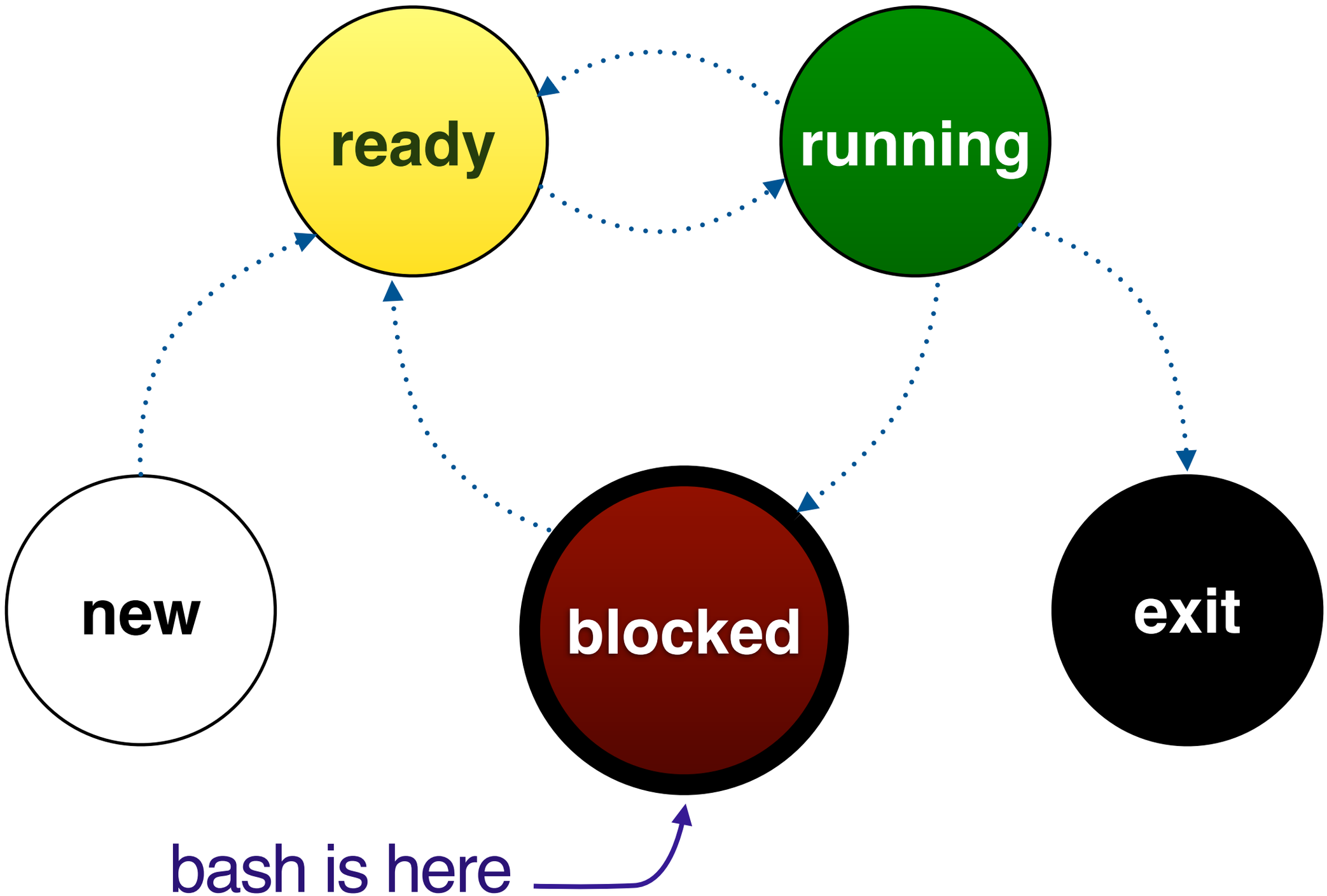
Lecture 12

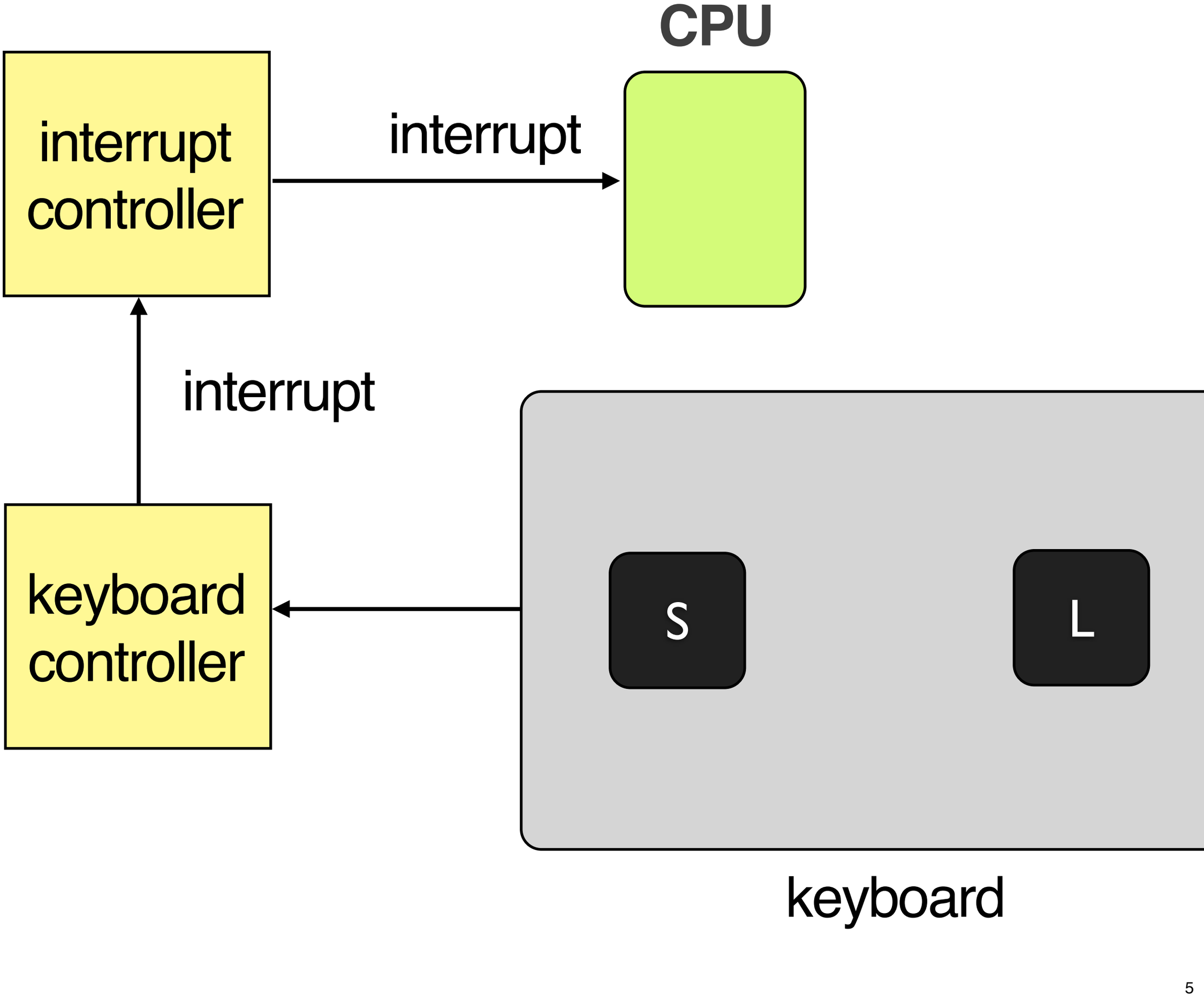
Putting Everything Together

9 November, 2010

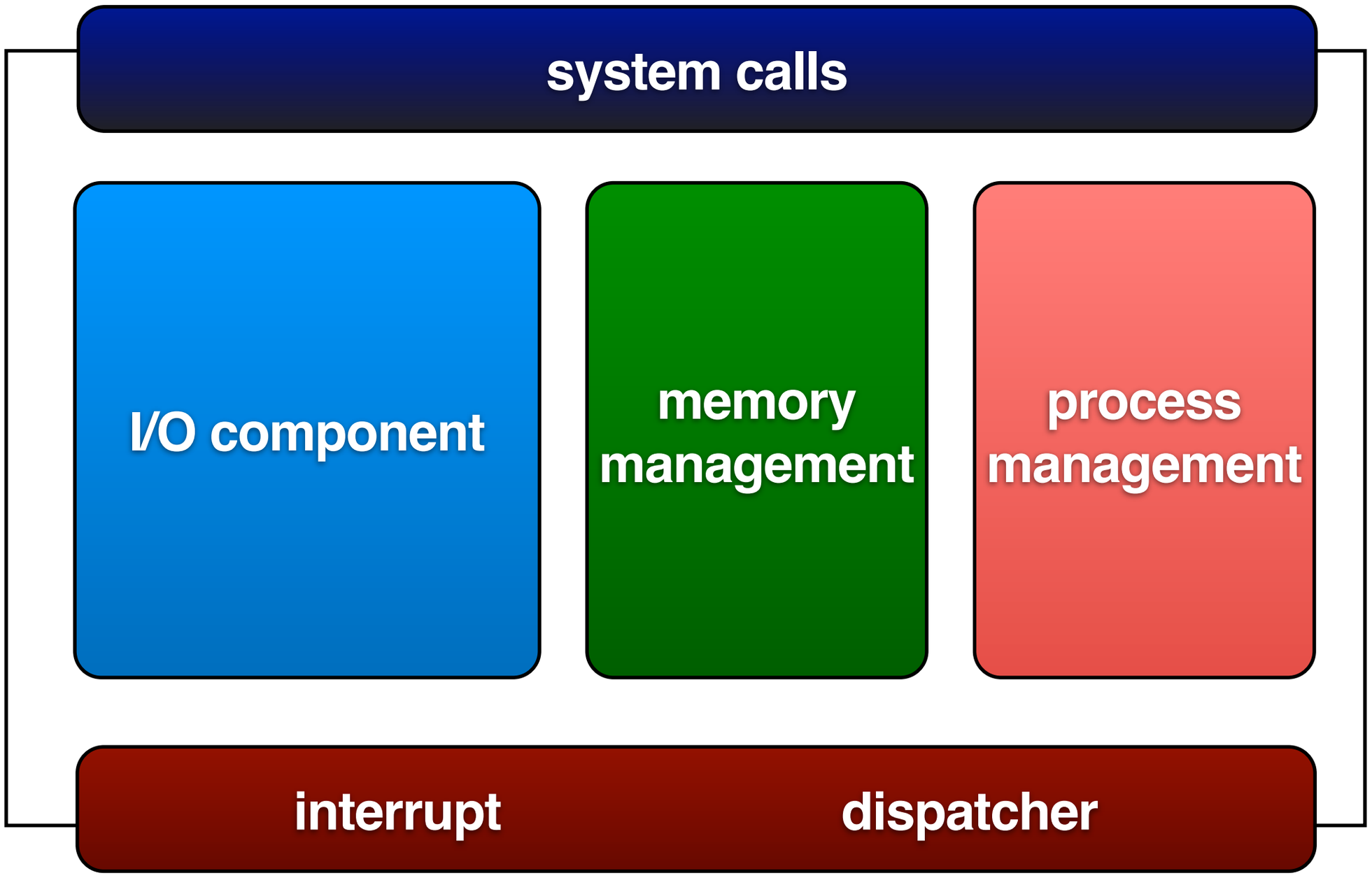
```
bash$ ls
foo bar a.out lab.c
bash$
```

