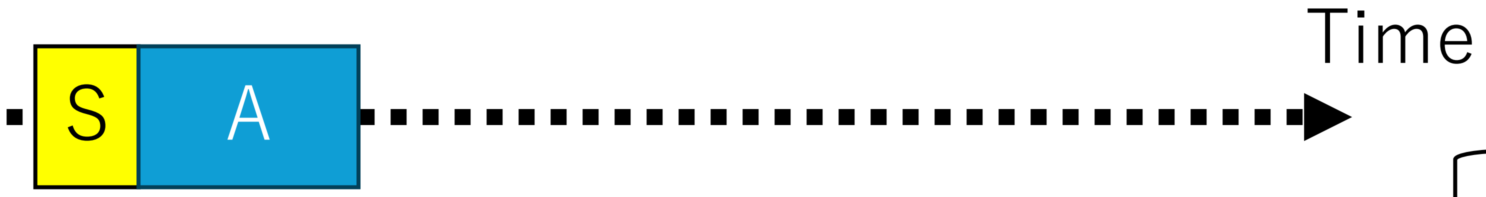
Priority Elevation（Shared Pattern)



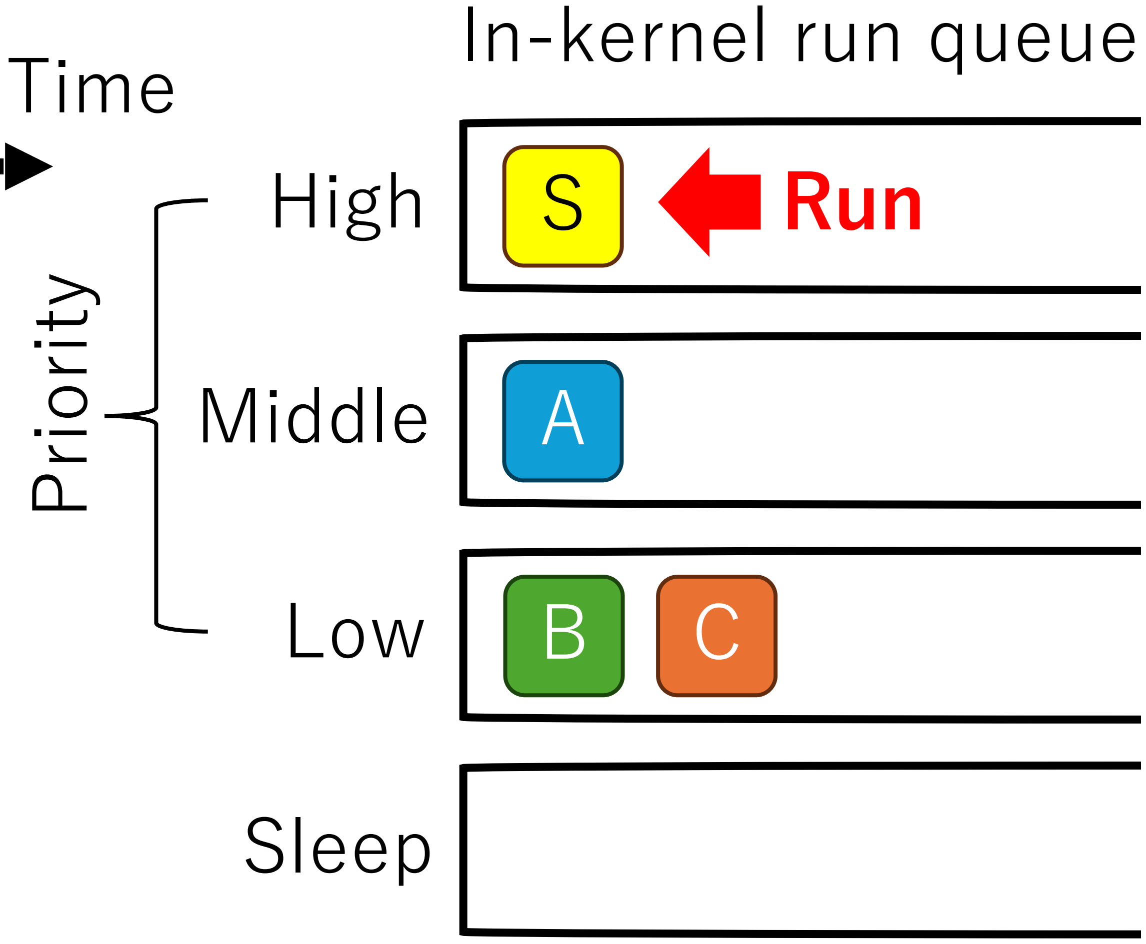
S : scheduler process

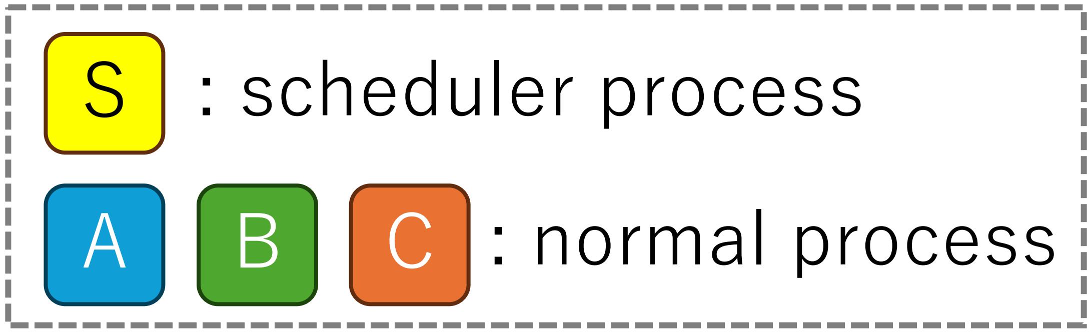
A B C : normal process

Priority Elevation (Shared Pattern)

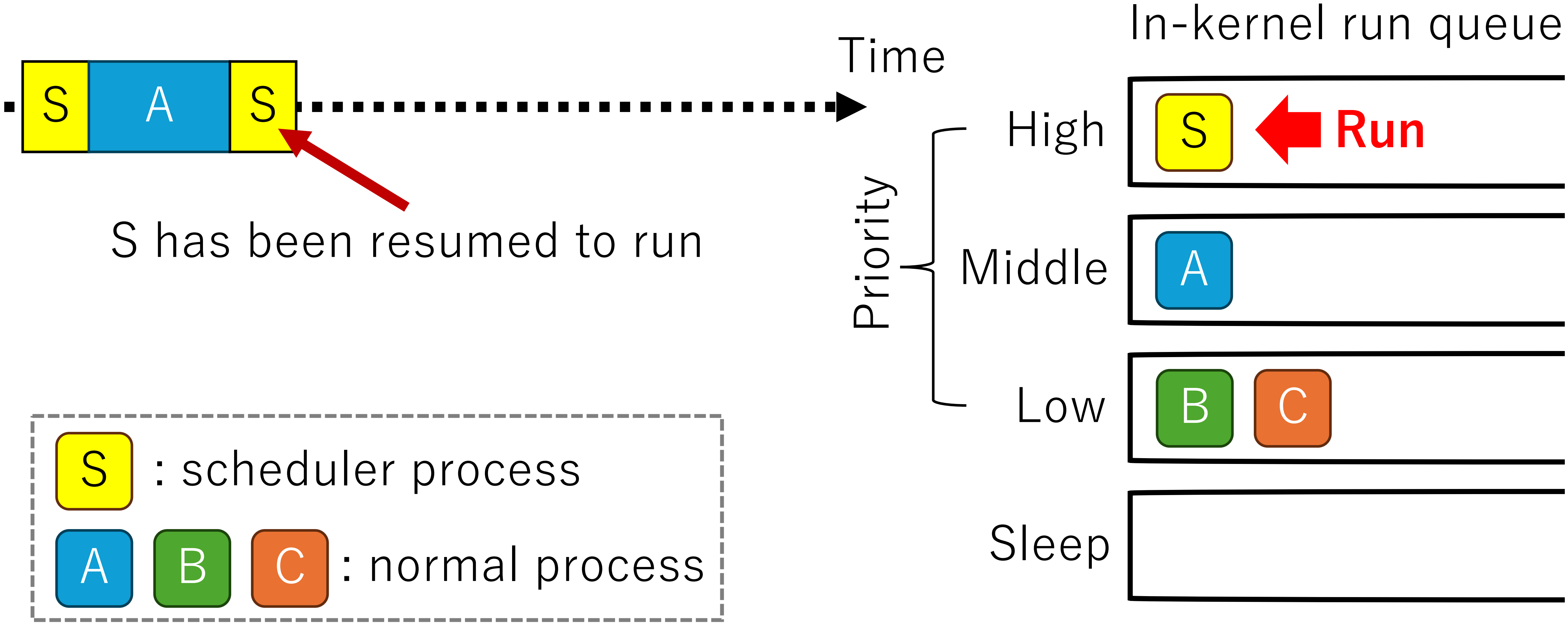


Execution is switched from A to S

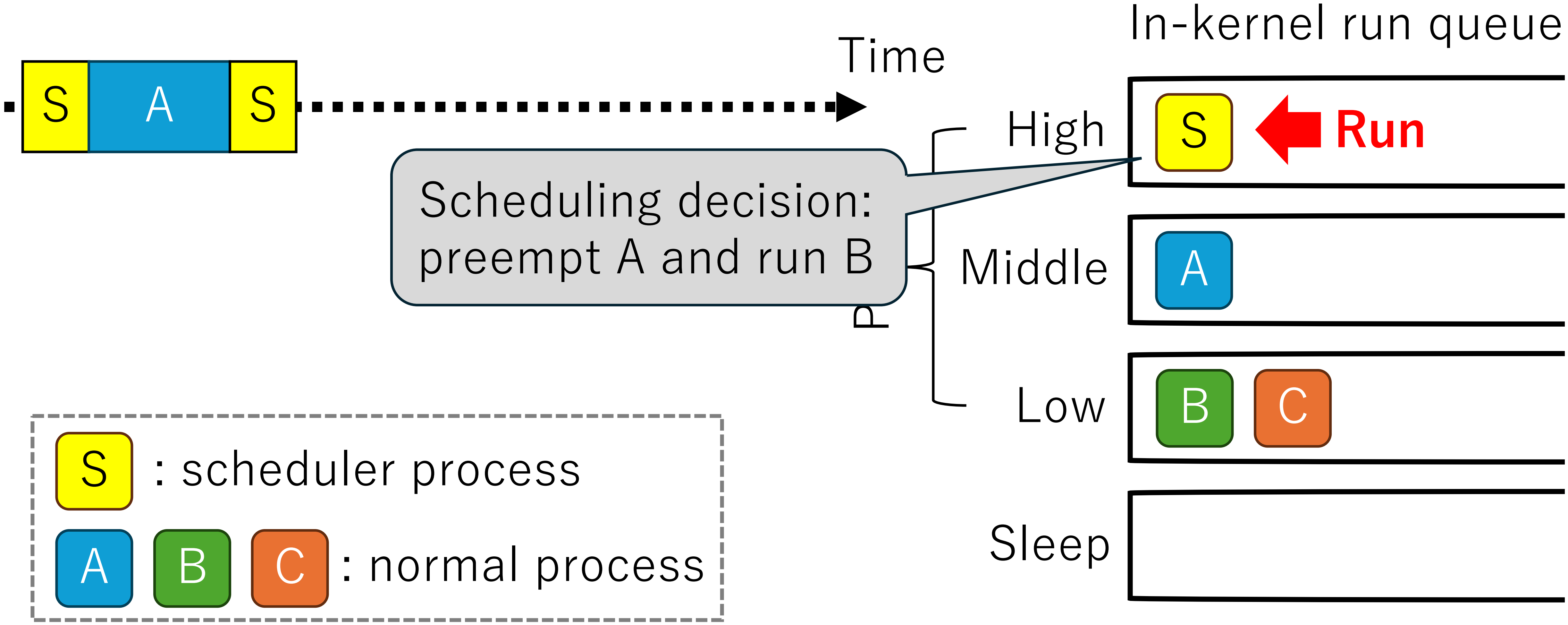




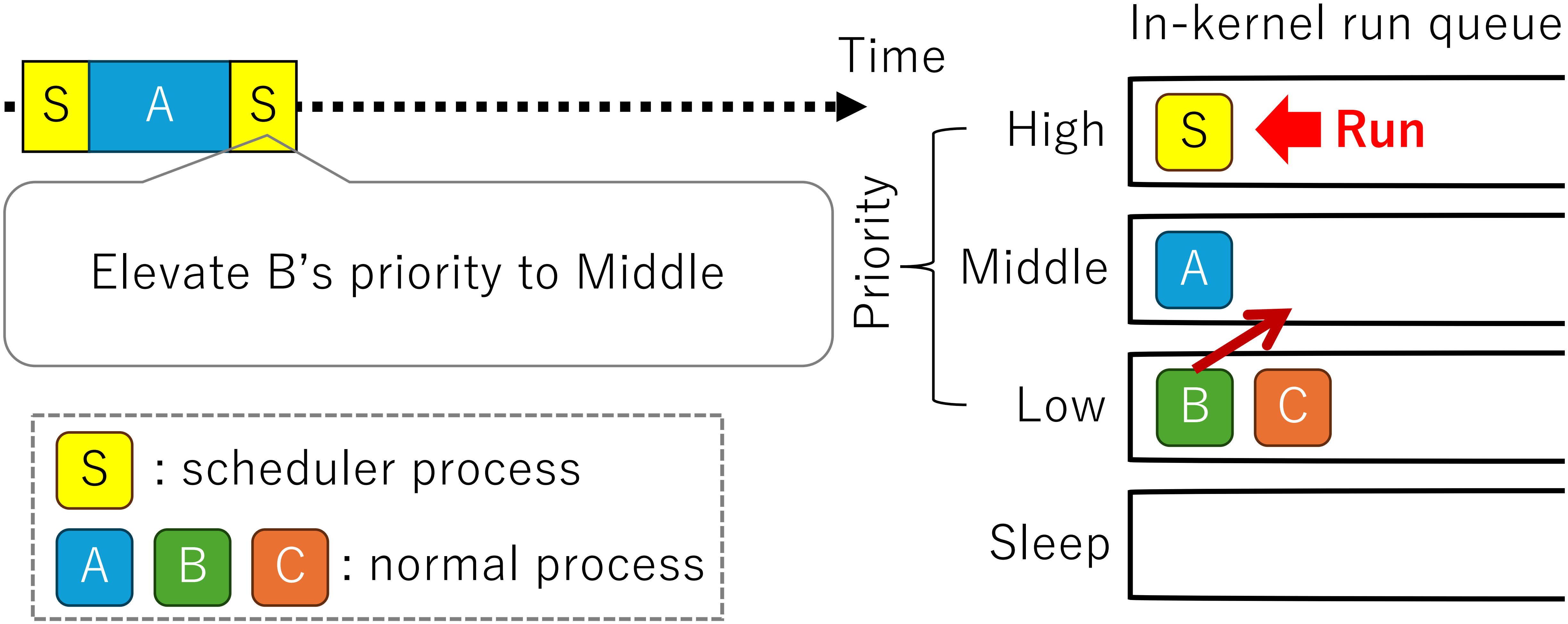
Priority Elevation（Shared Pattern）



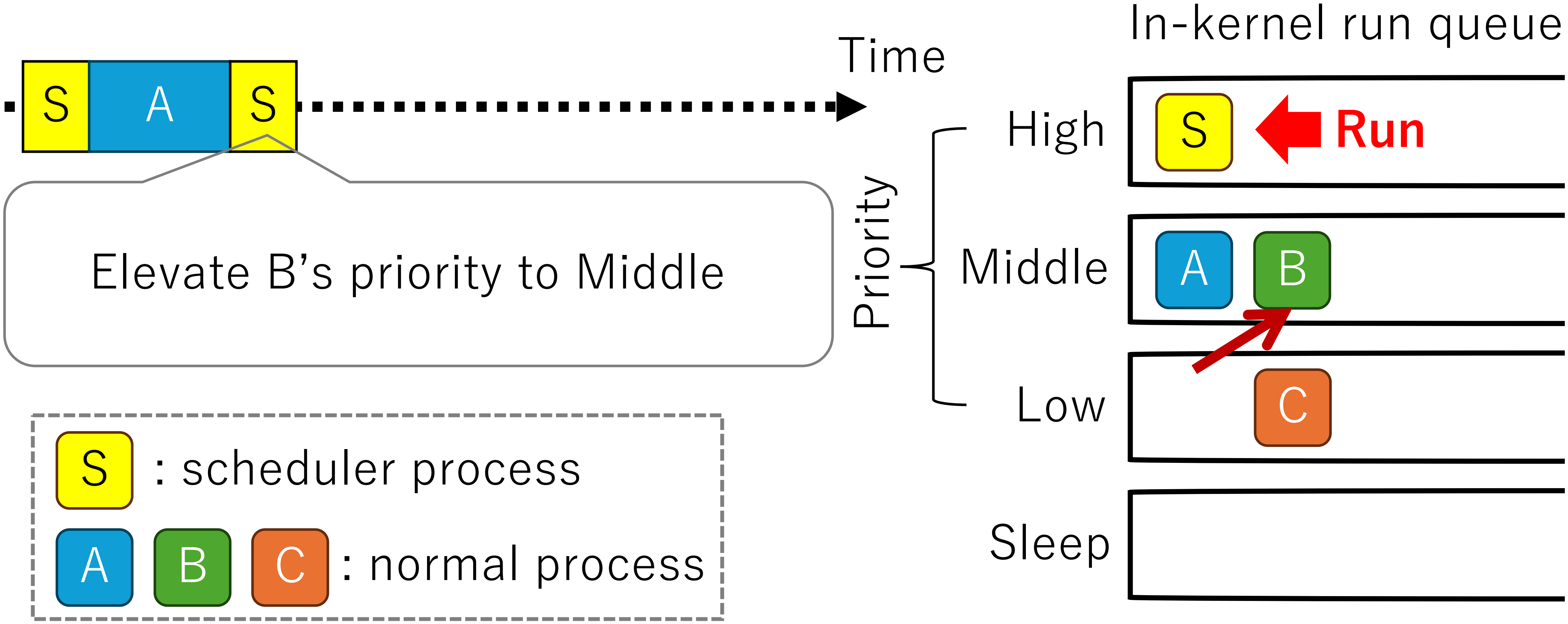
Priority Elevation (Shared Pattern)



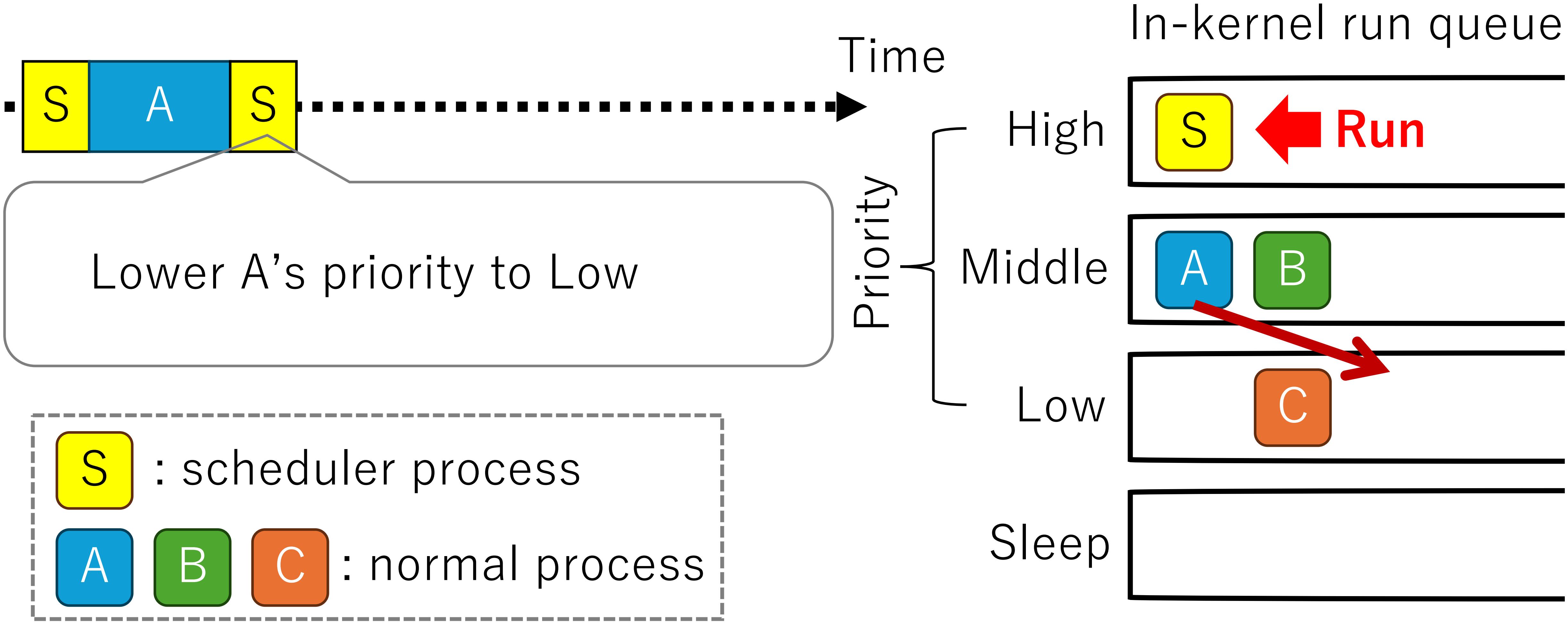
Priority Elevation (Shared Pattern)



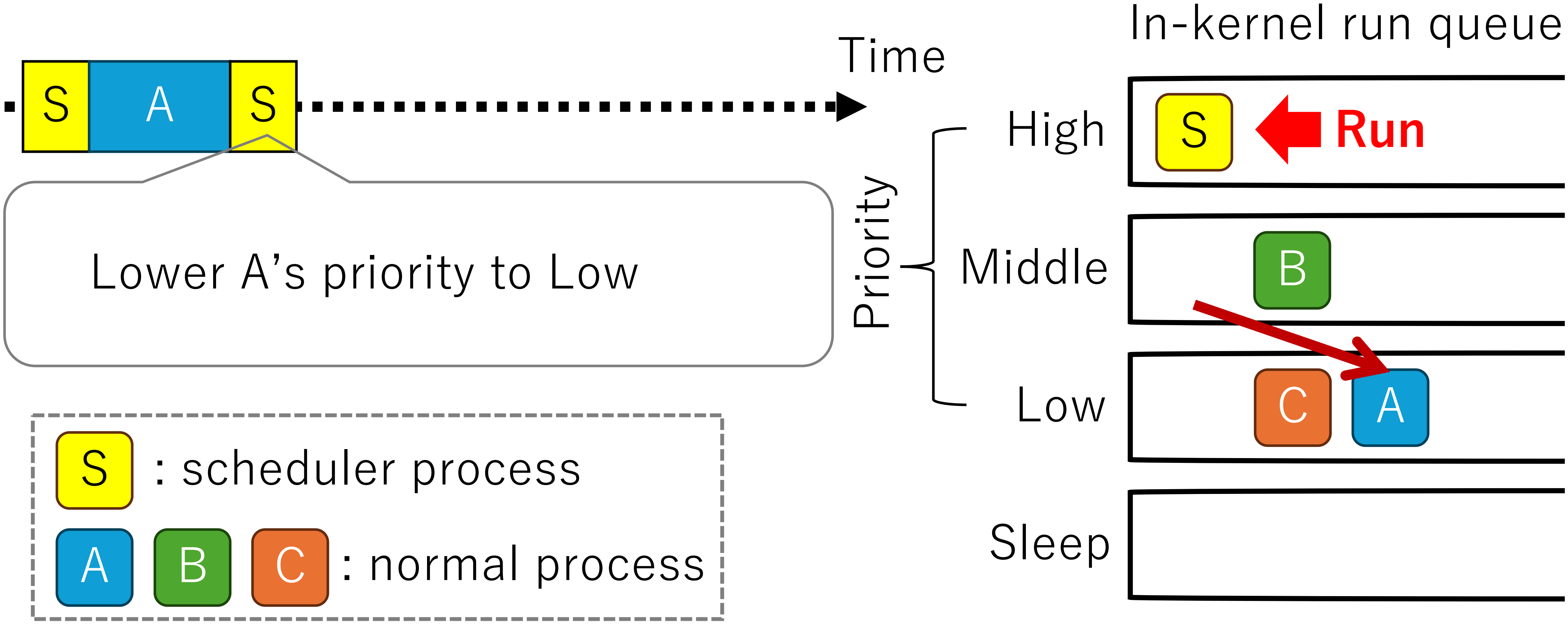
Priority Elevation (Shared Pattern)



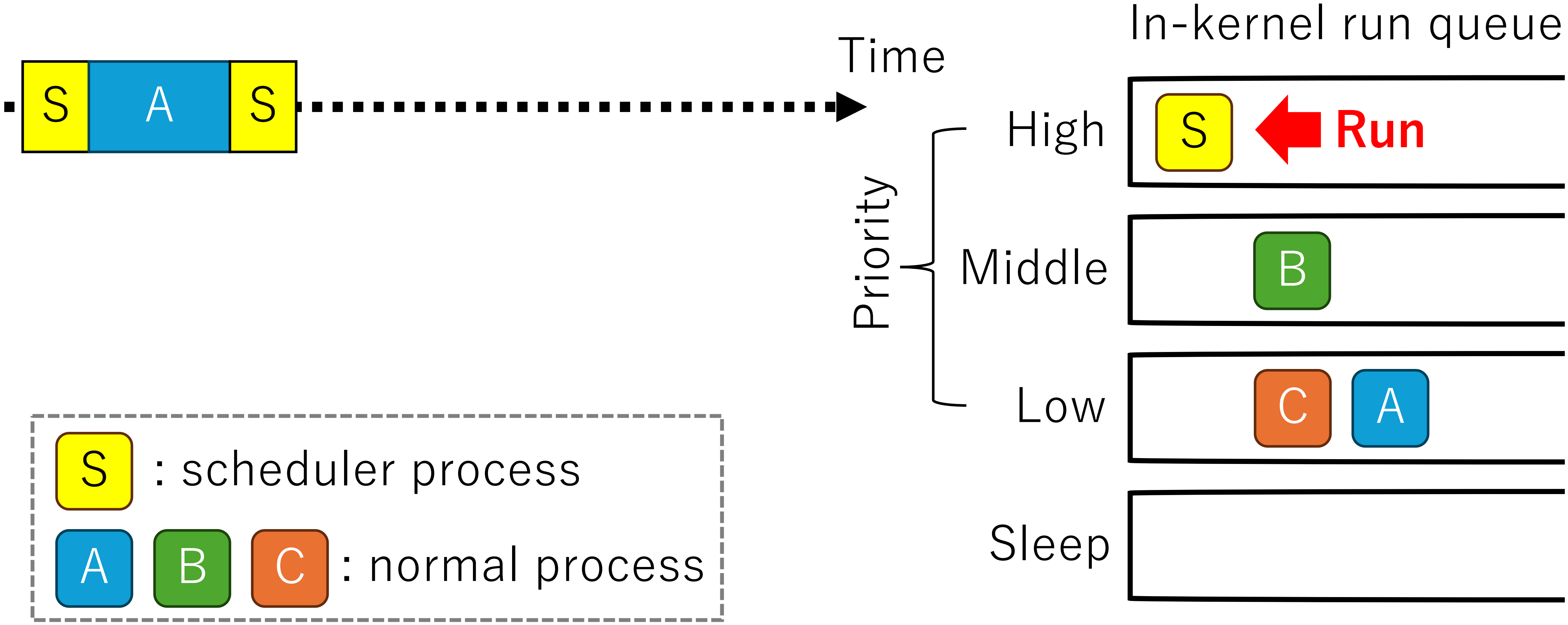
Priority Elevation (Shared Pattern)



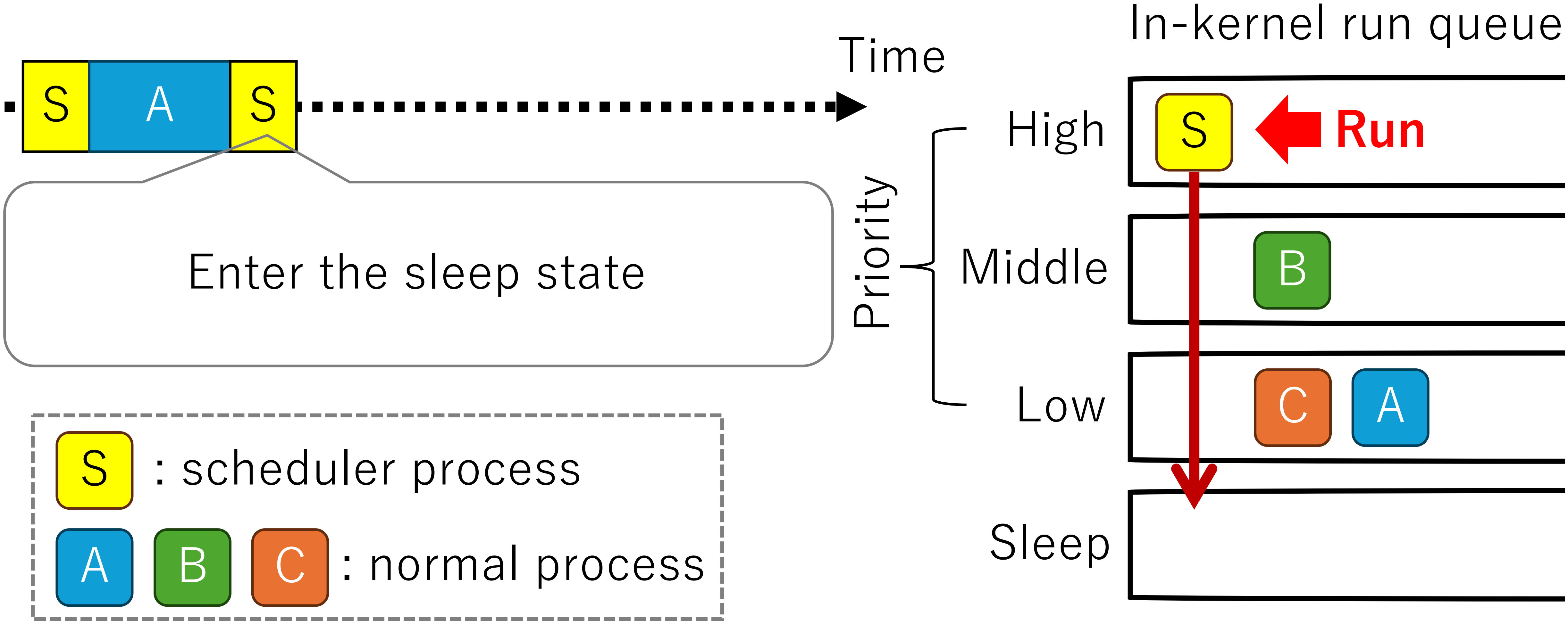
Priority Elevation (Shared Pattern)



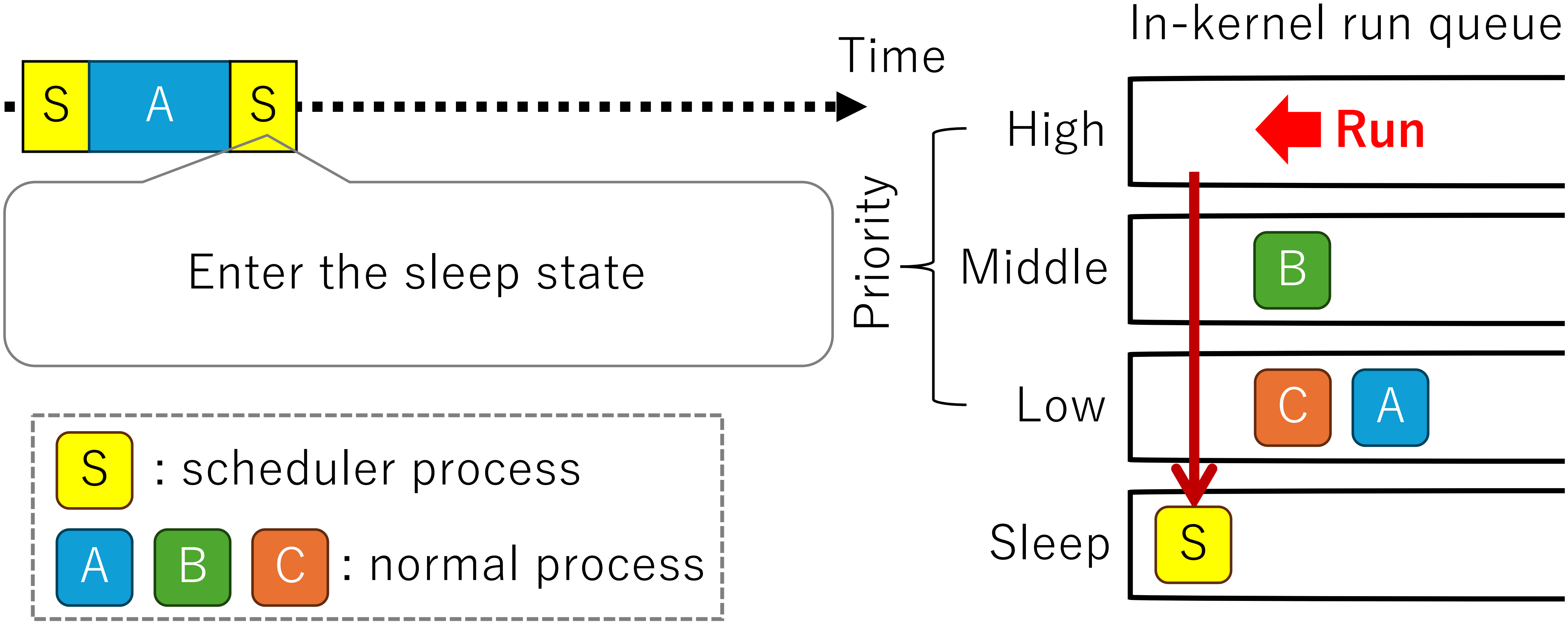
Priority Elevation (Shared Pattern)



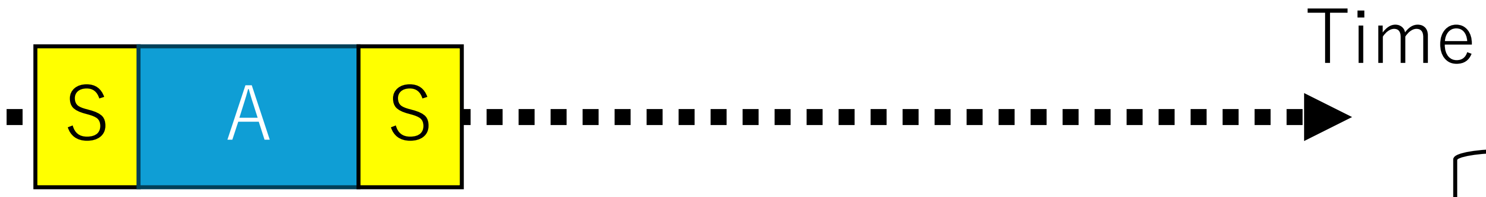
Priority Elevation（Shared Pattern）

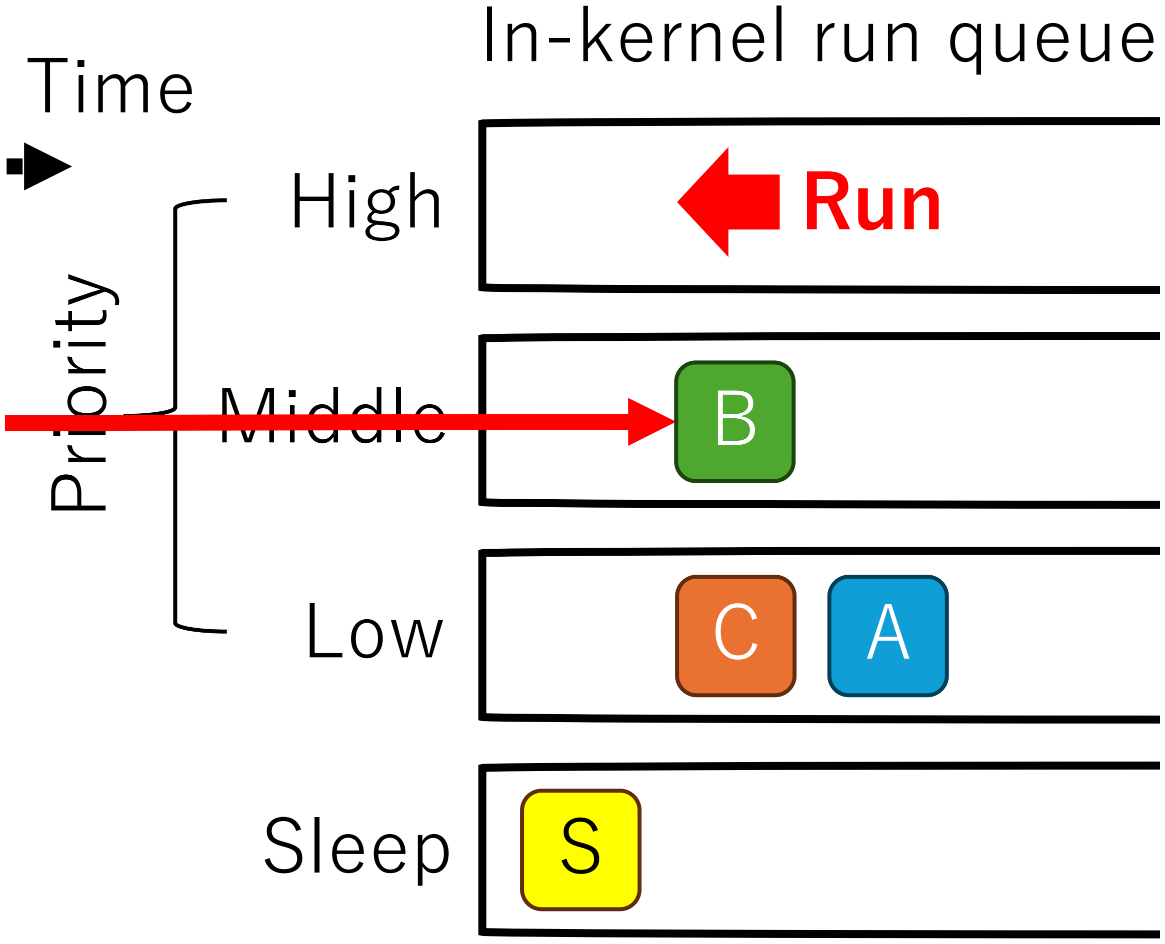


Priority Elevation（Shared Pattern)

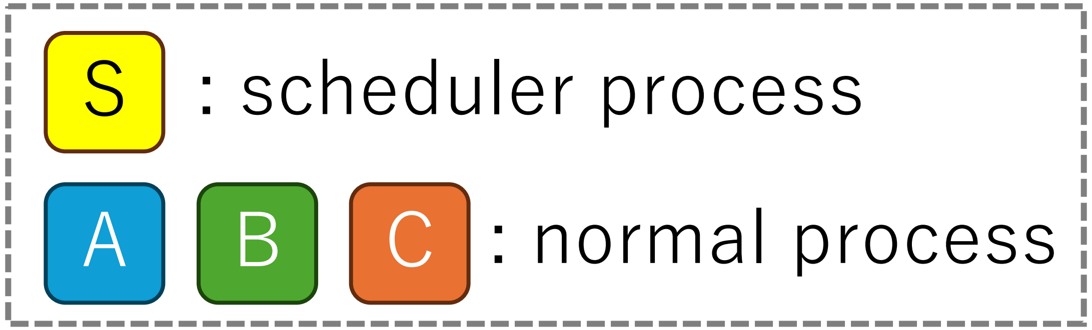


Priority Elevation (Shared Pattern)

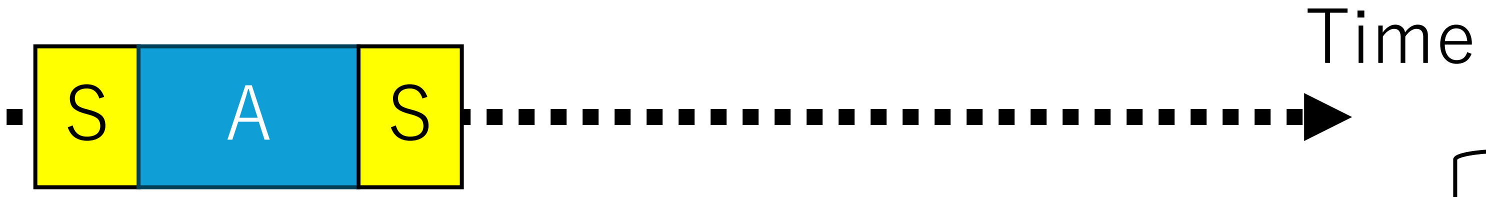




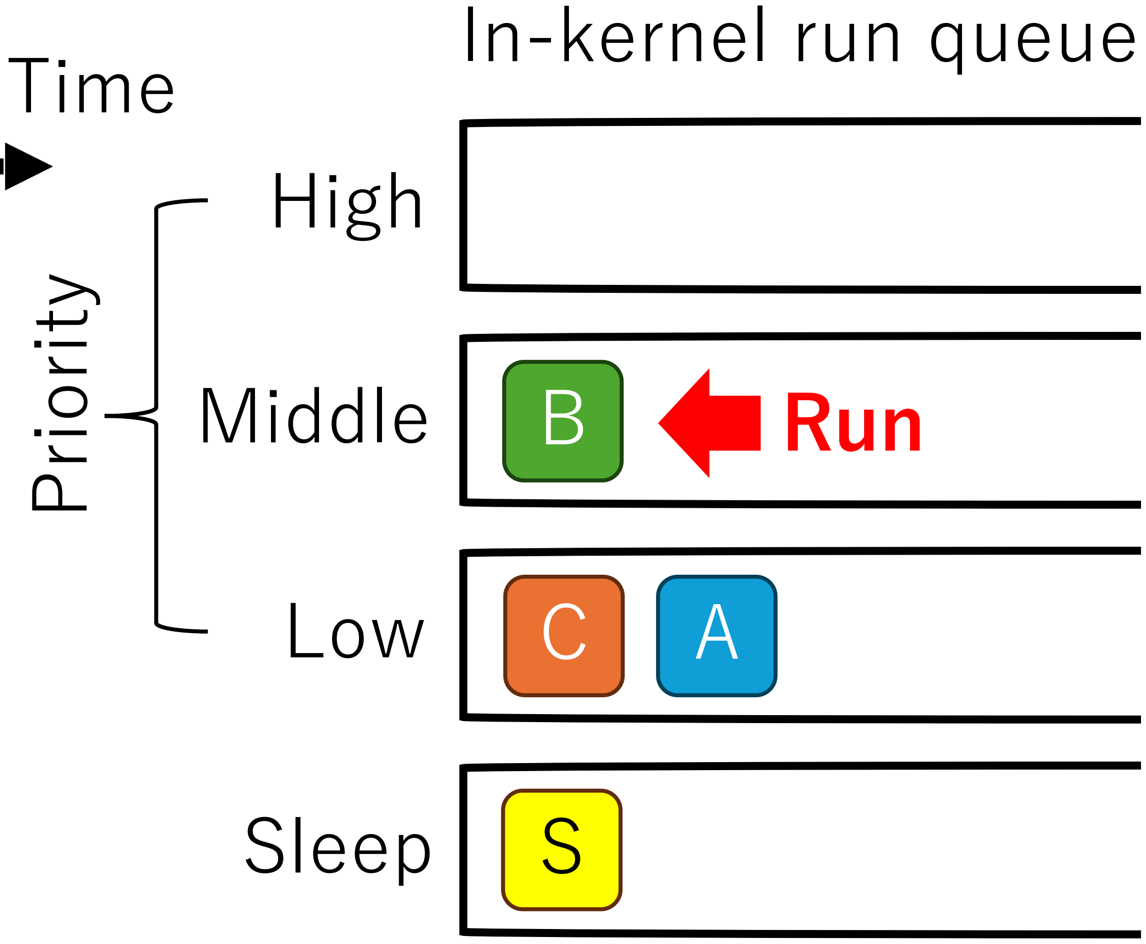
The kernel-space process scheduler picks up the highest priority process

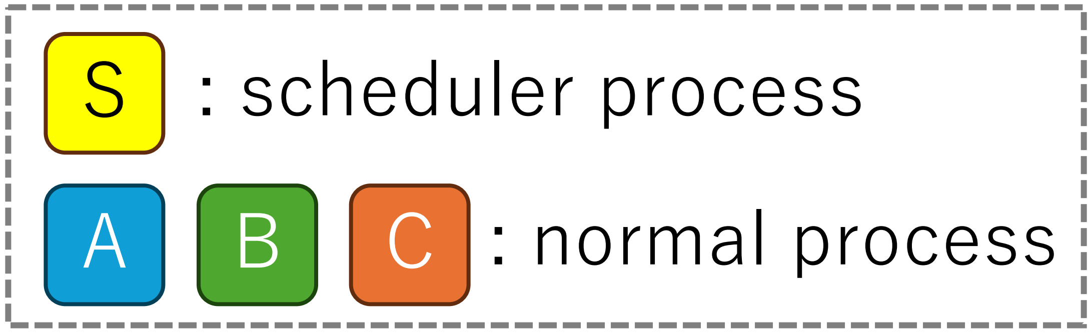


Priority Elevation (Shared Pattern)

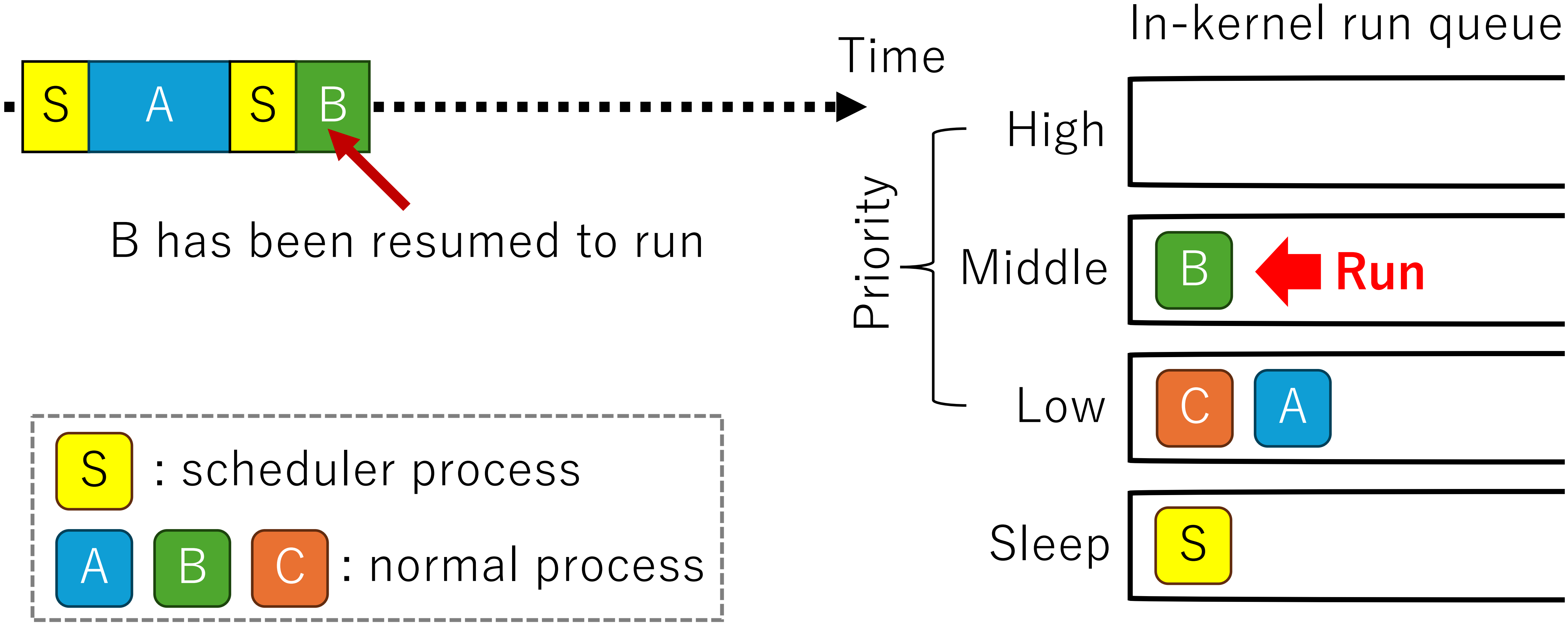


Execution is switched from S to B

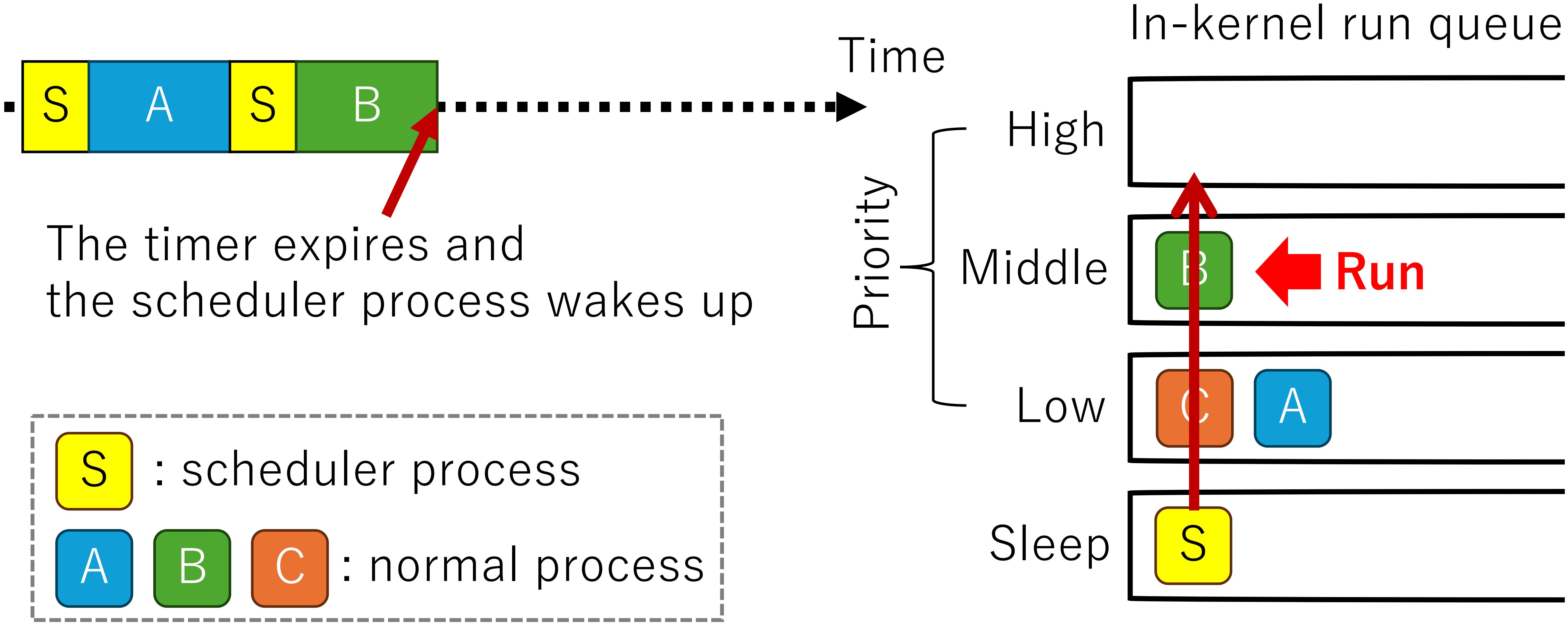




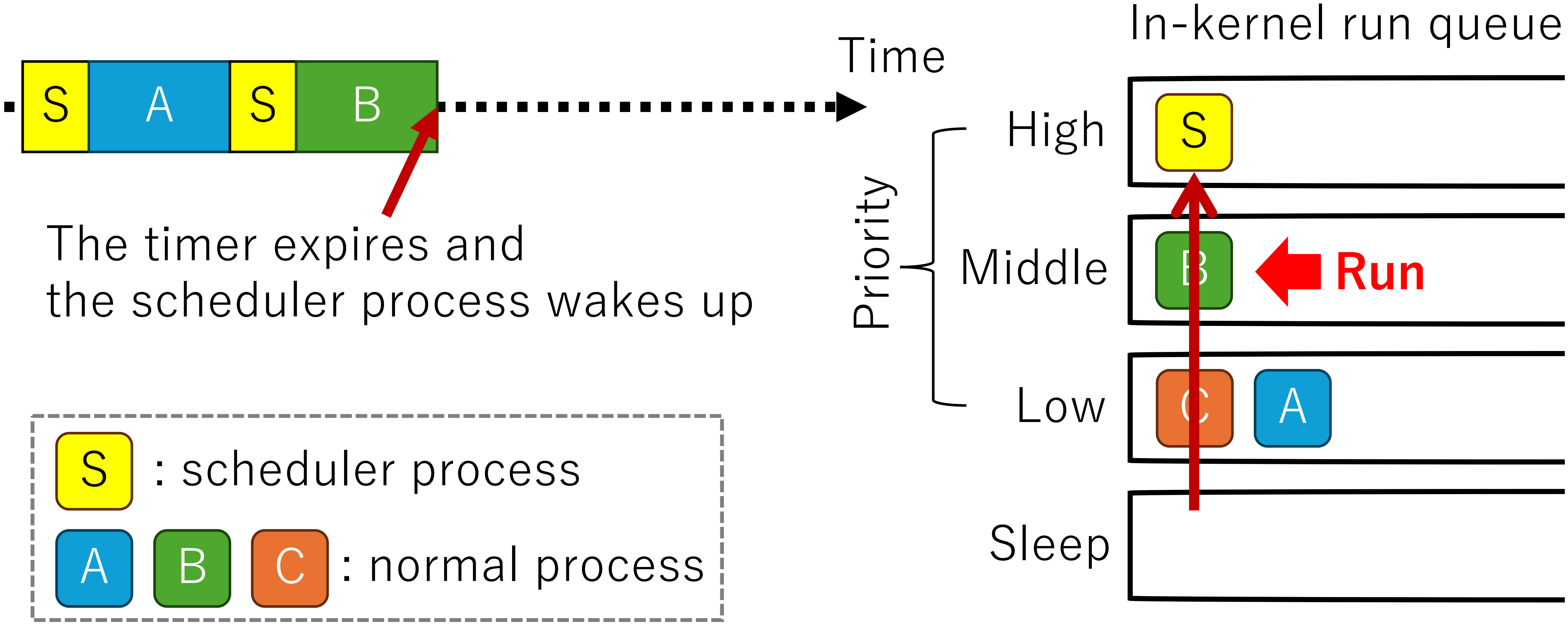
Priority Elevation (Shared Pattern)



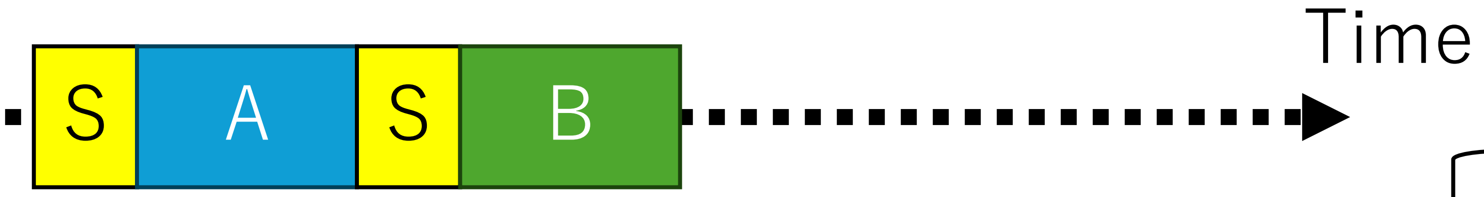
Priority Elevation（Shared Pattern)

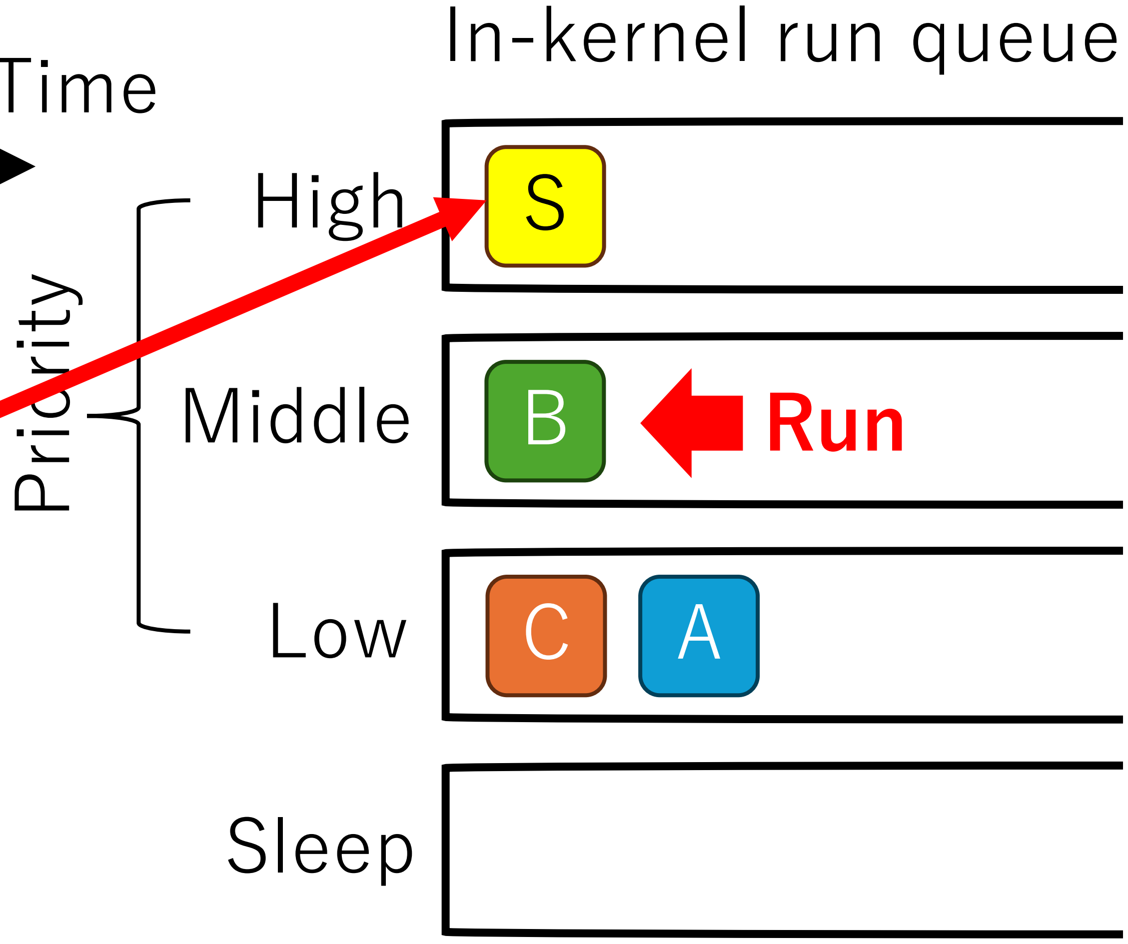


Priority Elevation (Shared Pattern)

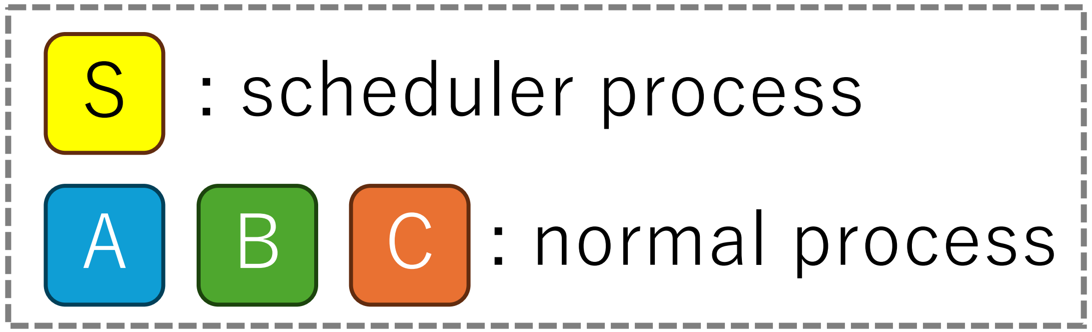


Priority Elevation（Shared Pattern）

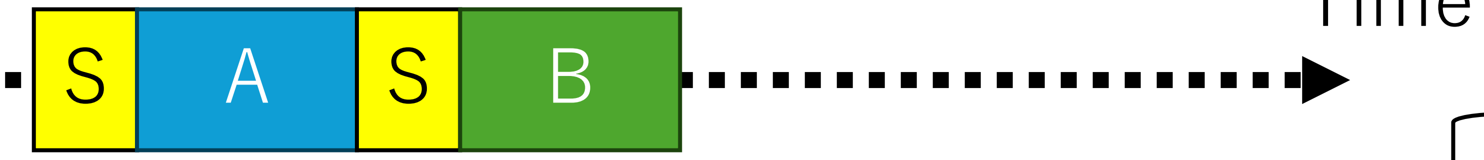




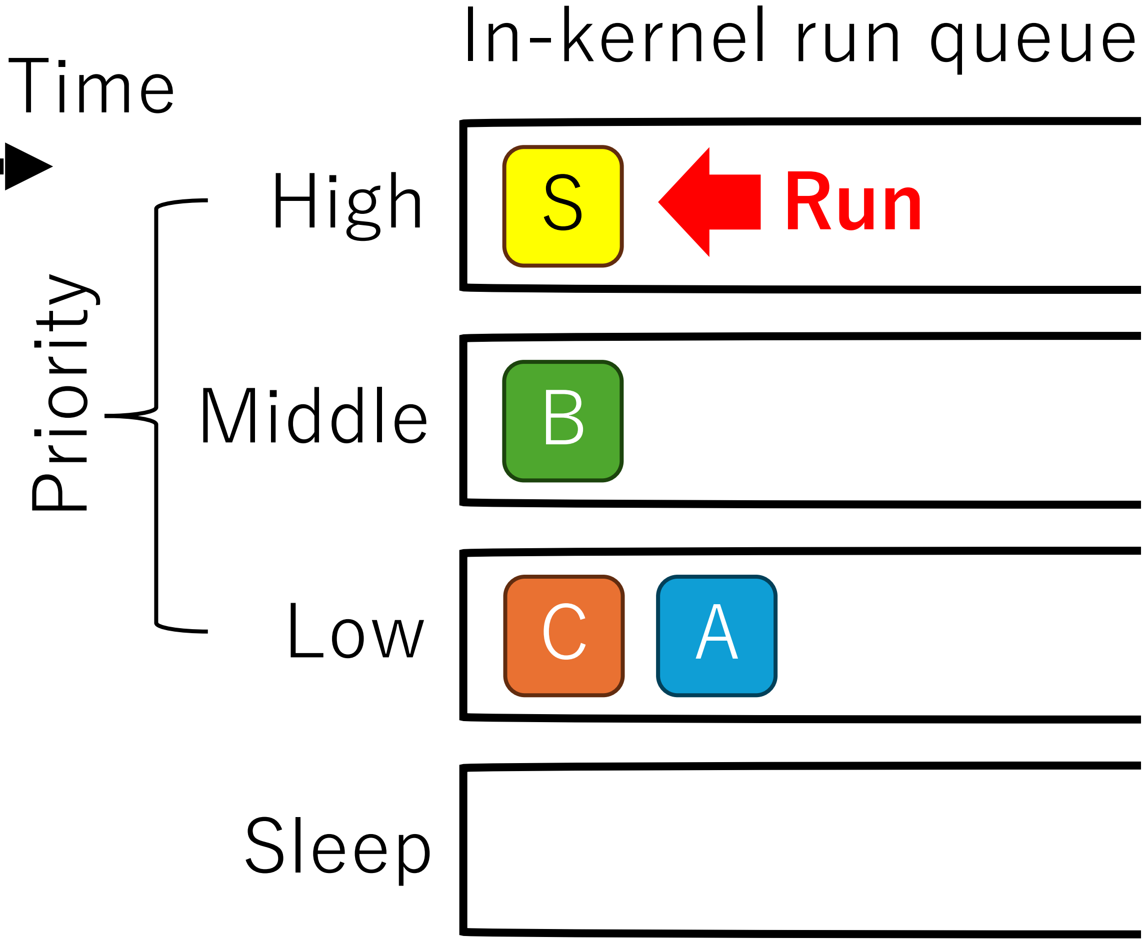
The kernel-space process scheduler picks up the highest priority process

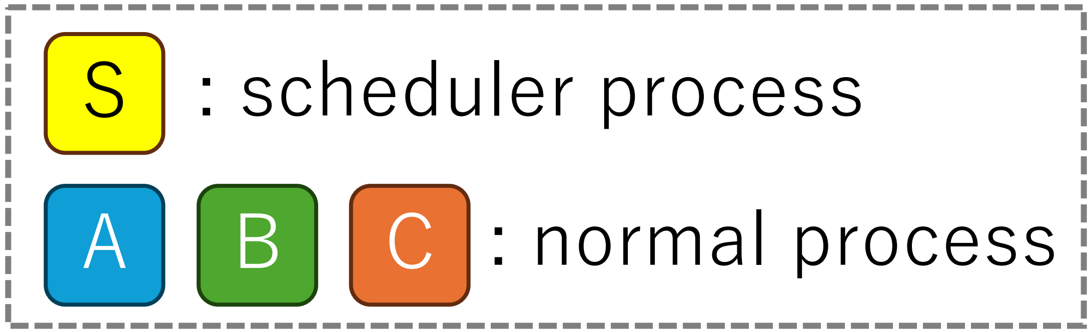


Priority Elevation (Shared Pattern)

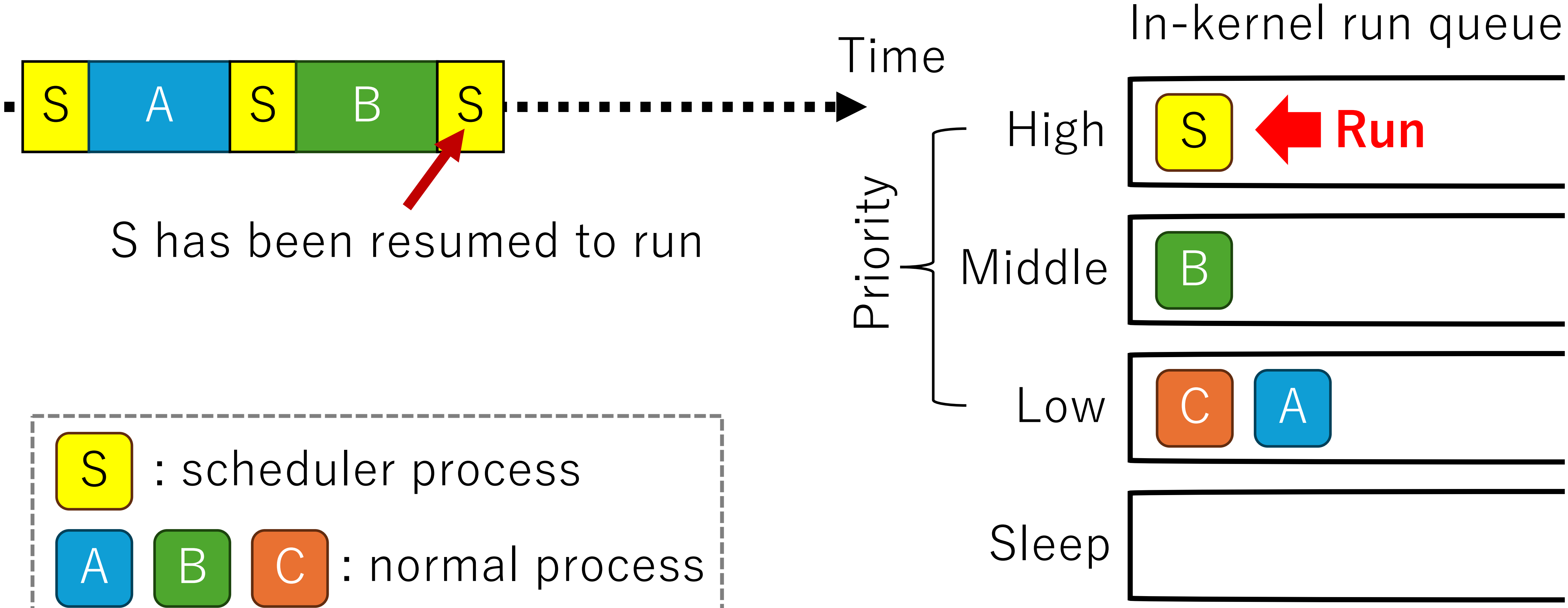


Execution is switched from B to S





Priority Elevation（Shared Pattern）

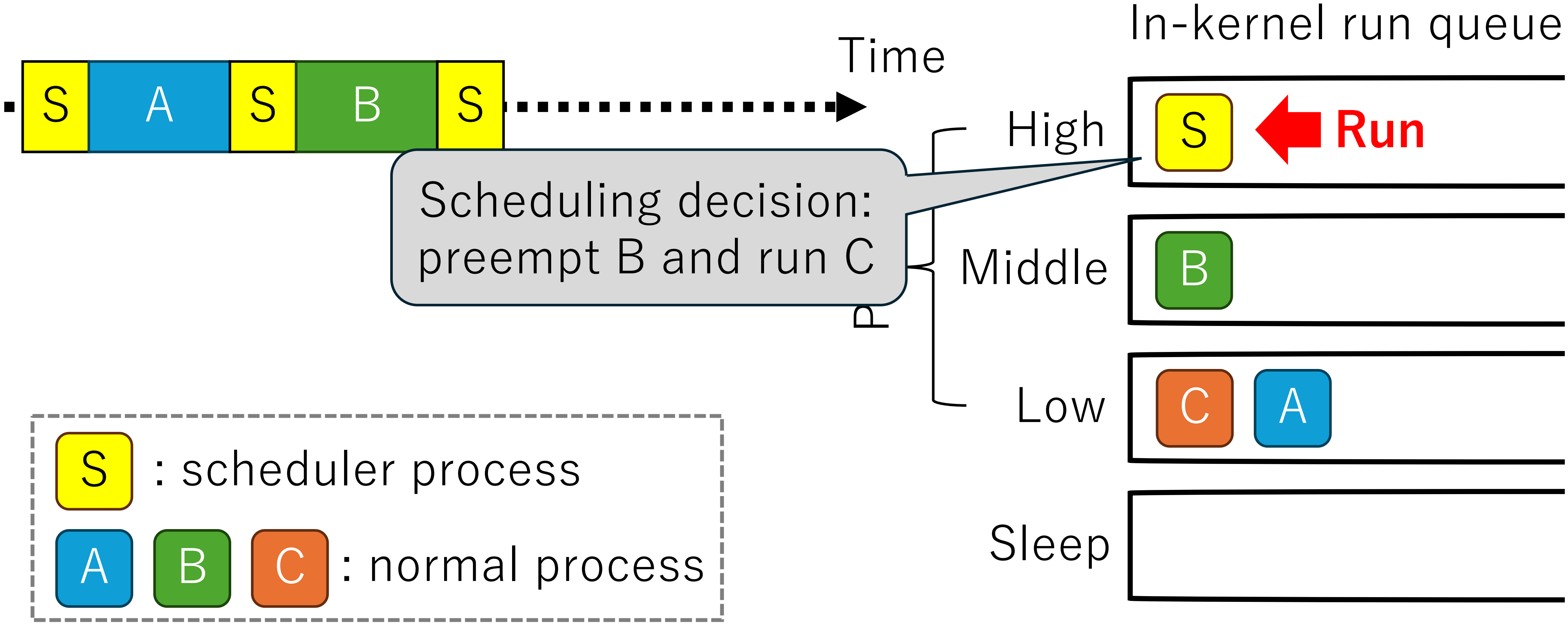


S : scheduler process

A B C : normal process

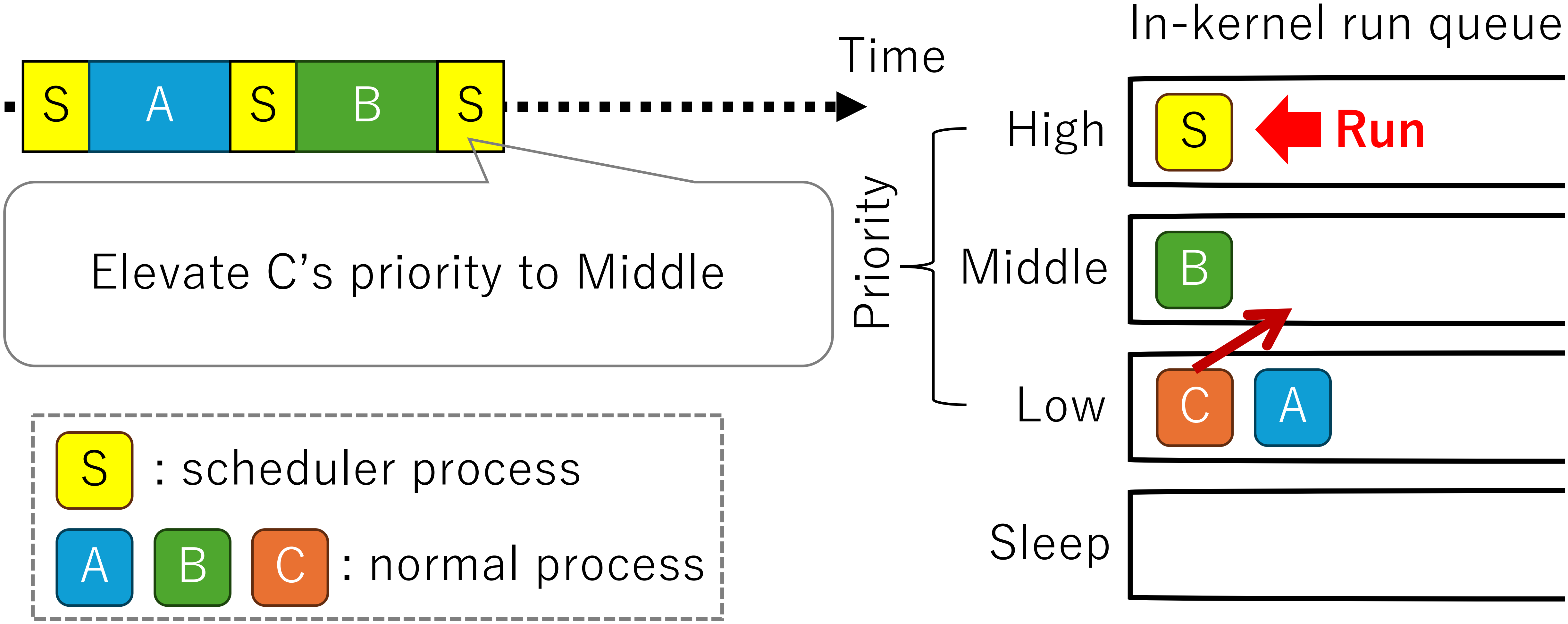
Priority Elevation（Shared Pattern)

We can do the same for C ...



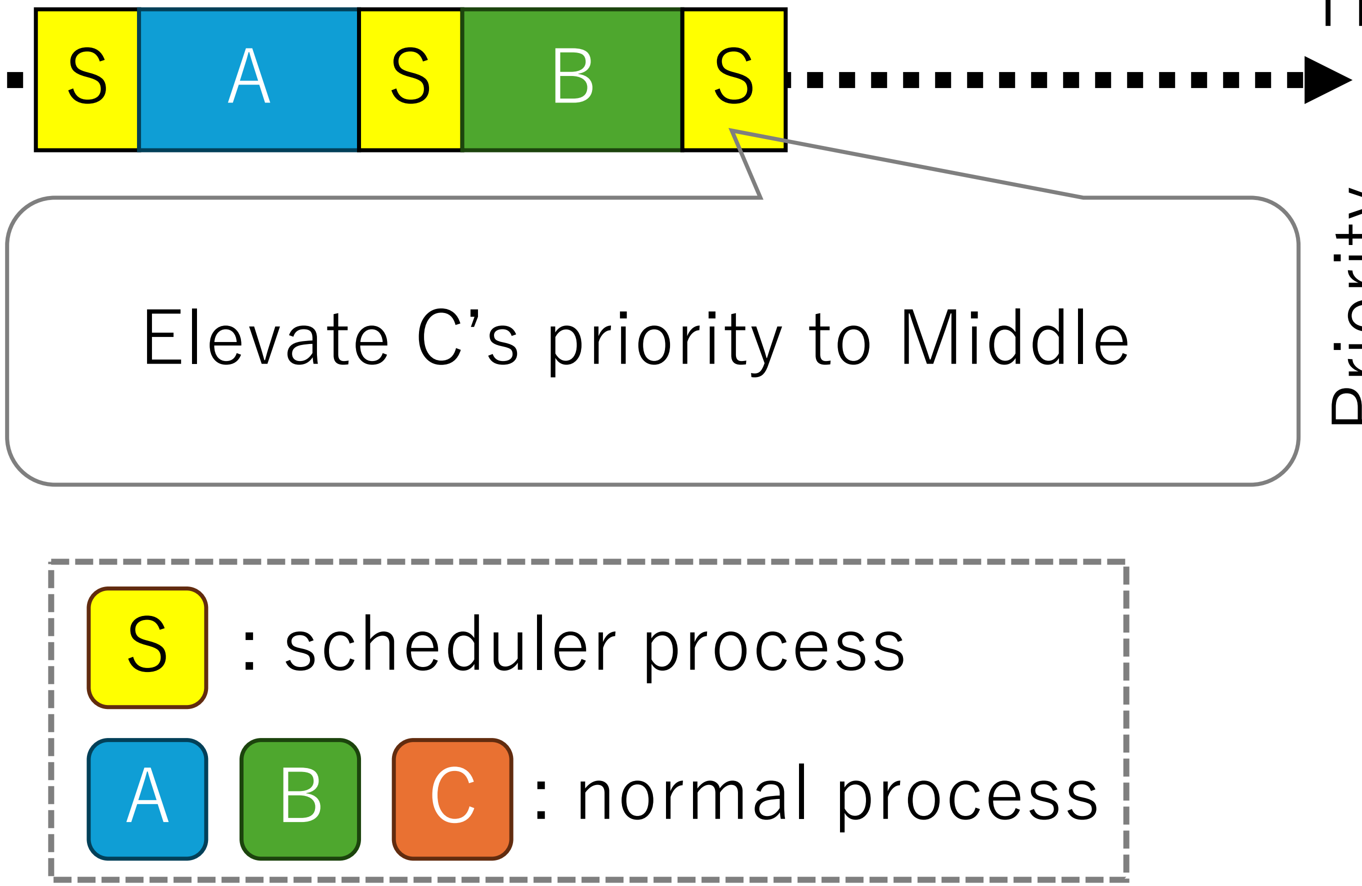
Priority Elevation (Shared Pattern)

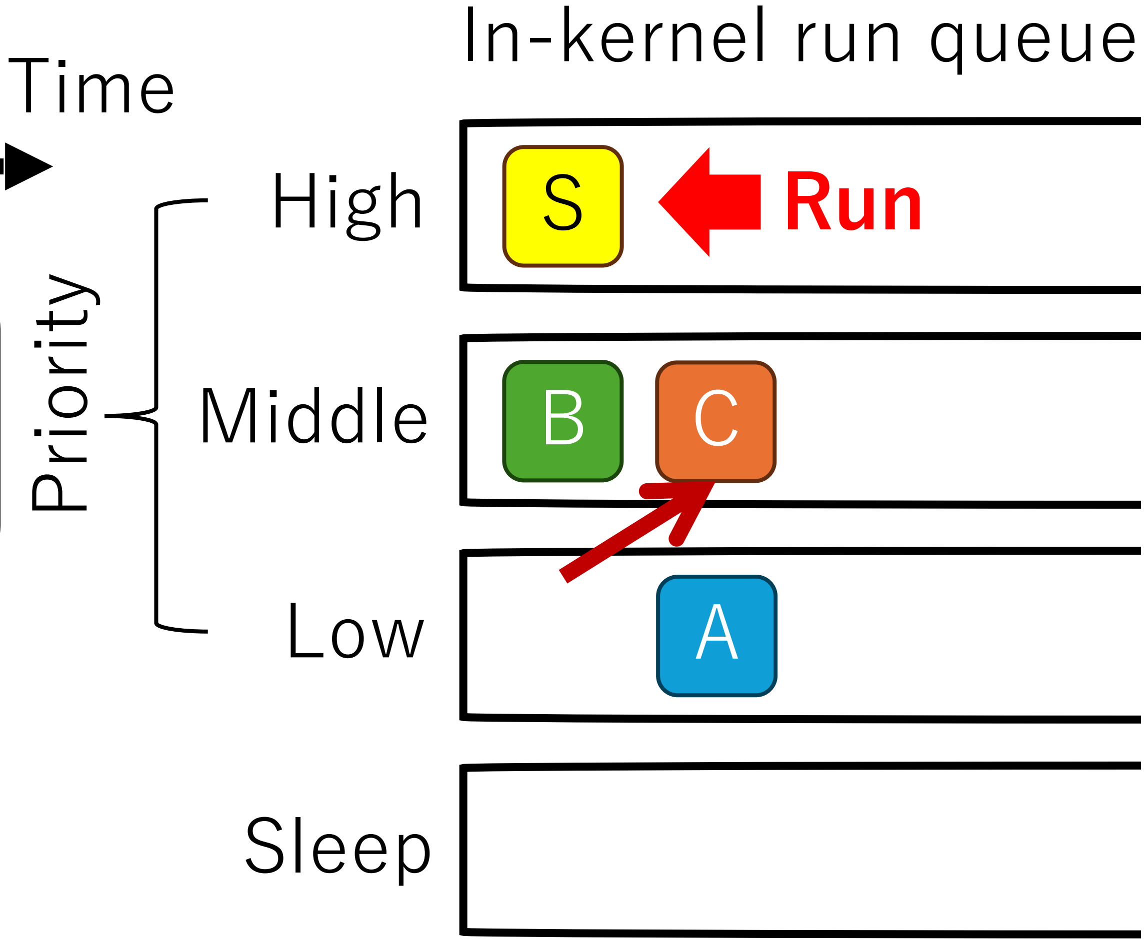
We can do the same for C ...



Priority Elevation (Shared Pattern)

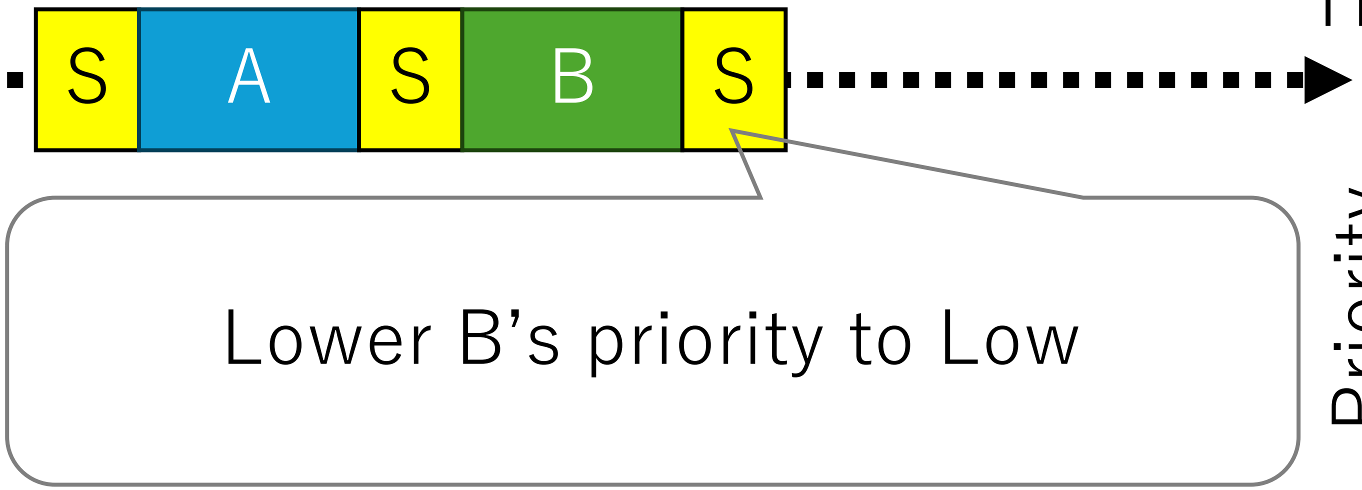
We can do the same for C …

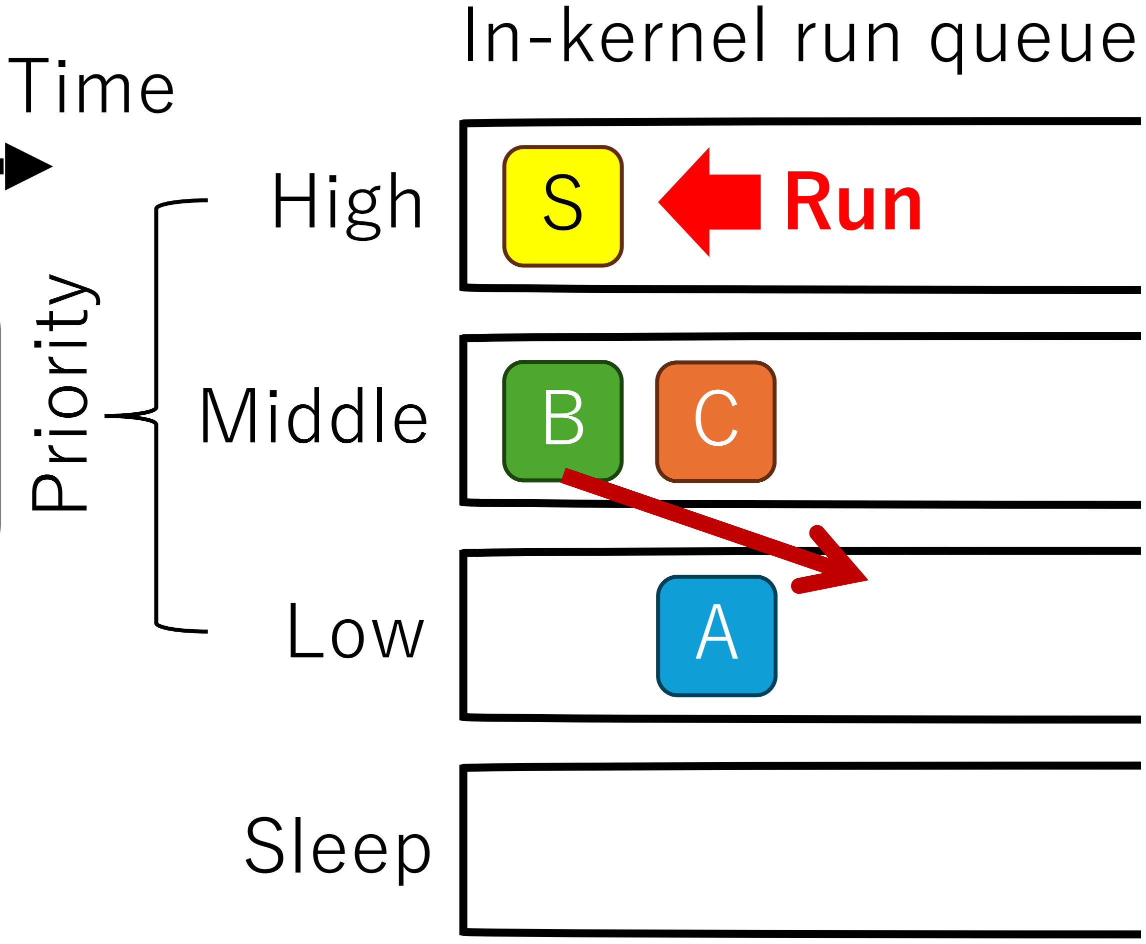


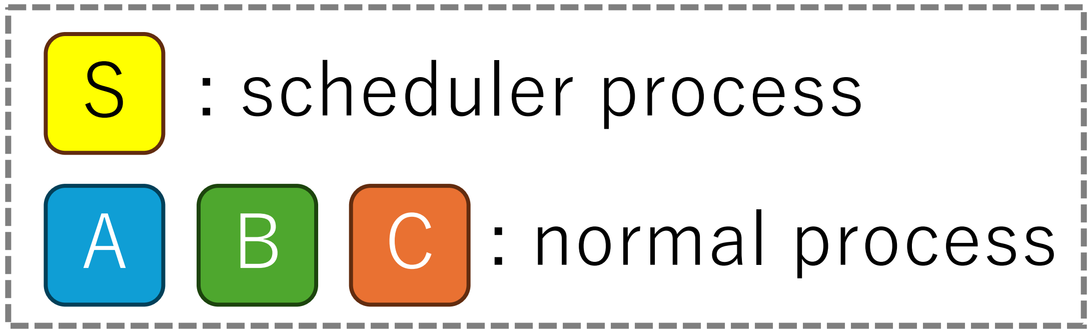


Priority Elevation（Shared Pattern）

We can do the same for C …

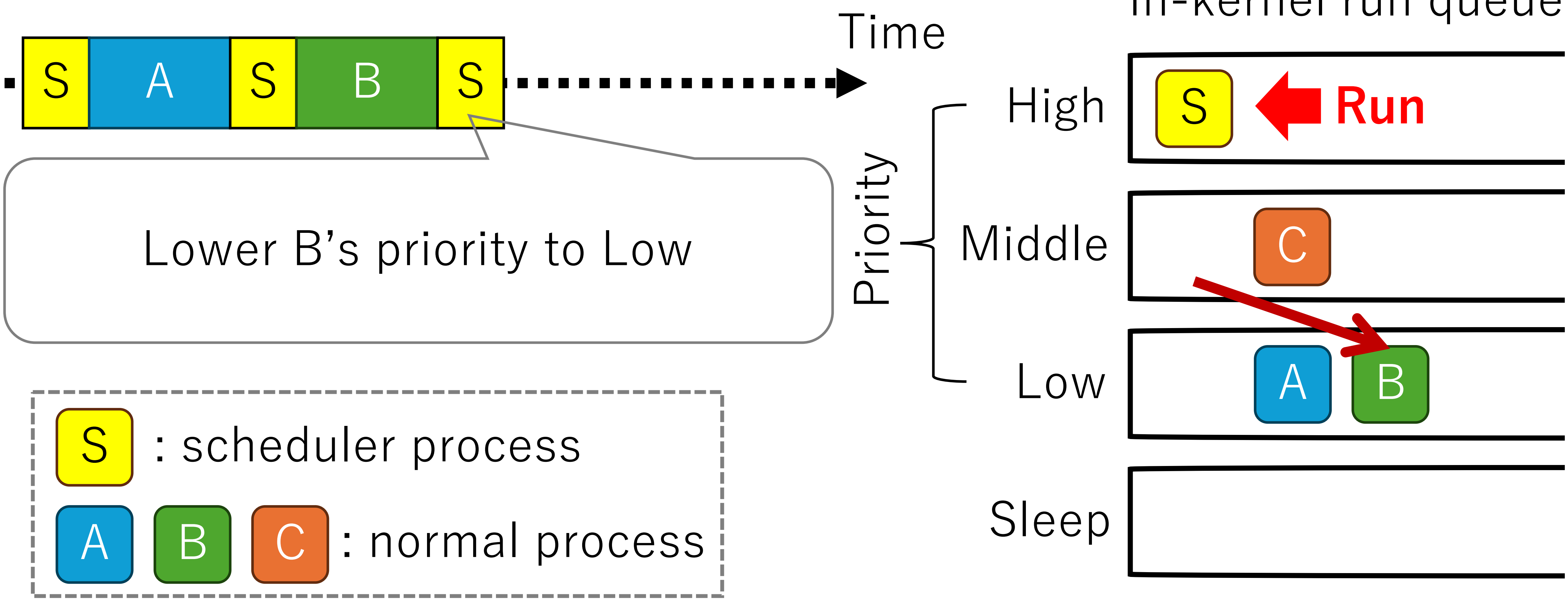






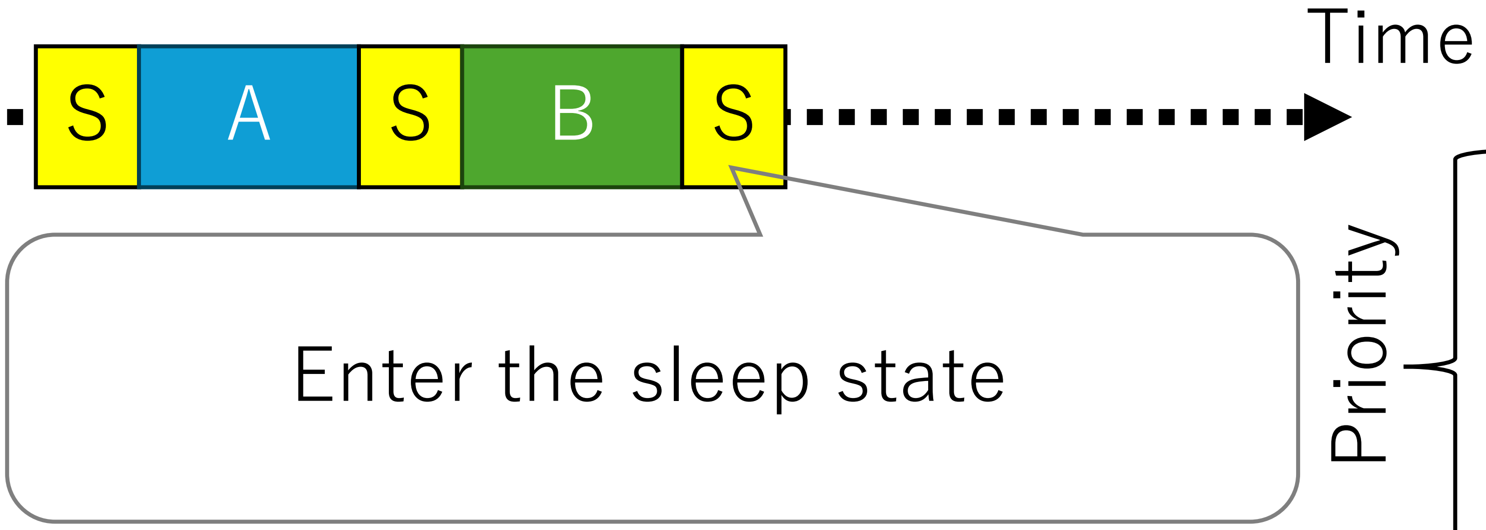
Priority Elevation (Shared Pattern)

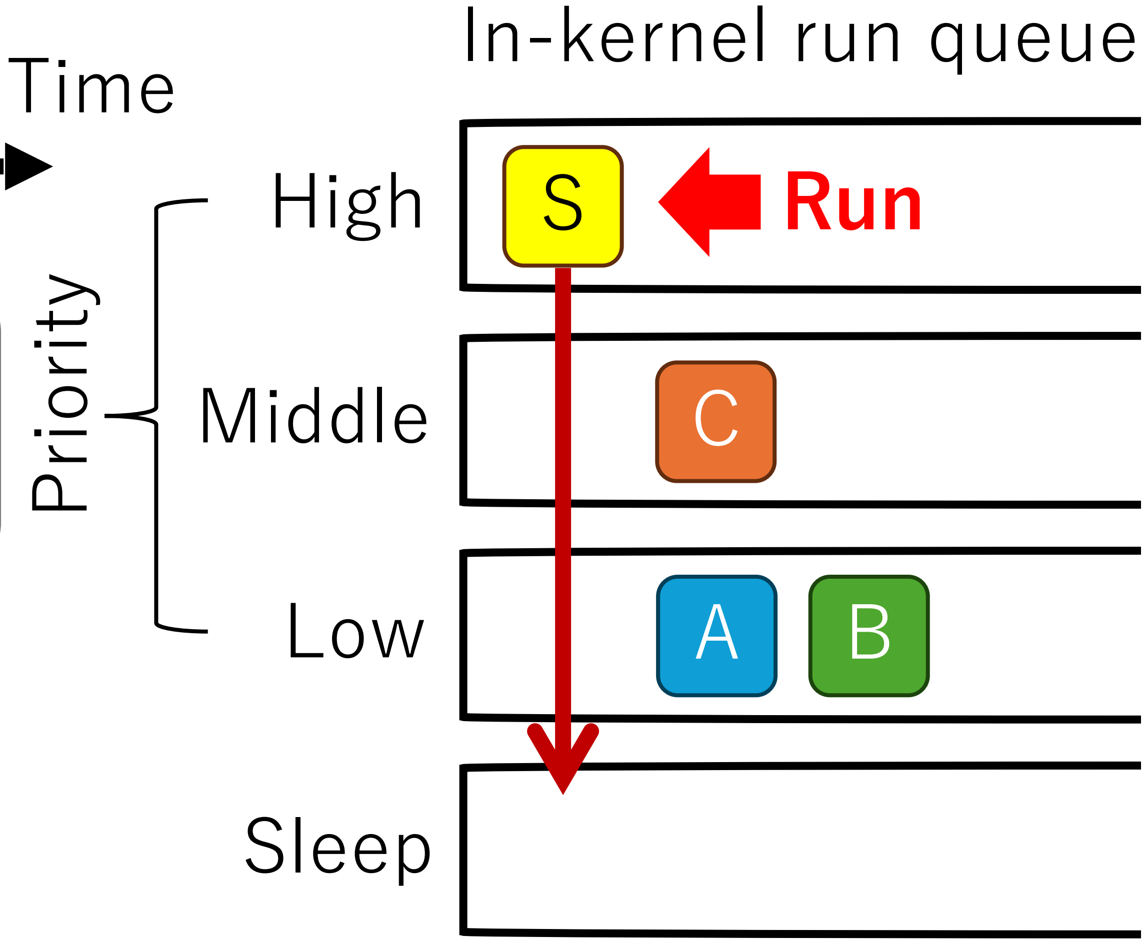
We can do the same for C …

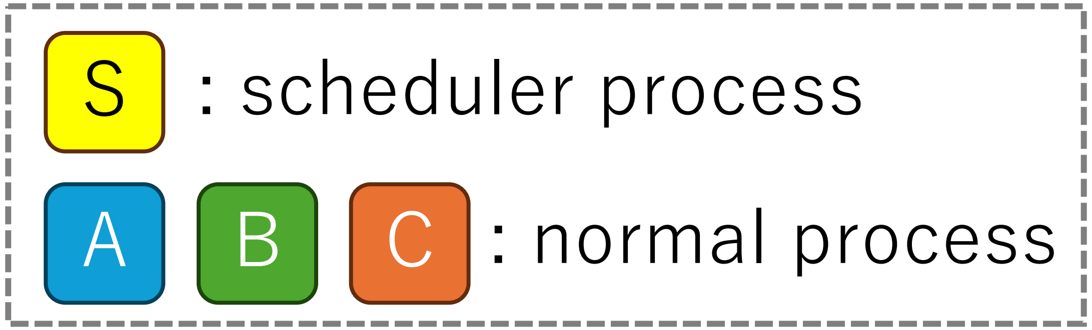


Priority Elevation (Shared Pattern)

We can do the same for C ...

