Time, Logical Time and Causality

Carlos Baquero
Distributed Systems Group
Universidade do Minho

MAPI 2007

Plan

We will try to cover a few of the many aspects of time and logical sequences of events in distributed systems:

- Time Synchronization
- Order Relations
- Logical Time and Causality
- Process Causality vs Data Causality

Global Snapshots and Termination will only be covered in the next talk, so we will carefully avoid them.

Time Synchronization
Time, which time?

Non relativistic real time can be tracked by clocks. But clocks have drift. Where drift is the variation between a clock's time and a reference clock.

- Quartz clocks drift at about 10^{-6} to 10^{-8} seconds per second.
- 10^{-6} amounts to about 1 second each 12 days. Not very good.
- Atomic clocks drift at about 10^{13} seconds per second.
- Coordinated Universal Time (UTC) is a high-precision atomic time standard. It closely tracks Universal Time (UT), that maps earth rotation, by adding leap seconds when needed.

Time Synchronization

Synchronization

External Synchronization

Measures expected precision with reference to an authoritative time source.
For an envelope $D > 0$, a UTC source S and at any given instant t we need to have $|S(t) - C_i(t)| \leq D$.

Internal Synchronization

Measures synchronization between two machines.
For an envelope $D > 0$, at any given instant t we need to have $|C_j(t) - C_i(t)| \leq D$.

A system with D external synchronization also depicts $2D$ internal synchronization.

Time Synchronization
Synchronous System

Consider the simple case of two node synchronization in a synchronous setting.

- Node C asks node S the time. S replies with time t and node C knows the transit time t_d. C can set its time to $t + t_d$.
- Tipically t_d varies in a range, $t_m \leq t_d \leq t_M$. Leading to a variation range of $t_M - t_m$.
- If we set in C time to $t + \frac{T_M - t_m}{2}$ one can achieve synchronization within an envelope D of $\frac{T_M - t_m}{2}$.

Asynchronous

In an asynchronous system t_d now varies in range $t_m \leq t_d \leq \infty$. Apparently, the envelope is now $D = \frac{\infty - t_m}{2} = \infty$. Not a very usefull bound, but its easy to do better.

Time Synchronization
Asynchronous System: Cristian's algorithm

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Two node synchronization in an asynchronous setting.

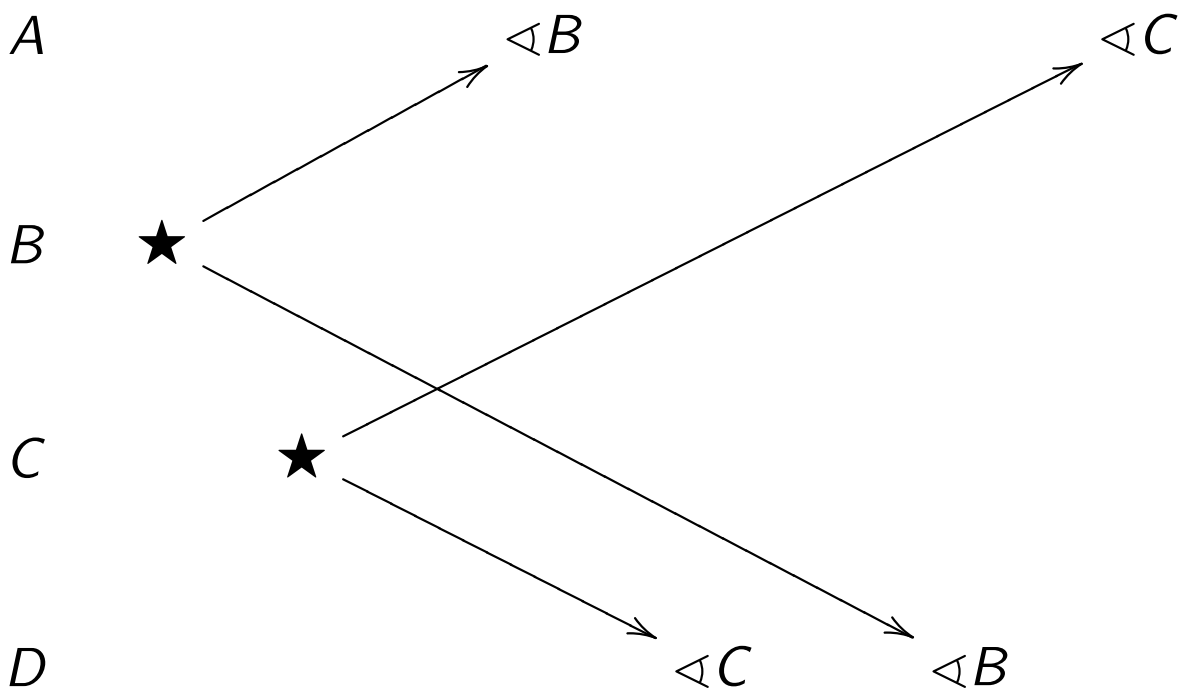
- Node C memorizes time $t_i = t$ asks node S the time. S replies with time t_s and node C memorizes the reception time $t_f = t$.
- Node C calculates the roundtrip time as $t_r = t_f - t_i$.
- C can set the time to $t_s + \frac{t_r}{2}$ and expect to have a synchronization of $D = \frac{t_r}{2}$.
- t_r can be made smaller if we adjust for a lower bound b on message transmition time. $t_r = t_f - (t_i + b)$.
- The algorithm can be repeated until we eventually observe a t_r that gives us a "tight enough" synchronization.

Time Synchronization
Summary

- Both in synchronous and asynchronous settings one can expect at most time synchronization in an envelope D. Synchronization can be usefull to coordinate access to shared channels; either to avoid two senders at the same time or to make shure that sender and receiver are both awake.
- With enough timing resolution, tight envelopes, and slow computation steps (or slow processors) one could expect to tottaly order a distributed computation.
  The resulting total order is not realistic and not always usefull, since it orders events that are in fact unrelated.
- Even on physical systems real time total ordering is not always consistent for diferent observers.

Time Synchronization
Ordering Explosions: Two independent ones

Carlos Baquero
Distributed Systems Group
Universidade do Minho



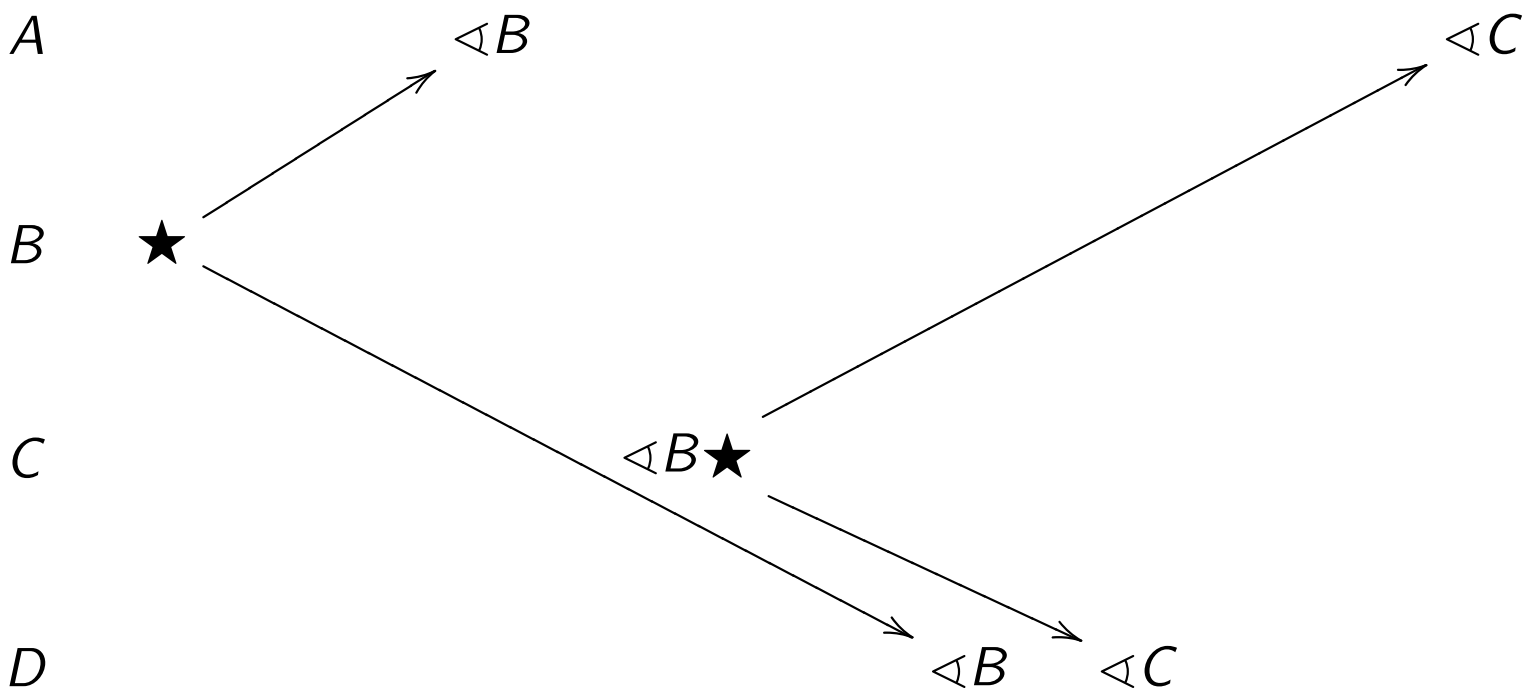
Observers

While A sees $\langle B, C \rangle$, D sees $\langle C, B \rangle$.

