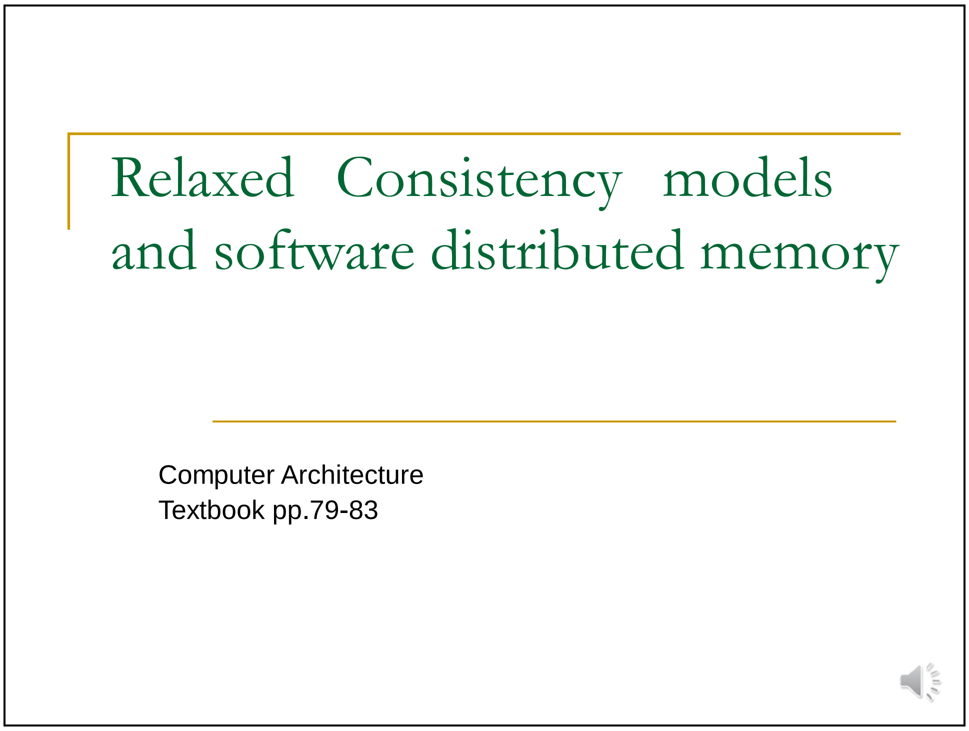
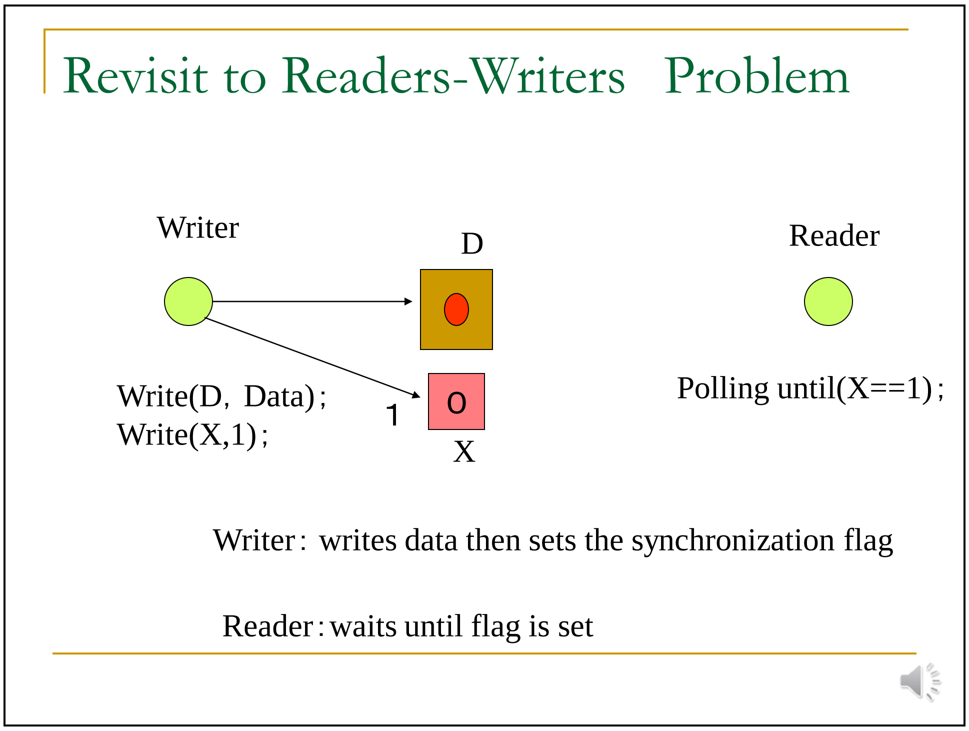
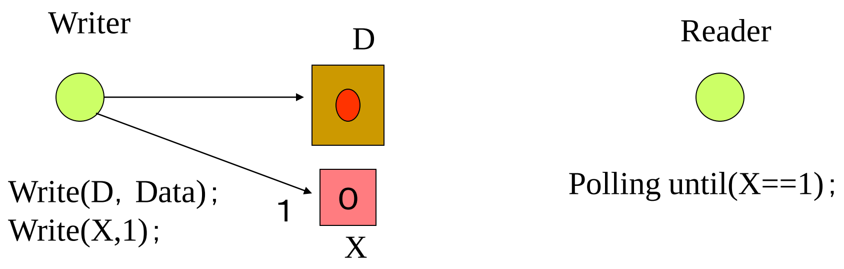


Relaxed Consistency models and software distributed memory

Computer Architecture
Textbook pp.79-83



Revisit to Readers-Writers Problem

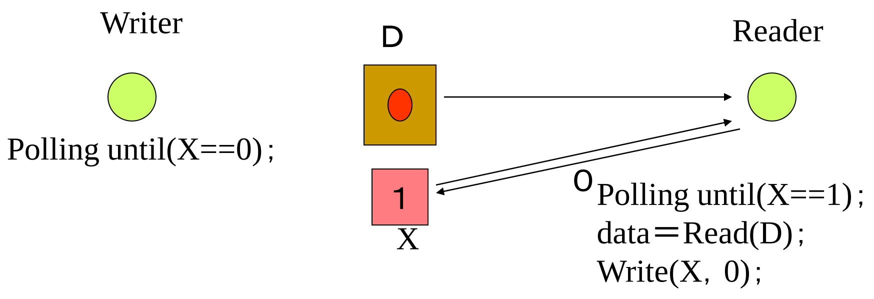


Writer: writes data then sets the synchronization flag

Reader: waits until flag is set

Let's revisit to the readers-writers problem. For sending data from the writer to the reader, I said first the writer writes the data, then write 1 to the synchronization variable.

Readers-Writers Problem



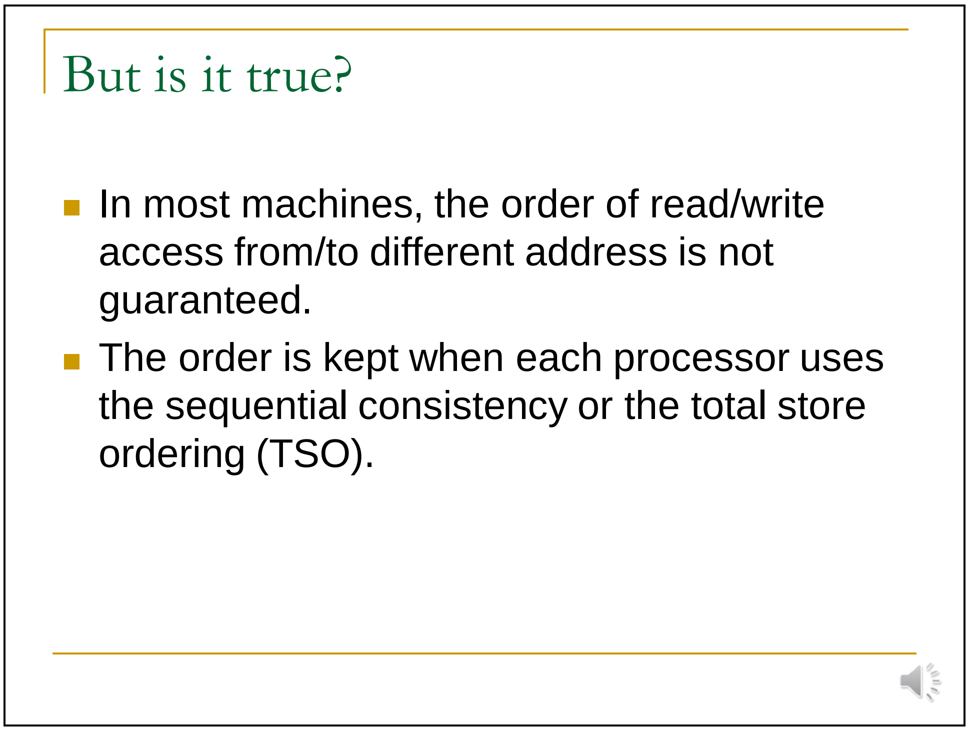
```
Polling until(X==0);
```

```
Polling until(X==1);
data=Read(D);
Write(X, 0);
```

Reader: reads data from D when flag is set, then resets the flag

Writer: waits for the reset of the flag

The reader reads the synchronization variable, and if it is 1, it reads the data from D. It seems to be correct. But is it true?

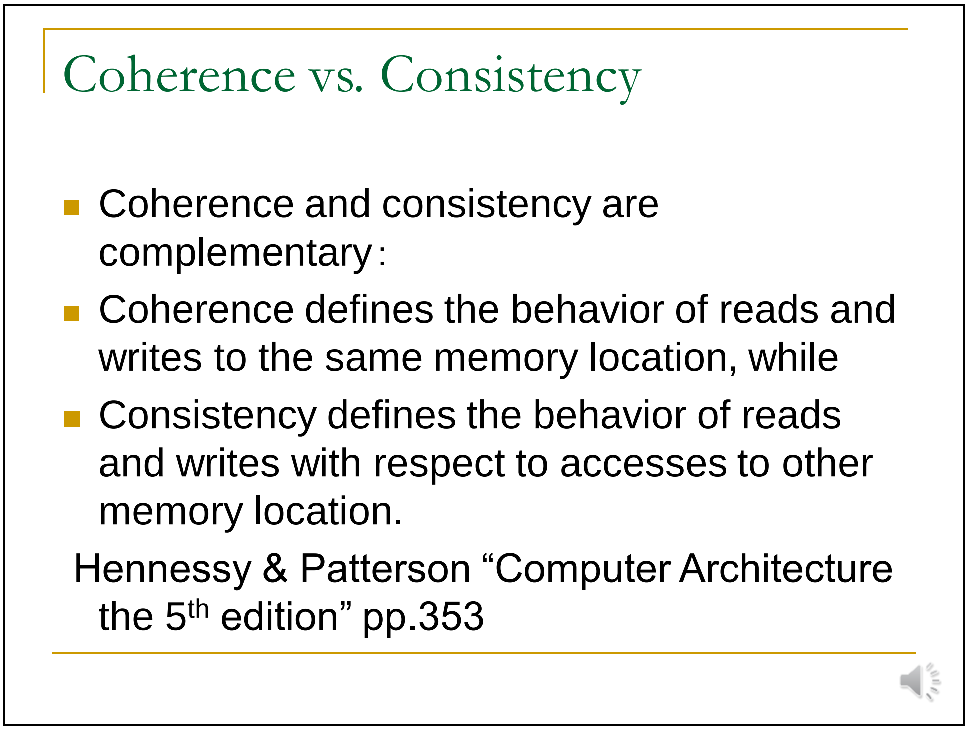


But is it true?

- In most machines, the order of read/write access from/to different address is not guaranteed.
- The order is kept when each processor uses the sequential consistency or the total store ordering (TSO).

If the order of read/write access from/to different address is guaranteed, it is true. But, in most recent machines do not keep it. Exactly speaking if the sequential consistency or the total store ordering is guaranteed it the machine, it can be done.

Today, I would like to talk about the problem.

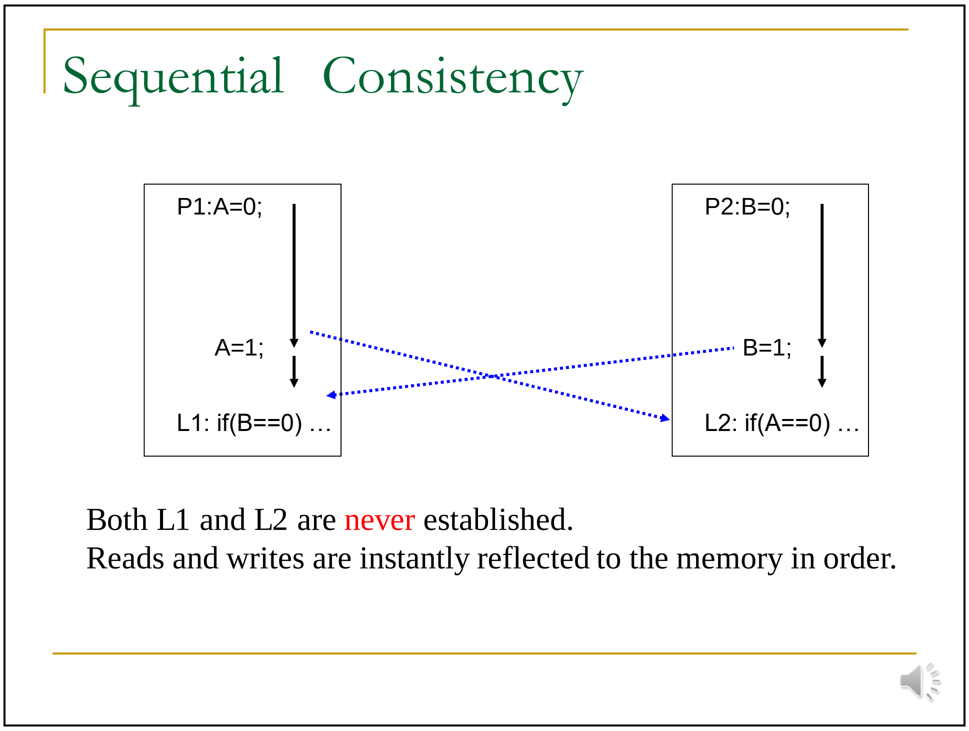


Coherence vs. Consistency

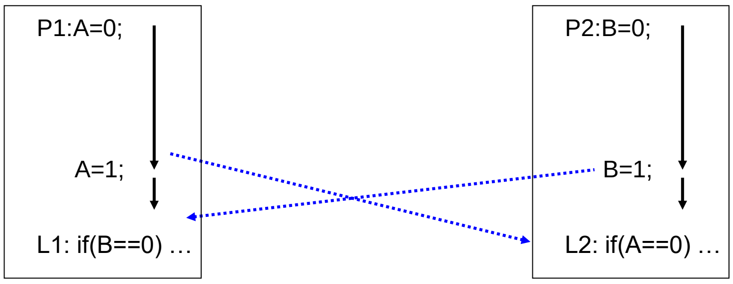
- Coherence and consistency are complementary:
- Coherence defines the behavior of reads and writes to the same memory location, while
- Consistency defines the behavior of reads and writes with respect to accesses to other memory location.

