Midterm Review

CS 4740: Cloud Computing

Fall 2024

Yue Cheng

University of Virginia

Logistics

- Monday, Oct 21, 3:30 pm – 4:45 pm
  - 75 minutes
  - Open-book, open-notes (you may use class notes, papers, and lab materials)

- Covering topics from lec-1 to lec-9b
  - Concurrency in Go (20%)
  - MapReduce and GFS (20%)
  - Time and clocks (45%)
  - Consistency (15%)

- Question types
  - High-level design questions
  - Multi-choice questions

Logistics (cont.)

- The exam will be remote
- The exam sheet will be available on `gradescope` at 3:15 pm (you will receive entry code to join the `gradescope` class later this week)
- You should work directly on `gradescope`
- Submission closes at 4:45 pm

Concurrency in Go

- Labs that were completed
  - Possible race condition bugs in labs
  - Go channels

MapReduce and GFS

- Why MapReduce
  - Google workload characteristics
- How MapReduce works
  - Paper
- How data flows within MapReduce and GFS
  - Use of local file system and use of GFS (Fig. 3 sort perf)
- Fault tolerance
  - Backup tasks; task idempotence

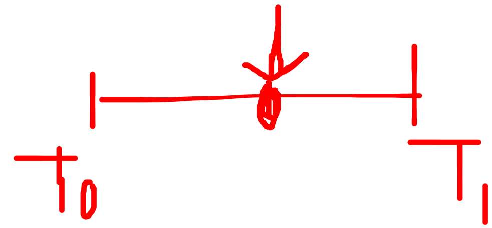
Time and clocks

- Cristian's algorithm

- Lamport Clock algorithm
  - Guarantees if $a \rightarrow b$, then $C(a) < C(b)$
  - How to guarantee a total order of events

- Vector Clock algorithm
  - If $V(a) < V(b)$, then $a \rightarrow b$
  - If $V(a) \not< V(b)$ and $V(b) \not< V(a)$, then $a \parallel b$
  - Can be used to infer when an event b was aware of / influenced by a

Linearizability



- Linearizability specifies that each concurrent operation **appears** to occur **instantaneously** and **exactly once** at some point in time between its invocation and its completion

- Linearizability == "appears to be a single machine"
  - Single machine processes requests one by one in the order it receives them
  - Will receive requests ordered by real-time in that order
  - Will receive requests in some order

Sequential consistency

- Sequential = Linearizability – real-time ordering
  1. All servers execute all ops in *some* identical sequential order
  2. Global ordering preserves each client's own local ordering

- With **concurrent** ops, "reordering" of ops (w.r.t. real-time ordering) acceptable, but all servers must see same order
  – e.g., linearizability cares about time
  sequential consistency cares about program order

