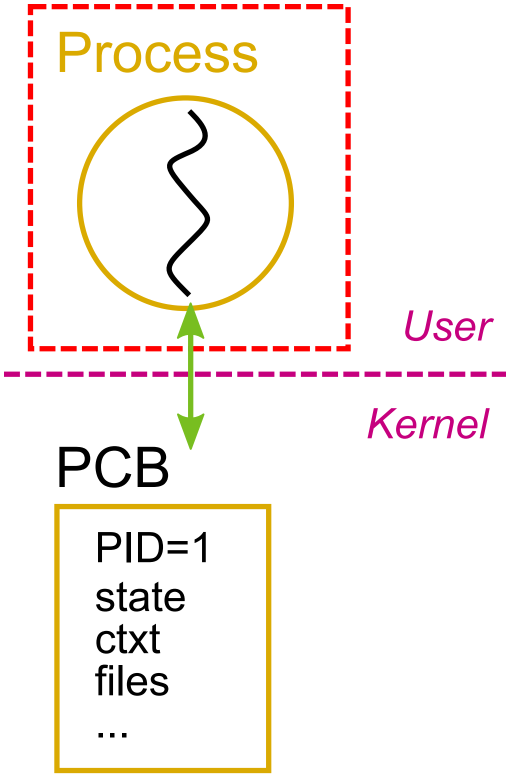
Process scheduling

Process

Definition (recap)

- A process is the abstraction used by the OS to execute programs
- Comprehensive set of features
  - Protection against other processes
  - Isolation from OS/kernel
  - Intuitive and easy-to-use interface (*syscalls*)
  - Portable, hides implementation details
  - Can be instantiated many times
  - Efficient and reasonable easy to implement



Characteristics

1. Address space
2. Environment
3. Execution flow

Process

Address space

- Each process has its own address space

```c
int i = 1;

int main(void)
{
    int j = 10;
    int *k = malloc(sizeof(int));

    *k = 4;

    if (fork()) {
        i = i + 1;
        j = j - 1;
        *k = *k * 1;
    } else {
        i = i + 2;
        j = j - 2;
        *k = *k * 2;
    }

    printf("i=%d, &i=%p\n", i, &i);
    printf("j=%d, &j=%p\n", j, &j);
    printf("k=%d, &k=%p\n", *k, &k);

    return 0;
}
```

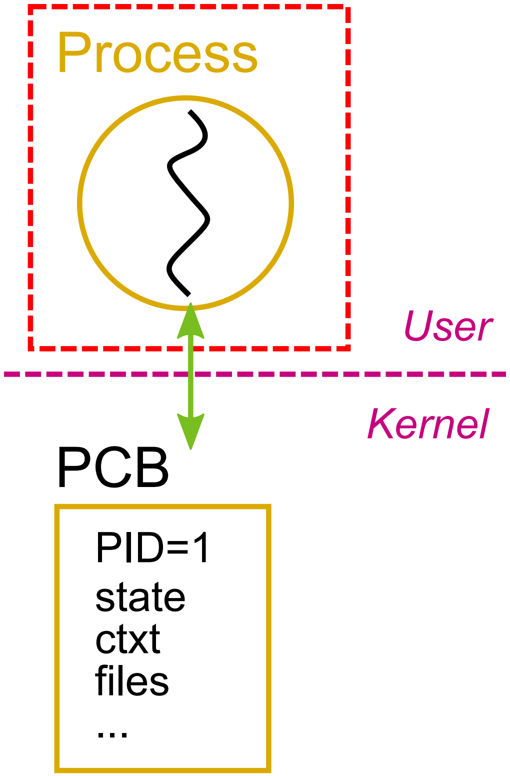
address_space.c

```
$ ./address_space
i=2, &i=0x5634b1f6f048
j=9, &j=0x7ffc70ffaaec
k=4, &k=0x7ffc70ffaaf0
i=3, &i=0x5634b1f6f048
j=8, &j=0x7ffc70ffaaec
k=8, &k=0x7ffc70ffaaf0
```

Process

Environment

- Contained in *PCB*
- Defines all the specific characteristics of a process
  - Process ID, Process group ID
  - User ID, Group ID
  - Link to parent process
  - List of memory segments
    - text, data, stack, heap
  - Open file tables
  - Working directory
  - Process state
  - Scheduling parameters
  - Space for saved context
    - PC, SP, general-purpose registers
  - Etc.

