ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
UNIVERSITY OF MICHIGAN

Final Exam Review Session

December 12, 2025

Exam Prep Tips

- Take past exams
  - Detailed solution to W15, W18, and F21 to be posted
- Review homeworks
- Review lecture slides
- Ask questions in office hours/Piazza
- **Take care of yourself**

Review Session Agenda

This review session will cover some of the tested final exam material:

1. Multiprocessors
2. Virtual Memory
3. LSQs (if time)

We will alternate between review slides and practice problems

Multiprocessors

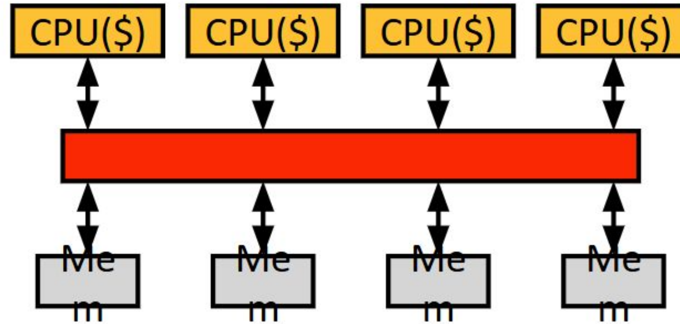
Shared Memory

We want multiple processors to work together. How should they communicate?

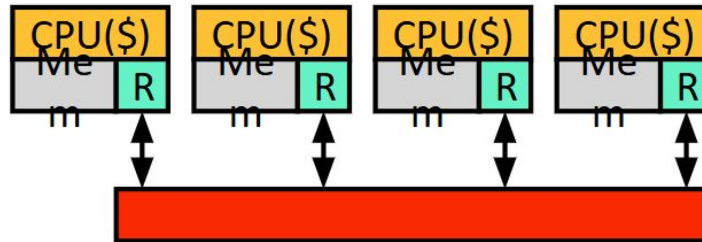
- Have them share a global address space
- Communication occurs via loads/stores to global address space
- For other communication methods/mechanisms, take 570

Organizing CPUs and Memory

UMA (Uniform Memory Access)

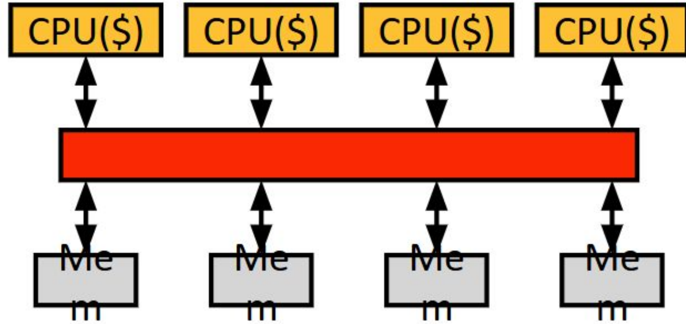


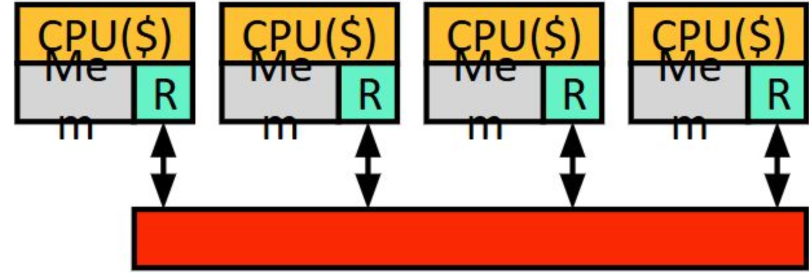
NUMA (Non-Uniform Memory Access)



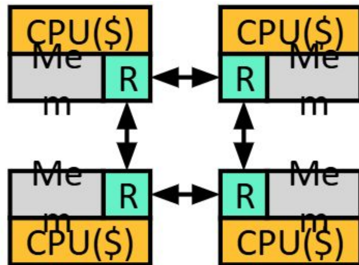
Communication Mediums

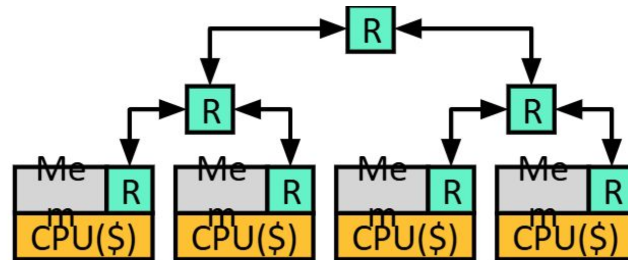
Bus-Based Systems





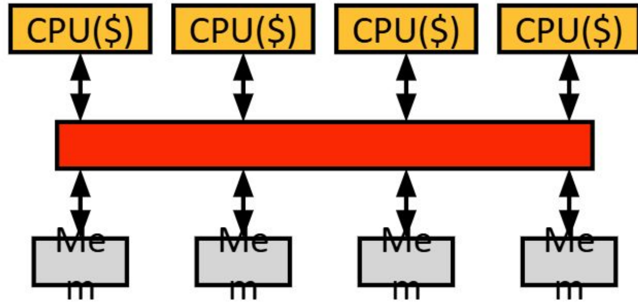
Point-to-point networks:



