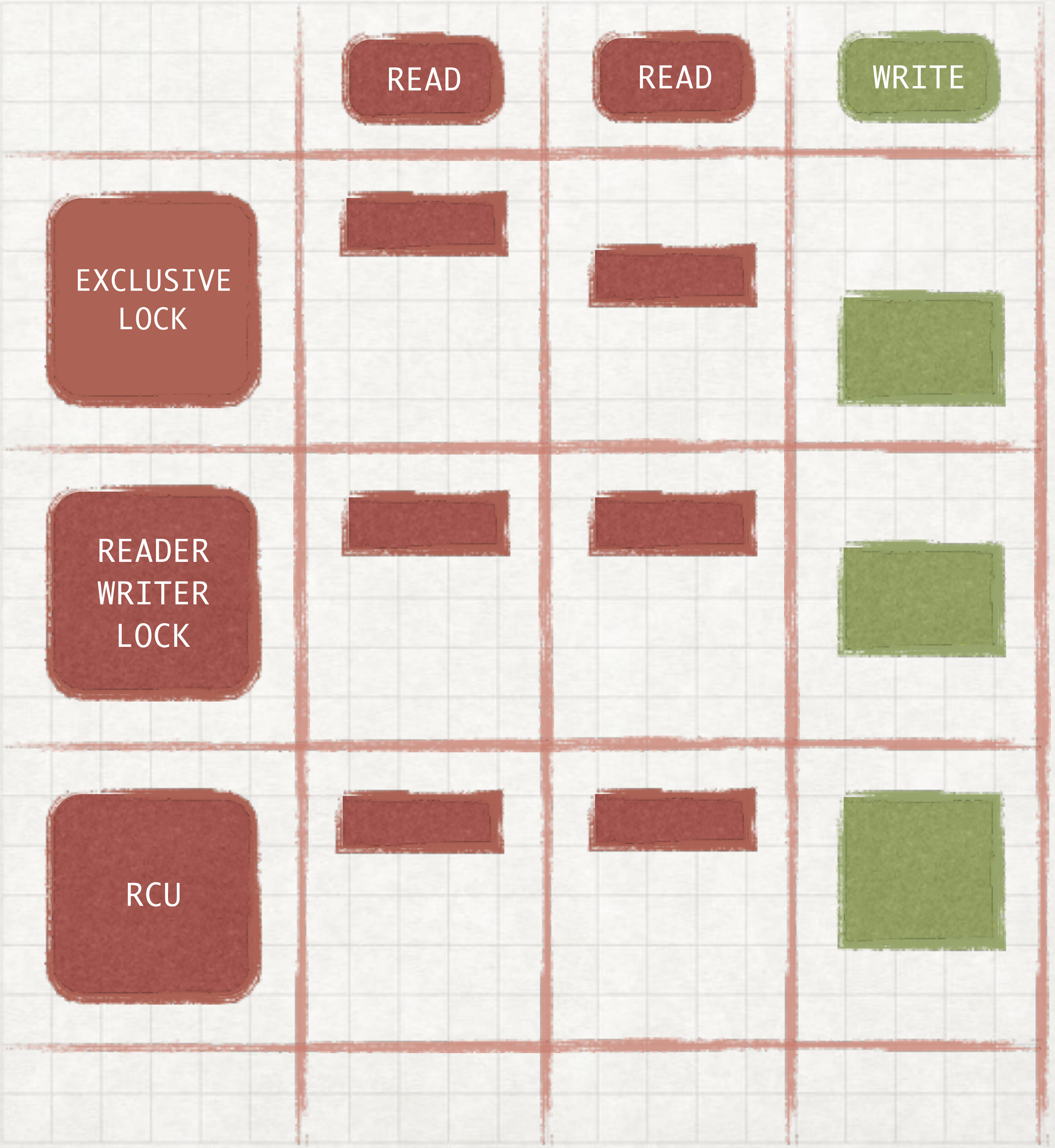
AN RCU WITH LOW SYNC LATENCY

PRCU

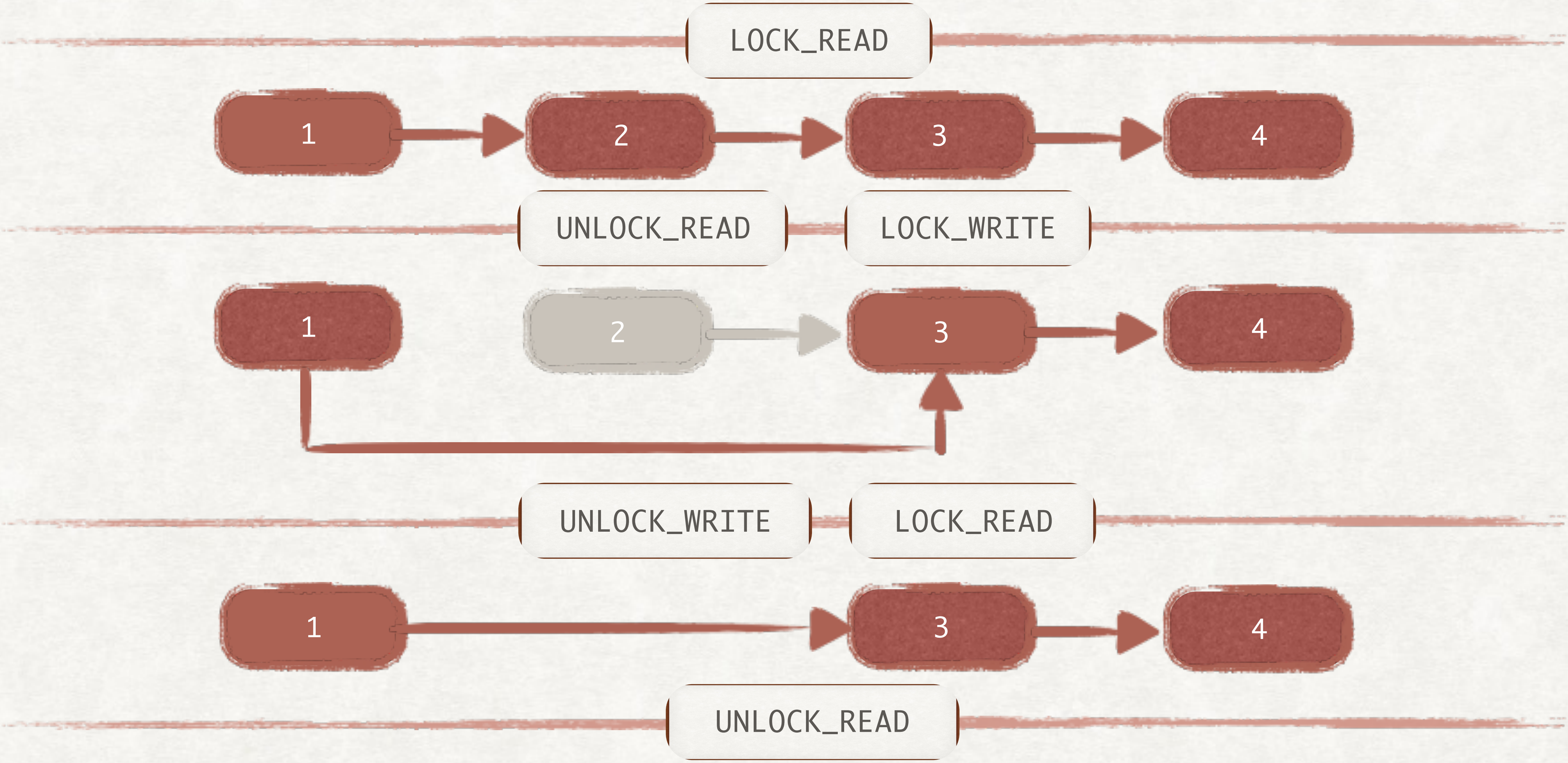
张恒
ZHANG HENG

WHAT IS RCU

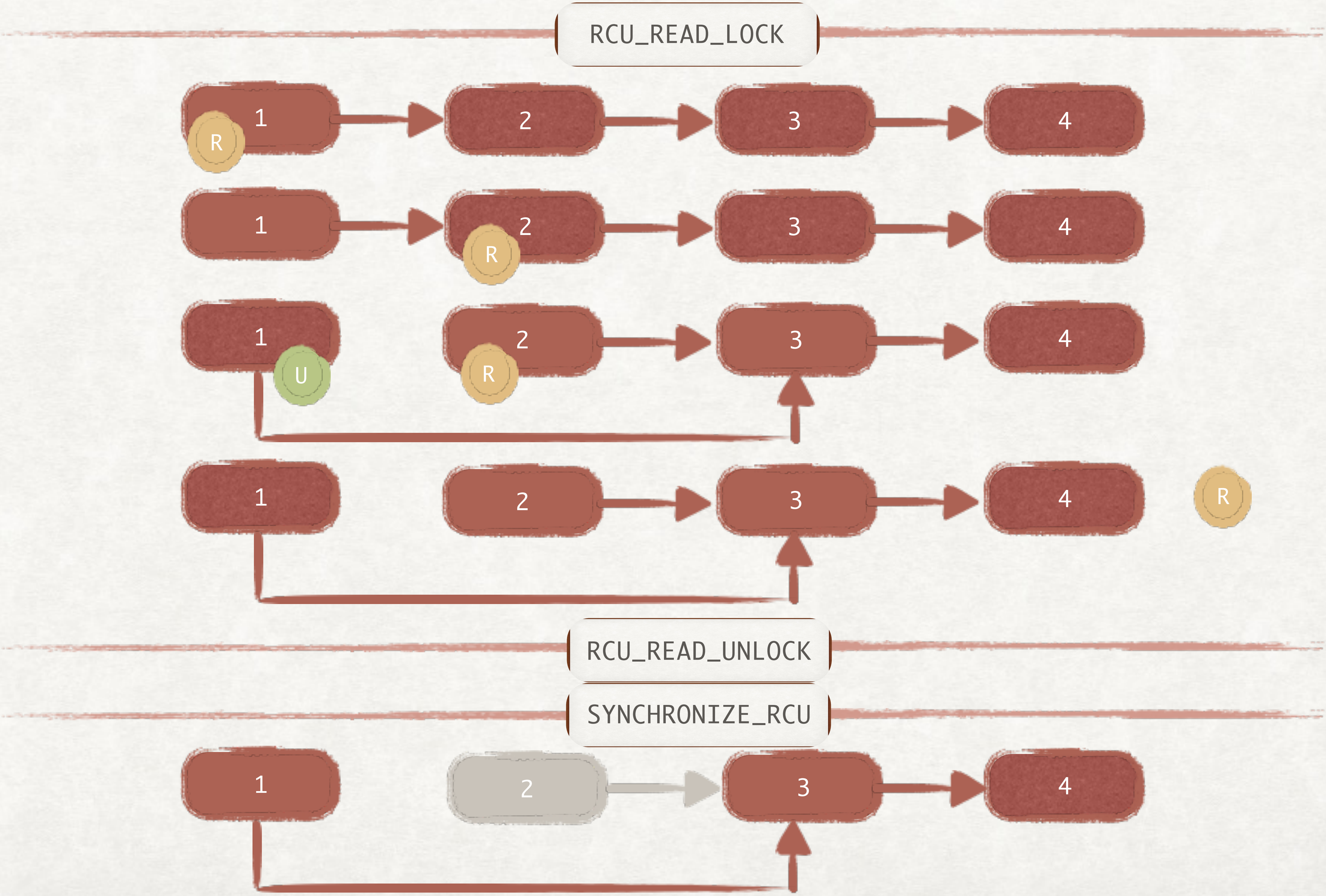
- Read-Copy Update
- A synchronization primitive which allows readers and writers execute concurrently
- Great for read-mostly data
- Two phase:
  - update on a new copy
  - reclaim the old copy (when it is safe)



