# Mining Temporal Invariants from Partially Ordered Logs


Ivan Beschastnikh
Yuriy Brun
Michael D. Ernst
Arvind Krishnamurthy
Thomas E. Anderson

University of Washington

# Motivating question

**I am a developer.**

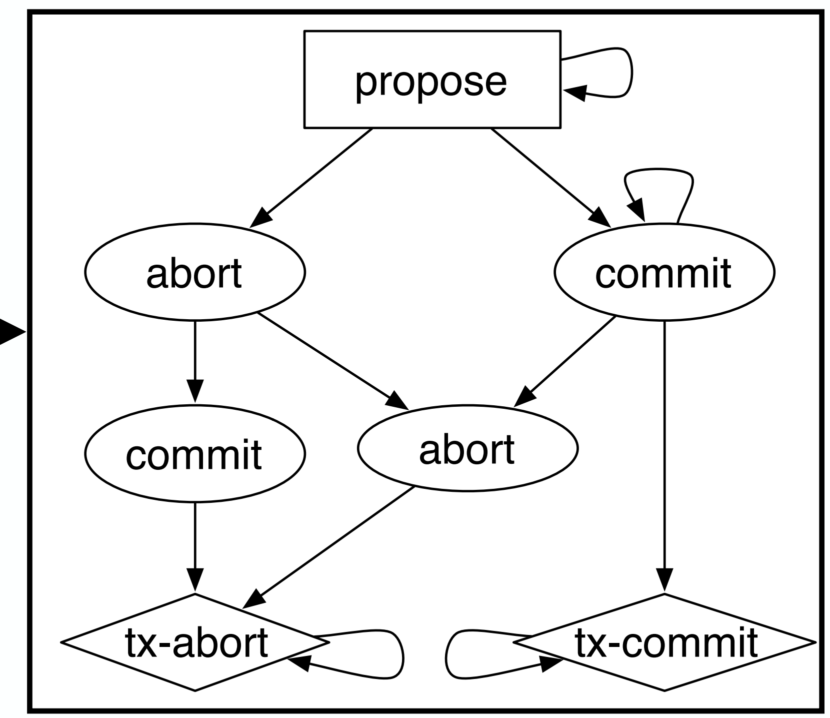
**Why does my system behave in a certain manner?**

# Synoptic (our prior work)

Input

## Synoptic

A tool that mines FSM models from logs




Output

# But, what if the question is ...

**Why does my concurrent system behave in a certain manner?**

# Log analysis of concurrent systems

- Concurrency is widespread and is becoming commonplace (Hadoop, Ajax, Multicore)
- Many log analysis tools exist to help understand sequential, but not concurrent systems
  - Assume totally ordered logs
  - Cannot reason about concurrent executions
  - Insufficient for debugging concurrency issues

# Log analysis of concurrent systems

- Concurrency is widespread and is becoming commonplace (Hadoop, Ajax, Multicore)
- Many log analysis tools exist to help understand

**Need to develop tools for concurrent systems logs**

  - Cannot reason about concurrent executions
  - Insufficient for debugging concurrency issues

# Our approach

- Mine the partially ordered log to extract temporal invariants between events
  - Capture the essence of what happened
  - Simple to understand
- Show invariants to the developer
  - May notice missing invariants
  - May find unexpected invariants
- Developer modifies and re-runs the system

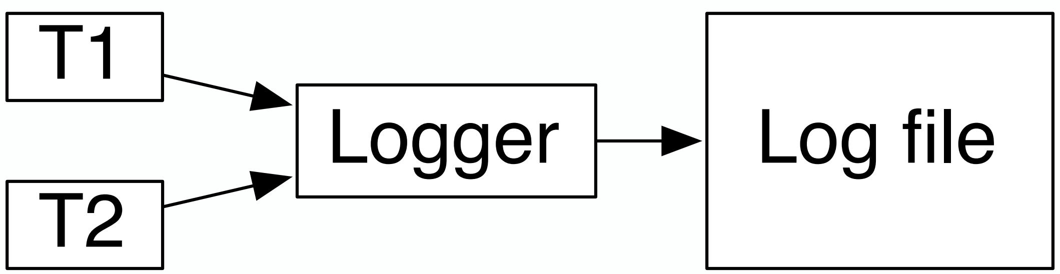
# Outline

- Motivation
- Why a total order is not enough
- Mining temporal invariants from concurrent executions
- Tool demo
- Two algorithms to mine temporal invariants
- Algorithms’ scalability evaluation

# Limitations of total order

- A system with two threads: T1, T2
  - T1 generates event a, T2 generates event b

- Logging pipeline:



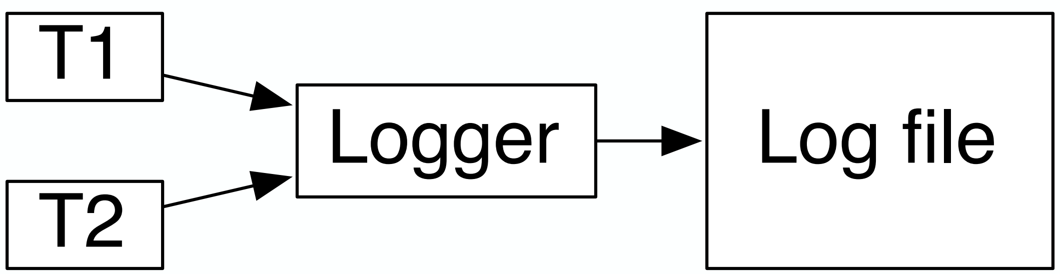

- Generated log file:

| | |
|---|---|
| 1 | a |
| 2 | b |

# Limitations of total order

- A system with two threads: T1, T2
  - T1 generates event a, T2 generates event b

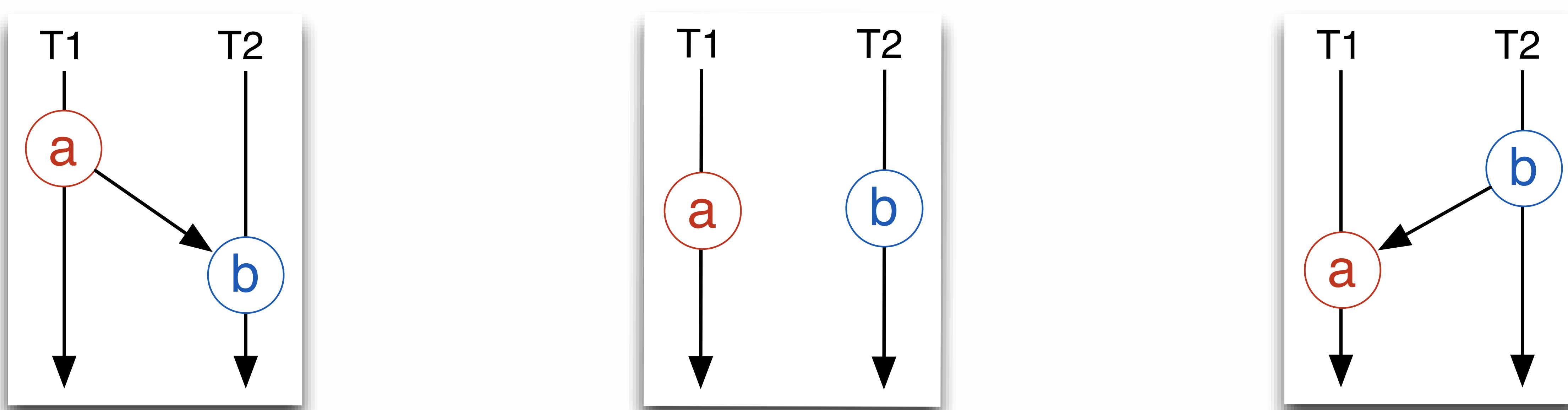
- Logging pipeline:




- Generated log file:

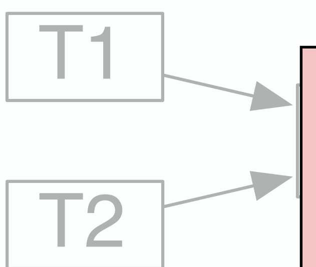
| | |
|---|---|
| 1 | a |
| 2 | b |

Which of these three systems generated the log?



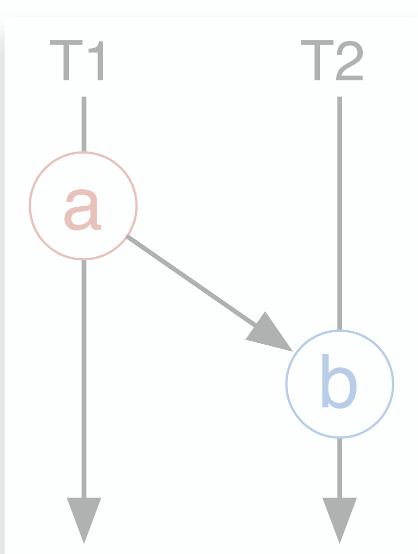
# Limitations of total order

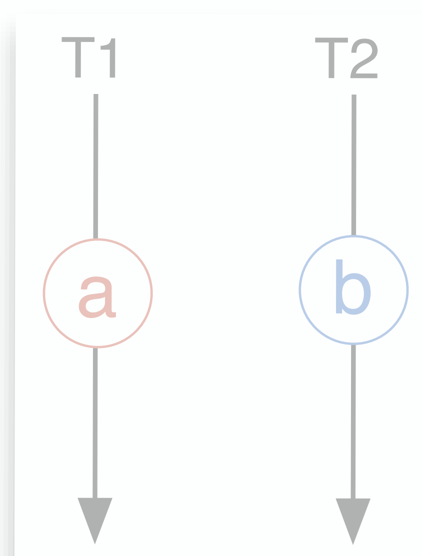
- A system with two threads: T1, T2
  - T1 generates event a, T2 generates event b
- Logging pipeline:

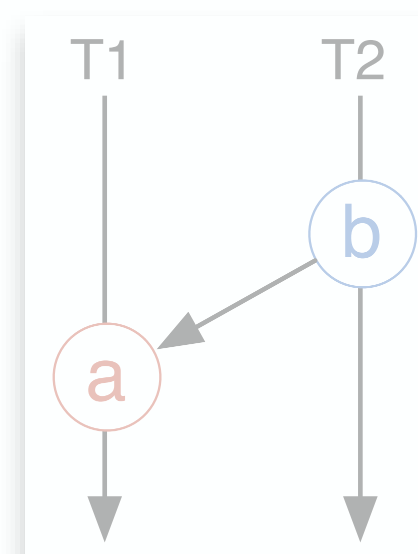



- Generated log file:

