CS 5412: LECTURE 6 TIMESTAMPED DATA

Ken Birman
Spring, 2019

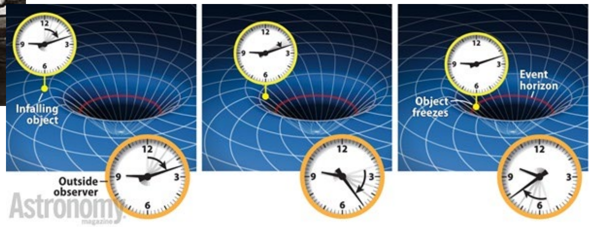
TODAY: DRILL DOWN ON TIME

Last time we discussed time more as an active aspect of a coordinated system (one of a few dimensions in which an IoT system might be active).

But once a sensor reading is captured and stored, there is also a temporal aspect to data analysis.

What can we say about time for data and events "inside" a data store?

TIME IN THE REAL WORLD



Einstein was first to really look closely at this topic.

It led to his theories of relativity and his Nobel Prize.

But Einstein was thinking about particles moving at near the speed of light, or near black holes. Do those ideas apply in other settings?

TIME IN COMPUTER SYSTEMS

Often, we put “timestamps” on IoT sensor records

In IoT, time is tricky to work with for many reasons:

- Even with GPS recievers, it can be hard to get a good fix, so time can drift
- IoT sensors often lack GPS and their clocks need to be reset via an event, but then might drift by seconds per day
- Sensors can also fail, and this includes their clocks.

Thus a timestamped event may have inaccurate time!

IN WHAT WAYS CAN WE TALK ABOUT TIME?

First, whenever we use time in an IoT setting, it is important to track the time source and the associated skew:

- Without GPS time, sensor time will drift by seconds/day
- With GPS time, clocks can be accurate to within about 1ms
- With special purpose hardware for synchronization, the machines in a cloud would be able to share a clock and be accurate to a few *us*.
- … but today's cloud computers don't have that form of shared clocks, and if virtualized, clocks can be quite inaccurate! A total mess!

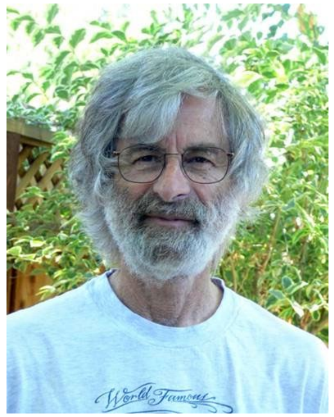
VENDORS PREFER LIMITED ACCURACY!

Several recent security problems have involved an attacker who places a monitoring program on the same machine that some security code is on. The attacker is assumed to have the source code for the application it is attacking.

The monitoring program measures timing properties of the memory and caching hardware at very high accuracy and is able to deduce contents of the memory state of the attacked program.

It seems doubtful that this would work, but several exploits show that it really *does* work! Even so, cloud vendors make it hard to measure time.

LAMPORT'S CAUSAL ORDER



Leslie Lamport is a famous distributed computing researcher

- Started out as a physicist and was inspired by Einstein, but went on to formalize distributed protocols, and won the Turing Award
- Primarily a theoretician, but he also was the author of Latex
- Especially good at elegant ways of posing problems and solving them

He suggested that an important aspect of consistency should involve "consistency with respect to past events". He calls this "causal" consistency

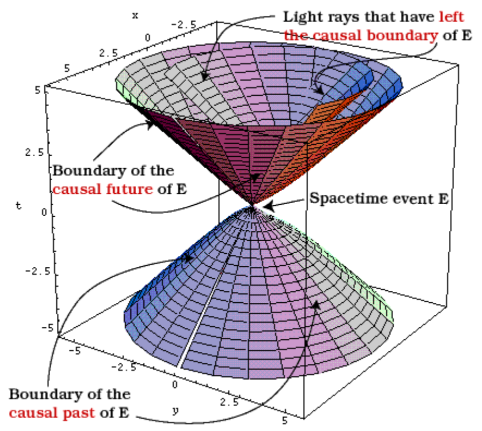
HOW DOES HE DEFINE CAUSALITY?

Suppose that event A occurs in a data center, and then later event B.

Did A “cause” B to happen?

- What if A was at 10am, and B at 11:30pm. Does knowing time help?
- What if A was a command to register a new student, and B was an internal action that creates her “meal card” account?
- What if A was an email from the department asking me about my teaching preferences, and B was my reply?

HOW DOES HE DEFINE CAUSALITY


Light rays that have left the causal boundary of E
Boundary of the causal future of E
Spacetime event E
Boundary of the causal past of E
x
t
y

For Leslie, event A causes event B if there was a computation that somehow was triggered by A, and B was part of it.   Inspired by physics!

But this is hard to discover automatically.

Instead, Leslie focused on *potential* causality: A "might" have caused B.

Under what conditions is this possible?

- Somehow, *information must flow* from A to B.

NOTATION FOR REPRESENTING CAUSALITY

Leslie proposes that we write A $\rightarrow$ B if A potentially caused B.

He suggests that we use the words “happened before” for $\rightarrow$

Now the question arises: is $\rightarrow$ just a mathematical concept, or can we build a practical tool for tracking causality in real systems?

WHY WOULD WE WANT TO TRACK A $\rightarrow$ B?

Consider the Securities and Exchange Commission.

For them, A might be “information about stock X” and B “a trade of X”.

An insider trade occurs if someone with non-public information takes advantage to trade a stock before that information comes out. So if “John learned that the IBM quantum computer showed promise”, then bought IBM stock, perhaps John violated the insider trading law.

LAMPORT'S POINT

Simply seeing data records in which John talks to his friend at IBM at 10:00am and then buys IBM stock at 10:01am might not be "proof" of criminality. These days the cloud might participate in all of these events.

If the records were timestamped by the identical clock, and the clock isn't faulty, this really would be proof.

But if the records came from different computers, clock imprecision could be creating an illusion. If we track actual $\rightarrow$, we would be confident.