LIST WITH RWLOCK



LIST WITH RCU



RCU_READ_LOCK

RCU_READ_UNLOCK

SYNCHRONIZE_RCU

RWLOCK OR RCU

SAME PROBLEM

- Reader-Writer Locks
  - when writers can enter CS

- RCU
  - when updaters can reclaim the resource

ENSURE ALL PREVIOUS READERS HAVE LEFT

RWLOCK OR RCU

DIFFERENT ATTITUDE

- Reader-Writer Locks
  - different algorithms for different workloads, e.g. rwsem, brlock …
- RCU
  - extremely low overhead on read side (fastpath)
  - extremely high overhead on write side

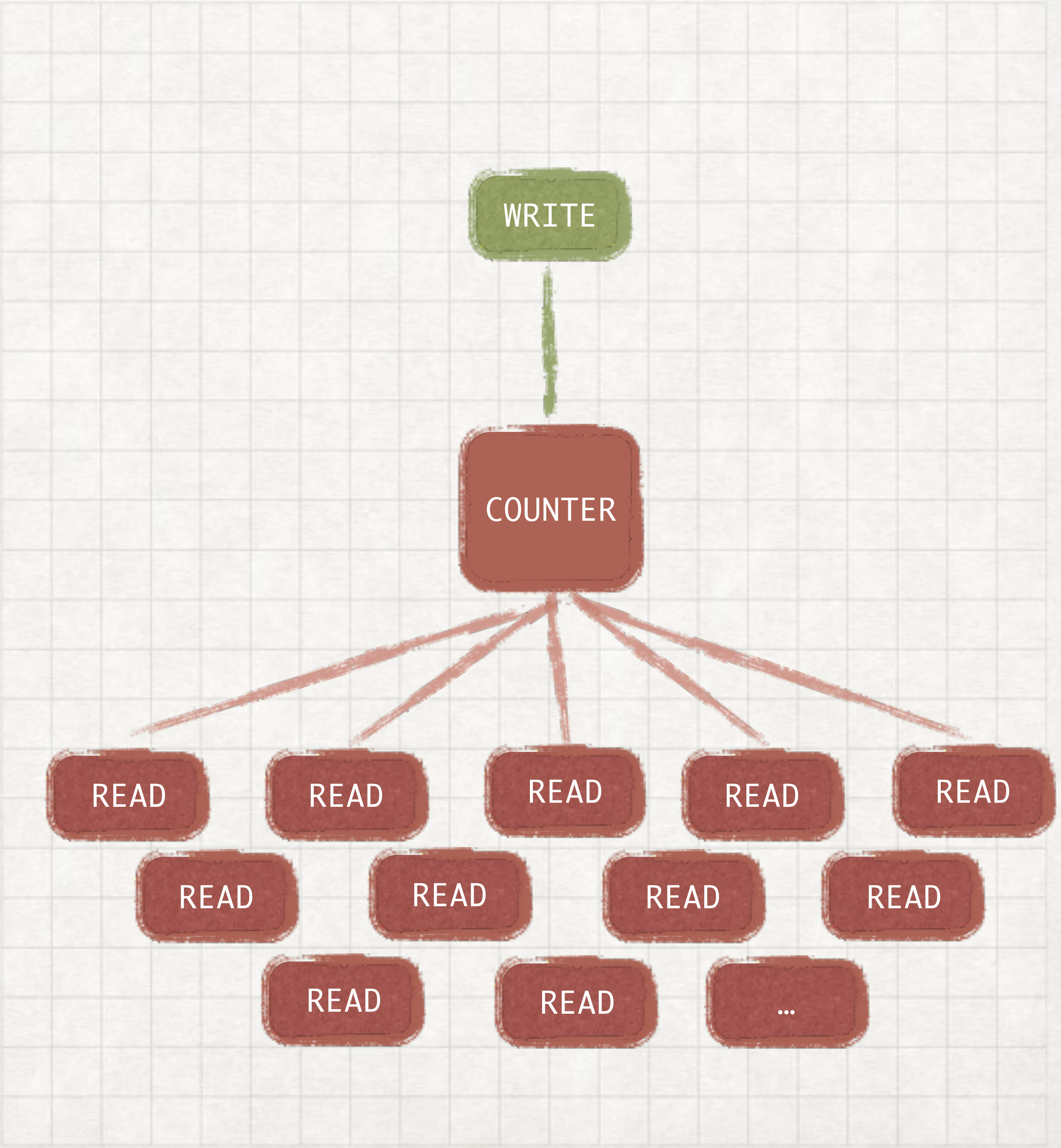
RWLOCK OR RCU

TRADEOFF

		READER	WRITER
RWSEM		HEAVY CONTENTION	LIGHTWEIGHT
CSNZI	COHORT LOCK	LIGHT CONTENTION	MORE ATOMIC OPS
BRLOCK		NO CONTENTION	MUCH ATOMIC OPS
PRWLOCK	PERCPU-RWSEM	NO FENCE	IPI
PRCU		NO BLOCKING	IPI
RCU		NO BLOCKING	IPI