If we really need a total order (e.g. to make a replicated state machine) maybe we can give an arbitrary order to these events. As long as no one can contradict these decisions.

Time Synchronization
Ordering Explosions: One triggers the next

Carlos Baquero
Distributed Systems Group
Universidade do Minho



Observers

Now, both A and D see $\langle B, C \rangle$.

If message propagation speed is uniform, independent observers make consistent observations of events that might be causaly related. Otherwise the world would be much more confusing ...

Order Relations

Before digging deeper into order in distributed systems lets review some notions of order relations.

Order

- Concerns the comparison between pairs of objects.
- Is a binary relation on a set of objects. In order $\langle B, <_B \rangle$ we have $<_B \subseteq B \times B$.
- Order is transitive. $a < b \wedge b < c \Rightarrow a < c$.
- Order is antisymmetric. $a < b \Rightarrow b \not< a$.

If we miss *antisymmetry* we only have a **preorder**.
Orders can be strict $<$ or non-strict $\leq$.

Order Relations

Non-strict order (or non strict partial order)

Let B be a set and $\leq$ a binary relation on B such that, for all $x, y, z \in B$:

reflexivity $x \leq x$.

antisymmetry $x \leq y$ and $y \leq x$ imply $x = y$.

transitivity $x \leq y$ and $y \leq z$ imply $x \leq z$.

In a **preorder** we can have $x \neq y$ and $x \leq y \wedge y \leq x$.
One also writes $x \parallel y$ to mean $x \not\leq y \wedge y \not\leq x$.

Chains and antichains

- If for all $x, y \in B$ either $x \leq y$ or $y \leq x$ we have a **chain**. Also known as **total order**, where all elements are comparable.
- We have an **antichain** if $x \leq y$ iff $x = y$.

Order Relations
Examples

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Sets

A set X can be ordered by set inclusion, yielding $\langle X, \subseteq \rangle$.
The powerset $\mathcal{P}(X)$, consisting of all subsets of X, is ordered by set inclusion.
Q: Does $\subseteq$ form a total order on $\mathcal{P}(X)$?
A: No, by counter example: $\{a, x, f\} \parallel \{x, b\}$.

Binary sequences

Exhibit a **prefix** ordering. Let $\mathbf{2}^*$ be the set of all finite binary strings, including $\langle\rangle$. For $x, y \in \mathbf{2}^*$ we have $x \leq y$ iff x is a finite initial substring of v. E.g. $0100 < 010011$, $010 \parallel 100$.

Order Relations
Examples

Coordinatewise (pointwise) order

Let $P_1, \ldots, P_n$ be ordered sets. The cartesian product $P_1 \times \cdots \times P_n$ can define a ordered set by pointwise order:

$$(x_1, \ldots, x_n) \leq (y_1, \ldots, y_n) \Leftrightarrow (\forall i) x_i \leq y_i \text{ in } P_i.$$

Lexicographic order

Let A, B be two ordered sets. The product $A \times B$ can have a **lexicographic order** defined by
$(x_1, x_2) \leq (y_1, y_2)$ if $x_1 < y_1$ or $(x_1 = y_1$ and $x_2 \leq y_2)$.
By iteration a lexicographic order can be defined on any finite product.

Order Relations
Relations among orders

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Order isomorphism

Given two partially ordered sets $\langle S, \leq_S \rangle$ and $\langle T, \leq_T \rangle$ an **order isomorphism** is a surjective (onto) total function $h : S \rightarrow T$ such that for all $u, v \in S$:
$h(u) \leq_T h(v)$ iff $u \leq_S v$.
We say that $\langle S, \leq_S \rangle$ and $\langle T, \leq_T \rangle$ are equivalente and that one **characterizes** the other and vice-versa.

A weaker form is

Order preserving

Given two partially ordered sets $\langle S, \leq_S \rangle$ and $\langle T, \leq_T \rangle$ an **order preserving** maping is a total function $h : S \rightarrow T$ such that for all $u, v \in S$:
$h(u) \leq_T h(v)$ if $u \leq_S v$.
We say that $\langle T, \leq_T \rangle$ is **consistent** with $\langle S, \leq_S \rangle$.

For instance, we will see that real time total ordering is consistent with causality.

Carlos Baquero
Distributed Systems Group
Universidade do Minho

- An asynchronous system with a collection of totally ordered processes $p_1, \ldots, p_n$.
- Reliable channels, not necessarely FIFO.
- Sequential processes, performing internal events, send events and corresponding receive events.

In each process p_i during a computation a *local history* is formed by the (potentially infinite) sequence of events: $h_i = \langle e_i^1, e_i^2, \ldots \rangle$. As expected, time between events varies.
h_i^k denotes an initial prefix of local history h_i containing the first k events.
The *global history* of the computation is the set $H = h_1 \cup \ldots \cup h_n$.

Carlos Baquero
Distributed Systems Group
Universidade do Minho

We can now define a causality relation in distributed systems.

Causality

Let $\langle H, \rightarrow \rangle$ be a global history H ordered by the smalest transitive binary relation $\rightarrow$ such that:

- $e_i^a \rightarrow e_i^b$ if $e_i^a, e_i^b \in H$ and $a < b$.
- $e_i^s \rightarrow e_j^r$ if e_i^s is a send event and e_j^r the corresponding receive event.

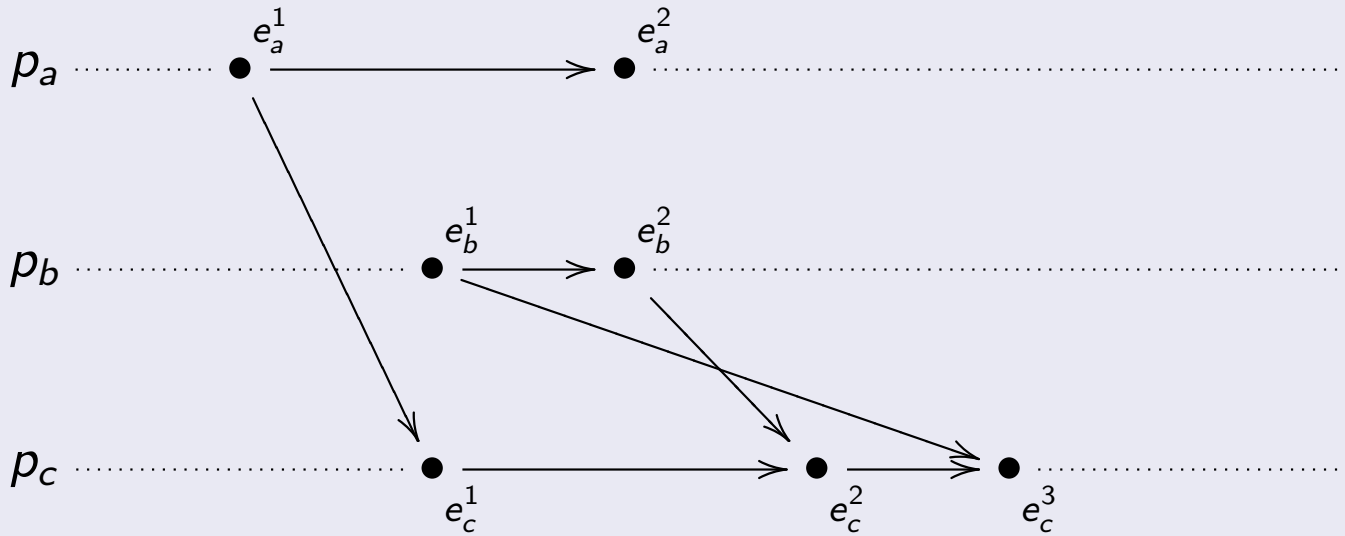
If $a \rightarrow b$ then a may have influenced b. In general we have potential causality.
On non trivial runs $\langle H, \rightarrow \rangle$ forms a partial order, and some events will be parallel $a \parallel b$ when neither $a \rightarrow b$ nor $b \rightarrow a$.

