CISC 3320 MW3

Thread and Multiprocessor Scheduling

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Acknowledgement

- These slides are a revision of the slides provided by the authors of the textbook via the publisher of the textbook.

Outline

- Thread Scheduling
- Multi-Processor Scheduling
  - Multicore CPUs, multithreaded cores, NUMA systems, heterogeneous multiprocessing
  - Memory stall, multithread Processor, and scheduling
  - Load balancing, processor affinity, and cache

Thread Scheduling

- Distinction between user-level and kernel-level threads
- When threads supported, threads scheduled, not processes
- User and kernel threads
  - One to one
  - Many to one
  - Many to many

User and Kernel Threads

- Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP (light-weight process)
  - Known as **process-contention scope (PCS)** since scheduling competition is within the process
  - Typically done via priority set by programmer
- Kernel thread scheduled onto available CPU is **system-contention scope (SCS)** – competition among all threads in system

Example: Pthread Scheduling

- API allows specifying either PCS or SCS during thread creation
  - PTHREAD_SCOPE_PROCESS schedules threads using PCS scheduling
  - PTHREAD_SCOPE_SYSTEM schedules threads using SCS scheduling
- Can be limited by OS
  - Linux supports only PTHREAD_SCOPE_SYSTEM
  - Open Solaris supports both before Solaris 9, but makes no distinction between the two since Solaris 9

Pthread Scheduling API

- `pthread_attr_getscope`
- `pthread_attr_setscope`

Questions?

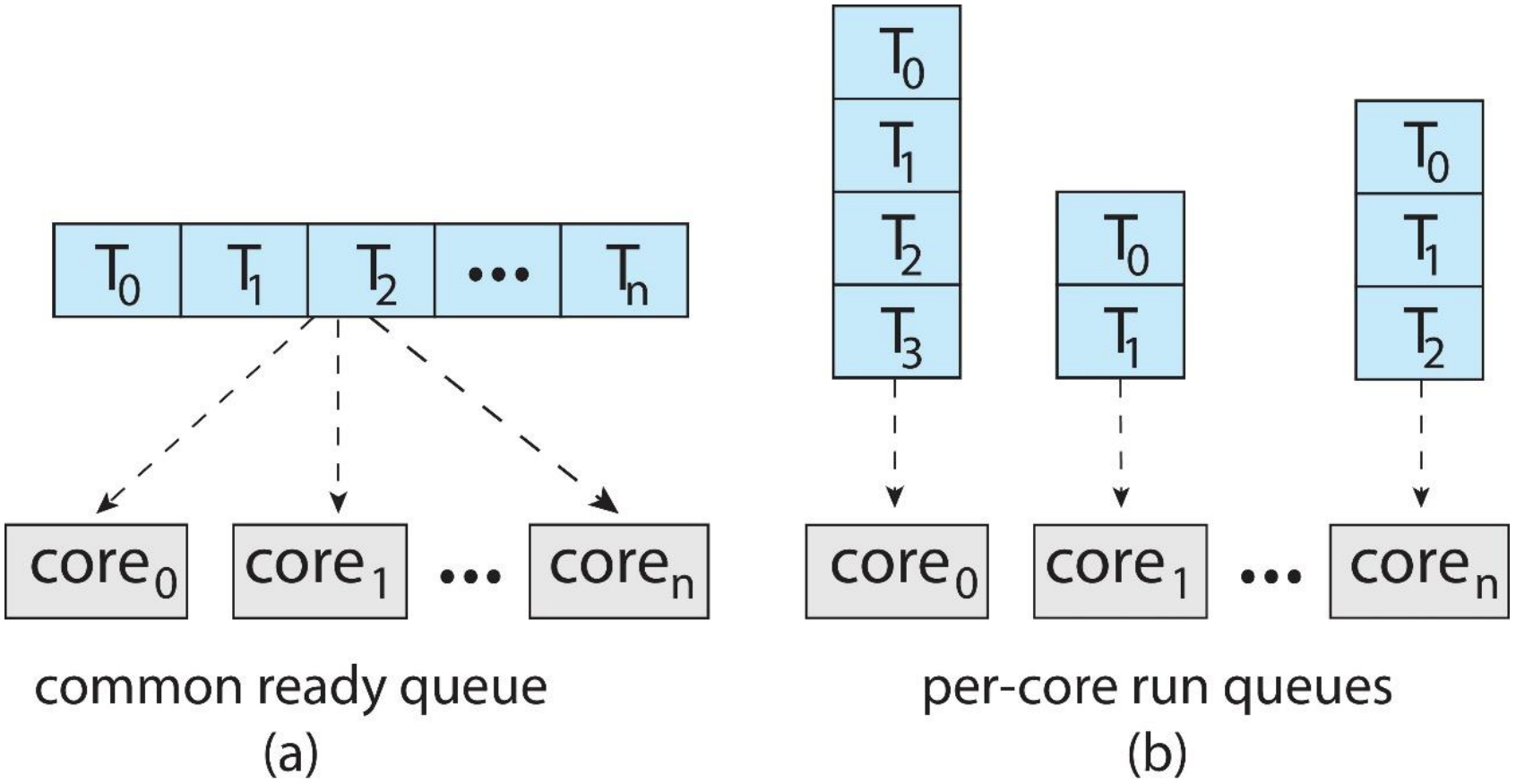
- Thread scheduling
- PCS and SCS
- Pthread example

