Distributed Systems essentials: Consistency models and Consensus

Some slides from Michael Freedman and others

Replication

- Motivation
  - Performance Enhancement
  - Enhanced availability
  - Fault tolerance
  - Scalability
    - tradeoff between benefits of replication and work required to keep replicas consistent
- Requirements
  - Memory Consistency (*not* Consistency in ACID)
    - Depends upon application
    - Requests for different replicas of the same logical data item should not obtain different results
  - Replica transparency desirable for most applications

Consistency Models

- Consistency Model is a contract between processes and a data store
  - if processes follow certain rules, then store will work "correctly"
- Needed for understanding how concurrent reads and writes behave with respect to shared data
- Relevant for shared memory multiprocessors
  - cache coherence algorithms; memory consistency models
- Shared databases, files
  - independent operations
  - transactions

What is consistency?

- Consistency model:
  - A constraint on the system state observable by applications
- Examples:
  - Local/disk memory :

Single object consistency, also called "coherence"

write x=5

read x (should be 5)

time

  - Database:

x:=x+1; y:=y-1

assert(x+y==const)

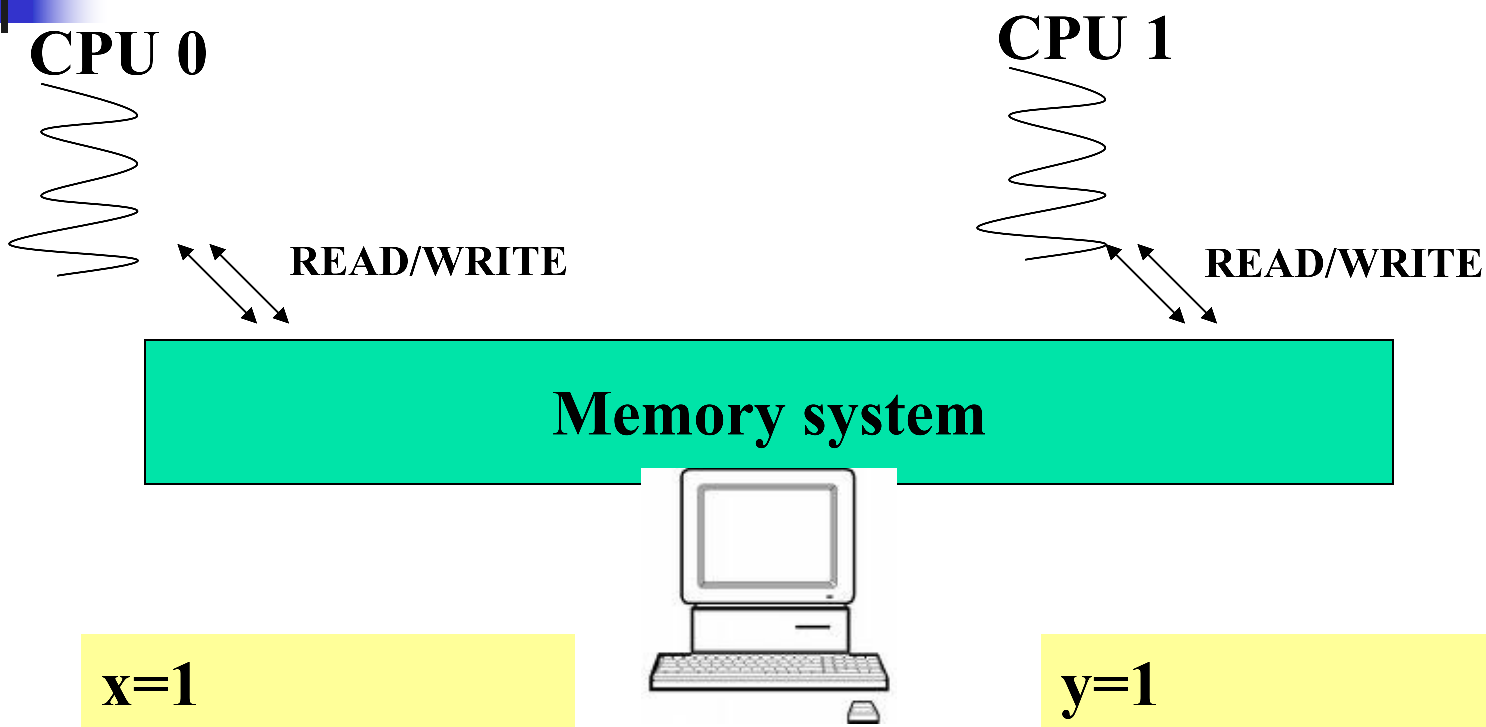
Consistency across >1 objects

time

Consistency challenges

- Consistency is hard in (distributed) systems:
  - Data replication (caching)
  - Concurrency
  - Failures
- No right or wrong consistency models
  - Tradeoff between ease of programmability and efficiency

Example application program




```
x=1
If y==0
  critical section
```

```
y=1
If x==0
  critical section
```

- Is this program correct?

