"Ecole IRISA" : distributed algorithms and models
Course 1 : some fundamentals

C. Jard, October 2006

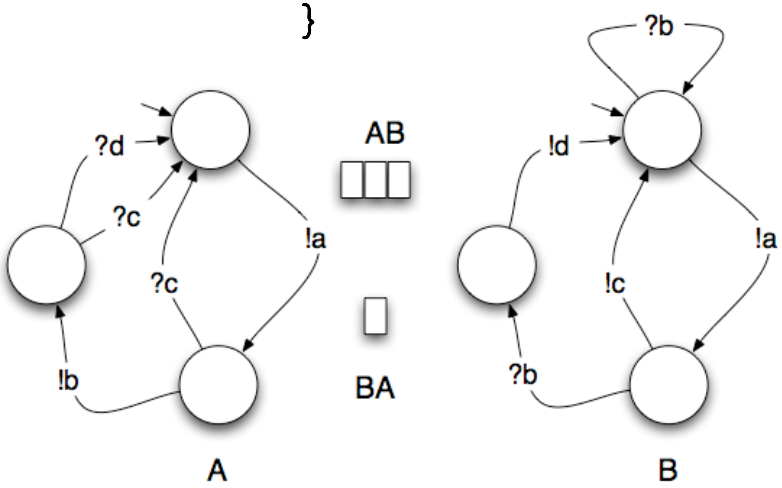
- 1.1 Distributed programs
- 1.2 Abstract behaviour
- 1.3 Causality
- 1.4 Lamport's coding
- 1.5 Vector clocks
- 1.6 Interval approximation
- 1.7 Notion of state for a distributed run
- 1.8 Global checking of traces
- 1.9 Distributed checking

1. A simple protocol (in Promela/Spin) : a first distributed program

```
mtype = {a,b,c,d};

/* a : connect_request
   b : disconnect_request
   c : distant_disconnect_request
   d : disconnect_confirm
*/

chan AB = [3] of {mtype};
chan BA = [1] of {mtype};
```

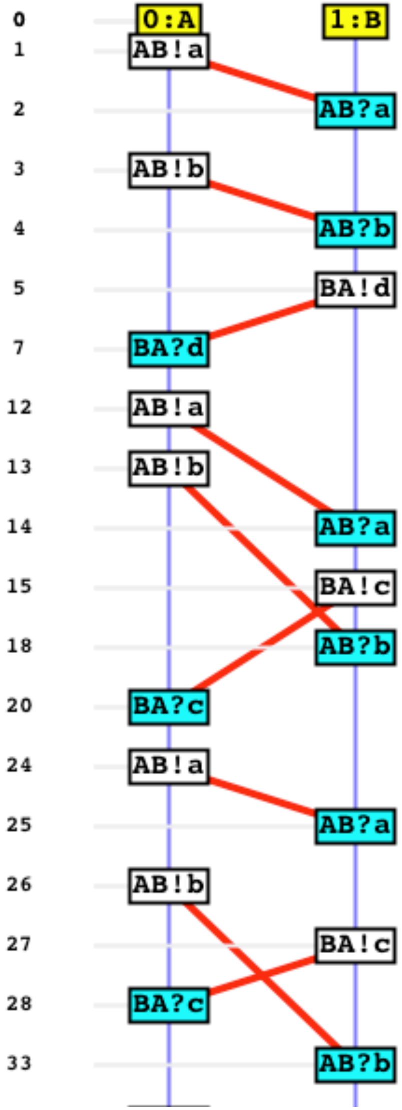
```
active proctype A()
{

    do
     :: AB!a;
       if
       :: BA?c;
       :: AB!b; if
                :: BA?c;
                :: BA?d
                fi
       fi
    od
}
```

```
active proctype B()
{
    do
     :: AB?b;
     :: AB?a;
       if
       :: AB?b; BA!d;
       :: BA!c
       fi
    od
}
```



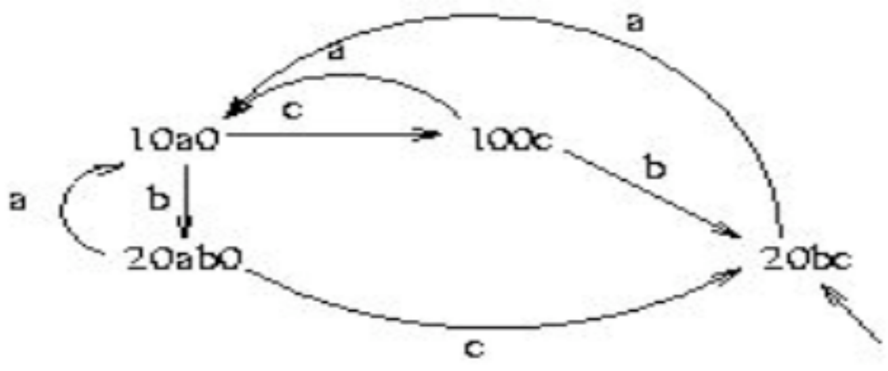
Example of scenario

Spin Version 4.0.7 -- 1 August 2003 -- Codec

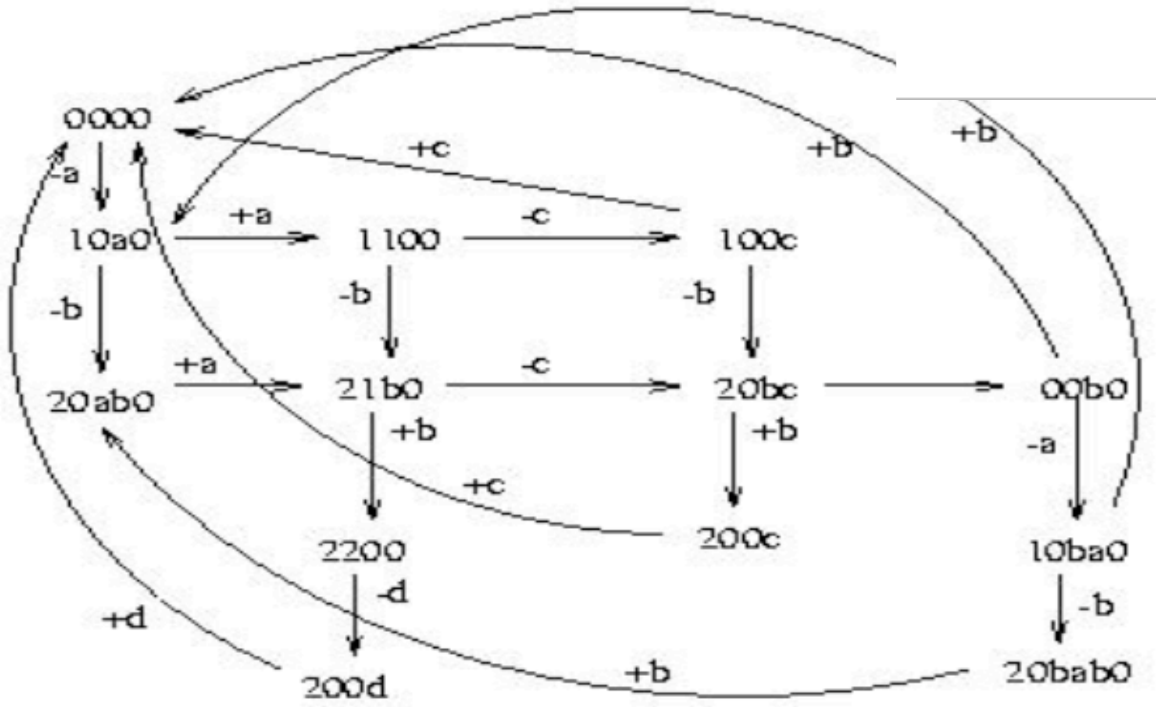



```
pwb-cj[27]% spin -a Codec
pwb-cj[27]% cc -o pan -DNOREDUCE -DSAFETY pan.c
pwb-cj[27]% ./pan
(Spin Version 4.0.7 -- 1 August 2003)

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 28 byte, depth reached 11, errors: 0
      13 states, stored
       8 states, matched
      21 transitions (= stored+matched)
       0 atomic steps
hash conflicts: 0 (resolved)
(max size 2^18 states)

1.533   memory usage (Mbyte)

unreached in proctype A
        line 25, state 13, "-end-"
        (1 of 13 states)
unreached in proctype B
        line 37, state 11, "-end-"
        (1 of 11 states)
```