**A totally ordered log is insufficient.**



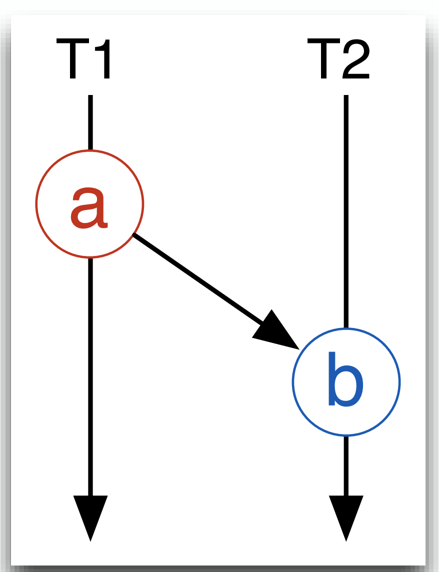



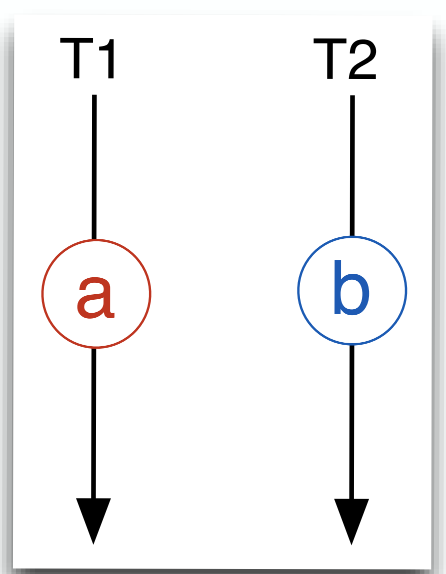
# Logging the partial order

- We know how to do this
  - Lamport defined the happens-before relation in 1978
  - Operationalized with vector clocks in 1988, 1989

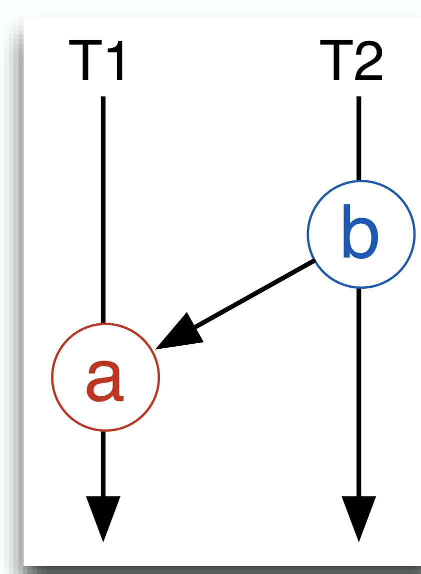
| | |
|---|---|
| [1,0] | a |
| [1,1] | b |




| | |
|---|---|
| [1,0] | a |
| [0,1] | b |

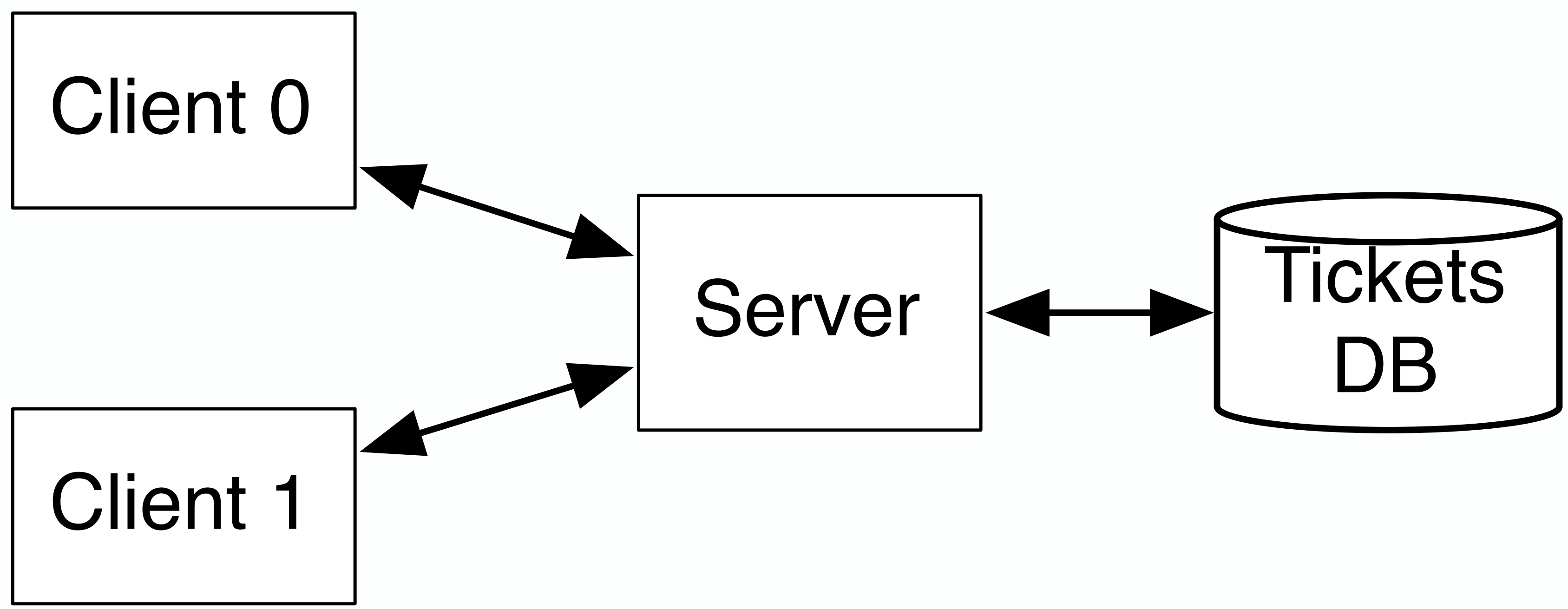



| | |
|---|---|
| [1,1] | a |
| [0,1] | b |



# Example system

- A server with tickets, two clients who buy tickets
- Each client checks availability of tickets and then buys a ticket

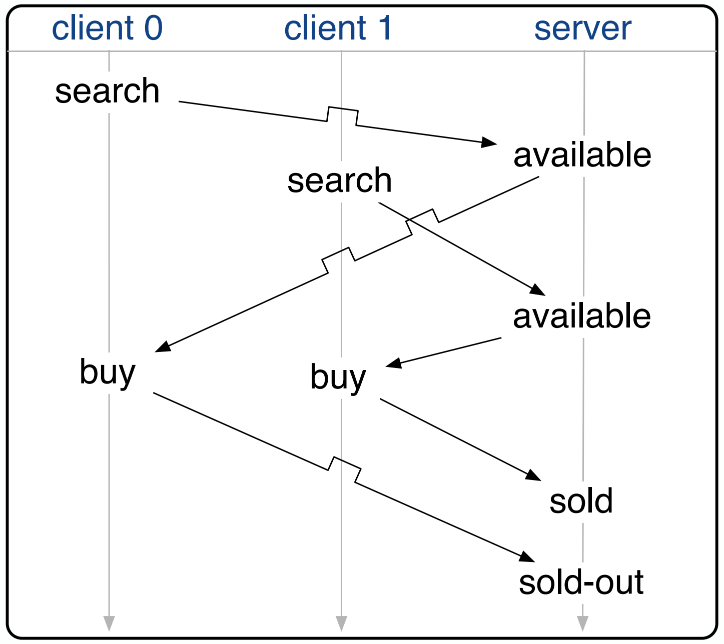


# Partial order is complex

## Partially ordered log :

[1,0,0] client 0: search for tickets to Portugal for 23/10/11
[0,1,0] client 1: search for tickets to Portugal for 23/10/11
[1,0,1] server: there is a ticket available for 505P
[1,1,2] server: there is a ticket available for 505P
[2,0,1] client 0: buy ticket
[2,1,3] server: sold
[1,2,2] client 1: buy ticket
[2,2,4] server: tickets sold out

## Execution:



# Partial order is complex

## Partially ordered log :

[1,0,0] client 0: search for tickets to Portugal for 23/10/11
[0,1,0] client 1: search for tickets to Portugal for 23/10/11
[1,0,1] server: there is a ticket available for 505P
[1,1,2] server: there is a ticket available for 505P
[2,0,1] client 0: buy ticket
[2,1,3] server: sold
[1,2,2] client 1: buy ticket
[2,2,4] server: tickets sold out