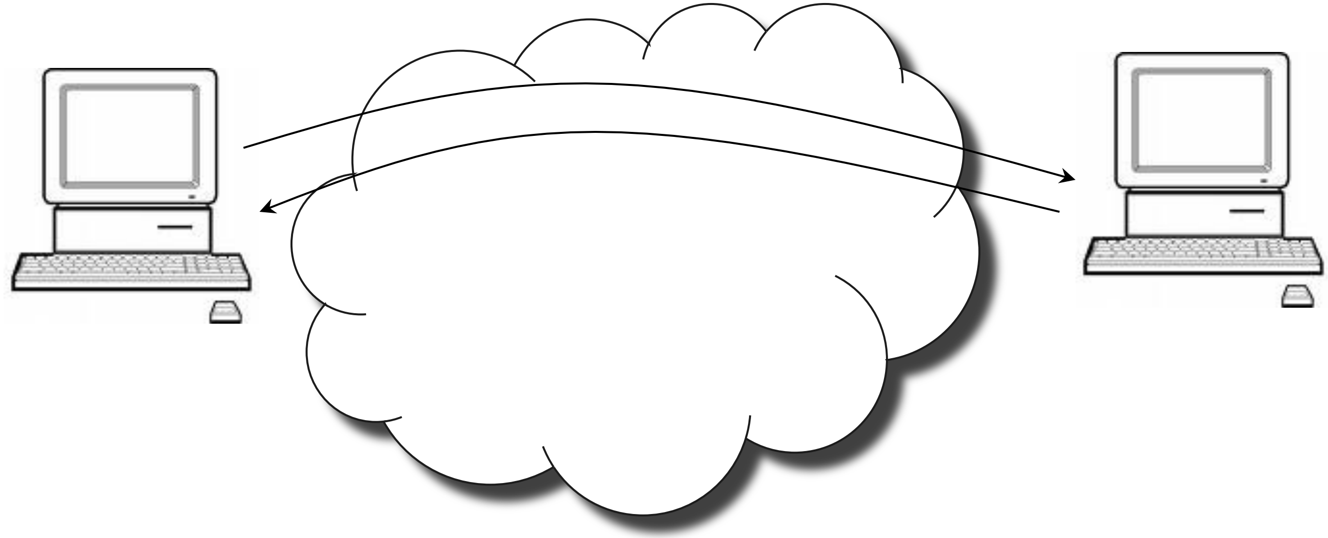
Example

x=1
If y==0
critical section

y=1
If x==0
critical section

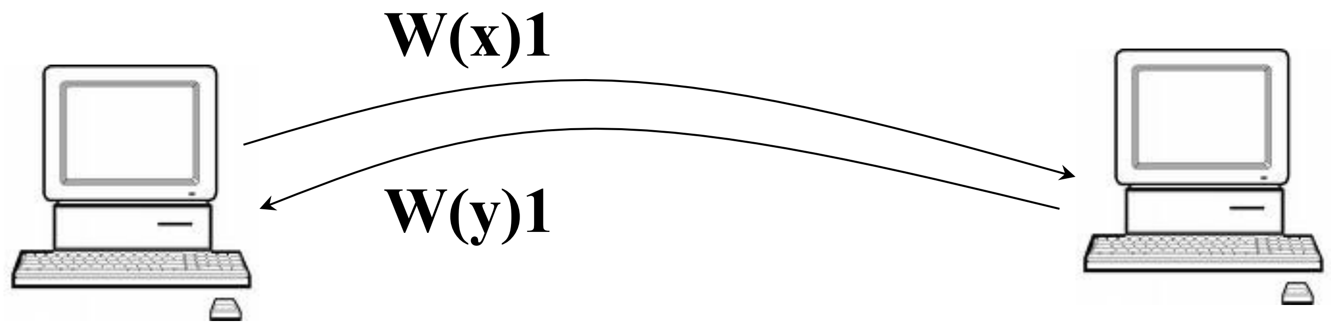
- CPU0's instruction stream: W(x) R(y)
- CPU1's instruction stream: W(y) R(x)
- Enumerate all possible inter-leavings:
  - W(x)1 R(y)0 W(y)1 R(x)1
  - W(x)1 W(y)1 R(y)1 R(x)1
  - W(x)1 W(y)1 R(x)1 R(y)1
  - ....
- None violates mutual exclusion

An example distributed shared memory



- Each node has a local copy of state
- Read from local state
- Send writes to the other node, but do not wait

