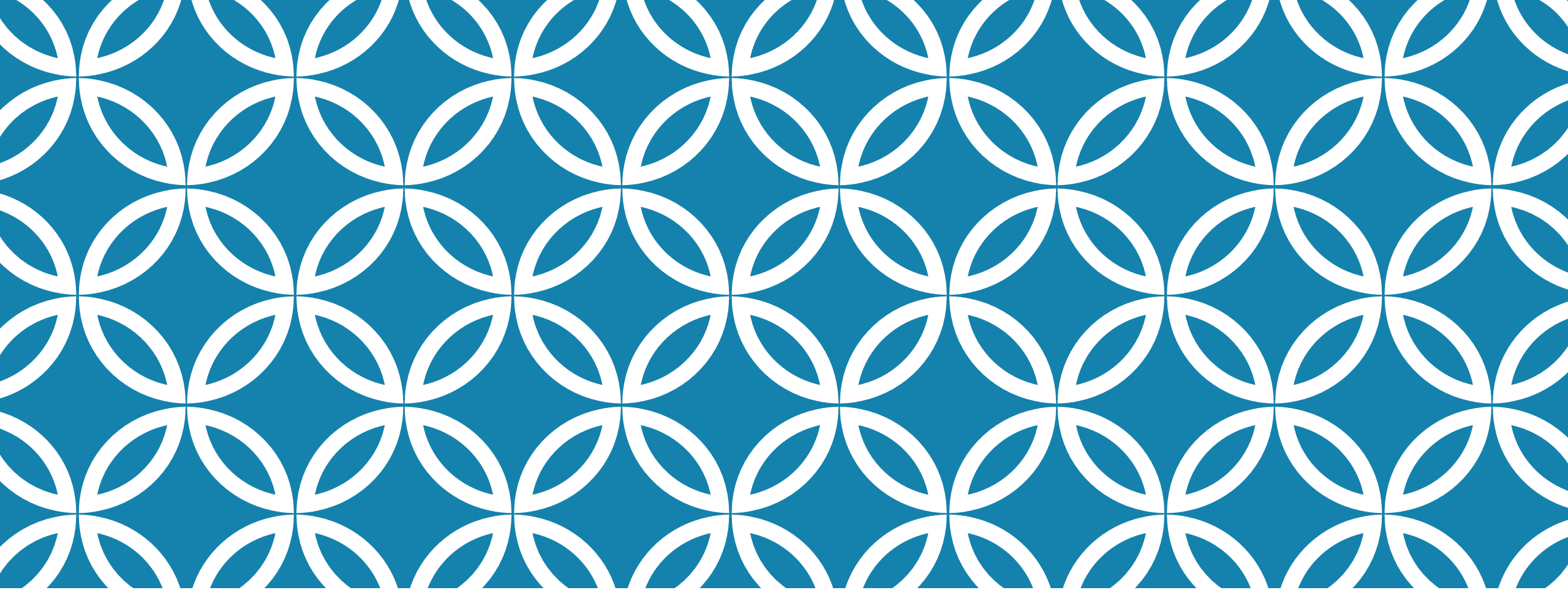


CONSISTENCY MODELS FOR DISTRIBUTED DATA AND SYSTEMS

Professor Ken Birman

CS4414/5416 Lecture 12

IDEA MAP FOR TODAY'S LECTURE

Reminder: Thread Concept

Lightweight vs. Heavyweight

Thread "context"

C++ mutex objects. Atomic data types.

The monitor pattern in C++

Problems monitors solve (and problems they don't solve)

Deadlocks and Livelocks

Distributed Computing and Consistency

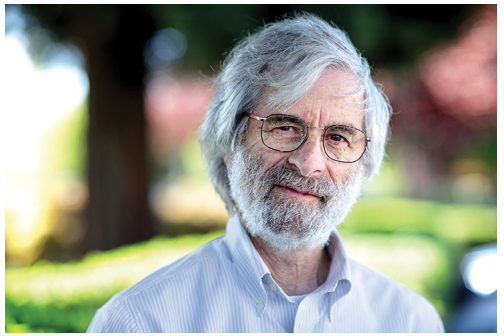
With collections of sharded data spread within a system and probably being updated continuously, don't we need a "consistency" policy?

MODELLING DISTRIBUTED SYSTEMS

Everything is a distributed system these days! Even your Apple watch is talking to your phone, and it talks to servers in the cloud

ML systems routinely harness multiple compute servers to gain more compute power. And we often spread large data sets over arrays of shards, just like Jim recommended. Then we do shard-by-shard computing.

MODELLING DISTRIBUTED SYSTEMS



The question of how powerful a distributed system really is arose long ago. Leslie Lamport is famous for studying it.

- Can a distributed system "mimic" a non-distributed one? If so, perhaps this should be our consistency model!
- What kinds of failures can be tolerated, and what methods work best? What kinds of failures really occur?

THE TOPIC GETS COMPLICATED!

Consider failure models…

- Cryptocurrency people favor a "Byzantine" failure model. Anything can happen, evil attacks are constant, collusion is a risk. This makes the "anonymous" style of blockchain costly
- Datacenter operators often focus on "clean" crash failures. But you can't always detect them (timeout is very inaccurate)
- And ML developers work with code that can crash for other reasons, like software bugs, running out of memory, etc

WE GO WITH A DATACENTER FAILURE MODEL (CRASH FAILURES)

Our focus is on the systems infrastructure and performance used for machine learning applications

So we adopt a crash failure model "combined" with an assumption that some computations "fail" for software reasons.

Failures of this kind are detectable and handled in software.

CONSISTENCY… THE BIG QUESTION

Once you set failure models to the side, we need to ask what our goals should be!

Transactional databases are a good place to start

- The data is in the database
- Applications read or do updates, or mixtures, *atomically*

WHAT ABOUT CONSISTENCY FOR DISTRIBUTED SYSTEMS?

Cornell's Derecho library has all of these features and is very fast

- **Normal message passing:** point to point or remote procedure call. Messages dropped in the network are automatically reissued. Check message id to avoid double execution.
- **Atomic multicast:** Messages to a "process group". All members receive them exactly once, in identical order.
- **Durable replicated updates:** Like atomic multicast but adds a way to also update data in files on persistent storage.

