

# Monitoring Multithreaded Distributed Computations

Vijay K. Garg

Department of Electrical and Computer Engineering  
The University of Texas at Austin  
Austin, TX 78712  
email: [garg@ece.utexas.edu](mailto:garg@ece.utexas.edu)

## Motivation: Software Faults

- Software faults are dominant reasons for system outages
- Approx 2 to 3 bugs per 1000 lines of code! [Gray and Reuter 93]

### Testing and Debugging:

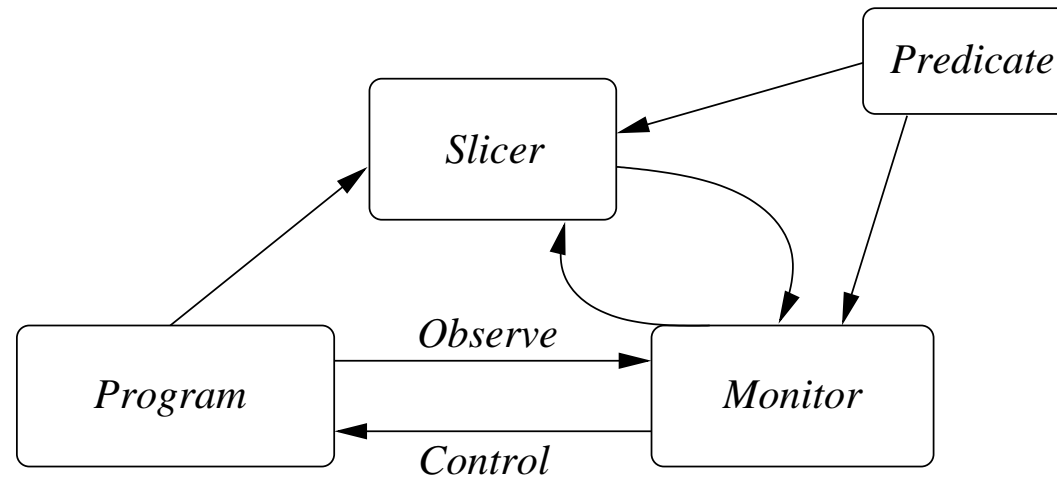
- Distributed programs are prone to errors.
- Traces need to be analyzed to locate bugs.

### Software Fault-Tolerance:

- Software fault detection
- Rollback Recovery

## Our Approach

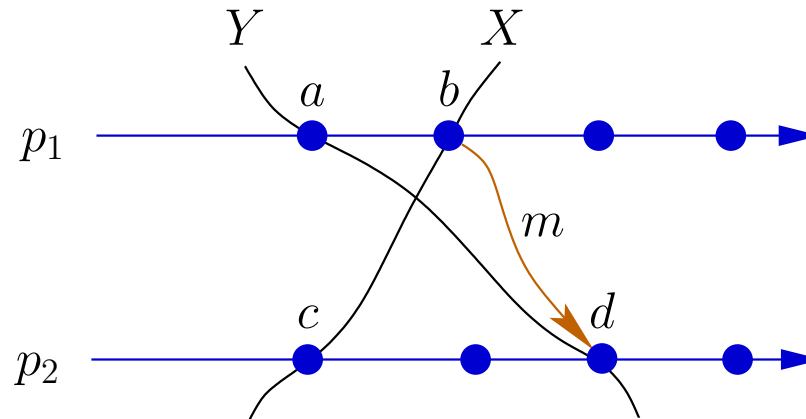
- Three abstractions defined: predicate detection, predicate control and slicing.



## Talk Outline

- Motivation
- Predicate Detection
- Computation Slicing
- Predicate Control
- Ongoing and Future Work

## System Model



**computation:** a set of events ordered by “happened before” relation

**consistent cut:** subset of events that have been executed so far

e.g.,  $X$  is a consistent cut but  $Y$  is not

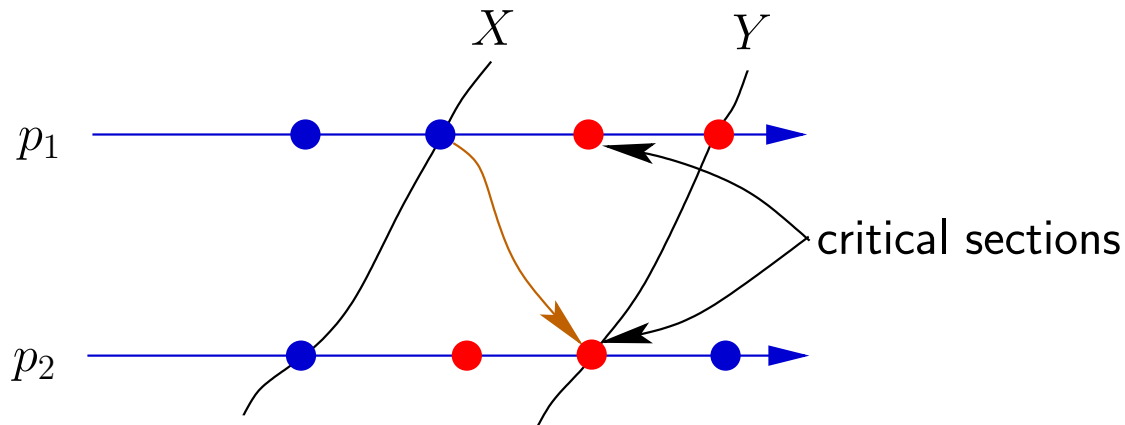
## Predicate Detection Problem

**Predicate:** A global condition expressed using variables on processes  
 e.g., more than one process is in critical section,  
 there is no token in the system

**Problem:** find a consistent cut that satisfies the given predicate

Exponential algorithm for general predicate [Cooper and Marzullo 91]

Polynomial algorithm for conjunctive predicate [Garg and Waldecker 91]

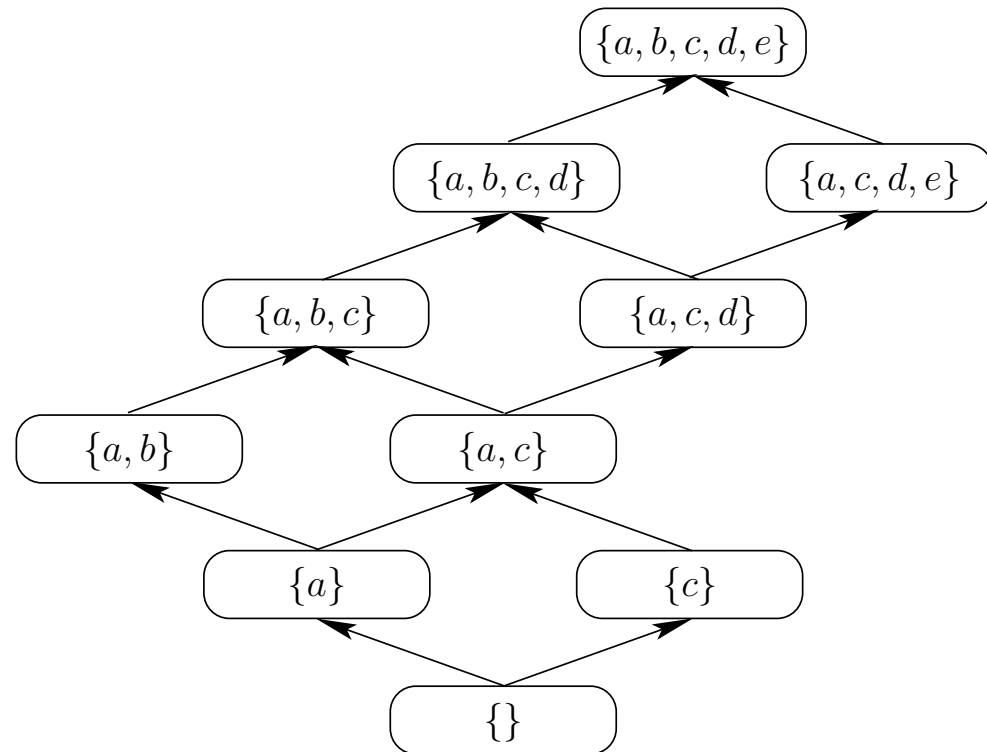
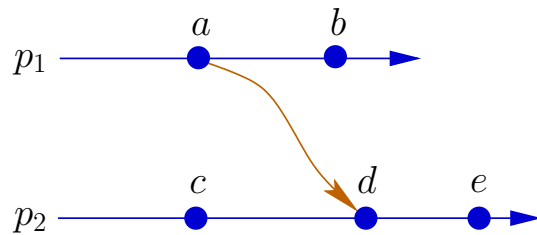


