CPSC 416 Distributed Systems

Winter 2022 Term 2 (January 17, 2023)

Tony Mason (fsgeek@cs.ubc.ca), Lecturer

Recognize this image?

The persistence of memory

Salvador Dali, 1931

Museum of Modern Art (MoMA) New York, NY

I chose this because it captures two critical aspects of distributed systems:

- The ability to *remember*
- The importance of *time* in remembrance.

Today's lecture will focus on *time* **because** it is the basis of memory in distributed systems.

Logistics

Deadlines

Collaboration Quiz – Past due. Please turn in ASAP if you have not already done so.

Project 1 Deadline – January 23, 2023 (Extended). 75% cap for late submissions

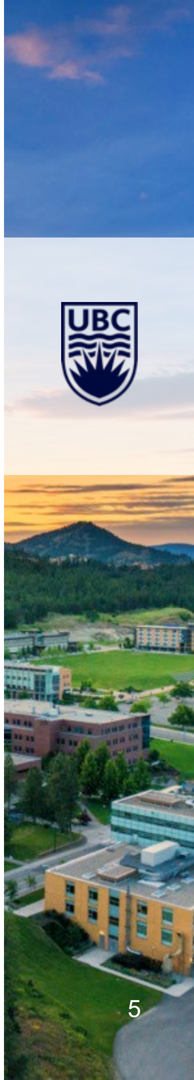
Project 2 Deadline – January 23, 2023 (Original). 75% cap for late submissions

Zero Penalty Drop Date – January 23, 2023. W standing after this date

TA Office Hours: TBA

Instructor:

- Zoom Office Hours (Tuesday) @ 13:00-14:00
- Discord (Casual) Office Hours (Thursday) @ 14:00-15:00



Recommended Readings

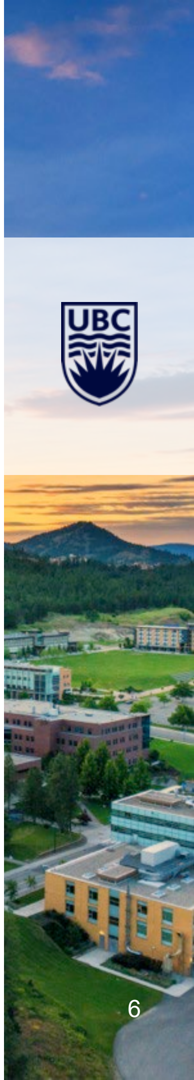
Spanner: TrueTime and external consistency

Lamport Clock

Network Time Protocol

Flash Boys (note, this is *not* a free resource; there is also a movie based on the book)

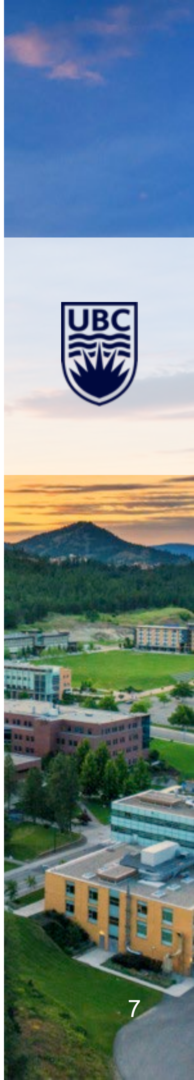
Flash Boys 2.0 (Academic Paper: Blockchain HFT tricks)



Alternative Path 1 (Challenge Mode)

Objectives:

- Identify a project of interested *related to distributed systems*
- Pick a team (3 total).
  - Shared responsibility:
    - Define project
    - Define requirements
    - Define interfaces
  - Solo Responsibility
    - One portion of the project
    - Conforms to project requirements and interfaces



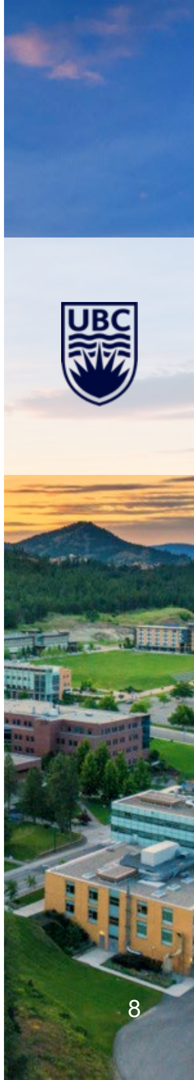
Alternative Path 1: Evaluation

Project

- Usefulness (as a distributed system)
- Project proposal quality
- Project report quality

Implementation

- Value to the project
  - Peer feedback
  - Instructional team review
- Value to other team members
  - Peer feedback
  - Instructional team review



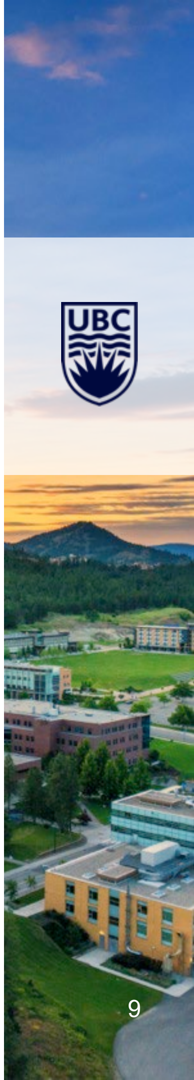
Alternative Path 1: Cost/Benefit Analysis

Benefits:

- Team-directed project
- *You* get to choose your team members
- Work on something useful
- FOSS encouraged – share it with others
- Can add up to 40% to your final grade

Costs:

- Teams can be difficult (“they did less work than I did.”)
- Requires *you* direct the project
- Open-ended



Example Project

DSLabs is *Java Based*

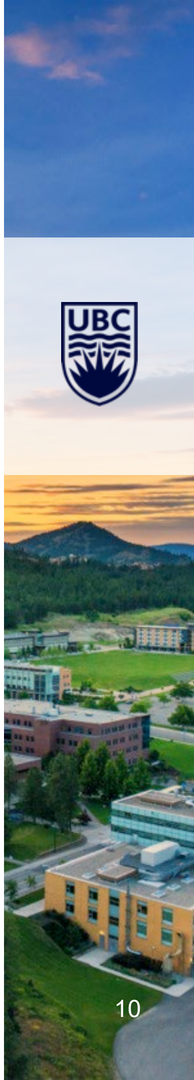
Java is the past. Rust is the future.

