Lecture 1 - Processes

Operating Systems 1

Warsaw University of Technology - Faculty of Mathematics and Information Science

Process Concept

A unit of work scheduled by the user

```
> man 7 ps
```

Process = a program in execution

not necessarily executing at the moment

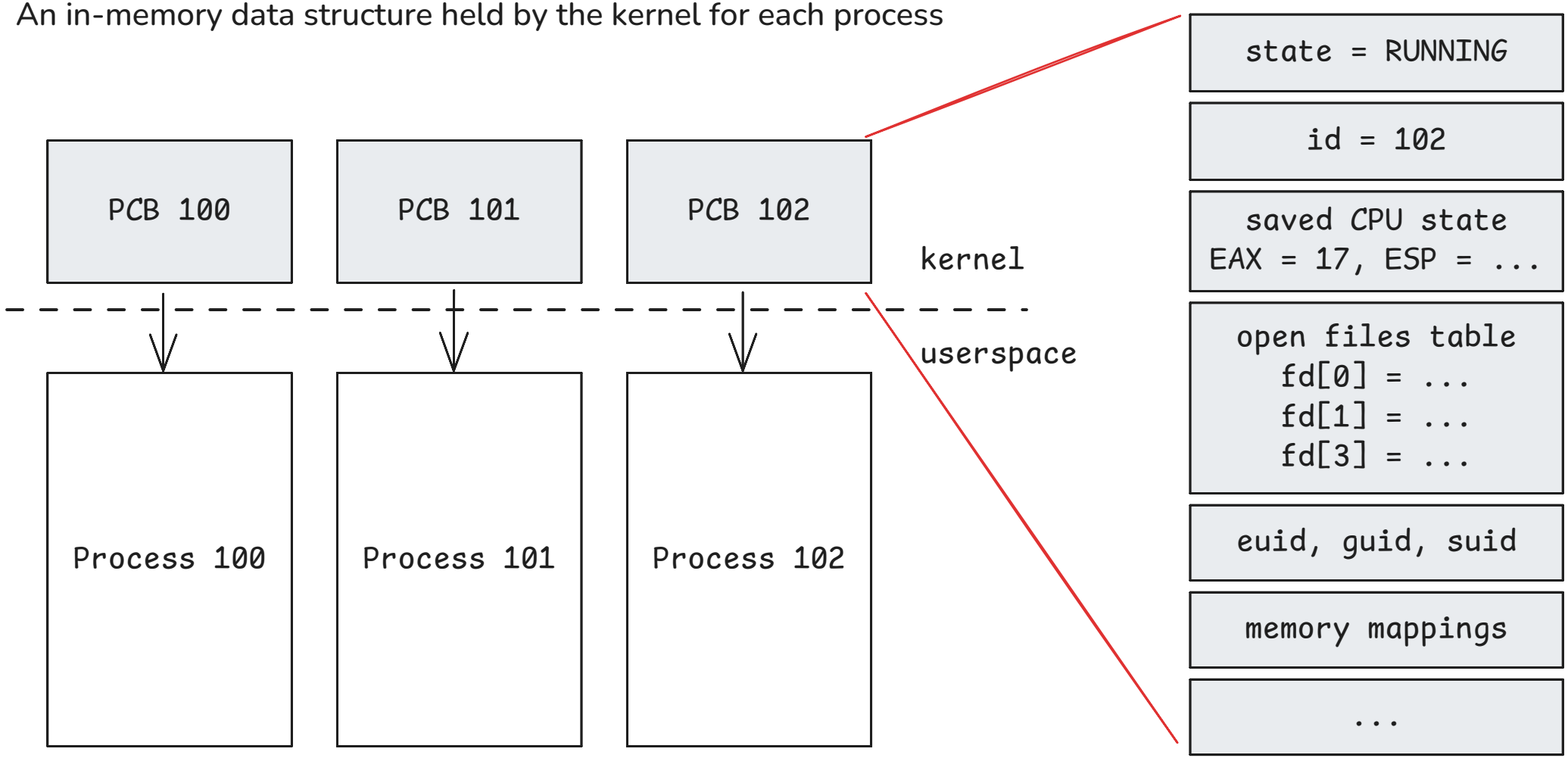
Other names: Task - in time-shared systems, Job - in batch systems

POSIX definition:

An address space with one or more threads executing within
that address space, and the required system resources for those threads

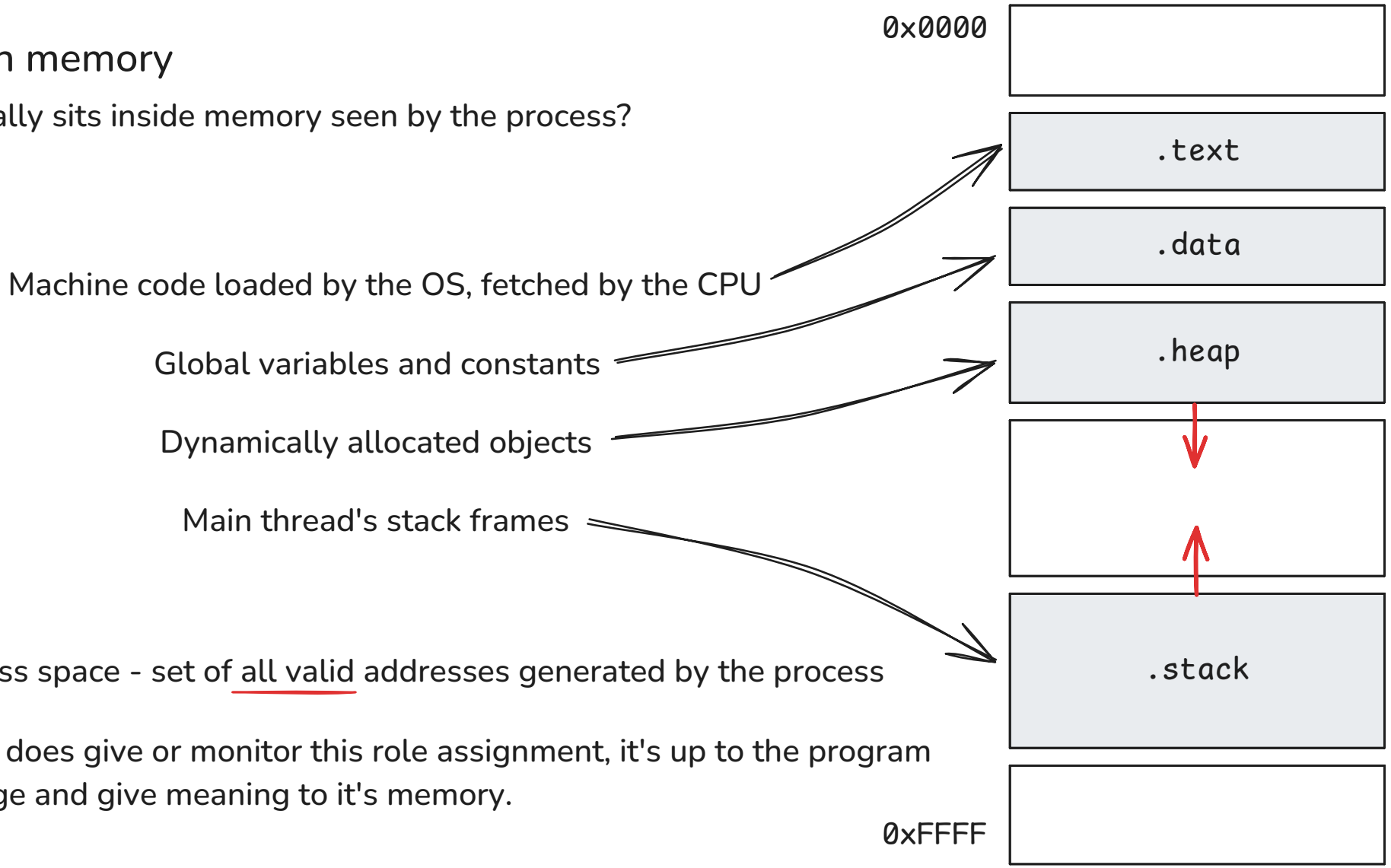
Process Control Block

An in-memory data structure held by the kernel for each process



Process in memory

What typically sits inside memory seen by the process?

