Fast Detection of Stable and Count Predicates in Parallel Computations

Himanshu Chauhan and Vijay K. Garg
University of Texas at Austin

Context: Verifying Parallel or Distributed Programs

- Correct parallel programs
    > not only difficult to implement
    > but also difficult to debug/
verify

Verifying Programs: Techniques

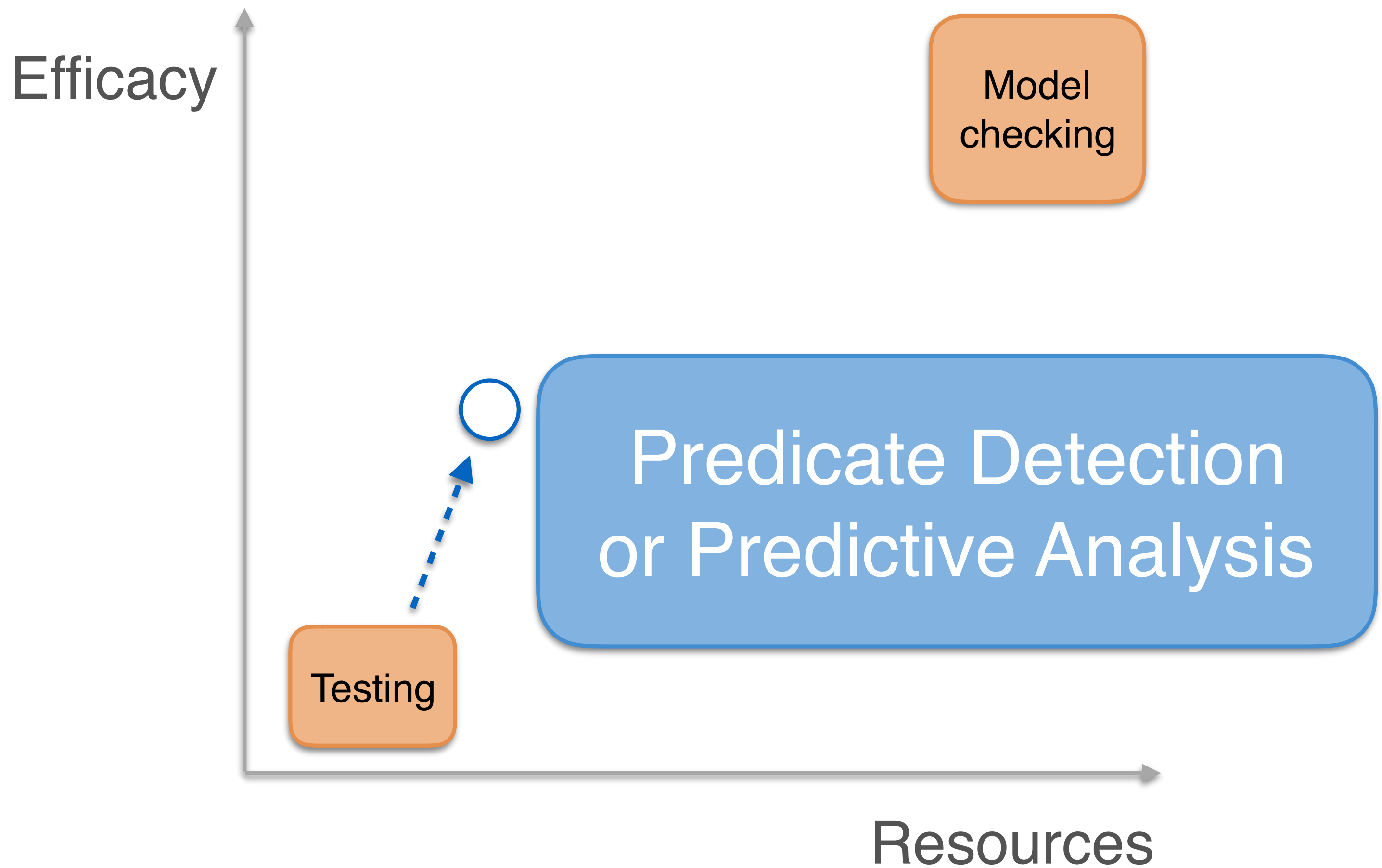
1. Testing

   run implemented program once and observe output

2. Model Checking

   check all possible states of the state machine model of the program

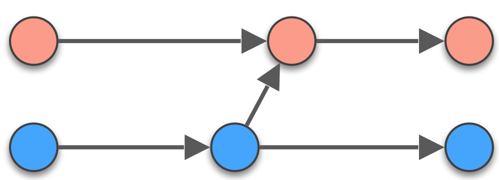