2. Abstract state graph



Concrete state space

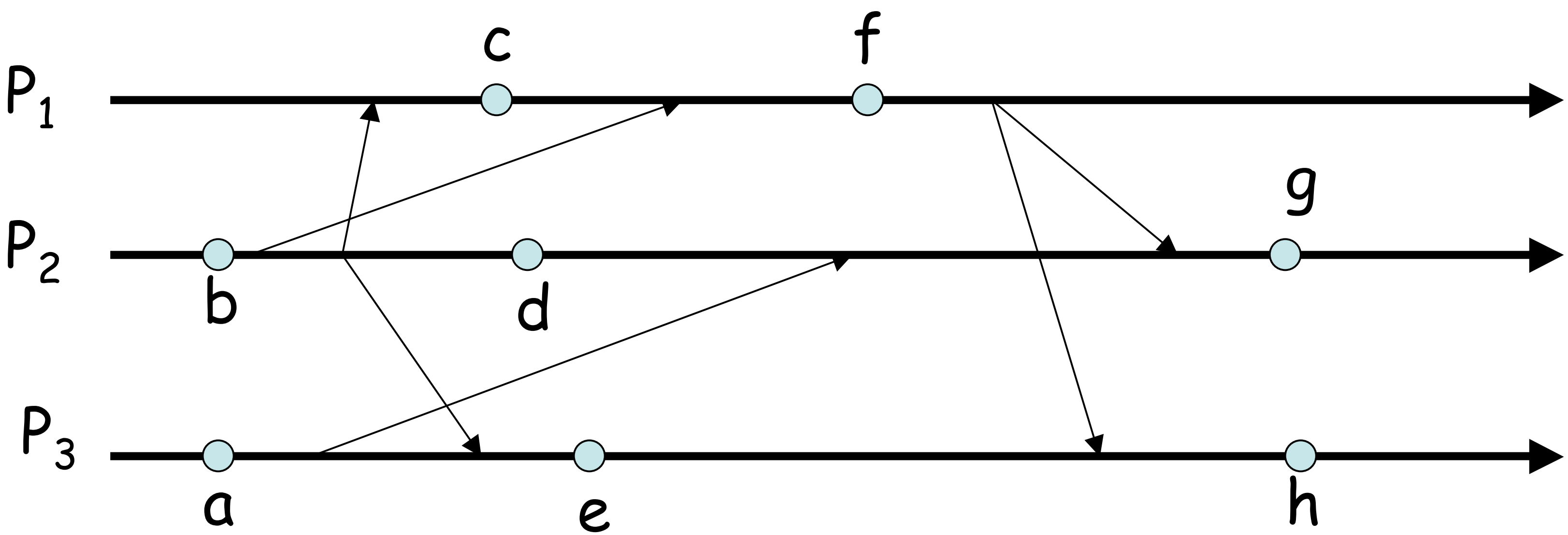


Abstract state graph
(only sendings of a,b
and c have been considered
as observable)

⇒ How recover
the causalities?

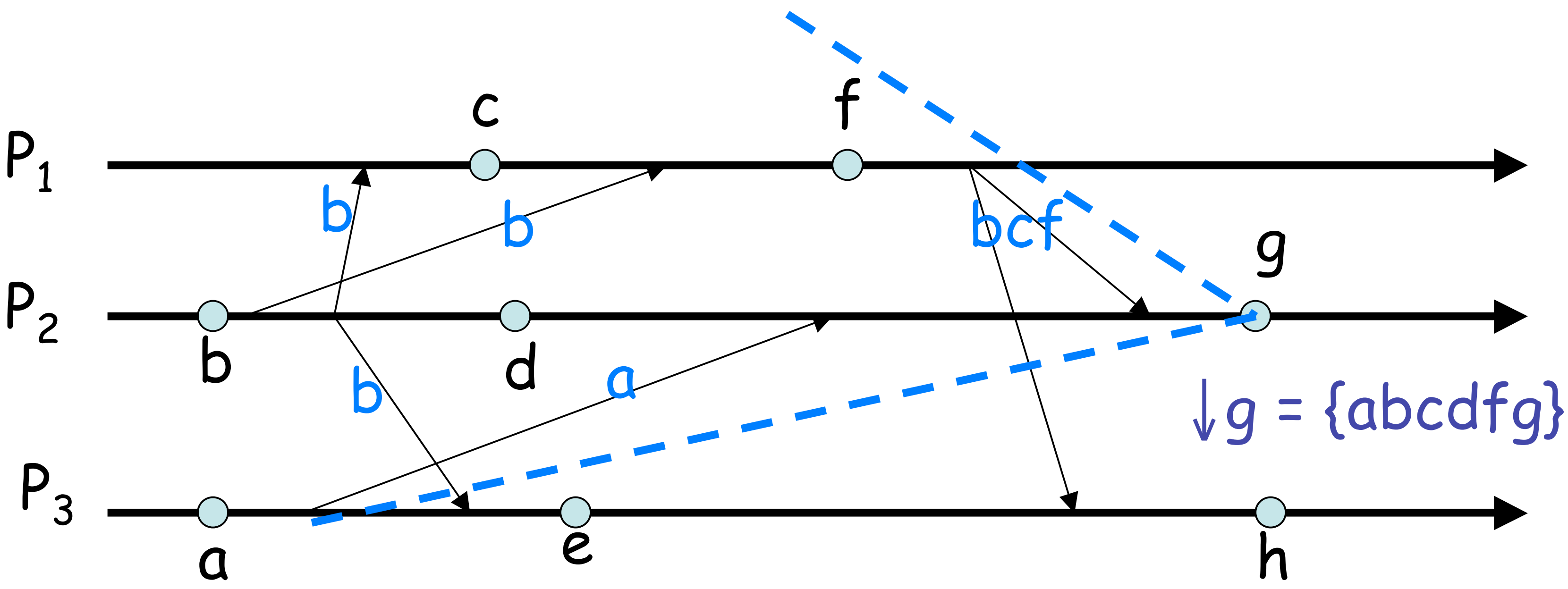
3. Causality between observable events (Lamport 78)

- N sequential processes (P_1 to P_n)
- Processes perform events during their life. Some of them are traced (the observable events)
- Communication by passing messages synchronises the process activity



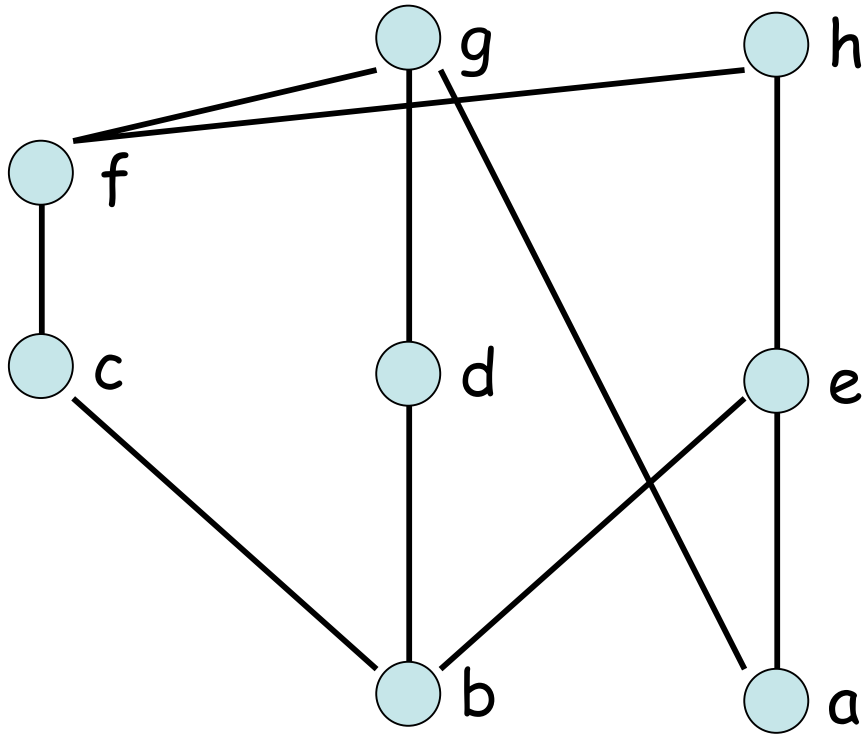
On-the-fly observation of causality (by instrumentation)

