Recitation 3

Process Groups &
Terminal Control

Table of Contents

Pipelines

Consider the following pipelined process:

```
# sleep 1 | sleep 20 | sleep 100
```

Q: How long does this process take?

A: 100 seconds

Q: Why?

A: Parallel Execution

```
# ps -o pid,args
    PID COMMAND
   1234 bash
   4100 sleep 1
   4101 sleep 20
   4102 sleep 100
   5300 ps -o pid,args
#
```

But how does the shell keep in track of these *multi*-process, *single* jobs?

Pipelines and Process Groups

```
root@57fb0c278533:~/cis3800# sleep 10 | sleep 10 | sleep 10&
[1] 31879
root@57fb0c278533:~/cis3800# ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    0     1     1     1 pts/0        1 Ss+      0   0:00 bash
  273   293   293   293 pts/1    31938 Ss       0   0:00 /bin/bash
  273 31513 31513 31513 pts/2    31513 Ss+      0   0:00 /bin/bash
  293 31877 31877   293 pts/1    31938 S        0   0:00 sleep 10
  293 31878 31877   293 pts/1    31938 S        0   0:00 sleep 10
  293 31879 31877   293 pts/1    31938 S        0   0:00 sleep 10
  293 31938 31938   293 pts/1    31938 R+       0   0:00 ps j
root@57fb0c278533:~/cis3800#
```

Process Groups

```
root@57fb0c278533:~/cis3800# sleep 10 | sleep 10 | sleep 10&
[1] 31879
root@57fb0c278533:~/cis3800# ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    0     1     1     1 pts/0        1 Ss+      0   0:00 bash
  273   293   293   293 pts/1    31938 Ss       0   0:00 /bin/bash
  273 31513 31513 31513 pts/2    31513 Ss+      0   0:00 /bin/bash
  293 31877 31877   293 pts/1    31938 S        0   0:00 sleep 10
  293 31878 31877   293 pts/1    31938 S        0   0:00 sleep 10
  293 31879 31877   293 pts/1    31938 S        0   0:00 sleep 10
  293 31938 31938   293 pts/1    31938 R+       0   0:00 ps j
root@57fb0c278533:~/cis3800#
```

pgid 293

/bin/bash (293)

pgid 31877

sleep 10 (31877)
sleep 10 (31878)
sleep 10 (31879)

Process Groups

```
root@57fb0c278533:~/cis3800# sleep 10 | sleep 10 | sleep 10&
[1] 31879
root@57fb0c278533:~/cis3800# ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    0     1     1     1 pts/0        1 Ss+      0   0:00 bash
  273   293   293   293 pts/1    31938 Ss       0   0:00 /bin/bash
  273 31513 31513 31513 pts/2    31513 Ss+      0   0:00 /bin/bash
  293 31877 31877   293 pts/1    31938 S        0   0:00 sleep 10
  293 31878 31877   293 pts/1    31938 S        0   0:00 sleep 10
  293 31879 31877   293 pts/1    31938 S        0   0:00 sleep 10
  293 31938 31938   293 pts/1    31938 R+       0   0:00 ps j
root@57fb0c278533:~/cis3800#
```

pgid 293

/bin/bash (293)

pgid **31877**

sleep 10 (**31877**)
sleep 10 (31878)
sleep 10 (31879)

man getpid setpgid

```
GETPID(2)                                Linux Programmer's Manual                                GETPID(2)

NAME
       getpid, getppid - get process identification

SYNOPSIS
       #include <sys/types.h>
       #include <unistd.h>

       pid_t getpid(void);
       pid_t getppid(void);
```

```
SETPGID(2)                               Linux Programmer's Manual                               SETPGID(2)

NAME
       setpgid, getpgid, setpgrp, getpgrp - set/get process group

SYNOPSIS
       #include <sys/types.h>
       #include <unistd.h>

       int setpgid(pid_t pid, pid_t pgid);
       pid_t getpgid(pid_t pid);

       pid_t getpgrp(void);                 /* POSIX.1 version */
       pid_t getpgrp(pid_t pid);            /* BSD version */

       int setpgrp(void);                   /* System V version */
       int setpgrp(pid_t pid, pid_t pgid);  /* BSD version */
```

