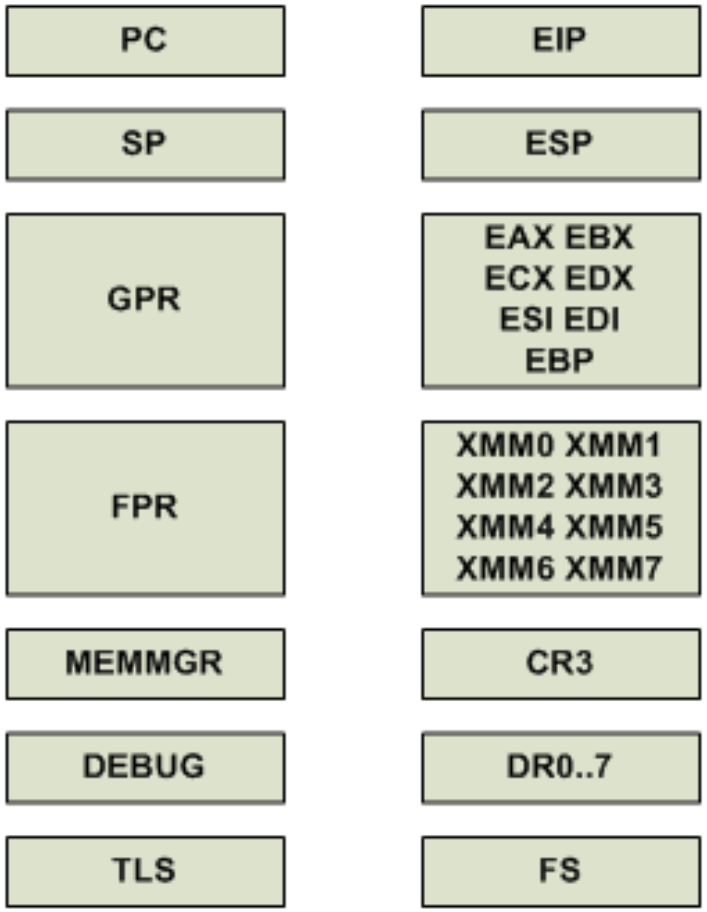
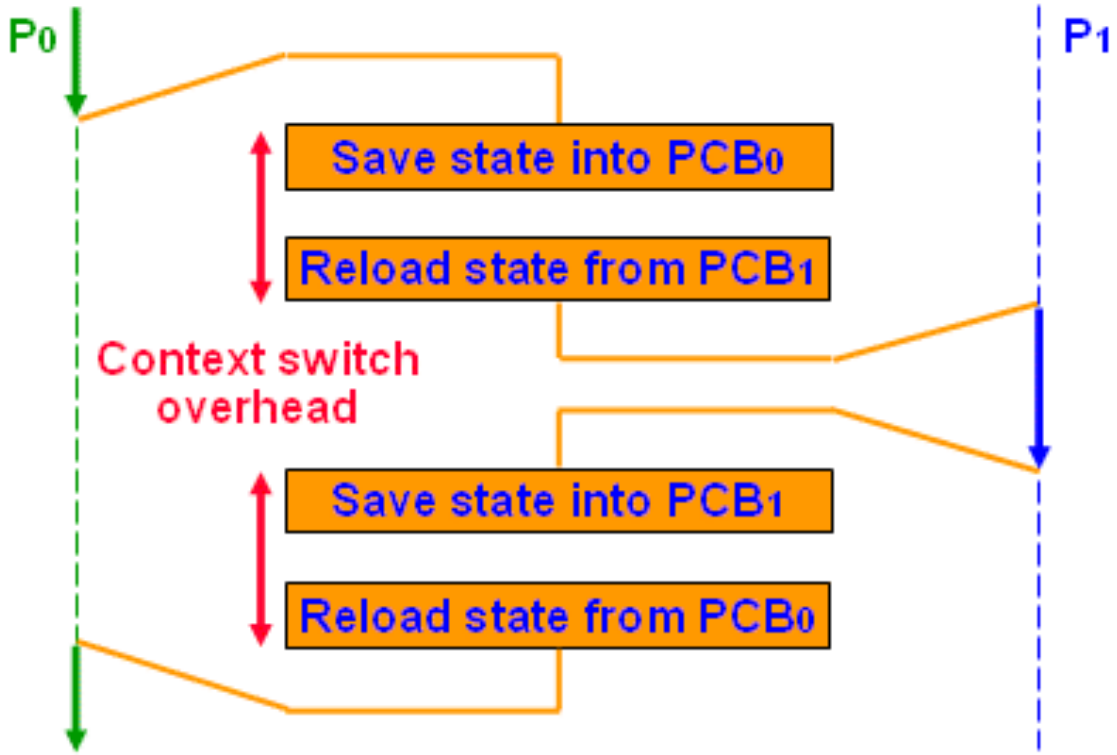
Процессы и потоки. Планирование задач.

Инициализация процессорных ресурсов

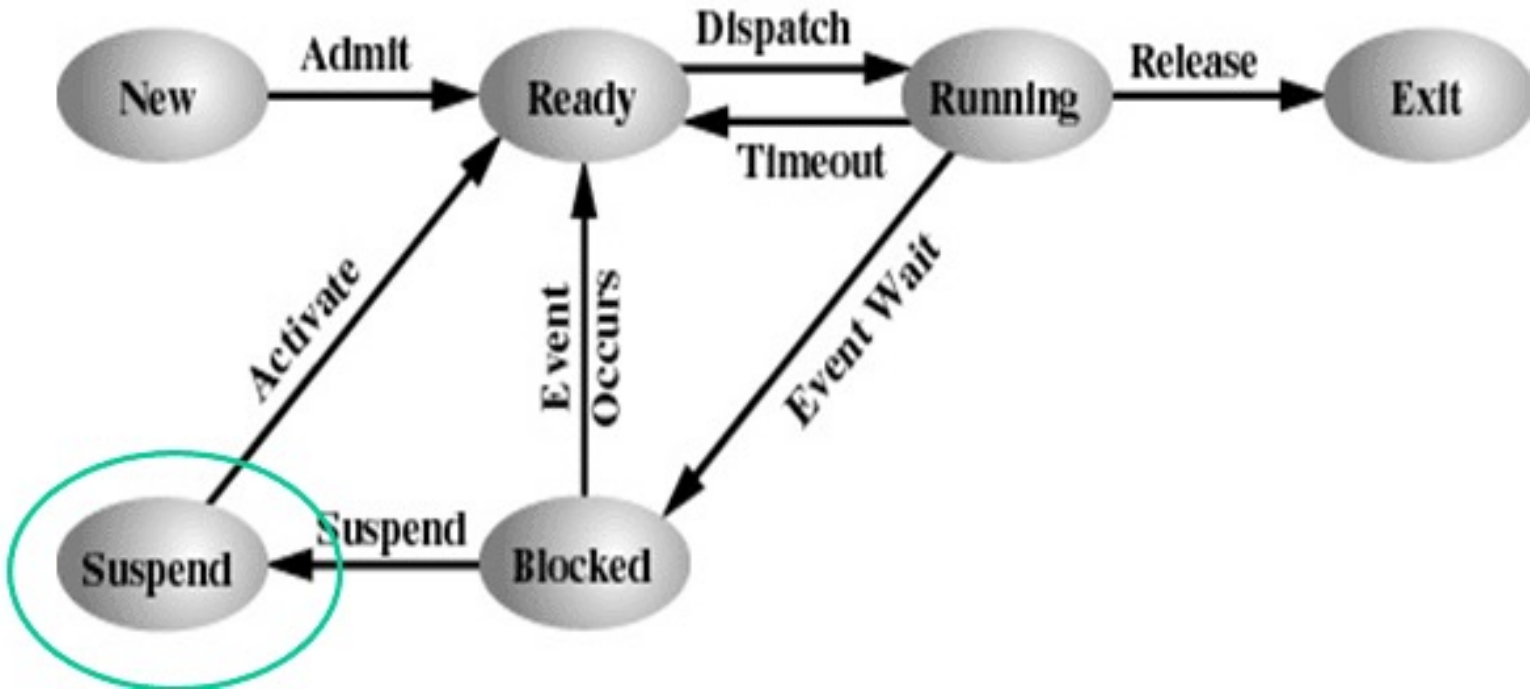
Контекст выполнения процесса



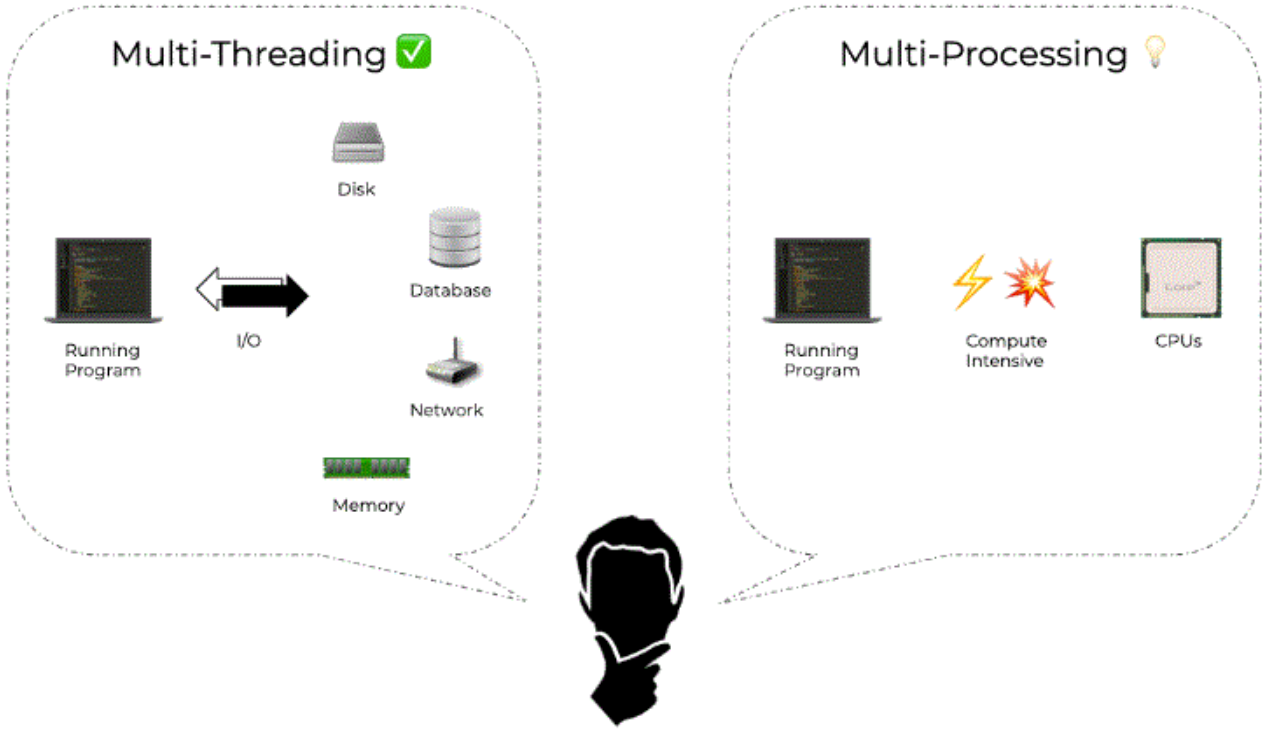
Переключения контекста



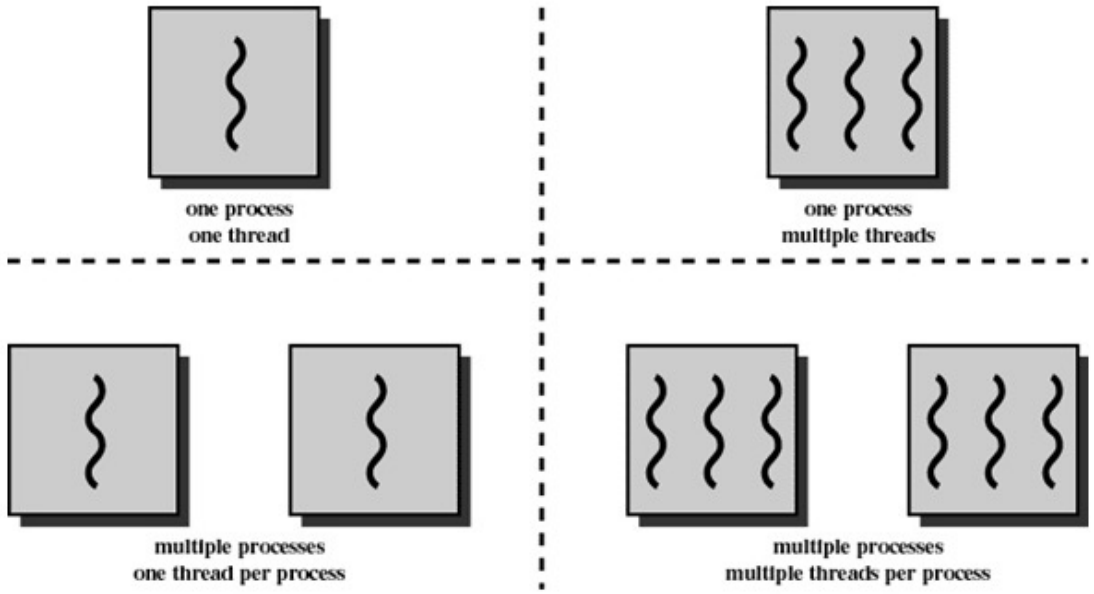
Состояния нити



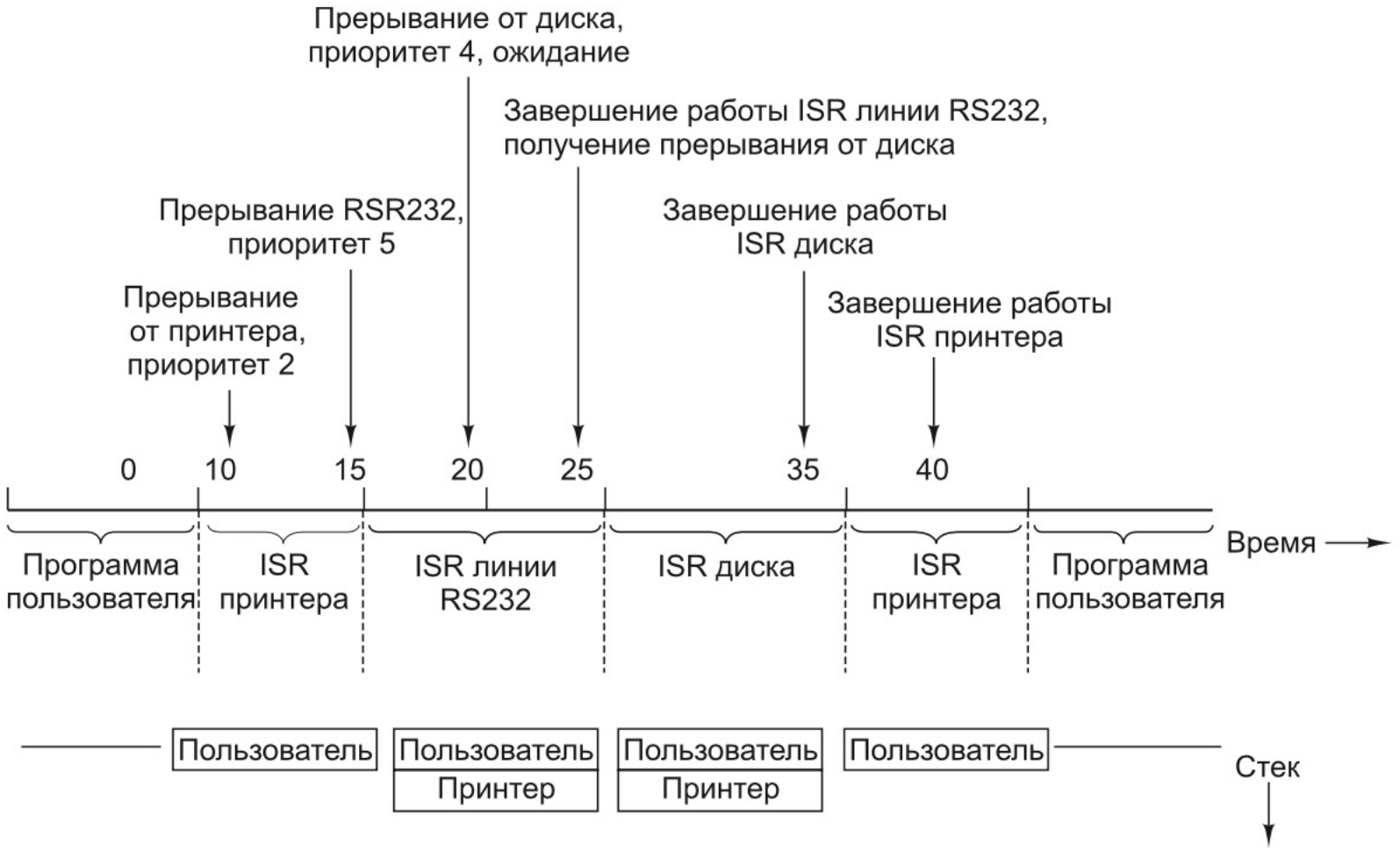
Процессы и потоки



Процессы и потоки

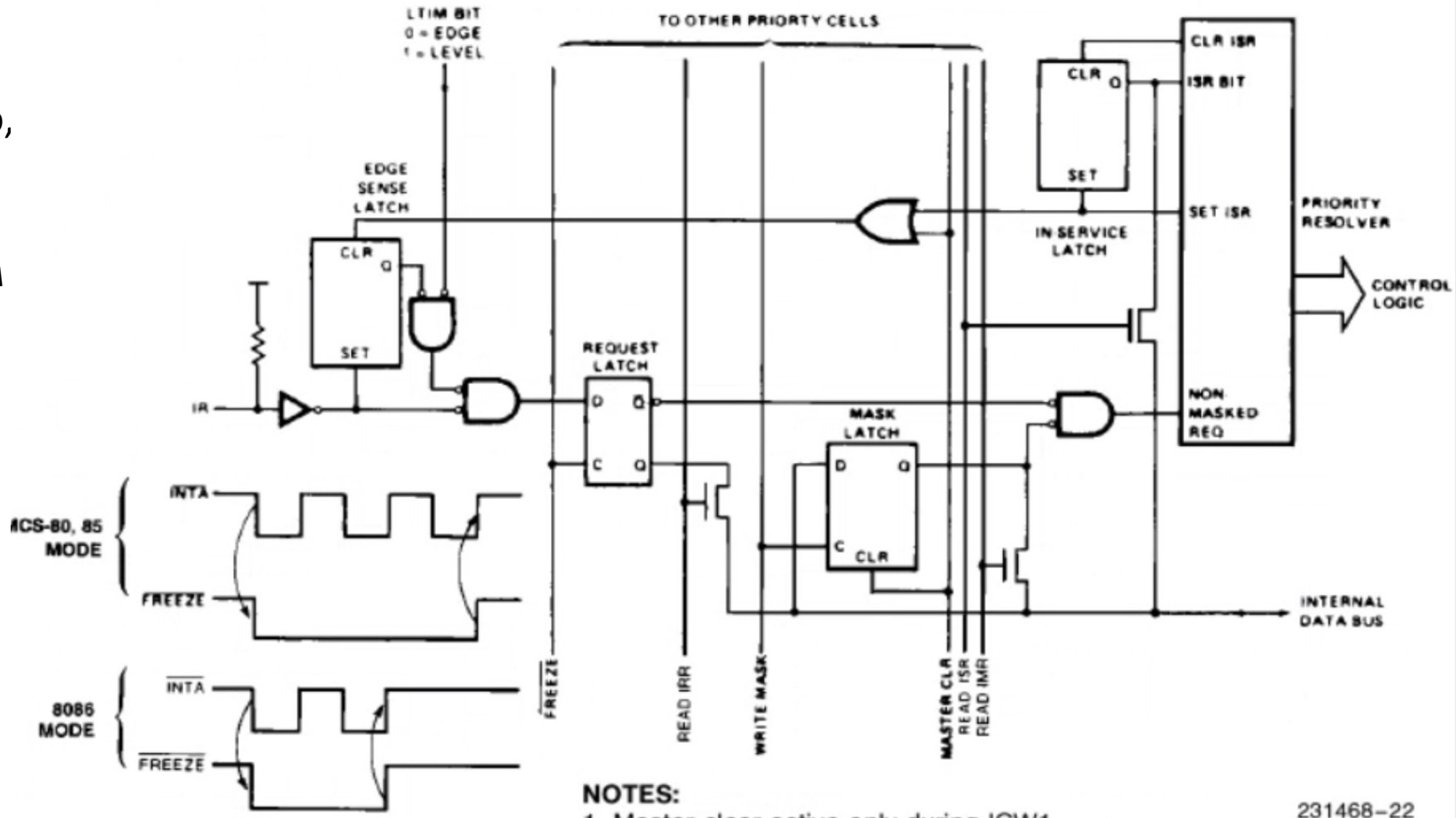


Прерывания


Прерывание от диска,
приоритет 4, ожидание
Завершение работы ISR линии RS232,
получение прерывания от диска
Прерывание RSR232,
приоритет 5
Завершение работы
ISR диска
Прерывание
от принтера,
приоритет 2
Завершение работы
ISR принтера
0
10
15
20
25
35
40
Программа
пользователя
ISR
принтера
ISR линии
RS232
ISR диска
ISR
принтера
Программа
пользователя
Время
Пользователь
Пользователь
Принтер
Пользователь
Принтер
Пользователь
Стек