bash$


ready
running
new
blocked
exit
bash is here


CPU
interrupt controller
interrupt
interrupt
keyboard controller
S
L
keyboard

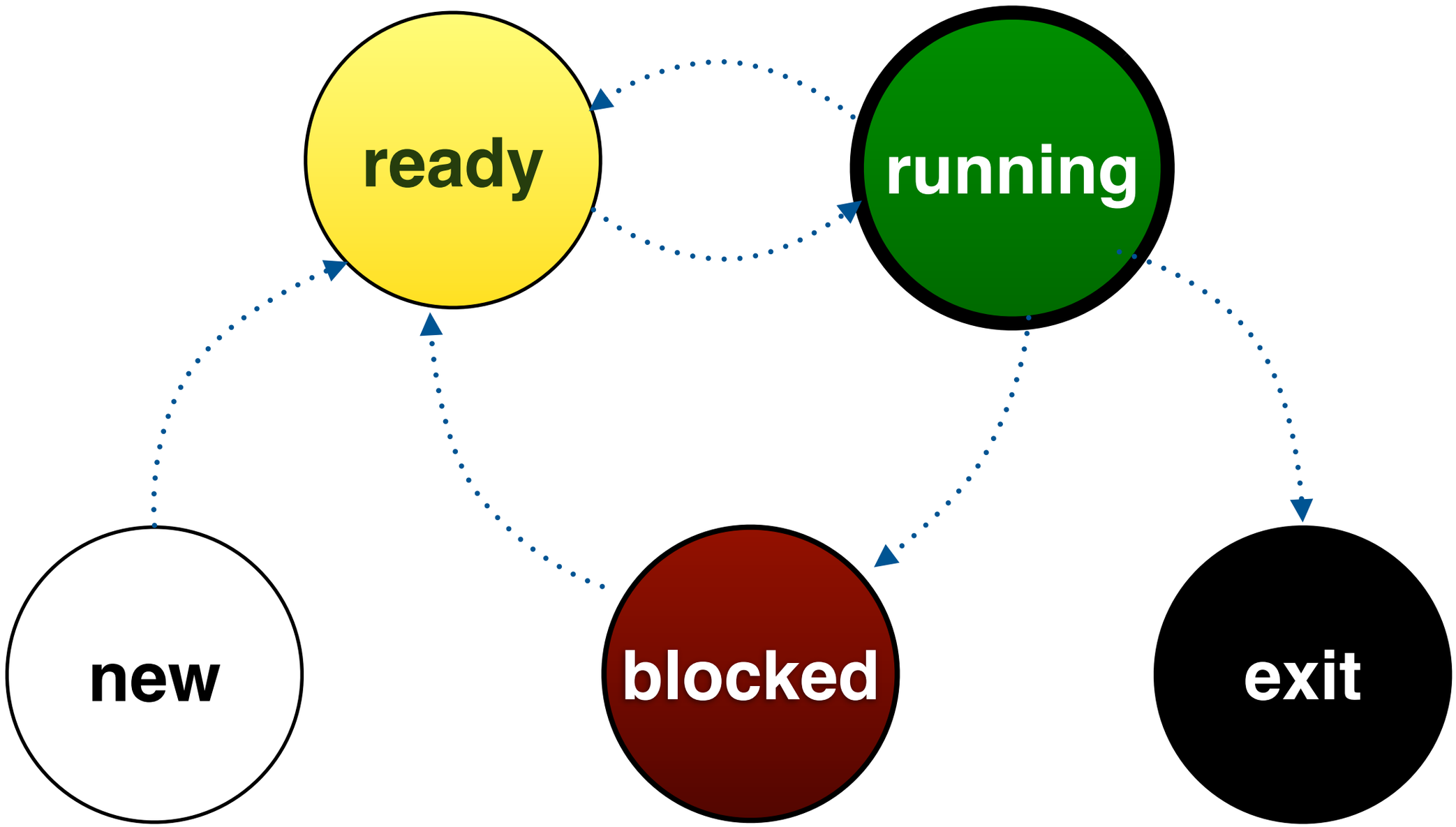
user program

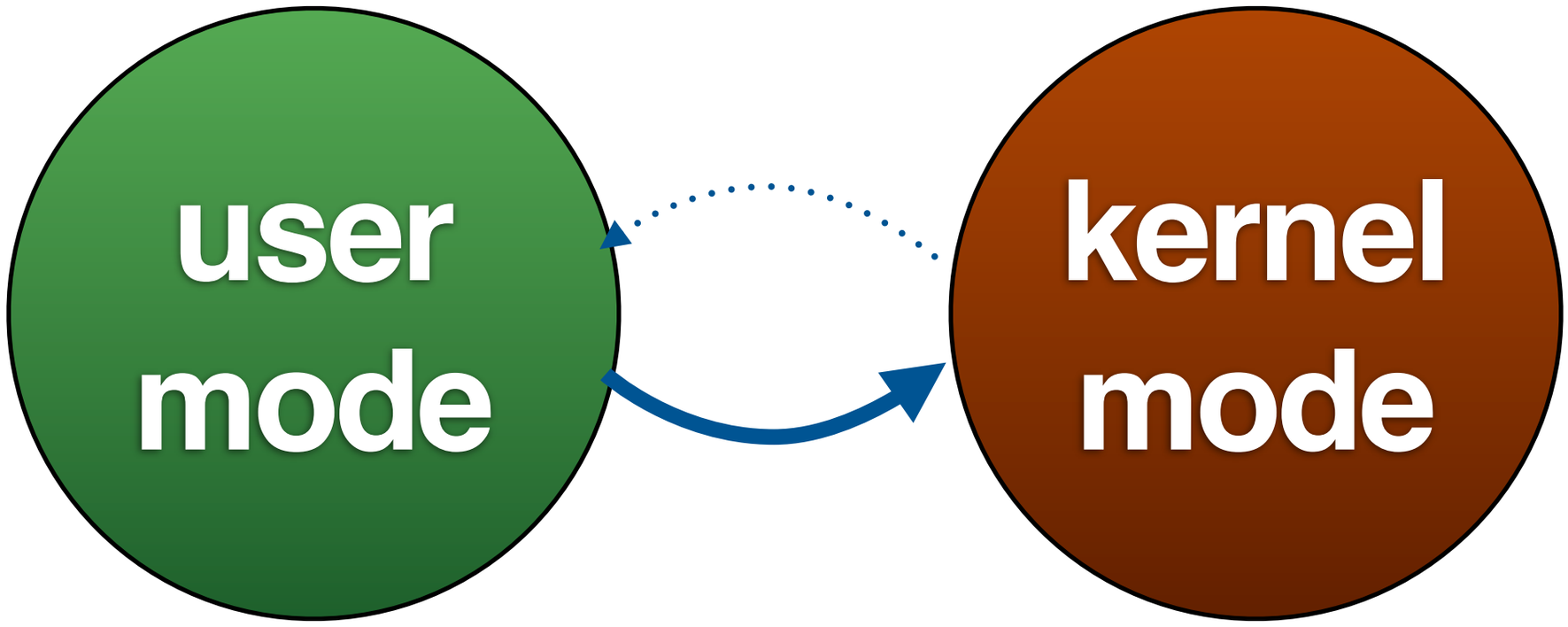


hardware

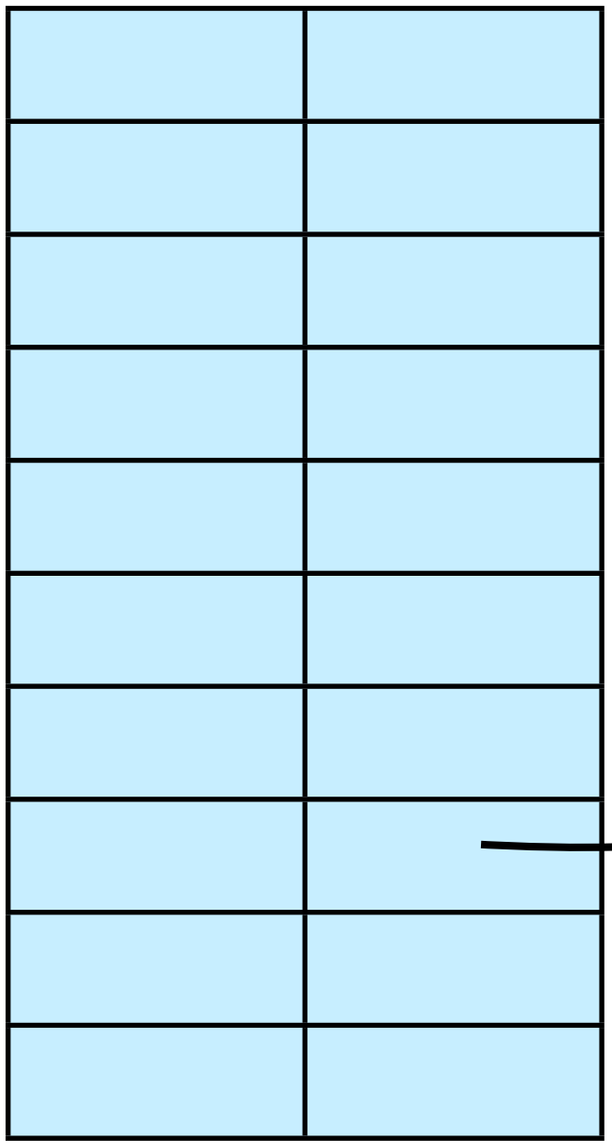
CPU

some process running




user
mode
kernel
mode

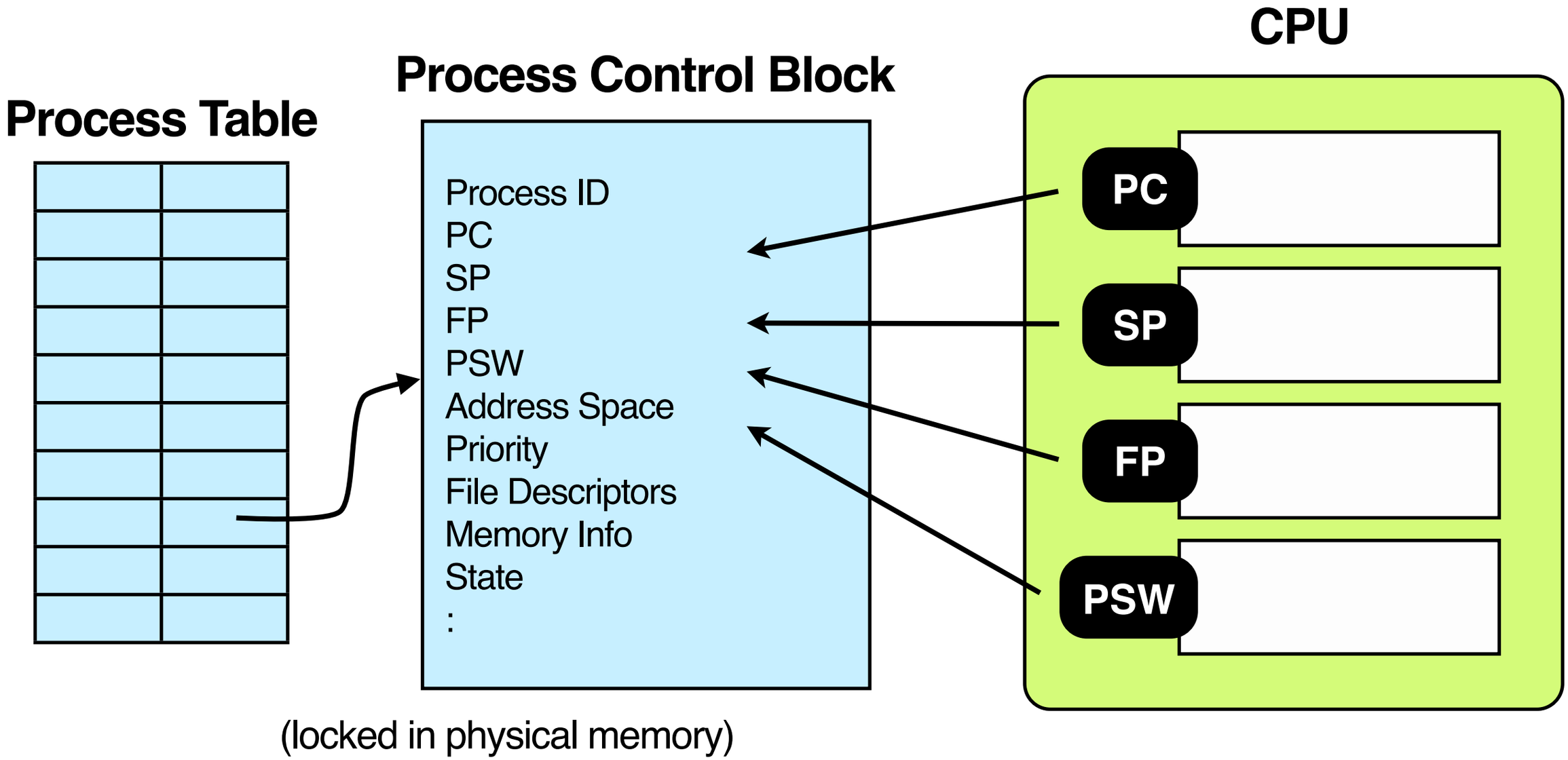
Process Table



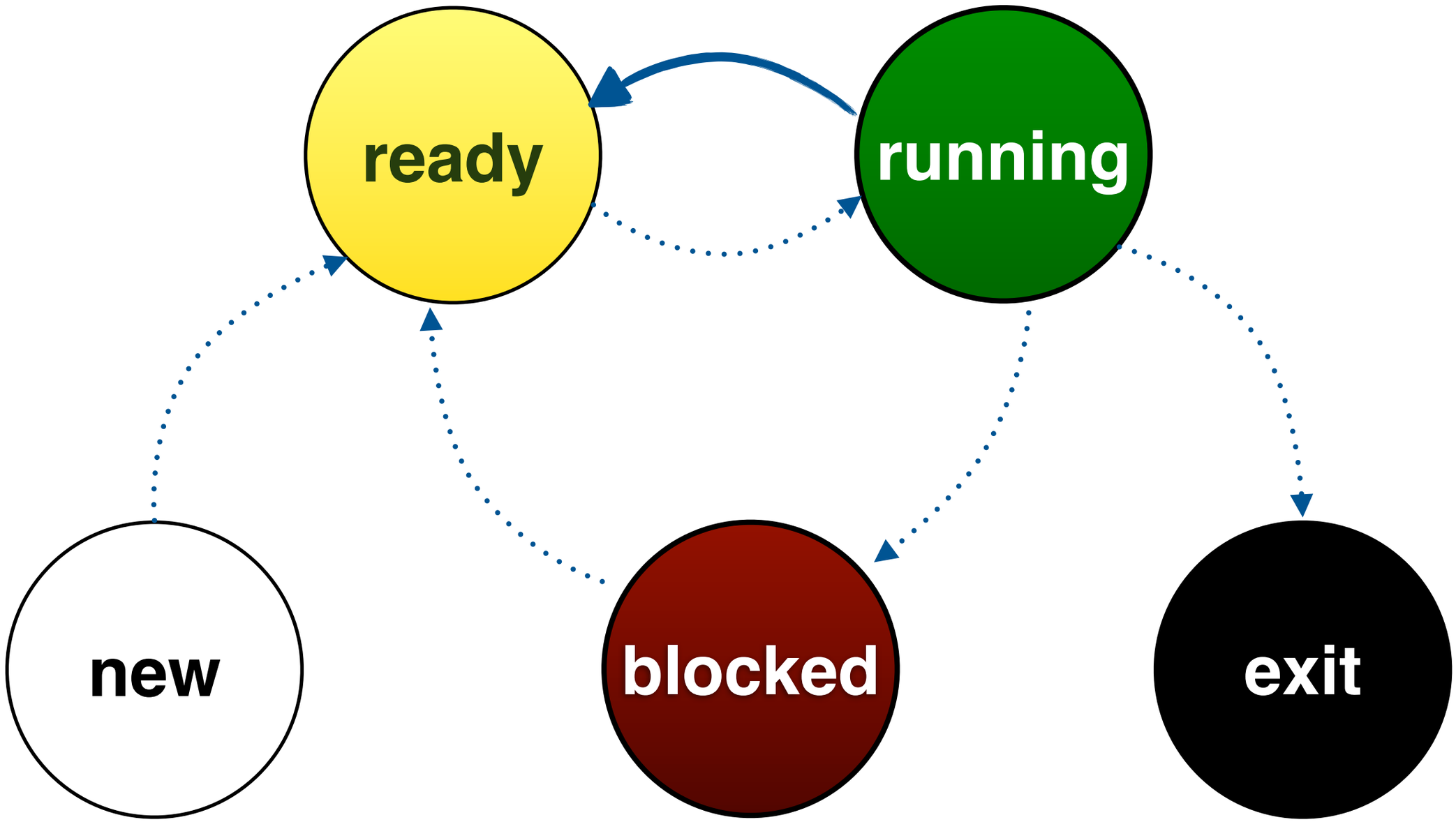
Process Control Block

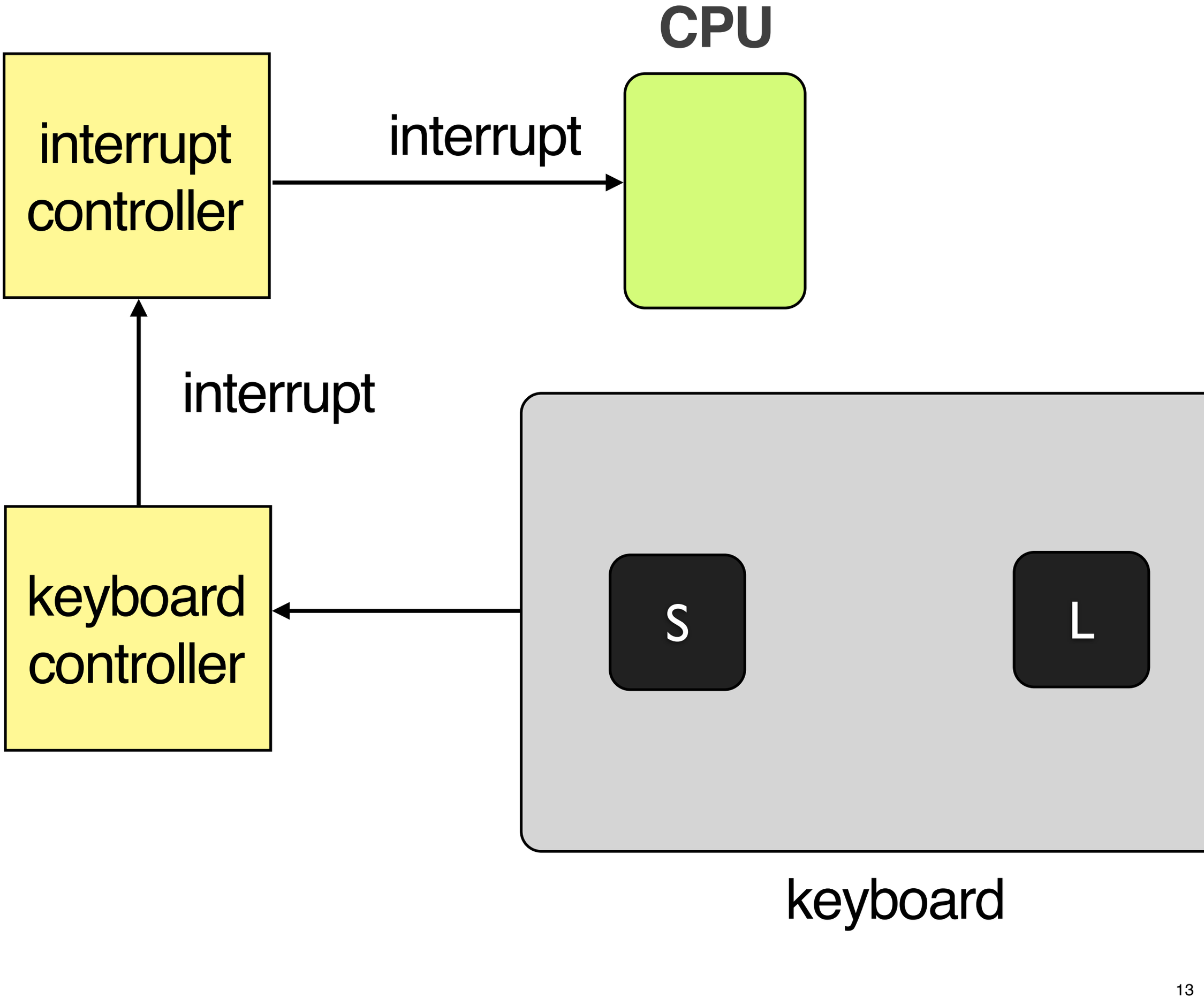
Process ID
PC
SP
FP
PSW
Address Space
Priority
File Descriptors
Memory Info
State

(locked in physical memory)

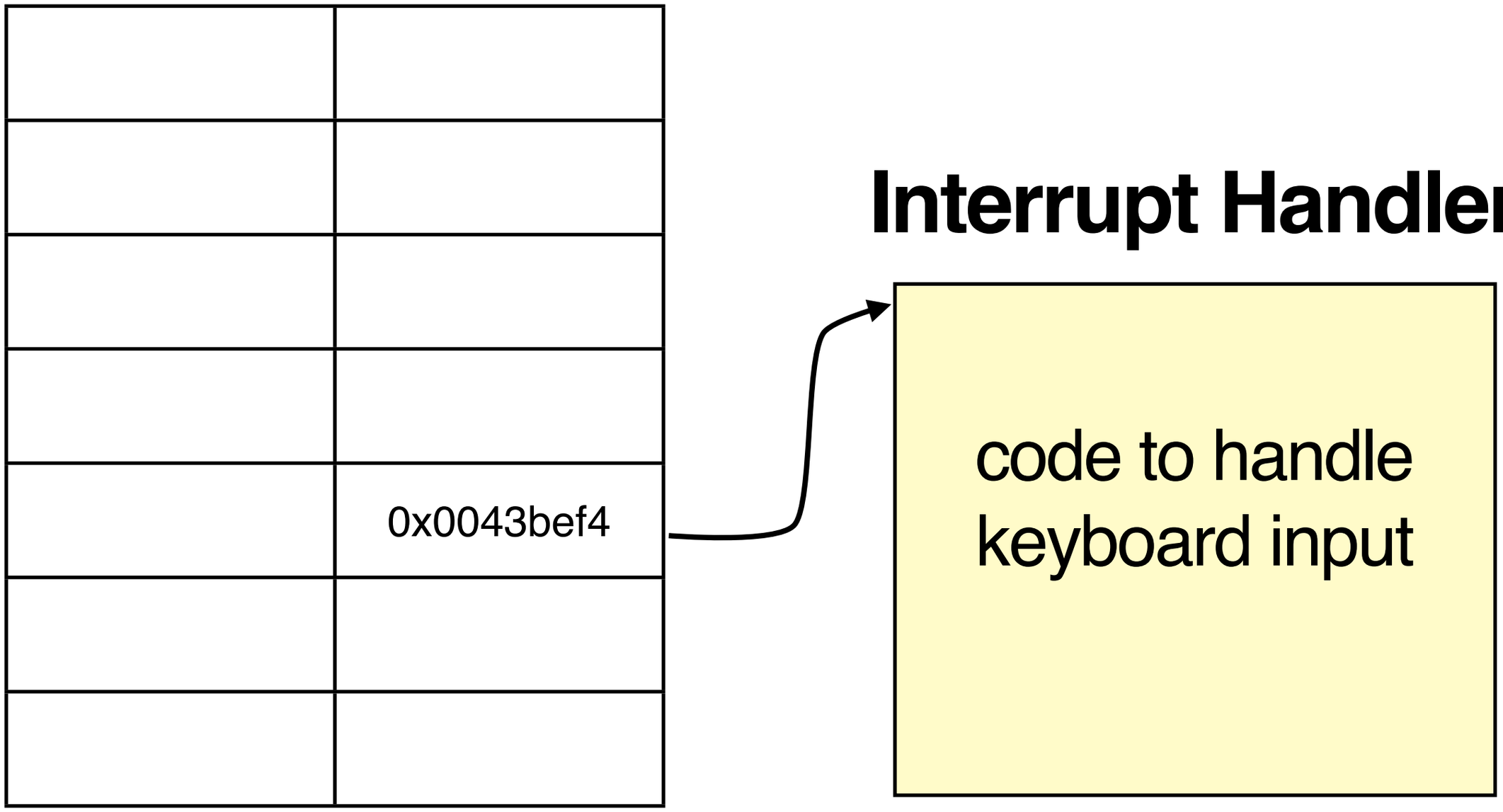


(locked in physical memory)

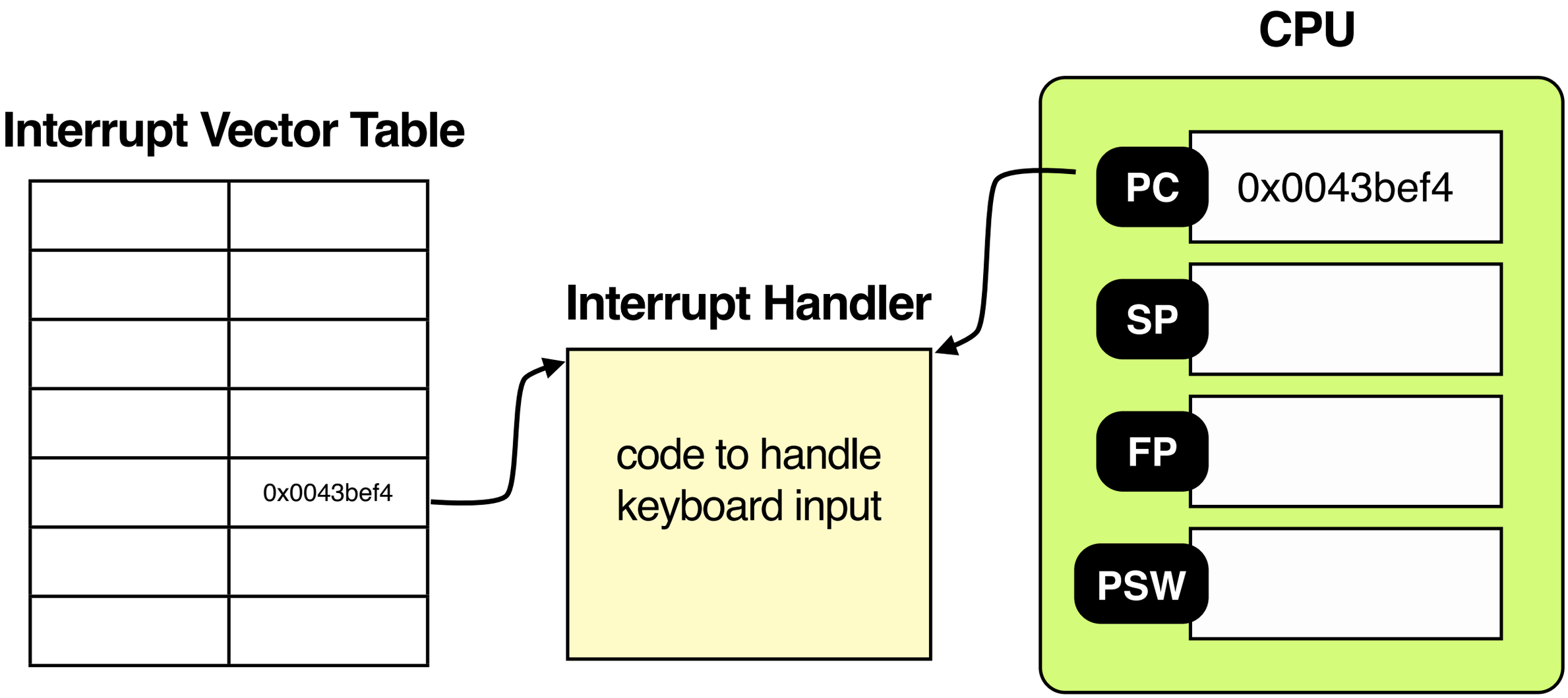

ready
running
new
blocked
exit

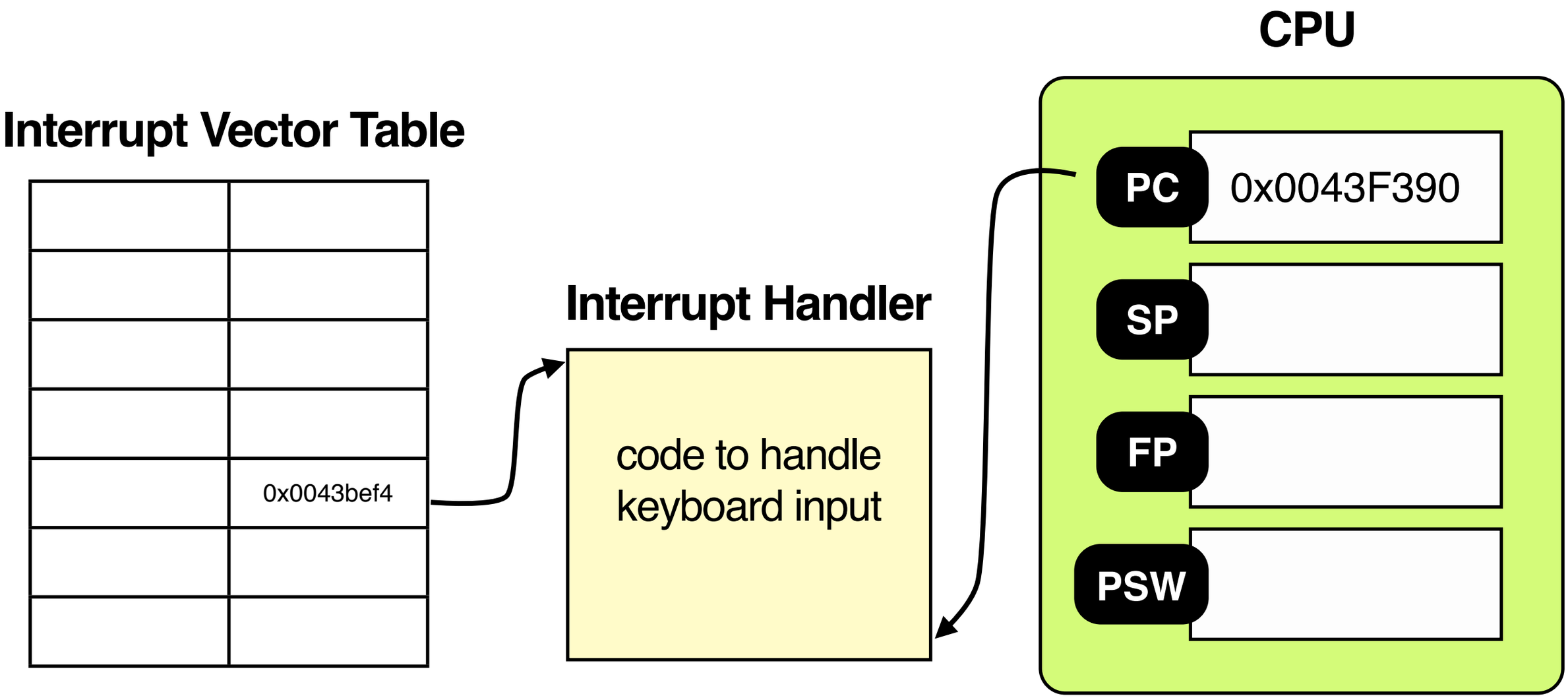

CPU
interrupt controller
interrupt
interrupt
keyboard controller
S
L
keyboard

Interrupt Vector Table



(locked in physical memory)


Interrupt Vector Table
0x0043bef4
Interrupt Handler
code to handle
keyboard input
CPU
PC
0x0043bef4
SP
FP
PSW


CPU
Interrupt Vector Table
0x0043bef4
Interrupt Handler
code to handle
keyboard input
PC
0x0043F390
SP
FP
PSW

browser

calendar

...

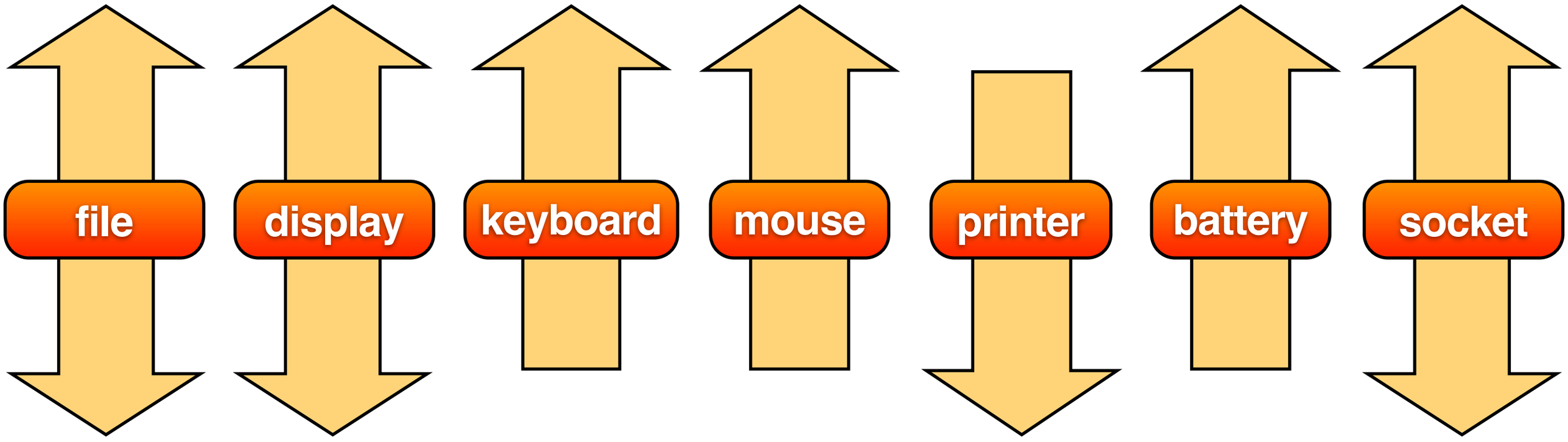
media player

compilers

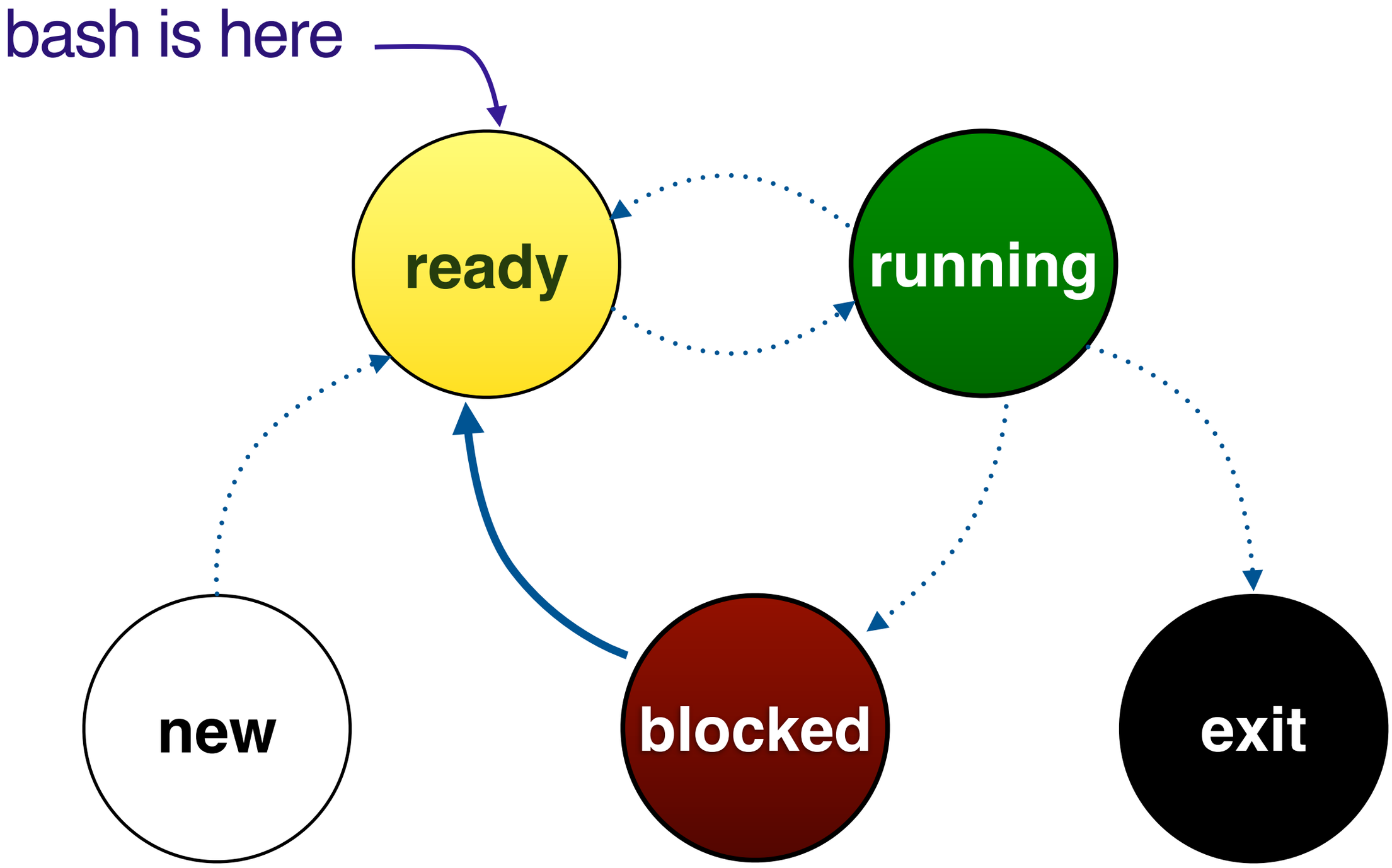
editors

...

