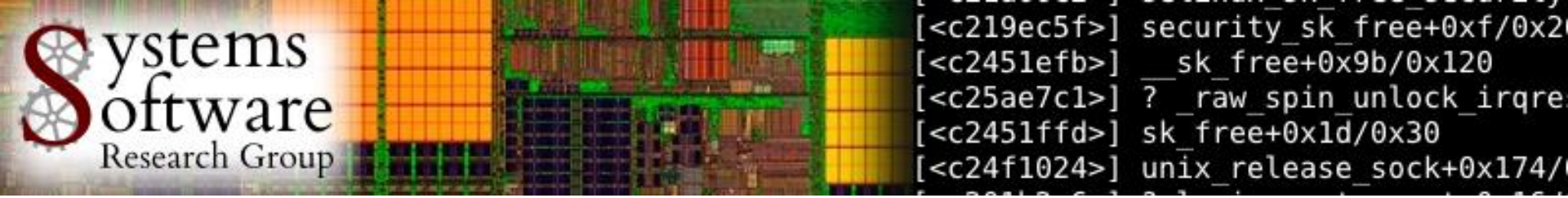


Software Transactional Memory (4)

Mohamed Mohamedin

Chapter 4 of TM Book

Lock-based STMs with Global Metadata

- A category of STMs where global metadata only is used
  - No per-object (location) metadata
    - Sometimes called Ownership record (Orec)
  - Low memory overhead
  - Low cache pressure
    - Less number of atomic operations
  - Must reduce the amount of interaction with the global metadata
    - From per-access to per-transaction

NOrec

- One global counter (acts as a global lock)
  - Global versioned lock
- Value-based validation
  - Read-set has the location and the read value
- Writes are buffered

Value-based Validation

- Detecting read-set invalidation by comparing actual values (instead of versions)

Address	Value
10	101
17	305
20	0
5	15
11	332

Memory
15
101
332
305
0

Value-based Validation

- Detecting read-set invalidation by comparing actual values (instead of versions)

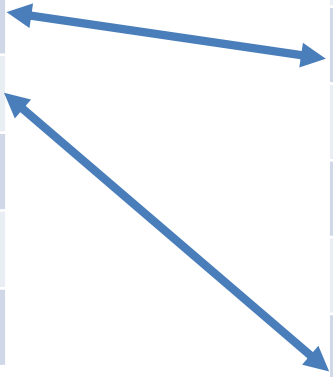
Address	Value
10	101
17	305
20	0
5	15
11	332

Memory
15
101
332
305
0

Value-based Validation

- Detecting read-set invalidation by comparing actual values (instead of versions)

Address	Value
10	101
17	305
20	0
5	15
11	332

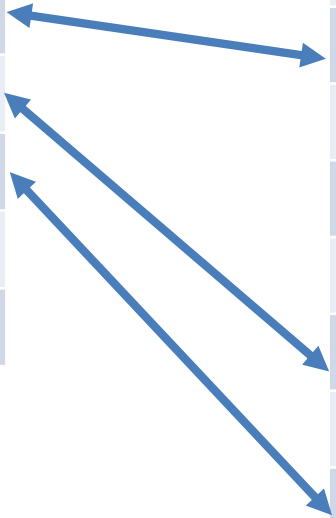


Memory
15
101
332
305
0

Value-based Validation

- Detecting read-set invalidation by comparing actual values (instead of versions)

