Time, State and Coordination in Distributed Systems

Prof. Nalini Venkatasubramanian
Distributed Systems Middleware

-includes slides/examples from
Indy Gupta (UIUC) and Kshemkalyani&Singhal (book slides)

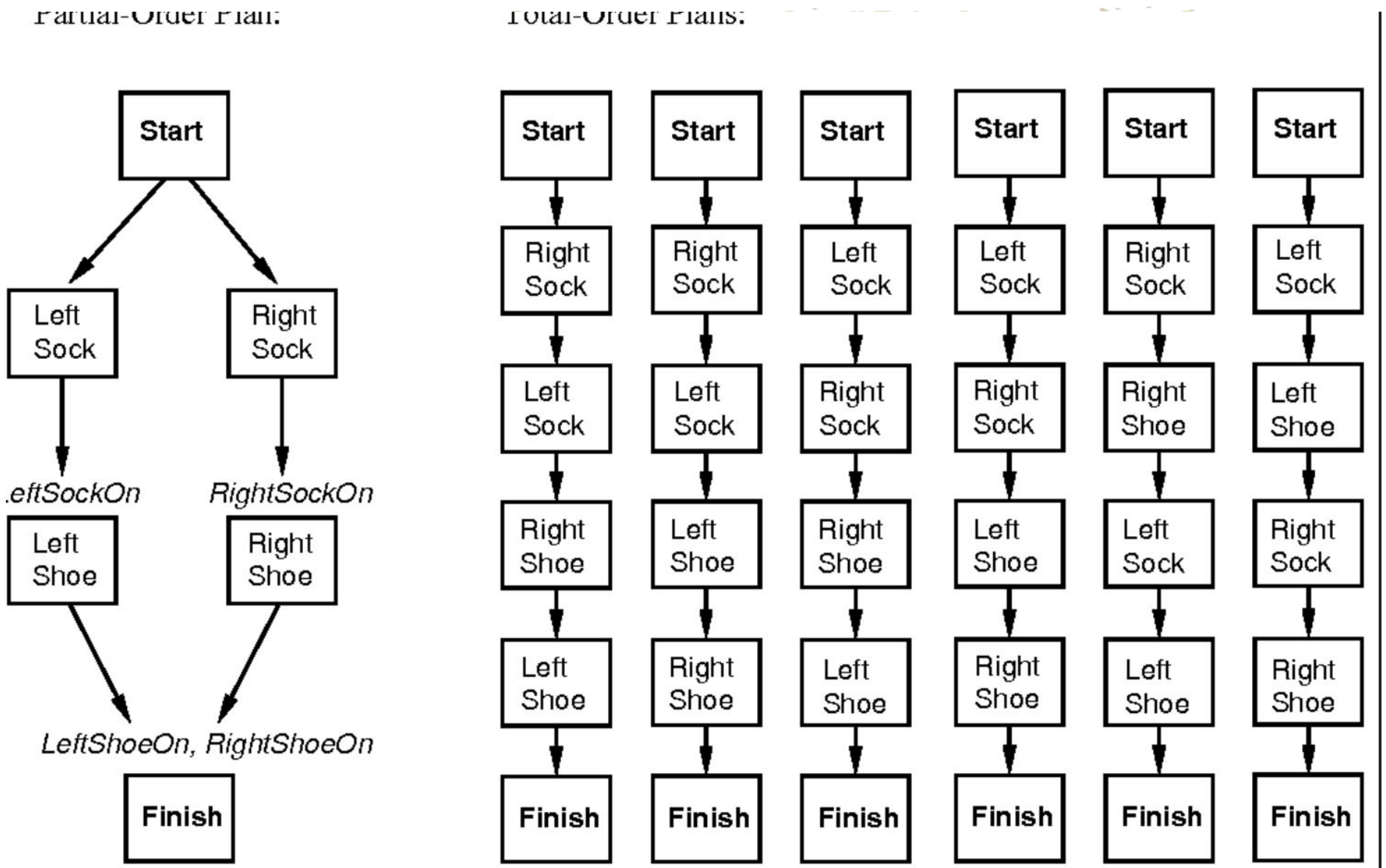
Global Time & Global States of Distributed Systems

- Asynchronous distributed systems consist of several *processes* without common memory which communicate (solely) via *messages* with unpredictable transmission delays
- Global time & global state are hard to realize in distributed systems
  - Rate of event occurrence is very high
  - Event execution times are very small
- We can only *approximate* the global view
  - *Simulate synchronous* distributed system on a given asynchronous system
  - *Simulate* a *global time* – Clocks (Physical and Logical)
  - *Simulate* a *global state* – Global Snapshots

The Concept of Time in Distributed Systems

- A standard time is a set of instants with a temporal precedence order $<$ satisfying certain conditions [Van Benthem 83]:
    - Irreflexivity
    - Transitivity
    - Linearity
    - Eternity ($\forall x \exists y: x<y$)
    - Density ($\forall x,y: x<y \rightarrow \exists z: x<z<y$)
  - Transitivity and Irreflexivity imply asymmetry
- A linearly ordered structure of time is not always adequate for distributed systems
  - Captures dependence, not independence of distributed activities
- Time as a partial order
  - A ***partially ordered system of vectors*** forming a *lattice* structure is a natural representation of time in a distributed system.

Partial vs. total order



Partial-Order Plan:
Total-Order Plans:
Start
Left Sock
Right Sock
LeftSockOn
RightSockOn
Left Shoe
Right Shoe
LeftShoeOn, RightShoeOn
Finish
Start → Right Sock → Left Sock → Right Shoe → Left Shoe → Finish
Start → Right Sock → Left Sock → Left Shoe → Right Shoe → Finish
Start → Left Sock → Right Sock → Right Shoe → Left Shoe → Finish
Start → Left Sock → Right Sock → Left Shoe → Right Shoe → Finish
Start → Right Sock → Right Shoe → Left Sock → Left Shoe → Finish
Start → Left Sock → Left Shoe → Right Sock → Right Shoe → Finish

Figure 11.6 A partial-order plan for putting on shoes and socks, and the six corresponding linearizations into total-order plans.

Global time in distributed systems

- An accurate notion of global time is difficult to achieve in distributed systems.
  - Uniform notion of time is necessary for correct operation of many applications (mission critical distributed control, online games/entertainment, financial apps, smart environments etc.)
- Clocks in a distributed system drift
  - Relative to each other
  - Relative to a real world clock
    - Determination of this real world clock itself may be an issue
- Clock synchronization is needed to simulate global time
  - Physical Clocks vs. Logical clocks
    - Physical clocks are logical clocks that must not deviate from the real-time by more than a certain amount.

We often derive causality of events from loosely synchronized clocks

Physical Clock Synchronization

Physical Clocks

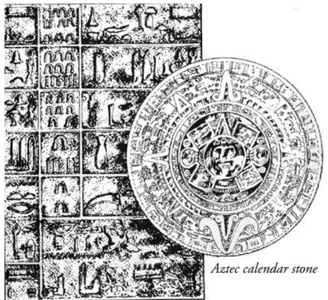
How do we measure real time?

Early – Stonehenge, sundials

13th -17th century: Mechanical clocks based on astronomical measurements

- Solar Day - Transit of the sun
- Solar Seconds - Solar Day/(3600*24)

Early water clock


Aztec calendar stone

Problem (1940): Rotation of earth varies!

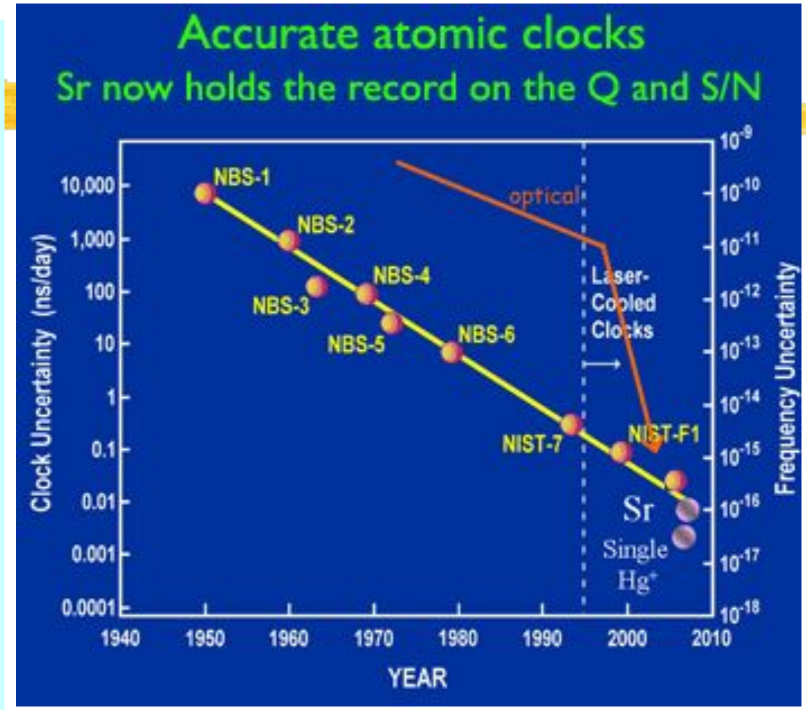
Mean solar second = average over many days

Date	Duration in mean solar time
February 11	24 hours
March 26	24 hours – 18.1 sec
May 14	24 hours
June 19	24 hours + 13.1 sec
July 26	24 hours
September 16	24 hours – 21.3 sec
November 3	24 hours
December 22	24 hours + 29.9 sec

Length of apparent solar day (1998)
– (*cf: wikipedia*)

Atomic Clocks

- 1948 - Counting transitions of a crystal (Cesium 133, quartz) used as atomic clock
  - crystal oscillates at a well known frequency
- 2014 – NIST-F2 Atomic clock
  - Accuracy: ± 1 sec in 300 mil years
  - NIST-F2 measures particular transitions in Cesium atom (9,192,631,770 vibrations per second), in much colder environment, minus 316F, than NIST-F1
- TAI - International Atomic Time
  - 9,192,631,779 transitions = 1 mean solar second in 1948

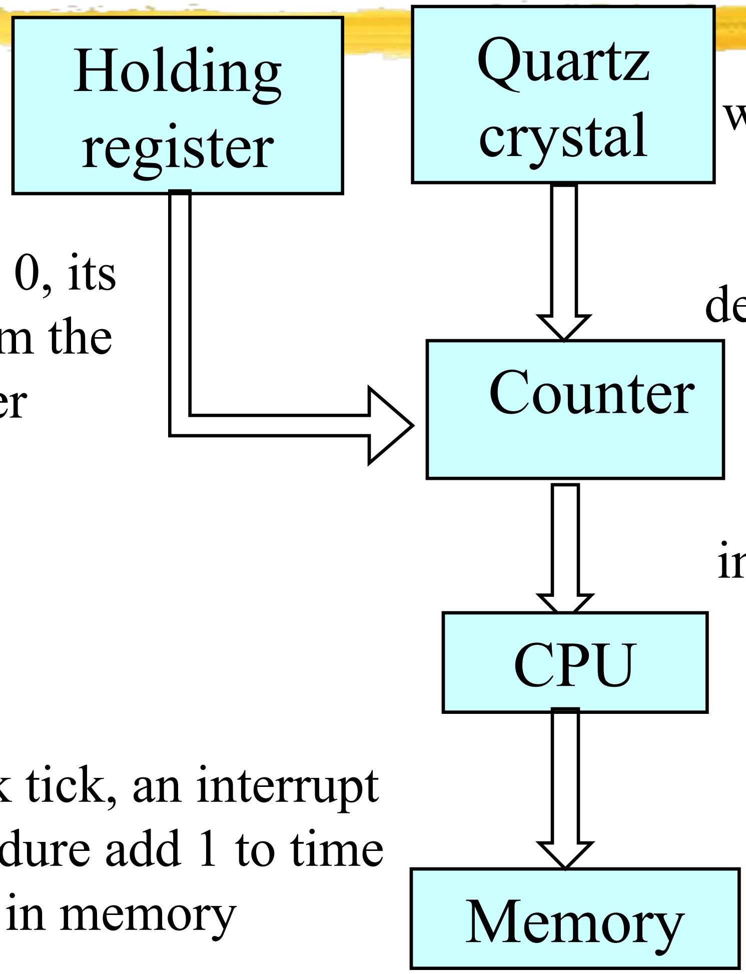


UTC (Universal Coordinated Time)
From time to time, UTC skips a solar second to stay in phase with the sun (30+ times since 1958)

UTC is broadcast by several sources (satellites…)

Next Generation Atomic Clocks -- NIST F2

How Clocks Work in Computers



Oscillation at a well-defined frequency

Each crystal oscillation decrements the counter by 1

When counter gets 0, its value reloaded from the holding register

When counter is 0, an interrupt is generated, which is call a **clock tick**

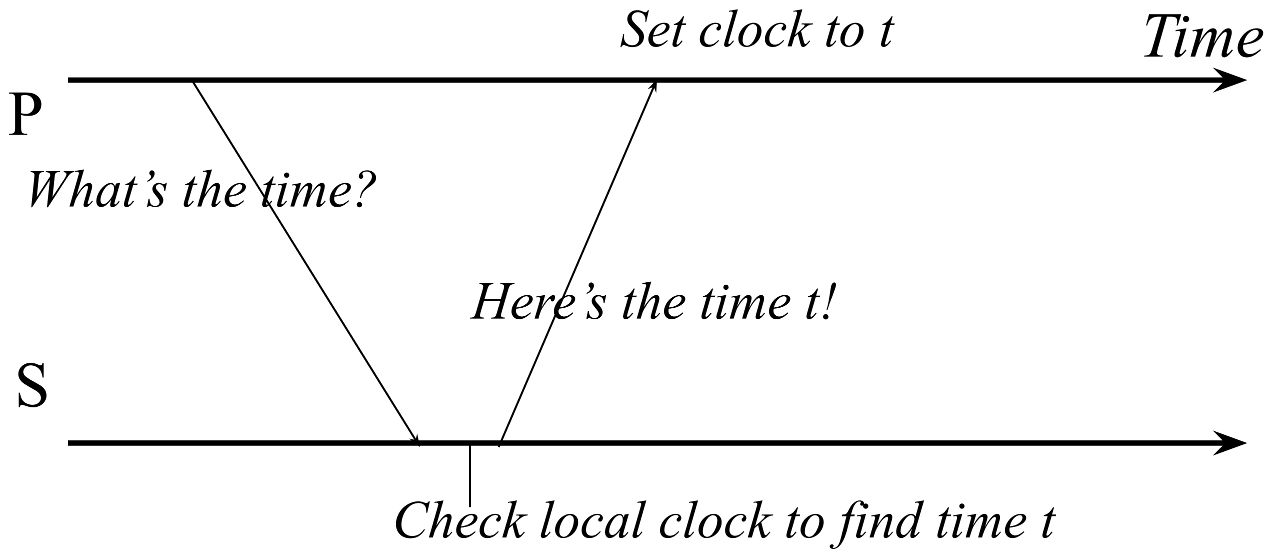
At each clock tick, an interrupt service procedure add 1 to time stored in memory

Accuracy of Computer Clocks

- Modern timer chips have a relative error of 1/100,000 - 0.86 seconds a day
- To maintain synchronized clocks
  - Can use UTC source (time server) to obtain current notion of time
  - Use solutions without UTC.

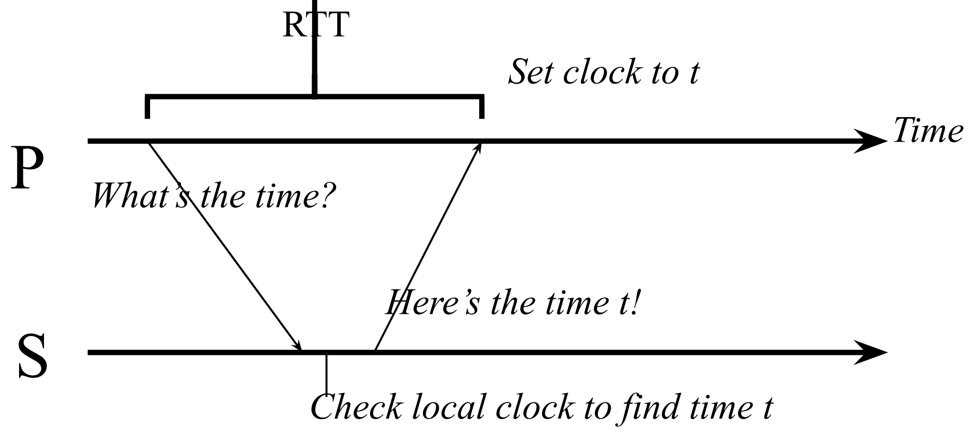
Cristian's (Time Server) Algorithm (external synchronization)

- Uses a *time server* (S) to synchronize clocks
  - Time server keeps the reference time (say UTC)
  - A client asks the time server for time, the server responds with its current time, and the client uses the received value to set its clock.



Cristian's Algorithm (cont.)

- But network round-trip time introduces errors…
  - By the time response message is received at P, time has moved on
  - Let **RTT = response-received-time – request-sent-time** (measurable at client),
  - If we know (a) min = minimum client-server one-way transmission time and (b) that the server timestamped the message at the last possible instant before sending it back
  - Then, the actual time could be between **[T+min,T+RTT— min]**



Cristian's Algorithm (cont.)

- Client sets its clock to halfway between T+min and T+RTT— min i.e., at T+RTT/2
  - ☹ Expected (i.e., average) skew in client clock time = (RTT/2 – *min*)
- Can increase clock value, should never decrease it.
- Can adjust speed of clock too (either up or down is ok)
- Multiple requests to increase accuracy
  - For unusually long RTTs, repeat the time request
  - For non-uniform RTTs
    - Drop values beyond threshold; Use averages (or weighted average)

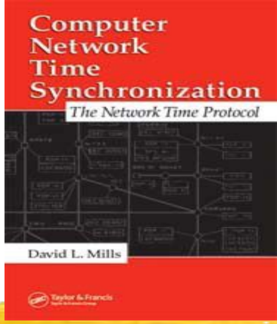
Berkeley UNIX algorithm

- One Version
  - One daemon without UTC
  - Periodically, this daemon polls and asks all the machines for their time
  - The machines respond.
  - The daemon computes an average time and then broadcasts this average time.
- Another Version
  - Master/daemon uses Cristian's algorithm to calculate time from multiple sources, removes outliers, computes average and broadcasts

Decentralized Averaging Algorithm

- Each machine has a daemon without UTC
- Periodically, at fixed agreed-upon times, each machine broadcasts its local time.
- Each of them calculates the average time by averaging all the received local times.