Logical Time and Causality

Preserving causal order

If we had a global time clock function $\mathcal{T} : H \rightarrow \mathbb{R}$ that would assign a real to each event. We would observe that the total order $\langle \mathcal{T}(H), < \rangle$ is consistent with $\langle H, \rightarrow \rangle$. Real time preserves the causal order.

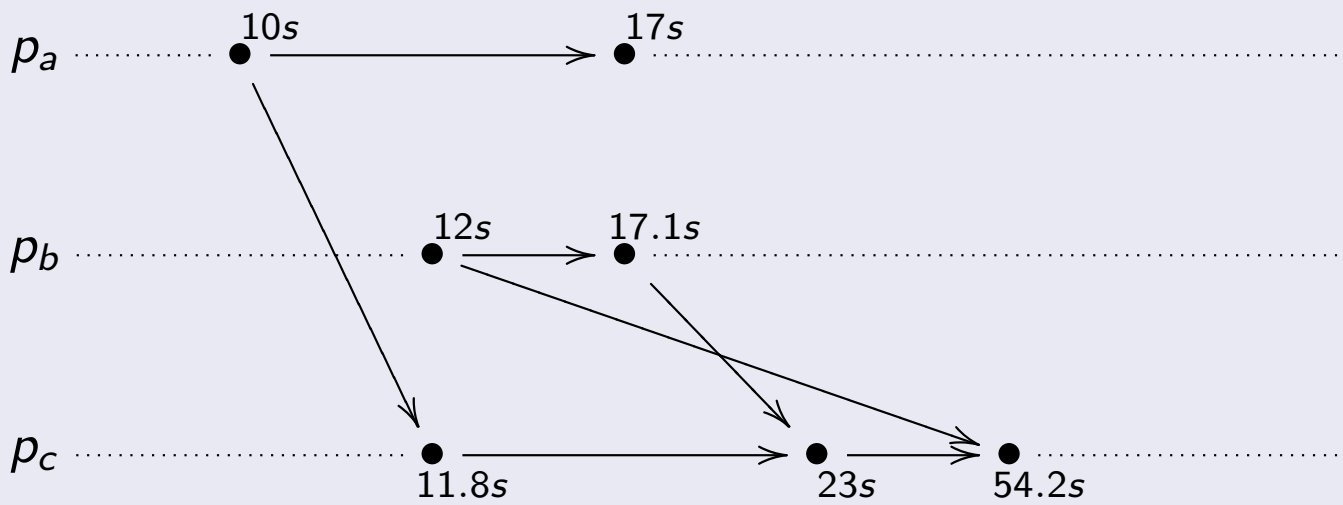
Run



Carlos Baquero
Distributed Systems Group
Universidade do Minho

If we had a global time clock function $\mathcal{T} : H \rightarrow \mathbb{R}$ that would assign a real to each event. We would observe that the total order $\langle \mathcal{T}(H), < \rangle$ is consistent with $\langle H, \rightarrow \rangle$. Real time preserves the causal order.

Run with real time tags



Notice that while $11.8s < 12s$ the corresponding events are parallel $e_c^1 \parallel e_b^1$ in the causal order.

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Being consistent with causality if often captured by a *clock condition*.

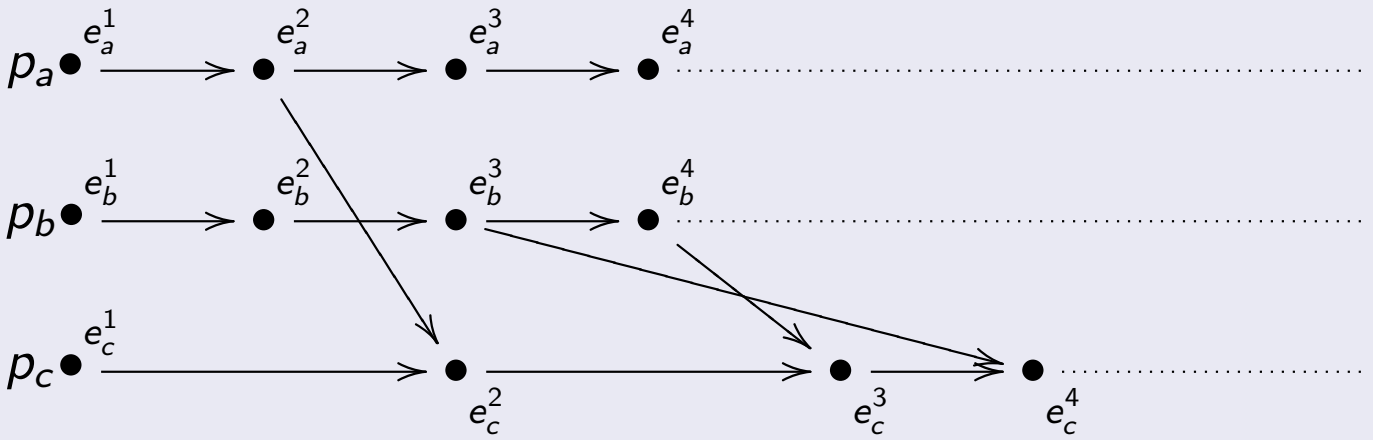
Clock Condition (Lamport 78)

A clock function $\mathcal{C} : H \rightarrow T$ and a ordered set $\langle T, < \rangle$ satisfies clock condition if:
For any events $a, b \in H$: if $a \rightarrow b$ then $\mathcal{C}(a) < \mathcal{C}(b)$.

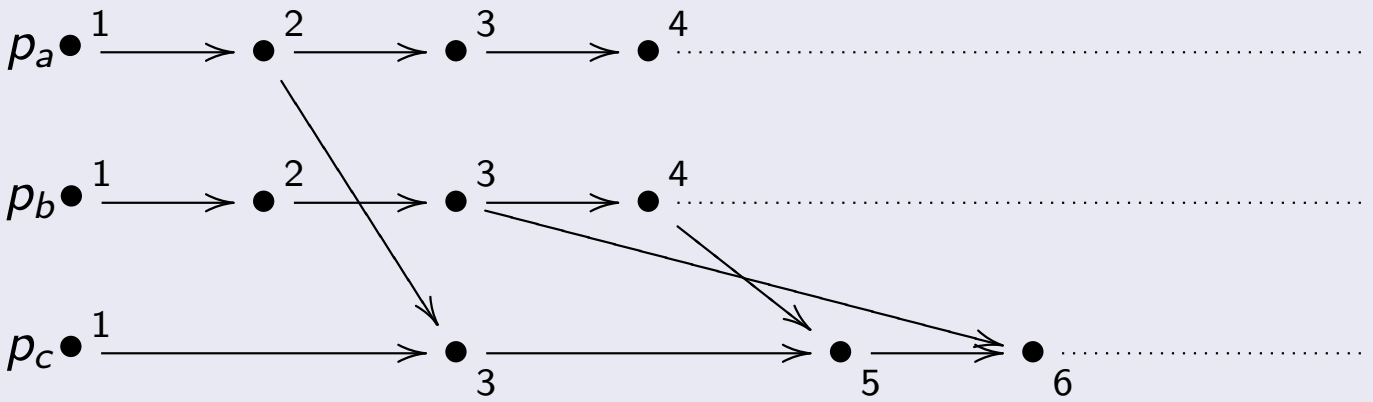
Notice that the timestamping function is necessarely one-to-one (injective) in order to satisfy the clock conditions and preserve the causal order.
Appart from real time there are other timestamping functions that satisfy this clock condition.

