Shared Virtual Memory

Context

- Parallel architectures & programming models
  - Bus-based shared memory multiprocessors
    - h/w support for coherent shared memory
    - can run both shared memory & message passing
    - scalable to 10's of nodes
  - Distributed memory machines/clusters of workstation
    - provides message passing interface
    - scalable up to 1000s of nodes
    - cheap! economies of scale, commodity shelf h/w

Distributed Shared Memory

- Radical idea: let us not have the hardware dictate what programming model we can use
- Provide a shared address space abstraction even on clusters

- Is this a good idea? What are the upsides/ downsides of this approach?

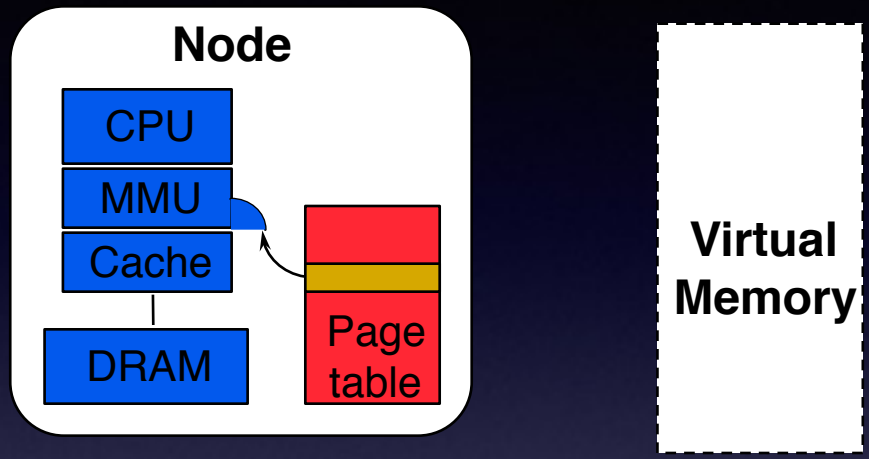
How do we provide this abstraction?

- Operating system support:
  - e.g., Ivy, Treadmarks, Munin
- Compiler support (Shasta)
  - minimize overhead through compiler analysis
  - object granularity as opposed to byte granularity
  - notions of immutable data, sharing patterns
- Limited hardware support (Wisconsin Wind Tunnel, DEC memory channel)

IVY Shared Virtual Memory

- Seminal system that sparked the entire field of DSM (distributed shared memory)
- Motivations:
  - sharing things on a network
  - "embassy" system to support a network file system between two different OSes
  - parallel **scheme** run time system on a cluster
- Focus: parallel computing and not distributed computing
  - less emphasis on request-reply, fault-tolerance, security

Traditional Virtual Memory



- Page Table entry:

Virt. page #	physical page #	valid

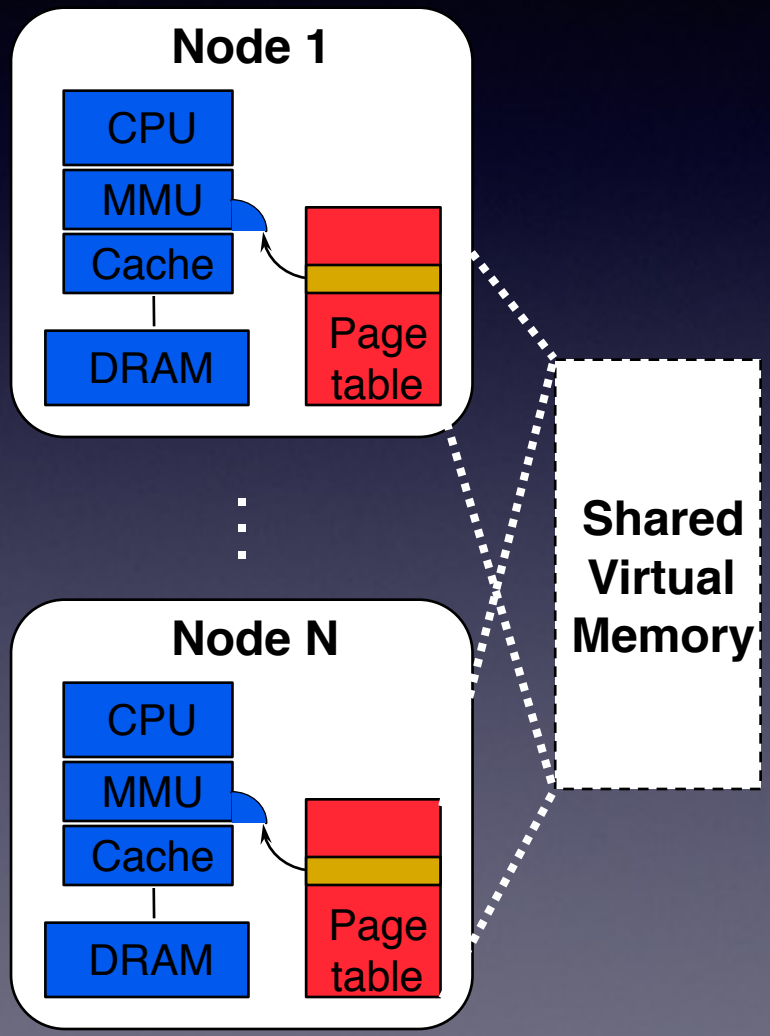
- If “valid”, translation exists
- If “not valid”, traps into the kernel, gets the page, re-executes trapped instruction
- Check is made for every access; TLB serves as a cache for the page table entries

