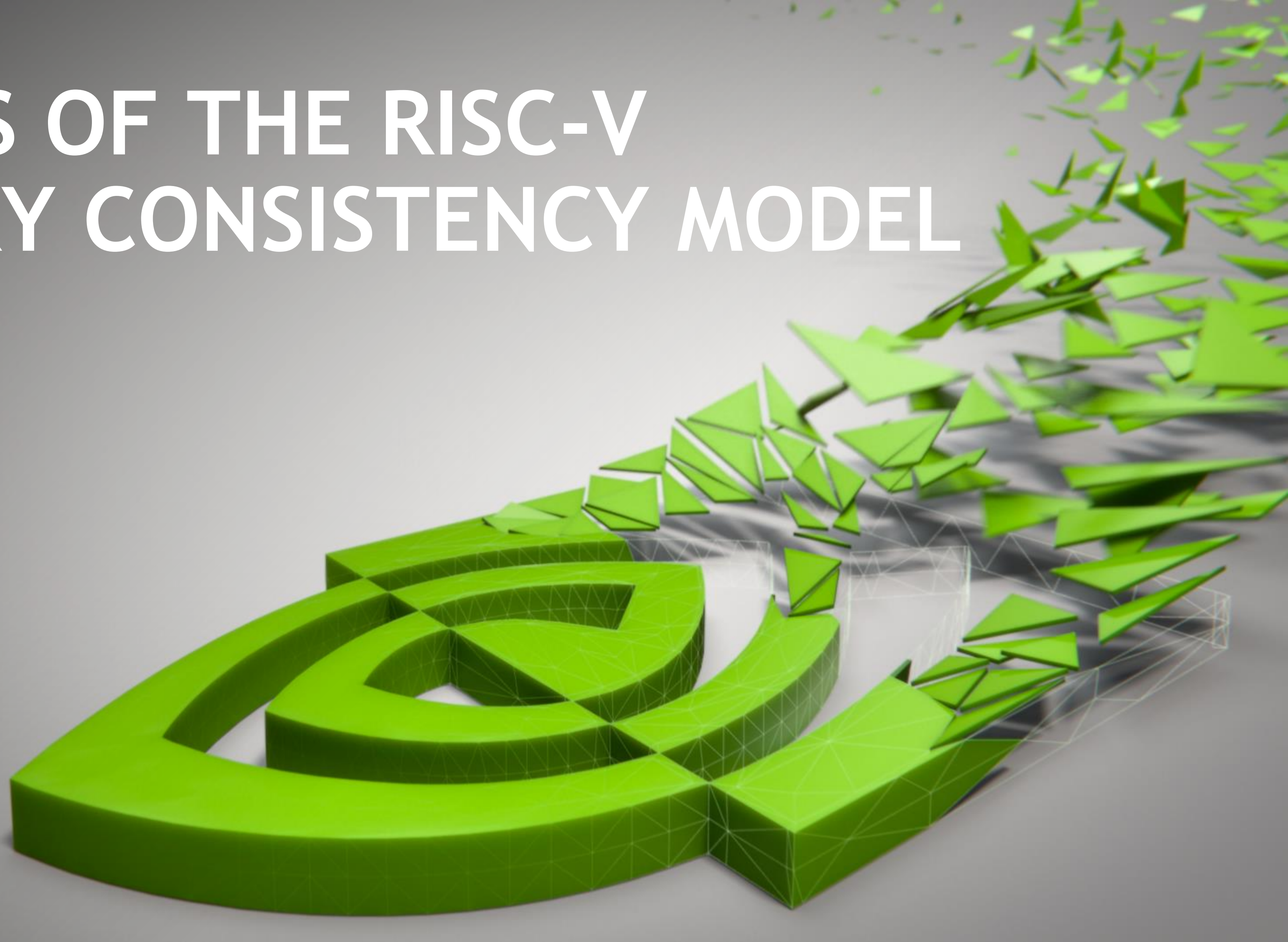
STATUS OF THE RISC-V MEMORY CONSISTENCY MODEL

Daniel Lustig

May 10, 2017



THE RISC-V MEMORY MODEL IS FINE!

- If you’re concerned about recent press, *don’t be!*
- We’re well aware of and on top of the issues
- We caught the spec bugs well before they’ll actually affect any implementations in practice
- Great example of the benefits of open-source ISA

MEMORY CONSISTENCY MODEL

The set of rules specifying the values that can be legally returned by memory loads

SEQUENTIAL CONSISTENCY

1. All threads are interleaved into a single “thread”
2. The interleaved thread respects each thread’s original instruction ordering (“program order”)

SEQUENTIAL CONSISTENCY

1. All threads are interleaved into a single “thread”
2. The interleaved thread respects each thread’s original instruction ordering (“program order”)
3. Loads return the value of the most recent store to the same address, according to the interleaving

SEQUENTIAL CONSISTENCY

1. All threads are interleaved into a single “thread”
2. ~~The interleaved thread respects each thread’s original instruction ordering (“program order”)~~
3. Loads return the value of the most recent store to the same address, according to the interleaving

For performance, most processors weaken rule #2

REORDERINGS ALLOWED BY RISC-V (AND OTHERS)

Second

First

	Ld	St
Ld	Y	Y
St	Y	Y

Sequential Consistency

	Ld	St
Ld	Y	Y
St	—	Y

TSO (x86)

	Ld	St
Ld	—	???
St	—	—

RISC-V

	Ld	St
Ld	—	—
St	—	—

Power, ARM

"Y" = ordering enforced by default

"—" = ordering not enforced by default

REORDERINGS ALLOWED BY RISC-V (AND OTHERS)

This is a common presentation of memory models, but it's a woefully incomplete picture!

Second / First

	Ld	St
Ld	Y	Y
St	Y	Y

Sequential Consistency

	Ld	St
Ld	Y	Y
St	—	Y

TSO (x86)

	Ld	St
Ld	—	???
St	—	—

RISC-V

	Ld	St
Ld	—	—
St	—	—

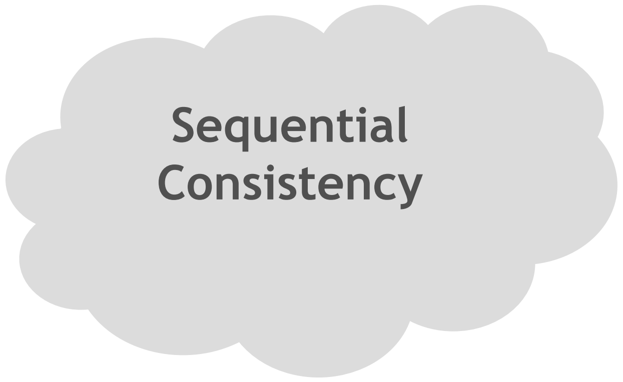
Power, ARM

"Y" = ordering enforced by default

"—" = ordering not enforced by default

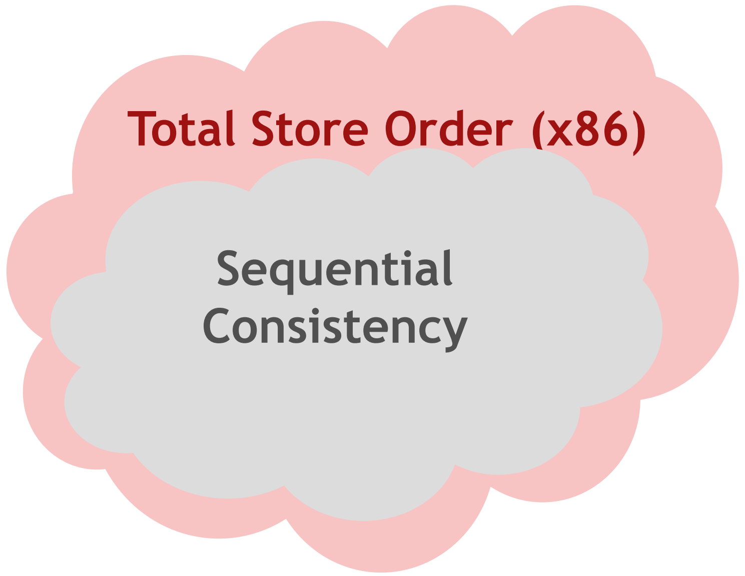
THE MODEL AS AN UPPER BOUND

Where should RISC-V draw the line?



