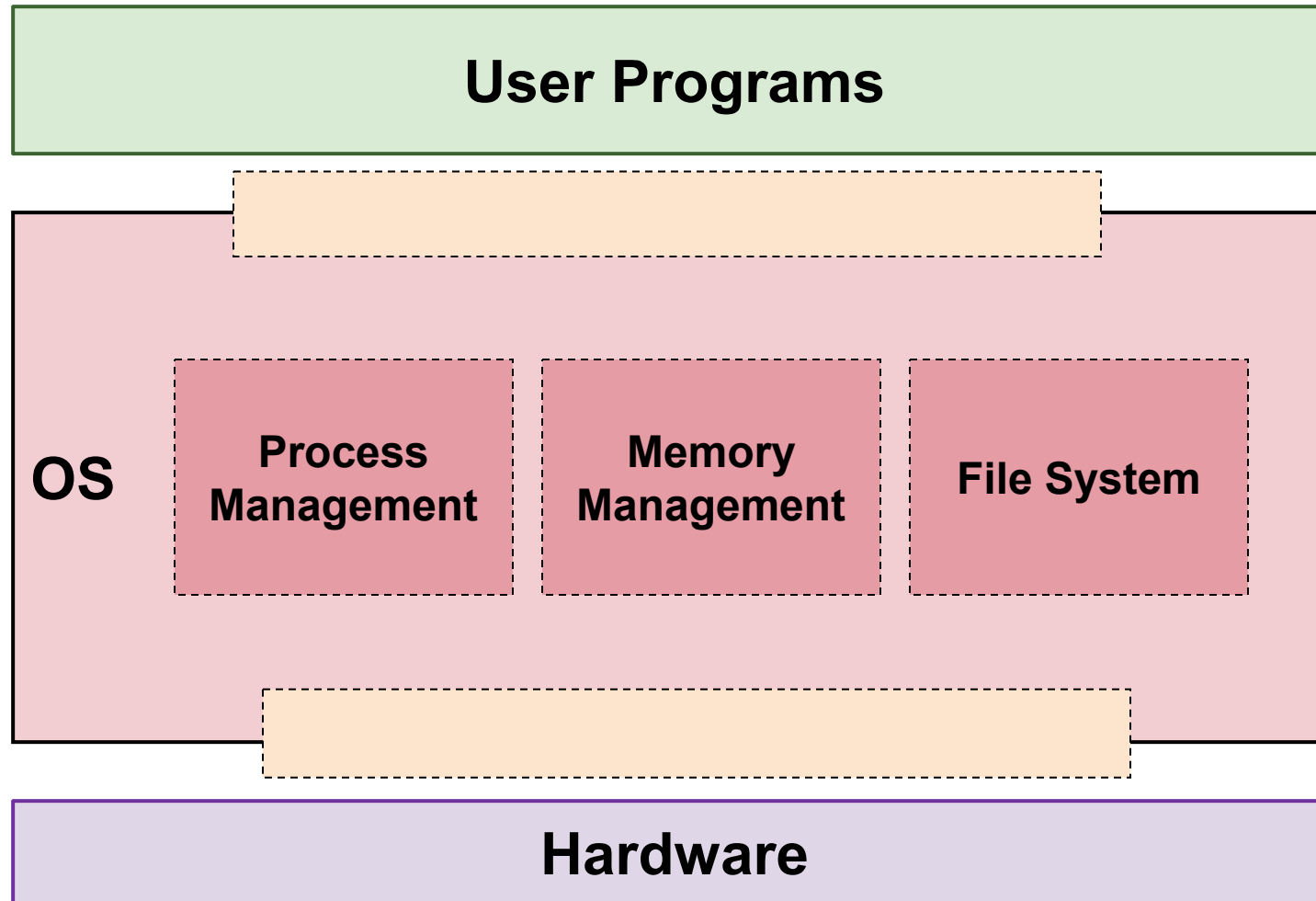


# Big Bang & Revision

---

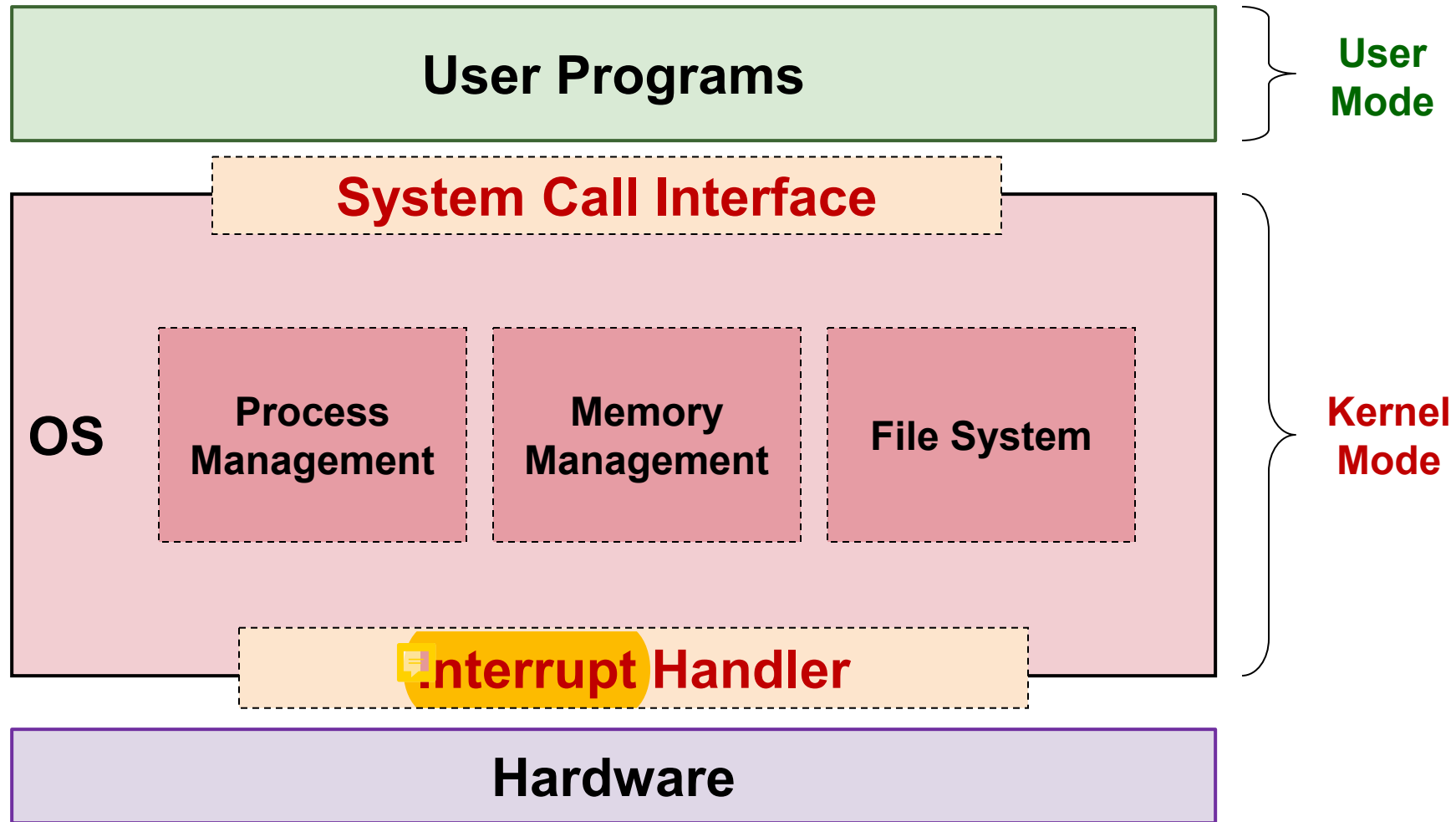
Lecture 12 (**Live** Version)

# OS Structure (Monolithic)



# OS Structure (Monolithic)

exception is generated by the instructions which are run by the user or OS  
ie: exception happen due to the code ran by user / OS



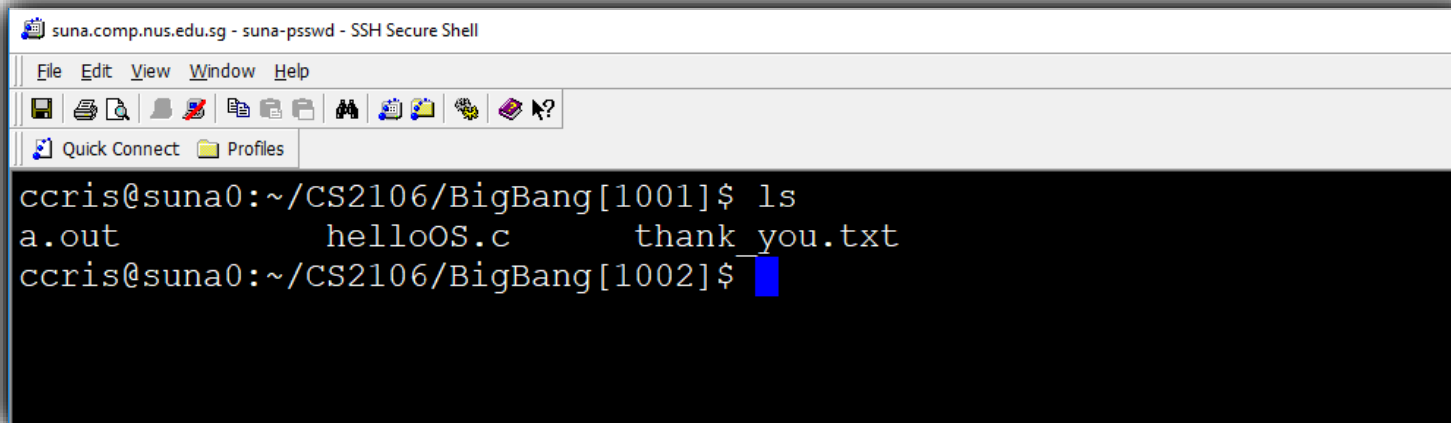
---

*23 + 11 + **lots of** hours just to learn about what happened in **1 second***

# THE STORY OF SHELL INTERPRETER

---

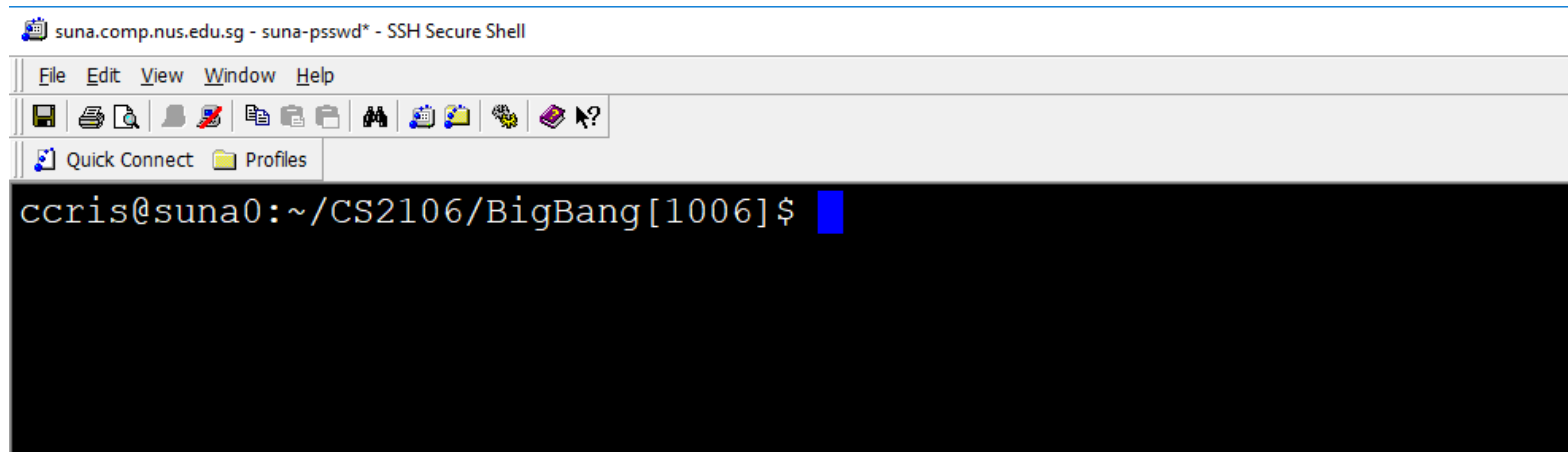
Ah.... Ah.... so simple..... **or is it?**



The image shows a screenshot of an SSH terminal window. The title bar reads "suna.comp.nus.edu.sg - suna-psswd - SSH Secure Shell". Below the title bar is a menu bar with "File", "Edit", "View", "Window", and "Help". Underneath the menu bar is a toolbar with various icons for file operations like save, print, and search. Below the toolbar is a tab bar with "Quick Connect" and "Profiles". The main area of the terminal is black with white text. It shows the prompt "ccris@suna0:~/CS2106/BigBang[1001]" followed by the command "ls". The output of the command is listed on the next line: "a.out", "helloOS.c", and "thank\_you.txt". The prompt then changes to "ccris@suna0:~/CS2106/BigBang[1002]" followed by a blue cursor.

```
suna.comp.nus.edu.sg - suna-psswd - SSH Secure Shell
File Edit View Window Help
[Toolbar icons]
Quick Connect Profiles
ccris@suna0:~/CS2106/BigBang[1001]$ ls
a.out          helloOS.c      thank_you.txt
ccris@suna0:~/CS2106/BigBang[1002]$
```

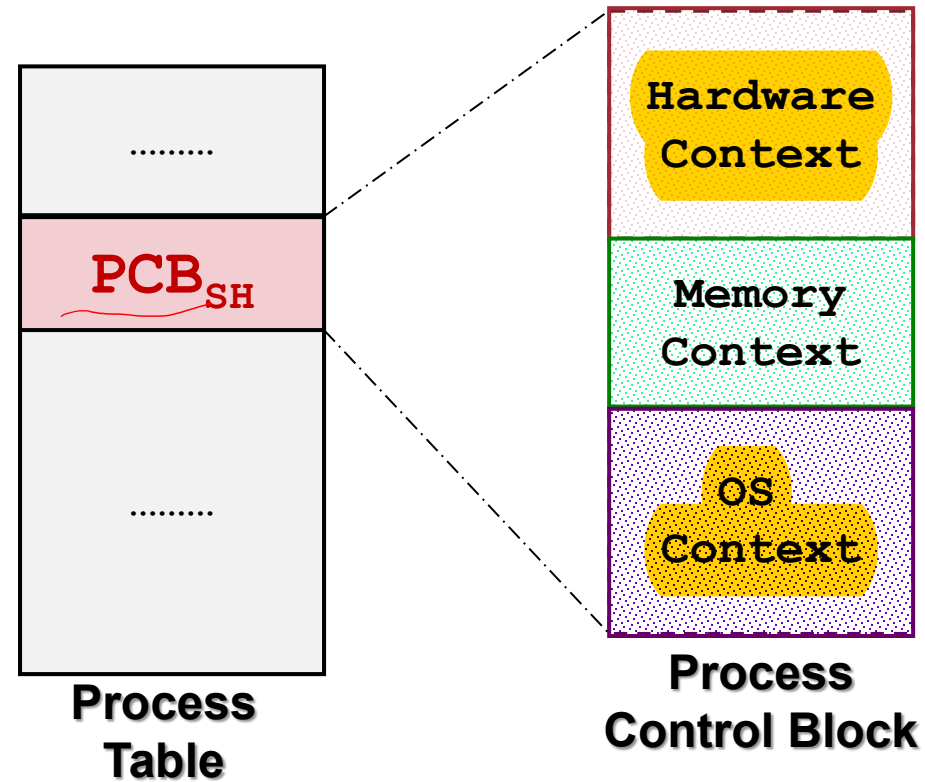
# Here we *go* . . . .