Hennessy & Patterson "Computer Architecture the 5th edition" pp.353

The words coherence and consistency are complement. That is coherence defines the behavior or reads and writes to the same memory location, while consistency is for other memory location. Today, I am going to treat only consistency, that is the case of two accesses are done to/from the different addresses.

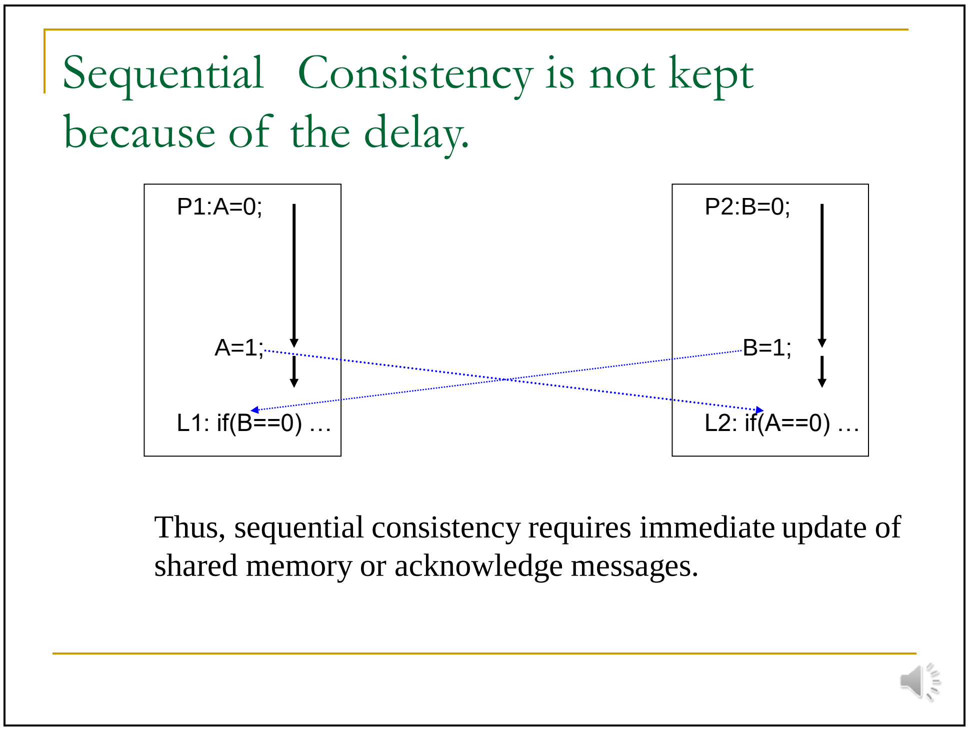


Sequential Consistency

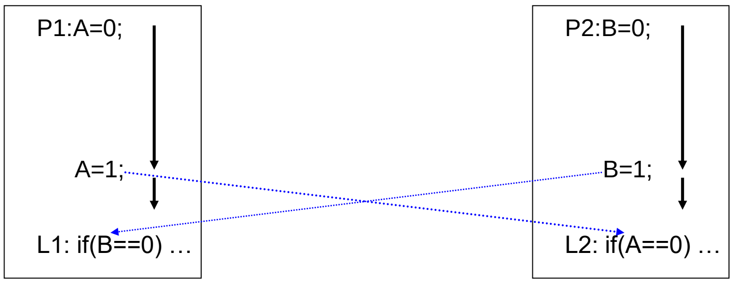


Both L1 and L2 are never established.
Reads and writes are instantly reflected to the memory in order.

First of all, let me explain about the sequential consistency. In this diagram, if both L1 and L2 are never established, the sequential consistency is guaranteed. I means that reads and writes are instantly reflected to the memory in order.

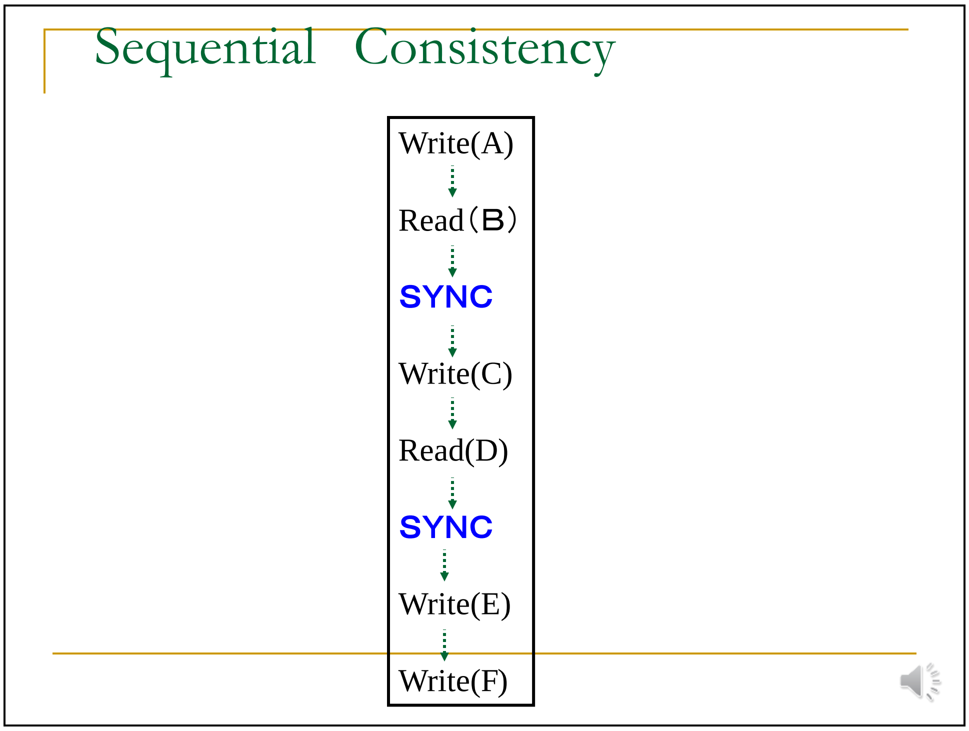


Sequential Consistency is not kept because of the delay.



Thus, sequential consistency requires immediate update of shared memory or acknowledge messages.

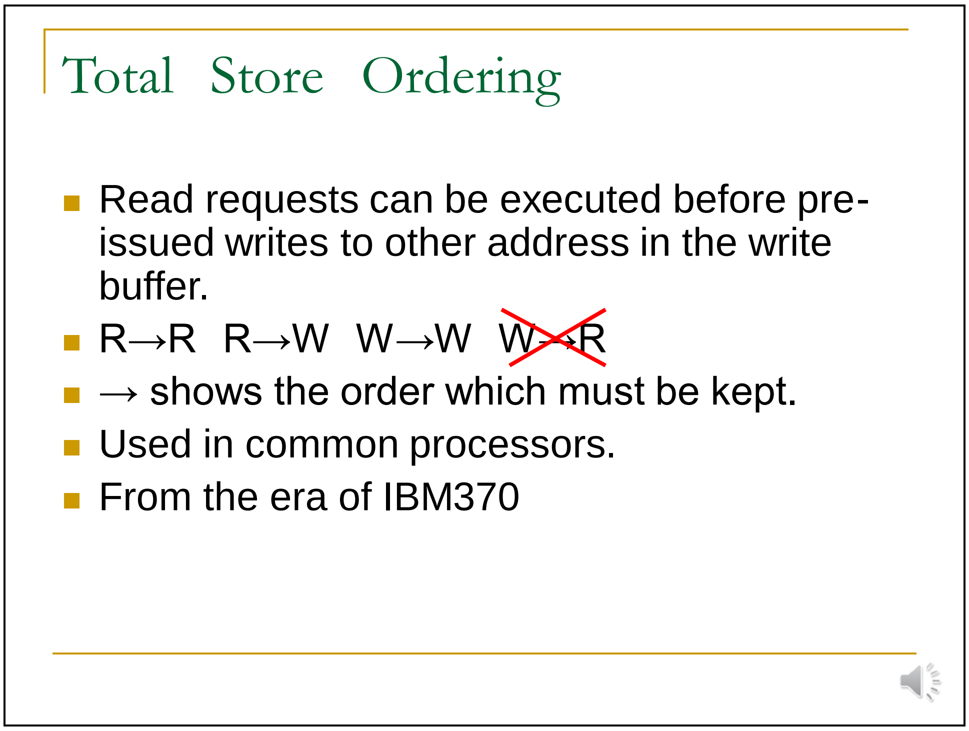
If it takes any delay to the memory or any change of the reference order, these two sentences can work at the same time. OK. let’s examine the problem for the access order in a single processor first.



Sequential Consistency

$$\begin{tikzcd}
\text{Write(A)} \arrow[d, dashed] \\
\text{Read(B)} \arrow[d, dashed] \\
\textbf{SYNC} \arrow[d, dashed] \\
\text{Write(C)} \arrow[d, dashed] \\
\text{Read(D)} \arrow[d, dashed] \\
\textbf{SYNC} \arrow[d, dashed] \\
\text{Write(E)} \arrow[d, dashed] \\
\text{Write(F)}
\end{tikzcd}$$

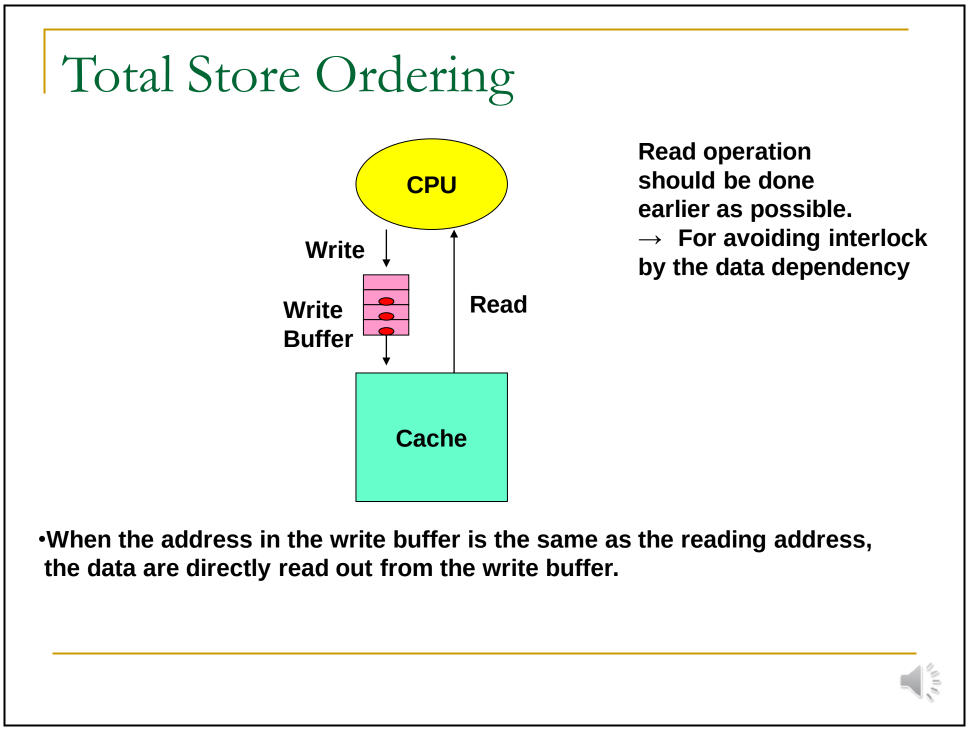
In the sequential consistency model, every access must be done in the order described in the program.



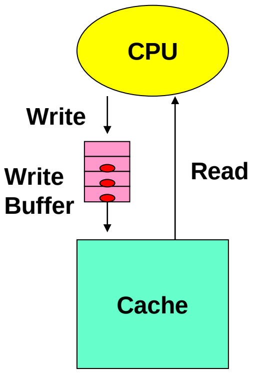
Total Store Ordering

- Read requests can be executed before pre-issued writes to other address in the write buffer.
- R→R R→W W→W ~~W→R~~
- → shows the order which must be kept.
- Used in common processors.
- From the era of IBM370

In order to enhance the performance of a single processor, read requests are often executed before pre-issued writes to other address in the write buffer. Here, this arrow mark shows the order which must be kept. This is used in common processors fromt the era of IBM370.



