Universidad
Carlos III de Madrid
www.uc3m.es

ARCOS

COMPUTER ARCHITECTURE

Symmetric Shared Memory

Universidad Carlos III de Madrid
www.uc3m.es

Contents

ARCOS

- **Introduction to multiprocessor architectures.**
- Centralized shared memory architectures.
- Cache coherence alternatives.
- Snooping protocol.
- Performance in SMP.

Multiprocessors importance growth

- Decrease in silicon use and energy efficiency as more **ILP** is exploited.
  - Cost of silicon and energy grows faster than performance.
- Increasing interest in **high performance servers**.
  - Cloud computing, software as a service.
- Data intensive applications growth.
  - Huge amounts of data on the Internet.

TLP: Thread Level Parallelism

- **TLP** implies existence of multiple program counters.
  - Assumes **MIMD**.
  - Generalized use of **TLP** outside scientific computing is relatively recent.
  - New applications:
    - Embedded applications.
    - Desktop.
    - High-end servers.

Multiprocessors

- A **multiprocessor** is a computer consisting of highly coupled processors with:
  - **Coordination** and use typically controlled by a single operating system.
  - **Sharing** memory through a single shared address space.

- **Software models**:
  - Parallel processing: Coupled set of threads cooperating.
  - Request processing: Executing independent processes originated by users.
  - Multiprogramming: Independent execution of multiple applications.

Universidad Carlos III de Madrid
www.uc3m.es

Multiprocessors

ARCOS

- Most common approximation:
  - From 2 to dozens of processors.
  - **Memory**.
    - Implies shared memory.
    - Not necessarily implies single physical memory.
- **Alternatives**:
  - CMP (Chip Multiprocessor) or multicore.
  - Multiple chips.
    - Each may (or may not) be multicore.
  - Multicomputer: Weakly coupled processor not sharing memory.
    - Used in large scale scientific computing.

Multiprocessors architectures

- Maximizing exploitation of multiprocessors:
  - With **n** processors, at least **n** processors are needed.

- **Threads identification**:
  - Explicitly identified by the programmer.
  - Created by operating system from requests.
  - Loop iterations generated by a parallel compiler.

High-level identification performed by programmer or system software with threads having **enough** number of instructions to execute.

Multiprocessors and shared memory

SMP

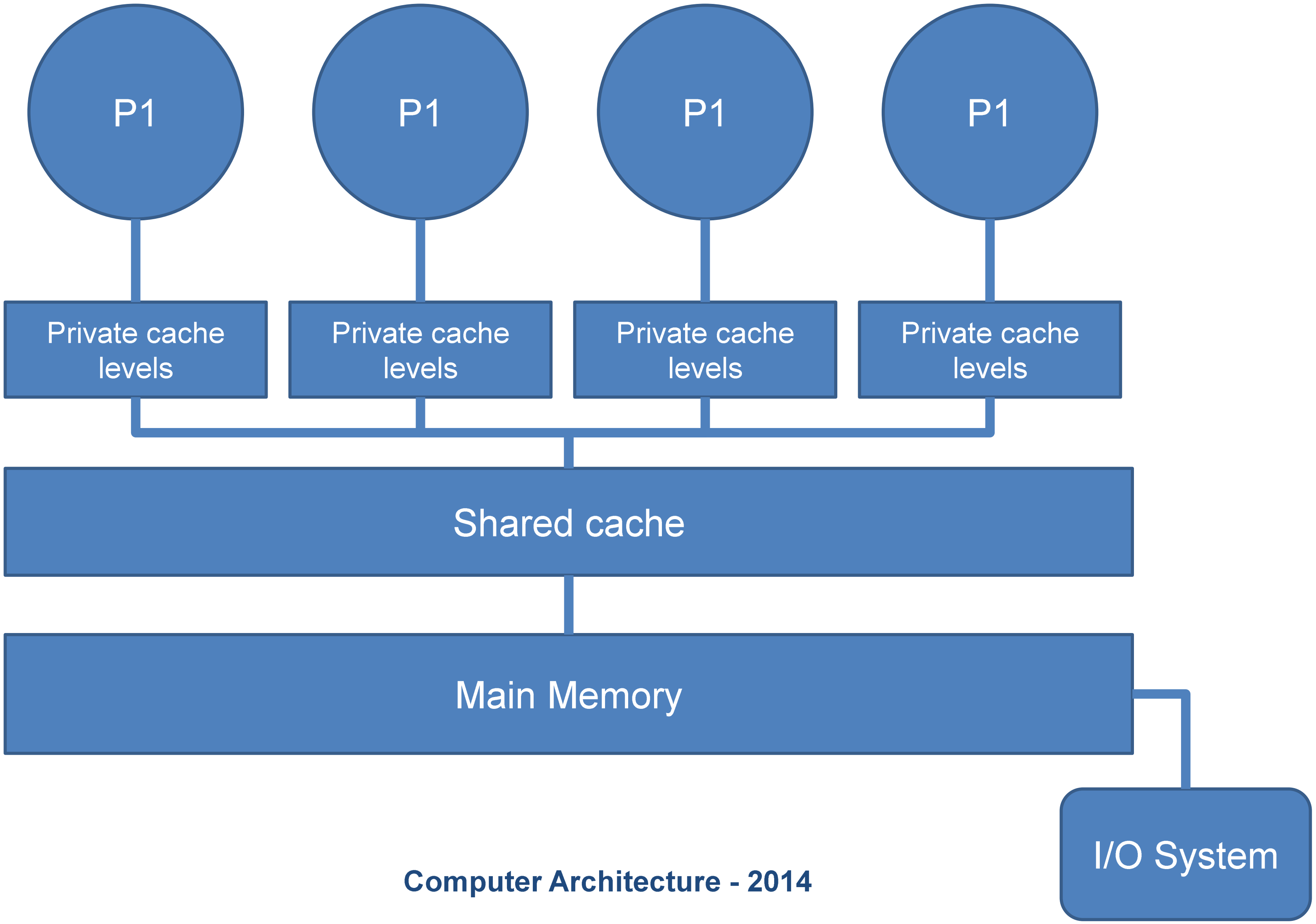
- Symmetric Multiprocessors
- Share a single centralized memory where all have equal access.
- All multi-cores are **SMPs**.
- **UMA**: Uniform Memory Access.
  - Memory latency is uniform.

