CSCI 1380 : Day 11

Time

Today

Ordering

1. Total ordering
2. FIFO ordering.
3. Causal ordering

Clocks

1. Logical clocks (Cockroach DB / YugaDB)
2. Vector clock (Amazon's Dynamo)

Ordering based on clocks

Global Snapshots

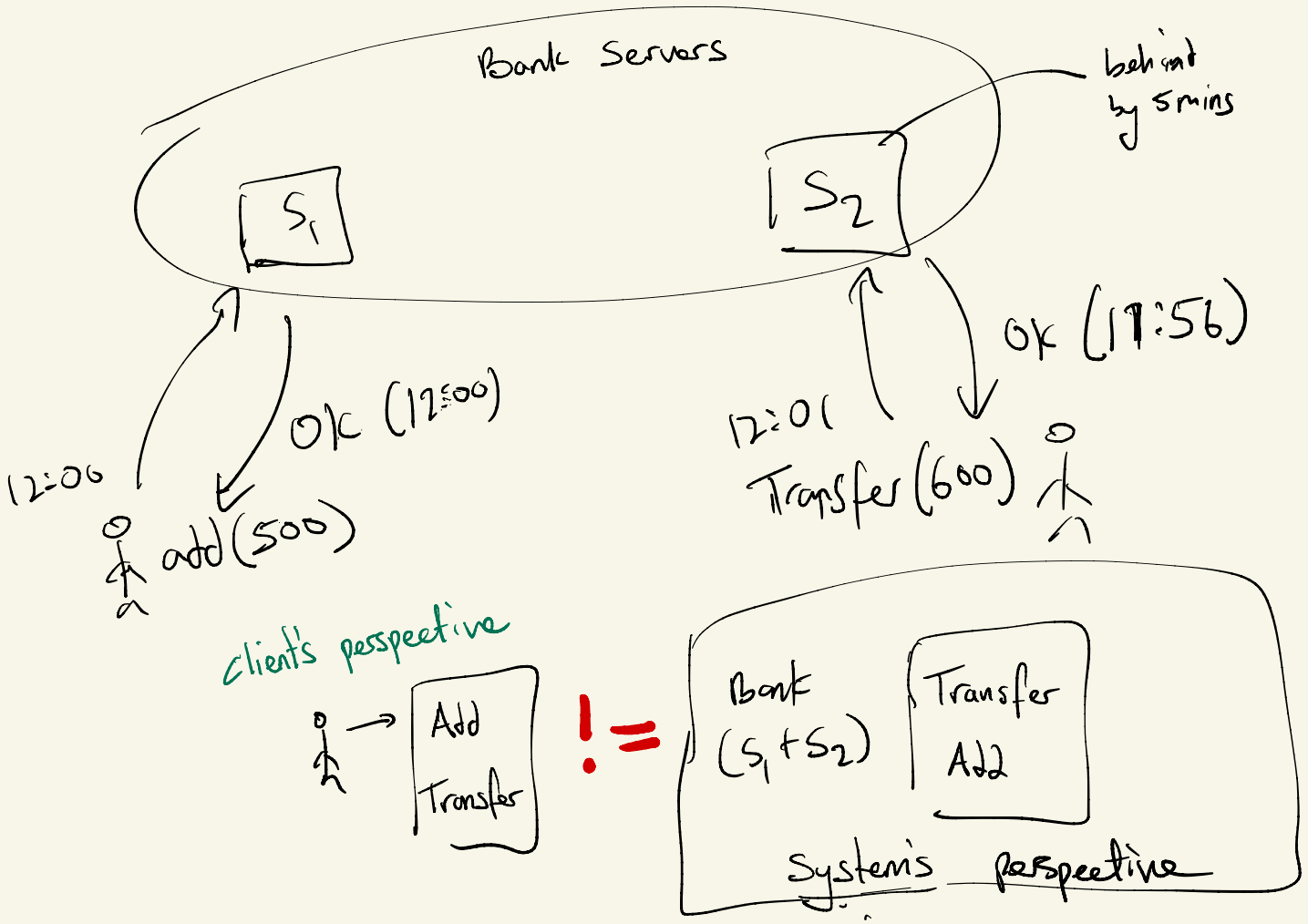
1. Identifying Inconsistent snapshots with Vector clocks

