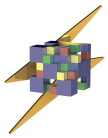
Lecture 12

Memory Consistency

이재진

서울대학교 컴퓨터공학부

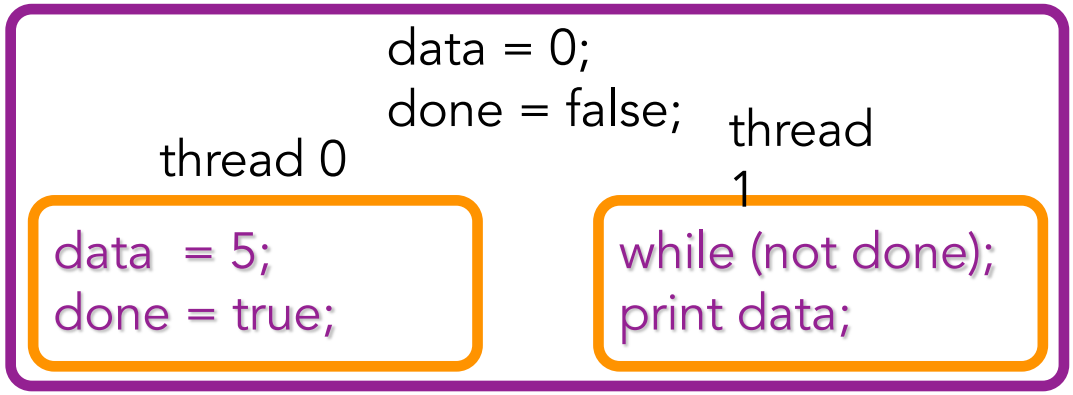
http://aces.snu.ac.kr



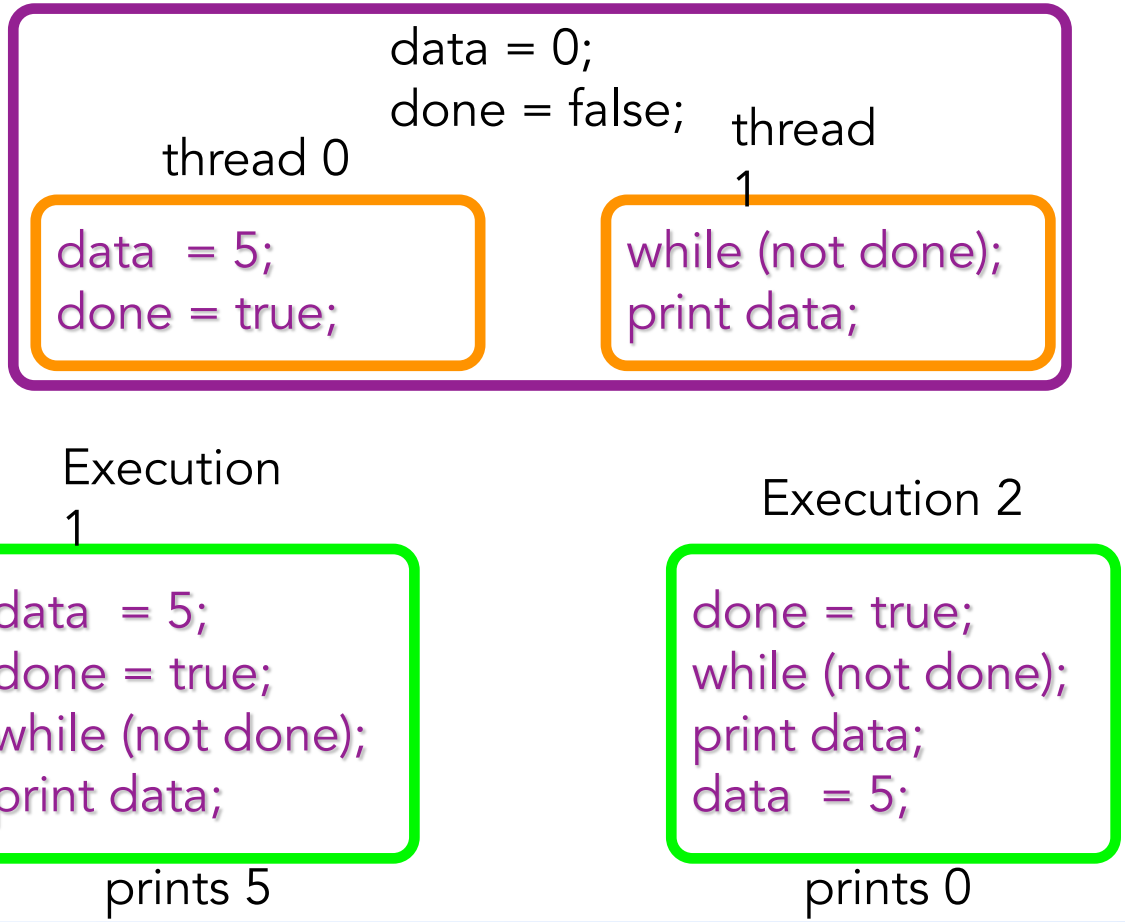
THUNDER Research Group
Seoul National University
서울대학교 천둥 연구실