shell

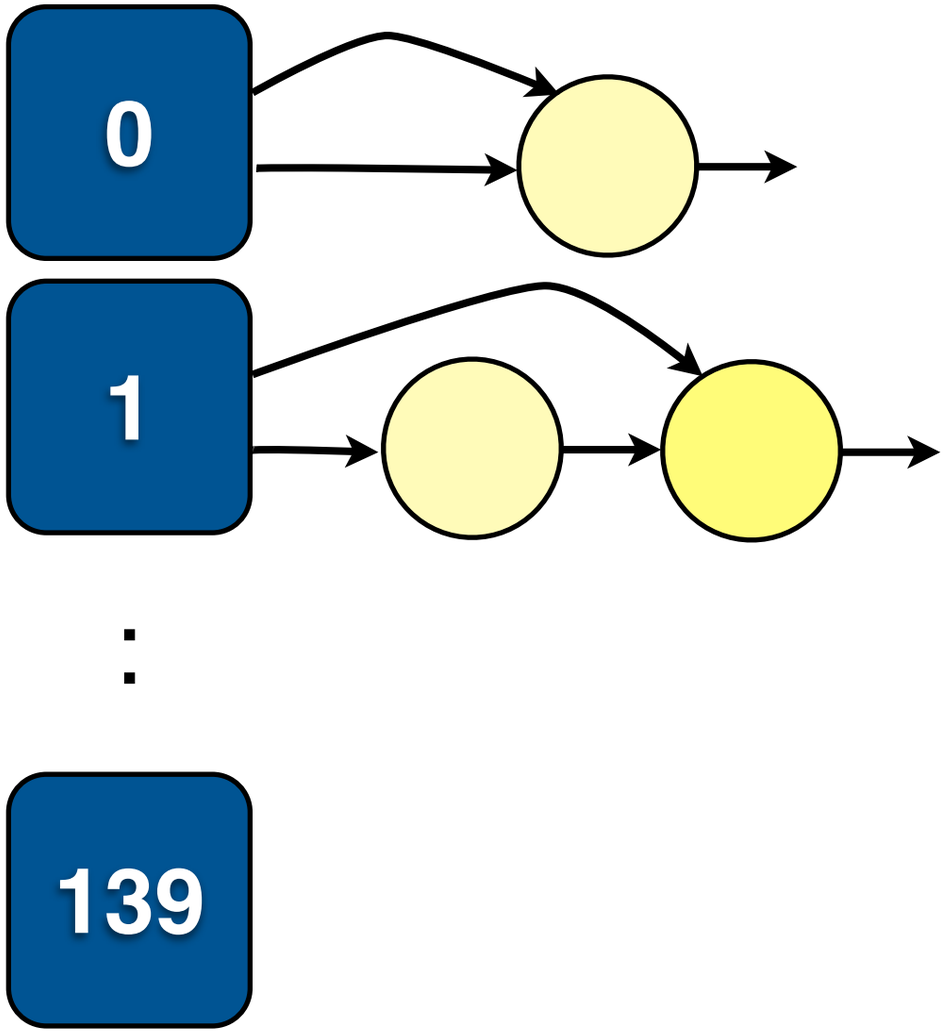


operating system

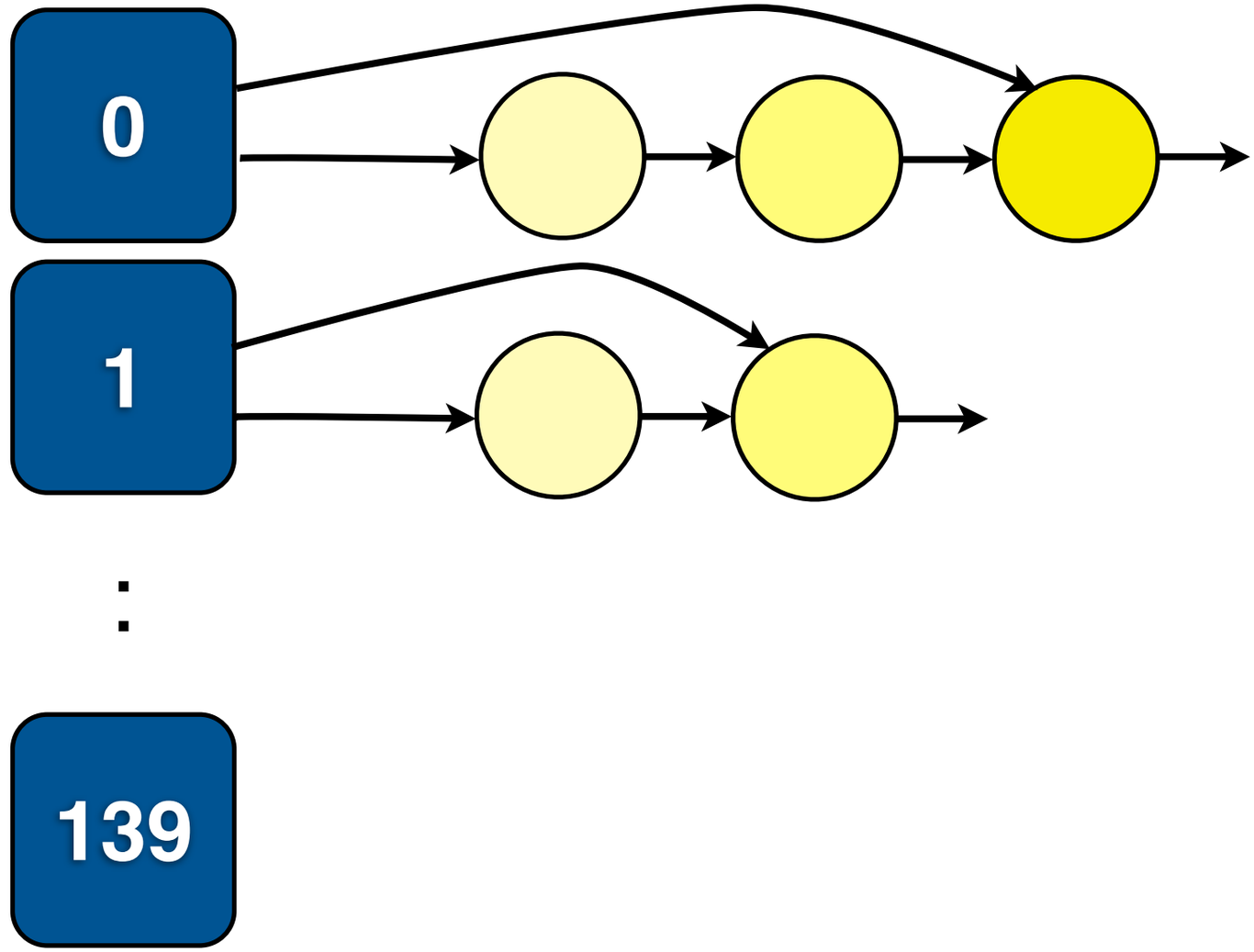

bash is here
ready
running
new
blocked
exit

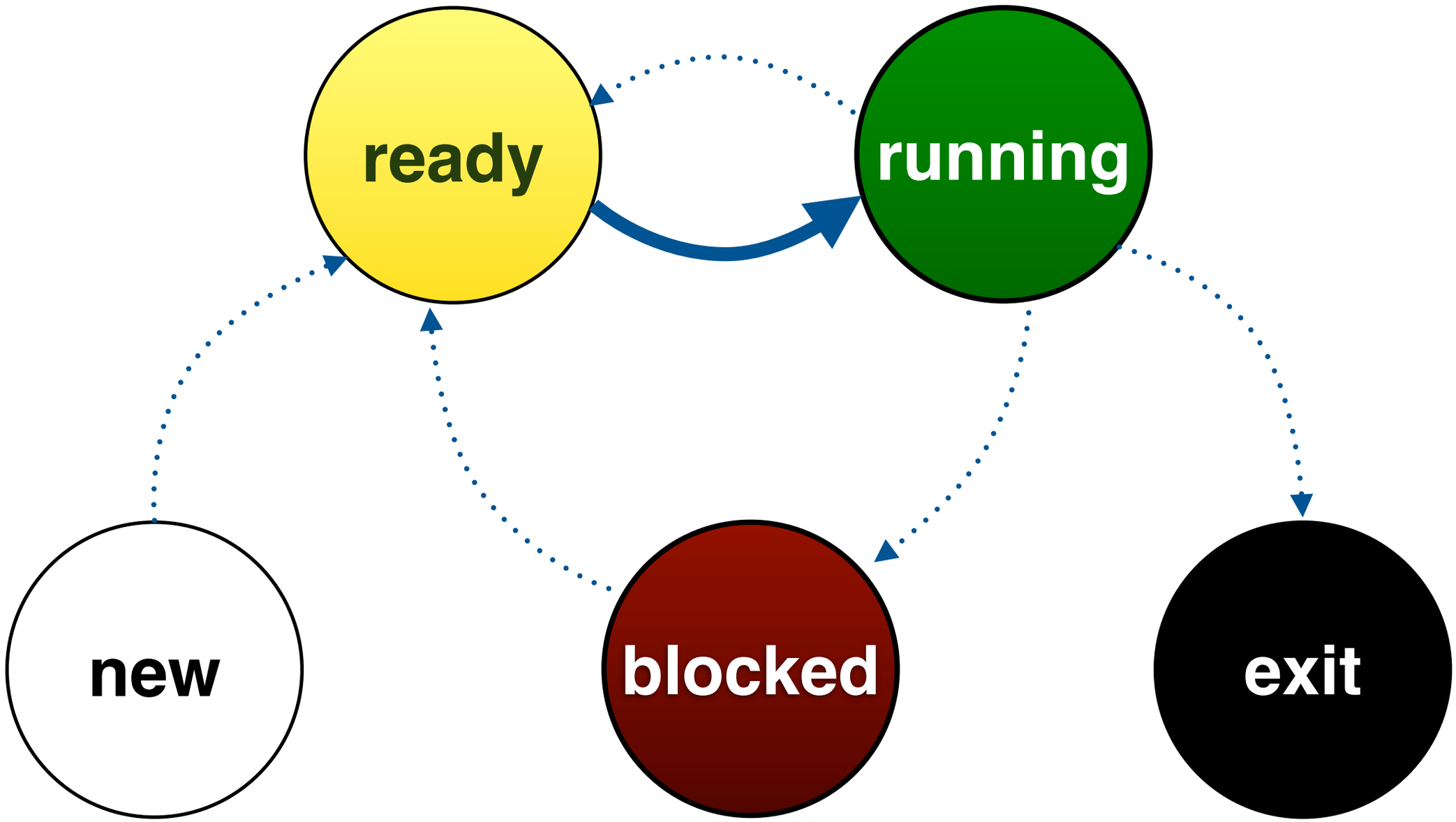
who to run next?

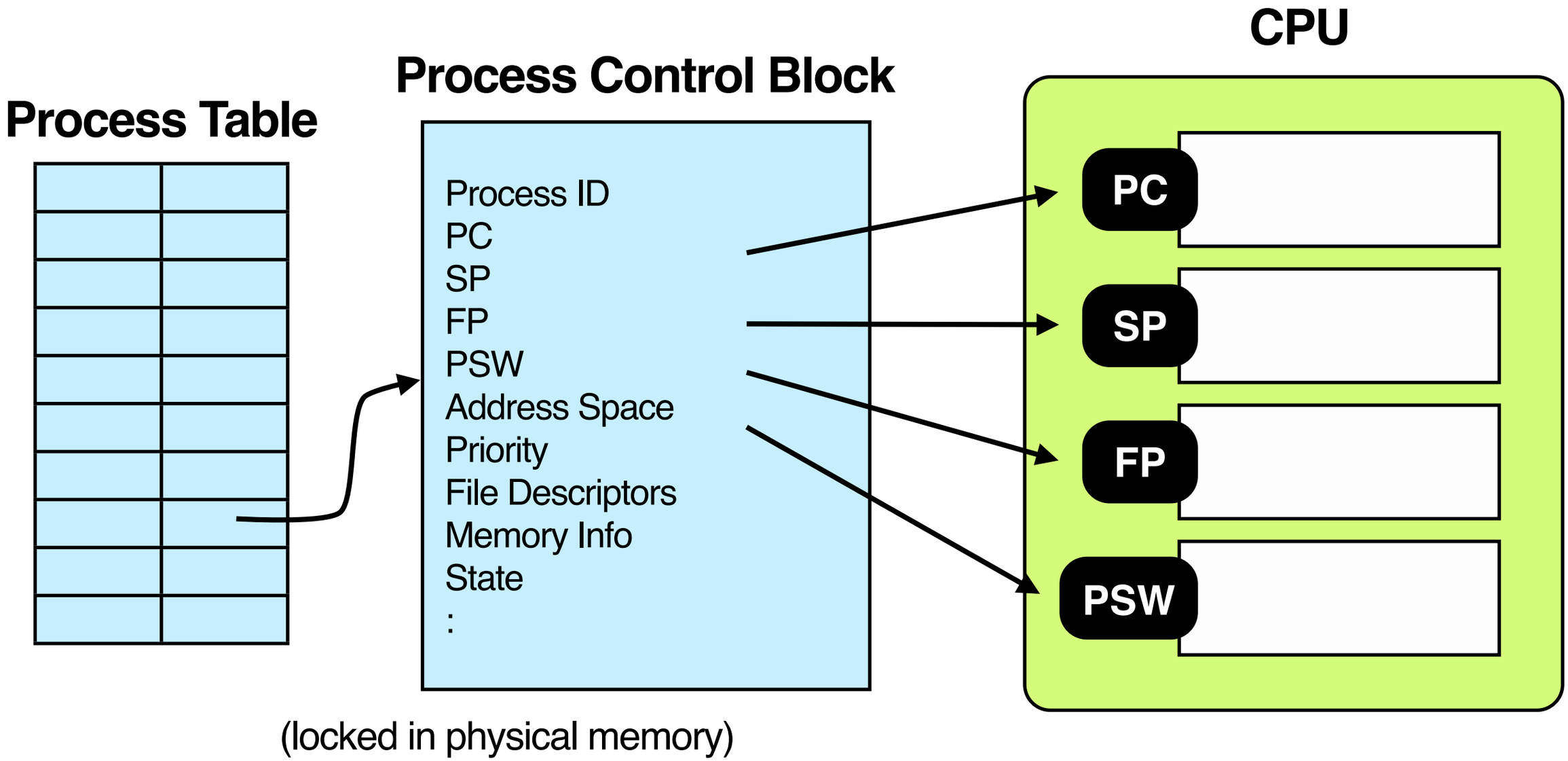
Active



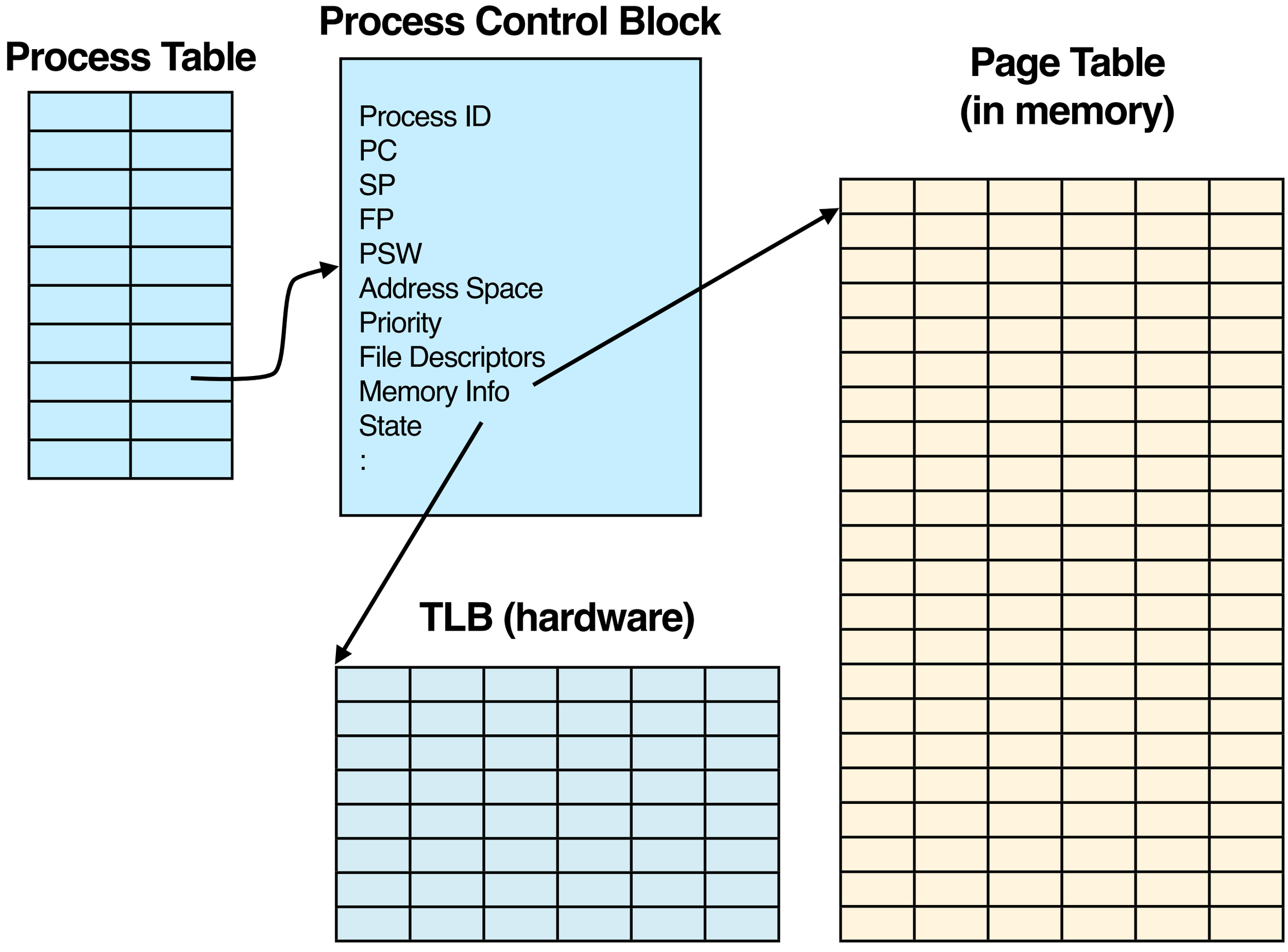
Expired

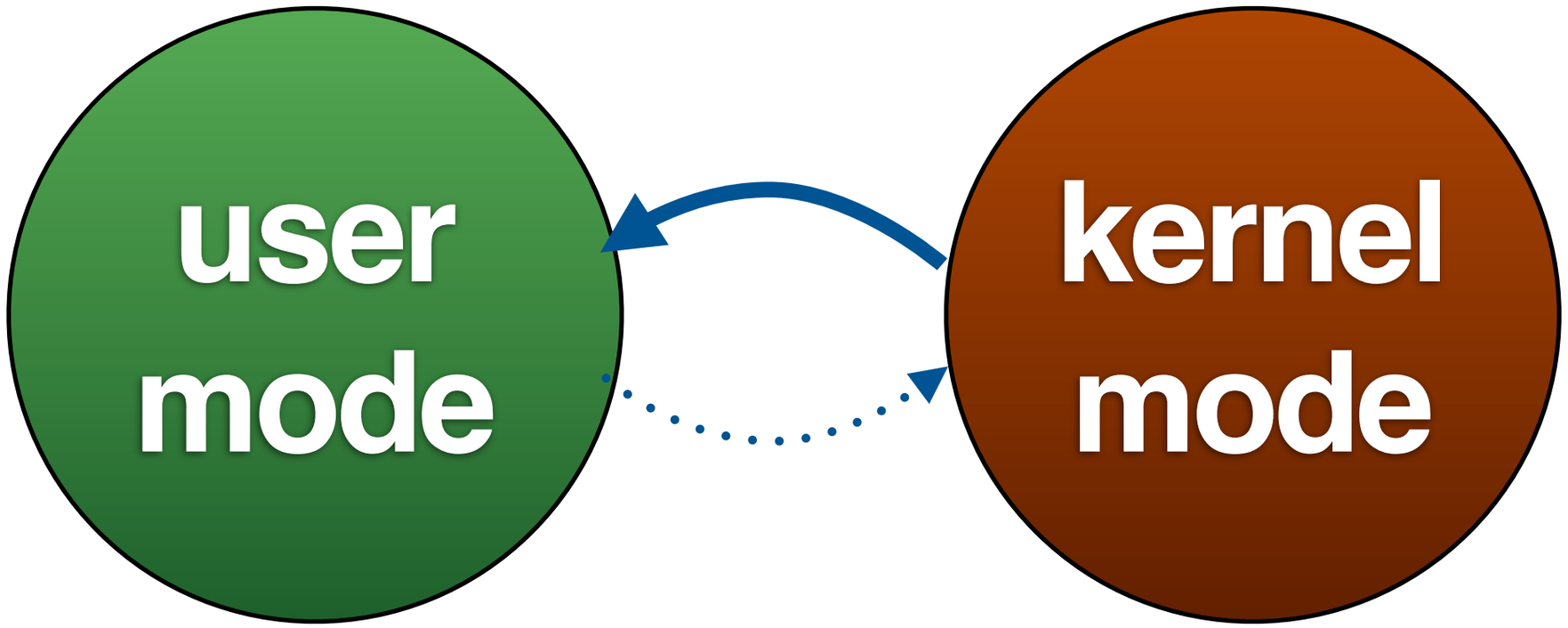



ready
running
new
blocked
exit



(locked in physical memory)


Process Table
Process Control Block
Process ID
PC
SP
FP
PSW
Address Space
Priority
File Descriptors
Memory Info
State
:
Page Table
(in memory)
TLB (hardware)


user
mode
kernel
mode

bash$

```
bash$ l
```

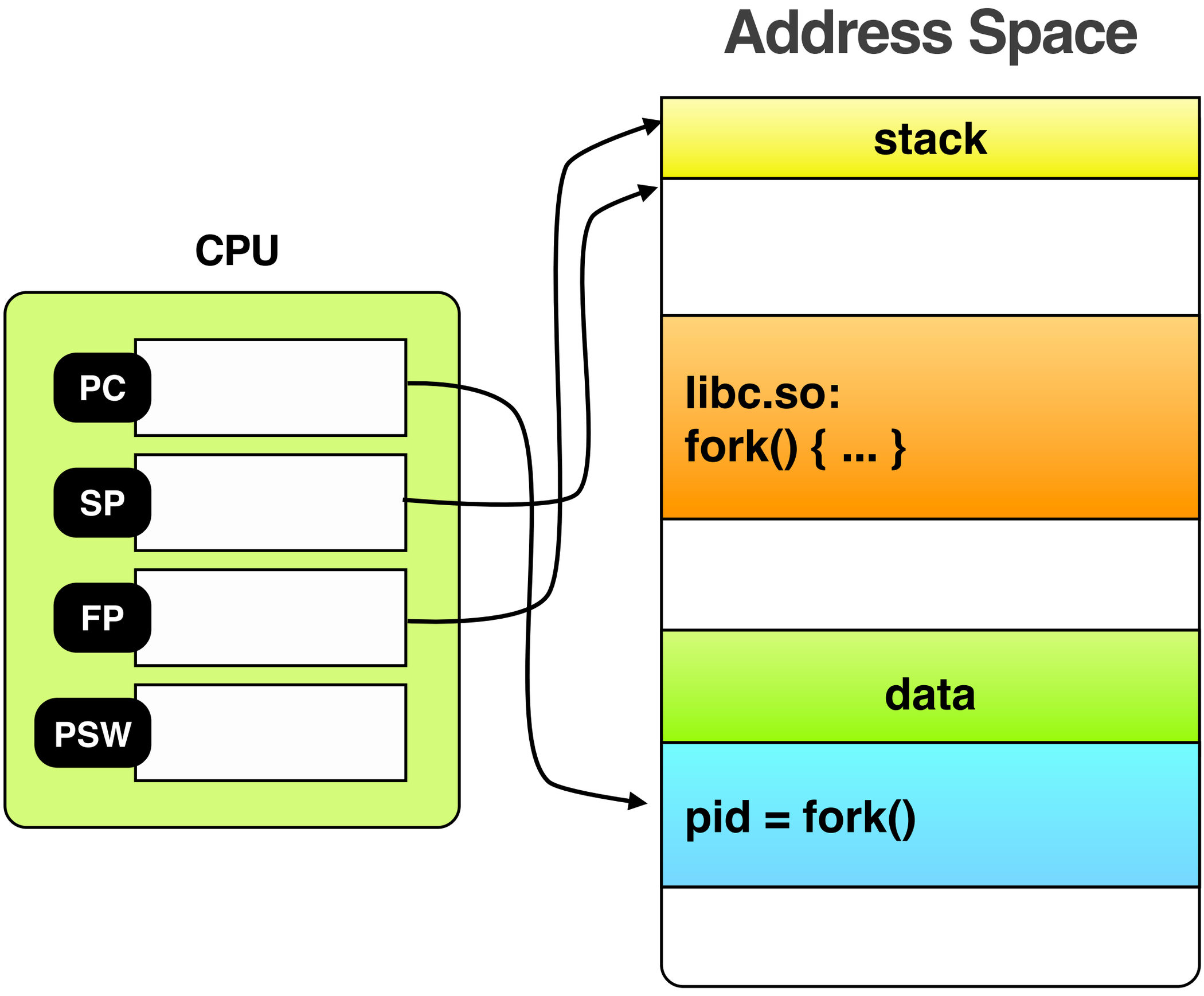
bash$ ls↵

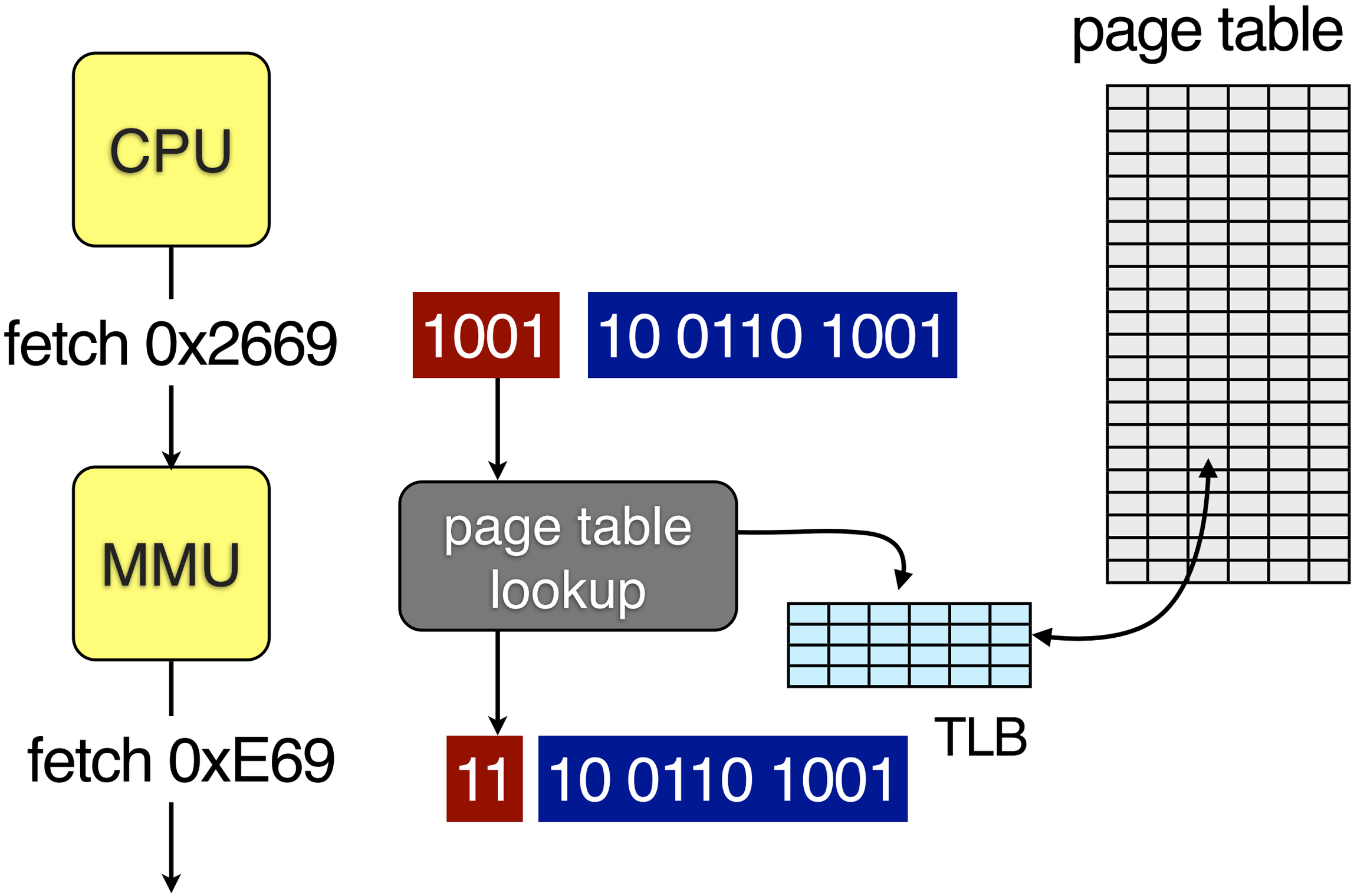
(repeat for ‘s’, ‘return’)

bash$ ls↵

(repeat for ‘s’, ‘return’)

fork()