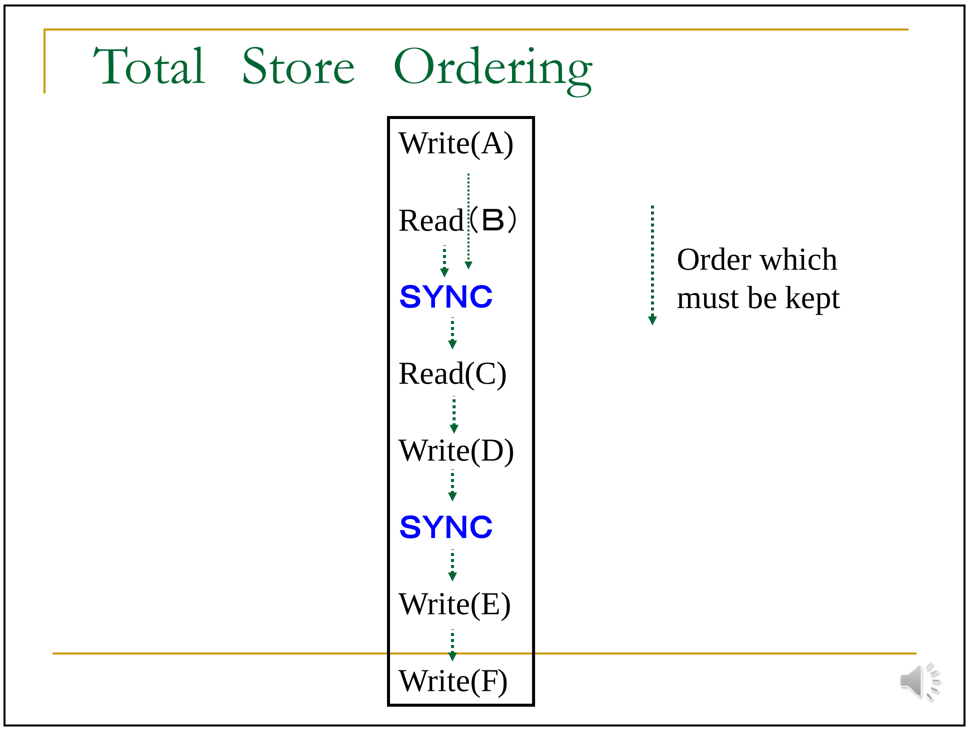
Total Store Ordering



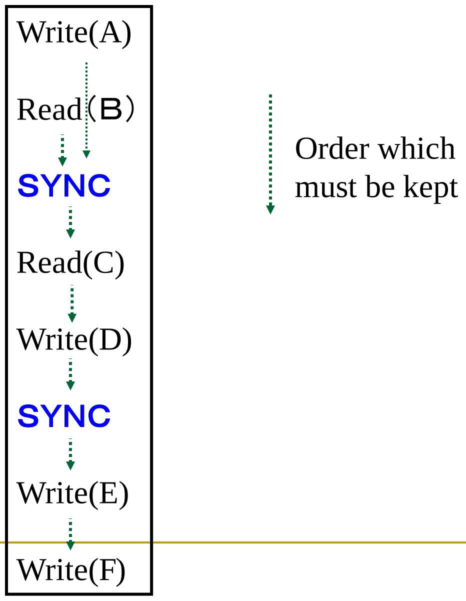
Read operation should be done earlier as possible.
→ For avoiding interlock by the data dependency

•When the address in the write buffer is the same as the reading address, the data are directly read out from the write buffer.

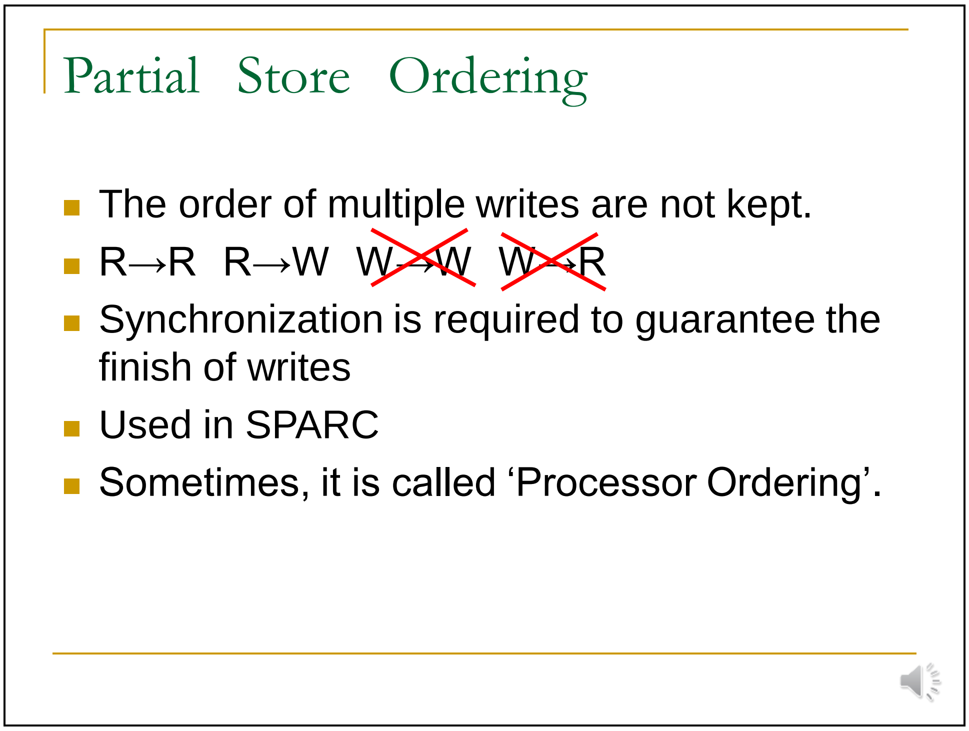
In order to avoid the interlock caused by the data dependency, the read operation must be done early as possible. So, it can be done before write requests in the write buffer. Of course, if the address in the write buffer is the same as the reading address, the data are directly read out from the write buffer. Note that we are today considering the consistency problem, not the coherent problem.



Total Store Ordering



The diagram shows the order which must be kept. The order between this write and read is not have to be kept in the total store ordering.

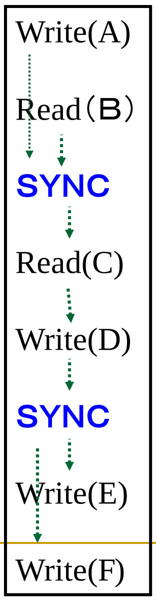


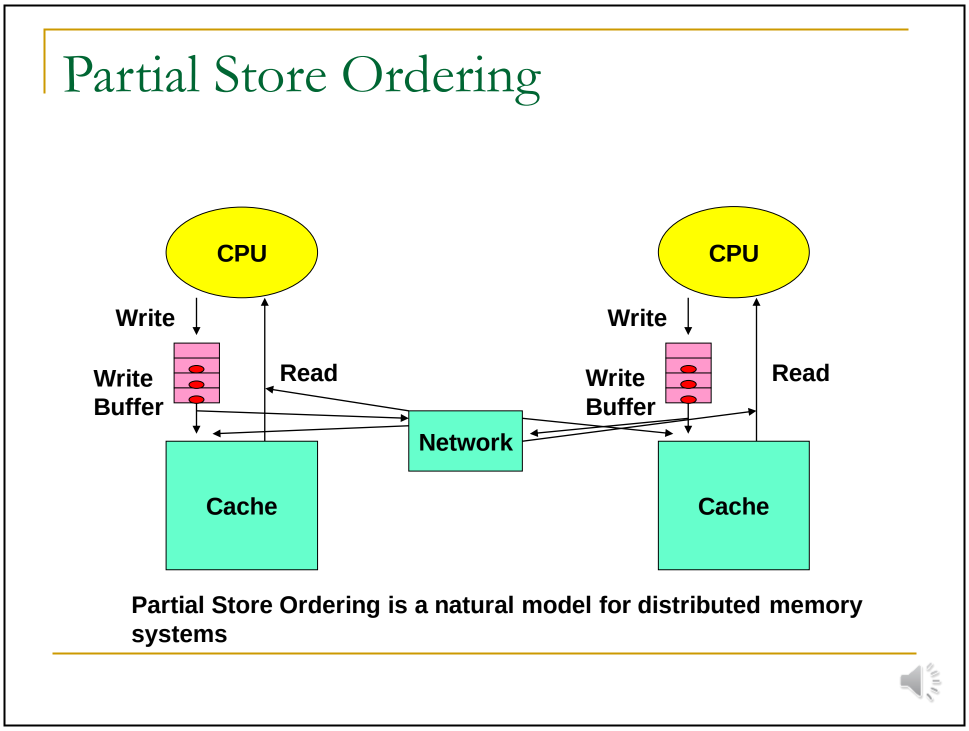
Partial Store Ordering

- The order of multiple writes are not kept.
- R→R R→W ~~W→W~~ ~~W→R~~
- Synchronization is required to guarantee the finish of writes
- Used in SPARC
- Sometimes, it is called ‘Processor Ordering’.

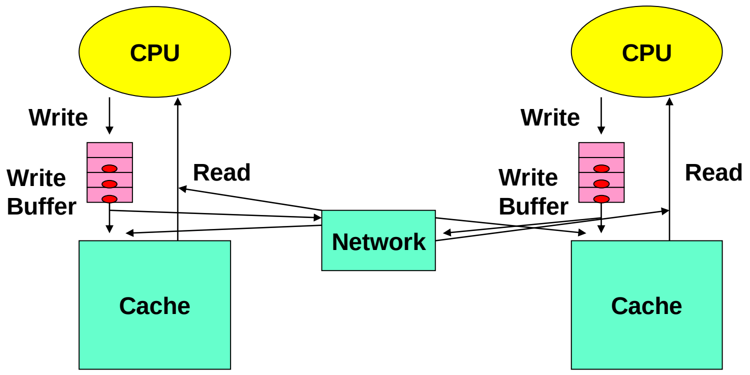
When there are multiple memory modules which have different access time, it is difficult to keep the order of multiple writes. So, the partial store ordering relaxes the order between two write requests. In this model, finish of writes must be guaranteed with the synchronization operations. This ordering model was used for SPARC microprocessors, and sometimes called the processor ordering.

Partial Store Ordering



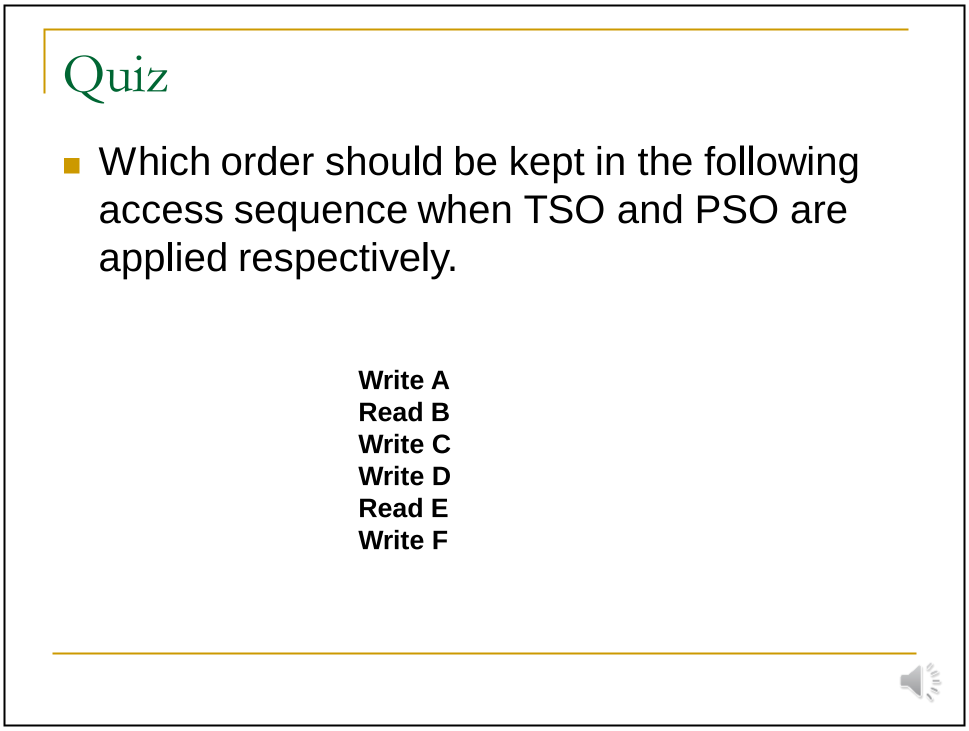


Partial Store Ordering



Partial Store Ordering is a natural model for distributed memory systems

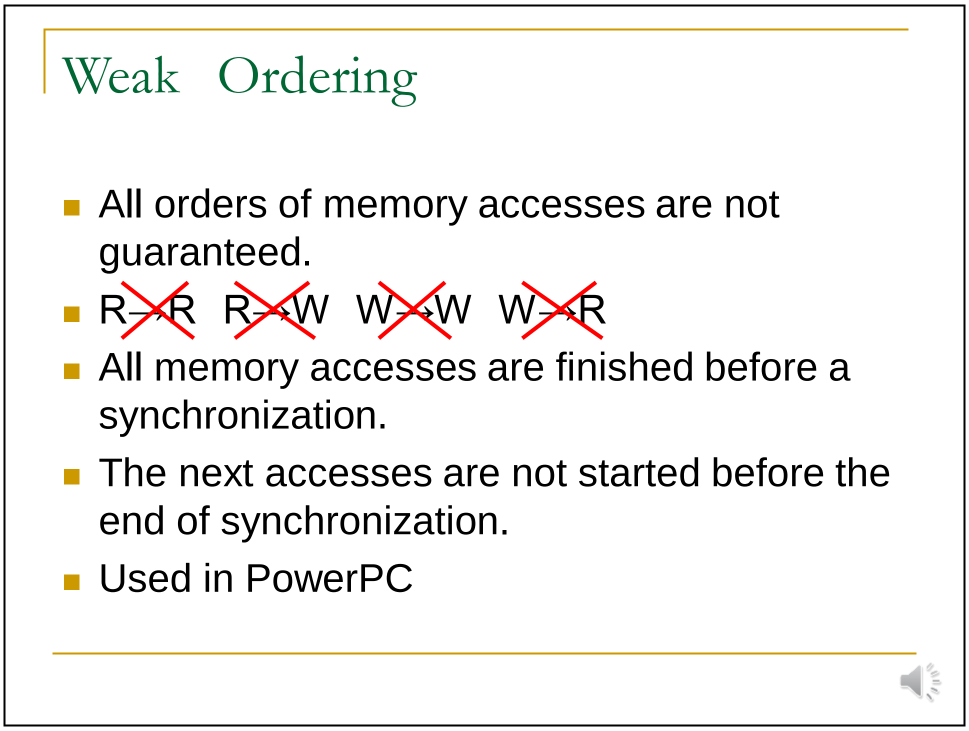
As this diagram shows, two write operations are issued to the different memory modules. That is, it is natural for distributed memory systems.


Quiz

- Which order should be kept in the following access sequence when TSO and PSO are applied respectively.