Terminal Control

- Only one process group has the “baton” which is terminal control
- Terminal controlling function gets signals along with ability to read terminal

Giving and Receiving Terminal Control

- Use tcsetpgrp to give terminal control
- Is there any issues with doing this? (hint: signals)
- Can get the pgid who has terminal control with tcgetpgrp

```
pid_t tcgetpgrp(int fd);
int tcsetpgrp(int fd, pid_t pgrp);
```

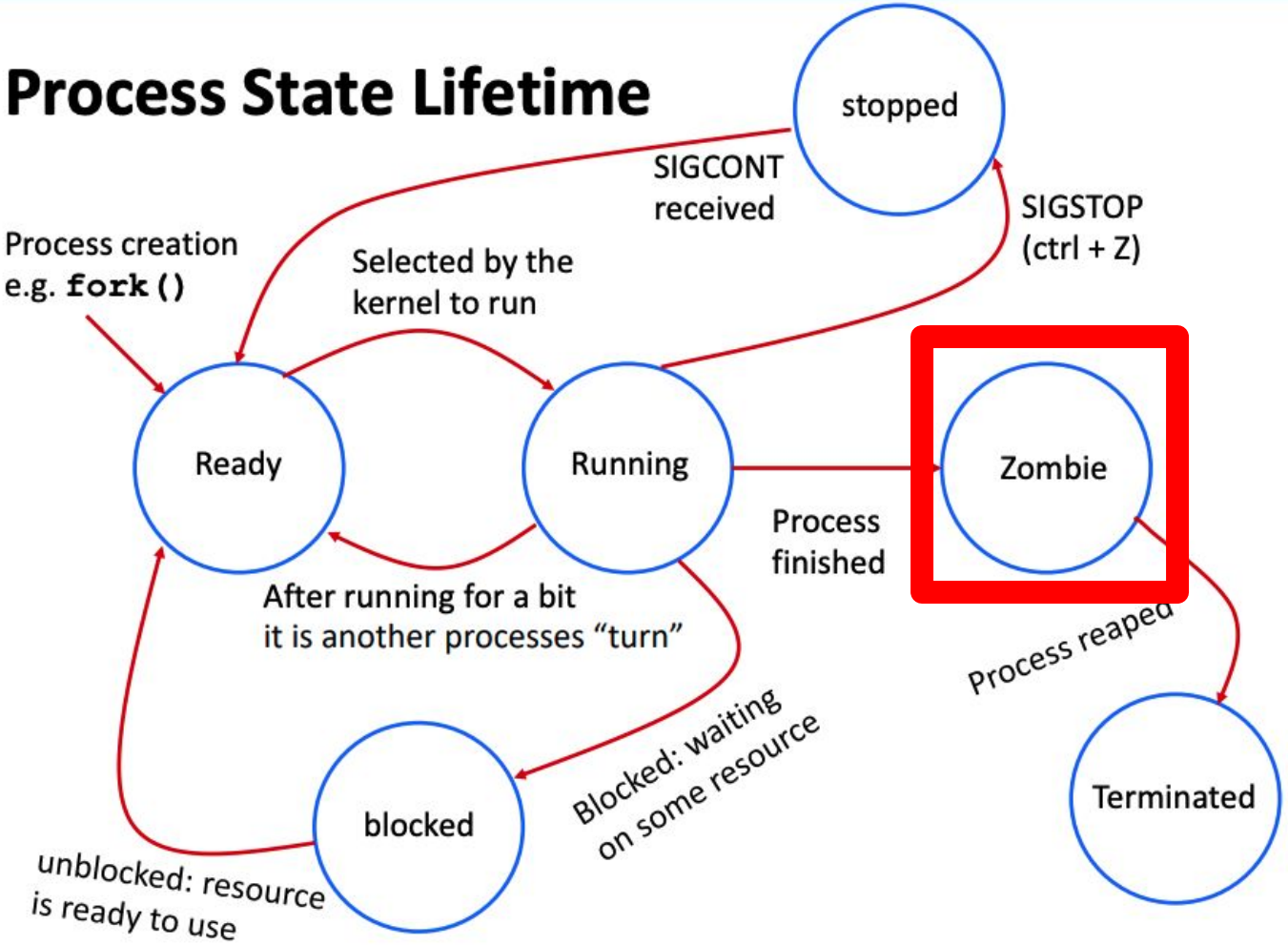
Signals in Process Groups

- Signals are relayed to all processes in a **process group**
- Terminal signals (SIGINT, SIGTSTP, etc) will be relayed to all processes in then **process group in the foreground**
- kill(2) can send signals to certain **process groups**
    - may also use killpg(2)