Shared Virtual Memory

- Pool of "shared pages": if not local, page is not mapped
- Page table entry access bits

Virt. page #	physical page #	valid	access

- H/w detects read access to invalid page
  - read faults
- H/w detects writes to mapped memory with no write access
  - write faults
- OS maintains consistency at VM page level
  - copying data
  - setting access bits



Issues

- Programming model (as in coherence, consistency, etc.)
- Correctness of implementation
- Performance related issues

Programming Model

- Contract between programmer and h/w
- Shared memory abstraction typically means two related concepts:
  - Coherence
  - Consistency model (e.g., sequential consistency, linearizability)

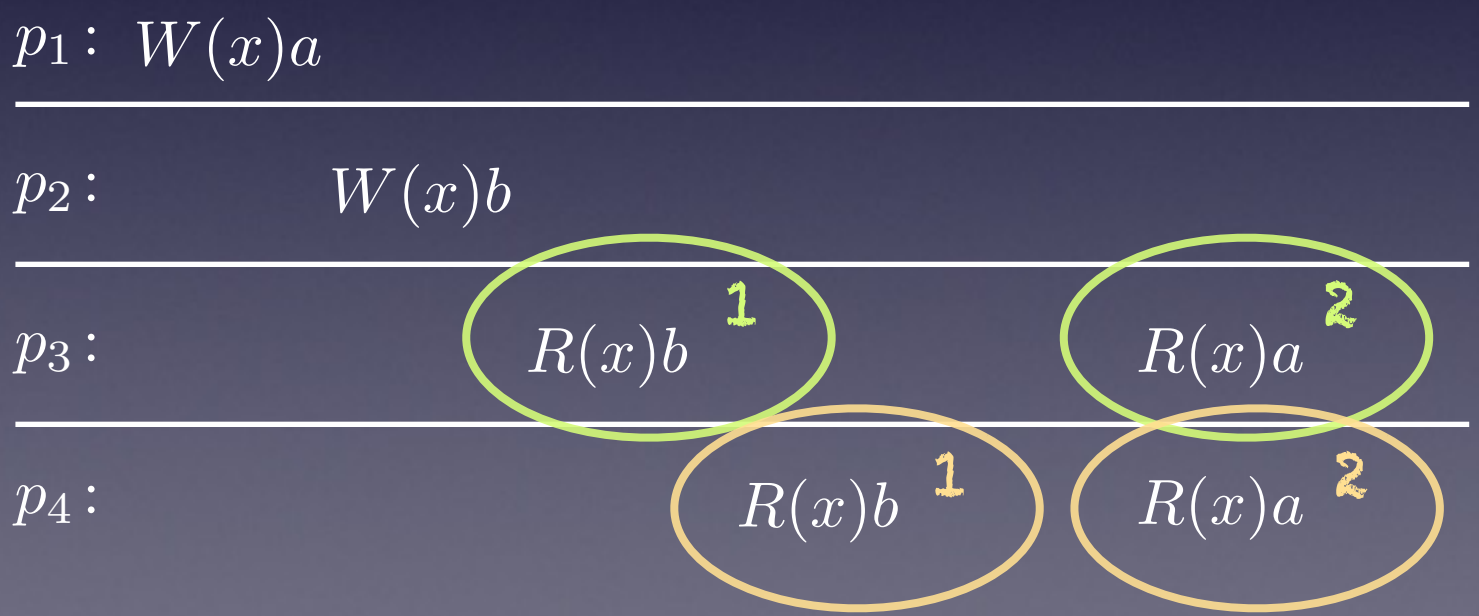
Coherence vs. Consistency

- Coherence: writes are propagated to other nodes; the writes to a particular memory location are seen in order

- Consistency:the writes to multiple distinct memory location or writes from multiple processors to the same location are seen in a well-defined order

Sequential Consistency

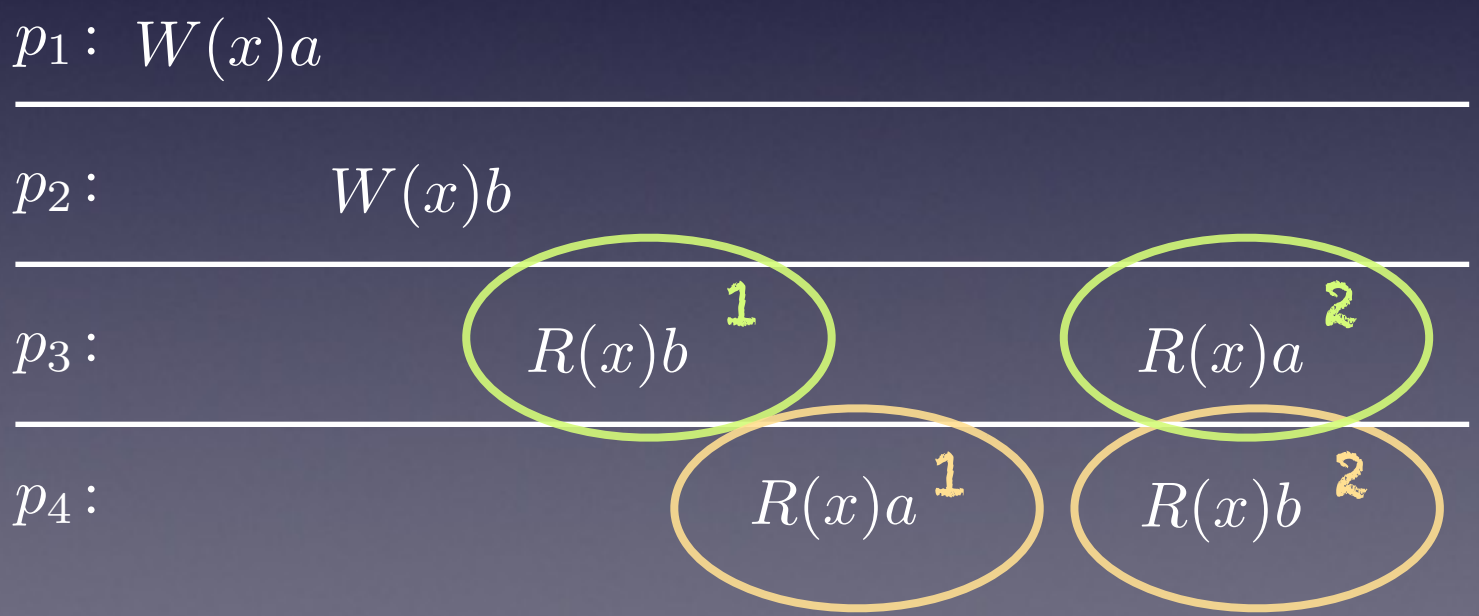
- "The result of any execution is the same as if the operations of all the processes were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program" (Lamport, 1979)



Is this data store sequentially consistent?