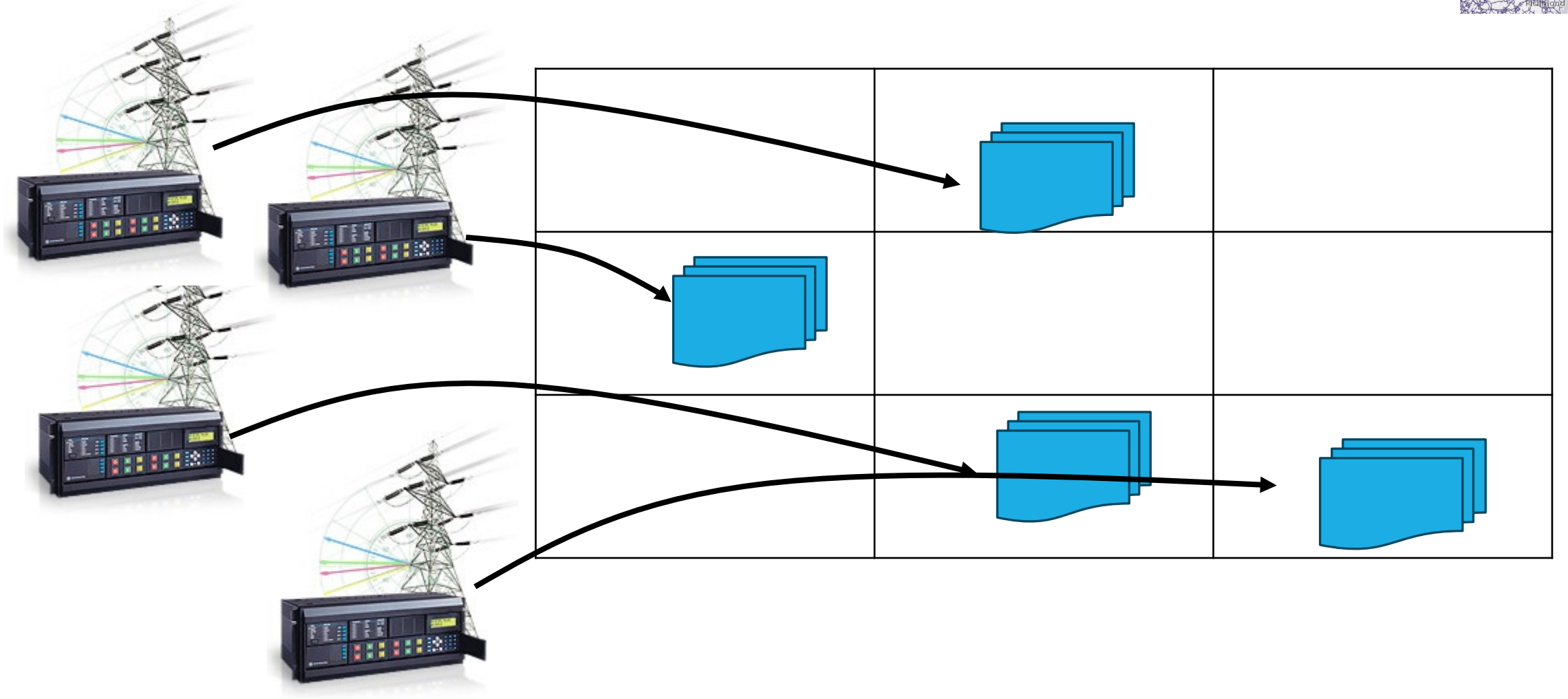
MONITORING A SMART POWER GRID

Start by deploying synchrophasors, capturing data at 20 eps, which forms a log containing timestamps

MONITORING A SMART POWER GRID

Each power bus maps to some location. Pretend for simplicity that they map to a simple grid

MONITORING A SMART POWER GRID



GIVEN LOTS OF DATA, WE CAN MAKE A MOVIE!

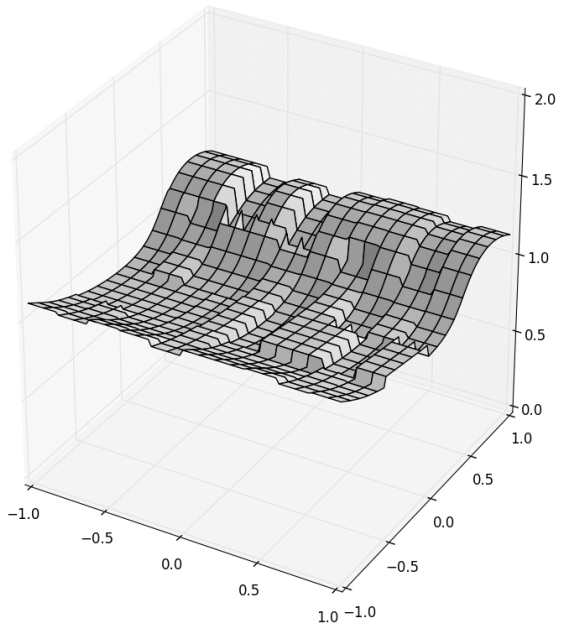
Just take **a series of snapshots**, at synchronized times

Then render the evolution of some property (we will focus on the phase of the phasor) over time. An ML might actually react.

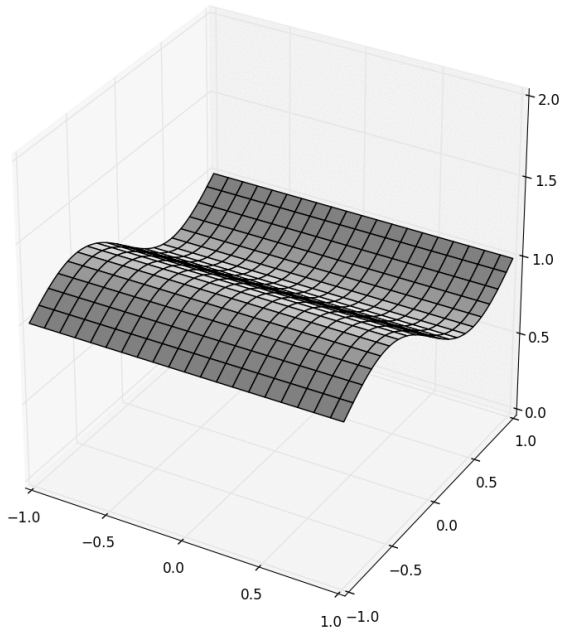
20fps is faster than needed. We'll use 4fps. But our grid will be larger: 20x20, hence 400 concurrent data flows

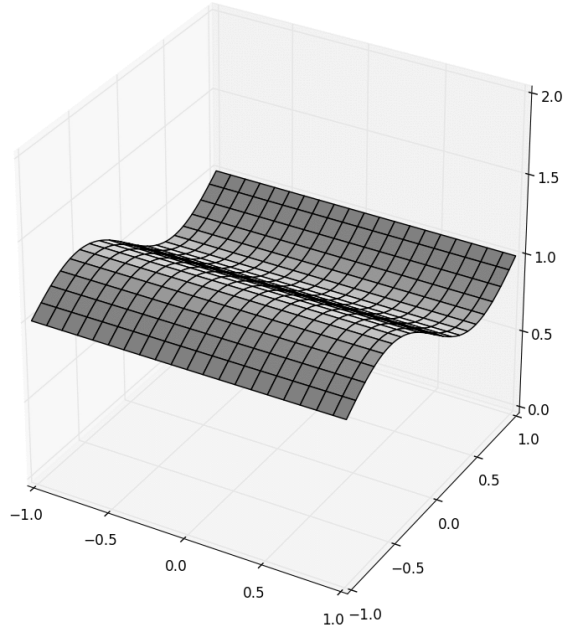
INCONSISTENCY THREATENS CORRECT BEHAVIOR!

HDFS



VORTEX USING SERVER CLOCKS… AND WITH SENSOR CLOCKS





Think of the self-driving cars that experience accidents… if an ML makes a mistake, is it because the ML is of low quality, or because it "saw" bad data?

WHAT WENT WRONG?

The animation on the left has inconsistencies

- Each “frame” is like a photo in a movie…
- But it turns out that on the left, each frame blends data from different times and sometimes, data we saw in one frame vanishes in the next frame.
- Inconsistency is like noise… and could confuse an AI system!