Address	Value
10	101
17	305
20	0
5	15
11	332

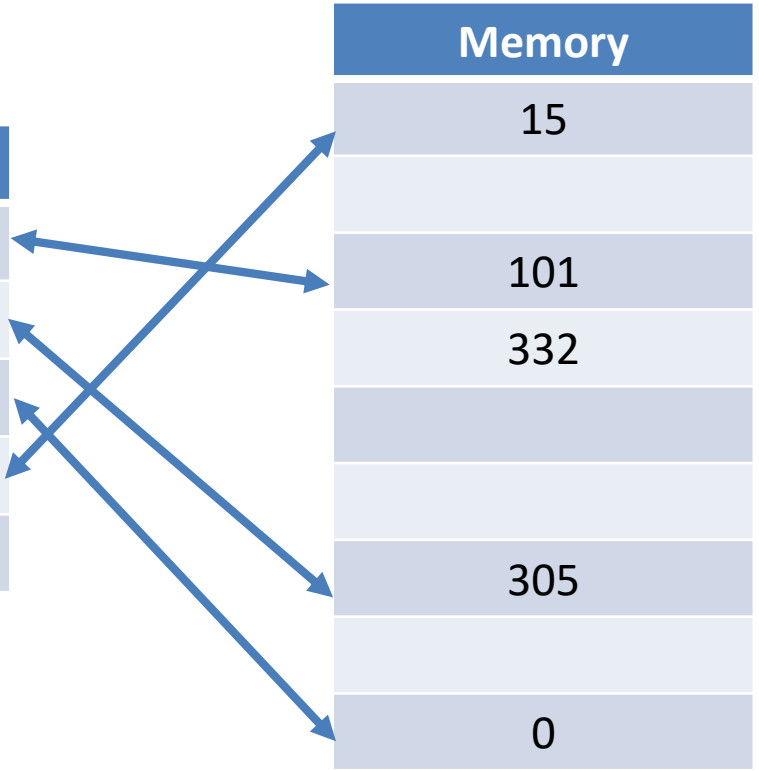


Memory
15
101
332
305
0

Value-based Validation

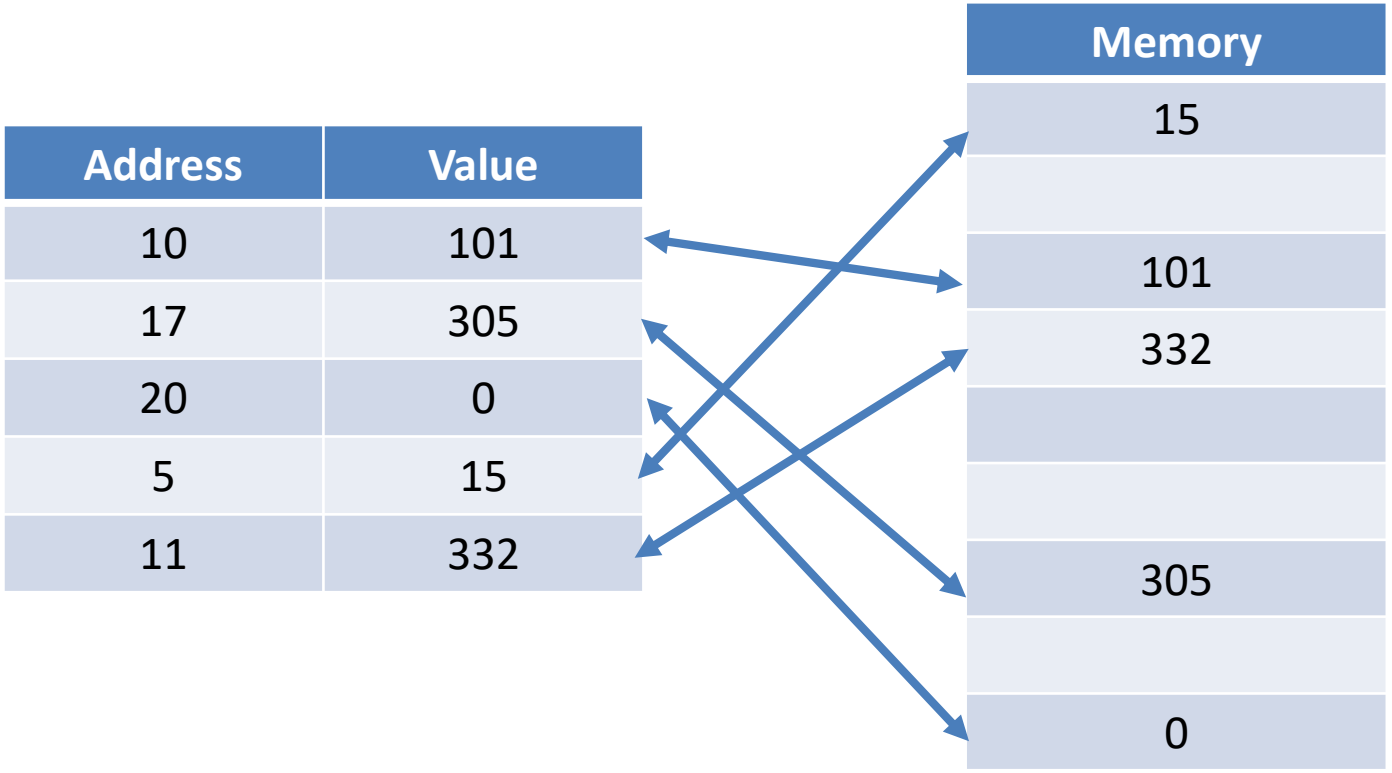
- Detecting read-set invalidation by comparing actual values (instead of versions)