Write A
Read B
Write C
Write D
Read E
Write F

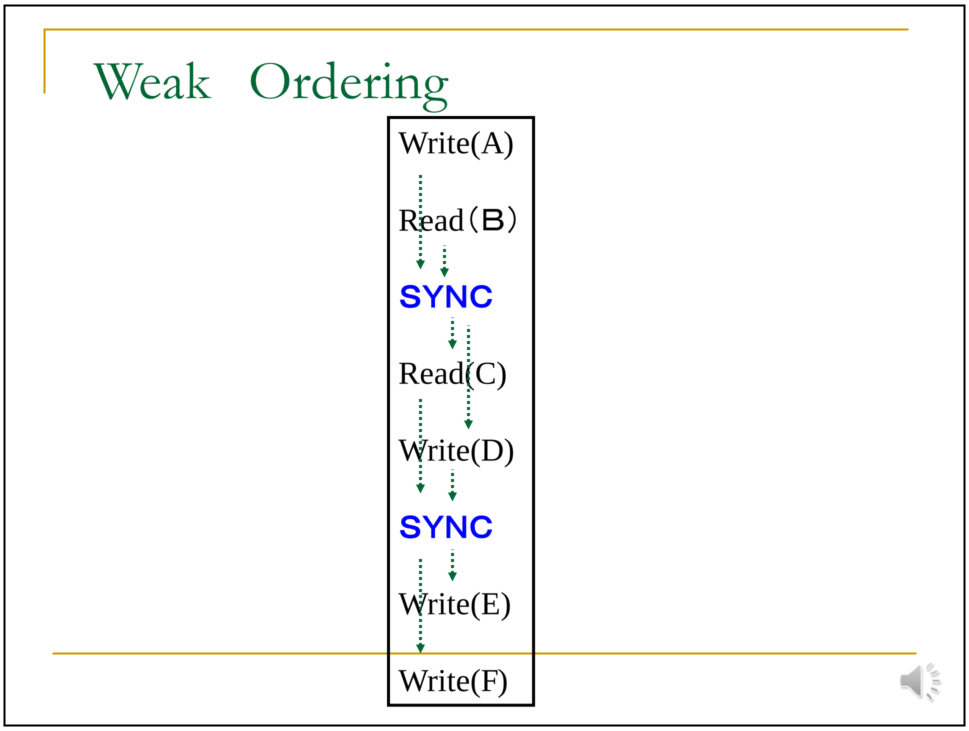
OK. then here is a quiz.



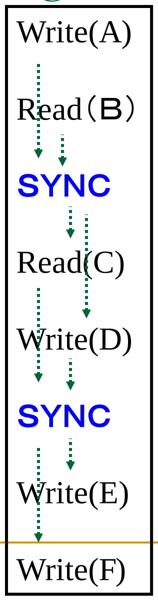
Weak Ordering

- All orders of memory accesses are not guaranteed.
- $\cancel{R \rightarrow R}$ $\cancel{R \rightarrow W}$ $\cancel{W \rightarrow W}$ $\cancel{W \rightarrow R}$
- All memory accesses are finished before a synchronization.
- The next accesses are not started before the end of synchronization.
- Used in PowerPC

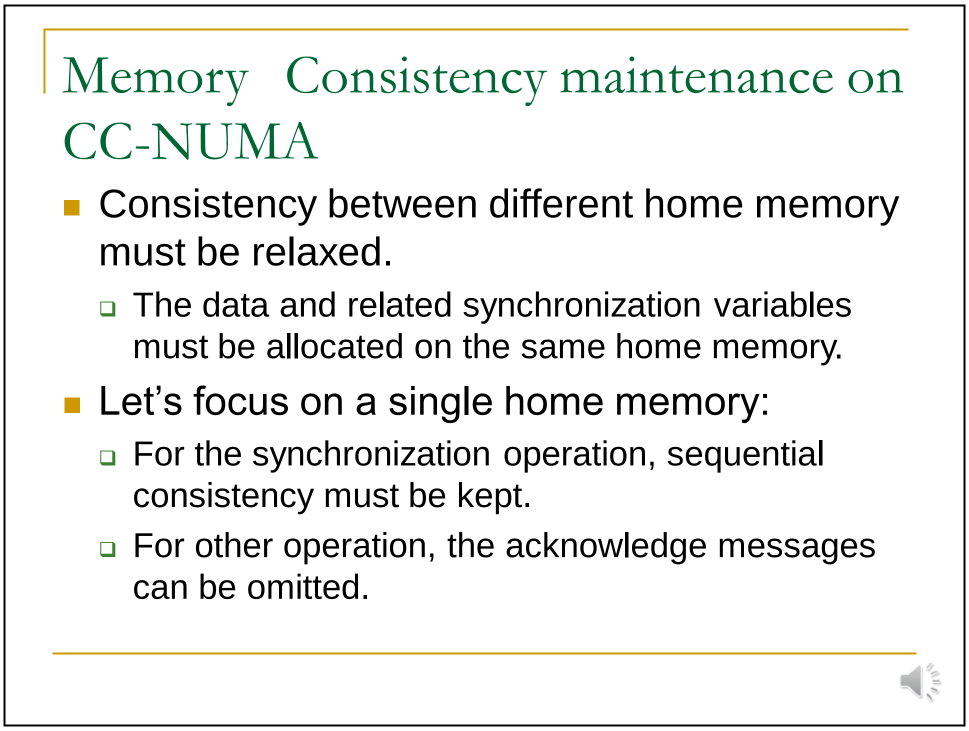
If there are several different memory modules, even for a single processor, it is difficult to keep the order of memory request. So, it is natural to relax all orders of memory accesses. We only need to keep the order between synchronization operation. That is, all memory accesses are finished before a synchronization operation, and the next access must not be started before the end of synchronization. This ordering model was adopted in IBM PowerPC.



Weak Ordering



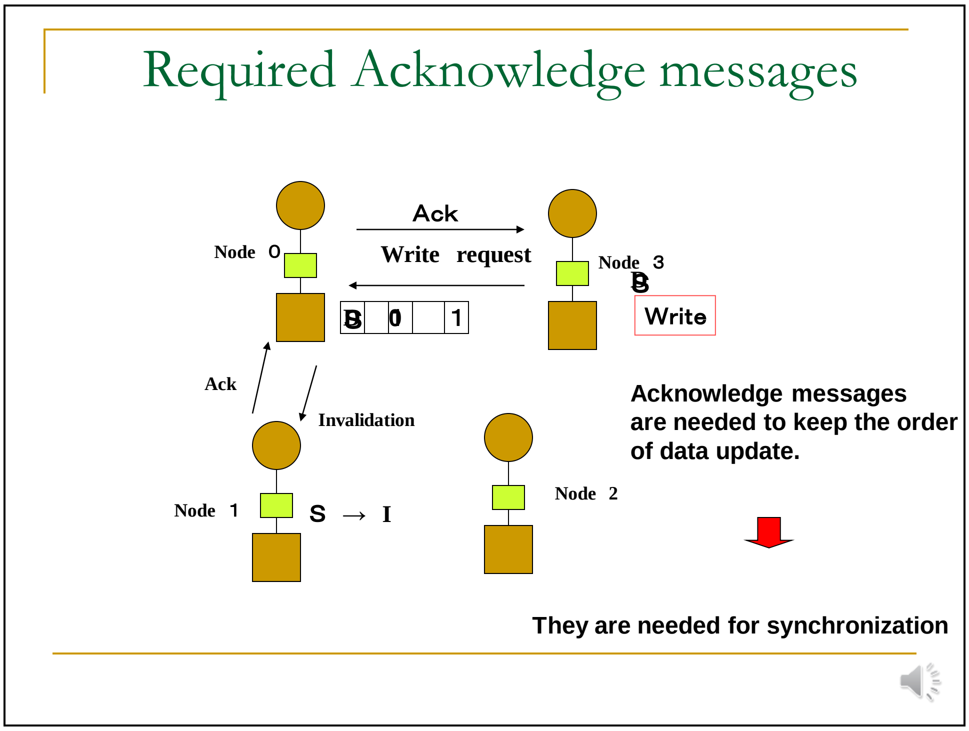
This is the diagram of the weak ordering.



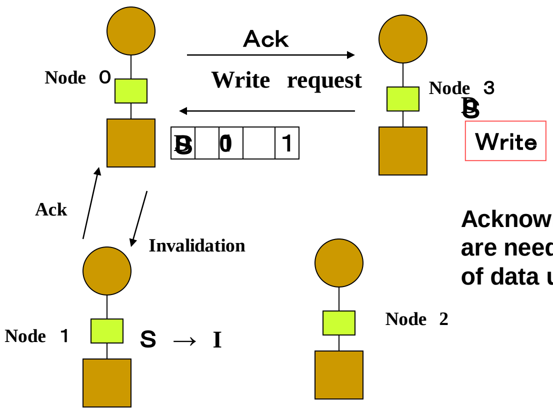
Memory Consistency maintenance on CC-NUMA

- Consistency between different home memory must be relaxed.
  - The data and related synchronization variables must be allocated on the same home memory.
- Let's focus on a single home memory:
  - For the synchronization operation, sequential consistency must be kept.
  - For other operation, the acknowledge messages can be omitted.

For keeping the memory consistency for CC-NUMA, first of all, consistency between different home memory must be relaxed. The data and related synchronization variables must be allocated on the same home memory.



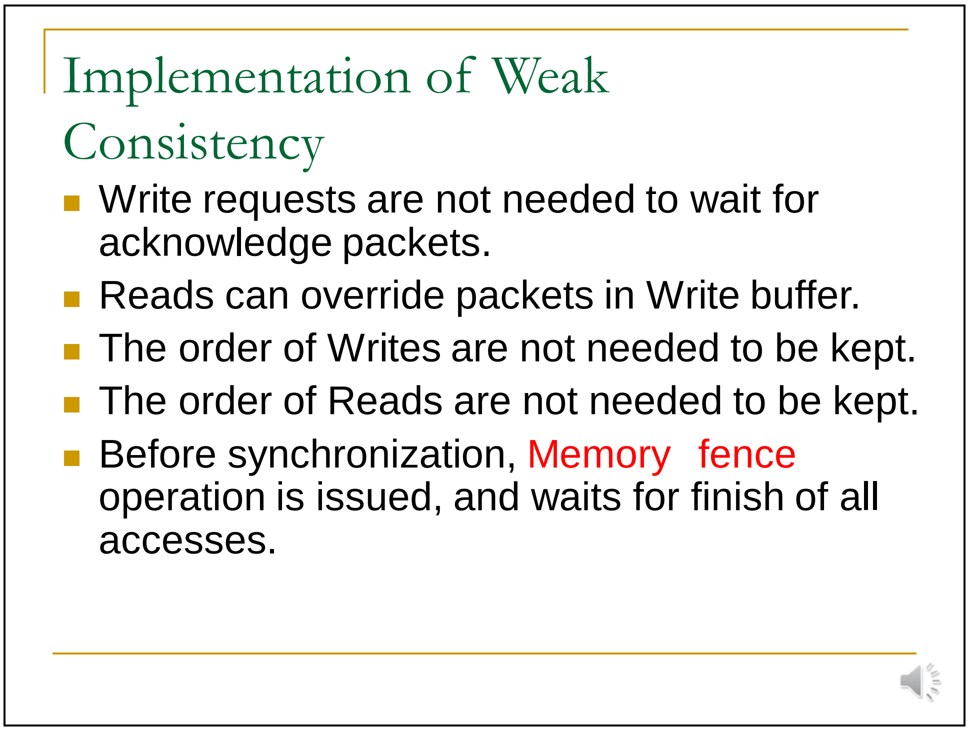
Required Acknowledge messages



Acknowledge messages are needed to keep the order of data update.

They are needed for synchronization

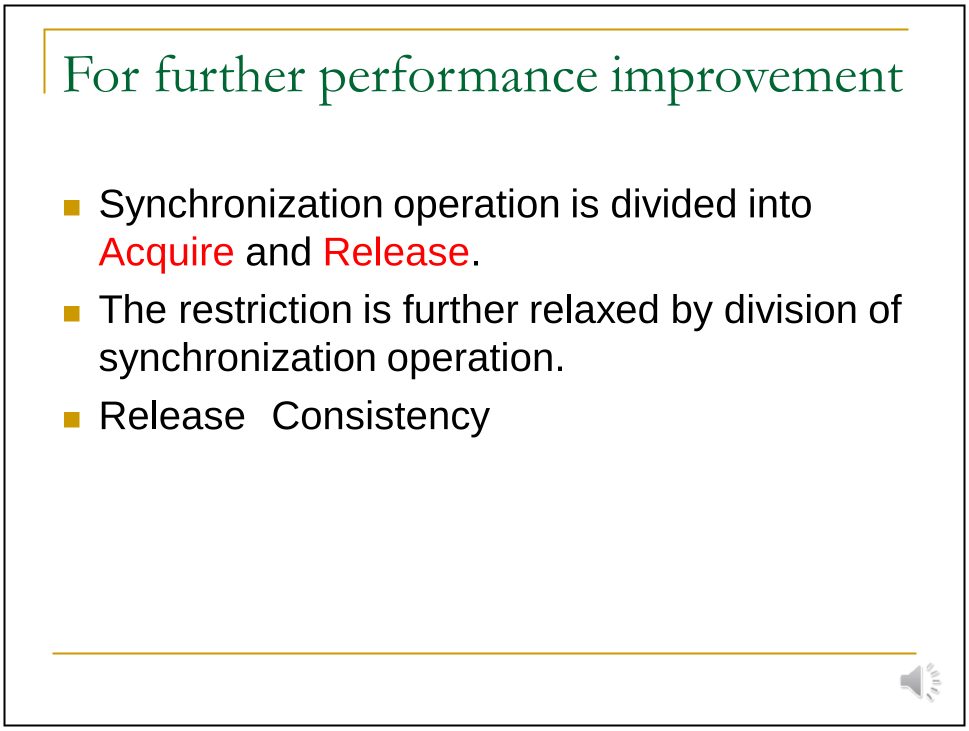
As I introduced in the previous lesson, a lot of acknowledge messages are needed for invalidation, but since the writing data uses a relaxed consistency model, they are not needed. The home can reply just after sending invalidation messages.



Implementation of Weak Consistency

- Write requests are not needed to wait for acknowledge packets.
- Reads can override packets in Write buffer.
- The order of Writes are not needed to be kept.
- The order of Reads are not needed to be kept.
- Before synchronization, Memory fence operation is issued, and waits for finish of all accesses.