Sequential Consistency

- "The result of any execution is the same as if the operations of all the processes were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program" (Lamport, 1979)



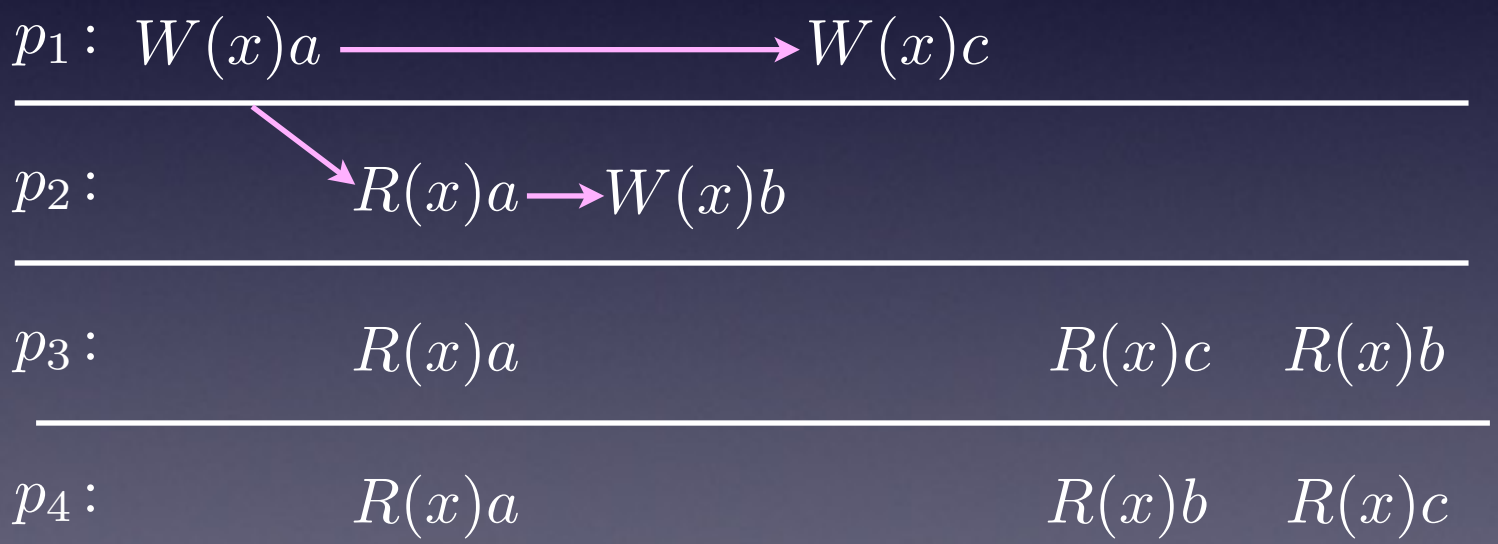
Is this data store sequentially consistent?

Other Consistency Models

- Can we have consistency models stronger than sequential consistency?
- How do we weaken sequential consistency?

Weakening Sequential Consistency: Causal Consistency

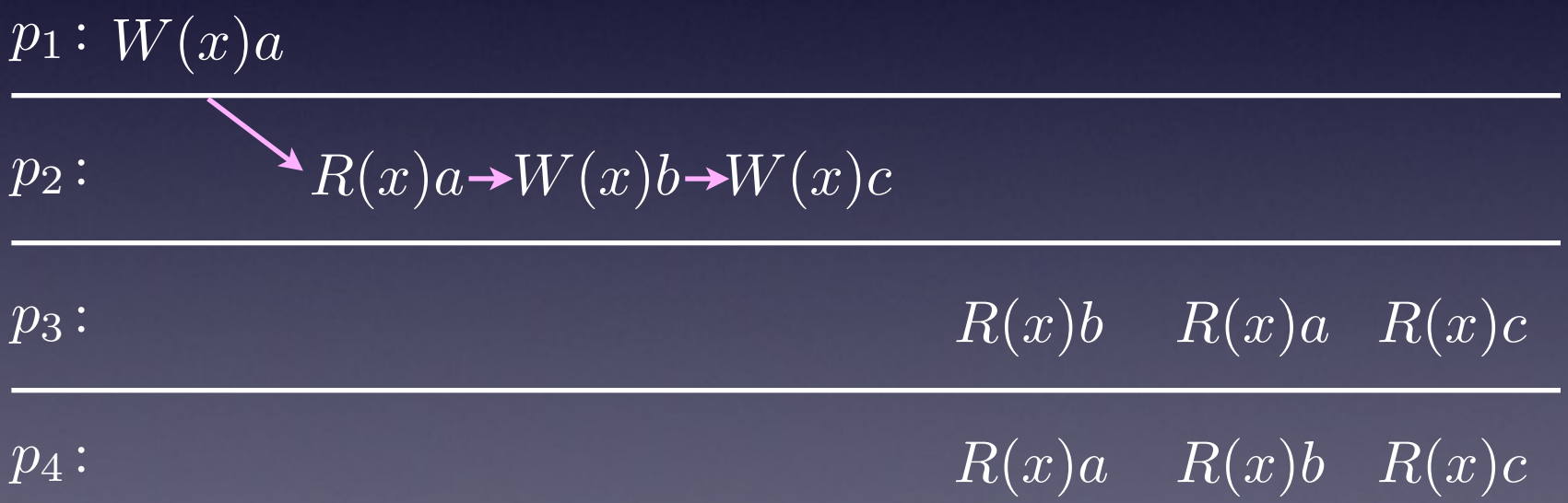
- Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines. (Hutto and Ahamad, 1990)



