On the Design and Implementation of DSM-Threads

Frank Mueller

Institut für Informatik
Humboldt-Universität zu Berlin
10099 Berlin

e-mail:
mueller@informatik.hu-berlin.de
WWW:
`http://www.informatik.hu-berlin.de/~mueller`

Overview

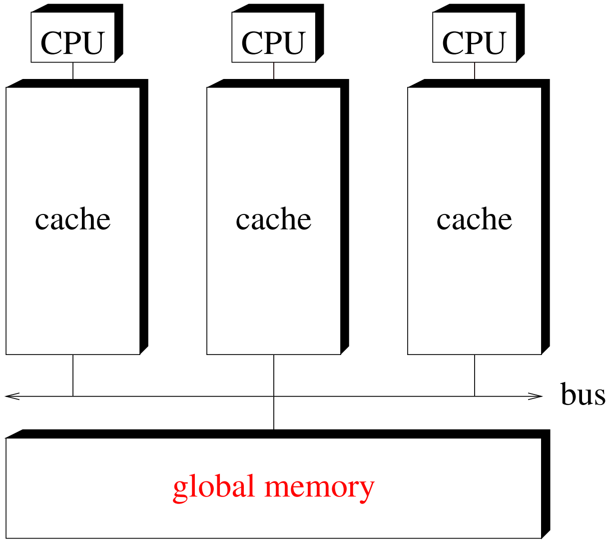
- motivation
- foundation
- distributed threads
- consistency models
- synchronization
- priorities
- heterogeneous environments
- speculative migration
- fault tolerance
- future directions

Motivation

- distributed threads over POSIX Threads
- shared memory via DSM
- concurrent $\Rightarrow$ distributed execution
- retain program model (shared memory)
- exploit NOWs

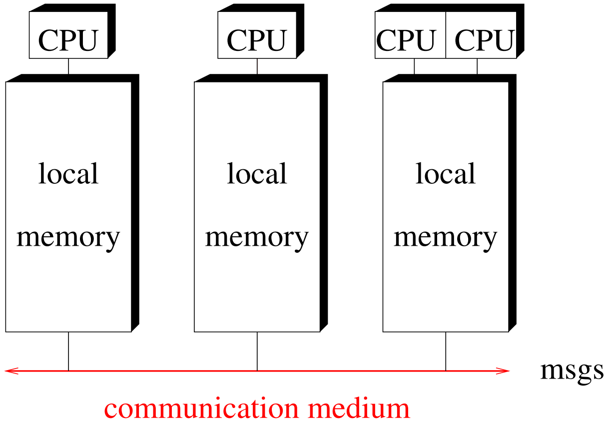
SMPs

+ concurrent programming
+ synchronization ⇒ consistency
+ transparent data access
– scalability limited
– special hardware
– latencies



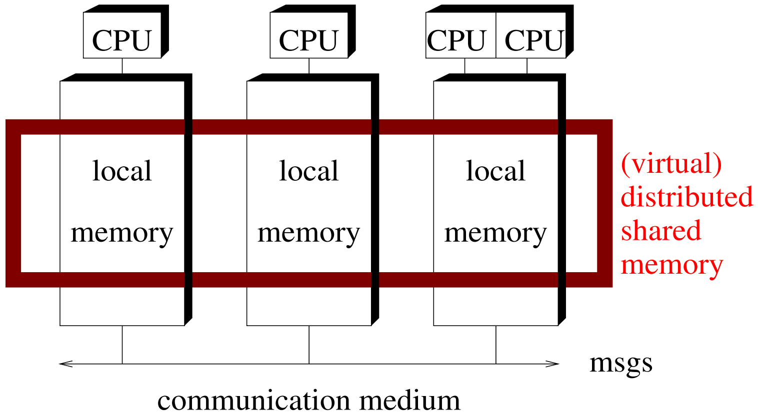
Distributed Systems

+ more scalable ⇒ medium
+ normal hardware, COTS
+ latency can be controlled
– no global state/data
– expensive messages
– explicit remote data references



DSM

+ concurrent programming
+ synchronization $\Rightarrow$ consistency
+ data access transparent
+ scalable
+ CTOS
+ fewer messages (caching)



Design Goals

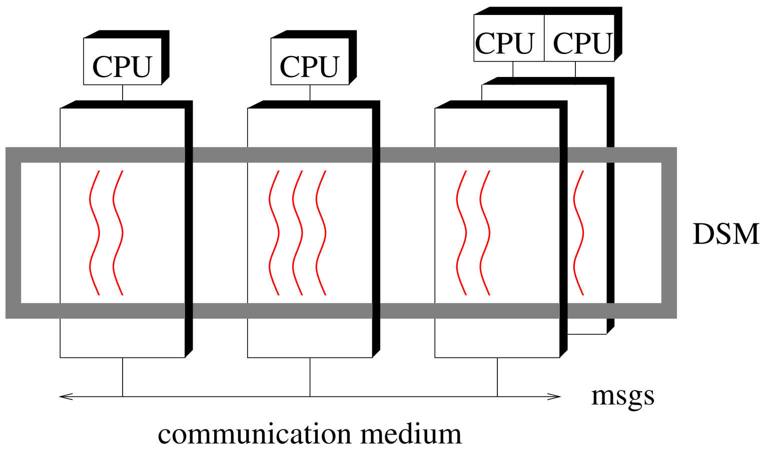
- portable: run-time system
- compatible: POSIX Threads
- communication: efficient, scalable
- extensible, tunable
- heterogeneous support
- modular (subsystems, protocols)
- multiple paradigms
- existing compilers/operating systems

Distributed Threads

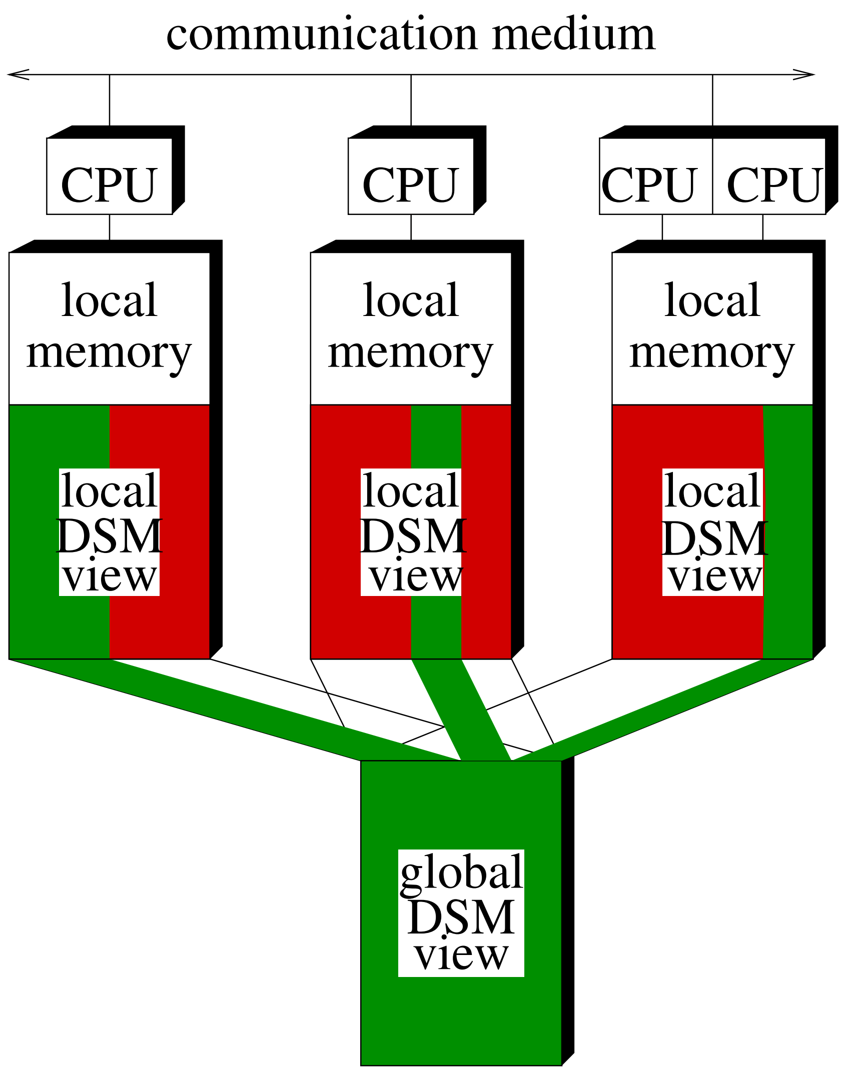
• POSIX Threads locally	`pthread_thread_create()`
• DSM Threads remote	`dsm_thread_create()`
• Synchronization:	
- local	`pthread_mutex/cond_*()`
- remote	`dsm_mutex/cond_*()`, `dsm_barrier()`
• data access:	
- local	efficient
- remote	1. costly (migration), 2.-k. cheap (local)
⇒ NUMA	

System Components

- node = (host, process)
- process:
  - DSM user threads
  - DSM system threads
  - POSIX user threads
- system threads: worker model
- protocol for access faults



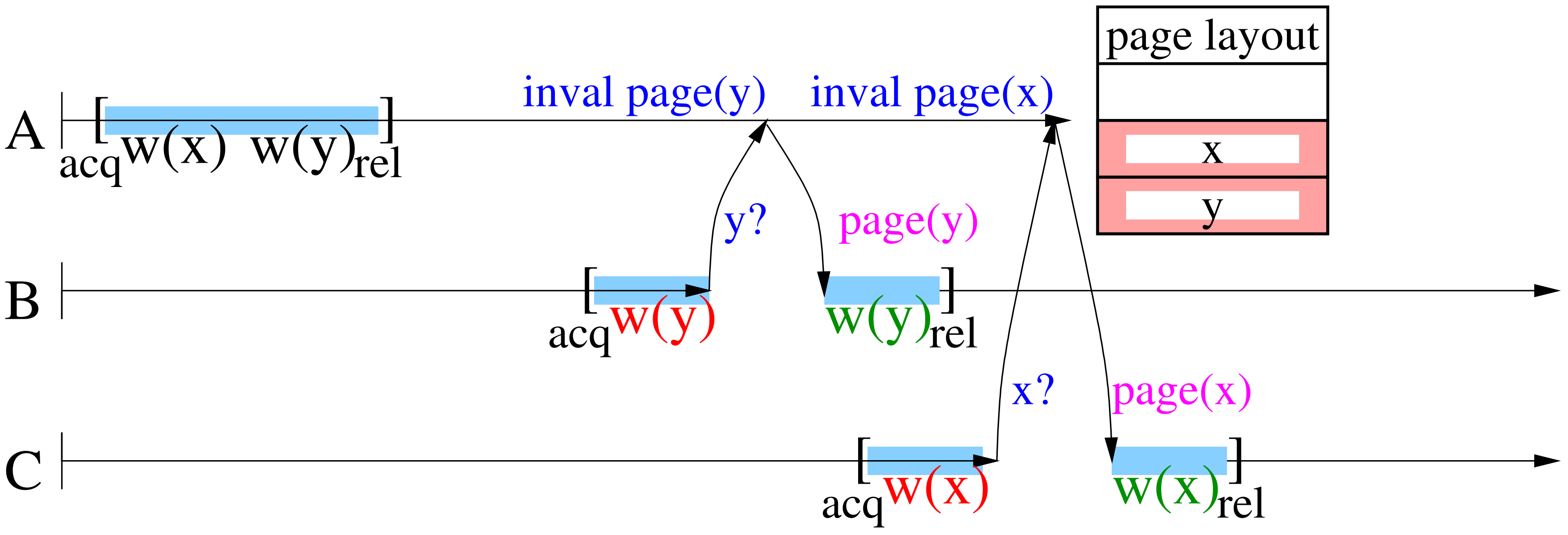
DSM Model


communication medium
CPU
CPU
CPU
CPU
local memory
local memory
local memory
local DSM view
local DSM view
local DSM view
global DSM view