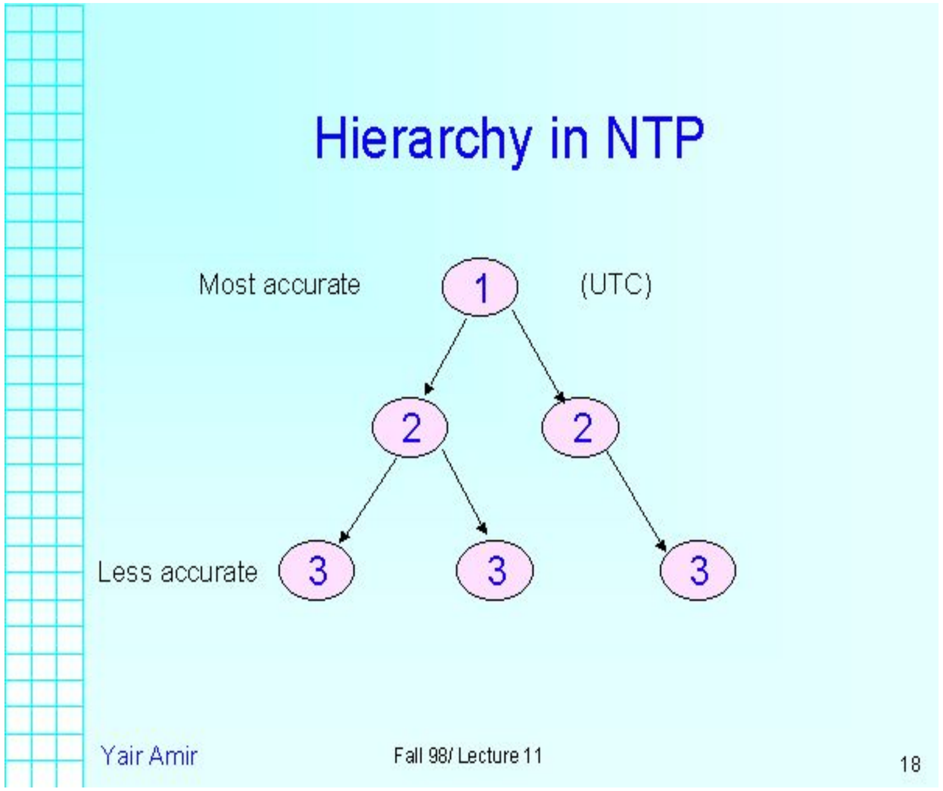
Network Time Protocol (NTP)


Computer Network Time Synchronization
The Network Time Protocol
David L. Mills
Taylor & Francis

- Most widely used physical clock synchronization protocol on the Internet (http://www.ntp.org)
  - Currently used: NTP V3 and V4
- 10-20 million NTP servers and clients in the Internet
- Claimed Accuracy (Varies)
  - milliseconds on WANs, submilliseconds on LANs, submicroseconds using a precision timesource
  - Nanosecond NTP in progress

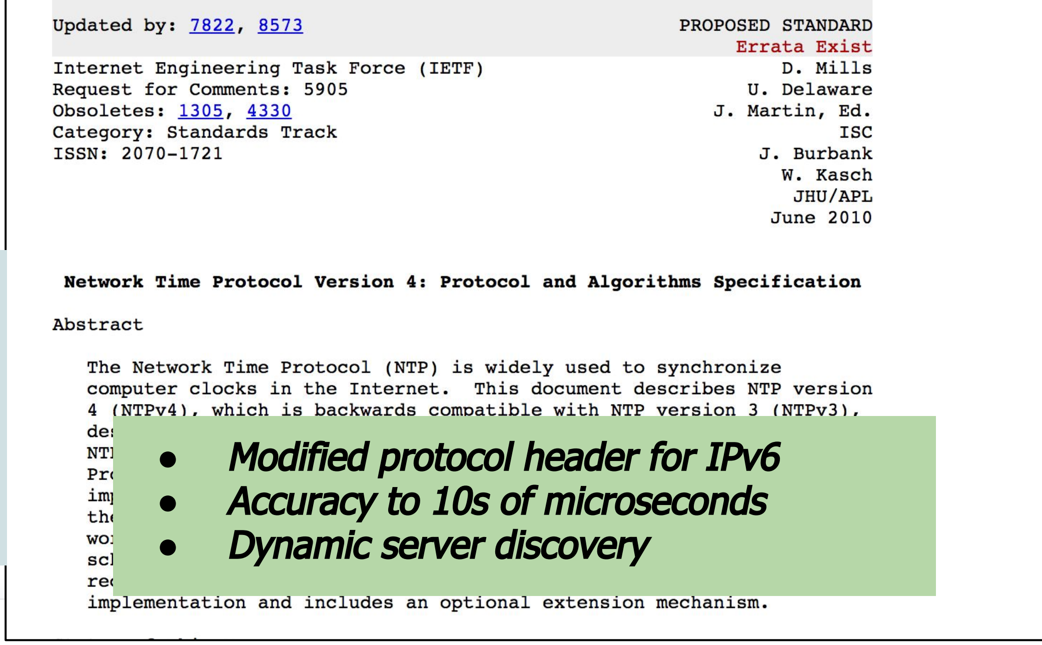
NTP Design

- Hierarchical tree of time servers.
  - The primary server at the root synchronizes with the UTC.
  - The next level contains secondary servers, which act as a backup to the primary server.
  - At the lowest level is the synchronization subnet which has the clients.
  - Variant of Cristian's algorithm that does not use RTT's, but multiple 1-way messages



NTP 4.0

- NTP Stratum 1 Servers
- NTP Stratum 2 Servers
- NTP Pool Servers



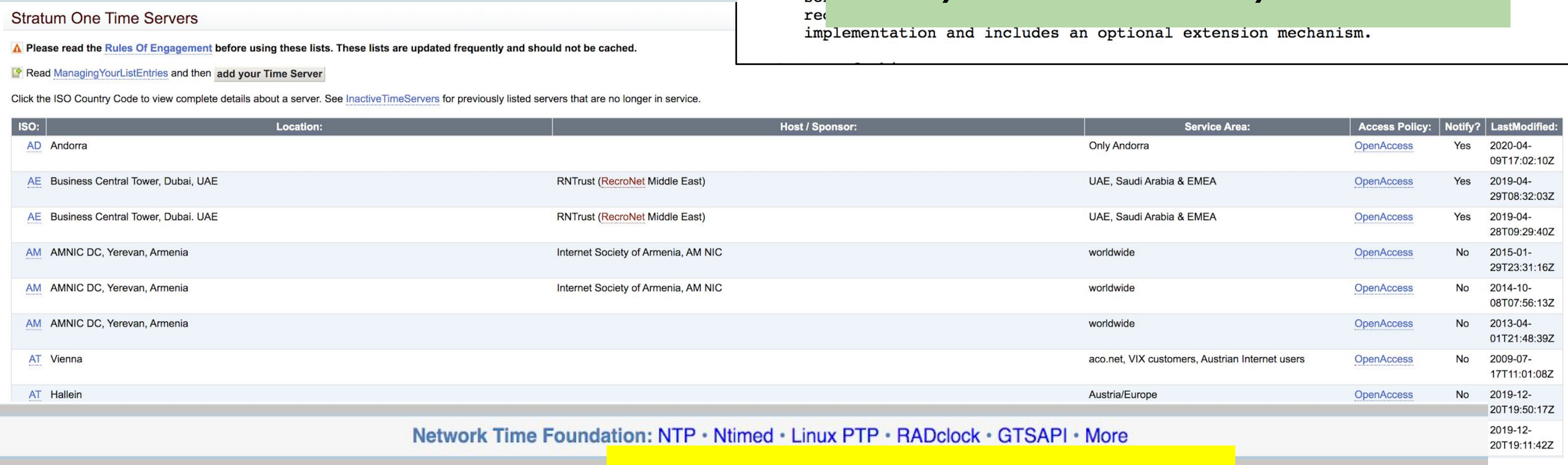
```
Updated by: 7822, 8573                                 PROPOSED STANDARD
                                                           Errata Exist
Internet Engineering Task Force (IETF)                         D. Mills
Request for Comments: 5905                                 U. Delaware
Obsoletes: 1305, 4330                                  J. Martin, Ed.
Category: Standards Track                                          ISC
ISSN: 2070-1721                                            J. Burbank
                                                             W. Kasch
                                                              JHU/APL
                                                            June 2010

 Network Time Protocol Version 4: Protocol and Algorithms Specification

Abstract

   The Network Time Protocol (NTP) is widely used to synchronize
   computer clocks in the Internet.  This document describes NTP version
   4 (NTPv4), which is backwards compatible with NTP version 3 (NTPv3),
   de
   NT
   Pr
   im
   the
   wo
   sc
   re
   implementation and includes an optional extension mechanism.
```

- *Modified protocol header for IPv6*
- *Accuracy to 10s of microseconds*
- *Dynamic server discovery*



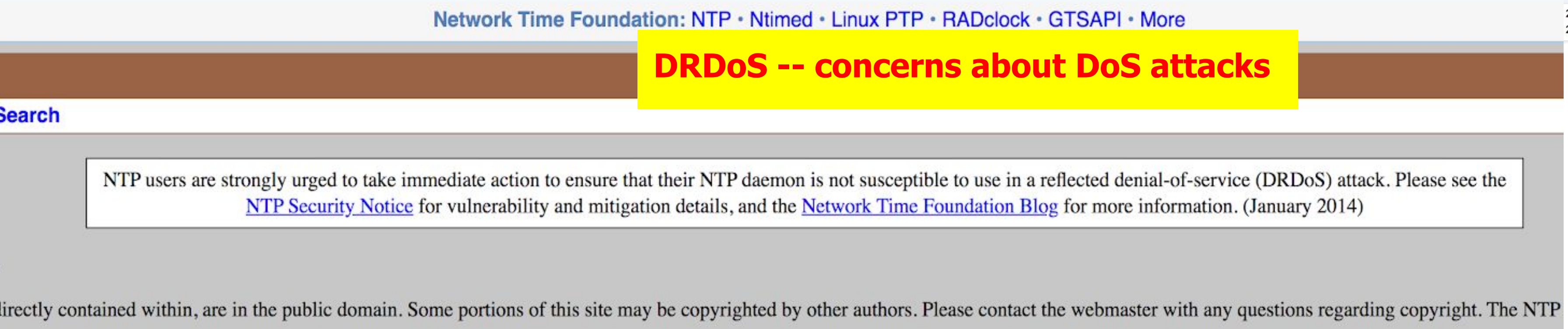
Stratum One Time Servers

⚠ **Please read the Rules Of Engagement before using these lists. These lists are updated frequently and should not be cached.**

Read ManagingYourListEntries and then **add your Time Server**

Click the ISO Country Code to view complete details about a server. See InactiveTimeServers for previously listed servers that are no longer in service.

ISO:	Location:	Host / Sponsor:	Service Area:	Access Policy:	Notify?	LastModified:
AD	Andorra		Only Andorra	OpenAccess	Yes	2020-04-09T17:02:10Z
AE	Business Central Tower, Dubai, UAE	RNTrust (RecroNet Middle East)	UAE, Saudi Arabia & EMEA	OpenAccess	Yes	2019-04-29T08:32:03Z
AE	Business Central Tower, Dubai. UAE	RNTrust (RecroNet Middle East)	UAE, Saudi Arabia & EMEA	OpenAccess	Yes	2019-04-28T09:29:40Z
AM	AMNIC DC, Yerevan, Armenia	Internet Society of Armenia, AM NIC	worldwide	OpenAccess	No	2015-01-29T23:31:16Z
AM	AMNIC DC, Yerevan, Armenia	Internet Society of Armenia, AM NIC	worldwide	OpenAccess	No	2014-10-08T07:56:13Z
AM	AMNIC DC, Yerevan, Armenia		worldwide	OpenAccess	No	2013-04-01T21:48:39Z
AT	Vienna		aco.net, VIX customers, Austrian Internet users	OpenAccess	No	2009-07-17T11:01:08Z
AT	Hallein		Austria/Europe	OpenAccess	No	2019-12-20T19:50:17Z
						2019-12-20T19:11:42Z



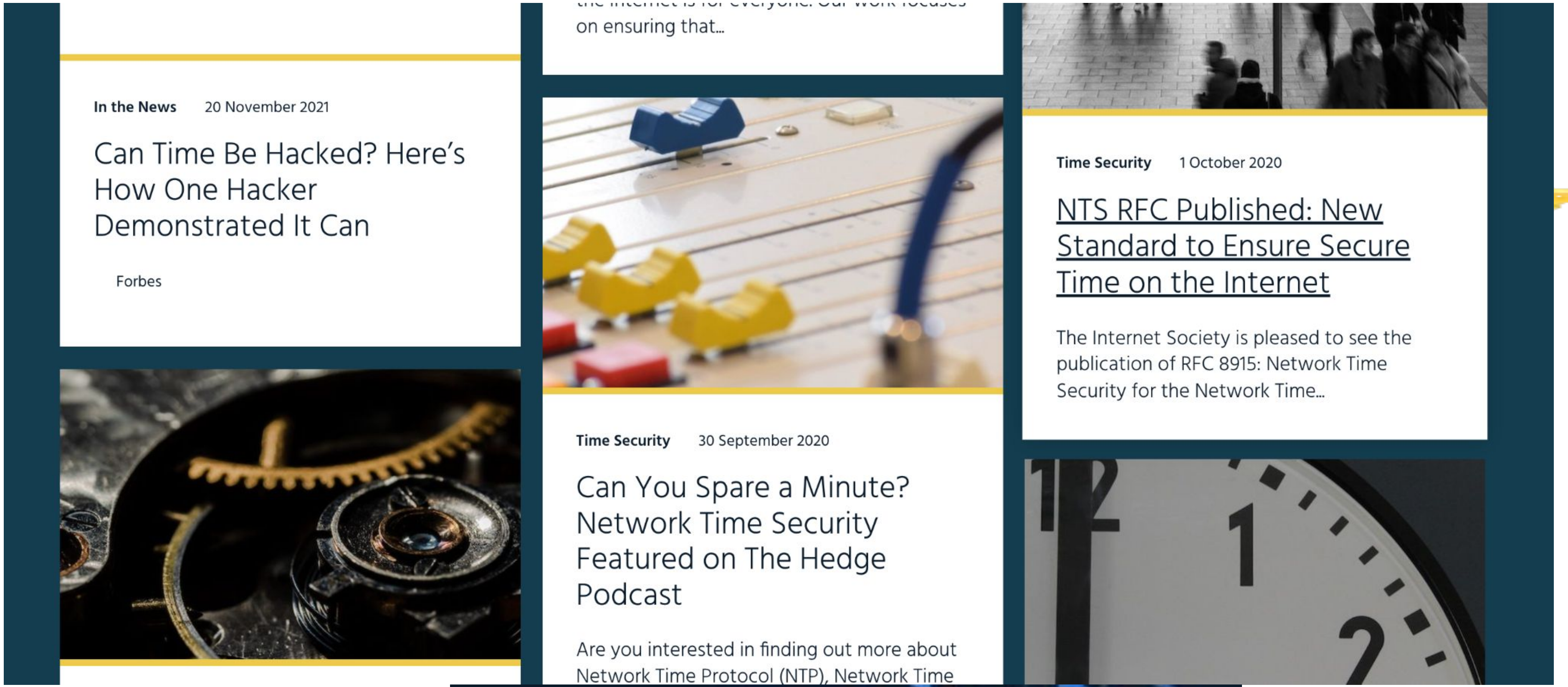
Network Time Foundation: NTP • Ntimed • Linux PTP • RADclock • GTSAPI • More

DRDoS -- concerns about DoS attacks

Search

NTP users are strongly urged to take immediate action to ensure that their NTP daemon is not susceptible to use in a reflected denial-of-service (DRDoS) attack. Please see the NTP Security Notice for vulnerability and mitigation details, and the Network Time Foundation Blog for more information. (January 2014)

lirectly contained within, are in the public domain. Some portions of this site may be copyrighted by other authors. Please contact the webmaster with any questions regarding copyright. The NTP



on ensuring that...

In the News 20 November 2021

Can Time Be Hacked? Here's How One Hacker Demonstrated It Can

Forbes

Time Security 30 September 2020

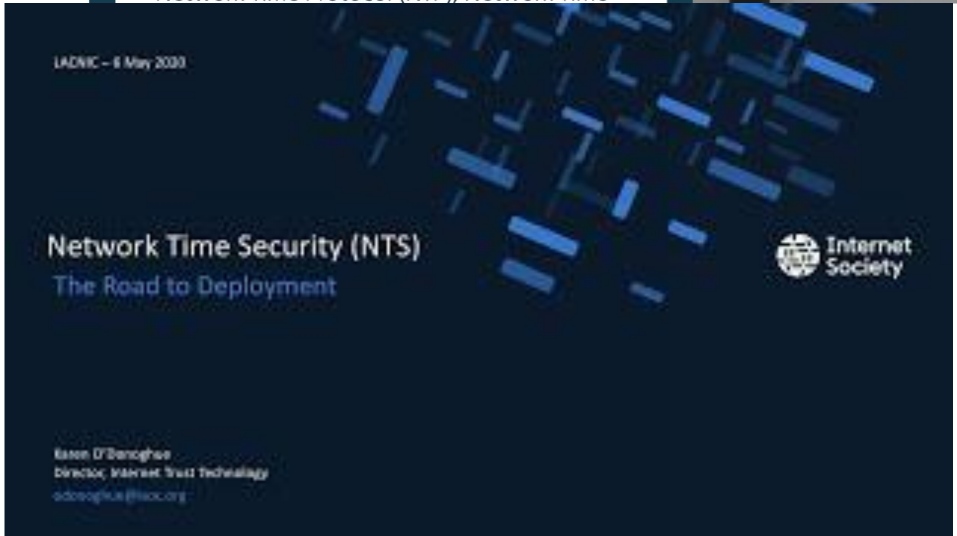
Can You Spare a Minute? Network Time Security Featured on The Hedge Podcast

Are you interested in finding out more about Network Time Protocol (NTP), Network Time

Time Security 1 October 2020

NTS RFC Published: New Standard to Ensure Secure Time on the Internet

The Internet Society is pleased to see the publication of RFC 8915: Network Time Security for the Network Time...



