Today

Distributed Subsystems (Lesson 7)

✓ * Global memory system

⇒ * Distributed Shared Memory

Friday

* Distributed file systems

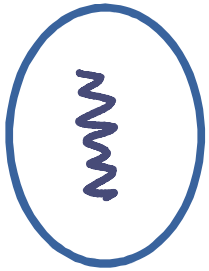
(*watch video*)

Thought Experiment

DSM

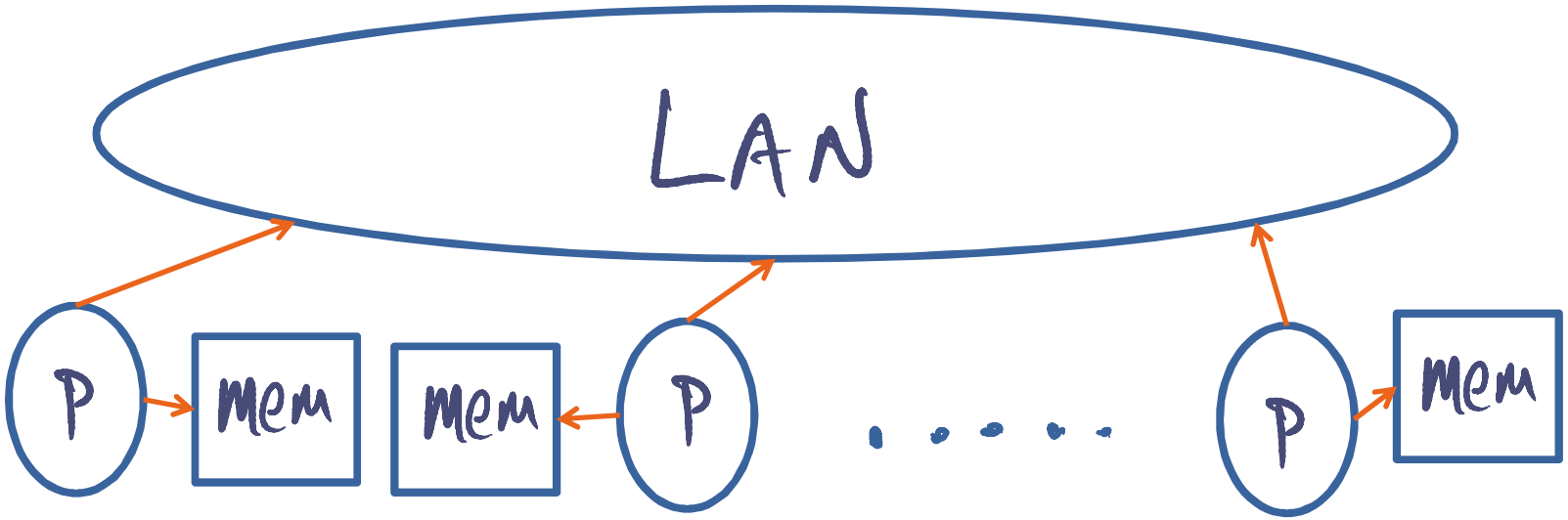
- Can we make the cluster appear like a shared memory machine?

Cluster as a parallel machine

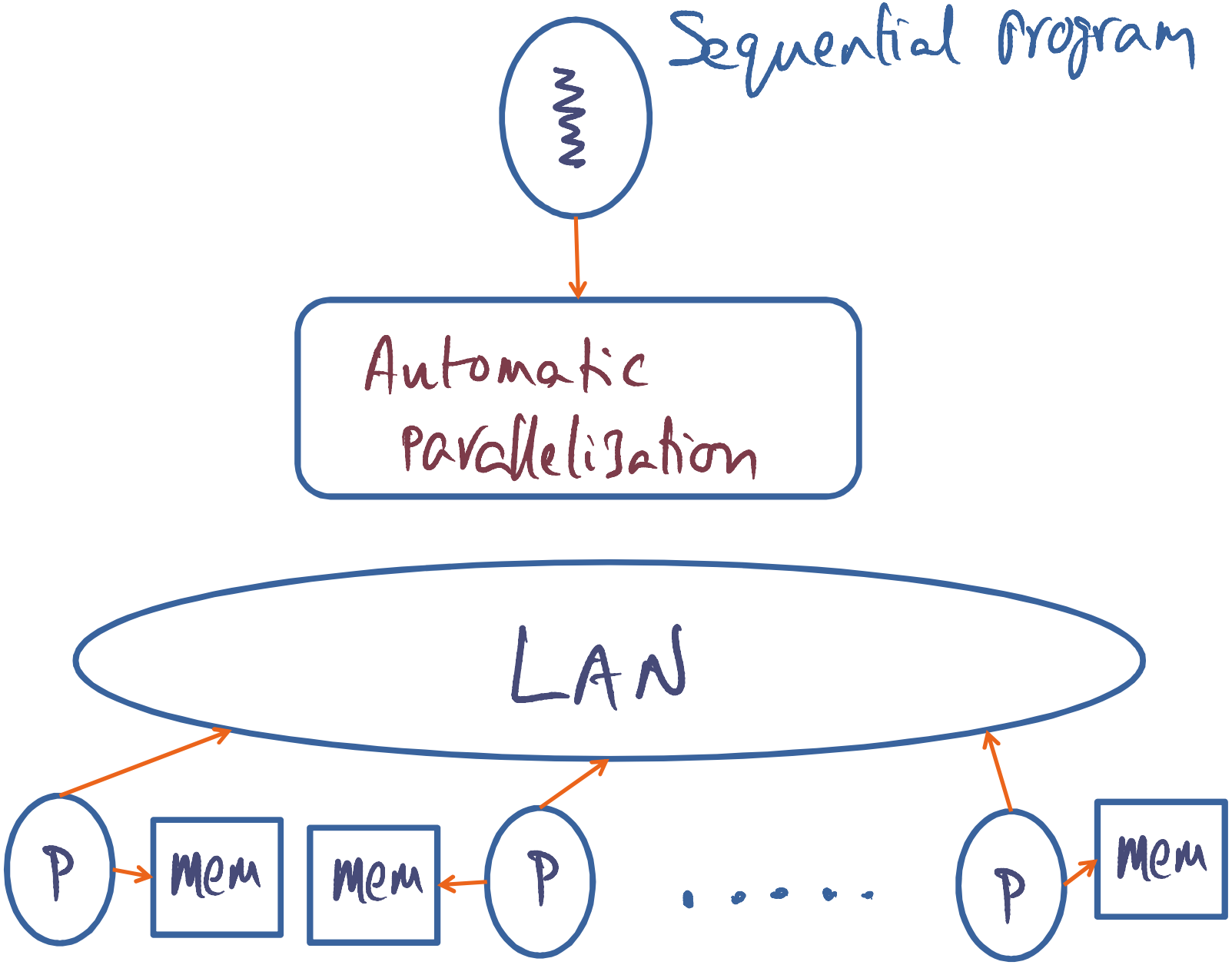


Sequential Program

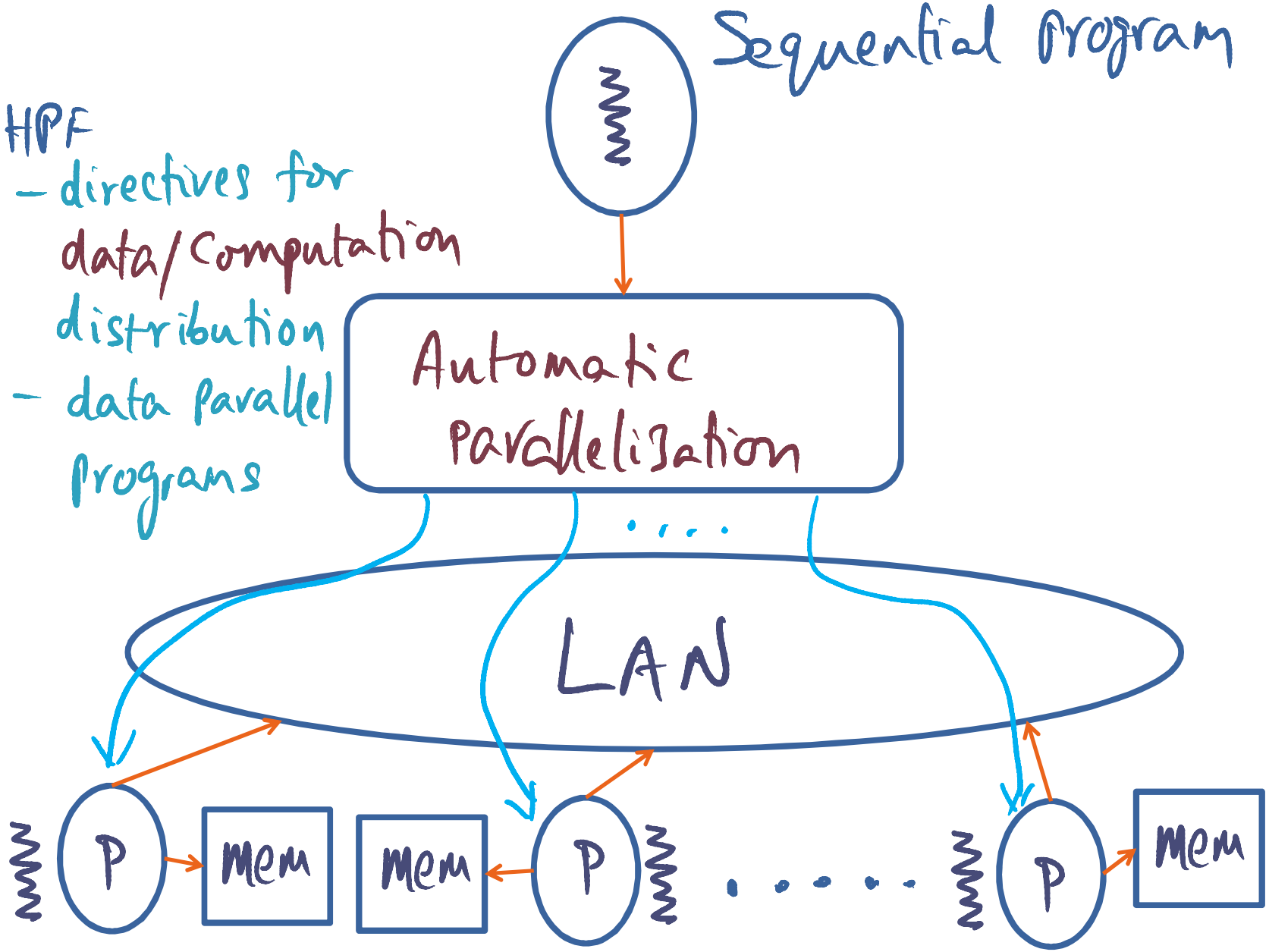
? How to exploit the cluster!