Is this data store sequentially consistent?
Causally consistent?

More Weakening: FIFO Consistency

- "Writes done by a single process are seen by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes" (PRAM consistency, Lipton and Sandberg 1988)



Is this data store causally consistent?

Is this data store FIFO consistent?

Programming Complexity

Process p_1

$x := 1$

if $(y = 0)$ then

kill (p_2)

Process p_2

$y := 1$

if $(x = 0)$ then

kill (p_1)

Initially, $x = y = 0$

What are the possible outcomes?

- What do you make out of these consistency models?

Ivy DSM

- Goal: provide sequentially consistent shared memory
- Baseline Implementation:
  - centralized manager
  - manager maintains the "owner" and the set of readers ("copyset")

Read Faults

- Handler on client:
  - asks manager
  - manager forwards request to owner
  - owner sends the page
  - requester sends an ACK to manager

Pseudocode

Read Fault Handler:

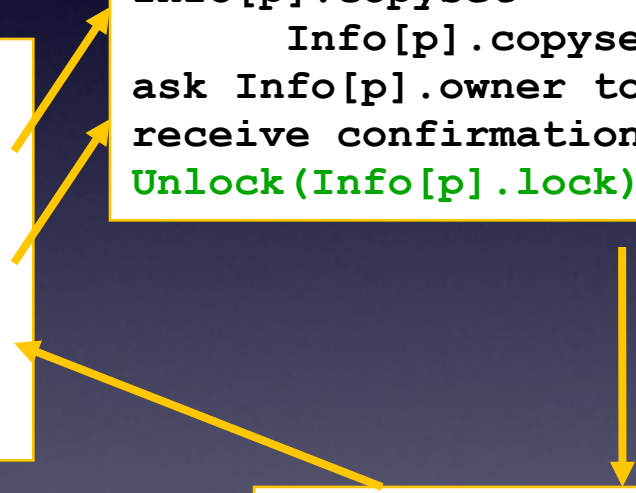
```
Lock(Ptable[p].lock);
ask manager for p;
receive p;
send confirmation to manager;
Ptable[p].access = read;
Unlock(Ptable[p].lock);
```

Manager:

```
Lock(Info[p].lock);
Info[p].copyset =
      Info[p].copyset U {reqNode};
ask Info[p].owner to send p;
receive confirmation from reqNode;
Unlock(Info[p].lock);
```

Read Server:

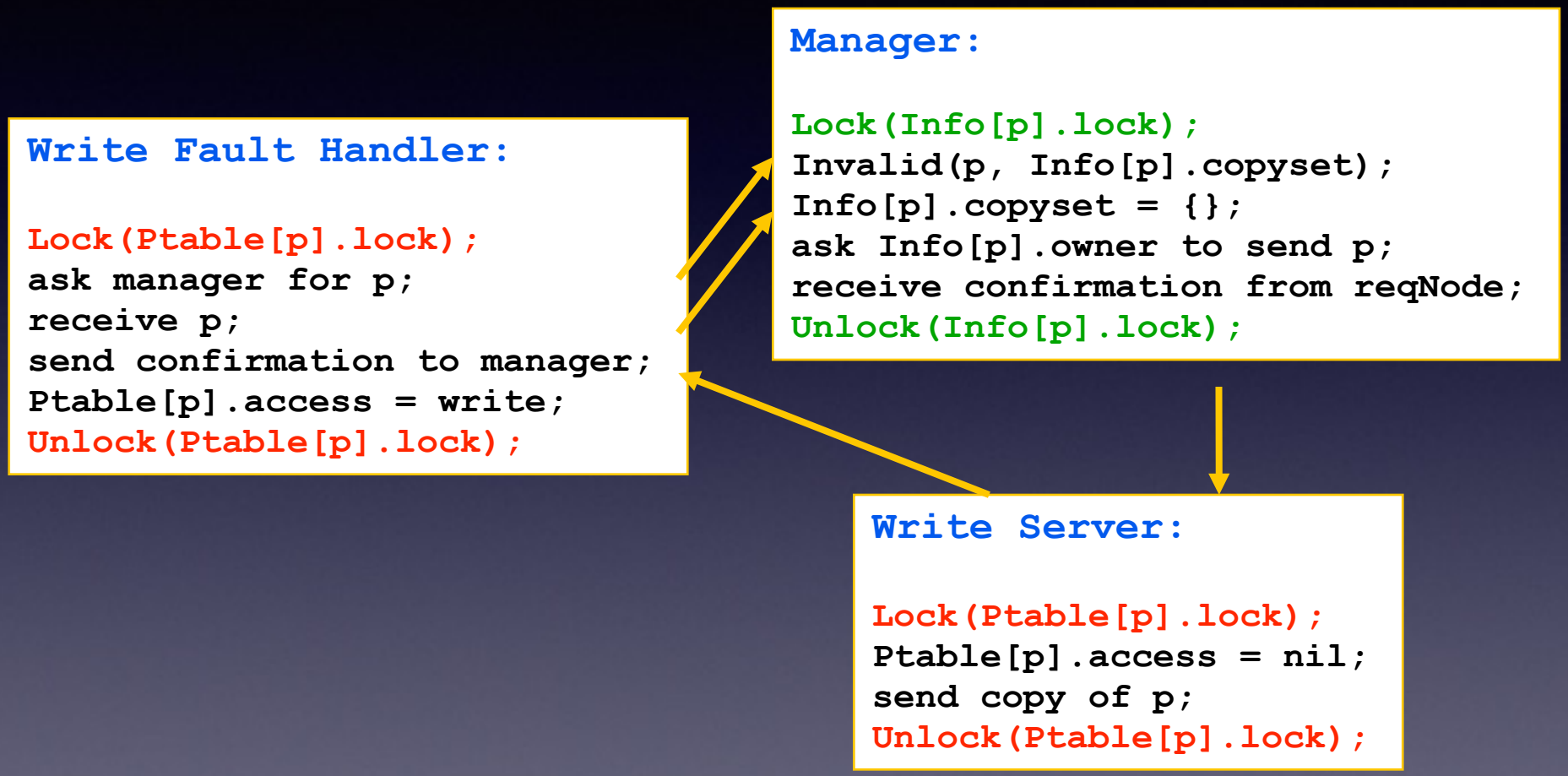
```
Lock(Ptable[p].lock);
Ptable[p].access = read;
send copy of p;
Unlock(Ptable[p].lock);
```



Write Faults

- Handling includes invalidations:
  - make request to manager
  - copies are invalidated
  - manager forwards request to owner
  - owner relinquishes page to requester
  - requester sends an ACK to the owner

Write Pseudocode



Scenarios

- Consider P1 and P2 caching a page with "read" perms
- What happens if both perform a "write" at the same time?

Question

- Can the confirmation messages be eliminated?

Scenarios

- Consider P1 is owner of page
- P2 performs a read
- P3 performs a write
- What if manager handles write before read is complete?

Improved Manager

- Owner serves as the manager for each page

Read Fault Handler:

```
Lock(Ptable[p].lock);
ask manager for p;
receive p;
Ptable[p].access = read;
Unlock(Ptable[p].lock);
```

Read Server:

```
Lock(Ptable[p].lock);
If I am owner {
    Ptable[p].access = read;
    Ptable[p].copyset =
        Ptable[p].copyset U {reqNode};
    send copy of p;
} else {
    forward request to probable owner;
}
Unlock(Ptable[p].lock);
```

Performance Questions

- In what situations will IVY perform well?