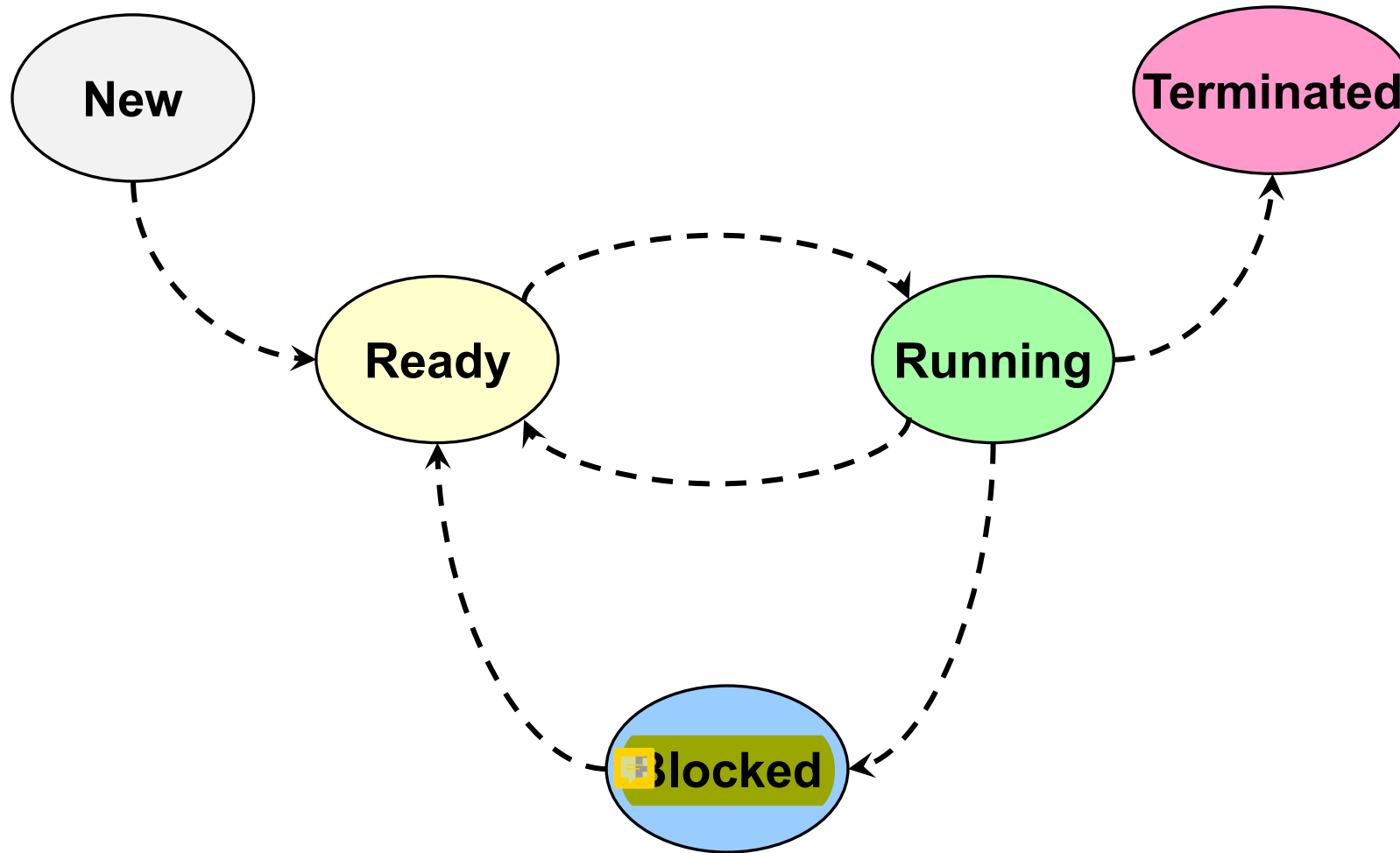
```
suna.comp.nus.edu.sg - suna-psswd* - SSH Secure Shell
File Edit View Window Help
[Toolbar icons]
Quick Connect Profiles
ccris@suna0:~/CS2106/BigBang[1006]$
```

# Process Table & Process Control Block

the shell interpreter is waiting for input - and since it is a process it will have an entry in the process table and its own process control block



# The Interpreter is in.....



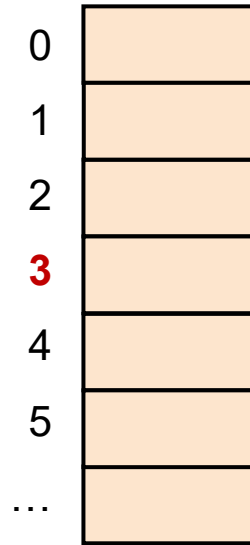


# User press "1"



if (iRQ == 3) then  
call Keyboard Handler();

or in this case can use function  
pointers to call the handler



## Keyboard Handler

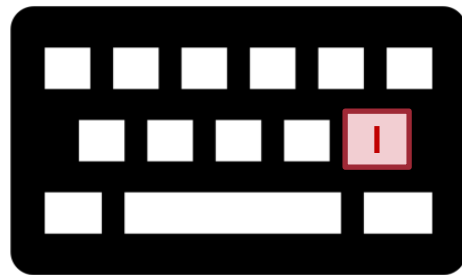
$\$r1 \leftarrow \text{key pressed}$

$\text{mem}[\$r2] \leftarrow \$r1$

Done!

## Interrupt Handler

3



Process X:  
Hey! ☹️



PC 123

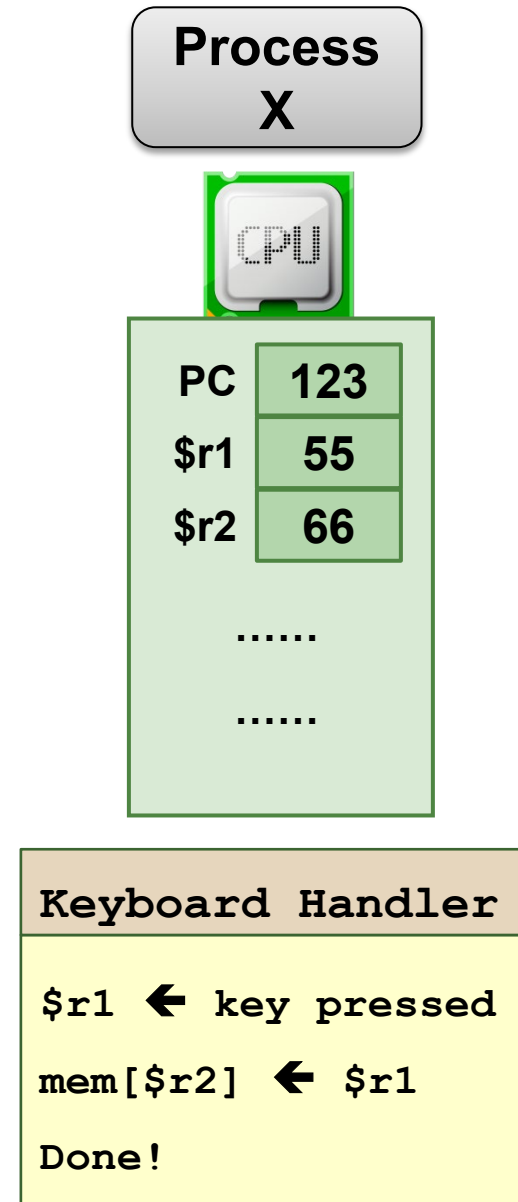
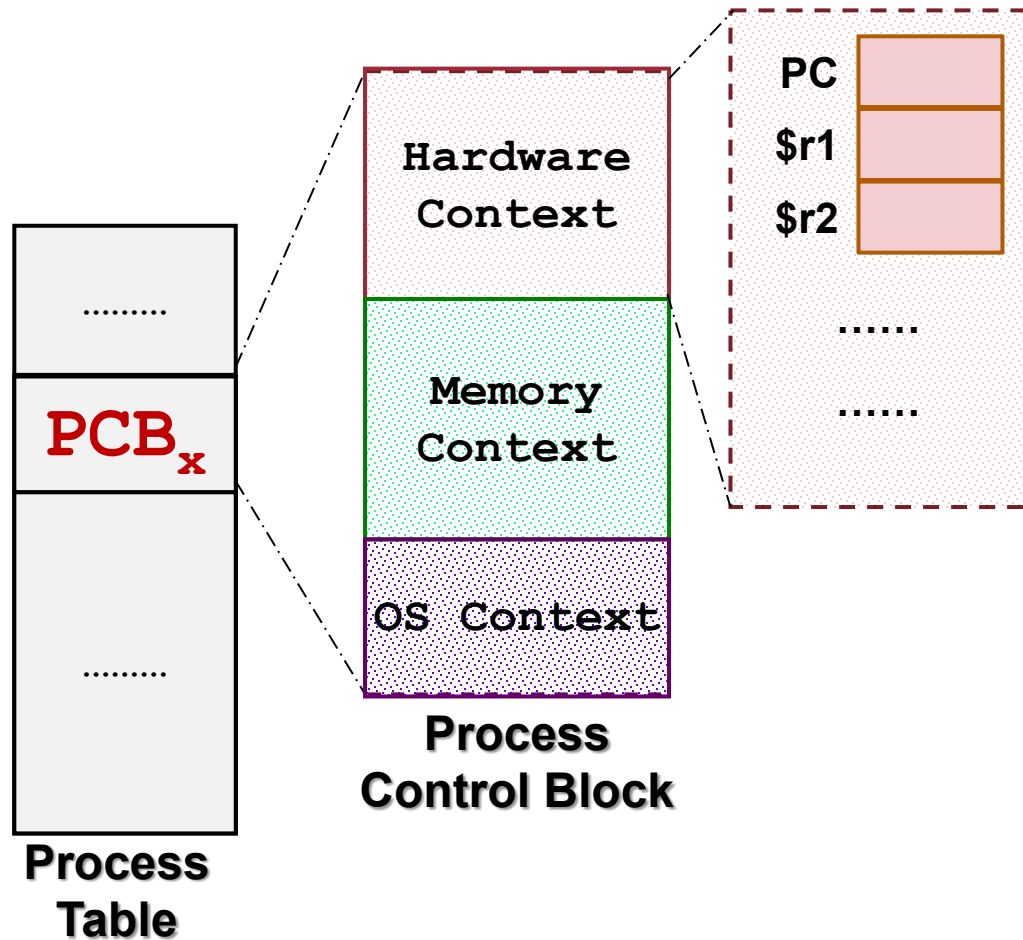
$\$r1$  55

$\$r2$  66

.....

.....

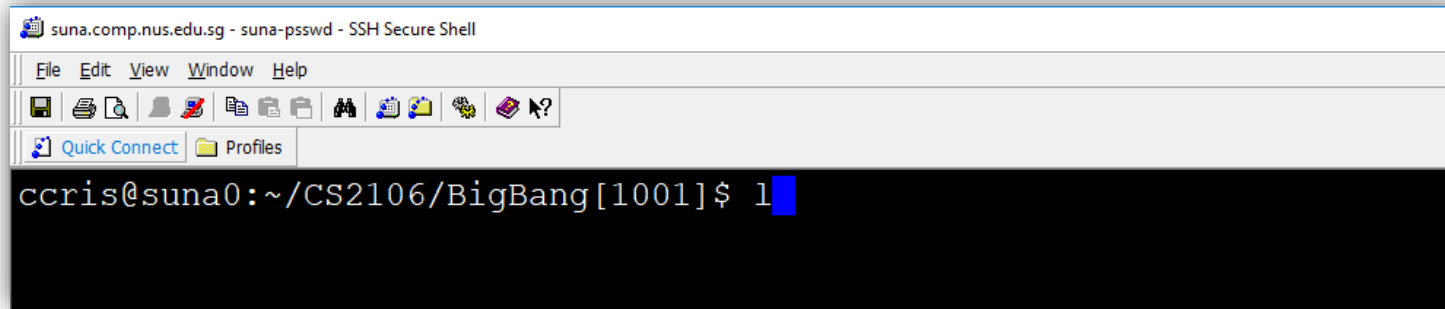
Sorry to **interrupt** you....



# Interrupt steps

- Give the sequence of steps for handling an interrupt.
  - ❑ Interrupt occurs
  - ❑ Save registers/CPU state
  - ❑ Perform the handler routine
  - ❑ Restore registers/CPU state
  - ❑ Return from interrupt

# Rinse and Repeat.....

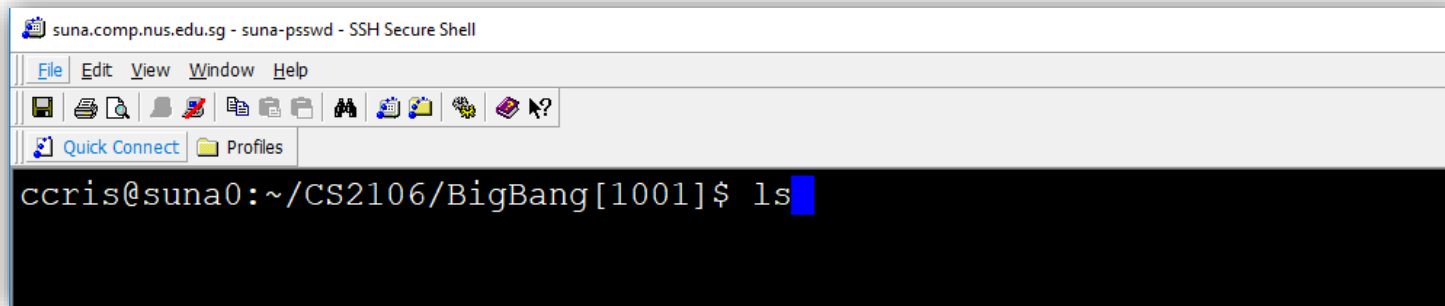


sunu.comp.nus.edu.sg - sunu-psswd - SSH Secure Shell

File Edit View Window Help

Quick Connect Profiles

```
ccris@sunu0:~/CS2106/BigBang[1001]$ l
```

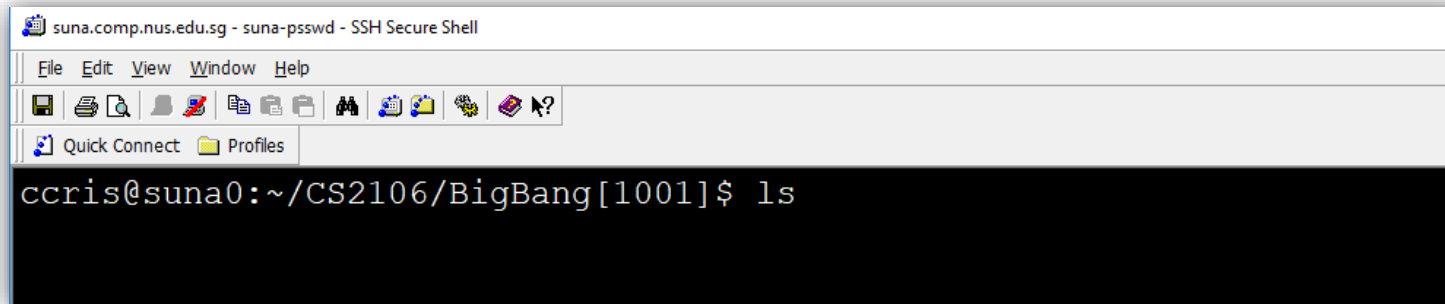


sunu.comp.nus.edu.sg - sunu-psswd - SSH Secure Shell

File Edit View Window Help

Quick Connect Profiles

```
ccris@sunu0:~/CS2106/BigBang[1001]$ ls
```



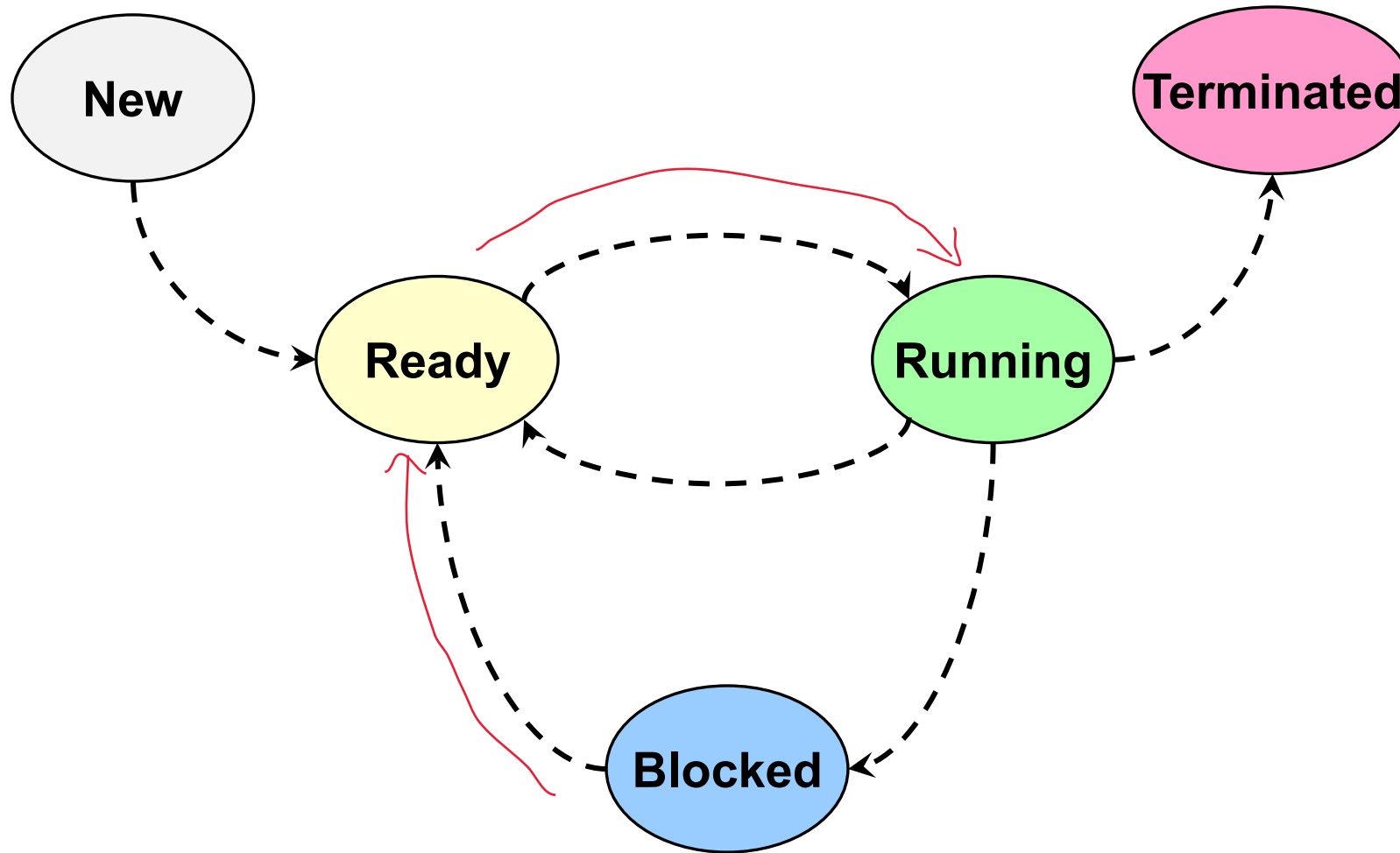
sunu.comp.nus.edu.sg - sunu-psswd - SSH Secure Shell

