Operating System (OS)
CS232

Scheduling Algorithm: Multiprocessor Scheduling

Dr. Muhammad Mobeen Movania

Outlines

- Multiprocessor Architecture
- Background of cache, synchronization and cache affinity
- Types of Multiprocessor Scheduling
  - Single-Queue Multiprocessor Scheduling
  - Multi-Queue Multiprocessor Scheduling
- Examples of both schedulers
- Summary

Multiprocessor Architecture

- Single CPU and Multi-CPU hardware
- Use of **hardware cache** and memory shared between cores

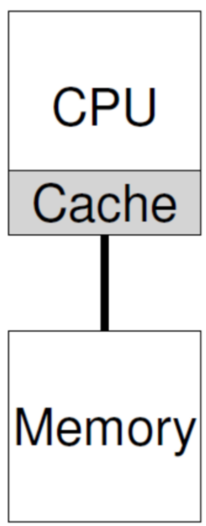


Figure 10.1: **Single CPU With Cache**

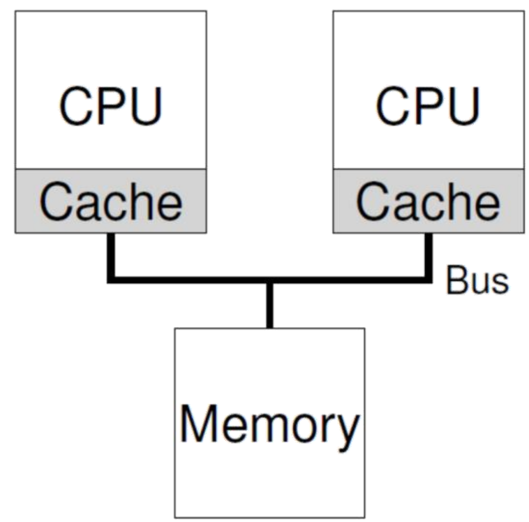


Figure 10.2: **Two CPUs With Caches Sharing Memory**

Why use cache?

- Usually there is a hierarchy of caches
  - Cache are small fast memories which contain copy of frequently access data from main memory
- Why
  - Access to data from cache is faster as compared to access from main memory
- Locality of data
  - Temporal locality (A piece of data accessed is likely to be accessed again in the near future e.g. iteration statements)
  - Spatial locality (A program accessing a data from address x will likely access data near x e.g. data in arrays
- Issues
  - Coherence (must be ensured so that all CPU caches and main memory has the last updated data)
  - Uses coherence protocols like directory or snooping protocols

Need of Synchronization

- When shared data is accessed from multiple processes or threads on differenct CPUs
  - Consistency of data must be ensured
  - Achieved through mutual exclusion primitives (like locks) or using lock-free data structures or atomic operations

Cache Affinity

- A process when run on a particular CPU, builds up a fair bit of state in the caches (and TLBs) of the CPU
- The next time the process runs, it is often advantageous to run it on the **same CPU**
- Why?
  - Program will run faster as its state is already available in cache/s
  - If run on a different CPU, the state will have to maintained again in cache/s

Types of Multiprocessor Scheduling

- Two types based on the number of queues used to schedule jobs
  - Single-queue Multiprocessor Scheduling (SQMS)
  - Multi-queue Multiprocessor Scheduling (MQMS)

Single-queue Multiprocessor Scheduling (SQMS)

- Key Idea:
  – Put all jobs that need to be scheduled into a single queue
- Advantages
  – Simple to implement, less overhead to maintain
- Issues
  – Synchronization issues
  – Non-scalable (cannot work on multiple CPUs and must use locking to enable on multiple CPUs)
  – Must be implemented to take benefit of cache affinity

Example - SQMS

- Example workload

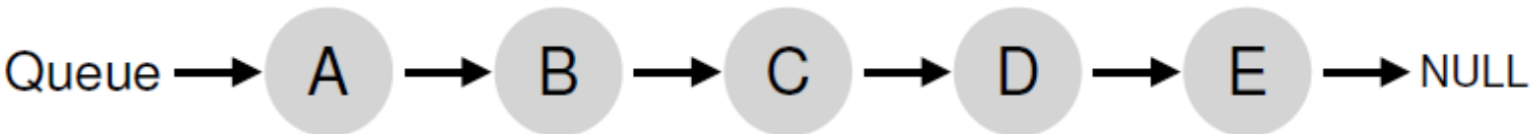
Queue → A → B → C → D → E → NULL

- Bad SQMS schedule (not using cache affinity)