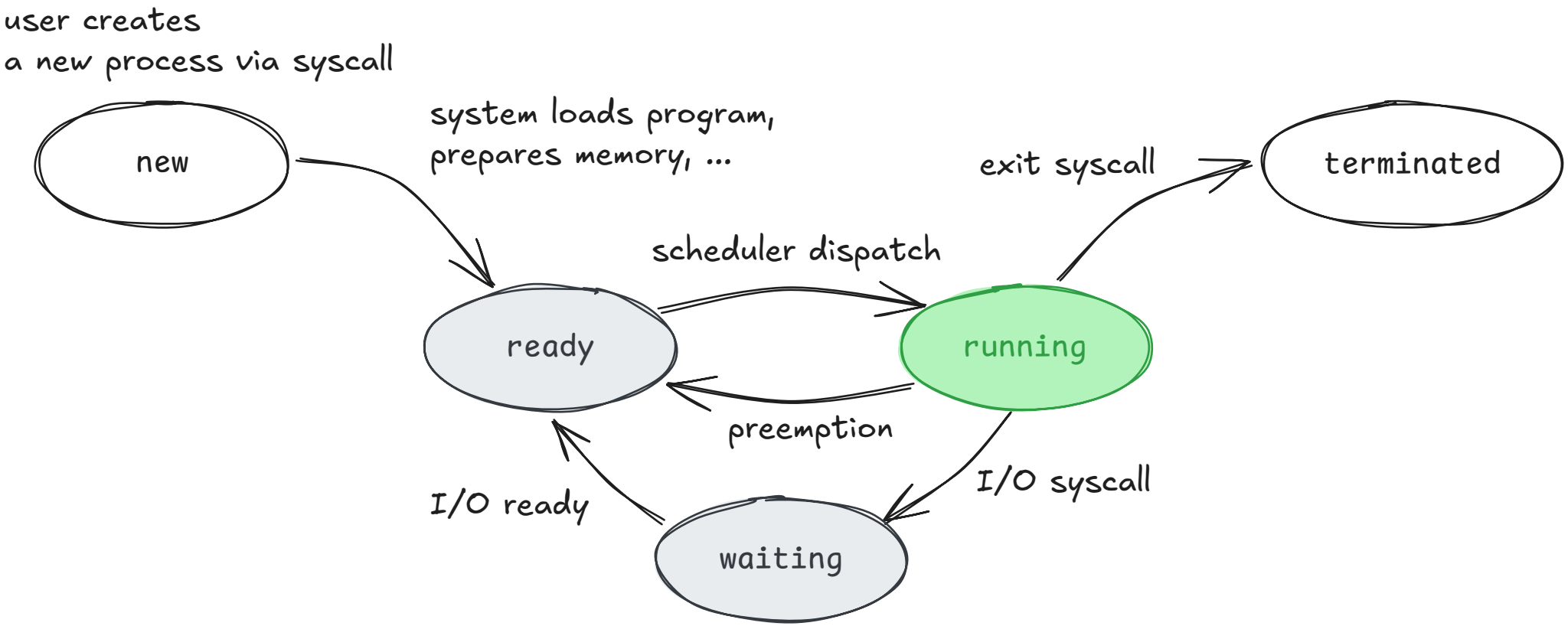


Address space - set of all valid addresses generated by the process

OS itself does give or monitor this role assignment, it's up to the program to manage and give meaning to it's memory.

