Scheduler

- ★ Processes' data
- ★ Processes scheduling

Preparation

- ★ Open the scenario:

  `home/students/inf/PUBLIC/SO/scenariusze/8`

- ★ Find bonus slides:

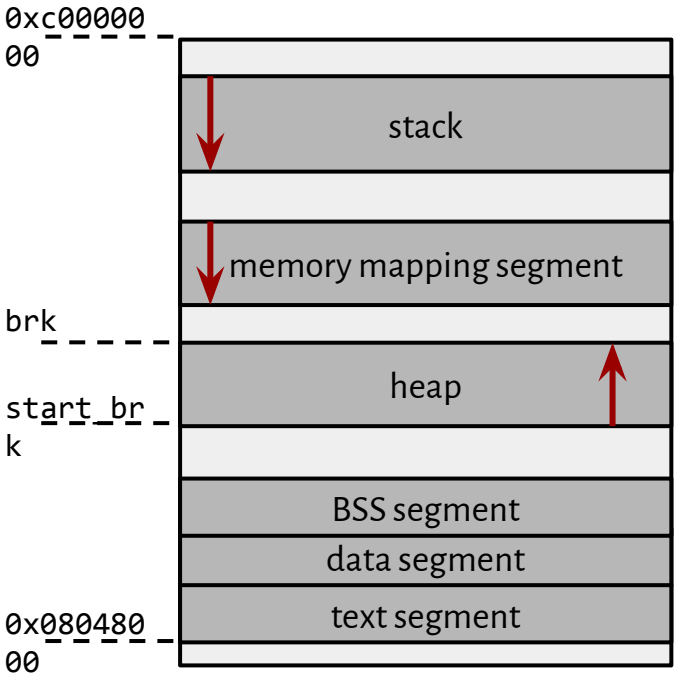
  `mimuw.edu.pl~/inga/SO2019/8`

- ★ Run MINIX on qemu

A concept of a **process**

Remember?

a group of related resources



Andrew S. Tanenbaum: *Operating Systems Design and Implementation*, 3rd Edition; p. 64 - 68

Processes data

memory mapping

the stack pointer (rsp)

the program counter (rip)

file descriptors

...

Processes data

memory mapping

the stack pointer (rsp)

the program counter (rip)

file descriptors

scheduling information

the state

...

Processes in MINIX

/usr/src/minix/servers/pm/mproc.h

/usr/src/minix/servers/vfs/fproc.h

Processes in MINIX

`/usr/src/minix/servers/pm/mproc.h` **`/usr/src/minix/servers/vfs/fproc.h`**

important PIDs

real/effective uid and gid

signal handling information

process' state

Processes in MINIX

`/usr/src/minix/servers/pm/mproc.h`

- important PIDs
- real/effective uid and gid
- signal handling information
- process' state

`/usr/src/minix/servers/vfs/fproc.h`

- the file descriptor table
- management of IO operations
- real/effective uid and gid
- segment sizes

Microkernel vs monolithic system

Microkernel vs monolithic system

Device Drivers

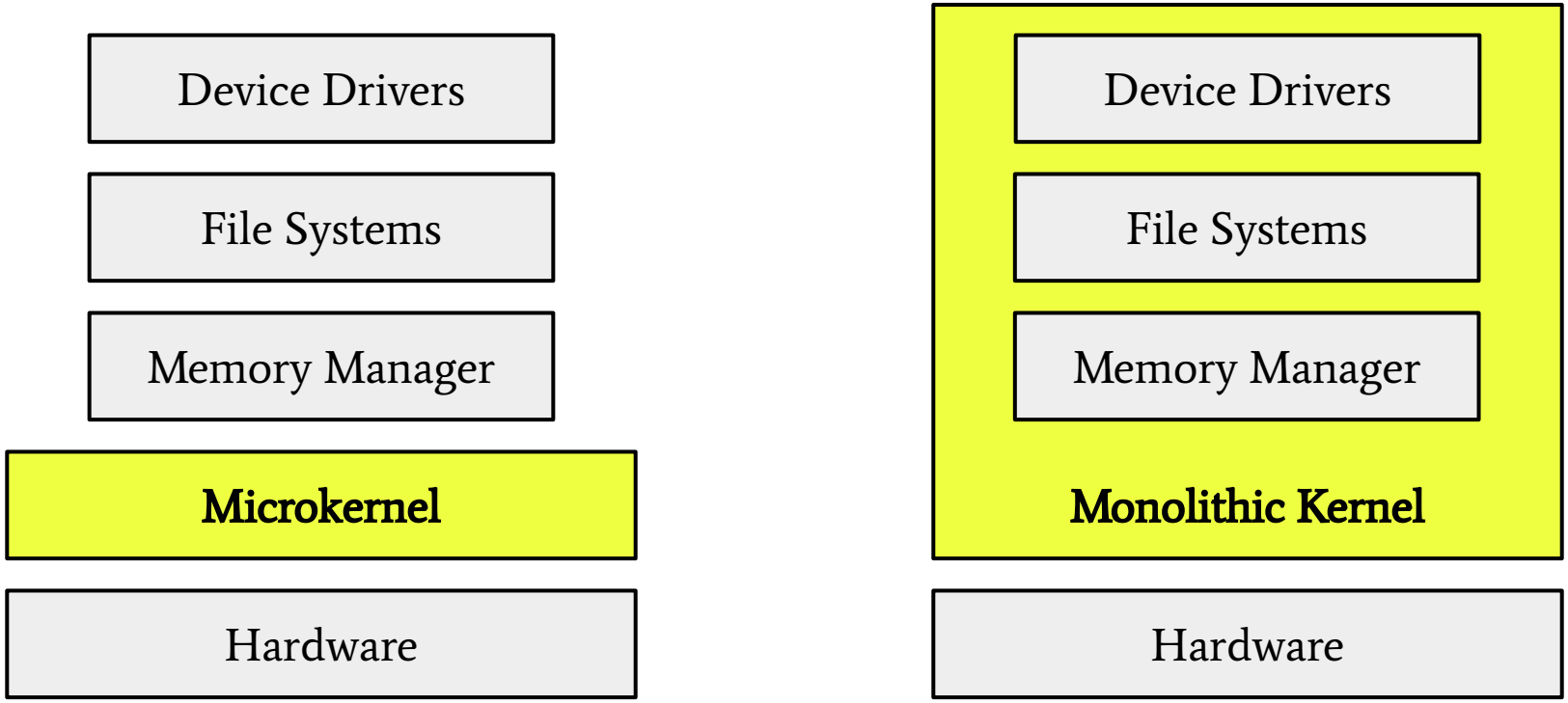
File Systems

Memory Manager

Microkernel

Hardware

Microkernel vs monolithic system



Microkernel vs monolithic system

"To me, writing a monolithic system in 1991 is a truly poor idea."

*"I've got more excuses than you have, and **Linux still beats the pants of minix in almost all areas**."*

*"That's one hell of a good excuse for some of the brain-damages of minix. I can only hope (and assume) that **Amoeba doesn't suck like minix does**."*

Christoph Lameter, *Extreme High Performance Computing or Why Microkernels Suck*

https://www.kernel.org/doc/ols/2007/ols2007v1-pages-251-262.pdf

Microkernel vs monolithic system

*"The argument in favor of a microkernel is that it allows a system to be **fail safe** since a **failure may be isolated** into one system component."*

Christoph Lameter, *Extreme High Performance Computing or Why Microkernels Suck*

https://www.kernel.org/doc/ols/2007/ols2007v1-pages-251-262.pdf

Microkernel vs monolithic system

*"The argument in favor of a microkernel is that it allows a system to be **fail safe** since a **failure may be isolated** into one system component."*

*"A failure of one key component typically includes also the loss of relevant **state information** about the application."*

Christoph Lameter, *Extreme High Performance Computing or Why Microkernels Suck*

https://www.kernel.org/doc/ols/2007/ols2007v1-pages-251-262.pdf

Microkernel vs monolithic system

*"The argument in favor of a microkernel is that it allows a system to be **fail safe** since a **failure may be isolated** into one system component."*

*"A failure of one key component typically includes also the loss of relevant **state information** about the application."*

*"However, the isolation of the operating system state into different modules will make it difficult to **track the overall system state** that needs to be preserved in order for **checkpointing** to work."*

https://en.wikipedia.org/wiki/Application_checkpointing

Christoph Lameter, *Extreme High Performance Computing or Why Microkernels Suck*

https://www.kernel.org/doc/ols/2007/ols2007v1-pages-251-262.pdf

Processes in MINIX kernel

`/usr/src/minix/kernel/proc.h`

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    struct stackframe_s p_reg;    /* process' registers saved in stack frame */
```

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h → **/usr/include/i386/stackframe.h**

```
struct stackframe_s {
    u16_t gs;
    u16_t fs;
    u16_t es;
    u16_t ds;
    reg_t di;
    reg_t si;
    reg_t fp;
    reg_t bx;
    reg_t dx;
    reg_t cx;
    reg_t retreg;

    reg_t pc;
    reg_t cs;
    reg_t psw;
    reg_t sp;
    reg_t ss;
};
```

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h → **/usr/include/i386/stackframe.h**

```
struct stackframe_s {
    u16_t gs;                   /* last item pushed by save */
    u16_t fs;                   /*  ^ */
    u16_t es;                   /*  | */
    u16_t ds;                   /*  | */
    reg_t di;                   /*  | */
    reg_t si;                   /*  | */
    reg_t fp;                   /*  | */
    reg_t bx;                   /*  | */
    reg_t dx;                   /*  | */
    reg_t cx;                   /*  | */
    reg_t retreg;               /* ax and above are all pushed by save */

    reg_t pc;                   /*  ^  last item pushed by interrupt */
    reg_t cs;                   /*  | */
    reg_t psw;                  /*  | */
    reg_t sp;                   /*  | */
    reg_t ss;                   /* these are pushed by CPU during interrupt */
};
```

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h → **/usr/include/i386/stackframe.h**

```
struct stackframe_s {
    u16_t gs;                   /* last item pushed by save */
    u16_t fs;                   /*  ^ */
    u16_t es;                   /*  | */
    u16_t ds;                   /*  | */
    reg_t di;                   /*  | */
    reg_t si;                   /*  | */
    reg_t fp;                   /*  | */
    reg_t bx;                   /*  | */
    reg_t dx;                   /*  | */
    reg_t cx;                   /*  | */
    reg_t retreg;               /* ax and above are all pushed by save */

    reg_t pc;                   /*  ^  last item pushed by interrupt */
    reg_t cs;                   /*  | */
    reg_t psw;                  /*  | */
    reg_t sp;                   /*  | */
    reg_t ss;                   /* these are pushed by CPU during interrupt */
};
```

data registers

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h → **/usr/include/i386/stackframe.h**

```
struct stackframe_s {
    u16_t gs;                   /* last item pushed by save */
    u16_t fs;                   /*  ^ */
    u16_t es;                   /*  | */
    u16_t ds;                   /*  | */
    reg_t di;                   /*  | */
    reg_t si;                   /*  | */
    reg_t fp;                   /*  | */
    reg_t bx;                   /*  | */
    reg_t dx;                   /*  | */
    reg_t cx;                   /*  | */
    reg_t retreg;               /* ax and above are all pushed by save */

    reg_t pc;                   /*  ^  last item pushed by interrupt */
    reg_t cs;                   /*  | */
    reg_t psw;                  /*  | */
    reg_t sp;                   /*  | */
    reg_t ss;                   /* these are pushed by CPU during interrupt */
};
```

data registers

index registers

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h → **/usr/include/i386/stackframe.h**

```
struct stackframe_s {
     u16_t gs;                    /* last item pushed by save */
     u16_t fs;                    /*   ^ */
     u16_t es;                    /*   | */
     u16_t ds;                    /*   | */
     reg_t di;                    /*   | */
     reg_t si;                    /*   | */
     reg_t fp;                    /*   | */
     reg_t bx;                    /*   | */
     reg_t dx;                    /*   | */
     reg_t cx;                    /*   | */
     reg_t retreg;                /* ax and above are all pushed by save */

     reg_t pc;                    /*   ^  last item pushed by interrupt */
     reg_t cs;                    /*   | */
     reg_t psw;                   /*   | */
     reg_t sp;                    /*   | */
     reg_t ss;                    /* these are pushed by CPU during interrupt */
};
```

data registers **index registers** **pointer registers**

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h → **/usr/include/i386/stackframe.h**

```
struct stackframe_s {
     u16_t gs;                  /* last item pushed by save */
     u16_t fs;                  /*   ^ */
     u16_t es;                  /*   | */
     u16_t ds;                  /*   | */
     reg_t di;                  /*   | */
     reg_t si;                  /*   | */
     reg_t fp;                  /*   | */
     reg_t bx;                  /*   | */
     reg_t dx;                  /*   | */
     reg_t cx;                  /*   | */
     reg_t retreg;              /* ax and above are all pushed by save */

     reg_t pc;                  /*   ^  last item pushed by interrupt */
     reg_t cs;                  /*   | */
     reg_t psw;                 /*   | */
     reg_t sp;                  /*   | */
     reg_t ss;                  /* these are pushed by CPU during interrupt */
};
```

data registers | **index registers** | **pointer registers** | **segment registers**

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h → **/usr/include/i386/stackframe.h**

