CS4740
CLOUD COMPUTING

Time and Coordination

Prof. Chang Lou, UVA CS, Spring 2024

REMINDER (LAB 1 DUE)

- Deadline: 2/18 (this Sunday)

- If you already finished: keep the branch "lab1" intact
  - We'll collect your codes at deadline (unless you use late tokens)
- If you haven't: keep track of time and implement soon
  - Come to office hours if you need help

GUEST TALK

- **Managing Cloud Health with AIOps**
  - Speaker: Cong Chen, Microsoft Azure
  - Date&Location: 2/21, next Wed class

- Cong Chen is a Principal Data Scientist Manager in Azure Edge and Platform group at Microsoft. He owns model development and quality improvement of a few AIOps solutions. Previously, he was responsible for detection, diagnosis and mitigation of resource leaks on Azure host servers. In this talk, he will share the experience and vision for how AIOps is transforming the way we manage the health of the ever-growing Azure cloud. He will demonstrate how automation and intelligence are crucial for achieving high availability and premier performance, with BRAIN and Gandalf as examples.

https://hoohacks.io/

CONTEXT

- We looked at RPC, a key concept in DS, and saw how failures creep up into semantics and challenge coordination.

- We now look at another key concept in DS, **Time**.
  - Let's see how unbounded network delays (a.k.a. network asynchrony) complicates the very basic concept.

WHY IS TIME IMPORTANT?

WHY IS TIME IMPORTANT?

- Needed for synchronization and coordination.
- Examples:
  - Mutual exclusion
  - Barrier
  - A running (toy) example: distributed debugging based on logs

EXAMPLE: DISTRIBUTED DEBUGGING

M1 (front end)

…
recv from cli
…
send to M2
…
recv from M2
…
send to cli
…

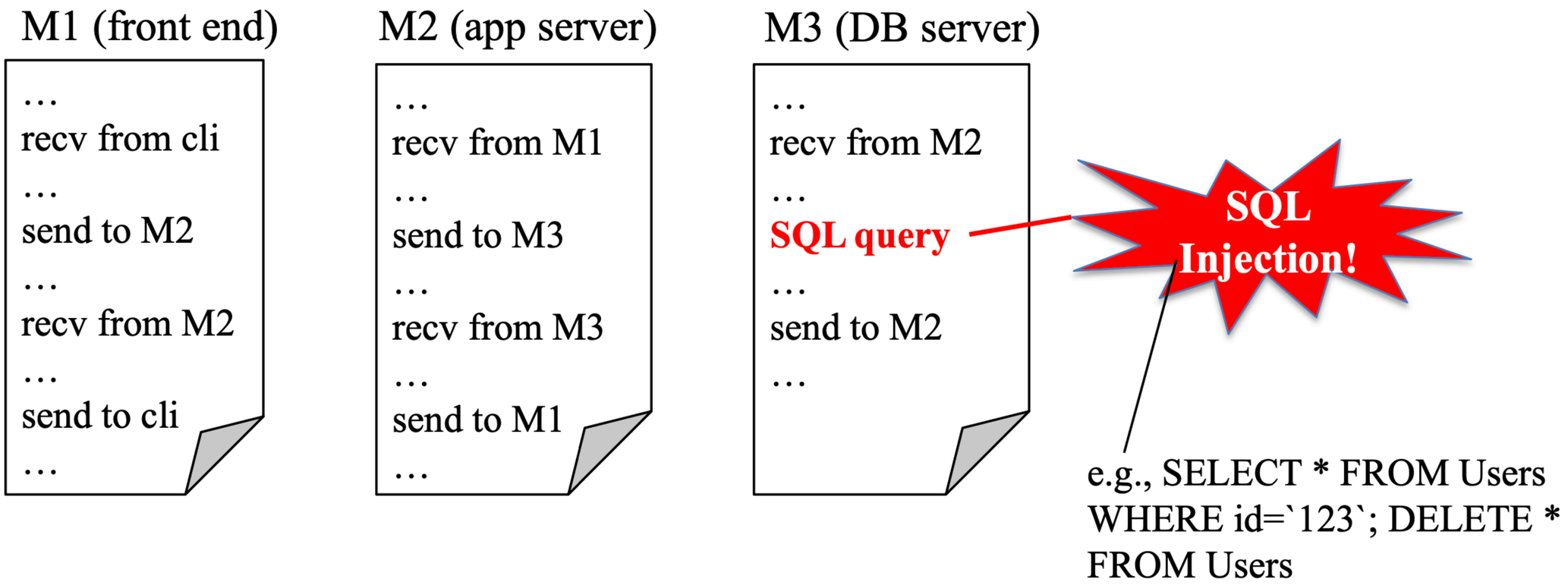
M2 (app server)

…
recv from M1
…
send to M3
…
recv from M3
…
send to M1
…

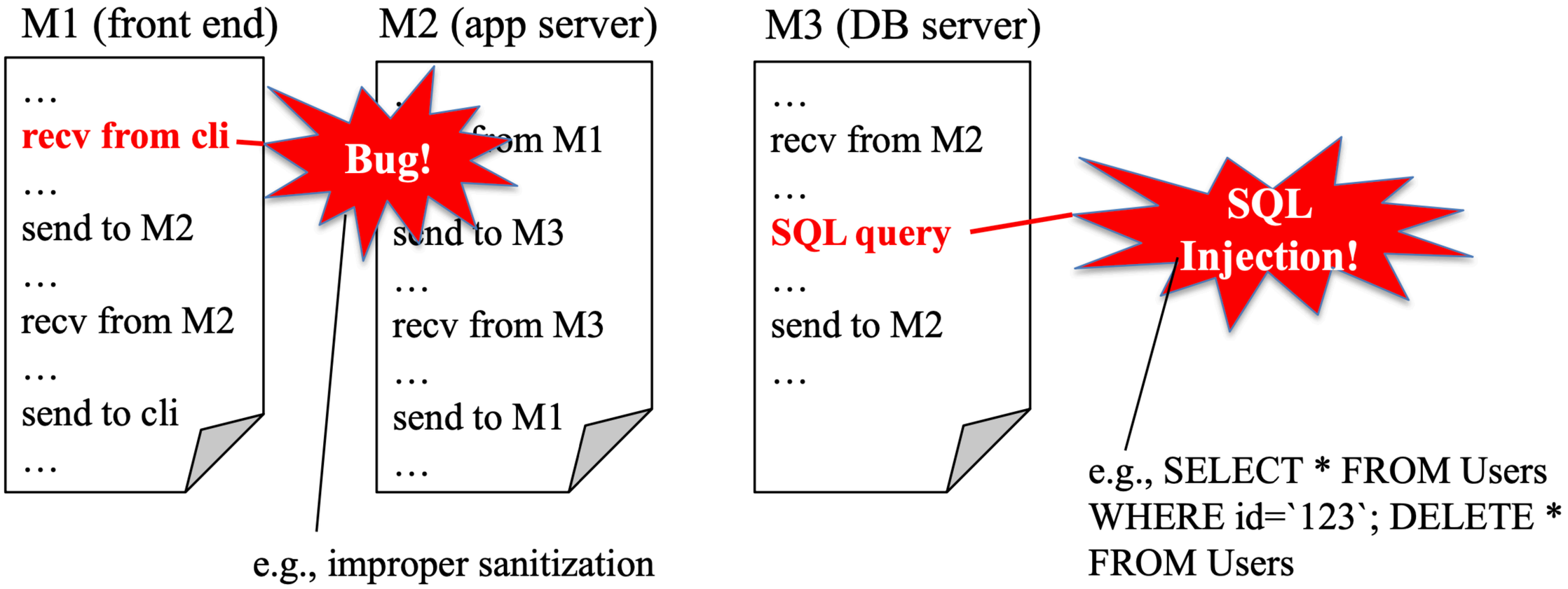
M3 (DB server)