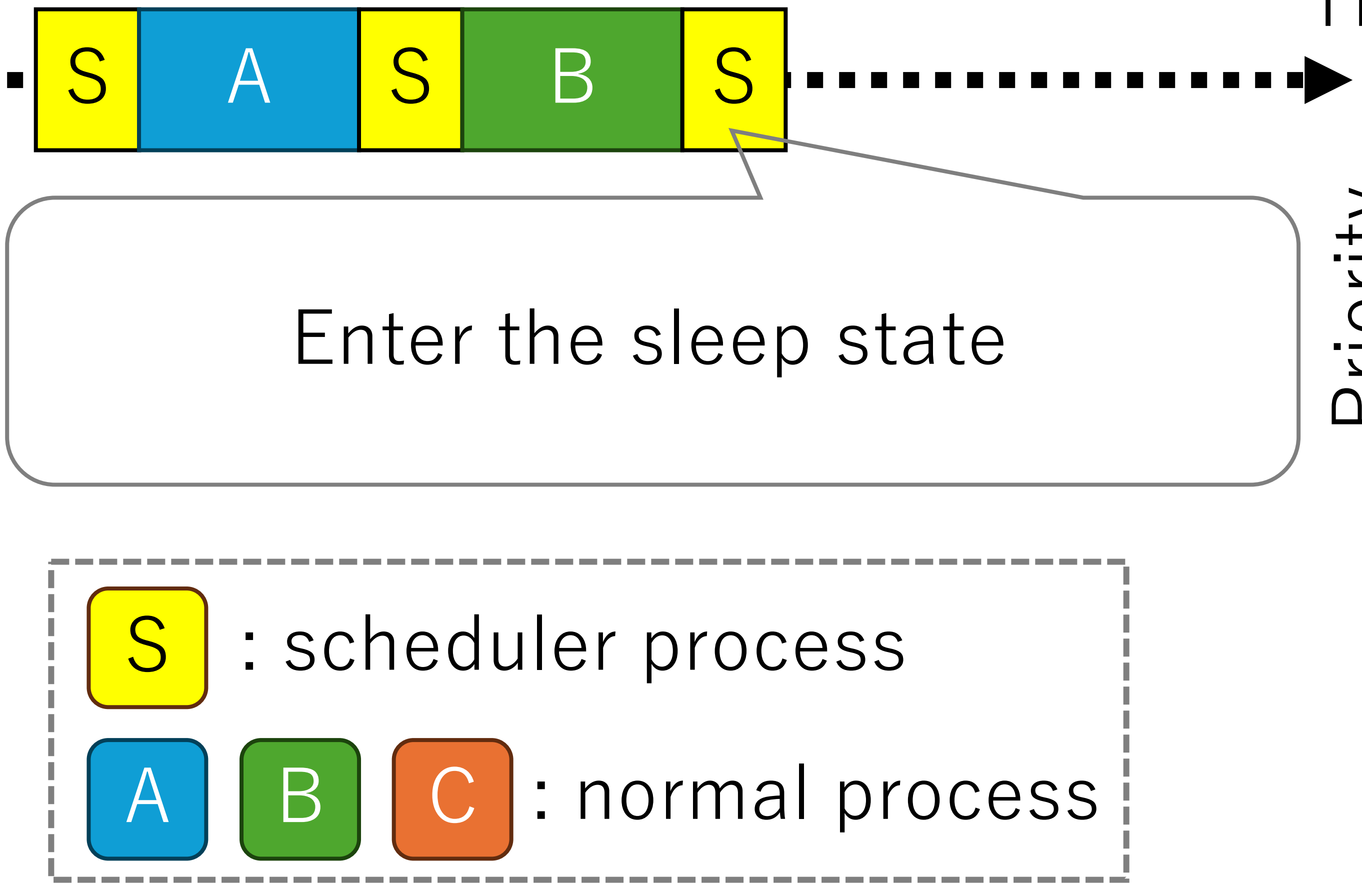


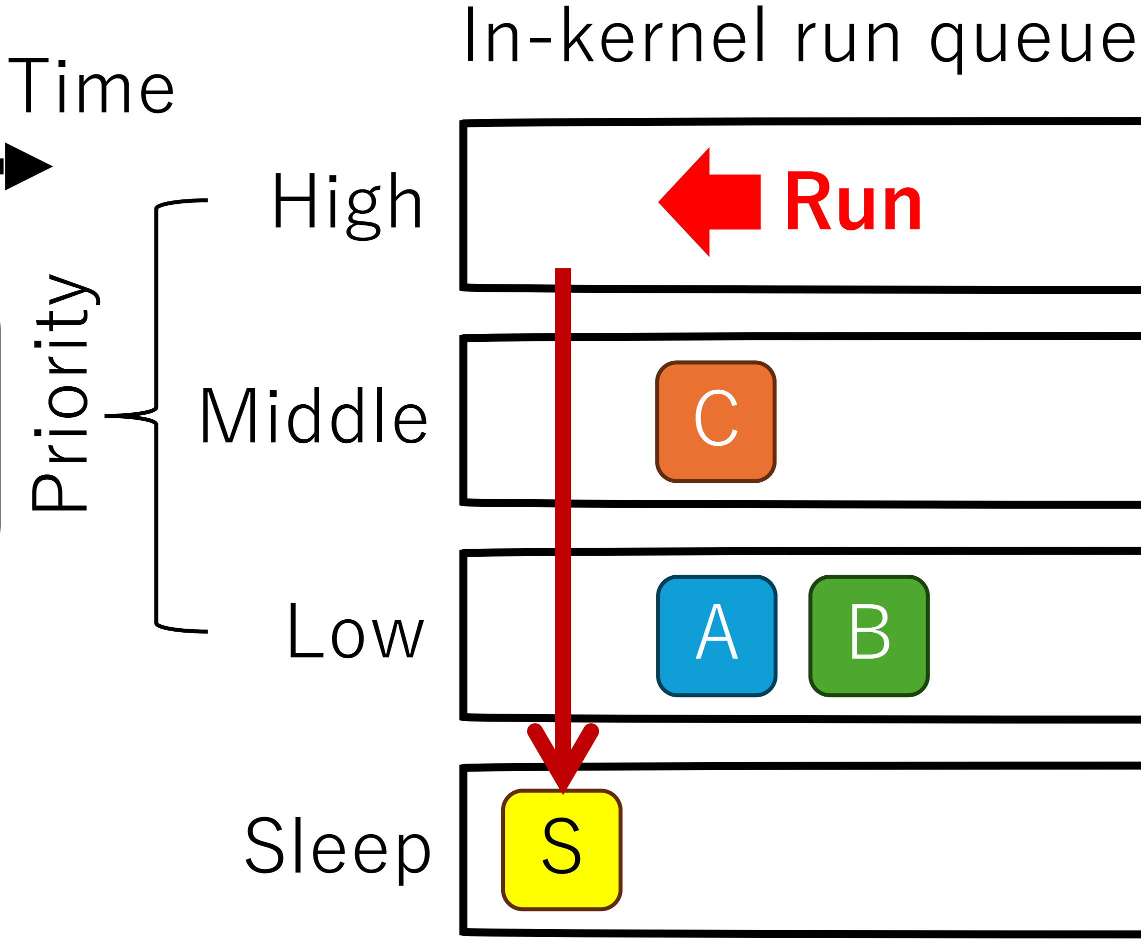




Priority Elevation（Shared Pattern）

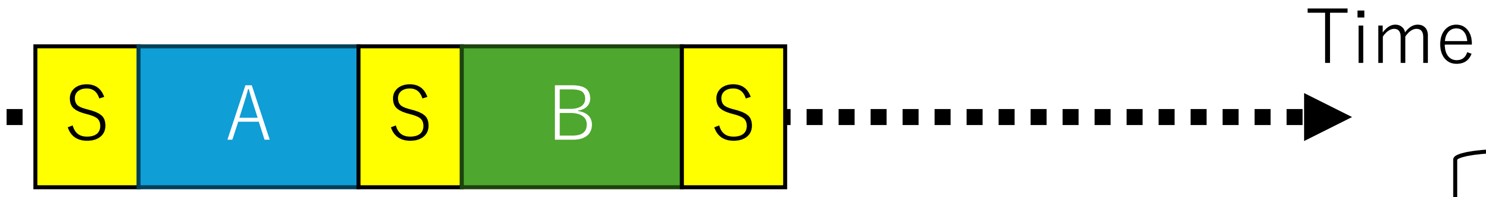
We can do the same for C ...



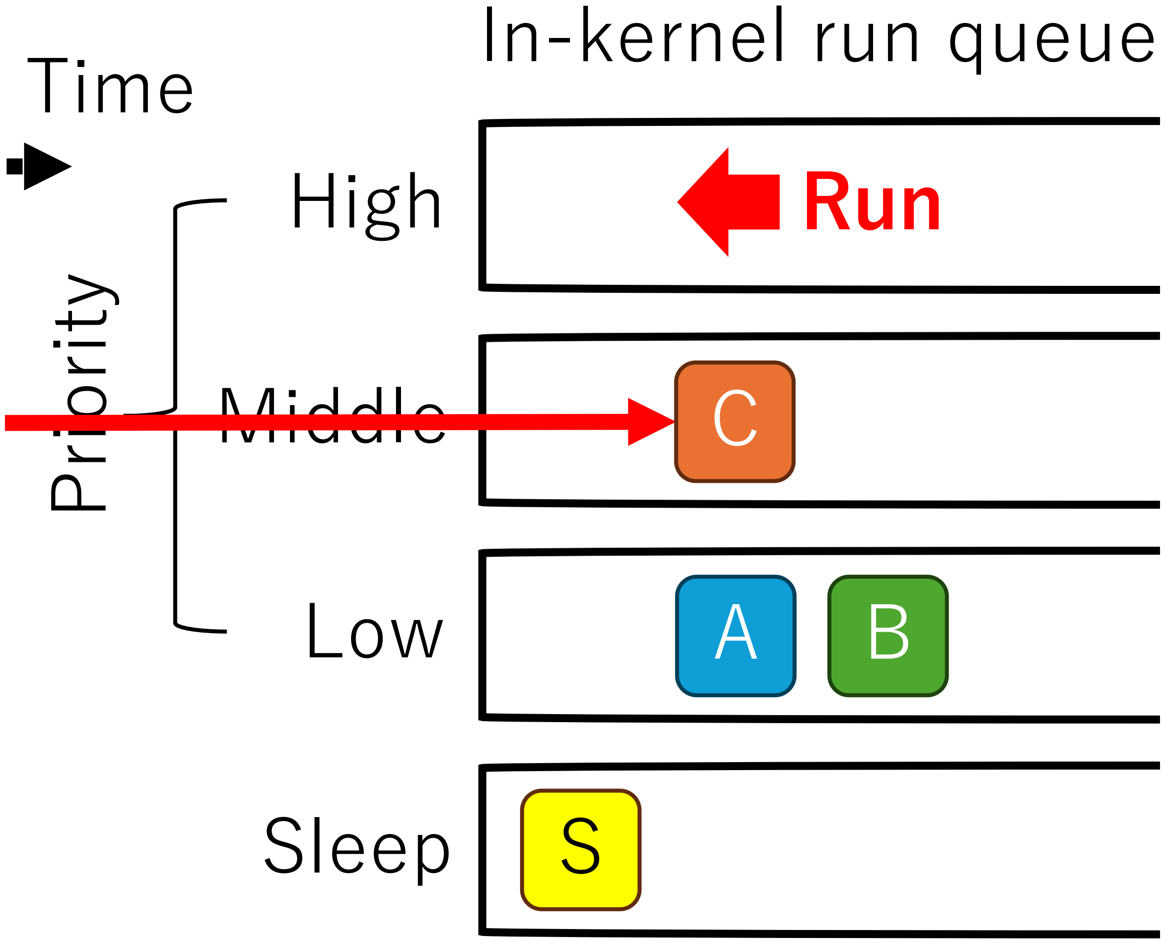


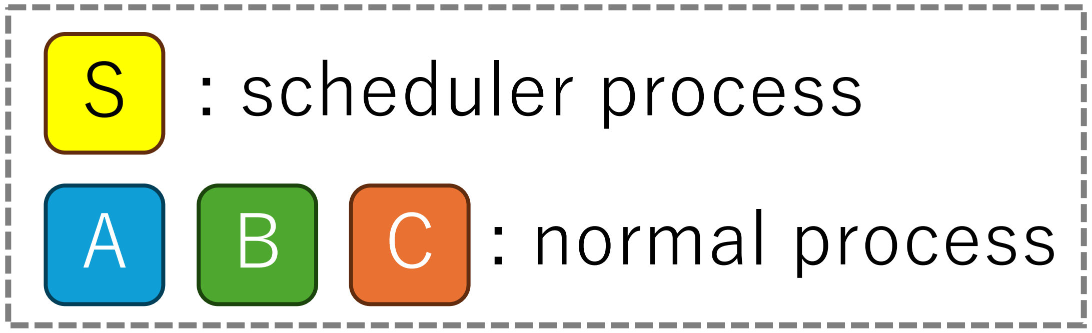
Priority Elevation (Shared Pattern)

We can do the same for C ...



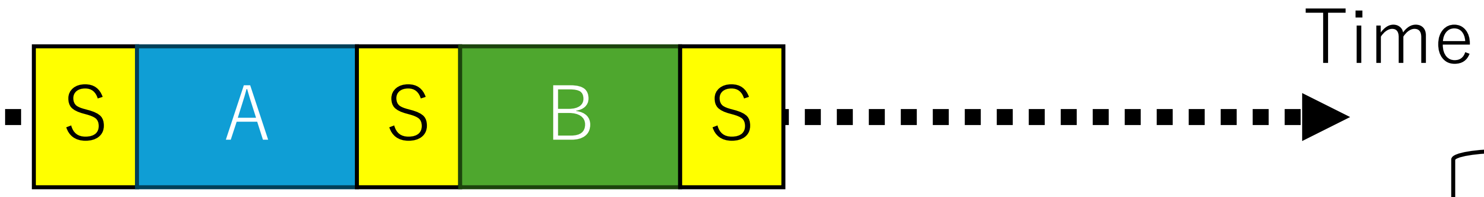
The kernel-space process scheduler picks up the highest priority process



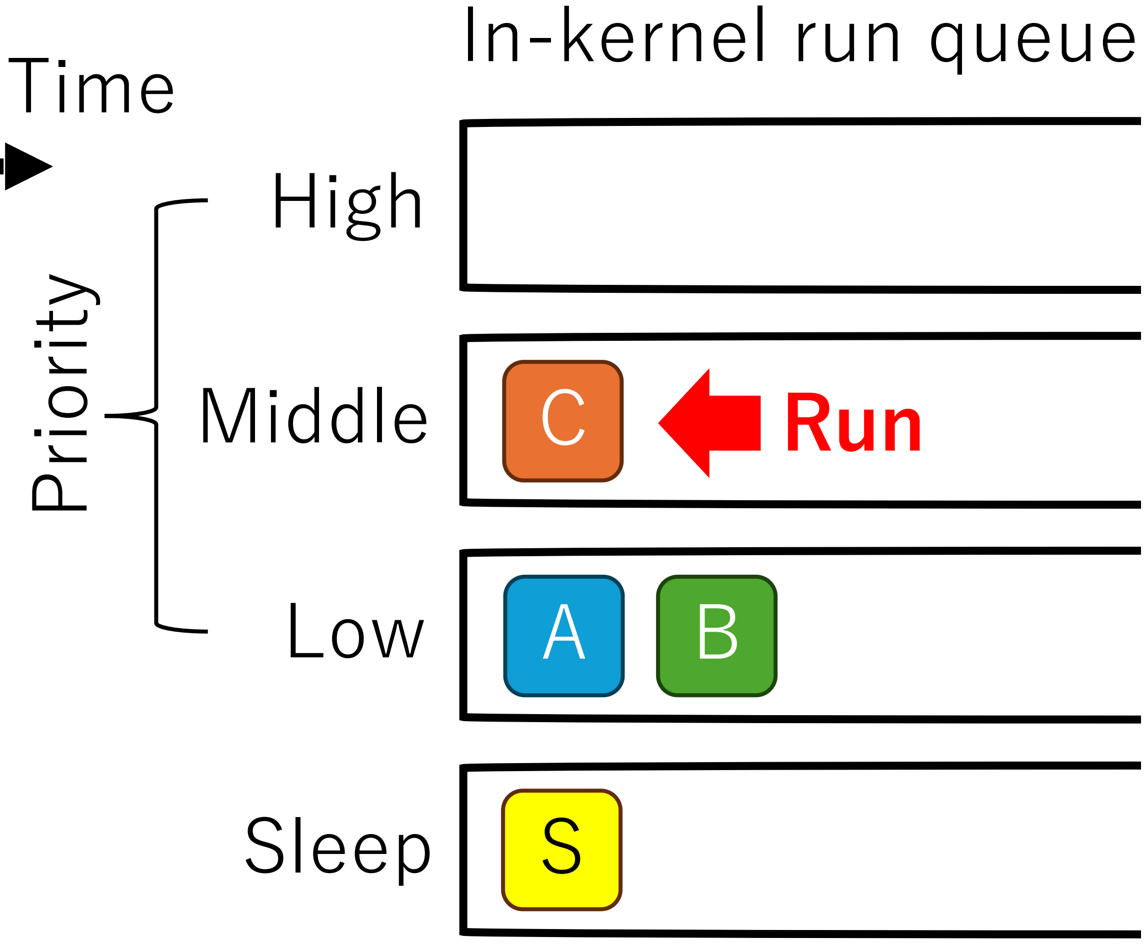


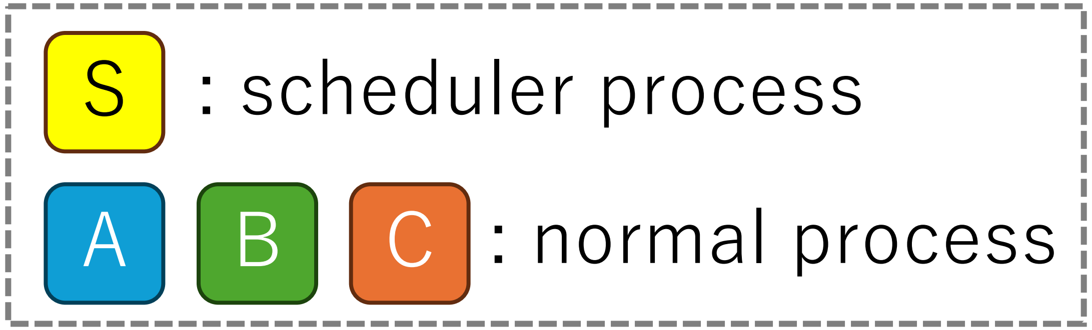
Priority Elevation（Shared Pattern)

We can do the same for C …



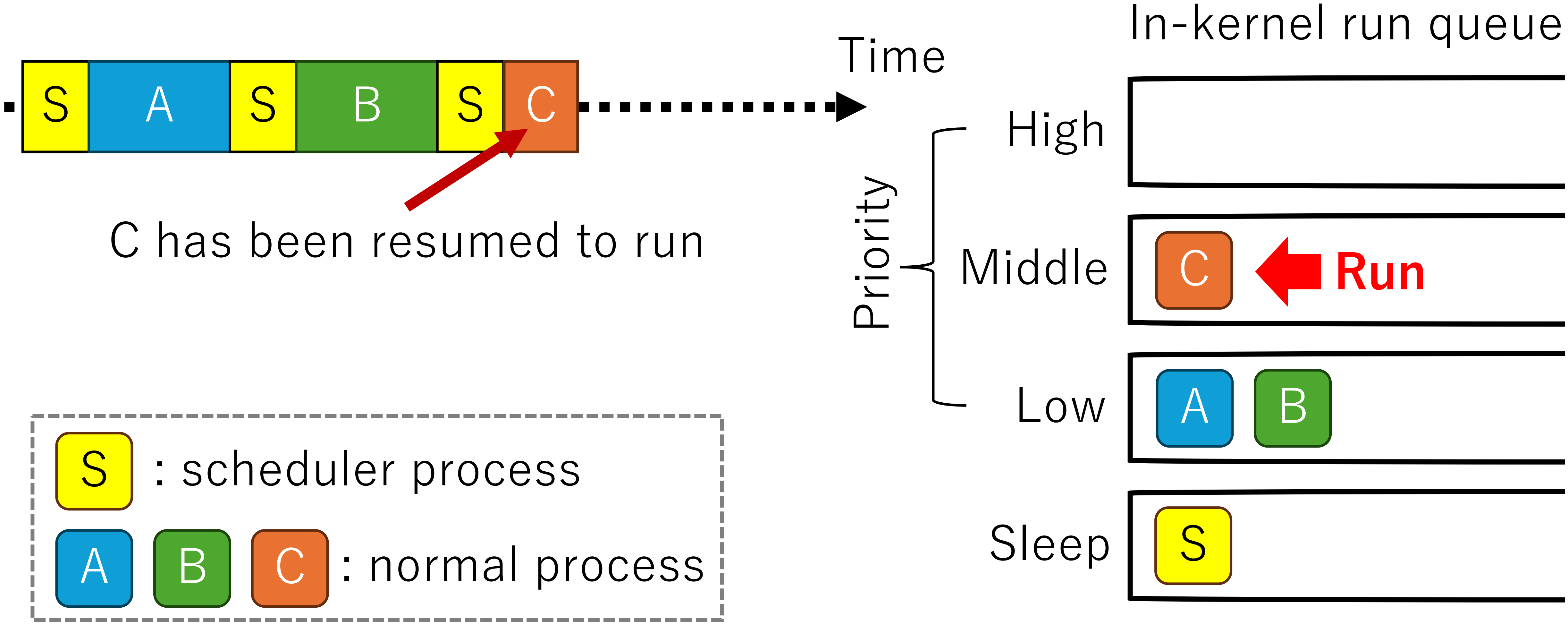
Execution is switched from S to C





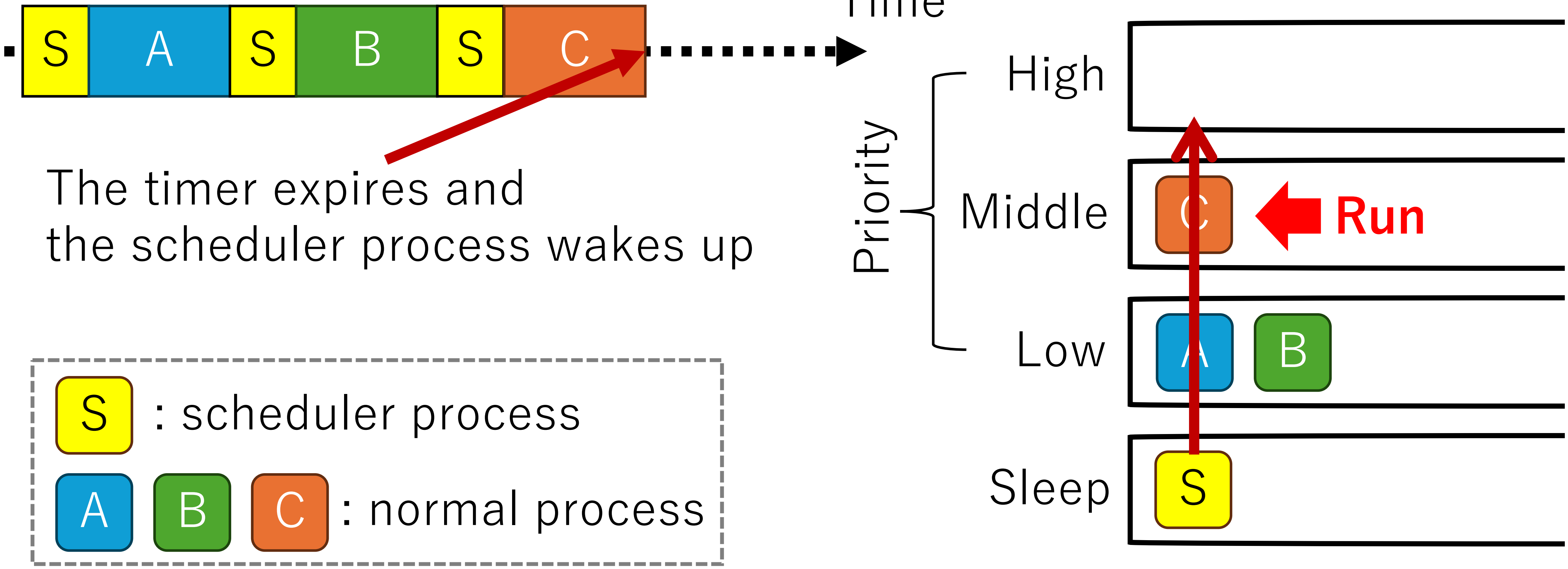
Priority Elevation (Shared Pattern)

We can do the same for C ...



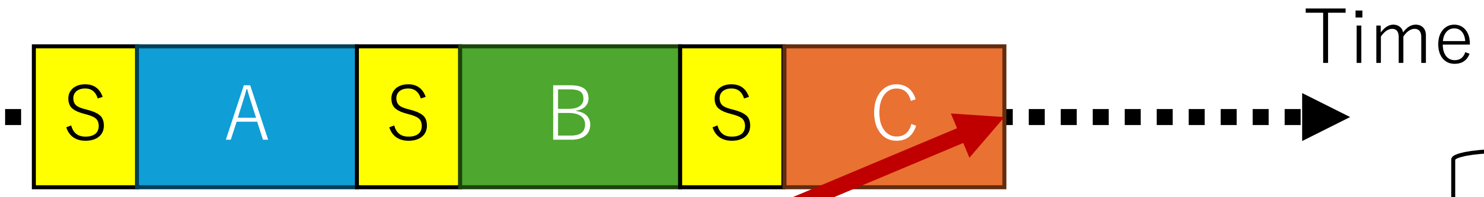
Priority Elevation (Shared Pattern)

We can do the same for C ...

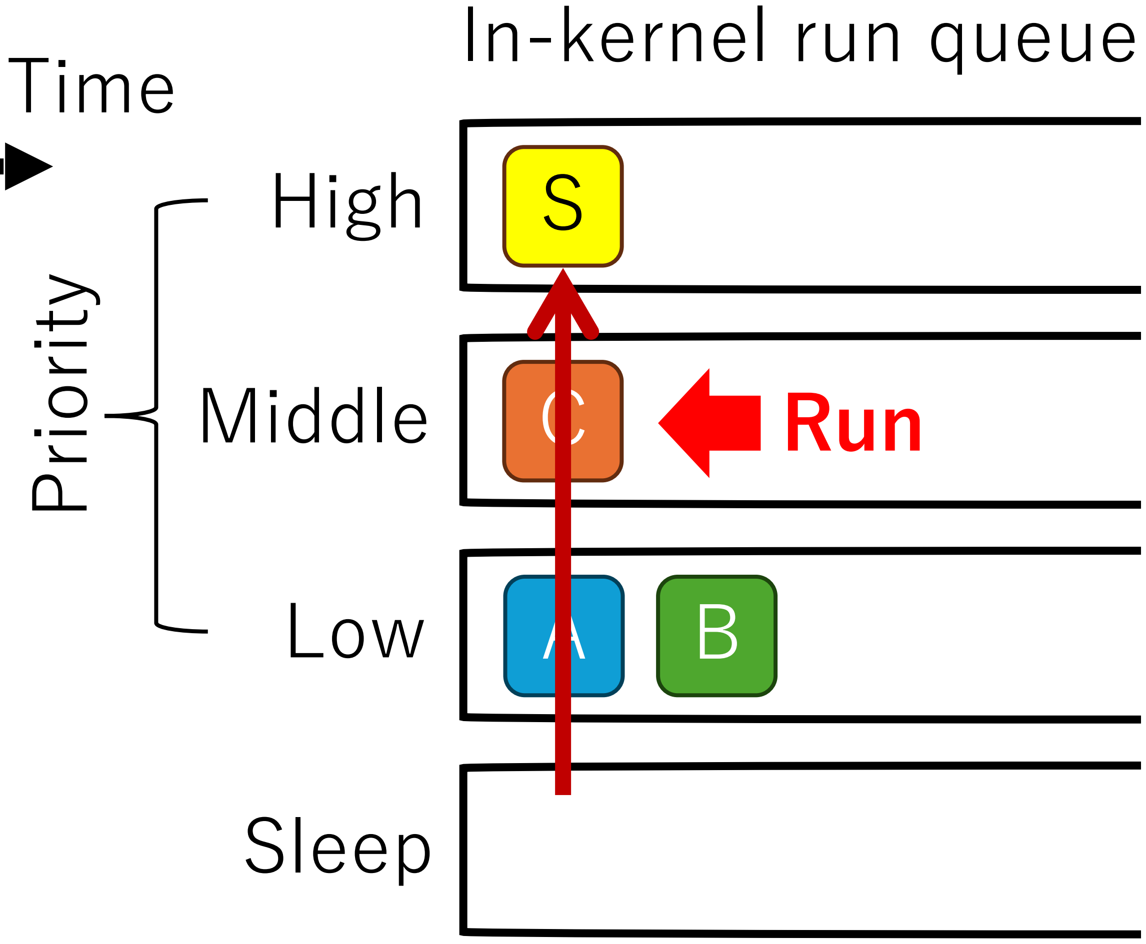


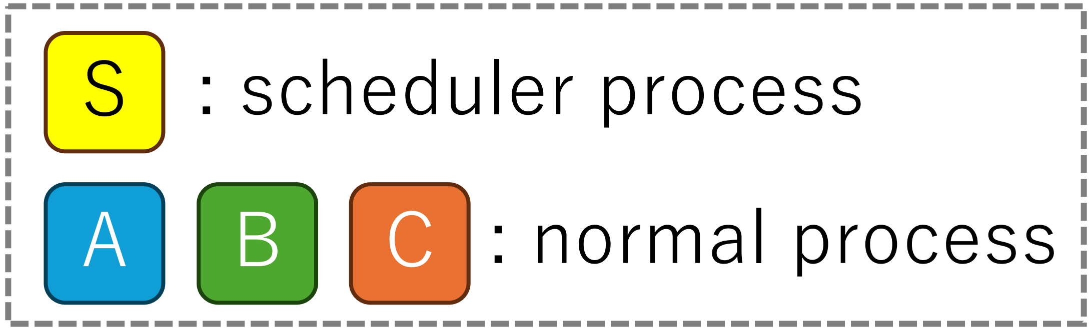
Priority Elevation (Shared Pattern)

We can do the same for C ...



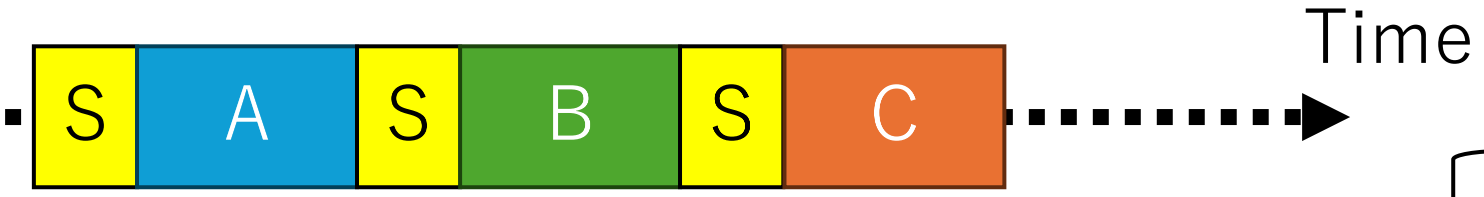
The timer expires and
the scheduler process wakes up



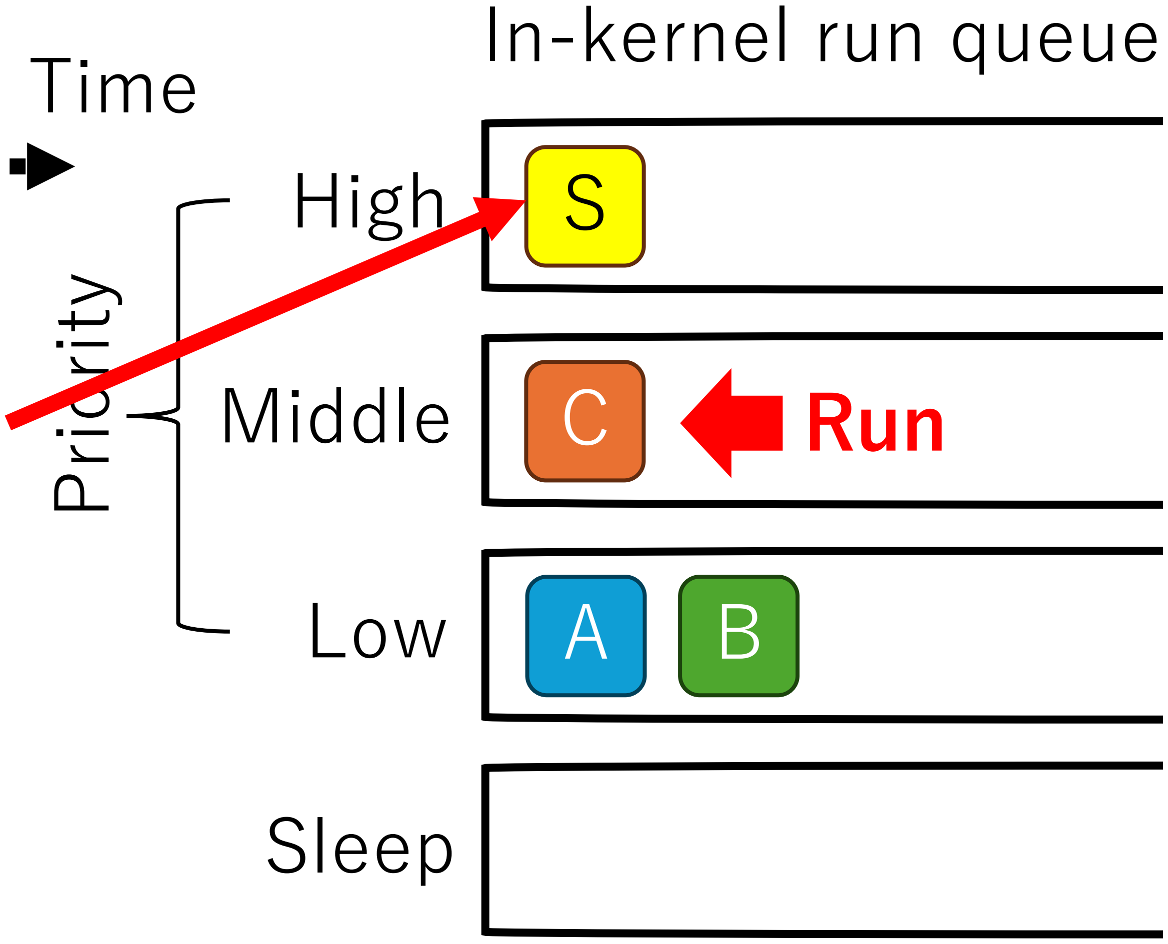


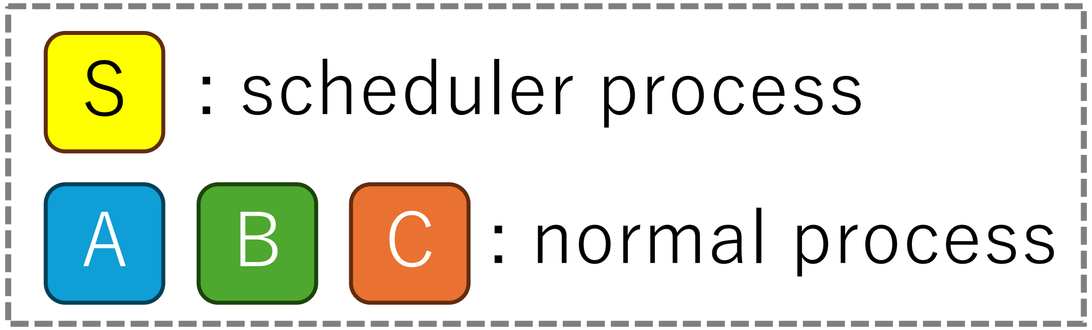
Priority Elevation (Shared Pattern)

We can do the same for C ...



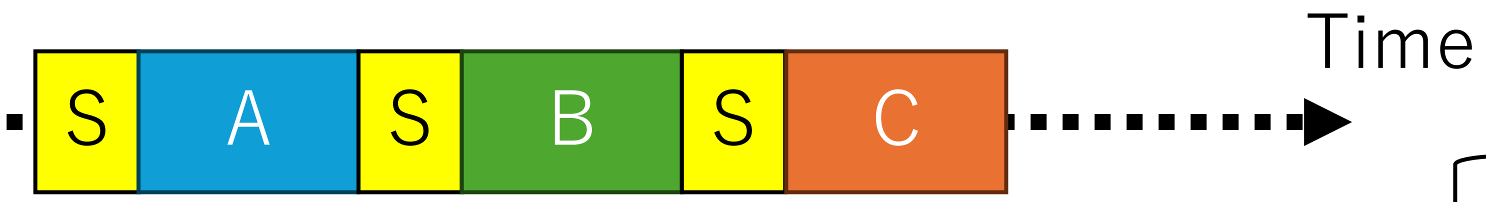
The kernel-space process scheduler picks up the highest priority process



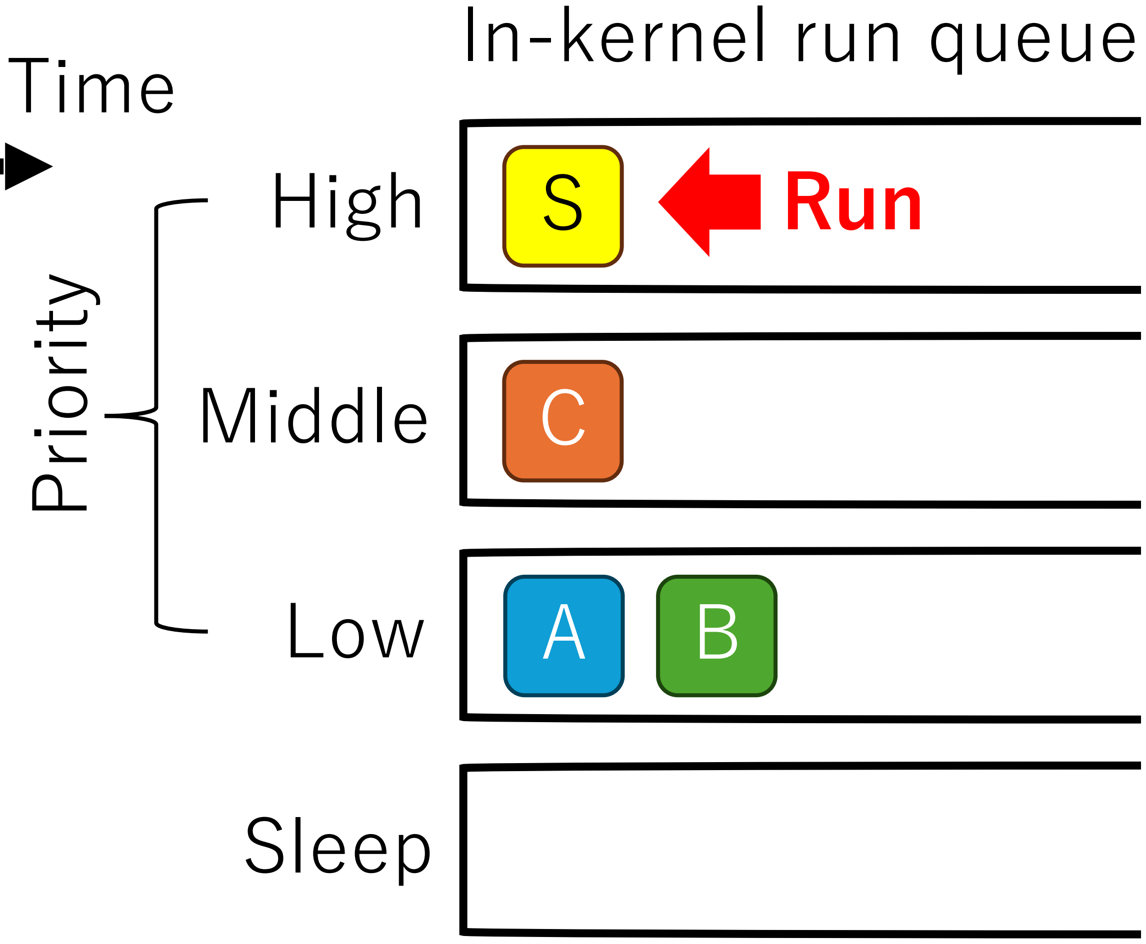


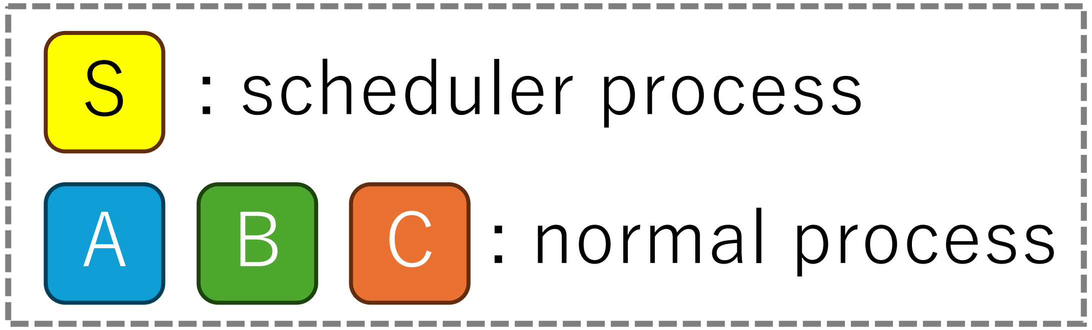
Priority Elevation（Shared Pattern)

We can do the same for C ...



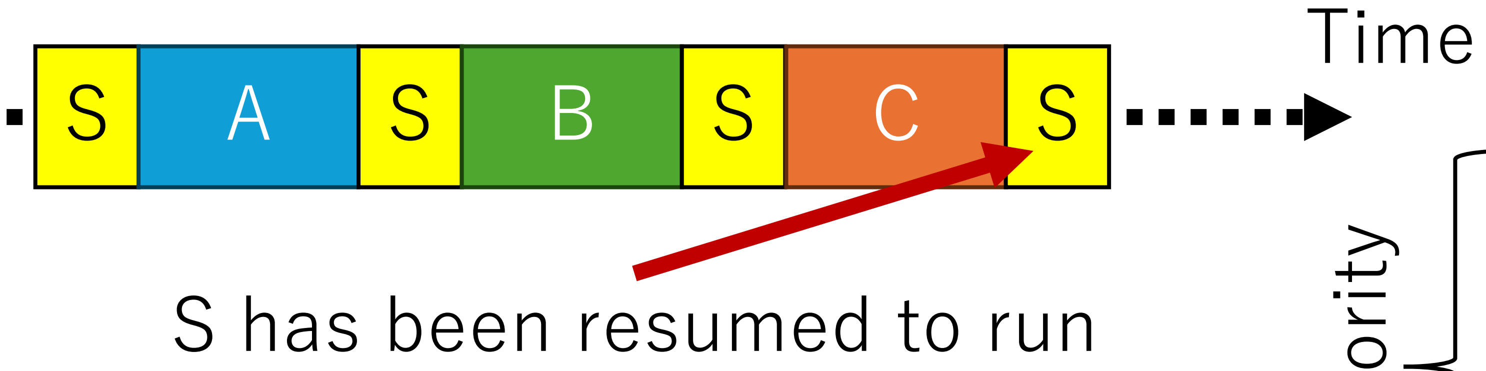
Execution is switched from C to S

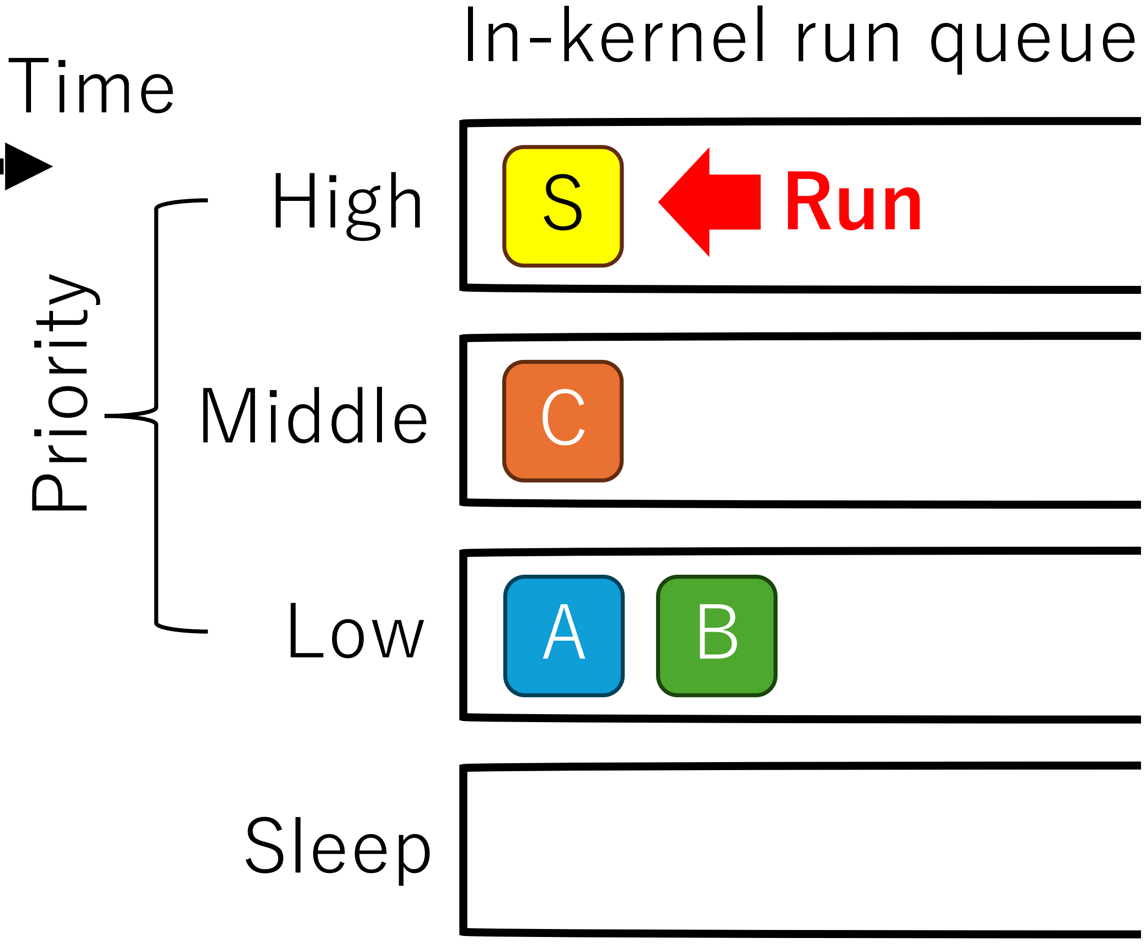


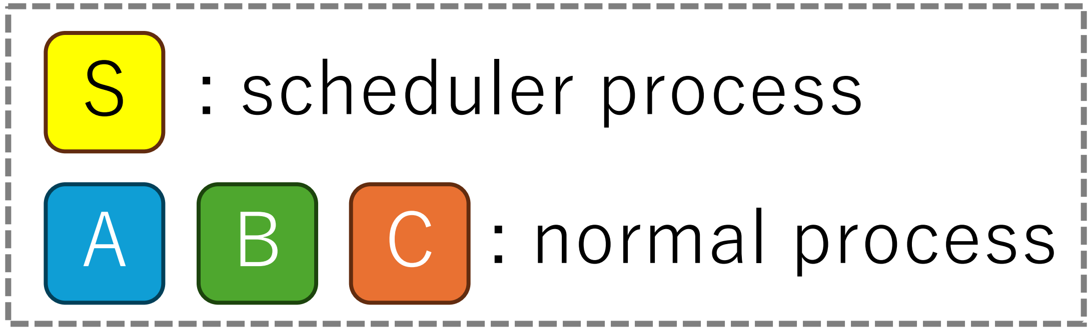


Priority Elevation (Shared Pattern)

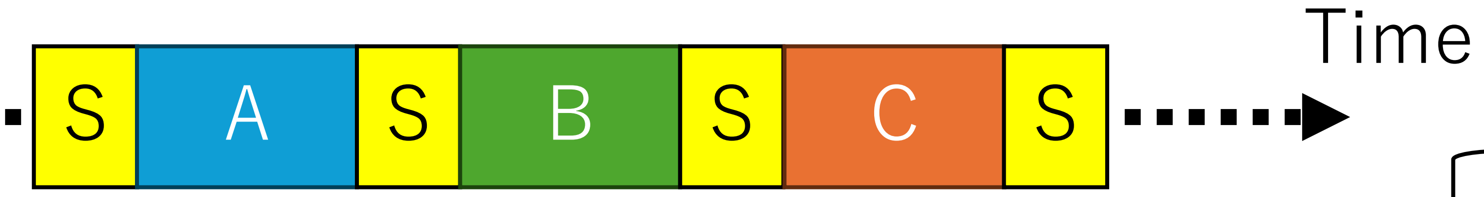
We can do the same for C ...



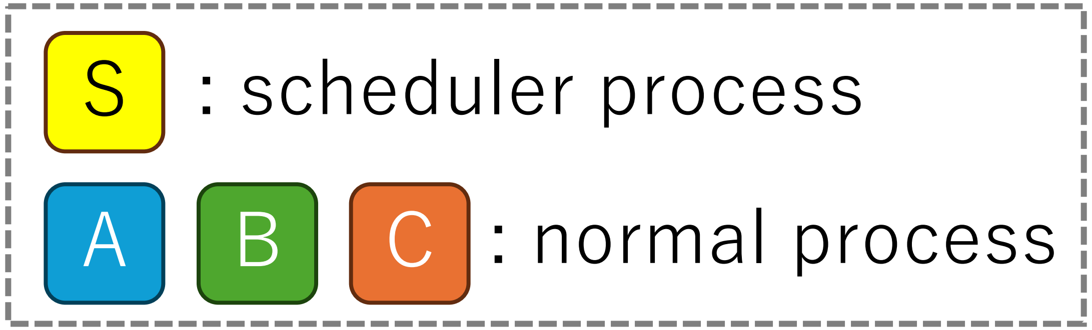


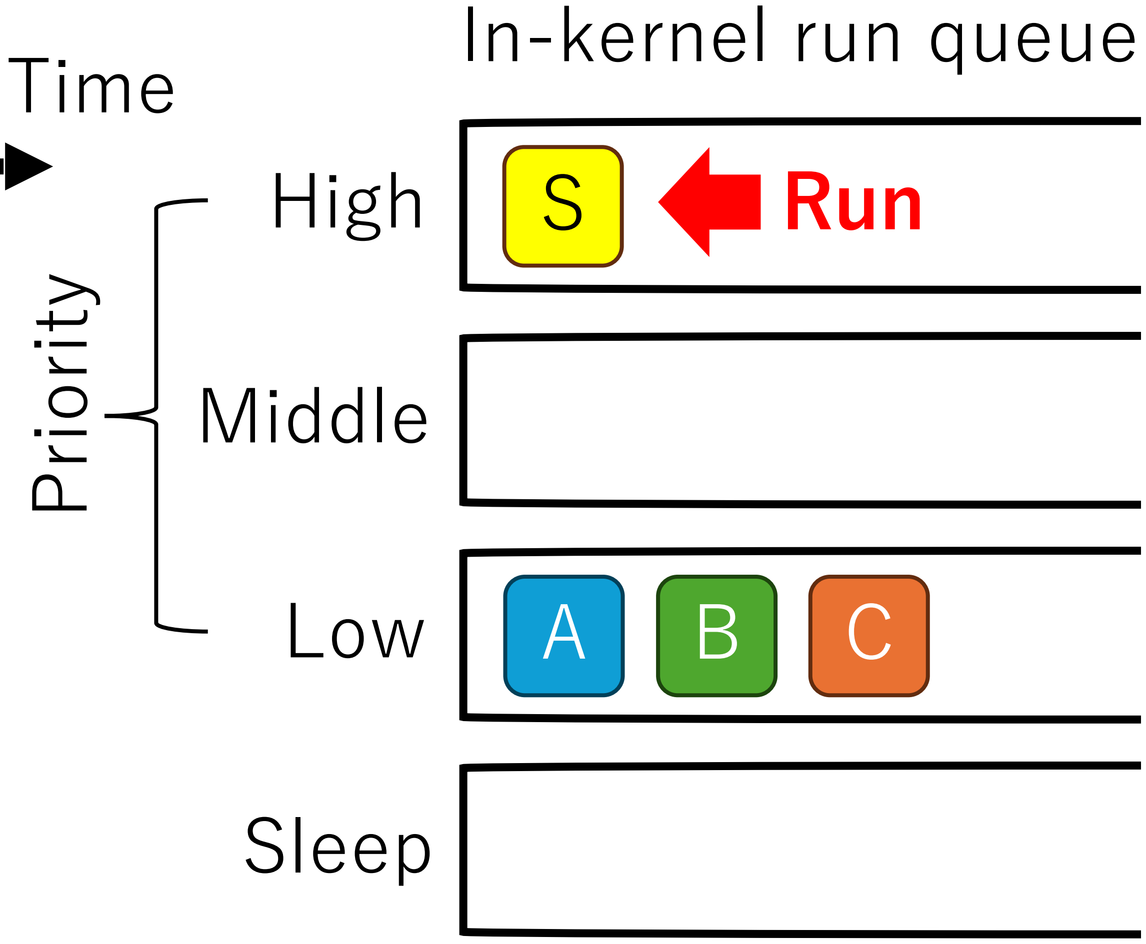


Priority Elevation（Shared Pattern）

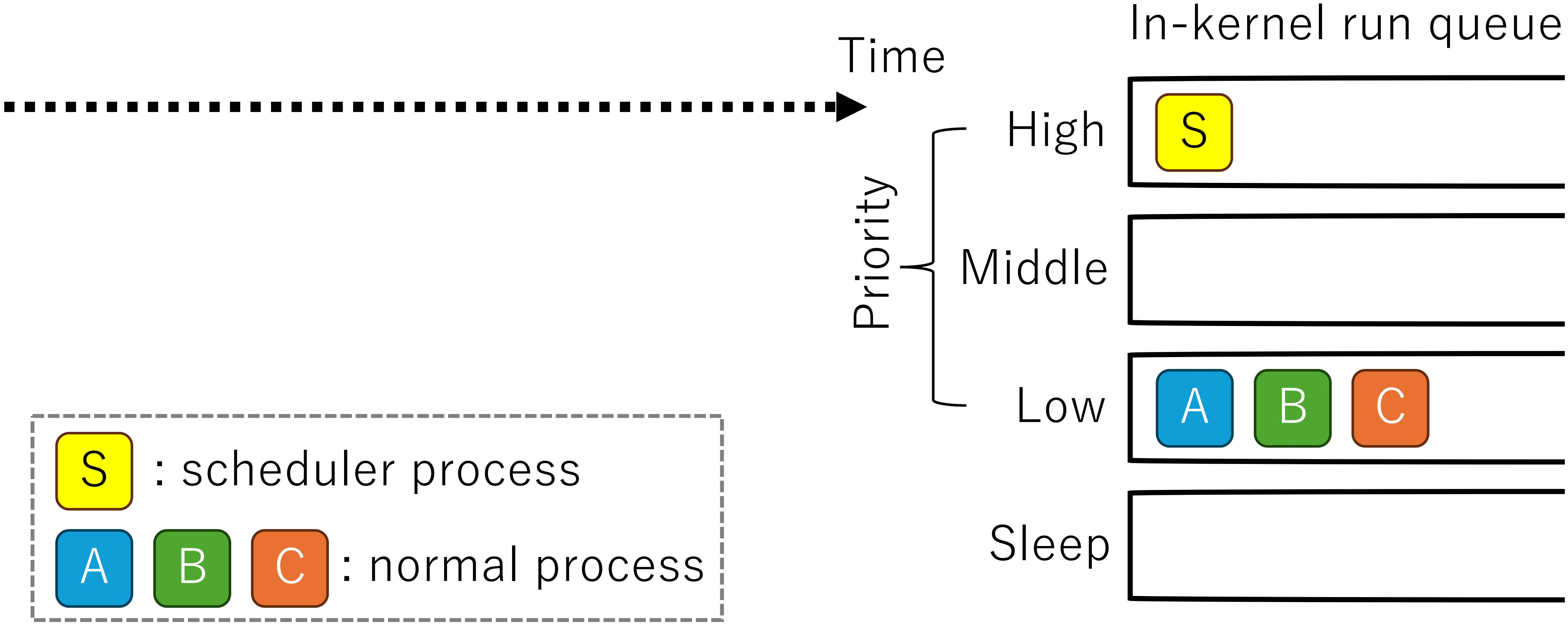


Continue this loop

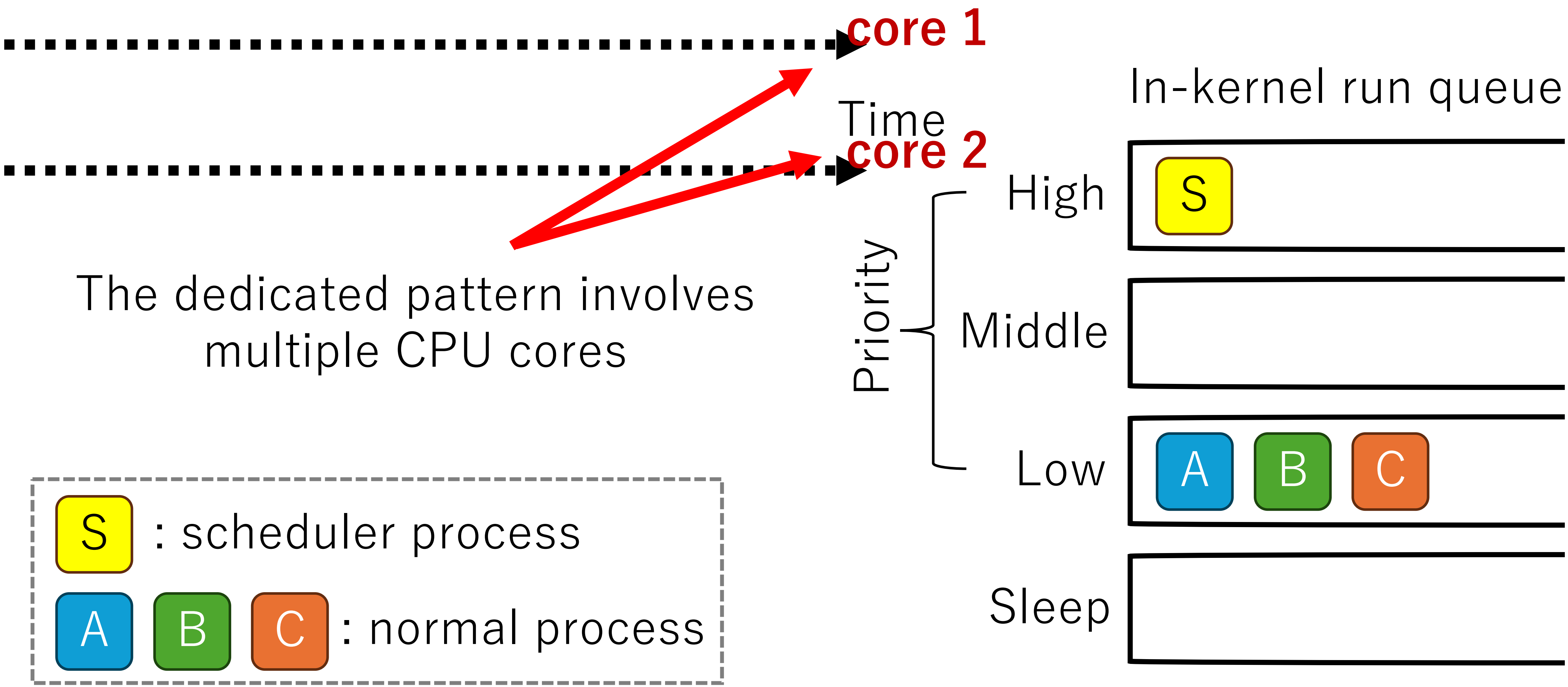




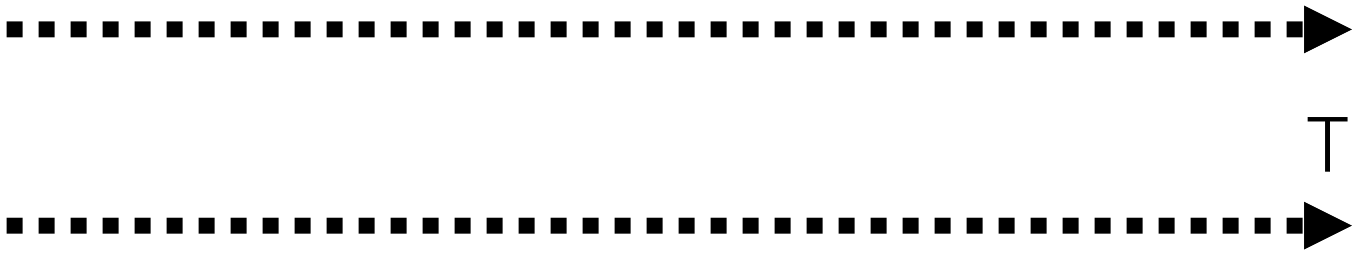
Priority Elevation (Dedicated Pattern)

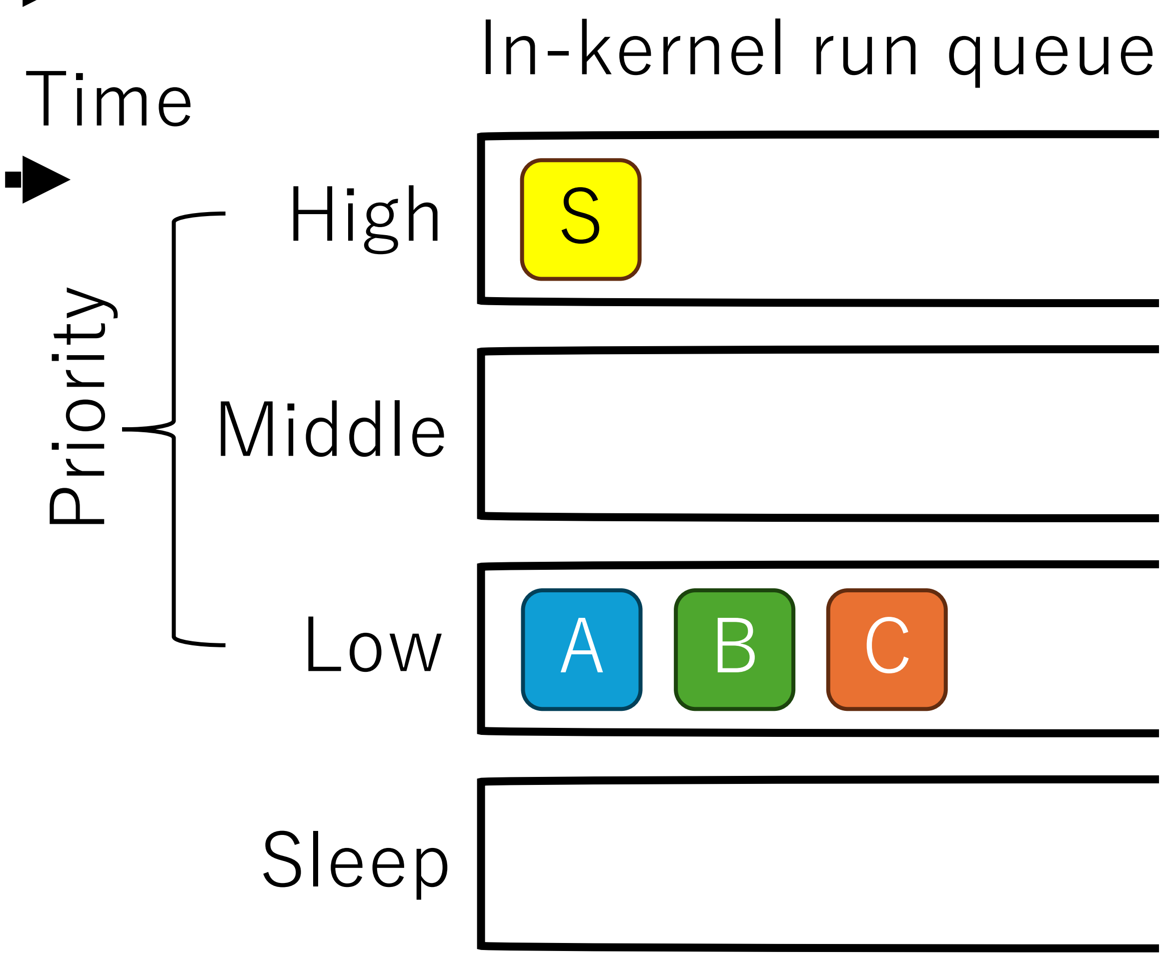


Priority Elevation（Dedicated Pattern)

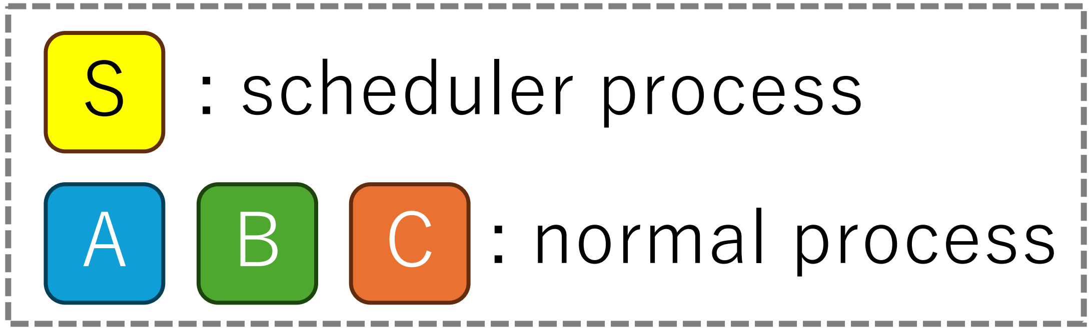


Priority Elevation (Dedicated Pattern)

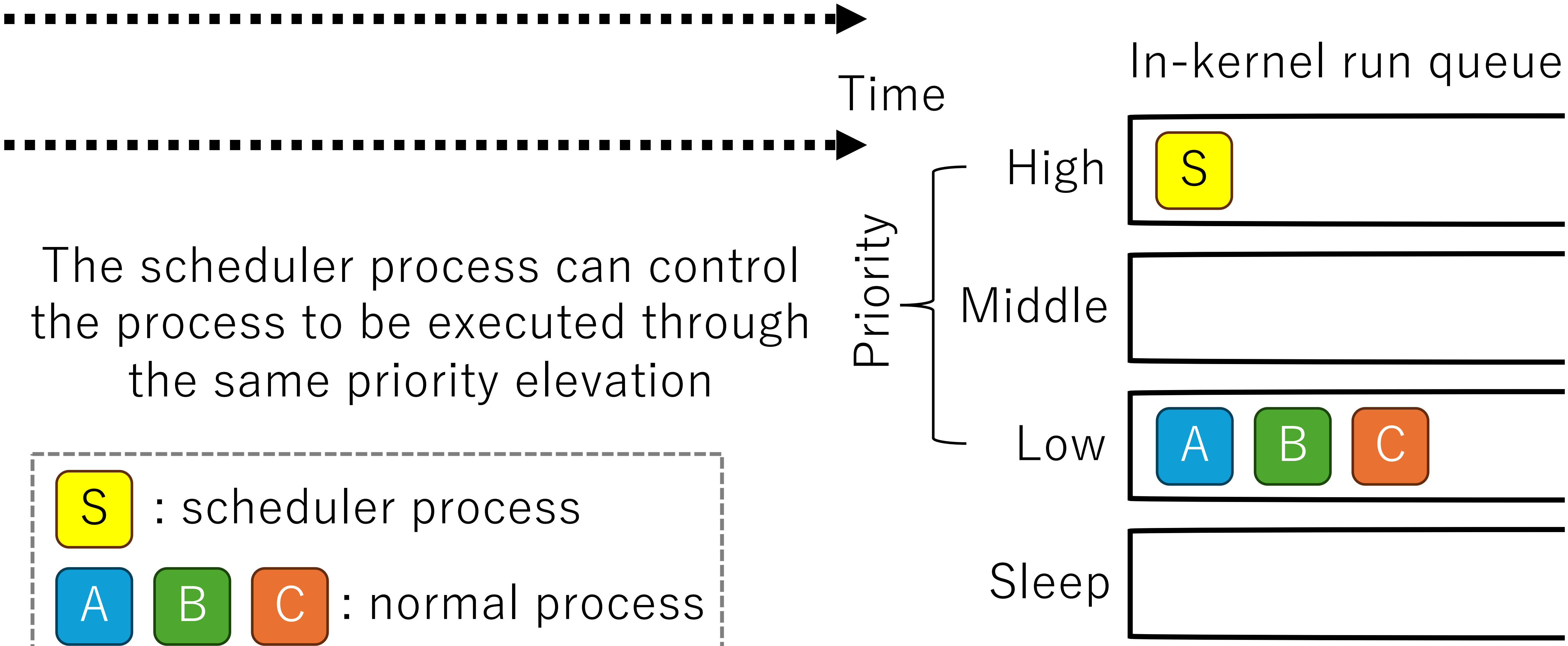




The basic behavior is the same as the shared pattern



Priority Elevation (Dedicated Pattern)

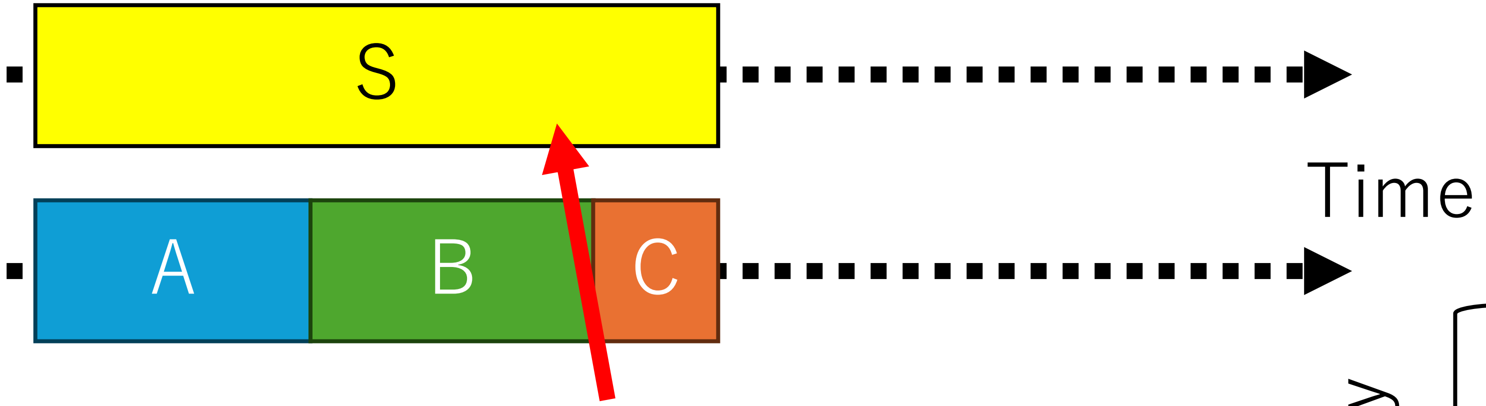


The scheduler process can control the process to be executed through the same priority elevation

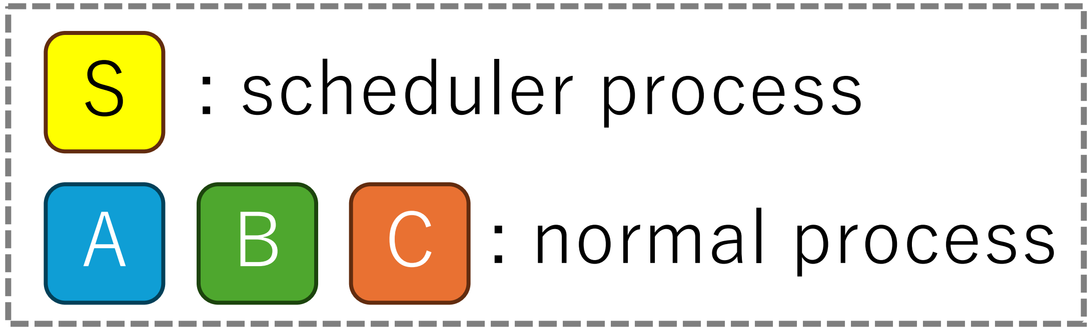
S : scheduler process

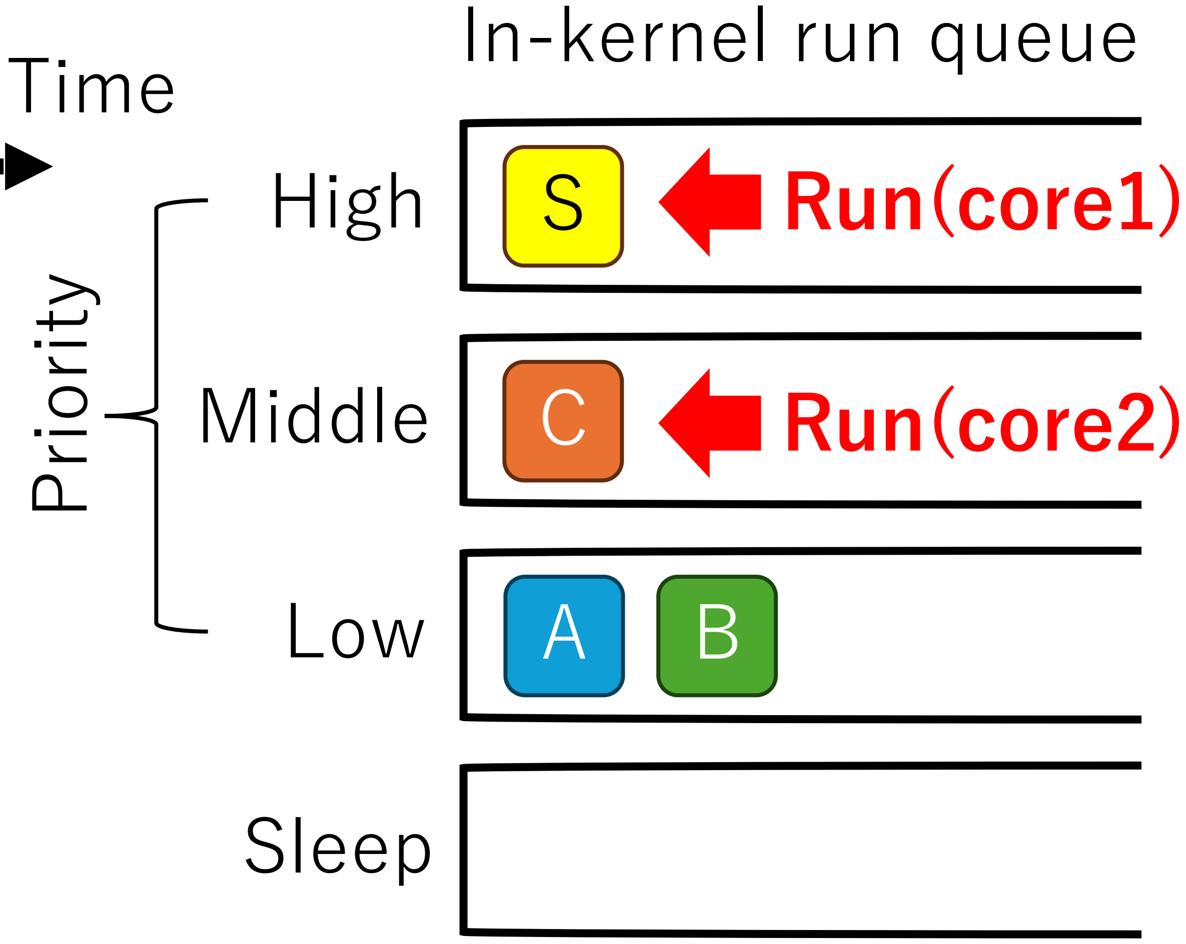
A B C : normal process

Priority Elevation（Dedicated Pattern)

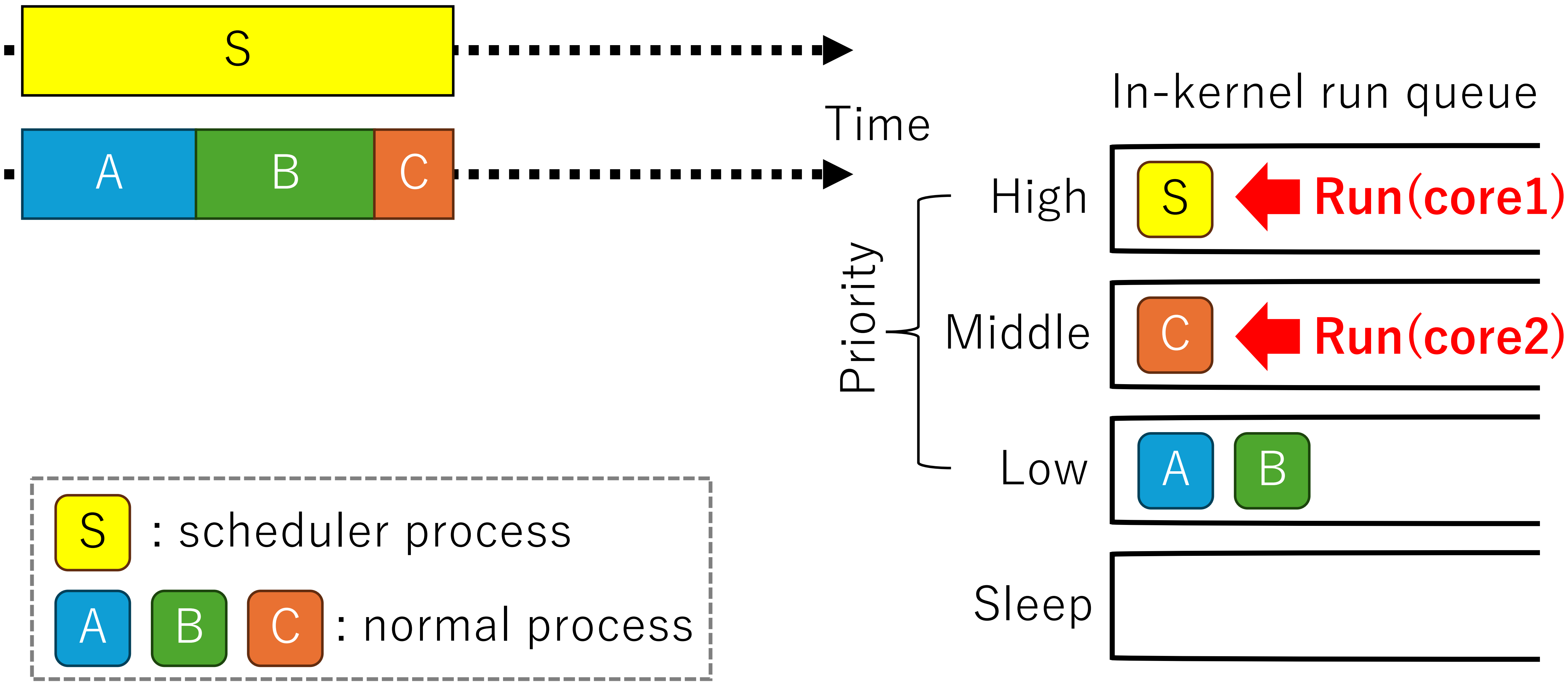


Difference from the shared pattern:
the scheduler process does not need to sleep to run a normal process

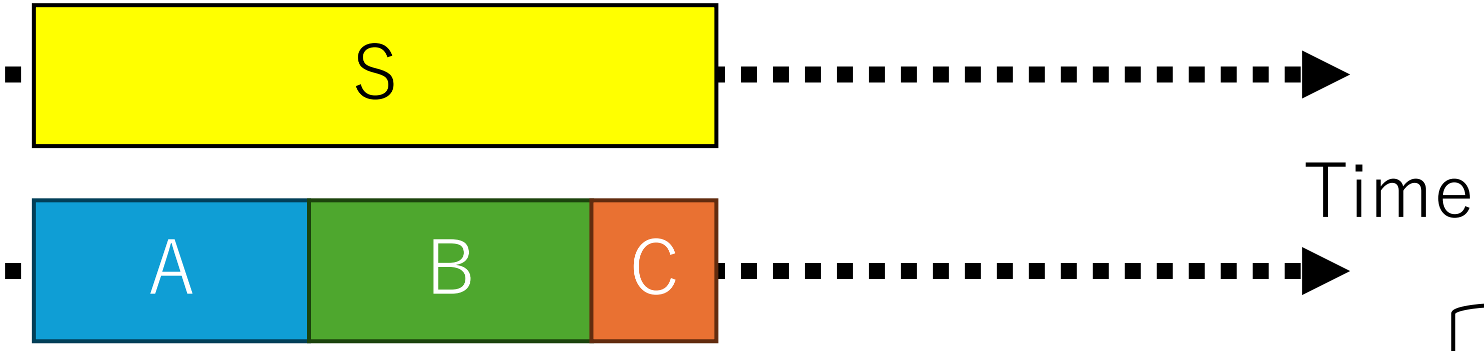


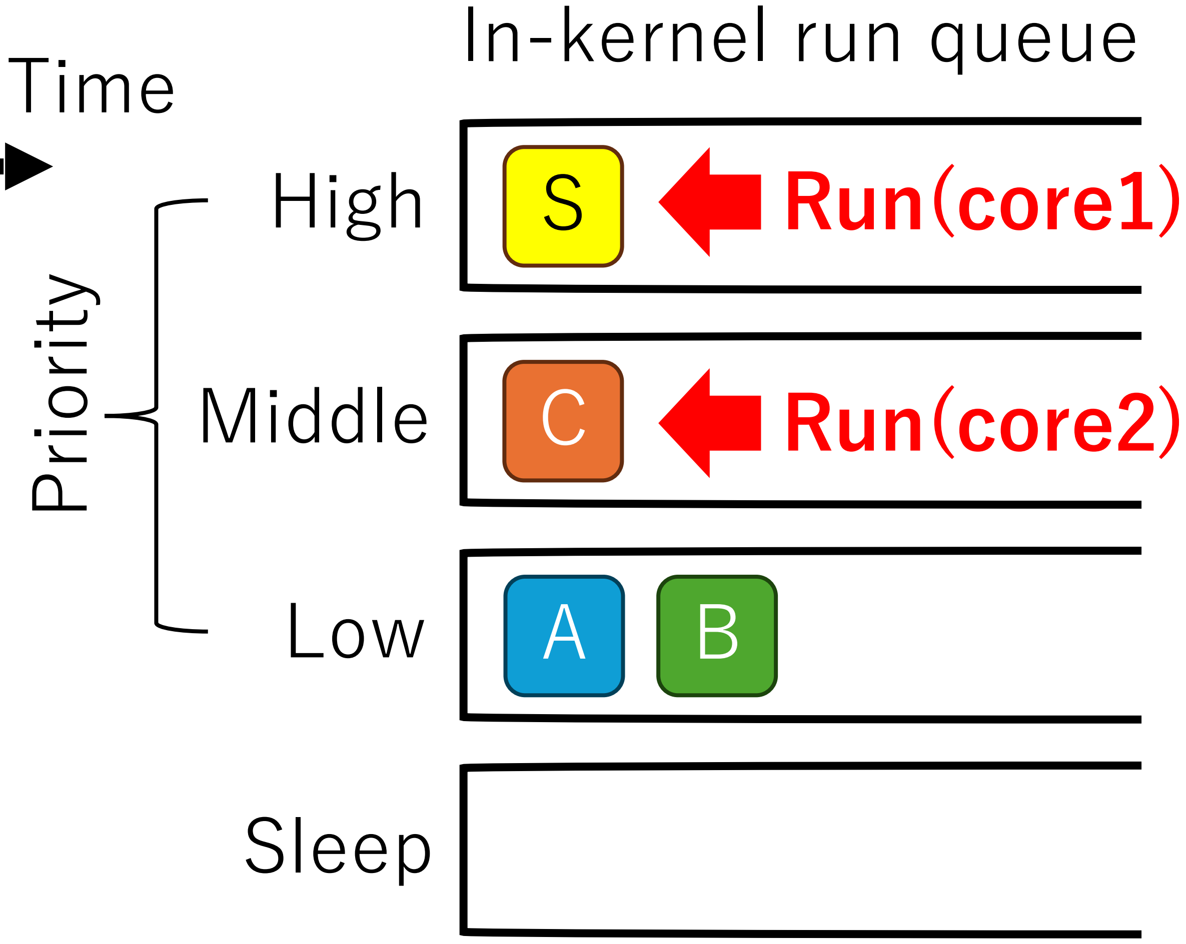


Limitation (for both shared and dedicated patterns)

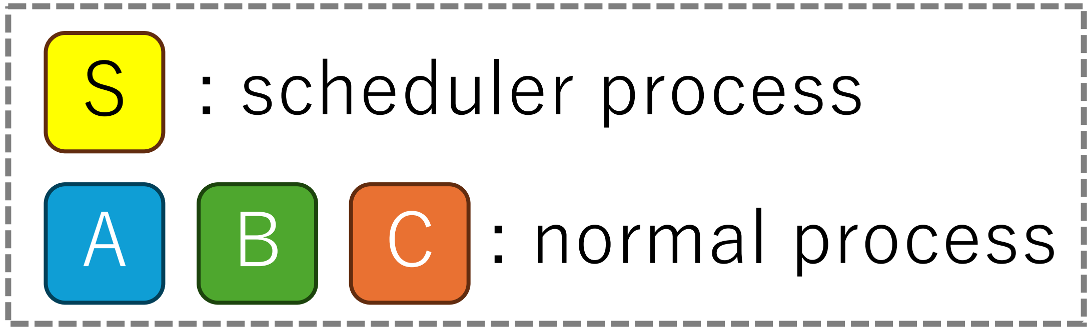


Limitation (for both shared and dedicated patterns)

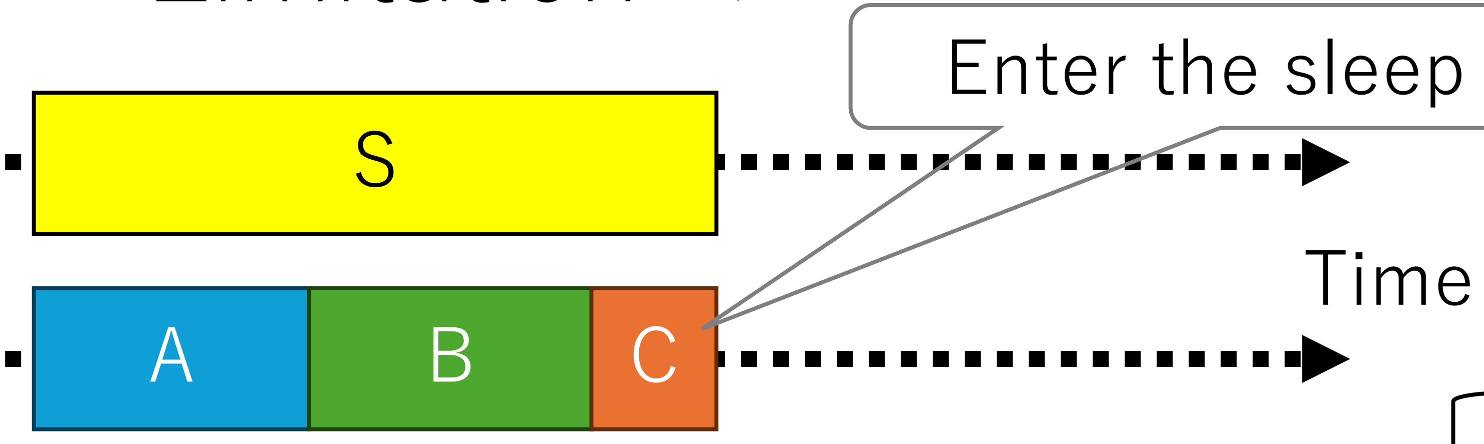


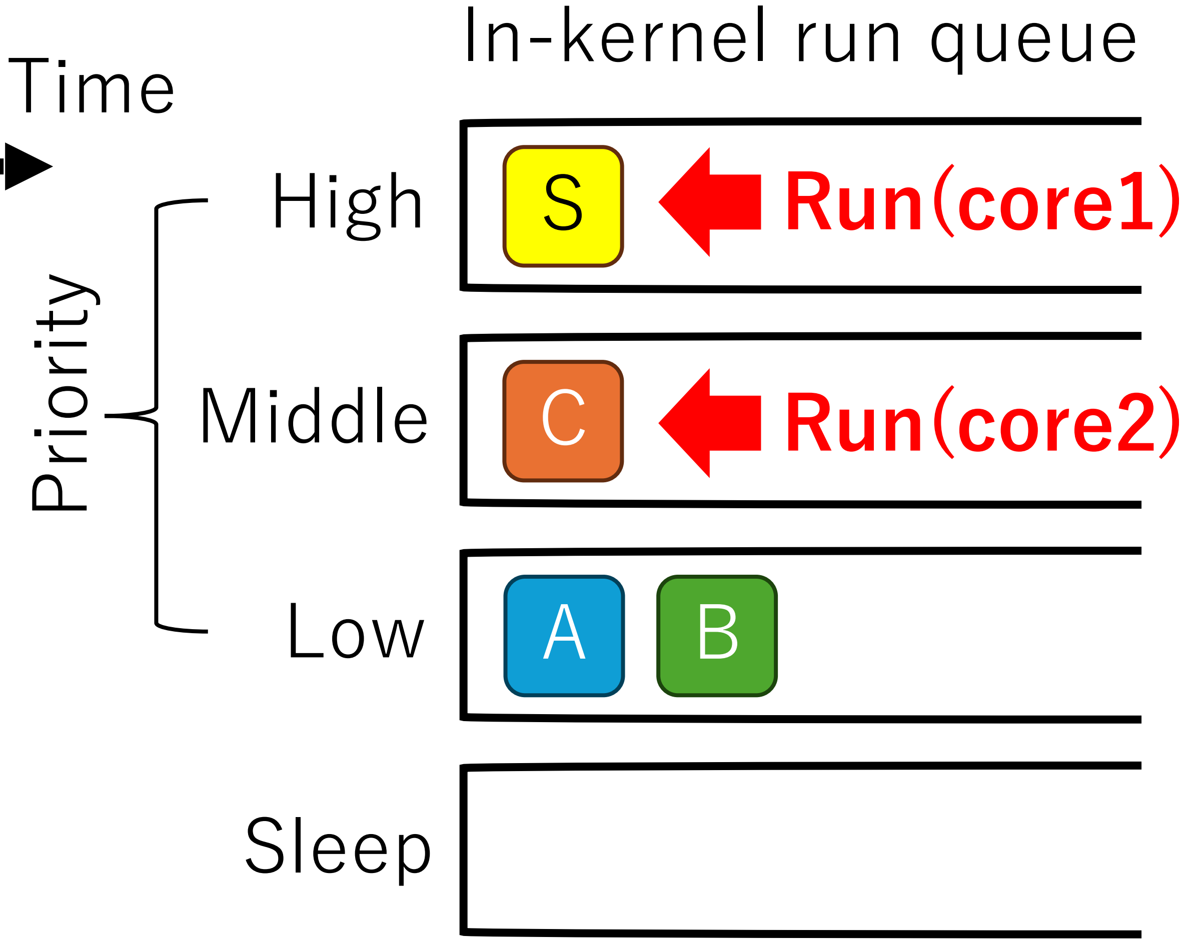


The scheduler process does not have a way to effectively detect a normal process's sleep/wake up

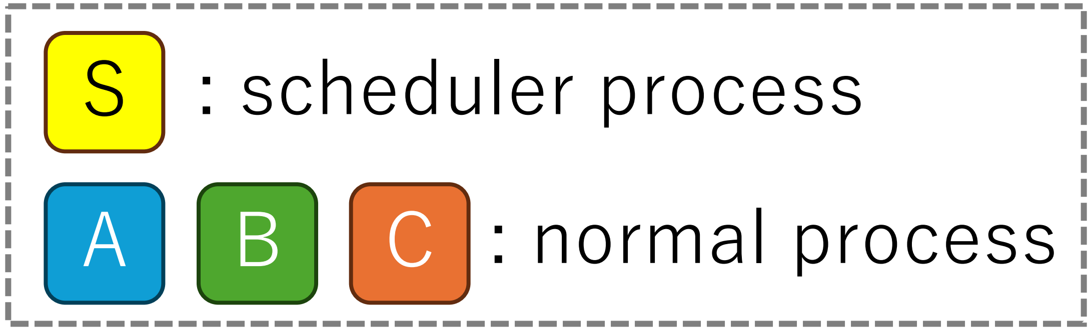


Limitation (for both shared and dedicated patterns)

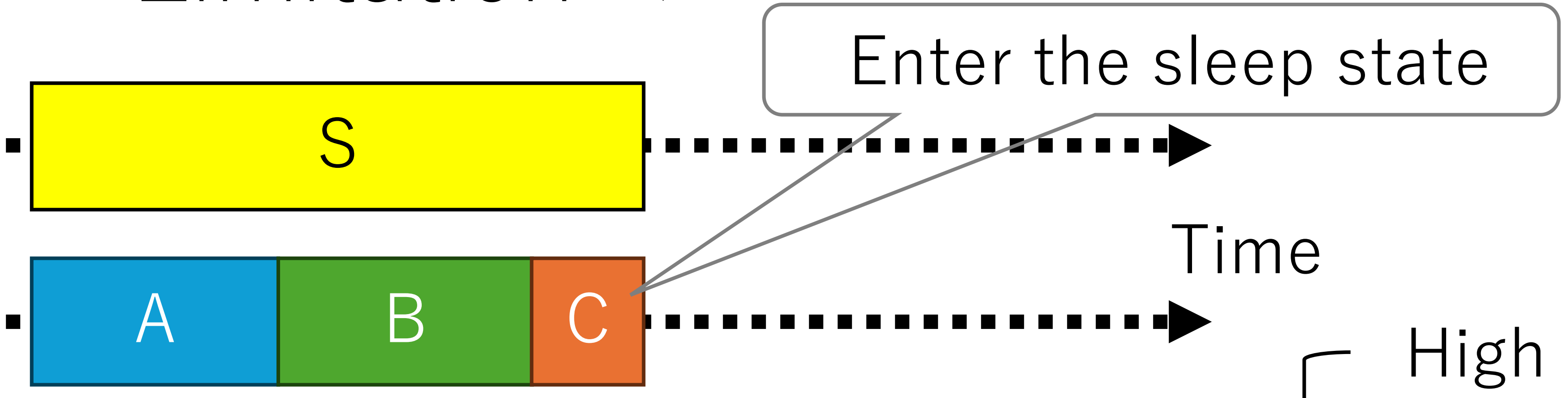


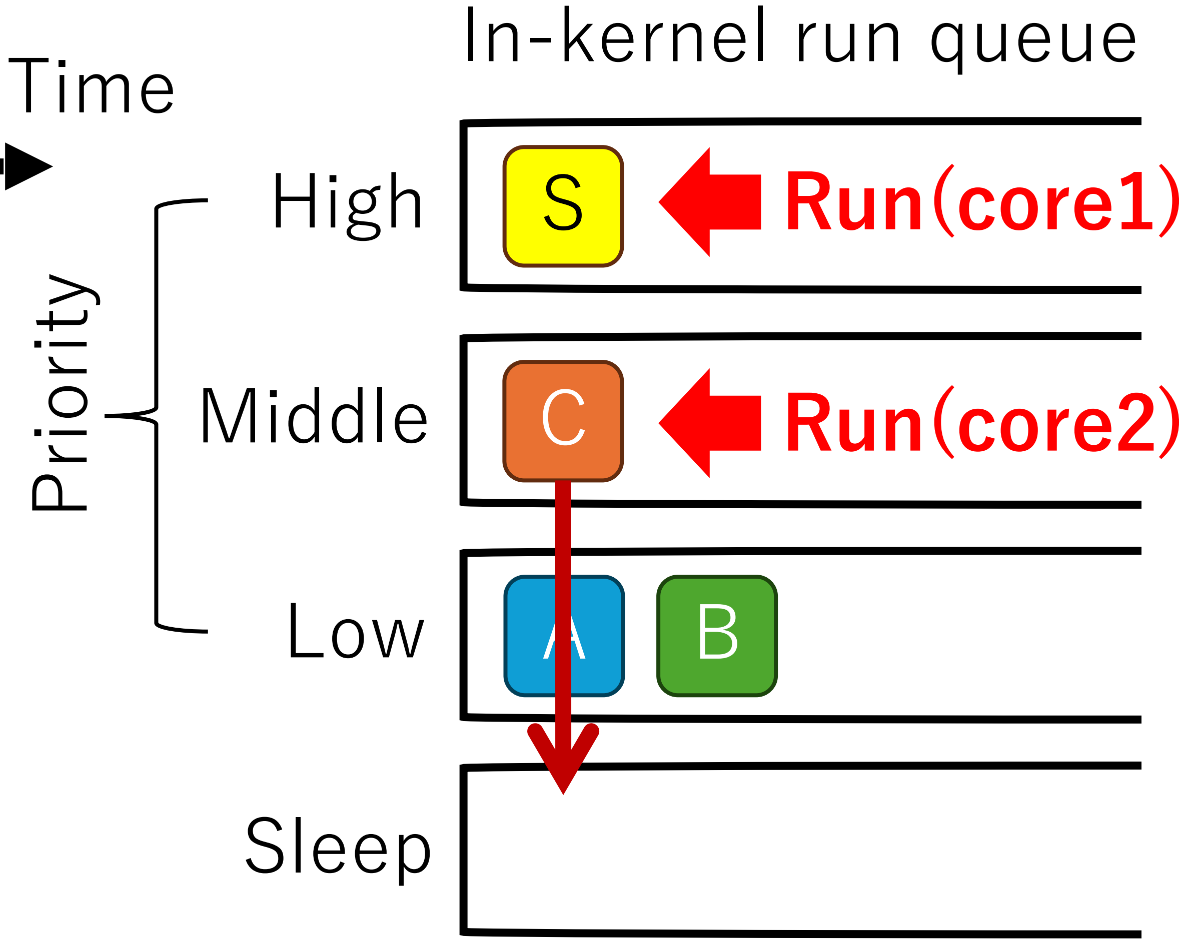


The scheduler process does not have a way to effectively detect a normal process's sleep/wake up



Limitation (for both shared and dedicated patterns)



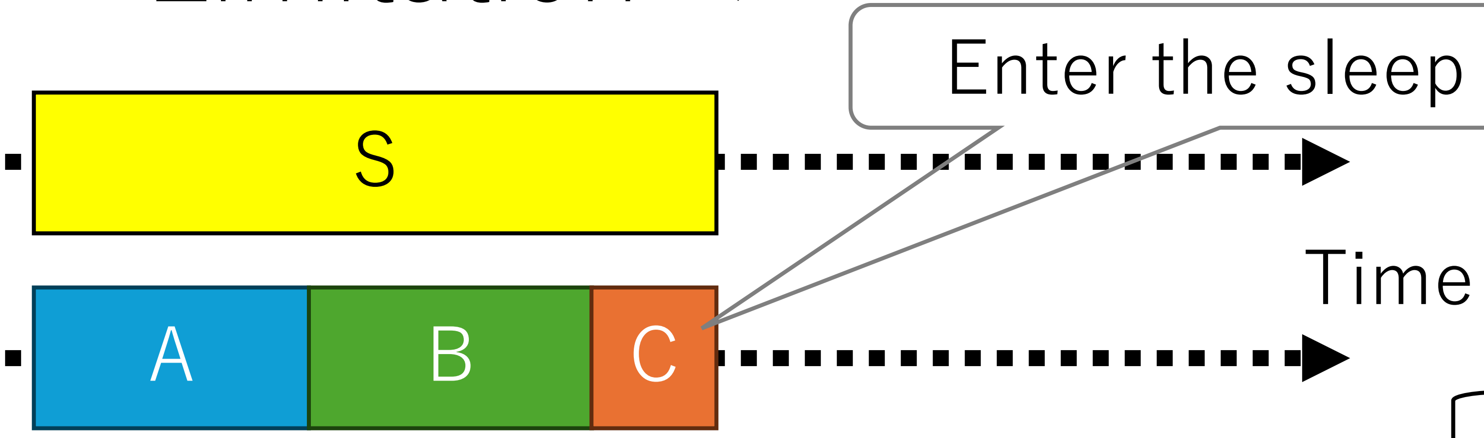


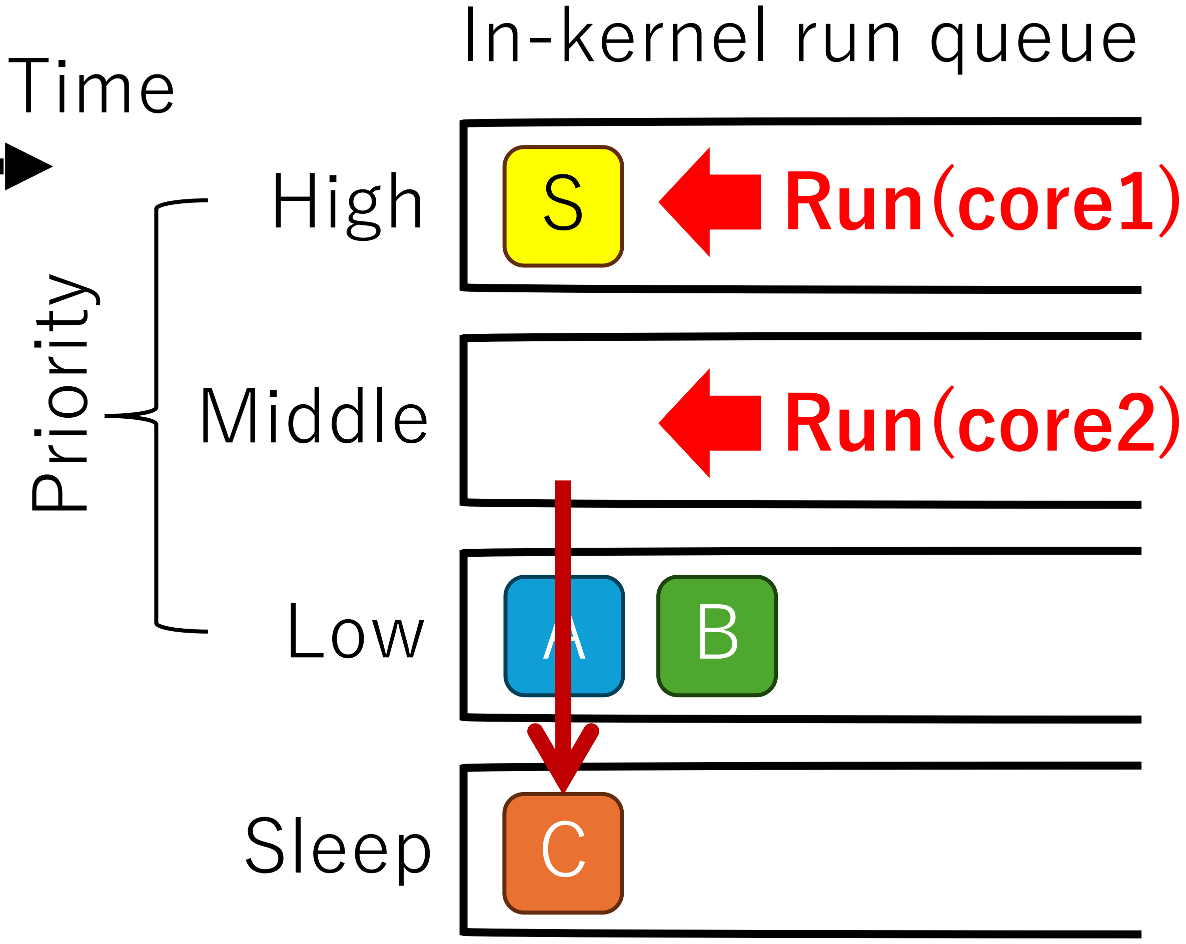
The scheduler process does not have a way to effectively detect a normal process's sleep/wake up

S : scheduler process

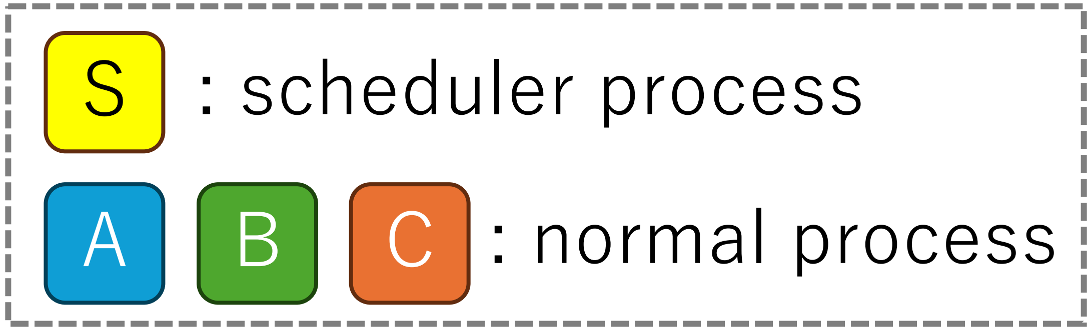
A B C : normal process

Limitation (for both shared and dedicated patterns)