Project is to convert DSLabs from Java to Rust

Contributions:

- Build test project
- Build test framework
- Build visualizer

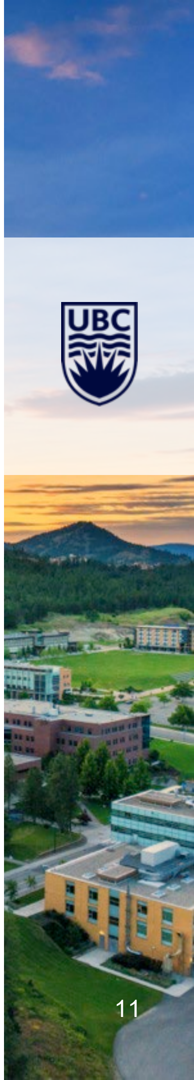
Note: this is *too big*. Do not propose this.



Networking Addendum

Remainder of Network Review is recorded and on-line.

Will *not* be covered in class.

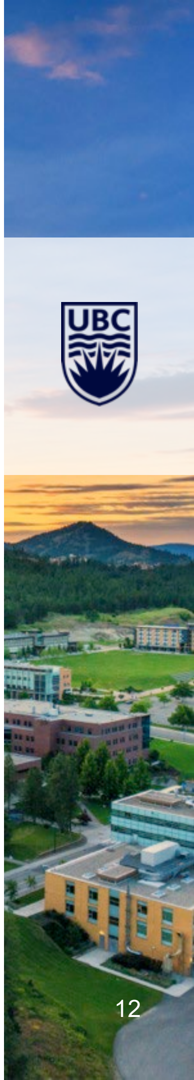


Questions?

Questions about the class?

Questions about the previous lecture?

Funny stories to share?

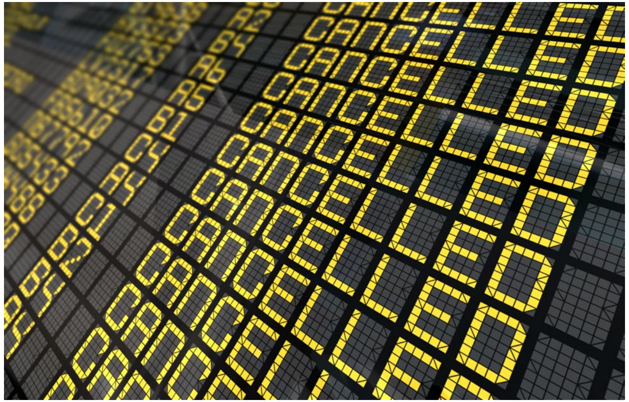


Today’s Failure

FAA Outage

Wednesday January 11, 2023

07:15 am EST



The FAA has ordered airlines to pause all domestic departures until 9 a.m. Eastern Time to allow the agency to validate the integrity of flight and safety information.

06:30 pm EST

Our preliminary work has traced the outage to a damaged database file. At this time, there is no evidence of a cyber attack. The FAA is working diligently to further pinpoint the causes of this issue and take all needed steps to prevent this kind of disruption from happening again.

Lesson Goals

Goals

Explore the *meaning* of "time"

