Recitation 04

Midterm Review!

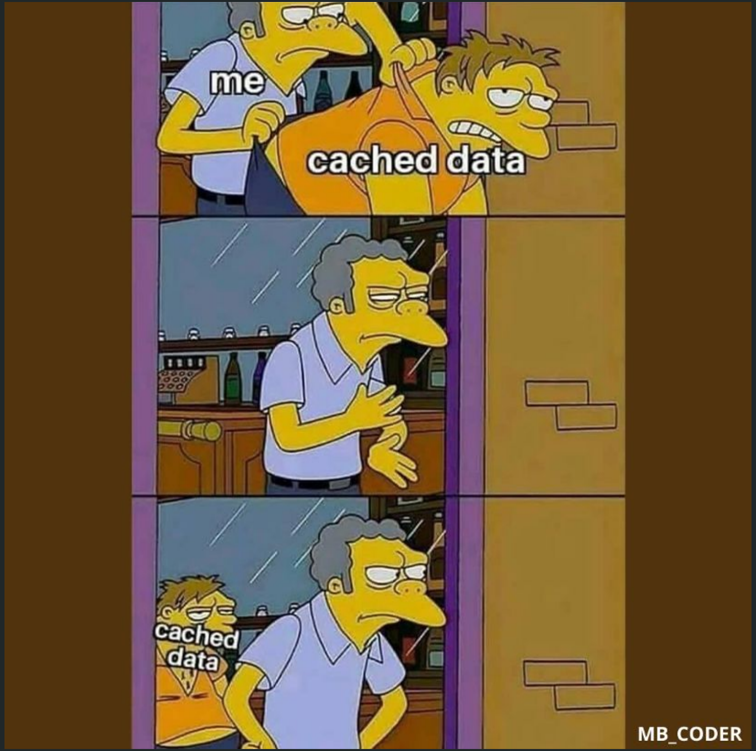


Table of Contents

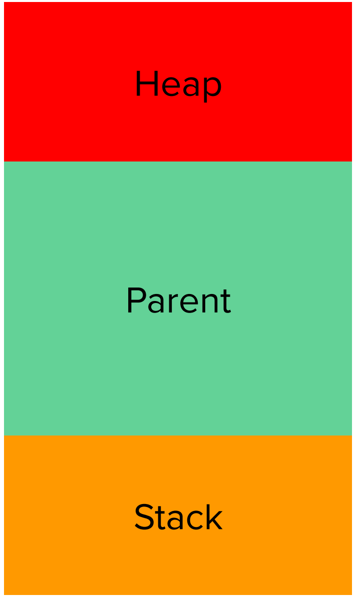
1. Brief overview of all topics
2. Some thinking questions
3. Any questions on Midterm??
4. Open OH for remaining time

List of Midterm Topics

1. Processes
2. Signals
3. Terminal Control
4. File System
5. Cache, Locality and Buffering

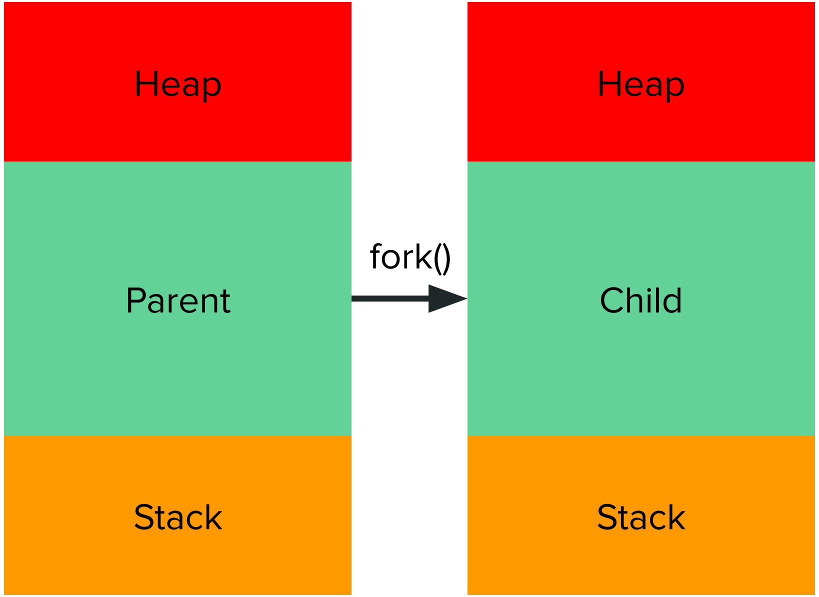
`shredder> echo hi`

Shredder - fork(), exec(), wait()



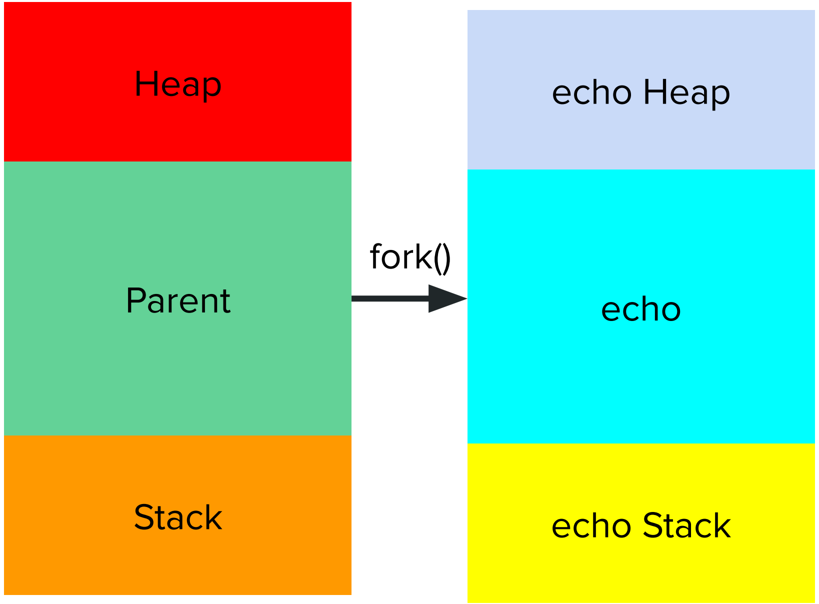
shredder> echo hi

Shredder - fork(), exec(), wait()



