Introduction to Concurrency (Processes, Threads, Interrupts, etc.)

Professor Hugh C. Lauer
CS-3013, Operating Systems

Concurrency — things happening at the same time

- Since the beginning of computing, management of concurrent activity has been a *central* issue
- Concurrency between computation and input or output
- Concurrency between computation and user
- Concurrency between independent activities that take place at same time
- Concurrency between parts of large computations that are divided up to improve performance
- …

Early 1960s

- **Programmers tried to write programs that would read from input devices and write to output devices in parallel with computing**
  - Card readers, paper tape, line printers, etc.
- **Challenges**
  - Keeping the *buffers* organize
  - Synchronizing between I/O … mputation
  - Computation getting ahe…
  - I/O activity getting ahead o…

Definition: *buffer* – a region of memory from which an I/O device gets data or into which an I/O device puts data

Late 1960s — Shared Computing Services

- **Multiple simultaneous, independent users of large computing facilities**
  - E.g., *Time Sharing* systems of university computing centers
- **Data centers of large enterprises**
  - Multiple accounting, administrative, and data processing activities over common databases
- **…**

Modern Workstations and PCs

- **Multiple windows in personal computer doing completely independent things**
  - *Word*, *Excel*, *Photoshop*, E-mail, music, etc.
- **Multiple activities within one application**
  - E.g., in Microsoft *Word*
    - Reading and interpreting keystrokes
    - Displaying what you typed
    - Formatting line and page breaks
    - Spell checking
    - Hyphenation
    - ….

Impossible to do all of these things in one single-threaded program and still get the performance and responsiveness expected by users.

Modern Game Implementations

- **Multiple characters in game**
  - Concurrently & independently active
- **Multiple constraints, challenges, and interactions among characters**
- **Multiple players**

Traditional Challenge for OS

- **Useful set of abstractions that help to**
  - Manage concurrency
  - Manage synchronization among concurrent activities
  - Communicate information in useful way among concurrent activities
  - Do it all efficiently

Technological Pressure

- **From early 1950s to early 2000s, single processor computers increased in speed by 2× every 18 months or so**
  - Moore's Law
- **Multiprocessing was somewhat of a *niche* problem**
  - I.e., computing systems with more than one CPU
  - Specialized computing centers, techniques

Technological Pressure (continued)

- **No longer!**
- **Modern microprocessor clock speeds are no longer increasing as predicted by Moore's Law**
- **Microprocessor density on chips still is!**

Sort of!

⇒ **multi-threaded and multi-core processors are now *de facto* standard**

  - Even on low-end PCs!S

Modern Challenge

- **Methods and abstractions to help software engineers and application designers …**
  - Take advantage of inherent concurrency in modern application systems
  - Exploit multi-processor and multi-core architectures that are becoming ubiquitous
  - Do so with relative ease

Fundamental Abstraction

- **Process**
- **… *aka* Task**
- **… *aka* Thread**
- **… *aka* Job**
- **… *aka* [other terms]**

Ostep, Chapter 4

Definition

- ***Process* (generic):– A *particular* execution of a *particular* program.**
  - Requires time, space, and (perhaps) other resources
- **Separate from all other executions of the same program**
  - Even those at the same time!
- **Separate from executions of other programs**

Process (continued)

- **Can be**
  - Interrupted
  - Suspended
  - Blocked
  - Unblocked
  - Started or continued
- **Fundamental *abstraction* of all modern operating systems**
- **Note: "Process" in Unix, Linux, and Windows is a heavyweight concept with more implications than this simple definition**