HOW LESLIE BUILDS UP TO A SOLUTION TO THIS PROBLEM

First, he focuses on ordering in message passing systems

Then he extends this to a concept of a distributed snapshot, like the ones used to make that little movie

LESLIE LAMPORT INVENTED MANY OF THESE MODELS FOR CONSISTENCY

He focused on the ways information can flow in a system.

In fact he introduced a special relationship operator: $\rightarrow$

- If event A occurs and A $\rightarrow$ B, then "A might have caused B" (information about A flowed through the system to B)
- A is in the past relative to B, and may even have triggered B.
- Action B may not make sense if we forget or roll back A.

LAMPORT'S CONSISTENCY MODEL

Lamport had the image in his mind of a distributed computation like an ML inference or training job.

Armed with causality, he started to think about concepts of time, handling of failures, checkpoints and rollback.

He proposed the idea of a "consistent cut".

TRACKING CAUSALITY: LAMPORT'S $A \rightarrow B$

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 $A \rightarrow B$

Drill down: **C**onsistency

DEVELOPING A SOLUTION

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

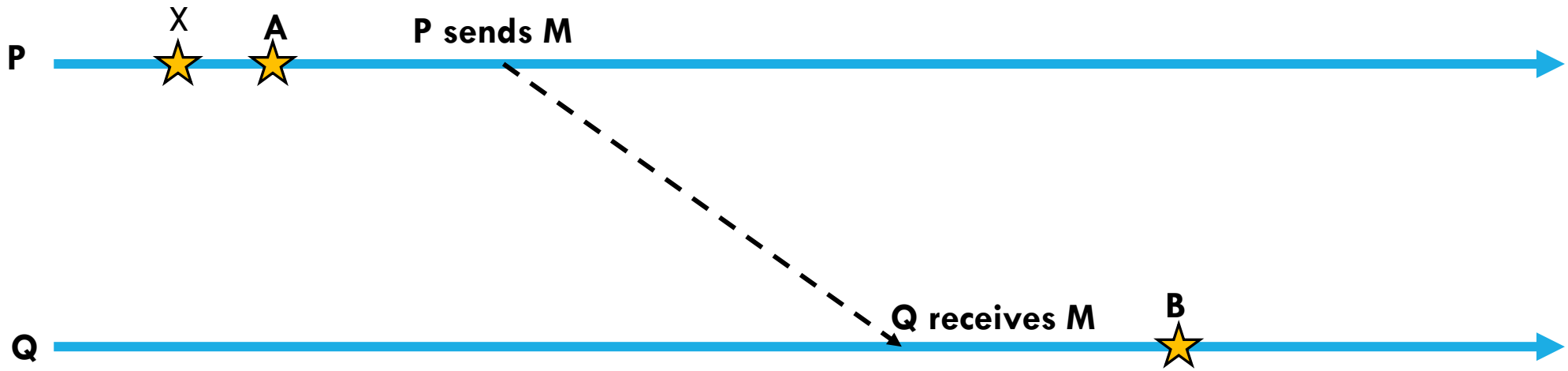
Call these LogicalClock_P and LogicalClock_Q etc.

Each time something happens, increment the clock.

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

Drill down: CAP **C**onsistency

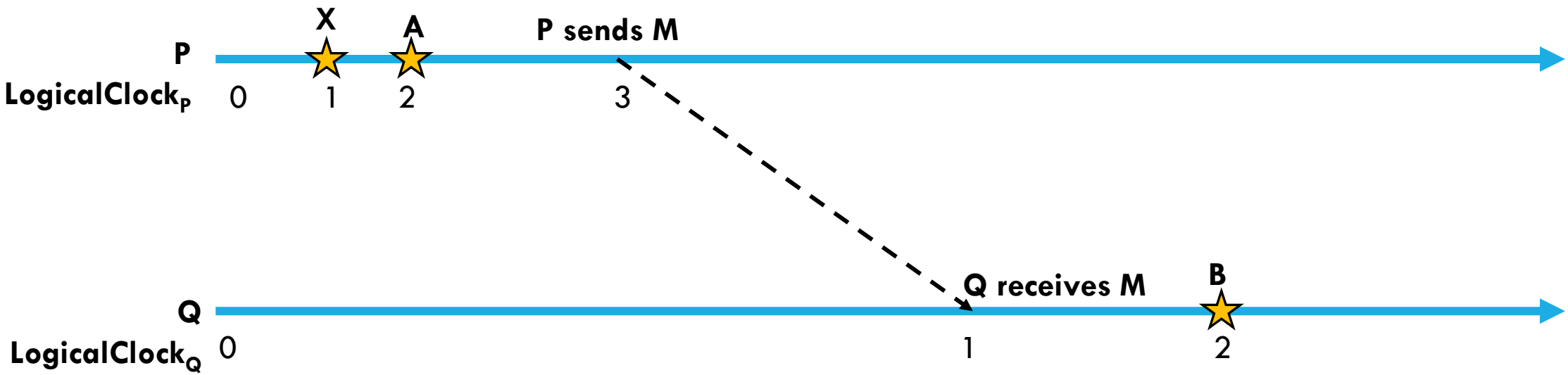
A SPACE-TIME DIAGRAM FOR THIS CASE



Drill down: **C**onsistency

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

Drill down: **C**onsistency

AHA!

Notice that the "receive" of M occurs when $\text{LogicalClock}_B = 1$.

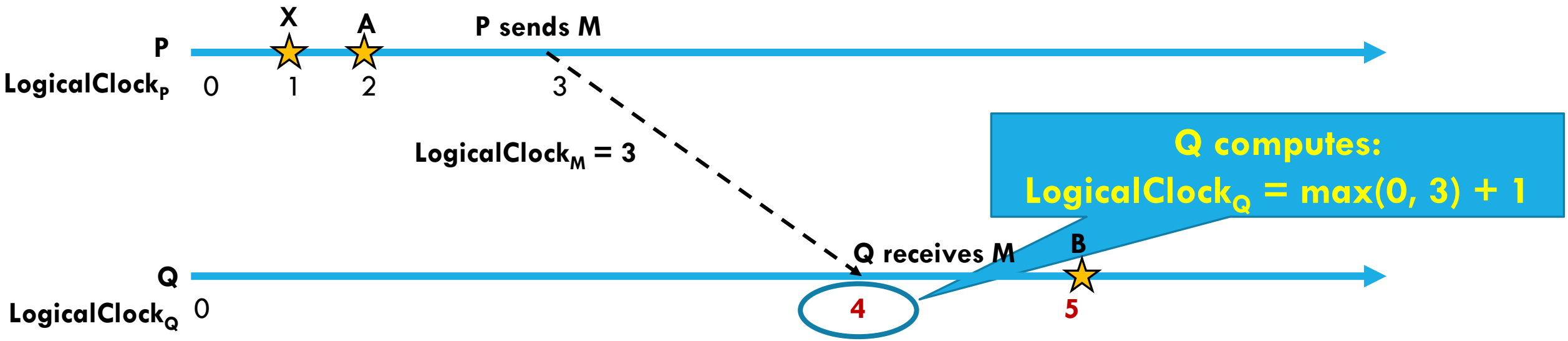
Yet the "send" of M was at $\text{LogicalClock}_A = 3$.