Address	Value
10	101
17	305
20	0
5	15
11	332



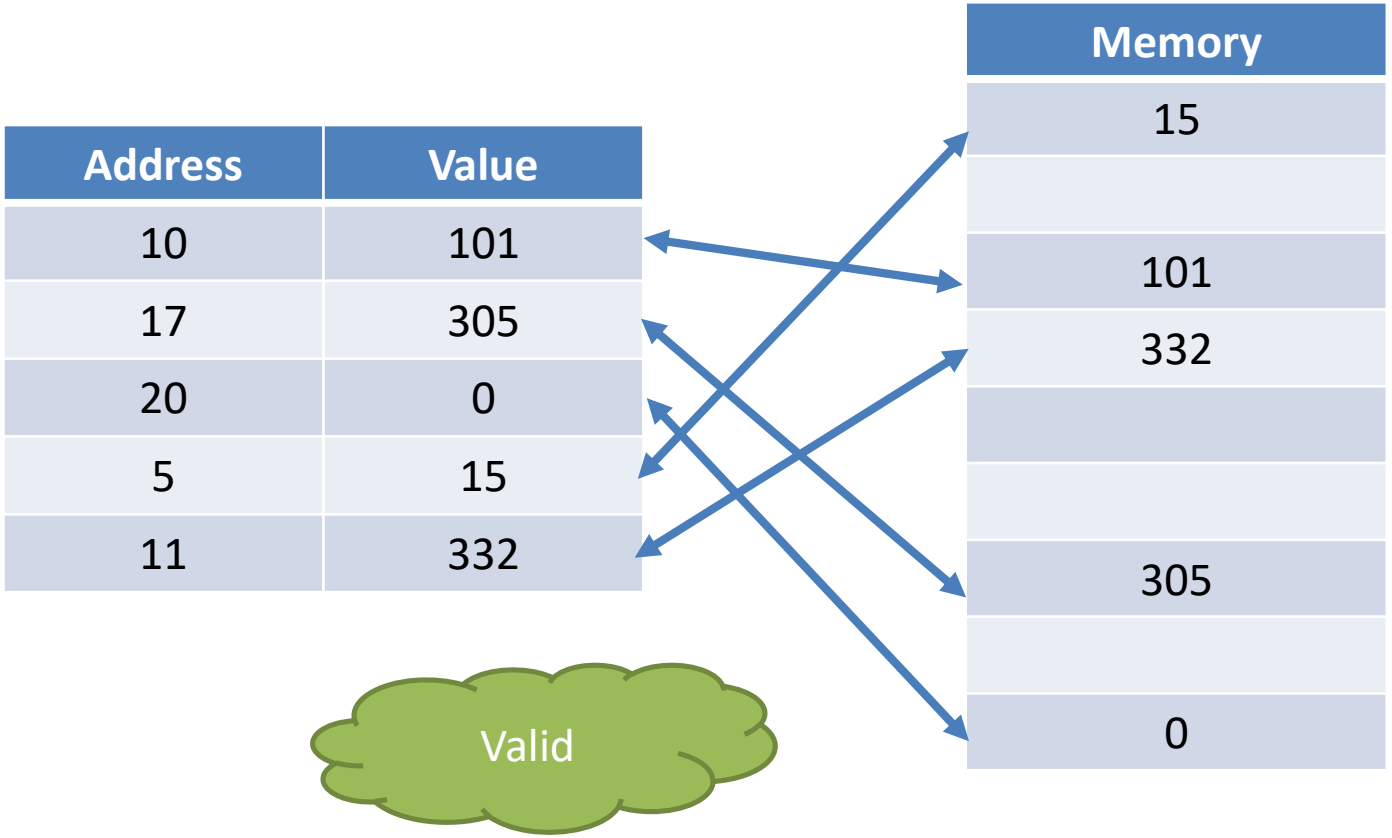
Value-based Validation

- Detecting read-set invalidation by comparing actual values (instead of versions)



Value-based Validation

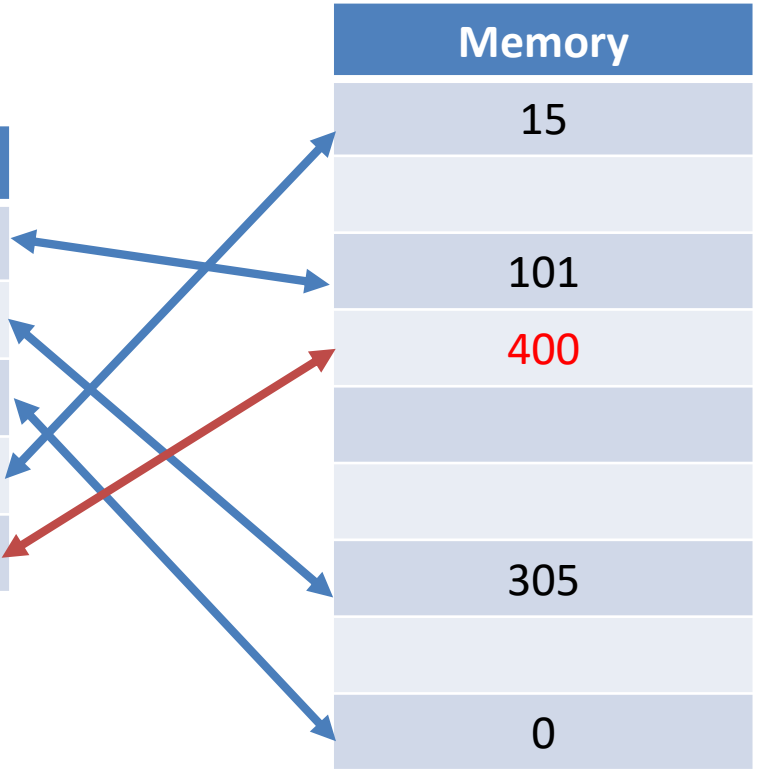
- Detecting read-set invalidation by comparing actual values (instead of versions)



Value-based Validation

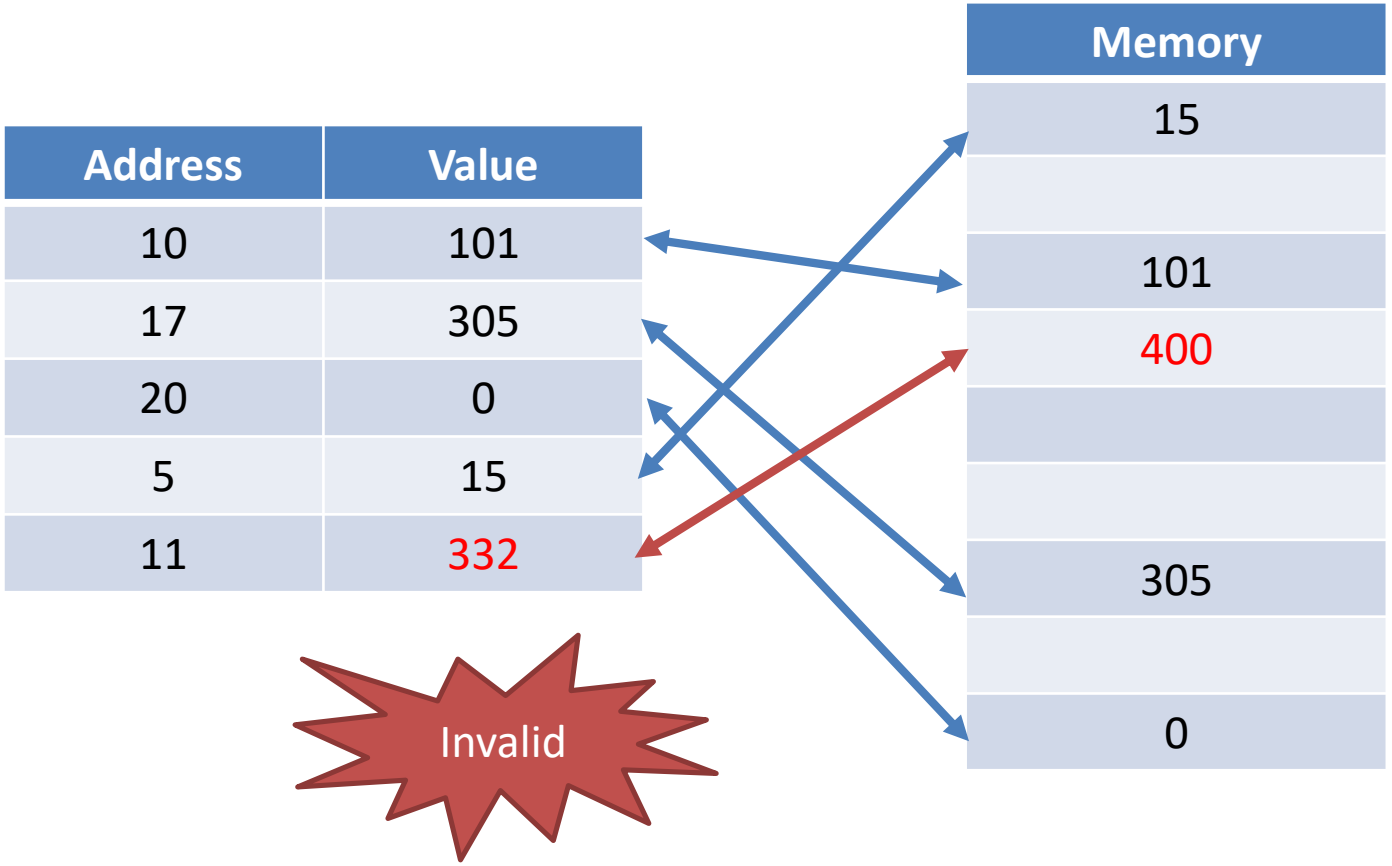
- Detecting read-set invalidation by comparing actual values (instead of versions)

