Adaptive DSM-Runtime Behavior via Speculative Data Distribution

Frank Mueller

Institut für Informatik
Humboldt-Universität zu Berlin
10099 Berlin (Germany)

e-mail:
mueller@informatik.hu-berlin.de

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

Overview

- motivation: Why DSM?
- speculation: data + locks
- access patterns
- results
- summary
- related work

DSM: Distributed Shared Memory

- use SMP programming model
- exploit networks of workstations

$\Rightarrow$ simulate shared memory in runtime system

- consistency models:
  - sequential consistency: page granularity, faults
  - entry consistency: objects associated with locks
  - release consistency: weaker model, parallel writes to distinct parts on page

Motivation: DSM Problems

- DSMs supply data on demand (when requested)
- access faults: trigger request

⇒ avoided by forwarding data before reference

- message overhead creates bottleneck
- false sharing causes unnecessary messages

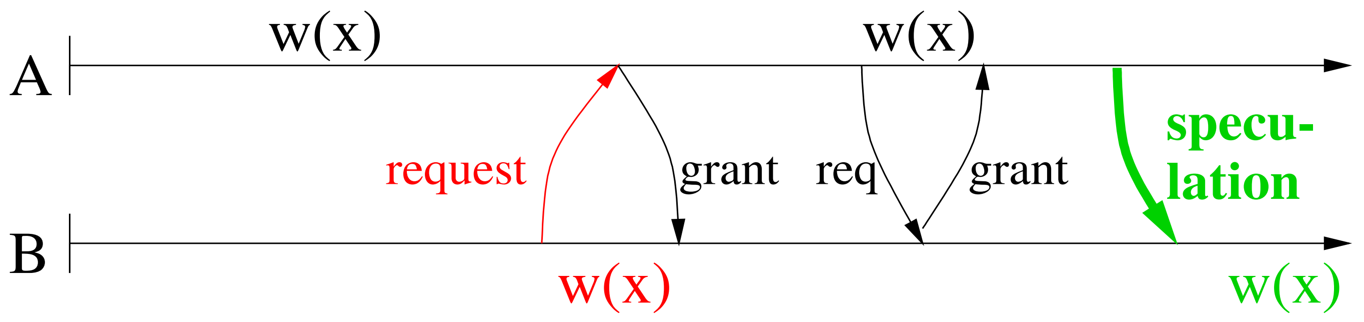
⇒ associate data with locks

⇒ record access patterns of data/locks

⇒ speculatively forward locks, piggyback data

Data Speculation

- record access patterns
- anticipate future requests
- speculatively forward data