Multiprocessor 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

Symmetric Multiprocessing

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


T0
T1
T2
...
Tn
core0
core1
...
coren
common ready queue
(a)
T0
T1
T2
T3
T0
T1
T0
T1
T2
core0
core1
...
coren
per-core run queues
(b)

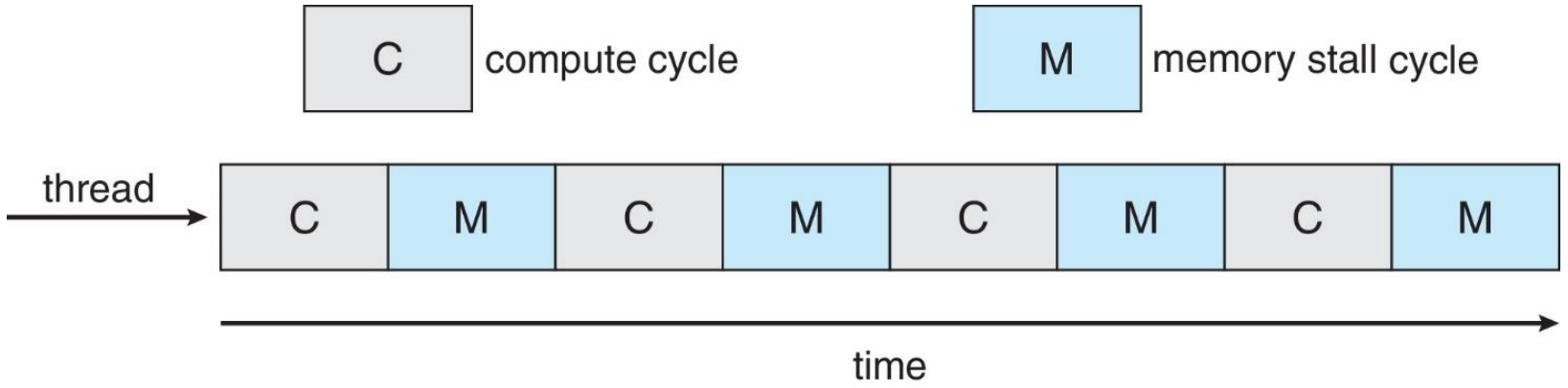
Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consumes less power

Memory Stall and Multithread Processor

- Memory stall
  - e.g., compare these two instructions
    - mov edx, eax
    - mov (1000), eax
  - Observation
    - memory is much slower than registers
  - In the second instruction above, the processor must wait significant amount of data for the data to be available