DSM

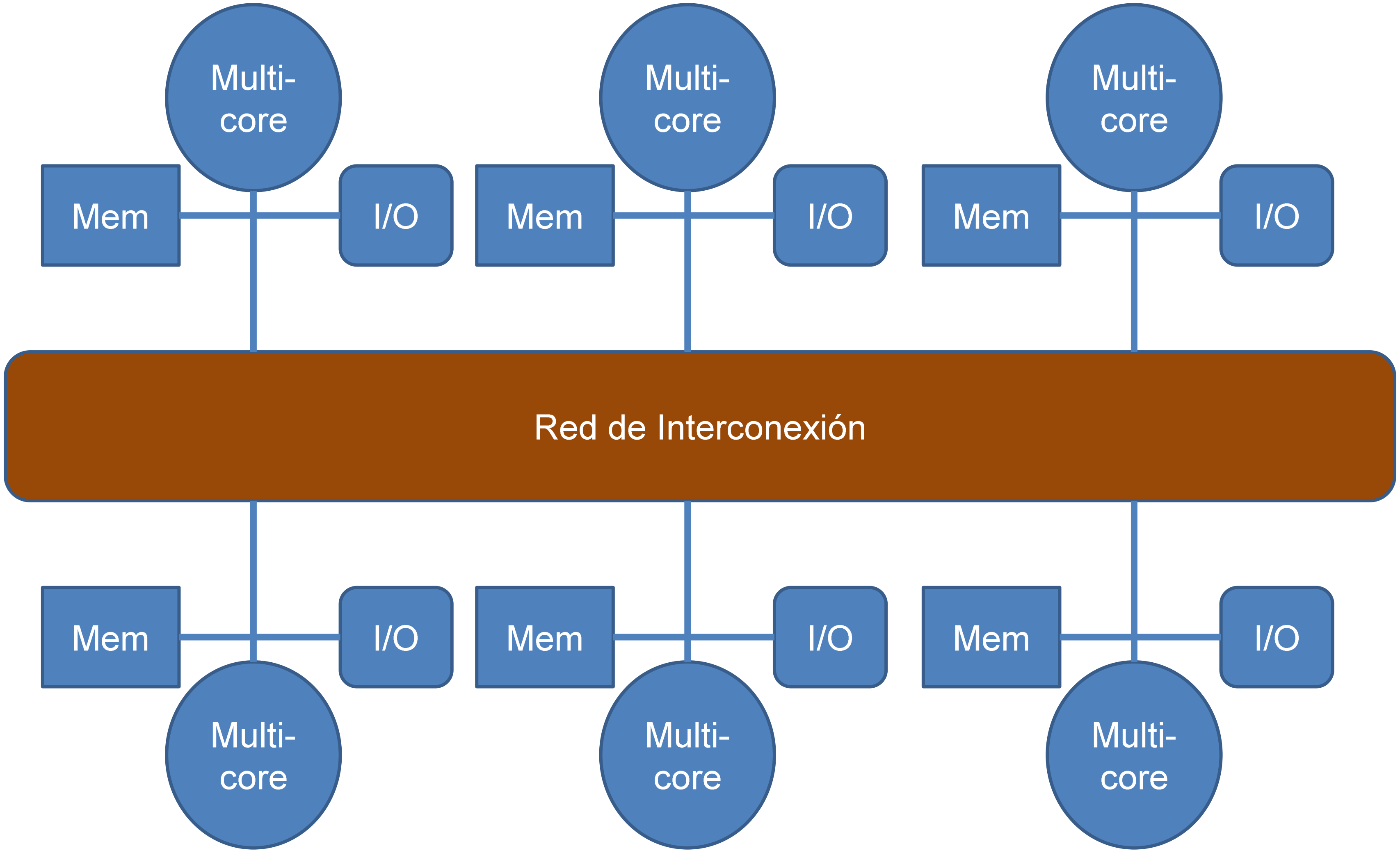
- Distributed Shared Memory
- Memory distributed across processors.
- Needed when number of processors high.
- Multiprocessors win multiple multicores -> **DSM**.
- **NUMA**: Non Uniform Memory Access.
  - Latency depends on accessed datum location.

Inter-thread communication through accessing globally shared memory locations.

