Scheduling

Abhijit A.M.
abhijit.comp@coep.ac.in

Credits: Slides from os-book.com

Necessity of scheduling

Multiprogramming

- **Ability to run multiple programs at a time**

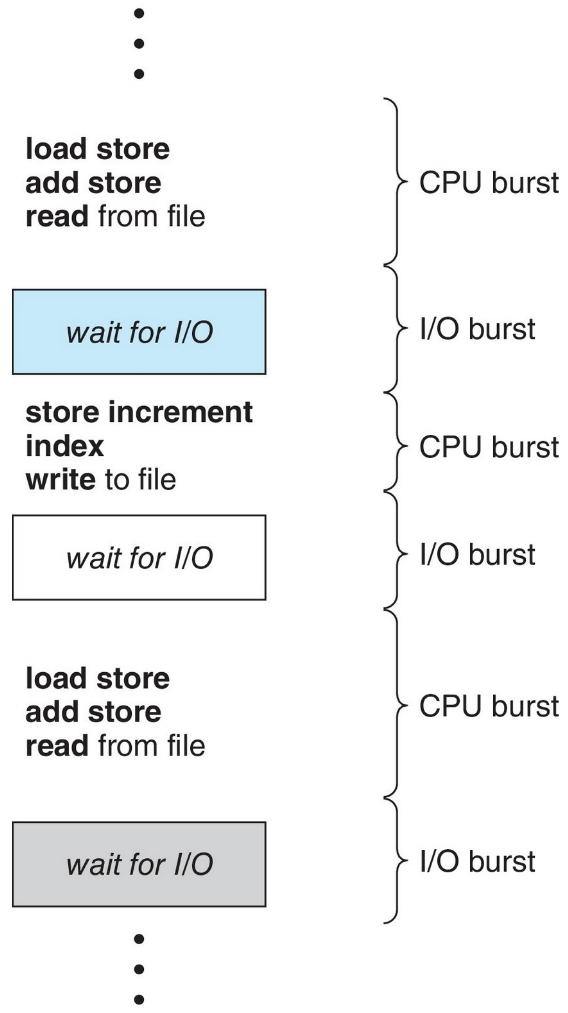
- **Increase use of CPU**
  - **CPU utilisation**

CPU Scheduling

- The task of selecting 'next' process/thread to execute on CPU and doing a context switch
- Scheduling algorithm
  - Criteria for selecting the 'next' process/thread and it's implementation
- Why is it important?
  - Affects performance !
  - Affects end user experience !
  - Involves money!

Observation: CPU, I/O Bursts

- Process can ‘wait’ for an event (disk I/O, read from keyboard, etc.)
- During this period another process can be scheduled
- CPU–I/O Burst Cycle:
  - Process execution consists of a cycle of CPU execution and I/O wait
  - CPU burst followed by I/O burst
  - CPU burst distribution is of main concern