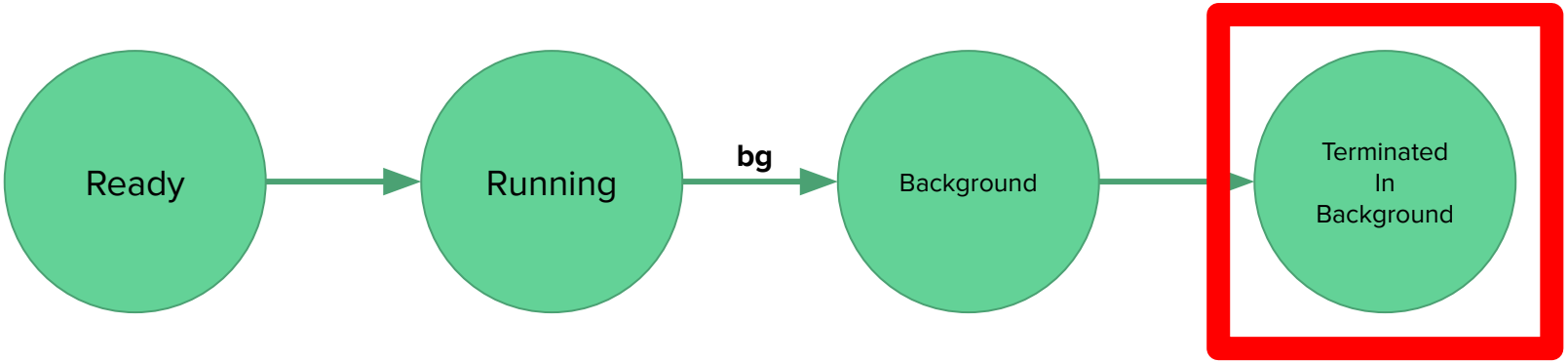
Zombies and Orphans

- Zombie Process: is a process that have finished execution but still has entry in the process table of the parent.
- Orphan Process: is a process whose parent process finished execution and does not exist anymore