Address	Value
10	101
17	305
20	0
5	15
11	332



Value-based Validation

- Detecting read-set invalidation by comparing actual values (instead of versions)



Value-based Validation

- T1
  - v1 = tx_read(x)
  - V2 = tx_read(x)
  - tx_commit()

X = 10 initially

- T2
  - tx_write(x, 50)
  - tx_commit()
- T3
  - tx_write(x, 10)
  - tx_commit()

Value-based Validation

- T1
  - v1 = tx_read(x)
  - V2 = tx_read(x)
  - tx_commit()

X = 10 initially

- T2
  - tx_write(x, 50)
  - tx_commit()
- T3
  - e(x, 10)
  - mit()

Value-based validation at commit time will not detect this issue

Value-based Validation

- T1
  - v1 = tx_read(x)
  - V2 = tx_read(x)
  - tx_commit()

X = 10 initially

- T2
  - tx_write(x, 50)
  - tx_commit()
- T3
  - tx_write(x, 10)
  - tx_commit()

Value-based validation after each read is expensive also!

NOrec

- tx_begin()
  - Reset read-set and write-set (write-buffer)
  - do
    - RV = Global-Clock
  - while((RV & 1) != 0)

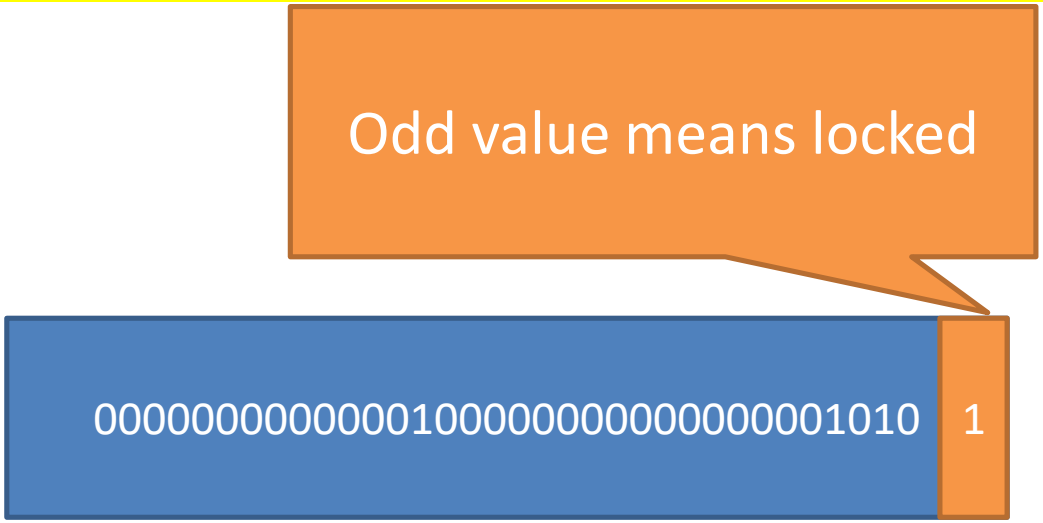
NOrec

- tx_begin()
  - Reset read-set and write-set (write-buffer)
  - do
    - RV = Global-Clock
  - while((RV & 1) != 0)

Global Clock acts as a lock as well as a counter
Odd value means locked
Even value means a version