File Edit View Window Help

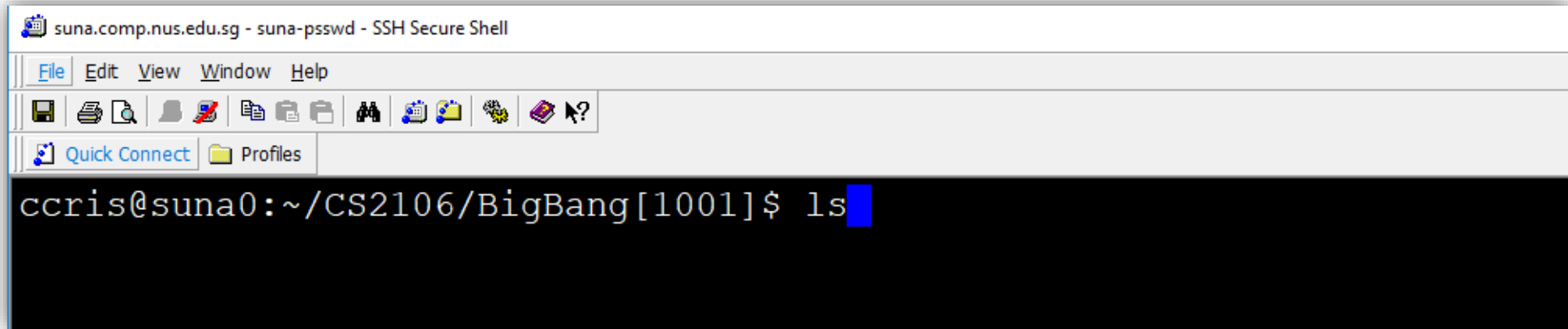
Quick Connect Profiles

```
ccris@sunu0:~/CS2106/BigBang[1001]$ ls
```

The **interpreter** is now...



User entered "ls", the interpreter will...



The screenshot shows a terminal window titled "suna.comp.nus.edu.sg - suna-psswd - SSH Secure Shell". The window has a menu bar with "File", "Edit", "View", "Window", and "Help". Below the menu bar is a toolbar with various icons. The terminal text shows the prompt "ccris@suna0:~/CS2106/BigBang[1001]" followed by the command "ls" which is highlighted with a blue cursor.

```
suna.comp.nus.edu.sg - suna-psswd - SSH Secure Shell
File Edit View Window Help
[Toolbar icons]
Quick Connect Profiles
ccris@suna0:~/CS2106/BigBang[1001]$ ls
```

# Typical Steps for Shell Interpreter

UserCmd ← read from keyboard

fork()



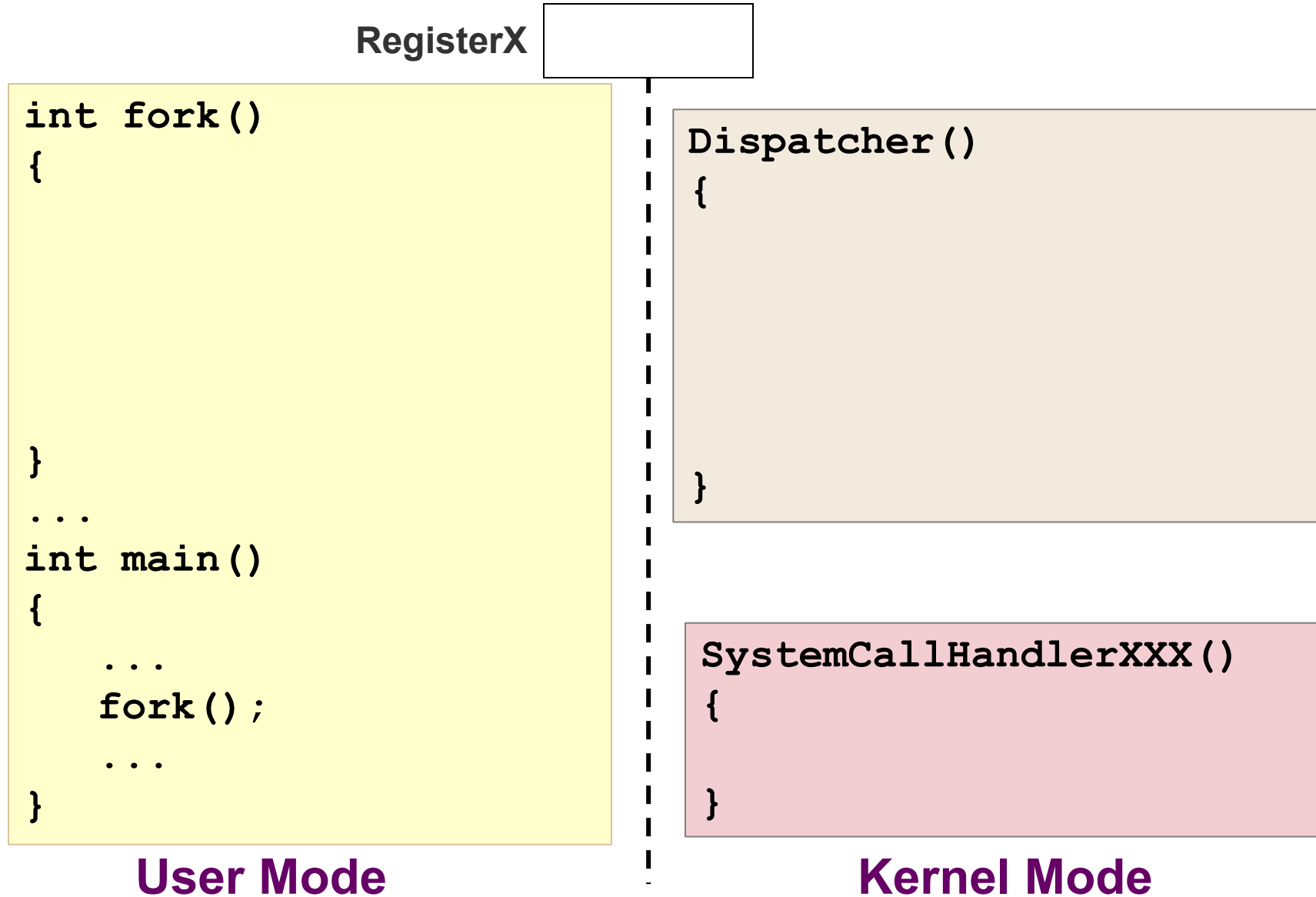
if I am the parent (i.e. the shell)

wait ( child to finish )

else

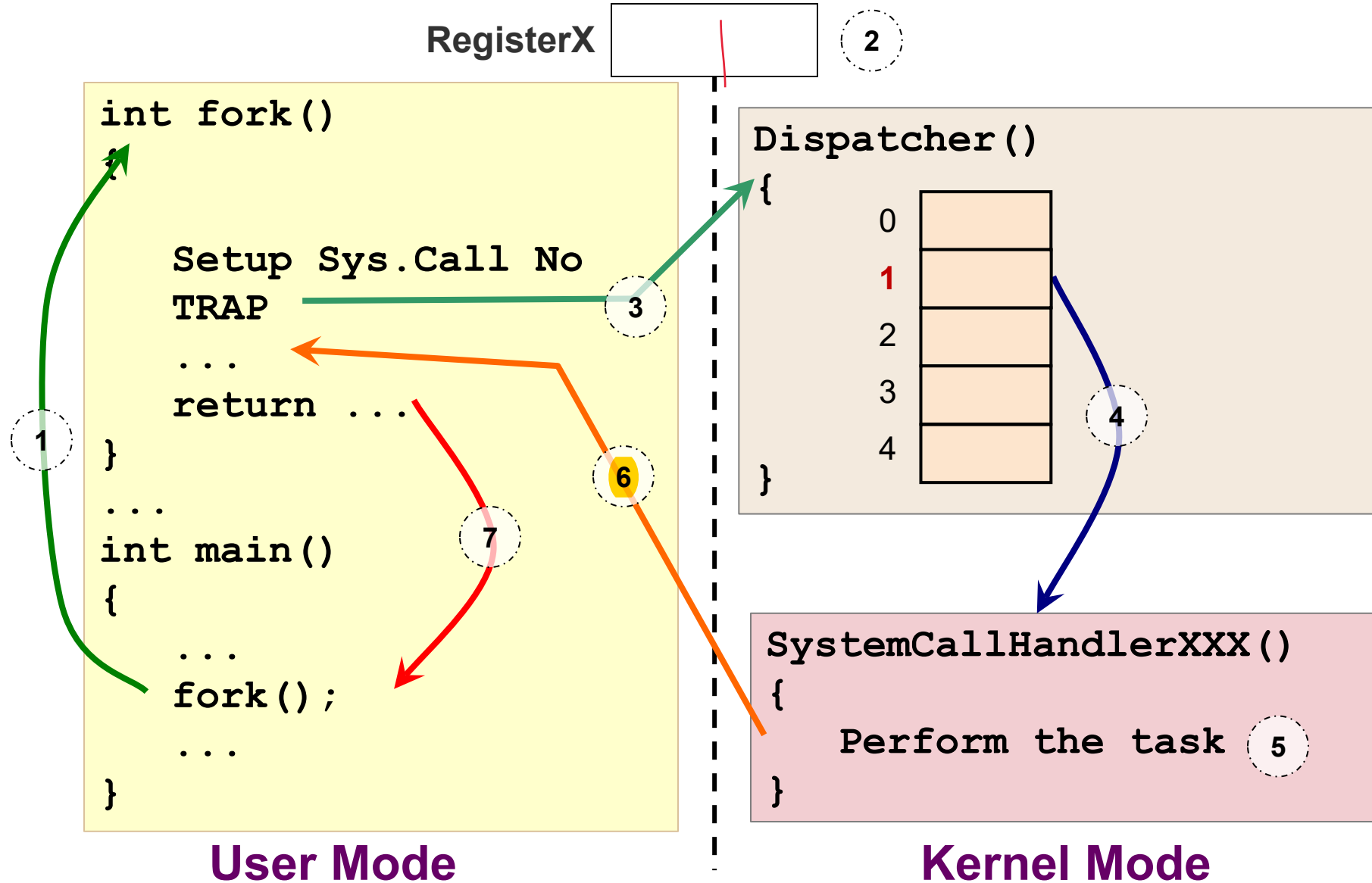
exec ( UserCmd )

# **fork()** involves a **system call**

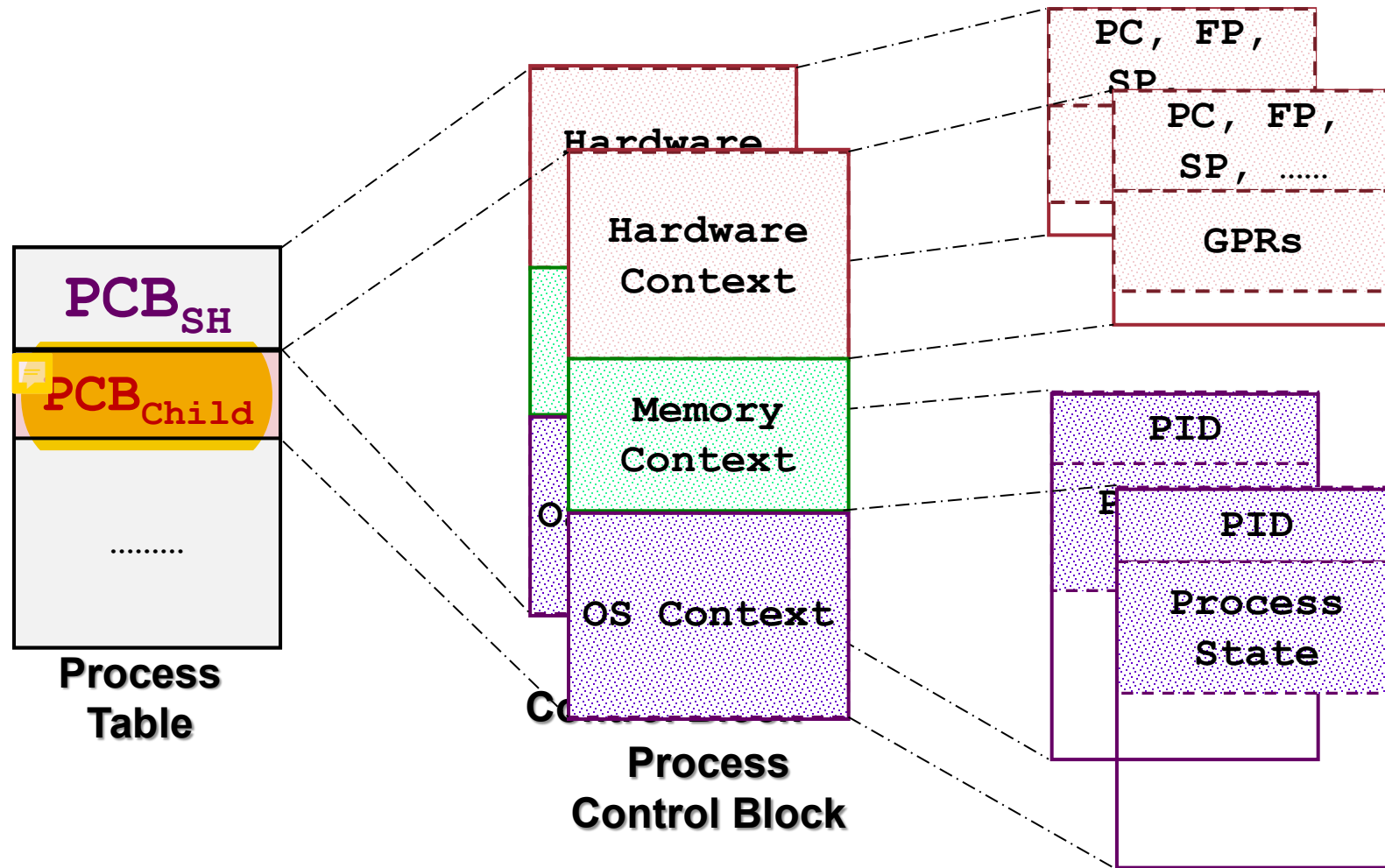




# **fork()** involves a **system call**

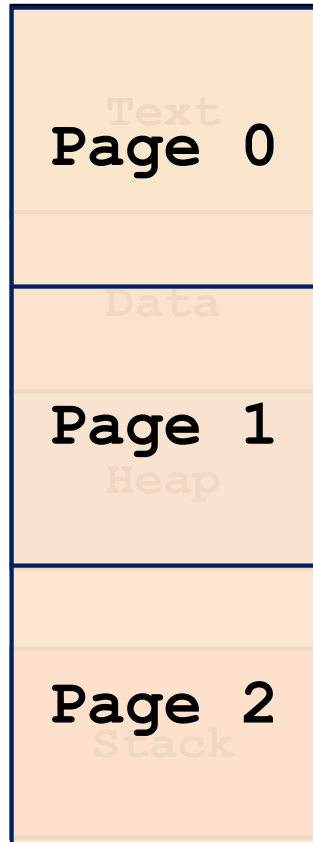


What is the **effect** of `fork( )`?