```
struct stackframe_s {
     u16_t gs;                  /* last item pushed by save */
     u16_t fs;                  /*   ^ */
     u16_t es;                  /*   | */
     u16_t ds;                  /*   | */
     reg_t di;                  /*   | */
     reg_t si;                  /*   | */
     reg_t fp;                  /*   | */
     reg_t bx;                  /*   | */
     reg_t dx;                  /*   | */
     reg_t cx;                  /*   | */
     reg_t retreg;              /* ax and above are all pushed by save */

     reg_t pc;                  /*   ^  last item pushed by interrupt */
     reg_t cs;                  /*   | */
     reg_t psw;                 /*   | */
     reg_t sp;                  /*   | */
     reg_t ss;                  /* these are pushed by CPU during interrupt */
};
```

data registers | **index registers** | **pointer registers** | **segment registers** | **program status word**

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    struct stackframe_s p_reg;    /* process' registers saved in stack frame */
    struct segframe p_seg;        /* segment descriptors                     */
```

Processes in MINIX kernel

`/usr/src/minix/kernel/proc.h` → **`/usr/include/i386/archtypes.h`**

```
typedef struct segframe {
    reg_t    p_cr3;                /* page table root */
    u32_t*   p_cr3_v;
    char *   fpu_state;
    int      p_kern_trap_style;
} segframe_t;
```

CR3

"CR3 enables the processor to translate linear addresses into physical addresses by locating the page directory and page tables for the current task. "

`https://en.wikipedia.org/wiki/Control_register#CR3`

FPU

"the vector processing units, which just happens to include the original floating point operations"

`http://wiki.osdev.org/FPU`

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    struct stackframe_s p_reg;    /* process' registers saved in stack frame */
    struct segframe p_seg;        /* segment descriptors                     */
    proc_nr_t p_nr;               /* number of this process (fast access)    */
```

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    struct stackframe_s p_reg;    /* process' registers saved in stack frame */
    struct segframe p_seg;        /* segment descriptors                     */
    proc_nr_t p_nr;               /* number of this process (fast access)    */
    struct priv *p_priv;          /* system privileges structure             */
```

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h → **/usr/src/minix/kernel/priv.h**

allowed I/O ports

allowed memory ranges

allowed IRQ (Interrupt Requests) lines

signal manager

pending signals

pending hardware interrupts

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    struct stackframe_s p_reg;    /* process' registers saved in stack frame */
    struct segframe p_seg;        /* segment descriptors                     */
    proc_nr_t p_nr;               /* number of this process (fast access)    */
    struct priv *p_priv;          /* system privileges structure             */

    volatile u32_t p_rts_flags;   /* process is runnable only if zero        */
    volatile u32_t p_misc_flags;  /* flags that do not suspend the process   */
```

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h → **/usr/src/minix/kernel/proc.h**

```
RUNTIME FLAGS

#define RTS_SLOT_FREE     0x01    /* process slot is free */
#define RTS_PROC_STOP     0x02    /* process has been stopped */
#define RTS_SENDING       0x04    /* process blocked trying to send */
#define RTS_RECEIVING     0x08    /* process blocked trying to receive */
#define RTS_SIGNALED      0x10    /* set when new kernel signal arrives */
#define RTS_SIG_PENDING   0x20    /* unready while signal being processed */
#define RTS_P_STOP        0x40    /* set when process is being traced */
...

MISCELLANEOUS FLAGS

#define MF_REPLY_PEND     0x001   /* reply to IPC_REQUEST is pending */
#define MF_VIRT_TIMER     0x002   /* process-virtual timer is running */
#define MF_PROF_TIMER     0x004   /* process-virtual profile timer is running */
...
#define MF_SC_ACTIVE      0x100   /* Syscall tracing: in a system call now */
#define MF_SC_DEFER       0x200   /* Syscall tracing: deferred system call */
#define MF_SC_TRACE       0x400   /* Syscall tracing: trigger syscall events */
...
#define MF_SPROF_SEEN     0x8000  /* profiling has seen this process */
```

BTW: bitflags

```
uint8_t features;              /* reserve 8 bits for the features */


enum Features {
  fast              = 0x01,
  reliable          = 0x02,
  stable            = 0x04,
  efficient         = 0x08,
  correct           = 0x10,
  unique            = 0x20,
  ...
};
```

BTW: bitflags

```
uint8_t features;              /* reserve 8 bits for the features */


enum Features {
  fast              = 0x01,    /* 00000001 */
  reliable          = 0x02,    /* 00000010 */
  stable            = 0x04,    /* 00000100 */
  efficient         = 0x08,    /* 00001000 */
  correct           = 0x10,    /* 00010000 */
  unique            = 0x20,    /* 00100000 */
  ...
};
```

BTW: bitflags

```
uint8_t features;                /* reserve 8 bits for the features */


enum Features {
  fast              = 0x01,     /* 00000001 */
  reliable          = 0x02,     /* 00000010 */
  stable            = 0x04,     /* 00000100 */
  efficient         = 0x08,     /* 00001000 */
  correct           = 0x10,     /* 00010000 */
  unique            = 0x20,     /* 00100000 */
  ...
};


features = fast | reliable | correct;     /* set flags */


if (features & efficient) { ... }         /* test flags */
```

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    struct stackframe_s p_reg;    /* process' registers saved in stack frame */
    struct segframe p_seg;        /* segment descriptors                     */
    proc_nr_t p_nr;               /* number of this process (fast access)    */
    struct priv *p_priv;          /* system privileges structure             */

    volatile u32_t p_rts_flags;   /* process is runnable only if zero        */
    volatile u32_t p_misc_flags;  /* flags that do not suspend the process   */

    char p_priority;              /* current process priority                */
    struct proc *p_scheduler;     /* who should get out of quantum msg       */
```

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    struct stackframe_s p_reg;    /* process' registers saved in stack frame */
    struct segframe p_seg;        /* segment descriptors                     */
    proc_nr_t p_nr;               /* number of this process (fast access)    */
    struct priv *p_priv;          /* system privileges structure             */

    volatile u32_t p_rts_flags;   /* process is runnable only if zero        */
    volatile u32_t p_misc_flags;  /* flags that do not suspend the process   */

    char p_priority;              /* current process priority                */
    struct proc *p_scheduler;     /* who should get out of quantum msg       */

    u64_t p_cpu_time_left;        /* time left to use the cpu                */
    unsigned p_cpu;               /* what CPU is the process running on      */
```

execution on a CPU

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    struct stackframe_s p_reg;    /* process' registers saved in stack frame */
    struct segframe p_seg;        /* segment descriptors                     */
    proc_nr_t p_nr;               /* number of this process (fast access)    */
    struct priv *p_priv;          /* system privileges structure             */

    volatile u32_t p_rts_flags;   /* process is runnable only if zero        */
    volatile u32_t p_misc_flags;  /* flags that do not suspend the process   */

    char p_priority;              /* current process priority                */
    struct proc *p_scheduler;     /* who should get out of quantum msg       */

    u64_t p_cpu_time_left;        /* time left to use the cpu                */
    unsigned p_cpu;               /* what CPU is the process running on      */

    clock_t p_user_time;          /* user time in ticks                      */
    clock_t p_sys_time;           /* sys time in ticks                       */
```

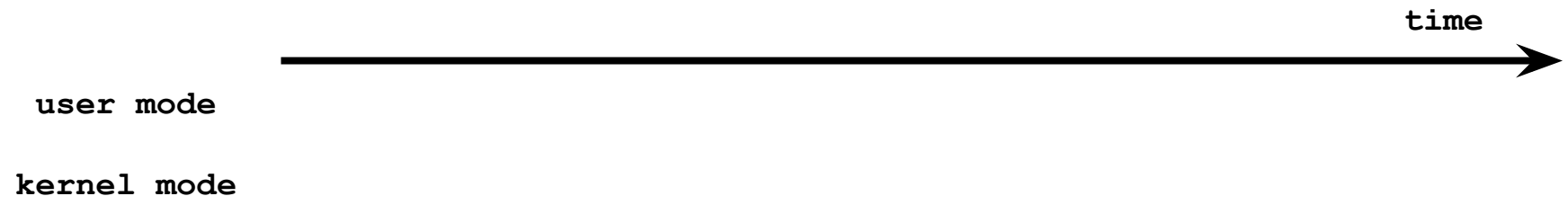
time taken

BTW: time

```
# time grep -nri “p_user_time” /usr/src/minix
```

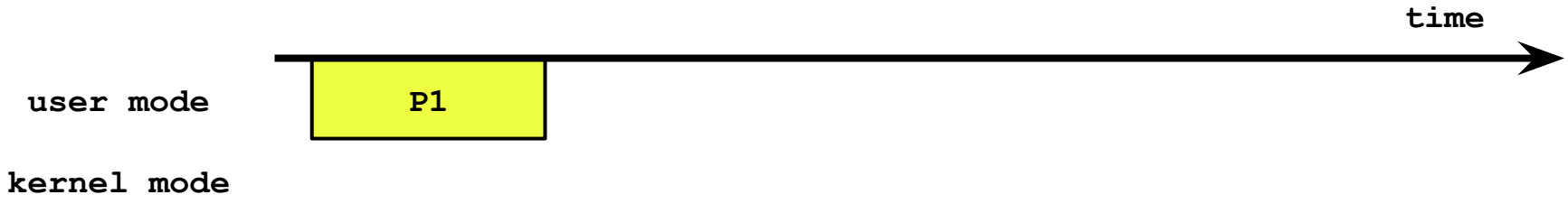
BTW: time

```
# time grep -nri "p_user_time" /usr/src/minix
```



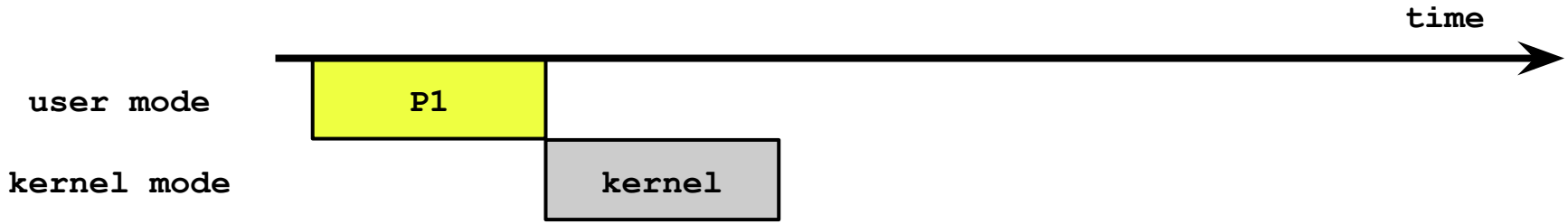
BTW: time

```
# time grep -nri "p_user_time" /usr/src/minix
```



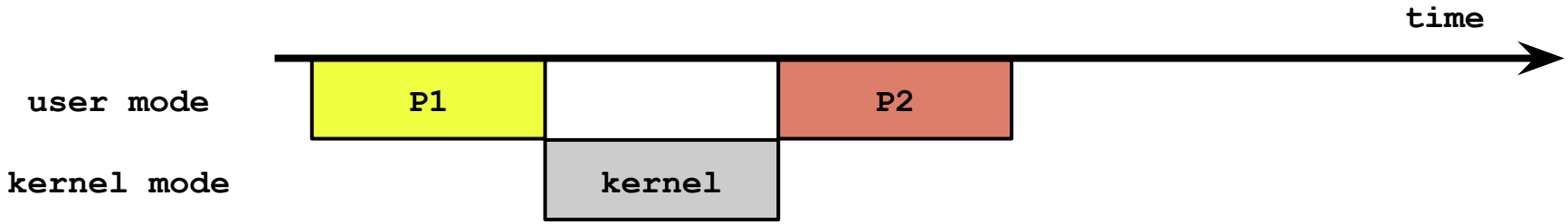
BTW: time

```
# time grep -nri "p_user_time" /usr/src/minix
```