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Run



Run with timestamping consistent with causality



Carlos Baquero
Distributed Systems Group
Universidade do Minho

Lamport Time $\mathcal{L}$

We can assign integer valued timestamps by a function $\mathcal{L} : H \rightarrow \mathbb{N}$ constructed as follows, with local knowledge:

- Initially all processes p_i set $\mathcal{L}_i$ to 1.
- On each internal event in p_i do $\mathcal{L}_i := \mathcal{L}_i + 1$.
- On a send event at p_i do $\mathcal{L}_i := \mathcal{L}_i + 1$ and attach $\mathcal{L}_i$ to the message.
- On a receive event at p_i with $\mathcal{L}_x$ attached do $\mathcal{L}_i := max(\mathcal{L}_i, \mathcal{L}_x) + 1$.

The value registred at $\mathcal{L}_i$ right after each event e_i^k is the one defining $\mathcal{L}(e_i^k)$. A positive integer could be used in place of 1.

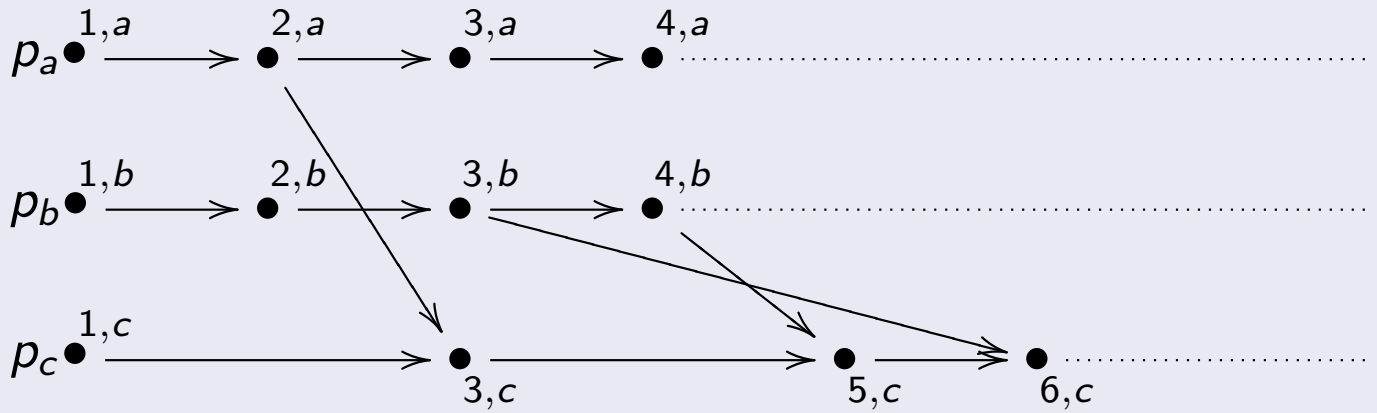
Notice that while Lamport Time $\mathcal{L}$ and Real Time $\mathcal{L}$ are both consistent with causality $\langle H, \rightarrow \rangle$, the mutual relation between $\mathcal{L}$ and $\mathcal{T}$ is not tipically consistent in non trivial runs.

Carlos Baquero
Distributed Systems Group
Universidade do Minho

We can further refine Lamport Time in order to obtain an injective function $\mathcal{L}^t$ that assigns a consistent total order to all events in H. It suffices to consider the lexicographic order on the pair formed by the Lamport Time and the process number.

Run with total order $\mathcal{L}^t$

Here, since processes have letters we assume the alphabetic order.



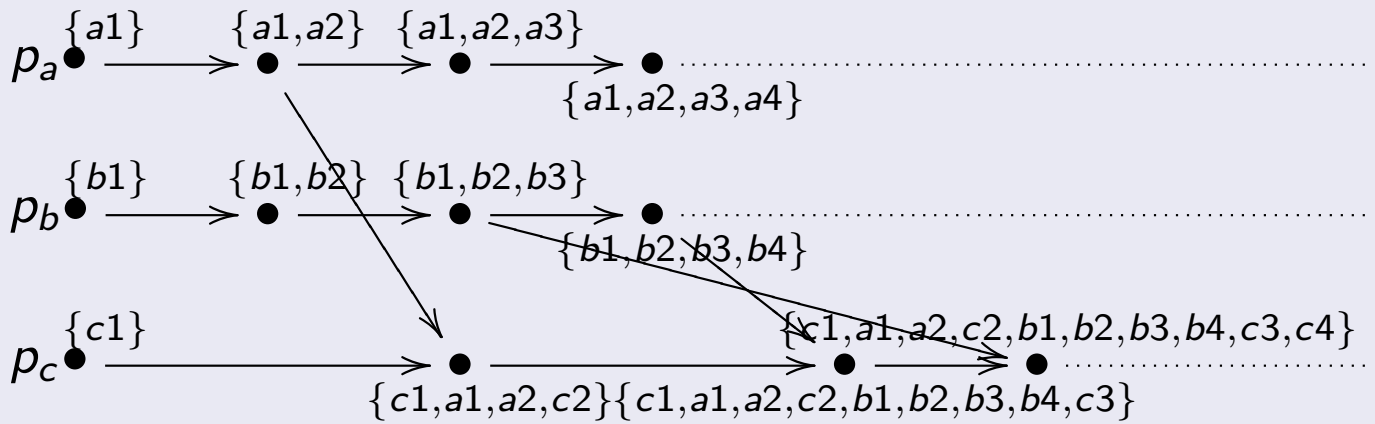
$(1,a)(1,b)(1,c)(2,a)(2,b)(3,a)(3,b)(3,c)(4,a)(4,b)(5,c)(6,c)$

This total order is usefull in many distributed algorithms (e.g. Lamport mutual exclusion algorithm), but it orders more events than causality. For other algorithms we need to capture causality precisely.

Carlos Baquero
Distributed Systems Group
Universidade do Minho

A simple timestamping mechanism that can characterize causality is to locally register the causal history $\mathcal{C} : H \rightarrow \mathcal{P}(H)$. This is done by collecting in a set each distinct event identifier. Notice that each process has a unique number and can maintain a sequential counter for its events.

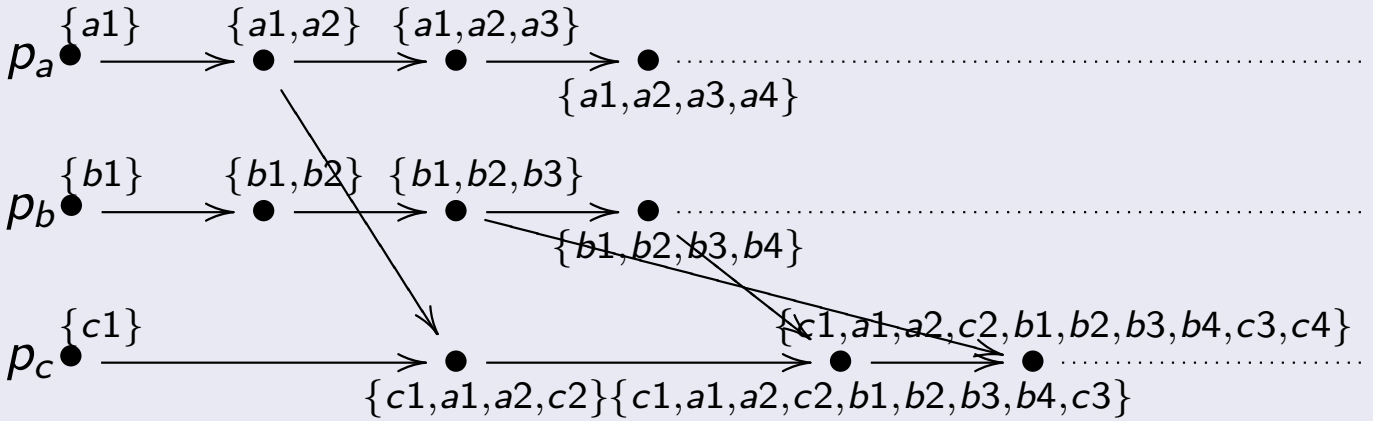
Run tagged with causal histories



$$e_a^2 \rightarrow e_c^2 \Leftrightarrow \mathcal{C}(e_a^2) \subseteq \mathcal{C}(e_c^2) \Leftrightarrow \{a1, a2\} \subseteq \{c1, a1, a2, c2\}$$
$$e_a^3 \not\rightarrow e_b^3 \Leftrightarrow \mathcal{C}(e_a^3) \not\subseteq \mathcal{C}(e_b^3) \Leftrightarrow \{a1, a2, a3\} \not\subseteq \{b1, b2, b3\}$$
$$e_b^3 \not\rightarrow e_a^3 \Leftrightarrow \mathcal{C}(e_b^3) \not\subseteq \mathcal{C}(e_a^3) \Leftrightarrow \{b1, b2, b3\} \not\subseteq \{a1, a2, a3\}$$