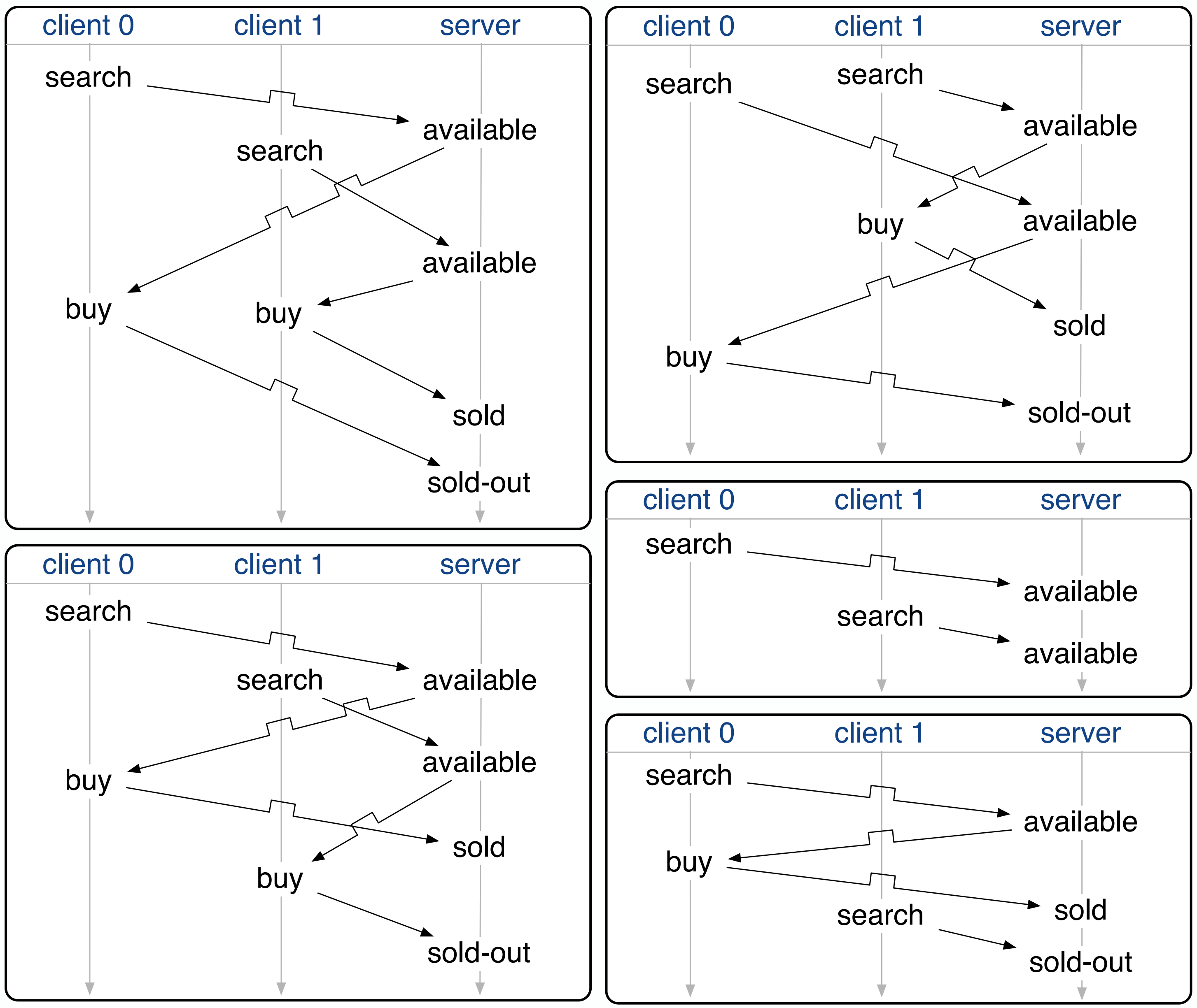
[1,0,0] client 0: search for tickets to Portugal for 23/10/11
[1,0,1] server: there is a ticket available for 505P
[2,0,1] client 0: buy ticket
[2,0,2] server: sold
[0,1,0] client 1: search for tickets to Portugal for 23/10/11
[2,1,3] server: tickets sold out

[0,1,0] client 1: search for tickets to Portugal for 23/10/11
[1,0,0] client 0: search for tickets to Portugal for 23/10/11
[0,1,1] server: there is a ticket available for 505P
[1,1,2] server: there is a ticket available for 505P
[0,2,1] client 1: buy ticket
[1,2,3] server: sold
[2,1,2] client 0: buy ticket
[2,2,4] server: tickets sold out

[1,0,0] client 0: search for tickets to Portugal for 23/10/11
[1,0,1] server: there is a ticket available for 505P
[0,1,0] client 1: search for tickets to Portugal for 23/10/11
[1,1,2] server: there is a ticket available for 505P
[1,2,2] client 1: buy ticket
[1,2,3] server: sold
[2,0,1] client 0: buy ticket
[2,2,4] server: tickets sold out

[1,0,0] client 0: search for tickets to Portugal for 23/10/11
[1,0,1] server: there is a ticket available for 505P
[0,1,0] client 1: search for tickets to Portugal for 23/10/11
[1,1,2] server: there is a ticket available for 505P

## Executions:



# Partial order is complex

## Partially ordered log :

[1,0,0] client 0: search for tickets to Portugal for 23/10/11
[0,1,0] client 1: search for tickets to Portugal for 23/10/11
[1,0,1] server: there is a ticket available for 505P
[1,1,2] server: there is a ticket available for 505P
[2,0,1] client 0: buy ticket
[2,1,3] server: sold
[1,2,2] client 1: buy ticket
[2,2,4] server: tickets sold out

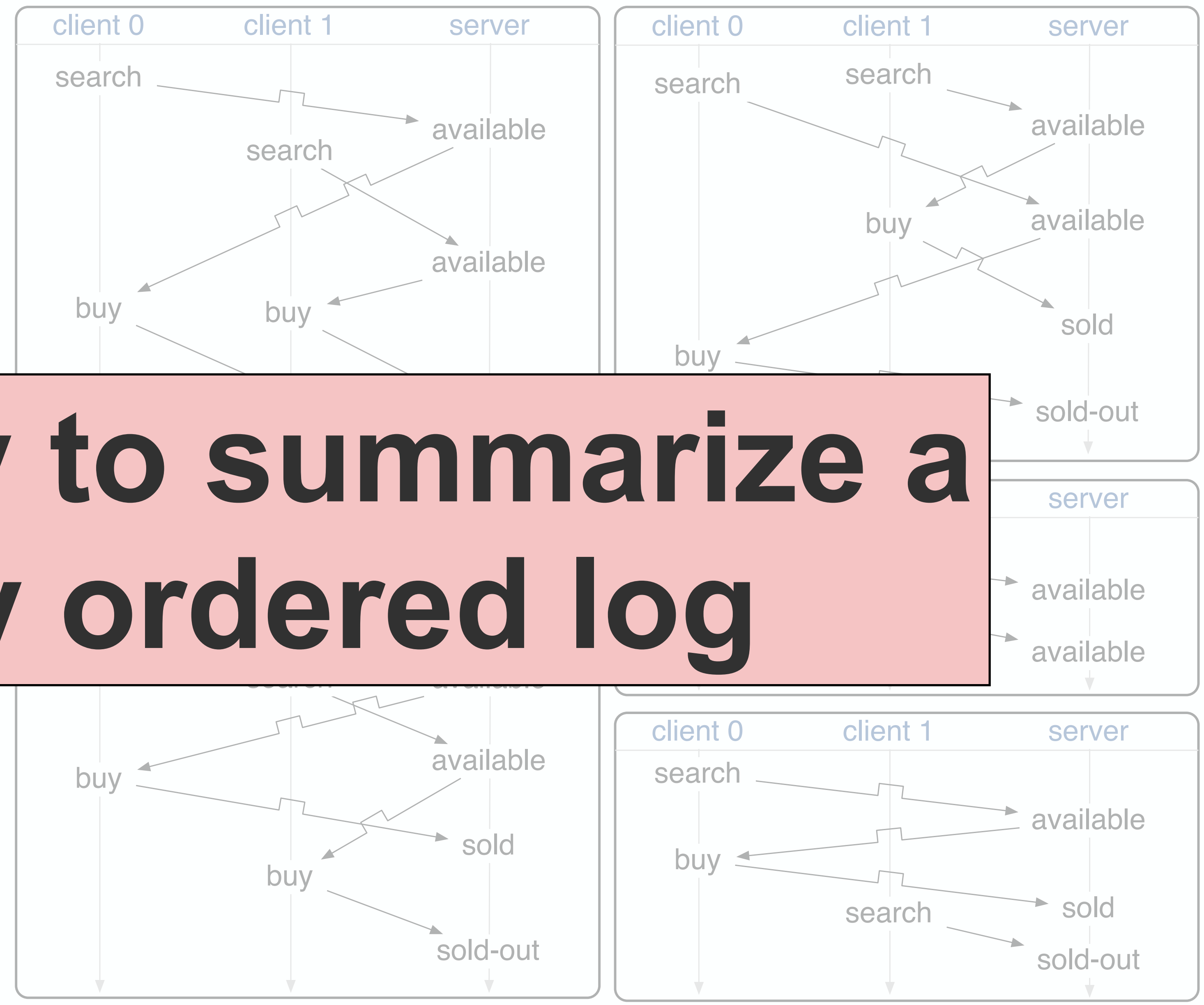
[1,0,0] client 0: search for tickets to Portugal for 23/10/11
[1,0,1] server: there is a ticket available for 505P
[2,0,1] client 0: b
[2,0,2] server: so
[0,1,0] client 1: s
[2,1,3] server: tic

[0,1,0] client 1: s
[1,0,0] client 0: s
[0,1,1] server: th
[1,1,2] server: th
[0,2,1] client 1: b
[1,2,3] server: sold
[2,1,2] client 0: buy ticket
[2,2,4] server: tickets sold out

[1,0,0] client 0: search for tickets to Portugal for 23/10/11
[1,0,1] server: there is a ticket available for 505P
[0,1,0] client 1: search for tickets to Portugal for 23/10/11
[1,1,2] server: there is a ticket available for 505P
[1,2,2] client 1: buy ticket
[1,2,3] server: sold
[2,0,1] client 0: buy ticket
[2,2,4] server: tickets sold out

[1,0,0] client 0: search for tickets to Portugal for 23/10/11
[1,0,1] server: there is a ticket available for 505P
[0,1,0] client 1: search for tickets to Portugal for 23/10/11
[1,1,2] server: there is a ticket available for 505P

