Time and Ordering in Streaming Distributed Systems

Zhenzhong Xu
Real-time Data Infrastructure
Netflix

@ZhenzhongXu

Time and Ordering in Streaming Distributed Systems

Zhenzhong Xu
Real-time Data Infrastructure

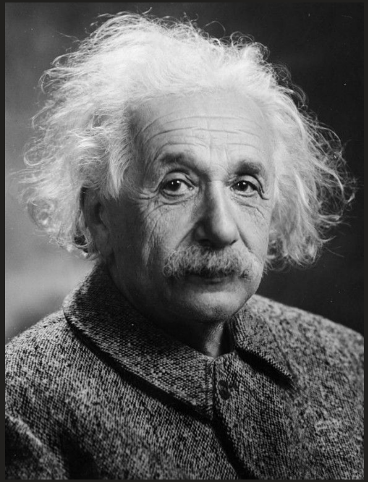
NETFLIX @ZhenzhongXu

Software engineers think of time as -

- Uniformly measurable
- One directional
- Infinite precision
- Time manifests ordering of events

“Time no longer appears to us as a gigantic, world-dominating chronos, nor as a primitive entity, but as something derived from phenomena themselves. It is a figment of my thinking.”

— Schrödinger, Erwin.



“Time is an illusion.”

— Einstein, Albert.

Distributed System

No shared memory, only message passing via **unreliable network** with **variable delays**, and the system may suffer from **partial failures**, **unreliable clocks** and processing **pauses**.

Stream processing connects distributed systems together, over space and time, designed with unbounded data set in mind.

Stream Processing at Netflix

- Keystone Data Pipeline
- Operation insights
- Business analytics
- Event sourcing pattern

Categories of streaming

- **Time agnostic**

- Transformation
- Filtering
- Projection
- Enrichment
- Inner joins

Categories of streaming

- Time agnostic
- **Approximation**

- Approximate top-n
- Streaming k-means
- etc

Categories of streaming

- Time agnostic
- Approximation
- **Windowing**

- Fixed / Tumbling
- Sliding / Hopping
- Session / Dynamic

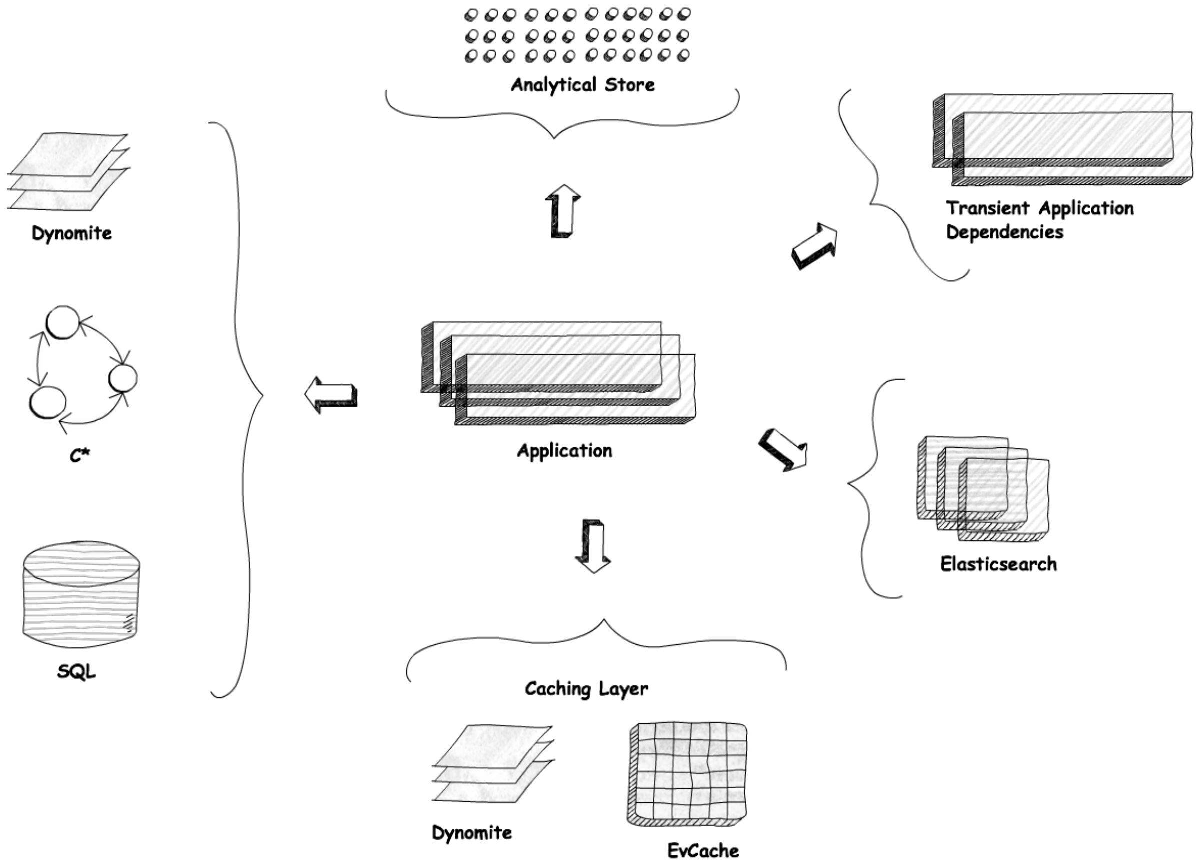
Project Delta

Eventual consistent, event-driven data synchronization platform

- Event sourcing
- Windowing

Challenges:

- Semantics of ordering
- Latency vs. durability
- CDC
- etc



NETFLIX

... via the three lens of time

- Uniformity of time
- Arrow of time
- Perception of time

#1

Uniformity of time

Time is a tool ...

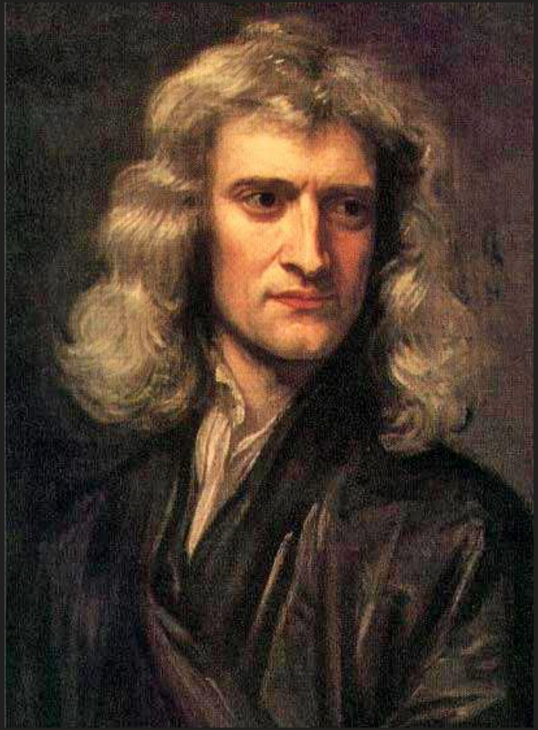
Time is a tool ...

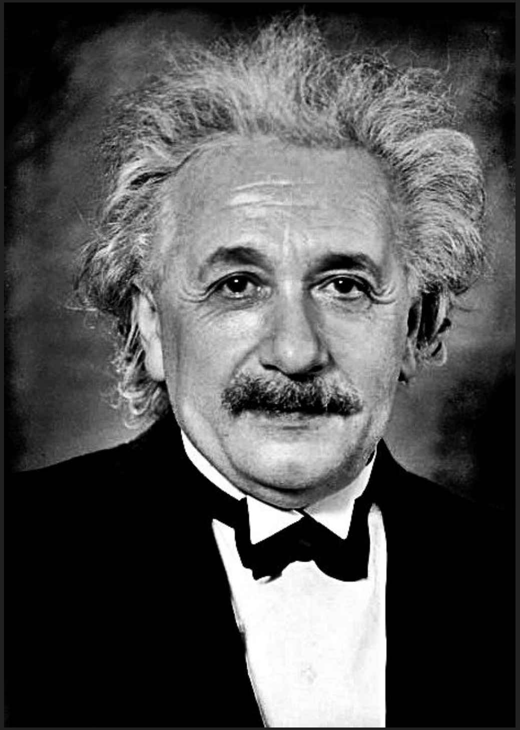
Time is a tool ...

Need for synchronization?

= uniform time

Standing on the shoulder of giants



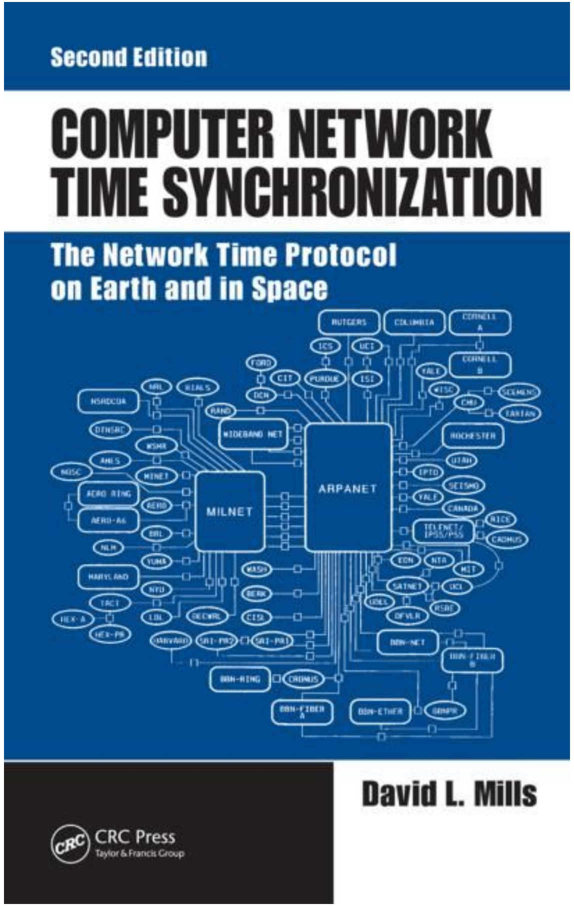


Time flows slower closer to a black hole

Scene from the movie Interstella, depicts time flows slow closer to the supermassive blackhole “Gargantuan”

Clock synchronization over network

"NTP can usually maintain time to within tens of milliseconds over the public Internet, and can achieve better than one millisecond accuracy in local area networks under ideal conditions. Asymmetric routes and network congestion can cause errors of 100 ms or more."



https://en.wikipedia.org/wiki/Network_Time_Protocol

Relying on synchronized wall clock timestamps?