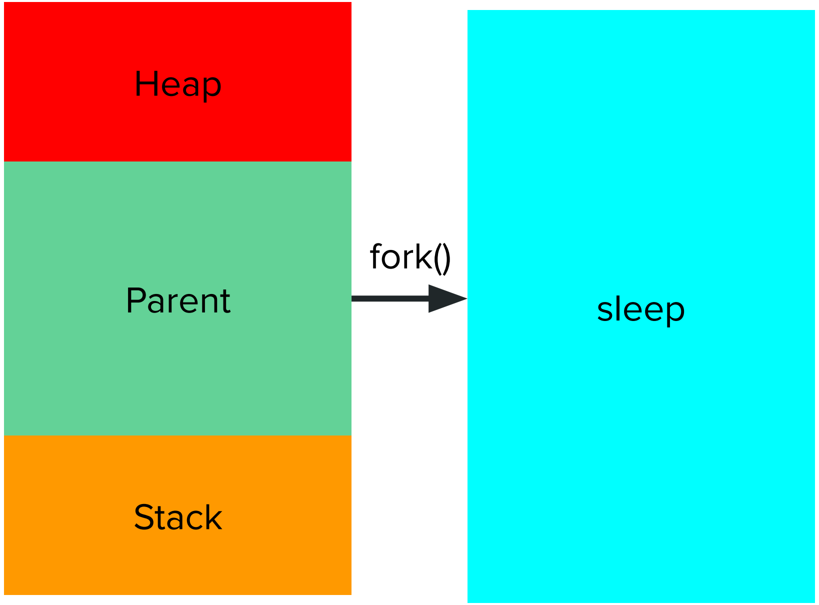
Shredder - fork(), exec(), wait()

```
shredder> echo hi
hi
```



wait(pid);

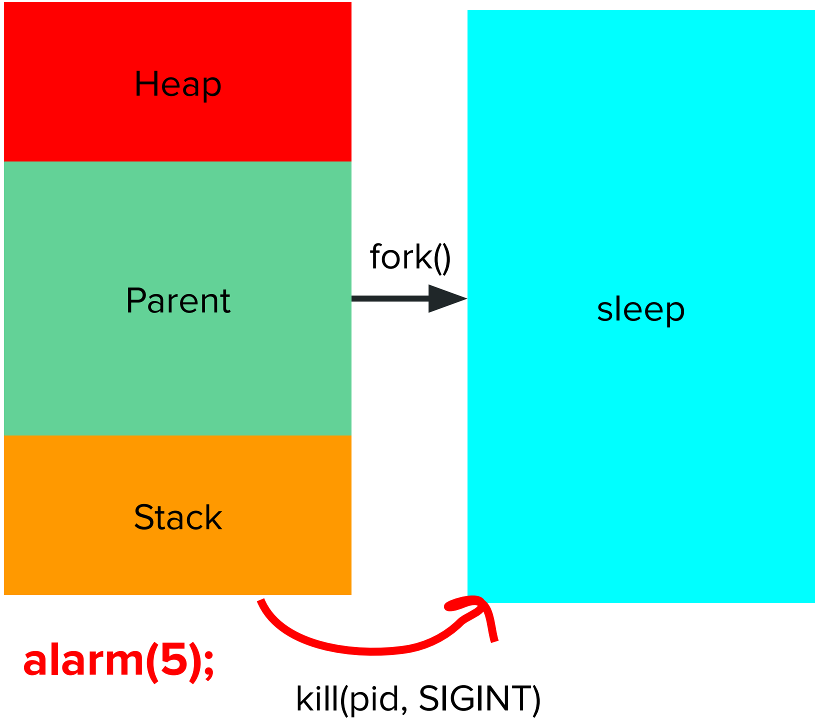
Shredder - alarm(), signal



alarm(5);

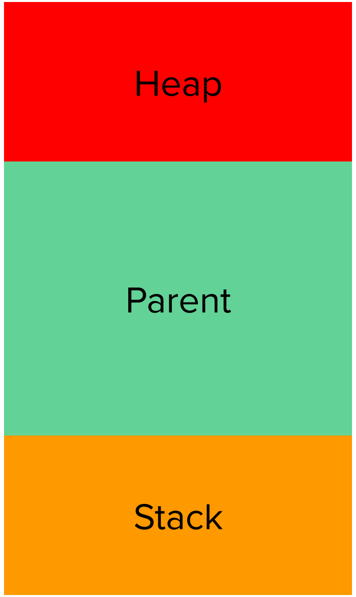
```
shredder> echo hi
hi
shredder> sleep 10
```

Shredder - alarm(), signal




```
shredder> echo hi
hi
shredder> sleep 10
```

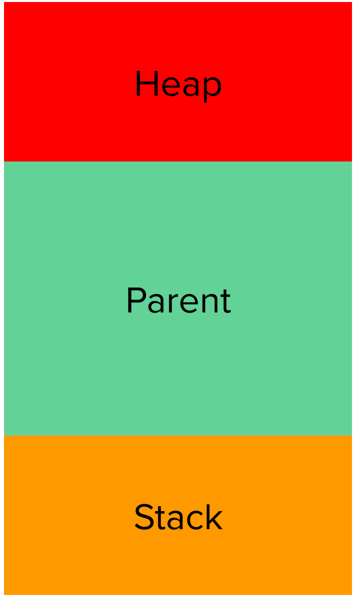
Shredder - alarm(), signal



alarm(5);
wait();

```
shredder> echo hi
hi
shredder> sleep 10
BWAHAHA...
```

Shredder - alarm(), signal

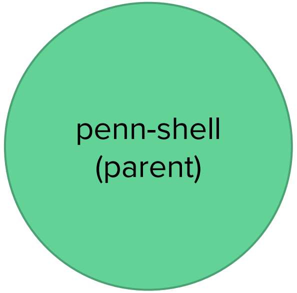


alarm(5);
wait();

```
shredder> echo hi
hi
shredder> sleep 10
BWAHAHA...
shredder>
```

Shell - pipe, redirection

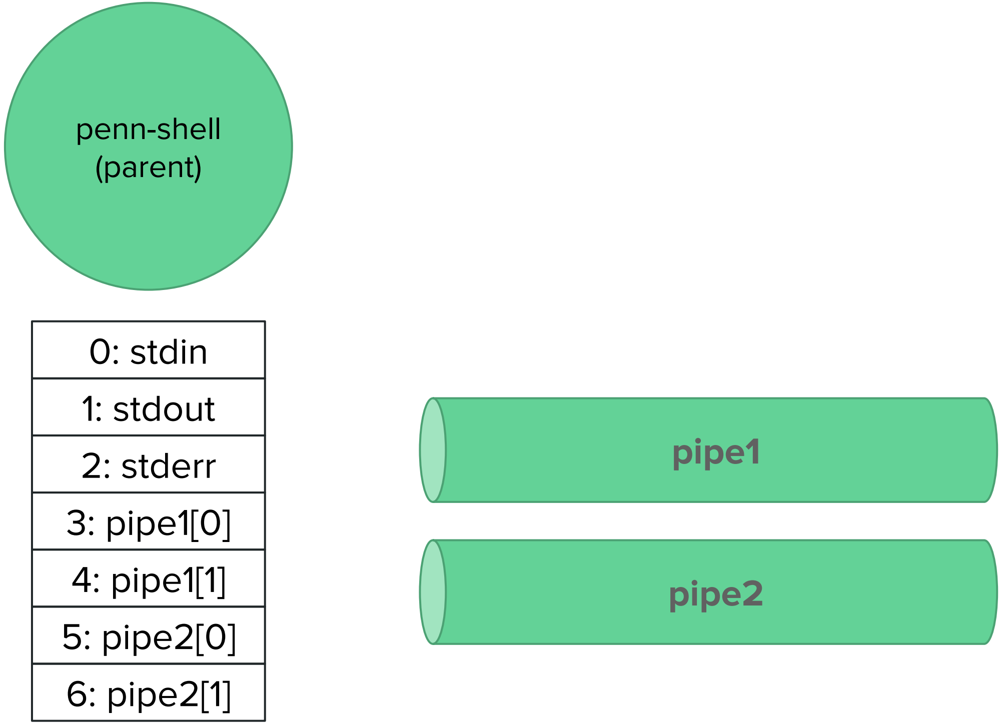
```
cat < file.txt | grep comrade | wc -l -c  > out.txt
```