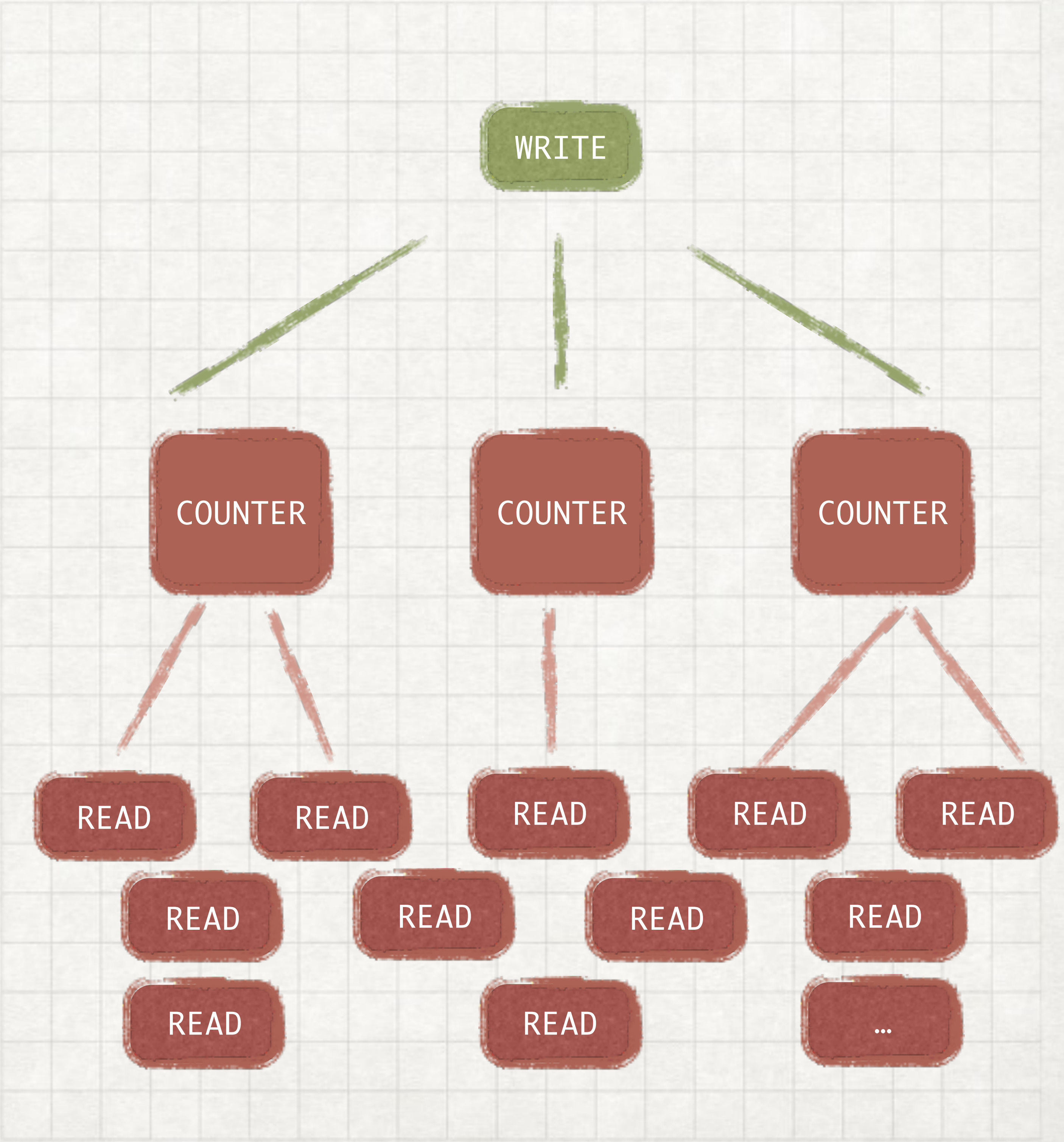
USING ATOMIC OPS

- Ref Counter - Single
  - R: heavy contention
  - W: check one counter



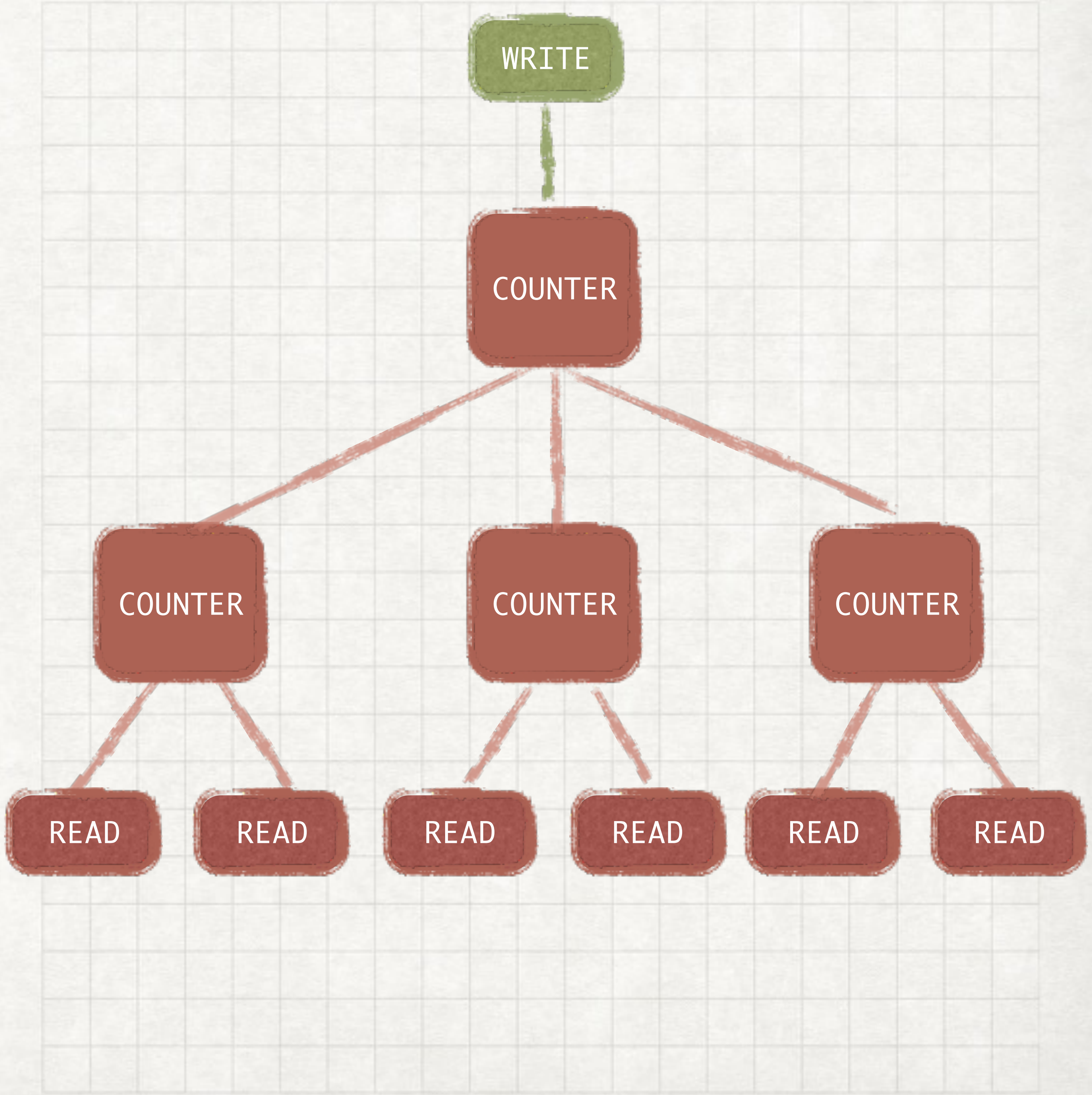
USING ATOMIC OPS

- Ref Counter - Multiple
  - R: contention reduced
  - W: check more counters



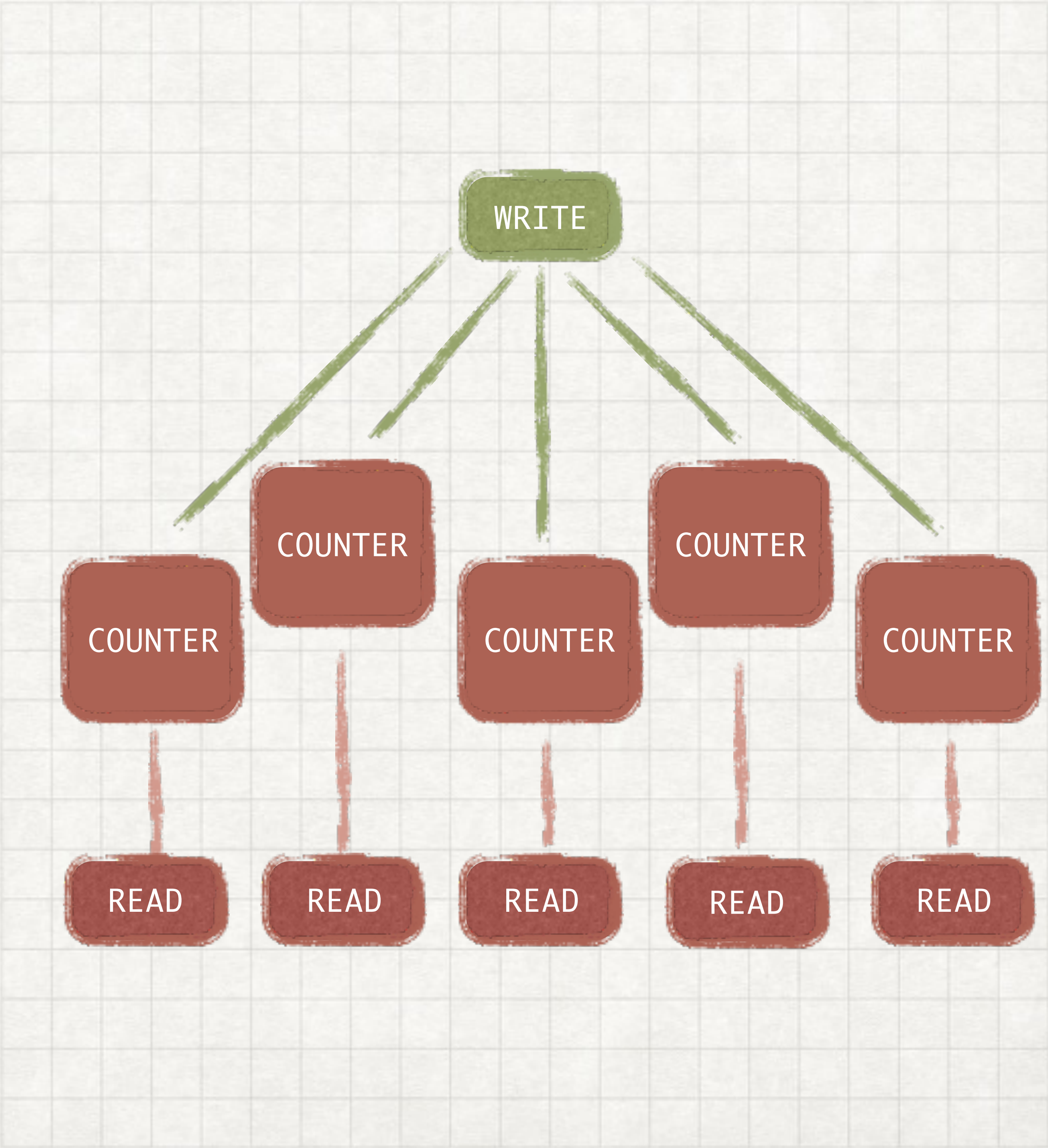
USING ATOMIC OPS

- CSNZI - Hierarchical
  - R: contention reduced, may increase latency
  - W: check one counter