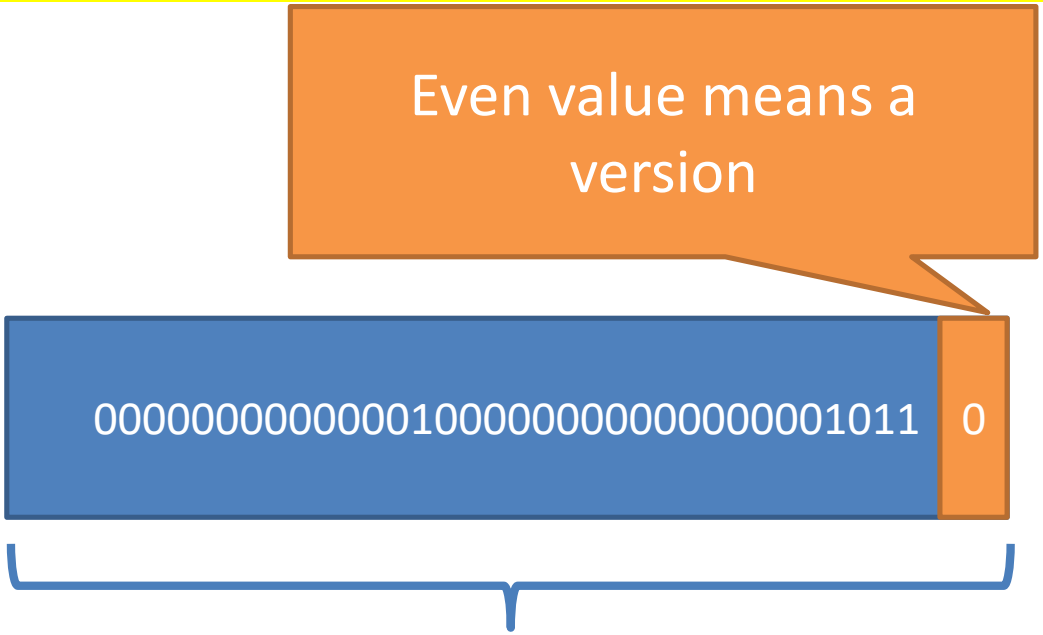
NOrec

- ((Global-Clock & 1) != 0) → Locked



NOrec

- ((Global-Clock & 1) == 0) → Unlocked



NOrec

```
void TXBegin()
1     do
2        snapshot = global_lock
3     while ((snapshot & 1) != 0)
```

NOrec

- tx_write(addr, value)
  - Add (or update) the addr and value to the write-set

NOrec

```
void  TXWrite(Address addr, Value val)
1     writes[addr] = val
```

NOrec

- tx_read(addr)
  - Find the addr is in the write-set
    - If found, return the value buffered in the write-set
  - val = *addr
  - while (RV != Global-Clock)
    - RV = tx_validate()
    - val = *addr
  - Add (addr & val) to read-set
  - Return val

NOrec

```
Value TXRead(Address addr)
1    if (writes.contains(addr))
2      return writes[addr]
3
4    val = *addr
5    while (snapshot != global_lock)
6      snapshot = Validate()
7      val = *addr
8
9    reads.append(address, value)
10   return val
```

NOrec

- tx_validate()
  - while(true)
    - time = Global-Clock
    - if ((time & 1) != 0) continue
    - for each entry in the read-set
      - if (*entry.addr != entry.val)
        - » tx_abort()
    - if (time == Global-Clock)
      - return time

NOrec

- tx_validate()
  - while(true)
    - time = Global-Clock
    - if ((time & 1) != 0) continue
    - for each entry in the read-set
      - if (*entry.addr != entry.val)
        - » tx_abort()
    - if (time == Global-Clock)
      - return time

Global Clock is locked
Wait until it is unlocked. Some tx is committing now

NOrec

- tx_validate()
  - while(true)
    - time = Global-Clock
    - if ((time & 1) != 0) continue
    - for each entry in the read-set
      - if (*entry.addr != entry.val)
        - » tx_abort()
    - if (time == Global-Clock)
      - return time

Global Clock is **not** changed during the validation, so, validation is successful. Otherwise, repeat again

NOrec

```
unsigned Validate()
1    while (true)
2      time = global_lock
3      if ((time & 1) != 0)
4        continue
5
6      for each (addr, val) in reads
7        if (*addr != val)
8          TXAbort() // abort will longjmp
9
10       if (time == global_lock)
11         return time
```

NOrec

- tx_commit()
  - if (write-set.size == 0) // read-only tx
    - return
  - while(!CAS(&Global-Clock, RV, RV+1))
    - RV = tx_validate()
  - //Write back
  - For each entry in the write-set
    - *entry.addr = entry.value
  - //Unlock and update global clock version
  - Global-Clock = RV+2

NOrec (Remember)

- tx_validate()
  - while(true)
    - time = Global-Clock
    - if ((time & 1) != 0) continue
    - for each entry in the read-set
      - if (*entry.addr != entry.val)
        - » tx_abort()
    - if (time == Global-Clock)
      - return time

NOrec

```
void TXCommit()
1     if (read-only transaction)
2       return
3
4     while (!CAS(&global_lock, snapshot, snapshot + 1))
5       snapshot = Validate()
6
7     for each (addr, val) in writes
8       *addr = val
9
10    global_lock = snapshot + 2 // one more than CAS above
```

NOrec

- tx_abort
  - //Just jump back to tx_begin to restart the transaction

ABA Problem

- Any algorithm depending on the following
  - v1 = x
  - CAS(&x, v1, new_val)
- Has a potential ABA problem!

ABA Problem

- T1
  - v1 = x
  - CAS(&x, v1, new_val)

X = 10 initially

ABA Problem

- T1
  - v1 = x
  - CAS(&x, v1, new_val)
- T2
  - x = 50

X = 10 initially

ABA Problem

- T1
  - v1 = x
  - CAS(&x, v1, new_val)
- T2
  - x = 50
- T3
  - x = x/5

X = 10 initially

ABA Problem

- T1
  - v1 = x
  - CAS(&x, v1, new_val)
- T2
  - x = 50
- T3
  - x = x/5

Will this CAS succeed?

X = 10 initially

ABA Problem

- T1
  - v1 = x
  - CAS(&x, v1, new_val)
- T2
  - x = 50
- T3
  - x = x/5

Will this CAS succeed?
YES!!!