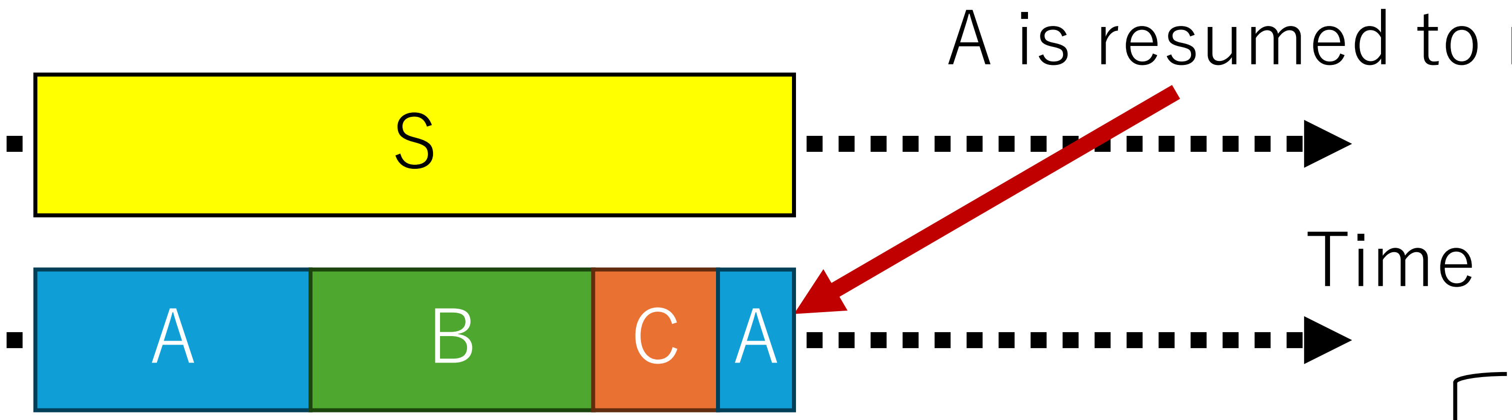




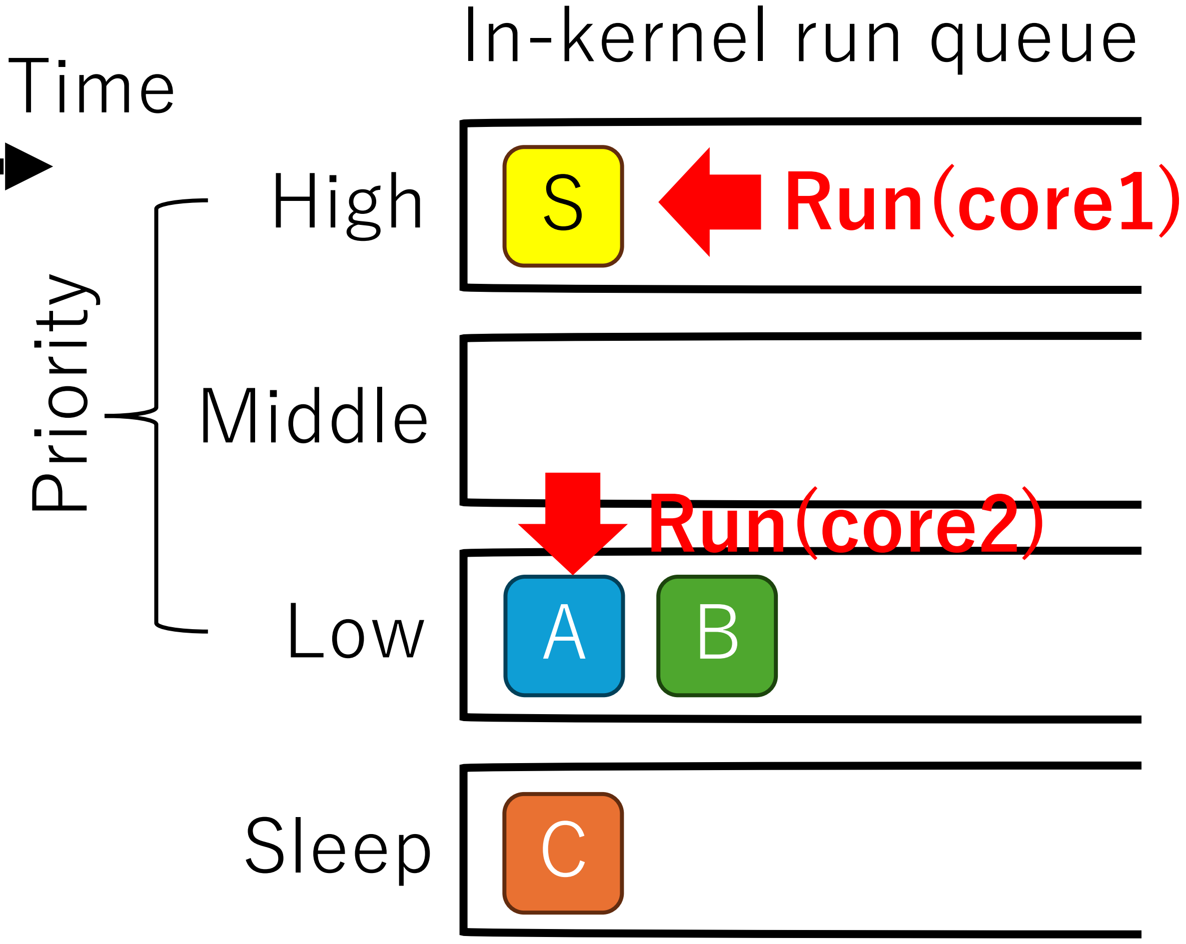
The scheduler process does not have
a way to effectively detect
a normal process's sleep/wake up

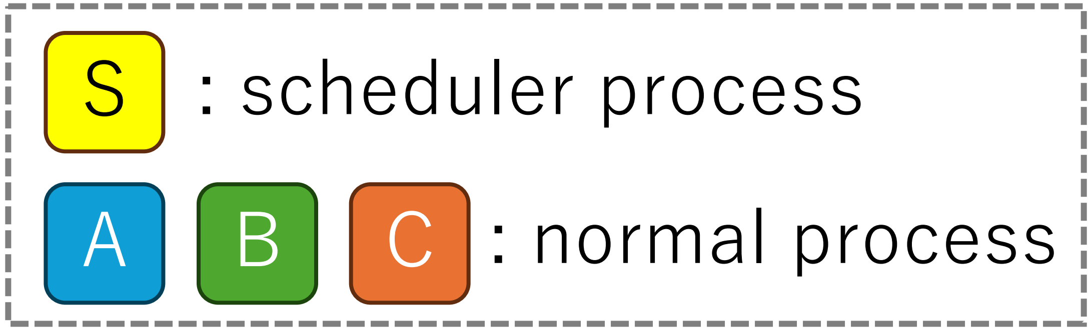


Limitation (for both shared and dedicated patterns)

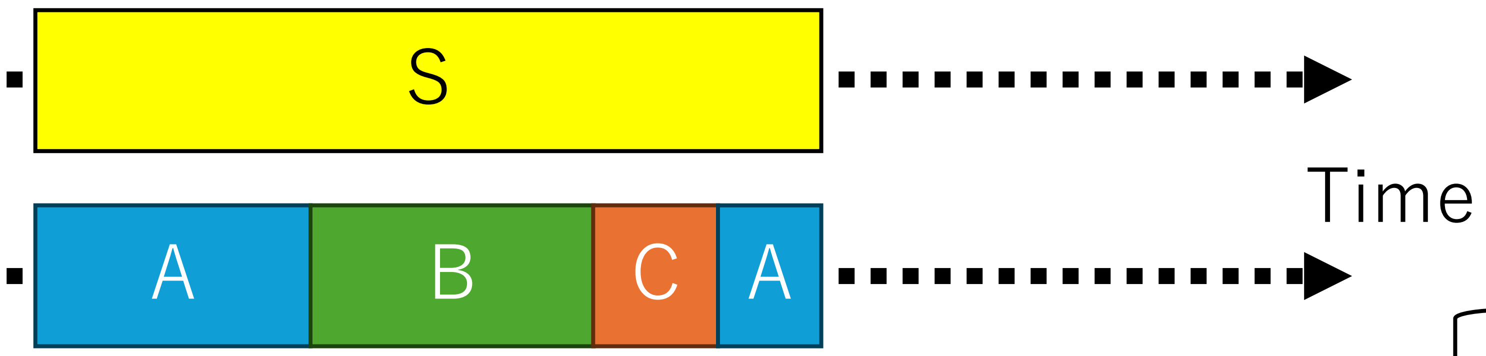


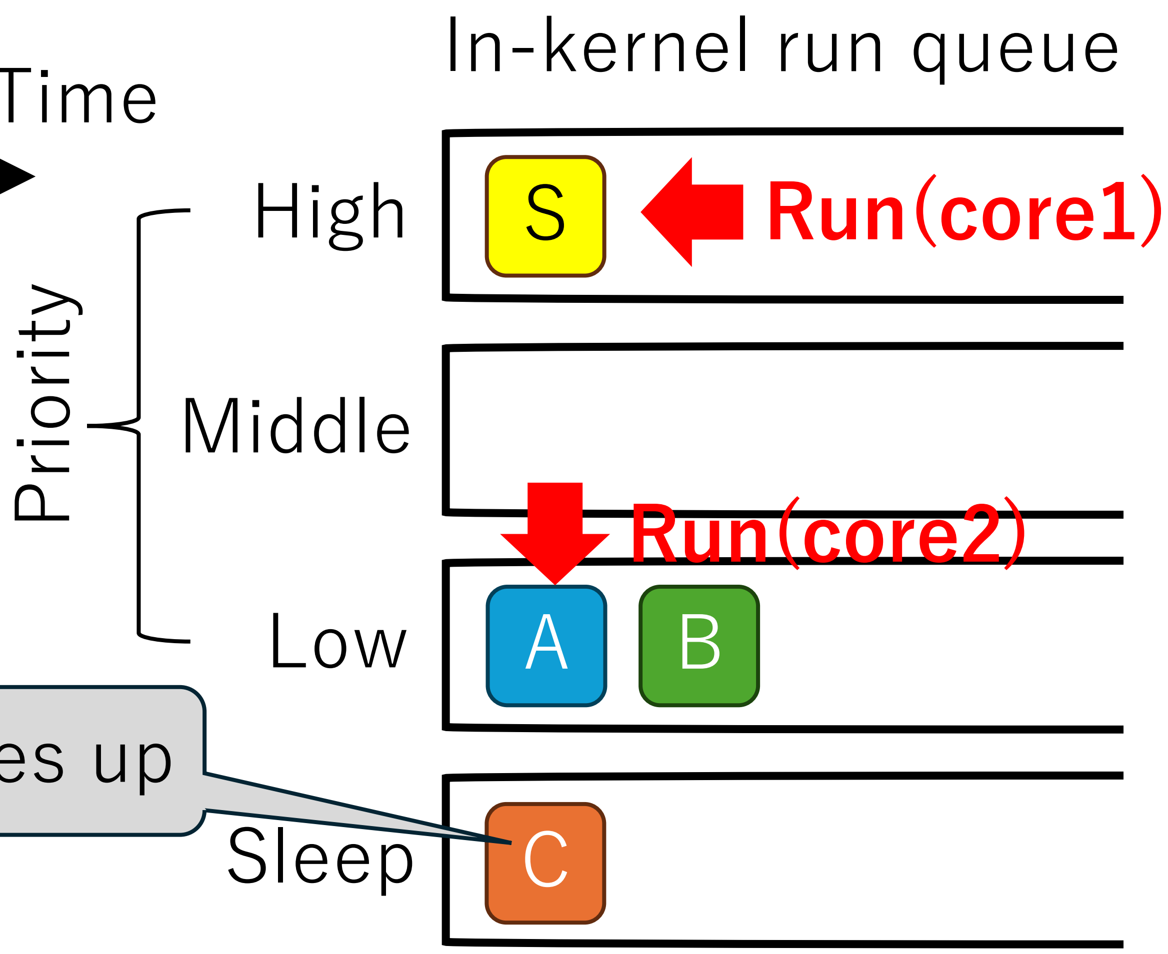
The scheduler process does not have
a way to effectively detect
a normal process’s sleep/wake up



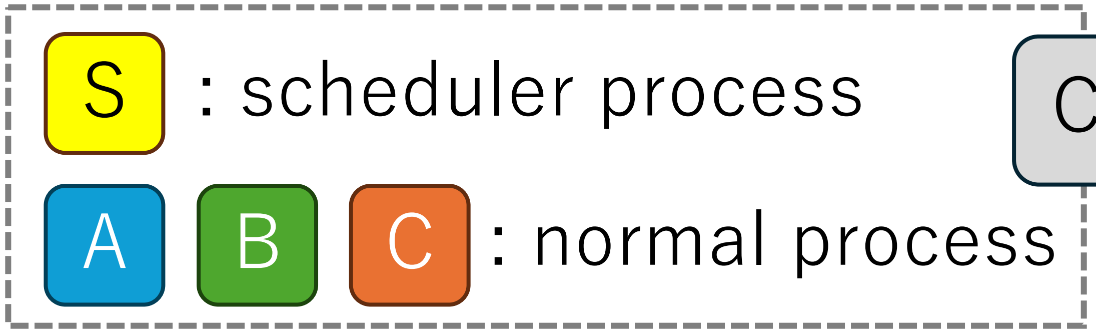


Limitation (for both shared and dedicated patterns)

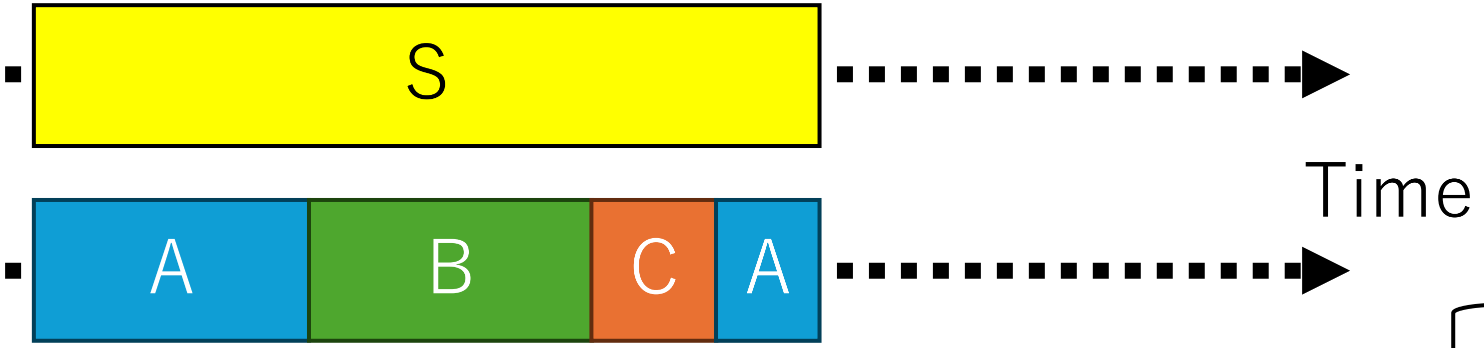




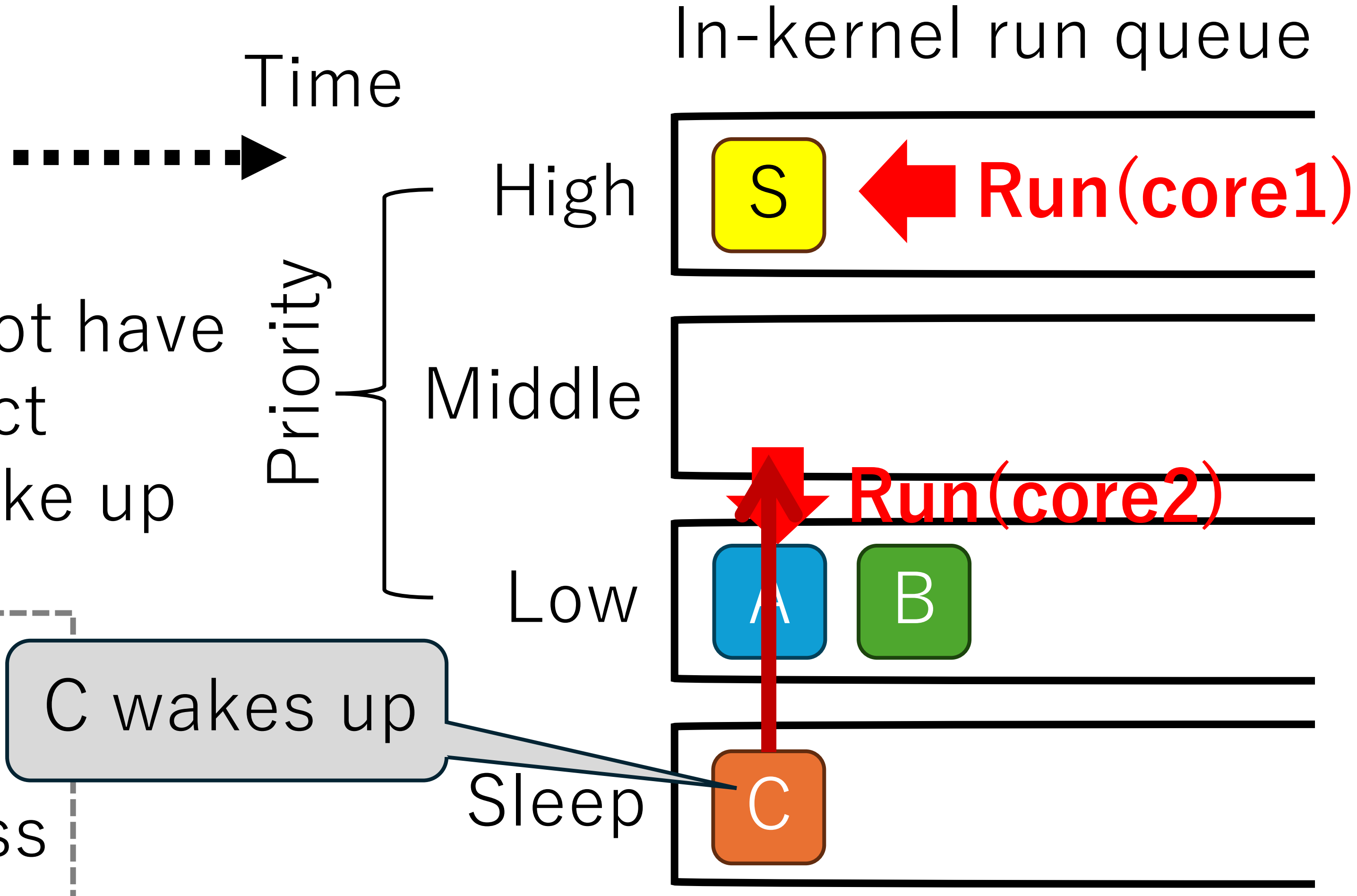
The scheduler process does not have a way to effectively detect a normal process's sleep/wake up

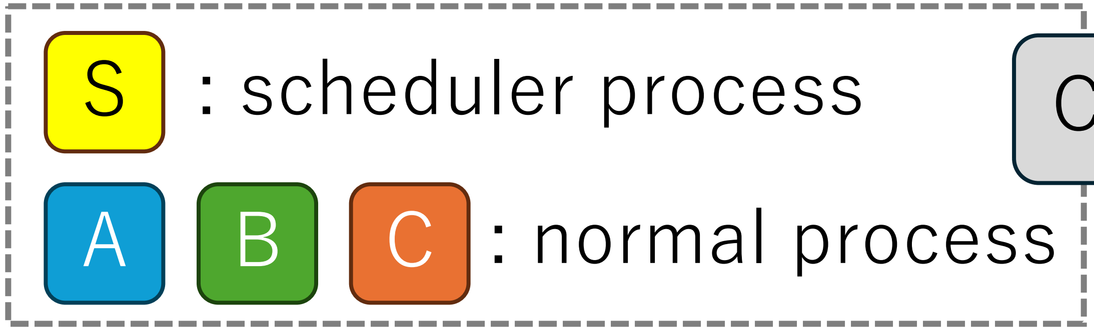


Limitation (for both shared and dedicated patterns)

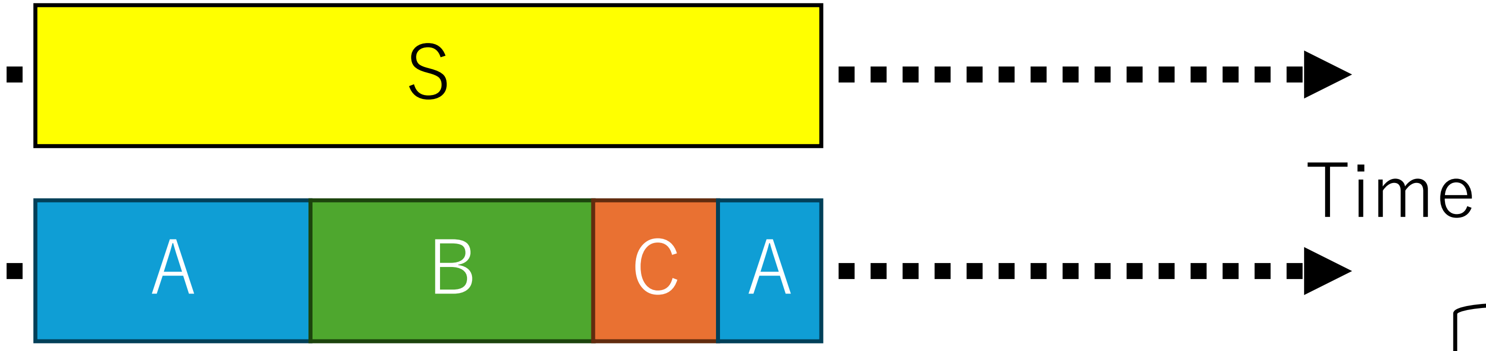


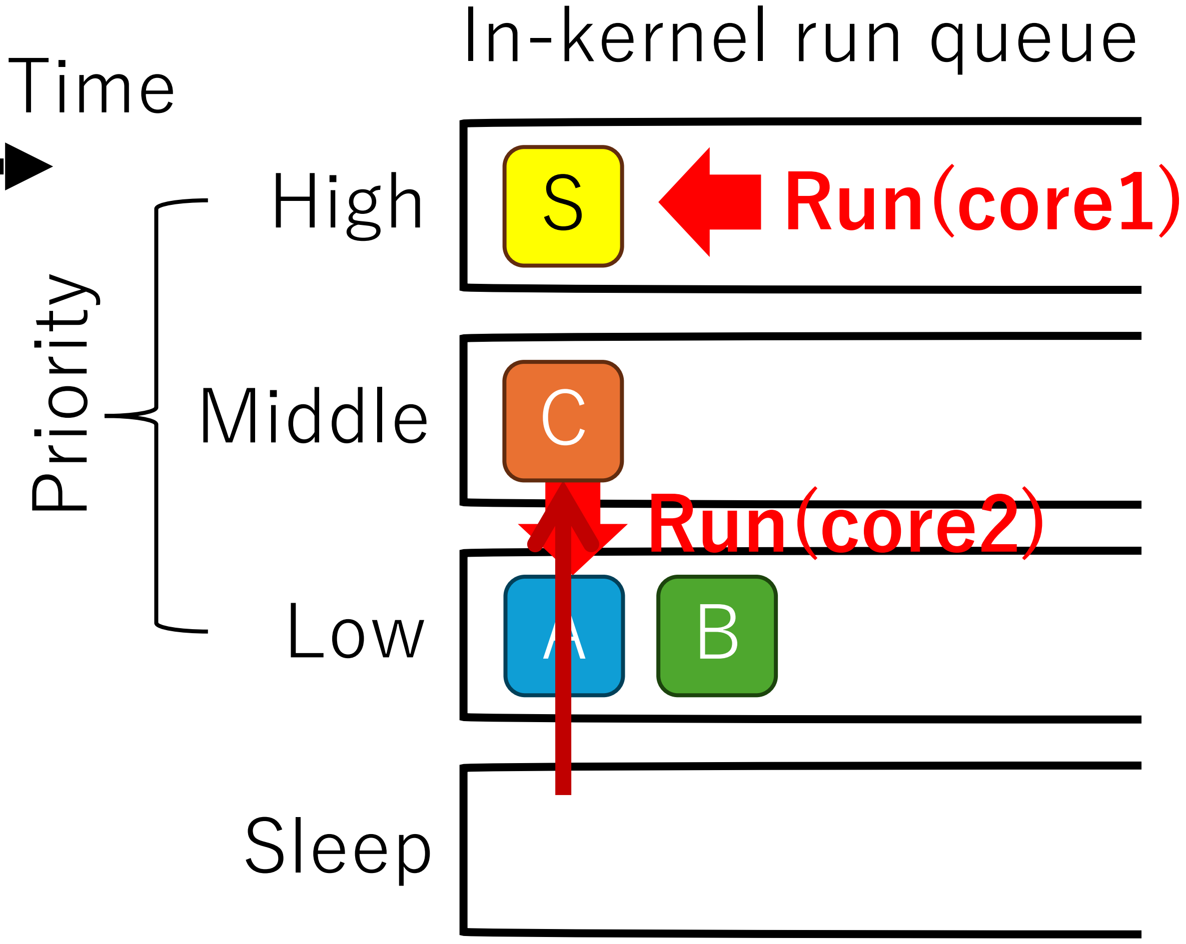
The scheduler process does not have a way to effectively detect a normal process's sleep/wake up



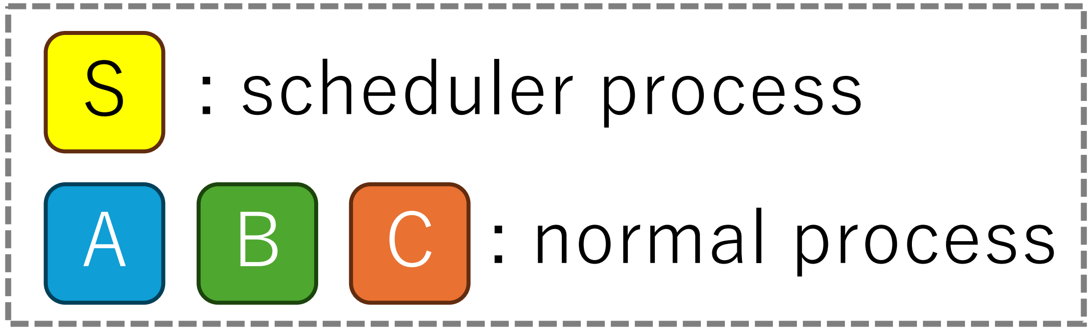


Limitation (for both shared and dedicated patterns)

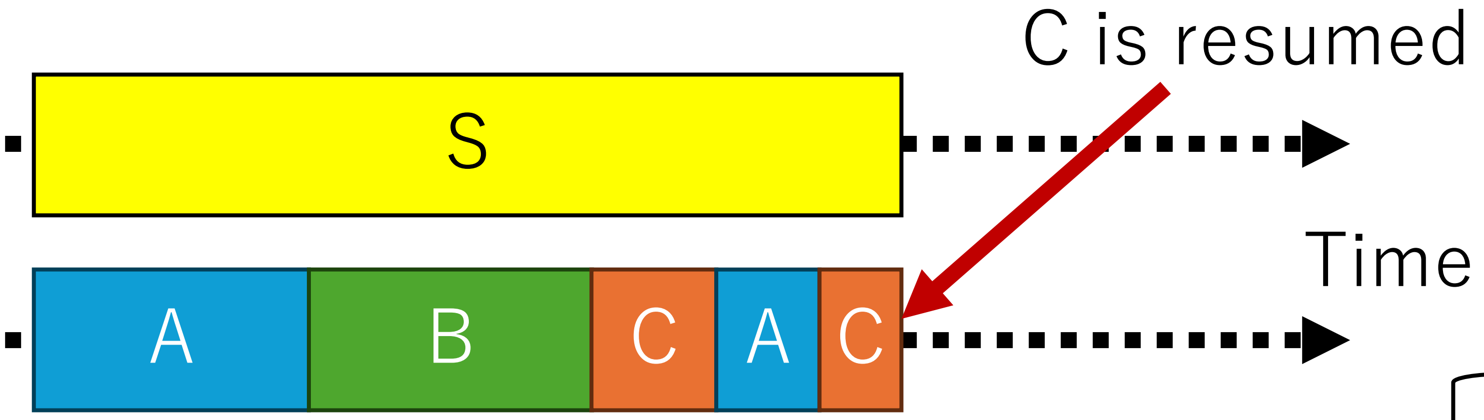


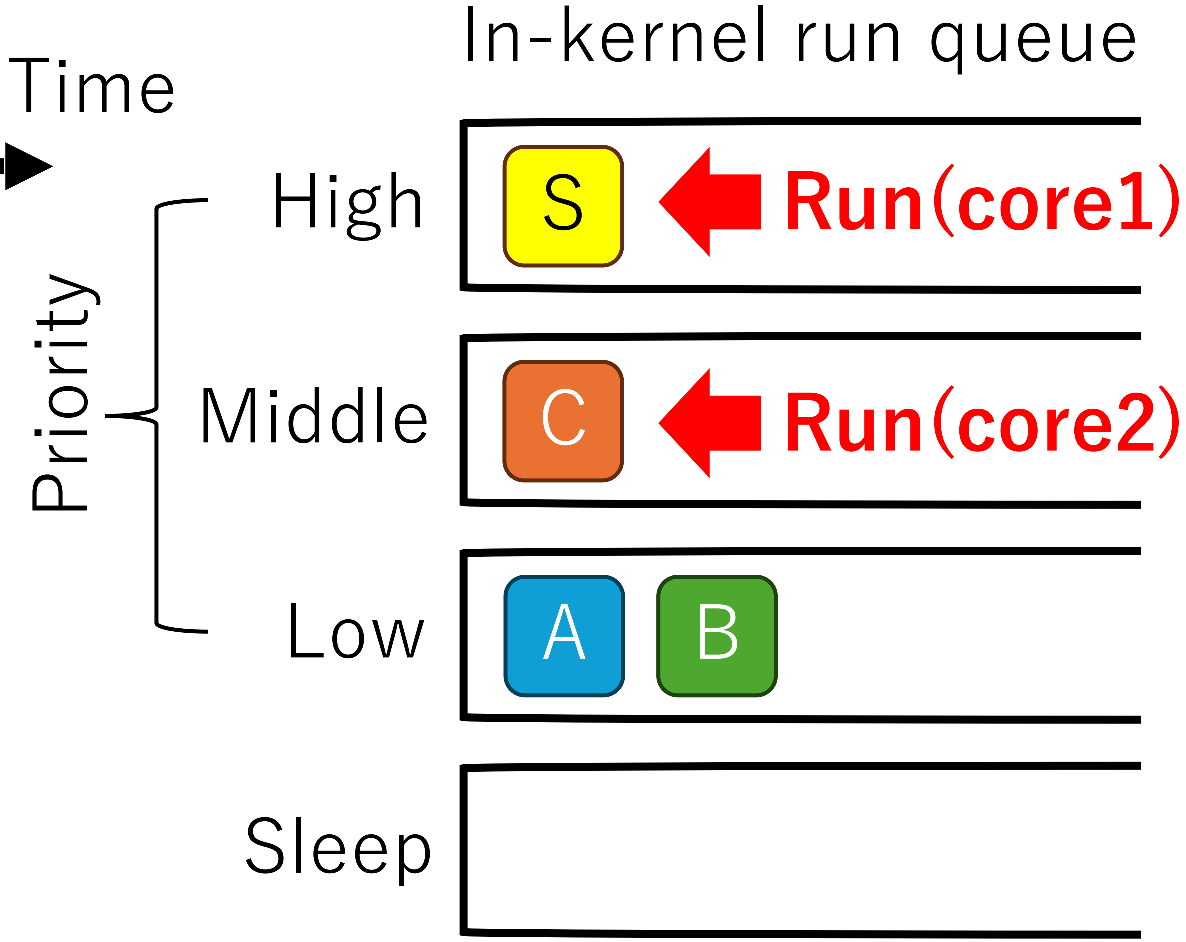


The scheduler process does not have a way to effectively detect a normal process's sleep/wake up

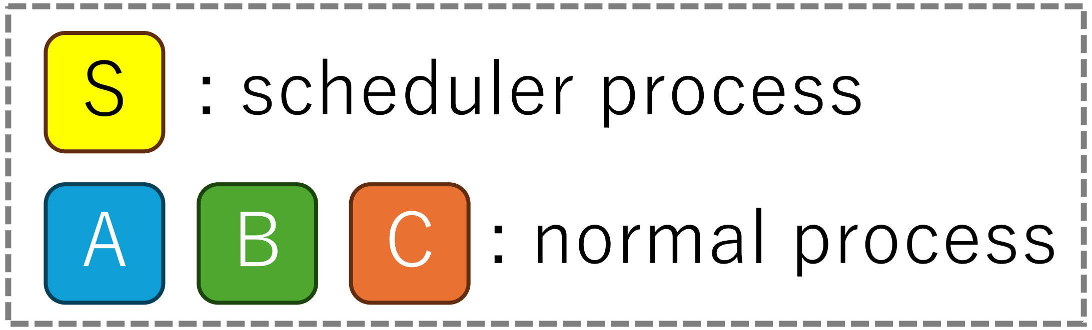


Limitation (for both shared and dedicated patterns)

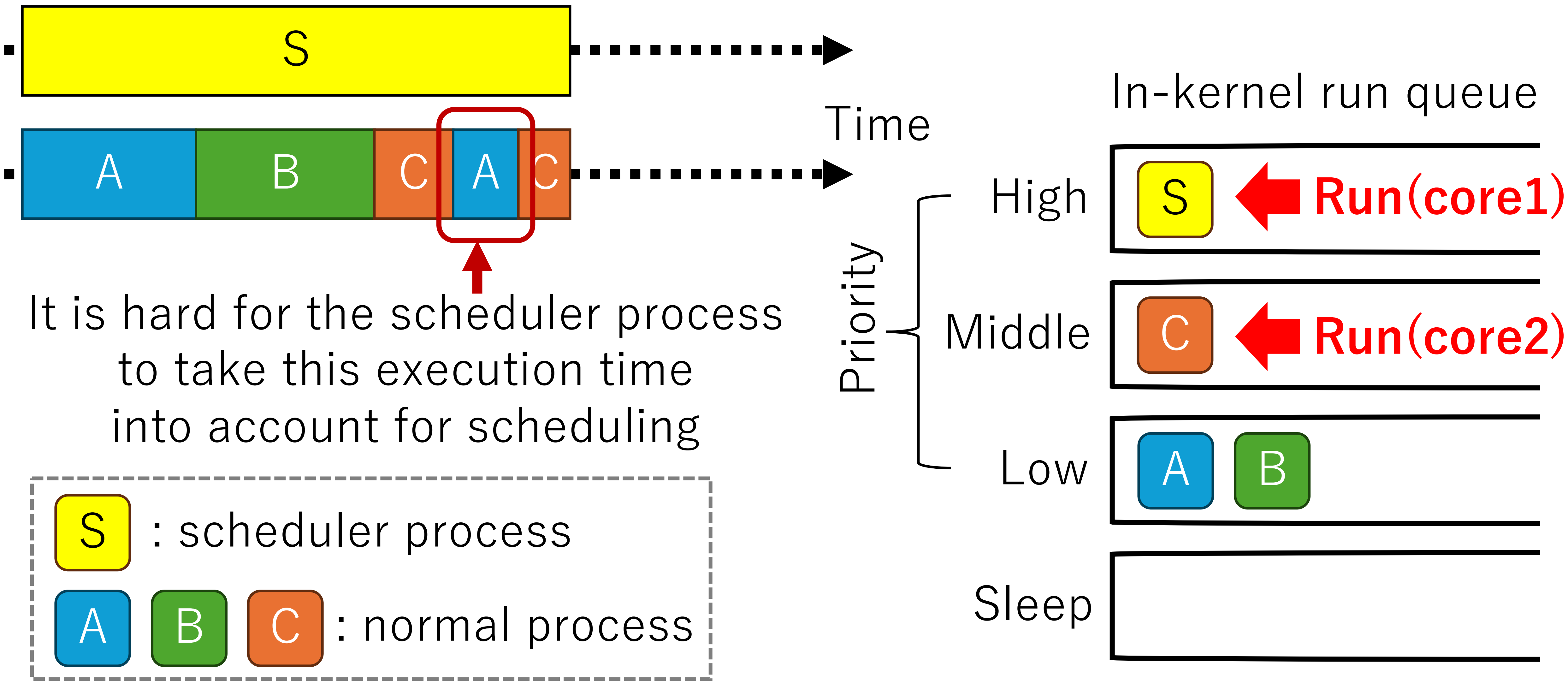




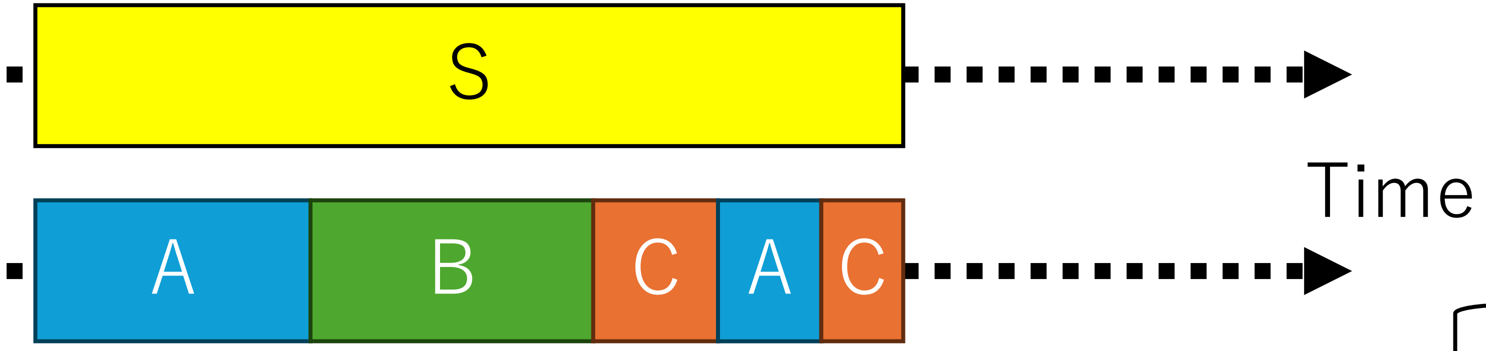
The scheduler process does not have a way to effectively detect a normal process's sleep/wake up

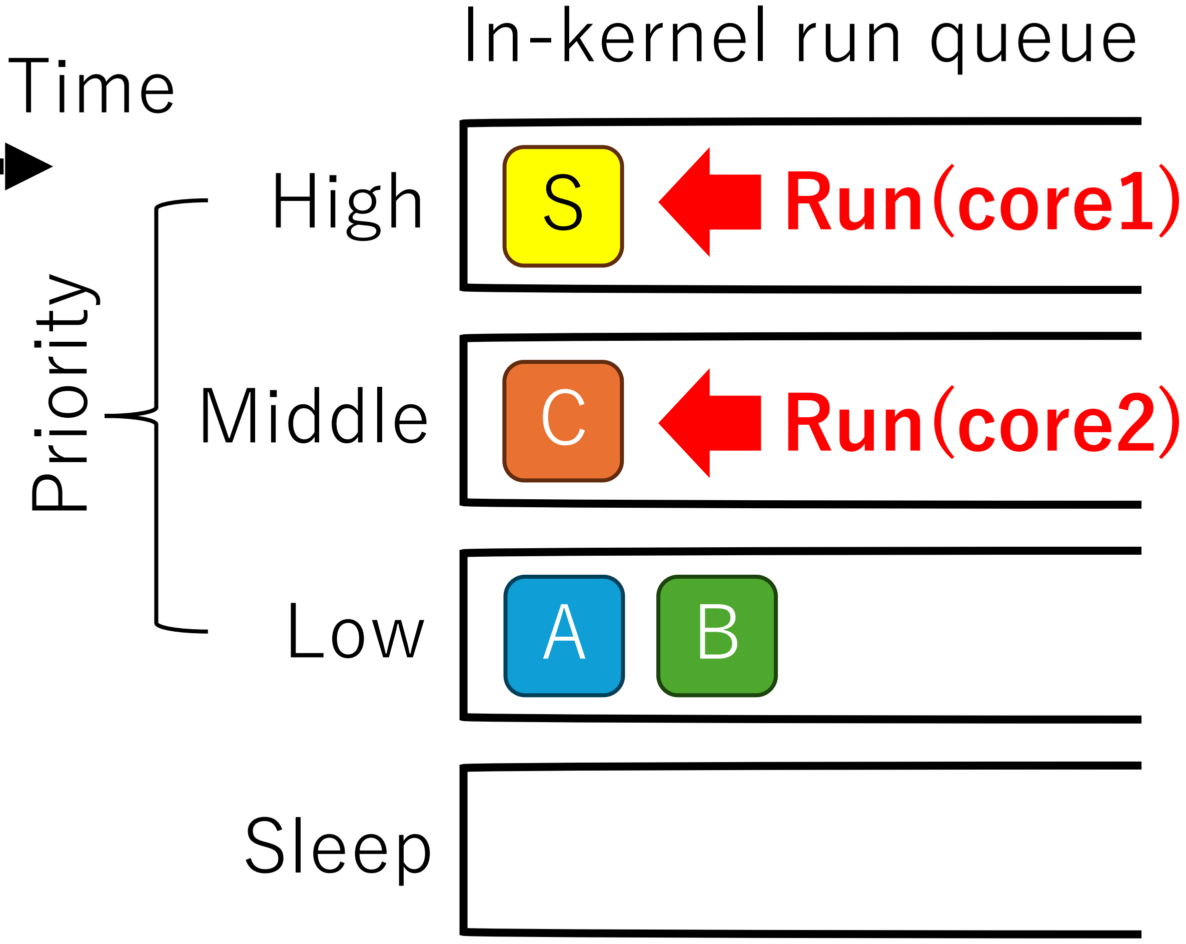


Limitation (for both shared and dedicated patterns)



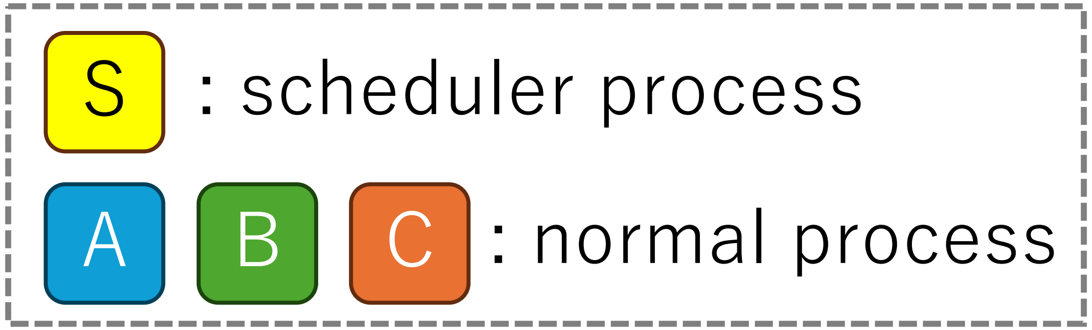
Limitation (for both shared and dedicated patterns)





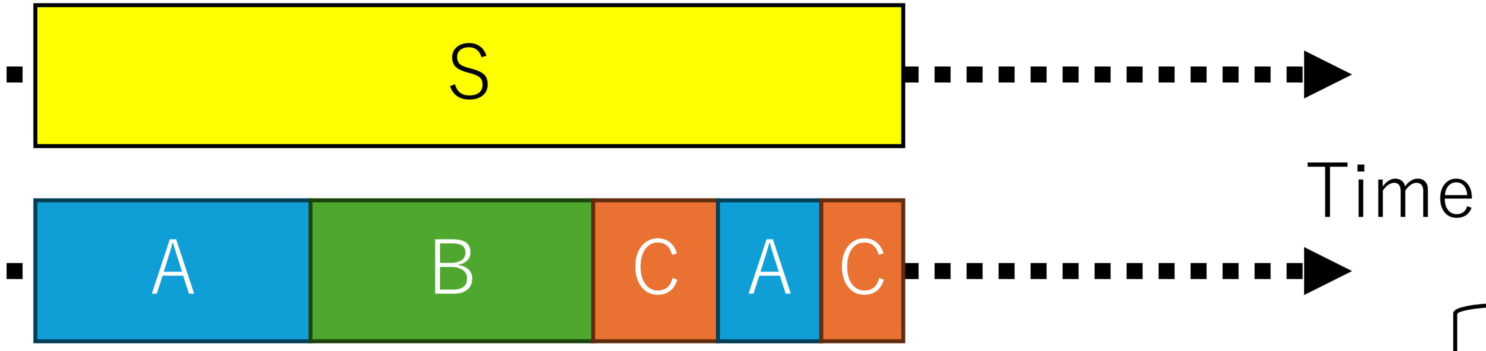
Limitation

The priority elevation trick is not appropriate for scheduling processes heavily relying on kernel event APIs



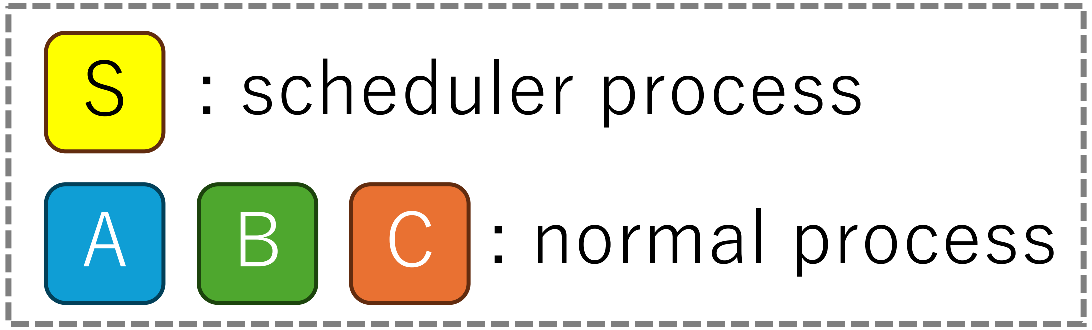
Limitation

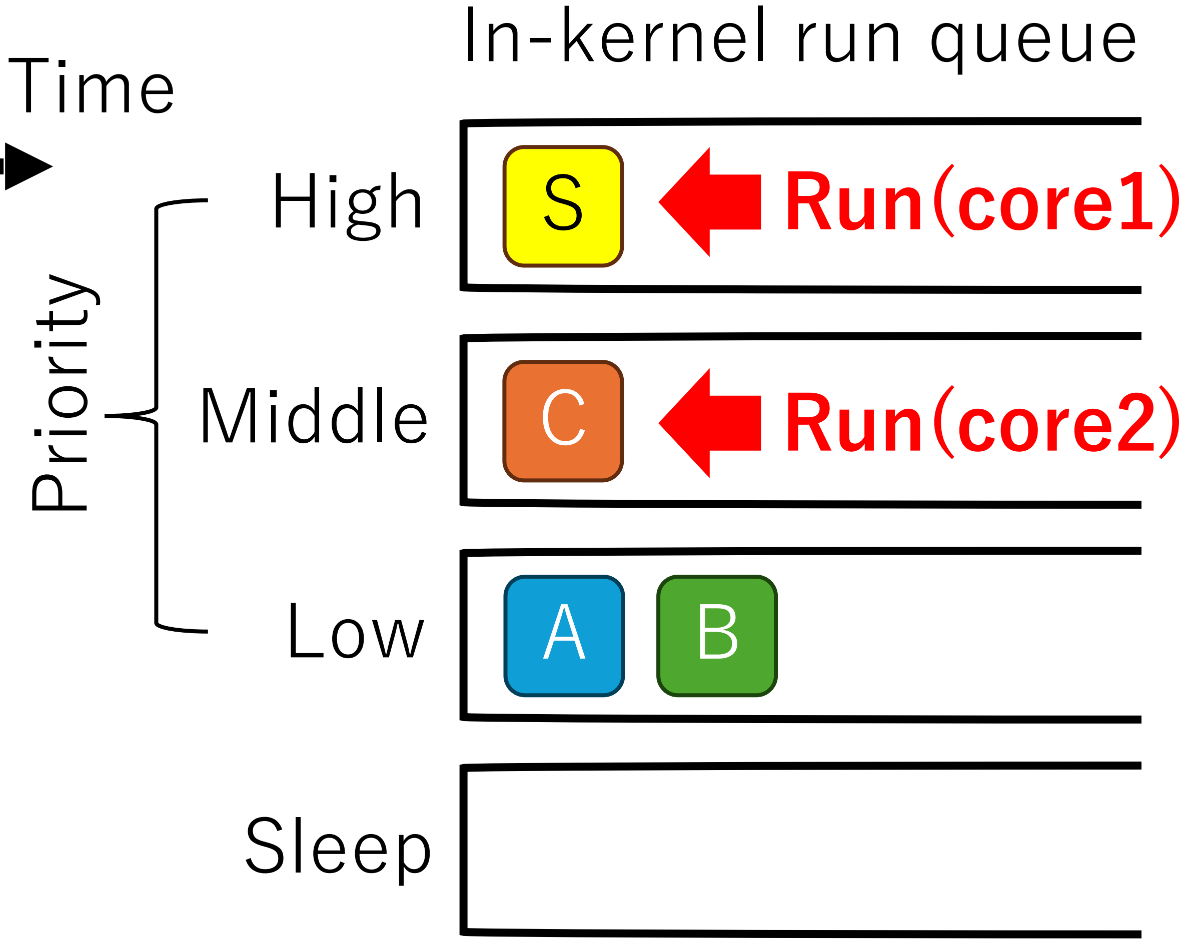
Despite this limitation,
the priority elevation trick has use cases



<u>Limitation</u>

The priority elevation trick is not appropriate for scheduling processes heavily relying on kernel event APIs

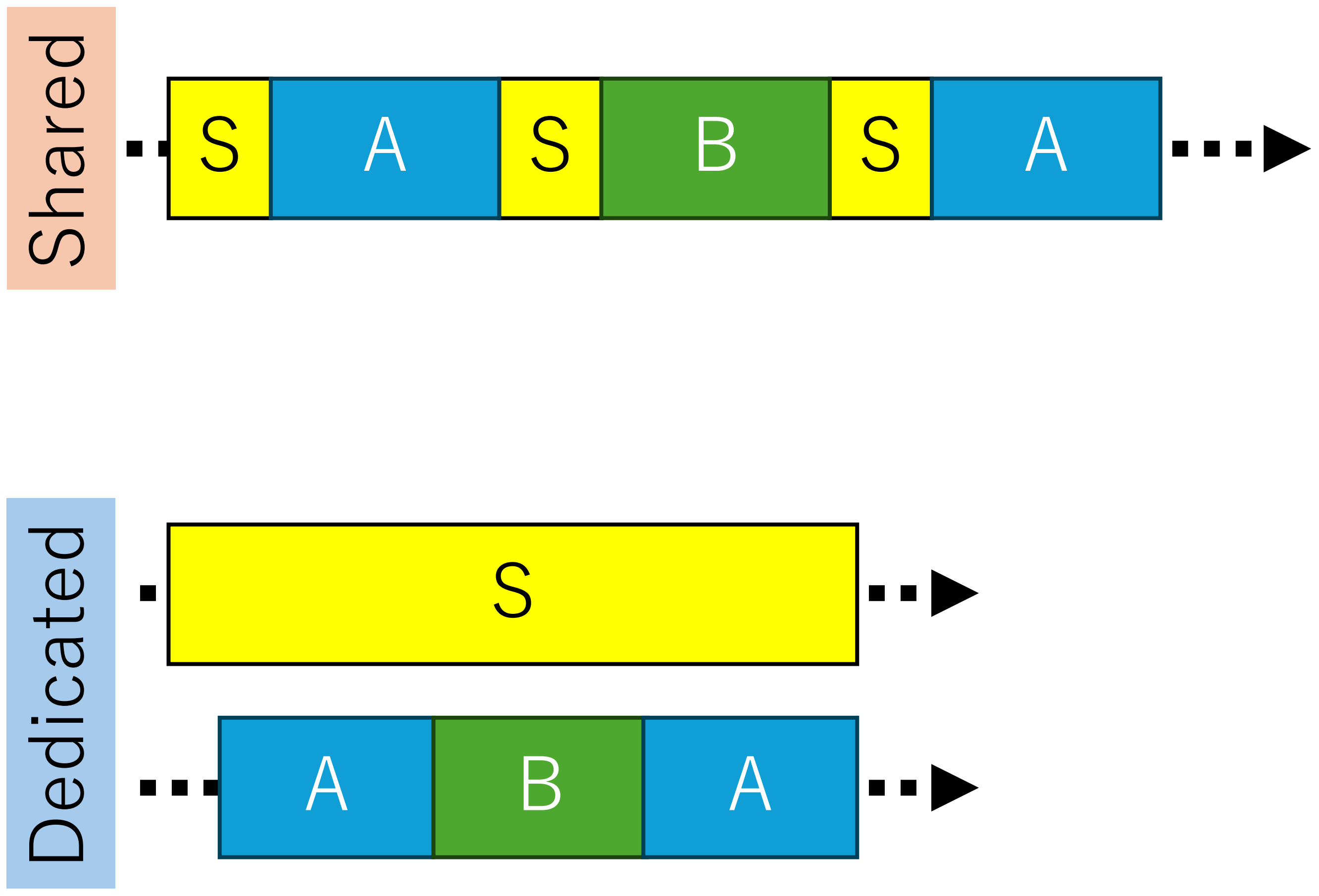




Evaluation

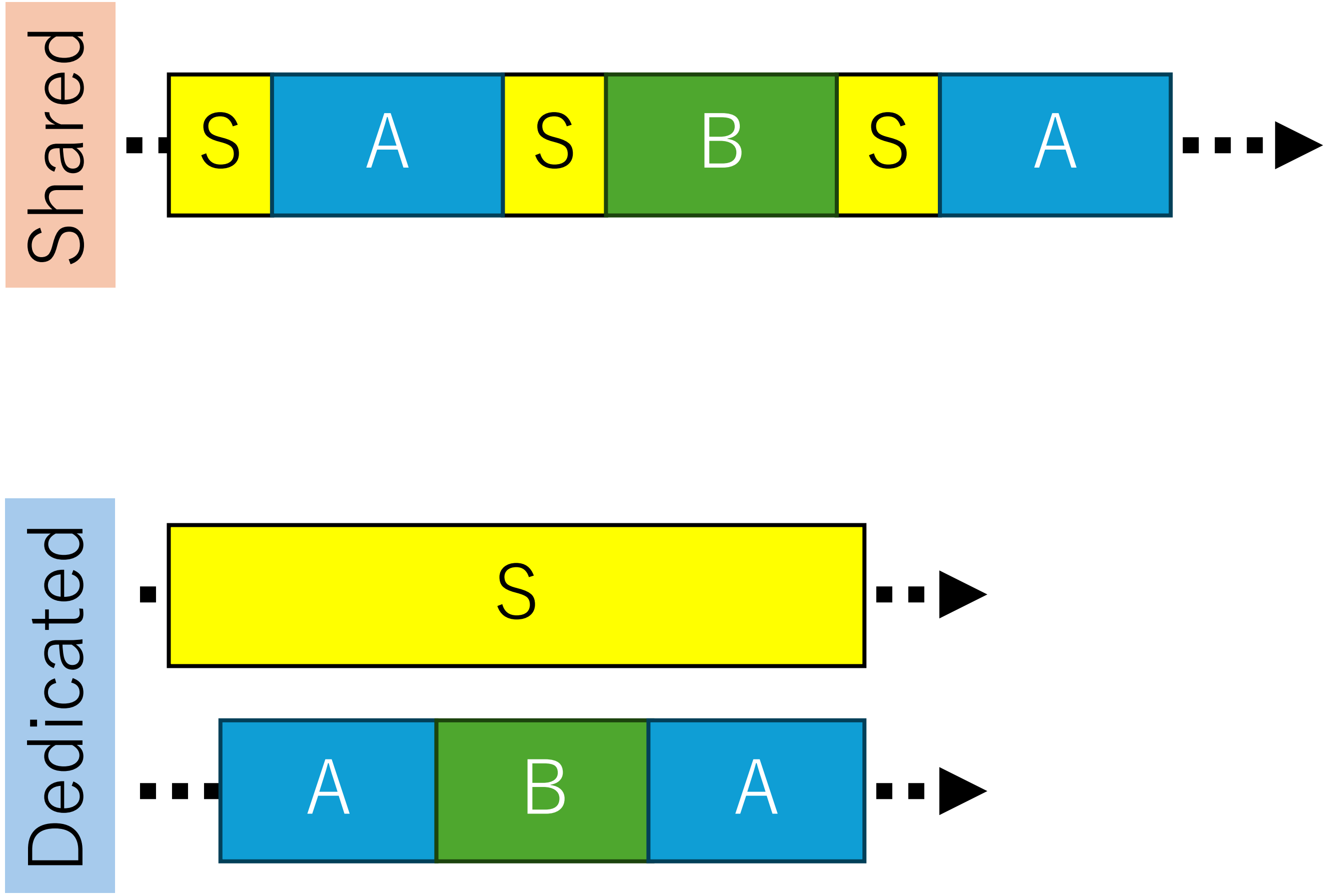
- How much are the overheads?
  - Delay
  - CPU overhead
- What are the use cases?
  - Microsecond-scale time slicing
  - Table-driven scheduling
  - Preemptive scheduling

Evaluation: Delay



Evaluation: Delay

Normal process A and B
run a busy loop

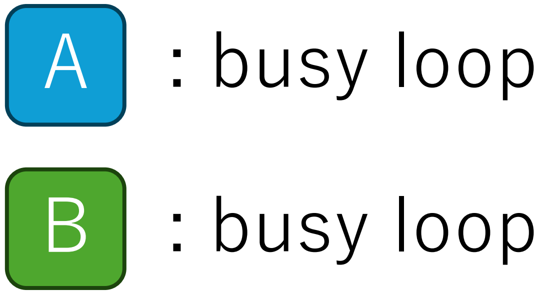


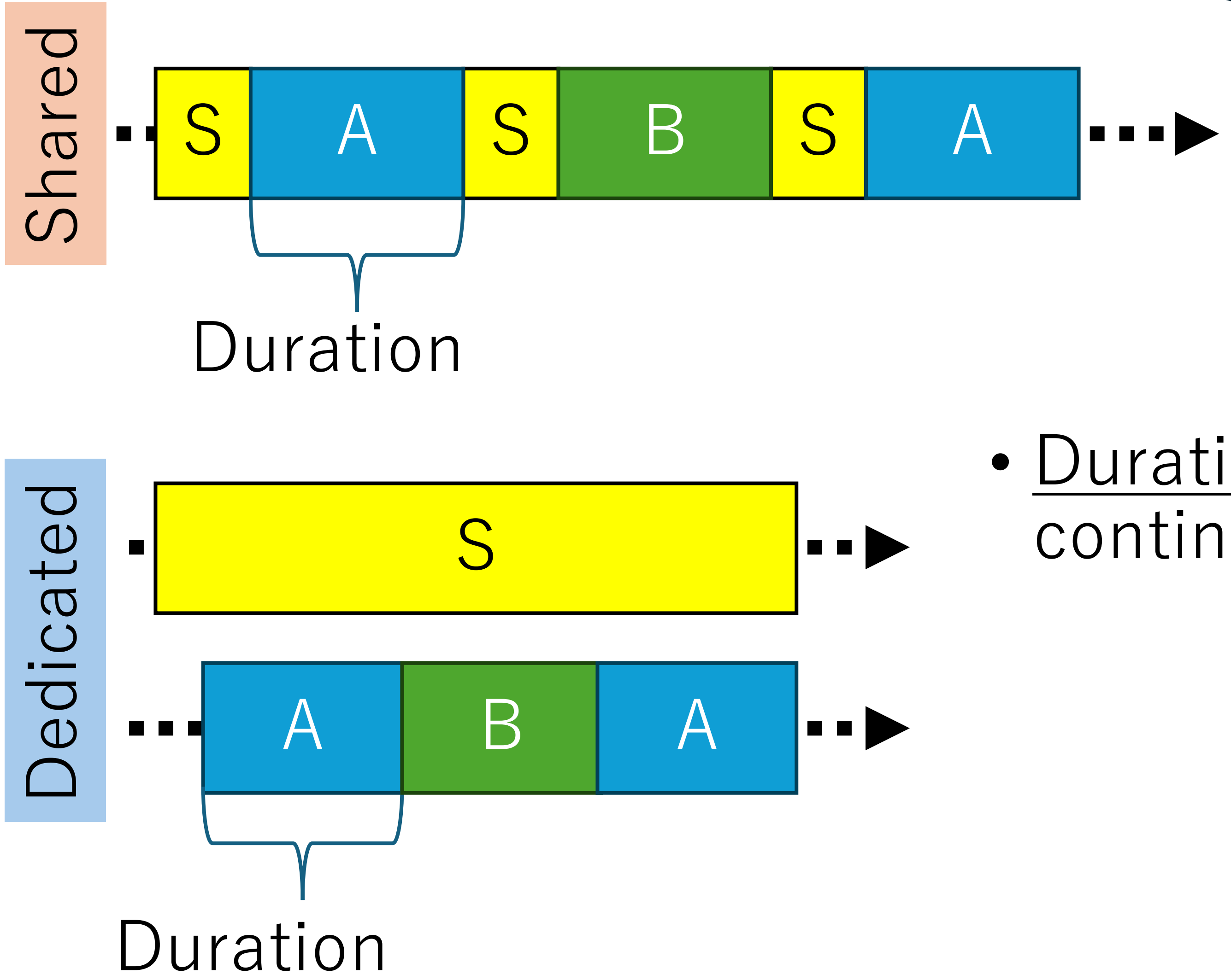
A : busy loop

B : busy loop

Evaluation: Delay

Normal process A and B run a busy loop





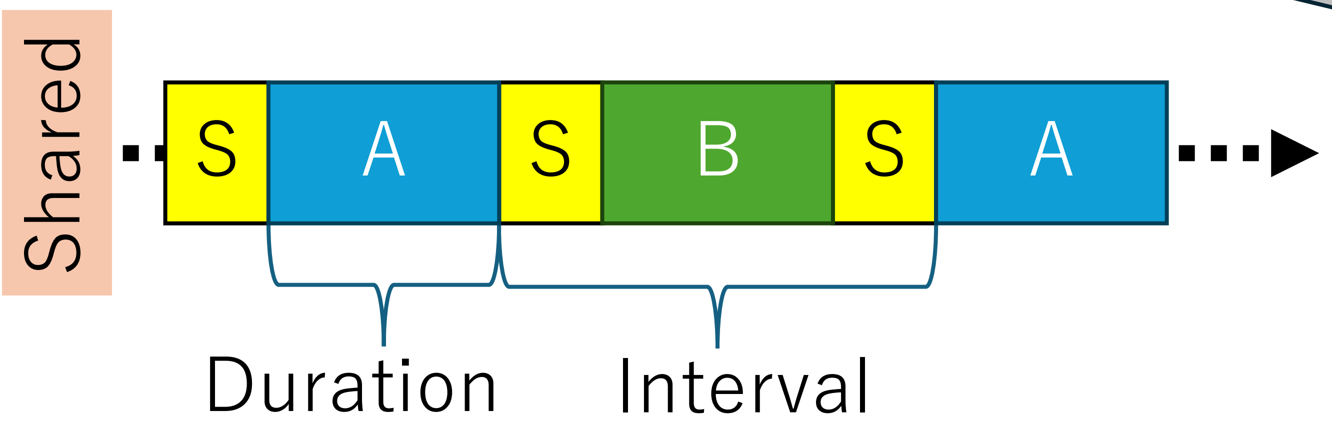
- Duration: time for which a process continues to run once it gets scheduled

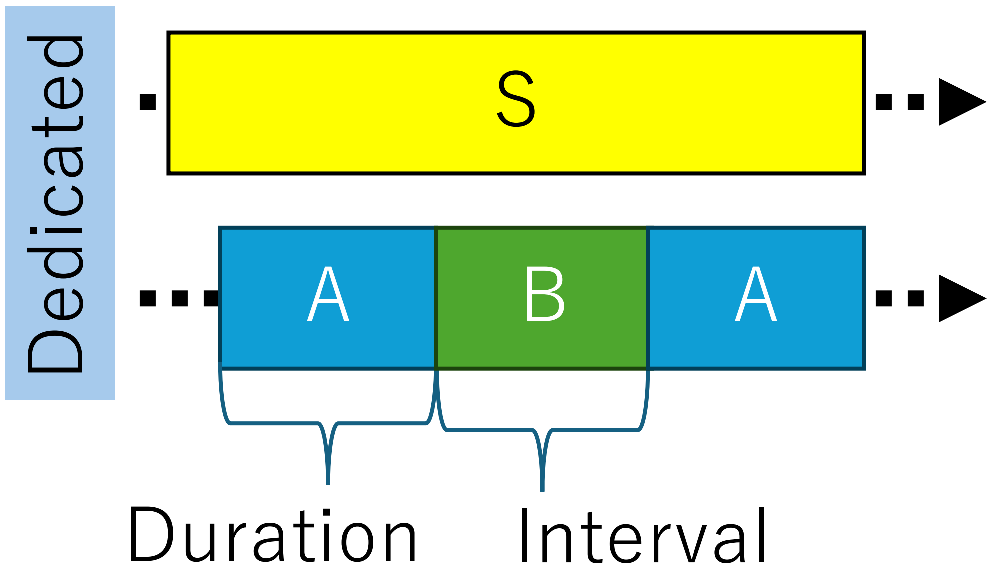
Evaluation: Delay

Normal process A and B run a busy loop

A : busy loop

B : busy loop

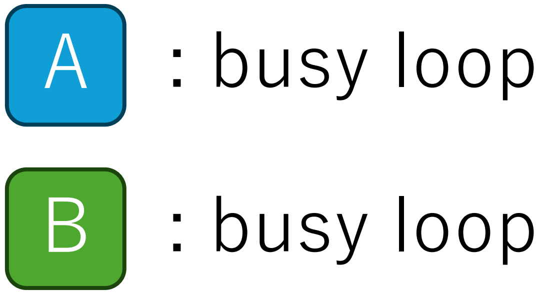


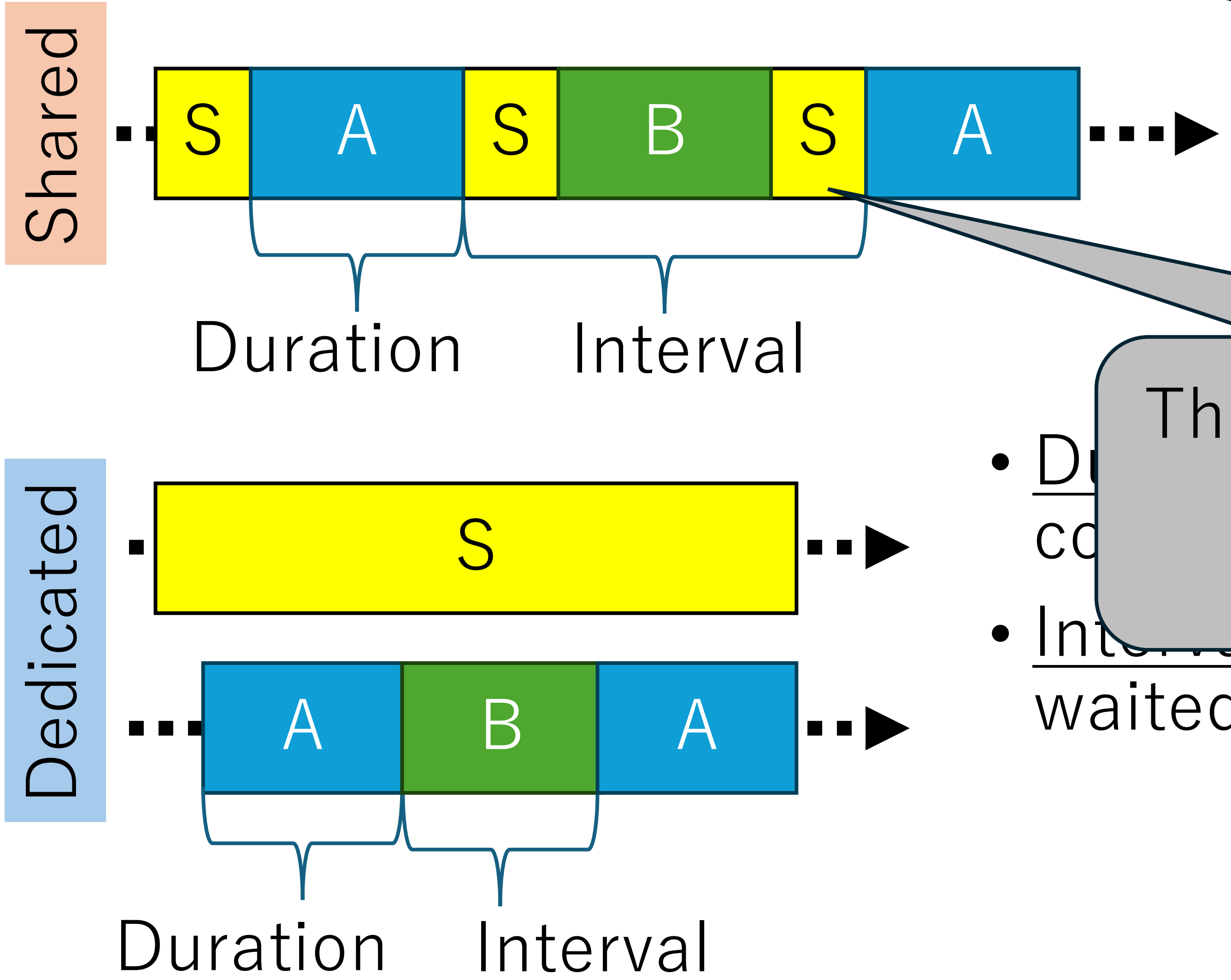


- Duration: time for which a process continues to run once it gets scheduled
- Interval: time a descheduled process waited until it gets scheduled again

Evaluation: Delay

Normal process A and B run a busy loop

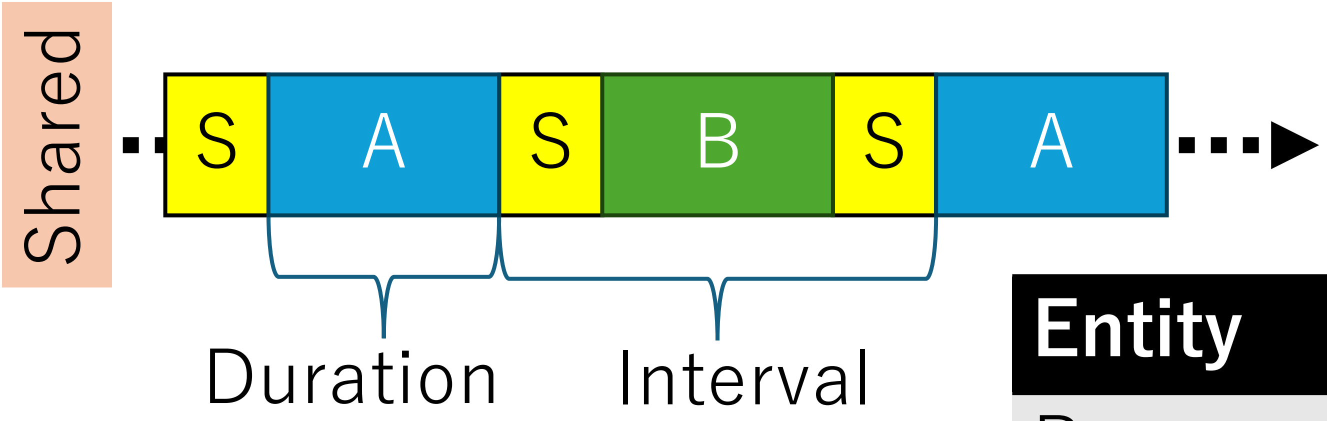


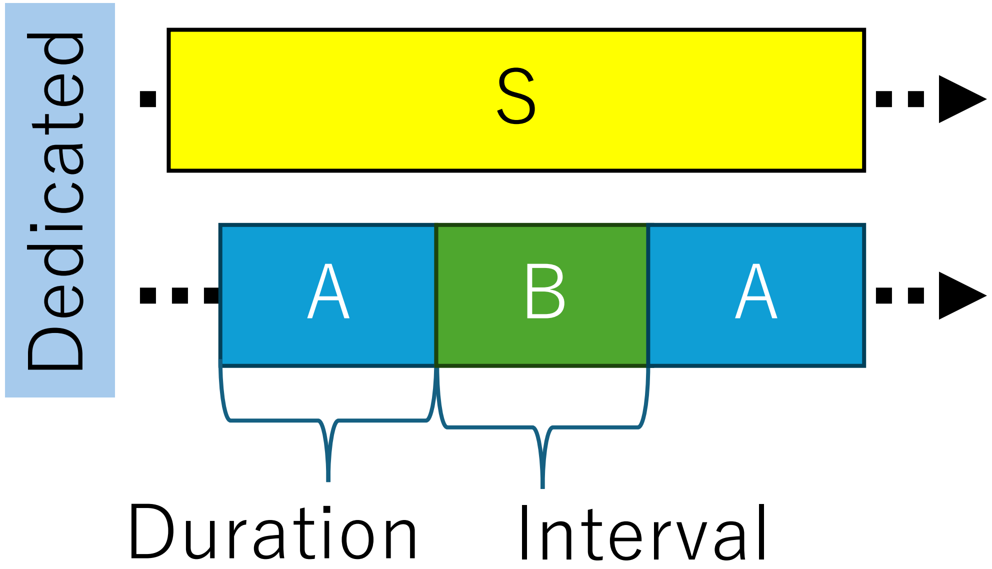


The scheduler process tries to switch two processes every 5 us

- Du… co… …uled
- Int… …ss waited until it gets scheduled again

Evaluation: Delay

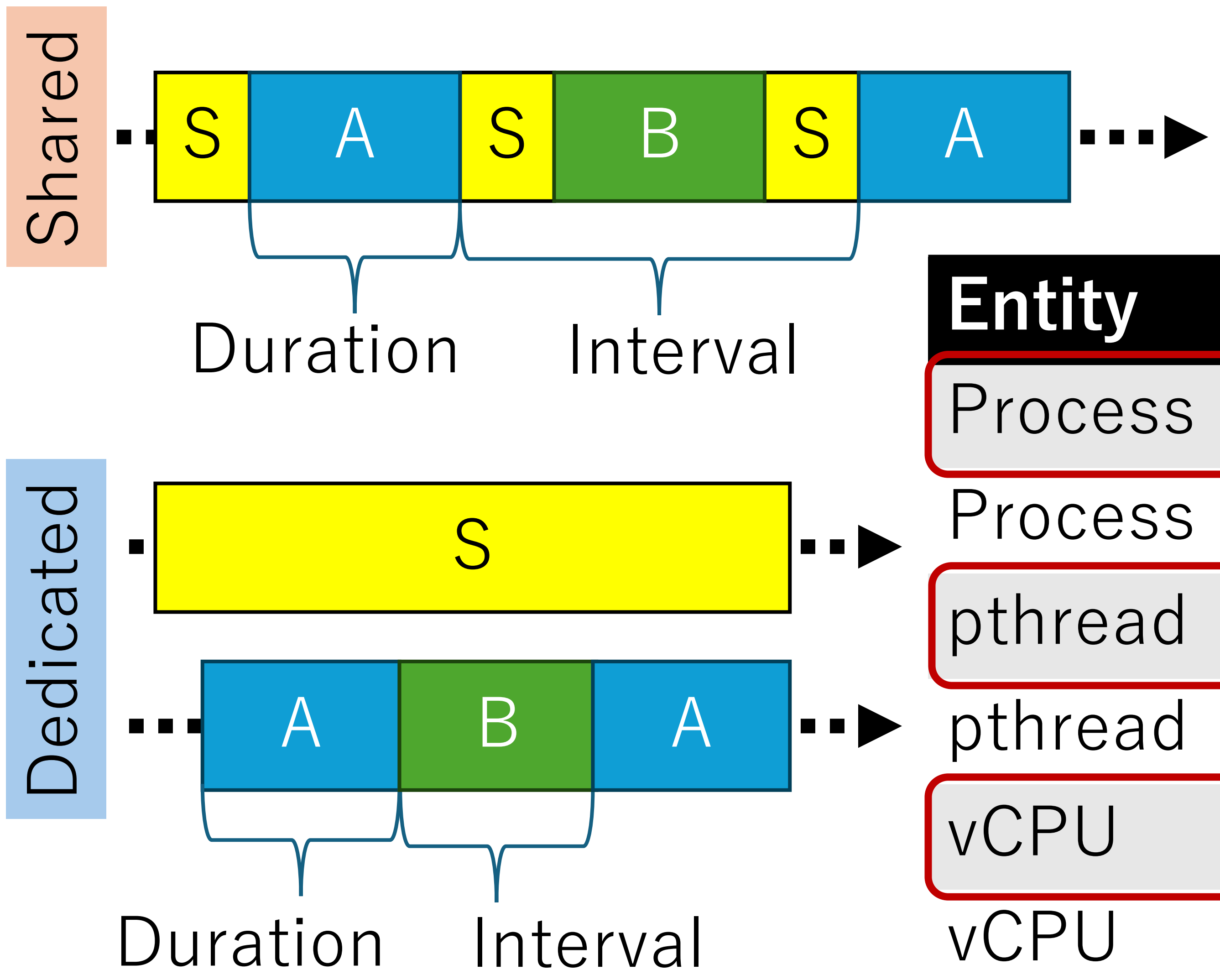




Entity	Pattern	Duration	Interval
Process	Shared	5.5 us	9.6 us
Process	Dedicated	5.0 us	5.0 us
pthread	Shared	5.5 us	9.2 us
pthread	Dedicated	5.0 us	5.0 us
vCPU	Shared	7.2 us	14.4 us
vCPU	Dedicated	5.0 us	5.0 us

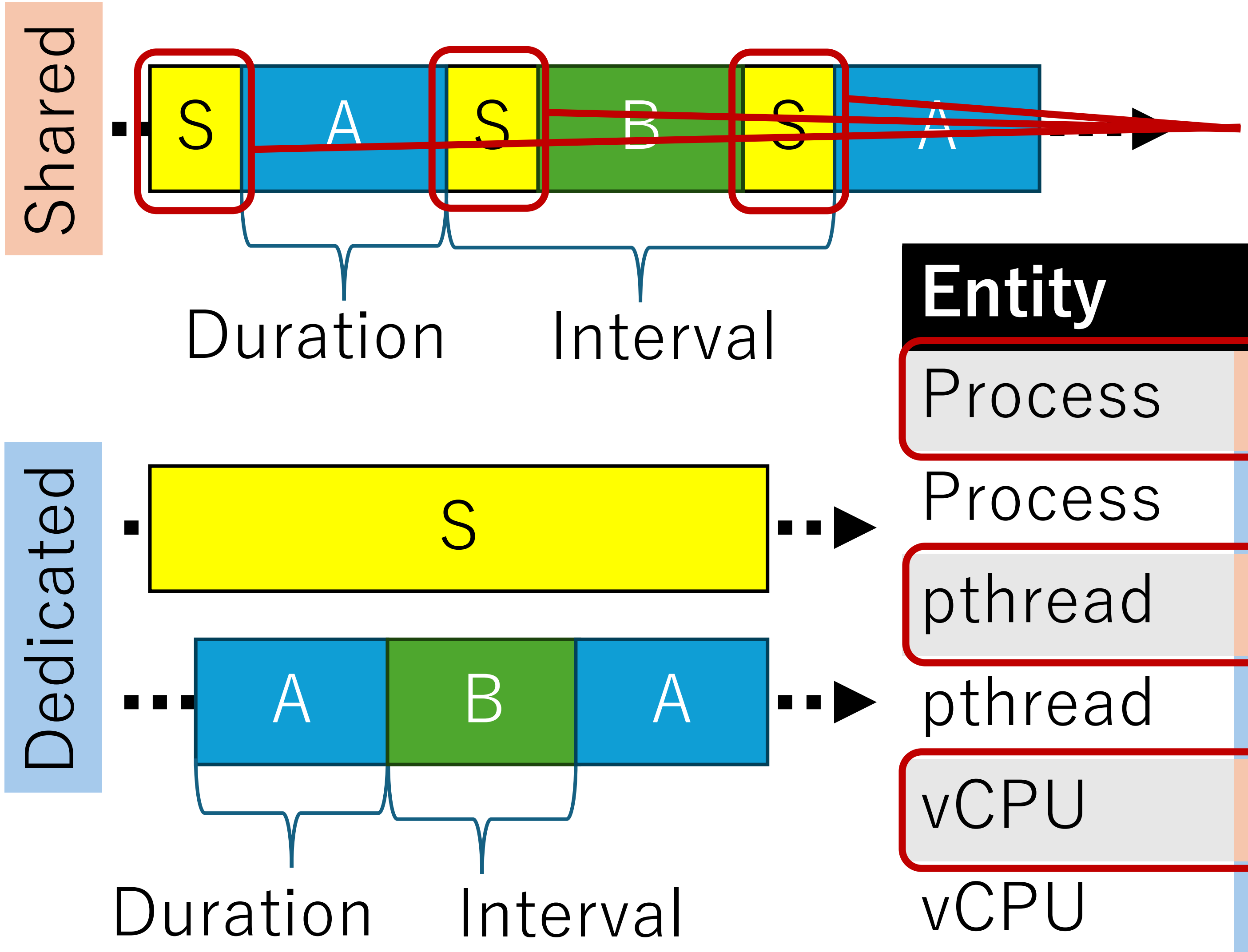
Evaluation: Delay

In the shared cases, delays are added



Entity	Pattern	Duration	Interval
Process	Shared	5.5 us	9.6 us
Process	Dedicated	5.0 us	5.0 us
pthread	Shared	5.5 us	9.2 us
pthread	Dedicated	5.0 us	5.0 us
vCPU	Shared	7.2 us	14.4 us
vCPU	Dedicated	5.0 us	5.0 us

Evaluation: Delay

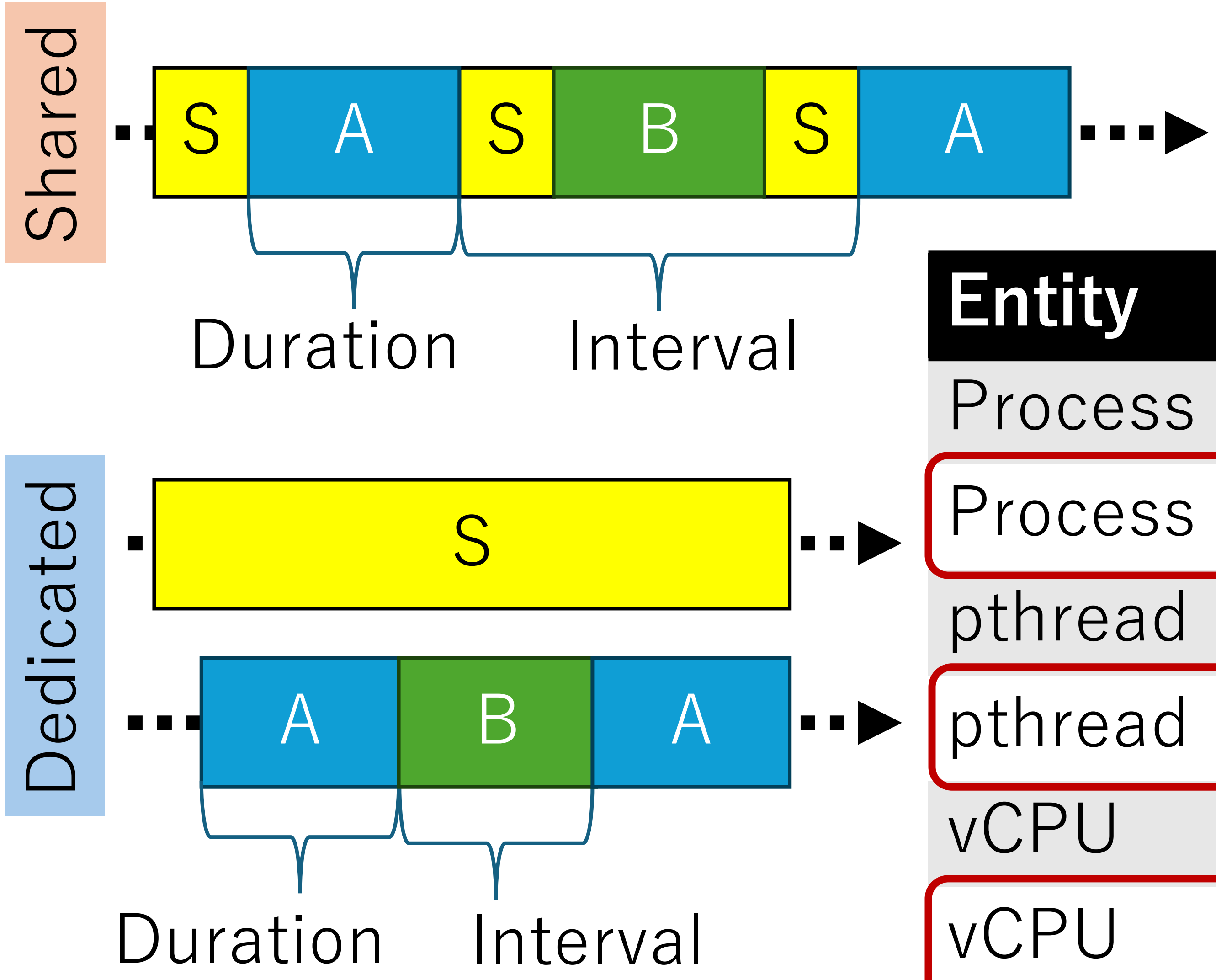


In the shared cases, delays are added

Delays are coming from the scheduler process

Entity	Pattern	Duration	Interval
Process	Shared	5.5 us	9.6 us
Process	Dedicated	5.0 us	5.0 us
pthread	Shared	5.5 us	9.2 us
pthread	Dedicated	5.0 us	5.0 us
vCPU	Shared	7.2 us	14.4 us
vCPU	Dedicated	5.0 us	5.0 us

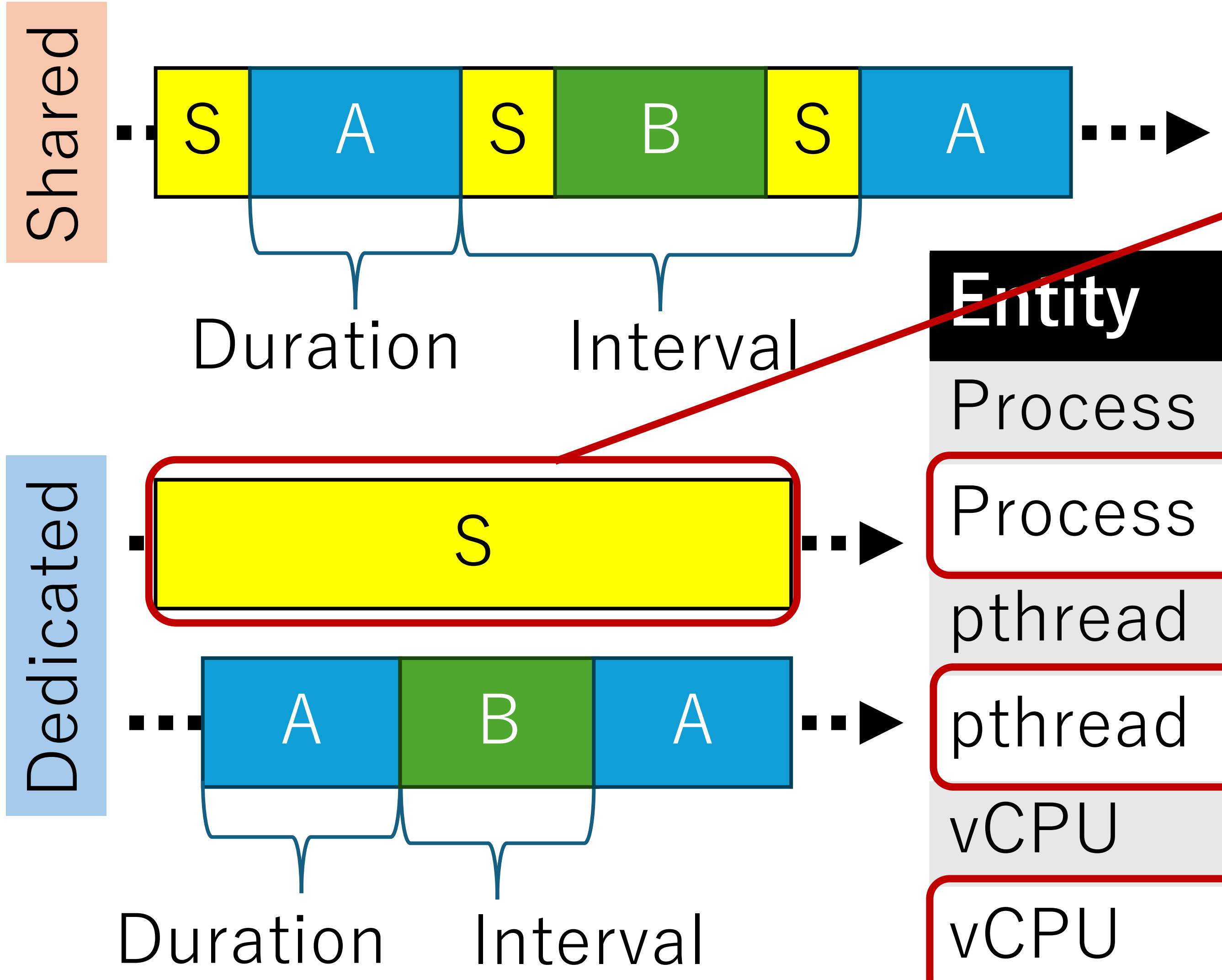
Evaluation: Delay



In the dedicated cases,
A and B are switched every 5 us
as intended

Entity	Pattern	Duration	Interval
Process	Shared	5.5 us	9.6 us
Process	Dedicated	5.0 us	5.0 us
pthread	Shared	5.5 us	9.2 us
pthread	Dedicated	5.0 us	5.0 us
vCPU	Shared	7.2 us	14.4 us
vCPU	Dedicated	5.0 us	5.0 us

Evaluation: Delay

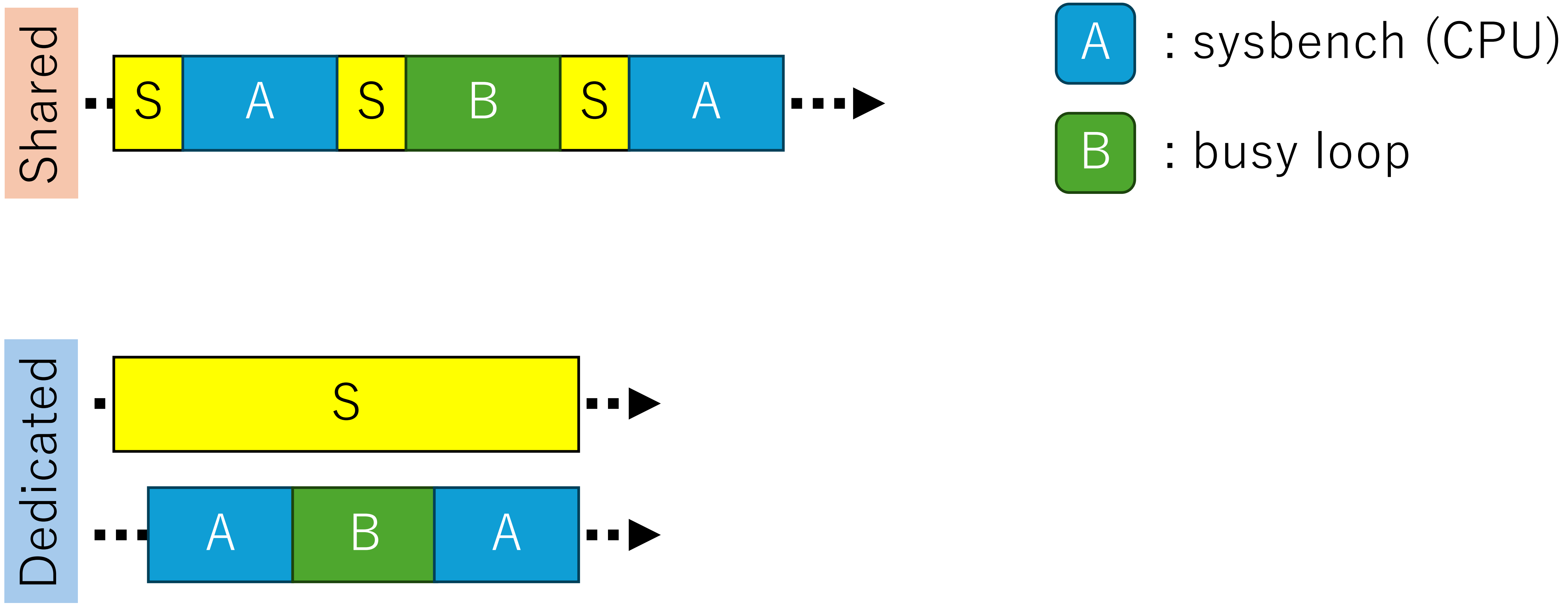


In the dedicated cases,
A and B are switched every 5 us

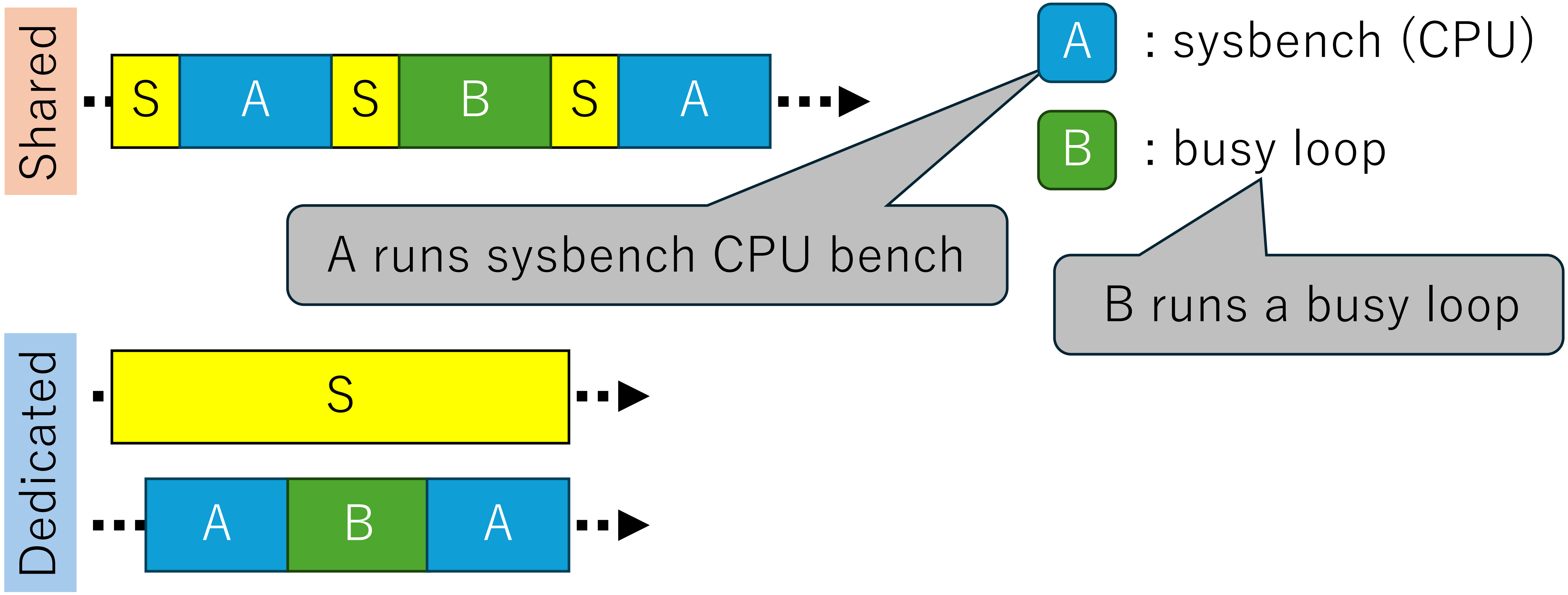
But, the dedicate case uses
one additional CPU core

Entity	Pattern	Duration	Interval
Process	Shared	5.5 us	9.6 us
Process	Dedicated	5.0 us	5.0 us
pthread	Shared	5.5 us	9.2 us
pthread	Dedicated	5.0 us	5.0 us
vCPU	Shared	7.2 us	14.4 us
vCPU	Dedicated	5.0 us	5.0 us

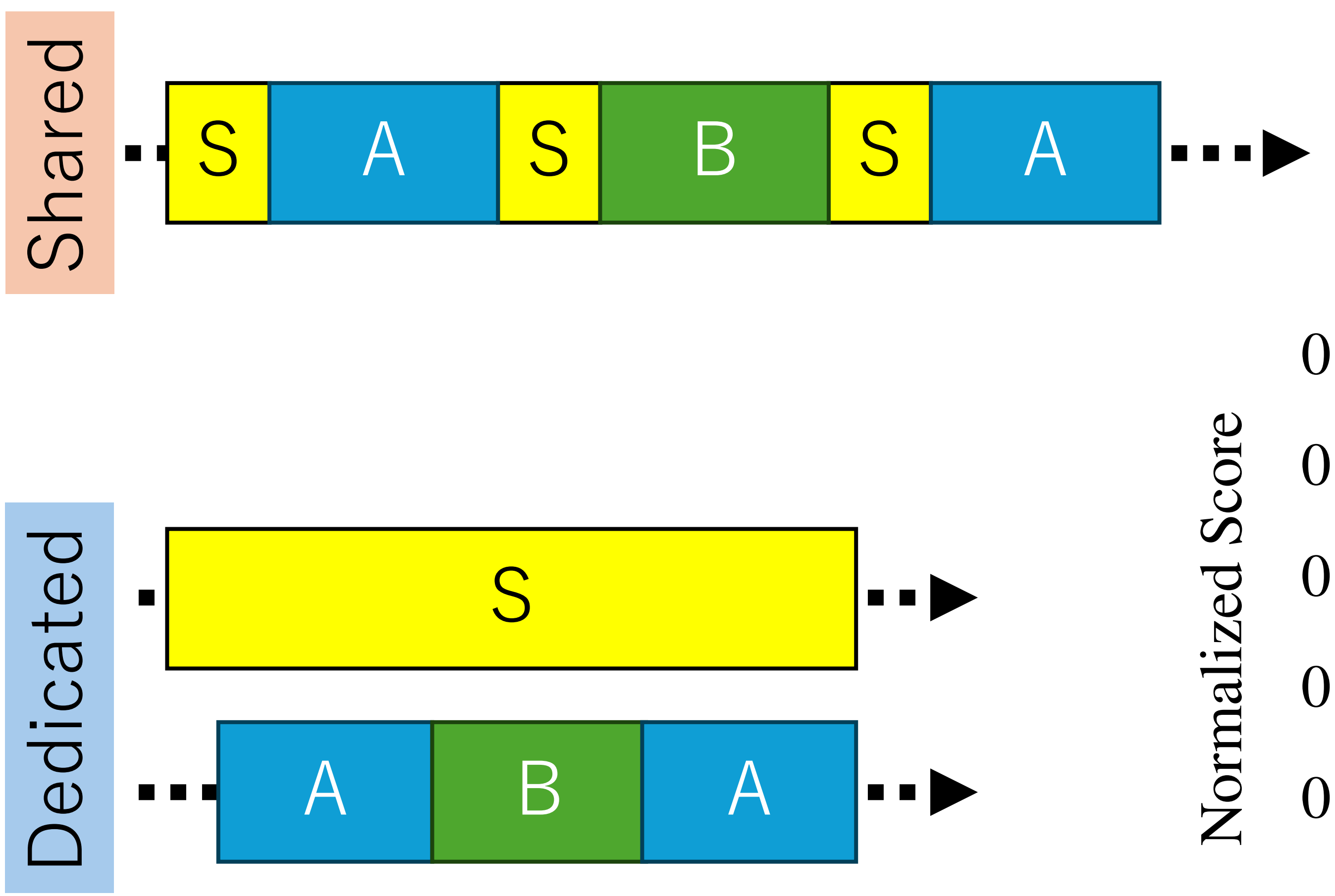
Evaluation: CPU Overhead

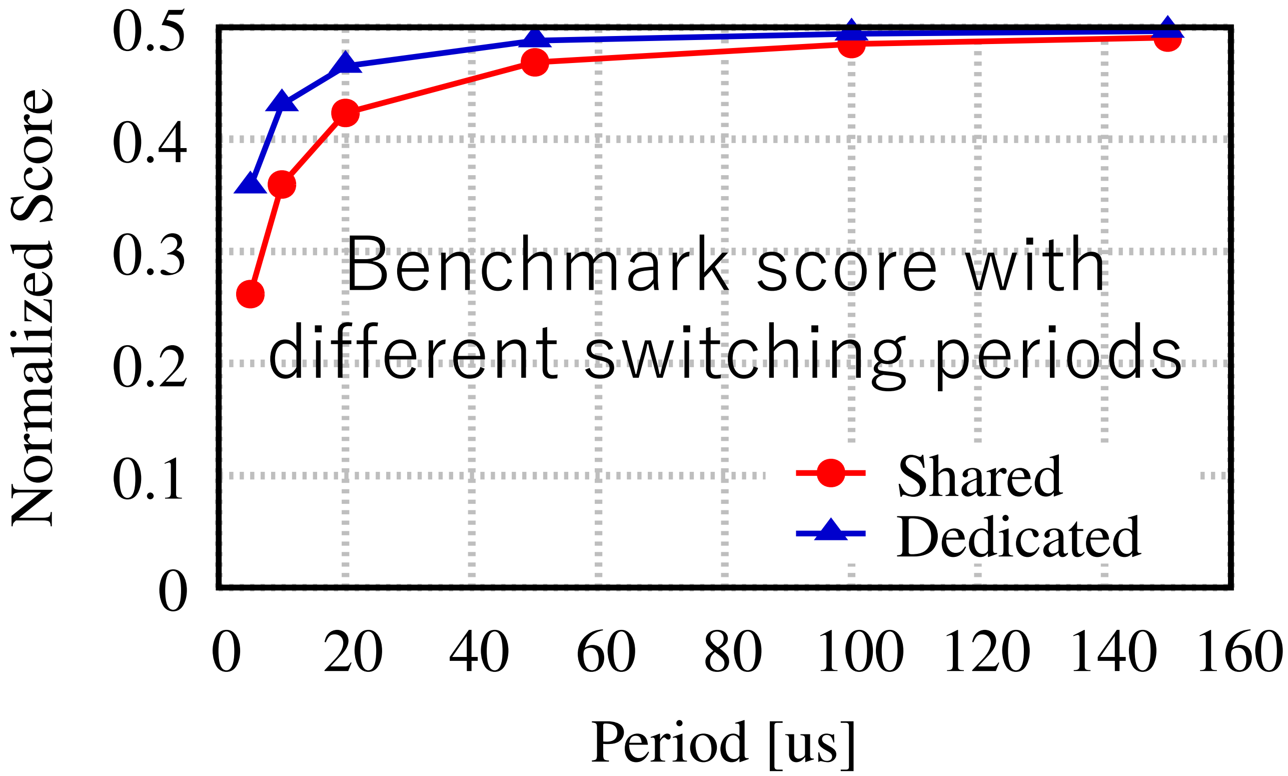


Evaluation: CPU Overhead



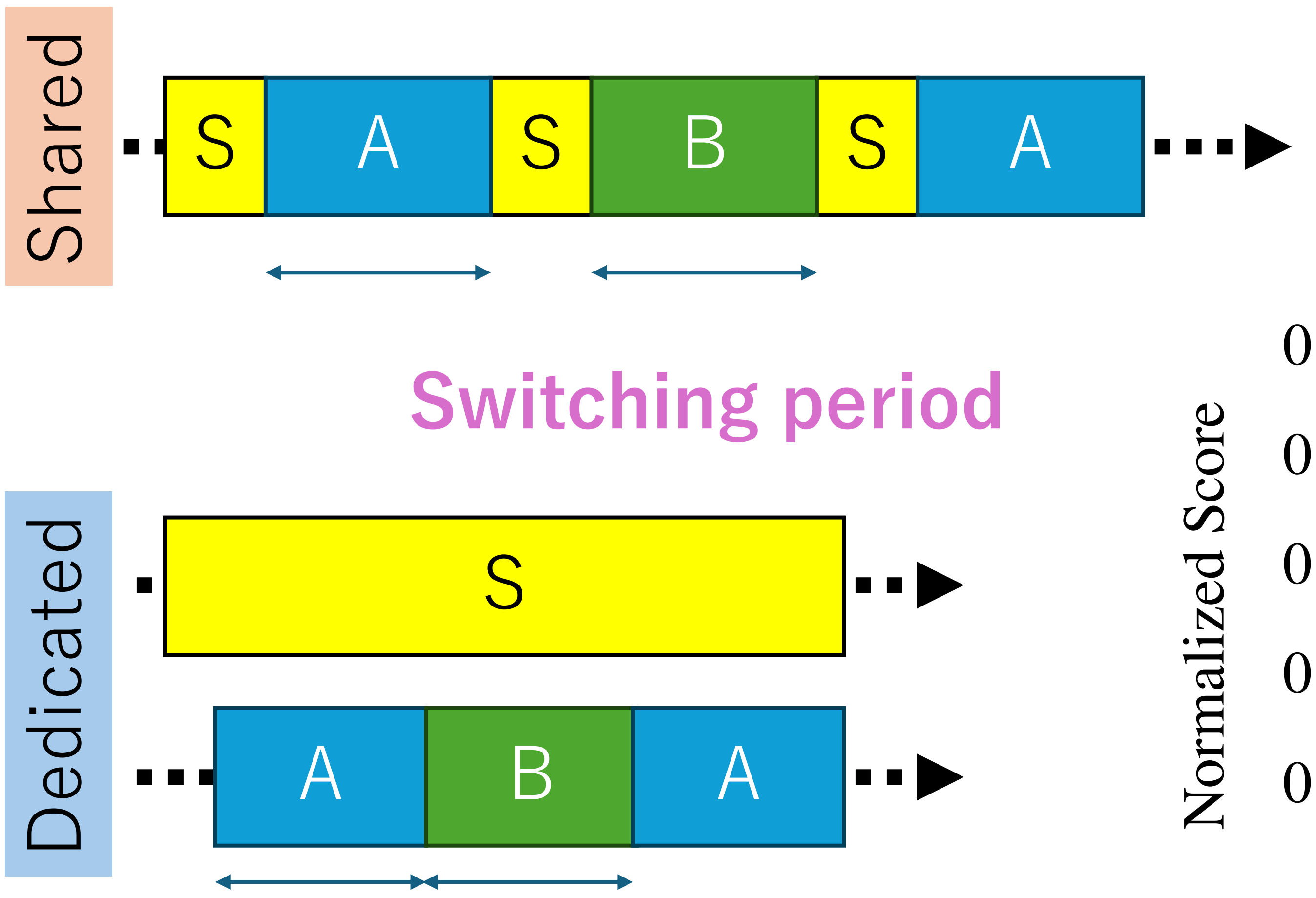
Evaluation: CPU Overhead





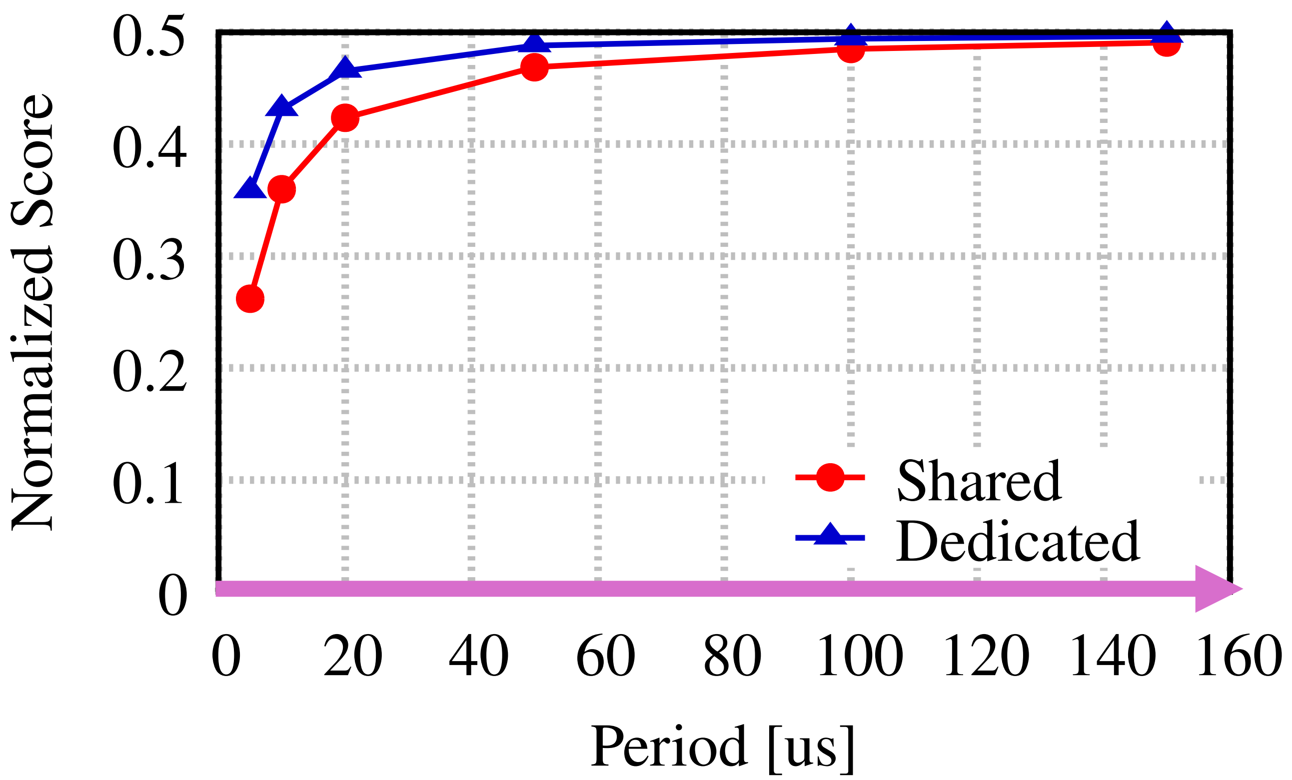
Benchmark score with different switching periods