## Motivation for Predicate Detection

*Dear Watson, you see but you do not observe...*

- Distributed Debugging, Testing
  - stop when the predicate  $q$  is true  
predicate  $q = (P1 \text{ is in critical section}) \text{ and } (P2 \text{ is in critical section})$ .
  - Detect if the program violates any invariant
- Fault-tolerance
  - Monitoring while the program is operational
- Distributed Active Rules
  - On global condition  $p$ , trigger rule  $a$
- General paradigm for observing Distributed Algorithms
  - Termination detection, deadlock detection, loss of token

## The Main Difficulty



Too many consistent cuts:

A computation may contain as many as  $O(k^n)$  consistent cuts

$k$ : maximum number of events on a process

$n$ : number of processes



## Predicate Detection for Special Cases

Exploit the structure of the predicate:

- **stable predicate:** [Chandy and Lamport 85]  
once the predicate becomes true, it stays true  
e.g., deadlock
- **unstable predicate:**
  - *observer independent predicate* [Charron-Bost et al 95]  
occurs in one interleaving  $\Rightarrow$  occurs in all interleavings  
e.g., any local predicate
  - *relational predicate:*  $x_1 + x_2 + \dots + x_n \geq k$  [Chase and Garg 95]  
e.g., violation of mutual exclusion
  - *linear predicate* [Chase and Garg 95]  
closed under intersection, e.g., conjunctive predicates such as there is no leader in the system

## Conjunctive Predicates

A predicate that can be expressed as  $l_1 \wedge l_2 \wedge \dots \wedge l_n$ , where  $l_i$  is local to  $P_i$ .

Examples:

- **mutual exclusion problem**: (P1 in CS) and (P2 in CS)
- **missing primary**: (P1 is secondary) and (P2 is secondary) and (P3 is secondary)

Detect errors that may be hidden in some run due to race conditions.

## Importance of Conjunctive Predicates

Sufficient for detection of any global

- **boolean expression of local predicates** which can be expressed as a disjunction of a small number of conjunctions.

*Example:*  $x, y$  and  $z$  are in three different processes. Then,

$$\text{even}(x) \wedge ((y < 0) \vee (z > 6))$$

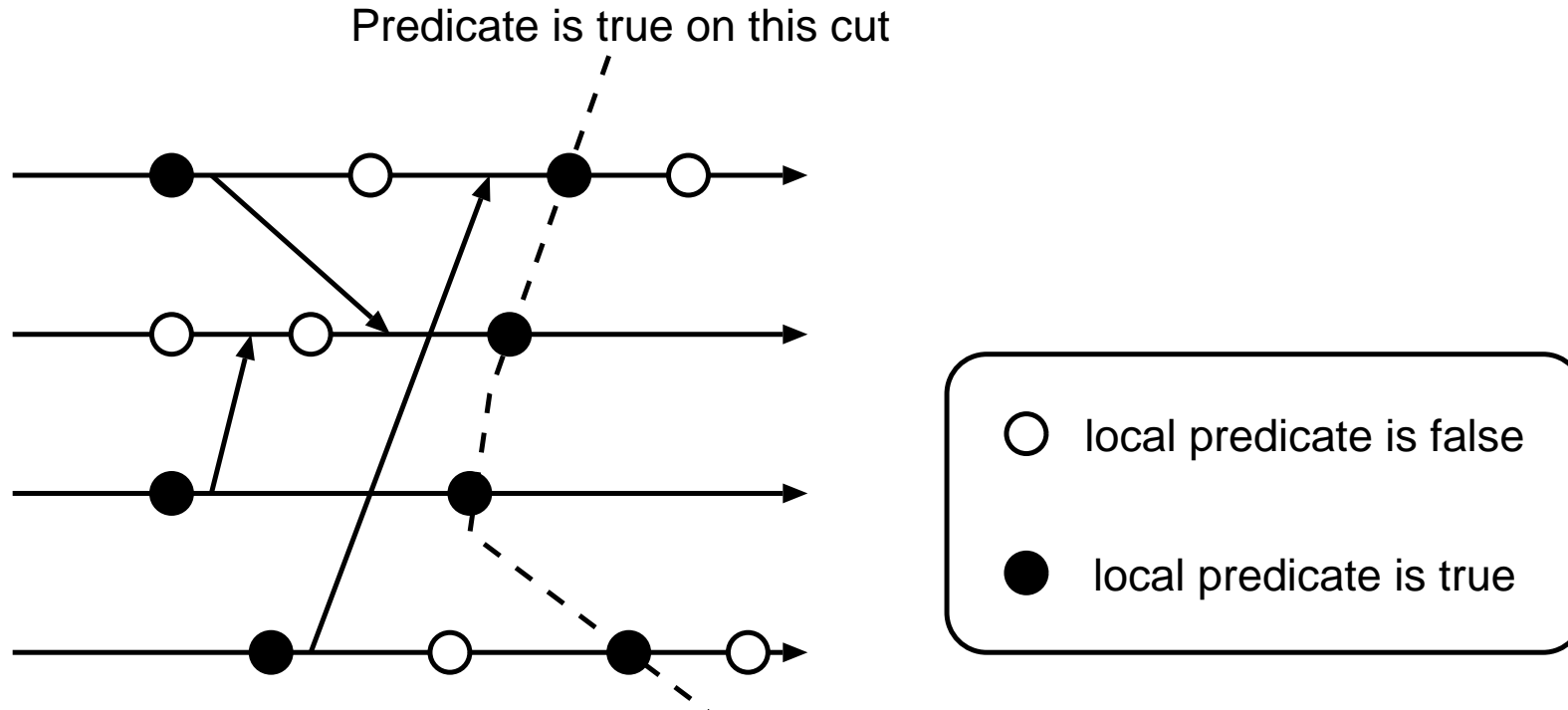
$\equiv$

$$(\text{even}(x) \wedge (y < 0)) \vee (\text{even}(x) \wedge (z > 6))$$

- **predicate satisfied by only a small number of values** *Example:*  $x$  and  $y$  are in different processes.