Контролер прерываний

- Если прерывание маскированно, выходим
- Если есть более приоритетные прерывания в обработке ставим очередь.
- Вызываем прерывание у процессора

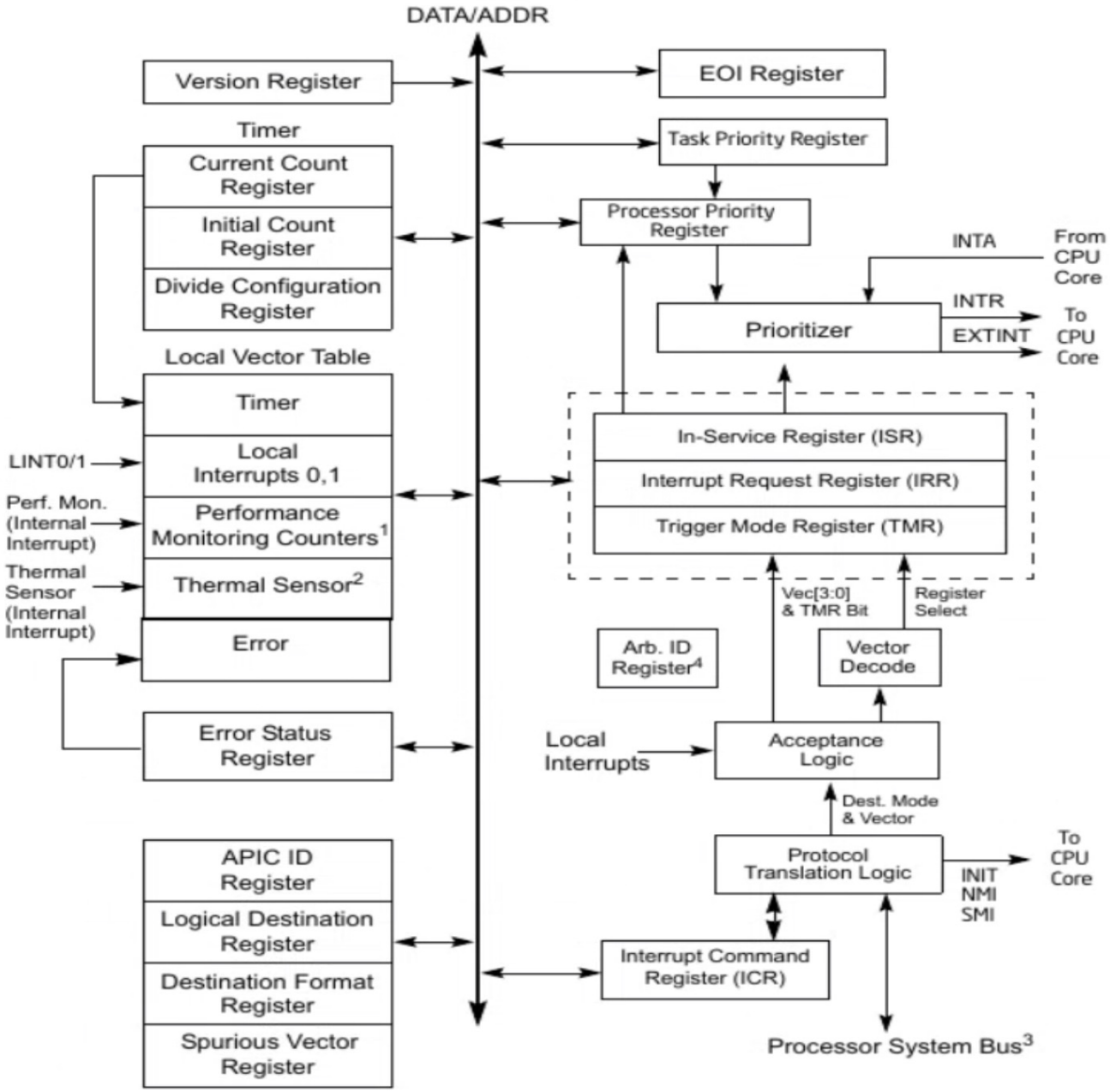


231468–22

NOTES:
1. Master clear active only during ICW1.
2. $\overline{\text{FREEZE}}$ is active during $\overline{\text{INTA}}$ and poll sequences only.
3. Truth Table for a D-Latch.

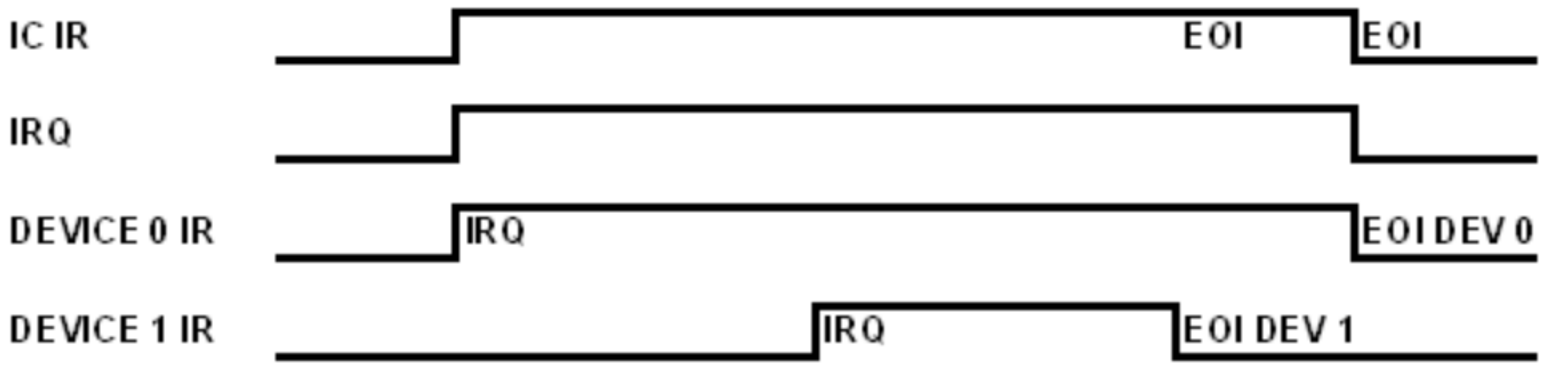
C	D	Q	Operation
1	Di	Di	Follow
0	X	Qn-1	Hold

APIC

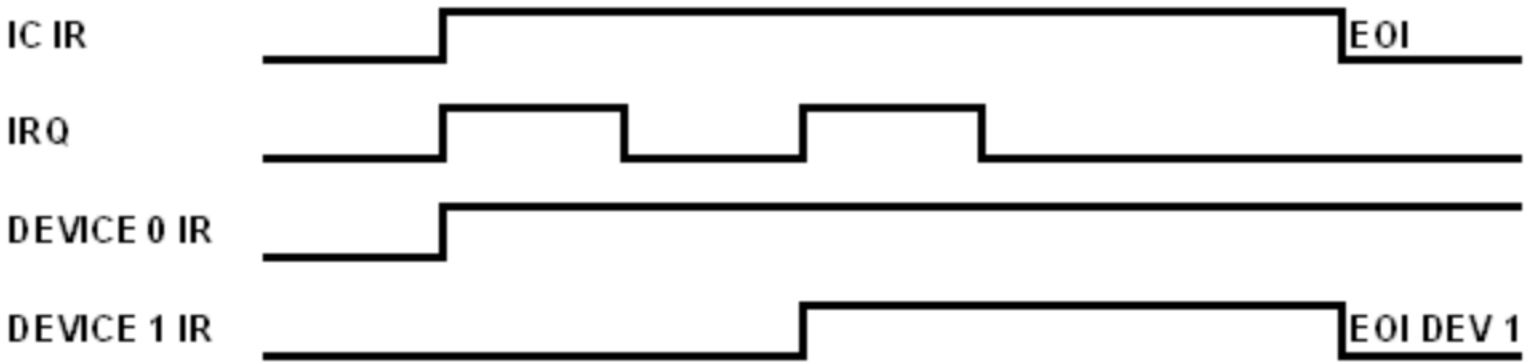


1. Introduced in P6 family processors.
2. Introduced in the Pentium 4 and Intel Xeon processors.
3. Three-wire APIC bus in P6 family and Pentium processors.
4. Not implemented in Pentium 4 and Intel Xeon processors.