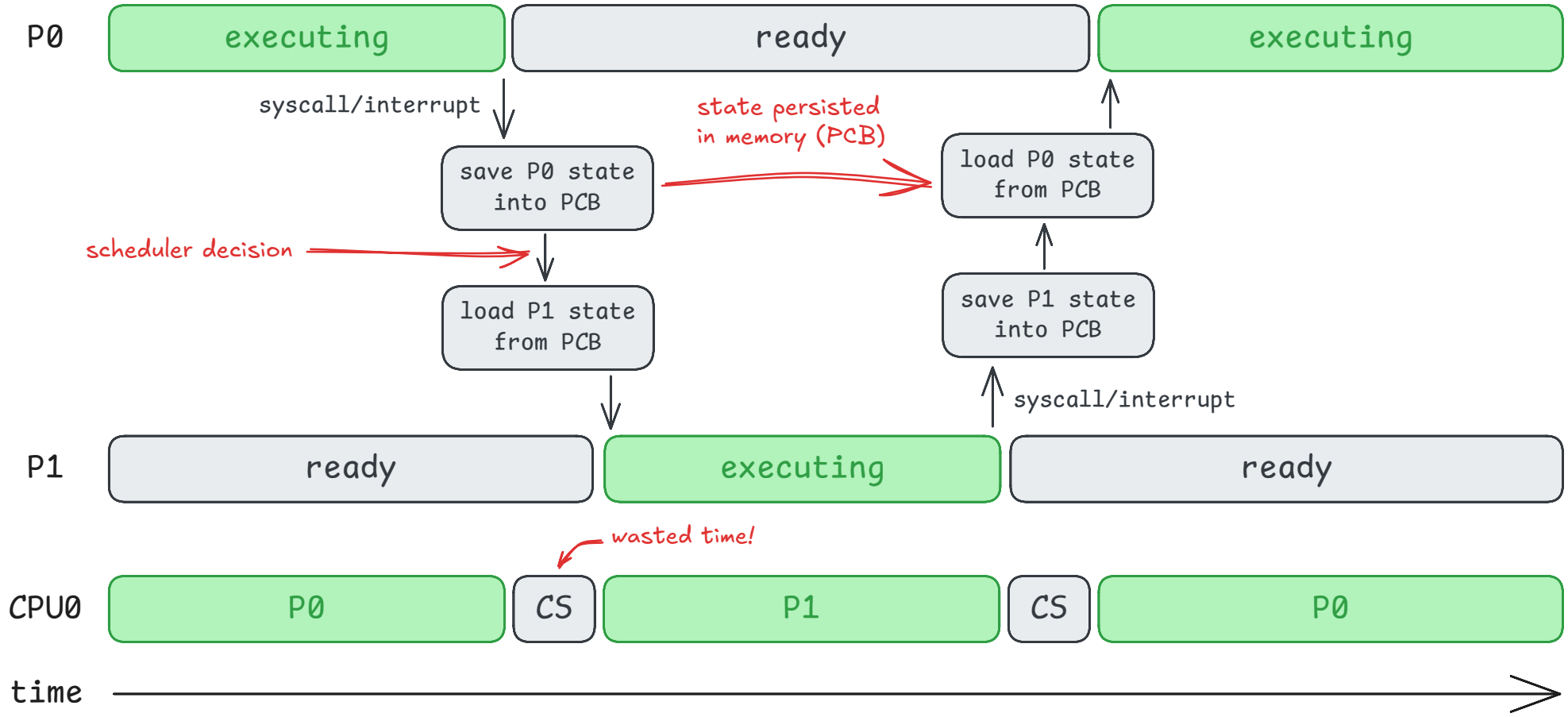
Process Lifetime

An in-memory data structure held by the kernel for each process



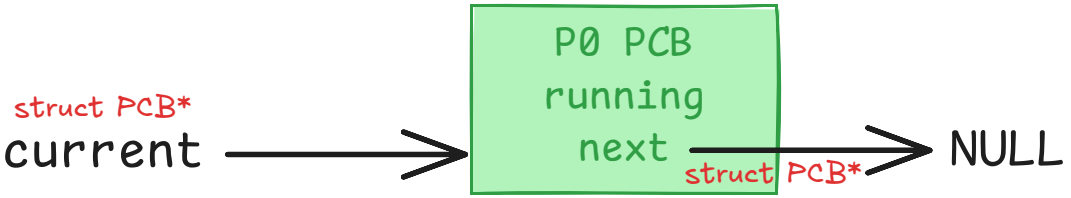
A context switch

What happens when CPU switches from one task to another

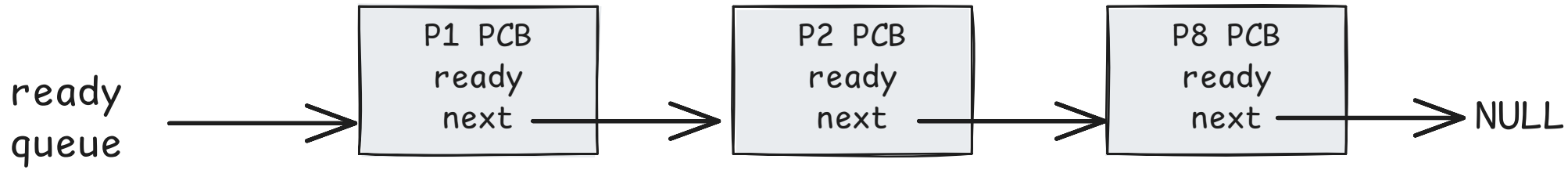


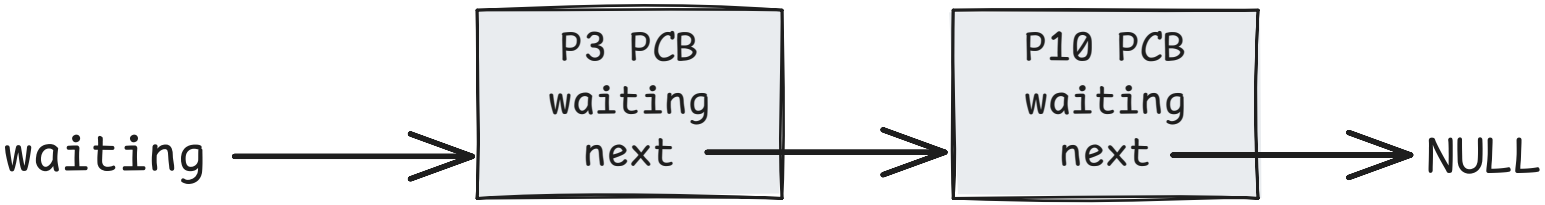
Meet the scheduler

Kernel algorithm which selects the next process to be dispatched on a CPU



PCBs are organized into a linked list.
The simplest scheduler picks the list head as the next process once currently running is preempted or needs to wait.

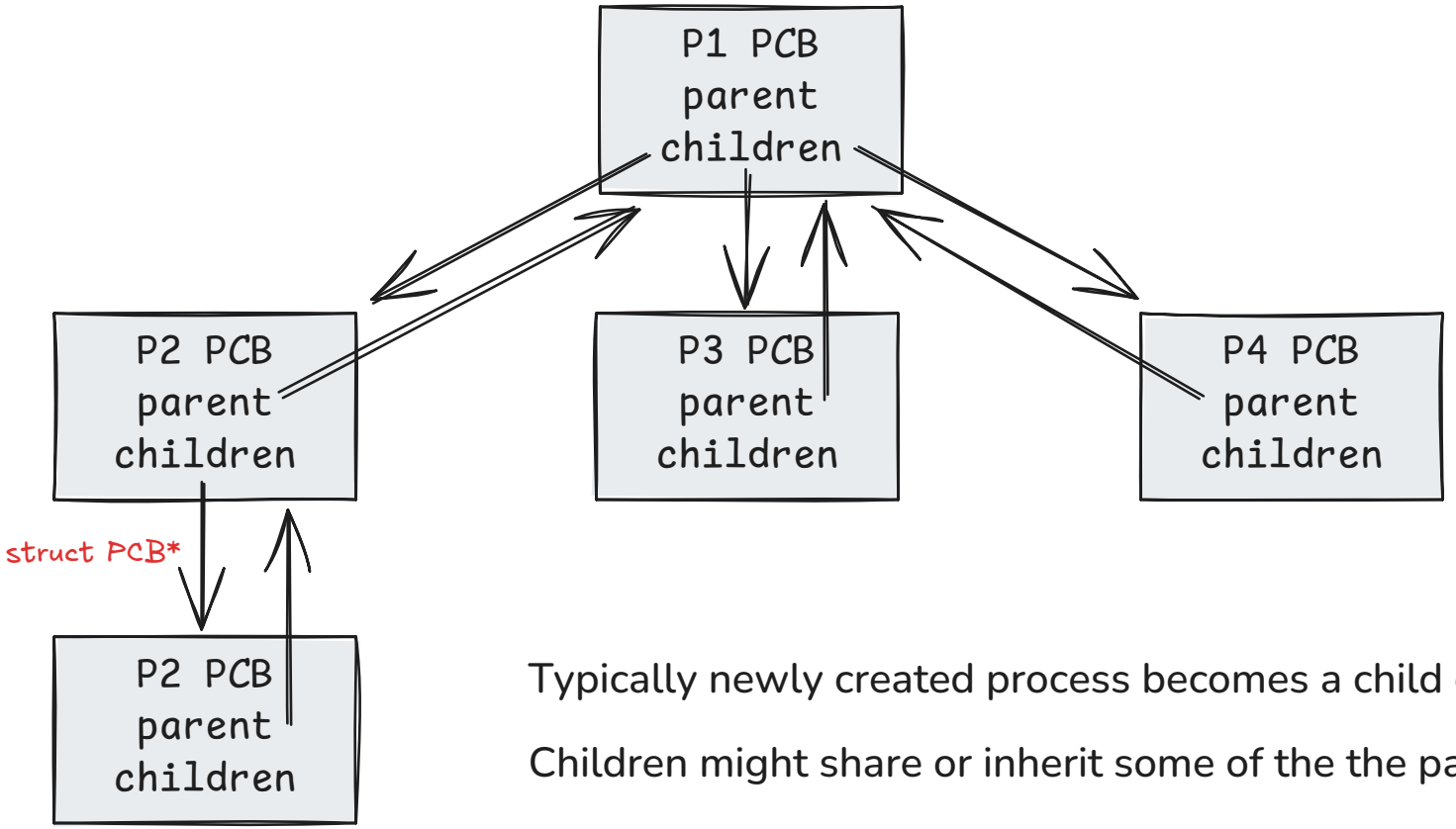




Process Hierarchy

```
> man 1 pstree
```

Commonly there exists parent-child relationship between processes



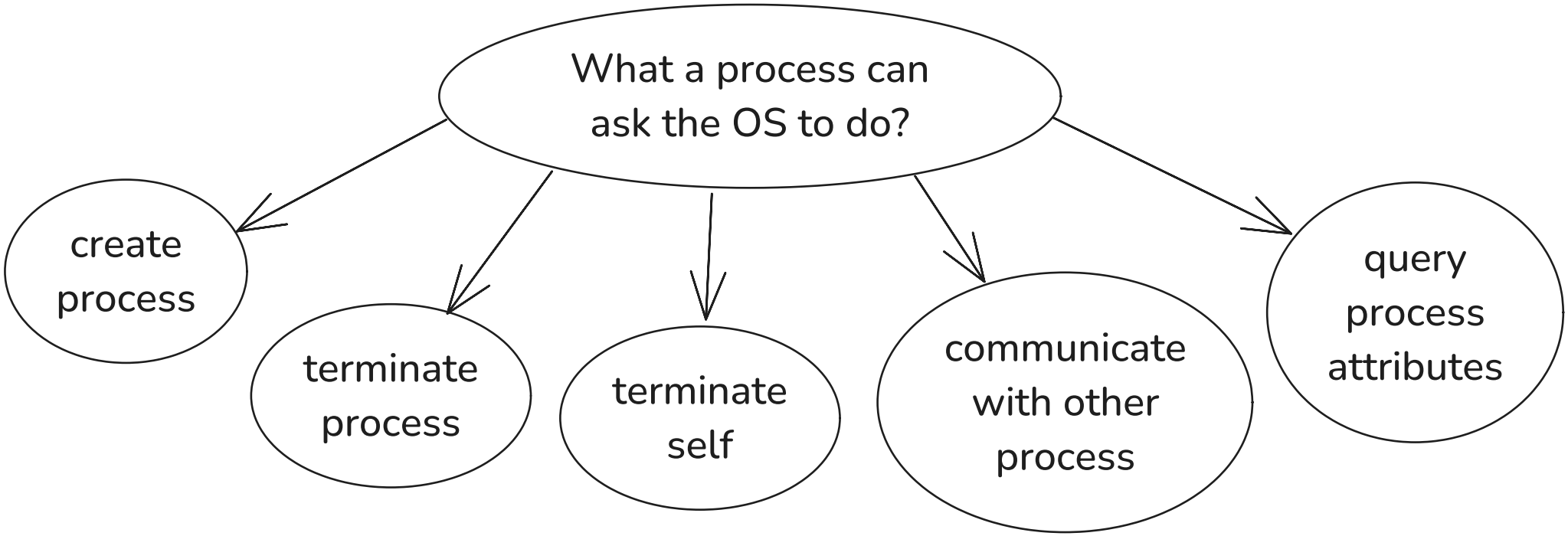
Typically newly created process becomes a child of it's creator.

