Операционные Системы

Потоки исполнения

April 28, 2017

Поток исполнения

- Поток исполнения - это код и его состояние (a. k. a. контекст)
  - код - набор инструкций в памяти, на который указывает регистр *RIP*;
  - контекст потока включает значения регистров и память.

Потоки и процессы

- Поток работает в контексте некоторого процесса
  - т. е. поток "живет" в логическом адресном пространстве процесса;
  - несколько потоков могут работать в рамках одного процесса;
  - процесс имеет как минимум один поток.

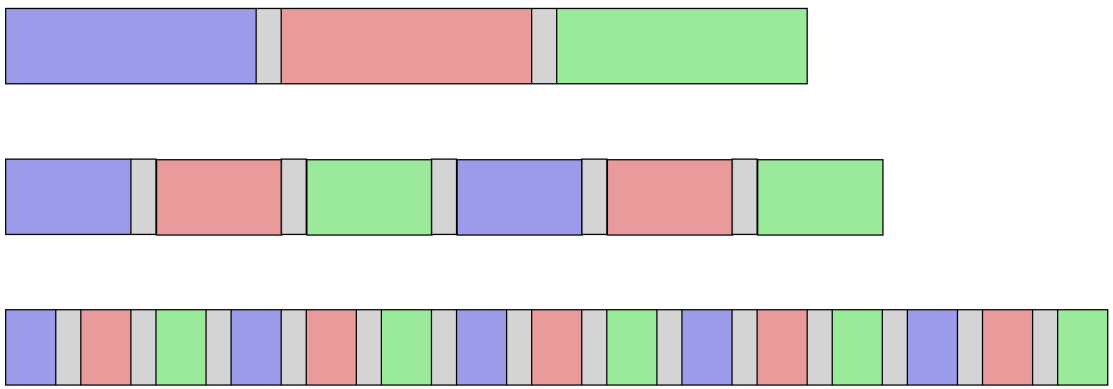
Стек потока

- Каждый поток исполнения имеет свой собственный стек
  - стек хранит адреса возвратов и локальные переменные;
  - для процессора стек - место в памяти, куда указывает *RSP*.

Многопоточность

- В системе могут *одновременно* работать несколько потоков исполнения
  - на нескольких ядрах процессора;
  - на одном ядре, создавая *иллюзию* одновременной работы.

Многопоточность



Переключение между потоками

- Для переключения между потоками необходимо:
  - сохранить контекст исполняемого потока;
  - восстановить контекст потока, на который мы переключаемся.

Пример переключения для x86

```
 1        .text
 2   switch_threads:
 3        pushq %rbx
 4        pushq %rbp
 5        pushq %r12
 6        pushq %r13
 7        pushq %r14
 8        pushq %r15
 9        pushfq
10
11        movq %rsp, (%rdi)
12        movq %rsi, %rsp
13
14        popfq
15        popq %r15
16        popq %r14
17        popq %r13
18        popq %r12
19        popq %rbp
20        popq %rbx
21
22        retq
```

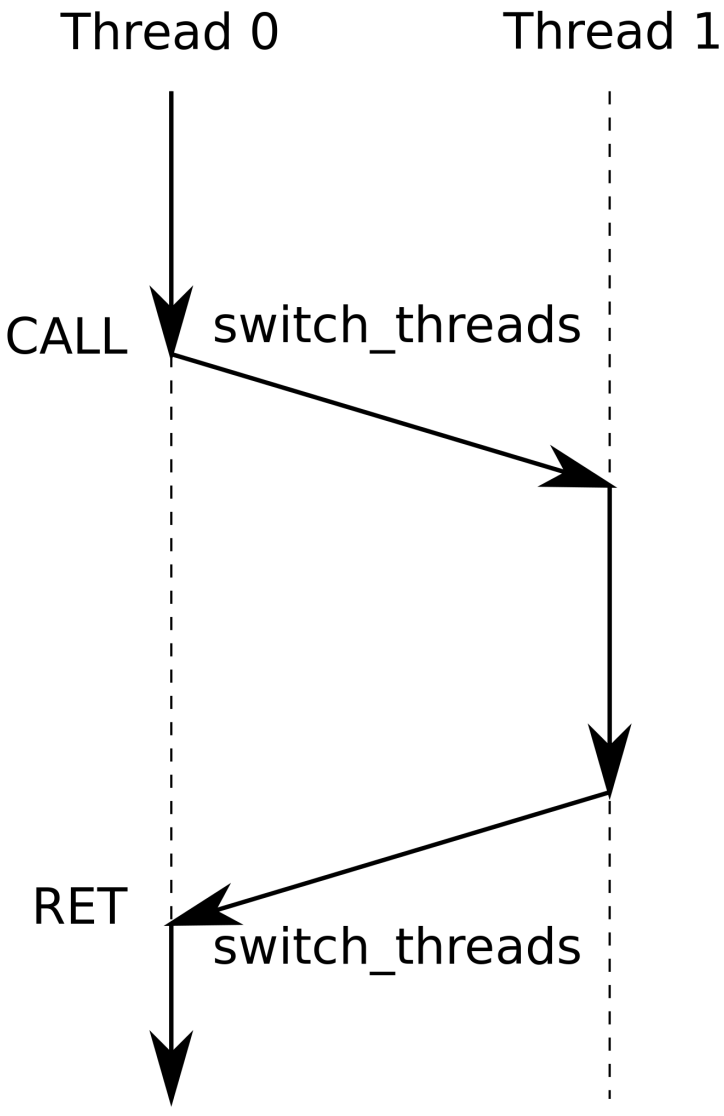
C API

- void switch_threads(void **prev, void *next);
- по завершении функции *prev будет указывать на сохраненный контекст;
- *next* указывает на сохраненный контекст потока, на который мы переключаемся.

switch_threads

```
 1        .text
 2   switch_threads:
 3        /* save contex on stack */
 4        pushq %rbx
 5        pushq %rbp
 6        pushq %r12
 7        pushq %r13
 8        pushq %r14
 9        pushq %r15
10        pushfq
11
12        /* rdi - the first argument */
13        movq %rsp, (%rdi)
14
15        /* rsi - the second argument */
16        movq %rsi, %rsp
17
18        /* restore from stack */
19        popfq
20        popq %r15
21        popq %r14
22        popq %r13
23        popq %r12
24        popq %rbp
25        popq %rbx
26
27        retq /* ! */
```

Переключение потоков



Создание нового потока

- Как создать новый поток и переключиться на него в первый раз?
  - нам нужно выделить место для хранения указателя на контекст;
  - нам нужно выделить место под стек нового потока;
  - нам нужно сохранить на стеке начальный контекст и сохранить указатель на него.

Начальный контекст

```
1   struct switch_frame {
2       uint64_t rflags;
3       uint64_t r15;
4       uint64_t r14;
5       uint64_t r13;
6       uint64_t r12;
7       uint64_t rbp;
8       uint64_t rbx;
9       uint64_t rip;
10  } __attribute__((packed));
```

Кооперативная многозадачность

- Невытесняющая (кооперативная) многозадачность
  - поток должен сам вызвать функцию переключения;
  - что если в коде содержится ошибка?
  - или мы обращаемся к библиотеке, которая выполняет долгую операцию?

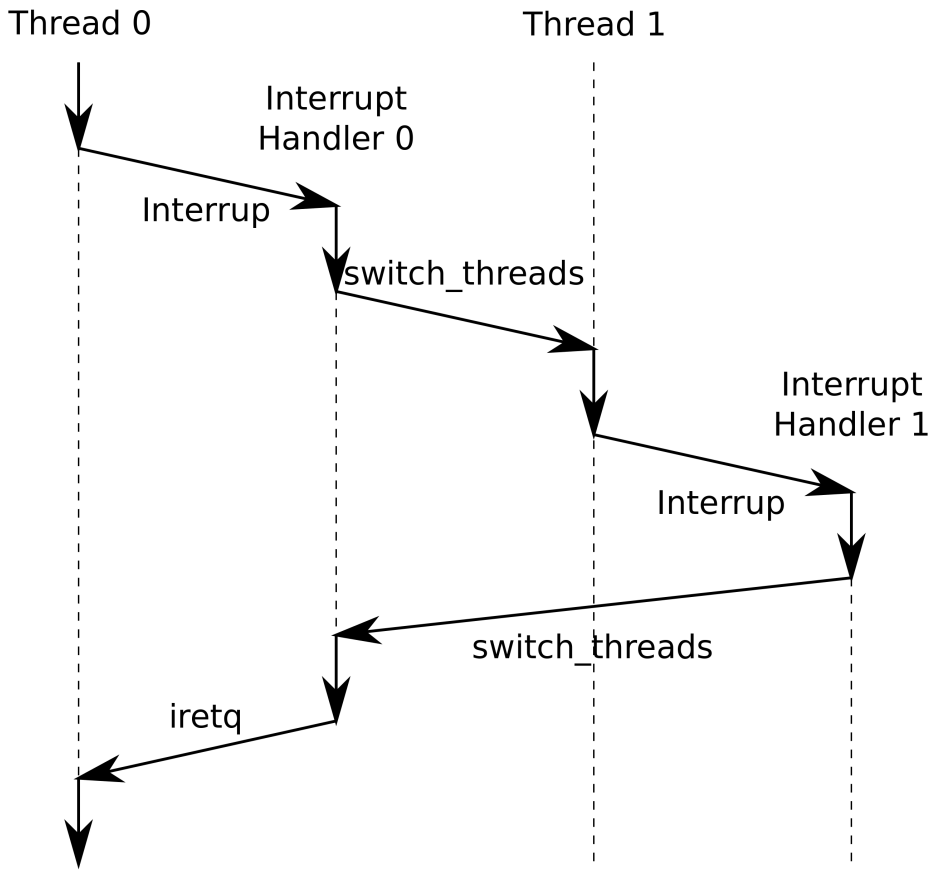
Вытесняющая многозадачность

- Вытесняющая (preemptive) многозадачность
  - поток снимается ОС с CPU "силой", по истечении *кванта* времени;
  - синхронизация потоков при этом усложняется;
  - как организовать вытесняющую многозадачность?