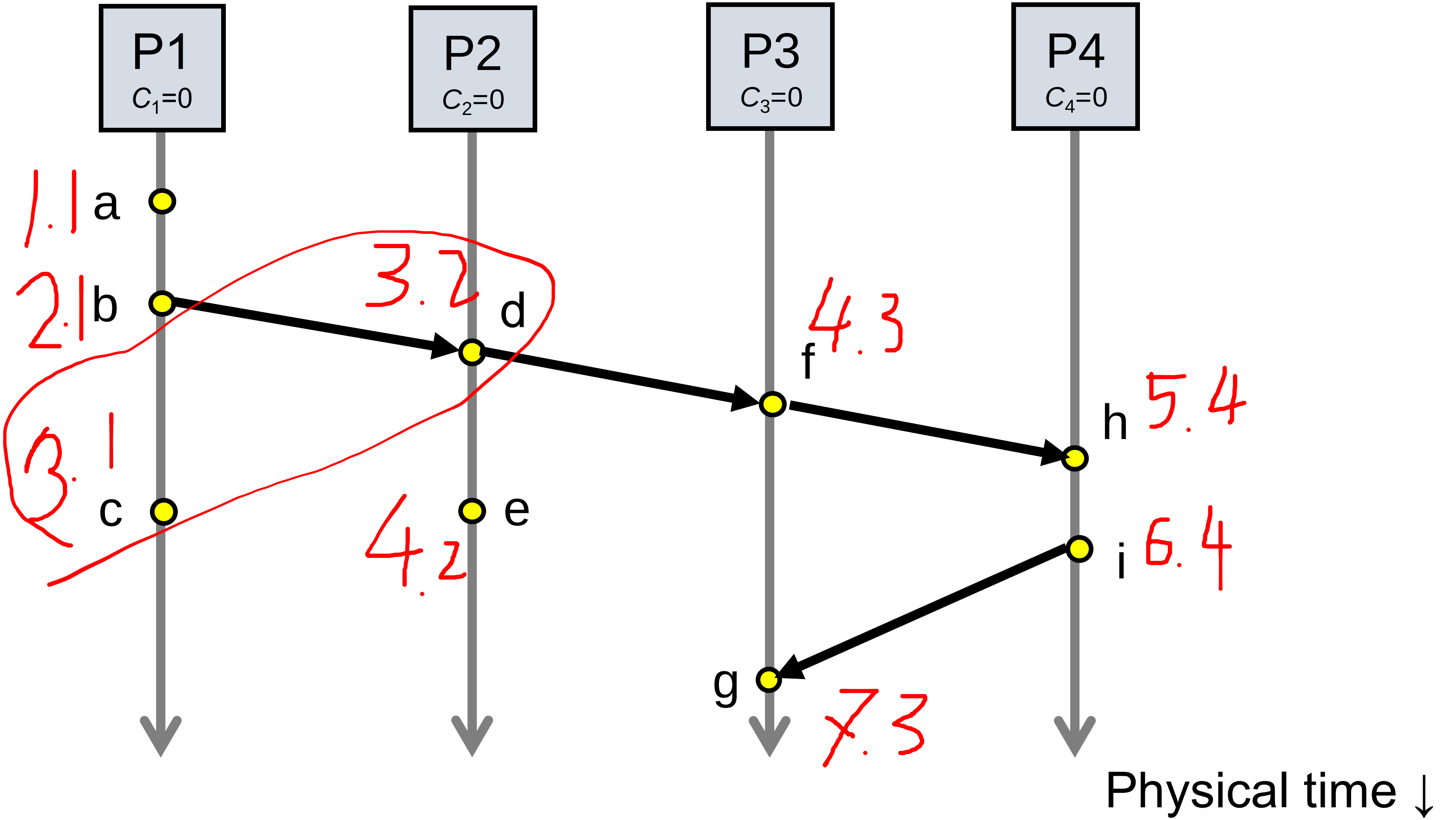
Causal consistency

1. Writes that are *potentially* causally related must be seen by all machines in same order

2. Concurrent writes may be seen in a different order on different machines

Concurrent: Ops not causally related

Quiz 1: Order all these events



Quiz 2: Valid sequence (causal)?

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

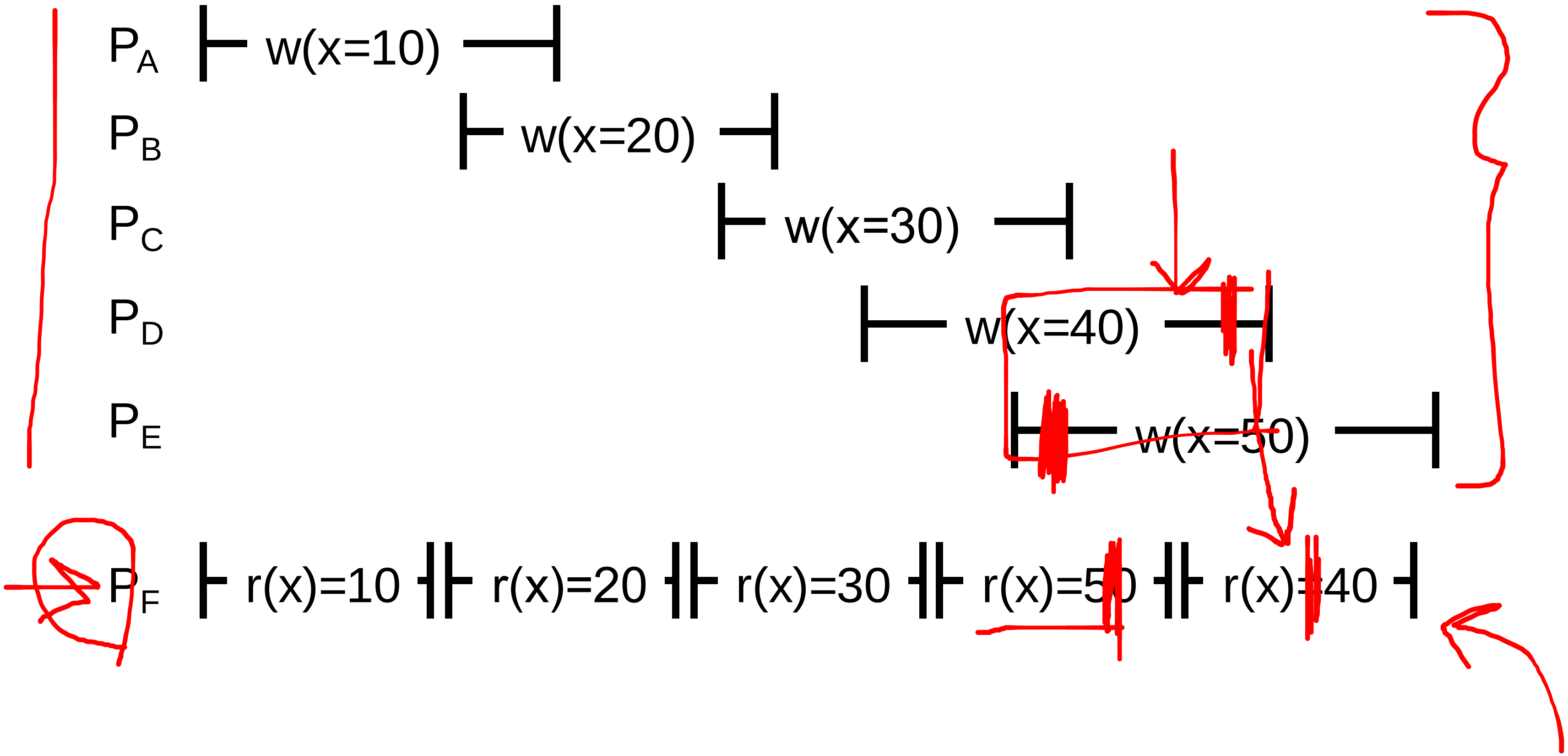
- Valid under causal consistency
- **Why?** *W(x)b* and *W(x)c* are concurrent
  - So all processes don't (need to) see them in same order
- P3 and P4 read the values 'a' and 'b' in order as potentially causally related. No 'causality' for 'c'.

Quiz 2: Valid sequence (sequential)?

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

- Invalid under sequential consistency
- **Why?** P3 and P4 see b and c in different order
- But fine for causal consistency
  - b and c are not causally dependent

More exercises about linearizability



More exercises about linearizability (cont.)