So Lamport proposes this idea:

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

Drill down: **C**onsistency

A SPACE-TIME DIAGRAM FOR THIS CASE



Drill down: **C**onsistency

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.

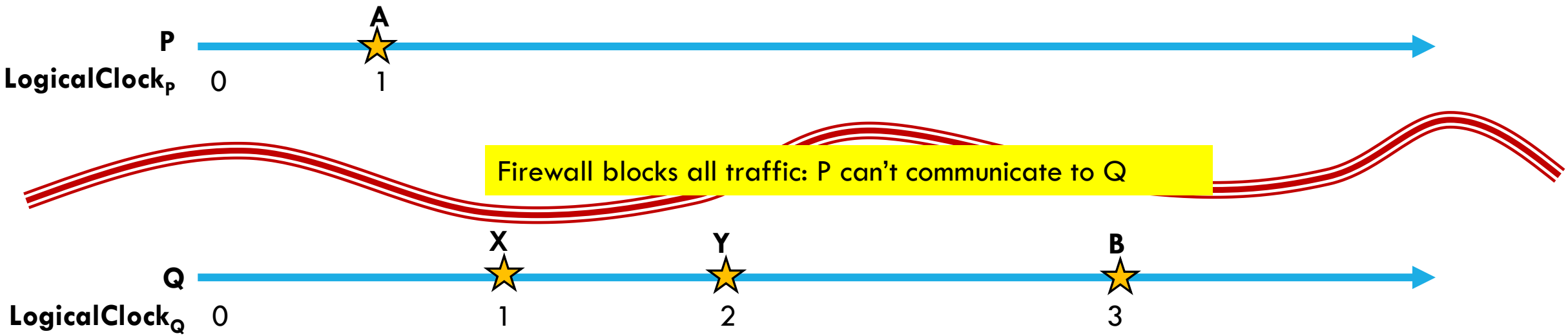
- If A $\rightarrow$ B, then LogicalClock(A) < LogicalClock (B)
- But… sometimes LogicalClock (A) < LogicalClock (B),

  yet A didn't happen before B!

Drill down: **C**onsistency

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")

Drill down: **C**onsistency

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.

Drill down: **C**onsistency

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:

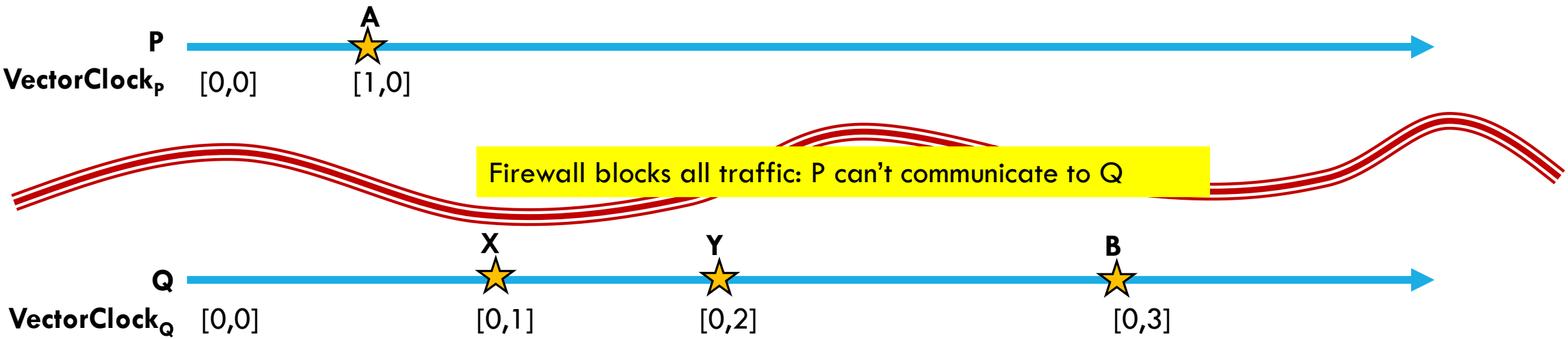
Define VT(A) < VT(B) if
VT(A) ≤ VT(B),
but VT(A) ≠ VT(B)

Now, VT(A) < VT(B) iff A → B

Drill down: **C**onsistency

A SPACE-TIME DIAGRAM FOR THIS CASE

Case A: Suppose that P and Q never interact.

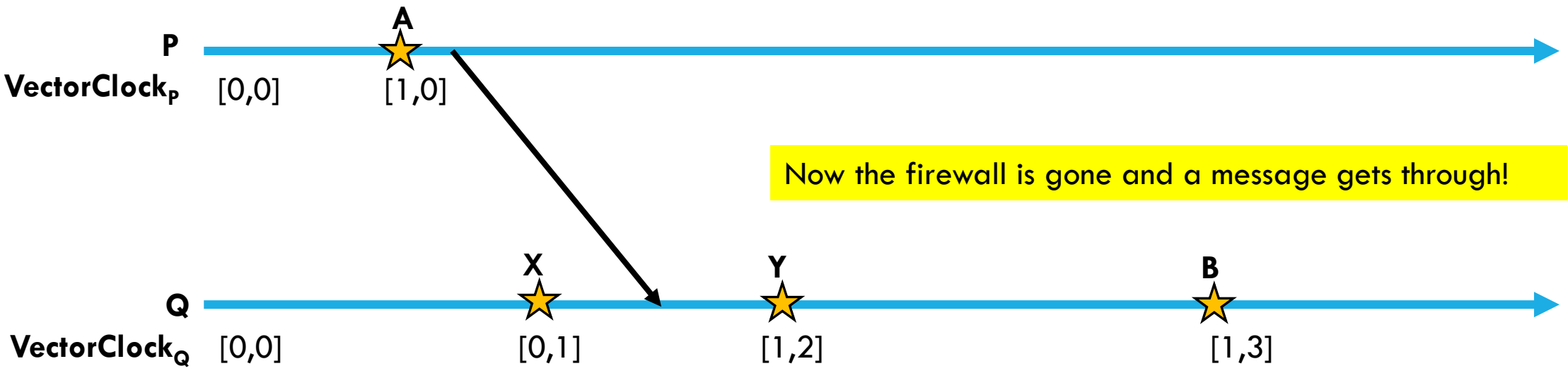


With vector clocks we can see that A is concurrent with X, Y and B. We can use the comparison rule to show this, for example that $\neg(A \rightarrow B)$ and $\neg(B \rightarrow A)$.

Drill down: **C**onsistency

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).

Drill down: **C**onsistency

SO WHY NOT ALWAYS USE VECTOR CLOCKS?