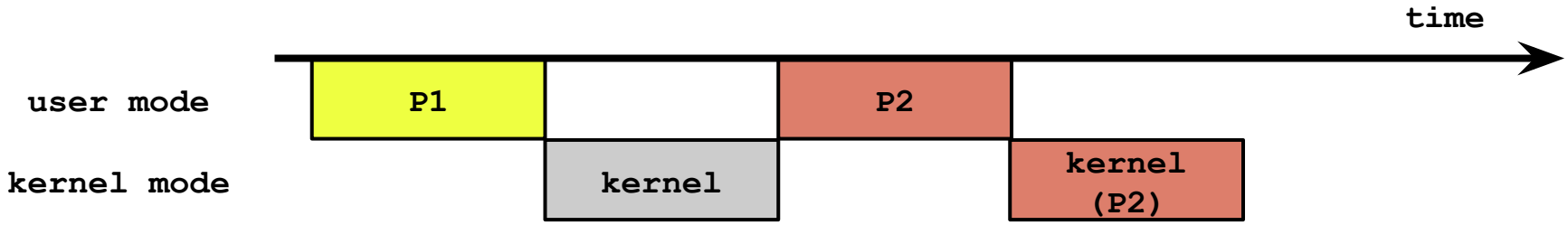
BTW: time

```
# time grep -nri "p_user_time" /usr/src/minix
```



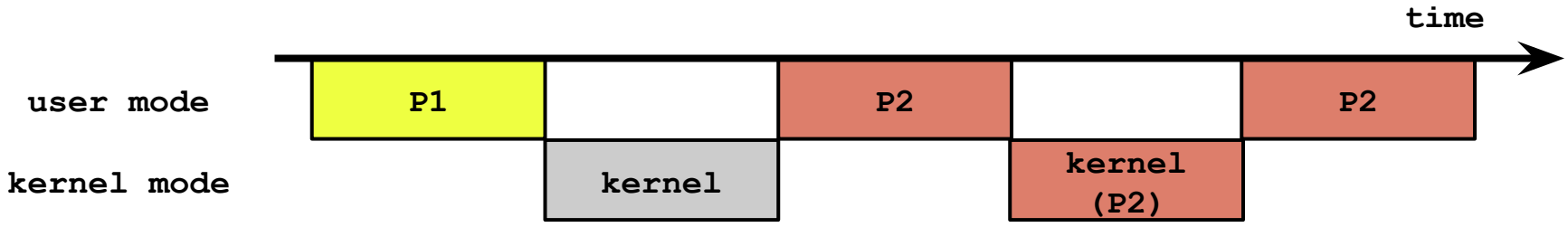
BTW: time

```
# time grep -nri "p_user_time" /usr/src/minix
```



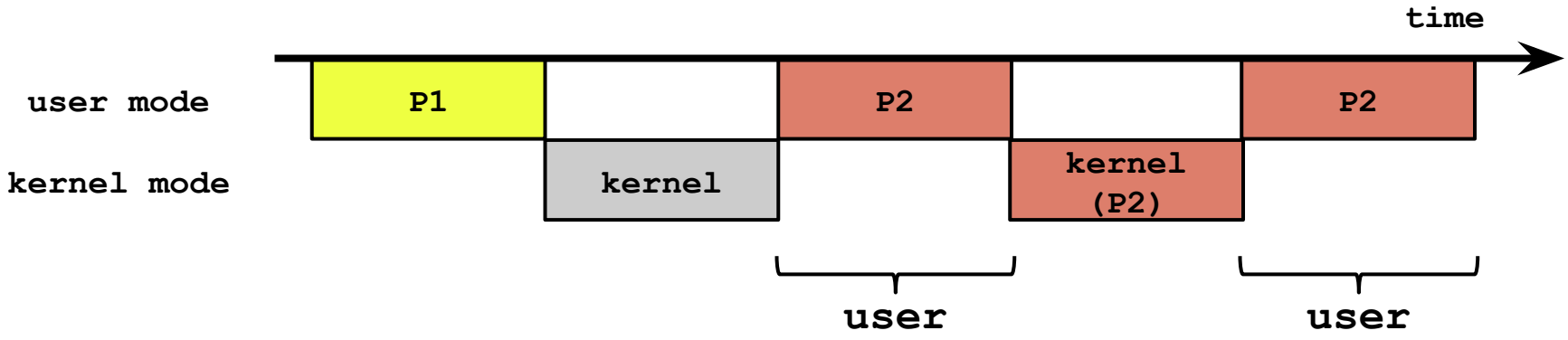
BTW: time

```
# time grep -nri "p_user_time" /usr/src/minix
```



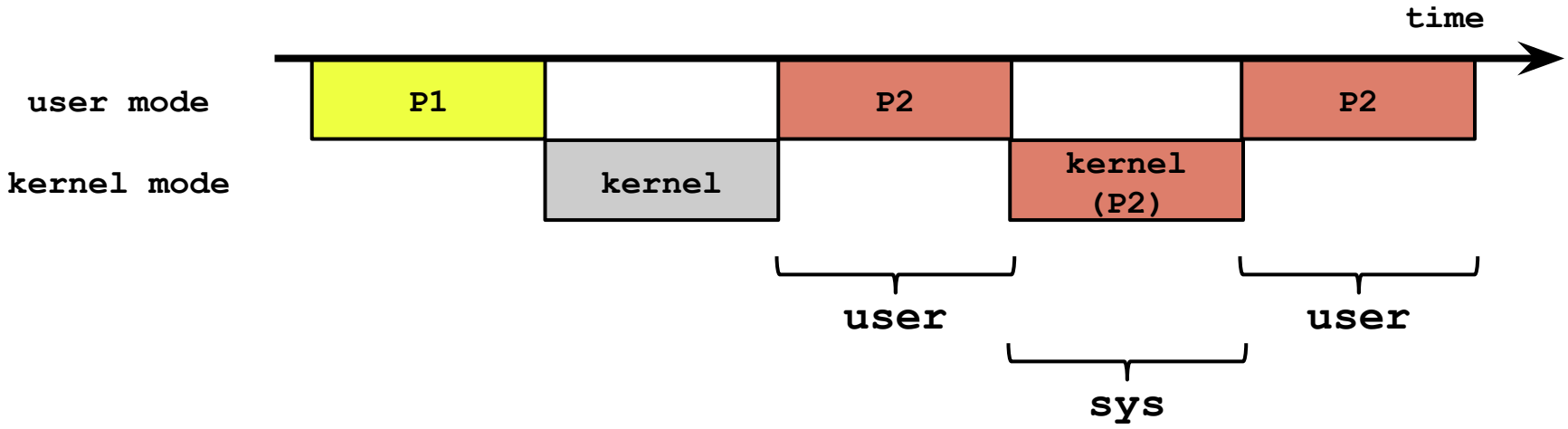
BTW: time

```
# time grep -nri "p_user_time" /usr/src/minix
```



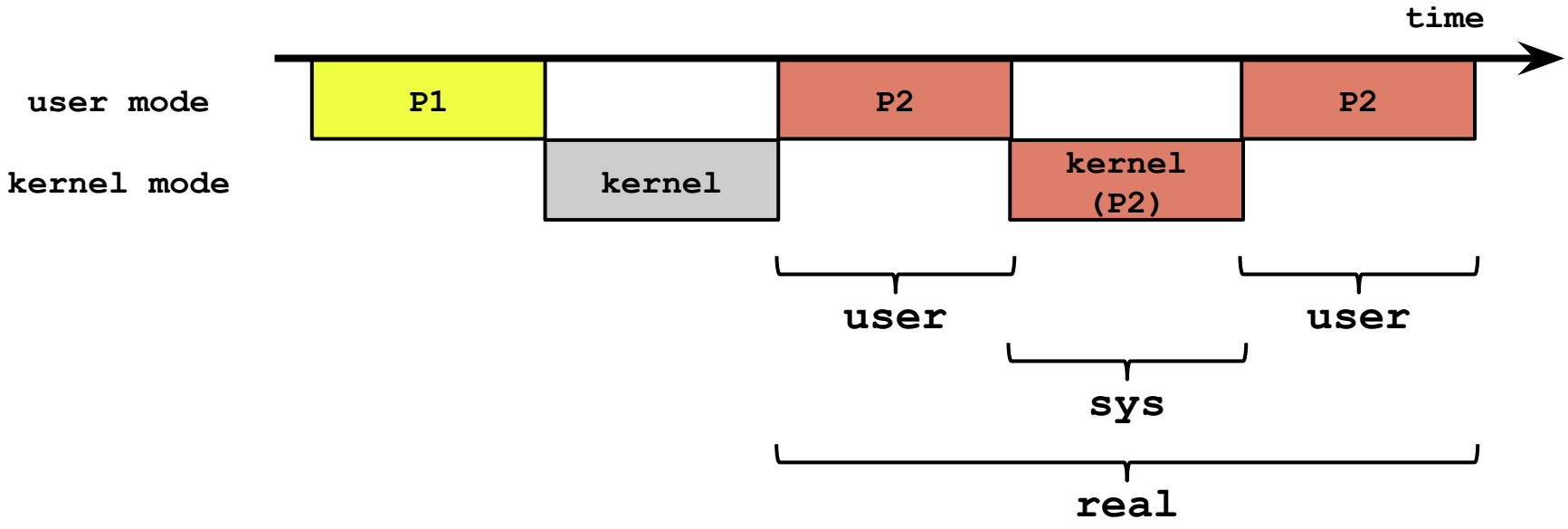
BTW: time

```
# time grep -nri "p_user_time" /usr/src/minix
```



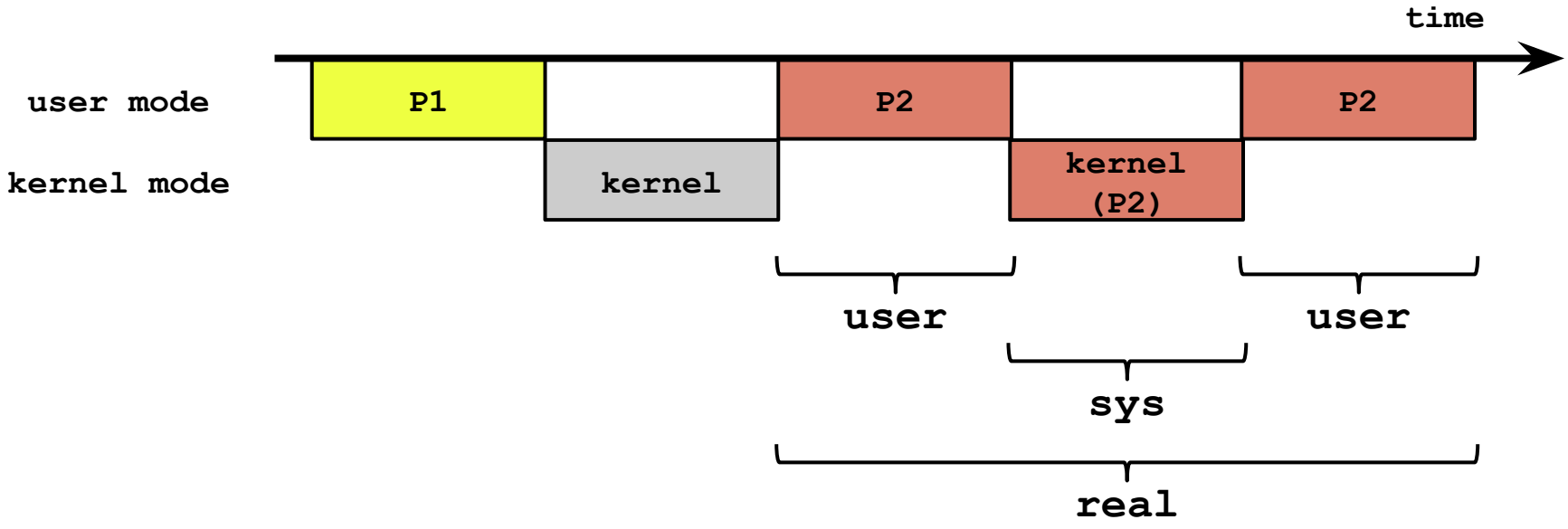
BTW: time

```
# time grep -nri "p_user_time" /usr/src/minix
```