Evaluation: CPU Overhead

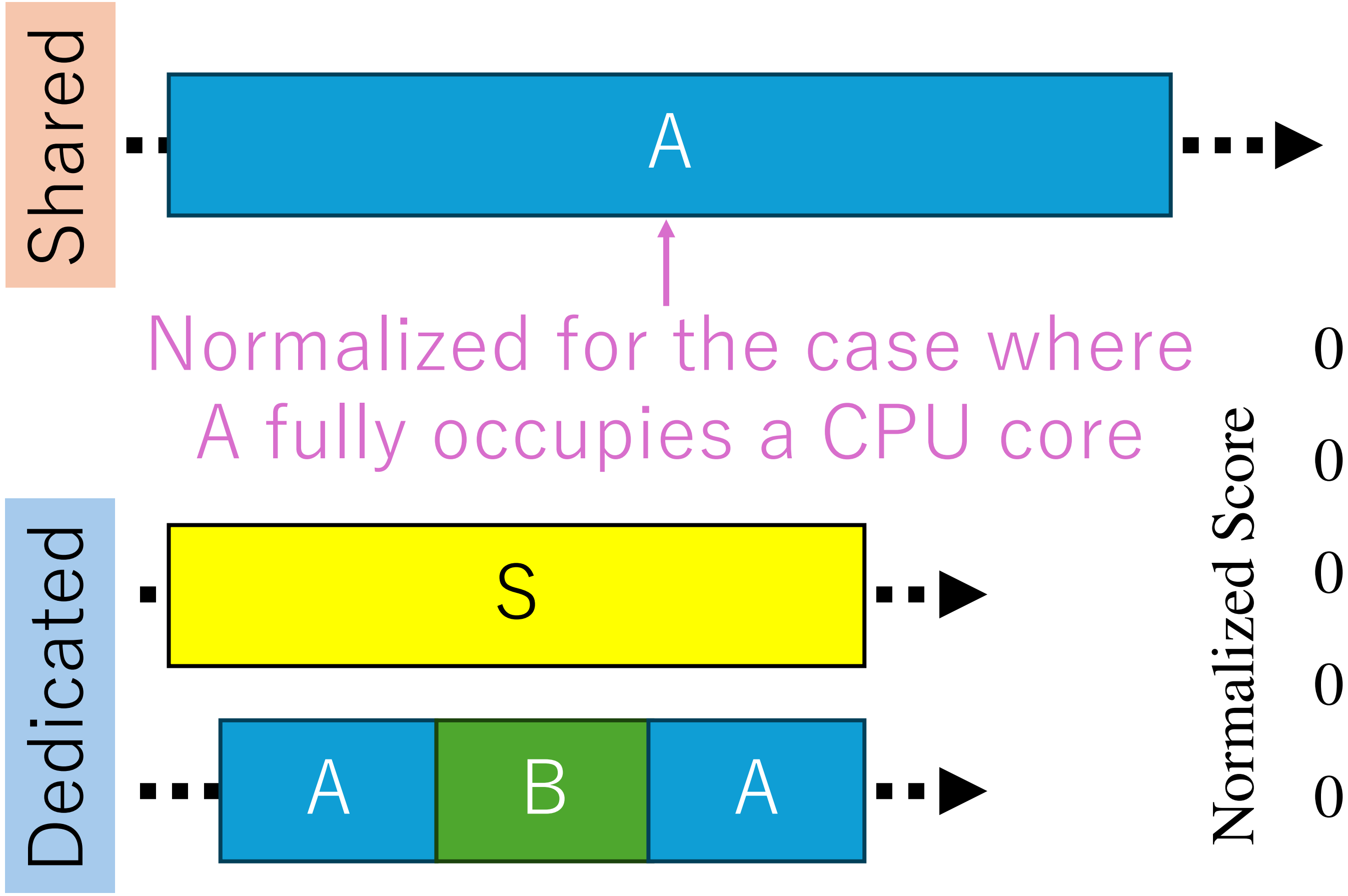


A : sysbench (CPU)

B : busy loop

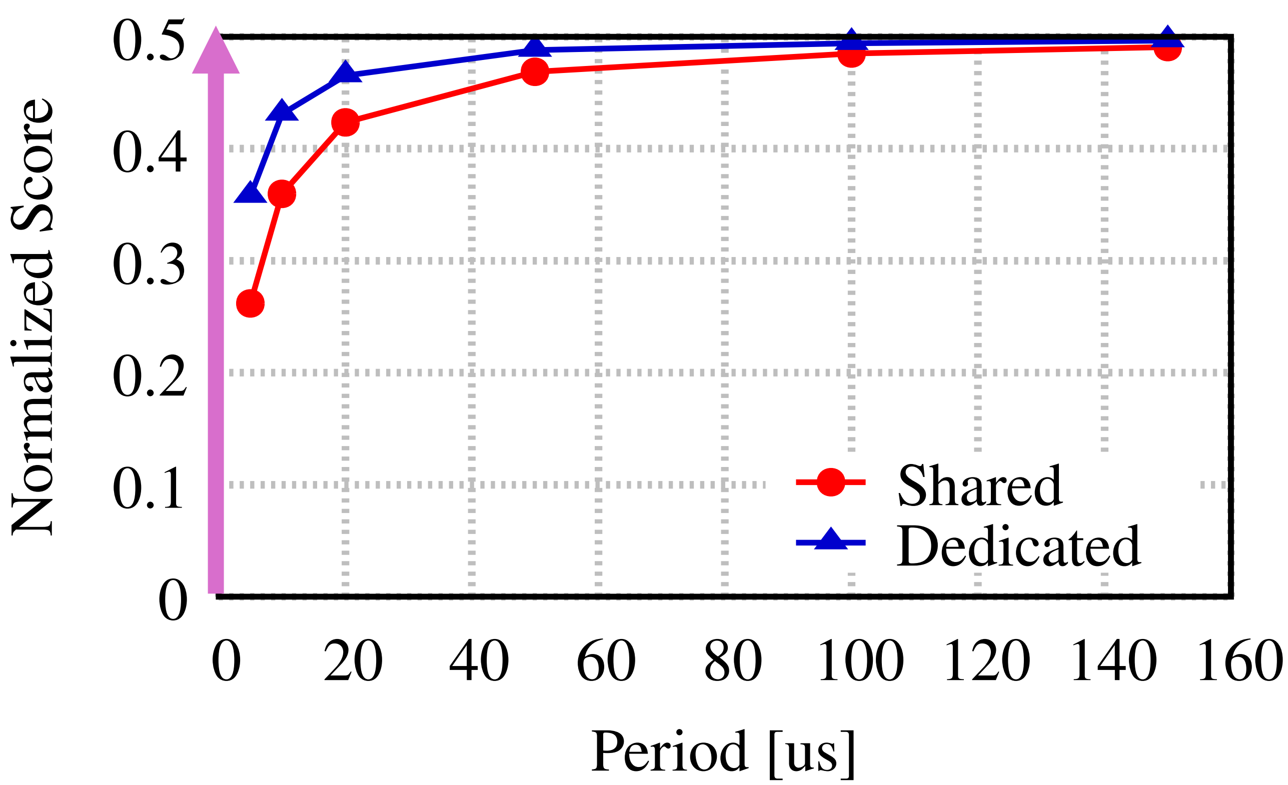


Evaluation: CPU Overhead

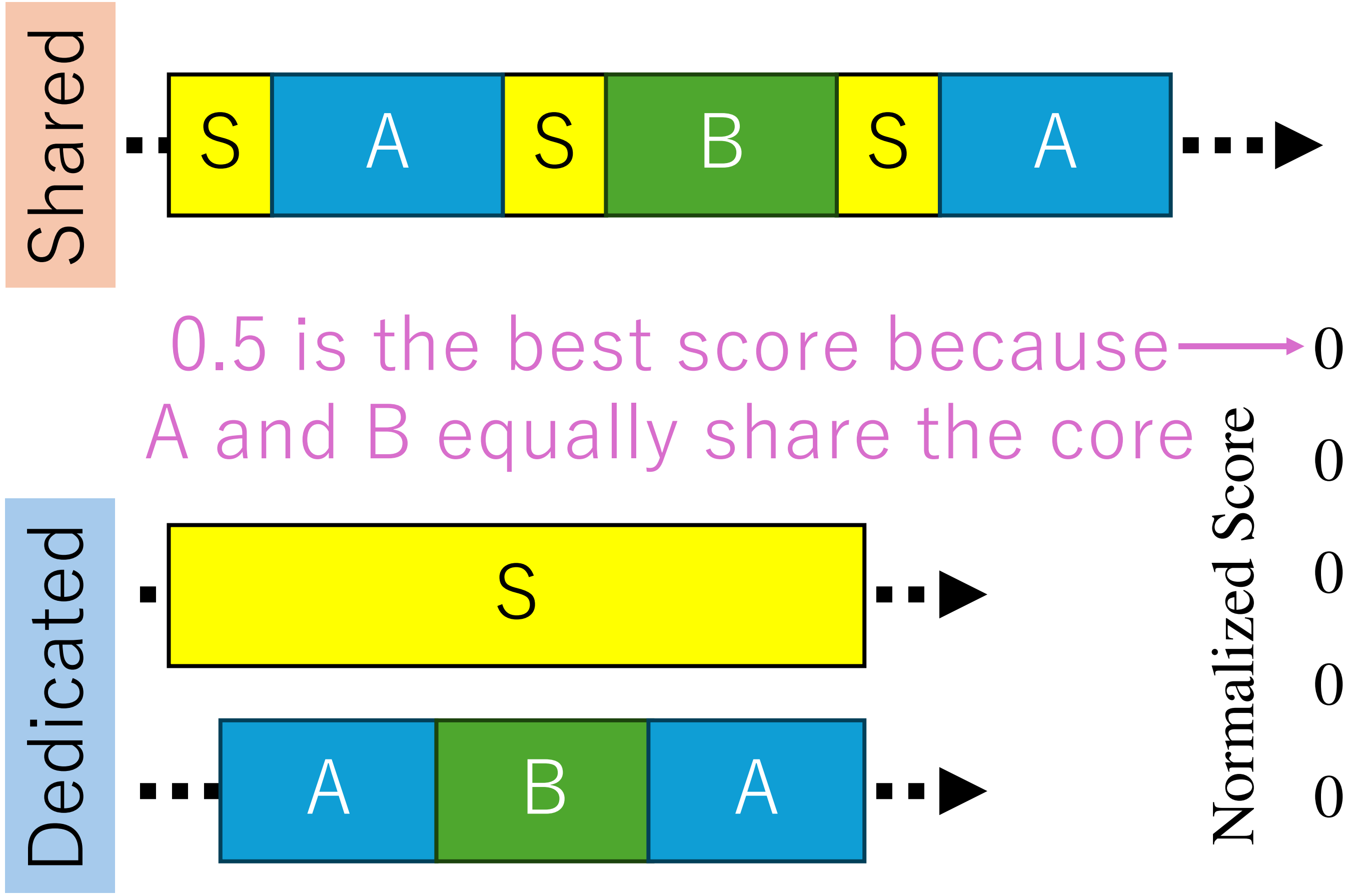


A : sysbench (CPU)

B : busy loop



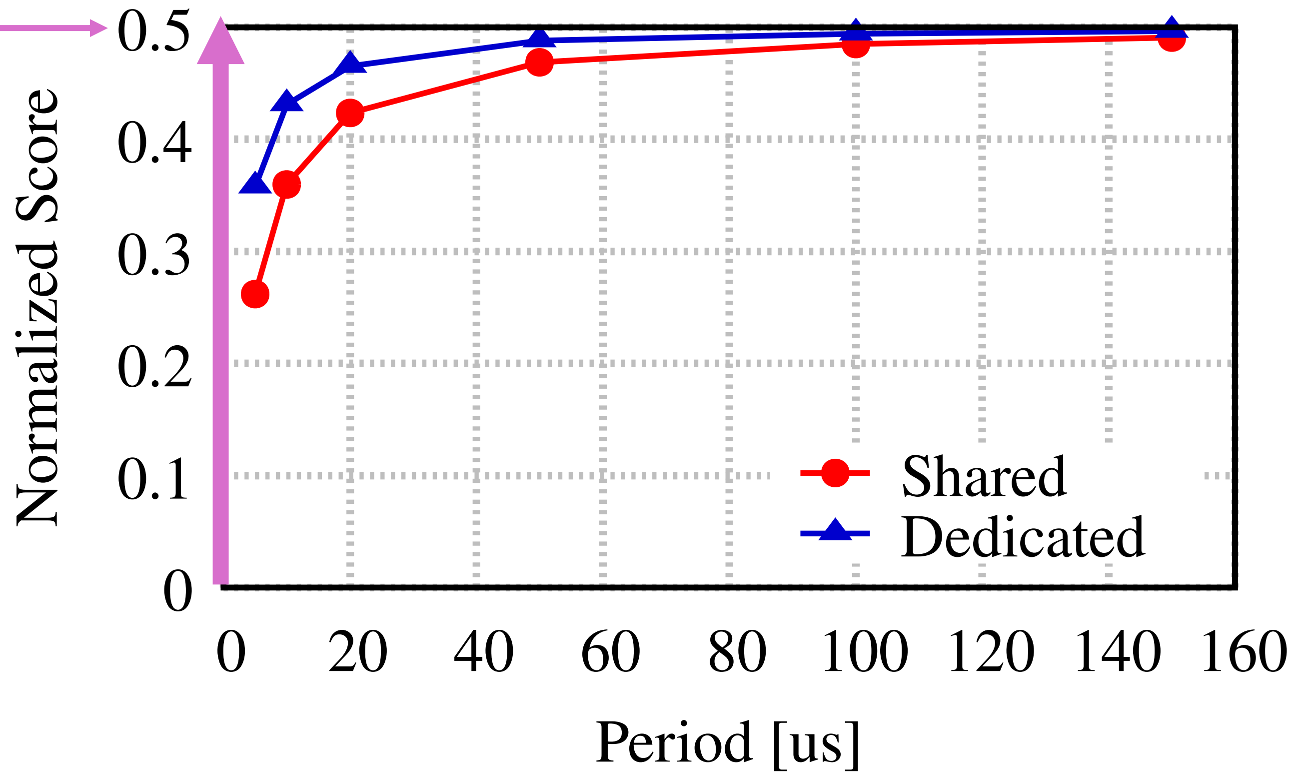
Evaluation: CPU Overhead



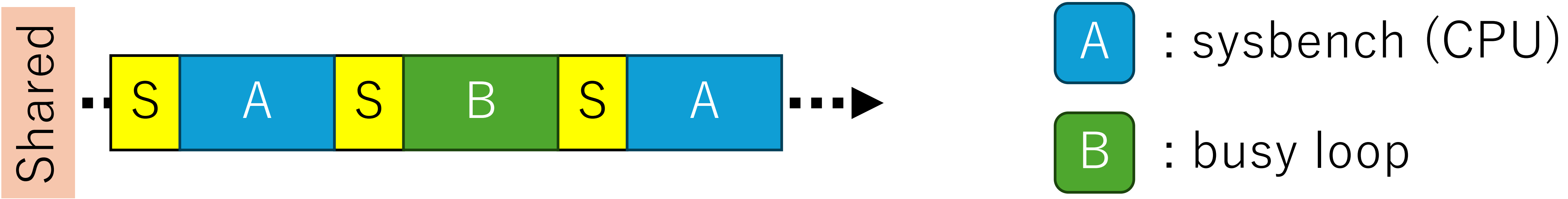
A : sysbench (CPU)

B : busy loop

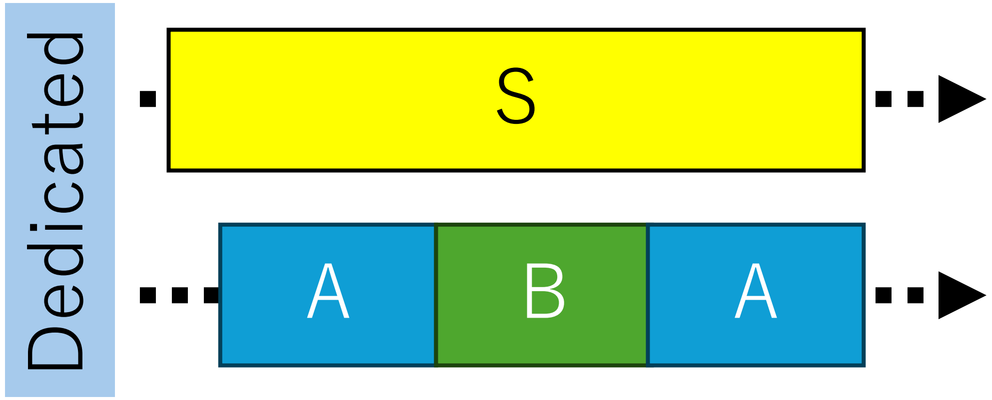
0.5 is the best score because
A and B equally share the core

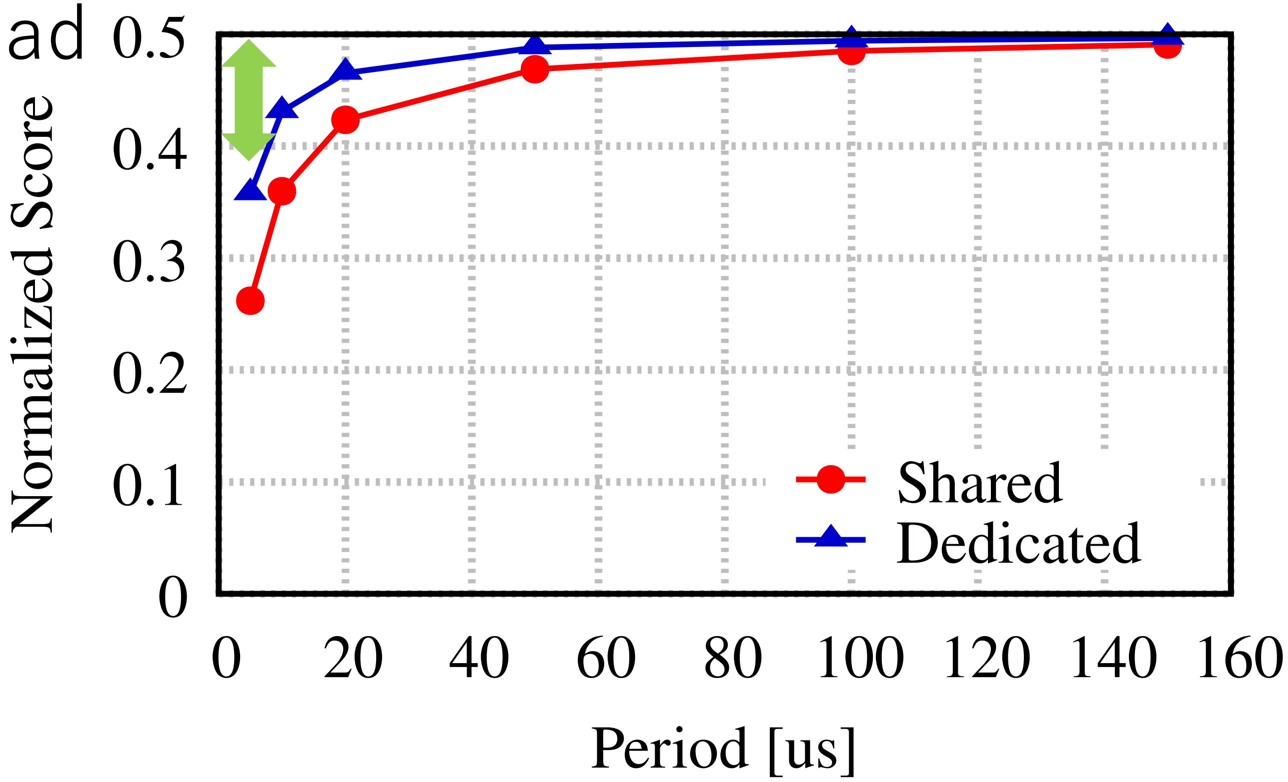


Evaluation: CPU Overhead

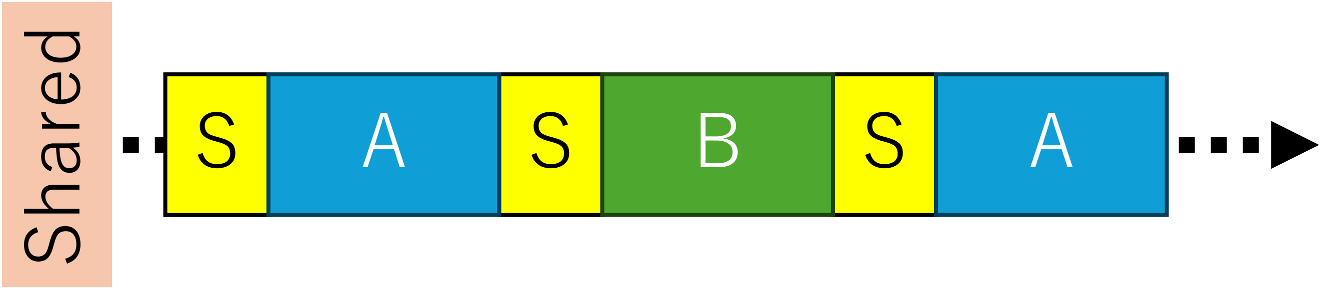


Gap to 0.5 shows the CPU overhead coming from the scheduling





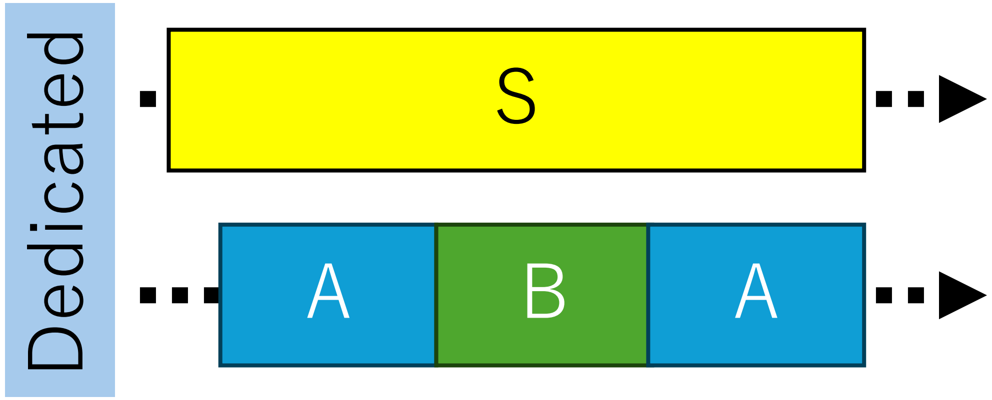
Evaluation: CPU Overhead

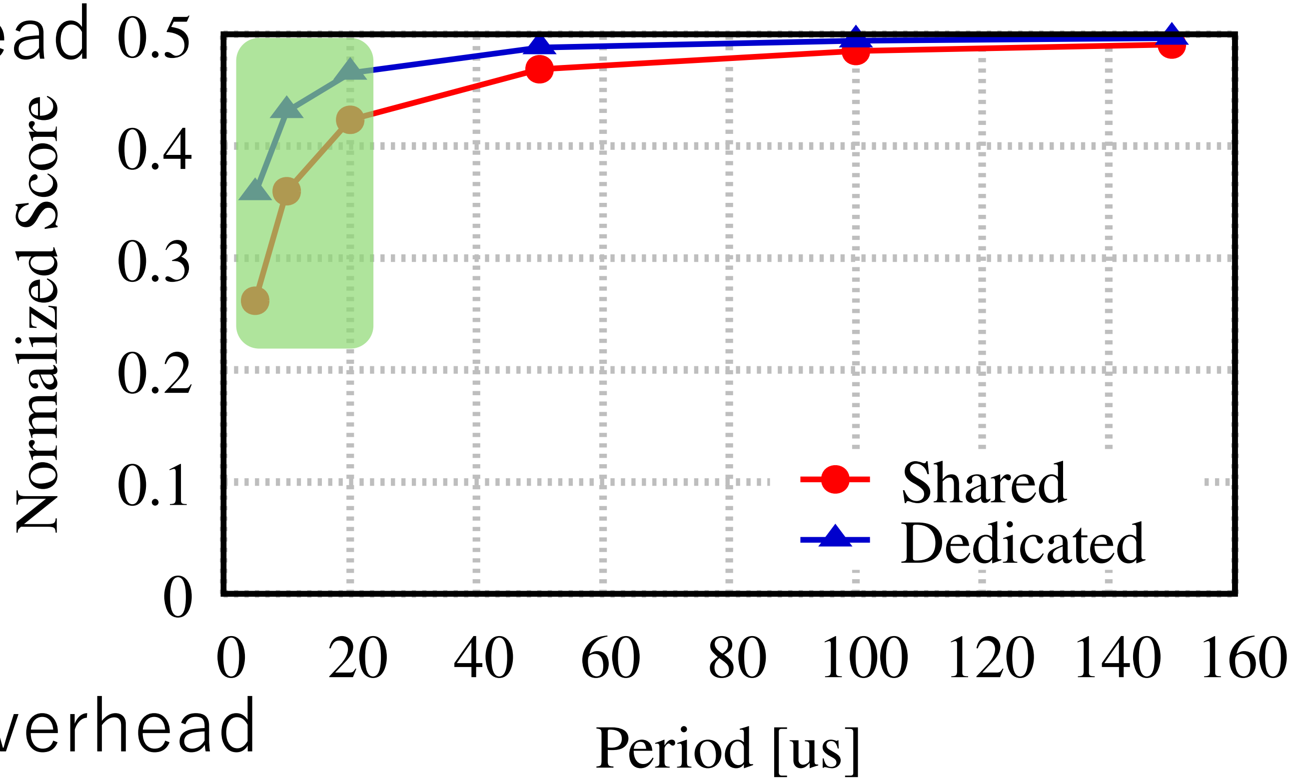


A : sysbench (CPU)

B : busy loop

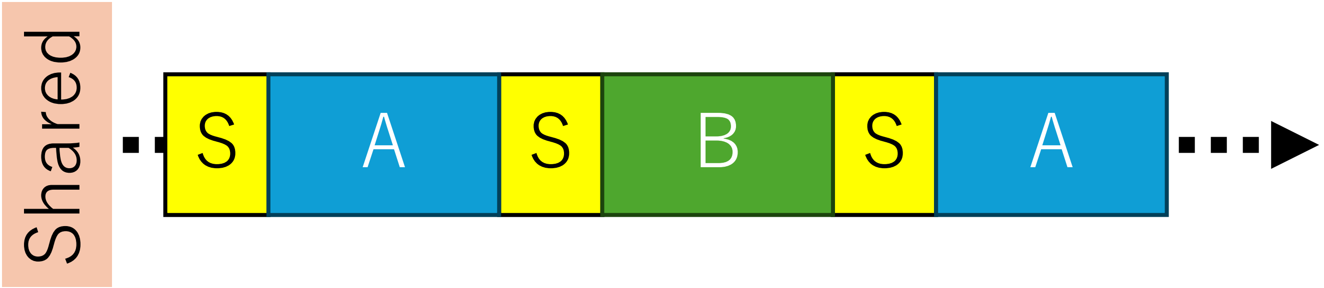
Gap to 0.5 shows the CPU overhead
coming from the scheduling





A shorter period leads to higher overhead

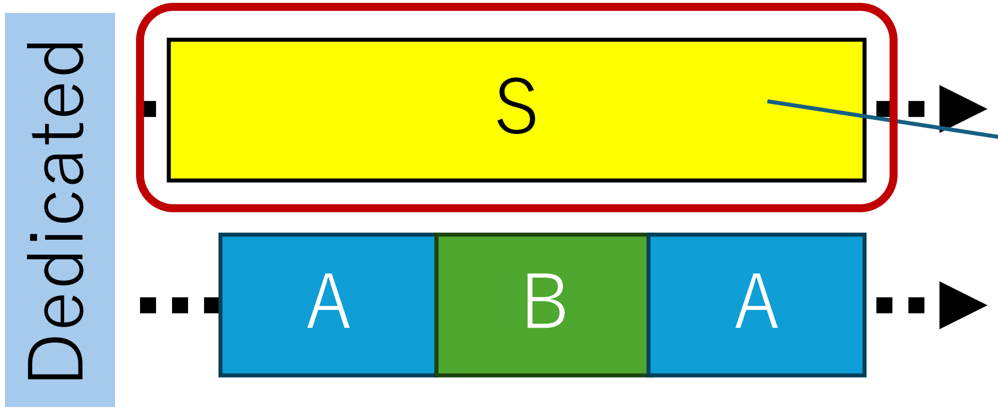
Evaluation: CPU Overhead

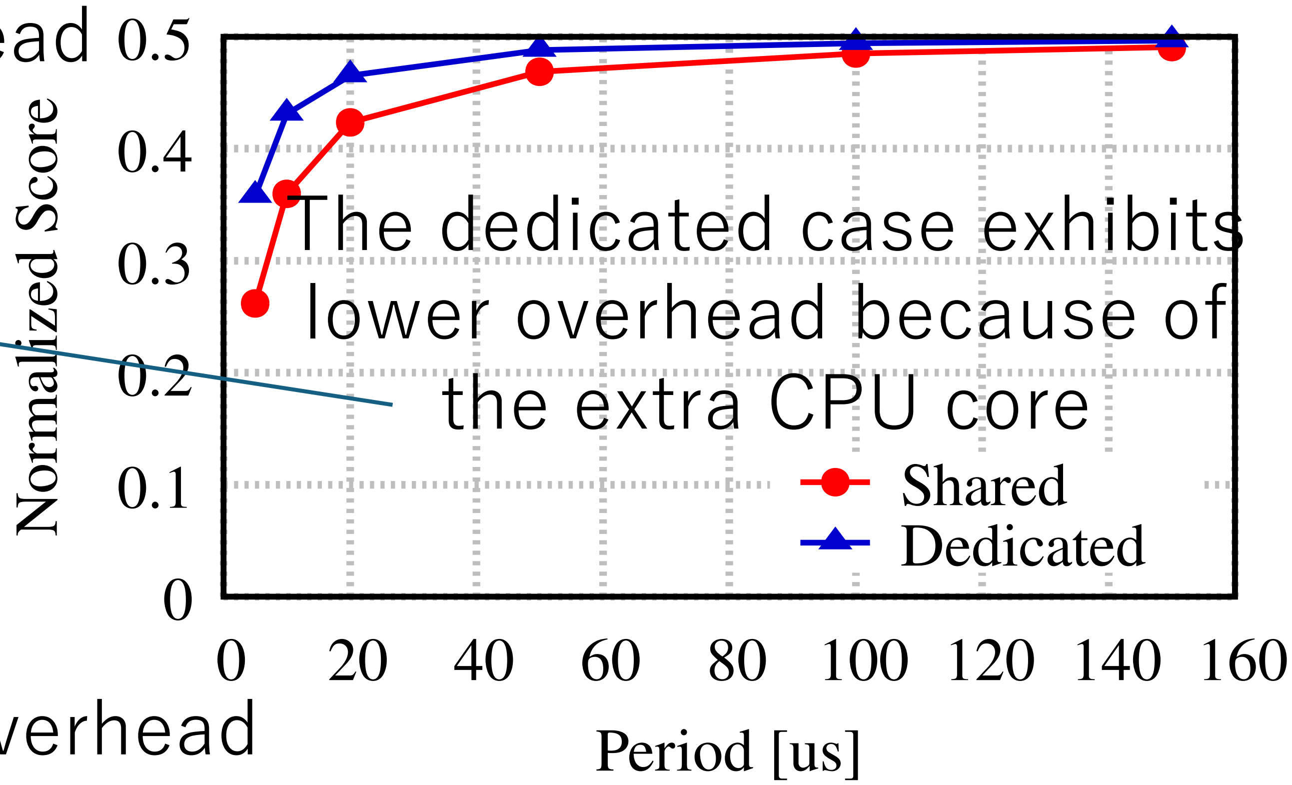


A : sysbench (CPU)

B : busy loop

Gap to 0.5 shows the CPU overhead coming from the scheduling





The dedicated case exhibits lower overhead because of the extra CPU core

A shorter period leads to higher overhead

Use Case: Microsecond-scale Time Slicing

- Previous work showed microsecond-scale time slicing contributes to application performance
  - vTurbo (USENIX ATC'13), micro-sliced cores (EuroSys'18)

Use Case: Microsecond-scale Time Slicing

- Previous work showed microsecond-scale time slicing contributes to application performance
  - vTurbo (USENIX ATC'13), micro-sliced cores (EuroSys'18)
- However, the minimum configurable time slice on Linux is 1 millisecond (ensured by the kernel build system)

Use Case: Microsecond-scale Time Slicing

- Previous work showed microsecond-scale time slicing contributes to application performance
  - vTurbo (USENIX ATC'13), micro-sliced cores (EuroSys'18)
- However, the minimum configurable time slice on Linux is 1 millisecond (ensured by the kernel build system)
- The priority elevation trick allows us to apply microsecond-scale time slices on unmodified Linux

Here, we see how it affects networked server performance

Use Case: Microsecond-scale Time Slicing

- We implemented a networked server for the experiments

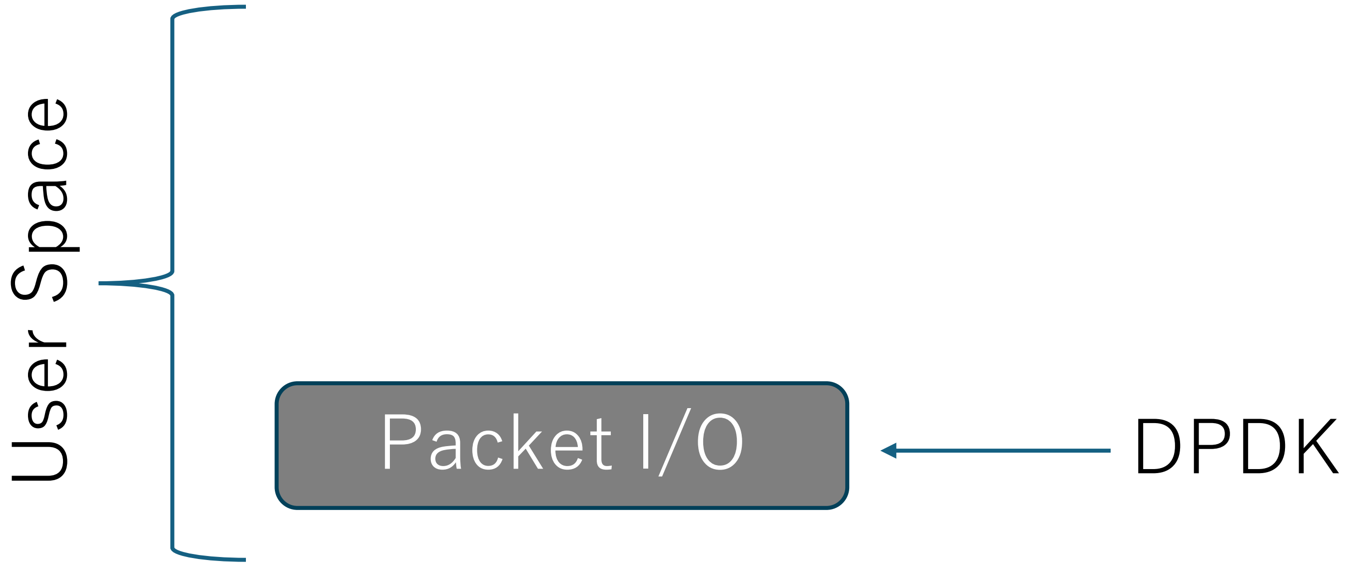
Use Case: Microsecond-scale Time Slicing

- We implemented a networked server for the experiments

User Space

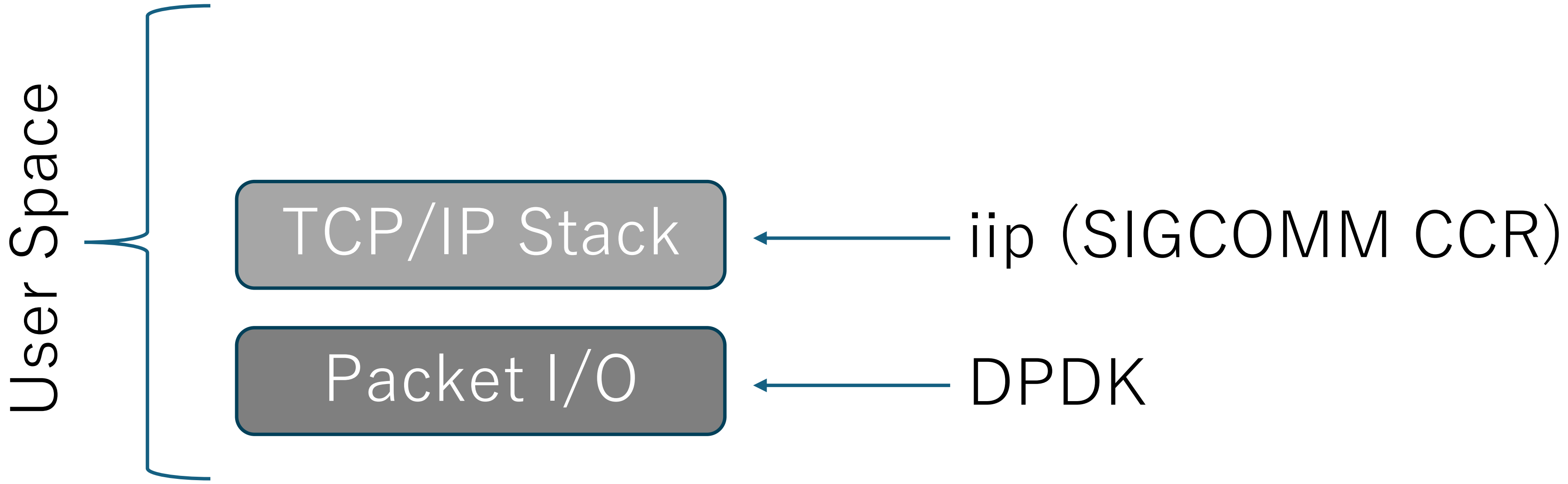
Use Case: Microsecond-scale Time Slicing

- We implemented a networked server for the experiments



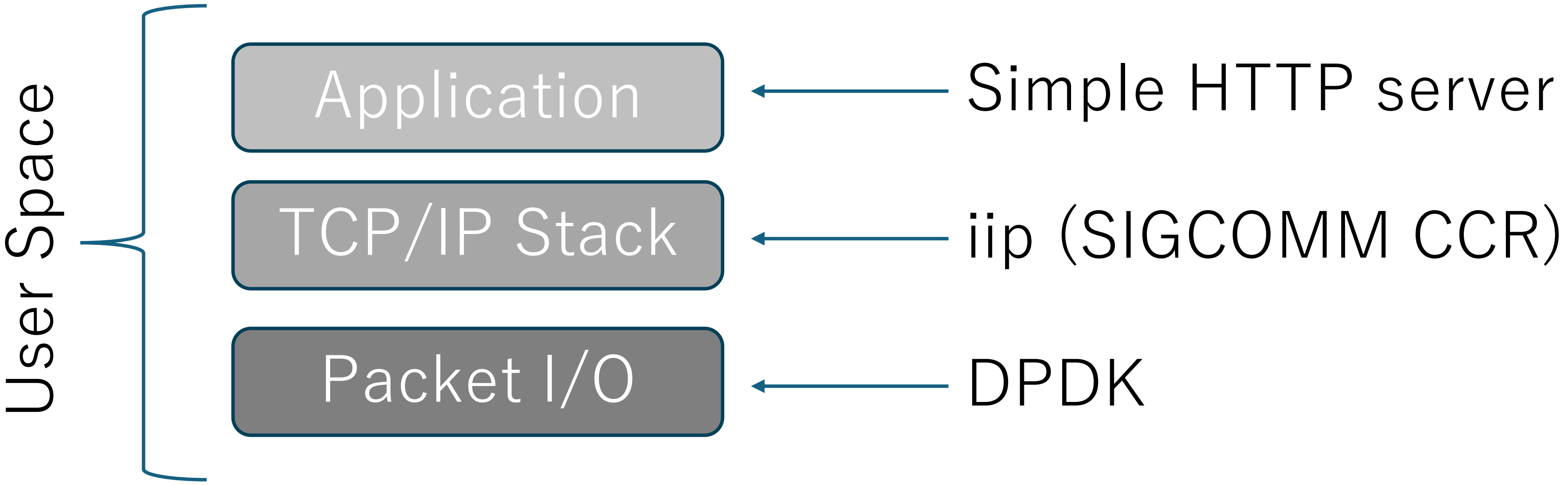
Use Case: Microsecond-scale Time Slicing

- We implemented a networked server for the experiments



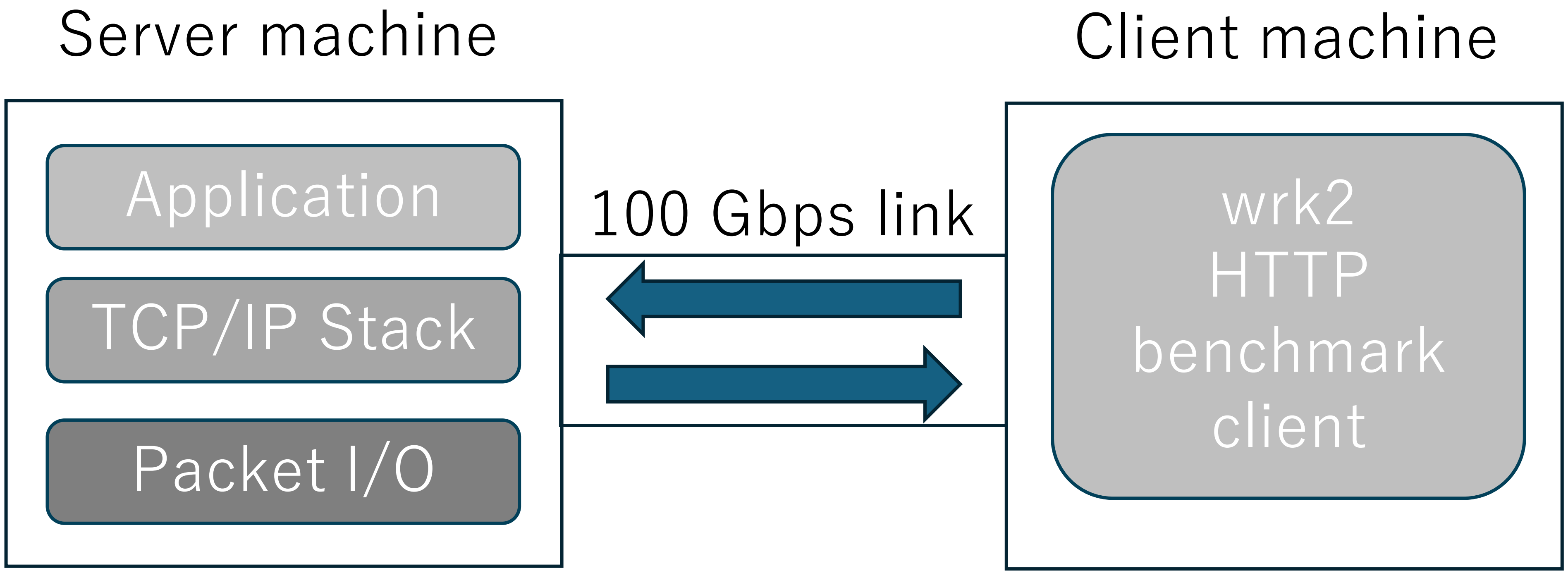
Use Case: Microsecond-scale Time Slicing

- We implemented a networked server for the experiments



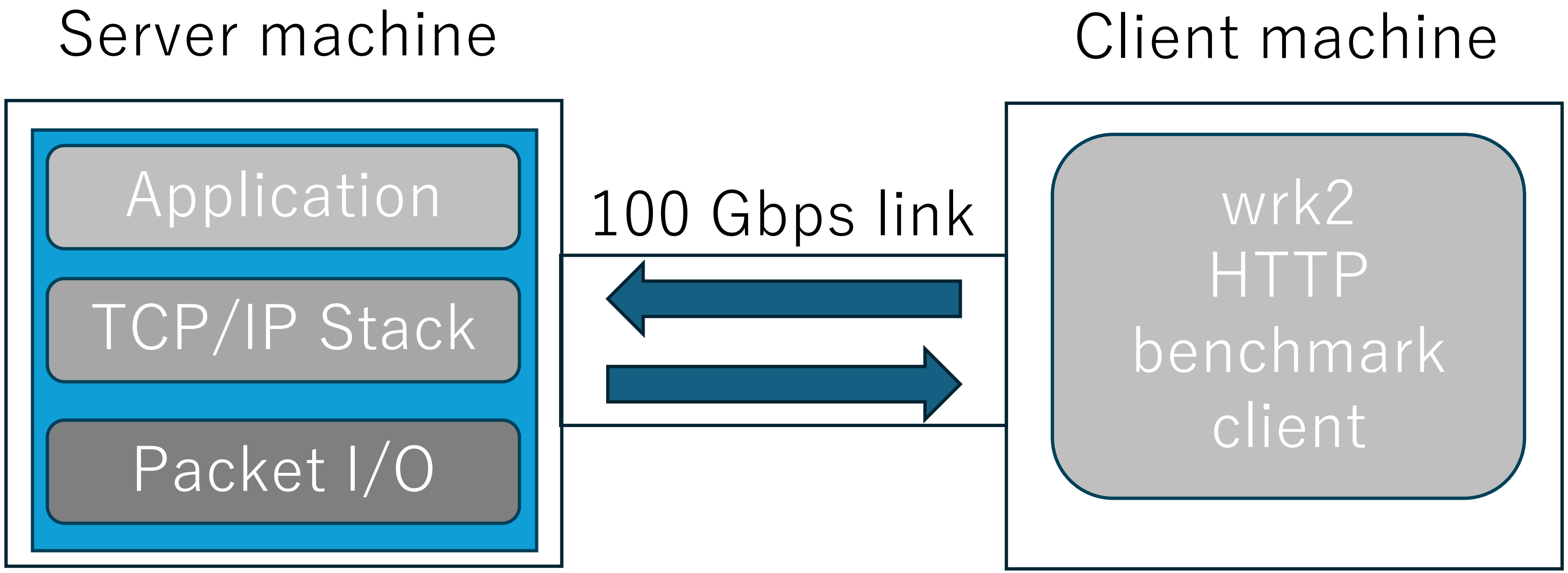
Use Case: Microsecond-scale Time Slicing

- The client machine runs wrk2 to send requests

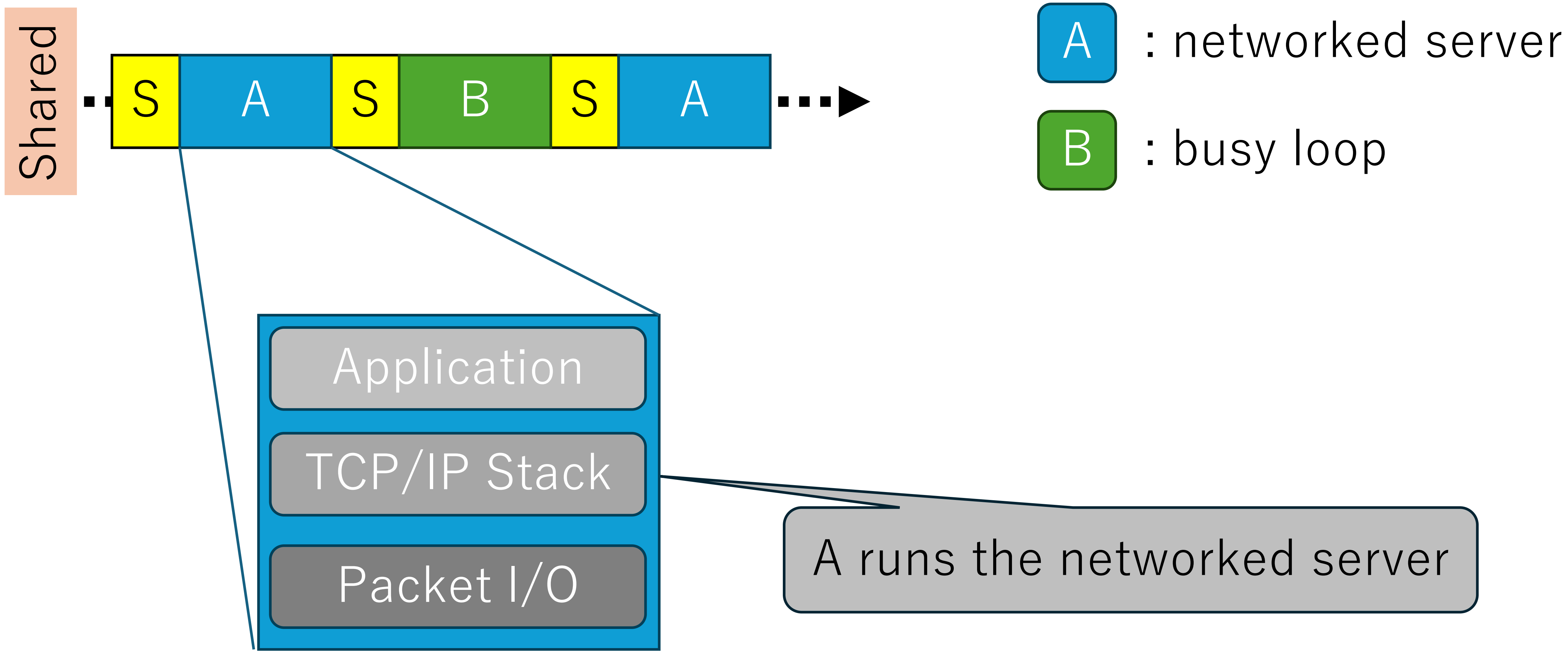


Use Case: Microsecond-scale Time Slicing

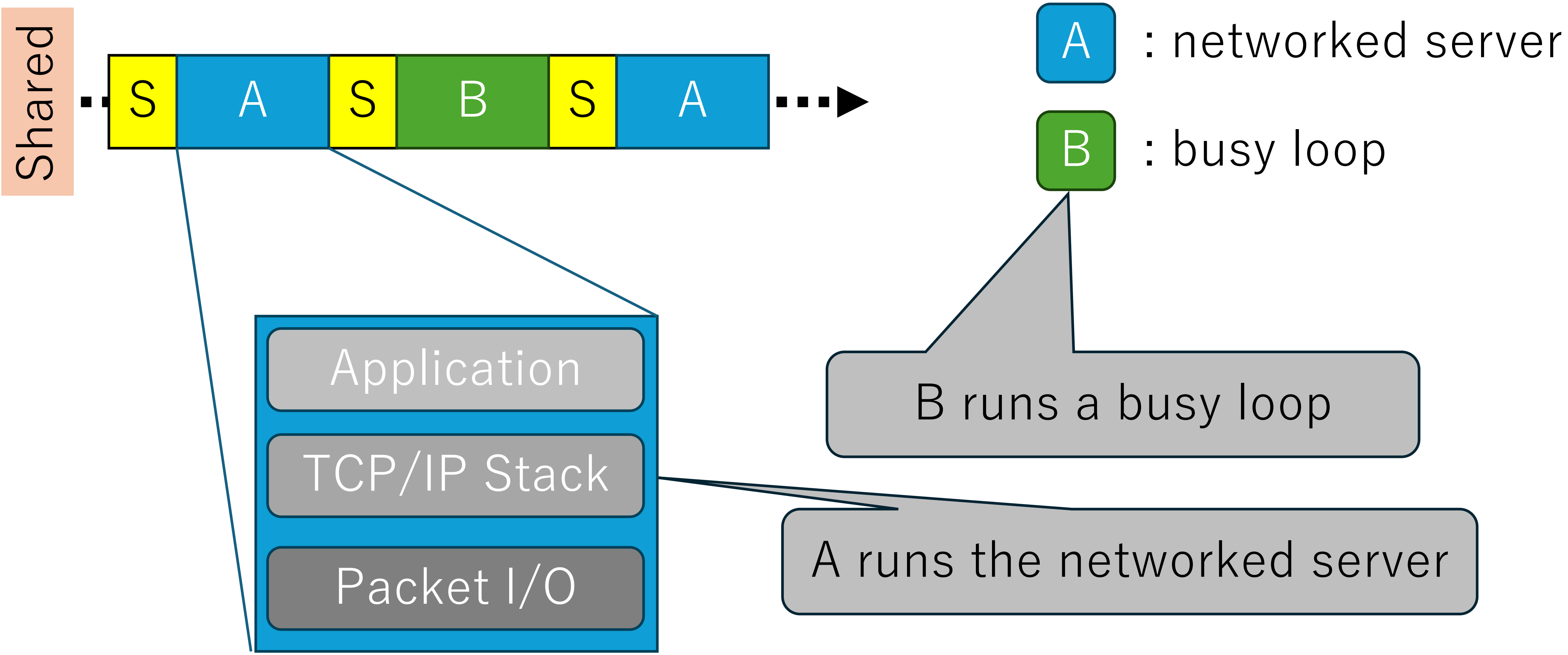
- The client machine runs wrk2 to send requests



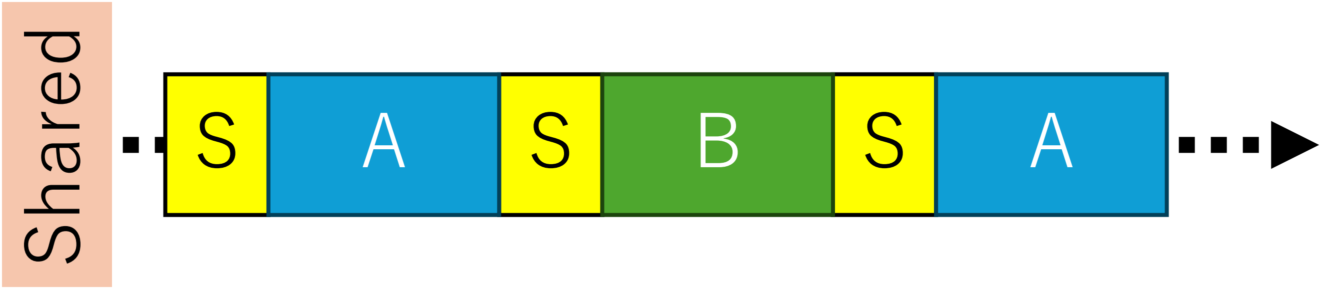
Use Case: Microsecond-scale Time Slicing



Use Case: Microsecond-scale Time Slicing



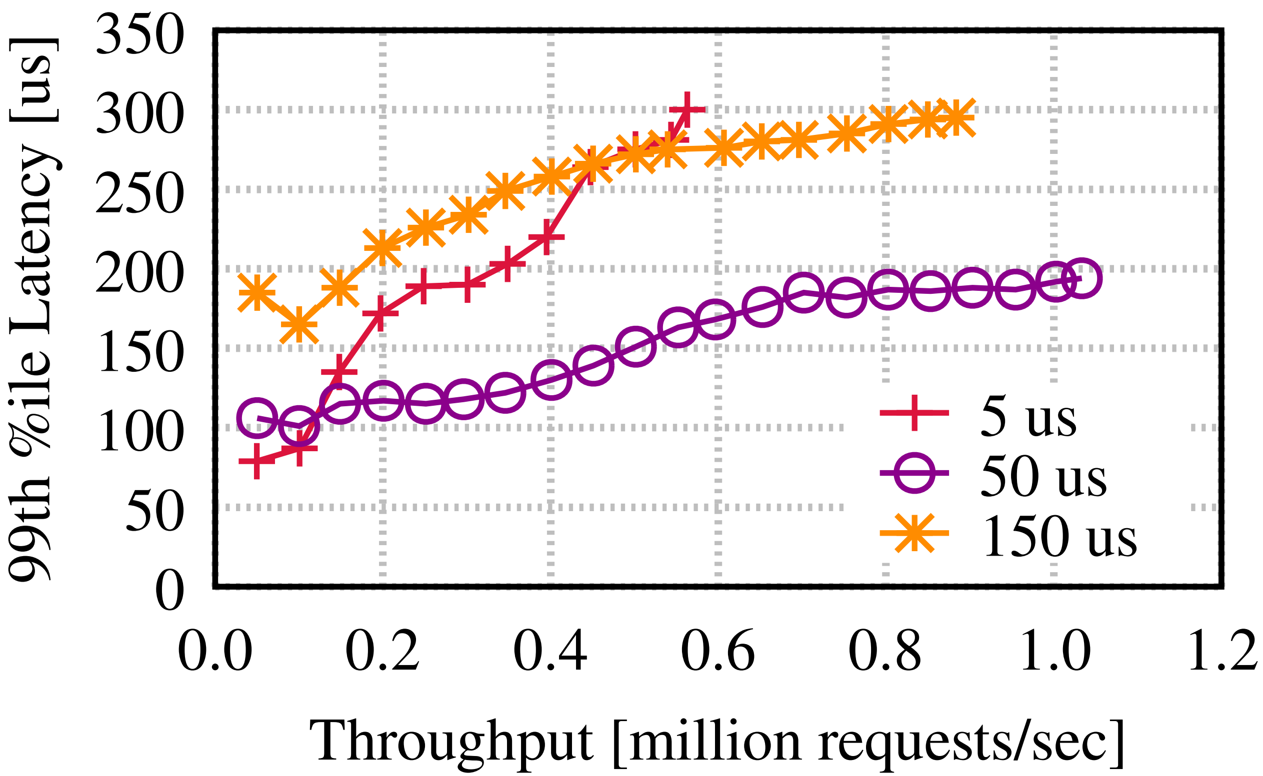
Use Case: Microsecond-scale Time Slicing



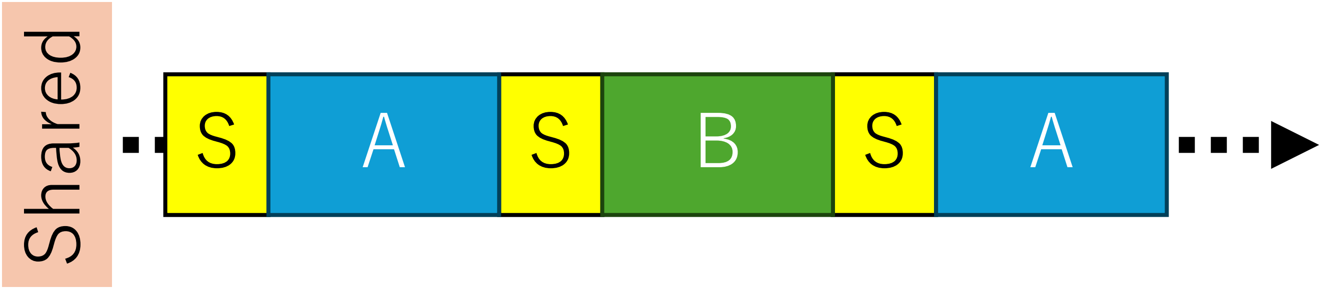
A : networked server

B : busy loop

Performance of
the networked server on A
with different time slices

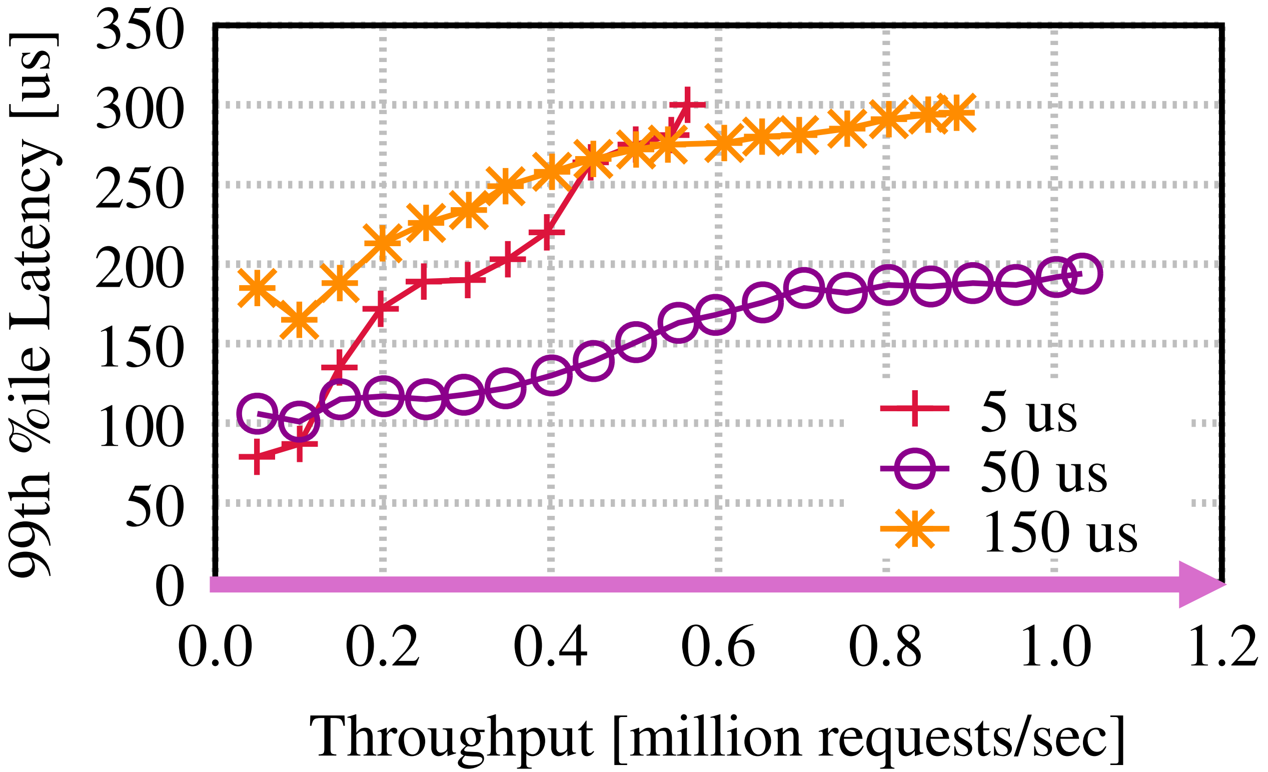


Use Case: Microsecond-scale Time Slicing

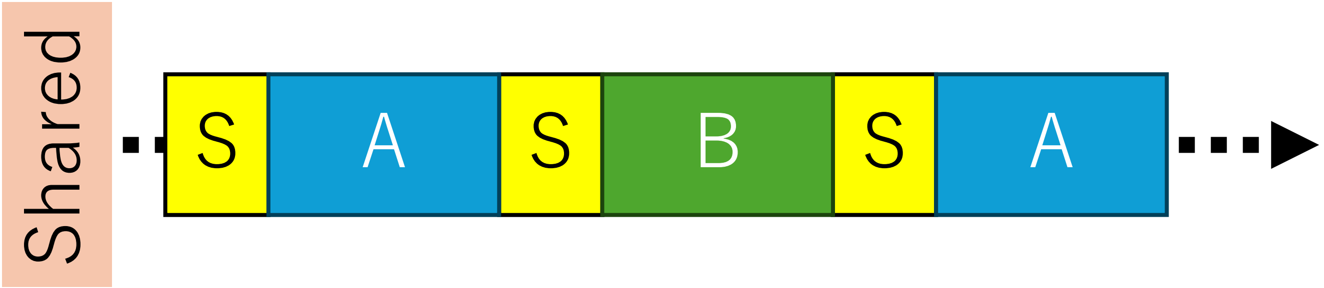


A : networked server

B : busy loop

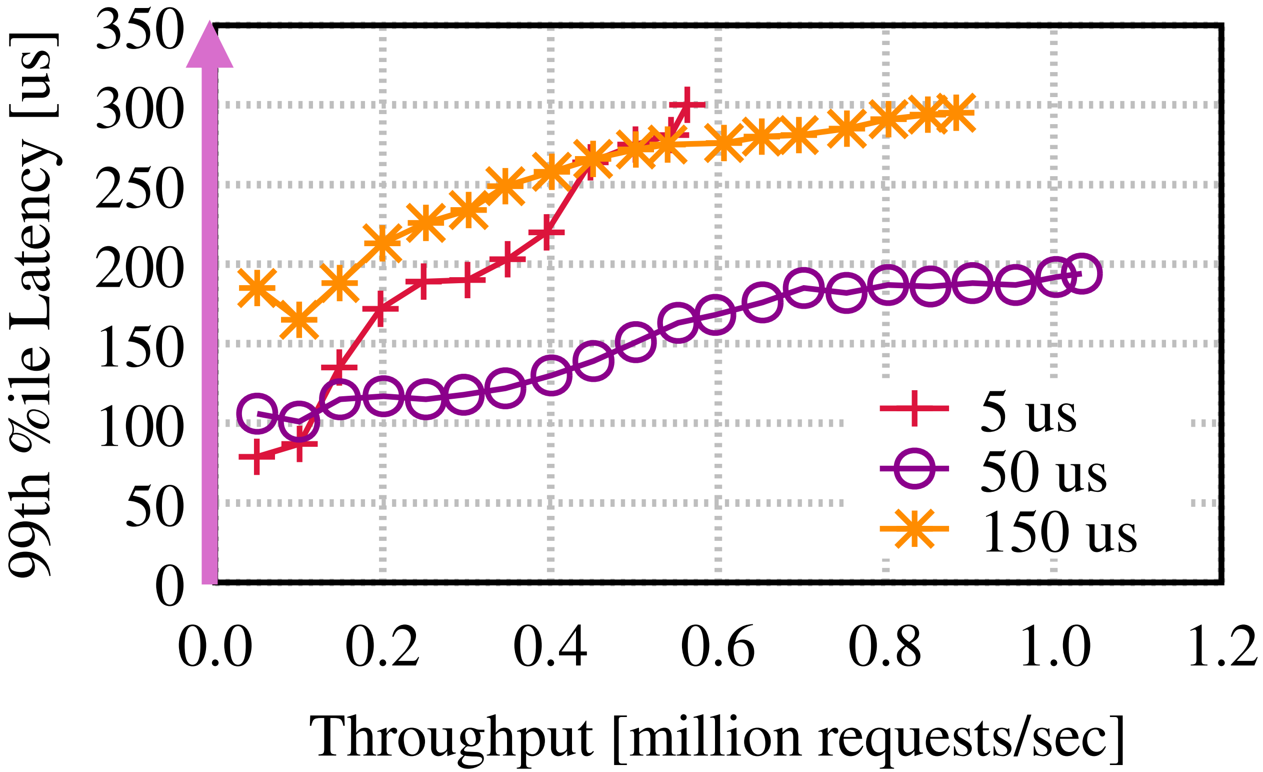


Use Case: Microsecond-scale Time Slicing

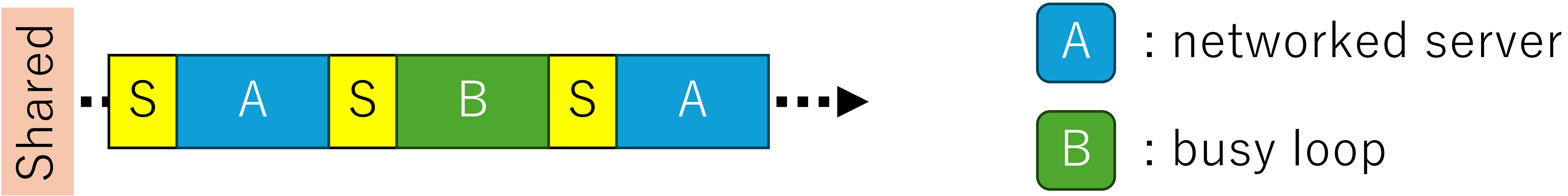


A : networked server

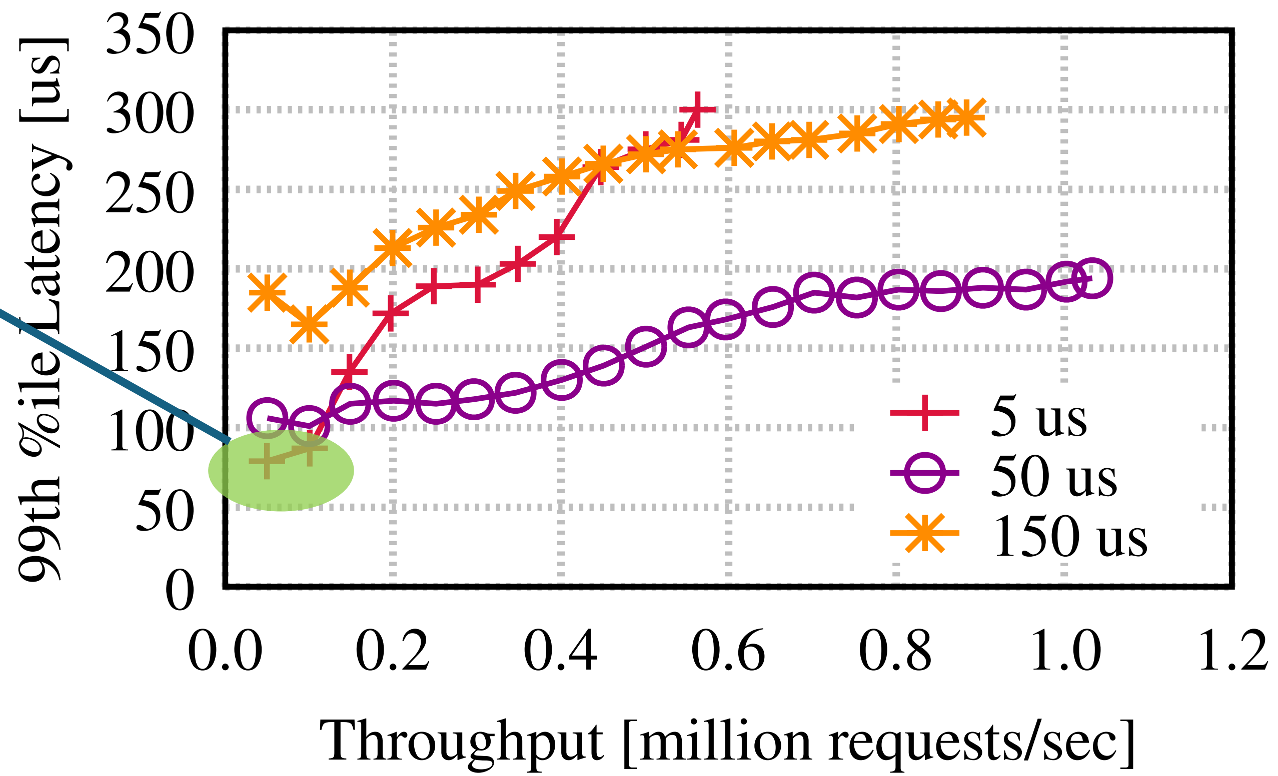
B : busy loop



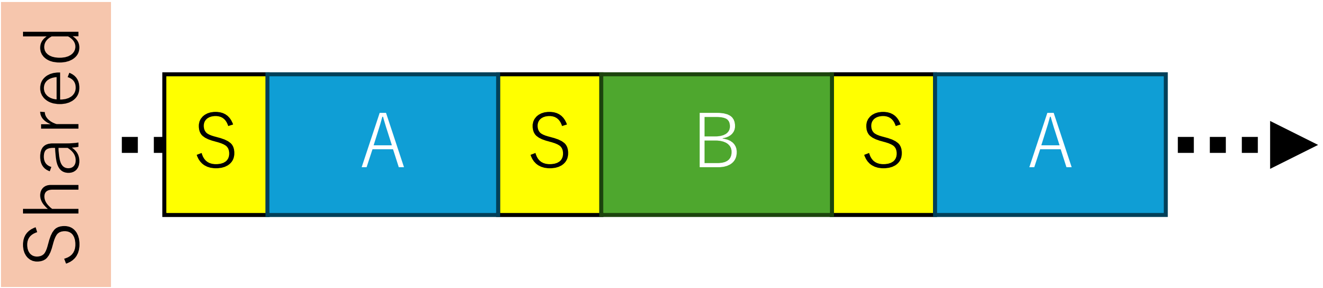
Use Case: Microsecond-scale Time Slicing



A shorter switching period leads to lower latency



Use Case: Microsecond-scale Time Slicing

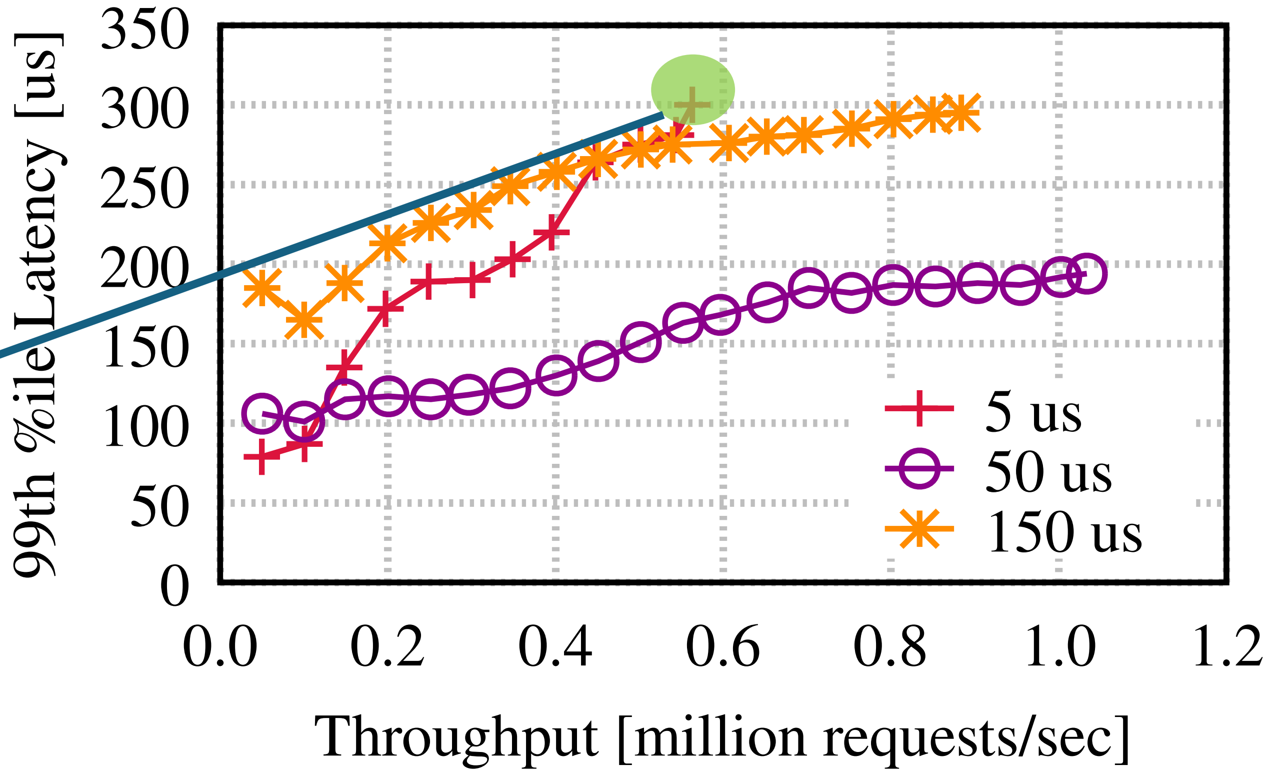


A : networked server

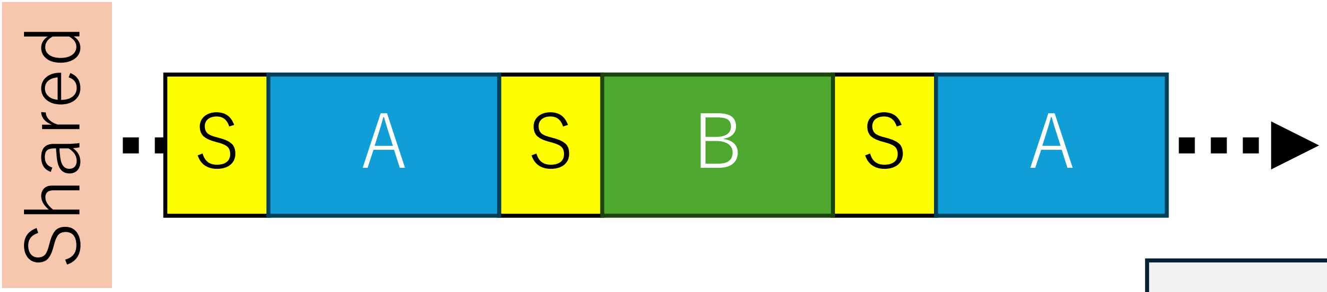
B : busy loop

A shorter switching period leads to lower latency

A too short switching period leads to low throughput



Use Case: Microsecond-scale Time Slicing

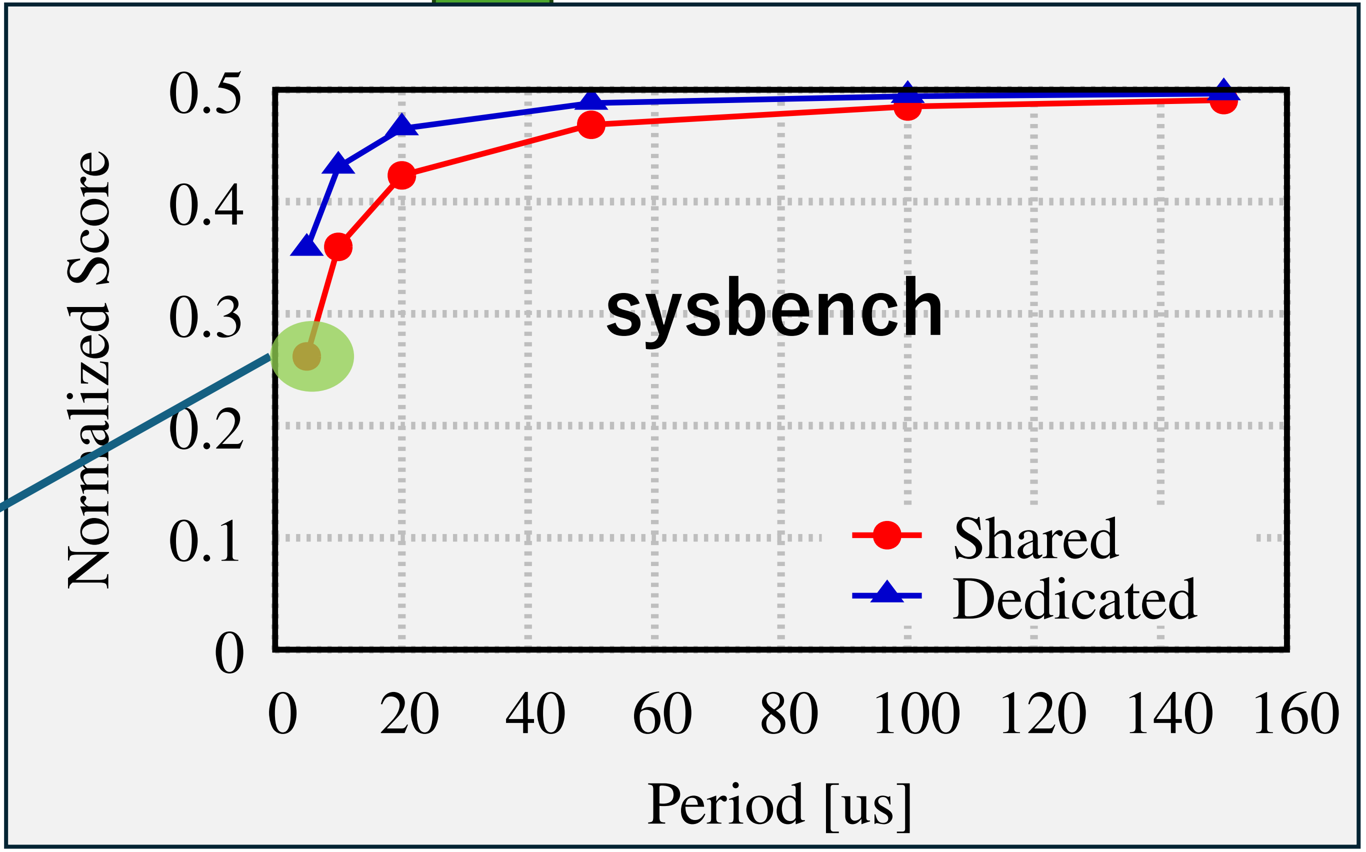


A : networked server

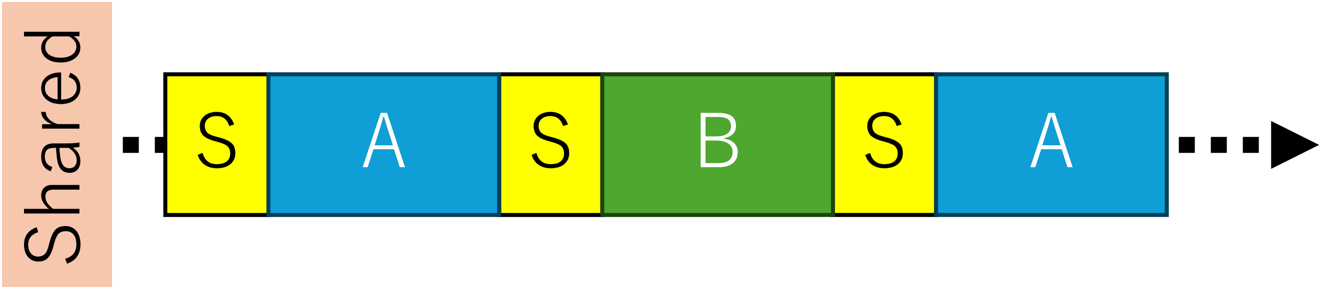
B : busy loop

A shorter switching period leads to lower latency

A too short switching period leads to low throughput because of the CPU overhead



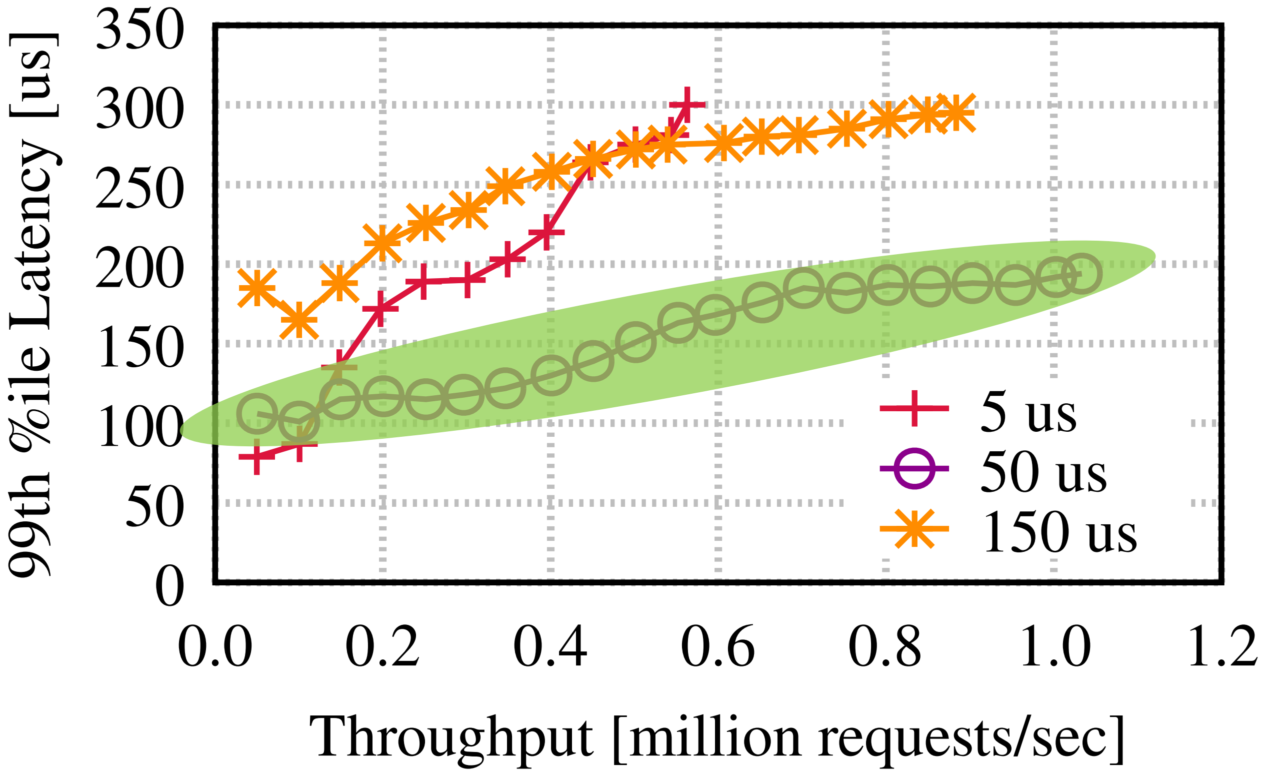
Use Case: Microsecond-scale Time Slicing



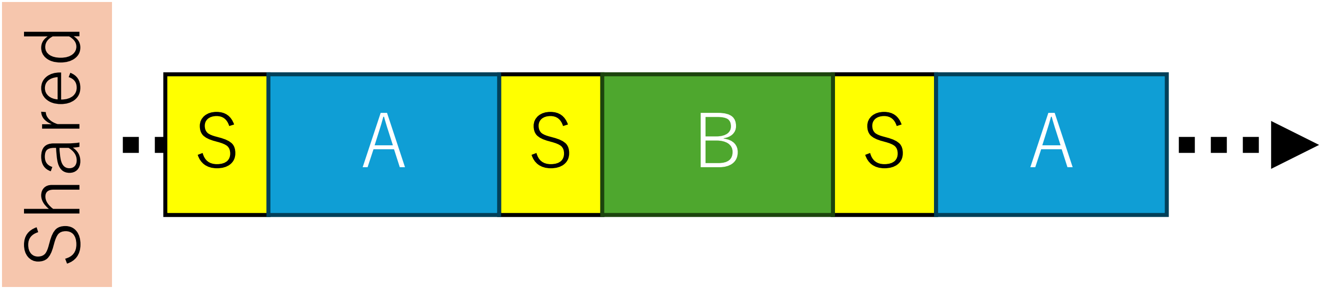
A : networked server

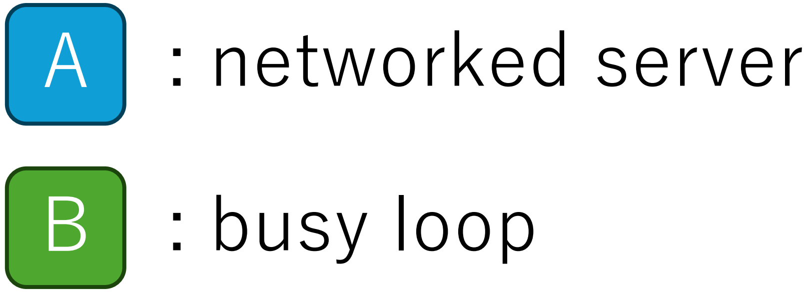
B : busy loop

Time slice setting is
crucial for performance



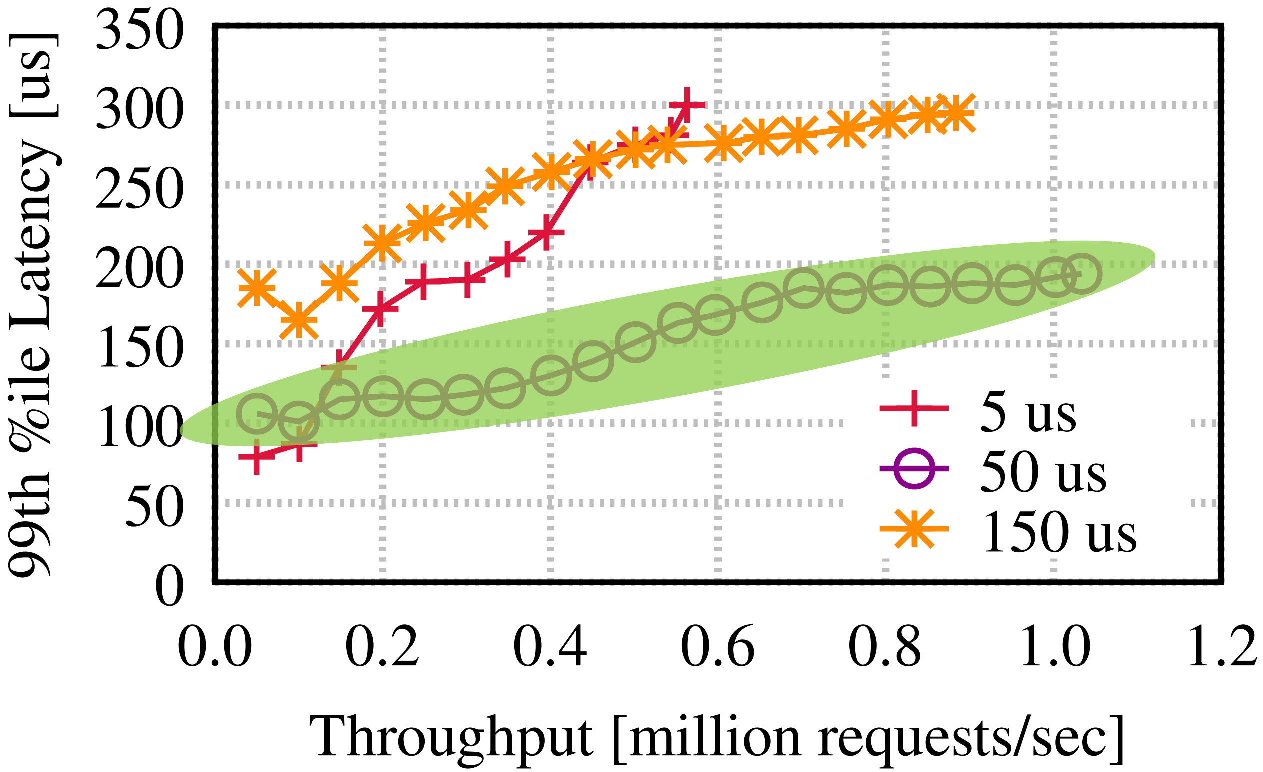
Use Case: Microsecond-scale Time Slicing





Time slice setting is crucial for performance

The priority elevation trick allows us to apply microsecond-scale time slicing



Use Case: Table-driven Scheduling

- Previous work showed that table-driven scheduling adopting static scheduling table improves application performance
  - Tableau (EuroSys’18)

Use Case: Table-driven Scheduling

- Previous work showed that table-driven scheduling adopting static scheduling table improves application performance
  - Tableau (EuroSys'18)

- However, Linux does not provide a configuration interface allowing users to install static scheduling tables

Use Case: Table-driven Scheduling

- Previous work showed that table-driven scheduling adopting static scheduling table improves application performance
  - Tableau (EuroSys’18)
- However, Linux does not provide a configuration interface allowing users to install static scheduling tables
- The priority elevation trick allows us to realize table-driven scheduling without changing the kernel

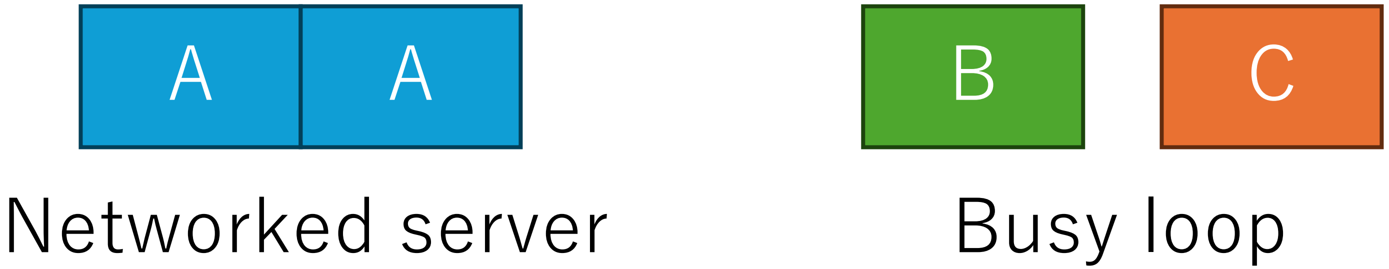
We see how it contributes to networked server performance

Use Case: Table-driven Scheduling

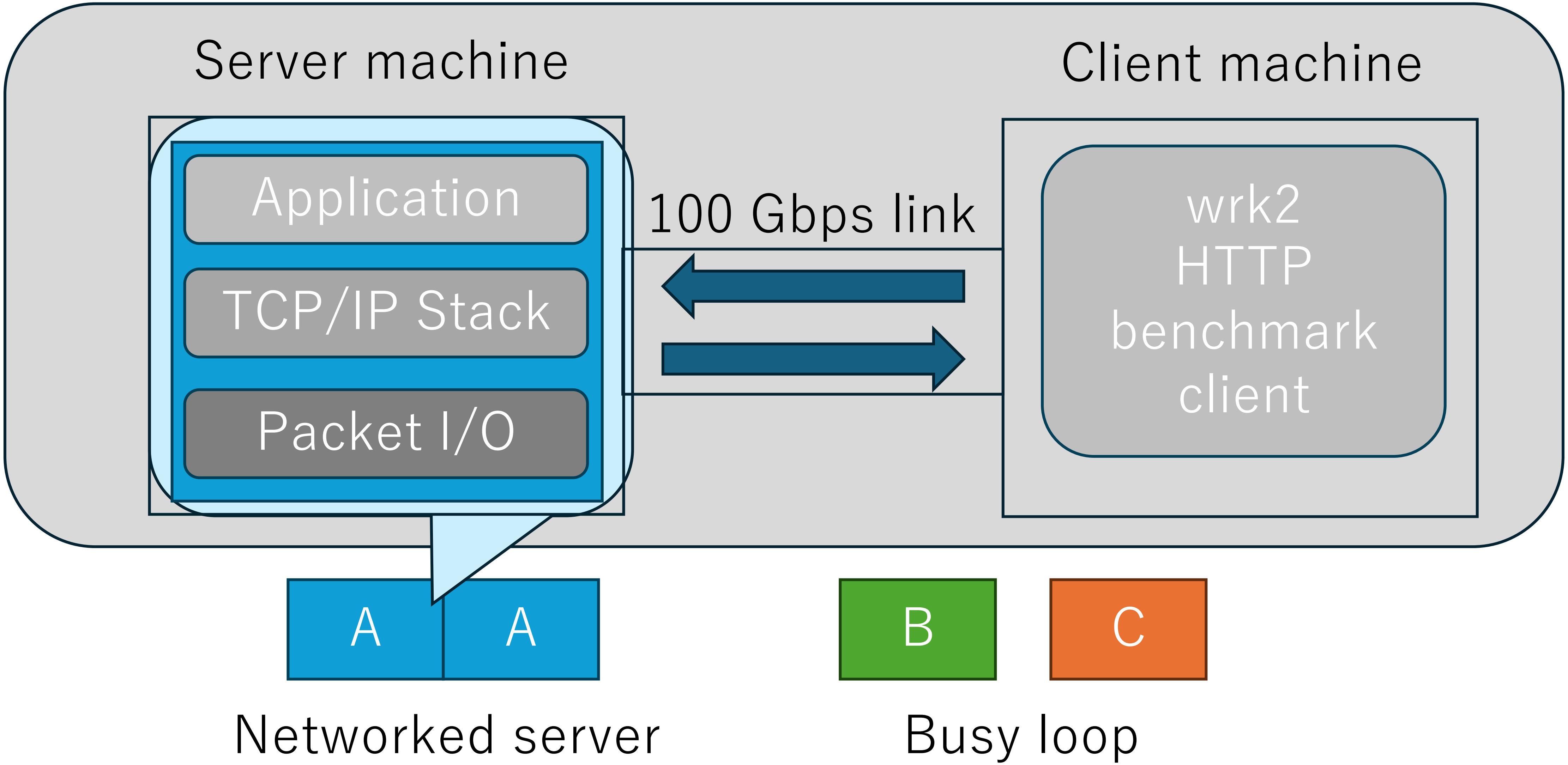
- Scenario: A, B, and C run on the same CPU core