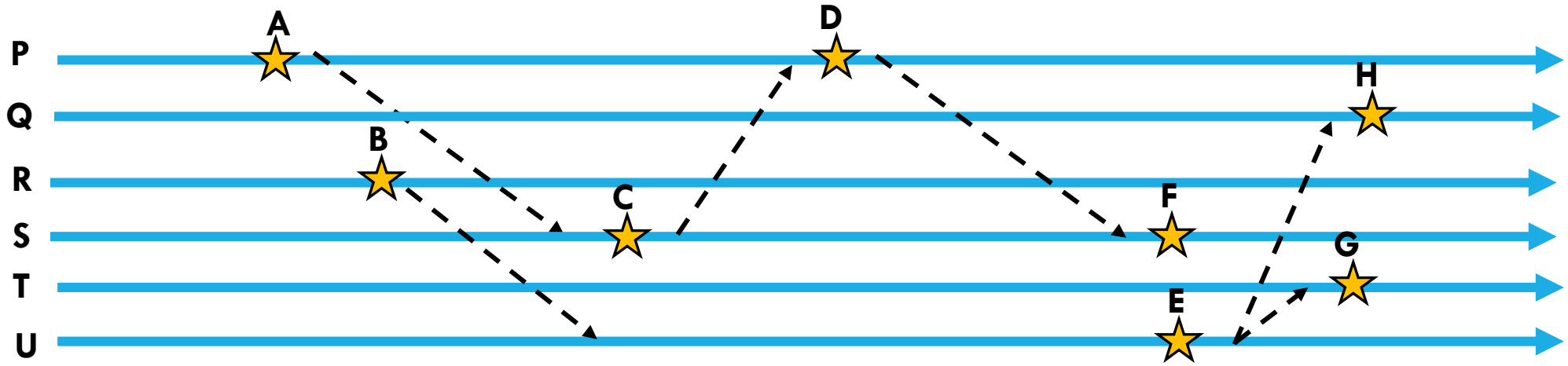
They represent happens-before with full accuracy, which is great.

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.

CONSISTENT SNAPSHOTS AND CUTS

Lamport and Mani Chandy built a solution to this problem using the basic A → B logical clock as a building block. Consider a time-space picture like this



Drill down: **C**onsistency

CONSISTENT CUTS AND SNAPSHOTS

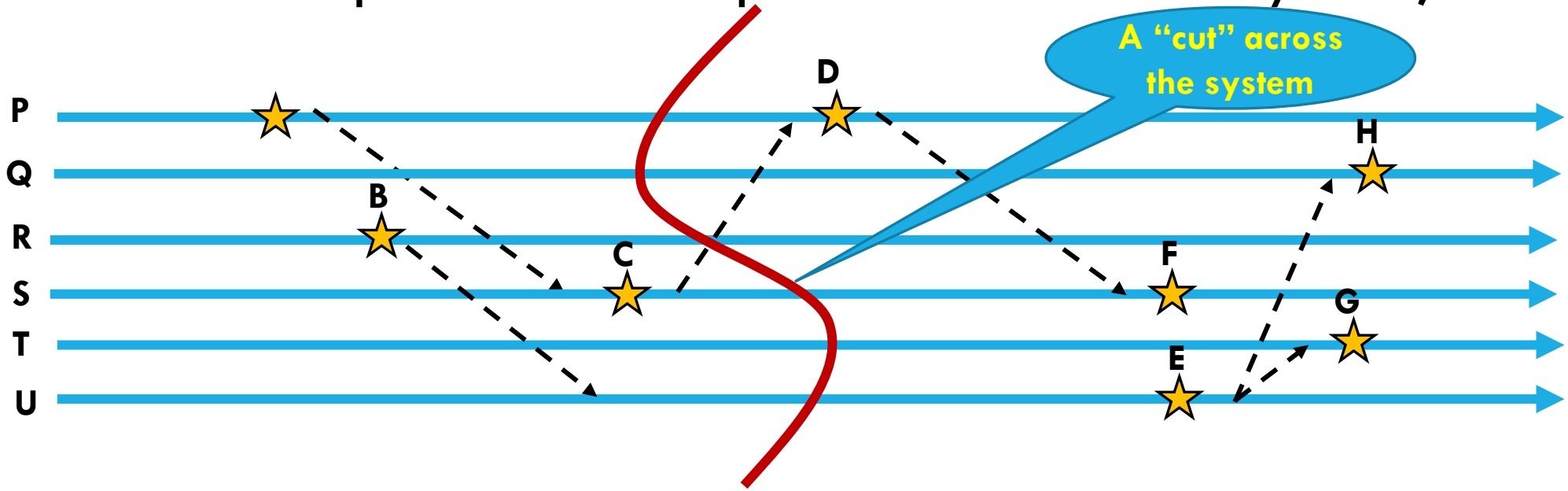
They asked: Suppose I visit each node, each at some point in time. Can we extend consistency to cover such a case ("consistent cut")

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?

Drill down: **C**onsistency

CONSISTENT CUTS AND SNAPSHOTS

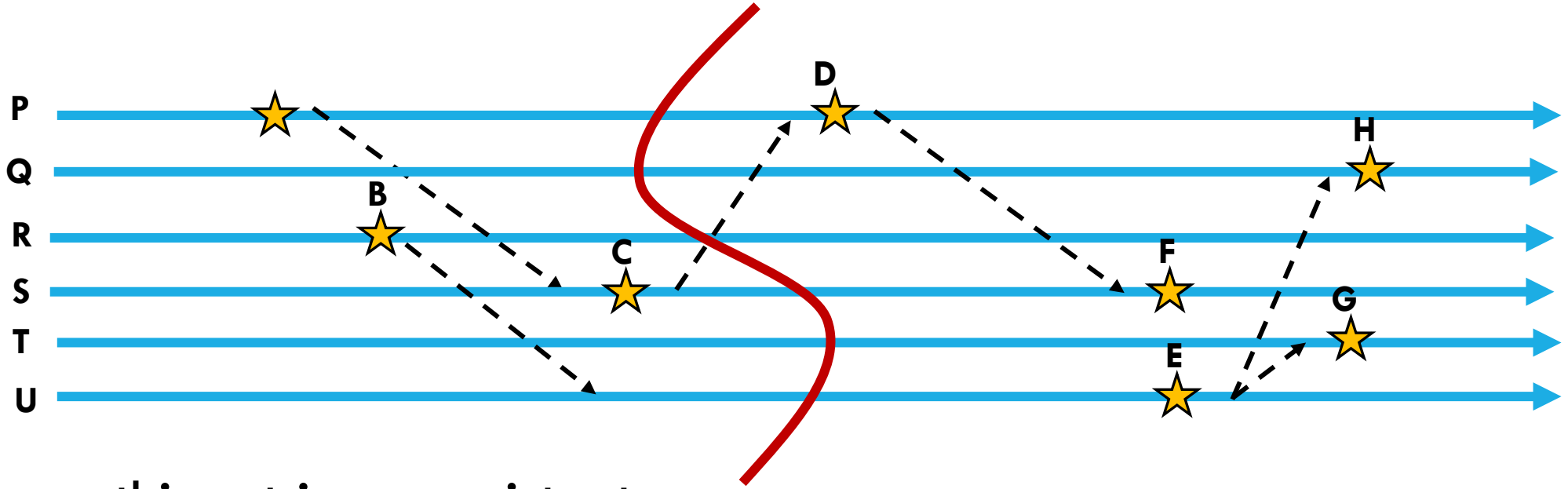
Recall: Lamport looks at "pictures" of such a system, like these