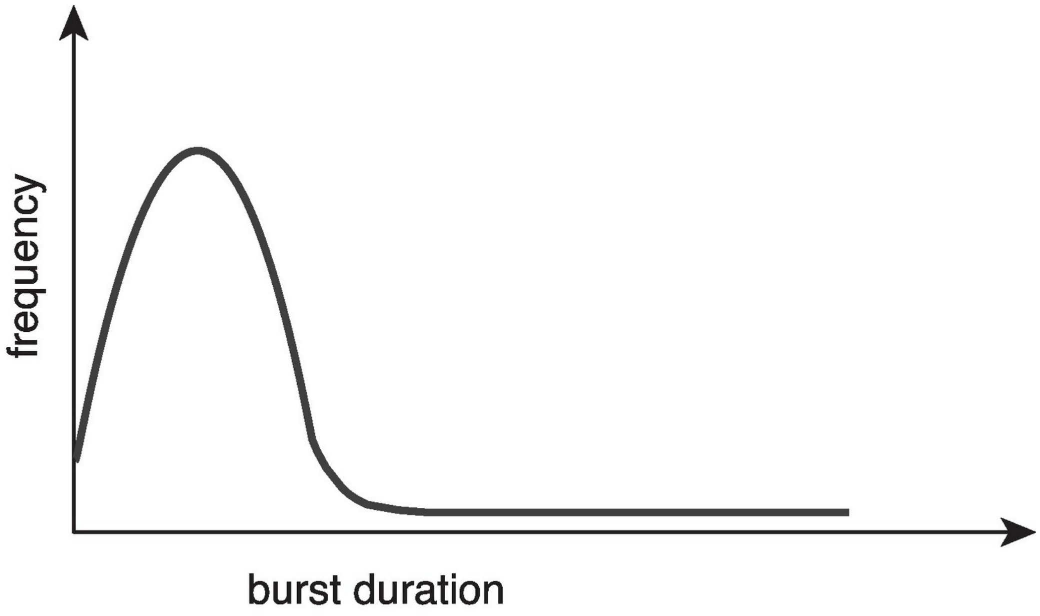
Let’s understand the problem

- **Programs have alternate CPU and I’O bursts**
  - **Some are CPU intensive**
  - **Some are I/O intensive**
  - **Some are mix of both**

A C code example:

```
f(int i, int j, int k) {
    j = k * i;          // CPU burst
    scanf("%d", &i);        // I/O burst
    k = i * j;// CPU burst
    printf("%d\n", k)://  I/O burst
    return k;
}
```

CPU bursts: observation



Large number of short bursts

Small number of longer bursts

Scheduler, what does it do?

- **From a list of processes, ready to run**
  - **Selects a process for execution**
  - **Allocates a CPU to the process for execution**
  - **Does “context switch”**

Context and Context Switch

- Context
  - Set of registers. Which ones?
  - All or some ?
  - Following registers on xv6 kernel: edi, esi, ebx, ebp, eip
    - Related to calling convention!
  - Linux kernel context: Much bigger!
  - Also includes: MMU setup

- Context switch
  - Process context -> kernel context
    - On interrupt, system call, or exception
  - Kernel context -> process context
    - Returning from system call, returning from interrupt handler, scheduling a process
  - In every switch, need to change to the set of registers “last in use” by that context, and also the MMU setup

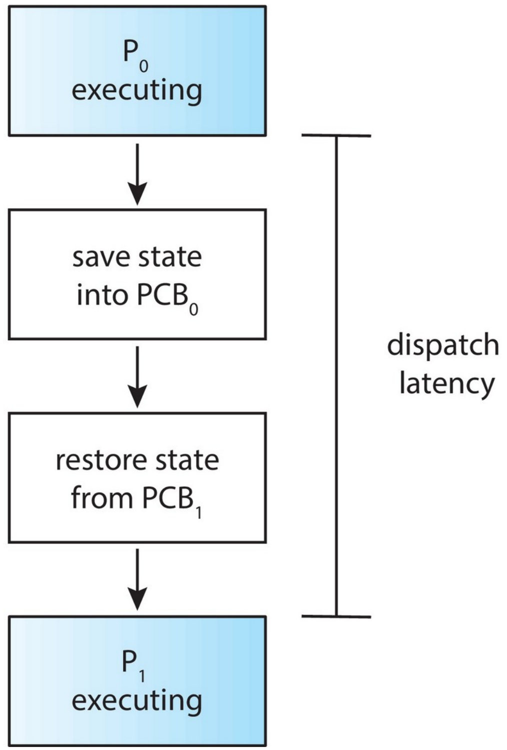
When is scheduler invoked?

1) Process Switches from running to waiting state
   - Waiting for I/O, etc.
2) Switches from running to ready state
   - E.g. on a timer interrupt
3) Switches from waiting to ready
   - I/O wait completed, now ready to run
4) Terminates

- Scheduling under 1 and 4 is nonpreemptive
- All other scheduling is preemptive
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

Dispatcher: A part of scheduler

- Gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- Dispatch latency
  - time taken to stop one process and start another running
- Xv6: swtch(), some tail end parts of sched(), trap(), trapret()



Dispatcher in action on Linux

- Run "`vmstat 1 3`"
  - Means run vmstat 3 times at 1 second delay
- In output, look at CPU:cs
  - Context switches every second

- Also for a process with pid 3323
- Run

  "`cat /proc/3323/status`"
  - See

    voluntary_ctxt_switches

    --> Process left CPU

    nonvoluntary_ctxt_switche

    --> Process was preempted

Scheduling criteria

- CPU utilization: Maximise
  - keep the CPU as busy as possible. Linux: idle task is scheduled when no process to be scheduled.
- Throughput : Maximise
  - # of processes that complete their execution per time unit
- Turnaround time : Minimise
  - amount of time to execute a particular process
- Waiting time : Minimise
  - amount of time a process has been waiting in the ready queue
- Response time : Minimise
  - amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)
-