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Run tagged with causal histories

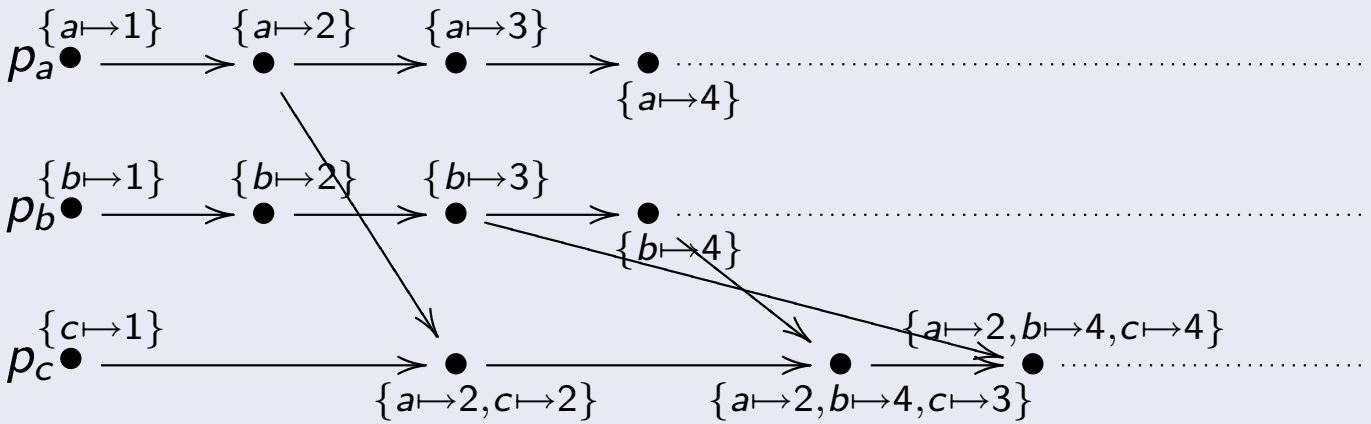


The problem of causal histories is their space complexity that grows linearly, $O(E)$, with the number of events E.
This can be solved by noticing that for all k and i and a causal history $\mathcal{C}_x$: if $e_i^k \in \mathcal{C}_x$ then $\{e_i^1, \ldots, e_i^{k-1}\} \subseteq \mathcal{C}_x$.
Consequently one only needs to register the index of the last event from each process.

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Vector clocks are compressed causal histories. $\mathcal{V} : H \rightarrow \mathbb{N}^n$ where n is the number of processes. They can be represented in a vector or as mappings from process names to integers.

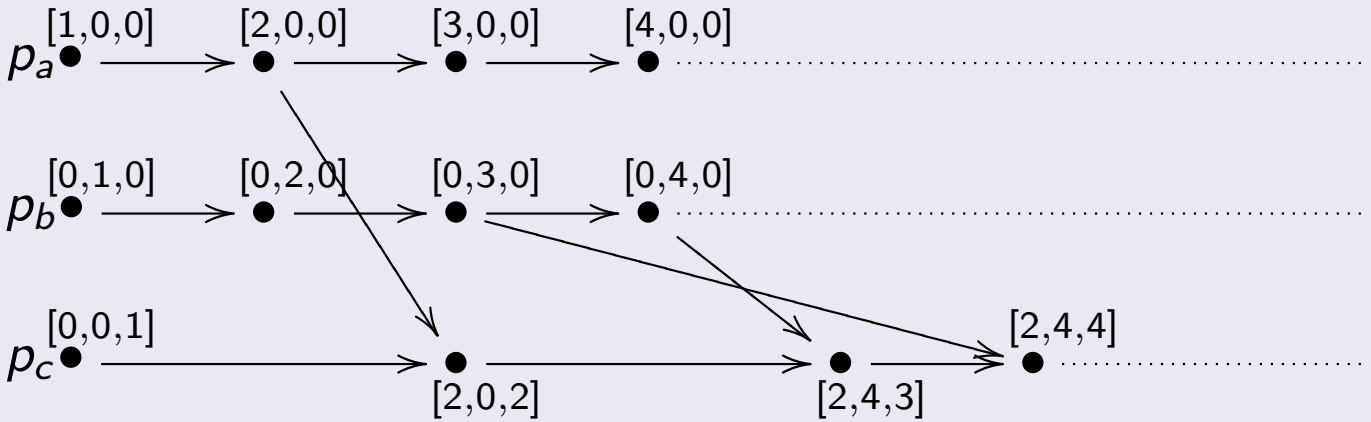
Run tagged with vector clocks



Vector clocks are used in many distributed algorithms. E.g. causal delivery of messages, an extension of FIFO delivery. They can be used as long as processes have unique ids. A total order on ids is only a convenience (trivially obtained from unique ids).

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Run tagged with vector clocks



The cordinatewise (pointwise) order on version vectors characterizes causality. They define identical partial orders.
$[2,0,0] < [2,0,2]$ but $[0,4,0] \parallel [2,0,2]$
Complexity is $O(N \log E)$ and $\mathcal{V}$ is known to be the most consise timestamping mechanism for process causality tracking.

Process Causality vs Data Causality

Carlos Baquero
Distributed Systems Group
Universidade do Minho

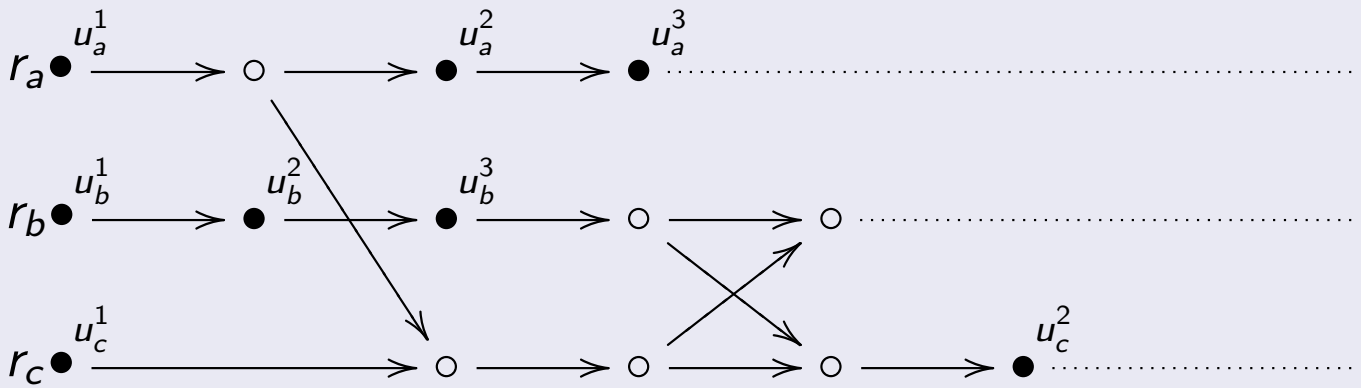
- Causality is formed as relevant events are colected in a run.
- In process causality the relevante events are *internal*, *send* and *receive* events.
- With causal histories a new event id is added in each case.
- In data causality we are concerned with the ordering of replicas subject to optimistic operation.
- The relevant events are *update* events on the replica state.
- If each update event is distinguished, two replicas that know the same set of update events are equivalente.

Data Causality

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Data causality is at the core of version control systems, replicated file systems, partitioned operation and optimistic replication in general.