LACNIC – 6 May 2020

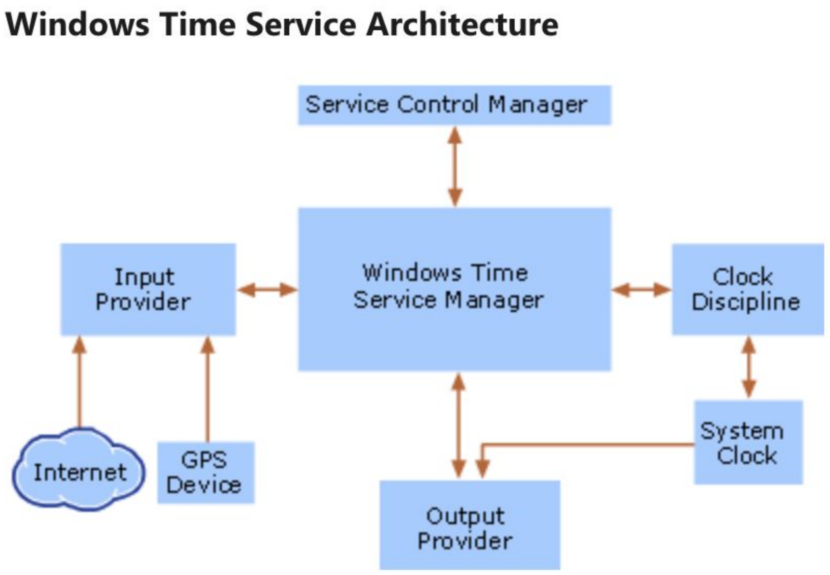
Network Time Security (NTS)

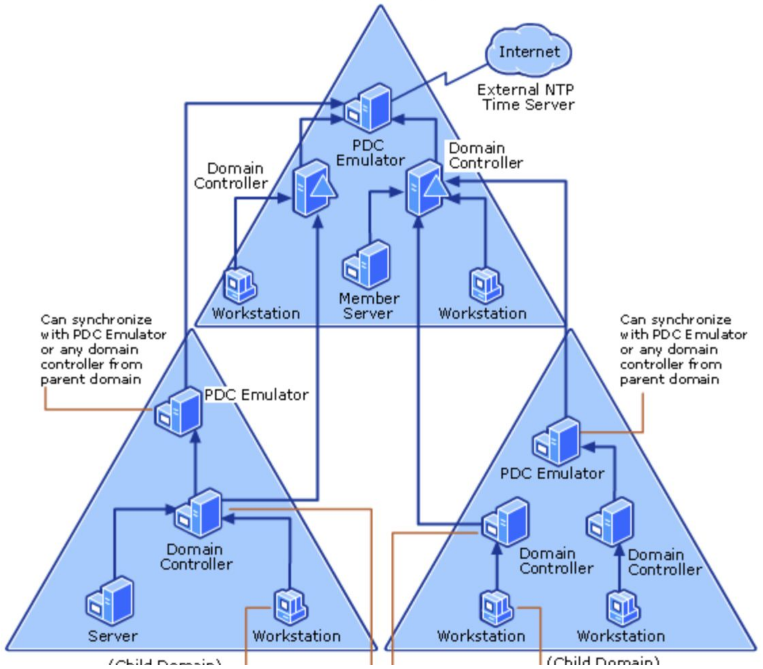
The Road to Deployment

Karen O'Donoghue
Director, Internet Trust Technology
odonoghue@isoc.org

Internet Society

W32 time -- Windows Time Service





The time synchronization process involves the following steps:

- Input providers request and receive time samples from configured NTP time sources.
- These time samples are then passed to the Windows Time Service Manager, which collects all the samples and passes them to the clock discipline subcomponent.
- The clock discipline subcomponent applies the NTP algorithms which results in the selection of the best time sample.
- The clock discipline subcomponent adjusts the time of the system clock to the most accurate time by either adjusting the clock rate or directly changing the time.

If a computer has been designated as a time server, it can send the time on to any computer requesting time synchronization at any point in this process.

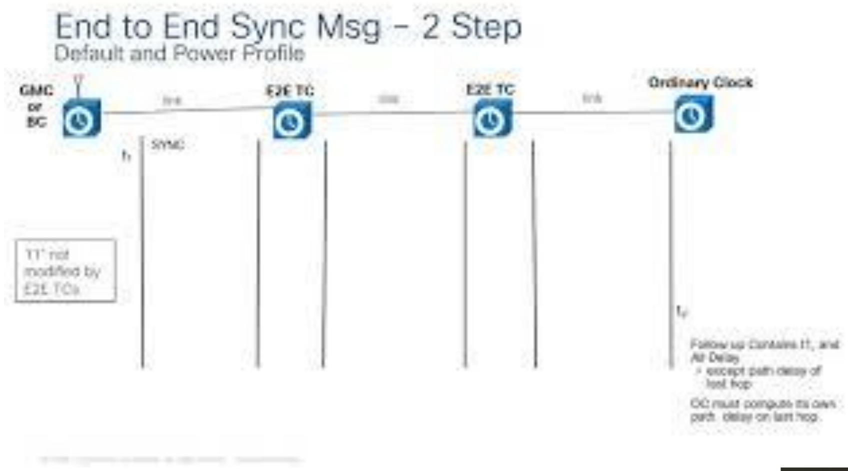
DCE Distributed Time Service

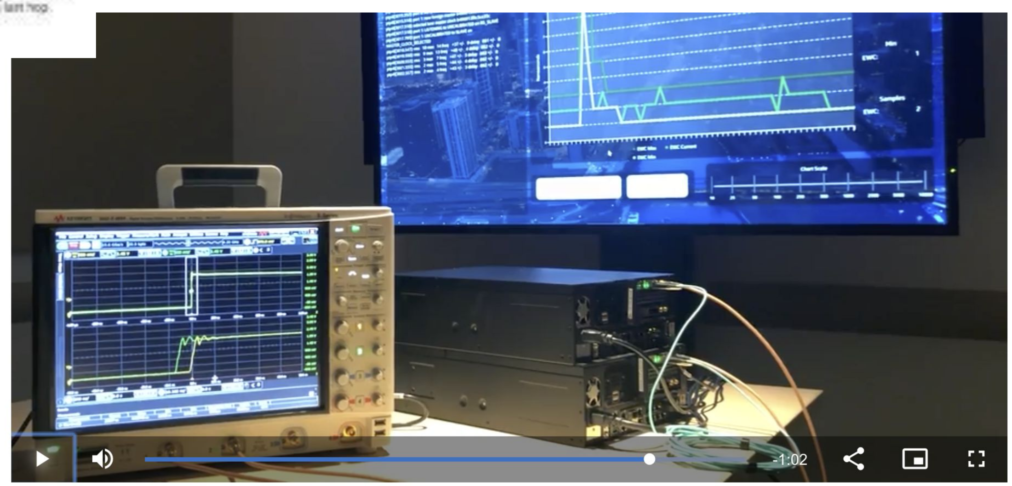
- Software service that provides precise, fault-tolerant clock synchronization for systems in local area networks (LANs) and wide area networks (WANs).
- determine duration, perform event sequencing and scheduling.
- Each machine is either a time server or a clerk
- software components on a group of cooperating systems;
- client obtains time from **DTS entity**
- DTS entities
    - **DTS server**
    - **DTS clerk** that obtain time from DTS servers on other hosts

Clock Synchronization in DCE

- DCE's time model is interval- based
  - Comparing 2 times may yield 3 answers
    - t1 < t2, t2 < t1, not determined
  - Periodically a clerk obtains time-intervals from several servers ,e.g. all the time servers on its LAN
  - Based on their answers, it computes a new time and gradually converges to it.
    - Compute the intersection where the intervals overlap. Clerks then adjust the system clocks of their client systems to the midpoint of the computed intersection.
    - When clerks receive a time interval that does not intersect with the majority, the clerks declare the non-intersecting value to be faulty.
    - Clerks ignore faulty values when computing new times, thereby ensuring that defective server clocks do not affect clients.

PTP (Precision Time protocol)

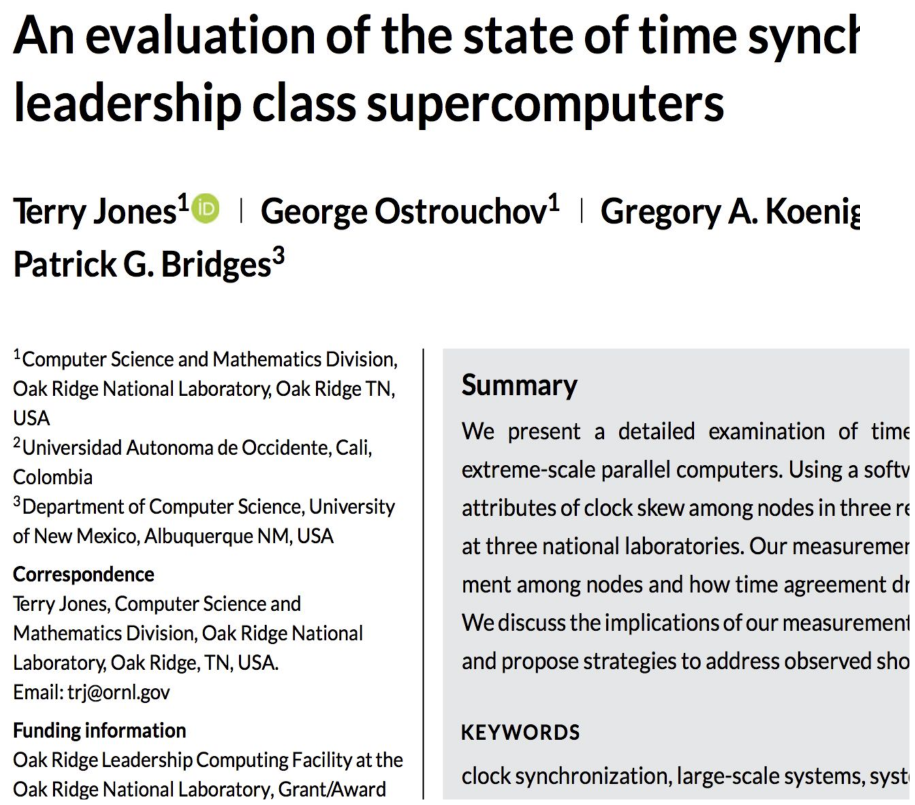

End to End Sync Msg – 2 Step
Default and Power Profile
GMC or BC
link
E2E TC
link
E2E TC
link
Ordinary Clock
SYNC
t1
T1' not modified by E2E TCs
t2
Follow up Contains t1, and All Delay
› except path delay of last hop
OC must compute its own path delay on last hop



5G Network Precision Timing

Advanced 5G use cases require more stringent latency requirements across the network from the central office to the edge. More precise clock synchronization can help solve this challenge. The Intel® Ethernet 700 Series Network Adapter with Hardware Enhanced Precision Time Protocol (PTP) brings high accuracy IEEE 1588 PTP synchronization signaling to the

Precise Time?



An evaluation of the state of time synch
leadership class supercomputers

Terry Jones[1] | George Ostrouchov[1] | Gregory A. Koenig
Patrick G. Bridges[3]

[1]Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge TN, USA
[2]Universidad Autonoma de Occidente, Cali, Colombia
[3]Department of Computer Science, University of New Mexico, Albuquerque NM, USA

Correspondence
Terry Jones, Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN, USA.
Email: trj@ornl.gov

Funding information
Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, Grant/Award

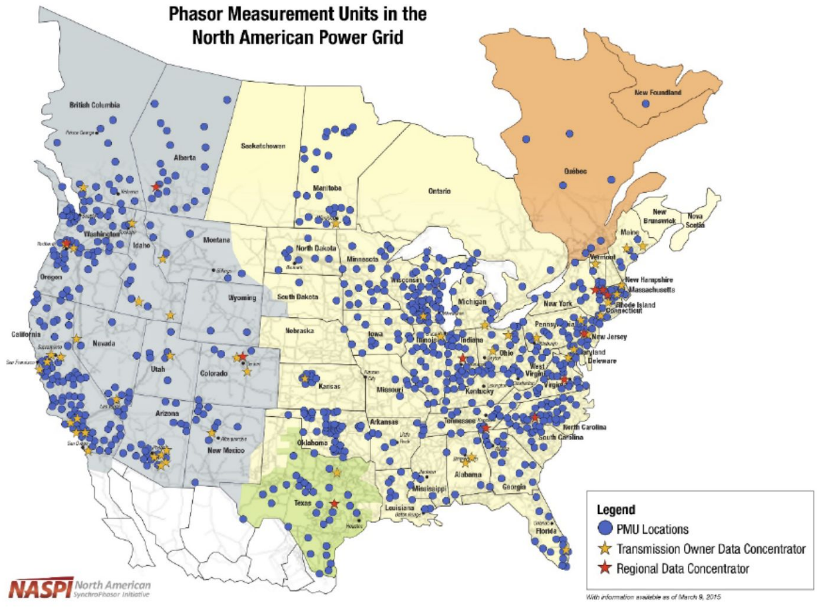
Summary

We present a detailed examination of time
extreme-scale parallel computers. Using a softw
attributes of clock skew among nodes in three re
at three national laboratories. Our measuremen
ment among nodes and how time agreement dr
We discuss the implications of our measurement
and propose strategies to address observed sho

KEYWORDS

clock synchronization, large-scale systems, syst

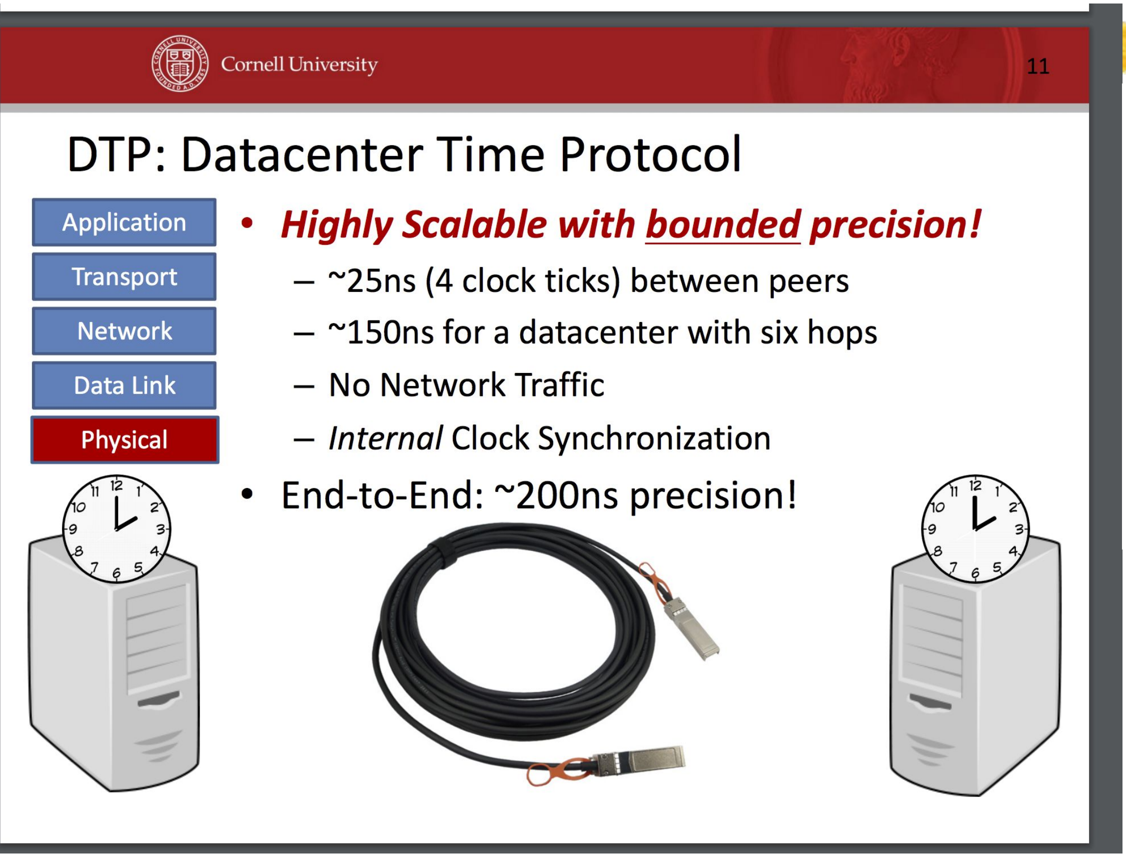
in 2015; many more have been installed since.


Phasor Measurement Units in the North American Power Grid
Legend
● PMU Locations
★ Transmission Owner Data Concentrator
★ Regional Data Concentrator
NASPI North American SynchroPhasor Initiative
With information available as of March 9, 2015

Courtesy of the North American Synchrophasor Initiative () and the U.S. Department of Energy.

Figure 1. Almost 1,800 PMUs were installed across North America in 2015 [NASPI 2015]

DTP (Datacenter Time protocol)



Cornell University 11

DTP: Datacenter Time Protocol

- ***Highly Scalable with bounded precision!***
  - ~25ns (4 clock ticks) between peers
  - ~150ns for a datacenter with six hops
  - No Network Traffic
  - *Internal* Clock Synchronization
- End-to-End: ~200ns precision!

- DTP Prototype
  - Terasic DE5 board with Altera Stratix V
  - Using Bluespec and Connectal framework

Ki Suh Lee, Han Wang, Vishal Shrivastav, and Hakim Weatherspoon. 2016. Globally Synchronized Time via Datacenter Networks. In Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16).