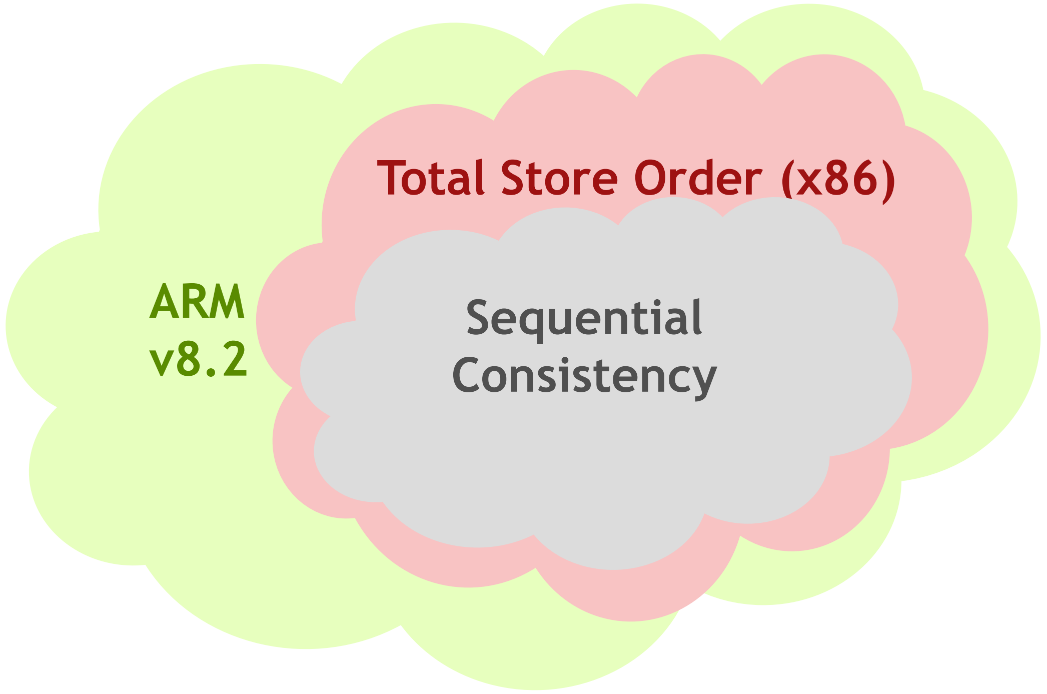
THE MODEL AS AN UPPER BOUND

Where should RISC-V draw the line?



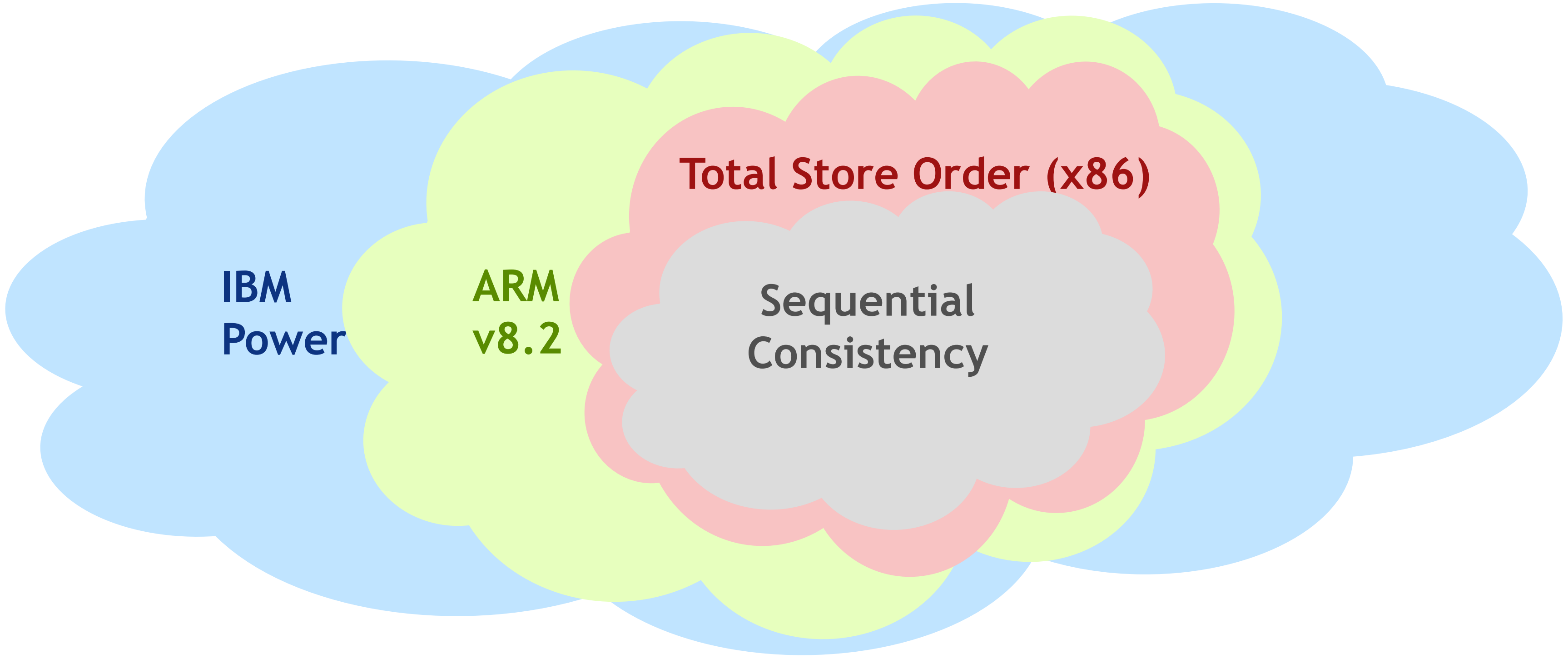
THE MODEL AS AN UPPER BOUND

Where should RISC-V draw the line?

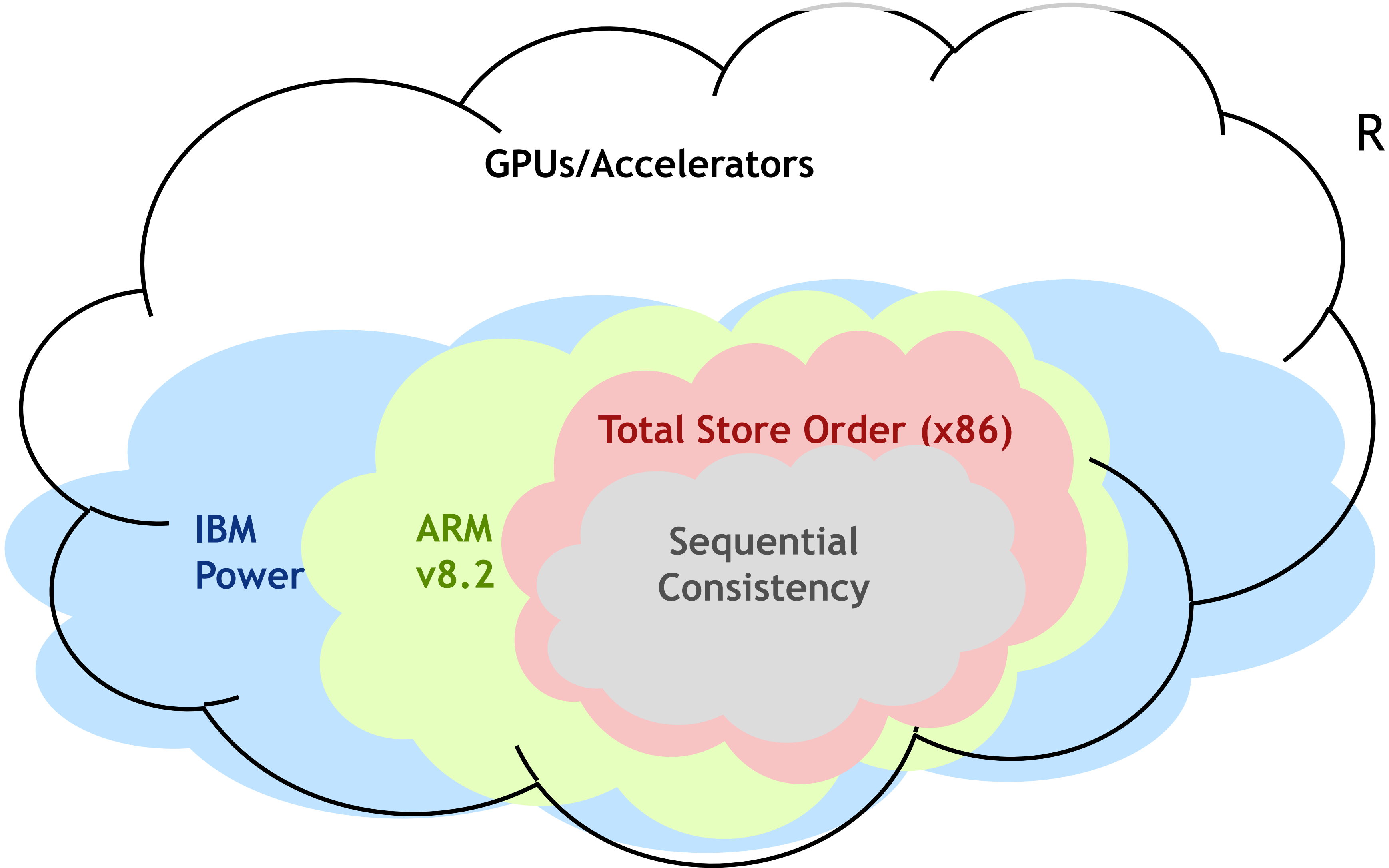


THE MODEL AS AN UPPER BOUND

Where should RISC-V draw the line?