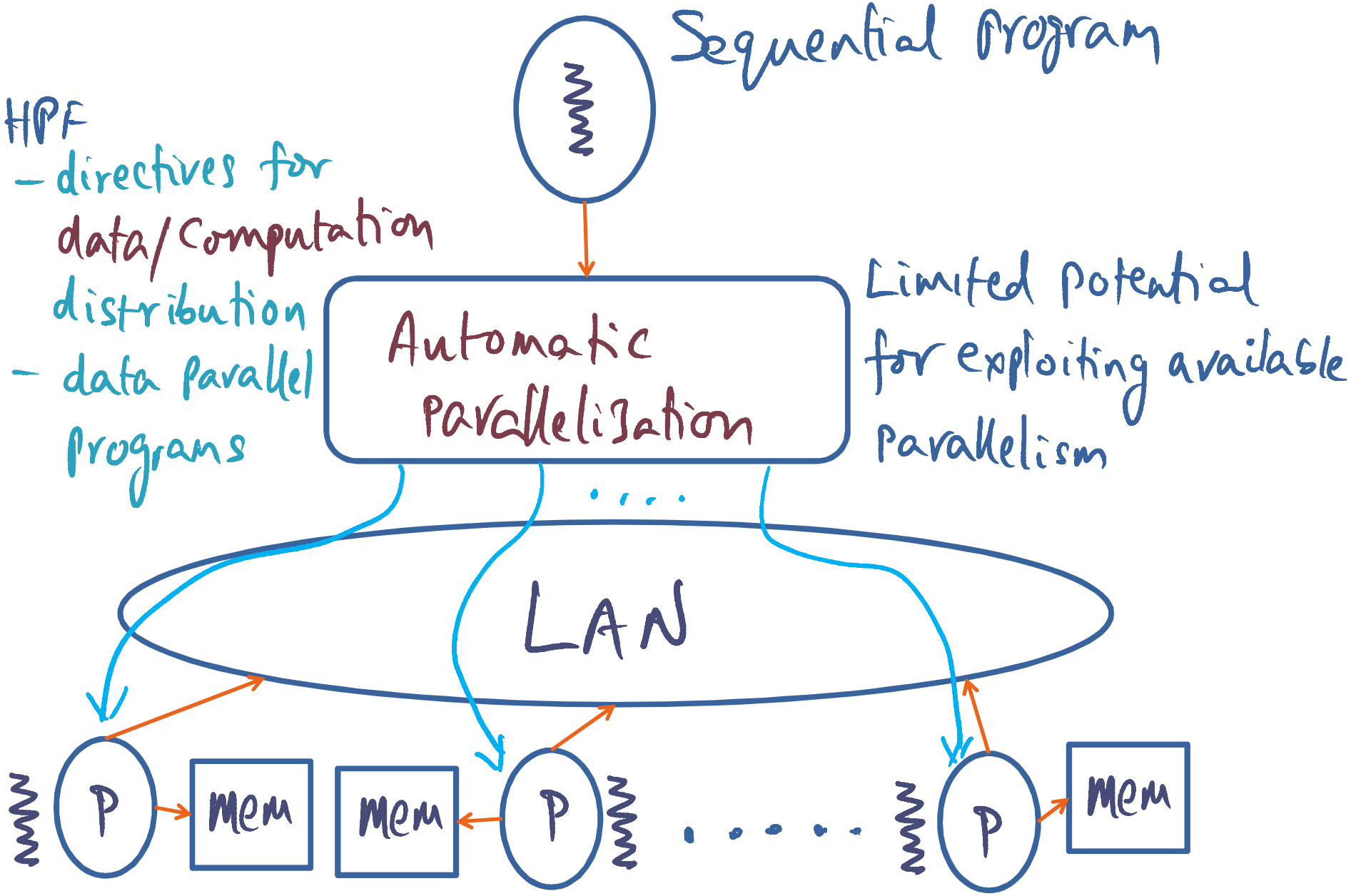
Cluster as a parallel machine



Cluster as a parallel machine

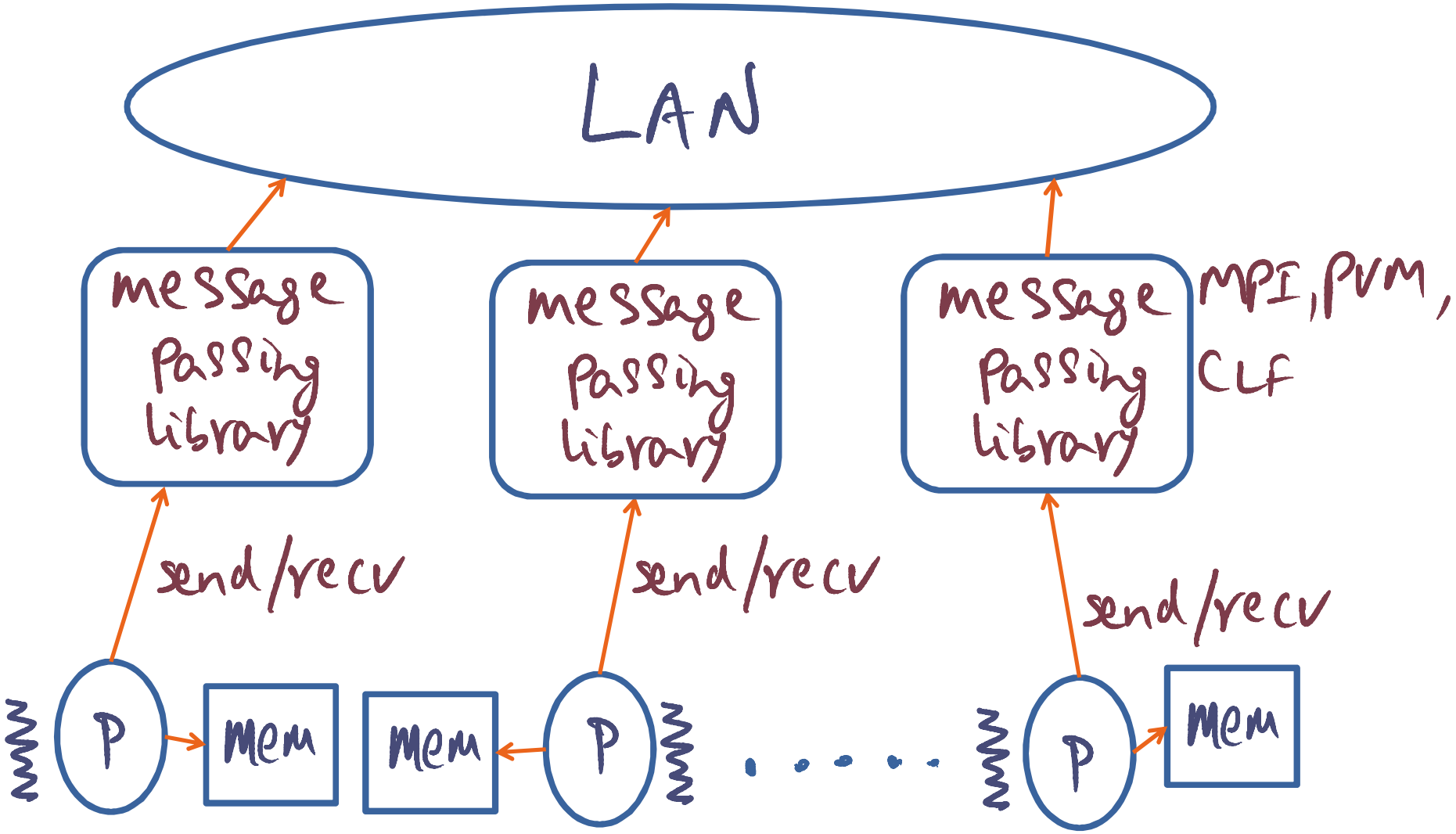


Cluster as a parallel machine



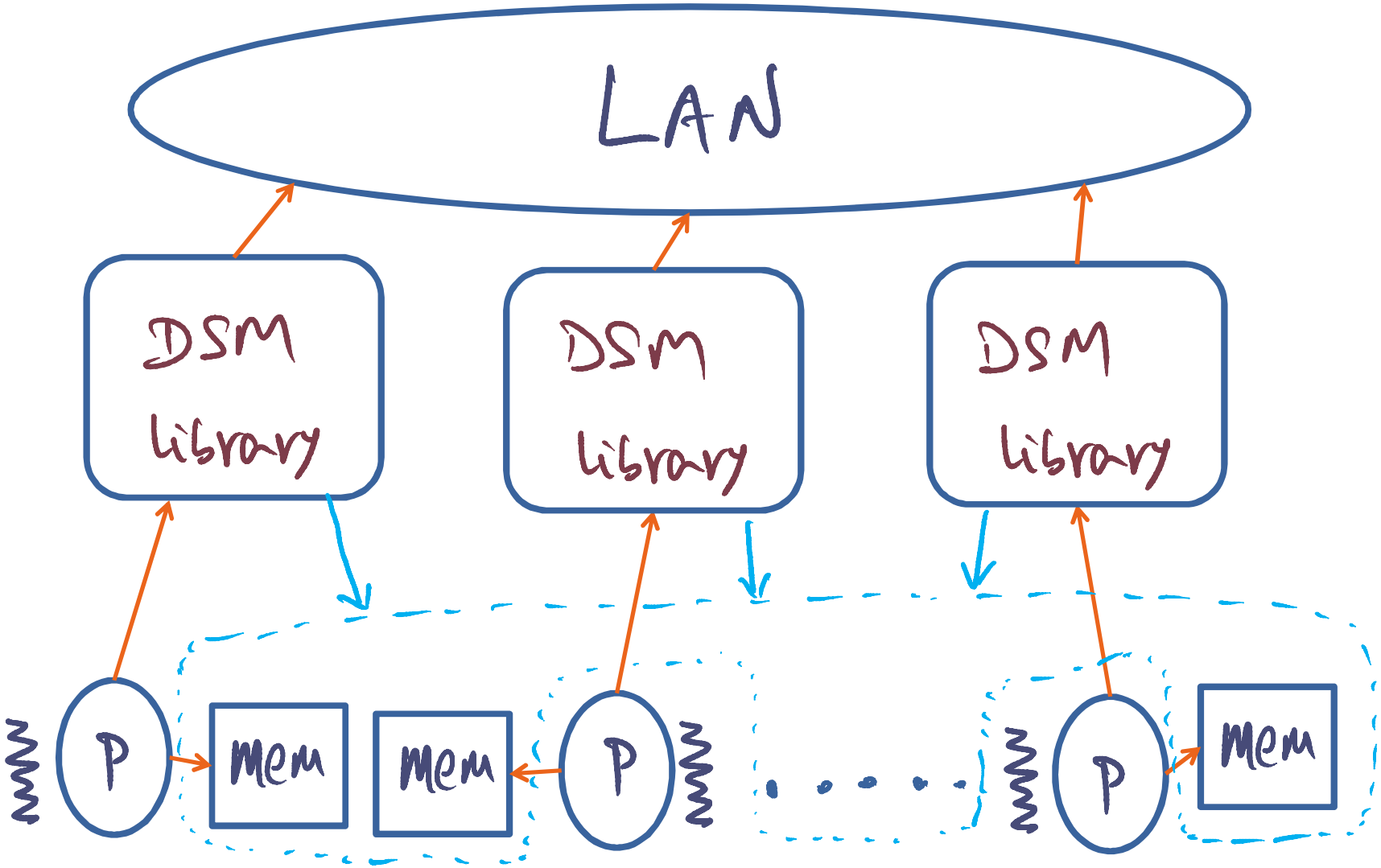
Cluster as a parallel machine

Parallel program: message passing

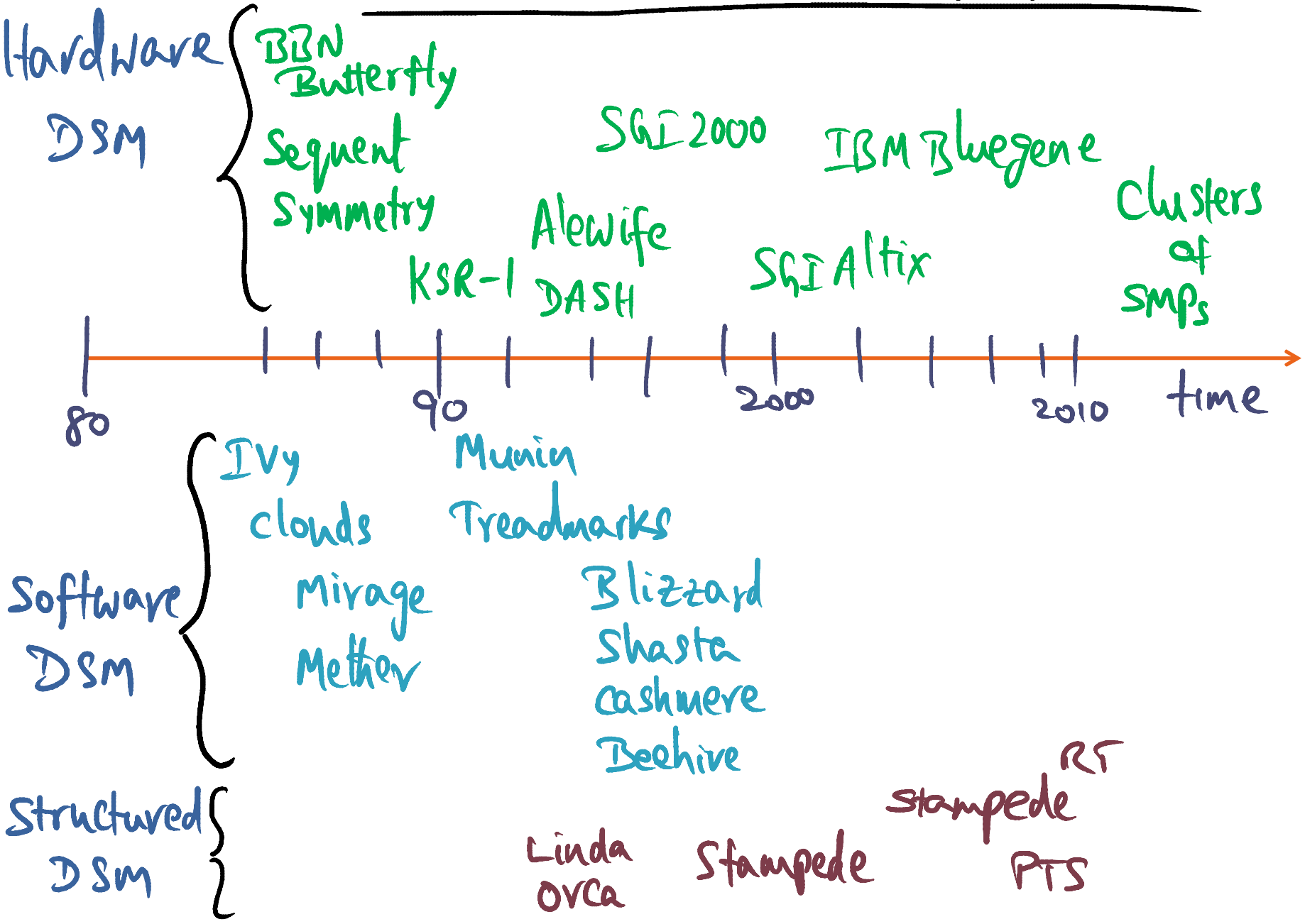


Cluster as a parallel machine

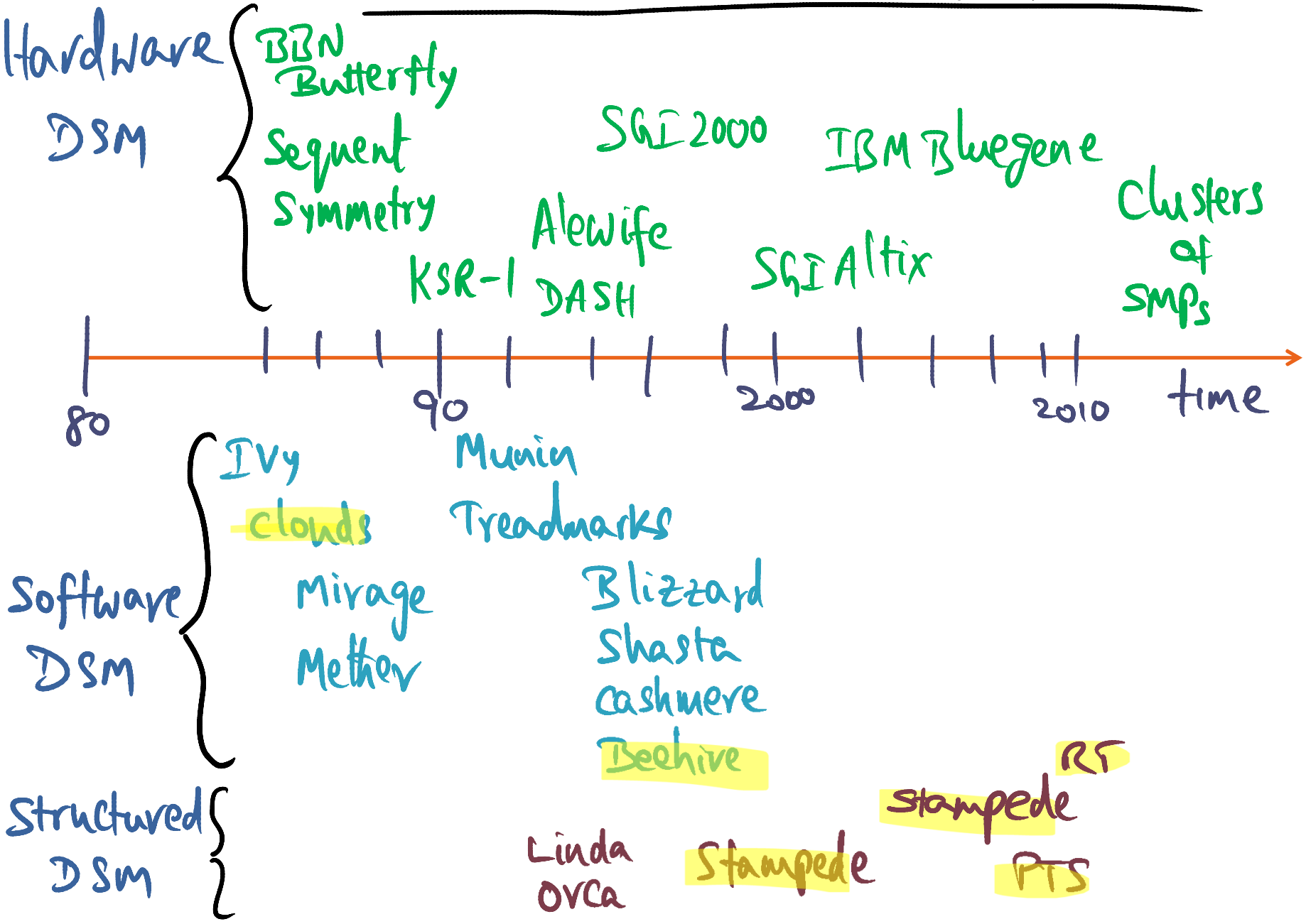
Parallel program: Distributed shared memory



History of shared memory systems

