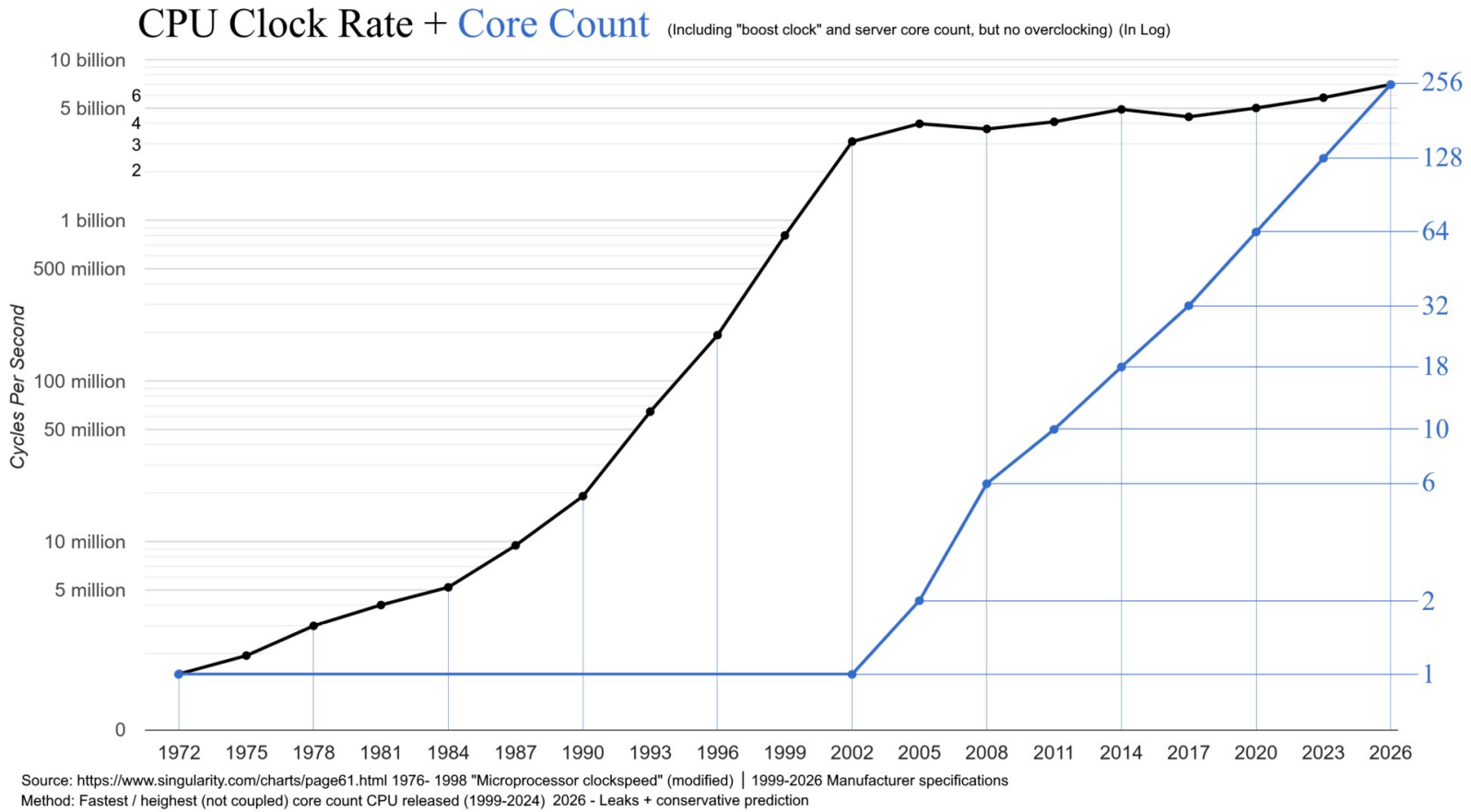
Multiprocessor Concurrency and Memory

Learning Outcomes

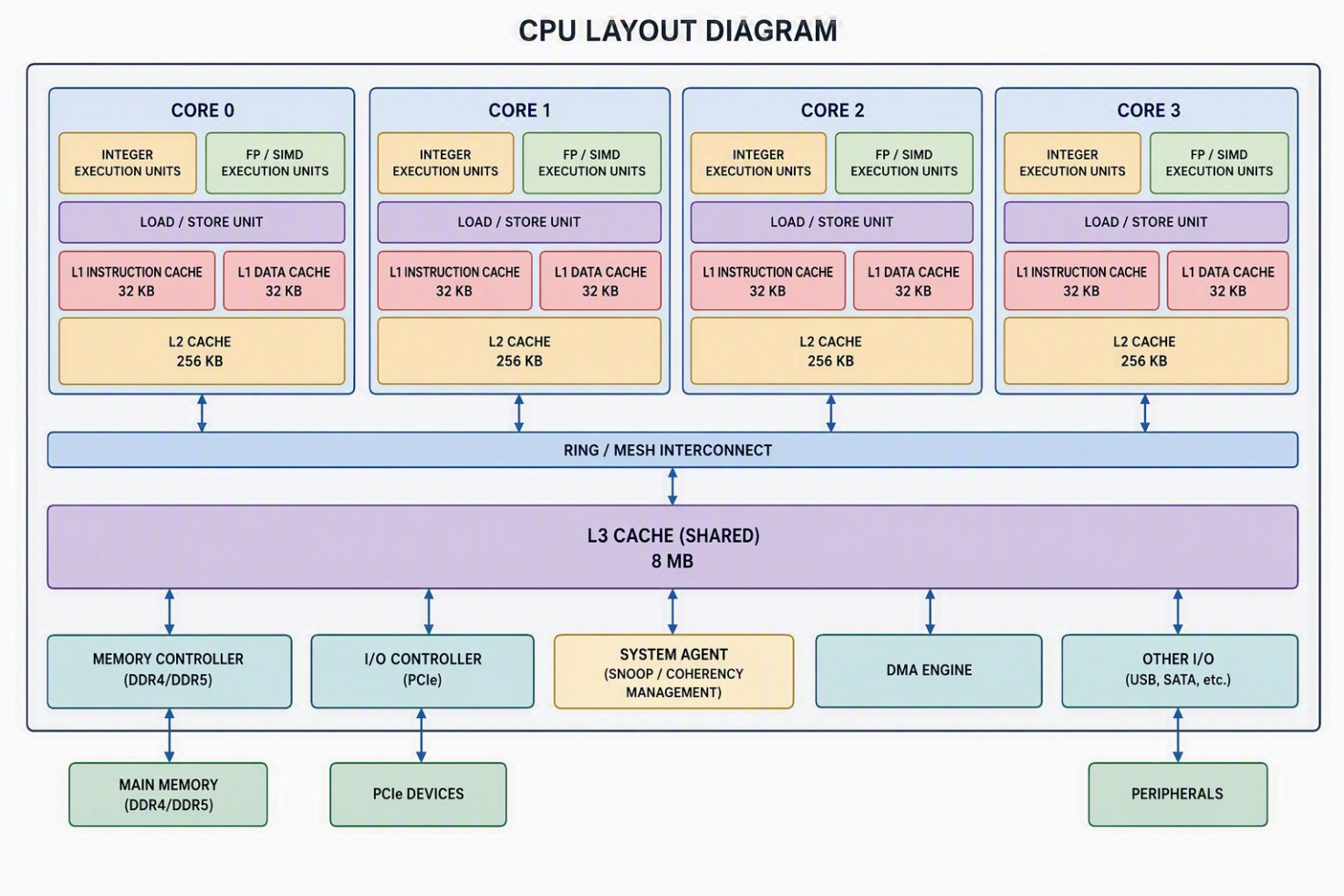
- An understanding of the various issues created by concurrent out-of-order access to memory.

- Note: this content is not assessible.

Context: CPUs are going Multi-Core



Context: CPUs are going Multi-Core



(Warning: this diagram invented by AI, might contain nonsense.)

Multi-Core Memory Access

- Programming on multi-core systems introduces “true parallel” execution as opposed to interleaved concurrency.
- Different cores and devices are connected to memory via separate buses, and may have different views of it.
- We need to be aware of inter-core memory semantics and its implications for concurrent programming.

A Litmus Test

Initial state: `x = 0, y = 0`	

Thread 1:
 	
`store y, 1`
`load r0, x`
	Thread 2:
	
	`store x, 1`
	`load r1, y`

Which outcomes are possible? Can we have `r0 == 0` and `r1 == 0` in the final state?