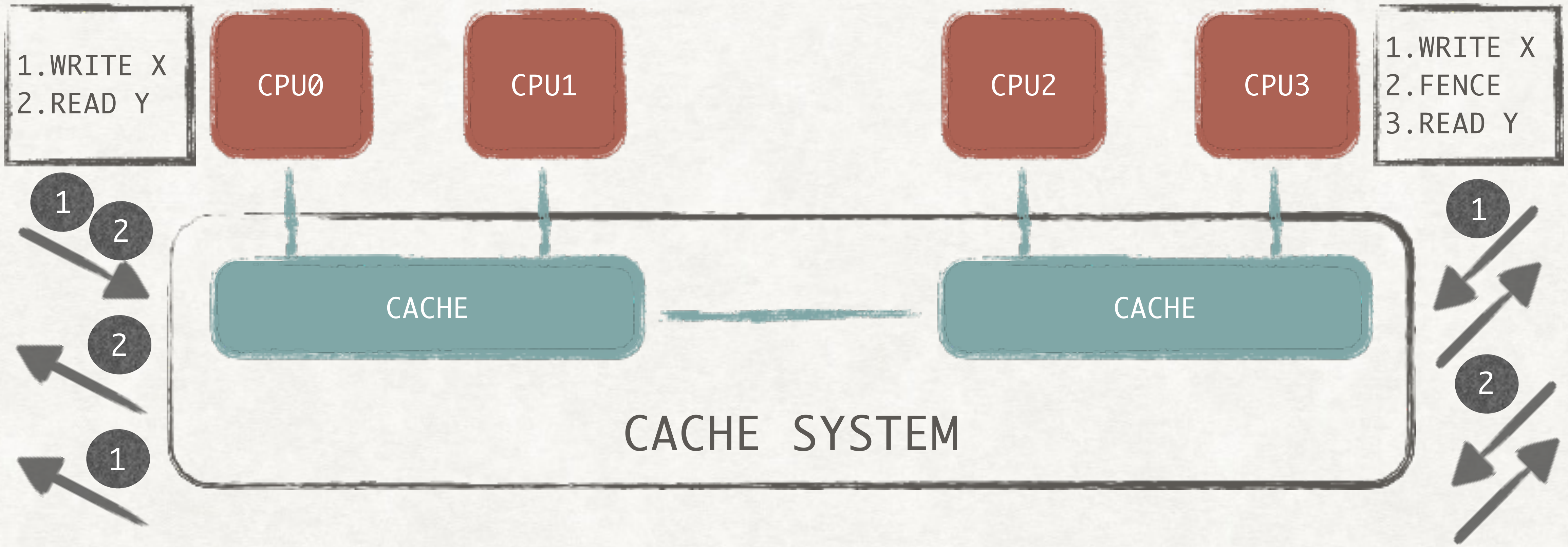
USING ATOMIC OPS

- Ref Counter - 1:1
  - R: No contention
  - W: check all counters



REMOVE FENCE

WHAT IS FENCE



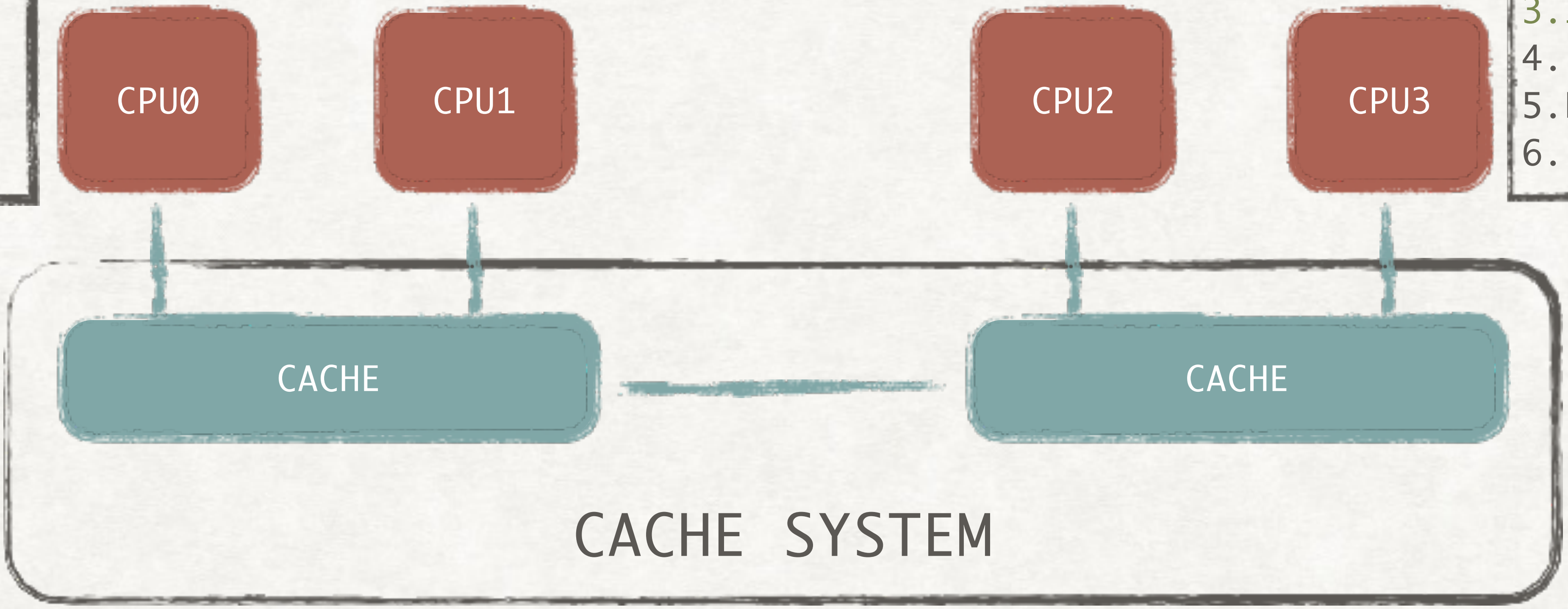
1.WRITE X
2.READ Y

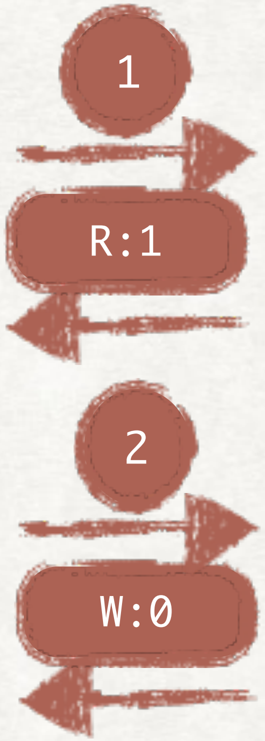
1.WRITE X
2.FENCE
3.READ Y

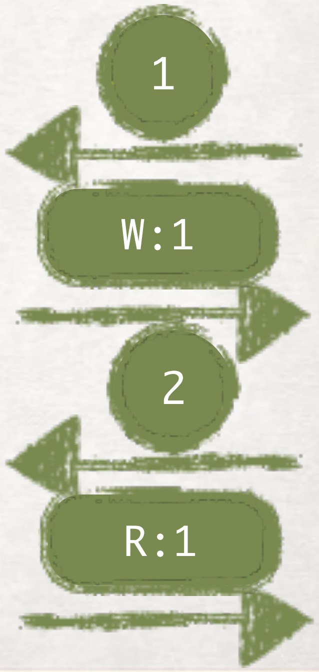
LOCK WITH FENCE

```
R = 1
FENCE;
IF (W == 0)
    ENTER CS
ELSE
    WAIT
```

```
W = 1;
FENCE;
IF (R == 0)
    ENTER CS
ELSE
    …
```





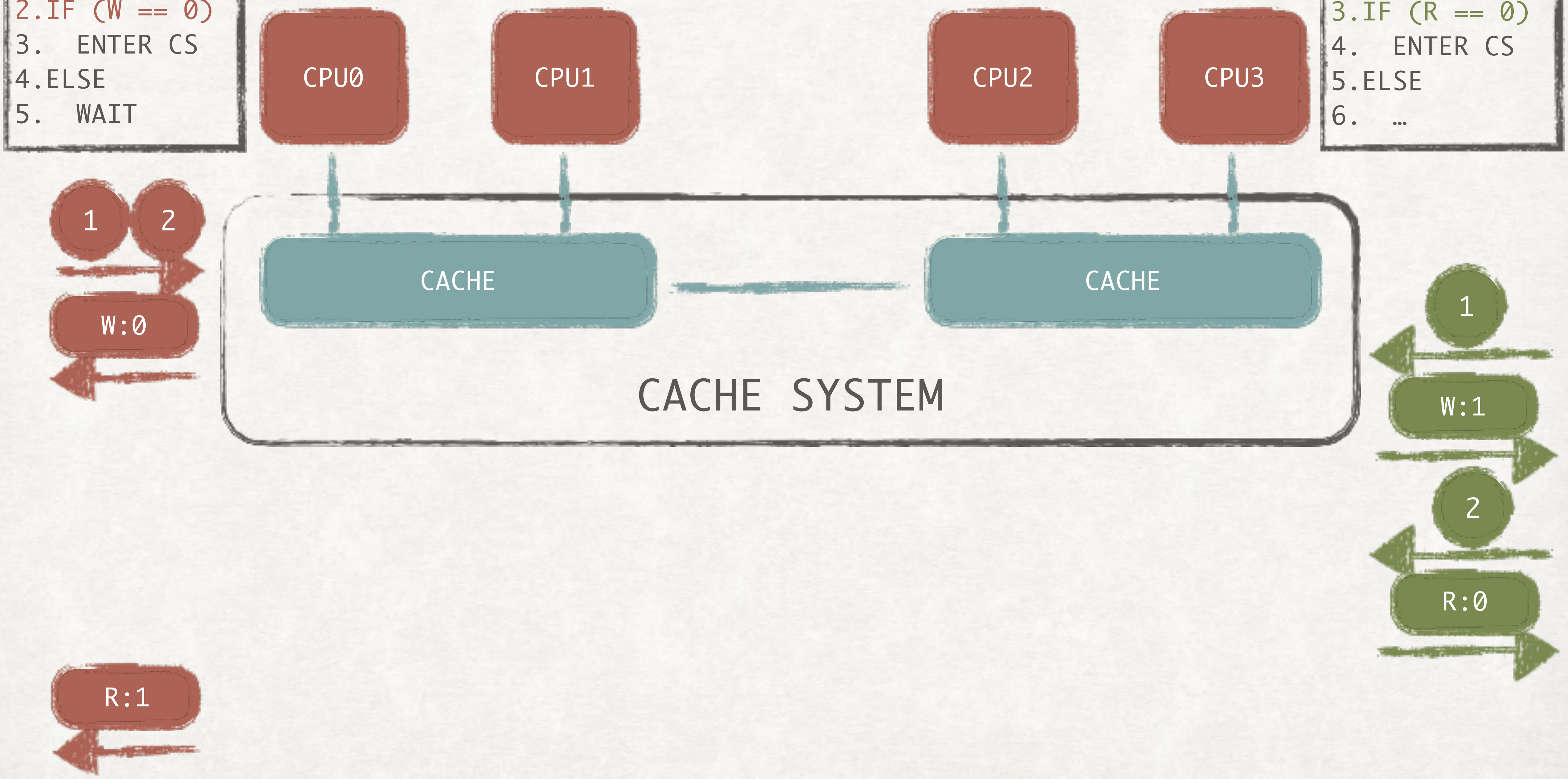


PROBLEM WITHOUT FENCE

INCONSISTENCY

```
1.R = 1
2.IF (W == 0)
3.   ENTER CS
4.ELSE
5.   WAIT
```

```
1.W = 1;
2.FENCE;
3.IF (R == 0)
4.   ENTER CS
5.ELSE
6.   …
```

