



Departament d'Arquitectura  
de Computadors

UNIVERSITAT POLITÈCNICA DE CATALUNYA

# Conceptes Avançats de Sistemes Operatius

Facultat d'Informàtica de Barcelona  
Dept. d'Arquitectura de Computadors

Curs 2018/19 Q2

Abstraccions del Sistema Operatiu

UNIVERSITAT POLITÈCNICA DE CATALUNYA  
BARCELONATECH

Facultat d'Informàtica de Barcelona

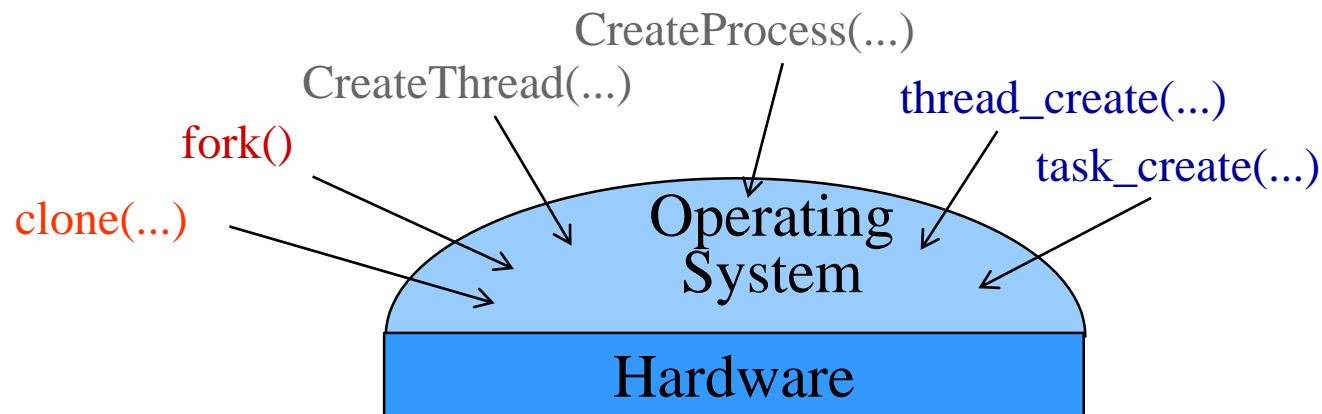


# Índex

- Què és un Sistema Operatiu?
- Abstraccions del sistema operatiu
- Fluxos de sistema
  - Mach
  - Linux
- Fluxos d'usuari: Pthreads
- Gestió de memòria

# Sistema Operatiu

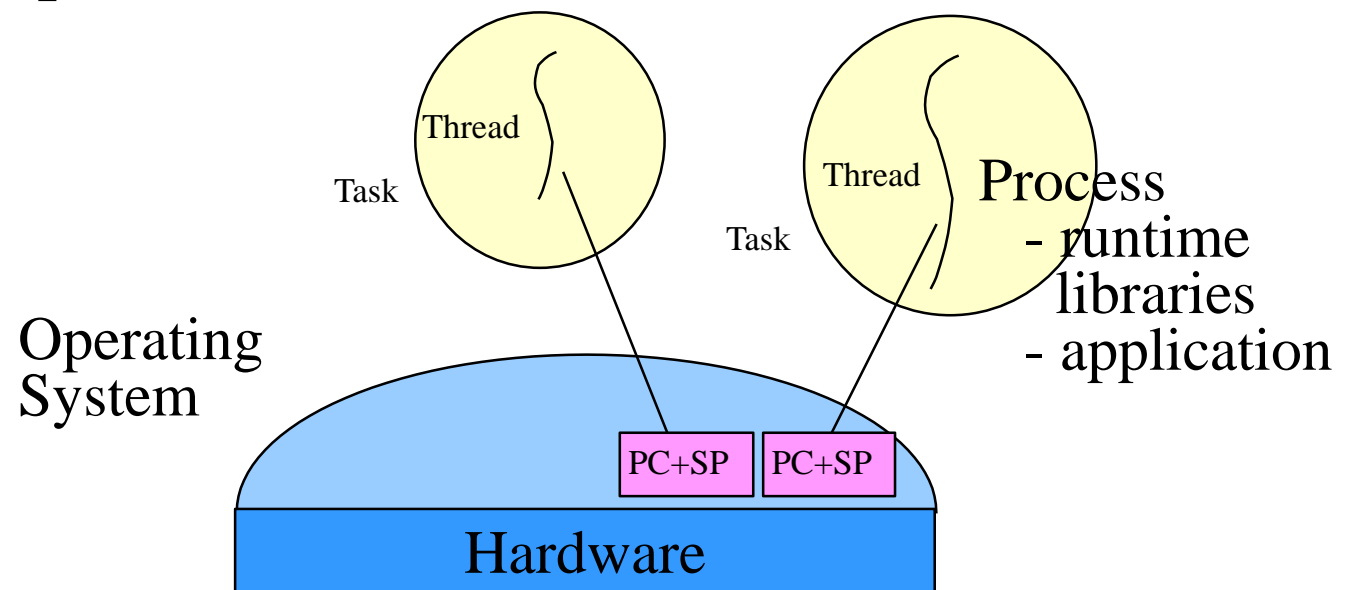
- Intermediari entre l'usuari i la màquina
- Proporciona un entorn d'execució entre convenient i eficient per executar programes
  - Servidors / sobre-taula / portàtils / mòbils
- Gestiona la màquina
- Ofereix protecció entre usuaris



# Abstraccions del Sistema Operatiu

## •Entorn d'execució

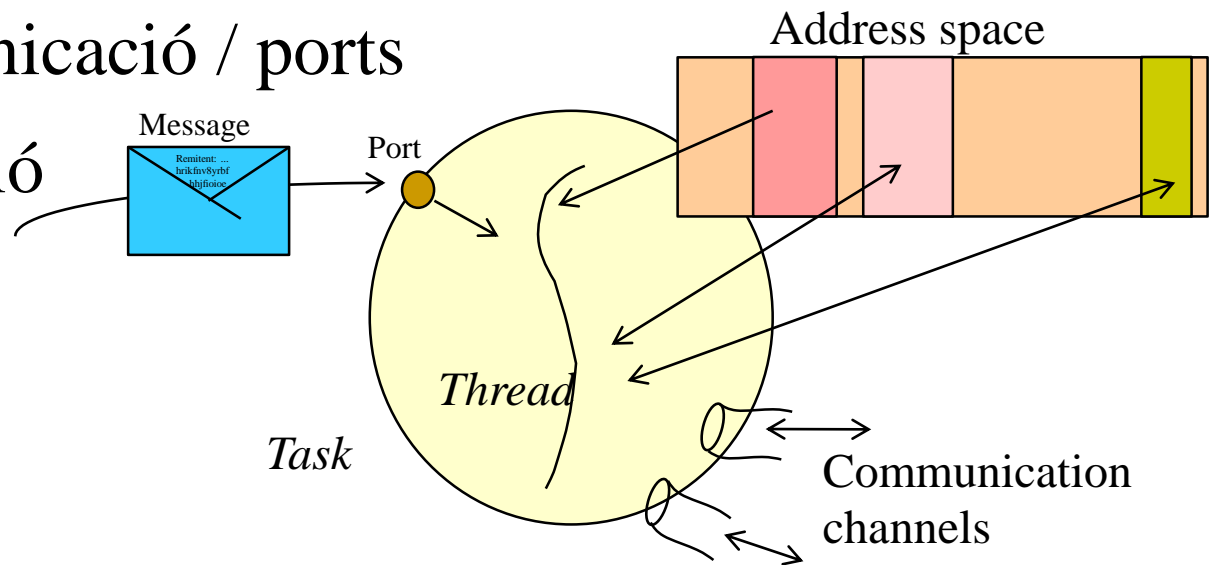
- Hardware
- Sistema operatiu
- Llibreries de suport
- Aplicacions



# Abstraccions del Sistema Operatiu

## • Definició de procés

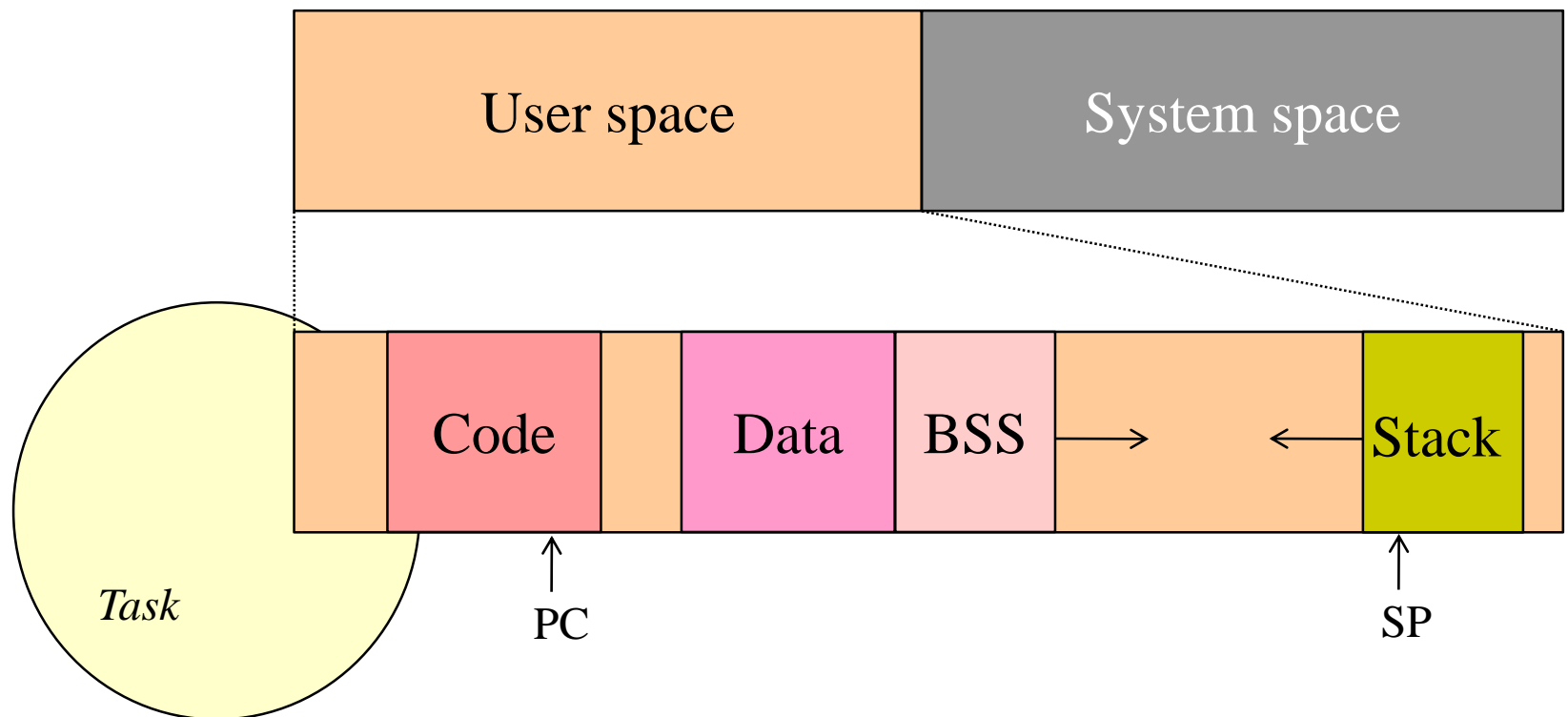
- Unitat d'assignació de recursos
- Entorn d'execució, del qual formen part...
  - Un (o més) espais d'adreces
  - Canals de comunicació / ports
  - Fluxos d'execució