SMP


P1
P1
P1
P1
Private cache levels
Private cache levels
Private cache levels
Private cache levels
Shared cache
Main Memory
I/O System

DSM


Multi-core
Mem
I/O
Multi-core
Mem
I/O
Multi-core
Mem
I/O
Red de Interconexión
Mem
I/O
Multi-core
Mem
I/O
Multi-core
Mem
I/O
Multi-core

Universidad Carlos III de Madrid
www.uc3m.es

Contents

ARCOS

SMP

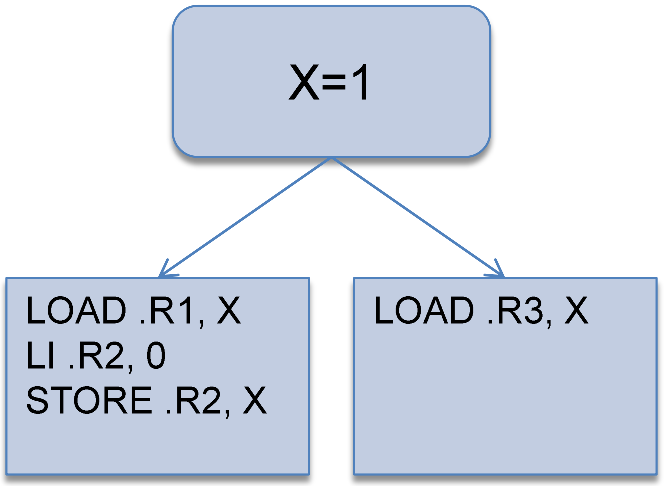
- Why?
  - Multi-level large caches reduce memory bandwidth demand.
- Evolution:
  - Single-core with memory in shared bus.
  - Connection to memory in separate bus only for memory.

SMP and cache memory

- **Types of data** in cache:
  - **Private data**: Data used by a single processor.
  - **Shared data**: Data used by multiple processors.

- **Problems** with **shared** data:
  - Datum must be replicated in multiple caches.
  - Contention is reduced.
    - Each processor accesses tis local copy.
  - If two processors modify their copies…
    - Cache coherence?

Cache coherence



Assuming write-through

Process	Instruction	P1 cache (address X)	P2 cache (address X)	Main memory (address X)
P1	Initially	Not present	Not present	1
P1	LOAD .R1, X	1	Not present	1
P1	LI .R2, 0	1	Not present	1
P2	LOAD .R3, X	1	1	1
P1	STORE .R2, X	0	1	0

Cache incoherence

- Why does incoherence happen?
  - State duality:
    - Global state → Main memory.
    - Local state → Private cache.

- A memory system is **coherent** if any read from a location returns the most recent value that has been written to that location.

- Two aspects:
  - **Coherence**: Which value does the read return?
  - **Consistency**: When does a read gets the written value?

Conditions for coherence

- **Program order preservation:**
  - A read by processor P from location X after a write by processor P to location X, without intermediate writes by any other processor, always returns the value written by P.
- **Coherent view of memory:**
  - A read by a processor from location X after a write by any other processor to location X, returns the written value if both operations are separated enough in time and there are no intermediate writes on X.
- **Write serialization:**
  - Two writes to the same location by two processors are seen in the same order by all the other processors.

Memory consistency

- Defines the point in time when a reading process will see a write.

- Coherence and consistency are complementary:
  - **Coherence**: Behavior of reads and writes to a single memory location.
  - **Consistency**: Behavior of reads and writes with respect to accesses to other memory locations.

- There are multiple consistency memory models.
  - **We will have a specific session for this problem.**

Universidad Carlos III de Madrid
www.uc3m.es

Contents

ARCOS

Properties with coherent multiprocessors

- A coherent multiprocessor offers:
  - Shared data migration.
    - A datum may move to a local cache and be used in a transparent way.
    - Reduces access latency to remote data and shared memory bandwidth demand.
  - Shared data replication simultaneously read.
    - Performs data copy in local cache.
    - Reduces access latency and read contention.
- Critical properties for performance:
  - **Solution**: Hardware protocol for keeping cache coherence.

Classes of cache coherence protocols

- **Directory based**:
  - Sharing state is kept in a directory.
  - SMP: Centralized directory in memory or in last level cache (LLC).
  - DSM: To avoid bottlenecks distributed directory is used (additional complexity).

- **Snooping**:
  - Each cache keeps sharing state for each block.
  - Caches are accessible through a broadcasting medium (bus).
  - All caches monitor broadcasting medium to determine if they have a copy of the block.

Contents

Strategies for cache coherence maintenance

- **Write invalidation**:
  - Guarantees that a processor has exclusive access to a block before performing a write.
  - Invalidates rest of copies that could be in other processors.
- **Write updates** (aka write broadcast):
  - Broadcasts all writes to all caches to modify a block.
  - Uses more bandwidth.

Invalidation is mot common strategy

Memory bus use

- Invalidation.
  - Processor acquires bus and broadcasts address to be invalidated.
  - All processors snoop the bus.
  - Each processor checks if they have the broadcasted address and invalidate it.
- There cannot be two simultaneous writes:
  - Exclusive use of bus serializes writes.
- Cache misses:
  - Write through:
    - Memory has the latest write.
  - Write back:
    - If a processor has a modified copy, answers cache miss to the other processor.

Implementation

- **Invalidation**:
  - Takes advantage of validity bit (V) associated to each block.
- **Writes**:
  - Need to know if there are other copies in cache.
    - If there are not other copies, the write is not broadcasted.
  - Sharing bit (S) added associated to each block.
  - When there is a write:
    - Bus invalidation generated.
    - Transition from shared state to exclusive state.
    - No need to send new invalidations.
  - When cache miss in other processor:
    - Transition from exclusive state to shared.