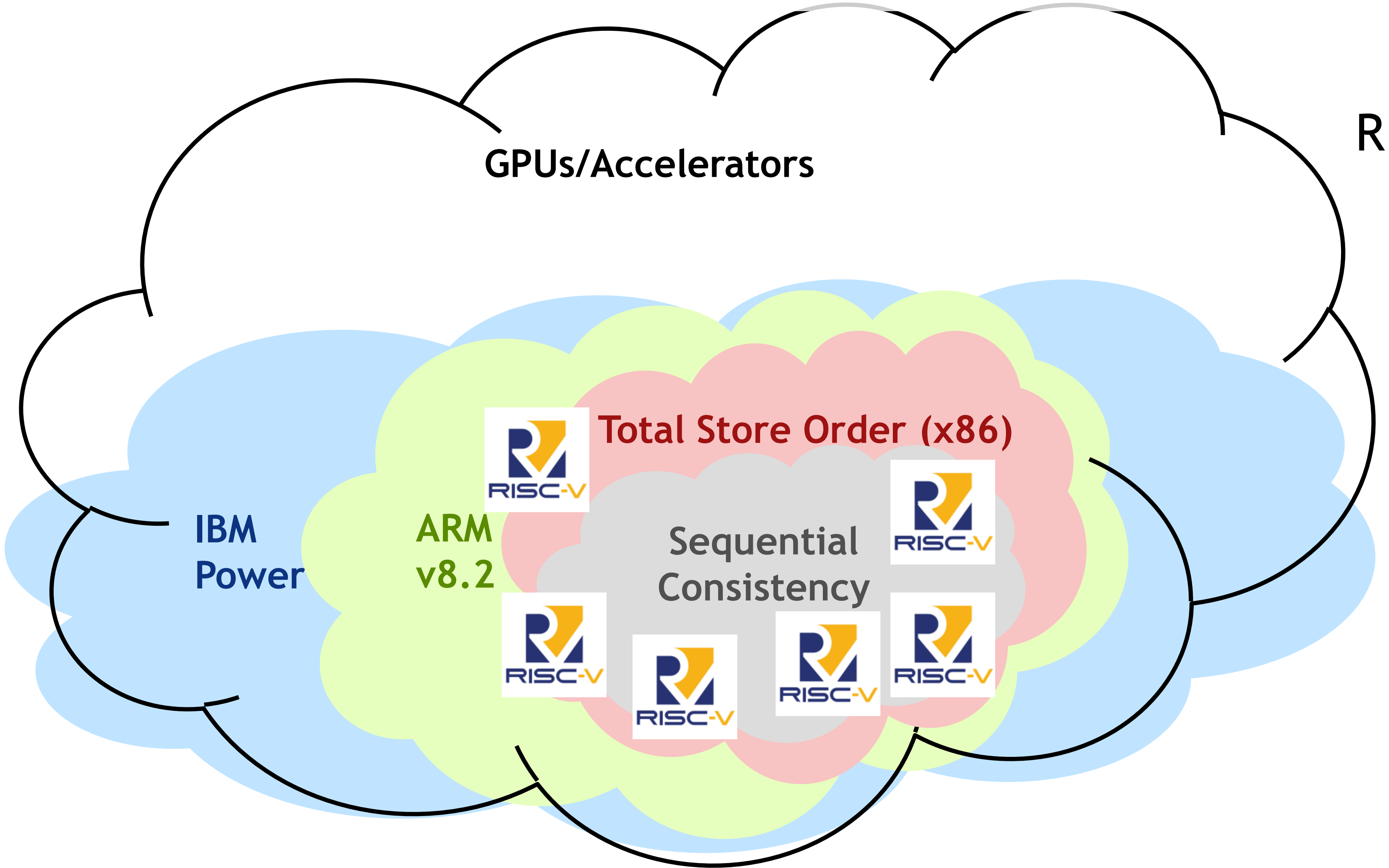


THE MODEL AS AN UPPER BOUND



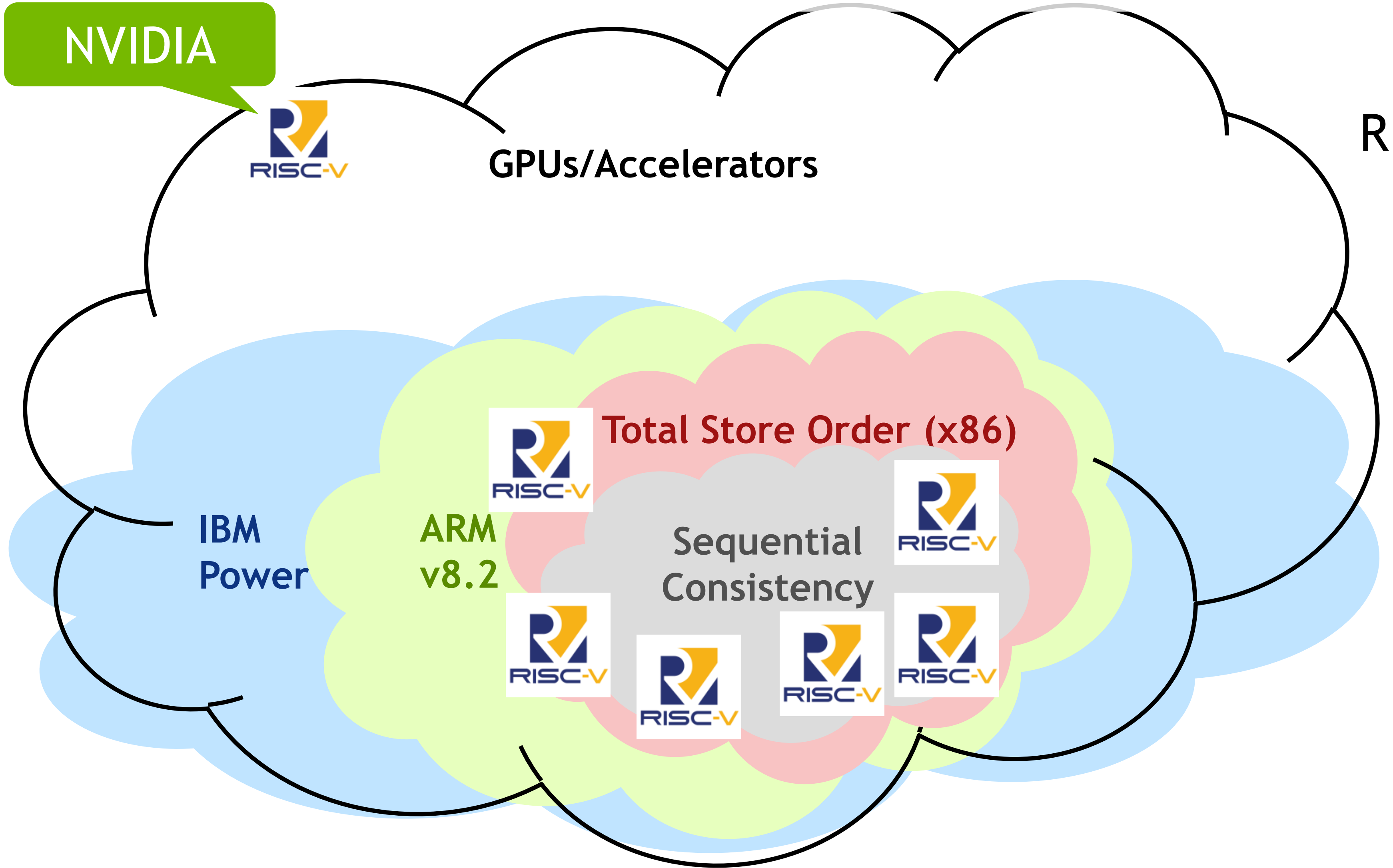
Where should RISC-V draw the line?