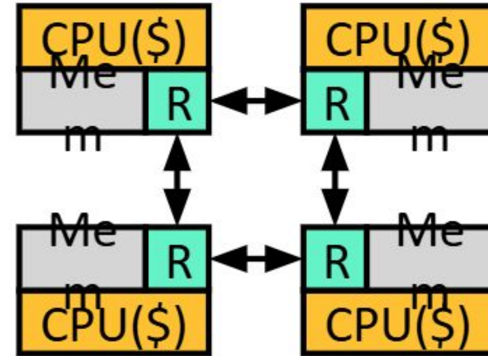


Two Basic Implementations

Snooping Bus MP:



Scalable MP:



Why is scalable MP more scalable that snooping bus?

What is cache coherence?

Any changes to a shared memory location are reflected across all caches holding that location.

Coherence is accomplished through a series of coherence messages communicated on the bus/network

Many different schemes:

- Valid/Invalid
- MSI
- MESI
- MOESI

Valid/Invalid Protocol

Allows multiple simultaneous readers of a cache line, but must write-through to memory

Processor Actions: Load, Store, Evict
Bus Messages: BusRd, BusWr

Let’s draw the valid-invalid protocol state diagram

MSI Protocol

MSI supports write-back caches. Three states:

1. Invalid - cache does not have a copy
2. Shared - cache has a read-only clean copy
3. Modified - cache has a the only copy, writeable/dirty

Processor Actions: Load, Store, Evict

Bus Messages: BusRd, BusRdX, BusInv, BusWB, BusReply

Let's draw the MSI protocol state diagram

Adding the E state (MESI)

Read followed by a write is a common access pattern. MSI requires 2 bus transactions for this.

Optimized by adding the E (exclusive) state

- Only copy, but clean
- When doing a store, cache line can **silently** transition to the modified state
- Cache line enters the E state if memory responds to read, not another cache.

MESI State Diagram

Invalid
Shared
Exclusive
Modified

Load / BusRd(reply from cache)
BusRdX / [BusReply], BusInv / --
Evict / --
BusRd / [BusReply]
Load / --
Load/ BusRd (reply from mem)
Evict / --
BusRdX / BusReply
BusRd / BusReply
Store / BusRdX
BusRdX / BusReply
Evict / BusWB
Store/ BusInv
BusRd / BusReply
Store / --
Load / --
Load, Store / --

Adding the O state (MOESI, MOSI)

MSI/MESI must writeback to memory on M→S transitions.

- Why?
- Such transitions are common with producer-consumer scenarios

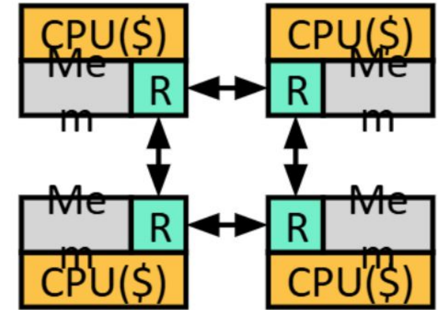
Optimized by adding the O (owned) state

- Shared copy, but dirty
- Processor in owned state is responsible for write-back on eviction
- Cache line enters owned state when previously in modified and responds to a BusRd.

Time for problems 1 and 2…

Directories

Scalable systems can send coherence messages only to the processors they concern through P2P interconnect



