Shared Memory Systems
Part III

Randal E. Bryant

CS 740
Dec. 3, 1997

Topics

- **Fixing network-based coherency protocol**
- **Adding features**
- **Buffering & resource considerations**
- **Comparison to “real” protocols**
  - SGI Origin

Network-Based Cache Coherency

Home-Based Protocol

- **Each block has "home"**
  - Memory controller tracking its status
  - Based on physical address
- **Home maintains**
  - Block status
  - Identity of copy holders
    - » 1 bit flag / processor
  - Only need entry when block has remote copy

Memory Controller 4

Block	Status	Copy Holders
24	shared	0 1 0 1 0 1 0 1
25	Exclusive	0 1 0 0 0 0 0 0
26	uncached	0 0 0 0 0 0 0 0

Block Status Values

- **Shared**
  - 1 or more remote, read-only copies
- **Exclusive**
  - Single, writeable copy in remote cache
- **Uncached**
  - No remote copies

Implementation Details

- **p Processor identifier**
- **a Block address**
- **d Block data**

Messages from Cache to Block Home

- **Cache requests**

Type	*Contents*	*Purpose*
READ	**p, a**	**Processor p wants to read a**
XREAD	**p, a**	**Processor p wants to write a**

- **Cache replies**

Type	*Contents*	*Purpose*
WRITEBACK	**p, a, d**	**Proc. p flushes exclusive block**
WRITEHOLD	**p, a, d**	**Proc. p shares exclusive block**
RELEASE	**p, a**	**Proc. p flushes clean block**

  – Treat evictions as unsolicited replies

Implementation Details (Cont.)

- p Processor identifier
- a Block address
- d Block data

Messages from Block Home to Cache

- Home requests

Type	*Contents*	*Purpose*
INVALIDATE	p, a	Give up clean copy
FETCH	p, a	Get readable copy from remote
XFETCH	p, a	Get writeable copy from remote

- Home replies

Type	*Contents*	*Purpose*
DATA	p, a, d	Reply to read / write request
NACK	p, a	Cannot fulfill request

Tricky Issues

Independent Actions between Cache & Home

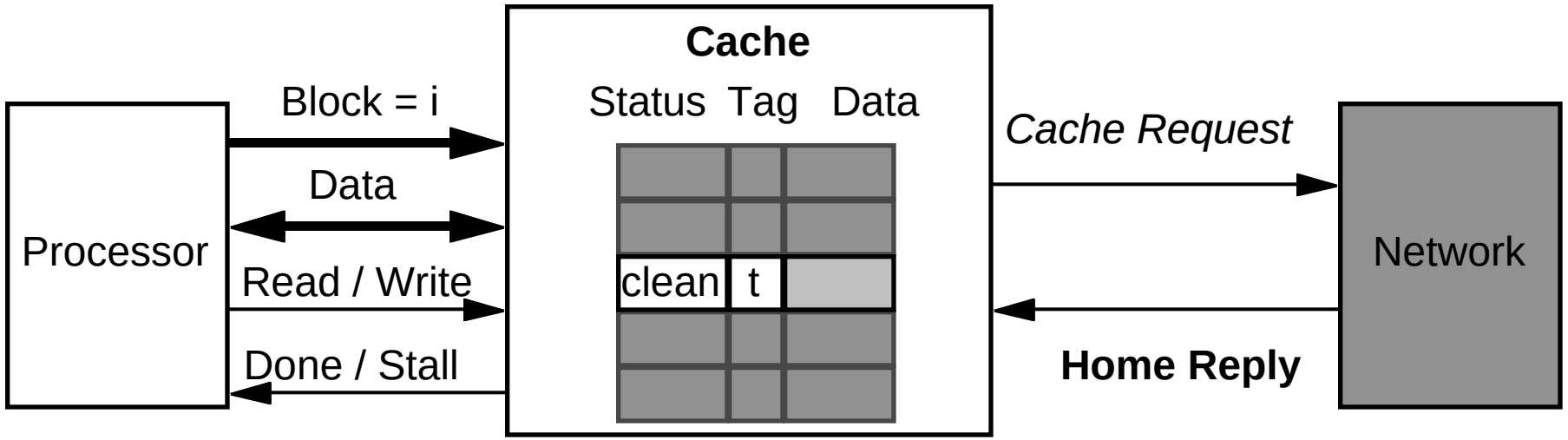
- **Cache evicts exclusive block while home in process of fetching**
  - Home will get *WRITEBACK* when expecting *WRITEHOLD*
  - Cache will receive **FETCH** for invalid block
- **Cache evicts exclusive block while home in process of xfetching**
  - Home will receive *WRITEBACK* generated by cache
  - Cache will receive **XFETCH** for invalid block