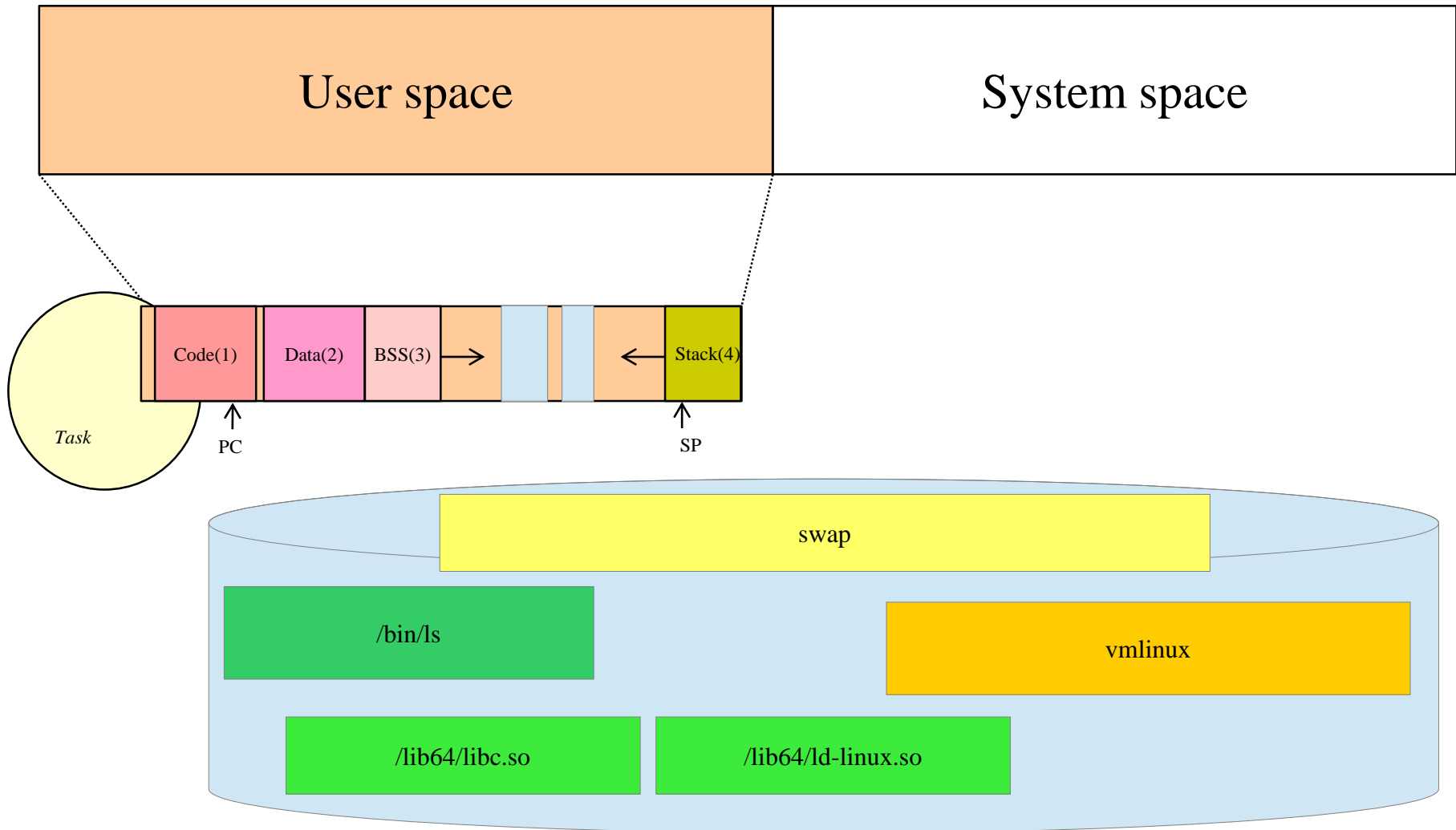
# Abstraccions del Sistema Operatiu

## •Espai d'adreces

- Espai virtual adreçable, conté codi, dades (data i BSS) i pila

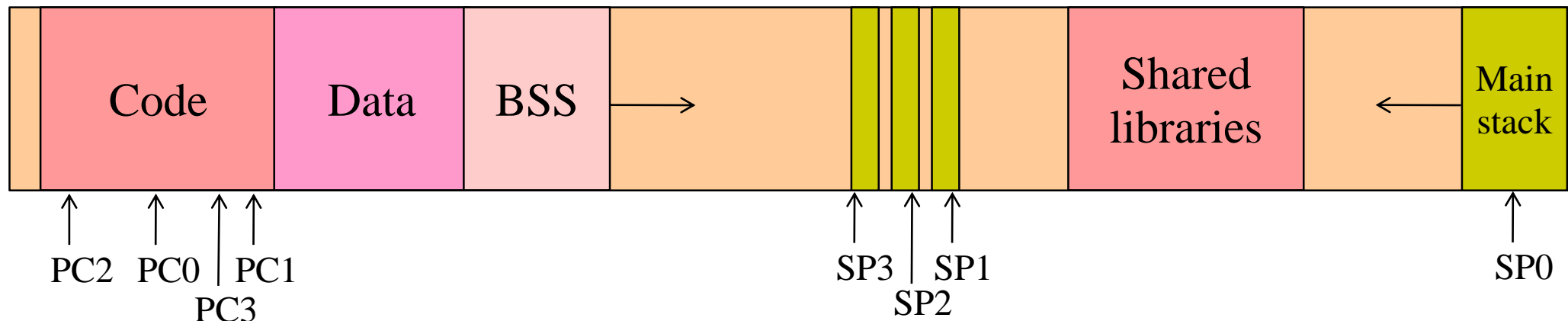


# Espai d'adreces



# Abstraccions del Sistema Operatiu

• Amb fluxos, l'estructura genèrica d'un espai d'adreces és:



- La pila “principal” creix automàticament, fins a:
  - `ulimit -s` (exemple: 8Mb)
- Les piles dels altres fluxos no creixen
  - Es podrien fer créixer atenent al signal `SIGSEGV`



# Abstraccions del Sistema Operatiu

• Com podem “veure” l'espai d'adreces dels processos?

- ps, top...
- /proc/<pid>/maps
- ... see also “ldd”

```
bash-4.2$ more /proc/2782/maps
00400000-00401000 r-xp 00000000 08:07 1854638
00600000-00601000 rw-p 00000000 08:07 1854638
7fd25c097000-7fd25c256000 r-xp 00000000 08:09 932929
7fd25c256000-7fd25c455000 ---p 001bf000 08:09 932929
7fd25c455000-7fd25c459000 r--p 001be000 08:09 932929
7fd25c459000-7fd25c45b000 rw-p 001c2000 08:09 932929
7fd25c45b000-7fd25c460000 rw-p 00000000 00:00 0
7fd25c460000-7fd25c561000 r-xp 00000000 08:09 933316
7fd25c561000-7fd25c760000 ---p 00101000 08:09 933316
7fd25c760000-7fd25c761000 r--p 00100000 08:09 933316
7fd25c761000-7fd25c762000 rw-p 00101000 08:09 933316
7fd25c762000-7fd25c785000 r-xp 00000000 08:09 942068
7fd25c94d000-7fd25c950000 rw-p 00000000 00:00 0
7fd25c983000-7fd25c985000 rw-p 00000000 00:00 0
7fd25c985000-7fd25c986000 r--p 00023000 08:09 942068
7fd25c986000-7fd25c988000 rw-p 00024000 08:09 942068
7fff02f2b000-7fff02f4c000 rw-p 00000000 00:00 0
7fff02fa6000-7fff02fa7000 r-xp 00000000 00:00 0
ffffffffffff600000-ffffffffffff601000 r-xp 00000000 00:00 0
```

```
# binari dynamic
/mnt/home/xavim/CASO14152q/slides/hello
/mnt/home/xavim/CASO14152q/slides/hello
/lib64/libc-2.17.so
/lib64/libc-2.17.so
/lib64/libc-2.17.so
/lib64/libc-2.17.so
/lib64/libm-2.17.so
/lib64/libm-2.17.so
/lib64/libm-2.17.so
/lib64/libm-2.17.so
/lib64/ld-2.17.so
/lib64/ld-2.17.so
/lib64/ld-2.17.so
[stack]
[vdso]
[vsyscall]
```

# Abstraccions del Sistema Operatiu

```
bash-4.2$ more /proc/2801/maps
00400000-004c4000 r-xp 00000000 08:07 1854639
006c3000-006c9000 rw-p 000c3000 08:07 1854639
006c9000-006cb000 rw-p 00000000 00:00 0
01448000-0146b000 rw-p 00000000 00:00 0
7f4f246ad000-7f4f246ae000 rw-p 00000000 00:00 0
7ffffaa919000-7ffffaa93a000 rw-p 00000000 00:00 0
7ffffaa9b9000-7ffffaa9ba000 r-xp 00000000 00:00 0
fffffffffff600000-fffffffffff601000 r-xp 00000000 00:00 0
```

```
# binari estàtic
/mnt/home/xavim/CASO14152q/slides/hello.static
/mnt/home/xavim/CASO14152q/slides/hello.static

[heap]

[stack]
[vdso]
[vsyscall]
```

– /proc/<pid>/mem

```
bash-4.2$ dd if=/proc/2890/mem ibs=1 skip=$((0x400000)) | od -x | head -10
dd: '/proc/2890/mem': cannot skip to specified offset
0000000 457f 464c 0102 0301 0000 0000 0000 0000
0000020 0002 003e 0001 0000 0fc0 0040 0000 0000
0000040 0040 0000 0000 0000 8c40 000c 0000 0000
0000060 0000 0000 0040 0038 0005 0040 0024 0021
0000100 0001 0000 0005 0000 0000 0000 0000 0000
0000120 0000 0040 0000 0000 0000 0040 0000 0000
0000140 39e0 000c 0000 0000 39e0 000c 0000 0000
0000160 0000 0020 0000 0000 0001 0000 0006 0000
0000200 39e0 000c 0000 0000 39e0 006c 0000 0000
0000220 39e0 006c 0000 0000 4a78 0000 0000 0000
```

