

CHAPTER 7: DISTRIBUTED SHARED MEMORY

DSM simulates a logical shared memory address space over a set of physically distributed local memory systems.

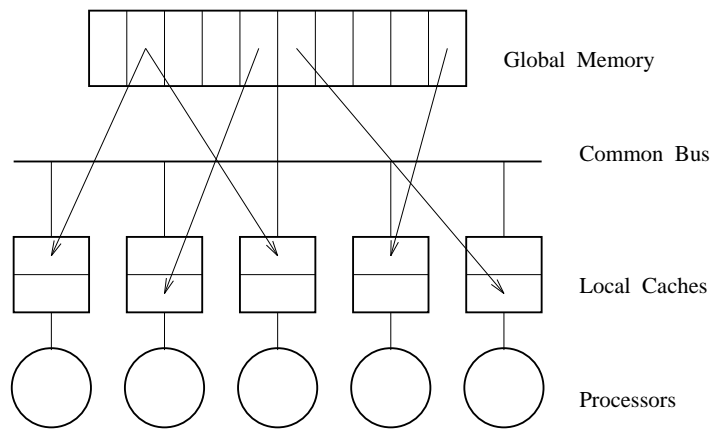
Why DSM?

- direct information sharing programming paradigm (transparency)
- multilevel memory access (locality)
- wealth of existing programs (portability)
- large physical memory
- scalable multiprocessor system

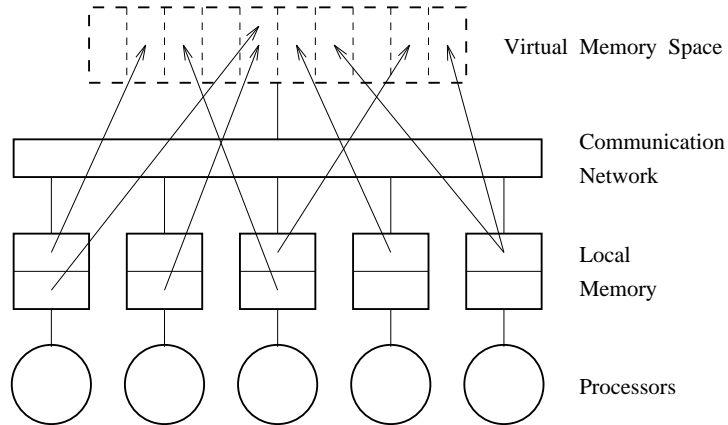
Chapter outline

- NUMA architectures: similarity between multiprocessor cache and DSM systems
- Memory consistency models: why is memory consistency a more critical problem in multiprocessor and DSM systems? how is memory consistency defined?
- Cache coherency protocols: implementation of consistency models
- DSM Implementation: applying the consistency models and coherency protocols to a DSM system

Nonuniform Memory Access (NUMA) architectures



(a) Multiprocessor cache architecture



(b) Distributed shared memory architecture

Common issues

Data consistency and coherency due to data placement, migration and replication

Memory consistency models

Atomic consistency

Sequential consistency

P_1 : $W(X)1$
 P_2 : $W(Y)2$
 P_3 : $R(Y)2$ $R(X)0$ $R(X)1$

Causal consistency

P_1 : $W(X)1$ $W(X)3$
 P_2 : $R(X)1$ $W(X)2$
 P_3 : $R(X)1$ $R(X)3$ $R(X)2$
 P_4 : $R(X)1$ $R(X)2$ $R(X)3$

Processor consistency

P_1 : $W(X)1$
 P_2 : $R(X)1$ $W(X)2$
 P_3 : $R(X)1$ $R(X)2$
 P_4 : $R(X)2$ $R(X)1$

Slow memory consistency

P_1 : $W(X)1$ $W(Y)2$ $W(X)3$
 P_2 : $R(Y)2$ $R(X)1$ $R(X)3$

The above models apply consistency constraints to all memory accesses.

Applying consistency constraints only to synchronization accesses.

Weak consistency

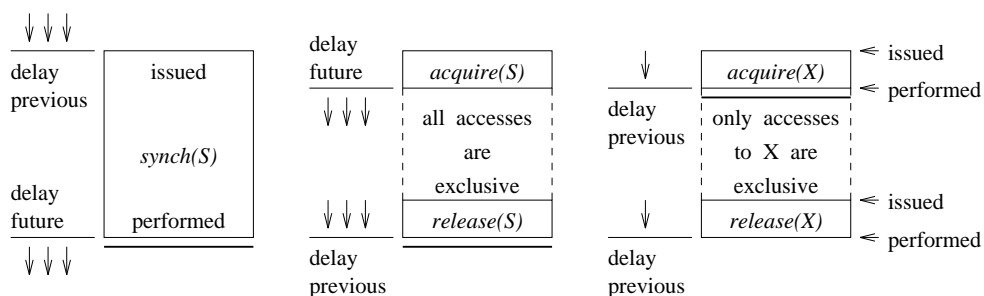
- Accesses to synchronization variables are sequentially consistent
- No access to a synchronization variable is issued by a processor before all previous *read/write* data accesses have been performed
- No *read/write* data access is issued by a processor before a previous access to a synchronization variable has been performed

Release consistency

The synchronization access ($synch(S)$) in the weak consistency model can be refined as a pair of $acquire(S)$ and $release(S)$ accesses. Shared variables in the critical section are made consistent when the release operation is performed.

Entry consistency

$acquire$ and $release$ are applied to general variables.