Address Space
CPU
PC
SP
FP
PSW
stack
libc.so:
fork() { ... }
data
pid = fork()


CPU
fetch 0x2669
MMU
fetch 0xE69
1001
10 0110 1001
page table lookup
11
10 0110 1001
page table
TLB

NRU

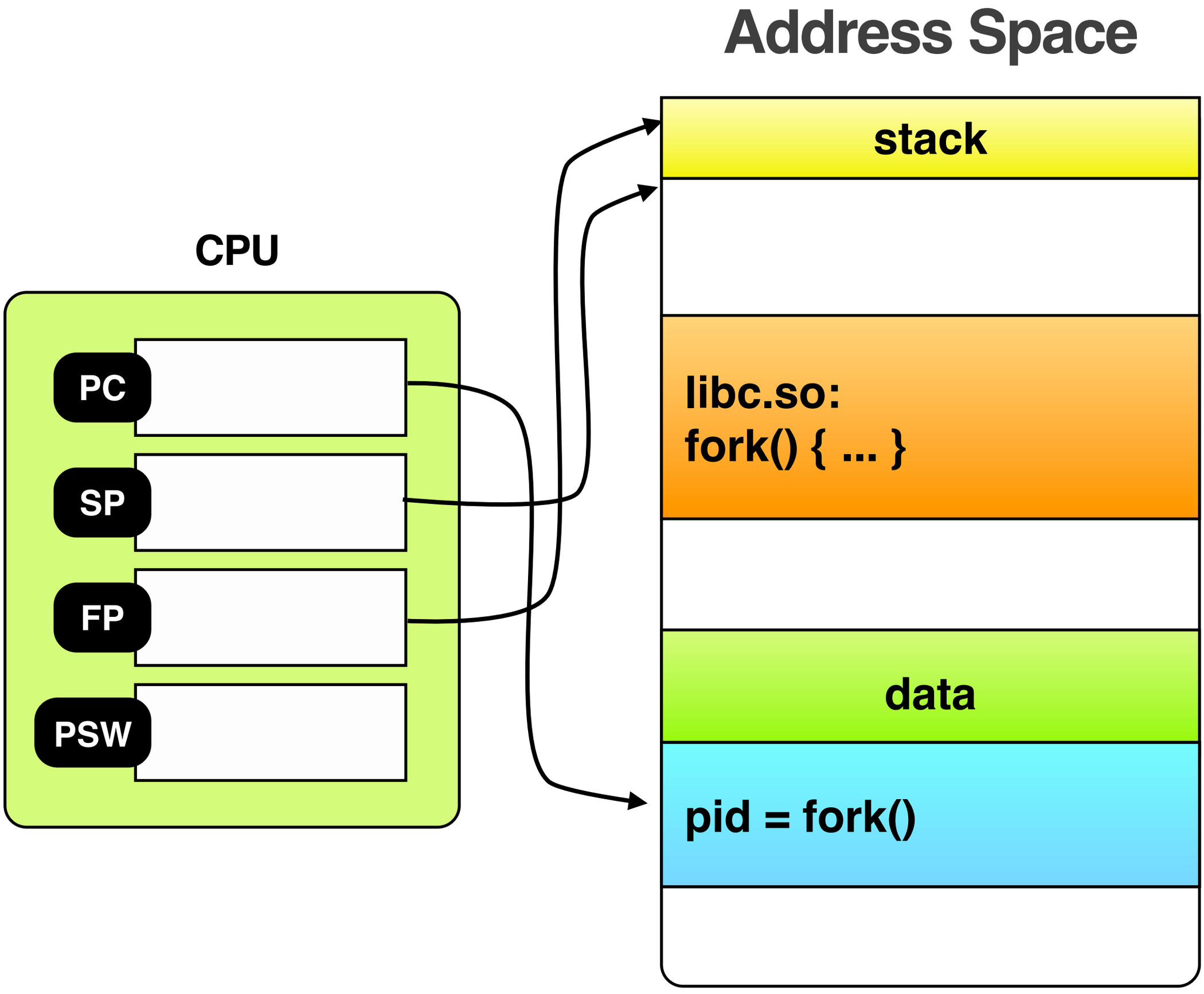
FIFO

SC

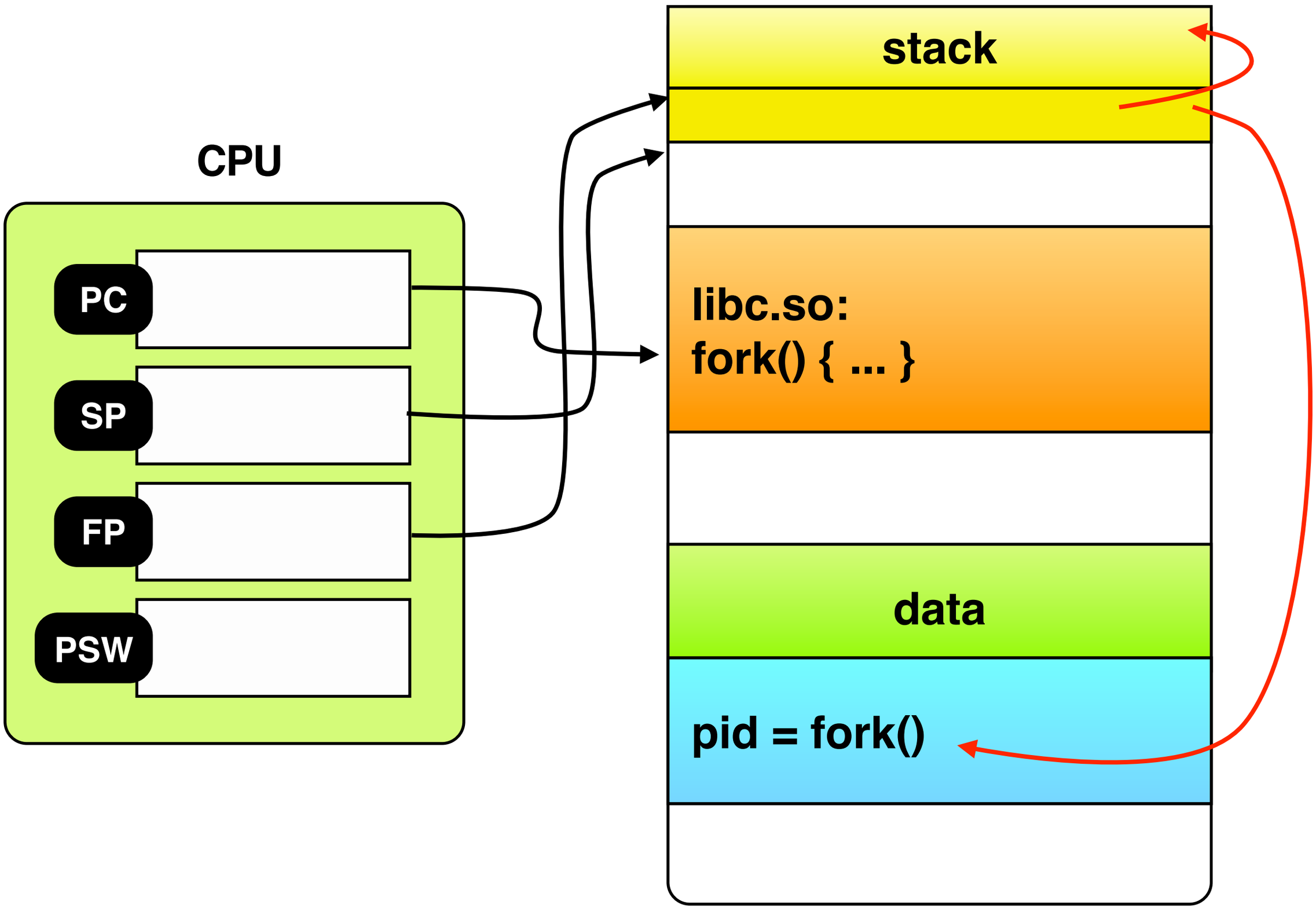
CLOCK

LRU

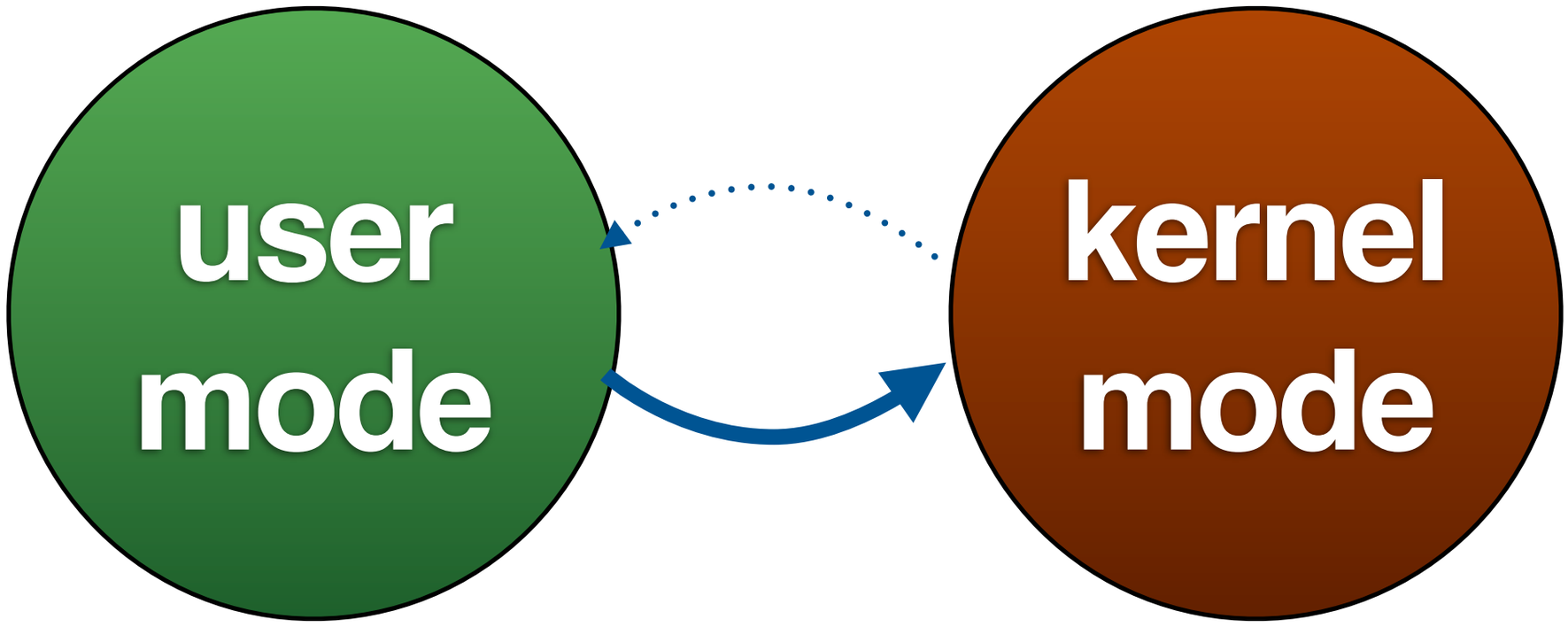
WSCLOCK

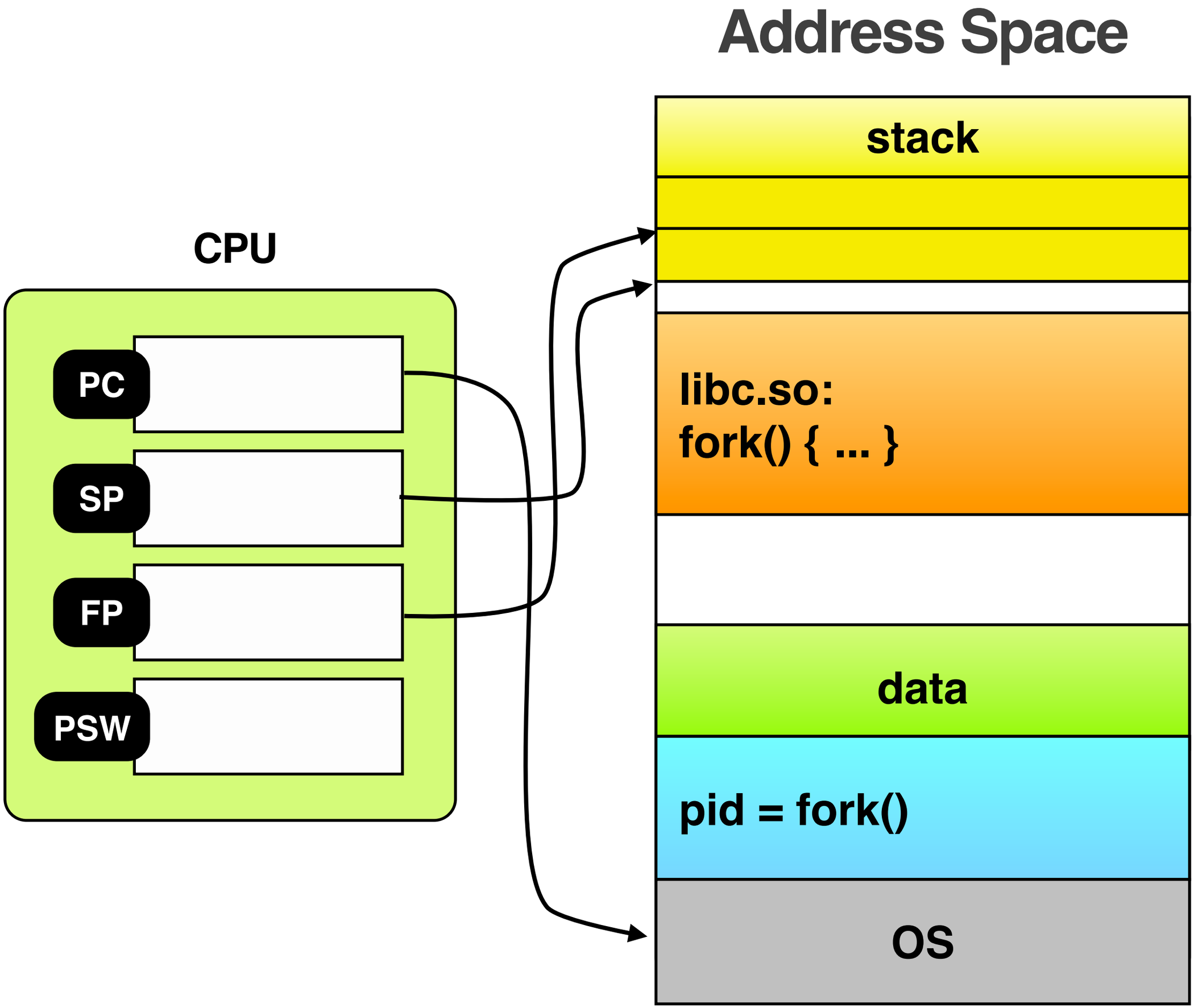

Address Space
CPU
PC
SP
FP
PSW
stack
libc.so:
fork() { ... }
data
pid = fork()

Address Space

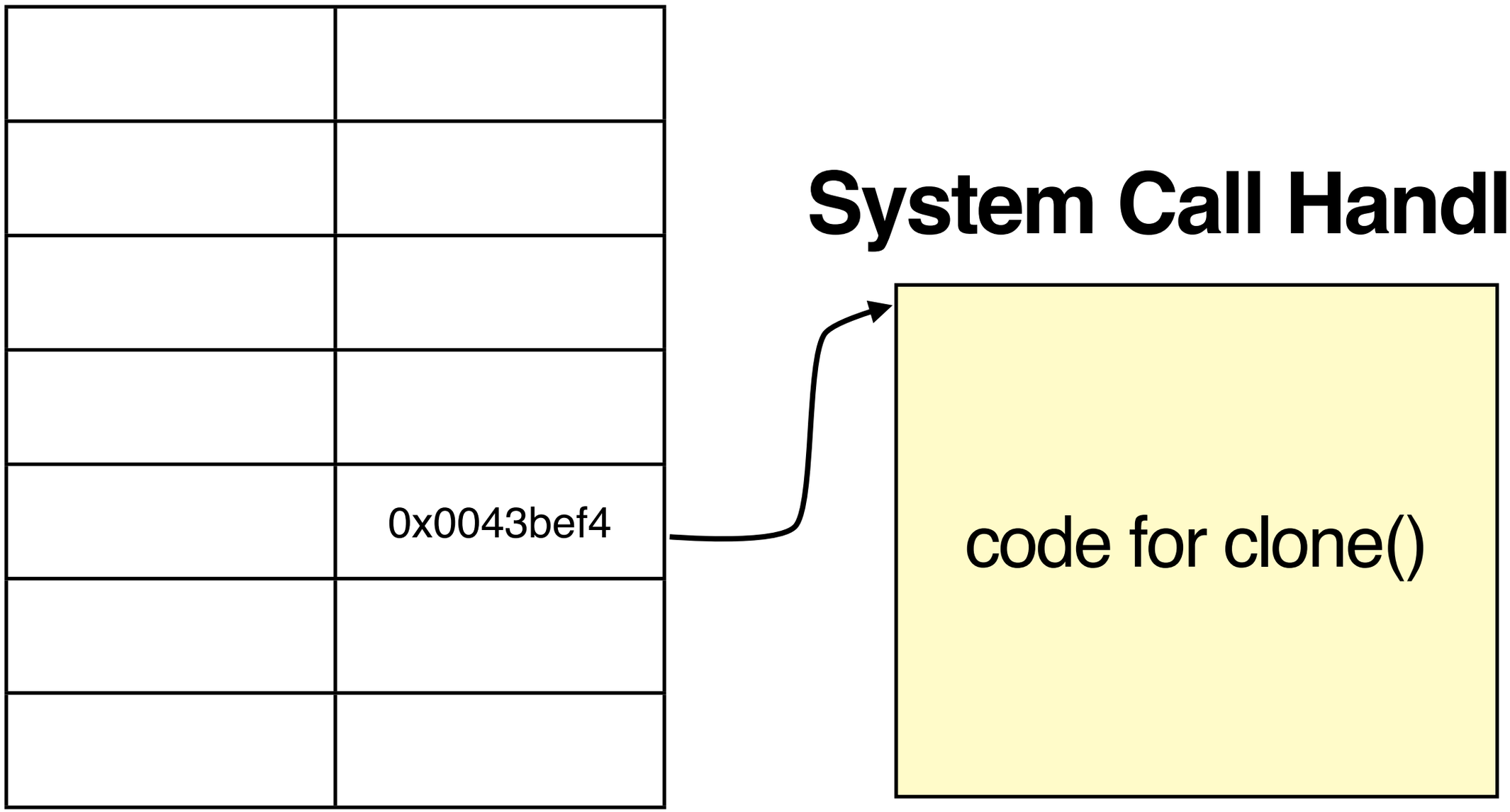


clone()

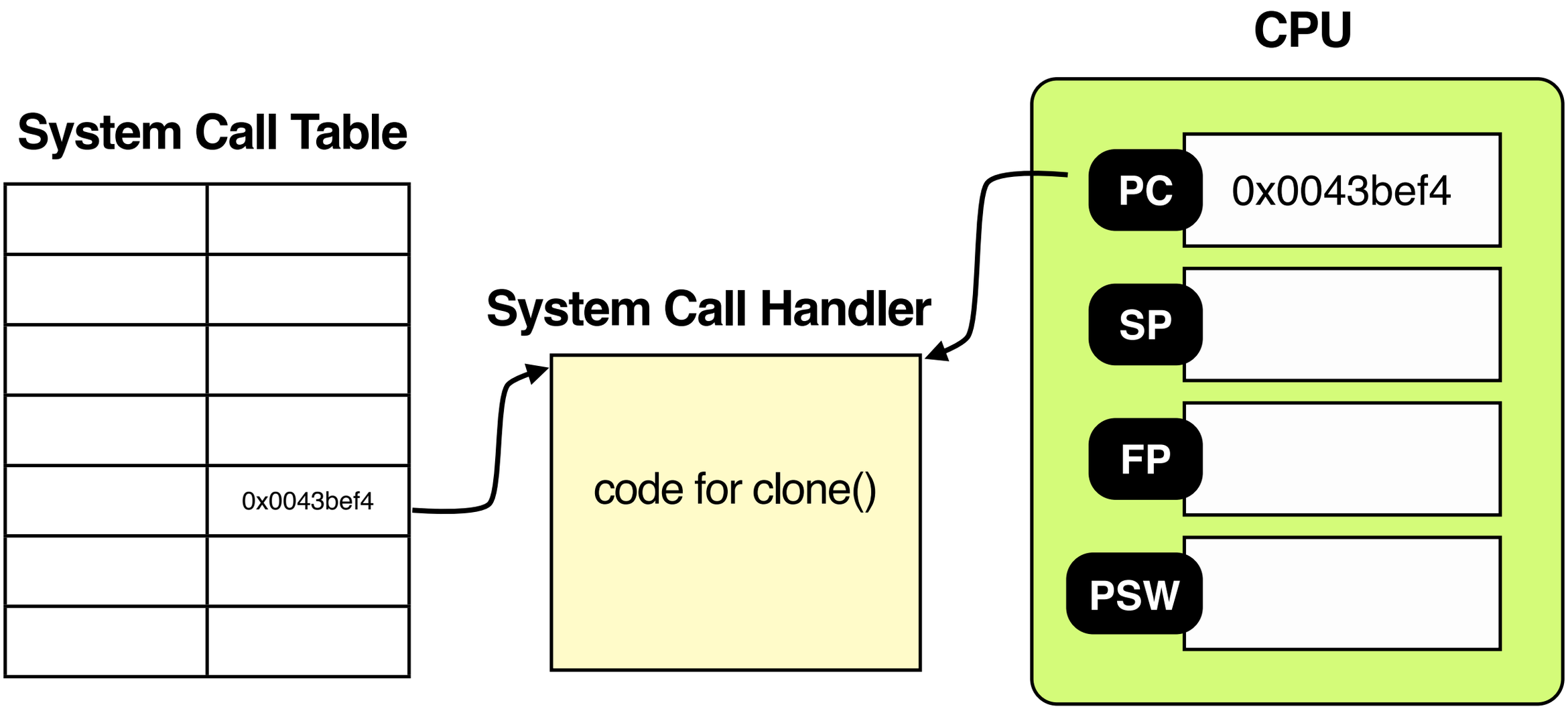

user
mode
kernel
mode


Address Space
CPU
PC
SP
FP
PSW
stack
libc.so:
fork() { ... }
data
pid = fork()
OS

