Using Order in Distributed Computing

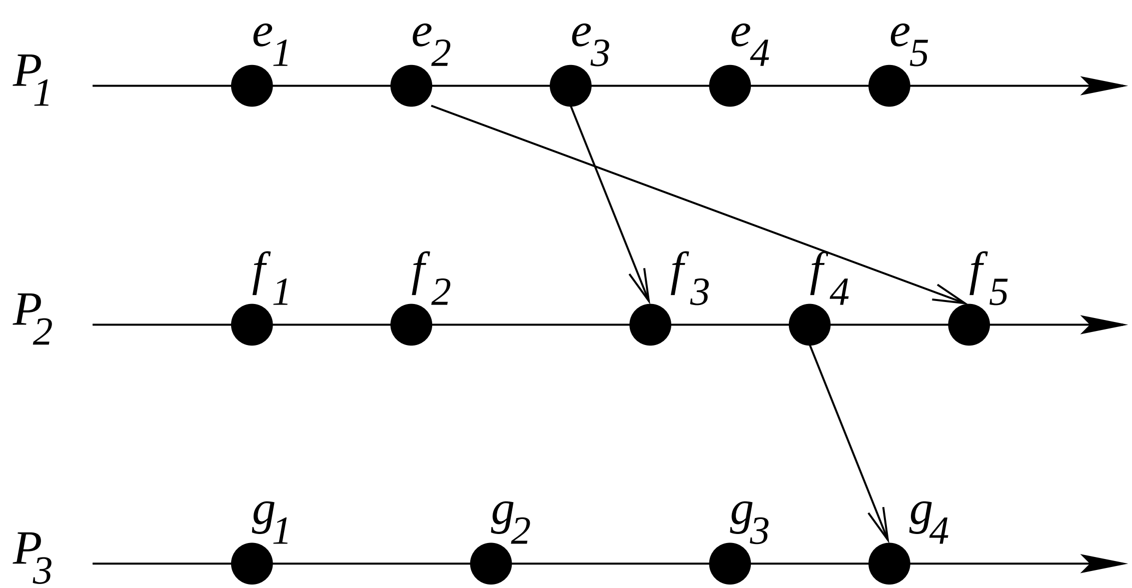
Vijay K. Garg
Neeraj Mittal (UT, Dallas)
Alper Sen (Freescale)
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712
email: garg@ece.utexas.edu

Happened-Before Relation in Distributed Computing

A computation is $(E, \rightarrow)$ where E is the set of events and $\rightarrow$ (happened-before) is the smallest transitive relation that includes:

(1) order within a process

(2) e is a send event and f is the receive implies $e \rightarrow f$.



[Lamport 78]

Talk Outline

- Happened-Before Relation
- Applications
  - Tracking Dependency: Chain decomposition, Dimension Theory
  - Detecting Global Predicates: Meet-closure, Chain merging, ideal enumeration
  - Computation Slicing: Birkhoff's representation theorem

Tracking Dependency

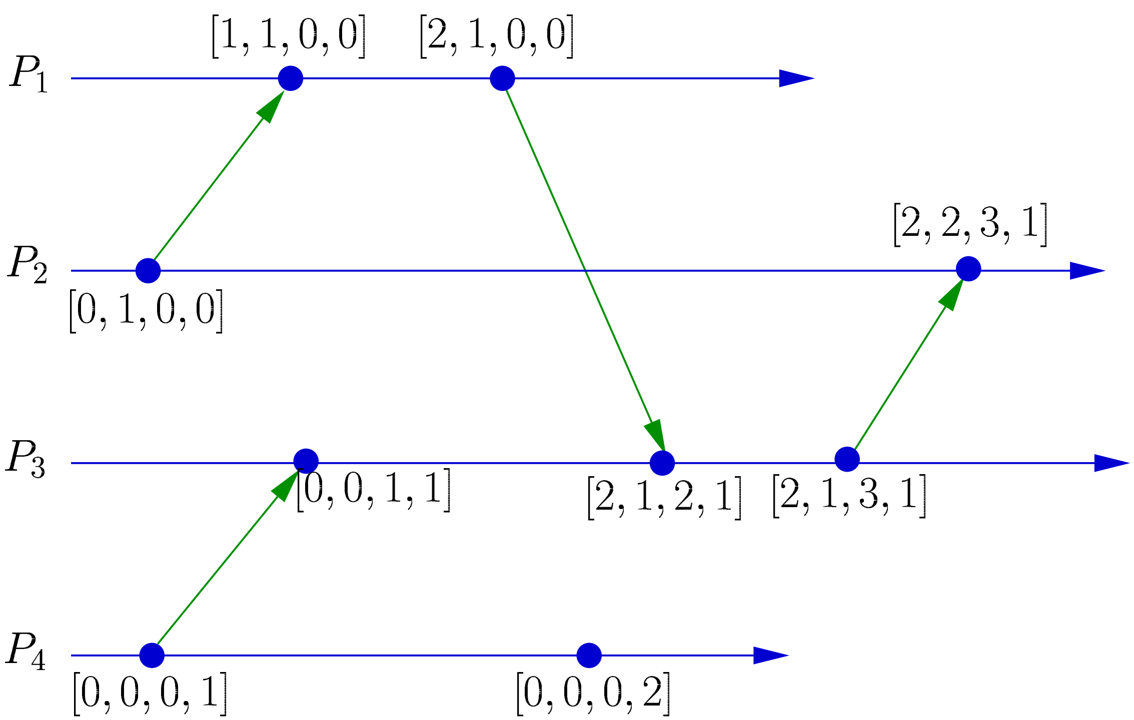
Motivation: Determine whether e happened before f.