Real Time is Bad

(1) hard to synchronize servers

(2) Time is not monotonic

Lookup the "Leap second" bug

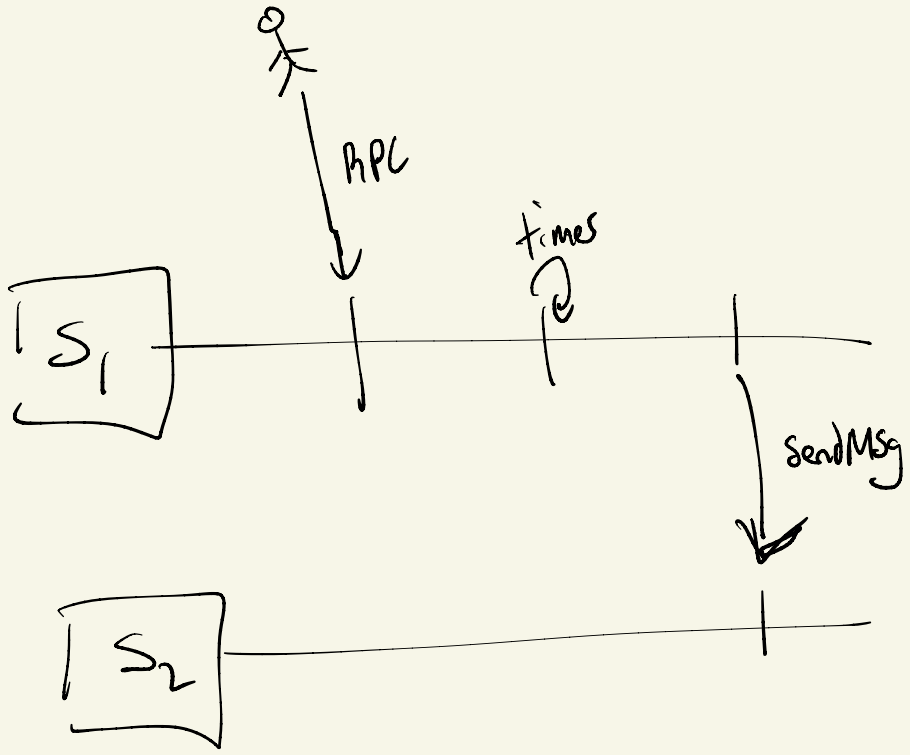


We want a monotonic clock

Event types / Assumptions

event Types

1. Receive
2. local (timer)
3. Send Msg

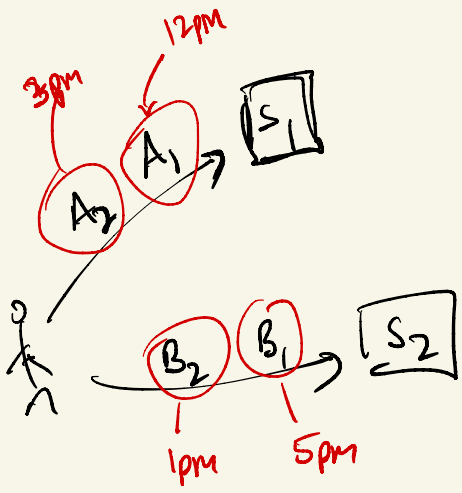


Assumption

1. all servers can dump events to external store (S3)

Type of order (total / FIFO / causal)