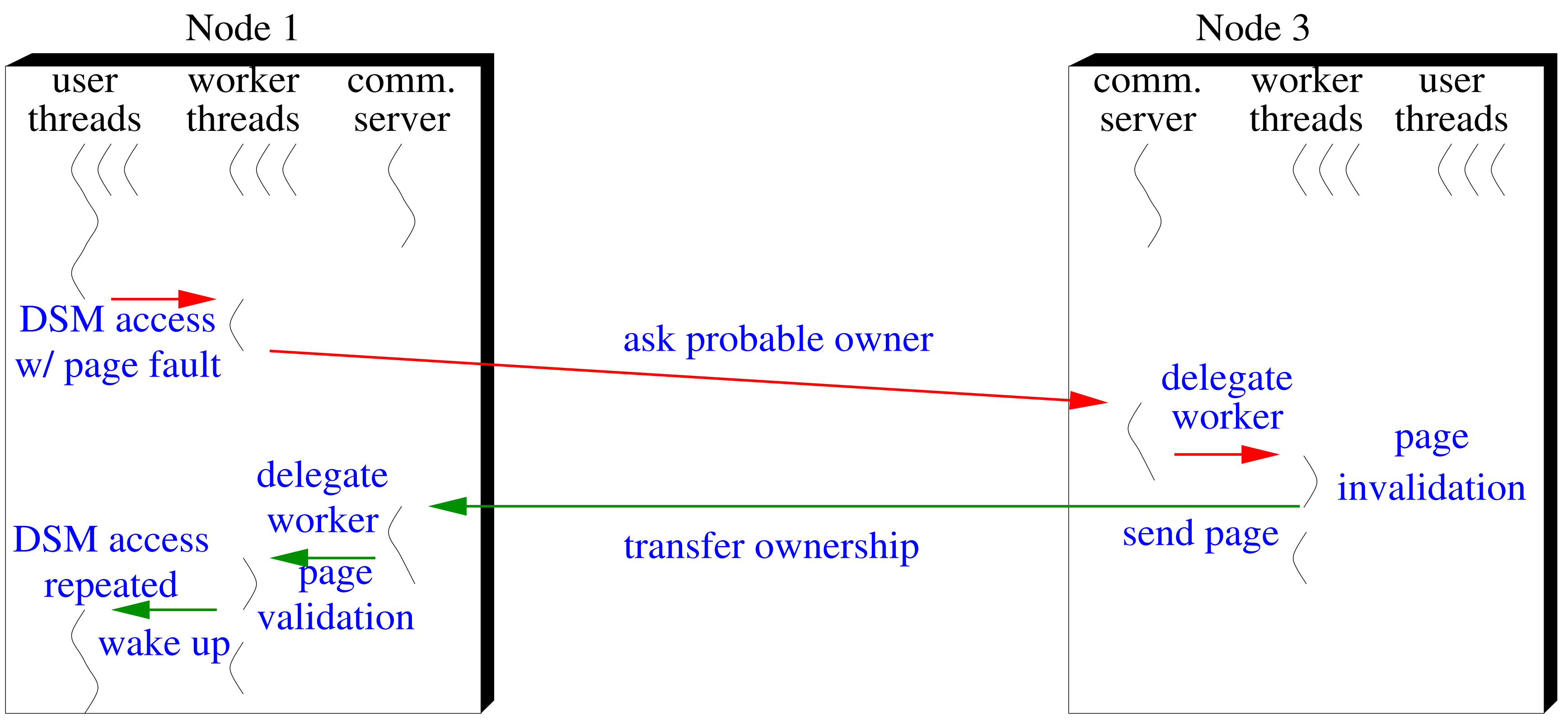
Consistency Models

- \+ SRSW/strict c.: conventional, 1-W $\rightarrow$ 1-R
- \+ MRSW/sequential c.: concurrent, n-W $\rightarrow$ last-R
- • MRMW/weak c.: send Ws when synchronizing
  - • eager release c.: broadcast Ws at release
  - \+ lazy release c.: Ws requested at acquire
  - \+ scope c.: encapsulate data implicitly with locks
- \+ OO/entry c.: encapsulate data explicitly with locks
- $\Rightarrow$ tuning: MRSW $\rightarrow$ MRMW $\rightarrow$ OO