0: stdin
1: stdout
2: stderr
3: pipe1[0]
4: pipe1[1]
5: pipe2[0]
6: pipe2[1]

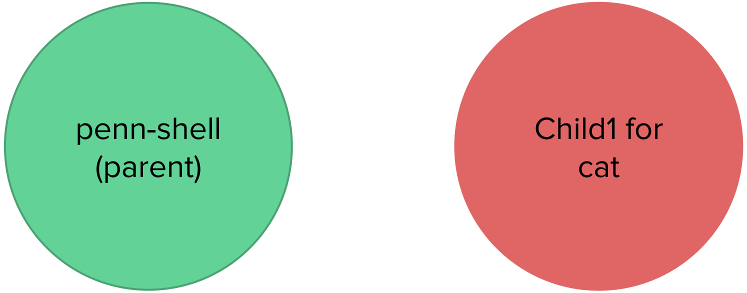
Shell - pipe, redirection

```
cat < file.txt | grep comrade | wc -l -c > out.txt
```



Shell - pipe, redirection

`cat < file.txt | grep comrade | wc -l -c > out.txt`



penn-shell (parent)
0: stdin
1: stdout
2: stderr
3: pipe1[0]
4: pipe1[1]
5: pipe2[0]
6: pipe2[1]

Child1 for cat
0: stdin
1: stdout
2: stderr
3: pipe1[0]
4: pipe1[1]
5: pipe2[0]
6: pipe2[1]

Shell - pipe, redirection

```
cat < file.txt | grep comrade | wc -l -c > out.txt
```

penn-shell (parent)	Child 1 for cat	Child 2 for grep
0: stdin	0: stdin	0: stdin
1: stdout	1: stdout	1: stdout
2: stderr	2: stderr	2: stderr
3: pipe1[0]	3: pipe1[0]	3: pipe1[0]
4: pipe1[1]	4: pipe1[1]	4: pipe1[1]
5: pipe2[0]	5: pipe2[0]	5: pipe2[0]
6: pipe2[1]	6: pipe2[1]	6: pipe2[1]

Shell - pipe, redirection

`cat < file.txt | grep comrade | wc -l -c > out.txt`

penn-shell (parent)	Child 1 for cat	Child 2 for grep	Child 3 for wc
0: stdin	0: stdin	0: stdin	0: stdin
1: stdout	1: stdout	1: stdout	1: stdout
2: stderr	2: stderr	2: stderr	2: stderr
3: pipe1[0]	3: pipe1[0]	3: pipe1[0]	3: pipe1[0]
4: pipe1[1]	4: pipe1[1]	4: pipe1[1]	4: pipe1[1]
5: pipe2[0]	5: pipe2[0]	5: pipe2[0]	5: pipe2[0]
6: pipe2[1]	6: pipe2[1]	6: pipe2[1]	6: pipe2[1]

"Redirect if needed!"

Shell - pipe, redirection

```
cat < file.txt | grep comrade | wc -l -c > out.txt
```

penn-shell (parent)	Child 1 for cat	Child 2 for grep	Child 3 for wc
0: stdin	**0: stdin**	0: stdin	0: stdin
1: stdout	1: stdout	1: stdout	**1: stdout**
2: stderr	2: stderr	2: stderr	2: stderr
3: pipe1[0]	3: pipe1[0]	3: pipe1[0]	3: pipe1[0]
4: pipe1[1]	4: pipe1[1]	4: pipe1[1]	4: pipe1[1]
5: pipe2[0]	5: pipe2[0]	5: pipe2[0]	5: pipe2[0]
6: pipe2[1]	6: pipe2[1]	6: pipe2[1]	6: pipe2[1]

"Redirect if needed!"

Shell - pipe, redirection

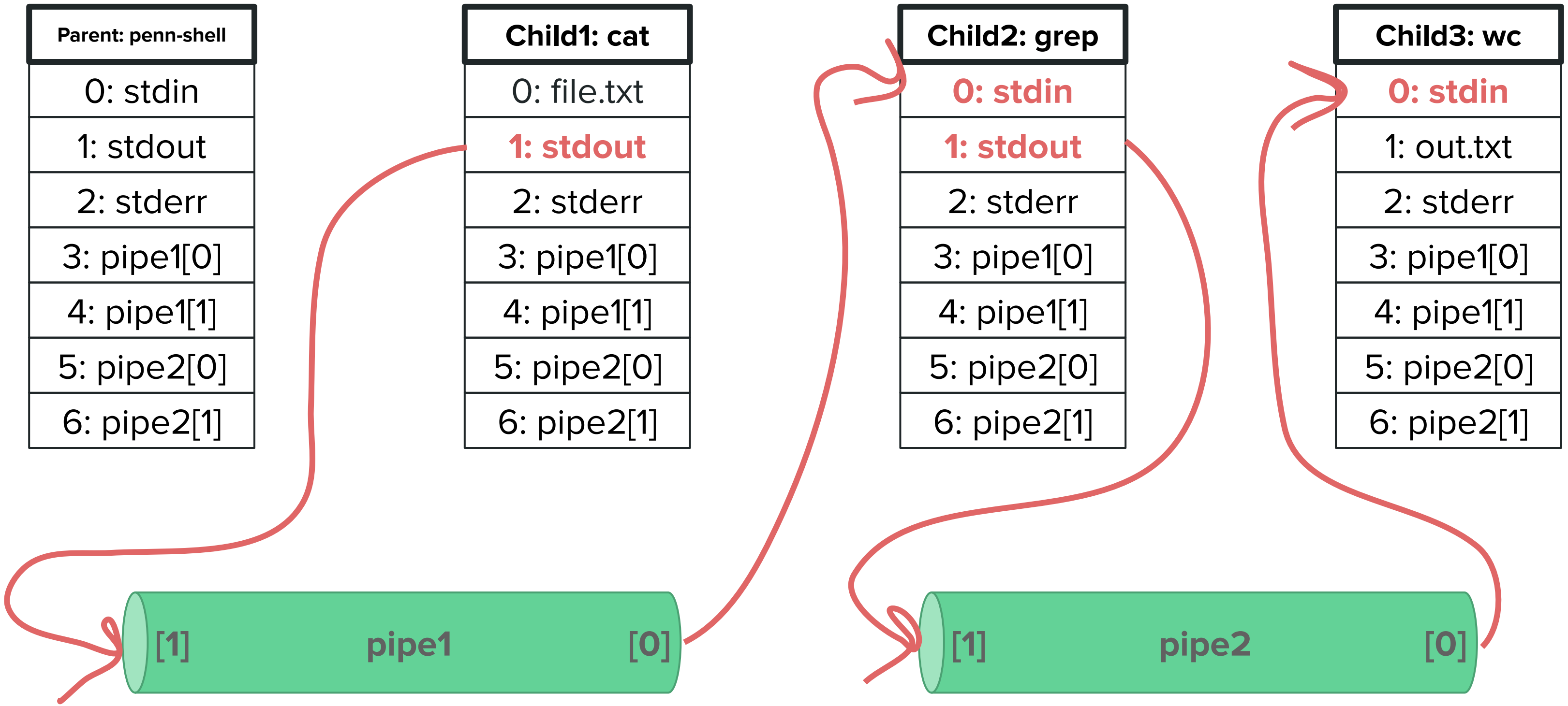
```
cat < file.txt | grep comrade | wc -l -c > out.txt
```