Independent Actions between Two Processor Caches

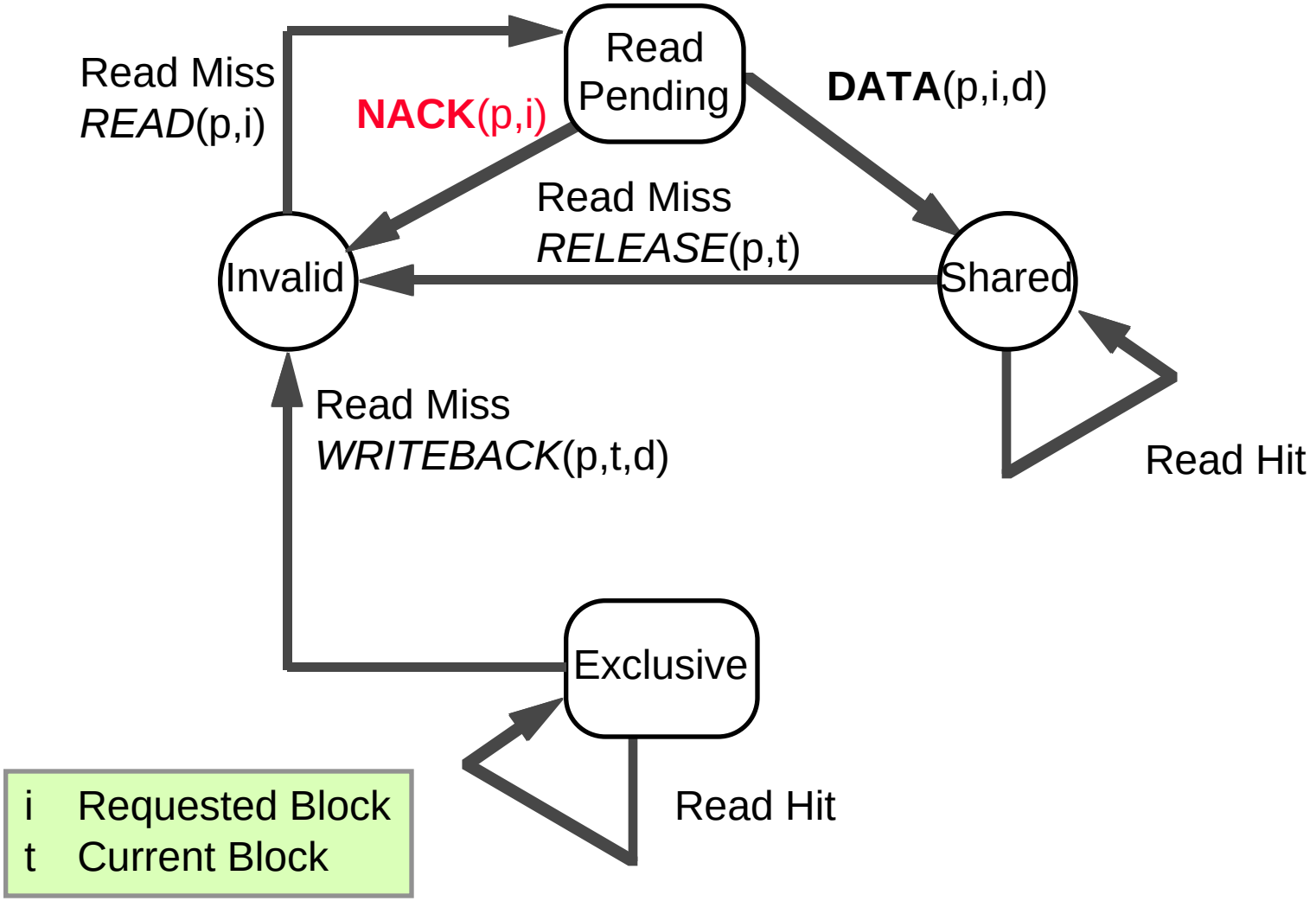
- **Processor r attempts read or write while home handling read or write request by processor p**
  - Home sends **NACK** to processor r
  - Processor r retries read or write operation
- **Potential for livelock**
  - Processor r keeps getting **NACK**'ed

Performing Processor Operations

- **Processor requests cache to perform load or store**
  - On word in cache block i
- **Cache line currently holds block t**
  - May or may not have i = t
- **Cache can either:**
  - Perform operation using local copy
  - Issue request message to network (*italicized*)
    - » Wait for reply (**bold**)
    - » Stall until completed