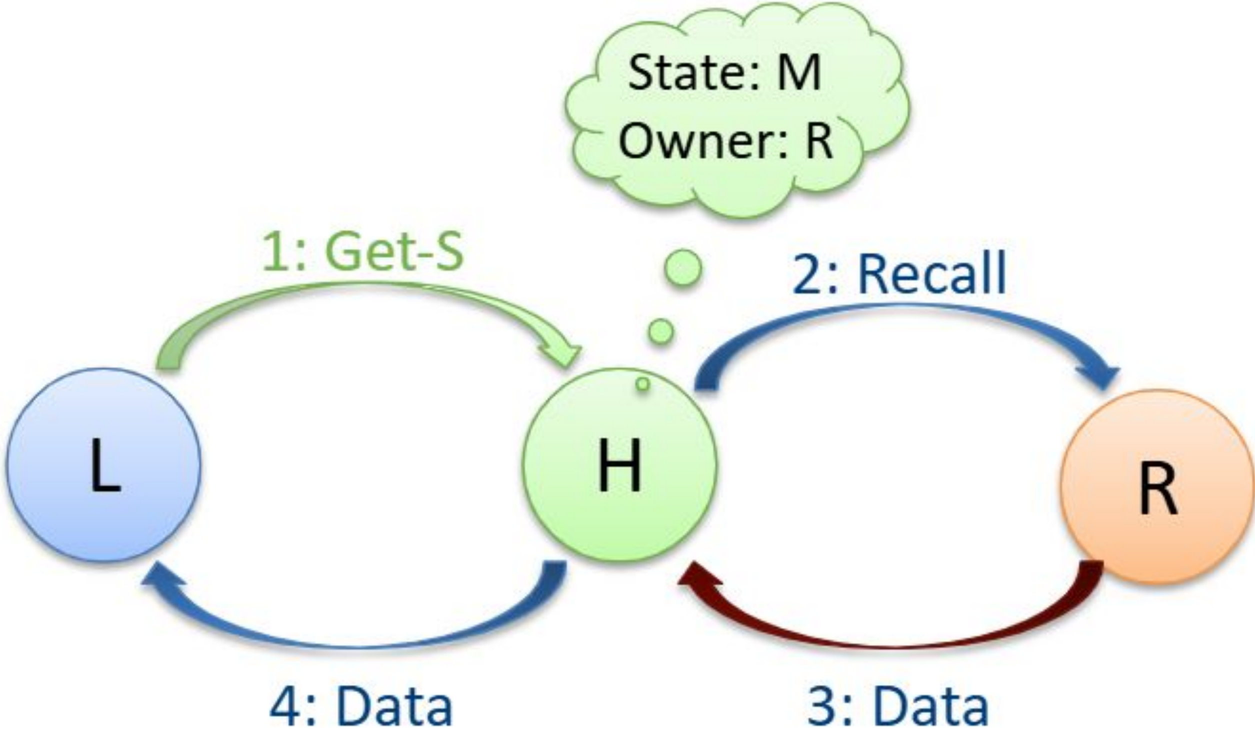
For each cache line, a directory keeps track of:

- Owner: which processor has a dirty copy (M)
- Sharers: which processor(s) have clean copies

In directory protocol, a processor sends a coherence request to home directory, and directory sends messages to the appropriate processors

Unified vs distributed

4-Hop Read



3-Hop Read

State: M
Owner: R

1: Get-S

2: Fwd-Get-S

L

H

R

3: Data

3: Data

Coherence vs Consistency

Coherence: All processors observe the most recent write to **a specific** memory location

Consistency: Order in which memory operations (loads and stores) appear to execute across different processors

- Consistency models specify rules about the ordering of memory operations across multiple addresses.

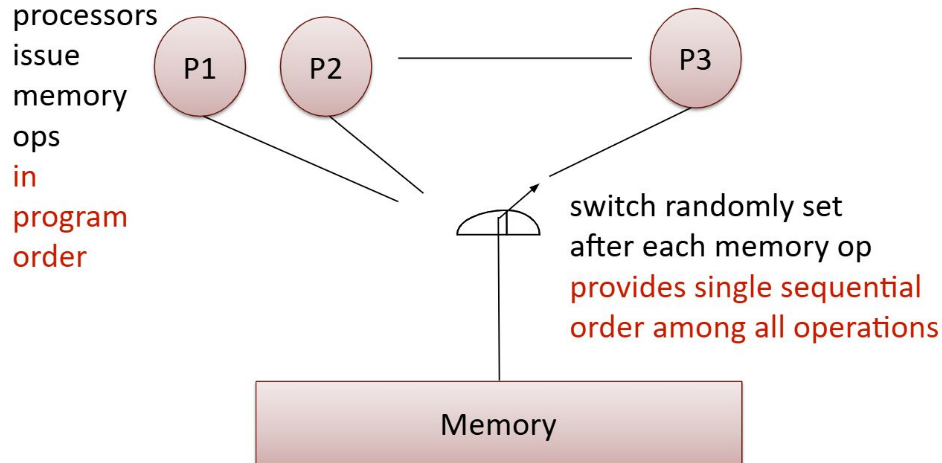
```
/* initial A = B = flag = 0 */

 P1              P2
A = 1;          while (flag == 0); /* spin */
B = 1;          print A;
flag = 1;       print B;
```

Sequential Consistency (SC)

Processors issue memory operations in program order and operations start and end atomically

SC restricts a lot of optimizations, other consistency models exist that aren't as restrictive



Virtual Memory

What's the point of virtual memory?