Children might share or inherit some of the the parent's resources.

Process lifetime related syscalls

Process management interface provided by the OS

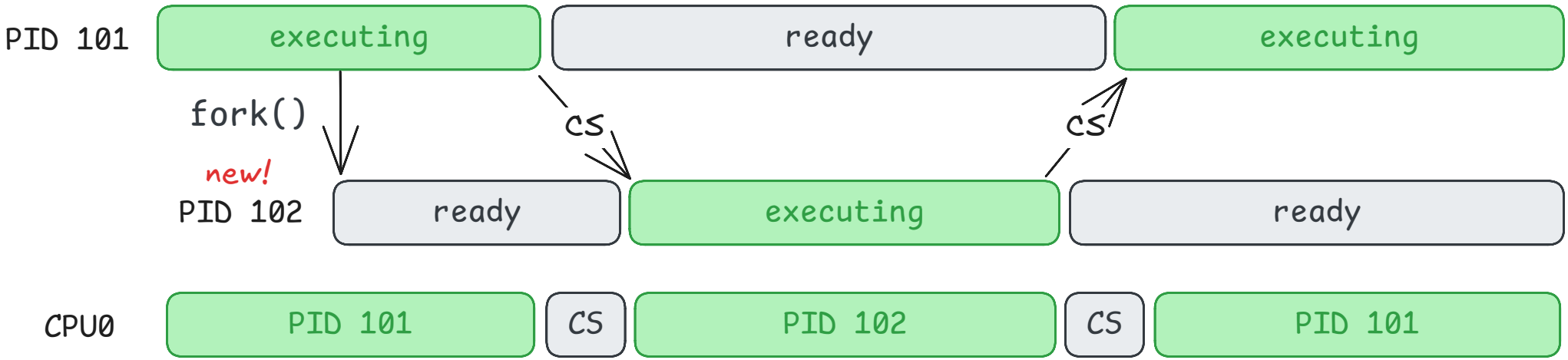
OS by itself is not interested in creating processes. It serves the user (applications).
User might want to run a process. The only way user can ask the OS to do something is through the syscall.



Process creation

> man 3p fork

The mighty fork() syscall

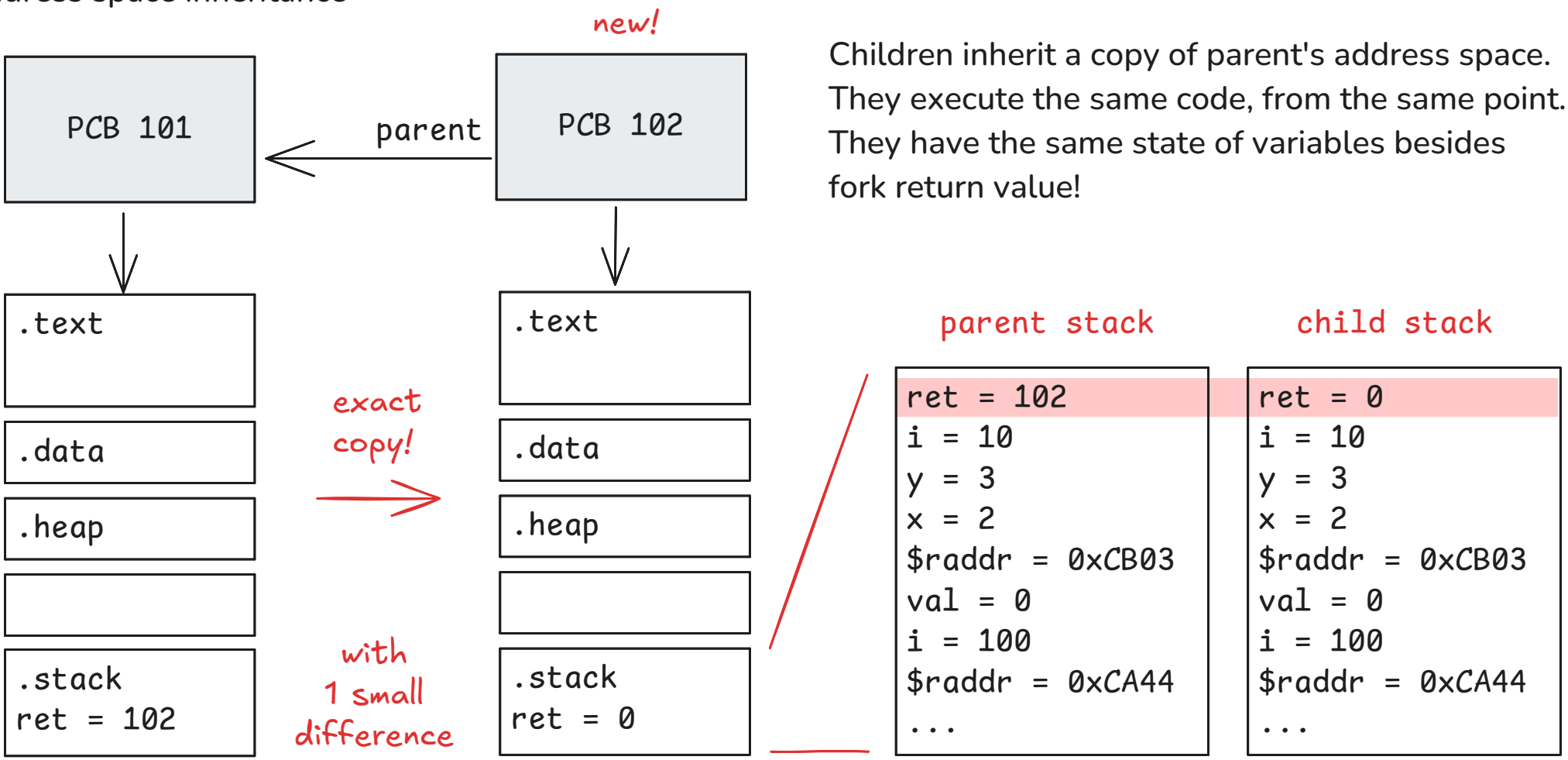


During fork() handling system allocates and initializes a new PCB along with necessary resources.
This new PCB is then yield to the scheduler as a new ready queue element.

Note: In case of multicore system the new process could execute (nearly) immediately on a distinct CPU.

What does the child do?

Address space inheritance



Children inherit a copy of parent's address space. They execute the same code, from the same point. They have the same state of variables besides fork return value!