- Interleaved execution: either `r0 == 1` or `r1 == 1`.

Experimenting with A Litmus Test

Initial state: `x = 0, y = 0`	

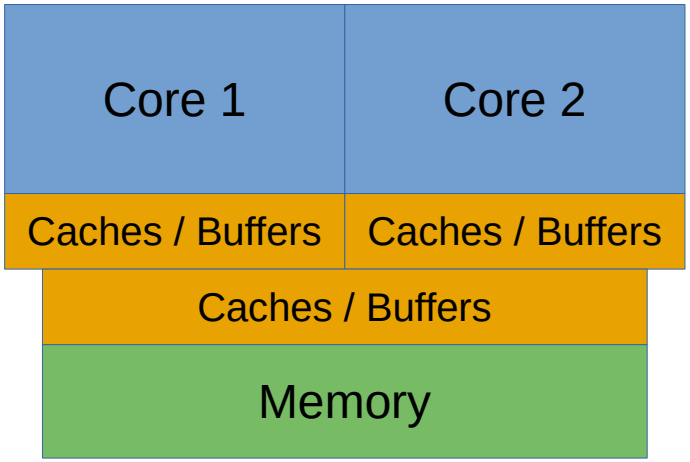
Thread 1:
 	
`store y, 1`
`load r0, x`
	Thread 2:
	
	`store x, 1`
	`load r1, y`

If we scale this example up to an experiment, we will see:

- Single-core system with interleaving: three outcomes.
- Nearly all multi-core systems: four possible outcomes.

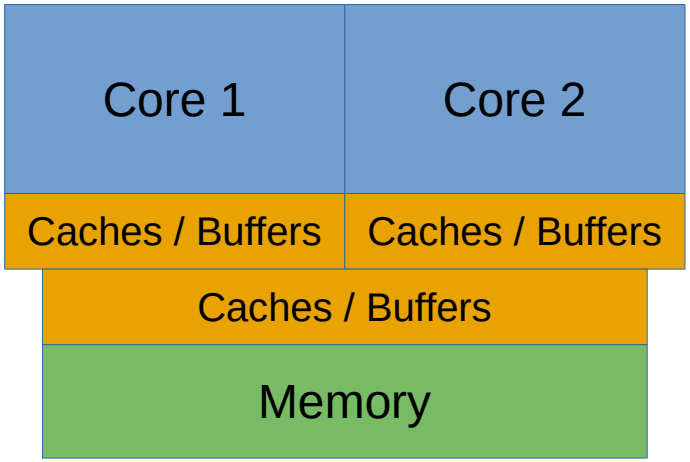
Store Buffering

- Both threads write then read variables x/y.
- The reads fetch old data.
- Why?
  - Caching old data.
  - Delaying of writes.
  - We'll see some more.
- Concurrency experts refer to these effects collectively as *store buffering*.



Store Buffering vs our Cache Model

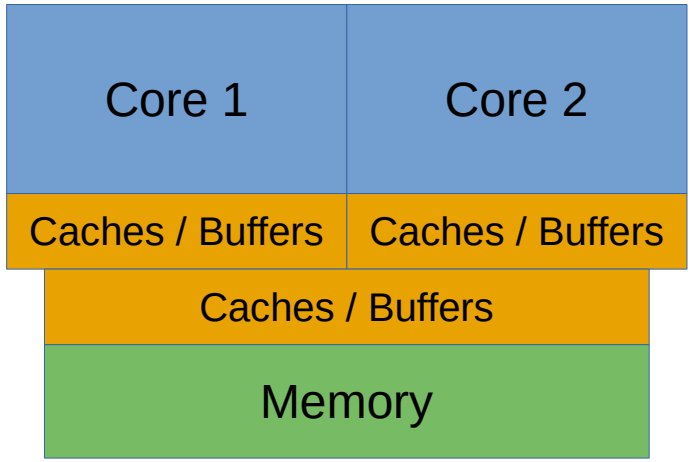
- We discussed a made-up cache model in the previous concurrency lecture.
- It would not permit store buffering.
  - Strong consistency semantics.
- We could permit store buffering by tweaking the invalidate and exclusive mechanisms.
  - *Weak memory* semantics.



Weak Memory Hierarchy

Test CPU architectures against weak memory semantics:

- Standardise litmus test.
- Compare to CPU manuals.
- Run on real CPUs.