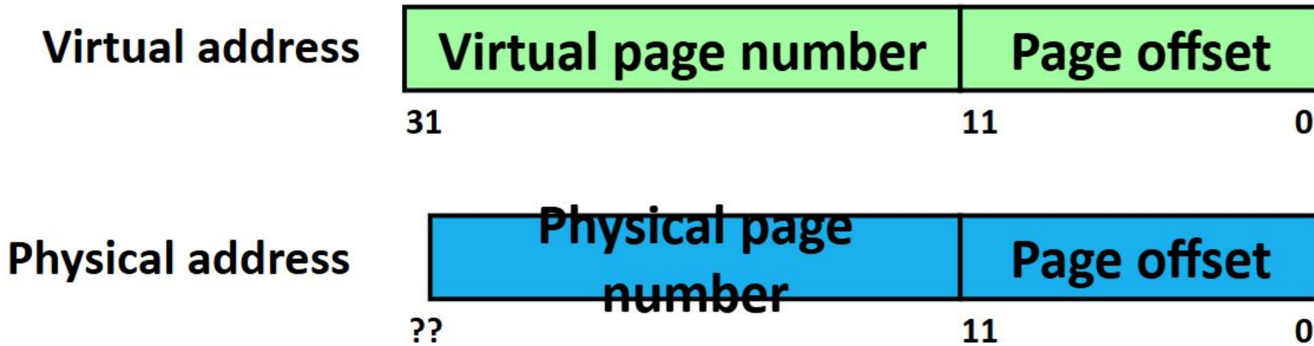
What’s the point of virtual memory?

- Allows each process to see/use a continuous large address space
- Privacy mechanism
- Mechanism for swapping data between DRAM and secondary storage (disk)

For each memory operation, hardware must translate the virtual address used by the program into a physical address

Pages

Page: Fundamental unit of memory management (typically 4 KB in size)



How do we translate a virtual address into a physical address?

Single Level Page Table

The *Page table register* points to the beginning of the page table in main memory

Single Level Page Table Example

Hierarchical Page Tables

Hashed/Inverted Page Table

Base of Table

PID

VPN

hash

Table Offset

+

PA of IPTE

Inverted Page Table

VPN	PID	PTE

Physical Memory

Address Translation Flowchart and TLB

Effective Address

TLB Lookup

≤ 1 pclk

miss

hit

100's pclk
by HW or SW

Page Table Walk

Protection Check

≤ 1 pclk

succeed

fail

denied

permitted

Update TLB

Page Fault
OS
Table Walk

10000's pclk

Protection Fault

Physical Address To Cache

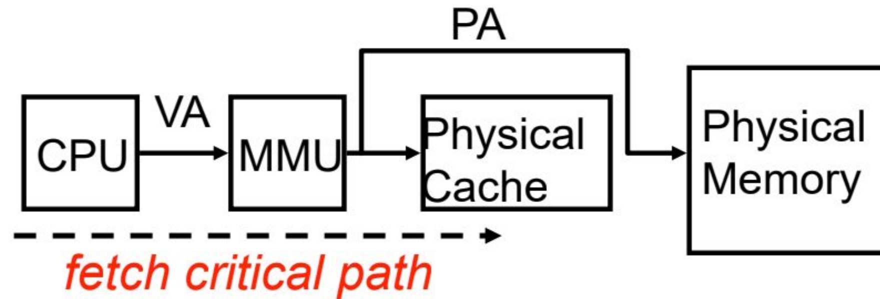
PIPT Caches

PIPT - Physically indexed, physically tagged

- i.e. set index and block tag come from physical address

Virtual-to-physical address translation must occur in its entirety before cache look-up

Limitations?



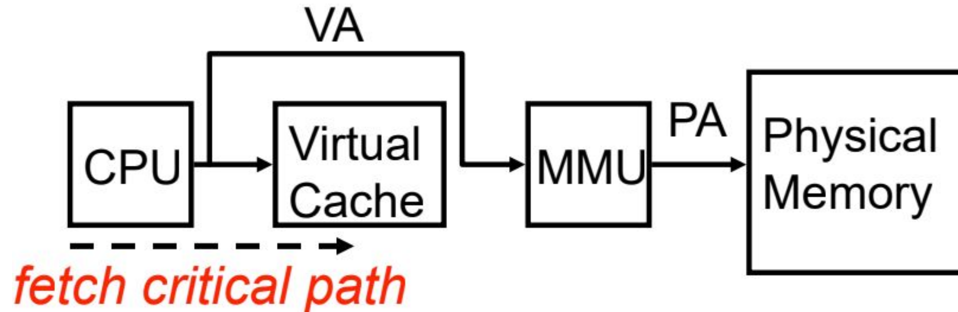
VIVT Caches

Virtually Indexed, Virtually Tagged Cache

- i.e. set index and block tag come from virtual address

Cache lookup can occur in parallel with address translation. Result of address translation used to physical memory lookup in case of cache miss.