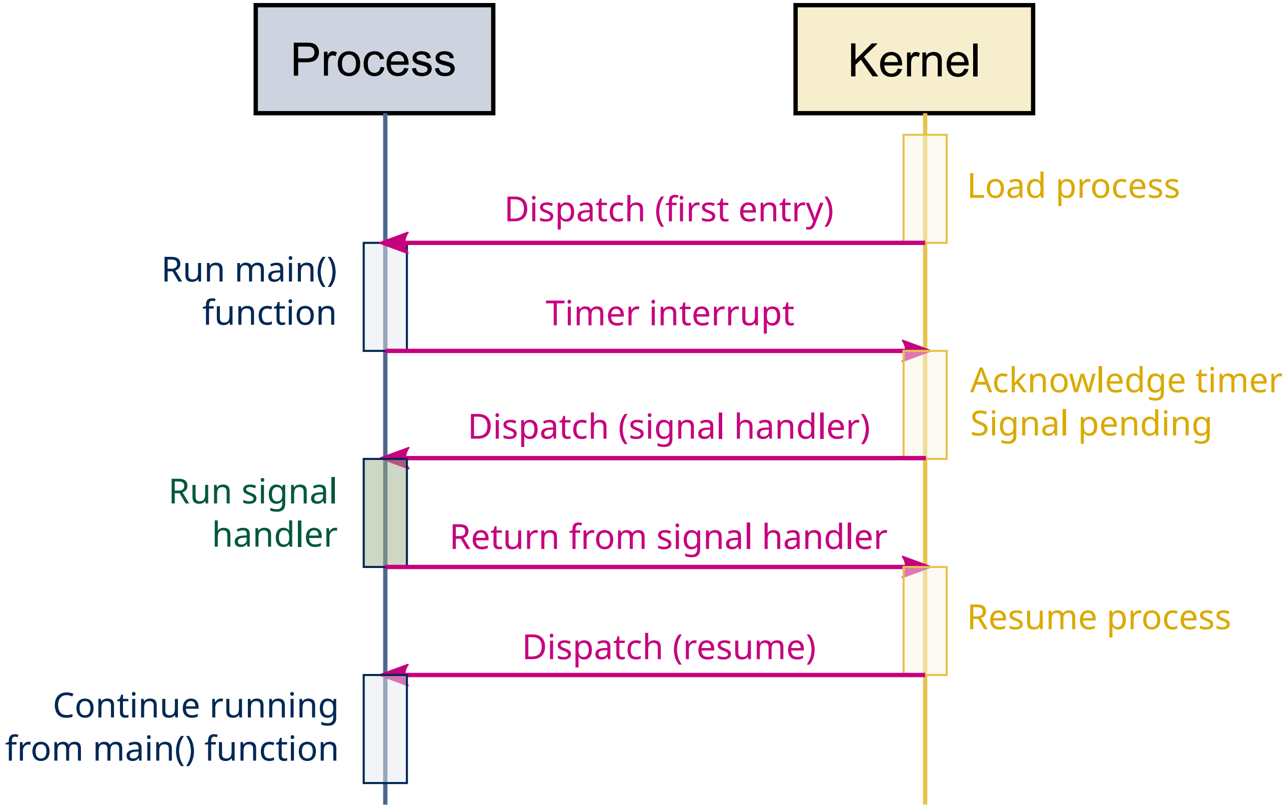


Process

Execution flow

- Single sequential execution stream
    - Statements executed in order
    - Can only be at one location in the code at a time
- Can only be (slightly) disrupted by signals

