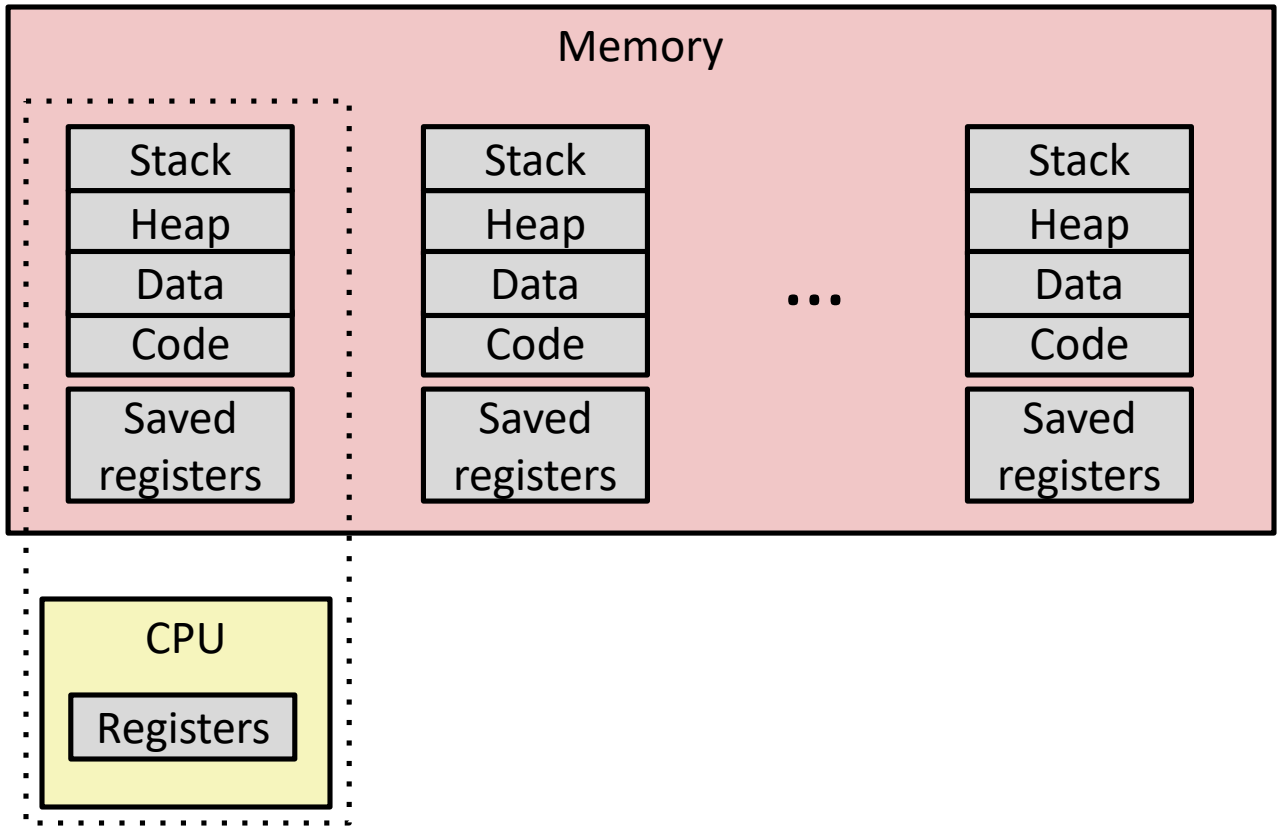
Architecture and OS, Process

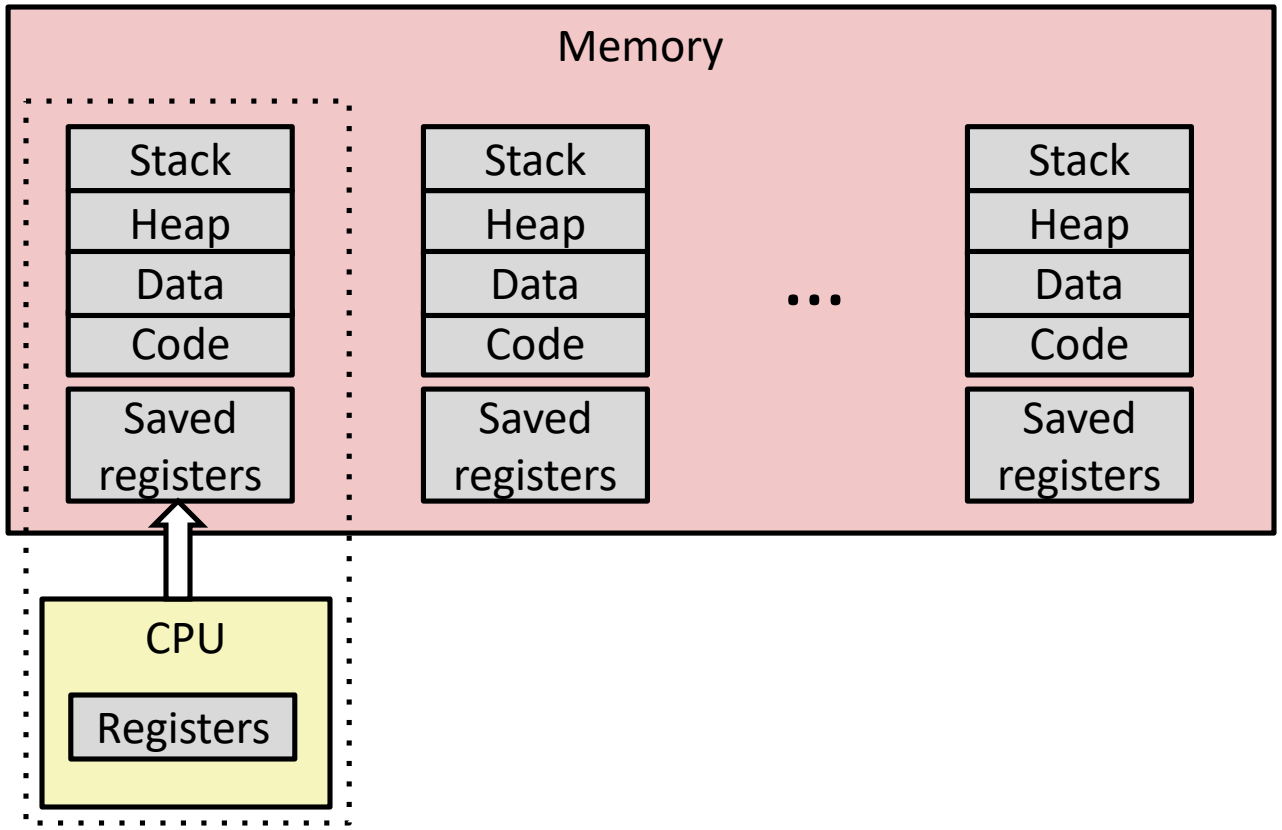
* Some material adapted from CS:APP, Prof. Bryant, O'Hallaron, Doeppner, Galvin, etc.