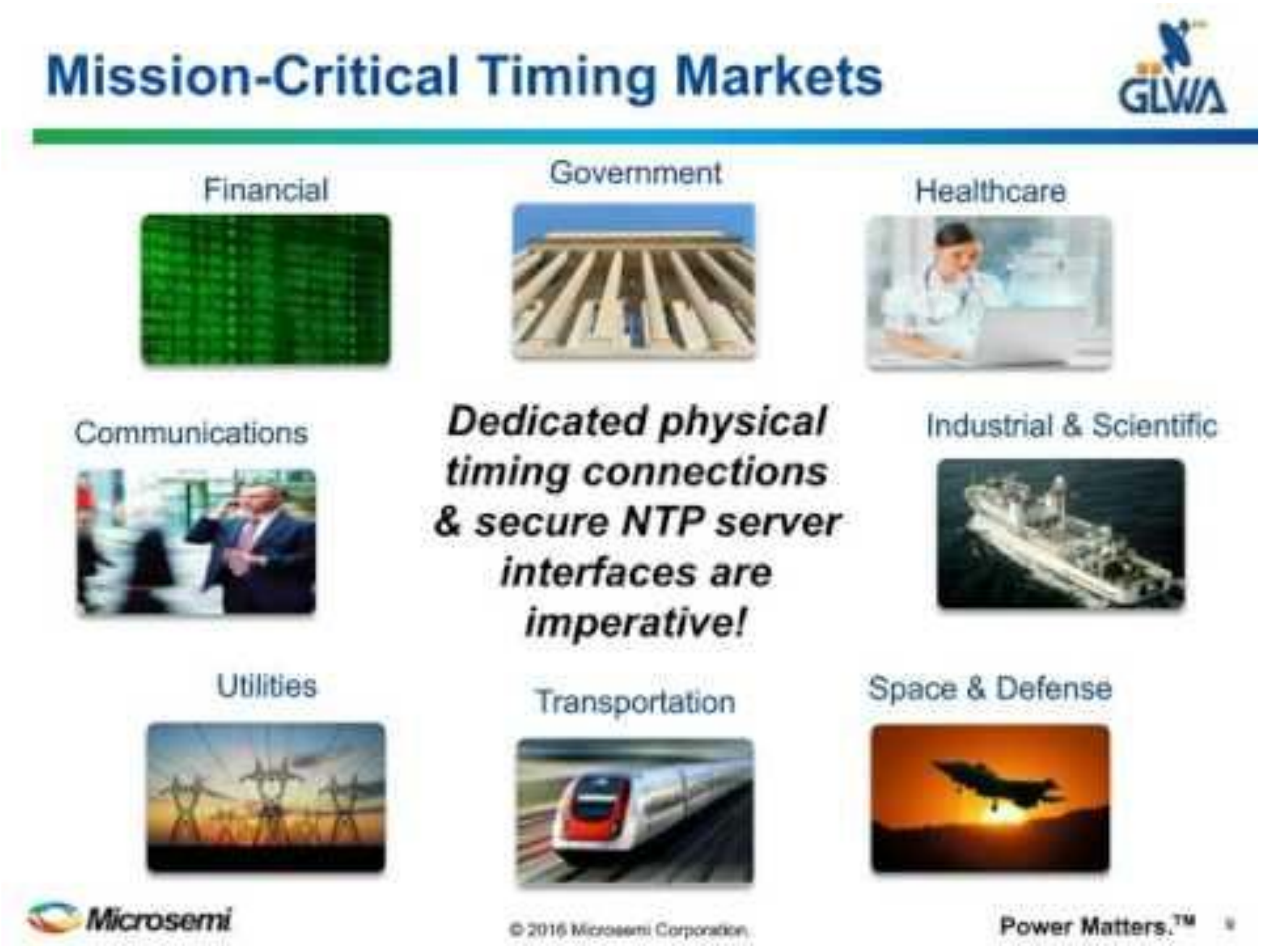
Exploiting a Natural Network Effect for Scalable, Fine-grained Clock Synchronization

Yilong Geng, Shiyu Liu, and Zi Yin, *Stanford University;* Ashish Naik, *Google Inc.;* Balaji Prabhakar and Mendel Rosenblum, *Stanford University;* Amin Vahdat, *Google Inc.*

https://www.usenix.org/conference/nsdi18/presentation/geng

Huygens - software clock synchronization system
- coded probes
- SVM for propogation time estimation
- natural network effect - use clocks to detect errors

Timing in Mission Critical Systems (a Panel)



Mission-Critical Timing Markets

Financial

Government

Healthcare

Communications

Dedicated physical timing connections & secure NTP server interfaces are imperative!

Industrial & Scientific

Utilities

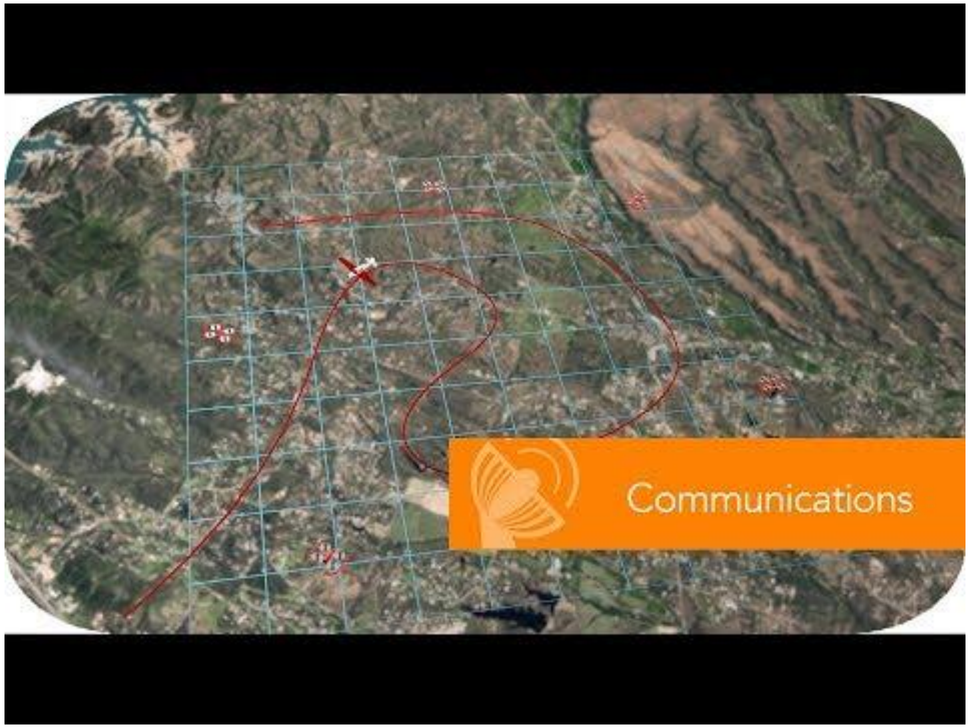
Transportation

Space & Defense

Microsemi

Power Matters.™

NASA example: Agile Local Positioning System



Logical Clock Synchronization

Causal Relations

- Distributed application results in a set of distributed events
  - Induces a partial order ☐ causal precedence relation
- Knowledge of this causal precedence relation is useful in reasoning about and analyzing the properties of distributed computations
  - Liveness and fairness in mutual exclusion
  - Consistency in replicated databases
  - Distributed debugging, checkpointing

Logical Clocks

- Used to determine causality in distributed systems
- Time is represented by non-negative integers
- Event structures represent distributed computation (in an abstract way)
  - A process can be viewed as consisting of a sequence of events, where an event is an atomic transition of the local state which happens in no time
  - Process Actions can be modeled using the 3 types of events
    - Send Message
    - Receive Message
    - Internal (change of state)

Logical Clocks

- A logical Clock C is some abstract mechanism which assigns to any event $e \in E$ the value $C(e)$ of some time domain T such that certain conditions are met
    - $C: E \rightarrow T$ :: T is a partially ordered set : $e<e' \rightarrow C(e)<C(e')$ holds
- Consequences of the clock condition [Morgan 85]:
    - Events occurring at a particular process are totally ordered by their local sequence of occurrence
        - If an event e occurs before event e' at some single process, then event e is assigned a logical time earlier than the logical time assigned to event e'
    - For any message sent from one process to another, the logical time of the send event is always earlier than the logical time of the receive event
        - Each receive event has a corresponding send event
        - Future can not influence the past (causality relation)

Event Ordering

- Lamport defined the “happens before” (=>) relation
  - If a and b are events in the same process, and a occurs before b, then a => b.
  - If a is the event of a message being sent by one process and b is the event of the message being received by another process, then a => b.
  - If X =>Y and Y=>Z then X => Z.

If a => b then time (a) => time (b)

Causal Ordering

- “Happens Before” also called causal ordering
- Possible to draw a causality relation between 2 events if
  - They happen in the same process
  - There is a chain of messages between them
- “Happens Before” notion is not straightforward in distributed systems
  - No guarantees of synchronized clocks
  - Communication latency

Implementation of Logical Clocks

- Requires
  - Data structures local to every process to represent logical time and
  - a protocol to update the data structures to ensure the consistency condition.
- Each process Pi maintains data structures that allow it the following two capabilities:
  - A local logical clock, denoted by LC_i , that helps process Pi measure its own progress.
  - A logical global clock, denoted by GCi , that is a representation of process Pi 's local view of the logical global time. Typically, lci is a part of gci
- The protocol ensures that a process's logical clock, and thus its view of the global time, is managed consistently.
  - The protocol consists of the following two rules:
    - R1: This rule governs how the local logical clock is updated by a process when it executes an event.
    - R2: This rule governs how a process updates its global logical clock to update its view of the global time and global progress.

Types of Logical Clocks

- Systems of logical clocks differ in their representation of logical time and also in the protocol to update the logical clocks.
- 3 kinds of logical clocks
    - Scalar
    - Vector
    - Matrix

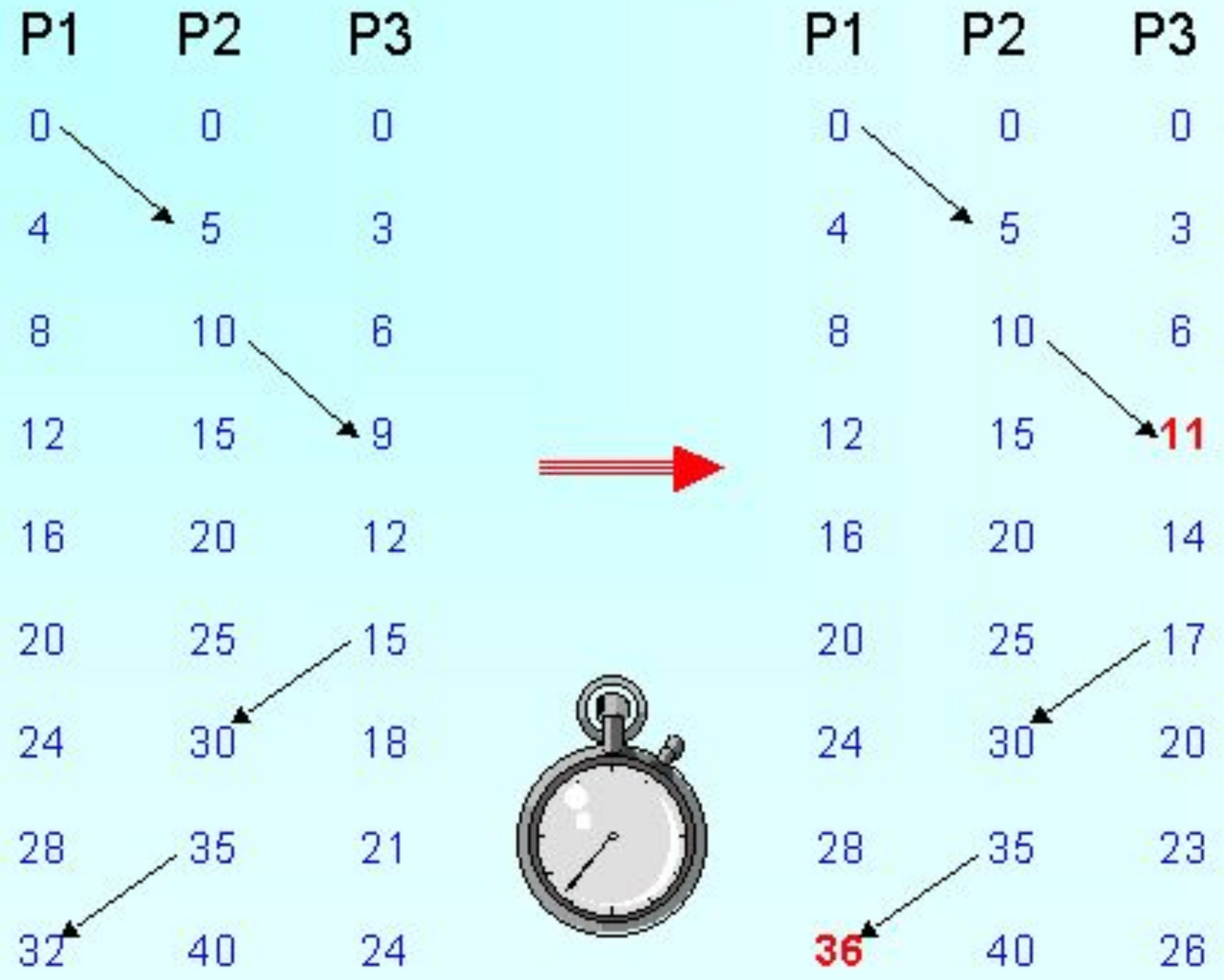
Scalar Logical Clocks - Lamport

- Proposed by Lamport in 1978 as an attempt to totally order events in a distributed system.
- Time domain is the set of non-negative integers.
- The logical local clock of a process pi and its local view of the global time are squashed into one integer variable Ci .
- Monotonically increasing counter
  - No relation with real clock
- Each process keeps its own logical clock used to timestamp events

Consistency with Scalar Clocks

- To guarantee the clock condition, local clocks must obey a simple protocol:
  - When executing an internal event or a send event at process P_i the clock C_i ticks
    - C_i += d (d>0)
  - When P_i sends a message m, it piggybacks a logical timestamp t which equals the time of the send event
  - When executing a receive event at P_i where a message with timestamp *t* is received, the clock is advanced
    - $C_i = \max(C_i, t) + d$ (d>0)
- Results in a partial ordering of events.

Lamport Logical Clock



P1	P2	P3
0	0	0
4	5	3
8	10	6
12	15	9
16	20	12
20	25	15
24	30	18
28	35	21
32	40	24

P1	P2	P3
0	0	0
4	5	3
8	10	6
12	15	**11**
16	20	14
20	25	17
24	30	20
28	35	23
36	40	26

Total Ordering

- Extending partial order to total order

time	Proc_id

- Global timestamps:
  - (Ta, Pa) where Ta is the local timestamp and Pa is the process id.
  - (Ta,Pa) < (Tb,Pb) iff
    - (Ta < Tb) or ((Ta = Tb) and (Pa < Pb))
  - Total order is consistent with partial order.

Properties of Scalar Clocks

- No Strong Consistency
- The system of scalar clocks is not strongly consistent; that is, for two events ei and ej , C(ei) < C(ej) does not imply ei → ej .
- Reason: In scalar clocks, logical local clock and logical global clock of a process are squashed into one, resulting in the loss of causal dependency information among events at different processes.

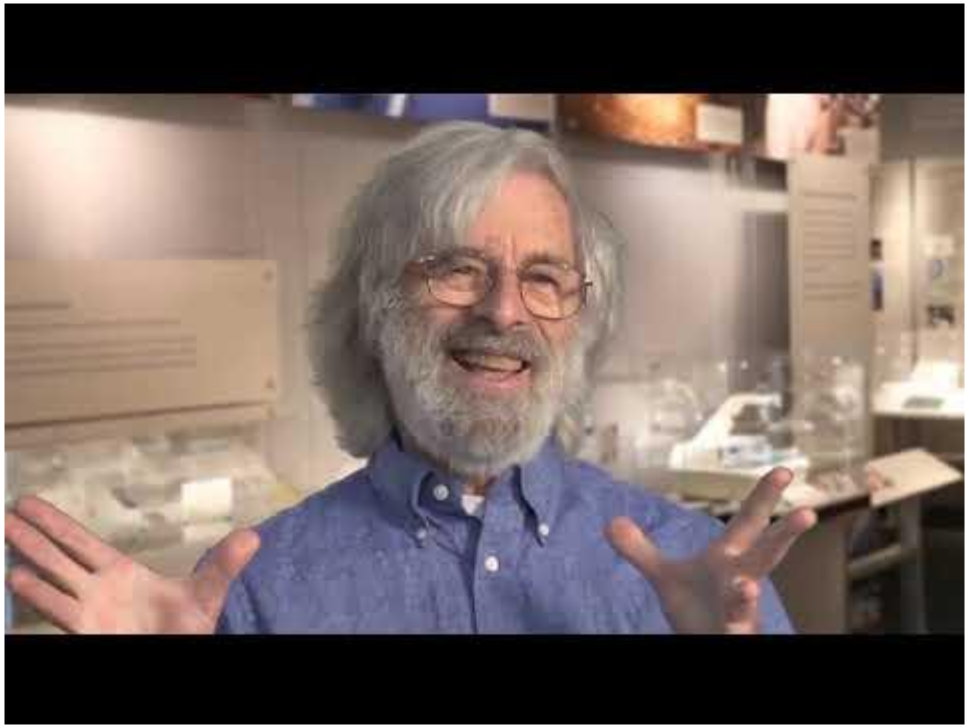
Independence

- Two events e,e′ are mutually independent (i.e. e||e′) if ~(e<e′)$\wedge$~(e′<e)
  - Two events are independent if they have the same timestamp
  - Events which are causally independent may get the same or different timestamps
- By looking at the timestamps of events it is not possible to assert that some event *could not* influence some other event
  - If C(e)<C(e′) then ~(e′<e) *however*, it *is not possible* to decide whether e<e′ or e||e′
  - C is an order *homomorphism* which preserves < but it does not preserves negations (i.e. obliterates a lot of structure by mapping E into a linear order)

Problems with Total Ordering

- A linearly ordered structure of time is not always adequate for distributed systems
  - captures dependence of events
  - loses independence of events - artificially enforces an ordering for events that need not be ordered – loses information
- Mapping partial ordered events onto a linearly ordered set of integers is *losing information*
    - Events which may happen simultaneously may get different timestamps as if they happen in some definite order.
- A partially ordered system of *vectors* forming a *lattice* structure is a natural representation of time in a distributed system

Lamport on clocks...