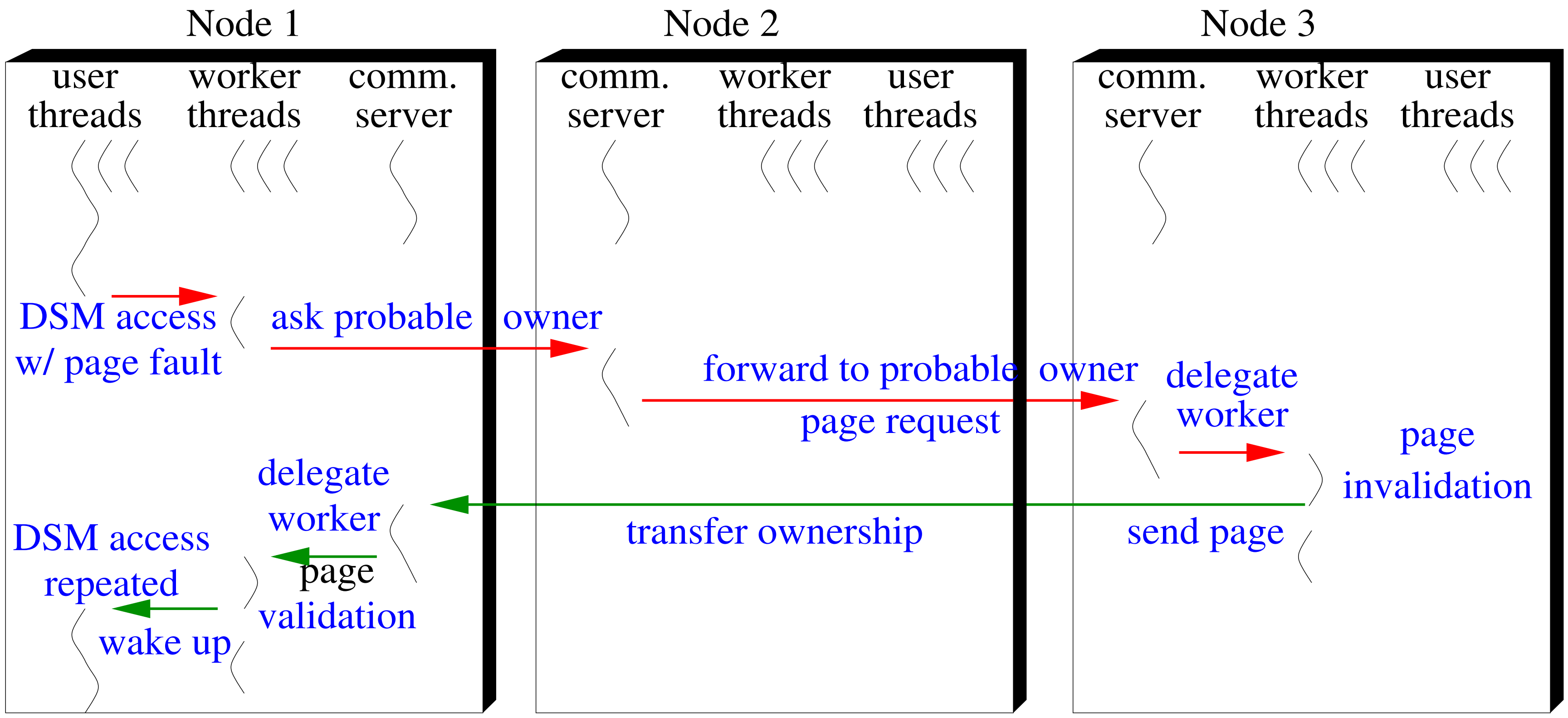
Sequential Consistency



Page Fault Model



Page Fault Model



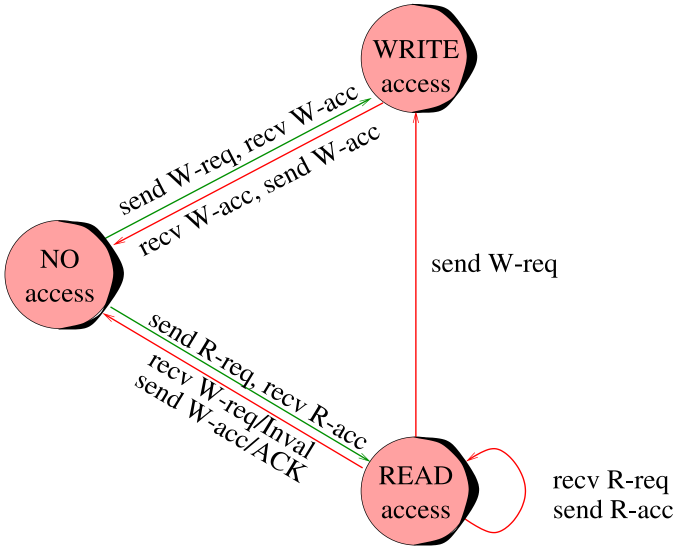
Communication

- sequential consistency
- multiple threads per node

⇒ asynchronous msg passing

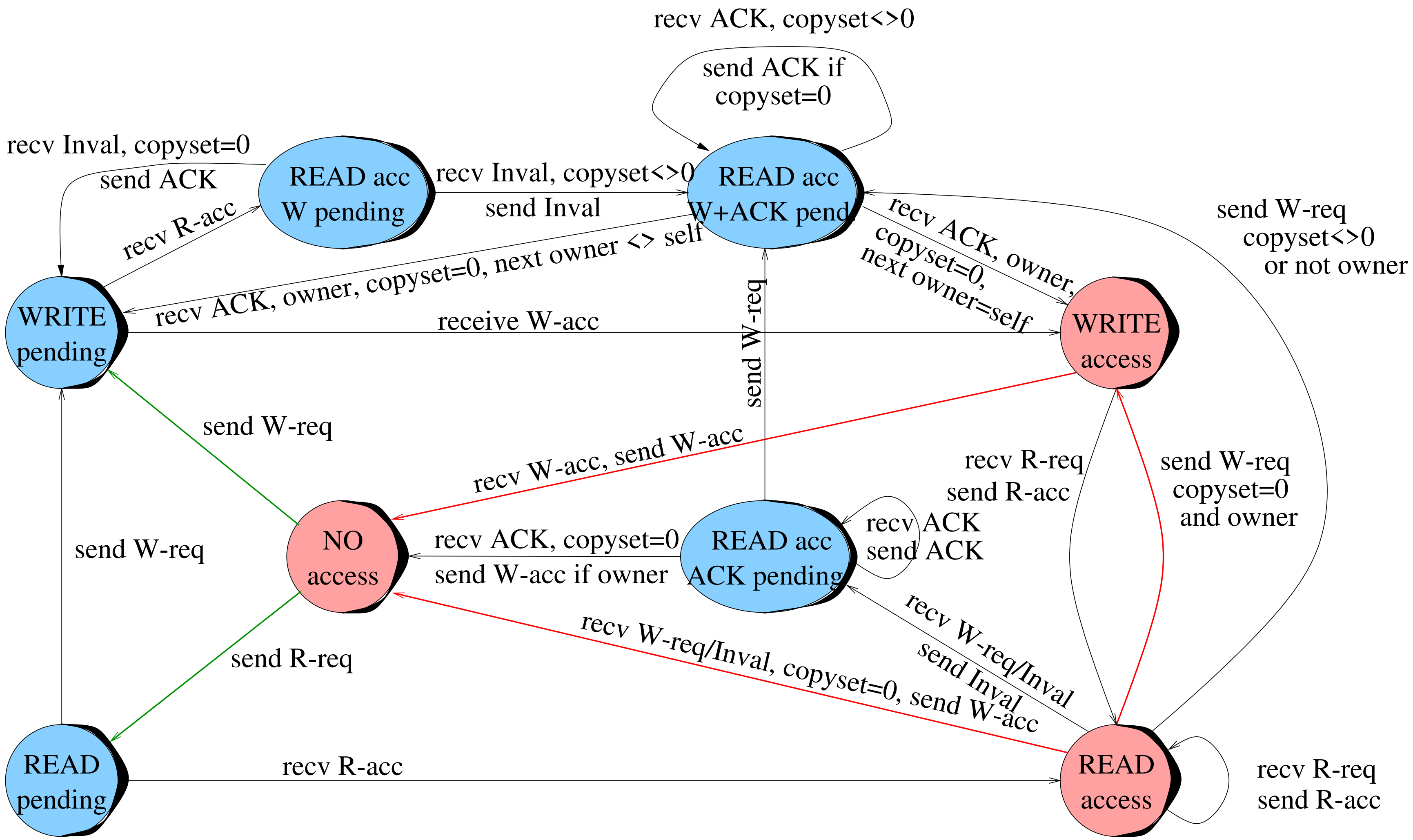
⇒ responsiveness

- extended consistency model
  - read data between request and response
  - forward requests while request pending
  - direct transitions (R ↔ W)