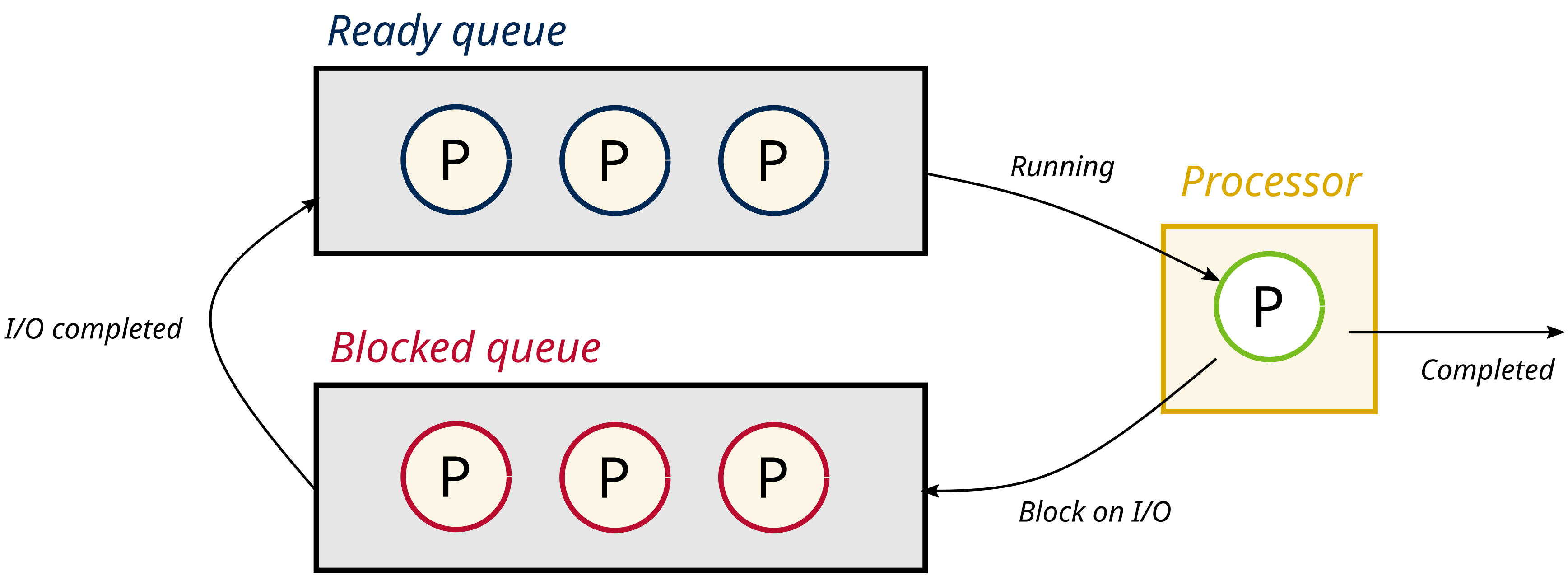
```
void sig_hndl(int signo)
{
    statement#1;
    statement#2;
    statement#3;
    ...
    statement#N;
}

int main(void)
{
    statement#1;
    statement#2;
    statement#3;
    statement#4;
    statement#5;
    ...
    statement#N;
}
```



Scheduling concepts

Definition

- Single-processor systems only allow one process to run at a time
- Scheduler in charge of determining which process should run
    - Ready queue contains all processes ready to run



Scheduling concepts

CPU-I/O burst cycles

```c
int main(void) {
    int fd;
    int i;
    char buf[256];

    fd = open("input.txt", O_RDONLY);       /* I/O burst */
    read(fd, buf, sizeof(buf));
    close(fd);

    for (i = 0; i < sizeof(buf); i++) {     /* CPU burst */
        if (isupper(buf[i]))
            buf[i] = tolower(buf[i]);
    }

    fd = open("output.txt", O_RDONLY);      /* I/O burst */
    write(fd, buf, sizeof(buf));
    close(fd);

    return 0;
}
```

CPU-bound vs I/O-bound

- CPU-bound processes (e.g., scientific calculations)
- I/O-bound processes (e.g., BitTorrent)
- Mix CPU-I/O-bound processes (e.g., compiler)

Scheduling concepts

Multitasking

- Goal of maximizing CPU utilization among multiple processes
- When process is performing I/O burst, give CPU to *next* process
- Scheduling *policy* determines which process is next

Cooperative

- Process can hold unto CPU during long CPU bursts
- Only yields voluntarily, or during I/O bursts

```
int main(void)
{
    for () {        /* CPU burst */
        ...
    }

    sched_yield();  /* Yield CPU */

    scanf();        /* I/O burst */
    ...
}
```

Preemptive

- Process can be forcefully suspended, even during long CPU bursts
- Use of hardware timer interrupts
- Ensures guarantee in CPU sharing between multiple processes