BTW: time

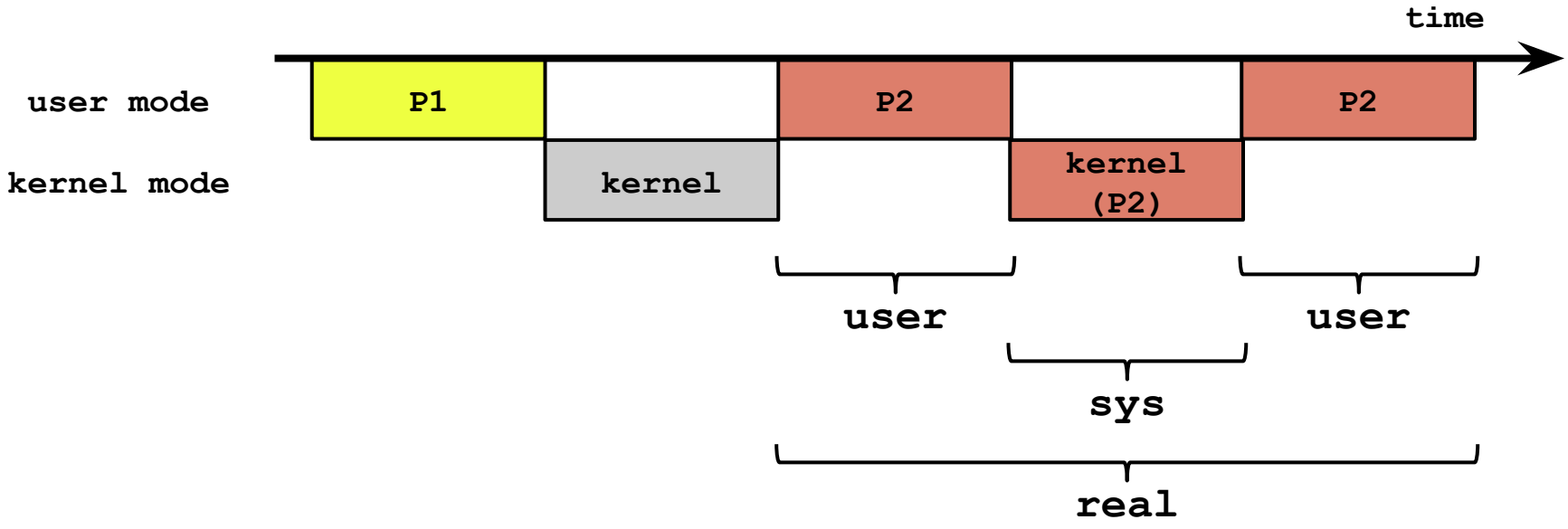
```
# time grep -nri "p_user_time" /usr/src/minix
```



Is is true? **real = user + sys**

BTW: time

```
# time grep -nri "p_user_time" /usr/src/minix
```



Is is true? **real = user + sys**

Nope. http://stackoverflow.com/a/2734965

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    struct stackframe_s p_reg;    /* process' registers saved in stack frame */
    struct segframe p_seg;        /* segment descriptors                     */
    proc_nr_t p_nr;               /* number of this process (fast access)    */
    struct priv *p_priv;          /* system privileges structure
*/
    volatile u32_t p_rts_flags;   /* process is runnable only if zero        */
    volatile u32_t p_misc_flags;  /* flags that do not suspend the process   */

    char p_priority;              /* current process priority                */
    struct proc *p_scheduler;     /* who should get out of quantum msg       */

    u64_t p_cpu_time_left;        /* time left to use the cpu                */
    unsigned p_cpu;               /* what CPU is the process running on      */

    clock_t p_user_time;          /* user time in ticks                      */
    clock_t p_sys_time;           /* sys time in ticks                       */

    clock_t p_virt_left;          /* number of ticks left on virtual timer   */
    clock_t p_prof_left;          /* number of ticks left on profile timer   */
```

time left

BTW: timers

`real`

`virtual`

`profile`

BTW: timers

real

ticks in real time

SIGALARM

virtual

profile

BTW: timers

real

ticks in real time

SIGALARM

virtual

ticks **only** when the process is running in **user mode**

SIGVTALARM

profile

BTW: timers

real

ticks in real time

SIGALARM

virtual

ticks **only** when the process is running in **user mode**

SIGVTALARM

profile

ticks when the process is running in **user mode** **and** when the system is executing on behalf of the process

SIGPROF

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    ...
    u64_t p_cycles;            /* how many cycles did the process use
*/
    u64_t p_kcall_cycles;      /* kernel cycles caused by this proc (kcall    */
    u64_t p_kipc_cycles;       /* cycles caused by this proc (ipc)            */
```

time used - measured in cycles http://stackoverflow.com/a/25744333

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    ...
    u64_t p_cycles;           /* how many cycles did the process use
*/
    u64_t p_kcall_cycles;     /* kernel cycles caused by this proc (kcall)   */
    u64_t p_kipc_cycles;      /* cycles caused by this proc (ipc)            */


    struct proc *p_nextready; /* pointer to next ready process               */
    struct proc *p_caller_q;  /* head of list of procs wishing to send       */
    struct proc *p_q_link;    /* link to next proc wishing to send           */
```

information about other processes

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    ...
    u64_t p_cycles;           /* how many cycles did the process use
*/
    u64_t p_kcall_cycles;     /* kernel cycles caused by this proc (kcall)   */
    u64_t p_kipc_cycles;      /* cycles caused by this proc (ipc)            */


    struct proc *p_nextready; /* pointer to next ready process               */
    struct proc *p_caller_q;  /* head of list of procs wishing to send       */
    struct proc *p_q_link;    /* link to next proc wishing to send           */

    endpoint_t p_endpoint;    /* endpoint number, generation-aware           */
    endpoint_t p_getfrom_e;   /* from whom does process want to receive?
*/
    endpoint_t p_sendto_e;    /* to whom does process want to send?          */
```

IPC - endpoint addresses

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    ...
    u64_t p_cycles;           /* how many cycles did the process use
*/
    u64_t p_kcall_cycles;     /* kernel cycles caused by this proc (kcall)   */
    u64_t p_kipc_cycles;      /* cycles caused by this proc (ipc)            */


    struct proc *p_nextready; /* pointer to next ready process               */
    struct proc *p_caller_q;  /* head of list of procs wishing to send       */
    struct proc *p_q_link;    /* link to next proc wishing to send           */

    endpoint_t p_endpoint;    /* endpoint number, generation-aware           */
    endpoint_t p_getfrom_e;   /* from whom does process want to receive?
*/
    endpoint_t p_sendto_e;    /* to whom does process want to send?          */

    vir_bytes p_delivermsg_vir; /* Virtual addr this proc wants message at   */
    message p_sendmsg;          /* Message from this process if SENDING      */
    message p_delivermsg;       /* Message for this process if MF_DELIVERMSG */
```

IPC - messages

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    ...
    struct p_vmrequest;            /* request to VM                    */
```

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    ...
    struct p_vmrequest;           /* request to VM                          */

    sigset_t p_pending;           /* bit map for pending kernel signals     */
    u64_t p_signal_received;
```

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

```
struct proc {
    ...
    struct p_vmrequest;           /* request to VM                          */

    sigset_t p_pending;           /* bit map for pending kernel signals     */
    u64_t p_signal_received;

    char p_name[PROC_NAME_LEN];   /* name of the process, including \0      */
```

Processes in MINIX kernel

/usr/src/minix/kernel/proc.h

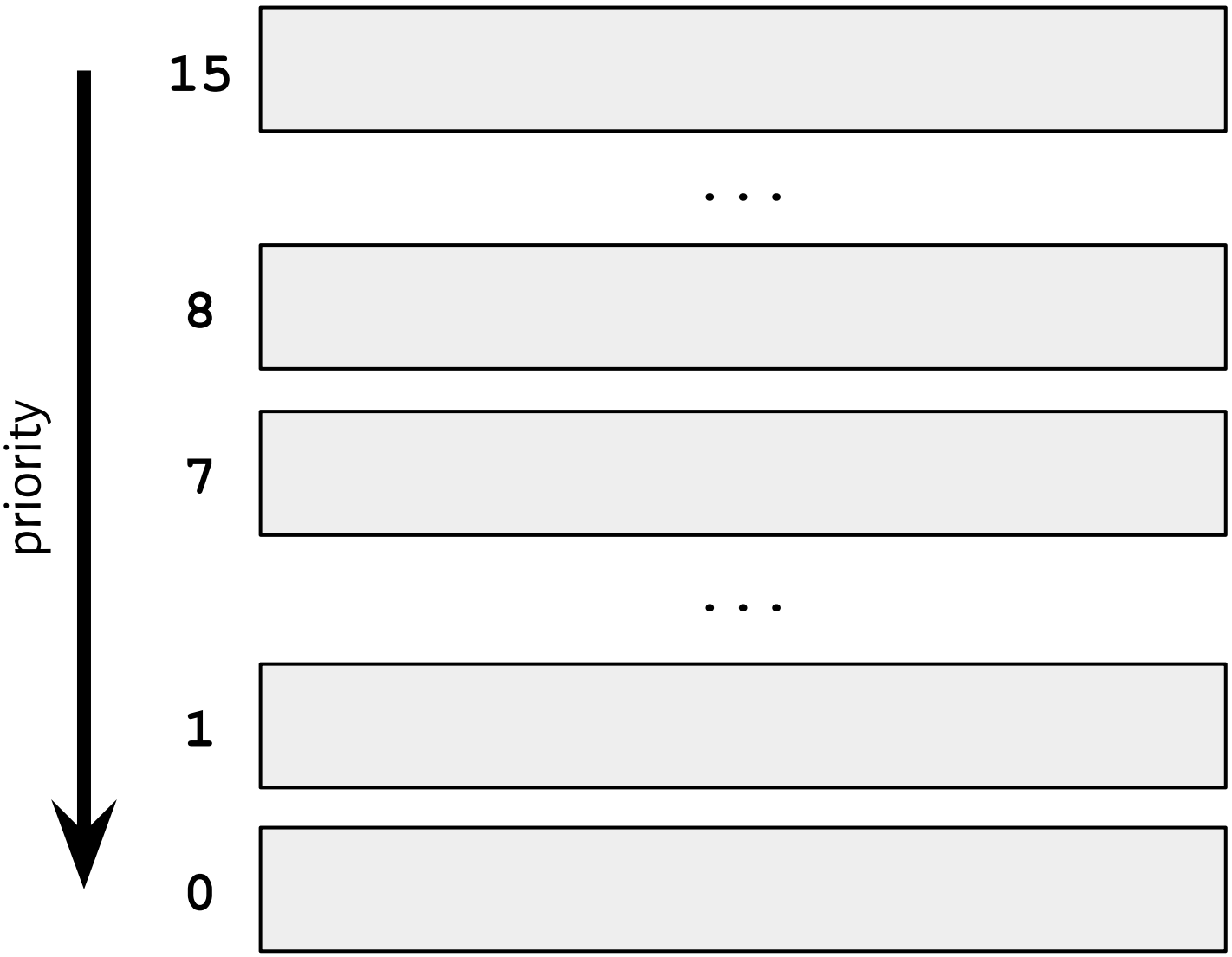
```
struct proc {
    ...
    struct p_vmrequest;             /* request to VM                          */

    sigset_t p_pending;             /* bit map for pending kernel signals     */
    u64_t p_signal_received;

    char p_name[PROC_NAME_LEN];    /* name of the process, including \0       */

    int p_found;                    /* consistency checking variables         */
    int p_magic;                    /* check validity of proc pointers
*/
```

Processes in MINIX kernel

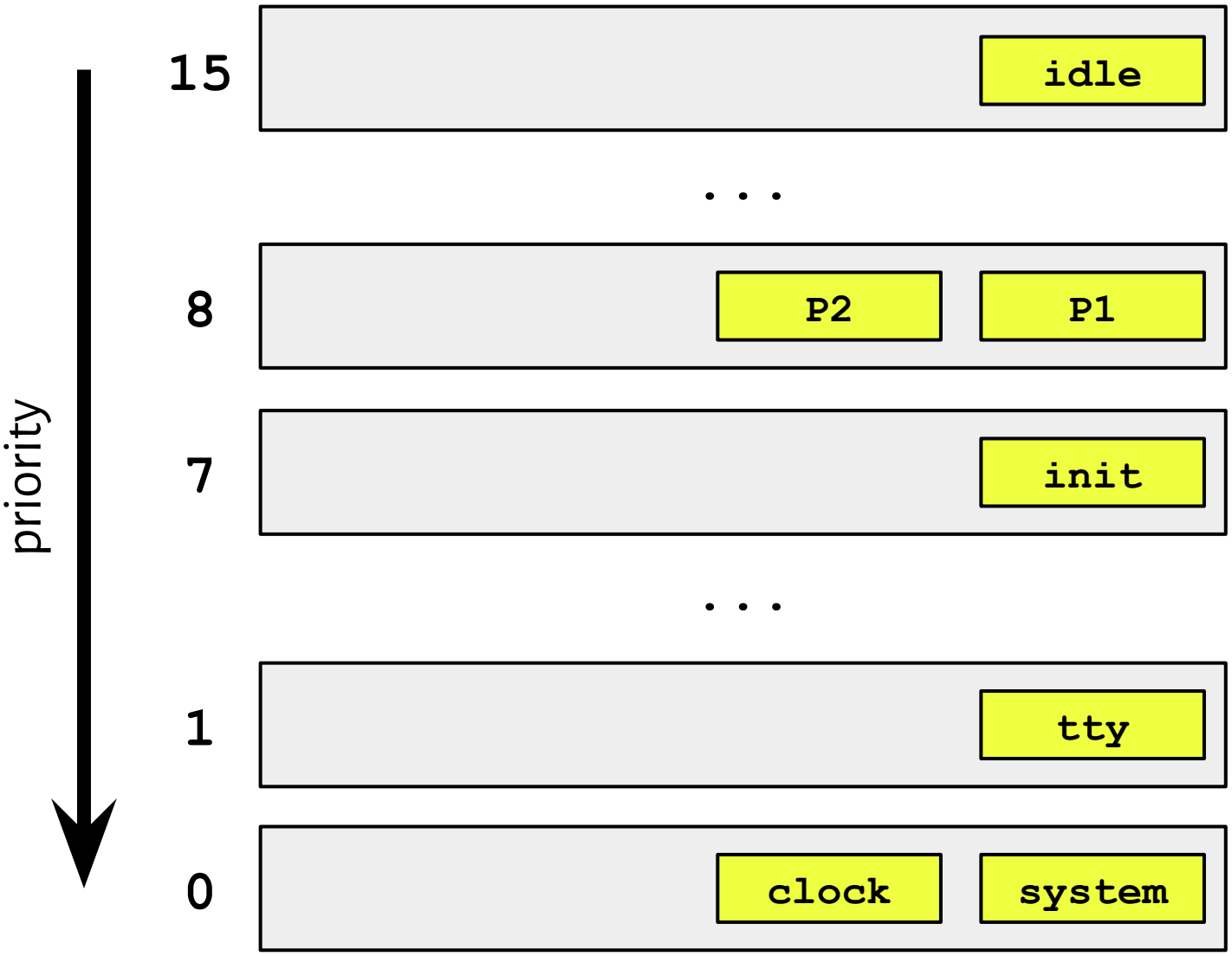
/usr/src/minix/kernel/proc.h