Сново о прерываниях

- Обработчик прерывания "прерывает" исполняемый код
  - но обработчик работает в контексте прерванного потока;
  - функцию переключения контекста можно вызвать от имени потока из обработчика прерываний.

Вытесняющая многозадачность



Таймер

- Таймер может генерировать прерывания с заданной периодичностью
  - Programmable Interval Timer (PIT, intel 8253) - IBM PC;
  - High Precision Event Timer (HPET);
  - Local APIC Timer.

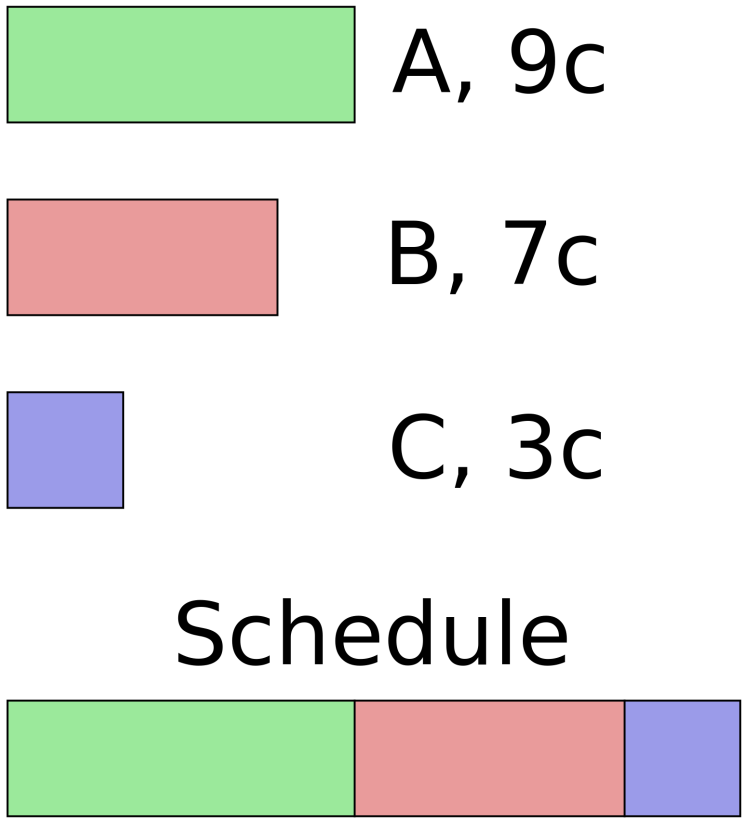
Планирование потоков

- Планировщик (scheduler) - компонент ОС, который определяет
  - когда переключаться с потока;
  - на какой поток переключаться.

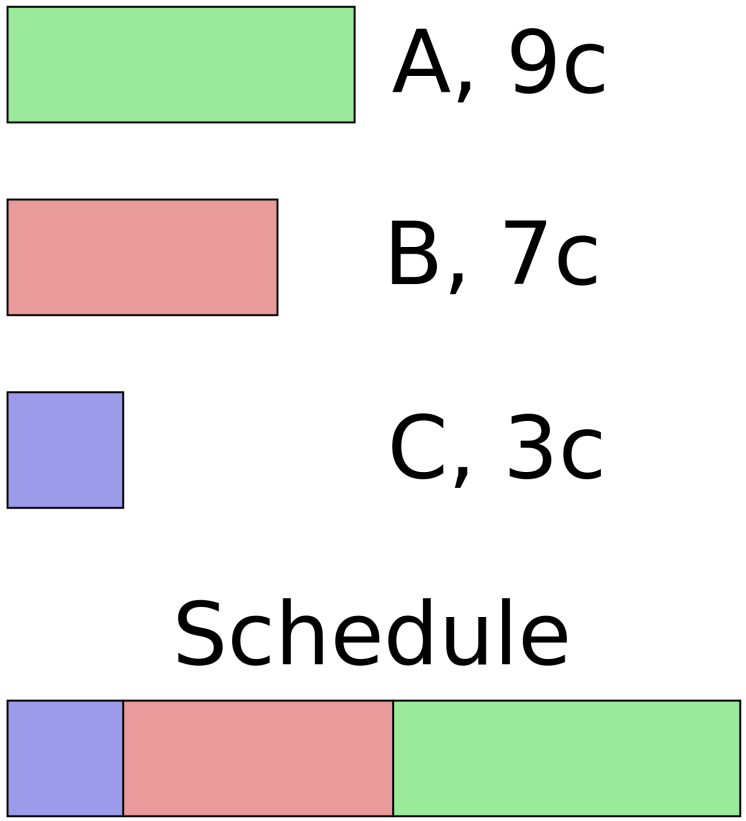
Простое планирование

- Рассмотрим простейшую задачу планирования
  - все задачи известны заранее;
  - про каждую задачу известно, сколько времени она займет;
  - задачи работают без переключений;
  - т. е. нам осталось только определить порядок.

Пропускная способность



Пропускная способность



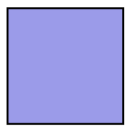
Среднее время ожидания

- Пусть все задачи принадлежат разным пользователям
  - пользователю важно, сколько ему нужно ждать завершения его задачи;
  - давайте в качестве метрики использовать среднее время ожидания.

Среднее время ожидания

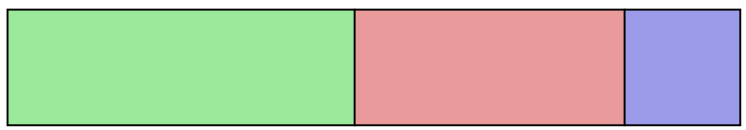
A, 9c

B, 7c



C, 3c

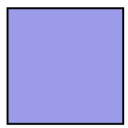
Schedule



Среднее время ожидания

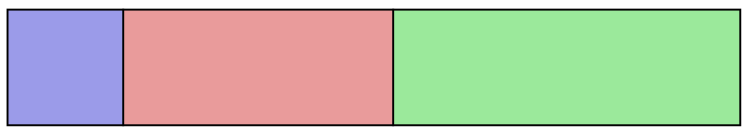
A, 9c

B, 7c



C, 3c

Schedule



Динамическое создание задач

- Зачастую все задачи не известны заранее
  - задачи могут создаваться в произвольные моменты времени;
  - каждая вновь появившаяся задача может изменить решение планировщика.

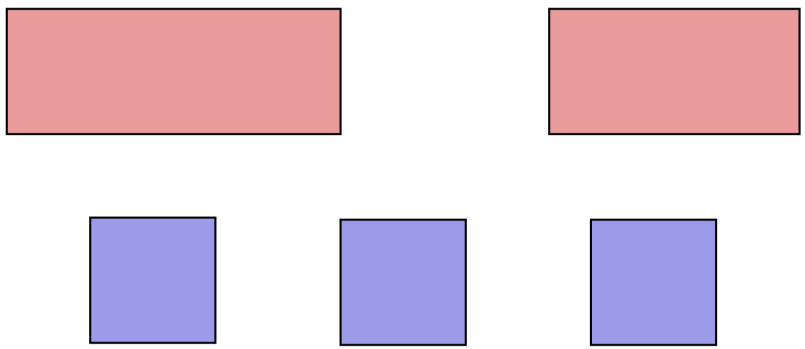
IO операции

- Задачи могут давать команды устройствам или ждать каких-то событий:
  - запись/чтение на/с HDD (порядка нескольких мс);
  - ждать входящих соединений по сети;
  - ждать, пока пользователь нажмет на клавишу.

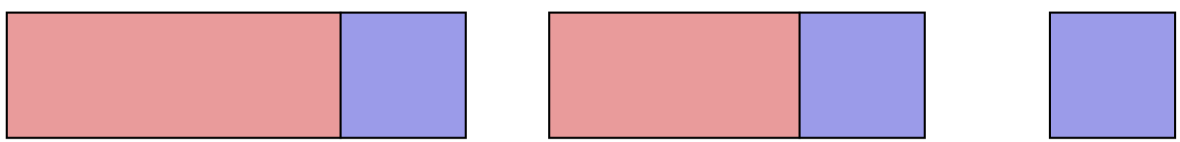
IO операции

- Пока задача ждет завершения IO операции, можно забрать у нее CPU
  - время ожидания может быть большим (1мс - очень много для CPU);
  - утилизация CPU - сколько времени CPU делал полезную работу.

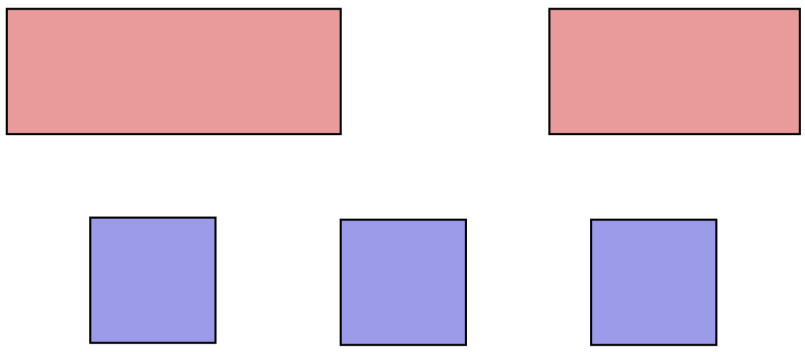
Утилизация



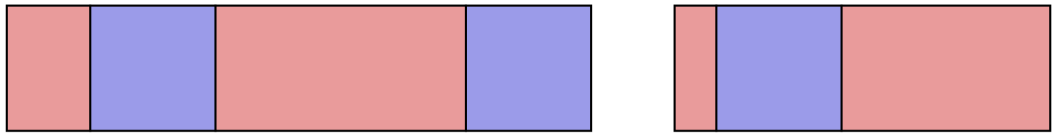
Schedule



Утилизация



Schedule



Информация о задаче

- Зачастую время работы задачи и расписание ее IO не известны
  - задачи могут влиять друг на друга или зависеть от внешних обстоятельств;
  - мы можем оценивать эти параметры и классифицировать задачи.