A Motivating Example

- Expect memory to respect order between accesses to different locations in a thread



The Effect of Reordering



Coherence Helps?

- Different orders of data accesses to shared memory yield different execution outcomes

- Data accesses may be reordered by
  - Compiler optimizations
  - Underlying architectures

- Coherence does not help
  - Only to a single location (i.e., a single cache block)

Memory Consistency Models

- Specify constraints on the order in which memory operations (from any processor) can appear to execute with respect to one another
  - When is the updates of memory by a processor visible to another processor?

- To balance programming complexity and performance
  - Used by the programmer to reason about correctness and possible results of a program
  - Used by the system designer to constrain how much accesses can be reordered by a compiler or hardware

- Contract between the programmer and the multiprocessor system

Memory Consistency Models (cont'd)

- Sequential consistency

- Relaxed memory consistency models
  - Processor consistency
  - Weak ordering
  - Release consistency
  - …

Sequential Consistency

- The observable outcome of a multithreaded program P is the same as the outcome of a single thread executing all operations in P
  - A total order between operations is defined (atomic operations)
  - In this total order, two operations from the same thread of P are executed in the order specified in P

Sequential Consistency (contd.)

- [Lamport, IEEE TOC, 1979] A multiprocessor system is sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program

- Known to be inefficient with hardware implementation

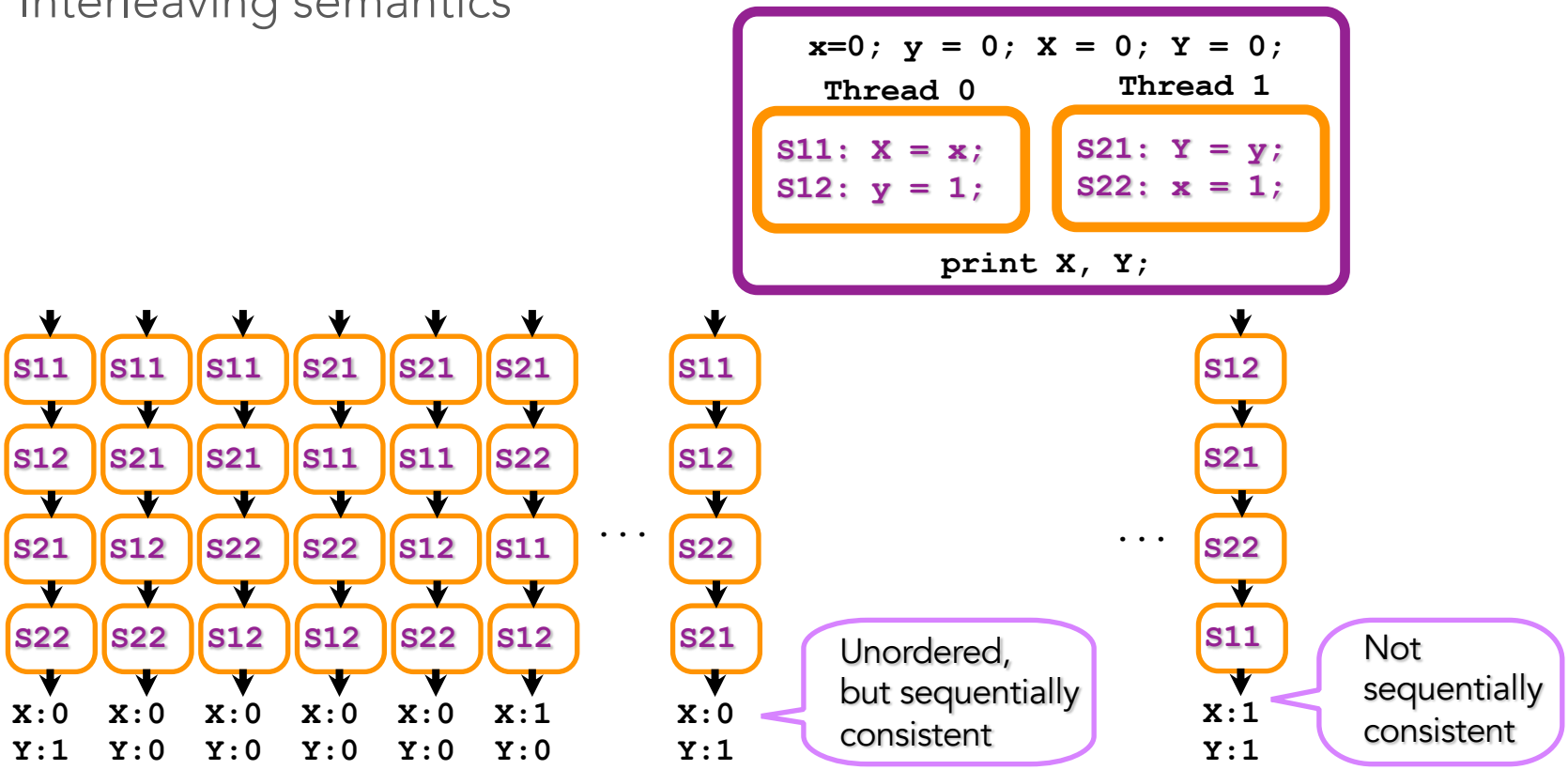
Program Order

- Order in which operations appear in source code
  - Straightforward translation of source code to assembly code
  - At most one memory operation per instruction

- But not the same as the order presented to hardware by the compiler

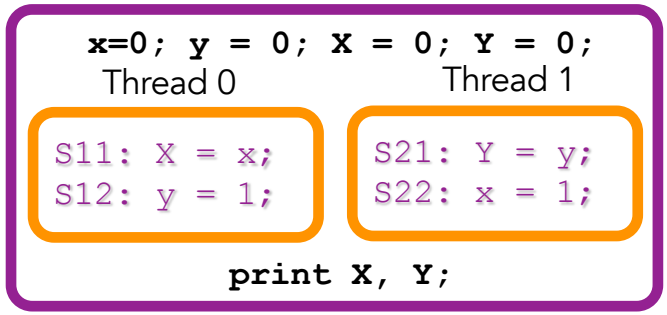
Possible Executions under SC

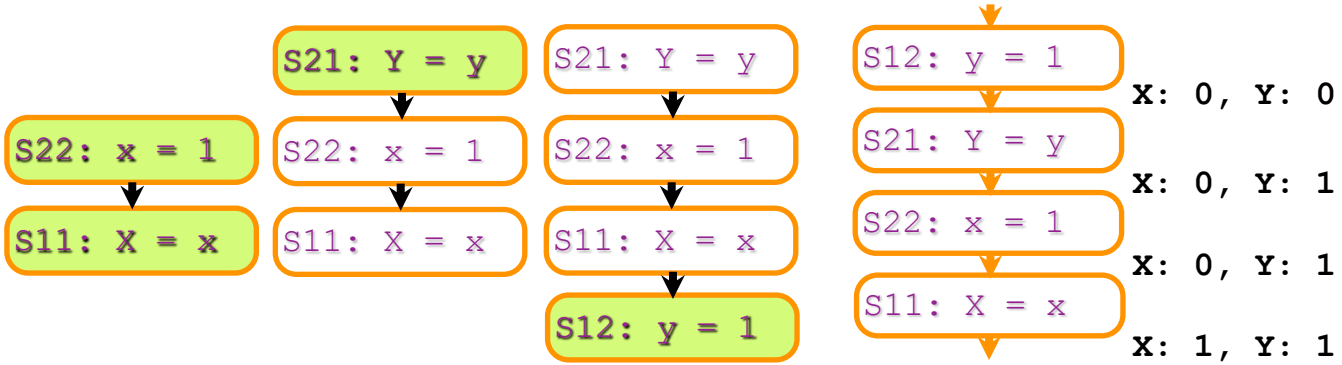
- Operations are atomic
- Interleaving semantics