-hello.static

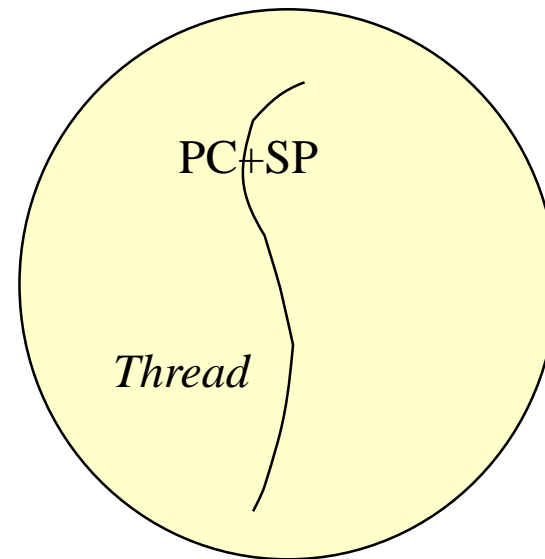
```
bash-4.2$ od -x hello.static |head -10
0000000 457f 464c 0102 0301 0000 0000 0000 0000
0000020 0002 003e 0001 0000 0fc0 0040 0000 0000
0000040 0040 0000 0000 0000 8c40 000c 0000 0000
0000060 0000 0000 0040 0038 0005 0040 0024 0021
0000100 0001 0000 0005 0000 0000 0000 0000 0000
0000120 0000 0040 0000 0000 0000 0040 0000 0000
0000140 39e0 000c 0000 0000 39e0 000c 0000 0000
0000160 0000 0020 0000 0000 0001 0000 0006 0000
0000200 39e0 000c 0000 0000 39e0 006c 0000 0000
0000220 39e0 006c 0000 0000 4a78 0000 0000 0000
```

# Abstraccions del Sistema Operatiu

## •Flux d'execució

- Mínim: PC + SP
- S'afegeixen: els registres del processador
  - Registres de control, flags
  - Enters
  - Coma flotant
  - Extensions multimèdia
  - Extensions SIMD

*Task*

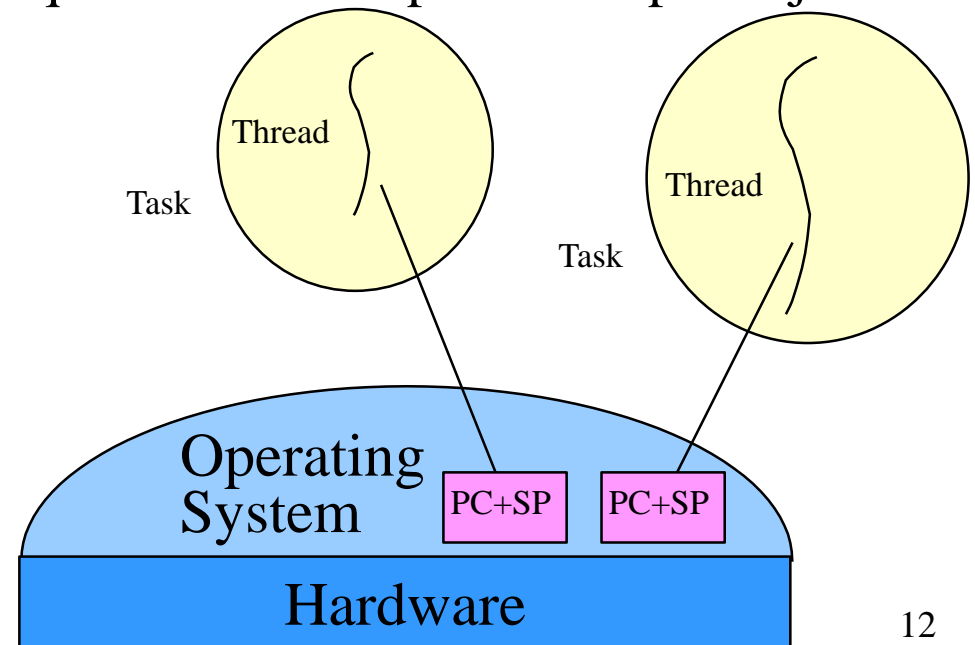


# Abstraccions del Sistema Operatiu

## • Propietats dels fluxos

### – De sistema i d'usuari

- Prioritat: importància
- Quantum: quantitat màxima de temps que haurà de passar mentre el flux s'estigui executant, abans que el sistema operatiu es planteji canviar de context

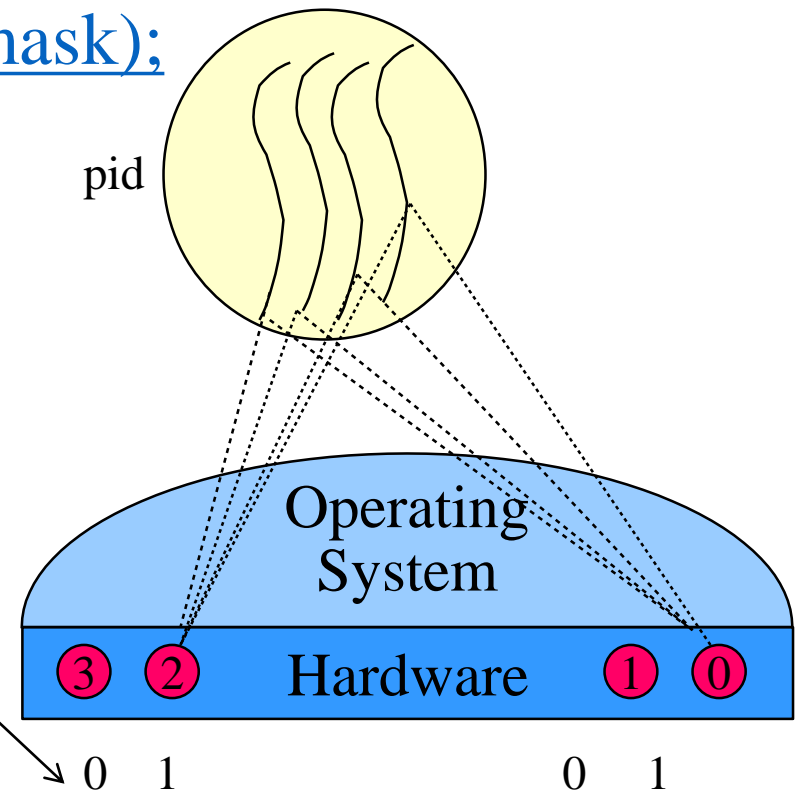


# Fluxos de sistema

- Oferts per la interfície del sistema operatiu
  - Preempció: interrupció de rellotge
  - Assignables als processadors (cores) de la màquina
    - [sched\\_setaffinity\(pid, size, mask\);](#)

```
cpu_set_t mask;  
CPU_ZERO(&mask); // 0000 (0x00)
```

```
CPU_SET(0, &mask); // 0001 (0x01)  
CPU_SET(2, &mask); // 0101 (0x05)  
sched_setaffinity(pid, 4, &mask);
```



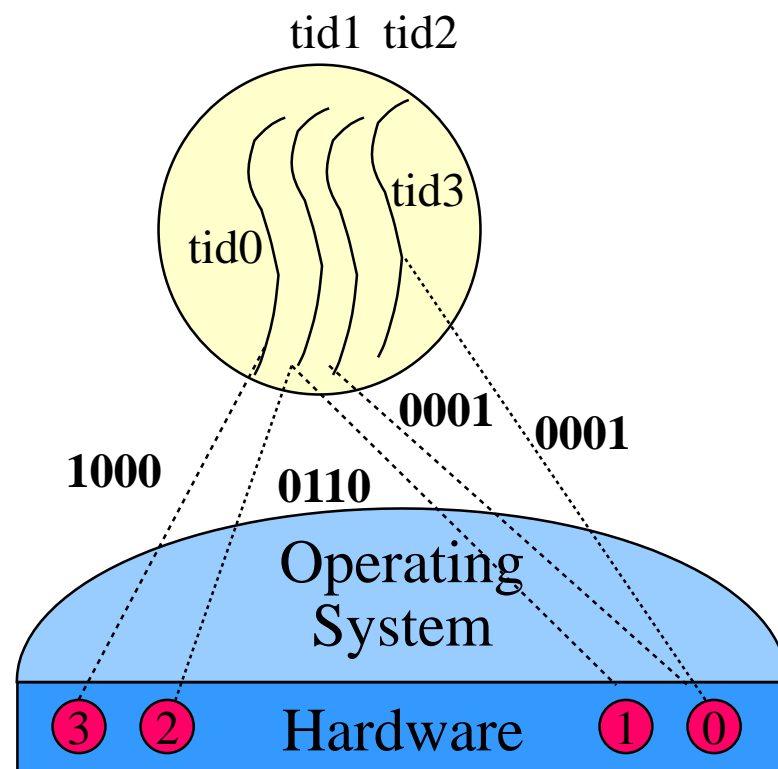
# Fluxos de sistema

## • Assignables individualment als processadors (cores) de la màquina

```
cpu_set_t mask;  
CPU_ZERO(&mask); // 0000 (0x00)  
CPU_SET(3, &mask); // 1000 (0x08)  
sched_setaffinity (tid0, 4, &mask);
```

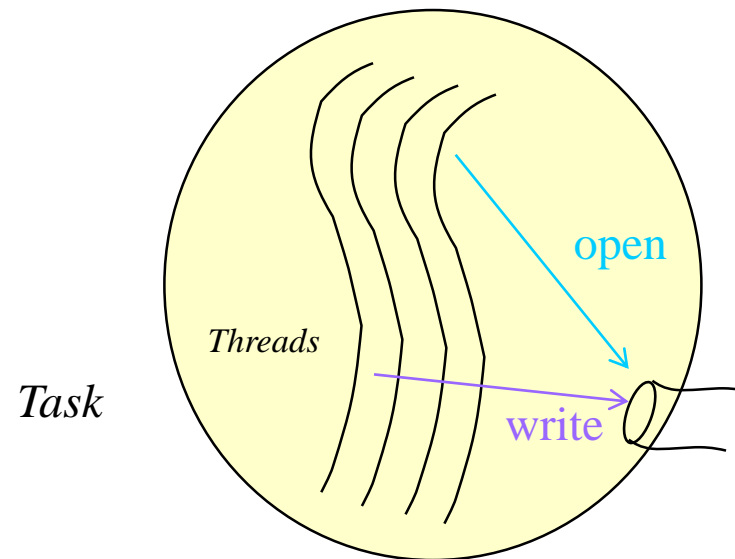
```
CPU_ZERO(&mask); // 0000 (0x00)  
CPU_SET(2, &mask); // 0100 (0x04)  
CPU_SET(1, &mask); // 0110 (0x06)  
sched_setaffinity (tid1, 4, &mask);
```

```
CPU_ZERO(&mask); // 0000 (0x00)  
CPU_SET(0, &mask); // 0001 (0x01)  
sched_setaffinity (tid2, 4, &mask);  
sched_setaffinity (tid3, 4, &mask);
```