TRACKING A $\rightarrow$ B

Leslie first considered normal clocks. But they don't track $\rightarrow$

- Here, he took his inspiration from Einstein
- "*Time is an illusion.*" Einstein went on to draw space-time diagrams.

So Leslie asked: "Can we use space-time diagrams as the basis of a new kind of "logical clock"?

- If A $\rightarrow$ B, then LogicalClock(A) < LogicalClock(B)
- If LogicalClock(A) < LogicalClock(B), then A $\rightarrow$ B

DEVELOPING A SOLUTION

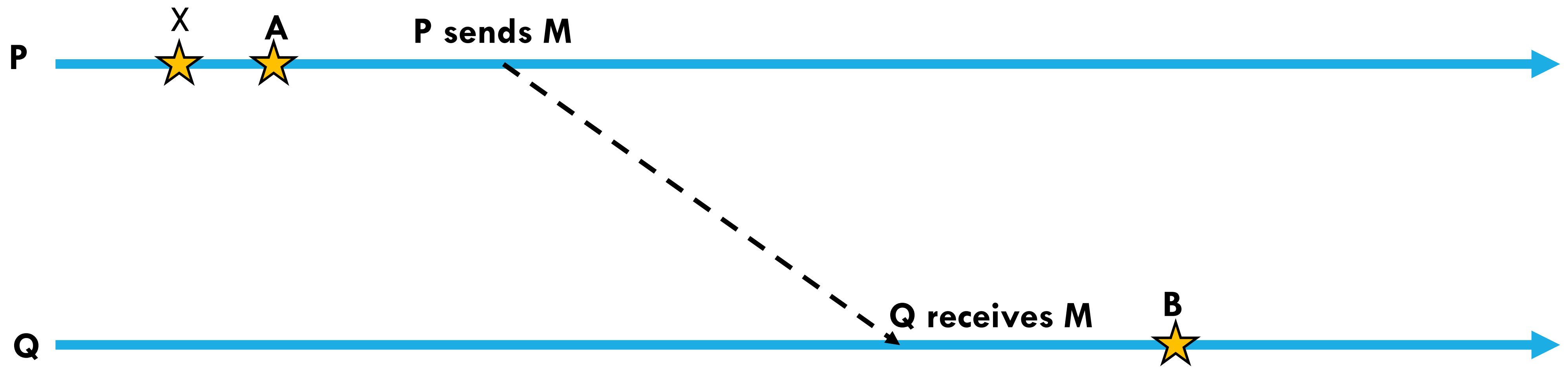
Suppose that every computer (P, Q, …) has a local, private integer

Call these LogicalClock_P and LogicalClock_Q etc.

Each time something happens, increment the clock.

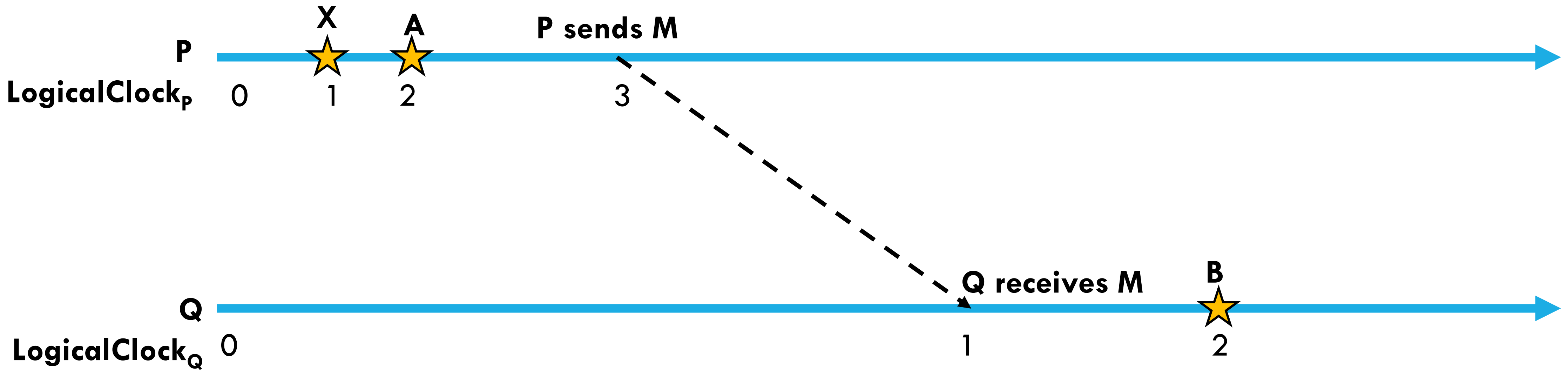
- Now, if A and B happen at P, the LogicalClock_P can tell us that $A \rightarrow B$.
- But what if A is on machine P, and B happens on Q?

A SPACE-TIME DIAGRAM FOR THIS CASE



A SPACE-TIME DIAGRAM FOR THIS CASE

Uncoordinated counters don’t solve our problem



Here, A and B end up with the identical Time, so we incorrectly conclude that A did not *happen before* B

AHA!

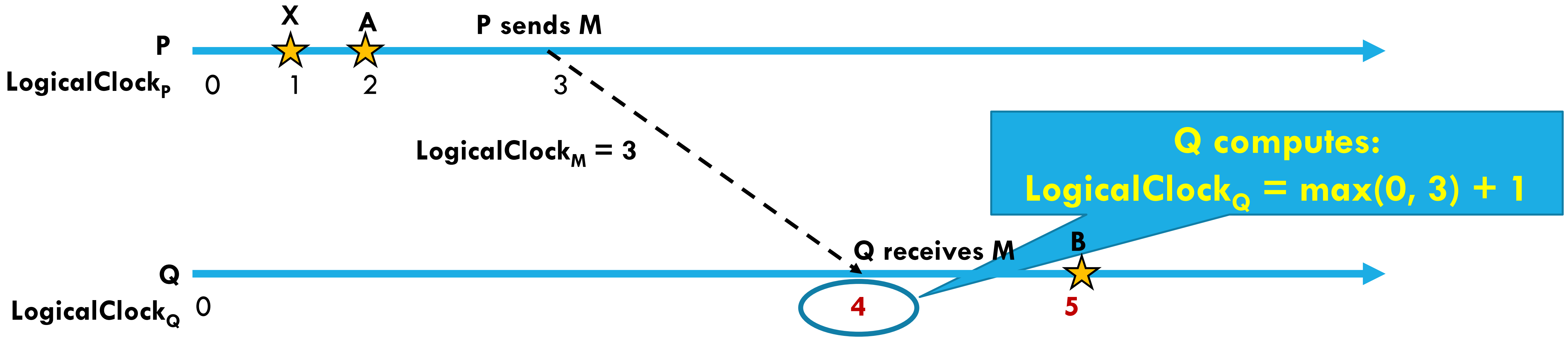
But notice that in the diagram, the “receive” occurs when $\text{LogicalClock}_B = 1$.