## Executions:




**Need a way to summarize a partially ordered log**

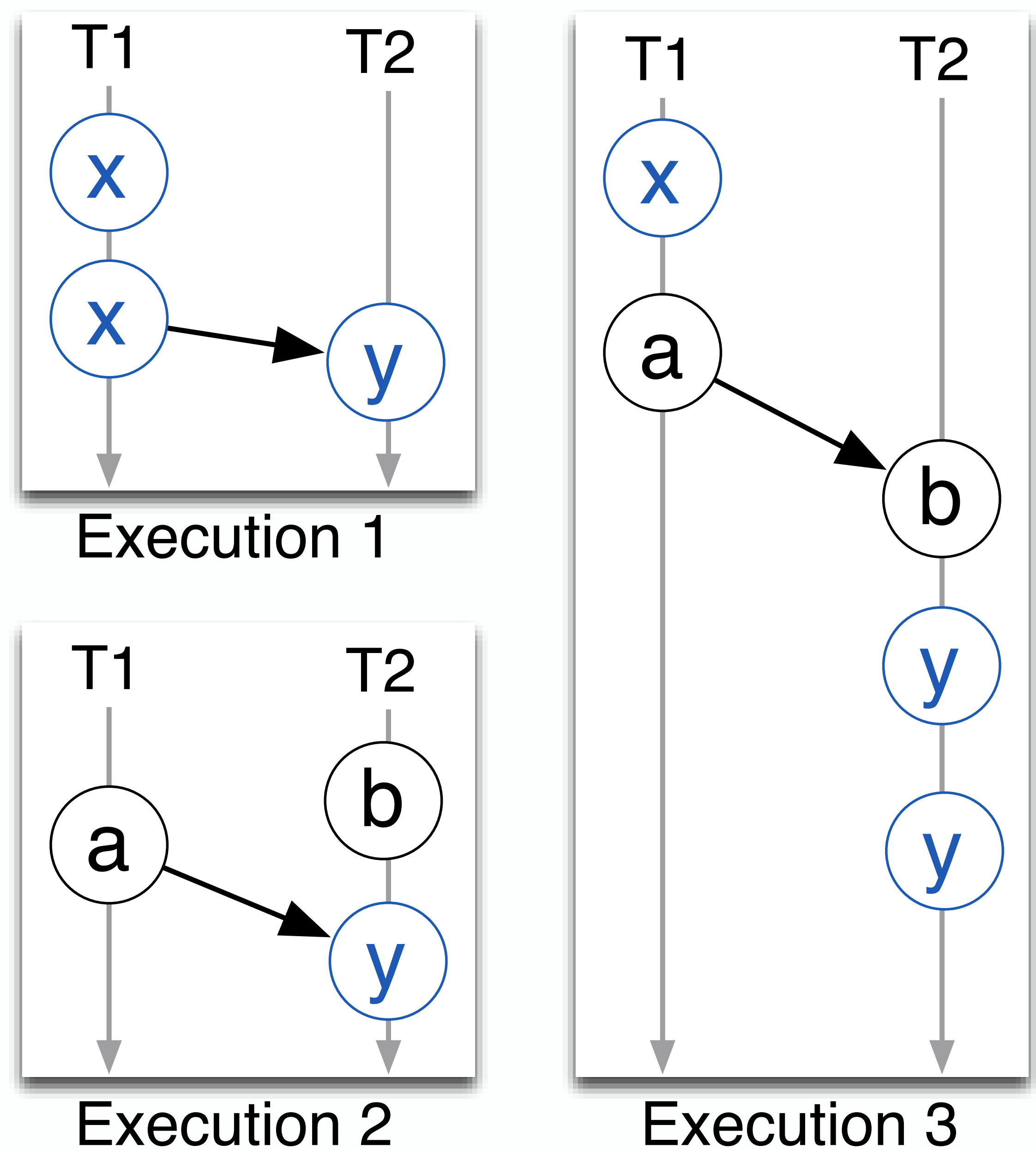
# Temporal invariants

- Mine the partially ordered log to extract temporal invariants between events

- Temporal invariants
  - True for all logged executions
    - Capture the essence of what happened
  - Simple to understand
    - Each invariants involves at most two hosts
    - Summarize the partial order

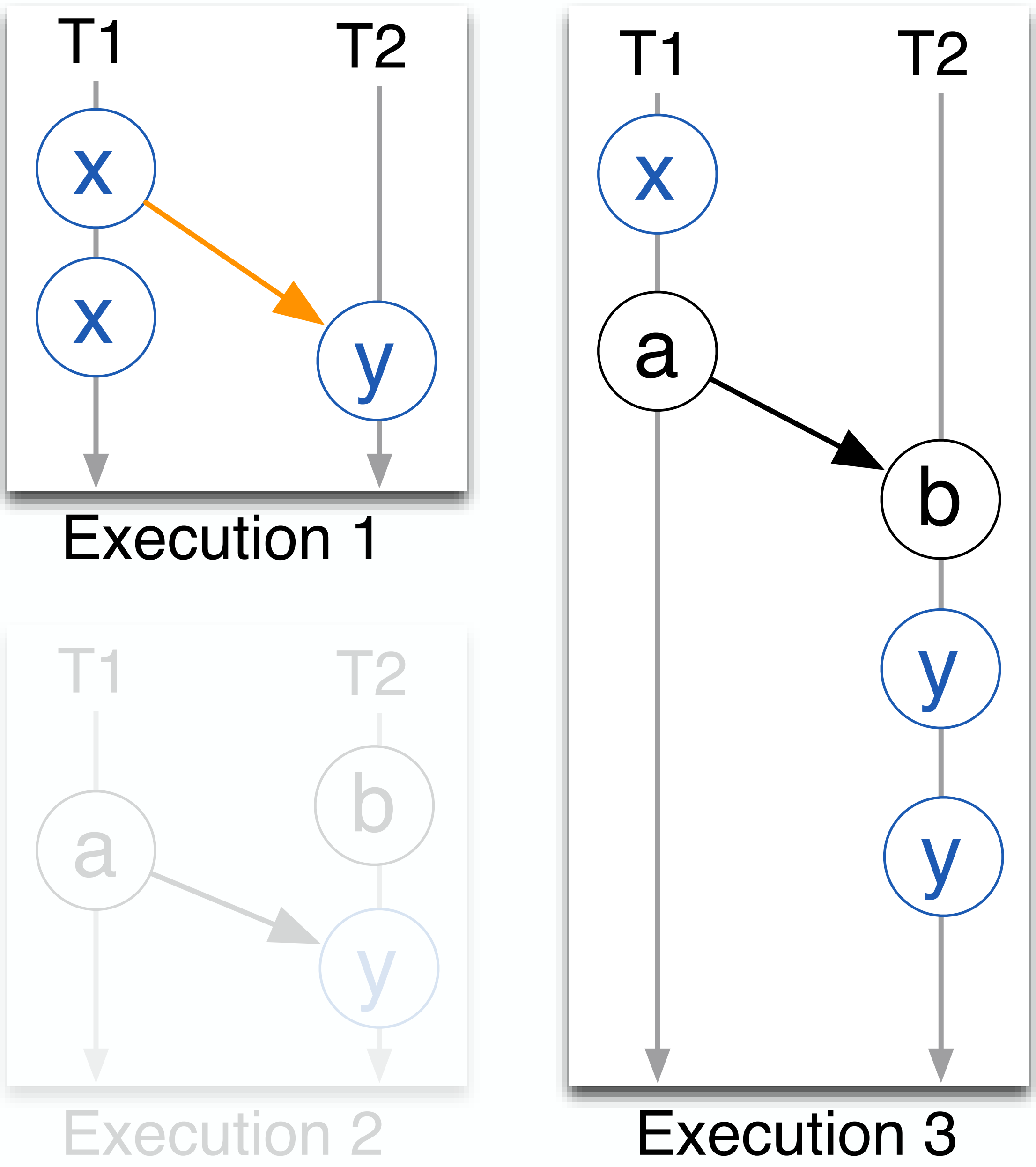
# Five temporal log invariants

| Invariant | Type |
|---|---|
| $x_1 \rightarrow y_2$ always followed by | liveness |
| $x_1 \leftarrow y_2$ always precedes | safety |
| $x_1 \not\rightarrow y_2$ never followed by | safety |
| $x_1 \parallel y_2$ always concurrent with | safety |
| $x_1 \nparallel y_2$ never concurrent with | safety |



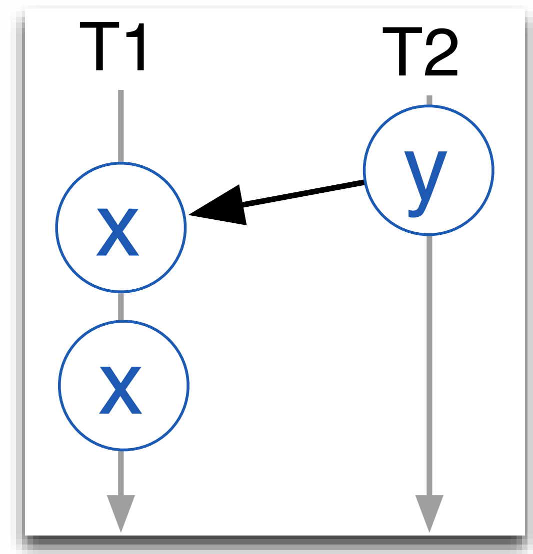
# Five temporal log invariants

| Invariant | Type |
|---|---|
| $x_1 \rightarrow y_2$ always followed by | liveness |
| $x_1 \leftarrow y_2$ always precedes | safety |
| $x_1 \not\rightarrow y_2$ never followed by | safety |
| $x_1 \parallel y_2$ always concurrent with | safety |
| $x_1 \nparallel y_2$ never concurrent with | safety |

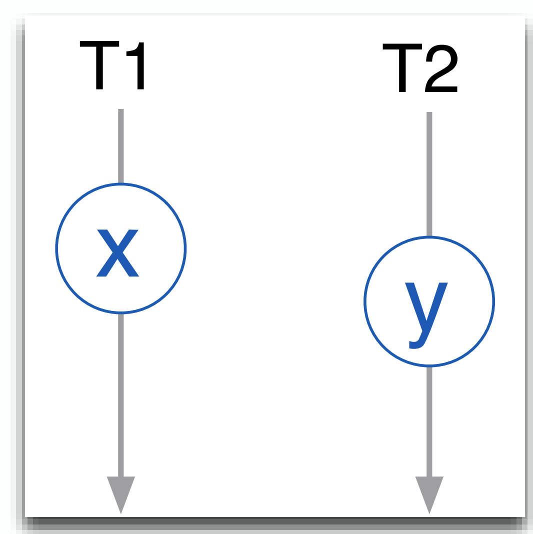


# Five temporal log invariants

| Invariant | Type |
|---|---|
| $x_1 \rightarrow y_2$ always followed by | liveness |
| $x_1 \leftarrow y_2$ always precedes | safety |
| $x_1 \not\rightarrow y_2$ never followed by | safety |
| $x_1 \parallel y_2$ always concurrent with | safety |
| $x_1 \nparallel y_2$ never concurrent with | safety |