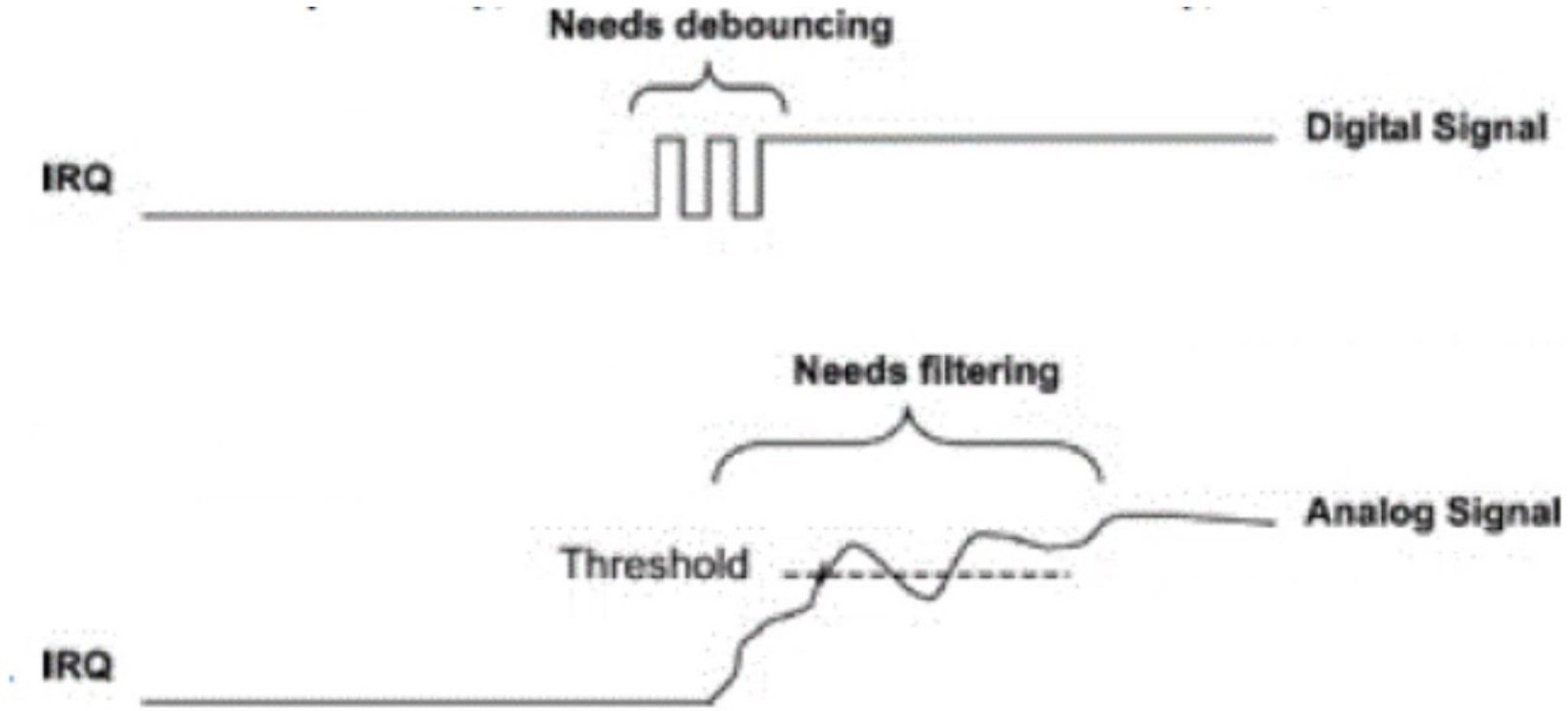
Прерывания по уровню



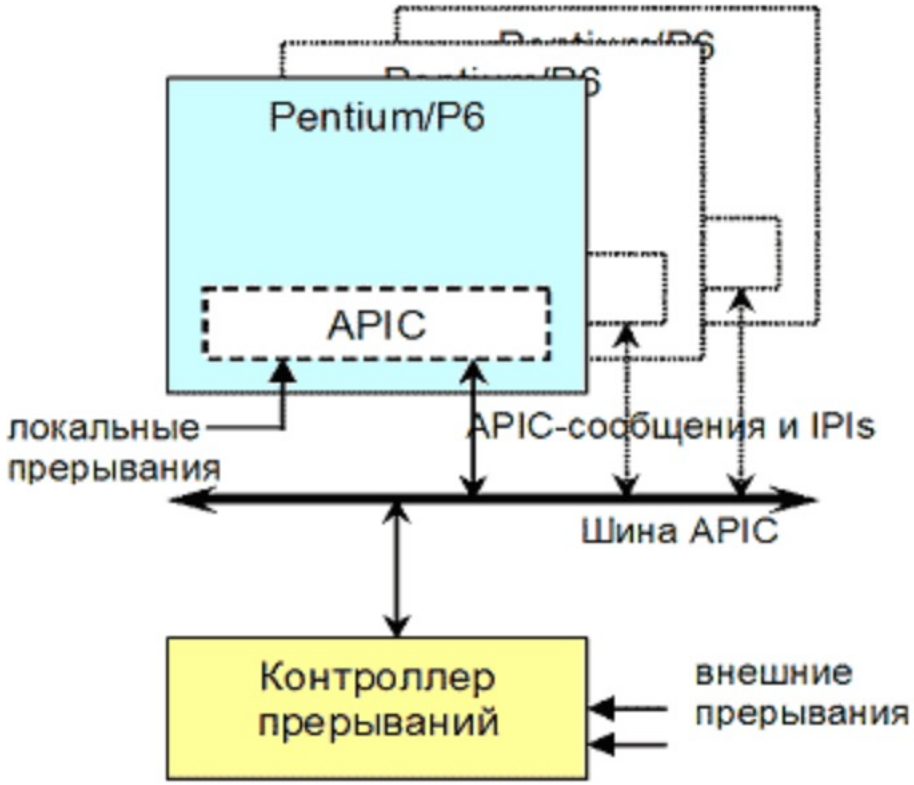
Прерывания по фронту



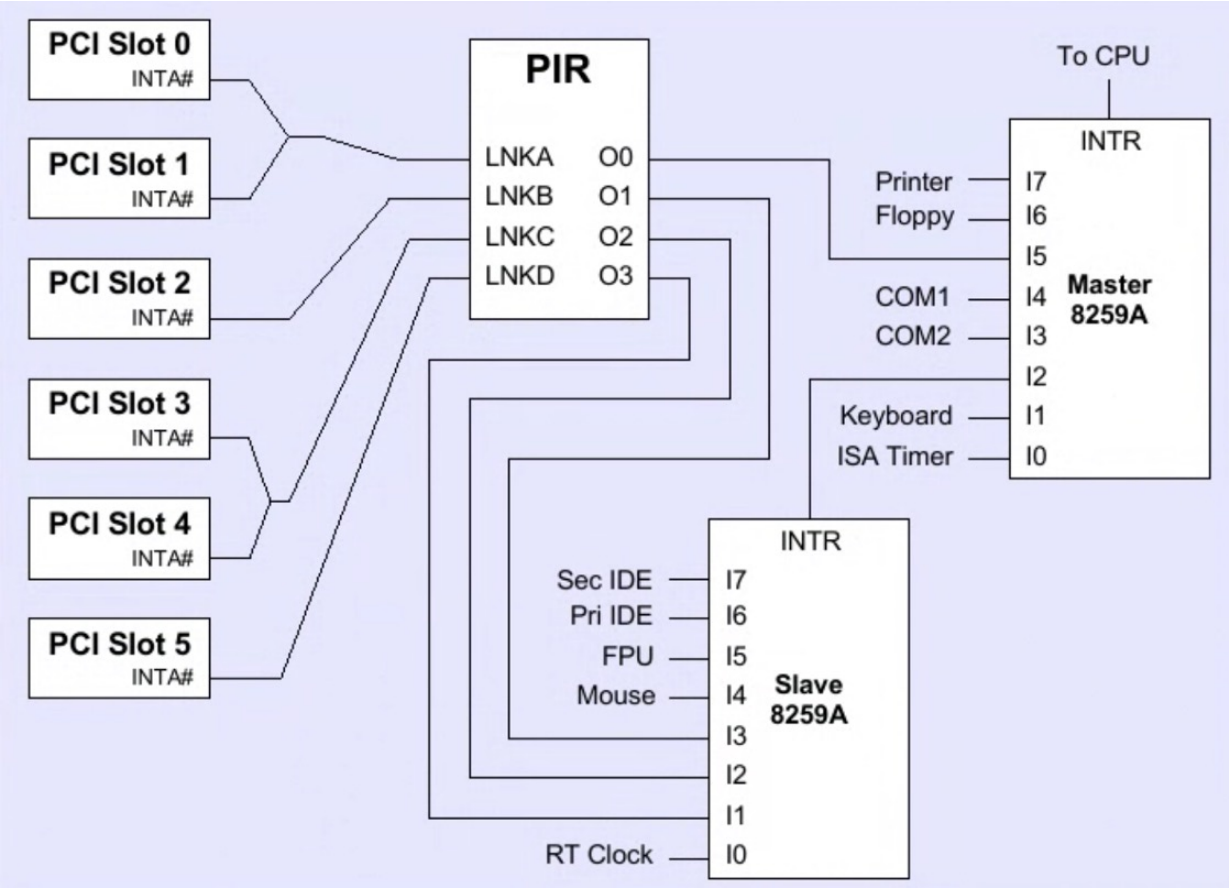
Spurous interrupt



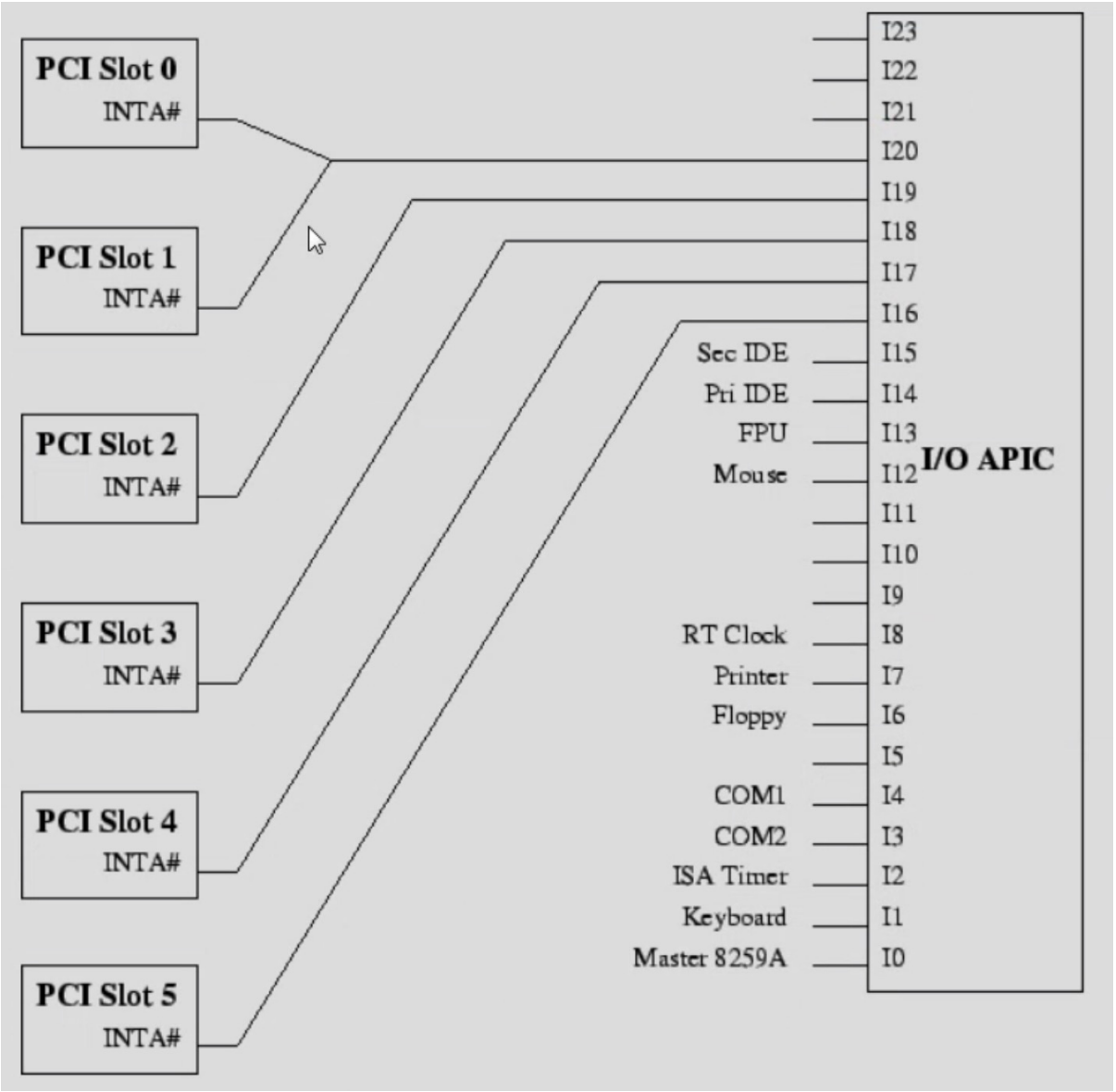
Общая схема обработки прерываний



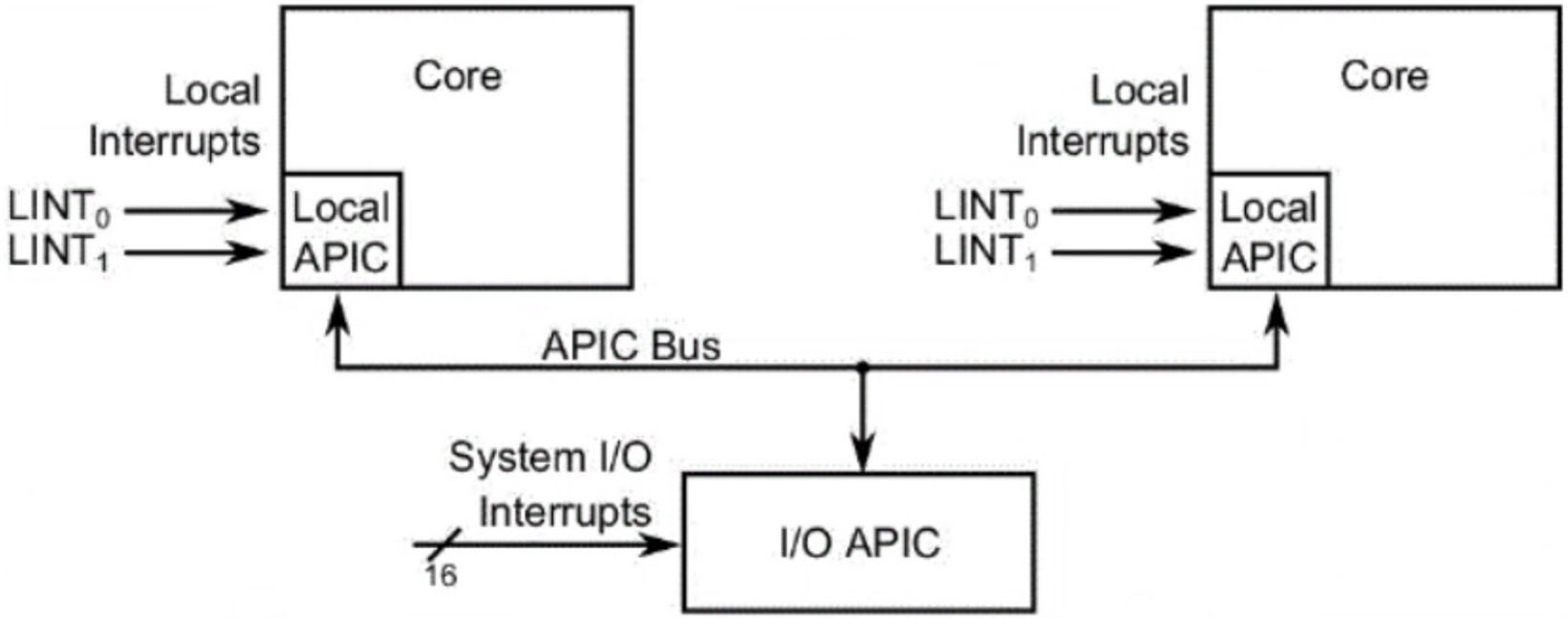
PIR – PCI Interrupt routing



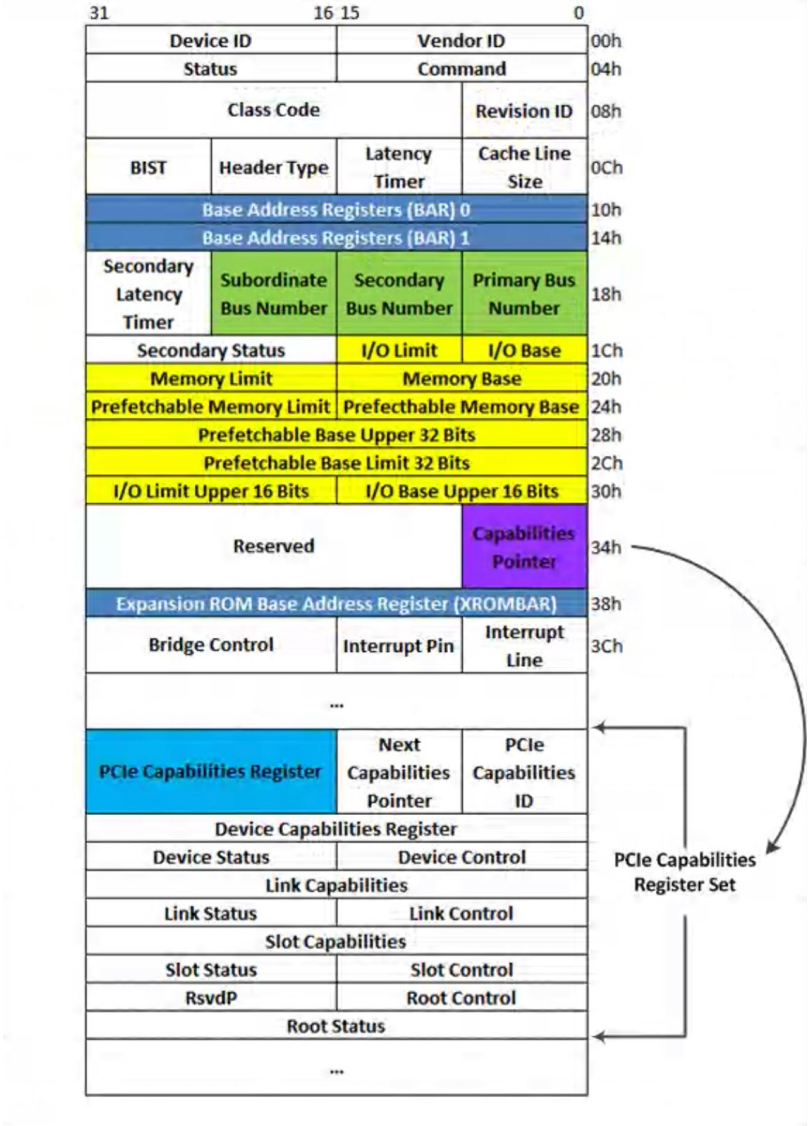
i/o apic


PCI Slot 0
INTA#
PCI Slot 1
INTA#
PCI Slot 2
INTA#
PCI Slot 3
INTA#
PCI Slot 4
INTA#
PCI Slot 5
INTA#
I23
I22
I21
I20
I19
I18
I17
I16
Sec IDE
I15
Pri IDE
I14
FPU
I13
Mouse
I12
I/O APIC
I11
I10
I9
RT Clock
I8
Printer
I7
Floppy
I6
I5
COM1
I4
COM2
I3
ISA Timer
I2
Keyboard
I1
Master 8259A
I0

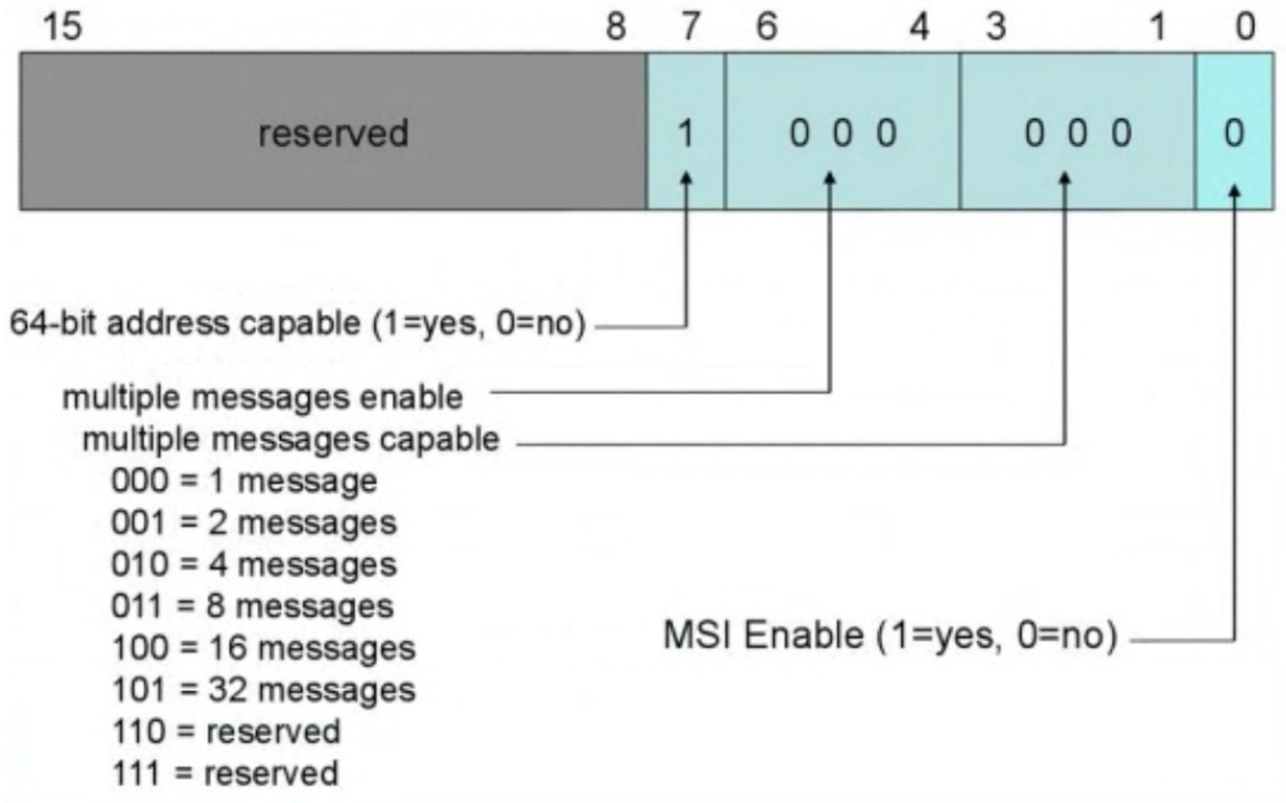
i/o apic


Local
Interrupts
LINT0
LINT1
Core
Local
APIC
Local
Interrupts
LINT0
LINT1
Core
Local
APIC
APIC Bus
System I/O
Interrupts
16
I/O APIC