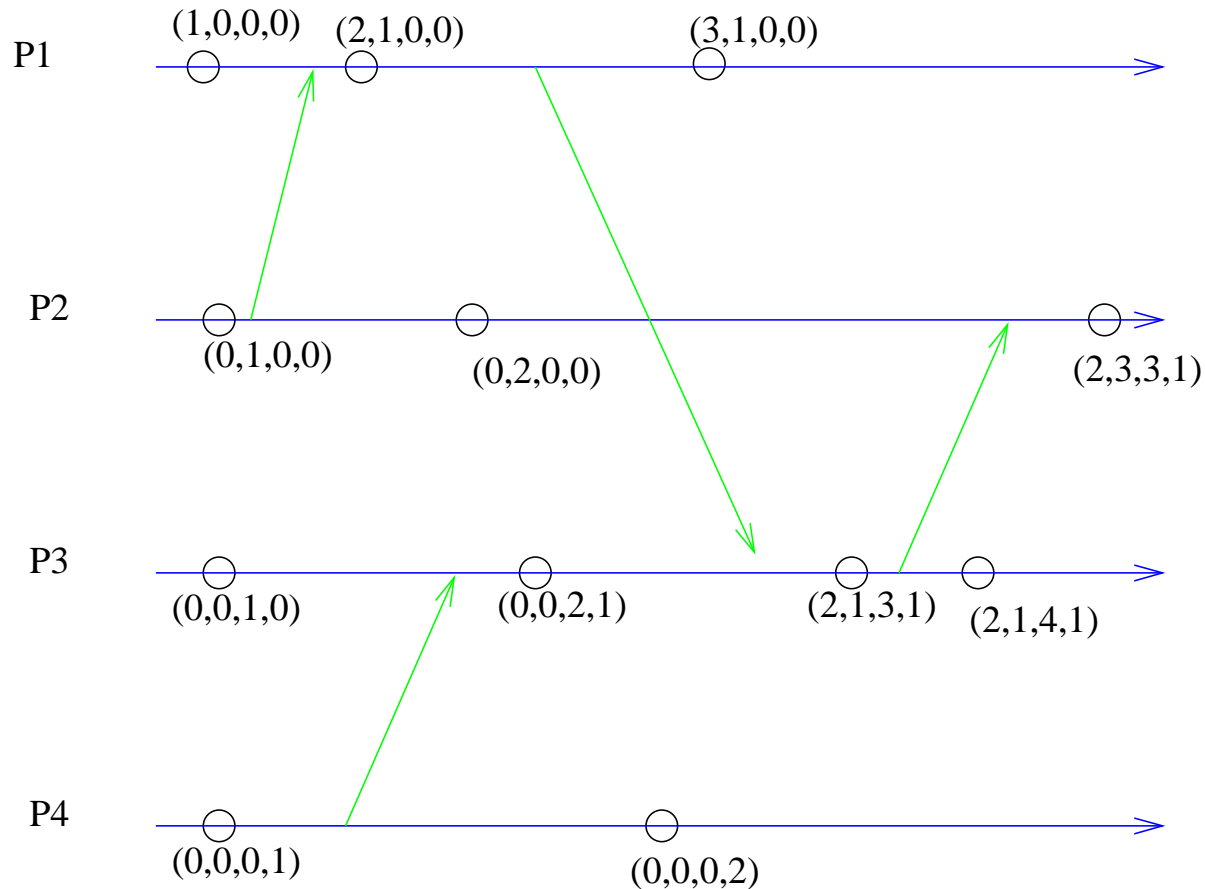
$(x = y)$  is not a *local* predicate but  $x$  and  $y$  are binary.

## Conditions for Conjunctive Predicates



Possibly  $(l_1 \wedge l_2 \wedge \dots \wedge l_n)$  is true **iff** there exist  $s_i$  in  $P_i$  such that  $l_i$  is true in state  $s_i$ , and  $s_i$  and  $s_j$  are incomparable for distinct  $i, j$ .

## Tracking Causality: Clocks in a Distributed System



**Result:**  $s$  happened before  $t$  iff the vector at  $s$  is less than the vector at  $t$ .  
 Vector Clocks [Fidge 89, Mattern 89]

## Dynamic Chain Clocks

Problem with vector clocks: scalability

Idea: Computing the “chains” in an online fashion [Aggarwal and Garg 04]

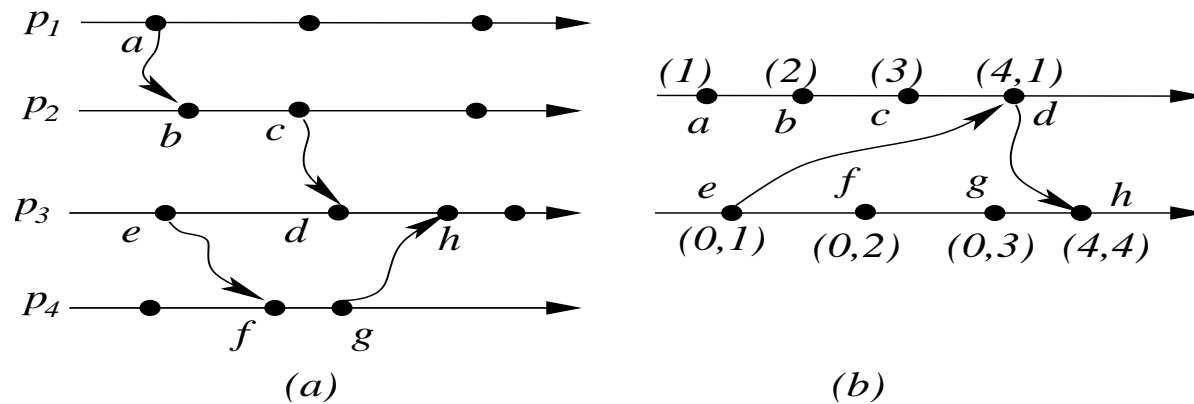
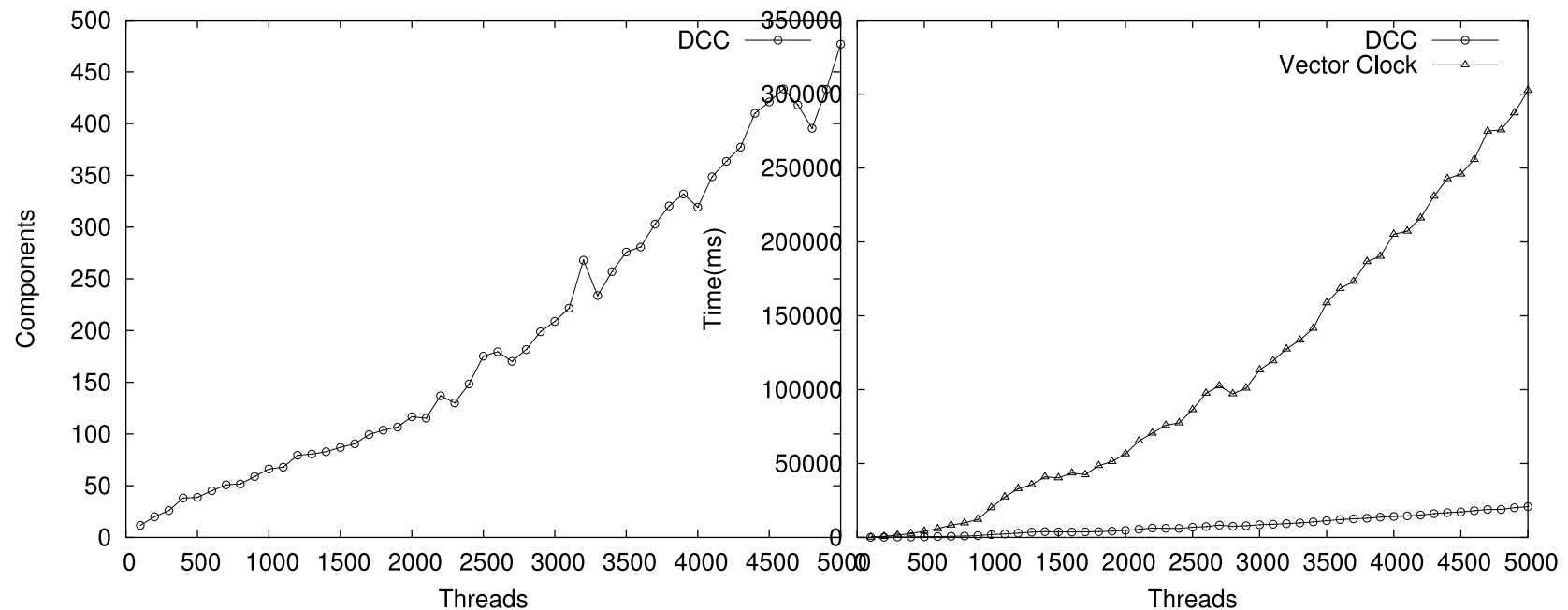