- Partial order relation of causality: $x \leq y$ iff it exists a path linking x to y in the chronogram
- Causal past: $\downarrow x = \{y \mid y \leq x\}$ $(x \leq y \Leftrightarrow \downarrow x \subseteq \downarrow y)$



The Hasse diagram

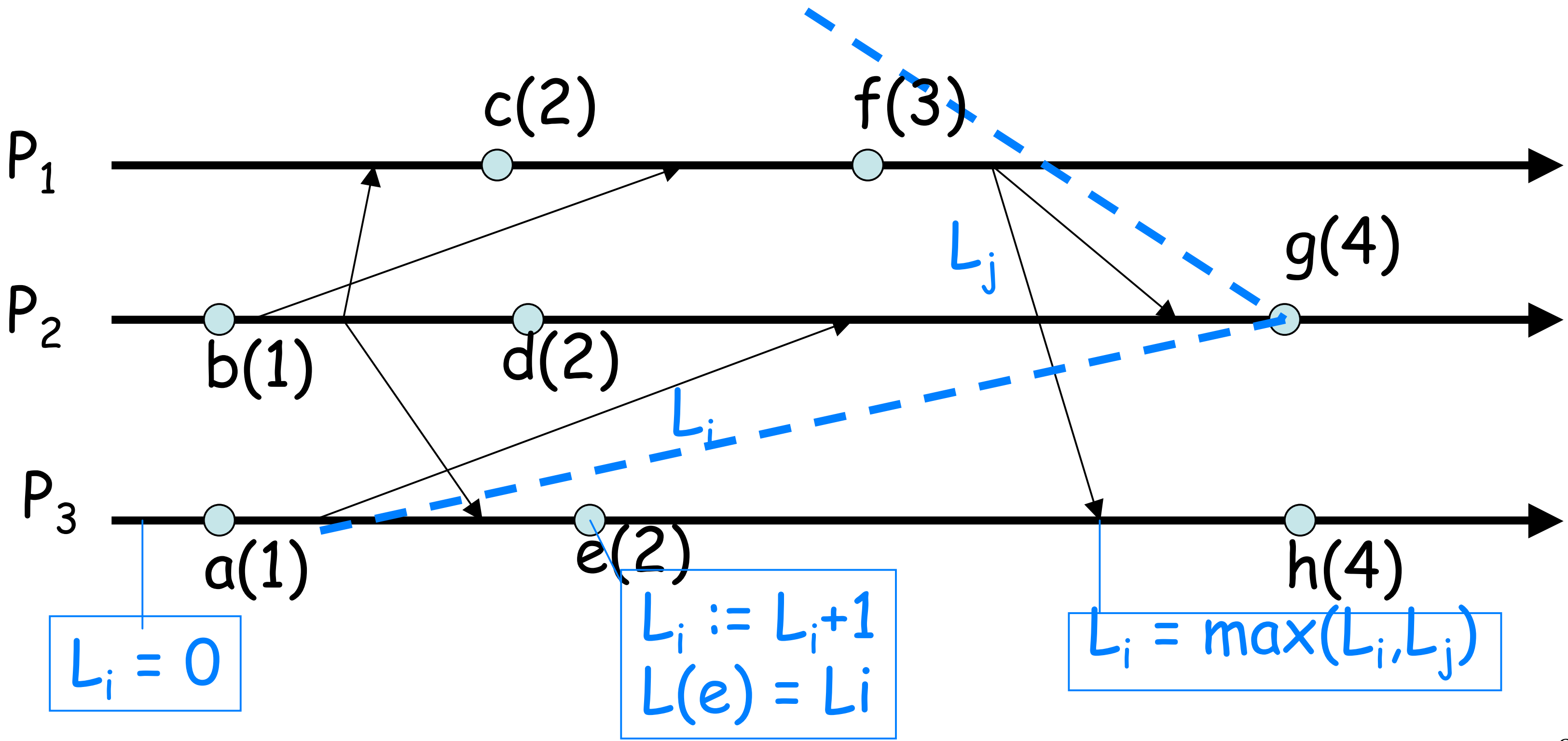
- Canonical form
- Bottom-up drawing
- Transitive reduction



Size: 15

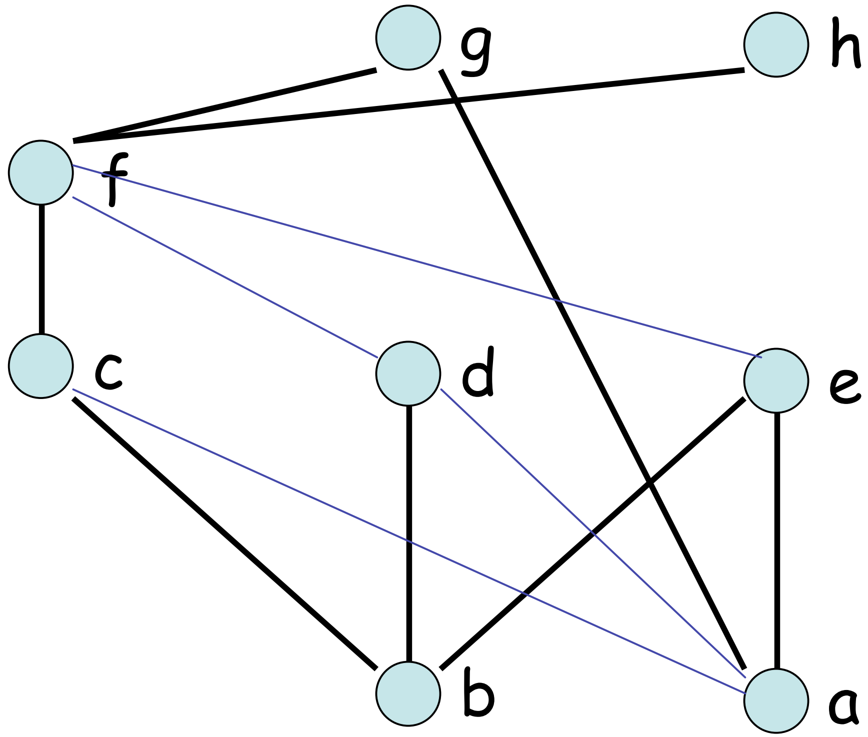
4. Lamport's approximate coding (1978)

- $L(x) = \text{height}(\downarrow x)$ (length of the longest path)
- $L(x) = \max \{L(y) \,/\, y < x\} + 1$
- $x \leq y \Rightarrow L(x) \leq L(y)$