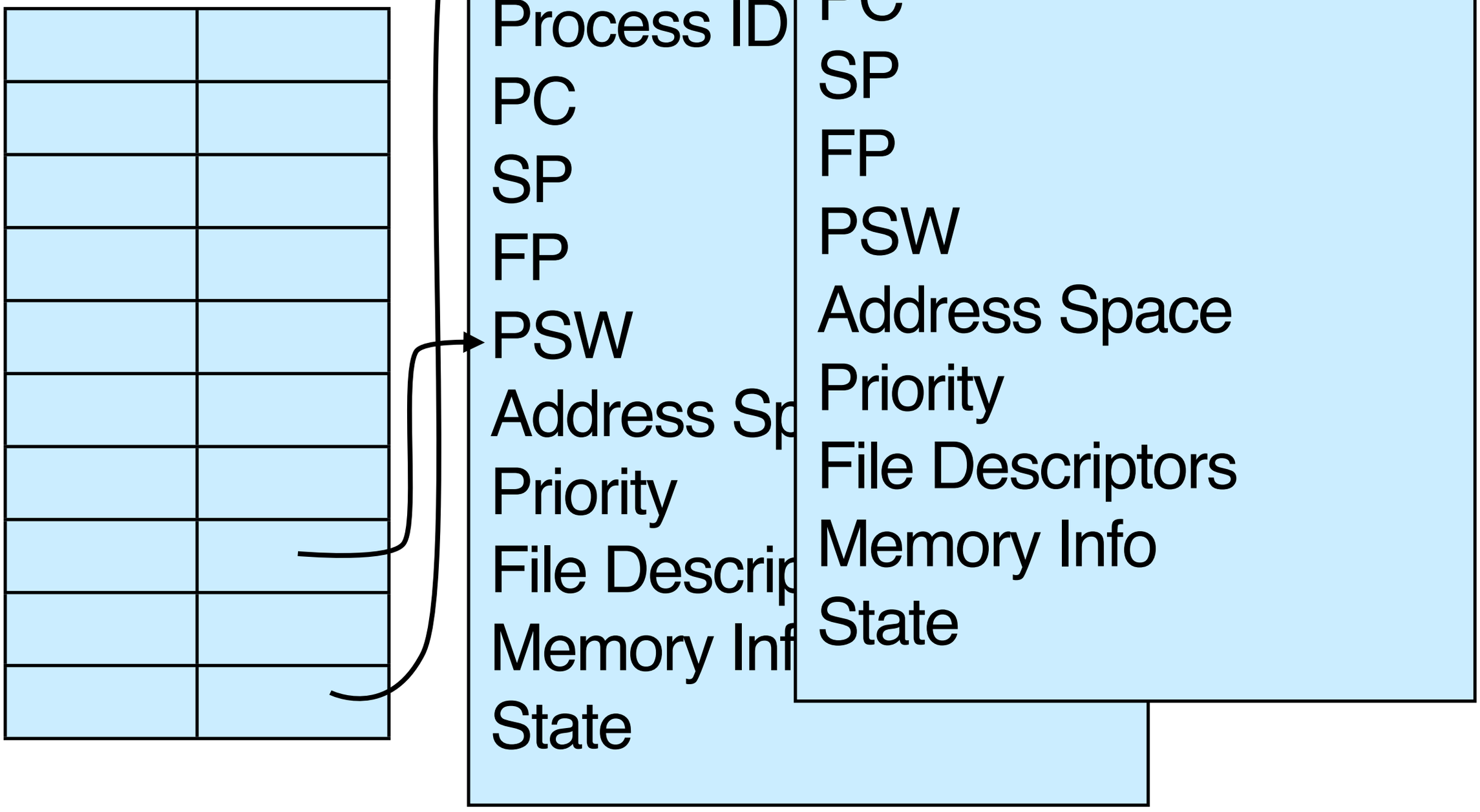
System Call Table



(locked in physical memory)


System Call Table
0x0043bef4
System Call Handler
code for clone()
CPU
PC
0x0043bef4
SP
FP
PSW

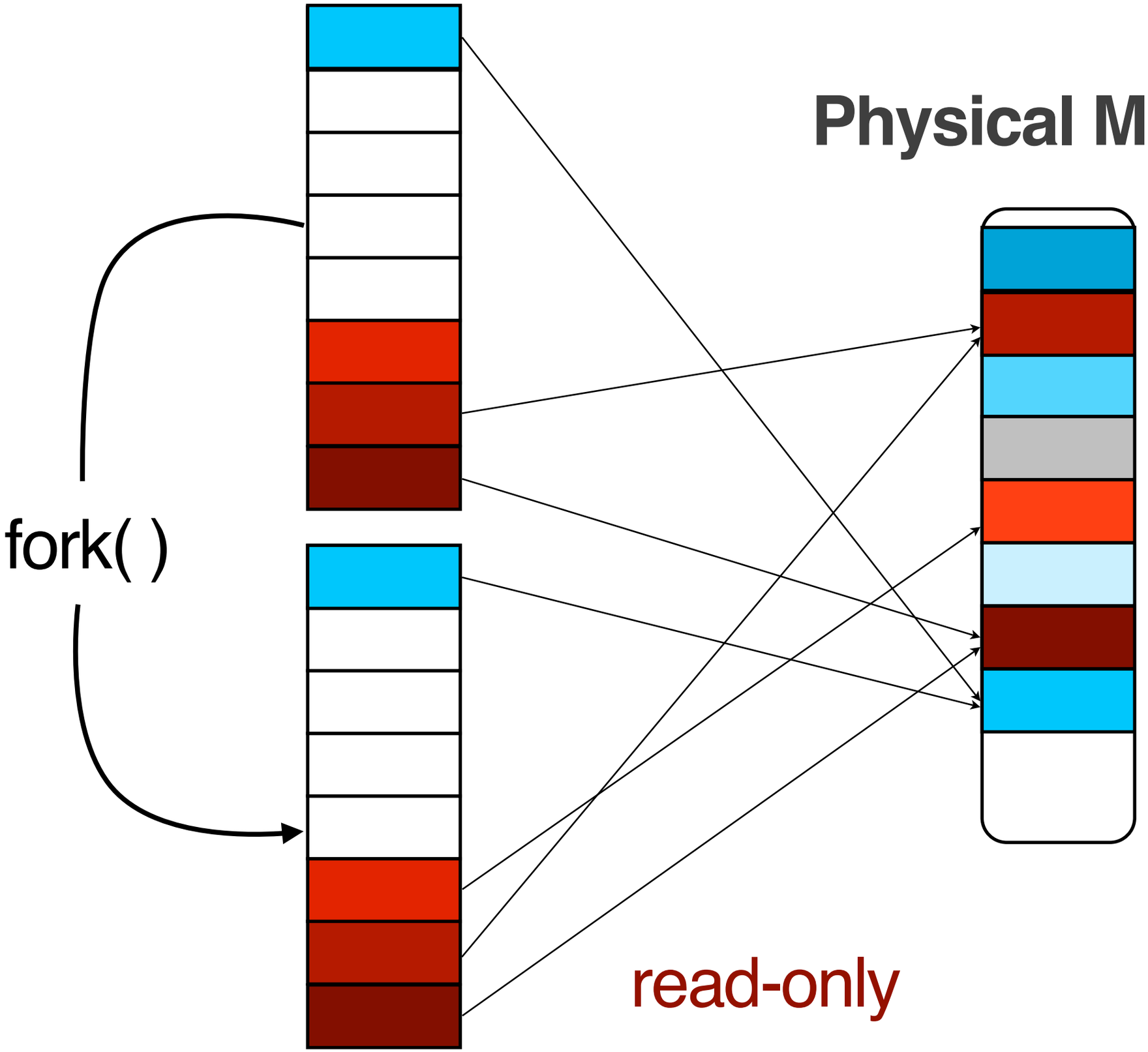
Process Table

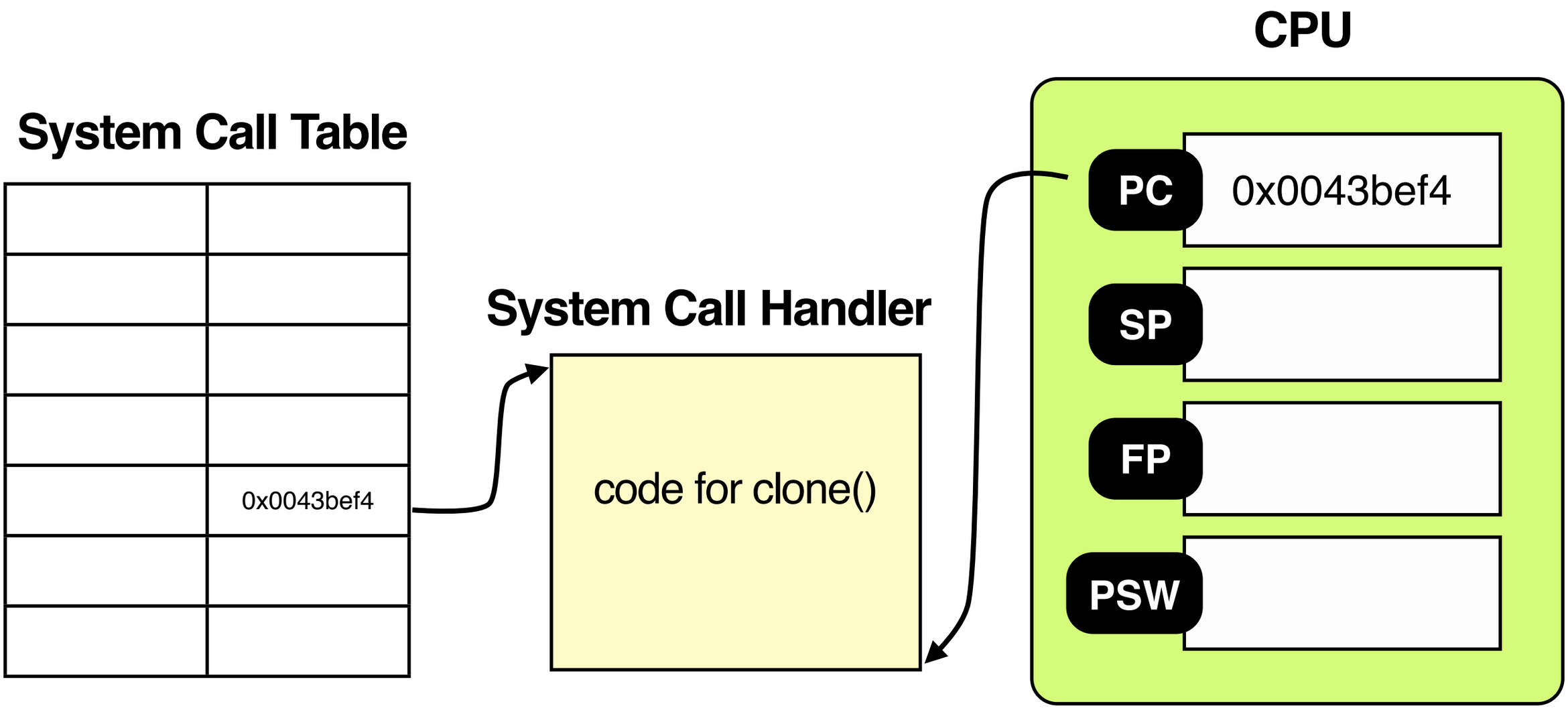


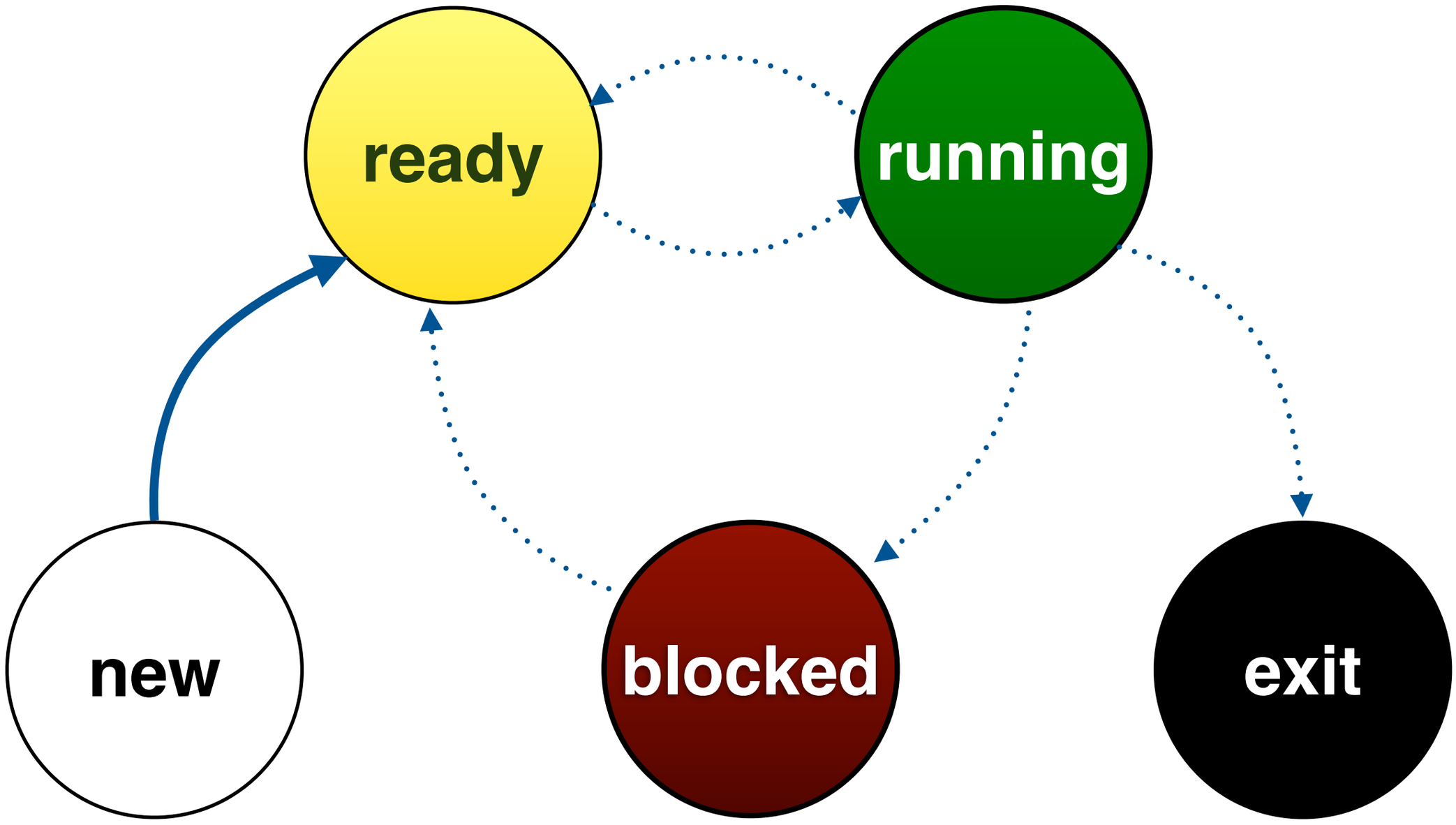
use critical regions

Address Spaces

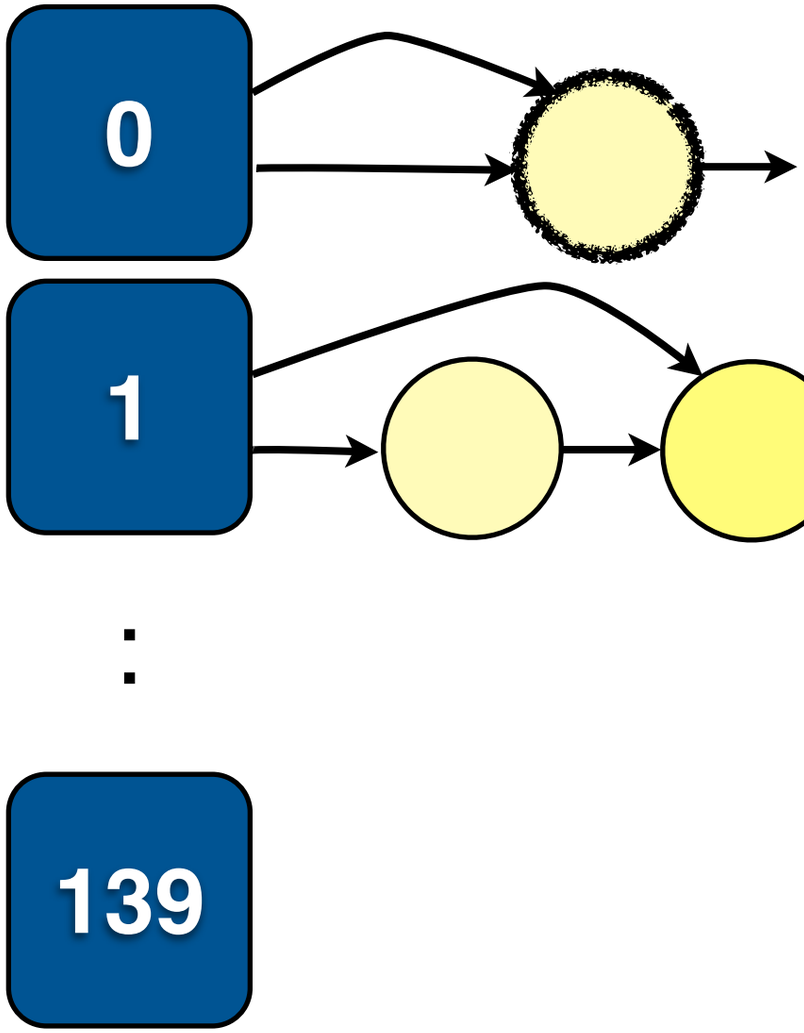
Physical Memory



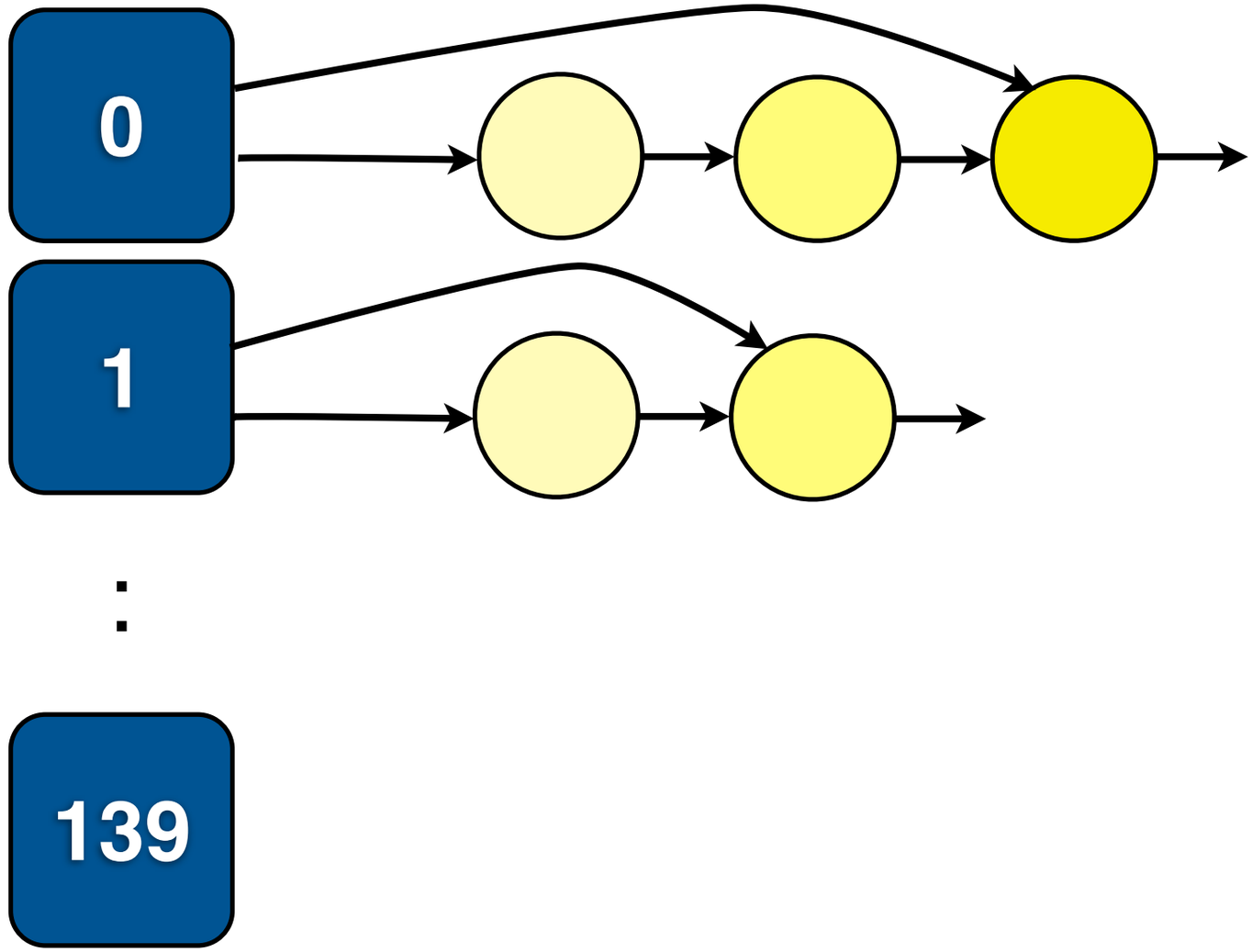

System Call Table
0x0043bef4
System Call Handler
code for clone()
CPU
PC
0x0043bef4
SP
FP
PSW