IO-bounded и CPU-bounded

- IO-bounded задачи - много IO, но мало вычислений:
  - например, текстовый редактор;
  - вообще приложения, ожидающие ввода пользователя.
- CPU-bounded задачи - много вычислений, но мало IO:
  - например, научные вычисления;
  - компиляция программ.

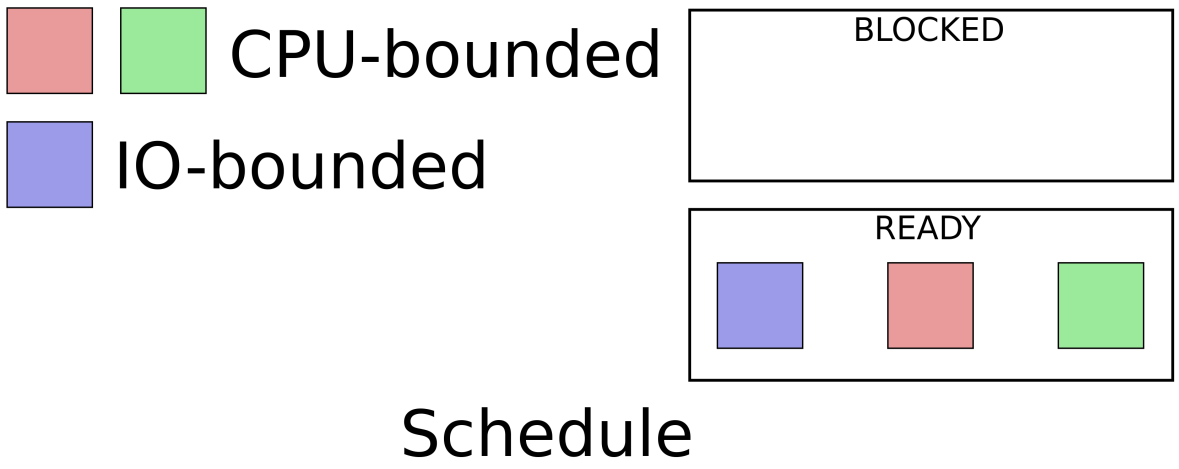
Round Robin

- Round Robin - выдаем потокам квант времени на CPU по очереди:
  - каждый новый поток встает в конец очереди;
  - потоки, дождавшиеся завершения, IO встают в конец очереди;
  - CPU отдается потоку в начале очереди;
  - поток, отработавший свой квант, встает в конец очереди.

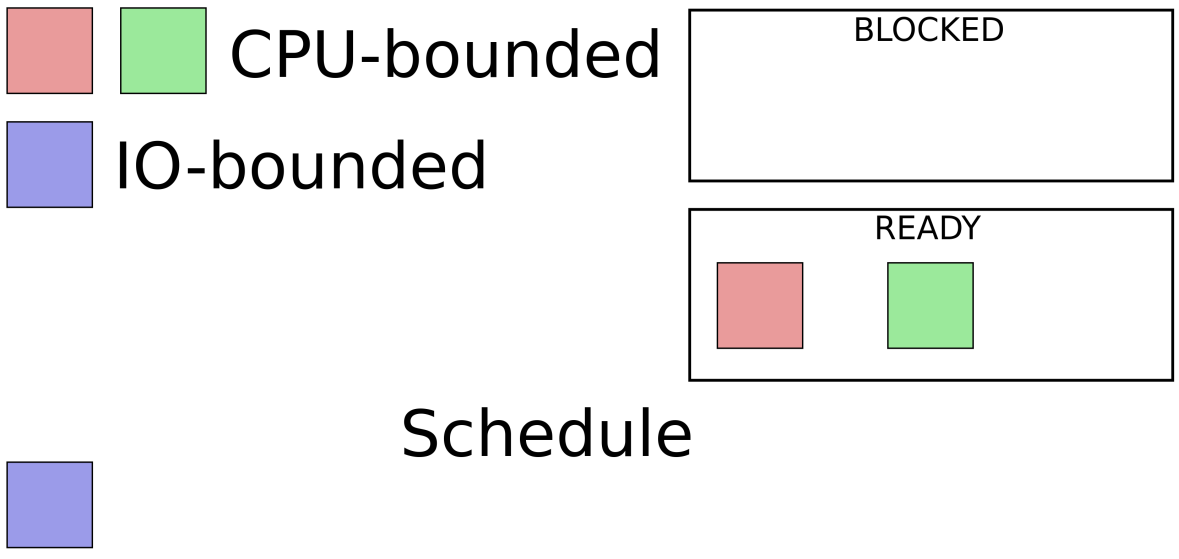
Достоинства Round Robin

- К списку достоинств Round Robin можно отнести:
  - подход очень прост;
  - время ожидания CPU ограничено - никто не голодает.

