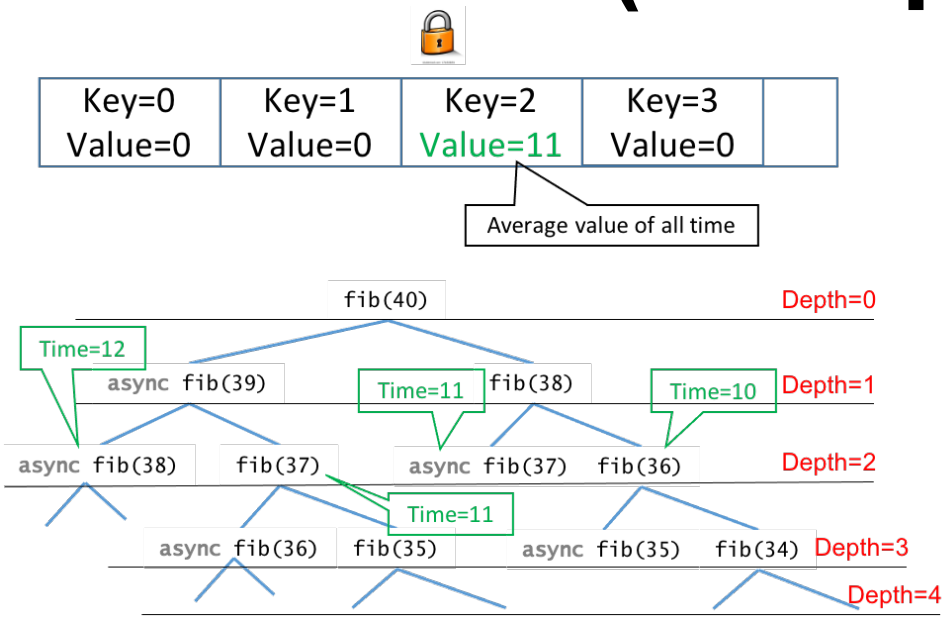
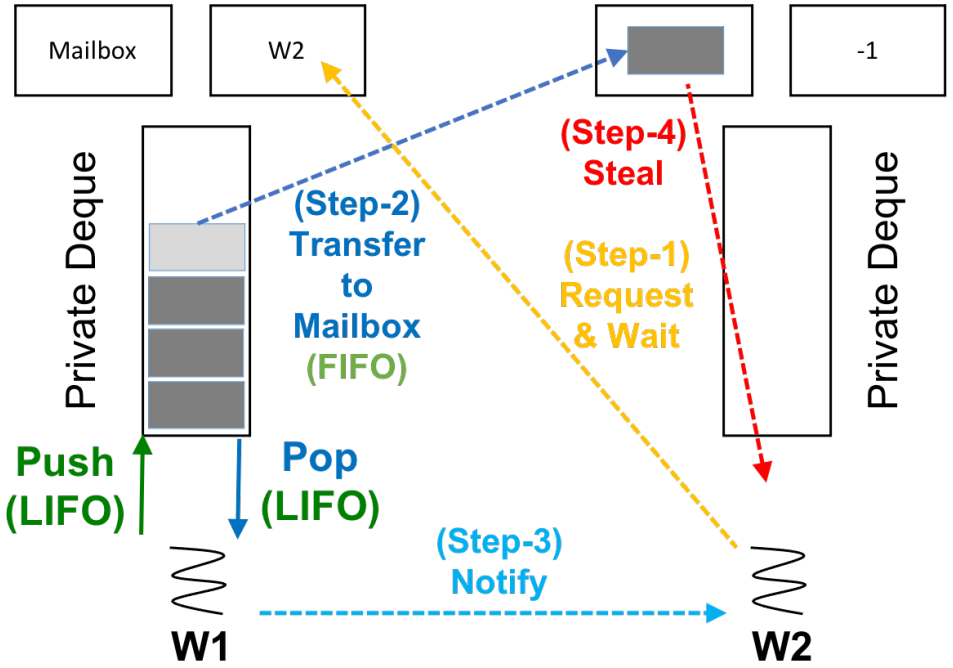
Lecture 08: Introduction to Memory Consistency

Vivek Kumar
Computer Science and Engineering
IIIT Delhi
vivekk@iiitd.ac.in

Last Lecture (Recap)



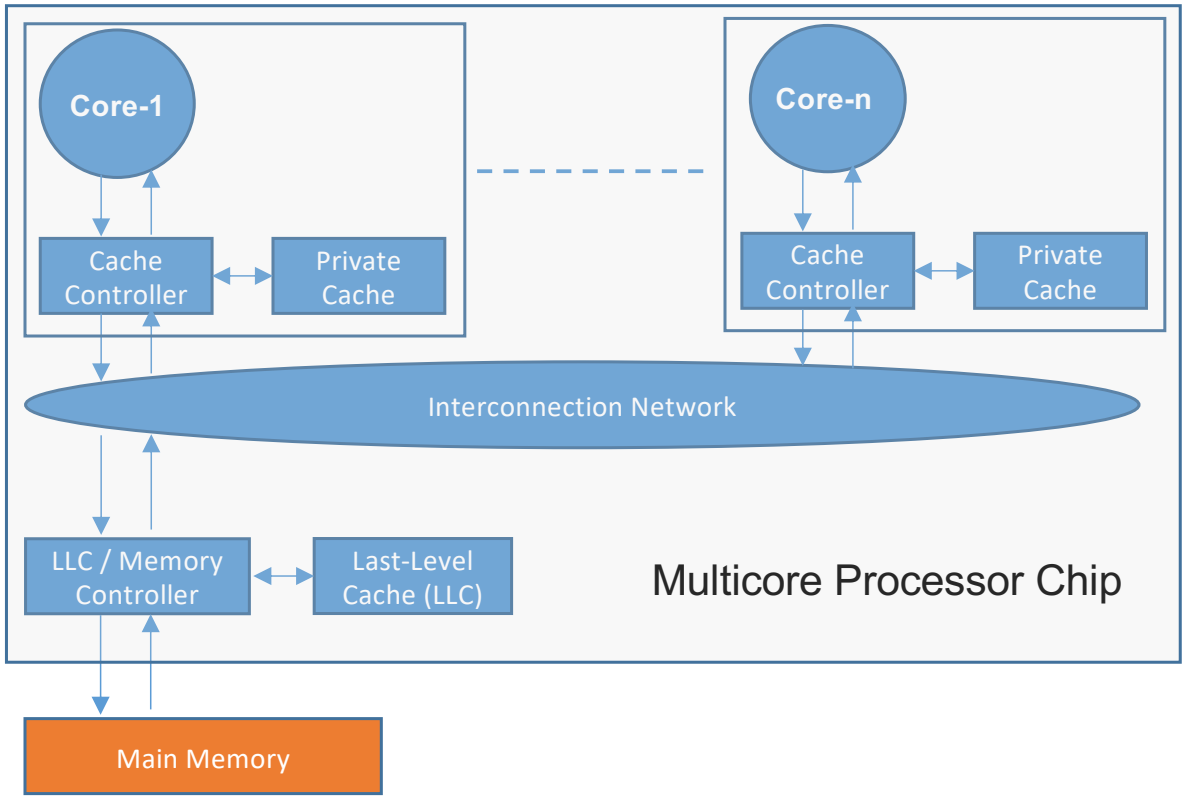


- Reducing concurrent accesses to private deque
- Automatically controlling task granularity

Today's Class

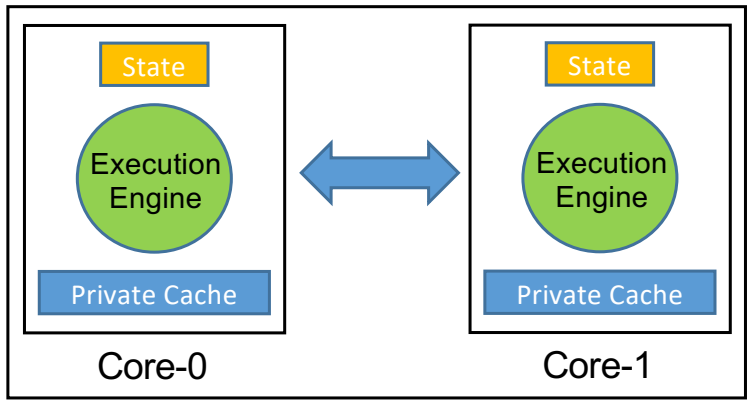
- Memory consistency problem
- Difference between coherence and consistency
- Sequential consistency

Multicore System Overview

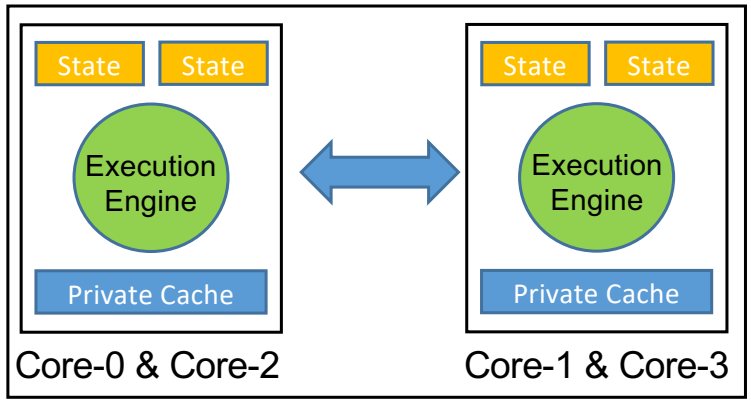


- Cache controller copes code/data to/from main memory
  - It runs the actual cache coherence algorithm
- LLC is shared with all the cores
- LLC is logically a memory side cache as it primarily serves to reduce the latency and increase bandwidth of memory accesses

Physical VS. Logical Cores



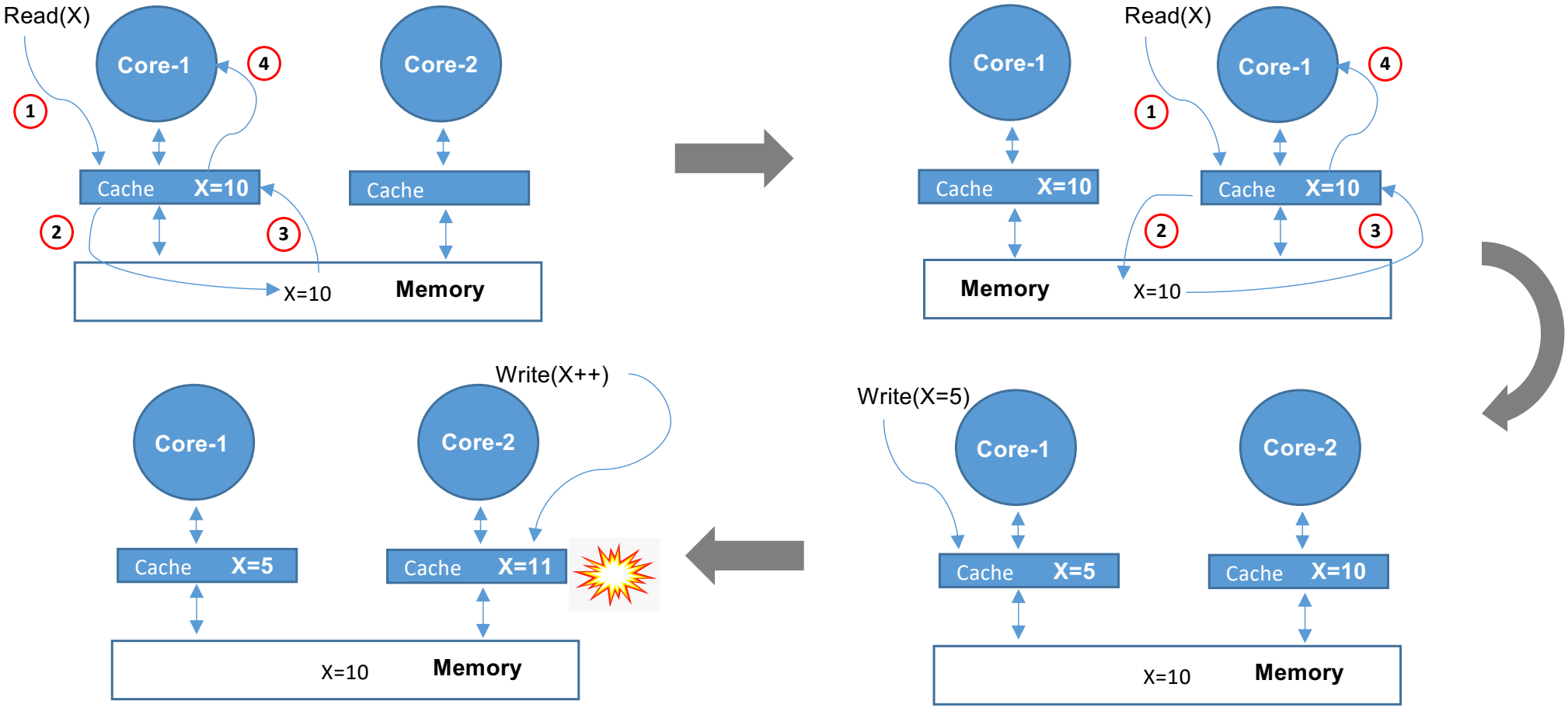
Dual-core processor with hyperthreading **DISABLED**