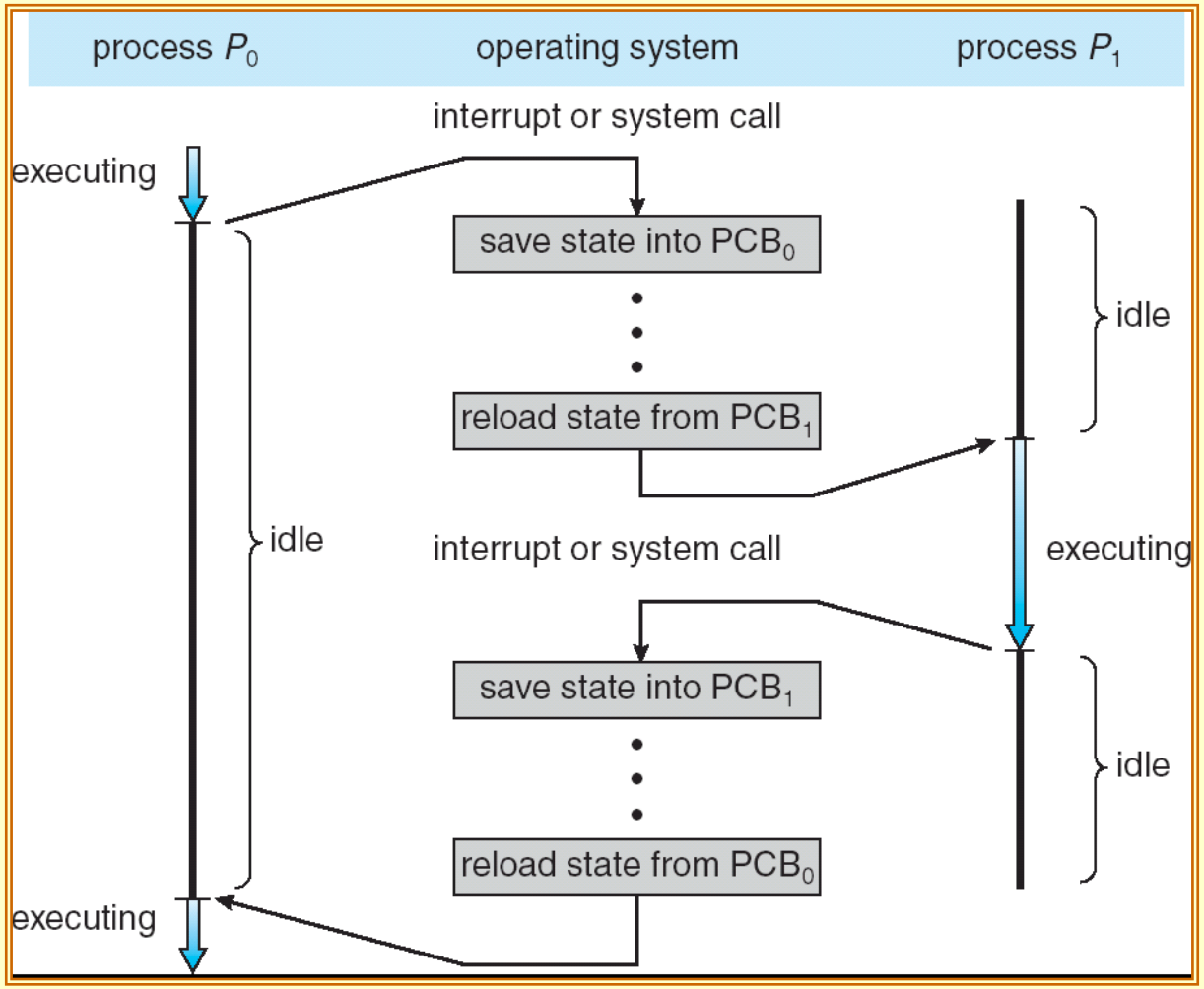
Process (a generic term – continued)

- Concept emerged and evolved in 1960s
- Intended to make sense out of mish-mash of concurrent programming techniques that bedeviled software engineers
- Analogous to police or taxi dispatcher!

Background – *Interrupts*

- **A mechanism in (nearly) all computers by which a running program can be suspended in order to cause processor to do something else**
- **Two kinds:–**
  - *Traps* – synchronous, caused by running program
    - Deliberate: e.g., system call
    - Error: divide by zero
  - *Interrupts* – asynchronous, spawned by some other concurrent activity or device.
- **Essential to the usefulness of computing systems**

Switching from process to process



Hardware Interrupt Mechanism

- **Upon receipt of electronic signal, the processor**
  - Saves current PSW to a fixed location
  - Loads new PSW from another fixed location
- **Definition: PSW — *Program Status Word***
  - *Program counter*
  - Condition code bits (comparison results)
  - Interrupt enable/disable bits
  - Other control and mode information
    - E.g., privilege level, access to special instructions, etc.
- **Occurs *between* machine instructions**
  - An abstraction in modern processors (see OSTEP, §34.6)