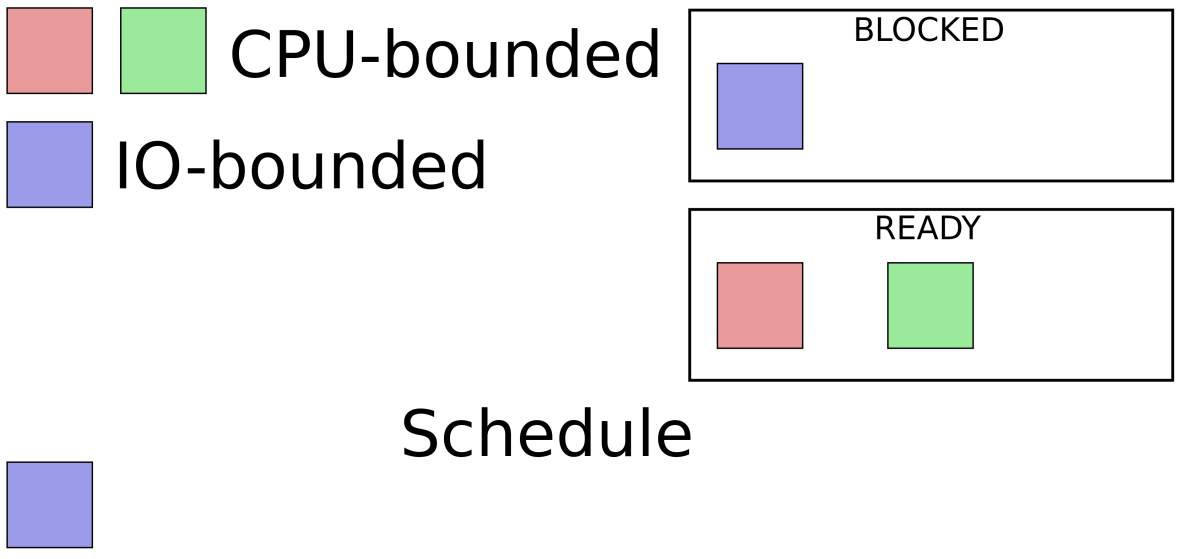
Round Robin



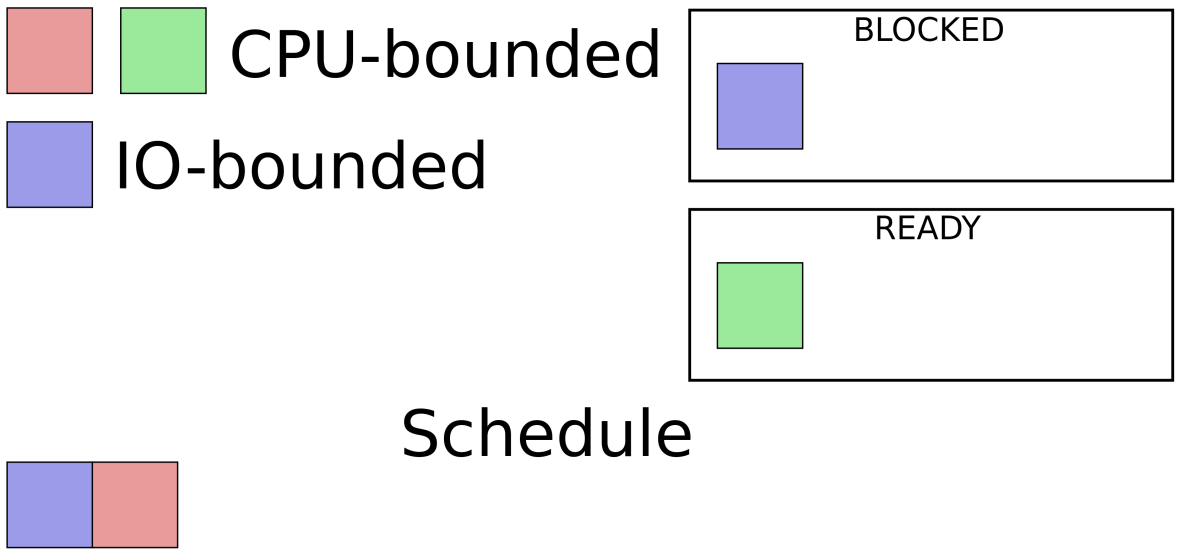
Round Robin



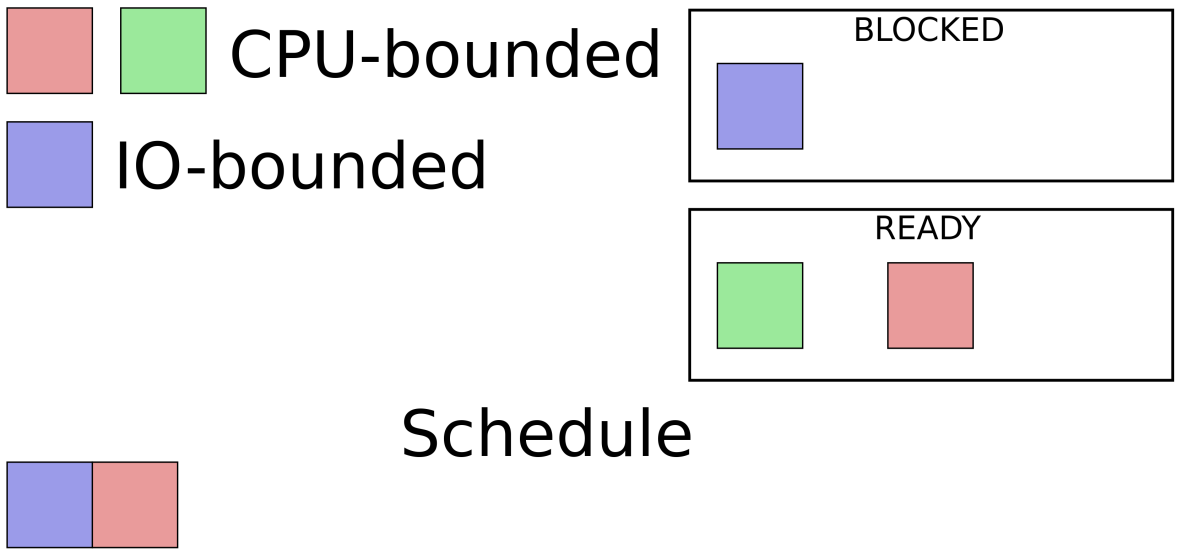
Round Robin



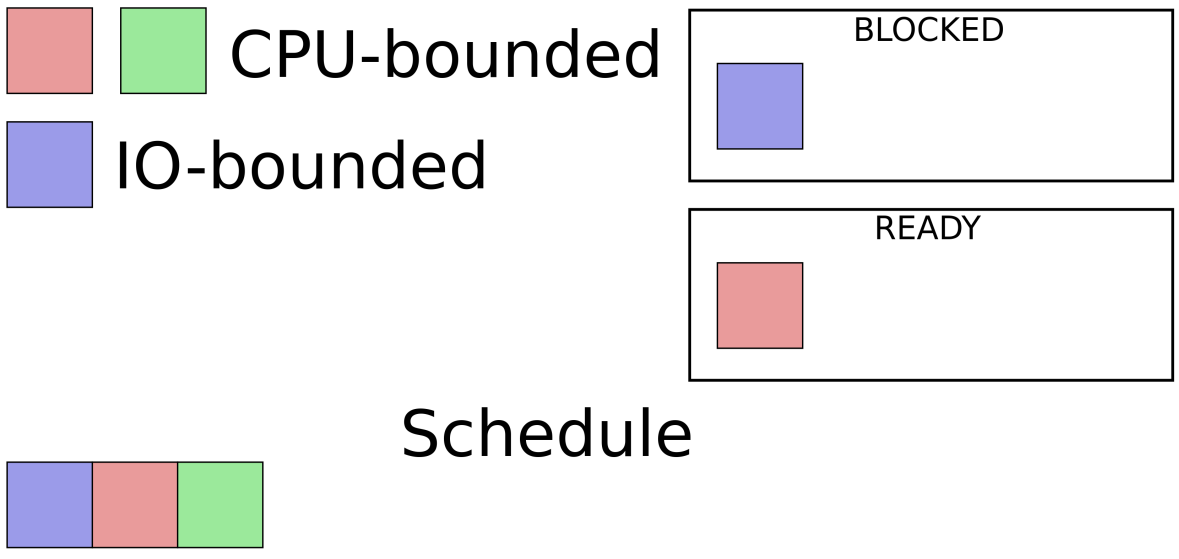
Round Robin



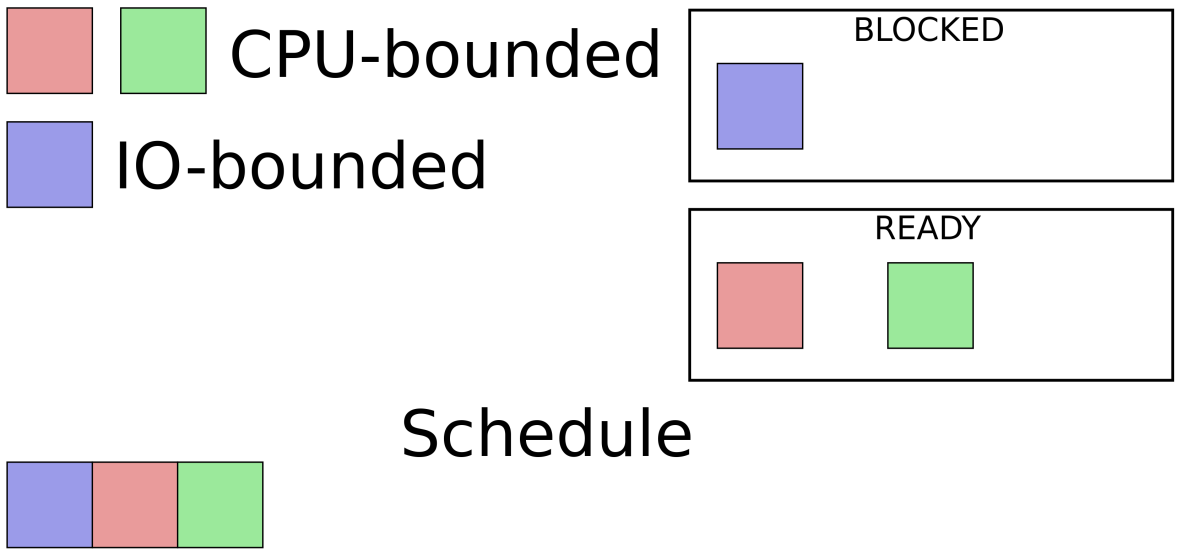
Round Robin



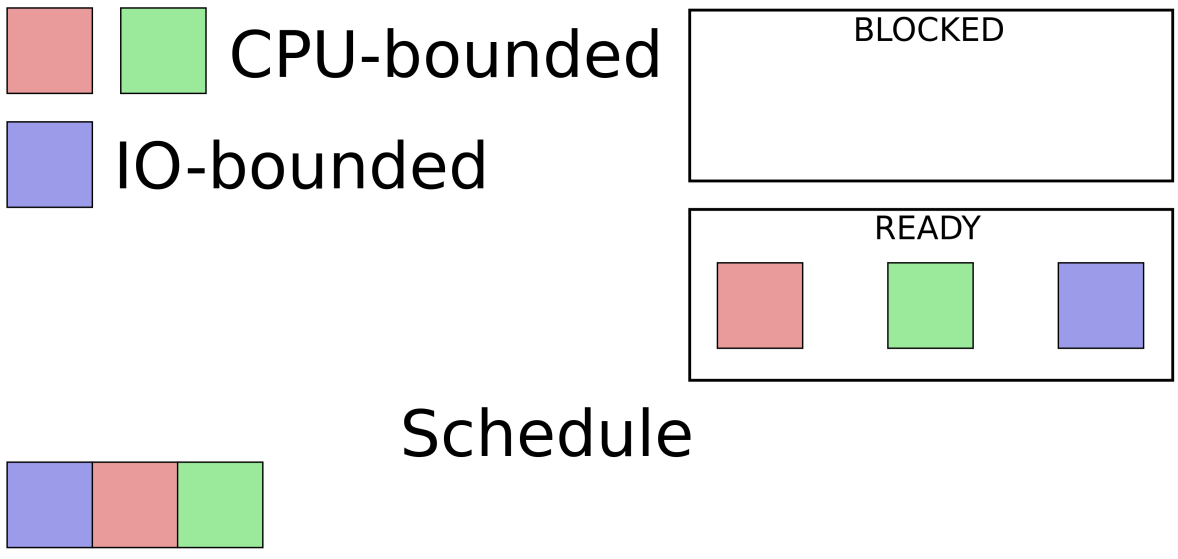
Round Robin