THE MODEL AS AN UPPER BOUND



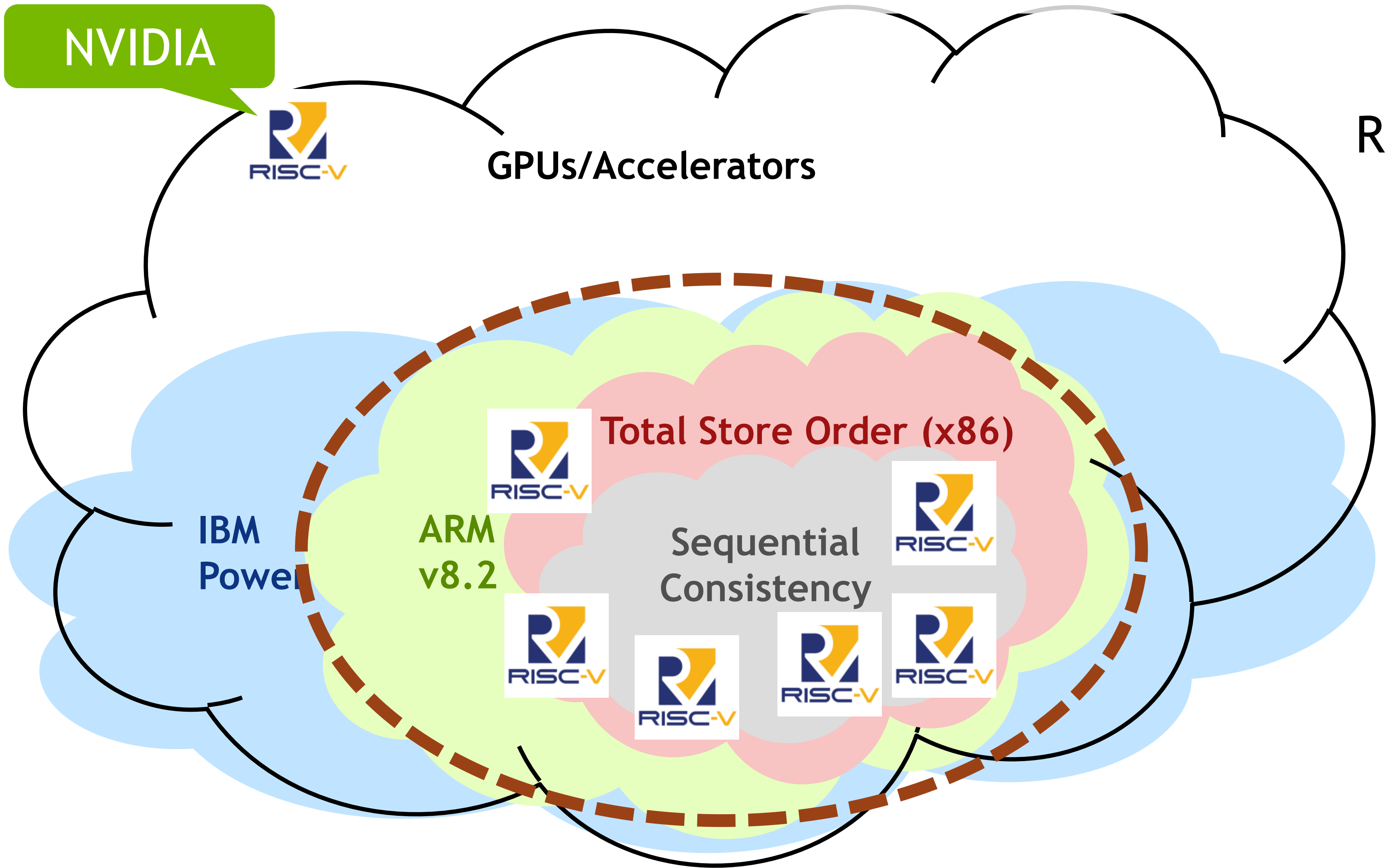
Where should RISC-V draw the line?

THE MODEL AS AN UPPER BOUND



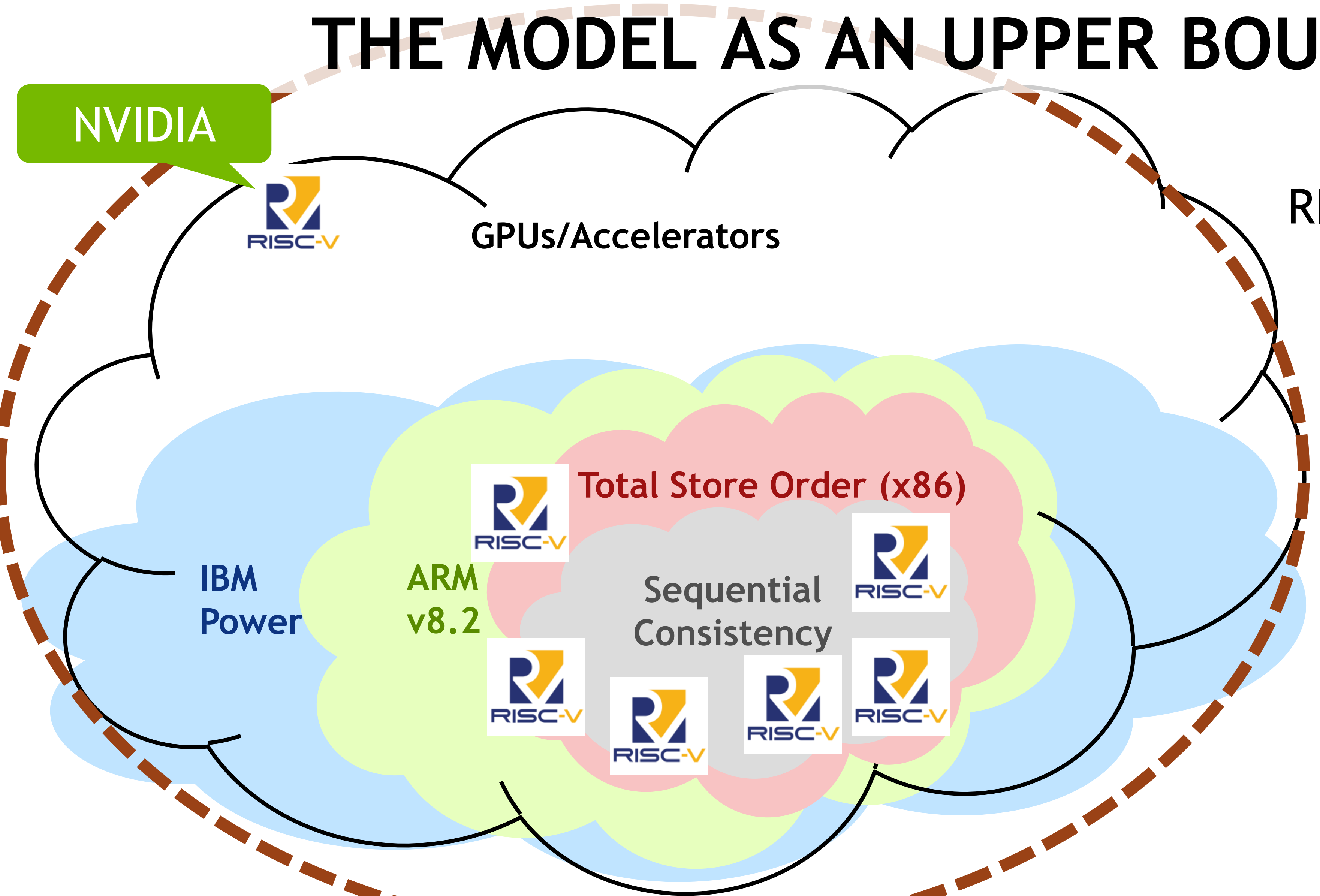
Where should RISC-V draw the line?

THE MODEL AS AN UPPER BOUND



Where should RISC-V draw the line?

THE MODEL AS AN UPPER BOUND



Where should RISC-V draw the line?

WEAKENING EVEN FURTHER…

1. All threads are interleaved into a single “thread”
2. ~~The interleaved thread respects each thread’s original instruction ordering (“program order”)~~
3. Loads return the value of the most recent store to the same address, according to the interleaving

For performance, most processors weaken rule #2

WEAKENING EVEN FURTHER…

1. ~~All threads are interleaved into a single "thread"~~
2. ~~The interleaved thread respects each thread's original instruction ordering ("program order")~~
3. Loads return the value of the most recent store to the same address, according to *(some other rules)*

For performance, most processors weaken rule #2

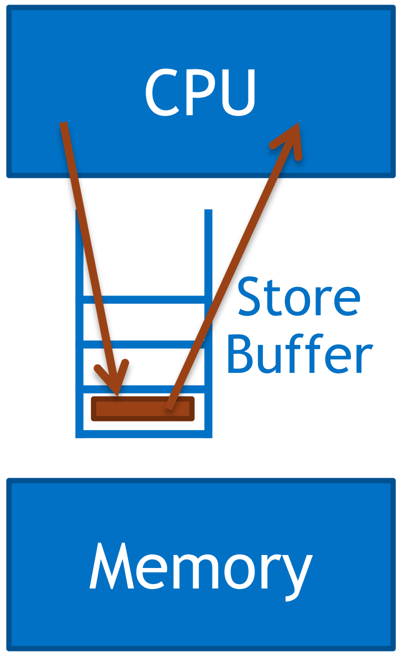
Most weaken rule #1 as well

STORE ATOMICITY

- Q: Can I think of an execution as an interleaving of the instructions in each thread (in some order)?