Interrupt handler

```
/* Enter with interrupts disabled */
Save registers & state of interrupted computation
Load registers & state needed by handler

Examine cause of interrupt
Take appropriate action (brief)

Reload registers & state of interrupted computation
Reload interrupted PSW and re-enable interrupts

                        or

Load registers & state of another computation
Load its PSW and re-enable interrupts
```

Requirements of interrupt handlers

- Fast
- Avoid possibilities of interminable waits
- Must not count on correctness of interrupted computation
- Must not get confused by multiple interrupts in close succession
- …

- More challenging on multiprocessor systems

Result

- **Interrupts make it possible to support concurrent activities**
  - Even on machines with only one processor
- **Don't help in establishing some kind of orderly way of thinking**
- **Need something more**

Result (continued)

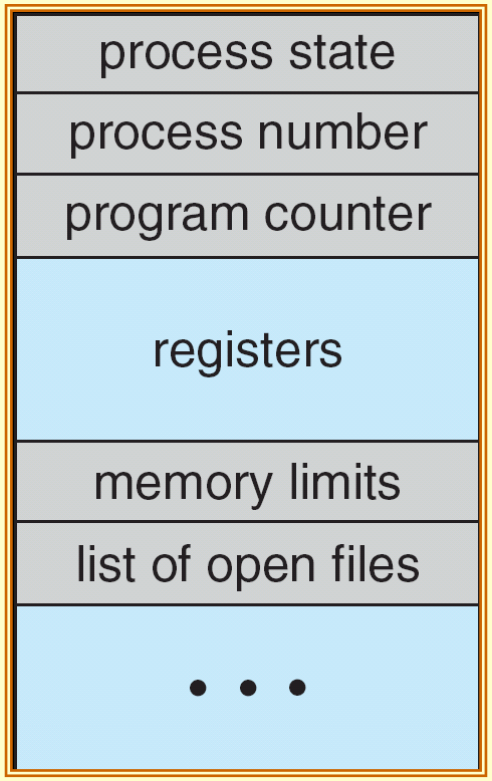
- **Hence, emergence of generic concept of *process***
  - (or whatever it is called in a particular operating system and environment)

- **Notion of *process* allows us to abstract interrupts and interleaving so that we can concentrate on each executing program separately**