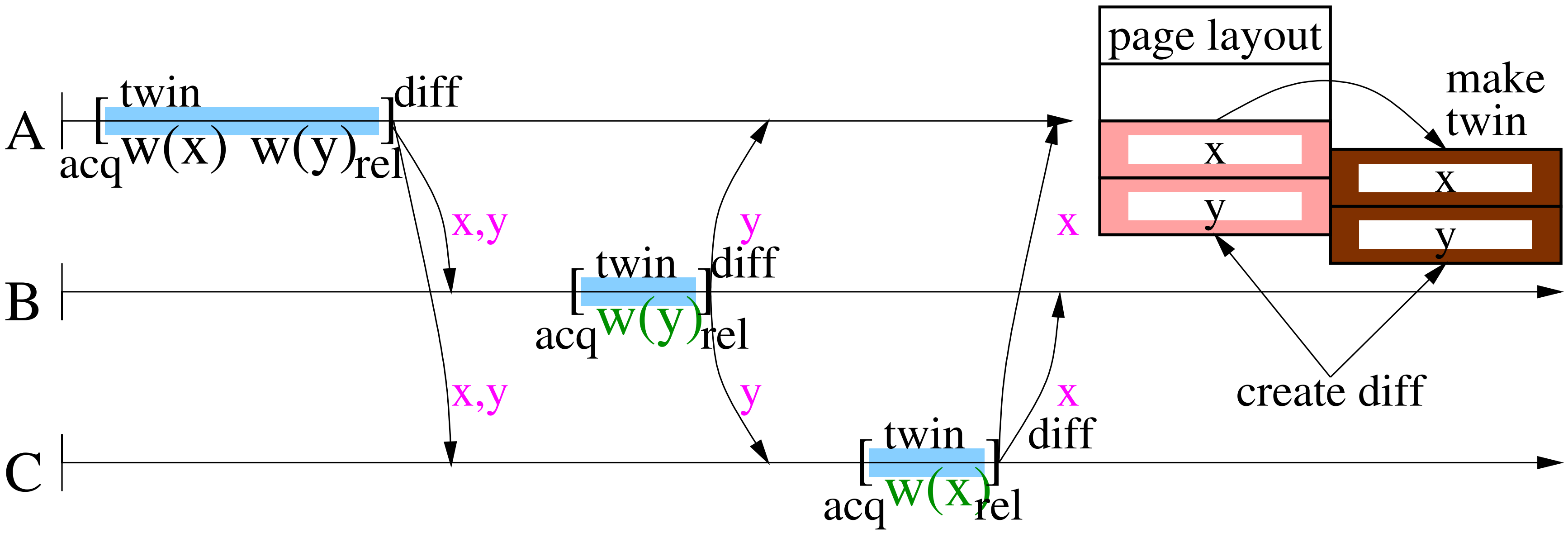


Synchronous MRSW

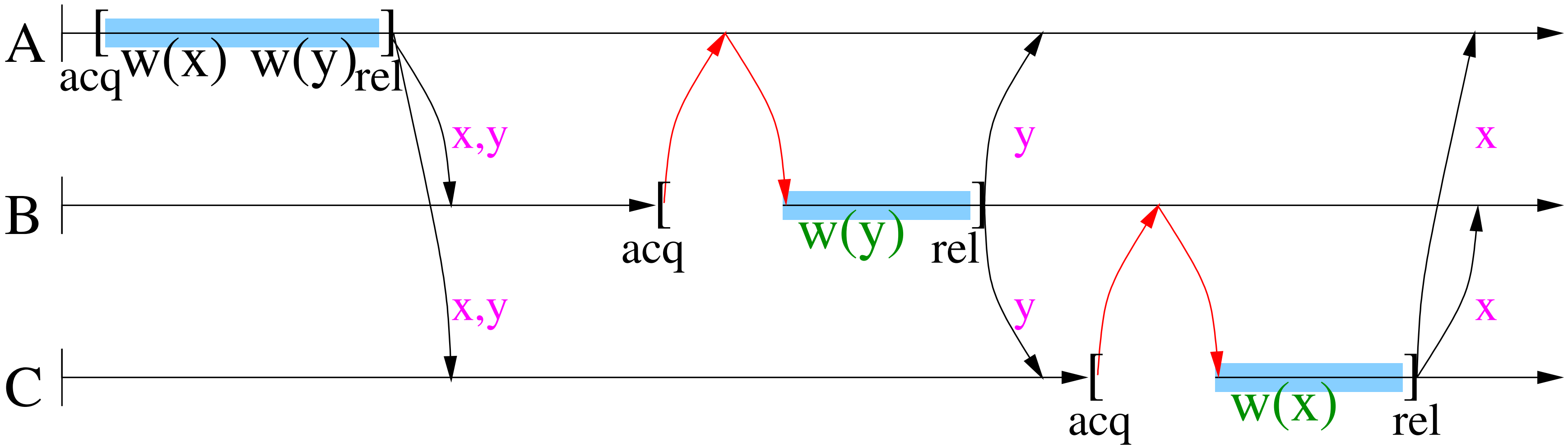
Asynchronous MRSW Consistency



Eager Release Consistency with Update



Eager Release Consistency with Locks

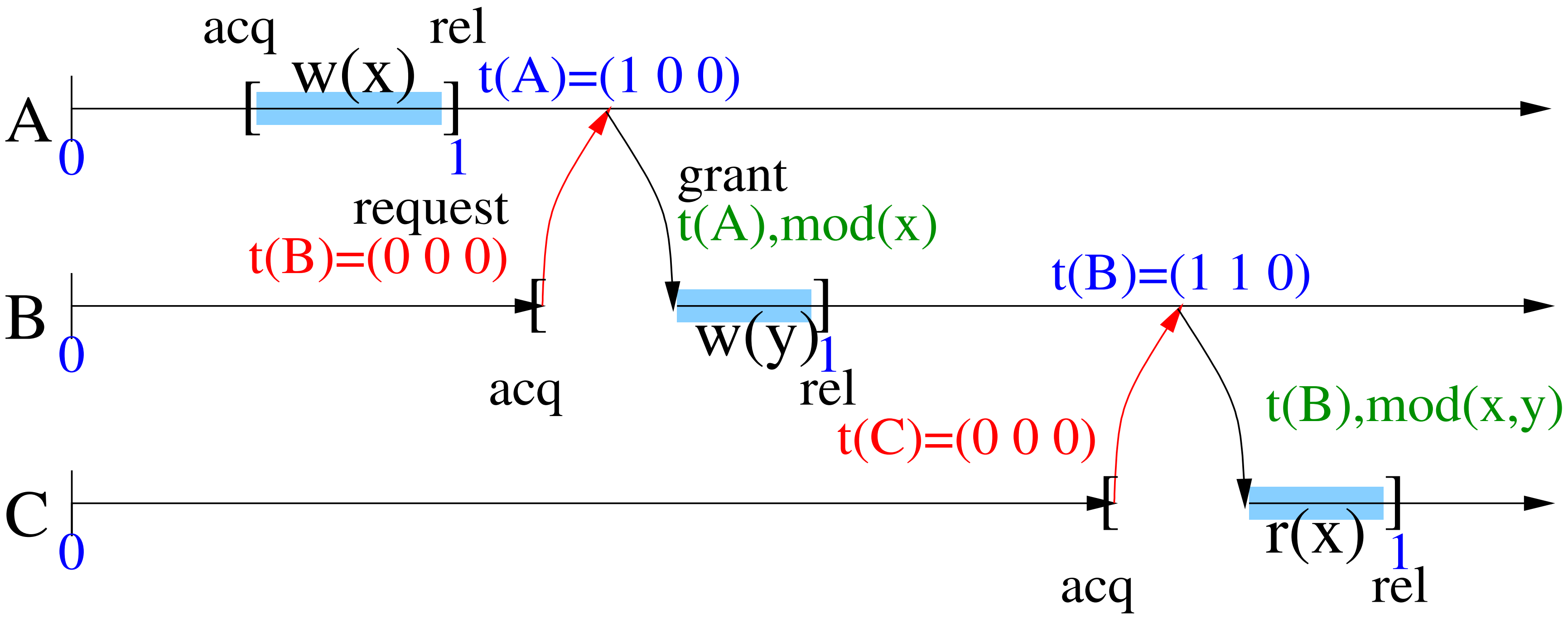


MRMW: Lazy Release Consistency

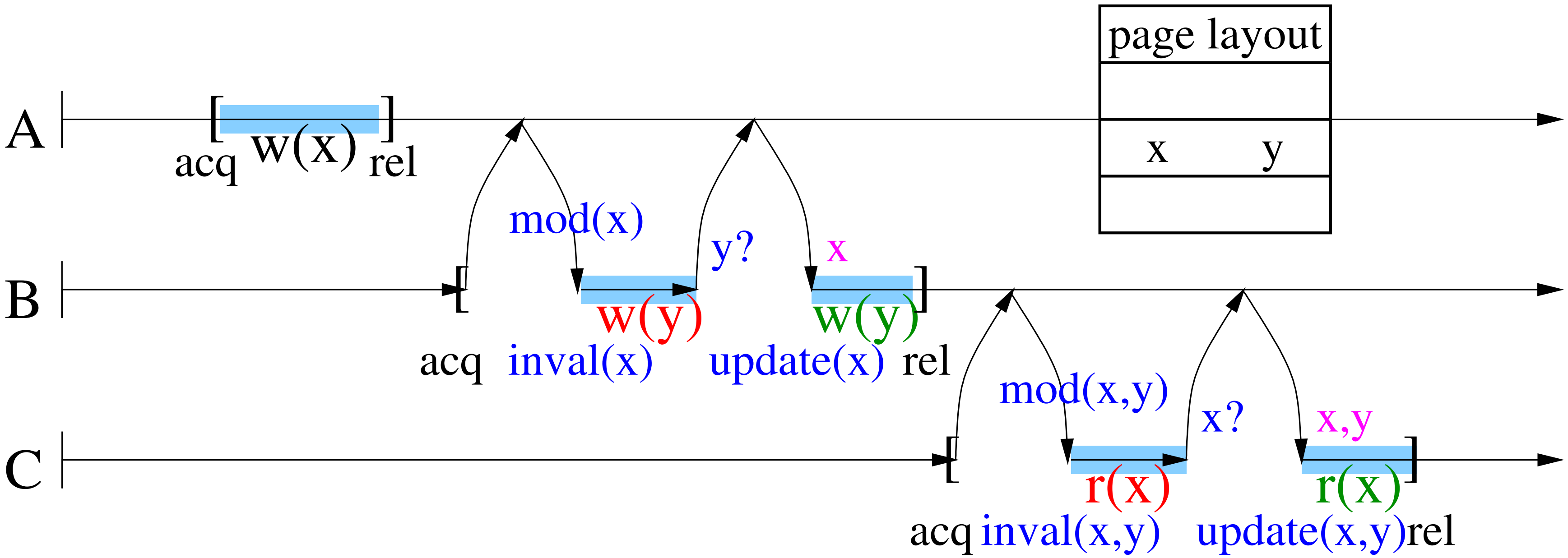
- Lock-Ops sequentially consistent
- R/W weak consistent
- happened-before-1 relation:
  - accesses $a1, a2$ on $p1$ and $a1$ before $a2 \Rightarrow a1 \xrightarrow{hb1} a2$
  - $a1$ release on $p1$ and $a2$ acquire with same access on $p2$ and $a2$ returns values of access in $a1 \Rightarrow a1 \xrightarrow{hb1} a2$
  - $a1 \xrightarrow{hb1} a2 \wedge a2 \xrightarrow{hb1} a3 \Rightarrow a1 \xrightarrow{hb1} a3$
- LRC: at acquire $a2$ update/inval. all accesses $a1$ w/ $a1 \xrightarrow{hb1} a2$
- vector of time stamps (per node)
- page copy at access fault (twin)
- diff at end of interval/upon a request

– nested critical sections, garbage collection

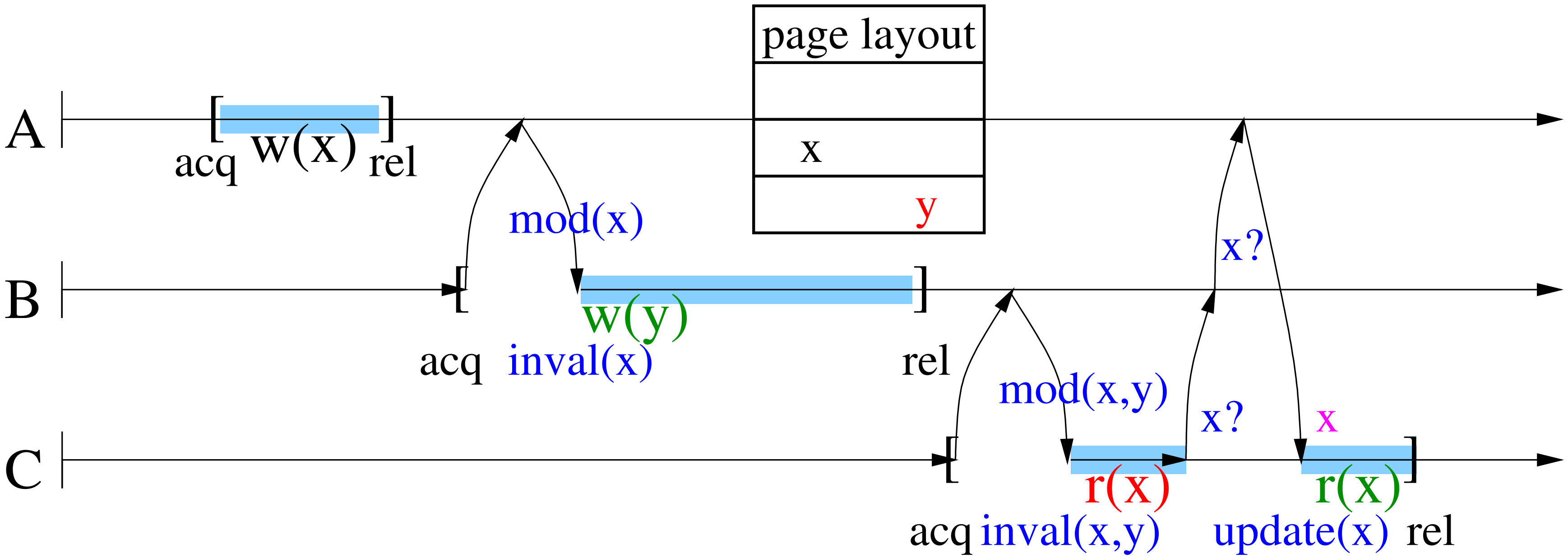
LRC and Timestamp Vectors



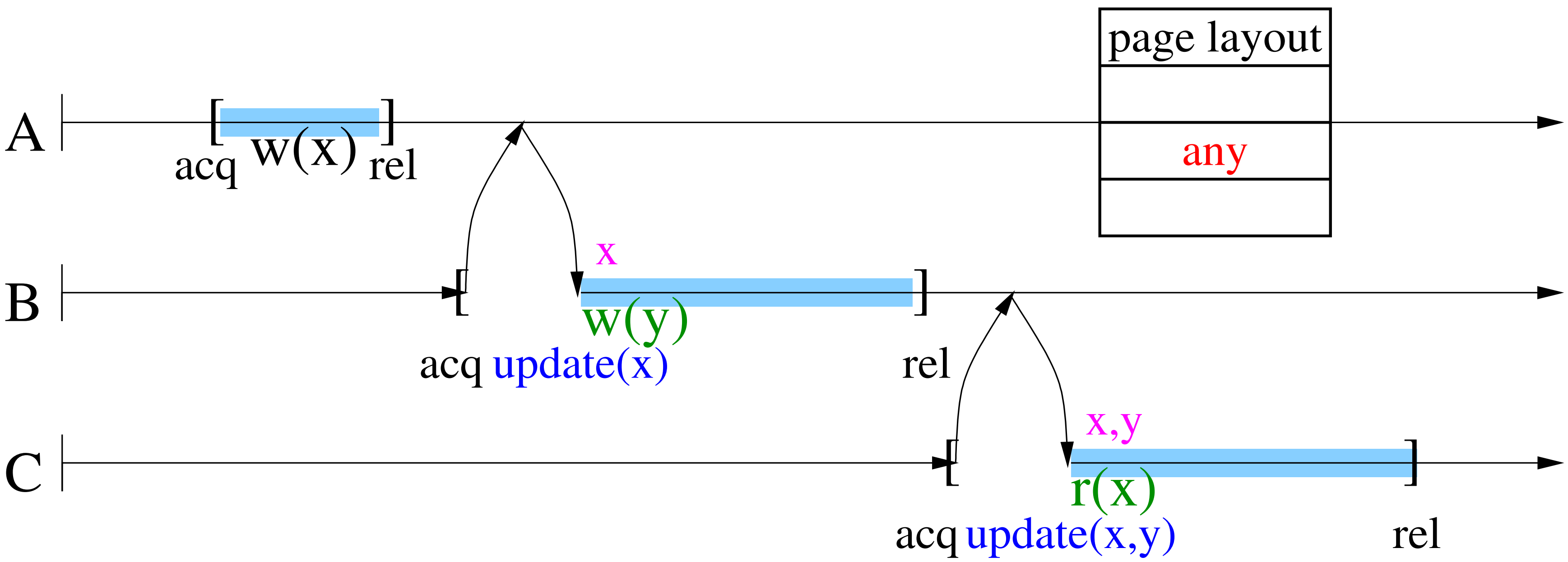
LRC with Invalidate (1)



LRC with Invalidate (2)