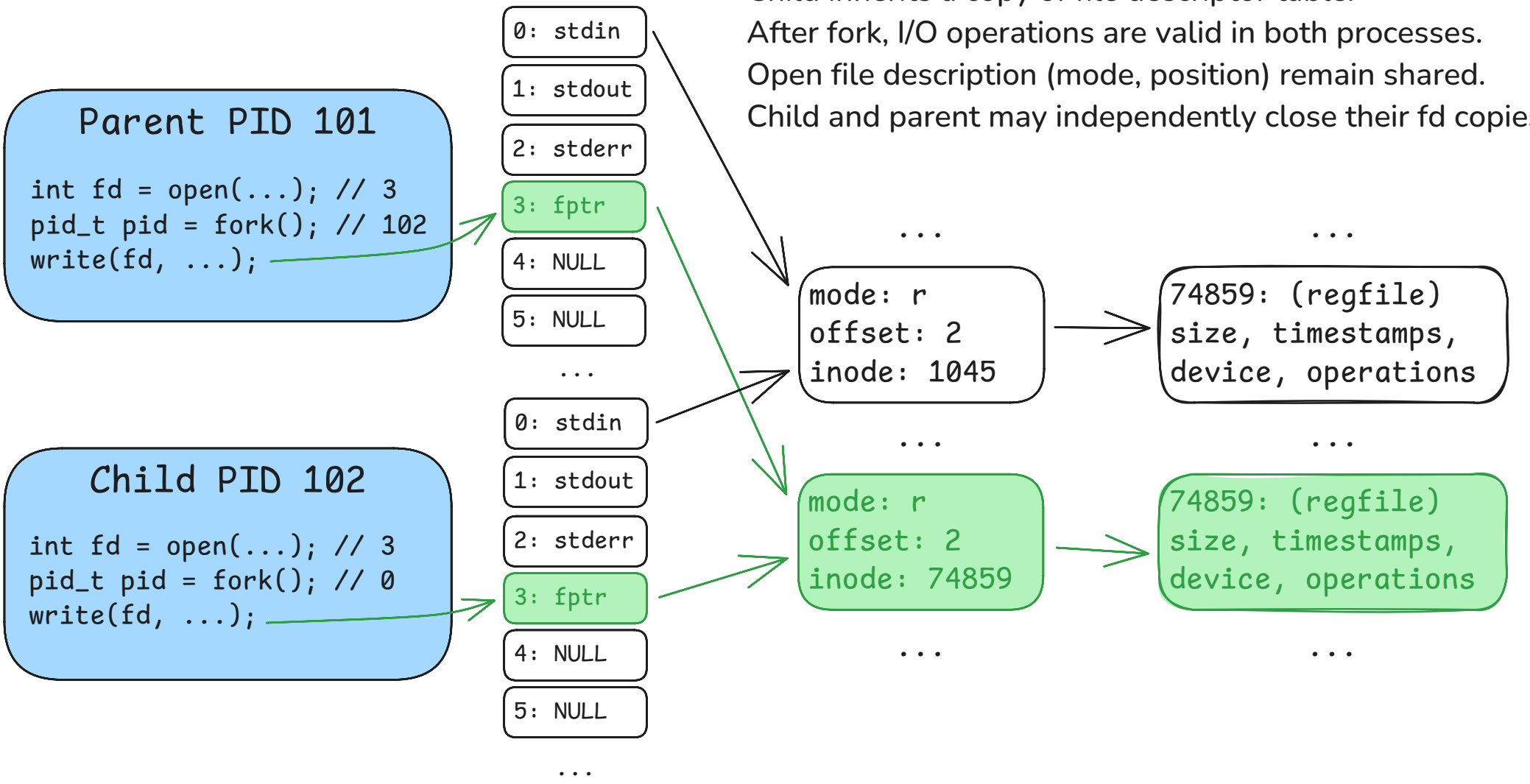
Process attribute inheritance

What beyond the memory is inherited (by default)?

INHERITED	NOT INHERITED
address space contents	timers
memory mappings	awaiting signals
open file descriptors	outsantding asynchronous operations
environment variables	threads
signal handlers	file locks
scheduling policy	memory locks
process priority	...
...	

File descriptor inheritance

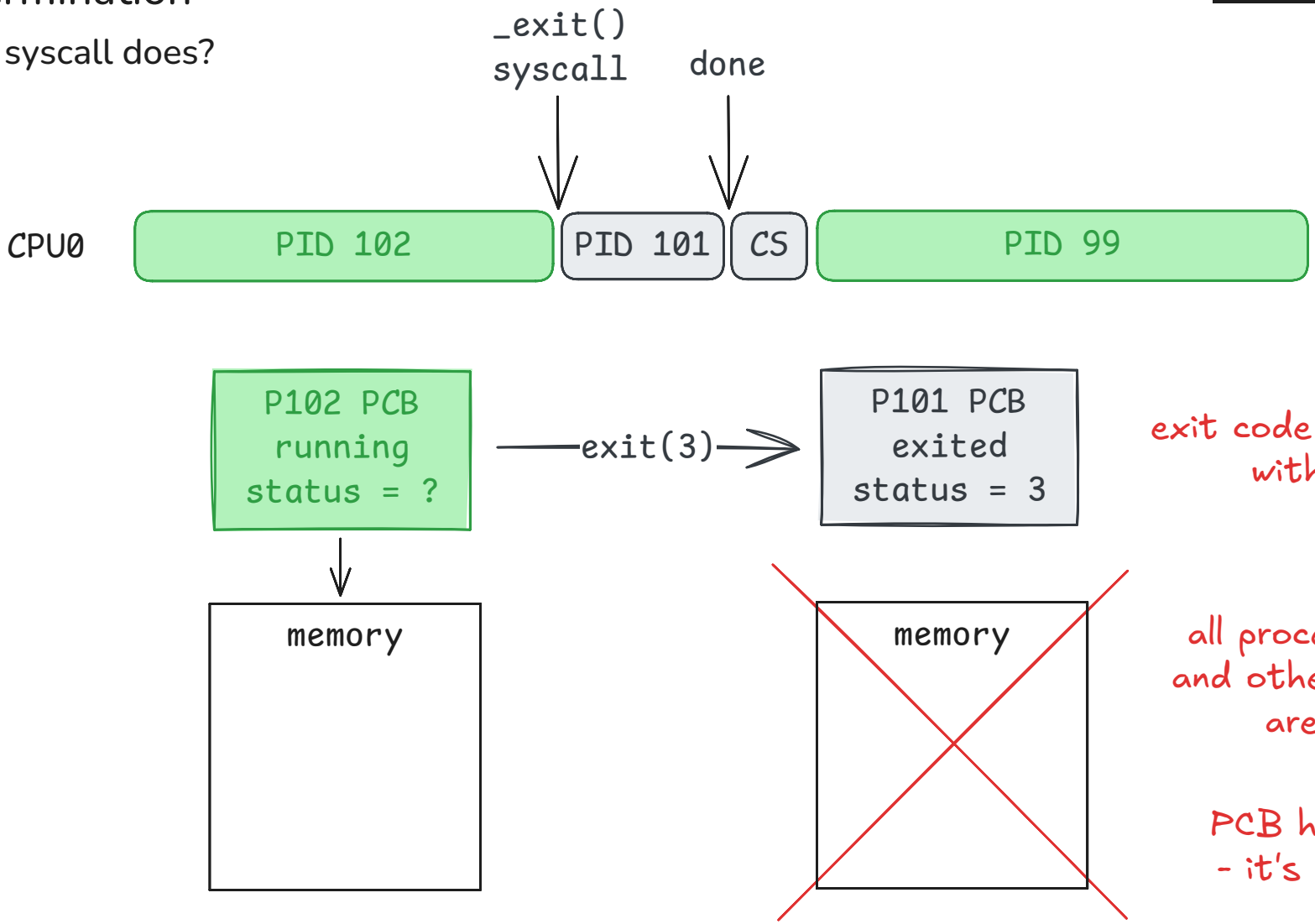
Child inherits a copy of file descriptor table.
After fork, I/O operations are valid in both processes.
Open file description (mode, position) remain shared.
Child and parent may independently close their fd copies.



Process termination

What _exit() syscall does?