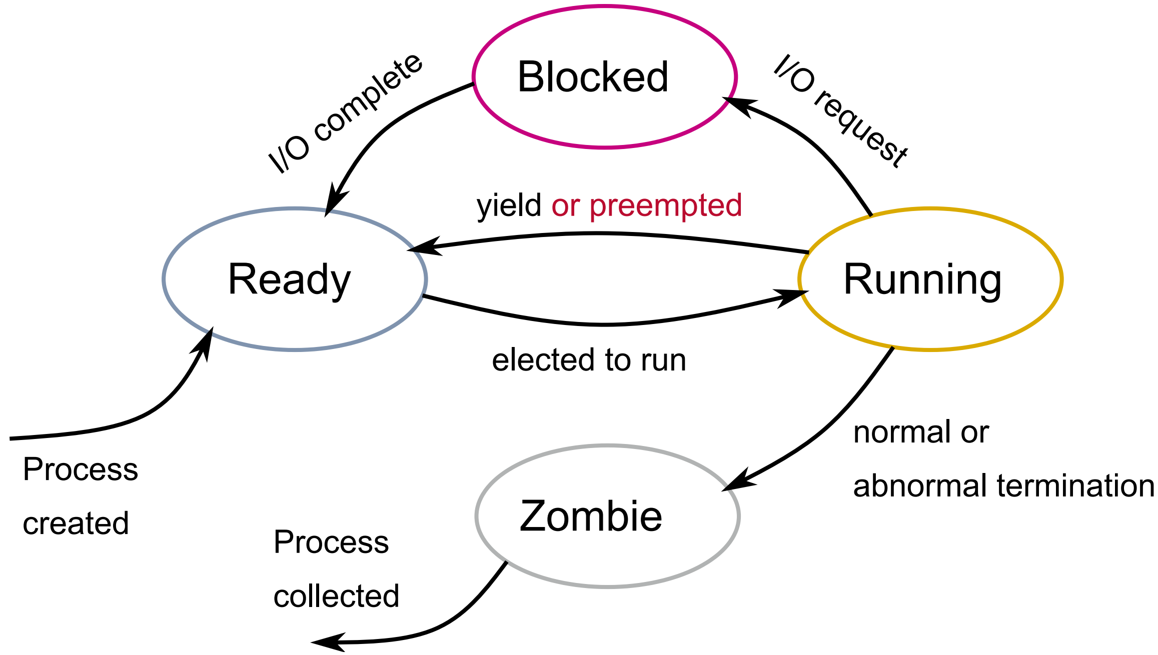
```
int main(void)
{
    for () {        /* CPU burst */
        ...
    }

    scanf();        /* I/O burst */
    ...
}
```

Scheduling concepts

Process lifecycle

Process states

- Ready
- Running
- Blocked
- Zombie



Orphaned processes

- Special scenario if parent's process terminates before process
- Depends on whether process is running from terminal or not
  - Delivery of `SIGHUP` signal or *reparenting*

ECS 150 - Process scheduling

Prof. Joël Porquet-Lupine

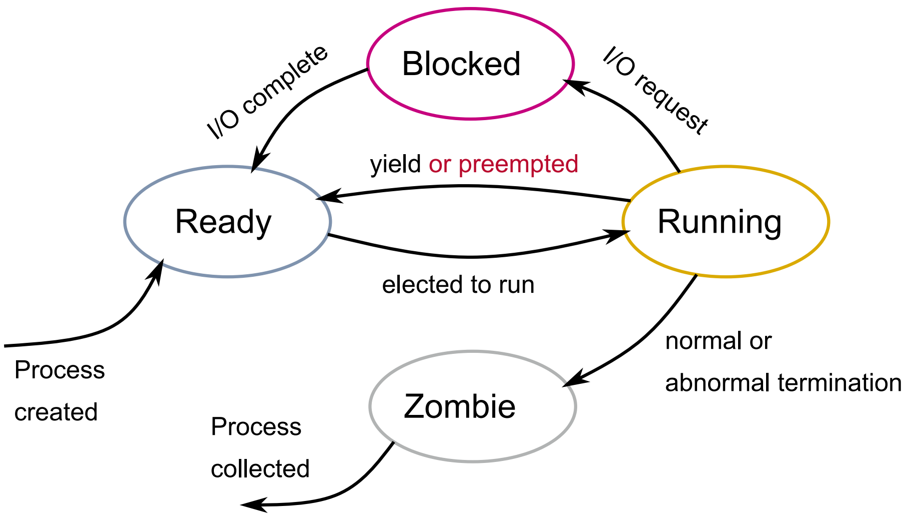
UC Davis - 2020/2021

Recap

Process

- Address space
  - Each process has it own address space
- Environment
  - Mostly represented by OS' PCB
- Execution flow
  - Single sequential execution stream