# Memory Space of a Process

virtual address space



Process  
Memory Space

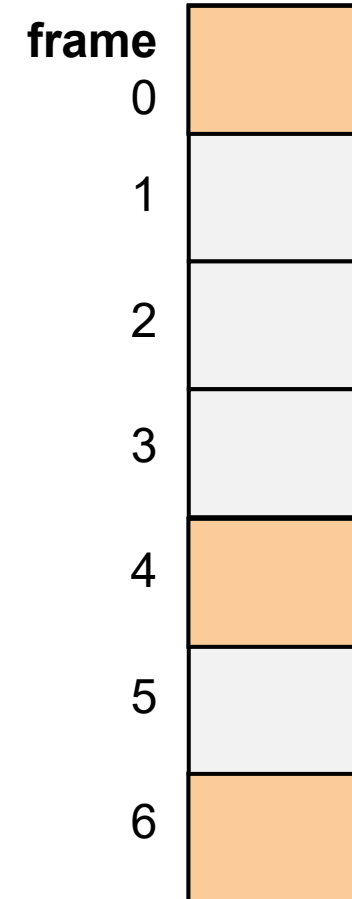
0	4
1	6
2	0

Page Table

p0	f4
p2	f0

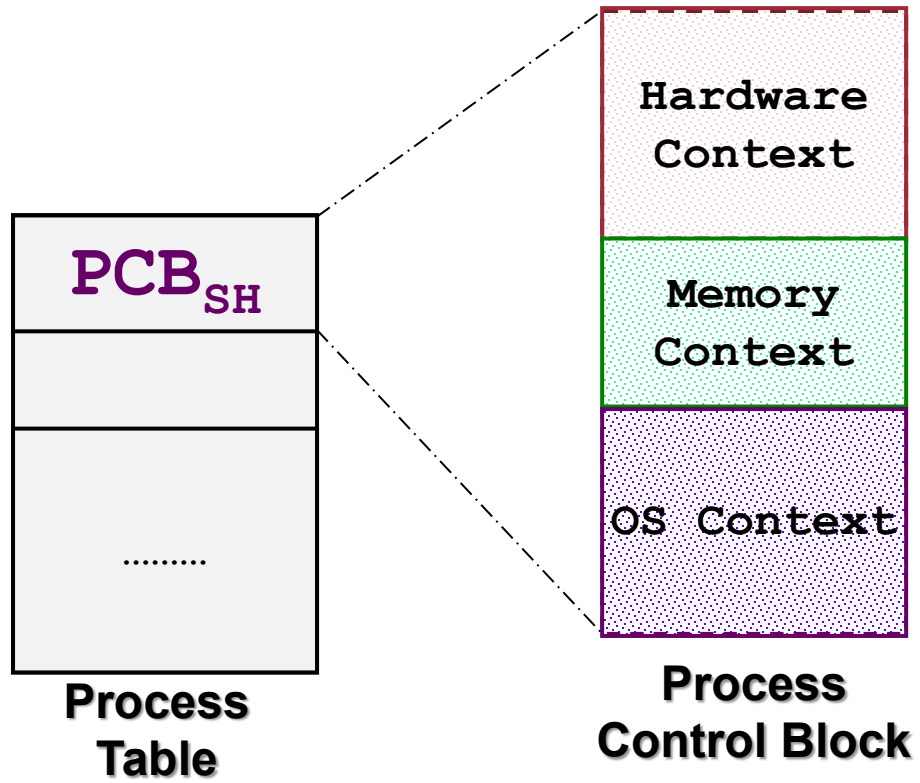
LB

physical address space



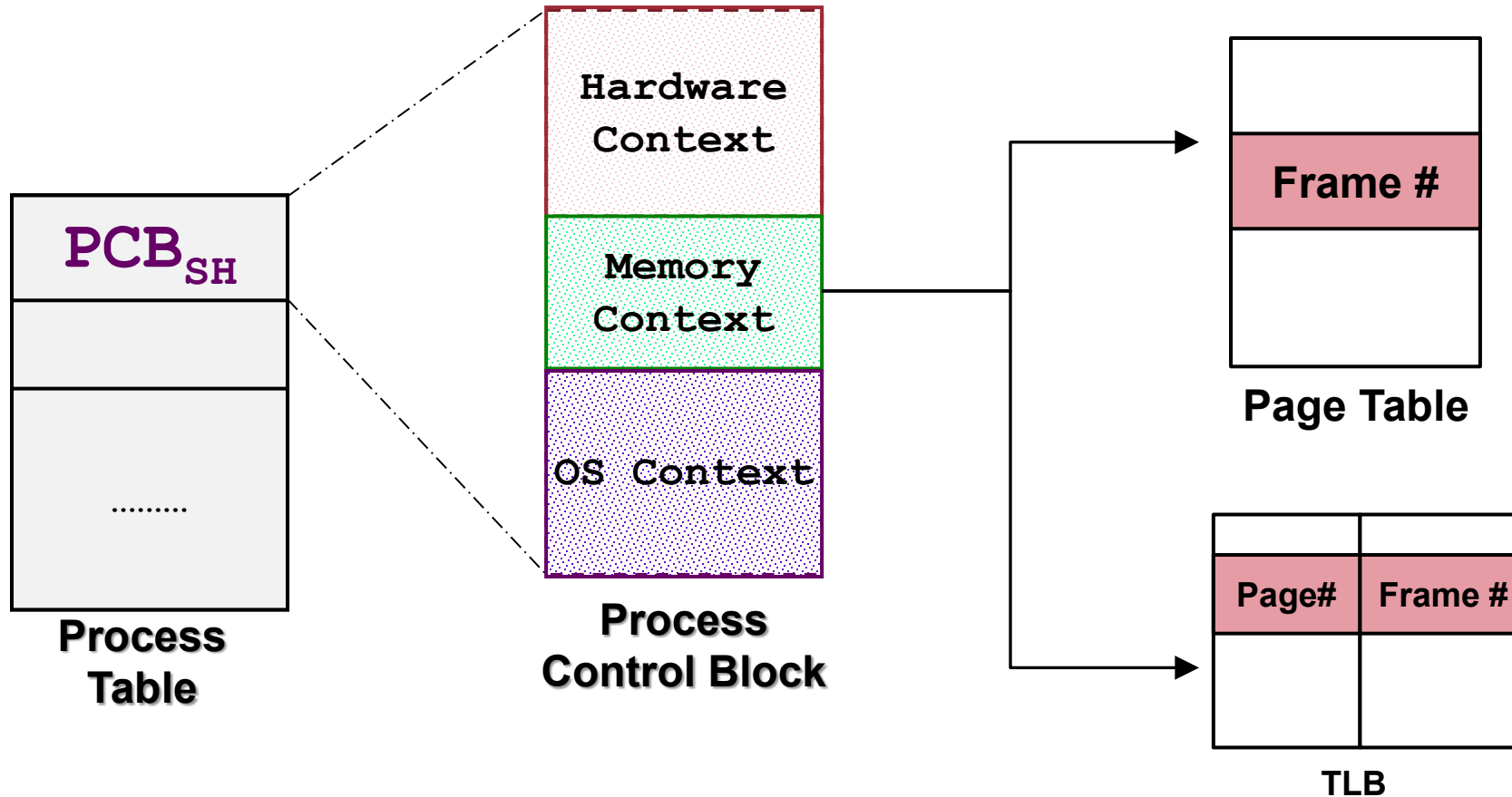
Physical  
Memory

# Memory Context = ?



# Memory Context = ?

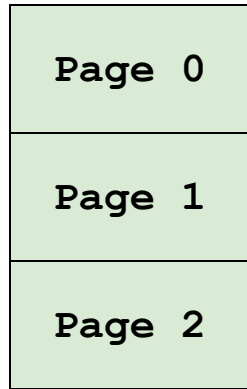
memory context will store a pointer to the Page Table (must) and TLB (not all the time)



What is the benefit of "duplication"

**A**

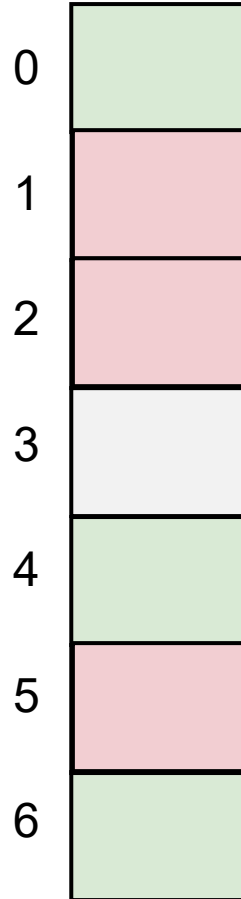
**B**



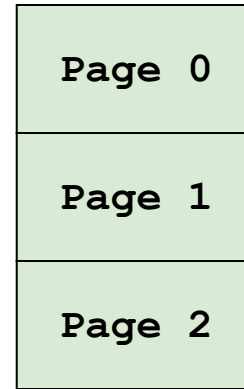
Memory Space

page table

0	4
1	6
2	0



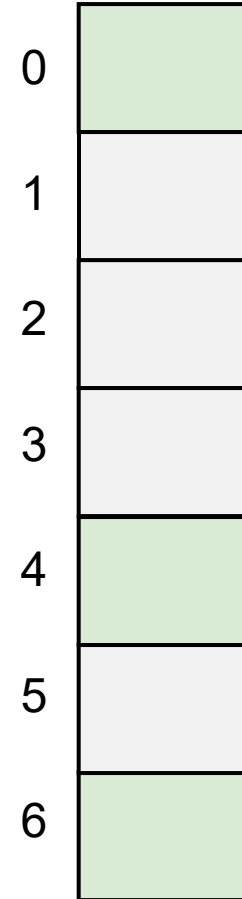
physical memory



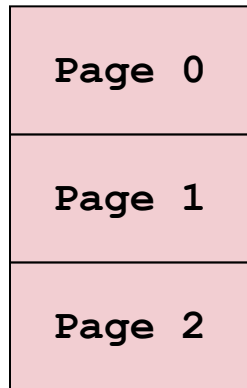
Memory Space

page table

0	4
1	6
2	0



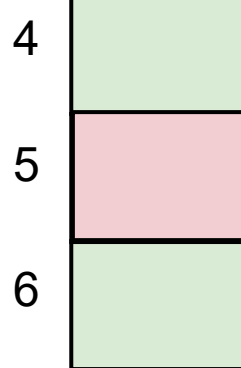
physical memory



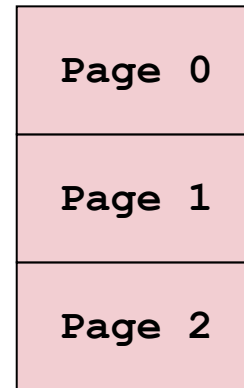
Memory Space

page table

0	5
1	2
2	1



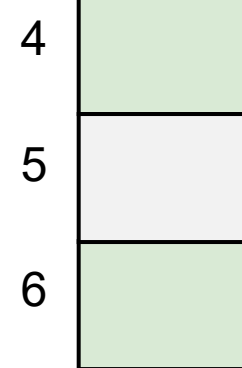
physical memory



Memory Space

page table

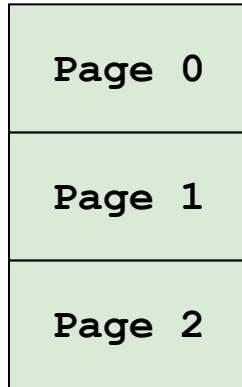
0	4
1	6
2	0



physical memory

# Duplicating Memory Space the **HARD WAY**

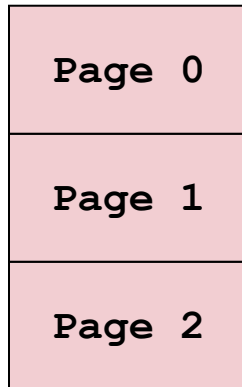
no need to do this since most of the time child process does not require to change much of the pre existing data