Problem: Given $(E, \rightarrow)$, assign timestamps v to events in E such that

$$\forall e, f \in E : e \rightarrow f \equiv v(e) < v(f)$$

Online Timestamps: Vector Clocks [Fidge 89, Mattern 89]:
Every process maintains a vector v of size N, the number of processes.
($v[k]$ at P_i = the number of events executed by P_k as known to P_i).

Vector Clocks in a Distributed System



all events: increment $v[i]$
send events: piggyback v
receive events: combine timestamps
Theorem:

$$e \rightarrow f \equiv v(e) < v(f)$$

Dynamic Chain Clocks

Problem with vector clocks: scalability, dynamic process structure

Idea: Computing the "chains" in an online fashion [Aggarwal and Garg 05] for relevant events

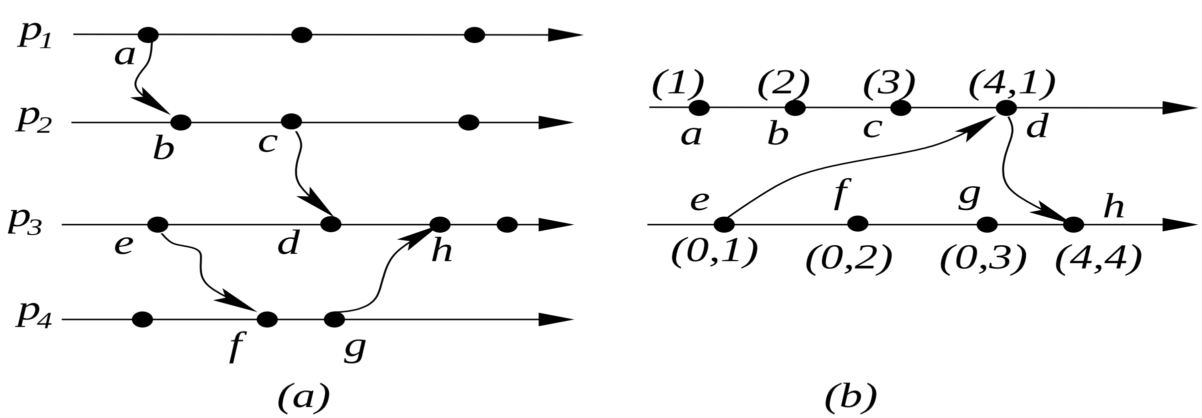
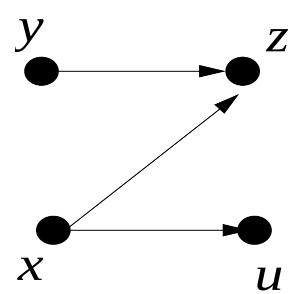