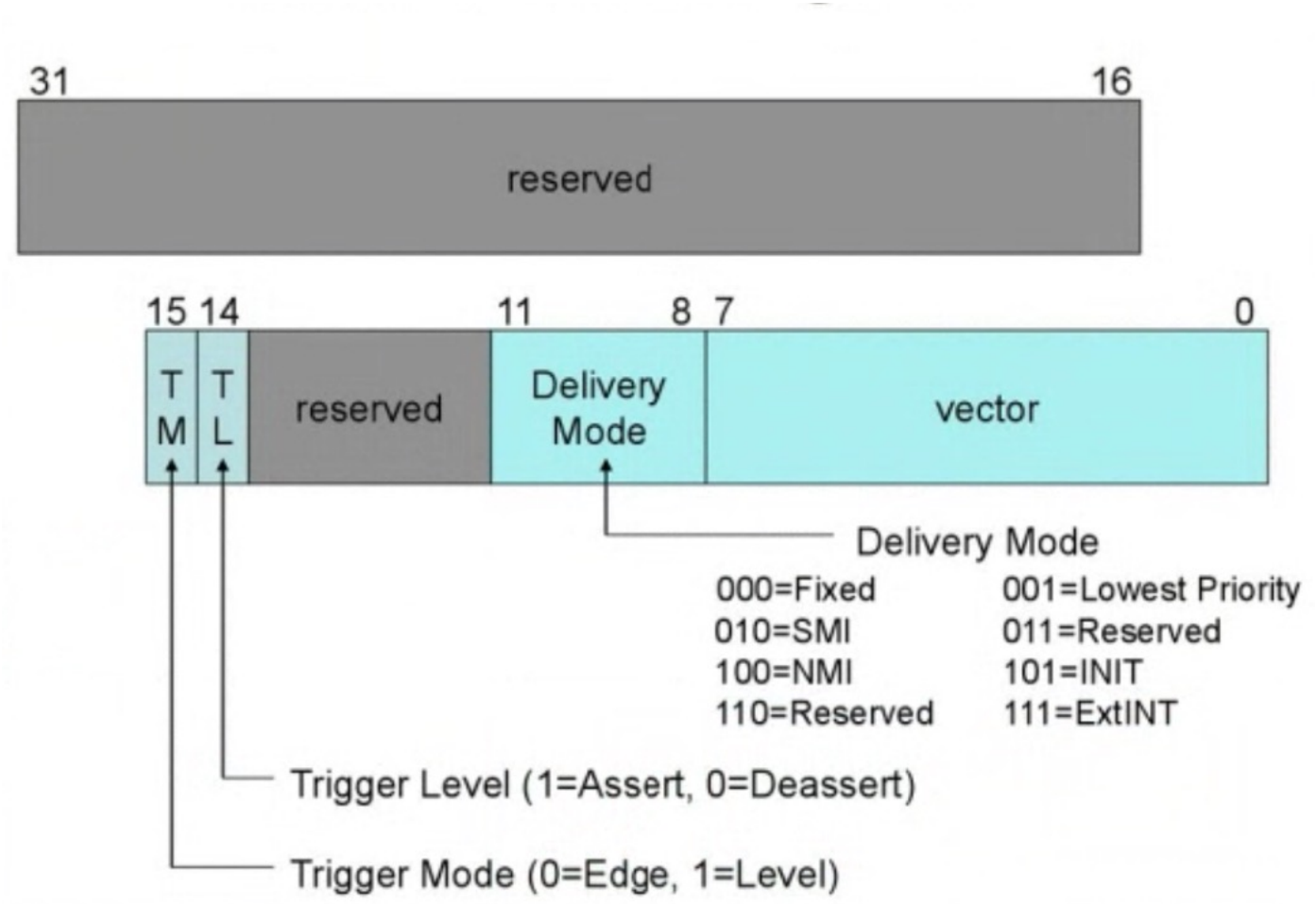
MSI – Message-signaled interrupt



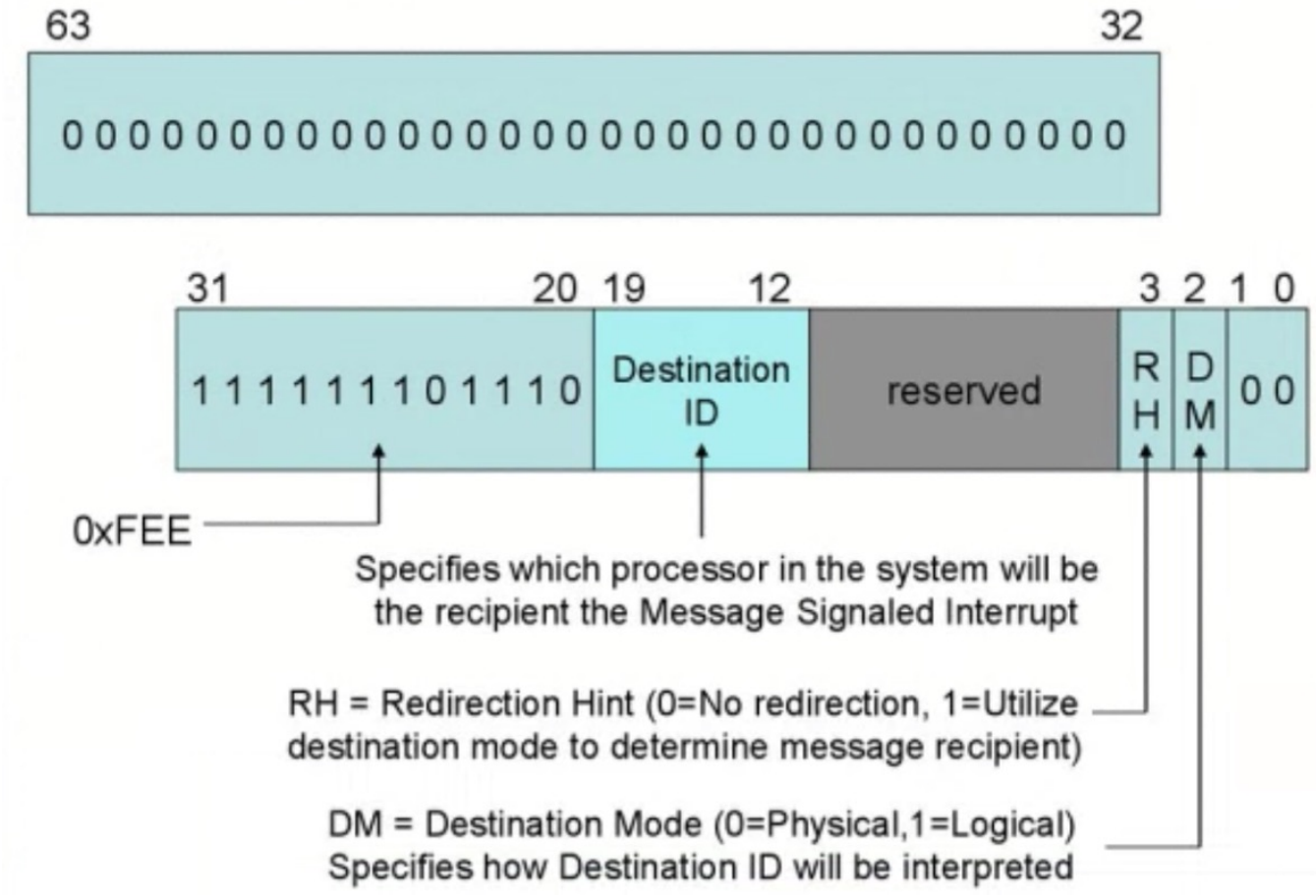
MSI control register



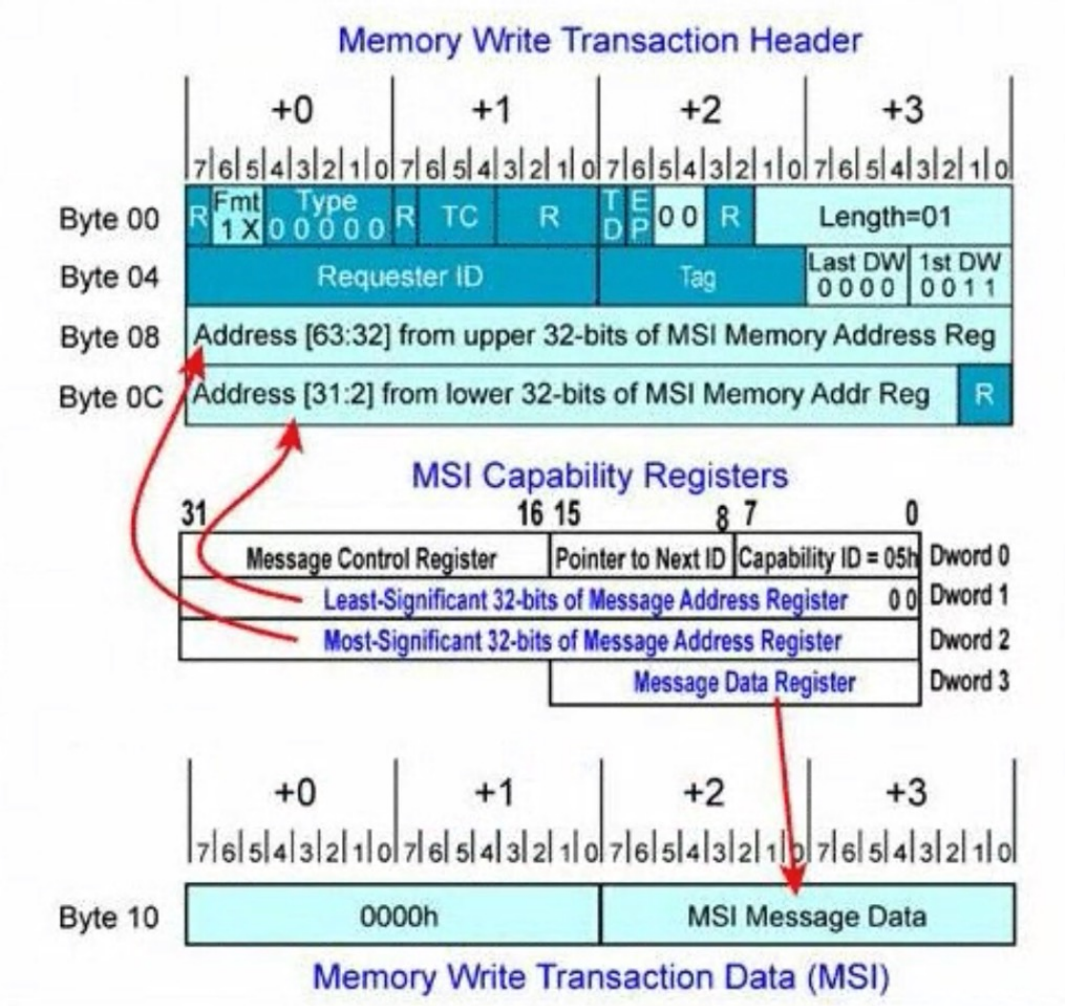
MSI data register



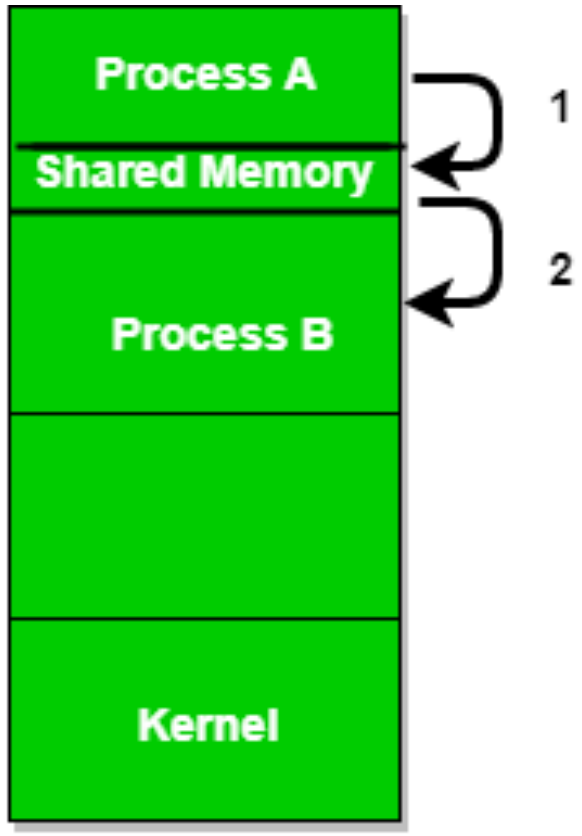
MSI address register

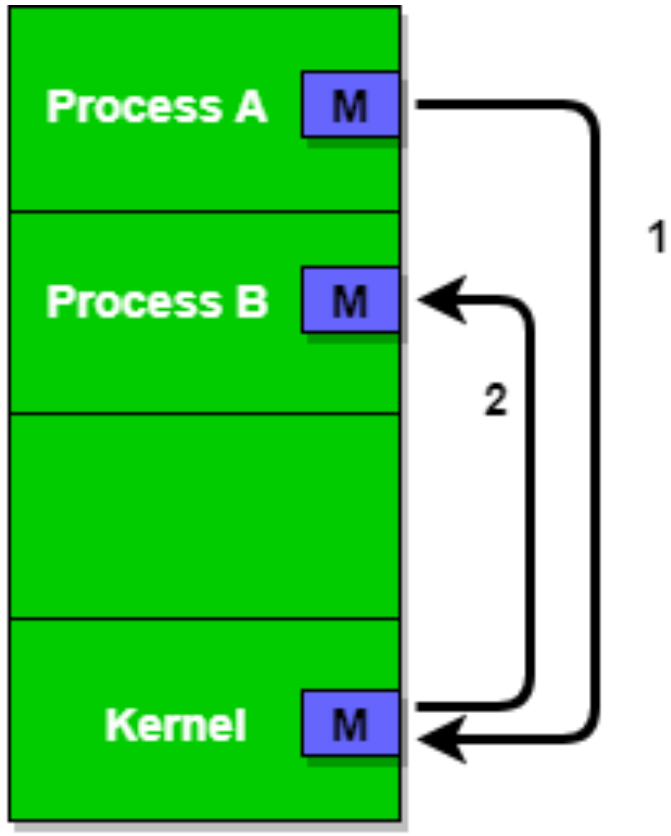


MSI memory transaction

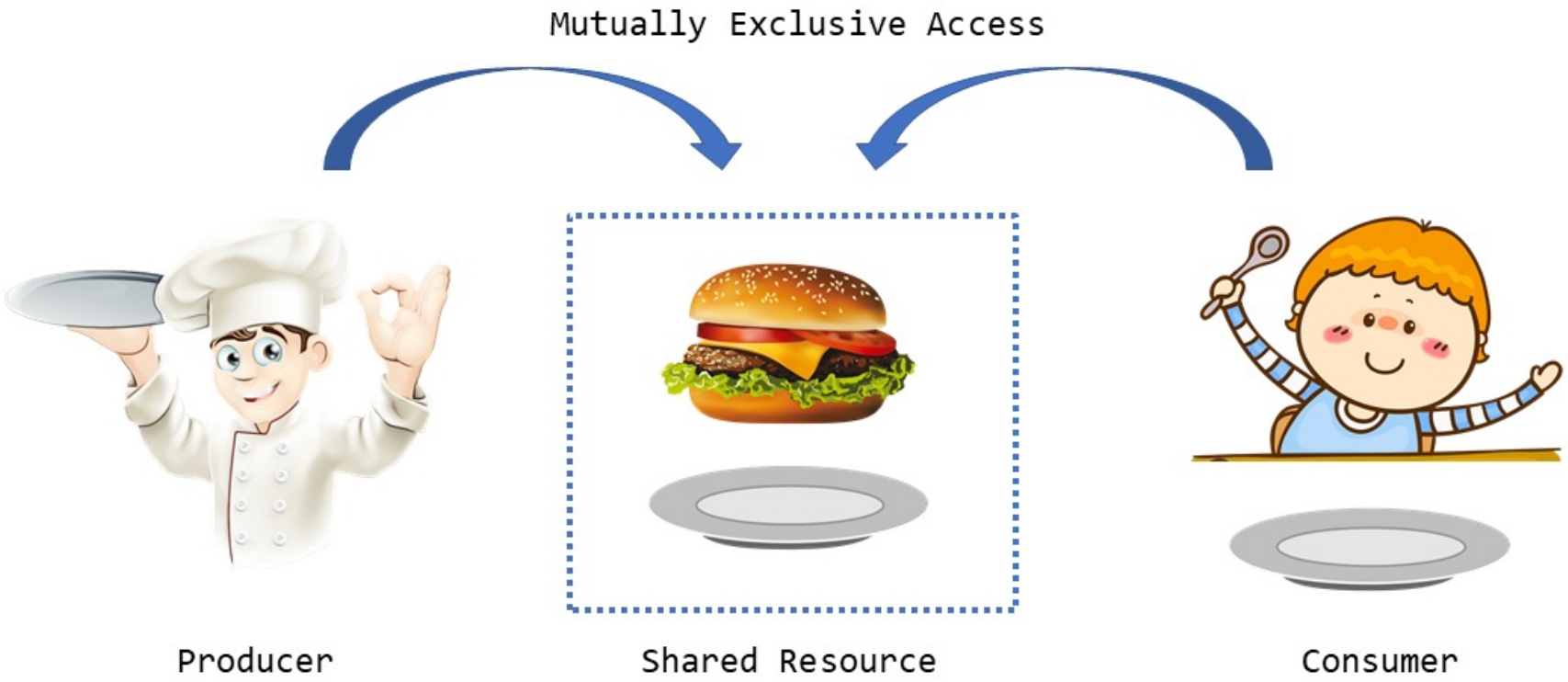


IPC


Process A
Shared Memory
Process B
Kernel
1
2


Process A
M
Process B
M
Kernel
M
1
2

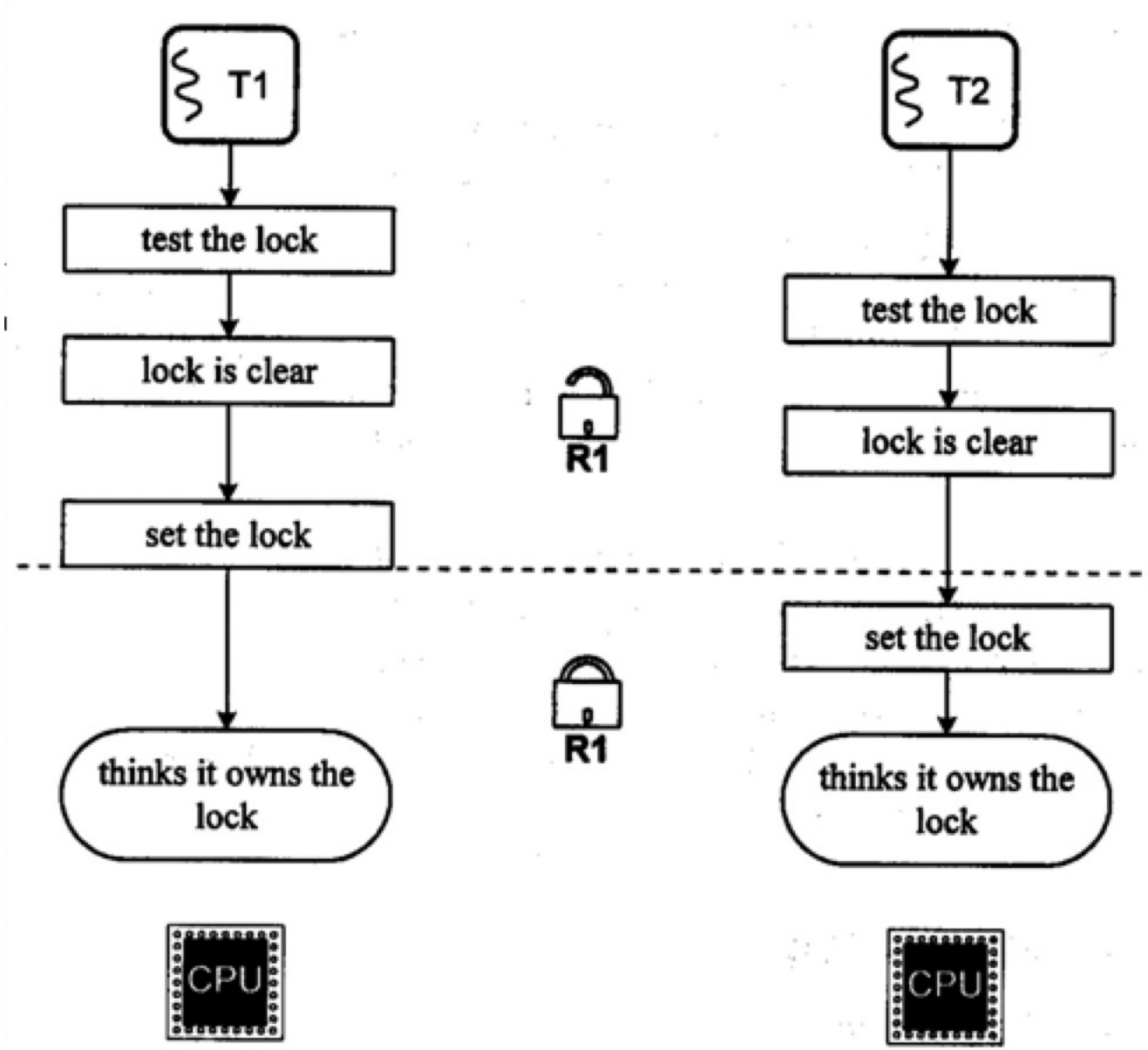
Producer-Consumer problem



spin/wait

spin mutex

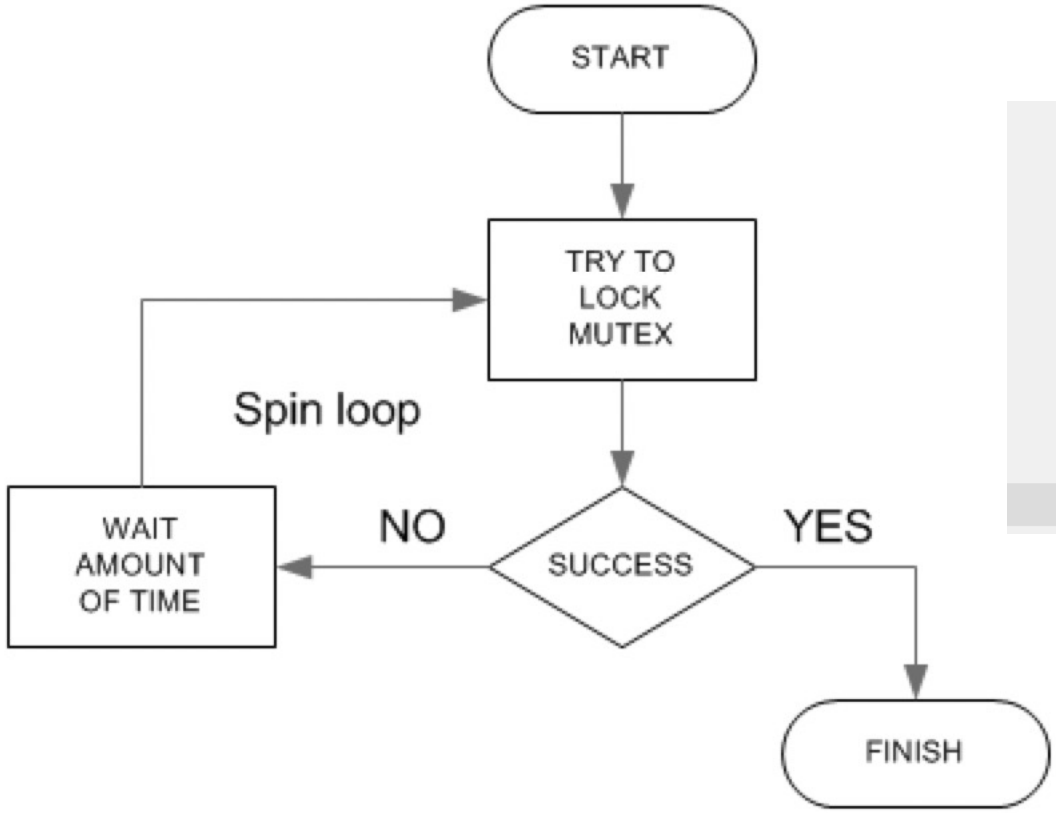
- init
- trylock
- lock
- unlock



spin mutex

```
1  ;-------------------------------------------------------------------
2  mutex_init(&mutex, MUTEX_UNLOCKED)      mov [mutex], 0
3  mutex_init(&mutex, MUTEX_LOCKED)        mov [mutex], 1
4  ;-------------------------------------------------------------------
5  bool    mutex_trylock(&mutex)
6                          mov eax, 1
7                          lock xchg   [mutex],eax
8                          or  eax,eax
9                          mov eax, FALSE
10                         mov ebx, TRUE
11                         cmovz   eax, ebx
12                         ret
13 ;-------------------------------------------------------------------
14 mutex_unlock(&mutex)              lock mov    [mutex],0
```