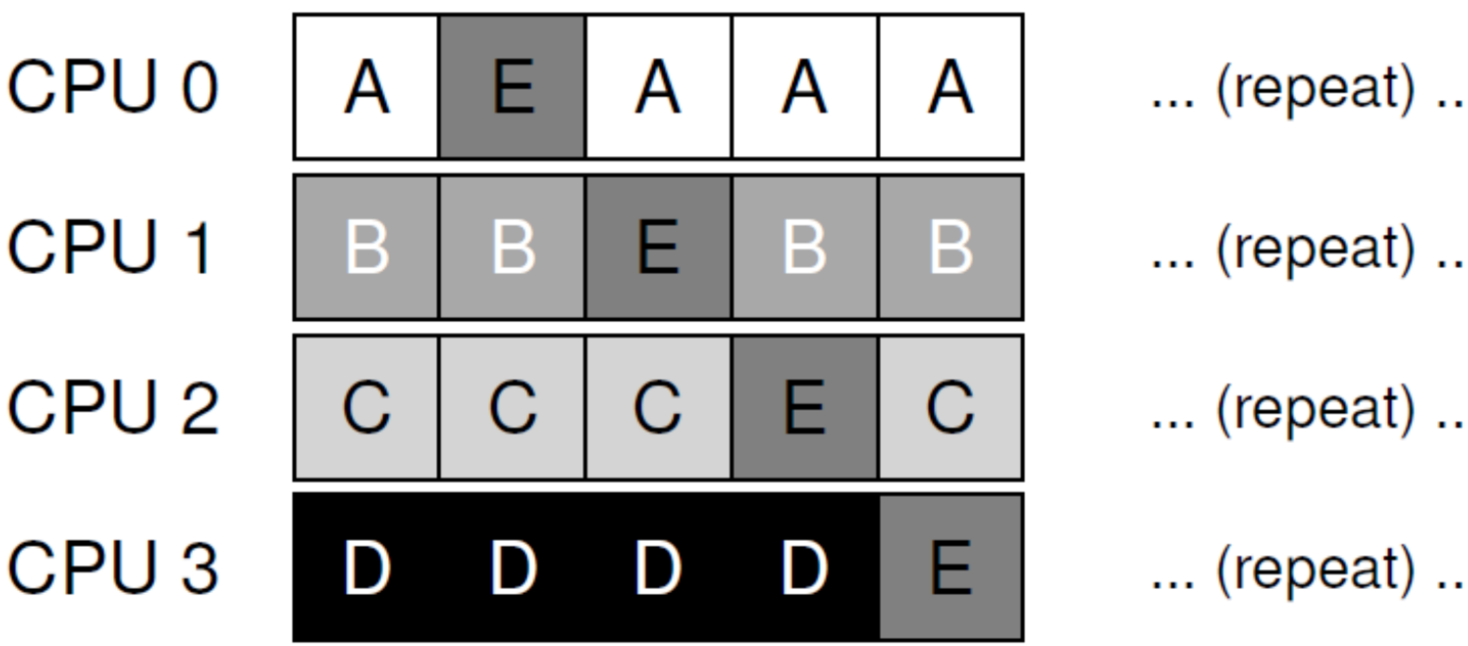
CPU 0	A	E	D	C	B	... (repeat) ...
CPU 1	B	A	E	D	C	... (repeat) ...
CPU 2	C	B	A	E	D	... (repeat) ...
CPU 3	D	C	B	A	E	... (repeat) ...

Example - SQMS

- Example workload



- Good SQMS schedule (using cache affinity)

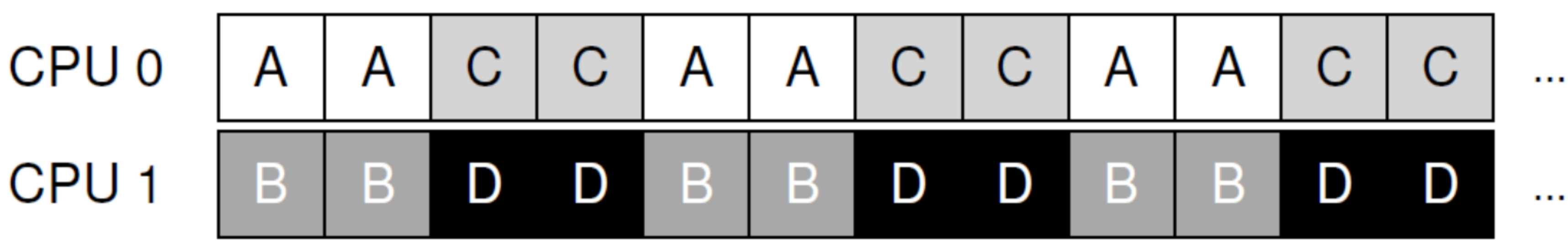


Multi-queue Multiprocessor Scheduling (MQMS)