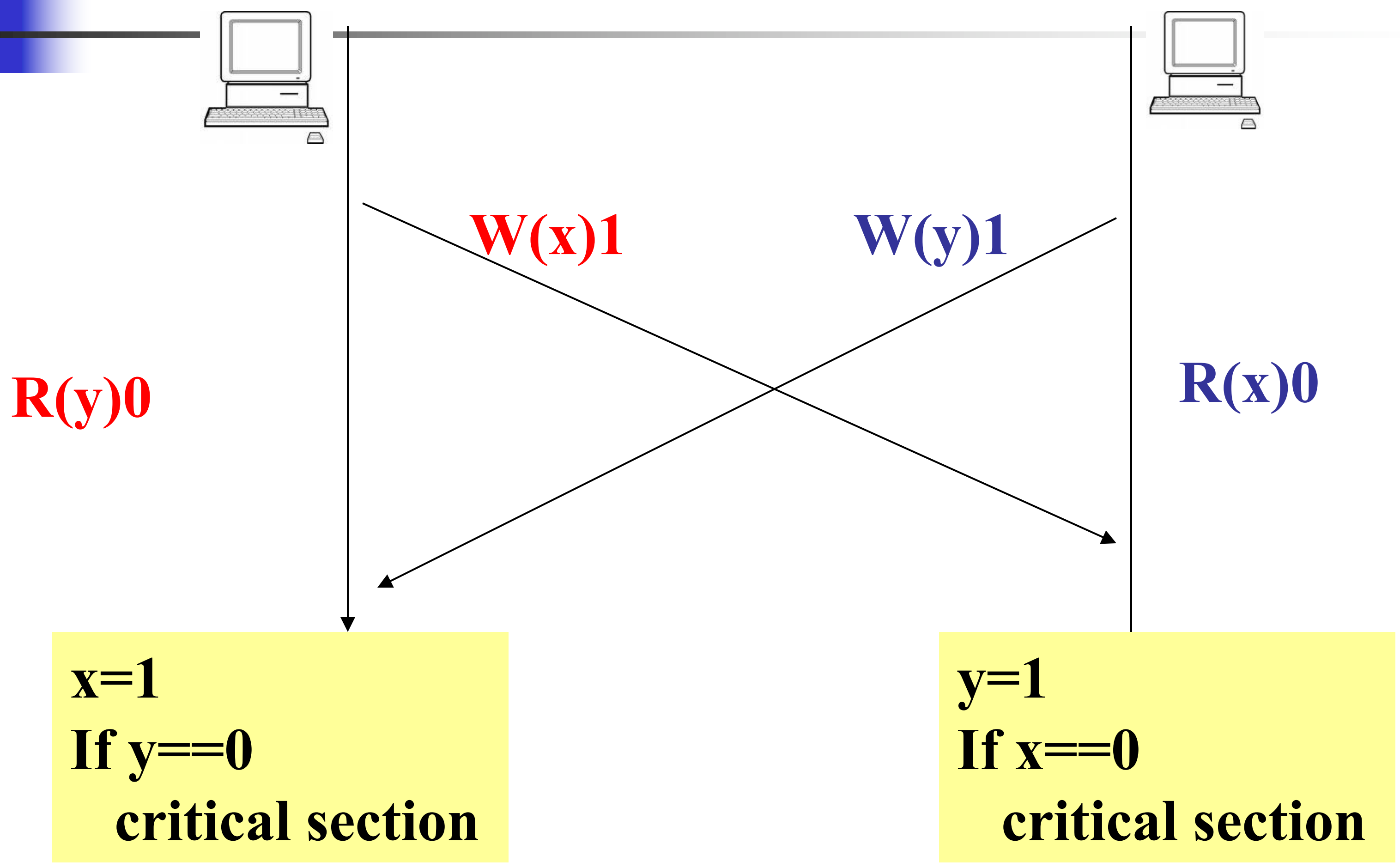
Consistency challenges: example



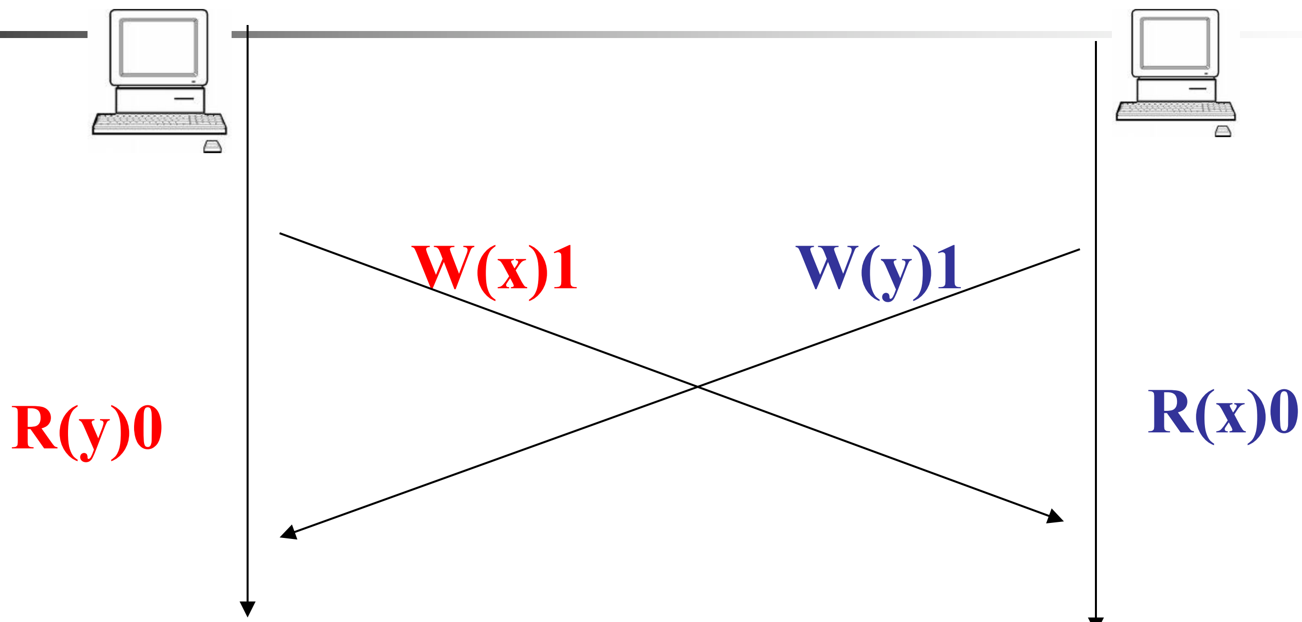
x=1
If y==0
critical section

y=1
If x==0
critical section

Does this work?