Use Case: Table-driven Scheduling

- Scenario: A, B, and C run on the same CPU core
- A runs the networked server, and B and C run a busy loop

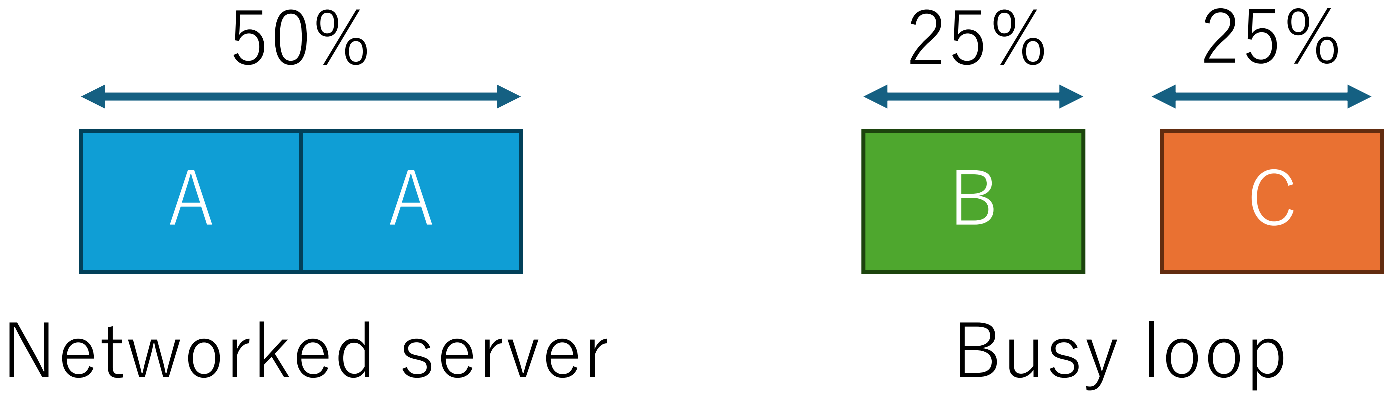


Use Case: Table-driven Scheduling



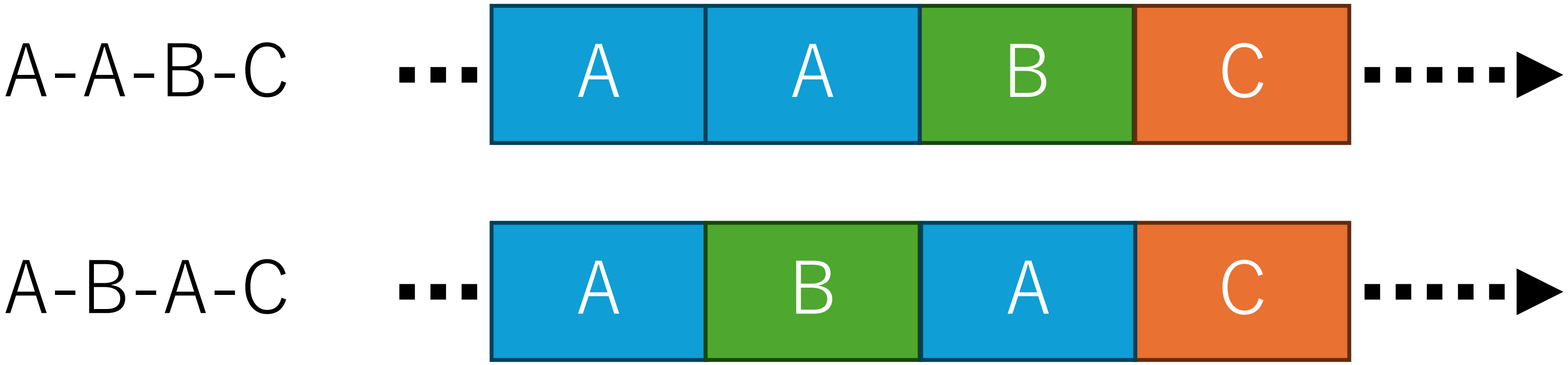
Use Case: Table-driven Scheduling

- Scenario: A, B, and C run on the same CPU core
- A runs the networked server, and B and C run a busy loop
- We assign 50% of CPU time to A and 25% to B and C each



Use Case: Table-driven Scheduling

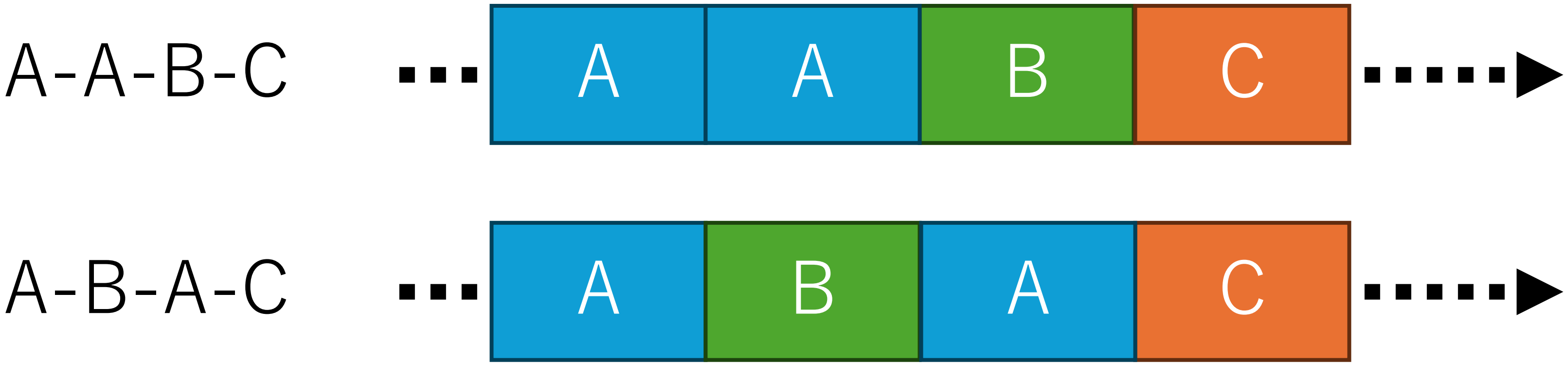
- Two ordering patterns: A-A-B-C and A-B-A-C



The kernel-space process scheduler does not offer an interface to specify the order of the scheduling

Use Case: Table-driven Scheduling

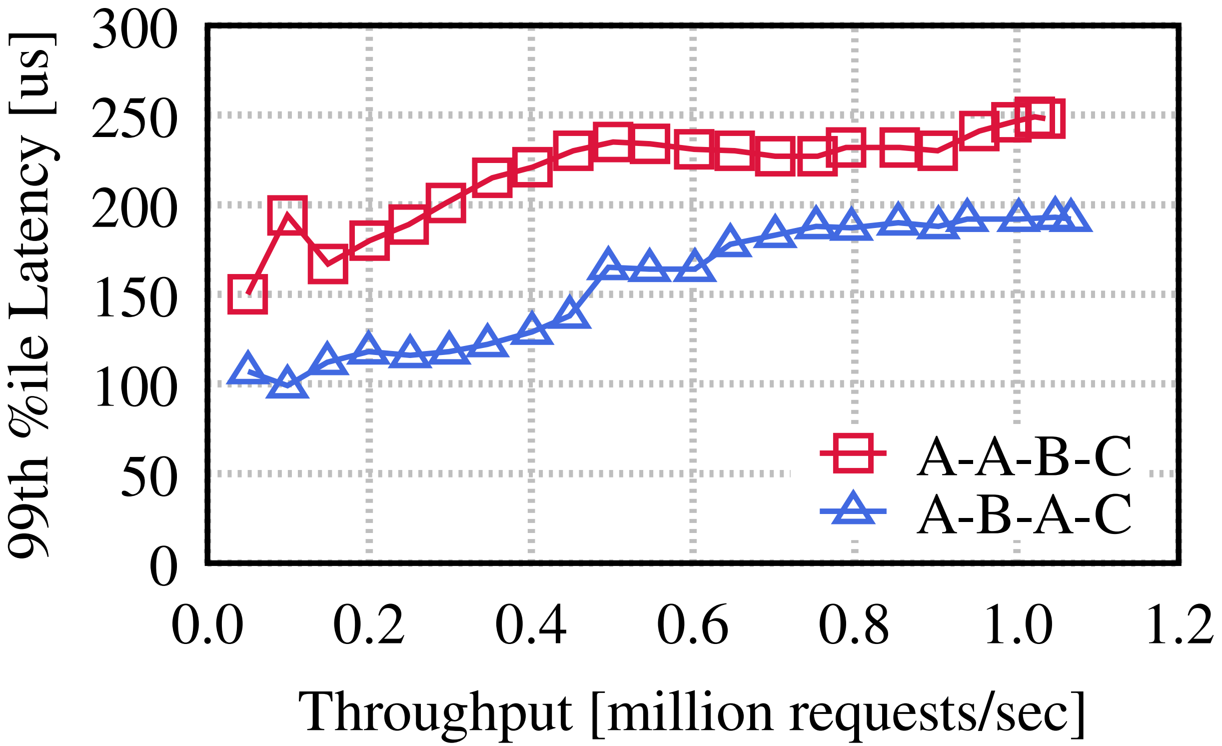
- Two ordering patterns: A-A-B-C and A-B-A-C



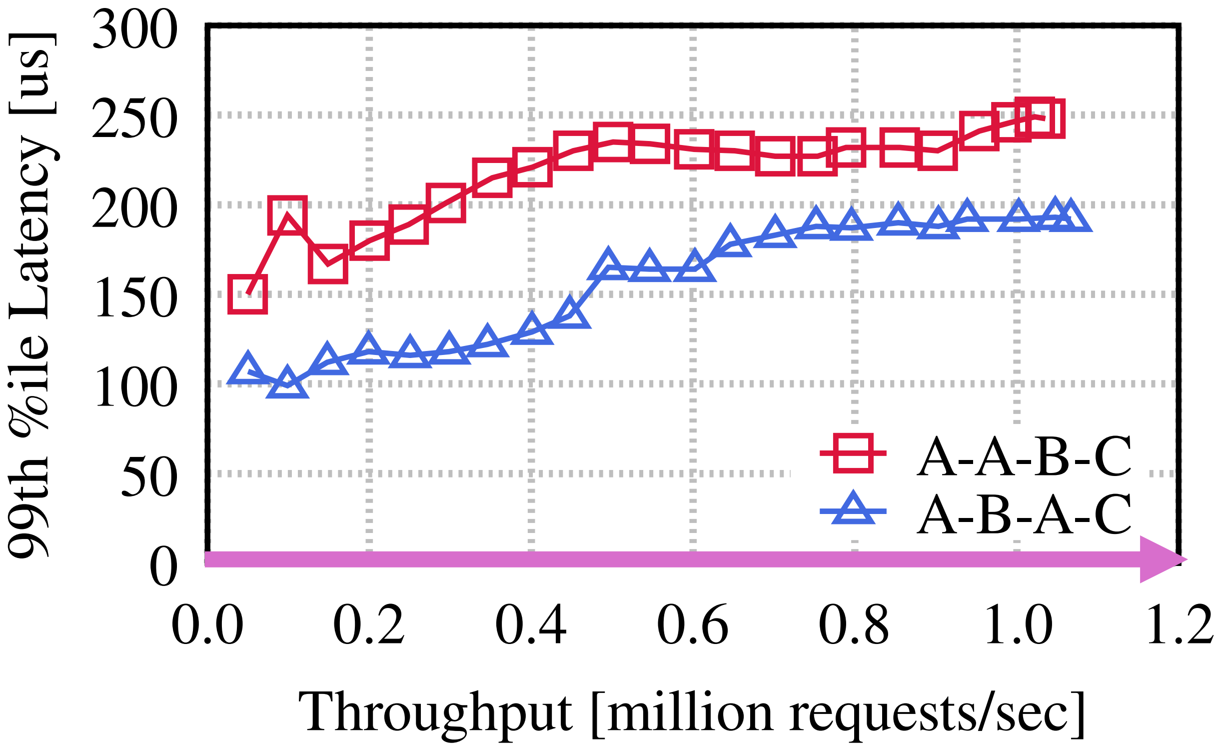
The priority elevation trick allows us to implement scheduling tables to ensure these ordering patterns

Use Case: Table-driven Scheduling

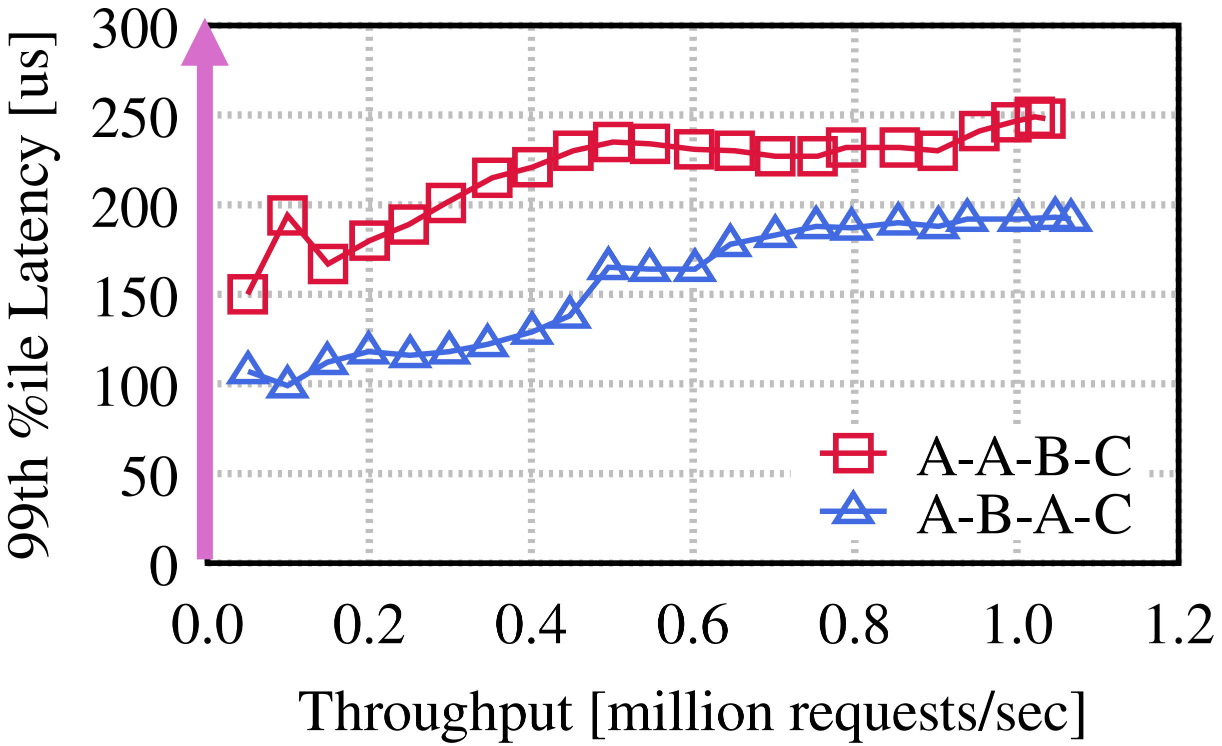
Performance of the networked server on A



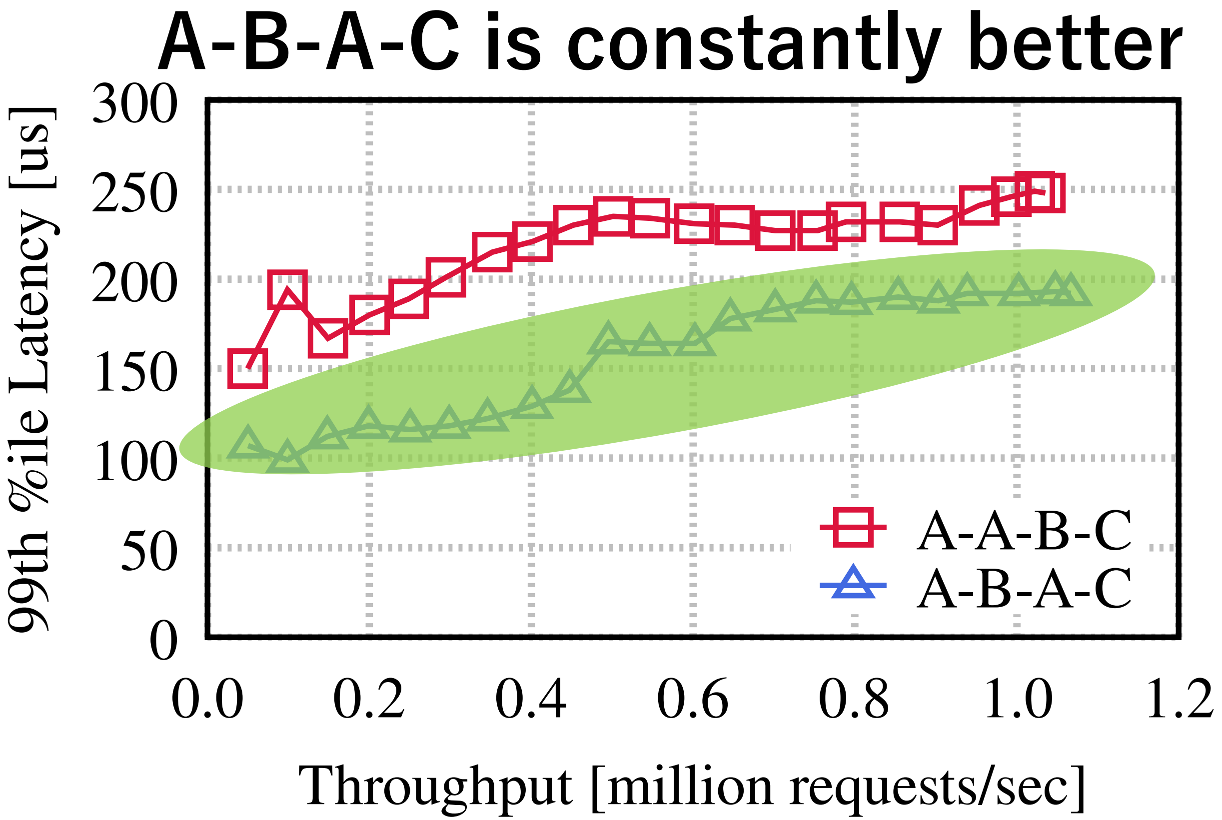
Use Case: Table-driven Scheduling



Use Case: Table-driven Scheduling

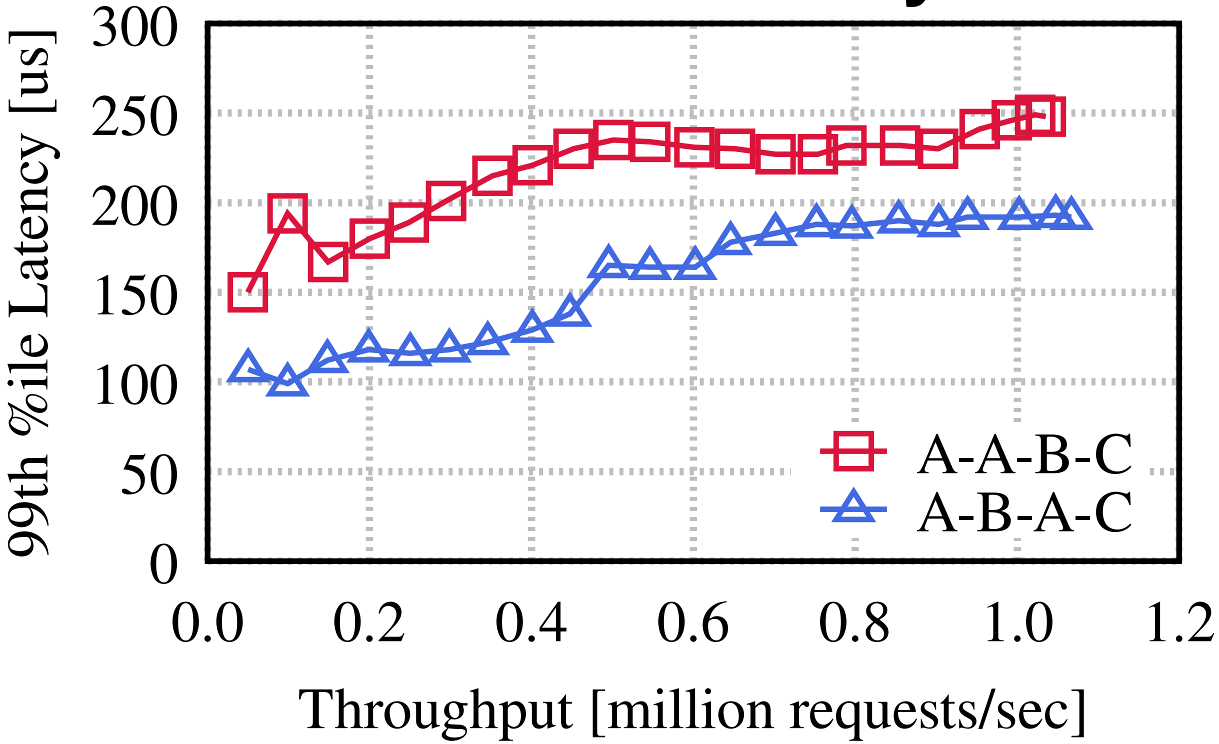


Use Case: Table-driven Scheduling

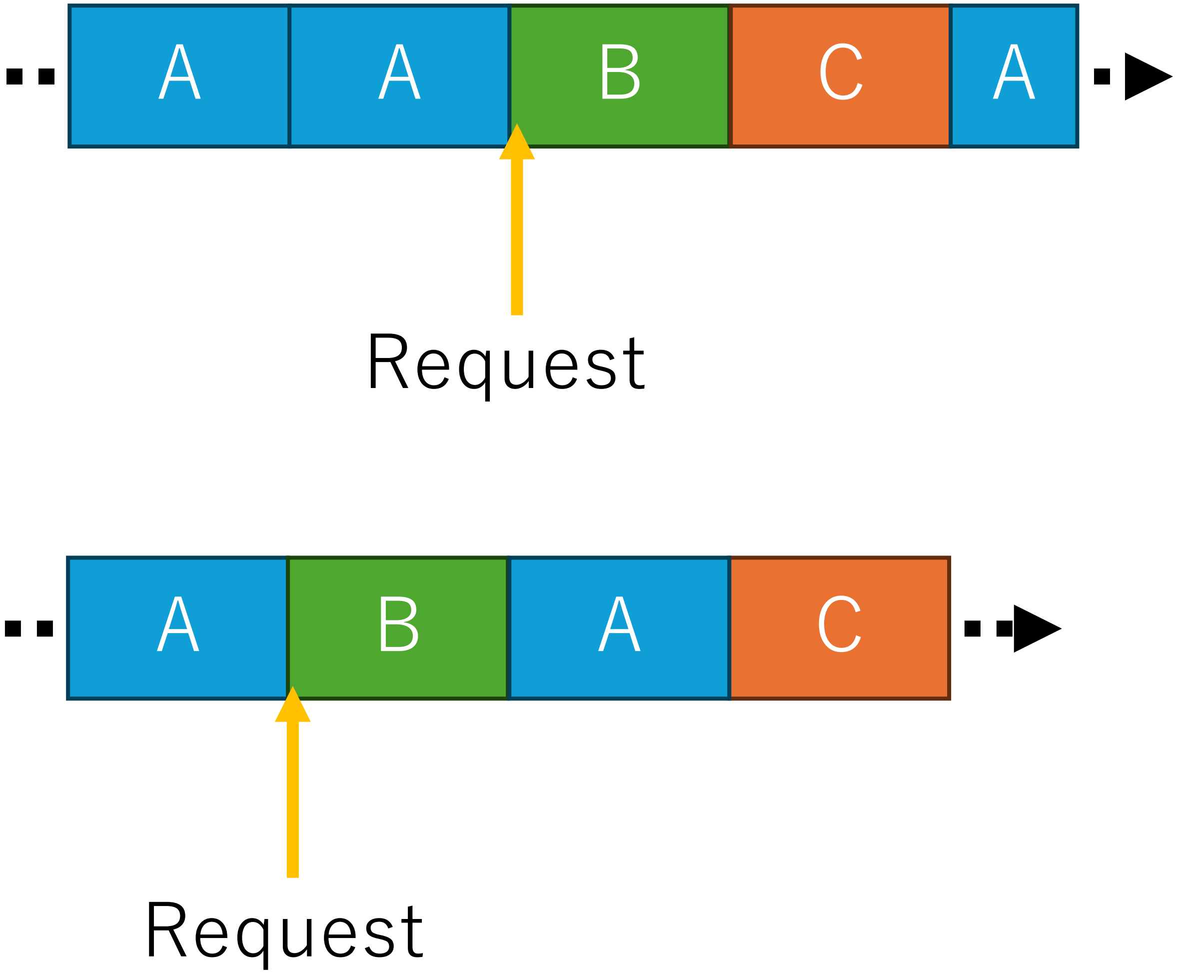


Use Case: Table-driven Scheduling

A-B-A-C is constantly better

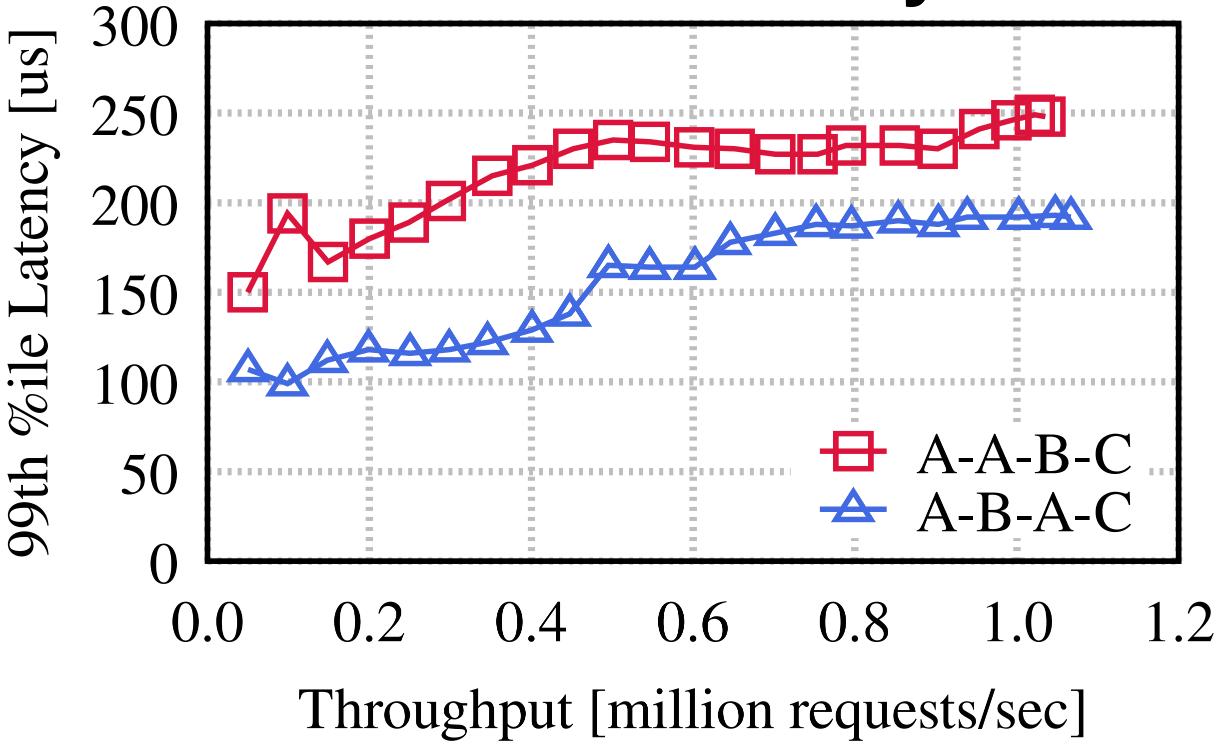


Use Case: Table-driven Scheduling

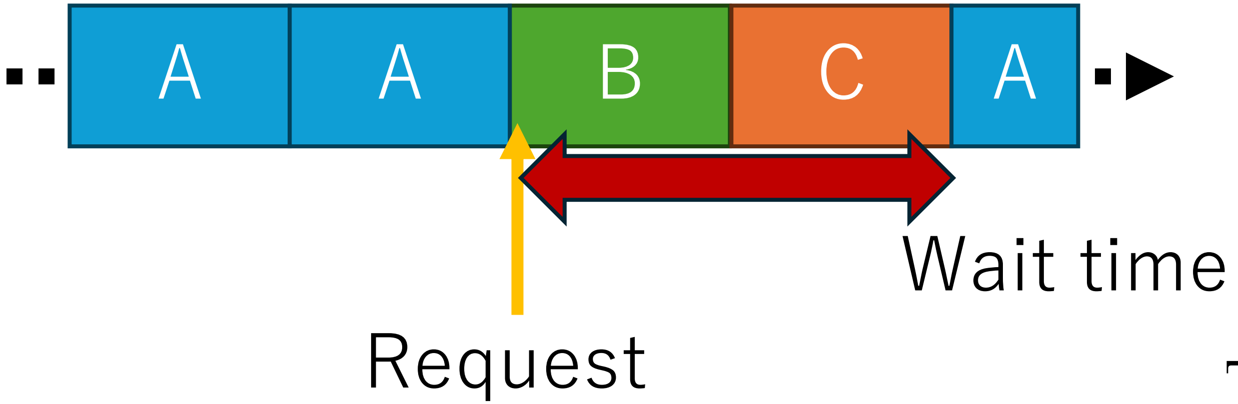


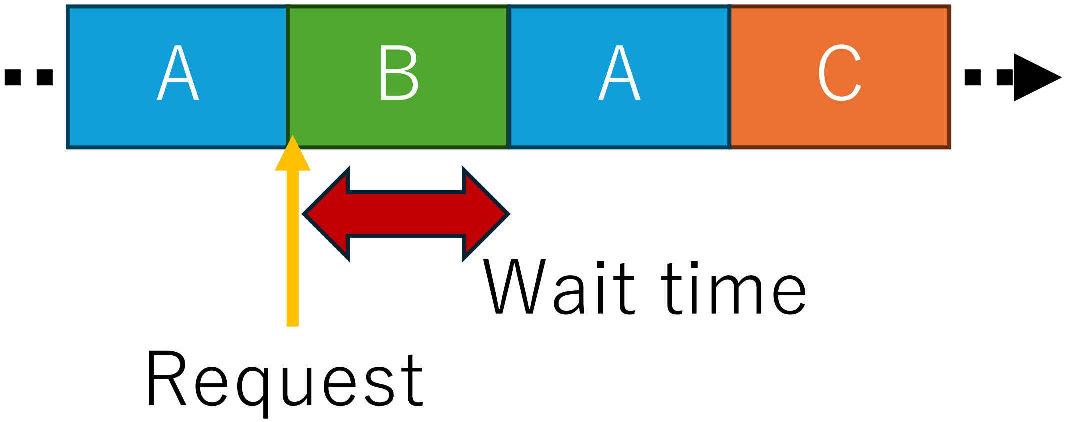
A request arrives when
B has been resumed

A-B-A-C is constantly better

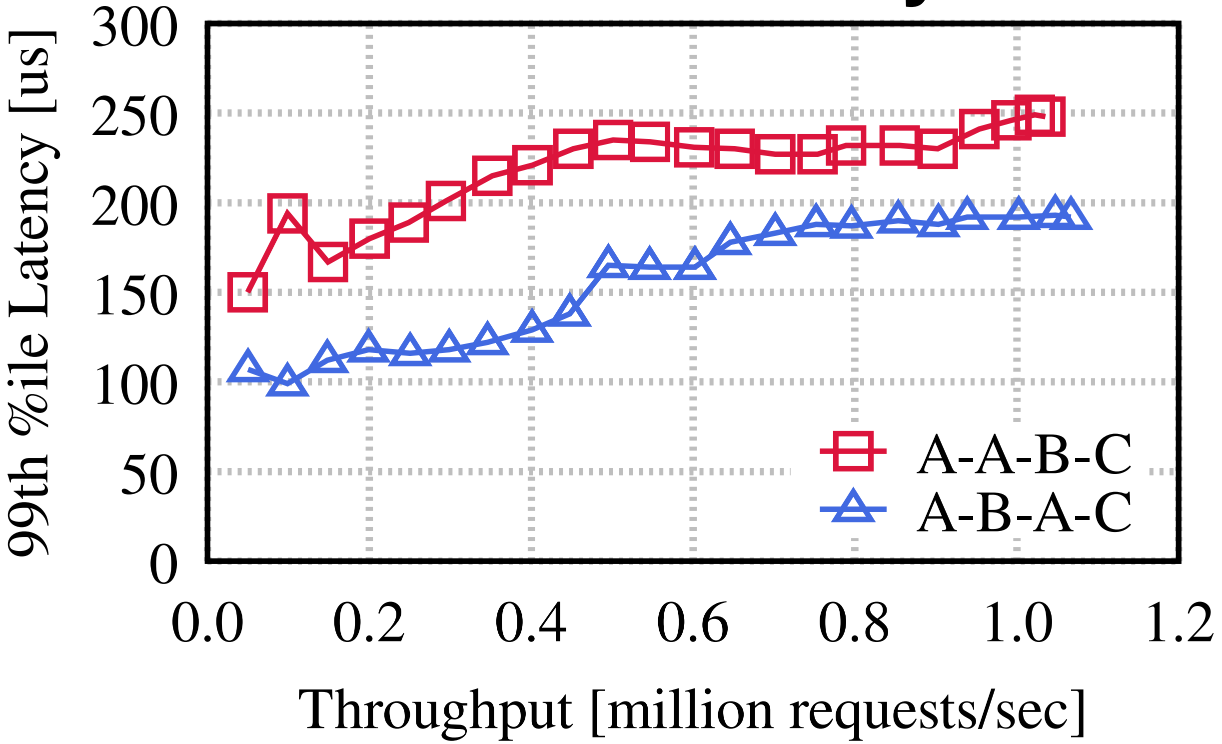


Use Case: Table-driven Scheduling

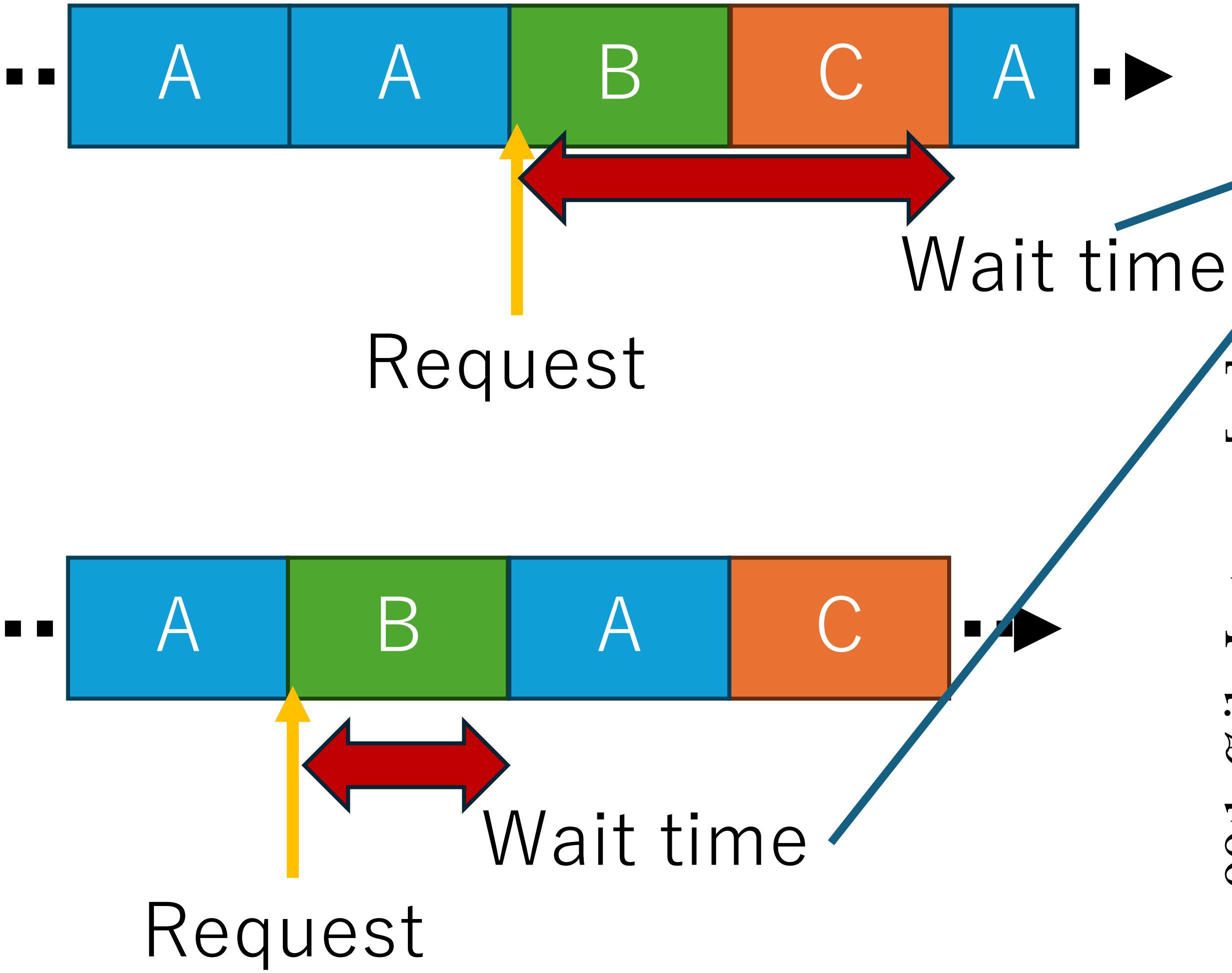




A-B-A-C is constantly better

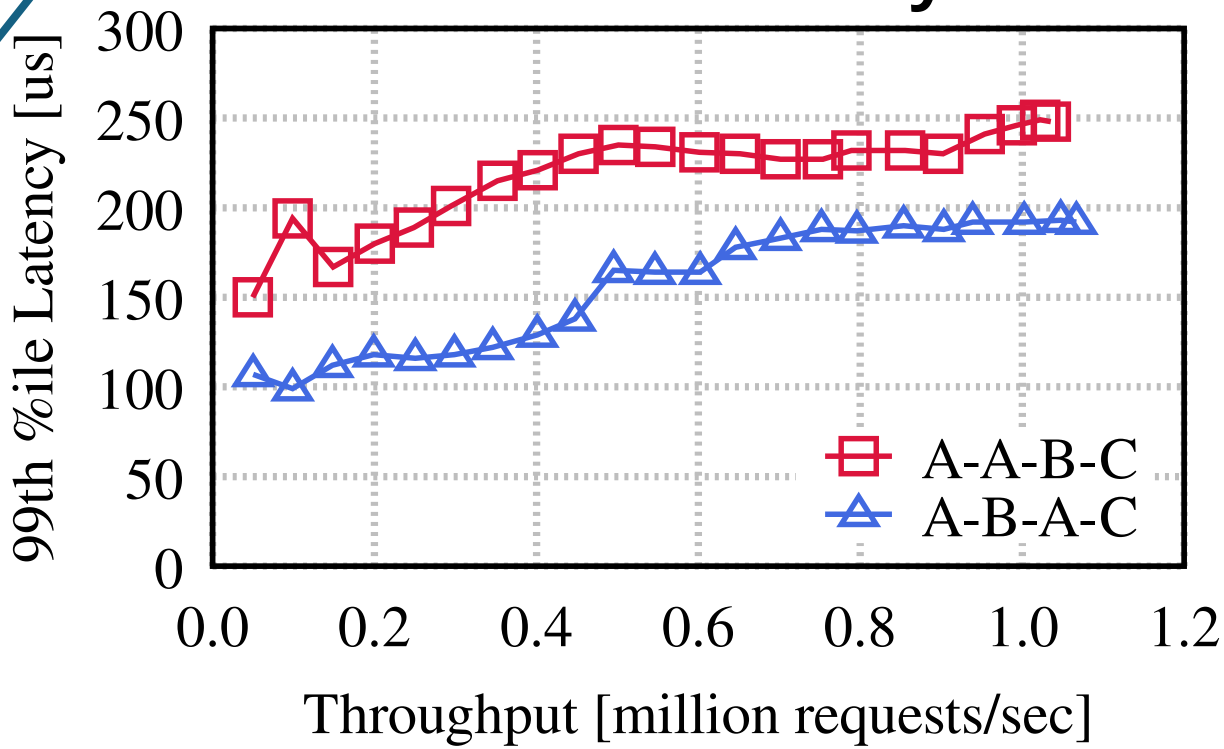


Use Case: Table-driven Scheduling



The wait time for this request is shorter in A-B-A-C

A-B-A-C is constantly better

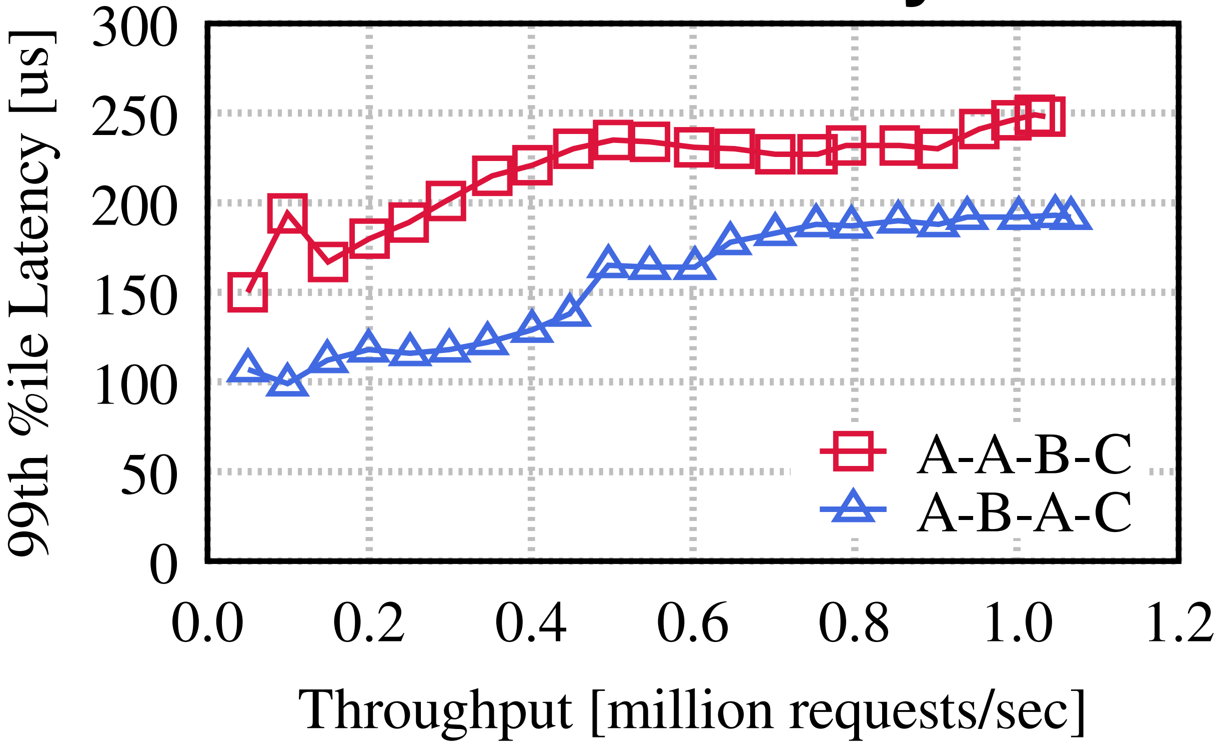


Use Case: Table-driven Scheduling

Table-driven scheduling realized by the priority elevation trick brings performance benefit

The wait time for this request is shorter in A-B-A-C

A-B-A-C is constantly better



Use Case: Preemptive Scheduling

- Previous work proposed to adopt preemptive scheduling to mitigate the head-of-line blocking issue
  - Shinjuku (NSDI’19)

Use Case: Preemptive Scheduling

- Previous work proposed to adopt preemptive scheduling to mitigate the head-of-line blocking issue
  - Shinjuku (NSDI'19)
- The original Shinjuku system is built as a specialized OS

Use Case: Preemptive Scheduling

- Previous work proposed to adopt preemptive scheduling to mitigate the head-of-line blocking issue
  - Shinjuku (NSDI'19)
- The original Shinjuku system is built as a specialized OS
- The priority elevation trick allows us to implement preemptive scheduling without changing the kernel

Use Case: Preemptive Scheduling

The head-of-line blocking issue

Use Case: Preemptive Scheduling

The head-of-line blocking issue

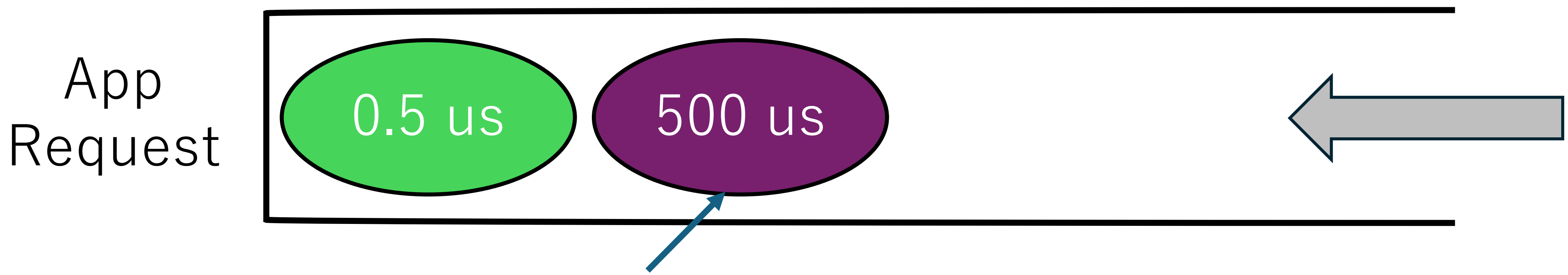
Use Case: Preemptive Scheduling

The head-of-line blocking issue

This request requires 0.5 us to generate a response

Use Case: Preemptive Scheduling

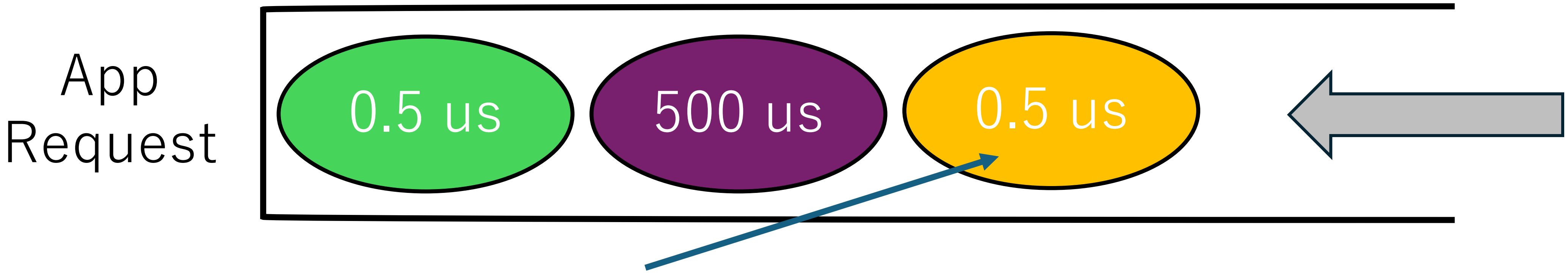
The head-of-line blocking issue



This request requires 500 us to generate a response

Use Case: Preemptive Scheduling

The head-of-line blocking issue

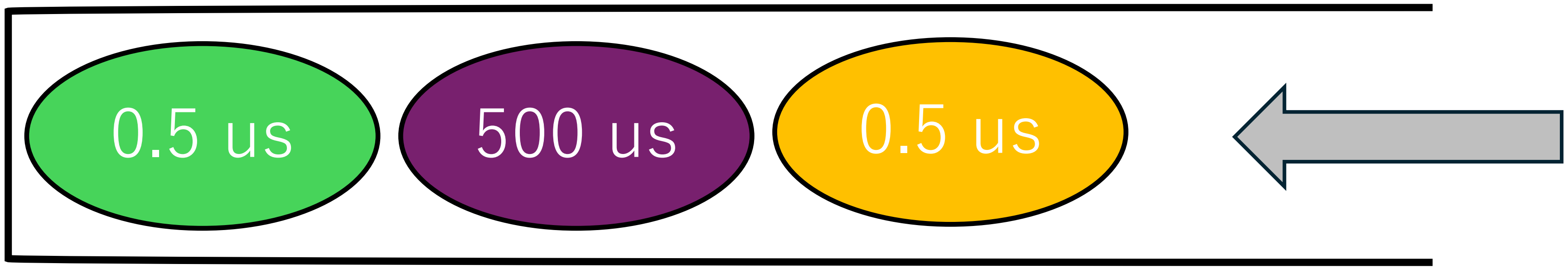


This request requires 0.5 us to generate a response

Use Case: Preemptive Scheduling

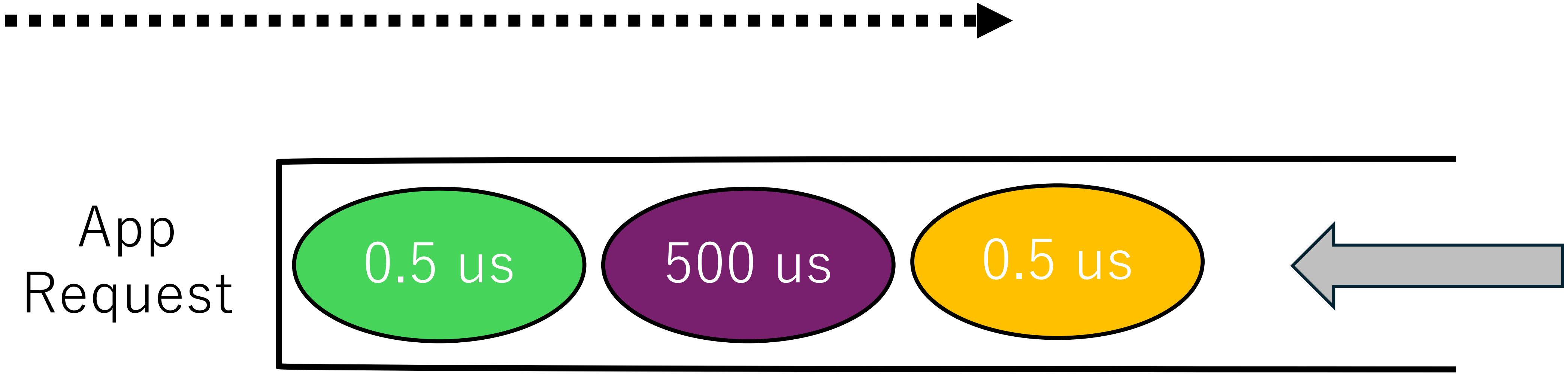
The head-of-line blocking issue

App
Request



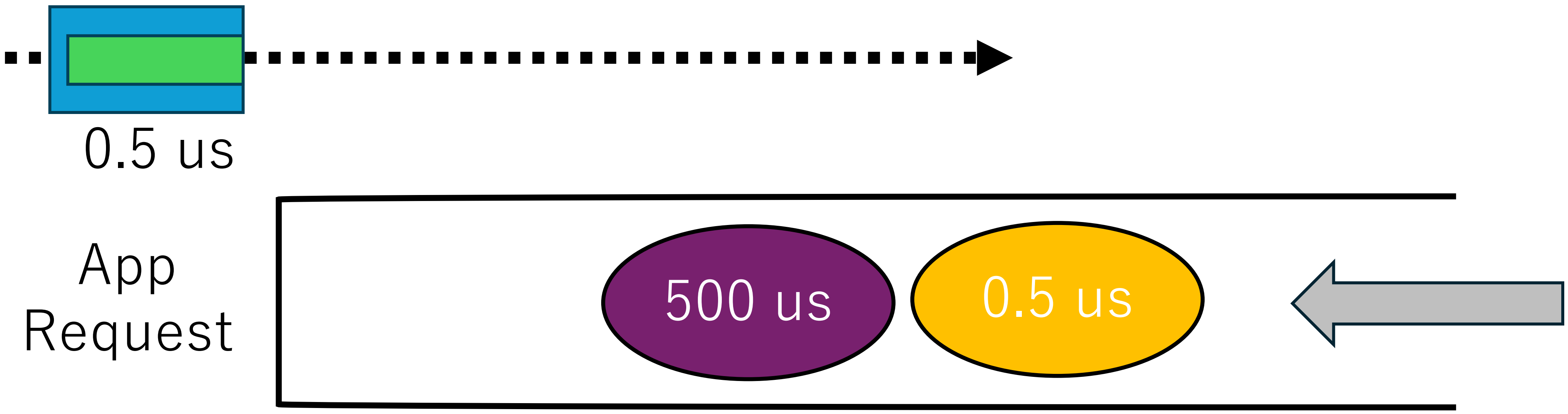
Use Case: Preemptive Scheduling

The head-of-line blocking issue



Use Case: Preemptive Scheduling

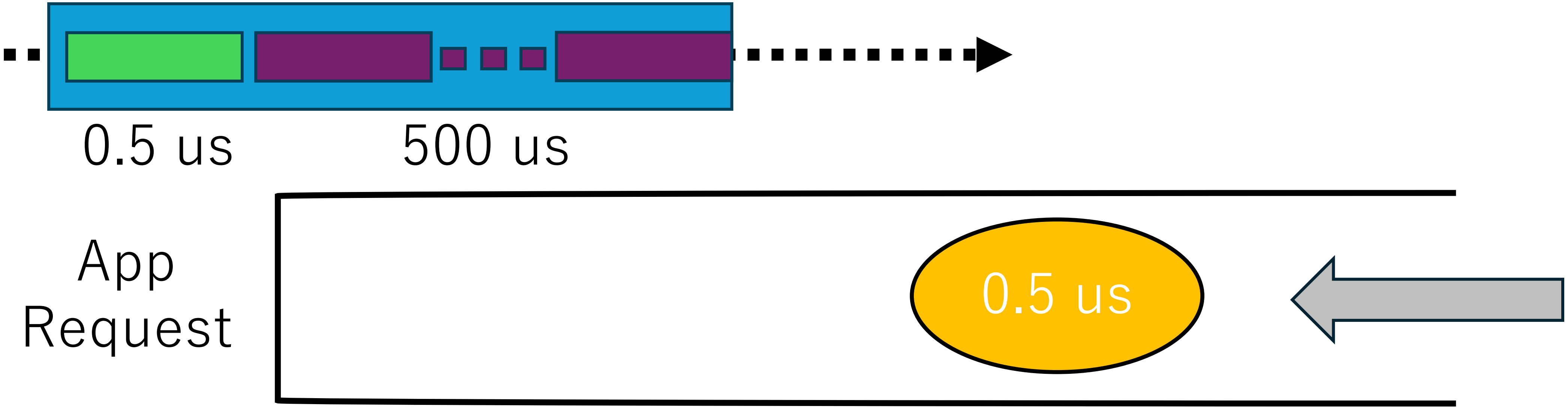
The head-of-line blocking issue



The request is processed in 0.5 us

Use Case: Preemptive Scheduling

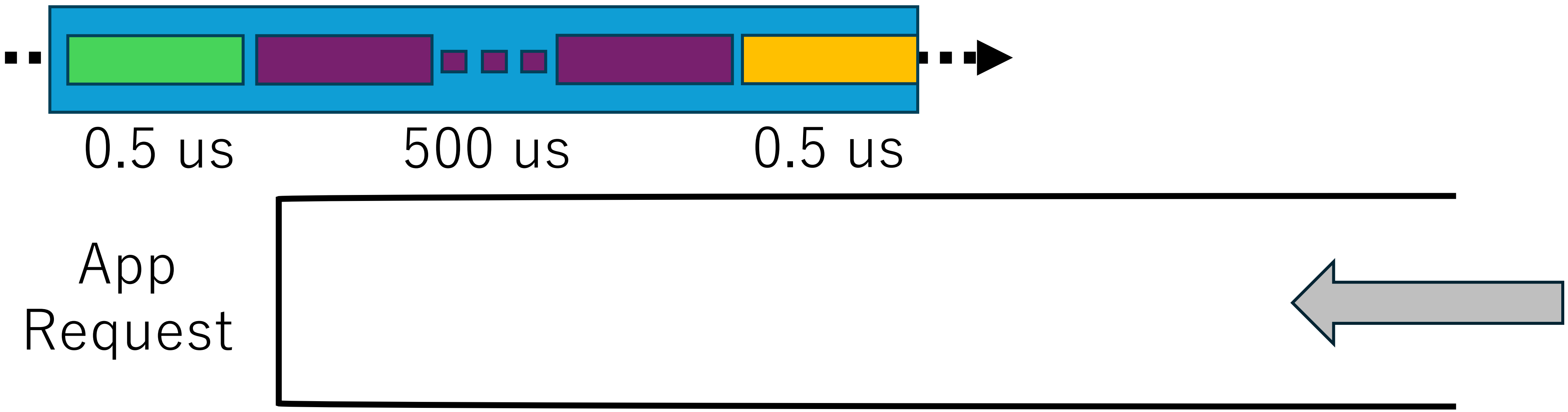
The head-of-line blocking issue



The request is processed in 500 us

Use Case: Preemptive Scheduling

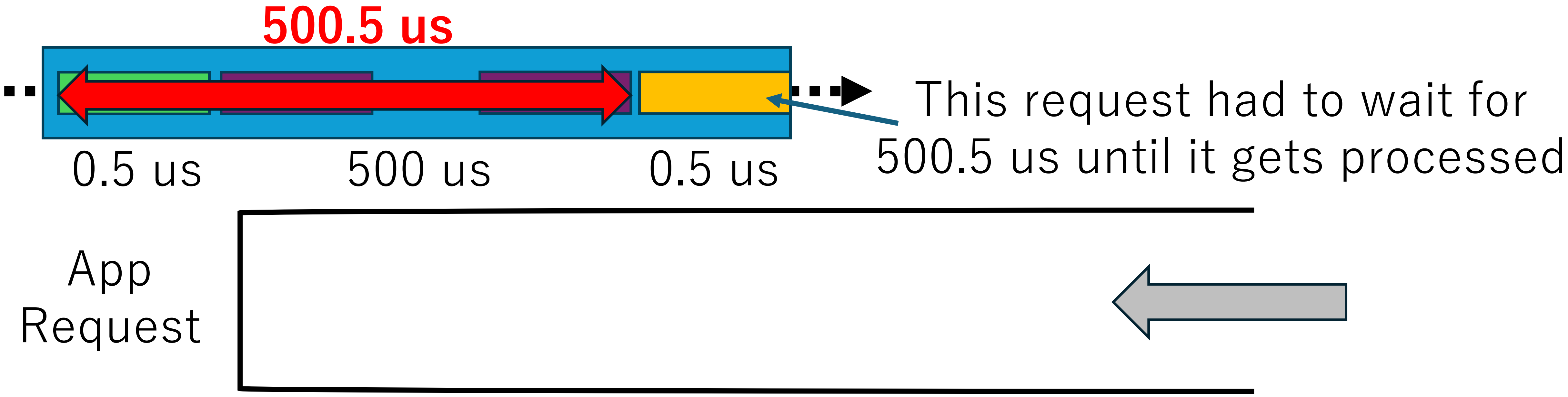
The head-of-line blocking issue



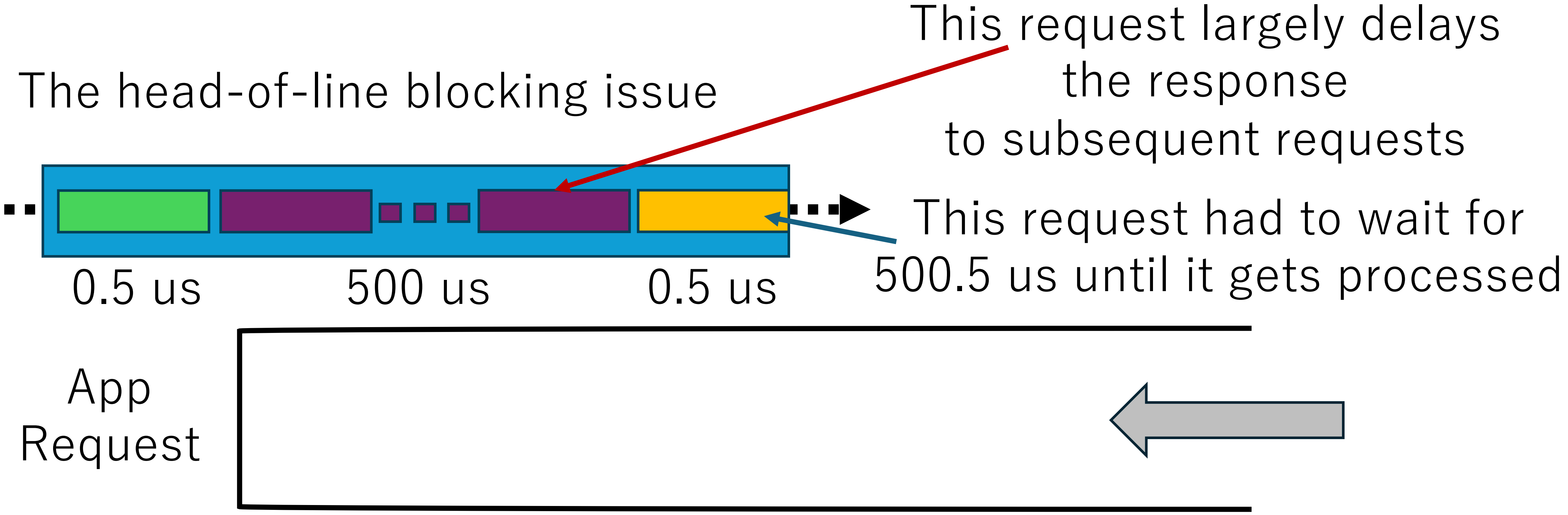
The request is processed in 0.5 us

Use Case: Preemptive Scheduling

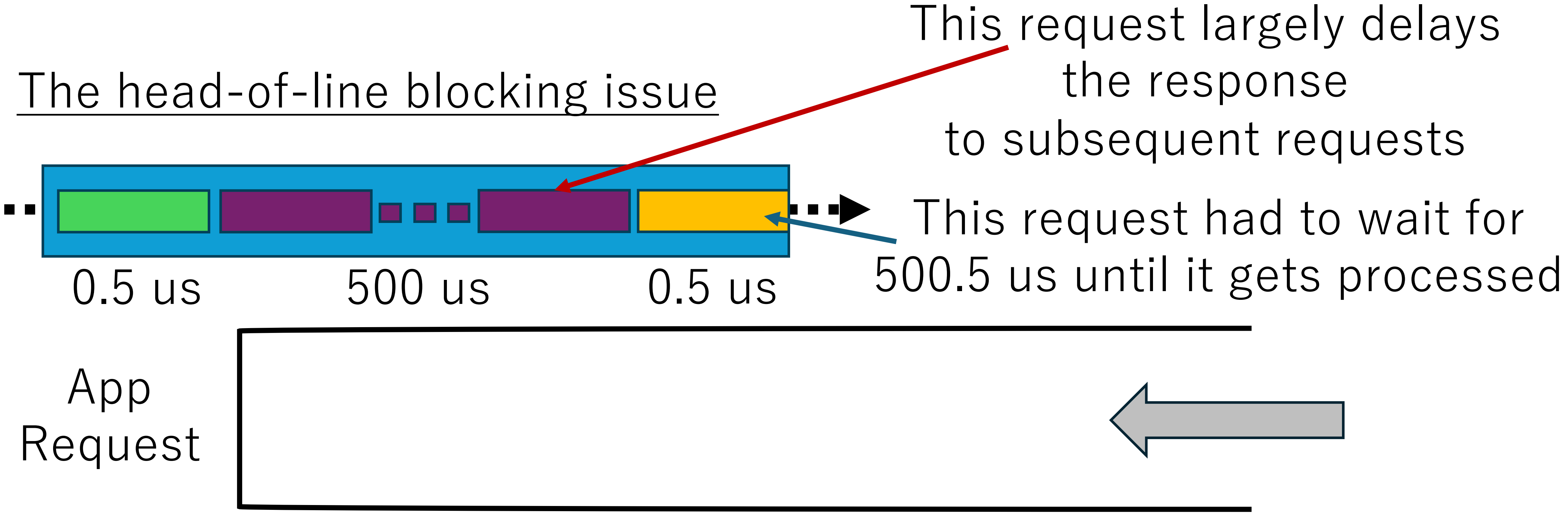
The head-of-line blocking issue



Use Case: Preemptive Scheduling

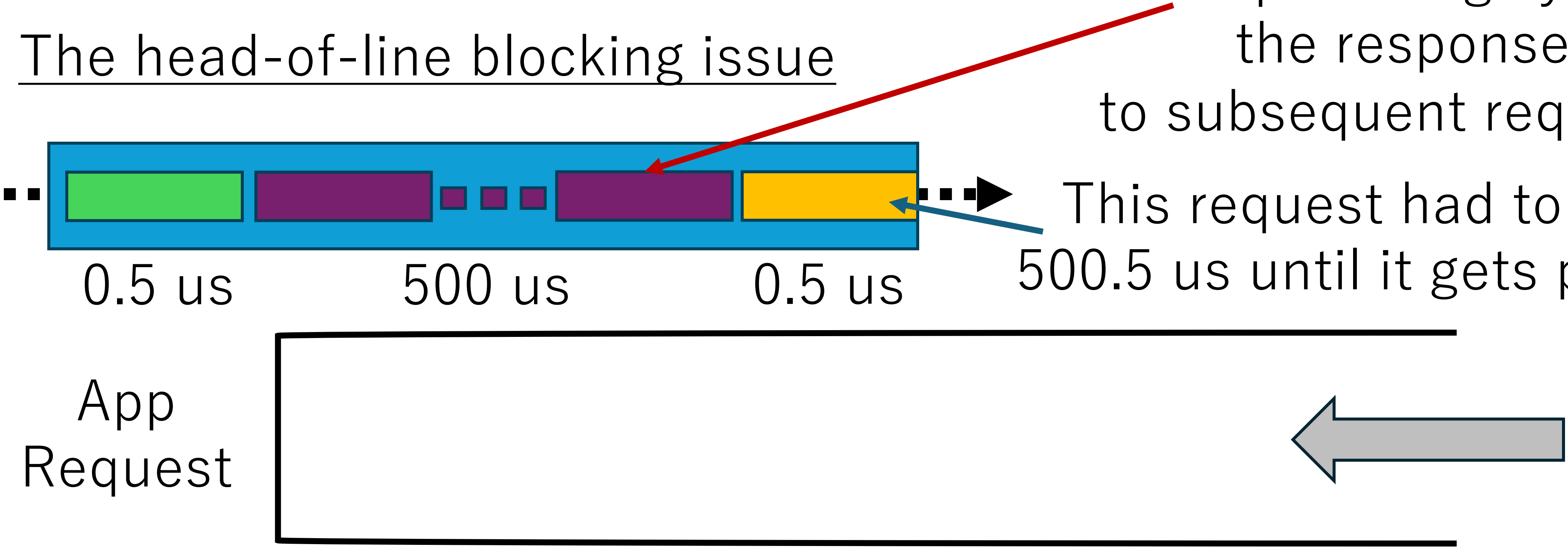


Use Case: Preemptive Scheduling



Use Case: Preemptive Scheduling

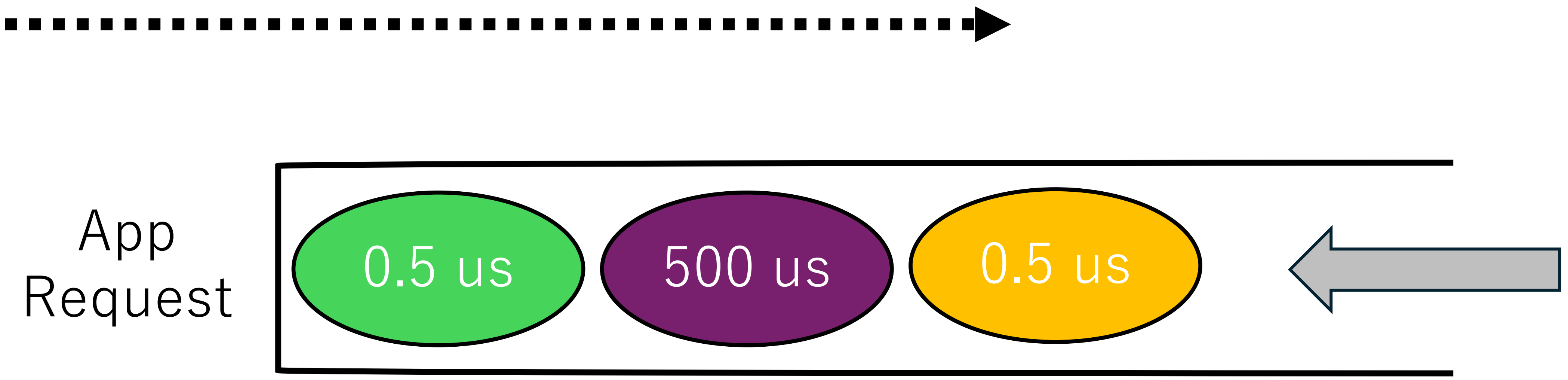
The head-of-line blocking issue



The Shinjuku (NSDI’19) work proposes to adopt preemptive scheduling to mitigate head-of-line blocking

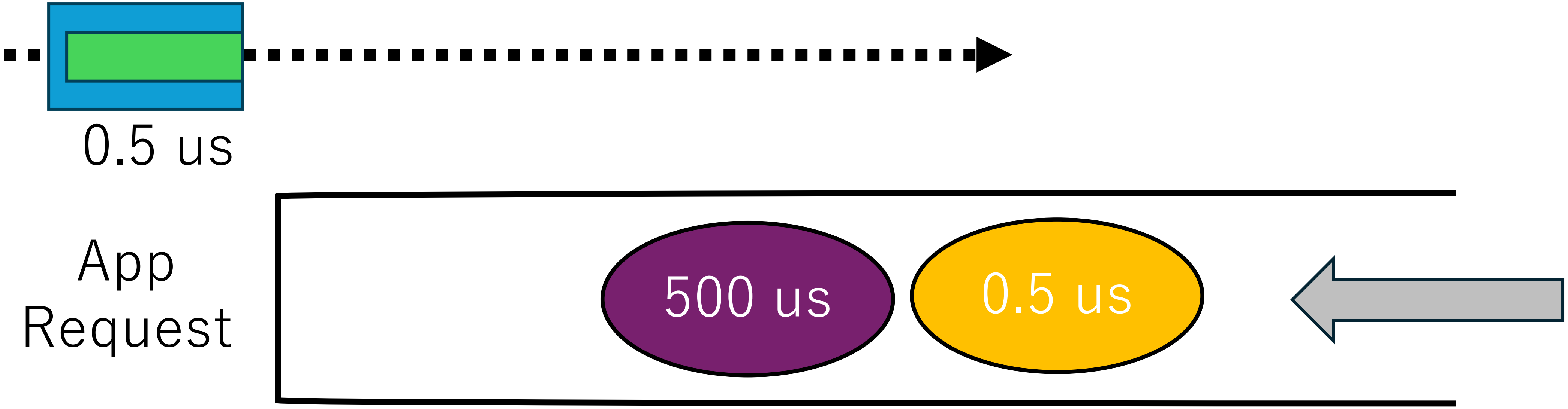
Use Case: Preemptive Scheduling

Preemptive scheduling



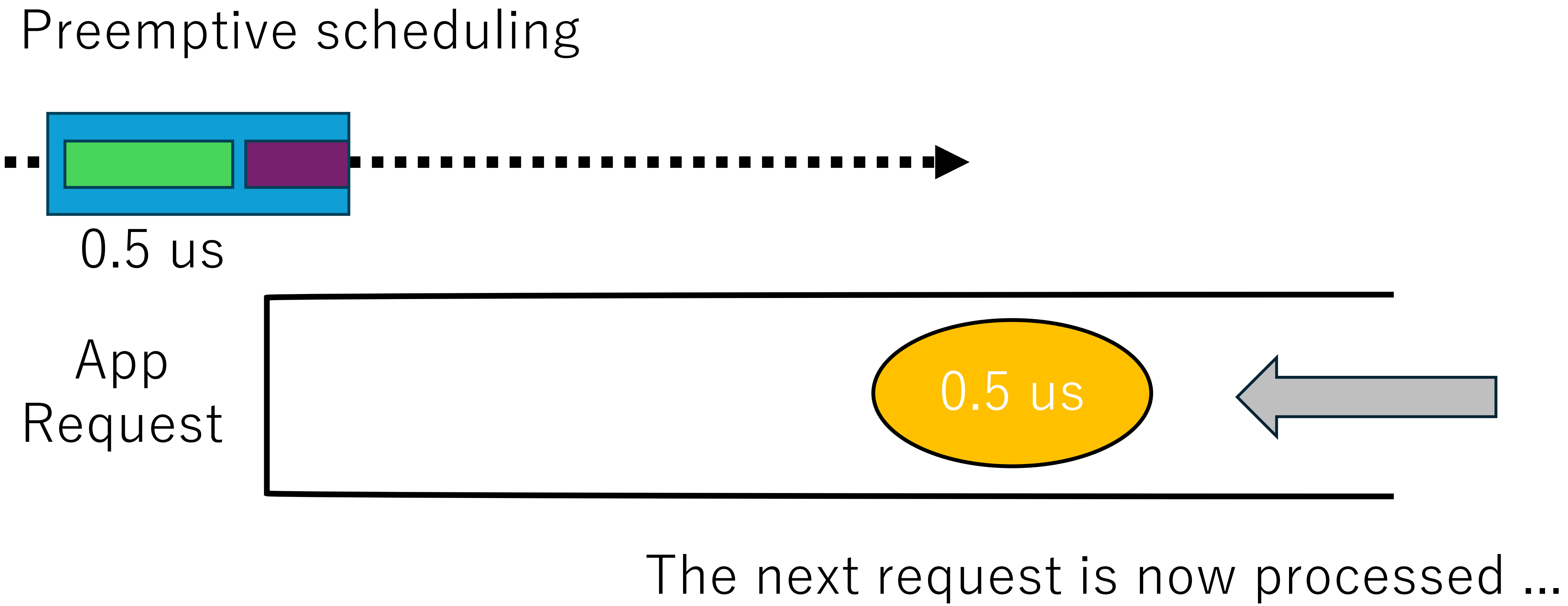
Use Case: Preemptive Scheduling

Preemptive scheduling

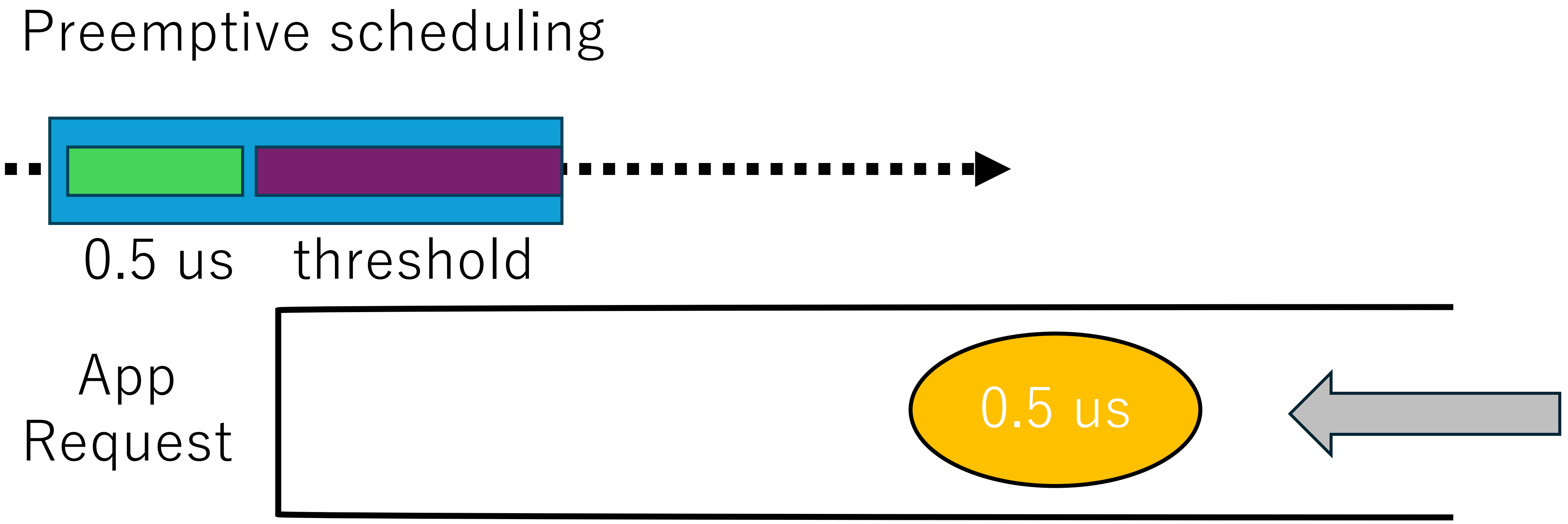


The request is processed in 0.5 us

Use Case: Preemptive Scheduling

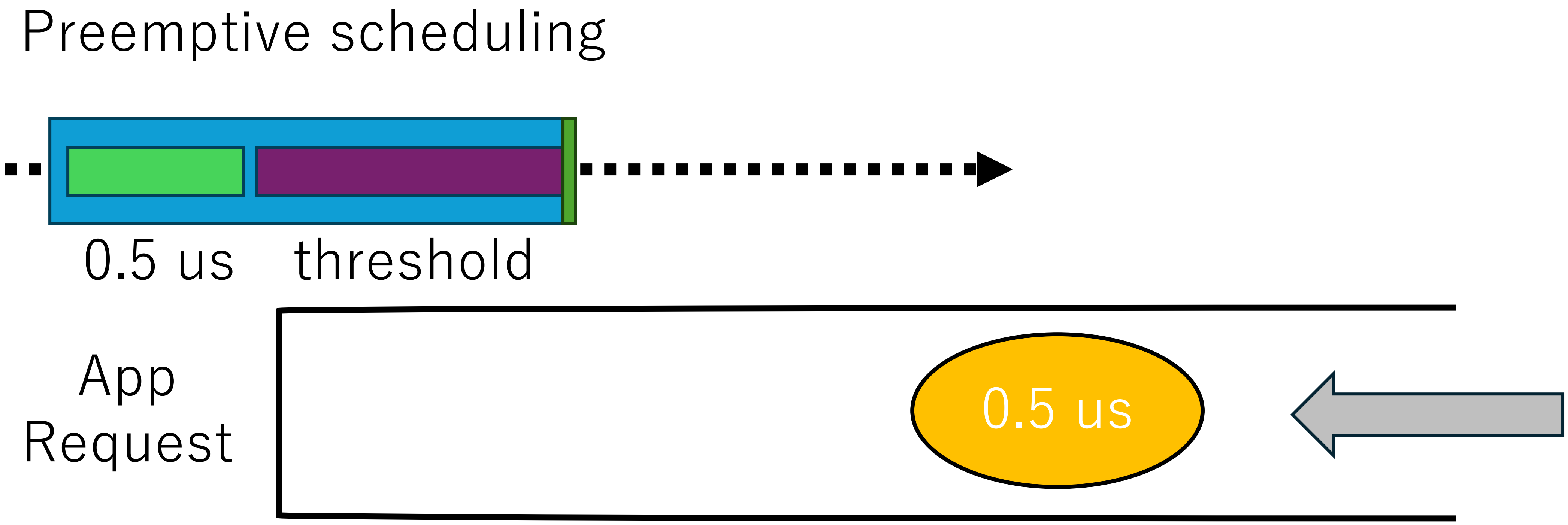


Use Case: Preemptive Scheduling



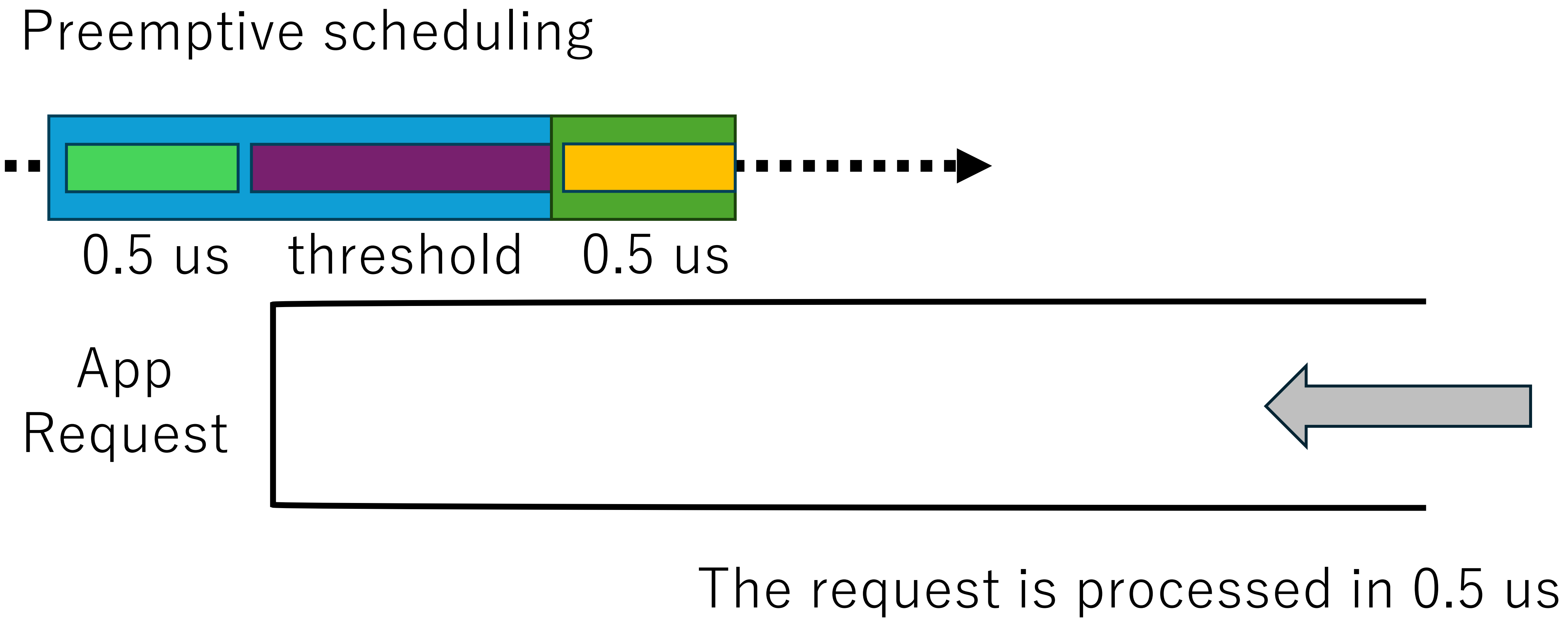
The request processing has not been finished within a preconfigured threshold

Use Case: Preemptive Scheduling



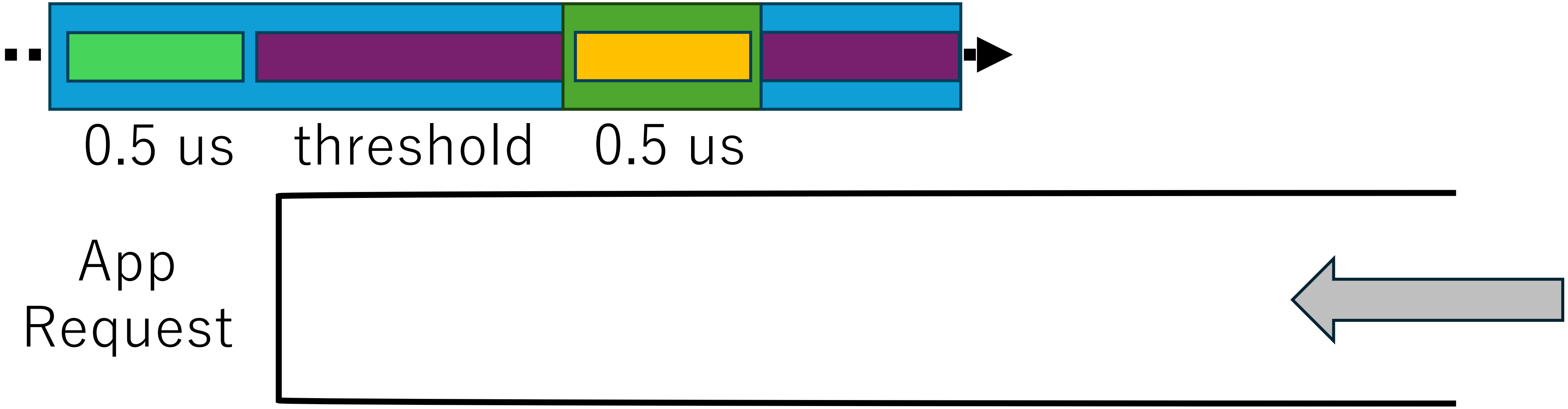
The preemptive scheduling policy preempts the currently running worker and runs another worker to process the subsequent request

Use Case: Preemptive Scheduling



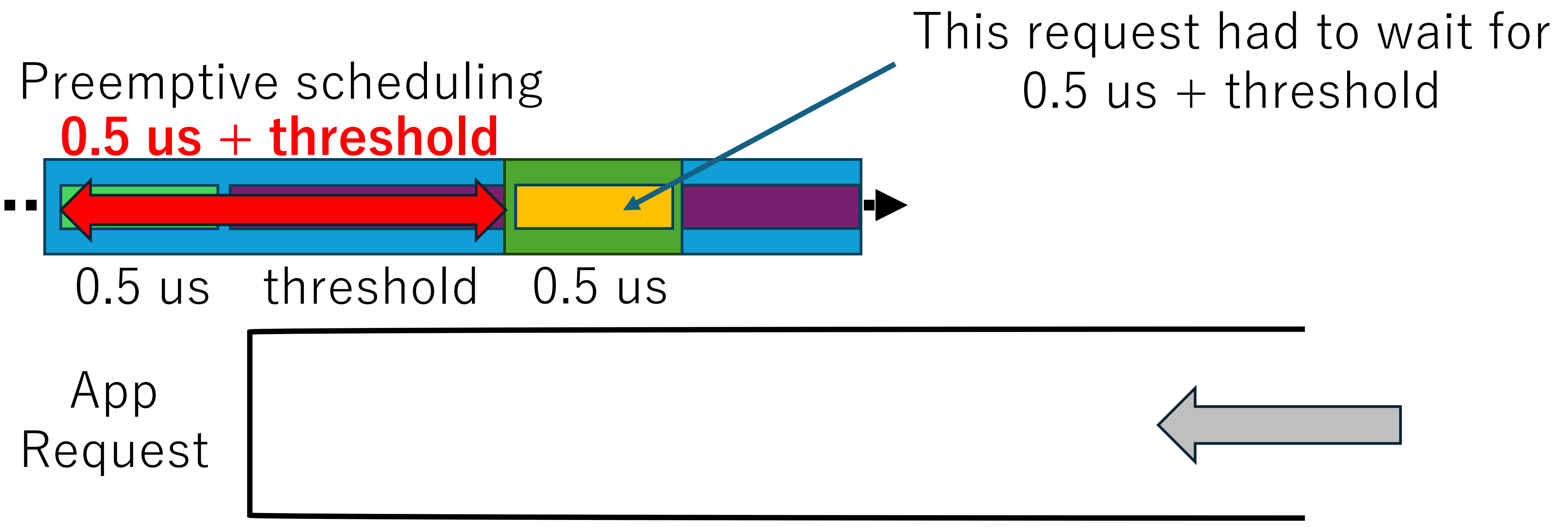
Use Case: Preemptive Scheduling

Preemptive scheduling



Then, the preempted worker is resumed

Use Case: Preemptive Scheduling

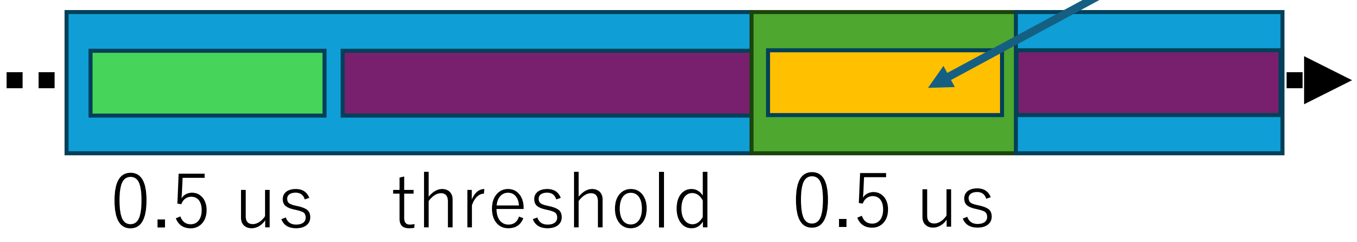


Then, the preempted worker is resumed

Use Case: Preemptive Scheduling

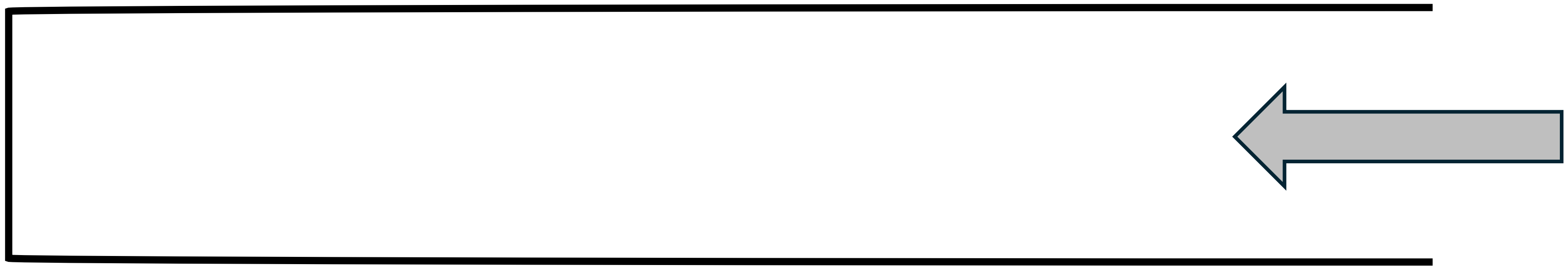
Preemptive scheduling

This request had to wait for
0.5 us + threshold



Head-of-line blocking
is mitigated

App
Request

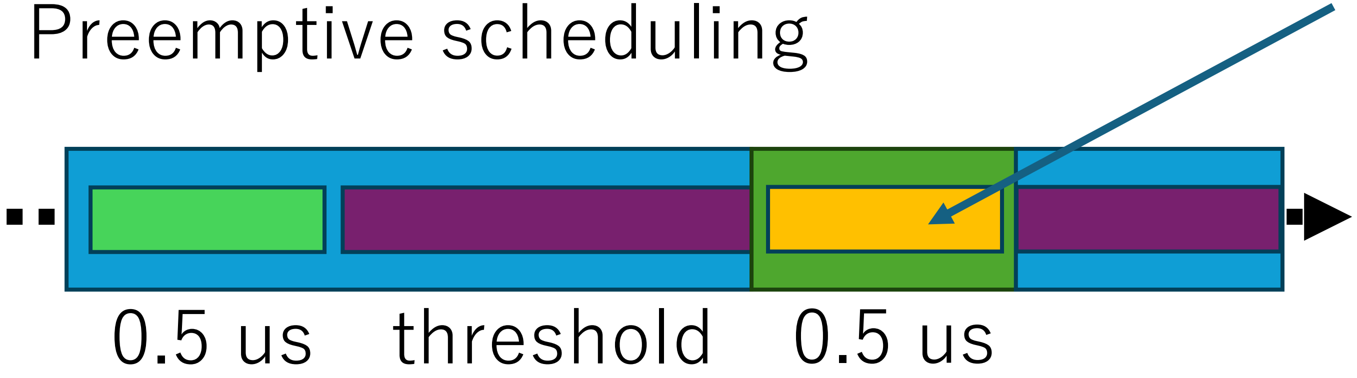


Then, the preempted worker is resumed

Use Case: Preemptive Scheduling

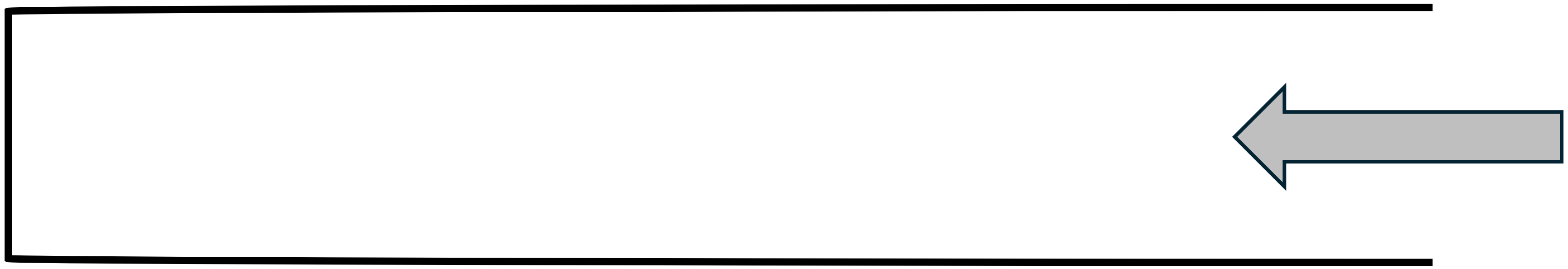
Preemptive scheduling

This request had to wait for
0.5 us + threshold



Head-of-line blocking
is mitigated

App
Request



Then, the preempted worker is resumed

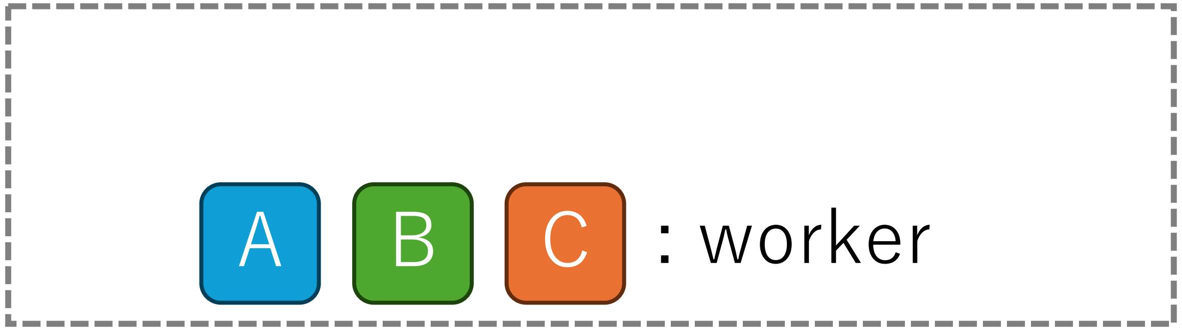
We implement an equivalent mechanism with the priority elevation trick

Use Case: Preemptive Scheduling

- Pthread types

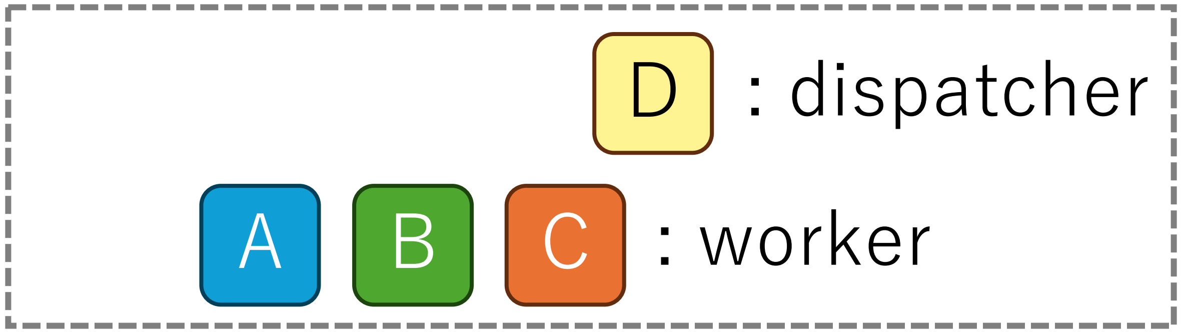
Use Case: Preemptive Scheduling

- Pthread types
  - A worker pthreads handle application-level requests



Use Case: Preemptive Scheduling

- Pthread types
  - A worker pthreads handle application-level requests
  - A dispatcher pthread
    - extracts application-level requests by performing TCP/IP processing for incoming packets
    - dispatches the requests to worker pthreads



Use Case: Preemptive Scheduling

- Pthread types
  - A worker pthreads handle application-level requests
  - A dispatcher pthread
    - extracts application-level requests by performing TCP/IP processing for incoming packets
    - dispatches the requests to worker pthreads
  - A switcher pthread preempts a worker pthread that continuously runs longer than a preconfigured threshold

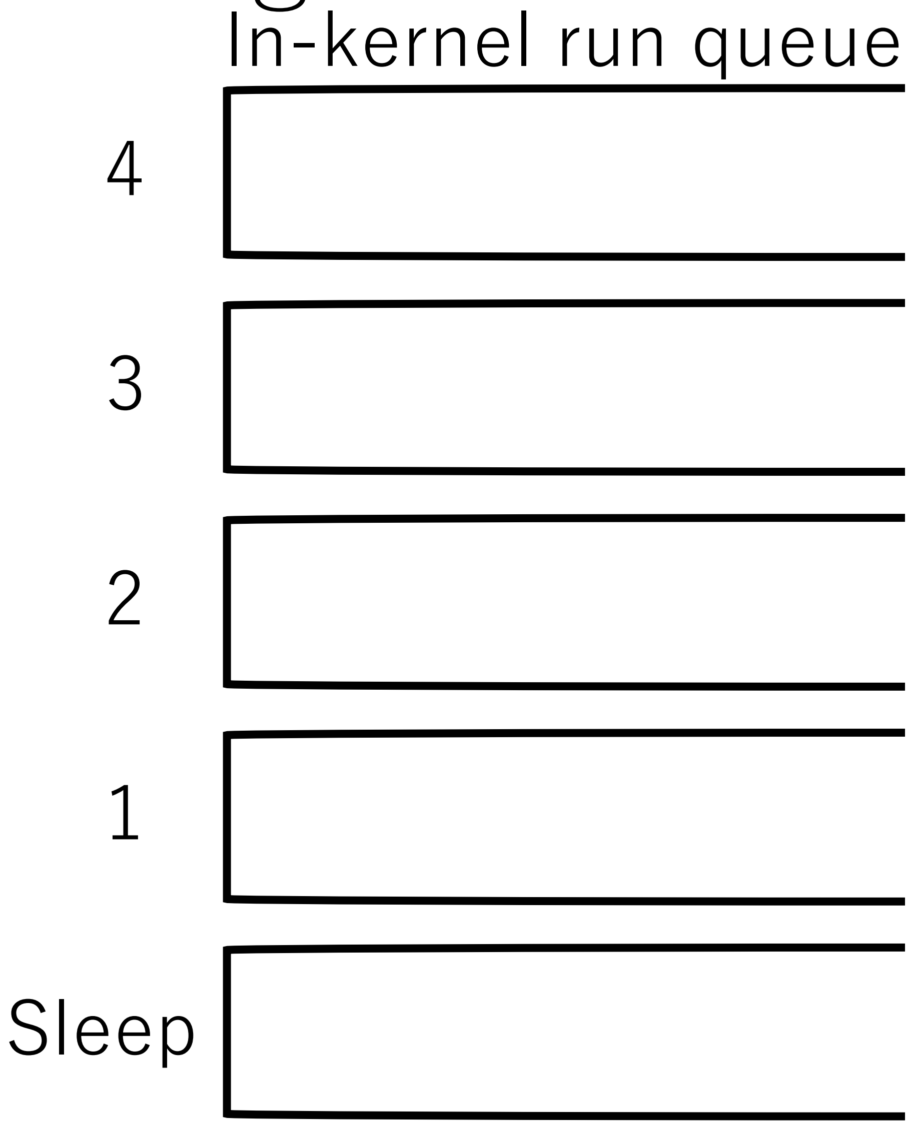
S : switcher D : dispatcher

A B C : worker

Use Case: Preemptive Scheduling

- Priority values
  - 4:
  - 3:
  - 2:
  - 1:

  (a higher value represents a higher priority)