spin lock



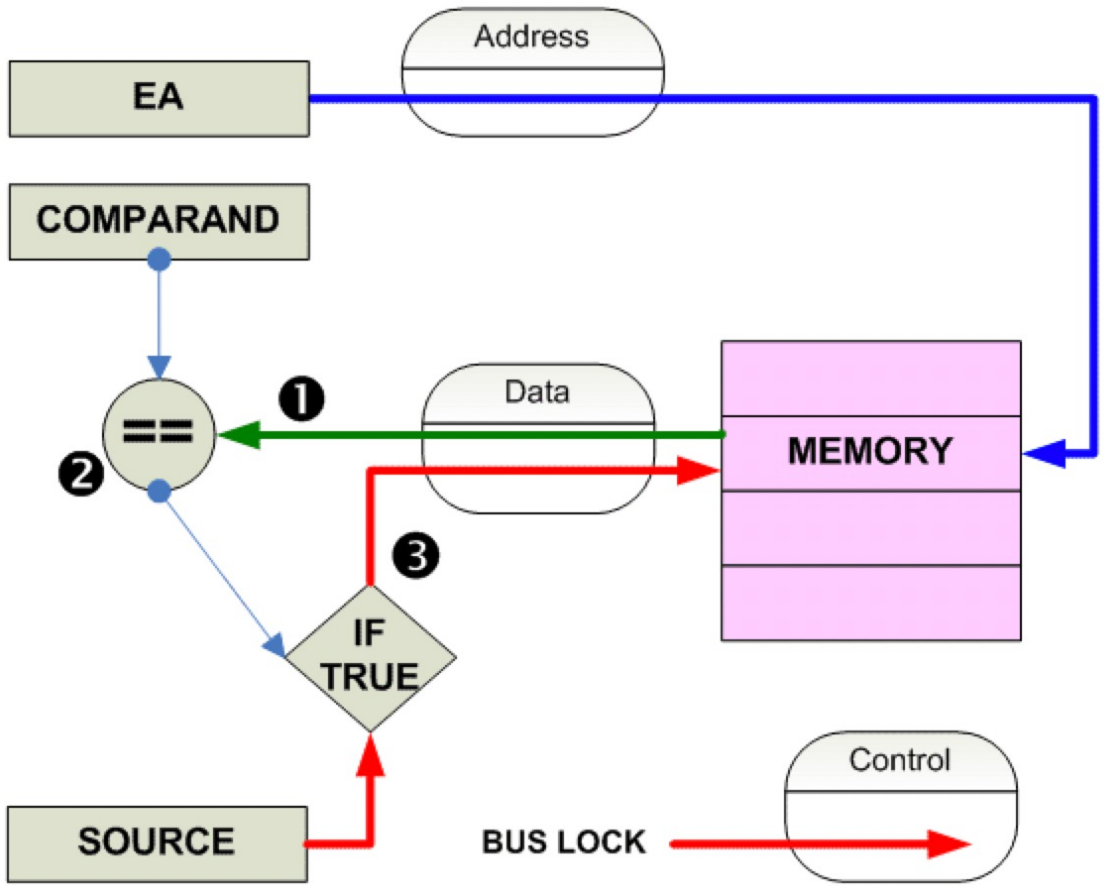

```
1   spinmutex_lock(&mutex)
2                           ?spin_loop:
3                           mov eax,1
4                           lock  xchg  [mutex],eax
5                           or  eax,eax
6                           jz  ?lock_success
7                           ;; wait
8                           jmp ?spin_loop
9                           ?lock_success:
10                          ret
```

spin lock

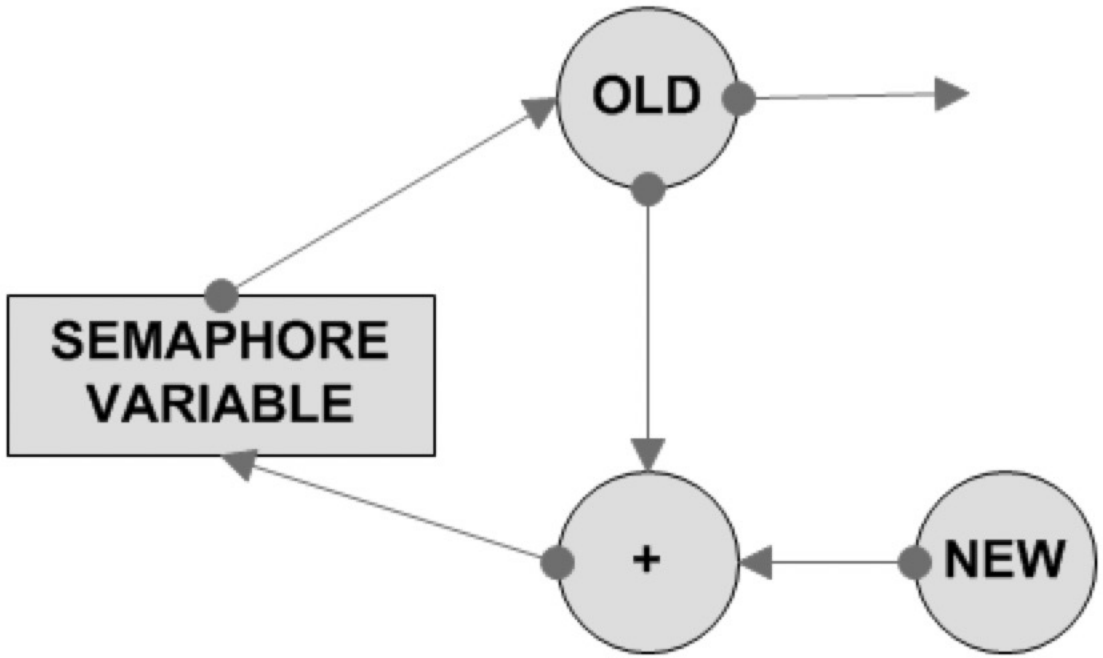
```
1  spinmutex_lock(&mutex)
2                          ?spin_loop:
3                          mov eax,1
4                          lock  xchg  [mutex],eax
5                          or  eax,eax
6                          jz  ?lock_success
7                          ;; wait
8                          jmp ?spin_loop
9                          ?lock_success:
10                         ret
```

```
1  spinmutex_lock(&mutex)
2                          ?spin_loop:
3                              mov eax,1
4                             lock xchg    [mutex],eax
5                              or  eax,eax
6                              jz  ?lock_success
7                              xor ecx,ecx
8                              xor edx,edx
9                              lea eax,[mutex]
10                             monitor
11                             mwait
12                             jmp ?spin_loop
13                         ?lock_success:
14                             ret
```

Atomics


Address
EA
COMPARAND
Data
MEMORY
==
1
2
3
IF
TRUE
Control
SOURCE
BUS LOCK

semaphore



semaphore

```
1  ------------------------------------------------------------------------
2  semaphore_init(&semaphore)          mov [semaphore], 0
3  ------------------------------------------------------------------------
4  bool    semaphore_trylock(&semaphore)
5                          mov eax, 1
6                          lock    xadd    [semaphore], eax
7                              cmp eax, MAX_VALUE
8                              mov eax, TRUE
9                              jnc ?success
10                             mov eax, -1
11                         lock    xadd    [semaphore],eax
12                             mov eax, FALSE
13                         ?success:
14                             ret
15 ------------------------------------------------------------------------
16 semaphore_unlock(&semaphore)        lock    dec [semaphore]
17 ------------------------------------------------------------------------
```

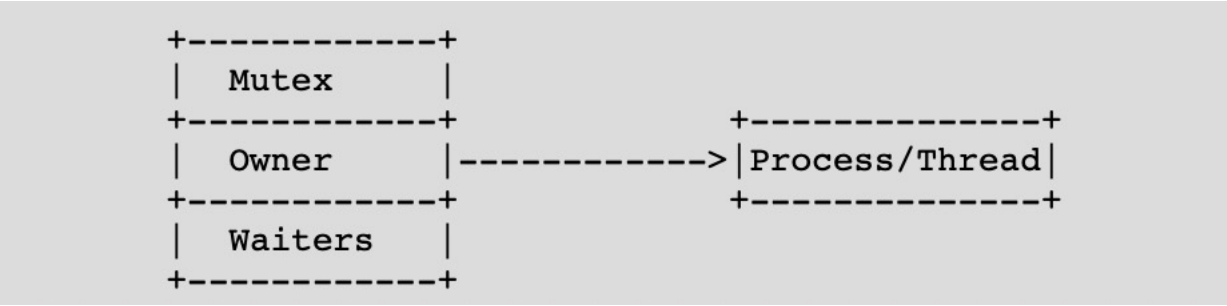
semaphore

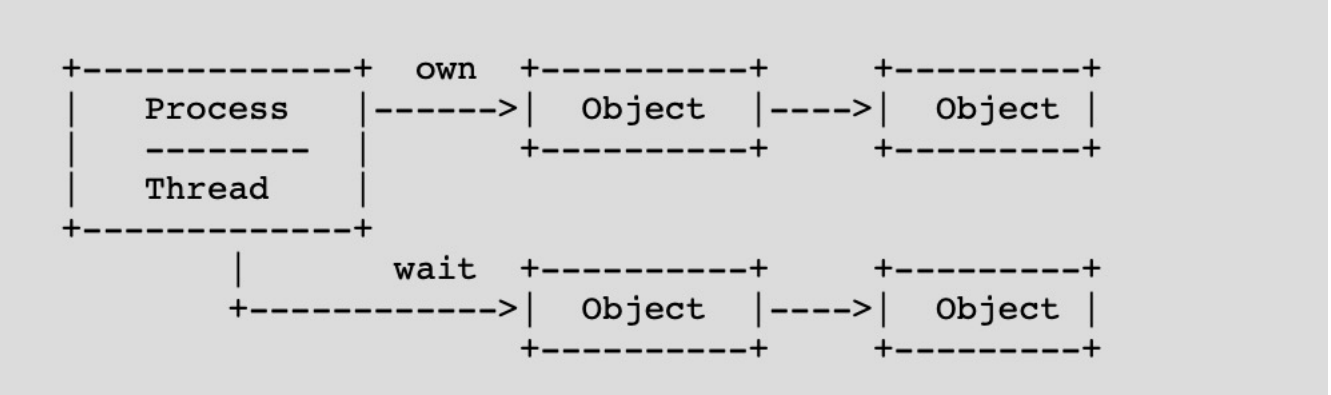
```
1  ----------------------------------------------------------------------
2  bool    semaphore_trylock(&semaphore)       spinmutex_lock(&mutex)
3                          if (*semaphore < MAX_VALUE)
4                          {
5                              (*semaphore)++;
6                              mutex_unlock(&mutex);
7                              return  TRUE;
8                          }
9                          mutex_unlock(&mutex);
10                         return  FALSE;
11 ----------------------------------------------------------------------
12 semaphore_unlock(&semaphore)        spinmutex_lock(&mutex)
13                         (*semaphore)--;
14                         mutex_unlock(&mutex)
15 ----------------------------------------------------------------------
16 spinsemaphore_lock(&semaphore)      while(semaphore_trylock(&semaphore) != TRUE) ;
17 ----------------------------------------------------------------------
```

1 proc

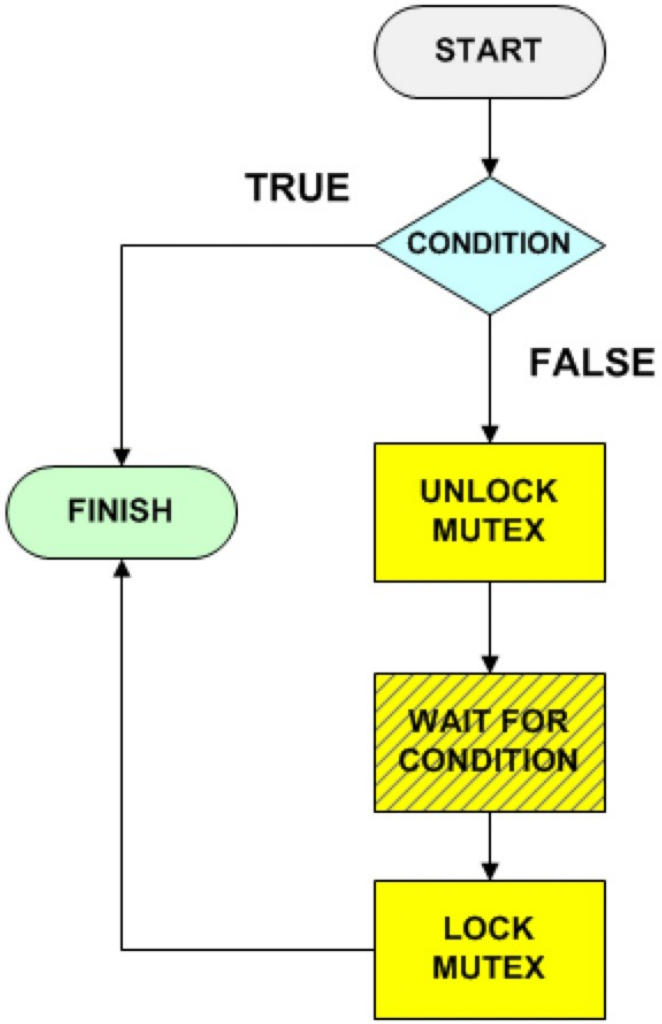
```
1  ------------------------------------------------------------------------
2      bool    sp_mutex_trylock(&mutex)
3                              cli
4                              mov eax,[mutex]
5                              mov [mutex], 1
6                              sti
7                              ret
8  ------------------------------------------------------------------------
9      bool    sp_semaphore_trylock(&mutex)
10                             cli
11                             mov eax, [sem]
12                             add eax, 1
13                             mov [sem], eax
14                             cmp eax, MAX_VALUE
15                             jnc ?success
16                             sub eax, 1
17                             mov [sem], eax
18                             mov eax, FALSE
19                             sti
20                             ret
21                         ?success:
22                             mov eax, TRUE
23                             sti
24                             ret
25 ------------------------------------------------------------------------
```

waitable objects

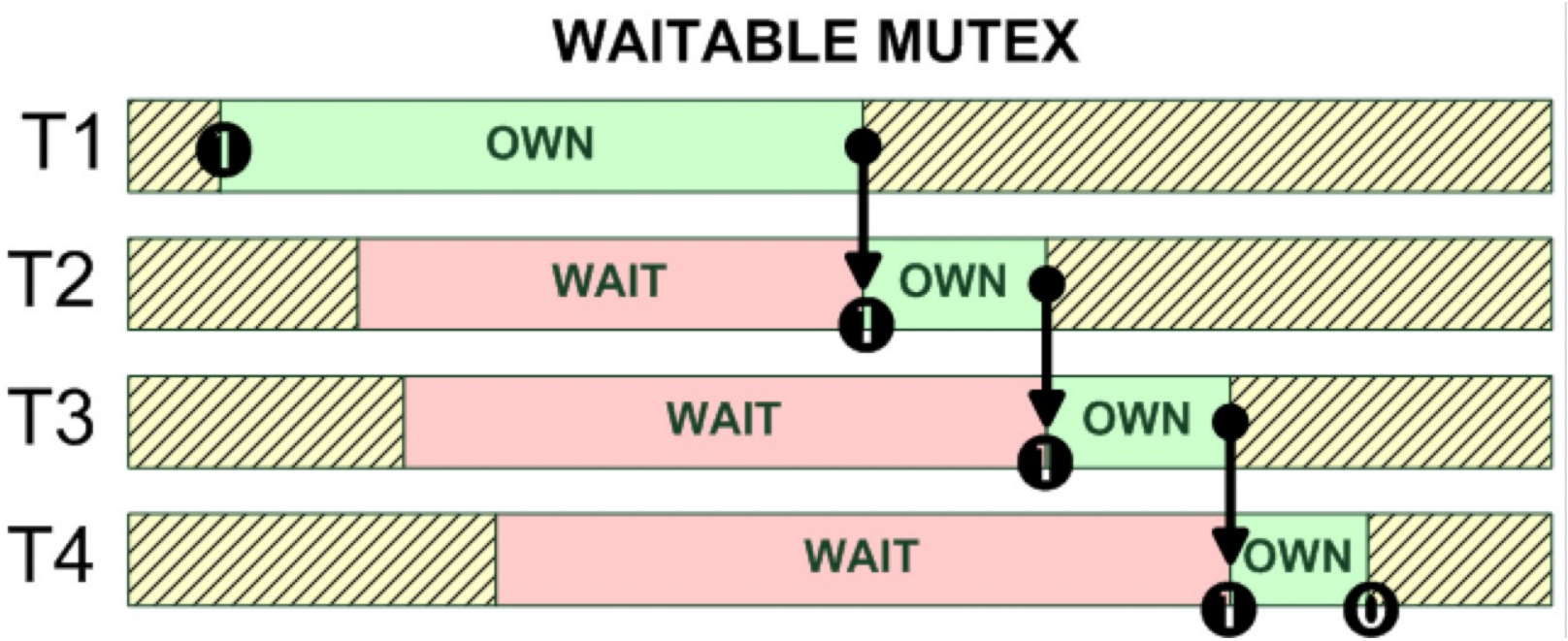




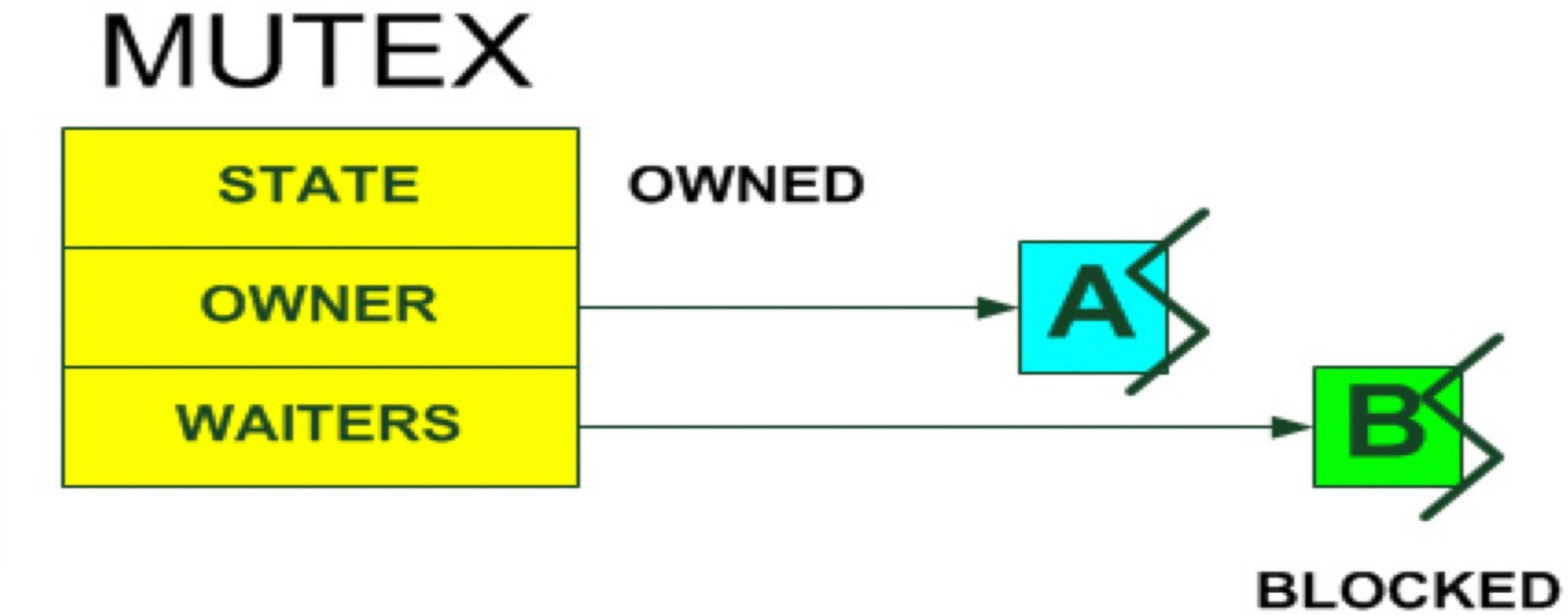
waitable condvar



waitable mutex



waitable mutex



APC (APPLICATION SPECIFIC CALLBACKS)