LRC with Update



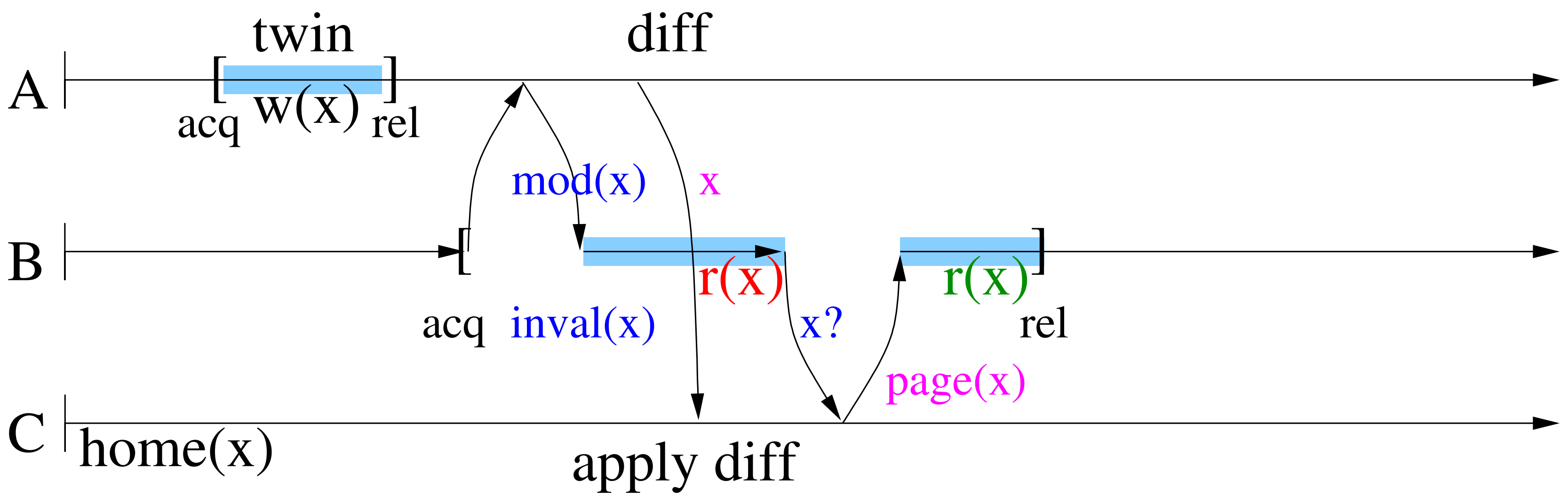
Scope Consistency

- home-based, page granularity, invalidate
- on pagefault: assoc. data (page) with lock
- on acquire: ask token holder
- token holder:
  - send mods of pages for lock
  - send diffs to home
- page fault: fetch page from home

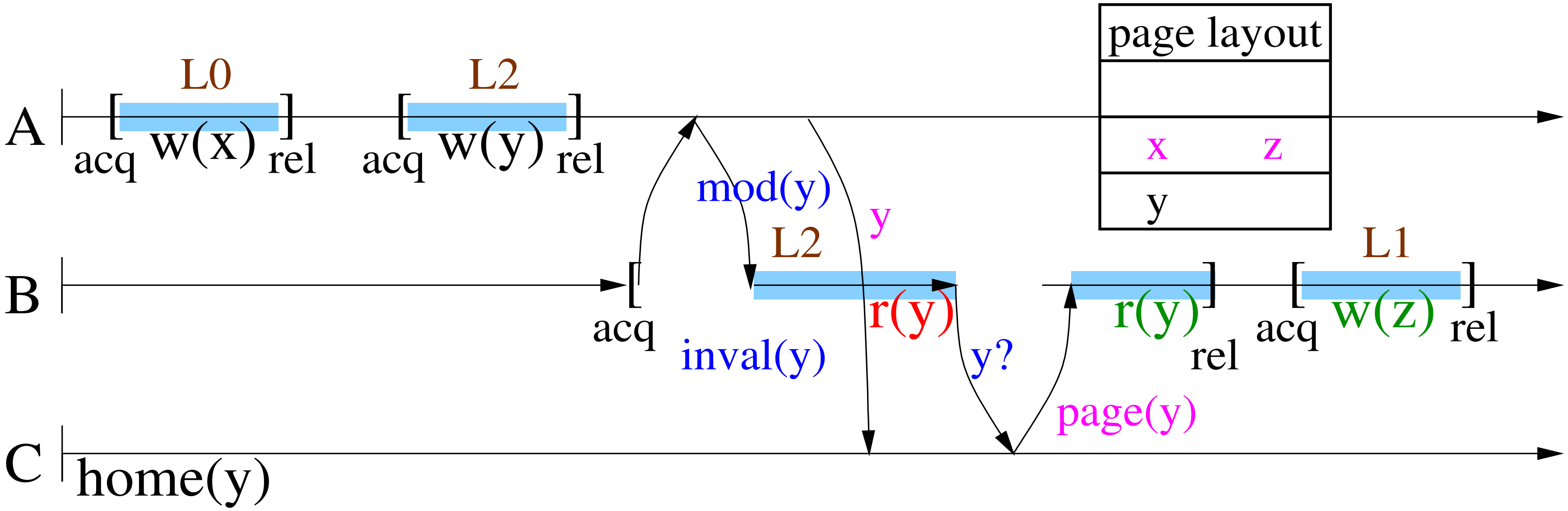
\+ reduces false sharing

– fetch entire pages (LRC: fetch diffs only)

Home-Based SC



SC avoids False Sharing



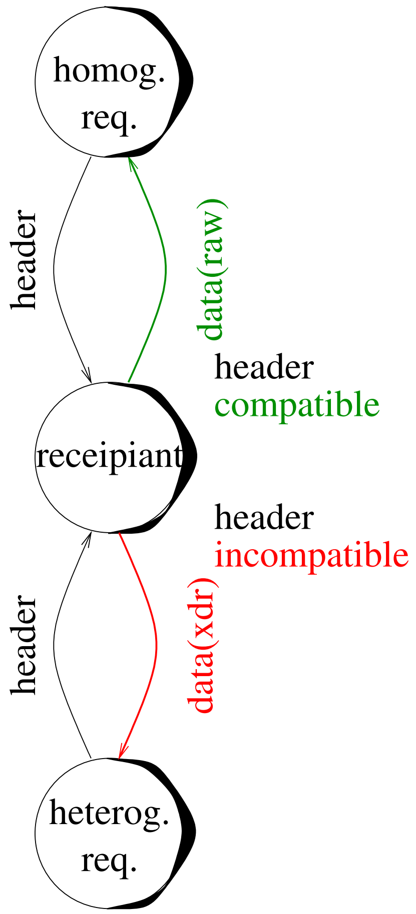
- LRC would have
  - sent mod(x,y)
  - caused fault on w(z)

Heterogeneous Support

- compatibility test via header
- DSM-data encoding through XDR (if required)
- exceptions:
  - same (index,offset) for DSM-data
  - different addresses OK
- alignment problem:
  1. same data, different page

     ⇒ page feed (preprocessor)
  2. data > 1 page

     ⇒ 2 pages contain one ADT
- type identification automatic/explicit
- no pointers yet, later (index,offset)

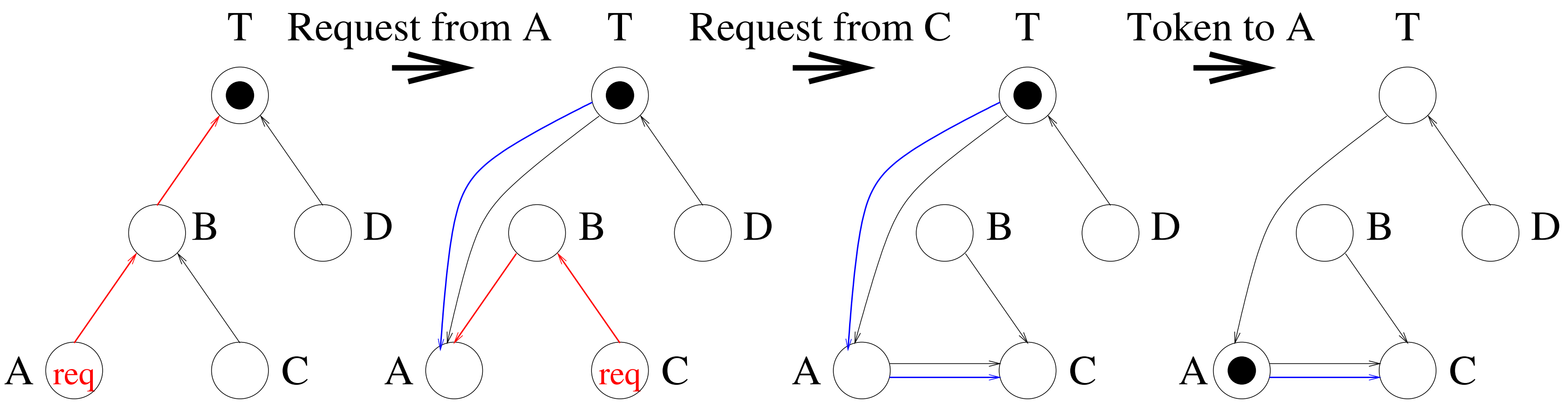


Synchronization

- Mutex
  1. as SRSW: `mutex_lock`
  2. as MRSW: `mutex_read/write_lock`
  3. with priorities
- condition variables as “page”
- barriers
  1. through mutex/cond var (shared variable)
  2. binary tree (decentralized)
  3. logP-tree (optimized)