Reasoning Based on SC

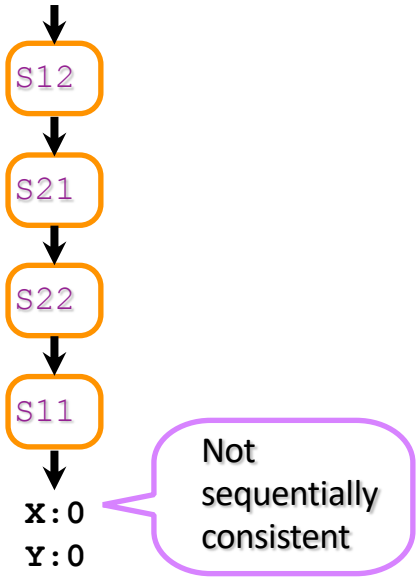
- If X is equal to 1 then Y should be 0
  - X is equal to 1 implies S22 is executed before S11
  - This implies S21 is executed before S12 due to the program order

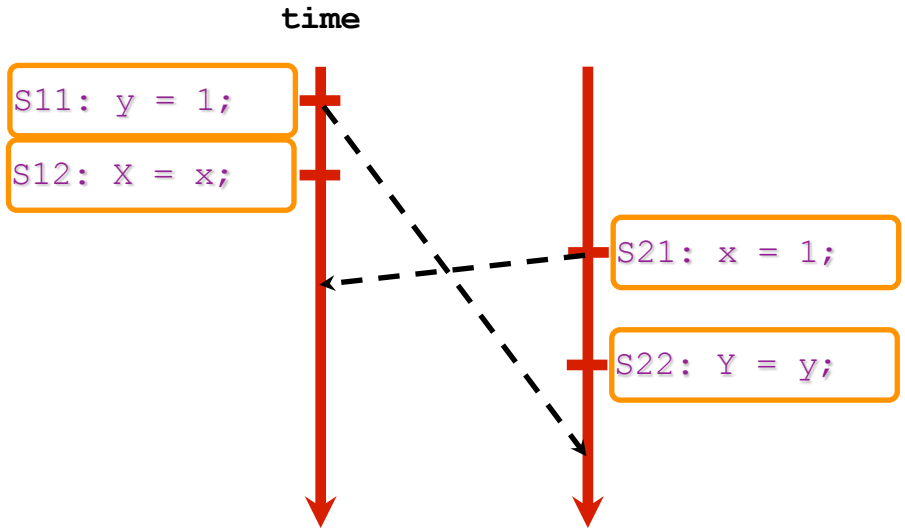




SC Violation

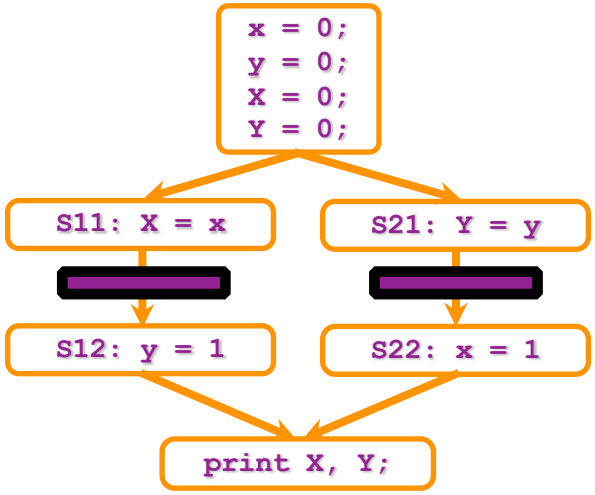
```
x=0; y = 0; X = 0; Y = 0;
Thread 0                Thread 1
S11: y = 1;             S21: x = 1;
S12: X = x;             S22: Y = y;
print X, Y;
```





Relaxed Memory Consistency Models

- Relaxations in program order
  - Preserve dependences
  - write → write, write → read, read → write, or read → read
- High performance, but hard to guarantee correctness for the programmer (difficult to program)
- Since S21 and S22 have no dependence, they can be reordered
- Memory fence (memory barrier) instructions prevent reordering of memory instructions



Performed with respect to ...