Vector Clocks

- Independently developed by Fidge, Mattern and Schmuck.
- Aim: To construct a mechanism by which each process gets an optimal approximation of global time
- Time representation
  - Set of n-dimensional non-negative integer vectors.
  - Each process has a clock C_i consisting of a vector of length *n*, where *n* is the total number of processes vt[1..n], where vt[j] is the local logical clock of Pj and describes the logical time progress at process Pj .
  - A process P_i ticks by incrementing its own component of its clock
    - $C_i[i] += 1$
  - The timestamp C(e) of an event e is the clock value after ticking
  - Each message gets a piggybacked timestamp consisting of the vector of the local clock
    - The process gets some knowledge about the other process' time approximation
    - $C_i = \sup(C_i, t) :: \sup(u,v) = w : w[i] = \max(u[i], v[i]), \ \forall i$

Vector Clocks example

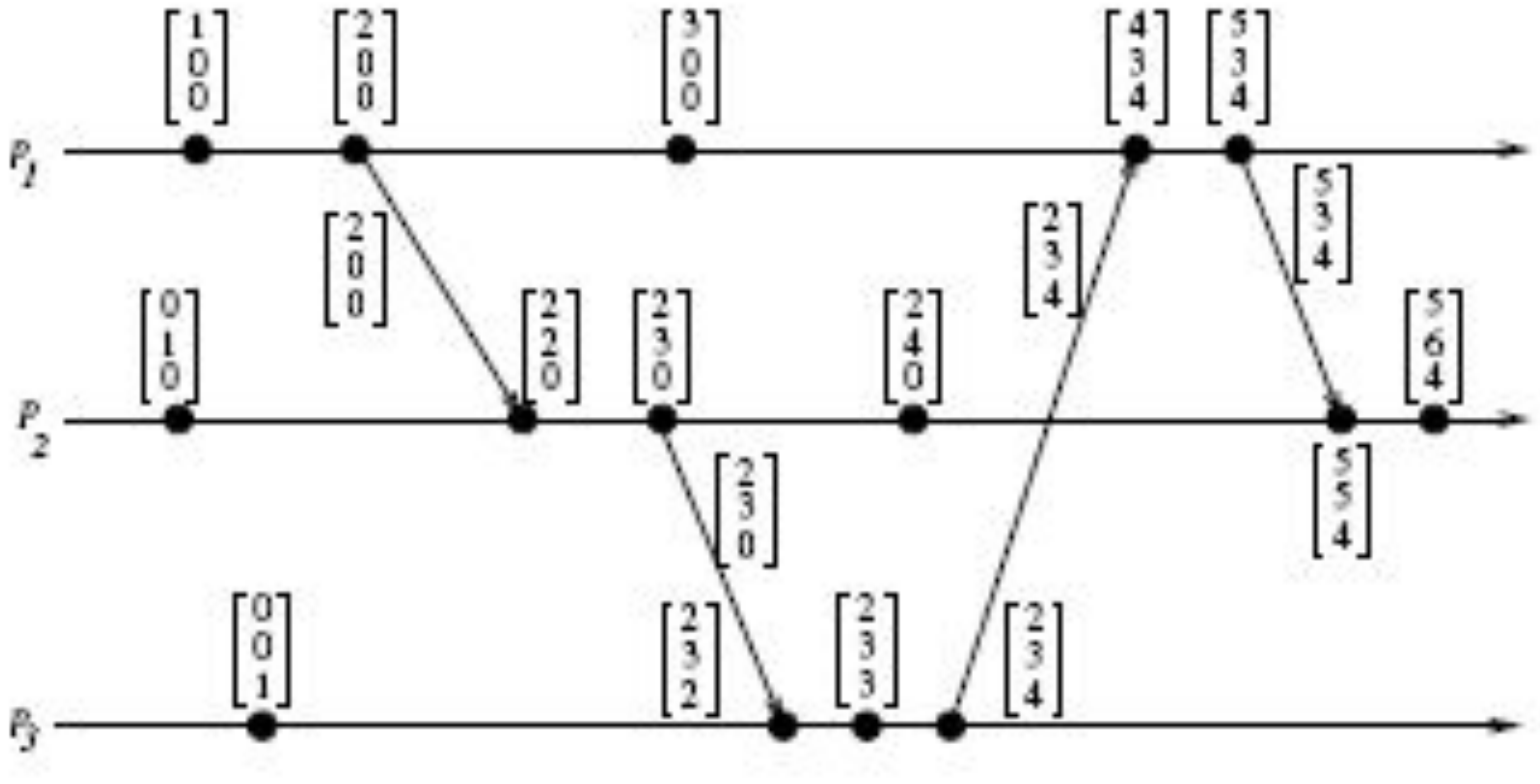


Figure 3.2: Evolution of vector time.

From A. Kshemkalyani and M. Singhal (Distributed Computing)

Vector Times (cont)

- Because of the transitive nature of the scheme, a process *may receive* time updates about clocks in non-neighboring process
- Since process P_i can advance the ith component of global time, it always has the most accurate knowledge of its local time
    - At any instant of real time $\forall i,j$: $C_i[i] \geq C_j[i]$

Structure of the Vector Time

- For two time vectors u,v
    - $u \le v$ iff $\forall i: u[i] \le v[i]$
    - $u < v$ iff $u \le v \wedge u \ne v$
    - $u || v$ iff $\sim(u<v) \wedge \sim(v<u)$ :: || is not transitive
- For an event set E,
    - $\forall e,e' \in E: e<e'$ iff $C(e)<C(e') \wedge e||e'$ iff $C(e)||C(e')$
- In order to determine if two events e,e' are causally related or not, just take their timestamps C(e) and C(e')
    - if $C(e)<C(e') \vee C(e')<C(e)$, then the events *are causally related*
    - Otherwise, they *are causally independent*

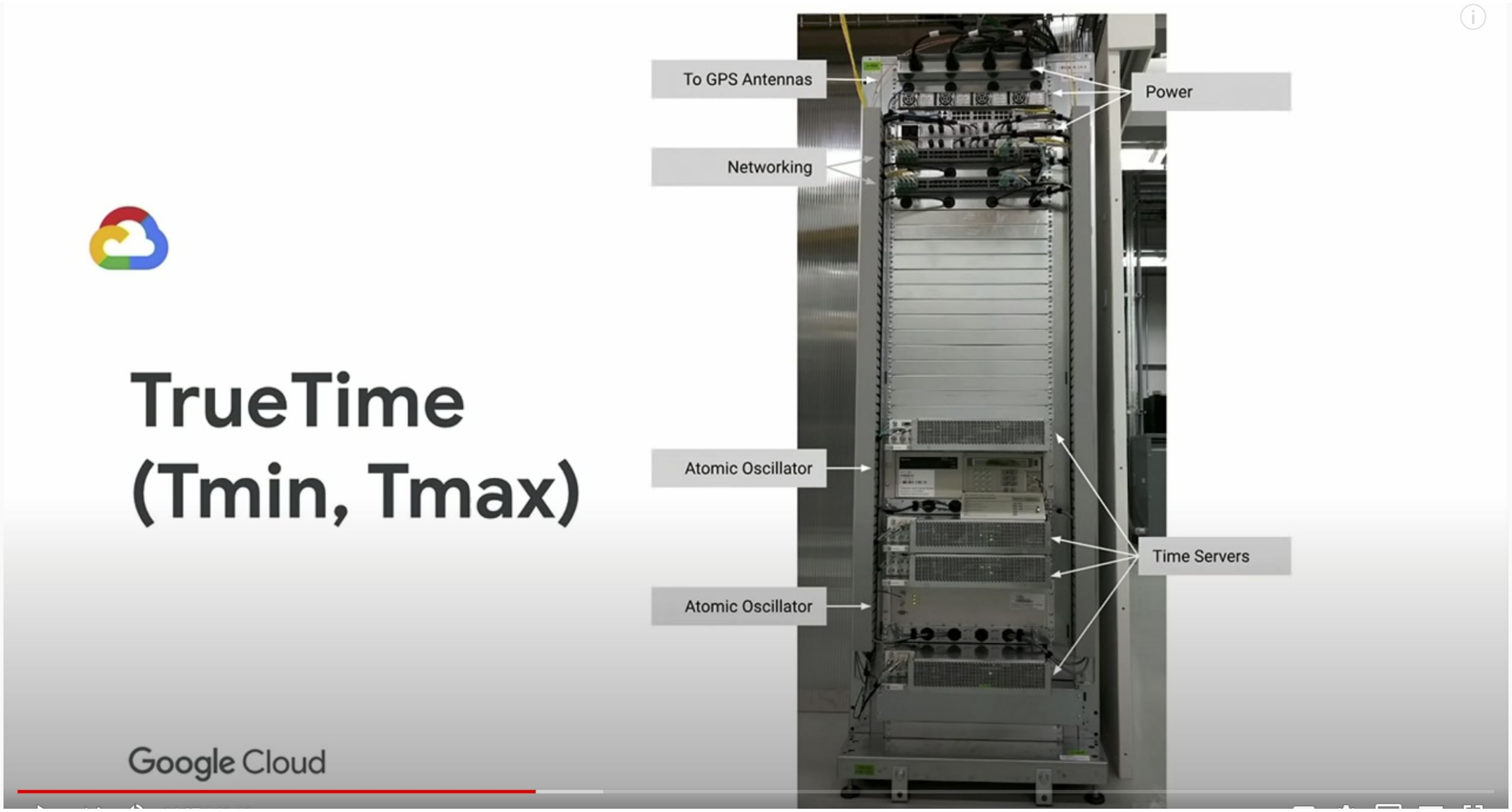
Matrix Time

- Vector time contains information about latest direct dependencies
  - What does Pi know about Pk
- Also contains info about latest direct dependencies of those dependencies
  - What does Pi know about what Pk knows about Pj
- Message and computation overheads are high
- Powerful and useful for applications like distributed garbage collection

Time Manager Operations

- Logical Clocks
  - C.adjust(L,T)
    - adjust the local time displayed by clock C to T (can be gradually, immediate, per clock sync period)
  - C.read
    - returns the current value of clock C
- Timers
  - TP.set(T) - reset the timer to timeout in T units
- Messages
  - receive(m,l); broadcast(m); forward(m,l)

Spanner: Google's Globally Distributed Database



TrueTime
(Tmin, Tmax)

Google Cloud

What is Spanner?

A **scalable**, **multi-version**, **globally-distributed**, **synchronously- replicated** database

- **Scalable**
  - Data centers and servers can be added or removed as needed for availability and performance
  - Automatic Load Balancing among datacenters
- **Multi-version**
  - Data has a timestamp, enabling historical data queries
- **Globally-Distributed**:
  - Data centers can be separated by great distances
  - Provides Locality and fault tolerance
- **Synchronously-replicated**:
  - Replication results in multiple up-to-date copies of data

Main idea:

– Get externally consistent view of globally distributed database

– Spanner = BigTable with timestamps + Paxos + TrueTime

Google’s Setting

- Dozens of zones (datacenters) across the globe
- Per zone, 100-1000s of servers
- Per server, 100-1000 partitions (tablets)
- Every tablet replicated for fault-tolerance (e.g., 5x) 18 Google’s Setting
    - client-application configurable

Heterogeneous Workloads

- SQL → NoSQL → NewSQL
  - Supports multiple types of query languages
- Large scale transactional databases
- Eventual consistency is not good enough:
  - Managing global money/warehouses/resources
  - Auctions, especially Google's advertisement platform
  - Social networks, Twitter
  - MapReduce over a globally changing dataset
- We need external consistency:
  - $T(e1(commit)) < T(e2(start)) \rightarrow s1 < s2$

Concepts

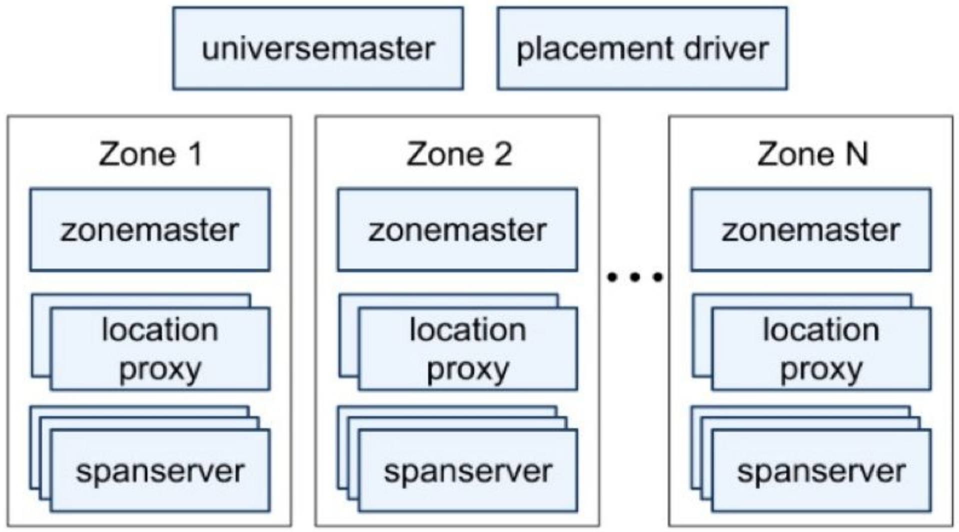
Spanner Guarantees

- **External consistency:** Transactions are processed in a way equivalent to a single machine processing all transactions. Analogous to distributed shared memory, which provides an abstraction for many memories as one centralized memory
- **Global consistency:** Commits that are made are visible to all replicas. Queries made from different data centers give the same values.

Provides full control over

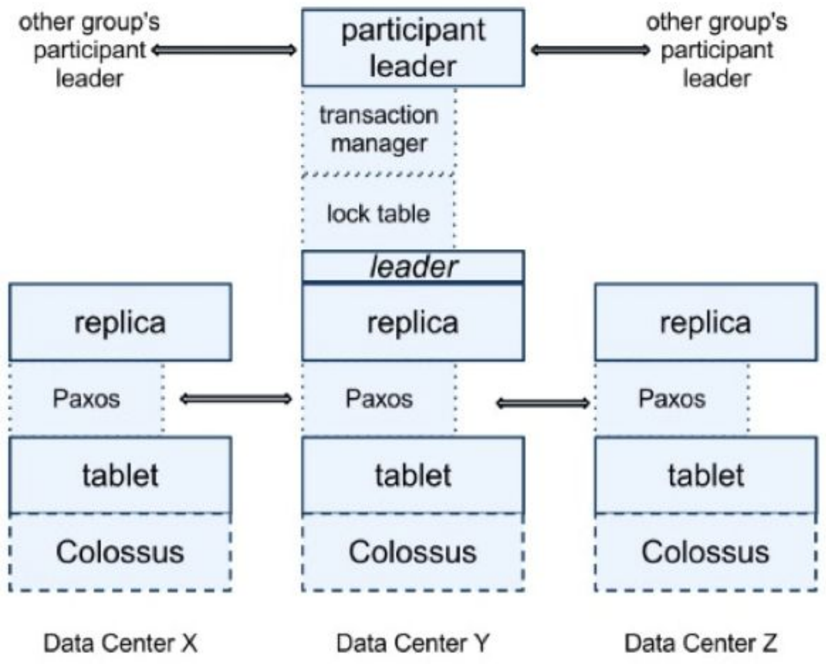
- How far data is from user (read latency)
- How far replicas are from each other (write latency)
- How many replicas (durability, availability, throughput)

Spanner: Storage Architecture



- **Universe:** The entire Spanner deployment
- **Zone:** A cluster of machines in a data center used to serve data to geographically close clients.
  - A universe has an arbitrary number of zones.
- **spanserver:** A store; A zone may have hundreds to thousands of spanservers.
- **zonemaster:** assigns data to spanservers; A zone has exactly one zonemaster
- **universemaster:** provides status information about all zones in universe for debugging
- **placement driver:** handles movement of data between zones

SpanServer Architecture and Data Model



Colossus:
A distributed File System

Spanner provides semi-relational interface with SQL-like query language:

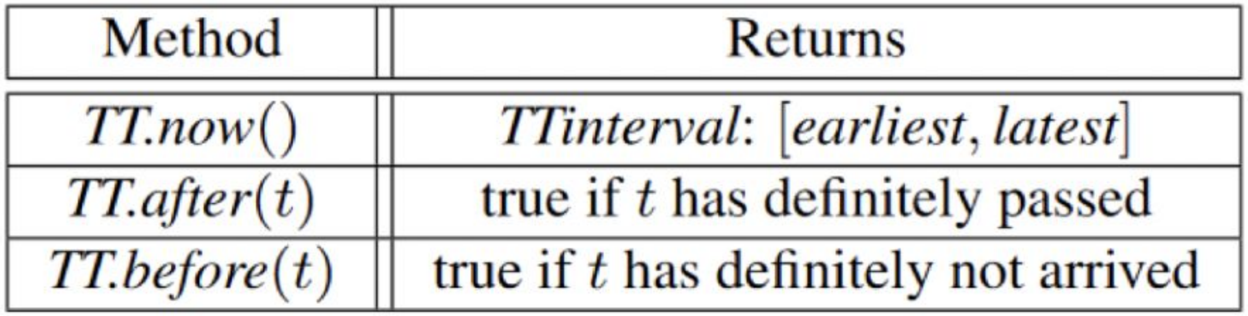
- Database: Split among many zones geographically separated
- Tables : Split among many tablets replicated across data centers
- Rows: Collection of values for a variety of data columns, timestamped.
- Columns: Values from the internal key-value store.