Figure 1: (a) A computation with 4 processes (b) The relevant subcomputation

Online Chain Decomposition

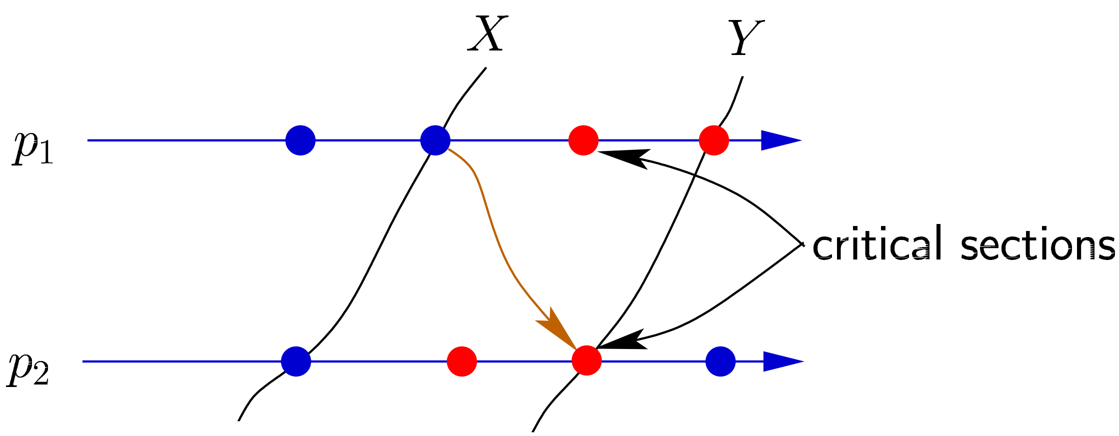
- Elements presented in a total order consistent with the poset
- Assign elements to chains as they arrive
- Game: Bob presents elements, Alice assigns them to chains
- For a poset of width k, Bob can force Alice to use $k(k+1)/2$ chains. [Felsner 97].
- An online algorithm that uses $O(k^2)$ chains with $O(k^2)$ comparisons per event. [Aggarwal and Garg 05]



Global Predicate Detection

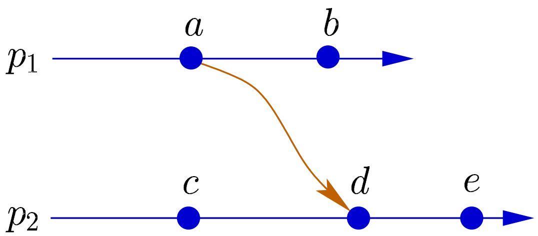
Predicate: A global condition expressed using variables on processes (a boolean function on the set of ideals of the poset) e.g., more than one process is in critical section.

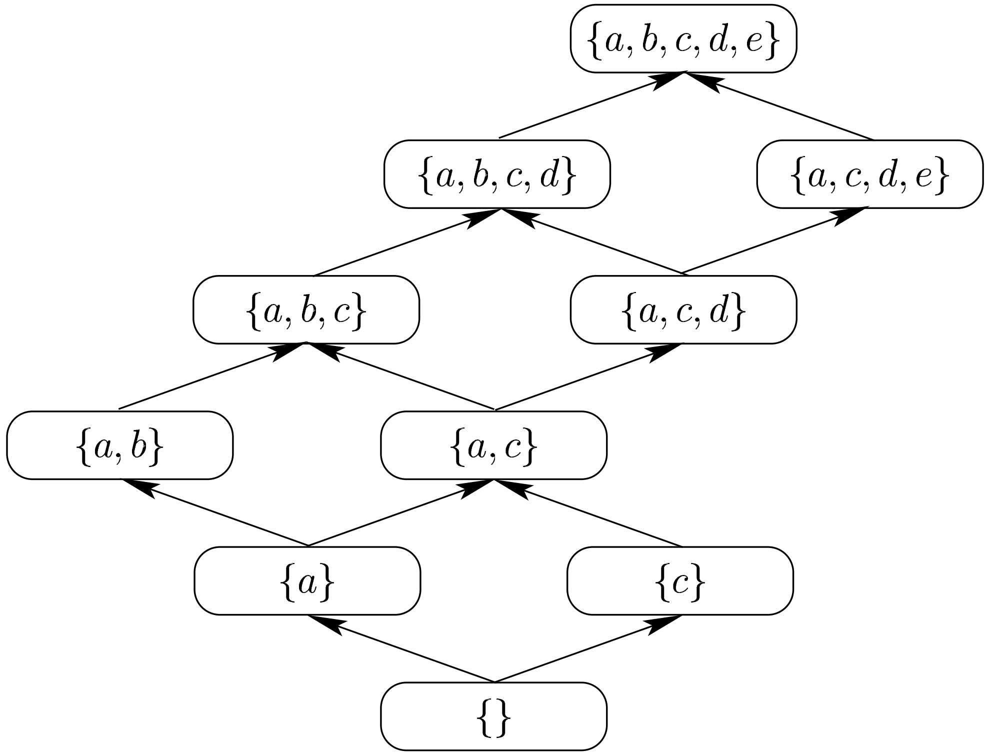
Problem: find an ideal (a consistent cut) that satisfies the given predicate