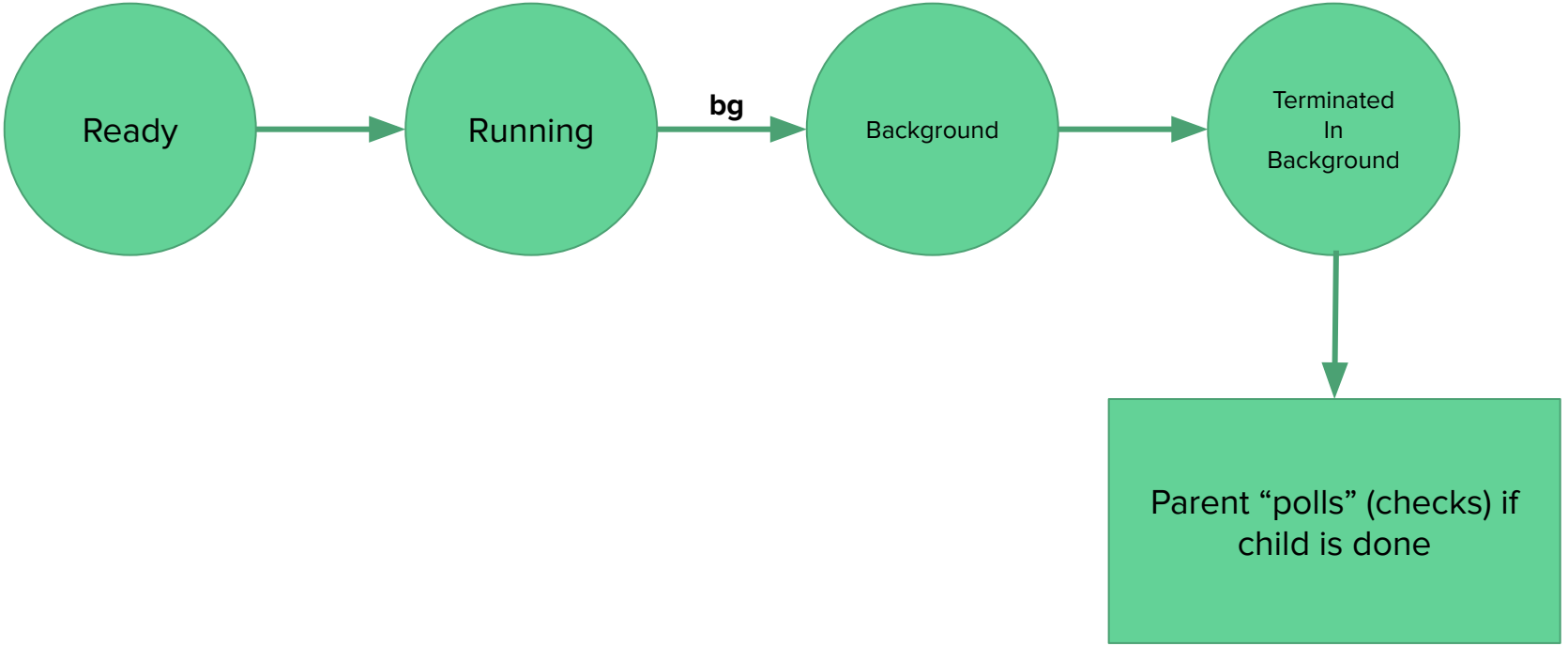
Process State Lifetime



Zombies In Background

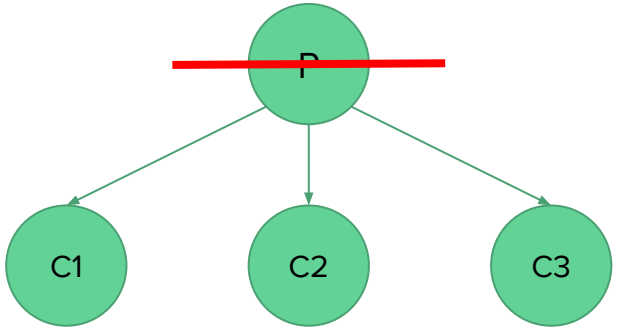


Zombies In Background



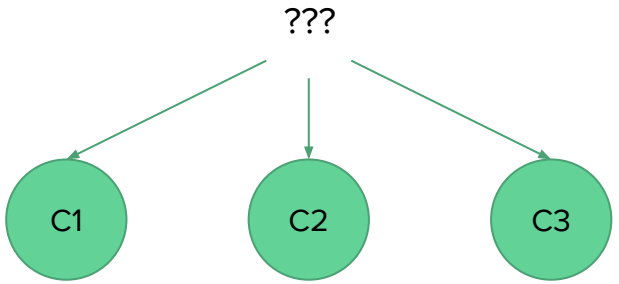
Orphans

```
int main() {
     for (i = 1 to 3) {
          fork();
          if (child) {
               execute(sleep 100);
          }
     }
     exit(0);
}
```



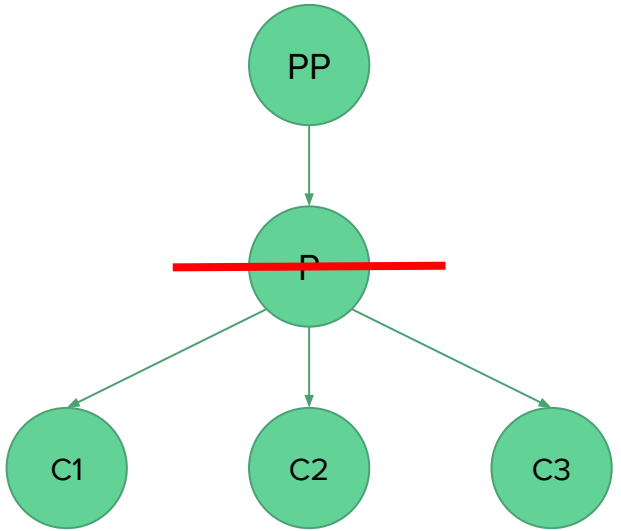
Orphans

```
int main() {
     for (i = 1 to 3) {
          fork();
          if (child) {
               execute(sleep 100);
          }
     }
     exit(0);
}
```