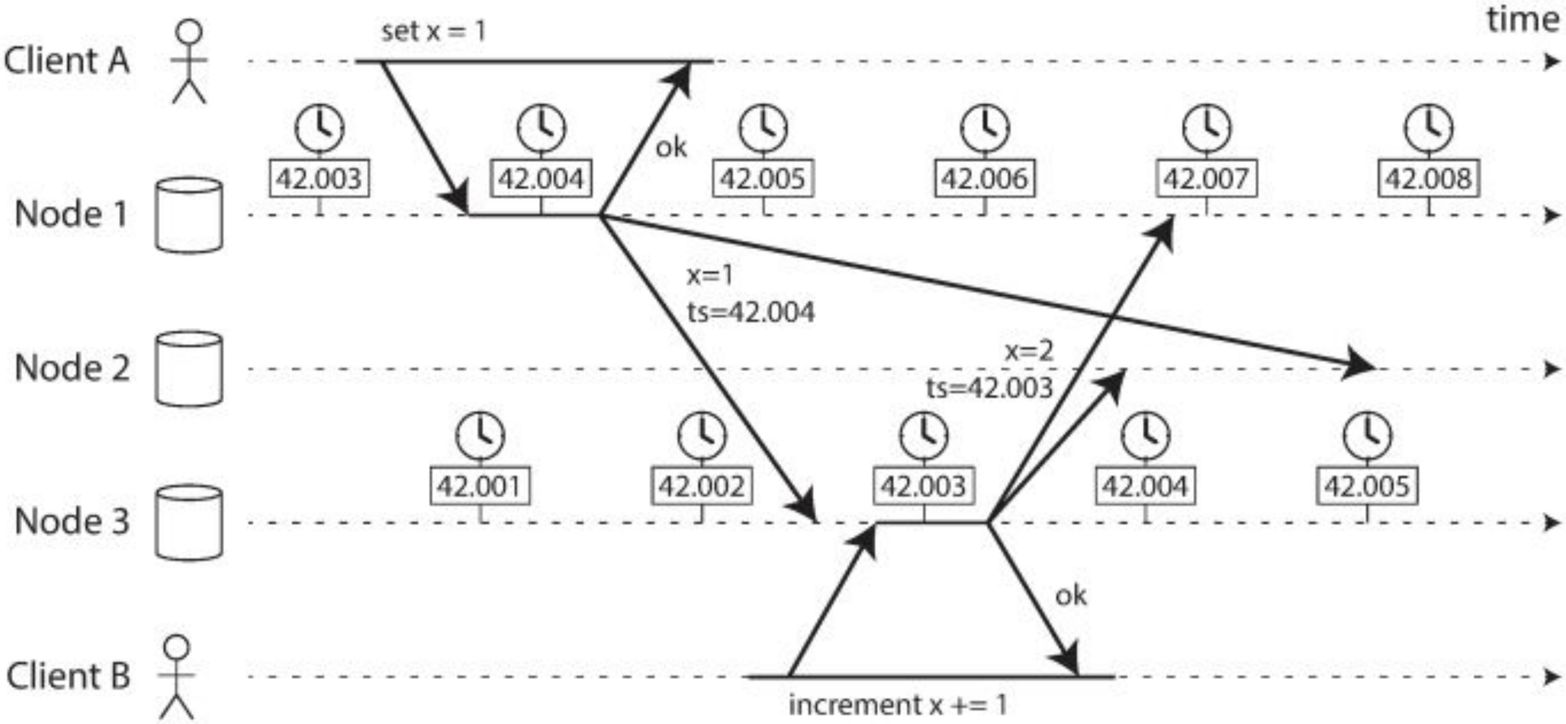
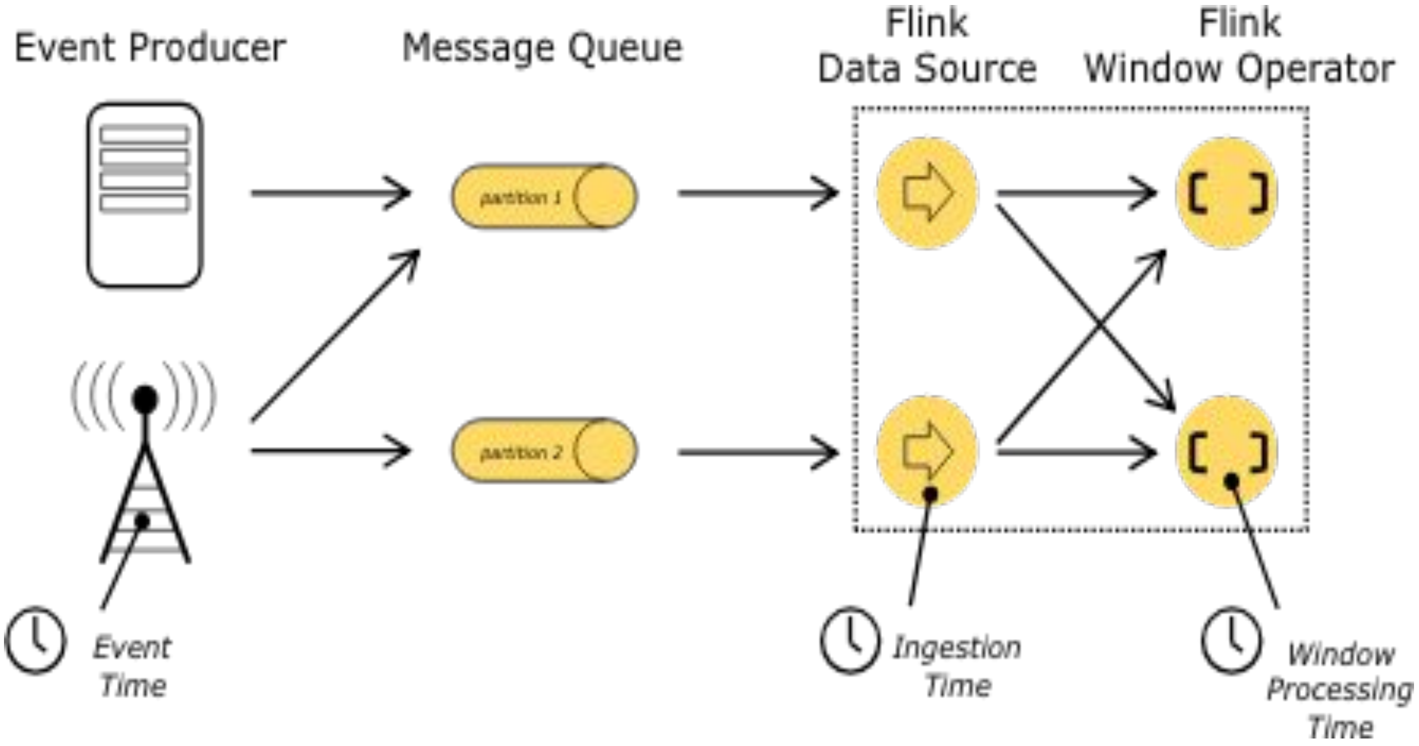


Figure referenced from Designing Data Intensive Applications by Martin Kleppmann, Chapter 8 Trouble with Distributed Systems

Time in Stream Processing



https://ci.apache.org/projects/flink/flink-docs-stable/concepts/programming-model.html

NETFLIX

Why time skews

- Information travel takes time
- Low power device
- Process failure
- Unpredictable network congestions
- Timeouts and unbounded delays
- Unreliable clock
- Process pauses
- etc

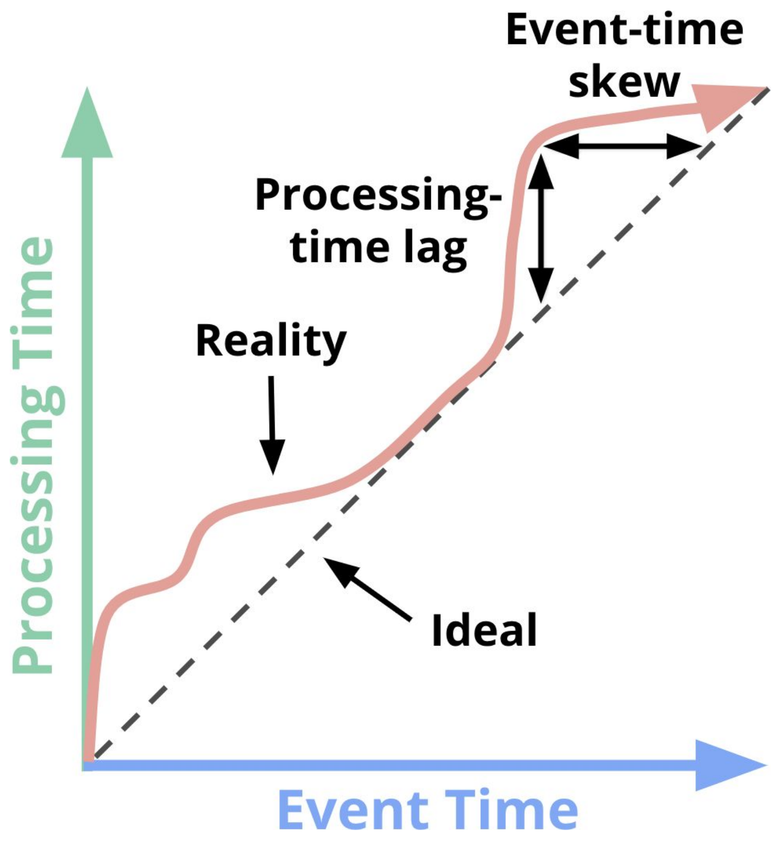


Figure referenced from Streaming Systems by Tyler Akidau el al, Chapter 1 Streaming 101

NETFLIX

Watermark in action

Animation referenced from Streaming Systems by Tyler Akidau el al, http://streamingbook.net/fig/2-11

NETFLIX

Use watermark to bound the uncertainties of time

- Allowed lateness

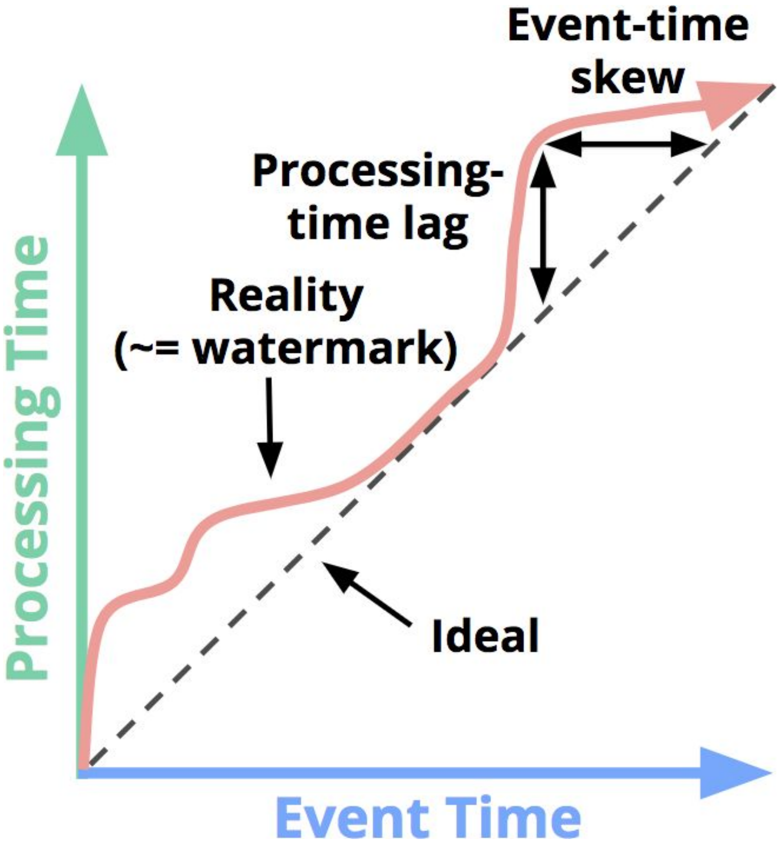
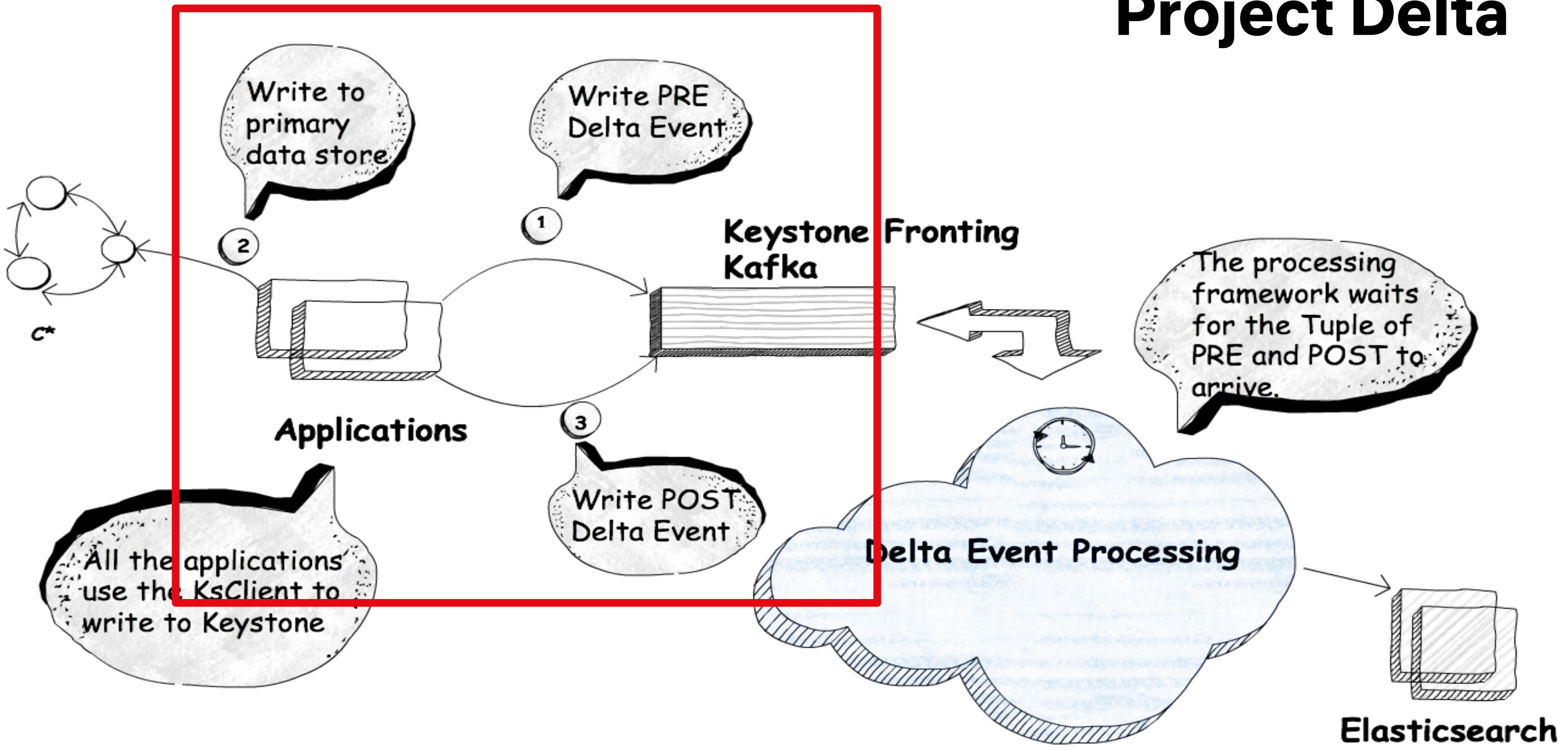