The Main Difficulty

Algorithm for general predicate

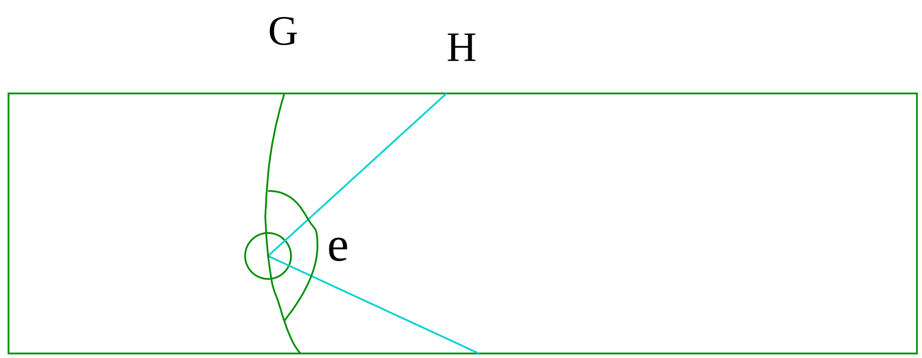




NP-complete
As many as $O(k^n)$ consistent cuts

k: number of events/process, n: number of processes

Detecting Linear Predicates



(Linearity): If B is false in G then there exists an event e, such that all "true" cuts greater than G include e.

$$\neg B(G) \ \Rightarrow \ (\exists e \in E - G : \forall H \supseteq G : B(H) \Rightarrow (e \in H))$$

(Advancement Property) can determine the "crucial" event in polynomial time

Theorem: Any linear predicate that satisfies advancement property can be detected efficiently.

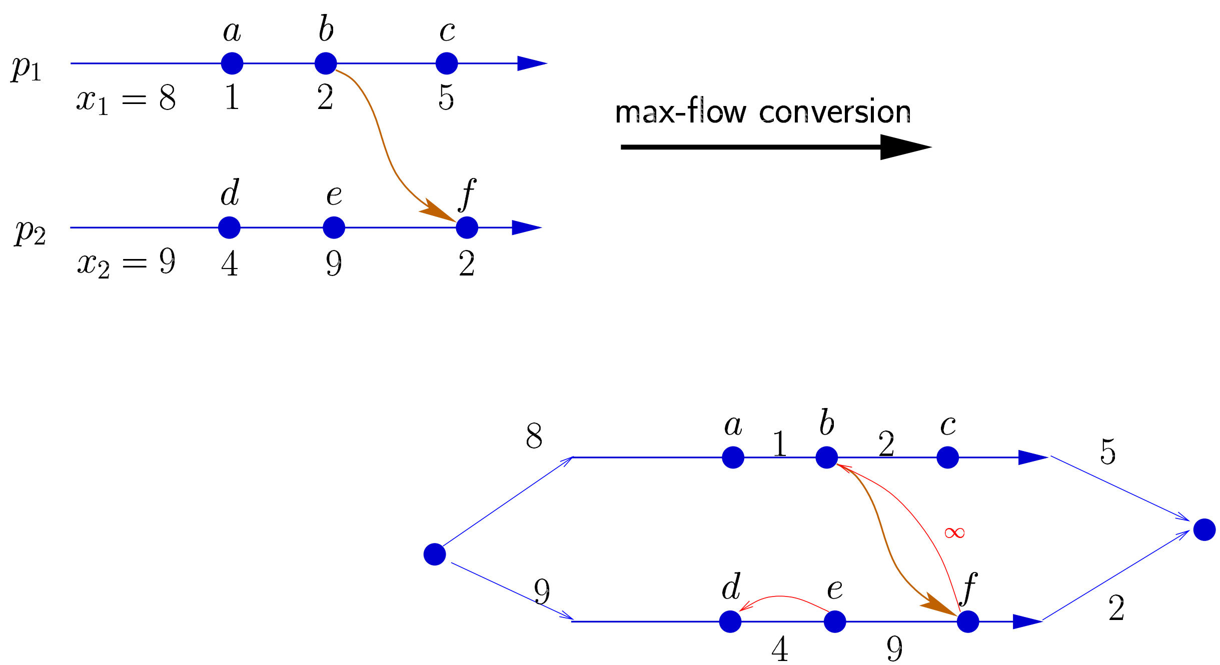
Theorem: [Chase and Garg 95] B is linear iff it is meet-closed.