- Challenges
- Reasoning

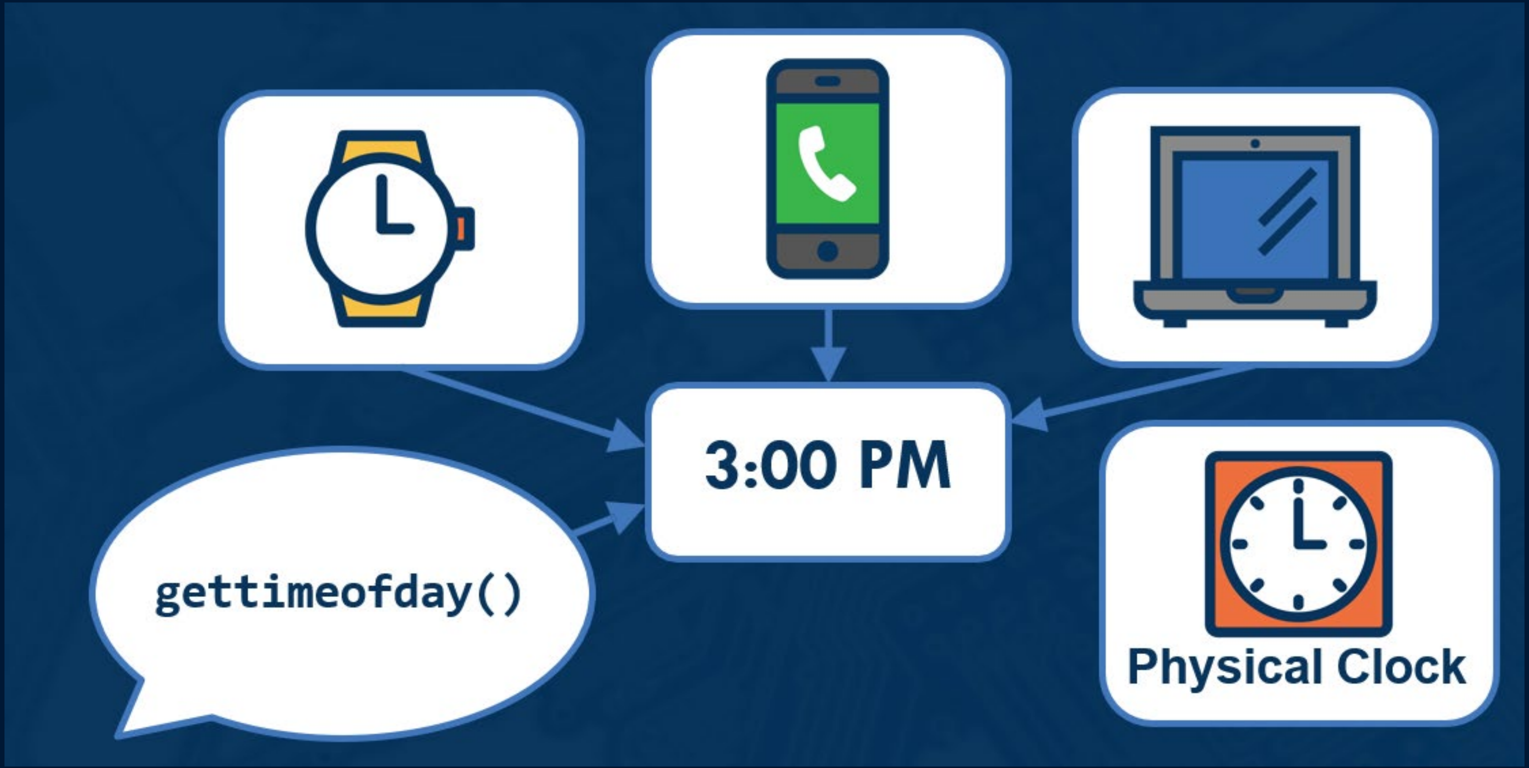
Discuss time

- Logical time
- Logical clock

Clock-based ordering models

- Scalar clocks
- Vector clocks
- Matrix clocks

Physical Time



Physical Clock Limitations

Computer clocks *drift*

- Oscillators are not perfect
- Deviates from *official* time

Time synchronization protocols (NTP)

- Fix computer clock
- May roll backwards

No universal time

- Different computers *typically* vary
- Global timestamp *could* solve this issue (Google Spanner)

Lamport clock

A *monotonic clock*

- Never rolls back
- Equal to or greater than the previous value

Any state-changing operation:

- Advances the clock

Defines an *order of events*

- Happens-Before relationship

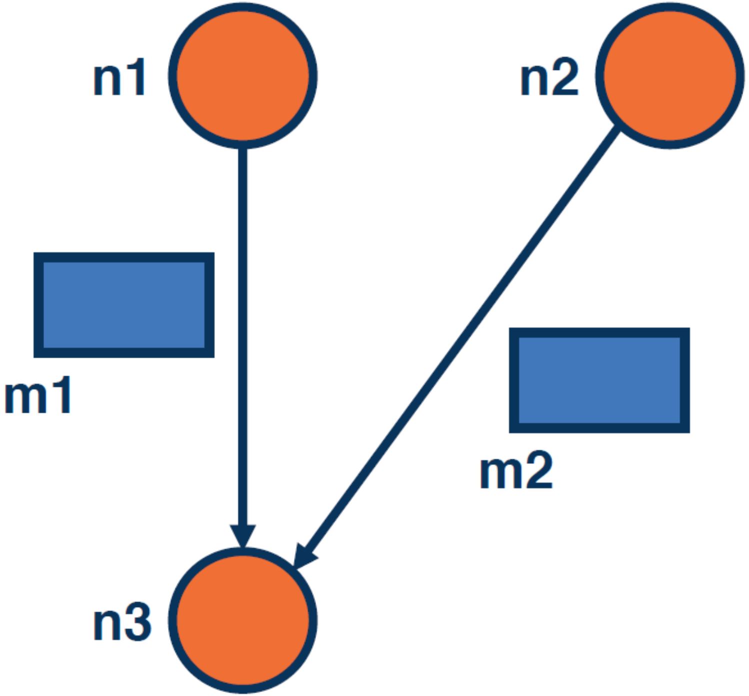
Why Time Matters

Challenges Measuring Time

Use receiver's clock

N_i sends message m_i

Order defined when <u>receiver</u> gets the message



Issues Using Receiver's Clock

Use receiver's clock

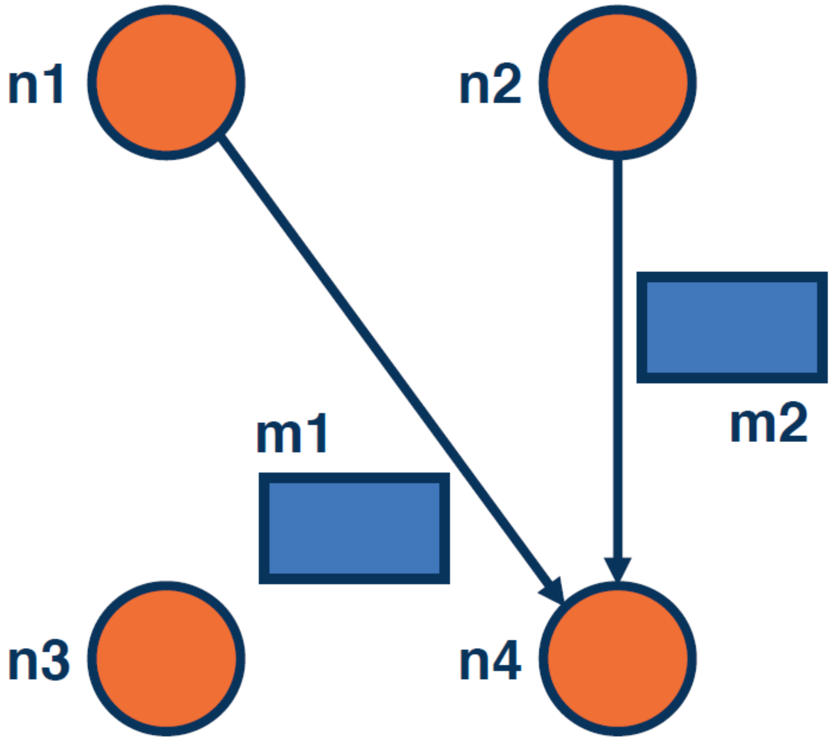
N_i sends message m_i

Order defined when <u>receiver</u> gets the message

What about:

- Network delays
- Lost messages

Not a viable solution

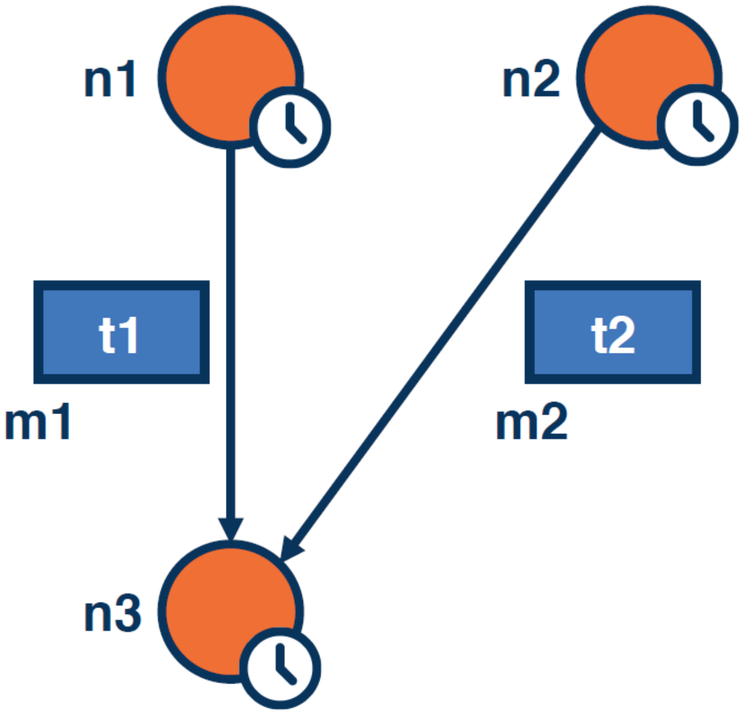


Use Sender's Clock

Each node uses its own clock

Message m_i uses time t_i (sender's time)

Receiver compares timestamps



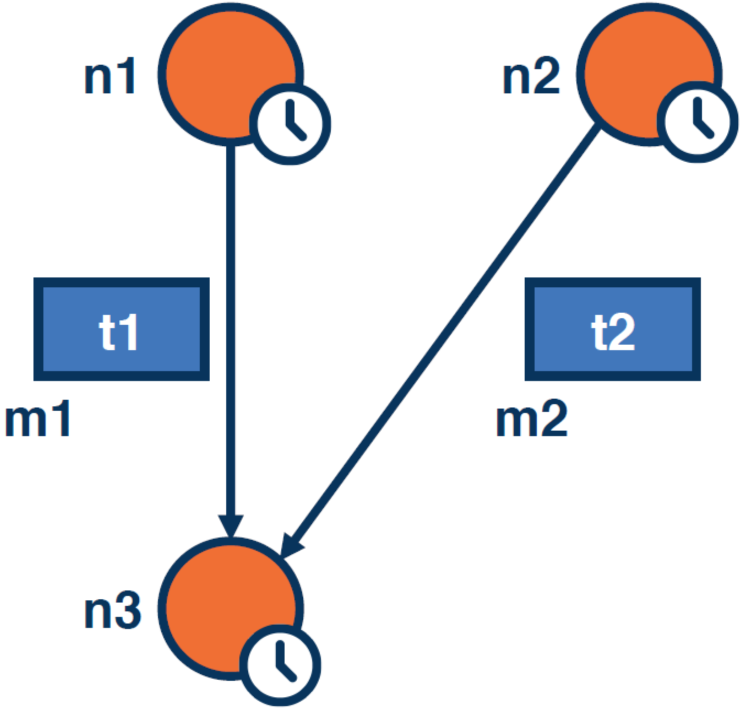
Senders' Clocks *Not* Synchronized

Each node uses its own clock

Message m_i uses time t_i (sender's time)

Receiver compares timestamps

Clocks are not *guaranteed to be synchronized.*



Questions

Do nodes **need to agree** on the global time?

Is the delivery time fixed?

- For messages
- For connections

Are network delays constant?

Can the network experience failures?

Could there be malicious nodes?

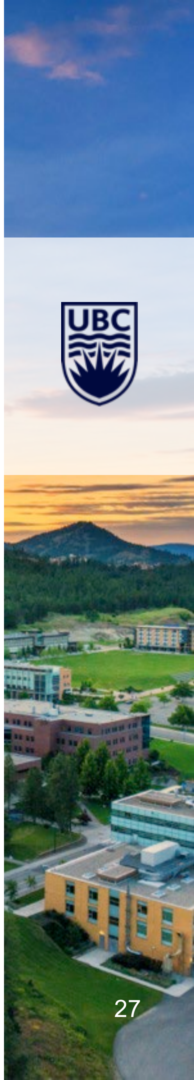
Logical Time

Alternative to Physical Time

Real time doesn’t work

Logical time

- Does not require global clock
- Strictly increase
- Useful for event ordering



Models of Logical Time

Logical Time: A Way to Capture Causality in Distributed Systems (1996)

Scalar (Lamport) clocks

Vector clocks

Matrix clocks

Notations (and Concepts)

p_i generates events $e_i^0, e_i^1, e_i^2, \dots, e_i^k, e_i^{\{k+1\}}, \dots, e_i^n$

Each $e_i^k \rightarrow e_i^{\{k+1\}}$:

- $\rightarrow$: “happens before” relationship
- $e_i^k \rightarrow e_i^{\{k+j\}}, j \geq 1$

A history H_i is an ordered sequence of events in p_i

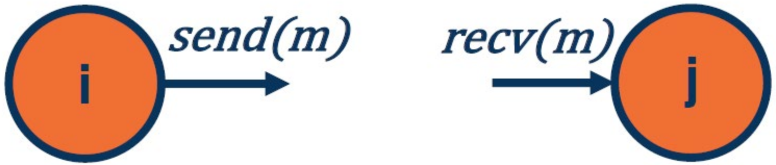
Notations (and Concepts)

Events:

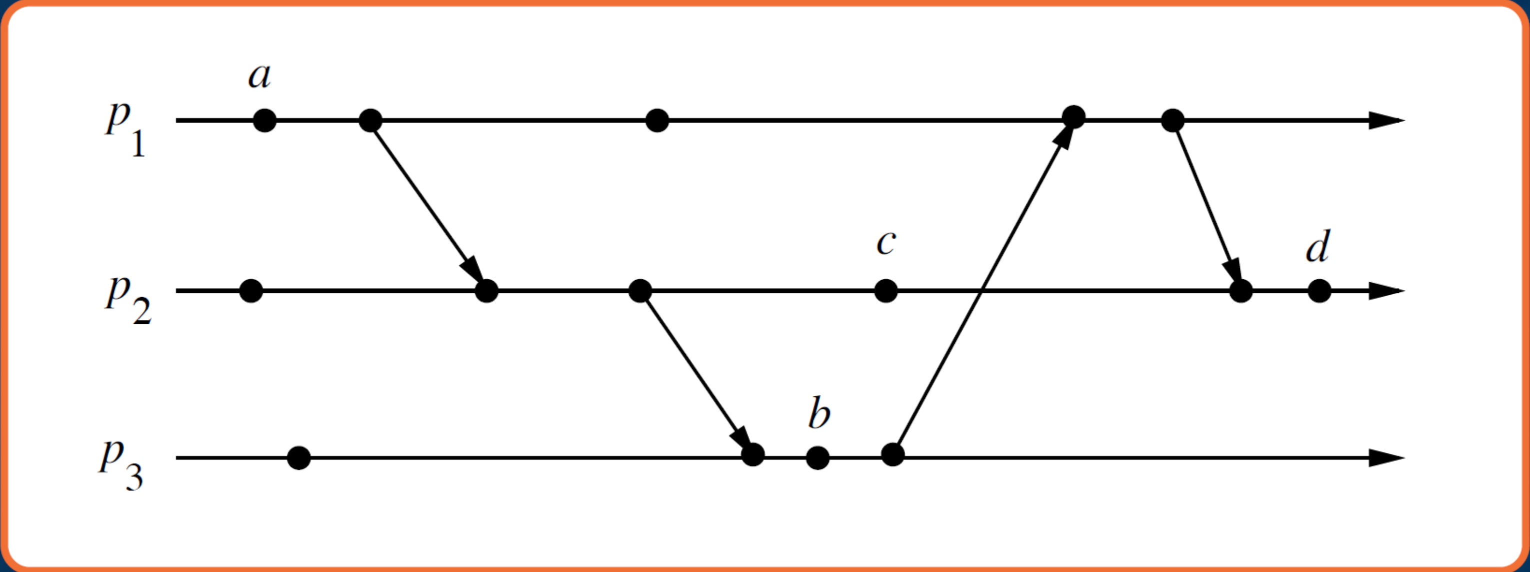
- $send(m), recv(m)$ *have visible output*

- *Node:* $recv_i(m), send_i(m+1)$

- *Nodes* i, j*:* $send_i(m), recv_j(m)$



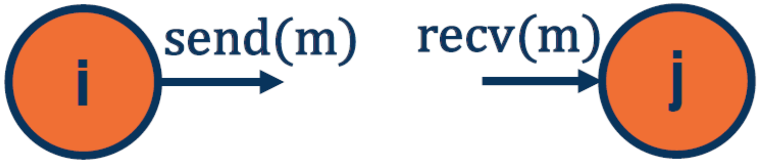
Notations (and Concepts)



Time Diagram of a Distributed Execution

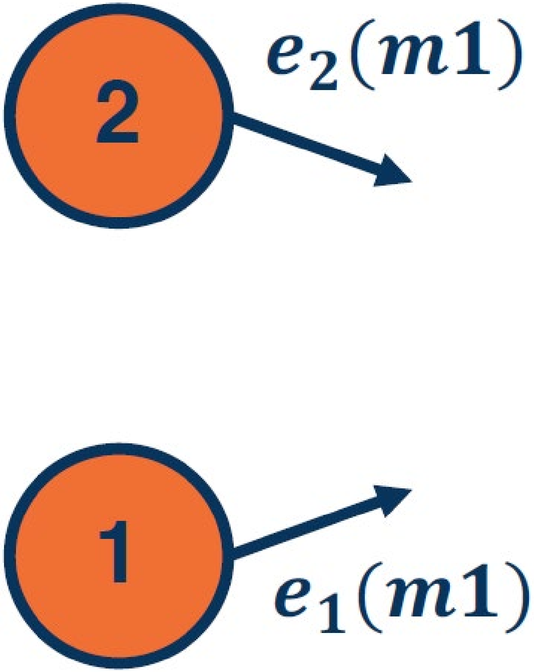
Ordering “concurrent events”?