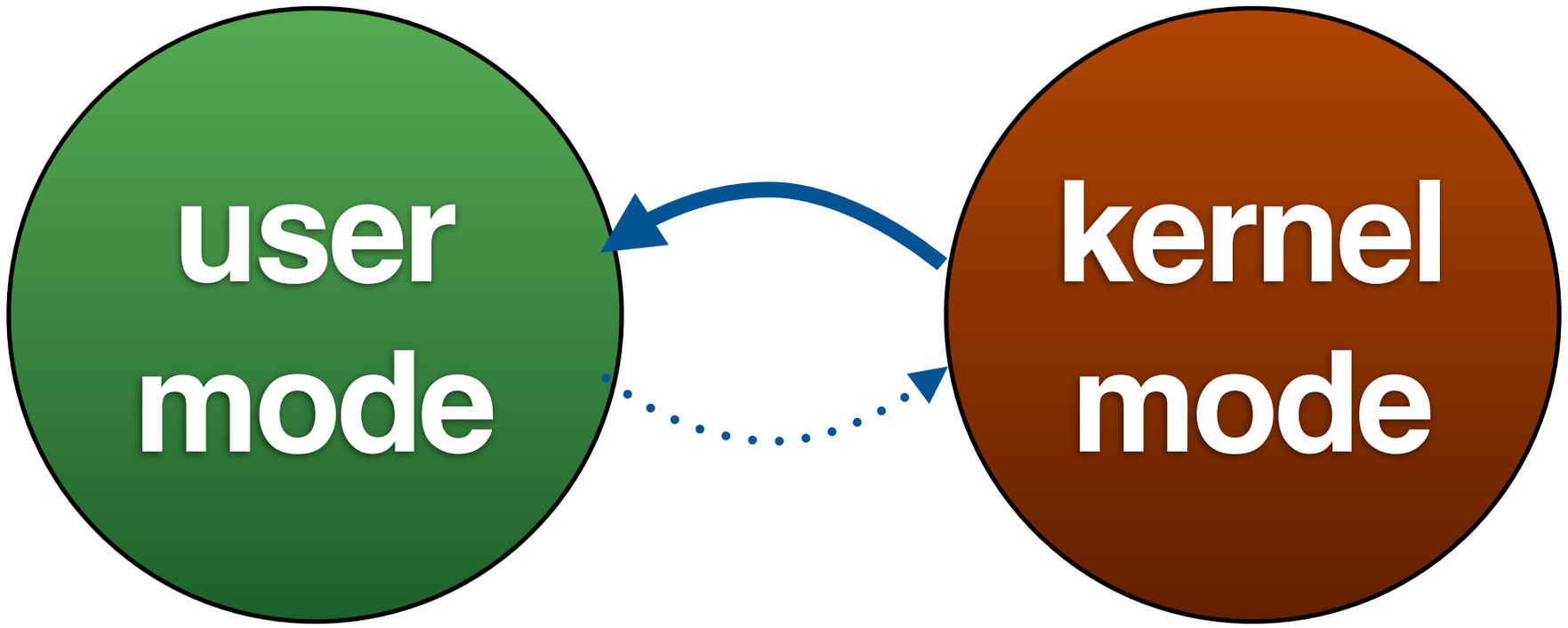

ready
running
new
blocked
exit

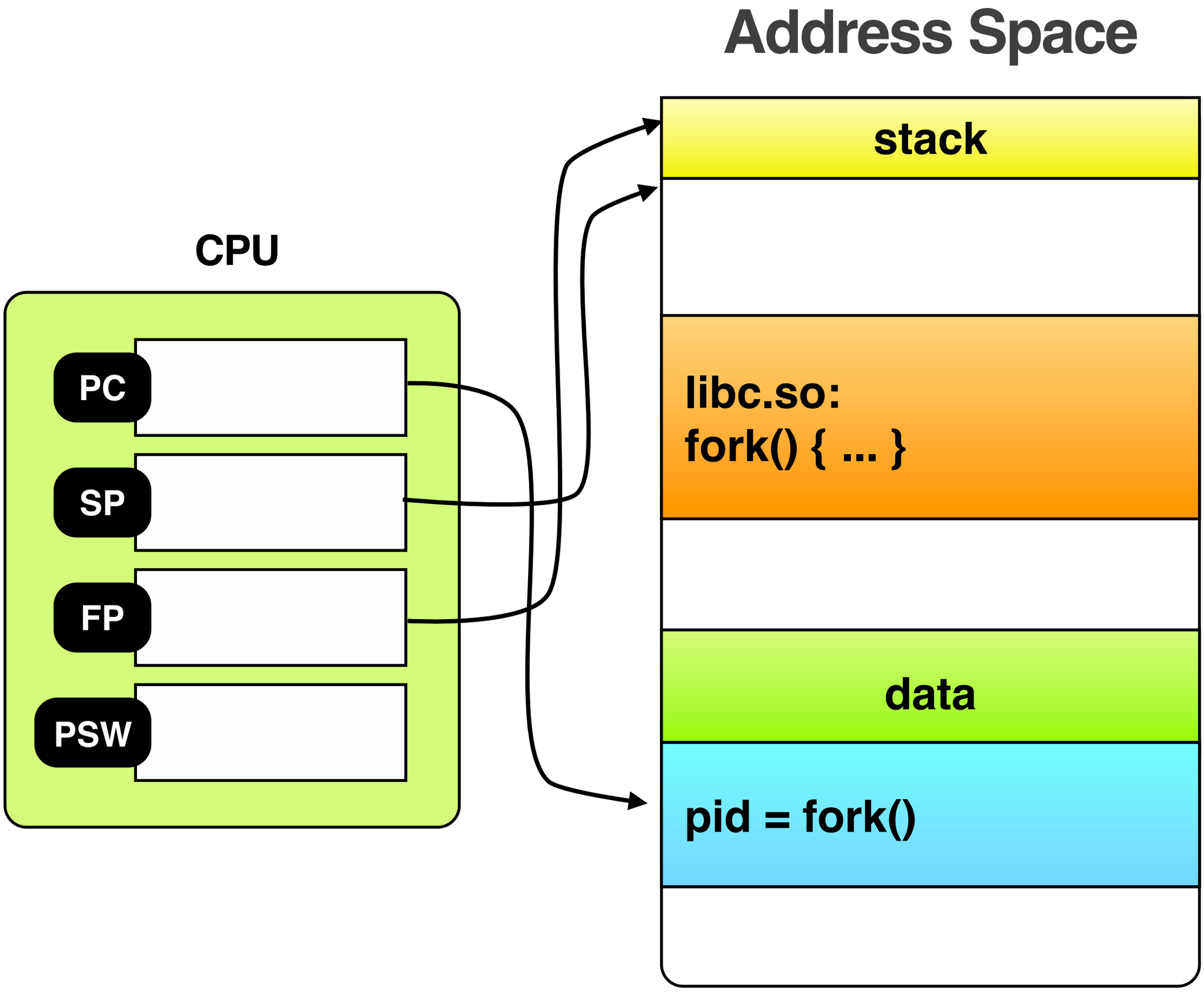
Active



Expired




user
mode
kernel
mode

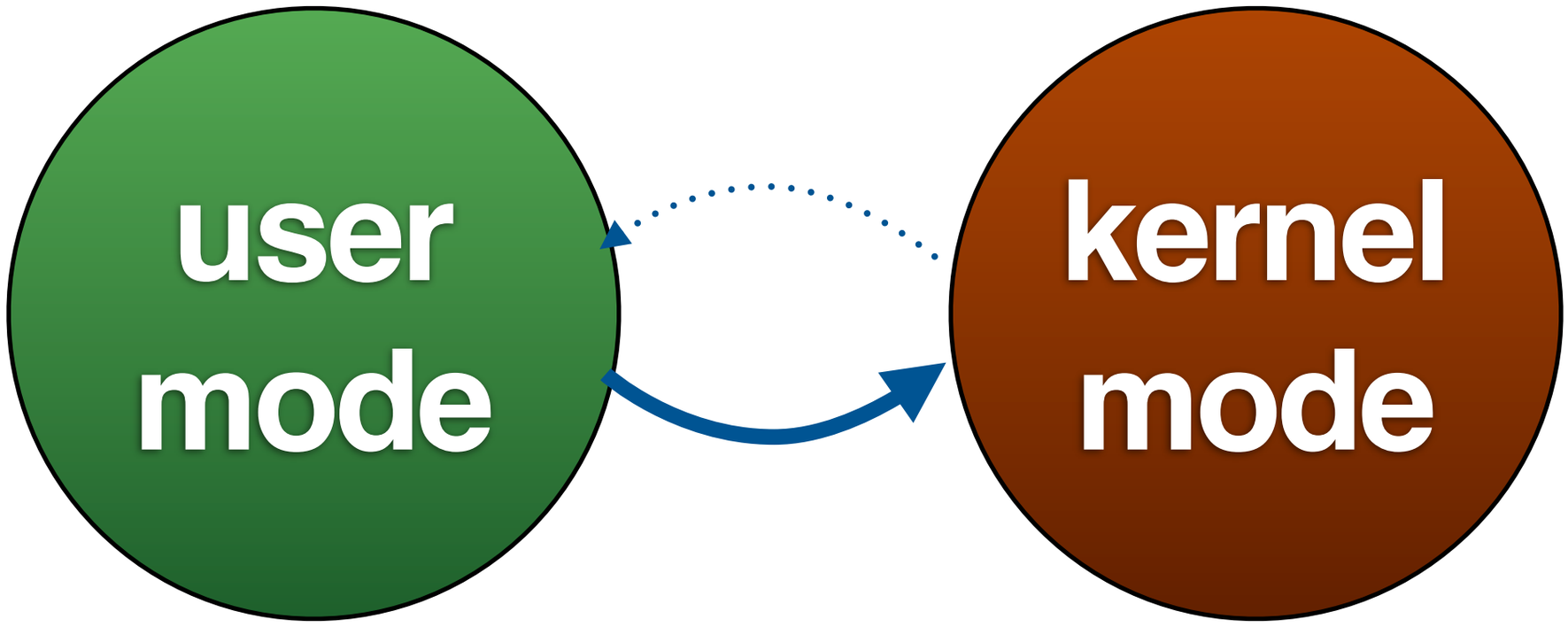

Address Space
CPU
PC
SP
FP
PSW
stack
libc.so:
fork() { ... }
data
pid = fork()

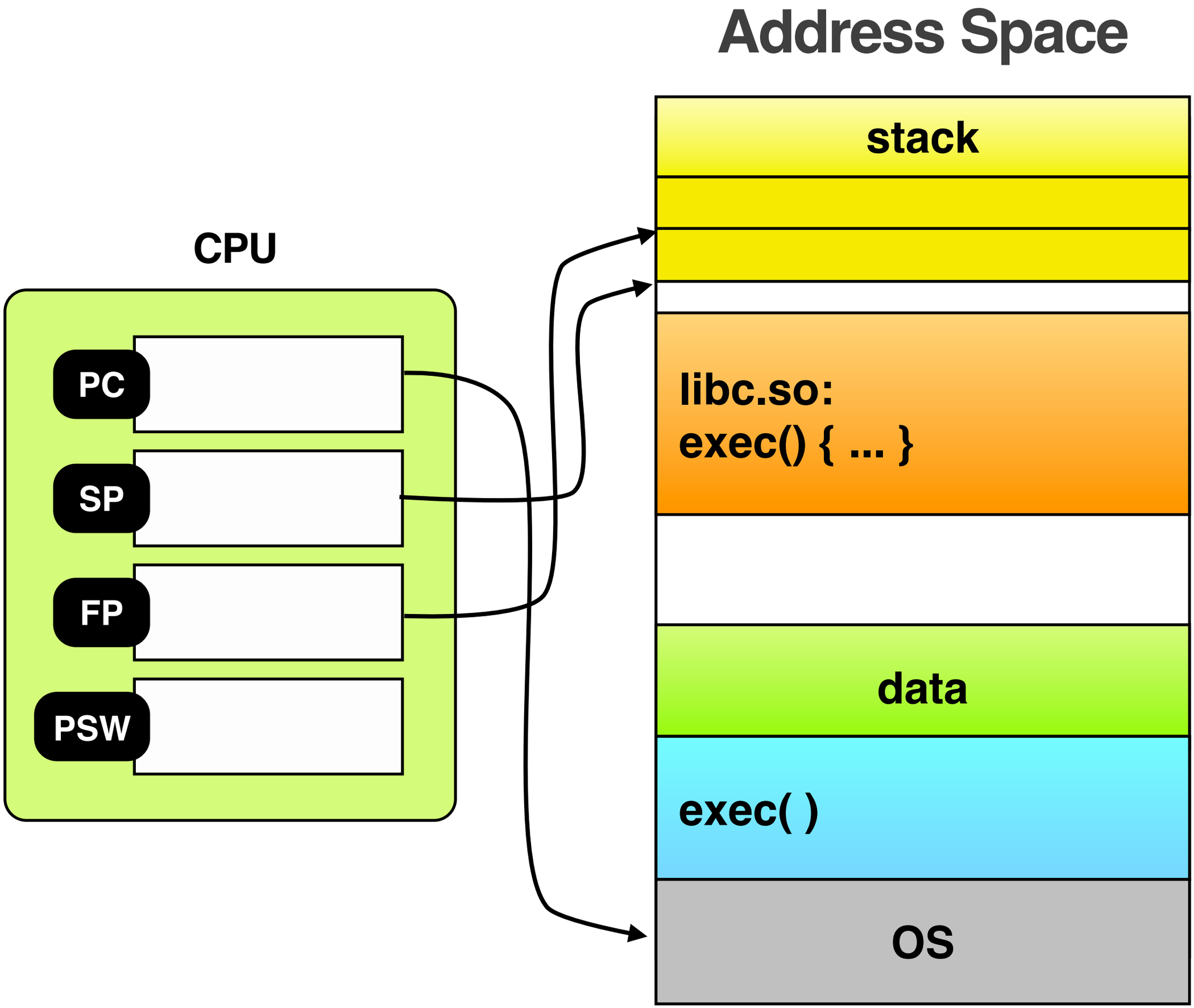
(bash)

wait()

(child)

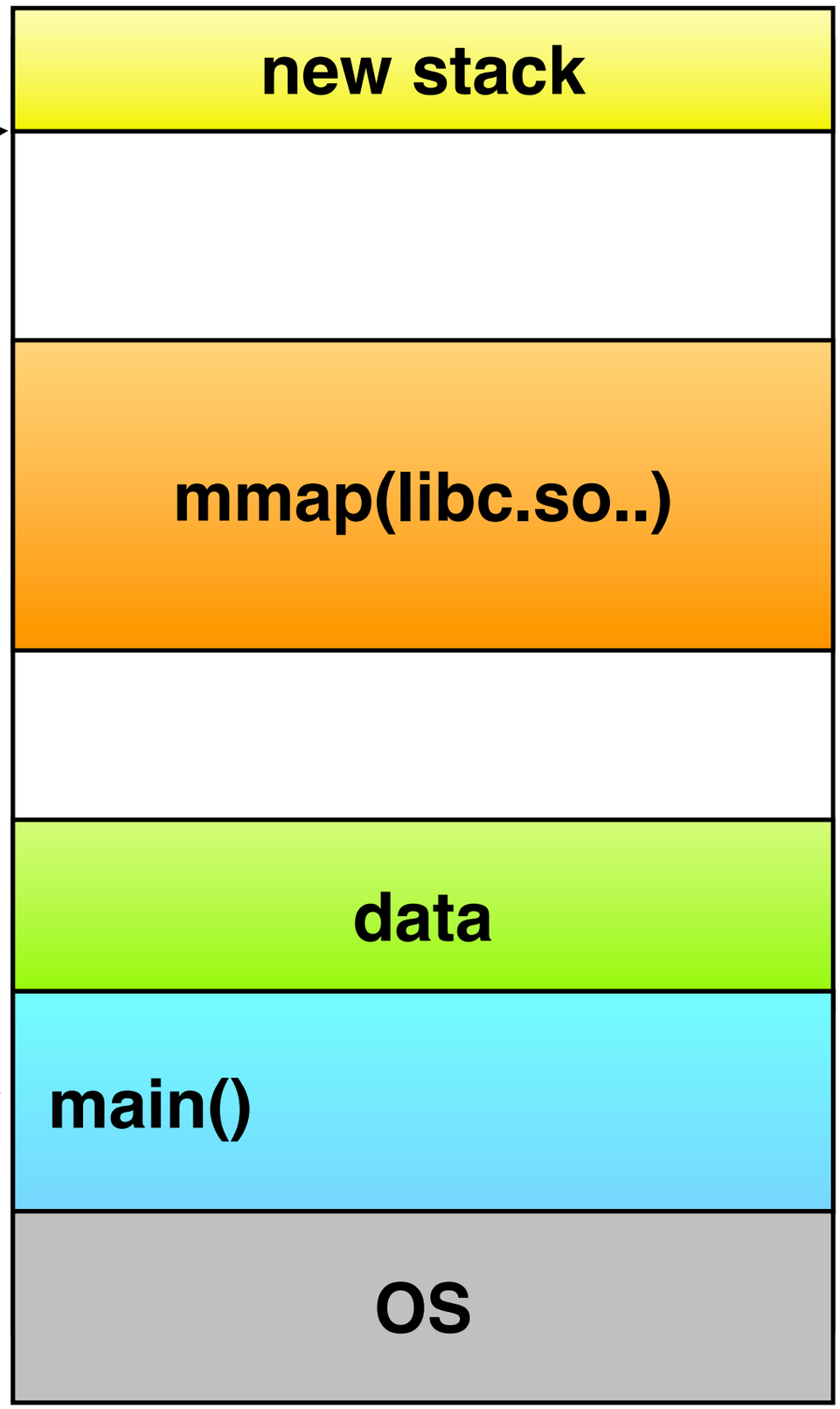
```
execvp(“ls”,”ls”)
```


user
mode
kernel
mode

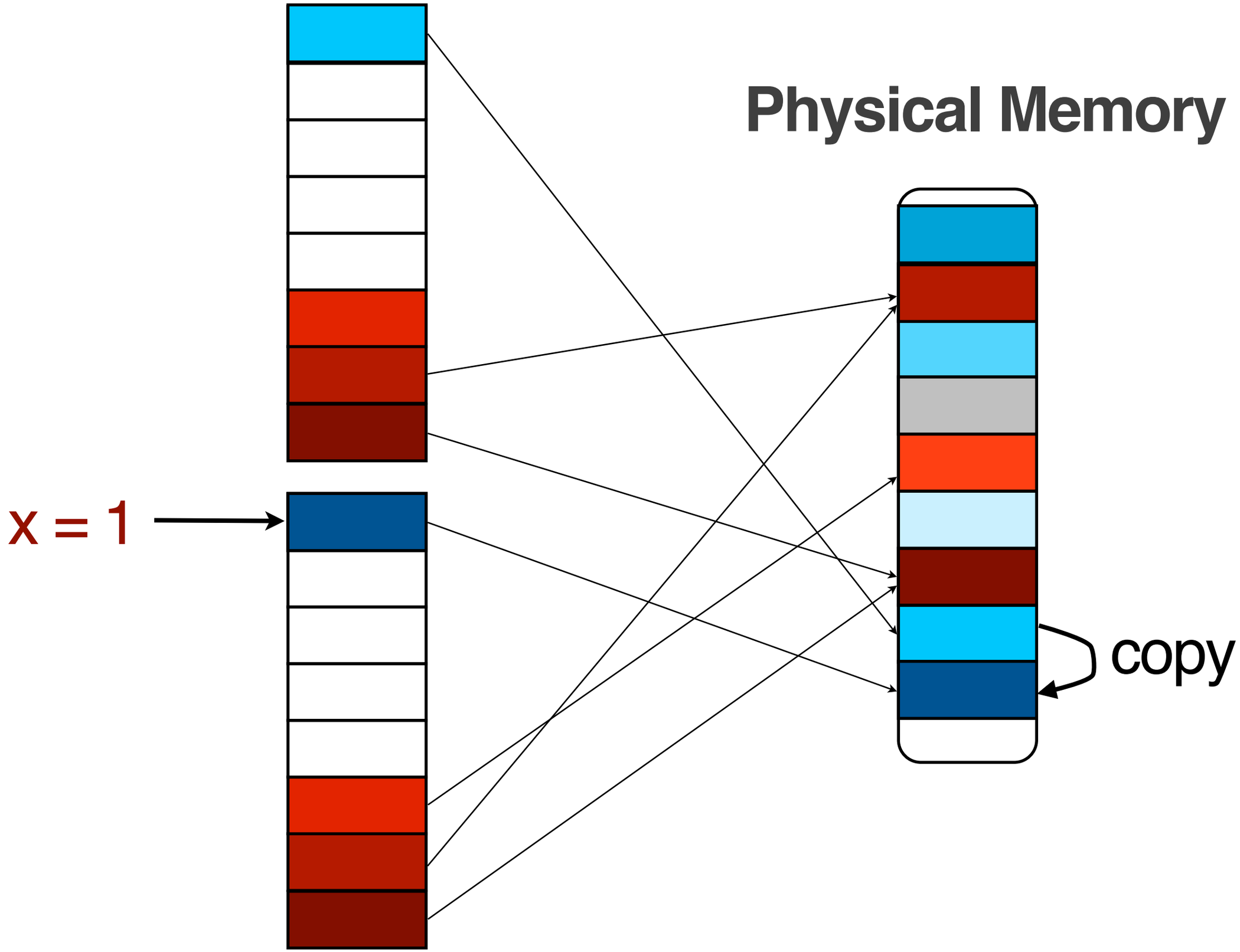

Address Space
CPU
PC
SP
FP
PSW
stack
libc.so:
exec() { ... }
data
exec( )
OS

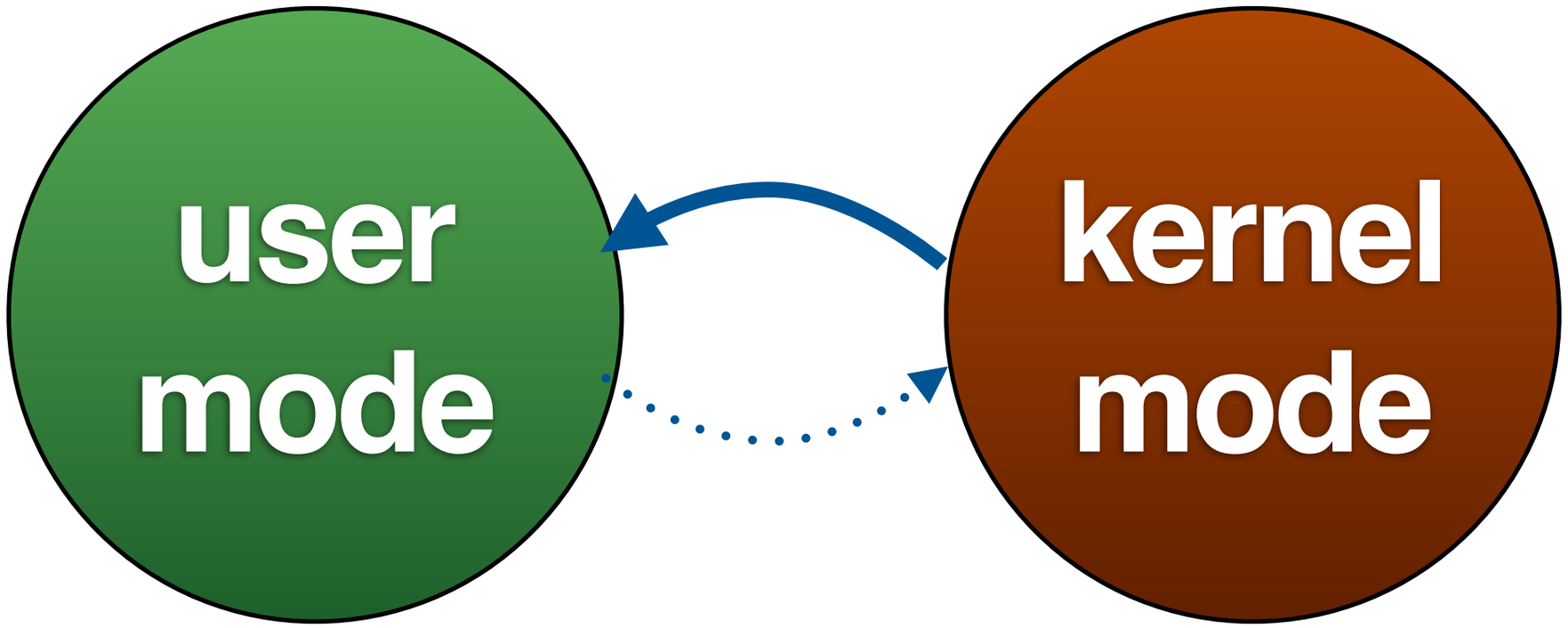
read dir entries (data blocks) of **/**
look for i-node number for **usr**
read i-node for **/usr**
read data blocks of **/usr**
look for i-node number for **bin**
read i-node for **/usr/bin**
read data blocks of **/usr/bin**
look for i-node number for **ls**
read data blocks for **/usr/bin/ls**

Process ID
PC
SP
FP
PSW
Address Space
Priority
File Descriptors
Memory Info
State