Universidad Carlos III de Madrid
www.uc3m.es

Basic Protocol

ARCOS

- Based in state machine for each cache block:
  - State changes generated by:
    - Processor requests.
    - Bus requests.
  - Actions:
    - State transitions.
    - Actions on the bus.
- Simple approximation with three states:
  - **M**: Block has been modified.
  - **S**: Block is shared.
  - **I**: Block has been invalidated.

Protocolo para acciones generadas por procesador

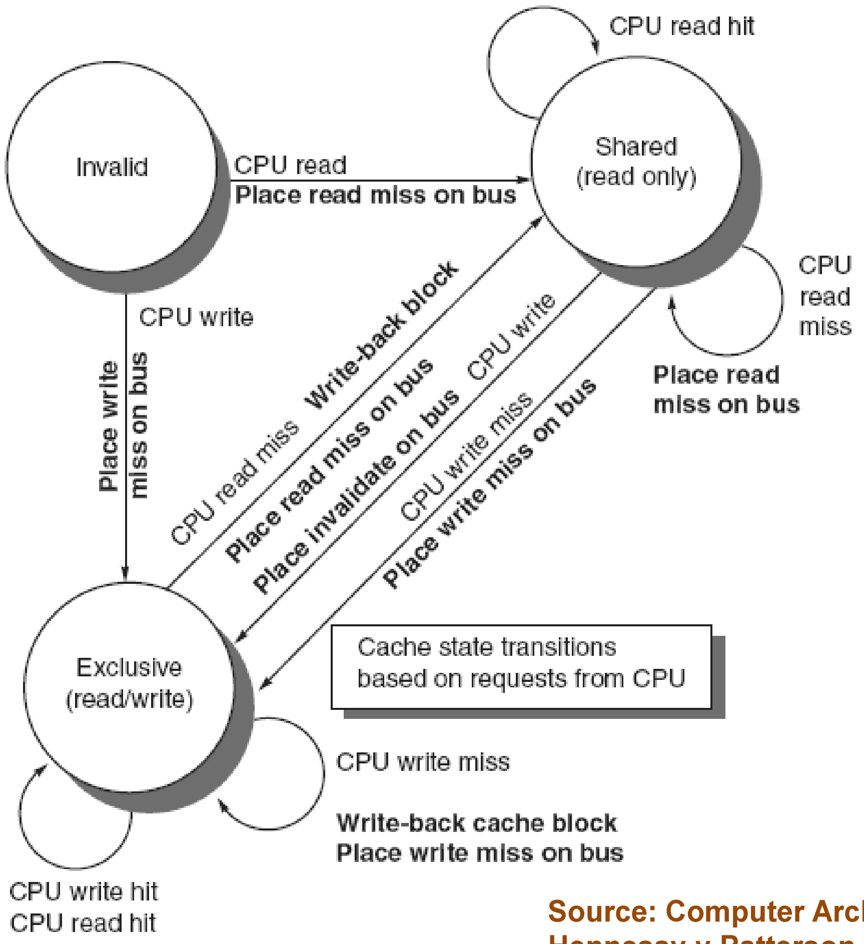
Request	State	Action type	Description
Read hit	S o M	Hit	Read data in local cache.
Read miss	I	Miss	Place read miss on bus.
Read miss	S	Replacement	Address conflict miss. Place read miss on bus.
Read miss	M	Replacement	Address conflict miss. Write back block, place read miss on bus
Write hit	M	Hit	Write data in local cache.
Write hit	S	Coherence	Place invalidate on bus.
Write miss	I	Miss	Place write miss on bus.
Write miss	S	Replacement	Address conflict miss. Place write miss on bus.
Write miss	M	Replacement	Address conflict miss. Write back block, place write miss on bus.

Protocolo de acciones generadas por bus

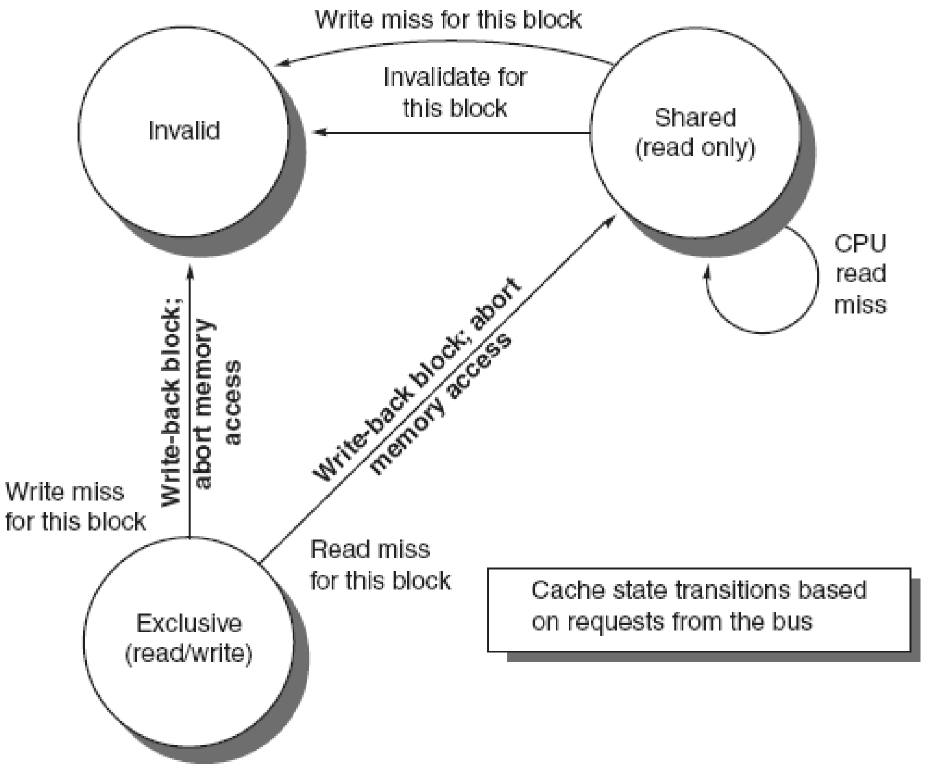
Request	State	Action type	Description
Read miss	S	-	Shared cache or memory service miss
Read miss	M	Coherence	Attempt to share data. Place cache block on bus and transition to shared.
Invalidate	S	Coherence	Attempt to write shared block. Invalidate block.
Write miss	S	Coherence	Attempt to write shared block. Invalidate block.
Write miss	M	Coherence	Attempt to write block that is exclusive elsewhere. Write-back cache block and make state invalid.