Dual-core processor with hyperthreading **ENABLED**

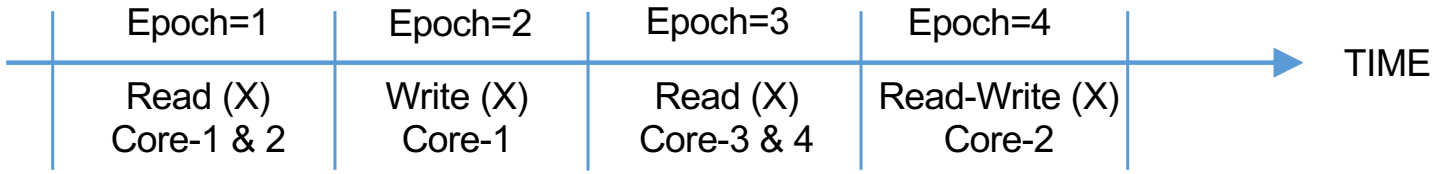
- Architectural state of a core are the registers (EBP, ESP, EIP, etc.)
- Logical cores of a processor share
  - Private cache
  - Execution engine
  - System bus interface
- If the execution of one of the logical core blocks (e.g., Core-0 waiting for a memory fetch from the DRAM) then the other logical core (Core-2) can resume its execution with its own state

The Cache Coherence Problem



Cache Coherence (CC)

- CC protocols gives the illusion of an atomic memory system without any caches
- CC definition / requirements
  - Single-Writer-Multiple-Reader (**write serialization**)
    - At any given instant only one core can WRITE or WRITE+READ a given memory location, or multiple cores can only READ that location
  - Value of a memory location is ***eventually*** propagated to all readers (**value propagation**)



Dividing a memory location's lifetime into epochs

Question

- Assume that for each memory locations in a CC system, there are N tokens, where N is the total number of cores
  - How to define memory access rule for a memory location X using these tokens?
    - If a core has all of the tokens, it may write the memory location. IF a core has one or more tokens, it may read the memory location

Multithreaded Programs

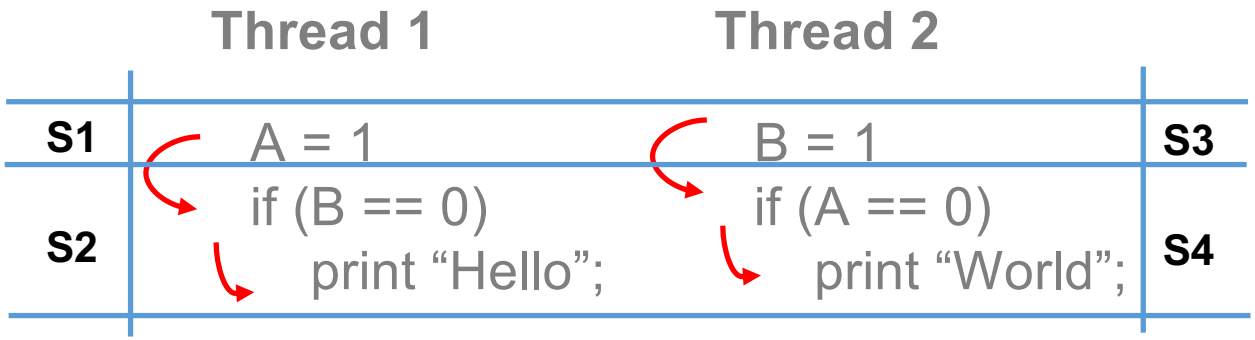
Initially A = B = 0

Thread 1	Thread 2
A = 1	B = 1
if (B == 0)	if (A == 0)
print “Hello”;	print “World”;

Question: What can be printed: “Hello”, “World”, or “Hello World”?

Why Memory Consistency Model?

- What does programmer think?
  - Cores atomically executes one instruction at a time in **program order**
    - Valid over a single core (sequential) processor



Let’s see if we can reproduce it on our machine

Question: According to you as a programmer, what are the valid program orders here?

1. S1 – S2 – S3 – S4
2. S3 – S4 – S1 – S2
3. S1 – S3 – S2 – S4
4. S2 – S1 – S3 – S4
5. S1 – S3 – S4 – S2
6. S2 – S1 – S4 – S3

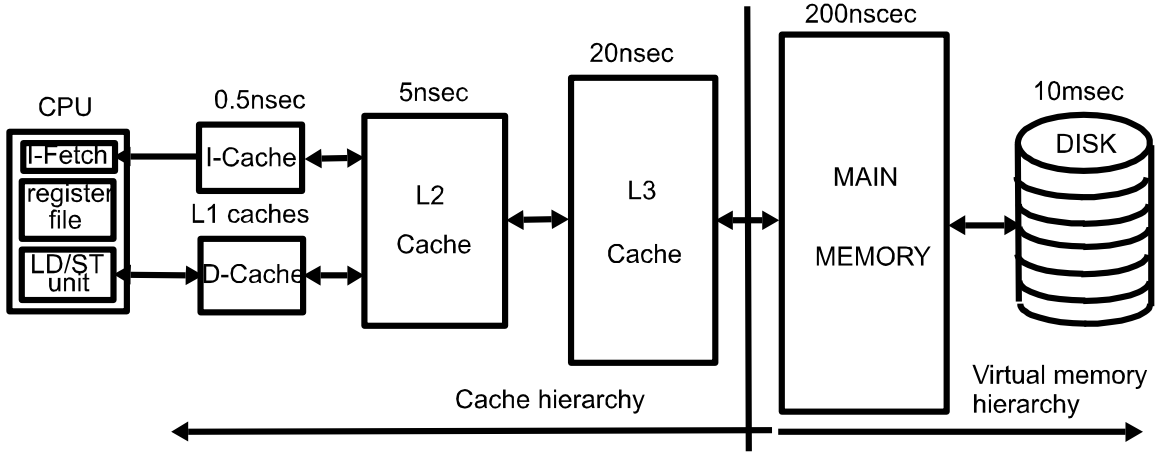
Why Memory Consistency Model?

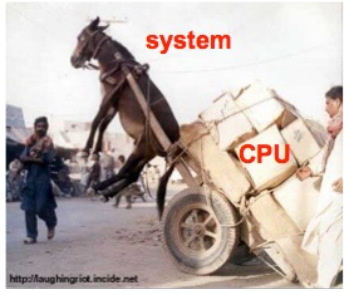
- What does programmer think?
  - Cores atomically executes one instruction at a time in **program order**
    - Valid over a single core (sequential) processor
- What cores can do?
  - Reorder memory operations to reduce memory access latency
    - Read-Read
    - Write-Write
    - Read-Write
    - Write-Read
- Memory consistency model defines what it means to “read a memory”, “write at a memory”, and the ”respective ordering”

Today's Class

- Memory consistency problem
- Difference between coherence and consistency
- Sequential consistency

Typical Memory Hierarchy

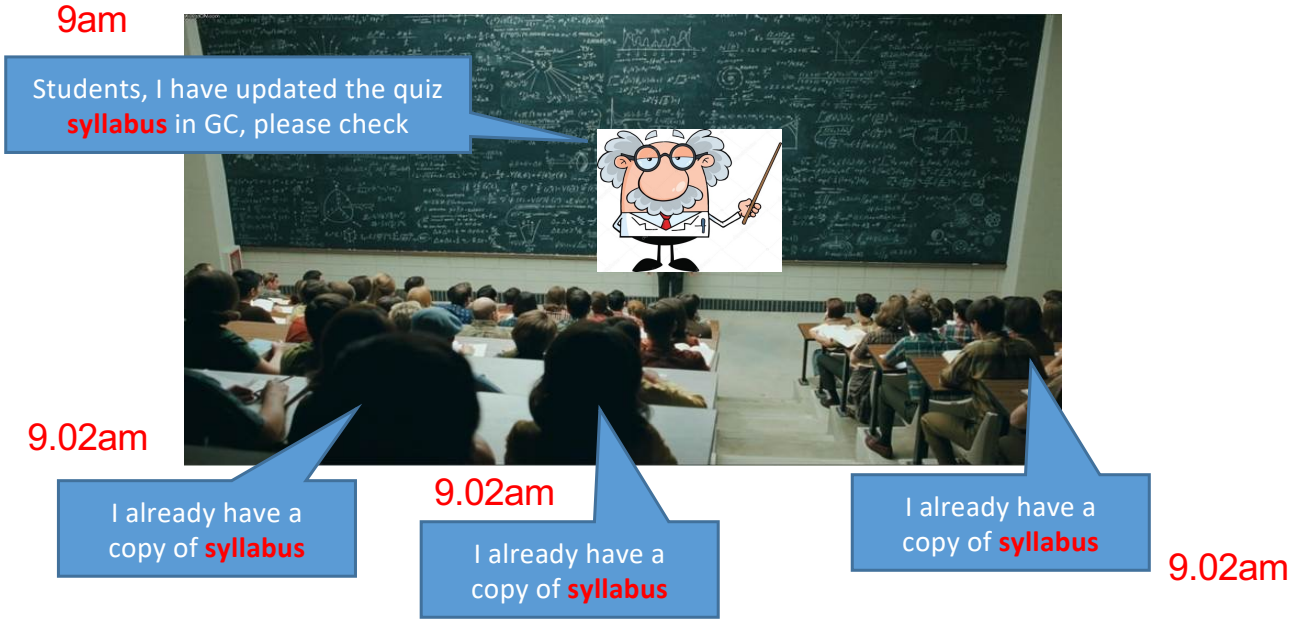




- Memory latency continues to limit the performance of modern out-of-order cores
  - Loads and Stores are very expensive
  - Instruction pipelining is one simple technique for improving memory latency

- Hiding Read (Load) latency
  - Prefetching loads that are delayed due to consistency model
  - Out-of-order execution of loads
    - Speculative execution to allow the processor to proceed even though consistency model requires memory accesses be delayed (e.g., Intel, AMD)
    - Further relaxing the memory consistency model (e.g., ARM, IBM)
- Hiding Write (Store) latency
  - Store Buffers – available on almost all modern processors (more on this later)