Limitations?



VIPT Caches

Virtually Indexed, Physically Tagged Cache

- i.e. set index from virtual address, tag from physical address

Can index into cache in parallel with address translation

- TLB lookup is generally faster than cache lookup
- Tag comparison uses translated address

Advantages:

- No need to invalidate on context switch
- Reduced memory fetch critical path

The Synonym Problem

Synonyms:
Two different virtual addresses that map to the same physical address.

Why is this a problem for our virtually-indexed caches?

The Synonym Problem

Synonyms:
Two different virtual addresses that map to the same physical address.

Why is this a problem for our virtually-indexed caches?

- If the two virtual addresses index into different sets in the cache, there could be two copies of the same data (sadness)
- A change in one copy will not get reflected in the other - leads to inconsistency

The Synonym Problem

Virtual Pg No. (VPN)

Tag | Index | Page Offset

v-k k g

TLB

p

PPN

p

Virtual Pg No. (VPN)

Tag | Index | Page Offset

Index | BO

a i b

D-cache

p

PPN

p

Data

=

Hit/Miss

Solutions to the synonym problem

Ensure that the same physical data always gets placed in the same cache set, no matter the virtual address.

Solution 1: Limit Cache Size

- Make sure set index is derived only from page offset bits

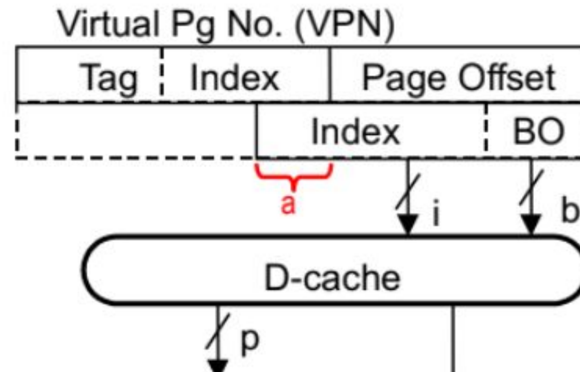
Page offset bits >= block offset bits + set index bits

Solutions to the synonym problem

Ensure that the same physical data always gets placed in the same cache set, no matter the virtual address.

Solution 2: Page Coloring

- OS guarantees that VPN bits 'a' used in the set index are the same for any synonyms



Load-Store Queues

LSQ Motivation

Correctness:

- Loads have potential dependencies on any prior stores
  - If they go to the same address, the store has the most up to date value
- Don’t want to sends stores to cache prior to retirement
  - Very hard to undo a speculative update to memory

Naive solution: stall loads until all prior stores have retired and written to cache

Optimizations:

- <u>Forwarding</u>: Before store writes to cache, forward data from store queue
  - Allows loads to issue before all prior stores *retire*
- <u>Speculation</u>: Send load to cache before knowing if there’s a dependency
  - Allows loads to issue before all prior stores *issue*

Requirements

Store Queue

- Must have a store queue to ensure correctness
  - Hold them until retirement
  - Ensure they write in order
- Very similar to the ROB

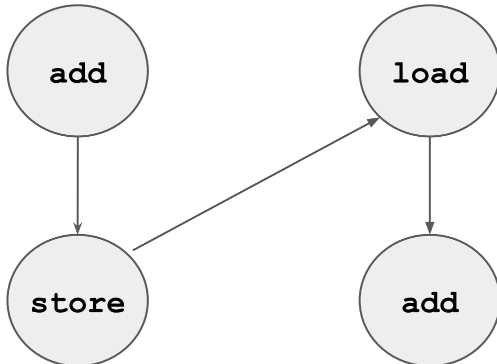
Load Queue:

- Only required to be ordered if doing speculation
  - Allocate from dispatch until all prior stores execute
- If only doing forwarding, can just buffer loads while waiting for data from cache
- May need to combine data from both store queue and data cache

Performance Analysis

With Forwarding

- Loads can only execute after stores complete
- Draw dependency from a load to a store



With Speculation

- Analyze ideal case with no mispeculations
- No dependency between loads and stores