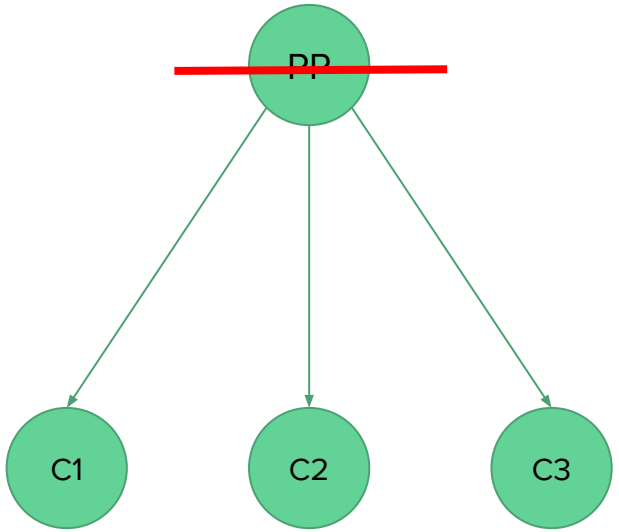
Orphans

```
int main() {
     for (i = 1 to 3) {
          fork();
          if (child) {
               execute(sleep 100);
          }
     }
     exit(0);
}
```



Orphans

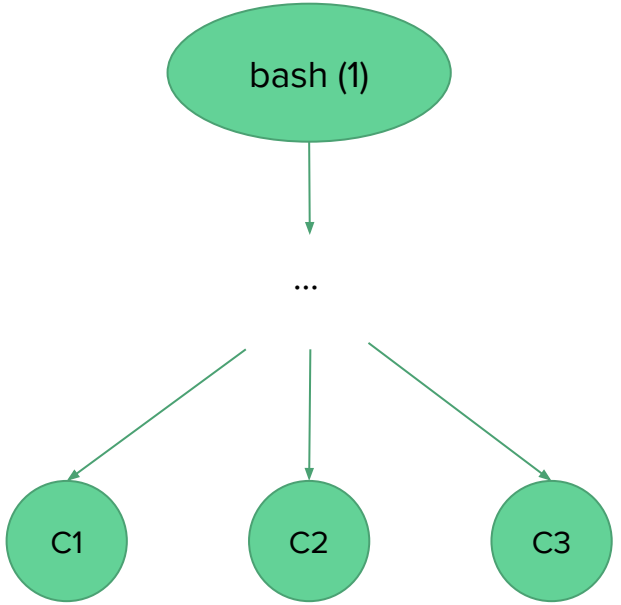
```
int main() {
     for (i = 1 to 3) {
          fork();
          if (child) {
               execute(sleep 100);
          }
     }
     exit(0);
}
```



Very Relevant in PennOS!

Orphans

```
int main() {
     for (i = 1 to 3) {
          fork();
          if (child) {
               execute(sleep 100);
          }
     }
     exit(0);
}
```



Waitpid

```
pid_t waitpid(pid_t pid, int *status, int options);
```

- Tracks the STATUS change in each child process group
- What are the different values of PID and options?
- What is the return value of waitpid()

Waitpid

```
pid_t waitpid(pid_t pid, int *status, int options);
```

Options:

- WNOHANG: do not wait for process to finish, but “collect” already finished or changed state process
- WUNTRACED: also return if the child stopped

Pid

- -1: wait for all children
- -(pgid): wait only for children from a specific process group
- Pid: wait only for a child with a specific pid