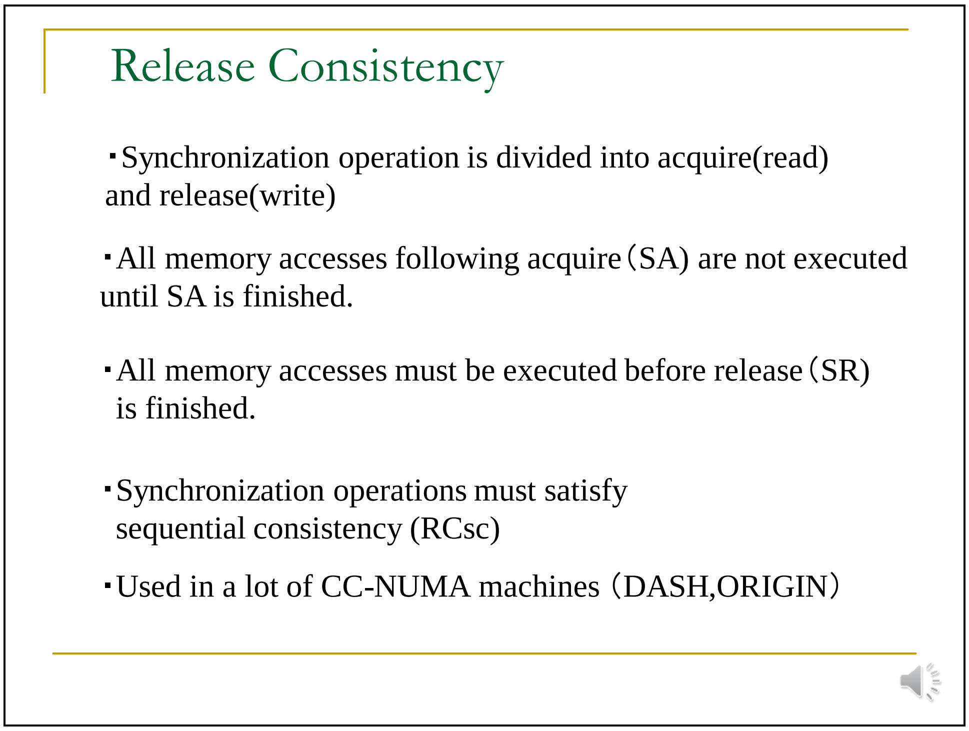
We need to provide the memory fence operation for implementing the synchronization. Once this operation is issued, no access for the memory is accepted. They must wait for the finish of the synchronization operation.



For further performance improvement

- Synchronization operation is divided into Acquire and Release.
- The restriction is further relaxed by division of synchronization operation.
- Release Consistency

For further performance improvement, the Stanford university proposed a further relaxed model for CC-NUMA machine. In this model, a synchronization operation is divided into Acquire and Release. Considering two types of synchronization operations, the restriction is further relaxed. This model is called a release consistency.



Release Consistency

・Synchronization operation is divided into acquire(read) and release(write)

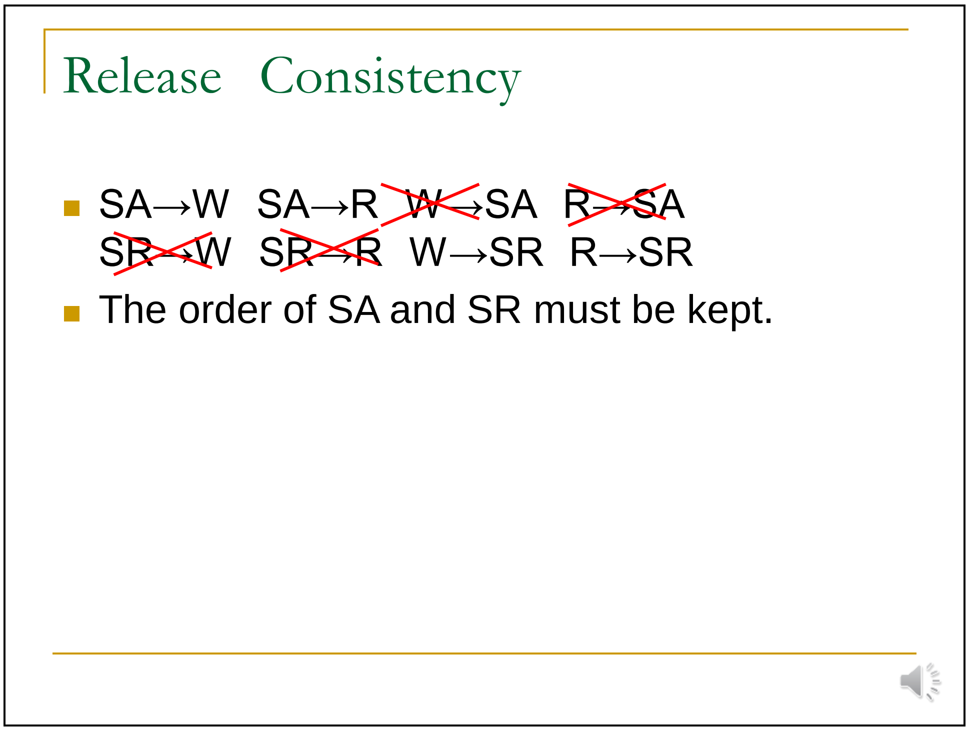
・All memory accesses following acquire（SA) are not executed until SA is finished.

・All memory accesses must be executed before release（SR) is finished.

・Synchronization operations must satisfy sequential consistency (RCsc)

・Used in a lot of CC-NUMA machines（DASH,ORIGIN）

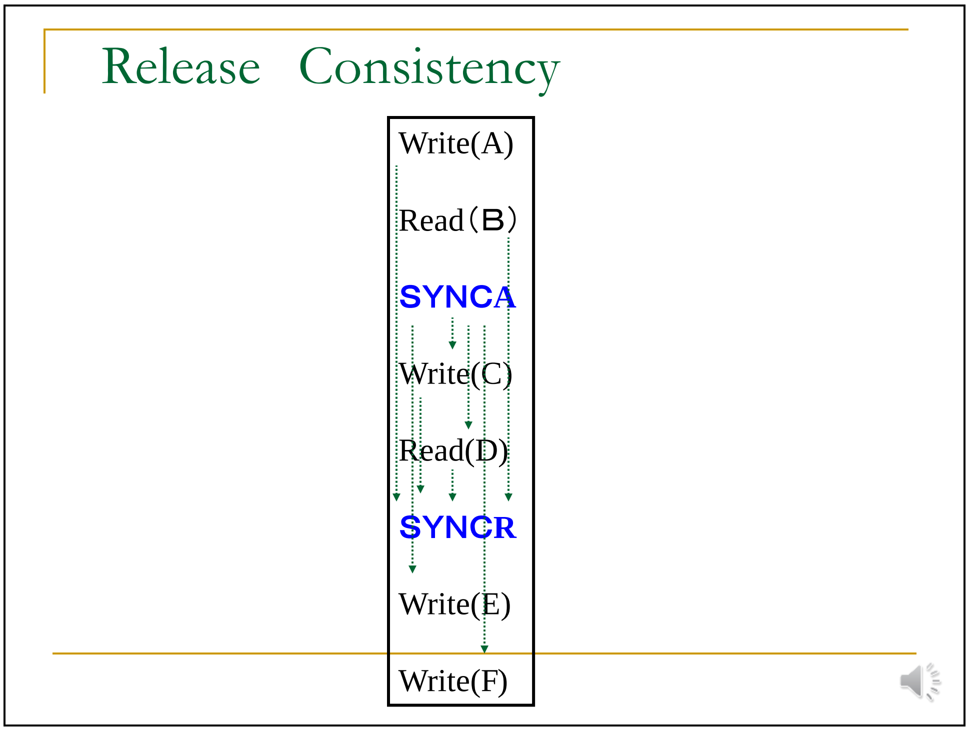
In this model, all memory accesses following acquire are not executed until the acquire is finished, and all memory accesses must be executed before release is finished. This is why it is called the release consistency model.



Release Consistency

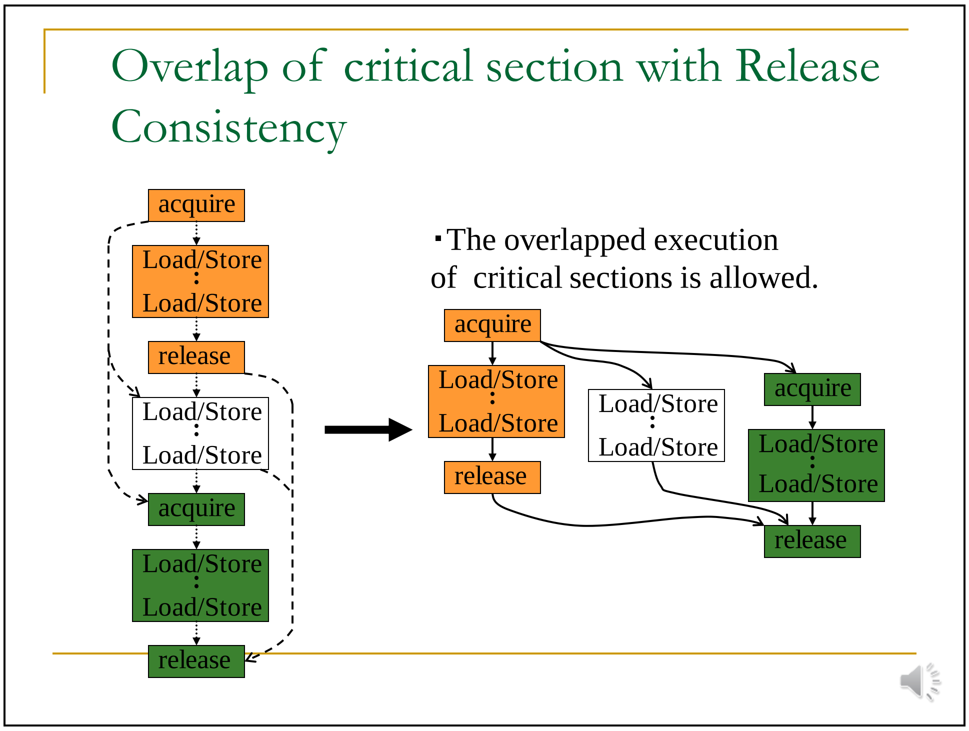
- SA→W SA→R ~~W→SA~~ ~~R→SA~~
  ~~SR→W~~ ~~SR→R~~ W→SR R→SR
- The order of SA and SR must be kept.

This is the relationship between acquire, release and other accesses. Of course, the order between SA and SR must be kept.



Release Consistency

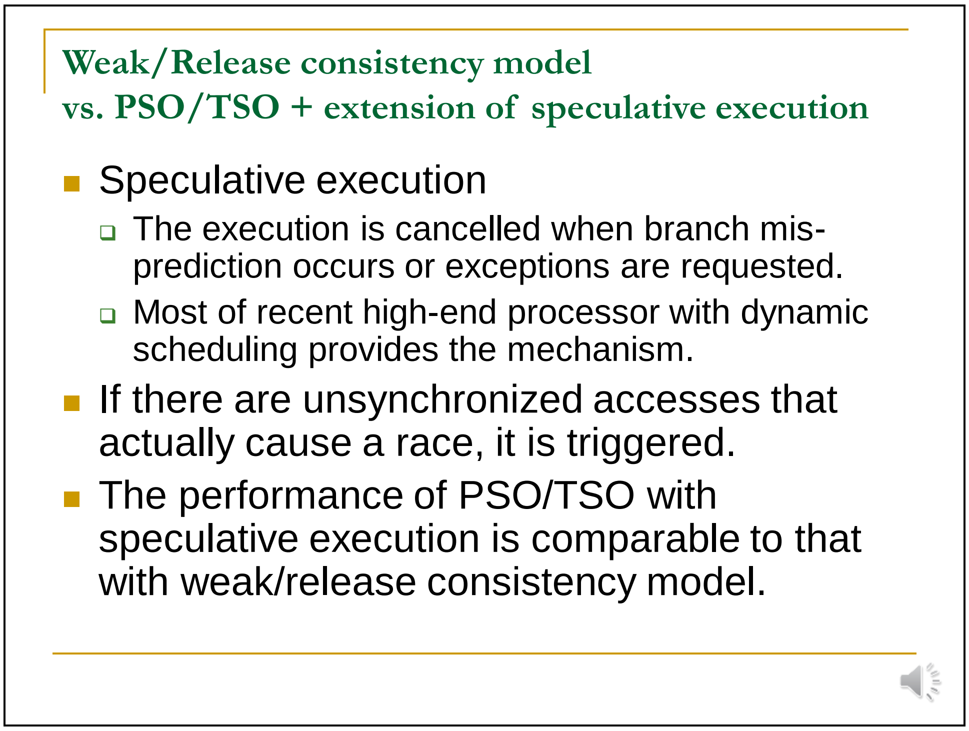
This shows the order which must be kept between multiple accesses.



Overlap of critical section with Release Consistency

- The overlapped execution of critical sections is allowed.

This diagram shows the benefit of the release consistency. Two critical sections can be executed in the overlapped manner.



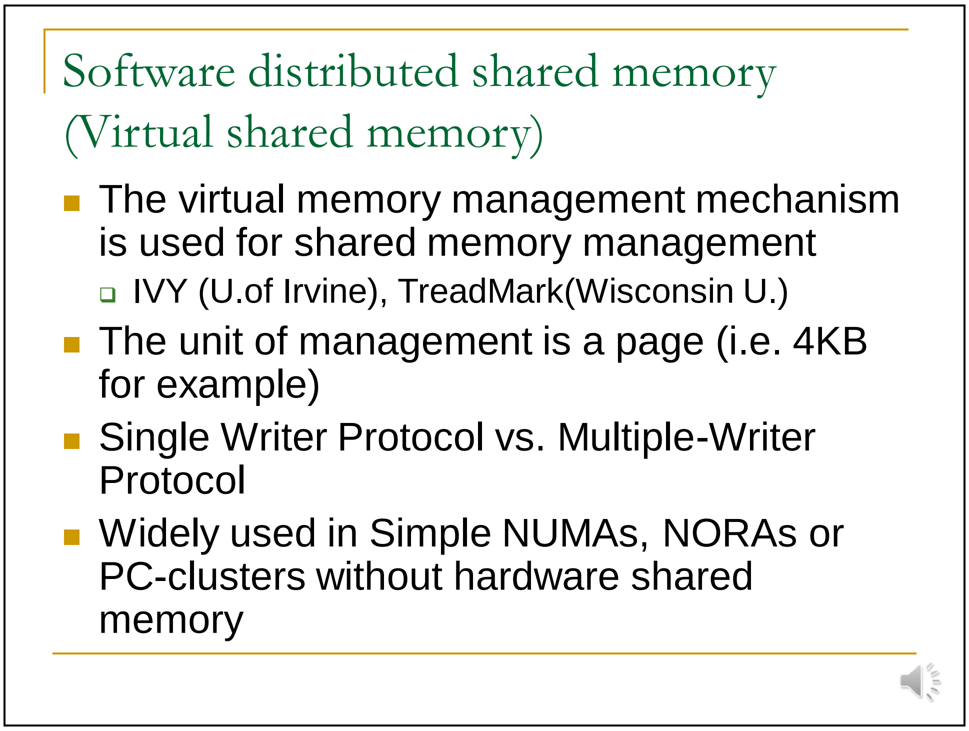
Weak/Release consistency model vs. PSO/TSO + extension of speculative execution

- Speculative execution
  - The execution is cancelled when branch mis-prediction occurs or exceptions are requested.
  - Most of recent high-end processor with dynamic scheduling provides the mechanism.
- If there are unsynchronized accesses that actually cause a race, it is triggered.
- The performance of PSO/TSO with speculative execution is comparable to that with weak/release consistency model.