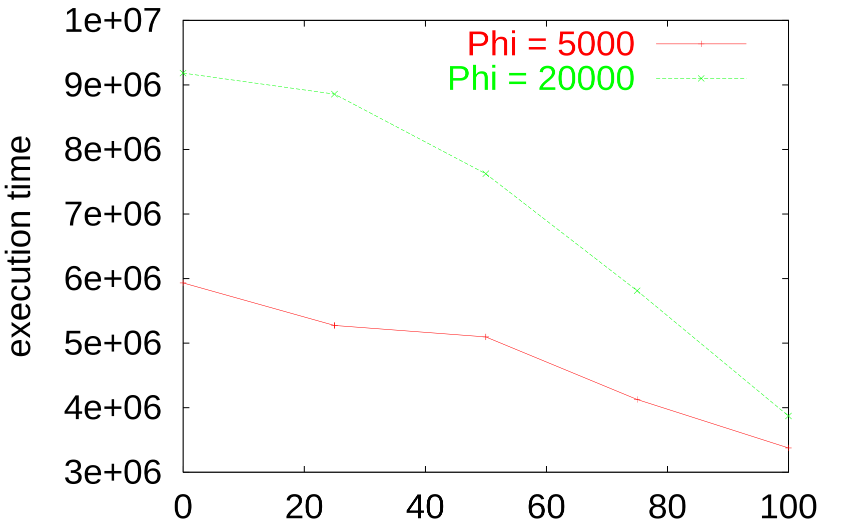
Simple Locks

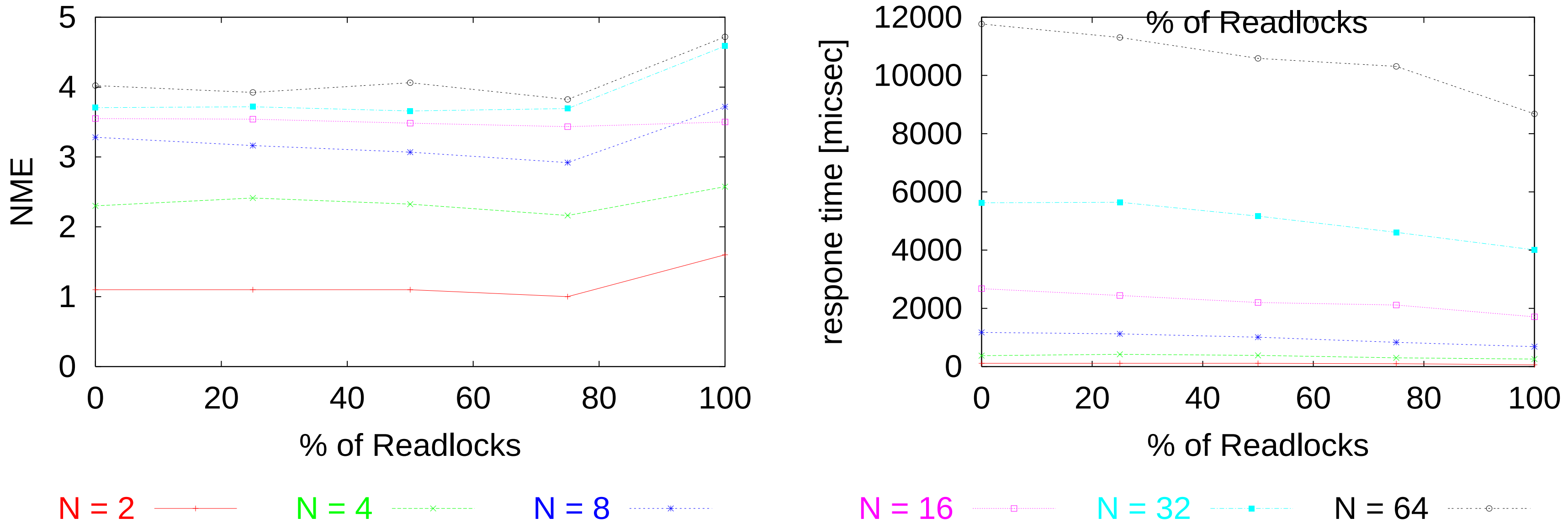
- by Naimi et. al.
- nodes form logical tree, root has mutex (token)
- edges $\Rightarrow$ probable owner (last requester)
- next $\Rightarrow$ next request (distributed queue)
- max. 1 request pro node
- k parallel requests $\Rightarrow$ $k+1$ trees in forest
- request received $\Rightarrow$ 1 tree, root $=$ last requester
- $|msgs|$: avg. $O(log(n))$, max. $O(n)$



MRSW Locks

- extended simple protocol
- allows concurrent reads
- forwards token: R1 → R2 ...
- sends reader done msg:
  R1 → R2 ...





Prioritized Locks

- extend lock protocols
- $|msgs|$: avg. $O(log(n))$, max. $O(n) \Rightarrow$ best protocol
- token-based
- local queues
- sorted by priority
- FIFO within 1 prio (local time approximation)
- asynchronous, out-of-order OK
- 30% longer response times (cf. without prio)
- real-time, Pthreads and time-sharing

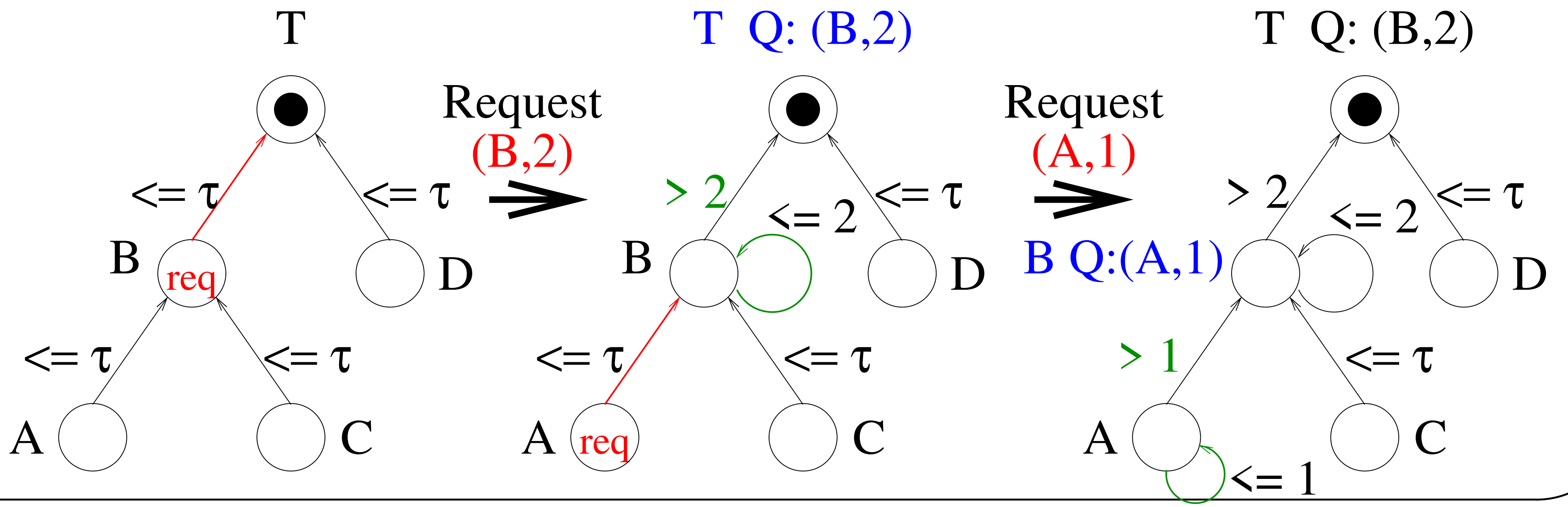
Priorities

- assume: strict priorities, FIFO within 1 prio
- idea: remember priorities upon forwarding (on edges)
- problem: insert in distributed queue (race)

$\Rightarrow$ local queues (prio request $\geq$ prio?)

- problem: FIFO within 1 prio

$\Rightarrow$ local time, approximate remote time *(relative fairness)*

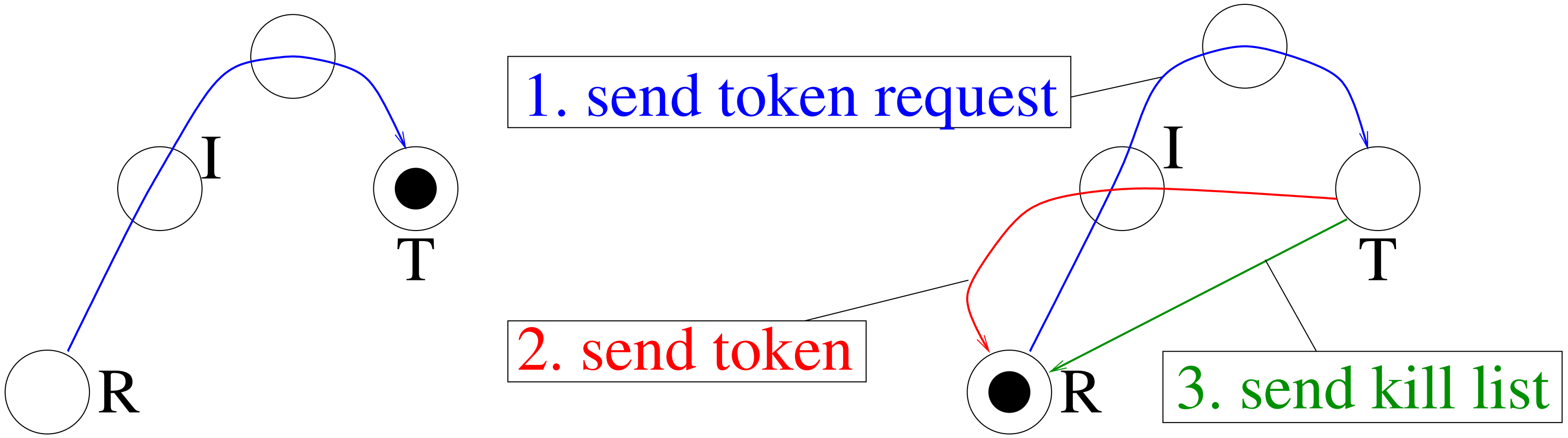


Forward Token

- token available: (Token,Q) $\rightarrow$ Head(Q)
- problem: trailing requests (starvation)

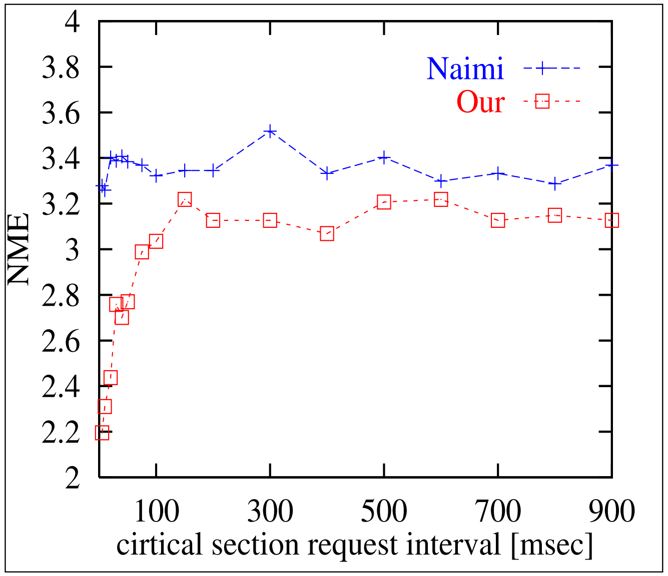
$\Rightarrow$ record forwarding, add kill list

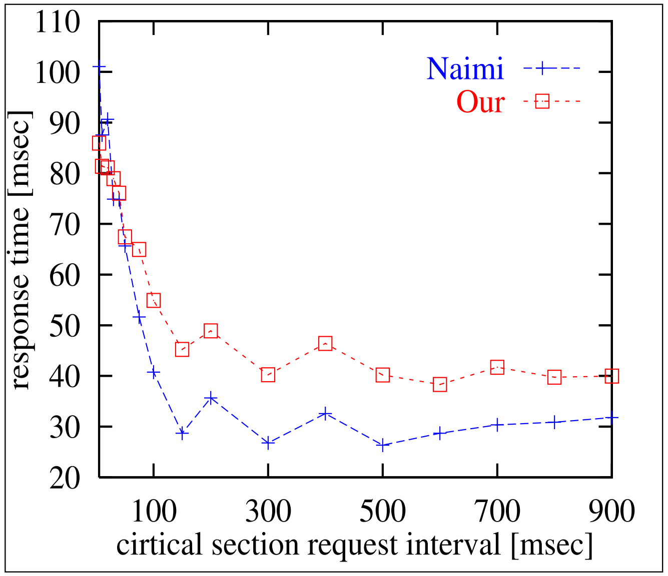
- no forward if target in kill list
- token automatically propagates to requester
- piggyback kill list with token



