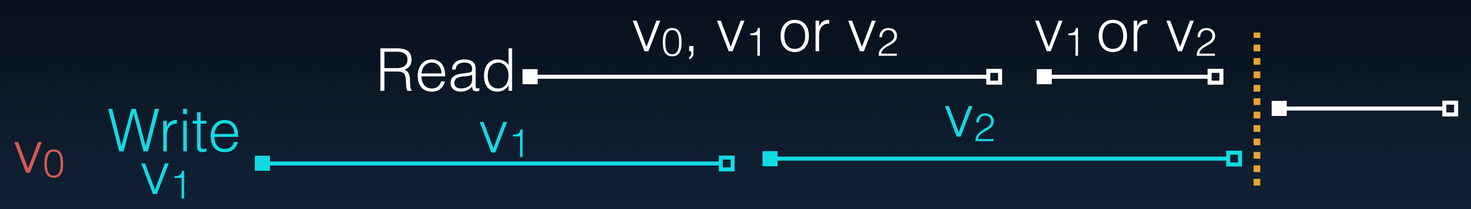
COL380

Introduction to
Parallel & Distributed Programming

Registers

Wait free

- **Regular**



  - ➡ Single writer

Quiescently consistent

  - ➡ Reads overlapping with a write return the newly written value or the one prior

- **Atomic**

A later read cannot see an earlier write

  - ➡ Multiple writers

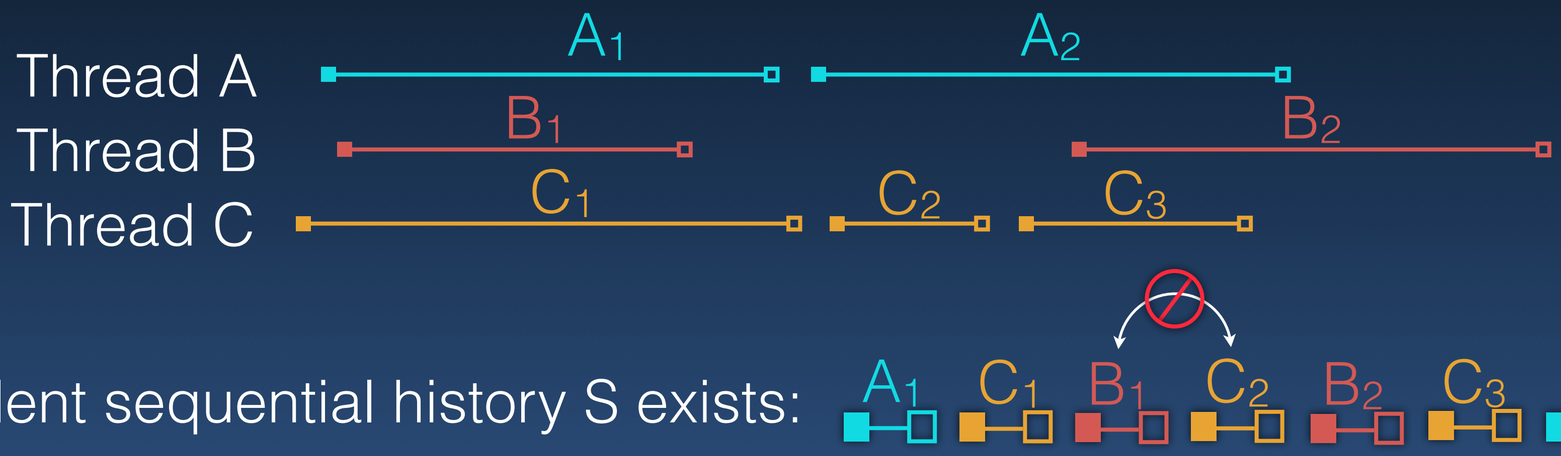
Linearizable

  - ➡ Read the value of latest write (*takes effect* at some point in its duration)