Figure 1: (a) A computation with 4 processes (b) The relevant subcomputation

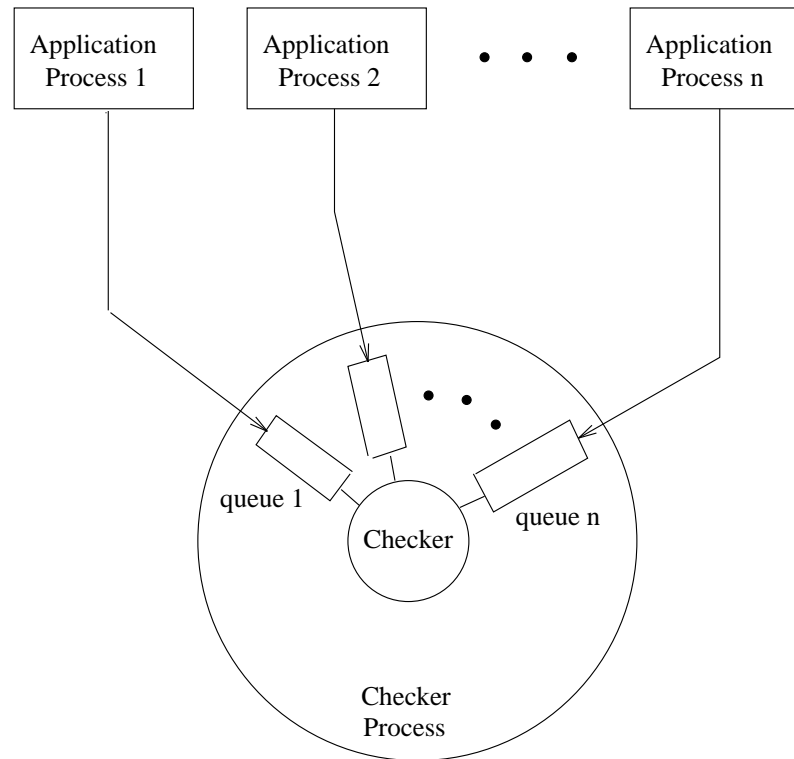
## Experimental Results

Simulation of a computation with 1% relevant events  
Measured

- number of components vs number of threads
- total time overhead vs number of threads



## Weak Conjunctive Predicates: Centralized Algorithm



[Garg and Waldecker 92] Each non-checker process maintains its local *vector* and sends to the checker process the chain clock whenever

- local predicate is true
- at most once in each message interval.



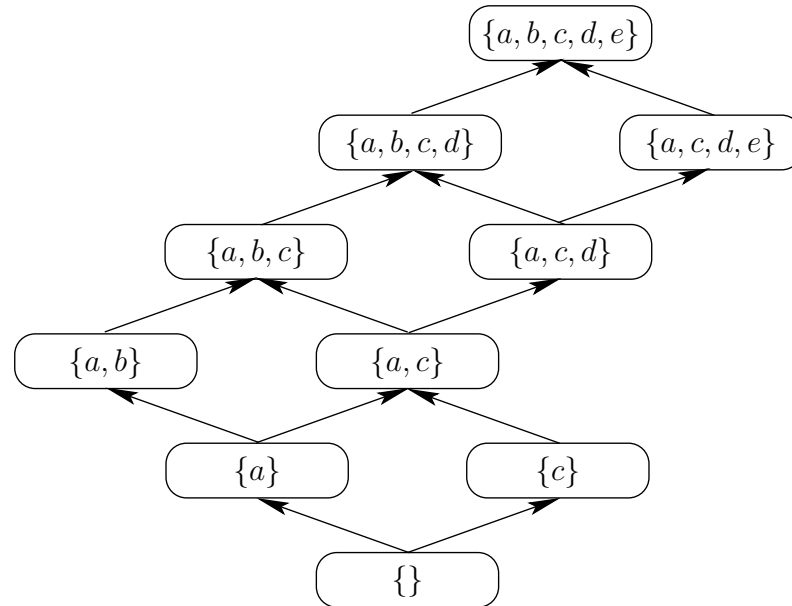
## Overhead: Checker processes

Space complexity:  $n$  queues, each containing at most  $m$  vectors

Time complexity:

- The algorithm for checker requires at most  $O(n^2m)$  comparisons.
- Any algorithm which determines whether there exists a set of incomparable vectors of size  $n$  in  $n$  chains of size at most  $m$ , makes at least  $mn(n - 1)/2$  comparisons. [Garg and Waldecker 94]

## Predicate Detection in General



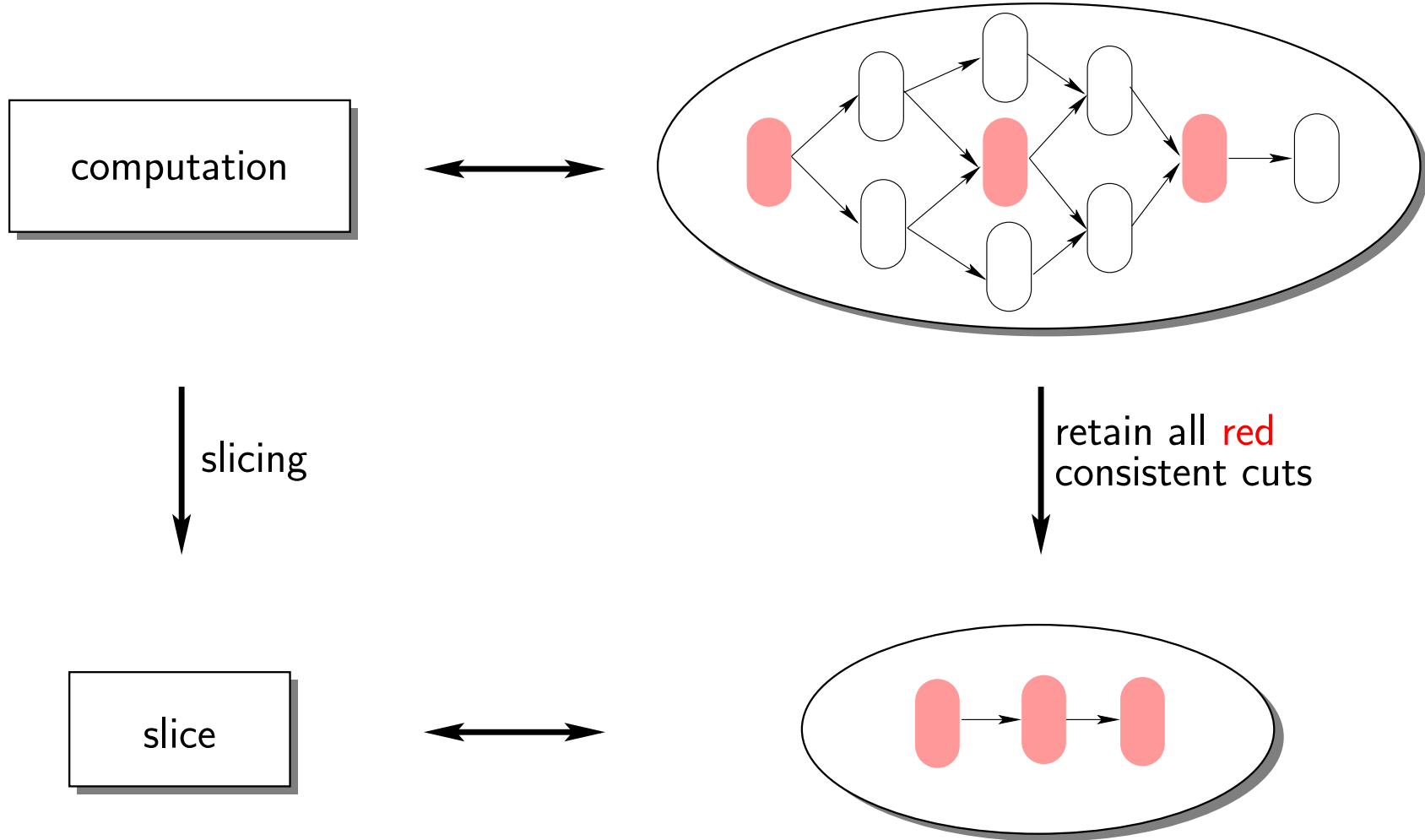
Construct the state-space (need to examine all consistent cuts)