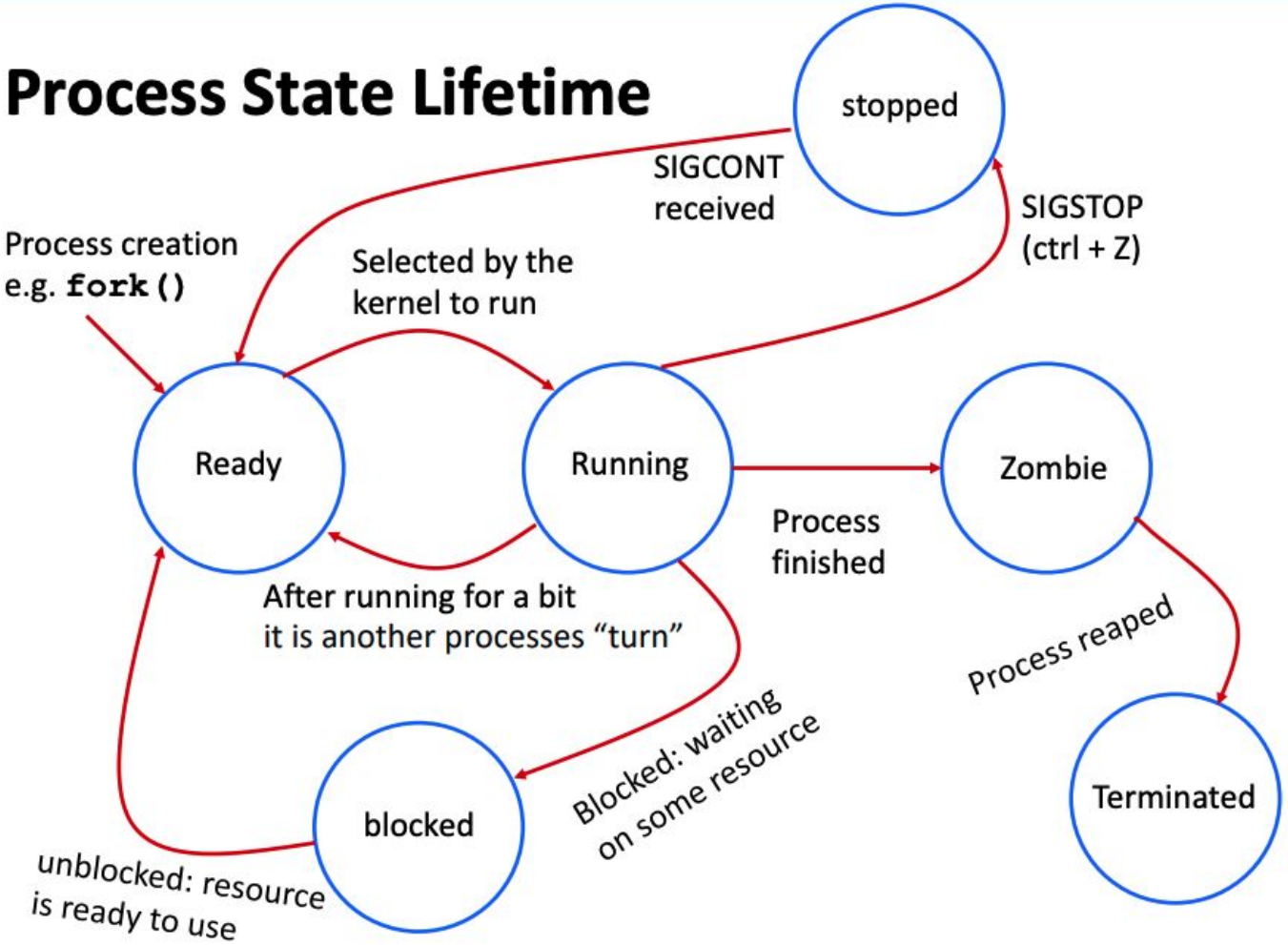
How it ties to Project 1

- How does waitpid and its options come up?
- When should terminal control change?
- Are there zombies which happen in project 1?
- How do process groups come up?