…
recv from M2
…
SQL query
…
send to M2
…

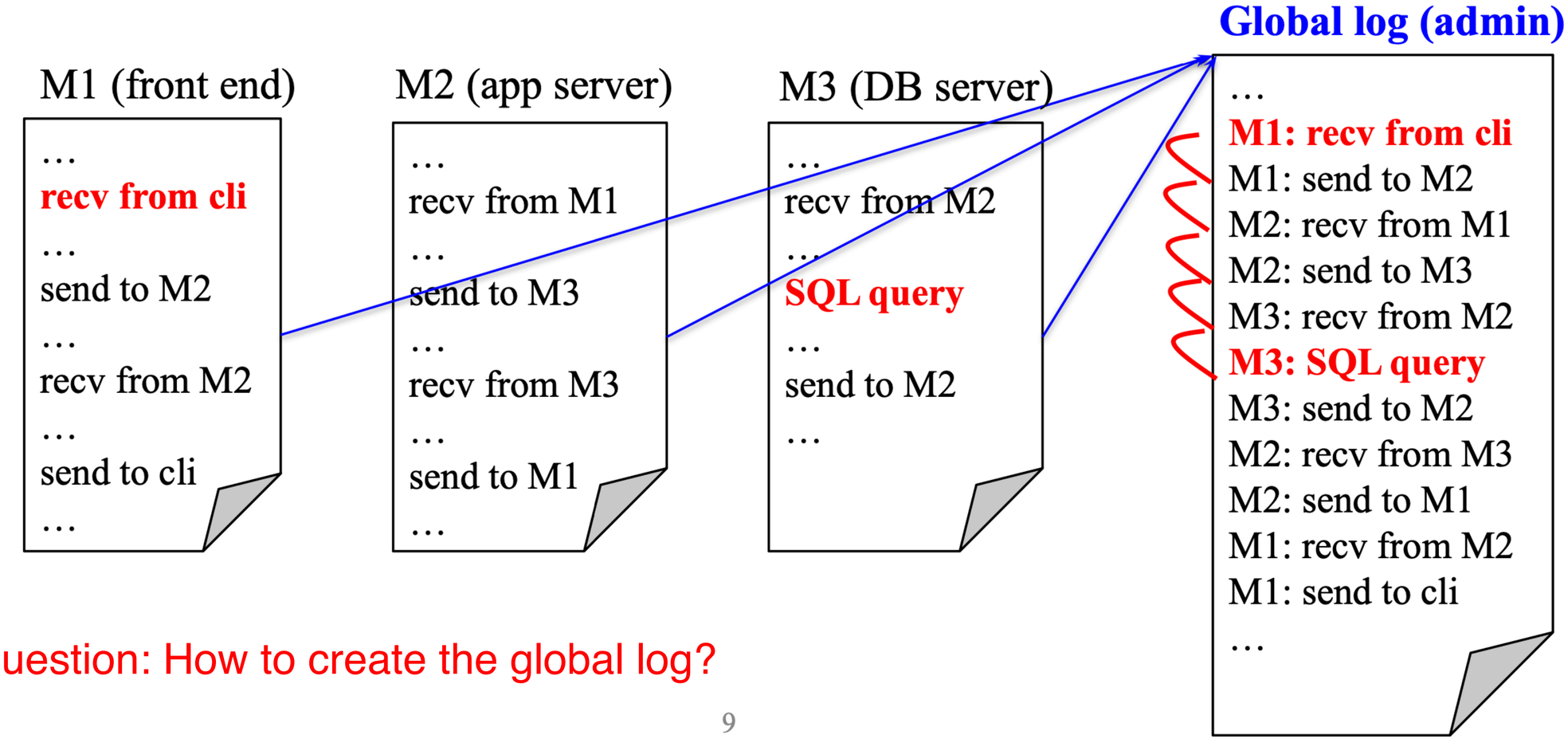
EXAMPLE: DISTRIBUTED DEBUGGING



EXAMPLE: DISTRIBUTED DEBUGGING

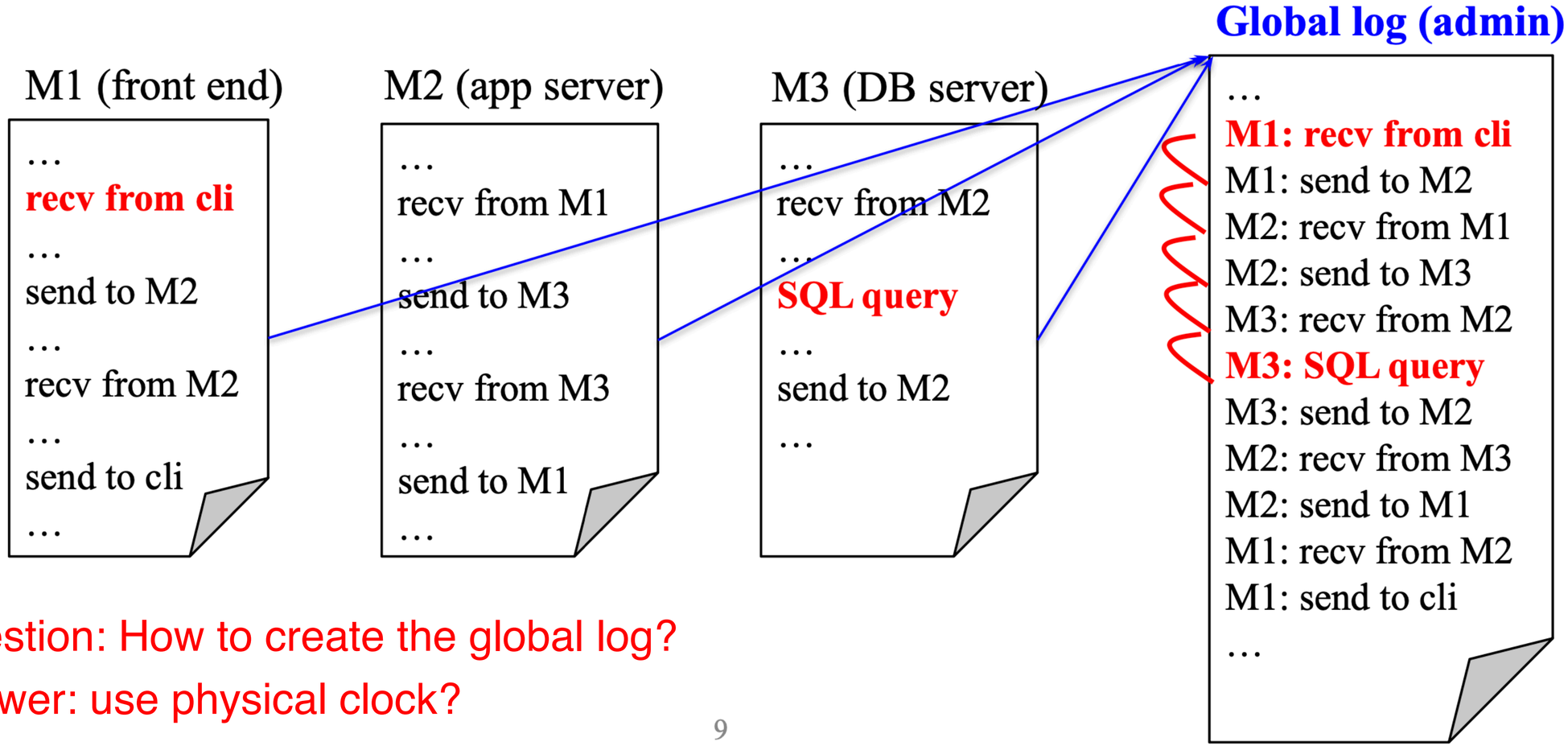


EXAMPLE: DISTRIBUTED DEBUGGING



Question: How to create the global log?

EXAMPLE: DISTRIBUTED DEBUGGING



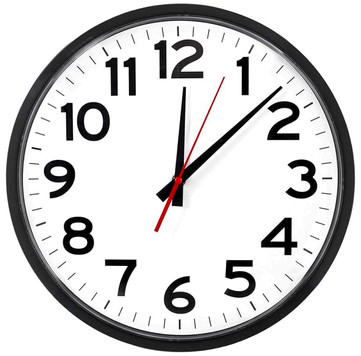
Question: How to create the global log?

Answer: use physical clock?

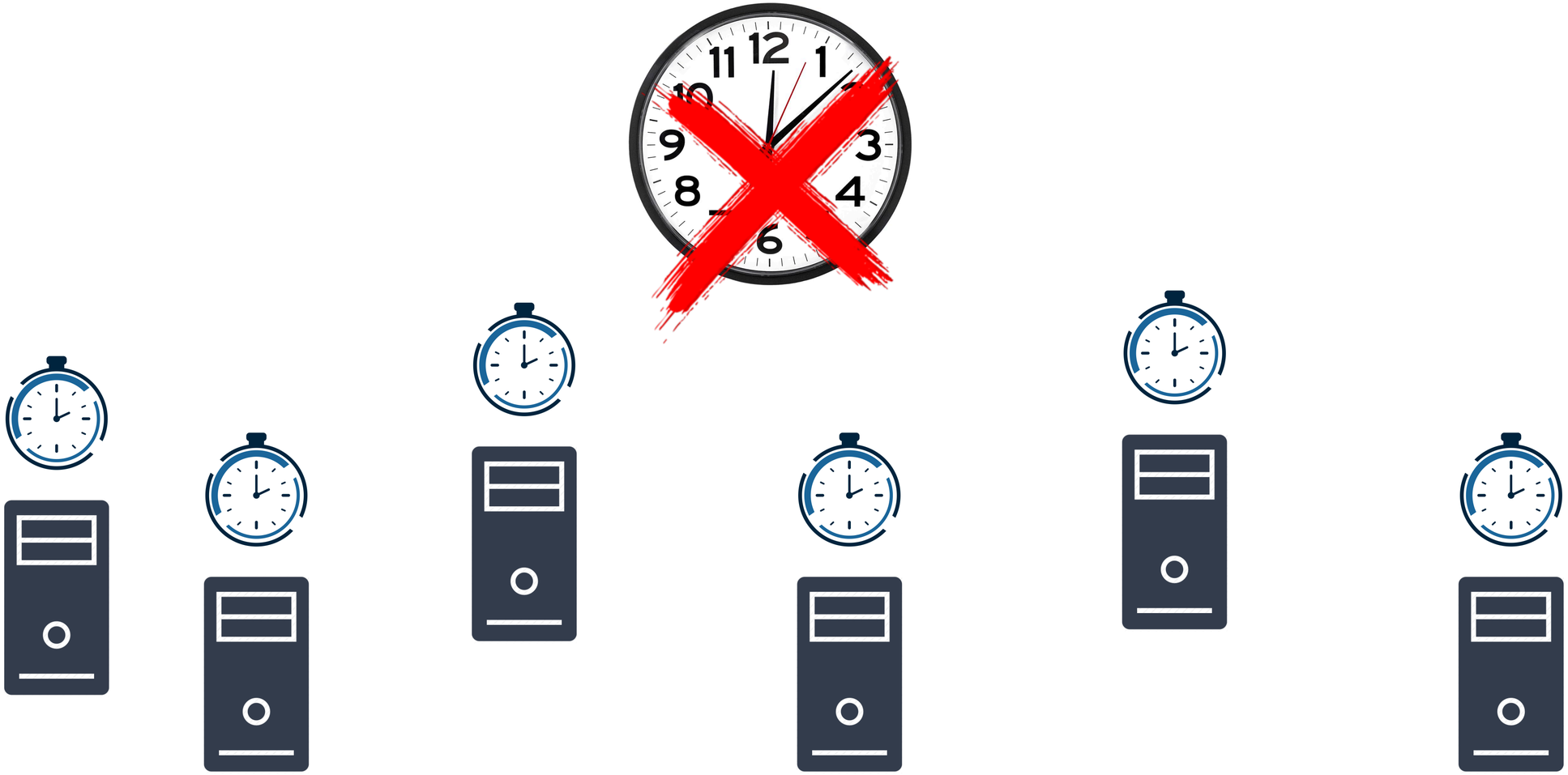
PROBLEM: CLOCK SYNCHRONIZATION IS HARD