Paxos State Machine: Participants in Paxos protocol for consistency

TrueTime

Goal: Provide globally synchronized time with sharp error bounds, i.e. bounded uncertainty

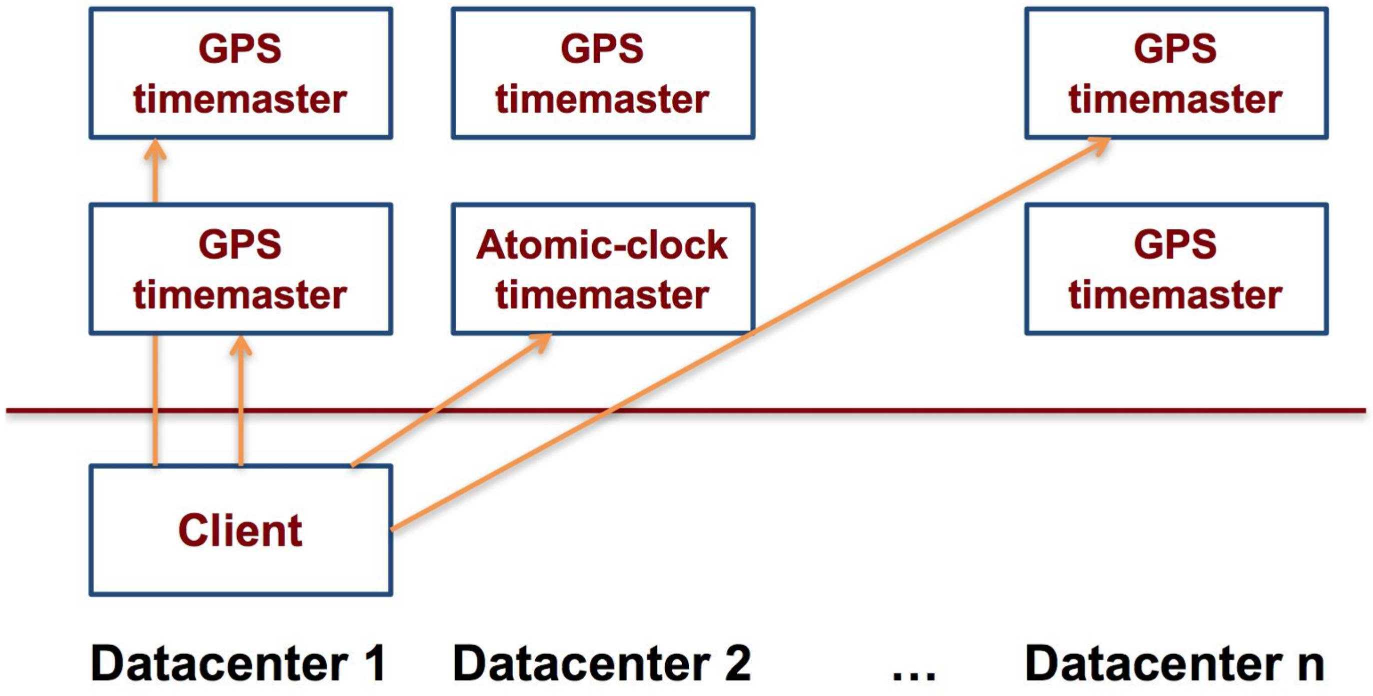
- Interval-based global time.
- Do not trust synchronization via NTP Implemented using GPS and "commodity" atomic clocks



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

- Spanner implements algorithms to make sure these guarantees are respected by the machines (non-conformists are evicted)
  - Uses a variant of Marzullo's algorithm (estimates time from multiple noisy sources)
  - Time accuracy on the order of 10ms (avg. error bound ~4 ms)

TrueTime Architecture



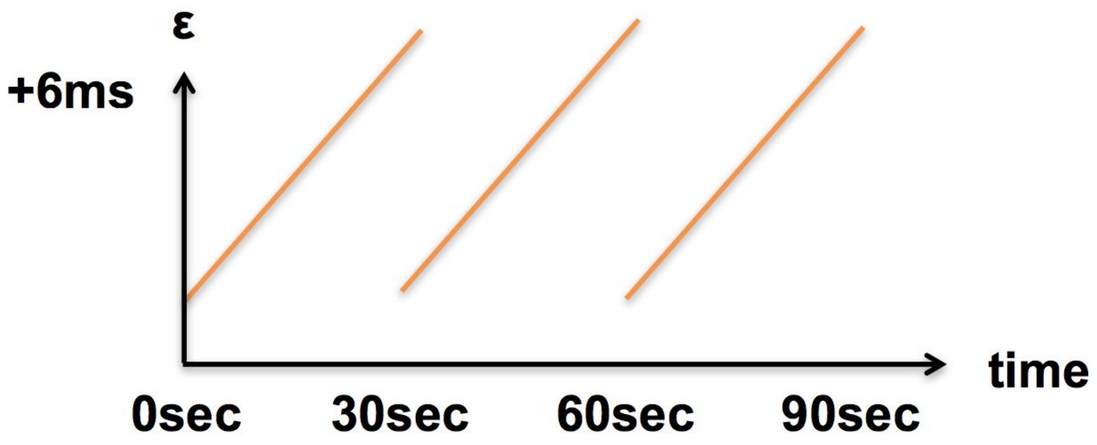
Compute reference [earliest, latest] = now ± ε

TrueTime implementation

now = reference now + local-clock offset

ε = reference ε + worst-case local-clock drift

  = 1ms + 200 μs/sec



What about faulty clocks? – Bad CPUs 6x more likely in 1 year of empirical data

References

- Princeton - COS 418: Distributed Systems course slides by Themis Melissaris and Daniel Suo
- Spanner: Google's Globally Distributed Database by Philipp Moritz
- https://youtu.be/NthK17nbpYs

Case Study: Amazon DynamoDB

Amazon serves millions of customers at peak time

- Require availability - "always-on" experience.
- Loss of service has financial consequences, erodes user trust

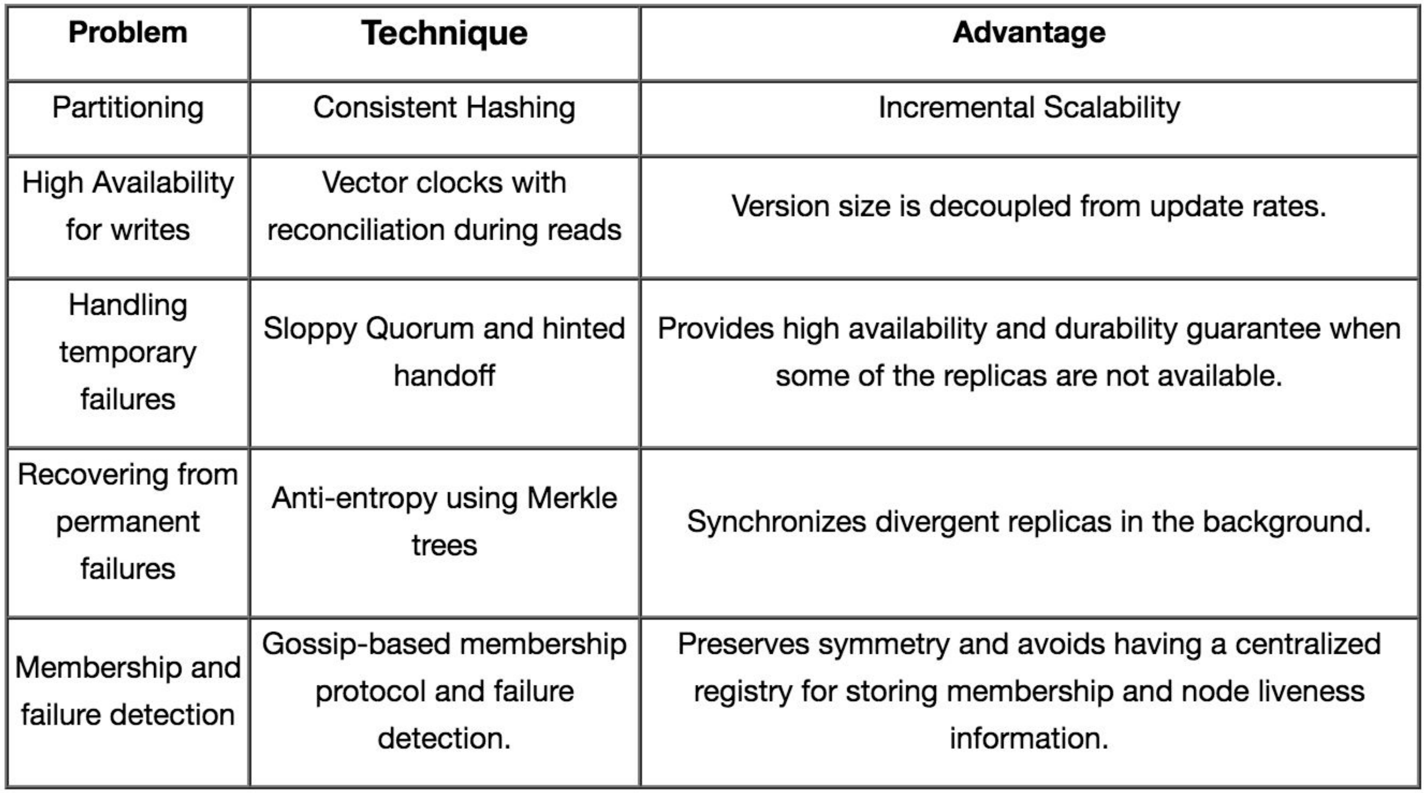
What is Dynamo?

- A key-value distributed data storage system
- Emphasis on Reliability, scalability and availability
- Used in AWS - highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services.

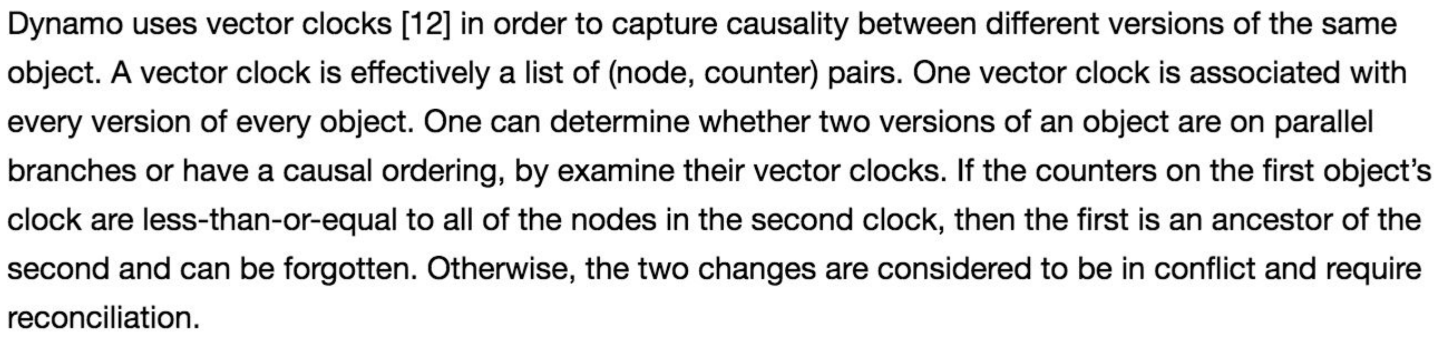
Dynamo Application Characteristics

- "always writeable" store, no updates are rejected despite failures/concurrent writes
- Single administrative domain where all nodes are assumed to be trusted.
- No hierarchical namespaces (e.g file systems) or complex relational schema
- Latency sensitive applications (99.9% of R/W operations < few 100ms)

Usecase: Amazon DynamoDB



Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.



Dynamo uses vector clocks [12] in order to capture causality between different versions of the same object. A vector clock is effectively a list of (node, counter) pairs. One vector clock is associated with every version of every object. One can determine whether two versions of an object are on parallel branches or have a causal ordering, by examine their vector clocks. If the counters on the first object's clock are less-than-or-equal to all of the nodes in the second clock, then the first is an ancestor of the second and can be forgotten. Otherwise, the two changes are considered to be in conflict and require reconciliation.

https://www.allthingsdistributed.com/2007/10/amazons_dynamo.html

Clock skews in Cassandra

Timestamps are critical for coordination in a distributed key-value store.

But issues can arise!!

-- Last-write wins problem..

pagerduty

Clock Skew, and other annoying realities in distributed systems

Donny Nadolny

donny@pagerduty.com

#CassandraSummit

Cassandra

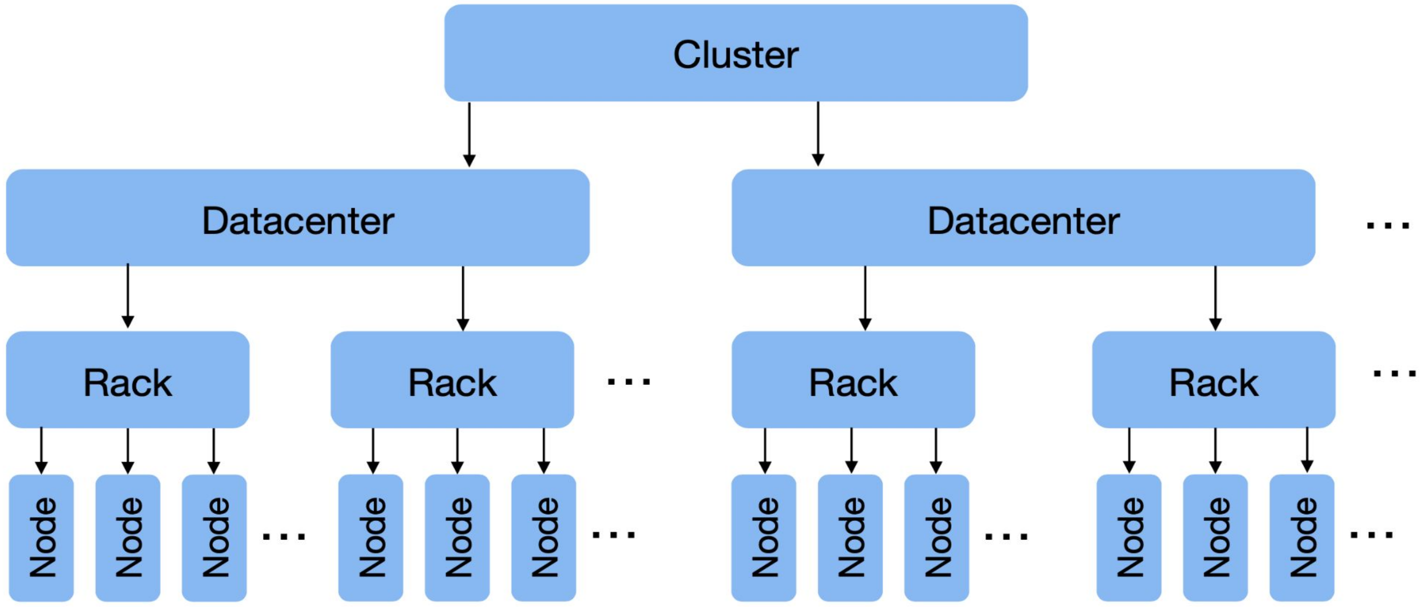
- Distributed, customizable, eventually-consistent NoSQL database
- Originated at Facebook in 2008
  - Currently an Apache open-source project
  - ACID relational database systems were too slow
  - CAP theorem: strong consistency → availability suffers
    - Performance also suffers (e.g., locking, running commit protocols)
  - Go with an eventually consistent model
  - Built as a combination of Amazon Dynamo DHT + Google Bigtable

Cassandra Design Goals

• High availability – no single point of failure - no central coordinator • Low latency
• Run on commodity hardware
• Linear performance increase with additional nodes

• Tunable consistency: Define replication # and policy
• Key-oriented queries
• Flexible data model: row can have different columns with different data types
• SQL-like query language (CQL - Cassandra Query Language)

Cassandra Machine Hierarchy

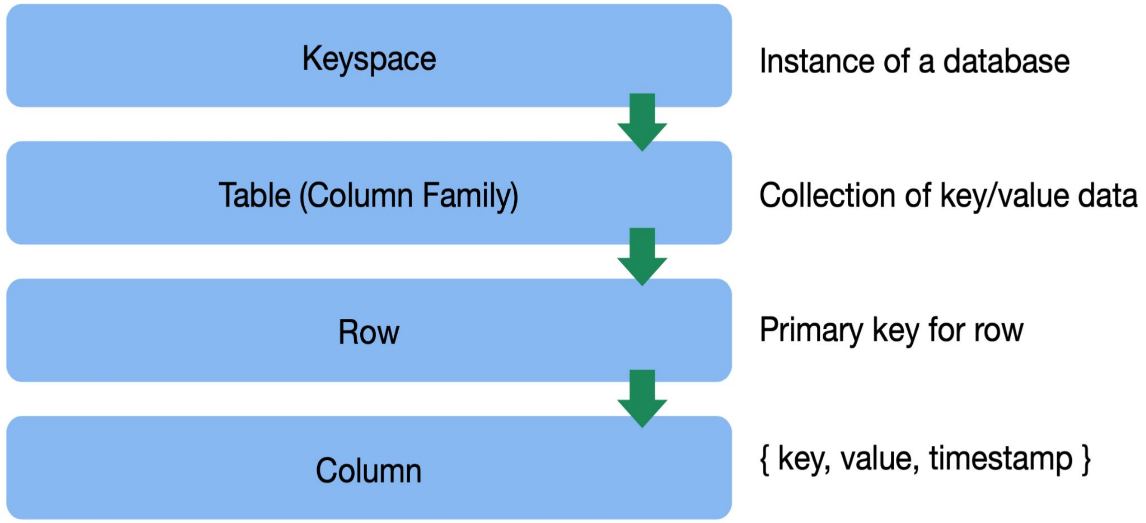


Cluster – Collection of machines that run Cassandra
– Nodes are arranged in a logical ring (like Chord/Dynamo) and data is replicated

- **Datacenter** – Machines in one location (data center)
- **Rack** – Machines in one rack (low latency – single switch connection)
- **Node** – Individual machine

Figure taken from Paul Krzyzanowski's slides. https://people.cs.rutgers.edu/~pxk/417/notes/pdf/09b-cassandra-slides.pdf

Cassandra Data Model



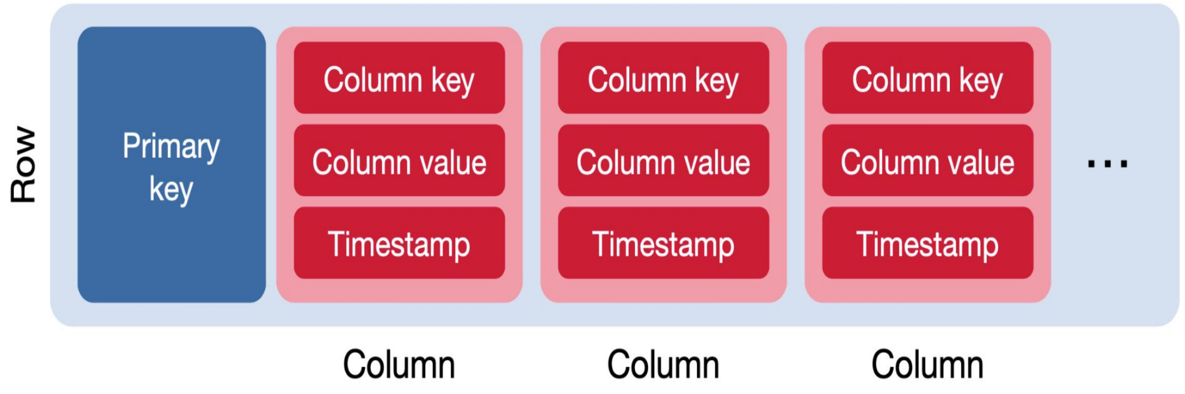


Figure taken from Paul Krzyzanowski's slides. https://people.cs.rutgers.edu/~pxk/417/notes/pdf/09b-cassandra-slides.pdf

Column: Fundamental unit of storage

– { name, value, timestamp }

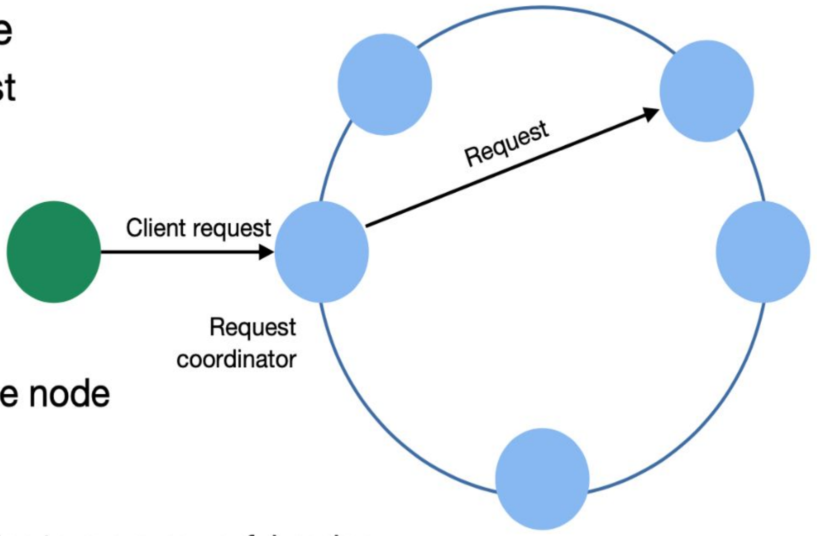
– Each column has a name (key) that can be queried for a value • Data types (>20) include alphabetic, numeric, blob, time, set, map