Universidad Carlos III de Madrid
www.uc3m.es

Protocol



Cache state transitions based on requests from CPU



Write miss for this block
Invalidate for this block
Invalid
Shared (read only)
CPU read miss
Write-back block; abort memory access
Write-back block; abort memory access
Write miss for this block
Read miss for this block
Exclusive (read/write)

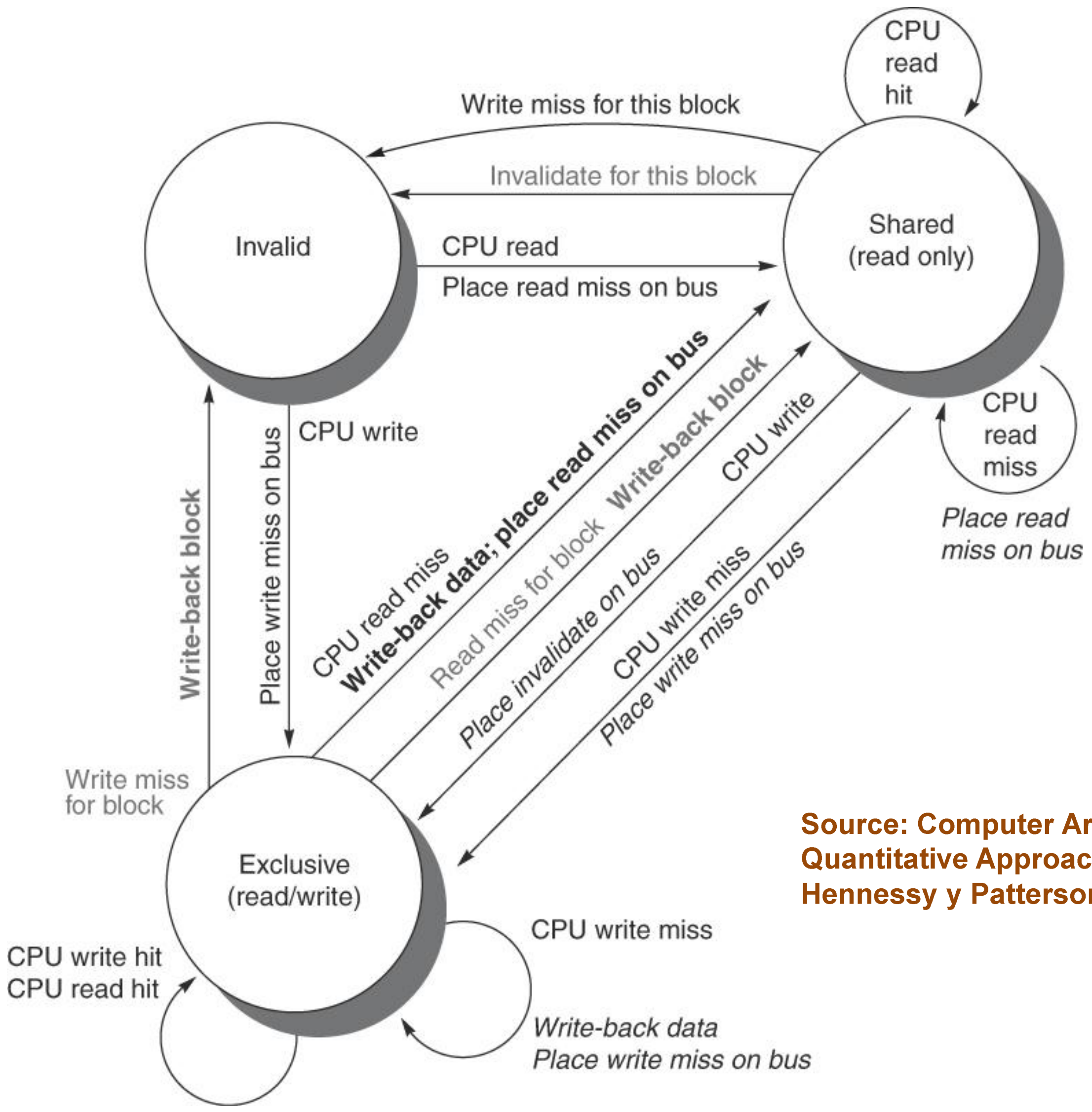
Cache state transitions based on requests from the bus

Source: Computer Architecture. A Quantitative Approach. Fifth Edition. Hennessy y Patterson.

Universidad Carlos III de Madrid
www.uc3m.es

Protocol

ARCOS



Source: Computer Architecture. A Quantitative Approach. Fifth Edition. Hennessy y Patterson.