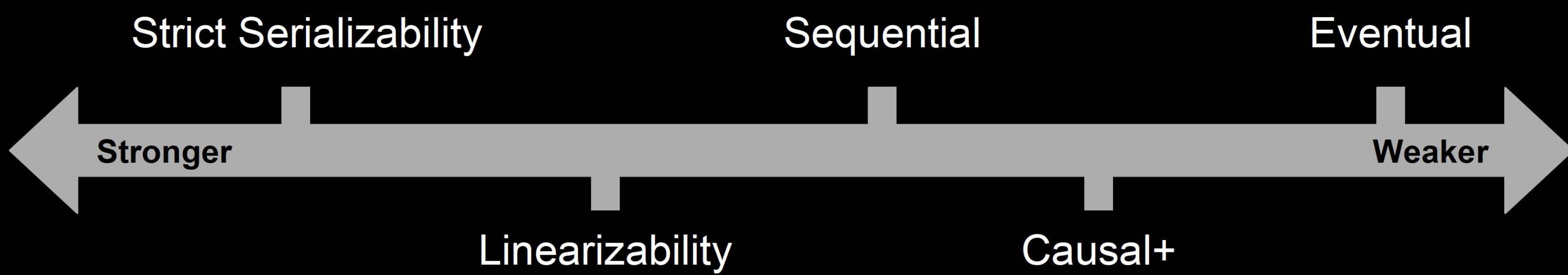
What went wrong?



CPU0 sees:
W(x)1
R(y)0

CPU1 sees:
W(y)1
R(x)0

Consistency Models



Linearizability example

Linearizable?

P1:	W(x)a			
P2:		W(x)b		
P3:			R(x)a	R(x)b
P4:			R(x)a	R(x)b

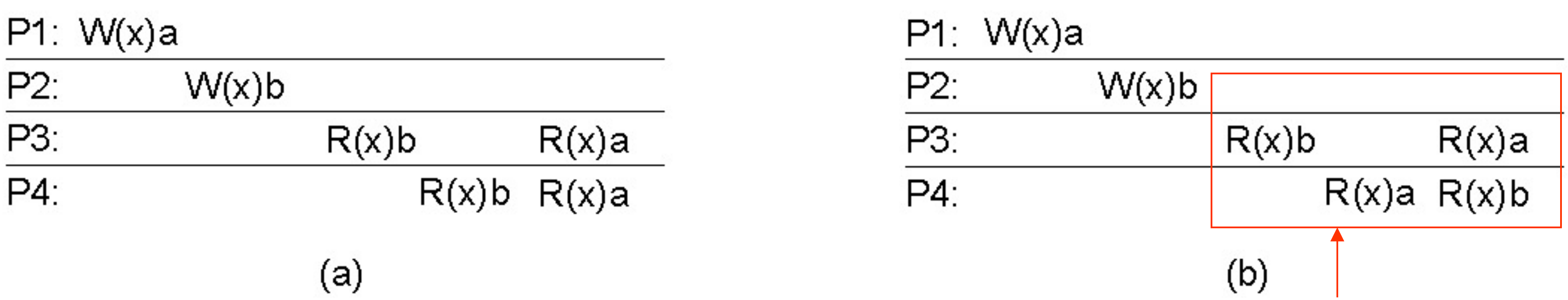
Linearizable?

P1:	W(x)a			
P2:		W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)b	R(x)a

Sequential Consistency - 1

Sequential consistency: the result of any execution is the same as if the read and write operations by all 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].

Note: Any valid interleaving is legal but all processes must see the same interleaving.



P3 and P4 disagree on the order of the writes

a) A sequentially consistent data store.

b) A data store that is not sequentially consistent.