Total = every server can place events in same order



S_1: $A_1\ B_1\ A_2\ B_2$ — S_2: $A_1\ B_1\ A_2\ B_2$ ✓

S_1: $B_2\ A_1\ A_2\ B_1$ — S_2: $B_2\ A_1\ A_2\ B_1$ ✓

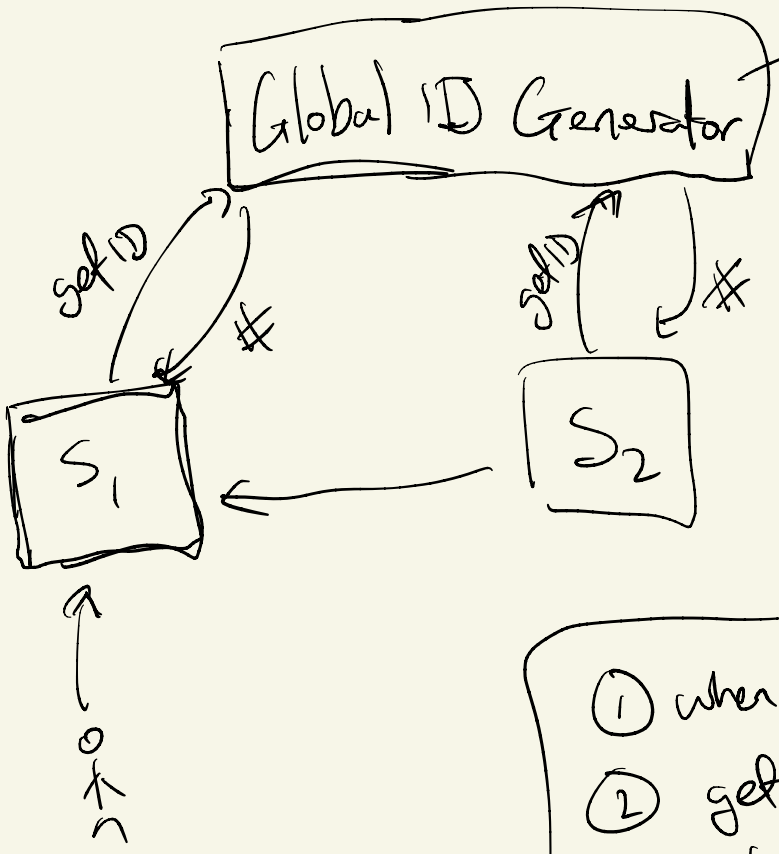
FIFO = every server orders event in order ~~that~~ it was received by the local server: they may not agree on ordering

the servers which got them A_1 & A_2 ∈ S_1

S_1: $A_1\ A_2\ B_1\ B_2$ — S_2: $B_1\ B_2\ A_1\ A_2$ ✓

S_1: $A_1\ B_1\ A_2\ B_2$ — S_2: $A_1\ B_1\ B_2\ A_2$ ✓

Both are FIFO because A_2 is always after A_1 & B_2 is always after A_1



Global server

1. when event happens
2. get ID from global server
3. process event

Time ++

1. use local time
2. find ways to break ties

This is not monotonic!!!

logical clocks address this

rules for logical clocks

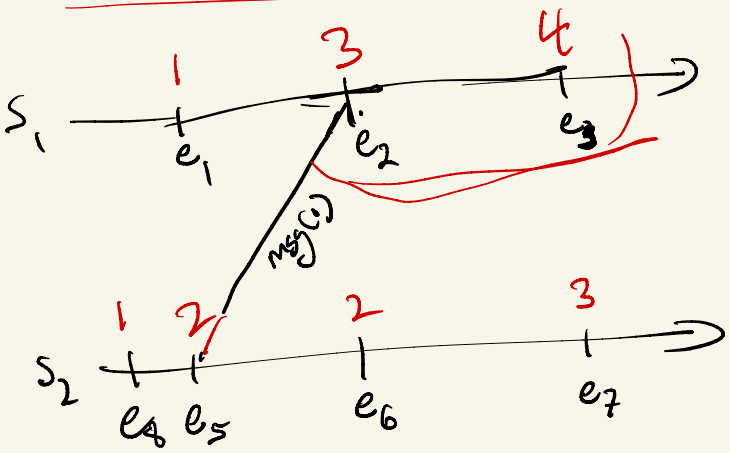
each s maintains a clock (C=0)

if event == (local | sendMsg)

C++ → include clock in msg

if event == recv

C = max(C, msg.C)++



e_1:1
e_2:2
e_3:3
e_5:1
e_6:2
e_7:3

sort based on clock & break ties using server IPs

e_1 e_4 e_5 e_2 ...