penn-shell (parent)	Child 1 for cat	Child 2 for grep	Child 3 for wc
0: stdin	**0: file.txt**	0: stdin	0: stdin
1: stdout	1: stdout	1: stdout	**1: out.txt**
2: stderr	2: stderr	2: stderr	2: stderr
3: pipe1[0]	3: pipe1[0]	3: pipe1[0]	3: pipe1[0]
4: pipe1[1]	4: pipe1[1]	4: pipe1[1]	4: pipe1[1]
5: pipe2[0]	5: pipe2[0]	5: pipe2[0]	5: pipe2[0]
6: pipe2[1]	6: pipe2[1]	6: pipe2[1]	6: pipe2[1]

“Connect children with pipes!”

Shell - pipe, redirection

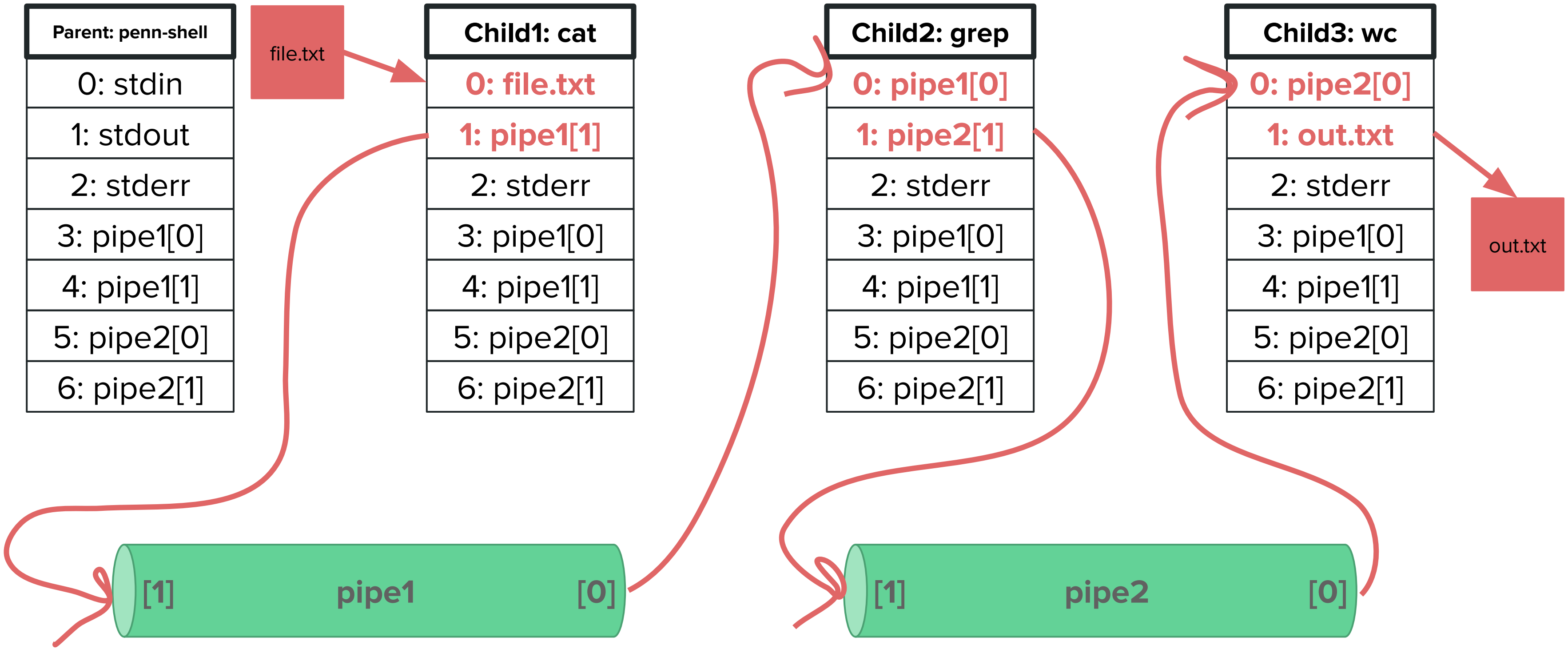
```
cat < file.txt | grep comrade | wc -l -c > out.txt
```



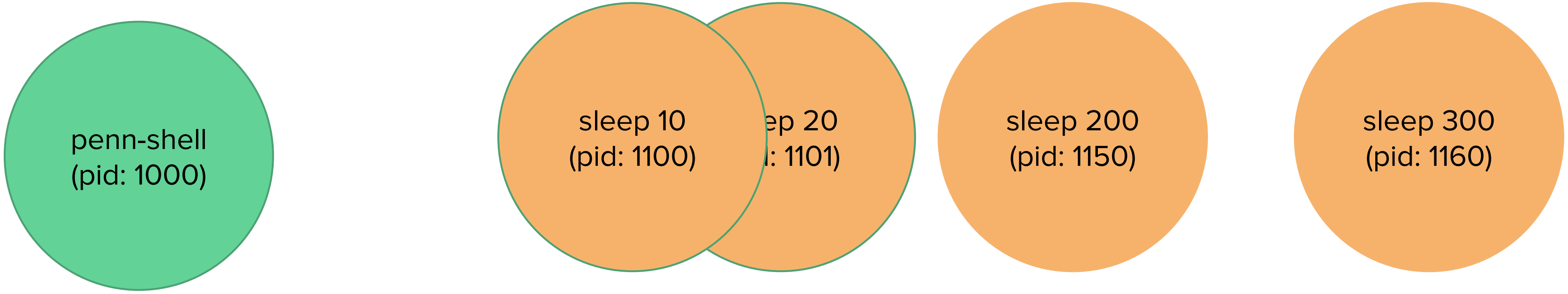
Content from "file.txt" will travel through pipes from Child 1 to 3
Each child will execute according to data they read from pipe/file