Run

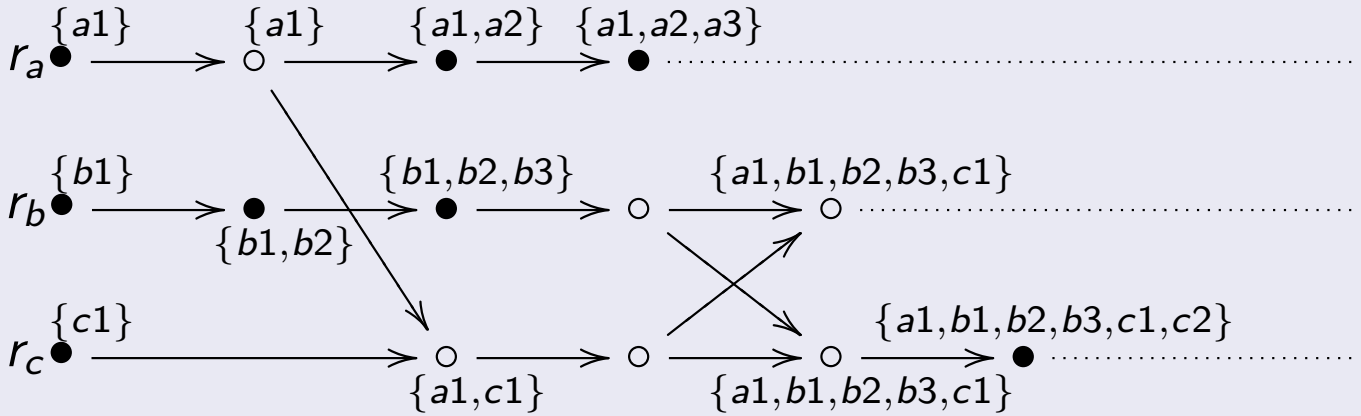


This run includes sending and receiving of messages but causality tracking can ignore these events.

Data Causality
Causal histories

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Run with causal histories

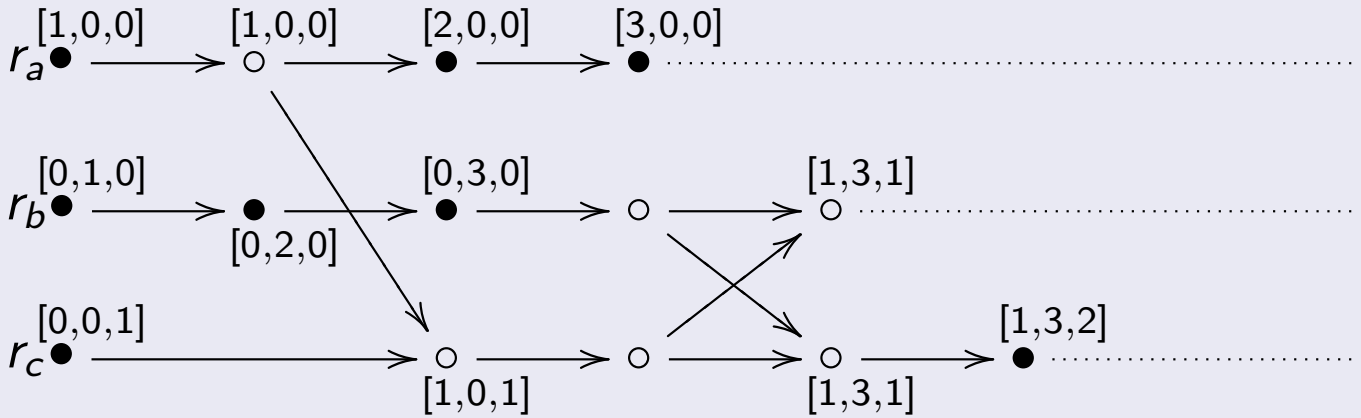


Unlike process causality, where all events depict different causal histories, here replicas can known the same set of events. In that case we say that replicas are equivalent.
At the end of this run we have the following relations among replicas, as observed by set inclusion:
$r_a \parallel r_b$ and $r_a \parallel r_c$ and $r_b < r_c$.

Data Causality
Version vectors

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Run with version vectors



Both version vectors and causal histories characterize data causality.
Are version vectors the most consise representation of data causality?
In fact, no, altough it looks like.
Altough we have a unbounded number of update events in data causality one is only concerned about the order among existing replicas. Those forming the frontier of the run.

Data Causality
Frontier configurations

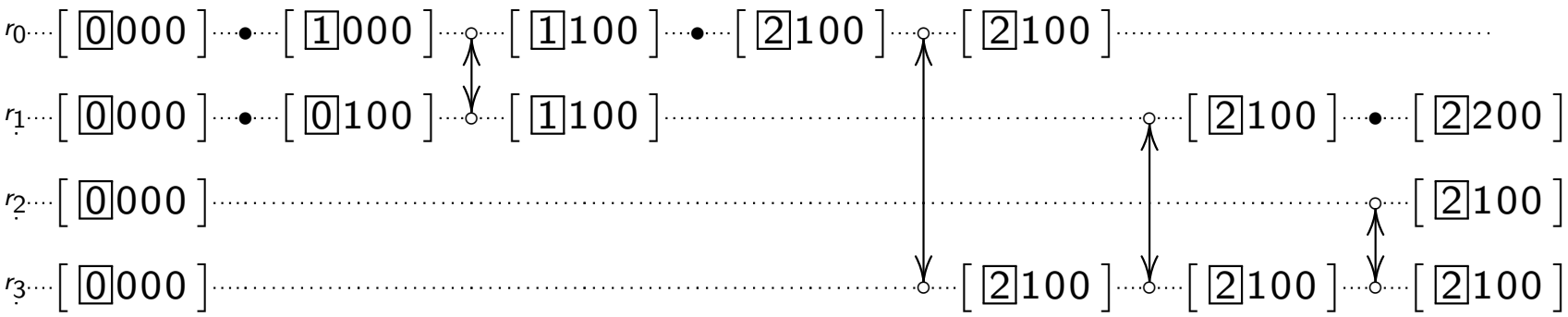
With two replicas the following cases are possible:

- $r_a = r_b$
- $r_a < r_b$
- $r_a > r_b$
- $r_a \parallel r_b$

With three replicas we have more cases, altough a finite number of them.
From "On the computer enumeration of finite topologies" they are found to be
$\{1 \mapsto 1, 2 \mapsto 4, 3 \mapsto 29, 4 \mapsto 355, 5 \mapsto 6942, 6 \mapsto 209527, 7 \mapsto 9535241\}$
Is there a local distributed algorithm that can characterize this partial order (possibly pre-order) with a bounded state?
This is possible with bounded version vectors.

Data Causality
Pointwise order

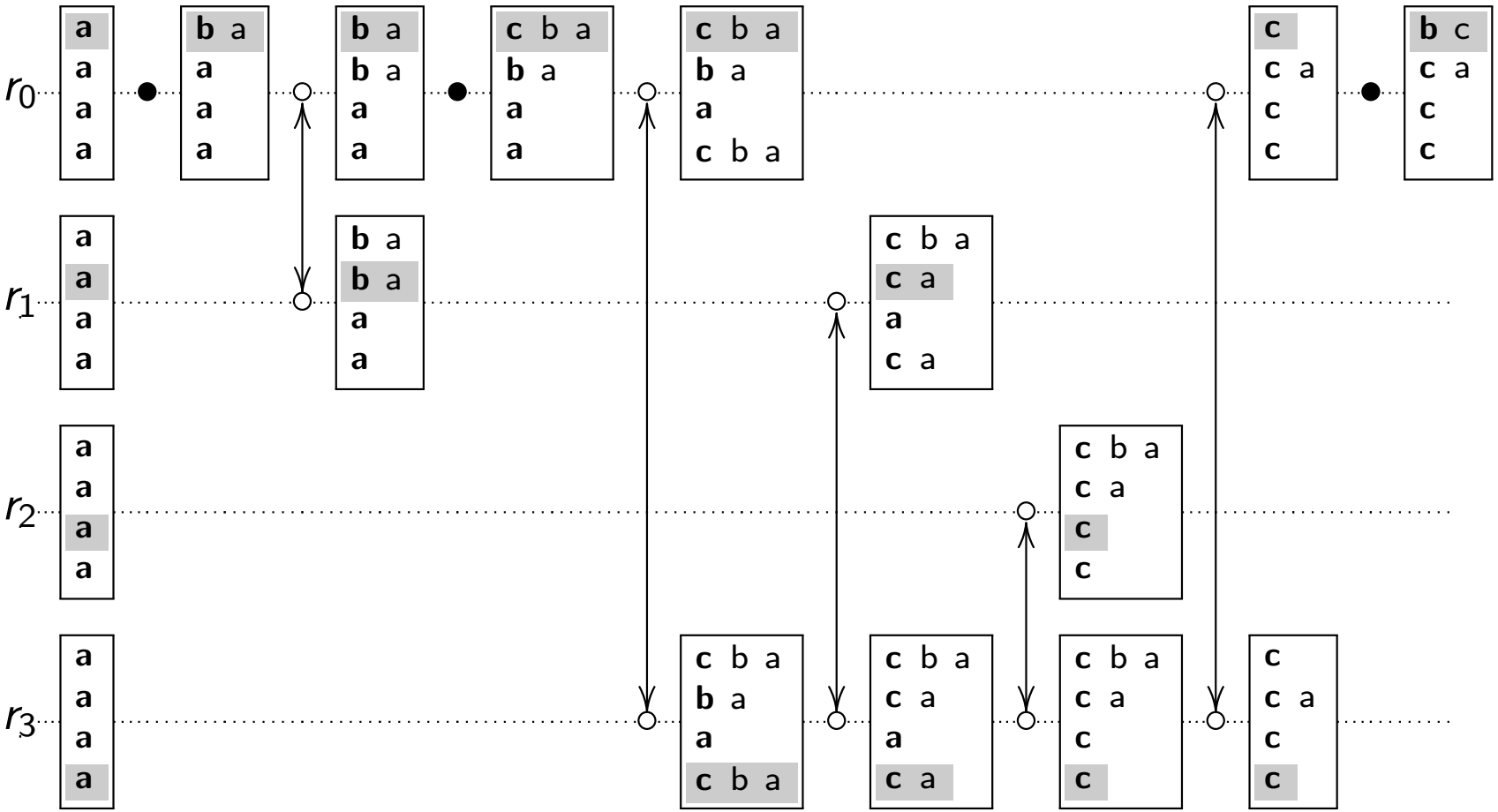
Carlos Baquero
Distributed Systems Group
Universidade do Minho