Drill down: **C**onsistency

CONSISTENT CUTS AND SNAPSHOTS

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

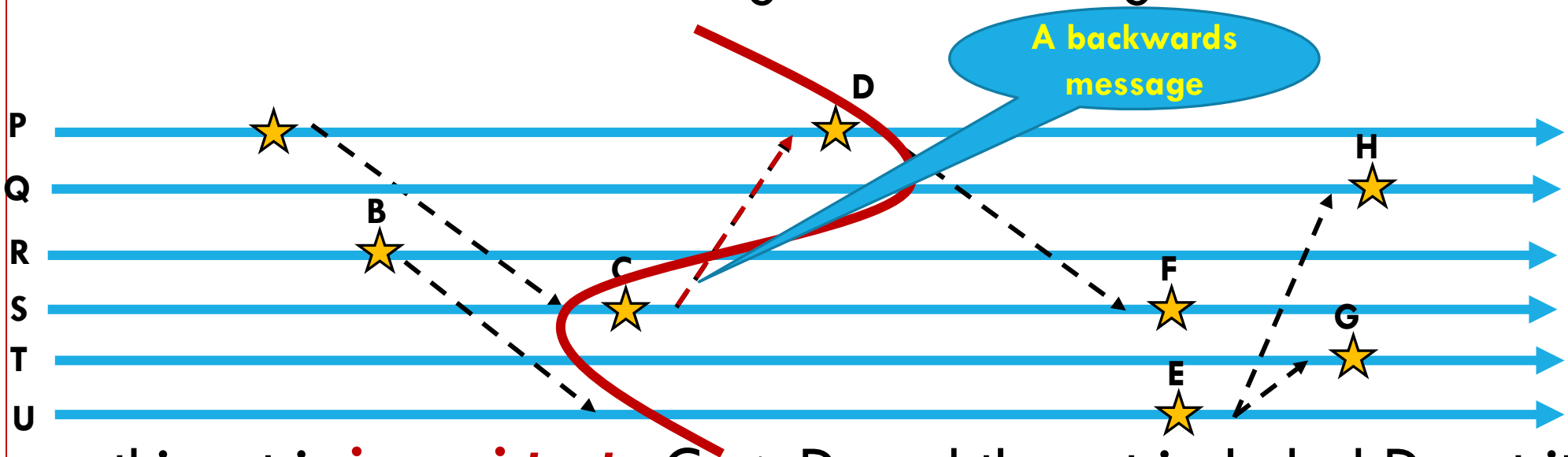


… this cut is a consistent one.

Drill down: **C**onsistency

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.

Drill down: **C**onsistency

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

HOW TO CREATE ONE?

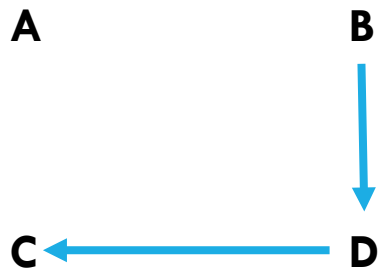
Chandy and Lamport give an algorithm based on a form of broadcast. It records the state of each process, and also snapshots messages still in the communication network. It works while the system is still running – no need to first freeze things.

A packaged version of this mechanism as a library is available in many systems. For example, it is one option for making a distributed checkpoint in a long-running application.

DEADLOCK DETECTION: "ASK EACH PROCESS WHAT IT IS CURRENTLY DOING"

With a guarantee of consistency, a cycle is a deadlock

But with inconsistency, we could be missing a "lock release" message that "allowed" some process to start running, after which it requested another lock.

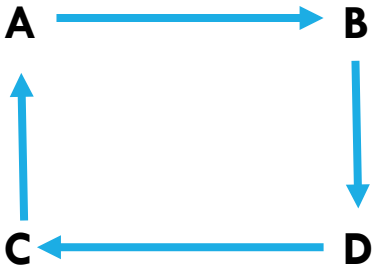


Here, D is waiting for C to release a lock. B is waiting for D.

DEADLOCK DETECTION: "ASK EACH PROCESS WHAT IT IS CURRENTLY DOING"

With a guarantee of consistency, a cycle is a deadlock

But with inconsistency, we could be missing a "lock release" message that "allowed" some process to start running, after which it requested another lock.

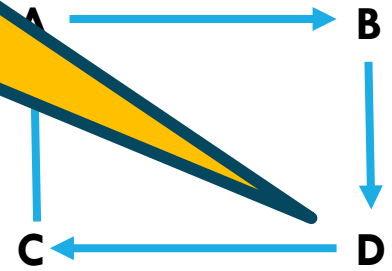


This picture makes sense if D is still waiting for C, and B for D.

DEADLOCK DETECTION: "ASK EACH PROCESS WHAT IT IS CURRENTLY DOING"

With a guarantee of co

Semantically, when D says "I'm waiting for C" it should mean "at the instant of the snapshot"… but the requests show up concurrently and not necessarily simultaneously!



But with inconsistency, we could be missing a "lock release" message that "allowed" some process to start running, after which it requested another lock.

It is wrong if D is no longer waiting for C… the deadlock could be an illusion!

HOW ARE CONSISTENT CUTS USED?

Automated deadlock detection really is used in distributed database systems

Banks allow users to transfer money from account to account. A consistent cut reports the right balances.

… Lorenzo Alvisi had a cool idea, too!

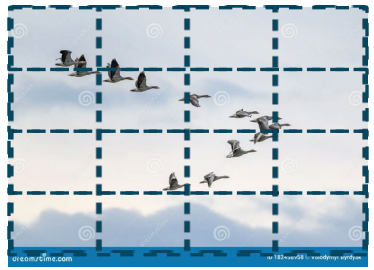
ML CHECKPOINTS AND ROLLBACK ENABLE FAULT TOLERANCE. BUT CAN WE CHECKPOINT EACH SUBSYSTEM SEPARATELY?

In a distributed system, if only parts of it roll back, the state might not be consistent. This was Lorenzo's PhD topic.

His algorithm allows asynchronous checkpoints, can log messages on sending side or receiver, and rolls back on consistent cuts.

Today we are looking at his methods in the context of checkpointing for ML training jobs with many subsystems and hours or days long training runs!

HOW ARE THEY NOT USED?



Counting geese? Don't use an array of cameras and a consistent cut: logical properties are internal to a system and don't always match outside real-world goals.

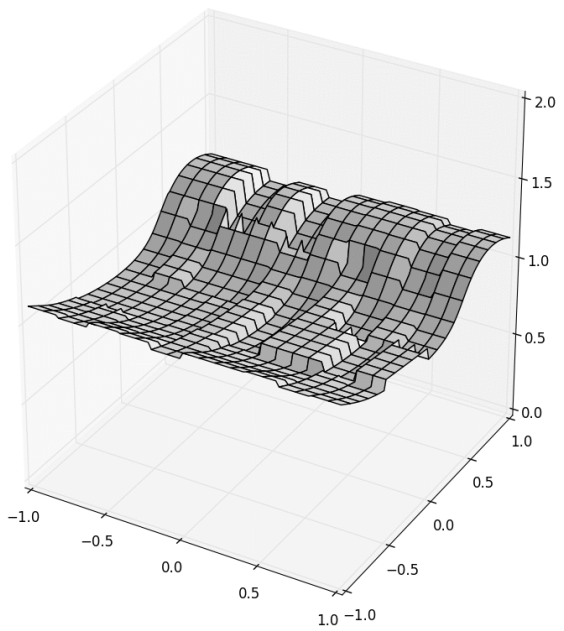
Although "a photographic snapshot" is a common intuition for a consistent cut or snapshot, this intuition is actually wrong.