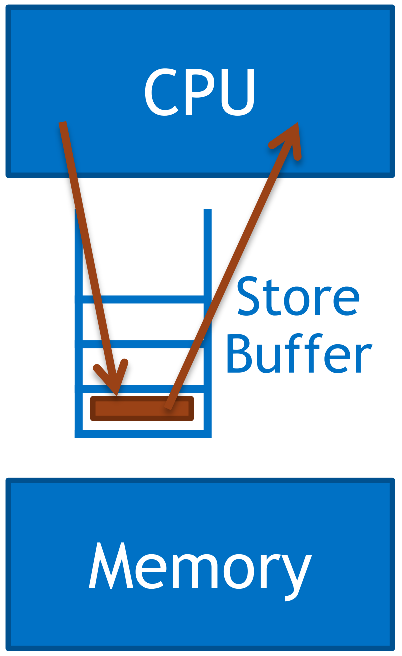
STORE ATOMICITY

- Q: Can I think of an execution as an interleaving of the instructions in each thread (in some order)?
- A: No! That would make it illegal to forward values from a store buffer!
    - Because with a store buffer, cores can read their own writes “early”



STORE ATOMICITY

- Option 1: forbid store buffer forwarding, keep a simpler memory model, sacrifice performance
- Option 2: change the memory model to allow store buffer forwarding, at the cost of a more complex model
- Nearly all processors today choose #2



STORE ATOMICITY

- Q: Can I think of an execution as an interleaving of the instructions in each thread (in some order), *with an exception for store buffer forwarding*?

STORE ATOMICITY

- Q: Can I think of an execution as an interleaving of the instructions in each thread (in some order), *with an exception for store buffer forwarding*?
- A: **Yes**, on x86 and ARMv8.2
  - simpler programming model

  No, on IBM Power and GPUs
  - more scalable; allows more HW optimizations

STORE ATOMICITY

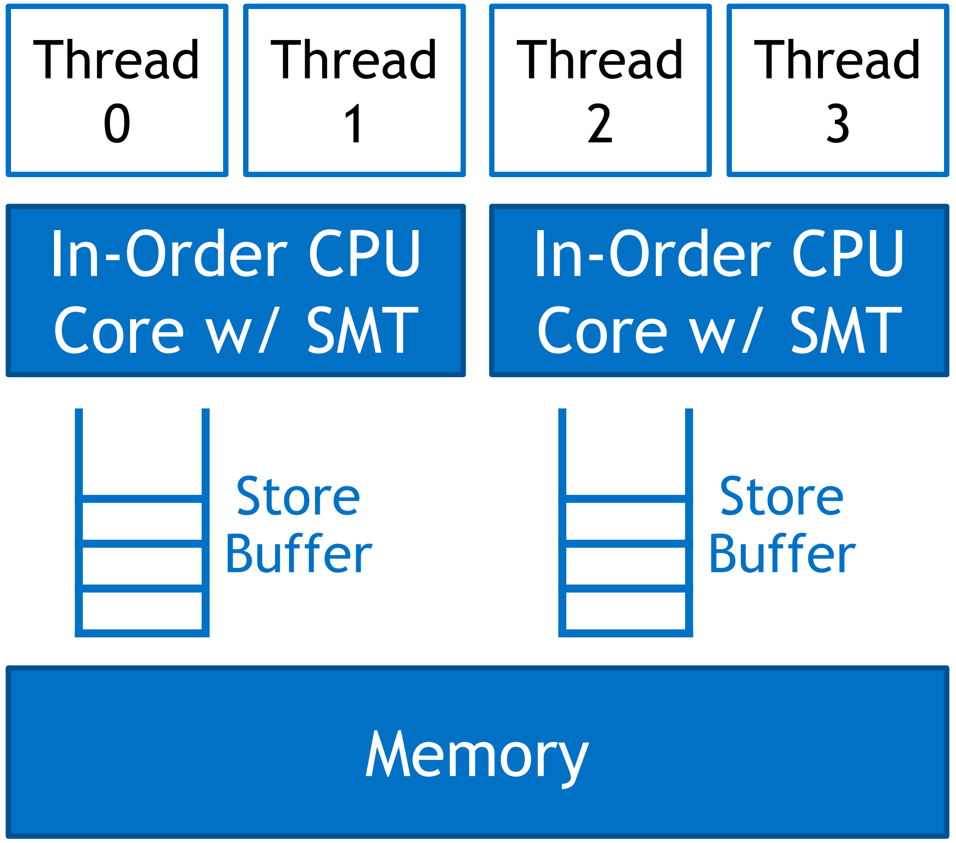
- Q: Can I think of an execution as an interleaving of the instructions in each thread (in some order), *with an exception for store buffer forwarding*?
- A: **Yes**, on x86 and ARMv8.2
  - simpler programming model

  No, on IBM Power and GPUs
  - more scalable; allows more HW optimizations

x86 and ARMv8.2 are **“(other-/weak-) multi-copy atomic”**

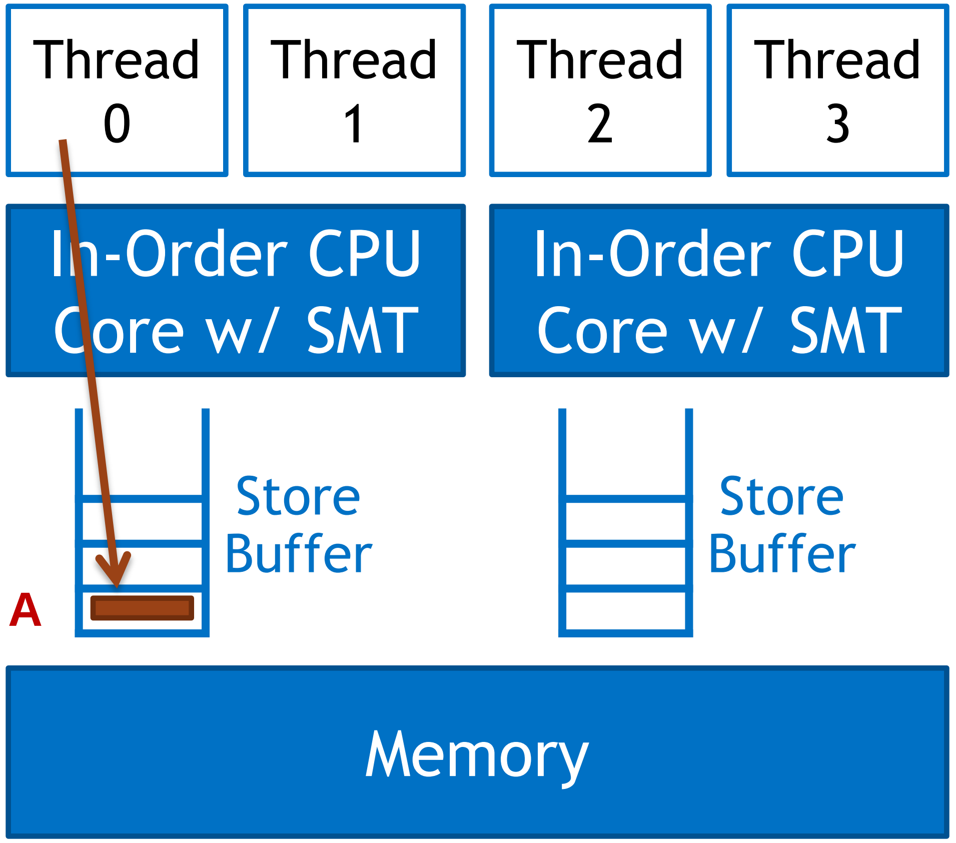
IBM Power and GPUs are not multi-copy atomic