Yet the “send” of M was at $\text{LogicalClock}_A = 3$.

So Lamport proposes this fix:

- Each time an interesting event occurs at P, increment LogicalClock_P
- If P sends M to Q, include LogicalClock_P in M. When Q receives M, $\text{LogicalClock}_Q = \text{Max}(\text{LogicalClock}_Q, \text{LogicalClock}_M) + 1$

A SPACE-TIME DIAGRAM FOR THIS CASE



WE NOW HAVE A CHEAP PARTIAL SOLUTION!

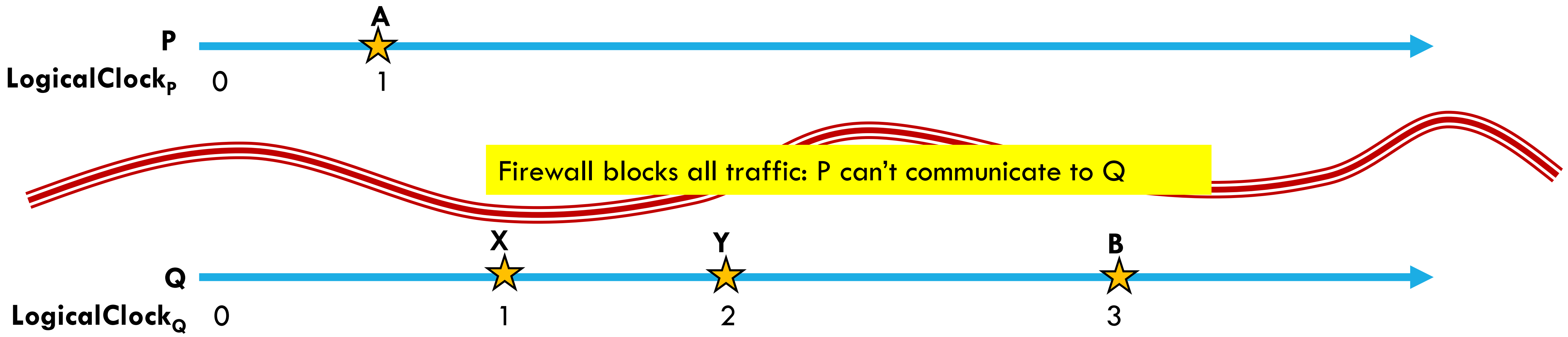
With Lamport’s logical clocks, we pay a small cost (one integer per machine, to keep the clock, and some space in the message)

Let’s use LogicalClock(X) to denote the relevant LogicalClock value for x. We can time-stamp events and messages.

- If A $\rightarrow$ B, then LogicalClock(A) < LogicalClock (B)
- But… if LogicalClock (A) < LogicalClock (B), *perhaps A didn’t happen before B!*

A SPACE-TIME DIAGRAM FOR THIS CASE

With logical clocks, even if P and Q never talk, we might have Time(A) < Time(B)



Here, if we claim that LogicalClock(A) < LogicalClock (B) $\Rightarrow$ A $\rightarrow$ B, this is nonsense! In fact $\neg$(A $\rightarrow$ B), $\neg$(B $\rightarrow$ A). (A and B are "concurrent")

LOGICAL CLOCKS ONLY WORK IN ONE DIRECTION.

They approximate the causal happens-before relationship, but only in an "if-then" sense, not "If and only if".

Lamport gives many examples where this is good enough.

We actually can do better, but at the "cost" of higher space overhead.

DETECTING INSIDER TRADING

The SEC* wants to detect that "John learned of the good news from Lilly, CEO of Zebra Corp. Then he purchased stock before the market heard."

If A was John learning, and B was the stock purchase, then the SEC wants to look at LogicalClock(A) < LogicalClock(B), and conclude "A → B".

But logical clocks don't let us conclude this. And John might insist that "I kept a log of call times, and I spoke to Lilly after the IBM market announcement. Perhaps some clock drifted and the SEC has its time sequence wrong."

* Securities Exchange Commission

INTUITION BEHIND VECTOR CLOCKS

Suppose that we had a fancier clock that could act like logical clocks do (with the "take the max, then add one" rule).

But instead of a single counter, what if it were to count "events in the causal past of this point in the execution", tracking events on a per-process basis?

For example, a VectorClock value for A = [5,7] might mean "event A happens after 5 events at P, and 7 events at Q".