Relational Predicates

Let $x_i \geq 0$ be variable at P_i. Predicates of the form [Groselj 93, Chase and Garg 95]

$$\Sigma x_i \geq k$$

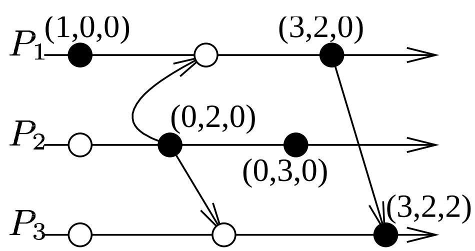
Algorithm: Consistent cut with minimum value = min cut in the flow graph



Relational Predicates: Binary Variables

Restriction: $x_i \in \{0, 1\}$

Theorem Exists an algorithm that merges N queues into $N-1$ queues in an online fashion. [Tomlinson and Garg 96]



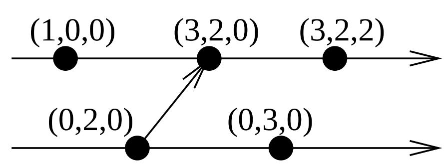
(a)

C_1	C_2	C_3
(1,0,0)	(0,2,0)	(3,2,2)
(3,2,0)	(0,3,0)	

(b)

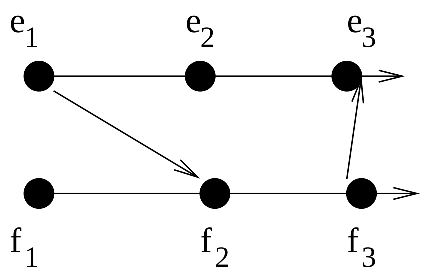
C_1	C_2
(1,0,0)	(0,2,0)
(3,2,0)	(0,3,0)
(3,2,2)	

(c)

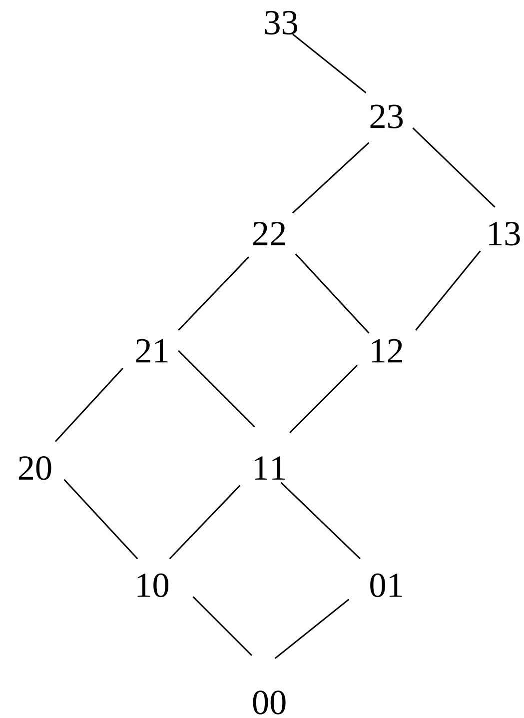


(d)

Detecting General Predicates



(a)



(b)