Multiprocessing: The (Traditional) Reality



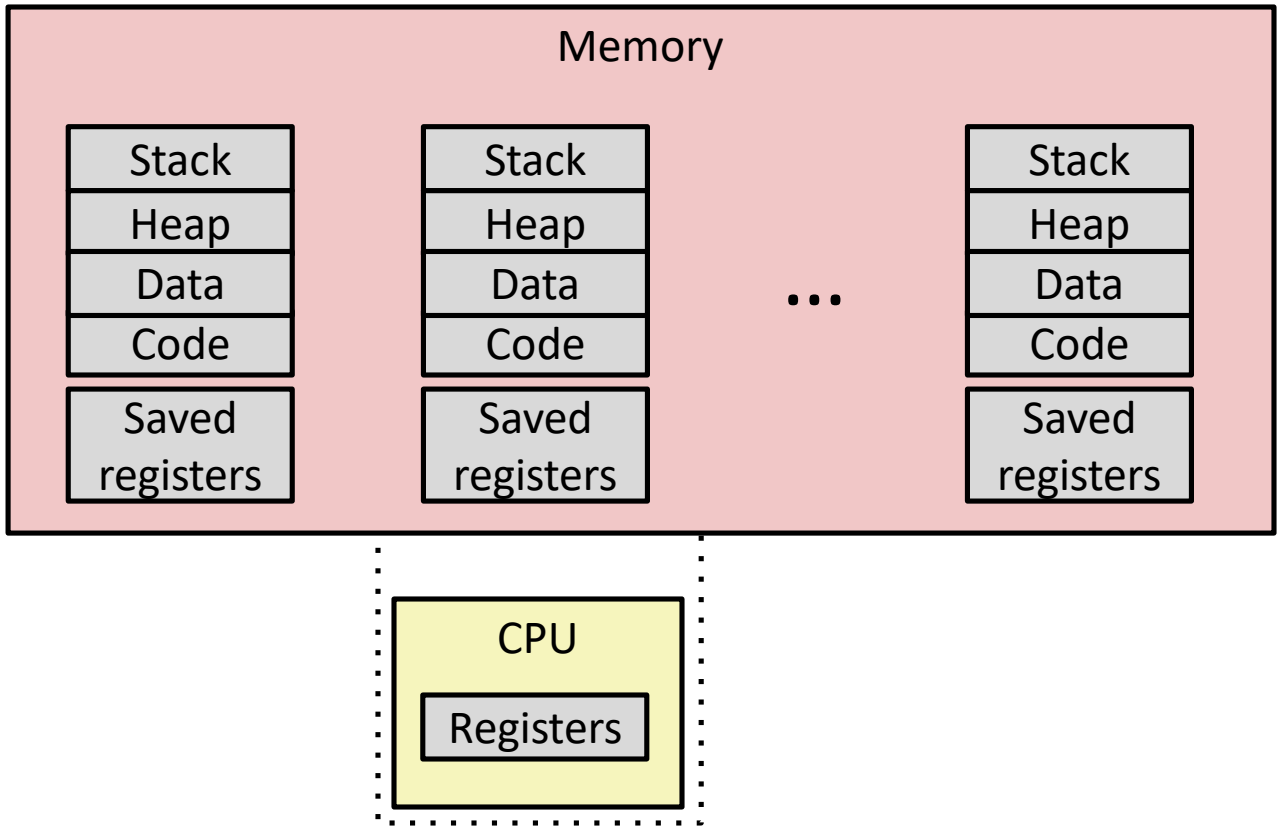
- Single processor executes multiple processes concurrently
  - Process executions interleaved (multitasking)
  - Address spaces managed by virtual memory system (later in course)
  - Register values for nonexecuting processes saved in memory