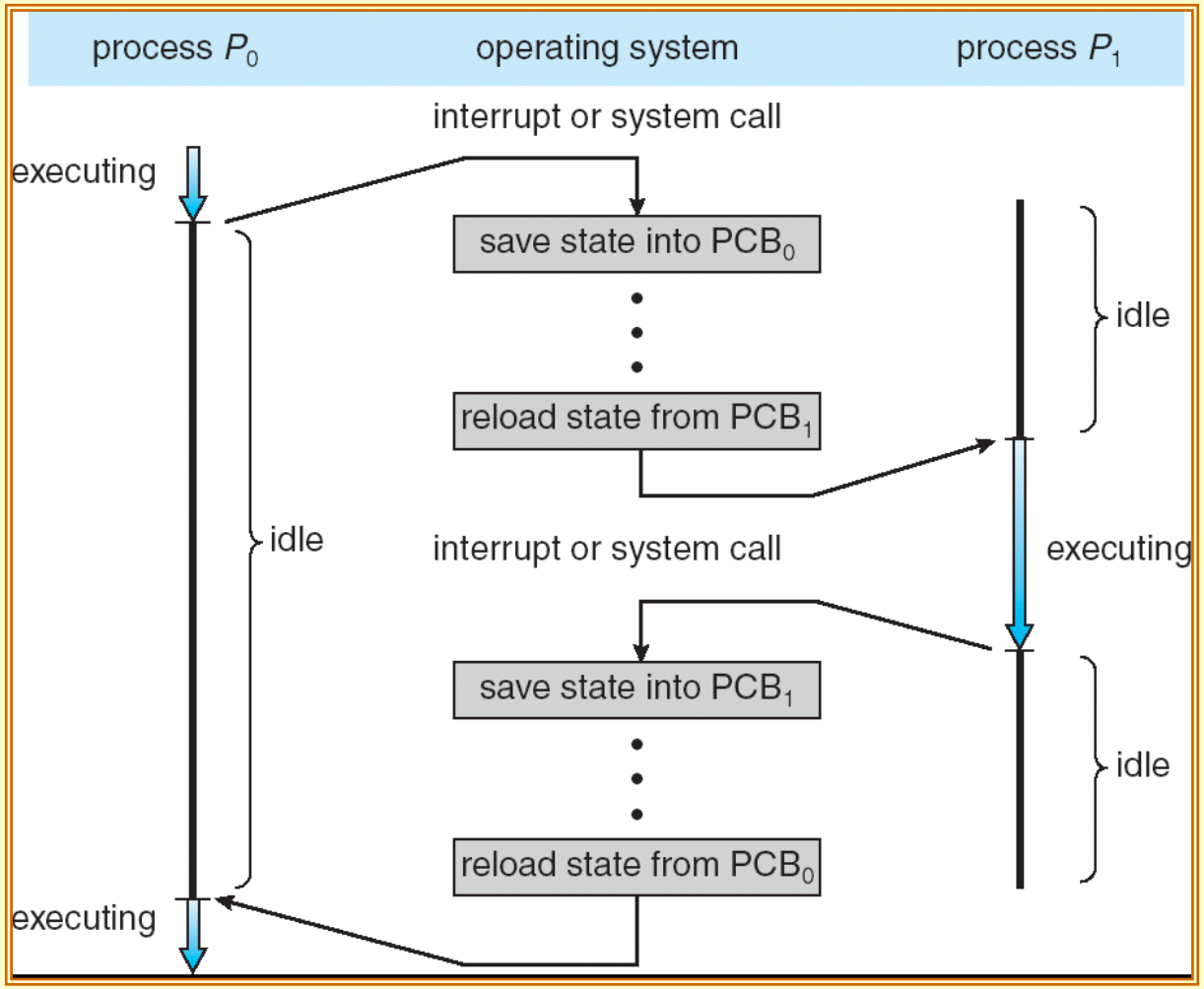
Information needed to implement processes

- **PSW (program status word)**
  - Program counter
  - Condition codes
  - Control information – e.g., privilege level, priority, etc
- **Registers, stack pointer, etc.**
  - Whatever hardware resources needed to compute
- **Administrative information for OS**
  - Owner, restrictions, resources, etc.
- **Other stuff …**

Process Control Block (PCB)
(example data structure in an OS)