MSI Protocol complexities

- Protocol assuming atomic operations.
  - Example: It is assumed that a write miss can be detected, bus acquired, and response received in a single uninterrupted action.
- If operations are not atomic:
  - Possibility of deadlock and/or race conditions.
- **Solution**:
  - Processor sending invalidation keeps bus ownership until invalidation arrives rest of processors.

Universidad Carlos III de Madrid
www.uc3m.es

Extension to MSI

- **MESI**
  - Adds exclusive state (E) signaling that block is in a single cache but is not modified.
  - Write of an E block des not generate invalidations.

- **MESIF**
  - Adds Forward (F):
    - Alternative to S signaling which node must answer to a request.
  - Used in Intel Core i7.

- **MOESI**
  - Add owned state (O) signaling that a block is not updated in memory.
  - Avoids memory writes.
  - Used in AMD Opteron.

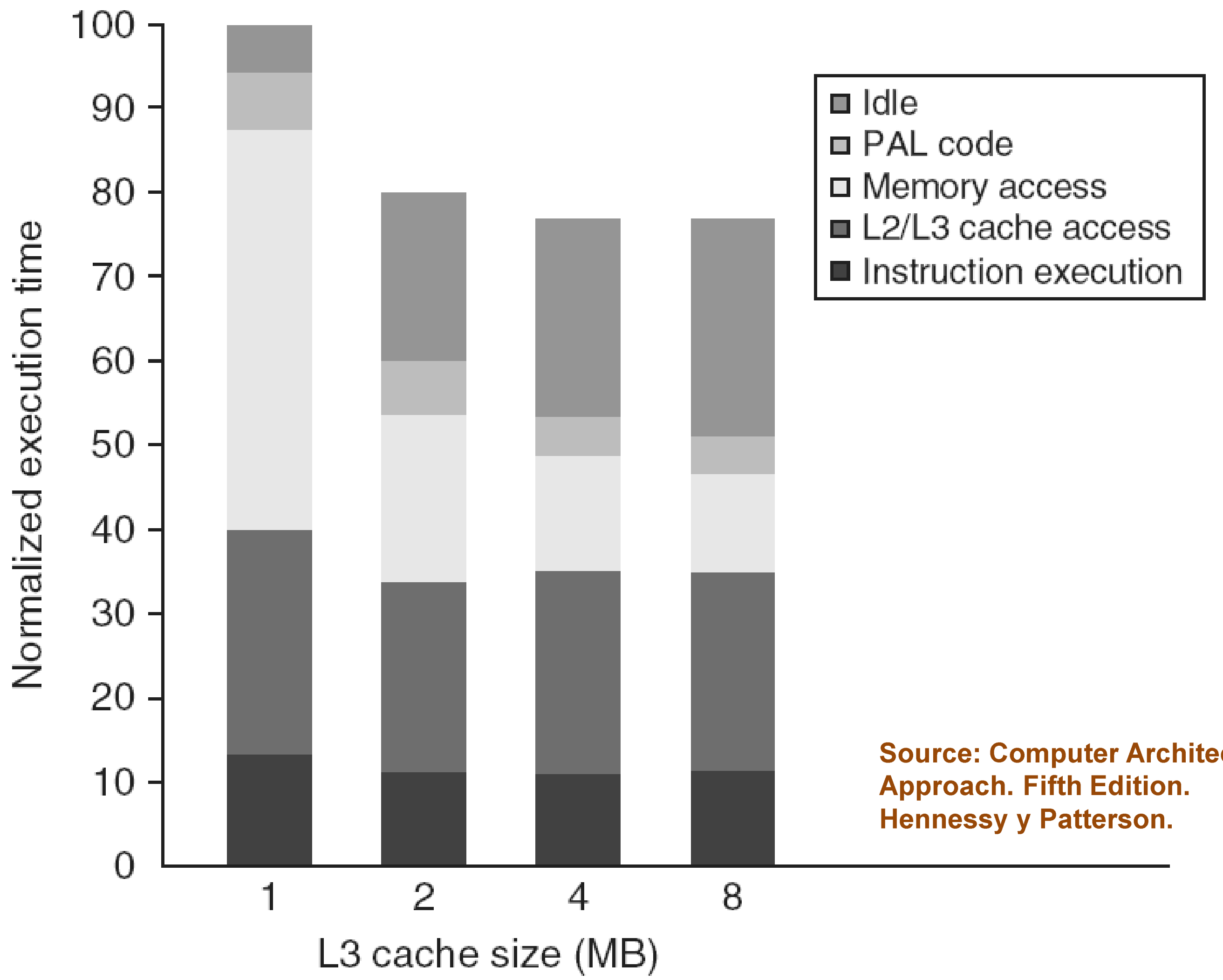
Contents

Performance

- Use of cache coherence policies has impact on miss rate.

- Coherence misses emerge.
  - True sharing misses:
    - A processor writes a shared block and invalidates.
    - Another processor reads the shared block.
  - False sharing misses:
    - A processor writes a shared block and invalidates.
    - Another processor reads a different word in the same block.