```
struct proc {
    ...
    struct p_vmrequest;           /* request to VM                          */

    sigset_t p_pending;           /* bit map for pending kernel signals     */
    u64_t p_signal_received;

    char p_name[PROC_NAME_LEN];   /* name of the process, including \0      */

    int p_found;                  /* consistency checking variables         */
    int p_magic;                  /* check validity of proc pointers        */

    struct { reg_t r1, r2, r3; } p_defer;
                                  /* if MF_SC_DEFER is set, this struct is valid
                                   * and contains the do_ipc() arguments
                                   * that are still to be executed          */
}
```

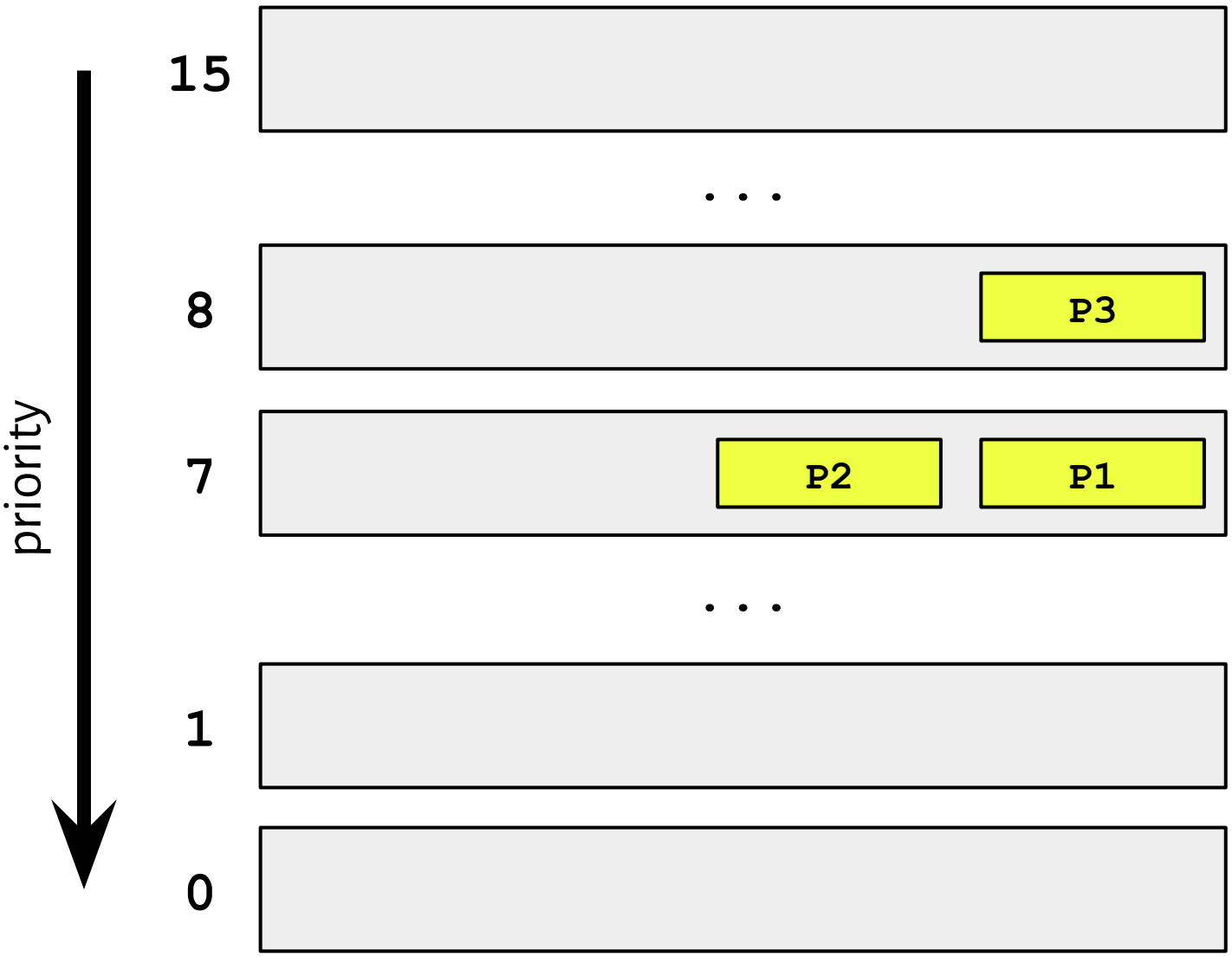
Processes scheduling



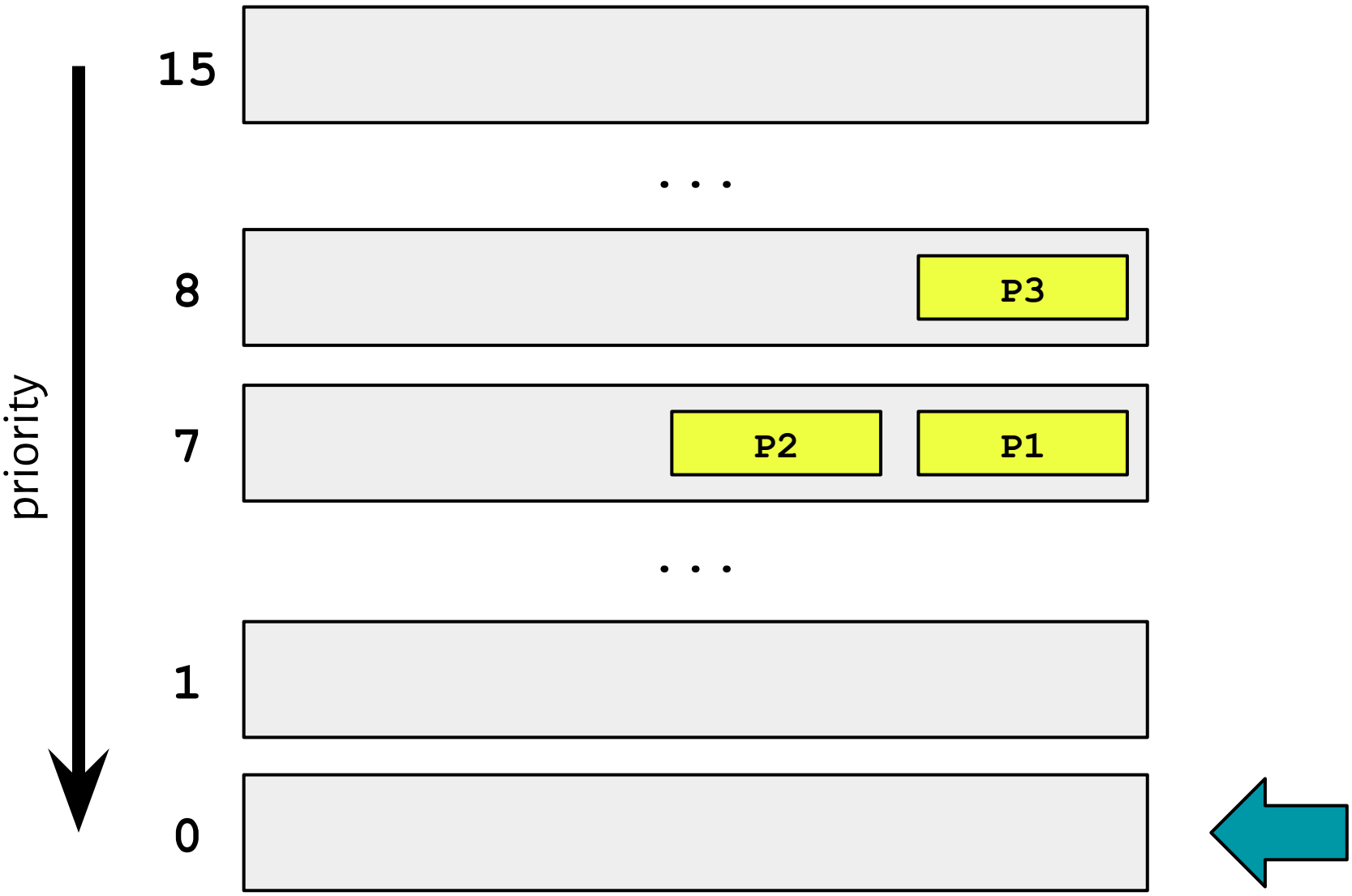
Processes scheduling



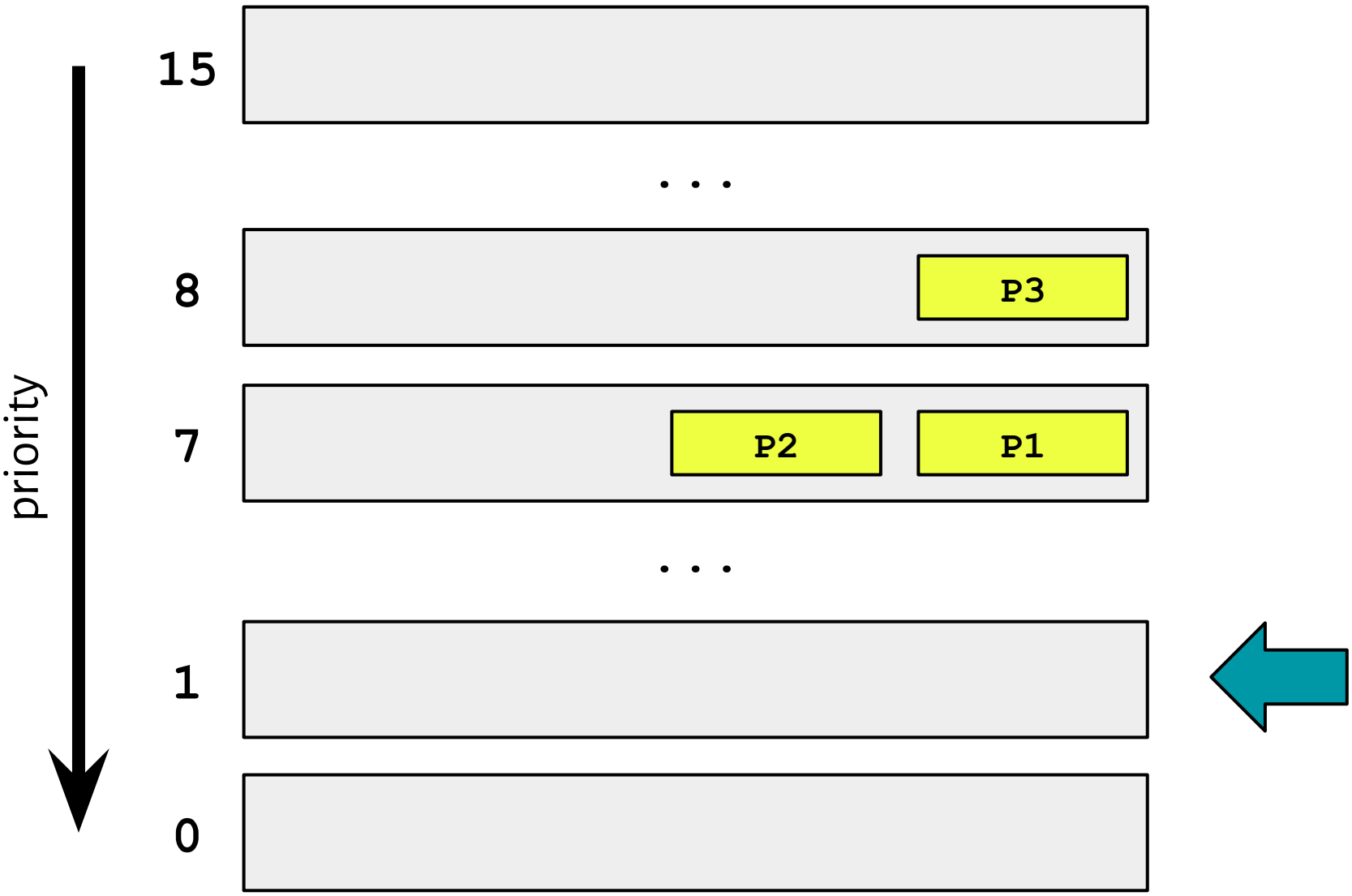