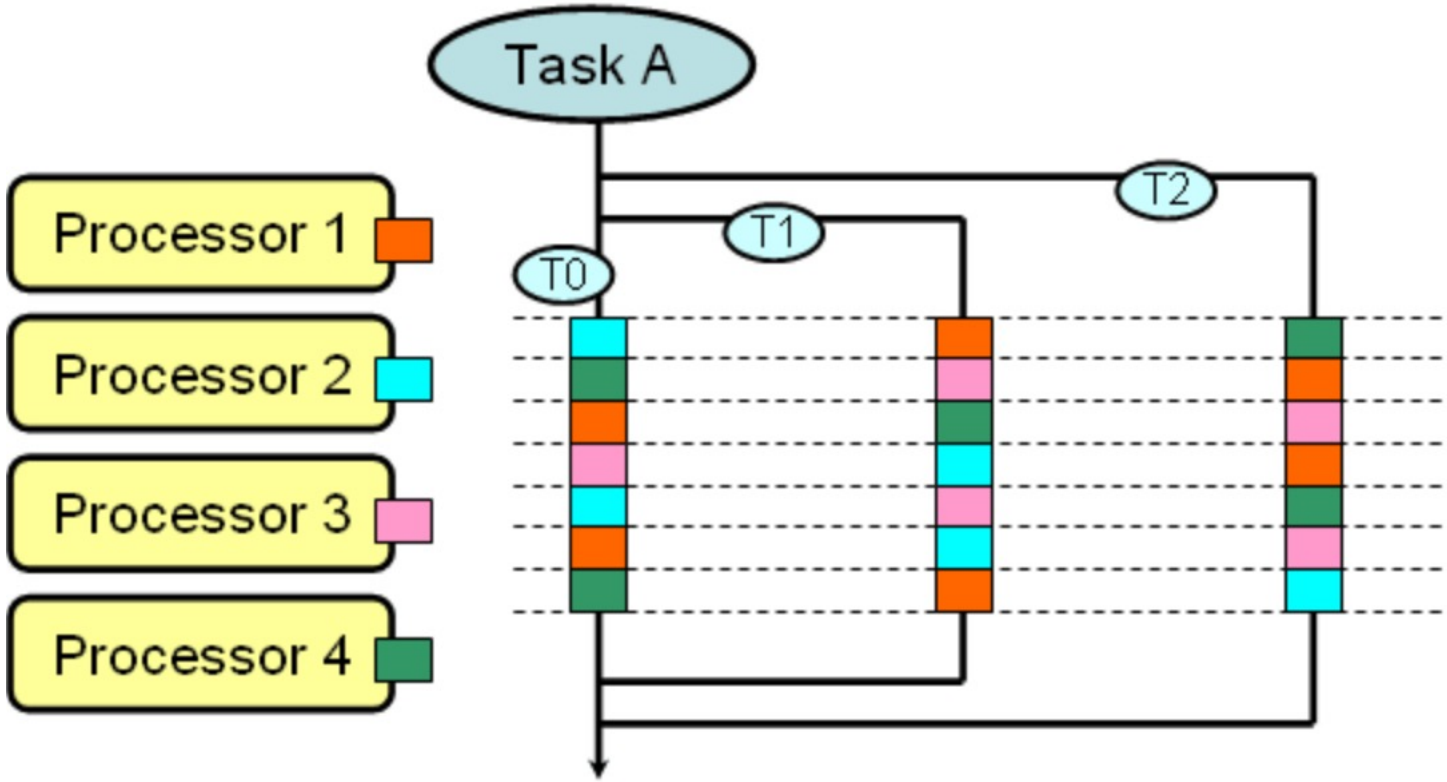
```
// POSIX

result = sigaction(int signalnum,
          const struct sigaction* action,
          struct sigaction* prevaction);


    struct sigaction
    {
        void (*sa_handler)();
        void (*sa_sigaction)(int, siginfo_t*, void*);
        sigset_t  sa_mask;
        int   sa_flags;
              SA_RESETHAND
              SA_NODEFER
              SA_RESTART
              SA_SIGINFO

    };
```

```
// signal.h
void* signal(int signum, void* handler);
        SIG_IGN         // ignore
        SIG_DFL     // default action

int   raise(int signum);

int   raise(int signum)
{
    return kill(getpid(), signum);
}

char*   strsignal(int signum)


union sigval
{
    int sival_int;
    void*   sival_ptr;
};

void* sigqueue(pid_t pid, int signum, const union sigval value);
```

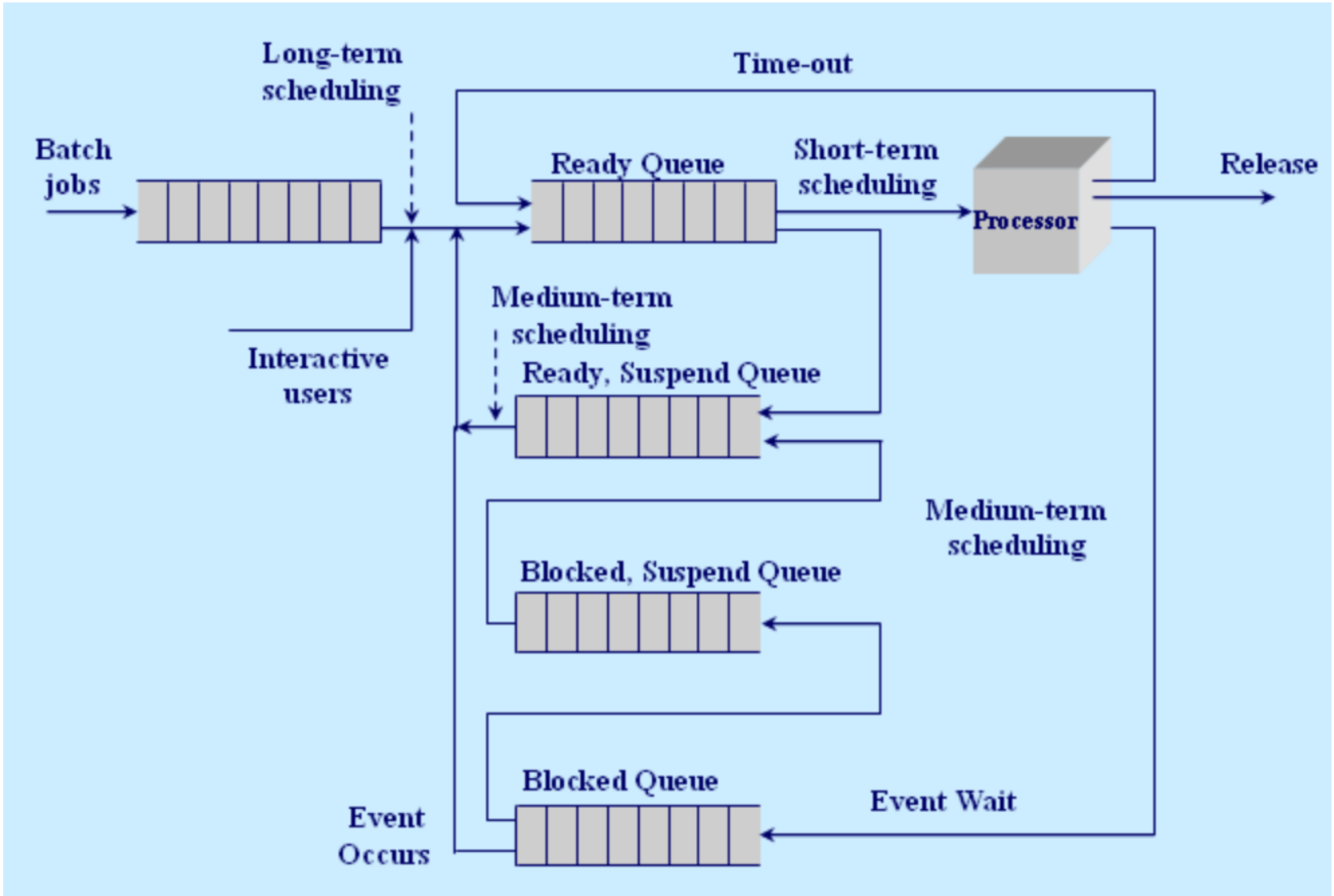
APC (APPLICATION SPECIFIC CALLBACKS)

```
1  NTSTATUS  NTAPI NtQueueApcThread(IN HANDLE ThreadHandle,
2              IN PUSER_APC_ROUTINE  ApcRoutine,
3              IN PVOID FirstArg,  // QueryUserAPC() APC address
4              IN PVOID SecondArg, // QueryUserAPC() APC argument
5              IN PVOID ThirdArg);
6
7
8  VOID NTAPI _NativeUserRoutine(PVOID First Arg, PVOID SecondArg, PVOID ThirdArg)
9  {
10     ((PAPCFUNC) FirstArg, ( (DWORD) SecondArg);
11 }
12
13 NTSTATUS NtAlertThread(HANDLE ThreadHandle);
14 NTSTATUS NtTestAlert(VOID);
```