Process lifecycle



Scheduling concepts

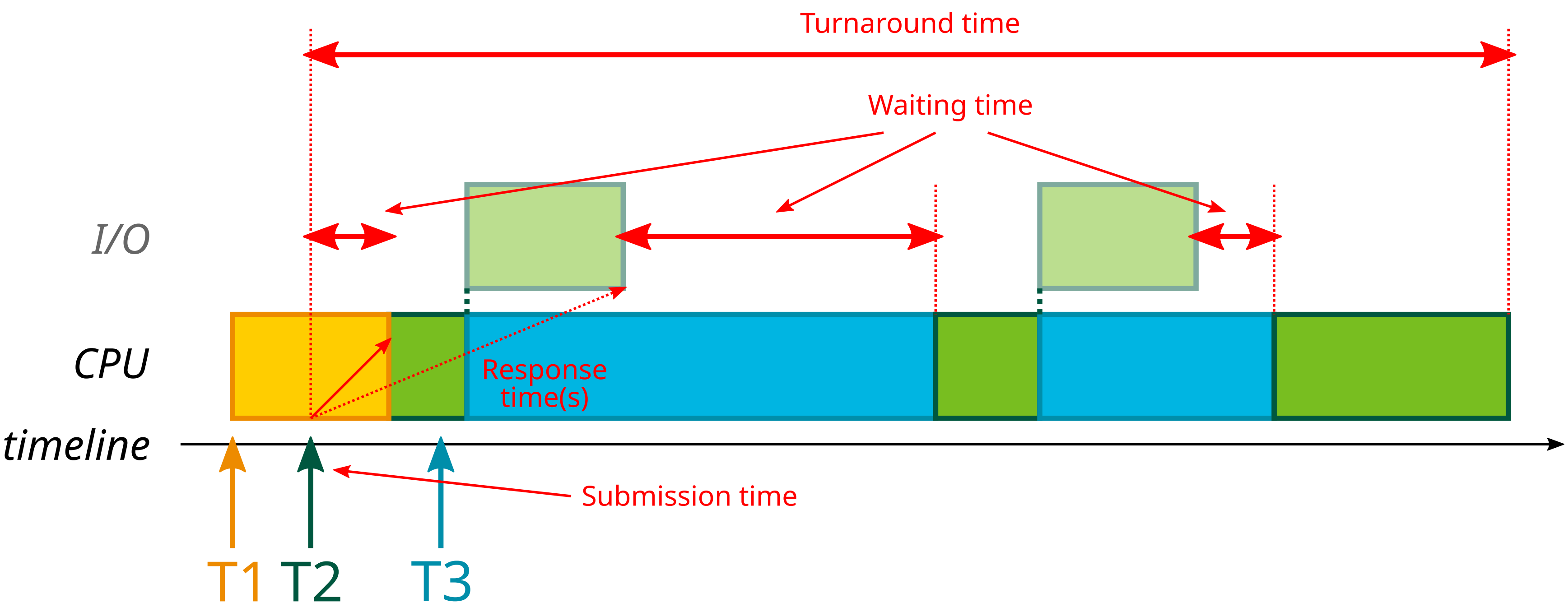
- Share processor resource among *ready* processes
- CPU bursts vs IO bursts
  - CPU-bound vs IO-bound processes

```
/* I/O burst */
fd = open("input.txt", O_RDONLY);
read(fd, buf, sizeof(buf));
close(fd);

/* CPU burst */
for (i = 0; i < sizeof(buf); i++) {
    if (isupper(buf[i]))
        buf[i] = tolower(buf[i]);
}
```

- Cooperative vs preemptive

Scheduling algorithms

Vocabulary



- **Submission time**: time at which a process is created
- **Turnaround time**: total time between process submission and completion
- **Response time**: time between process submission and first execution or first response (e.g., screen output, or input from user)
- **Waiting time**: total time spent in the ready queue