- Quartz oscillator sensitive to temperature, age, vibration, radiation
  - Accuracy ~one part per million: one second of clock drift over 12 days

- The network is:
  - Asynchronous: arbitrary message delays
  - Best-effort: messages don’t always arrive


12
1
2
3
4
5
6
7
8
9
10
11

THERE IS NO GLOBAL TIME!



AGENDA

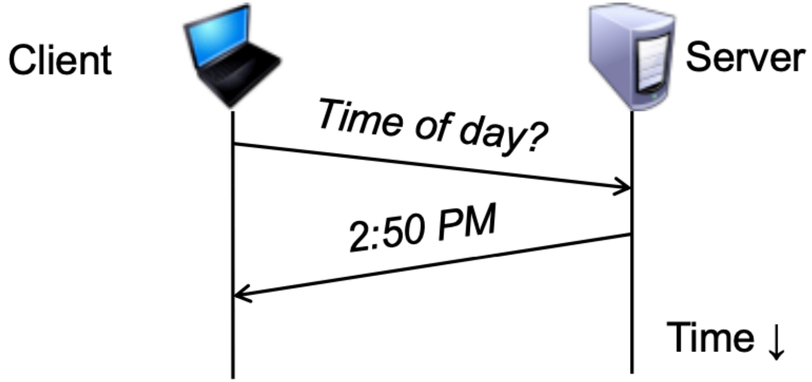
- **Physical clocks**
  - Synchronization challenges and protocols
- Logical clocks
  - Lamport clock protocol

JUST USE COORDINATED UNIVERSAL TIME?