Figure referenced from Streaming Systems by Tyler Akidau el al, Chapter 2 Going Streaming: When and How

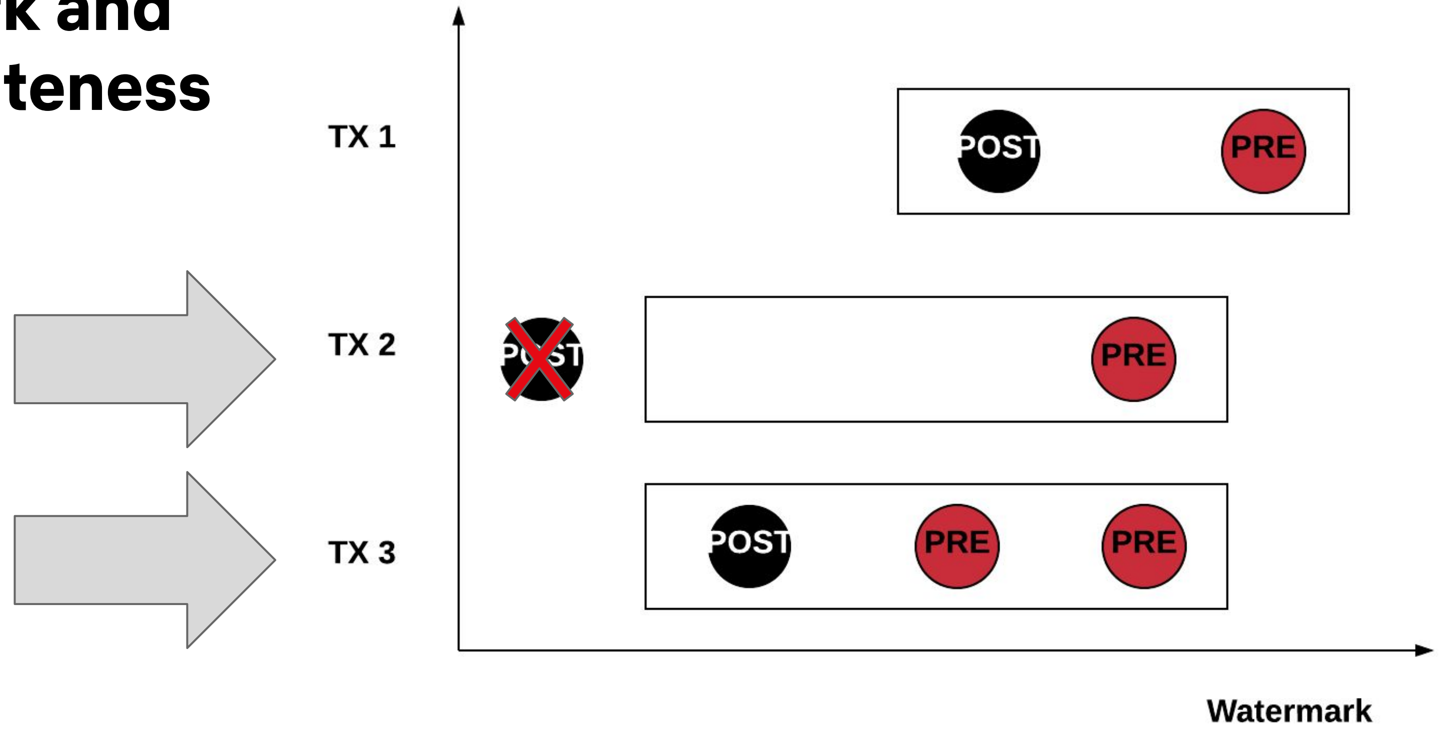
Project Delta



Watermark and allowed lateness

POST arrives outside allowed lateness boundary

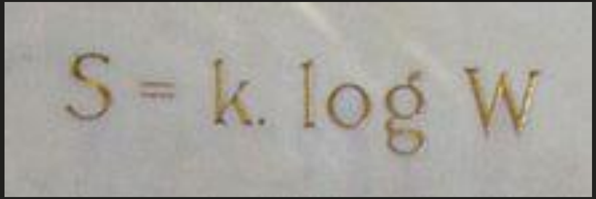
PRE event duplicated

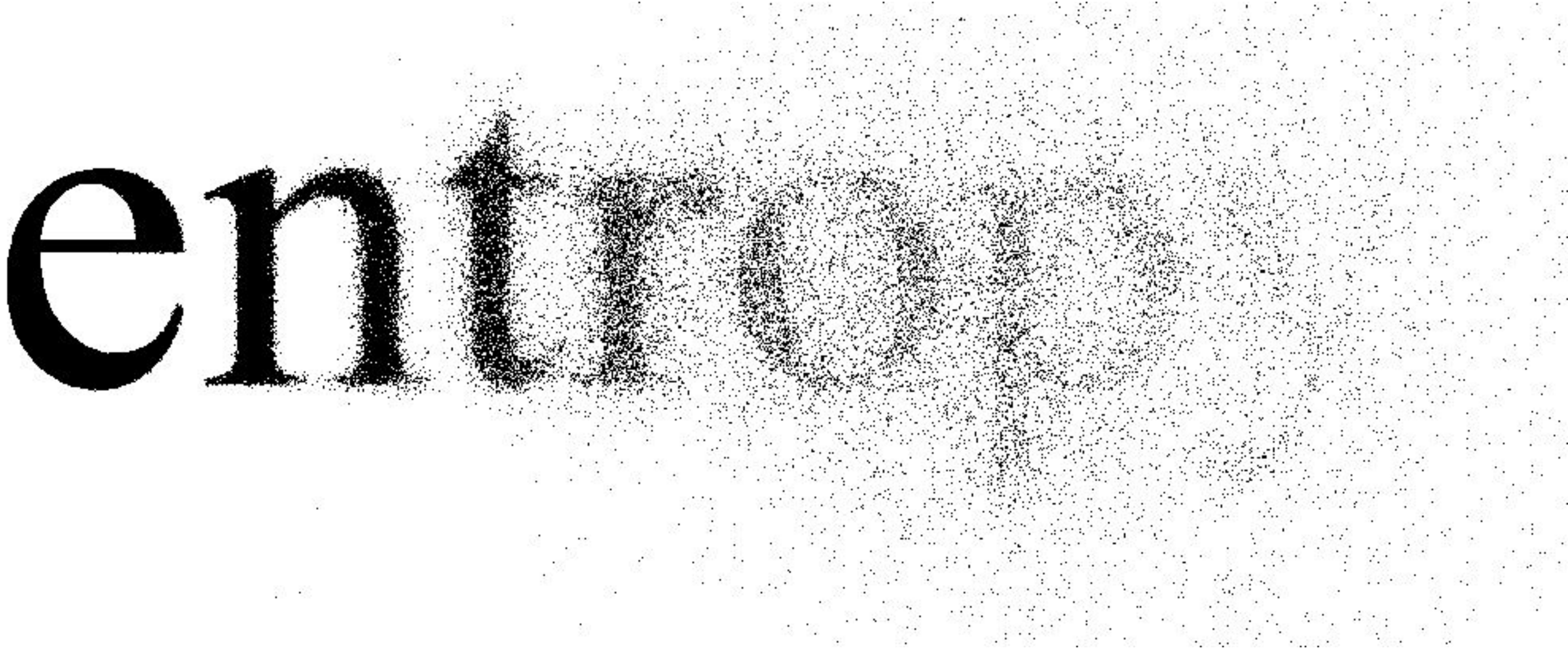


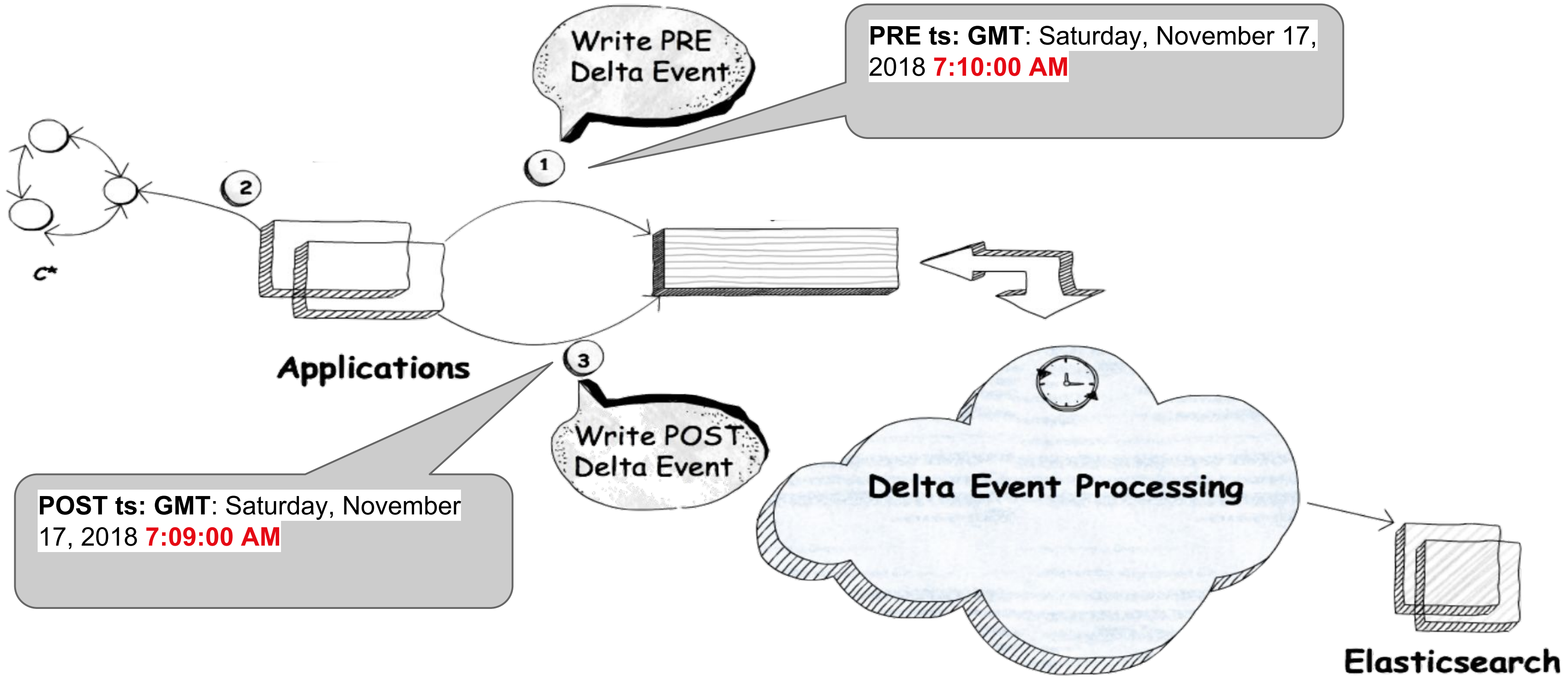
#2

Arrow of time

Boltzmann’s entropy formula

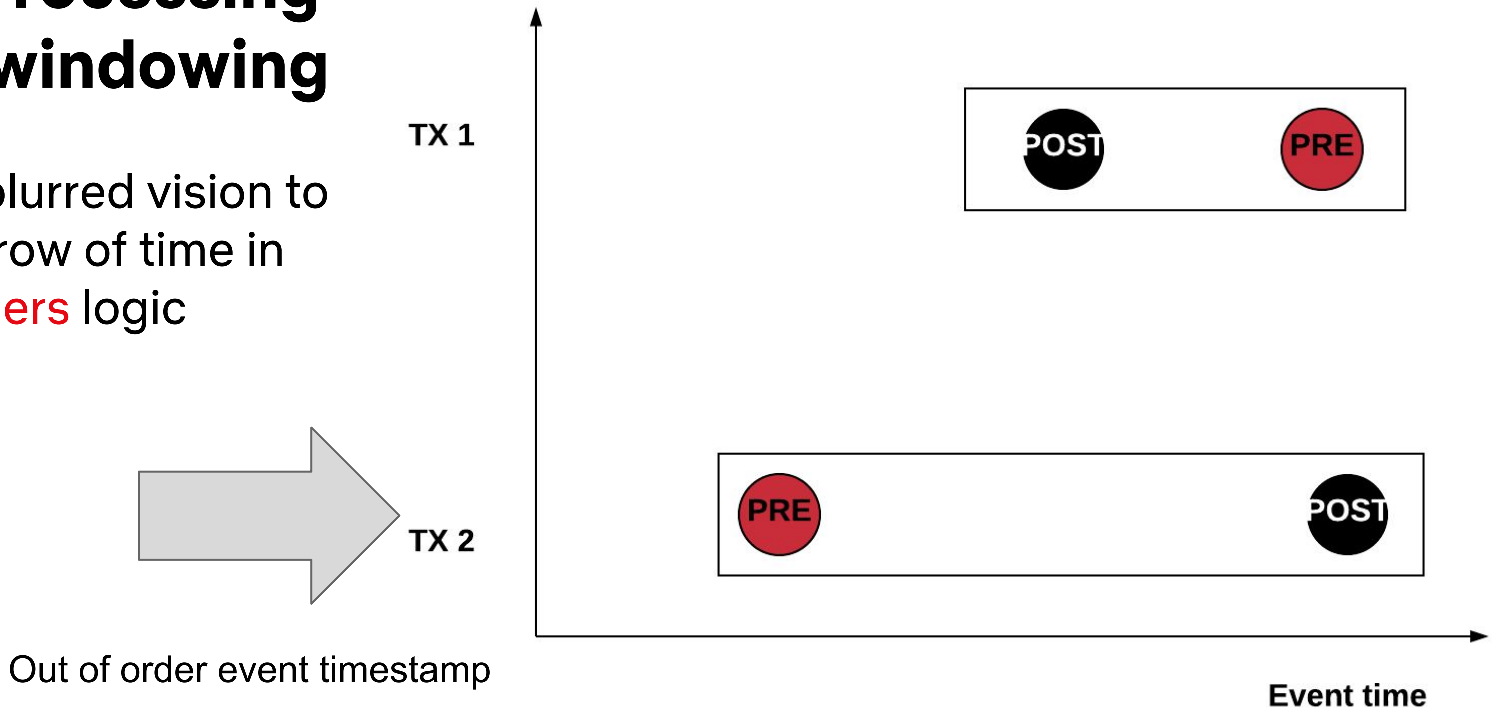



entrop


Write PRE
Delta Event
PRE ts: GMT: Saturday, November 17, 2018 7:10:00 AM
1
2
C*
Applications
3
Write POST
Delta Event
POST ts: GMT: Saturday, November 17, 2018 7:09:00 AM
Delta Event Processing
Elasticsearch

Stream Processing
Custom windowing

Embed our blurred vision to represent arrow of time in custom triggers logic



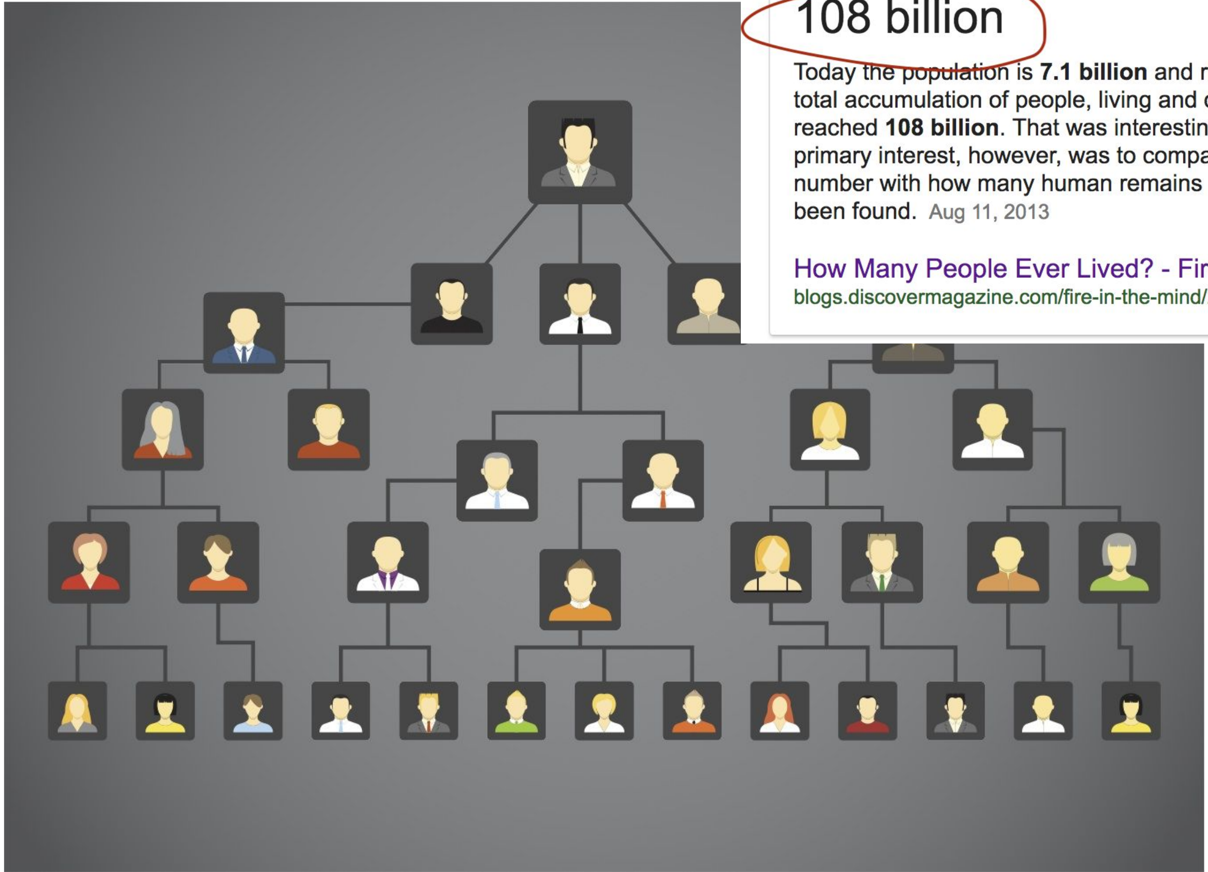
#3