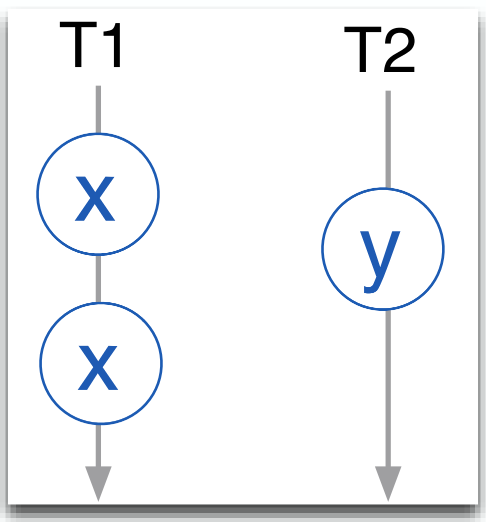

Execution 1



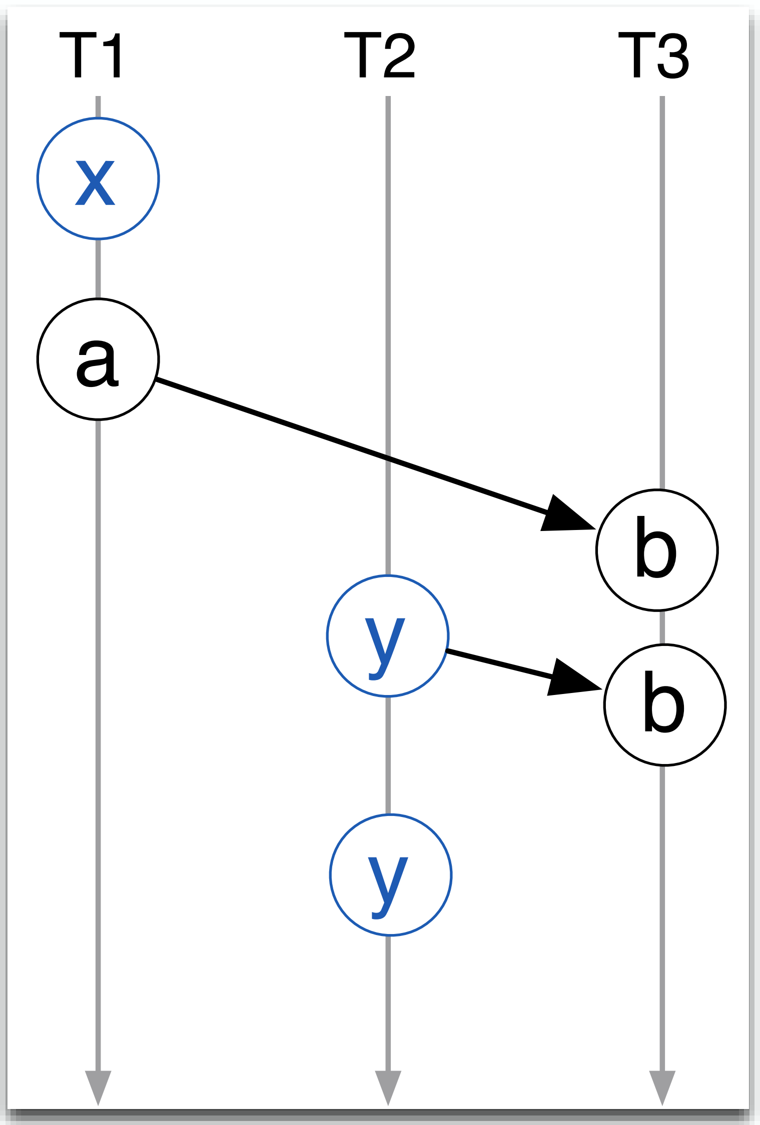

Execution 2

# Five temporal log invariants

| Invariant | Type |
|---|---|
| $x_1 \rightarrow y_2$ always followed by | liveness |
| $x_1 \leftarrow y_2$ always precedes | safety |
| $x_1 \not\rightarrow y_2$ never followed by | safety |
| $x_1 \parallel y_2$ always concurrent with | safety |
| $x_1 \nparallel y_2$ never concurrent with | safety |



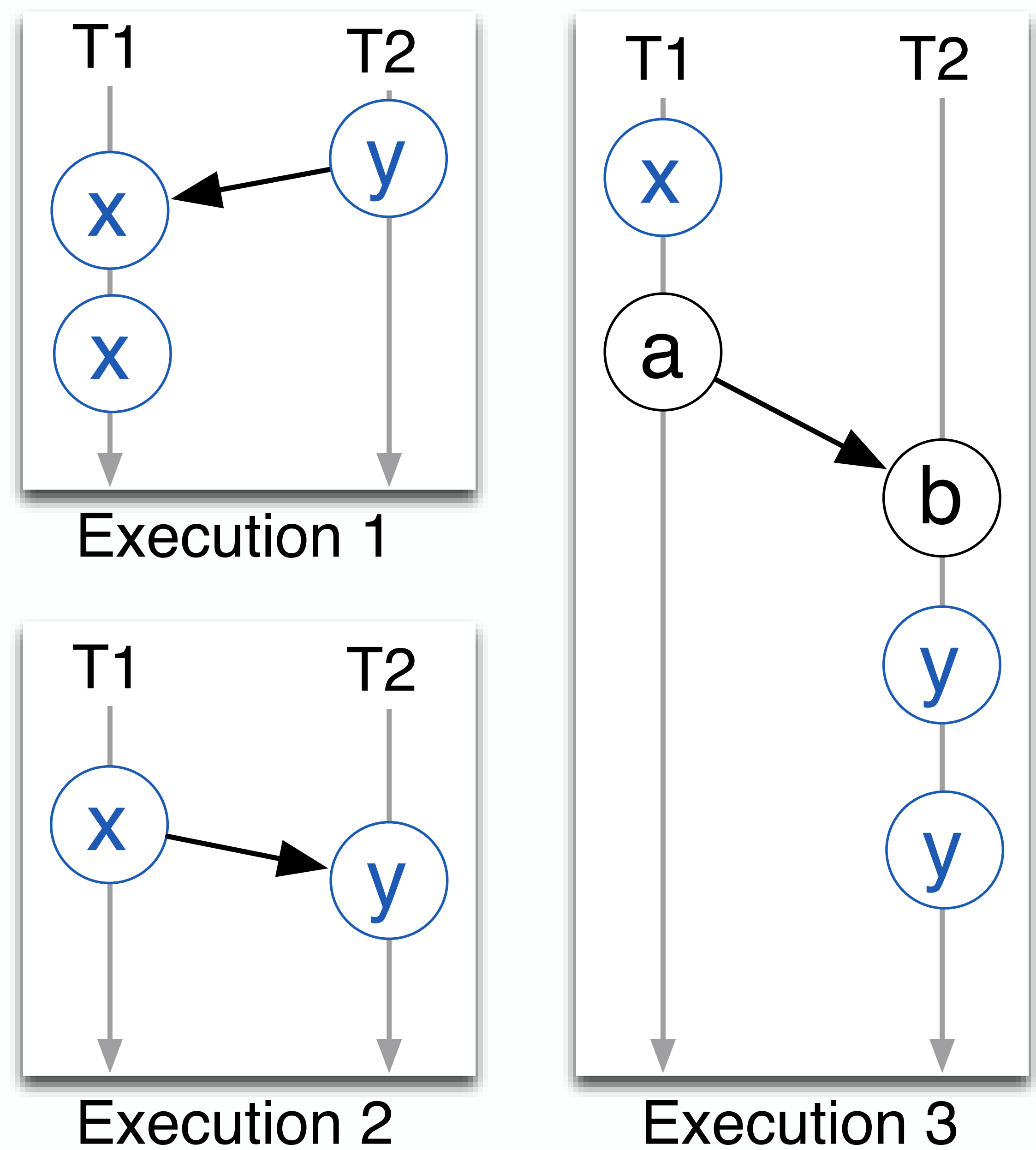
Execution 1



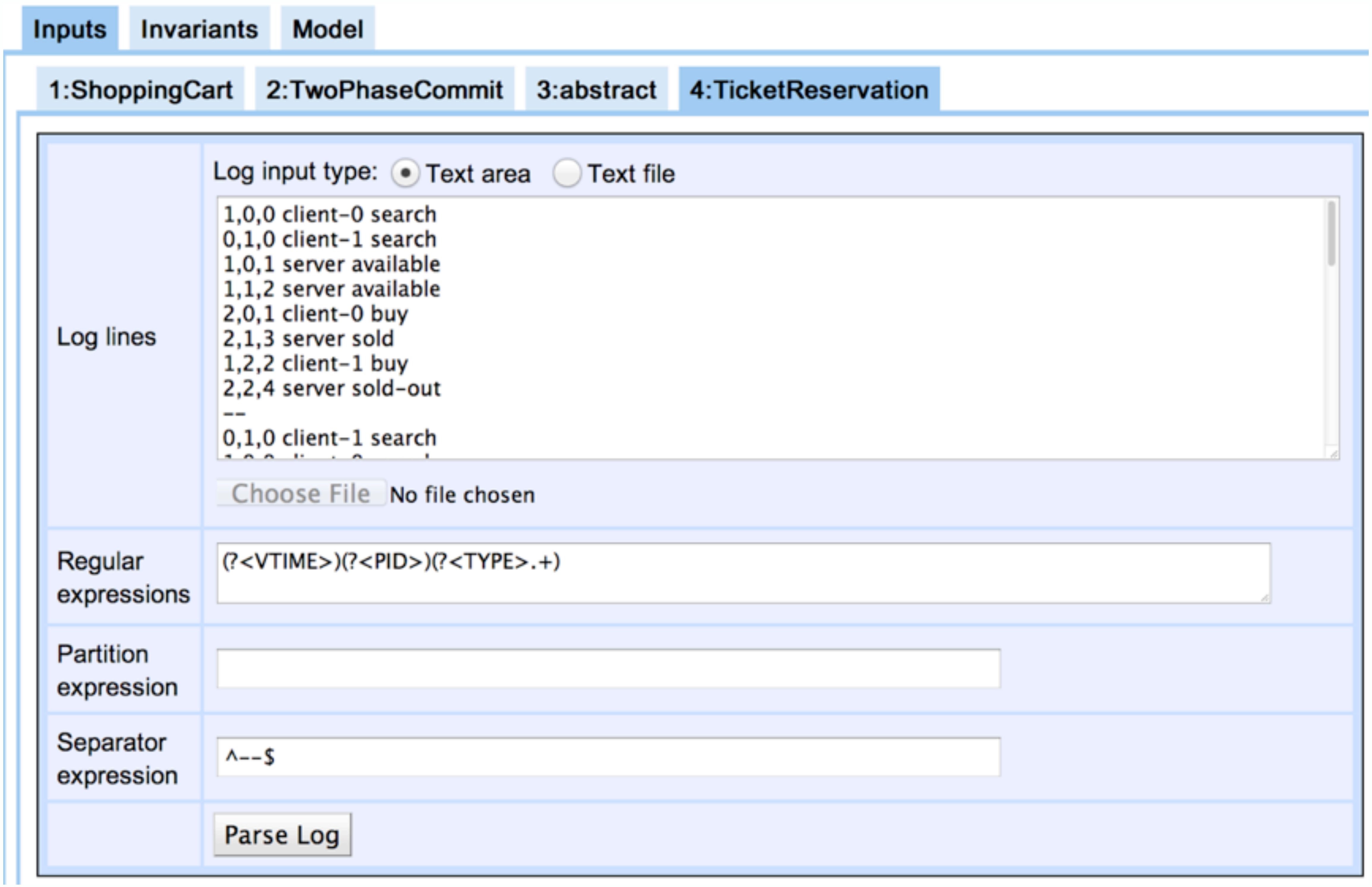
Execution 2

# Five temporal log invariants

| Invariant | Type |
| --- | --- |
| $x_1 \rightarrow y_2$ always followed by | liveness |
| $x_1 \leftarrow y_2$ always precedes | safety |
| $x_1 \nrightarrow y_2$ never followed by | safety |
| $x_1 \parallel y_2$ always concurrent with | safety |
| $x_1 \nparallel y_2$ never concurrent with | safety |