BFS: 00, 01, 10, 11, 20, 12, 21, 13, 22, 23, 33

DFS: 00, 10, 20, 21, 22, 23, 33, 11, 12, 13, 01

Lexical: 00, 01, 10, 11, 12, 13, 20, 21, 22, 23, 33

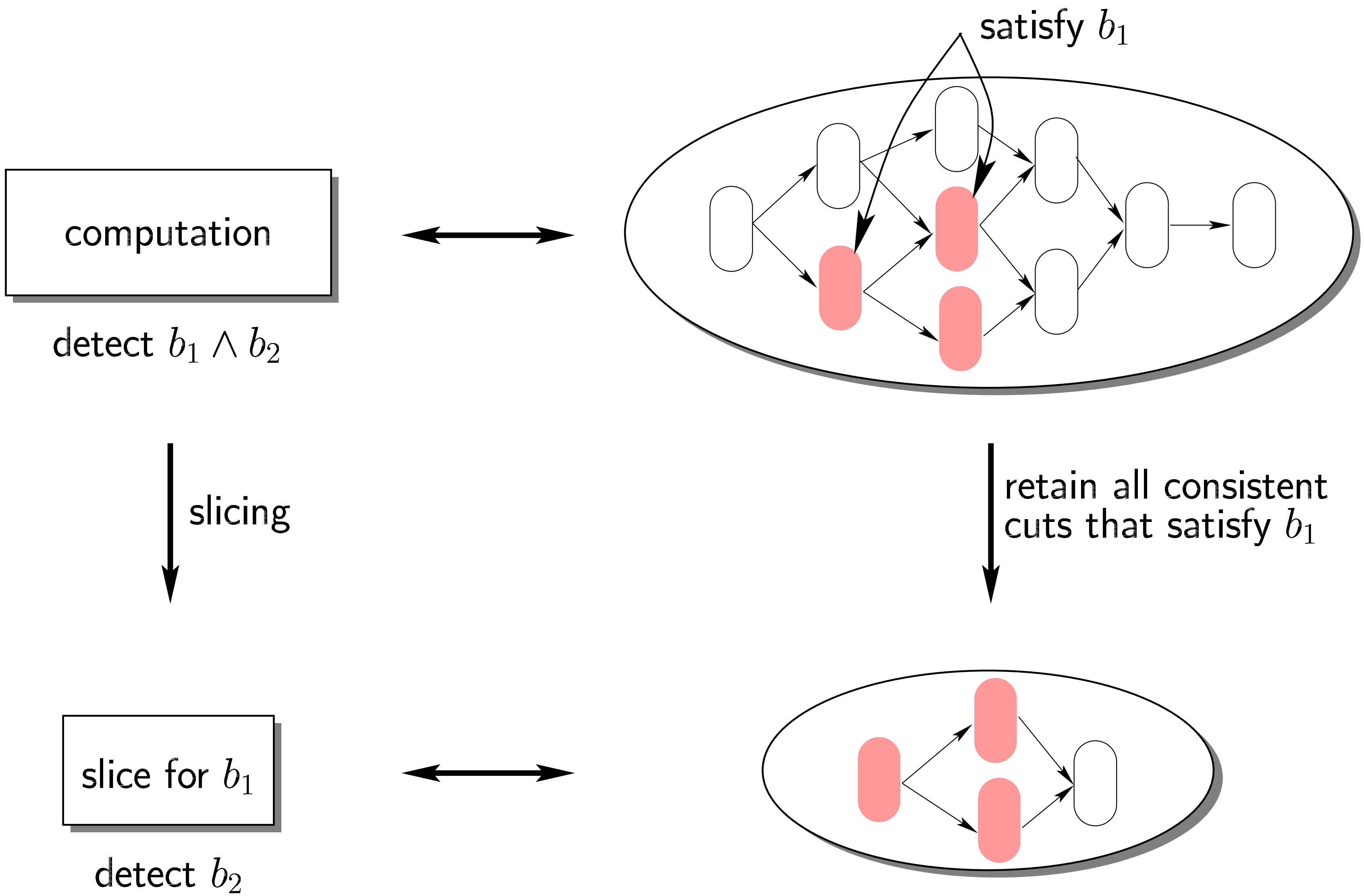
(c)

Enumerate all consistent cuts (ideals) of the poset *breadth first manner* [Cooper and Marzullo 91], *depth first manner* [Alagar and Venkatesan 94], lexical order [Garg 03].