Memory Stall

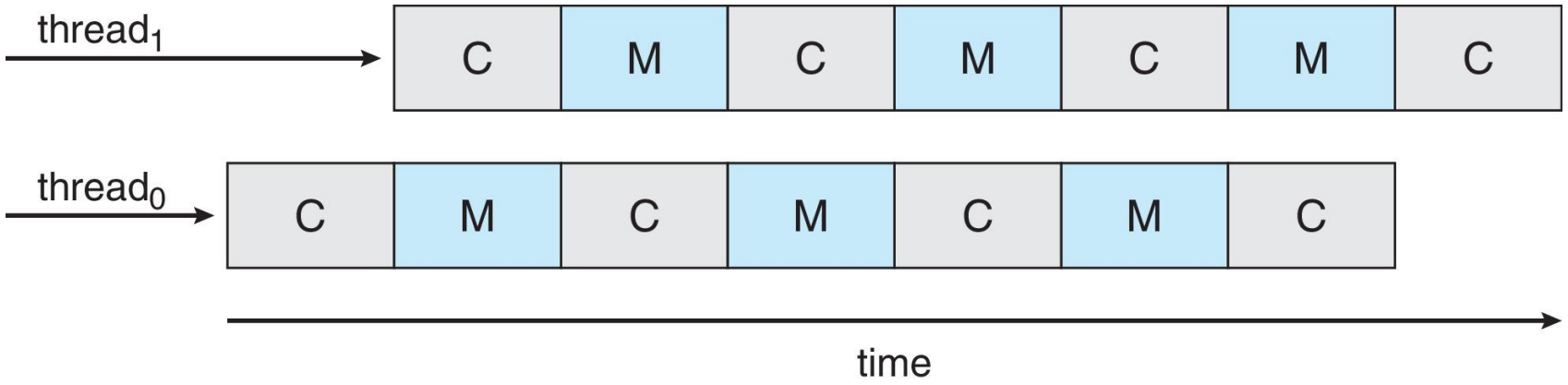


Addressing Memory Stall

- Takes advantage of memory stall to make progress on another (hardware) thread while memory retrieve happens
- Many recent hardware designs have implemented multithreaded processing cores
  - Two (or more) hardware threads are assigned to each core.
  - In this way, if one hardware thread stalls while waiting for memory, the core can switch to another thread.
- Called chip multithreading (CMT)
  - Intel calls it “hyperthreading”

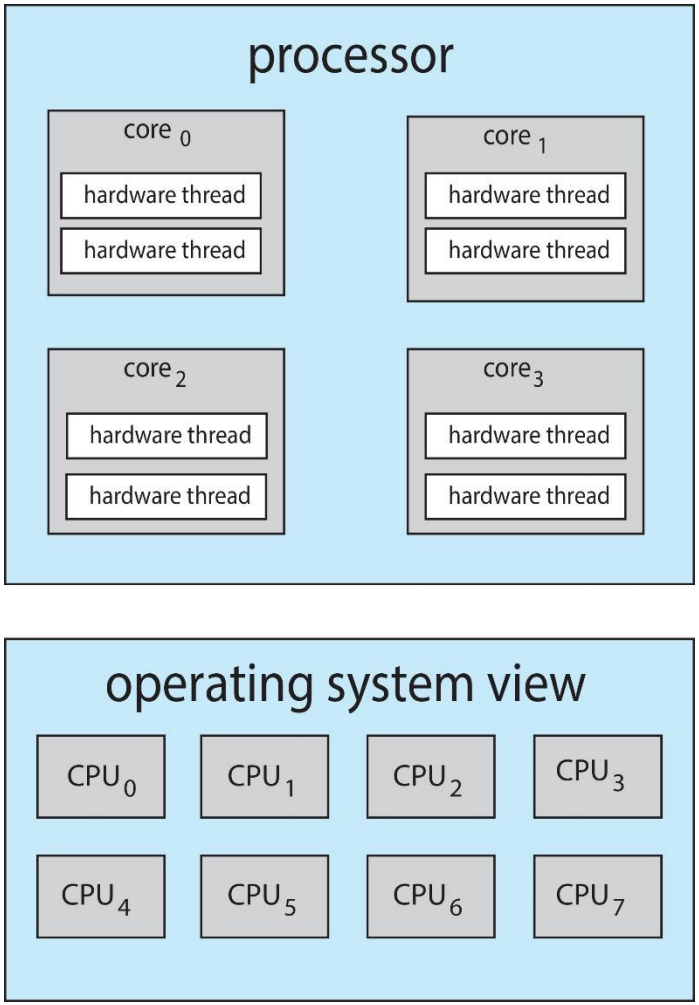
Chip Multithreading

- Each core has > 1 hardware threads.
- If one thread has a memory stall, switch to another thread!
- Each (hardware) thread appears to be a (logical) CPU to an operating system