Linearizable

- Operations appear to take effect a some instant

  ➡ between their start and end



1. Each thread's history is retained in S
2. Non 'overlapping' operations retain order

Composable

Strict Consistency

- Read(x) returns the value stored by the ‘most recent’ write(x)
  - ➡ must appear as if operations are completed instantaneously
  - ➡ requires knowledge of absolute global time
  - ➡ intuitive to reason in this model
- Linearizability is weaker
  - ➡ accesses have a linearization point between their start and completion
    - ▸ and appear to occur instantly at the linearization point
  - ➡ Need a notion of a global clock
  - ➡ Useful for reasoning

Sequential Consistency

- “A multiprocessor is sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.” [Lamport, 1979]

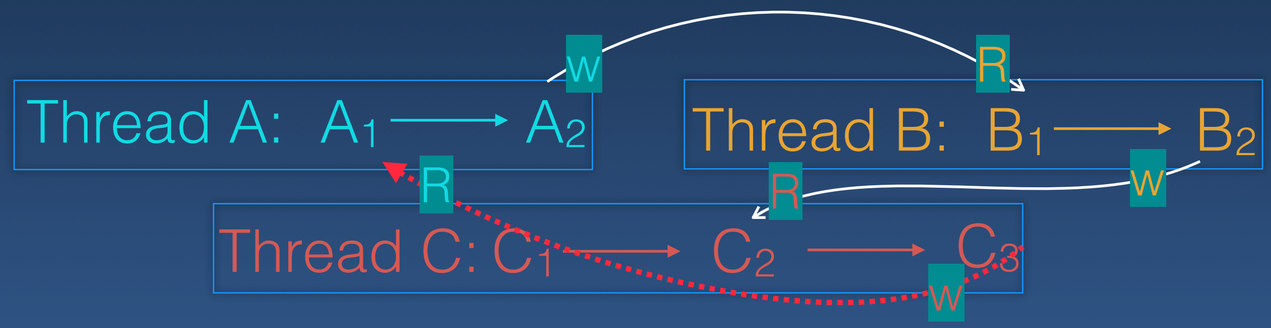
< Weaker than Linearizability

Sequentially Consistent

Thread A: X = 5 — EnQ(5)

Thread B: X = 3 — EnQ(3) | Read X (3) — DeQ is 3 | Read X (5) — DeQ is 5

- No global notion of time
  - ➡ Only consistent Order



Sequential History

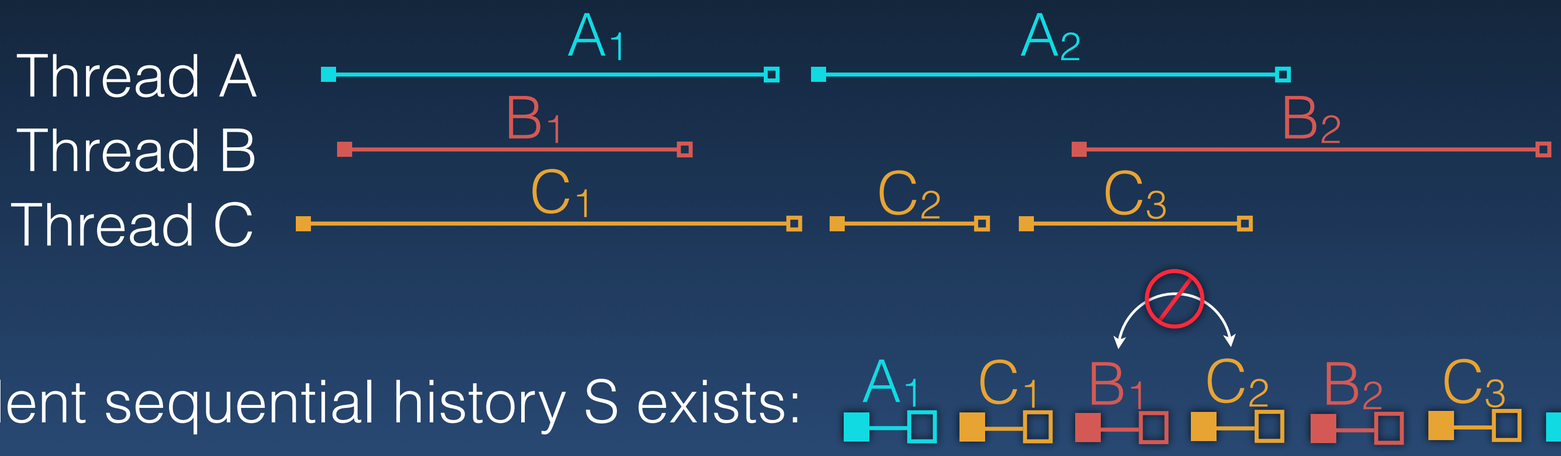
$A_1 \rightarrow A_2$ $C_1 \rightarrow C_2 \rightarrow C_3$ $B_1 \rightarrow B_2$