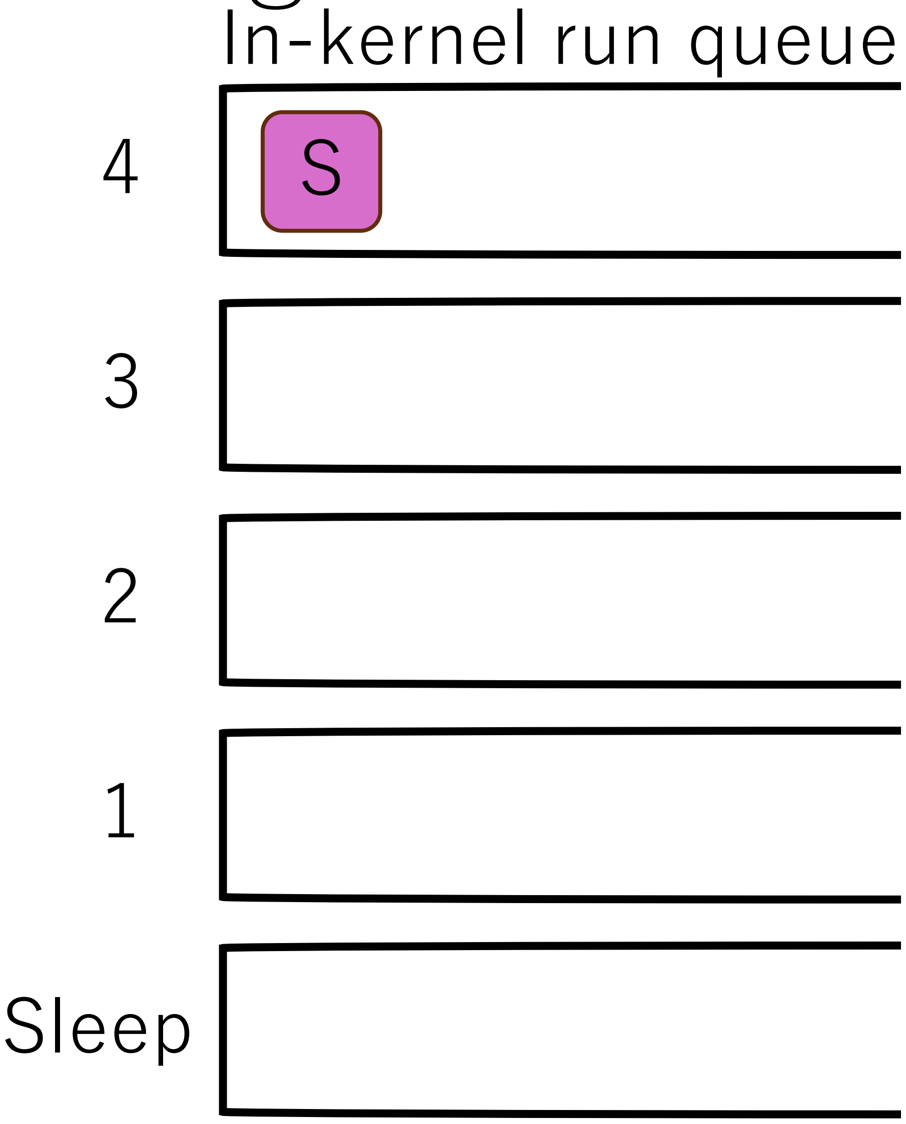
S : switcher D : dispatcher

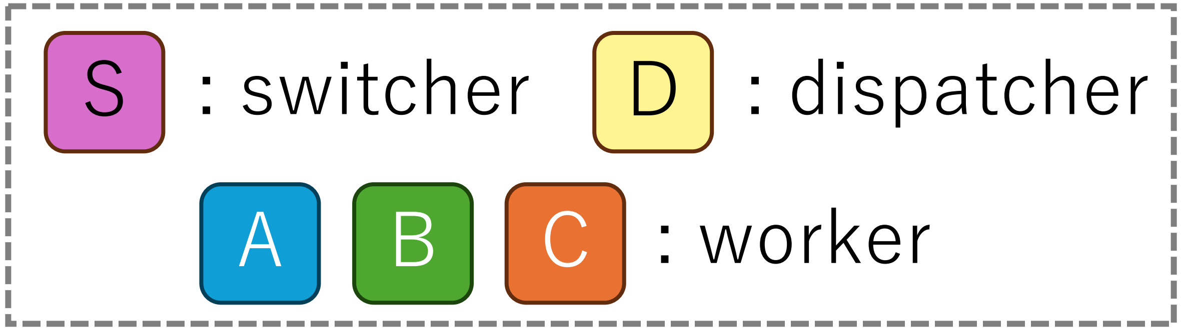
A B C : worker

Use Case: Preemptive Scheduling

- Priority values
  - 4: the switcher
  - 3:
  - 2:
  - 1:

  (a higher value represents a higher priority)

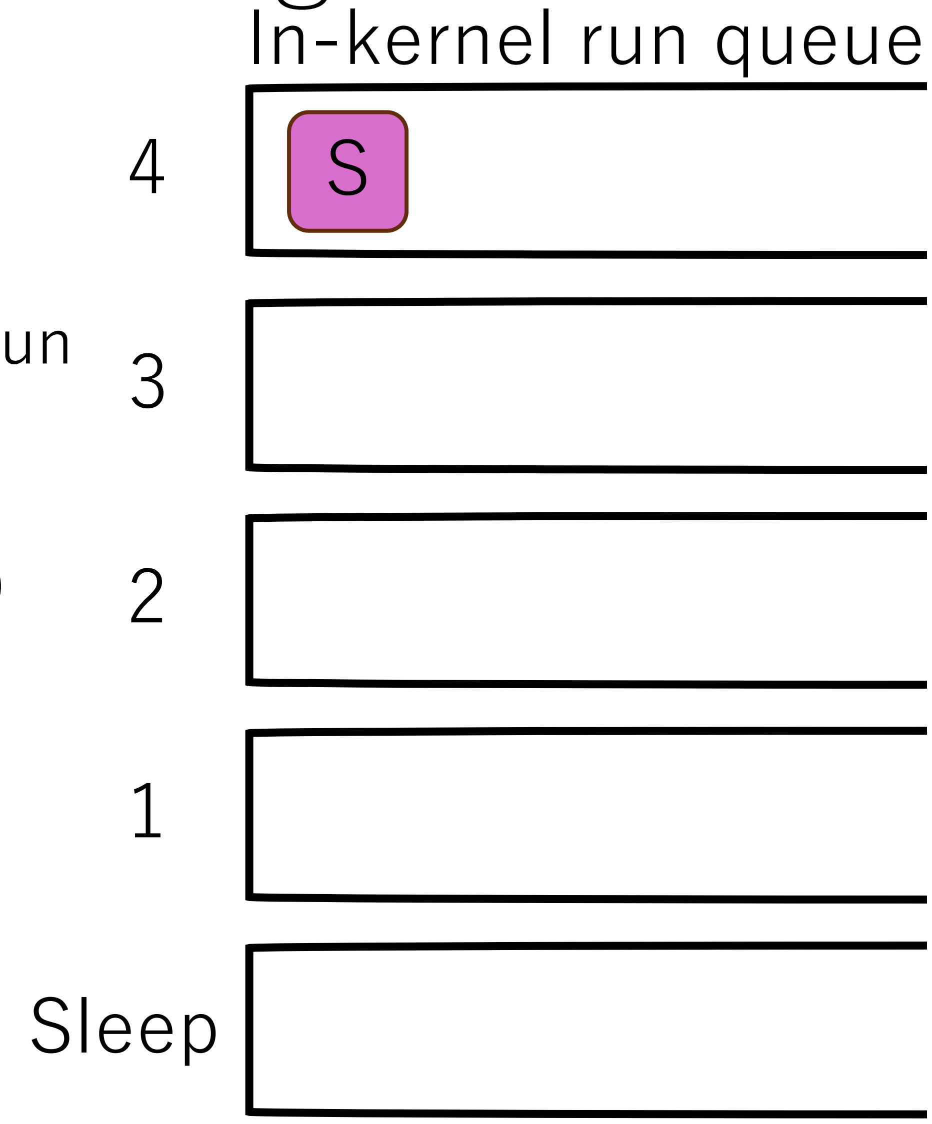




Use Case: Preemptive Scheduling

- Priority values
  - 4: the switcher
  - 3: a worker allowed by the dispatcher to run
  - 2:
  - 1:

  (a higher value represents a higher priority)

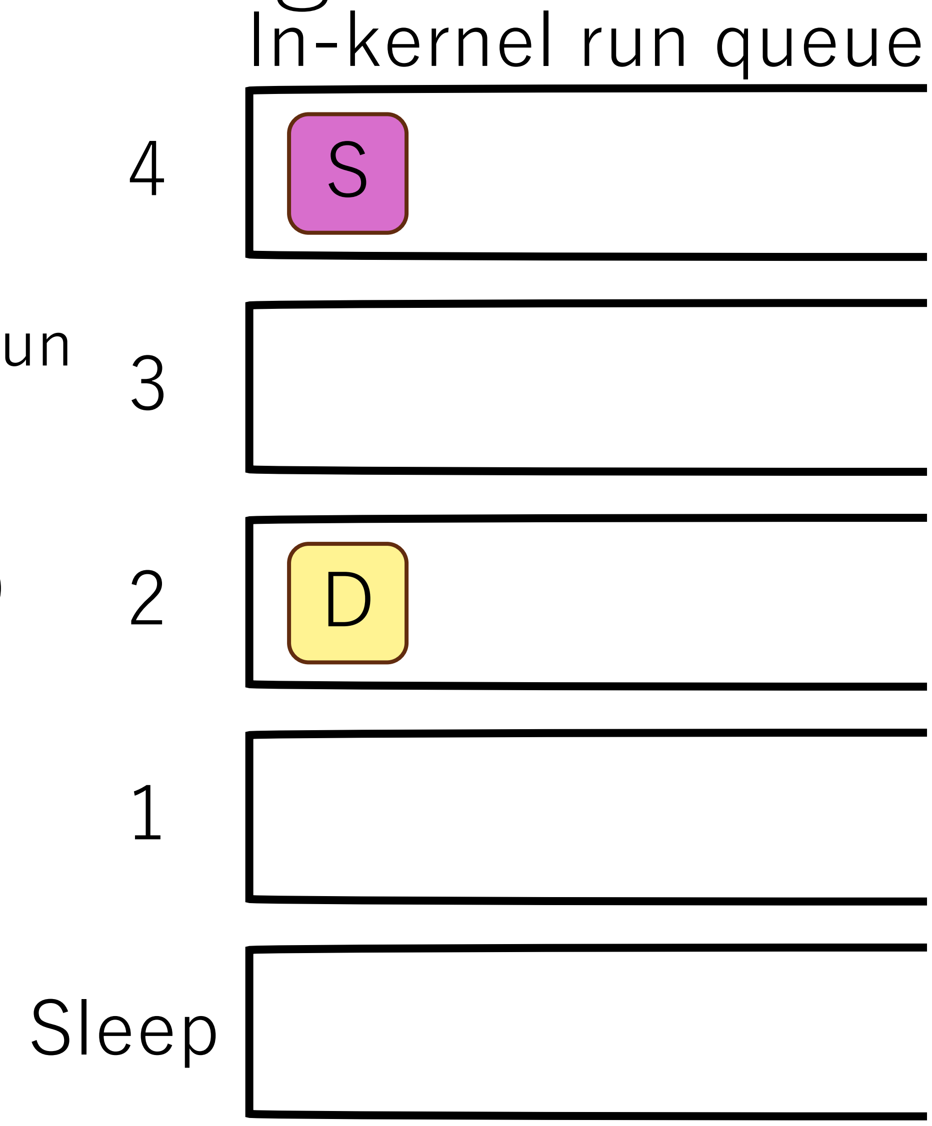


S : switcher D : dispatcher
A B C : worker

Use Case: Preemptive Scheduling

- Priority values
  - 4: the switcher
  - 3: a worker allowed by the dispatcher to run
  - 2: the dispatcher
  - 1:

  (a higher value represents a higher priority)



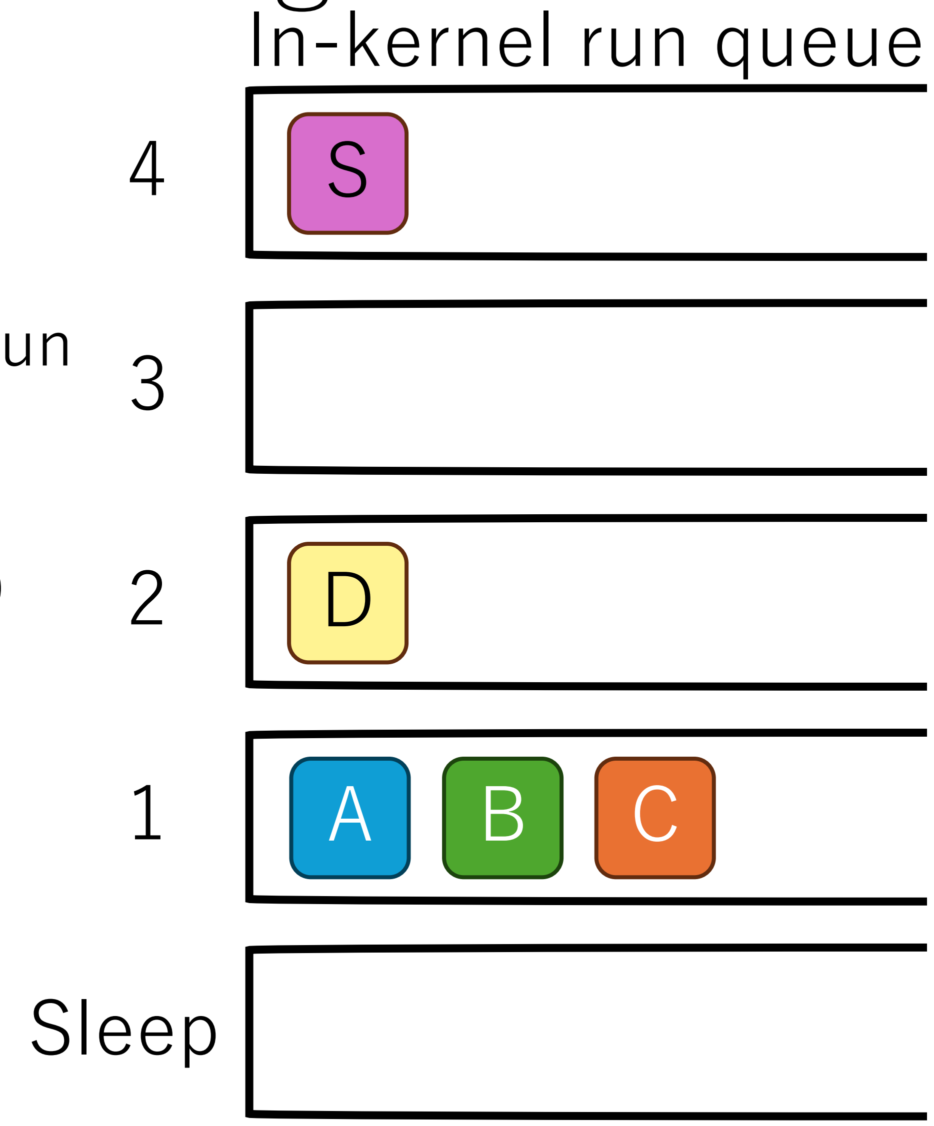
S : switcher D : dispatcher

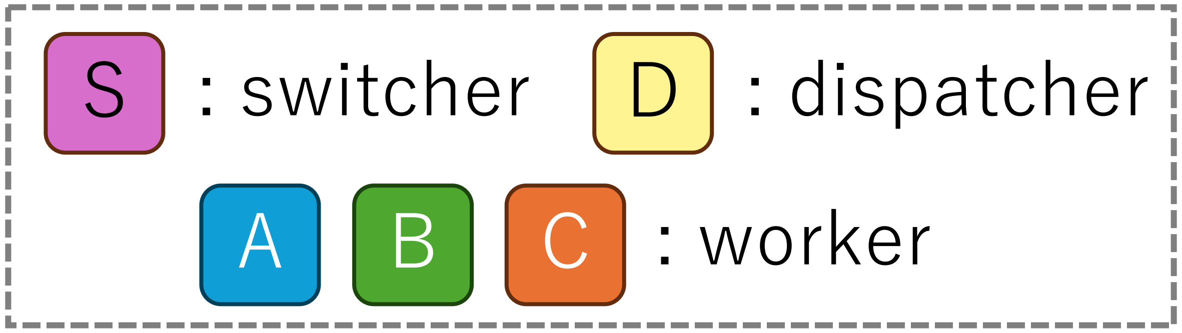
A B C : worker

Use Case: Preemptive Scheduling

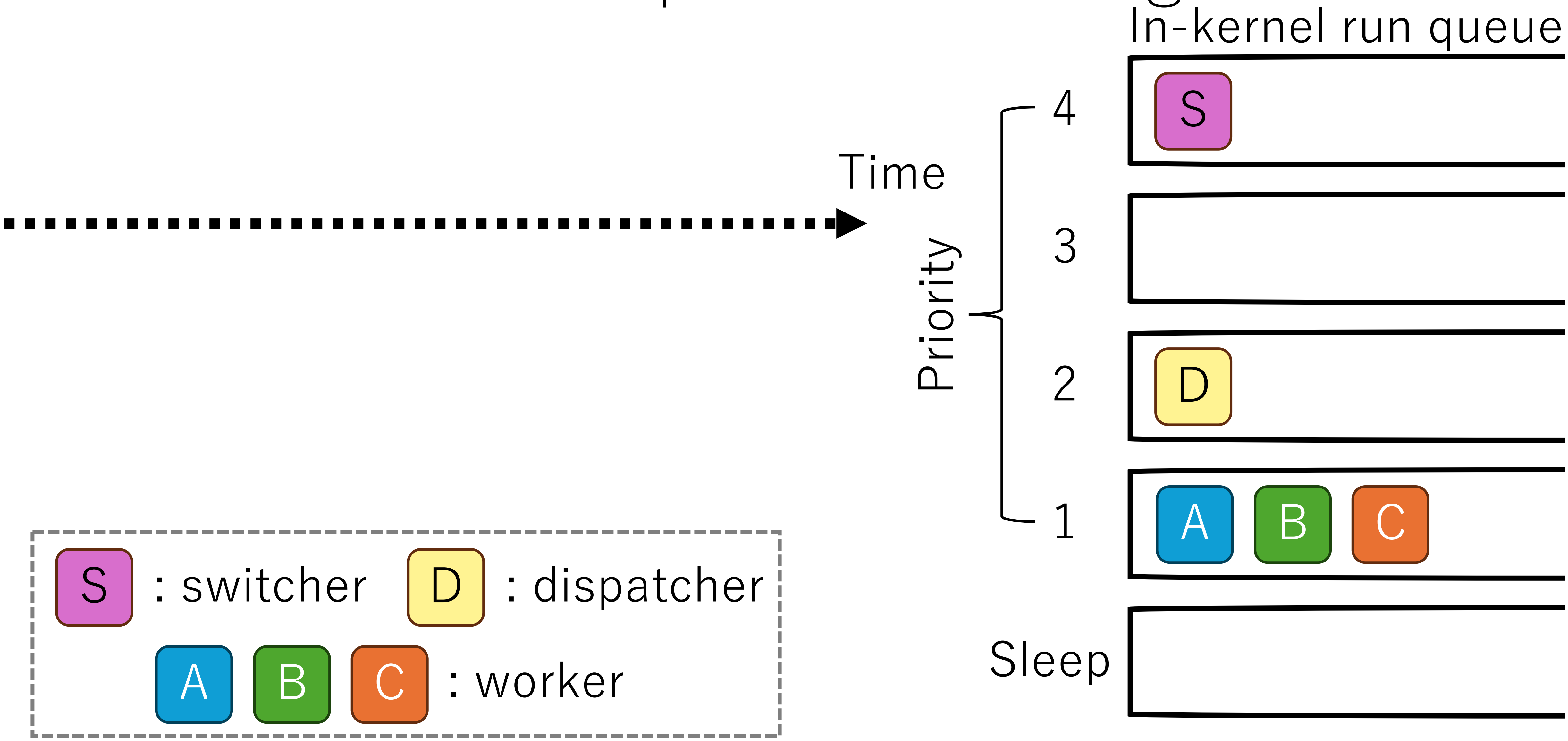
- Priority values
  - 4: the switcher
  - 3: a worker allowed by the dispatcher to run
  - 2: the dispatcher
  - 1: workers that are not allowed to run

  (a higher value represents a higher priority)

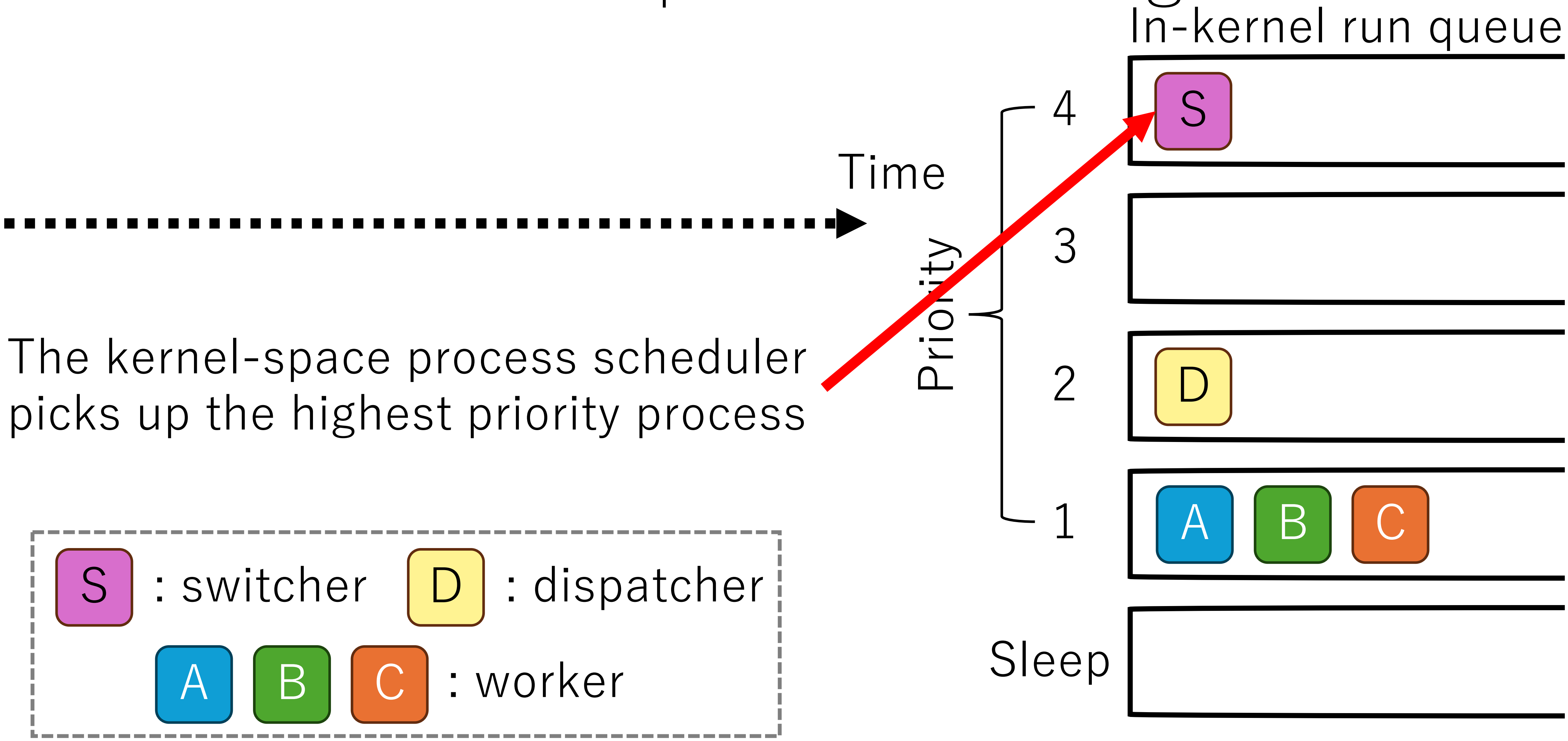




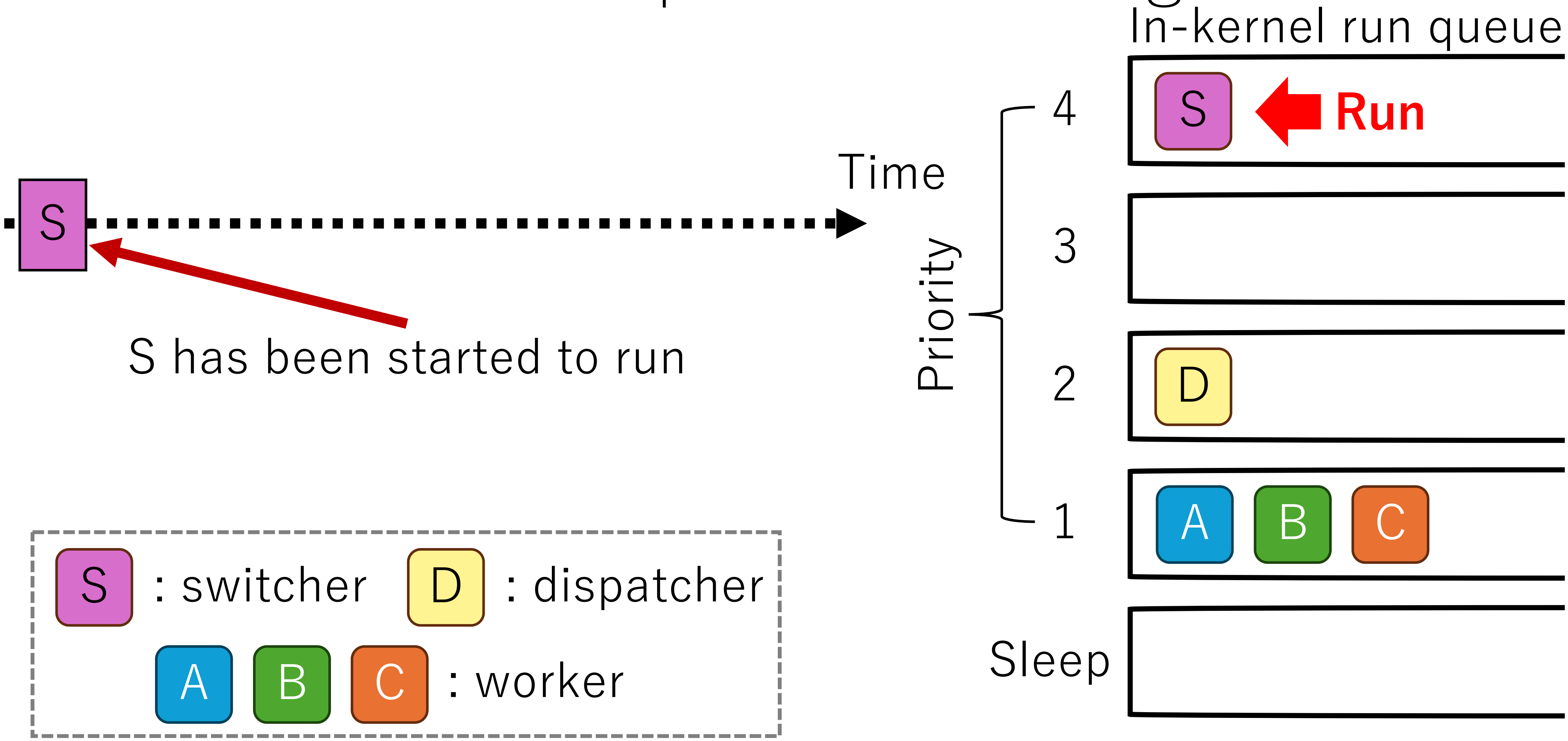
Use Case: Preemptive Scheduling



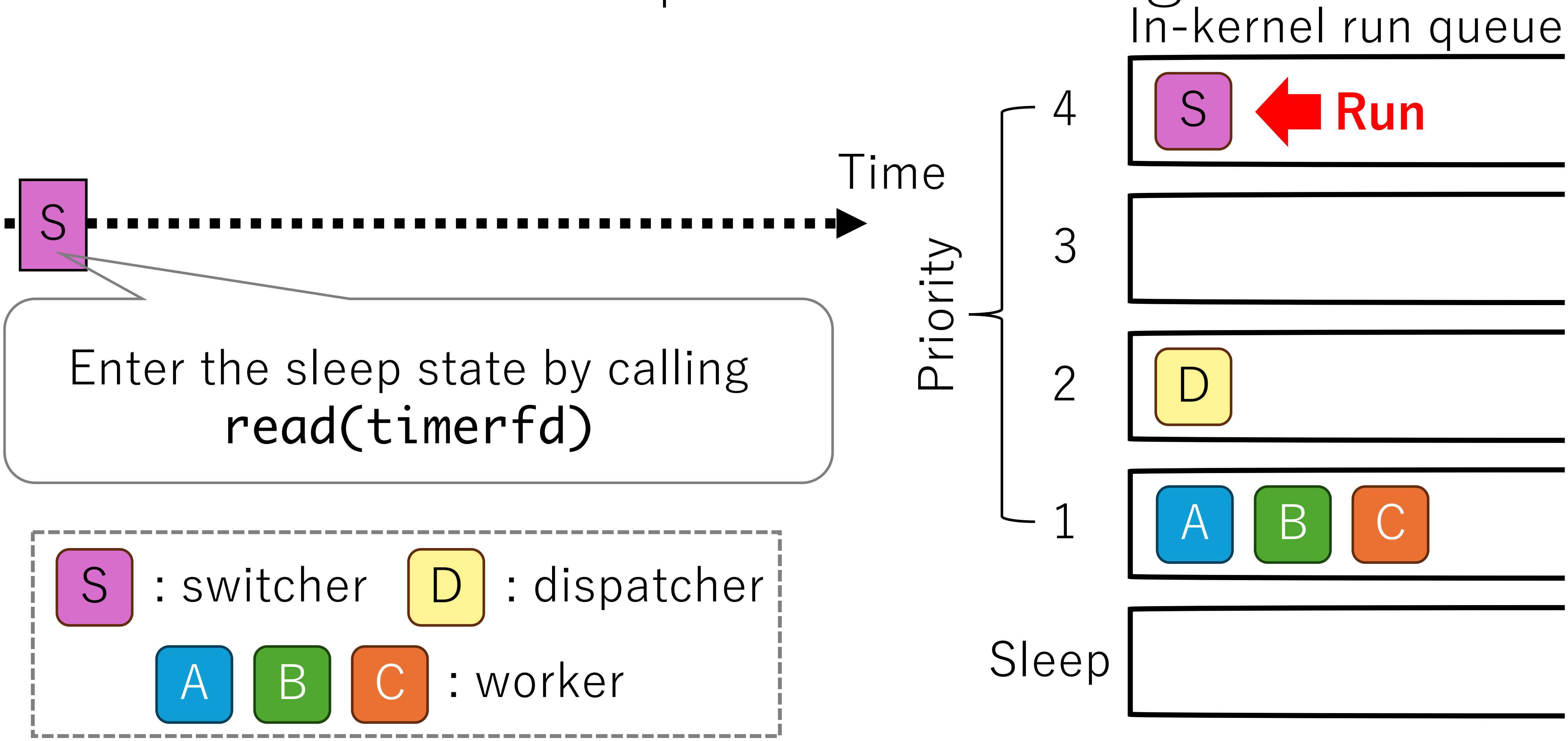
Use Case: Preemptive Scheduling



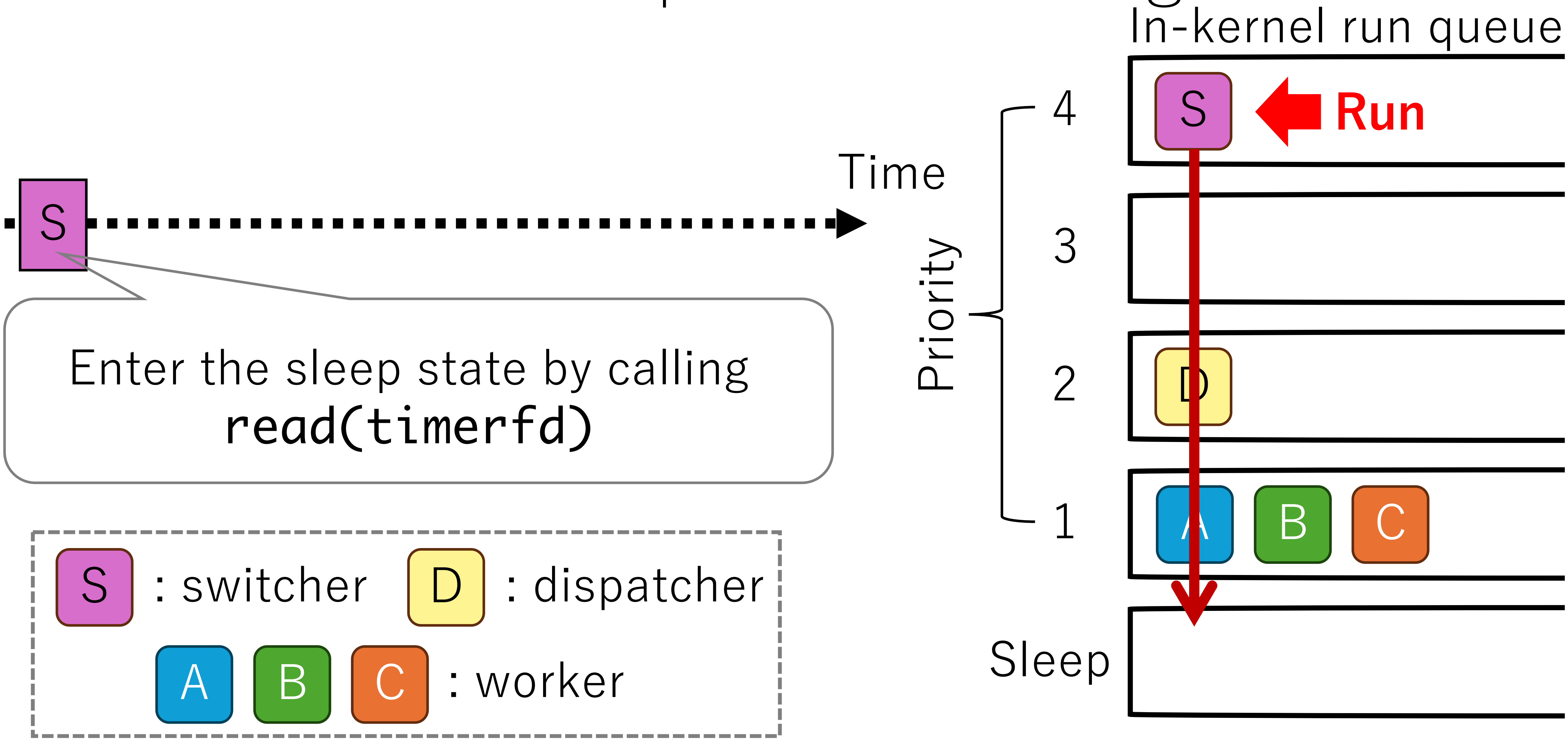
Use Case: Preemptive Scheduling



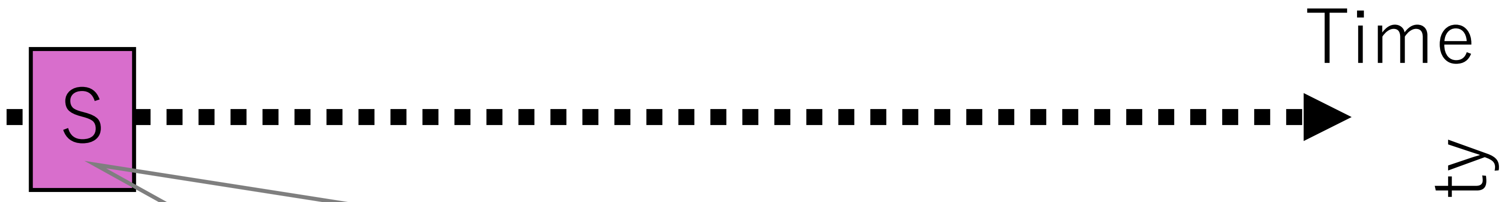
Use Case: Preemptive Scheduling

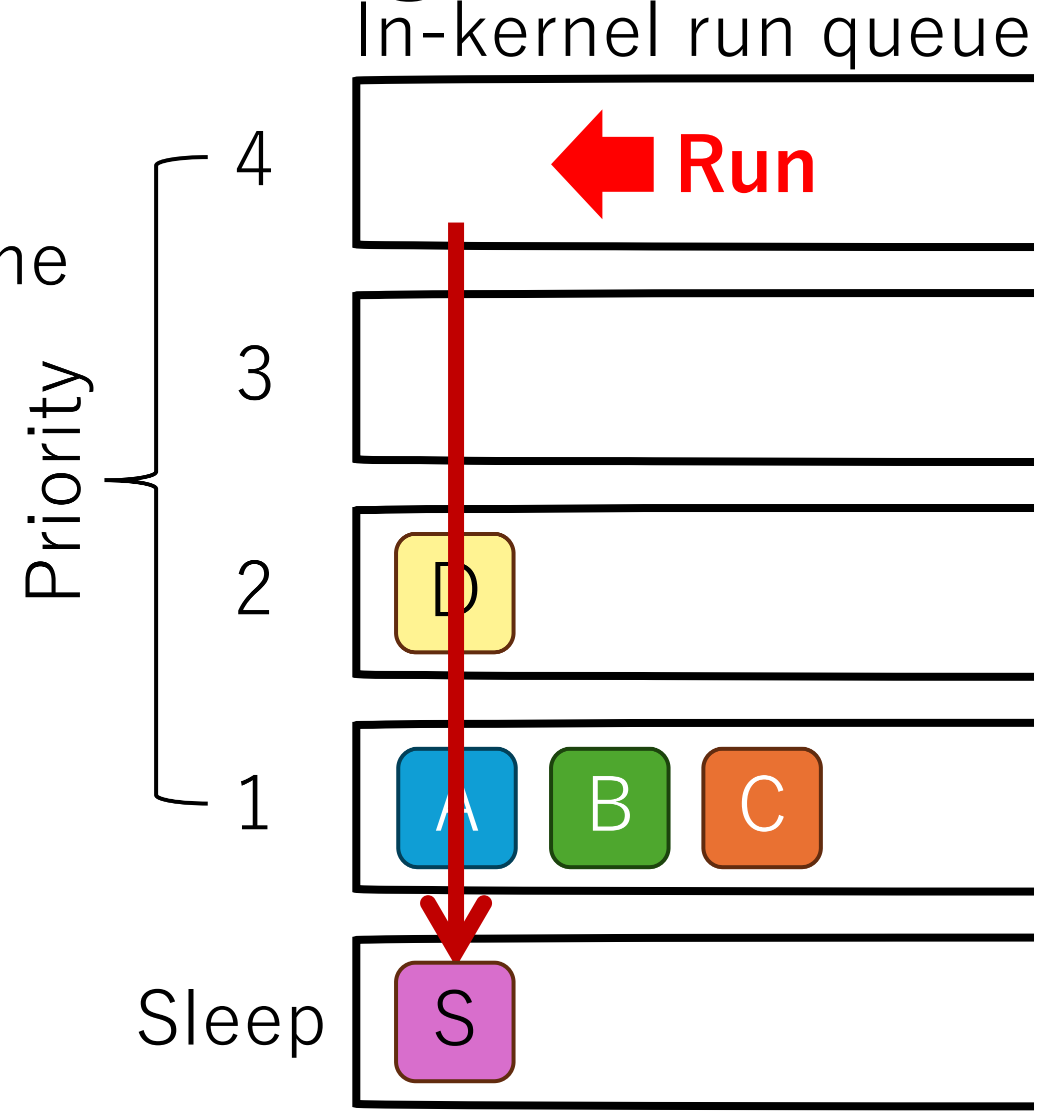


Use Case: Preemptive Scheduling

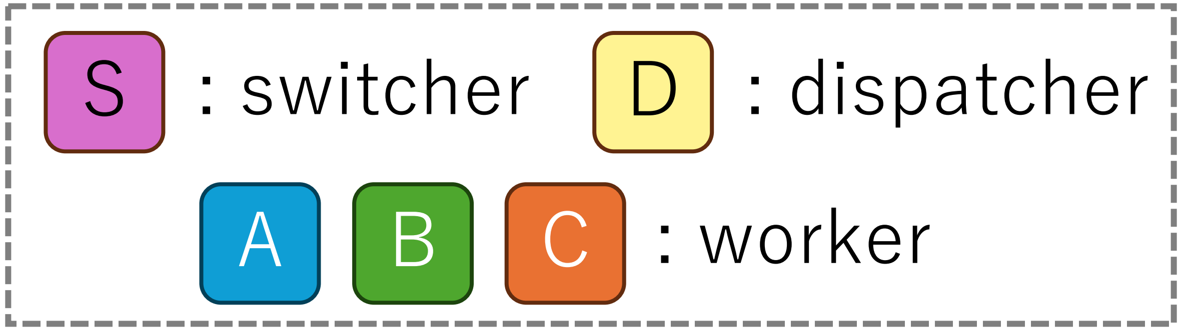


Use Case: Preemptive Scheduling

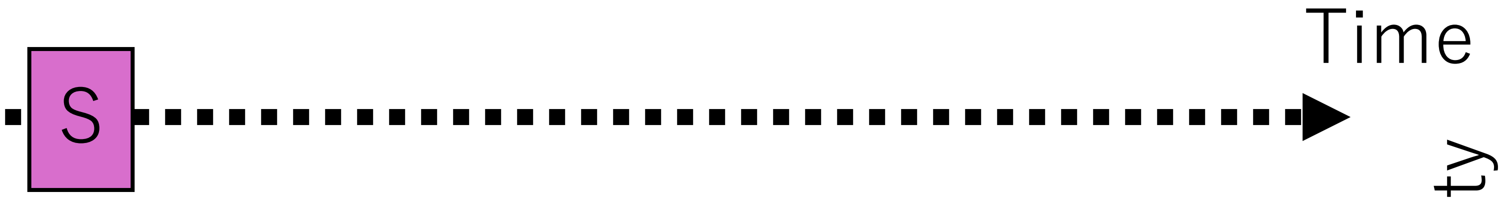


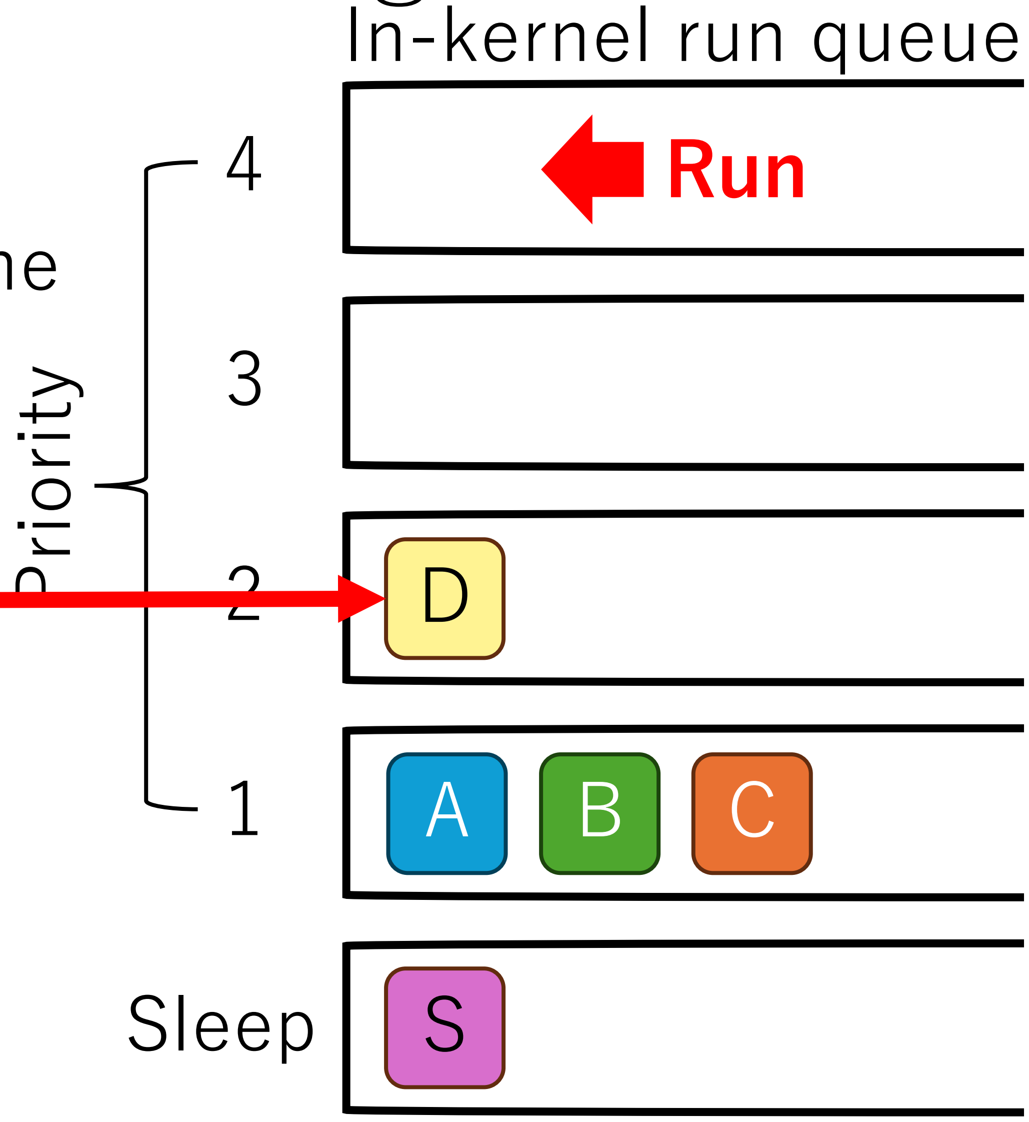


Enter the sleep state by calling
`read(timerfd)`

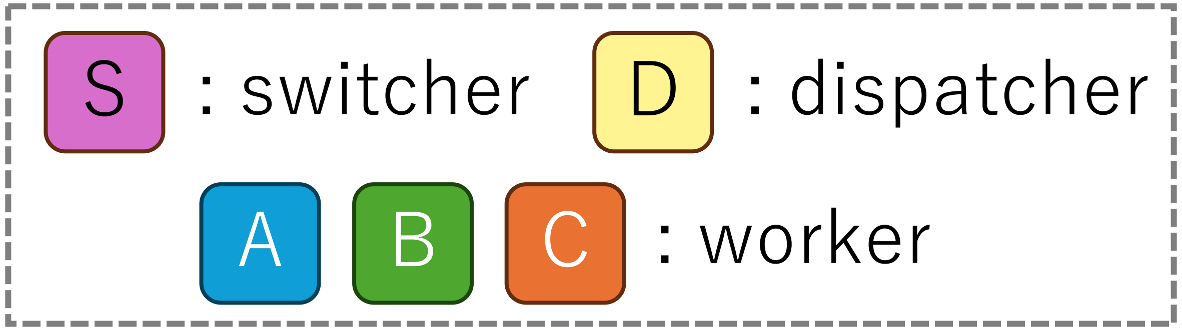


Use Case: Preemptive Scheduling

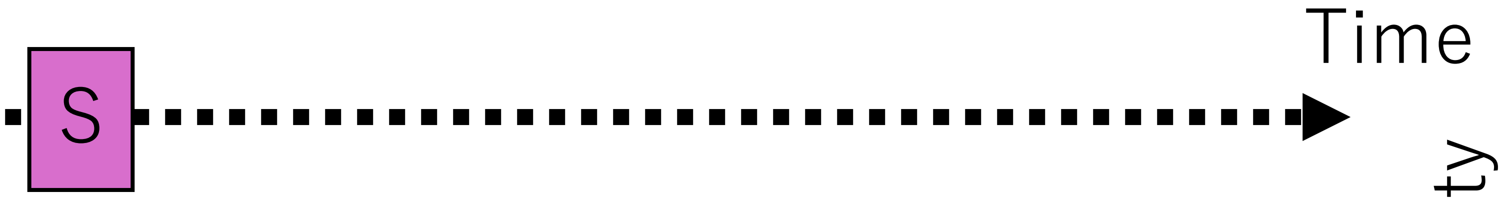




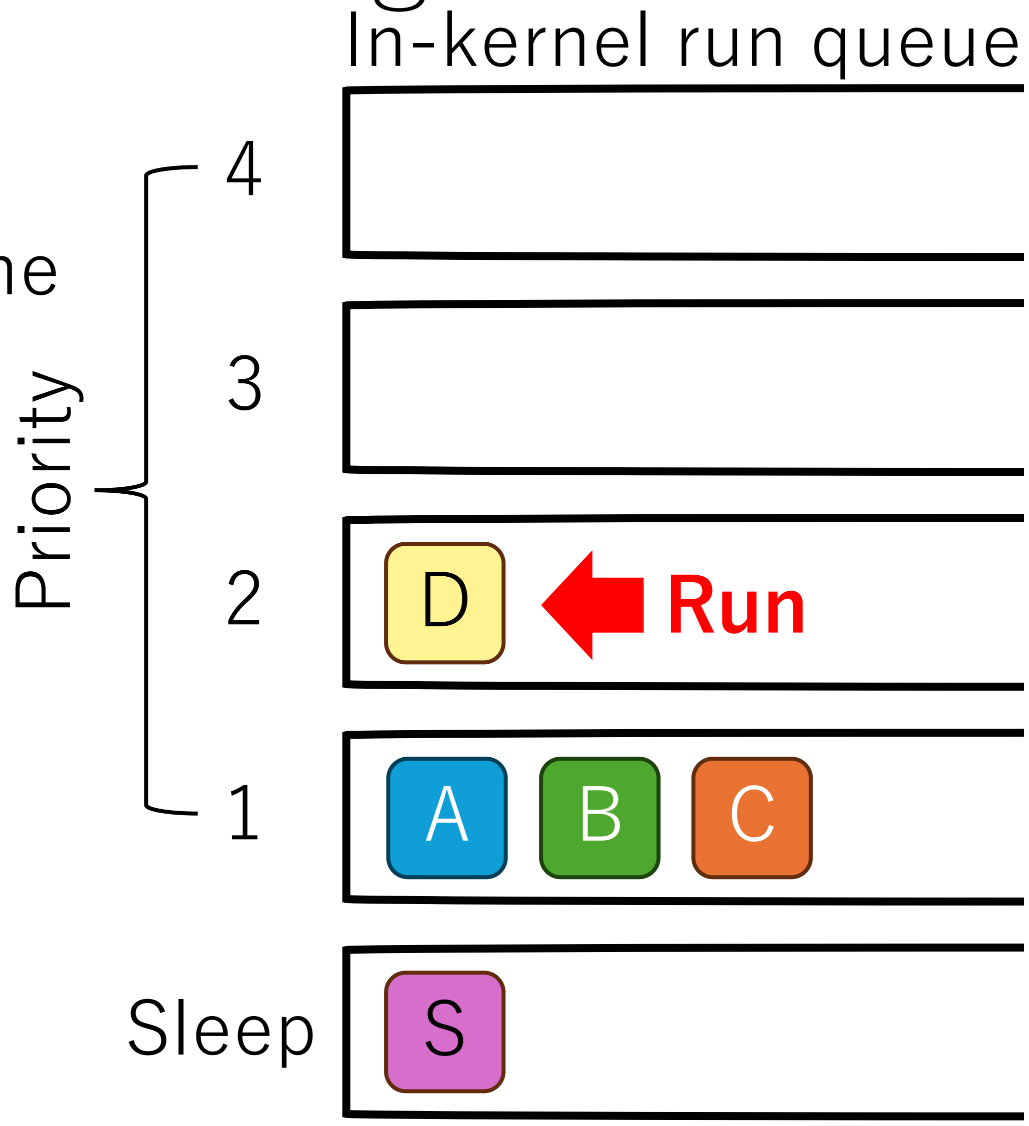
The kernel-space process scheduler picks up the highest priority process

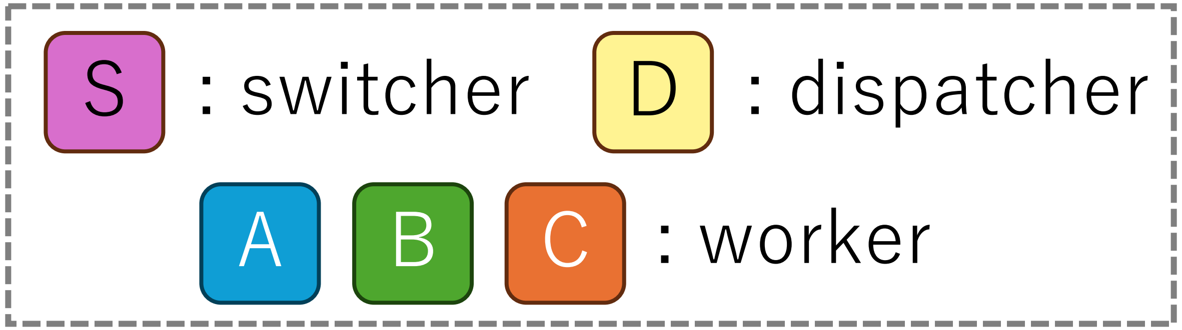


Use Case: Preemptive Scheduling

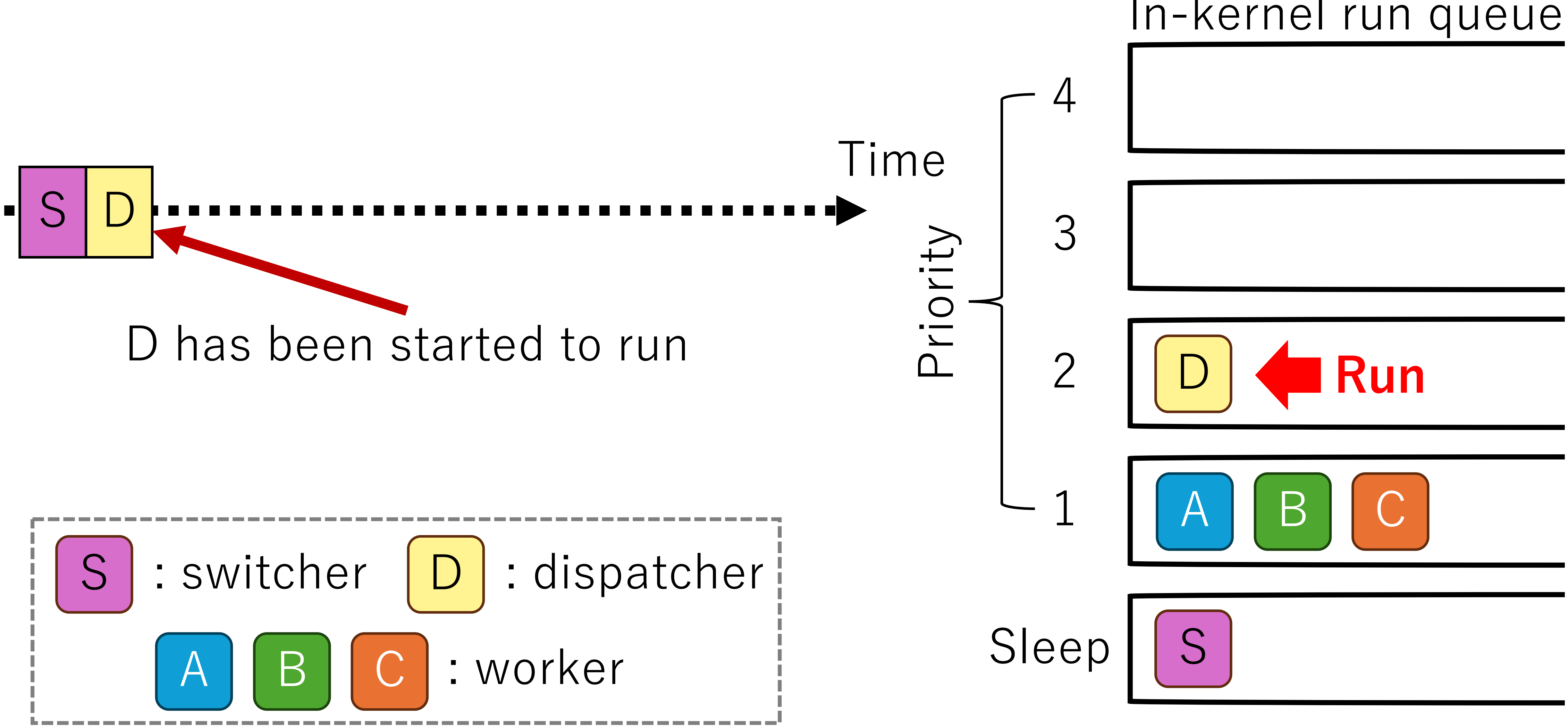


Execution is switched from S to D

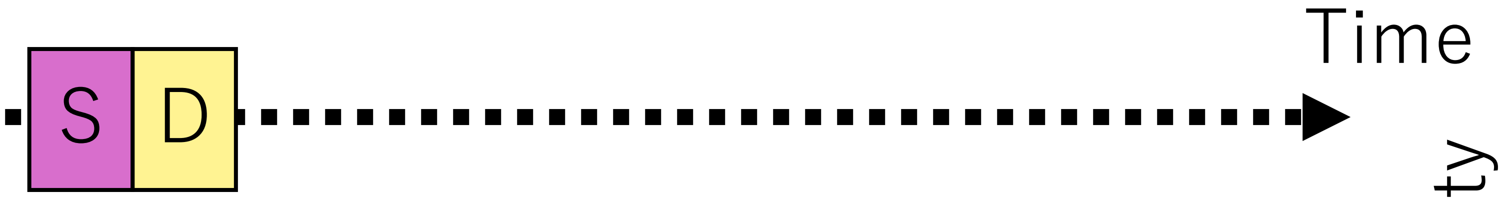


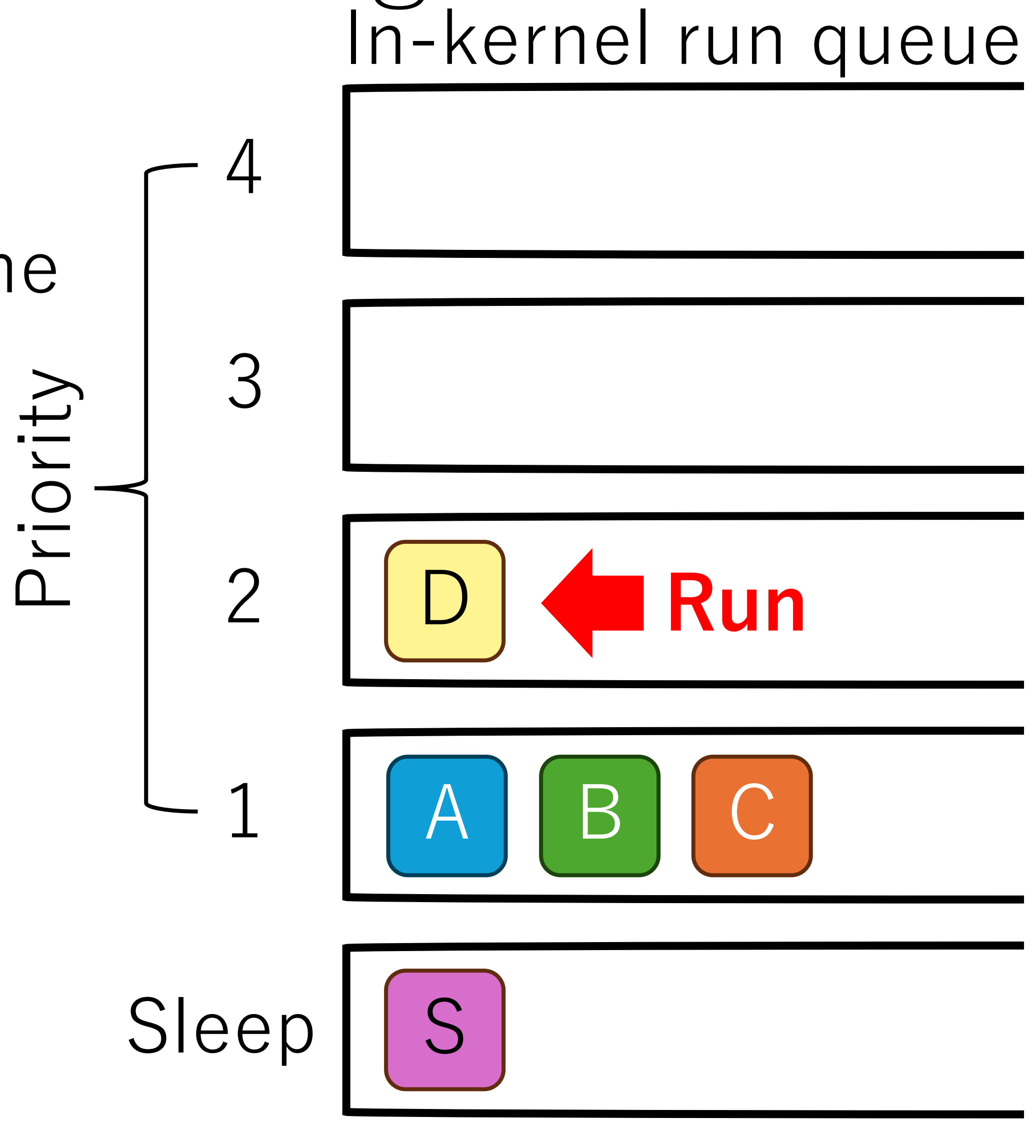


Use Case: Preemptive Scheduling

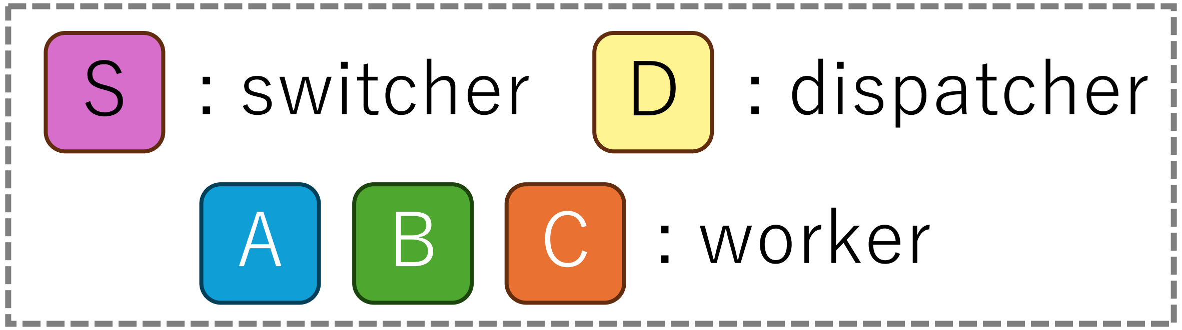


Use Case: Preemptive Scheduling

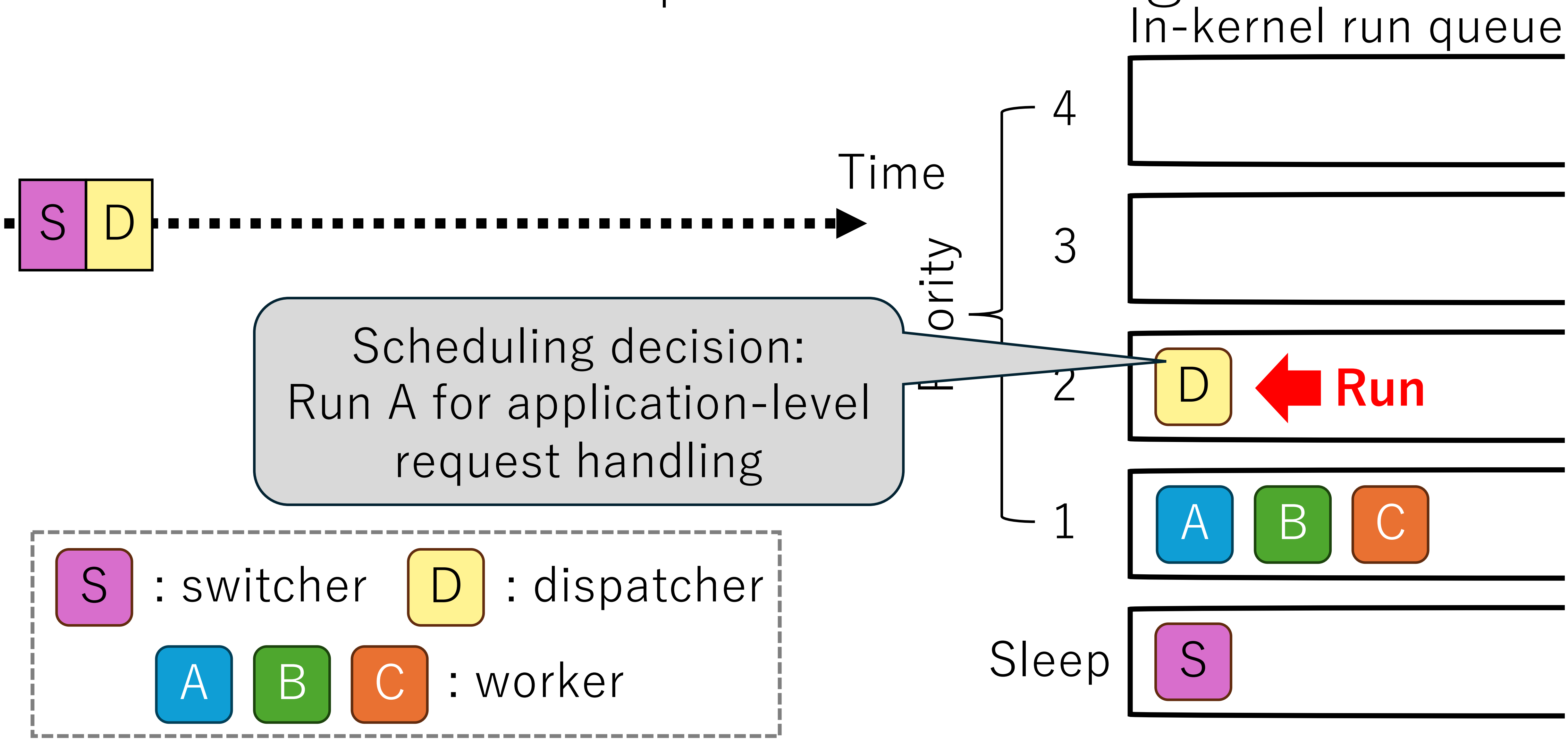




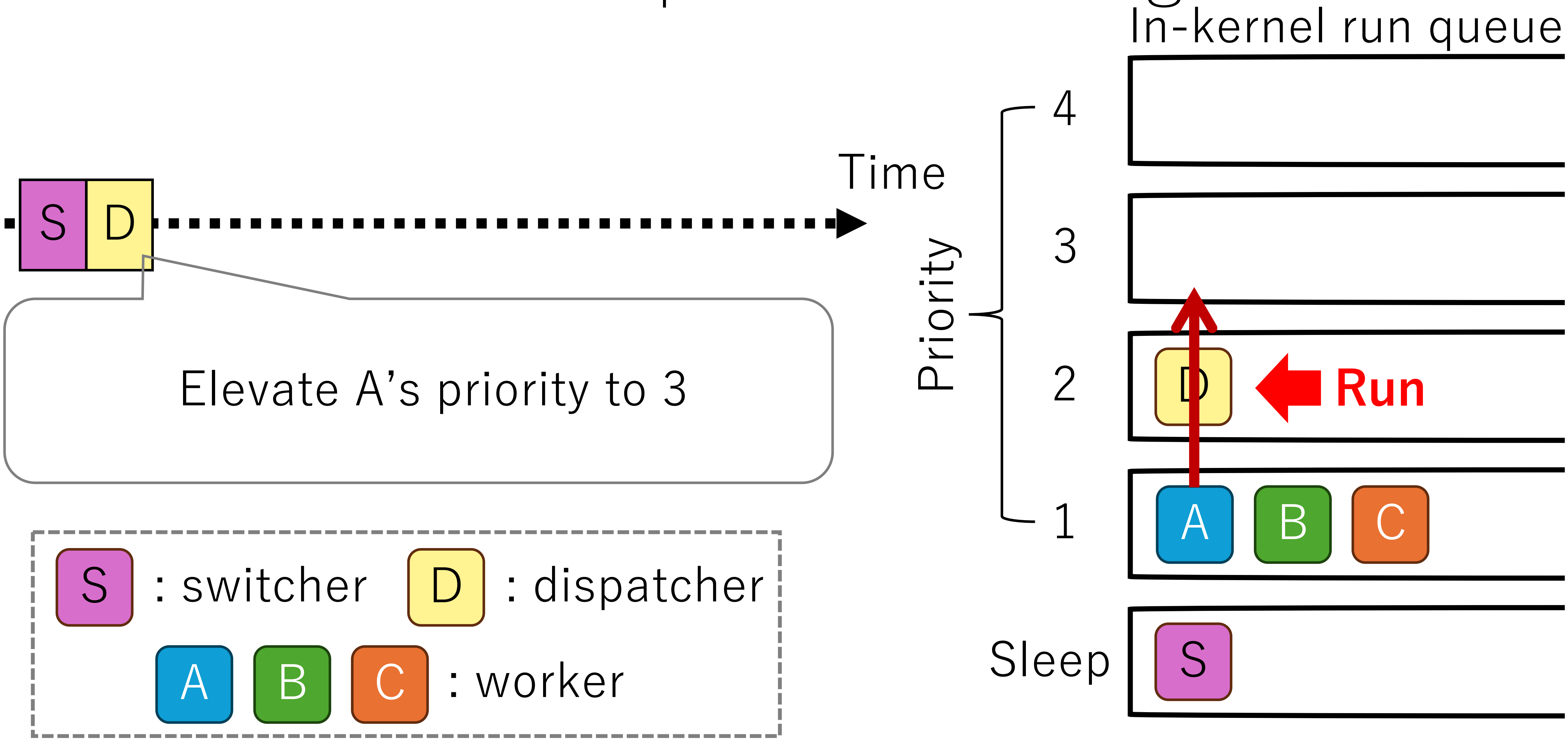
D handles incoming packets and dispatch requests to a worker