Switching from process to process

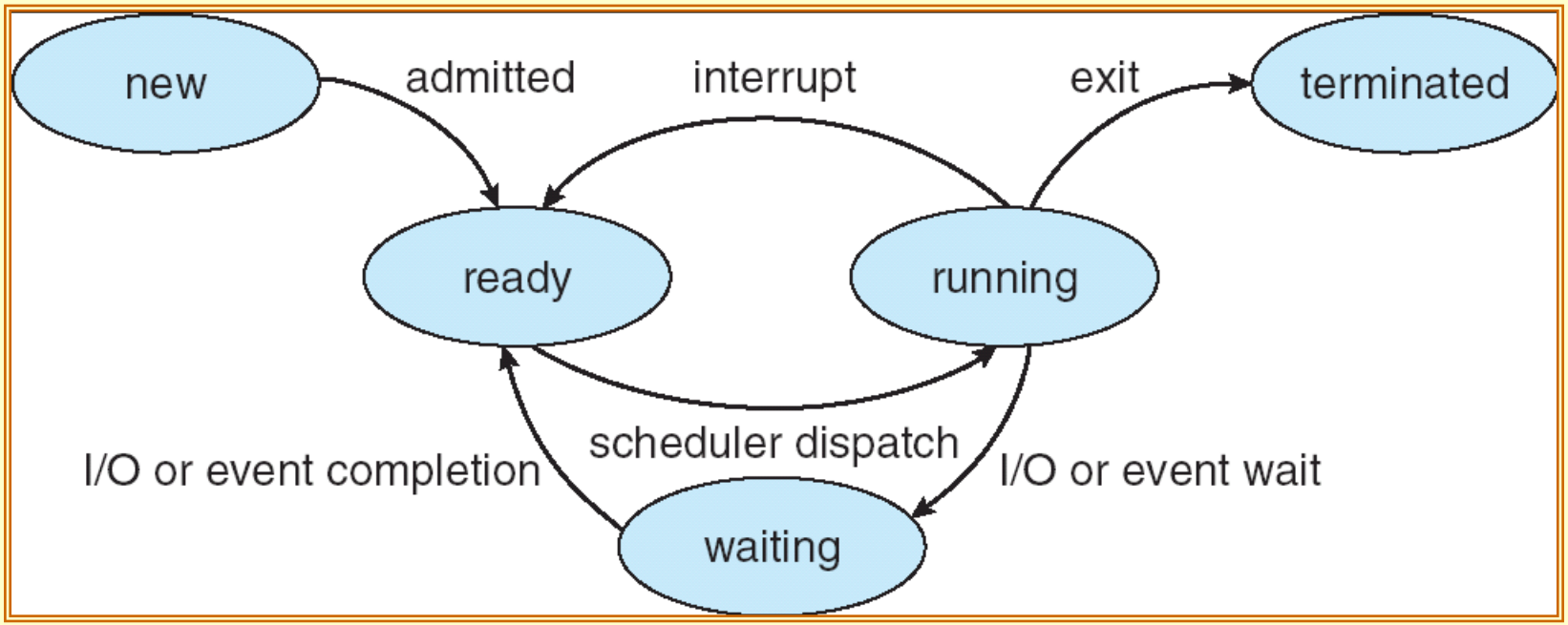


Result

- **A very clean way of thinking about separate computations**

- **Processes can *appear* be executing in parallel**
  - Even on a single processor machine
- **Processes really *can* execute in parallel**
  - Multi-processor, multi-core, or multi-threaded hardware
- **…**

Process States



The Fundamental Abstraction of the OS

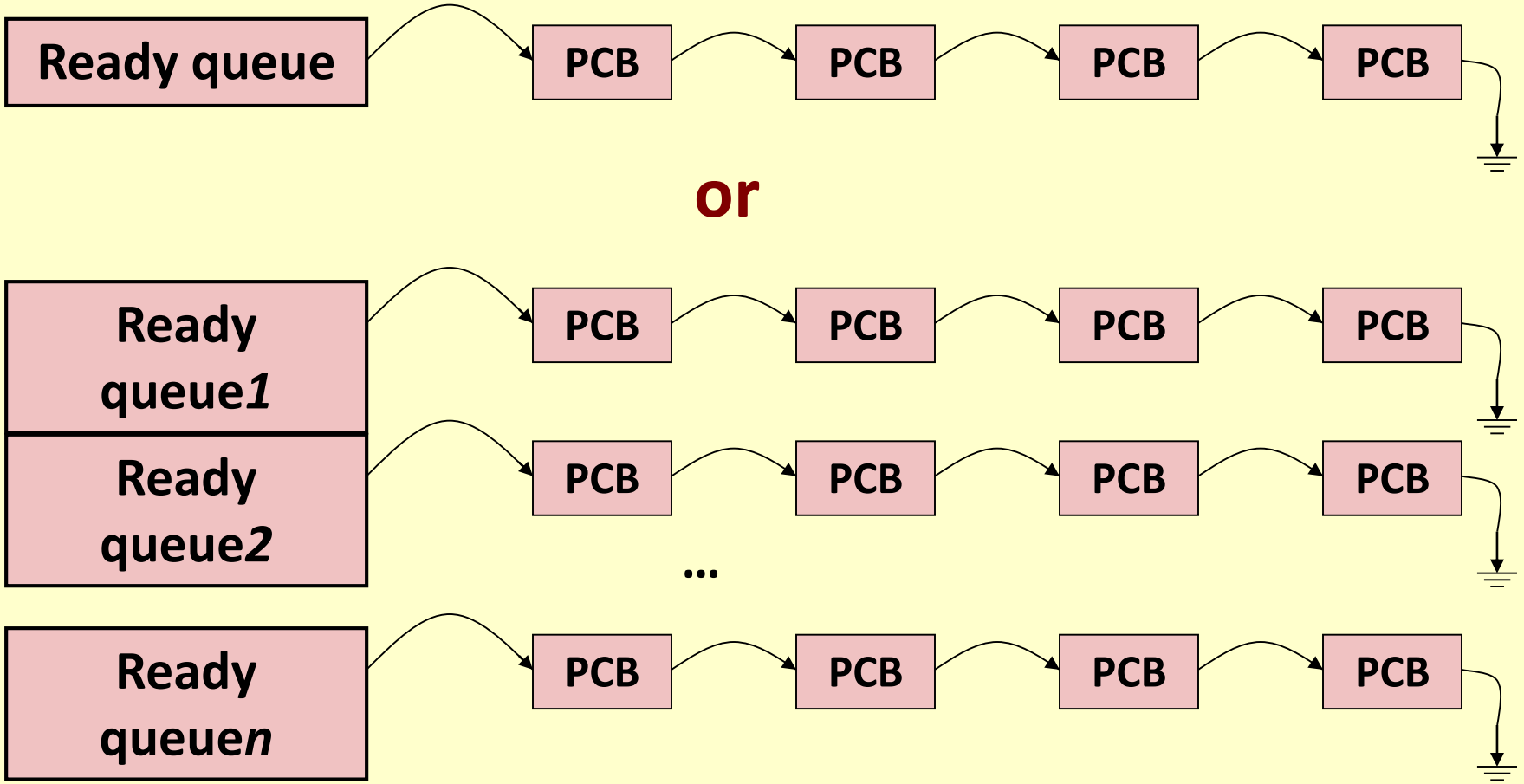
- Each process has its own "virtual" processor
- Each process can be thought of as an independent computation