Recently, most of high performance processors provide the speculative execution. This mechanism can be used instead of the release consistency model. That is, if there are unsynchronized that actually causes a race, the roll back is trigered. However, the detection is difficult and the rollback has a large overhead, so it is difficult to say which approach is better.

Glossary 1

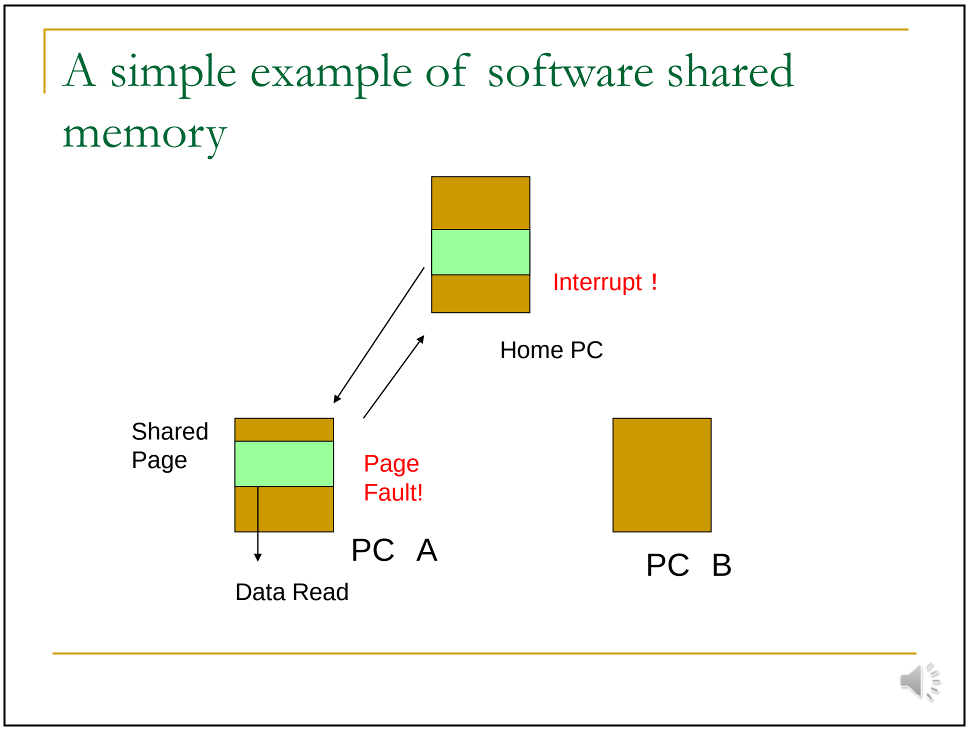
- Consistency Model: Consistencyは一貫性のことで、Snoop Cacheの所で出てきたが、異なったアドレスに対して考える場合に使う言葉。一方、Coherenceは同じアドレスに対して考える場合に用いる。
- Sequential Consistency model: 最も厳しいモデル、全アクセスの順序が保証される
- Relaxed Consistency model:Sequential Consistecy modelが厳しいすぎるので、これを緩めたモデル
- TSO(Total Store Ordering):書き込みの全順序を保証するモデル
- PSO(Partial Store Ordering):書き込みの順序を同期、読み出しが出てくる場合のみ保証するモデル
- Weak Consistency　弱い一貫性、同期のときのみ一貫性が保証される
- Release Consistency　同期のリリース時にのみ一般性が保証される。Acquire（獲得）がロック、Release（解放）がアンロック
- Synchronization, Critical Section：同期、際どい領域



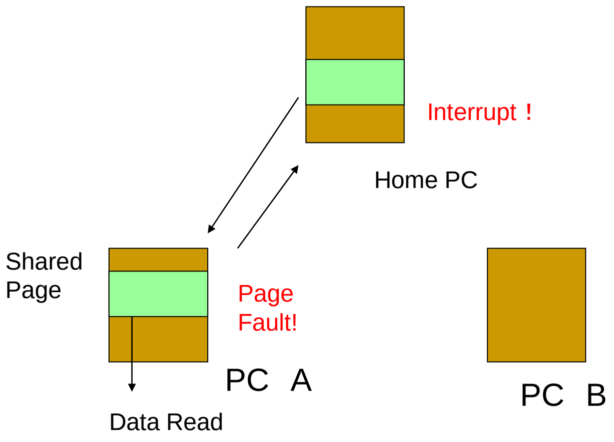
Software distributed shared memory (Virtual shared memory)

- The virtual memory management mechanism is used for shared memory management
  - IVY (U.of Irvine), TreadMark(Wisconsin U.)
- The unit of management is a page (i.e. 4KB for example)
- Single Writer Protocol vs. Multiple-Writer Protocol
- Widely used in Simple NUMAs, NORAs or PC-clusters without hardware shared memory

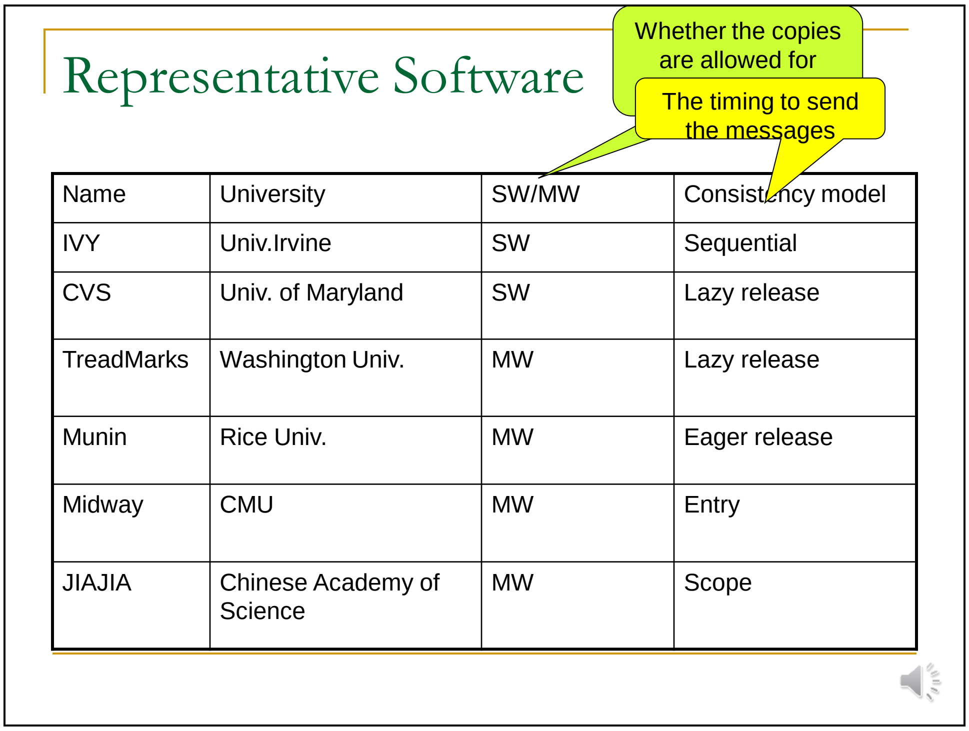
In the next segment, I am going to talk about the software distributed shared memory. This mechanism is introduced for the machine without the cache coherent mechanism. The ideal is to use the virtual memory management mechanism for keeping the shared memory. So, the unit of management is a page, not a cache block.



A simple example of software shared memory



Let me explain a simple example of software shared memory. Assume three processors share software shared memory. When a PC A requests to read a shared page which is allocated on this home PC, it causes the page fault. Instead of getting the page from the disk, it sends the request to the home PC. When the request message is received, the home PC is interrupted and the software manager is invoked. It returned the requested page to the PC A.



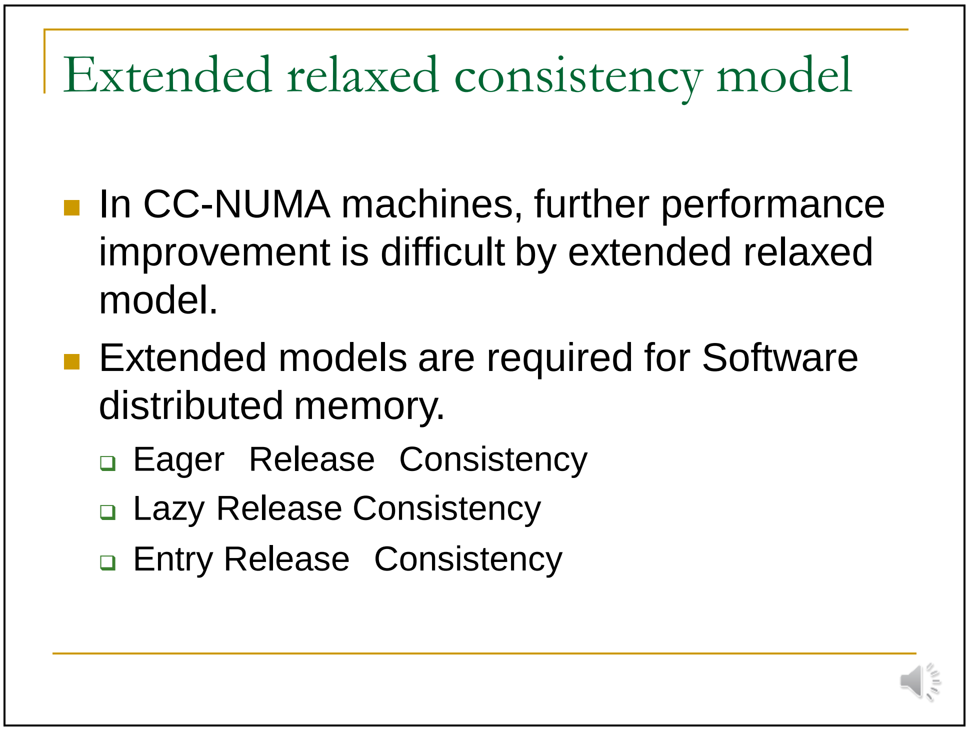
Representative Software

Whether the copies are allowed for

The timing to send the messages

Name	University	SW/MW	Consistency model
IVY	Univ.Irvine	SW	Sequential
CVS	Univ. of Maryland	SW	Lazy release
TreadMarks	Washington Univ.	MW	Lazy release
Munin	Rice Univ.	MW	Eager release
Midway	CMU	MW	Entry
JIAJIA	Chinese Academy of Science	MW	Scope

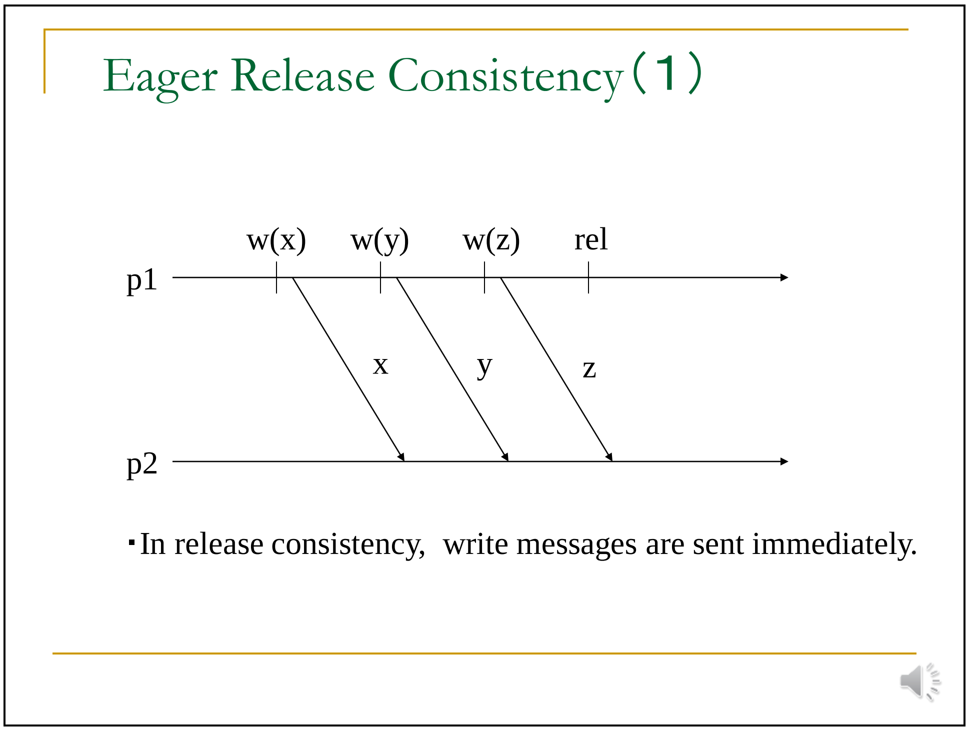
This table shows the representative software distributed memory systems. It is classified into single writer and multiple writers. And the timing to send the messages.



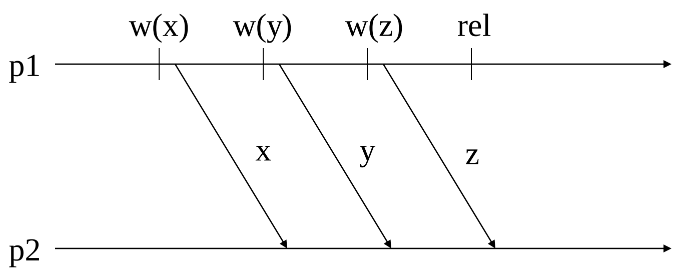
Extended relaxed consistency model

- In CC-NUMA machines, further performance improvement is difficult by extended relaxed model.
- Extended models are required for Software distributed memory.
  - Eager Release Consistency
  - Lazy Release Consistency
  - Entry Release Consistency

In CC-NUMA machines, further performance improvement is difficult by extending the release consistency. However, for Software distributed memory, extended models are required. Thus, various types of extended consistency model has been proposed.