Verifying Parallel Programs



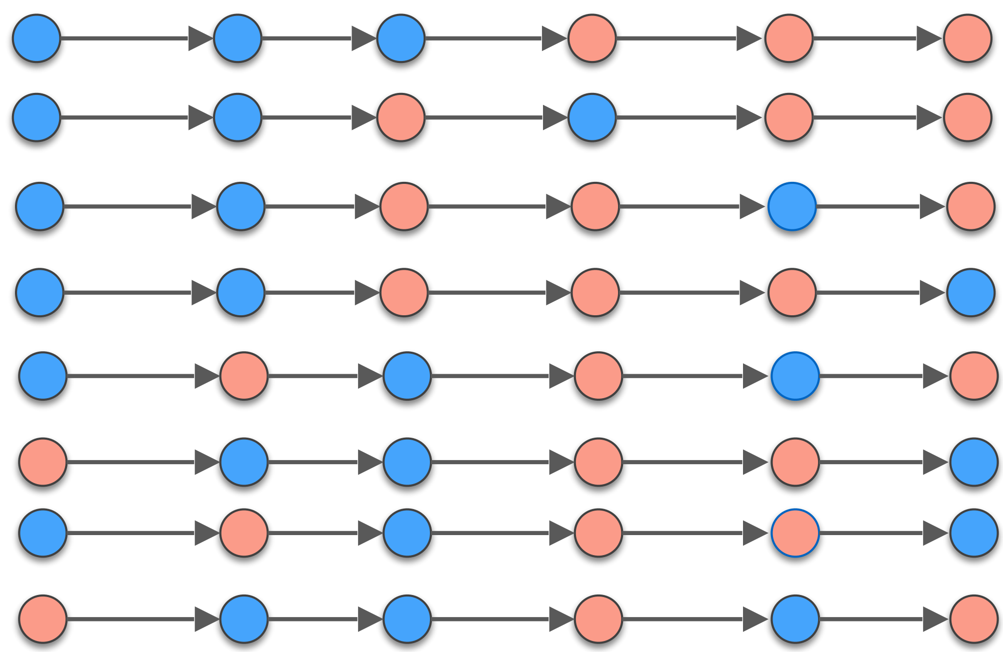
Predicate Detection

- Run program once
  1. model execution as partial order
  2. verify model for correctness

One partial order

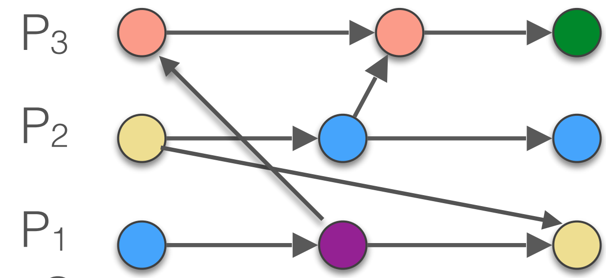


Many total orders



Analyzing Computations

Example Questions:



Is it possible that two red messages can be delivered before the first yellow event?

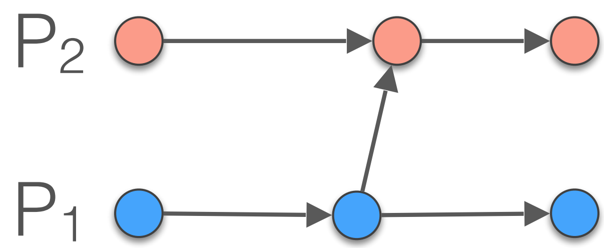
Is it possible that process P1 (master) finishes executing three events before second event on P3 happened?