A_1 $C_1 \rightarrow A_2$ $B_1 \rightarrow B_2 \rightarrow C_2 \rightarrow C_3$

Linearizable

- Operations appear to take effect a some instant

  ➡ between their start and end

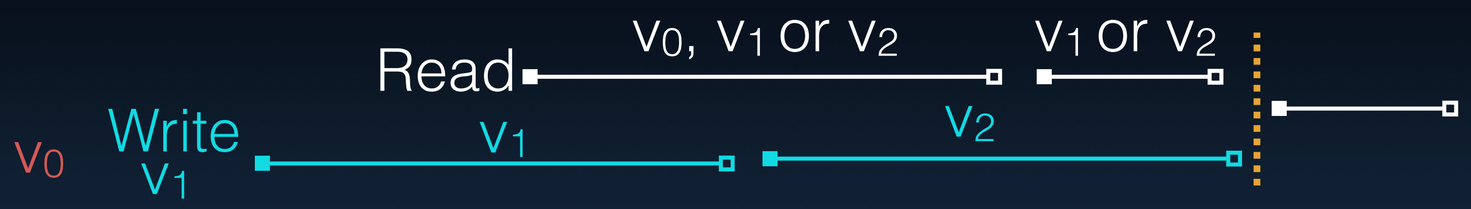


1. Each thread's history is retained in S
2. Non 'overlapping' operations retain order

Composable

Wait free

- **Regular**



  - ➡ Single writer

Quiescently consistent

  - ➡ Reads overlapping with a write return the newly written value or the one prior

- **Atomic**

A later read cannot see an earlier write

  - ➡ Multiple writers

Linearizable

  - ➡ Read the value of latest write (*takes effect* at some point in its duration)

SRSW Binary Safe Register → SRSW Binary Regular Register → MRMW *m*-valued Atomic Register

Register Construction

Boolean -> m-valued

Regular Register

Reg -> Atomic

(Value, Clock)

Writer: Increment Clock
Reader: Cache last read
Reuse, if new value is stale

SW Atomic-> MW Atomic

(Value, Clock) (Value, Clock) •• (Value, Clock)

Writers: Write in their lane,
updating clock to latest+1
Readers: Read the value with latest clock

SRSW Binary Safe Register → SRSW Binary Regular Register → MRMW *m*-valued Atomic Register

Subodh Kumar

Register Snapshot

MRSW m-valued Atomic Registers

Writers: Write <value, clock++>
Snapshot: Read all values twice
Same timestamps ⇒ done!

Obstruction Free

~~Wait free?~~

Writers: Make Snapshot
Write <value, clock++, snapshot>

Snapshot:

```
Copy1 = Sequential Copy of all Register-sets
Set wrote_i = false for all Registers i
Repeat until done:
    Copy2 = Sequential Copy of all Register-sets
    Sequentially compare times of Copy1 and Copy2:
        if(Copy1.clock_i != Copy2.clock_i)
            if wrote_i:  return Copy2.Snapshot_i
            else: wrote_i  = true; Copy1 = Copy2
return Copy1
```

Subodh Kumar