- *breadth first manner* [Cooper and Marzullo 91]  
space-complexity: number of consistent cuts
- *depth first manner* [Alagar and Venkatesan 94]  
space-complexity: number of events
- *lexical order* [Garg 03]  
space-complexity: number of processes

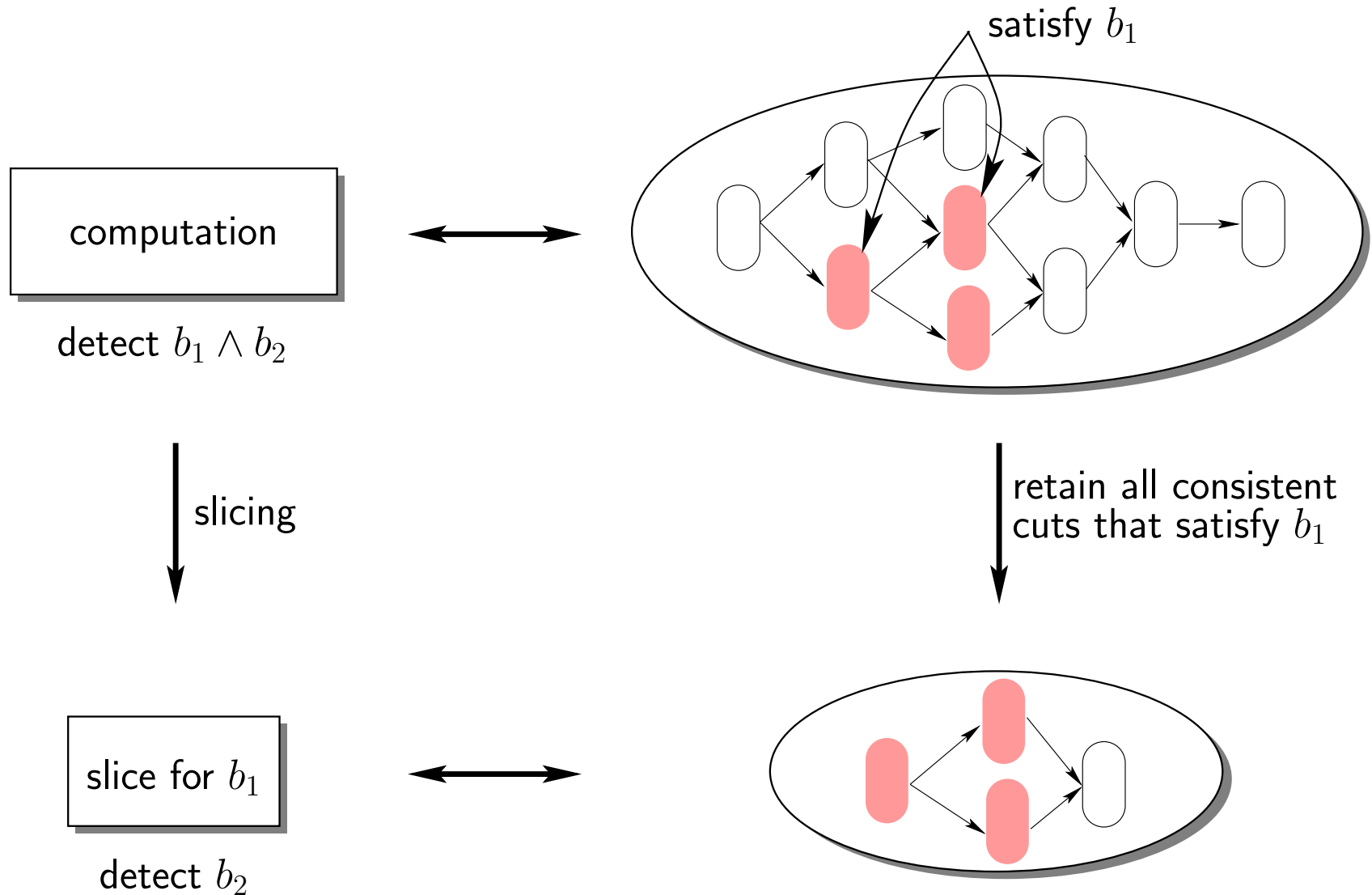
## Talk Outline

- Motivation
- Predicate Detection Problem
- **Computation Slicing**
- Predicate Control
- Ongoing and Future Work

# The Main Idea of Computation Slicing

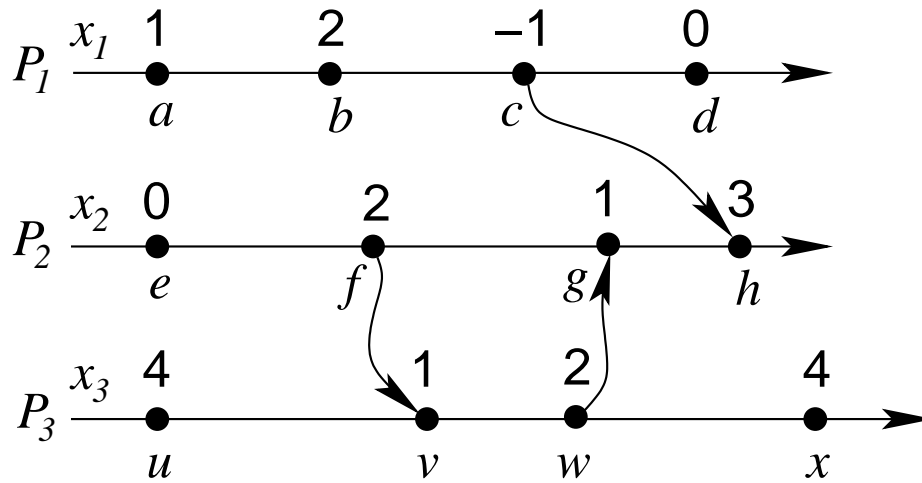


## How does Computation Slicing Help?

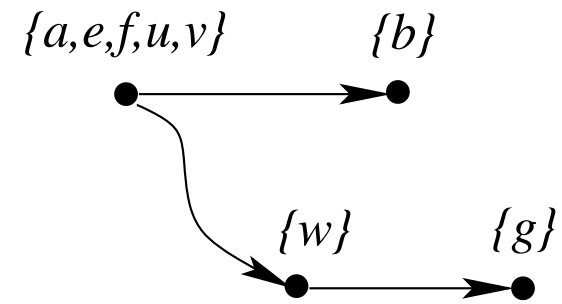


## Example

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



(a)