**Memory  
Space**

0	4
1	6
2	0

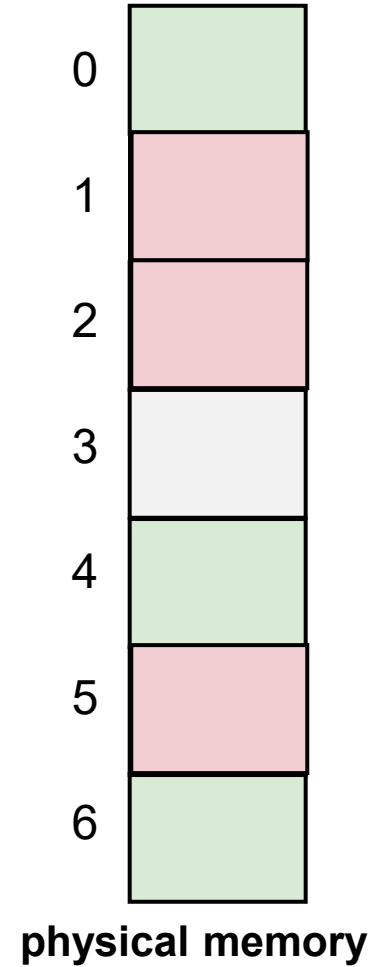
page table



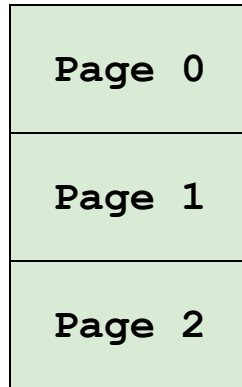
**Memory  
Space**

0	5
1	2
2	1

page table



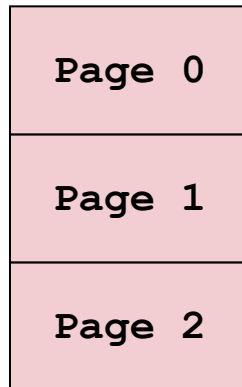
# Copy on Write



**Memory  
Space**

0	4
1	6
2	0

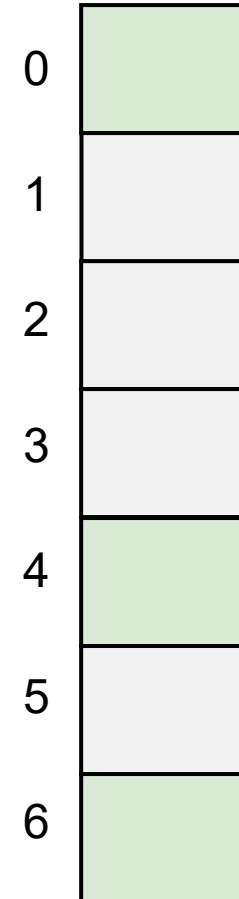
page table



**Memory  
Space**

0	4
1	6
2	0

page table

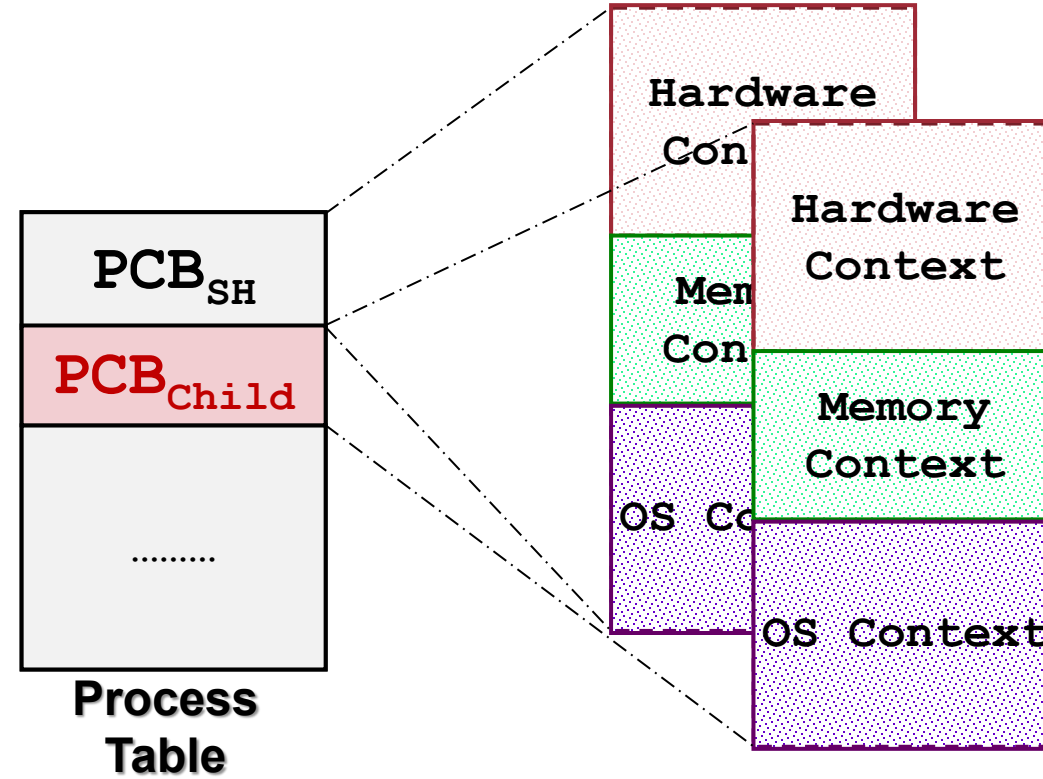


**physical memory**

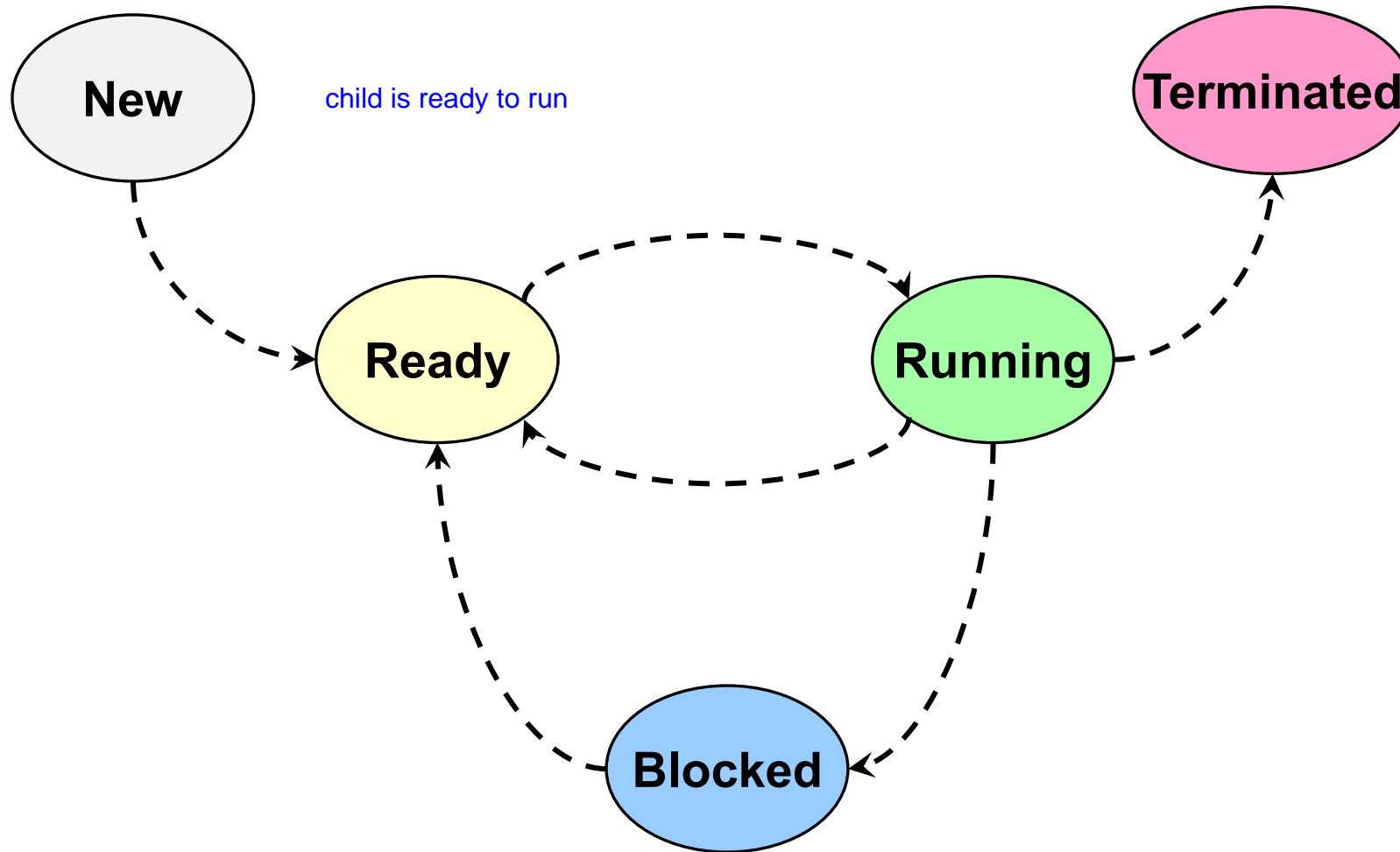
initially the child will only have read permissions - when an exception occurs this will mean that the child is going to write something and thus now there is a need to allocate new frame for this edited page



So, Effect of **fork**( )

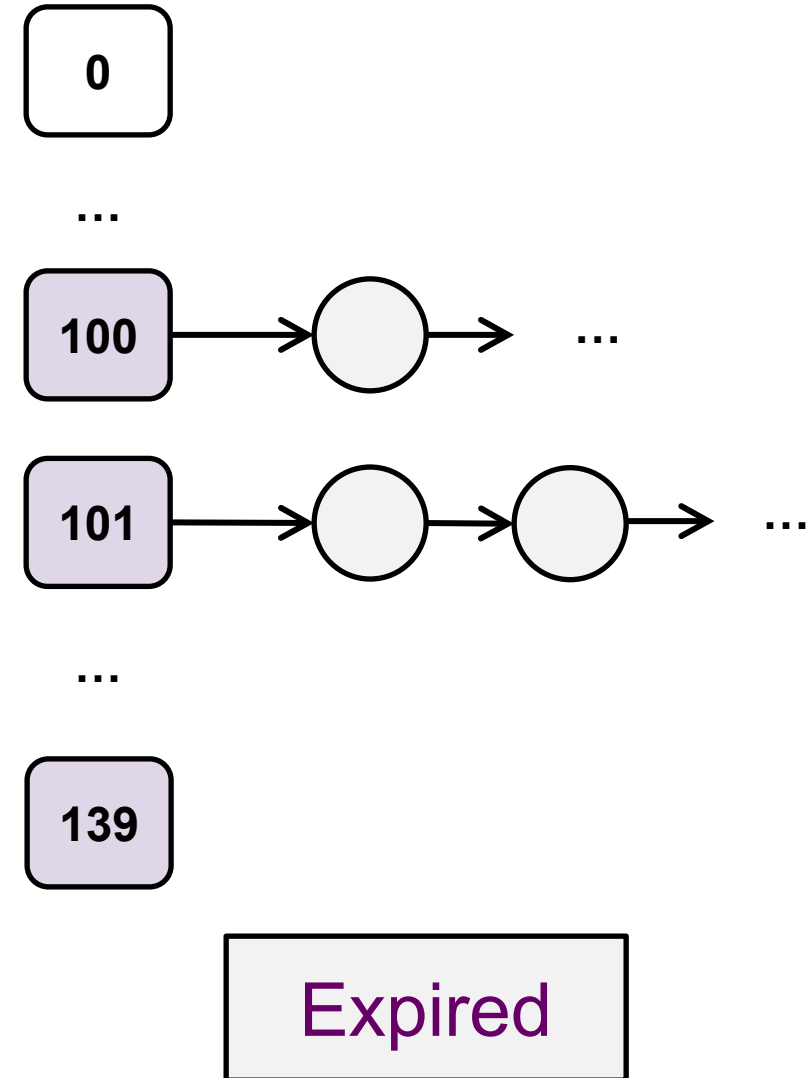
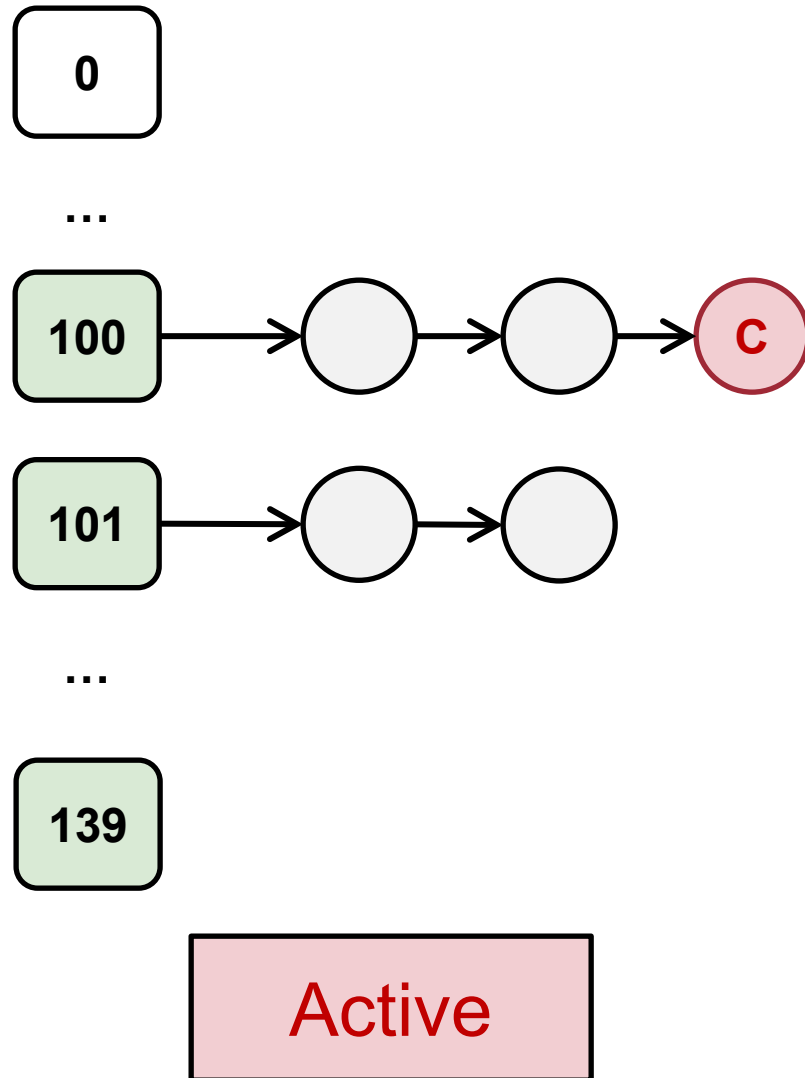


# Child, welcome to the world!



# Child, welcome to the Queue!

MLFQ means child might have the highest priority to run



# **fork()** finishes and returns

RegisterX

```
int fork()
{
    Setup Sys.Call No
    TRAP
    ...
    return ...
}
...
int main()
{
    ...
    fork();
    ...
}
```

**User Mode**

Dispatcher()

{	
0	
<b>1</b>	
2	
3	
4	
}	

SystemCallHandlerXXX()

```
{
    Perform the task
}
```

**Kernel Mode**

# Typical steps for **Shell Interpreter**

```
UserCmd ← Read from keyboard
```

```
fork()
```

```
if I am the parent (i.e. the shell)
```

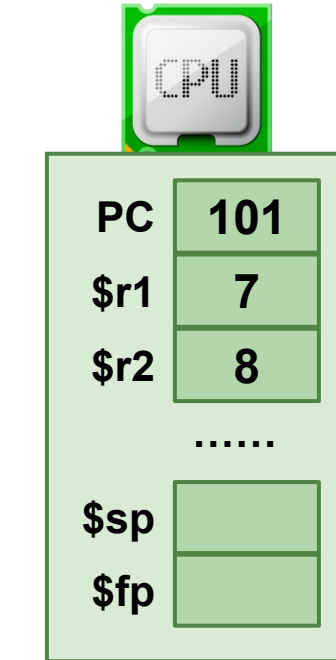
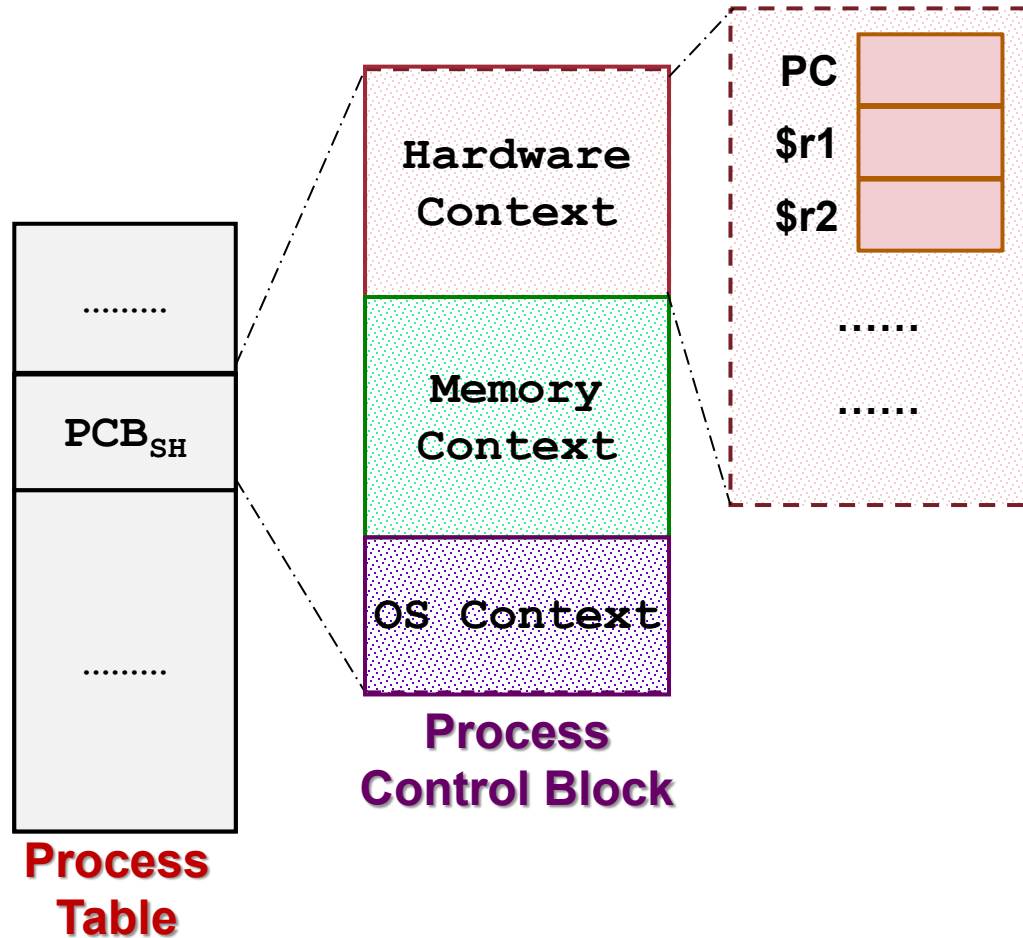
```
    wait ( child to finish )
```

```
else
```

```
    exec ( UserCmd )
```

**wait()**: The interpreter will....

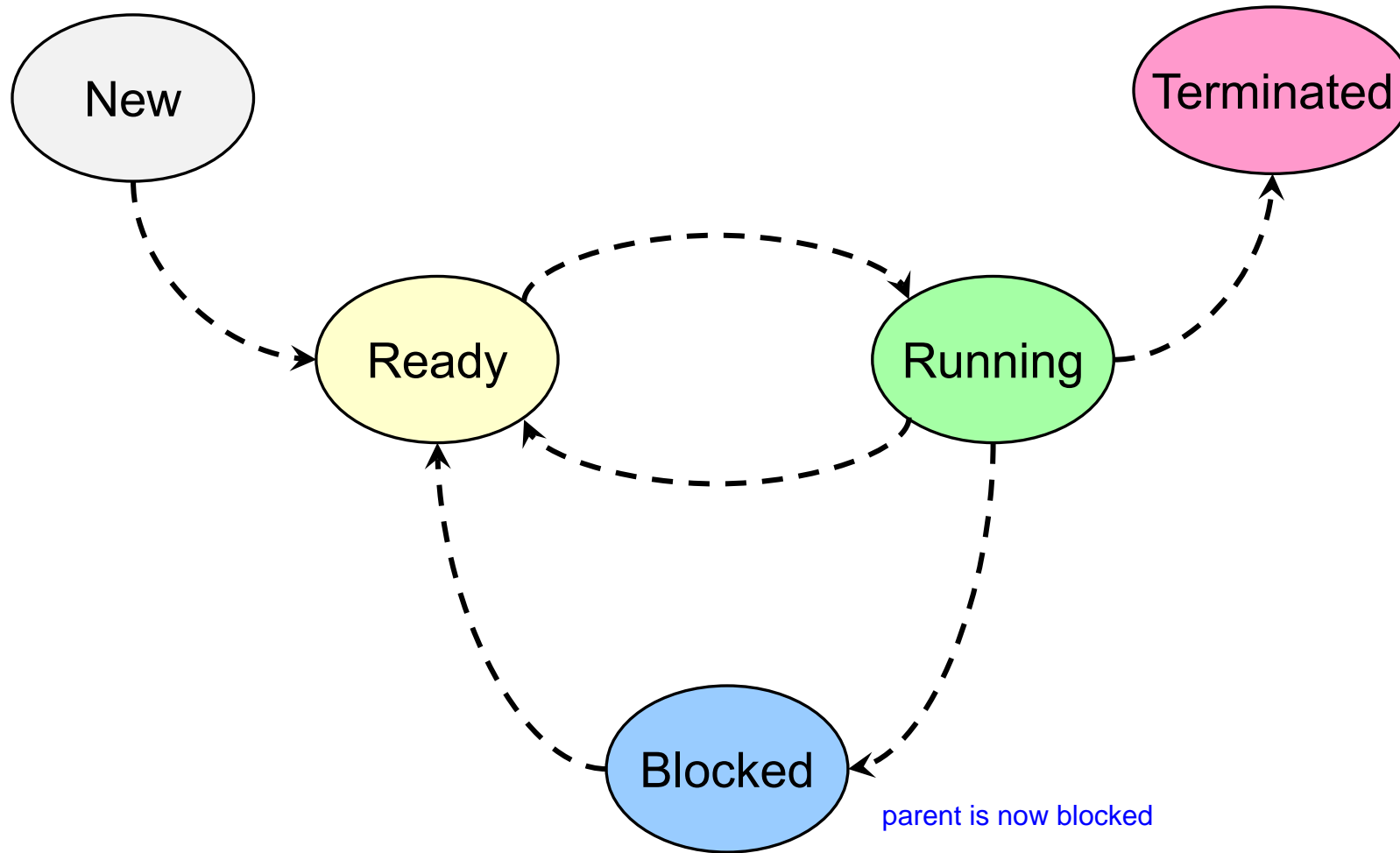
now the parent process will give up its processing time and save all its current information to the hardware context