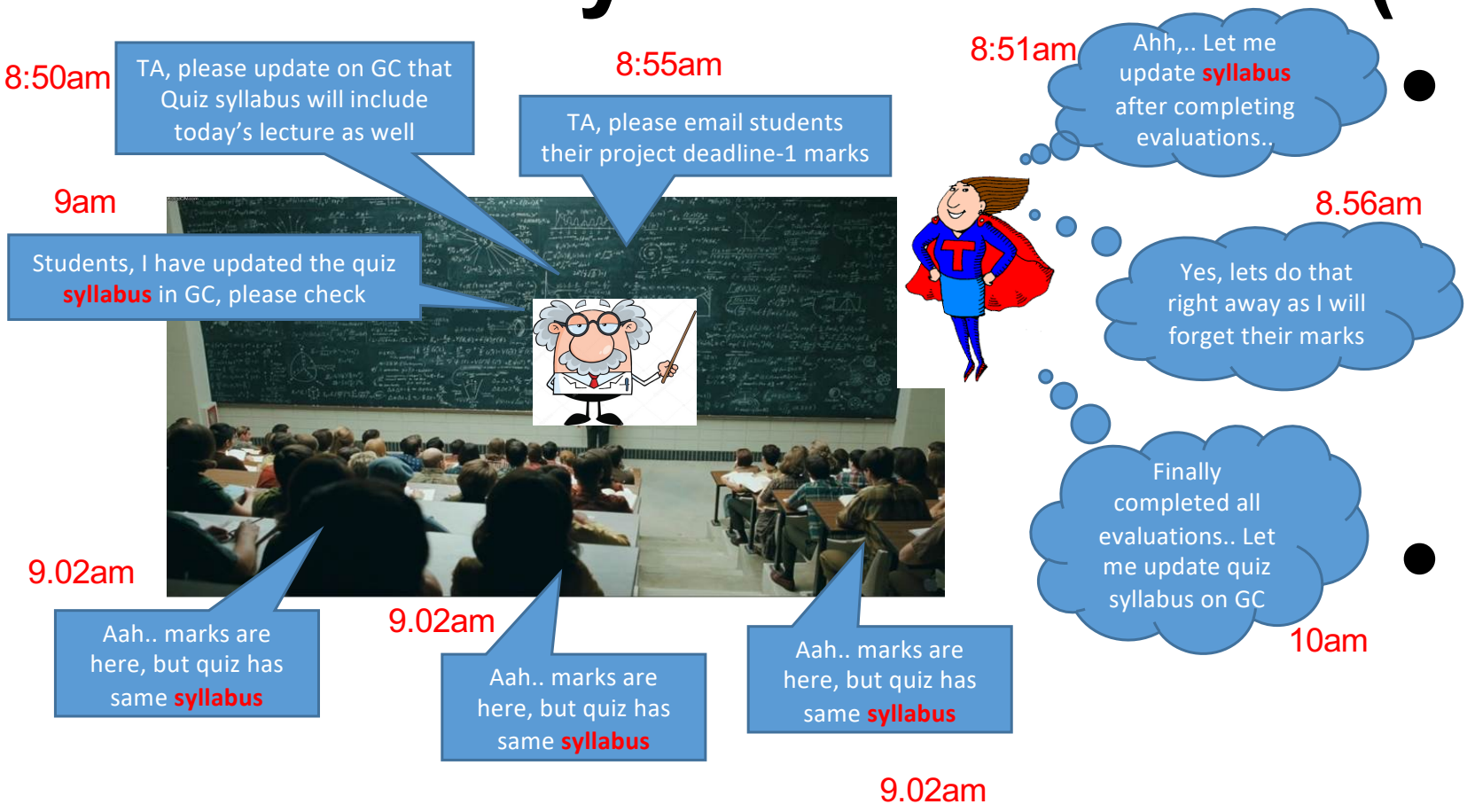
Consistency VS. Coherence (Example)



- Coherence assures that values written by one processor are made visible to other processors

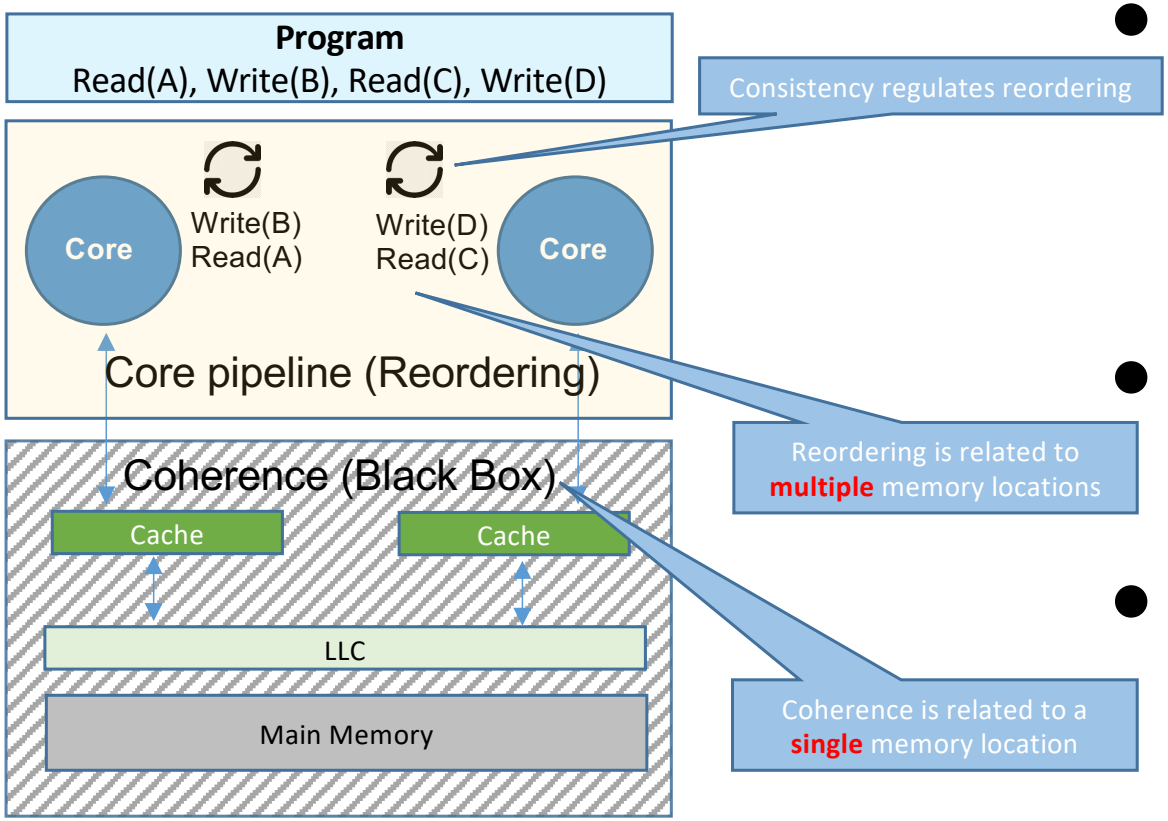
Each student has a stale copy of shared variable “syllabus”. They will **eventually** get the updated copy of “syllabus” due to coherence’s value propagation property

Consistency VS. Coherence (Example)



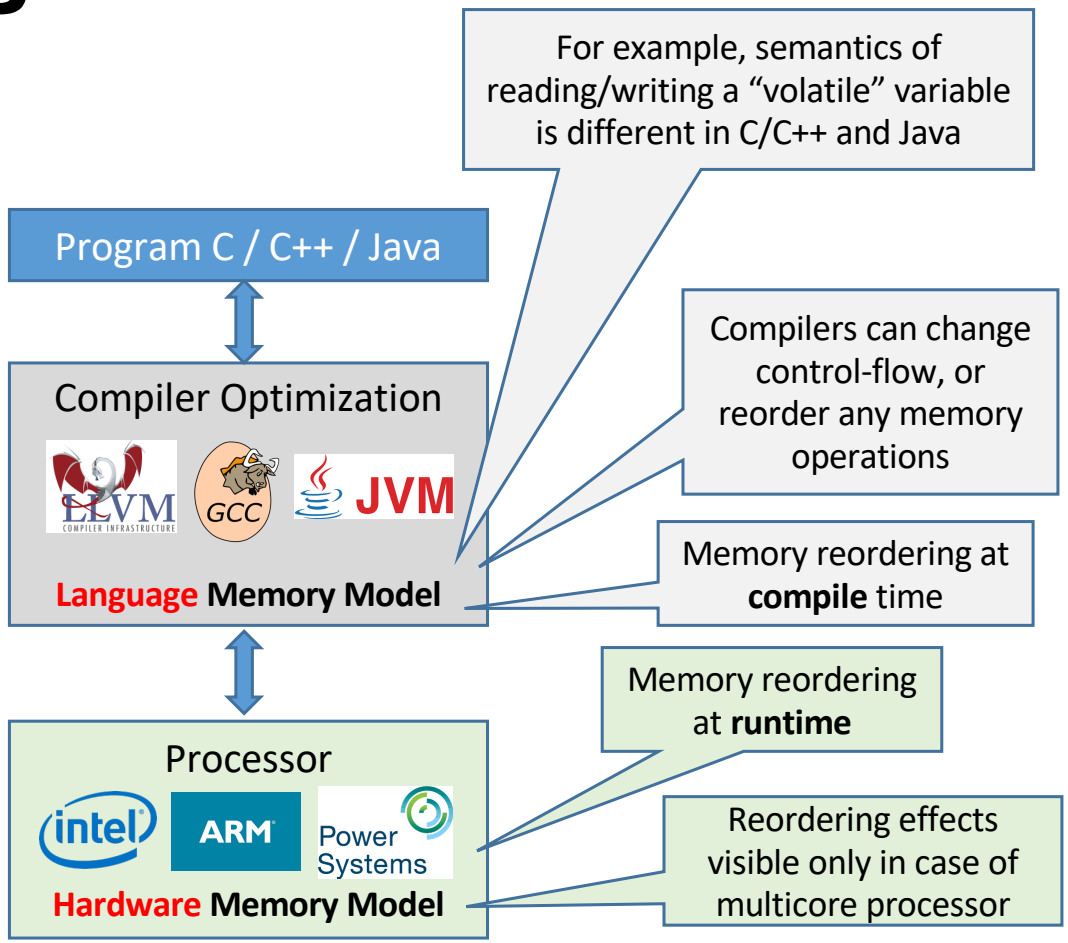
- Consistency insures that writes to different locations will be seen in an order that makes sense, given the source code
- Coherence insures that writes to a particular location will be seen in order

Consistency VS. Coherence

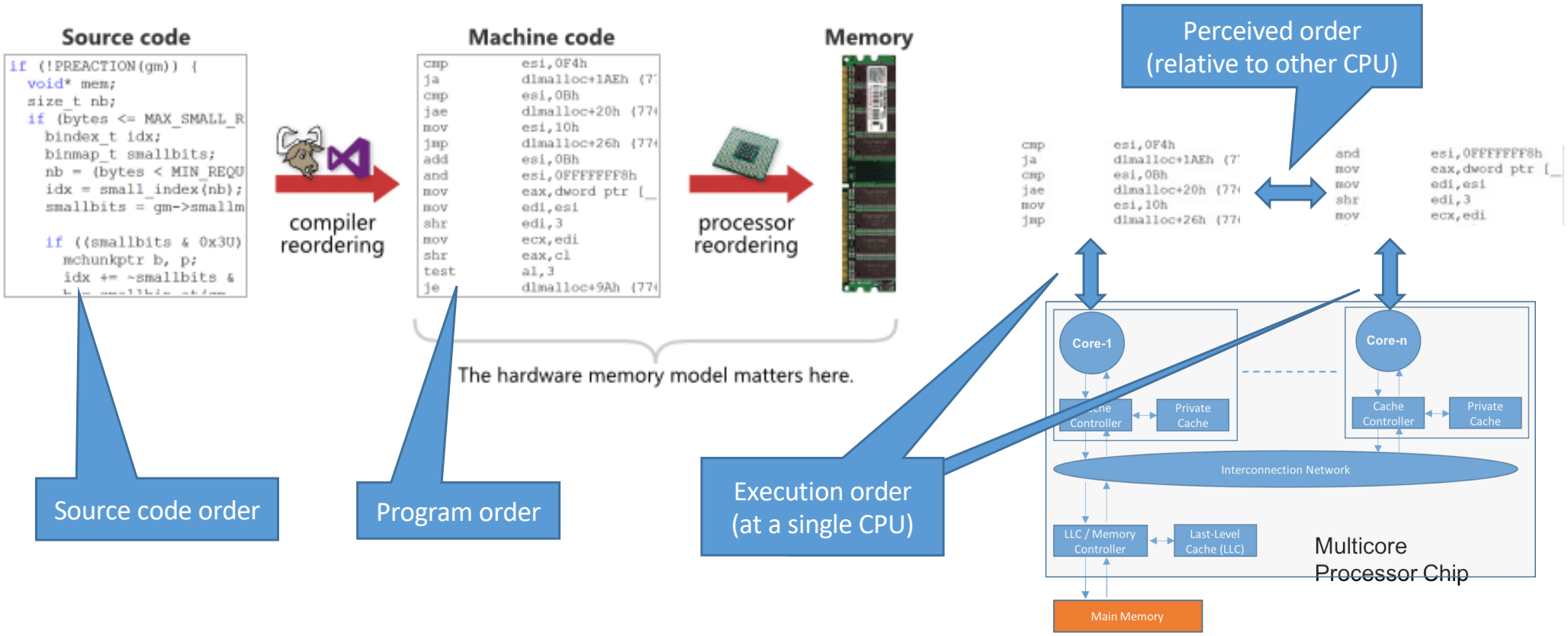


- Coherence is not equal to memory consistency
  - Core pipeline uses coherence as an abstraction on memory system
  - Specific to a single shared memory location
- Core pipeline can present memory operations to coherence system in an order different than the program order
- Memory consistency uses cache coherence as a useful black box
  - Related to all shared memory locations