- UTC is broadcast from radio stations on land and satellite (e.g., the Global Positioning System)
  - Computers with receivers can synchronize their clocks with these timing signals
- Signals from land-based stations are accurate to about 0.1–10 milliseconds
- Signals from GPS are accurate to about one microsecond
  - *Why can't we put GPS receivers on all our computers?*

SYNCHRONIZATION TO A TIME SERVER

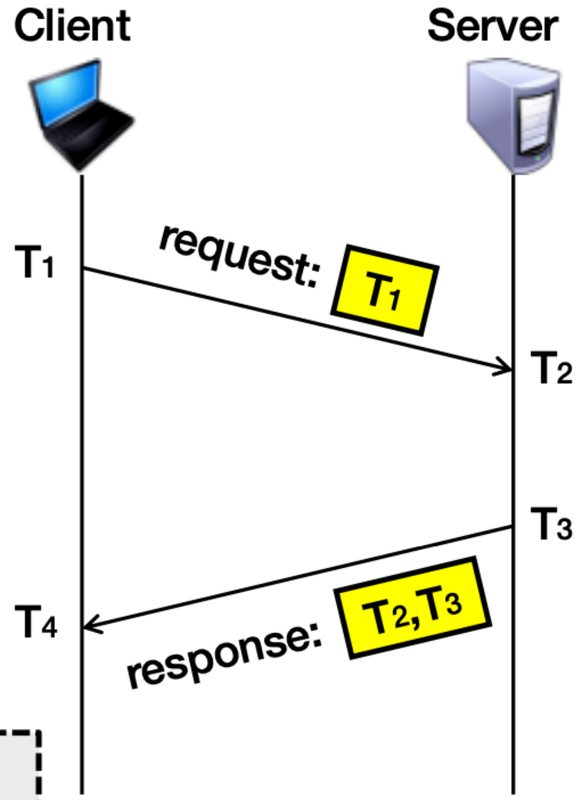
- Suppose a server with an accurate clock (e.g., GPS-receiver)
  - Could simply issue an RPC to obtain the time:



- But this doesn't account for network latency
  - Message delays will have outdated server's answer

CRISTIAN'S ALGORITHM: OUTLINE

- 1. Client sends a request packet, timestamped with its local clock T1
- 2. Server timestamps its receipt of the request T2 with its local clock
- 3. Server sends a response packet with its local clock T3 and T2
- 4. Client locally timestamps its receipt of the server's response T4



How can the client use these timestamps to synchronize its local clock to the server's local clock?

CRISTIAN'S ALGORITHM: OFFSET SAMPLE CALCULATION

Goal: Client sets clock ← $T_3 + \delta_{resp}$

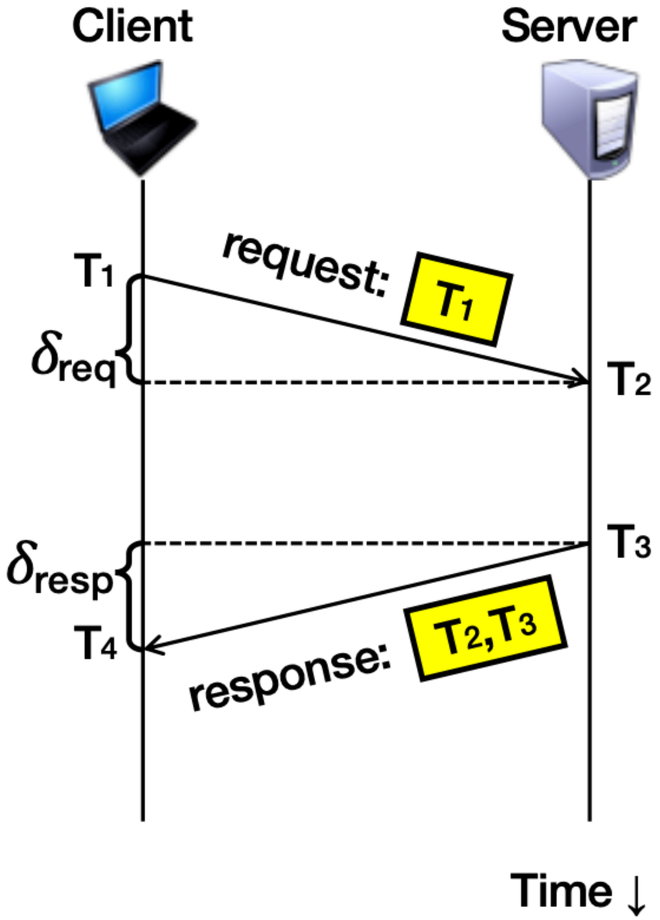
- Client samples round trip time (δ)

  $\delta = \delta_{req} + \delta_{resp} = (T_4 - T_1) - (T_3 - T_2)$

  - But client knows δ, not δ_{resp}

Assume: $\delta_{req} \approx \delta_{resp}$

Client sets clock ← $T_3 + \frac{1}{2}\delta$



CLOCK SYNCHRONIZATION: TAKE-AWAYS

- Clocks on different systems will always behave differently
  - Disagreement between machines can result in undesirable behavior
- NTP clock synchronization
  - Rely on timestamps to estimate network delays
  - 100s μs–ms accuracy
  - Clocks never exactly synchronized
- Often inadequate for distributed systems
  - Often need to reason about the order of events
  - Might need precision on the order of ns

AGENDA

- Physical clocks
  - Synchronization challenges and protocols
- **Logical clocks**
  - Lamport clock protocol

LOGICAL CLOCKS

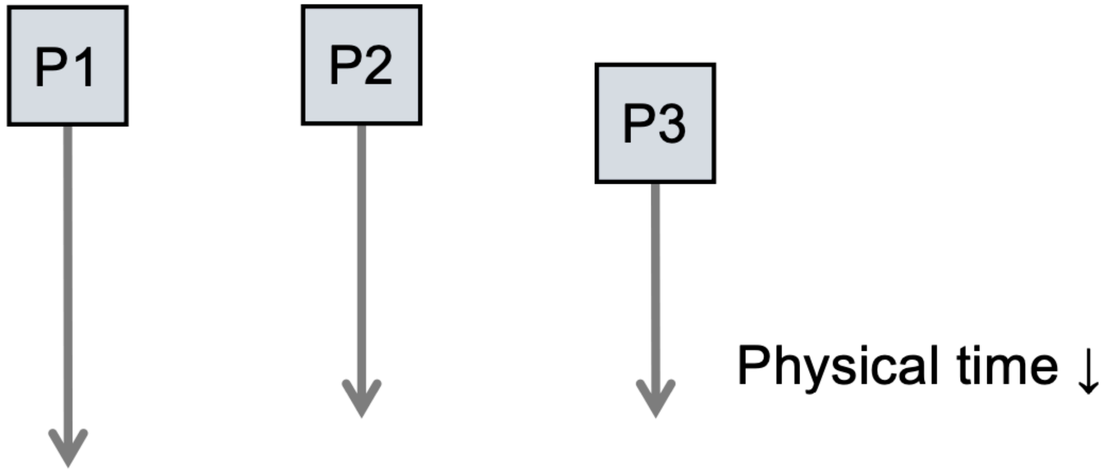
- Leslie Lamport, parent of DS, observed that most coordination in distributed systems (e.g., for mutual exclusion, barriers, complete event log) doesn’t require a global notion of real time!
- Most coordination only needs a global order of discrete events.
- E.g., in the distributed debugging example, you only need order between dependent events that could possibly have caused the failure.
- Achieving a global order of events is easier to guarantee than achieving zero-error real-time synchronization.
- This is why many foundational DS protocols rely on logical clocks.