# DEMO


Inputs
Invariants
Model
1:ShoppingCart
2:TwoPhaseCommit
3:abstract
4:TicketReservation
Log lines
Log input type: Text area Text file
1,0,0 client-0 search
0,1,0 client-1 search
1,0,1 server available
1,1,2 server available
2,0,1 client-0 buy
2,1,3 server sold
1,2,2 client-1 buy
2,2,4 server sold-out
--
0,1,0 client-1 search
Choose File No file chosen
Regular expressions
(?<VTIME>)(?<PID>)(?<TYPE>.+)
Partition expression
Separator expression
^--$
Parse Log

# Mined invariants

Inputs | **Invariants** | Model

| AlwaysFollowedBy | NeverFollowedBy | AlwaysPrecedes | AlwaysConcurrentWith |
|---|---|---|---|
| INITIAL, search_client-0 | search_client-0, search_client-0 | search_client-0, buy_client-0 | search_client-1, search_client-0 |
| search_client-0, available_server | available_server, search_client-0 | search_client-0, sold_server | buy_client-1, buy_client-0 |
| INITIAL, search_client-1 | buy_client-0, search_client-0 | search_client-0, sold-out_server | |
| INITIAL, available_server | sold_server, search_client-0 | search_client-1, buy_client-1 | |
| buy_client-0, sold-out_server | buy_client-1, search_client-0 | search_client-1, sold-out_server | |
| sold_server, sold-out_server | sold-out_server, search_client-0 | available_server, buy_client-0 | |
| buy_client-1, sold-out_server | search_client-1, search_client-1 | available_server, sold_server | |
| | available_server, search_client-1 | available_server, buy_client-1 | |

# Mined invariants

| $\rightarrow$ | $\leftarrow$ | $\not\rightarrow$ | $\parallel$ | $\nparallel$ |
|---|---|---|---|---|
| always followed by | always precedes | never followed by | always concurrent with | never concurrent with |

| |
|---|
| $\text{available}_s \leftarrow \text{buy}_{c_0}$ |
| $\text{available}_s \leftarrow \text{buy}_{c_1}$ |
| $\text{sold-out}_s \not\rightarrow \text{buy}_{c_0}$ |
| $\text{sold-out}_s \not\rightarrow \text{buy}_{c_1}$ |

| |
|---|
| $\text{sold-out}_s \not\rightarrow \text{sold}_s$ |
| $\text{sold}_s \leftarrow \text{sold-out}_s$ |
| $\text{buy}_{c_0} \parallel \text{buy}_{c_1}$ |
| $\text{search}_{c_0} \parallel \text{search}_{c_1}$ |

| |
|---|
| $\text{buy}_{c_0} \rightarrow \text{sold-out}_s$ |
| $\text{buy}_{c_1} \rightarrow \text{sold-out}_s$ |
| $\text{buy}_{c_0} \leftarrow \text{sold-out}_s$ |

# Mined invariants

| $\rightarrow$ | $\leftarrow$ | $\not\rightarrow$ | $\parallel$ | $\nparallel$ |
|---|---|---|---|---|
| always followed by | always precedes | never followed by | always concurrent with | never concurrent with |

| |
|---|
| $\text{available}_s \leftarrow \text{buy}_{c_0}$ |
| $\text{available}_s \leftarrow \text{buy}_{c_1}$ |
| $\text{sold-out}_s \not\rightarrow \text{buy}_{c_0}$ |
| $\text{sold-out}_s \not\rightarrow \text{buy}_{c_1}$ |

| |
|---|
| $\text{sold-out}_s \not\rightarrow \text{sold}_s$ |
| $\text{sold}_s \leftarrow \text{sold-out}_s$ |
| $\text{buy}_{c_0} \parallel \text{buy}_{c_1}$ |
| $\text{search}_{c_0} \parallel \text{search}_{c_1}$ |

| |
|---|
| $\text{buy}_{c_0} \rightarrow \text{sold-out}_s$ |
| $\text{buy}_{c_1} \rightarrow \text{sold-out}_s$ |
| $\text{buy}_{c_0} \leftarrow \text{sold-out}_s$ |

Server event

Client events

# Mined invariants

| always followed by | always precedes | never followed by | always concurrent with | never concurrent with |
|---|---|---|---|---|
| $\rightarrow$ | $\leftarrow$ | $\not\rightarrow$ | $\parallel$ | $\nparallel$ |

| |
|---|
| $\text{available}_s \leftarrow \text{buy}_{c_0}$ |
| $\text{available}_s \leftarrow \text{buy}_{c_1}$ |
| $\text{sold-out}_s \not\rightarrow \text{buy}_{c_0}$ |
| $\text{sold-out}_s \not\rightarrow \text{buy}_{c_1}$ |

Temporal orderings between server and client events

| |
|---|
| $\text{sold-out}_s \not\rightarrow \text{sold}_s$ |
| $\text{sold}_s \leftarrow \text{sold-out}_s$ |
| $\text{buy}_{c_0} \parallel \text{buy}_{c_1}$ |
| $\text{search}_{c_0} \parallel \text{search}_{c_1}$ |

| |
|---|
| $\text{buy}_{c_0} \rightarrow \text{sold-out}_s$ |
| $\text{buy}_{c_1} \rightarrow \text{sold-out}_s$ |
| $\text{buy}_{c_0} \leftarrow \text{sold-out}_s$ |

# Mined invariants

| $\rightarrow$ always followed by | $\leftarrow$ always precedes | $\not\rightarrow$ never followed by | $\parallel$ always concurrent with | $\nparallel$ never concurrent with |
|---|---|---|---|---|

Server-side correctness invariants

| |
|---|
| $\text{available}_s \leftarrow \text{buy}_{c_0}$ |
| $\text{available}_s \leftarrow \text{buy}_{c_1}$ |
| $\text{sold-out}_s \not\rightarrow \text{buy}_{c_0}$ |
| $\text{sold-out}_s \not\rightarrow \text{buy}_{c_1}$ |

Temporal orderings between server and client events

| |
|---|
| $\boxed{\text{sold-out}_s \not\rightarrow \text{sold}_s}$ |
| $\boxed{\text{sold}_s \leftarrow \text{sold-out}_s}$ |
| $\text{buy}_{c_0} \parallel \text{buy}_{c_1}$ |
| $\text{search}_{c_0} \parallel \text{search}_{c_1}$ |

| |
|---|
| $\text{buy}_{c_0} \rightarrow \text{sold-out}_s$ |
| $\text{buy}_{c_1} \rightarrow \text{sold-out}_s$ |
| $\text{buy}_{c_0} \leftarrow \text{sold-out}_s$ |

# Mined invariants

| $\rightarrow$ | $\leftarrow$ | $\not\rightarrow$ | $\parallel$ | $\nparallel$ |
|---|---|---|---|---|
| always followed by | always precedes | never followed by | always concurrent with | never concurrent with |

Server-side correctness invariants

| |
|---|
| $\text{sold-out}_s \not\rightarrow \text{sold}_s$ |
| $\text{sold}_s \leftarrow \text{sold-out}_s$ |
| $\text{buy}_{c_0} \parallel \text{buy}_{c_1}$ |
| $\text{search}_{c_0} \parallel \text{search}_{c_1}$ |

Concurrency between clients

| |
|---|
| $\text{available}_s \leftarrow \text{buy}_{c_0}$ |
| $\text{available}_s \leftarrow \text{buy}_{c_1}$ |
| $\text{sold-out}_s \not\rightarrow \text{buy}_{c_0}$ |
| $\text{sold-out}_s \not\rightarrow \text{buy}_{c_1}$ |

Temporal orderings between server and client events

| |
|---|
| $\text{buy}_{c_0} \rightarrow \text{sold-out}_s$ |
| $\text{buy}_{c_1} \rightarrow \text{sold-out}_s$ |
| $\text{buy}_{c_0} \leftarrow \text{sold-out}_s$ |

# Mined invariants

| | | | | |
|---|---|---|---|---|
| $\rightarrow$ | $\leftarrow$ | $\not\rightarrow$ | $\parallel$ | $\nparallel$ |
| always followed by | always precedes | never followed by | always concurrent with | never concurrent with |

Server-side
correctness invariants

| |
|---|
| $\text{available}_s \leftarrow \text{buy}_{c_0}$ |
| $\text{available}_s \leftarrow \text{buy}_{c_1}$ |
| $\text{sold-out}_s \not\rightarrow \text{buy}_{c_0}$ |
| $\text{sold-out}_s \not\rightarrow \text{buy}_{c_1}$ |

Temporal orderings between server and client events

| |
|---|
| $\text{sold-out}_s \not\rightarrow \text{sold}_s$ |
| $\text{sold}_s \leftarrow \text{sold-out}_s$ |
| $\text{buy}_{c_0} \parallel \text{buy}_{c_1}$ |
| $\text{search}_{c_0} \parallel \text{search}_{c_1}$ |

Concurrency between clients

| |
|---|
| $\text{buy}_{c_0} \rightarrow \text{sold-out}_s$ |
| $\text{buy}_{c_1} \rightarrow \text{sold-out}_s$ |
| $\text{buy}_{c_0} \leftarrow \text{sold-out}_s$ |

Over-fit invariants

# Outline

- Motivation
- Why a total order is not enough
- Mining temporal invariants from concurrent executions
- Tool demo
- Two algorithms to mine temporal invariants
- Algorithms' scalability evaluation