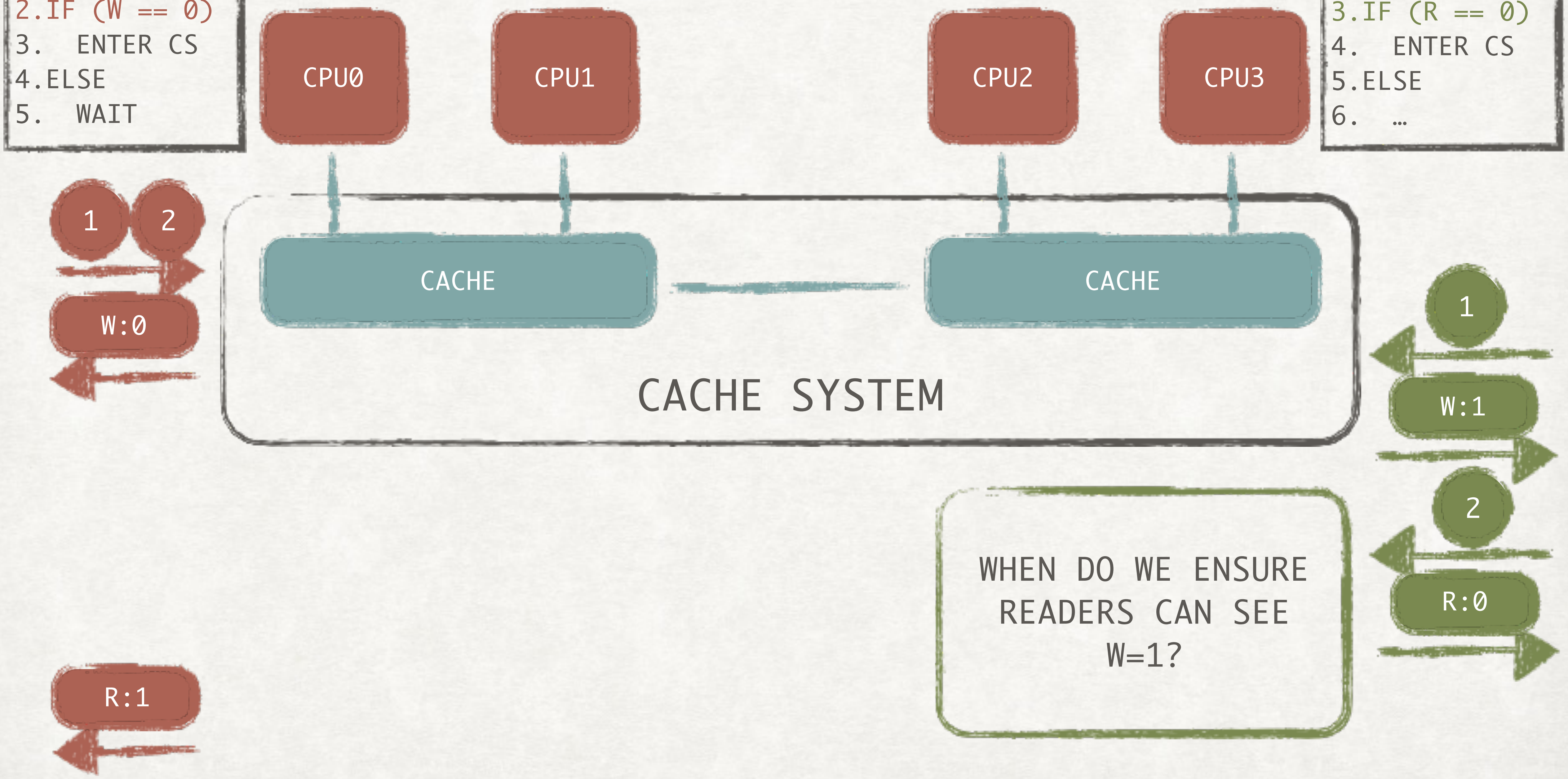


PROBLEM WITHOUT FENCE

INCONSISTENCY

```
1.R = 1
2.IF (W == 0)
3.   ENTER CS
4.ELSE
5.   WAIT
```

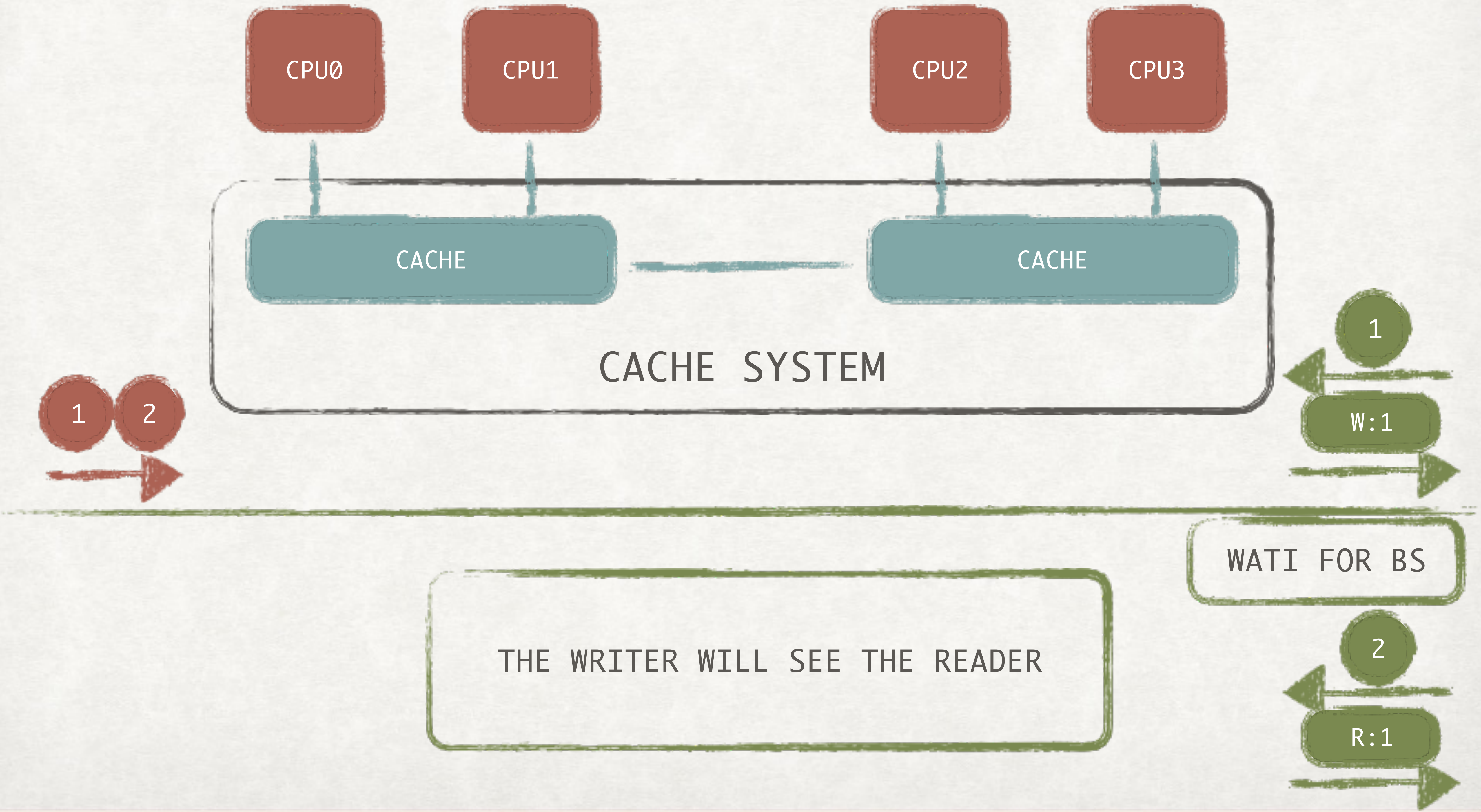
```
1.W = 1;
2.FENCE;
3.IF (R == 0)
4.   ENTER CS
5.ELSE
6.   …
```



WHEN DO WE ENSURE READERS CAN SEE W=1?