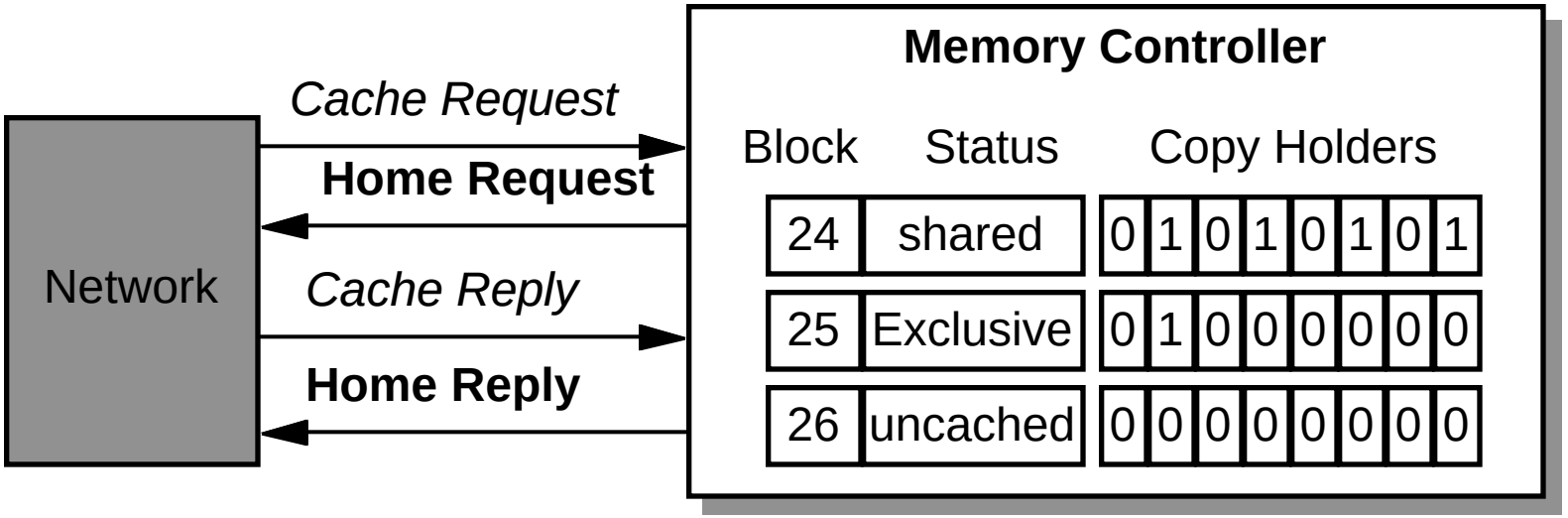
Cache Handling of Processor Read



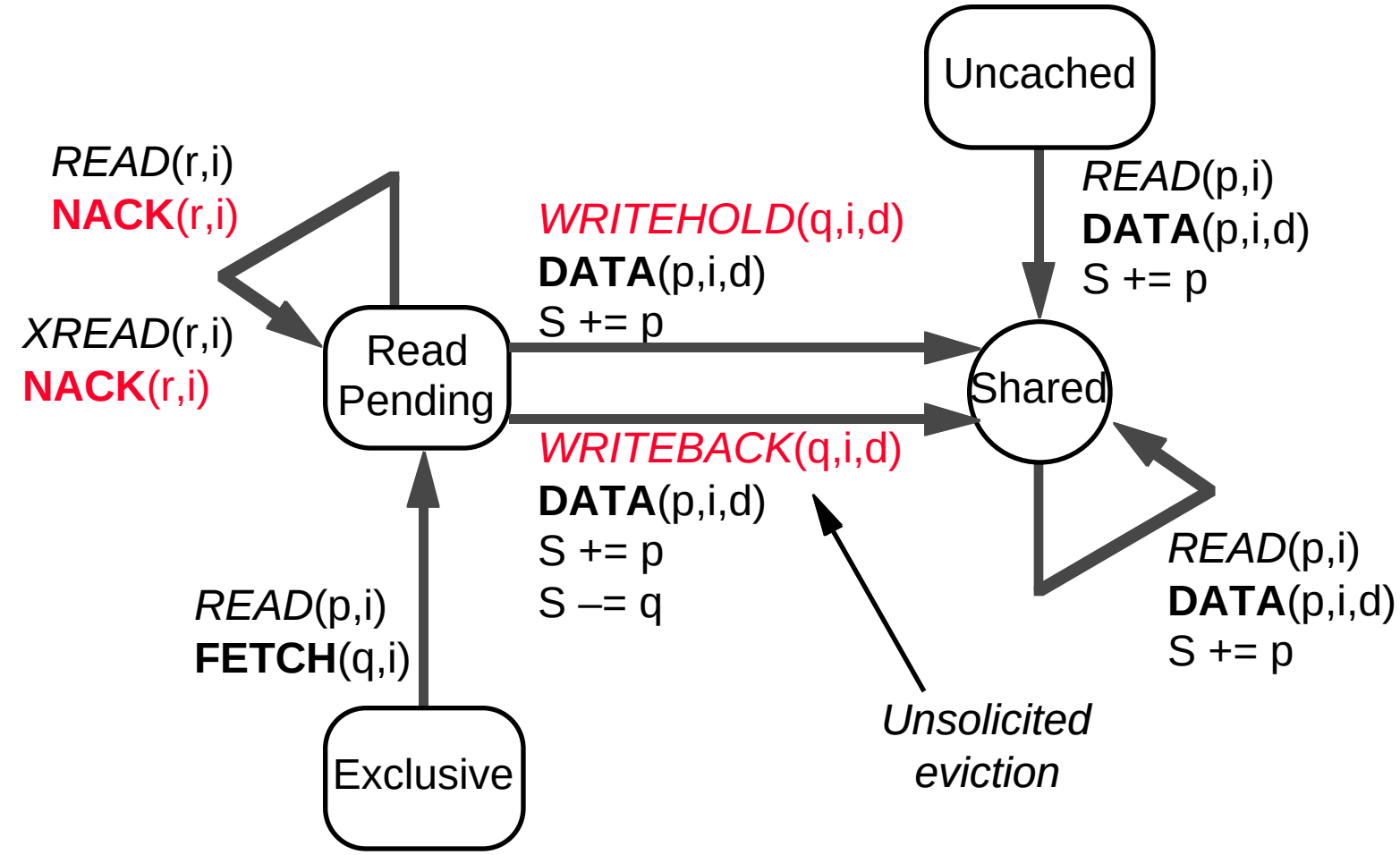
Processing by Home

Home Memory Controller

- **Receives requests from caches**
  - To handle processor reads & writes
  - Unsolicited evictions
- **May need to retrieve or invalidate remote copies**
  - Sends requests to copy holders
  - Copy holders send replies
- **Replies to requestor**
- **Send NACK if unable to satisfy request**