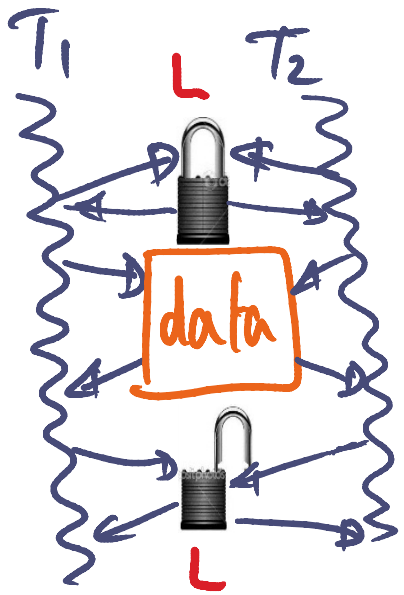


History of shared memory systems

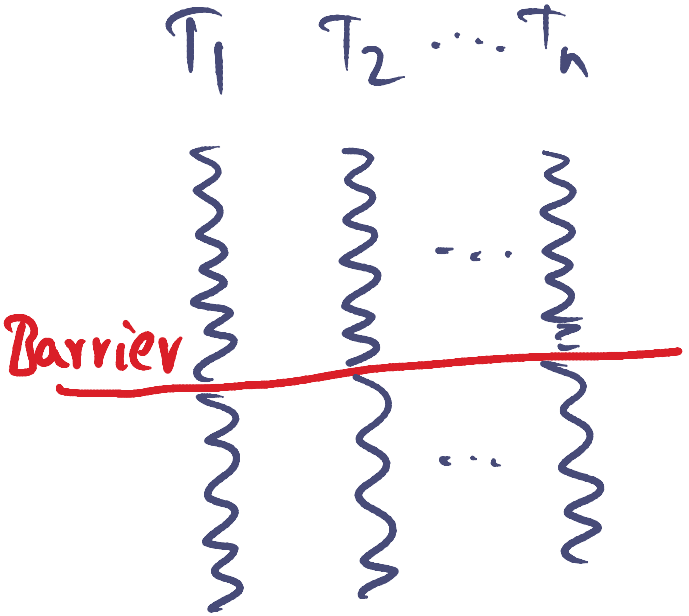


Shared memory Programming

Lock



Barrier



Two types of memory accesses

- normal r/w to shared data
- r/w to synchronization variables

Memory Consistency

↓

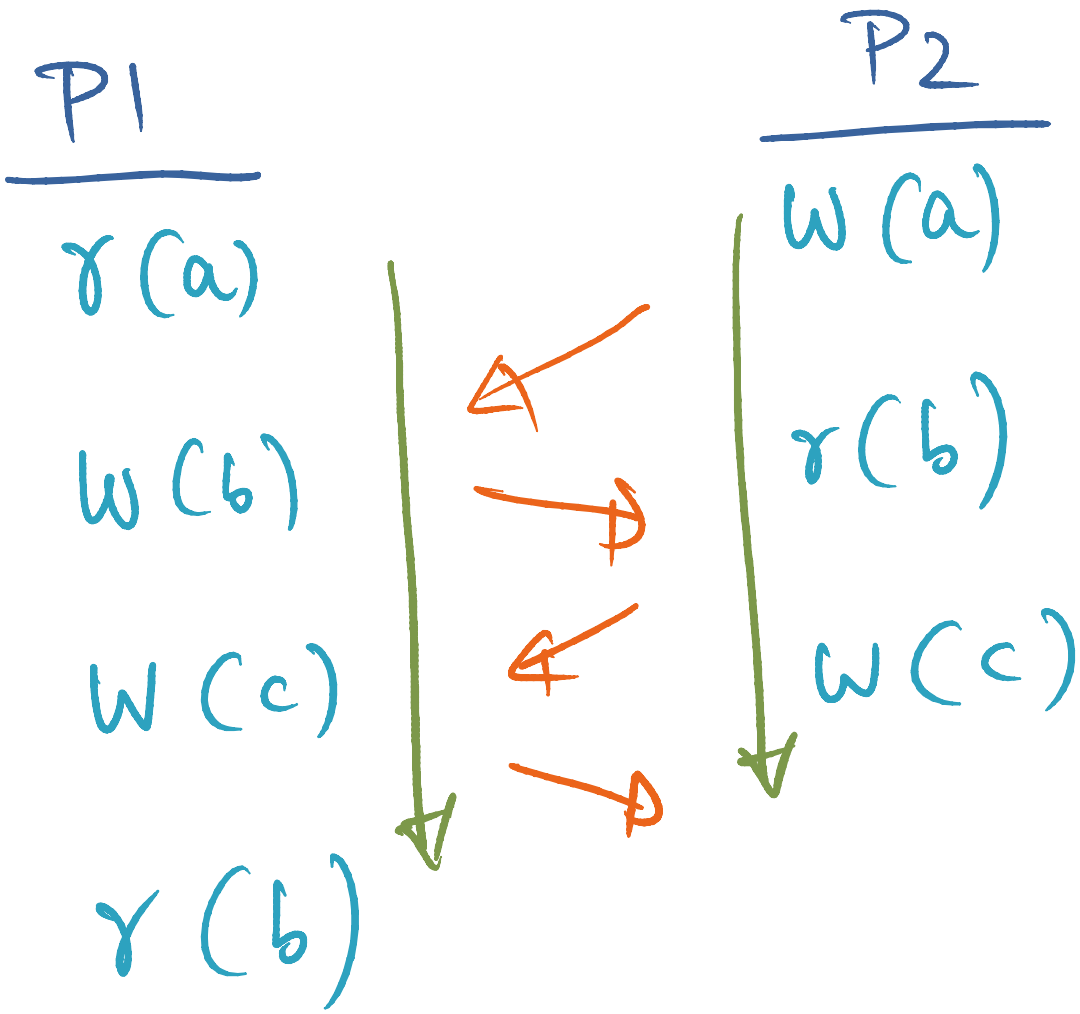
What is the model Presented to the Programmer?

Cache Coherence

↓

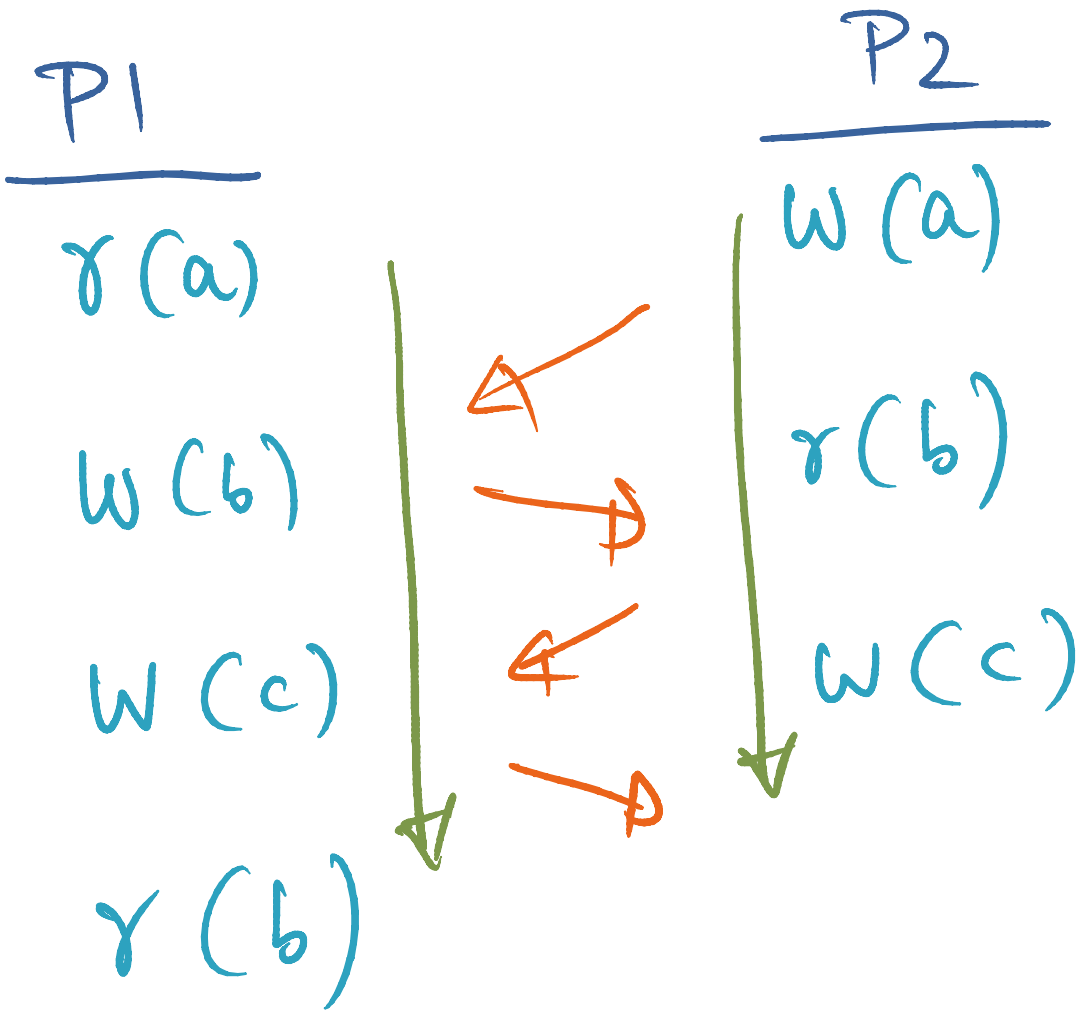
How is the System implementing the model in the presence of Private caches?