$e_1 \rightarrow e_2 \Longrightarrow e_1$ happened before e_2



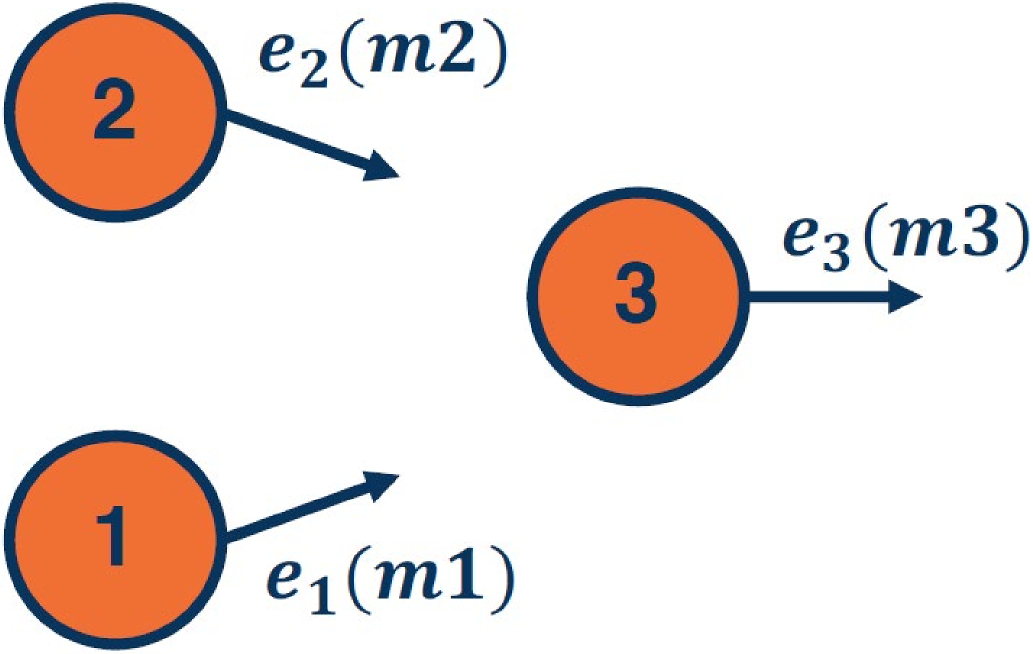
Concurrent Event

$e_1 \nrightarrow e_2$ and $e_2 \nrightarrow e_1 \Rightarrow e_1 || e_2$



Concurrent Event

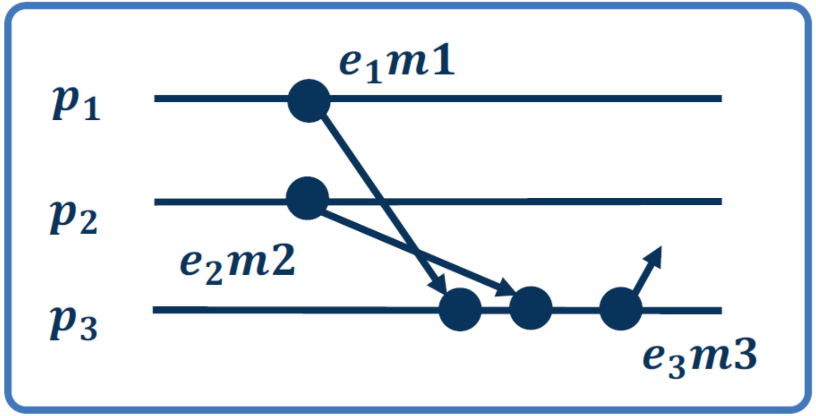
$e_1 \nrightarrow e_2$ and $e_2 \nrightarrow e$

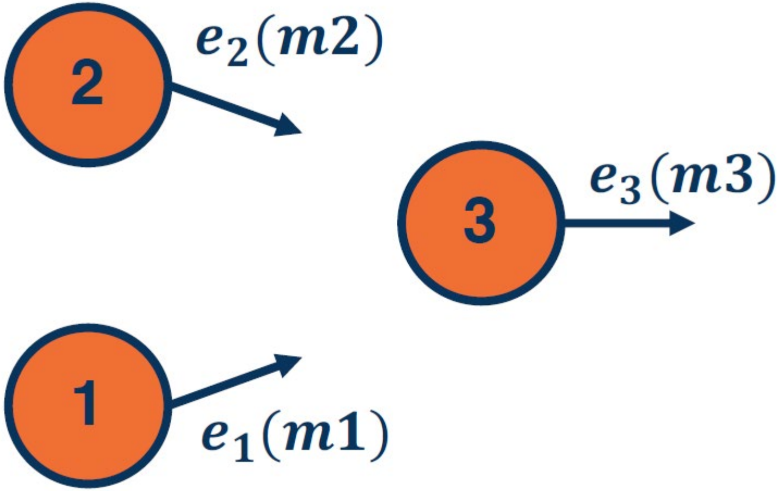


Concurrent Event

$e_1 \nrightarrow e_2$ and $e_2 \nrightarrow e_1 \Rightarrow$

$e_1 || e_2$





Logical Clock

$\forall\, e_i \in \{e_0, e_1, \ldots, e_n\}$ a *Logical Clock* C produces timestamps $C(e_i)$

Clock consistency condition:

- If $e_i \rightarrow e_j \Rightarrow C(e_i) < C(e_j)$
- Monotonic

$e_i \,||\, e_j \Rightarrow C(e_i) \,||\, C(e_j)$

Strong clock consistency:

$e_i \rightarrow e_j \Leftrightarrow C(e_i) < C(e_j)$

Summary: Logical Clock

For any event in a distributed system, a logical clock

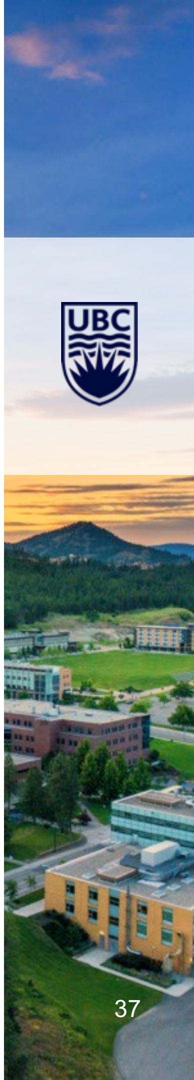
- T is the universe of timestamps
- C is a timestamp
- $C(e) \in T$

The *event history* is a set of partially ordered events in T.

- *Partially* because events may be concurrent

The function C

- Tells how timestamps are chosen (“incrementing the clock”)



Lamport Clock

Time, Clocks, and the Ordering of Events in a Distributed System

Rule 1: Before executing a state change ("event"), a process p_i executes:

$$C_i := C_i + d \ (d > 0)$$

Note: C_i represents the current clock value of process p_i.

Rule 2: Each message uses the clock value of that message's sender at sending time.

Thus, when a process p_i receives a message with timestamp C_t, it does the following:

- $C_i := \max(C_i, C_t)$
- Execute rule 1
- Deliver the message

Lamport Clock

Requirement (same process): $a \rightarrow b \Rightarrow C_i(a) < C_i(b)$

- Rule 1: p_i increases C_i after each event.
- Requirement is satisfied

Requirement (different processes): $a \rightarrow b \Rightarrow C_i(a) < C_j(b)$

- Rule 2: $T_m = C_i(a) = C_i$ (message timestamp)

  $C_j(b) = MAX(T_{\{m++\}}, C_j)$

Lamport Clock: Examples

$2@p_1 > 1@p_1$

$3@p_1 || 3@p_2$

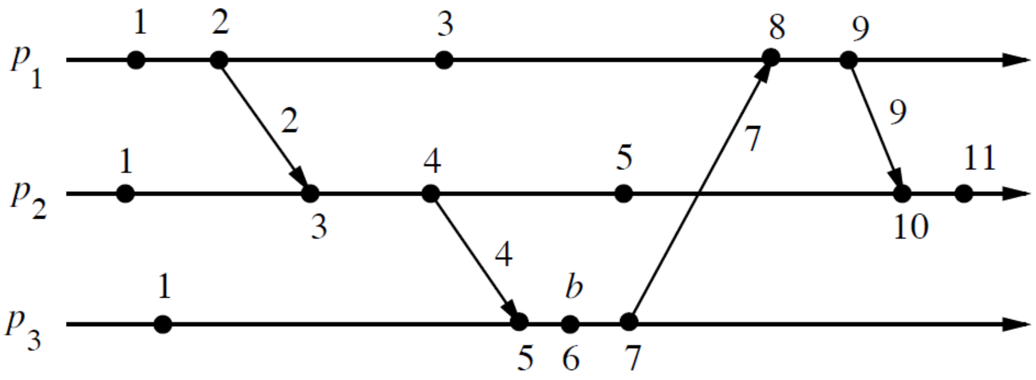
$4@p_1 ?? \ 3@p_2$

Tiebreaker: use process ID

- $3@p_1 < 3@p_2$

Consistent (only)