Language v/s Hardware Memory Model



Memory Ordering at Different Stages



Picture (1) source: https://preshing.com/images/hardware-matters.png

Today's Class

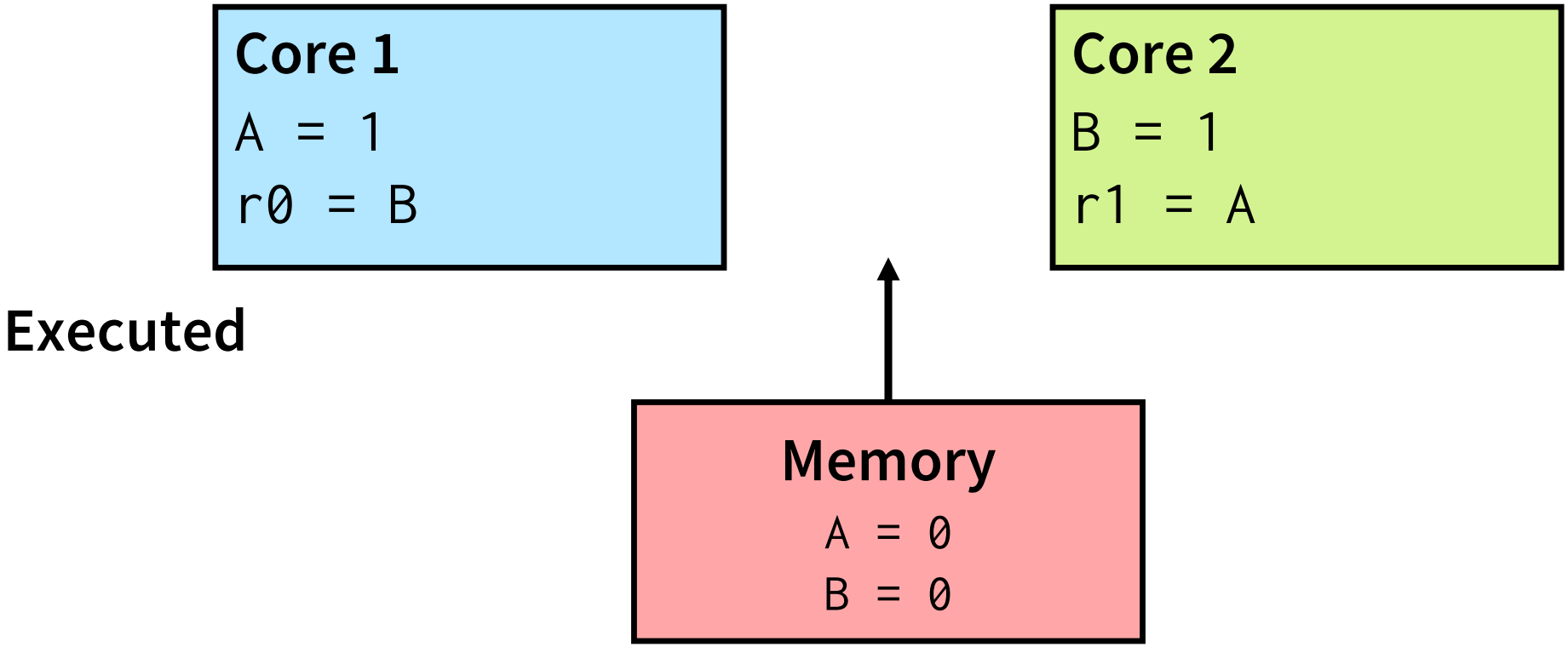
- Memory consistency problem
- Difference between coherence and consistency
- Sequential consistency

Sequential Consistency

- Programmer's productivity is high if the memory operations in his parallel program executes in the program order
  - Sequential consistency
    - Lamport defined SC (1979)
- Let's assume there is a processor that supports sequential consistency to provide high productivity for the programmer
  - Although it suffers on performance
- What that processor has to do for supporting sequential consistency?