VECTOR CLOCKS ARE EASY TO IMPLEMENT

A vector clock has one entry per machine. VT(A) = [3, 0, 7, 1]

- If an event occurs at P, P increments *its own entry* in the vector
- When Q receives M from P, Q computes an entry-by-entry max, then increments its own entry (because a "receive" is an event, too)

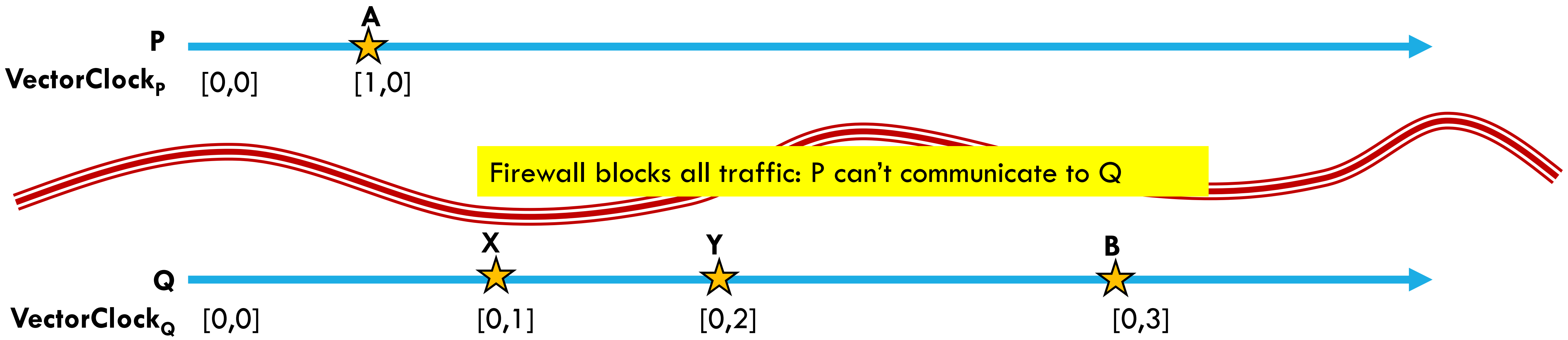
VectorClock comparison rule:

Define VT(A) < VT(B) if
VT(A) ≤ VT(B),
but VT(A) ≠ VT(B)

Now, VT(A) < VT(B) iff A → B

A SPACE-TIME DIAGRAM FOR THIS CASE

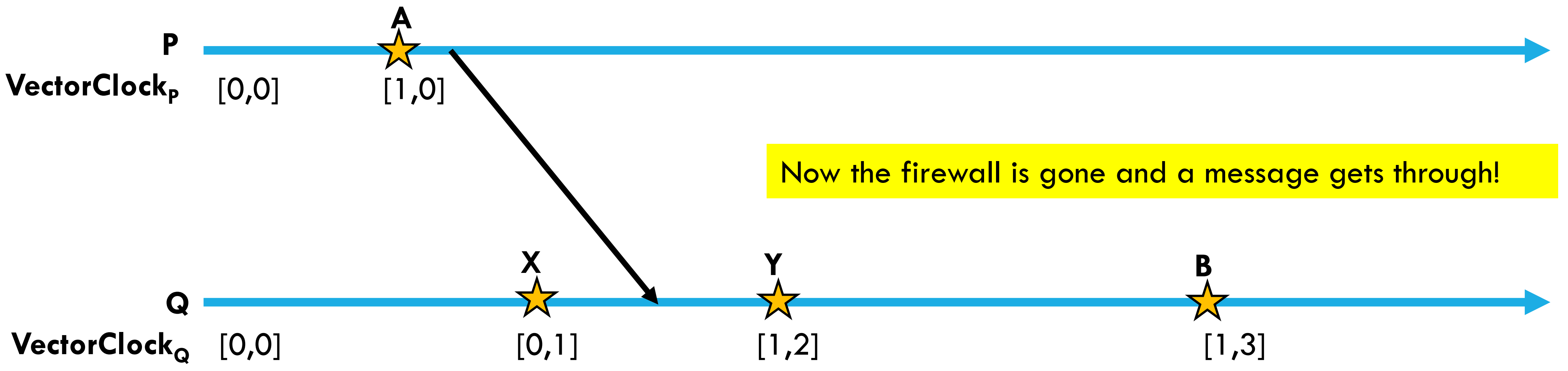
Case A: Suppose that P and Q never interact.



With vector clocks we can see that A is concurrent with X, Y and B. We can use the comparison rule to show this, for example that $\neg(A \rightarrow B)$ and $\neg(B \rightarrow A)$.

A SPACE-TIME DIAGRAM FOR THIS CASE

Case B: P sends a message to Q after A, and it is received before B at Q.



The vector timestamps show that A happens before B (and also, before Y).

VECTOR CLOCKS SOLVE THE SEC PROBLEM!

A: John spoke to his friend Lilly.

Then the message M was to tell his stock broker to “Buy IBM futures ASAP!” B was the purchase. Our goal: Deduce that $A \rightarrow B$ using just a database with information about A, and information about B, including timestamps.

We just saw that

$$\text{VectorClock}(A) < \text{VectorClock}(B) \quad \Rightarrow \quad A \rightarrow B!$$

SO WHY NOT <u>ALWAYS</u> USE VECTOR CLOCKS?

They represent happens-before with full accuracy, which is great.

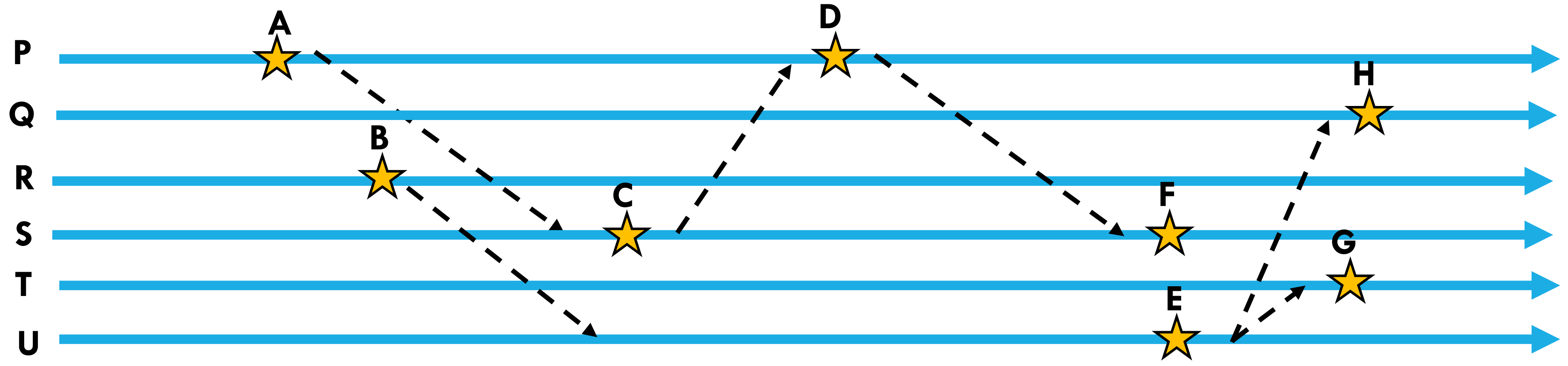
But you need one vector entry per process in your application. For a small μ-service this would be fine, but if the vector would become large, the overheads are an issue.