Perception of time

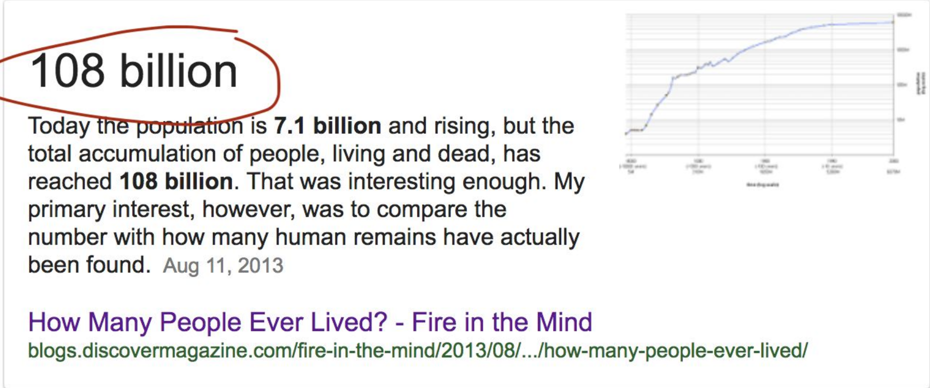
This is a story about me and my uncle ...

Me,
4 years old

My uncle, 2
years old

Imagine an ancestry tree includes all modern human beings ...





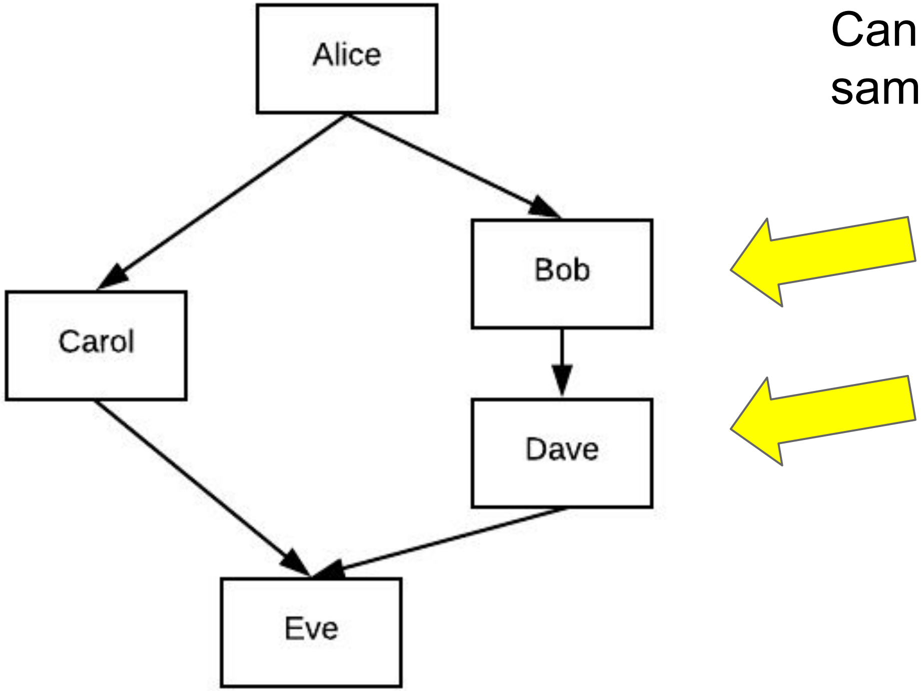
108 billion

Today the population is **7.1 billion** and rising, but the total accumulation of people, living and dead, has reached **108 billion**. That was interesting enough. My primary interest, however, was to compare the number with how many human remains have actually been found. Aug 11, 2013

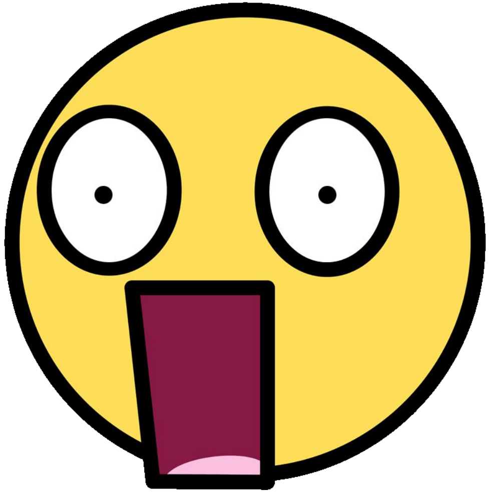
How Many People Ever Lived? - Fire in the Mind

blogs.discovermagazine.com/fire-in-the-mind/2013/08/.../how-many-people-ever-lived/

When forcing a global generation order...

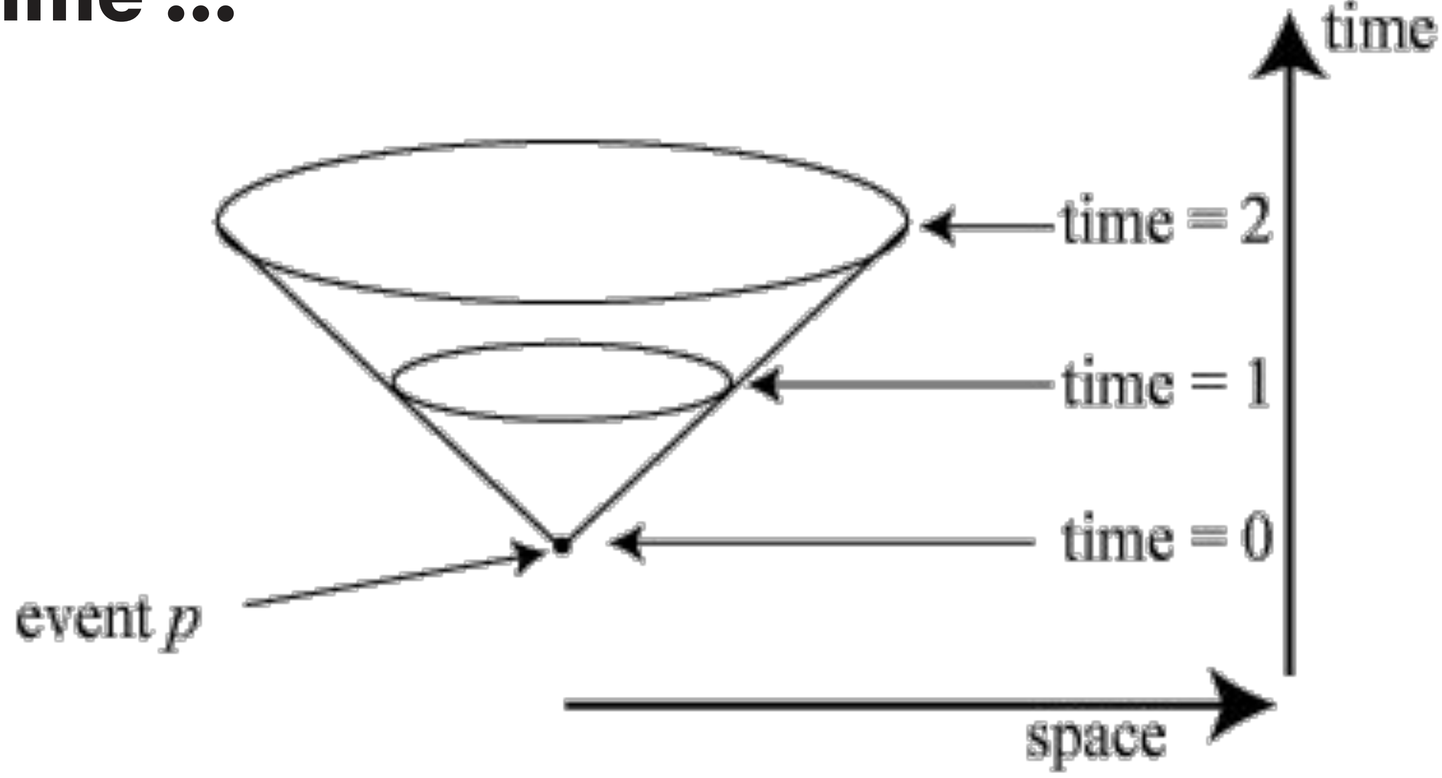


Can Bob and Dave be logically the same generation?