- A read by a processor P is performed with respect to another processor Q when a write by Q cannot affect the value returned by the read

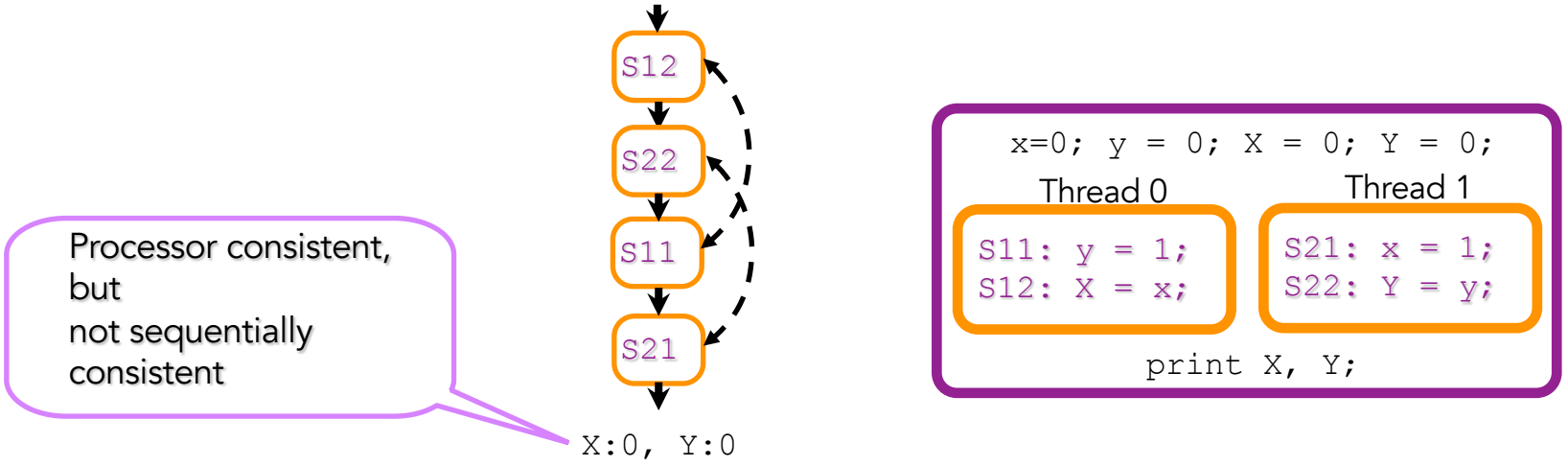
- A write by a processor P is performed with respect to another processor Q when a read by Q returns the value written by the write

Processor Consistency (PC)

- Definition
  - Before a read is allowed to perform with respect to any other processor, all previous read must be performed
  - Before a write is allowed to perform with respect to any other processor all previous accesses must be performed

Processor Consistency (cont'd)

- Relaxing write → read order
  - Allow a read to bypass (complete before) an earlier write in program order
  - Write buffer with read bypassing



Weak Ordering

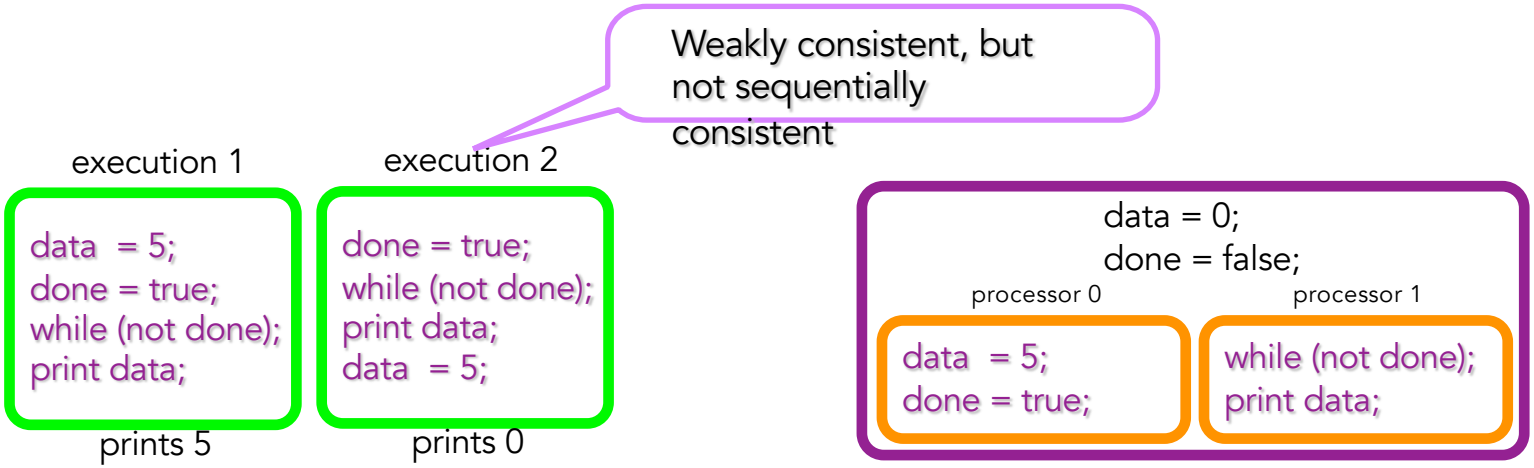
- Relax all program orders
- No program orders are guaranteed by the underlying hardware or compiler optimizations
- Except synchronization operations
  - Need to distinguish between ordinary reads/writes and synchronization operations