So, we try to use a LogicalClock before considering a VectorClock.

MORE FUN WITH CAUSALITY

Working with Mani Chandy (CalTech), Lamport also showed that you can use → to define "now" in a way that makes sense even for a fully distributed system

He draws a complex space-time picture, perhaps this one:



CONSISTENT CUTS AND SNAPSHOTS

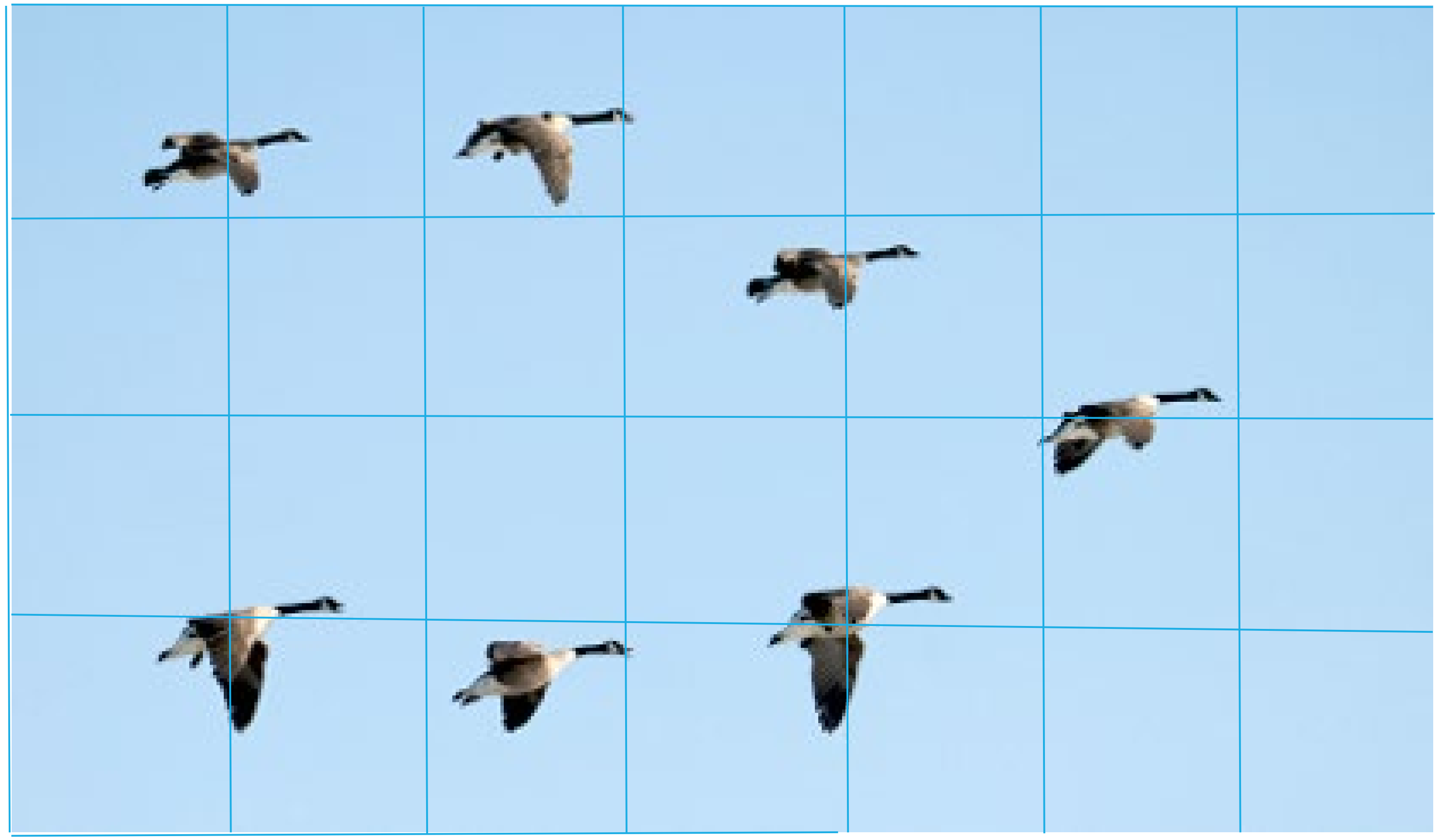
They asked: Suppose I visit each node, each at some point in time. Can we extend consistency to cover such a case ("consistent cut")

Or even fancier: what if each node makes a checkpoint for me when I visit it along a cut. Can we end up with a "consistent snapshot", like a photo?