Relative performance as L3 size increases

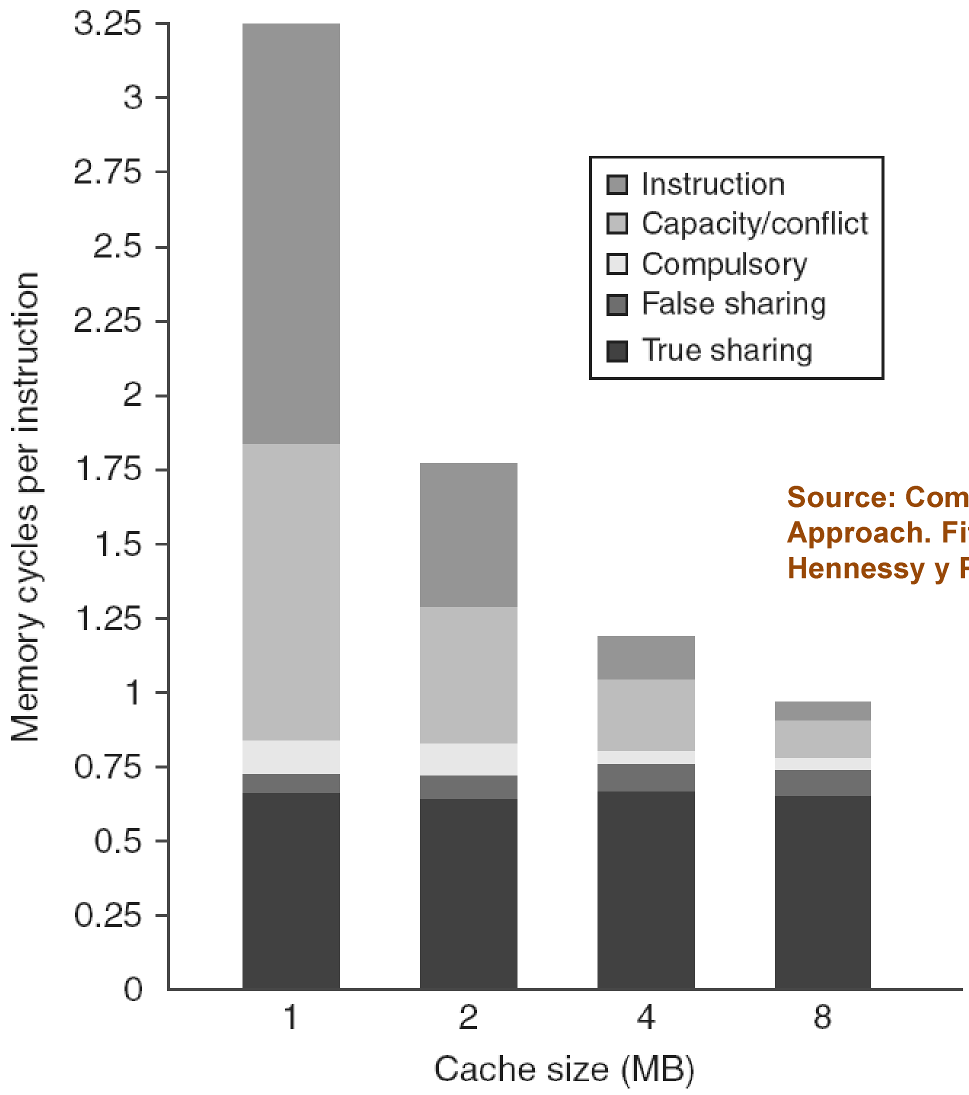


Source: Computer Architecture. A Quantitative Approach. Fifth Edition. Hennessy y Patterson.