The weak-order approximation

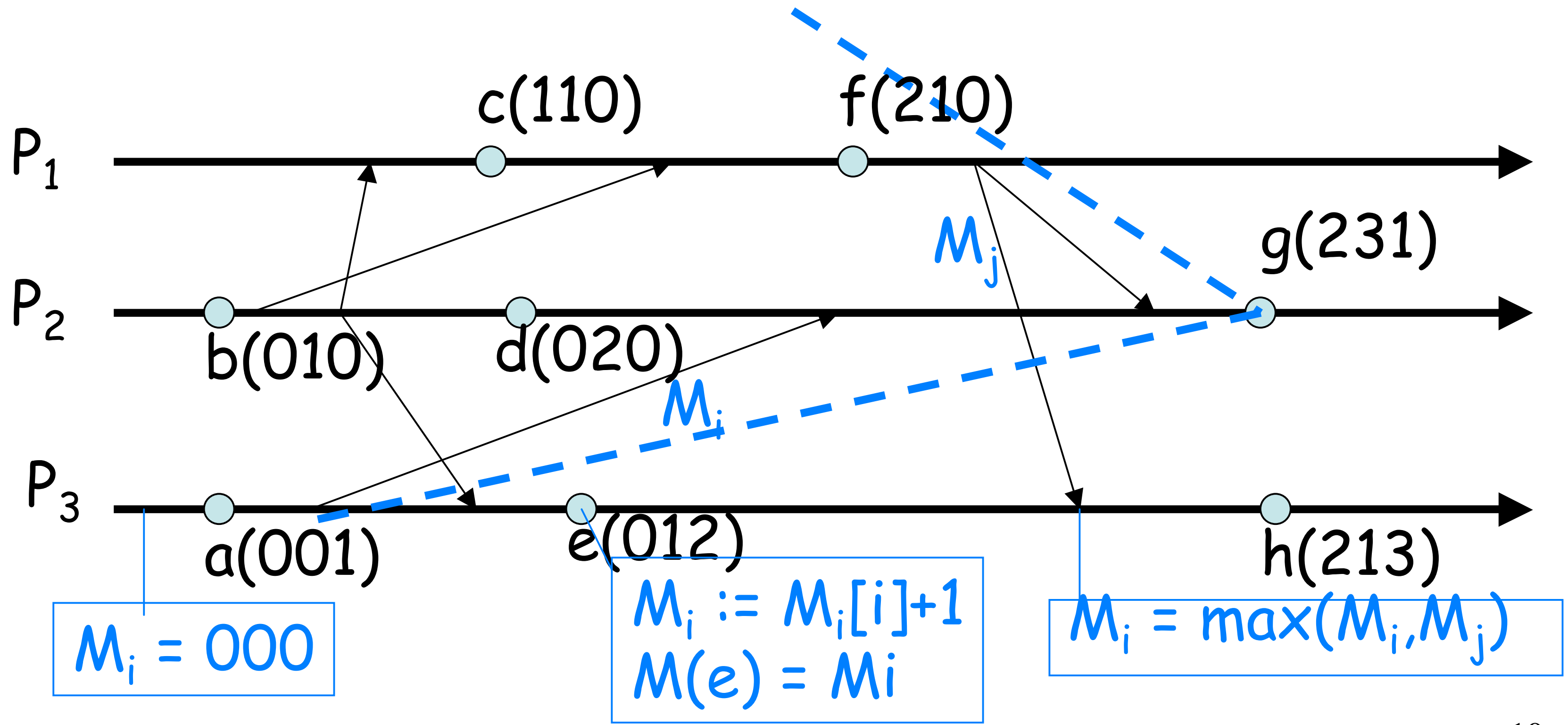
- Weak order = chain in which some elements are replaced by antichains
- A level structured order



Size: 24 (9 added comparabilities)

5. The Fidge & Mattern's exact coding (1988)

- $M(\downarrow x) = (|\downarrow x \cap P_i\ |)_{i=1..n}$ ("vector clock")
- $x \leq y \Leftrightarrow M(x) \leq M(y)$ (Dilworth's theorem, 1950 : an order of width n can be decomposed into n disjoint chains)

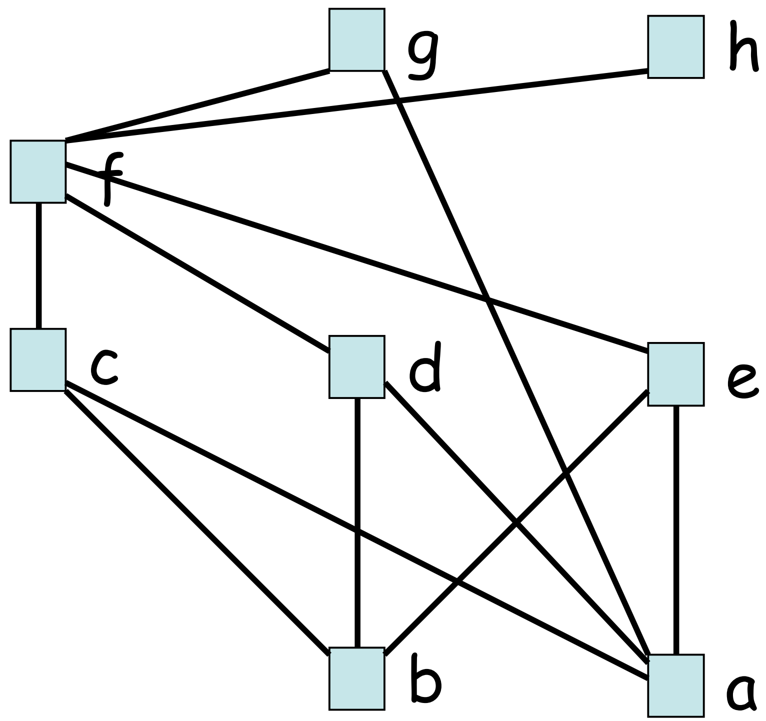


6. The quest of a constant size coding: the interval timestamps (Diehl & Jard 1992)

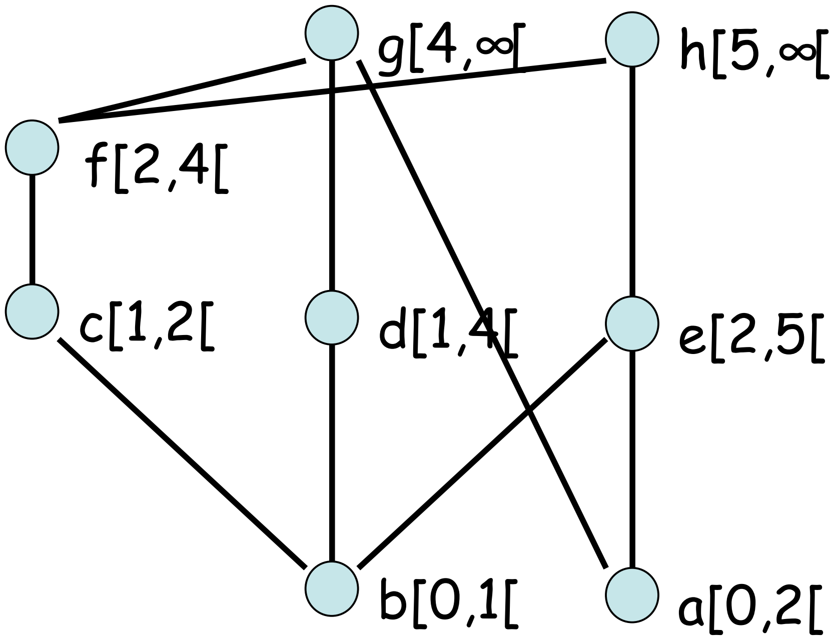
- The order given by the Lamport's timestamps is an interval order:

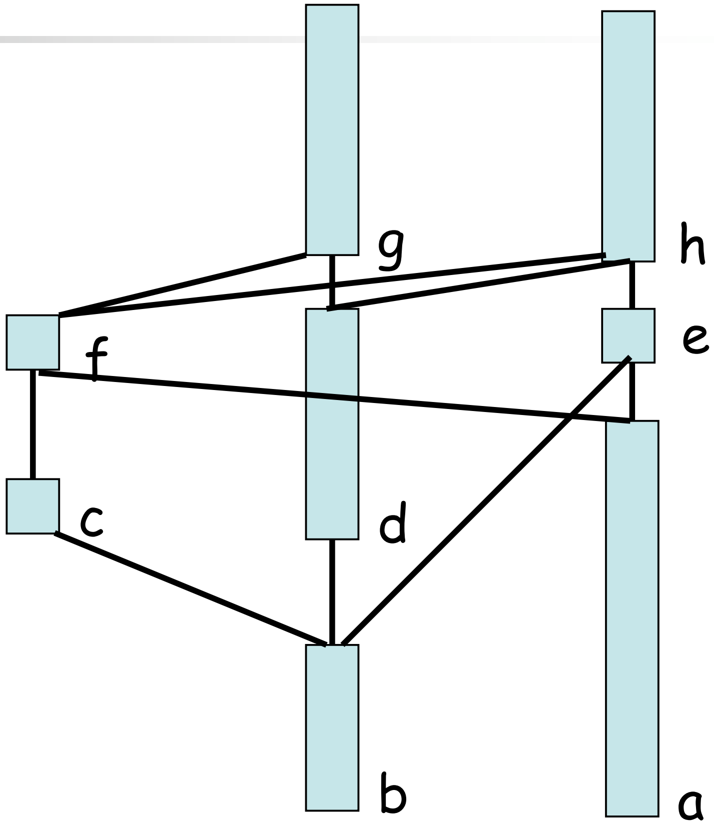
  [L(x), L(x)+1[

- Hence, the idea to change the sup to improve the accuracy of the approximation, while keeping a constant size timestamp -> "interval timestamping"