Partial ordering, not causality



Lamport Clock (Observations)

Clock provides estimate of events

Recall: $C_i \coloneqq C_i + d \ (d > 0)$

- If $d = 1$ then C_i is a lower bound on preceding events

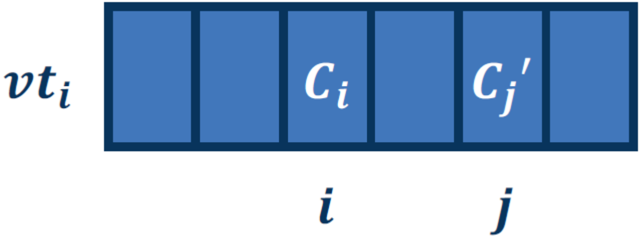
Consistent: $e_1 \rightarrow e_2 \Rightarrow C(e_1) < C(e_2)$

Not strongly consistent: $C(e_1) < C(e_2) \nRightarrow e_1 \rightarrow e_2$

Vector Clock

Construct a *vector* using linear (Lamport) clock values

Process p_i has clock vt_i



$vt_i[i] \Rightarrow p_i$'s Lamport Clock C_i

$vt_i[j] \Rightarrow p_i$'s last observed value of p_j's Lamport Clock C_j'

Each process tracks its view of time at other nodes

Vector Clock Rules

Rule 1: Before executing a state change ("event"):

$$vt_i[i] \coloneqq vt_i[i] + d \ (d > 0)$$

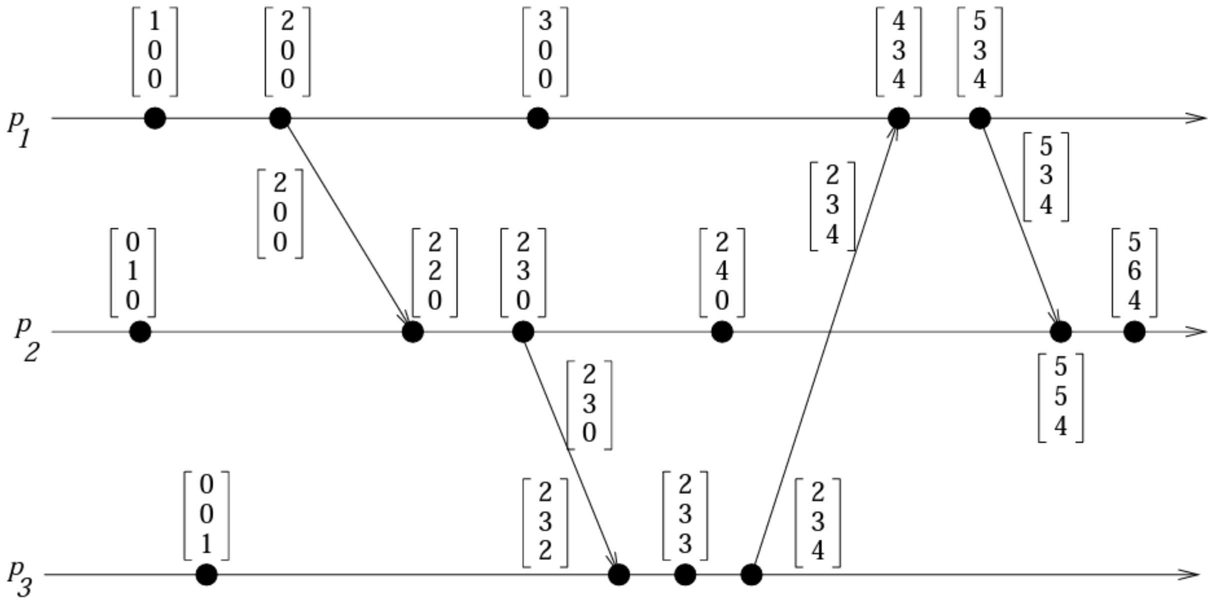
Rule 2:

- Current vector clock of p_i sent with each message (vt_i)
- Receiver:
  - Updates its vector clock

    $$1 \leq k \leq n : vt_i[k] \coloneqq \max(vt_i, vt[k])$$

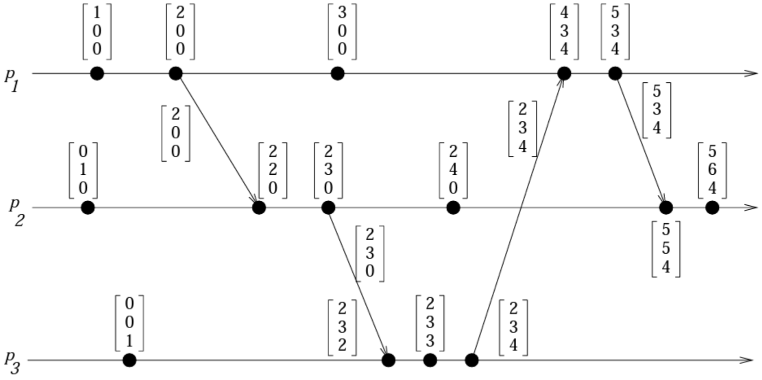
  - Executes Rules 1
  - Delivers the message