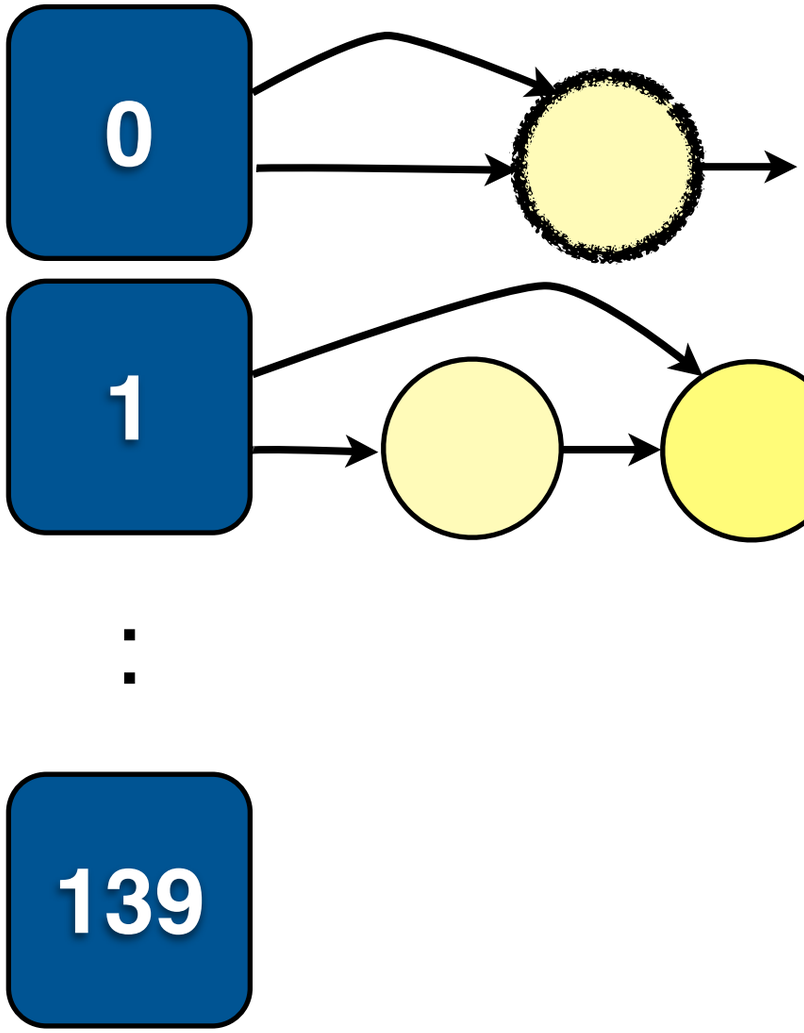


Address Spaces

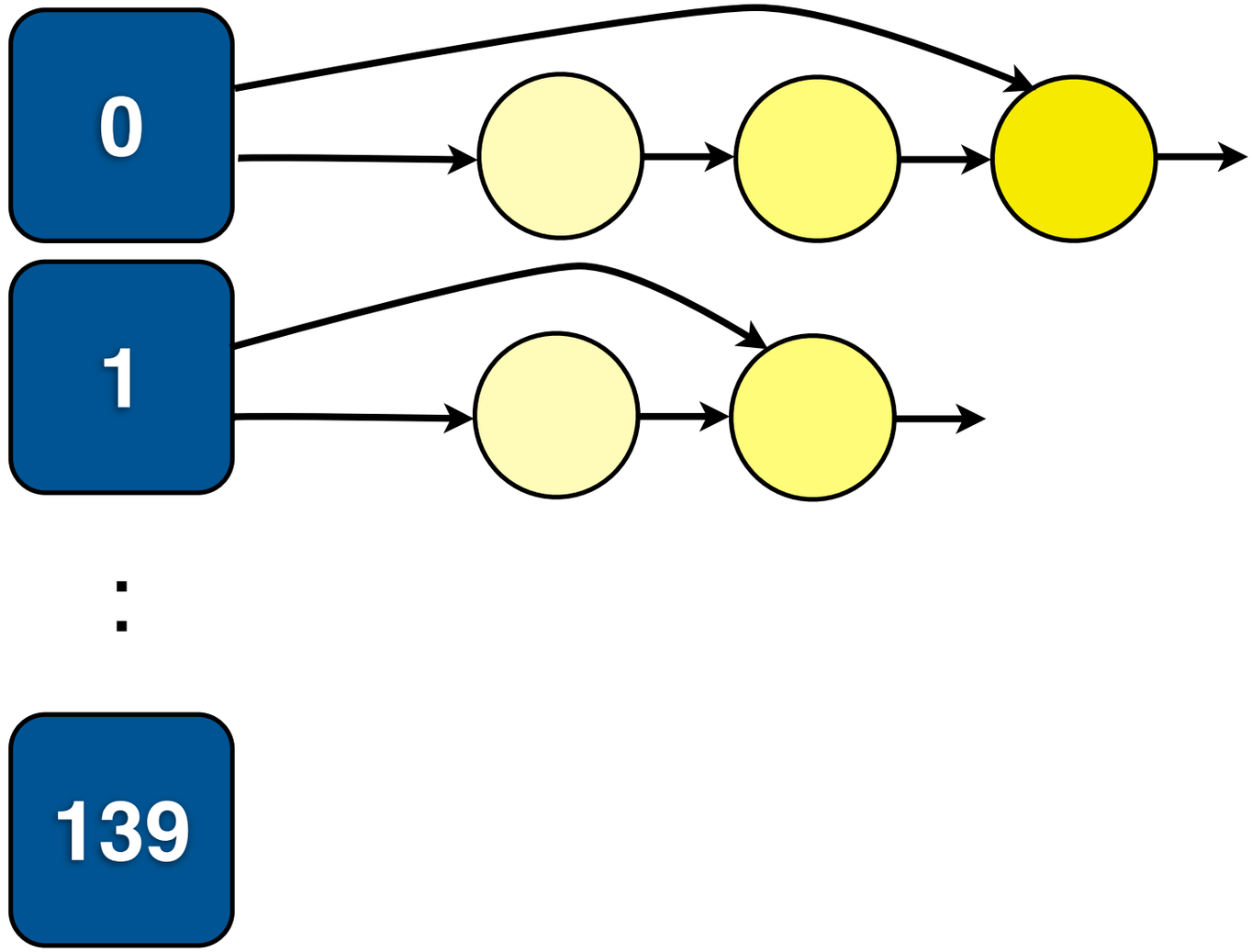


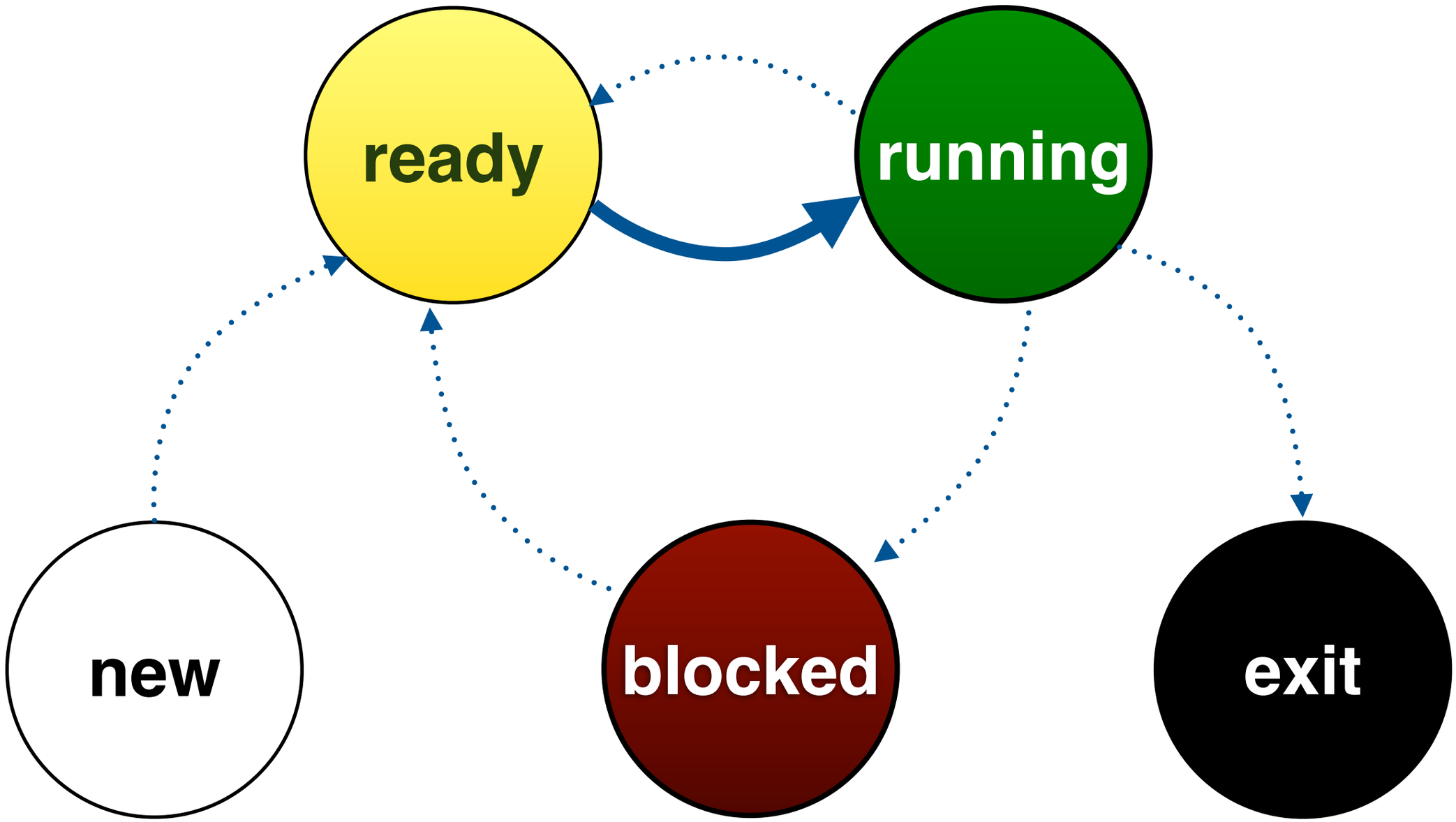

user
mode
kernel
mode

Active



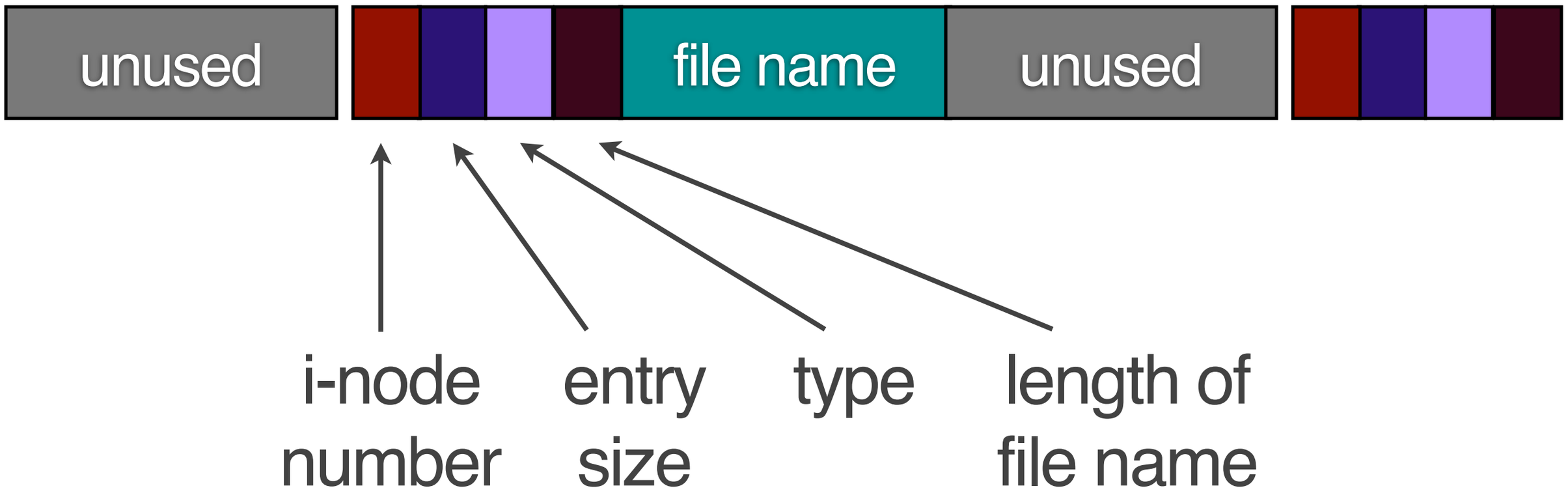
Expired




ready
running
new
blocked
exit

Process ID
PC
SP
FP
PSW
Address Space
Priority
File Descriptors
Memory Info
State
File Systems Info

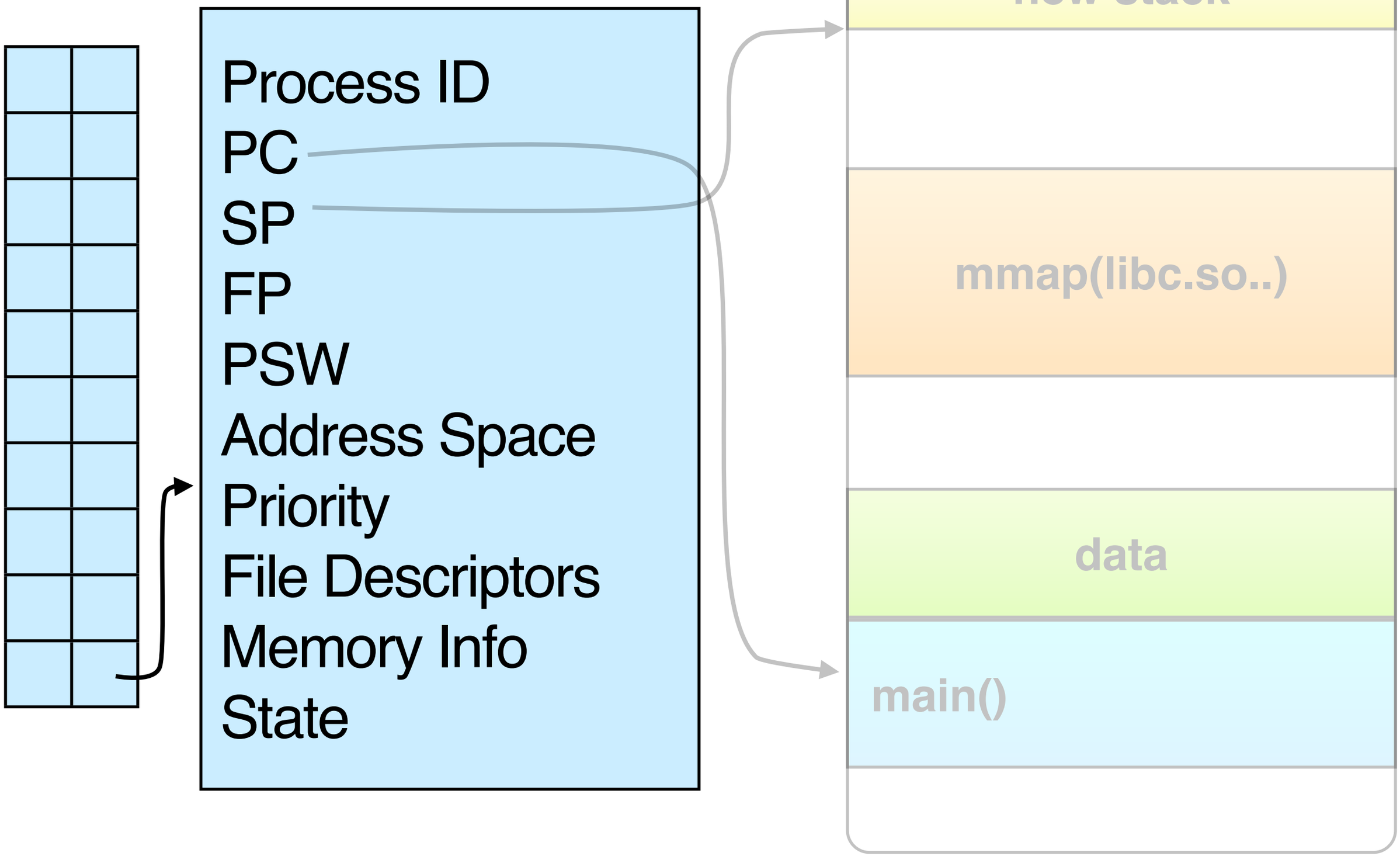
read directory entries of current working directory



```
bash$ ls
foo bar a.out lab.c
```

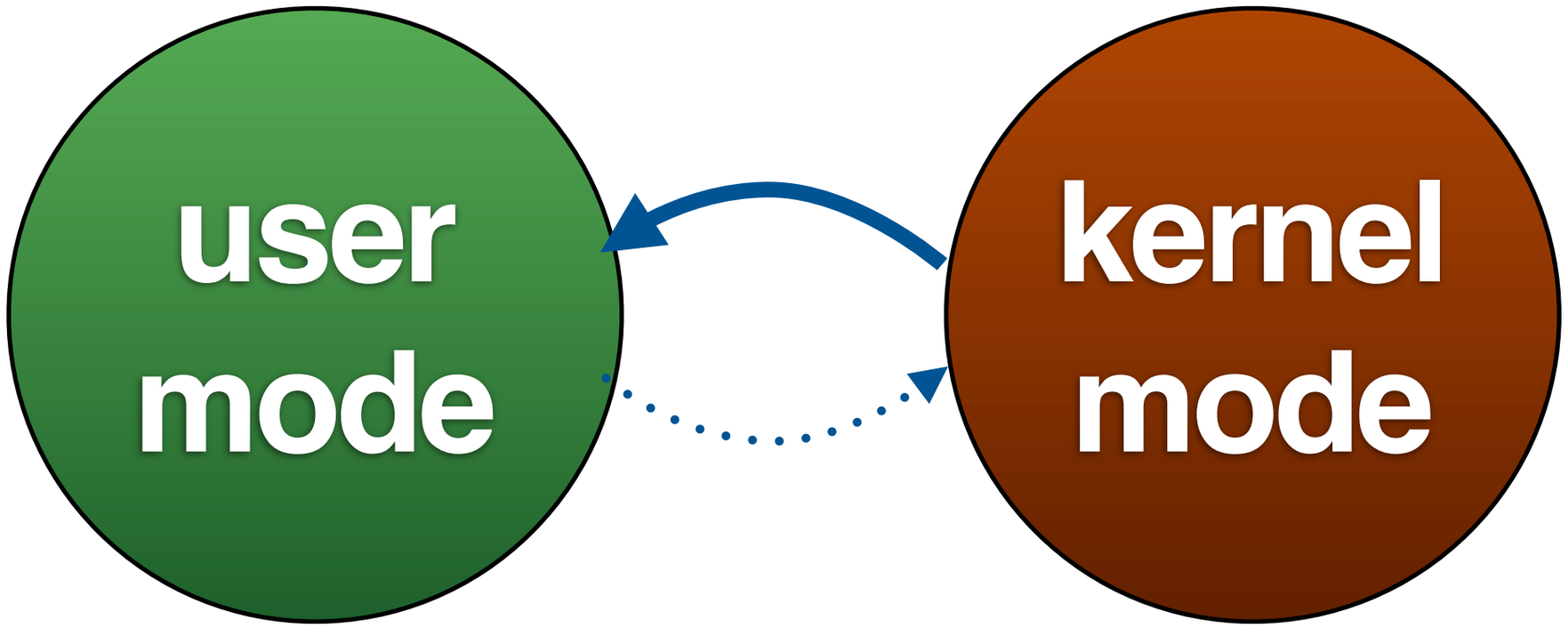
exit()

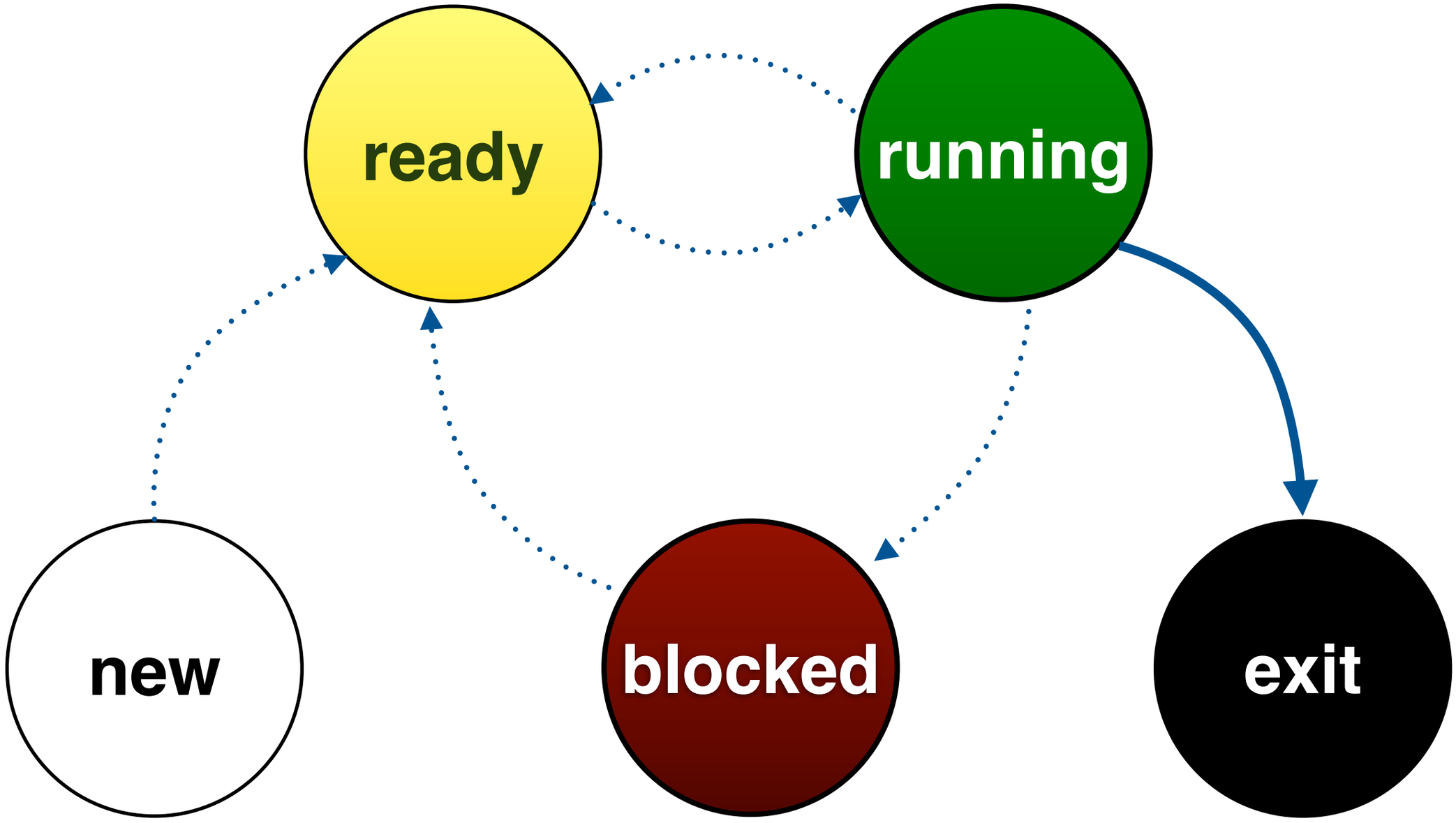
Process Table



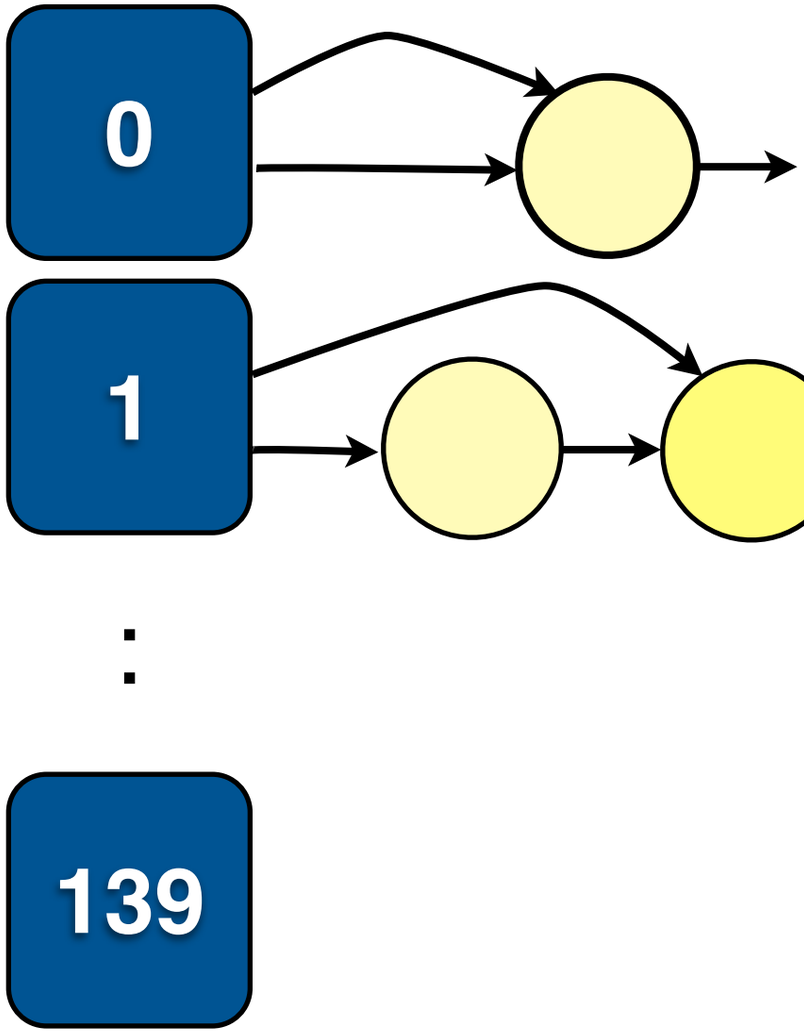
SIGCHLD

Process ID
PC
SP
FP
PSW
Address Space
Priority
File Descriptors
Memory Info
State
File Systems Info
Signal Handlers
Signal Queue