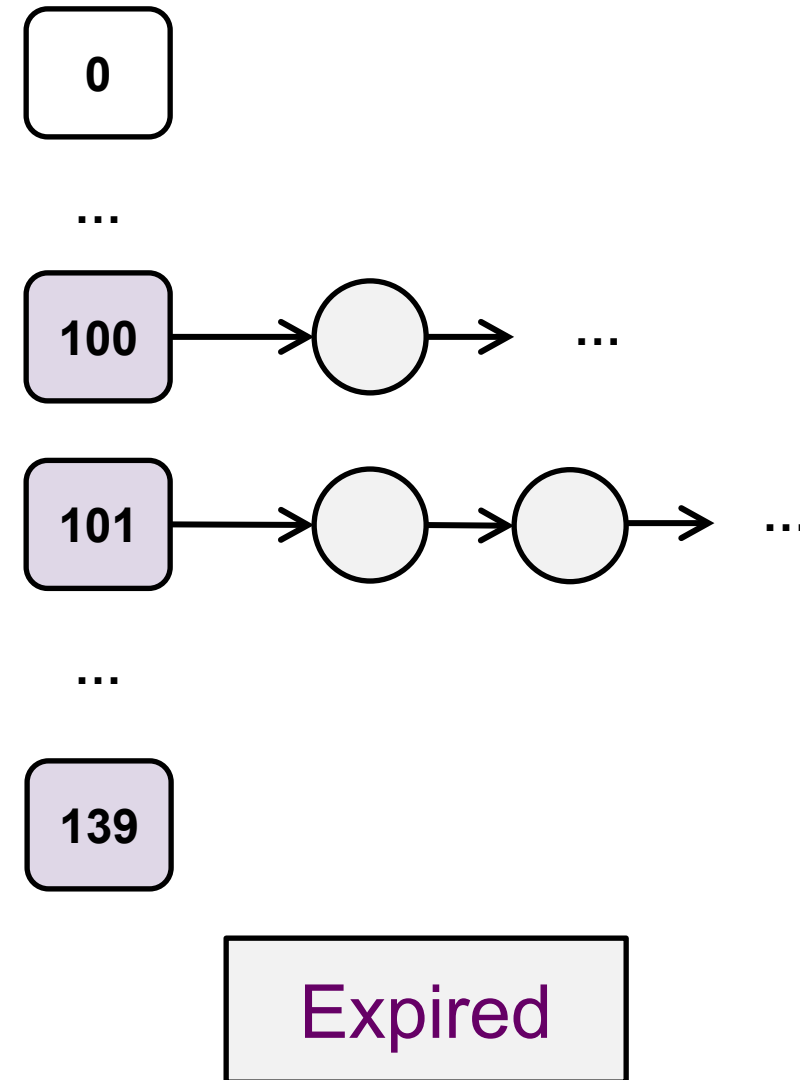
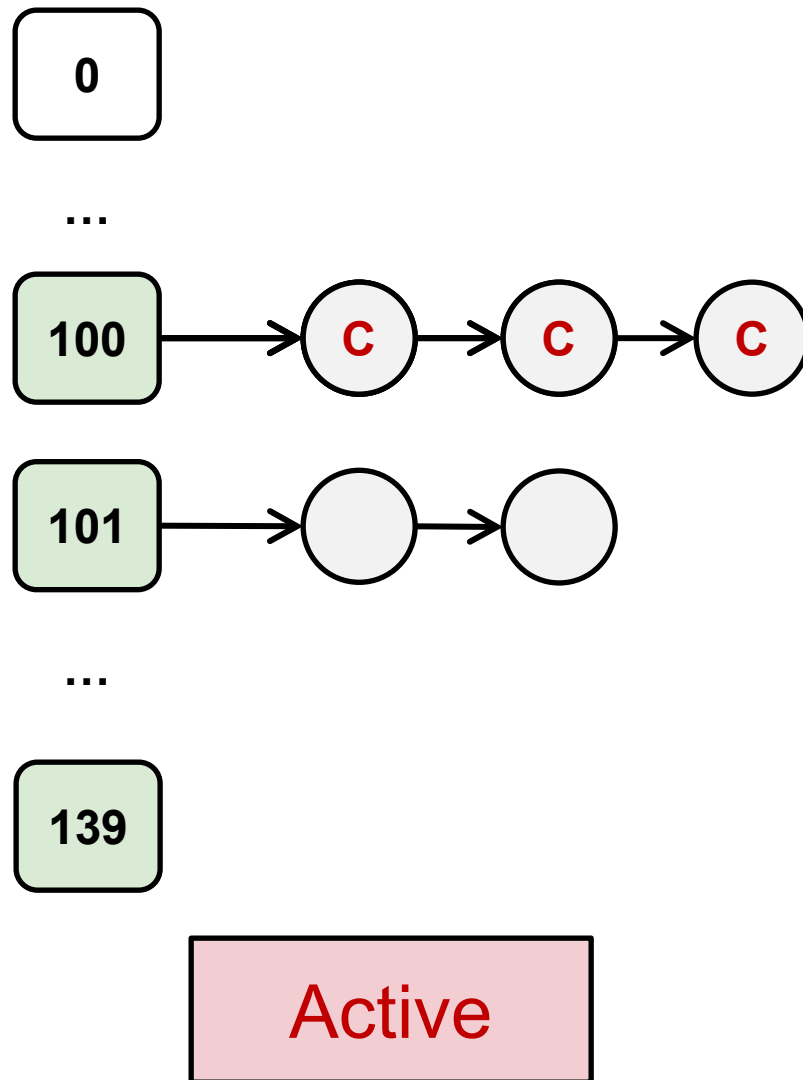
p0	f4
p2	f0

**TLB**

# The interpreter is now...



Hmm... **who** gets to **run**?





# Typical steps for **Shell Interpreter**

UserCmd ← Read from keyboard

**fork()**

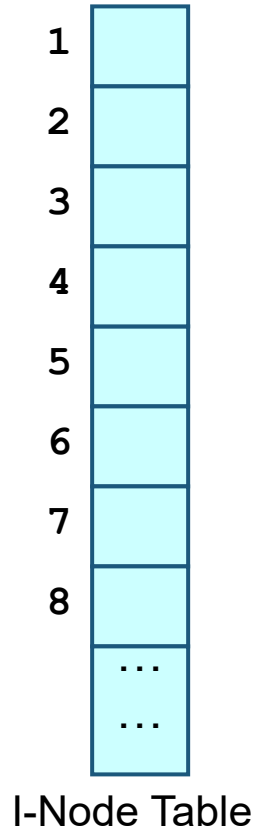
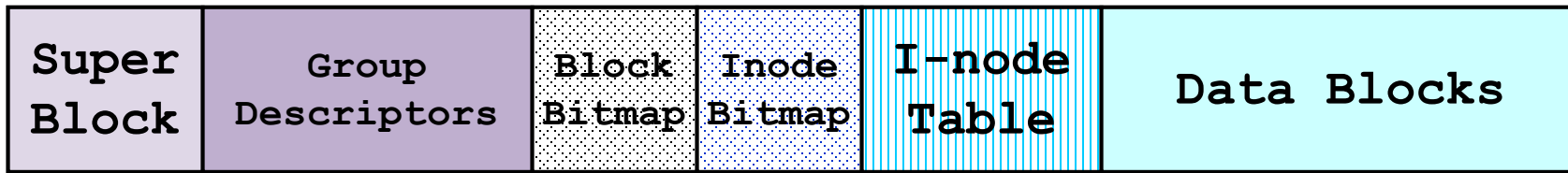
if I am the parent (i.e. the shell)

wait ( child to finish )

**else**

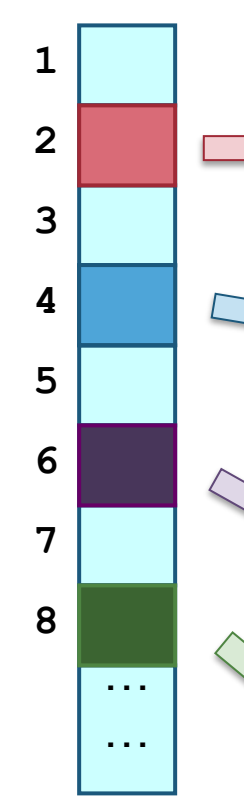
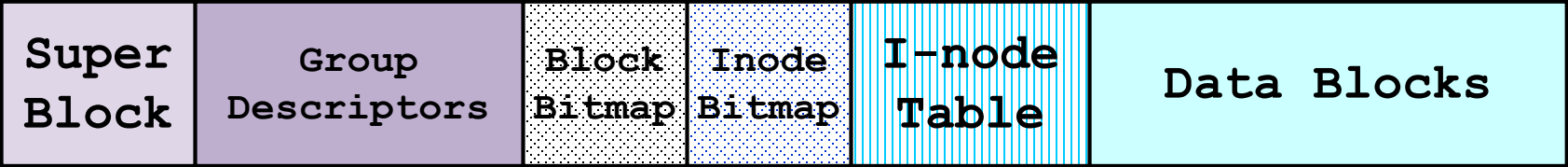
**exec ( UserCmd )**



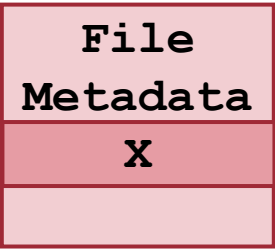
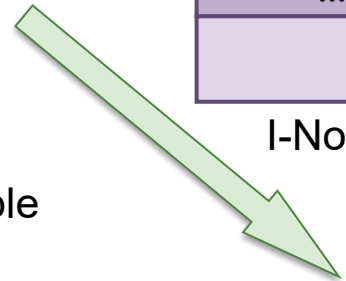
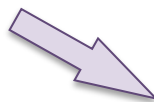
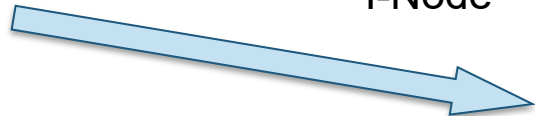


**/usr/bin/ls**

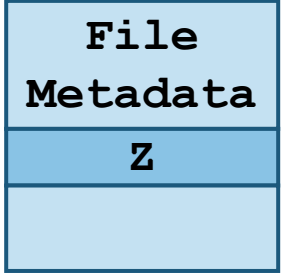
FYI - inodes can be accessed even if its not from the same group - since each group will only have a fixed number of inodes



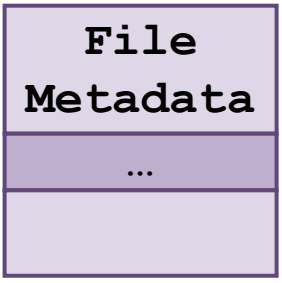
I-node Table



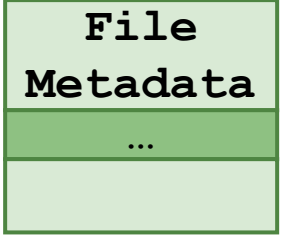
I-Node



I-Node

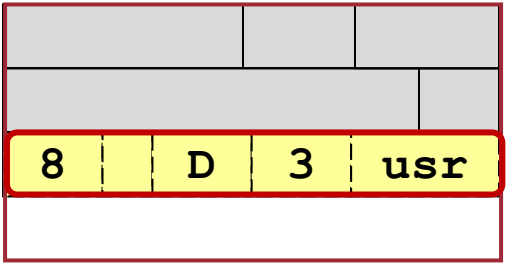


I-Node

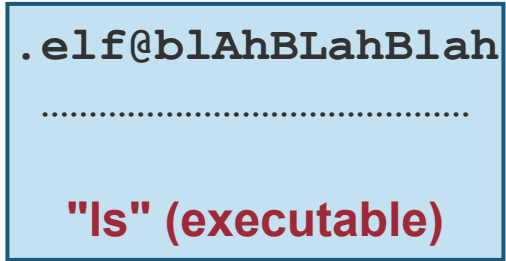


I-Node

`/usr/bin/ls`

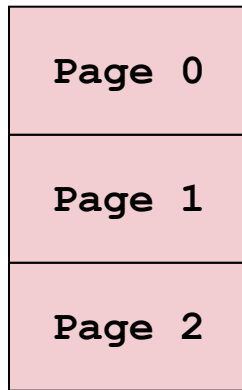


Disk Block X



Disk Block Z

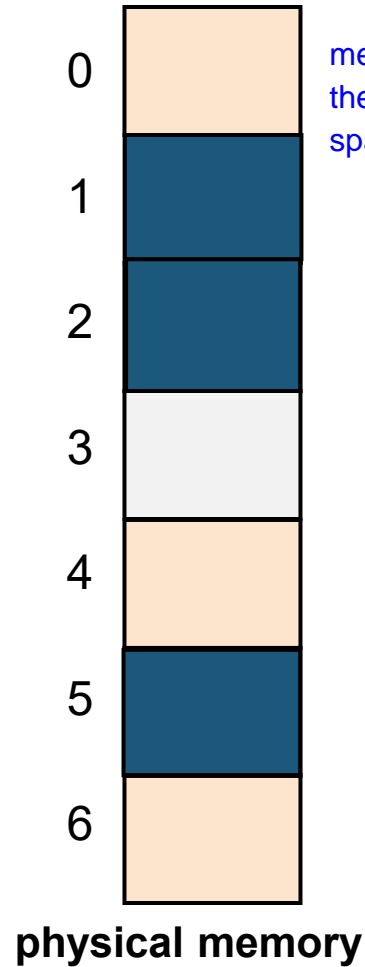
# Memory Content Replaced



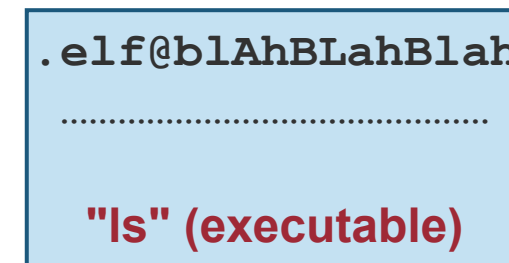
**Memory  
Space**

0	5
1	2
2	1

page table

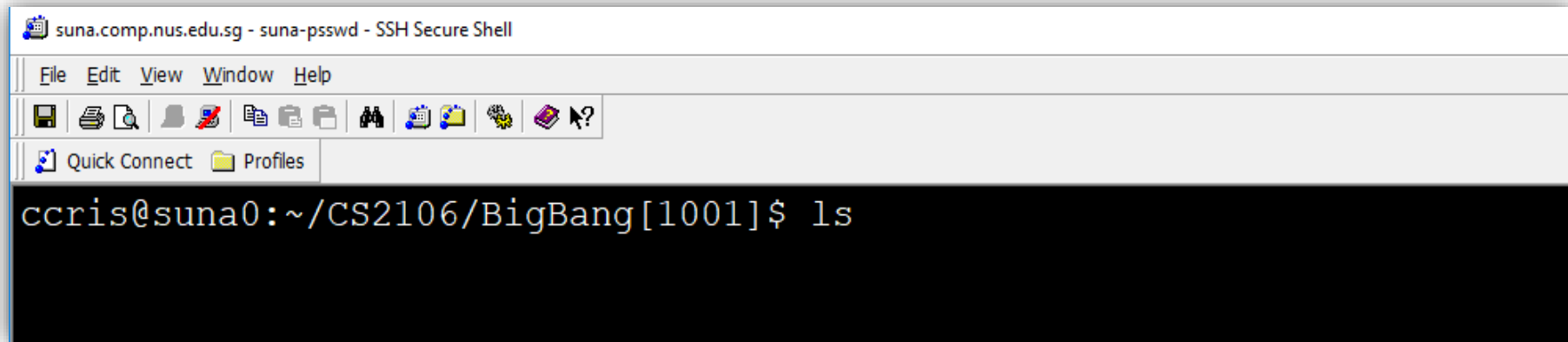


means now the text section of the child process will be replaced with the executable text - thus the frames which belong to the memory space of the child will be rewritten with `ls`



Disk Block Z

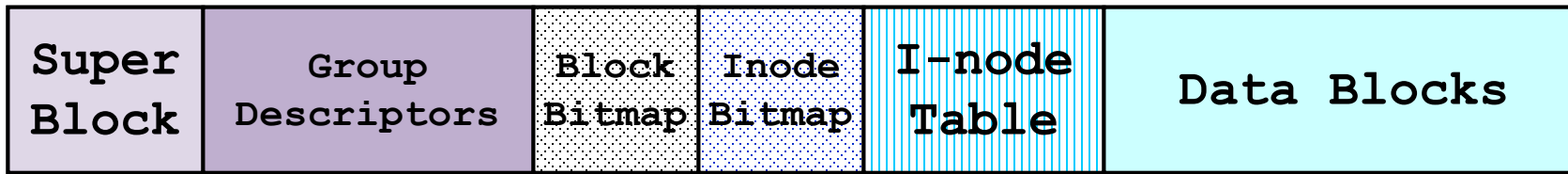
Child is now "**ls**", what next?