Shell - pipe, redirection

```
cat < file.txt | grep comrade | wc -l -c > out.txt
```



Shell - process group, terminal control



status: running
terminal control: yes

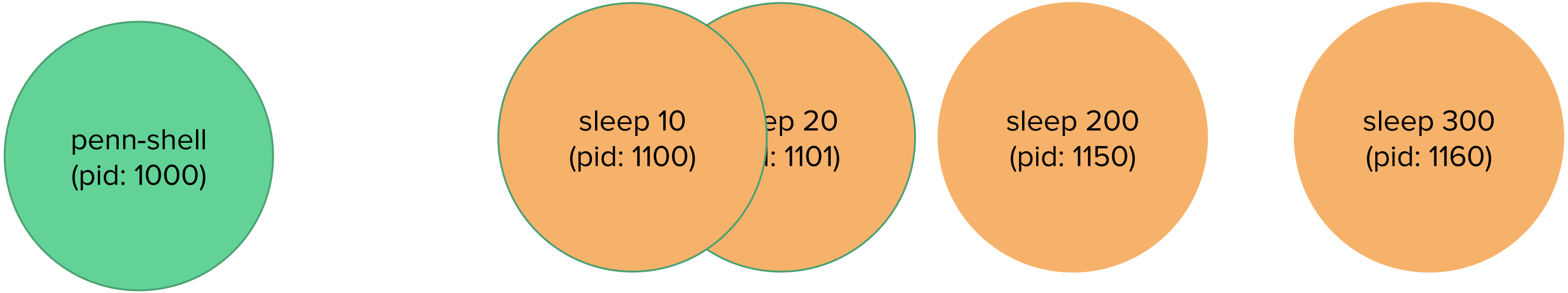
status: running
terminal control: no
ppid: 1000
pgid: 1100

status: running
terminal control: no
ppid: 1000
pgid: 1150

status: running
terminal control: no
ppid: 1000
pgid: 1160

```
shell> sleep 10 | sleep 20 &
shell> sleep 200 &
shell> sleep 300 &
shell>
```

Shell - process group, terminal control



status: running
terminal control: **no**

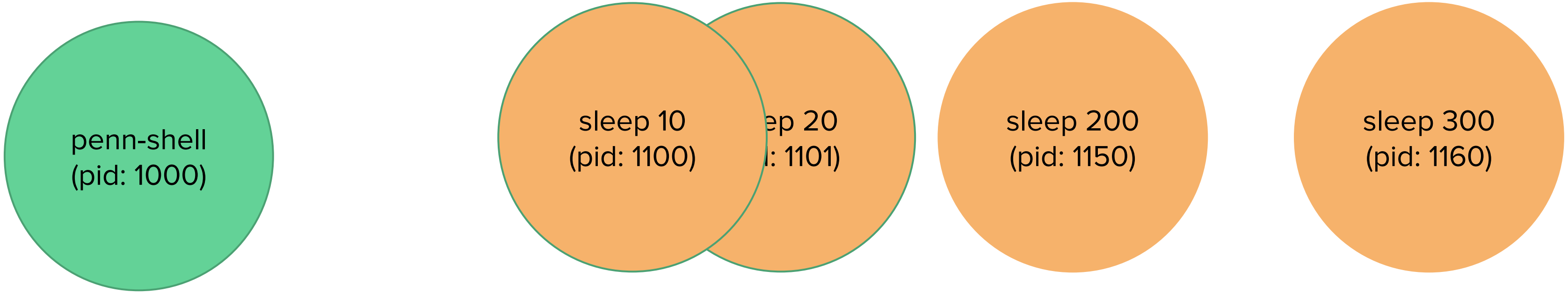
status: running
terminal control: no
ppid: 1000
pgid: 1100

status: running
terminal control: no
ppid: 1000
pgid: 1150

status: running
terminal control: **yes**
ppid: 1000
pgid: 1160

```
shell> sleep 10 | sleep 20 &
shell> sleep 200 &
shell> sleep 300 &
shell> fg
    (running sleep 300)
```

Shell - process group, terminal control



status: running
terminal control: no

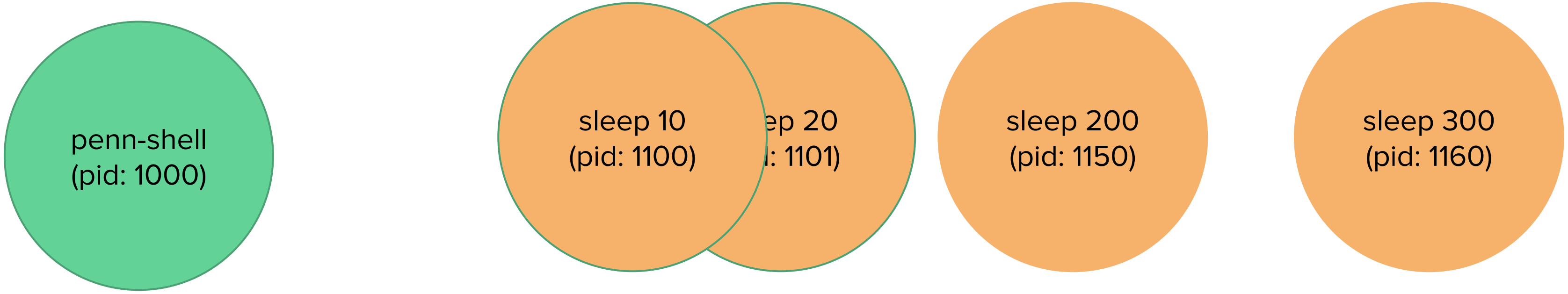
status: terminated (zombie)
terminal control: no
ppid: 1000
pgid: 1100

status: running
terminal control: no
ppid: 1000
pgid: 1150

status: running
terminal control: yes
ppid: 1000
pgid: 1160

```
shell> sleep 10 | sleep 20 &
shell> sleep 200 &
shell> sleep 300 &
shell> fg
     (running sleep 300)
     (sleep 10 | sleep 20 finishes in background)
```