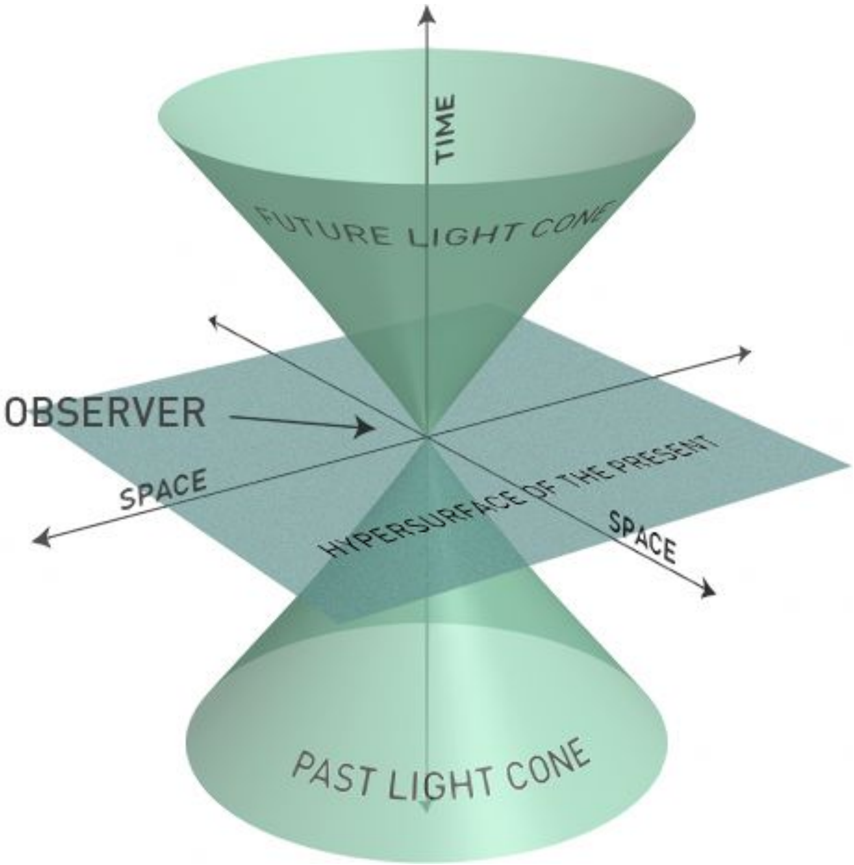


What’s the meaning of “now”?

Light travels in a cone shape over time ...

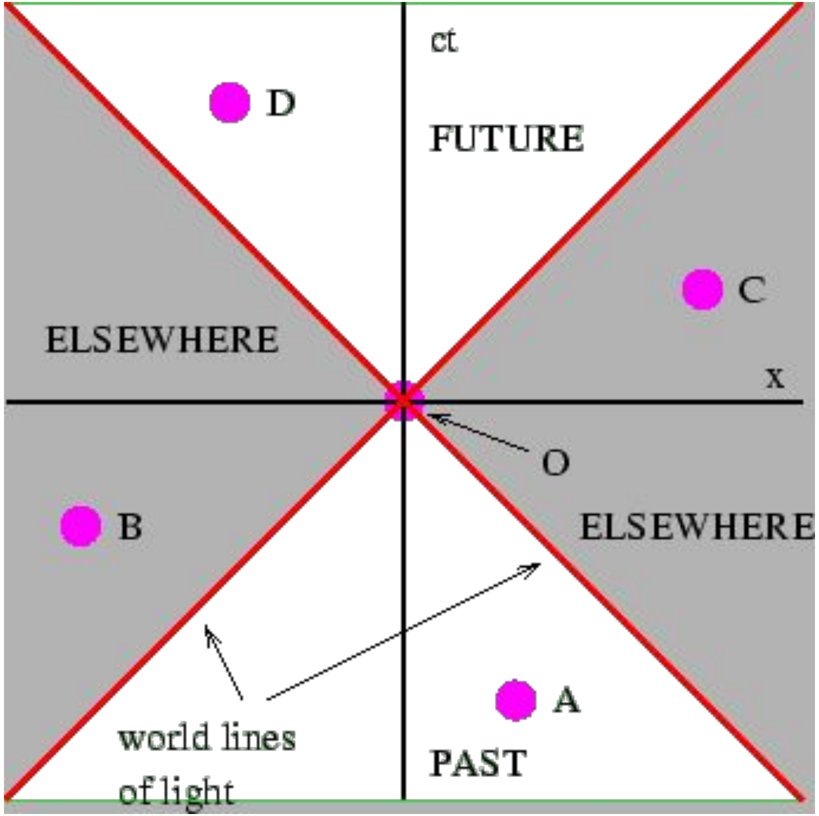


The light cone representing the past, present, and future ...

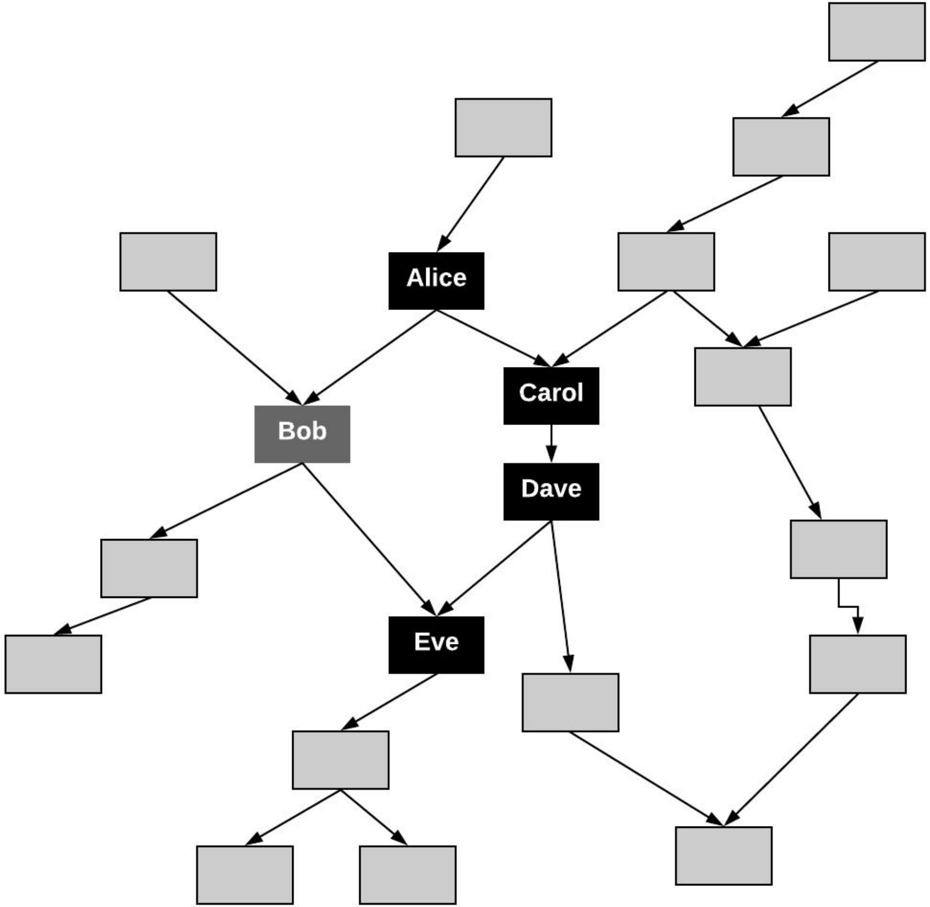


https://en.wikipedia.org/wiki/Light_cone

Light cone spacetime diagram

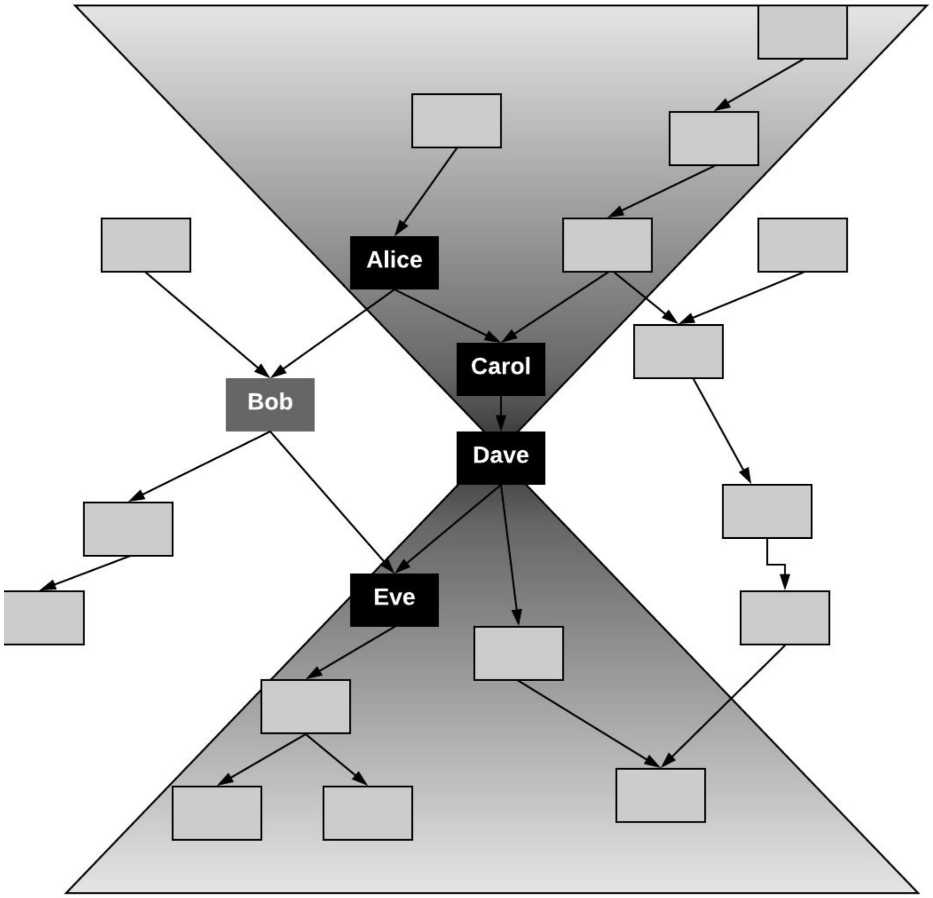


Revisit the ancestry tree



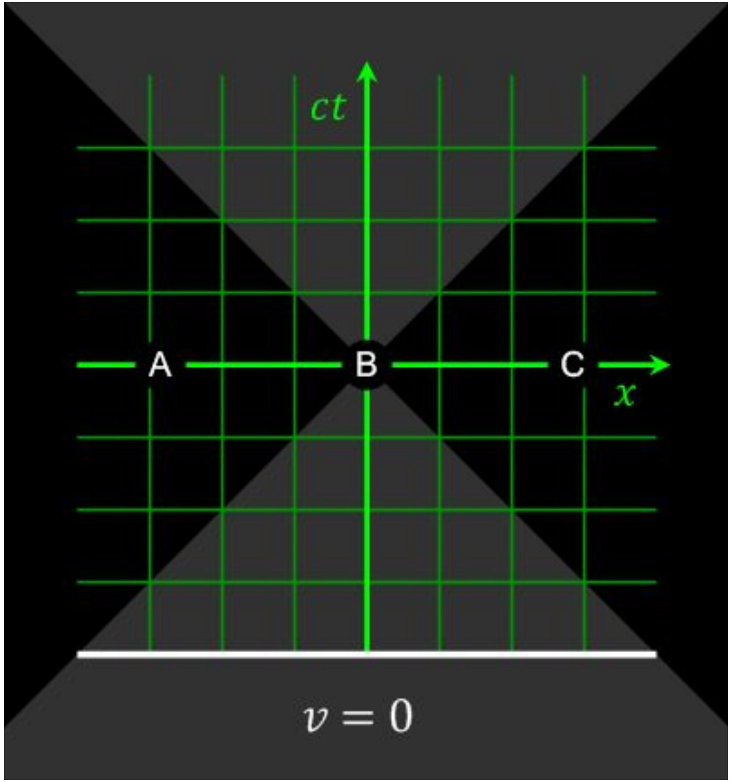
NETFLIX

The cone shape shows the causal/partial ordering from Dave's frame of reference.