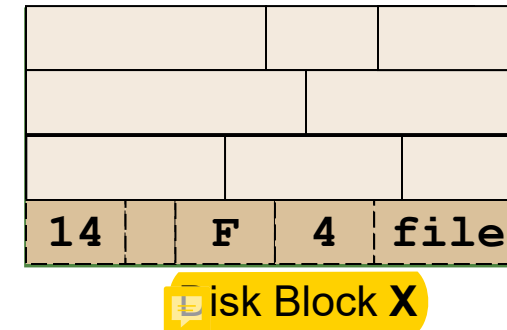
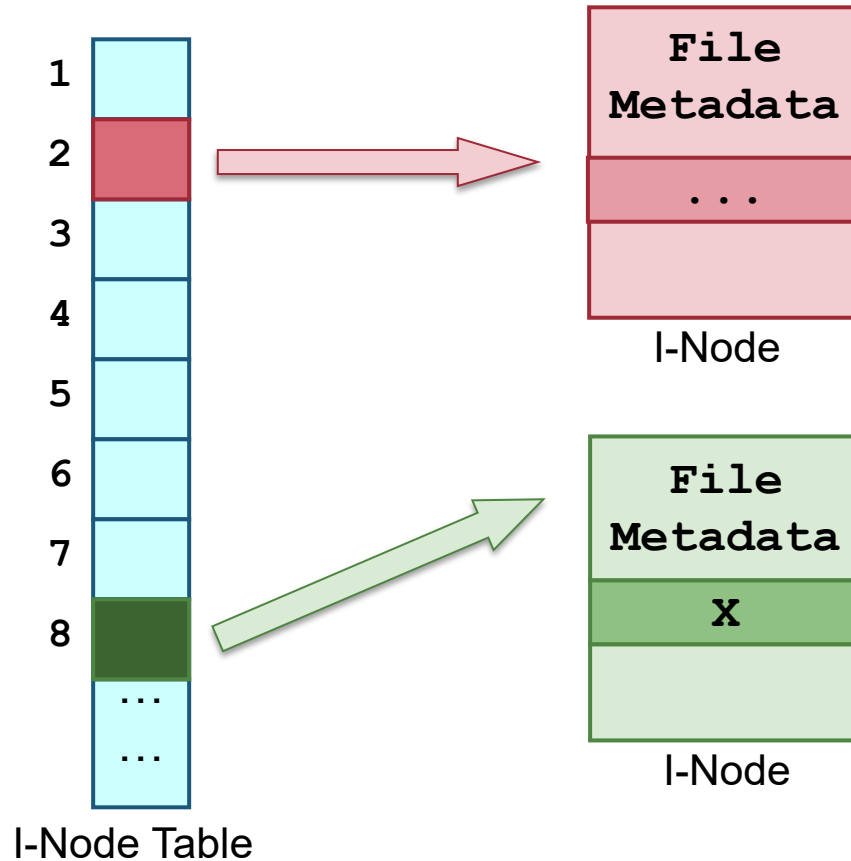
The screenshot shows a terminal window titled 'suna.comp.nus.edu.sg - suna-psswd - SSH Secure Shell'. The window has a menu bar with 'File', 'Edit', 'View', 'Window', and 'Help'. Below the menu bar is a toolbar with various icons. The terminal text shows the prompt 'ccris@suna0:~/CS2106/BigBang[1001]\$' followed by the command 'ls'.

```
suna.comp.nus.edu.sg - suna-psswd - SSH Secure Shell
File Edit View Window Help
[Toolbar icons]
Quick Connect Profiles
ccris@suna0:~/CS2106/BigBang[1001]$ ls
```

Listing **/.../.../BigBang**

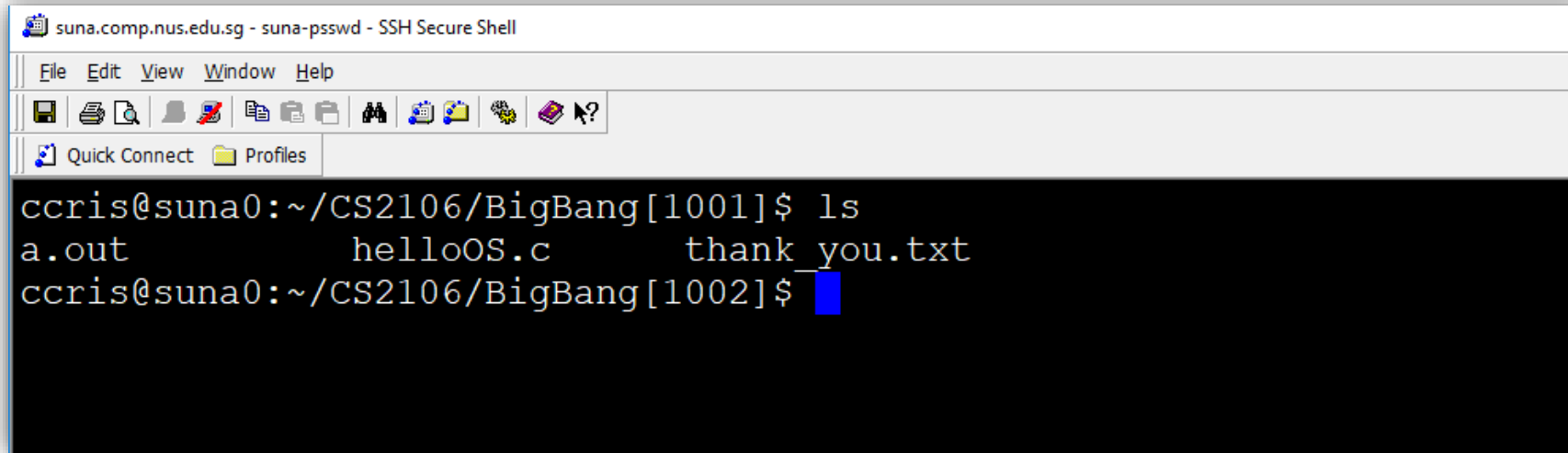


Listing **/.../.../BigBang**



ls -l

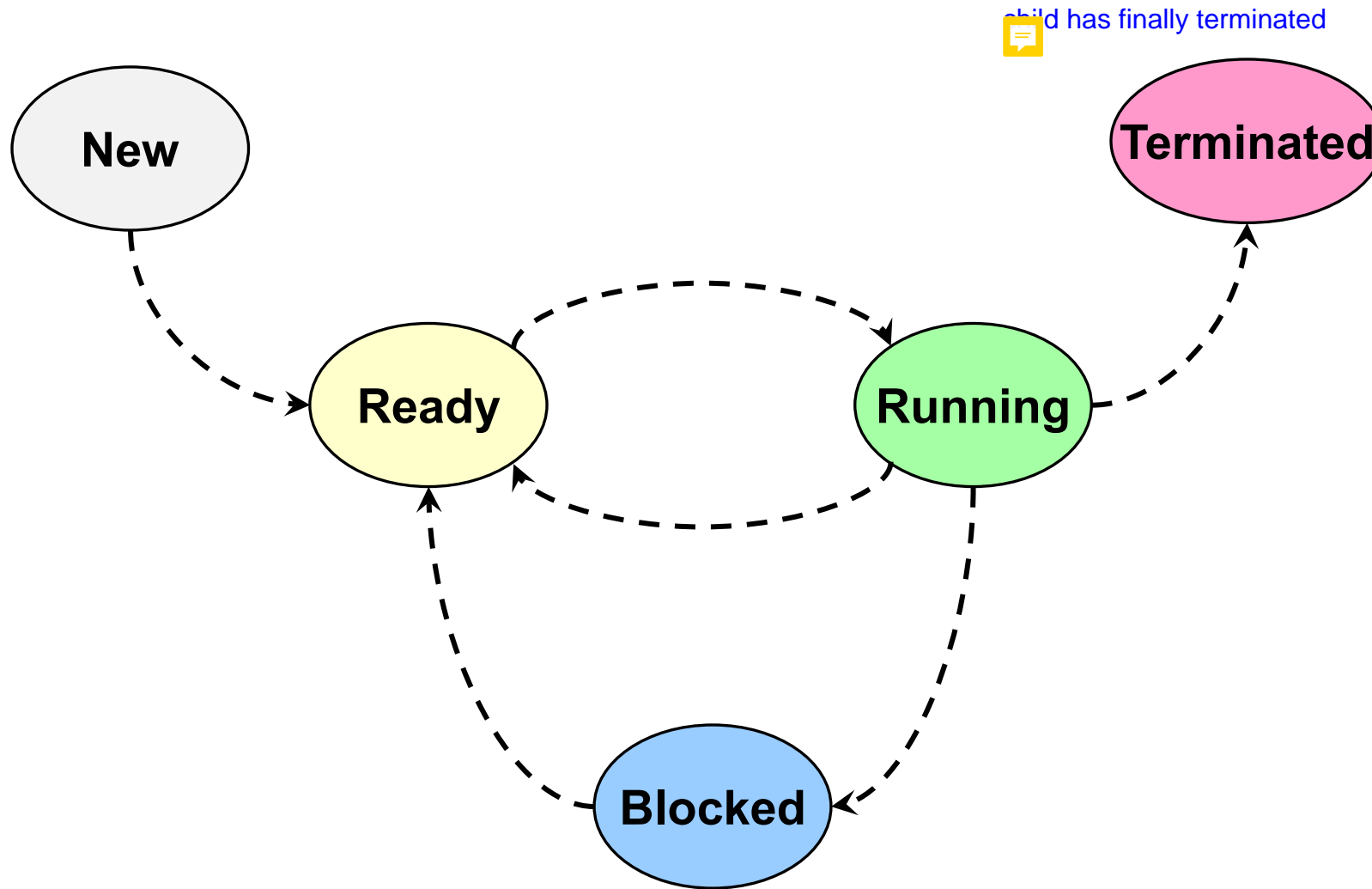
"ls" prints the directory content



The screenshot shows a terminal window titled "suna.comp.nus.edu.sg - suna-psswd - SSH Secure Shell". The window has a menu bar with "File", "Edit", "View", "Window", and "Help". Below the menu bar is a toolbar with various icons. The terminal content shows the user "ccris" at host "suna0" in the directory "~/CS2106/BigBang". The user enters the command "ls" and the output is displayed on the next line: "a.out", "helloOS.c", and "thank\_you.txt". The prompt changes to "[1002]" after the command is executed.

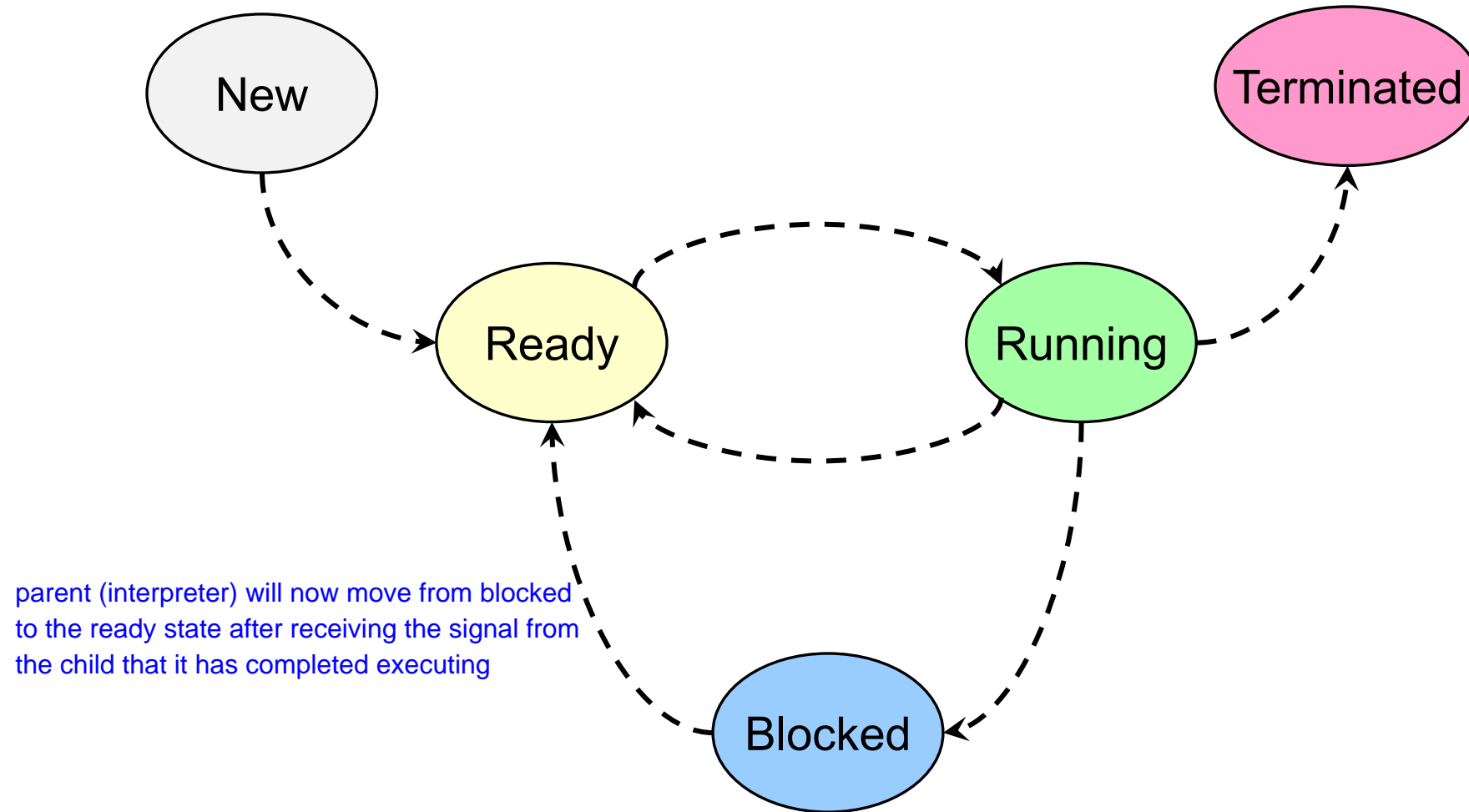
```
suna.comp.nus.edu.sg - suna-psswd - SSH Secure Shell
File Edit View Window Help
a.out helloOS.c thank_you.txt
ccris@suna0:~/CS2106/BigBang[1001]$ ls
a.out helloOS.c thank_you.txt
ccris@suna0:~/CS2106/BigBang[1002]$
```