LOGICAL CLOCK REQUIREMENTS

- Lamport posited two requirements for logical clocks:
  - They must preserve **program order** (i.e., the order of events in one process needs to be preserved by the logical clock)
  - They must preserve **message order** (i.e., a message sent event always needs to precede that message's receipt event in the logical clock).
- These two requirements capture all internal **causality** between any two events in the system.

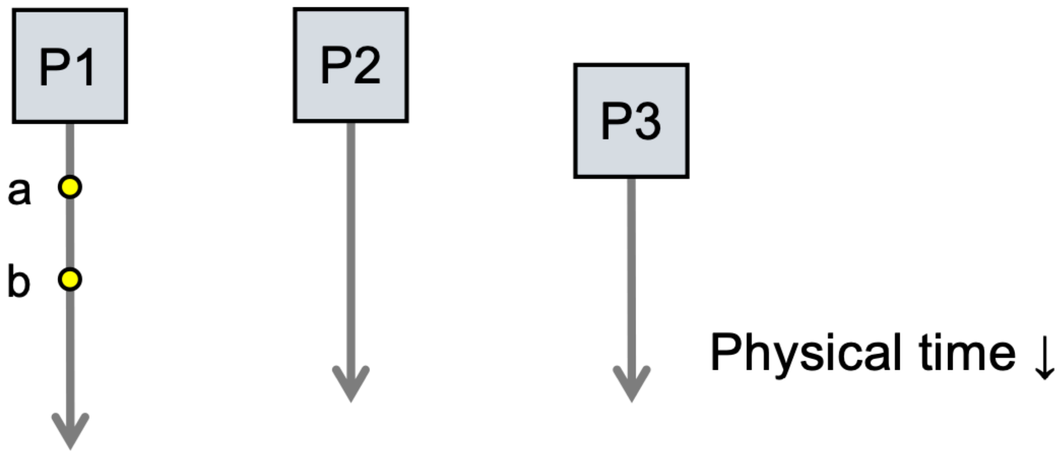
DEFINING “HAPPENS-BEFORE”

- Consider three processes: P1, P2, and P3
- Notation: Event a happens before event b (a->b)



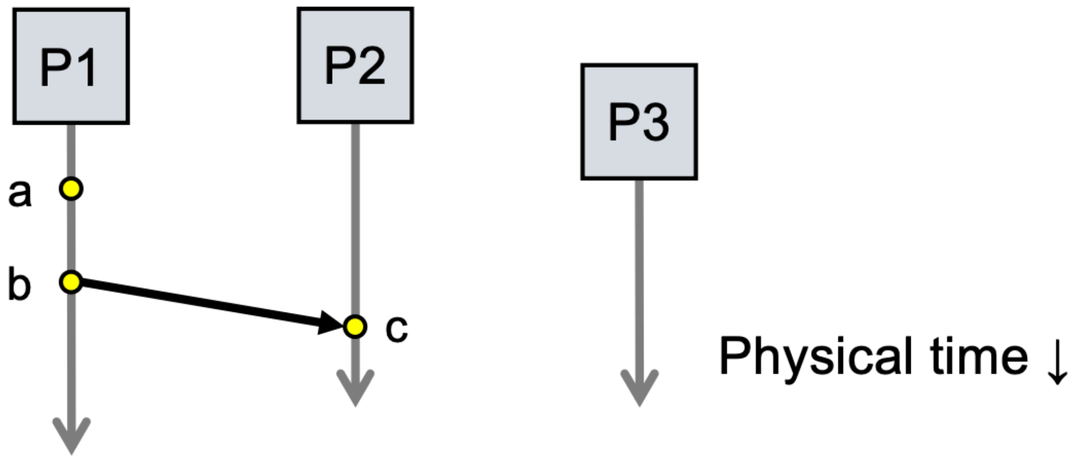
DEFINING “HAPPENS-BEFORE”

– 1. If same process and a occurs before b, then a -> b



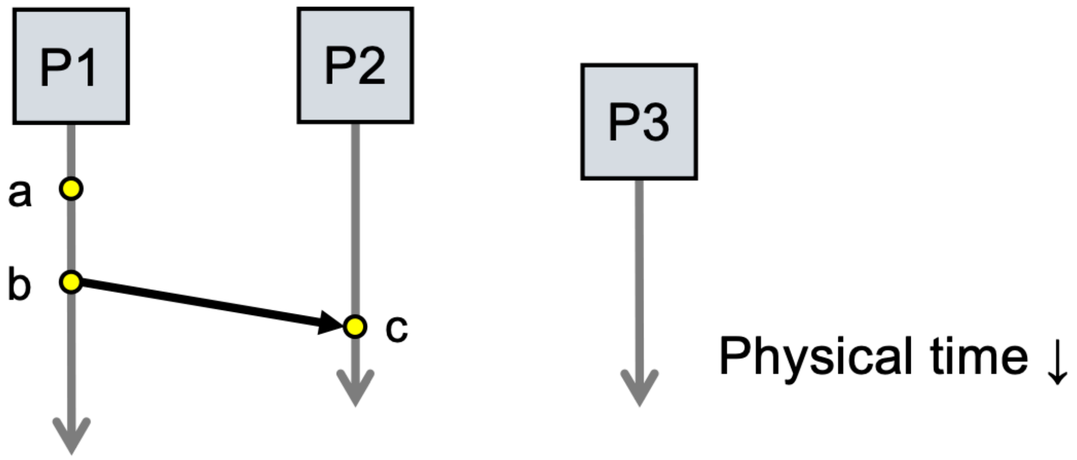
DEFINING “HAPPENS-BEFORE”

- 1. If same process and a occurs before b, then a -> b
- 2. If c is a message receipt of b, then b -> c