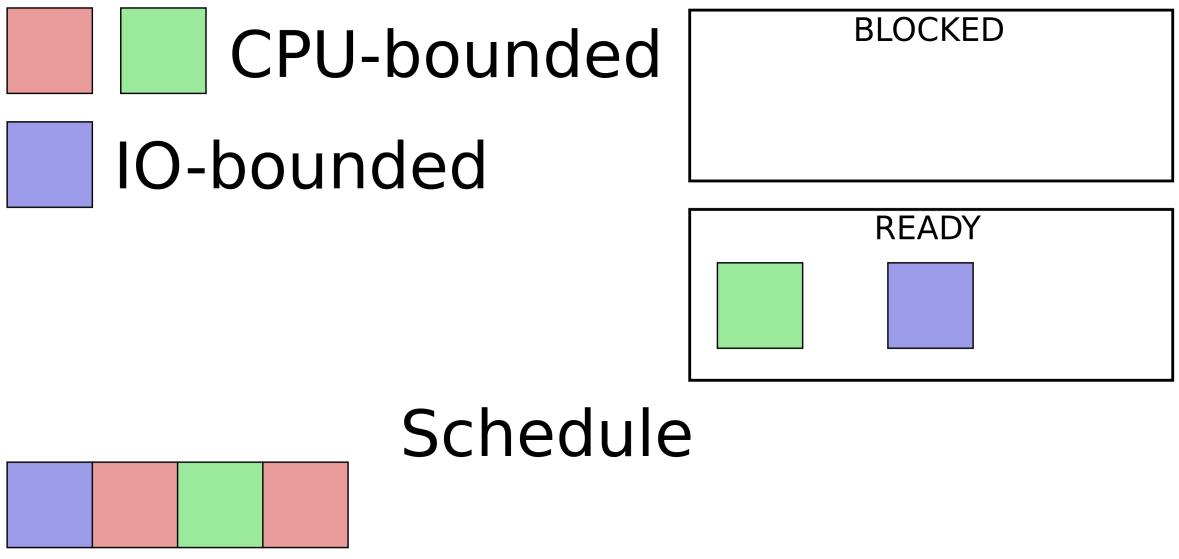
Round Robin



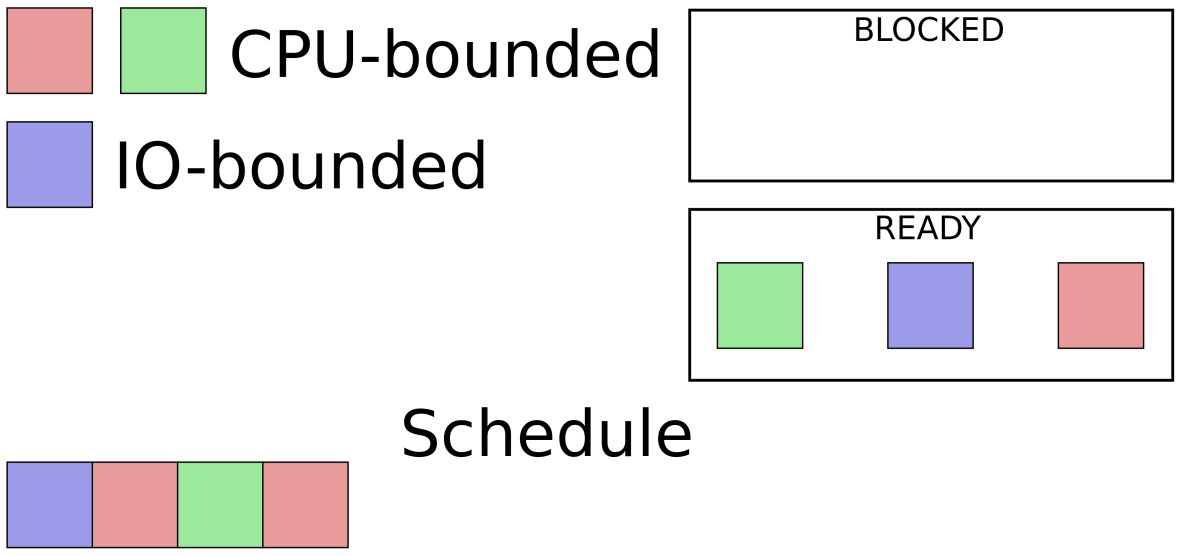
Round Robin



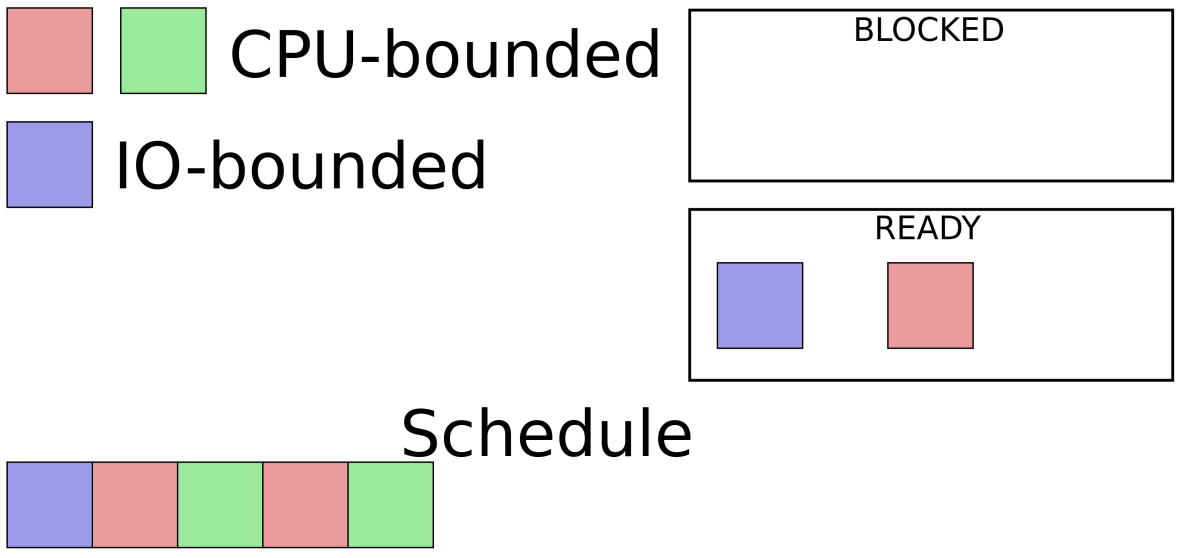
Round Robin



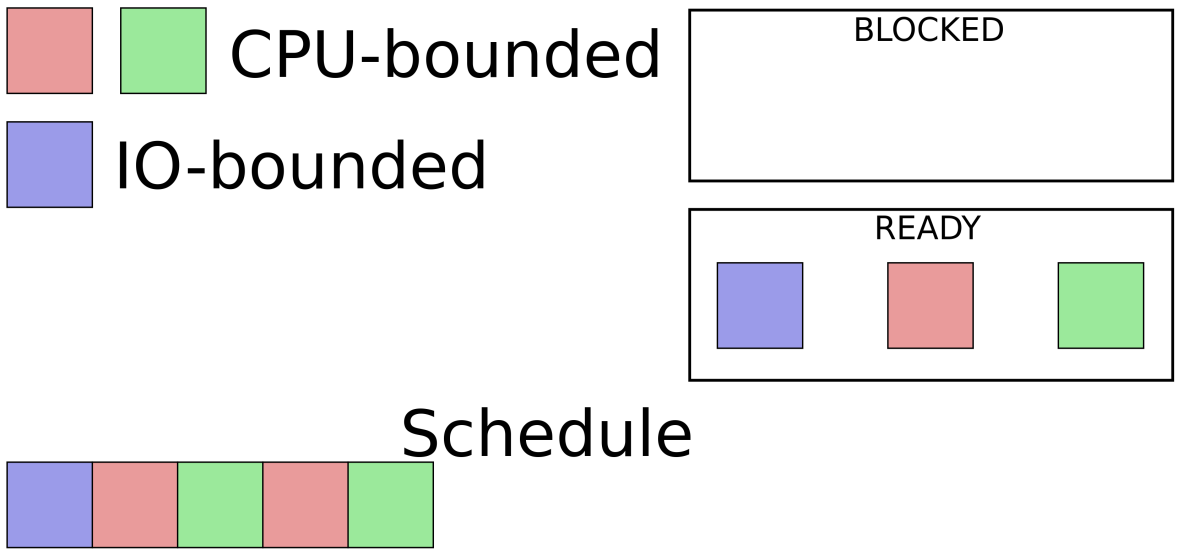
Round Robin



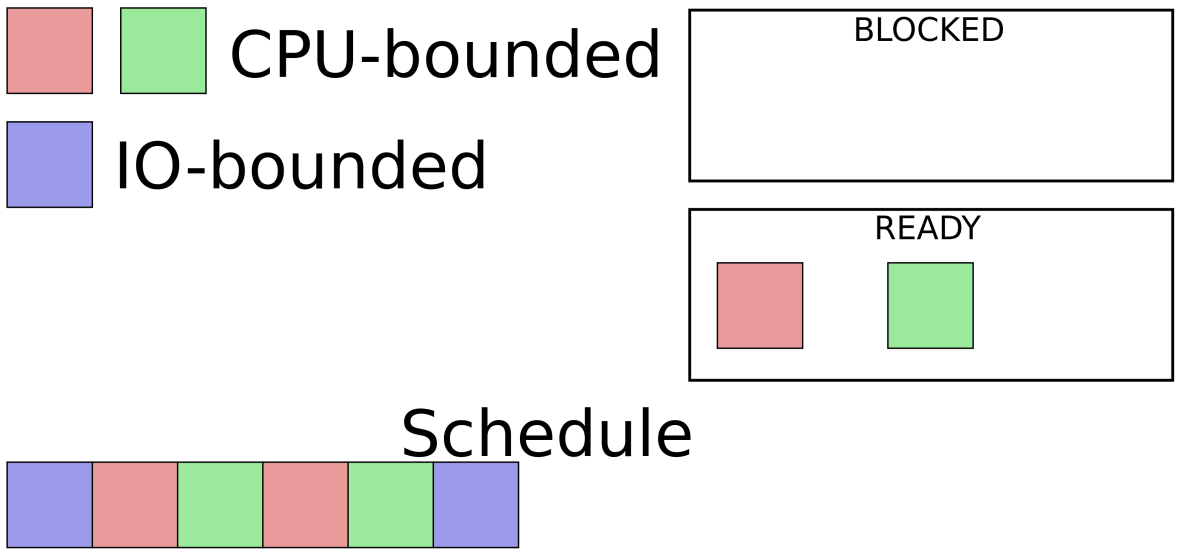
Round Robin



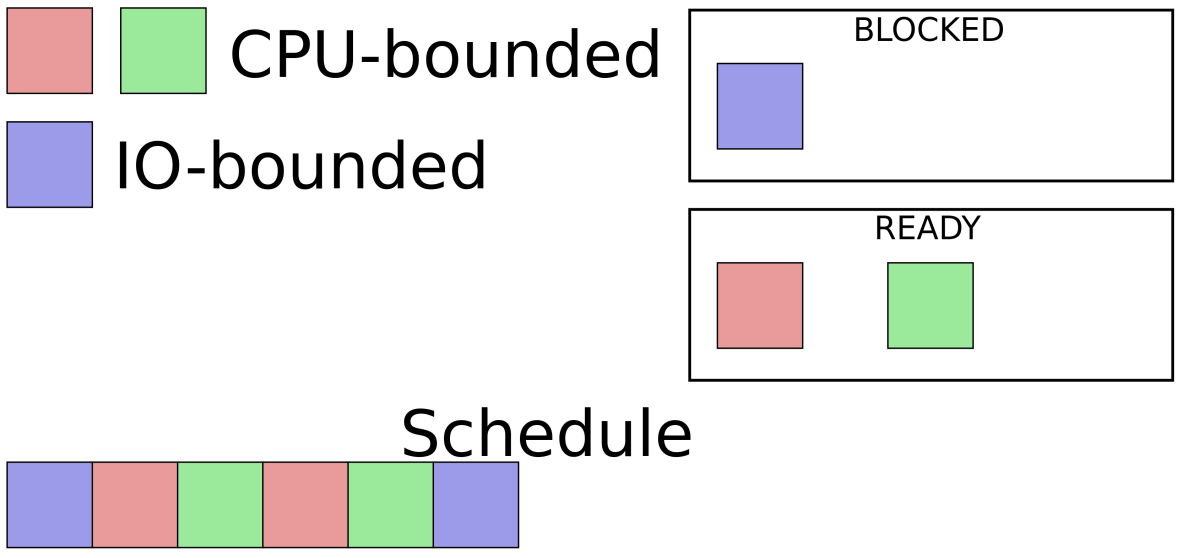
Round Robin