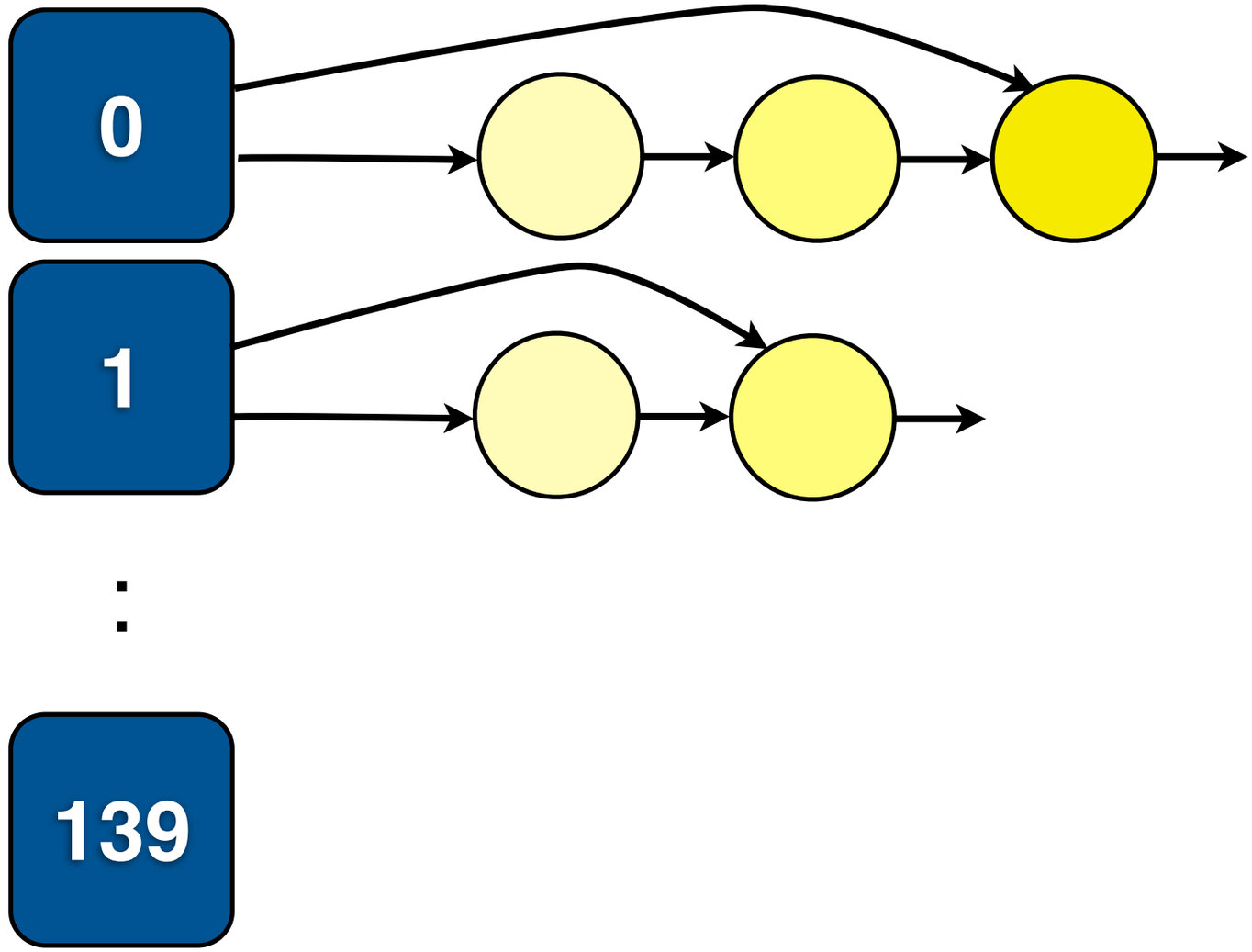

user
mode
kernel
mode


ready
running
new
blocked
exit

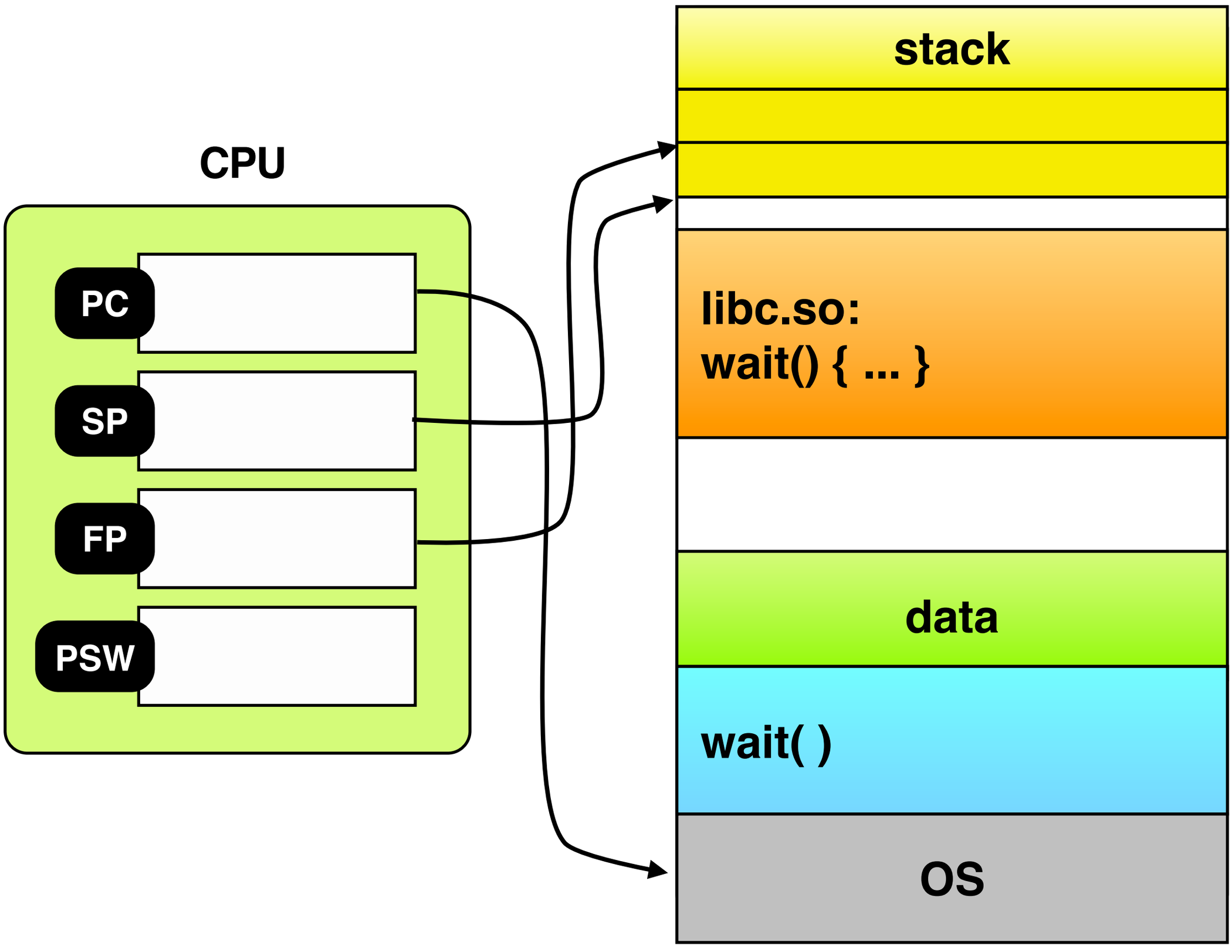
Active

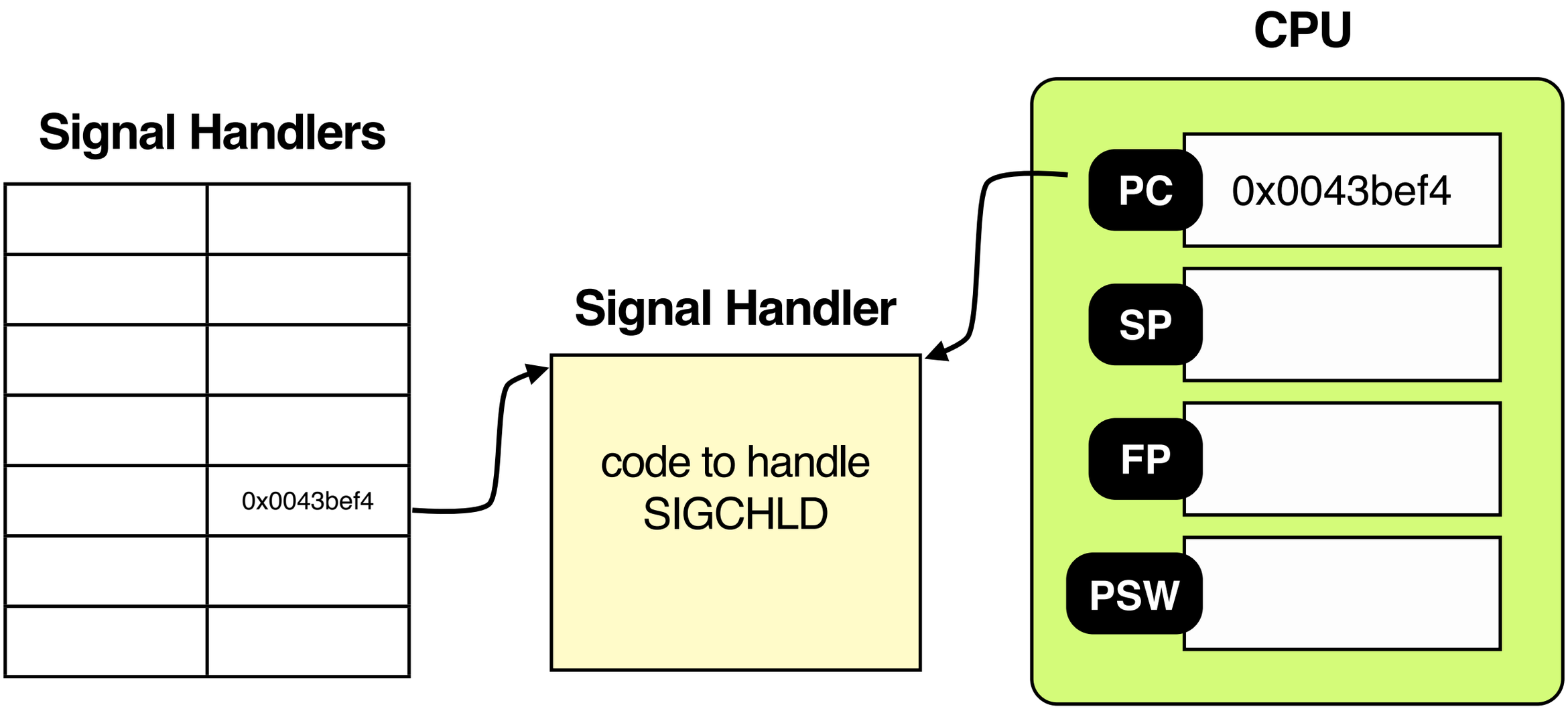


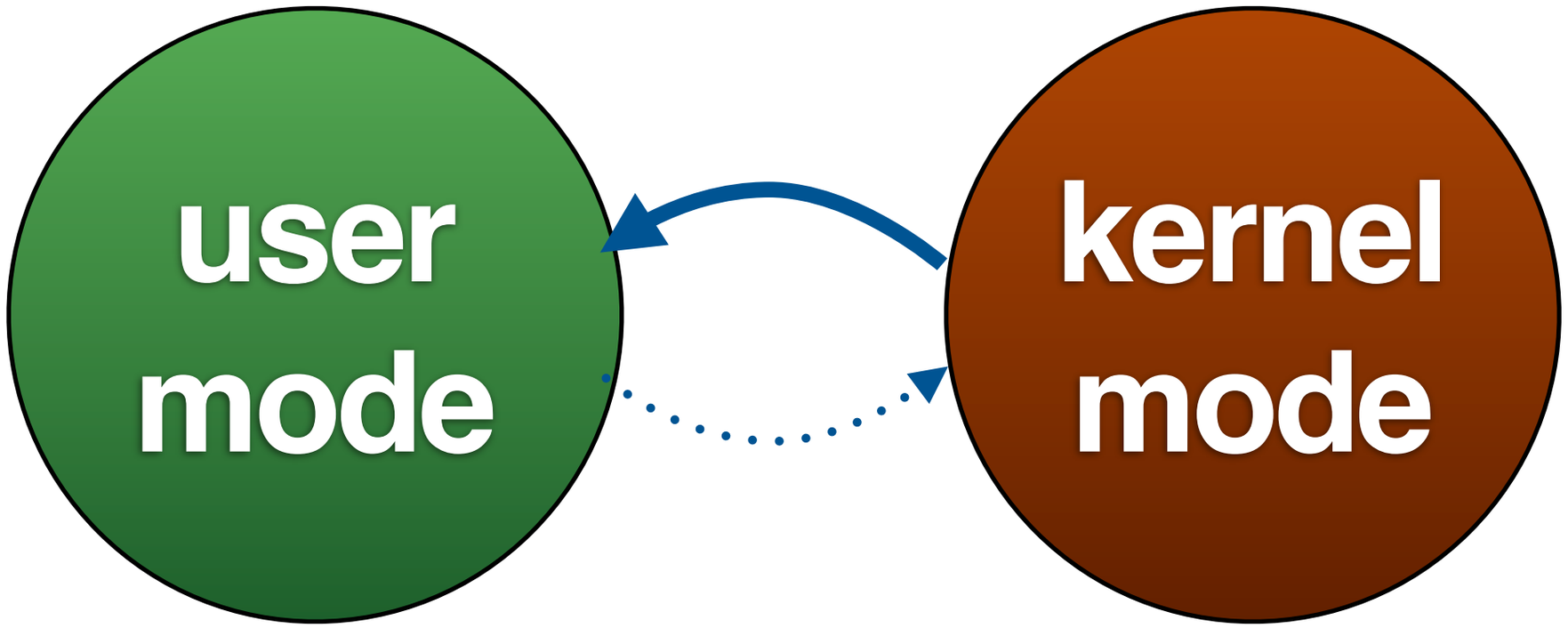
Expired

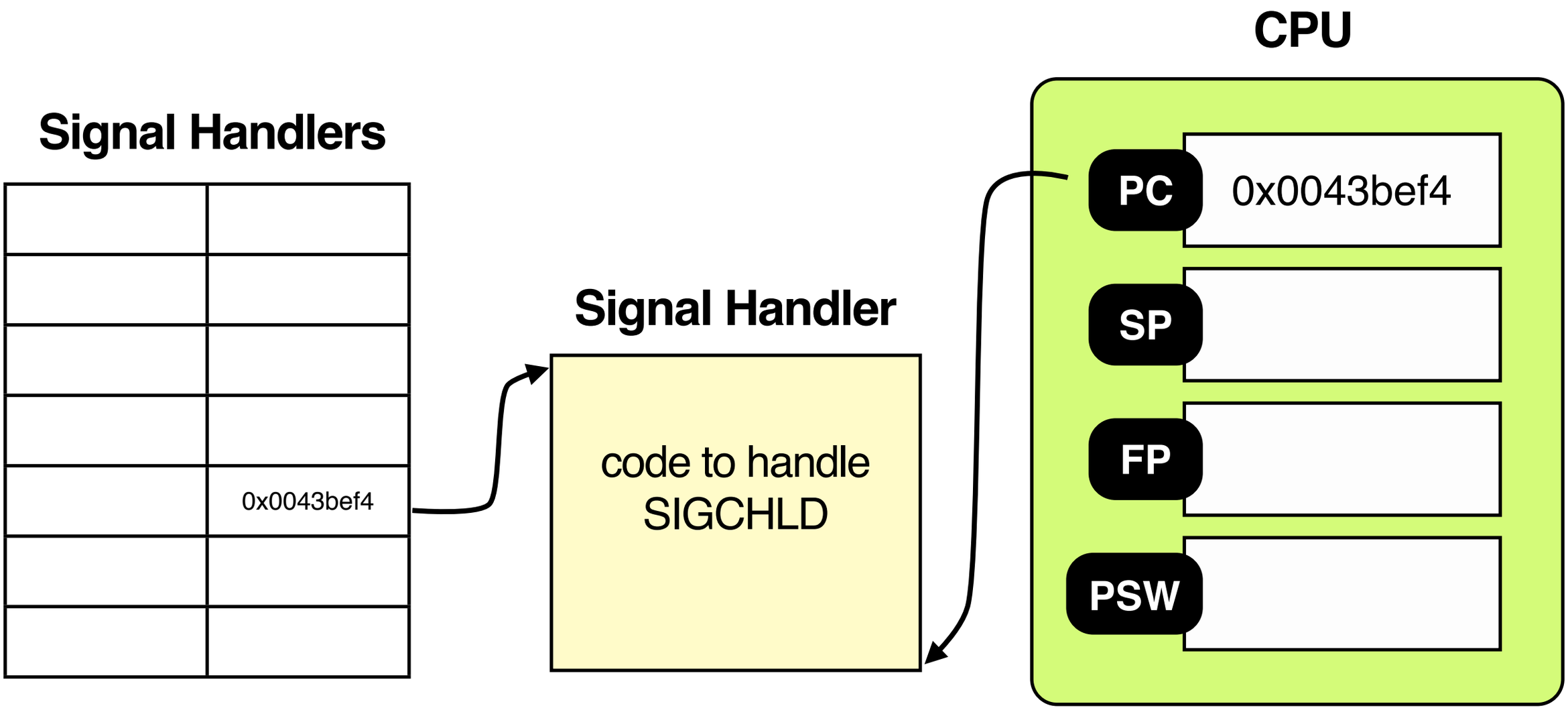


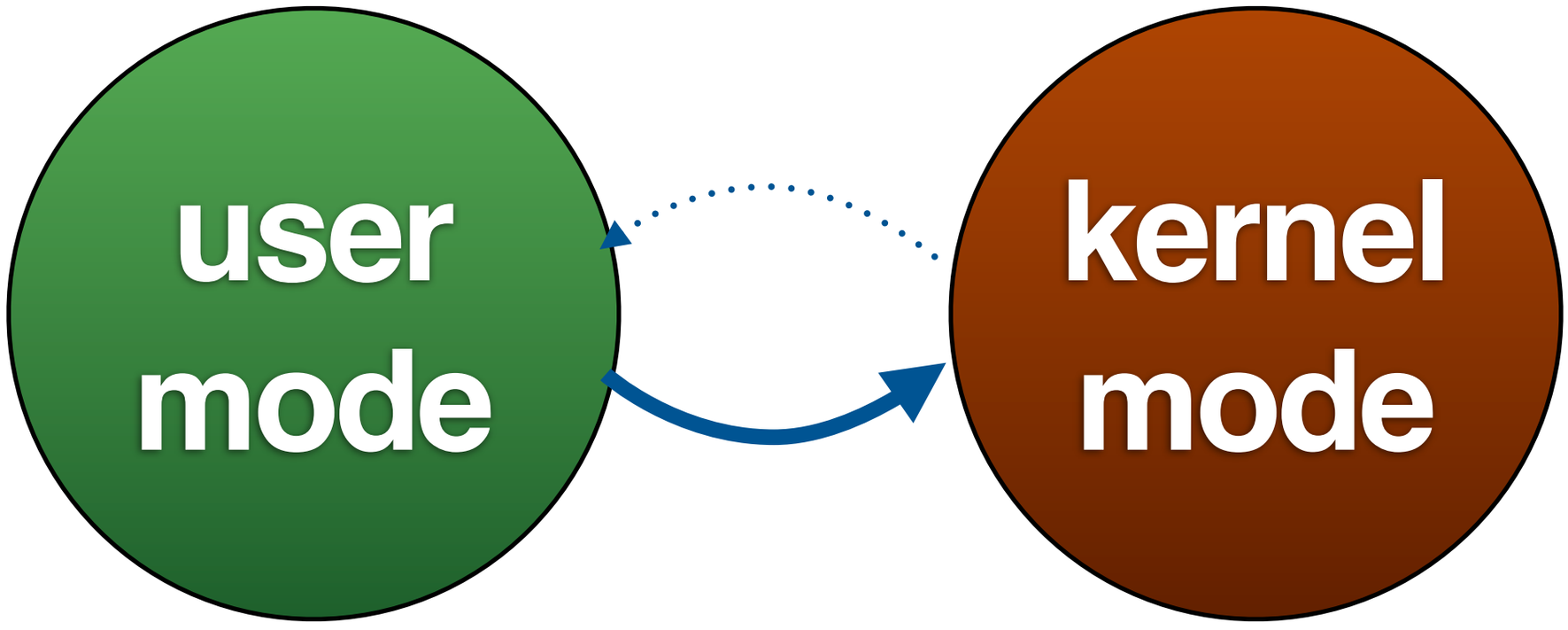
Address Space



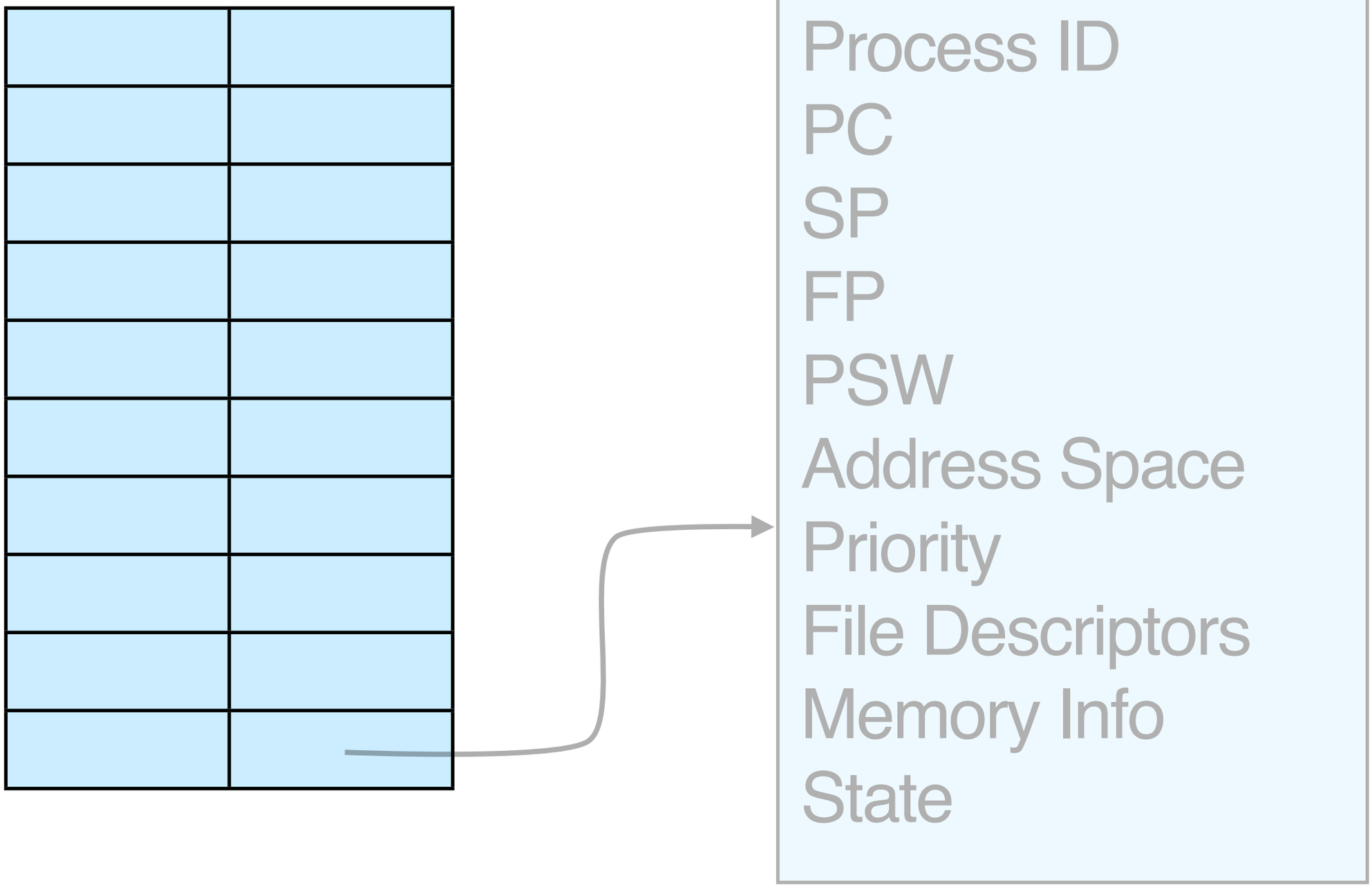

CPU
Signal Handlers
0x0043bef4
Signal Handler
code to handle
SIGCHLD
PC
0x0043bef4
SP
FP
PSW

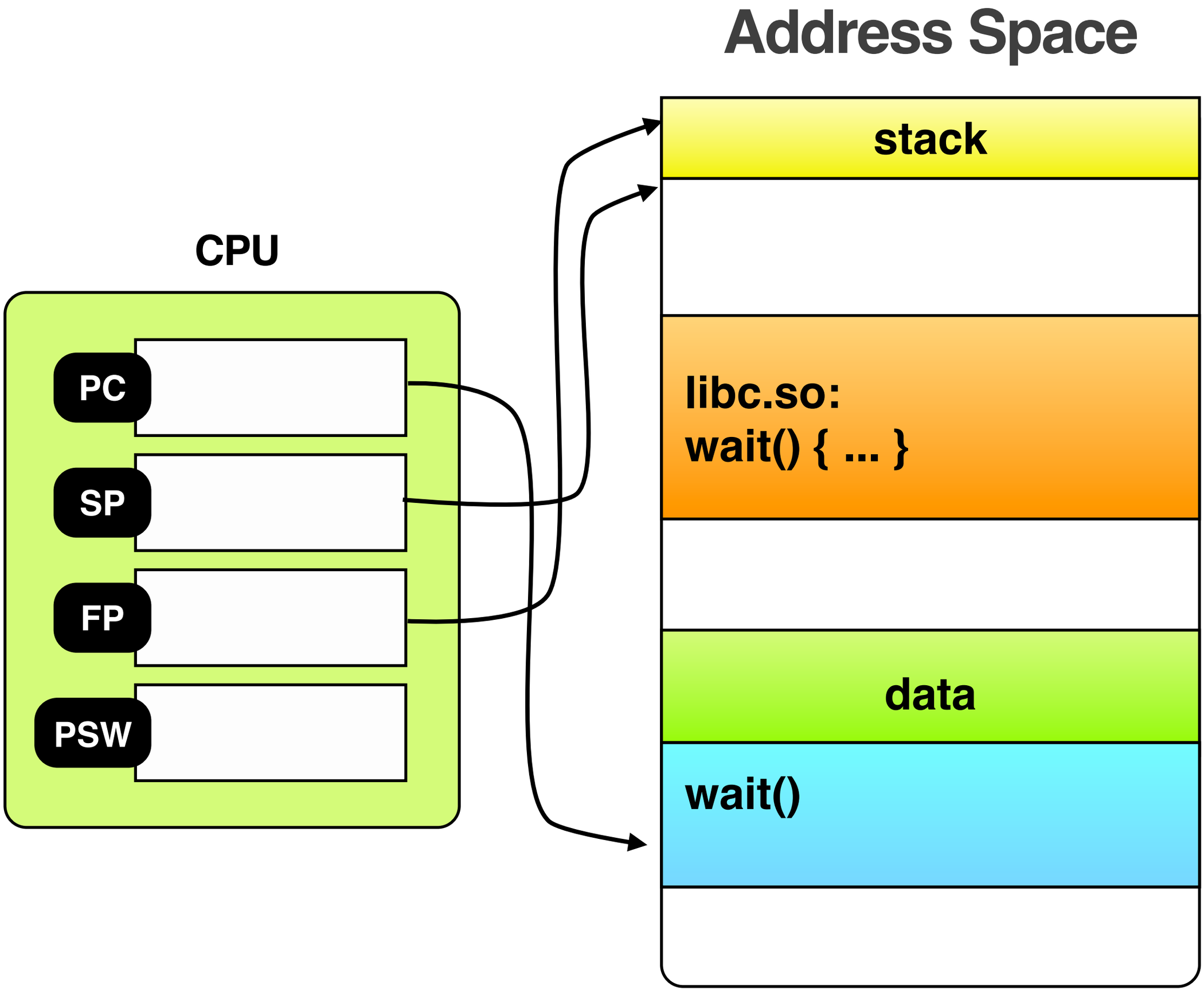

user
mode
kernel
mode


Signal Handlers
0x0043bef4
Signal Handler
code to handle
SIGCHLD
CPU
PC
0x0043bef4
SP
FP
PSW


user
mode
kernel
mode

Process Table




Address Space
CPU
PC
SP
FP
PSW
stack
libc.so:
wait() { ... }
data
wait()

```
bash$ ls
foo bar a.out lab.c
bash$
```

what to learn from OS course (beside OS):

1. complex systems

2. abstraction + interface design

3. concurrency

4. resource management

5. performance trade-off

OS “Bag of Tricks”

add a layer

to hide complexity

caching

to exploit locality

locking

to prevent race conditions

learning from history

to predict future

important stuff that we missed..

OS for Multi-Processors

Multimedia OS

Real-Time OS

Mobile OS

Virtual Machines

Security

THE END