Possible to map and analyze useful applications:

- Paxos implementation [distributed]
- Concurrent modification/data-race/critical section violation [shared memory]

Predicate Detection: Background

Computation: Trace of a Parallel Program



E: set of events

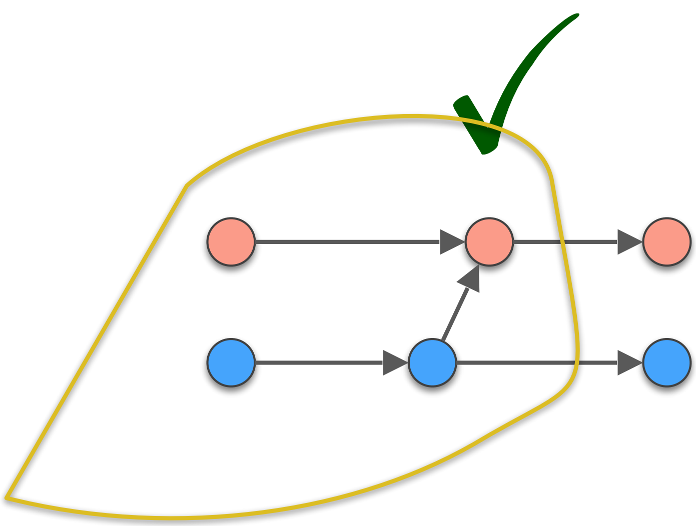
happened-before relation

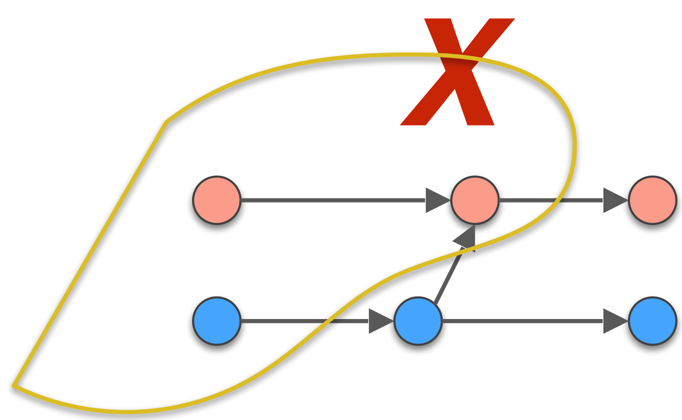
process order + causal dependency

Consistent Cut

Snapshot of computation that is consistent with the happened-before order.

If a cut *G* includes event *e* then it must include
every event that happened before *e*.





Verifying Computation for Correctness

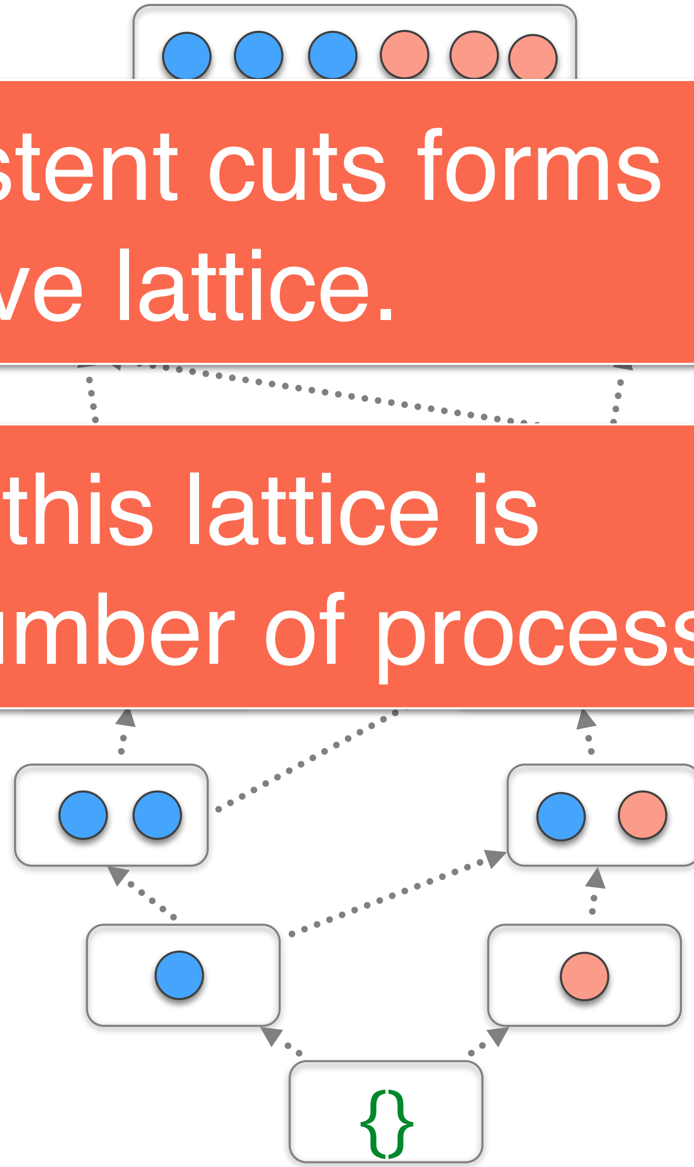
- Check all consistent cuts of the computation against a predicate B