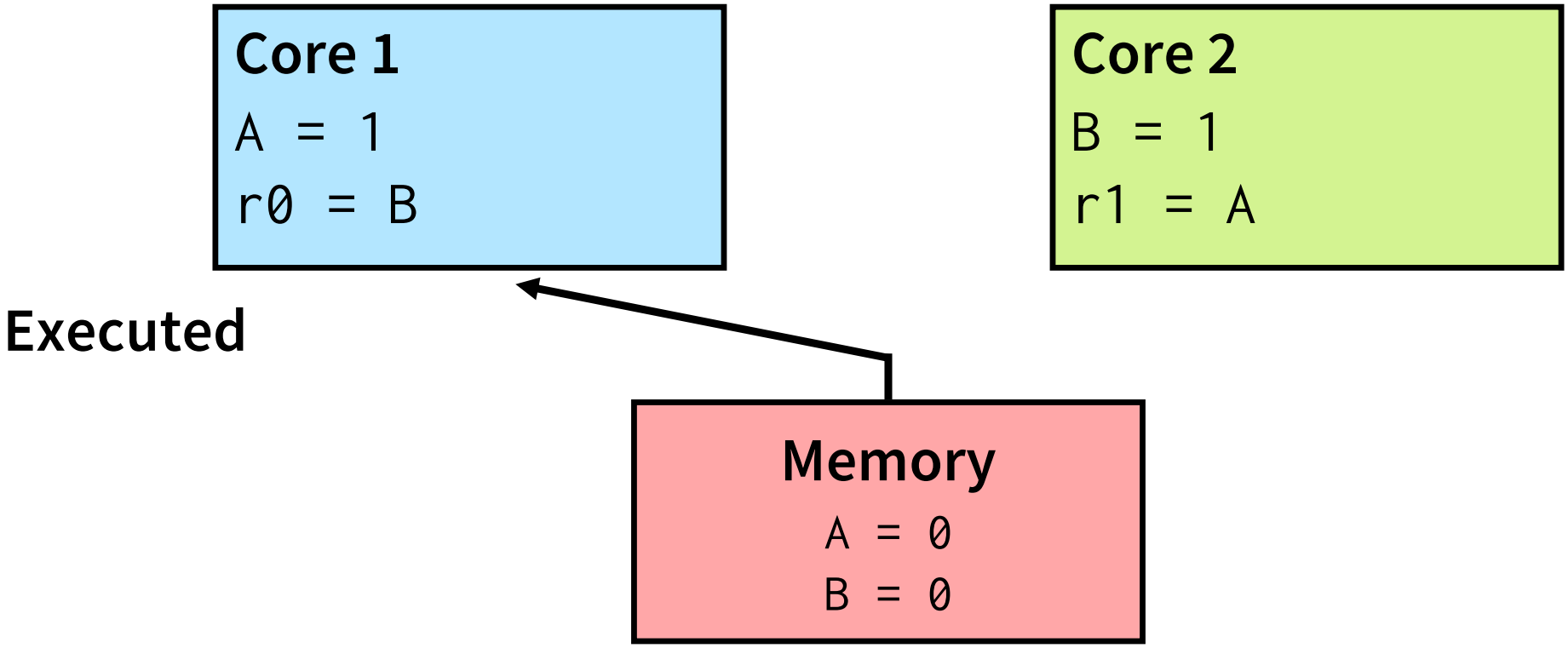
Sequential Consistency

Can be seen as a ”switch” running one instruction at a time



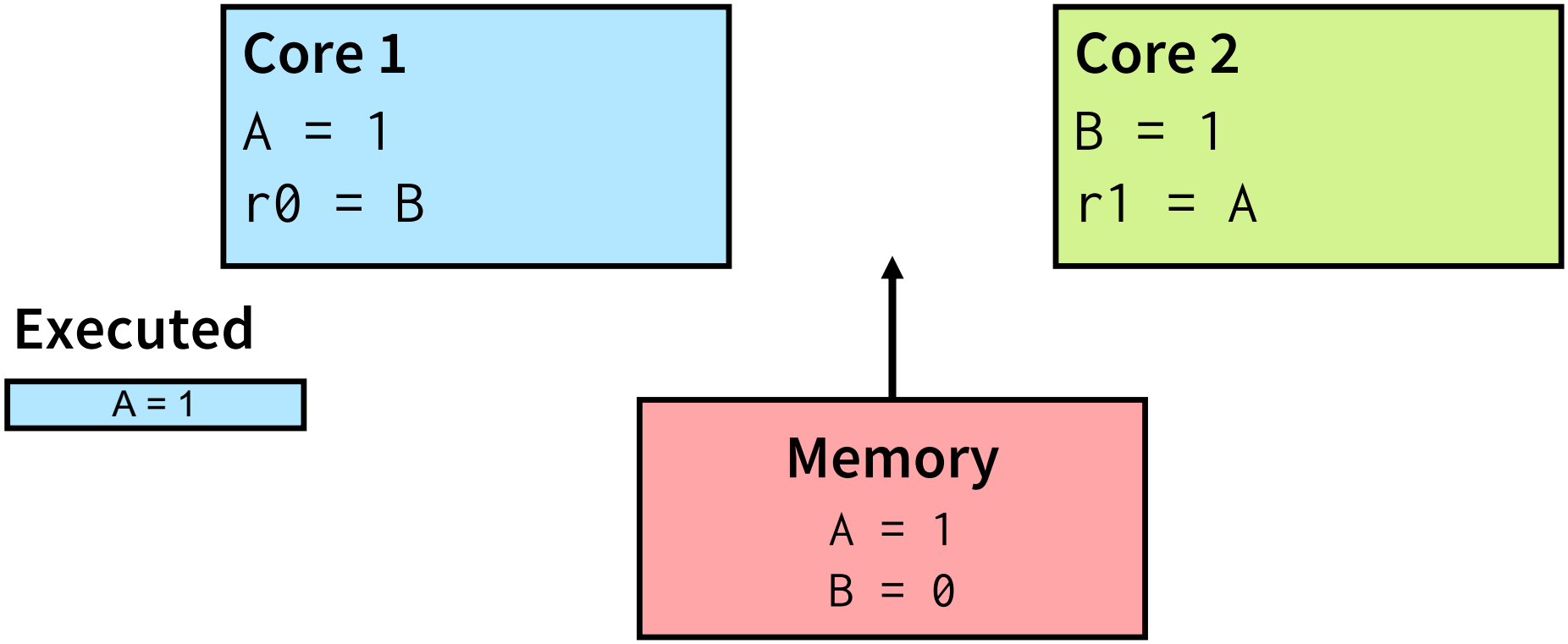
Sequential Consistency

Can be seen as a ”switch” running one instruction at a time



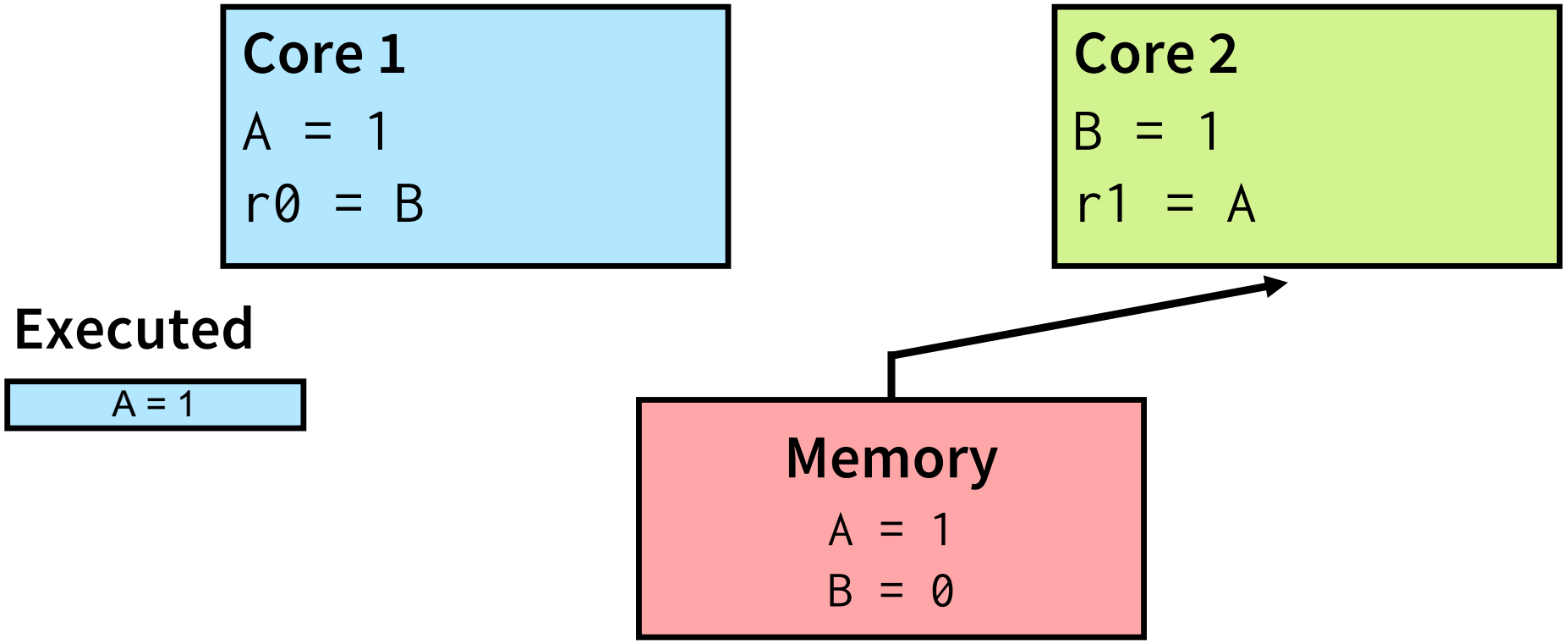
Sequential Consistency

Can be seen as a ”switch” running one instruction at a time



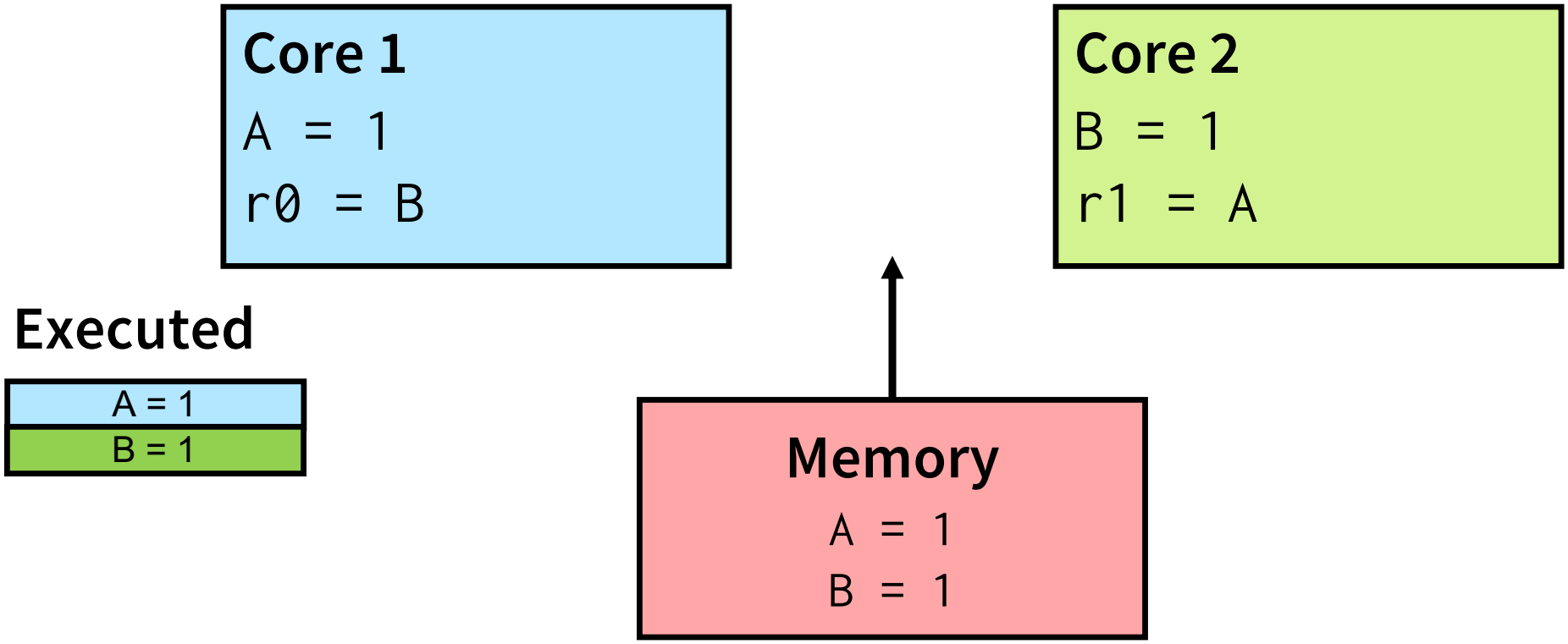
Sequential Consistency

Can be seen as a ”switch” running one instruction at a time



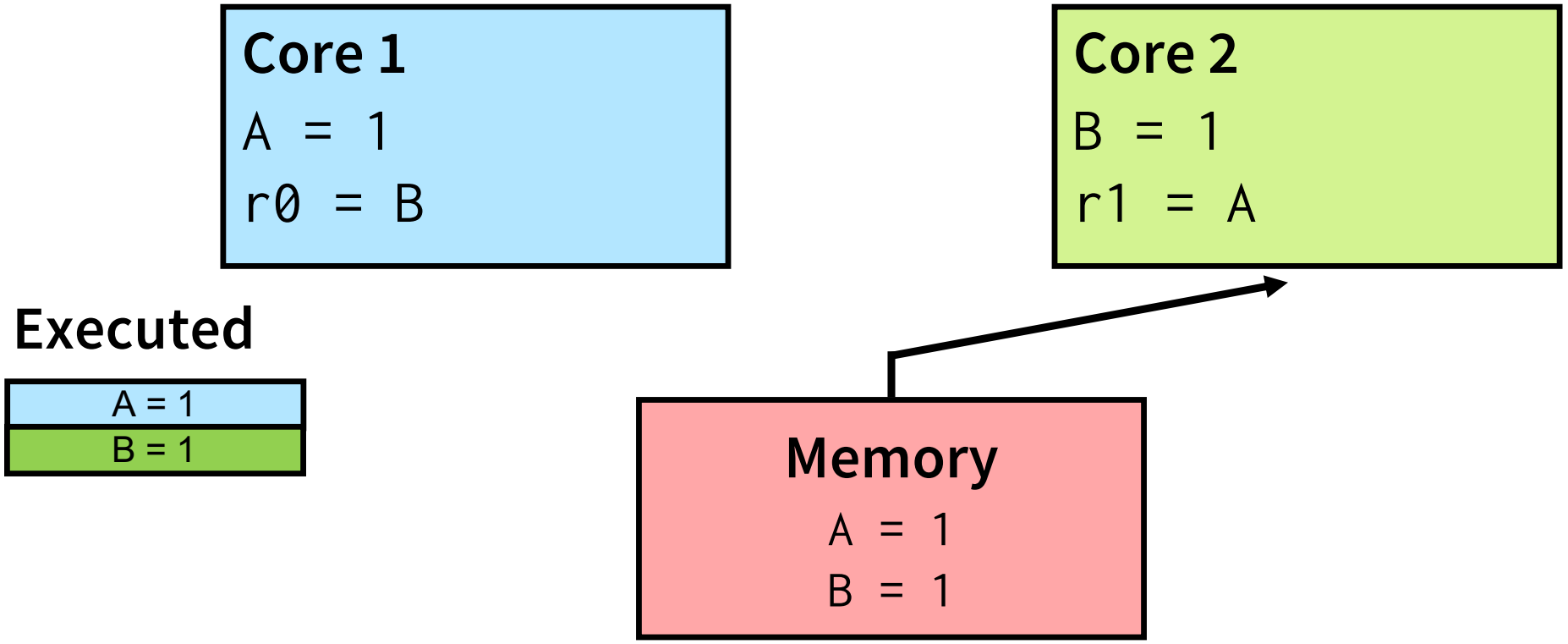
Sequential Consistency

Can be seen as a ”switch” running one instruction at a time



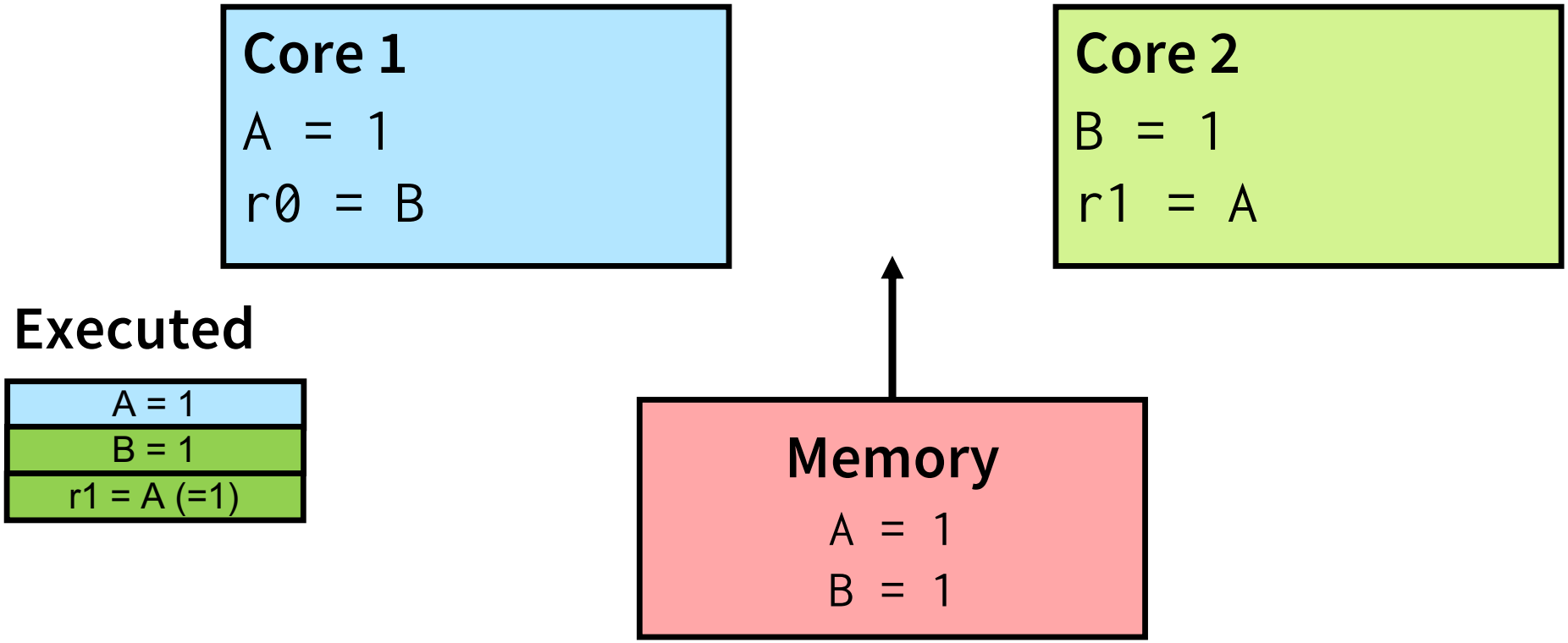
Sequential Consistency

Can be seen as a ”switch” running one instruction at a time



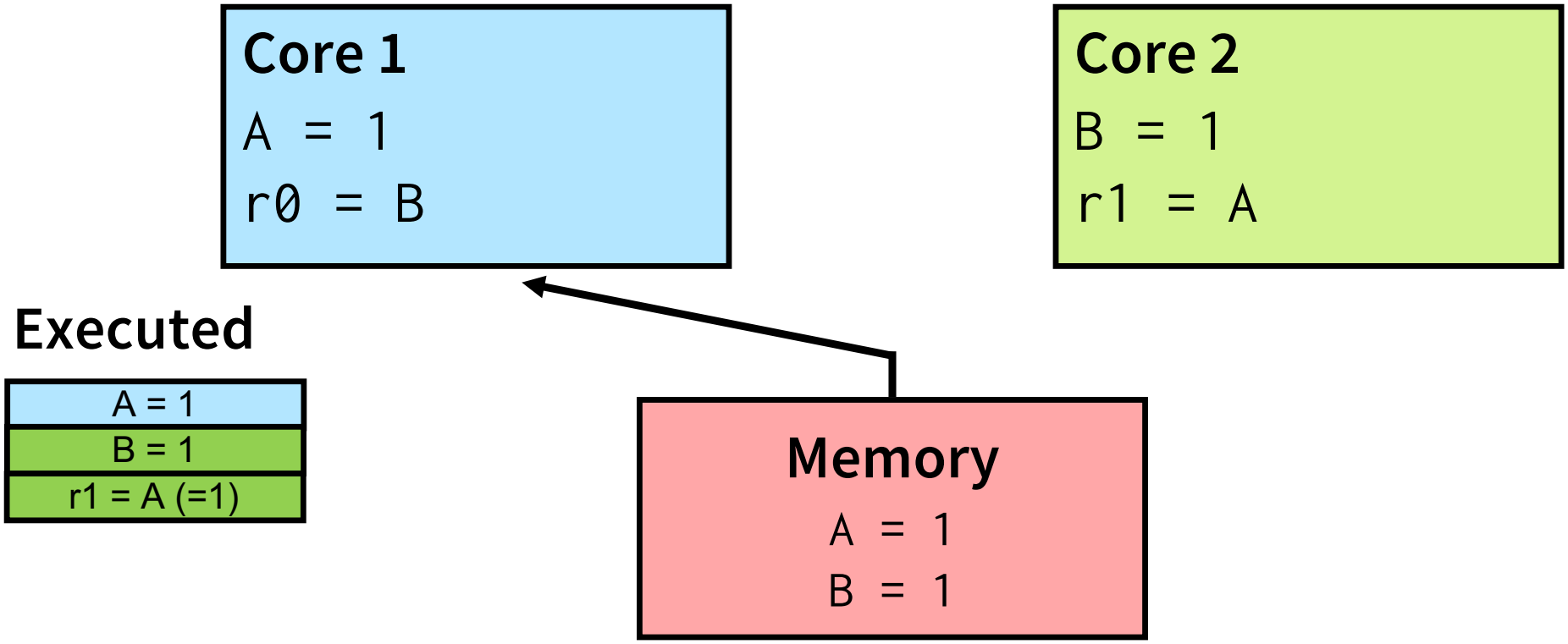
Sequential Consistency

Can be seen as a ”switch” running one instruction at a time



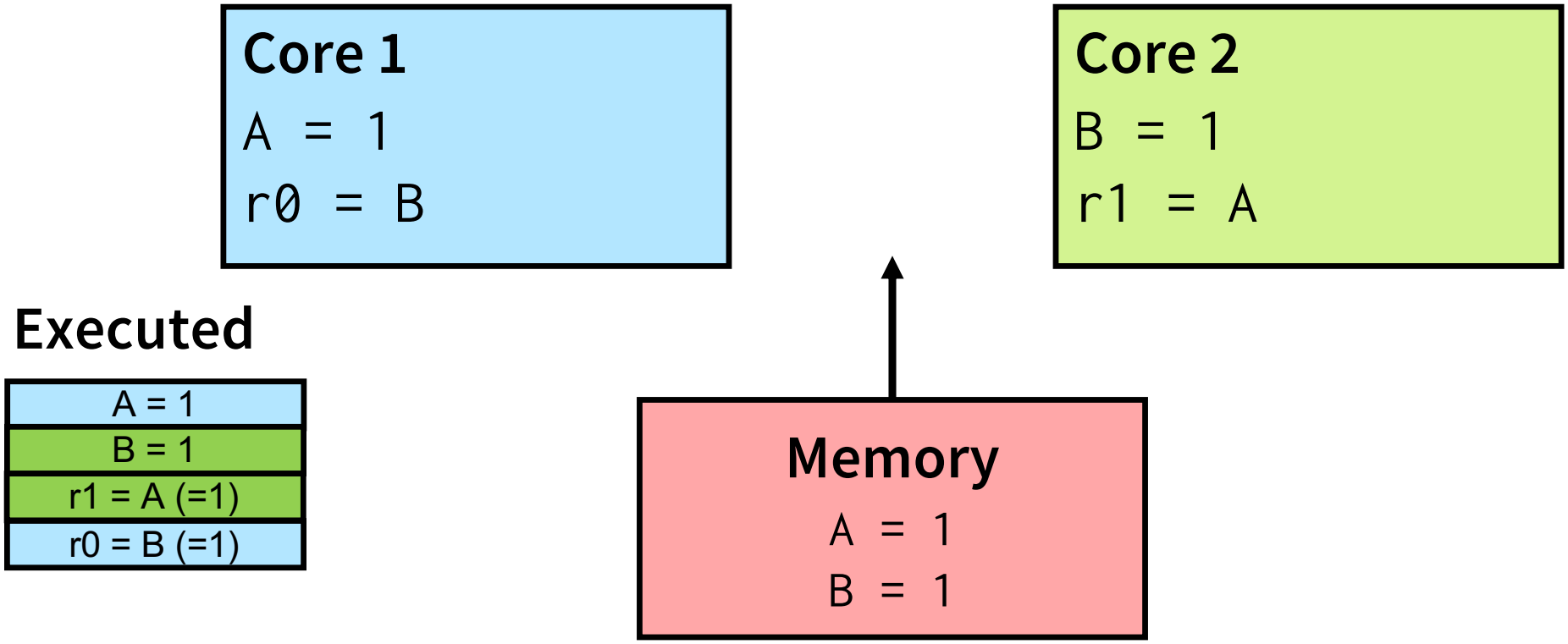