Pros and Cons

\+ reduced latency

\+ fewer messages

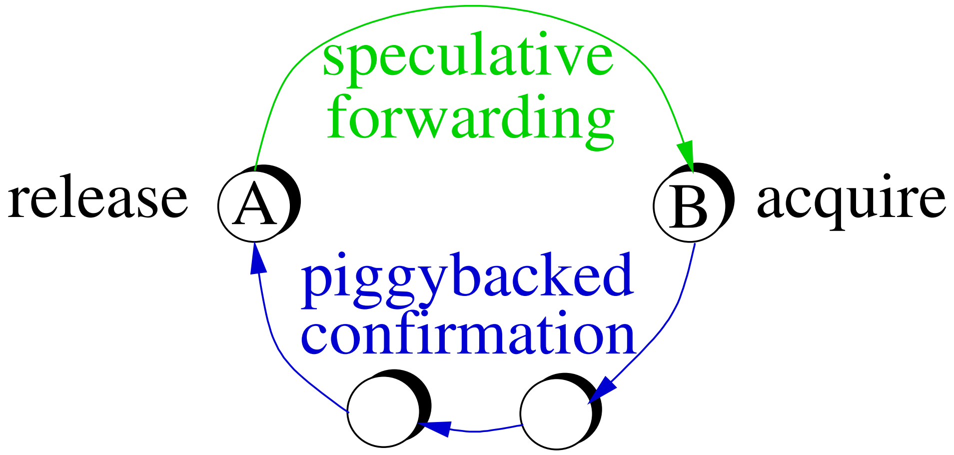
\+ for locks and data

– false speculation

  ⇒ confirmation msg, opt. inhibit speculation

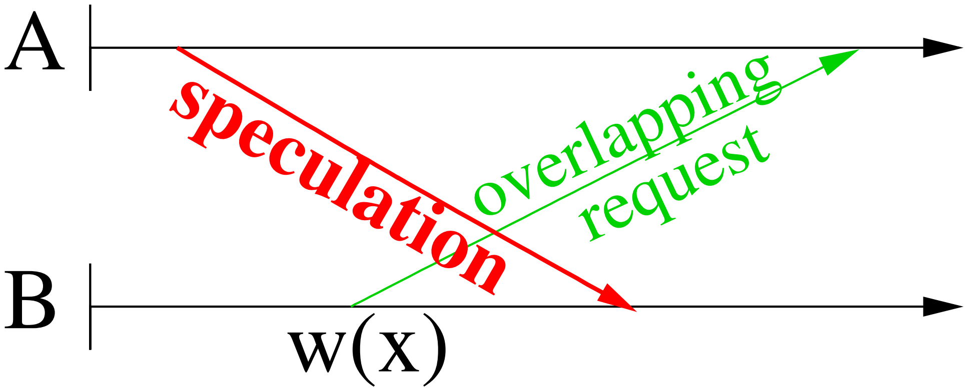
  ⇒ program annotations (control patterns)

– protocol slightly more complex



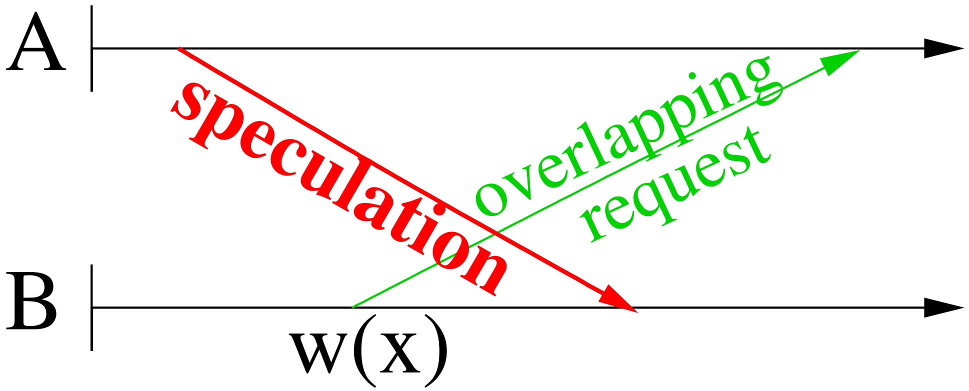
Modifications of Protocols

- race between forwarding and request



⇒ protocol adapted:
- request after forward: silent consumption
- receive w/o request: update
- precedence: demand-driven before speculation

- race between forwarding and request



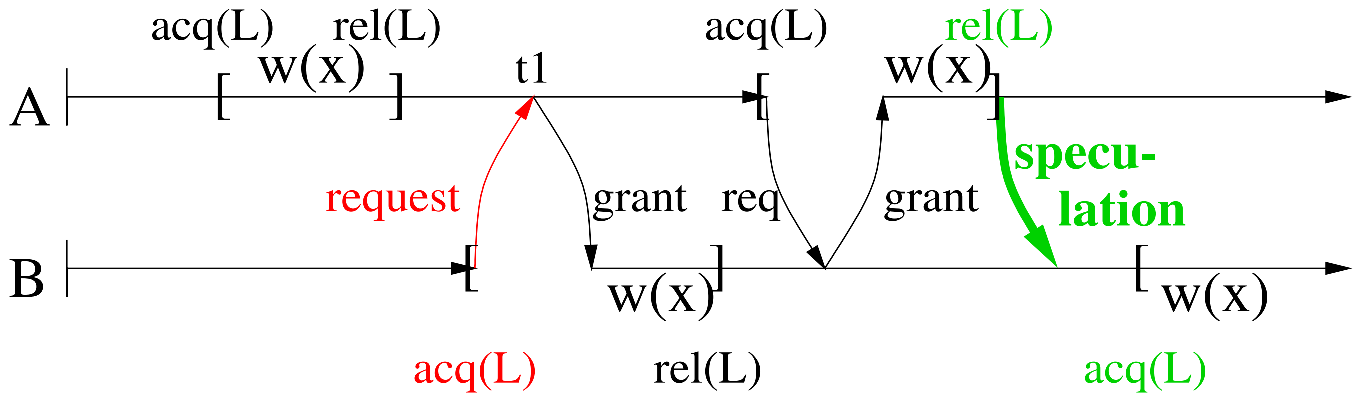
⇒ protocol adapted:

- request after forward: silent consumption ⇒ sometimes covered (dyn. distr. manager, asynchr. token mutex) ⇒ requires in-order msg handling on each node
- receive w/o request: update ⇒ NEW
- precedence: demand-driven before speculation ⇒ reduce false speculation

Entry Consistency

- associate data with lock (explicit)
- piggyback data with locks (implicit)

⇒ speculation:
  - record lock access patterns
  - forward lock upon release



- receive request at owner: record requester
- at next round of locking: forward to recorded requester upon unlock

+ reduced latency
+ reduced messages

- barriers: handled as sequence of release/acquire

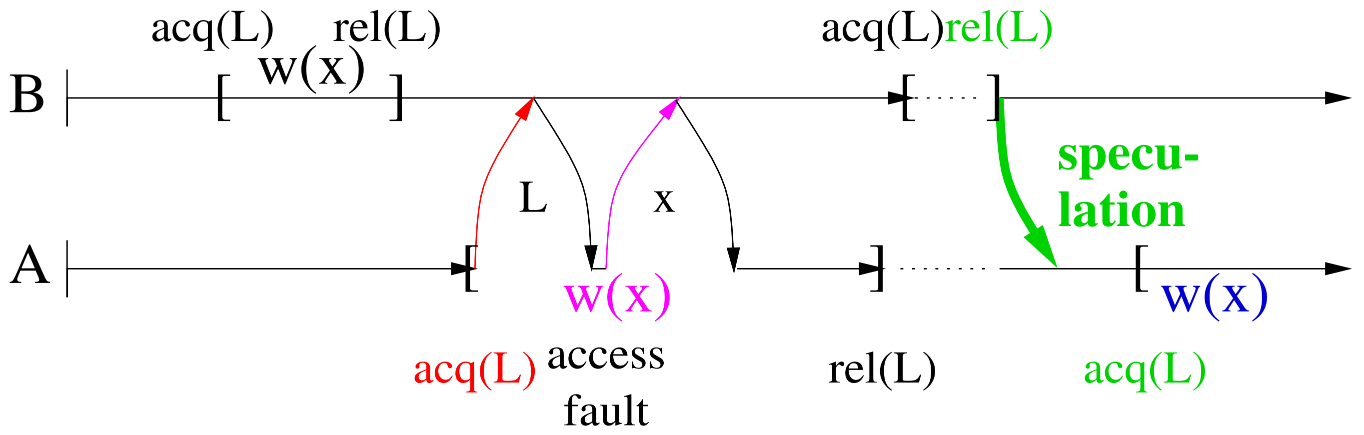
Sequential Consistency

- no explicit data/lock association
- SRSW on page granularity
- synchronization for data consistency

$\Rightarrow$ speculation:
  - for synchronization ops (locks/barriers)
  - for data by implicit lock association

Implicit Data Association

- page fault on A:
  - associate: page $\leftrightarrow$ last lock
  - request page, piggyback association
- next release on B:
  - speculative forward of lock and data (piggybacked)



$\Rightarrow$ speculation at 2 levels: locks + data
- reduced lock requests
- no page faults, no data requests

$\Rightarrow$ lower latency, fewer messages

- compare lock/data destinations
- if equal, reduced chance of false speculation
- if different, inhibit protocol