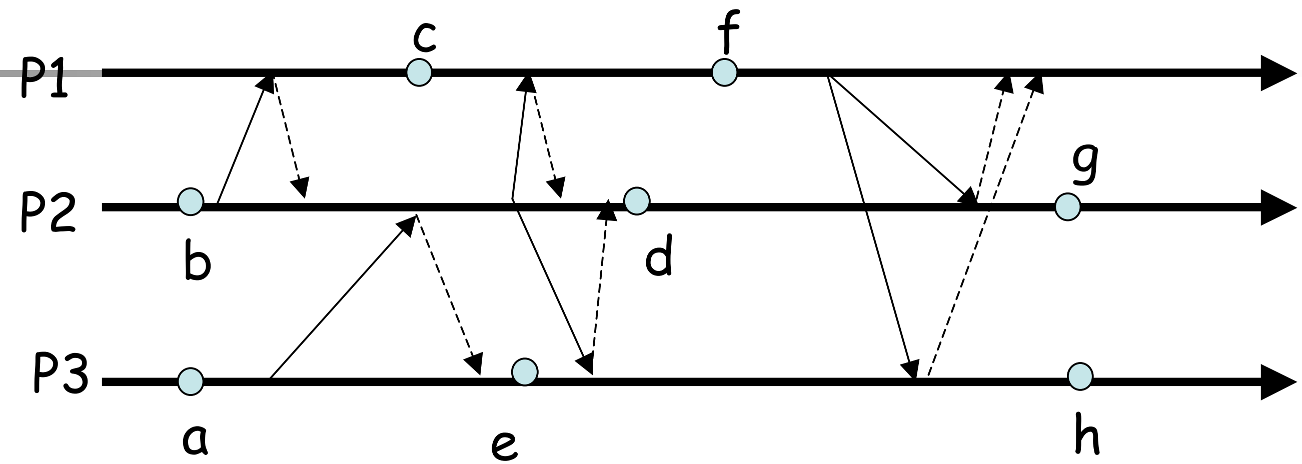
- $I^-(x) = |{\downarrow}x| - 1$
- $I^+(x) = \min \{I^-(y) \,/\, x < y\}$
- Theorem: if the causal order is an interval order, it is an exact coding scheme





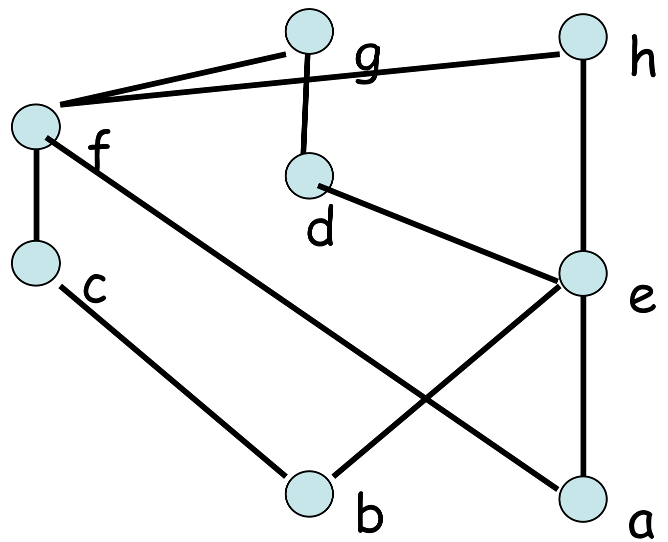
Size : 16

How to achieve an on-line mechanism? The RPC case



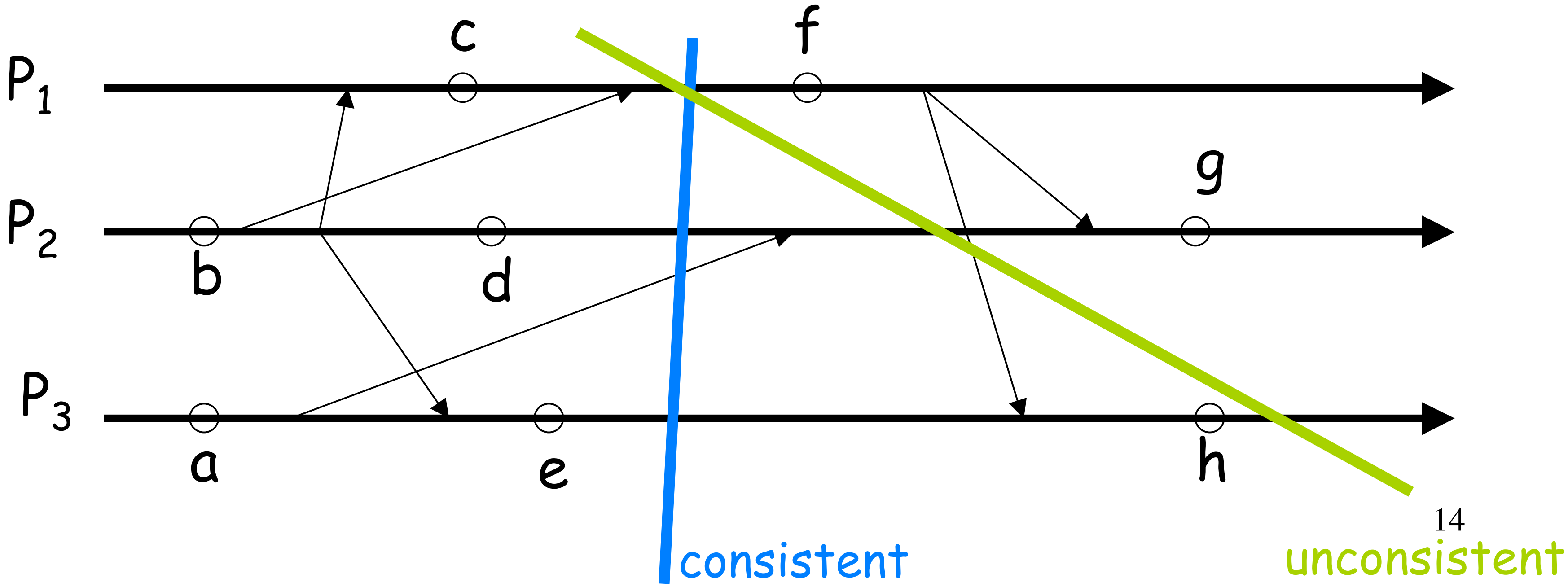
<u>Questions</u> :

- Define the timestamping algorithm
- Does it exist an exact coding of size 2?



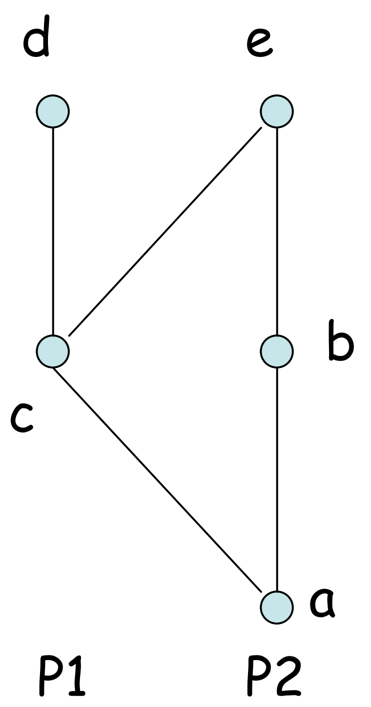
7. Notion of state for a distributed run

- The role of state is to code the past (cuts)
- Only consistent cuts are reachable
- These are downward-closed subsets: $X \subseteq E \;/\; \downarrow X = X$