THINKING ON HARDWARE I

BOUNDED STALENESS



CPU0

CPU1

CPU2

CPU3

CACHE

CACHE

CACHE SYSTEM

1

2

1

W:1

WATI FOR BS

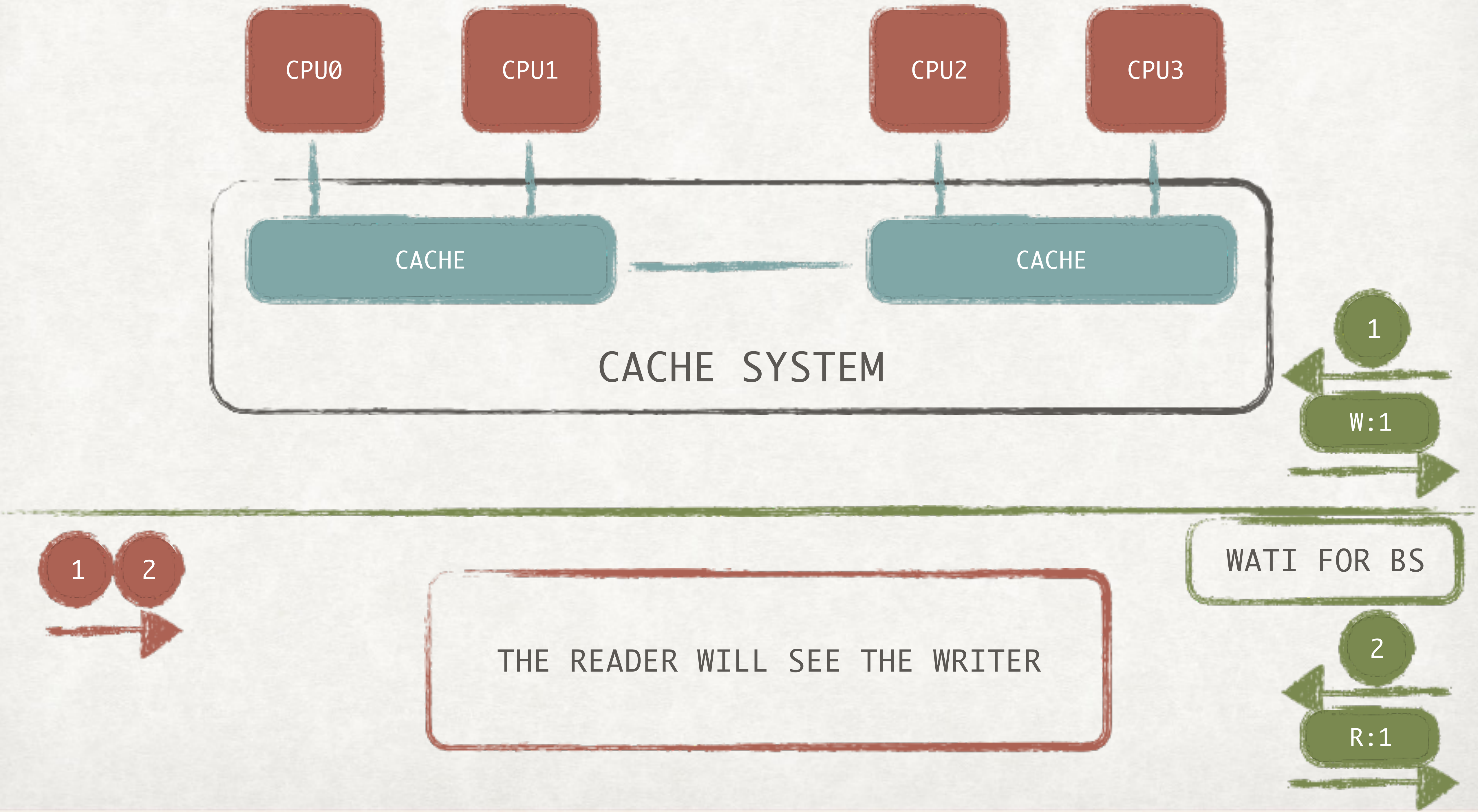
2

R:1

THE WRITER WILL SEE THE READER

THINKING ON HARDWARE I

BOUNDED STALENESS



THE READER WILL SEE THE WRITER

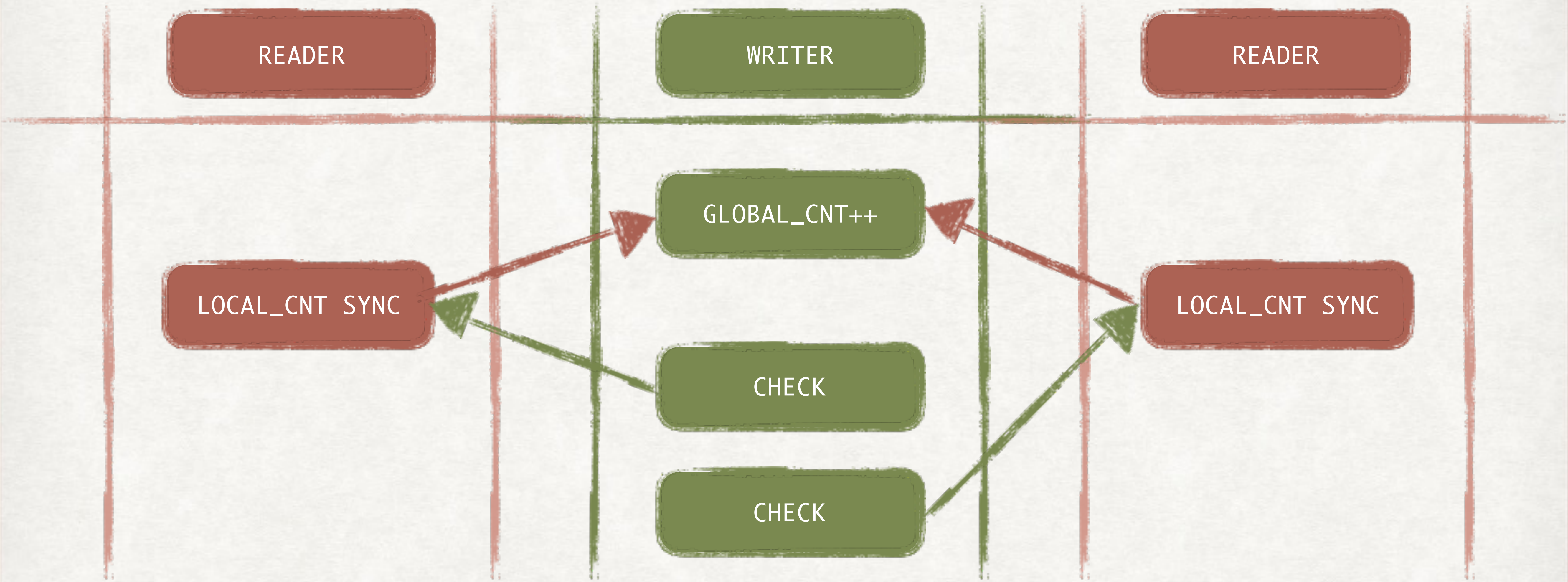
THINKING ON HARDWARE II

BIASED FENCE

- Target : the buffer of issued instructions
- Light fence:
  - Async or Passively report the buffer info
- Heavy fence:
  - Wait for remote cores' previously issued instructions committing