X = 10 initially

ABA Problem

- T1
  - v1 = x
  - CAS(&x, v1, new_val)
- T2
  - x = 50
- T3
  - x = x/5

At this time x is 10 which is the same as v1, although x is changed in between these two lines

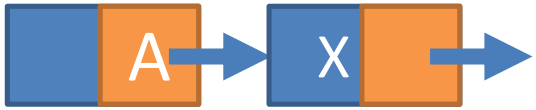
X = 10 initially

ABA Problem

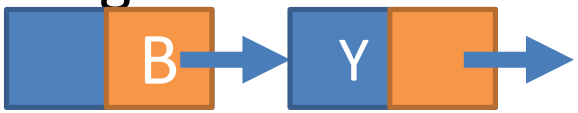
- From Wikipedia
  - Process P1 reads value A from shared memory,
  - P1 is preempted, allowing process P2 to run,
  - P2 modifies the shared memory value A to value B and back to A before preemption,
  - P1 begins execution again, sees that the shared memory value has not changed and continues.
- Why it is a problem?
  - 'A' can be a pointer value?!

ABA Problem

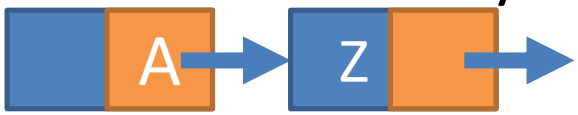
- Consider a lined list:



- 'A' is a pointer pointing to node X
- Node X is deleted and memory is reclaimed
  - 'A' pointer is changed to 'B'



- Another node is inserted in the same position
  - Due to memory allocator optimization, the new node is allocated in the same memory of node X



- So, now pointer has value 'A' again while pointing to another node!

ABA Problem

- Solution:
  - Use tagged pointers
    - Some of the lower bits of the pointer is used as marker
      - Remember markable reference
      - Limited
    - Use Double-word CAS if an architecture support it
      - Now we have enough mark which acts as a version
  - Use Deferred Reclamation
    - Garbage Collector
    - Other techniques similar to Hazard Pointers

Value-based Validation & ABA

- Value-based Validation suffers from ABA problem!
  - But, if both the pointer and value pointer at are in the read-set, then ABA is solved
- What about version-based validation?

Example

- tx_begin()
- p = tx_read(pointer)
- val = *p
- .
- .
- .
- tx_commit()

Is this transaction suffer from ABA problem with NOrec?

Example

- tx_begin()
- p = tx_read(pointer)
- val = *p
- .
- .
- .
- tx_commit()

Is this transaction suffer from ABA problem with NOrec?

YES

Example

- tx_begin()
- p = tx_read(pointer)
- val = tx_read(*p)
- .
- .
- .
- tx_commit()

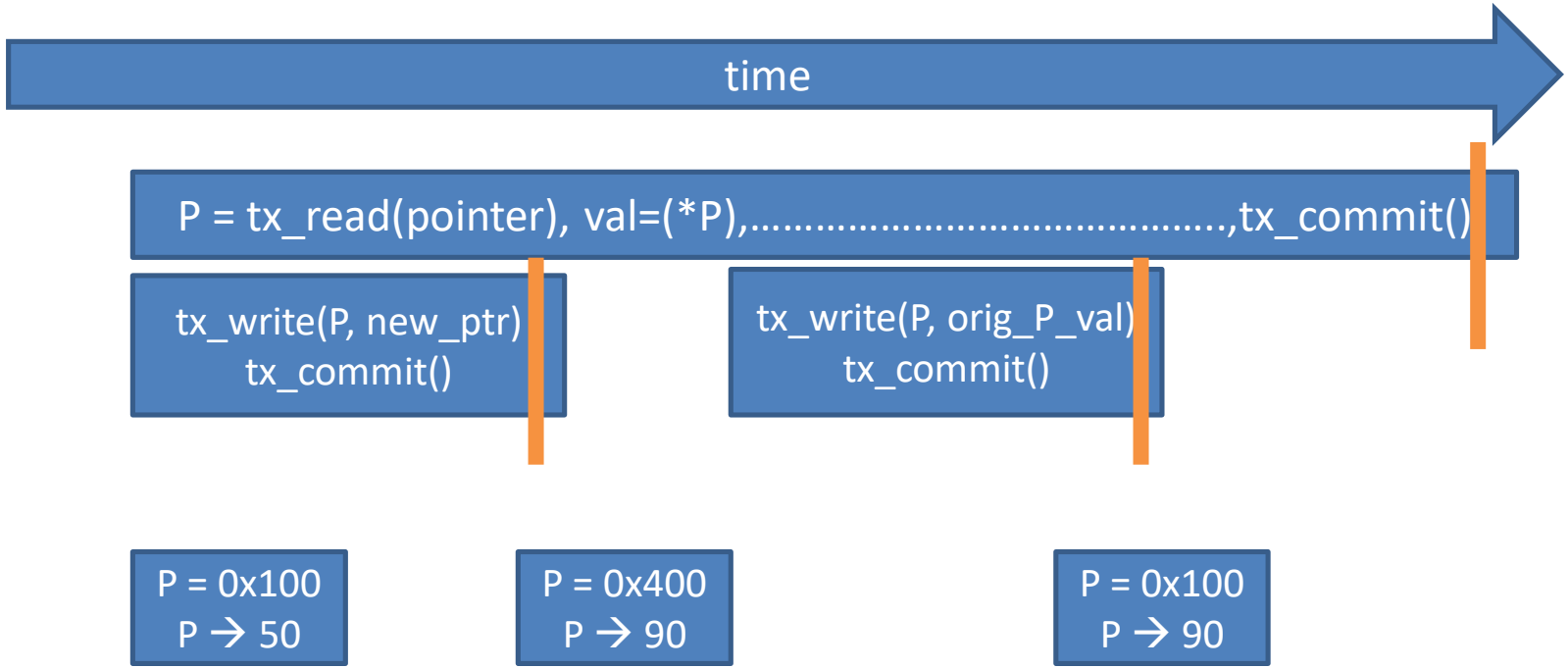
Both the pointer and the value are in the read-set now