Multiprocessing: The (Traditional) Reality



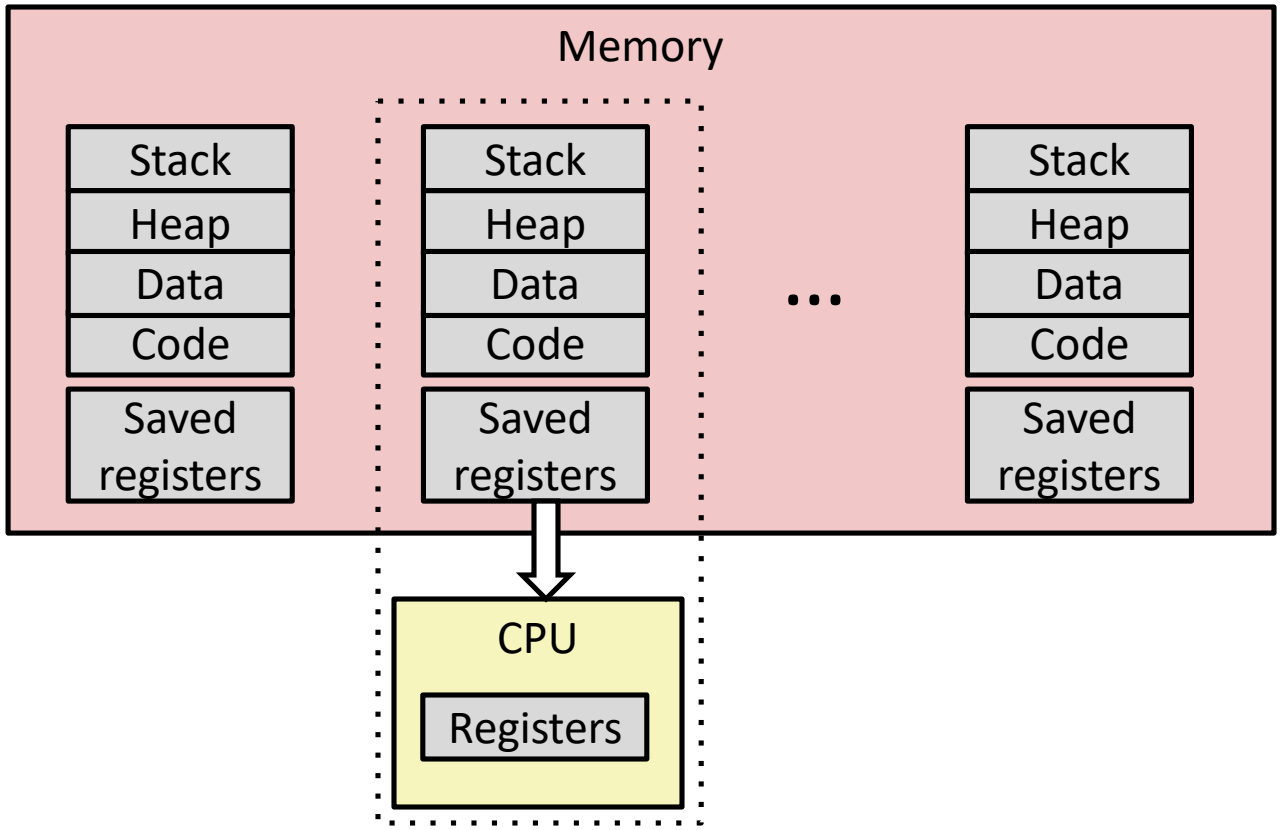
- Save current registers in memory

Multiprocessing: The (Traditional) Reality



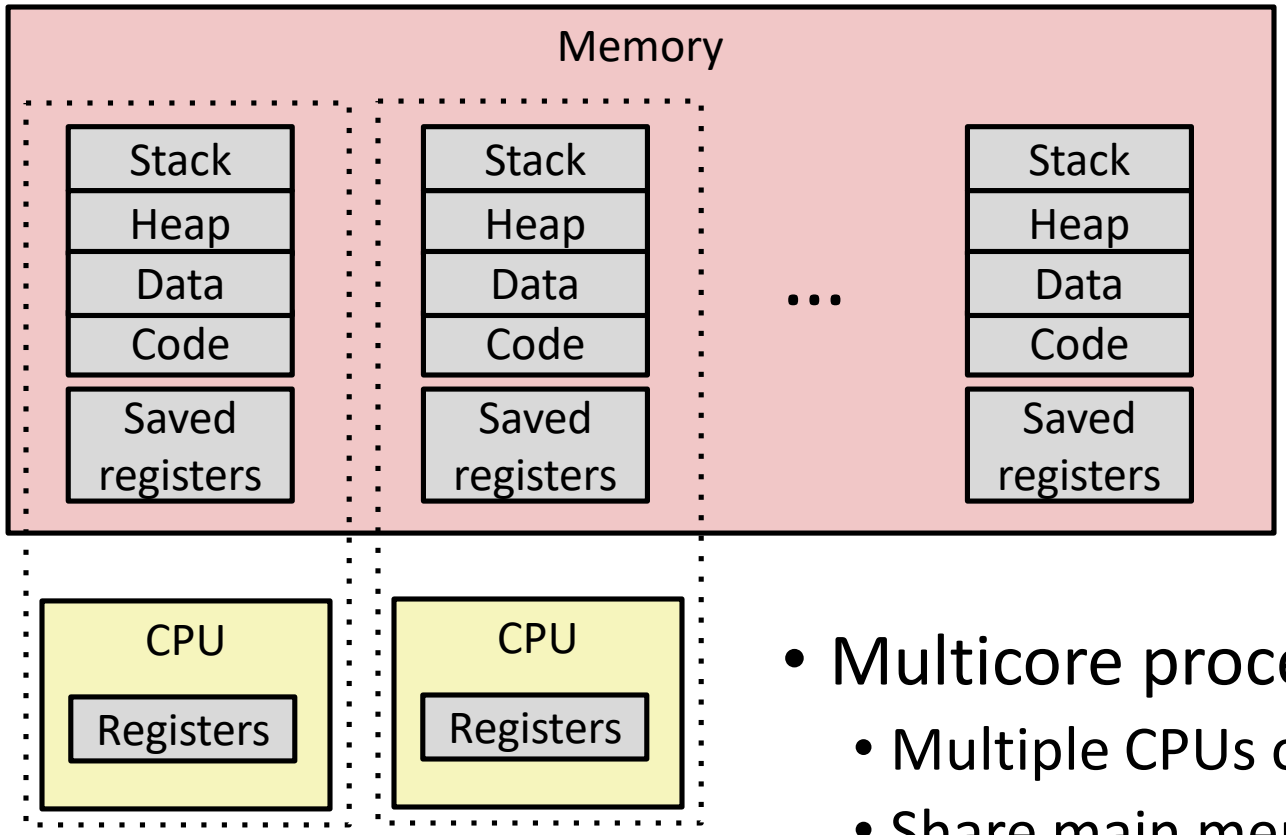
- Schedule next process for execution

Multiprocessing: The (Traditional) Reality



- Load saved registers and switch address space (context switch)

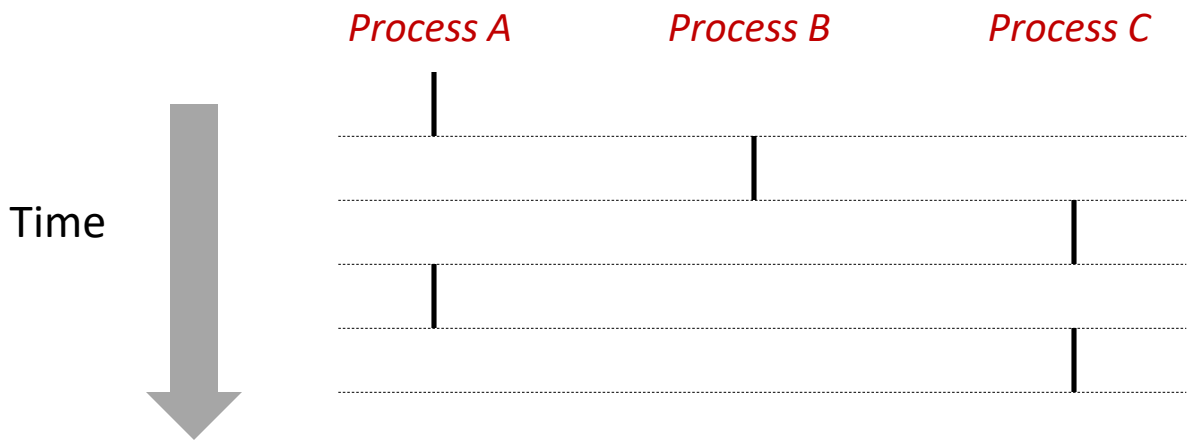
Multiprocessing: The (Modern) Reality



- Multicore processors