– **Timestamp enables conflict resolution among replicas • Usually created by client – synchronized clocks assumed**

– Columns can also be given an optional expiration timestamp (time to live)

Cassandra Storage/query Model

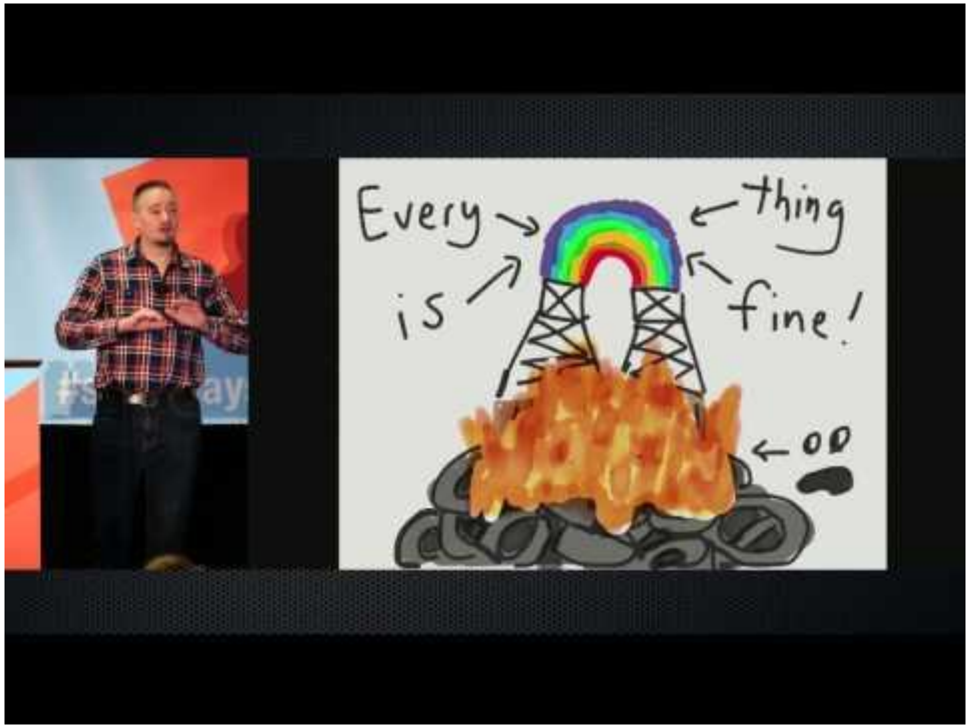
- A client request can go to any Cassandra node
  - That node will act as a coordinator for that request
  - Communication uses Apache Thrift RPC
- Dynamo/Chord style distributed hash table
  - Systems arranged in a logical ring – each node responsible for a hash range
  - Coordinator hashes the partition key to identify the node
  - Virtual nodes, vnodes (like Dynamo)
    - Each node can own a many hash ranges
    - Makes it easy to alleviate hotspots and give more vnodes to more powerful nodes
  - Replicas are found by following the ring clockwise



- **Replica reconciliation** – what happens when there are conflicts?
  - Not like Dynamo's vector clocks
  - Last write wins model where every mutation is **timestamped (including deletes) and then the latest version of data is the "winning" value**

Figure taken from Paul Krzyzanowski's slides. https://people.cs.rutgers.edu/~pxk/417/notes/pdf/09b-cassandra-slides.pdf

Coordination Problems



Creating Global State

- The notions of global time and global state are closely related
- Goal: Compute, for a process, without *freezing* the whole computation, the *best possible approximation* of a global state [Chandy & Lamport 85]
- A global state that *could* have occurred
  - No process in the system can decide whether the state did really occur
  - Guarantee stable properties (i.e. once they become true, they remain true)

What constitutes global state?

- **Global Snapshot** = **Global State**

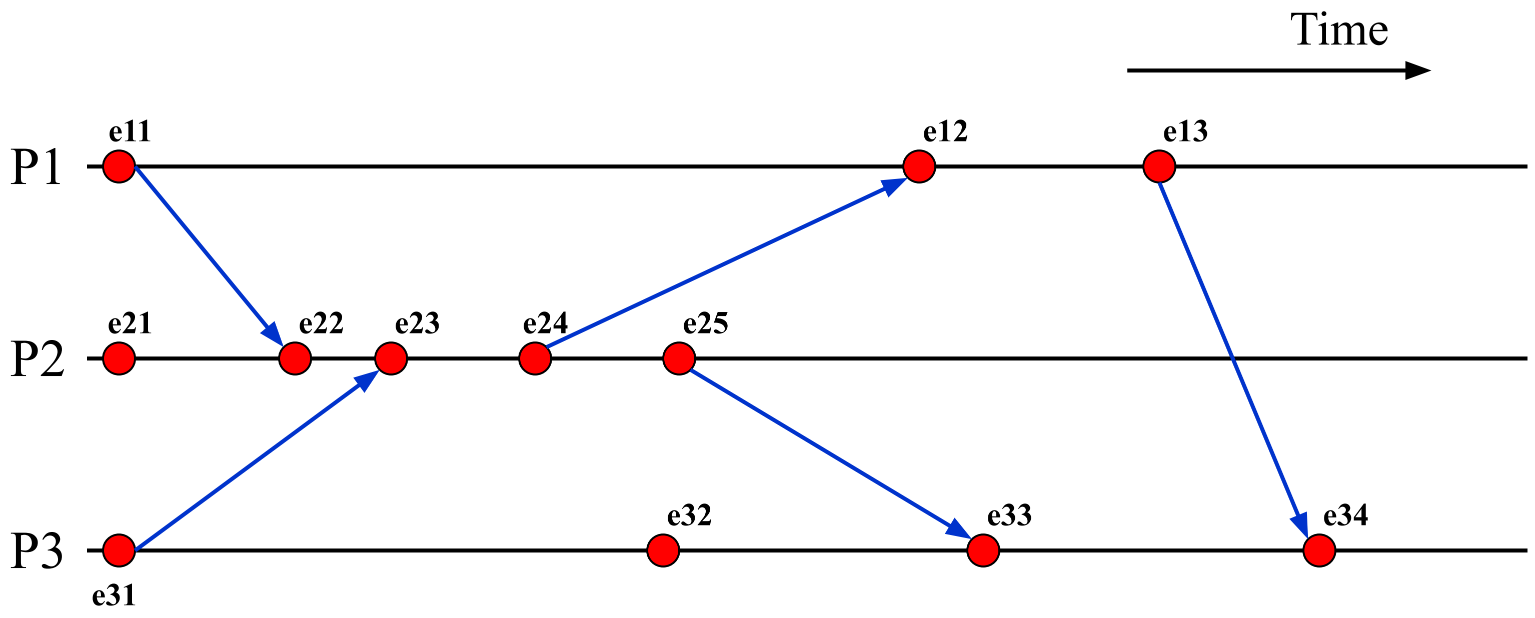
  Individual state of *each process* in the distributed system

  +

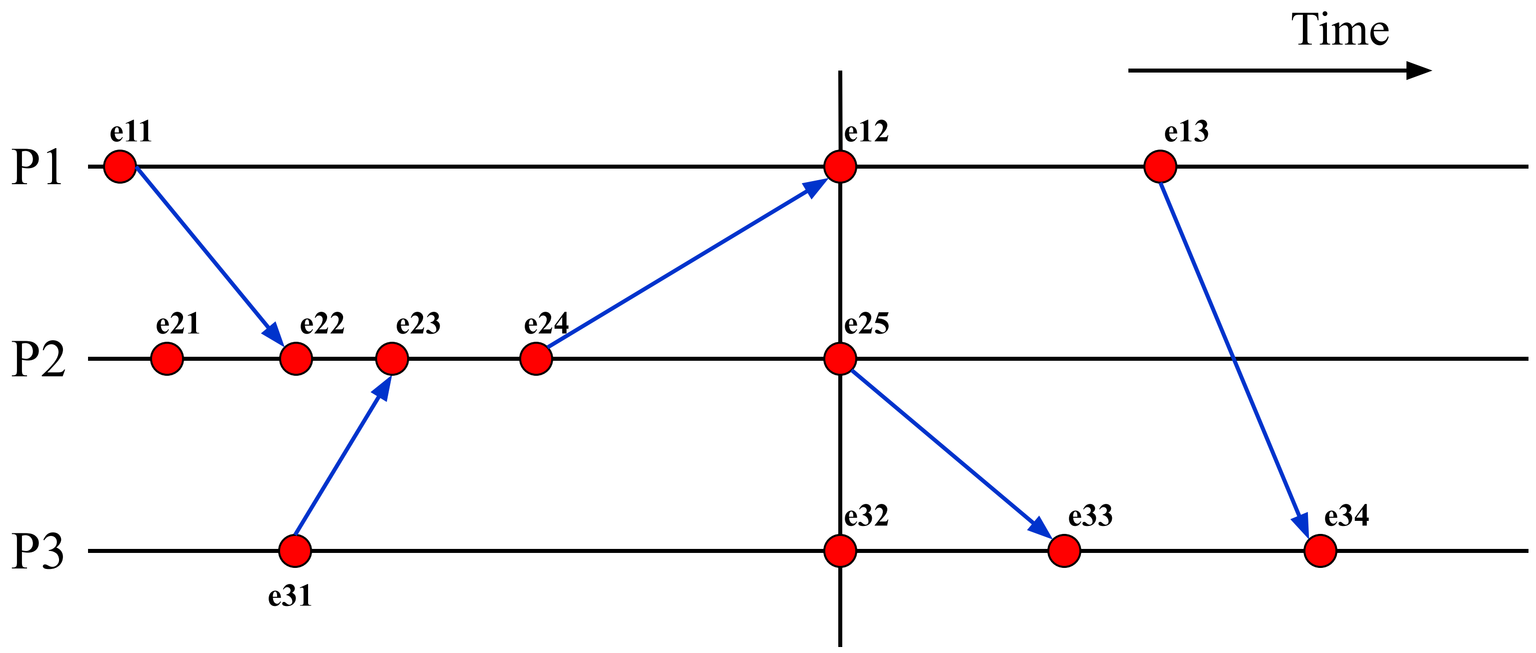
  Individual state of *each communication channel* in the distributed system

- Capture the instantaneous *state* of <u>each process</u>
- Capture the instantaneous *state* of <u>each communication channel</u>, i.e., *messages* in transit on the channels

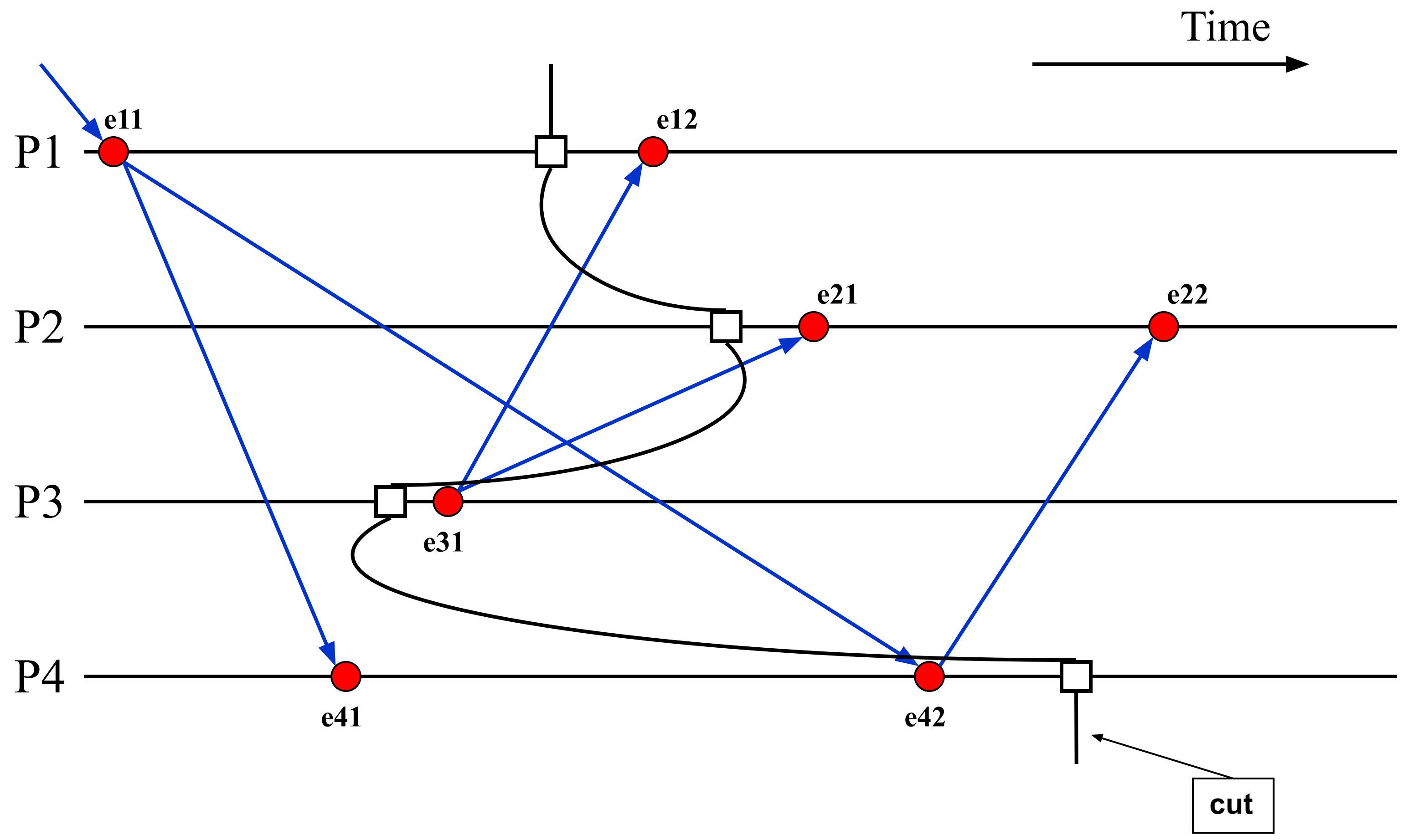
Event Diagram



Equivalent Event Diagram



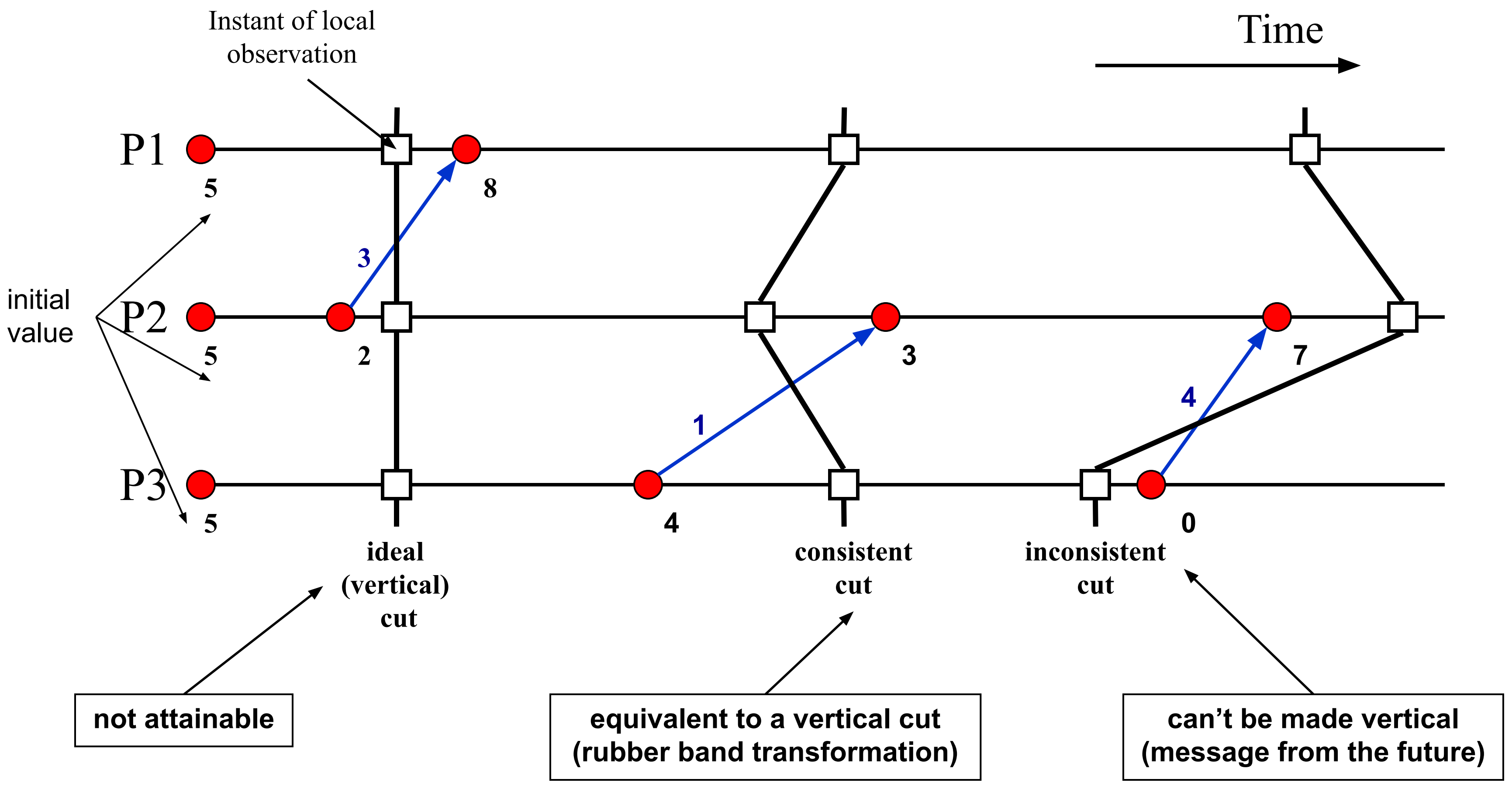
Rubber Band Transformation



Consistent Cuts

- A cut (or time slice) is a zigzag line cutting a time diagram into 2 parts (past and future)
  - E is augmented with a cut event c_i for each process P_i: $E' = E \cup \{c_i, \ldots, c_n\}$ ∴
    - A cut C of an event set E is a finite subset $C \subseteq E$: $e \in C \wedge e' <_l e \rightarrow e' \in C$
    - A cut C_1 is later than C_2 if $C_1 \supseteq C_2$
    - A consistent cut C of an event set E is a finite subset $C \subseteq E$: $e \in C \wedge e' < e \rightarrow e' \in C$
      - i.e. a cut is consistent if every message received was previously sent (but not necessarily vice versa!)