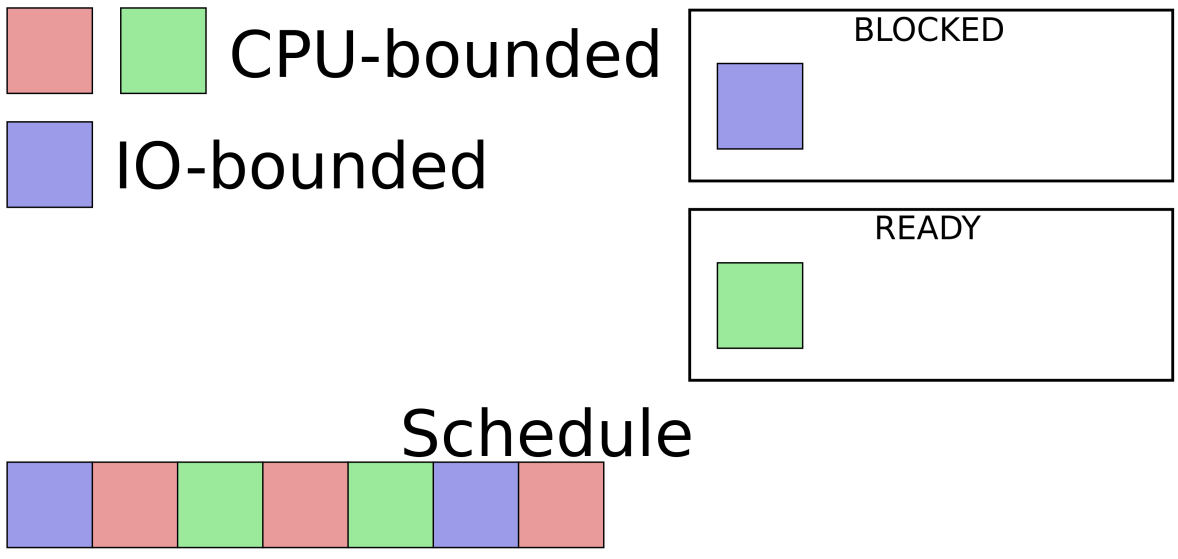
Round Robin



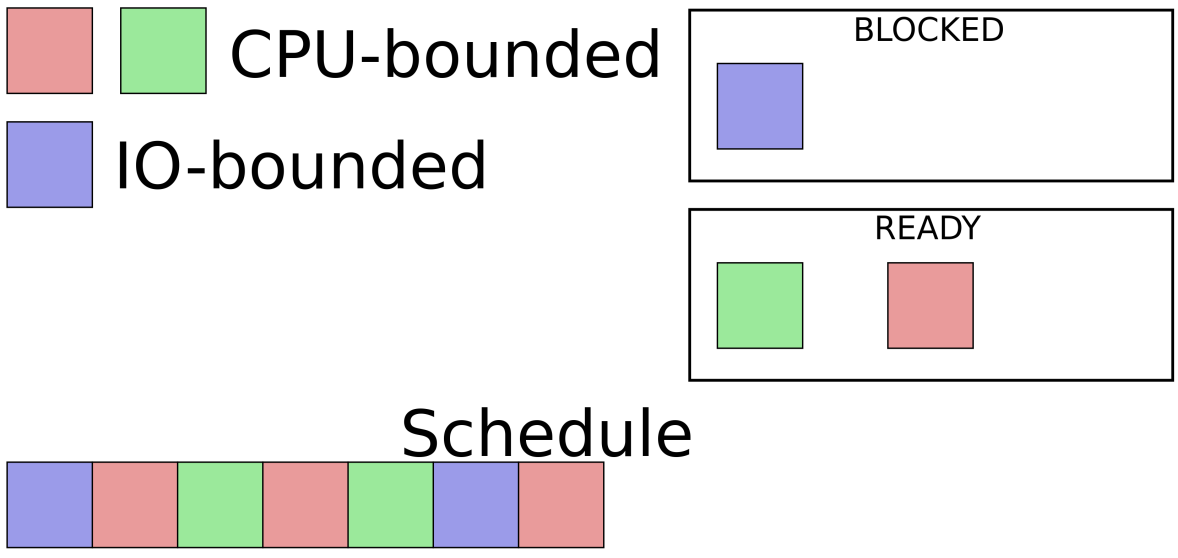
Round Robin



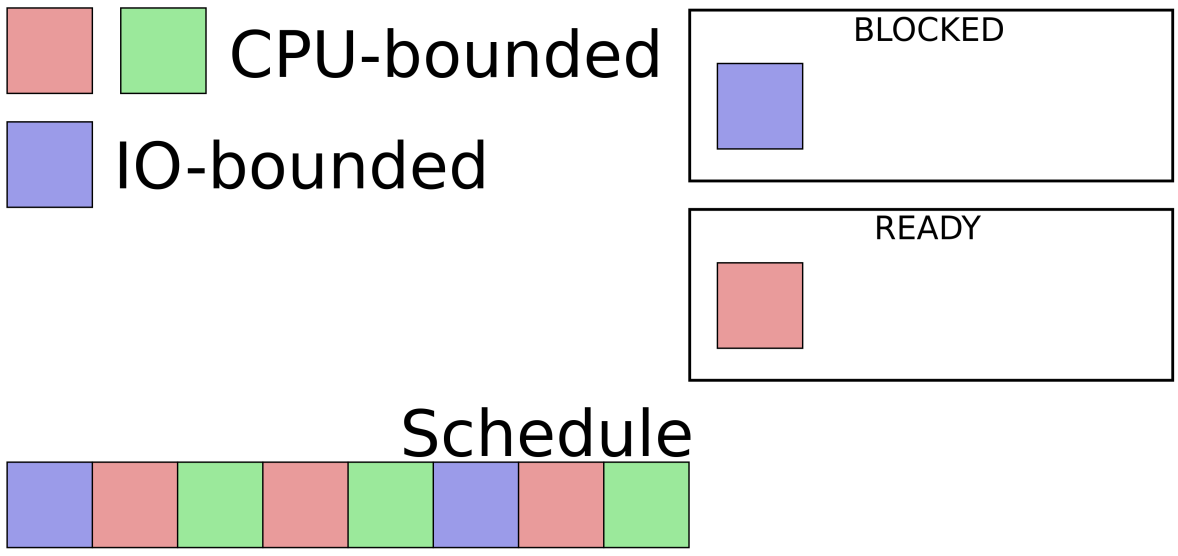
Round Robin



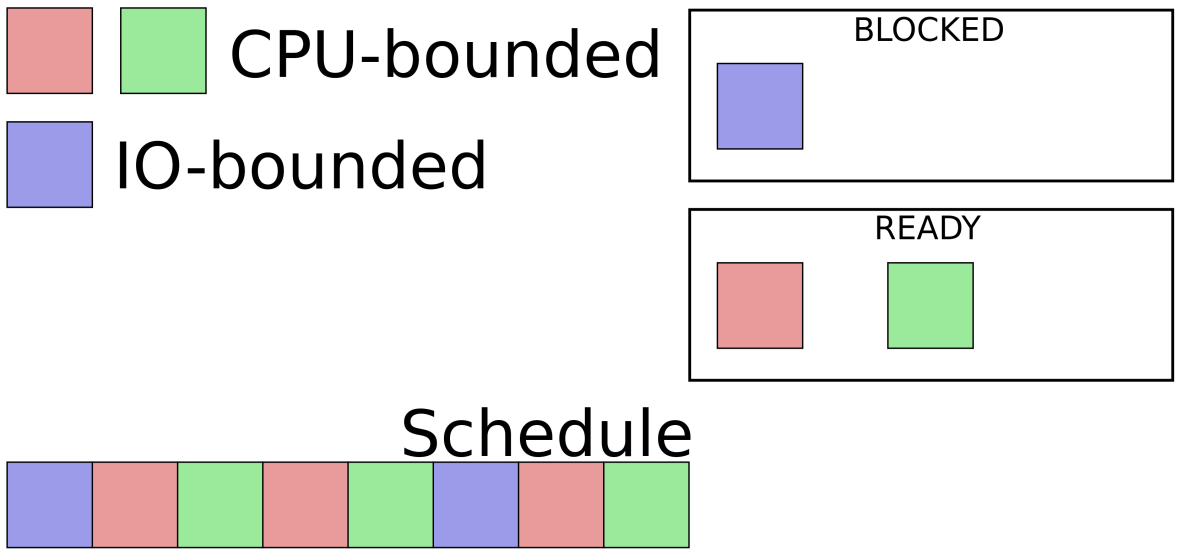
Round Robin



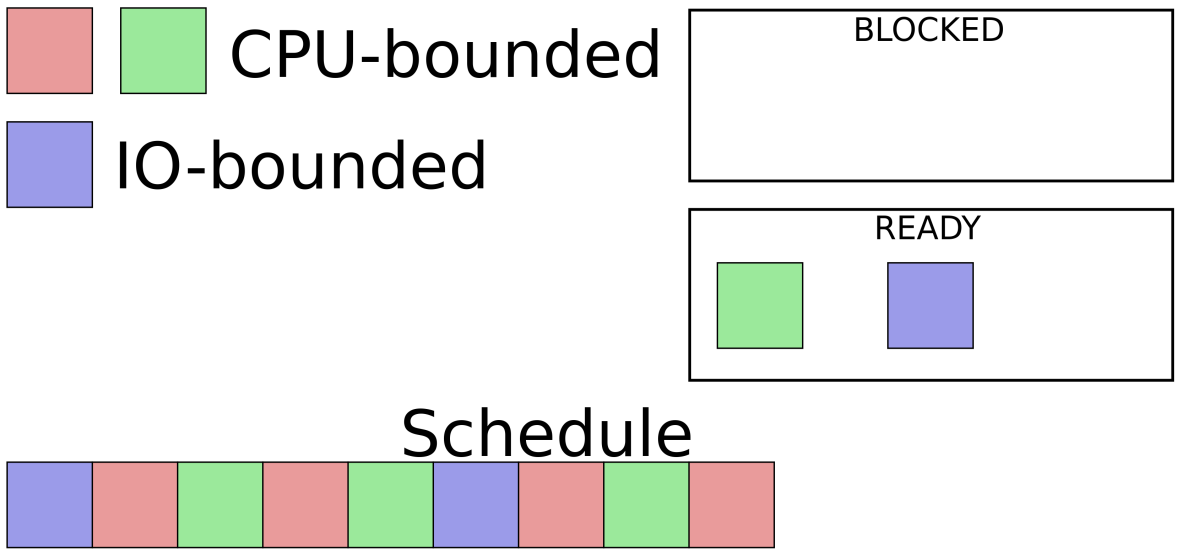
Round Robin



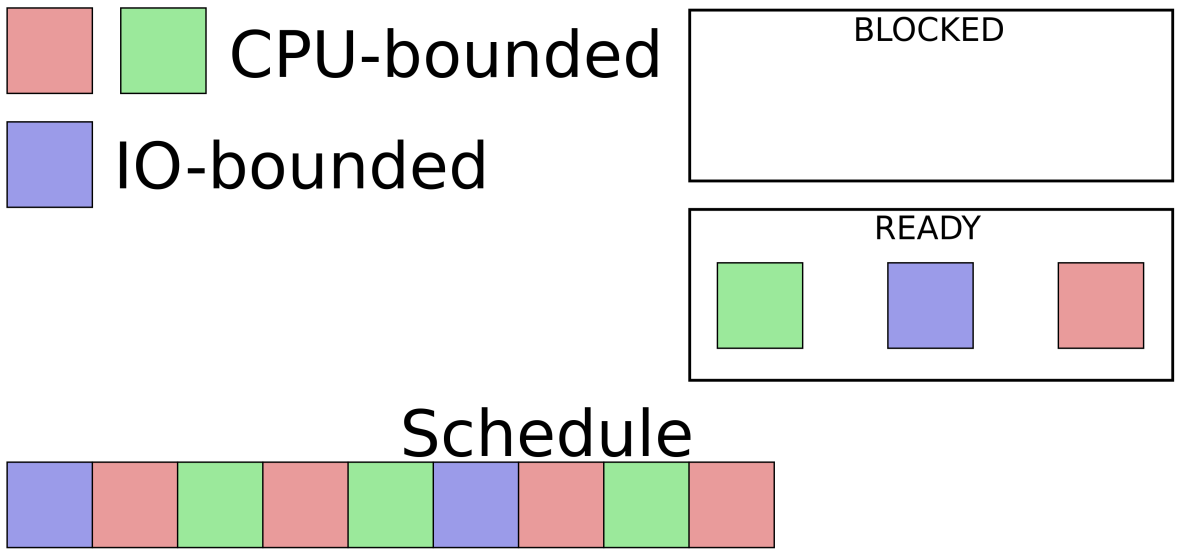
Round Robin



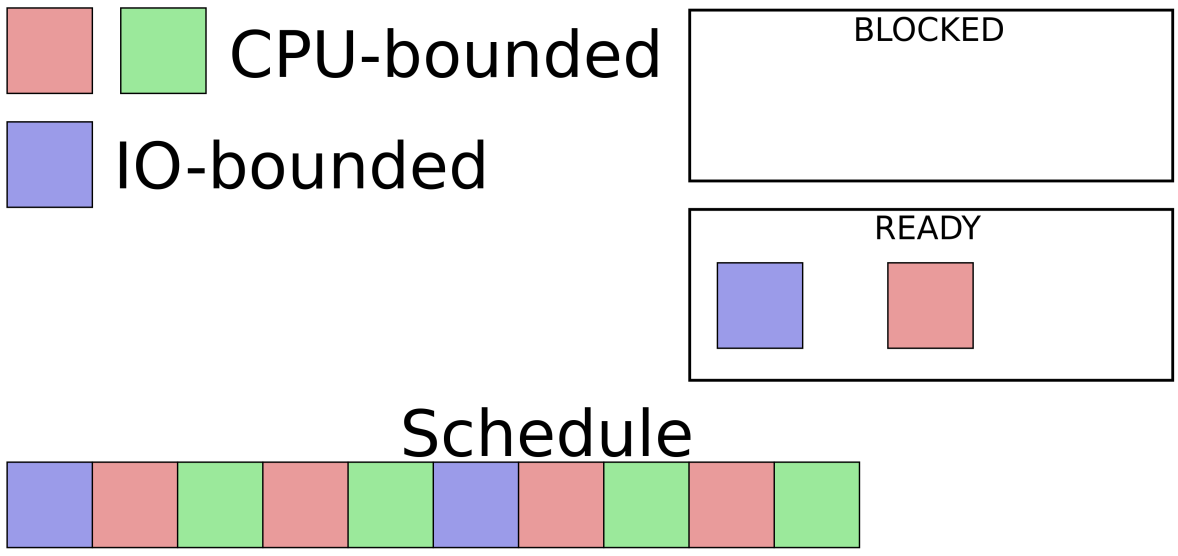