# Fluxos de sistema

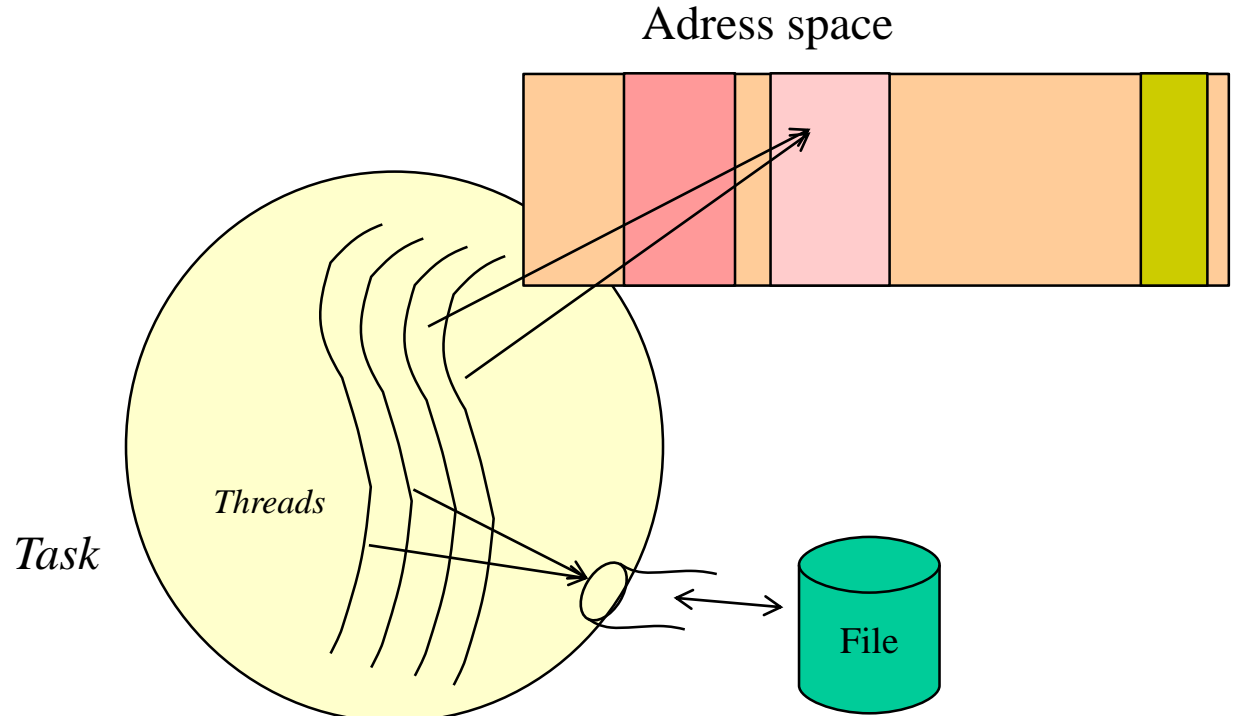
- Comparteixen tots els recursos del procés
- Poden demanar nous recursos pel procés
- Poden alliberar recursos



# Fluxos de sistema

•Sincronització necessària per:

- Accedir a dades compartides (usuari/sistema)
- Accedir a recursos compartits (sistema)

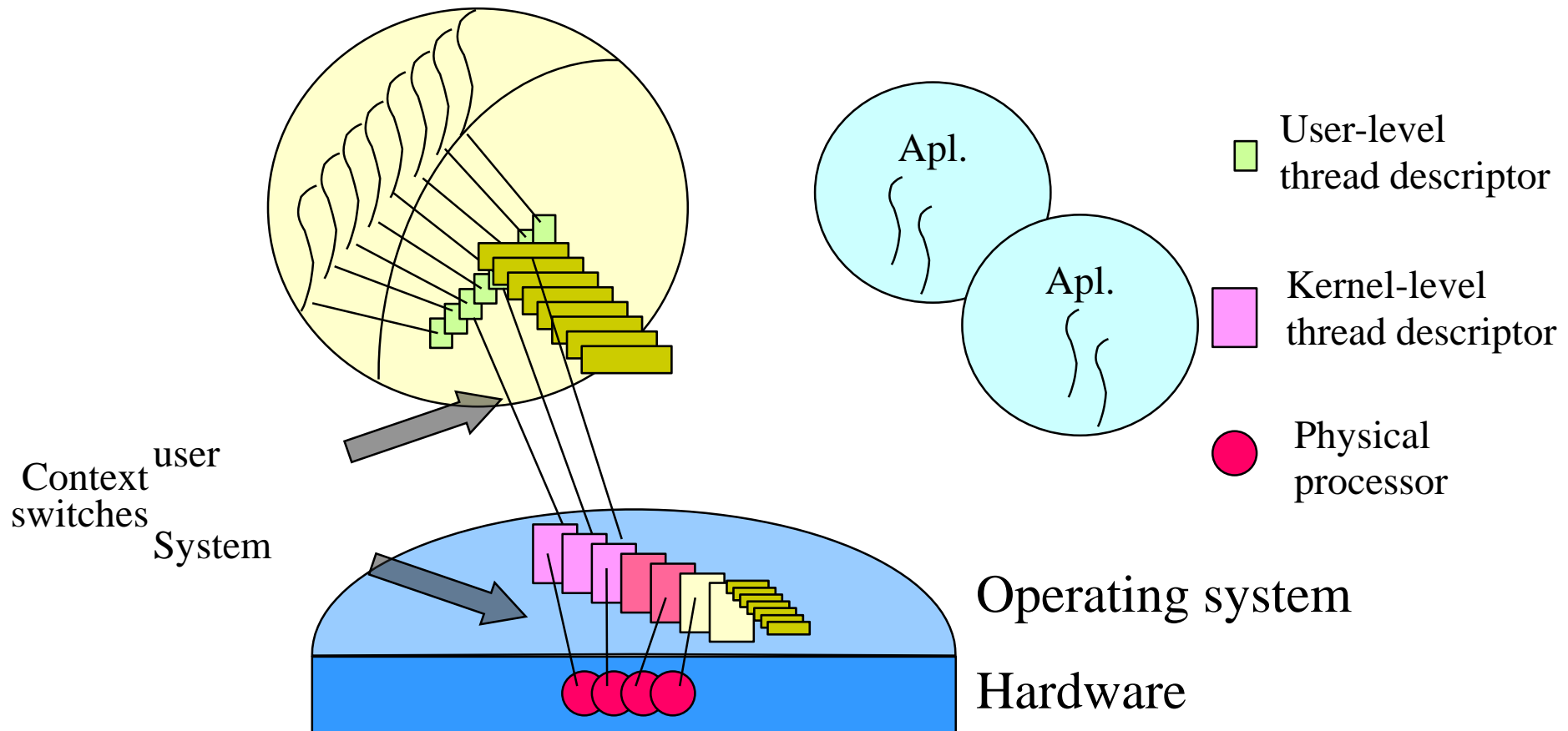




# Nivells de fluxos

## •Entorn híbrid (N:M)

- Gestionat per una llibreria de suport (Pthreads)
- Processadors  $\leq$  fluxos de sistema  $\leq$  fluxos d'usuari



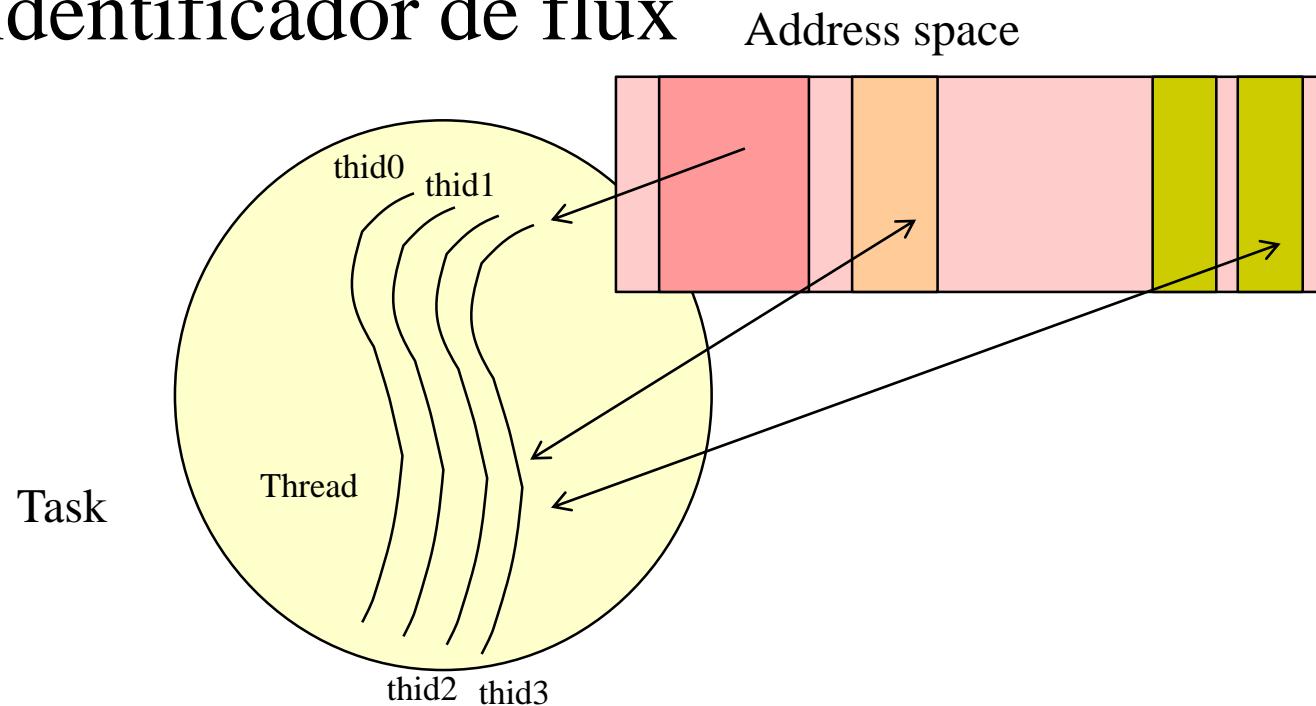
# Fluxos de sistema

- Avantatges (comparats amb els processos)
  - Poden explotar paral·lelisme dins de les aplicacions
  - Lleugers
    - Canvi de context més eficient que entre processos
      - No cal invalidar tot el TLB
    - Reutilitzen dades portades per altres fluxos
      - Similaritat amb el “hyperthreading”
  - Estalvien recursos
    - Compartits amb els altres fluxos del procés