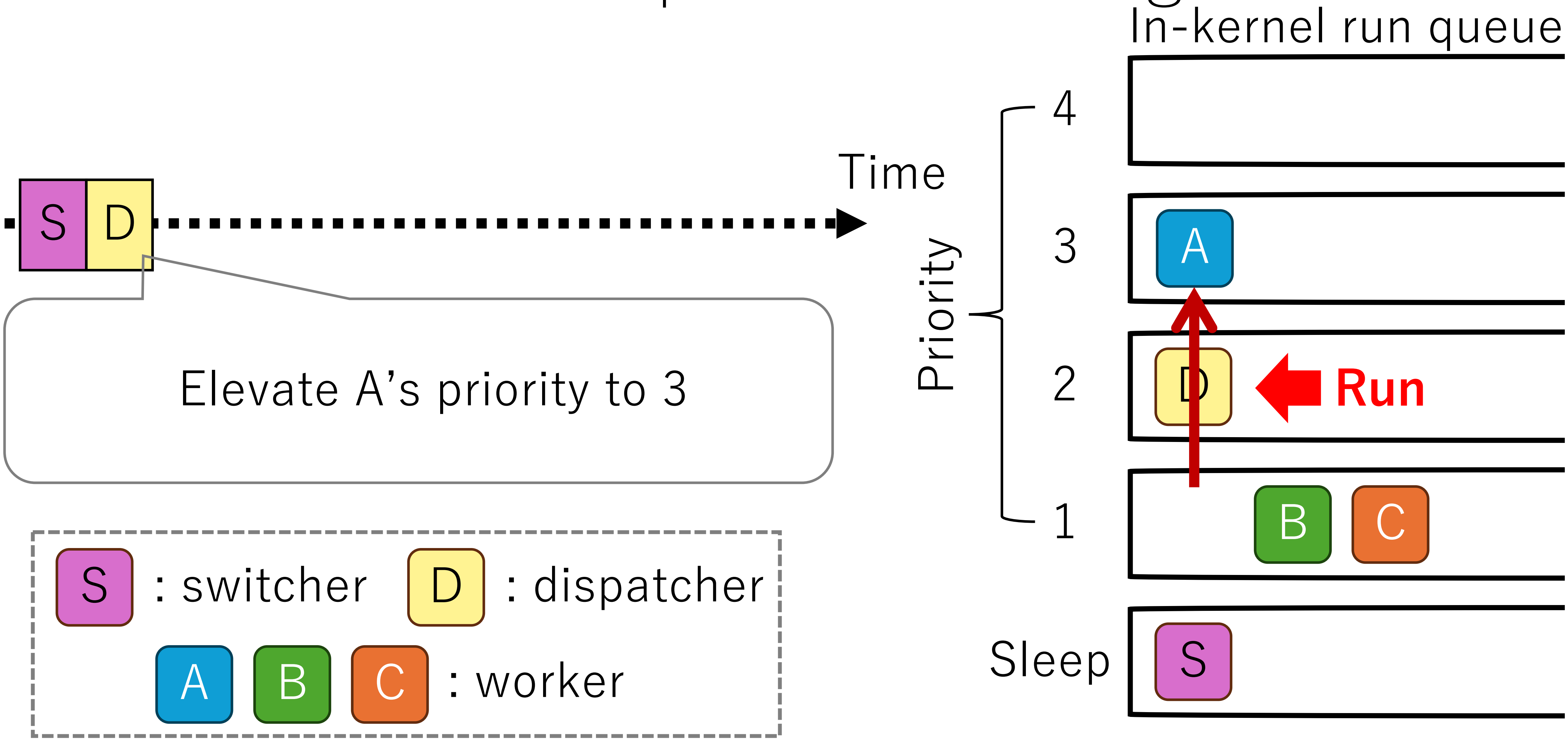
Use Case: Preemptive Scheduling



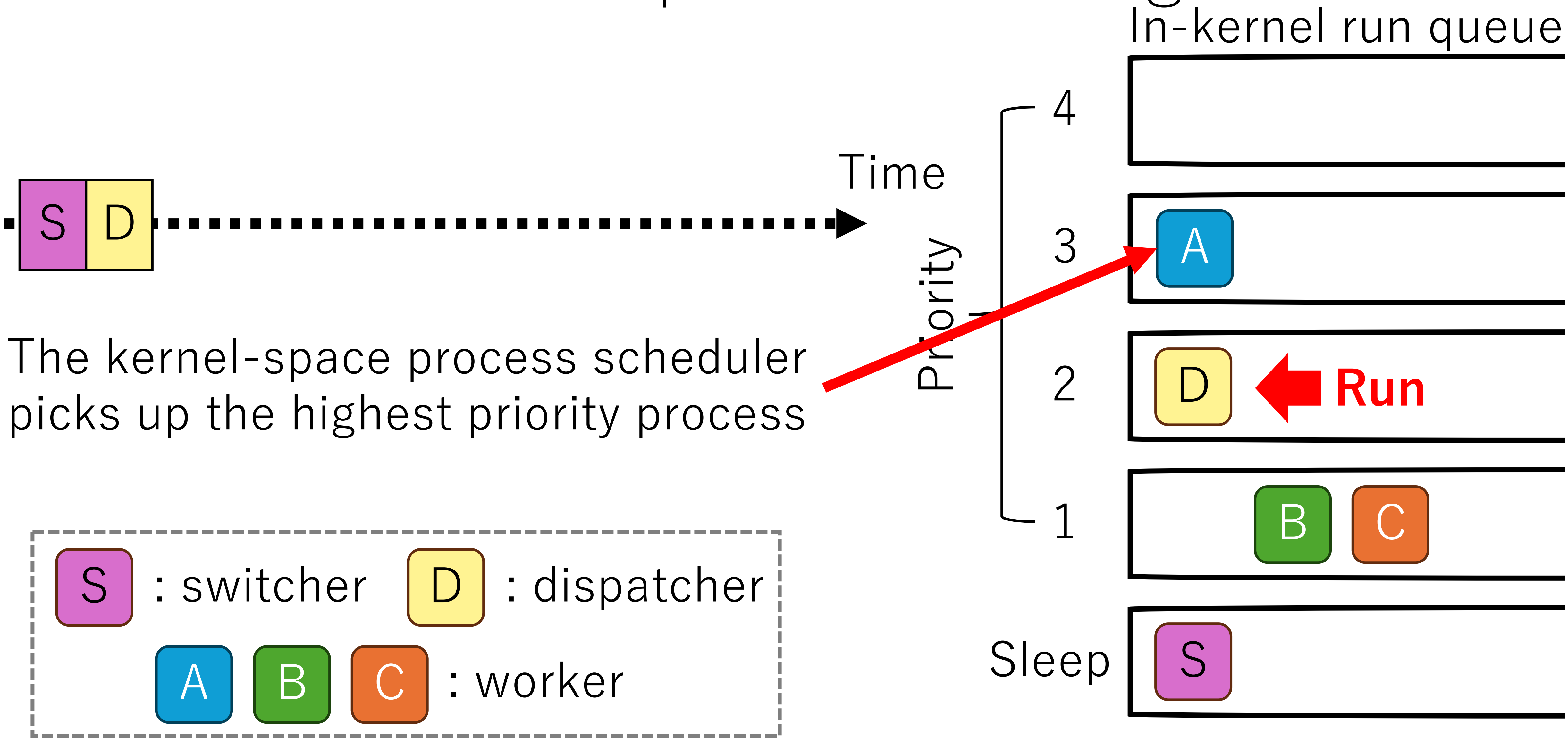
Use Case: Preemptive Scheduling



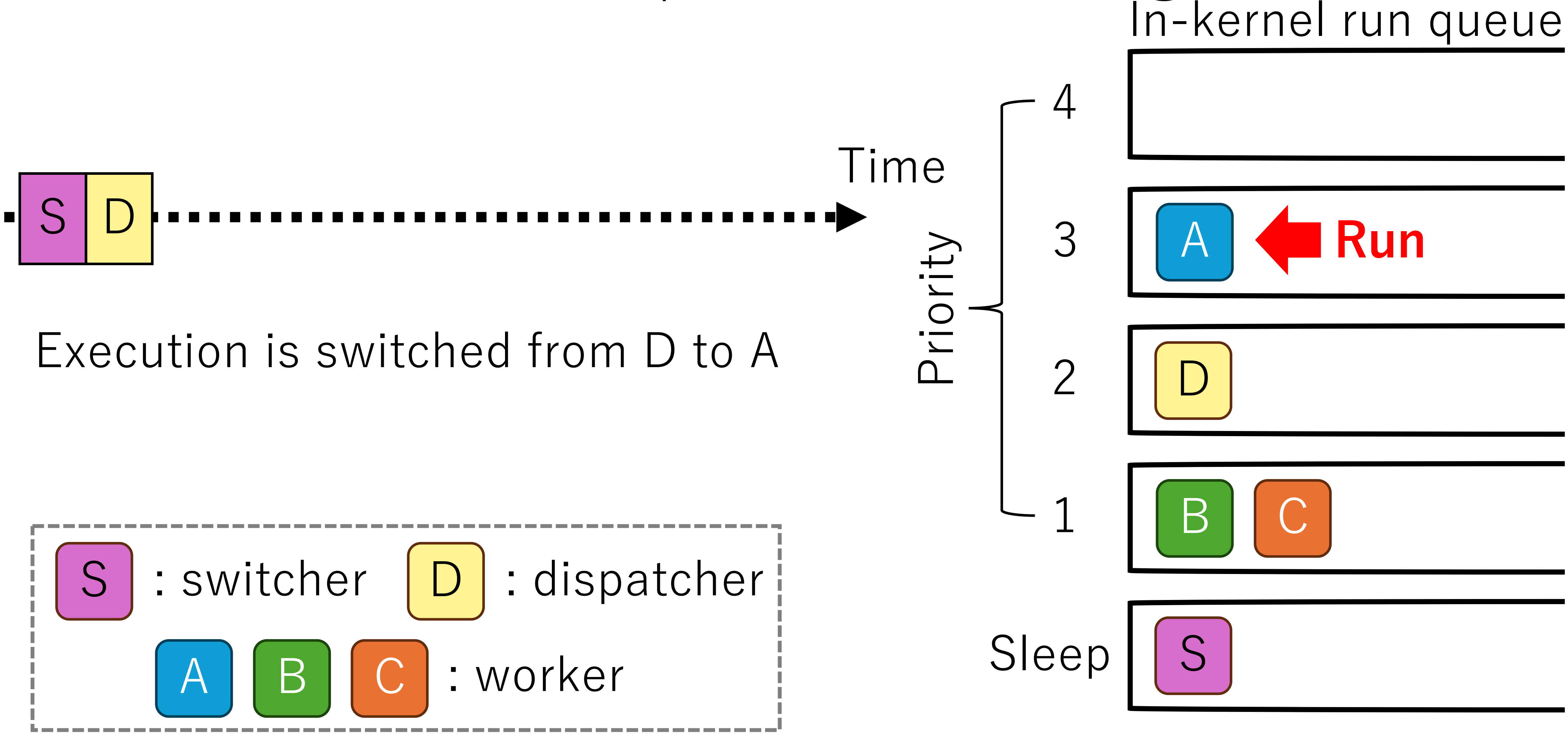
Use Case: Preemptive Scheduling



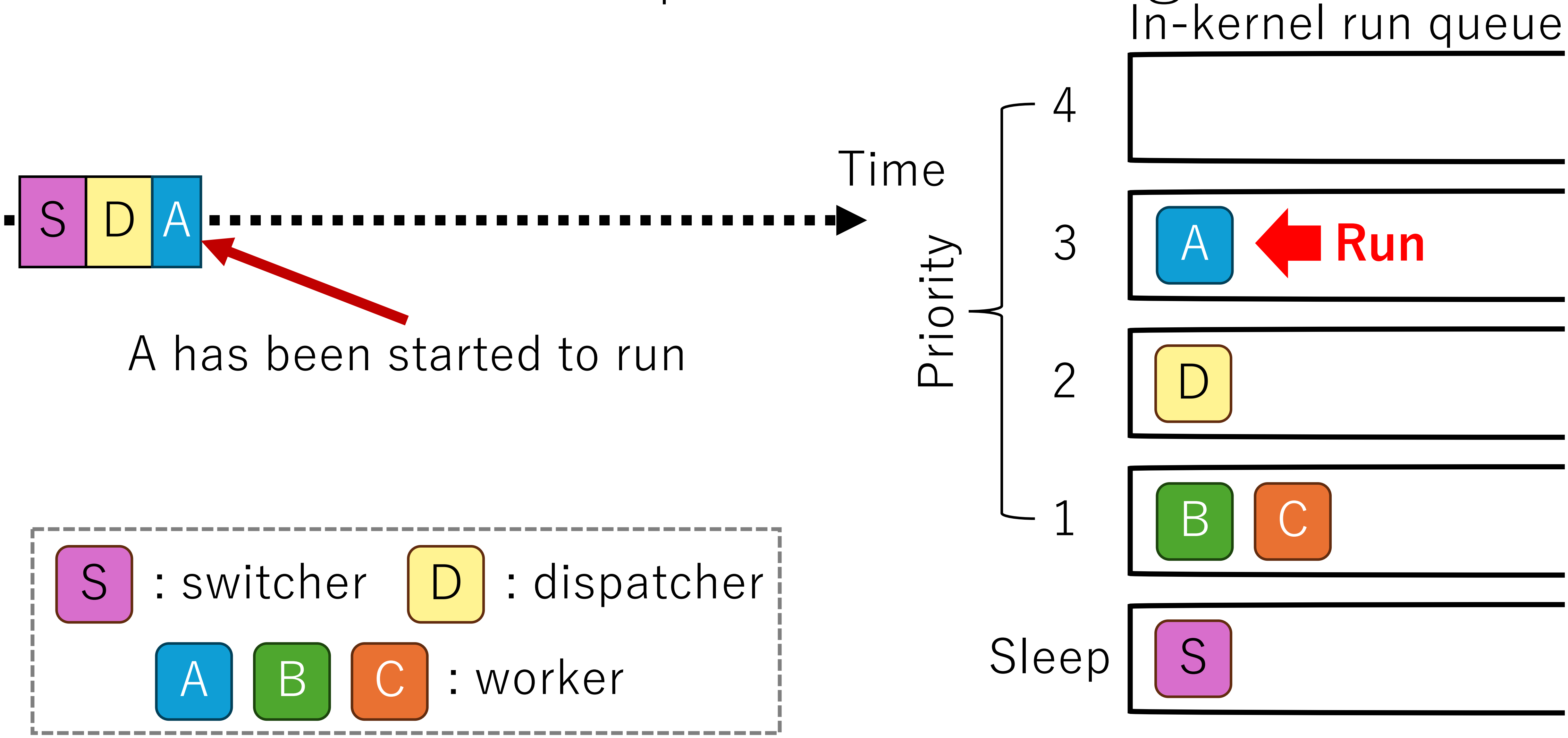
Use Case: Preemptive Scheduling



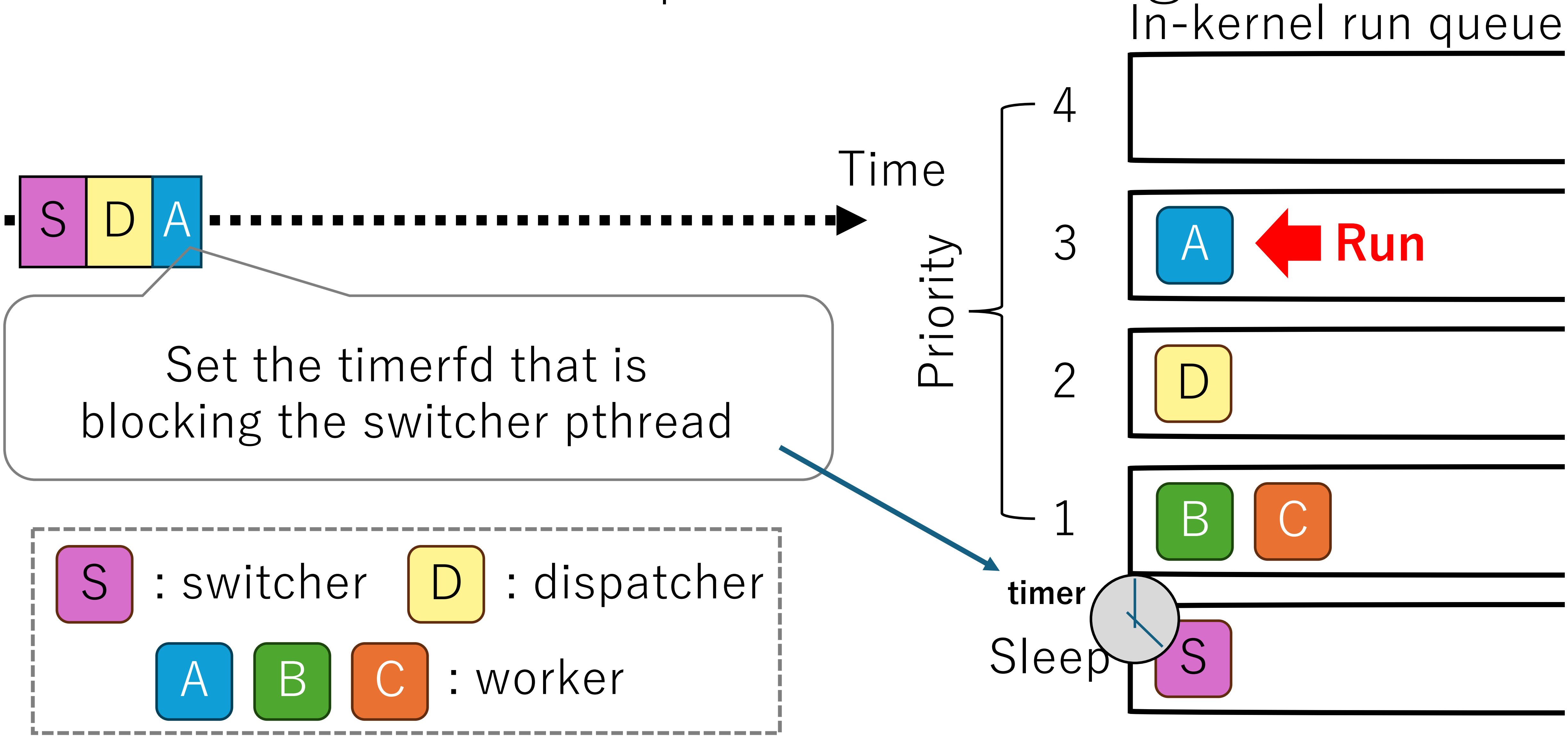
Use Case: Preemptive Scheduling



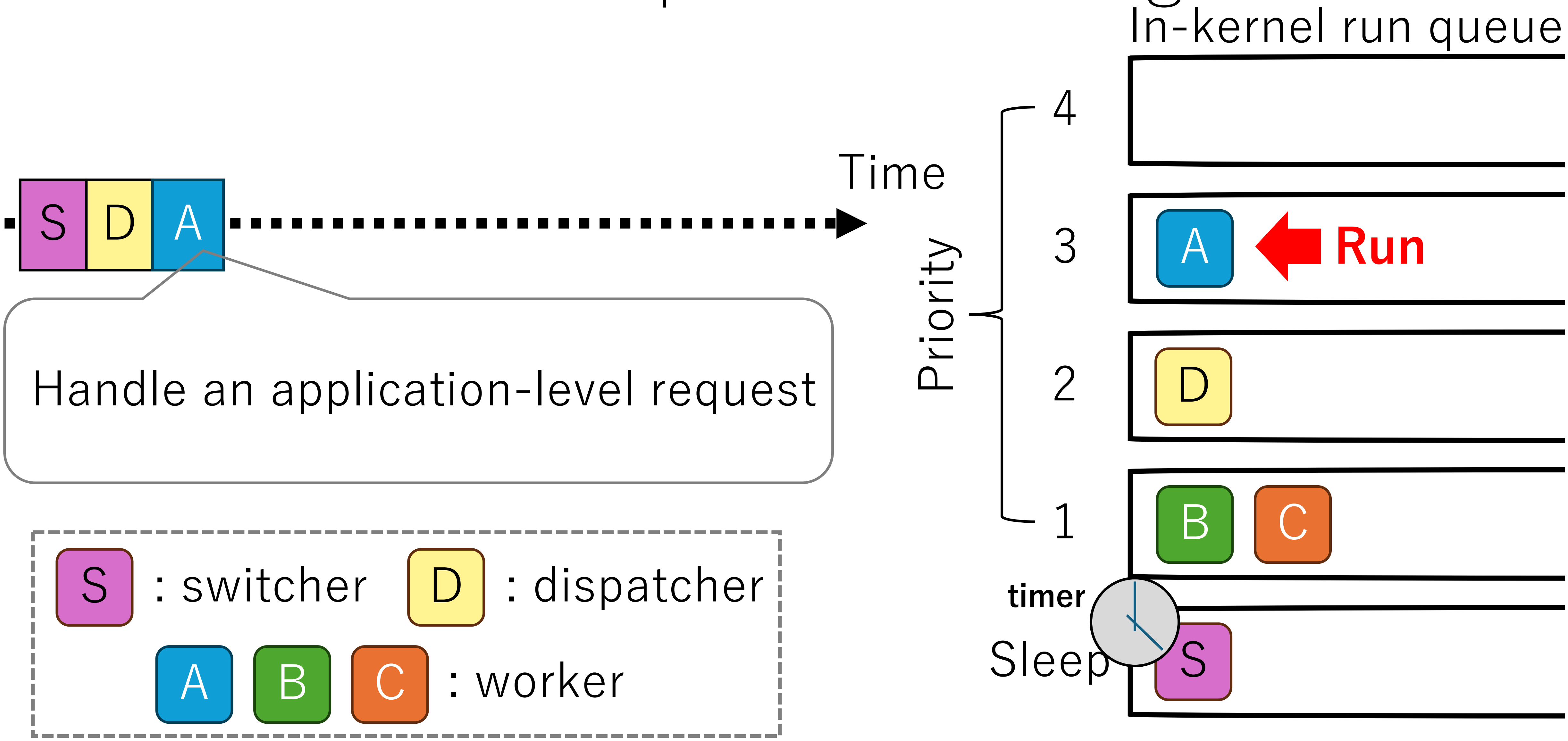
Use Case: Preemptive Scheduling



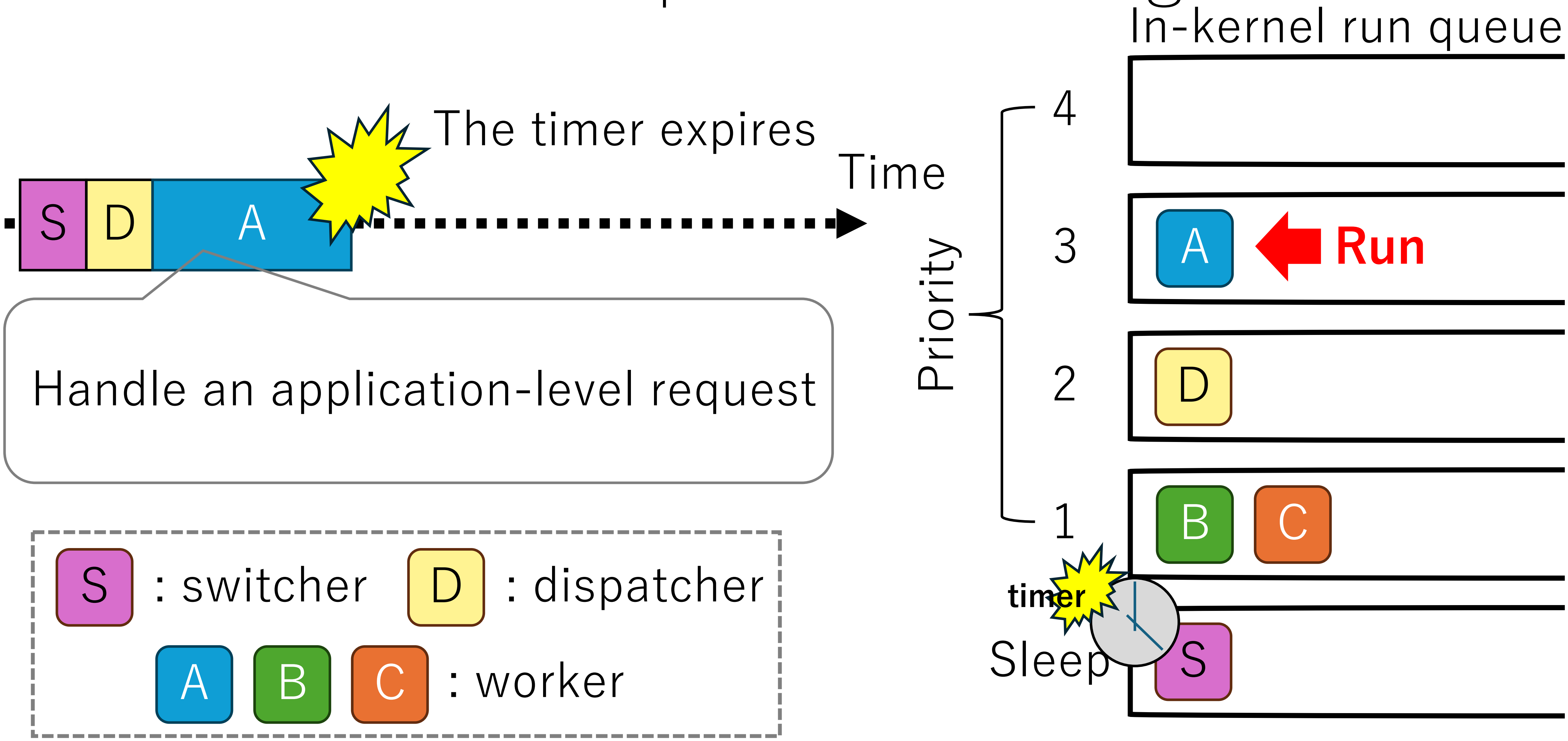
Use Case: Preemptive Scheduling



Use Case: Preemptive Scheduling

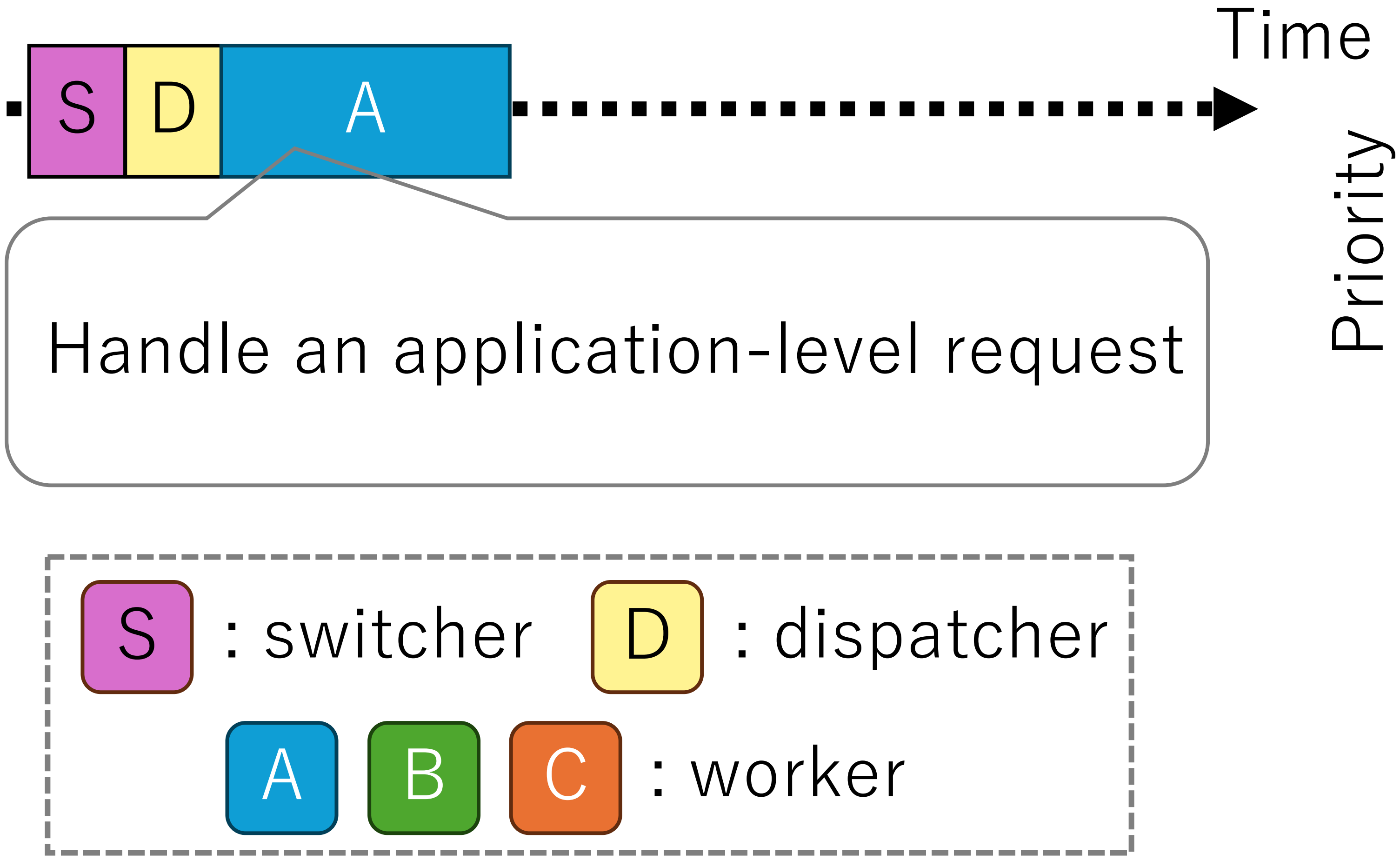


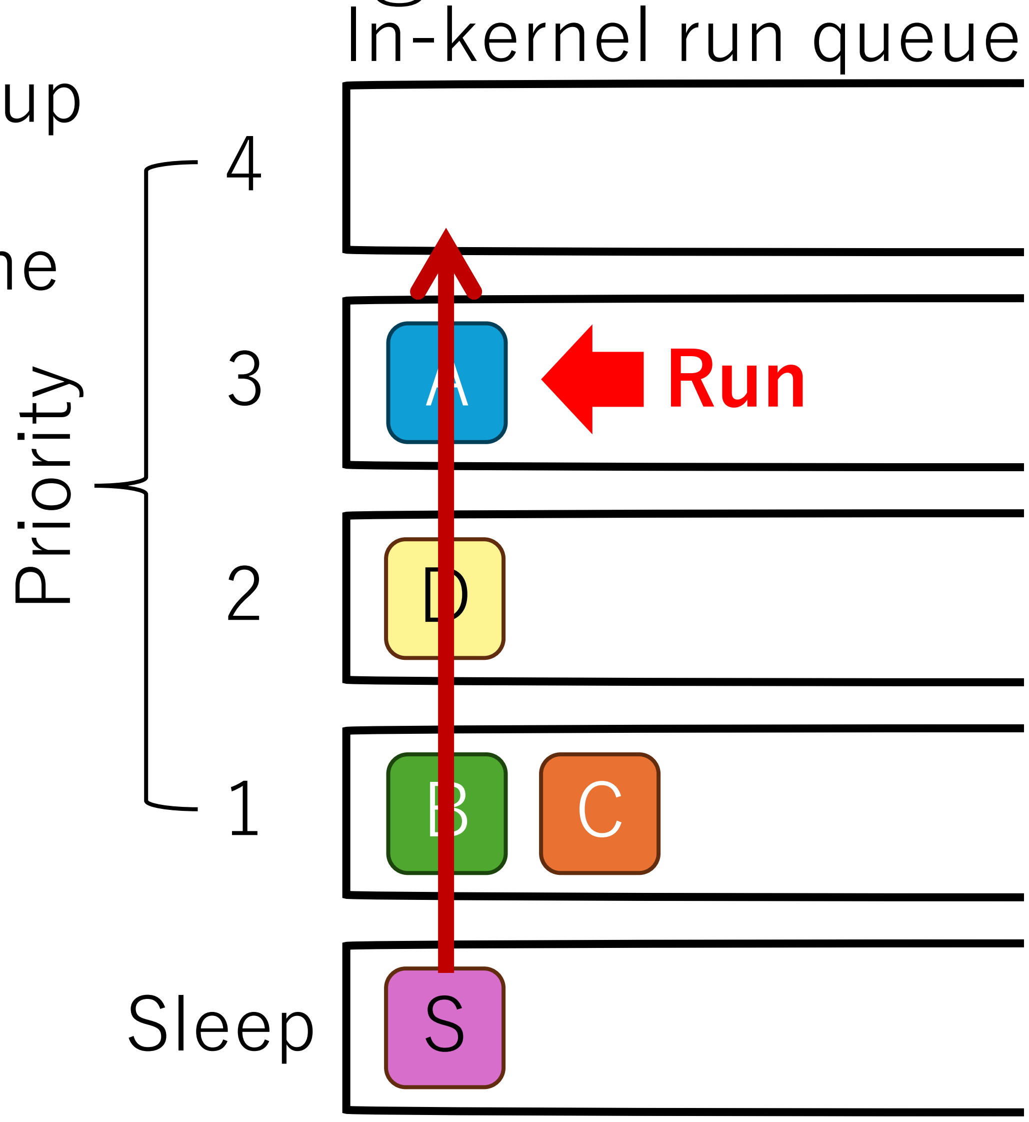
Use Case: Preemptive Scheduling



Use Case: Preemptive Scheduling

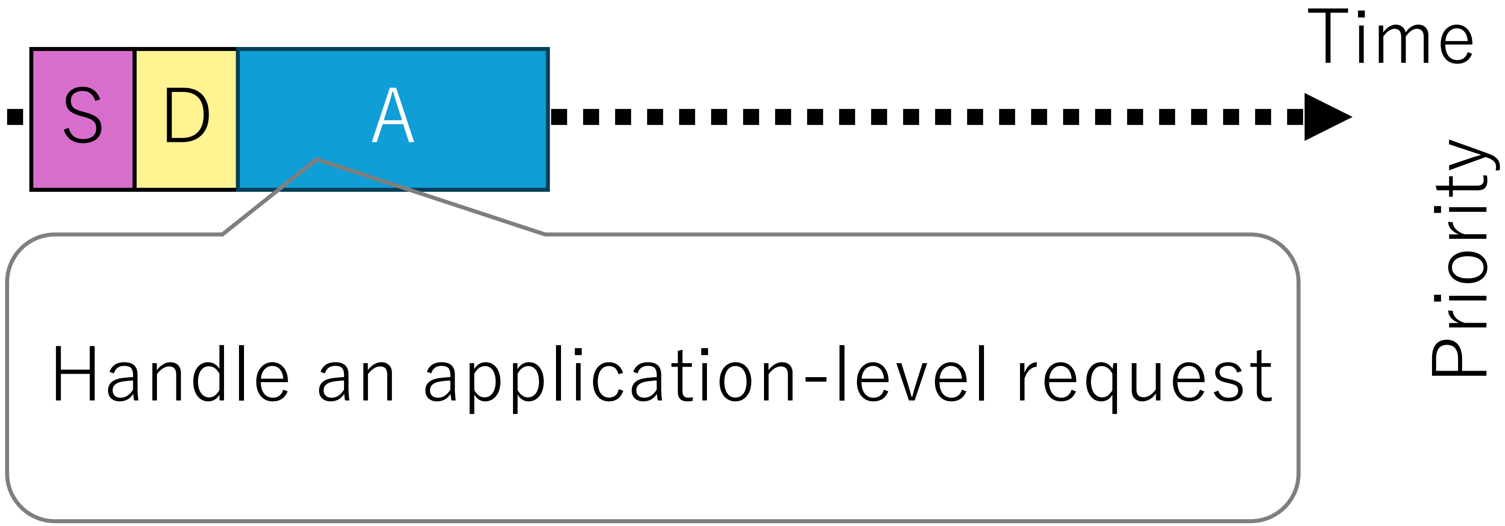
The switcher pthread wakes up

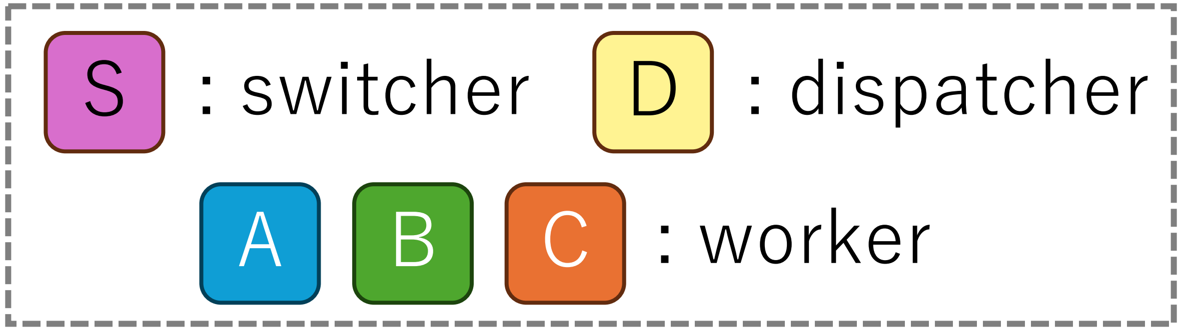


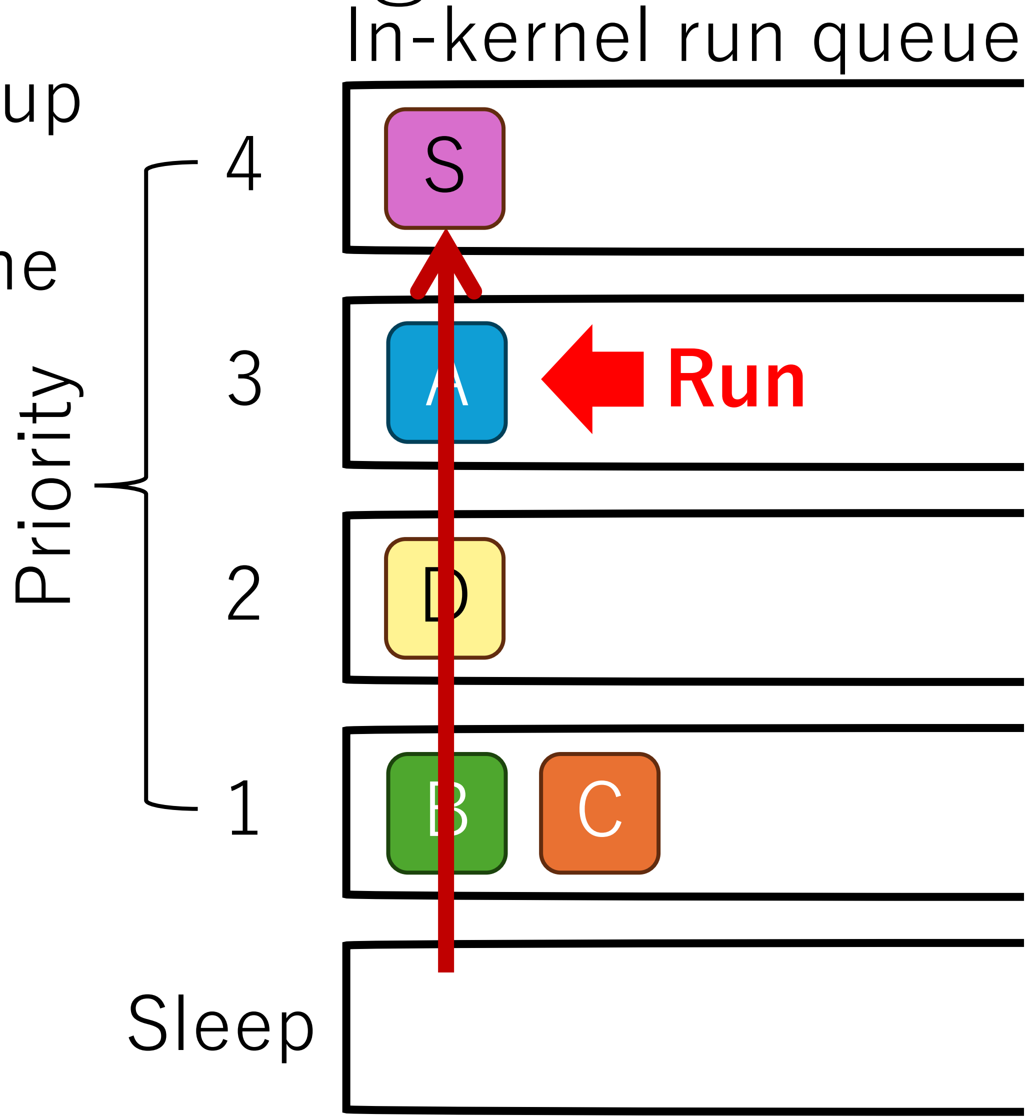


Use Case: Preemptive Scheduling

The switcher pthread wakes up

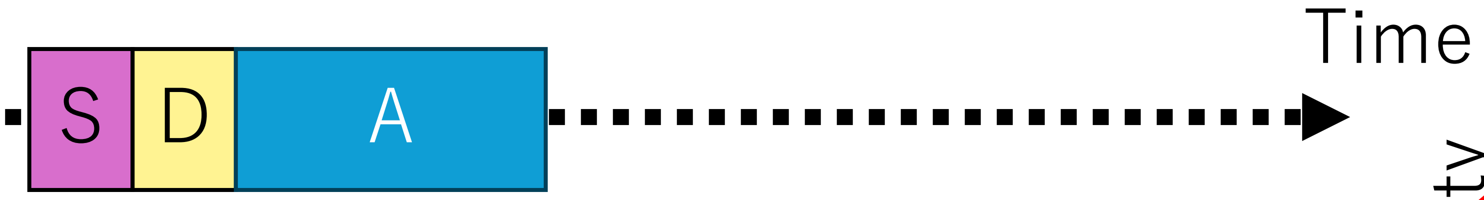




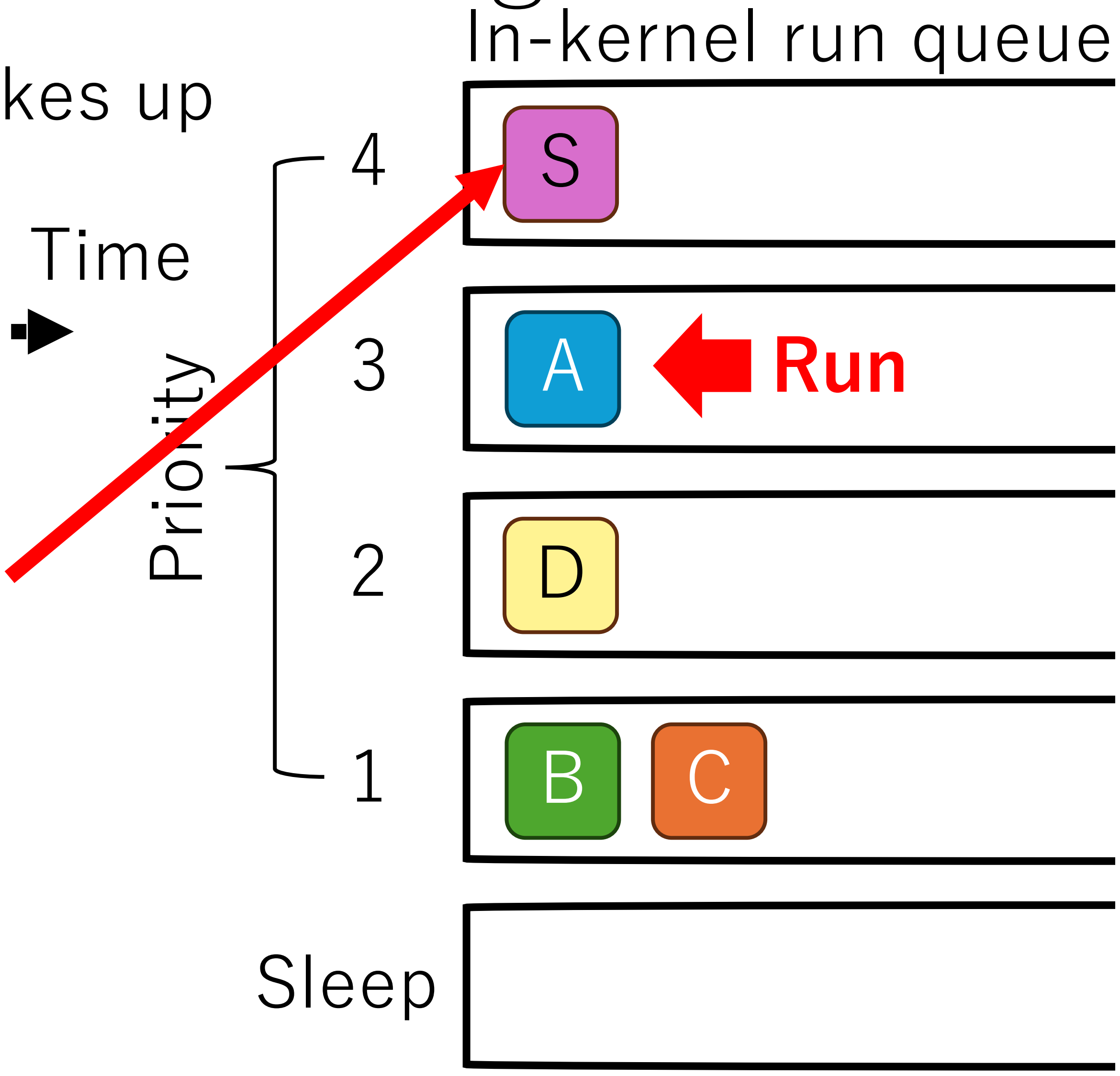


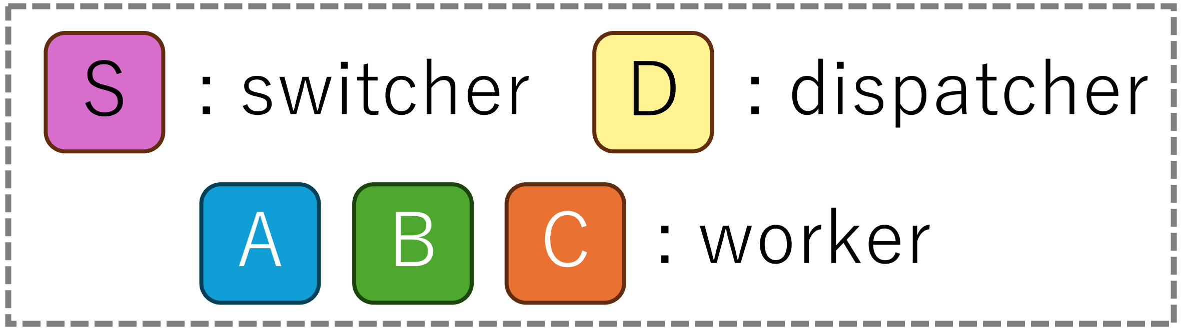
Use Case: Preemptive Scheduling

The switcher pthread wakes up

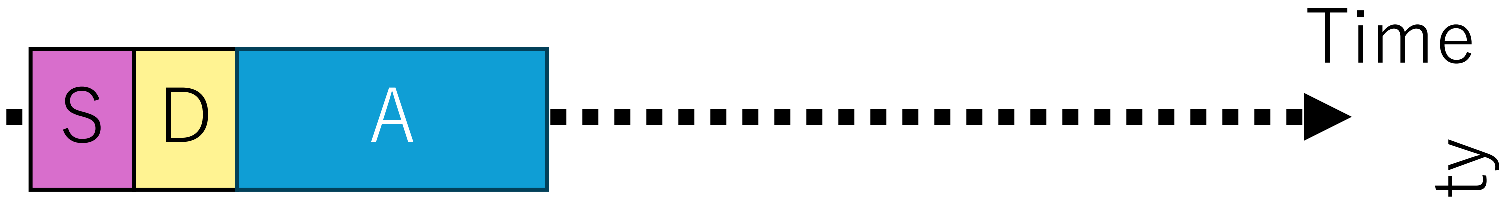


The kernel-space process scheduler
picks up the highest priority process

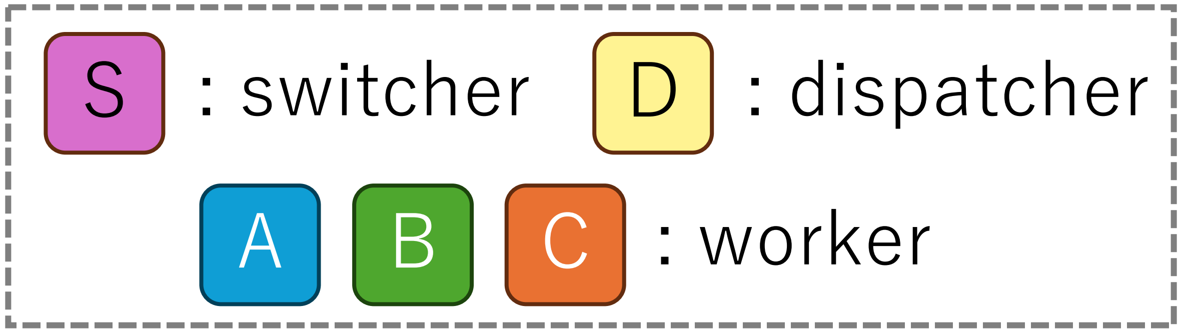


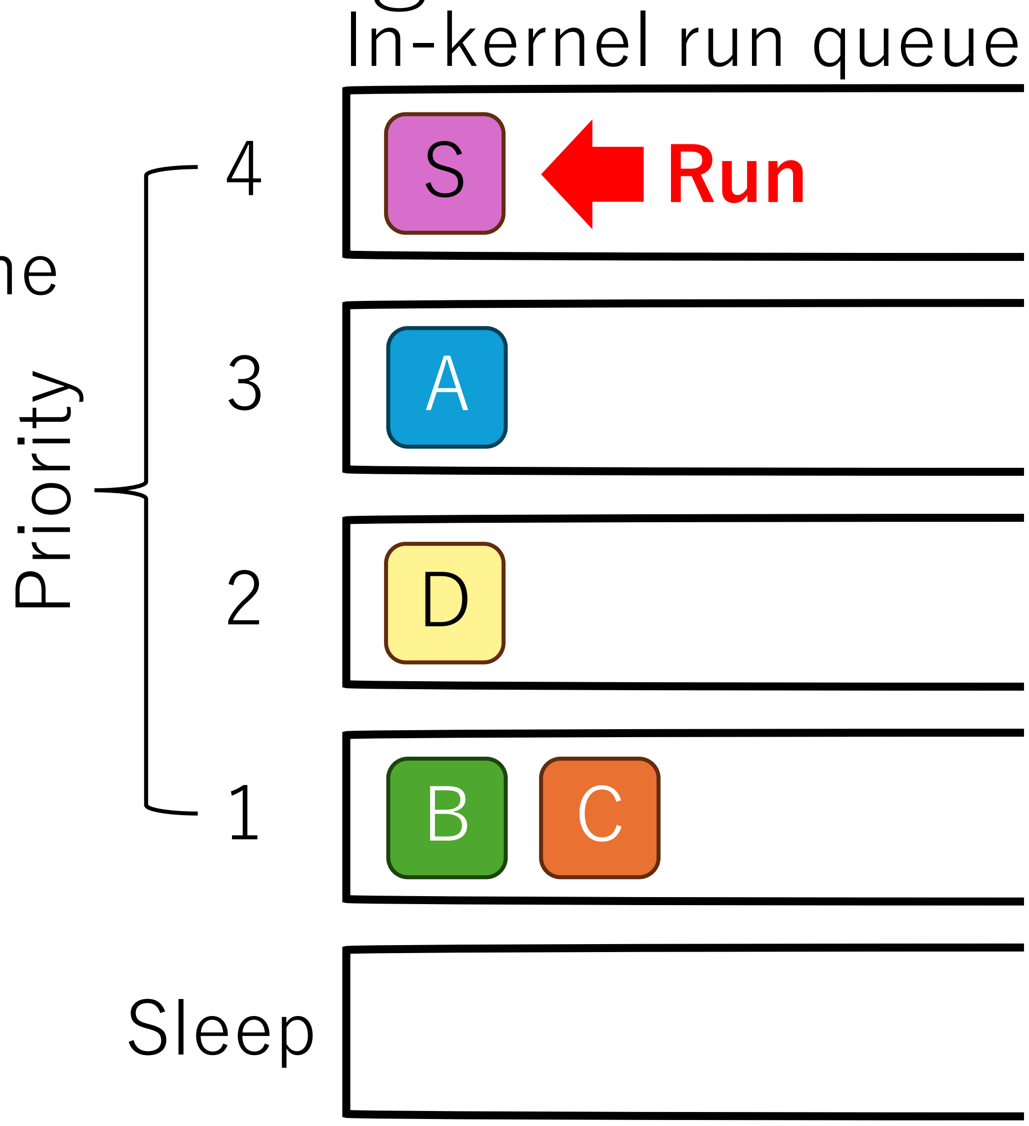


Use Case: Preemptive Scheduling

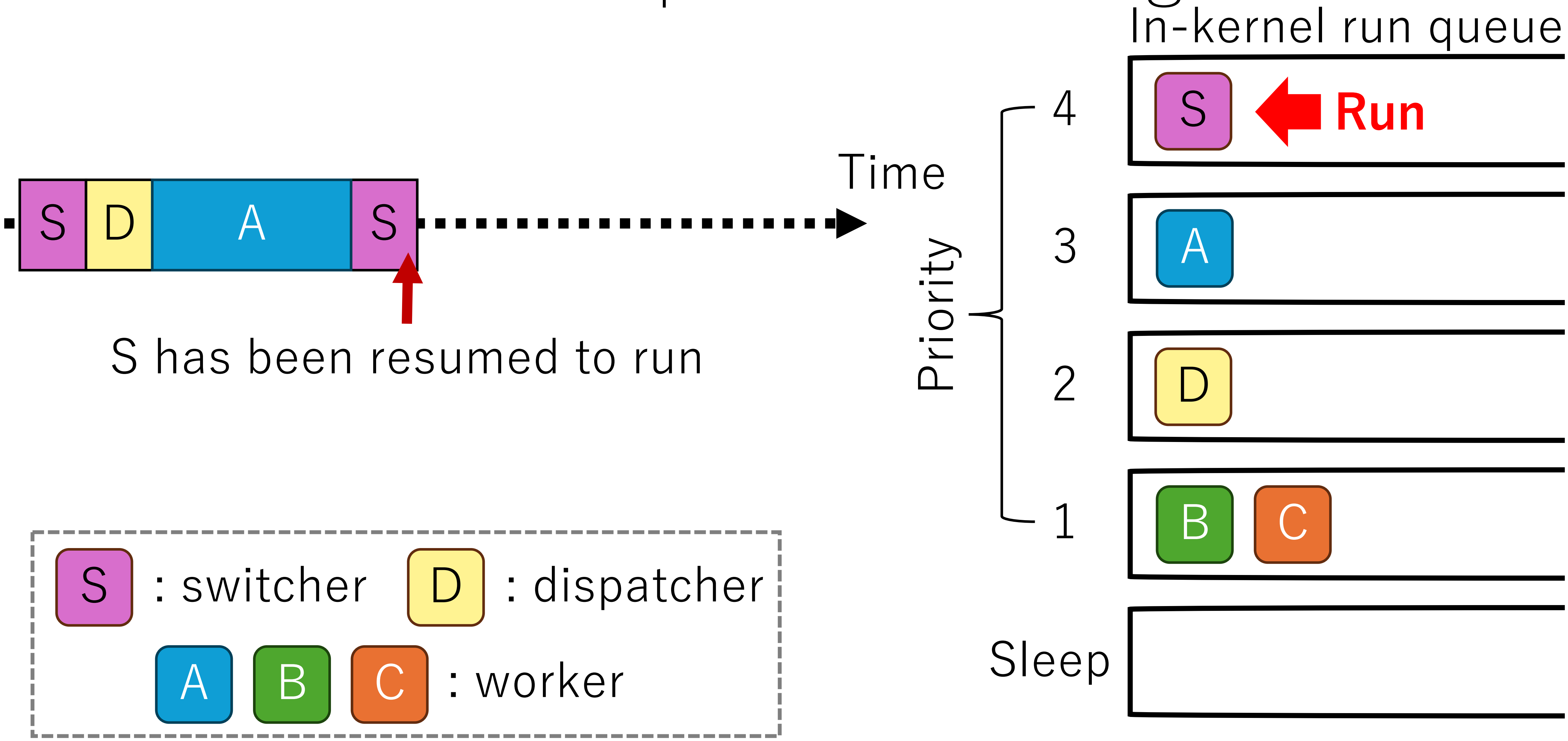


Execution is switched from A to S

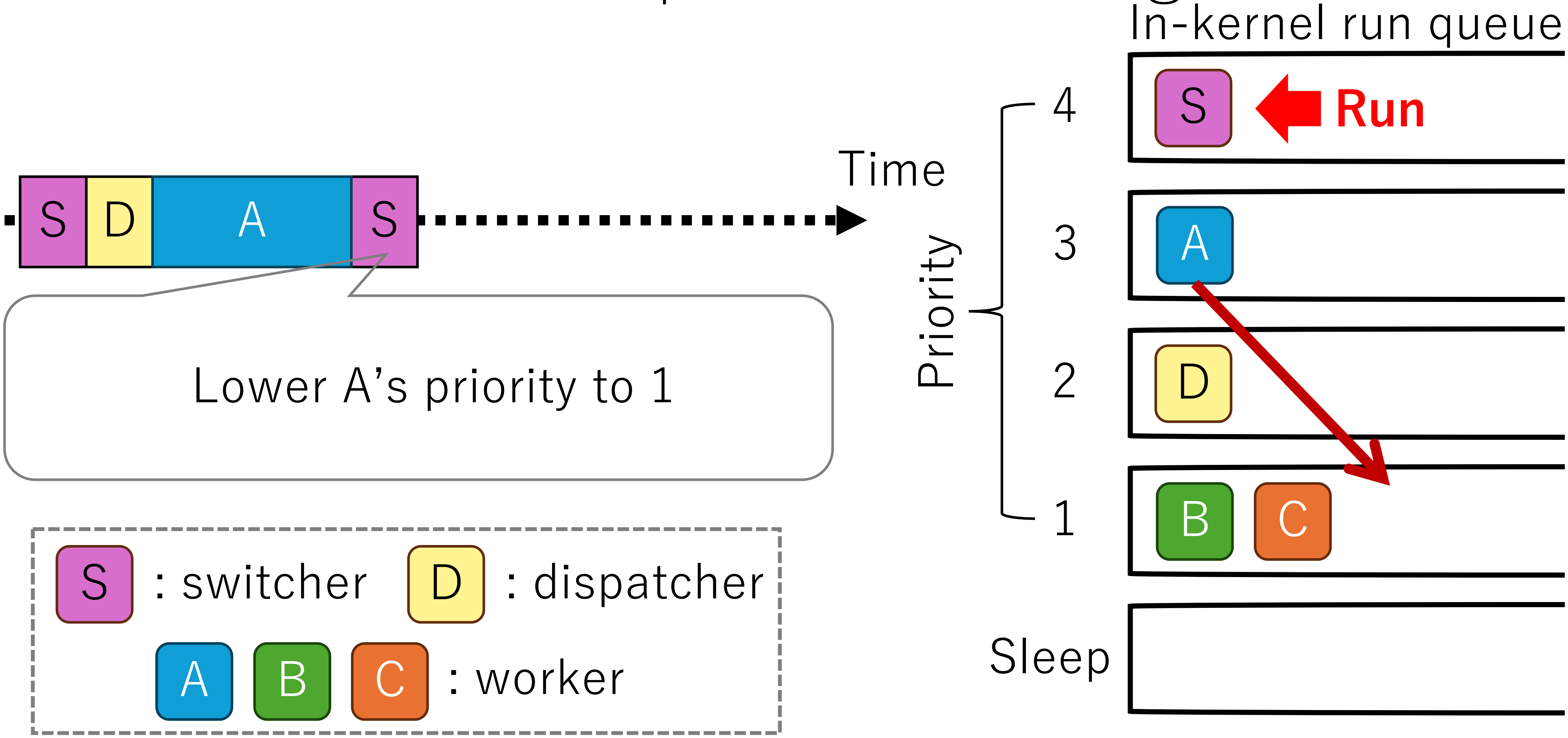




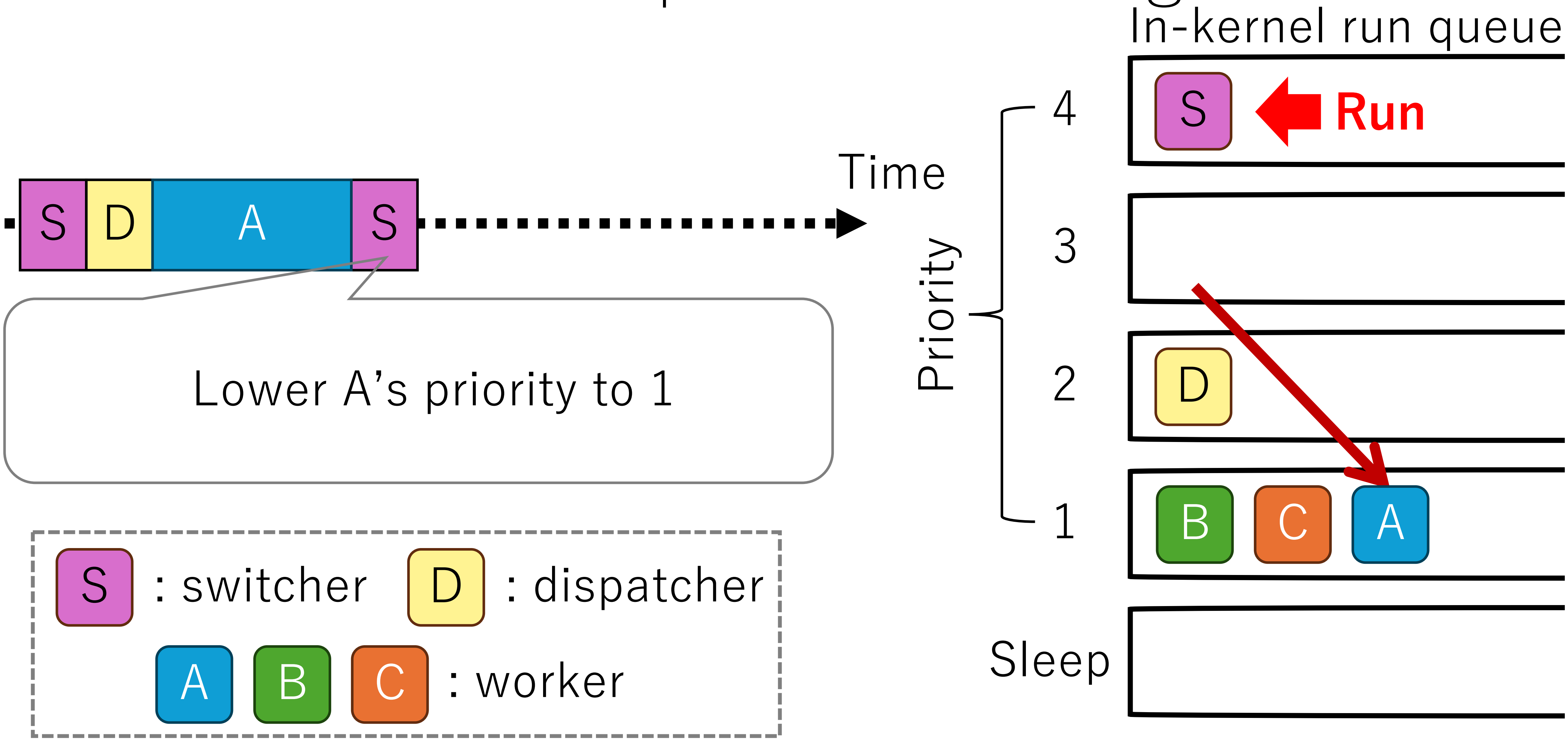
Use Case: Preemptive Scheduling



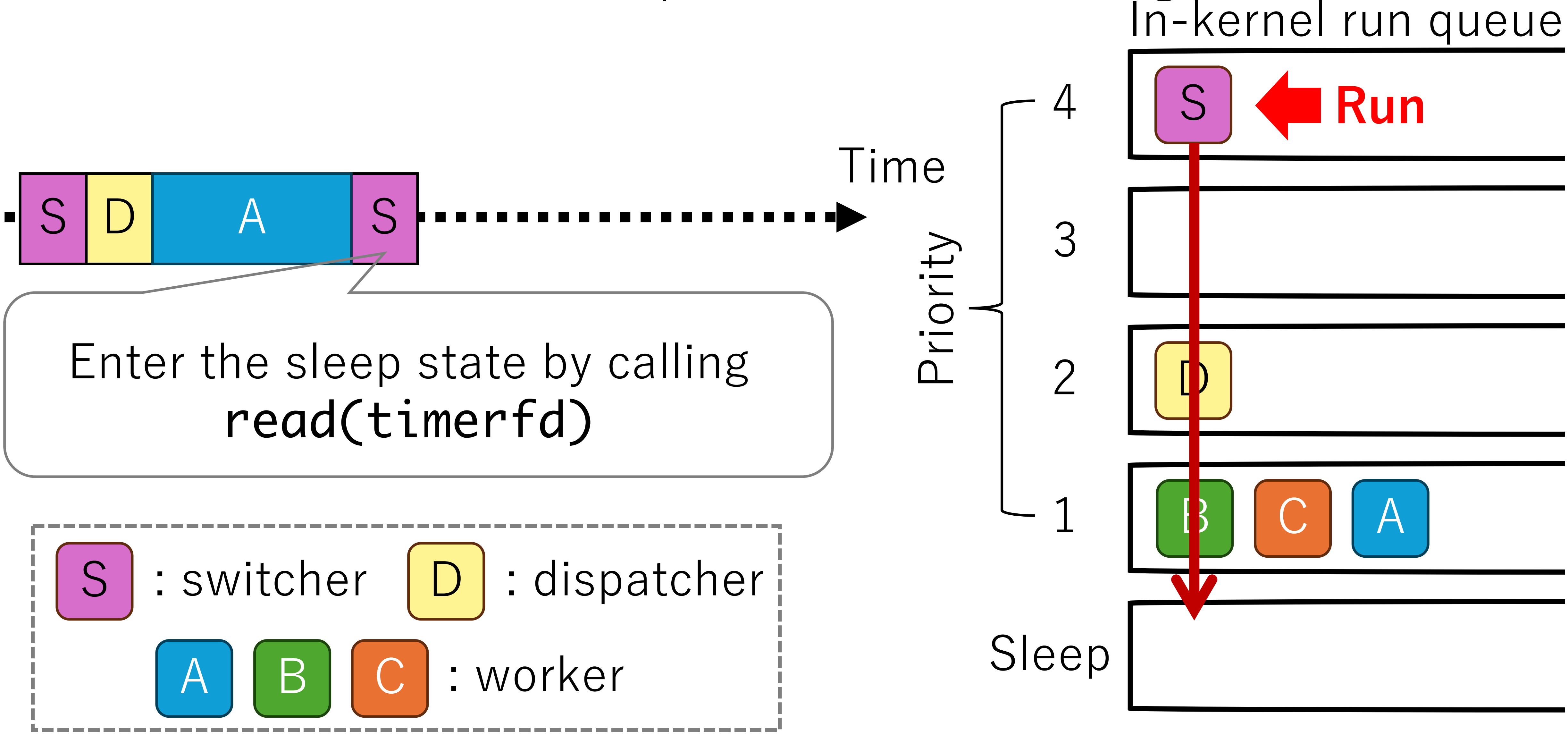
Use Case: Preemptive Scheduling



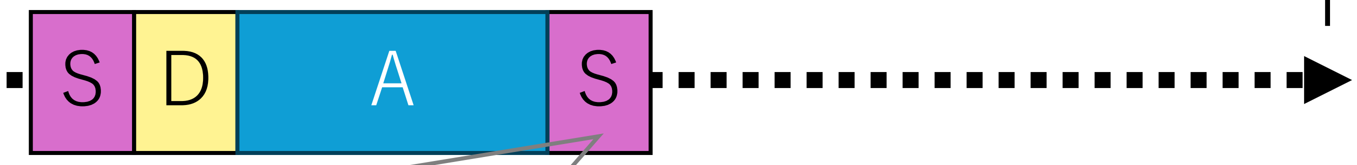
Use Case: Preemptive Scheduling



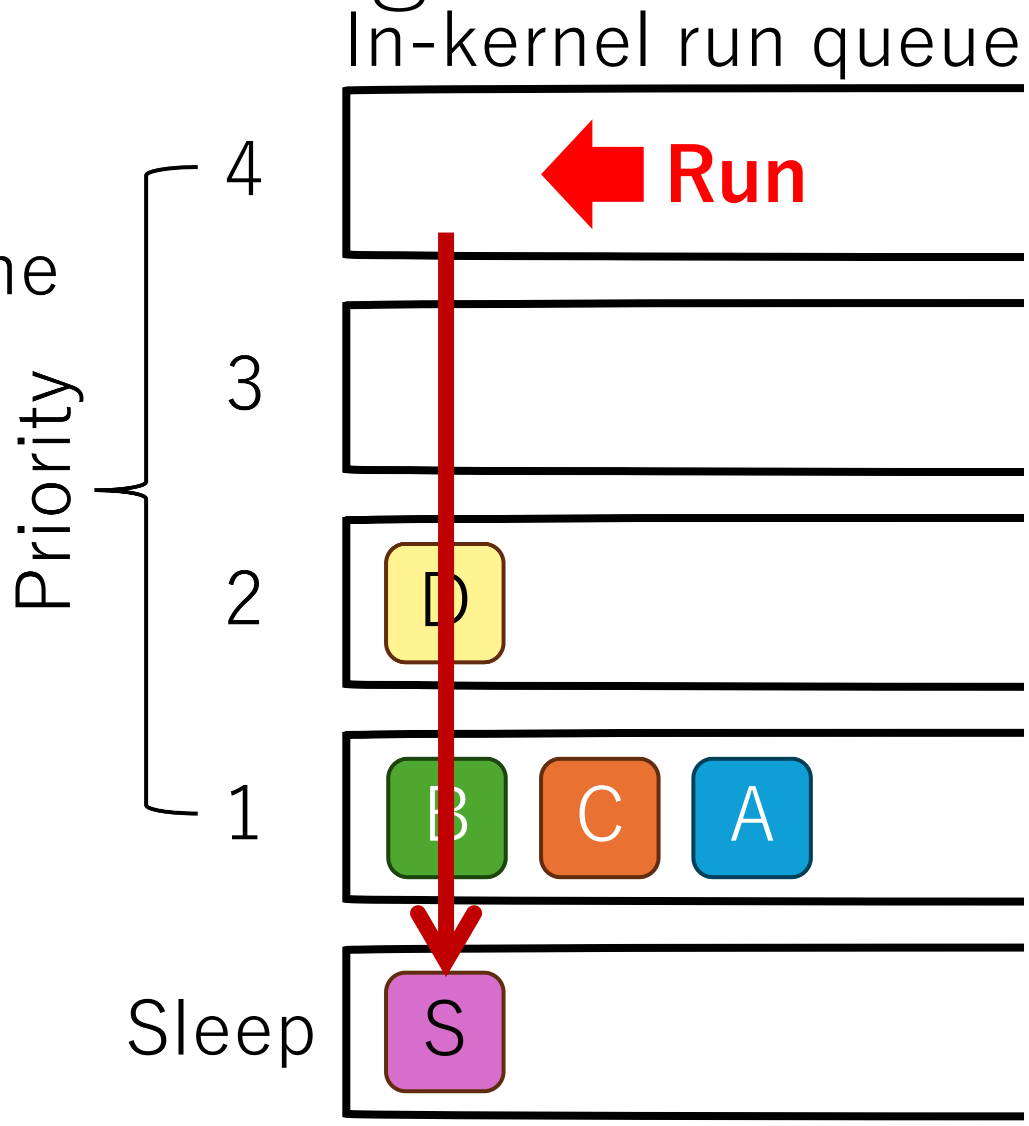
Use Case: Preemptive Scheduling



Use Case: Preemptive Scheduling



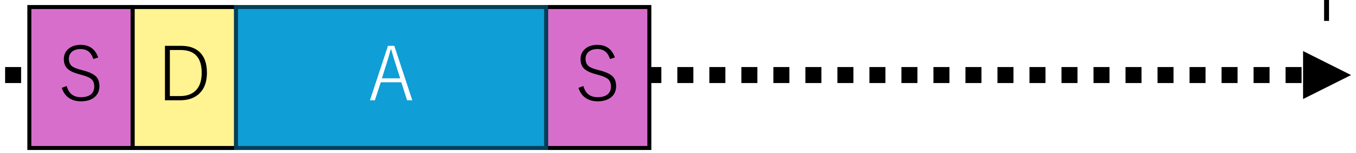
Enter the sleep state by calling
`read(timerfd)`



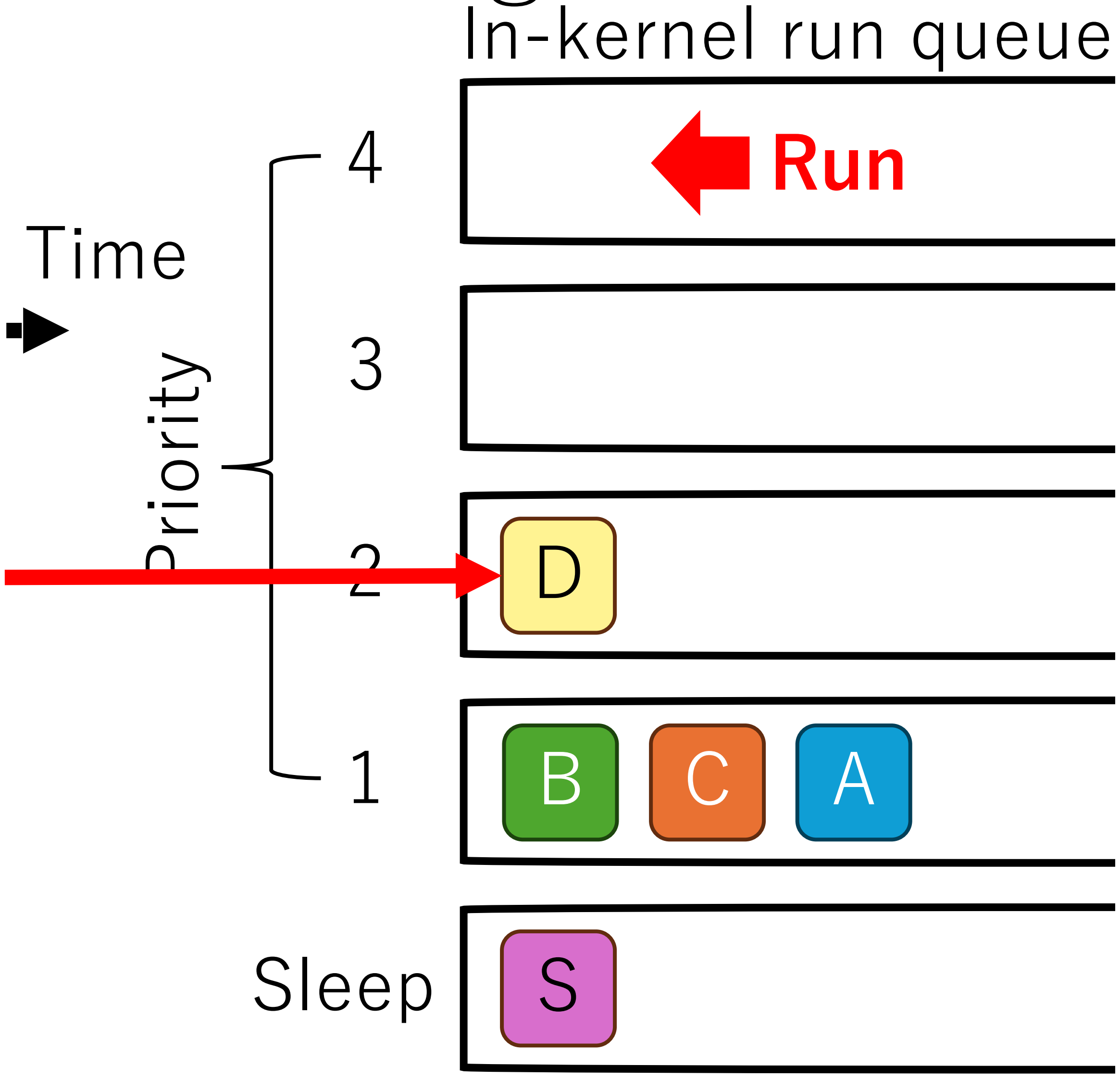
S : switcher D : dispatcher

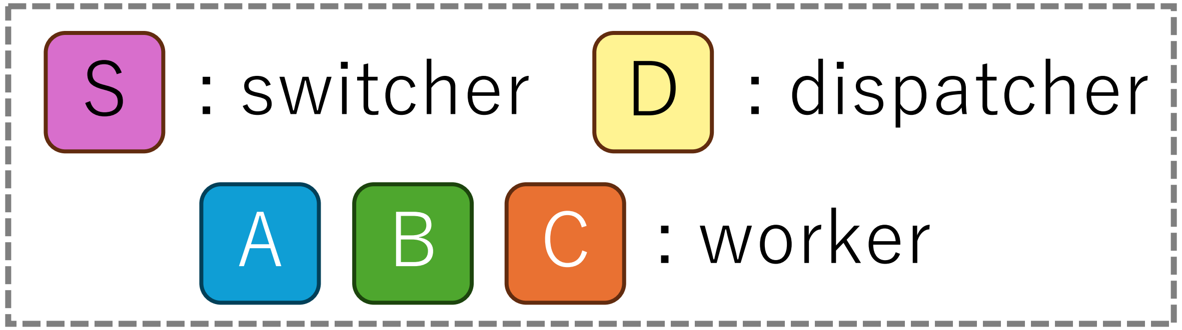
A B C : worker

Use Case: Preemptive Scheduling

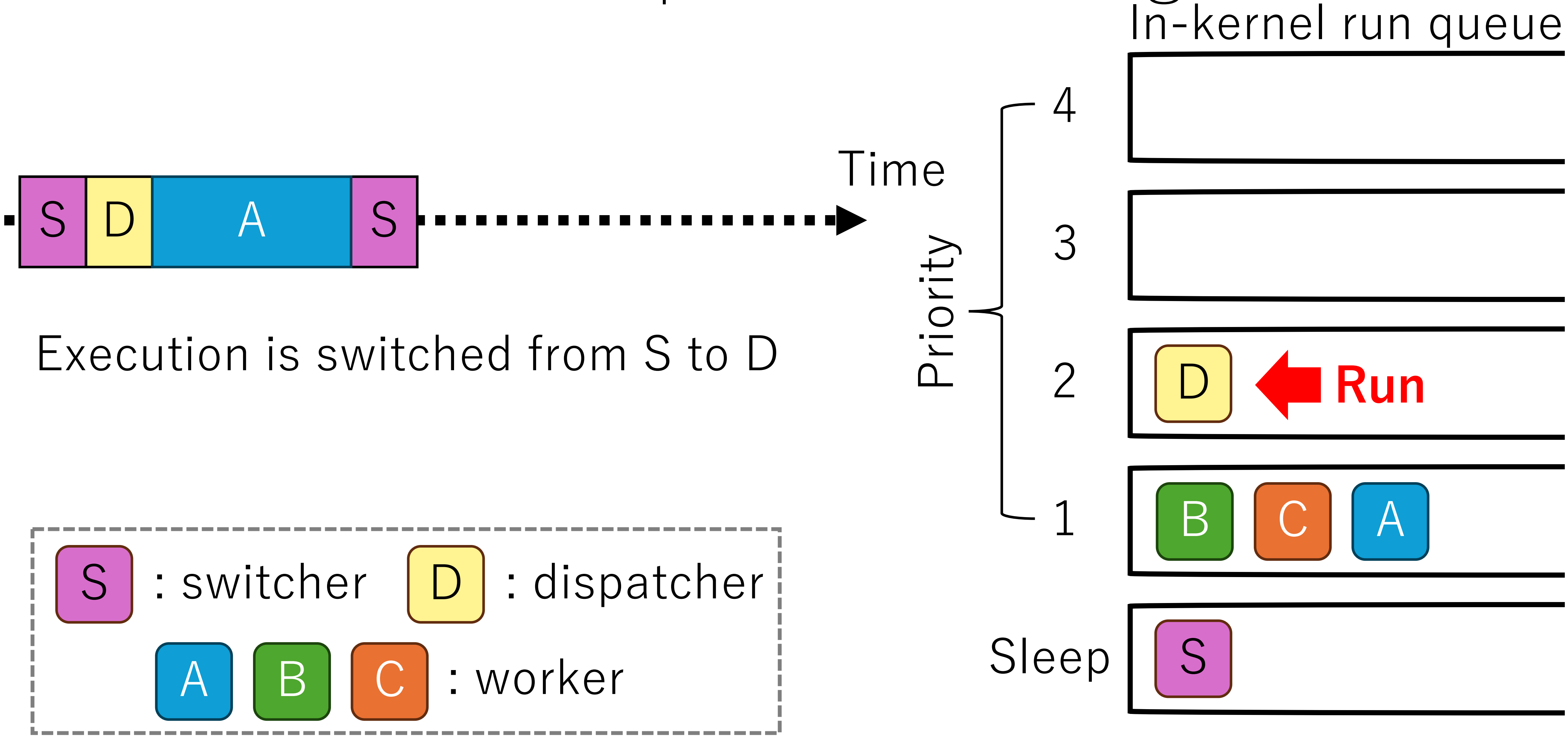


The kernel-space process scheduler picks up the highest priority process

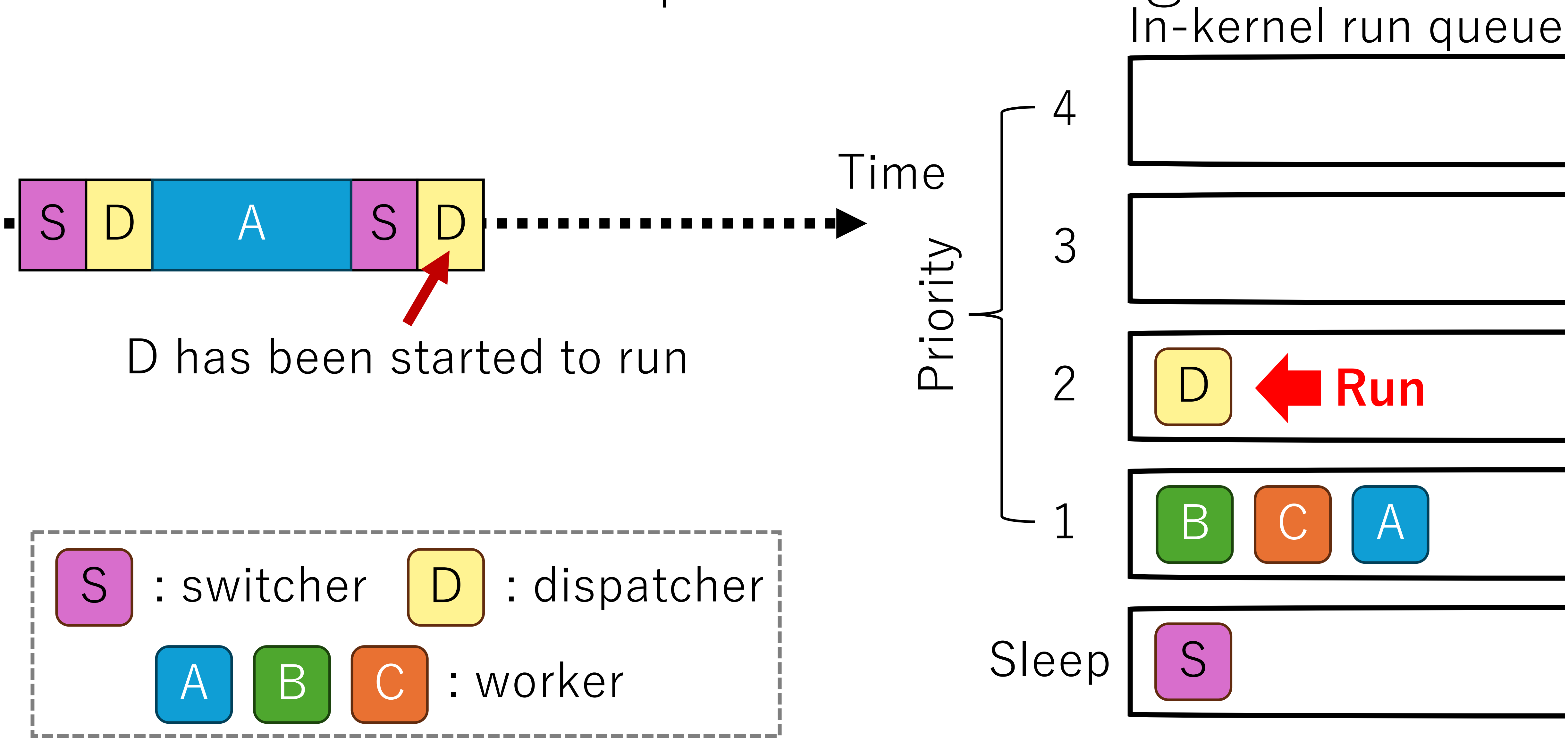




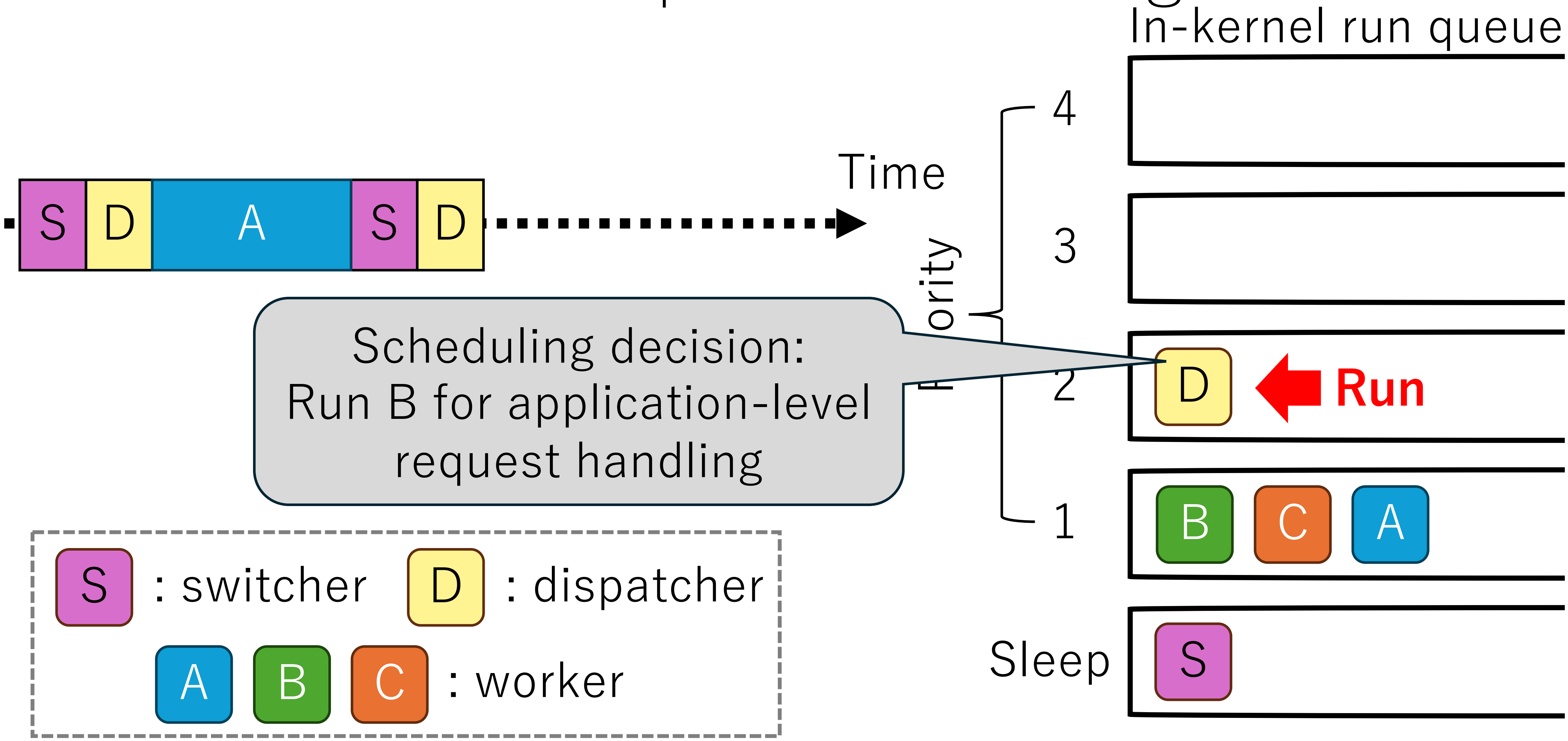
Use Case: Preemptive Scheduling



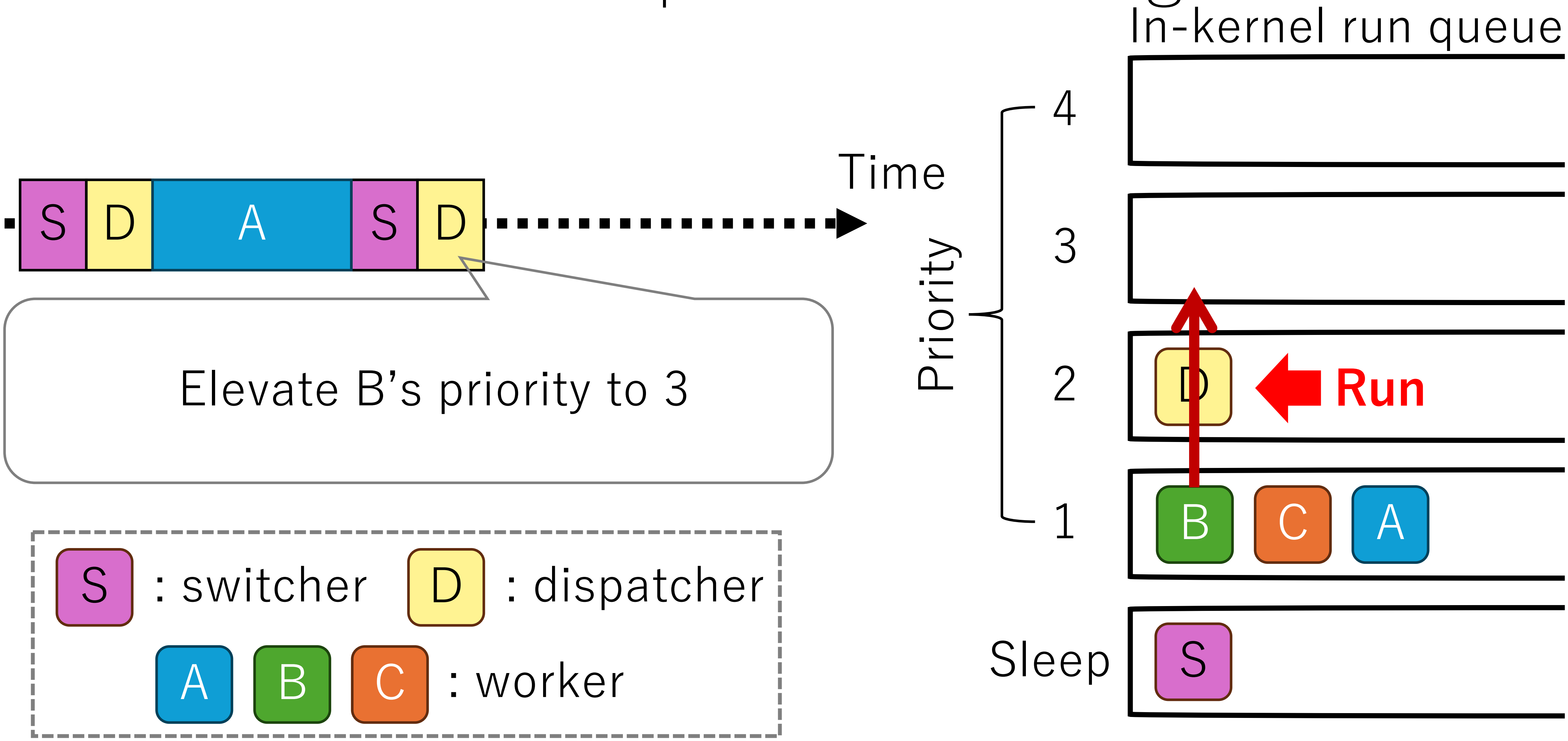
Use Case: Preemptive Scheduling



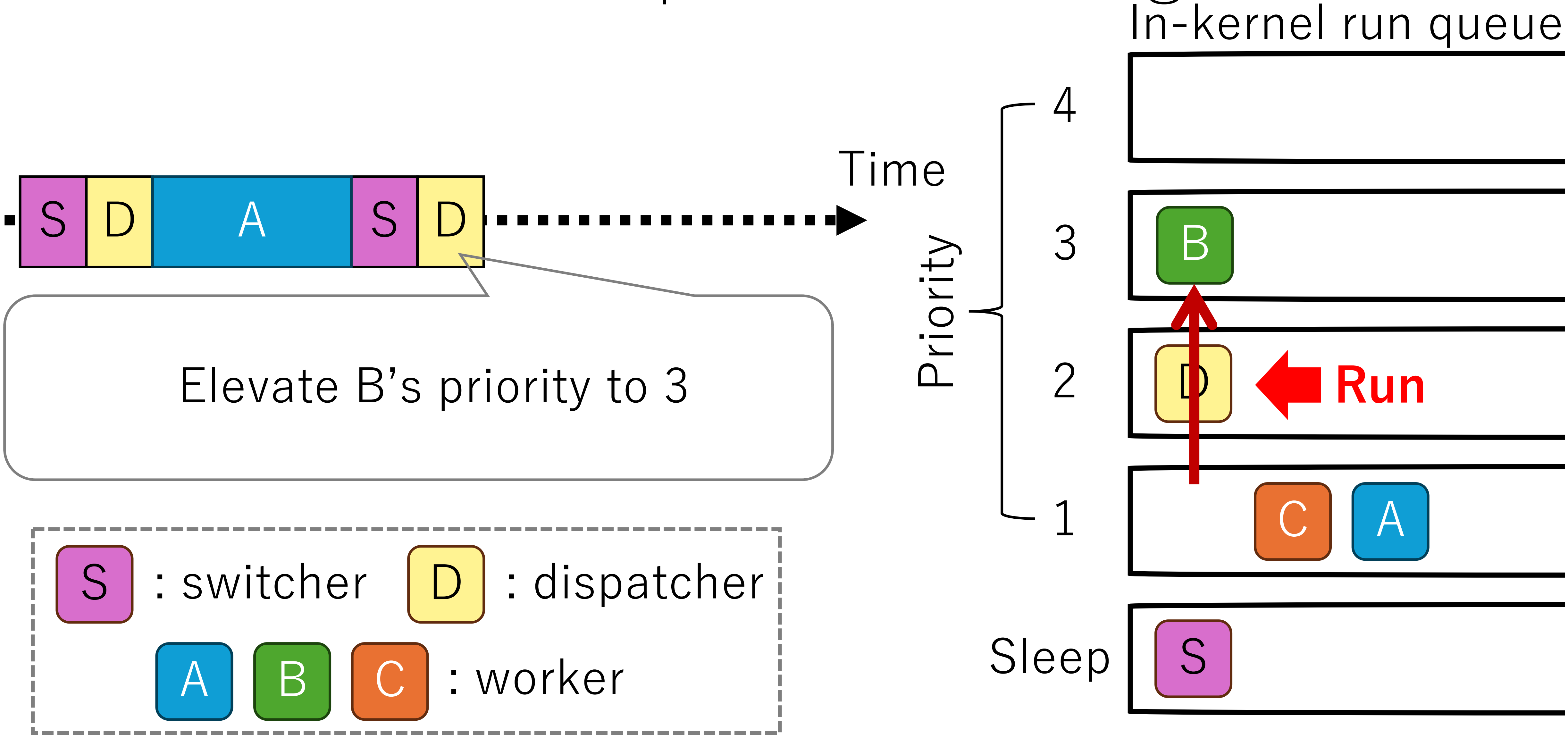
Use Case: Preemptive Scheduling



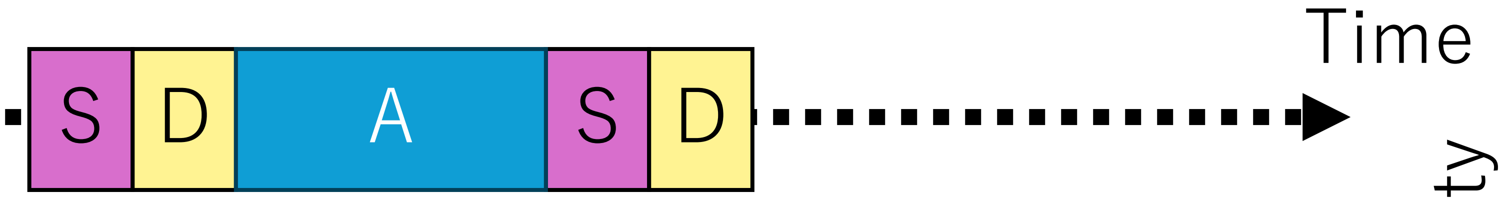
Use Case: Preemptive Scheduling



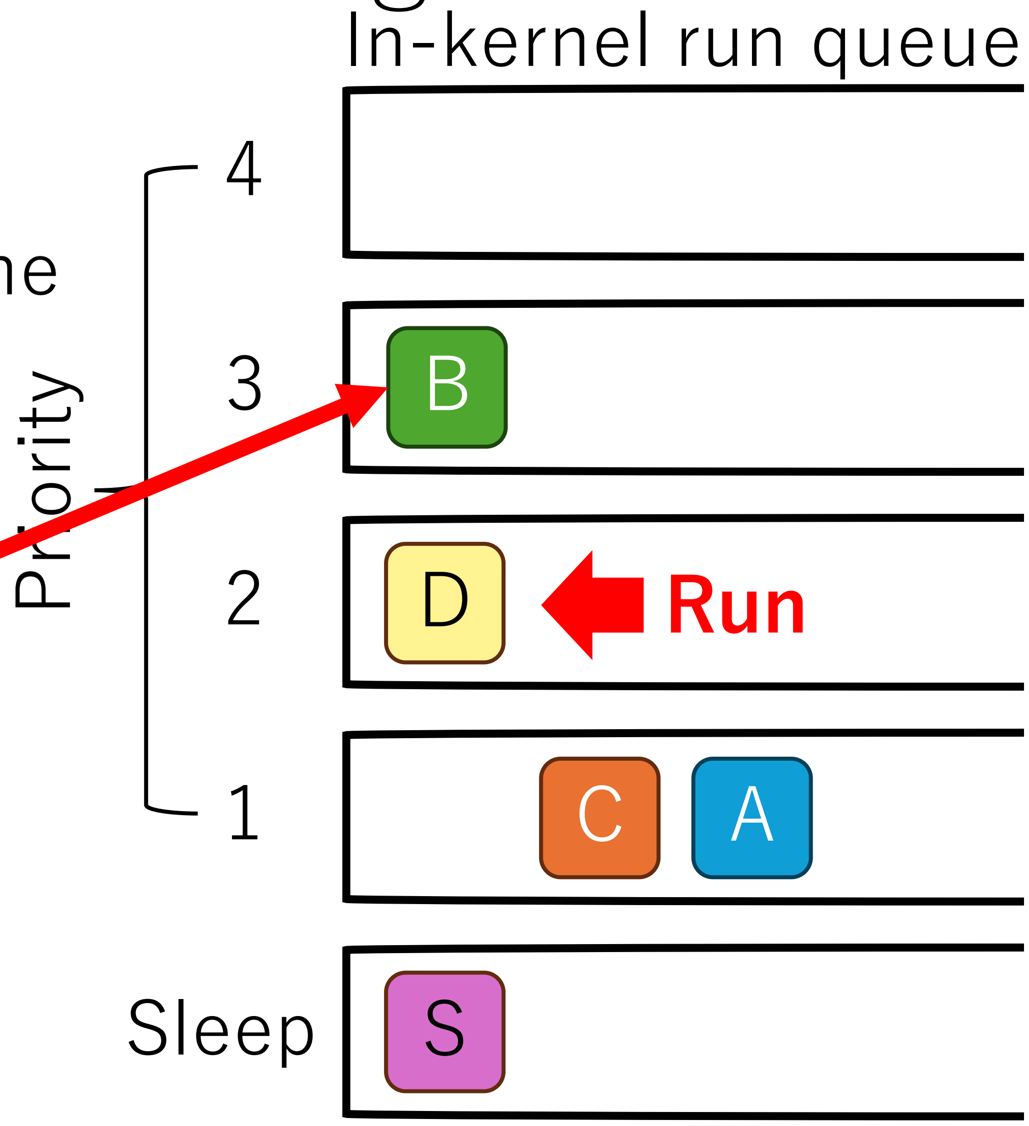
Use Case: Preemptive Scheduling

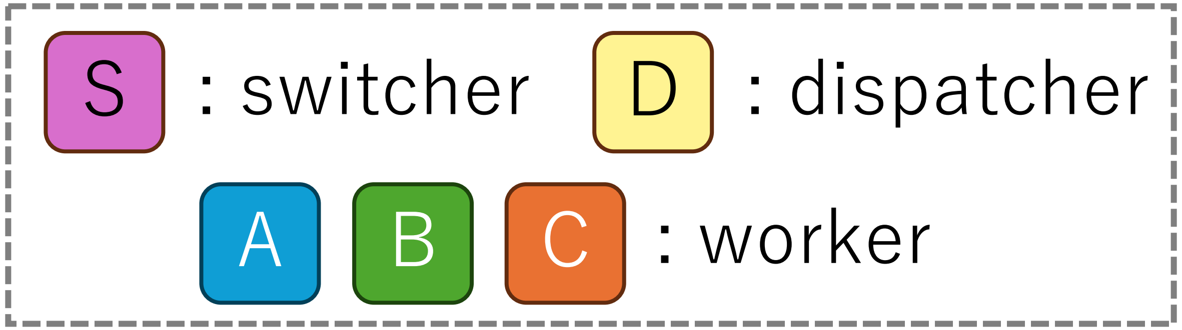


Use Case: Preemptive Scheduling

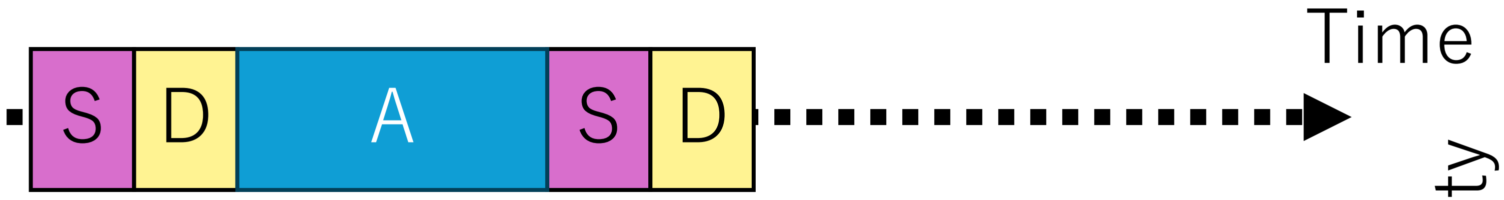


The kernel-space process scheduler picks up the highest priority process

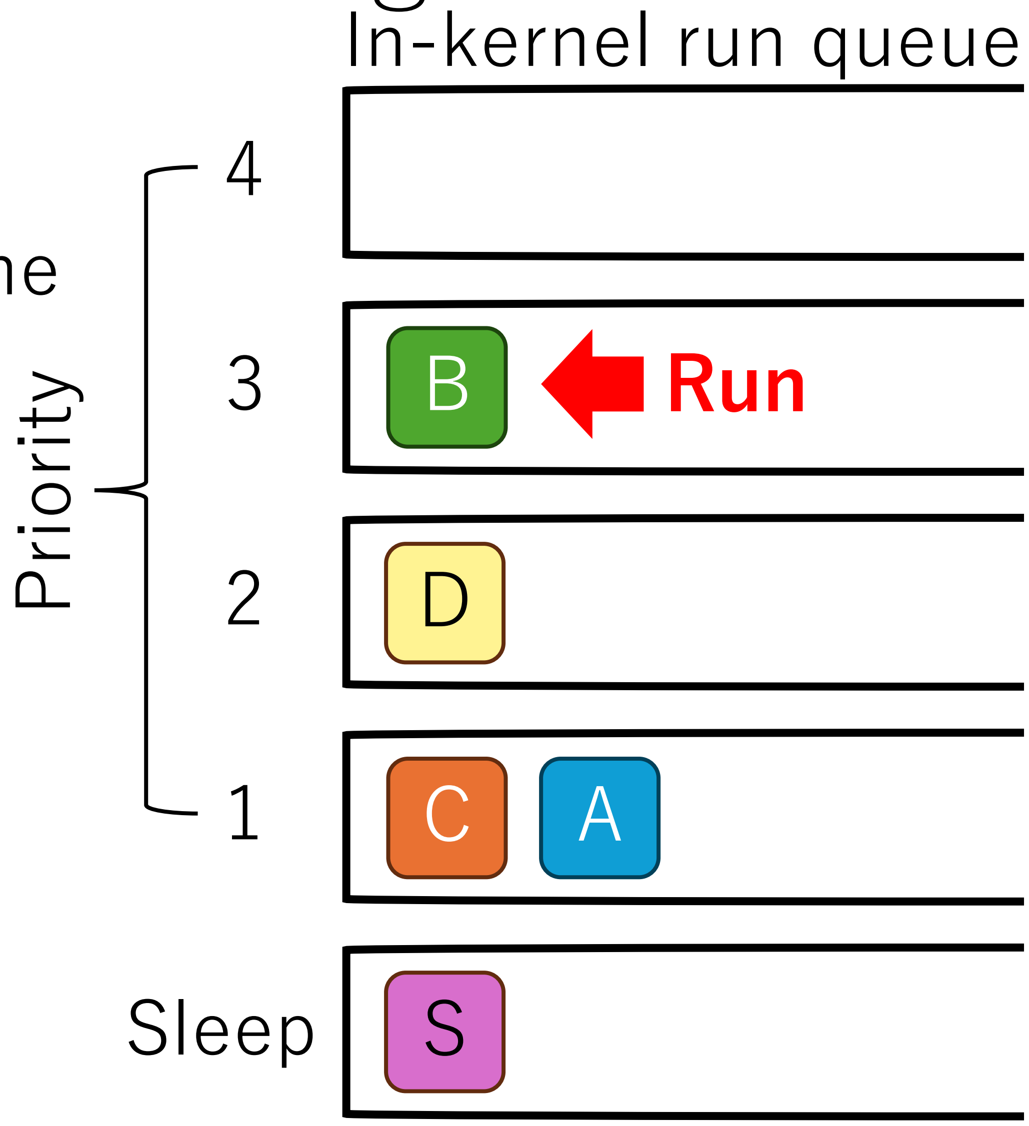


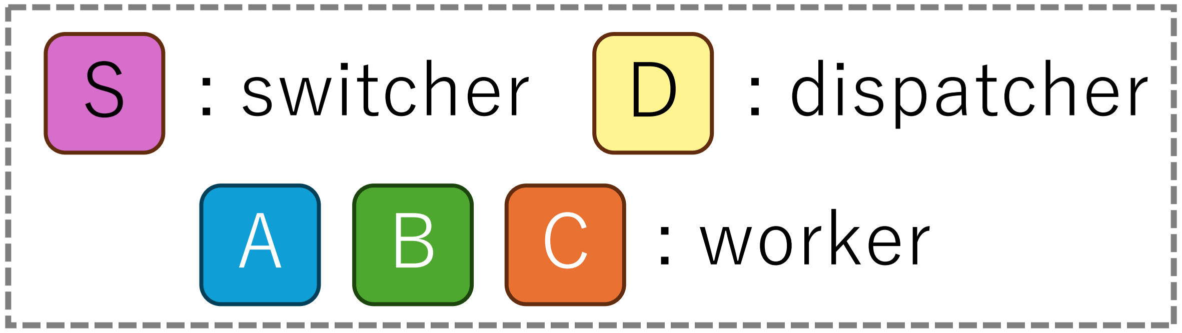


Use Case: Preemptive Scheduling

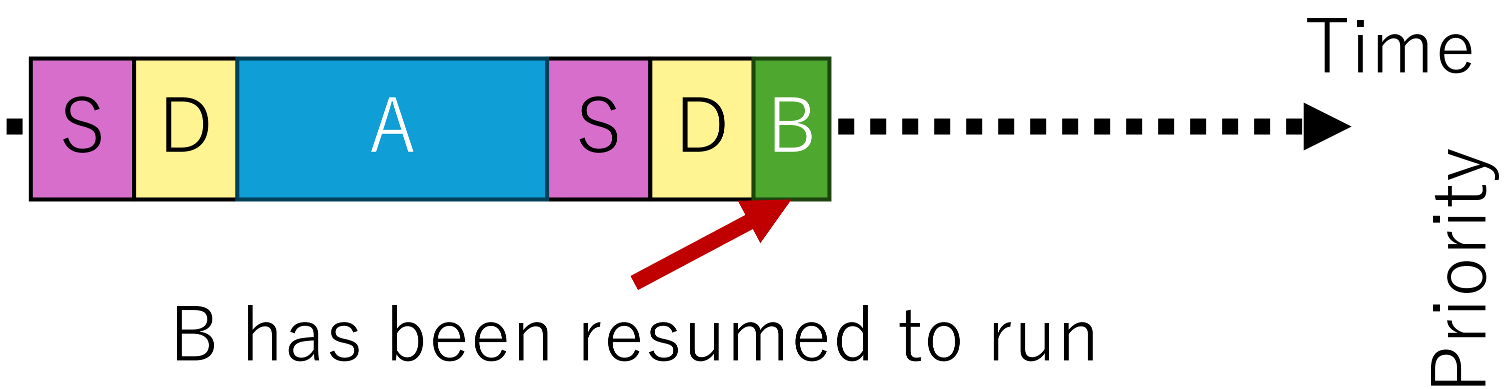


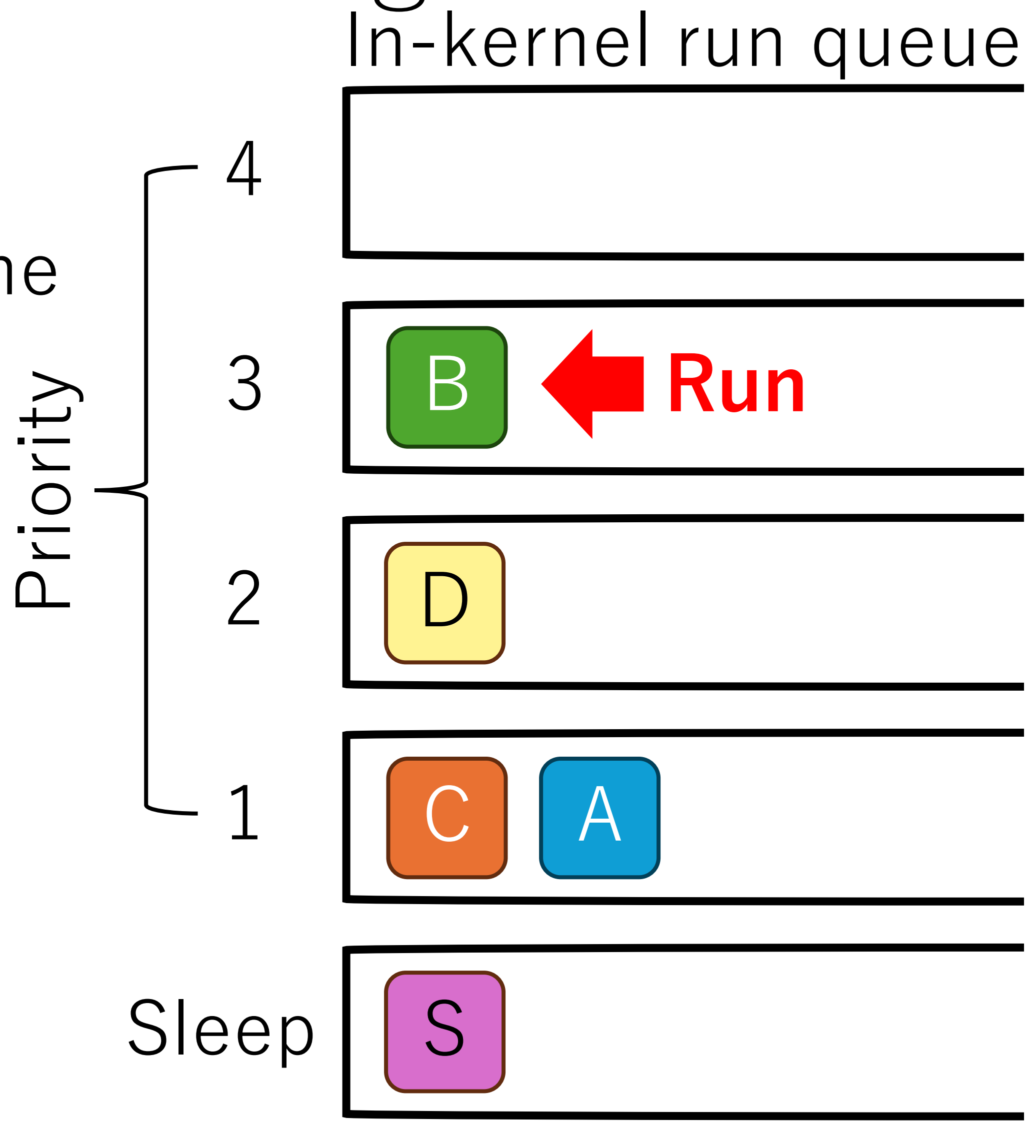
Execution is switched from D to B

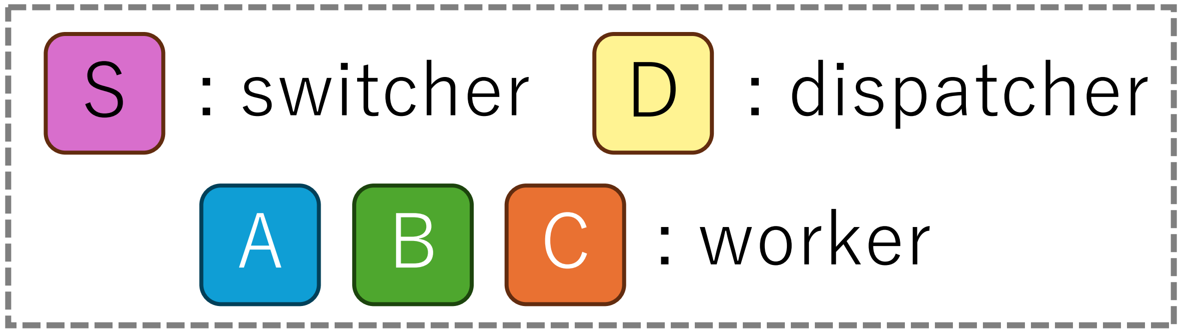




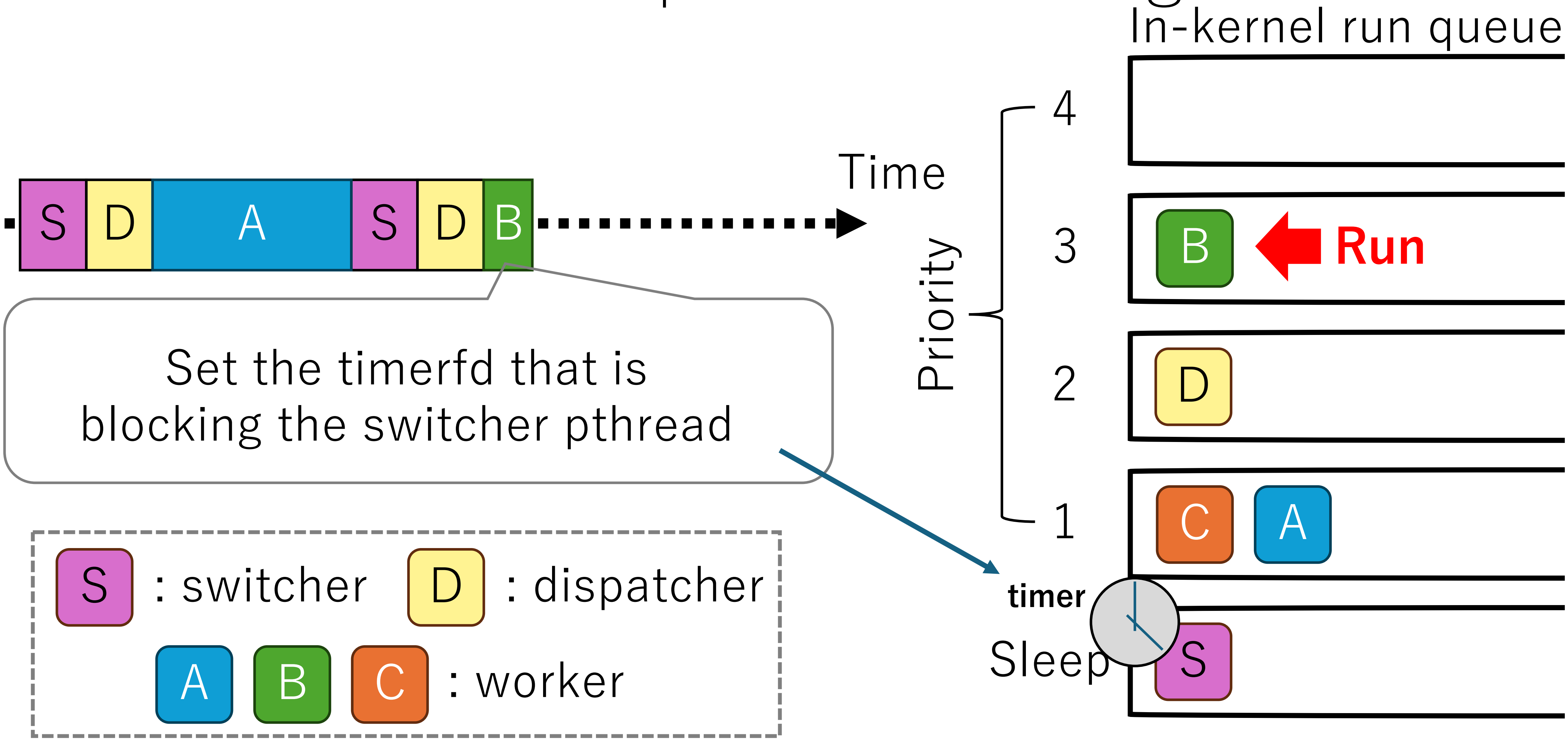
Use Case: Preemptive Scheduling



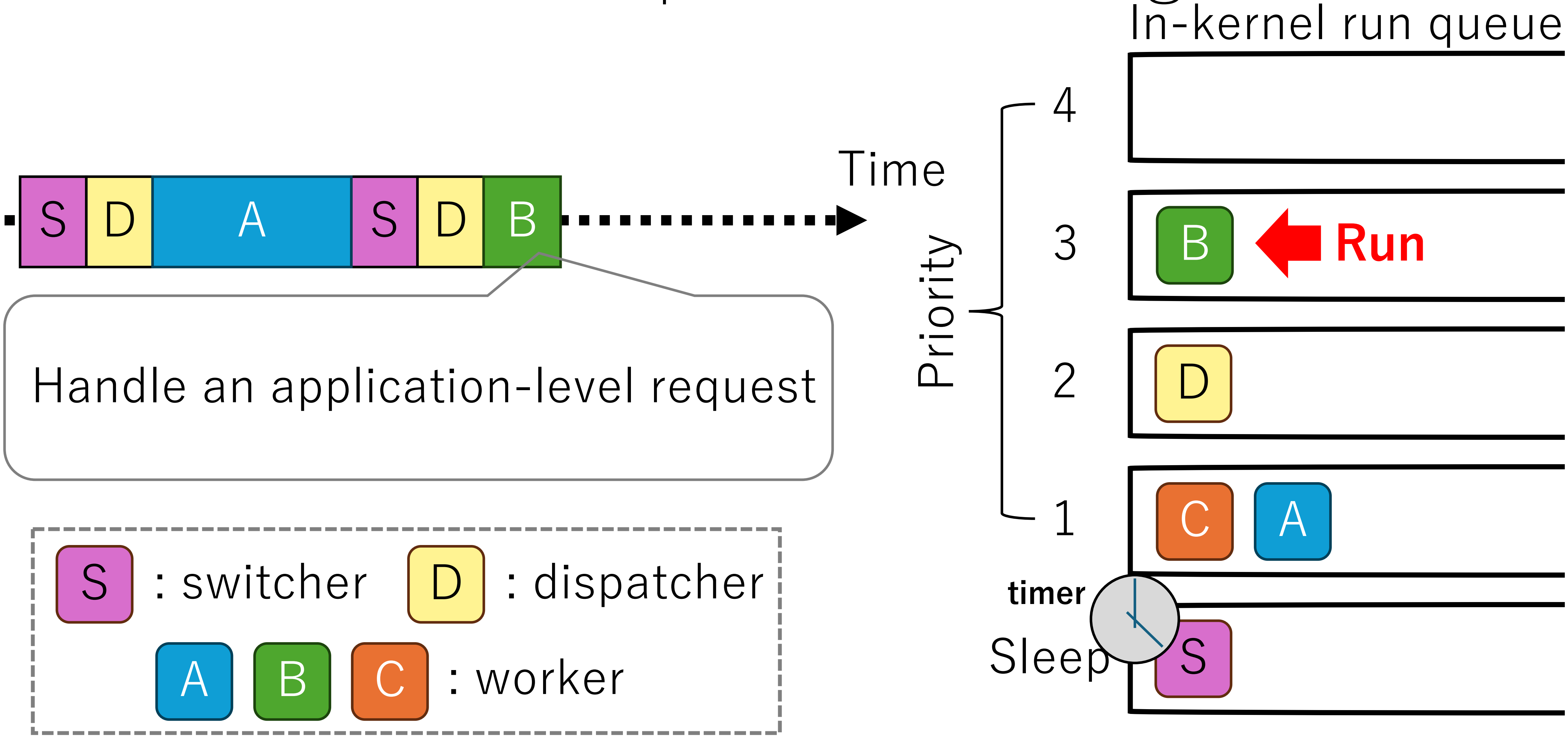




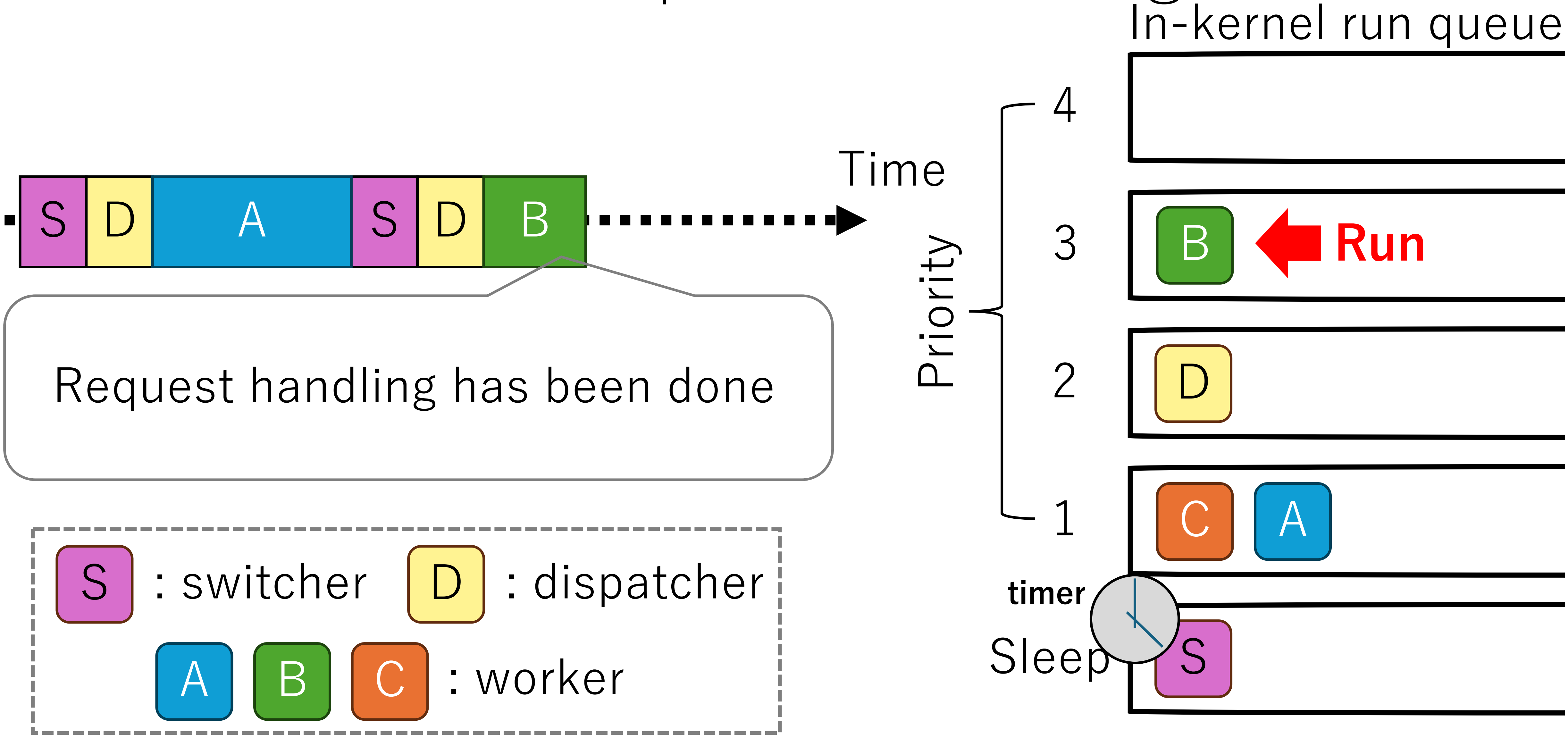
Use Case: Preemptive Scheduling



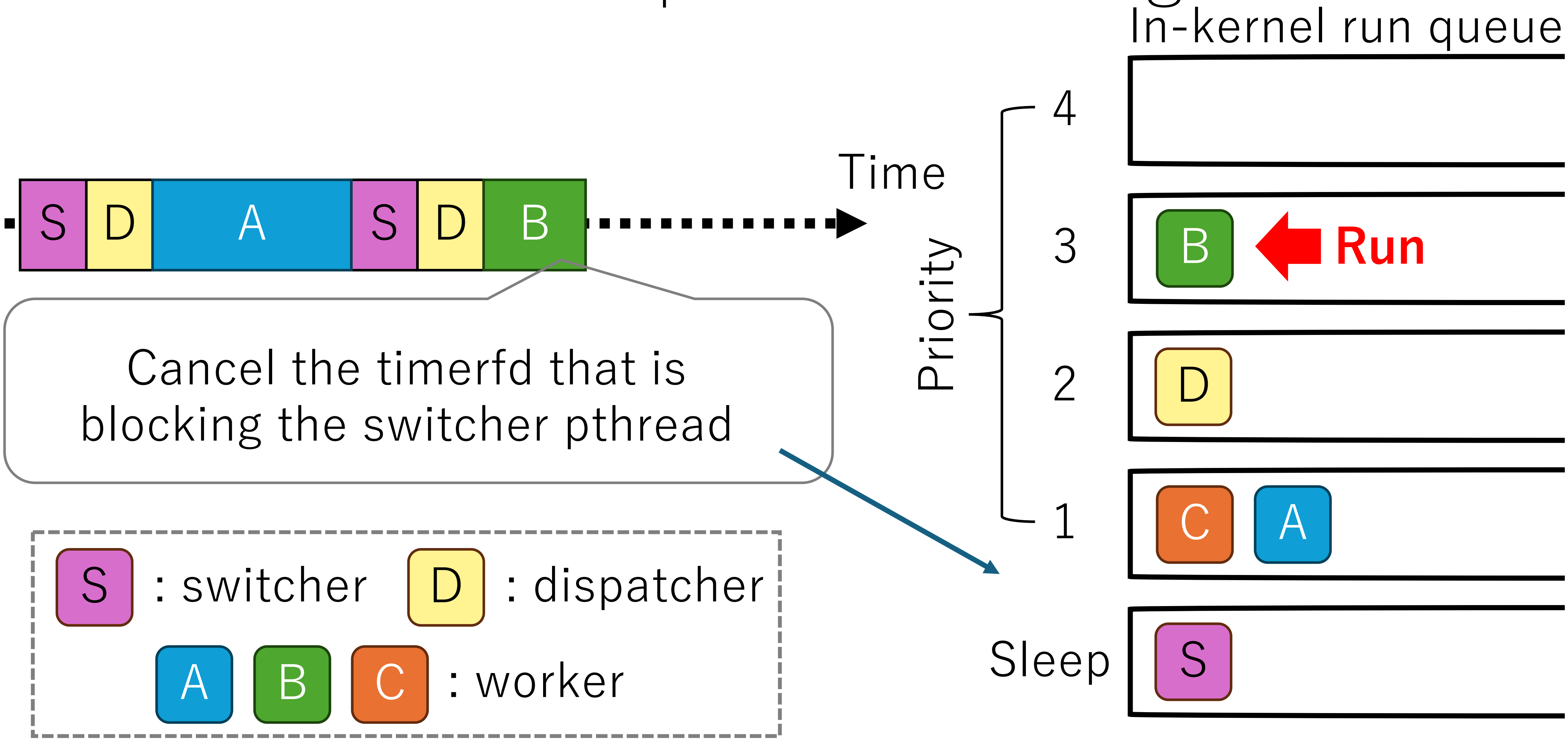
Use Case: Preemptive Scheduling



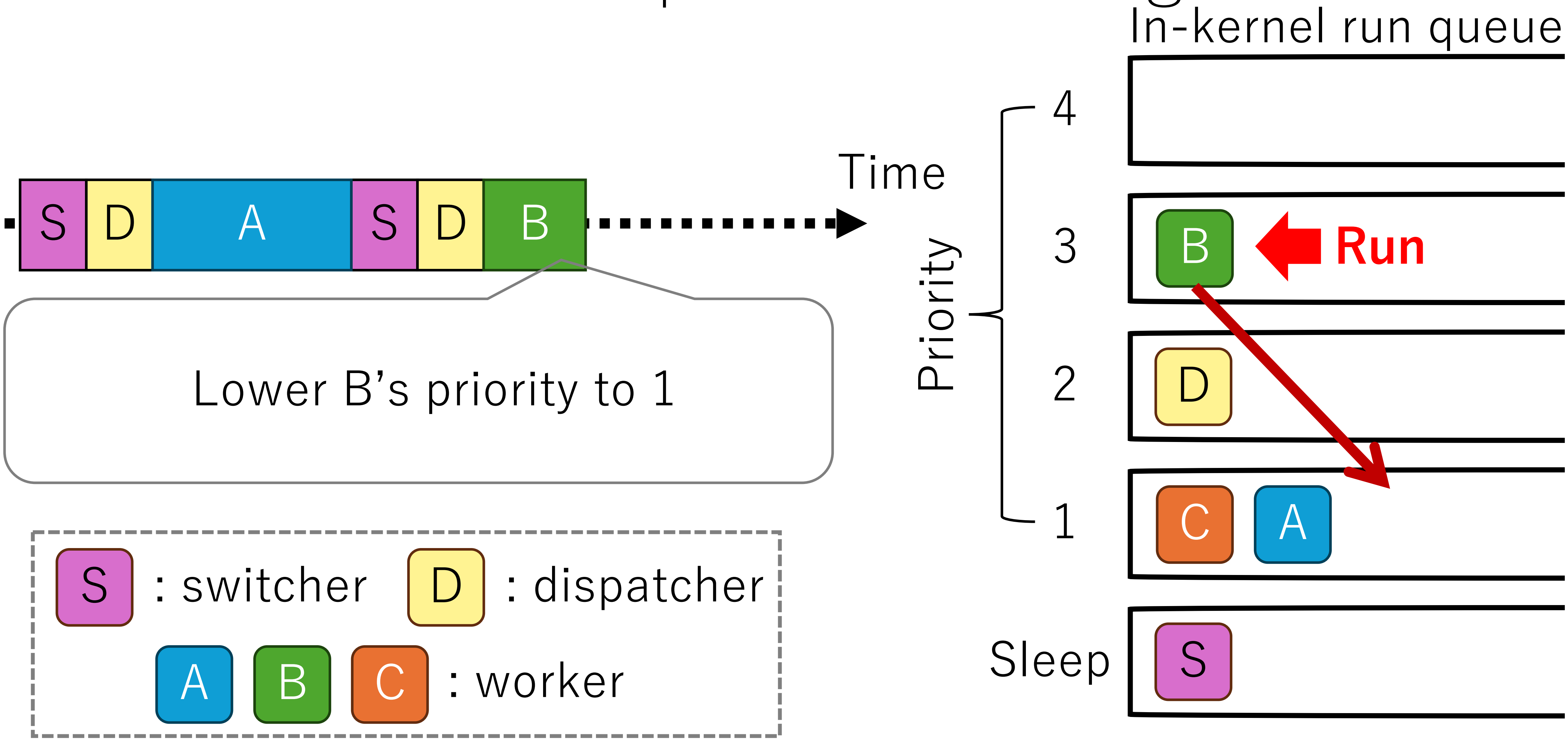
Use Case: Preemptive Scheduling



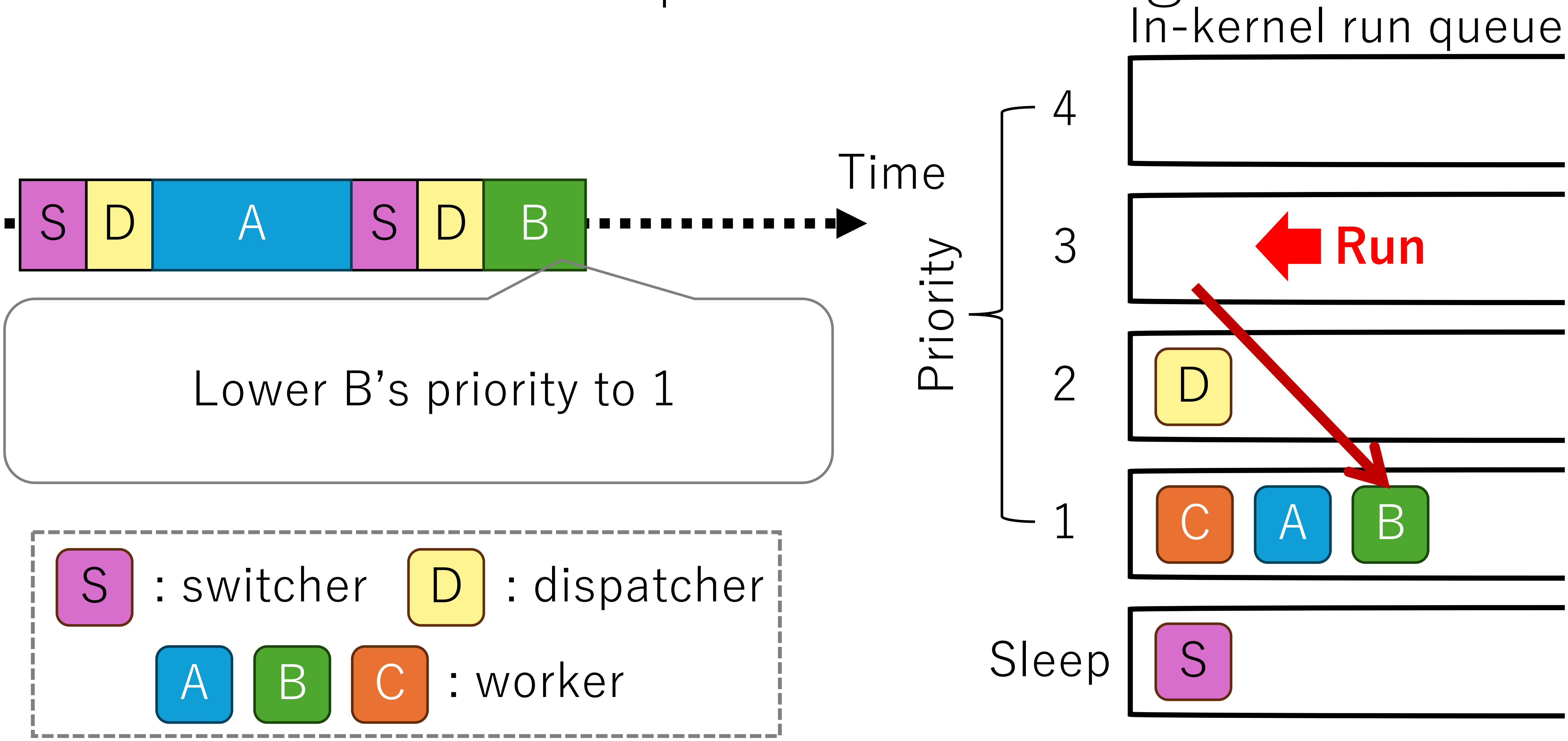
Use Case: Preemptive Scheduling



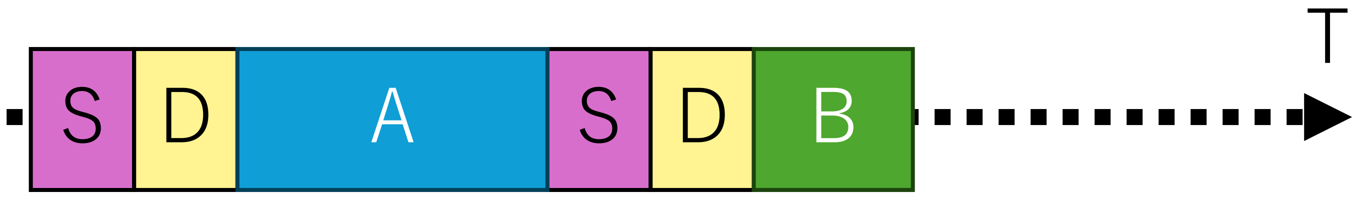
Use Case: Preemptive Scheduling

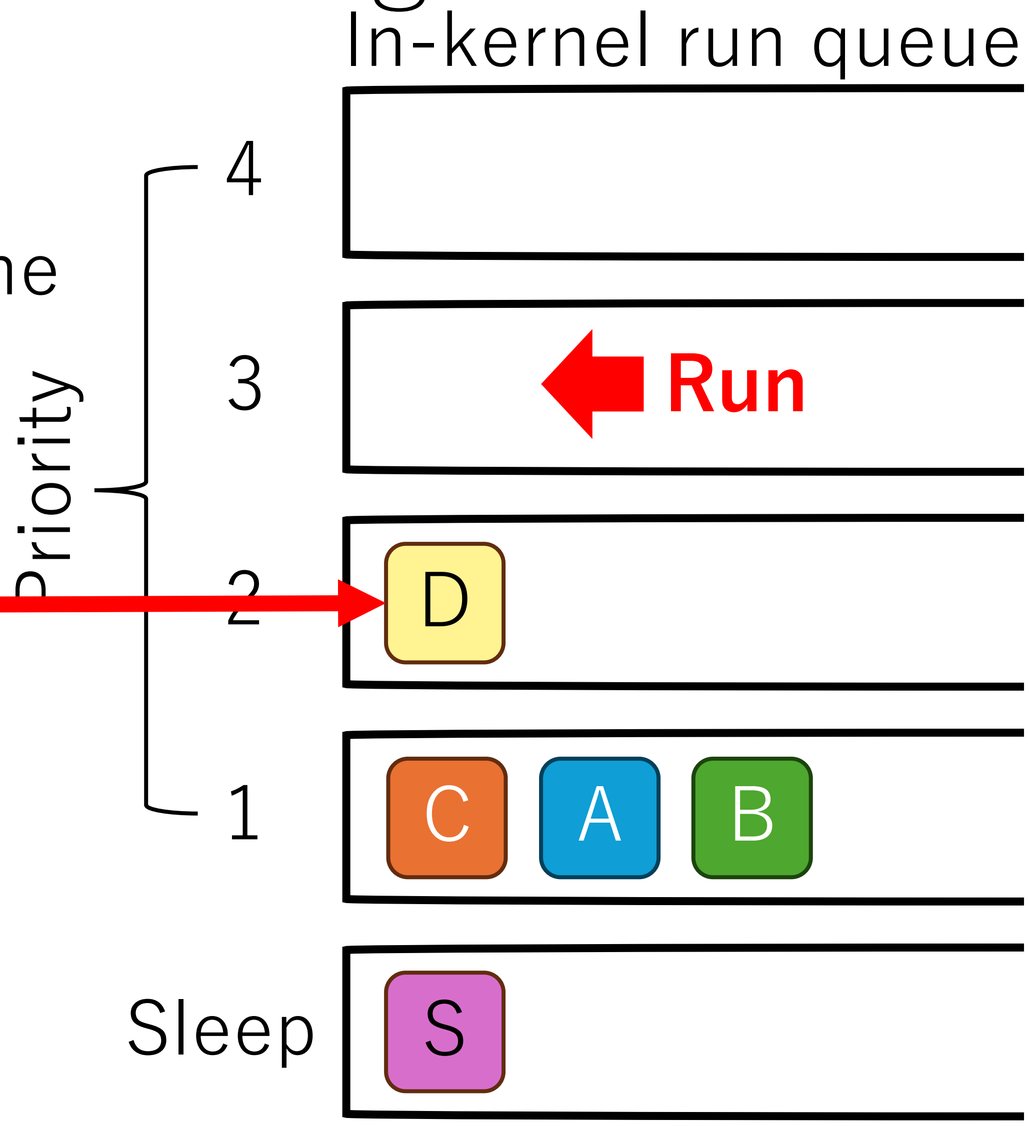


Use Case: Preemptive Scheduling

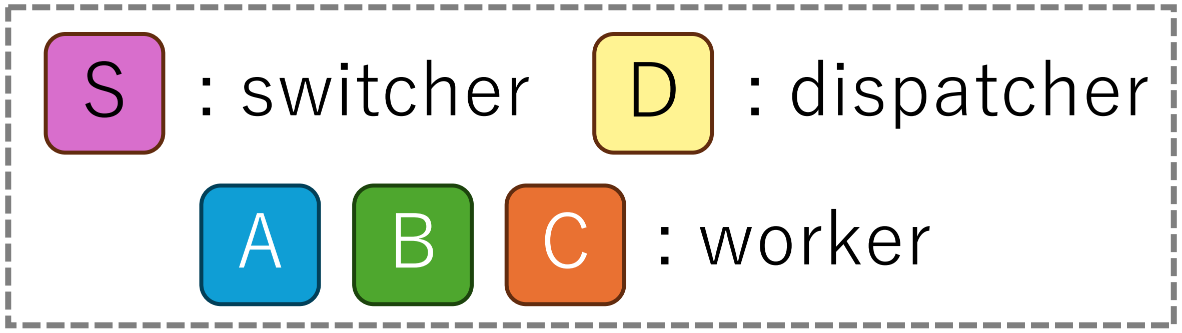


Use Case: Preemptive Scheduling

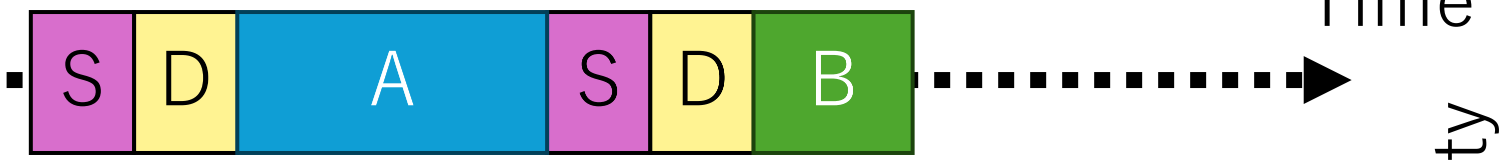




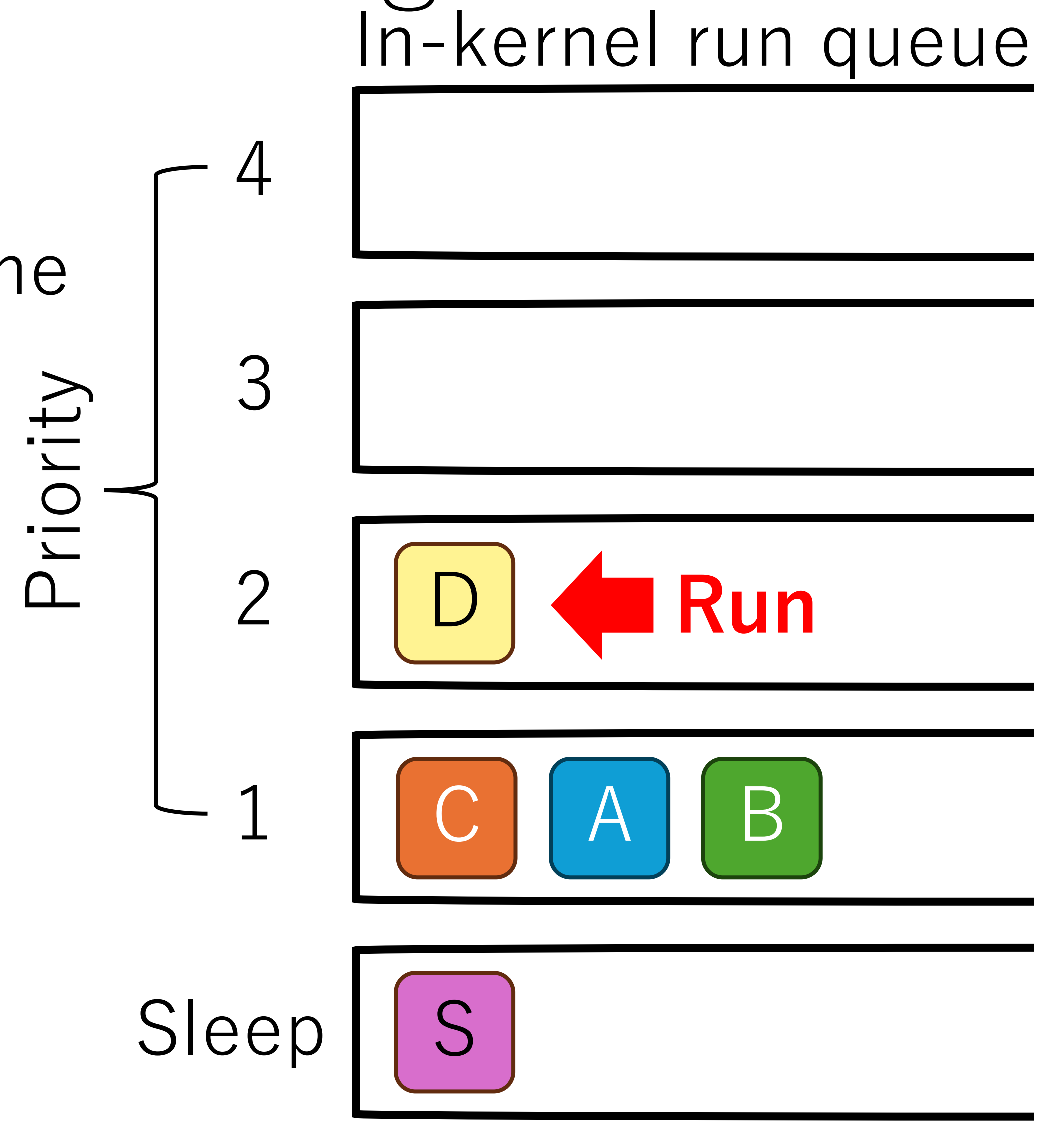
The kernel-space process scheduler picks up the highest priority process