Lorentz transformation

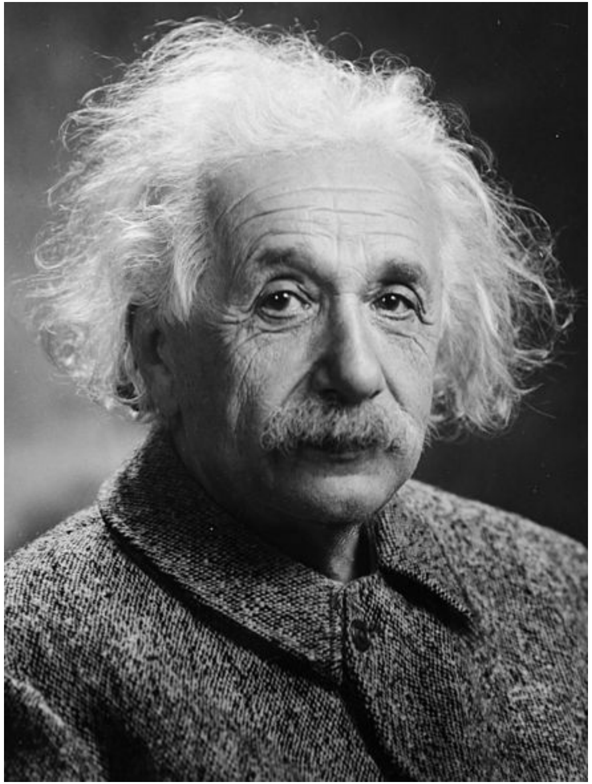
Observers in different frame of references perceive different ordering of events



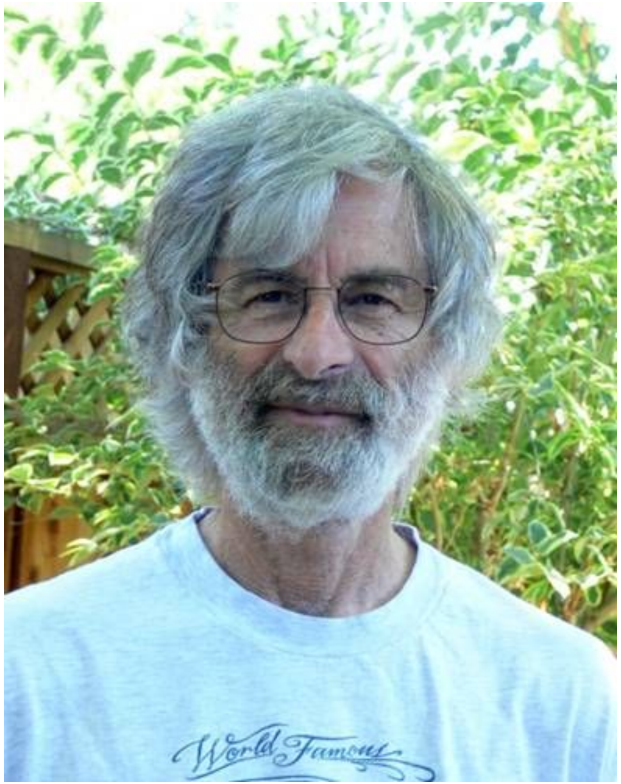
Relativity of **Simultaneity**

Time and Ordering depends on **frame of reference** (space **and** time!)

There is no deterministic global ordering.



NETFLIX



Operating Systems

R. Stockton Gaines Editor

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

Leslie Lamport
Massachusetts Computer Associates, Inc.

The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed algorithm is given for synchronizing a system of logical clocks which can be used to totally order the events. The use of the total ordering is illustrated with a method for solving synchronization problems. The algorithm is then specialized for synchronizing physical clocks, and a bound is derived on how far out of synchrony the clocks can become.

Key Words and Phrases: distributed systems, computer networks, clock synchronization, multiprocess systems

CR Categories: 4.32, 5.29

Introduction

The concept of time is fundamental to our way of thinking. It is derived from the more basic concept of the order in which events occur. We say that something happened at 3:15 if it occurred *after* our clock read 3:15 and *before* it read 3:16. The concept of the temporal ordering of events pervades our thinking about systems. For example, in an airline reservation system we specify that a request for a reservation should be granted if it is made *before* the flight is filled. However, we will see that this concept must be carefully reexamined when considering events in a distributed system.

This work was supported by the Advanced Research Projects Agency of the Department of Defense and Rome Air Development Center. It was monitored by Rome Air Development Center under contract number F 30602-76-C-0094.

Author's address: Computer Science Laboratory, SRI International, 333 Ravenswood Ave., Menlo Park CA 94025.

A distributed system consists of a collection of distinct processes which are spatially separated, and which communicate with one another by exchanging messages. A network of interconnected computers, such as the ARPA net, is a distributed system. A single computer can also be viewed as a distributed system in which the central control unit, the memory units, and the input-output channels are separate processes. A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process.

We will concern ourselves primarily with systems of spatially separated computers. However, many of our remarks will apply more generally. In particular, a multiprocessing system on a single computer involves problems similar to those of a distributed system because of the unpredictable order in which certain events can occur.

In a distributed system, it is sometimes impossible to say that one of two events occurred first. The relation "happened before" is therefore only a partial ordering of the events in the system. We have found that problems often arise because people are not fully aware of this fact and its implications.

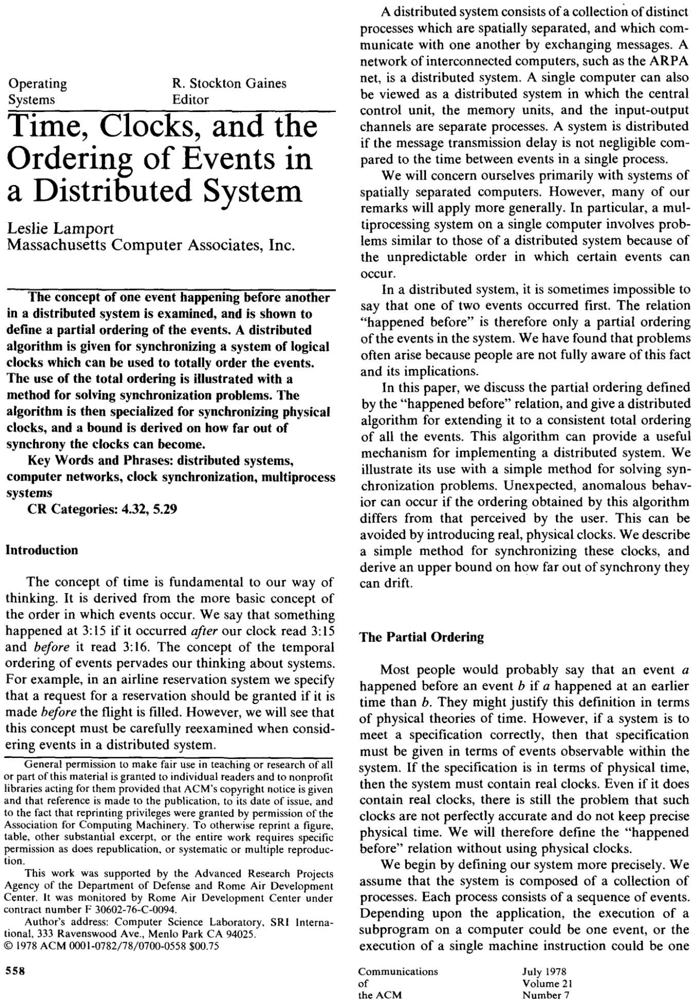
In this paper, we discuss the partial ordering defined by the "happened before" relation, and give a distributed algorithm for extending it to a consistent total ordering of all the events. This algorithm can provide a useful mechanism for implementing a distributed system. We illustrate its use with a simple method for solving synchronization problems. Unexpected, anomalous behavior can occur if the ordering obtained by this algorithm differs from that perceived by the user. This can be avoided by introducing real, physical clocks. We describe a simple method for synchronizing these clocks, and derive an upper bound on how far out of synchrony they can drift.

The Partial Ordering

Most people would probably say that an event a happened before an event b if a happened at an earlier time than b. They might justify this definition in terms of physical theories of time. However, if a system is to meet a specification correctly, then that specification must be given in terms of events observable within the system. If the specification is in terms of physical time, then the system must contain real clocks. Even if it does contain real clocks, there is still the problem that such clocks are not perfectly accurate and do not keep precise physical time. We will therefore define the "happened before" relation without using physical clocks.

We begin by defining our system more precisely. We assume that the system is composed of a collection of processes. Each process consists of a sequence of events. Depending upon the application, the execution of a subprogram on a computer could be one event, or the execution of a single machine instruction could be one

NETFLIX



Operating Systems — R. Stockton Gaines, Editor

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

Leslie Lamport
Massachusetts Computer Associates, Inc.

The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed algorithm is given for synchronizing a system of logical clocks which can be used to totally order the events. The use of the total ordering is illustrated with a method for solving synchronization problems. The algorithm is then specialized for synchronizing physical clocks, and a bound is derived on how far out of synchrony the clocks can become.

Key Words and Phrases: distributed systems, computer networks, clock synchronization, multiprocess systems

CR Categories: 4.32, 5.29

Introduction

The concept of time is fundamental to our way of thinking. It is derived from the more basic concept of the order in which events occur. We say that something happened at 3:15 if it occurred *after* our clock read 3:15 and *before* it read 3:16. The concept of the temporal ordering of events pervades our thinking about systems. For example, in an airline reservation system we specify that a request for a reservation should be granted if it is made *before* the flight is filled. However, we will see that this concept must be carefully reexamined when considering events in a distributed system.

This work was supported by the Advanced Research Projects Agency of the Department of Defense and Rome Air Development Center. It was monitored by Rome Air Development Center under contract number F 30602-76-C-0094.

Author's address: Computer Science Laboratory, SRI International, 333 Ravenswood Ave., Menlo Park CA 94025.

A distributed system consists of a collection of distinct processes which are spatially separated, and which communicate with one another by exchanging messages. A network of interconnected computers, such as the ARPA net, is a distributed system. A single computer can also be viewed as a distributed system in which the central control unit, the memory units, and the input-output channels are separate processes. A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process.

We will concern ourselves primarily with systems of spatially separated computers. However, many of our remarks will apply more generally. In particular, a multiprocessing system on a single computer involves problems similar to those of a distributed system because of the unpredictable order in which certain events can occur.

In a distributed system, it is sometimes impossible to say that one of two events occurred first. The relation "happened before" is therefore only a partial ordering of the events in the system. We have found that problems often arise because people are not fully aware of this fact and its implications.

In this paper, we discuss the partial ordering defined by the "happened before" relation, and give a distributed algorithm for extending it to a consistent total ordering of all the events. This algorithm can provide a useful mechanism for implementing a distributed system. We illustrate its use with a simple method for solving synchronization problems. Unexpected, anomalous behavior can occur if the ordering obtained by this algorithm differs from that perceived by the user. This can be avoided by introducing real, physical clocks. We describe a simple method for synchronizing these clocks, and derive an upper bound on how far out of synchrony they can drift.

The Partial Ordering

Most people would probably say that an event *a* happened before an event *b* if *a* happened at an earlier time than *b*. They might justify this definition in terms of physical theories of time. However, if a system is to meet a specification correctly, then that specification must be given in terms of events observable within the system. If the specification is in terms of physical time, then the system must contain real clocks. Even if it does contain real clocks, there is still the problem that such clocks are not perfectly accurate and do not keep precise physical time. We will therefore define the "happened before" relation without using physical clocks.

We begin by defining our system more precisely. We assume that the system is composed of a collection of processes. Each process consists of a sequence of events. Depending upon the application, the execution of a subprogram on a computer could be one event, or the execution of a single machine instruction could be one

In a distributed system, it is sometimes impossible to say that one of two events occurred first. The relation "happened before" is therefore only a partial ordering of the events in the system.