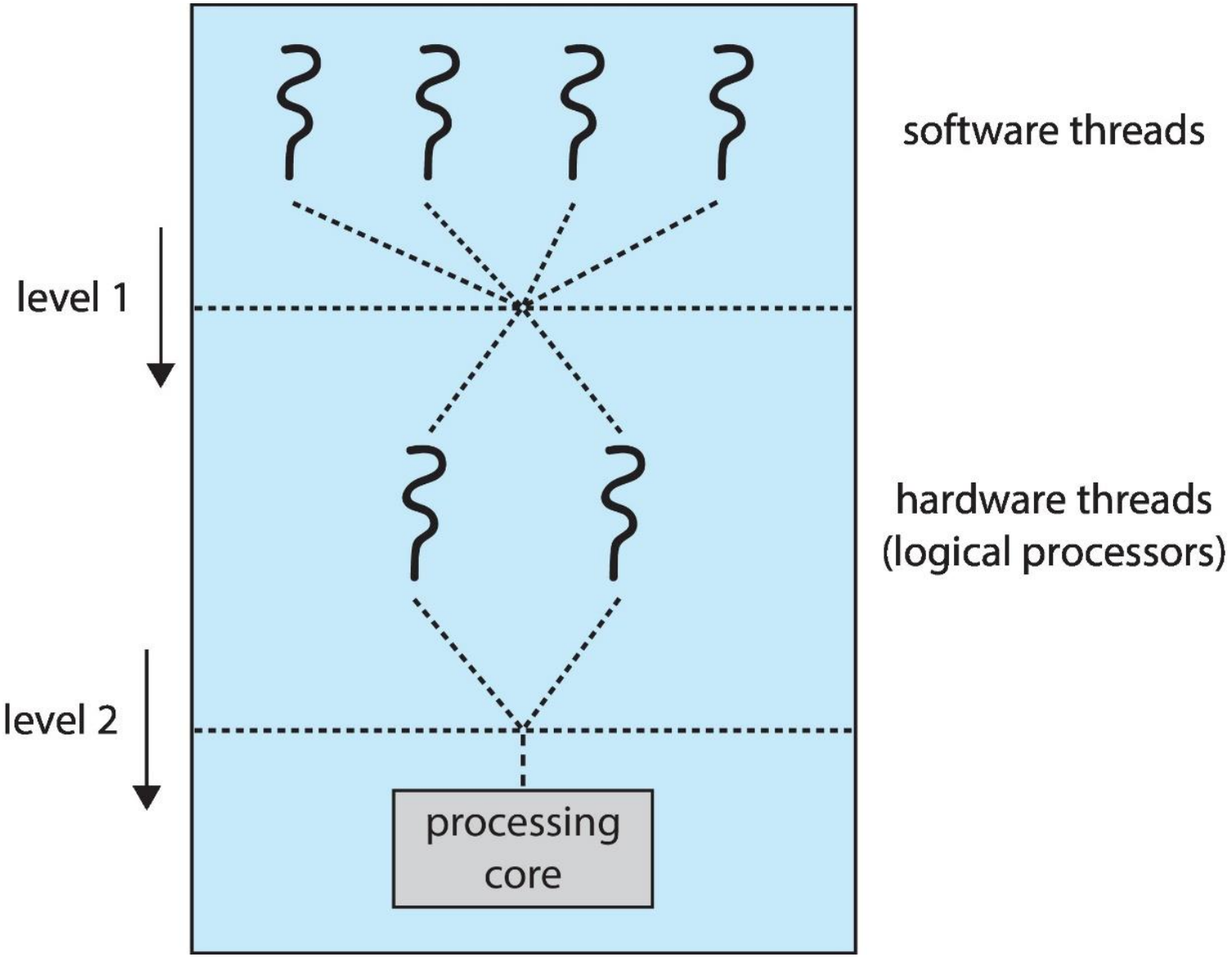
Multithreaded Multicore System: Example

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



Multithreaded Multicore System: Scheduling

- Two levels of scheduling:
  - The operating system deciding which software thread to run on a logical CPU
  - How each core decides which hardware thread to run on the physical core.


software threads
level 1
hardware threads
(logical processors)
level 2
processing
core

Load Balancing

- If SMP, need to keep all CPUs loaded for efficiency
- Load balancing attempts to keep workload evenly distributed
- Push migration
  - periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- Pull migration
  - idle processors pulls waiting task from busy processor

Processor Affinity

- A thread having affinity for a processor (i.e. “processor affinity”)
  - When a thread has been running on one processor, the cache contents of that processor stores the memory accesses by that thread.