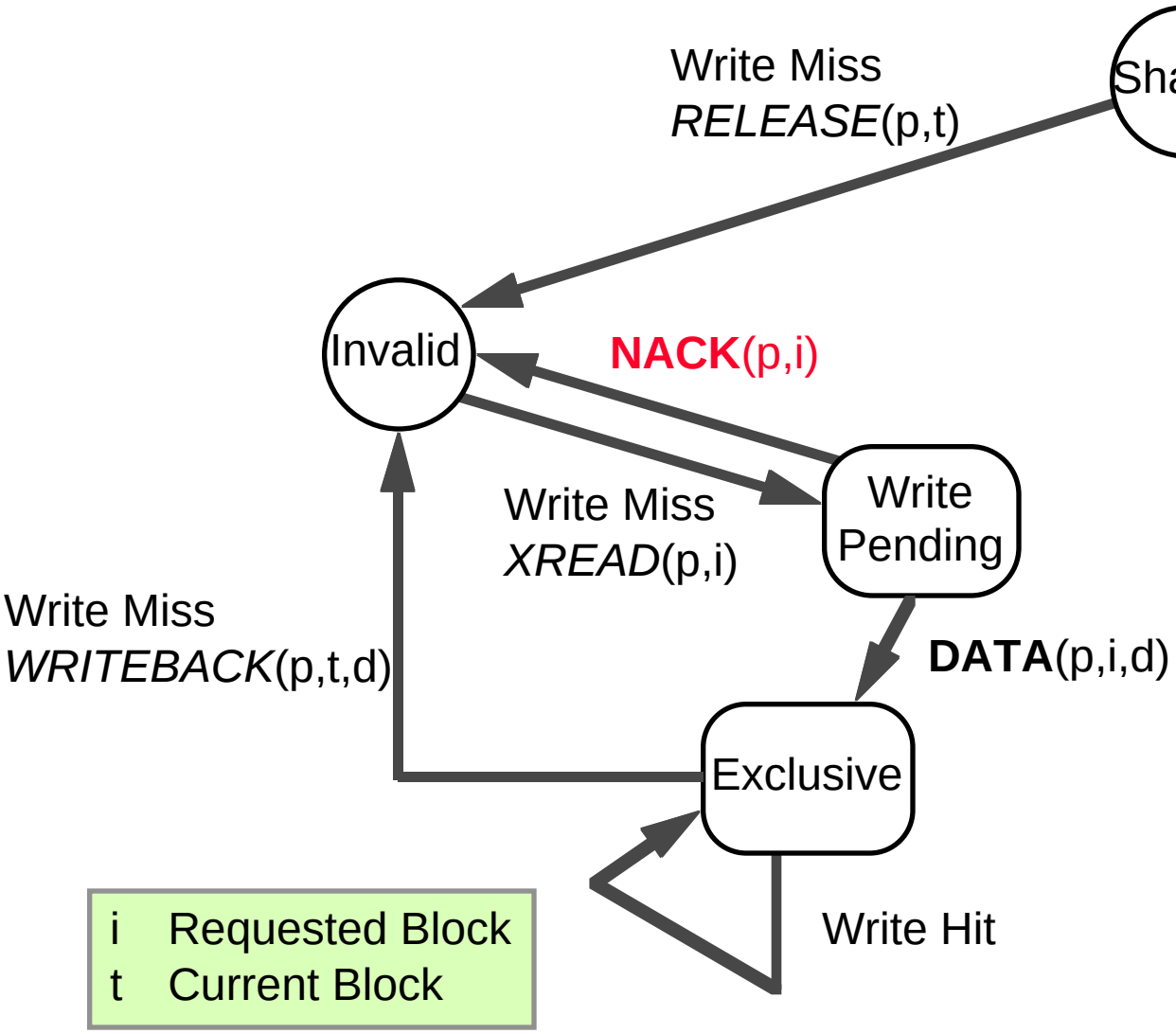
Home Handling of Processor Read



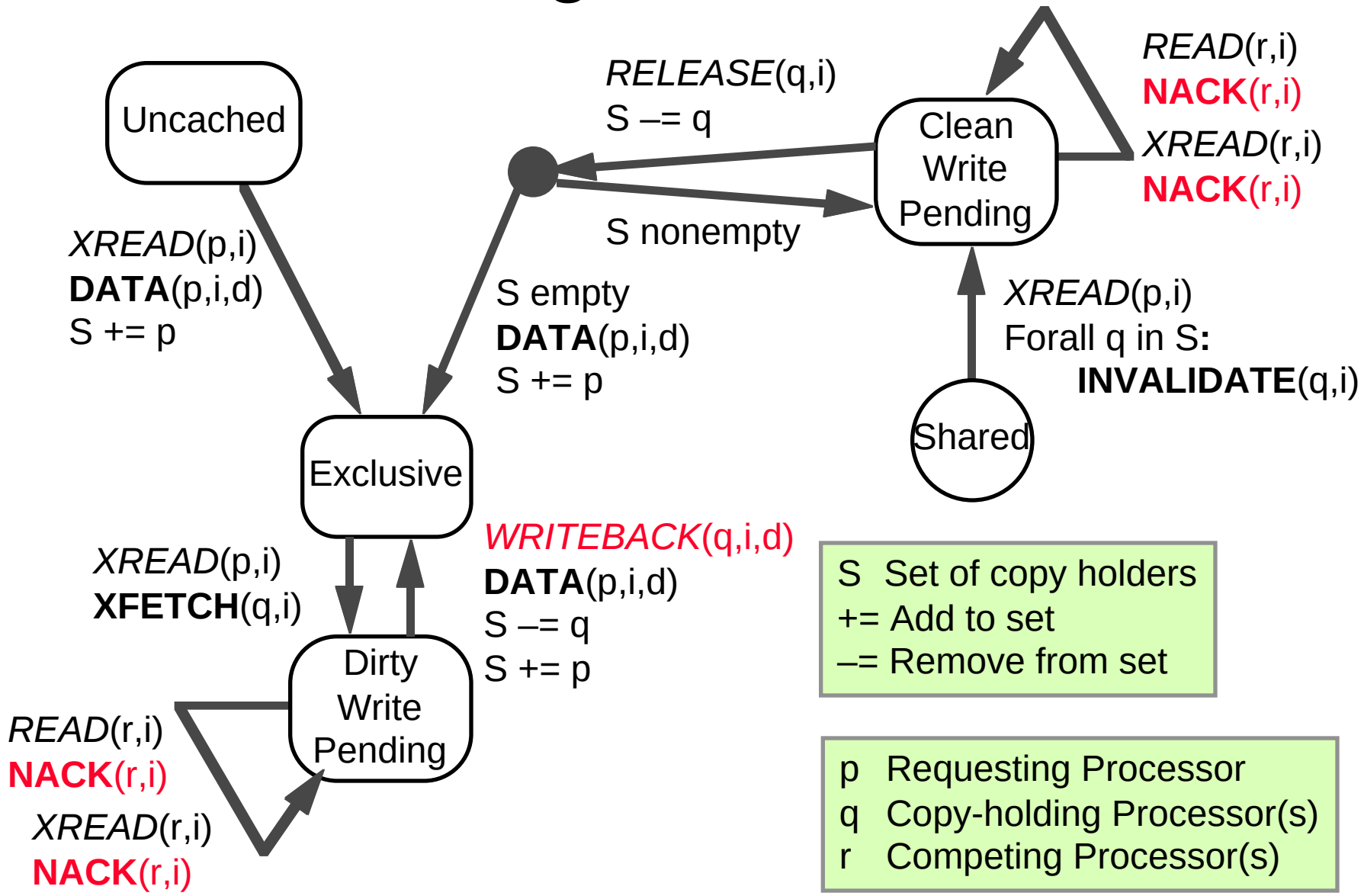
p	Requesting Processor
q	Copy-holding Processor
r	Competing Processor(s)