Sequential Consistency

Can be seen as a ”switch” running one instruction at a time



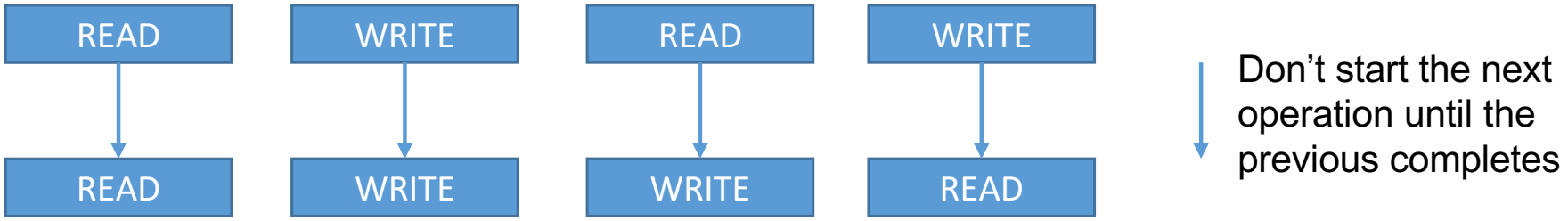
Sequential Consistency

Can be seen as a ”switch” running one instruction at a time

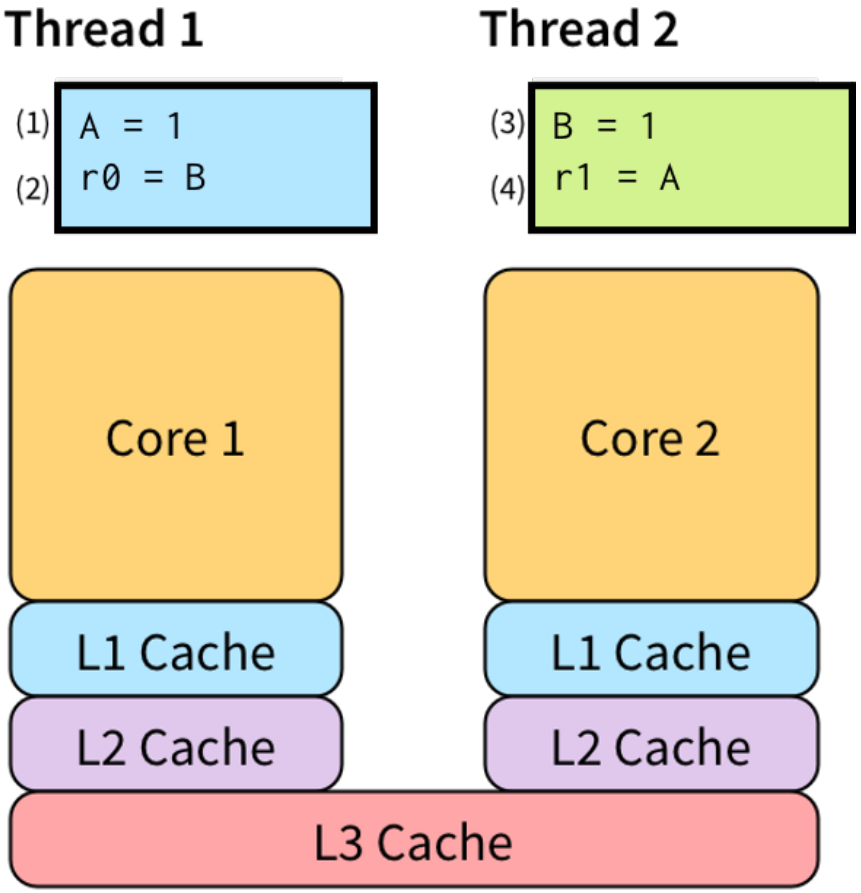


Summary: Sequential Consistency

- Each core execute its read/write in the program order
  - Irrespective of whether they are at same or different addresses
- Every read from address A gets its value from the last write before it to the same address A
  - Operations must appear atomic—one operation must complete before the next one is issued



The Problem with SC



- With a single view of memory, we can't run (2) until (1) has become visible to every other thread
  - On a modern CPU, that's a very expensive operation due to the cache hierarchy
- The only shared memory between the two cores is all the way back at the L3 cache, which often takes upwards of 90 cycles to access