Sequential Consistency - 2

Process P1	Process P2	Process P3
x = 1; print (y, z);	y = 1; print (x, z);	z = 1; print (x, y);

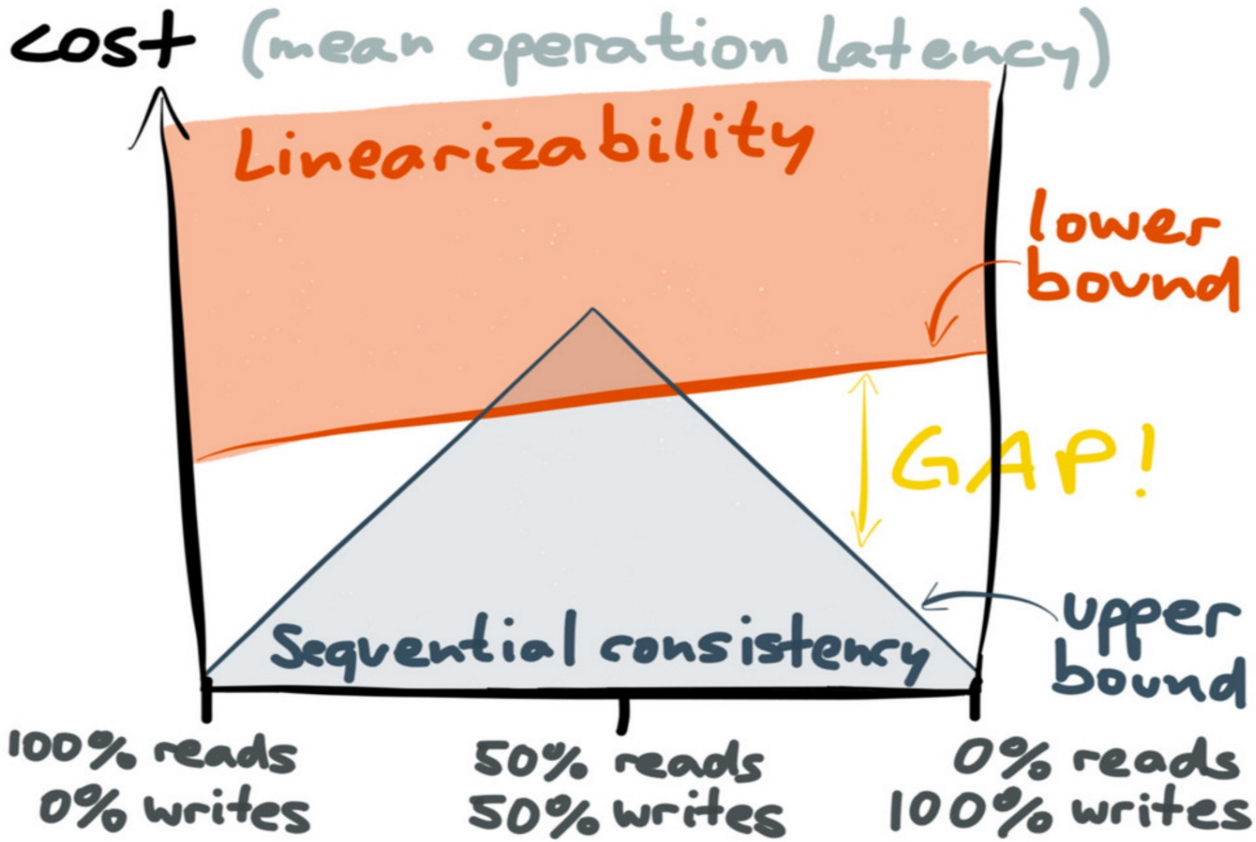
(a)	(b)	(c)	(d)

`x = 1;`
`print (y, z);`
`y = 1;`
`print (x, z);`
`z = 1;`
`print (x, y);`
	`x = 1;`
	`y = 1;`
	`print (x,z);`
	`print(y, z);`
	`z = 1;`
	`print (x, y);`
		`y = 1;`
		`z = 1;`
		`print (x, y);`
		`print (x, z);`
		`x = 1;`
		`print (y, z);`
			`y = 1;`
			`x = 1;`
			`z = 1;`
			`print (x, z);`
			`print (y, z);`
			`print (x, y);`

Prints: 001011	Prints: 101011	Prints: 010111	Prints: 111111

(a)-(d) are all legal interleavings.