To be studied later

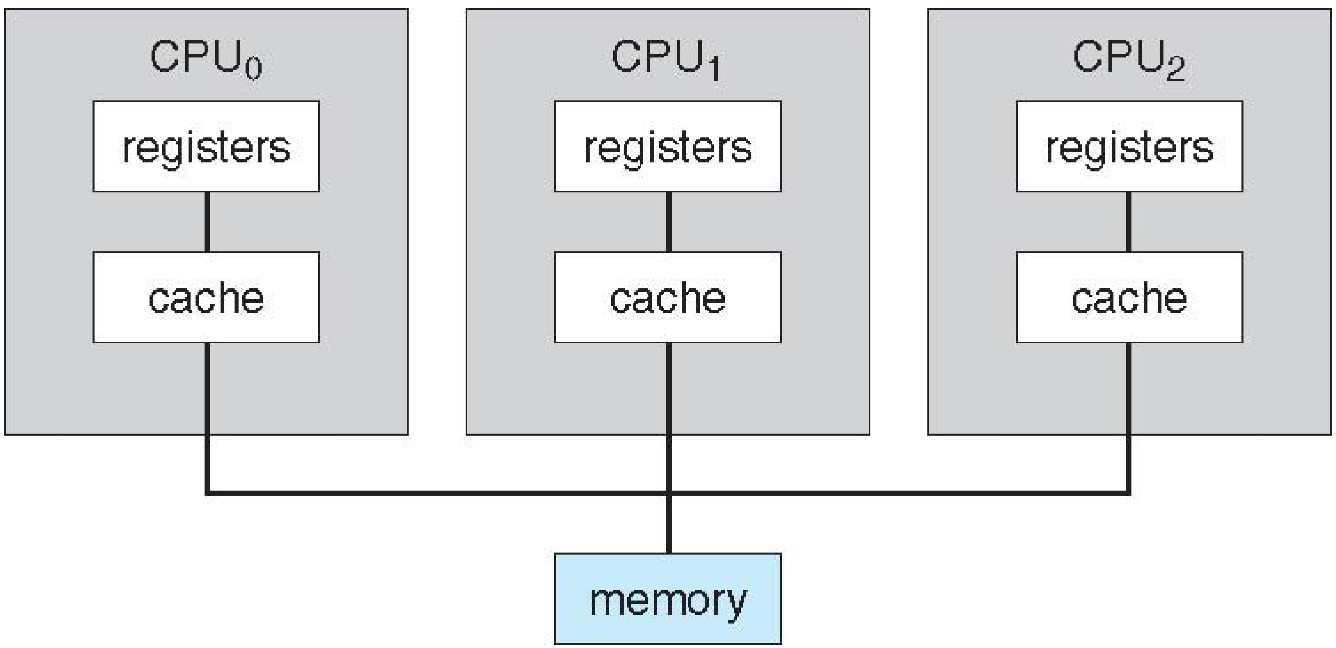
- Evaluation of scheduling criteria
- Different scheduling algorithms, and their characteristics
    - Round Robin, FIFO, Shortest Job First, Priority, Multi-level Queue, etc.
- Thread scheduling