(a) Request Reaches Token (b) Request Trails Token

Simulated Communication (Shared Memory)

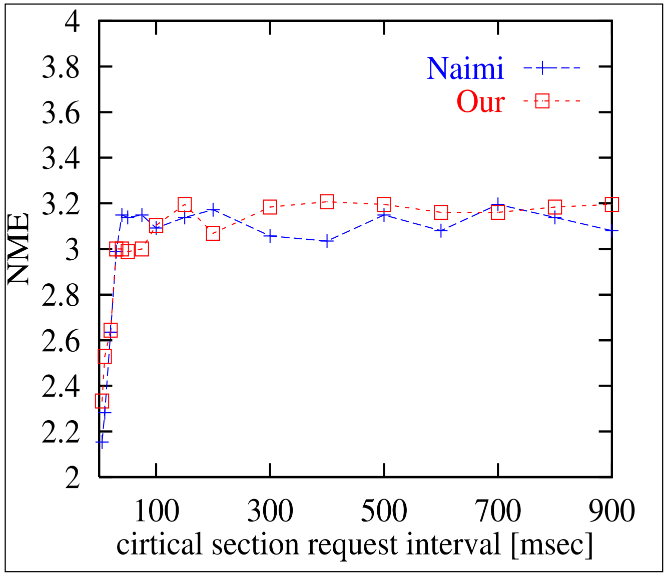


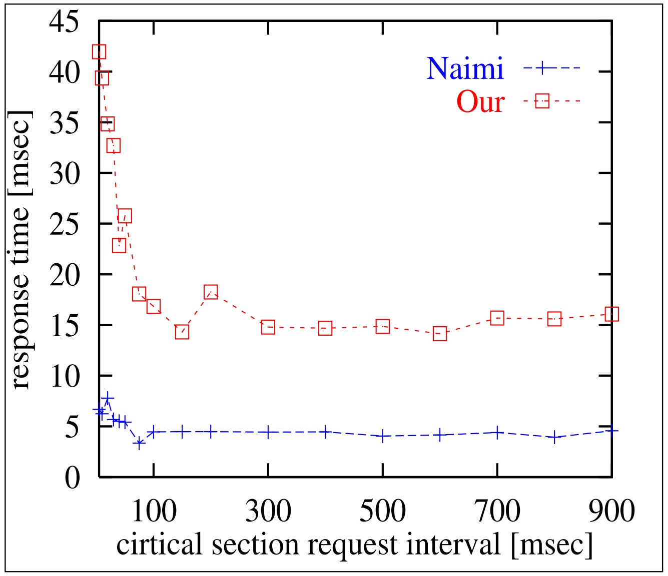


- 3 msgs per lock (fast arrival: only 2)
- t(Naimi) = 1/3 t(ours)

$\Rightarrow$ our protocol overhead (priorities)

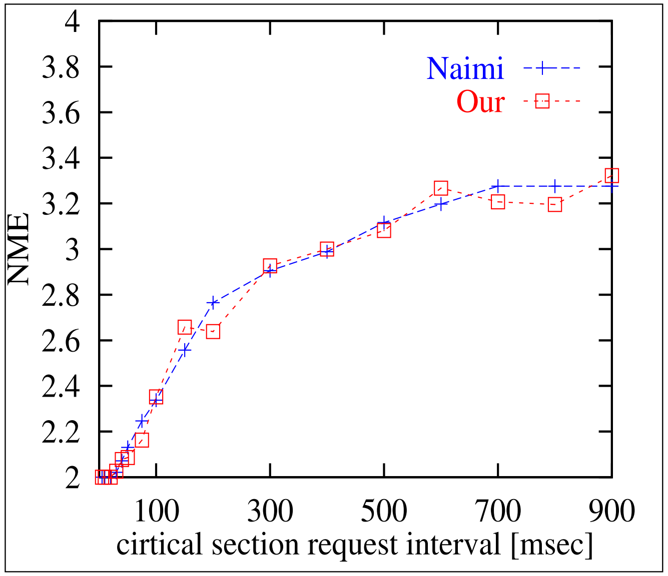
TCP Communication (SMP, 2 Processors)

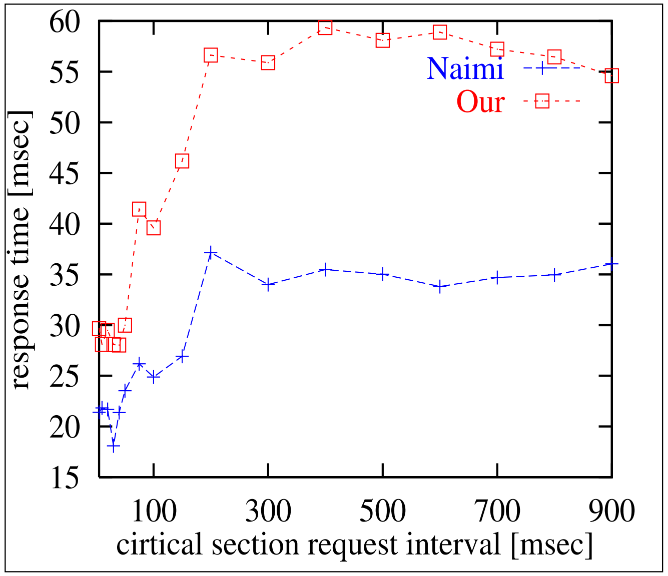




- $|our\ msgs| < |msgs\ Naimi| \Rightarrow$ local Qs
- t(Naimi) = 3/4 t(ours) $\Rightarrow$ fewer msgs
- less competition $\Rightarrow$ shorter response times

TCP Communication (distributed)





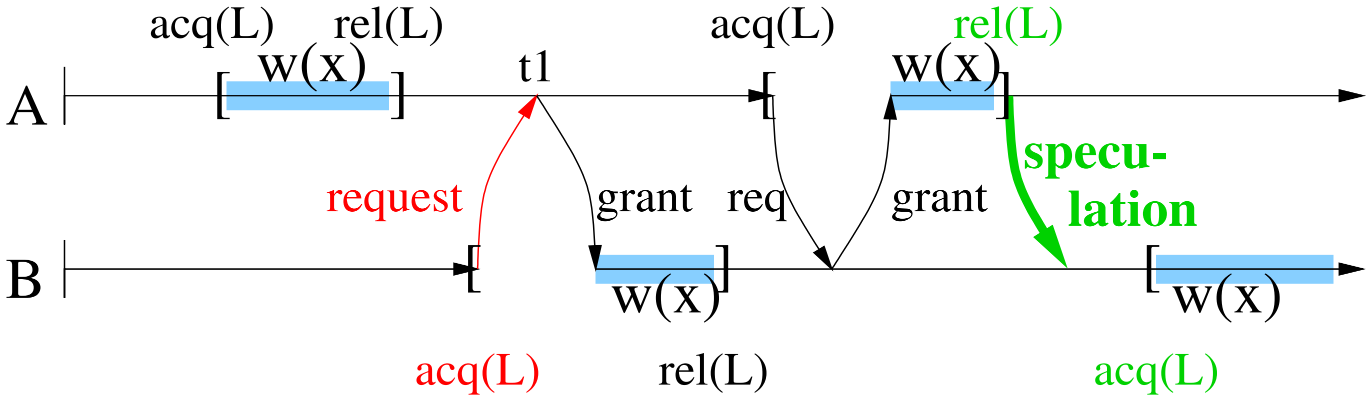
- $|msgs|$ same
- t(Naimi) = 2/3 t(ours) for 3 msgs
- shorter intervals: many local locks (stat. error)

Dynamic Memory: DSM-Heap

- no size limitation
- decentralized allocation `dsm_malloc`
- distributed free list
- protected by distr. mutex (piggyback free list)
- page migration as before
- pointer handling
  - pointers $\rightarrow$ DSM heap only
  - registration
  - translation: normal form (Index, Offset)
  - used for free list
- homogeneous
  - nodes may have different layout
  - but same page indices
- heterogeneous: (Index,Offset), data types explicit

Speculative Locks

- OO: speculative lock protocol
  - receiver recorded upon release
  - forward at next release
  - ⇒ 50% fewer msgs



Speculative Page and Lock Migration

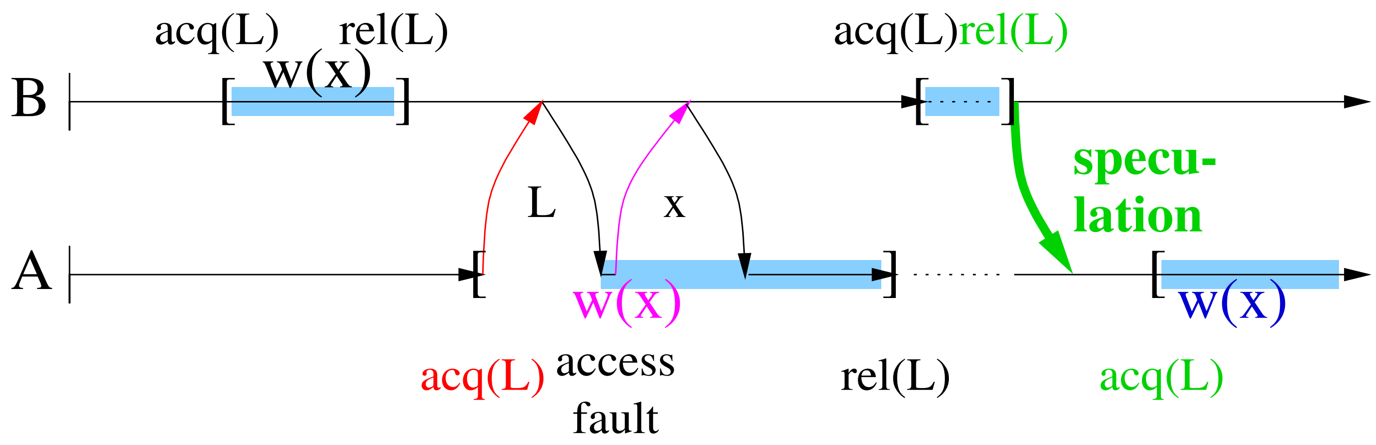
- page request associated w/ last acquire
- page forward at next release