EXAMPLE: SIMULTANEOUS MULTITHREADING



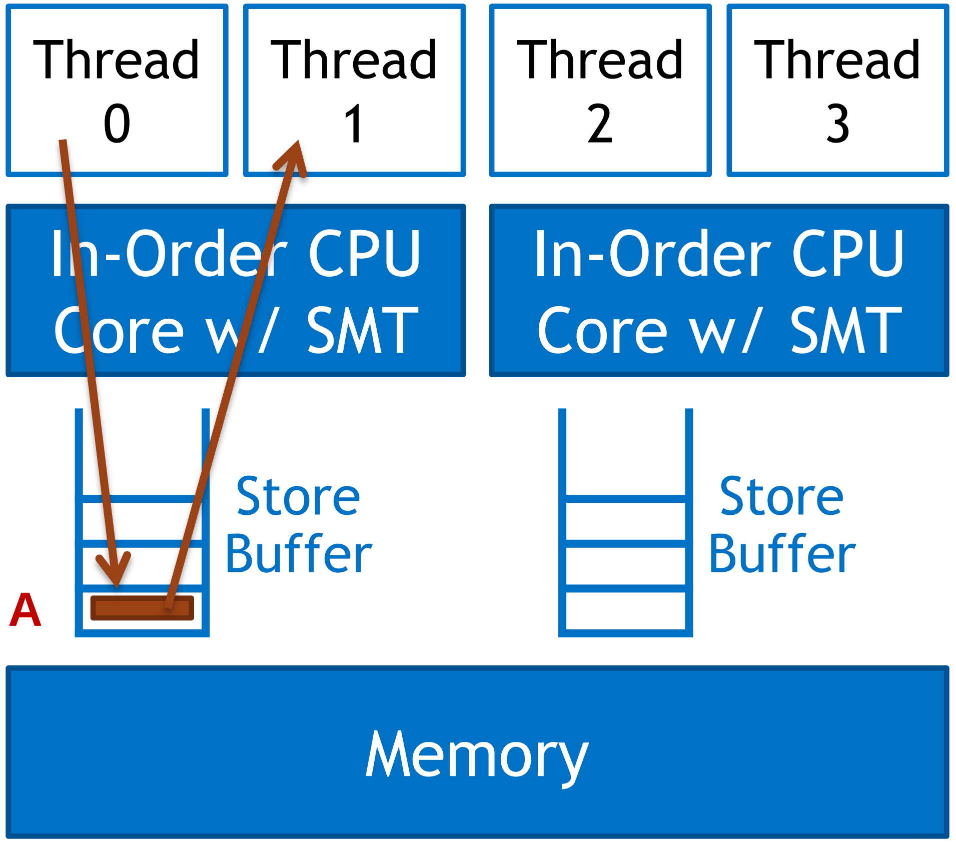
- Consider the store buffer forwarding a store value from one thread to another

EXAMPLE: SIMULTANEOUS MULTITHREADING



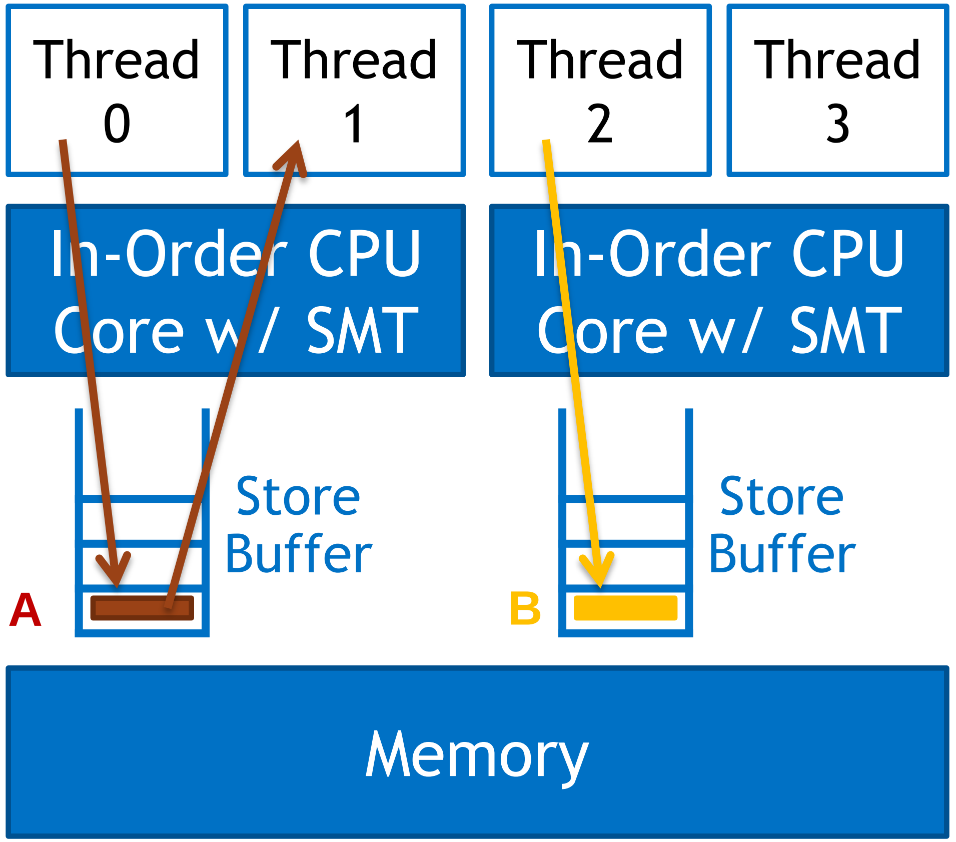
- Consider the store buffer forwarding a store value from one thread to another

EXAMPLE: SIMULTANEOUS MULTITHREADING



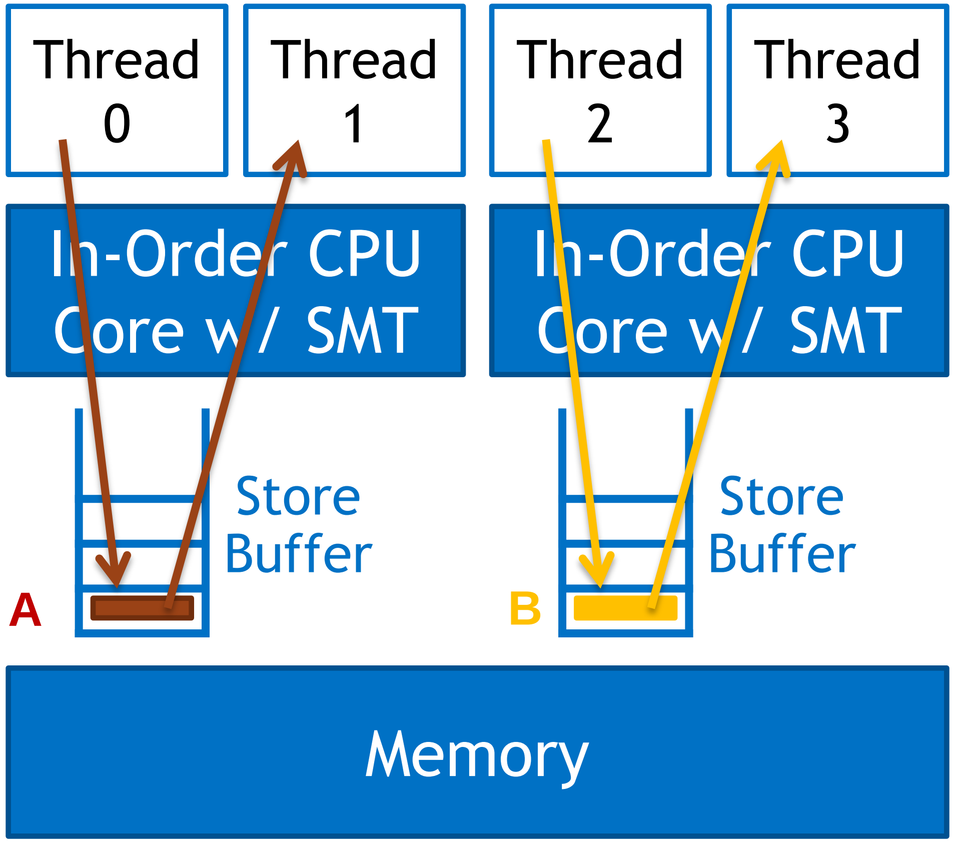
- Consider the store buffer forwarding a store value from one thread to another