Since version vectors are define with pointwise order it is enough do find a bounded replacement for each component that defines a total order in all frontiers.

Data Causality
Bounded version vectors

Carlos Baquero
Distributed Systems Group
Universidade do Minho



Data Causality
Bounded version vectors

Bounded version vectors can characterize data causality with a state that is independent on the number of updates. The required state is polynomial with respect to the number of replicas.
Let U be the number of updates, and N the number of replicas.

- Traditional version vectors have scale $O(N \log_2(U))$
- Bounded version vectors have scale $O(N^3 \log_2(N))$

Consequently, the bounded approach can only be efficient for very small numbers of replicas or extremely high update rates.
In addition, synchronizations must be bidirectional.

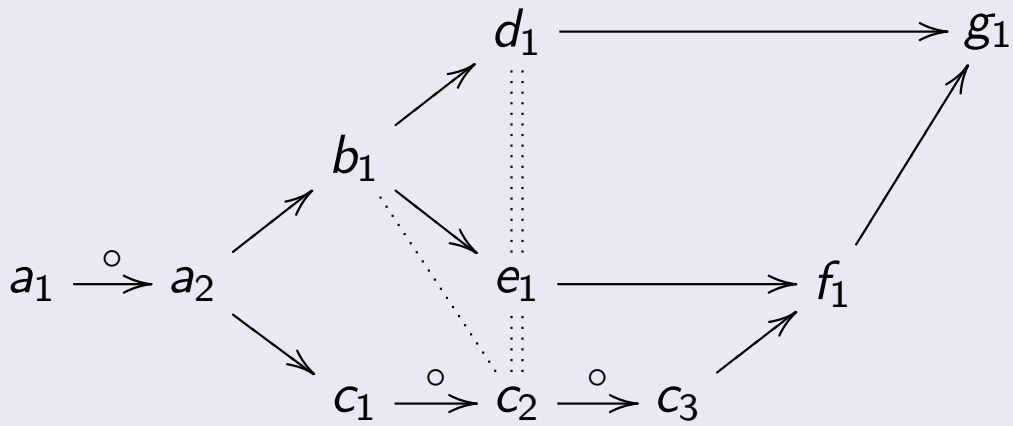
Data Causality
Dynamic Number of Replicas

Carlos Baquero
Distributed Systems Group
Universidade do Minho

The previous mechanisms assumed a known number of replicas of global naming. This might not be possible in partitioned settings, just where optimistic replication is more needed.

Setting

- Replica forking, update and synchronization.
- Variable number of replicas: variable-width frontier.
- Example: ad-hoc file copying and updating.



Carlos Baquero
Distributed Systems Group
Universidade do Minho

Identity component

- Local management of the namespace.
- Distinguishes a replica from all coexisting ones.
- Available namespace from which other replicas can be generated.

Update component

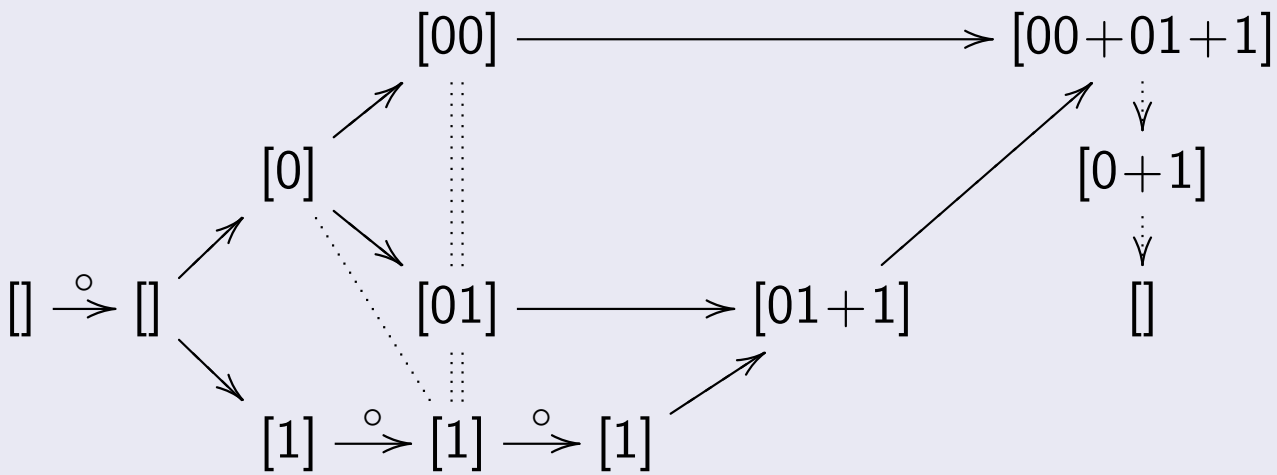
- Records when changes were applied.
- Identity-like value collected from ancestors.
- Global comparison of replicas.

Other features

- Both components are a set of binary strings.
- No map from identifiers to counters is kept.
- No counters are used at all.
- Structure can grow and shrink.

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Identity component

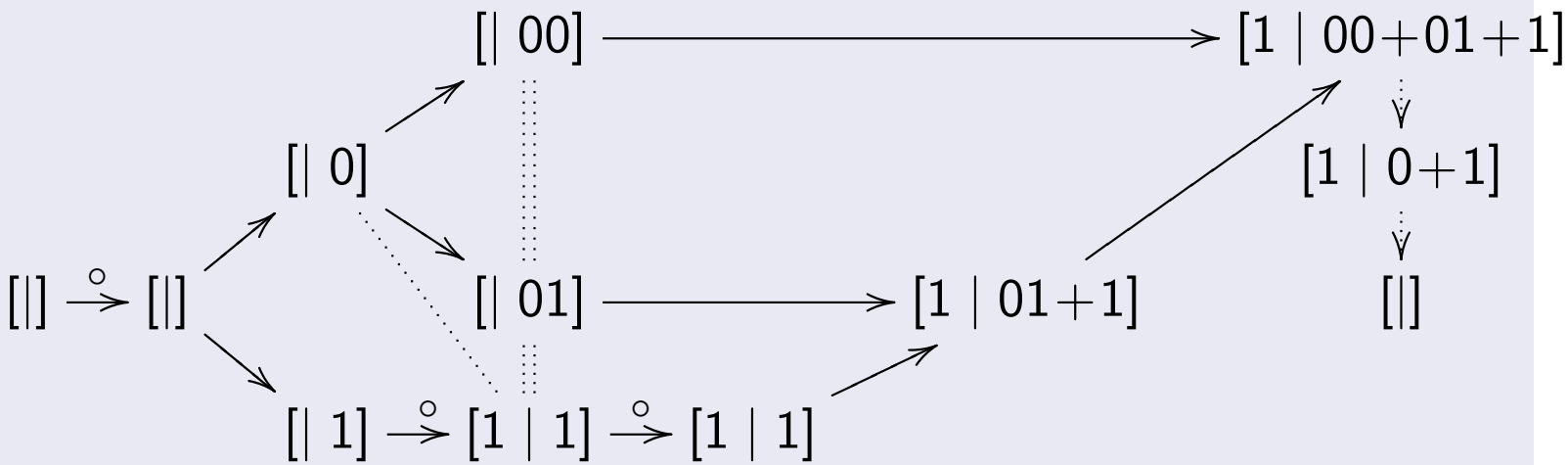


- An update causes no change on the id.
- Forks append either 0 or 1 to each string in the id.
- A join merges the sets of string.
- A possible simplification is attempted upon a join.