Sequential Consistency (SC)

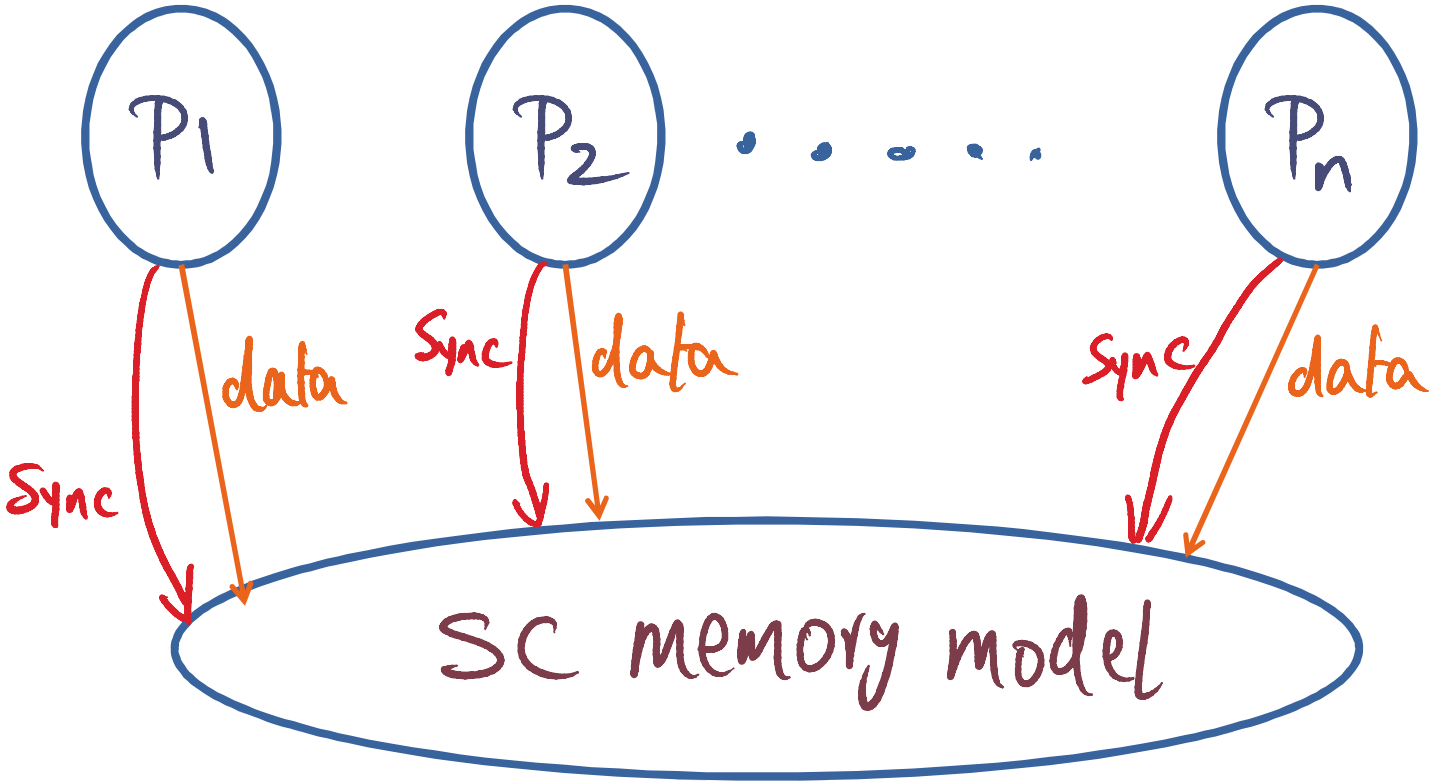


Program order + arbitrary interleaving

Sequential Consistency (SC)



Program order + arbitrary interleaving



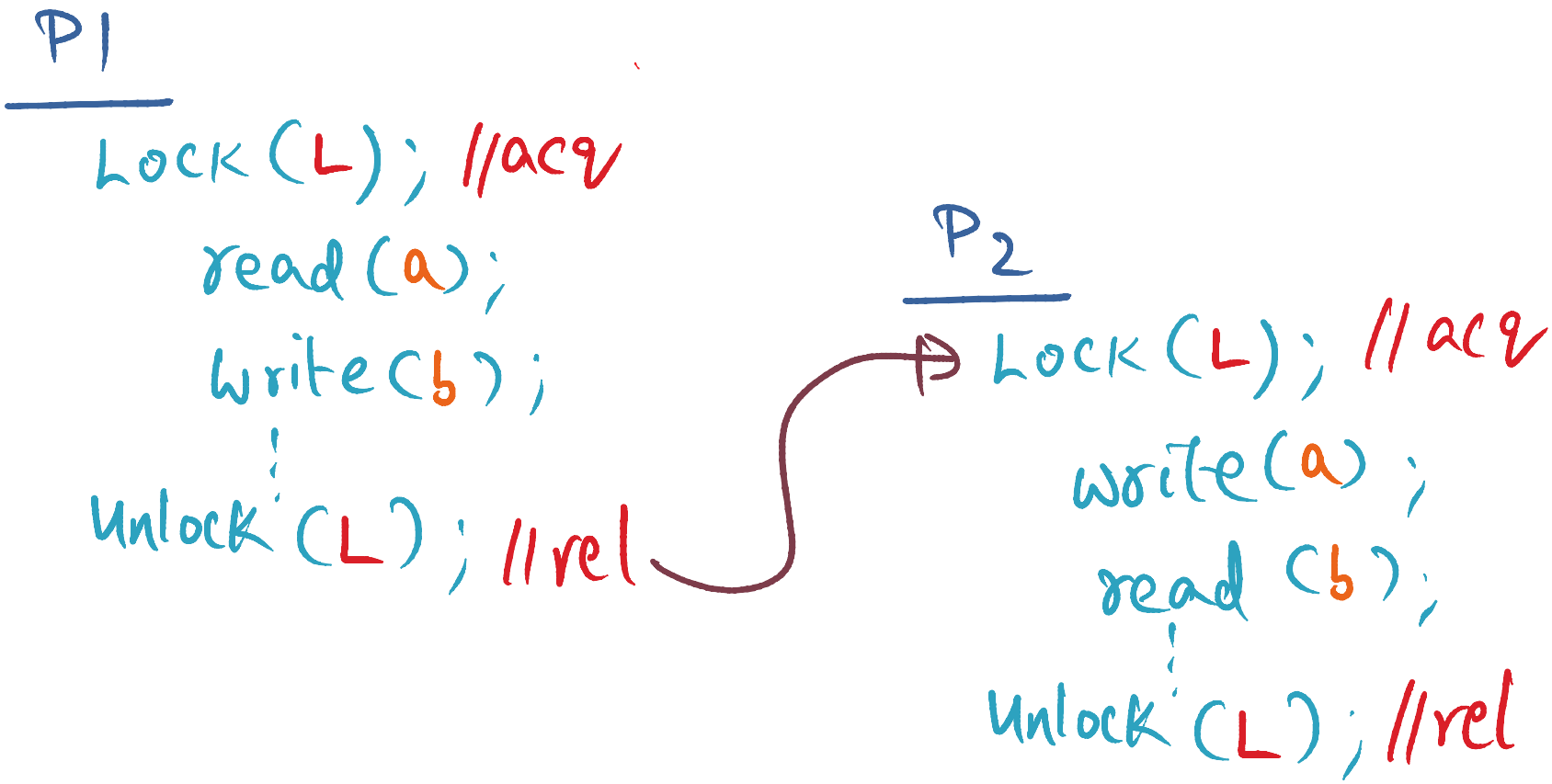
SC does not distinguish between

data r/w and Synchronization r/w

upshot?

Coherence action on every r/w access

Typical Parallel Program



Typical Parallel Program

P1

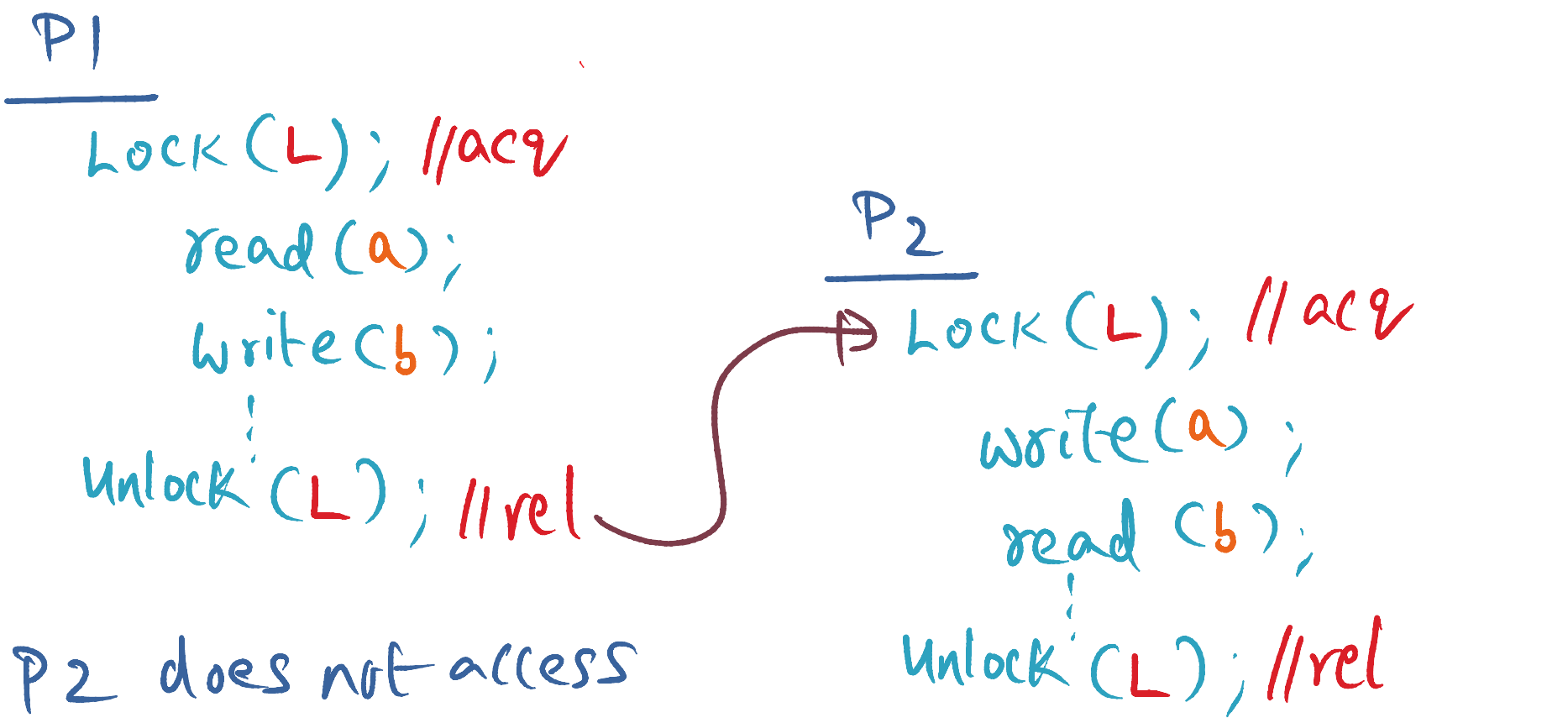
```
Lock(L); //acq
read(a);
write(b);
Unlock(L); //rel
```