CONSISTENT AND INCONSISTENT SNAPSHOT

Truth: 7 Geese in a V formation

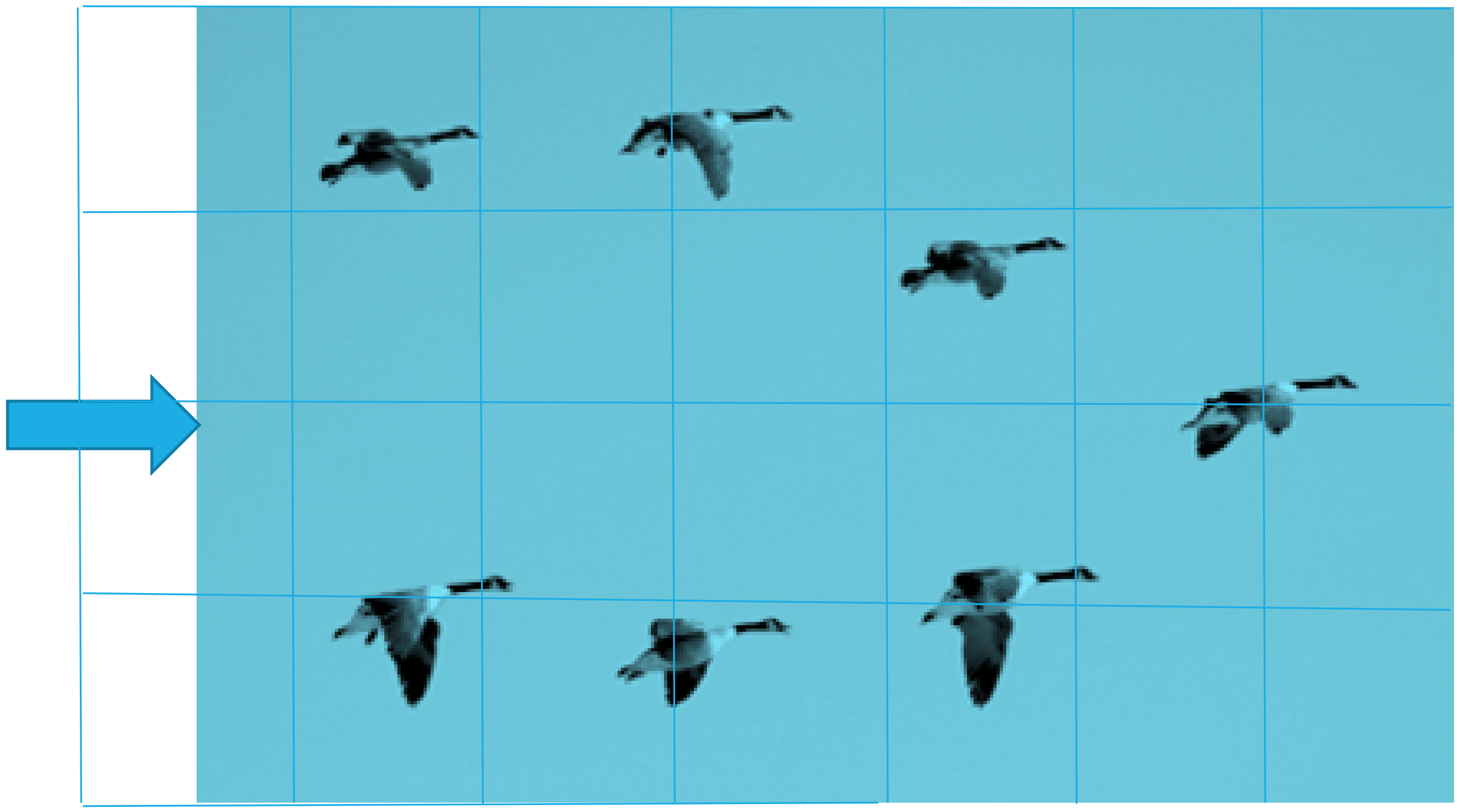
Imagine taking photos of our geese one by one and creating a tiled mashup