Decoupled physical processors from the running of programs!

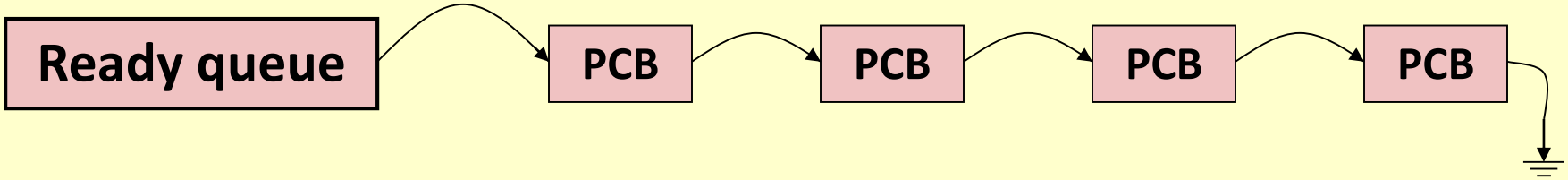
- On a fast enough physical processor, processes can look like they are really running concurrently
- On multi-processor or multi-core systems, processes really *do* run concurrently

Questions?

Implementation of Processes



Implementation



- **Action – *dispatch* a process to CPU**
  - Remove first PCB from ready queue
  - Load hardware registers and PSW
  - *Return from interrupt* or trap
- **Action – *interrupt* a process**
  - Save PSW and hardware registers in PCB
  - If not blocked, insert PCB back into *ReadyQueue* (in some order); otherwise, link it to some other queue or list
  - Take appropriate action
  - Dispatch same or another process from *ReadyQueue*

Timer interrupts

- **Can be used to enforce "fair sharing"**
- **Current process goes to end of *ReadyQueue***
  - *After* other processes of equal or higher priority
- **Simulates concurrent execution of multiple processes on same processor**

Processes – Switching

- When a process is *running*, its hardware state is in the processor – PC, processor registers, etc.
- When the OS *suspends* running a process, it saves the hardware state in the PCB
- When the OS *dispatches* a process, it restores the hardware state from the PCB

Definition – *Context Switch*

- **The act of switching a processor from one process to another**
  - E.g., upon interrupt or some kind of wait for event
- **Not a big deal in simple systems and processors**
- ***Very big deal* in large systems such as**
  - Linux and Windows
  - Pentium, Core i7, etc.

Many microseconds!

Definition — *Scheduling*

- The art and science of deciding *which* process to dispatch next …
- … and for how long …
- … and on which processor

Topic for later in this course

Questions?

Next Topic – Processes in
Unix, Linux, and Windows