P2

```
Lock(L); //acq
write(a);
read(b);
Unlock(L); //rel
```

P2 does not access data P1 releases L

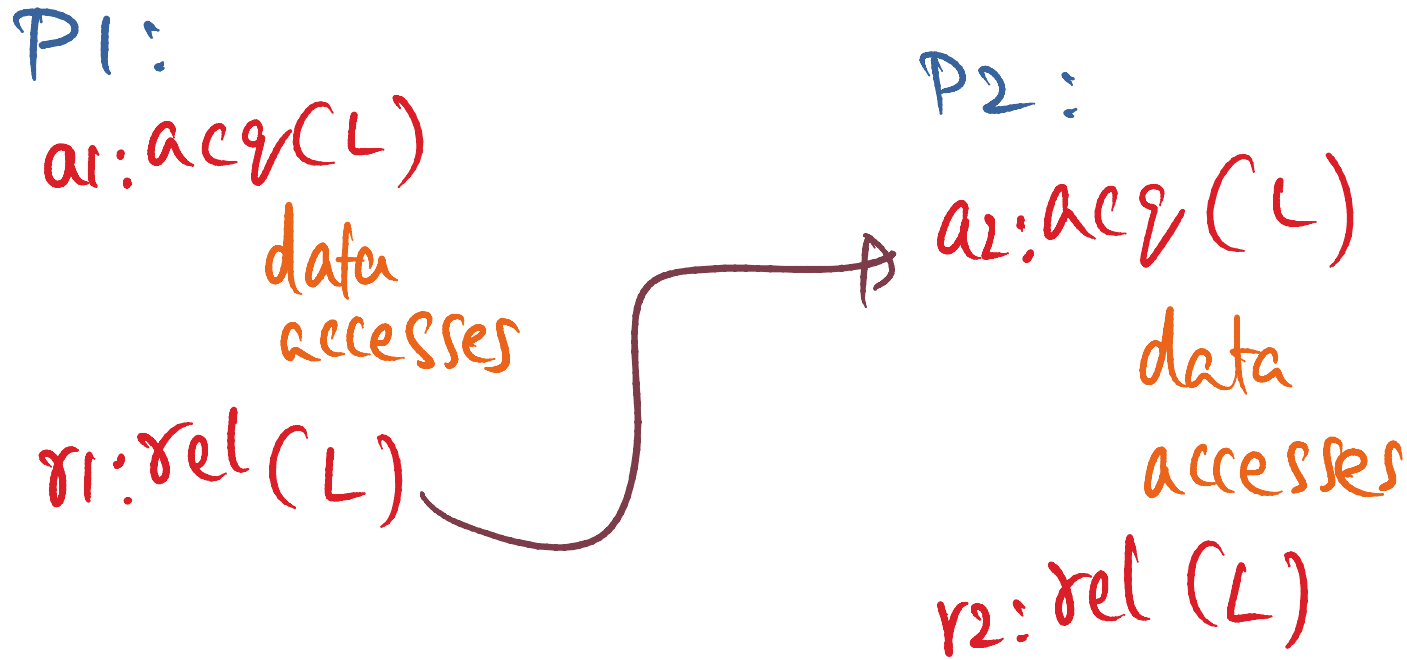
Typical Parallel Program



P2 does not access data P1 releases L

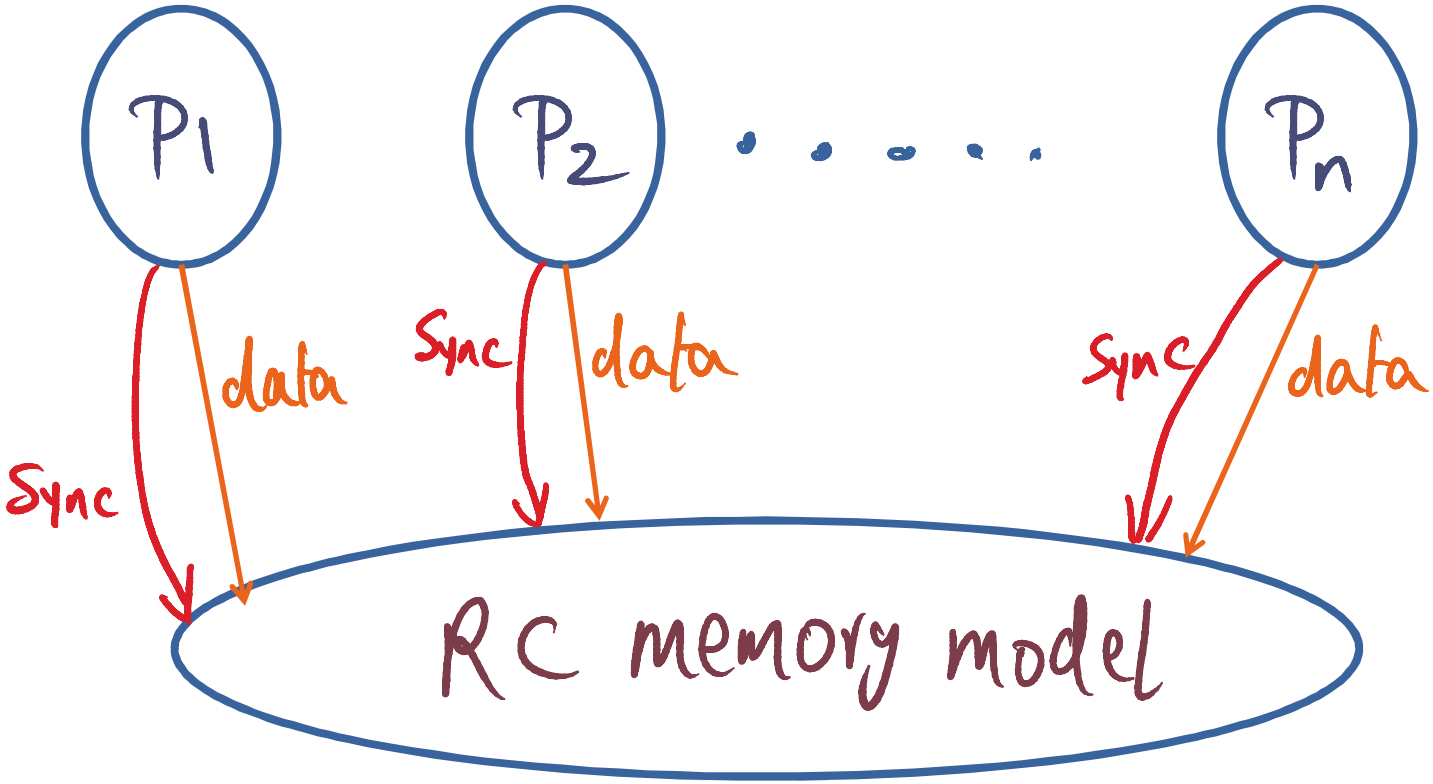
⇒ no need for coherence action for a and b until release

Release Consistency



If P1 : r1 $\xrightarrow{hb}$ P2 : a2

- All Coherence actions Prior to P1 : r1 should be complete befor P2 : a2



RC distinguishes between

data r/w and Synchronization r/w

upshot ?

Coherence action only when lock released

Example

Programmer's intent

P1

modify struct(A)

—X—

P2

wait for mod

use struct (A)

Example

Programmer's intent

P1

modify struct(A)

P2

wait for mod

use struct (A)

—X—

P2

```
Lock(L);
  While (flag==0)
P2 here ⟶ Wait (C,L);
Unlock(L);
use(A);
```

Example

Programmer's intent

P1

modify struct(A)

P2

wait for mod

use struct (A)

—X—

P1

```
modify(A);
```

P2

```
Lock(L);
  while(flag==0)
P2 here ⟶ wait(c,L);
Unlock(L);
use(A);
```

Example

Programmer's intent

P1

```
modify struct(A)
```

P2

```
wait for mod
use struct (A)
```

—X—

P1

```
modify(A);
Lock(L);
    flag = 1;
    signal(c);
unlock(L);
```

P2

```
Lock(L);
  while (flag == 0)
P2 here ⟶ wait (c, L);
unlock(L);
use(A);
```

Example

Programmer's intent

P1

modify struct(A)

P2

wait for mod
use struct (A)

—X—

P1

```
modify(A);
Lock(L);
    flag = 1;
    signal(C);
unlock(L);
```

P2

```
Lock(L);
  while (flag == 0)
P2 here ⟶ wait(C, L);
Unlock(L);
use(A);
```

Coherence actions complete

Example

Programmer's intent

P1

modify struct(A)

P2

wait for mod

use struct (A)

—X—

P1

```
modify(A);
Lock(L);
    flag = 1;
    signal(c);
unlock(L);
```

P2

```
Lock(L);
  while (flag == 0)
P2 here ⟶ wait(c, L);
Unlock(L);
use(A);
```

Coherence actions complete

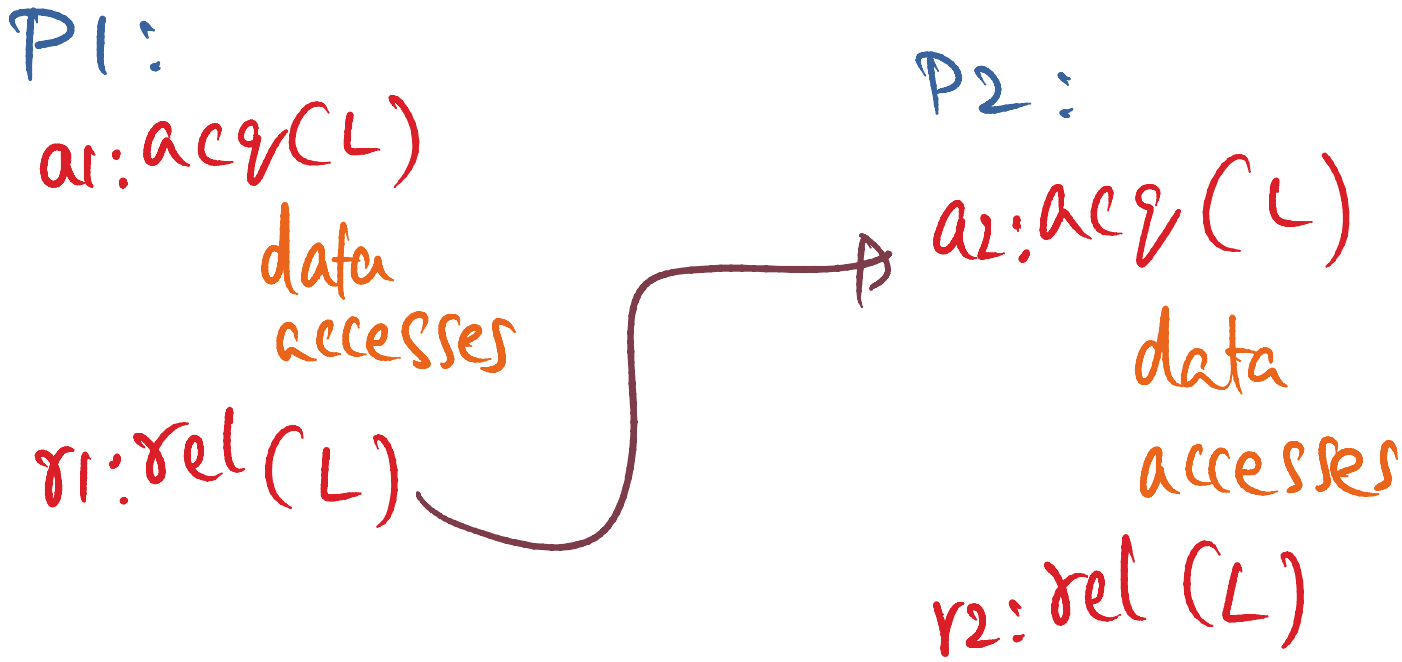
Advantage of RC over SC

- no waiting for coherence actions on every memory access

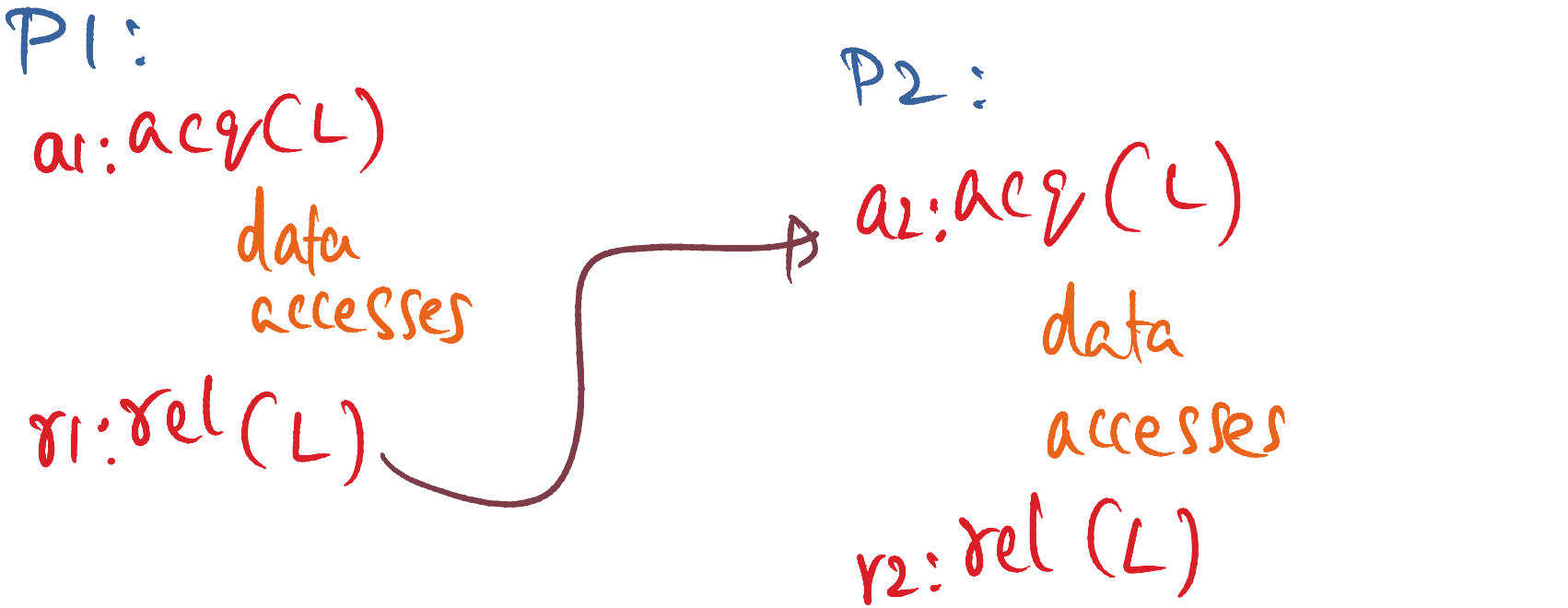
  ⇒ overlap Computation with Communication