Linearizability vs. Sequential Consistency

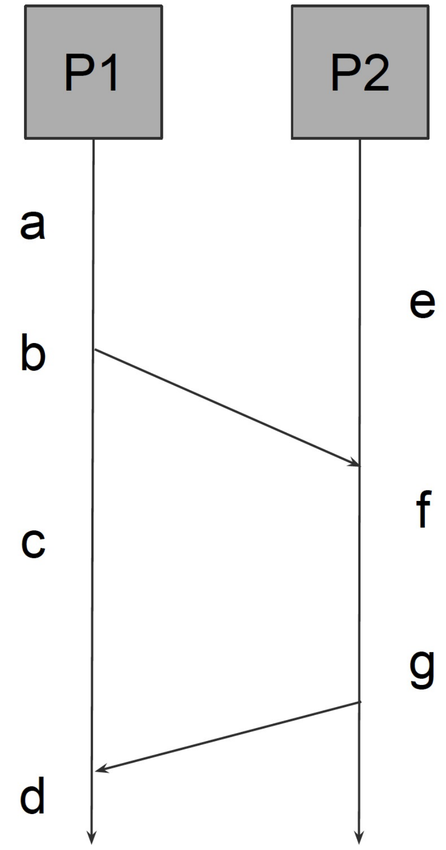


Causal+ consistency

- Partial order: only causally related ops seen the same order everywhere
  - + means replicas eventually converge
  - Concurrent ops may be ordered differently
- Pro: better performance/concurrency
- Cons: Need to reason about concurrency

Causal consistency

Ops	Concurrent
a,b	
a,e	
a,g	
c,e	
c,d	
d,g	
d,f	
e,g	
a,d	



In a nutshell

- Strict consistency: total order, real-time guarantees over transactions
- Linearizability: total order, real-time guarantees over operations
- Sequential consistency: total order + program order
- Causal+ consistency: Causally ordered operations (+ refers to eventual agreement)
- Eventual consistency: we eventually agree