Vector Clock



Evolution of Vector Time

Vector Clock: Comparisons



$$v_h = v_k \Leftrightarrow \forall x : v_h[x] = v_k[x]$$

$$v_h \leq v_k \Leftrightarrow \forall x : v_h[x] \leq v_{k[x]}$$

$$v_h < v_k \Leftrightarrow (v_h \leq v_k) \land (\exists x : v_h[x] < v_k[x])$$

$$v_h || v_k \Leftrightarrow \neg(v_h < v_k) \land \neg(v_k < v_h)$$

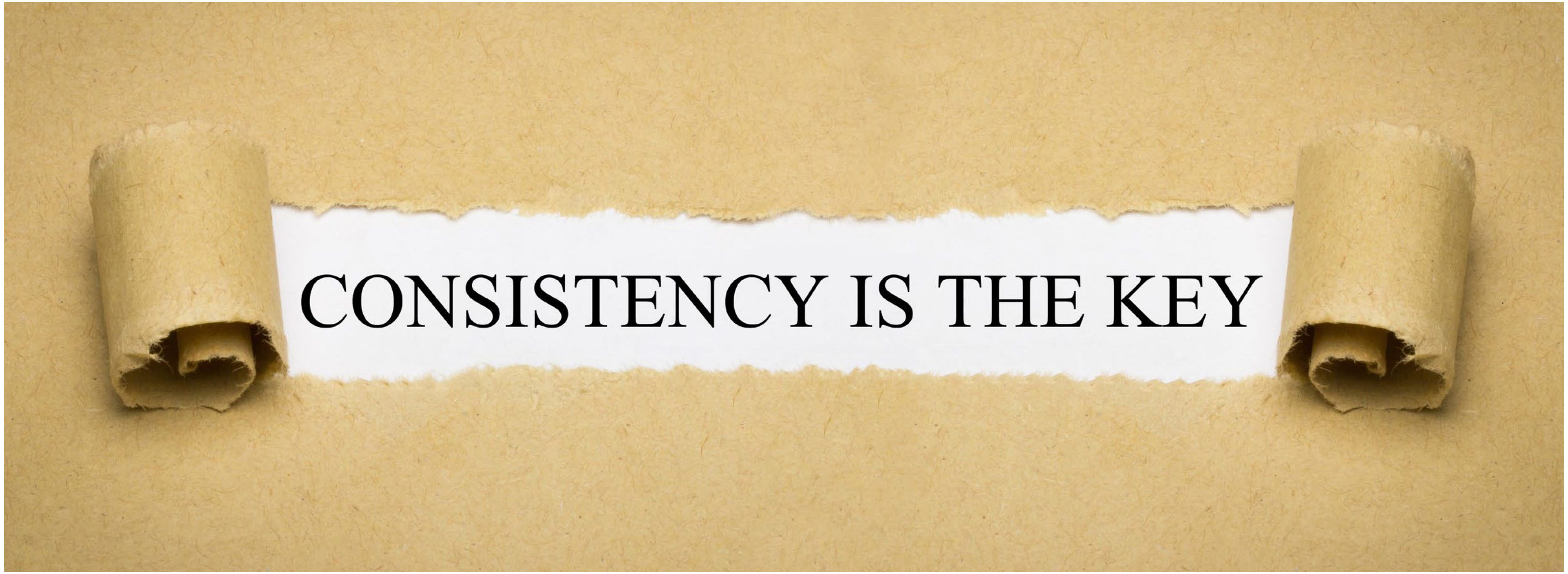
Vector Clock: Properties

If event x has timestamp v_h and event y has timestamp v_k then:

$$x \rightarrow y \Leftrightarrow v_h < v_k$$

$$x \,||\, y \Leftrightarrow v_h || \, v_k$$

Implication: there is an isomorphism between the vector timestamps and the set of partially ordered events.


CONSISTENCY IS THE KEY

Vector Clocks: Strong Consistency

Cost? Size grows linearly for the participants in the distributed operation.

Recall: $vt_i[i] \coloneqq vt_i[i] + d \; (d > 0)$

- If $d = 1$ then $vt_i[i]$ is the number of events that have occurred at p_i.
- If an event e has timestamp v_h, $v_h[j]$ is the number of events executed by process p_j that causally precede event e. Thus, $\sum v_h[j] - 1$ is the total number of events that causally precede e in the distributed computation.

Cost can be optimized by using a *delta* approach (Singhal-Kshemkalyani, Raynal-Singhal)

Matrix Clocks

Timestamp is a *matrix*. Process p_i has logical clock $mt_i[i, i]$, which tracks its scalar clock.

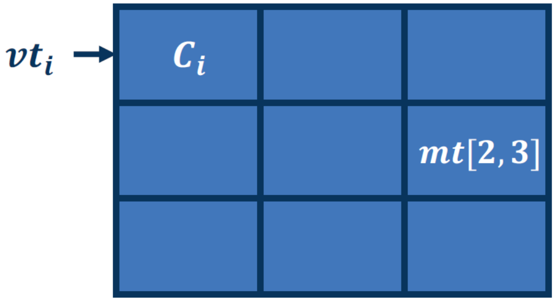
mt_i represents p_i's view of (logical) global time (e.g., its vector clock.)

$mt_i[k, l]$ represents p_i's view of what p_k knows about p_l's state.

Thus, each process has its own (possibly unique) view of the global state.

Why?

- Garbage collection!
- If $\min(mt_i[k, j]) > t$ then
  all processes know what happened prior to time t.



Lesson Summary

Summary

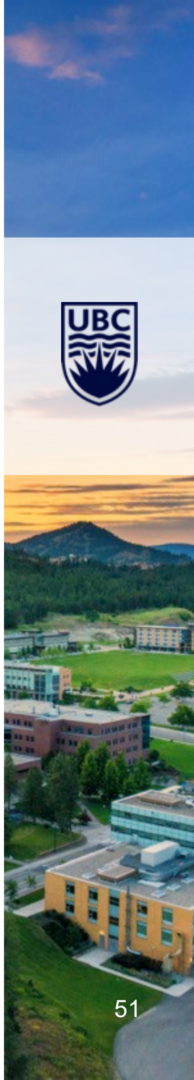
Explained the **challenges** associated with **reasoning about time** in distributed systems

Introduced the **notion of logical time** and **logical clock**

Described some models of **logical clocks**:

- Scalar (Lamport)
- Vector
- Matrix

Discussed partial ordering, strengths, and limitations of clock types.



Questions?


UBC
THE UNIVERSITY OF BRITISH COLUMBIA
THE UNIVERSITY OF BRITISH