EXAMPLE: SIMULTANEOUS MULTITHREADING



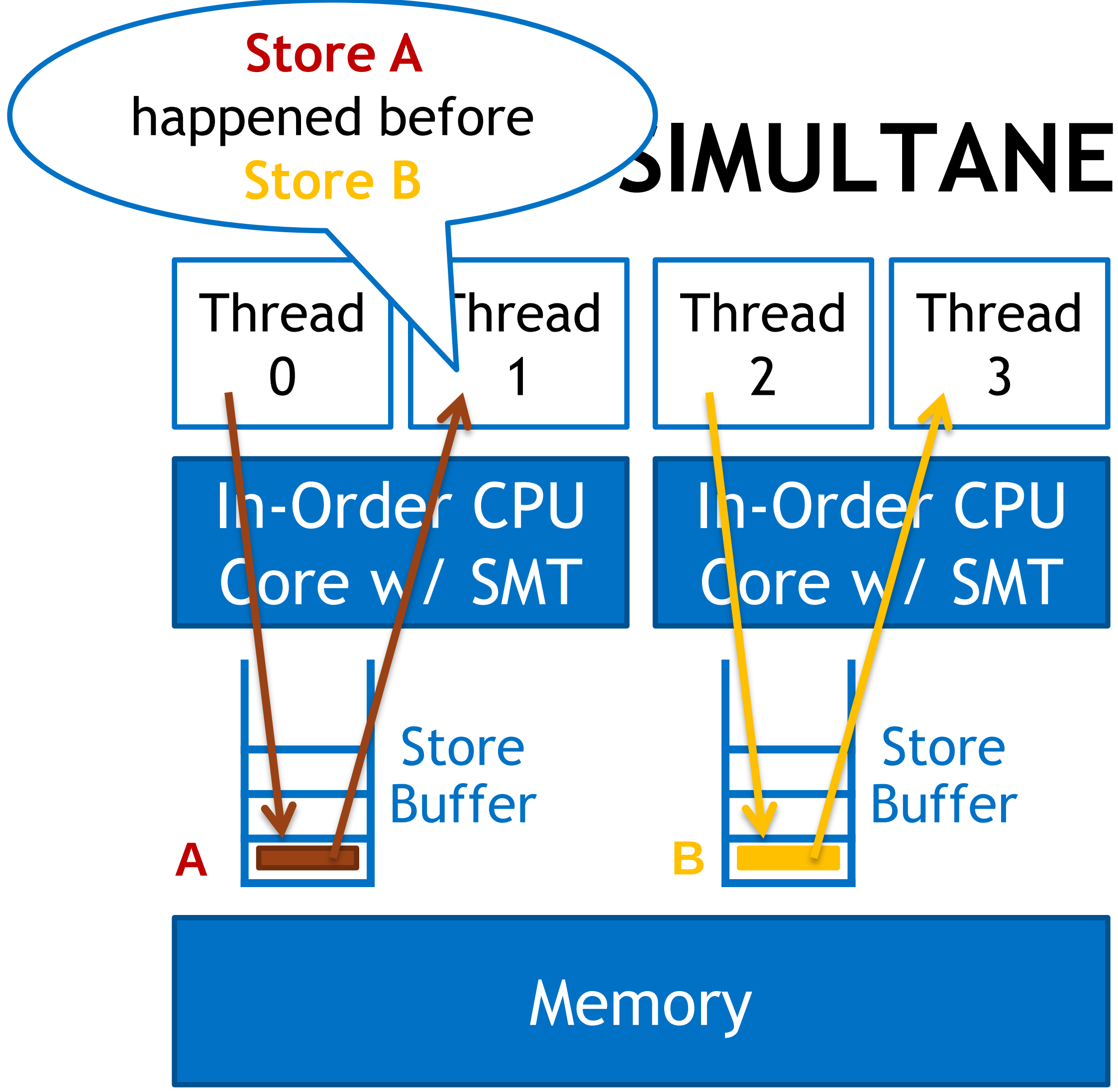
- Consider the store buffer forwarding a store value from one thread to another

EXAMPLE: SIMULTANEOUS MULTITHREADING



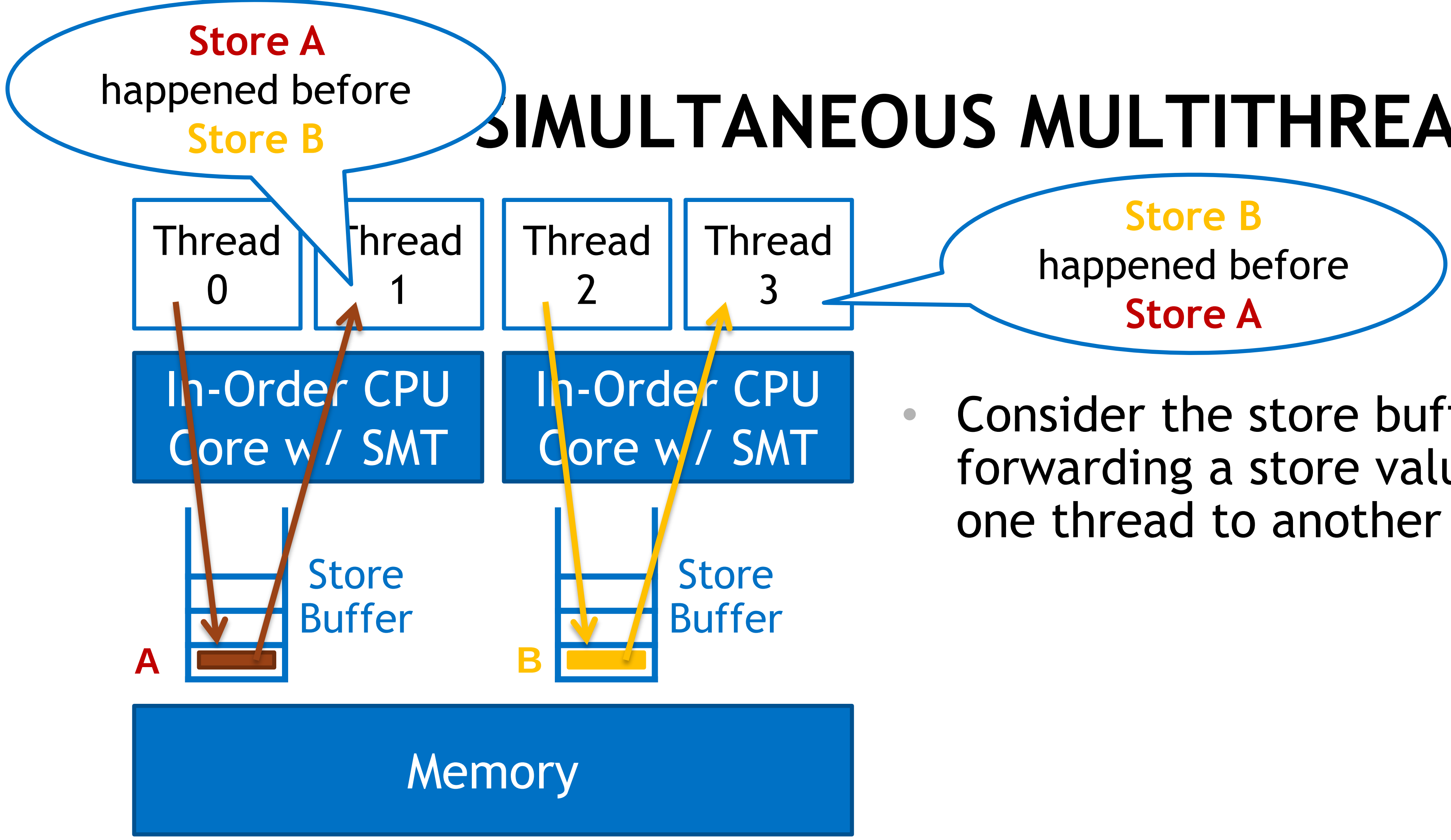
- Consider the store buffer forwarding a store value from one thread to another