  - Multiple CPUs on single chip
  - Share main memory (and some caches)
  - Each can execute a separate process
    - Scheduling of processors onto cores done by kernel

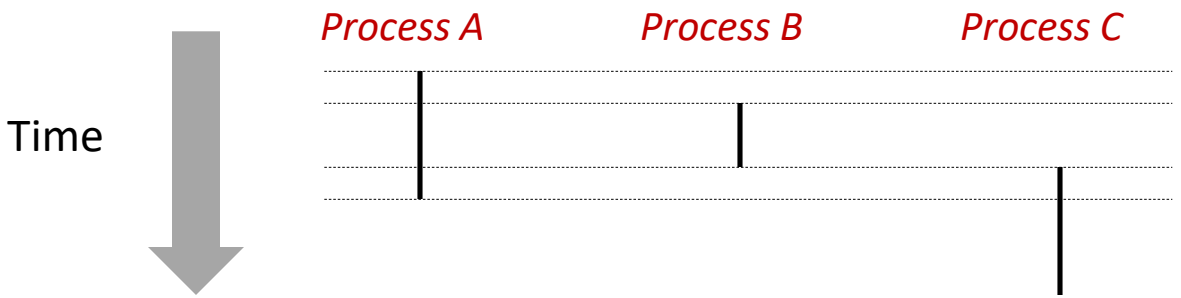
Concurrent Processes

- Each process is a logical control flow.
- Two processes *run concurrently* (*are concurrent)* if their flows overlap in time
- Otherwise, they are *sequential*
- Examples (running on single core):
  - Concurrent: A & B, A & C
  - Sequential: B & C