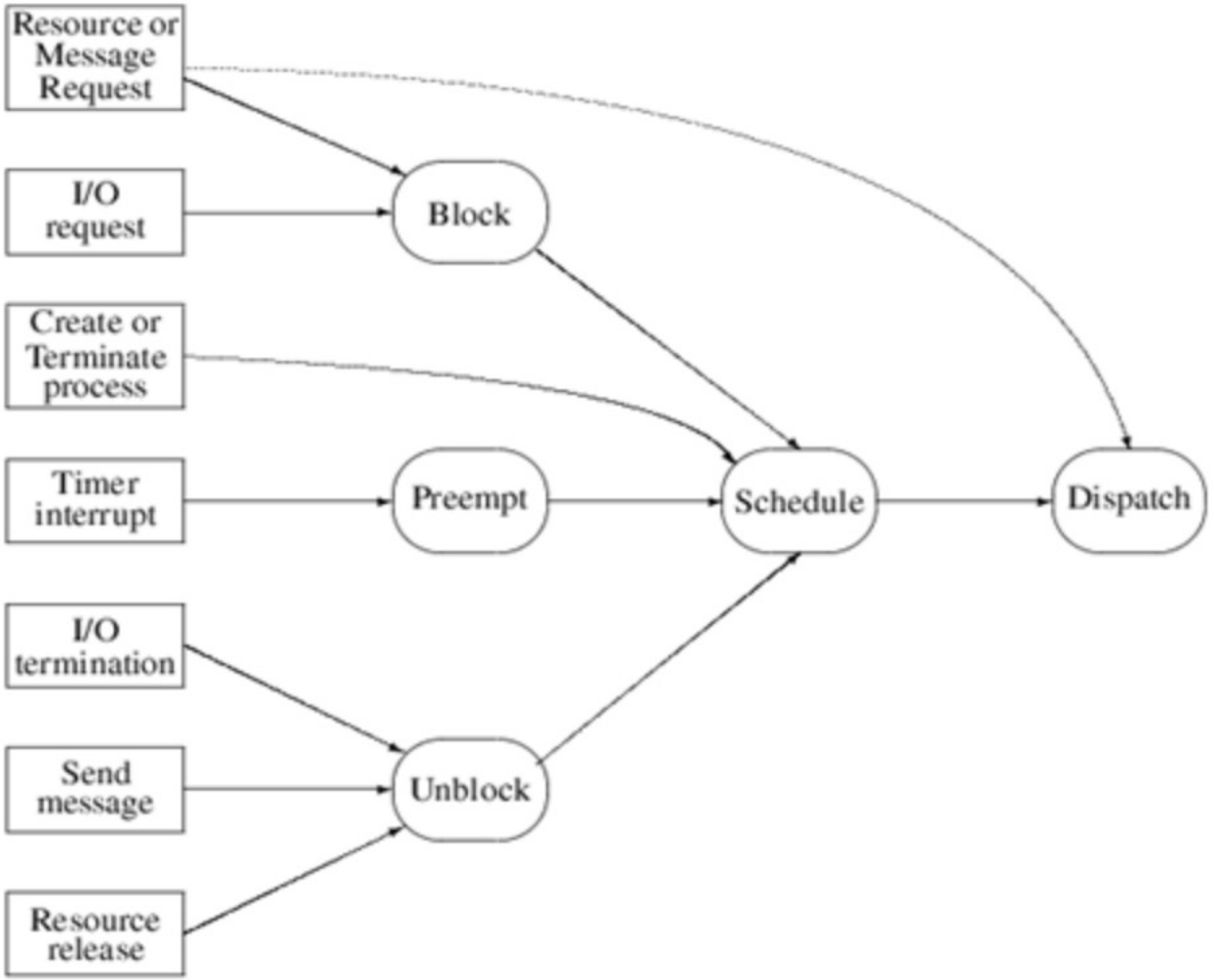
Планировка задач



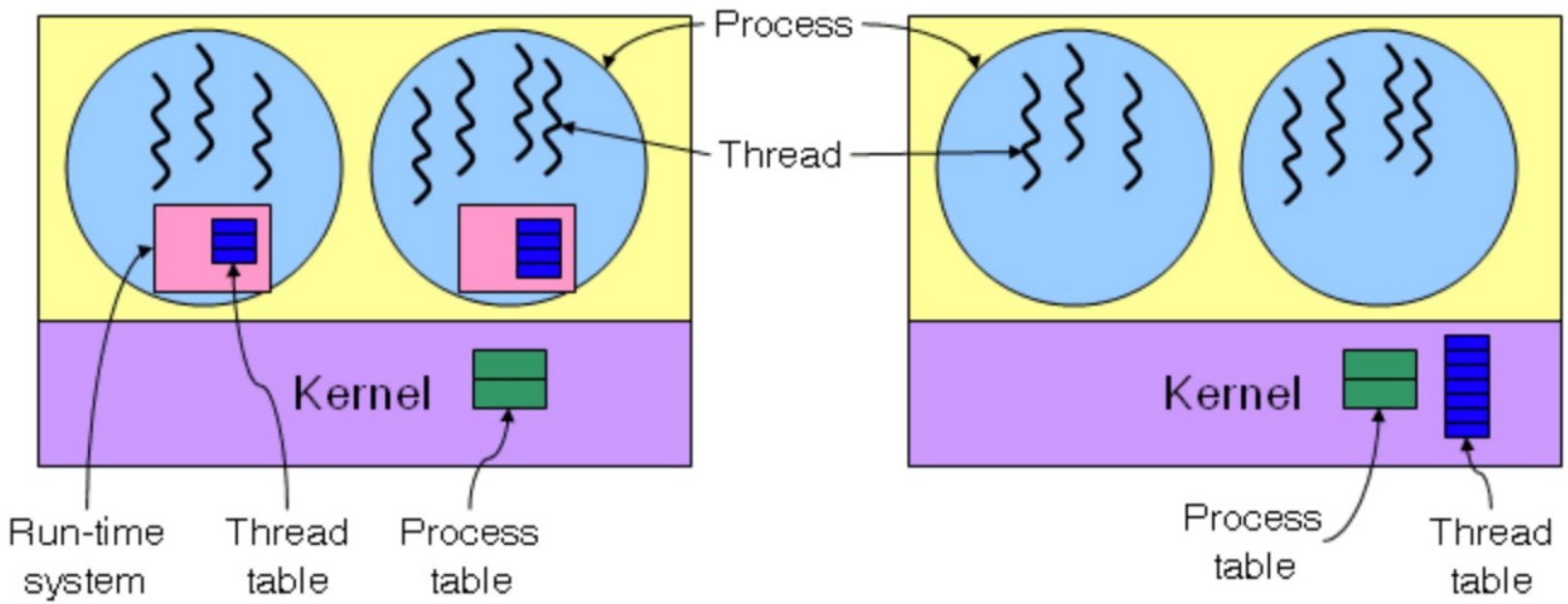
Планировщик



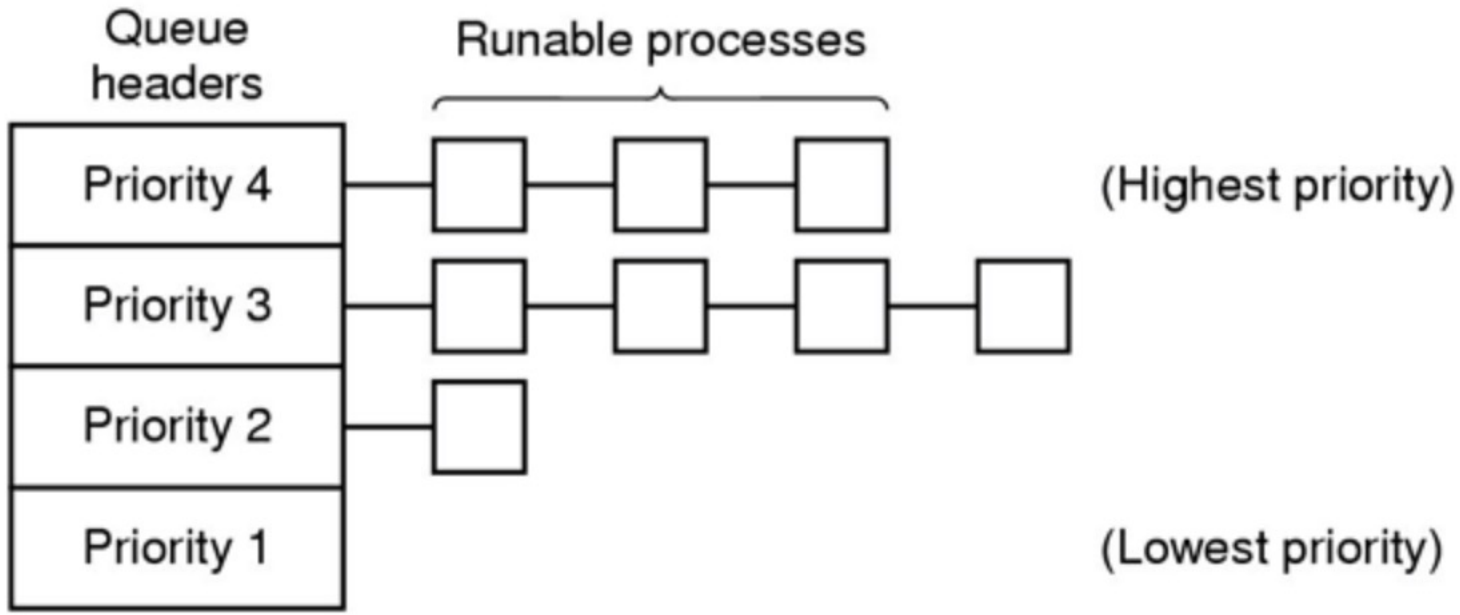
Перепланировка



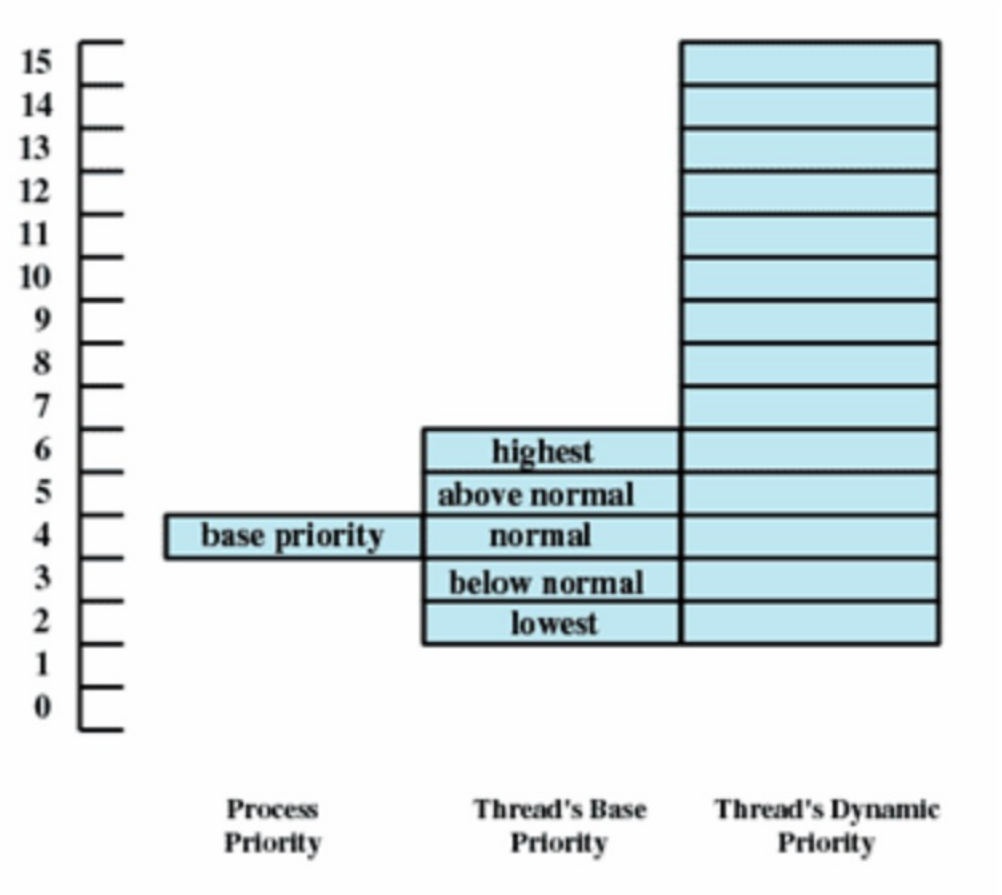
Стратегии планирования



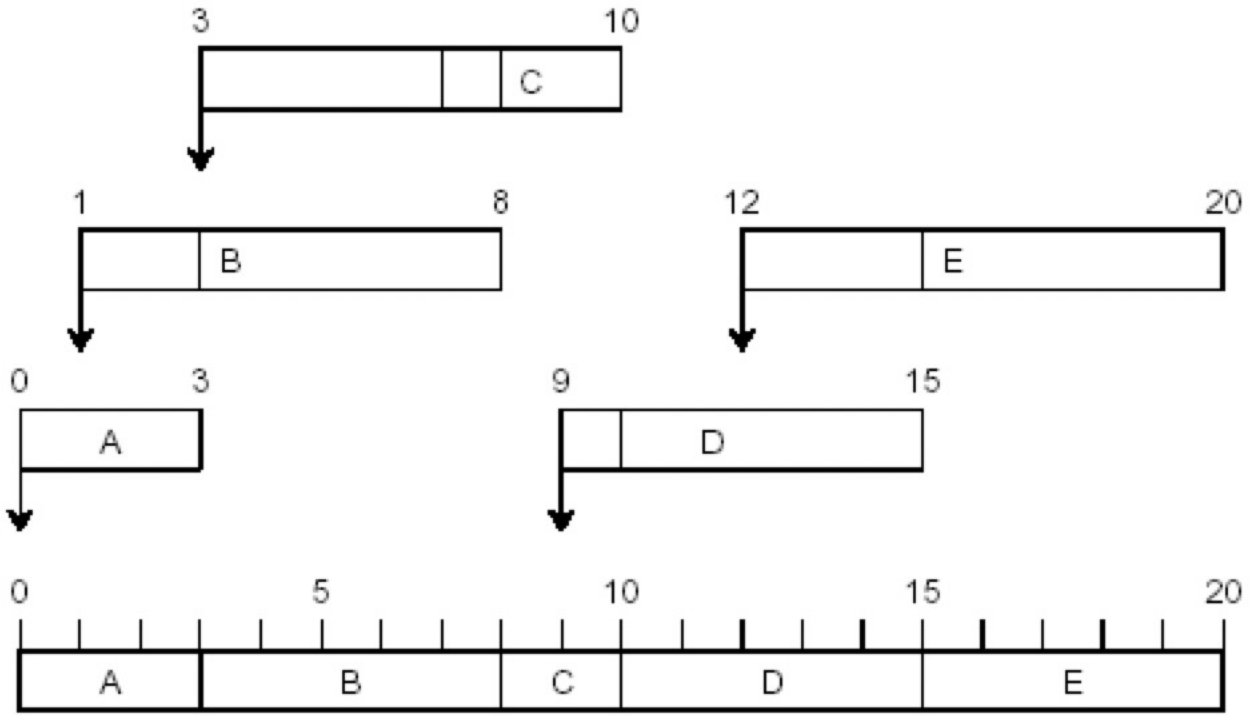
Планировка с приоритетами



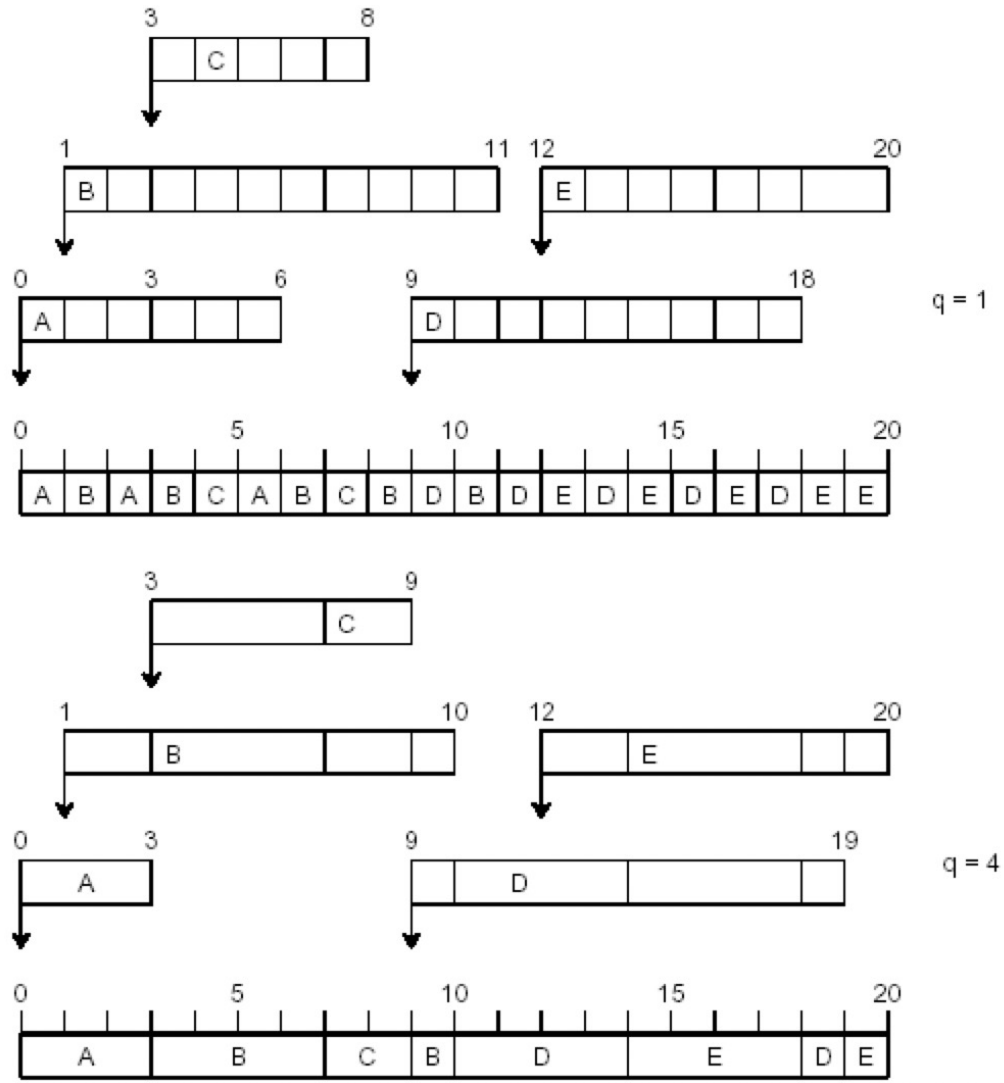
Планировка с приоритетами



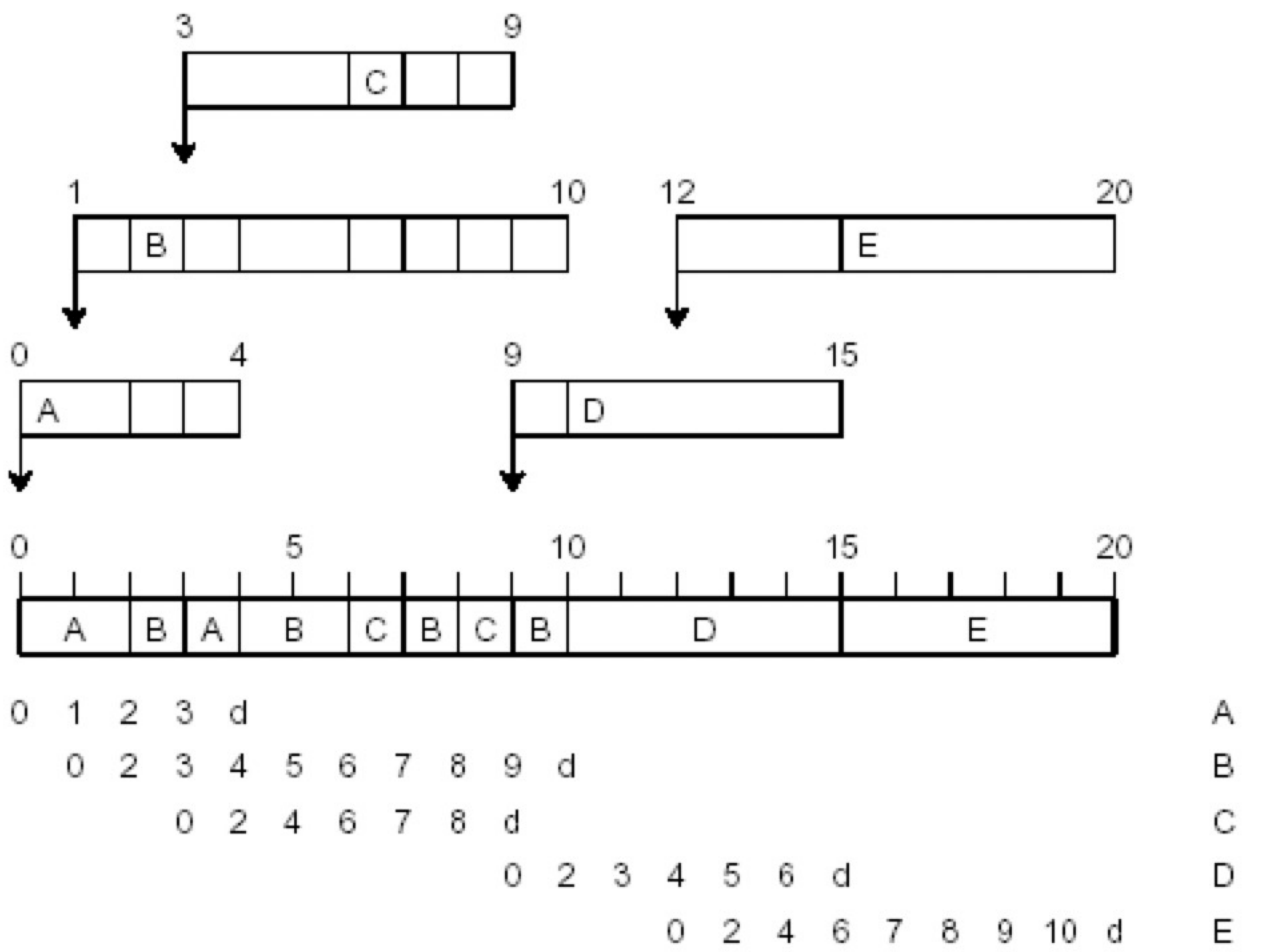
FCFS (First Come First Serving)



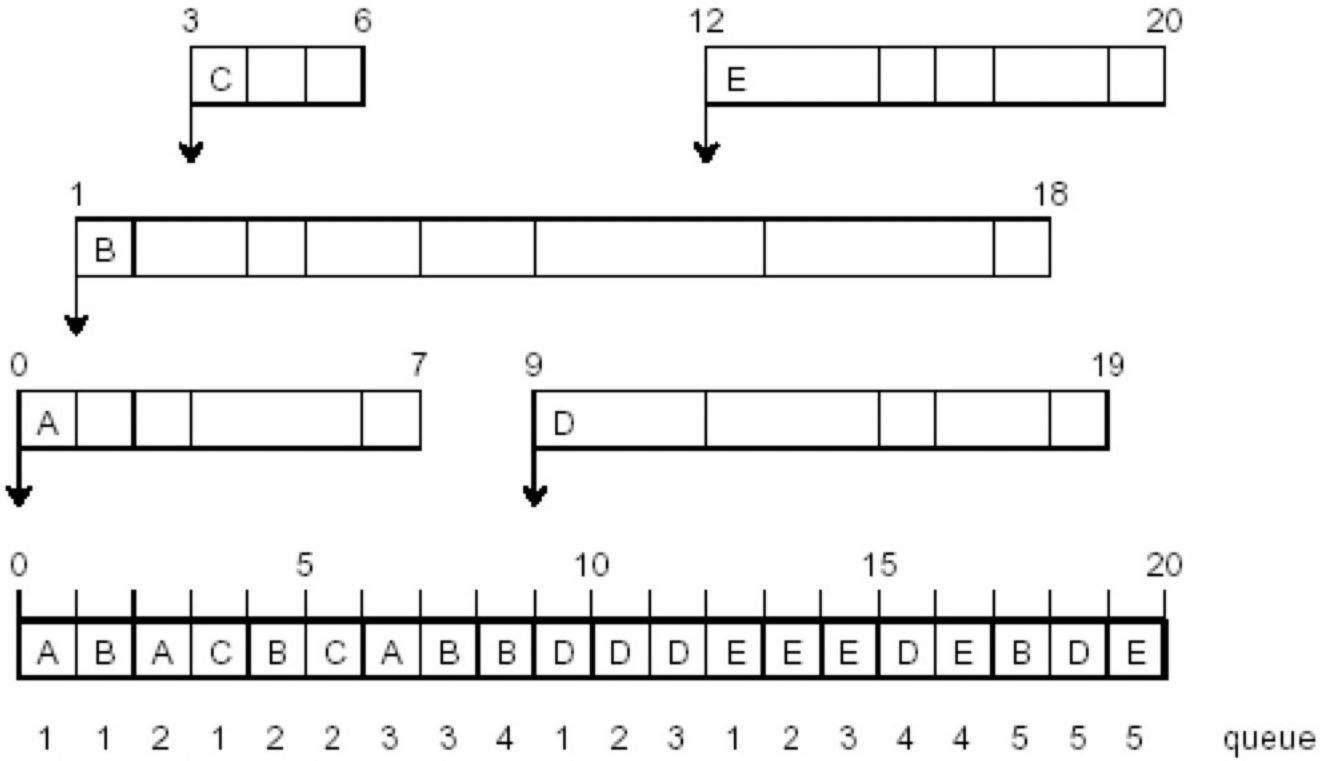
RR (Round Robin)

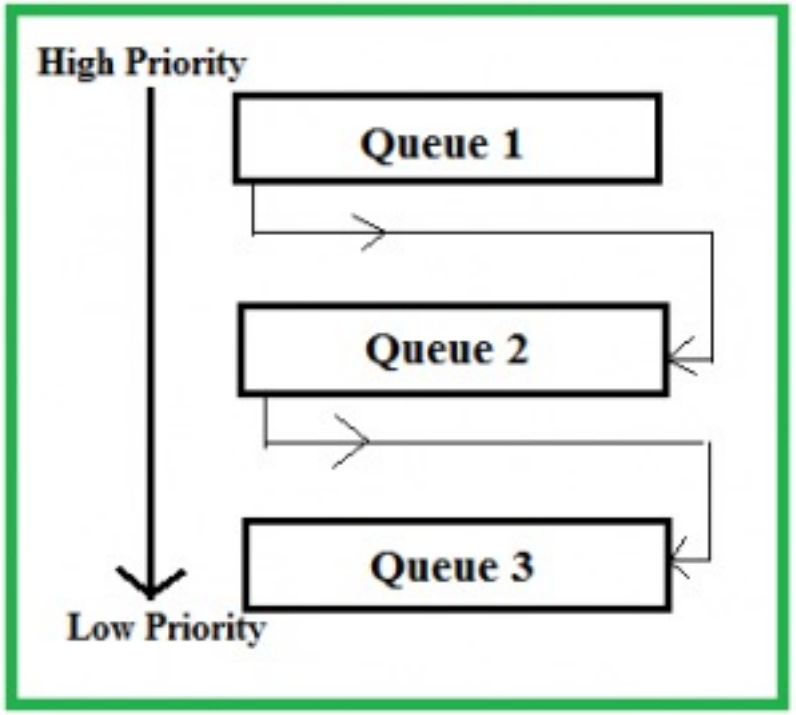


SRR (Selfish Round Robin)

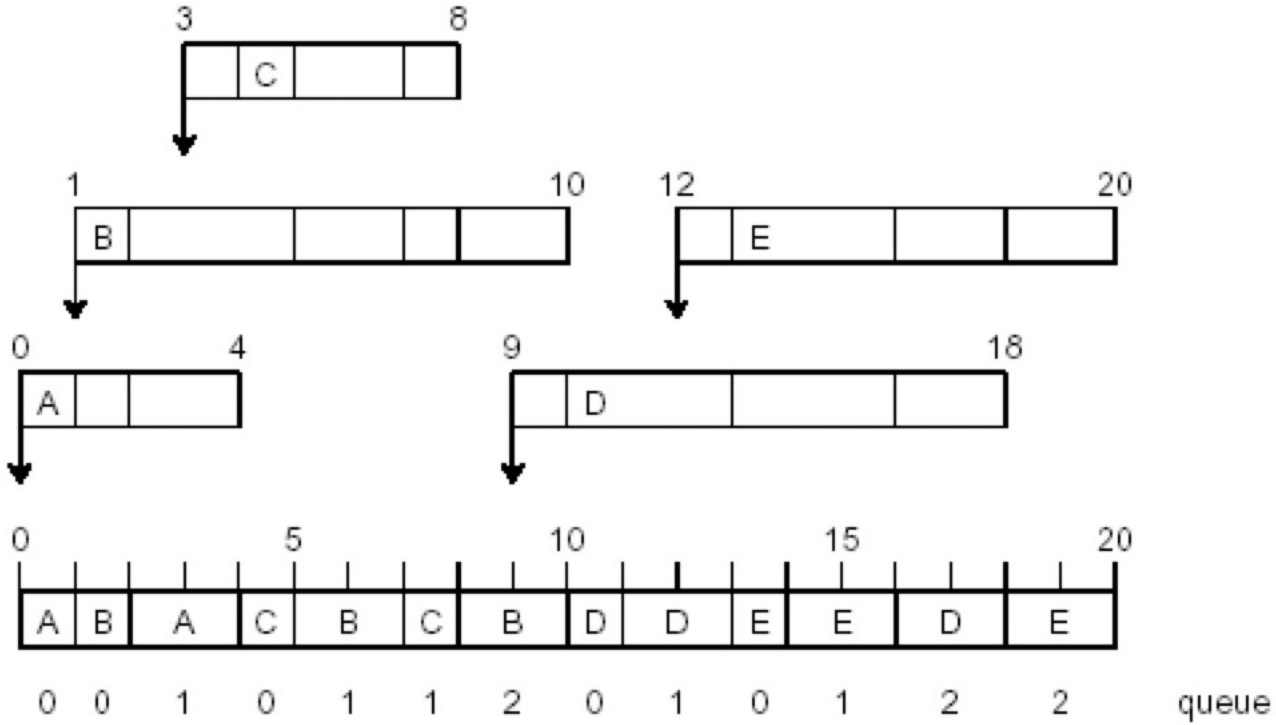


Multilevel Feedback Queue Scheduling (MLFQ)