NETFLIX

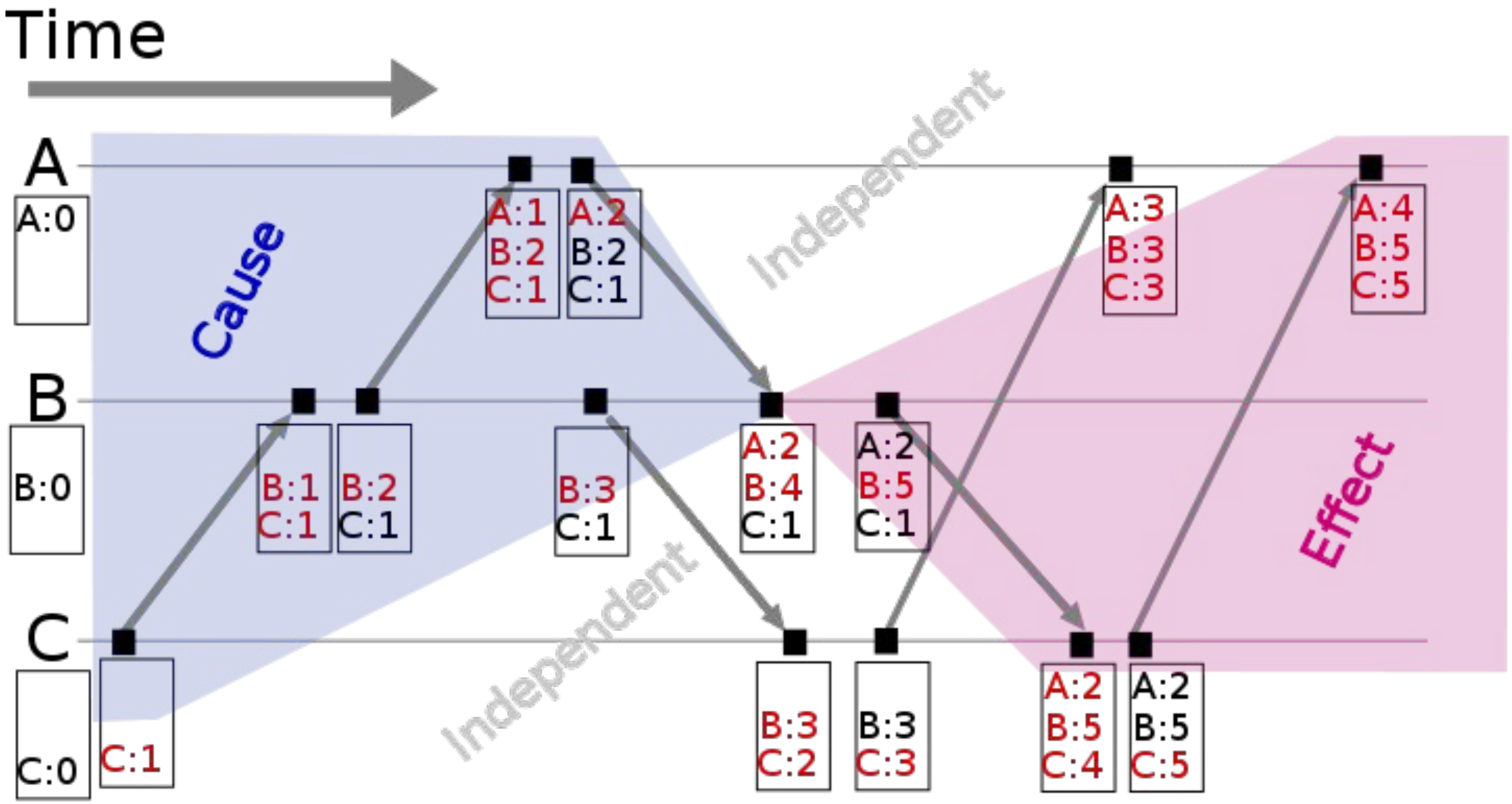


Figure referenced from wikipedia: https://en.wikipedia.org/wiki/Vector_clock

Partial/Causal ordering

An **irreflexive partial ordering** on a set A is a relation on A that satisfies three properties.

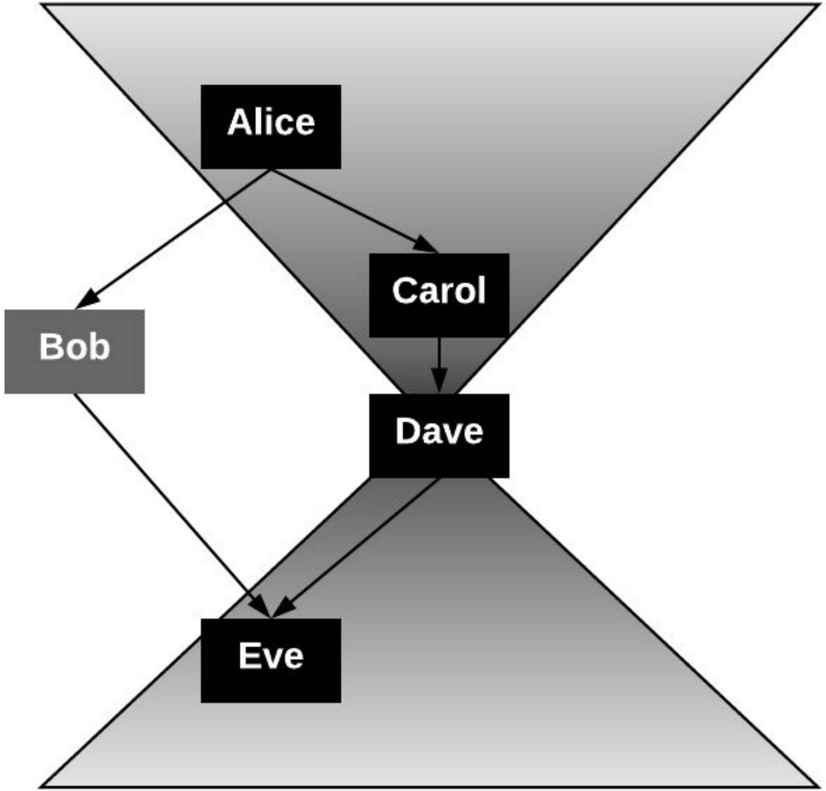
1. **irreflexivity**: a $\not<$ a
2. **antisymmetry**: if a < b then b $\not<$ a
3. **transitivity**: if a < b and b < c then a < c

Total ordering

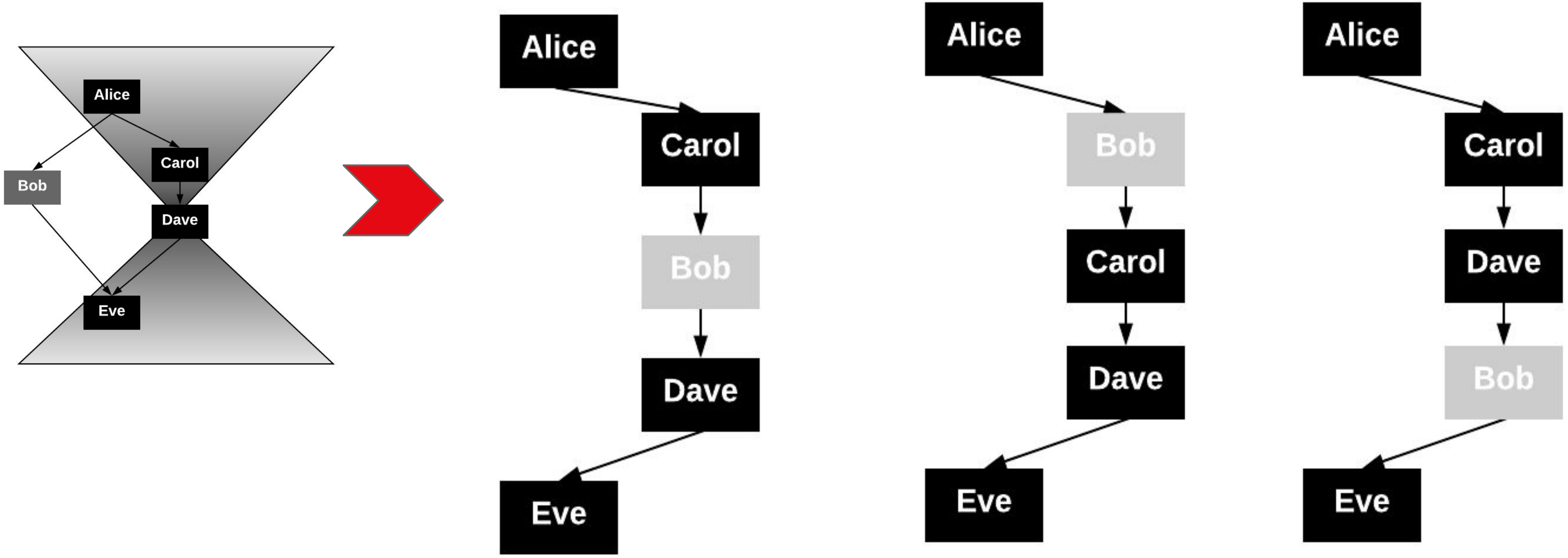
An **irreflexive total ordering** is a irreflexive partial ordering that satisfies another condition.

4. **totality**: if $a \neq b$ then $a < b$ or $b < a$.

Causal/Partial vs Total ordering



Causal/Partial vs Total ordering



Distributed Consensus and **Atomic Broadcast** is the same thing!

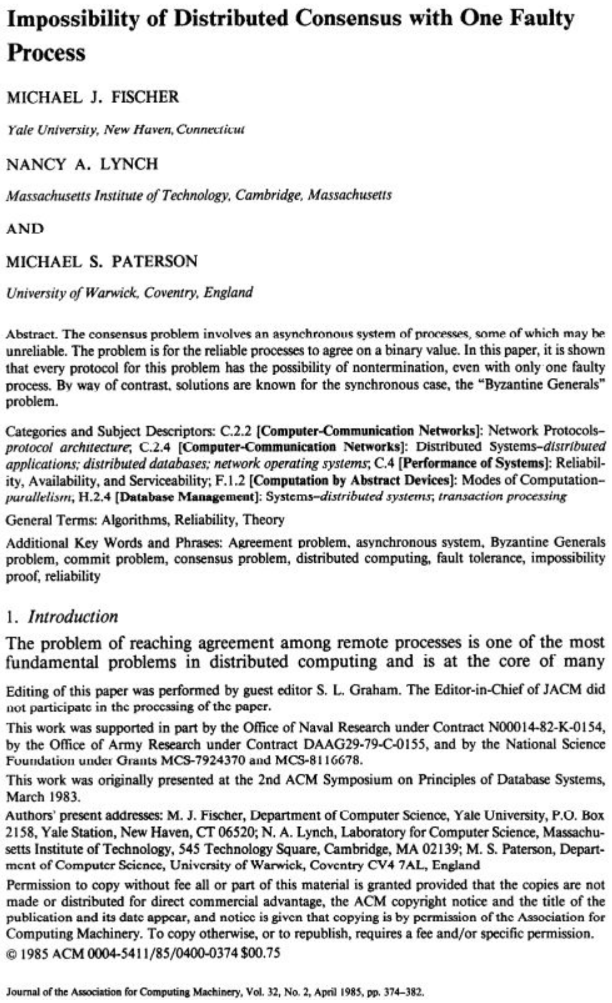
Both requires **total order** broadcast.

Linearizability

… to make a system appear as if there is only a single copy of the data.

Linearizability is the C in CAP theorem.
(practically no CA system, only CP)

Linearizability requires total ordering…



Impossibility of Distributed Consensus with One Faulty Process

MICHAEL J. FISCHER

Yale University, New Haven, Connecticut

NANCY A. LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

MICHAEL S. PATERSON

University of Warwick, Coventry, England

Abstract. The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of nontermination, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols–*protocol architecture*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems–*distributed applications; distributed databases; network operating systems*; C.4 [**Performance of Systems**]: Reliability, Availability, and Serviceability; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation–*parallelism*; H.2.4 [**Database Management**]: Systems–*distributed systems; transaction processing*

General Terms: Algorithms, Reliability, Theory

Additional Key Words and Phrases: Agreement problem, asynchronous system, Byzantine Generals problem, commit problem, consensus problem, distributed computing, fault tolerance, impossibility proof, reliability

1. *Introduction*

The problem of reaching agreement among remote processes is one of the most fundamental problems in distributed computing and is at the core of many

Editing of this paper was performed by guest editor S. L. Graham. The Editor-in-Chief of JACM did not participate in the processing of the paper.

This work was supported in part by the Office of Naval Research under Contract N00014-82-K-0154, by the Office of Army Research under Contract DAAG29-79-C-0155, and by the National Science Foundation under Grants MCS-7924370 and MCS-8116678.