Round Robin



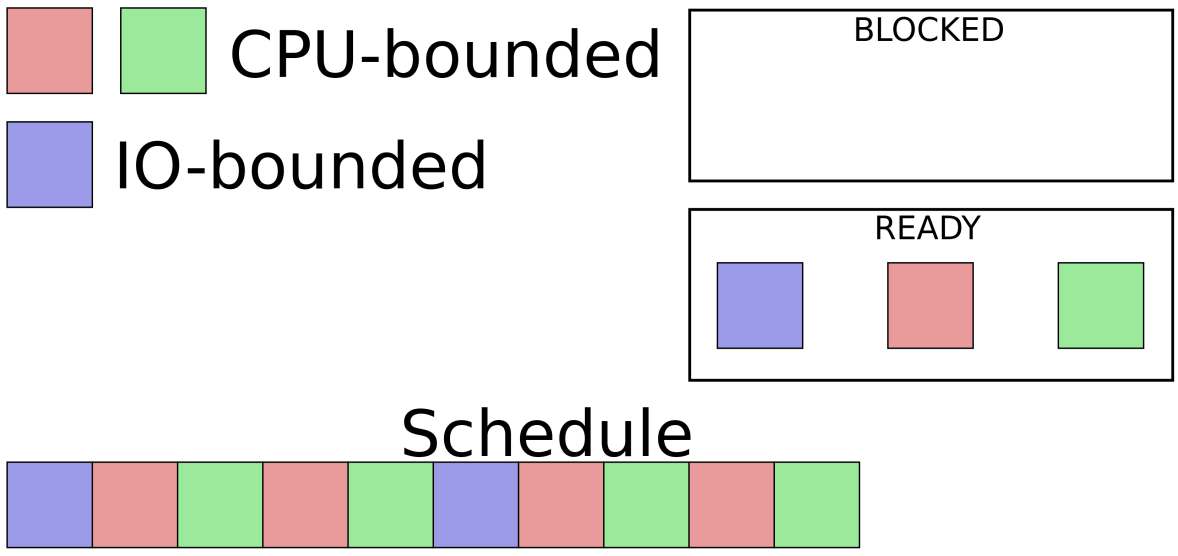
Round Robin



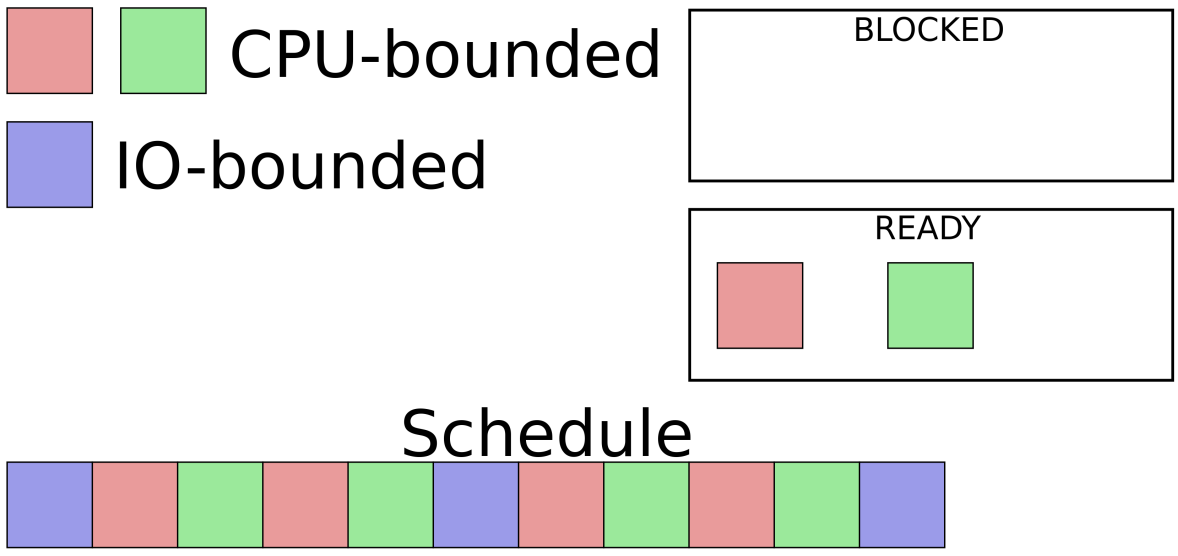
Round Robin



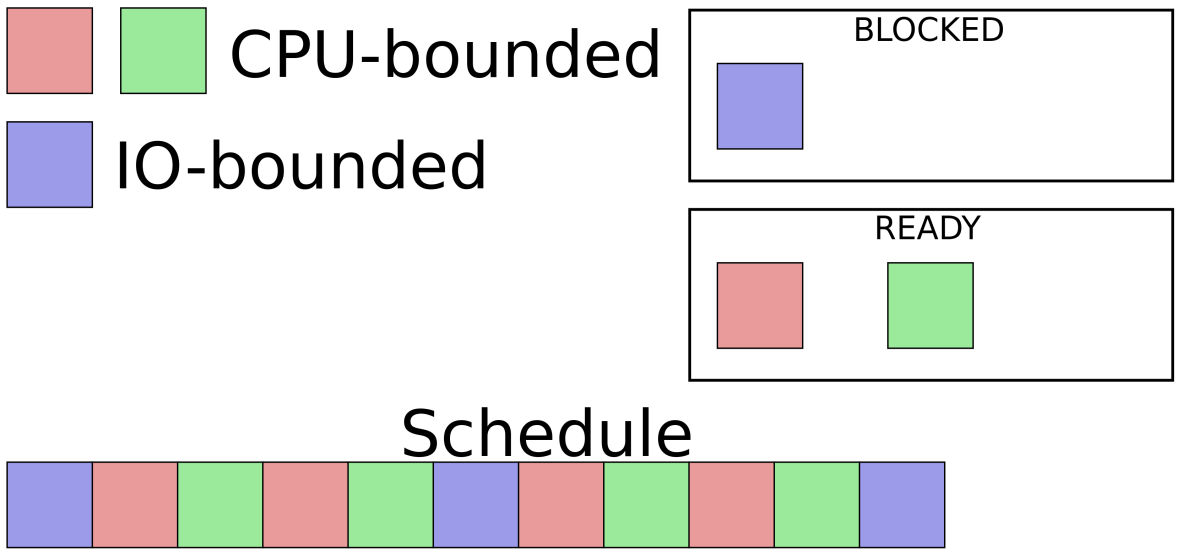
Round Robin



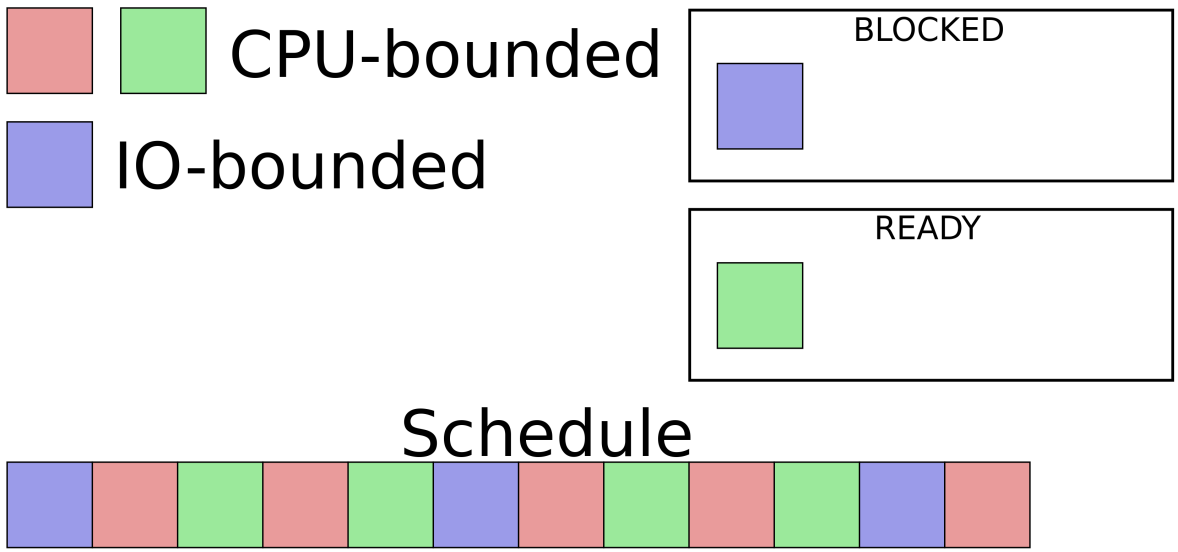
Round Robin



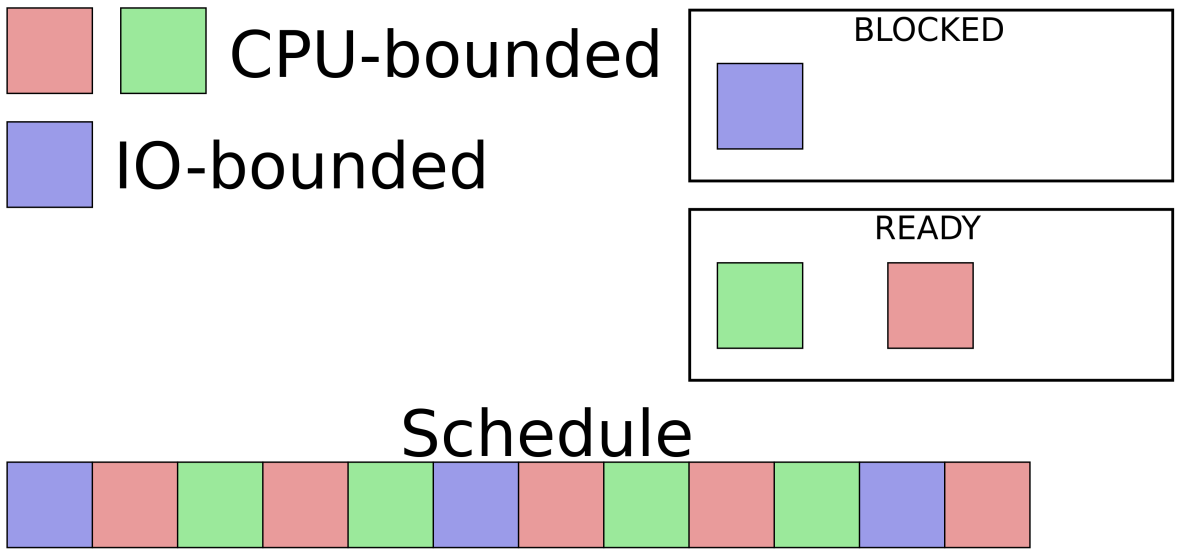
Round Robin



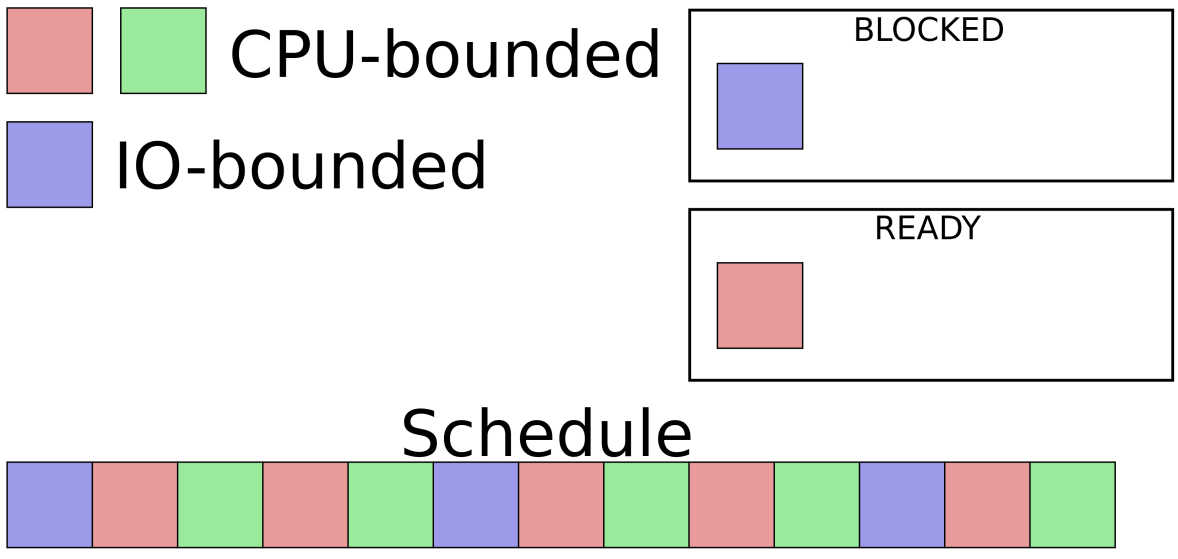
Round Robin



Round Robin



Round Robin



Выбор кванта времени

- Из каких соображений стоит выбирать квант времени?
  - чем больше квант
    - тем меньше доля времени на переключение;
    - тем больше время отклика;
  - чем меньше квант
    - тем больше доля времени на переключение;
    - тем меньше время отклика.

Выбор кванта времени