With various litmus tests, we can group CPUs by the strength of their memory semantics.

Modern Weak Memory Hierarchy

Memory strength hierarchy:

- Sequentially consistent.
- x86 (with TSO).
- Recent ARM (with MCA) and RISC-V.
- Recent-ish ARM (without MCA).

Older CPUs are much less consistent:

- Alpha, Power, Itanium.

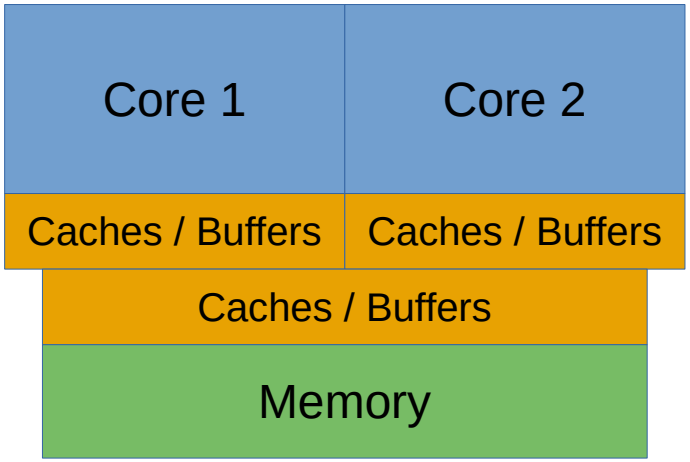
Strong research/university contribution to weak memory investigation:

- Cambridge work on TSO and MCA.

Real-World Weak Memory Effects

Weak (out-of-order) memory effects are caused by various mechanisms in the real world:

- Permissive cache coherence protocol.
- Write buffers.
- System hierarchy.
- Speculation.
- CPU Instruction re-ordering.

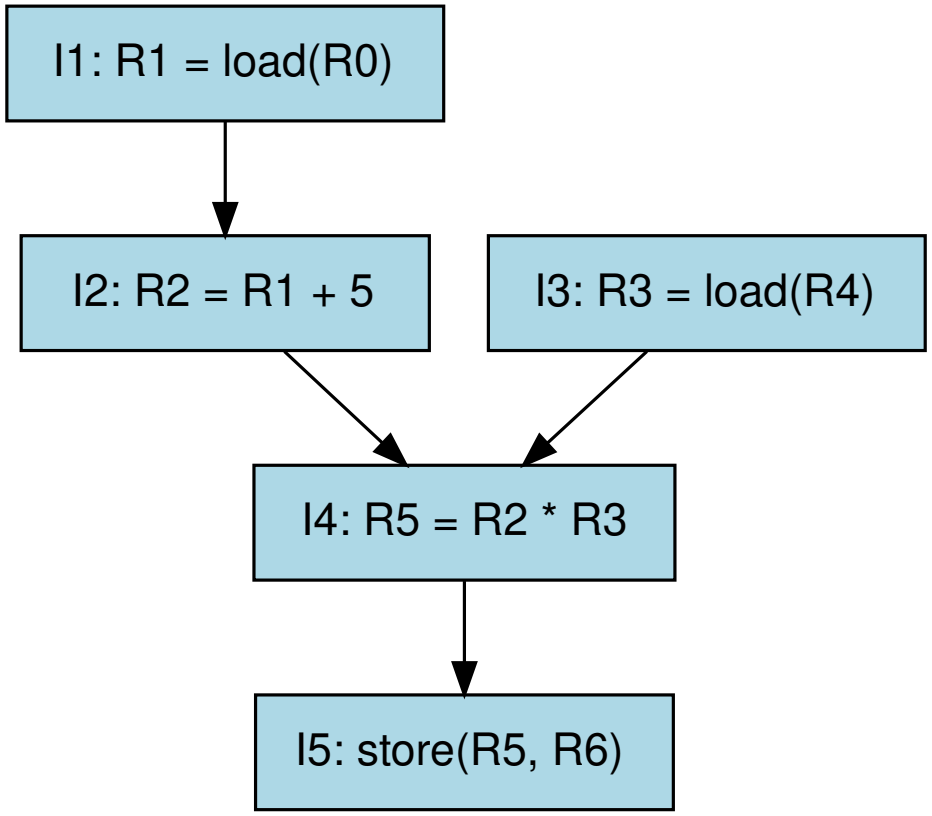


Weak Memory Virtualisation

- Recall virtualisation process:
  - Yesterday's system.
  - Today's compatibility trick.
  - Tomorrow's interface.
- What is a CPU register or read/write instruction?
  - Modern system simulating an older in-order system.
- CPUs guarantee single threaded compatibility.