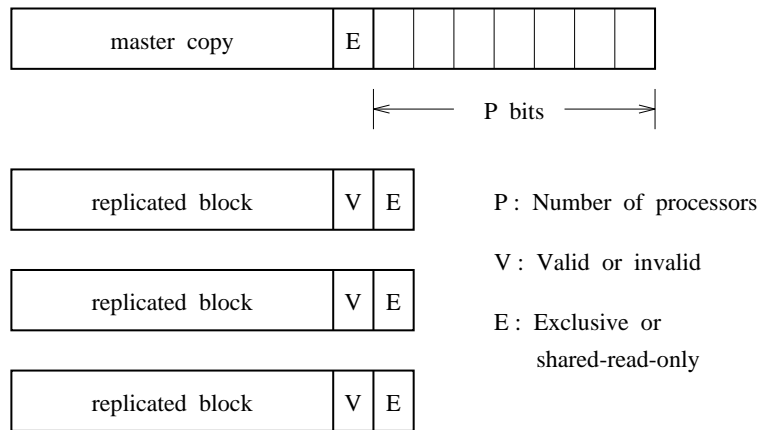


(a) Weak consistency

(b) Release consistency

(c) Entry consistency

Cache directory



Cache coherency protocols

write-invalidate and write-update

Write-invalidate

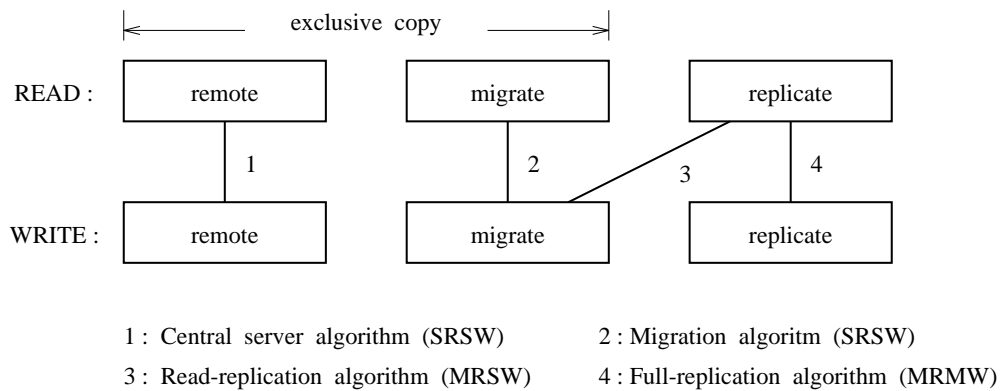
- Read hit
- Read miss: transfer block, set P-, V-, and E-bit.
- Write hit: invalidate cache copies, write and set E-bit
- Write miss

Hardware mechanisms

- Directory-based
- Snooping cache

DSM implementation

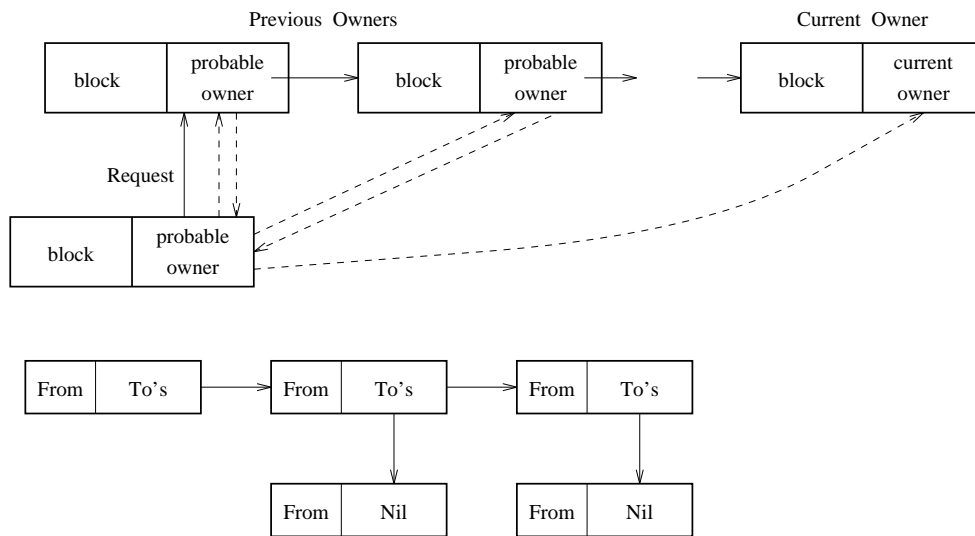
Memory management algorithms



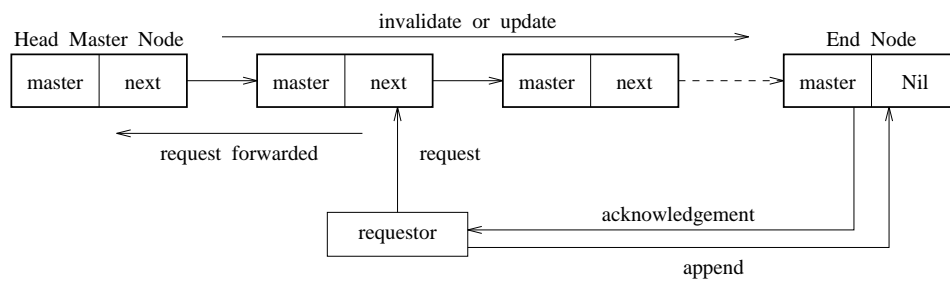
- Read-remote-write-remote: long network delay, trivial consistency
- Read-migrate-write-migrate: trashing and false sharing
- Read-replicate-write-migrate: write-invalidate
- Read-replicate-write-replicate; full concurrency, atomic update

Distributed implementation of *directory*

Block owner and copy list:



(a) Spanning tree representation of copy set



(b) Linked list representation of copy set