There is no connection between logical time (used for cuts) and physical time (used by a camera).

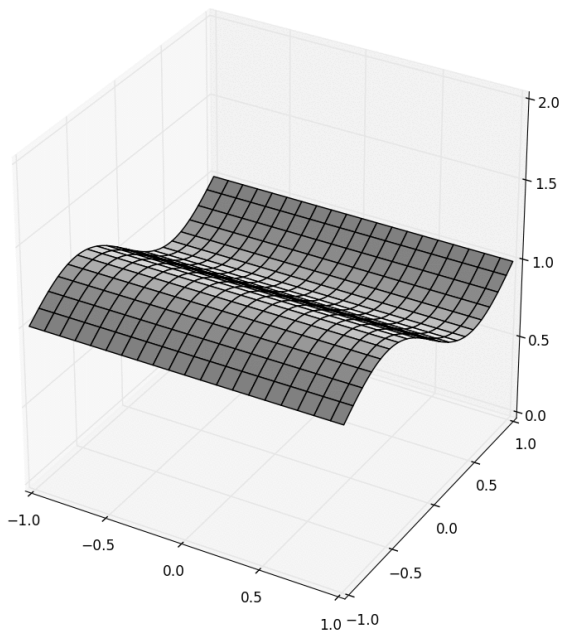
A consistent cut is only a snapshot if the things in the system are all software – not if the system has some tie to the outside real world. If you need a photograph of the real world, don't use consistent cuts!

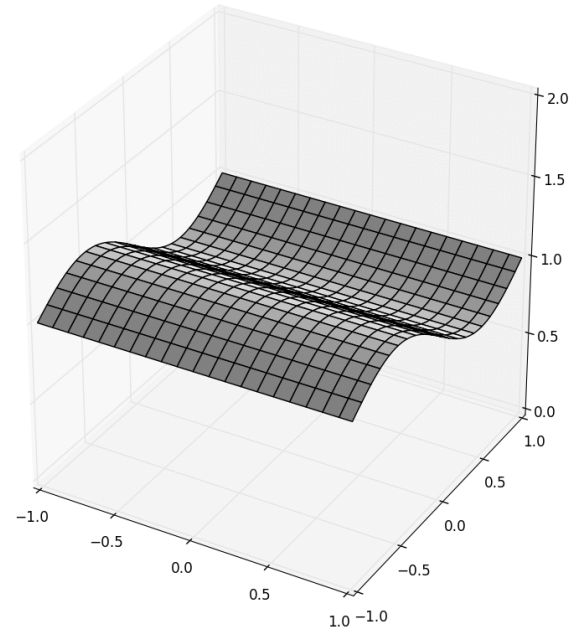
… AND REMEMBER OUR SMART GRID?

HDFS



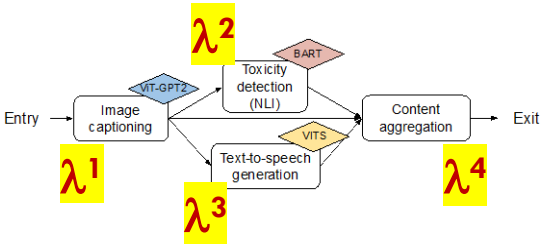
VORTEX USING SERVER CLOCKS… AND WITH SENSOR CLOCKS





Vortex uses a variation on the Chandy/Lamport algorithm to ensure that when you read data, you'll see a consistent snapshot. HDFS doesn't.

HOW DOES VORTEX DO IT?
STATE MACHINE REPLICATION PLUS CONSISTENT CUTS



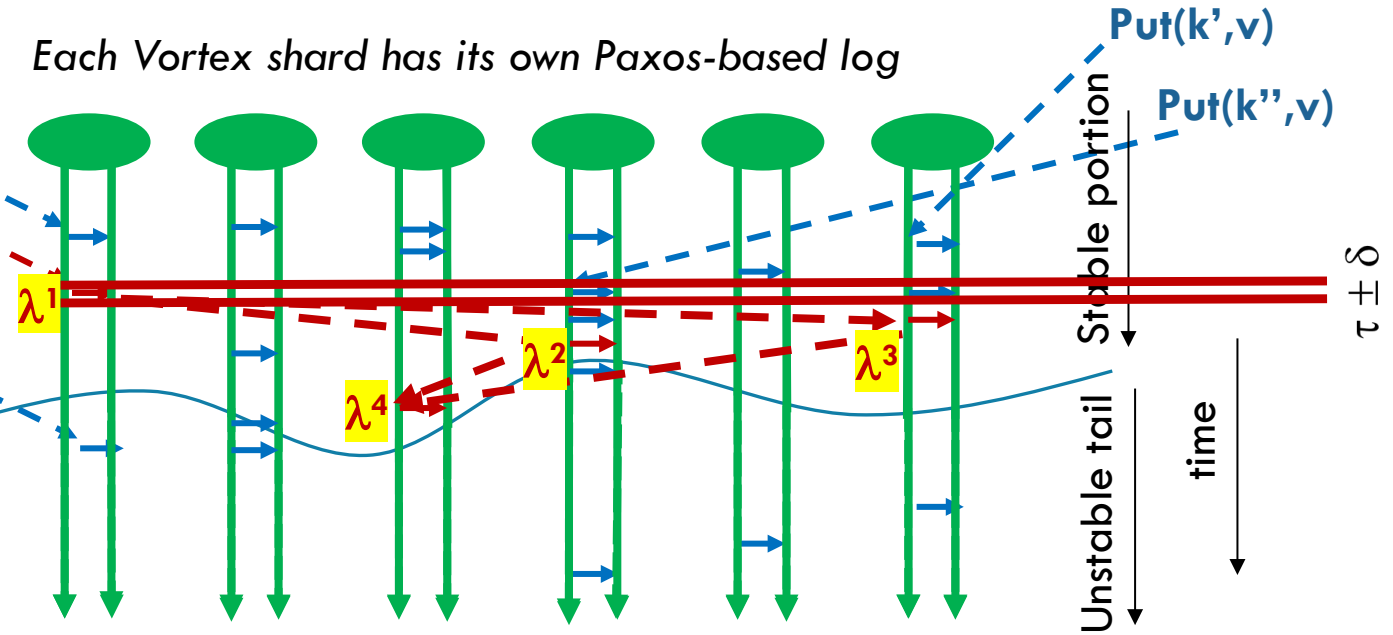
(ε, τ) → Put(k,v)

Put(k,v)

Put(k''',v)

Each λ is triggered by an upcall from a "watcher" monitoring some key (or pattern)

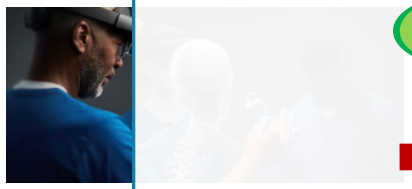
Each Vortex shard has its own Paxos-based log