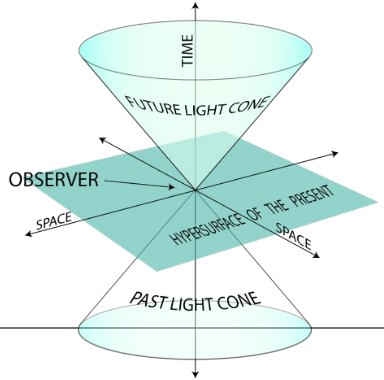
This work was originally presented at the 2nd ACM Symposium on Principles of Database Systems, March 1983.

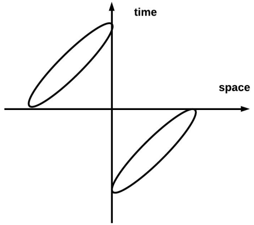
Authors' present addresses: M. J. Fischer, Department of Computer Science, Yale University, P.O. Box 2158, Yale Station, New Haven, CT 06520; N. A. Lynch, Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139; M. S. Paterson, Department of Computer Science, University of Warwick, Coventry CV4 7AL, England

Journal of the Association for Computing Machinery, Vol. 32, No. 2, April 1985, pp. 374–382.

Consensus in a **synchronous** environment can be resilient to faults.

FLP result shows that in an **async** setting, where only one processor might crash, there is **no** distributed algorithm that solves the consensus problem.

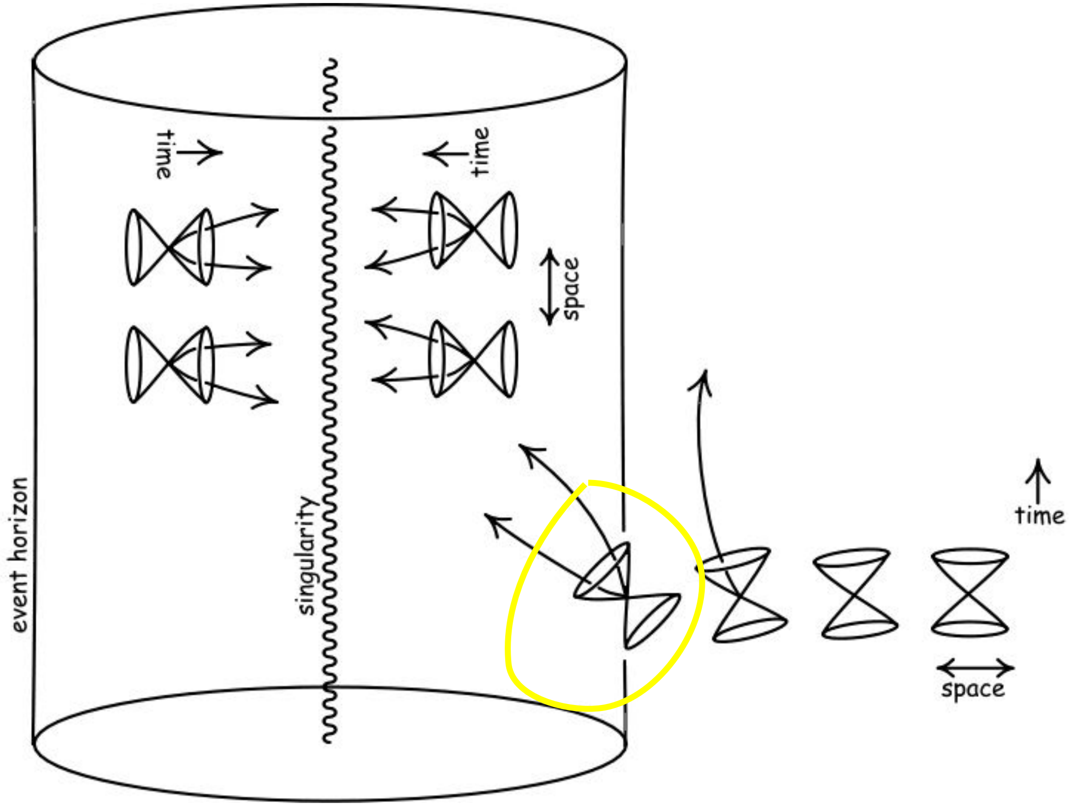


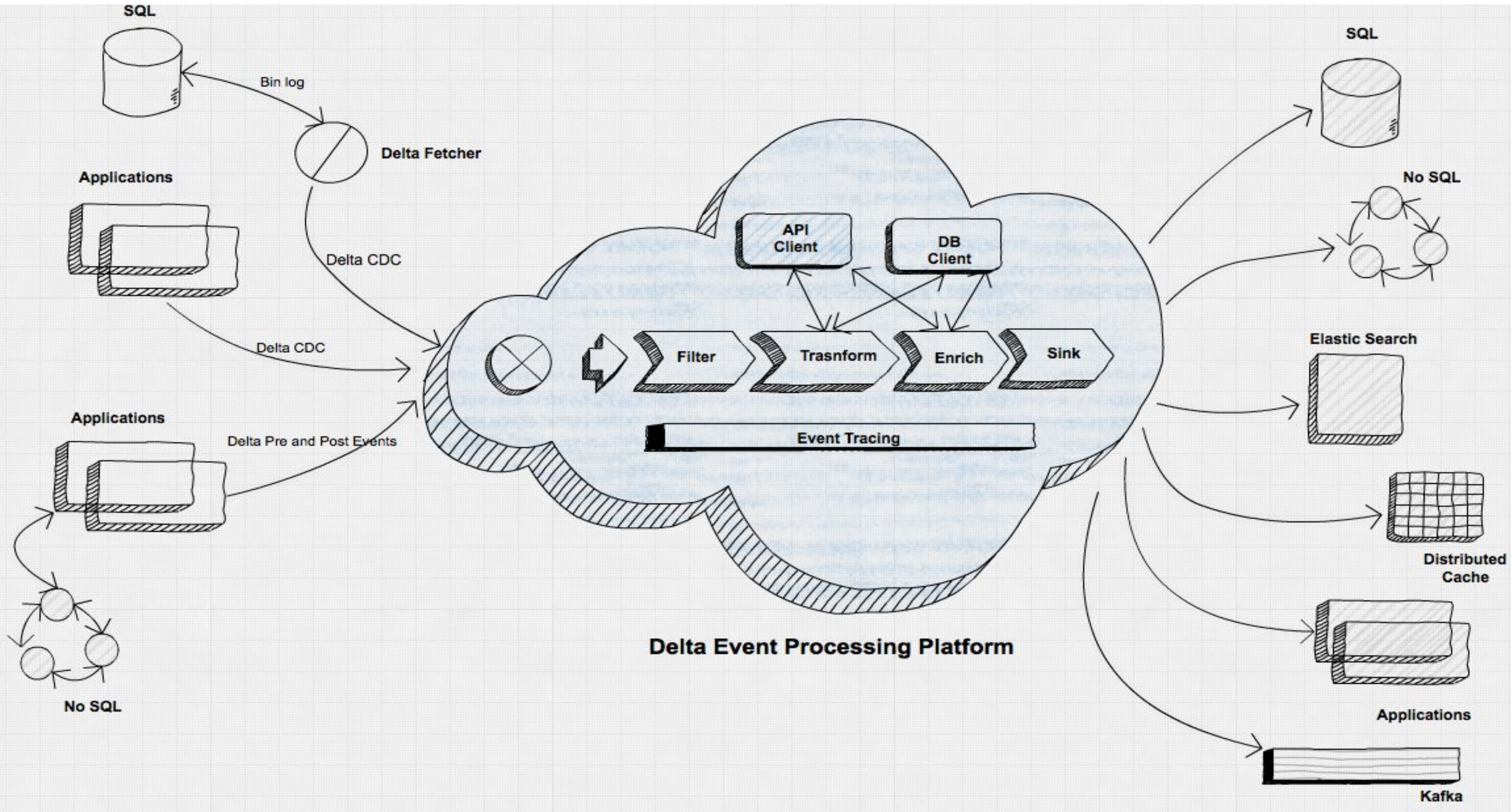


NETFLIX

What happens when an event get close to a black hole’s event horizon

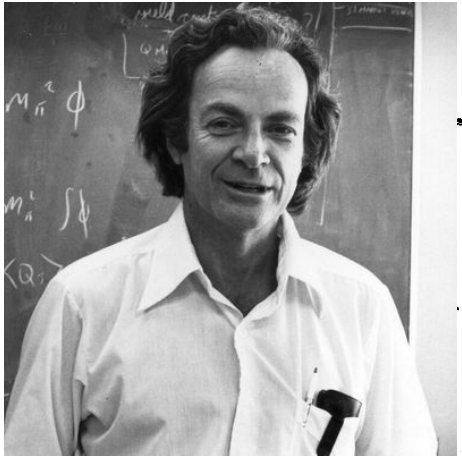
This is very similar to how process fails in distributed systems, observer will never be able to tell whether the process crashed or simply will take long time to respond

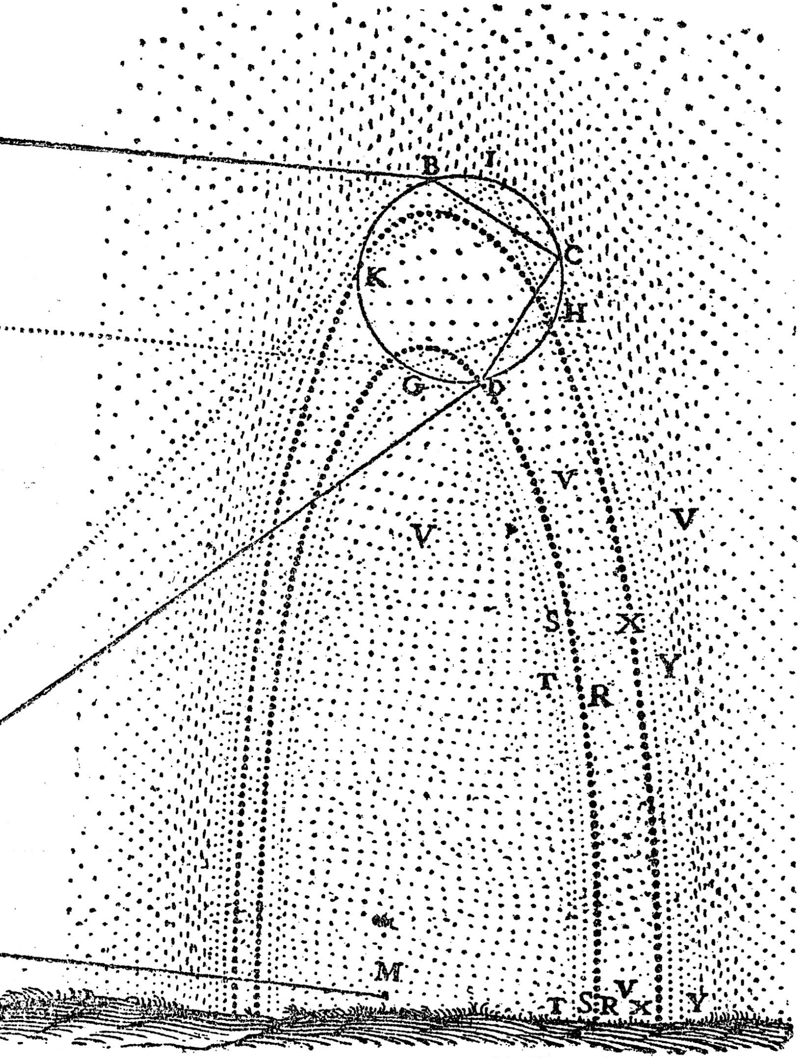


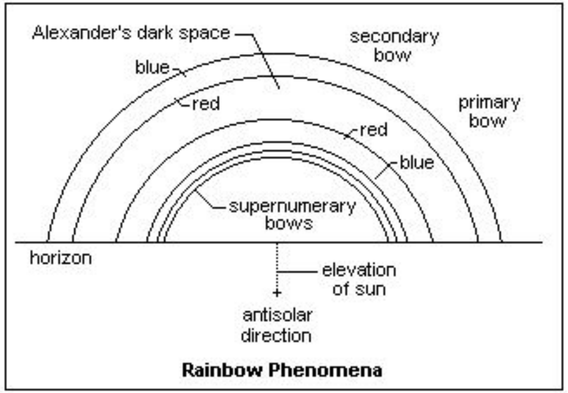

SQL
Bin log
Delta Fetcher
Applications
Delta CDC
Delta CDC
Applications
Delta Pre and Post Events
No SQL
API Client
DB Client
Filter
Trasnform
Enrich
Sink
Event Tracing
Delta Event Processing Platform
SQL
No SQL
Elastic Search
Distributed Cache
Applications
Kafka

The 3 lens of time

- No uniformity of time
- Blurred direction of time
- Limited perception of time







Rainbow Phenomena

Thank you.

NETFLIX

goto; copenhagen

GOTO Copenhagen 2018
Conference Nov. 19 - 21

Click 'Rate Session' to rate session and ask questions.