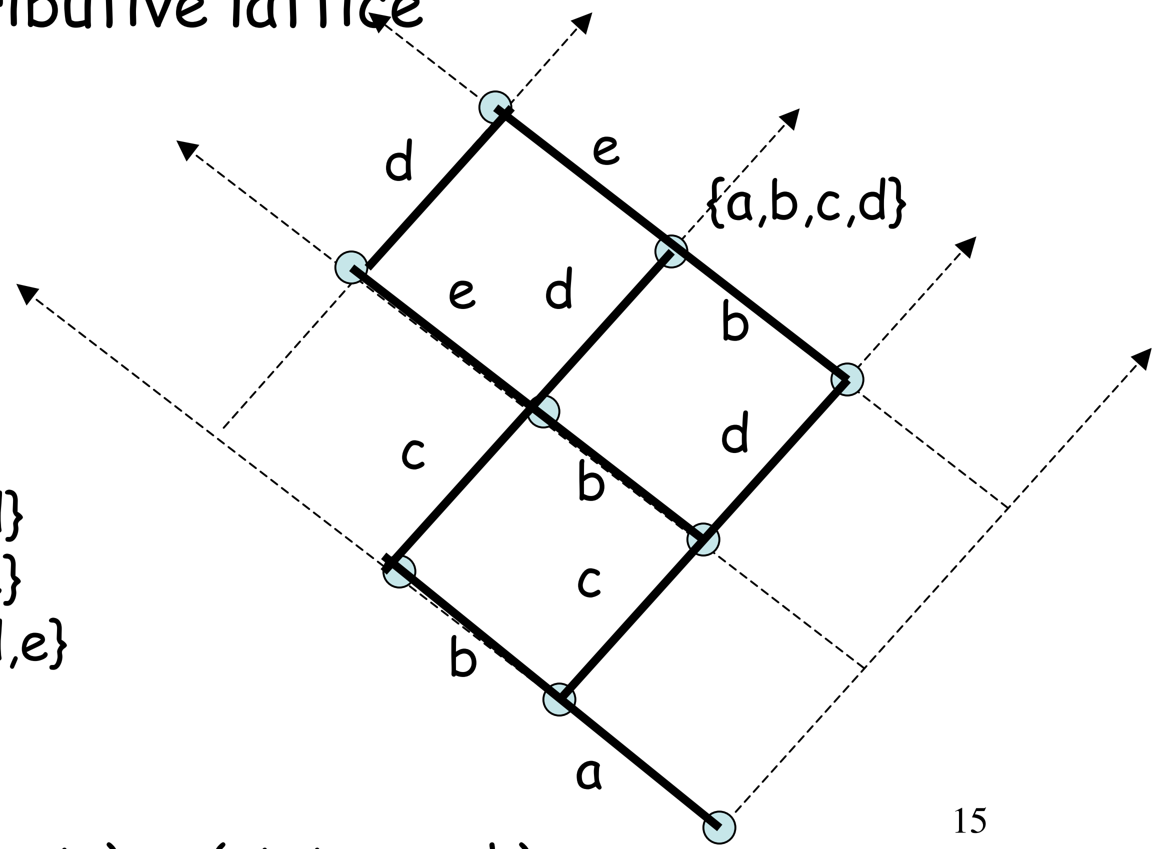


The state lattice

- The set of consistent cuts, ordered by set inclusion is a distributive lattice



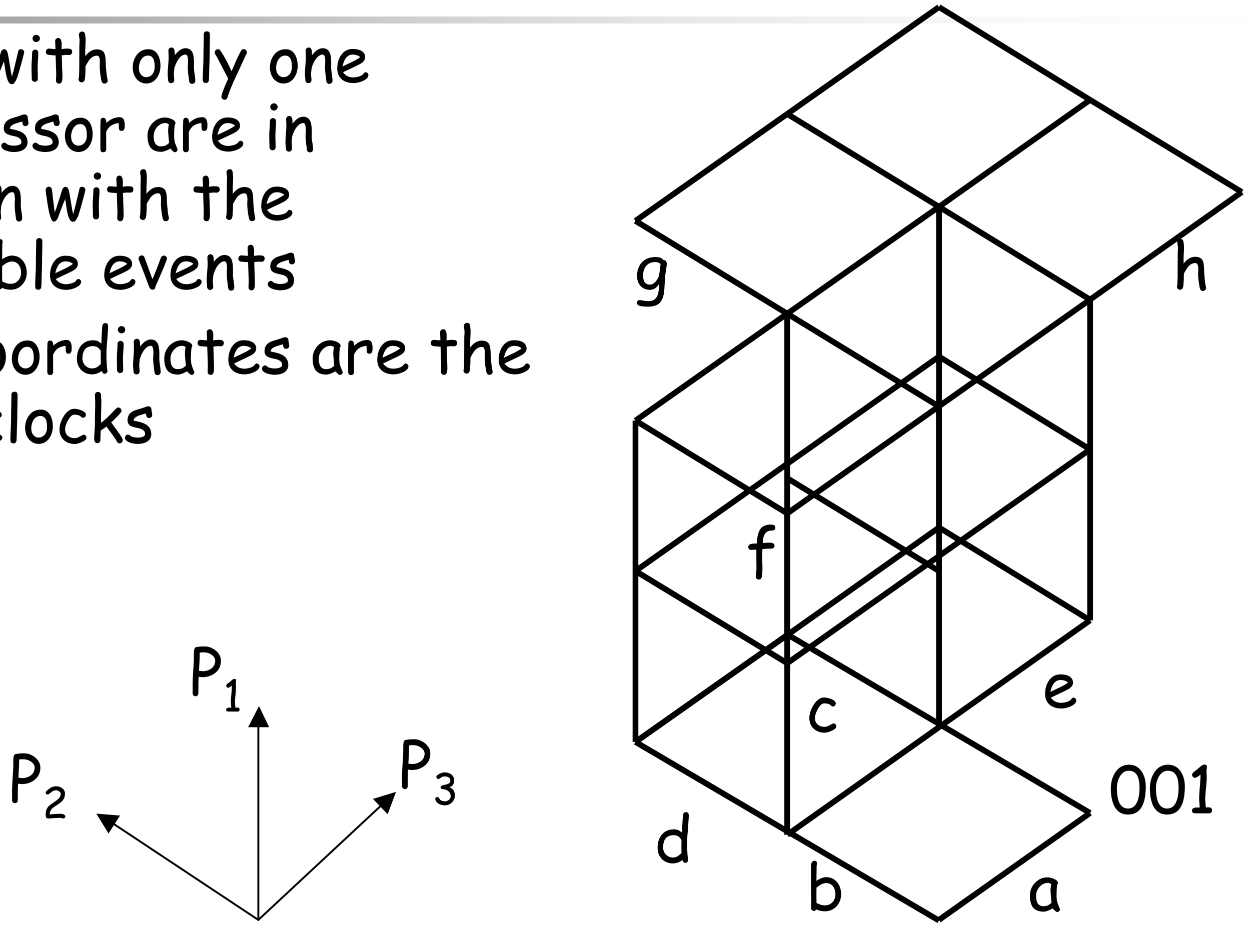
{}
{a}
{a,b}
{a,c}
{a,c,d}
{a,b,c}
{a,b,c,d}
{a,b,c,e}
{a,b,c,d,e}



(Hasse diagram) (cons. cuts) (state graph)