SIMULTANEOUS MULTITHREADING



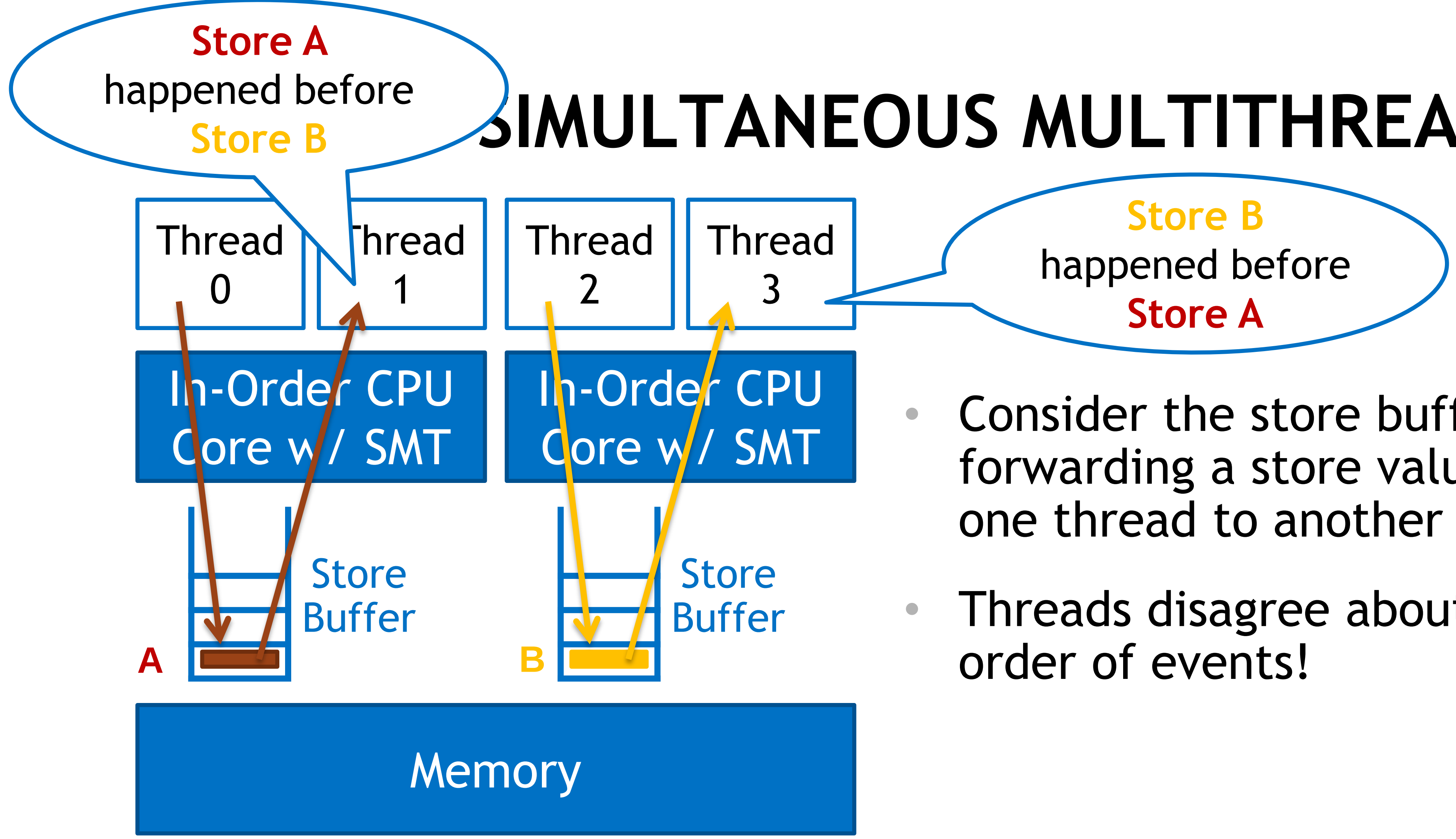
- Consider the store buffer forwarding a store value from one thread to another

SIMULTANEOUS MULTITHREADING



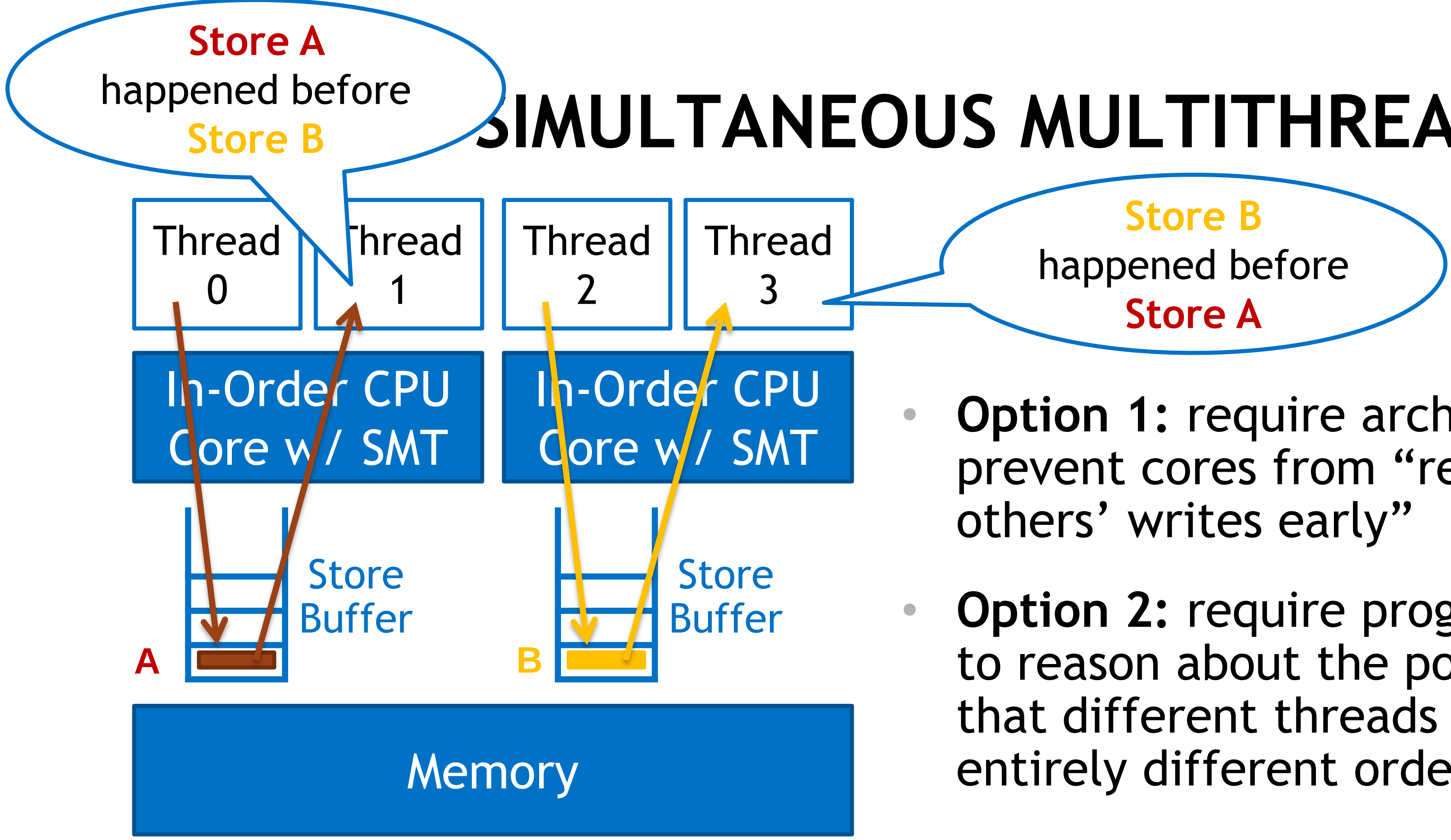
- Consider the store buffer forwarding a store value from one thread to another

SIMULTANEOUS MULTITHREADING



- Consider the store buffer forwarding a store value from one thread to another
- Threads disagree about the order of events!

SIMULTANEOUS MULTITHREADING



- **Option 1:** require architects to prevent cores from “reading others’ writes early”
- **Option 2:** require programmers to reason about the possibility that different threads see entirely different orderings

PENDING/POSSIBLE CHANGES TO THE MODEL

Feature	Status
Multi-copy atomicity	Major debate!
Enforce same-address ordering (including load-load pairs)	**Required!**
Forbid load-store reordering (for accesses to different addresses)	Still sorting out the details!
Enforce ordering of address/control/data-dependent instructions	
Which FENCE types? (.pr, .pw, .sr, .sw? Other?)	

HOW DOES THE MODEL AFFECT YOU?

- Programmers: it doesn’t, unless you’re writing assembly
- Compiler writers: this is really important! Let’s talk!
- Architects of simple cores/SoCs: this shouldn’t affect you, but if you get more aggressive, check back in
- Architects of high-performance cores/SoCs: this will affect how aggressive you can be, and will determine how much complexity can/can’t get exposed beyond the ISA. Let’s talk!

IT’S ALWAYS SAFE TO BE CONSERVATIVE

- **If your architecture is simple and conservative (in-order pipeline, simple memory design, etc.), it will be compliant with any model we’ll use**
  - e.g., if the model chooses to allow non-atomicity, your implementation can still safely be multi-copy atomic
  - e.g., if you want to ignore the .pr, .pw, .sr, and .sw fence bits, and just always do a full fence, that’s fine too
- The memory model committee will publish more specific and concrete guidance

CONCLUSIONS

- RISC-V memory model details are still being worked out
  - Expected timeline: months, not years
- These details largely only affect more aggressive future implementations; <u>today's designs are unaffected</u>
- Memory model committee will deliver spec + guidance
- If you're considering an aggressive design, or just want to get clarification or more detail, come talk to us!

dlustig@nvidia.com