- cannot distinguish R/W page faults (in a portable way)

$\Rightarrow$ avoid fault for R-access, then fault for W-access + msgs
$\Rightarrow$ 2 faults + save 4 msgs for a W-access

– false speculation hard to detect
  - could detect at page faults
  - better to avoid page faults (no detection)

Association Protocol

Node A

```
acquire(lock object):
  log as last lock object;
```

```
page fault:
  log prefetch info by associating
    fault address with last lock object;
  send page request to B
    with prefetch info;
```

```
release(lock object):
  delete lock object from log;
```

Node B

```
receive page request from A:
  log received prefetch info
    in two-level history;
  send page to A;
```

```
acquire(lock object):
  log last lock object;
```

```
release(lock object):
  speculatively forward lock object
    to node A according to prefetch info;
```

Simple Lock

Node A

```
acquire(lock);
X = 1;                lock <-> X
release(lock);
```

Node B

```
acquire(lock);


A = X;
release(lock);
```

Nested Locks

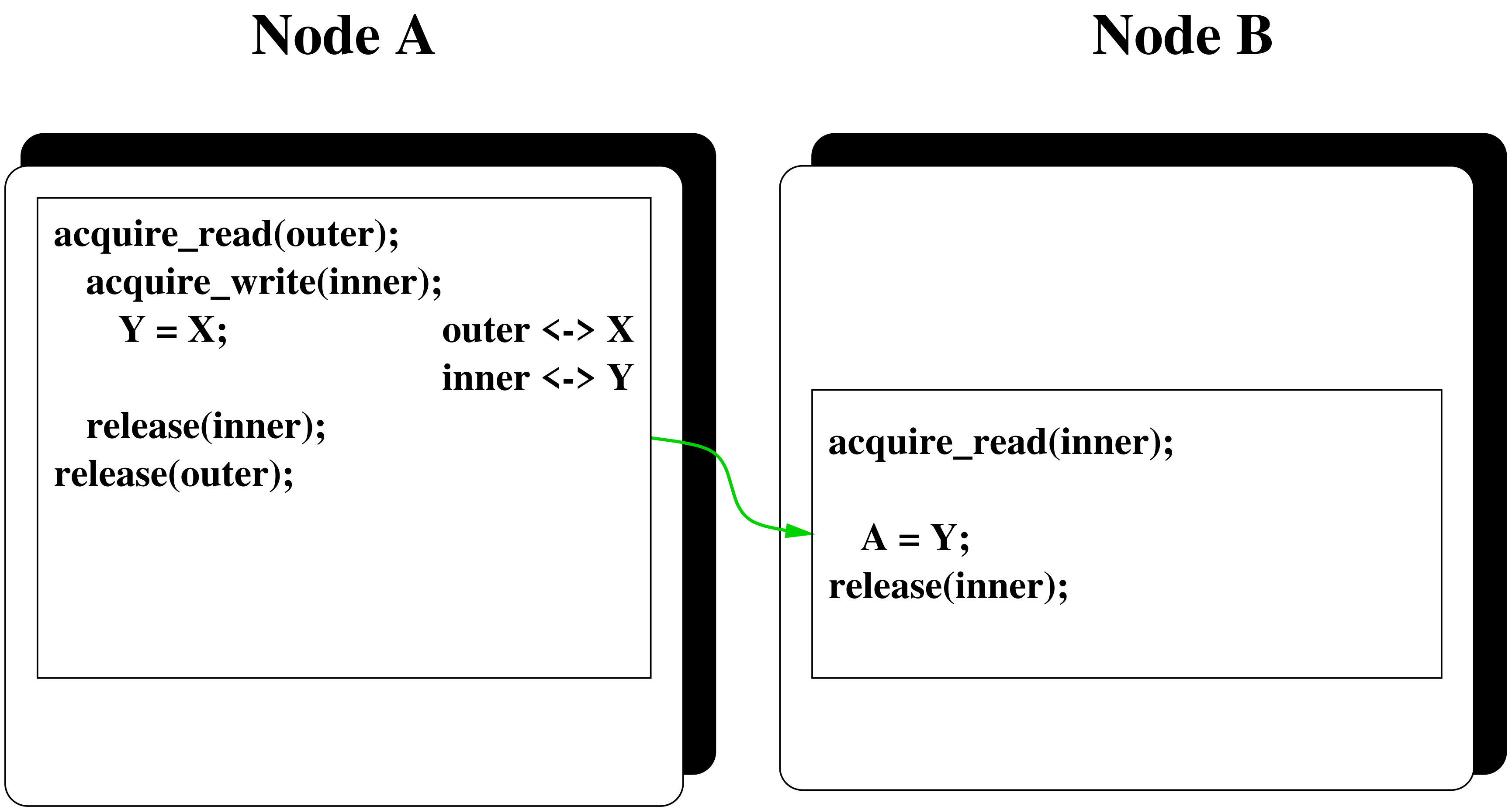
Node A

```
acquire(outer);
  X = 1;                    outer <-> X
  acquire(inner);
    Y = 2;                  inner <-> Y
  release(inner);
release(outer);
```