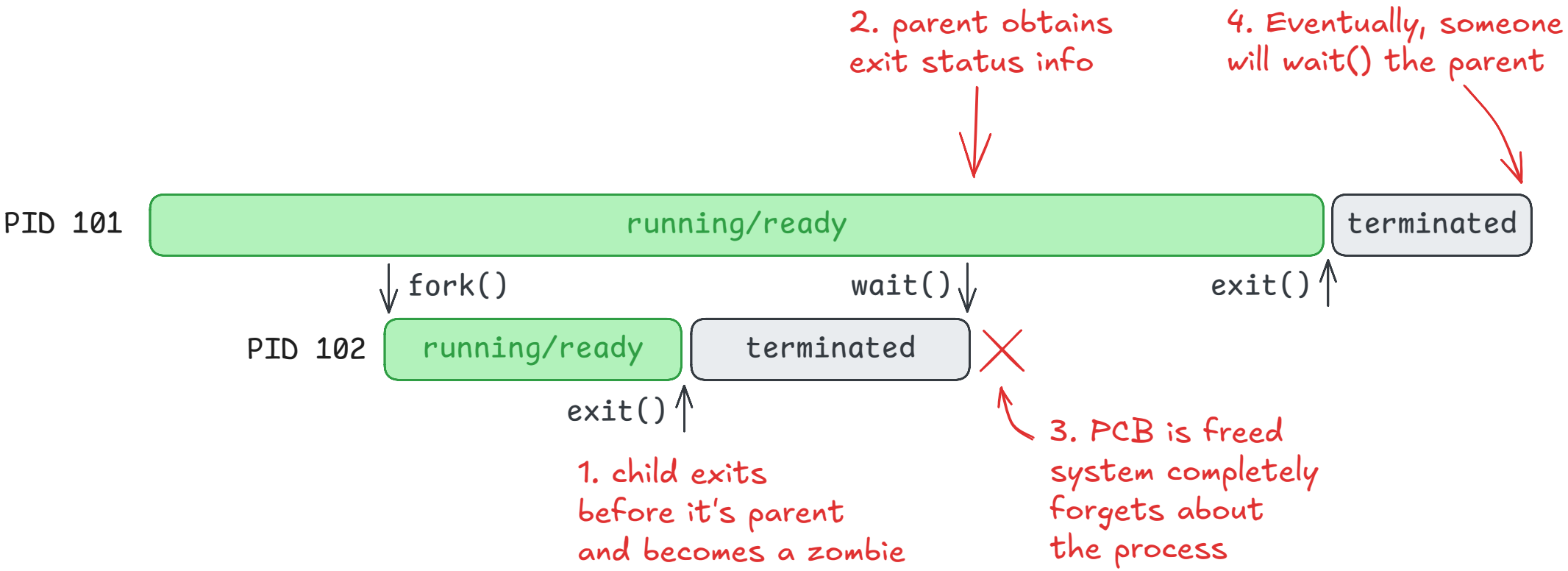
```
> man 3p exit
```



When this exited PCB will die?

```
> man 3p wait
```

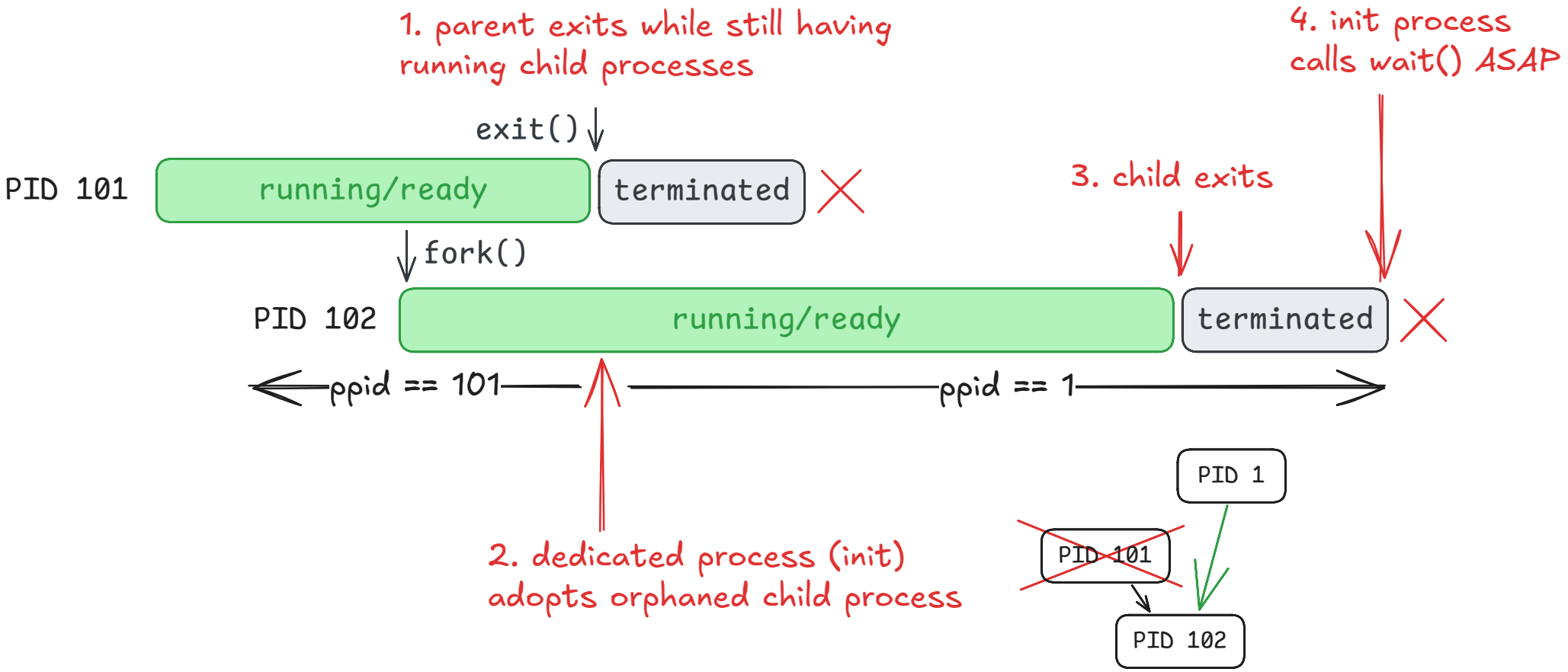
Parent must have a chance to consume the exit status



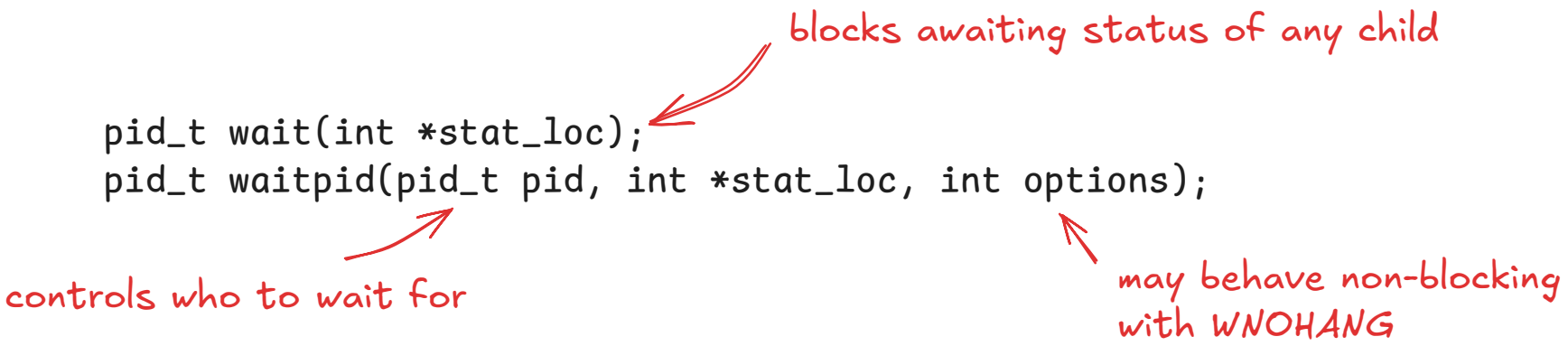
What if parent exits before the child without calling wait()?

Orphaned processes

If parent dies prematurely OS automatically reparents its children



Decoding exit status information



What is returned via stat_loc? → termination reason (exited, signaled or stopped)
+ reason dependent info (exit code or lethal signal or stopping signal)

```
WIFEXITED(status) # True if child exited normally
WIFSIGNAED(status) # True if child terminated due to signal
WIFSTOPPED(status) # True if child has stopped
```