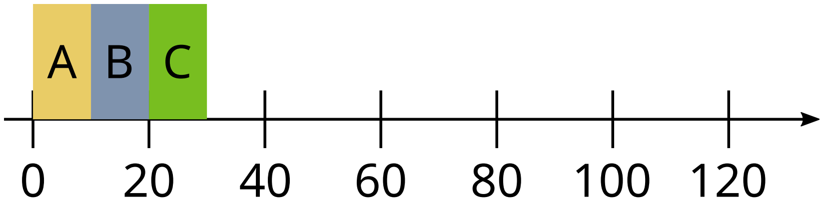
Scheduling algorithms

FCFS (or FIFO)

- First-Come, First-Served
- Most simple scheduling algorithm (e.g., queue at DMV)

Example 1

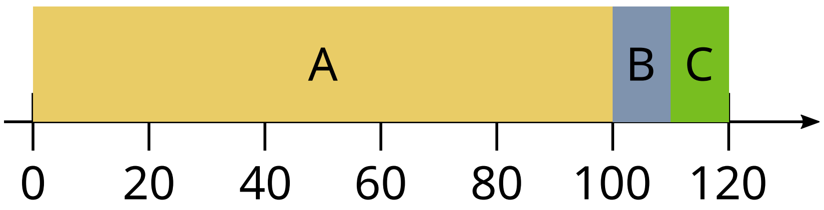
Task	Submission	Length
A	0	10
B	0	10
C	0	10



- Avg turnaround time: $\frac{10+20+30}{3} = 20$

Example 2

Task	Submission	Length
A	0	100
B	0	10
C	0	10



- Avg turnaround time: $\frac{100+100+110}{3} = 103.33$
- Problem known as *convoy effect*

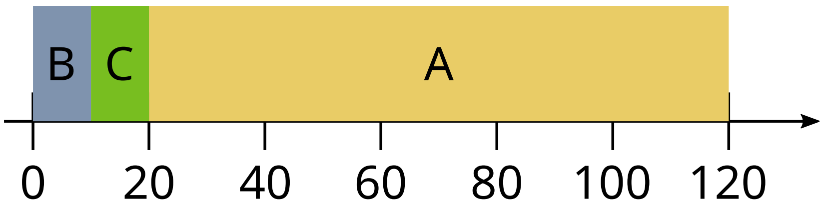
Scheduling algorithms

SJF

- Shortest Job First
- *Optimal* scheduling but requires to know task lengths in advance
  - Use predictions instead (based on past behavior)

Example 1

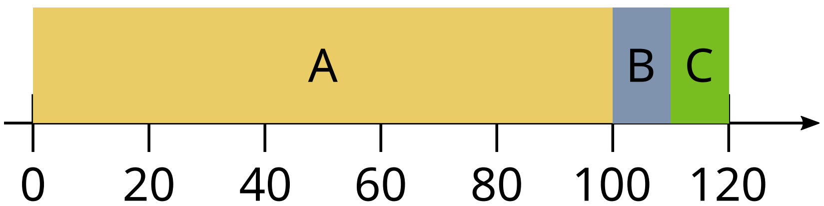
Task	Submission	Length
A	0	100
B	0	10
C	0	10



- Avg turnaround time: $\frac{10+20+120}{3} = 50$

Example 2

Task	Submission	Length
A	0	100
B	10	10
C	10	10



- Avg turnaround time: $\frac{100+100+110}{3} = 103.33$

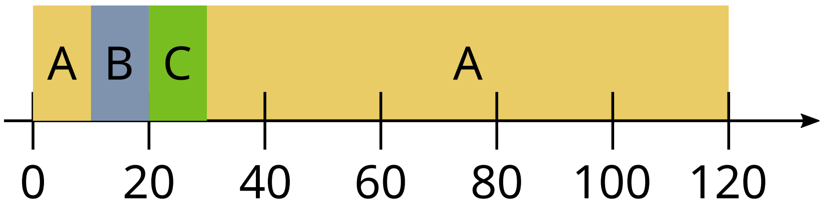
Scheduling algorithms

Preemptive SJF

- Also known as SRTF (Shortest Remaining Time First)
- New shorter jobs can interrupt longer jobs