CPU Re-Ordering

```
R1 = load(R0);
R2 = R1 + 5;
R3 = load(R4);
R5 = R2 * R3;
store(R5, R6);
```



Modern large CPUs do this kind of optimisation aggressively.

Modern Weak Memory Categories

TSO Total Store Order:

- Writes may be delayed in write buffer.
- Local core sees writes in write buffer.
- Write buffer is released in order.
- Write releases form a global order.

MCA Multi-Copy Atomic vs Non-Atomic:

- Write release events may or may not be visible to other parties at the same time.

Scratch Space for MCA Diagram

Problematic Real-World Litmus Test

The “Load Buffering” Litmus test:

Initial state: `x = 0, y = 0`	

Thread 1:
 	
`load r0, x`
`store y, 1`
	Thread 2:
	
	`load r1, y`
	`store x, 1`

Can we have both `r0 == 1` and `r1 == 1` in the final state?

Dubious Real-World Litmus Test

An out-of-thin-air (OOTA) litmus test:

Initial state: `x = 0, y = 0`	

Thread 1:
 	
`load r0, x`
`add r2, x, 1`
`store y, r2`
	Thread 2:
	
	`load r1, y`
	`add r3, y, 1`
	`store x, r3`

Obviously real hardware does things in *some* order.

But the *specification* of this hardware may lose precision.

C11/C++11 Memory Ordering

The C11/C++11 standard introduced a standard hierarchy of memory consistency strengths:

- `memory_order_relaxed`
- `memory_order_consume` (now deprecated)
- `memory_order_acquire`
- `memory_order_release`
- `memory_order_acq_rel`
- `memory_order_seq_cst`

See https://en.cppreference.com/c/atomic/memory_order .

Acquire/Release Semantics

Initial state: `flag = 0, x = 0`	

Thread 1:
 	
`write x, 12`
`write flag, 1, o_release`
	Thread 2:
	
	`...`
	
	`load r1, flag, o_acquire`
	`bz r1, #try_again`
	`load r2, x`

We note the different kinds of dependencies and study the possible effective orderings.

Dependency Graph Scratch Space

When concurrency experts describe a litmus test, there is usually no actual program.

An Example

See ARM AArch64 manual Appendix K11.3.3, example ticket lock.

https://support.arm.com/documentation/ddi0487/maa/-Part-K-Appendixes/-Appendix-K11-Barrier-Litmus-Tests/-K11-3-Load-Acquire-Exclusive--Store-Release-Exclusive-and-barriers/-K11-3-3-Ticket-locks

(I’m not sure if it would be OK to reproduce it in these slides.)

Note the pronunciation:

- LDAXR: load-acquire-exclusive-register
- STLRH: store-release-register-halfword

Good News

The rules for “ordinary” code aren’t that complicated.

- Single threaded program:
  - Nothing to worry about.

- Multi-threaded program:
  - Eliminate *data races* from the program.
  - Use regular (relaxed) memory accesses.
  - Use a well-tested or studied synchronisaton mechanism.

A Complication

We've been talking about concurrency between instruction reads and writes.

- Mostly: concurrency between CPU data caches.

What about:

- Instruction fetch?
- Hardware page-table walk and TLB load?
- I/O devices (DMA)?
- GPUs and heterogeneous CPUs?
- The page table dirty bit.

Standard PTE Adjustment Routine

Suppose an OS wants to modify a page-table entry from one valid mapping to another.

Most architectures mandate a routine sometimes called "break before make":

- Write an invalid/zero PTE.
- Memory barrier or cache clean.
- TLB broadcast-invalidate.
- Another barrier.
- Write new PTE.
- Recommended: Another barrier.

This Lecture

This lecture: Modern *weak* memory semantics and complications that arise from it.

- Genuine parallelism.
- Visible out-of-order execution.
- Concurrency types and litmus tests.
- C11 and similar memory ordering types.
- Concurrency in non-CPU devices and OS support.

- There are open research problems in this space.