Node B

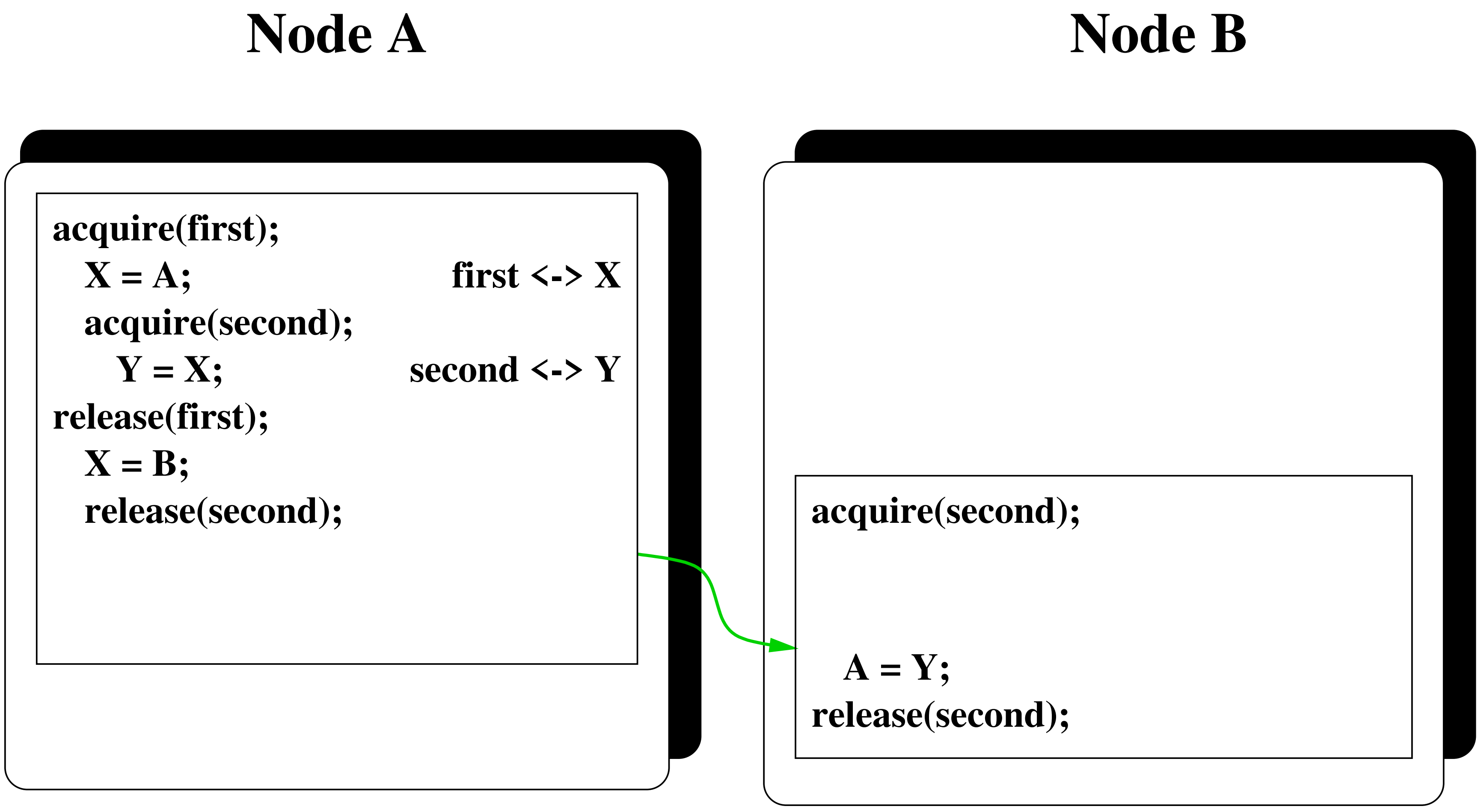
```
acquire(inner);

  A = Y;
release(inner);
```