# Child **exits**





# The interpreter **hears about it**....



# Typical steps for **Shell Interpreter**

UserCmd ← Read from keyboard



**fork()**

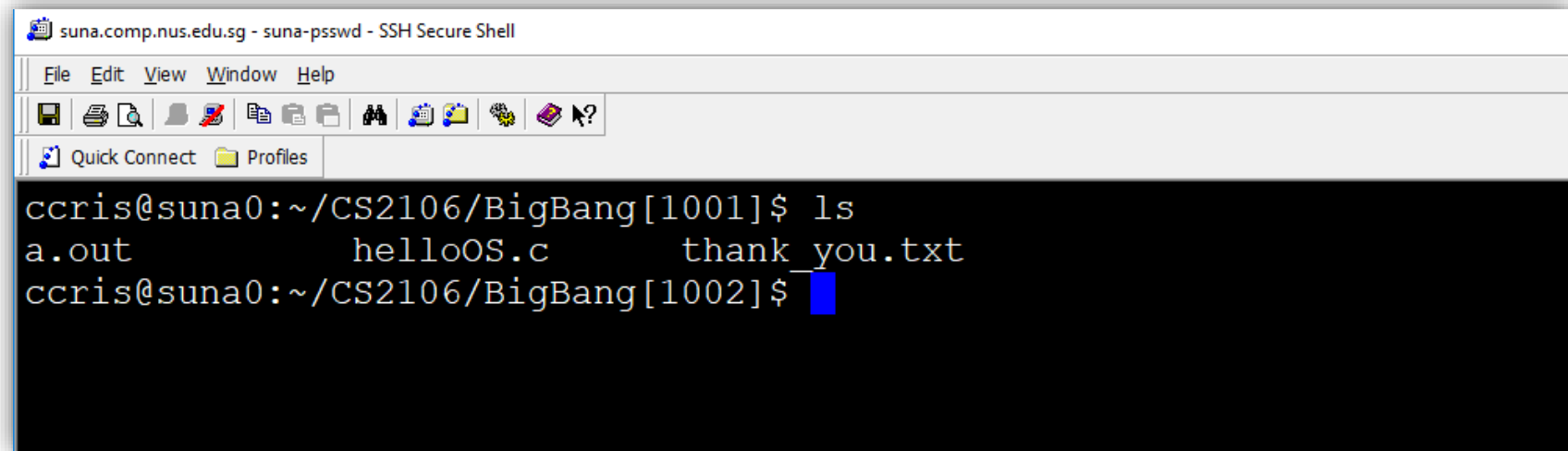
if I am the parent (i.e. the shell)

wait ( child to finish )

**else**

**exec( UserCmd )**

# Woohoo!!



The image shows a screenshot of an SSH terminal window. The title bar reads "suna.comp.nus.edu.sg - suna-psswd - SSH Secure Shell". The menu bar includes "File", "Edit", "View", "Window", and "Help". The toolbar contains icons for file operations like save, print, and copy. Below the toolbar, there are tabs for "Quick Connect" and "Profiles". The terminal area has a black background with white text. The prompt is "ccris@suna0:~/CS2106/BigBang[1001]". The user has entered the command "ls", and the output is "a.out", "helloOS.c", and "thank\_you.txt". The prompt has changed to "[1002]" and there is a blue cursor at the end of the line.

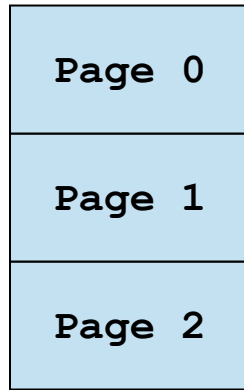
```
suna.comp.nus.edu.sg - suna-psswd - SSH Secure Shell
File Edit View Window Help
[Icons]
Quick Connect Profiles
ccris@suna0:~/CS2106/BigBang[1001]$ ls
a.out          helloOS.c      thank_you.txt
ccris@suna0:~/CS2106/BigBang[1002]$
```

---

**WE SHOULD SHARE!**

---

# How to **Share Memory**?

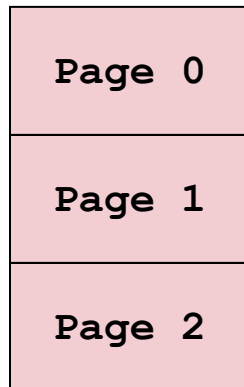


**Memory  
Space**

0	4
1	6
2	0

page table

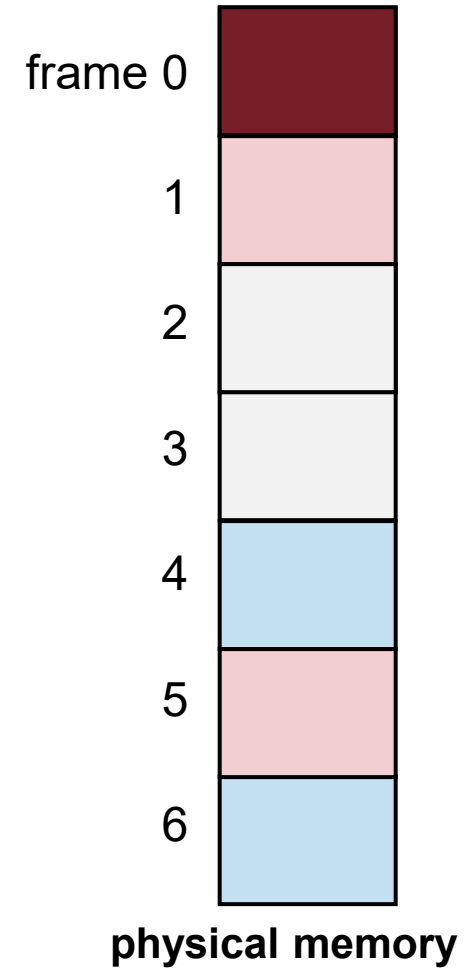
when different processes point to the same frame



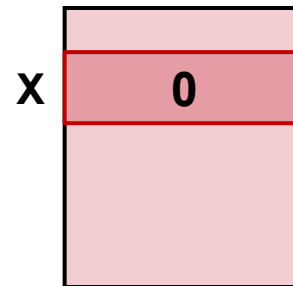
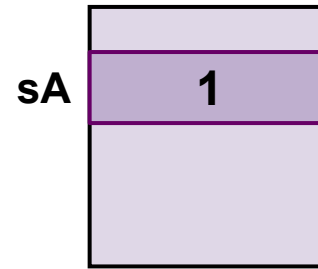
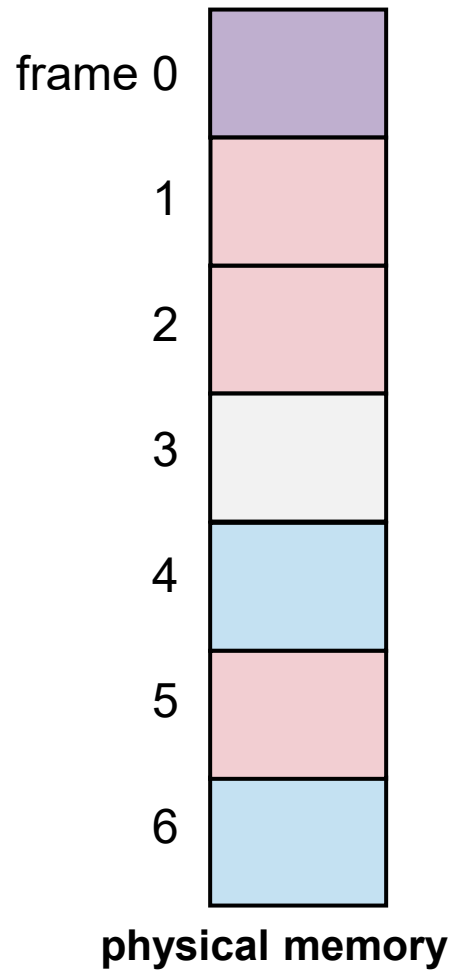
**Memory  
Space**

0	5
1	0
2	1

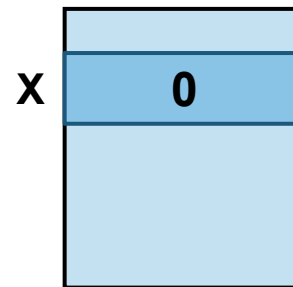
page table



(  $X = X + 1;$        $sA = sA + 1;$  )



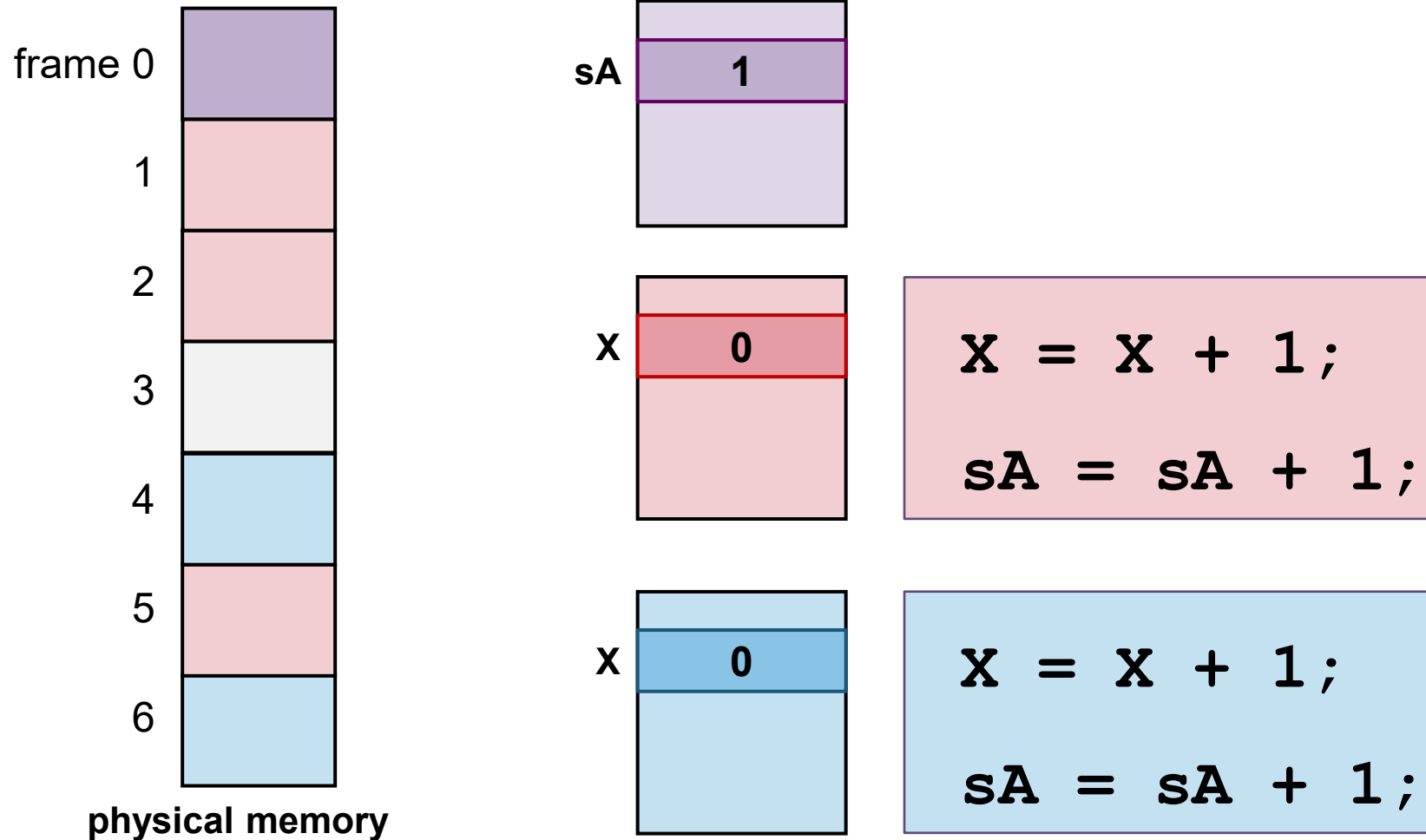
$X = X + 1;$   
 $sA = sA + 1;$



$X = X + 1;$   
 $sA = sA + 1;$

# Semaphore to the rescue!

semaphores create a critical section to ensure that access to a shared memory is done in a sequence



---

# Concurrency

- Race conditions
  - Critical Section
  - Semaphore
  - Classical synchronization problems
-



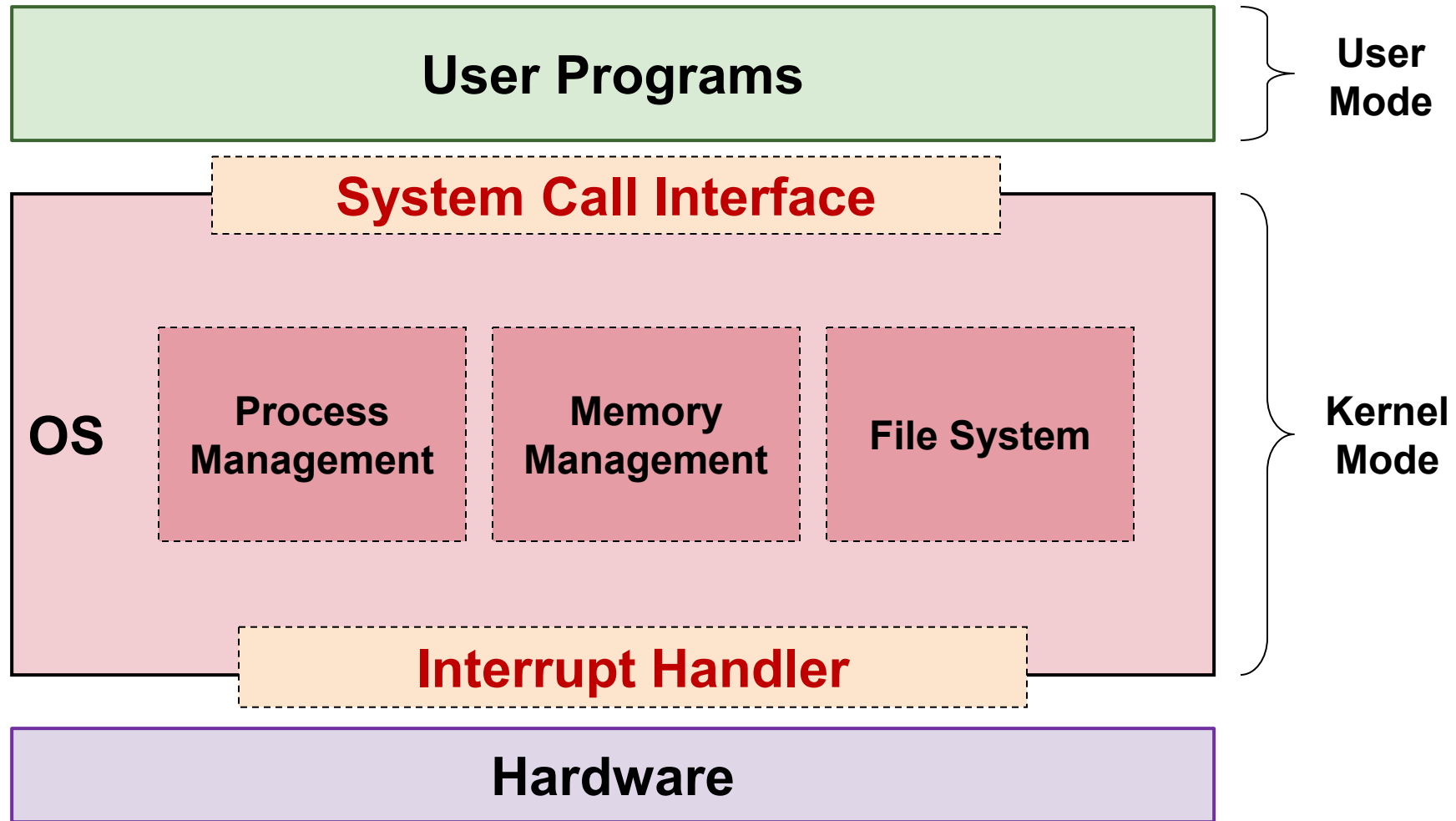
---

Phew..... "quick summary" now

**WHAT HAVE WE LEARNED?**

---

# Operating System



---

**WHAT ELSE  
HAVE WE LEARNED?**

---

---

## Side **Benefits**....

- Design of **complex system**
  - **Abstraction** and **Interface**
  - **Resource Management**
  - Performance Trade Off ( **time** vs **space** )
-

---

# What's **next**?

- **System Security**
  - **Parallel Computing/ Concurrent Programming**
  - **Computer Architecture**
  - **Compilers**
-

---

OH... THE **EXAM** 😊

---

---

# The plan...

- Like the midterm
    - ❑ F2F in MPSH1 an MPSH2
    - ❑ Open book with printed materials
  - Backup in place:
    - ❑ LumiNUS quiz
    - ❑ Zoom proctoring
    - ❑ Record your screen
    - ❑ Refer to PDF materials
  - Email us early to book a consultation slot
-

# Important to **know**

- **Rough percentage of coverage**
  - Lecture 1 to Lecture 5 = **~25%**
  - Lecture 6 to Lecture 11 = **~75%**
- **MCQ questions**
- **Short questions**
  - Write short answers
- **Open book**



---

It's Over!

---

Goodbye! Say Hi if you see us in school!