Talk Outline

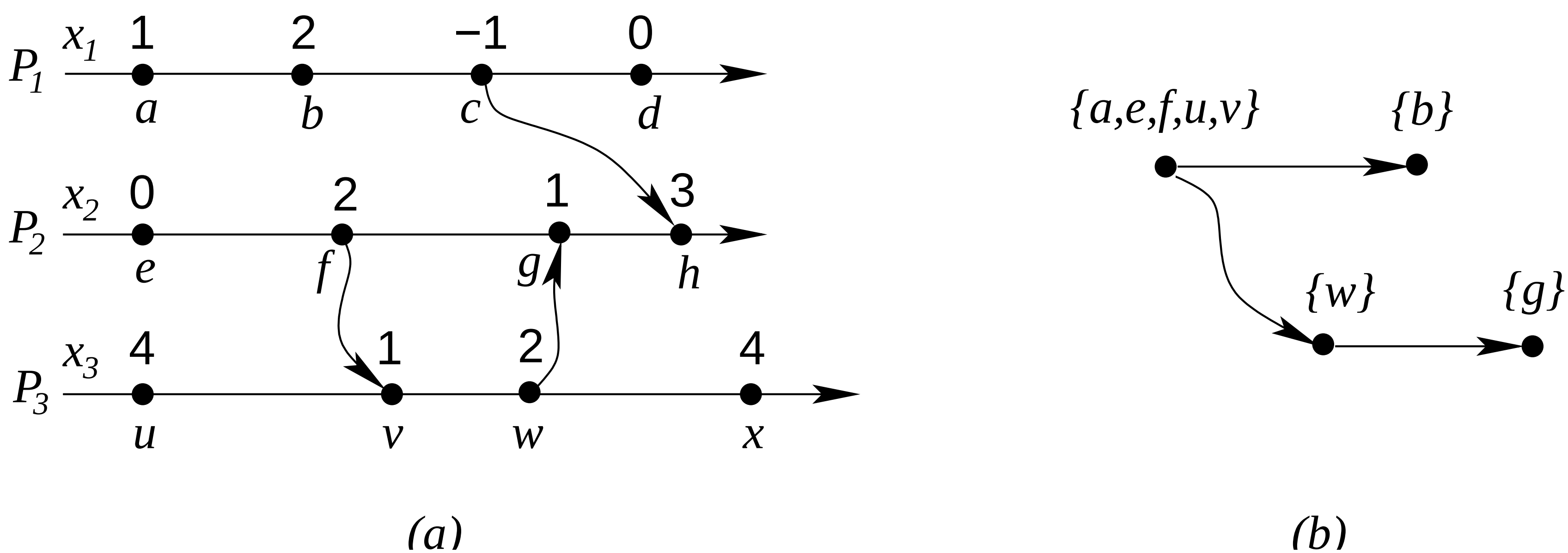
- Happened-Before Relation
- Tracking Dependency
- Detecting Global Predicates
- Computation Slicing: Using Birkhoff's Theorem

Motivation for Computation Slicing



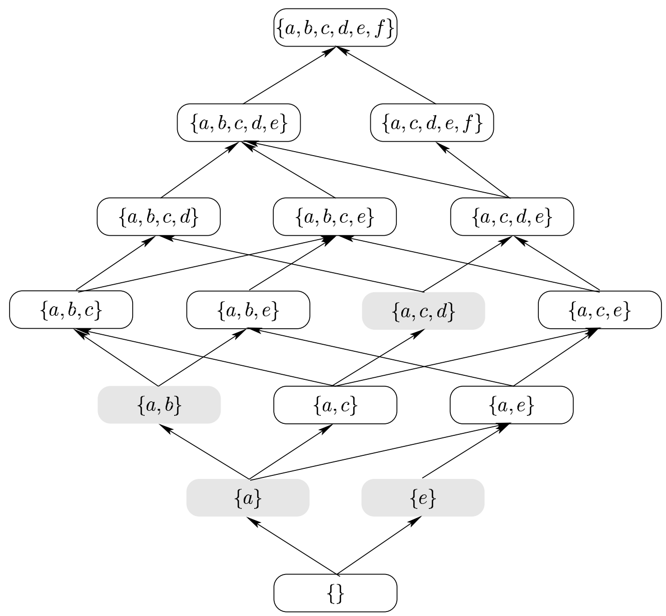
Example

Detect predicate $(x_1 * x_2 + x_3 < 5) \wedge (x_1 \geq 1) \wedge (x_3 \leq 3)$



Slice with respect to $(x_1 \geq 1) \wedge (x_3 \leq 3)$

Computation Slice



Problem Given $(E, \rightarrow)$, and a global predicate B, give the **smallest** sublattice containing B.

Application of Birkhoff's Theorem: The sublattice is distributive and therefore can be represented using its join-irreducible elements.

Results

Theorem: Let L be a FDL generated by the graph P. For every sublattice L', there exists a graph P' obtained by adding edges to P that generates L'.

Efficient algorithms for

- **general predicate:**
  Theorem: Given a computation, if a predicate b can be detected efficiently then the slice for b can also be computed efficiently. [Mittal, Sen and Garg 03]
- **linear predicates:** Direct computation of join-irreducibles [Garg and Mittal 01]
- **Combining slices**: Boolean operators
- **Temporal Logic Operators**: EF, AG, EG

Conclusions and Ongoing Work

Applications:

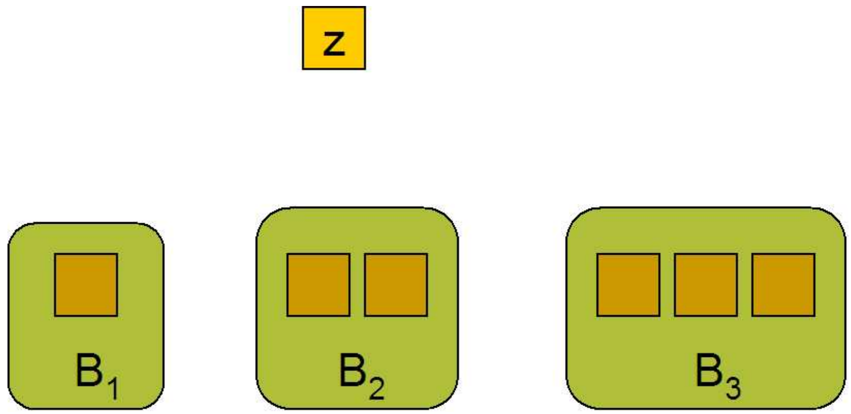
- Tracking Dependency: Chain decomposition, Dimension Theory
- Detecting Global Predicates: Meet-closure, Chain merging, ideal enumeration
- Computation Slicing: Birkhoff's theorem

Ongoing Work

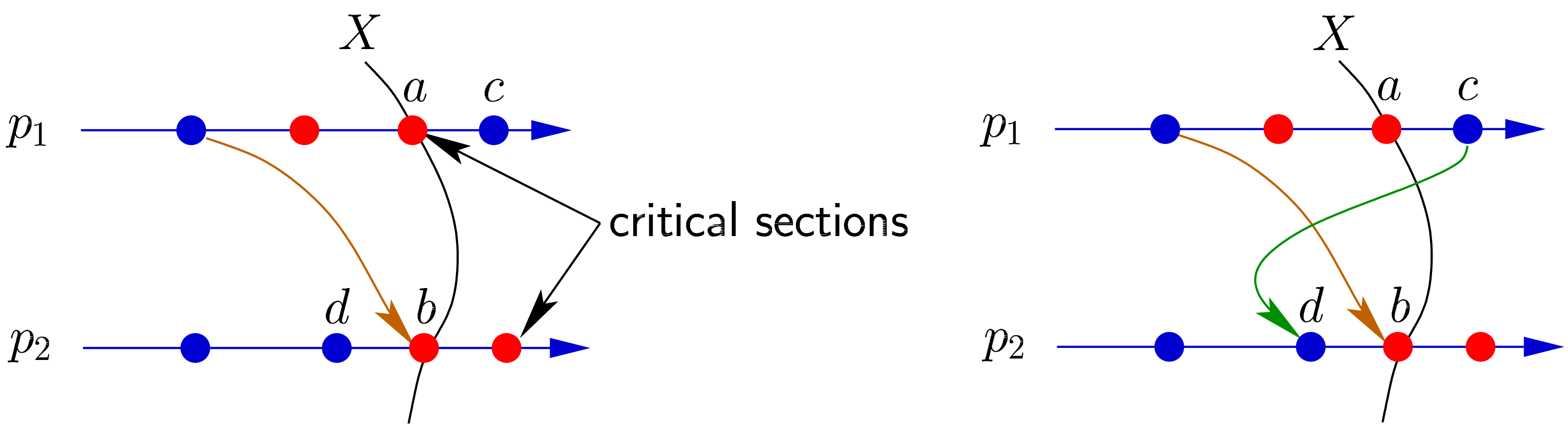
- Checking Temporal Logic Formulas on Infinite Posets
- Multislice Representation of Predicates

Online Chain Decomposition

- Use k sets of queues $B_1, B_2, ..., B_k$. The set B_i has i queues with the invariant that no head of any queue is comparable to the head of any other queue.
- For a new element z, insert it into the first queue q in B_i with its head less than z.
- Swap remaining queues in B_i with queues in B_{i-1}.


z
B1
B2
B3

Controlled Re-execution



Add the synchronization necessary to maintain safety property

e.g., mutual exclusion

Efficient algorithms for computing the synchronization for:

- **Locks** [Tarafdar, Garg DISC98]
  - *time-complexity:* $O(nm)$

- **disjunctive predicate** [Mittal, Garg 00]

  e.g., $(n-1)$-mutual exclusion

  - *time-complexity:* $O(m^2)$
  - minimizes the number of synchronization arrows

- **region predicate** [Mittal, Garg 00]

  e.g., virtual clocks of processes are "approximately" synchronized

  - *time-complexity:* $O(nm^2)$
  - maximizes the concurrency in the controlled computation

n: number of processes, m: number of events