Read/Write Locks



- separate logging: read locks + write locks

Overlapping Locks



- lock queues

Release Consistency

- eager: broadcast diff at release
- lazy: compute diffs on demand, send to requester
- hybrid: centralized manager
  - diff at release, send diff with lock grant
  - approximate copy set (who might be interested?), grows
- speculation:
  - add lock speculation
  - allow data migration:
    - approximate copy set at root of invalidation
    - page and copy set piggybacked with lock
    - remove from copy set if false speculation

Access Patterns

- regular access patterns:
  - temporal locality (access miss at lock)

  $\Rightarrow$ histories of depth 1
  - avoid prefetch for irregular patterns

  $\Rightarrow$ histories of depth 2 (wait for 1st repetition)

Test Environment

- DSM-Threads
- user-interface: POSIX Threads
- threads distributed on nodes
- shared memory through DSM
- each node multi-threaded
- asynchronous message passing
- locks: token-based distributed mutual exclusion
- seq. consistency: async. extension of Li/Hudak
- entry consistency: like Bershad (but Token-locks)

Results

- ping-pong test
- 10 rounds of acquire, R+W access, release on 2 nodes

Speculative Locking			Speculative Data Forwarding		
Speculation	No	Yes	Speculation	No	Yes
Messages	38	**22**	Messages	76	**8**
Local Locks	1	**18**	Page Faults	18	**4**

- larger applications (Splash-2)
- problems: jitter

⇒ seemingly irregular access patterns

- left part: lock protocol w + w/o speculation
- local locks: lock already forwarded
- 50% fewer msgs

- right part: data messages only
- w/ and w/o speculative forwarding of data at releases
- page faults: may be avoided by data forwarding
- only start-up cost of $4n$ msgs for n nodes
- no data msgs afterwards (all piggybacked w/ lock)