Cuts (Summary)


Instant of local observation
Time
P1
P2
P3
initial value
5
8
3
5
2
3
1
7
4
5
4
0
ideal (vertical) cut
consistent cut
inconsistent cut
not attainable
equivalent to a vertical cut (rubber band transformation)
can't be made vertical (message from the future)

Consistent Cuts

- Some Theorems
  - For a consistent cut consisting of cut events $c_i, \ldots, c_n$, no pair of cut events is causally related. i.e $\forall c_i, c_j \sim(c_i < c_j) \wedge \sim(c_j < c_i)$
  - **For any time diagram with a consistent cut consisting of cut events $c_i, \ldots, c_n$, there is an equivalent time diagram where $c_i, \ldots, c_n$ occur simultaneously. i.e. where the cut line forms a straight vertical line**
    - **All cut events of a consistent cut *can occur simultaneously***

Global States of Consistent Cuts

- The global state of a distributed system is a collection of the local states of the processes and the channels.
- A *global state* computed along a consistent cut is correct
- The *global state* of a consistent cut comprises the local state of each process at the time the cut event happens and the set of all messages sent but not yet received
- The *snapshot problem* consists in designing an efficient protocol which yields only consistent cuts and to collect the local state information
  - Messages crossing the cut must be captured
  - Chandy & Lamport presented an algorithm assuming that message transmission is FIFO

System Model for Global Snapshots

- The system consists of a collection of n processes p1, p2, ..., pn that are connected by channels.
- There are no globally shared memory and physical global clock and processes communicate by passing messages through communication channels.
- C_{ij} denotes the channel from process pi to process pj and its state is denoted by SC_{ij} .
- The actions performed by a process are modeled as three types of events:
  - Internal events,the message send event and the message receive event.
  - For a message mij that is sent by process pi to process pj , let send(m_{ij}) and rec(m_{ij}) denote its send and receive events.

Process States and Messages in transit

- At any instant, the state of process pi , denoted by LSi , is a result of the sequence of all the events executed by pi till that instant.
- For an event e and a process state LSi , e∈LSi iff e belongs to the sequence of events that have taken process pi to state LSi .
- For an event e and a process state LSi , e (not in) LSi iff e does not belong to the sequence of events that have taken process pi to state LSi .
- For a channel Cij , the following set of messages can be defined based on the local states of the processes pi and pj

  Transit: transit(LSi , LSj) = {mij |send(mij) ∈ LSi V
  rec(mij) (not in) LSj }

Distributed Global Snapshot: Requirements

- Snapshot should not interfere with normal application actions, and it should not require application to stop sending messages
- Each process is able to record its own state
  - Process state: Application-defined state or, in the worst case:
  - its heap, registers, program counter, code, etc. (essentially the coredump)
- Global state is collected in a distributed manner
- Any process may initiate the snapshot
  - Assume just one snapshot run for now

Chandy-Lamport Distributed Snapshot Algorithm

- Assumes FIFO communication in channels
- Uses a control message, called a marker to separate messages in the channels.
  - After a site has recorded its snapshot, it sends a marker, along all of its outgoing channels before sending out any more messages.
  - The marker separates the messages in the channel into those to be included in the snapshot from those not to be recorded in the snapshot.
- A process must record its snapshot no later than when it receives a marker on any of its incoming channels.
- The algorithm terminates after each process has received a marker on all of its incoming channels.
- All the local snapshots get disseminated to all other processes and all the processes can determine the global state.

Chandy-Lamport Distributed Snapshot Algorithm

```
Marker receiving rule for Process Pi
  If (Pi has not yet recorded its state) it
      records its process state now
      records the state of c as the empty set
      turns on recording of messages arriving over other channels
  else
      Pi records the state of c as the set of messages received over c
      since it saved its state
```

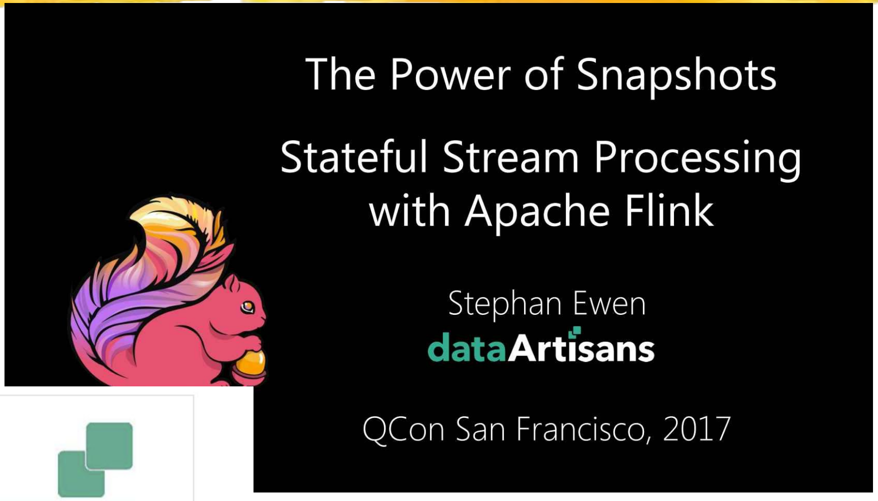
```
Marker sending rule for Process Pi
  After Pi has recorded its state, for each outgoing channel c:
      Pi sends one marker message over c
            (before it sends any other message over c)
```

Computing Global States without FIFO Assumption

- In a non-FIFO system, a marker cannot be used to delineate messages into those to be recorded in the global state from those not to be recorded in the global state.
- In a non-FIFO system, either some degree of inhibition or piggybacking of control information on computation messages to capture out-of-sequence messages.
- Lai-Yang Algorithm (uses coloring)
- Mattern’s Algorithm (uses vector-clocks)

Distributed Snapshots - Flink

https://www.infoq.com/presentations/distributed-stream-processing-flink/

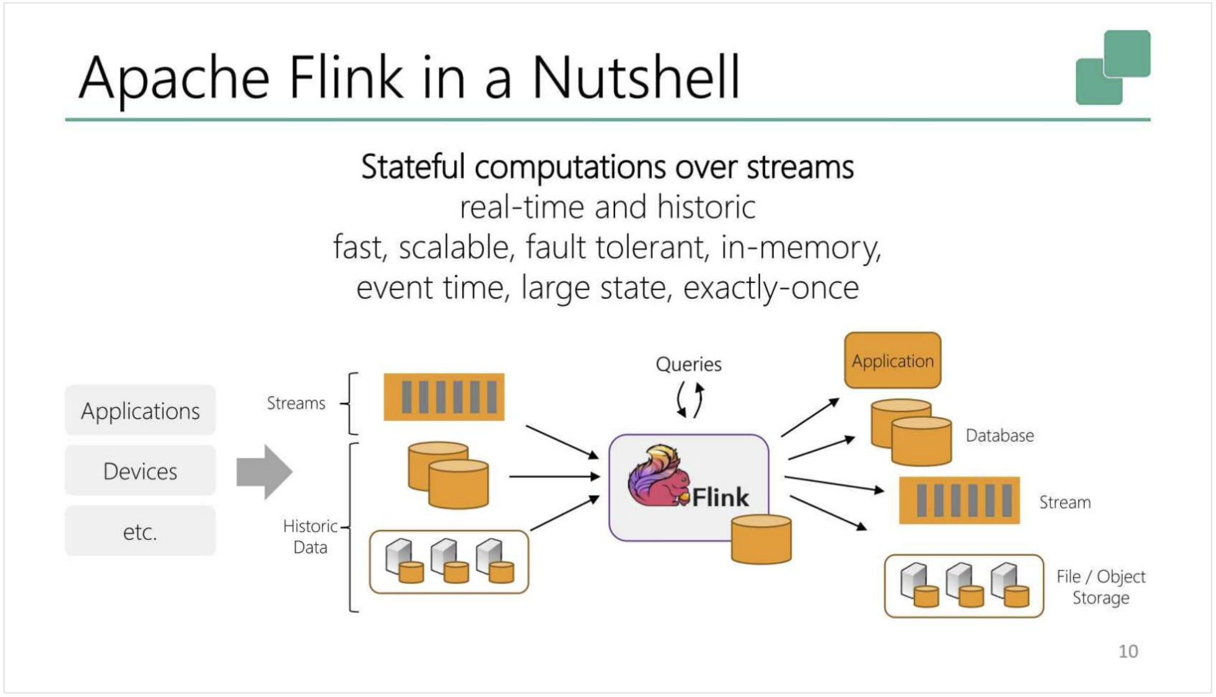

The Power of Snapshots

Stateful Stream Processing with Apache Flink

Stephan Ewen

dataArtisans

QCon San Francisco, 2017

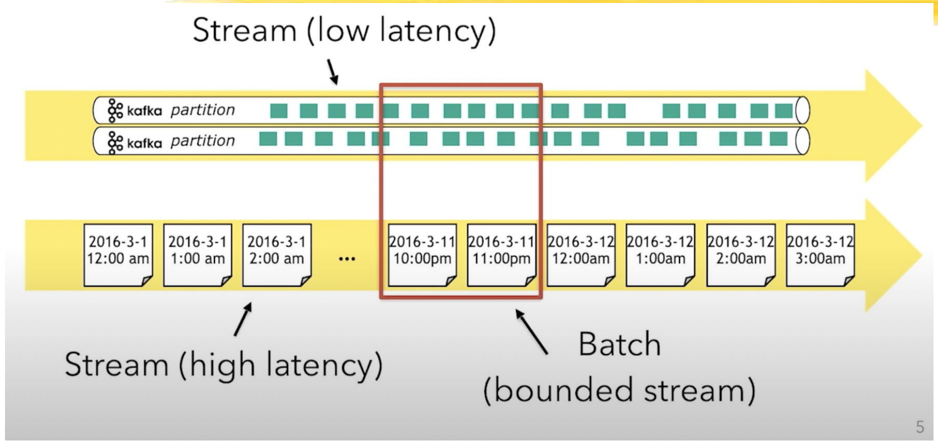

Apache Flink in a Nutshell

Stateful computations over streams

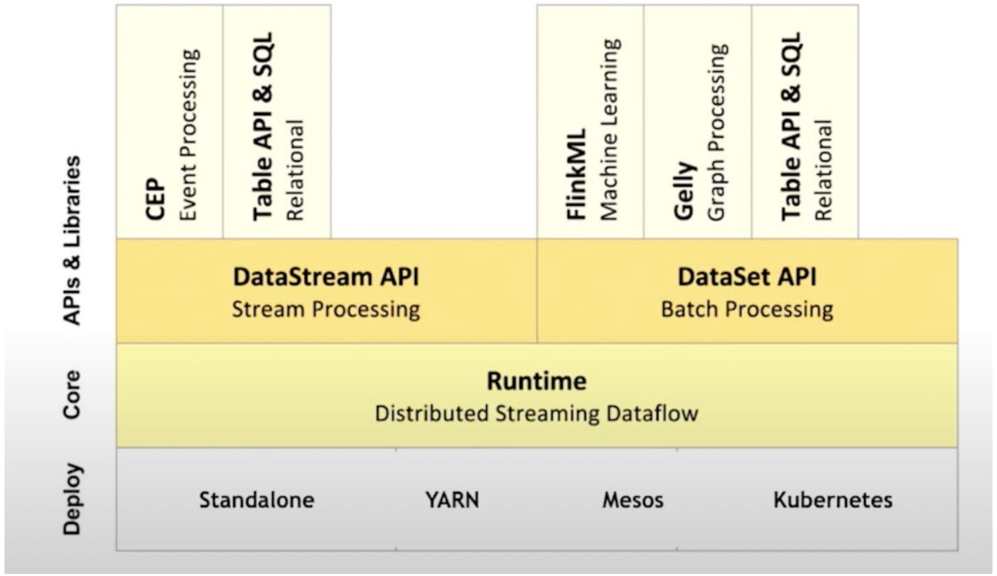
real-time and historic

fast, scalable, fault tolerant, in-memory, event time, large state, exactly-once

Apache Flink



(Streaming + Batch Workloads)



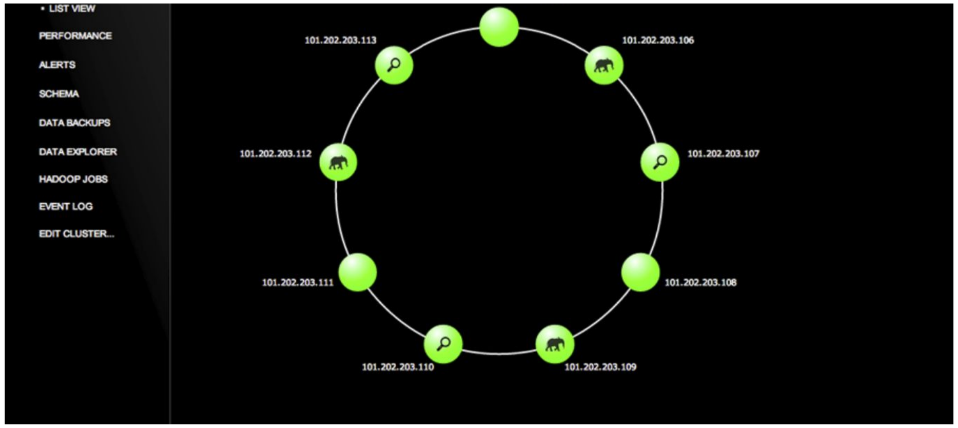
APIs & Libraries
Core
Deploy
CEP Event Processing
Table API & SQL Relational
FlinkML Machine Learning
Gelly Graph Processing
Table API & SQL Relational
DataStream API Stream Processing
DataSet API Batch Processing
Runtime Distributed Streaming Dataflow
Standalone
YARN
Mesos
Kubernetes

(Flink Component Stack)

Snapshots with operator graphs in Apache Flink

Snapshots: Cassandra Clusters

Clusters use snapshots for fault tolerance



One of our customers was running a 17 node cassandra cluster (DSE) on AWC EC2 and we had to come up with a backup strategy for it. The customer was running a fairly large environment with multiple application servers running across availability zones and we also had to automate EBS snapshots for them.

DSE was deployed using the default placement strategy(NetworkTopologyStrategy) with a replication factor of `2`. While we automated EBS snapshots for all the instances using awesome script available at AWS missing tools and it is a multi datacenter deployment with replication enabled, we can neither rely on volume snapshots for transactionally consistent backups nor can replication take care of

Cassandra Snapshot, Clone, Time-travel

- Take unlimited cluster snapshots
- Restore or refresh a cluster to any point-in-time using snapshots

Robin Hyper-Converged Kubernetes Platform provides out of the box support for application time travel. Cluster level distributed snapshots at pre-defined intervals can be really useful to restore the entire pipeline or parts of it if anything goes wrong.

Robin Systems recommends admins to take snapshots before making any major changes. Whether you are upgrading the software version or making a configuration change make sure to have a snapshot. If anything goes wrong the entire cluster can be restored to the last known snapshot in a

More Snapshot Algorithms

Computing Global States without FIFO Assumption - Lai-Yang Algorithm

- Uses a ***coloring*** scheme that works as follows
  - White (before snapshot); Red (after snapshot)
  - Every process is initially white and turns red while taking a snapshot. The equivalent of the "Marker Sending Rule" (virtual broadcast) is executed when a process turns red.
  - Every message sent by a white (red) process is colored white (red).
  - Thus, a white (red) message is a message that was sent before (after) the sender of that message recorded its local snapshot.
  - Every white process takes its snapshot at its convenience, but no later than the instant it receives a red message.

Computing Global States without FIFO Assumption - Lai-Yang Algorithm (cont.)

- Every white process records a history of all white messages sent or received by it along each channel.
- When a process turns red, it sends these histories along with its snapshot to the initiator process that collects the global snapshot.
- Determining Messages in transit (i.e. White messages received by red process)
  - The initiator process evaluates transit(LSi, LSj) to compute the state of a channel Cij as given below:
  - SCij = {white messages sent by pi on Cij –
    white messages received by pj on Cij}
  - = { send (Mij)|send(mij)$\in$LSi} – {rec(mij)| rec(mij)$\in$LSj}.

Computing Global States without FIFO Assumption: Termination

- First method
  - Each process I keeps a counter cntri that indicates the difference between the number of white messages it has sent and received before recording its snapshot, i.e number of messages still in transit.
  - It reports this value to the initiator along with its snapshot and forwards all white messages, it receives henceforth, to the initiator.
  - Snapshot collection terminates when the initiator has received Σi cntri number of forwarded white messages.
- Second method
  - Each red message sent by a process piggybacks the value of the number of white messages sent on that channel before the local state recording. Each process keeps a counter for the number of white messages received on each channel.
  - Termination – Process receives as many white messages on each channel as the value piggybacked on red messages received on that channel.

Computing Global States without FIFO Assumption: Mattern's Algorithm

- Uses Vector Clocks
  - All process agree on some future virtual time *s* or a set of virtual time instants $s_1, \ldots s_n$ which are mutually concurrent and did not yet occur
  - A process takes its local snapshot at virtual time *s*
  - After time *s* the local snapshots are collected to construct a global snapshot
    - P_i ticks and then fixes its next time $s = C_i + (0,\ldots,0,1,0,\ldots,0)$ to be the common snapshot time
    - P_i broadcasts *s*
    - P_i blocks waiting for all the acknowledgements
    - P_i ticks again (setting $C_i = s$), takes its snapshot and broadcast a dummy message (i.e. force everybody else to advance their clocks to a value $\geq$ s)
    - Each process takes its snapshot and sends it to P_i when its local clock becomes $\geq$ s

Computing Global States without FIFO Assumption (Mattern cont)

- Inventing a n+1 virtual process whose clock is managed by P_i
- P_i can use its clock and because the virtual clock C_{n+1} ticks only when P_i initiates a new run of snapshot :
  - The first n components of the vector can be omitted
  - The first broadcast phase is unnecessary
  - Counter modulo 2
- Termination
  - Distributed termination detection algorithm [Mattern 87]