CONSISTENT AND INCONSISTENT SNAPSHOT

Truth: 7 Geese in a V formation

But suppose they were in motion while you did this



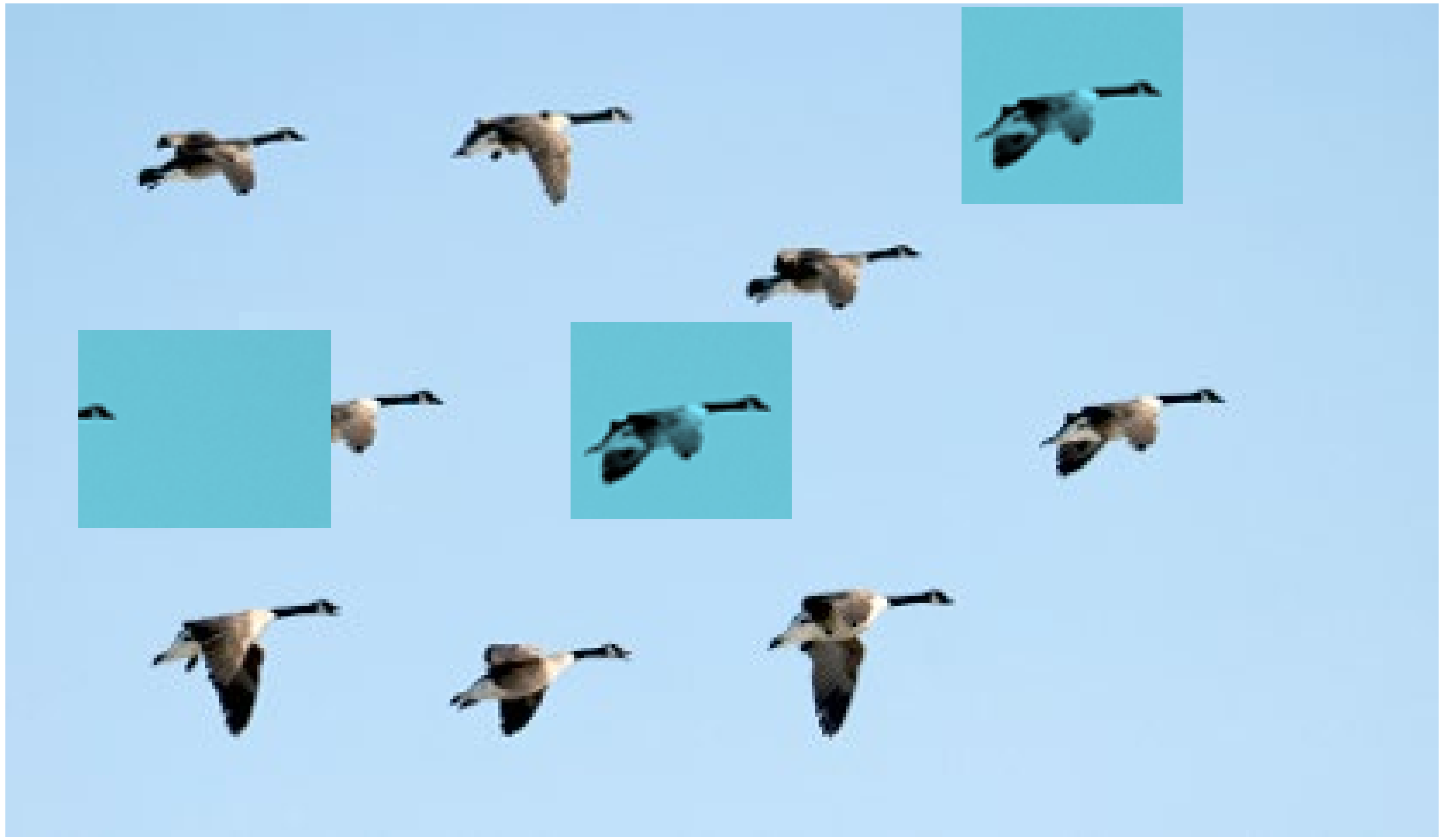
Drill down: **C**onsistency

CONSISTENT AND INCONSISTENT SNAPSHOT

Truth: 7 Geese in a V formation

Without coordination, some vanish, some are duplicated!

You might even see a goose that shifted to avoid collision with another goose, but see that other goose in a much earlier place, before it even got close.



Drill down: Consistency

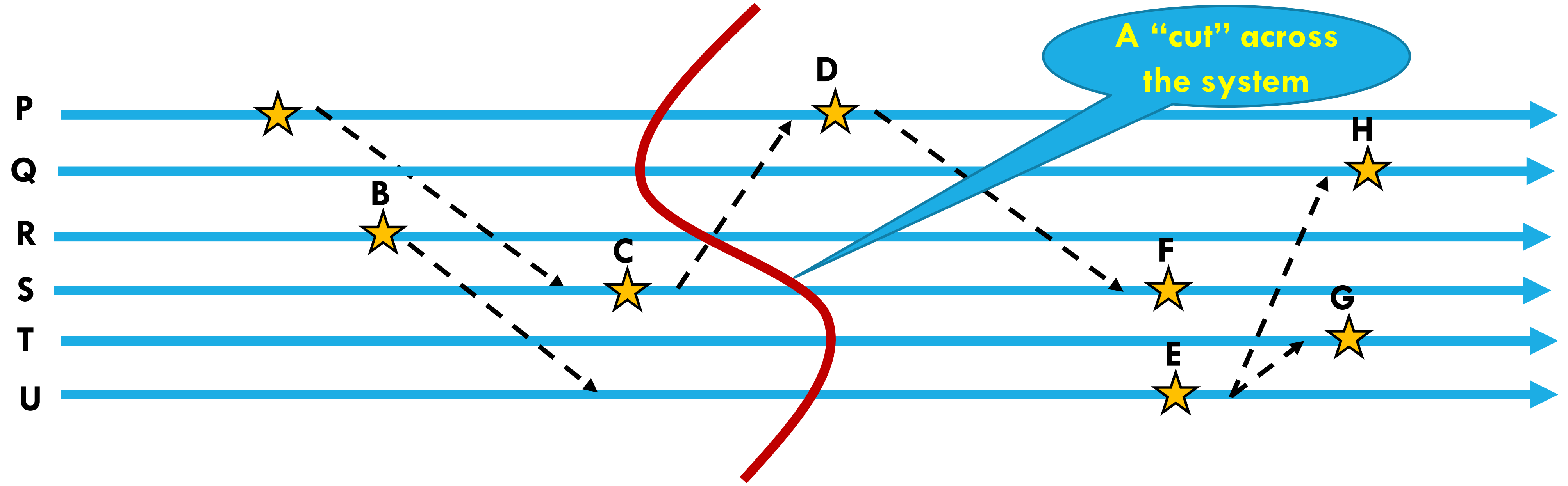
CONSISTENT SNAPSHOT

If we use the method of Chandy and Lamport we get a consistent snapshot: there won't be any duplicates or mashup effect!

Goal of a consistent snapshot is to let us combine data from multiple processes (machines) in a distributed system, but only count each thing once, with no causal gaps or duplication.