- NP-complete [Chase and Garg 98]

Set of all Consistent Cuts

The set of consistent cuts forms a distributive lattice.

The size of this lattice is exponential in *n* (number of processes).



Enumerating all consistent cuts satisfying B

- Brute force 1:

  for all subsets G of E do

  if consistent(G) and B(G): enumerate G

Generates all cuts

- BFS: [Cooper Marzullo 92]

```
current: list of the global states initially contains initial state;
repeat
    for all G in current: if B(G) then enumerate G
    last := current;
    current = global states reached from last in one step;
until (current is empty)
```

Other Algorithms: DFS, Lex, QuickLex..

Generate all consistent cuts

Enumerating all consistent cuts satisfying B

- Need to enumerate those and only those consistent cuts that satisfy B
- The time to compute a consistent cut should be polynomial in the number of events
- NP-completeness for general B implies we need to exploit the structure of the predicate

Regular Predicates

- S_B : set of consistent cuts satisfying B

- B is regular if S_B is a sublattice of L.
  (e.g. all processes are red and all channels are empty)
- slice: a computation that generates exactly S_B [Garg and Mittal 01]

- Enumerate all consistent cuts of slice

What if B is not regular?

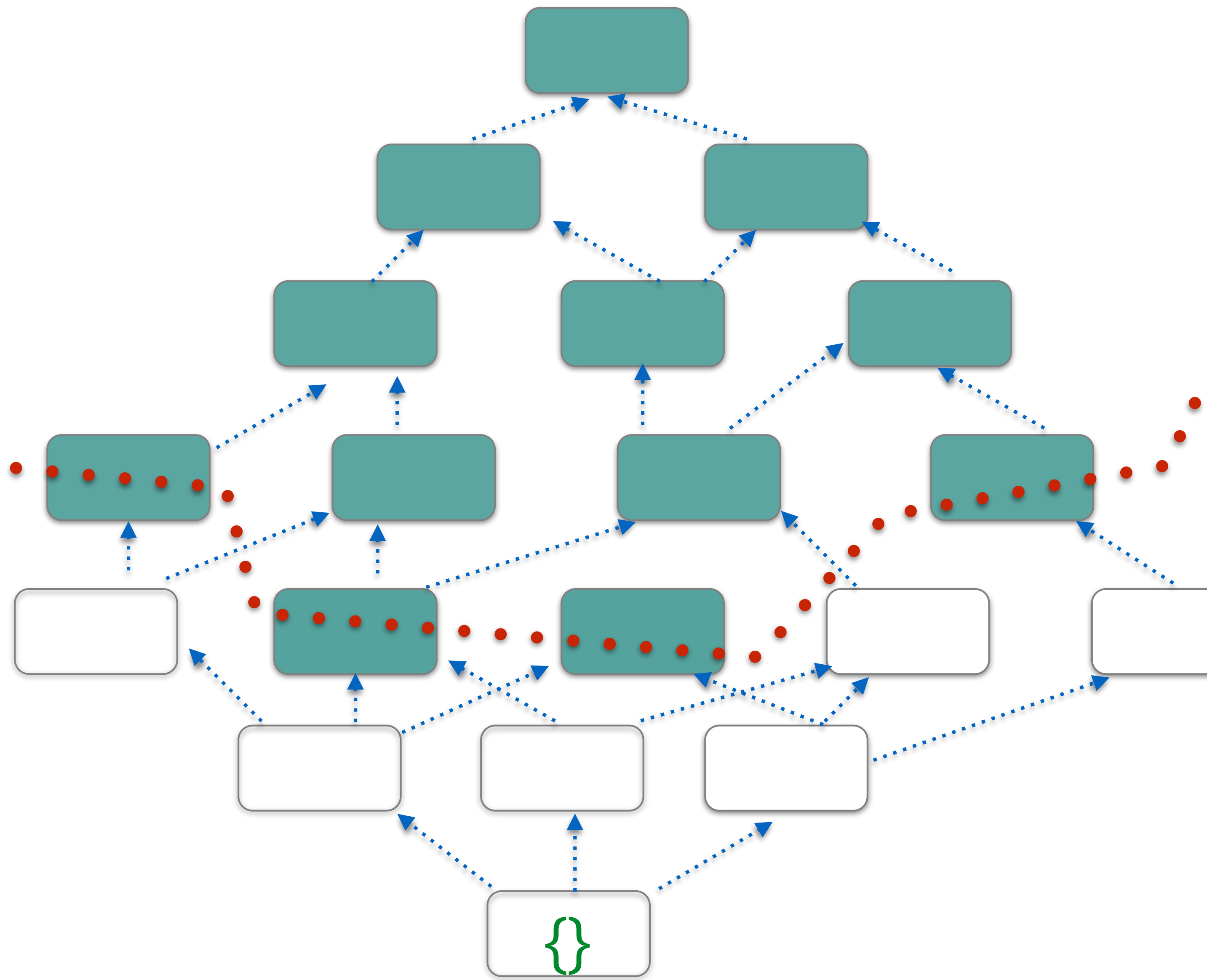
Enumerating Stable and Count Predicate Detectes