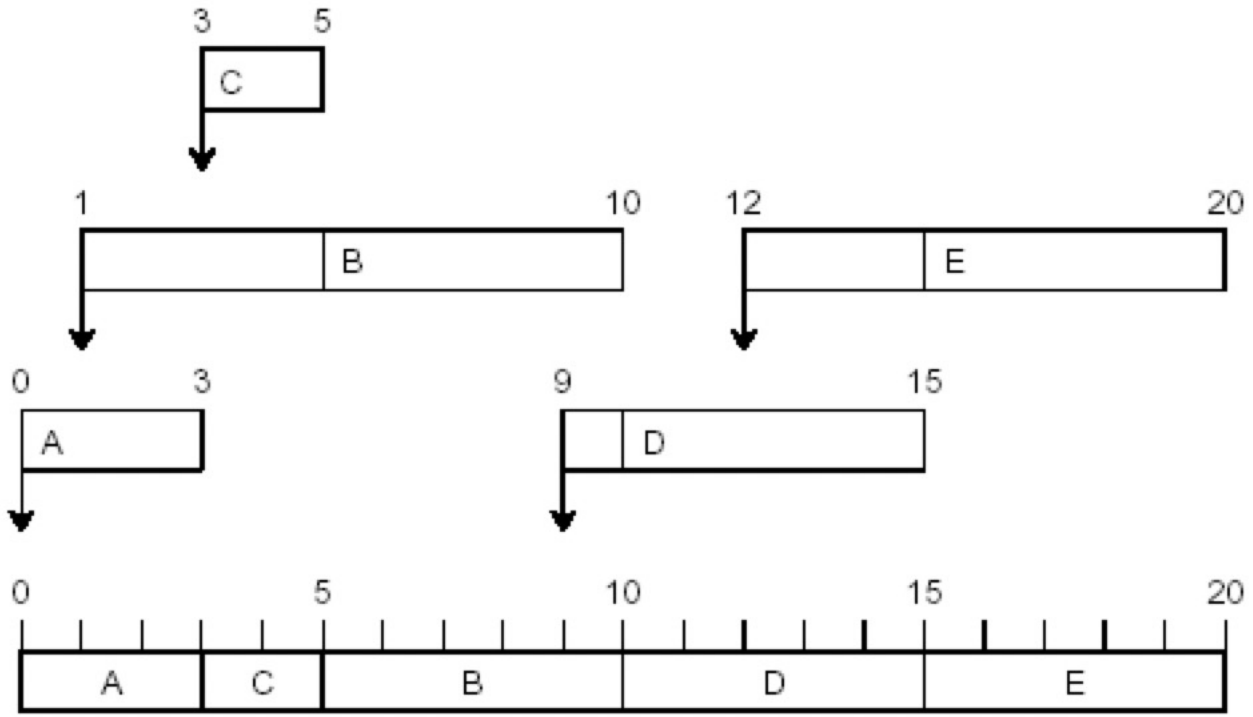




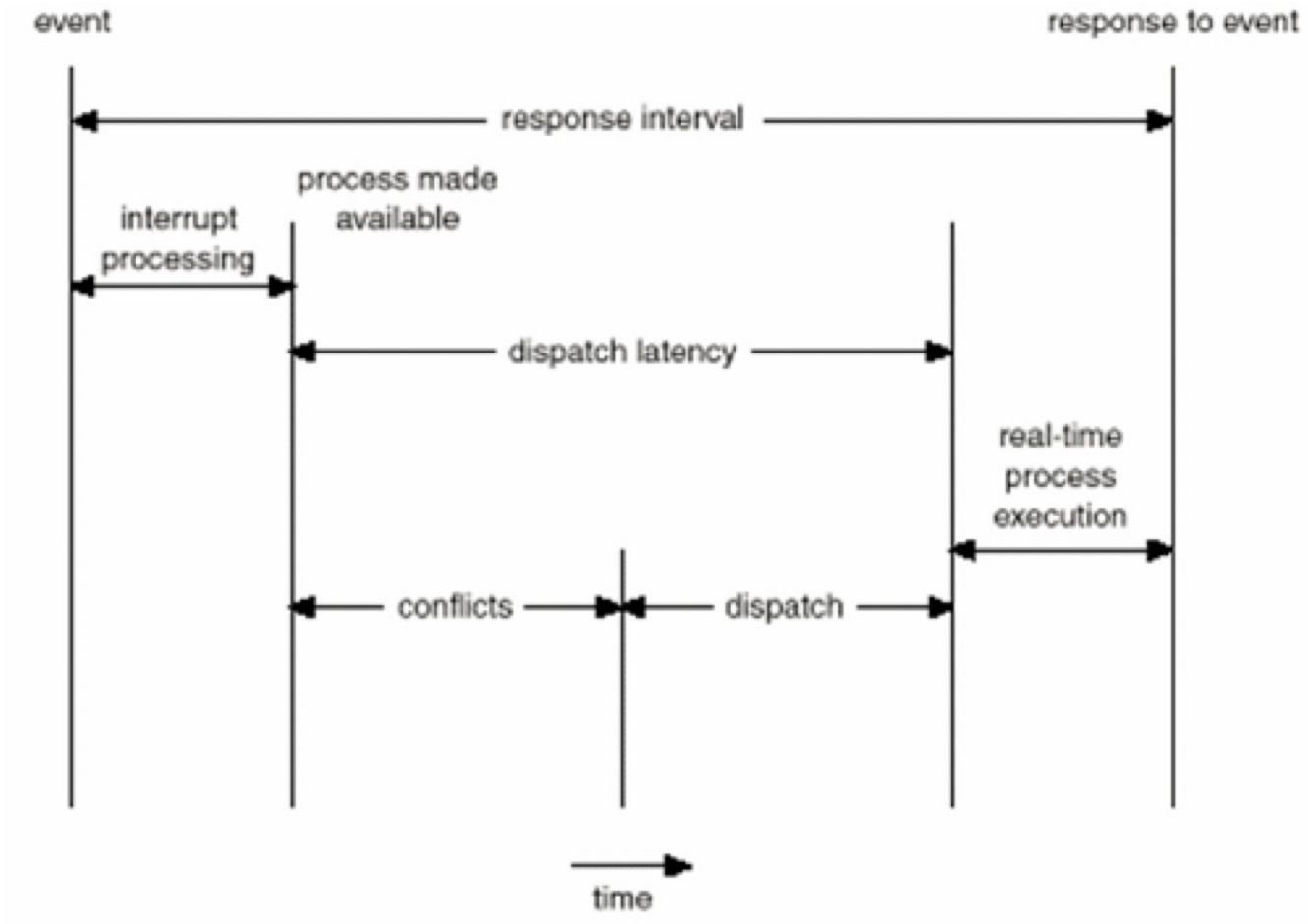
expFB $t = b^c$ $f = \frac{1}{b^c}$



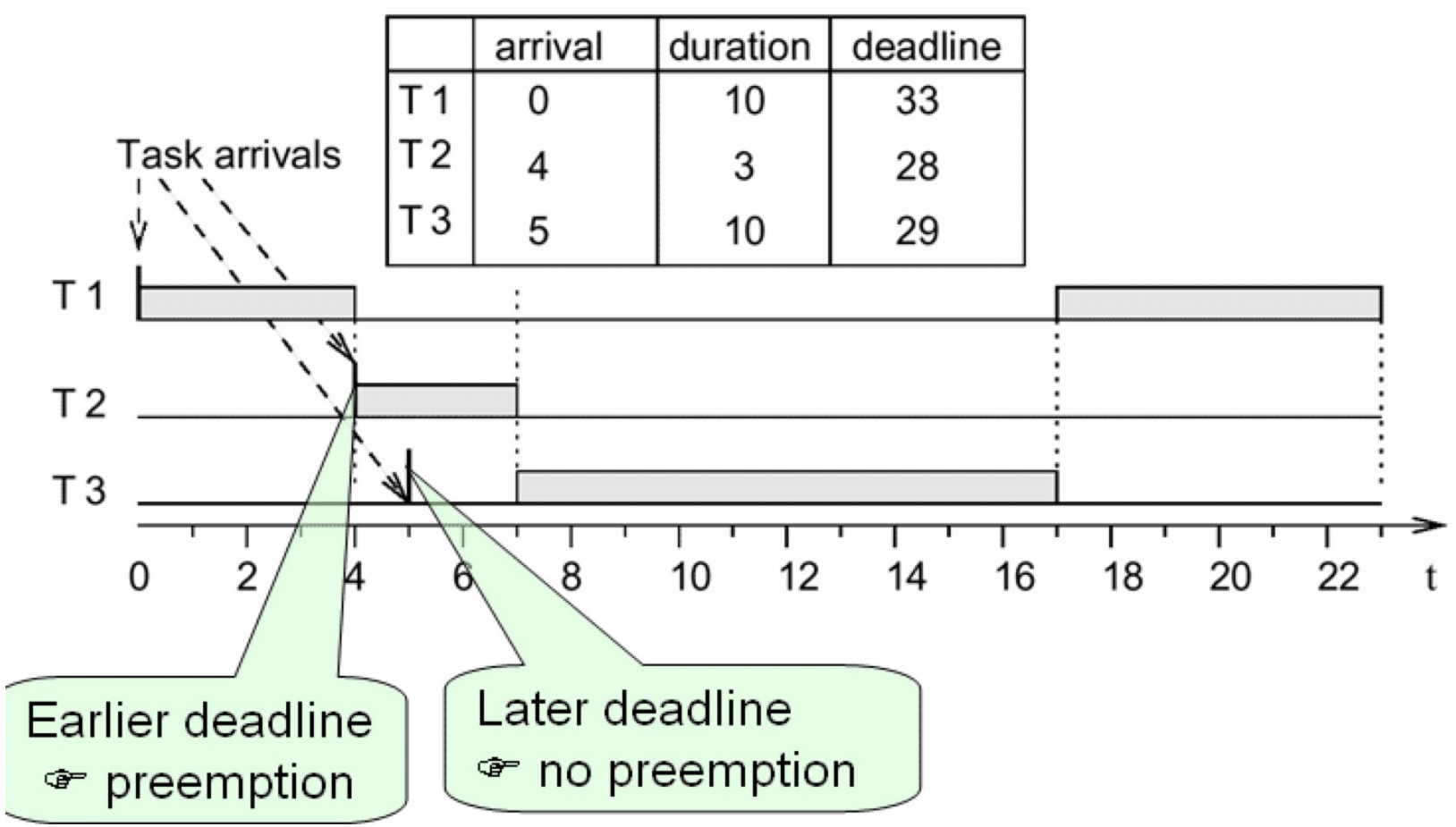
SPN (Shortest Process Next):



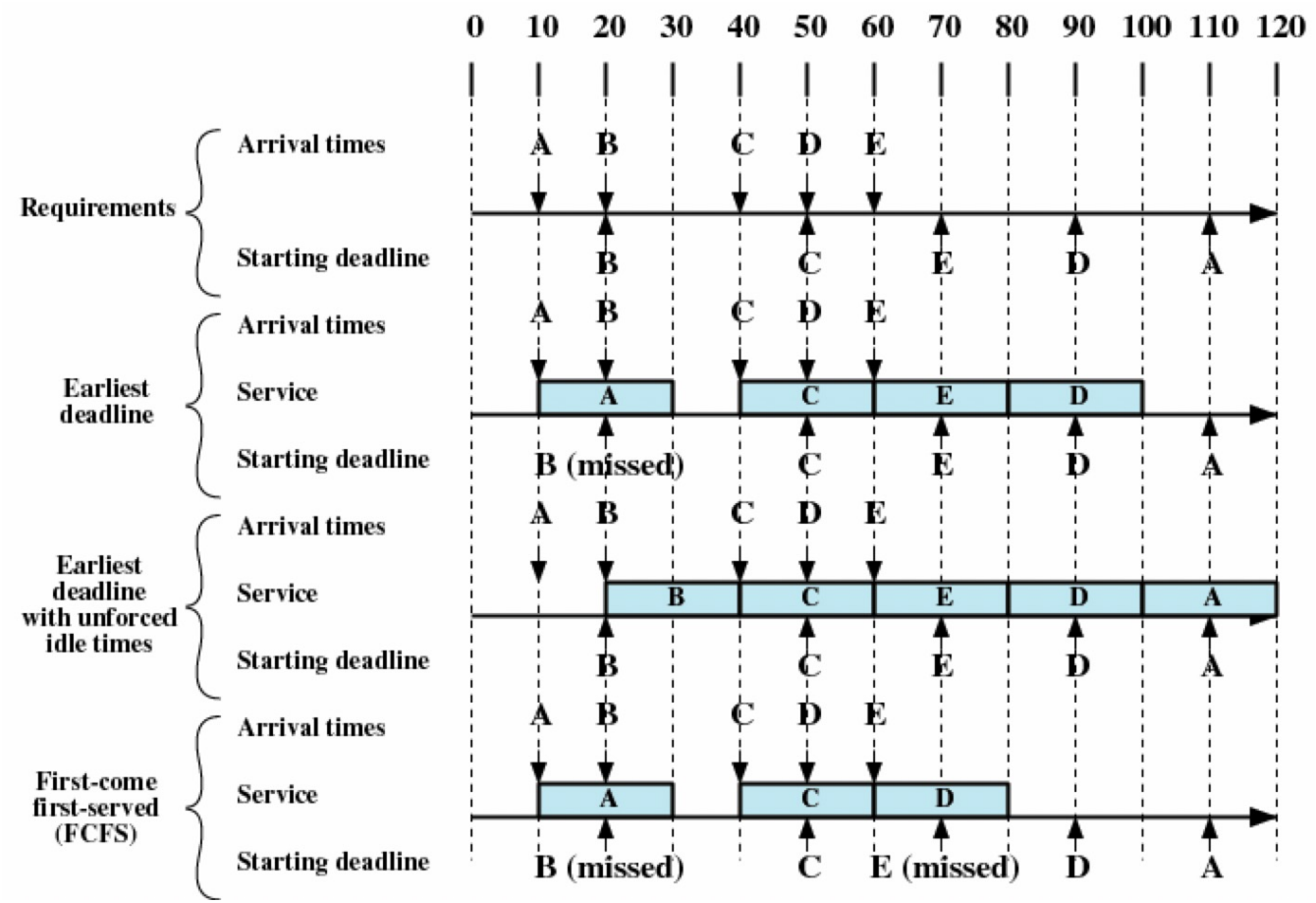
Real time Scheduling



EDF (Earliest Deadline first)



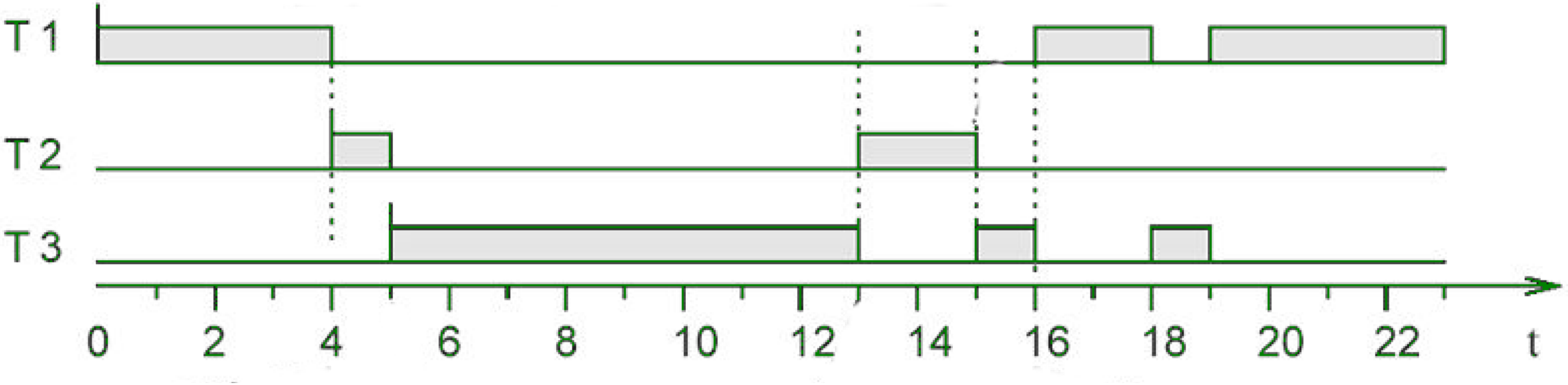
EDF (Earliest Deadline first)



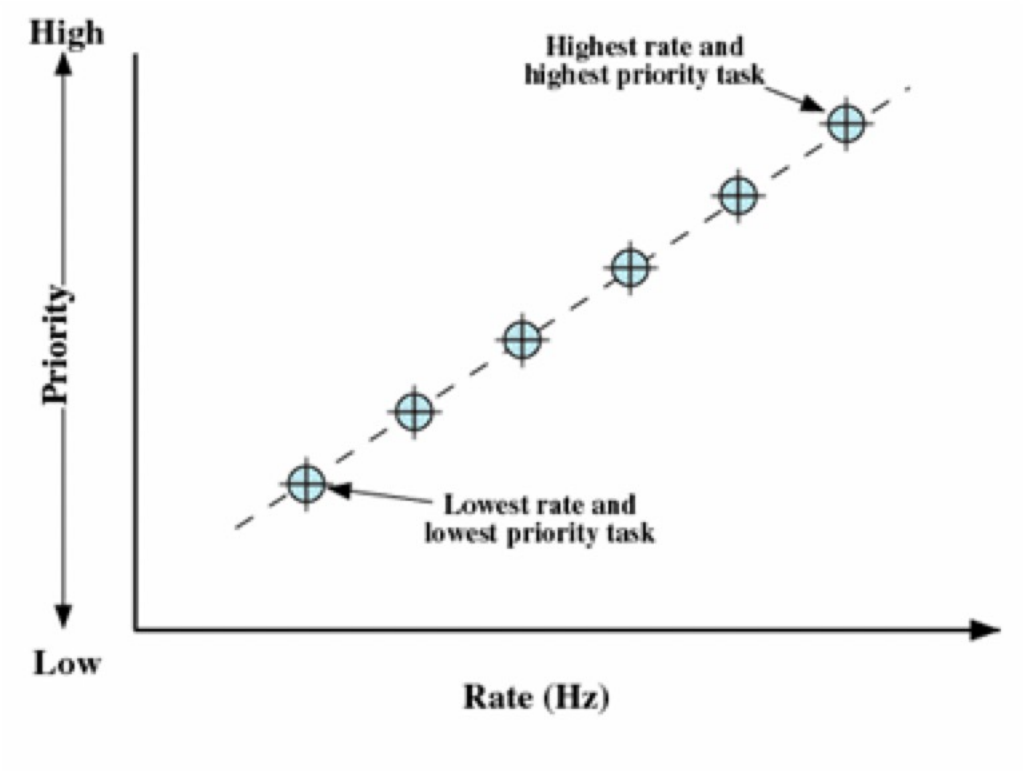
LL/LST (Least Laxity, Least Slack Time First)

	arrival	duration	deadline
T 1	0	10	33
T 2	4	3	28
T 3	5	10	29

slack_time = (D - t - e')



RMS (Rate Monotonic Scheduling)



$$\sum_{k=1}^{n} \frac{Ci}{Ti} \leq U = n\left(2^{1/n} - 1\right)$$