Example

Task	Submission	Length
A	0	100
B	10	10
C	10	10



- Avg turnaround time: $\frac{120+10+20}{3} = 50$
- Can lead to *starvation*

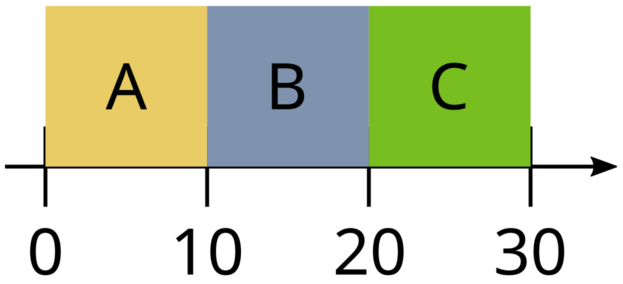
Scheduling algorithms

Turnaround time vs response time

- Optimizing for turnaround time great for (old) batch systems
  - Length of tasks known (or predicted) in advance
  - Tasks mostly CPU-bound
- With interactive systems, need to optimize for response time
  - User wants reactivity
  - Tasks of unknown length

SJF (again)

Task	Submission	Length
A	0	10
B	0	10
C	0	10



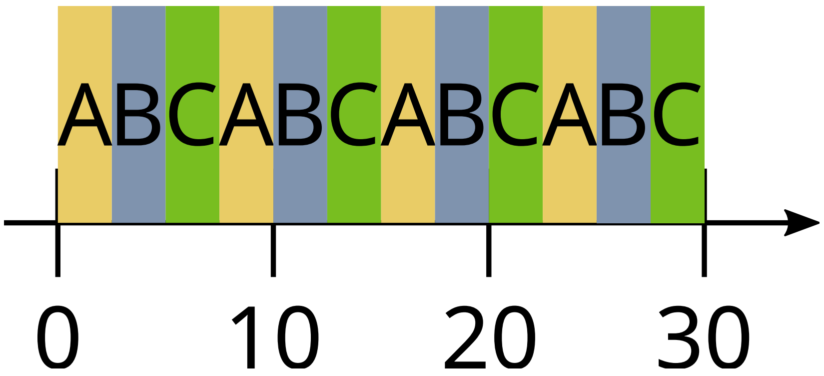
- Avg turnaround time: $\frac{10+20+30}{3} = 20$
- Avg response time: $\frac{0+10+20}{3} = 10$

Scheduling algorithms

Round-robin (RR)

- Tasks run only for a (short) *time slice* at a time
- Relies on preemption (via timer interrupts)

Task	Submission	Length
A	0	10
B	0	10
C	0	10



- Avg response time: $\frac{0+2.5+5}{3} = 2.5$
- Avg turnaround time: $\frac{25+27.5+30}{3} = 27.5$