S	Set of copy holders
+=	Add to set
–=	Remove from set

Cache Handling of Processor Write



Home Handling of Processor Write

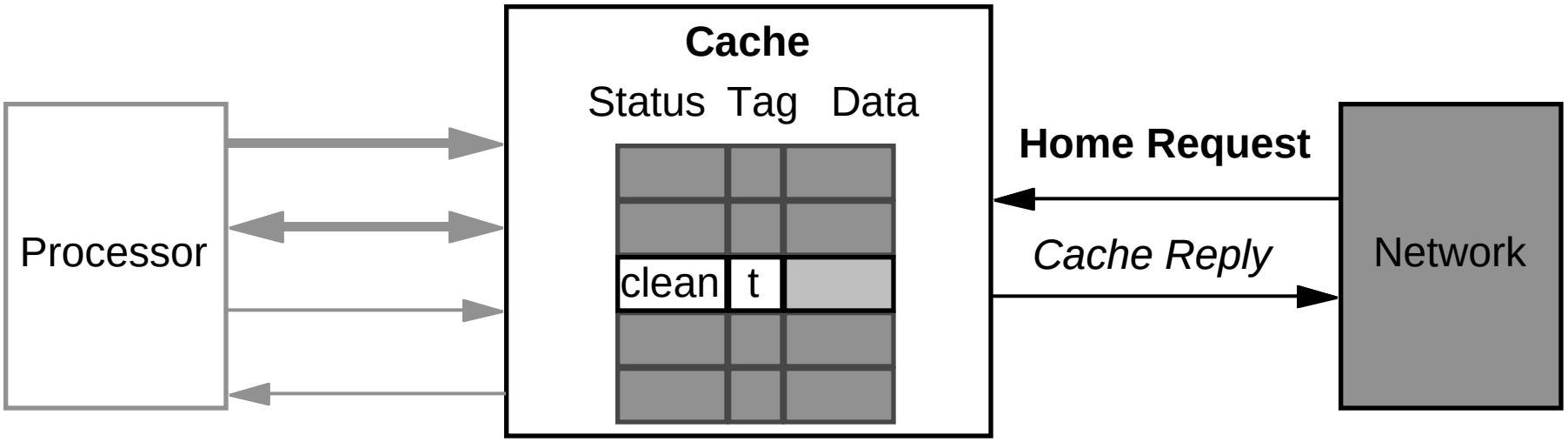


S Set of copy holders
+= Add to set
–= Remove from set

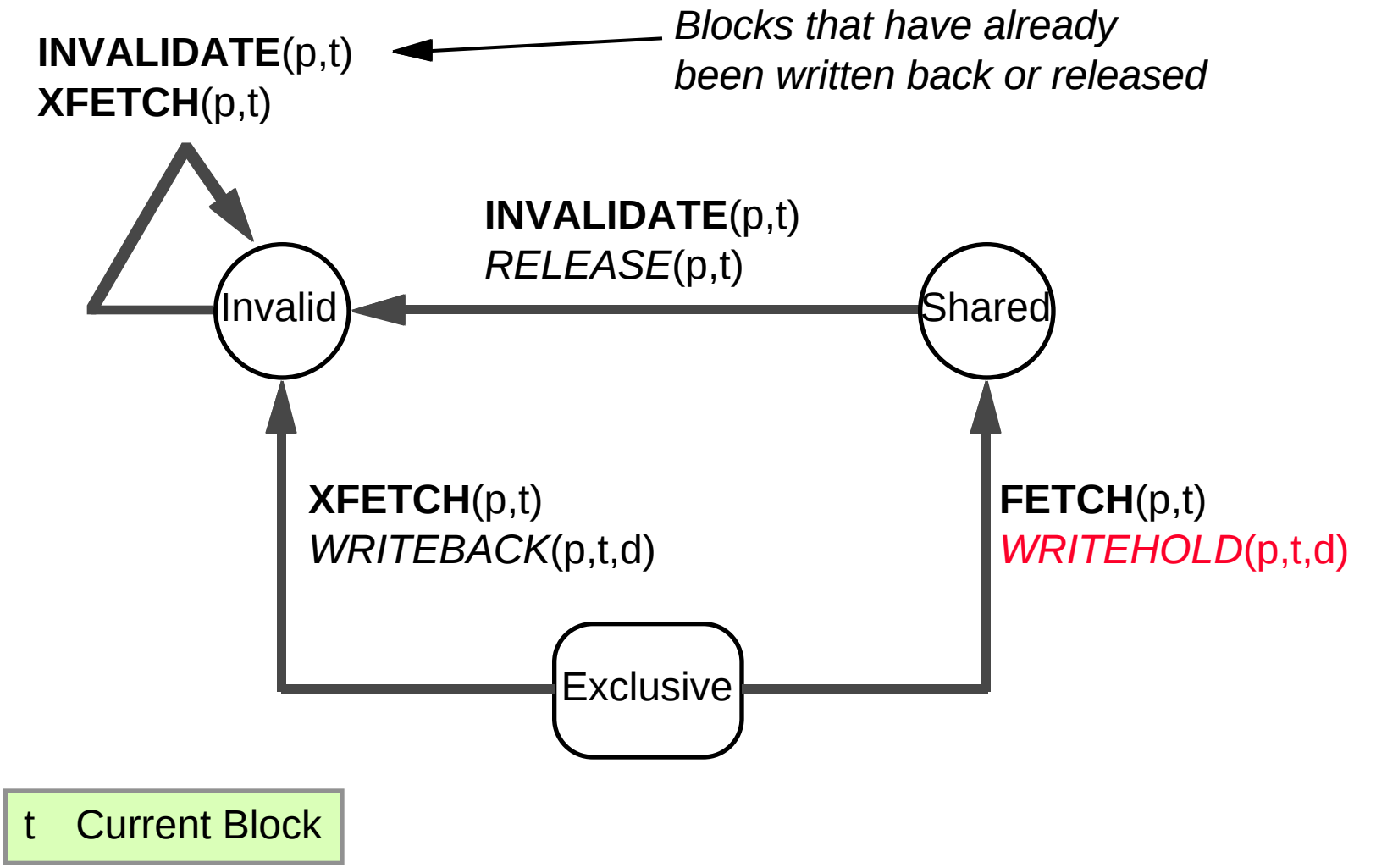
p Requesting Processor
q Copy-holding Processor(s)
r Competing Processor(s)

Network Monitoring by Cache

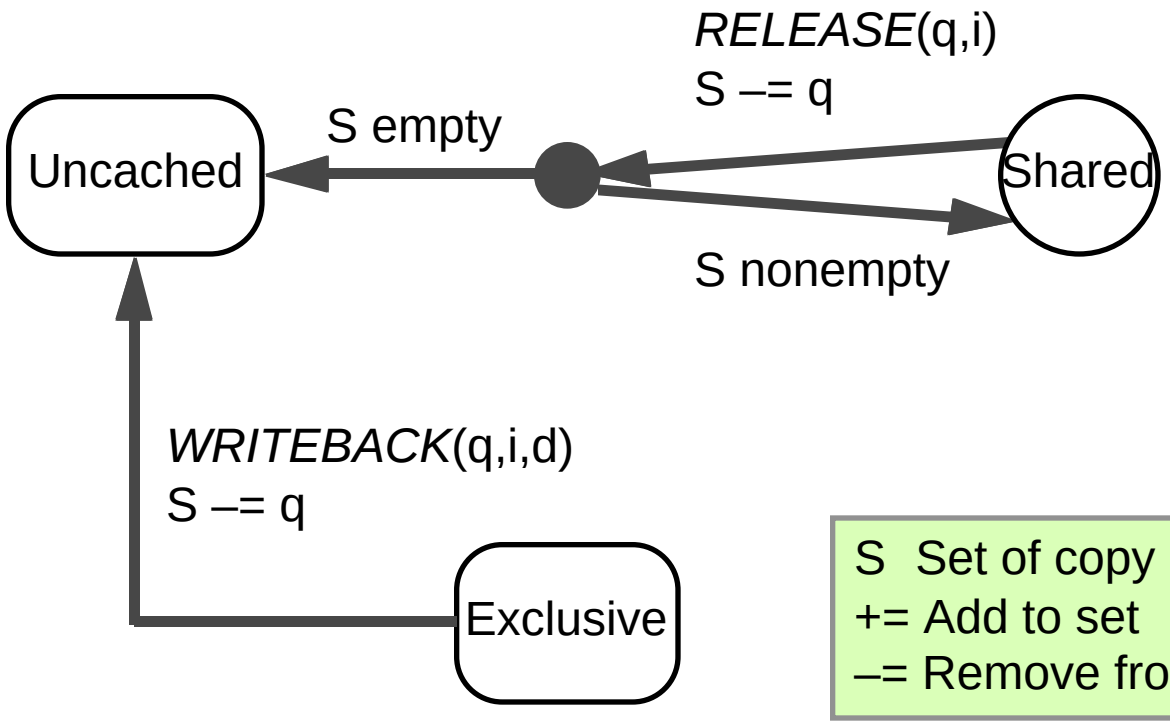
- **Cache receives commands from network**
  - Unlike snooping, only get messages regarding currently-held blocks
- **Possible actions**
  - Invalidate entry and release copy
  - Allow sharing of exclusively-held block
  - Surrender copy of block