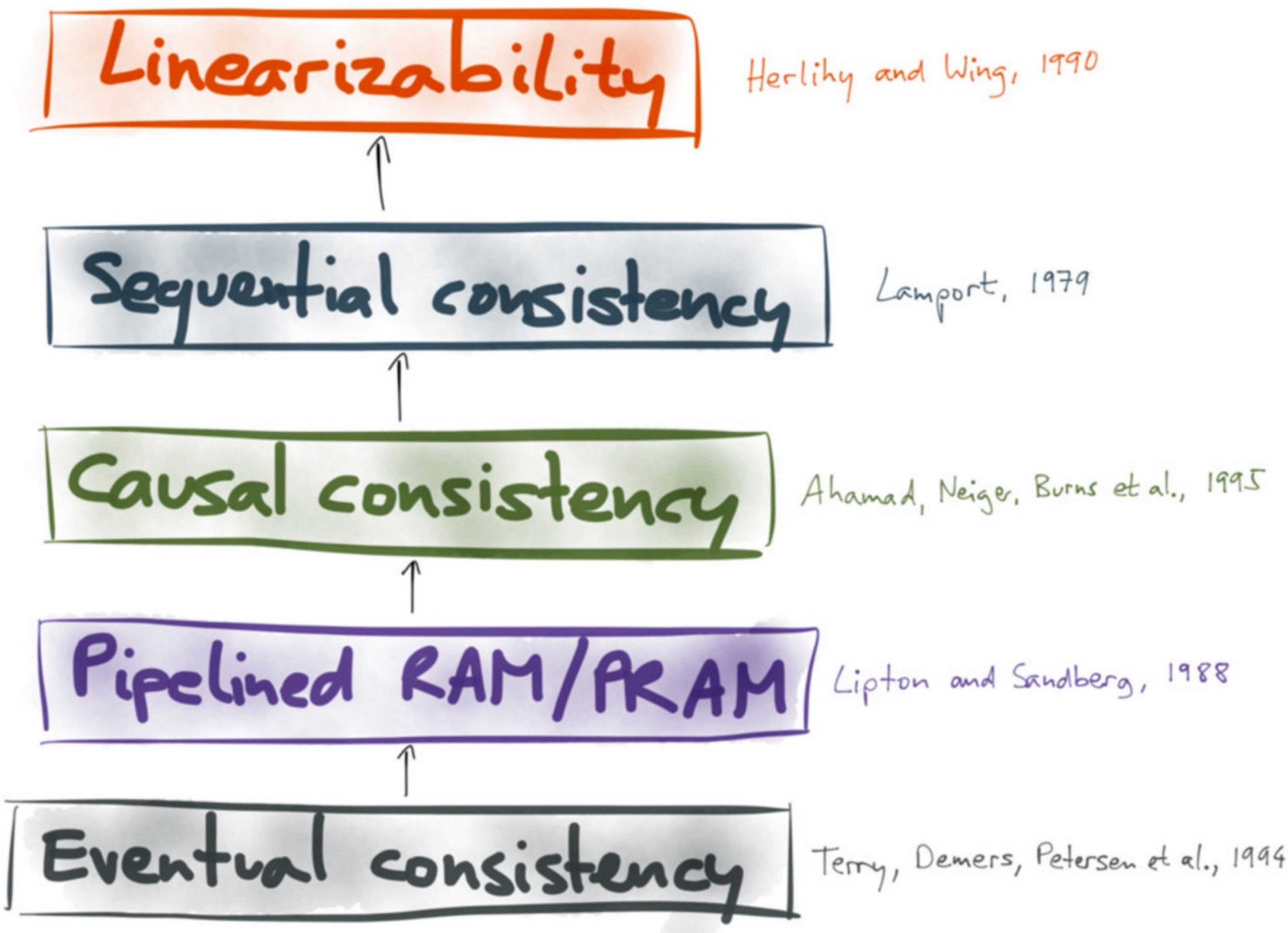


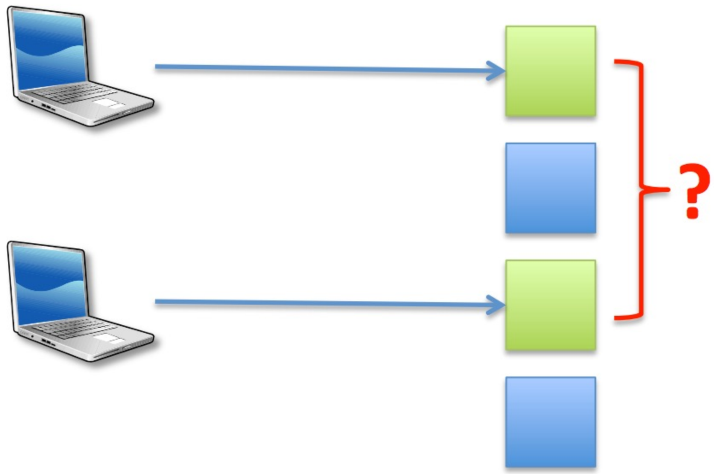
Figure from Martin Klepmann

CAP theorem

- Consistency, Availability, Partition resilient
  - Can only have two of three
  - Proposed by Brewer (2000), and proved/formalized by Gilbert and Lynch (2002)
- General/intuitive but not precise
  - Brewer in 2012: "Misleading because it tended to oversimplify the tension between the properties" https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed
- Still CAP was influential
  - BASE vs. ACID, NoSQL, …

CONSENSUS INTRO

What do clients see?



- Distributed stores use replication
  - Fault tolerance and scalability
  - Does replication necessitate inconsistency?
    - Harder to program, confusing for clients

Problem

- How to reach consensus/data consistency in distributed system that can tolerate non-malicious failures?
- We saw some consistency models – how to implement them?

Another perspective

- Lock is the easiest way to manage concurrency
  - Mutex and semaphore.
  - Read and write locks.
- In distributed system:
  - No master for issuing locks.
  - Failures.