# Fluxos de sistema

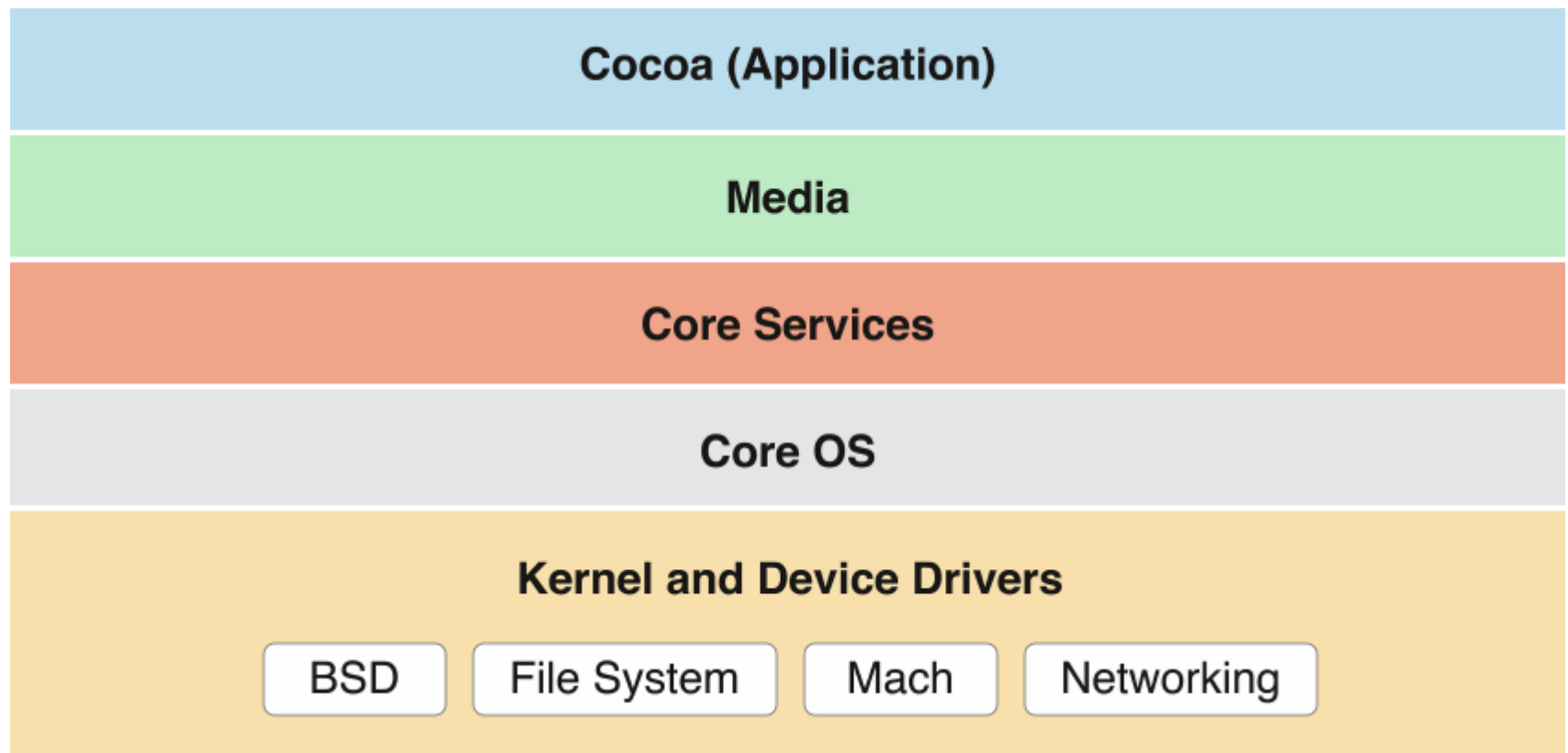
## •Opció 1 d'implementació

- Abstracció independent
- Amb identificador de flux



# Fluxos de sistema

•Exemplos: Mach, Tru64 UNIX (AlphaAXP)\*, Solaris, HP-UX, Mac OS-X\*, GNU Hurd\*, Windows

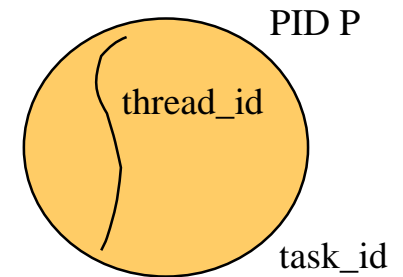


# Fluxos de sistema

## •Mach

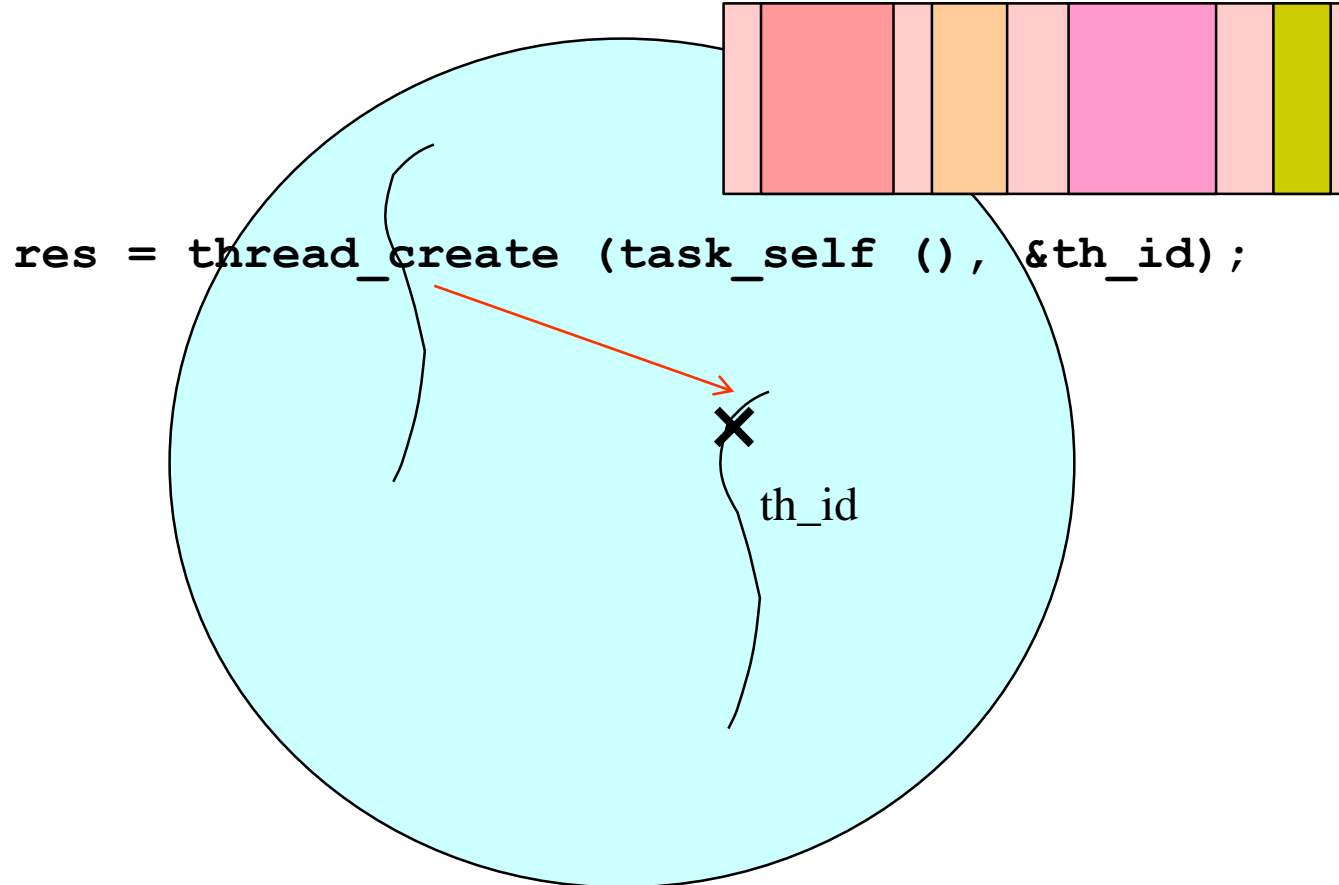
- Tasks identificades amb task\_id
  - `task_id = task_self();`
- Exemple de creació d'un flux

```
res = thread_create (task_id, &thread_id);  
res = thread_get_state (thread_id, &context, size);  
// Initialize PC, SP and other registers needed  
res = thread_set_state (thread_id, &context, size);  
res = thread_resume (thread_id);
```



# Fluxos de sistema

`.thread_create(...)`



# Fluxos de sistema

## •thread\_create(...)

### Example

```
res = thread_create (task_self (), &thid);  
if (res!=KERN_SUCCESS) {  
    mach_error ("thread_create", res);  
    exit (1);  
}  
res = thread_get_state (thid, &context, sizeof (context));  
context.pc = func;  
context.sp = malloc (stack_size) + stack_size;  
context.a0 = argument;  
res = thread_set_state (thid, &context, sizeof (context));  
res = thread_resume (thid);  
printf ("New thread id %d\n", thid);
```



(continua)

# Fluxos de sistema

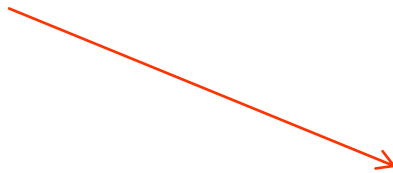
•thread\_create(...)

## Example

```
res = thread_create (task_self (), &thid);
```

```
...
```

```
res = thread_resume (thid);
```



```
void func (int argument)
```

```
{
```

```
    printf ("thread %d, argument %d\n",
```

```
           thread_self (), argument);
```

```
    thread_terminate (thread_self ());
```

```
}
```



# Fluxos de sistema

## •Conversió d'identificadors

### – versió 1

- `pid_for_task(task_id, &pid);`
  - Retorna el pid de la task
- `task_for_pid(task_self(), pid, &task);`
  - Retorna el task\_id del procés indicat per pid

### – versió 2 (la del Debian-Hurd de la pràctica)

- `task_t pid2task (pid_t pid);`
- `pid_t task2pid (task_t task);`

# Fluxos de sistema

- Windows – crear fluxos en altres processos

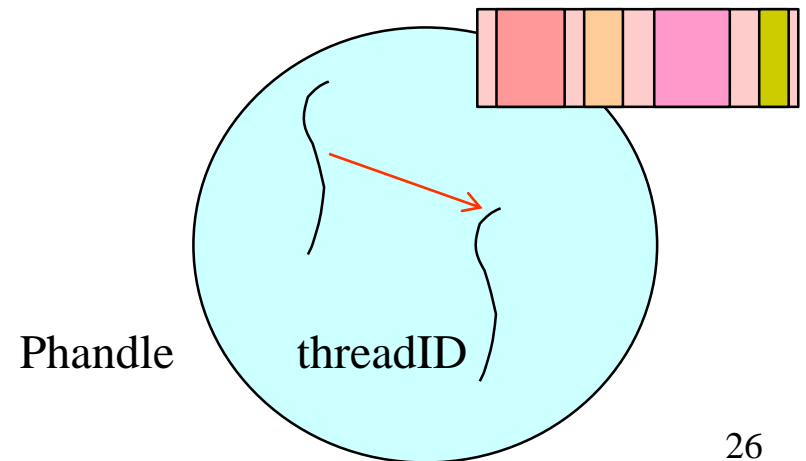
- CreateRemoteThread(...);
- CreateRemoteThreadEx (Phandle, attr, stackSize, function, argument, creationFlags, attrList, &threadID);

- CreateThread (... sense handle, ni attrList... )

- Crea un flux en el propi procés

<http://msdn.microsoft.com/en-us/library/>

- Windows Development
- System Services
- Processes and Threads



# Fluxos de sistema

## •Opció 2 d'implementació: Linux

- Crides menys estructurades
  - `fork()` / `clone()`
  - `pause()`
  - `signal()` / `kill()`, `SIGCONT`, `SIGSTOP`
  - `nice()`
  - `exit()`
- Funcionalitat reduïda
- No és prou versàtil, en general no es poden assignar recursos a altres processos
  - I de vegades tampoc se'ls poden canviar algunes característiques
- Solució: `clone(...)`

# Suport a Linux

## •Processos compartits (Linux clones)

- `pid = clone(funció, stack, flags, argument);`

### Example

```
pid = clone (func, usp, flags, argument);
```

```
if (pid<0) {
```

```
    perror ("clone");
```

```
    exit (1);
```

```
}
```

```
printf ("New clone pid %d\n", pid);
```

```
res = wait (&status);
```

```
...
```

waitpid!!!

```
void func (void * arg)
```

```
{
```

```
    int argument = (int) arg;
```

```
    printf ("clone %d, arg %d\n",  
            getpid (), argument);
```

```
    exit (0);
```

```
}
```

# Suport a Linux

## •Processos compartits (Linux clones)

- `pid = clone(funció, stack, flags, argument);`

### Example

```
pid = clone (func, usp, flags, argument);
```

```
if (pid<0) {  
    perror ("clone");  
    exit (1);  
}
```

```
printf ("New clone pid %d\n", pid);
```

```
res = waitpid (pid, &status, options);
```

```
...
```

```
void func (void * arg)  
{
```

```
    int argument = (int) arg;  
    printf ("clone %d, arg %d\n",  
           getpid (), argument);  
    exit (0);
```

```
}
```