User View of Concurrent Processes

- Control flows for concurrent processes are physically disjoint in time
- However, we can think of concurrent processes as running in parallel with each other

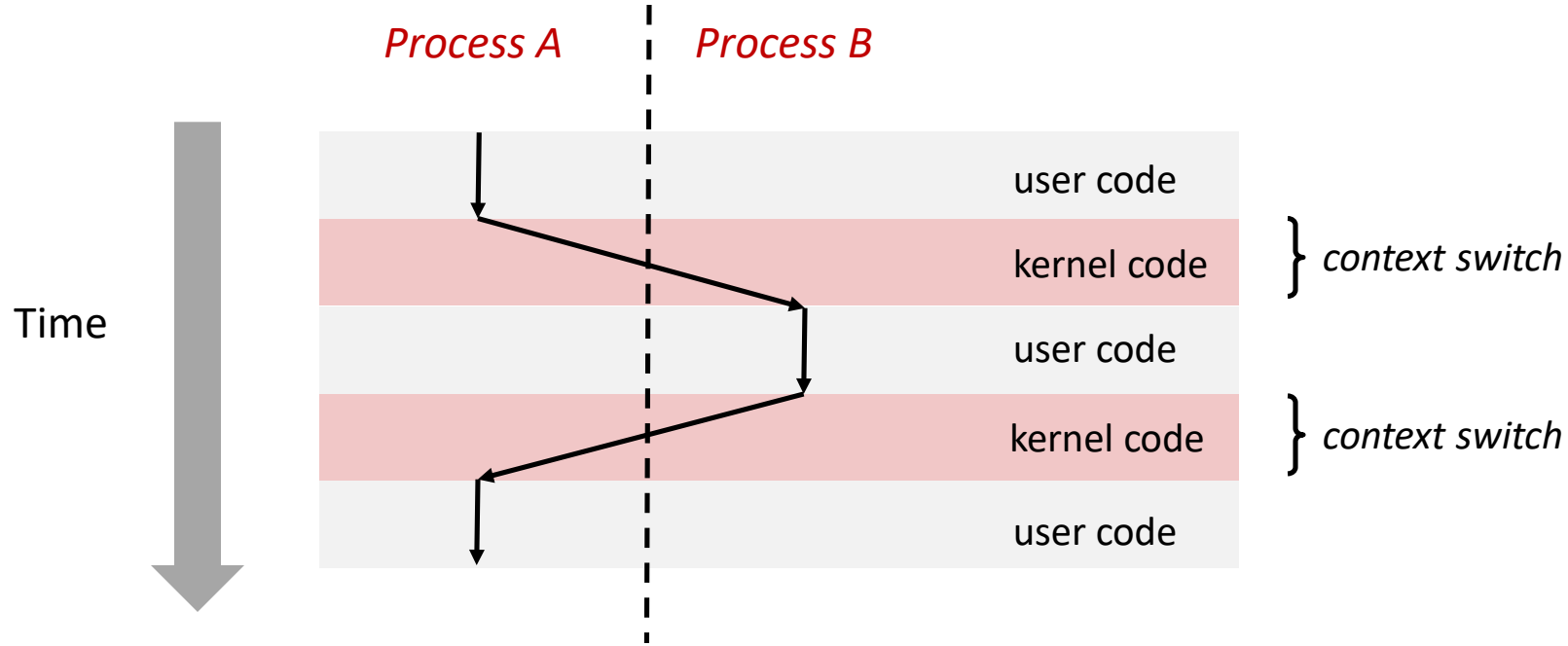


Context Switching

- Processes are managed by a shared chunk of memory-resident OS code called the *kernel*
- Control flow passes from one process to another via a *context switch*

Context Switching

- Processes are managed by a shared chunk of memory-resident OS code called the *kernel*
- Control flow passes from one process to another via a *context switch*