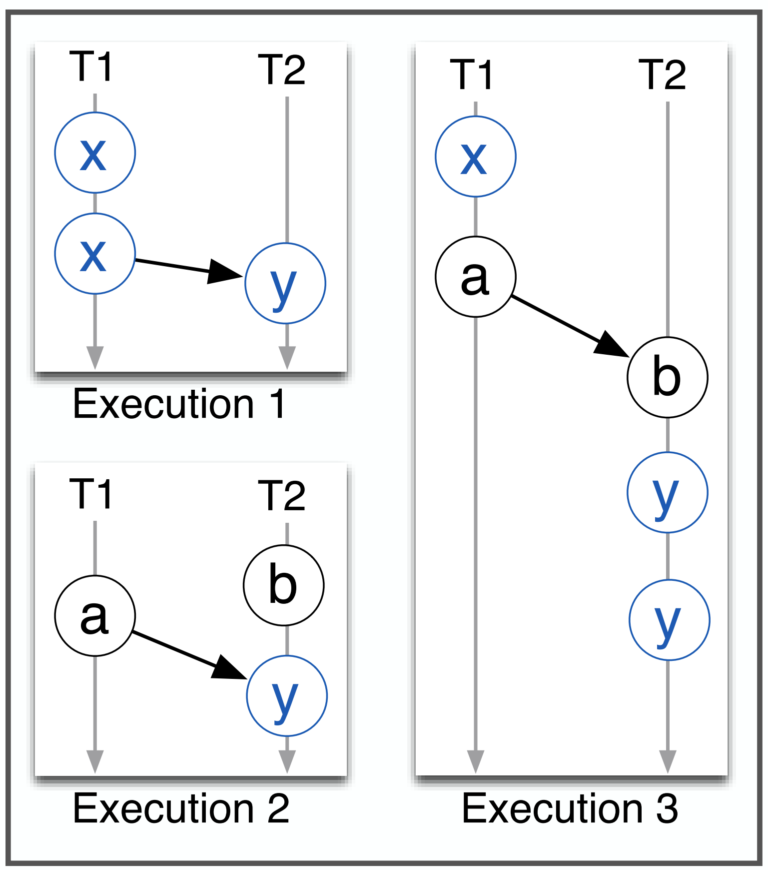
# Algorithms to mine invariants

1. An algorithm based on the transitive closure
2. A co-occurrence counting algorithm (v1)
3. A modified co-occurrence counting algorithm (v2) that omits “never concurrent with”

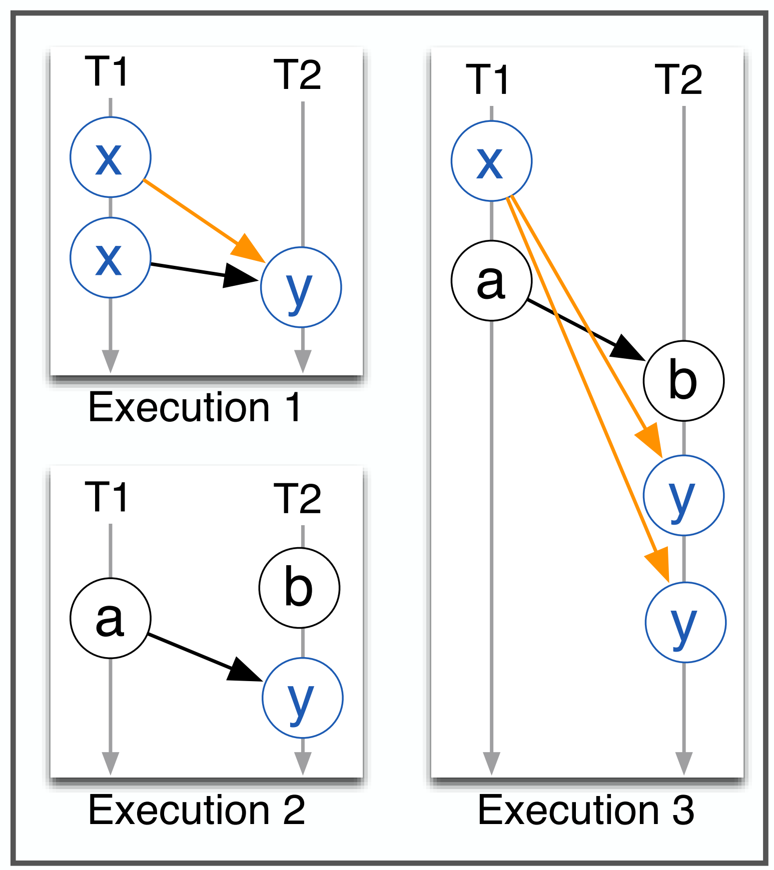
More details in the paper

# Transitive closure mining

- Compute the transitive closure of all execution DAGs
- Use the transitive closure to compute invariants