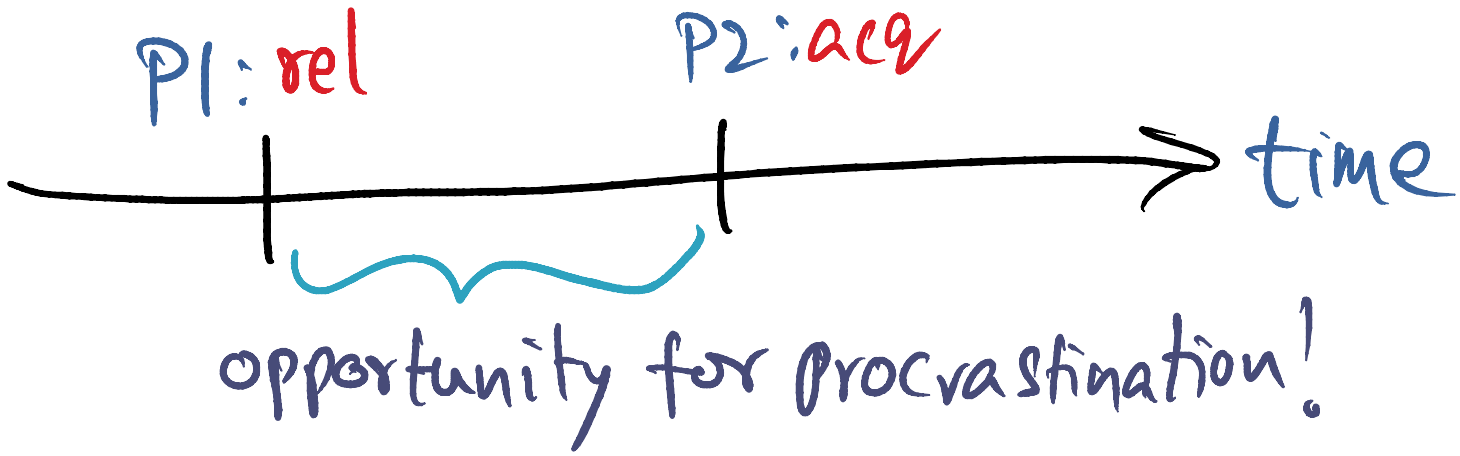
  ⇒ better performance for RC over SC

Lazy RC

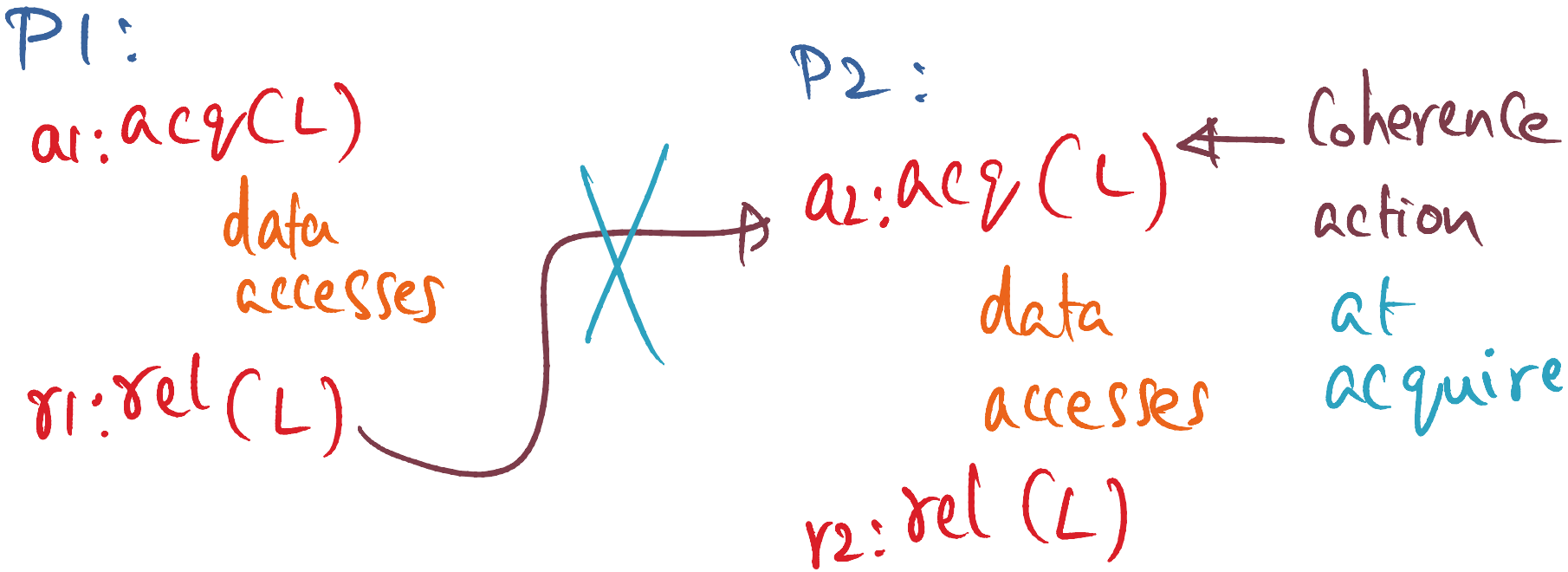

P1:
a1: acq(L)
data
accesses
r1: rel(L)
P2:
a2: acq(L)
data
accesses
r2: rel(L)

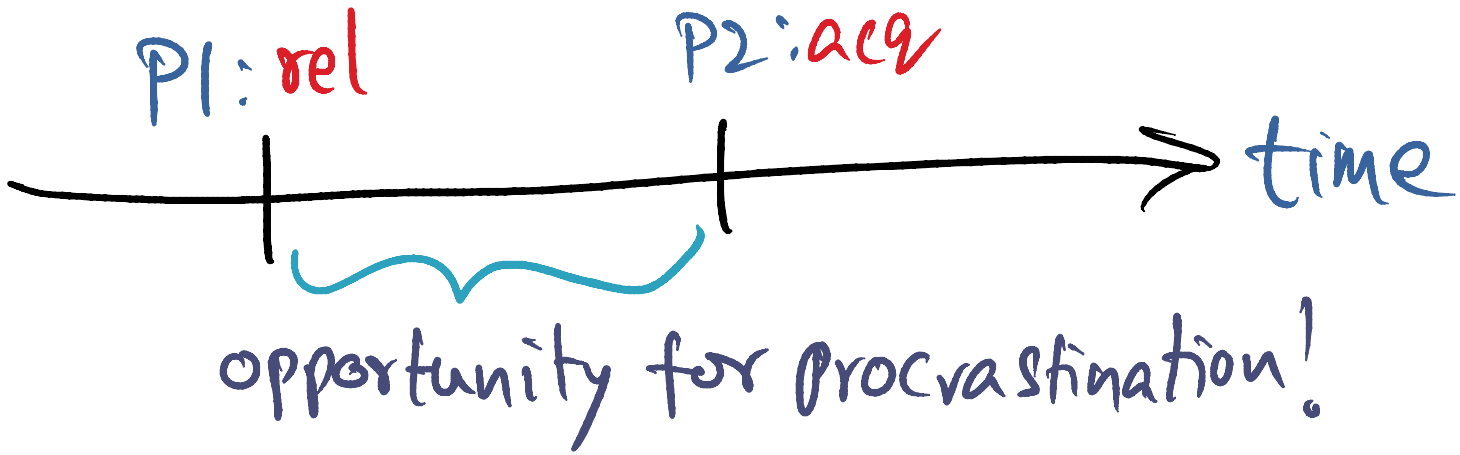
Lazy RC


P1:
a1: acq(L)
data accesses
r1: rel(L)
P2:
a2: acq(L)
data accesses
r2: rel(L)

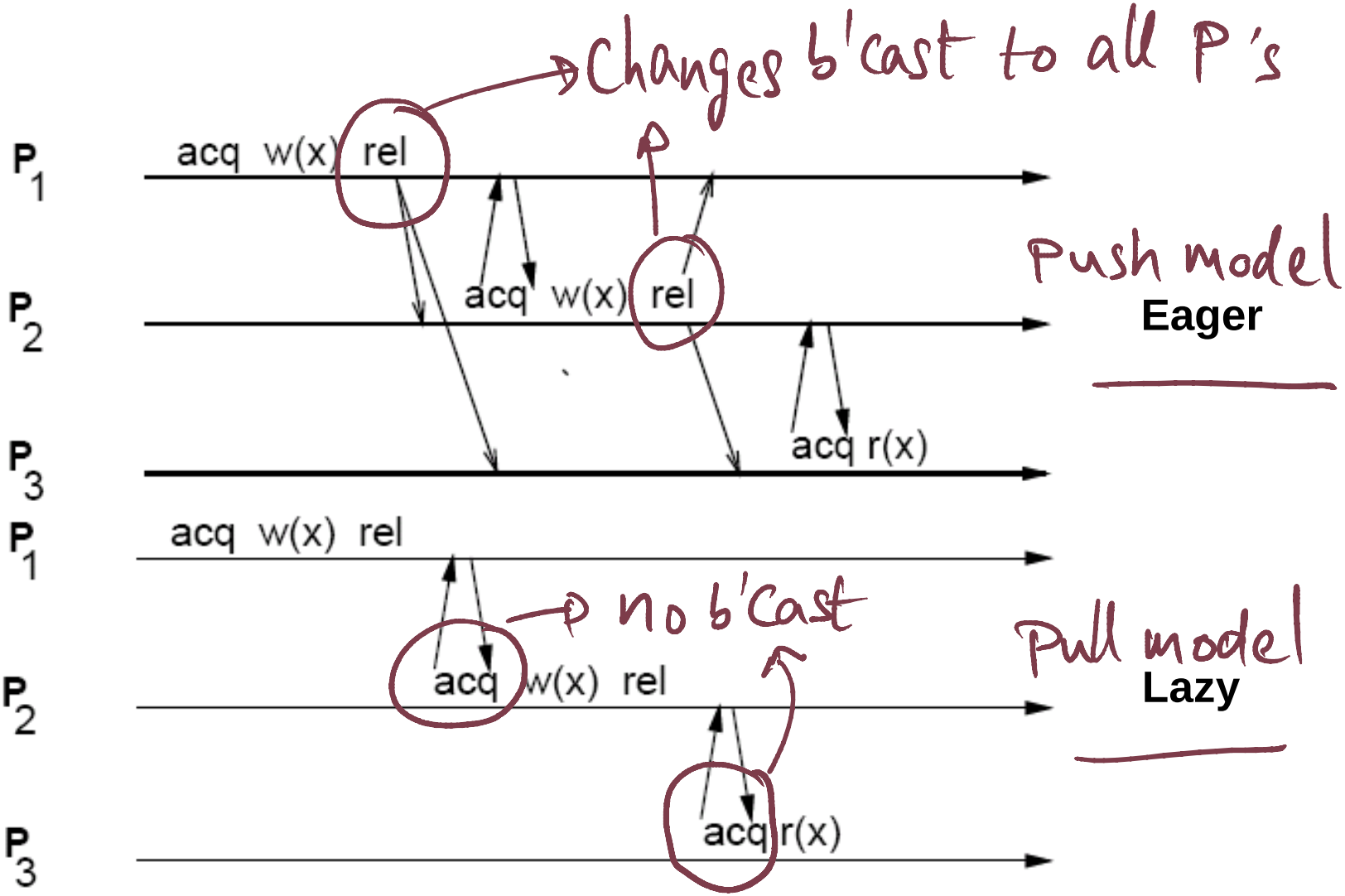

P1: rel
P2: acq
time
opportunity for procrastination!

Lazy RC


P1:
a1: acq(L)
data accesses
r1: rel(L)
P2:
a2: acq(L)
data accesses
r2: rel(L)
Coherence action at acquire


P1: rel
P2: acq
time
opportunity for procrastination!

Eager Vs. Lazy RC



What are the Pros of Lazy over Eager?

What are the Cons of Lazy over Eager?

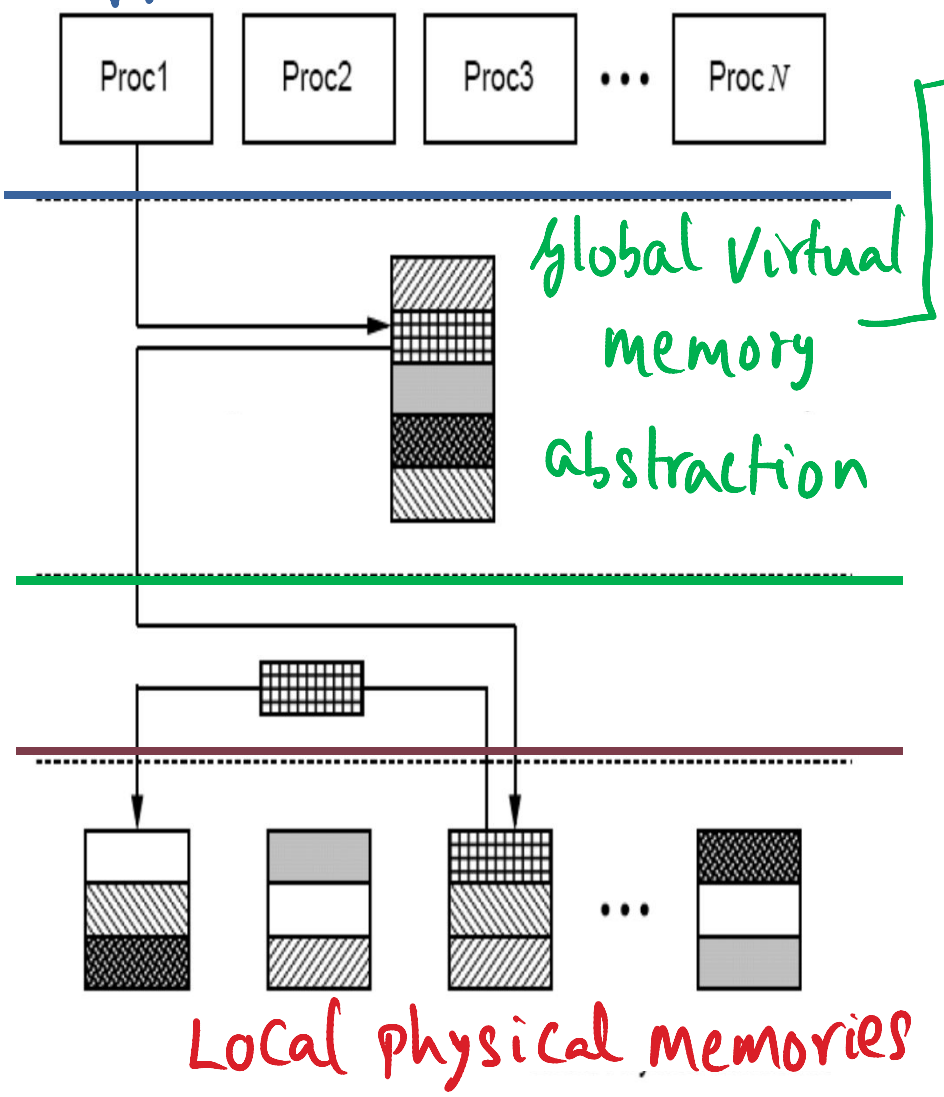
What are the Pros of Lazy over Eager?

Less messages

What are the Cons of Lazy over Eager?