Multi Processor Scheduling

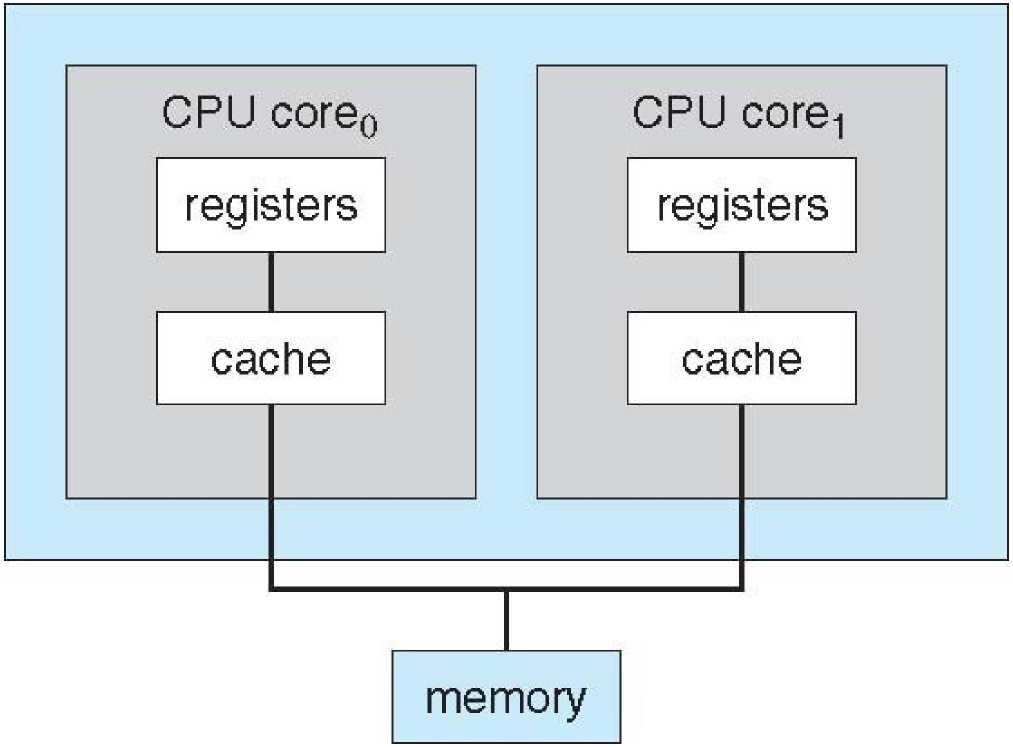
Multiprocessor systems

- **Each processor has separate set of registers**
  - **All: eip, esp, cr3, eax, ebx, etc.**
- **Each processor runs independently of others**
- **Main difference is in how do they access memory**

Symmetric multiprocessing (SMP)



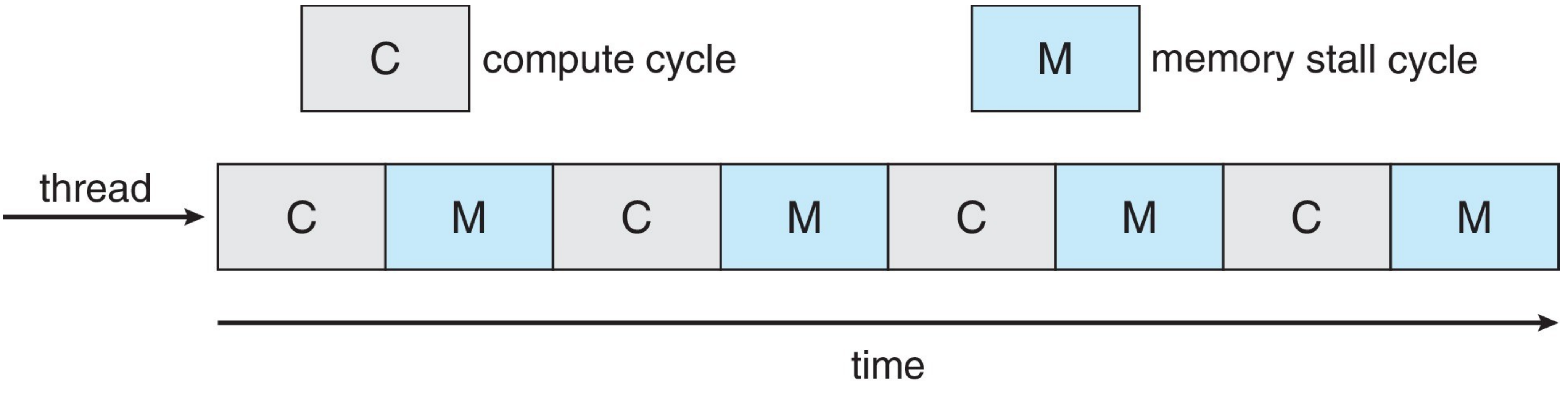
Multicore systems (also SMP)



No difference from the perspective of OS. The hardware ensures that OS sees multiple processors and not multiple cores.

Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consumes less power
- Multiple threads per core also growing
  - **Takes advantage of memory stall to make progress on another thread while memory retrieve happens**



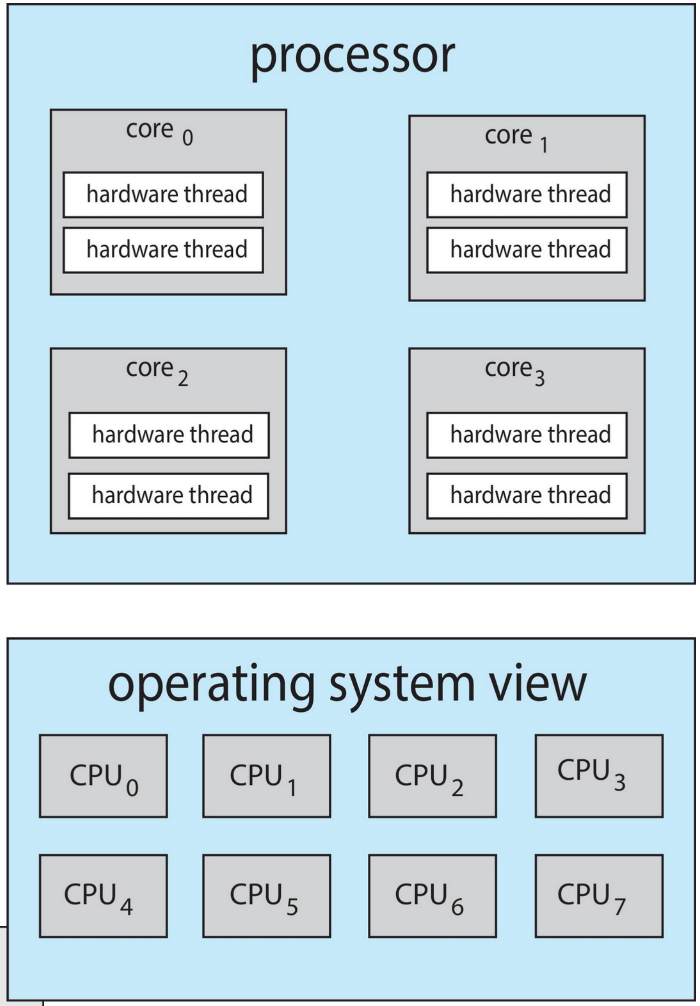
Multithreaded Multicore System

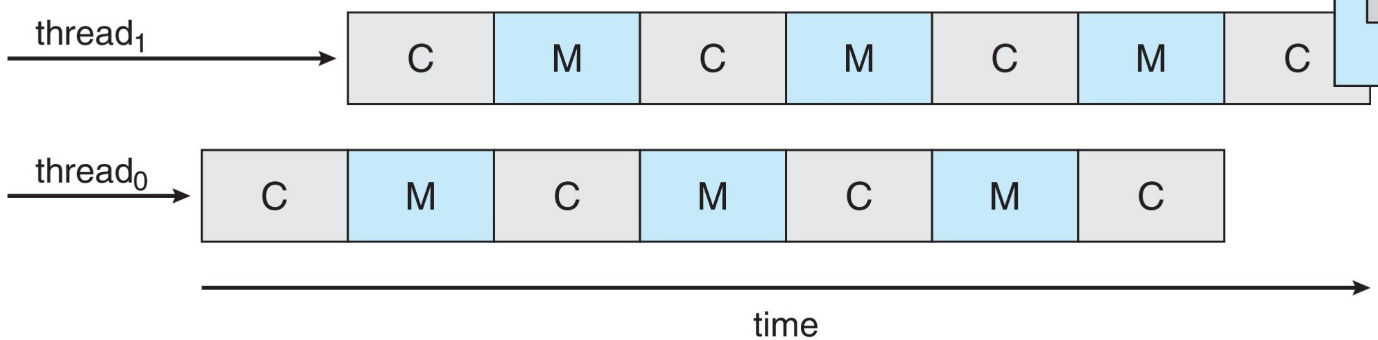
Each core has > 1 hardware threads.

Chip-multithreading (CMT) assigns each core multiple hardware threads. (Intel refers to this as **hyperthreading**.)

On a quad-core system with 2 hardware threads per core, the operating system sees 8 logical processors.

If one thread has a memory stall, switch to another thread!