Processes scheduling



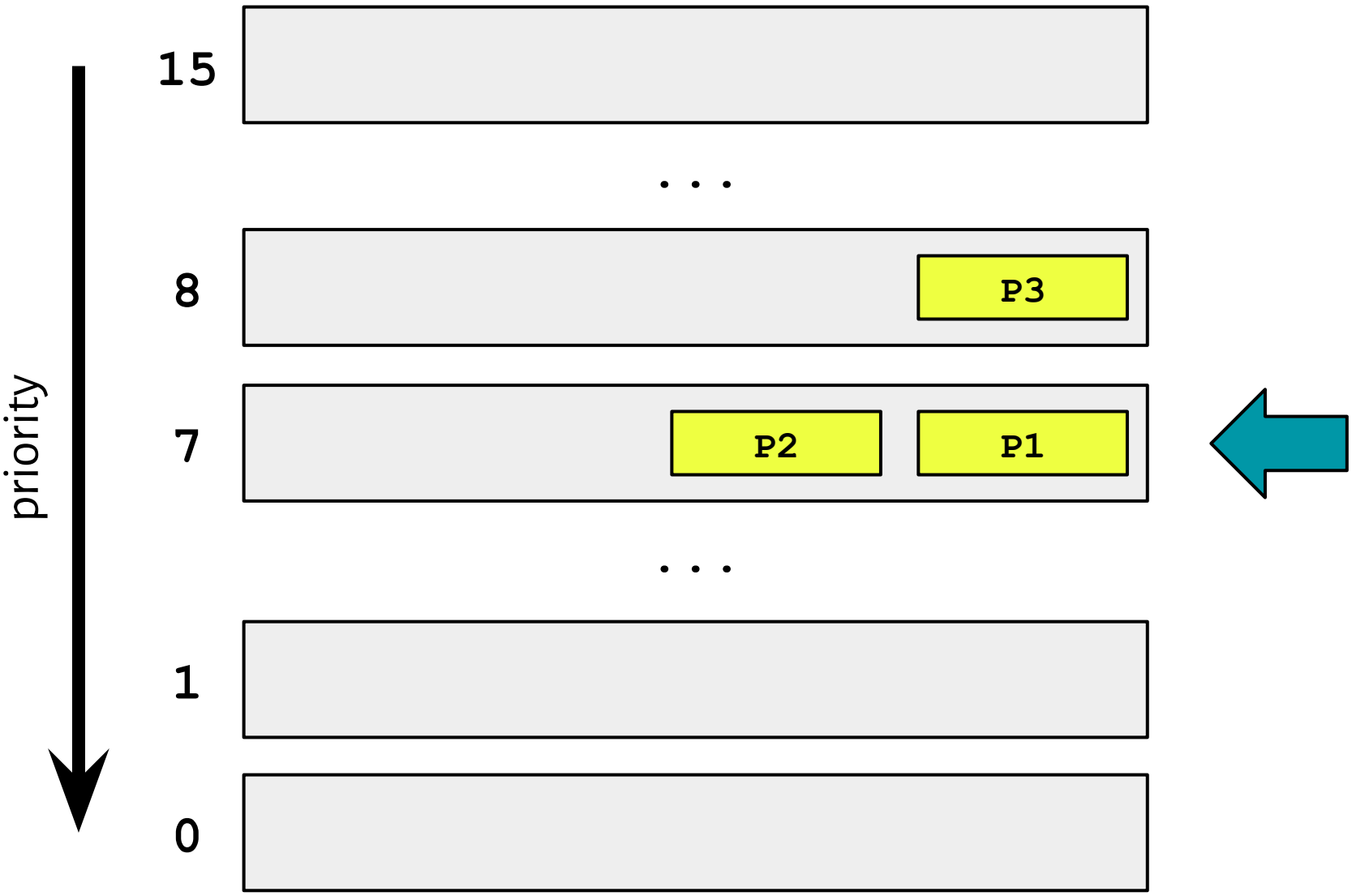
Processes scheduling - kernel



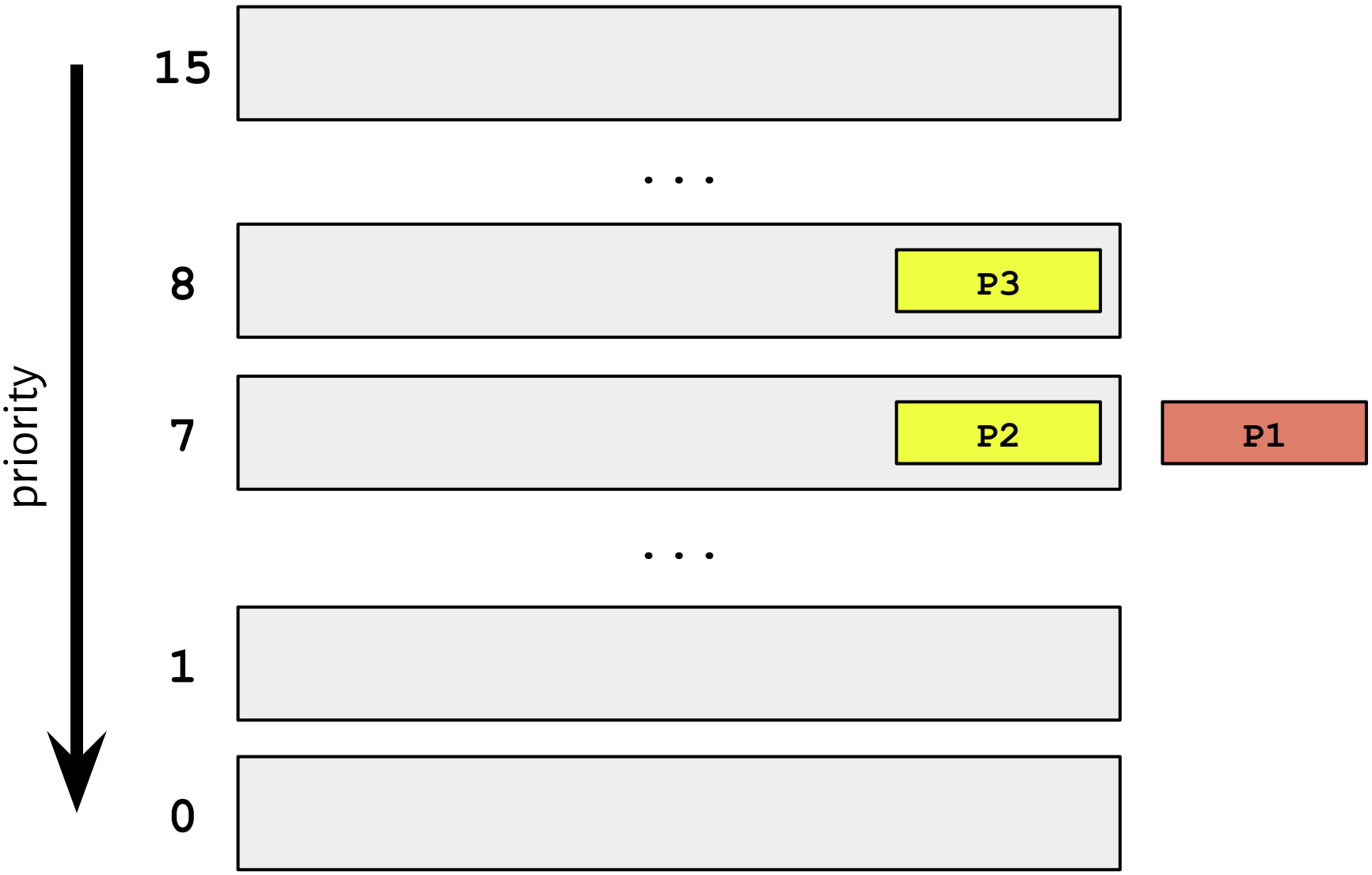
Processes scheduling - kernel



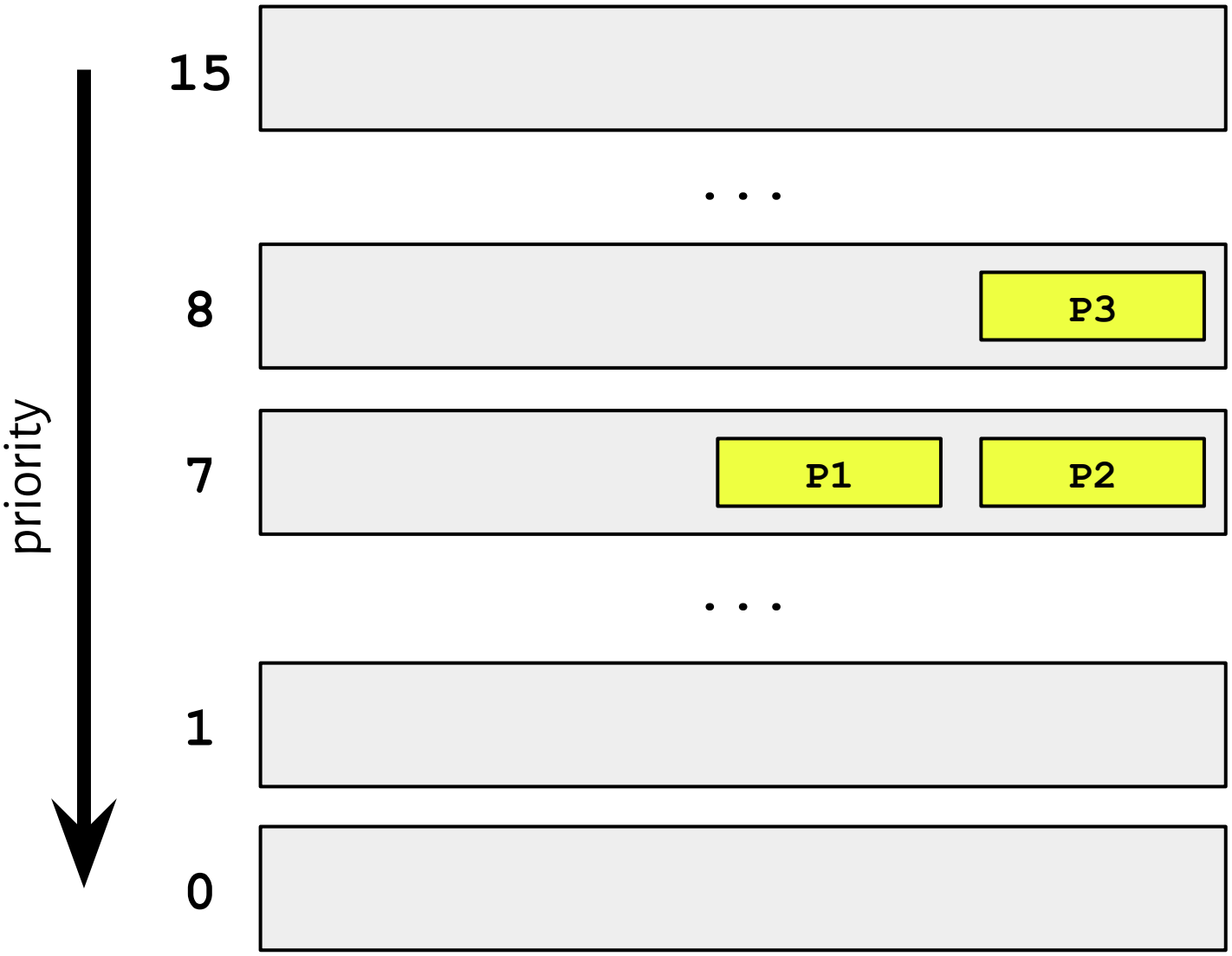
Processes scheduling - kernel



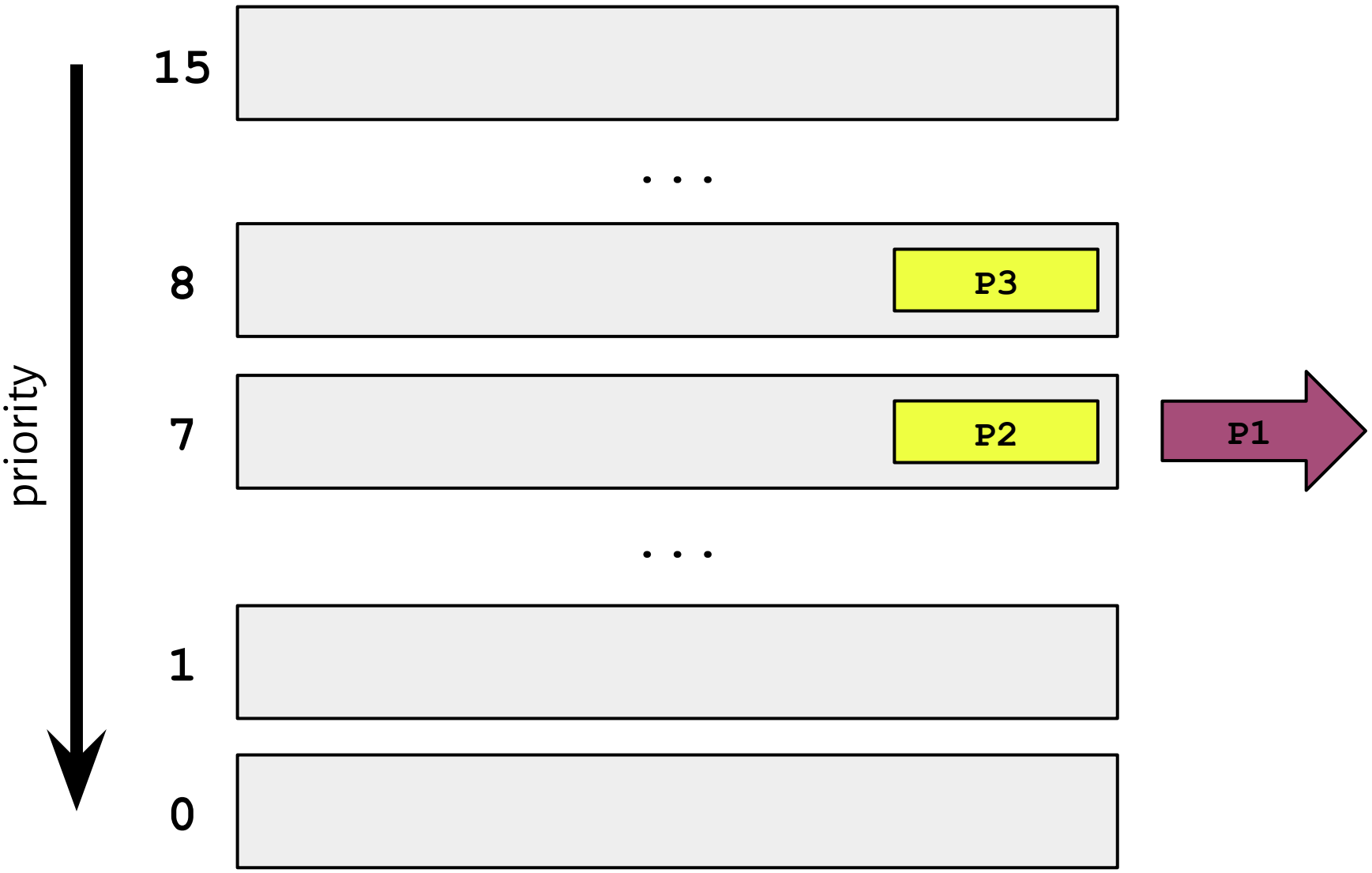
Processes scheduling - kernel



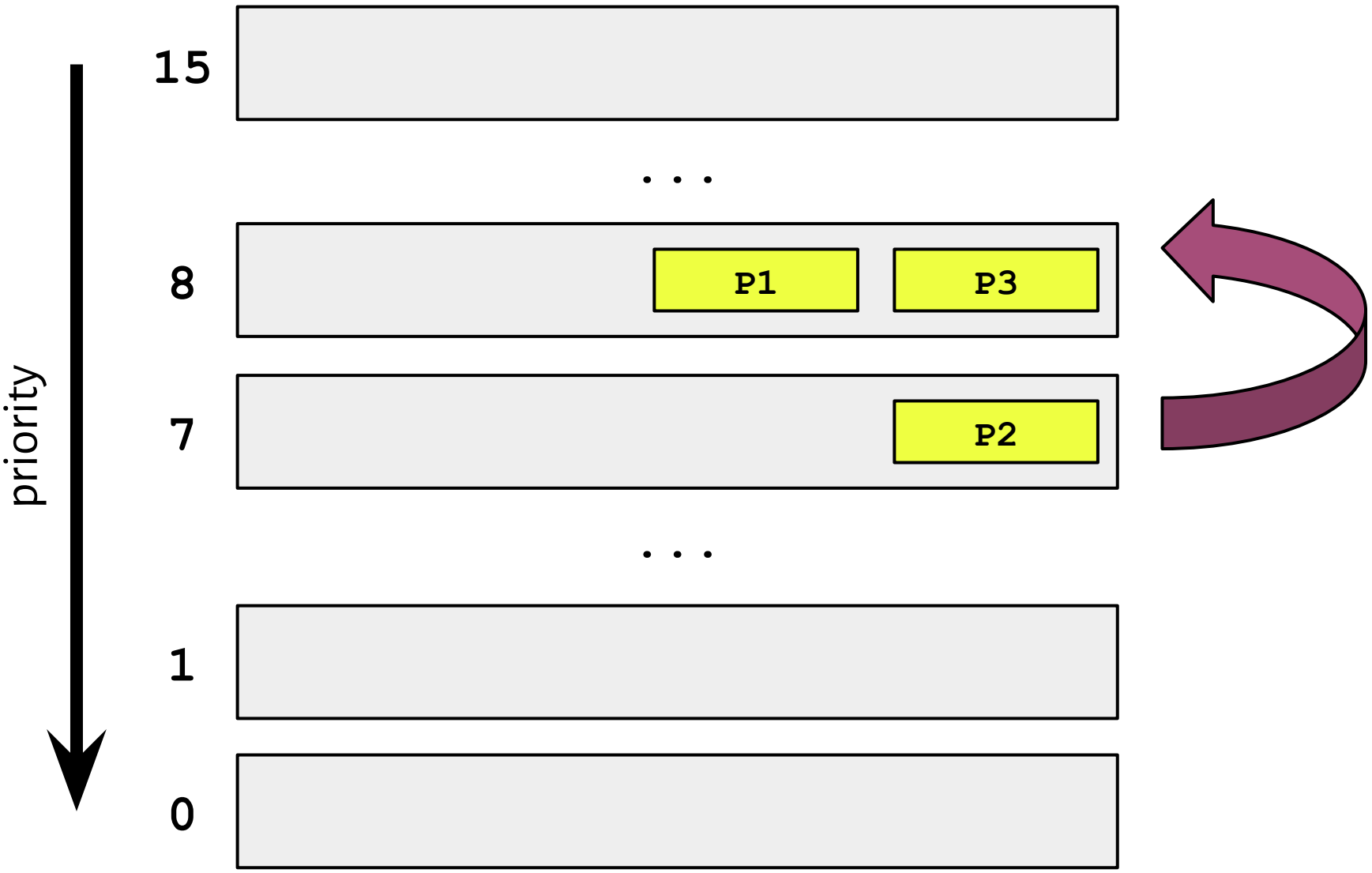
Processes scheduling - kernel



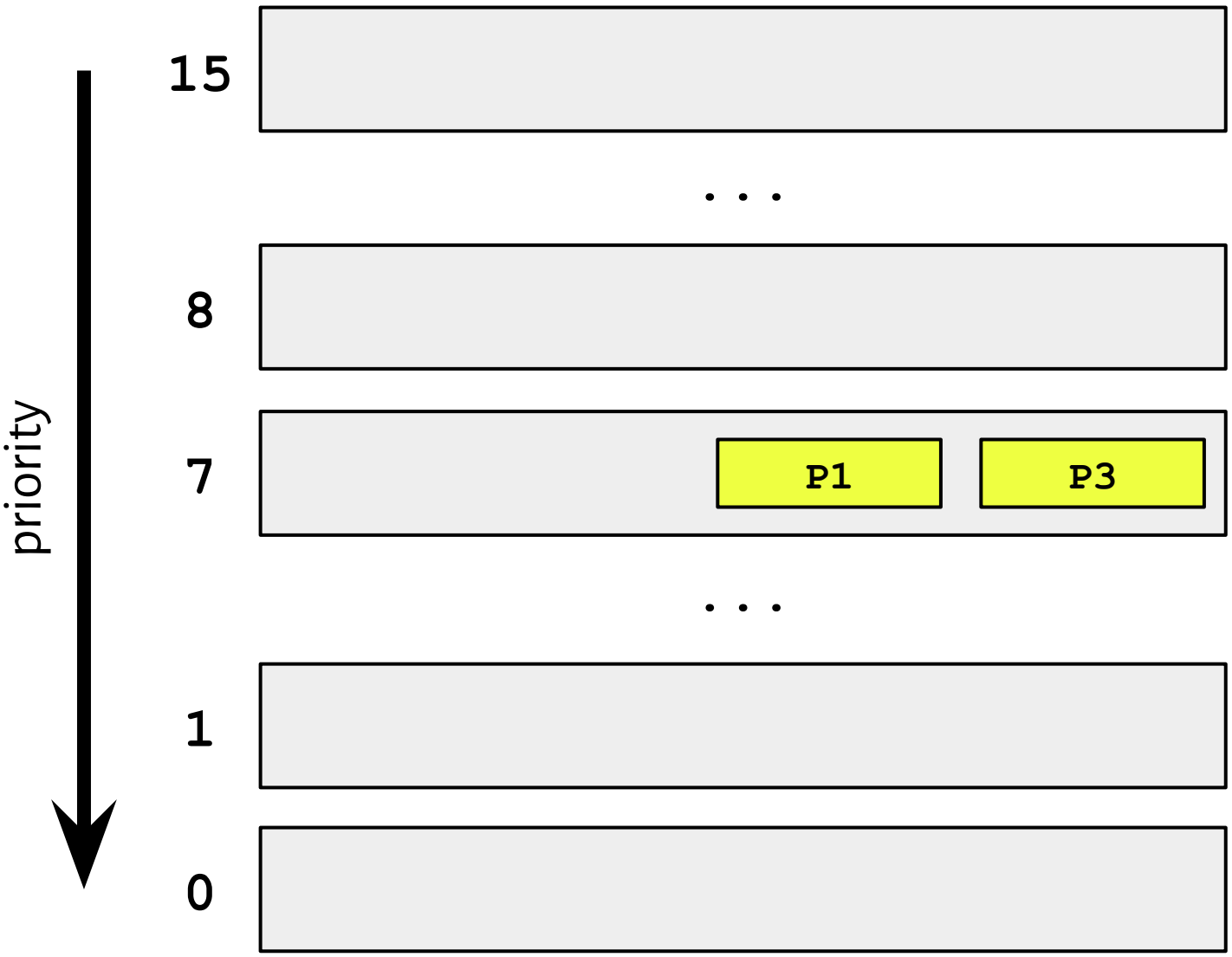
Processes scheduling - sched