Cache Handling of Network Commands



Home Handling of Unsolicited Evictions



S Set of copy holders
+= Add to set
–= Remove from set

q Copy-Holding Processor

Optimization 1: Make Clean Copy Exclusive

Cache Request

GIVEME	**p, a**	**Want exclusive version of currently-held block**

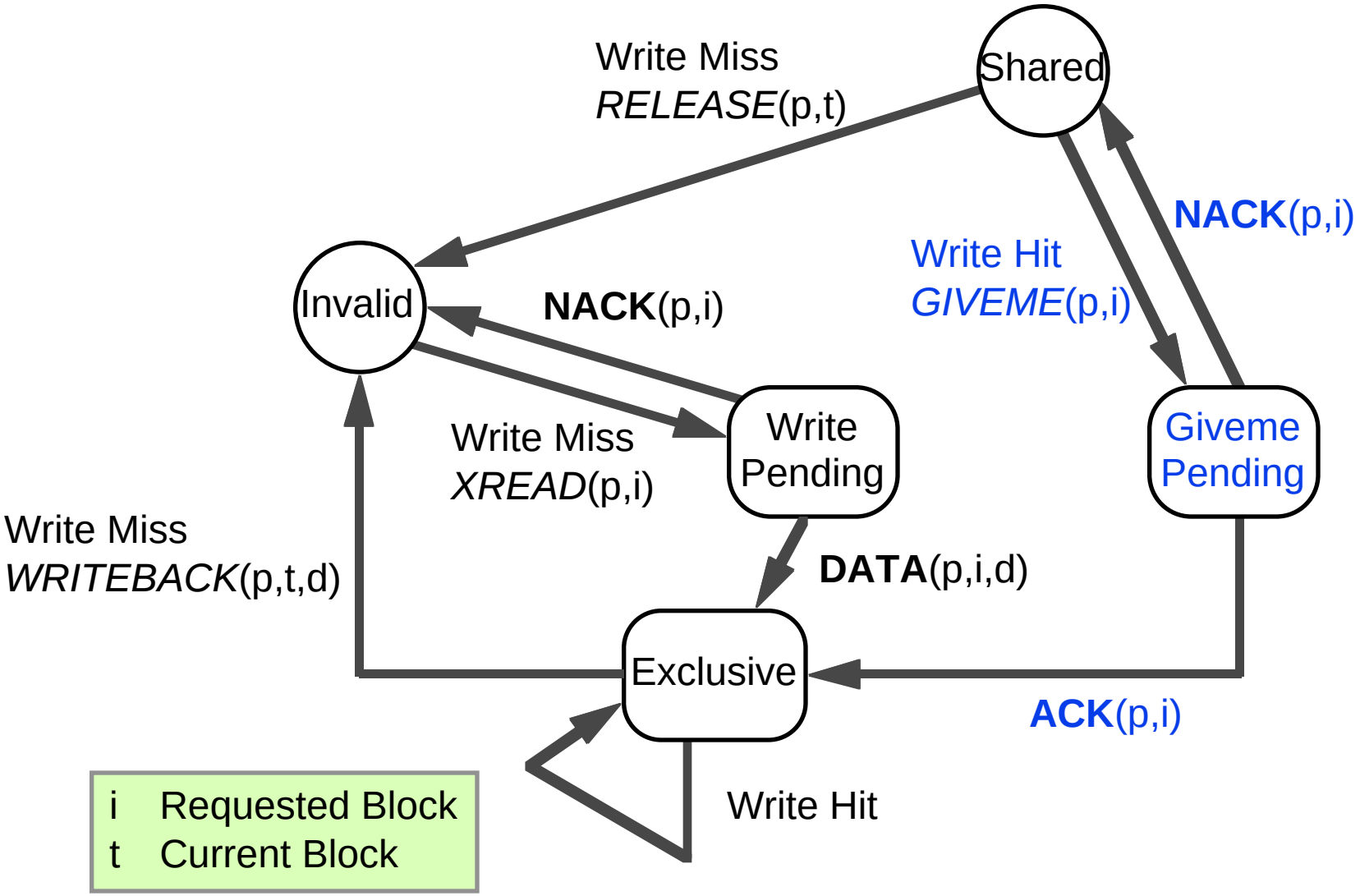
Home Response

ACK	**p, a**	**Permission granted**
NACK	**p, a**	**Cannot fulfill request**

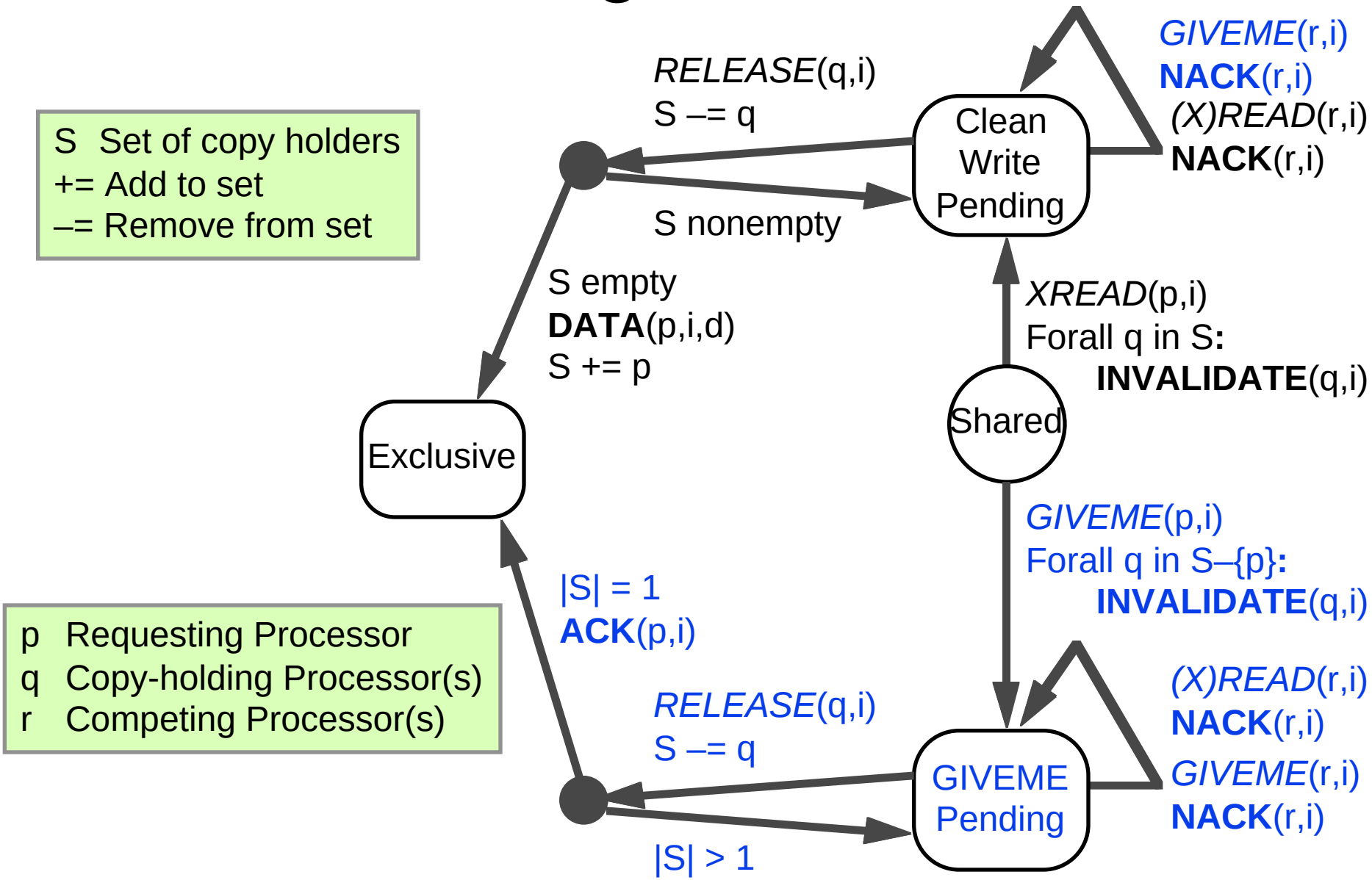
Motivation

- **Avoids need to transmit data**
  - Minor consideration
- **Potential for fast acknowledgement**
  - Next optimization
- **Allows processor to detect whether any writes since last read**
  - If receive ACK
  - To implement store-conditional