⇒ page and lock speculation

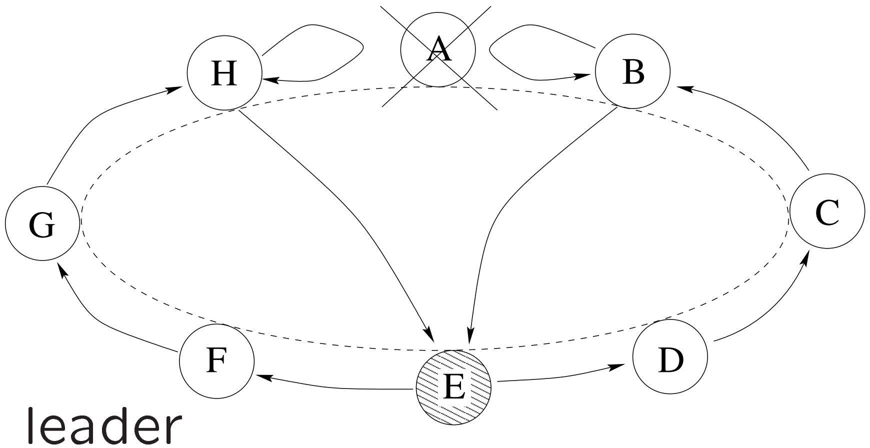
⇒ at best **no** page faults, data/lock requests

- successful speculation acknowledged

⇒ small latencies, fewer msgs



Fault Tolerance



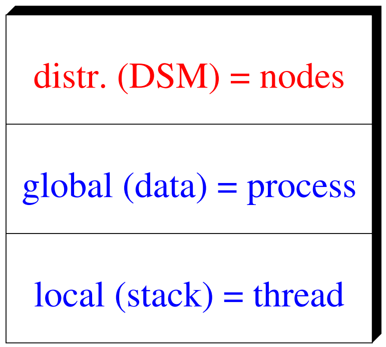
- fault model: single node failure
- detection model: timeout
- recovery model: asynchronous
  - bidirectional ring
  - structural recovery by elected leader
  - leader calculates stable state
  - leader sends state to nodes
  - working nodes continue during recovery
- multiple node failure ⇒ multiple rings

\+ works well for certain protocols (mutex)

– also need checkpointing for roll-back of computation

Implementation

- run-time system (library)
- POSIX.1: SunOS and Linux
- POSIX.1c: FSU Pthreads
- thread distribution implicit or explicit
- C, Ada95, [C++] (Gcc, Gnat)
- Ethernet (TCP + UDP), SCI, Myrinet (Madelaine)



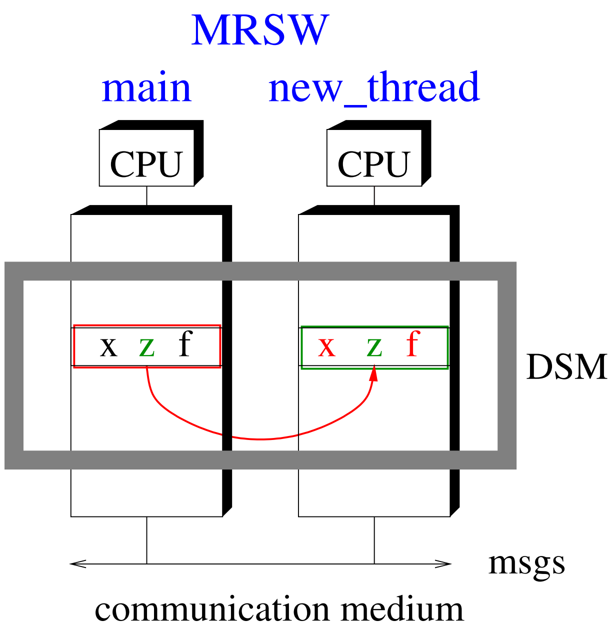
Memory Model

Portability

- POSIX 1003.1 + .1c (Threads)
- Gnu Tools
- remote execution
- page fault for SRSW/MRSW/MRMW
- XDR for heterogeneous
- msg passing

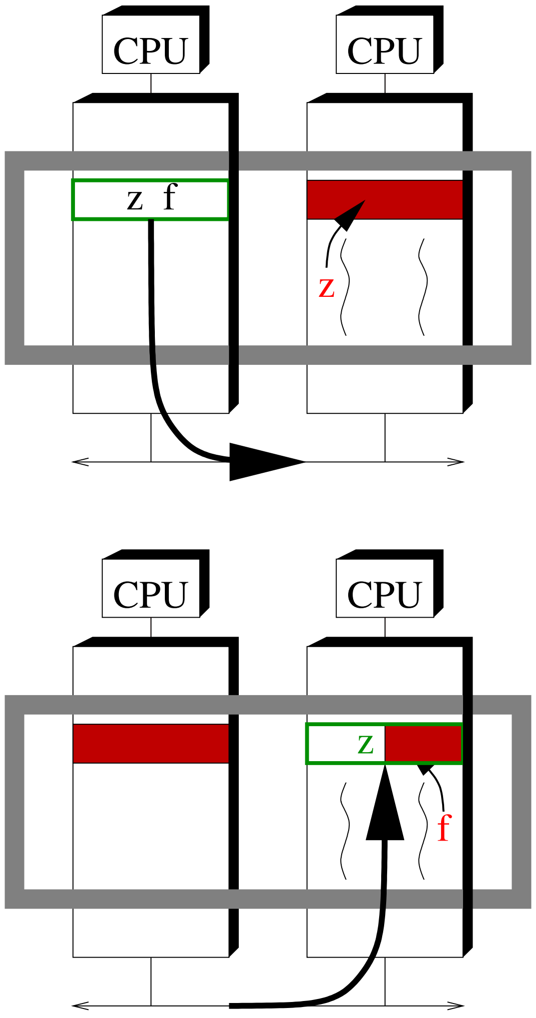
Sample Program

```
#include <dsm.h>

DSM_DATA_BEGIN /* DSM data, shared by all nodes */
volatile int x = 1;
volatile int z = 2;
volatile float f = 1.0;
DSM_DATA_END

int y = 2; /* global data, separate for each node */

void new_thread (void *arg)
{
  f = (float) ( x + y + z );
}     W          R       R

void main (int argc, char **argv)
{
  dsm_attr_t attr; /* local data */
  dsm_thread_t dsm_thread;
  ...
  dsm_init(argc, argv);
  ...
  dsm_attr_init(&attr);
  dsm_attr_setnode(&attr, node); /* node DSM-thread runs on */
  dsm_thread_create(&dsm_thread, &attr, new_thread, arg);
  z = 4;
    W
...
  dsm_exit(NULL);
}
```



Details

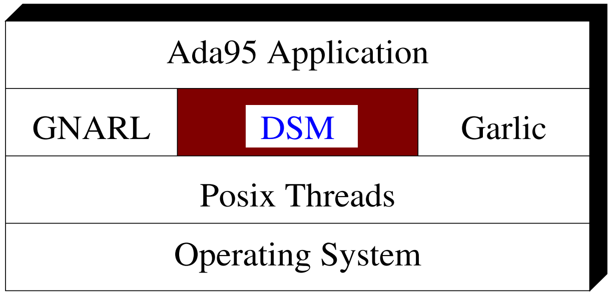
- page protocols:
  - + `mprotect, SEGV`
  - + `mmap` race
  - – R/W fault
- Communication:
  - + asynchronous `IO`
  - + thread- / process-blocking
  - + out-of-order
  - – loss, duplication



Ada95: Passive Partitions

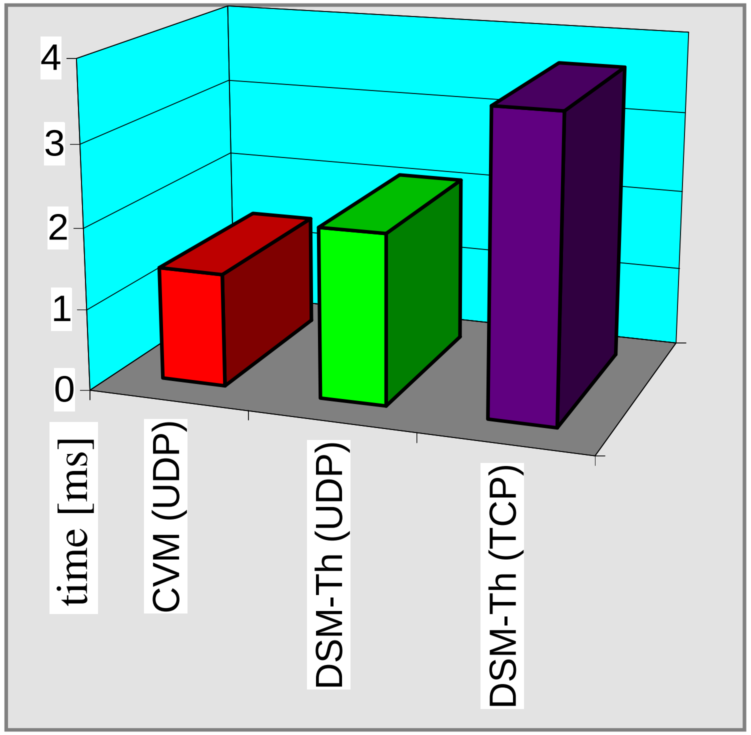
- active part.: distributed tasks (Glade)
- passive part.: distributed data
- DSM-Threads integrated in Gnat
- passive data $\rightarrow$ MRSW
- protected objects $\rightarrow$ OO (entry)

$\Rightarrow$ DSM + distributed synchronization



Preliminary Results

Description	Time
Entire Page Exchange	54 ms
Page Fault and Signal Handling	2 ms
Invalidation	4 ms
Validation	2 ms
2 TCP Messages	46 ms



Future Directions

- lazy release consistency
- scope consistency
- thread migration
- fault tolerance
- logP-barriers
- VIA
- tuning
- benchmarks: Splash, Pthreads applications

Related Work

- Li/Hudak: Ivy
- Berkeley: NOW
- Rice: TreadMarks / UMD: CVM
- Utah: Quarks
- CMU: Midway
- Princeton: Scope
- MIT/Boulder: CRL
- Arizona: Filaments
- Vrije: Orca
- Technion: Millipede

Summary

- DSM-Threads differs from other DSMs
- per-node multi-threading
- asynchronous communication
- out-of-order message passing OK
- responsiveness guaranteed
- API like Pthreads
- choice of memory consistency models
- speculative forwarding
- decentralized synchronization

DSM-Trends

- old applications:
  - parallel programs
  - large data sets
- new applications:
  - Pthreads applications
  - distributed file systems (XFS)
  - databases
  - DNS
  - Web-caching (dsquid)
  - WAN-clusters (Globe, Legion)

`http://www.informatik.hu-berlin.de/~mueller/dsm-threads`