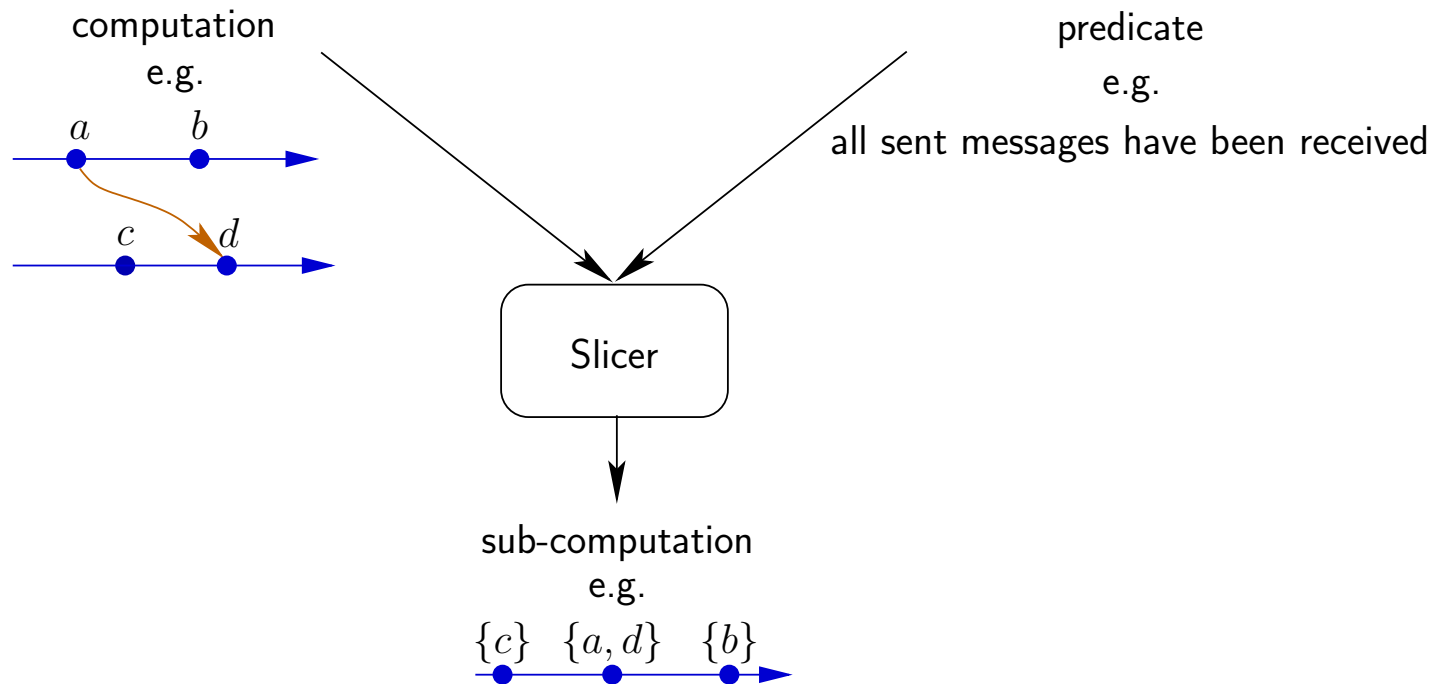
(b)

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

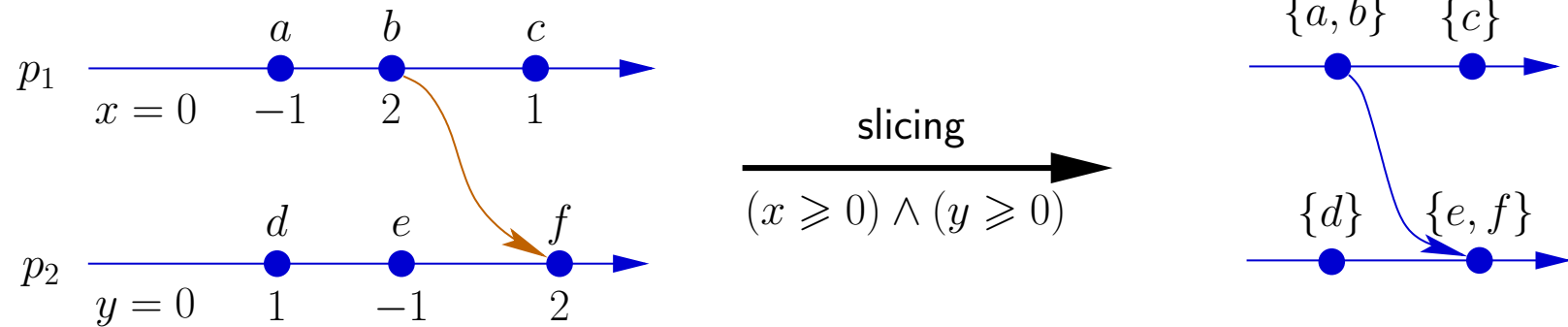
## Computation Slice

**computation slice:** a sub-computation such that: [Mittal and Garg 01]

- (1) it contains **all** consistent cuts of the computation satisfying the given predicate, and
- (2) it contains the **least** number of consistent cuts

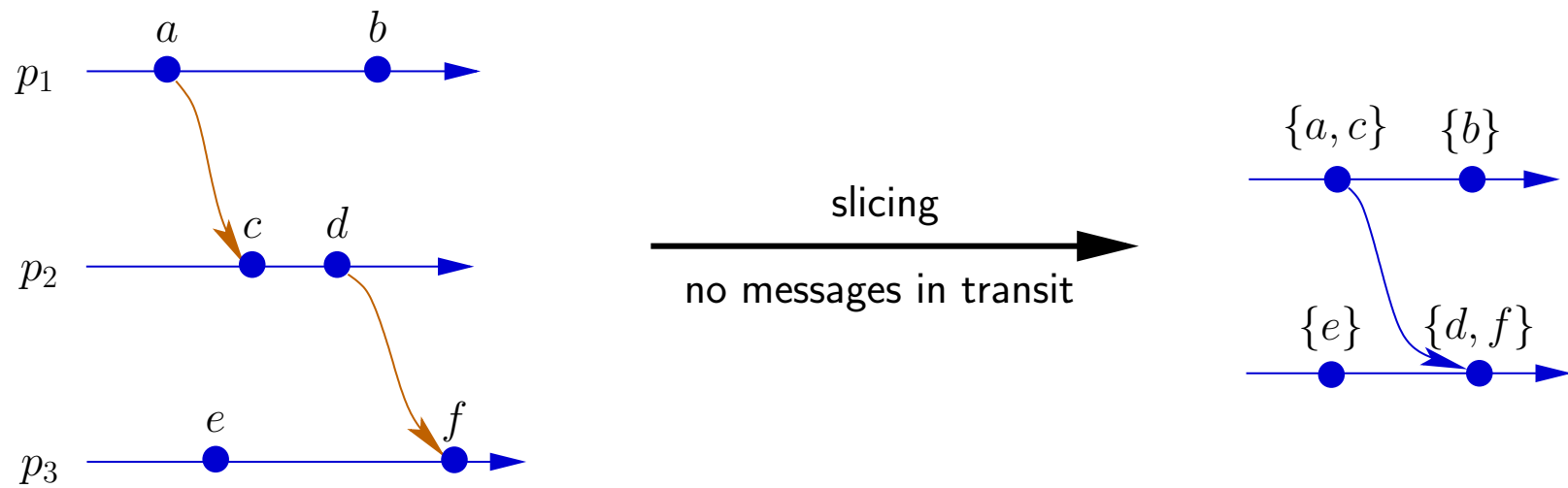


## Example: Conjunctive Predicate





## Example: Channel Predicate



## Regular Predicate

**regular predicate:** the set of consistent cuts satisfying the predicate is closed under intersection and union [Garg and Mittal 01]