Vector Clock

Rules for Vector Clocks

each server maintains a vector of clock ($VC[N]$)

# of servers in system

```
initialize VC[] = 0
if (event == (send | local))
        VC[i]++
        sendMsg(VC)
else if (event == recv)
        for j to N {
        VC[j] = max(VC[j], msg.VC[j])
        }
        VC[i]++
```

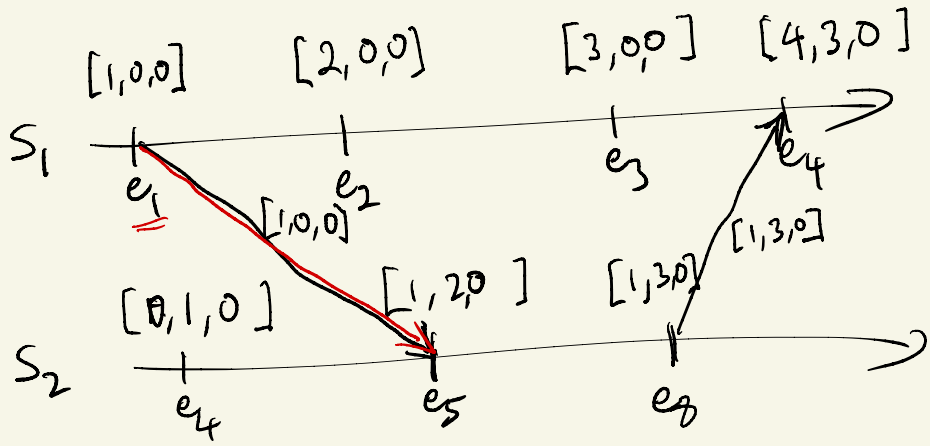
for server 5 VC[5]++

Performance problems

1. maintain VC
2. send VC in each msg

How to determine N (when cluster is dynamic)

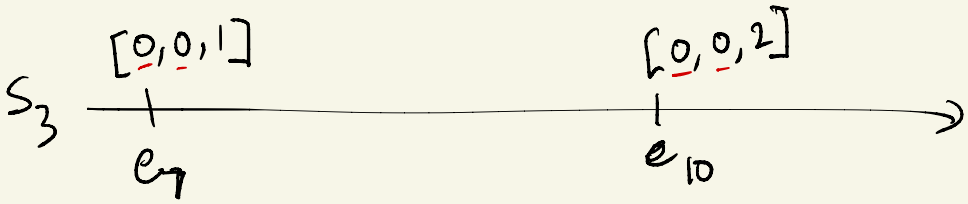
1. virtual nodes (issues with nodes leaving)
2. place a limit on your system (< 1000 nodes) — Kubernetes
3. fix the number of nodes (config with all) — Dynamo



$e_5 = \max([0,1,0], [1,3,0])$

$[1,1,0]$

$[1,2,0]$



$e_4 = \max(e_3, msg_{e8}) = \max([1,3,0], [3,0,0])$

$[3,3,0]$

$e_4 = [4,3,0]$

Observations

without msgs you can't learn about another server's clock

Causal ordering (benefits of VC)

"happens before"

A happens before B] A & B are events

$VC_A \leq VC_B$

all clocks in VC_B must be equal / greater than VC_A
& at least one of VC_B elements must be greater than VC_A