Example

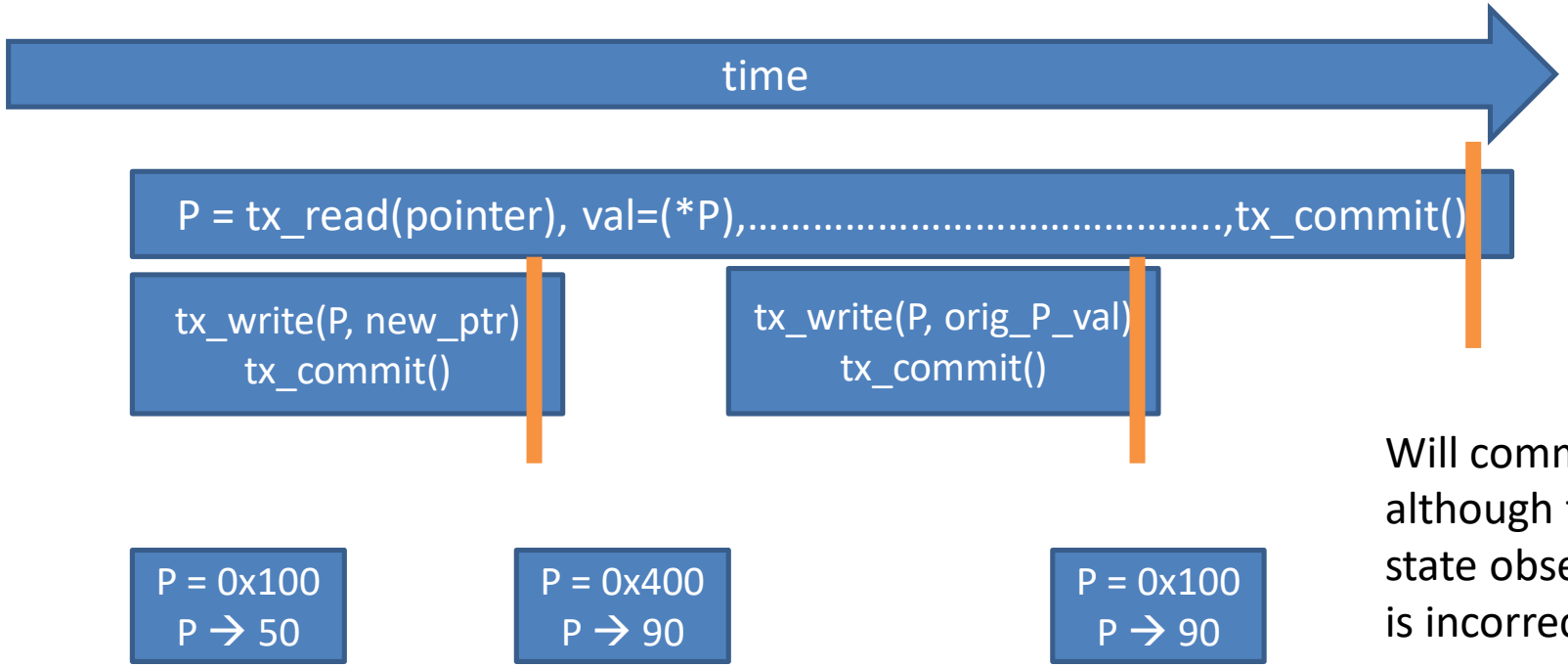
- tx_begin()
- val = tx_read(*pointer)
- .
- .
- .
- .
- tx_commit()