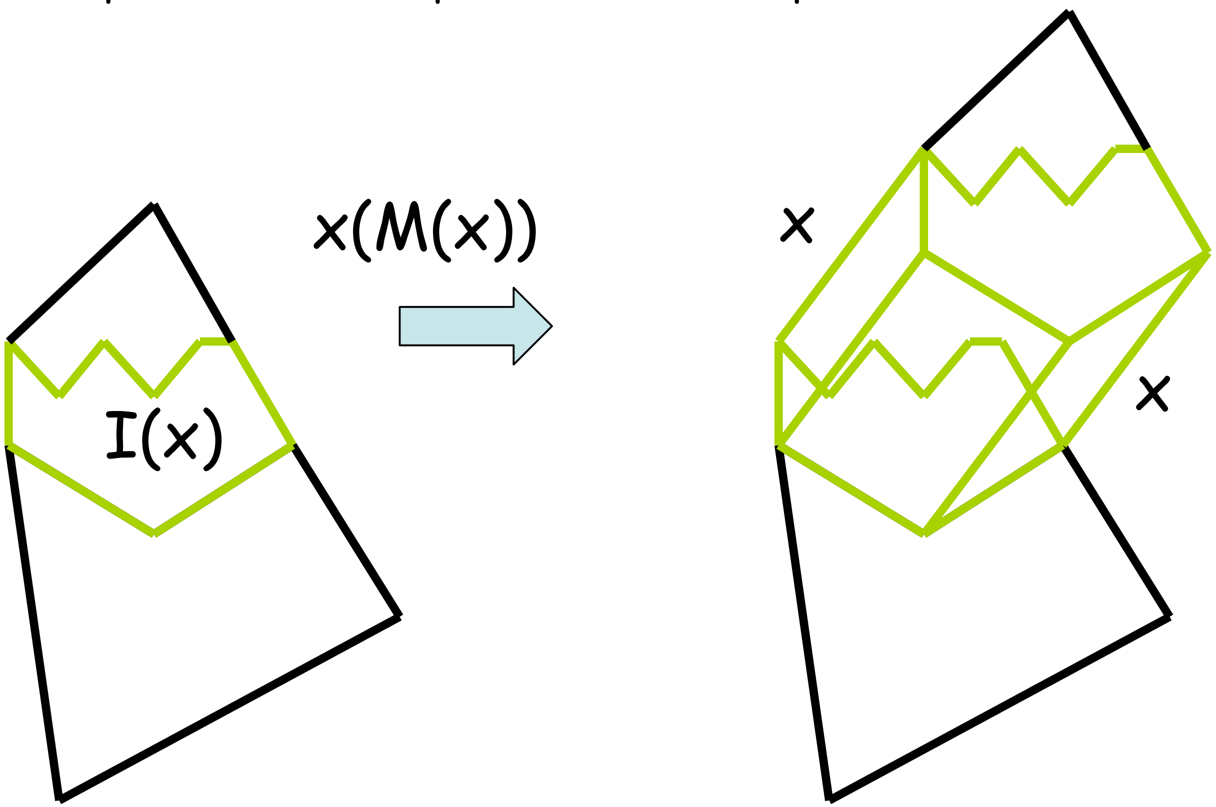
Can be embedded in a n-dimensional grid

- States with only one predecessor are in bijection with the observable events
- Their coordinates are the vector clocks

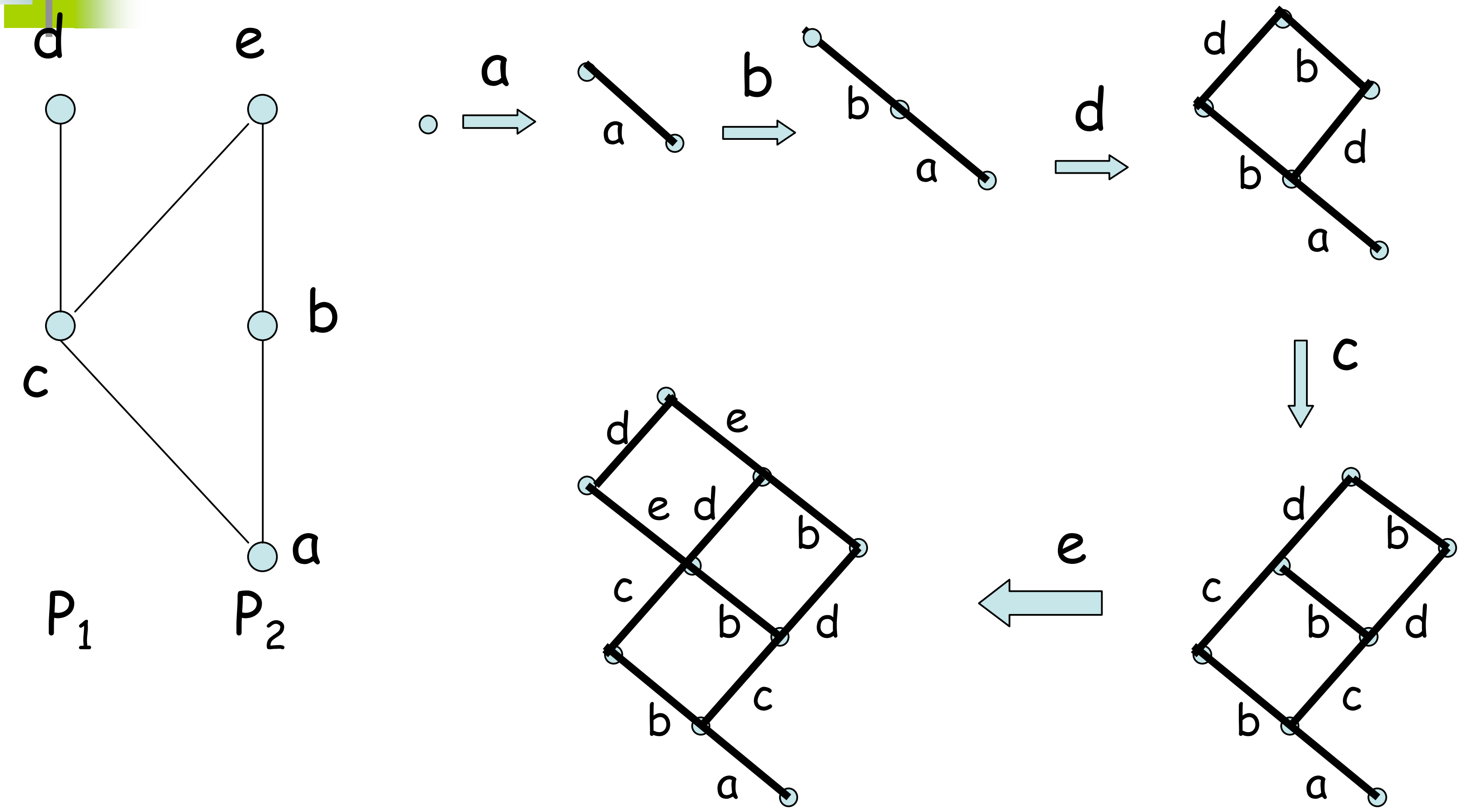


Its on-line construction [Jard & Rampon 1993]

- Can be built in linear time w.r.t. its size (exponential in the size of the order)
- Allows us to build in linear time the abstract state graph of a deterministic system: compare with the exponential state explosion of automata networks



An example of on-the-fly construction

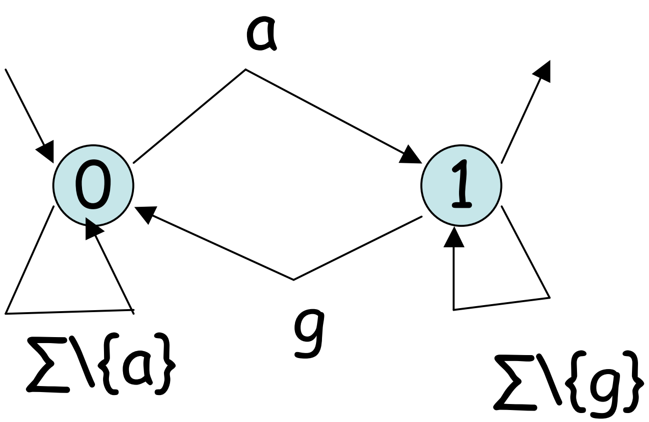


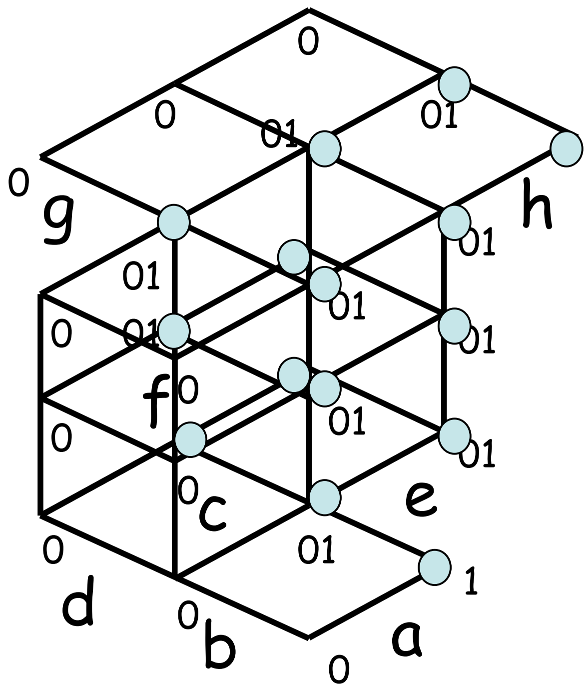
8. Trace checking (regular properties)