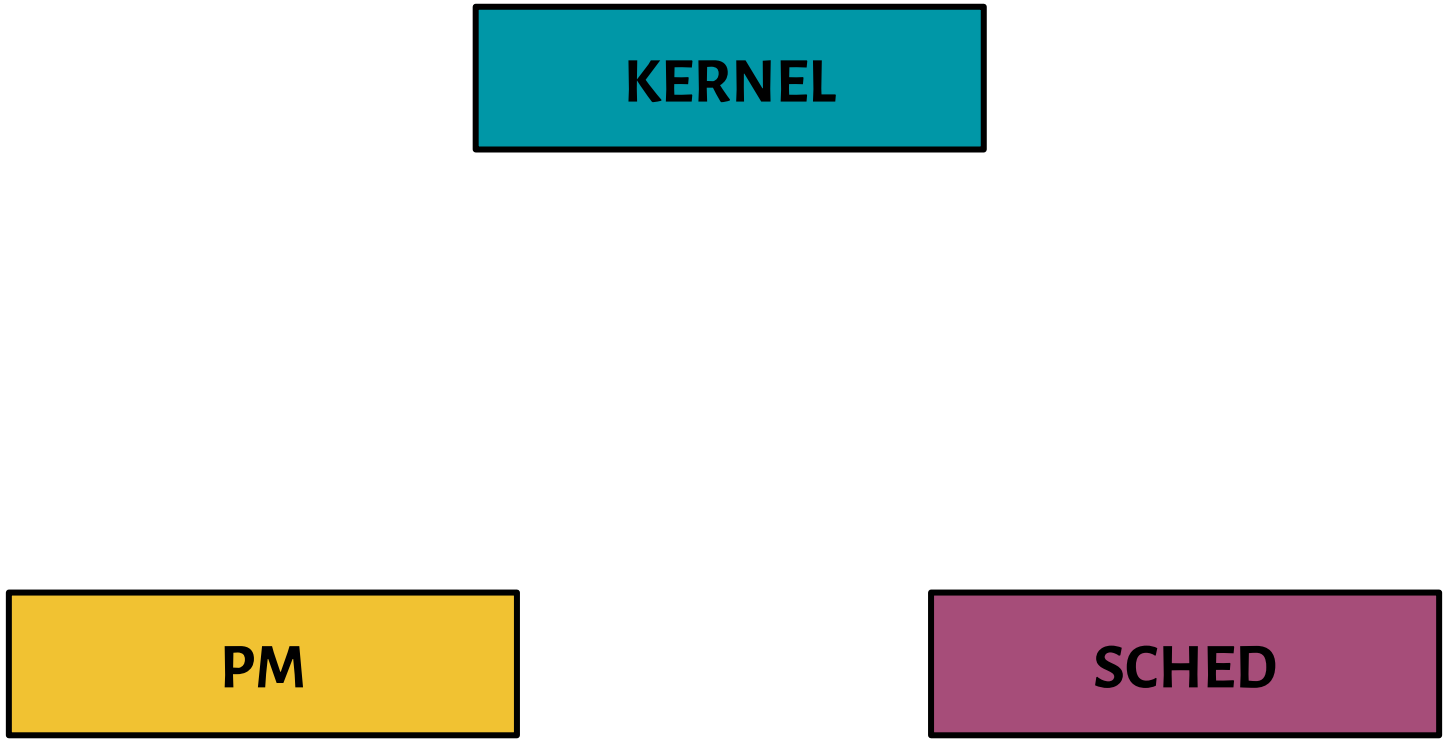
Processes scheduling - sched



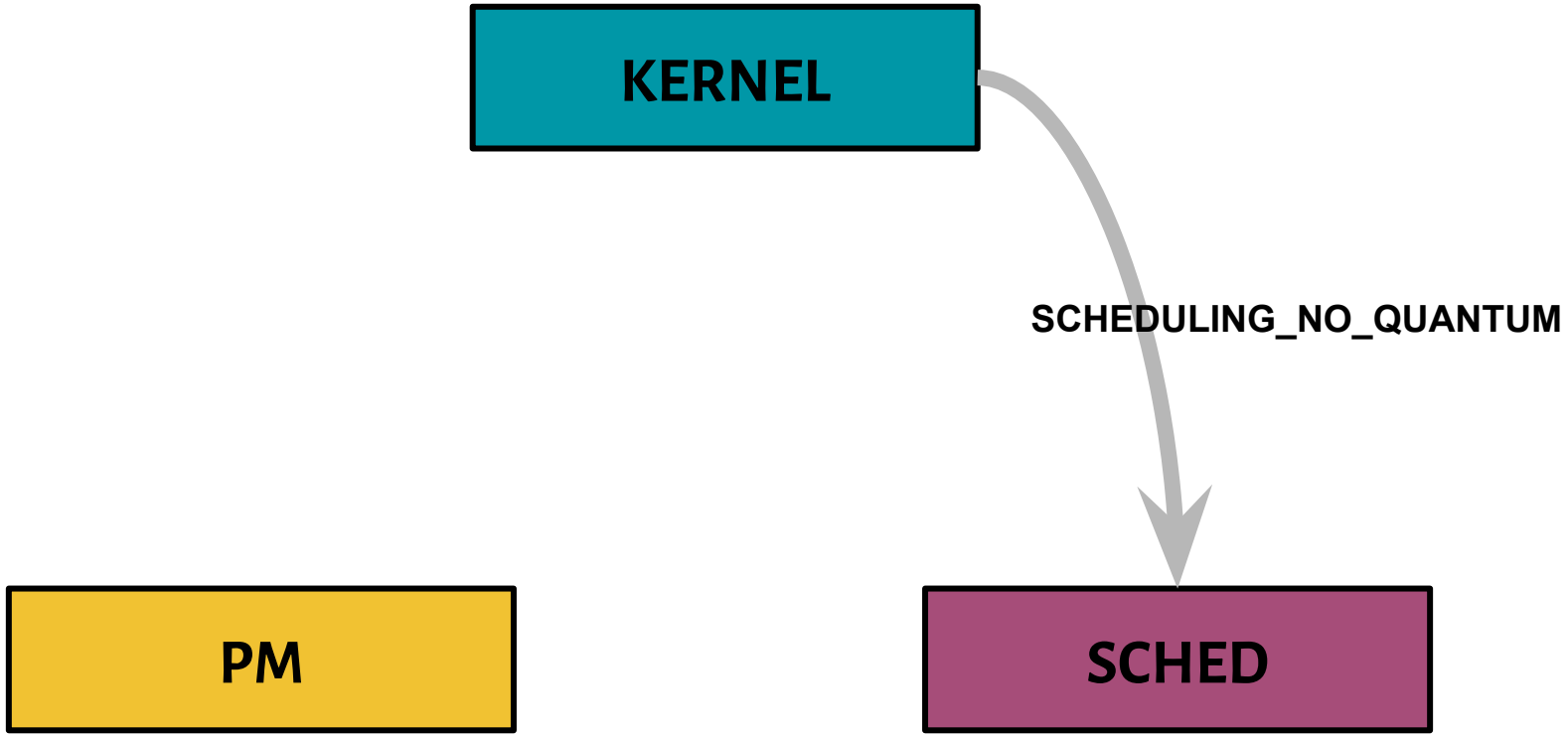
Processes scheduling - sched



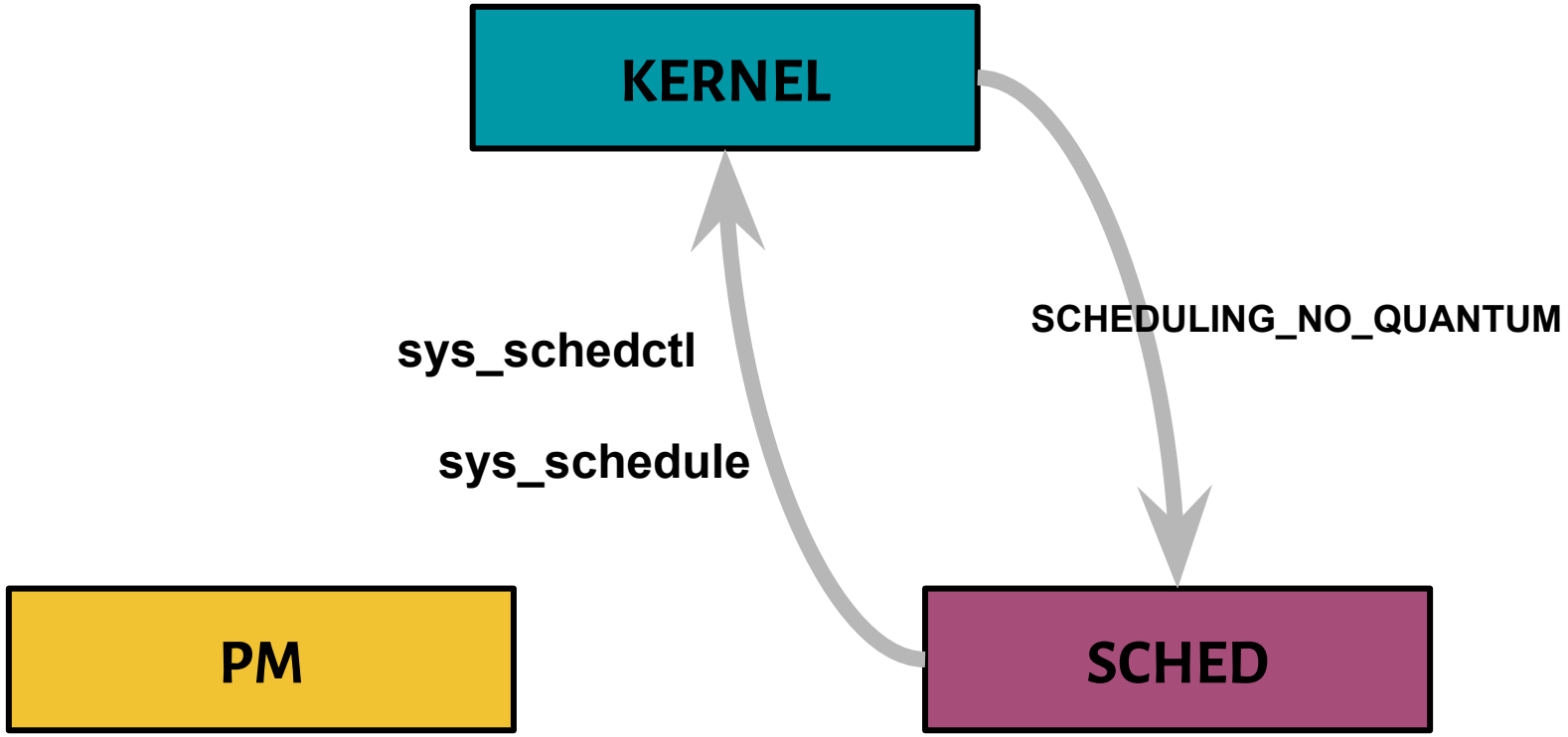
Processes scheduling



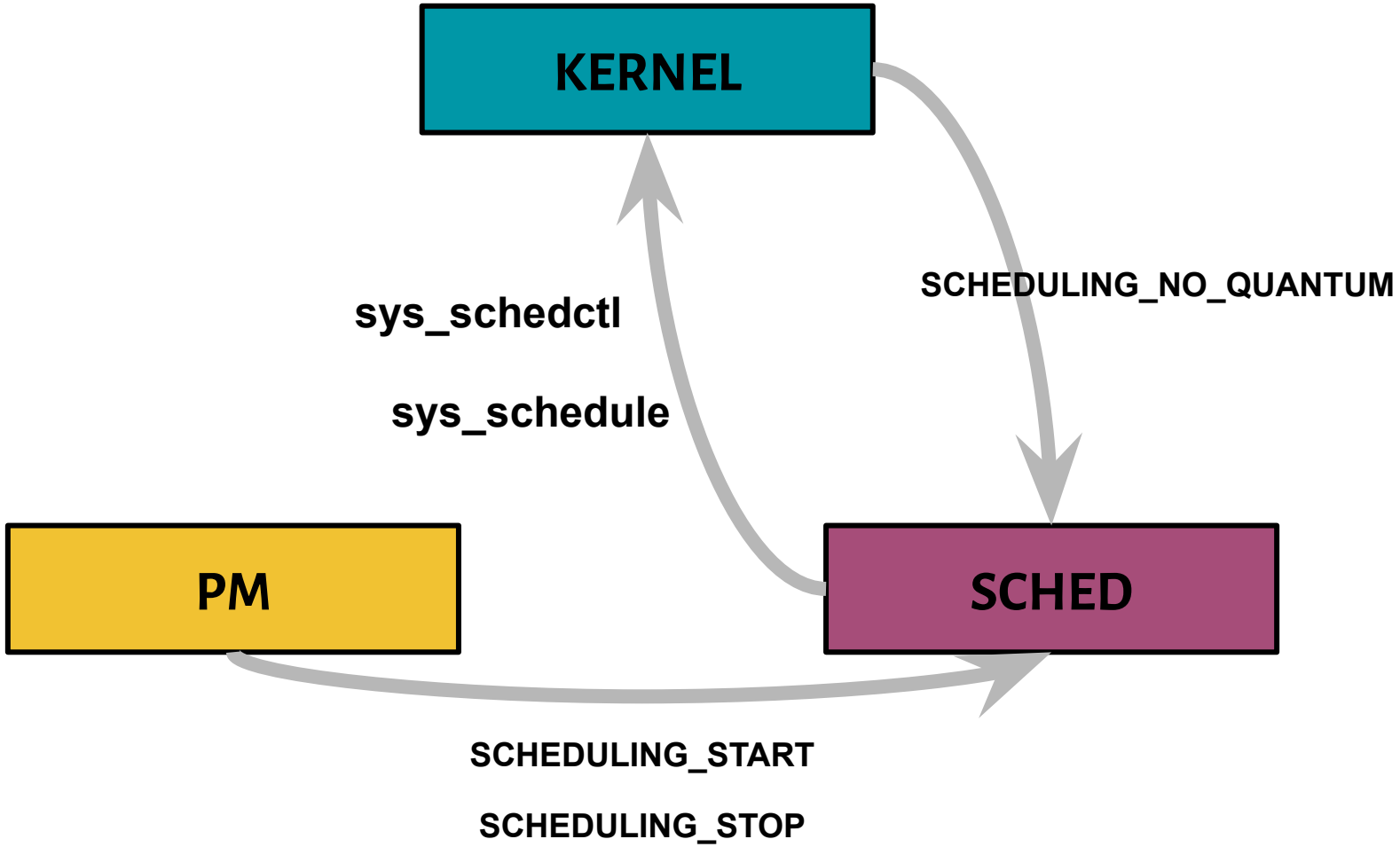
Processes scheduling



Processes scheduling



Processes scheduling



Zadanie H3

```
# cd /usr/src/minix/servers/sched
# make; make install
# cd /usr/src/releasetools
# make do-hdboot
# reboot
```