# Suport a Linux

## •Linux clones, flags:

- CLONE\_VM, compartir l'espai d'adrees
- CLONE\_FILES, compartir els descriptors de fitxer
- CLONE\_FS, compartir directoris arrel i actual
- CLONE\_SIGHAND, compartir programació de signals
- CLONE\_THREAD, compartir el pid
- ... altres
- Signal a enviar al pare, quan el fill acabi

## •UNIX fork() és un cas especial de clone

- I així s'implementa
  - clone (NULL, NULL, SIGCHLD, NULL);

# Suport a Linux

## •Wait, waitpid, waitid, wait3, wait4

- no funcionen correctament per esperar clones que comparteixen el PID
- creï qui creï, tenen el mateix pid, i ppid

```
bash-4.2$ ./clones
clone 5633, 6060(6062), arg 1
clone 5633, 6060(6061), arg 0
clone 5633, 6060(6063), arg 2
clone 5633, 6060(6064), arg 3
waitpid: No child processes
clone 5633, 6060(6067), arg 0
clone 5633, 6060(6066), arg 0
waitpid: No child processes
clone 5633, 6060(6068), arg 1
clone 5633, 6060(6065), arg 0
clone 5633, 6060(6069), arg 1
clone 5633, 6060(6070), arg 1
waitpid: No child processes
```

```
waitpid: No child processes
clone 5633, 6060(6071), arg 0
clone 5633, 6060(6072), arg 1
waitpid: No child processes
clone 5633, 6060(6074), arg 0
waitpid: No child processes
clone 5633, 6060(6075), arg 0
clone 5633, 6060(6077), arg 1
clone 5633, 6060(6076), arg 1
clone 5633, 6060(6078), arg 0
clone 5633, 6060(6083), arg 1
waitpid: No child processes
clone 5633, 6060(6079), arg 0
clone 5633, 6060(6081), arg 1
clone 5633, 6060(6084), arg 0
```

# Suport a Linux

- Conclusió: no es pot fer servir cap crida a sistema per esperar que un flux (clone) acabi
  - S'ha de fer:
    - amb variables compartides
    - es por aprofitar la interfície de clone(...)
  - Demo: clones-ok.c



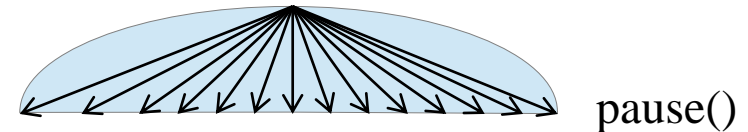
# Pthreads

- Llibreria de suport (de nivell usuari)
  - Defineix la interfície per una gestió fàcil dels fluxos
    - Creació, destrucció
    - Característiques
    - Prioritat
    - Política de planificació
  - Principal objectiu: portabilitat

# Pthreads

- Quants pthreads suporta un procés en Linux?
  - Fem un programa per detectar el límit

```
void * function (void * arg)
{
    pause();
}
```



```
int main()
{
    int i = 0, limit, res;
    while (i < N) {
        res = pthread_create(&threads[i], NULL, function, (void *) (unsigned long) i);
        if (res != 0) {
            fprintf(stderr, "pthread_create: %s\n", strerror(res));
            break;
        }
        ++i;
    }
    ...
}
```

# Pthreads

• `ps -p <pid> \`  
`-mo pid,nlwp,tid,psr,policy,ni,pri,\`  
`flags,rip,rsp,cputime,etime,cmd`

PID	NLWP	TID	PSR	POL	NI	PRI	F	RIP	RSP	TIME	ELAPSED	CMD
3025	81	-	-	-	-	-	0	-	-	00:00:00	51:57	./howmanypthreads
-	-	3025	3	TS	0	19	0	00007f950e465222	00007fff46205ba0	00:00:00	51:57	-
-	-	3026	1	TS	0	19	1	00007f950e46af9d	00007f950e091f00	00:00:00	51:57	-
-	-	3027	2	TS	0	19	1	00007f950e46af9d	00007f950d890f00	00:00:00	51:57	-
-	-	3028	1	TS	0	19	1	00007f950e46af9d	00007f950d08ff00	00:00:00	51:57	-
-	-	3029	1	TS	0	19	1	00007f950e46af9d	00007f950c88ef00	00:00:00	51:57	-
-	-	3030	1	TS	0	19	1	00007f950e46af9d	00007f950c08df00	00:00:00	51:57	-
-	-	3031	1	TS	0	19	1	00007f950e46af9d	00007f950b88cf00	00:00:00	51:57	-
-	-	3032	1	TS	0	19	1	00007f950e46af9d	00007f950b08bf00	00:00:00	51:57	-

# Pthreads

## • Creació d'un pthread

- Demana una estructura “descriptor de flux”
- Demana una pila i construeix un “stack frame”
  - Funció, argument
- Crea un clone (Linux) o un flux de sistema (Mach)
  - Compartint tots els recursos, inclòs el “pid”

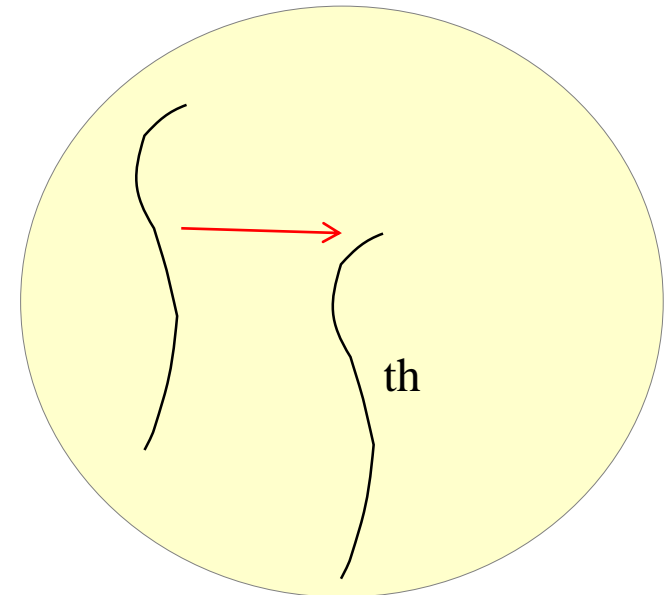
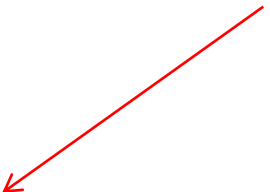
```
#include <pthread.h>
int pthread_create(
pthread_t      * thread ,
pthread_attr_t * attr ,
void *         (* start_routine) (void *) ,
void           * arg );
```

# Pthreads

## •Exemple de creació d'un flux

```
void main ()
{
    int res;
    pthread_t th;
    ...
    res = pthread_create (&th, NULL, func, argument);
    ...
}

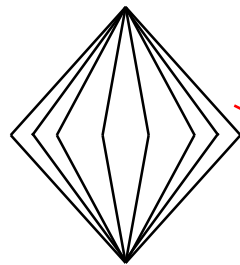
void * func (void * argument)
{
    printf ("argument %d\n", (int) argument);
}
```



# Pthreads

- Suporta diferents tipus de paral·lelisme
  - Join (estructurat) / detach (no estructurat)

**Fork / join**

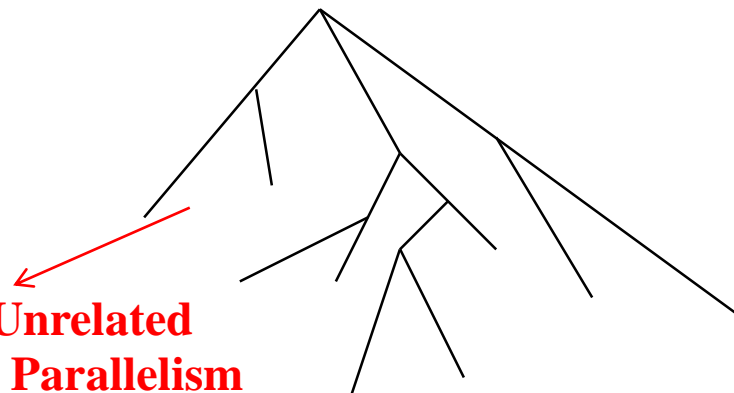


**Join**

Gets the termination code of the pthreads

**Like  
Loop/Task  
Parallelism**

**Unstructured**



**Like Unrelated  
Tasks Parallelism**

**Detach**

The application does not need to get the pthreads termination code

# Pthreads

## •Join / detach

- `pthread_exit(estat)` salva l'estat de finalització en l'estructura del pthread
- `pthread_join(pth, &status)` espera que el pthread “pth” acabi i recupera el codi de finalització
- `pthread_detach(pth)` marca l'estructura del pthread indicant que no s'esperarà per ell
- El descriptor del pthread es pot alliberar després de:
  - Haver fet un `pthread_exit` (el thread) i un `pthread_join()`
  - Haver fet un `pthread_detach` i un `pthread_exit` (el thread)

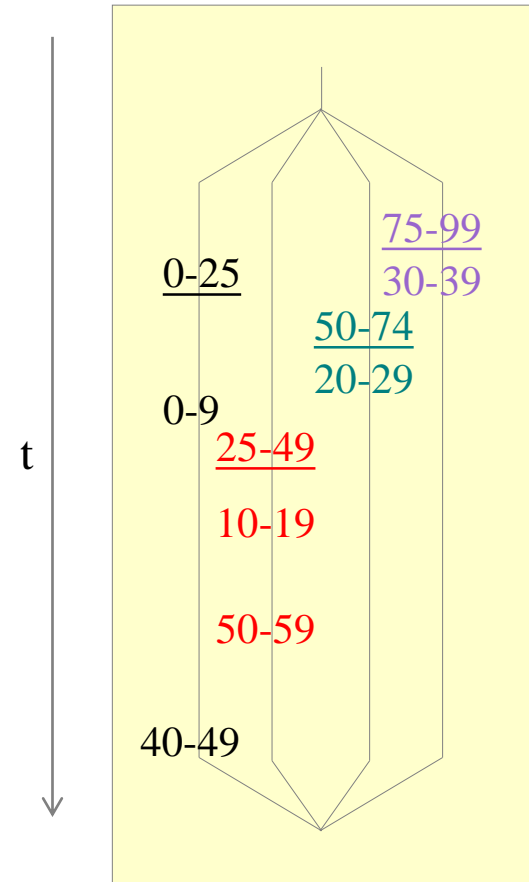
# Pthreads

• Habitualment s'usa com a base per implementar altres models

– OpenMP / OmpSs...

```
#pragma omp parallel
{
  #pragma omp for schedule (STATIC) nowait
  for (i=0; i<100; i++)
    A[i] = B[i] + x*B[i-1] + y*B[i+1];

  #pragma omp for schedule (STATIC,10)
  for (i=0; i<60; i++)
    A[i] = C[i] + y*C[i+1];
}
```