Universidad Carlos III de Madrid
www.uc3m.es

Contributions to L3 misses

ARCOS

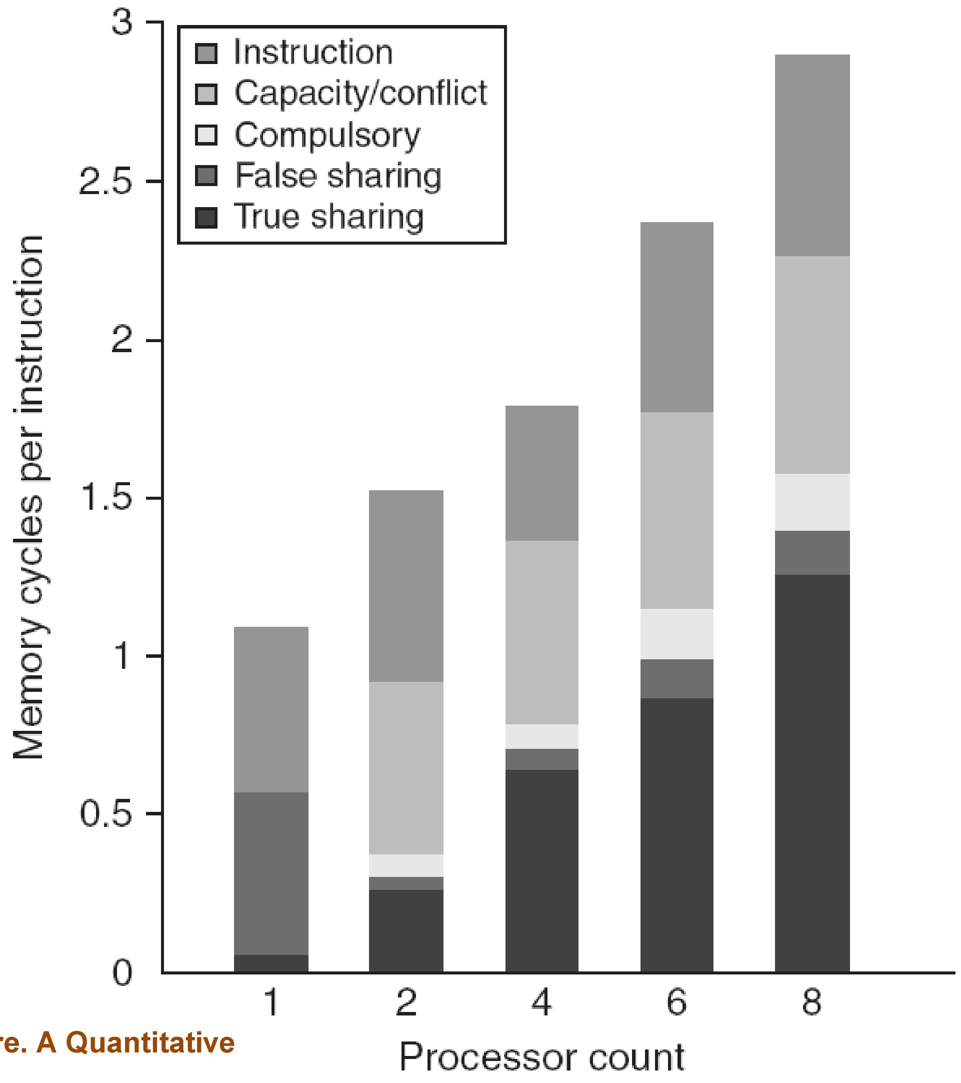


Source: Computer Architecture. A Quantitative Approach. Fifth Edition.
Hennessy y Patterson.

Universidad Carlos III de Madrid
www.uc3m.es

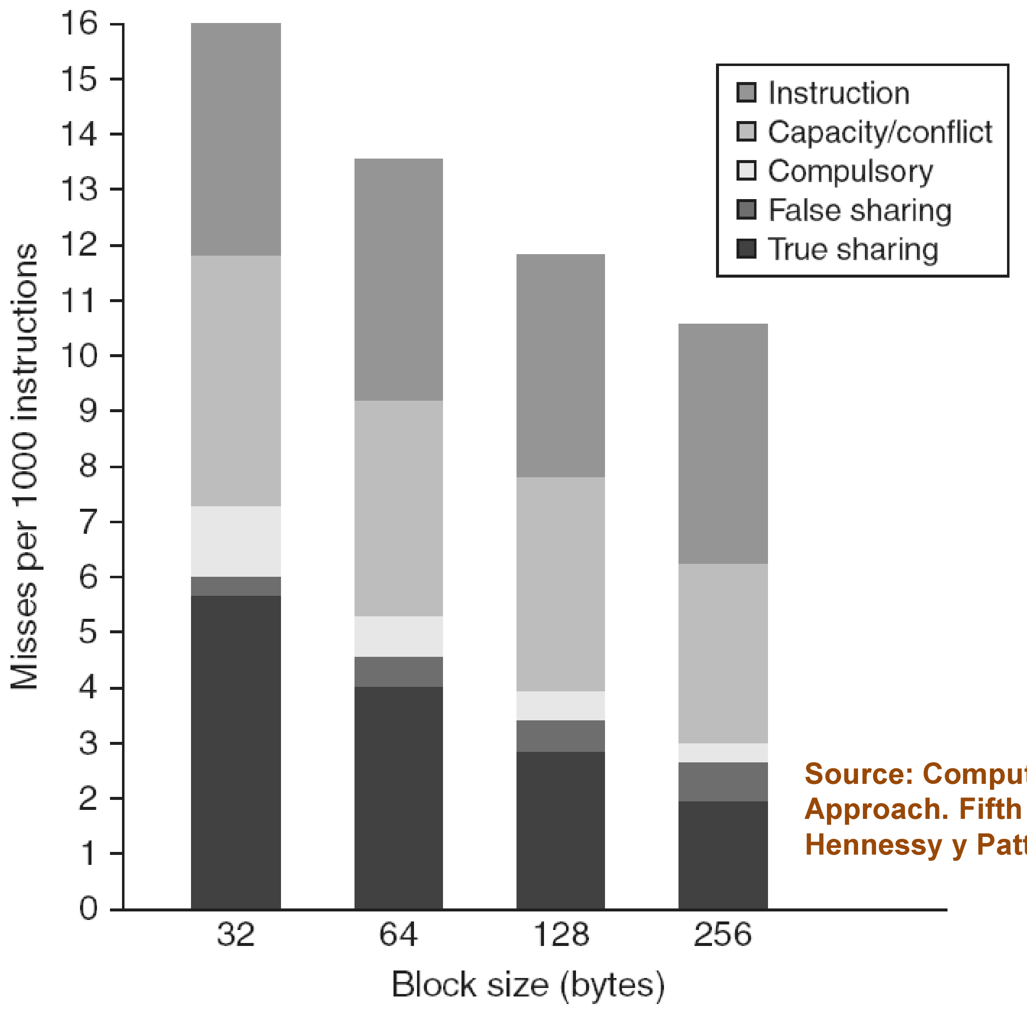
Increasing processors number

ARCOS



Source: Computer Architecture. A Quantitative Approach. Fifth Edition. Hennessy y Patterson.

Increasing block size



Source: Computer Architecture. A Quantitative Approach. Fifth Edition.
Hennessy y Patterson.

Universidad Carlos III de Madrid
www.uc3m.es

Reference

ARCOS

- **Computer Architecture. A Quantitative Approach.**
  Fifth Edition.
  Hennessy y Patterson.
  Sections: 5.1, 5.2 y 5.3

- Exercises: 5.1, 5.2, 5.3, 5.4, 5.5 y 5.6