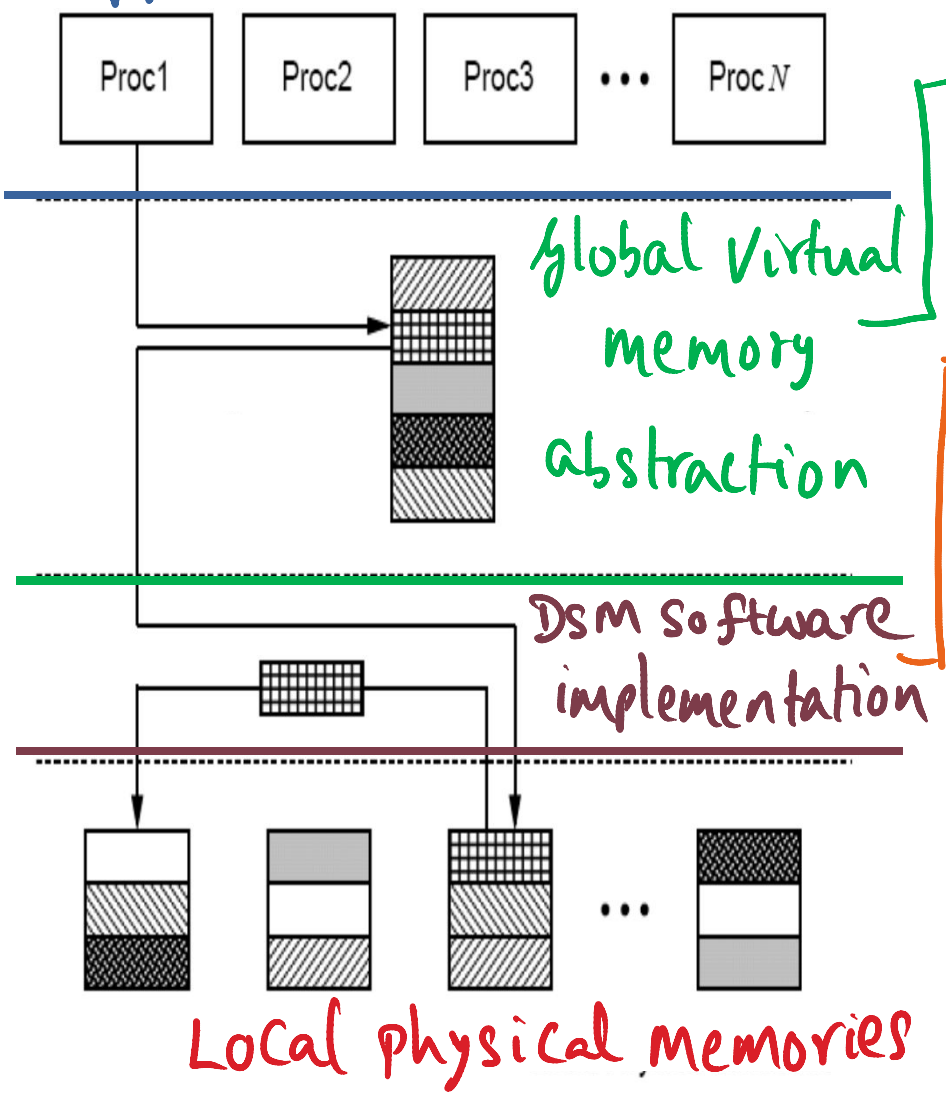
More latency at acquire

Software DSM



- Address space partitioned
- Address equivalence
- Distributed ownership

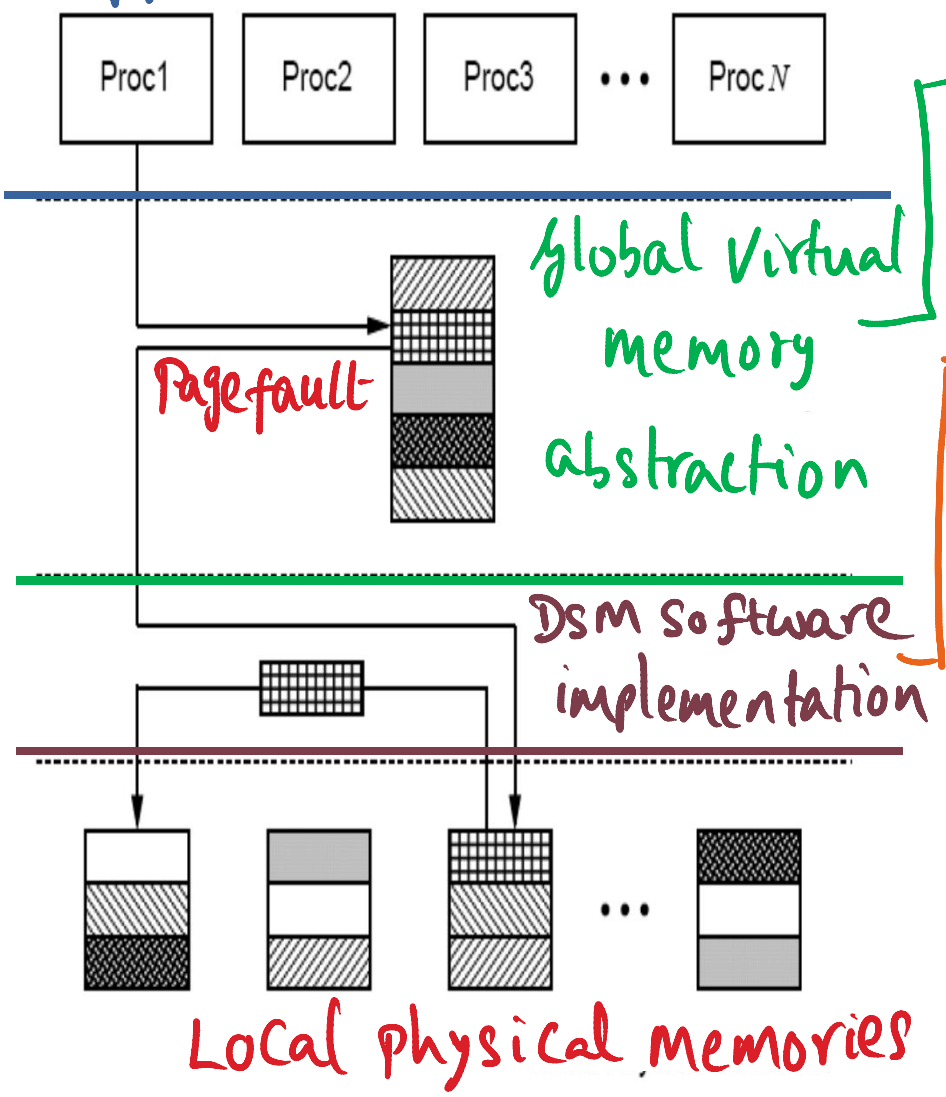
Software DSM



- Address space partitioned
- Address equivalence
- Distributed ownership

→ Contact owner of page to get current copy

Software DSM

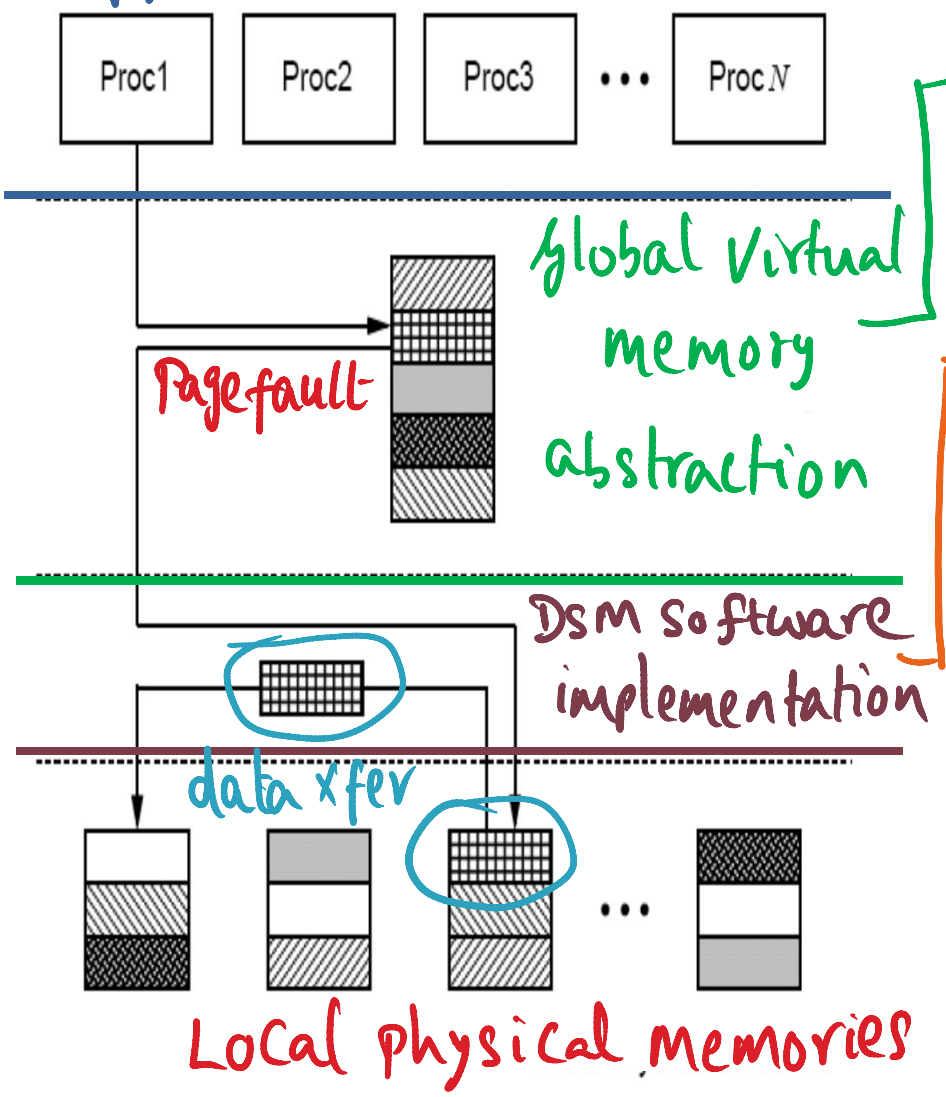


Address space partitioned
Address equivalence
Distributed ownership

Contact owner of page to get current copy

pf → DSM software → !! →

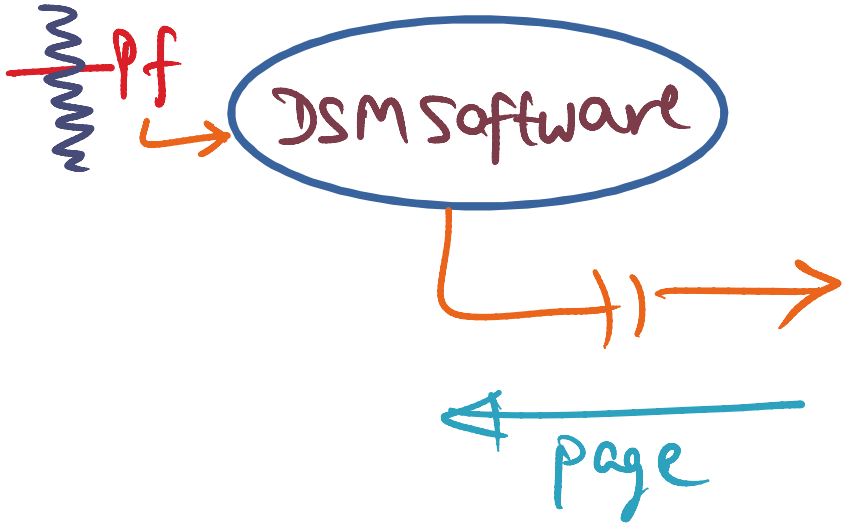
Software DSM



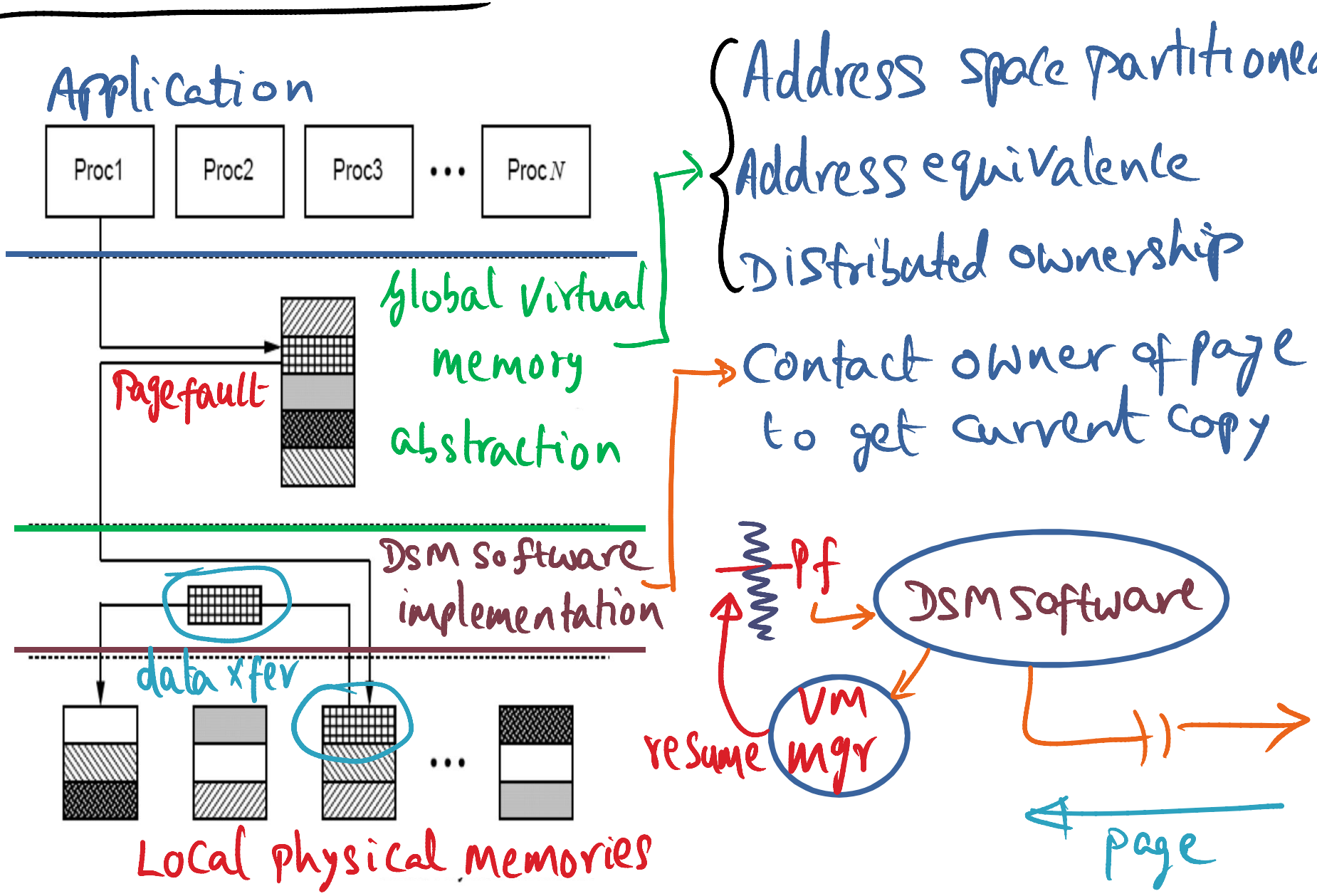
- Address space partitioned
- Address equivalence
- Distributed ownership