Cache Handling of Processor Write



Home Handling of Processor Write



Optimization 2: Fast Acknowledgement

Idea

- Enable processor write before other copies invalidated
- Only works if home has valid copy

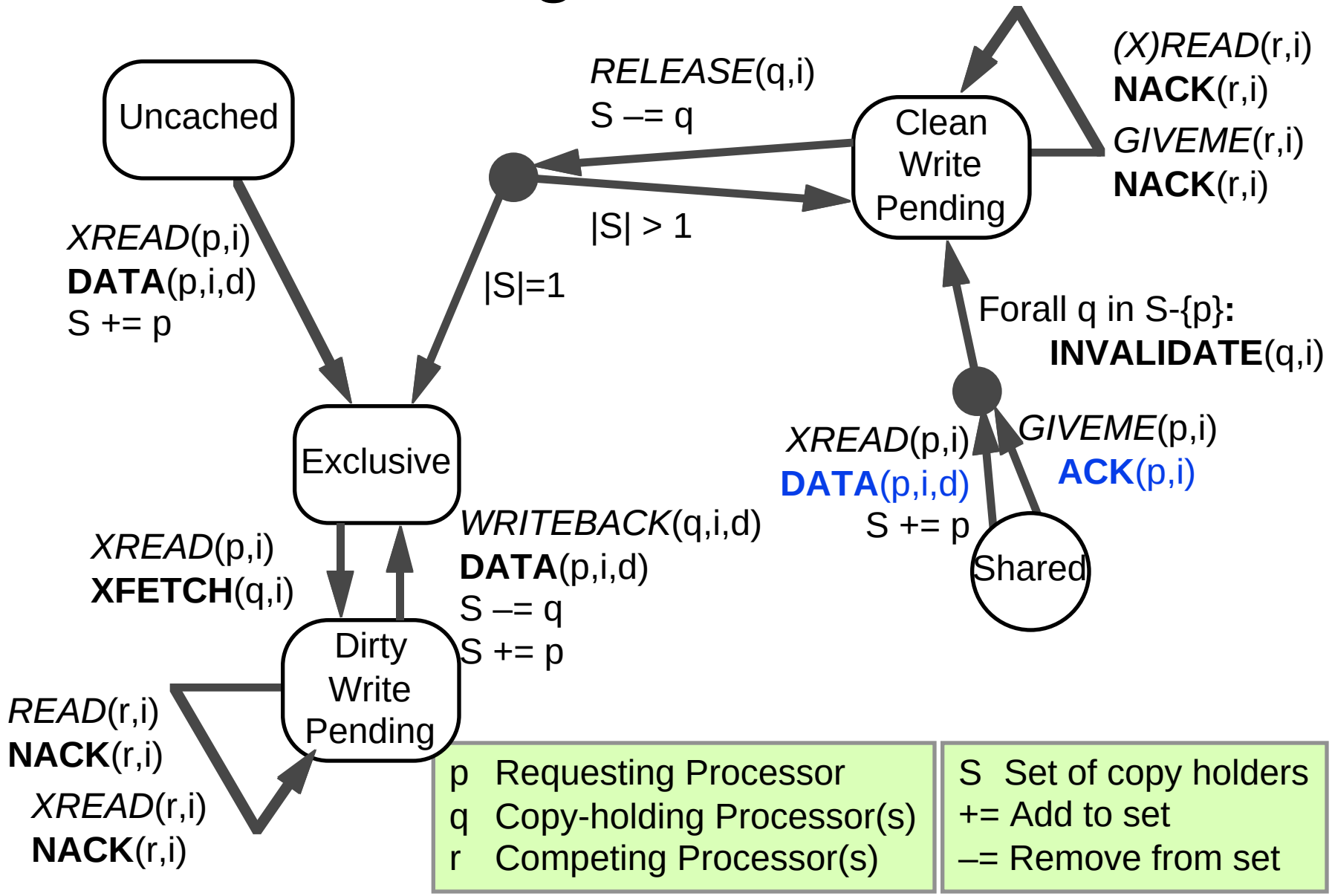
Motivation

- Reduces write latency

Risk

- Possible sequential inconsistency
- Other processors continue to read old data

Home Handling of Processor Write



Optimization 3: Data Forwarding

Idea

- Copy holding processor sends data directly to requestor
- Also sends message to home

Motivation

- Reduce latency

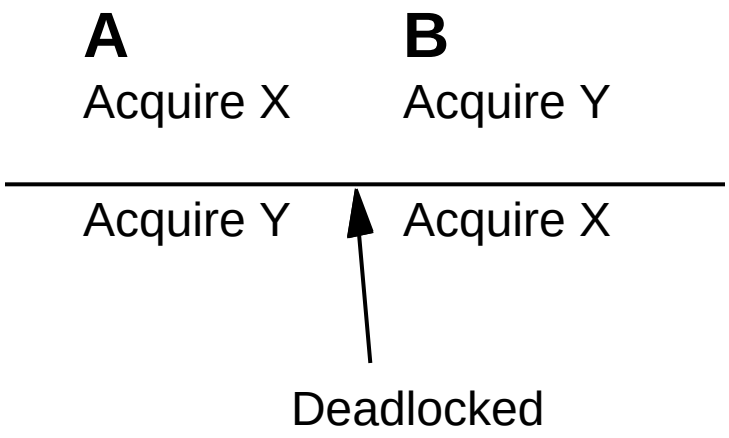
Challenges

- New possibilities for deadlock & inconsistency
- Protocol extension left as exercise