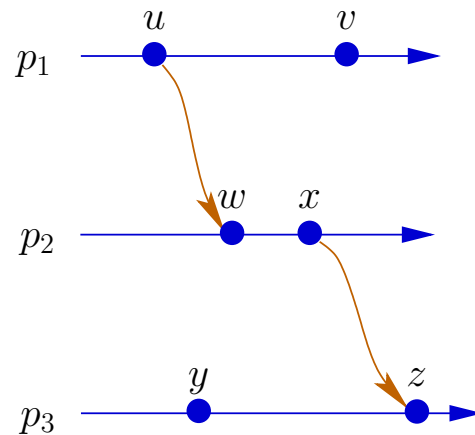
Examples:

- conjunctive predicate—conjunction of local predicates
- there are at most (or at least)  $k$  messages in transit from process  $i$  to process  $j$
- every “request” message has been “acknowledged” in the system

The class of regular predicates is *closed* under conjunction:

If  $b_1$  and  $b_2$  are regular predicates then so is  $b_1 \wedge b_2$

## Computing the Slice for Regular Predicate



$b =$  “no messages in transit”

Algorithm:

Step 1: Compute the least consistent cut  $L$  that satisfies  $b$

$$L = \{\}$$

Step 2: Compute the greatest consistent cut  $G$  that satisfies  $b$

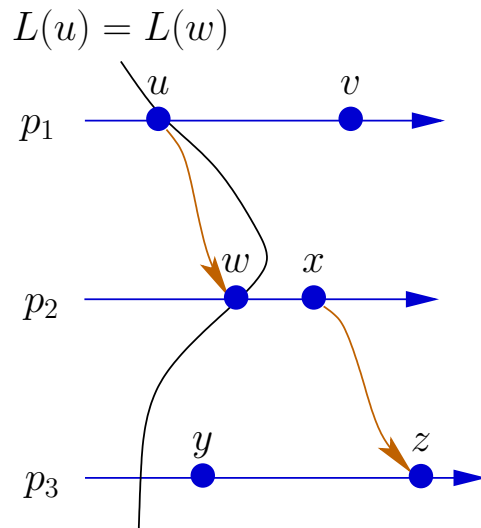
$$G = \{u, v, w, x, y, z\}$$

## Computing the Slice for Regular Predicate

Algorithm:

Step 3: For every event  $e \in G - L$ , compute  $L(e)$  defined as:

- (1)  $L(e)$  contains  $e$
- (2)  $L(e)$  satisfies  $b$
- (3)  $L(e)$  is the least consistent cut satisfying (1) and (2)



$$L(u) = \{u, w\}$$

$$L(v) = \{u, v, w\}$$

$$L(w) = \{u, w\} \text{ (duplicate)}$$

$$L(x) = \{u, w, x, y, z\}$$

$$L(y) = \{y\}$$

$$L(z) = \{u, w, x, y, z\} \text{ (duplicate)}$$

## Results

Efficient polynomial-time algorithms for computing the slice for:

- **regular predicate:** [Garg and Mittal 01]

*time-complexity:*

- general:  $O(n^2m)$
- special cases (e.g., conjunctive predicate):  $O(m)$

- **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 02]

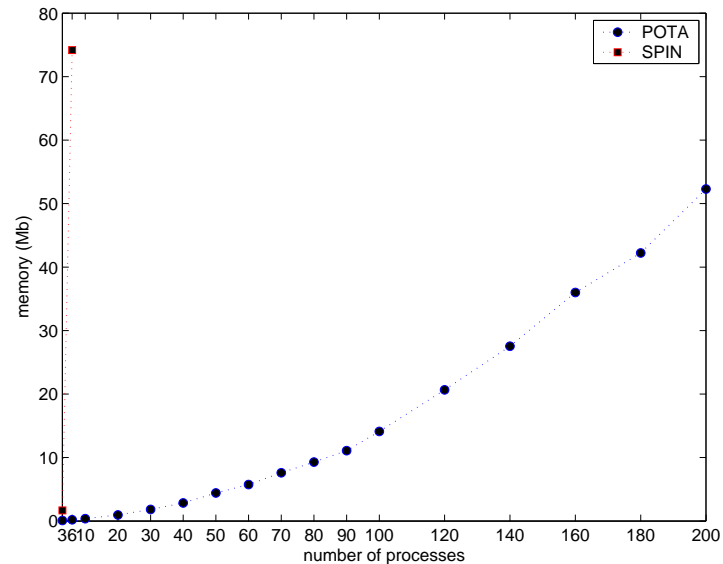
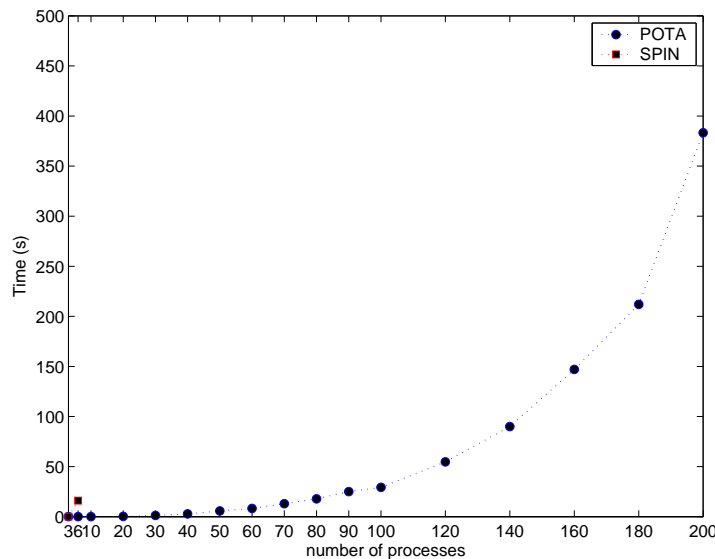
$n$ : number of processes

$m$ : number of events

## Experimental Evaluation: Dining Philosophers Verification

**POTA**: Partial Order Trace Analyzer (based on slicing) [Sen and Garg 03]

**SPIN**: A widely used model checking tool [Holzmann 97]



**SPIN**: 250 seconds for  $n = 6$ , runs out of memory for  $n > 6$ .

**POTA**: can handle  $n = 200$ . Used 400 seconds.

**Predicate**: Two neighboring dining philosophers do not eat concurrently

## Talk Outline

- Motivation
- Predicate Detection Problem
- Computation Slicing
- **Predicate Control**
- Ongoing and Future Work

## Motivation for Control

*Who controls the past controls the future, who controls the present controls the past...*

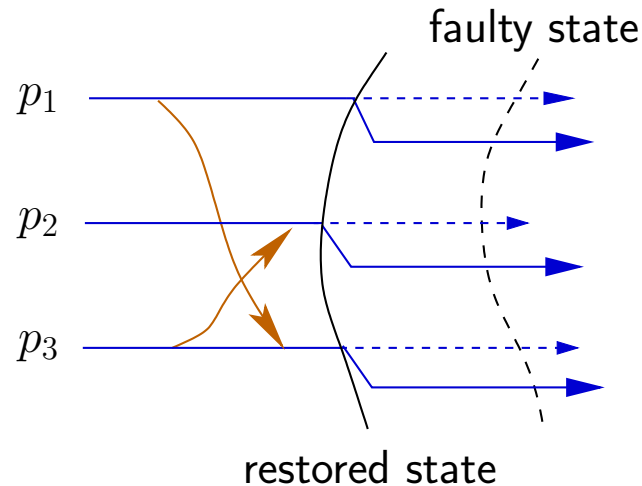
*George Orwell,*

*Nineteen Eighty-Four.*

- maintain global invariants or proper order of events  
Examples: Distributed Debugging
  - ensure that  $busy_1 \vee busy_2$  is always true
  - ensure that  $m_1$  is delivered before  $m_2$
- Resource Allocation
  - maintain  $\neg CS_1 \vee \neg CS_2$
- Fault tolerance
  - On fault, rollback and execute under control
- Adaptive policies