Global virtual memory abstraction

Contact owner of page to get current copy



Software DSM



LRC With Multi-Writer Coherence Protocol

P1

Lock(L);

x ←
y ←
z ←
} X, Y, Z pages Modified in C.S ⇒ X_d, Y_d, Z_d } diffs

Unlock(L);

LRC With Multi-Writer Coherence Protocol

P1

Lock(L);

x ←
y ←
z ←
} X, Y, Z pages Modified in C.S ⇒ X_d, Y_d, Z_d } diffs

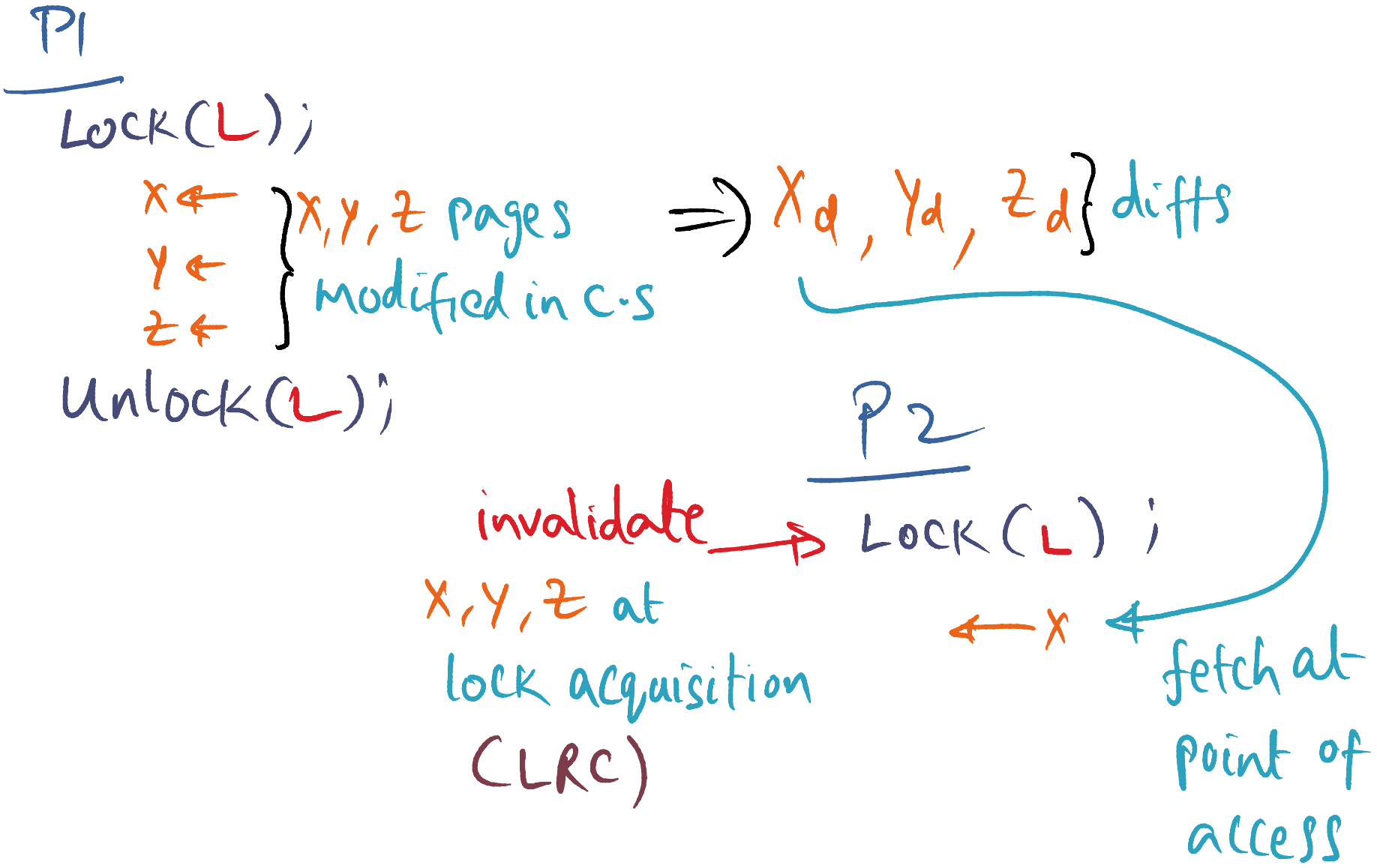
Unlock(L);

P2

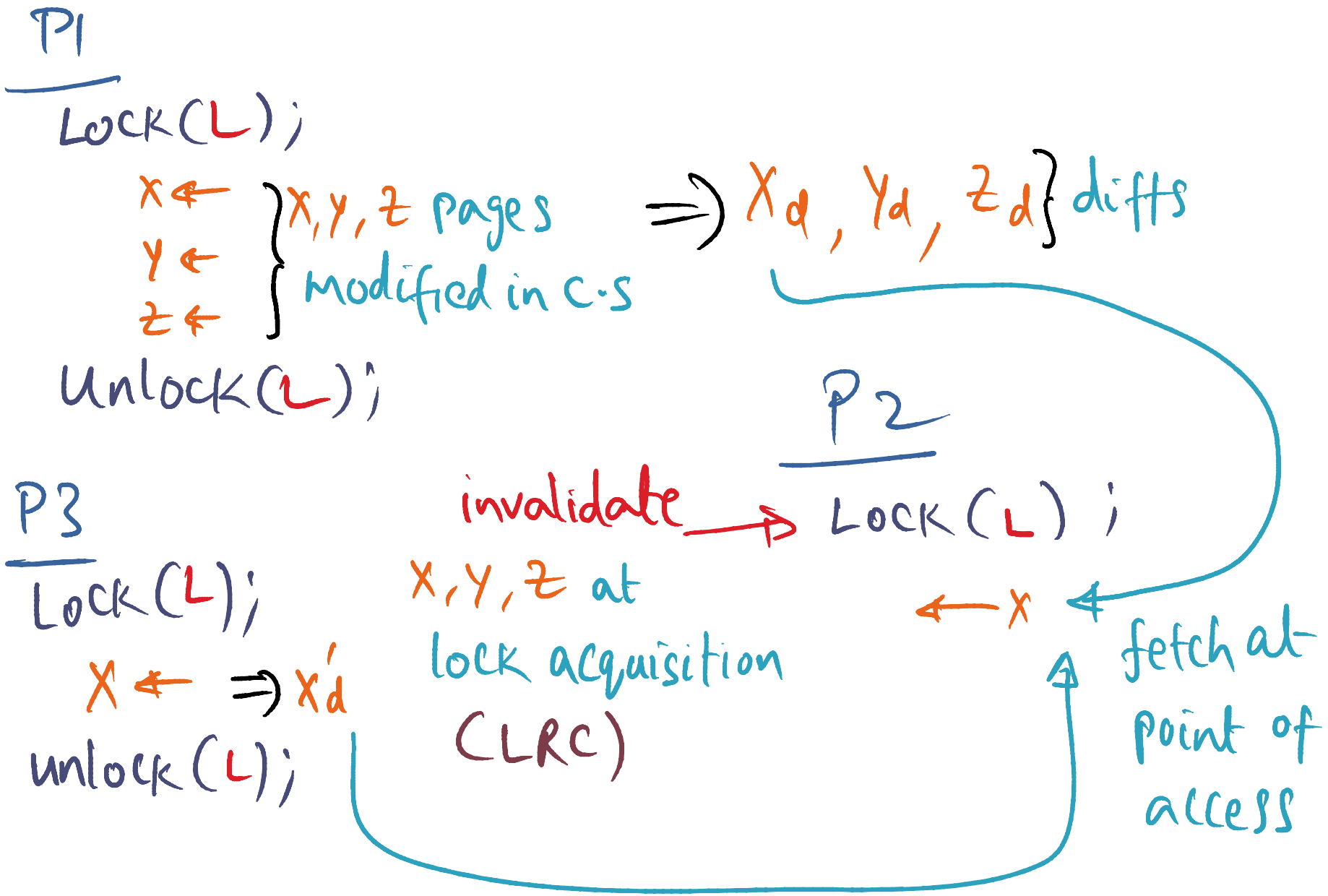
Lock(L);

invalidate → X, Y, Z at lock acquisition (LRC)

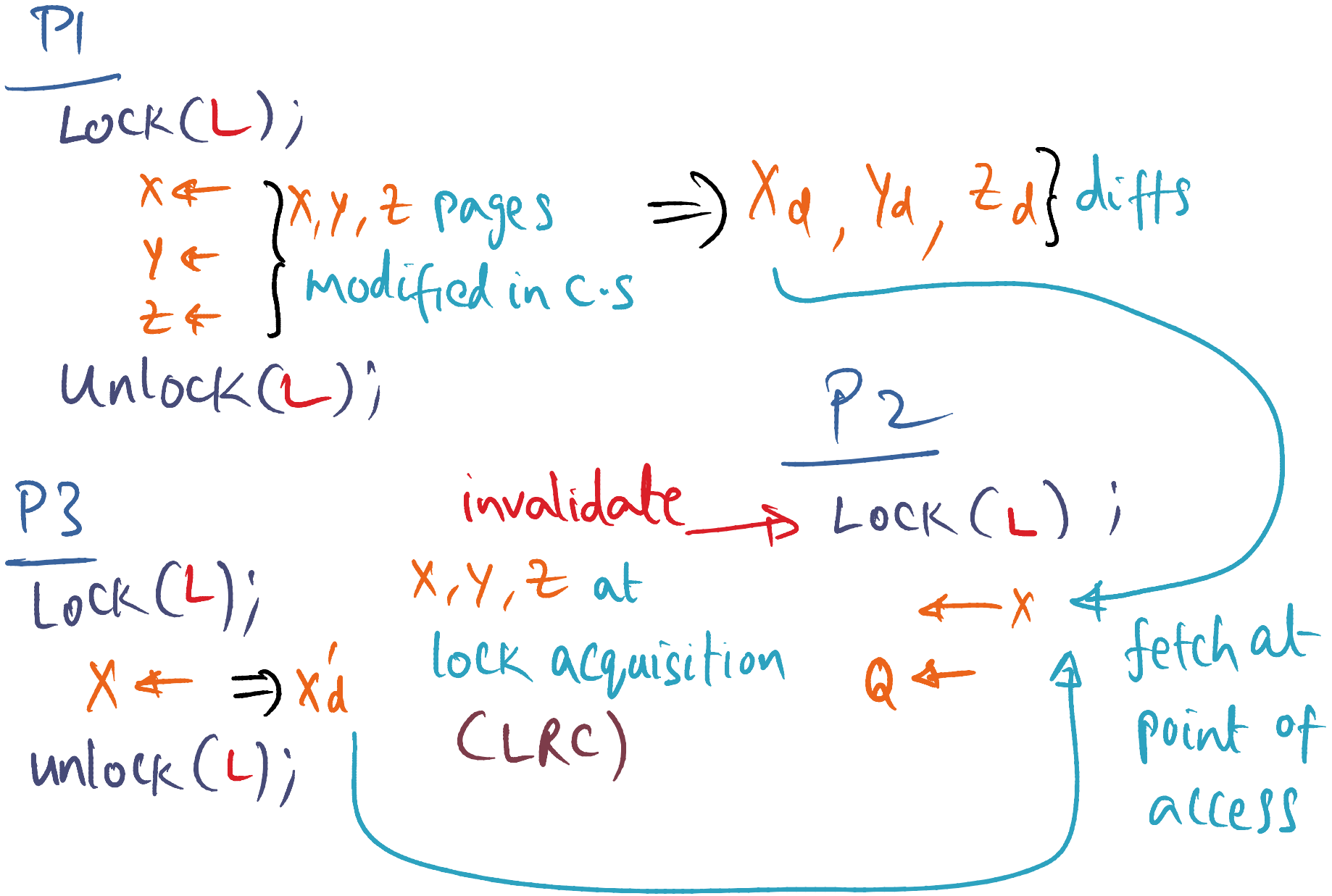
LRC With Multi-Writer Coherence Protocol



LRC With Multi-Writer Coherence Protocol



LRC With Multi-Writer Coherence Protocol



Software DSM