Deadlock Principles

Conditions for Deadlock

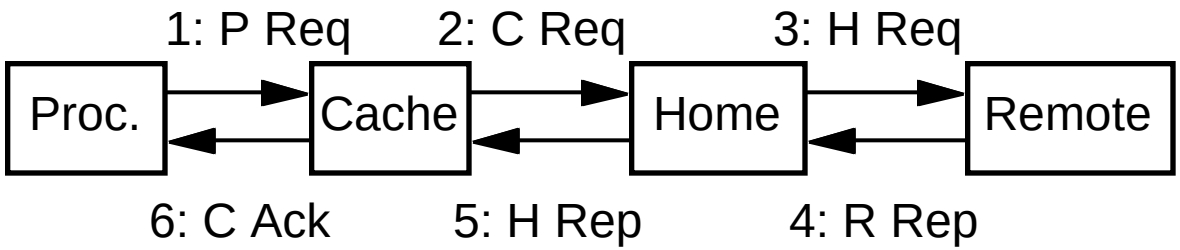
- **Finite resources**
  - Network links
  - Buffers
- **Circular dependence**
  - Independent agents A, B, …
  - Require same resources
  - Acquire in different order



Avoidance

- **Provide independent resources**
- **Acquire in same order**
- **Preemption**
  - Force agent to give up already-acquired resource

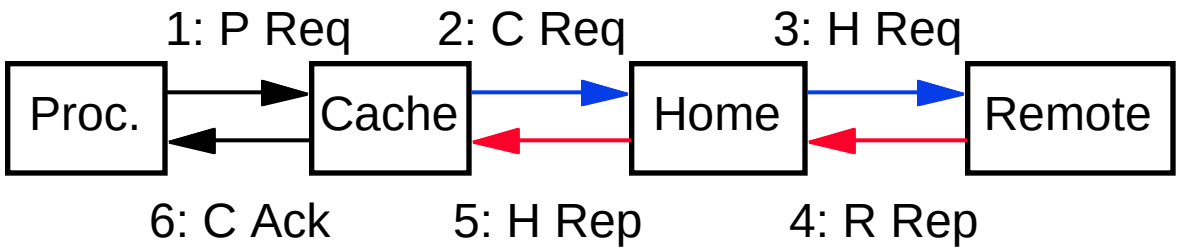
Resources for Memory Transaction



Sources of Resource Contention

- **Processor / memory system**
  - Resolve by stalling processor
  - Processor cannot hold onto memory resources
- **Handling single memory transaction**
  - Real possibility
- **Handling multiple memory transactions by single processor**
  - Generally serialize if cannot guarantee success
- **Handling multiple memory transactions by independent processors**
  - Real possibility

Protocol Deadlock Avoidance



Request-Reply Dependencies

- **Provide separate routing resources for request & reply messages**
  - Physically distinct or separate virtual channels
- **Require buffer for reply to be allocated before make request**
  - Stall processor if buffers not available at cache

Contentions Eliminated

- **Between any request & any reply**
- **Between any two replies**
  - Resources allocated at receiver beforehand
  - Processing of reply at receiver serves only to release resources

Out of Order Message Delivery

Assumed FIFO Ordering Between Sender & Receiver

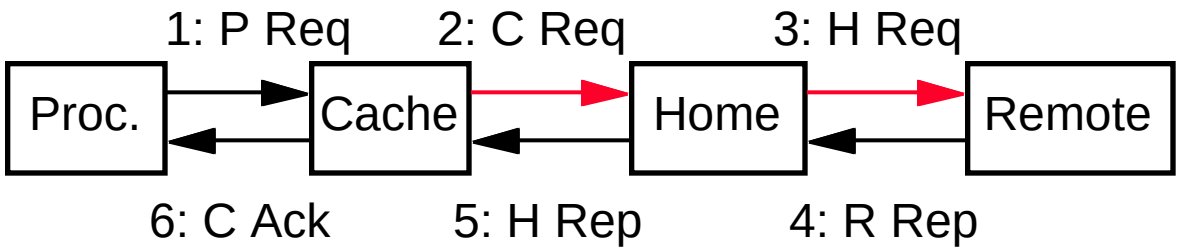
- **Cannot guarantee between separate request & reply networks**
  - Attempting to reorder at receiver could lead to deadlock

Example Problem

- **Cache sends *READ* request**
- **Home sends DATA reply**
  - Reply network
- **Home sends INVALIDATE request due to some other transaction**
  - Request network
- **Messages received at cache out-of-order**
  - Get INVALIDATE while in Pending Read state
- **Handling**
  - Respond with *RELEASE*
  - Wait for & discard **DATA**
  - Retry

Other Problems Still Exist!

Protocol Deadlock Avoidance (Cont.)



Request-Request Dependencies

- **Request by cache can generate request(s) by home**
  - May not have sufficient buffer resources
  - May cause network contention
- **Home can reply with NACK if cannot guarantee resource availability**
  - Processor will stall while cache retries
- **Timeout**
  - Home can give up on pending operation
  - Release resources
  - Respond to requestor with NACK

Fairness / Livelock

Livelock Possibilities

- Cache will keep getting NACK'ed
- Processor will be unable to complete read or write operation

Is This a Problem?

- Don't know of protocol that guarantees fairness
- Even if single memory operation guaranteed to complete, hard to guarantee fairness of synchronization protocol

Avoidance

- Each memory transaction has priority
- Dynamically increase as get NACK'ed
- Preempt lower priority transactions

SGI Origin System

Similar to Example Protocol

- **Both derivatives of Stanford DASH protocol**

Enhanced Implementation Features

- **Designed to handle out-of-order messages at all levels**
  - Important adaptive routing
- **Support "clean exclusive" state**
  - Give to read request when block unshared
  - Expedites subsequent write
- **Forwarding of data from remote copy holder**
  - Also in DASH
- **Evictions of clean blocks don't generate network traffic**
- **More effective handling of request-request deadlock avoidance**
  - Send back information needed for requesting cache to handle invalidations
- **Variable granularity directory representation**
  - 64-bit vector represents either 64 clustered processors or 64 processor sets

SGI Origin Performance

Processor

- 195 MHz R10000

Latencies

Level	*Nanoseconds*	*Clock Cycles*
• L1 cache	5	1
• L2 cache	56	11
• Local memory	310	60
• 4P avg. remote memory	540	105
• 128P avg. remote memory	945	184

Dealing with High Latency

- Support prefetch operations
- Automatic page migration in virtual memory system