Shell - process group, terminal control



penn-shell (pid: 1000)
status: running
terminal control: **yes**

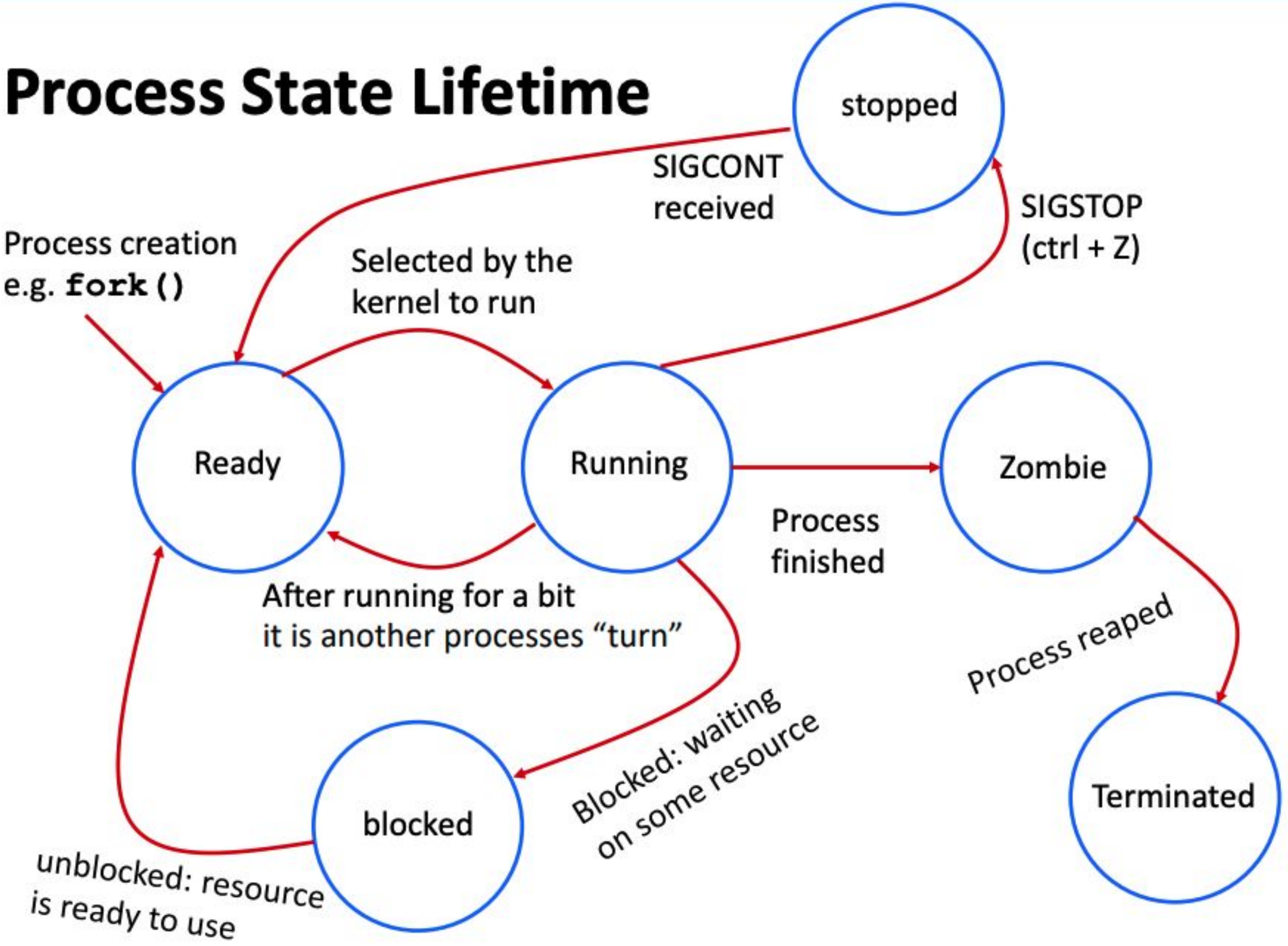
sleep 10 | sleep 20
status: **terminated (zombie)**
terminal control: no
ppid: 1000
pgid: 1100

sleep 200
status: running
terminal control: no
ppid: 1000
pgid: 1150

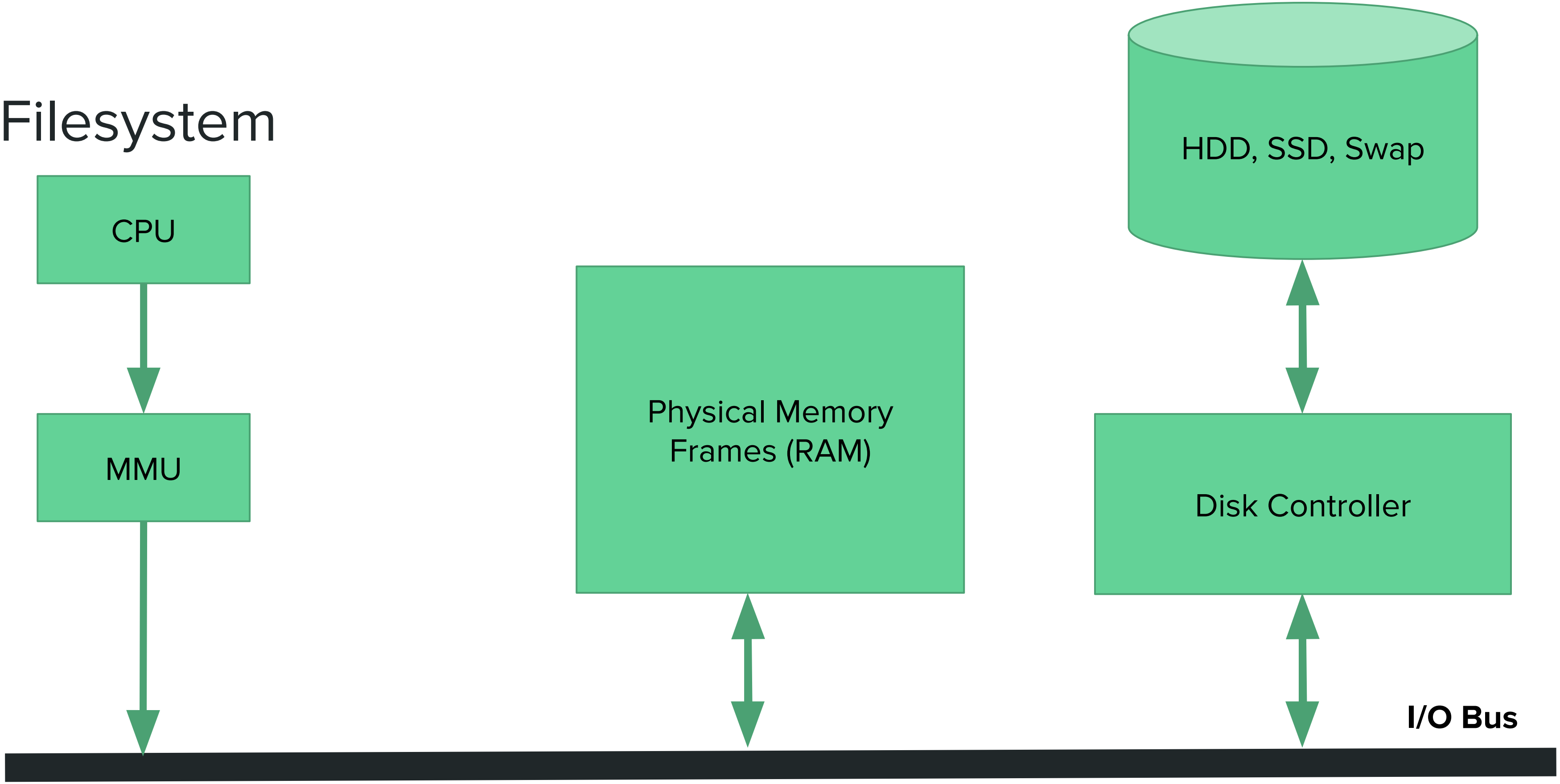
sleep 300
status: **stopped**
terminal control: **no**
ppid: 1000
pgid: 1160

```
shell> sleep 10 | sleep 20 &
shell> sleep 200 &
shell> sleep 300 &
shell> fg
      (running sleep 300)
      (sleep 10 | sleep 20 finishes in background)
^Z                                   (SIGTSTP sent to foreground process, shells re-gains TC)
Finished: sleep 10 | sleep 20        (Any terminated child in background is waited on)
shell>                               (Reprompt)
```