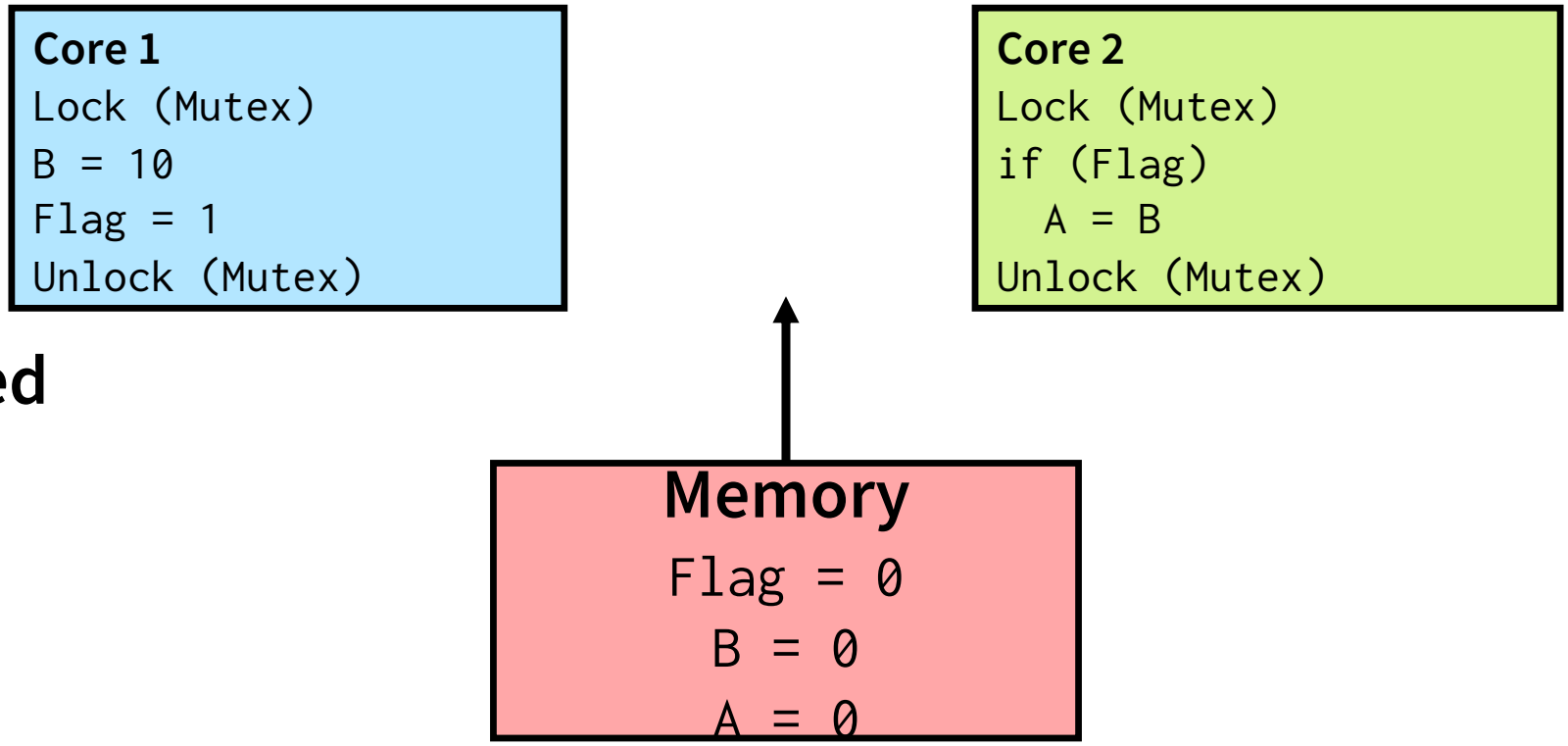
Sequential Consistency Supported!

- Is there support for sequential consistency in modern programming language?
  - Yes, **but** only for **one** particular case in almost every language!
    - Only for code block that is Data Race Free (DRF)
    - Even without using any special construct, you get it for free inside critical sections that are executed within mutex lock/unlock operations
      - We will see in lecture 10 about other features in C++11 to support DRF
  - No sequential consistency for rest of the program (racy code!)

Sequential Consistency for Data Race Free (DRF)

Cores executing synchronized code blocks

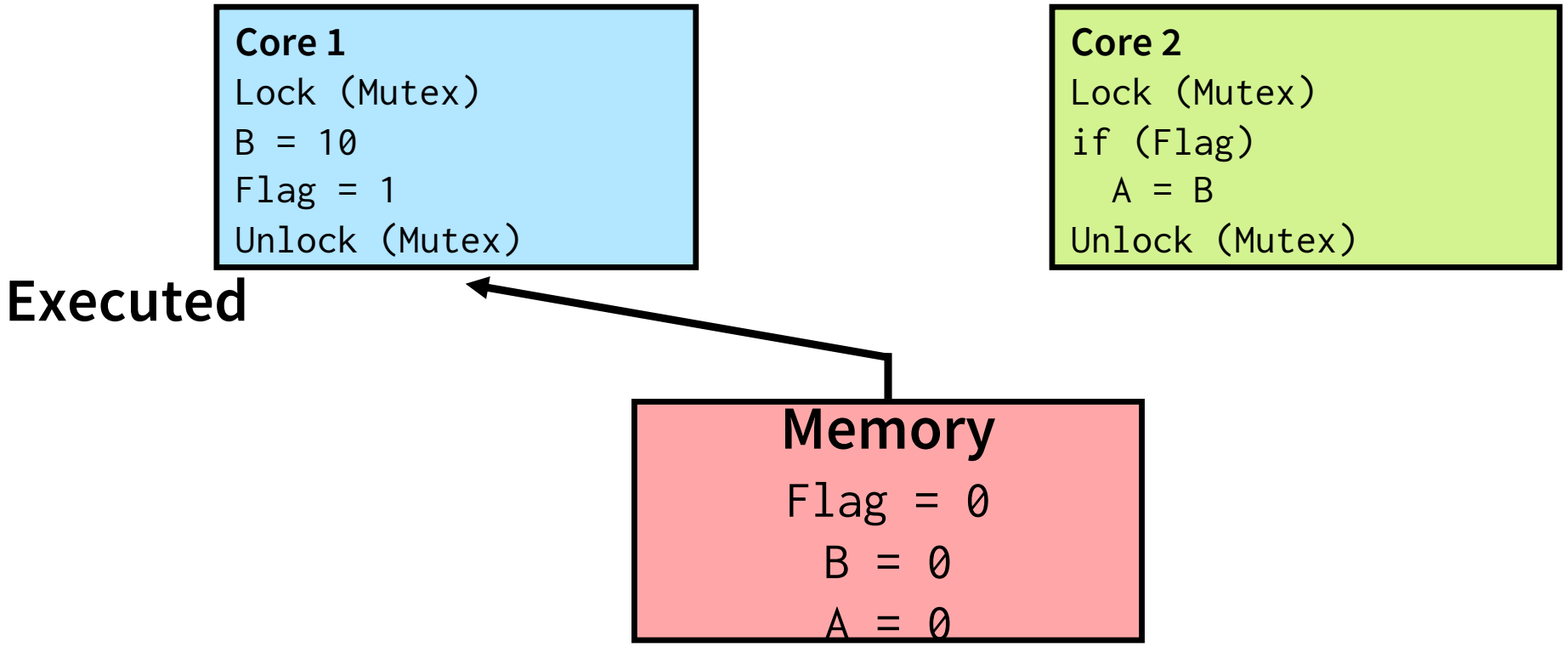
- Can be seen as a ”switch” running one instruction at a time



Sequential Consistency for Data Race Free (DRF)

Cores executing synchronized code blocks

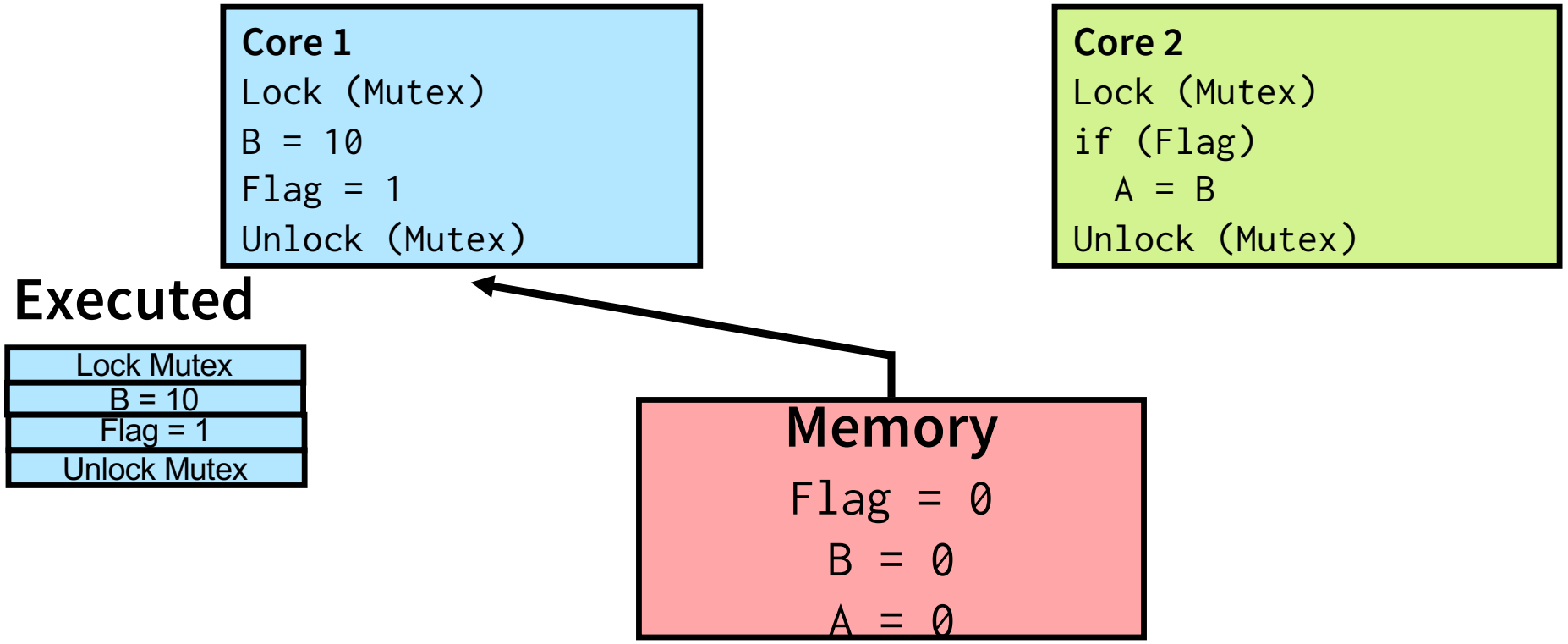
- Can be seen as a ”switch” running one instruction at a time



Sequential Consistency for Data Race Free (DRF)

Cores executing synchronized code blocks

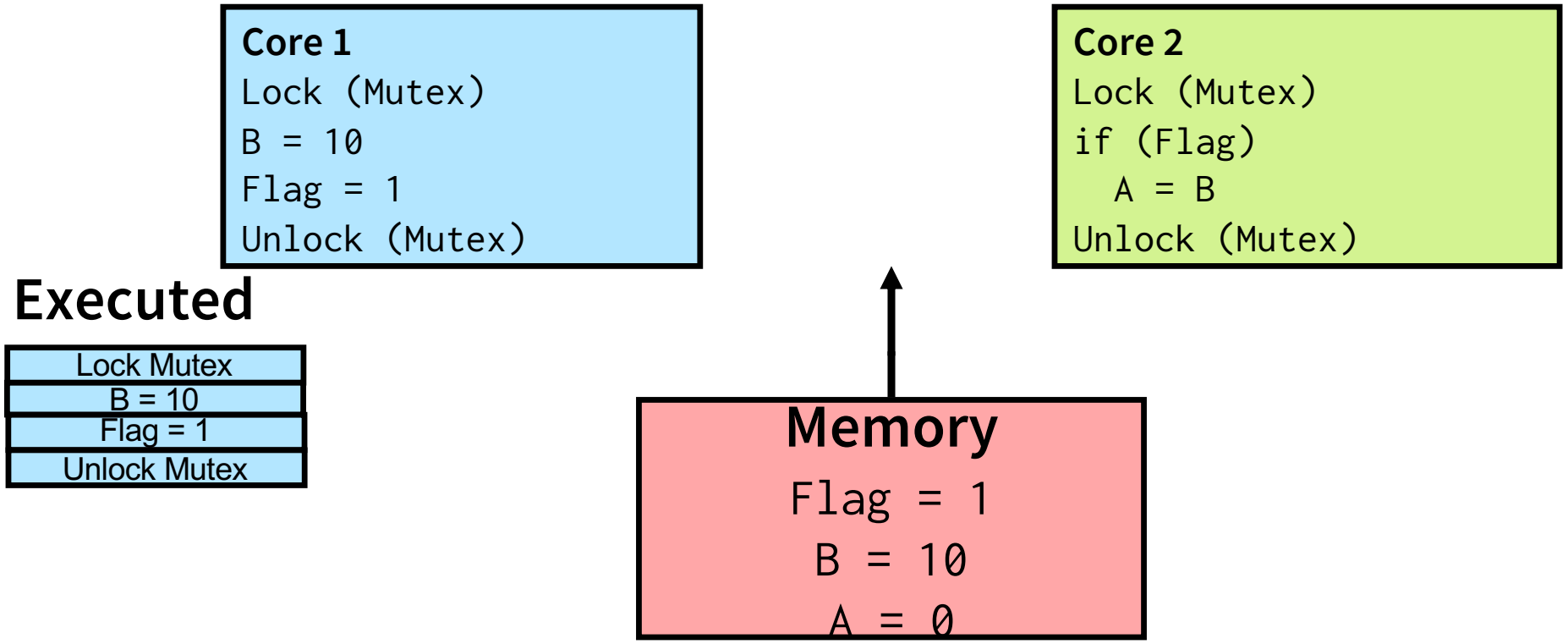
- Can be seen as a ”switch” running one instruction at a time