Or, by reading the value of the pointer, we are safe

Example

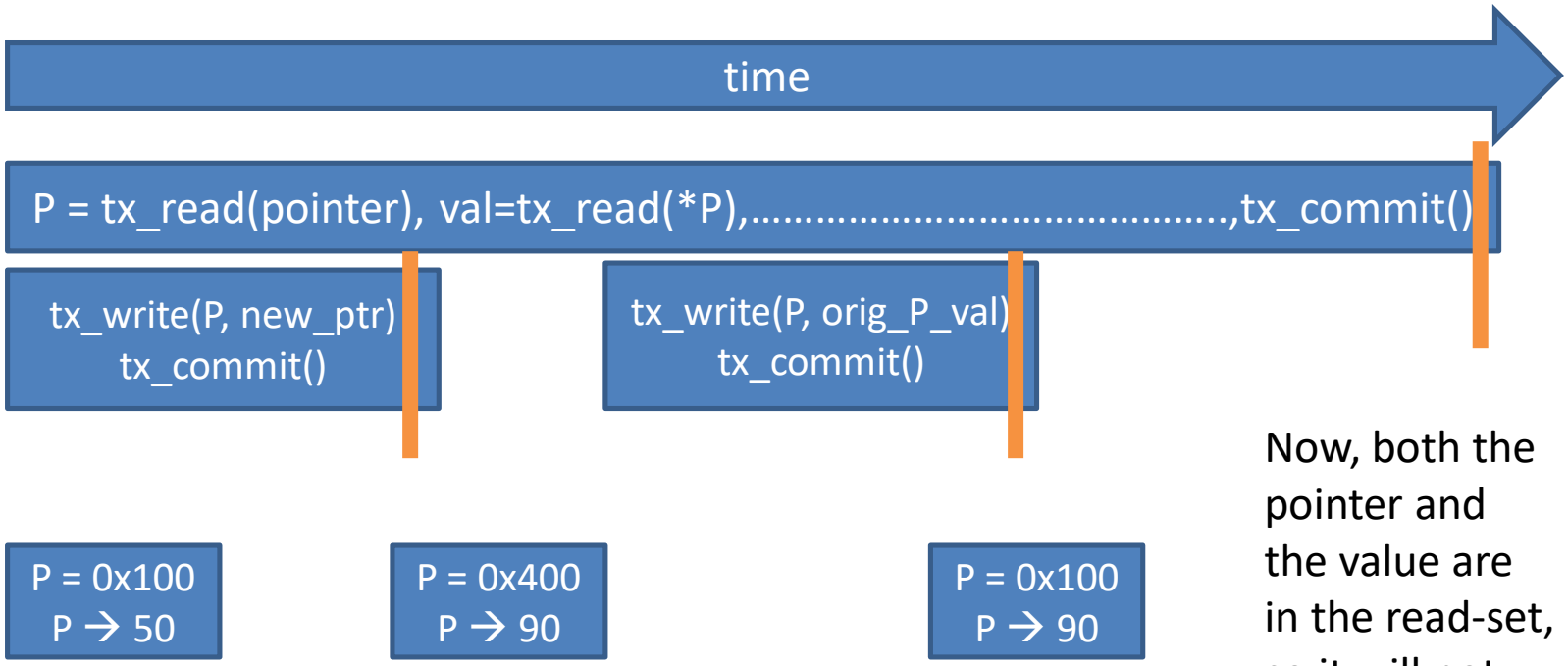

time
P = tx_read(pointer), val=(*P),……..,tx_commit()
tx_write(P, new_ptr)
tx_commit()
tx_write(P, orig_P_val)
tx_commit()
P = 0x100
P → 50
P = 0x400
P → 90
P = 0x100
P → 90

Example



Will commit although the state observed is incorrect

Example



Now, both the pointer and the value are in the read-set, so it will not commit

Other Issues of NOrec

- Commits are serialized
  - Only one transaction is allowed to commit at a time
  - Limits scalability of the system

Is NOrec Opaque?

- Does it suffer from zombies?

T1:

- tx_begin
- tx_write(x, 2)
- tx_write(y, 4)
- tx_commit()

T2:

- tx_begin
- v1 = tx_read(y)

Initially x=1, y=2
Invariant: y = 2x

- v2 = tx_read(x)
- tx_write(z, 1/(v1 – v2))
- tx_commit()

Is NOrec Opaque?

- Does it suffer from zombies?

T1:

- tx_begin
- tx_write(x, 2)
- tx_write(y, 4)
- tx_commit()

T2:

- tx_begin
- v1 = tx_read(y)
- v2 = tx_read(x)
- tx_write(z, 1/(v1 – v2))
- tx_commit()

NO, it validates after every read. The change in the Global-Clock and in the value of y will be detected

NOrec

T1:

- tx_begin
- v1 = tx_read(x)
- tx_write(y, v1 + 10)
- tx_commit()

T2:

- tx_begin
- v1 = tx_read(x)
- v2 = tx_read(y)
- tx_write(y, v1 + v2)
- tx_write(x, v1 + 1)
- tx_commit()

NOrec

T1:

- tx_begin
- v1 = tx_read(x)
- tx_write(y, v1 + 10)
- tx_commit()

T2:

- tx_begin
- v1 = tx_read(x)
- v2 = tx_read(y)
- tx_write(y, v1 + v2)
- tx_write(x, v1 + 1)
- tx_commit()

Will both transactions commit?
Are they Linearizable?

NOrec

T1:

- tx_begin
- v1 = tx_read(x)
- tx_write(y, v1 + 10)
- tx_commit()

T2:

- tx_begin
- v1 = tx_read(x)
- v2 = tx_read(y)
- tx_write(y, v1 + v2)
- tx_write(x, v1 + 1)
- tx_commit()

Will both transactions commit?
Yes, NOrec support timestamp extension?

NOrec

T1:

- tx_begin
- v1 = tx_read(x)
- tx_write(y, v1 + 10)
- tx_commit()

T2:

- tx_begin
- v1 = tx_read(x)
- v2 = tx_read(y)
- tx_write(y, v1 + v2)
- tx_write(x, v1 + 1)
- tx_commit()

NOrec

T1:

- tx_begin
- v1 = tx_read(x)
- tx_write(y, v1 + 10)
- tx_commit()

T2:

- tx_begin
- v1 = tx_read(x)
- v2 = tx_read(y)
- tx_write(y, v1 + v2)
- tx_write(x, v1 + 1)
- tx_commit()

Will both transactions commit?
Are they Linearizable?

NOrec

T1:

- tx_begin
- tx_write(x, 15)

- v1 = tx_read(x)
- tx_write(y, v1 + 15)
- tx_commit()

T2:

- tx_begin
- v1 = tx_read(x)
- v2 = tx_read(y)
- tx_write(y, v1 + v2)
- tx_write(x, v1 + 1)
- tx_commit()

NOrec

T1:

- tx_begin
- tx_write(x, 15)

- v1 = tx_read(x)
- tx_write(y, v1 + 15)
- tx_commit()

T2:

- tx_begin
- v1 = tx_read(x)
- v2 = tx_read(y)
- tx_write(y, v1 + v2)
- tx_write(x, v1 + 1)
- tx_commit()

What about this scenario?

NOrec

T1:

- tx_begin
- v1 = tx_read(x)
- tx_write(y, v1 + 15)
- tx_commit()

T2:

- tx_begin
- v1 = tx_read(x)
- v2 = tx_read(y)
- tx_write(y, v1 + v2)
- tx_write(x, v1 + 1)
- tx_commit()

NOrec

T1:

- tx_begin

T2:

- tx_begin
- v1 = tx_read(x)
- v2 = tx_read(y)
- tx_write(y, v1 + v2)
- tx_write(x, v1 + 1)
- tx_commit()

- v1 = tx_read(x)
- tx_write(y, v1 + 15)
- tx_commit()

What about this scenario?