Complex Access Patterns

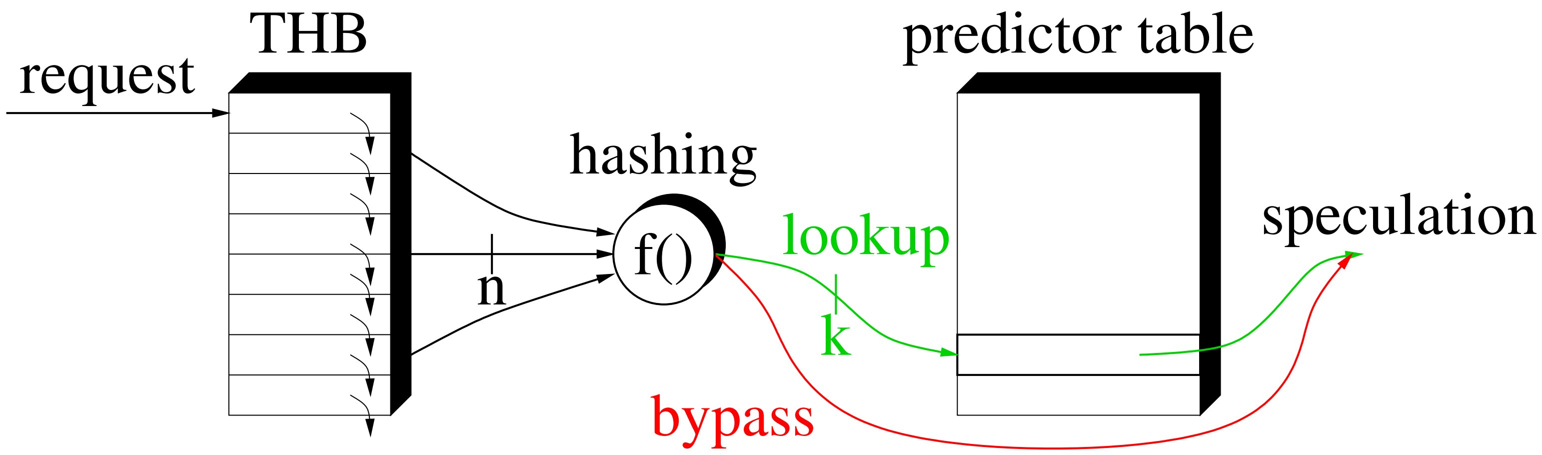
- similarity to branch prediction
- two-level histories
- query options: page/lock address or **fault address**
- target history buffer: composes access pattern
- predictor table: what should be forwarded where?

- similarity to branch prediction
  - DSM: record addresses of pages/locks
  - branches: taken or not

  ⇒ closer to indirect jumps
- two-level histories
  - THB: target history buffer (global or for each page/lock)
  - PT: predictor table w/ 2^k patterns
  - PT indexed by hashed value
  - PT entry: prefetch set (not really prefetching but forwarding)

  ⇒ access patterns piggybacked w/ page/lock to predictor node

Two-Level Histories

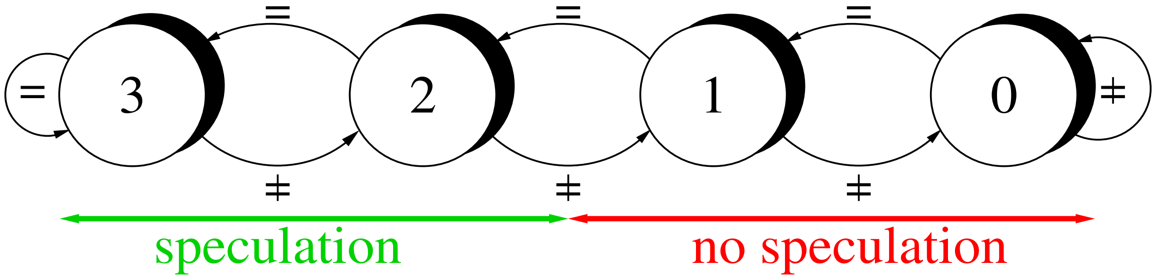


- spatial locality: prefetching
  - column indexing (small strides): hash high bits
  - row indexing (large strides): bypass, fetch next page

Irregular Access Patterns

- delay: two correct speculations before active
- two-bit saturating counters in PT

⇒ limits false speculation



Related Work

- hybrid LRC with heuristics (Keleher)
- scope consistency (Iftode/Li)
- affinity entry consistency (Bianchini)
- prefetching with page faults (Stenstroem, Bianchini)
- dynamic access histories (Schuster: migration)
- compiler-directed access patterns (HPF, Loewenthal: filaments)
- branch prediction (Smith, Stark/Patt, ...)

Summary

- methods for speculative data distribution
  - entry consistency: forward locks
  - sequential consistency: forward locks + data
  - release consistency: forward locks + data, check success
- benefits:
  - fewer lock messages (down to 50%)
  - fewer data messages (down to start-up constant)
  - fewer page faults (down to start-up constant)

  $\Rightarrow$ reduced latencies, even when forward/request overlap
- problems:
  - some changes of protocols
  - false speculation, jitter

  $\Rightarrow$ enhanced speculation:
  - two-level histories
  - confirmation