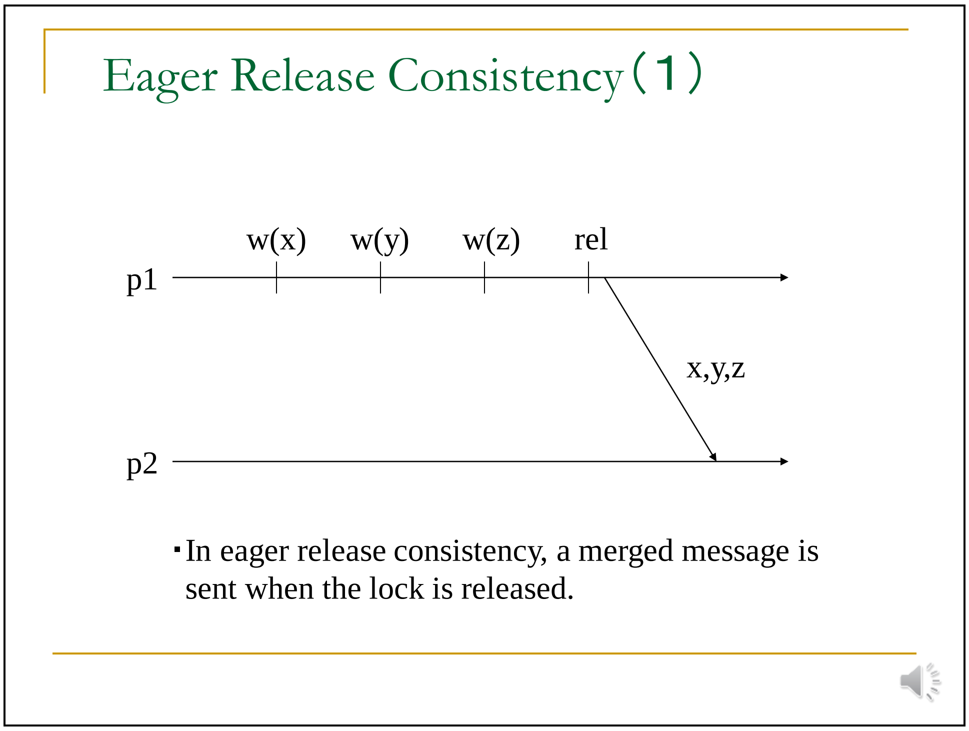


Eager Release Consistency（1）

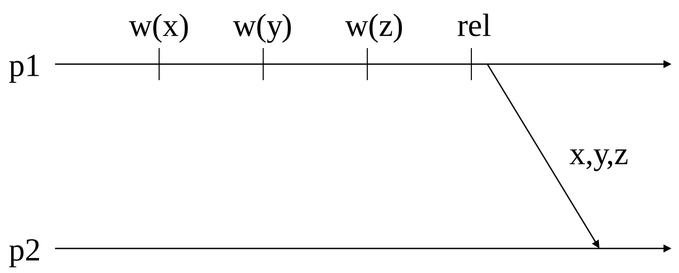


- In release consistency, write messages are sent immediately.

Eager Release Consistency is used in Munin. It reduces the number of message transfers. In release consistency, write messages are sent immediately.

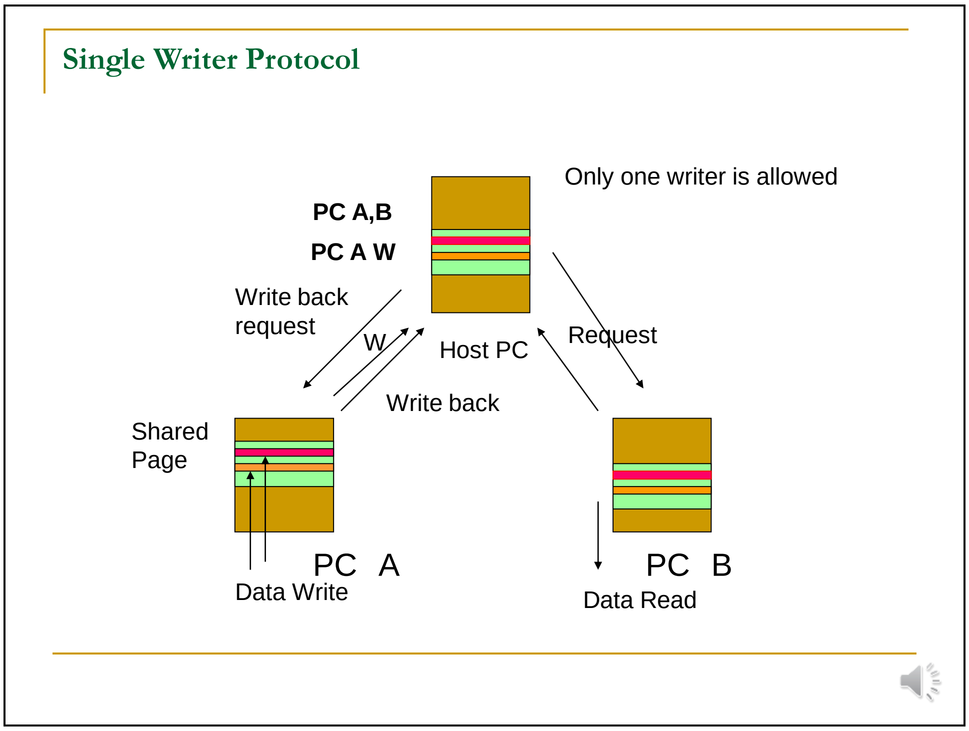


Eager Release Consistency（1）



- In eager release consistency, a merged message is sent when the lock is released.

However, in eager release consistency, a merged message is sent when the lock is released.

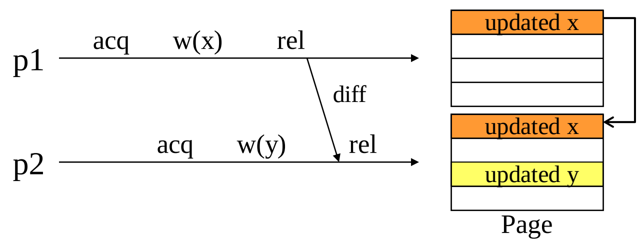


Single Writer Protocol

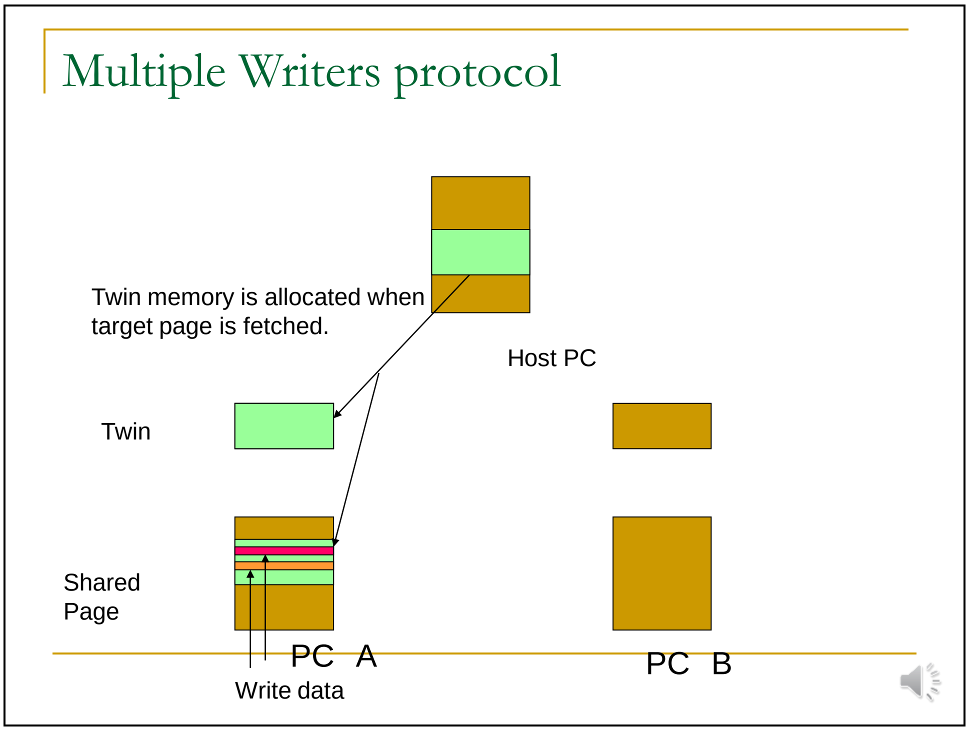
This diagram shows the situation, in this case, only one writer is allowed.

Eager Release Consistency (2)

▪In Multiple-Writer Protocol, only difference is sent when released.



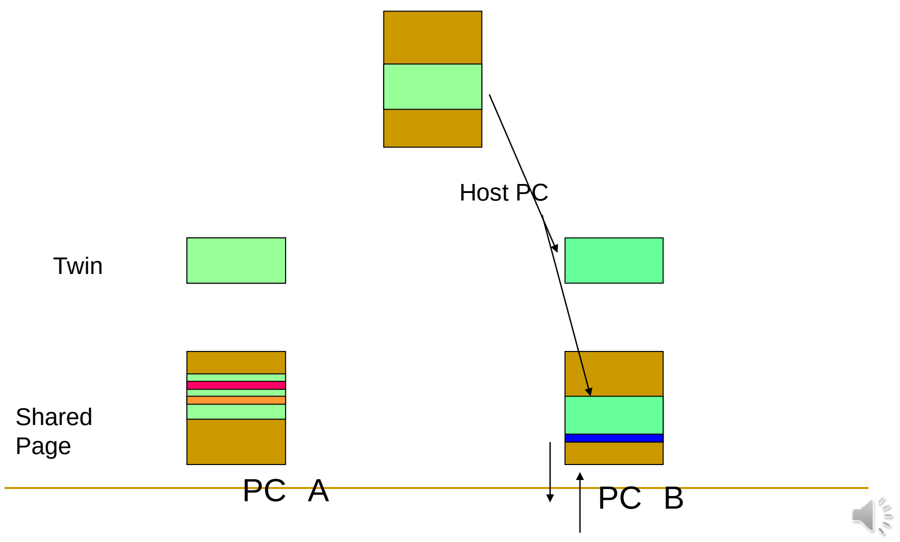
Eager release consistency can be extended to multiple write protocol. In this case, only difference is sent when released.



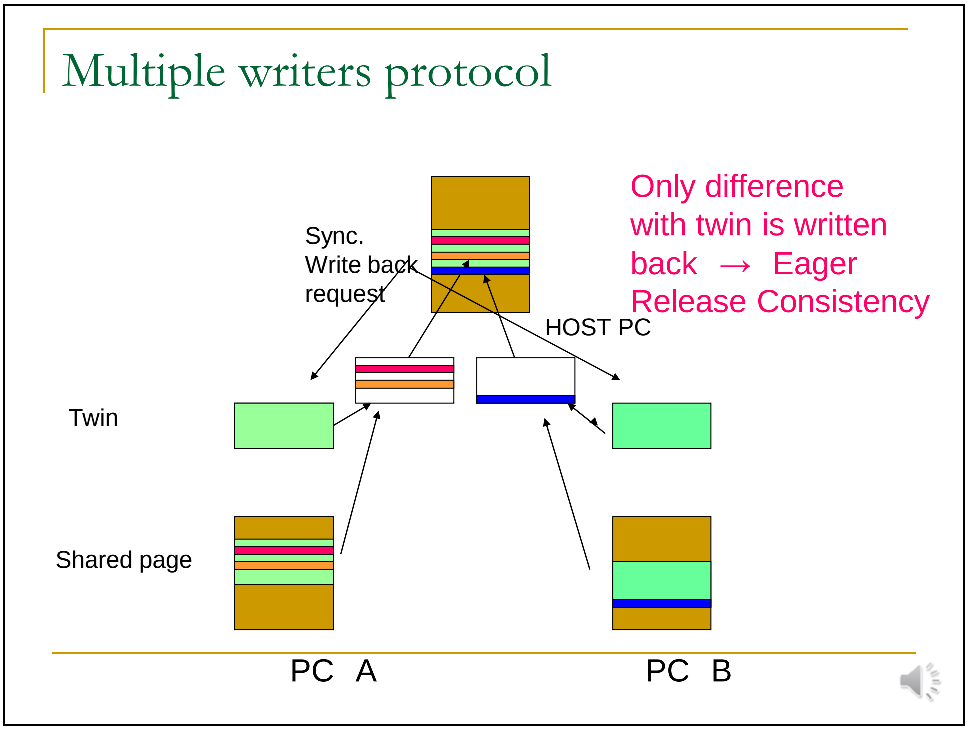
Multiple Writers protocol

This diagram shows how the multiple writers protocol works. In this case, a twin memory is allocated when the target page is fetched.

Multiple Writers protocol



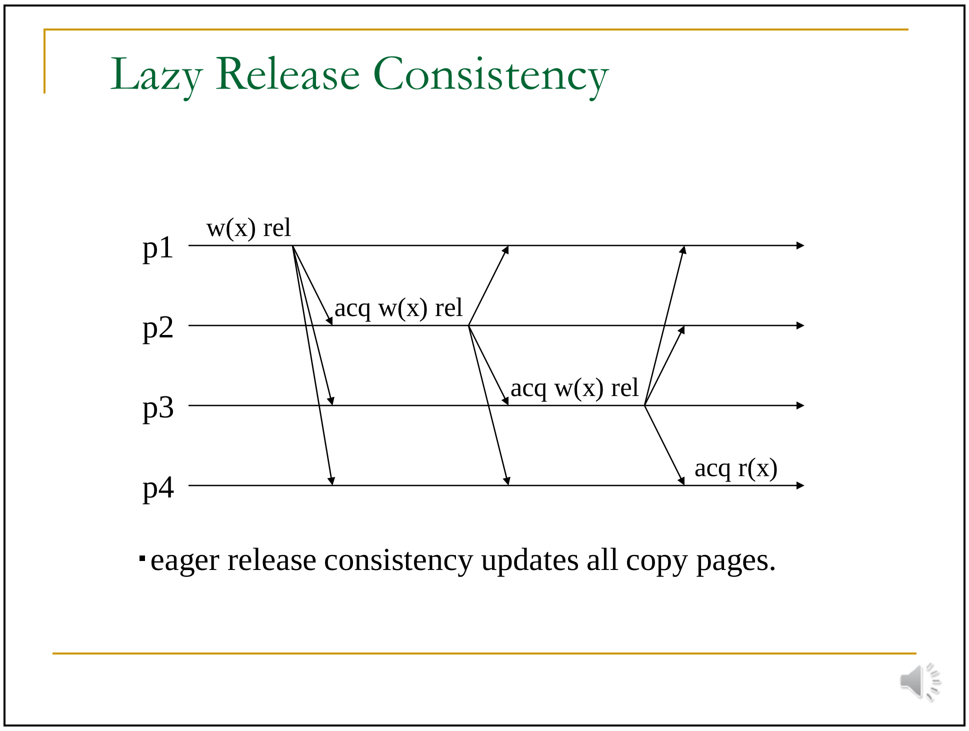
Multiple writers protocol allows for multiple PUs to write the same page at the same time.