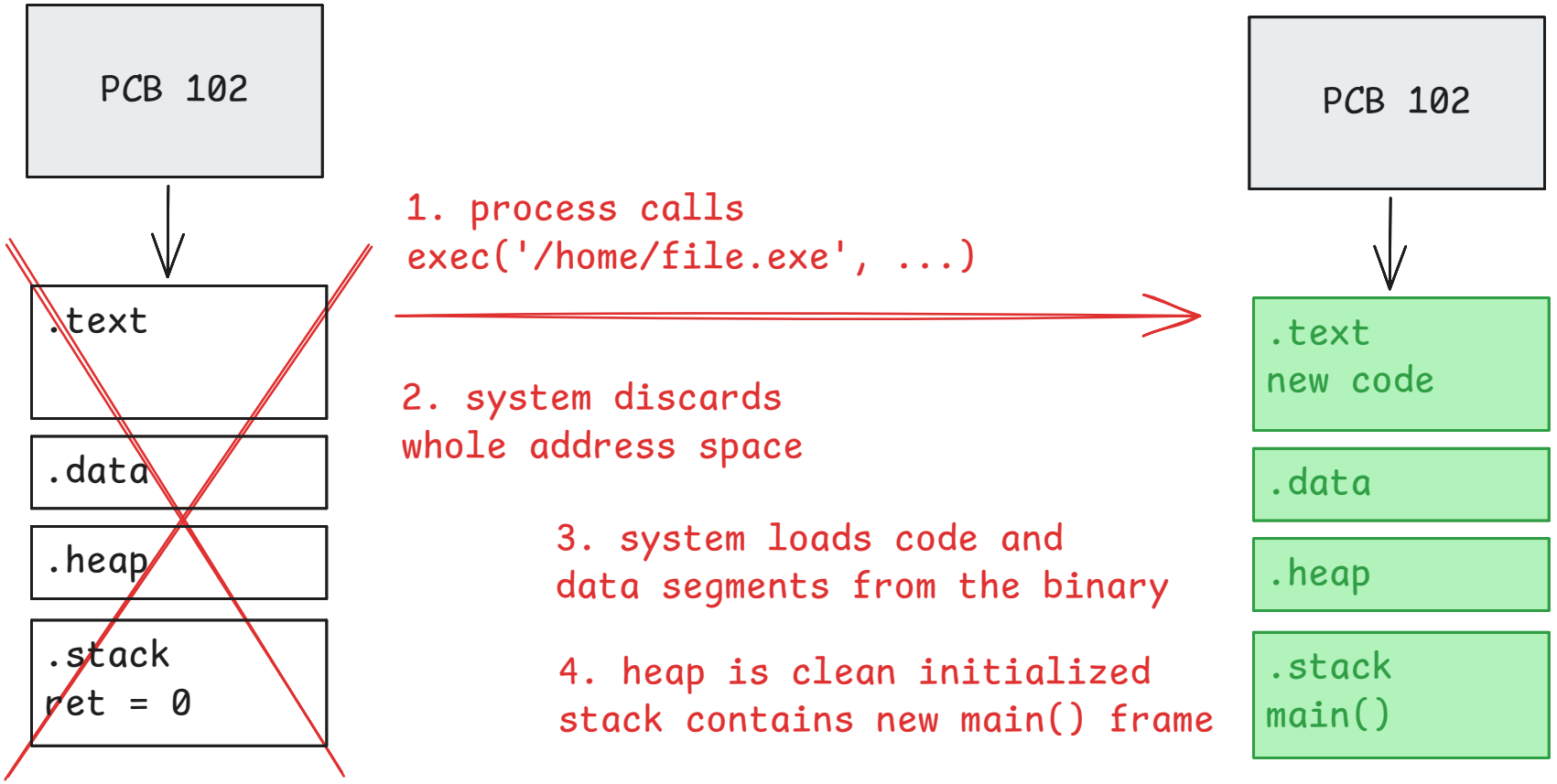

```
WIFEXITED(status) # True if child exited normally
WIFSIGNAED(status) # True if child terminated due to signal
WIFSTOPPED(status) # True if child has stopped
```

Executing different code

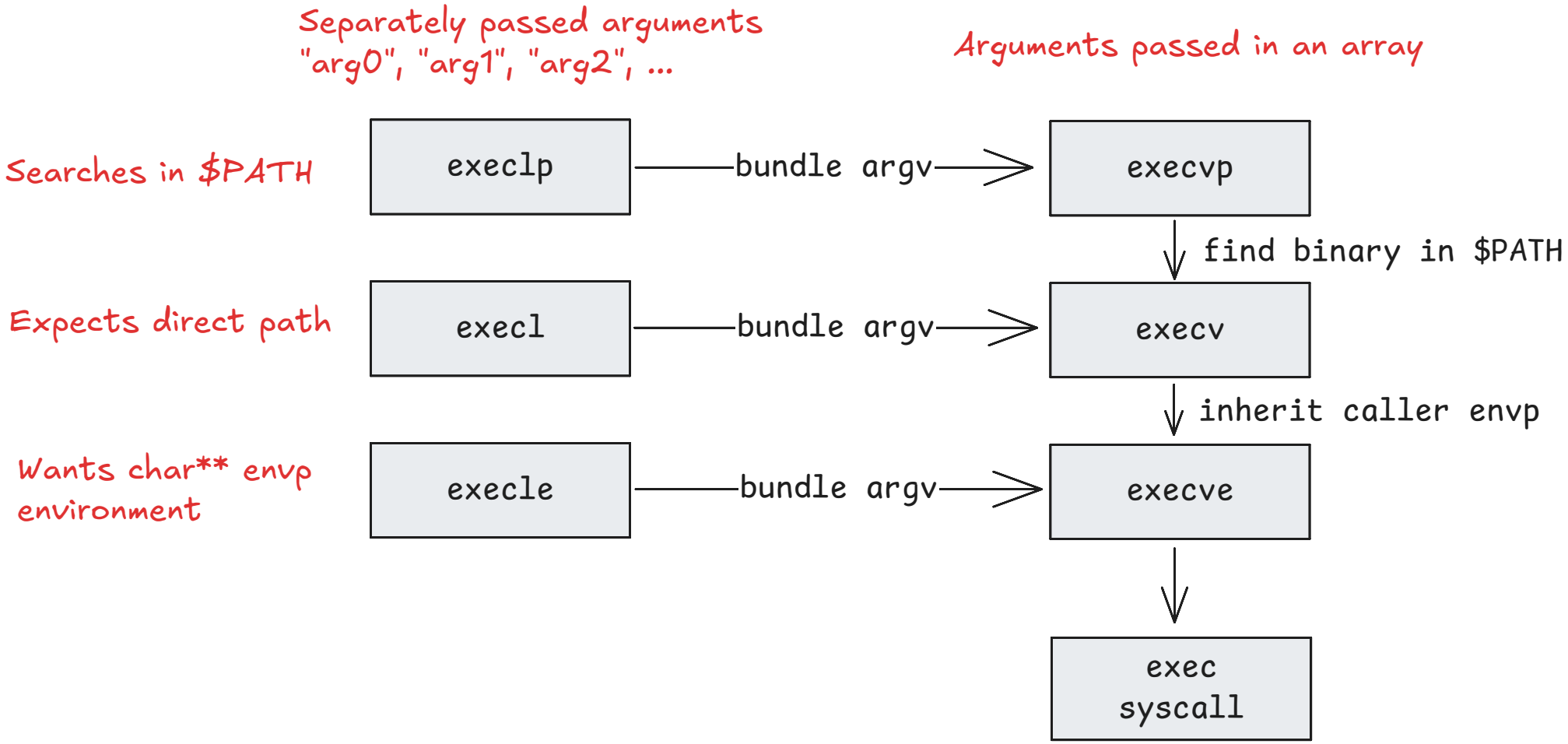
```
> man 3p exec
```

Meet the exec() syscall

With fork child has the same code as parent. exec() provides means of running a different binary.

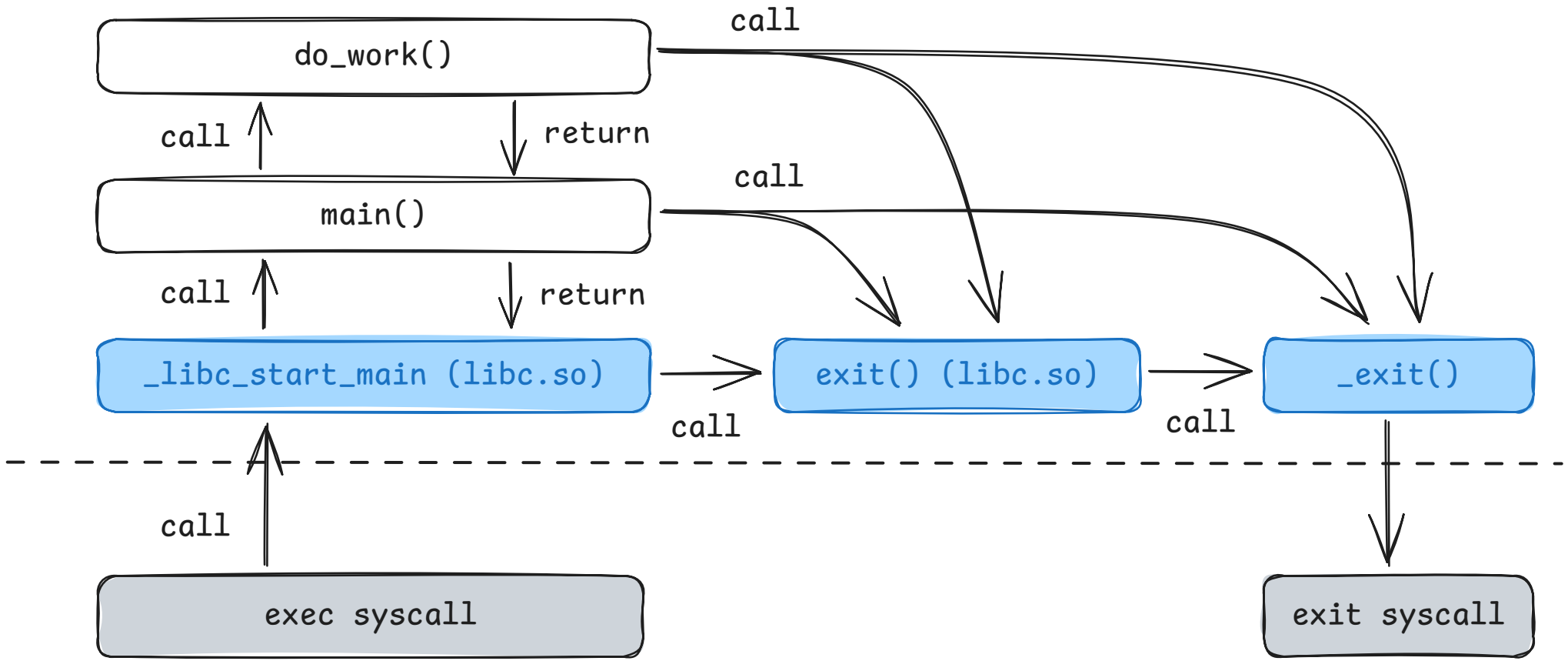


The exec() flavours



What happens after exit() call?