More on SMP systems

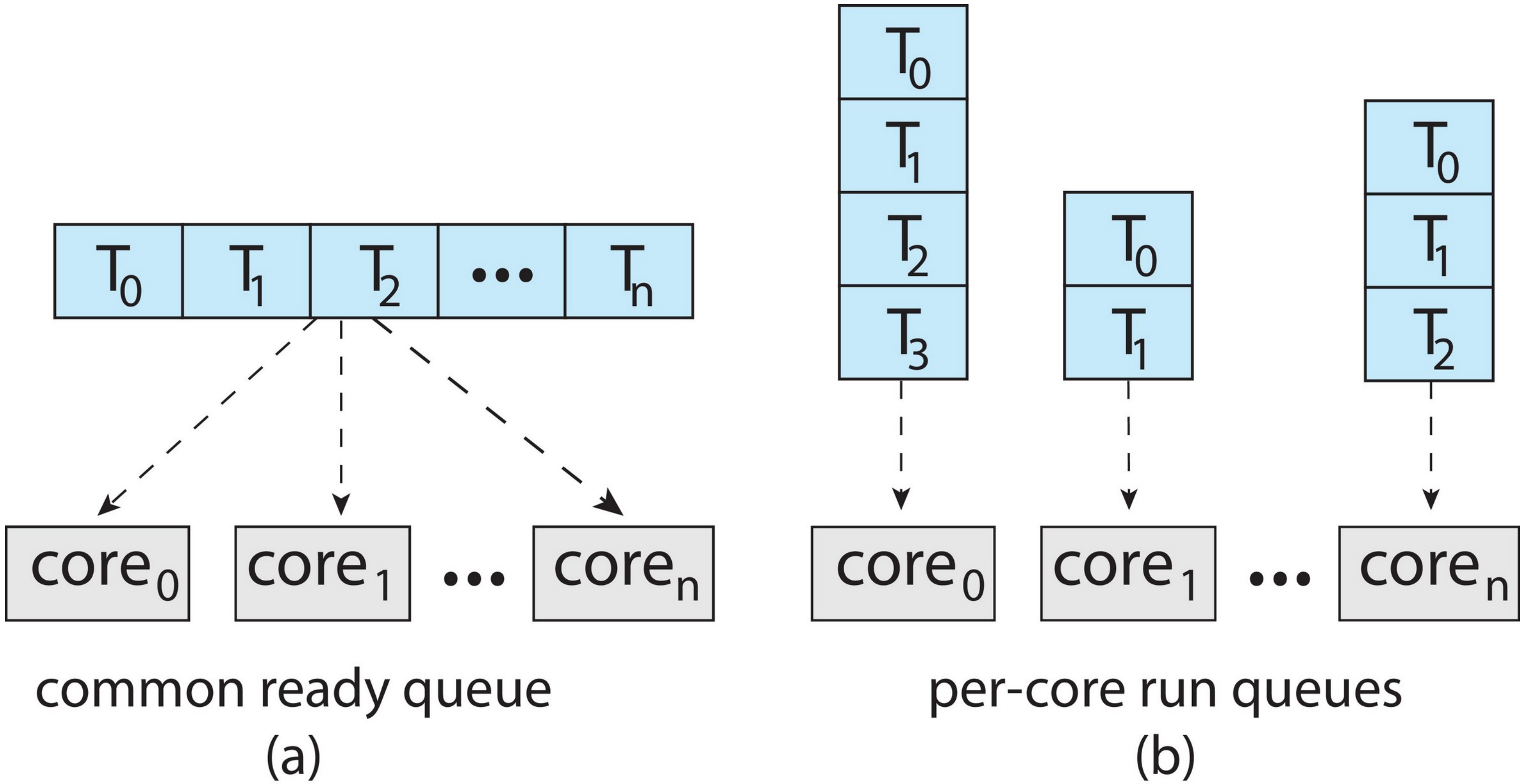
- During booting – each CPU needs to be turned on
  - Special I/O instructions writing to particular ports
  - See lapicstartap() in xv6
  - Need to setup CR3 on each processor
  - Segmentation, Page tables are shared (same memory for all CPUs)
- All processors will keep running independently of each other
- Different interrupts on each processor – each needs IDT setup
- Each processor will be running processes, interrupt handlers, syscalls
- Synchronization problems !
- How to do scheduling ?

Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- Multiprocess may be any one of the following architectures:
  - Multicore CPUs
  - Multithreaded cores
  - NUMA systems
  - Heterogeneous multiprocessing

Multiple-Processor Scheduling

- Symmetric multiprocessing (SMP) is where each processor is self scheduling.
- All threads may be in a common ready queue (a)
- Each processor may have its own private queue of threads (b)



SMP in xv6

- Only one process queue
- No load balancing, no affinity (more later)
- A process may run any CPU-burst /alloted-time-quantum on any processor randomly

End