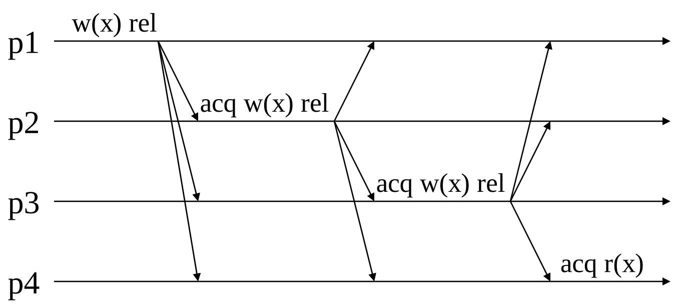
Multiple writers protocol

Only difference with twin is written back → Eager Release Consistency

When write back is needed, the only difference with twin is written back.

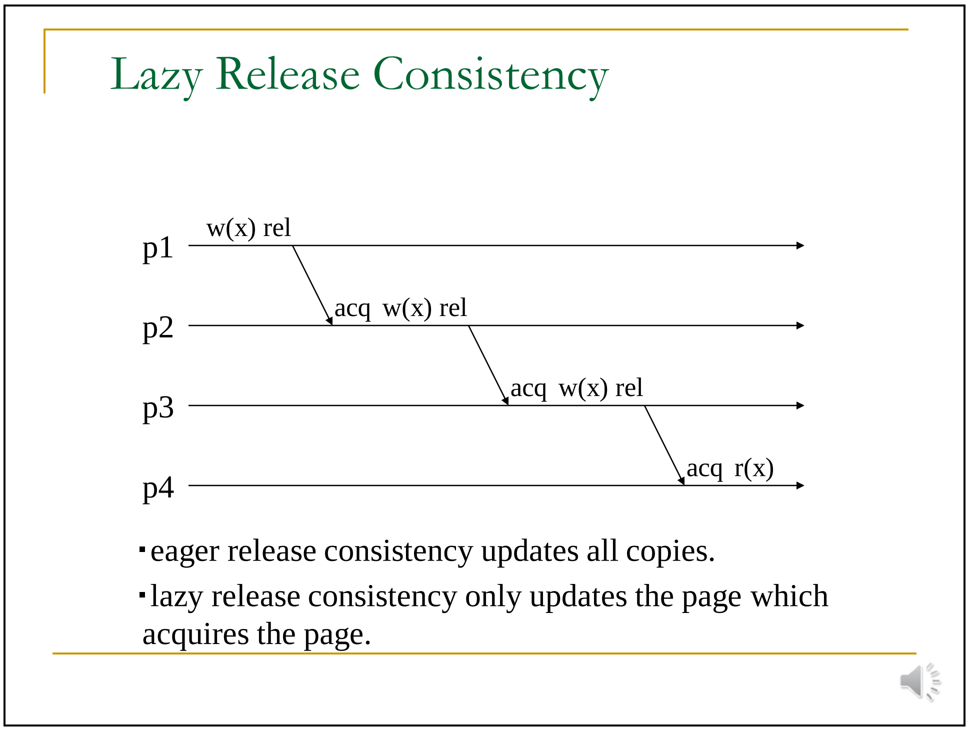


Lazy Release Consistency

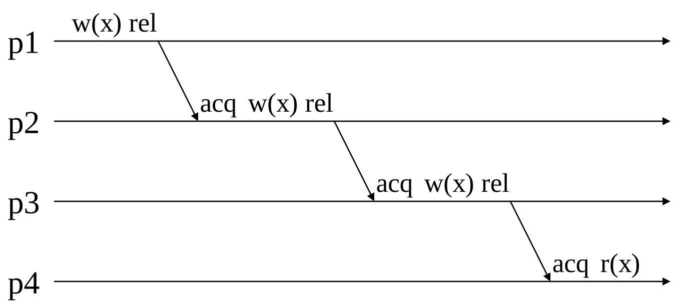


- eager release consistency updates all copy pages.

The eager release consistency updates all copies pages when they are released.

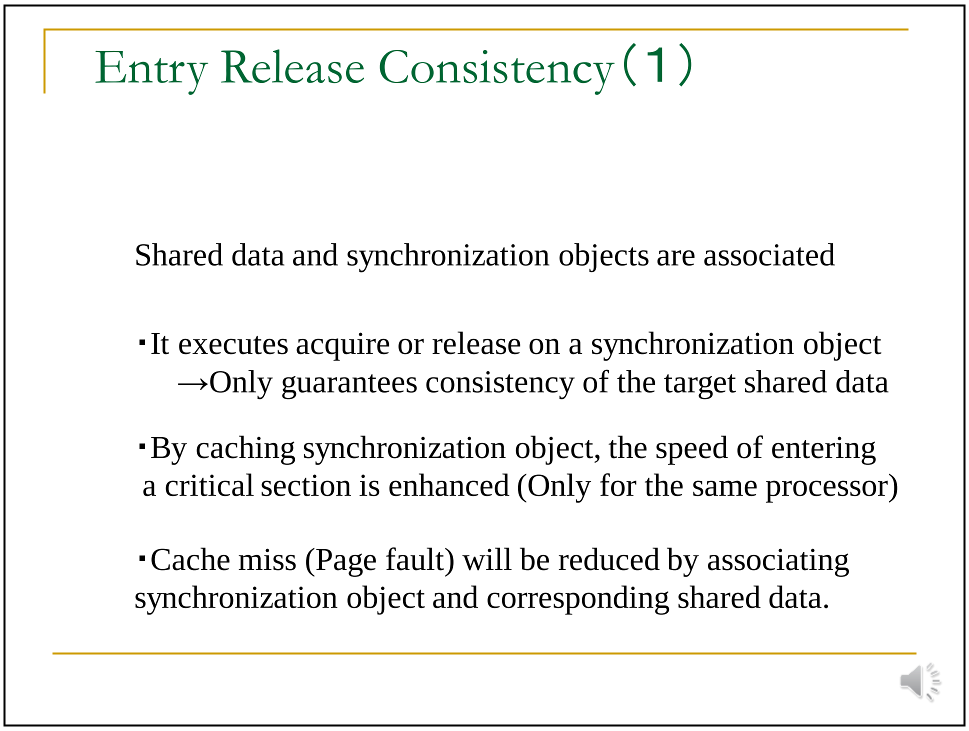


Lazy Release Consistency



- eager release consistency updates all copies.
- lazy release consistency only updates the page which acquires the page.

In order to reduce the number of messages, lazy release consistency only updates page which



Entry Release Consistency（1）

Shared data and synchronization objects are associated

･It executes acquire or release on a synchronization object
→Only guarantees consistency of the target shared data

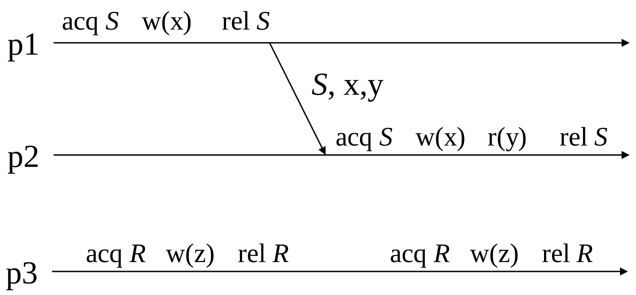
･By caching synchronization object, the speed of entering a critical section is enhanced (Only for the same processor)

･Cache miss (Page fault) will be reduced by associating synchronization object and corresponding shared data.

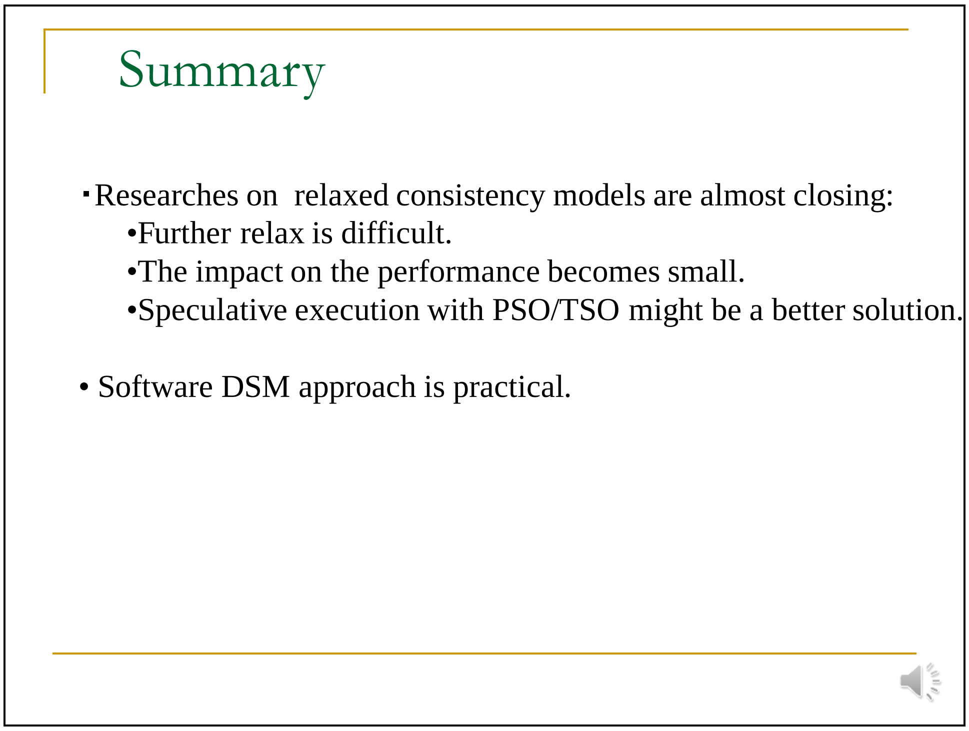
Entry release consistency fixes the combination of a shared data and associated synchronization object. The operation for keeping consistency is done only for the target data.

Entry Release Consistency (2)

- synchronization object *S* ⇔ shared data x,y
- synchronization object *R* ⇔ shared data z


p1
acq *S* w(x) rel *S*
S, x,y
p2
acq *S* w(x) r(y) rel *S*
p3
acq *R* w(z) rel *R* acq *R* w(z) rel *R*

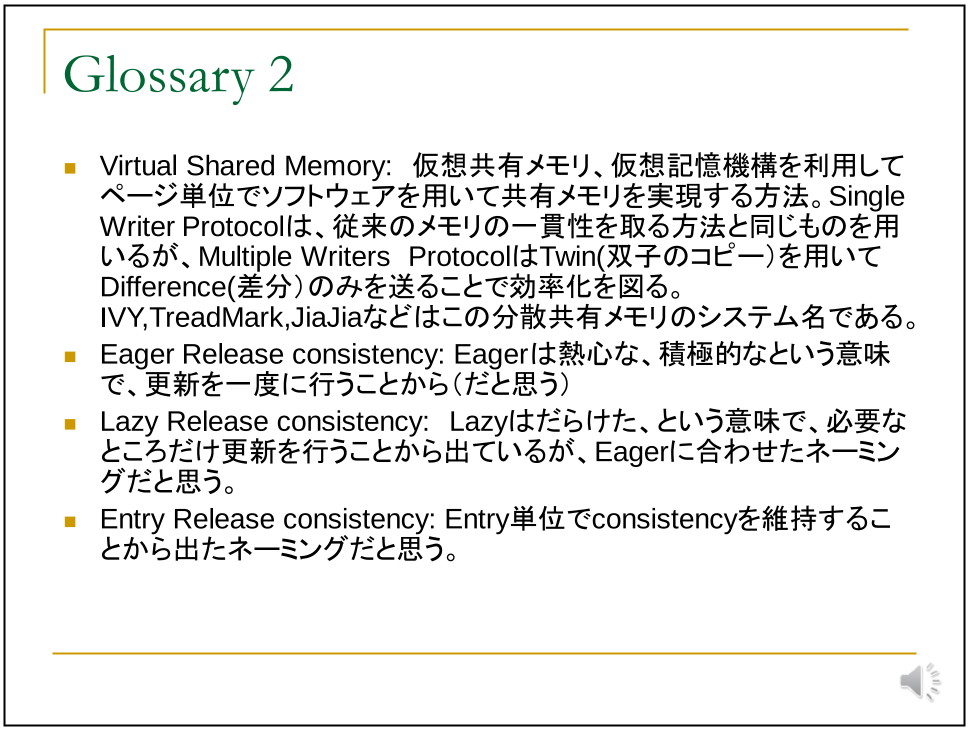
Assume that the synchronization object S is associated into the shared data x,y and synchronization object R is associated to the shared data z. The illustrated data exchange is only required.



Summary

- Researches on relaxed consistency models are almost closing:
  - Further relax is difficult.
  - The impact on the performance becomes small.
  - Speculative execution with PSO/TSO might be a better solution.
- Software DSM approach is practical.

Let me explain today's lesson.



Glossary 2

- Virtual Shared Memory:　仮想共有メモリ、仮想記憶機構を利用してページ単位でソフトウェアを用いて共有メモリを実現する方法。Single Writer Protocolは、従来のメモリの一貫性を取る方法と同じものを用いるが、Multiple Writers　ProtocolはTwin(双子のコピー)を用いてDifference(差分)のみを送ることで効率化を図る。IVY,TreadMark,JiaJiaなどはこの分散共有メモリのシステム名である。
- Eager Release consistency: Eagerは熱心な、積極的なという意味で、更新を一度に行うことから(だと思う)
- Lazy Release consistency:　Lazyはだらけた、という意味で、必要なところだけ更新を行うことから出ているが、Eagerに合わせたネーミングだと思う。
- Entry Release consistency: Entry単位でconsistencyを維持することから出たネーミングだと思う。

Exercise

- Which order should be kept in the following access sequence when TSO,PSO and WO are applied respectively.

SYNC
Write
Write
Read
Read
SYNC
Read
Write
Write
SYNC