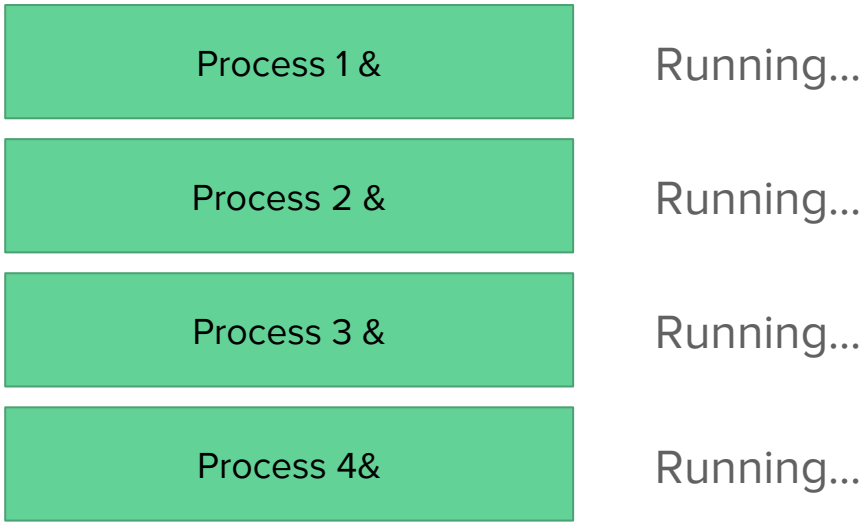
How it ties to Project 1

- Depending on the job being either foreground or background, will have to use different waitpid arguments
- Terminal control will have to go to the foreground process
- When background jobs first finish, before they are waited on they temporarily zombies
- Jobs will be separated into process groups

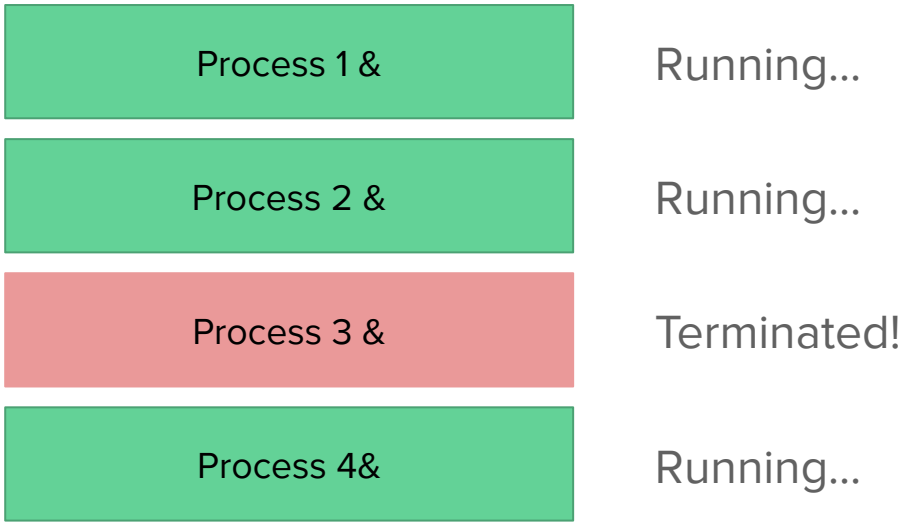
Process State Lifetime



Polling


Process 1 &
Running...
Process 2 &
Running...
Process 3 &
Running...
Process 4&
Running...

Polling



How do we know it terminated?
➜waitpid(2)

Polling

```
penn-shell> sleep 100 &

penn-shell> ls

[output of ls]

penn-shell>
```

We want to ‘poll’ for any child process status updates

Polling

- Think about what information you want to store in the job deque
  - Process group id? Each pid? Status?
- How do we want to make sure that EVERY process in a particular pipelined job is done?
  - `sleep 1 | sleep 2 | sleep 100` is only finished when sleep 100 is done
- Be careful not to be “busy waiting”, but only polling for each job once
- Be aware that STOP, CONTINUE are also tracked by waitpid(2)

Wrap Up

- We'll post recording/slides on the website soon
- Quick reminder: Penn Shell due This Friday. Late deadline next Tuesday
  - Read man pages! They really help
- Next week will be Midterm Review (Midterm Thursday)
- Any questions?