$[1,0,1] \leq [1,0,2]$

greater

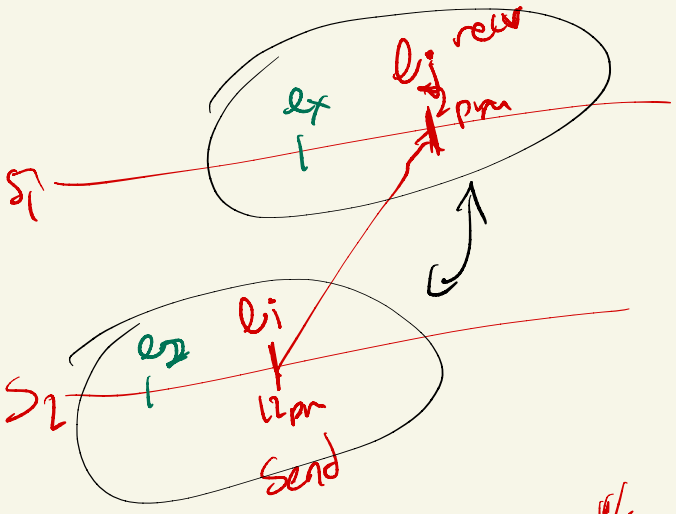
happens before the other

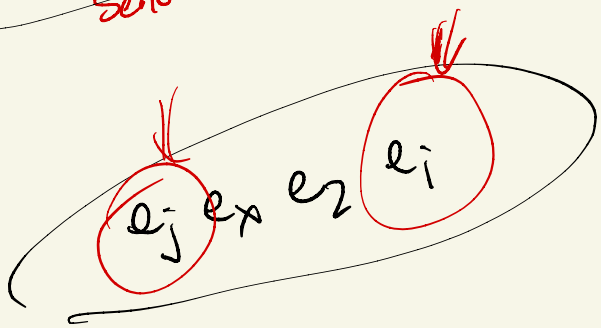
Summary

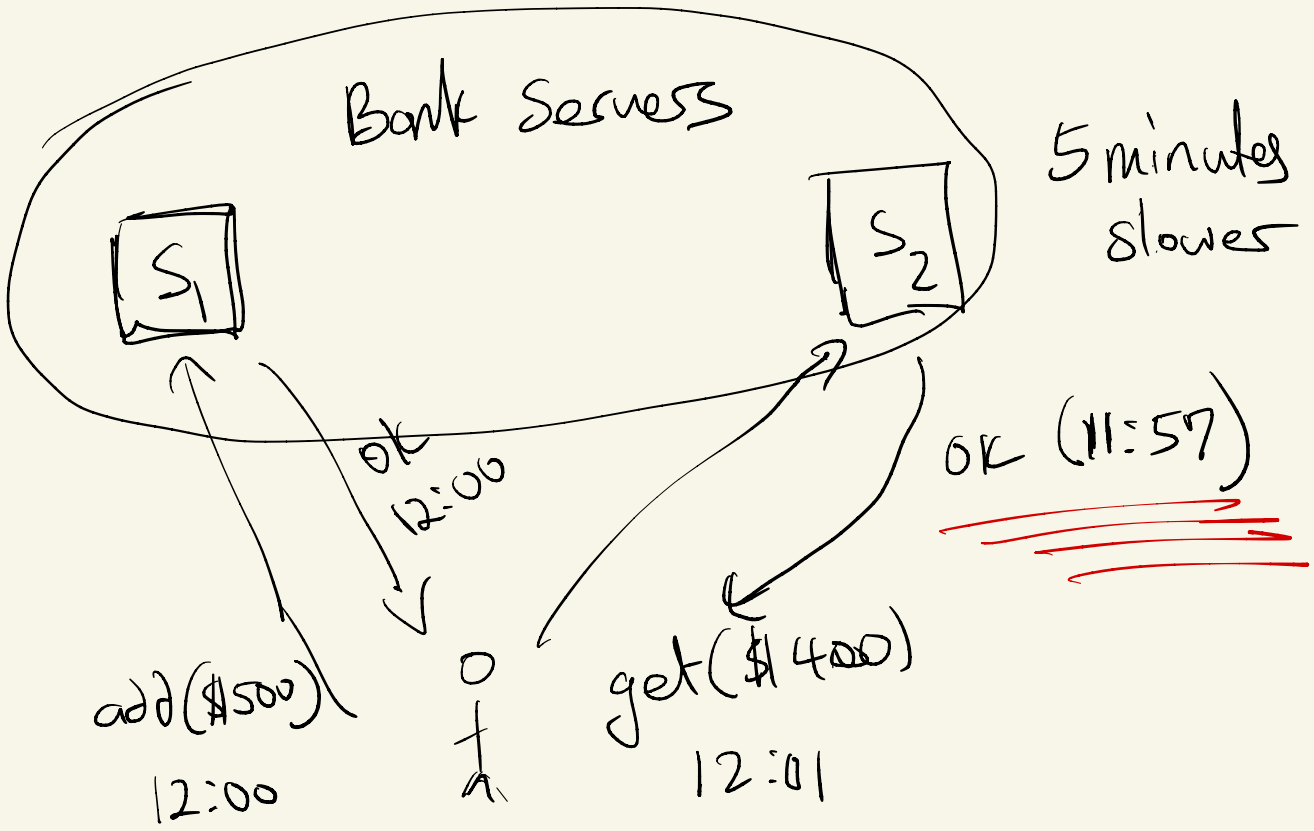
real time is bad (not monotonic)

FIFO / CAUSAL / TOTAL

Logical / Vector clocks







People care about "time"

Distributed systems care about ordering:

add → get
or
get → add

} what is the right ordering