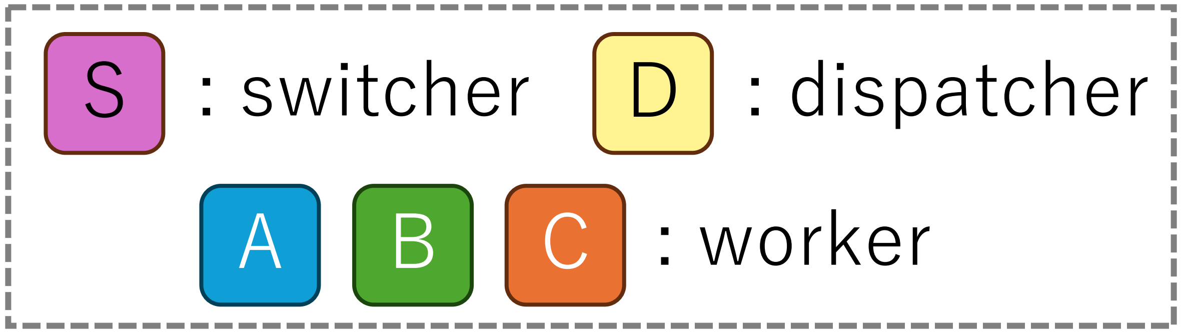


Use Case: Preemptive Scheduling

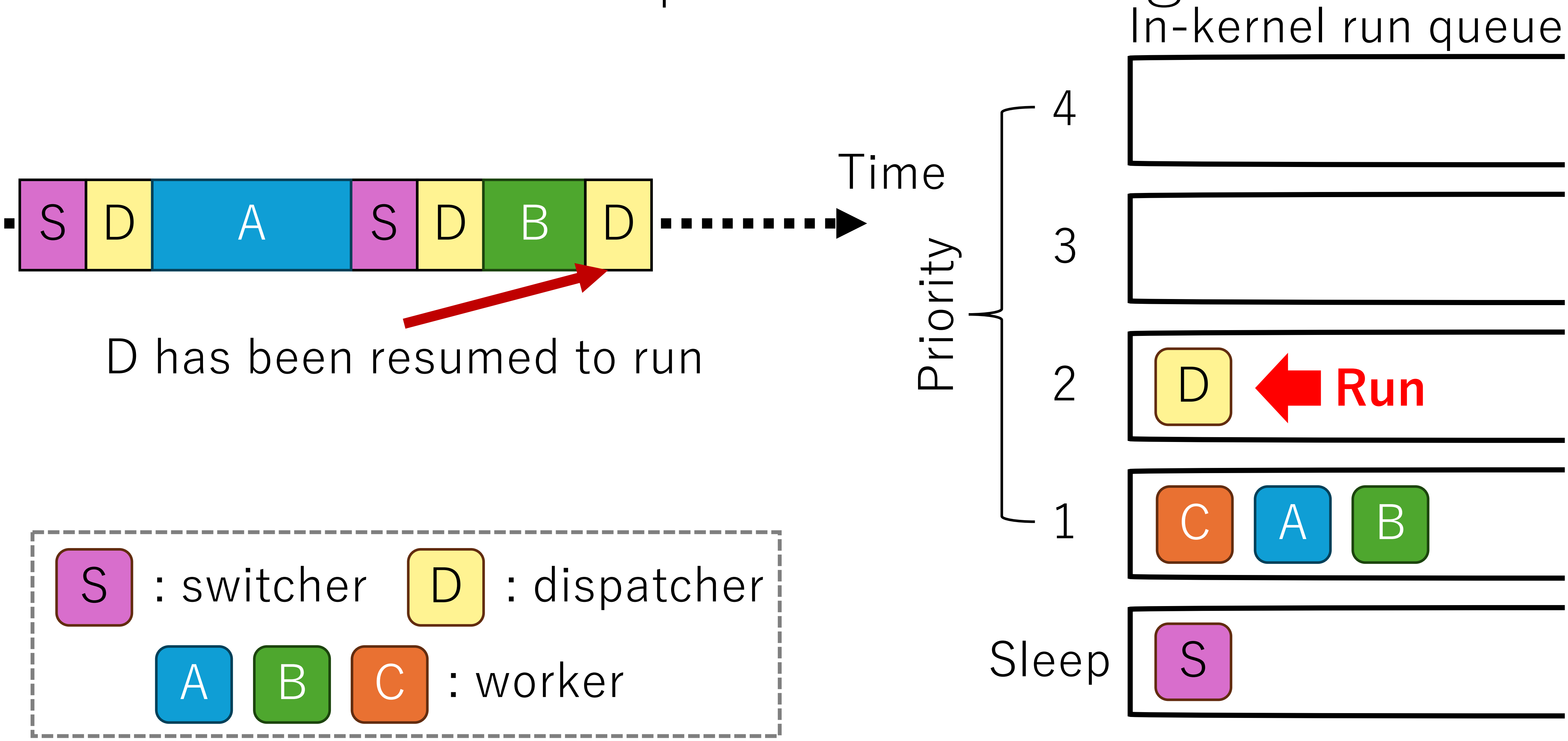


Execution is switched from B to D

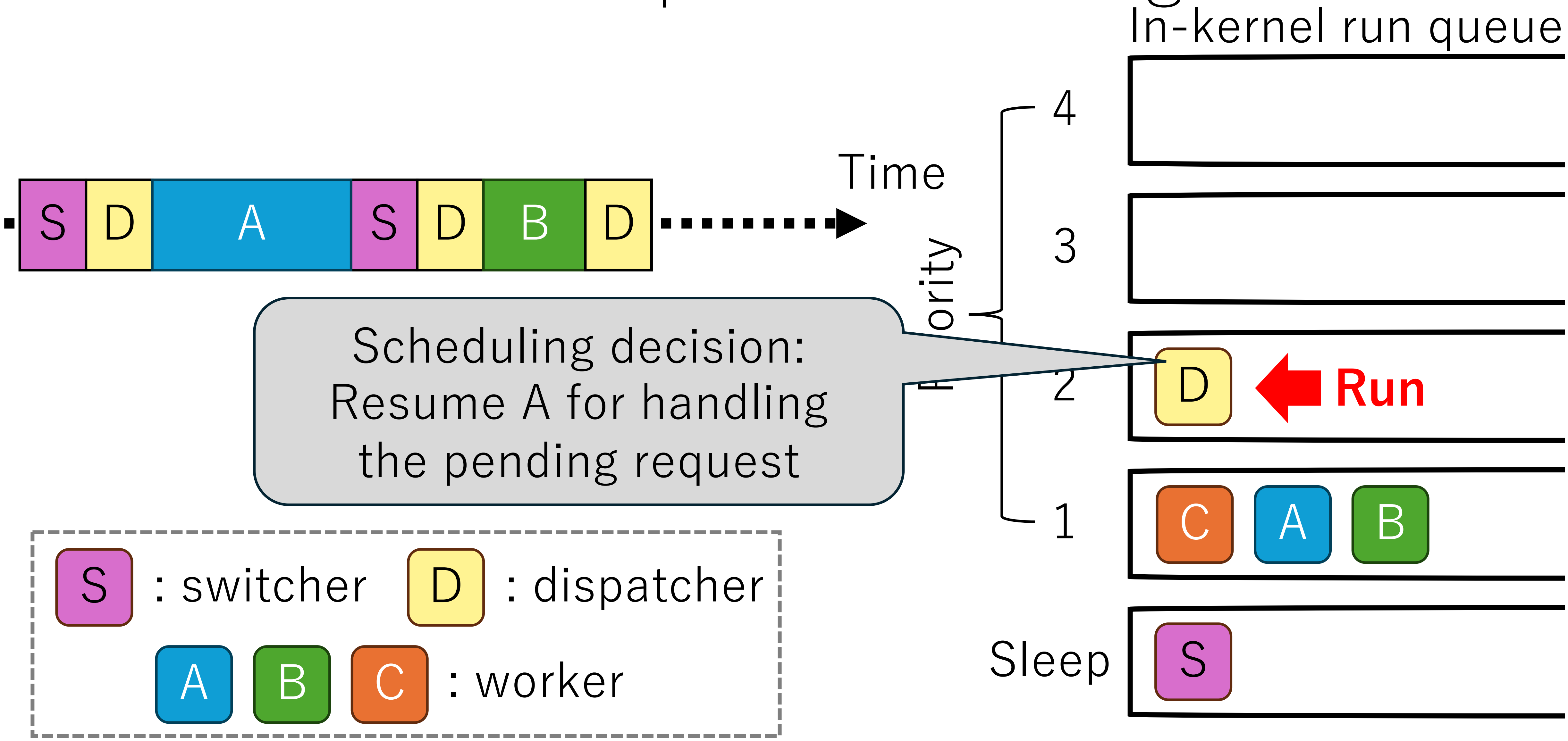




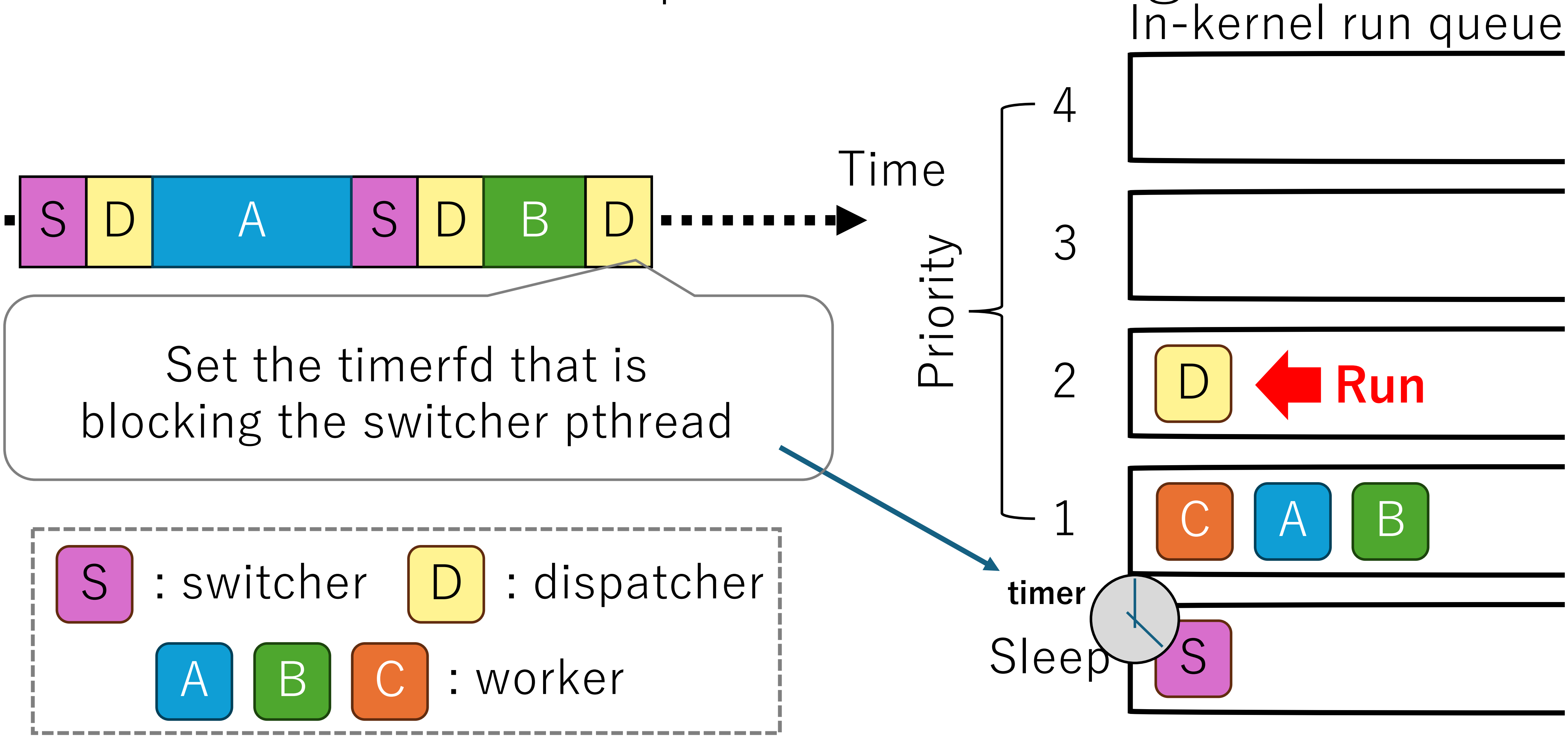
Use Case: Preemptive Scheduling



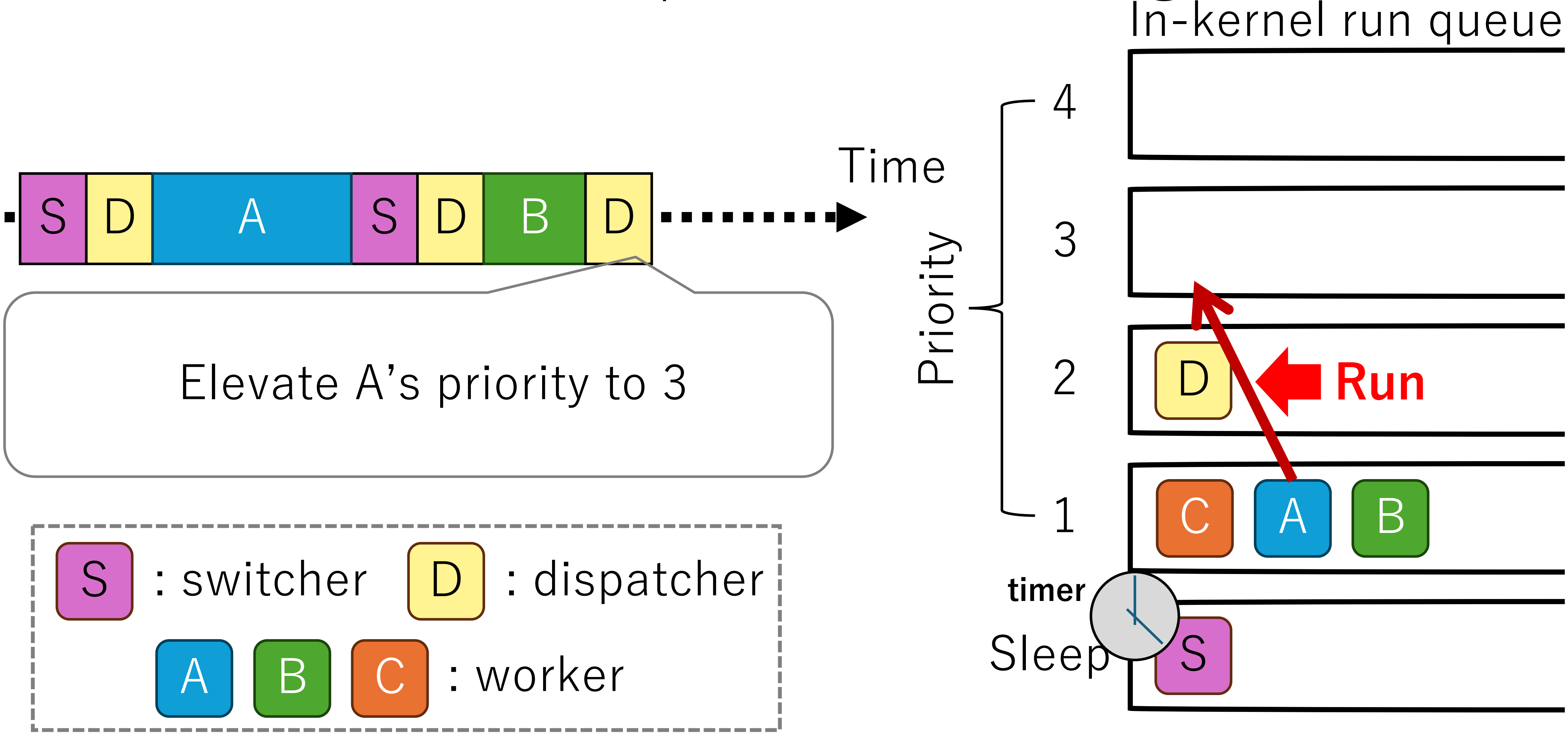
Use Case: Preemptive Scheduling



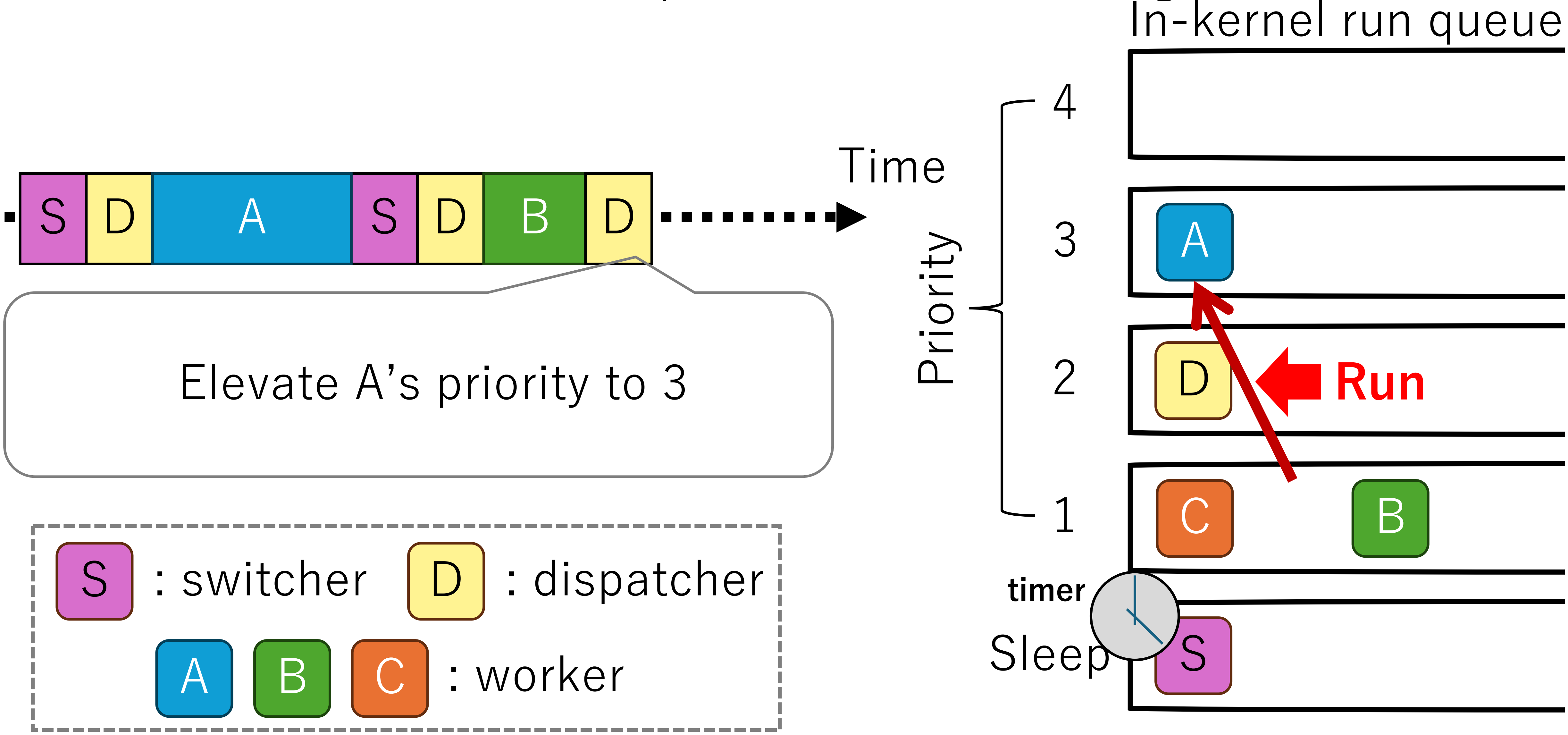
Use Case: Preemptive Scheduling



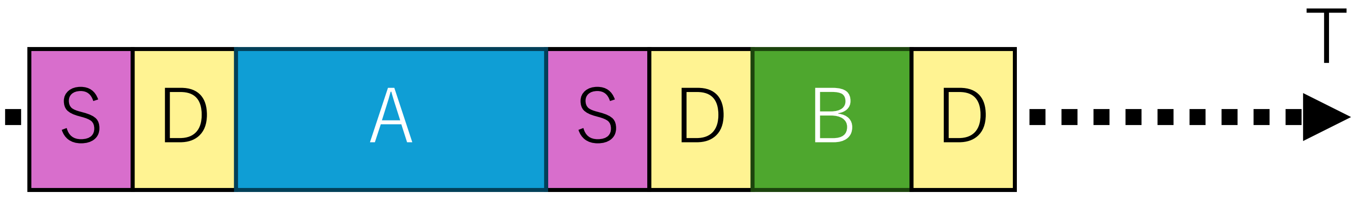
Use Case: Preemptive Scheduling

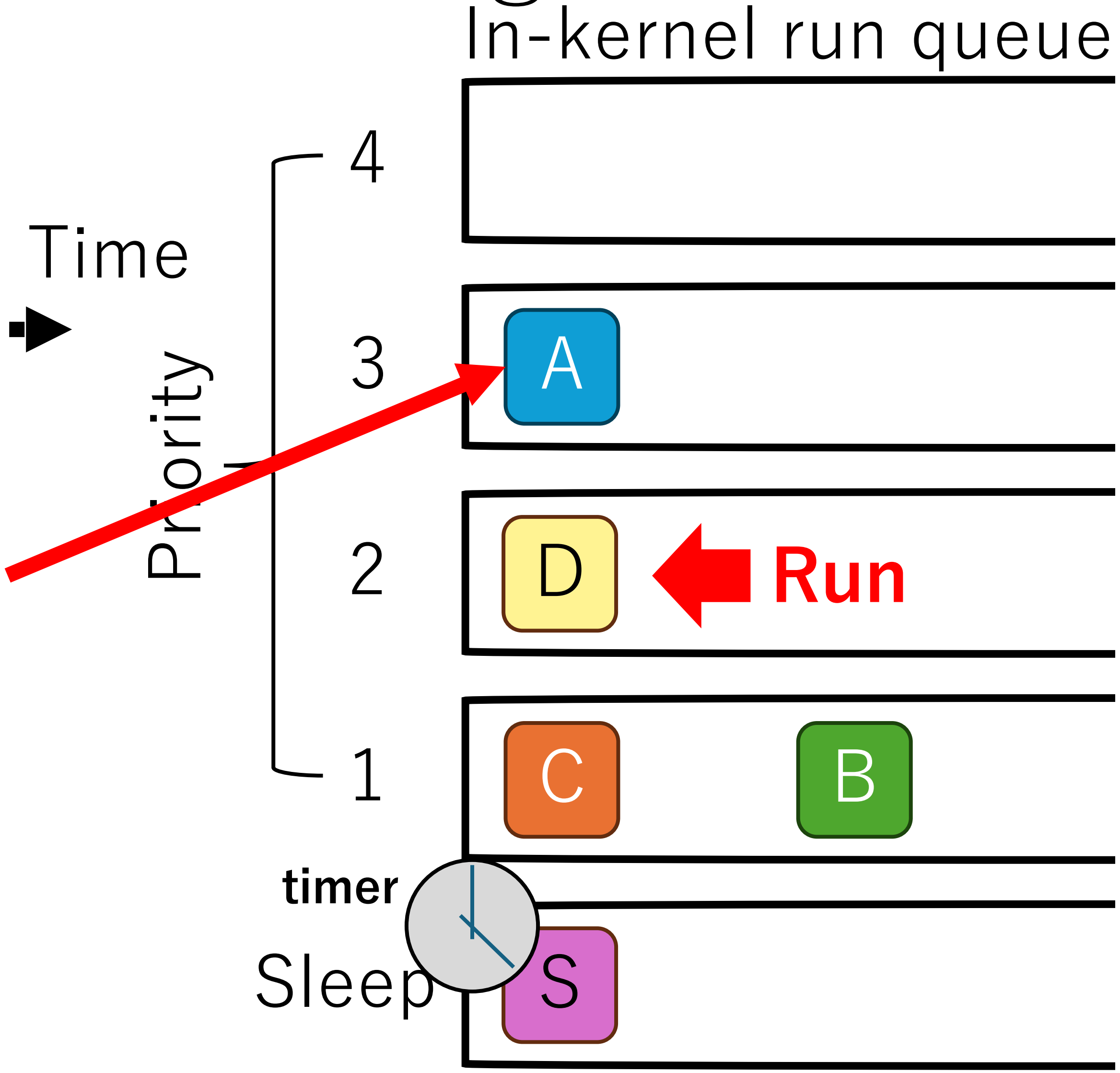


Use Case: Preemptive Scheduling

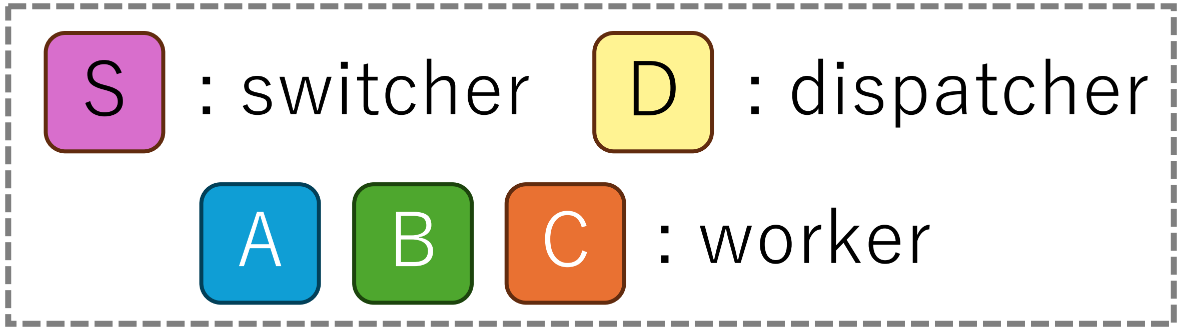


Use Case: Preemptive Scheduling

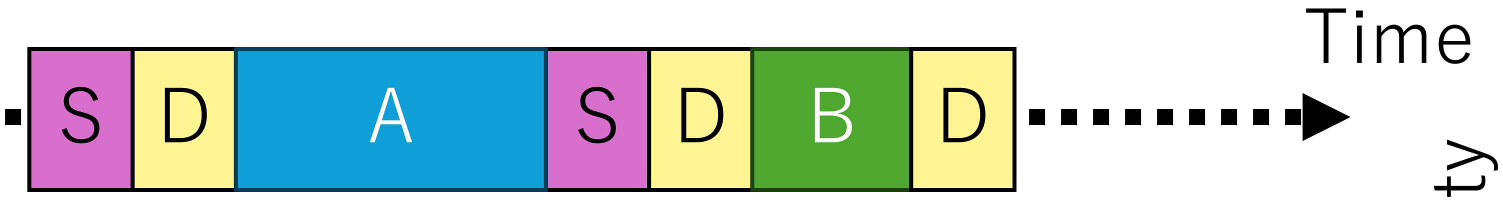




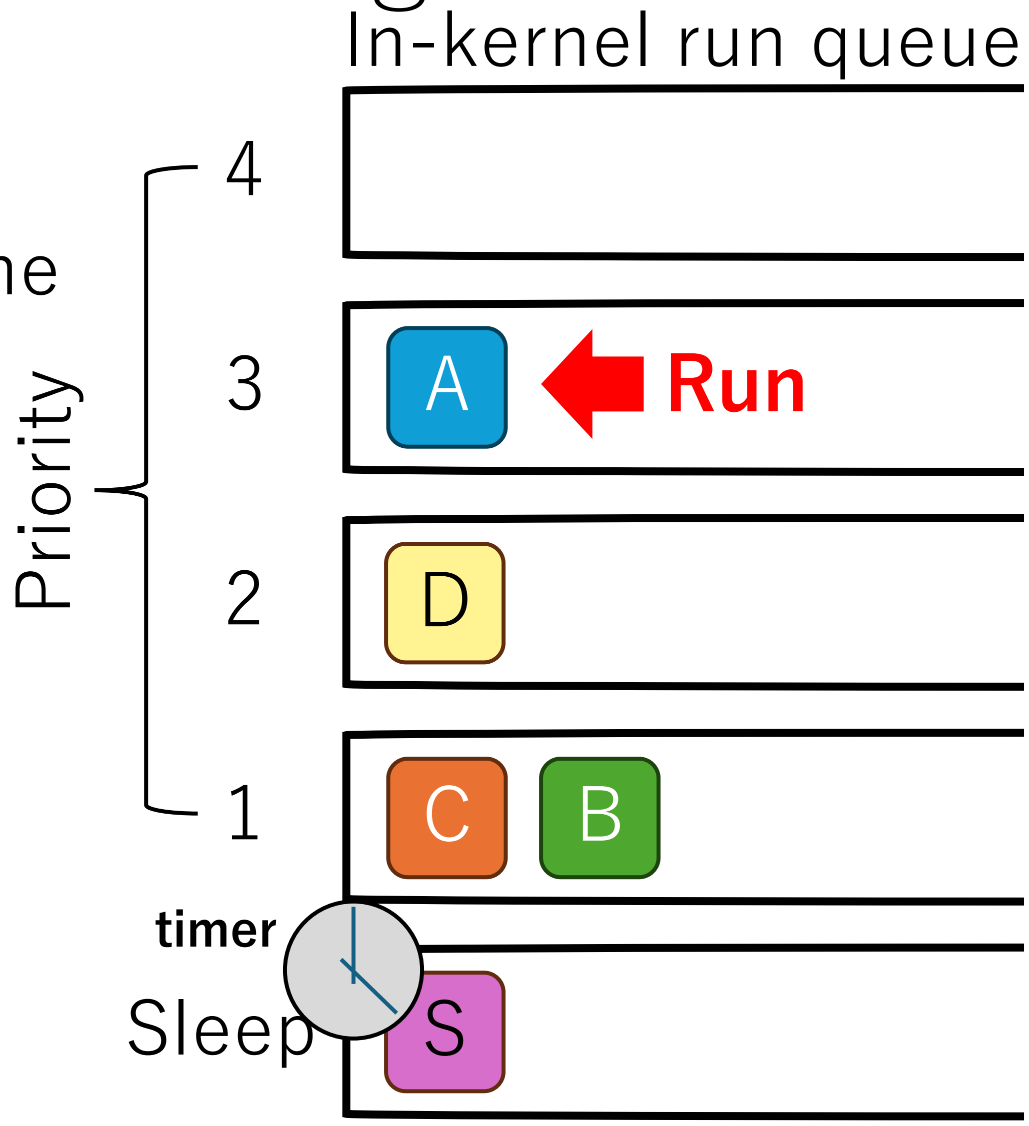
The kernel-space process scheduler picks up the highest priority process

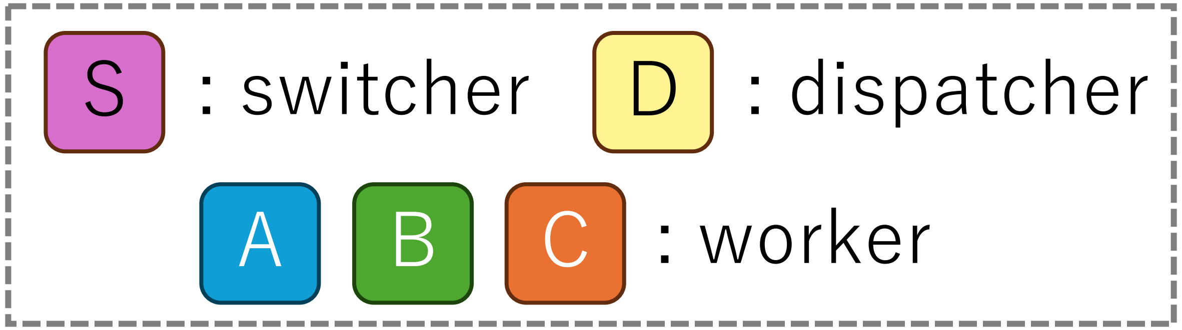


Use Case: Preemptive Scheduling

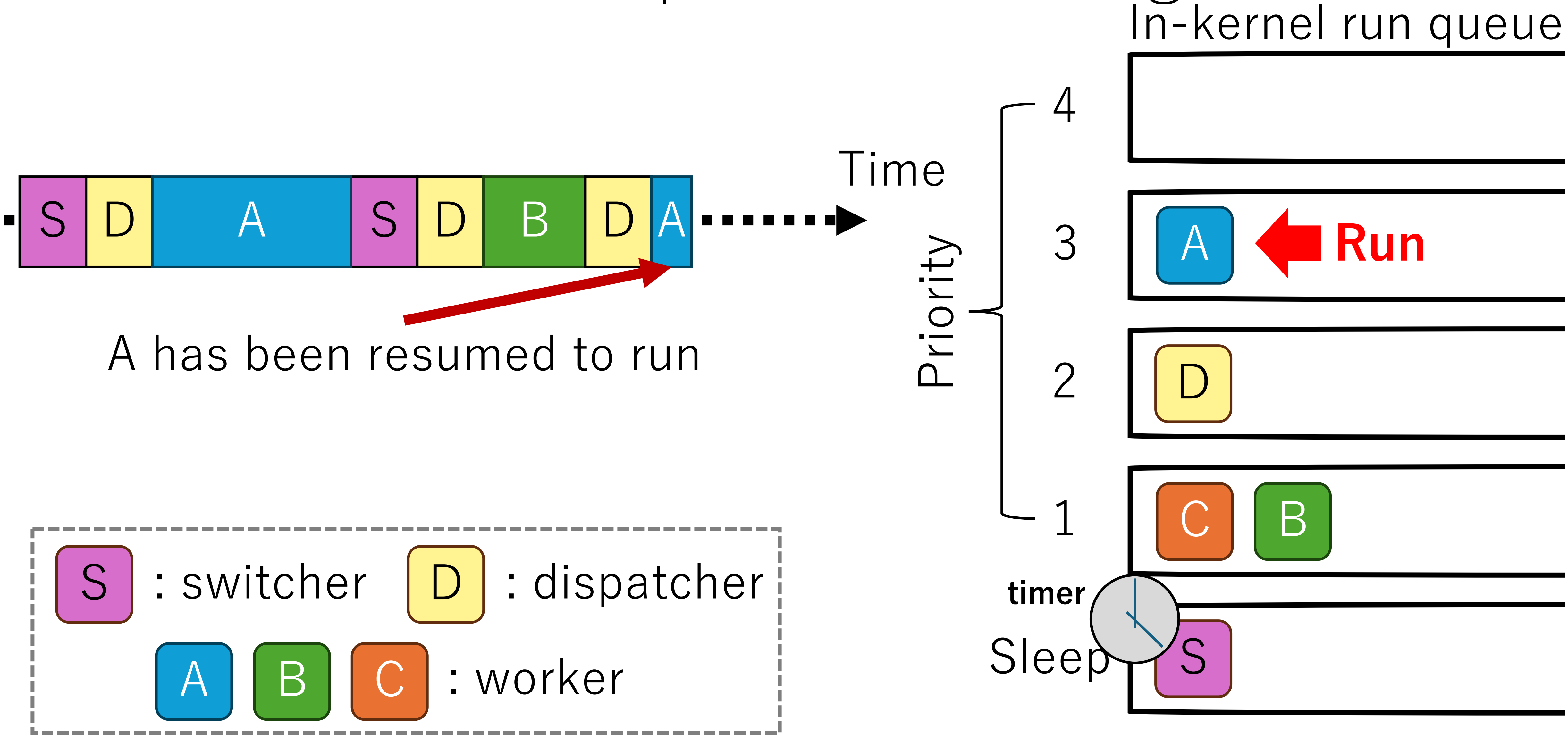


Execution is switched from D to A

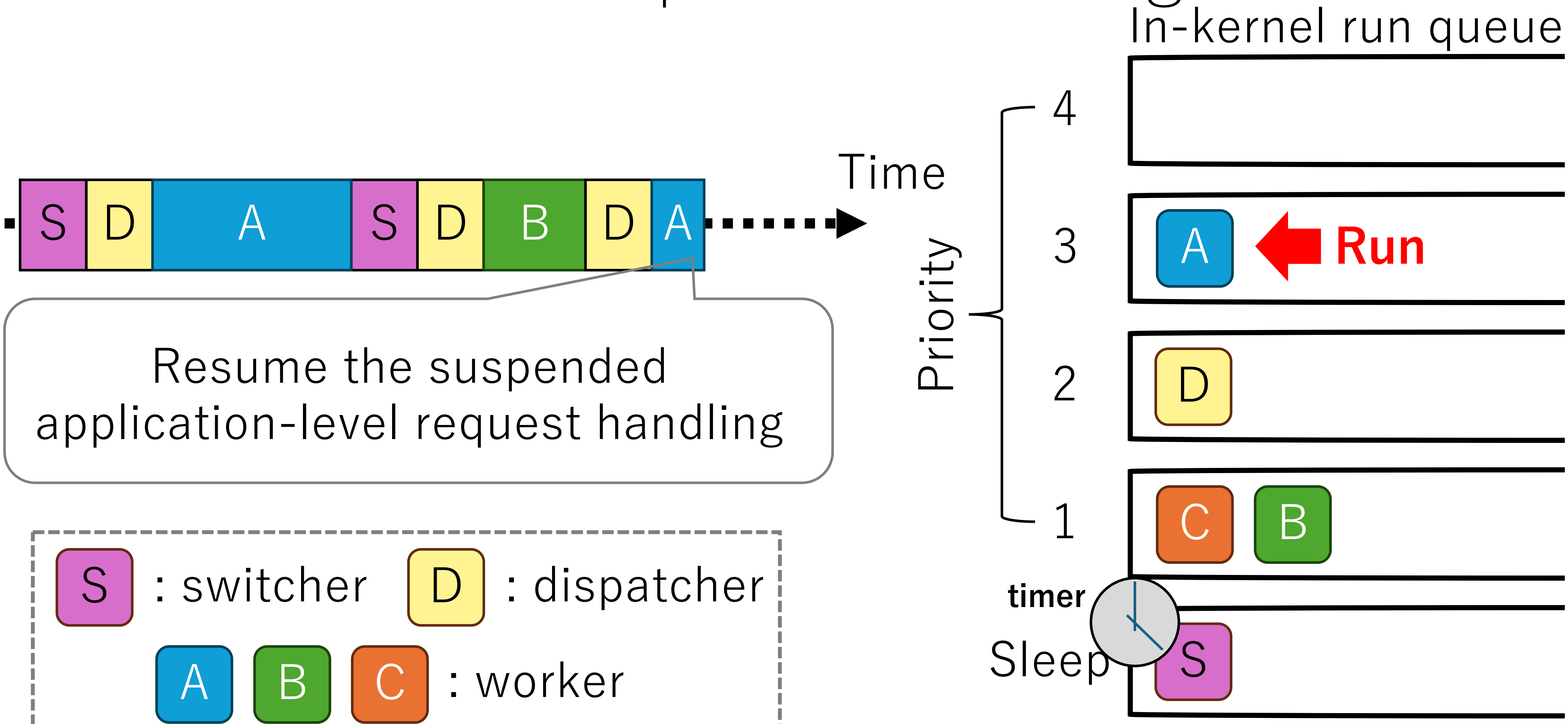




Use Case: Preemptive Scheduling



Use Case: Preemptive Scheduling

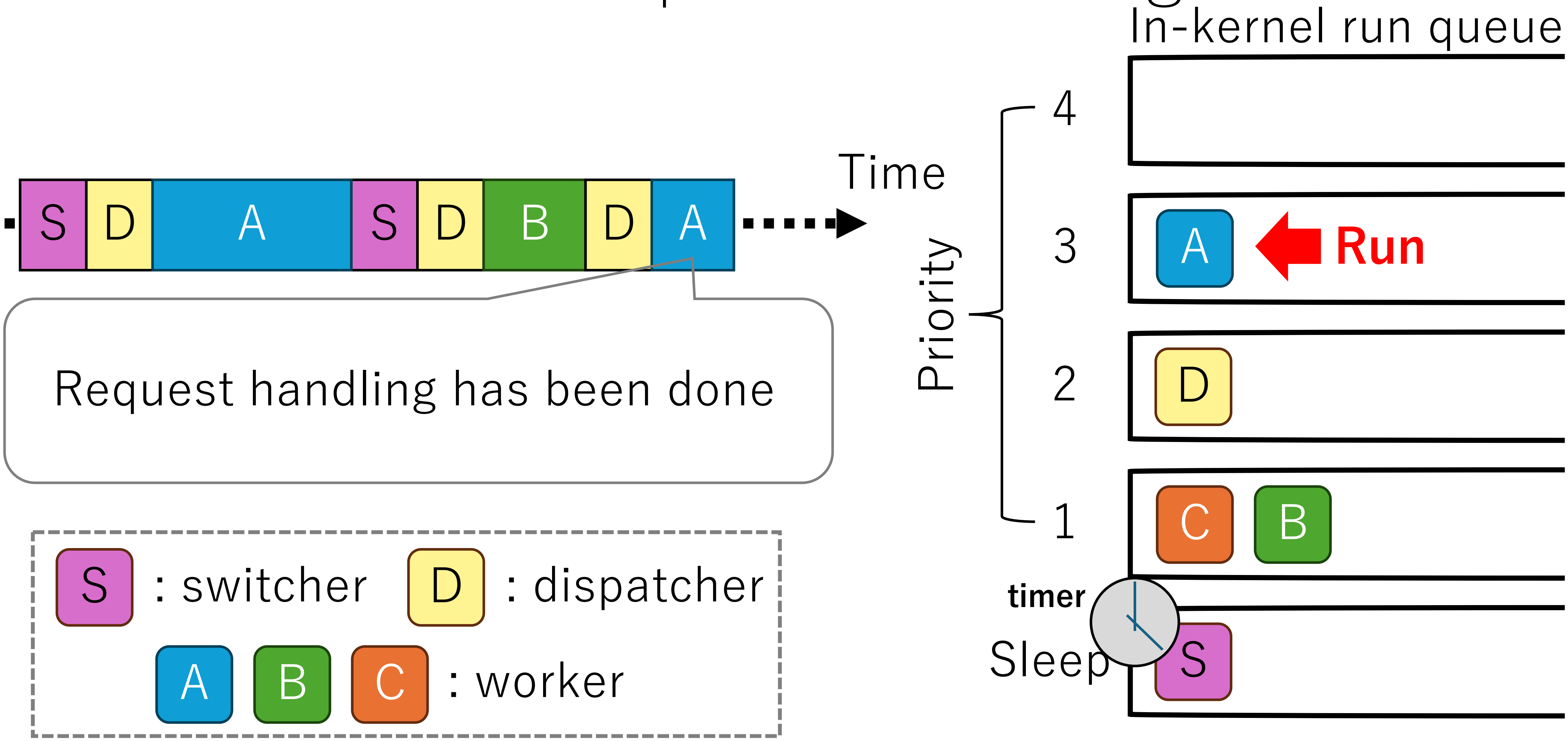


Resume the suspended application-level request handling

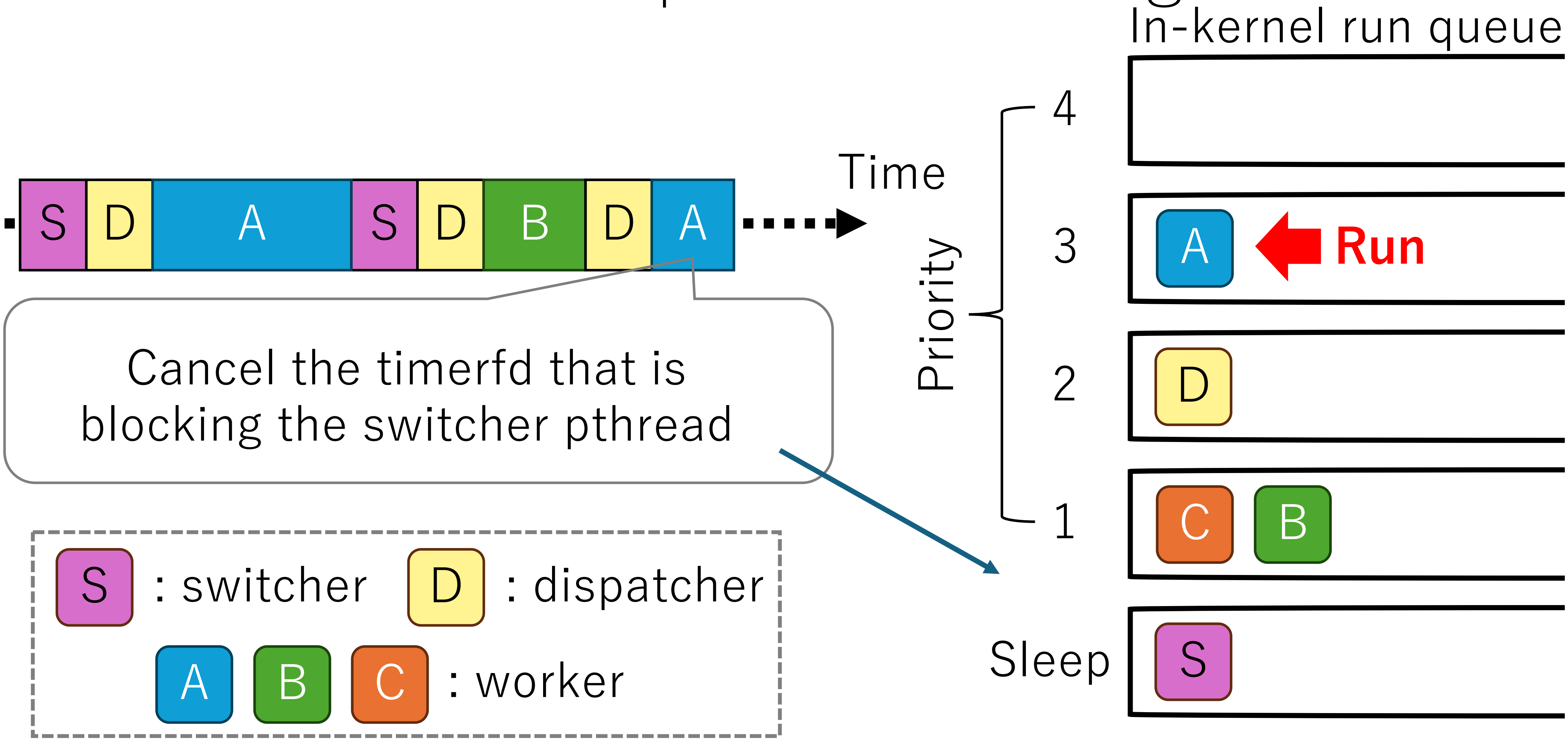
S : switcher D : dispatcher

A B C : worker

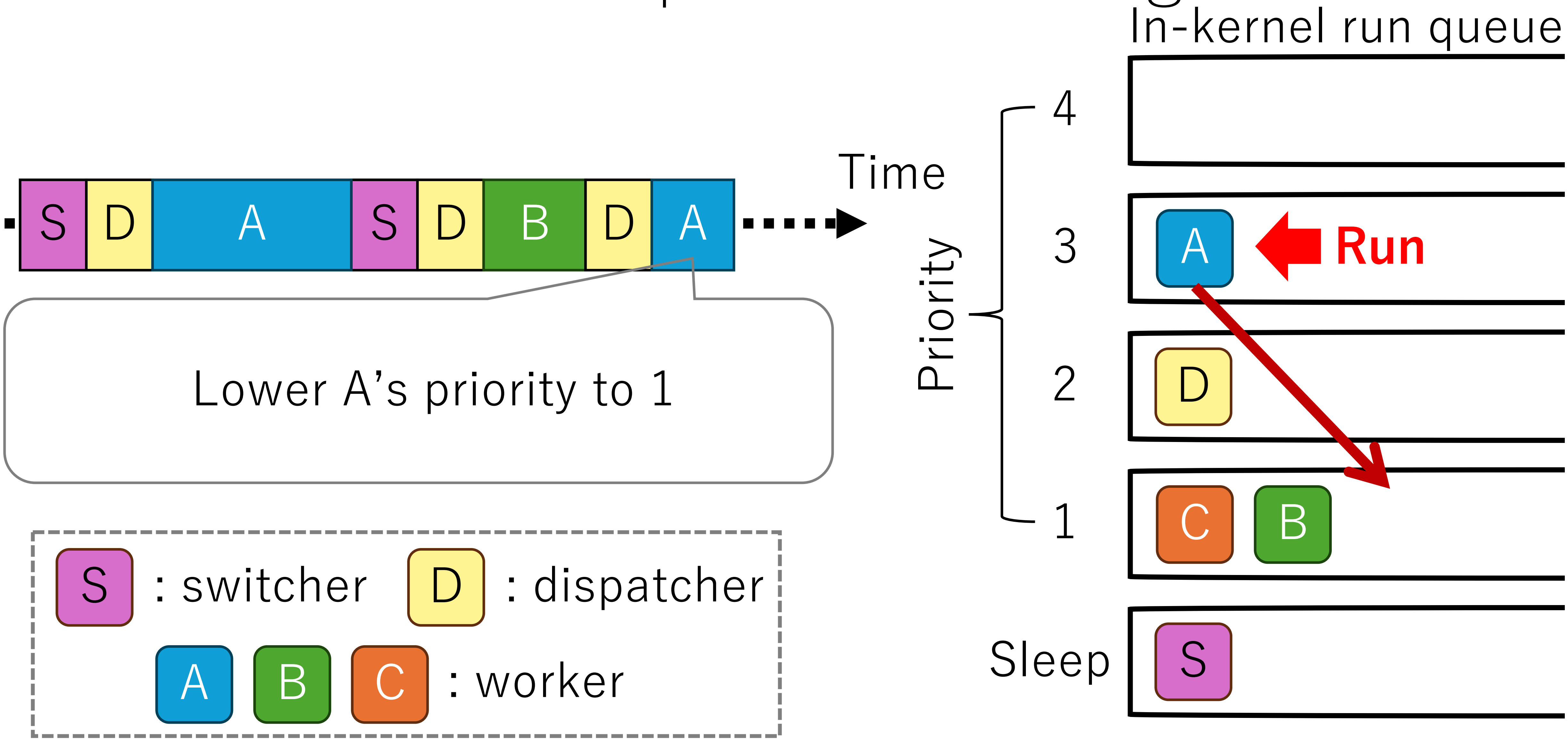
Use Case: Preemptive Scheduling



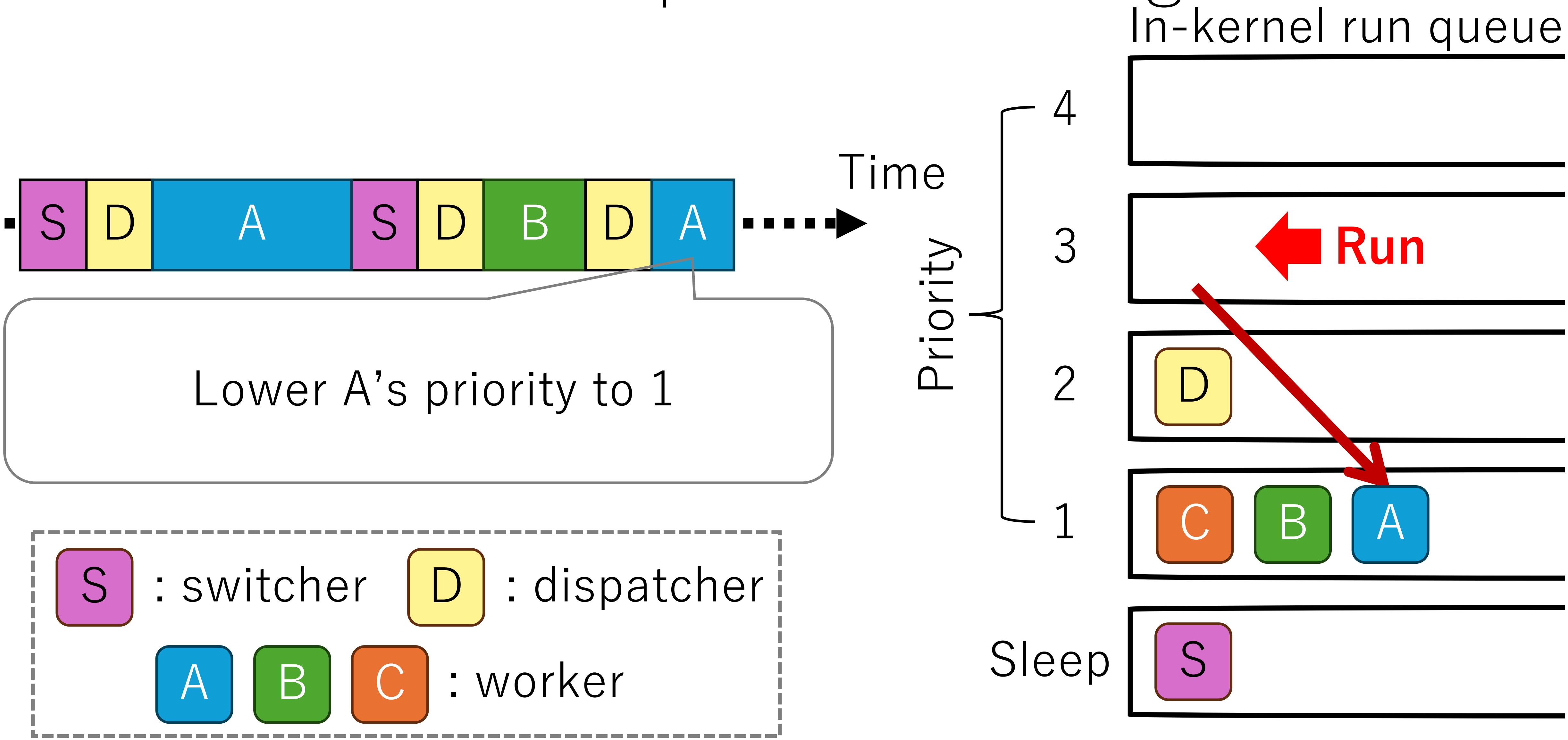
Use Case: Preemptive Scheduling



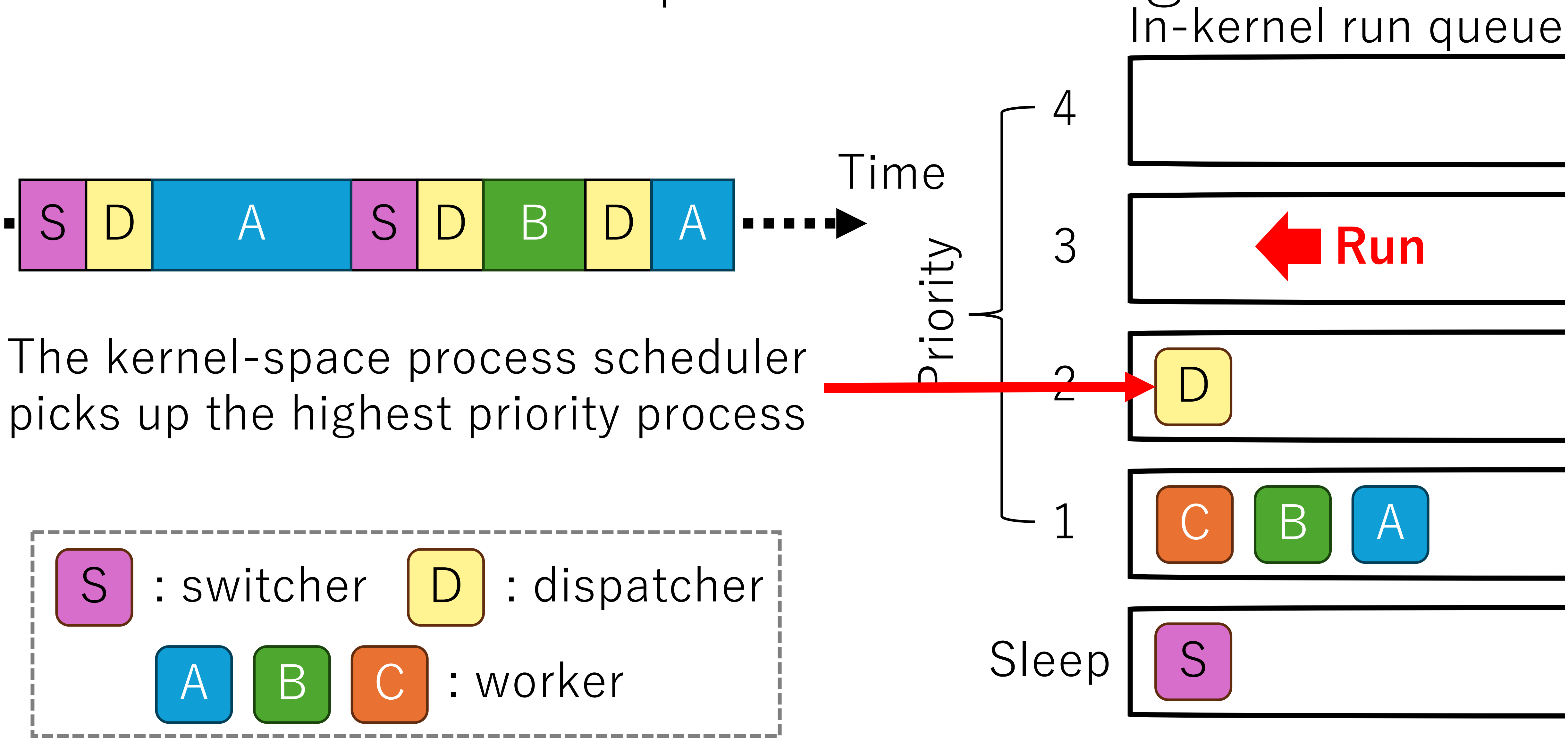
Use Case: Preemptive Scheduling



Use Case: Preemptive Scheduling

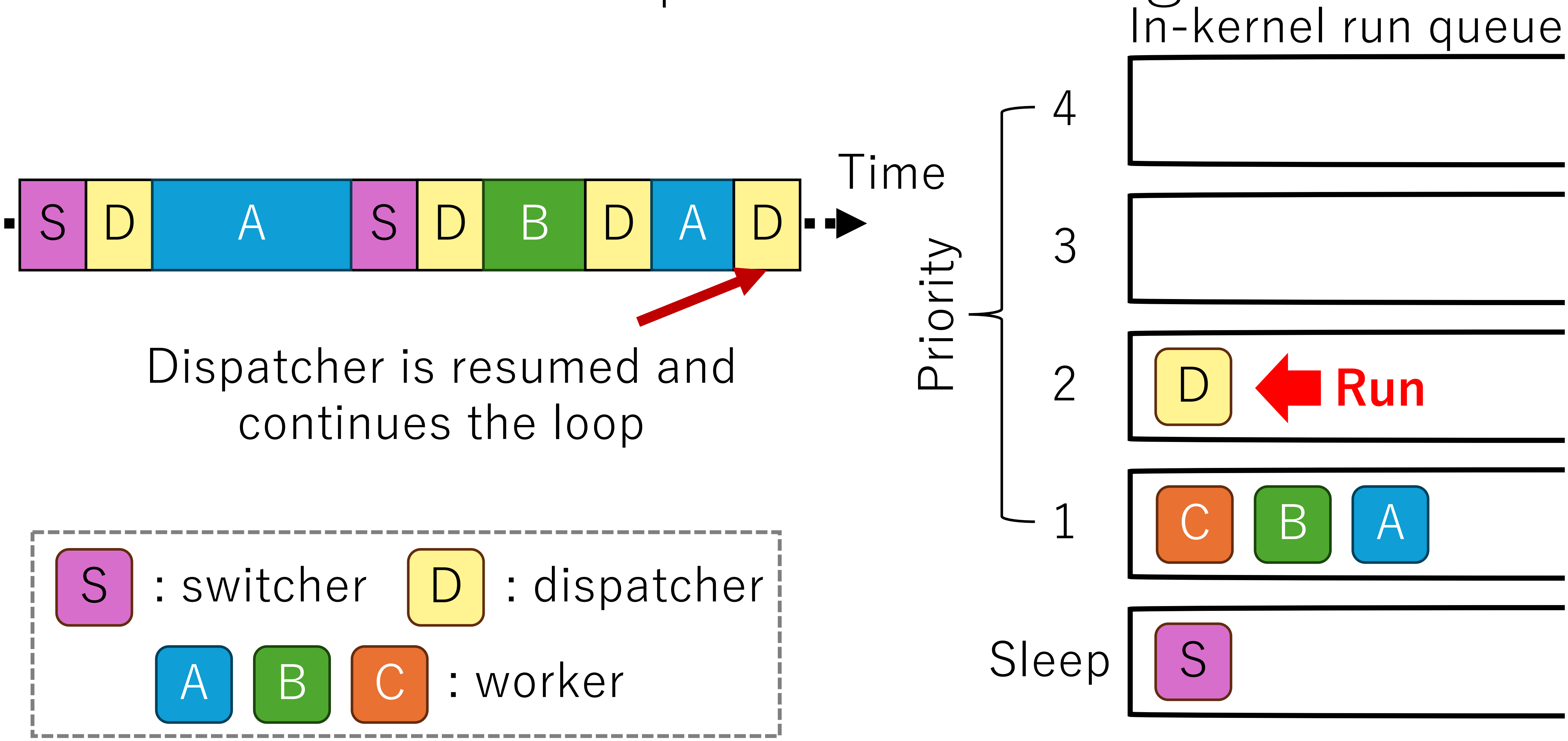


Use Case: Preemptive Scheduling



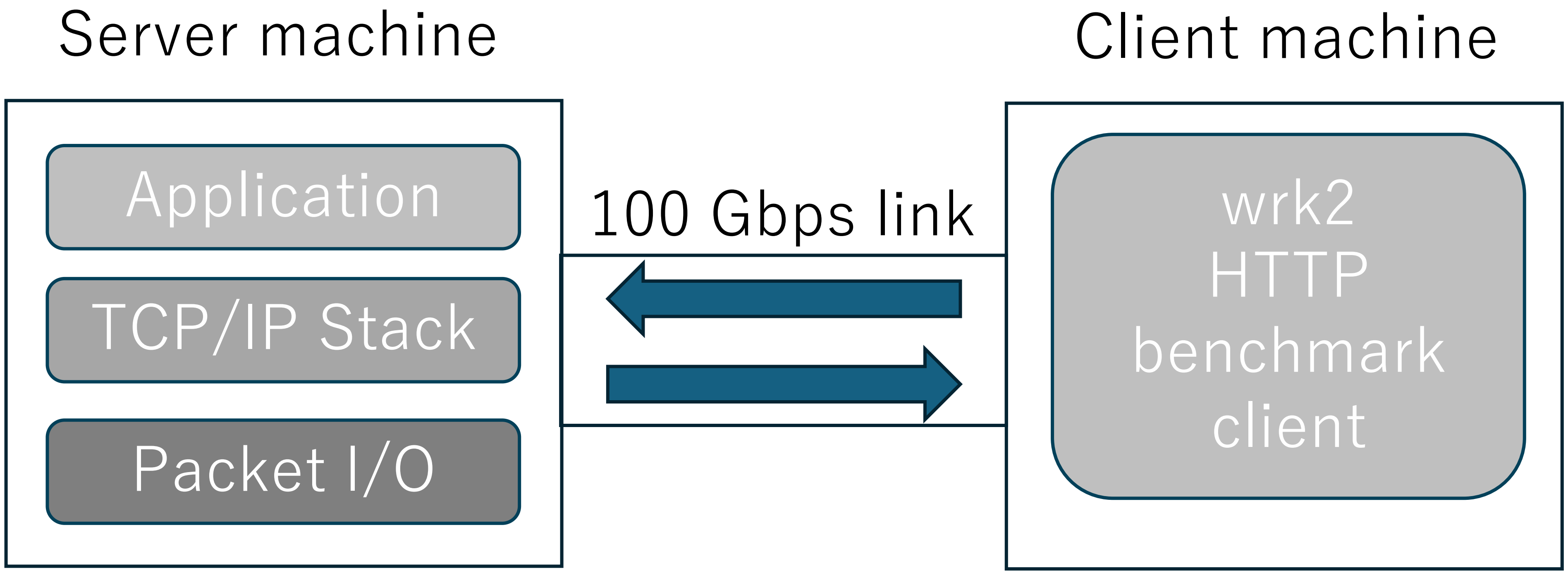
The kernel-space process scheduler picks up the highest priority process

Use Case: Preemptive Scheduling



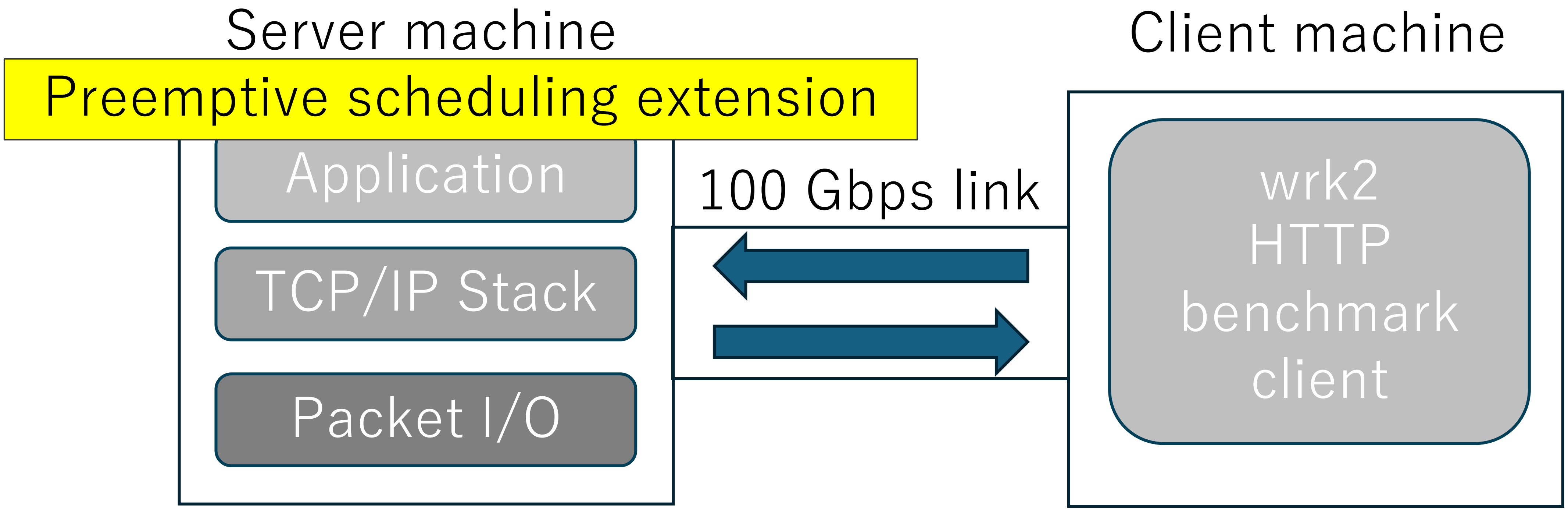
Use Case: Preemptive Scheduling

- The client machine runs wrk2 to send requests

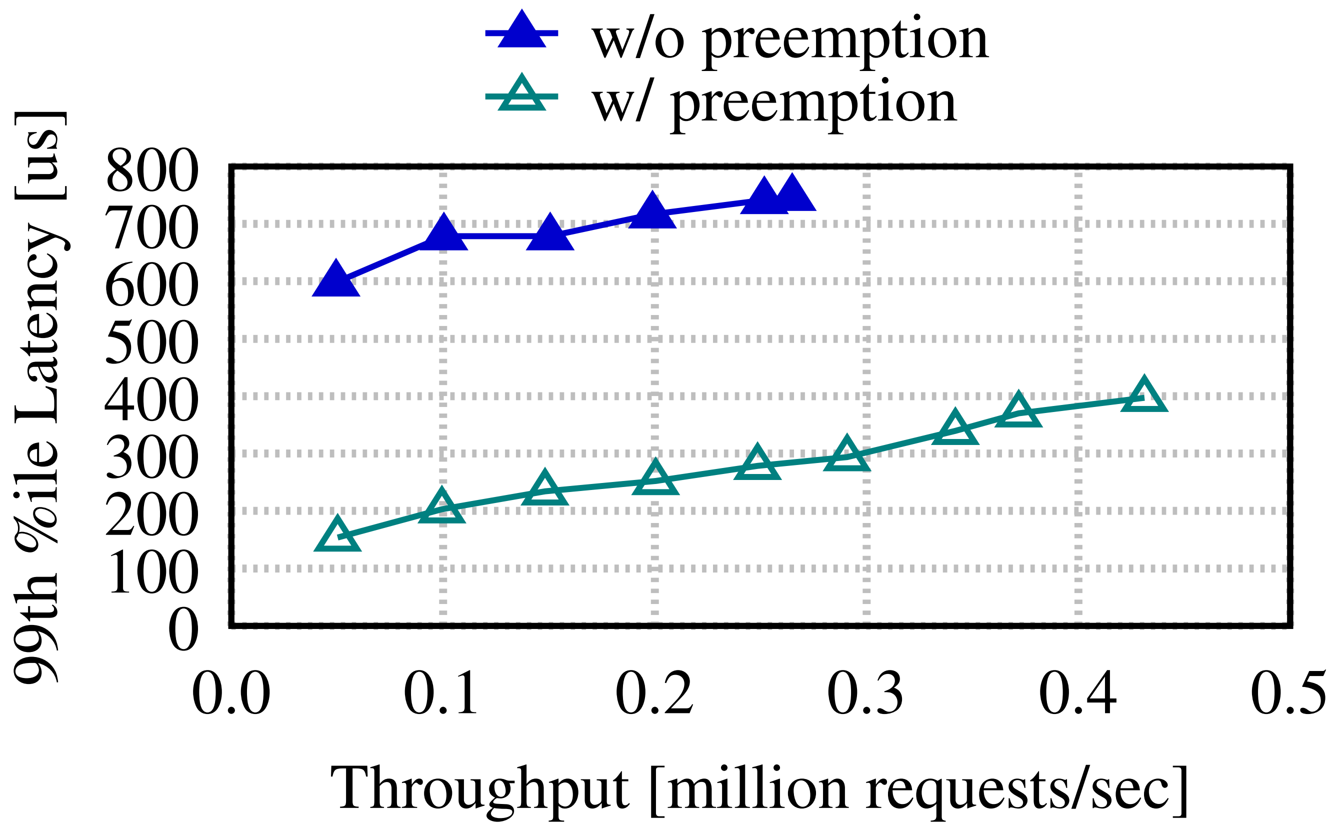


Use Case: Preemptive Scheduling

- The client machine runs wrk2 to send requests

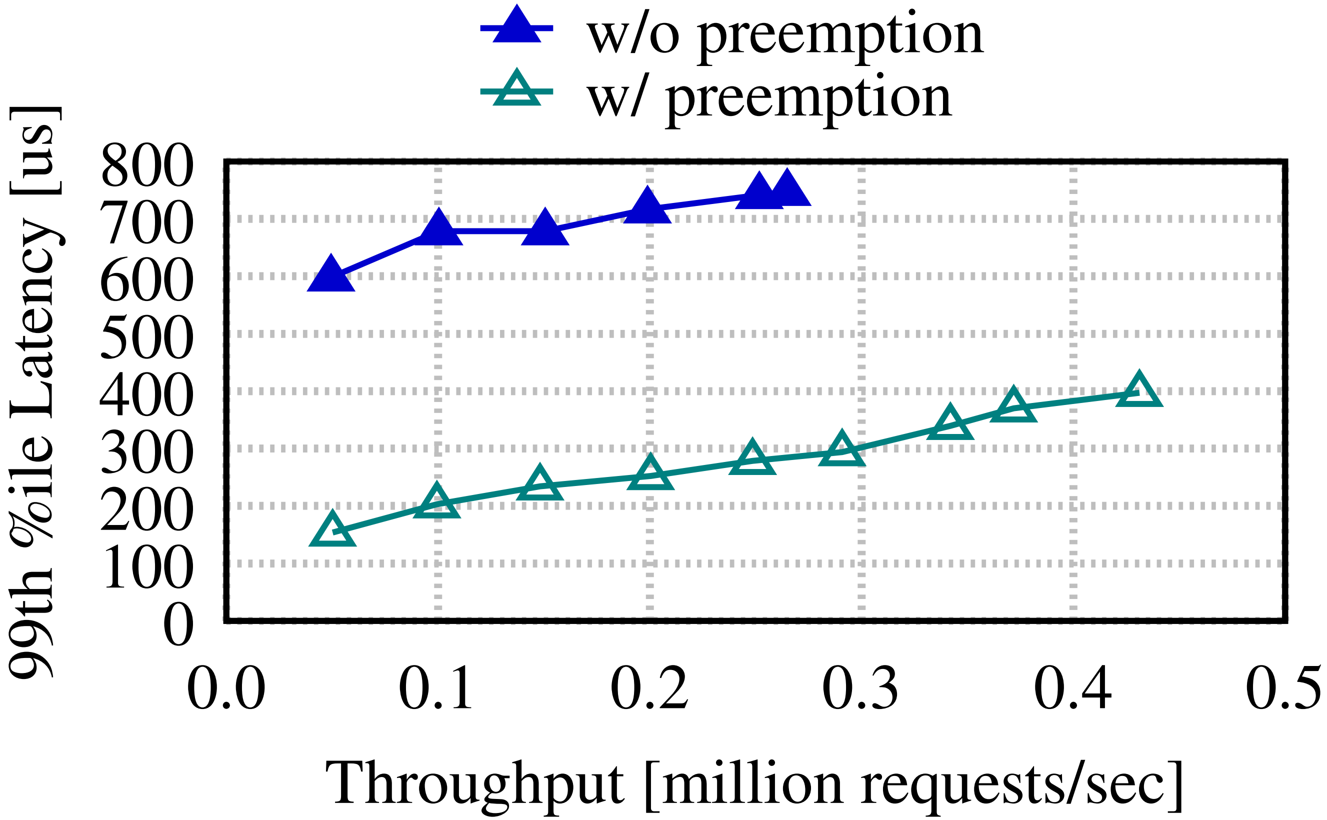


Use Case: Preemptive Scheduling



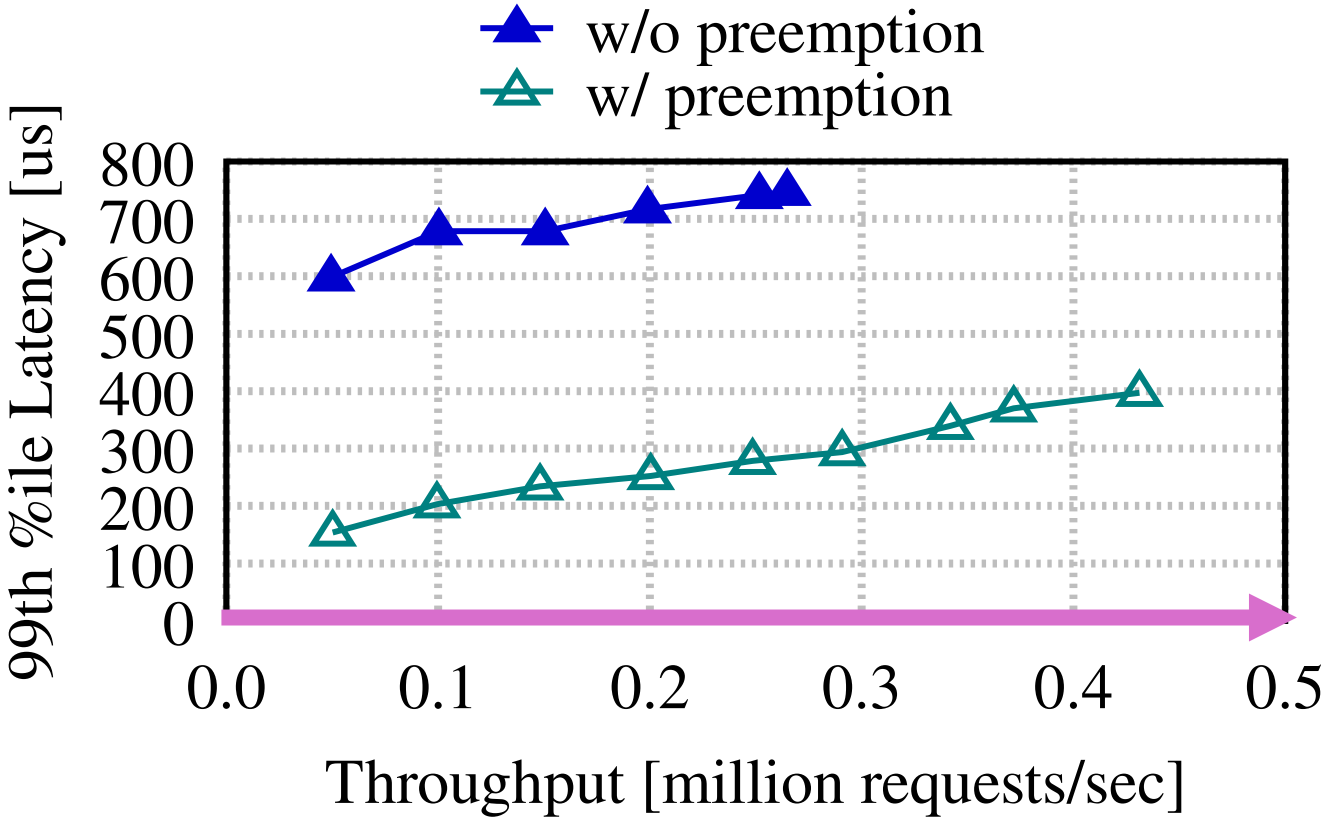
Use Case: Preemptive Scheduling

99.5% requests require 0.5 us
0.5% requests require 500 us



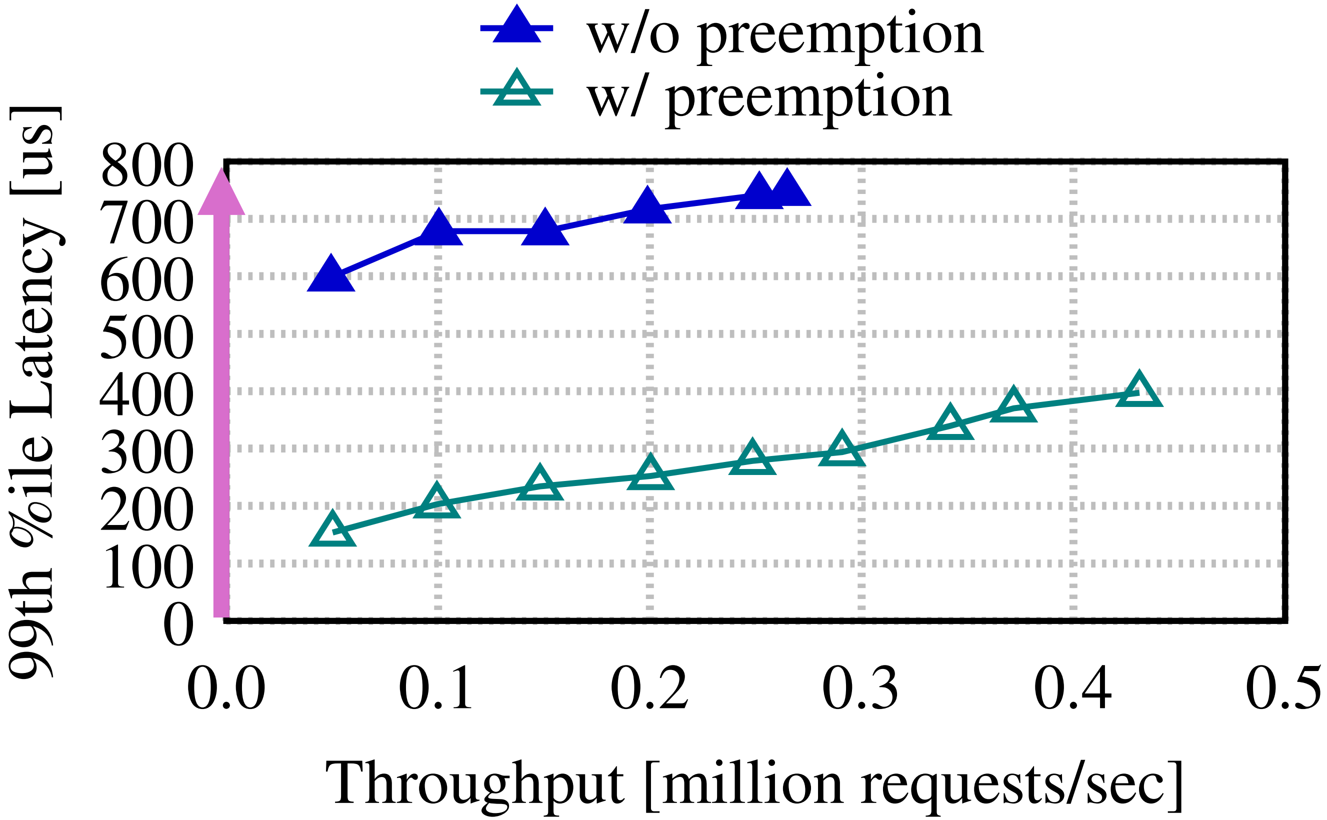
Use Case: Preemptive Scheduling

99.5% requests require 0.5 us
0.5% requests require 500 us



Use Case: Preemptive Scheduling

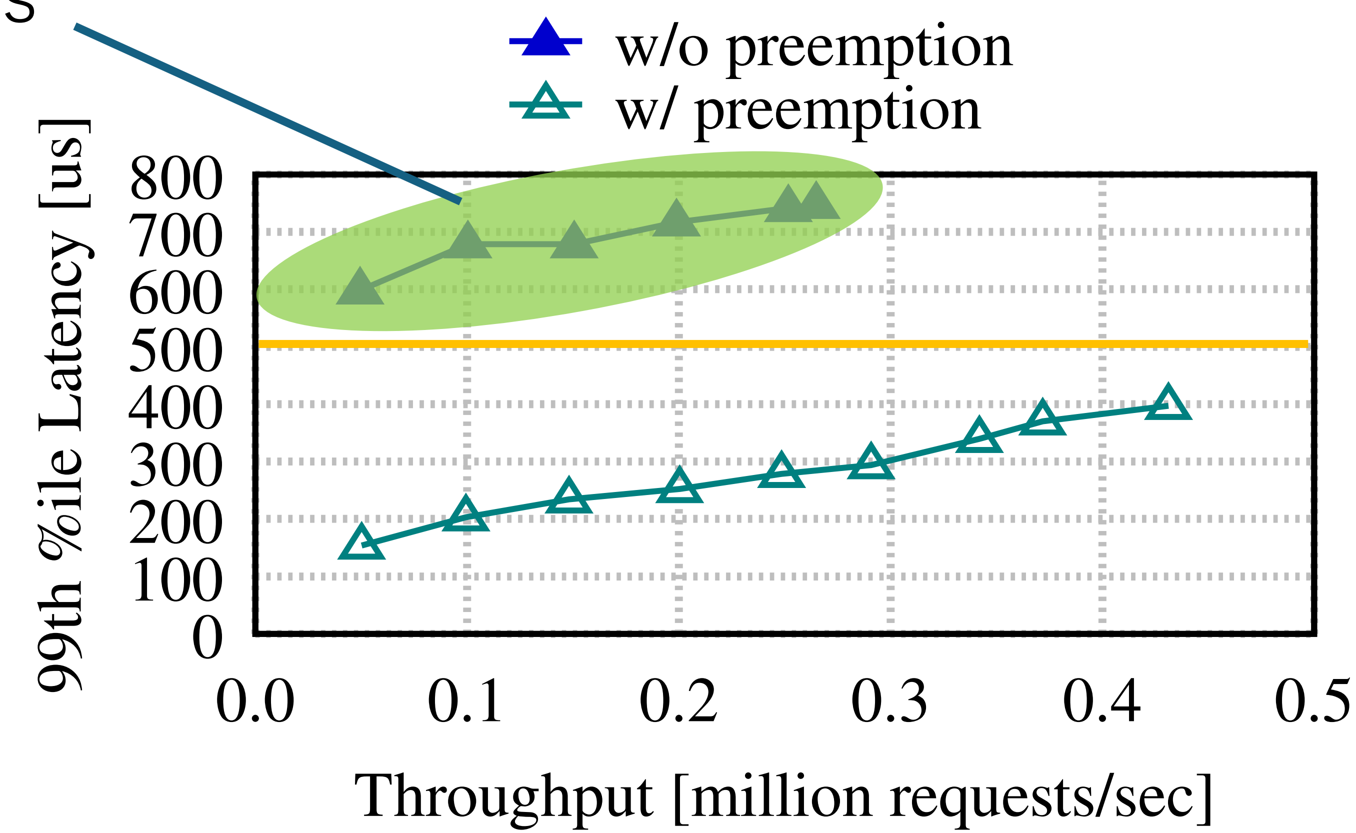
99.5% requests require 0.5 us
0.5% requests require 500 us



Use Case: Preemptive Scheduling

99.5% requests require 0.5 us
0.5% requests require 500 us

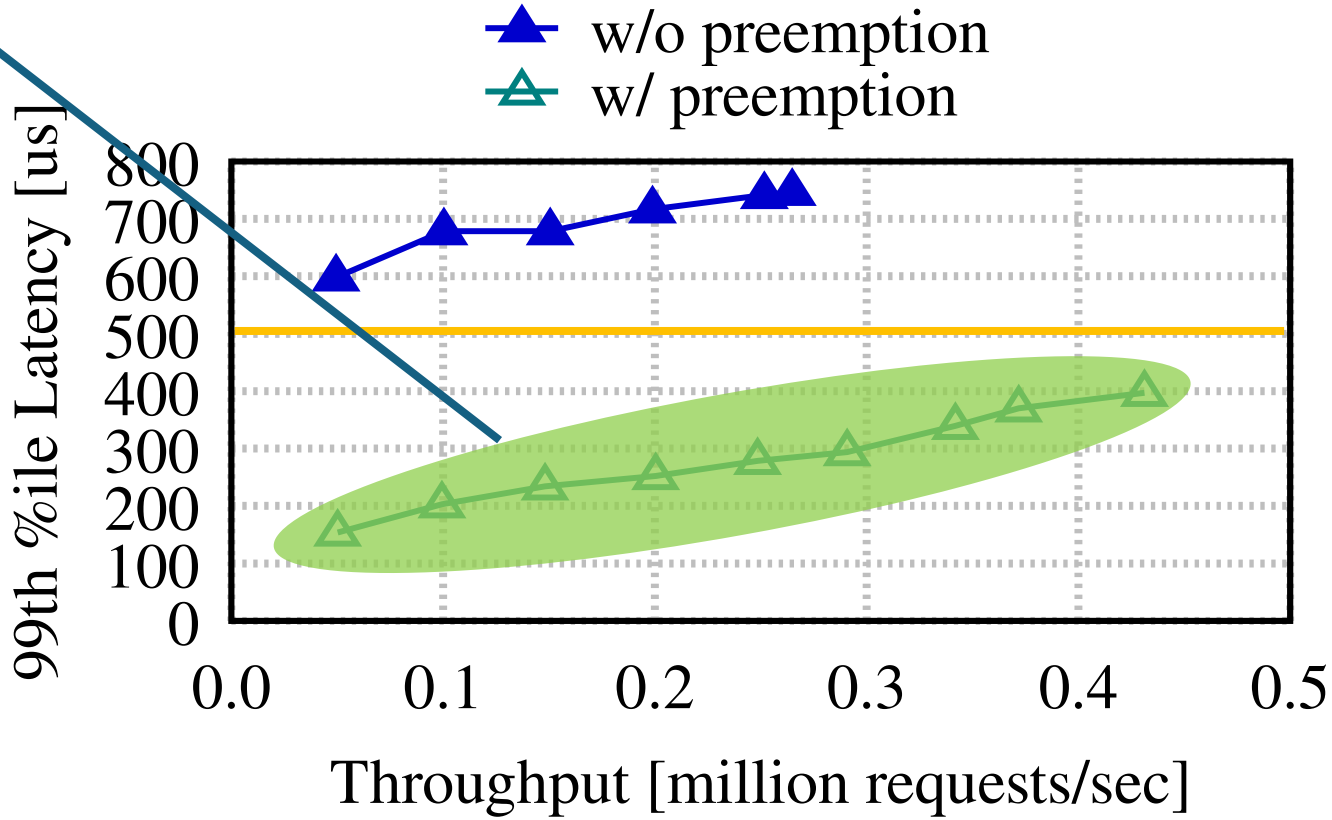
When preemptive scheduling is not activated, the latency goes higher than 500 us



Use Case: Preemptive Scheduling

When preemptive scheduling is activated, the latency goes lower than 500 us

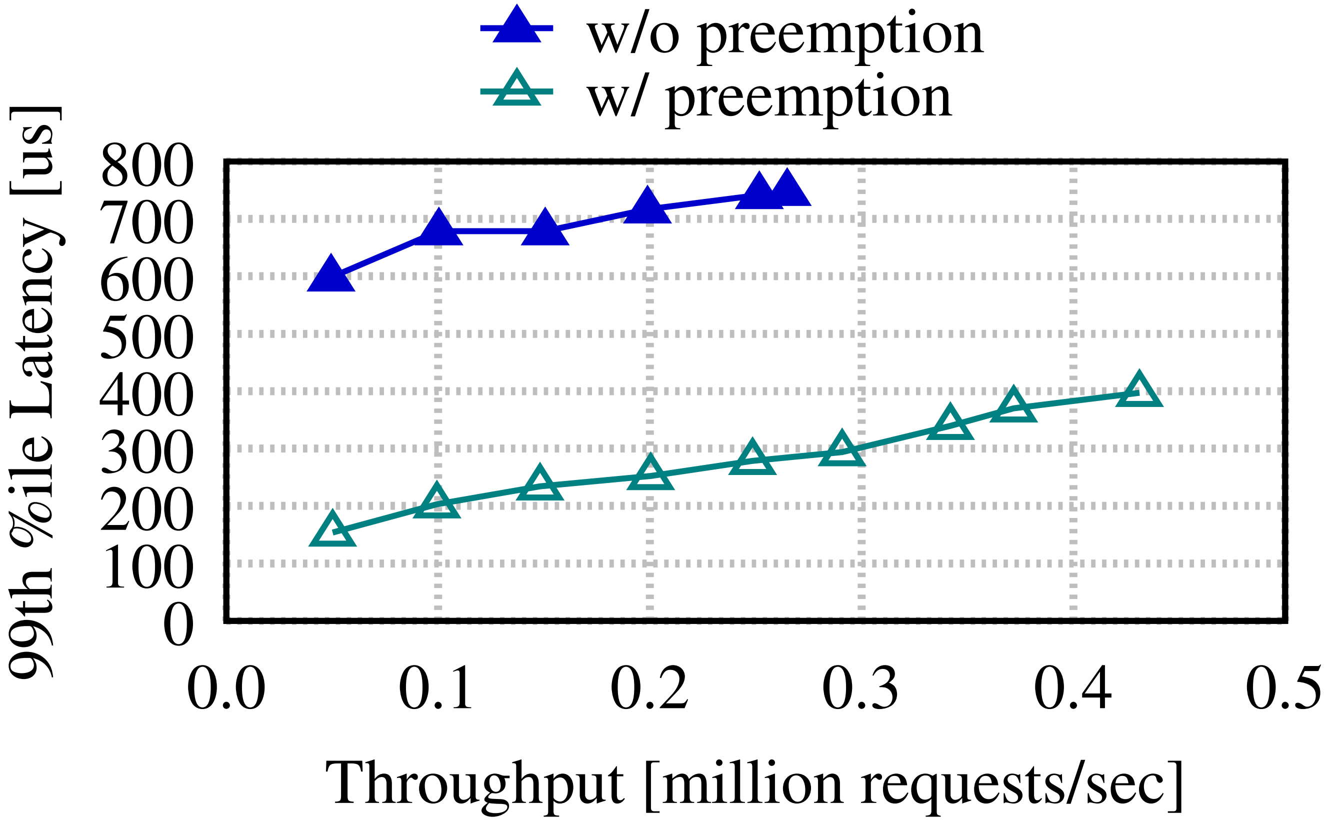
99.5% requests require 0.5 us
0.5% requests require 500 us



Use Case: Preemptive Scheduling

Preemptive scheduling realized by the priority elevation trick successfully mitigated the head-of-line blocking issue

99.5% requests require 0.5 us
0.5% requests require 500 us



Summary

- The priority elevation trick allows us to control scheduling to some extent while only using common OS features
  - We think that this trick is sufficient for many use cases
  - We hope this work contributes to researchers and developers who wish to have a quick and easy utility for scheduler development

Supplemental Materials

https://github.com/yasukata/priority-elevation-trick