Stable Predicate

Predicate that once becomes true in the computation stays true.
[Chandy and Lamport 85]

Examples: "Every process is in round > k", "at least *k* events have been executed", "Process Pi has sent k messages".

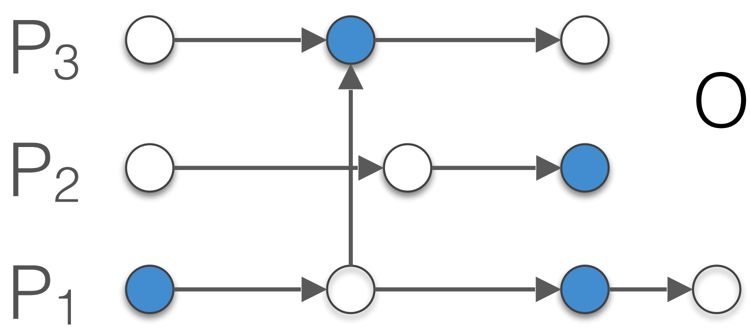
Consistent Cuts satisfying Stable Predicate



Count Predicates

Predicate that take the form:
"exactly *k* *c*-colored events have been executed.

Examples:
"exactly 3 blue events have been executed"

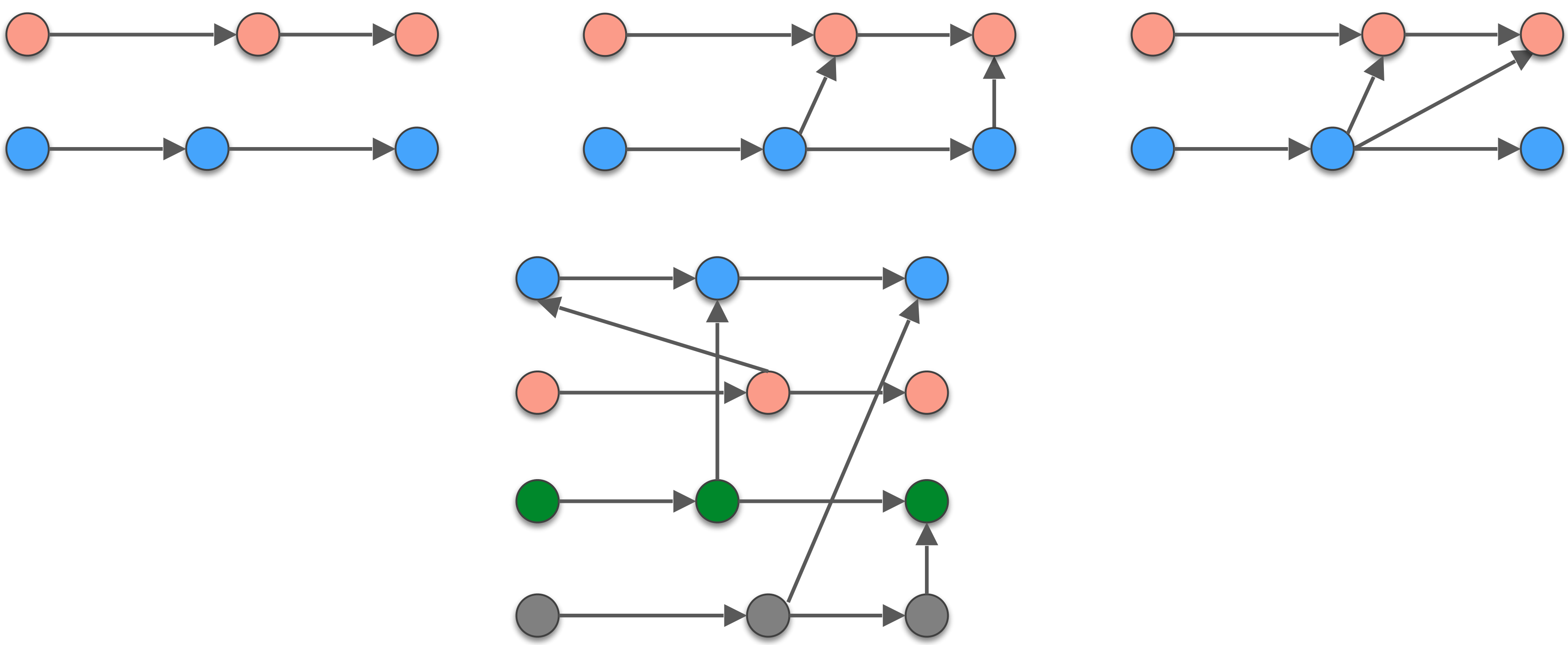


Only 8 such cuts

Total # of consistent cuts = 64

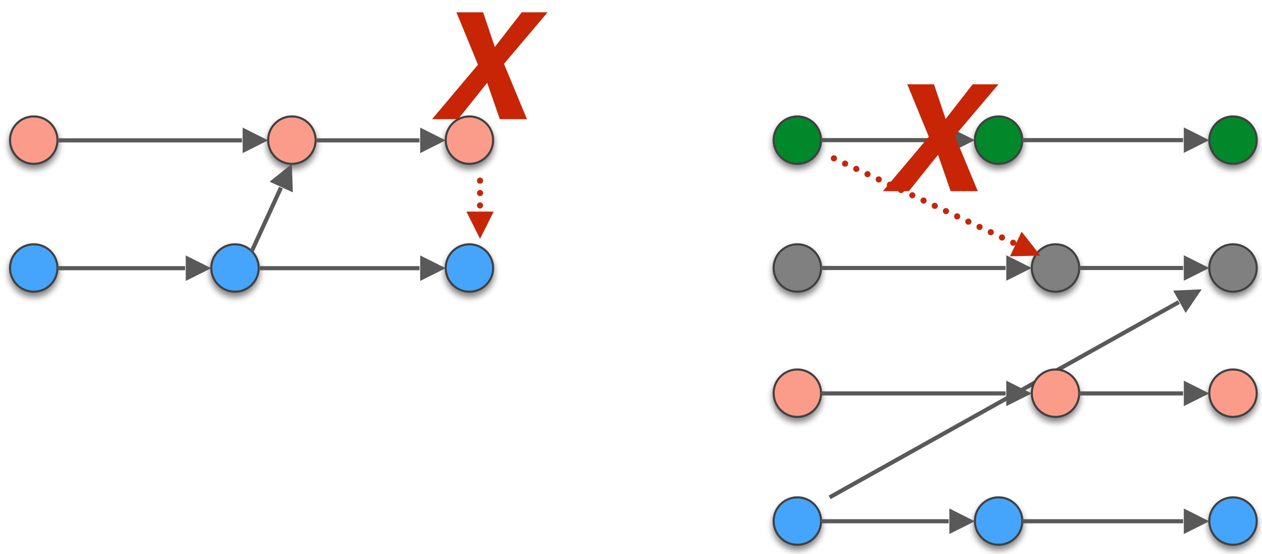
Uniflow Chain Partition

- Arrangement of computation dependencies (happened-before edges) across chains go only down to up.