Log




Transitive Closure

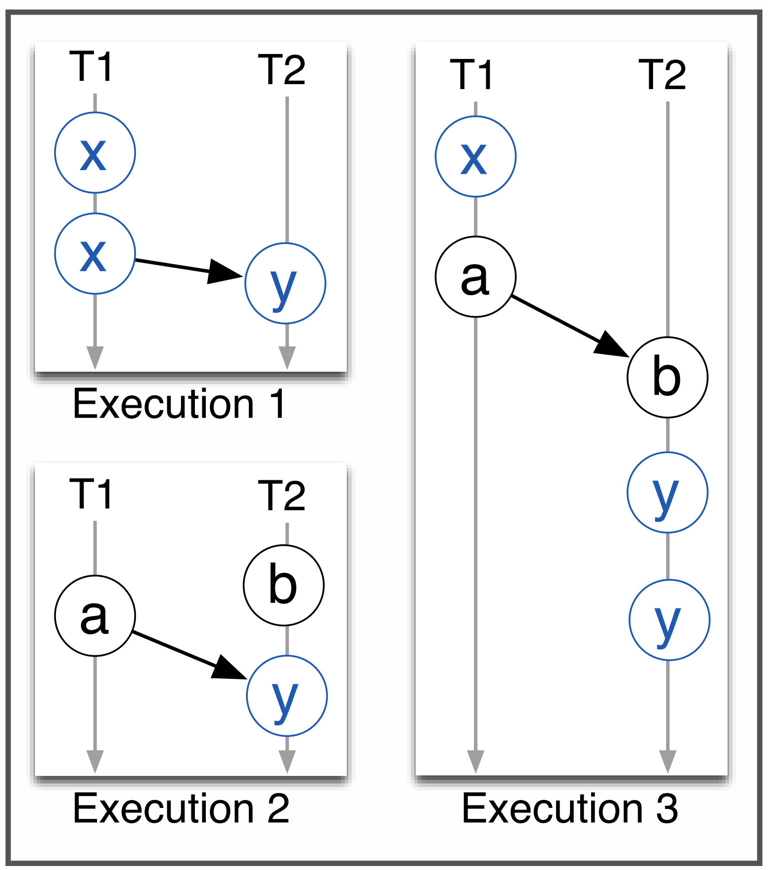
$$x_1 \rightarrow y_2$$

always followed by

Invariants

# Co-occurrence counting mining

- Count the number of times events co-occur
- Use counts to compute invariants




Log

| event | total count |
|---|---|
| x1 | 3 |
| y2 | 4 |
| ... | ... |

| event pair | # co-occurrences |
|---|---|
| x1,y2 | 3 |
| ... | ... |

Counts

$$x_1 \rightarrow y_2$$

always followed by

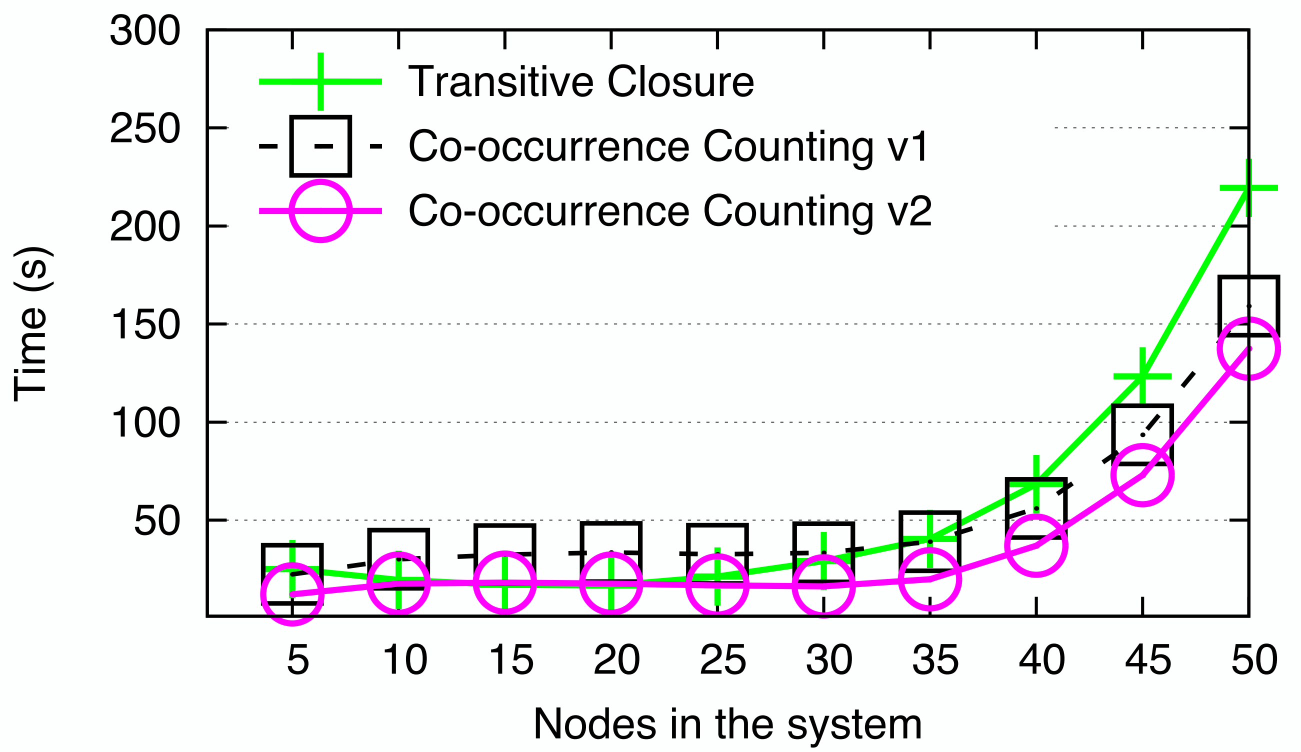
Invariants

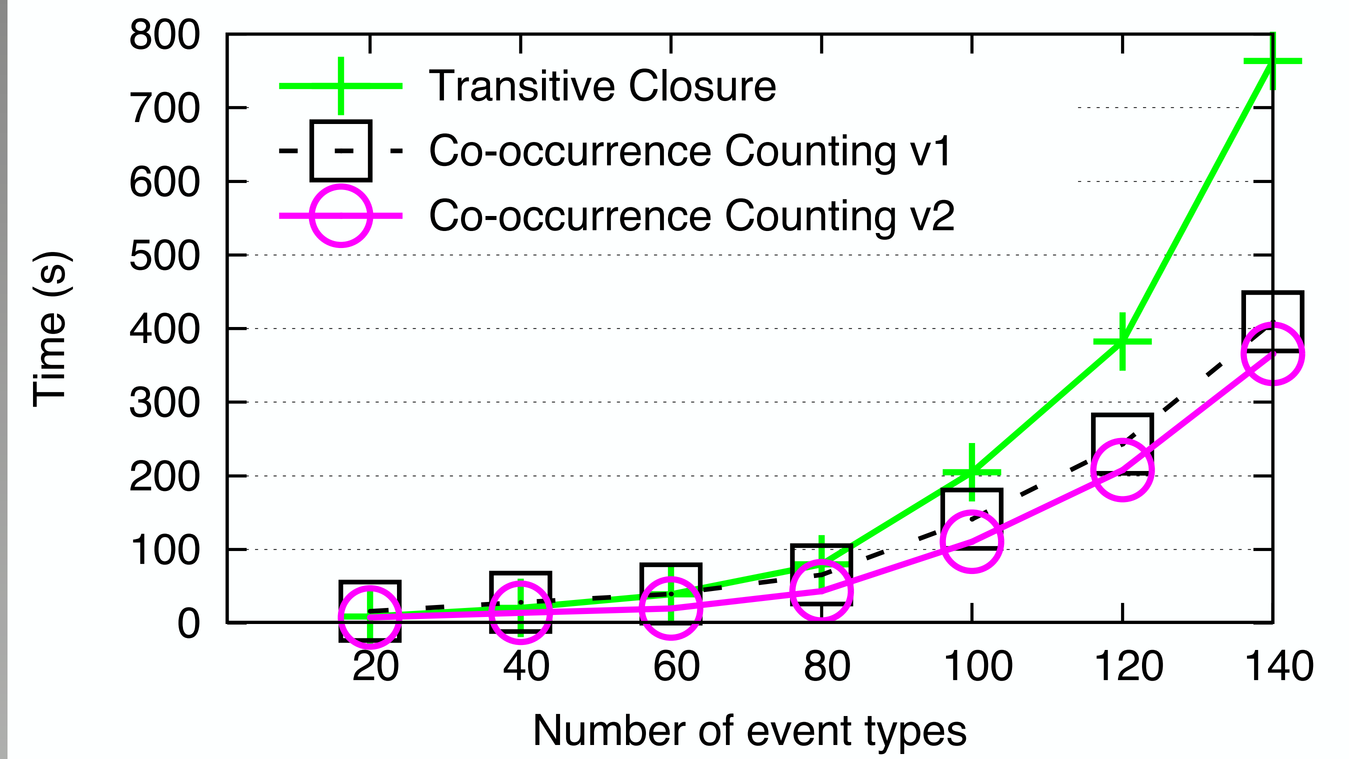
# Evaluation methodology

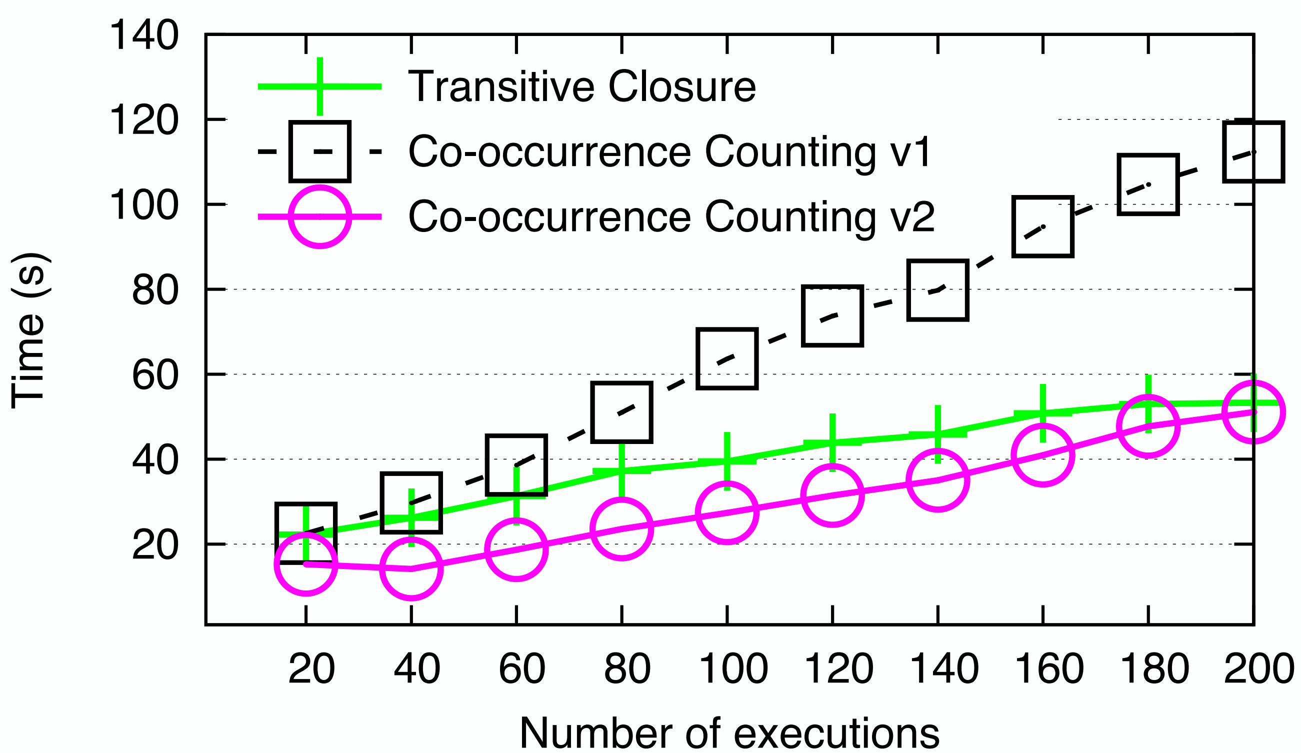
- A discrete time simulator of a distributed system with H hosts that use vector clocks to maintain a partial order
- Each host generates a total of E events
- Each event is one of T types
- Hosts communicate with probability 0.3
- Invariants are mined from the resulting log

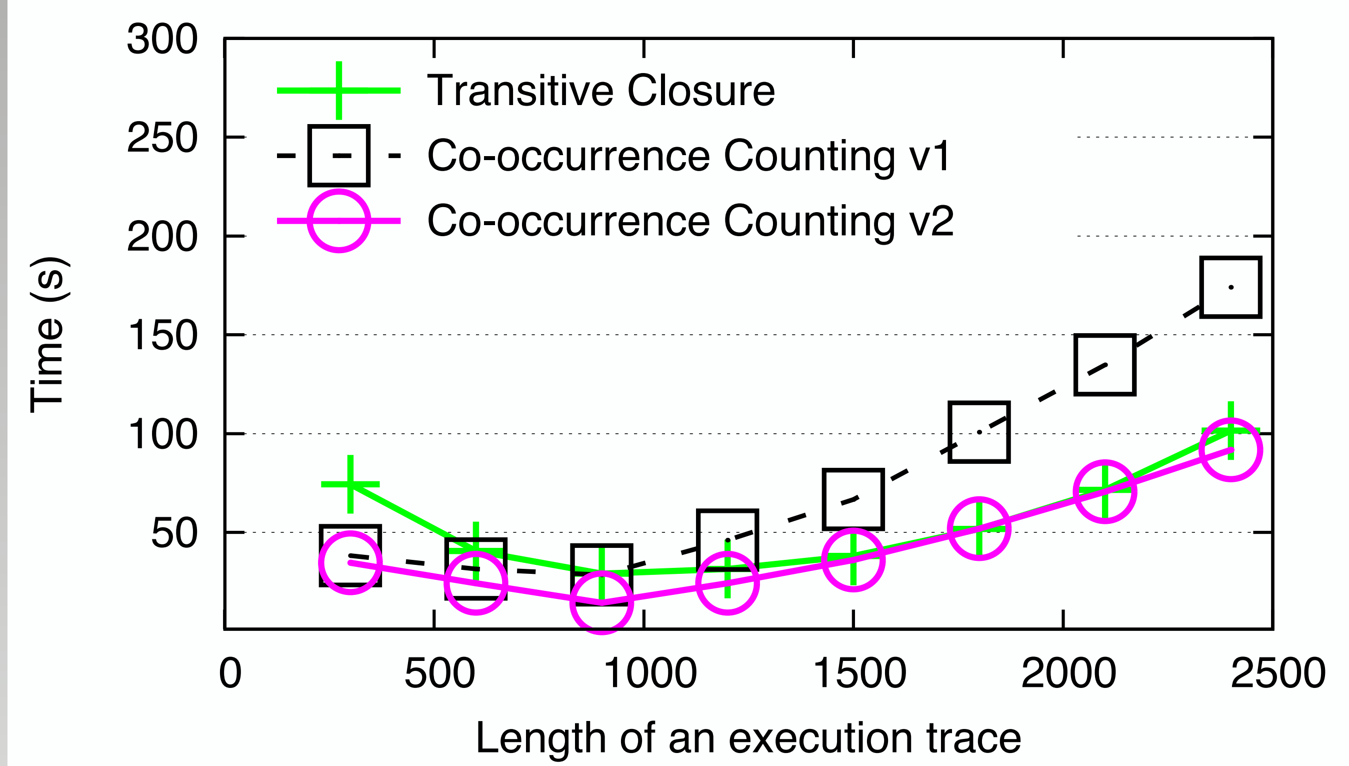
Vary each variable to evaluate algorithm scalability

# Scalability results



# Limitations and future work

- Logging the partial order explicitly has a performance penalty: extra network traffic/computation/state
  - Has been previously studied

Charron-Bost IPL 1991

Khotimsky and Zhuklinets ICATM 1999

- Invariants are a summary and do not provide a complete view

Dwyer et al. ICSE 1999

- Visualization of distributed traces

Edwards et al. IPDPS 1994

# Conclusion

- Studying logs of concurrent systems is becoming increasingly important
- Temporal invariants can help explain a complex concurrent system log
- Presented algorithms to mine five types of temporal invariants

Try it!

http://synoptic.googlecode.com