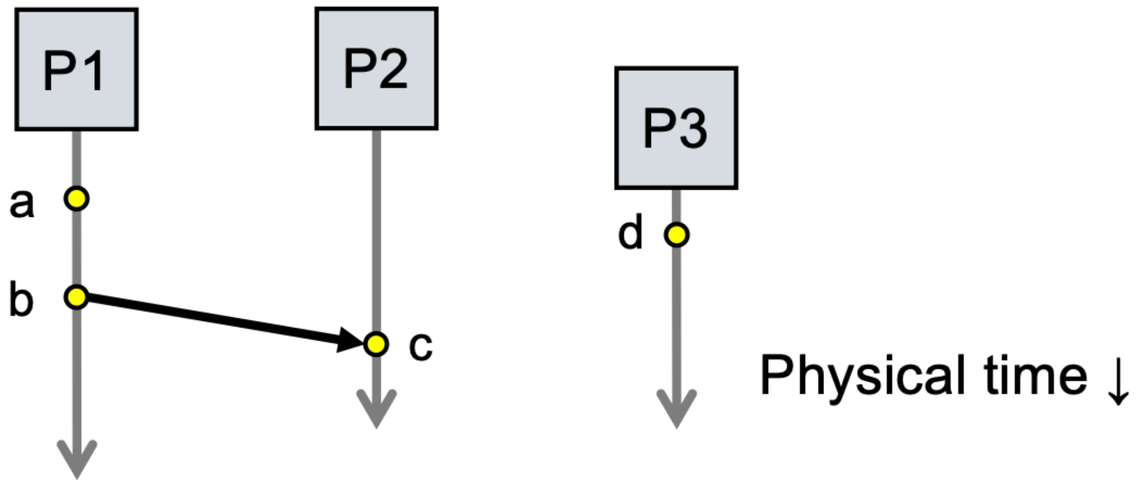
DEFINING “HAPPENS-BEFORE”

- 1. If same process and a occurs before b, then a -> b
- 2. If c is a message receipt of b, then b -> c
- 3. If a->b and b->c, then a->c



DEFINING “HAPPENS-BEFORE”

- Not all events are related by a
- a, d not related by a so concurrent, written as a ll d



LOGICAL CLOCK SYNCHRONIZATION PROTOCOL

- Lamport clock protocol [Lamport-1978].
- Setup:
  - Process = individual node in a distributed system
  - Processes communicate by messages (e.g., RPCs)
  - Events can be **messages** or system-specific **events** (e.g., write to file, read from file, whatever makes sense for the specific distributed system).
  - View each process in the distributed system as a **state machine**: has some initial state, events cause it to move from one state to another.

LAMPORT CLOCK PROTOCOL

- Each process Pi maintains a local counter, Ci
- Each process Pi increments Ci between any two successive events
- Each process piggybacks timestamp Tm on a message it sends out, where Tm is value of Ci at the time of sending m
- Upon receiving m at process Pj:
  - Pj sets its counter Cj to max(Cj, Tm+1)
  - The receipt of m is a separate event that then separately advances C (i.e., C ++)

Node Pi's state machine:
On local event:
- Ci++

On message send:
- Piggyback Ci to msg.
- Ci++

On message(Tm) receive:
- Ci = max(Ci, Tm+1)
- Ci++

GETTING A GLOBAL ORDER

- The preceding protocol gives a partial order of all causally dependent events.
- Often we need a global order on which all processes agree.
- To obtain that, use logical clock to set the order. Use process IDs as the tie breaker.
  - E.g.: use (Logical timestamp).(process ID) as your timestamp.

DISTRIBUTED DEBUGGING EXAMPLE

C_1	M1 (front end)
0	0.1 op11 rcv cli
1	1.1 op12 …
2	?? op13 snd M2
?	?? op14 …
?	?? op15 rcv M2
?	?? op16 …
?	?? op17 snd cli
?	…

C_2	M2 (app server)
0	?? op21 rcv M1
?	?? op22 …
?	?? op23 ...
?	?? op24 snd M3
?	?? op25 …
?	?? op26 …
?	?? op27 …
?	?? op28 …
?	?? op26 rcv M3
?	?? op27 …
?	?? op28 snd M1
?	…

C_3	M3 (DB)
0	0.3 op31 …
?	?? op32 …
?	?? op33 …
?	?? op34 rcv M2
?	?? op35 SQL
?	?? op36 …
?	?? op37 snd M2
?	…

Global Log

TODO: Timestamp the ops in each machine's log using logical clocks, then assemble the global log by merge-sorting them.

(assume Ci=0 initially)

Breakout Activity!

ACTIVITY (10 MINUTES)

- Assign logical timestamps to operations in each log, then sort the operations by timestamp in global log. A few entries have already been filled in as examples.
- Hint: As you go through the operations, keep track of the logical clock value at each machine, C1-3. Use the Lamport clock protocol to update the clocks (the algorithm is pasted on the right).
- Hint: It may be useful to first draw happens-before arrows between message sends and their receipts so you know when clock synchronization happens.
- Hint: Use a totally ordered clock: timestamp is Ci.i.

Node Pi's state machine:
On local event:
- Ci++

On message send:
- Piggyback Ci to msg.
- Ci++

On message(Tm) receive:
- Ci = max(Ci, Tm+1)
- Ci++

STUDENT WORKSHEET

C_1	M1 (front end)
0	0.1 op11 rcv cli
1	1.1 op12 …
2	?? op13 snd M2
?	?? op14 …
?	?? op15 rcv M2
?	?? op16 …
?	?? op17 snd cli
?	…

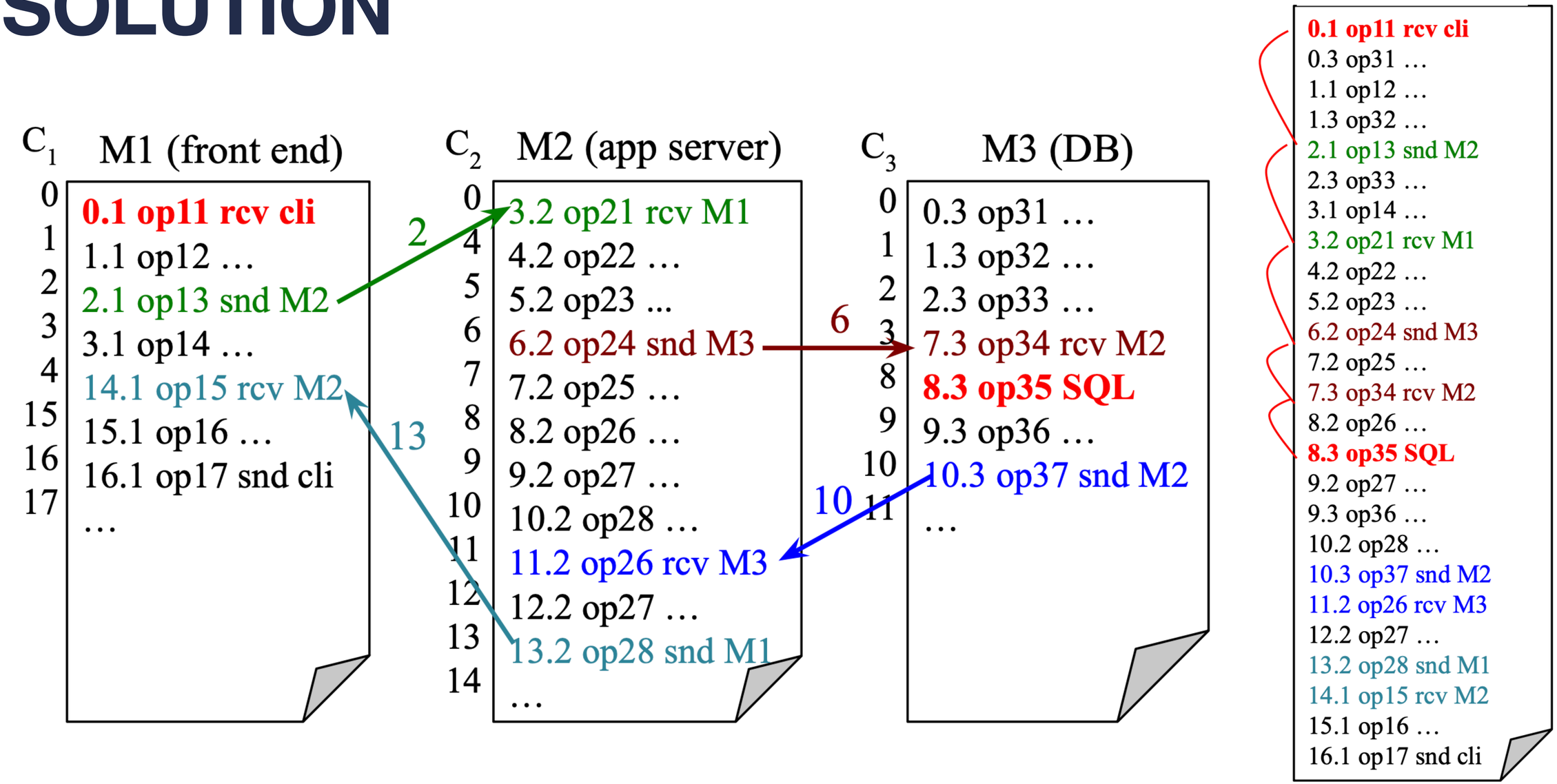
C_2	M2 (app server)
0	3.2 op21 rcv M1
?	?? op22 …
?	?? op23 ...
?	?? op24 snd M3
?	?? op25 …
?	?? op26 …
?	?? op27 …
?	?? op28 …
?	?? op26 rcv M3
?	?? op27 …
?	?? op28 snd M1
?	…