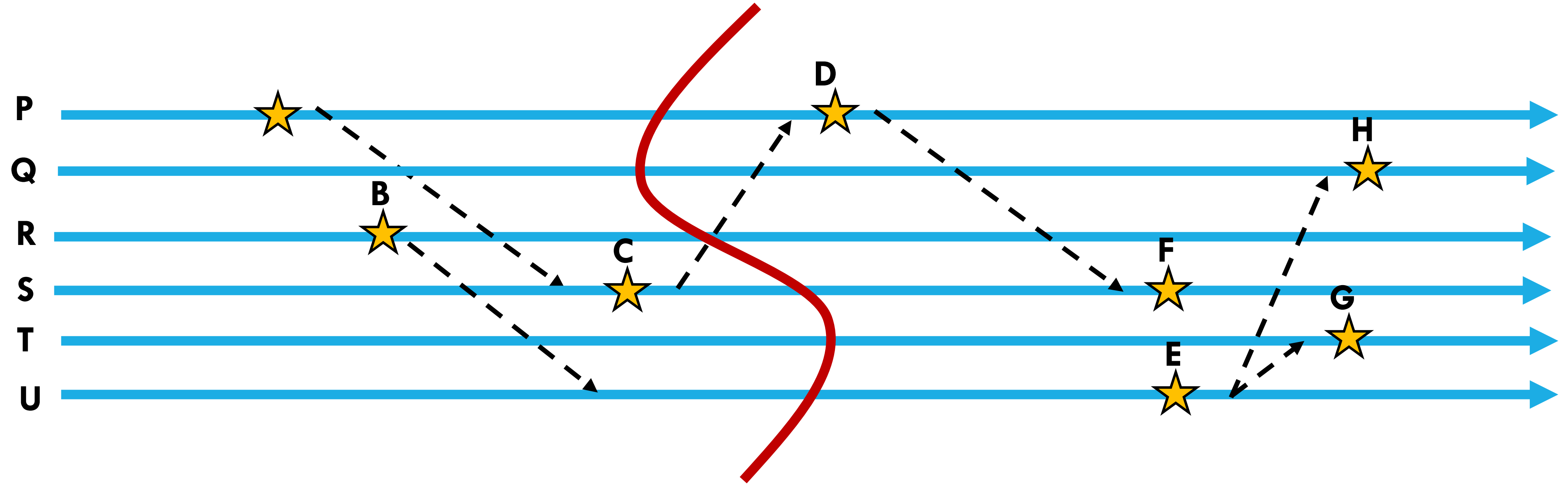
CONSISTENT CUTS AND SNAPSHOTS

Recall: Lamport looks at "pictures" of such a system, like these



CONSISTENT CUTS AND SNAPSHOTS

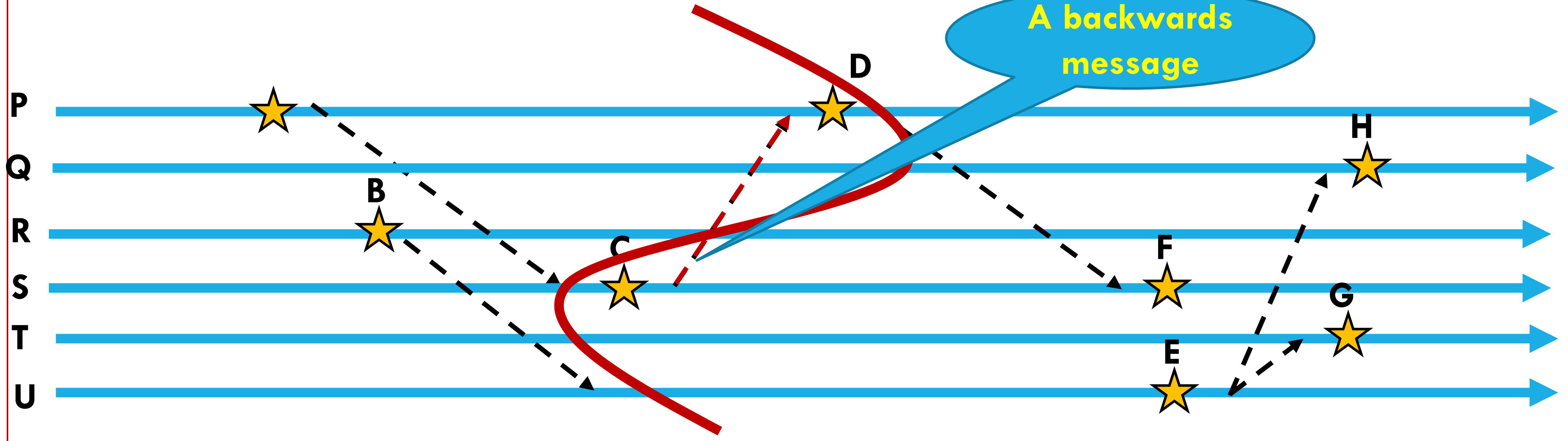
A cut is consistent if no “message arrows” go backwards through it



… this cut is a consistent one.

CONSISTENT CUTS AND SNAPSHOTS

A cut is inconsistent if "message arrows" do go backwards through it



… this cut is ***inconsistent***. C → D, and the cut included D, yet it omits C.

A CONSISTENT CUT IS LIKE A PHOTO

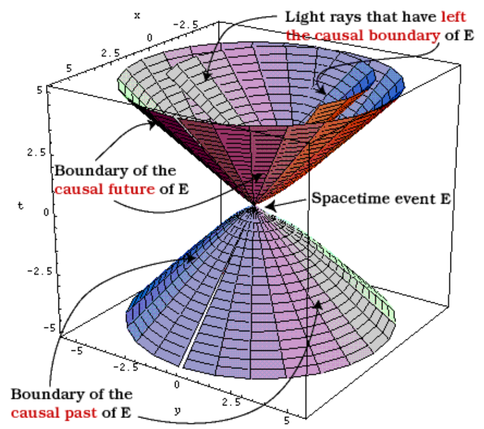
It shows a state the system *might actually have once been in*

You could use that state for garbage collection, or to do an audit of a bank, or to detect deadlocks.

But an inconsistent cut is *broken*. It omits parts of the past and any conclusion from it would be incorrect. A real system *could never* have been in an inconsistent state of this kind.

Drill down: **C**onsistency

ARE THEY UNIQUE?


Light rays that have left the causal boundary of E
Boundary of the causal future of E
Spacetime event E
Boundary of the causal past of E
x
y
t

Suppose you are standing on a timeline of some process at time T.

You want to know what the associated consistent cut/snapshot would be. In fact there isn't just one!

Events on which your state depends occurred in the past. Events depending on your event are in the future.

But this leaves a lot of freedom to include or exclude "concurrent" states.

Extra: not important, will cover if time permits

HOW TO CREATE ONE?

In fact any set of concurrent events forms a consistent snapshop.

They give protocols that will create an event "do a local snapshot now" such that those events are concurrent, one per process.

This is a mechanism found in many systems. For example, it is the very best way to do a distributed checkpoint in a long-running application.

Extra: not important, will cover if time permits

DISCOVERY BY THEO (OUR TA!)

Within these concurrent event sets, suppose we look for a consistent cut with actual time values (clock values) are close as possible to the current clock time at T, but still along a consistent cut.

If we use a type of *hybrid* timestamp invented by Sandeep Kulkarni, there is a deterministic algorithm for doing this. These have both a real-time part and a new kind of logical timestamp part.

With Theo's algorithm, you can always find the identical consistent cut. We'll see how he uses this idea in the next lecture.

Extra: not important, will cover if time permits

SUMMARY

Systems with time, but imperfect clocks, should use causality as their "time measuring tool", not actual time on a clock.

If we want to put a timestamp on an event in this approach, we should use Lamport's "causal" timestamp approach, or a vector timestamp.

- A causal timestamp is just one integer, so many systems use it
- A vector timestamp would cover all cases, but needs one integer *per machine*. So these vectors can be too large for practical use.