Data Causality
Version Stamps

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Update component



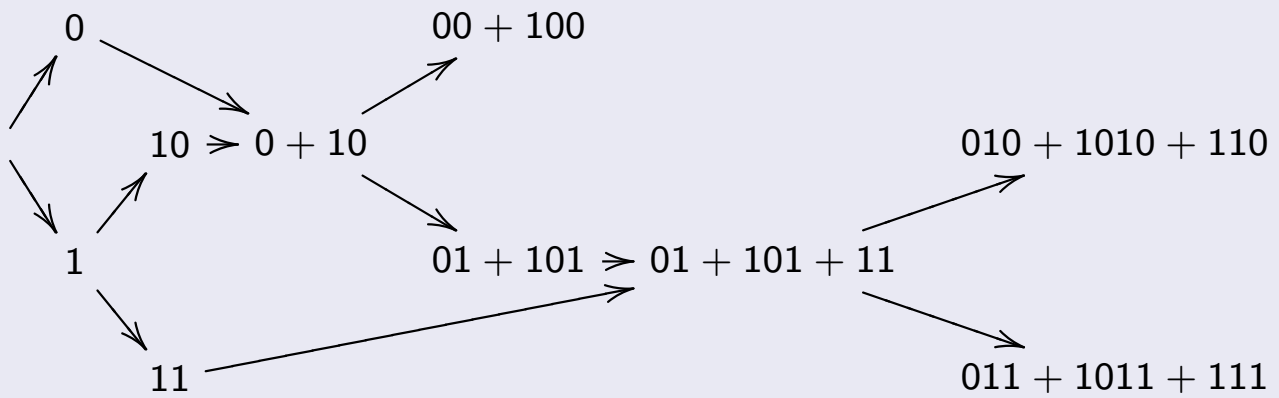
- Updates copy id into update.
- A fork causes no change in the update component.
- A join merges the update components.
- A simplification upon join also reflects in update.

Data Causality

Version Stamps: Pollution of the Namespace

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Patologic run



- Pattern: join and fork again with alternating replicas.
- Leads to an overly refined namespace that cannot be simplified.
- This pattern can often occur in a real usage scenario.
- Copy of the identity to the update component aggravates the problem.
- Although correct practical application is severely compromised.

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Insights

- Identity management in Version Stamps is too restrictive.
- Partly due to the avoidance of counters.
- Combination of counters and a more flexible identity management strategy.

Key observation regarding identity management

- Each identity must be different from the other replicas.
- One fragment of its identity is enough.

Carlos Baquero
Distributed Systems Group
Universidade do Minho

Identity component

- Contains a locally generated identifier.
- Set of binary strings.

Update component

- Keeps a compressed causal history.
- Set of tuples.
- Each tuple is composed by a binary string and a counter.

Data Causality

The Dynamic Map Clock Mechanism: Operations

Update

- Chooses one fragment from the identity component.
- And increments its counter on the update component.
- No need to use the same fragment in each update.

Fork

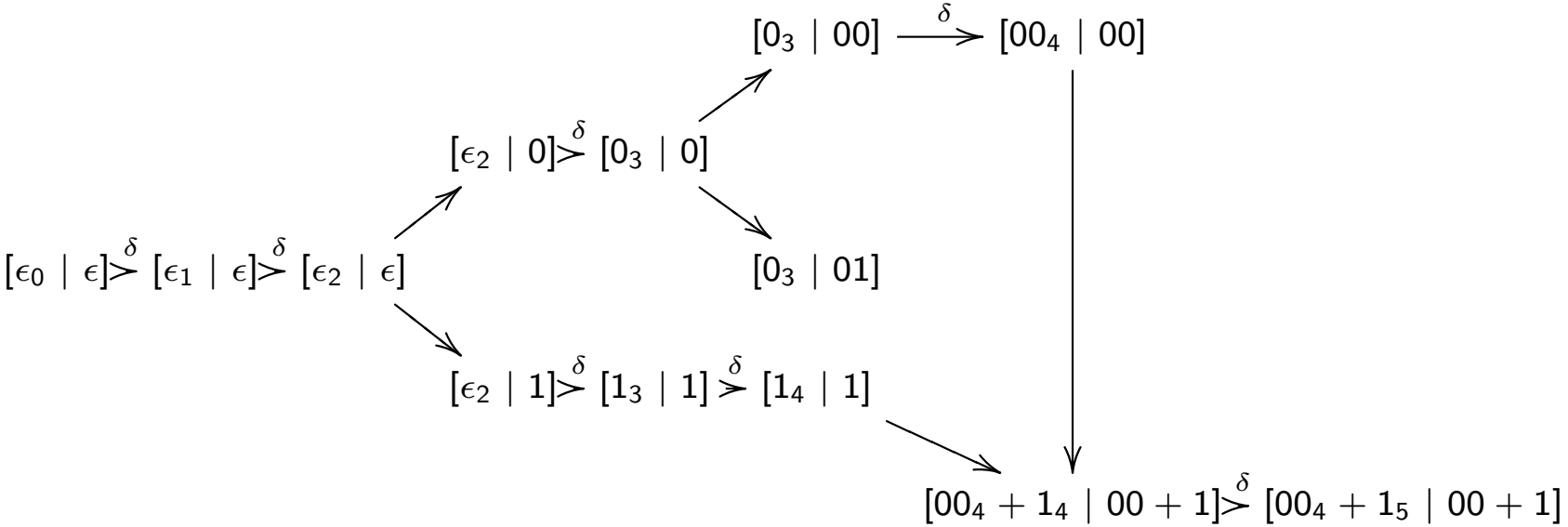
- Either specializes a fragment (0 derives 00 and 00).
- Or partitions the identity fragment set (0 + 11 derives 0 and 11).

Join

- Merges identity components and attempts its simplification.
- Merges update components (compressed causal histories).

Data Causality
The Dynamic Map Clock Mechanism

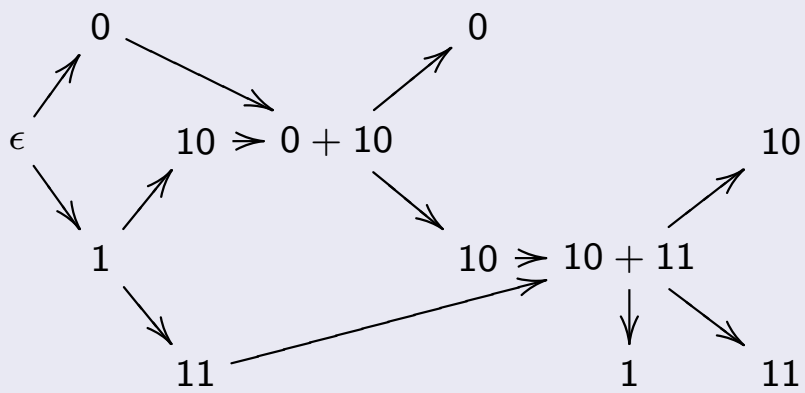
Carlos Baquero
Distributed Systems Group
Universidade do Minho



Data Causality

Non-pollution of the name space

Patologic run



Bibliography

- Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms. Özalp Babaoğlu, Keith Marzullo. 1993.
- Version Stamps: Decentralized Version vectors. Paulo Almeida, Carlos Baquero, Victor Fonte. ICDCS 2002.
- Bounded Version Vectors. Bacelar Almeida, Paulo Almeida, Carlos Baquero. DISC 2004.
- Improving on Version Stamps. Paulo Almeida, Carlos Baquero, Victor Fonte. RDDS 2007.