Sequential Consistency for Data Race Free (DRF)

Cores executing synchronized code blocks

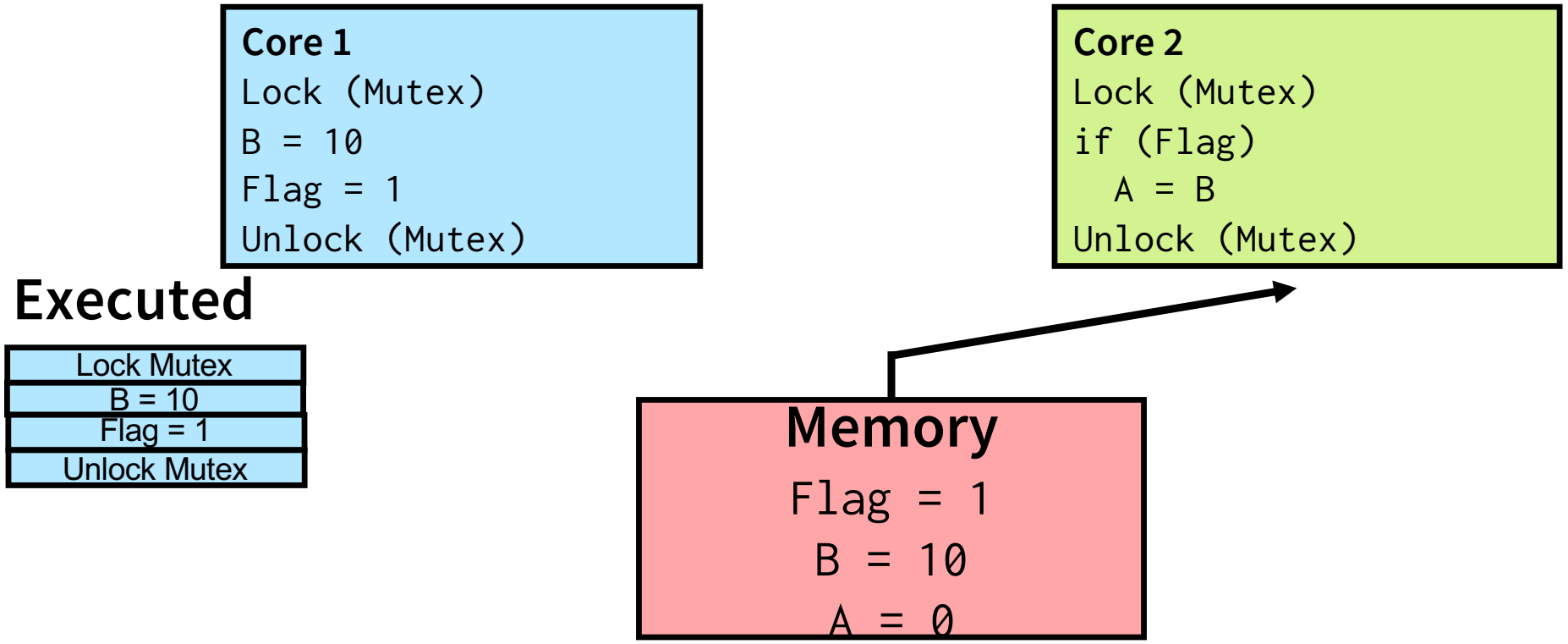
- Can be seen as a ”switch” running one instruction at a time



Sequential Consistency for Data Race Free (DRF)

Cores executing synchronized code blocks

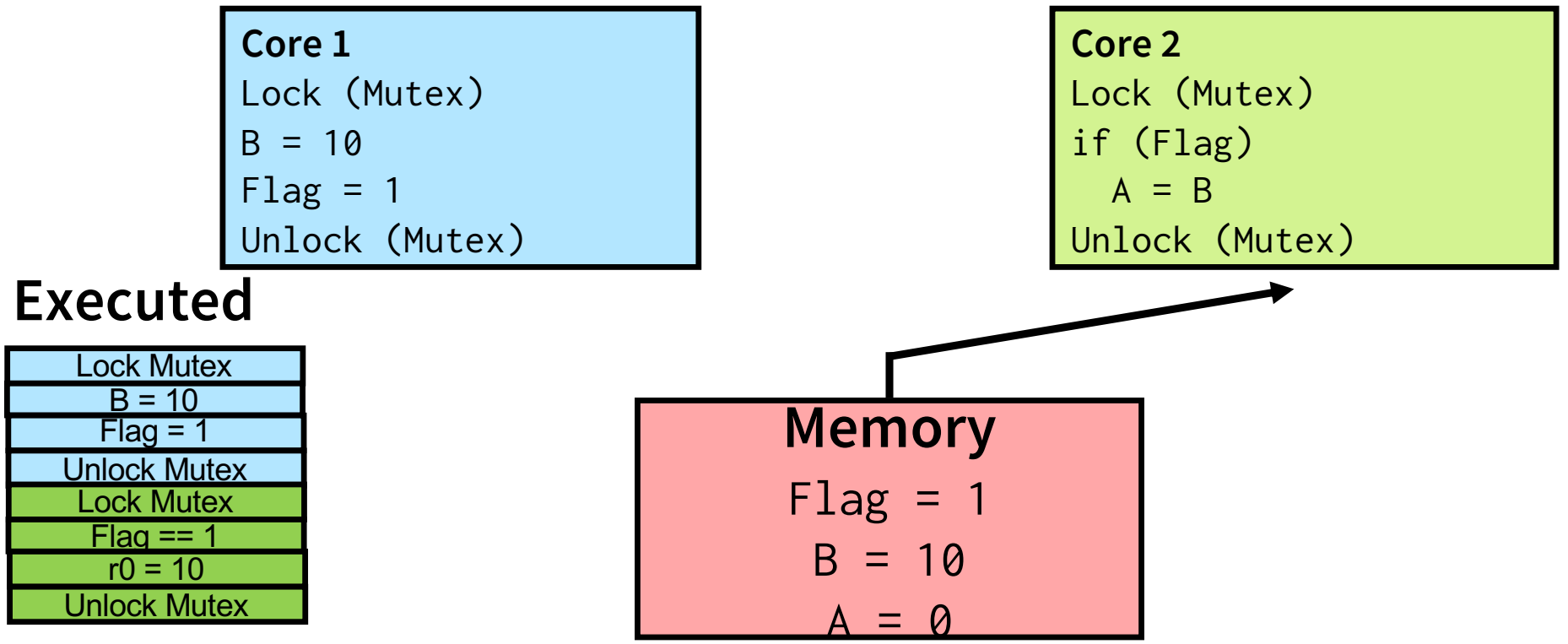
- Can be seen as a ”switch” running one instruction at a time



Sequential Consistency for Data Race Free (DRF)

Cores executing synchronized code blocks

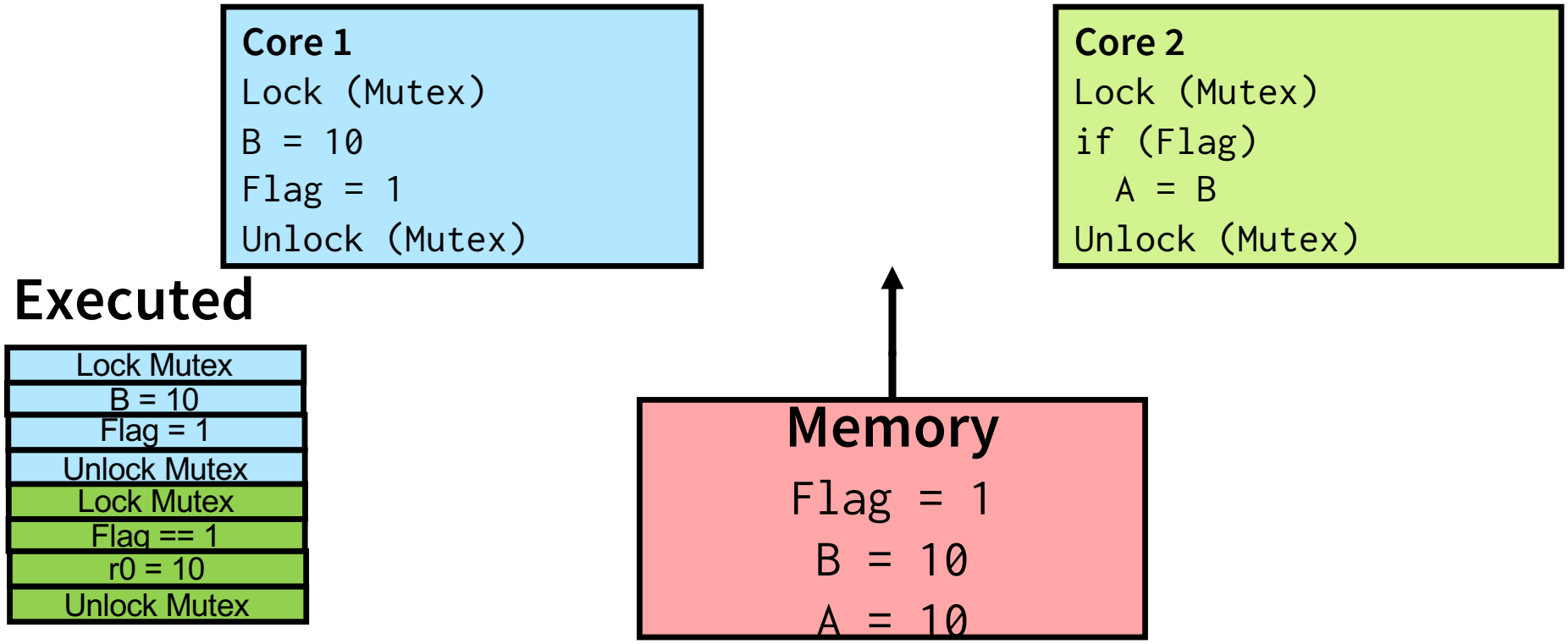
- Can be seen as a ”switch” running one instruction at a time



Sequential Consistency for Data Race Free (DRF)

Cores executing synchronized code blocks

- Can be seen as a ”switch” running one instruction at a time



Reference Materials

- https://www.youtube.com/watch?v=mDYLRX2pbFw
- Preshing
  - https://preshing.com/20120515/memory-reordering-caught-in-the-act/

Next Lecture (L #09)

- Hardware memory consistency model
- Extra lecture on Saturday (24/09) at 2.30pm to compensate for missing regular lecture of 30/09