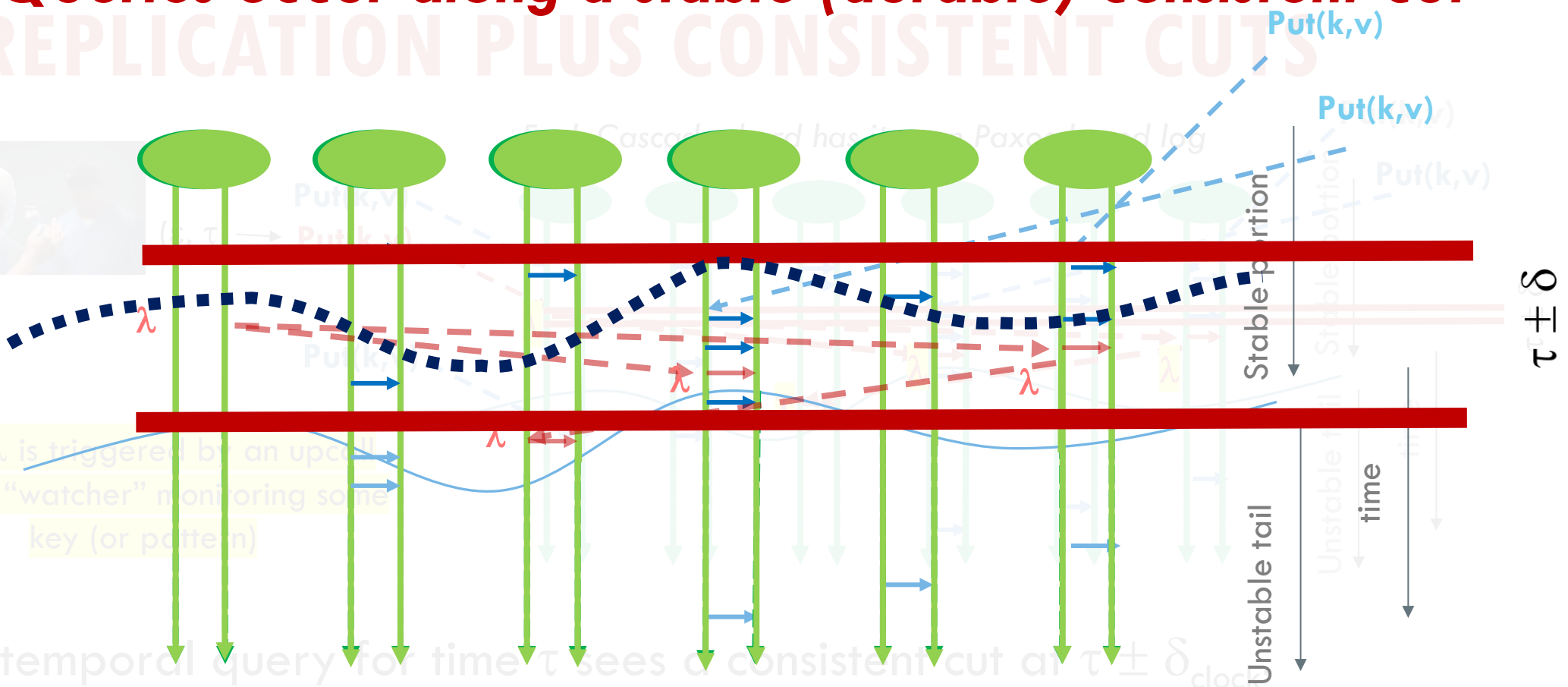


A temporal query for time τ sees a consistent cut at $\tau \pm \delta_{clock}$.

Queries to unstable data must wait, but updates are stable within 50us.

Queries occur along a stable (durable) consistent cut





Each λ is triggered by an update from a "watcher" monitoring some key (or pattern)

A temporal query for time τ sees a consistent cut at τ ± δ_{clock}

Queries to unstable data must wait, but updates are stable within 50us.

SUMMARY

Lamport (and Einstein) both believe that clocks are an ill-defined way to think about time: too much depends on frame of reference

Clocks can also fail in very unintuitive ways.

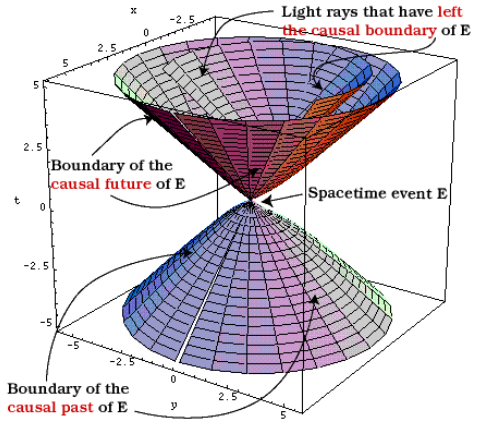
Logical notions of consistency and time lead to very elegant designs and yet can be pretty far from “wallclock time”

THOUGHT QUESTIONS

We use the term “snapshot” but is this the right intuition? Do snapshots ever have any connection to clock time?

Could snapshots be made to match clock time?

ARE CONSISTENT CUTS UNIQUE?

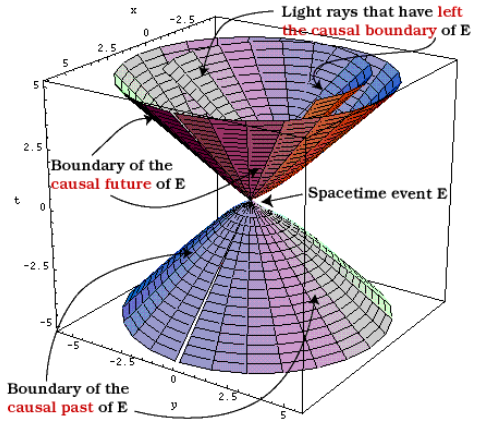


Suppose you are interested in the system state corresponding to when some process P was at time T_P according to its local clock.

You decide to create a consistent cut/snapshot where P's state would be exactly the one it had at time P_T

Is there just one that could match, or could there be many?

THOUGHT QUESTION



What is the "most skewed" consistent cut that can still be consistent? Does it have any relationship at all to "instant in real-time" or does it reveal that a consistent cut is often *definitely* not a single instant in real-time?

Hint: the Einstein space-time diagram is helpful for visualizing this.

THOUGHT QUESTION

Lamport pointed out that to audit a bank, you should use a consistent snapshot.

What are some examples of **errors that could confuse an audit** if you collect the states from all the bank's branches without making a snapshot (and without freezing everything)?

Why does a consistent snapshot avoid those errors?

THOUGHT QUESTION

How can concept of real-time from synchronized clocks be integrated into the concept of a consistent snapshot?

[Read more: The Freeze-Frame File System. Weijia Song, Theo Gkountouvas, Qi Chen, Zhen Xiao, Ken Birman. ACM Symposium on Operating Systems Principles (SOCC 2016). Santa Clara, CA, October 05 - 07, 2016.]

EXAM PREPARATION ADVICE

Exams sometimes have questions about potential causality ($\rightarrow$), or about the idea of a consistency cut/snapshot

But we never ask you about algorithms Lamport for implementing them: a topic for the advanced distributed systems course.

The need for this property is our focus, not the way tools get it.