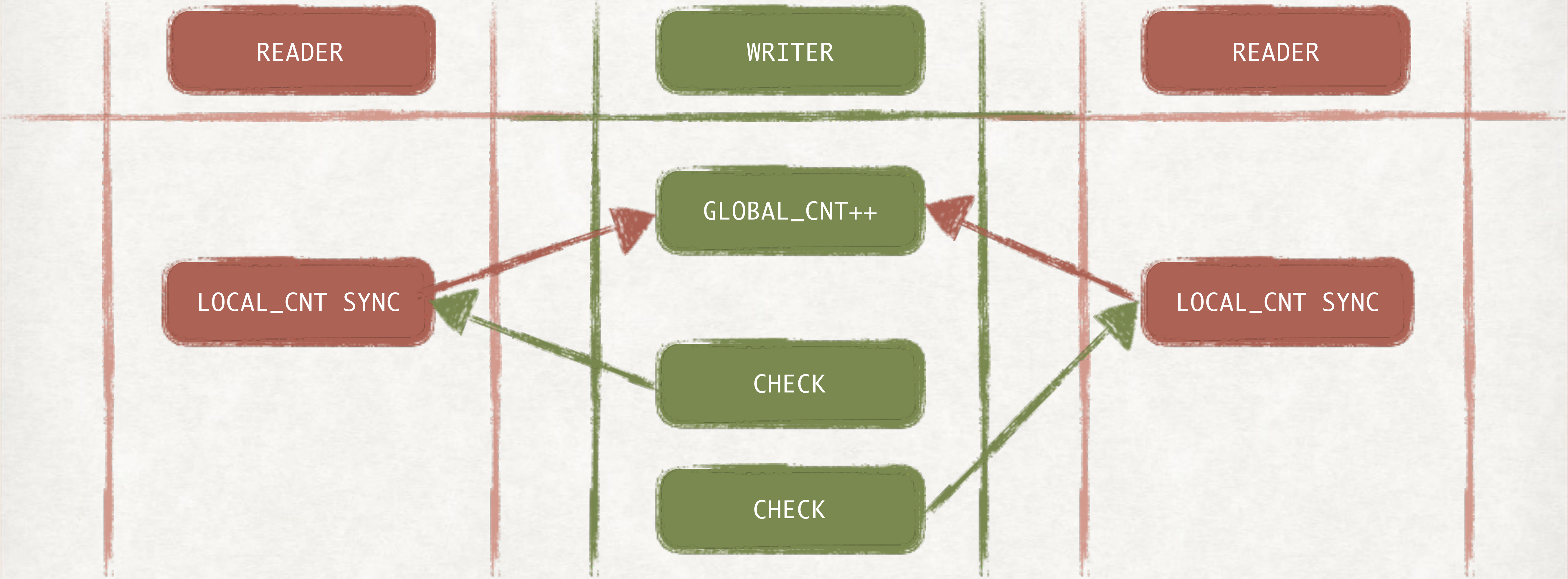
SOLUTION ON SOFTWARE

MONOTONE VERSION



SOLUTION ON SOFTWARE

MONOTONE VERSION

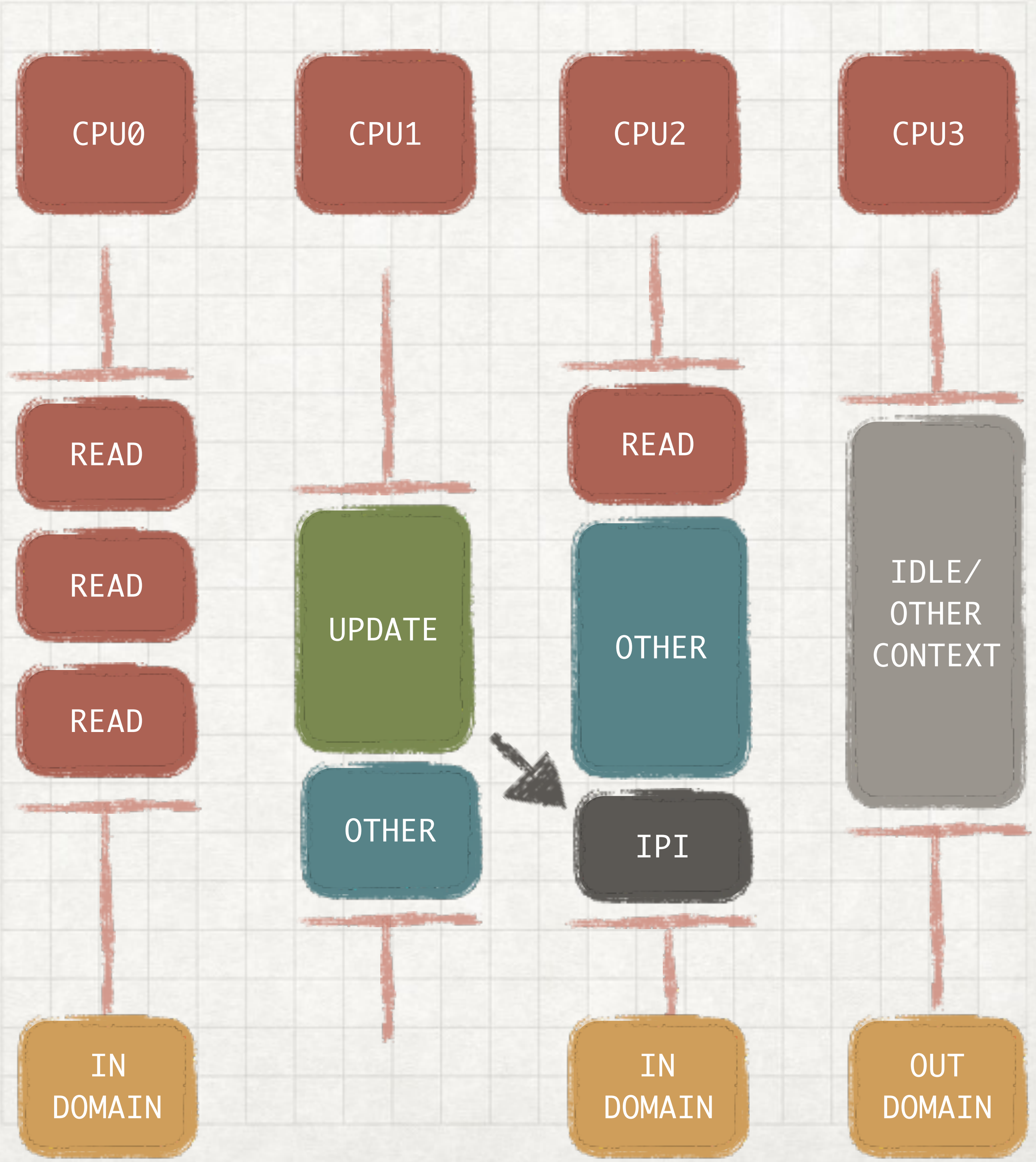


WHAT IF READERS HAVE NO CHANCE TO SYNC

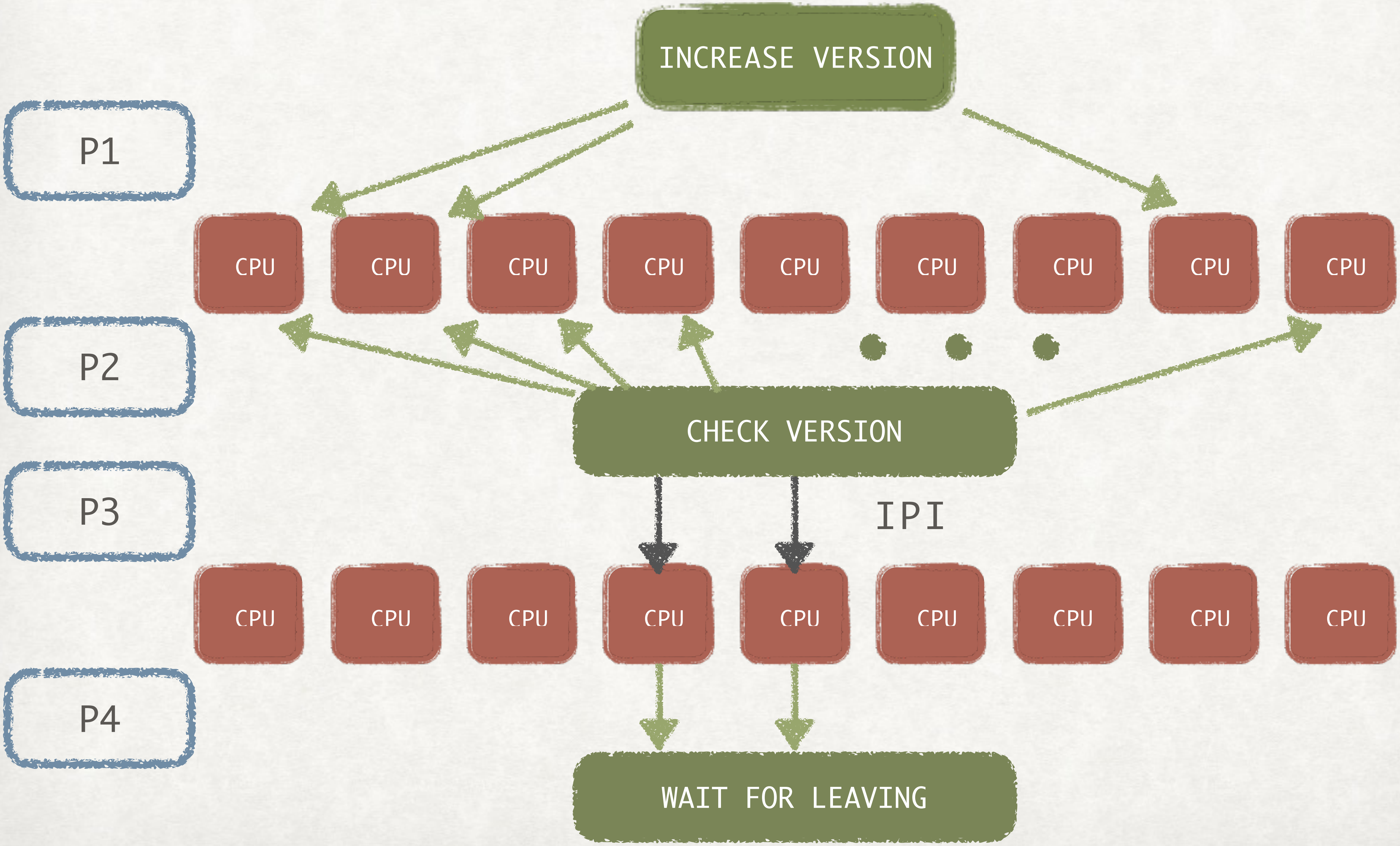
SUPPLEMENT

EVENTS & REDUCE EVENTS

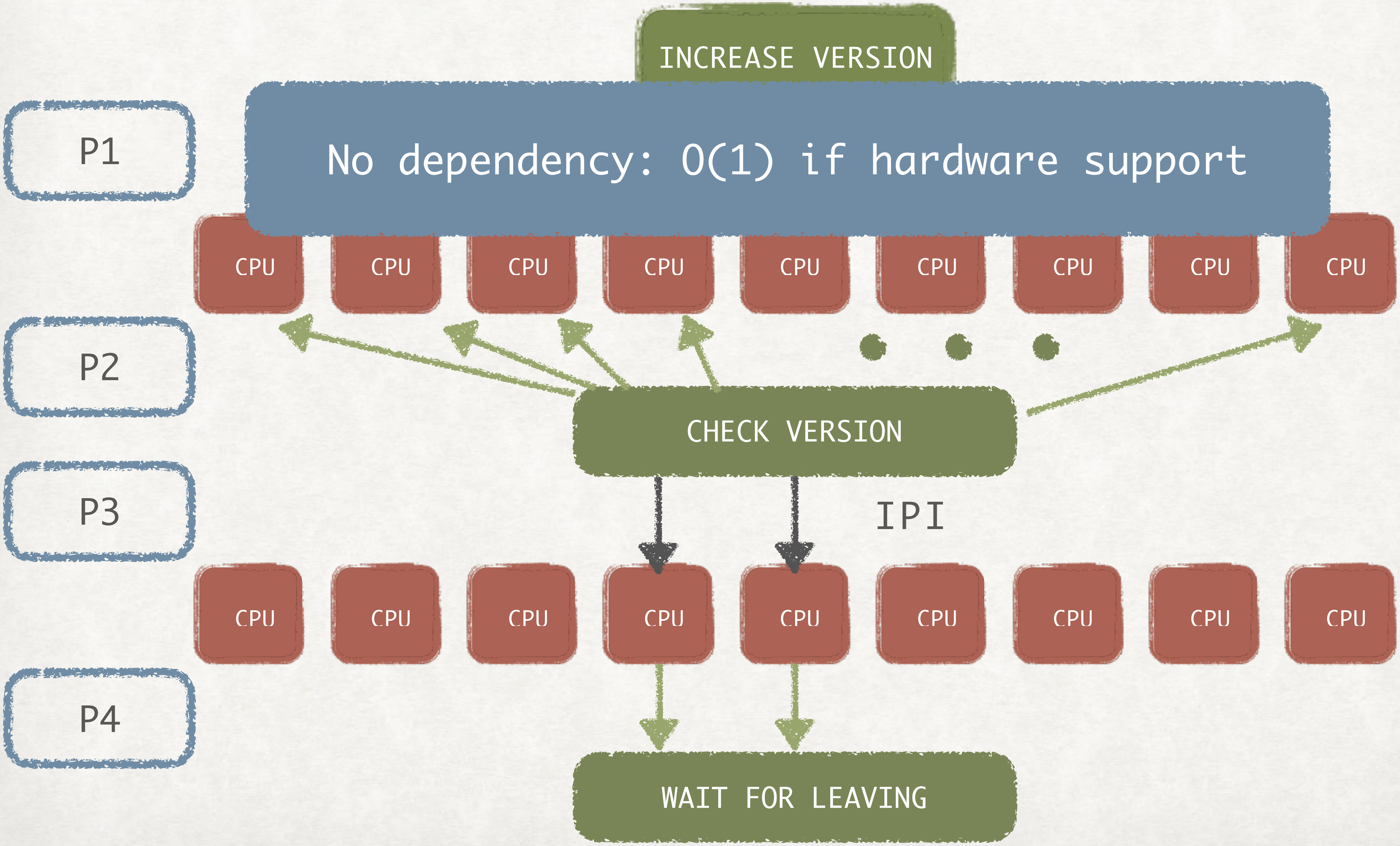
- Event
  - IPI
- Reducing Events
  - Domain