Processor Cache and Scheduling

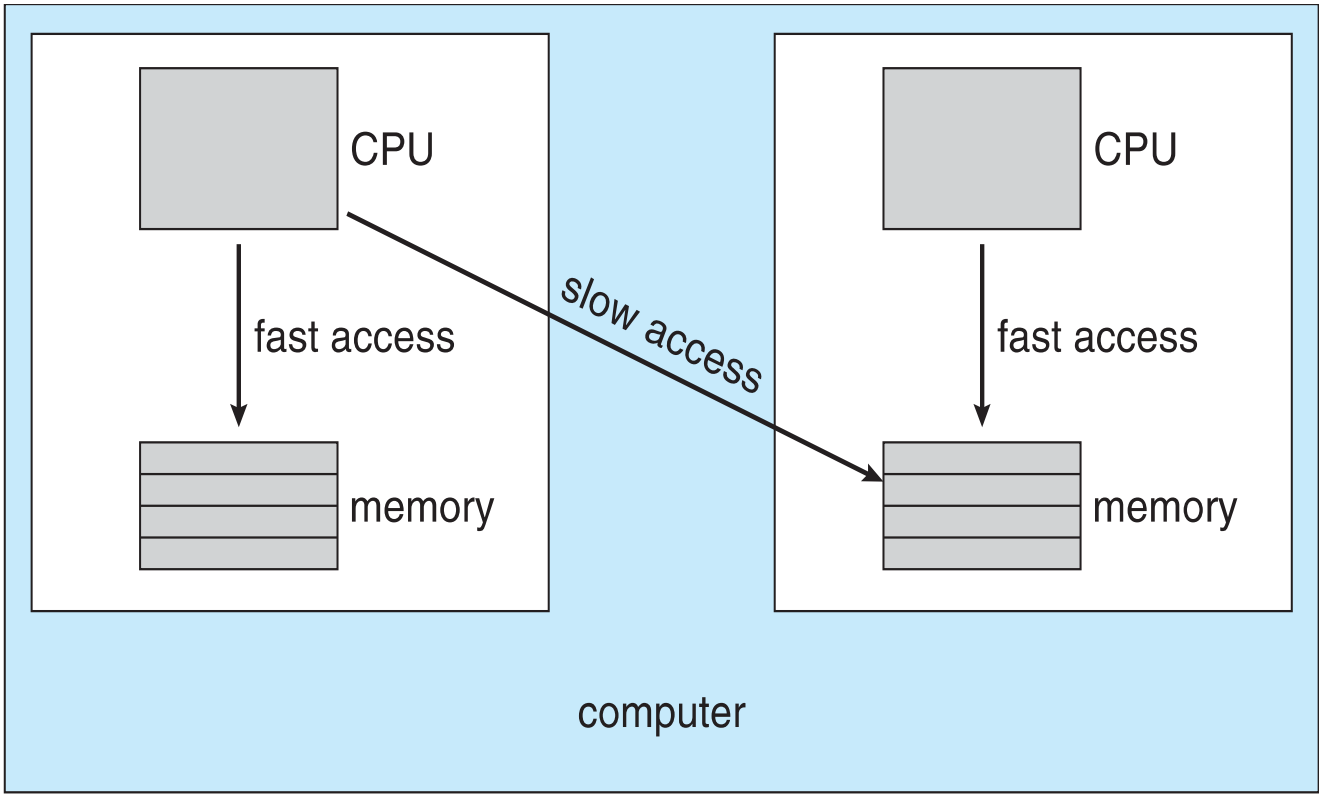
- If a thread is scheduled on a new processor, that processor's cache must be repopulated.
  - With private, per-processor ready queues, a thread is always scheduled on the same processor and can therefore benefit from the contents of a warm cache.
  - If the thread migrates to another processor, e.g., due to load balancing. The contents of cache memory must be invalidated for the first processor, and the cache for the second processor must be repopulated.

Setting Processor Affinity

- Soft affinity
  - the operating system attempts to keep a thread running on the same processor, but no guarantees.
- Hard affinity
  - allows a process to specify a set of processors it may run on.
- Example
  - Linux implements both soft affinity
  - The sched_setaffinity() system call supports hard affinity by allowing a thread to specify the set of CPUs on which it is eligible to run.

NUMA and CPU Scheduling

- Non-uniform memory access
  - Fast and slow memory access
- If the operating system is NUMA-aware, it will assign memory closes to the CPU the thread is running on.


CPU
CPU
fast access
slow access
fast access
memory
memory
computer

Questions?

- Multiprocessor scheduling
- Two level scheduling for chip multithreading
- Loading balancing
- Processor affinity, cache, and scheduling
- NUMA and scheduling

Heterogeneous Multiprocessing

- Symmetric multiprocessing (SMP)
  - All processors are identical in terms of their capabilities
- Heterogenous multiprocessing (HMP)
  - Although running the same instructors, processors may vary by their clock speed or power management

HMP Example

- ARM processor's big Little architecture
  - higher-performance big cores and many energy efficient LITTLE cores
  - Big cores consume greater energy and therefore should only be used for short periods of time.
  - Likewise, little cores use less energy and can therefore be used for longer periods.
- CPU scheduling should take these into consideration

Questions?

- Concept of HMP
- Scheduling for HMP