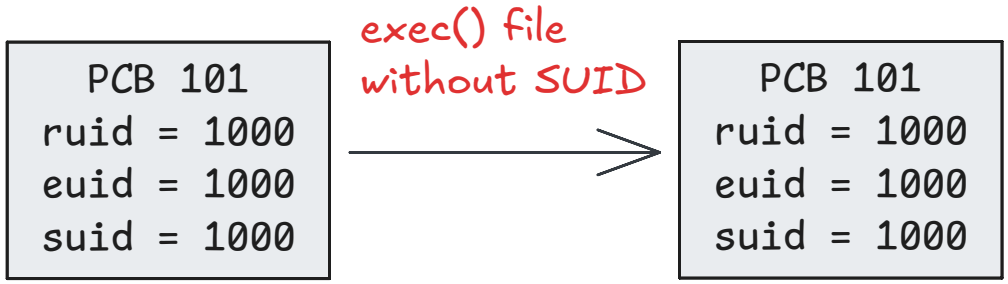
And before the main()?



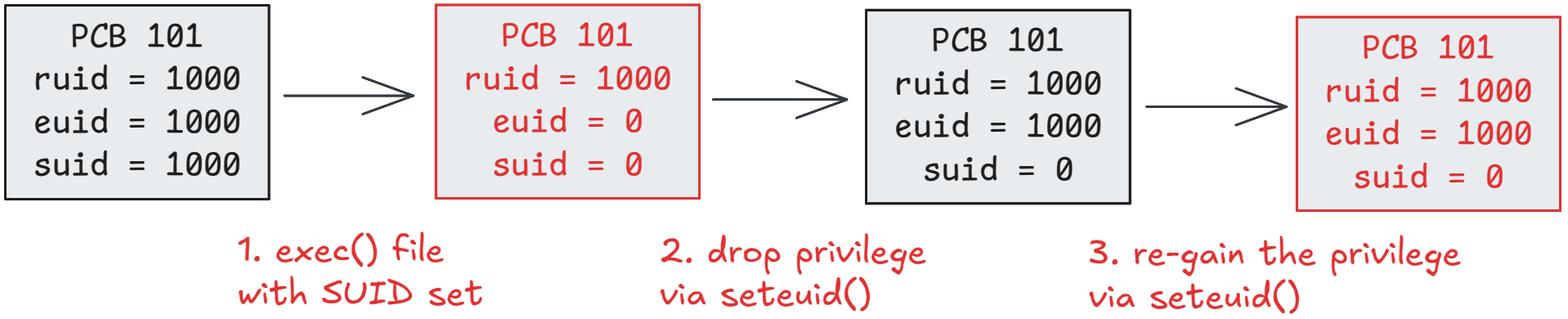
The process user ID

```
> man 3p seteuid
```

Normally during exec process user ID remains unchanged

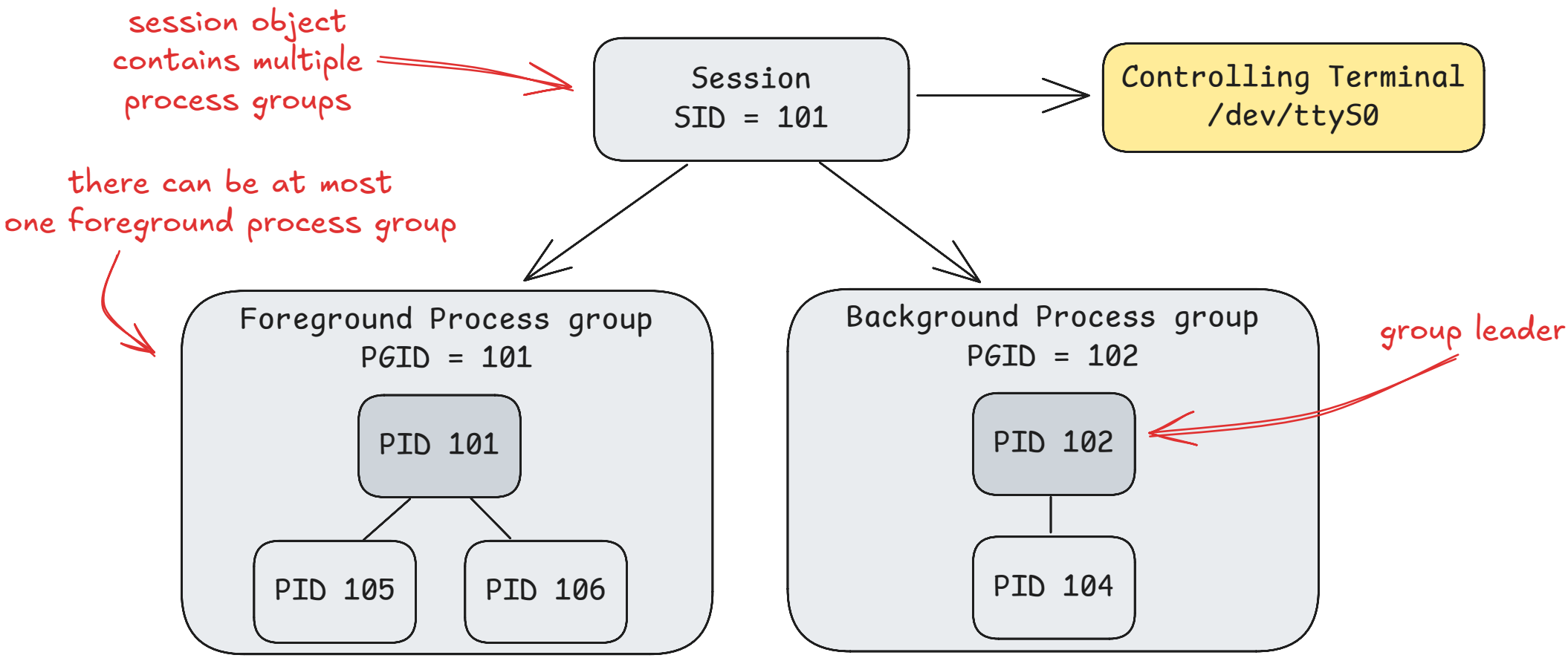


One can exec a program with SUID bit set in the mode filed of the inode to get privileges of the file owner



Process groups

```
> man 3p setpgid
> man 3p setsid
```



After fork() child inherits process group ID and session ID - this can be changed later on.

Fork alternatives

Advanced tools for creating new processes

```
> man 2 vfork
> man 2 clone
```

```
pid_t vfork(void);
```

Faster process creation for cases where child immediately exec()'s a different binary.
In contrast to fork() child process shares the address space (no copy overhead).
Parent is stopped until child calls exec() or _exit().

```
int clone(int (*fn)(void *), void *stack, int flags, void *arg, ...
/* pid_t *parent_tid, void *tls, pid_t *child_tid */ );
```

Provides fine-grained control over what is shared and what is not via flags bitfield.

```
CLONE_PARENT - sharing PPID of the parent process
CLONE_FS - sharing file system root, current directory, umask
CLONE_FILES - sharing the file descriptor table
CLONE_SIGHAND - sharing signal table
CLONE_VM - sharing virtual memory
CLONE_VFORK - stops the parent process, until the child terminates
```