## Rollback Recovery for Software Faults

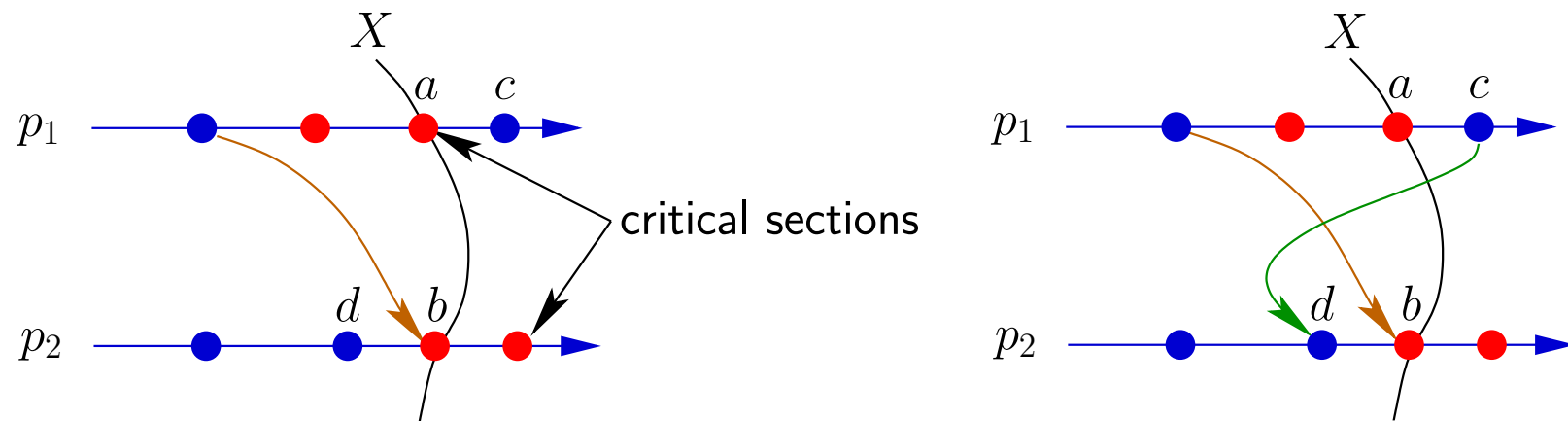


### Re-execution Problem:

To re-execute in order to avoid a recurrence of a previously detected failure

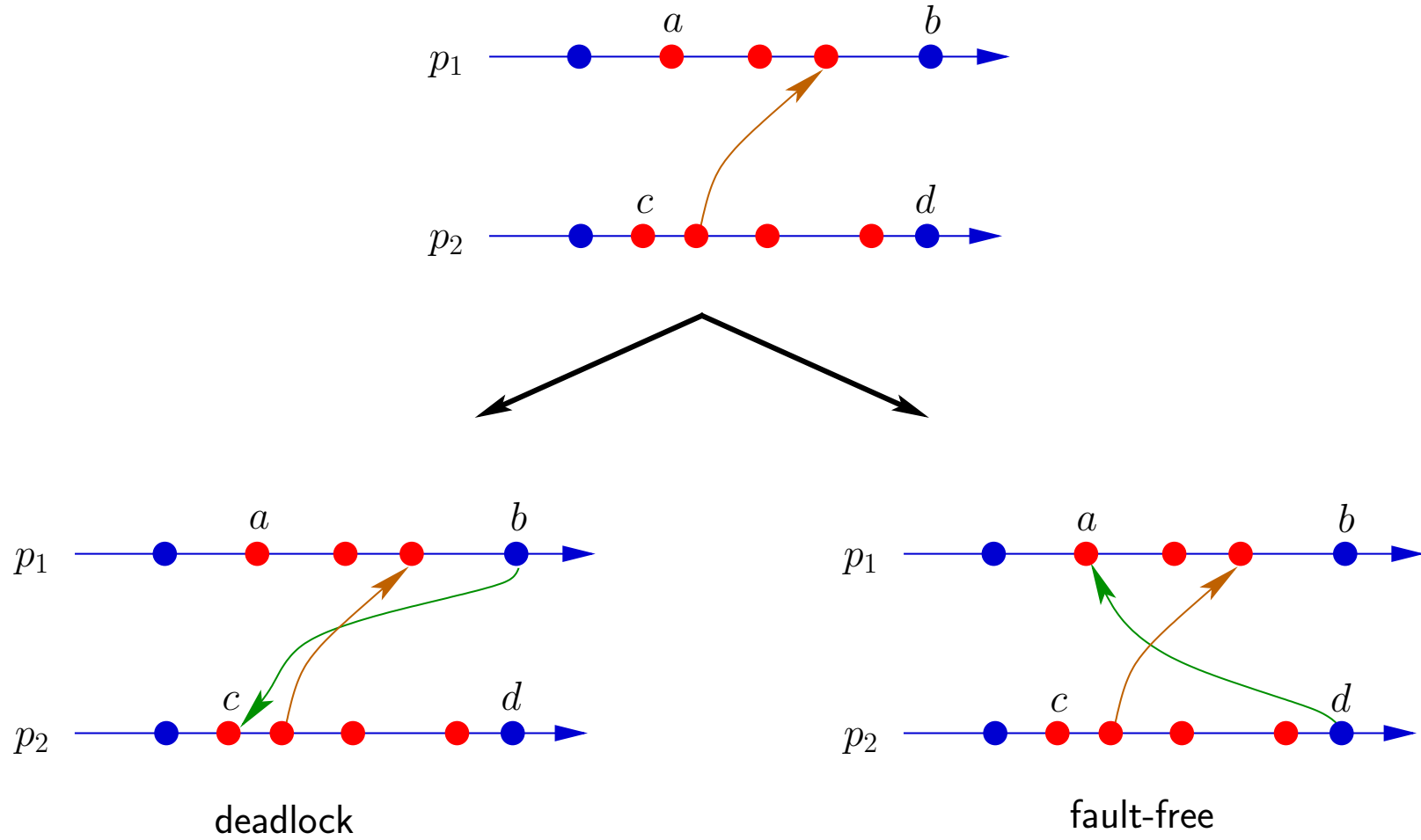
- Progressive Retry [Wang *et al* 97]
- Controlled Re-execution [Tarafdar and Garg 98]

## Controlled Re-execution



Add the synchronization necessary to maintain safety property  
e.g., mutual exclusion

# The Main Difficulty



## Results

Efficient algorithms for computing the synchronization for:

- **Locks** [Tarafdar and Garg 98]
  - *time-complexity*:  $O(nm)$
- **disjunctive predicate** [Mittal and Garg 00]
  - e.g.,  $(n - 1)$ -mutual exclusion
  - *time-complexity*:  $O(m^2)$
  - minimizes the number of synchronization arrows
- **region predicate** [Mittal and 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

## Ongoing and Future Work

- **Predicate Detection**: Temporal logic predicates (**Anurag Agarwal**), Online relational predicates (**Selma Ikiz**)
- **Slicing**: Distributed and online algorithms (**Vinit Ogale**)
- **Predicate Control**: Controlling message order (**Arindam Chakraborty**)
- **Model Checking**: Based on predicate detection and slicing (**Sujatha Kashyap**)

## Acknowledgments

Collaborators on various ideas **Anurag Agarwal** **Craig M. Chase**, **Eddy Fromentin**, **Richard Kilgore**, **Ratnesh Kumar**, **James R. Mitchell**, **Neeraj Mittal**, **Venkat V. Murty**, **Michel Raynal**, **Alper Sen**, **Chakarat Skawratonand**, **Ashis Tarafdar**, **Alex I. Tomlinson**, and **Brian Waldecker**