- Property $\Phi = \langle \Sigma, Q, q_0, F, \delta \rangle$,
  defines a langage $L(\Phi) = \{u \in \Sigma^* \mid \delta^*(u,q_0) \in F\}$
- State graph: transitions labelled with Σ,
  given a state I, Paths(I) is the set of words leading to I
- I satisfies Φ iff Paths(I) intersects $L(\Phi)$
- Equivalently: I satisfies Φ iff $\Phi(I)$ intersects F
  where $\Phi(I)$ is the set of states of the automaton Φ reachable by paths ending at I
- A trace satisfies Φ iff the maximal state (Σ) satisfies Φ

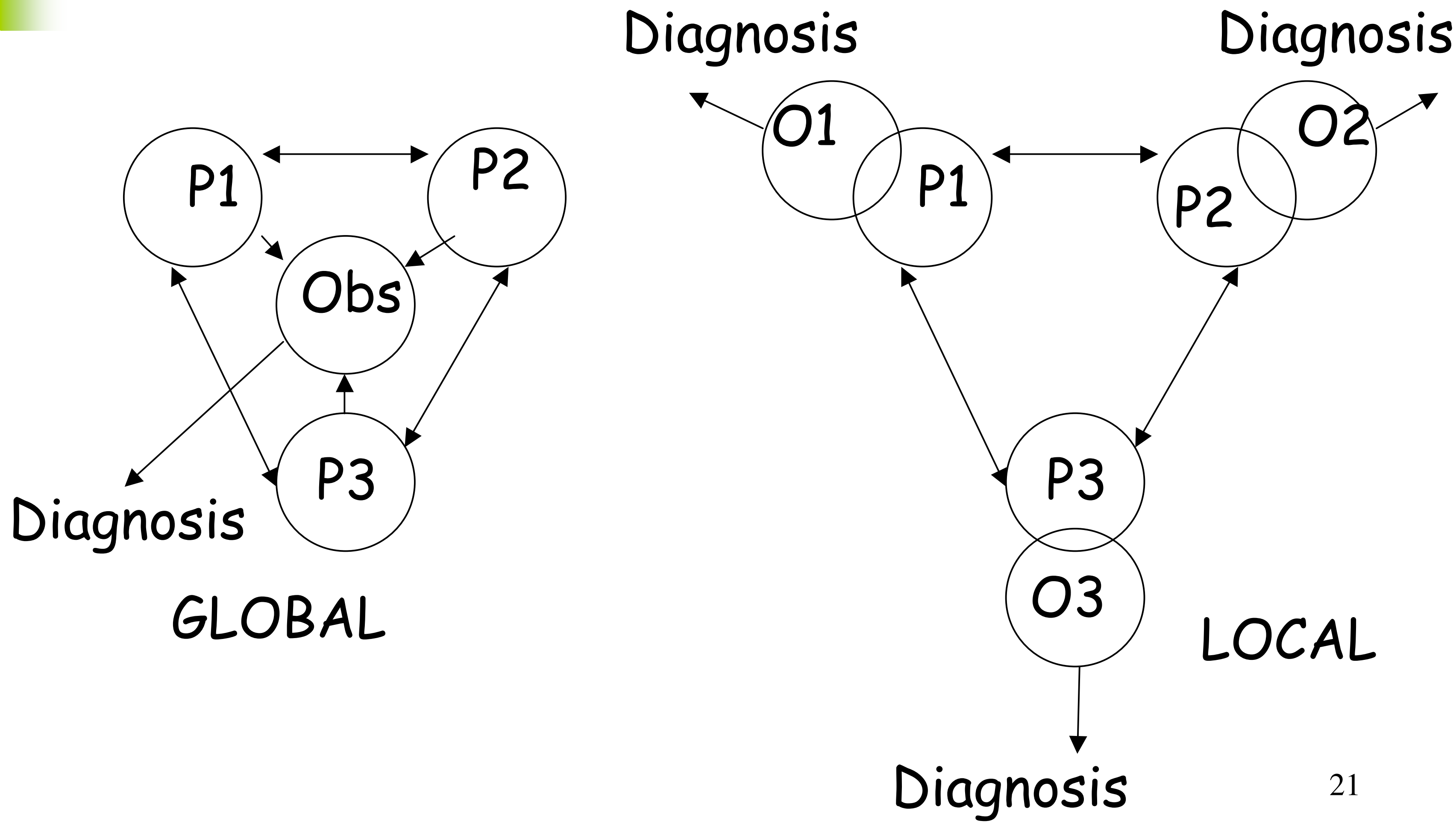
The (global) algorithm "trace checking"

- Φ(I) can be computed from the Φs of the immediate predecessors in the lattice (requires a causal observation) :
  $\Phi(I) = \{ \delta(label(J\text{->}I),q) \text{ for } q \in \Phi(J) \text{ and } J \in \downarrow I \}$
- Checking is done on-the-fly during the lattice construction





Local distributed or global checking?



Distributed checking [Jard & Raynal 1995]

Global :

- Causal dependency tracking
- On-line construction of the state lattice
- Verification during construction

Local :

- Restricted class of properties (causal flows)
- Distributed verification (timestamps extended with a state information)

9. Distributed trace checking (local regular properties)

- Properties are on the causal past of the observable events
- An observable event x satisfies a property Φ iff it exists a path ending in x in the Hasse diagram of the observed order such that the corresponding path is accepted by Φ
- Causal ordering case
- Can be computed on-the-fly and in a distributed manner

Principle

- The automaton Φ is know from all the processes
- x satisfies Φ is locally computed on the process which has produced x
- The state information is acquired (and piggybacked) by the messages of the observed execution
- Each process Pi maintain 2 arrays: LOi[1..n] and SLOi[1..n]. LOi[j] is the rank of the the last event observed Pj in the current past of Pi. SLOi[j] is the corresponding state information (of Φ) (when LOi[j] is maximal)

Algorithm (on Pi)

- **Data:** LOi[1..n] of integer; SLOi[1..n] of set of states
- **Init:** forall j, LOi[j]:=0; SLOi[j]:={q0}
- Upon observation of event x:
  LOi[i]:=LOi[i]+1;
  SLOi[i]:={δ(q,x)} forall k, forall q in SLOi[k];
  forall j#i, SLOi[j]:={}
- When sending a message to Pk:
  LOi and SLOi are added to the message

- Upon reception of msg(LOk,SLOk) from Pk:

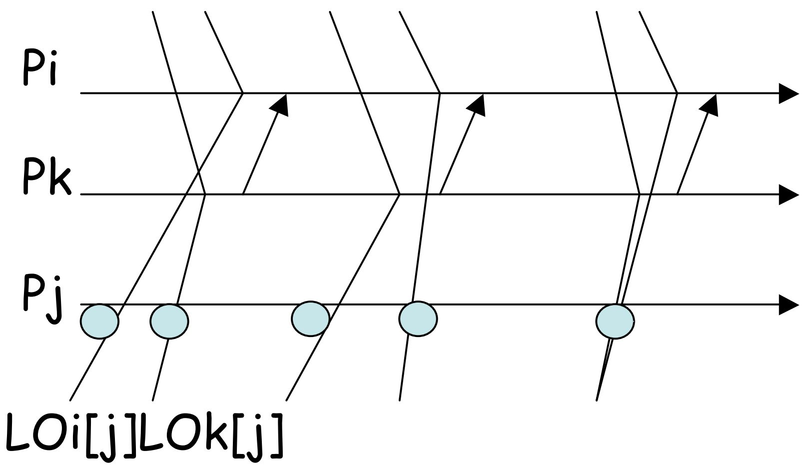
forall j, if

LOi[j]<LOk[j] then SLOi[j]:=SLOk[j]; LOi[j]:=LOk[j]

LOi[j]>LOk[j] then skip

LOi[j]=LOk[j] then

if SLOi[j]#{} and SLOk[j]={} then SLOi[j]:={}



SLOi[j]	{}	X	{}	X
SLOk[j]	{}	X	X	{}
New SLOi[j]	{}	X	{}	{}