Process State Lifetime



Filesystem


HDD, SSD, Swap
CPU
Physical Memory
Frames (RAM)
MMU
Disk Controller
I/O Bus

Filesystem

- Interfaces, abstractions to organize/access data in disk
- Sequence of bytes ⇒ Blocks of bytes ⇒ Files
- How and where to access?
    - Directory file
    - Bitmap
    - FAT
    - Inodes

Filesystem - Some questions

- What are the pros and cons of each method?
- Is bitmap alone enough for a filesystem API? If not, how can we modify?
- Contiguous allocation vs Linked List allocation
- Internal Fragmentation and External Fragmentation
- Difference between Inodes and FAT?

Buffering

```
int main() {
  for (i is [1,3]) {
    pid = fork();
    if (pid == 0) print("hi");
  }
}
```

How many children?

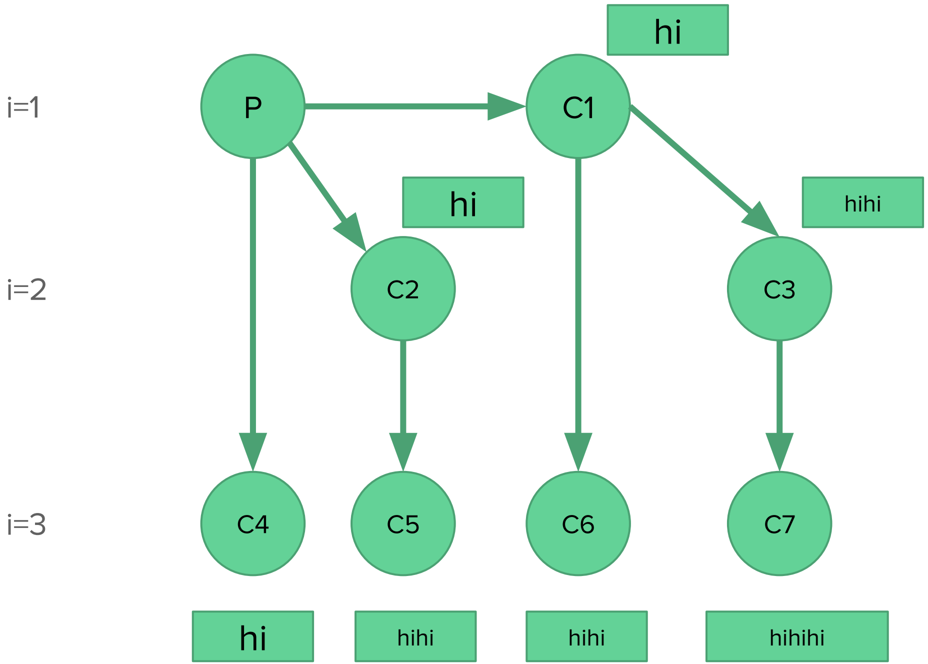
7

How many "hi\n"?