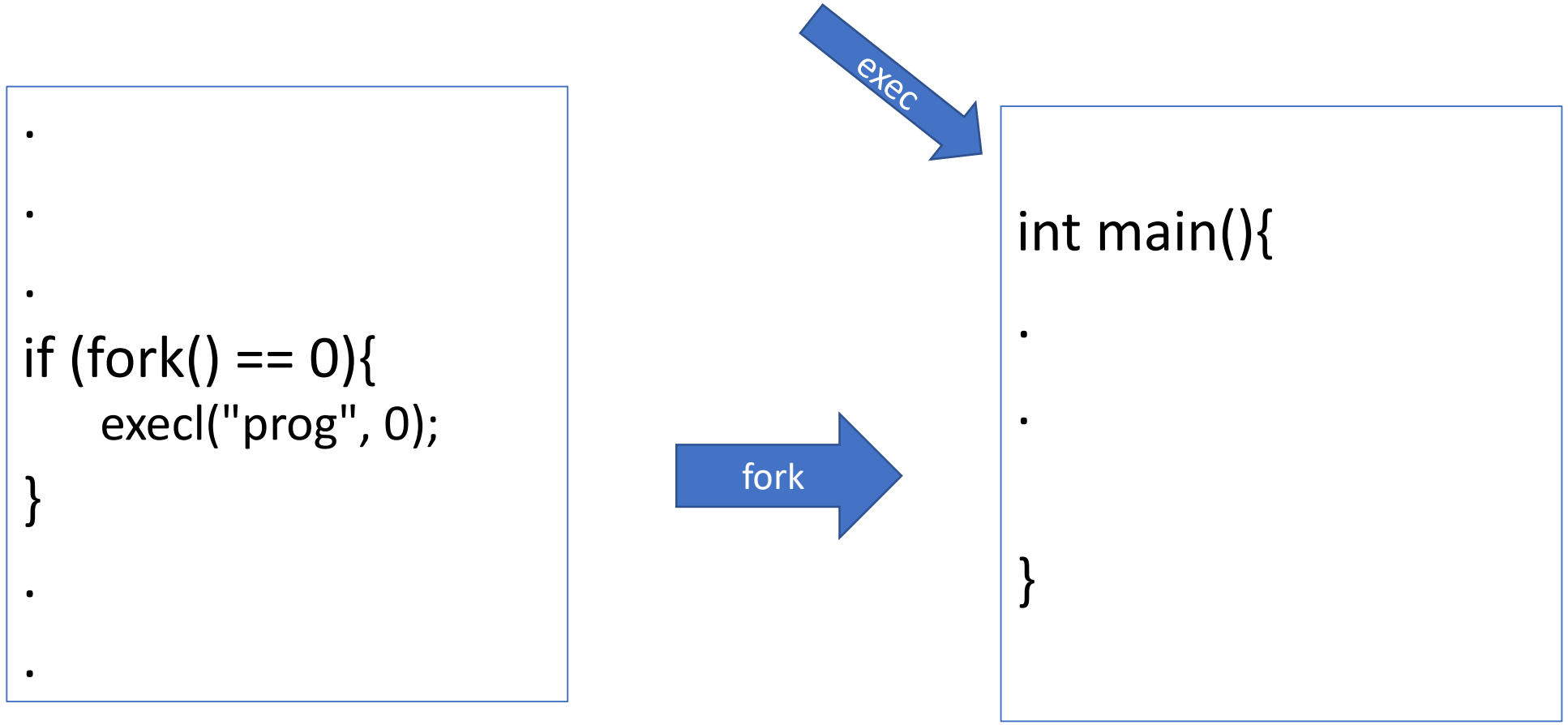
Obtaining Process IDs

- `pid_t getpid(void)`
  - Returns PID of current process
- `pid_t getppid(void)`
  - Returns PID of parent process

Creating Your Own Processes

```
#include <unistd.h>
int main( ) {
    pid_t pid;
    if ((pid = fork()) == 0) {
        /* new process starts running here */
    }
    /* old process continues here */
}
```

Putting Programs into Processes



Exec

- Family of related routines
  - we concentrate on one:
    - execv(program, argv)

```
char *argv[] = {“./MyProg", "12", (void *)0};
if (fork() == 0) {
        execv("./MyProg", argv);
}
```

Loading a New Image