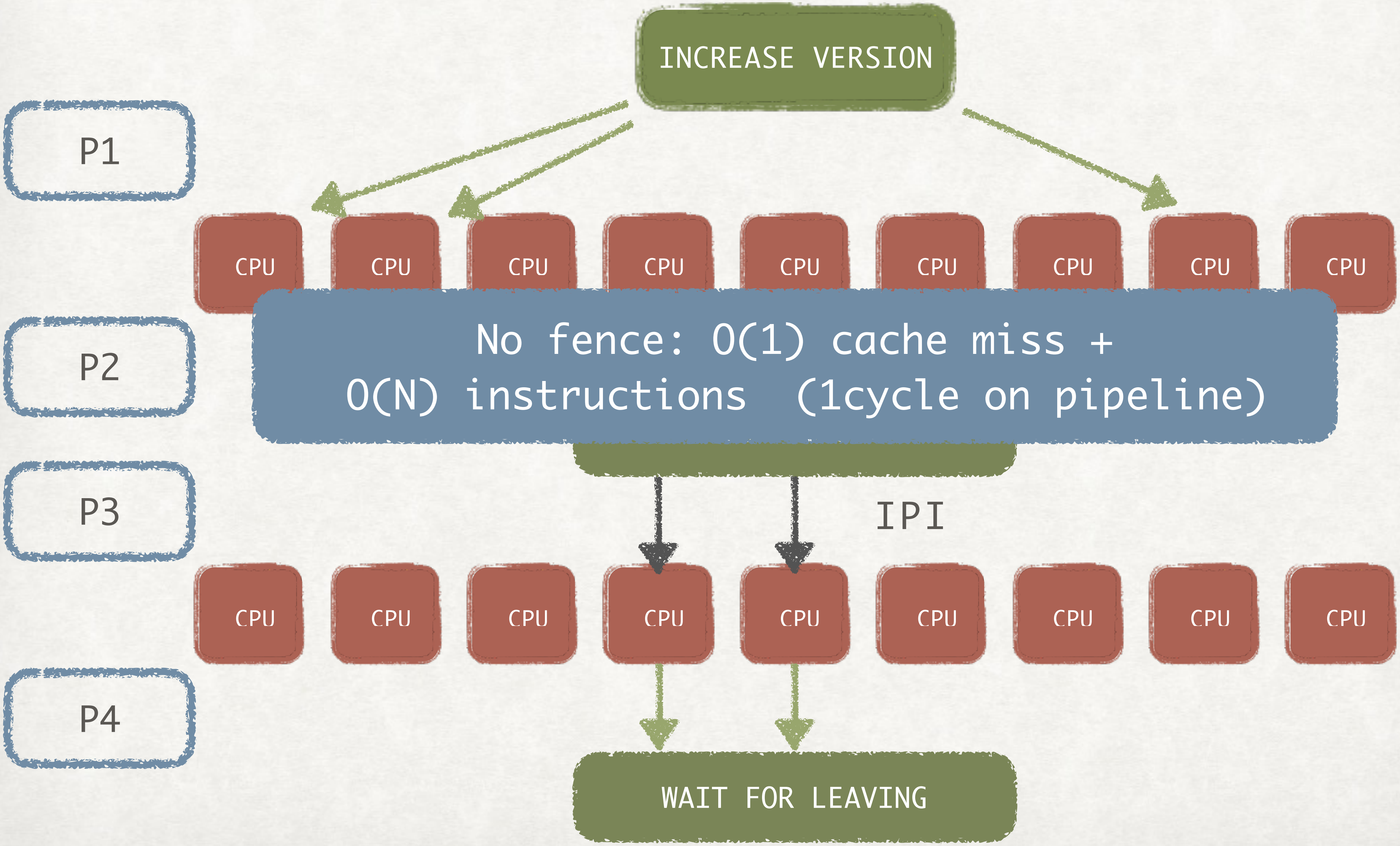
PRCU SYNCHRONIZATION PHASES



SYNCHRONIZATION PHASES

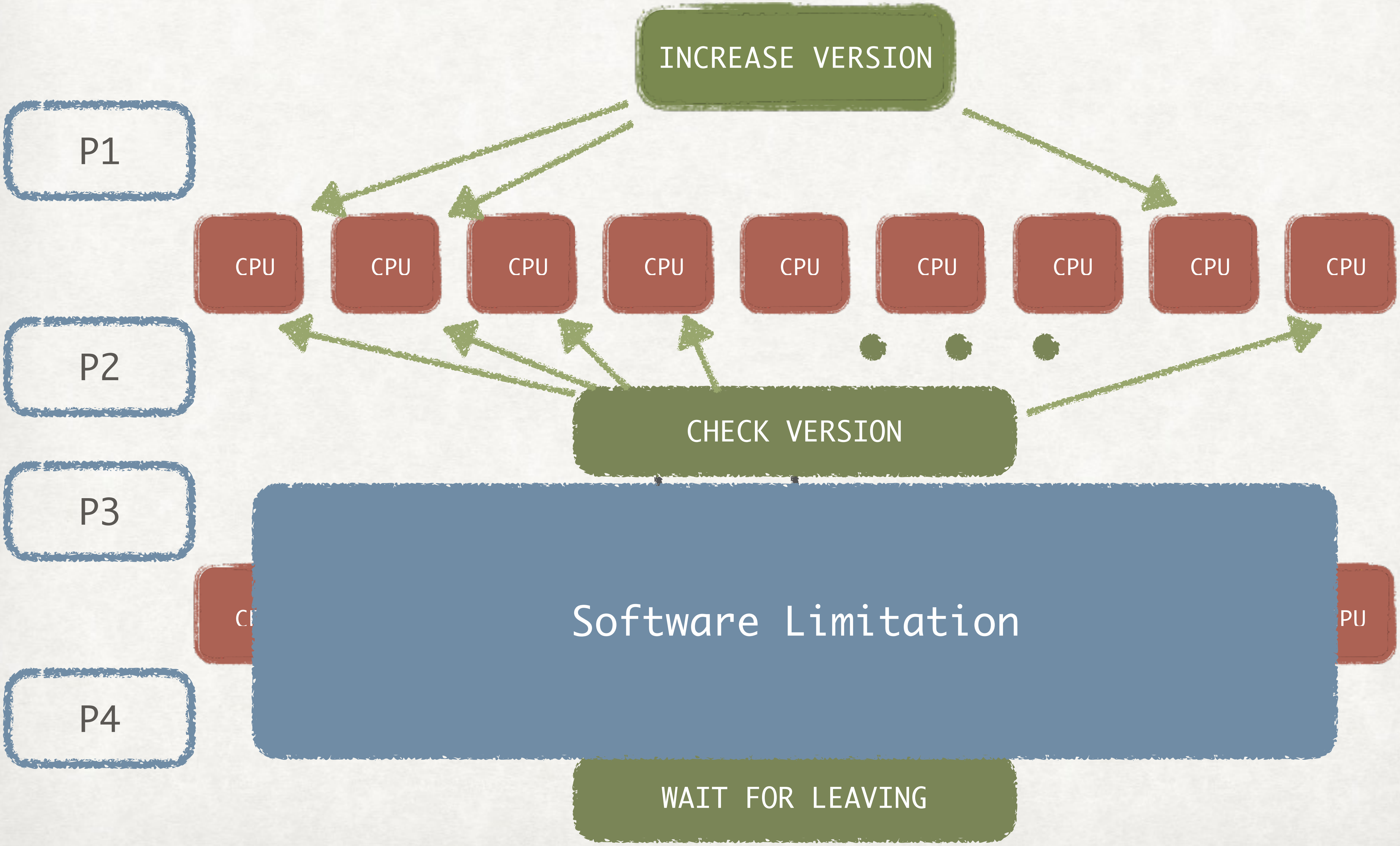


SYNCHRONIZATION PHASES



No fence: O(1) cache miss +
O(N) instructions (1cycle on pipeline)

SYNCHRONIZATION PHASES



Software Limitation

CORRECTNESS

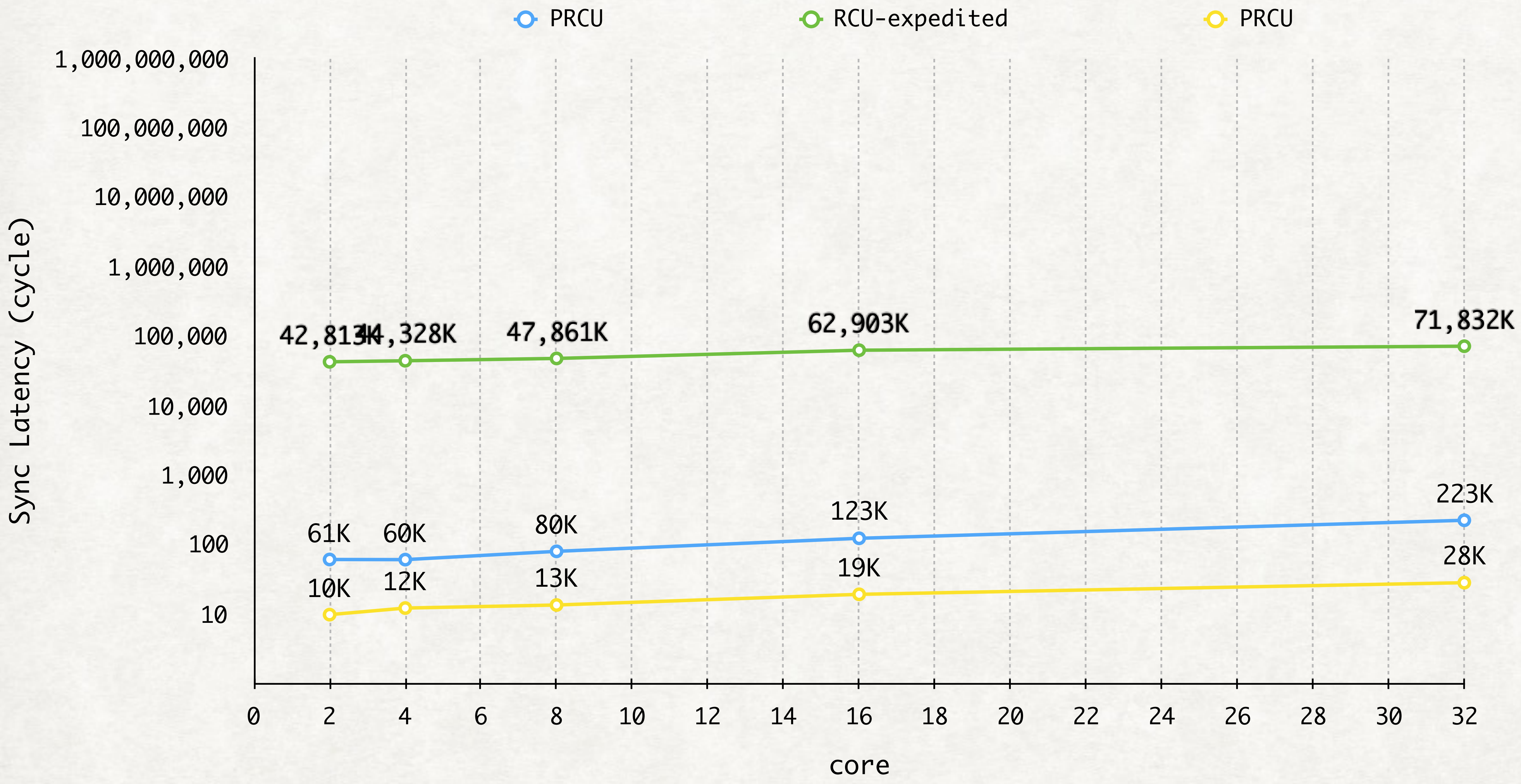
- Testing
  - Pass rcutorture(—torture rcu)
- Formal Verification
  - Pass model checking

FORMAL VERIFICATION

- Tool
  - CBMC, https://github.com/diffblue/cbmc
- Target
  - prcu_read_lock, prcu_read_unlock, synchronized_prcu
- Hardware
  - 16 cores, Intel Xeon CPU@2.4GHz, 16G Memory
- Configuration
  - 2 reader threads + 1 writer thread + 1 main thread (+ 3 interrupt threads)
  - safety + liveness
  - Memory model : SC, TSO, PSO

PERFORMANCE

COMPARE WITH TREE RCU (LINUX 4.0.5)



SUMMARY

- Introduce a problem on reader-writer synchronization
- A solution call PRCU which has low latency on ideal hardwares
- Proof correctness with testing and formal verification
- Code: https://github.com/lihao/linux-prcu

THANKS