- Key Idea:
  - Multiple scheduling queues with each following a particular scheduling discipline like RR etc.
  - A job is put in one scheduling queue using some heuristic
- Advantages
  - Each job is scheduled independently
  - No issues of synchronization
  - Intrinsically provide cache affinity
- Issues
  - Load imbalance (might be solved through migration)

Example - MQMS

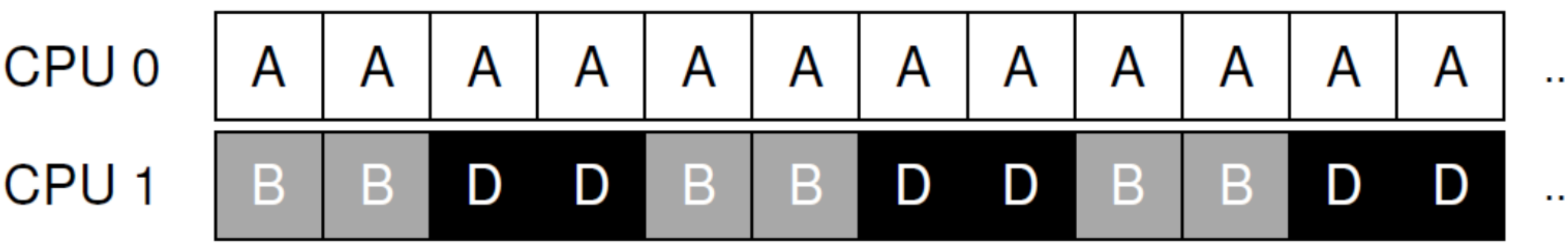
- Example workload



Example – MQMS (Load Imbalance 1)

- Example workload

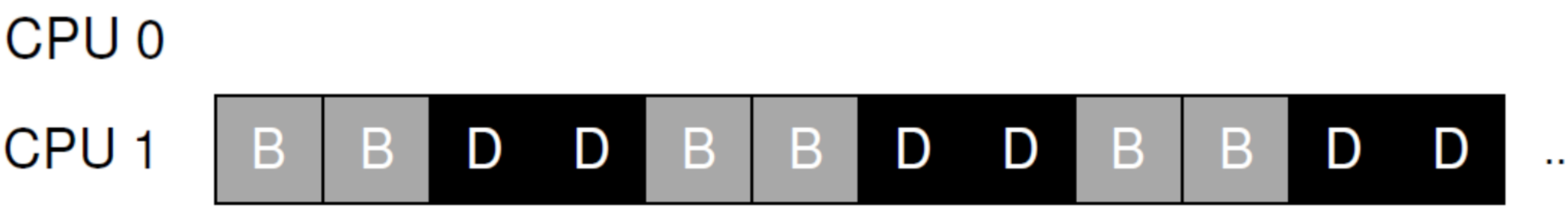
- Issue
  - Process A gets twice as much CPU time as B and D



Example – MQMS (Load Imbalance 2)

- Example workload

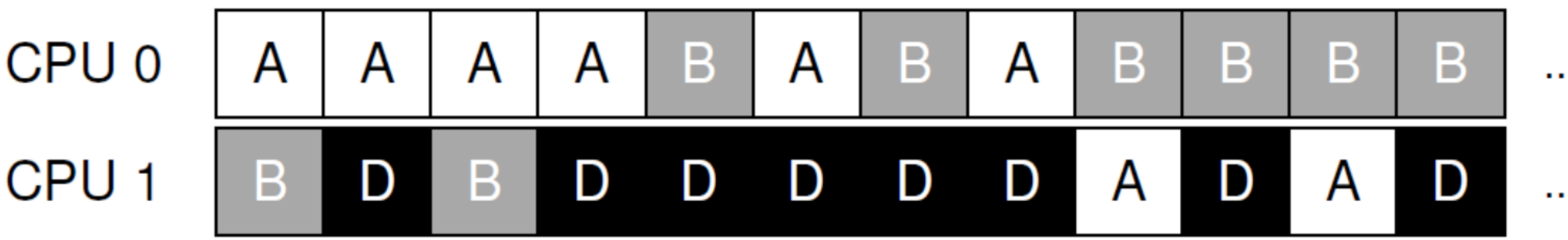
- Issue
  - CPU 0 is idle ☹



Example – MQMS
(Solving Load Imbalance 1 through Migration)

- Example workload

- Resolution
  - Process B is moved on CPU 0 so load is balanced



Summary

- We introduced two approaches of multiprocessor scheduling
  - Single-Queue Multiprocessor Scheduling (SQMS)
  - Multi-Queue Multiprocessor Scheduling (MQMS)
- SQMS is easy to build and load balance but tough to scale and cannot use cache affinity
- MQMS scales well and uses cache affinity but has to be load balanced to properly use CPUs
- Building a general purpose scheduler is a daunting task.