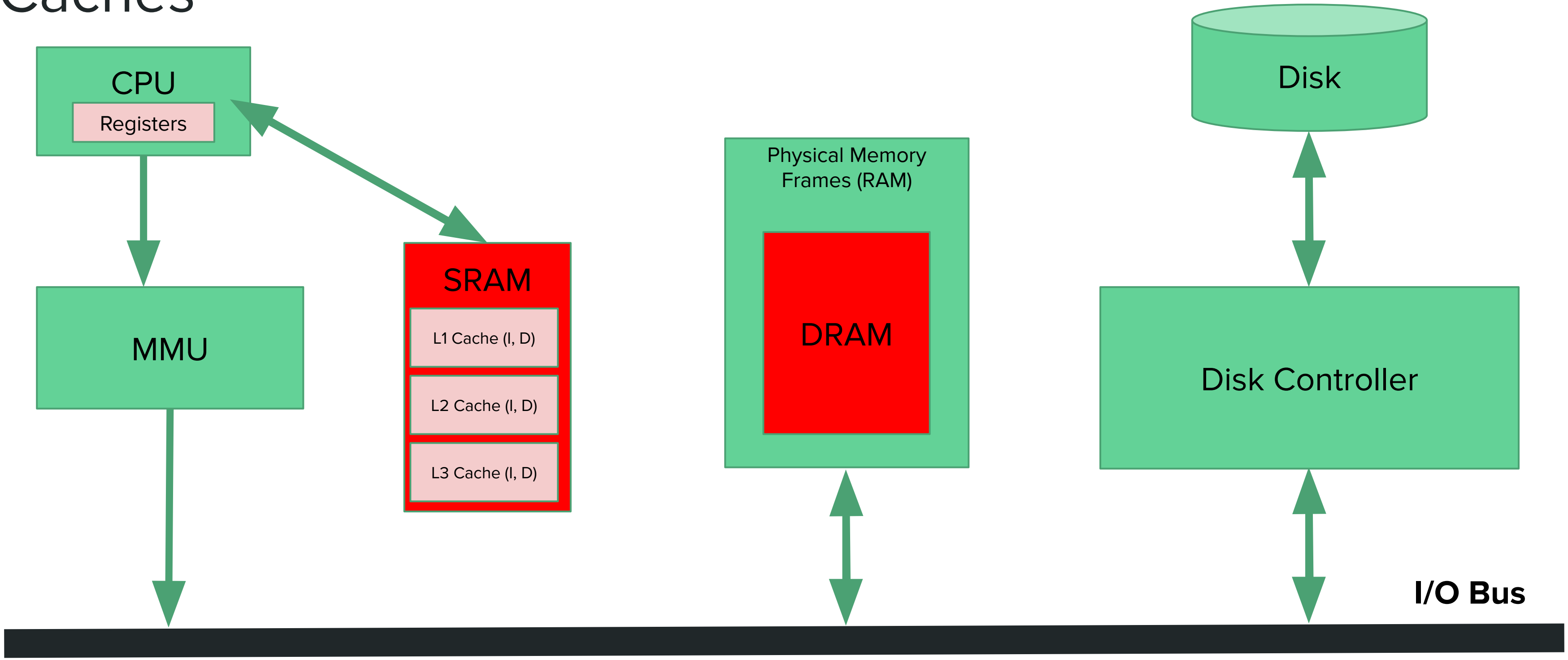
12

How to resolve?

Why do we buffer?



Caches



CPU
Registers
Disk
SRAM
L1 Cache (I, D)
L2 Cache (I, D)
L3 Cache (I, D)
Physical Memory
Frames (RAM)
DRAM
MMU
Disk Controller
I/O Bus

Least Recently Used (LRU)

- Memory is limited
- Which line of memory to evict/replace when we run out of memory?
  - Least Recently Used
- Advantages of LRU
  - Generally good performance, we are evicting a page that is “least” frequently used
  - Reduces number of page faults
- Disadvantages of LRU
  - Quite costly to find the LRU page
- Think of a scenario where LRU may actually hurt performance
  - Sequential Access: If some sequential access pattern forces LRU to evict and re-allocate parts of memory, we will have poor performance
- How else can we design the eviction policy?

Some Thinking Questions 1: What might go wrong here?

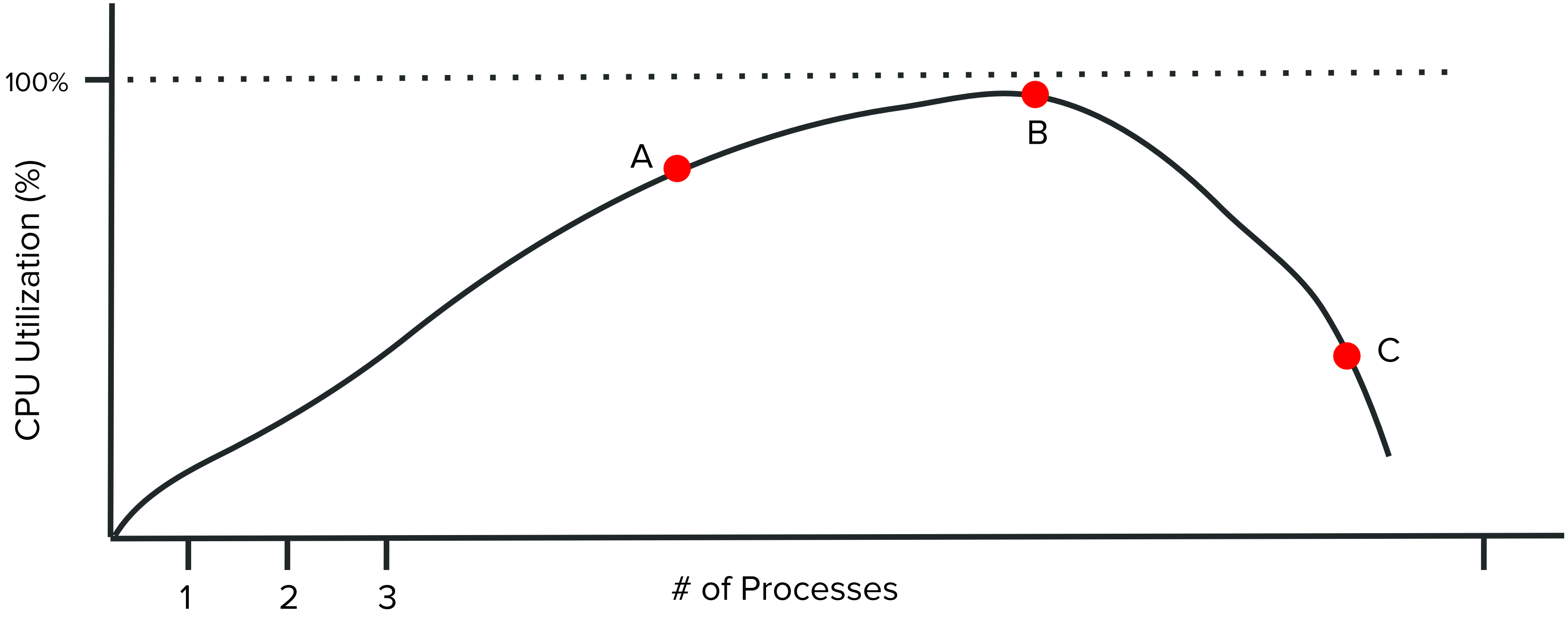
```
1   void poll() {
2         for (all background jobs) {
3               waitpid(-pgid, status, WNOHANG |
4   WUNTRACED);
5               modify_queue();
6         }
7   }
8
9   while (true) {
10        poll();
11        prompt_and_get_command();
12        pipe();
13        for (all children) fork(), setpgid();
14              pipe(), redirect(), exec()
15        if (background) { waitpid(-1, status, WNOHANG); }
16        else { waitpid(-1, status, WUNTRACED); }
    }
```

1. We are waiting for only 1 child process in a job in lines 13, 14
   a. What if “sleep 10 | sleep 100” ?
2. We are waiting for “any child process” waitpid(-1) in line 13
   a. What if there were zombied processes in the background?

Some Thinking Questions 2

Consider this graph of CPU Utilization vs # of Processes Running

What is happening at each point?



A: Context Switching becomes a problem

B: SUM(memory utilization) > RAM, so we start using more SWAP file

C: Thrashing: Most of the memory access causes a page fault, and we use SWAP a lot

Some Thinking Questions 3

Suppose our cache block size is 4096 bytes and we had a data structure that is 128 bytes big. If we had an array of this structure, what principle could we take advantage of when we fetch this array sequentially?

Spatial locality

What if the data struct is 4096 bytes big?

Cache is almost meaningless since only one data struct can fit in it

More Practice Problems

https://www.seas.upenn.edu/~cis3800/24sp/exams/midterm

Any Questions?