C_3	M3 (DB)
0	0.3 op31 …
?	?? op32 …
?	?? op33 …
?	?? op34 rcv M2
?	?? op35 SQL
?	?? op36 …
?	?? op37 snd M2
?	…

0.1 op11 rcv cli

… enter all events in order of their logical timestamp

SOLUTION


C1 M1 (front end)
0 0.1 op11 rcv cli
1 1.1 op12 …
2 2.1 op13 snd M2
3 3.1 op14 …
4 14.1 op15 rcv M2
15 15.1 op16 …
16 16.1 op17 snd cli
17 …
C2 M2 (app server)
0 3.2 op21 rcv M1
4 4.2 op22 …
5 5.2 op23 ...
6 6.2 op24 snd M3
7 7.2 op25 …
8 8.2 op26 …
9 9.2 op27 …
10 10.2 op28 …
11 11.2 op26 rcv M3
12 12.2 op27 …
13 13.2 op28 snd M1
14 …
C3 M3 (DB)
0 0.3 op31 …
1 1.3 op32 …
2 2.3 op33 …
3 7.3 op34 rcv M2
8 8.3 op35 SQL
9 9.3 op36 …
10 10.3 op37 snd M2
11 …
2
6
10
13
0.1 op11 rcv cli
0.3 op31 …
1.1 op12 …
1.3 op32 …
2.1 op13 snd M2
2.3 op33 …
3.1 op14 …
3.2 op21 rcv M1
4.2 op22 …
5.2 op23 …
6.2 op24 snd M3
7.2 op25 …
7.3 op34 rcv M2
8.2 op26 …
8.3 op35 SQL
9.2 op27 …
9.3 op36 …
10.2 op28 …
10.3 op37 snd M2
11.2 op26 rcv M3
12.2 op27 …
13.2 op28 snd M1
14.1 op15 rcv M2
15.1 op16 …
16.1 op17 snd cli

PLUSES AND MINUSES OF LAMPORT CLOCKS

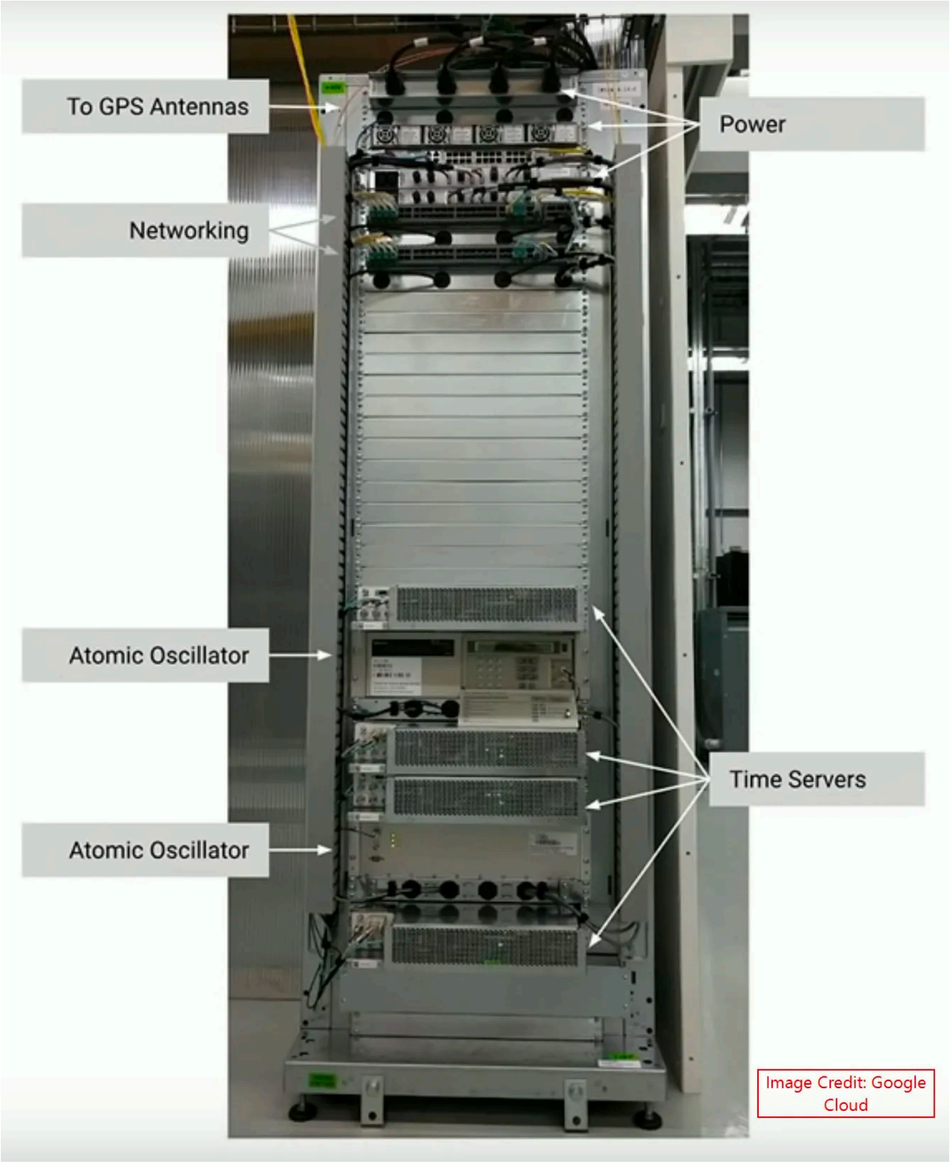
- Advantages
  - Respect causality, which can address many coordination problems in distributed systems.
- Disadvantages
  - Capturing causality is sometimes insufficient, as there can be events outside the system that have causal influence on the evolution of the system. The ordering doesn’t capture these relationships.
  - Lamport clock ordering doesn’t actually imply causality/influence, just potential influence. Hence, the order can be too much order, affecting performance/ scalability.