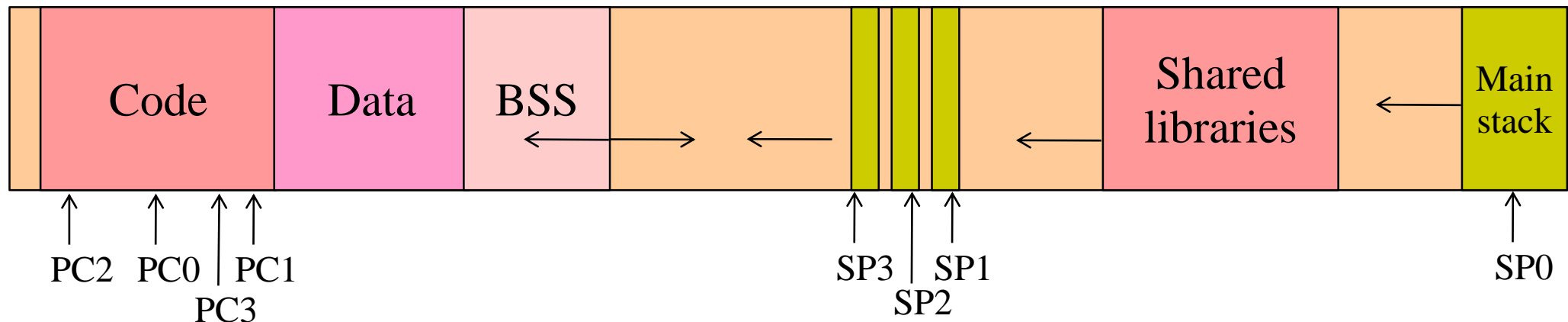
# Gestió de memòria

• `brk()` / `sbrk()`: break point

– Increment (+/-)

– Set

Interfície de Sistema



# Gestió de memòria

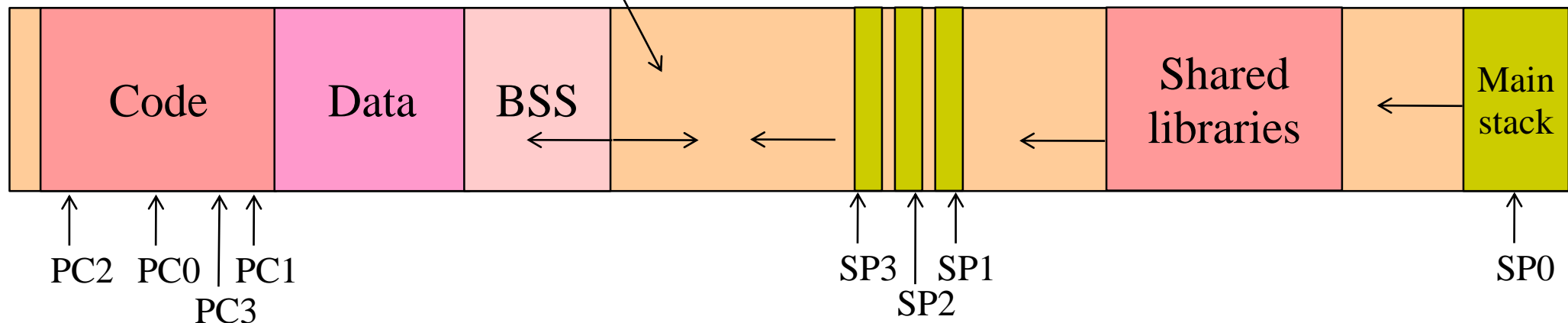
- malloc()/free()

- memalign()

- sobre brk()

- i/o mmap()

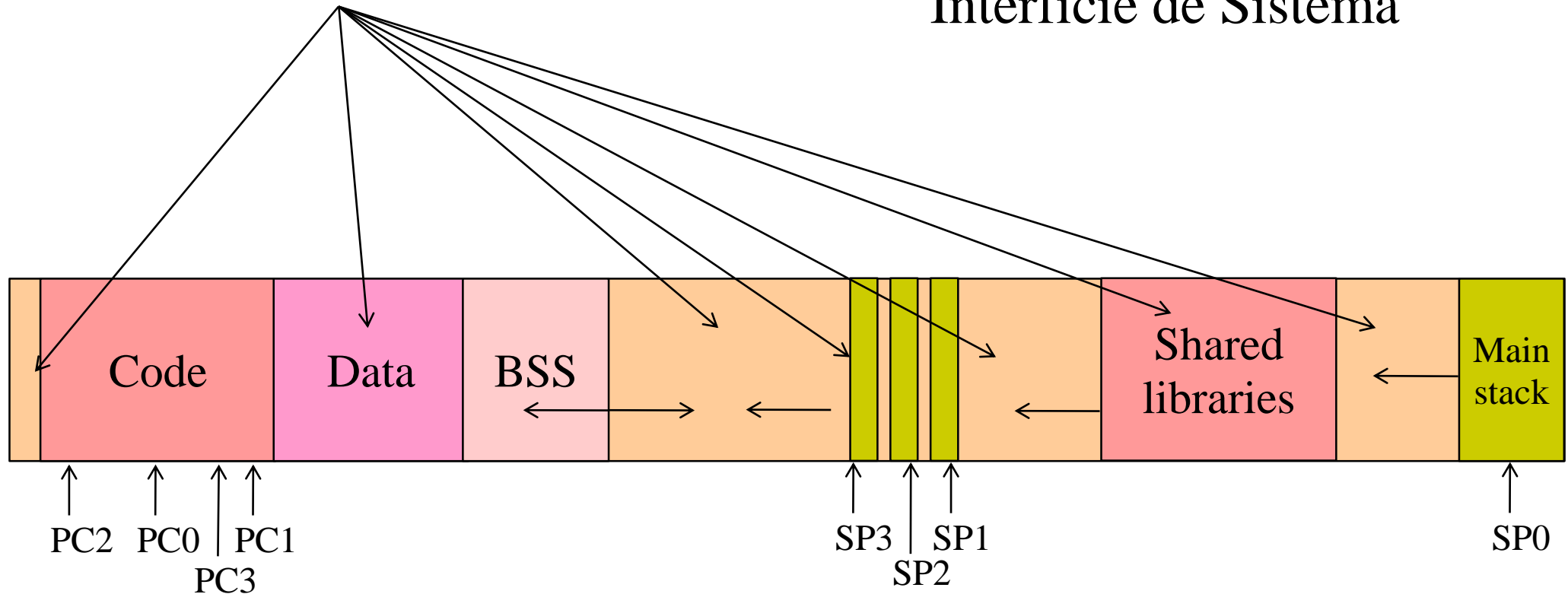
Interfície d'Usuari



# Gestió de memòria

•mmap()/munmap()