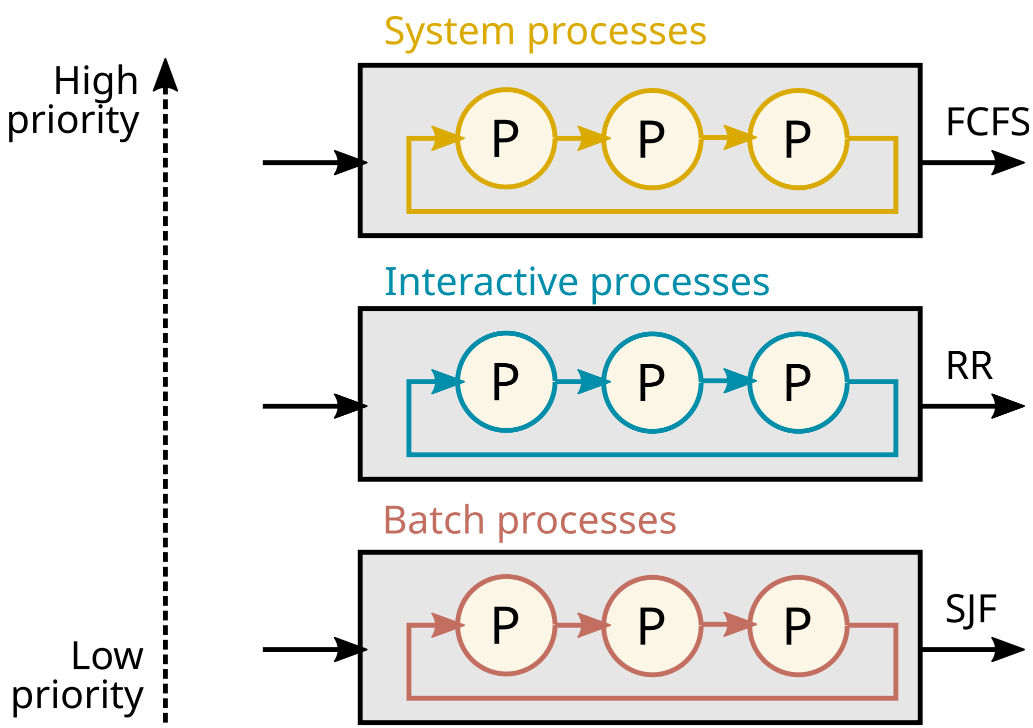
Characteristics

- Prevents starvation
- Time slice duration matters
  - Response time vs context switching overhead
- Poor turnaround time

Scheduling algorithms

Multi-level queue scheduling

- Classify tasks into categories
  - E.g., *foreground* (interactive) tasks vs *background* (batch) tasks
- Give different priority to each category
  - E.g., Interactive > batch
- Schedule each categorize differently
  - E.g., optimize for response time or turnaround time



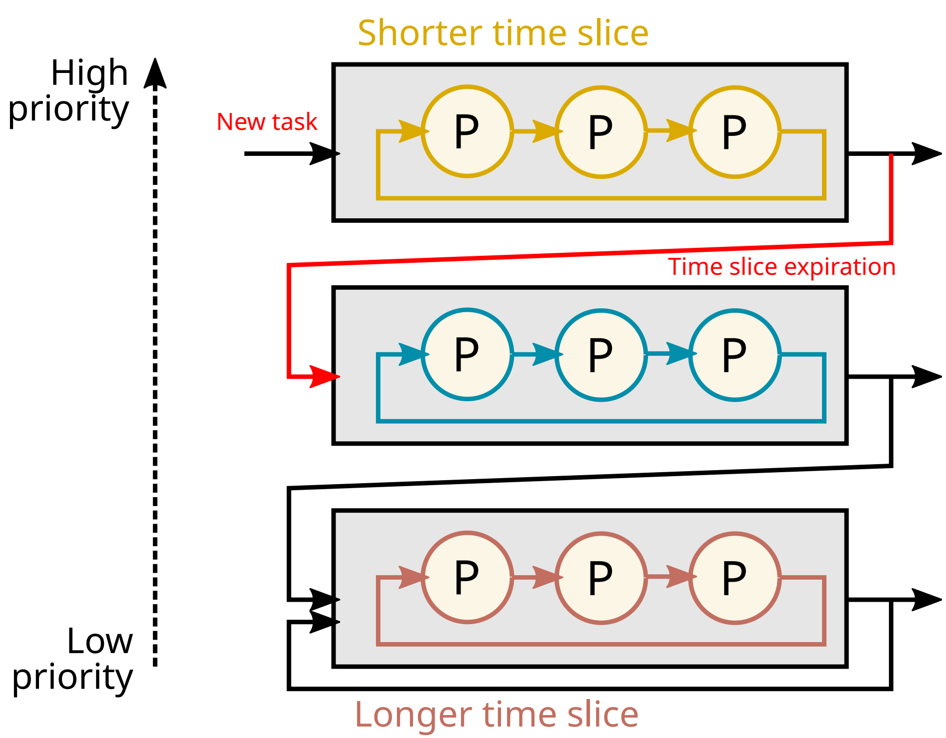
Characteristics

- Adapt to each task's type
- More flexible than strict FCFS, SJF, or RR algorithm
- Potential starvation issues

Scheduling algorithms

Multi-level feedback queue

- No predetermined classification
  - All process start from highest priority
- Dynamic change based on actual behavior
  - CPU-bound processes move to lower priorities
  - I/O-bound processes stay at or move up to higher priorities



Characteristics

- Responsiveness
- Low overhead
- Prevents starvation
  - Increase task's priority if not getting fair share
- Used in real OSes
  - Windows, macOS, Linux (old versions)