AGENDA

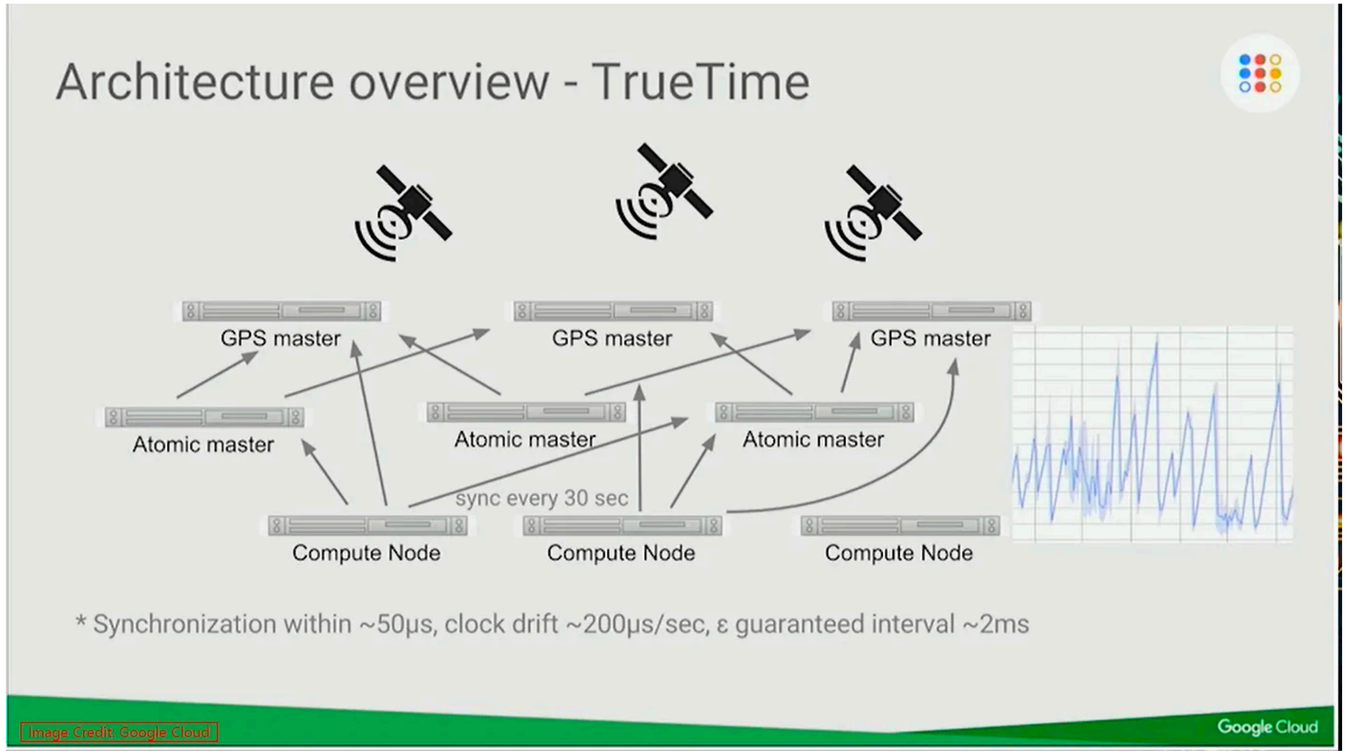
- Physical clocks
  - Synchronization challenges and protocols
- Logical clocks
  - Lamport clock protocol
- **(*) Google TrueTime**

GOOGLE TRUETIME

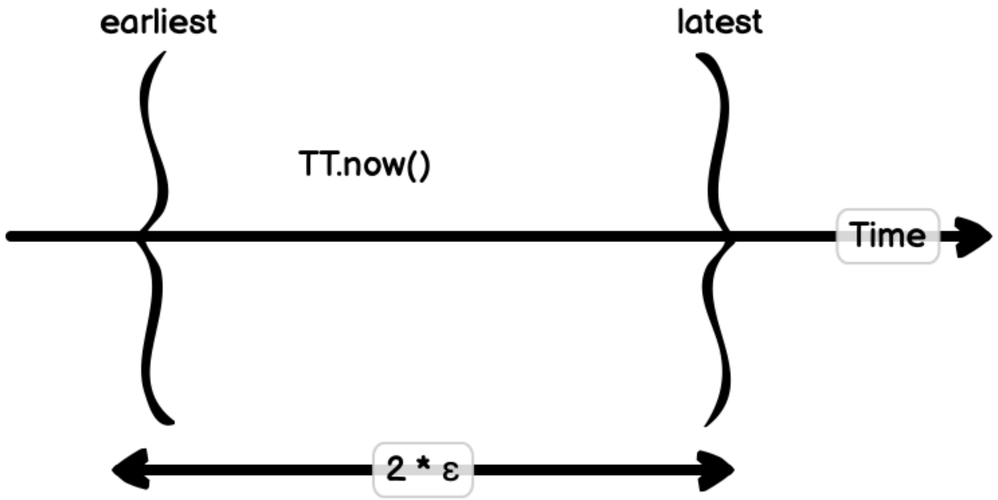
- A global synchronized clock with bounded non-zero error
  - if T2 starts to commit after T1 finishes committing, then the timestamp for T2 is greater than the timestamp for T1
- Underlying source of time: a combination of GPS receivers and atomic clocks
  - GPS Time Master: These nodes are equipped with GPS receivers which receive GPS signals include time information directly from satellites.
  - Armageddon Master: These nodes are equipped with local Atomic clocks. Atomic clocks are used as a supplement to GPS time masters in case satellite connections become unavailable.



GOOGLE TRUETIME API



Method	Returns
$TT.now()$	$TTinterval$: $[earliest, latest]$
$TT.after(t)$	true if t has definitely passed
$TT.before(t)$	true if t has definitely not arrived



TAKEAWAYS

- Time is crucial to distributed system coordination.
  - Disagreement between machines can result in undesirable behavior.
- Approaches:
  - Physical time: Often inadequate for distributed systems, need ns precision
  - Logical time: Lamport clocks, happens-before relation
- Next class: **Agreement**

ACKNOWLEDGEMENT

THIS COURSE IS DEVELOPED HEAVILY BASED ON COURSE MATERIALS SHARED BY PROF. INDRANIL GUPTA, PROF. ROBERT MORRIS, PROF. MICHAEL FREEDMAN, PROF. KYLE JAMIESON, PROF. WYATT LLOYD AND PROF. ROXANA GEAMBASU. MANY APPRECIATIONS FOR GENEROUSLY SHARING THEIR MATERIALS AND TEACHING INSIGHTS.