Weak Ordering (contd.)

- Definition
  - Before an ordinary read/write is allowed to perform w.r.t. any processor, all previous synchronization operations must be performed w.r.t. every processor
  - Before a synchronization operation is allowed to perform w.r.t. any processor, all previous ordinary reads/writes must be performed w.r.t. every processor
  - Synchronization operations obey sequential consistency

Weak Ordering (contd.)

- Multiple read/write requests can be outstanding at the same time
  - Hide read and write latency
- The ordering with respect to a synchronization operation (sync) must be guaranteed
  - read/write → sync, sync → read/write



Release Consistency

- Relax all program orders, but not w.r.t. synchronization operations

- Two separate synchronization operations
  - Acquire: a read operation such as lock(V)
  - Release: a write operation such as unlock(V)

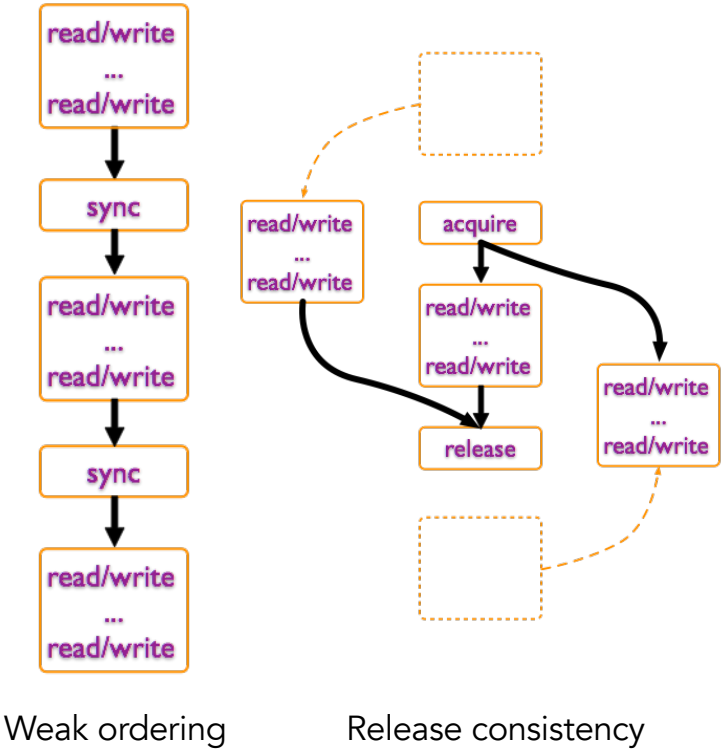
- Need to distinguish between ordinary reads/writes and synchronization operations
  - Additionally distinguish between acquire and release operations

Release Consistency (cont'd)

- Definition
  - Before an ordinary read/write is allowed to perform w.r.t. any processor, all previous acquires must be performed w.r.t. every processor
  - Before a release is allowed to perform w.r.t. any processor, all previous ordinary reads/writes must be performed w.r.t. every processor
  - Acquire and release operations are sequentially consistent (or processor consistent)

Release Consistency (cont'd)

- Specific memory access ordering with respect to acquire/release must be guaranteed
  - read/write → release
  - acquire → read/write
- Memory accesses in the critical section do not wait or delay reads/writes outside the critical section
  - Reads/writes following a release (unlock) do not have to be delayed for the release to complete
  - An acquire (lock) needs not to be delayed for previous reads/writes to complete



Weak ordering Release consistency

Understanding RC