Interfície de Sistema

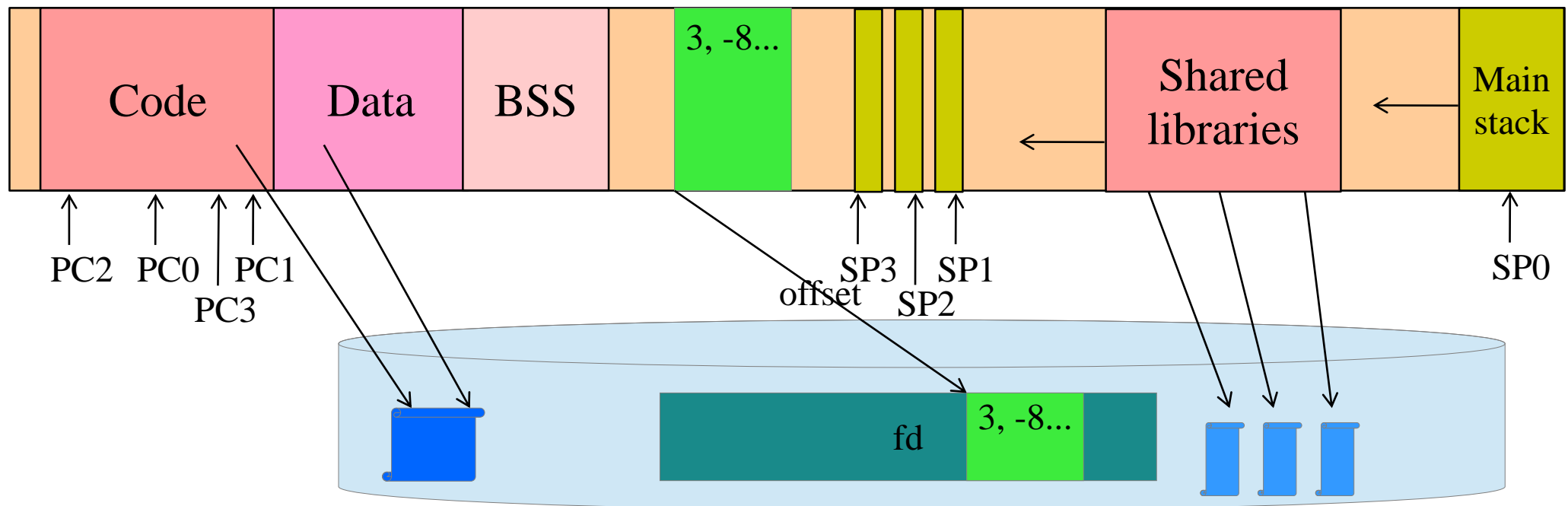


# Gestió de memòria

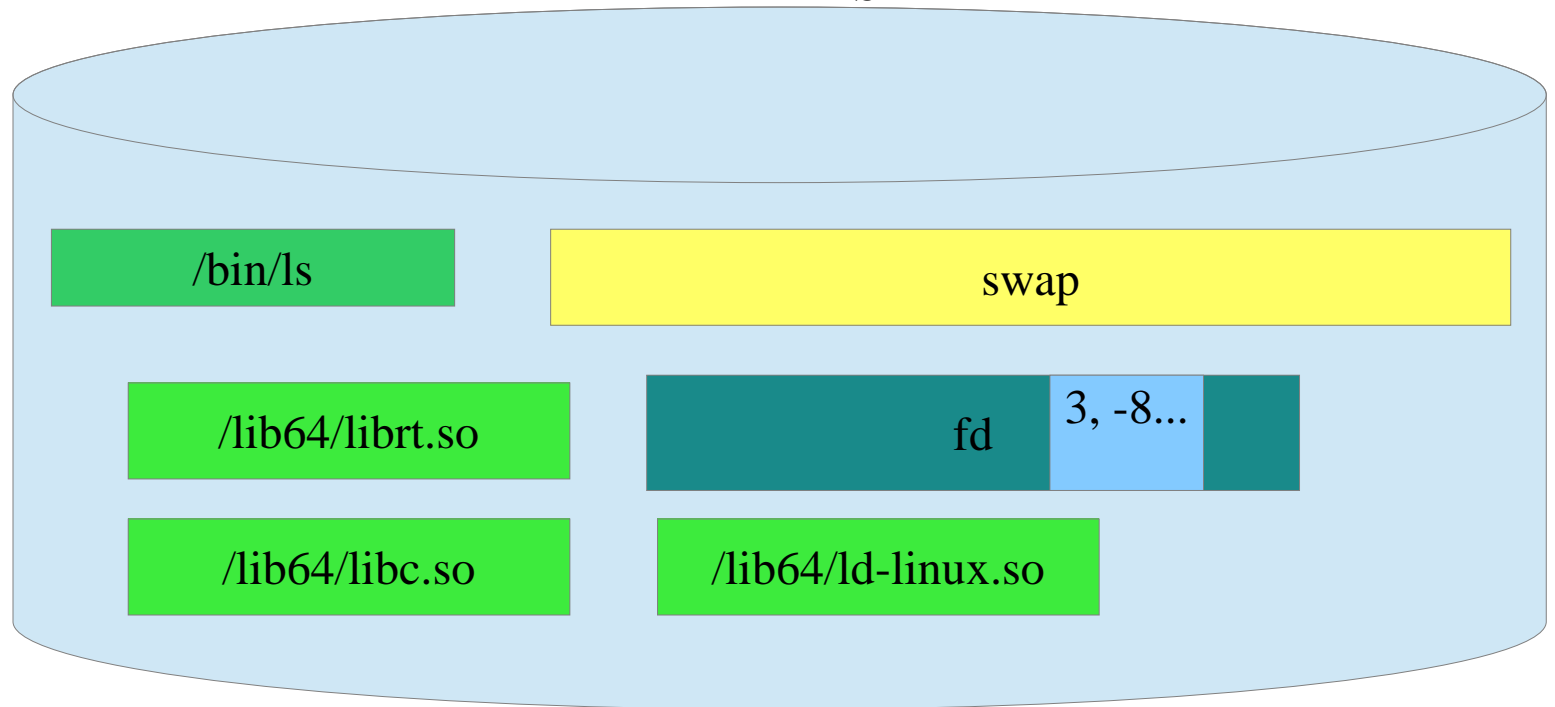
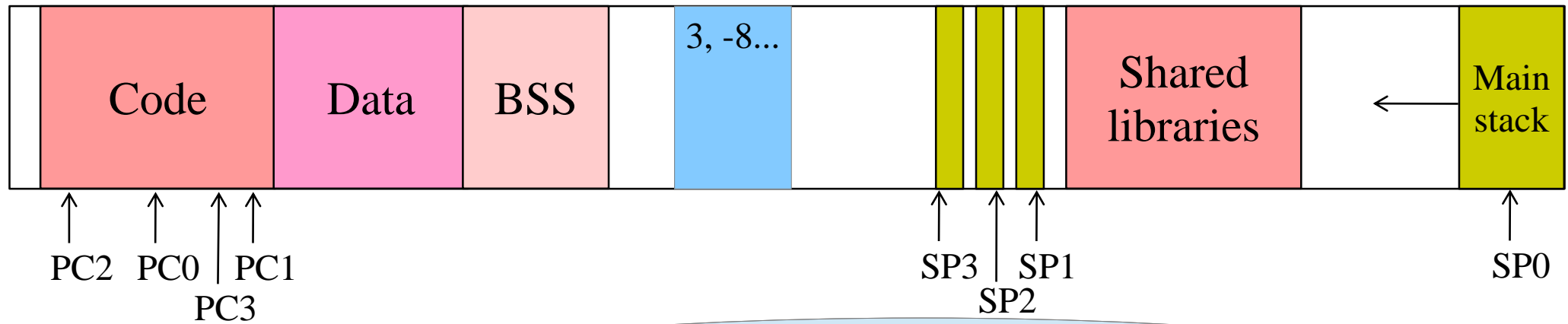
`mmap (addr, size, permissions, flags, fd, offset);`

– Permet demanar:

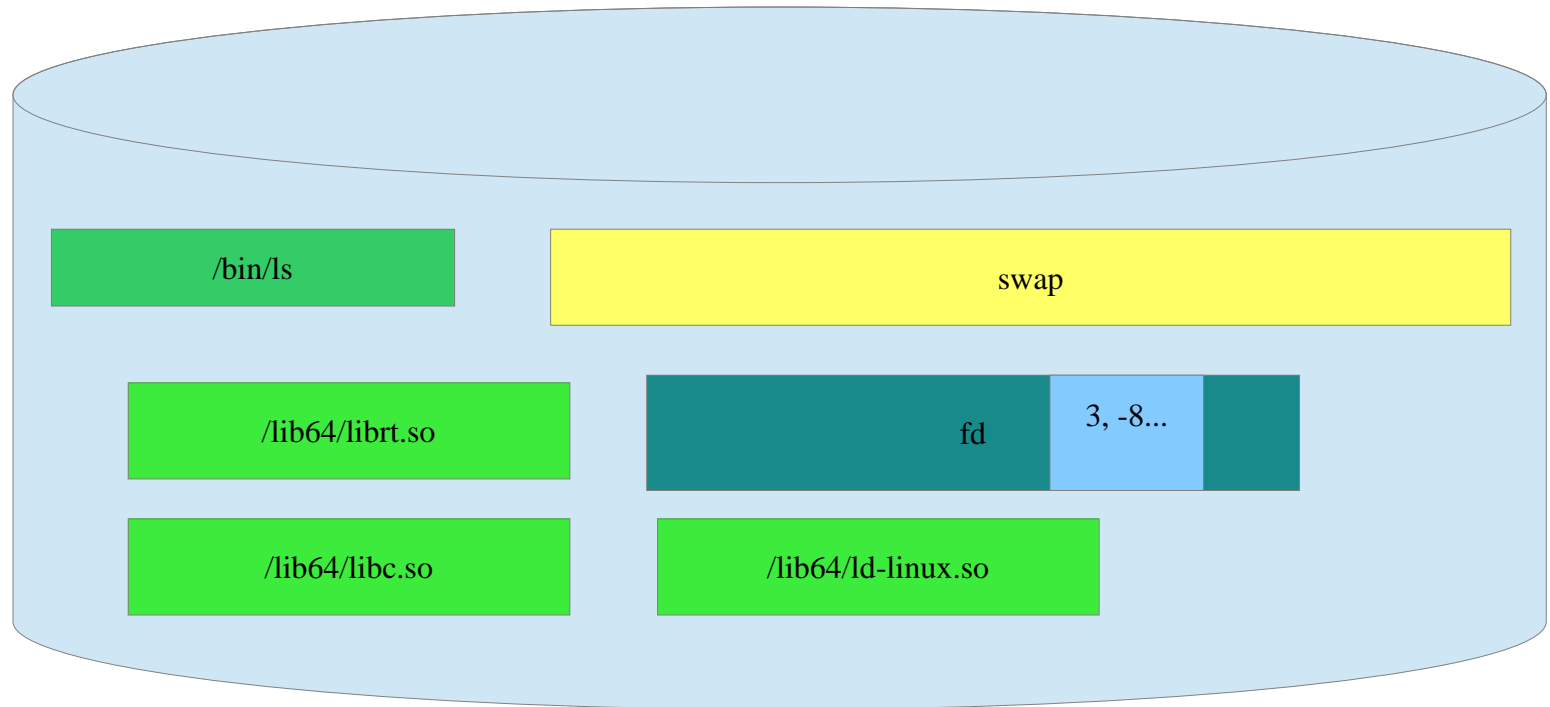
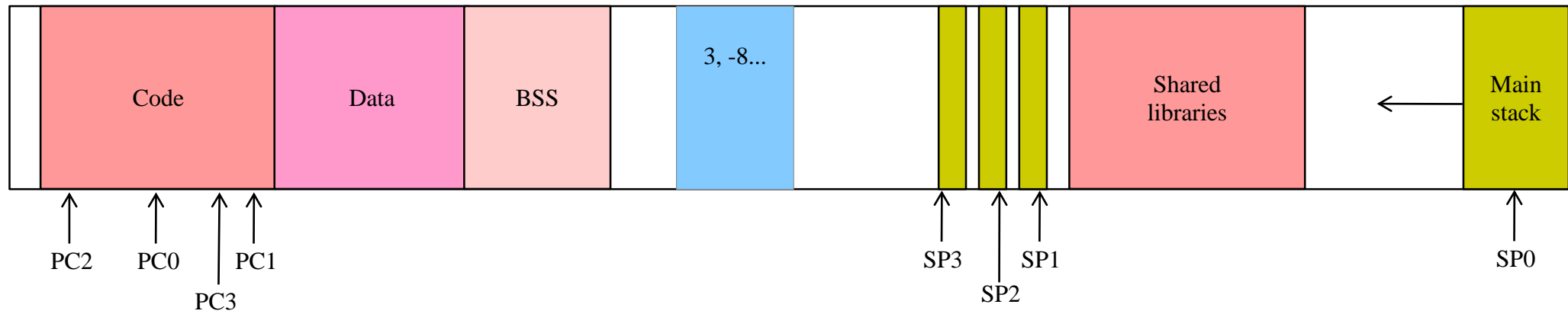
- Memòria anònima: virtualitzada a l'àrea de swap
- Memòria mapejada en un fitxer



# Activitat

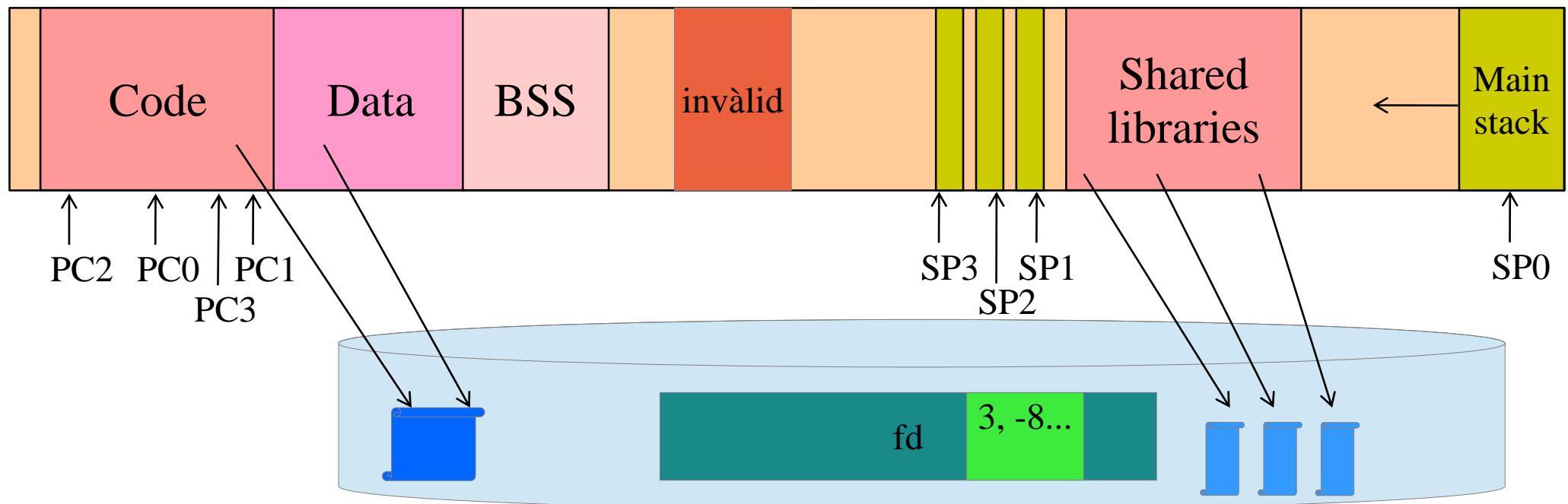


# Activitat



# Gestió de memòria

- `munmap (addr, size);`
  - Allibera la memòria, de forma que la regió queda invàlida
    - accessos causaran segmentation fault



# Per a la setmana del 4 de març

## •Documentar-se

- Abstraccions i interfície de Mach
- Suport hardware necessari per a l'exclusió mútua