Uniflow Chain Partition

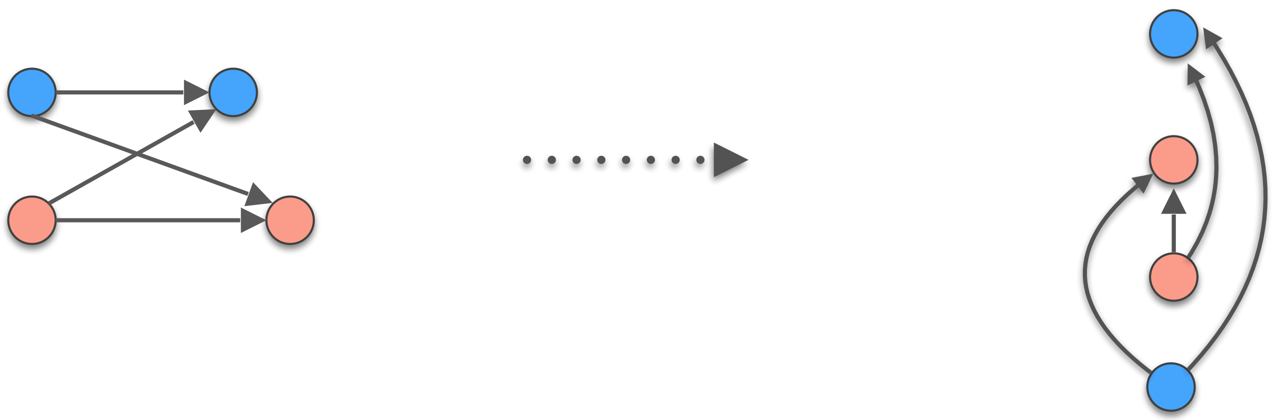
- Arrangement of computation dependencies (happened-before edges) flow either left to right, or down to up.



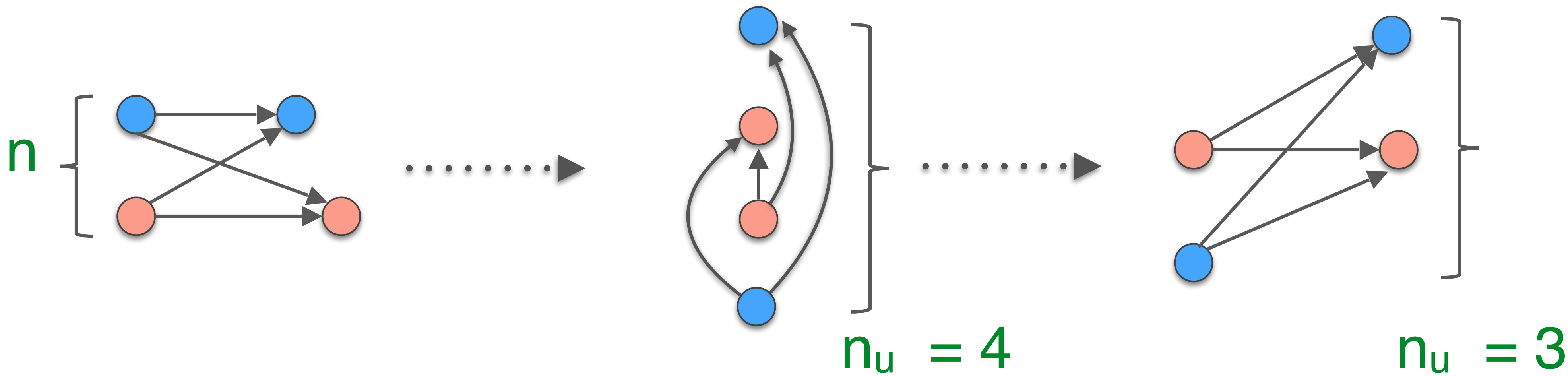
Uniflow Chain Partition

Lemma: Every computation has a uniflow partition.

Proof: Topological sort.



Optimal Uniflow Chain Partition

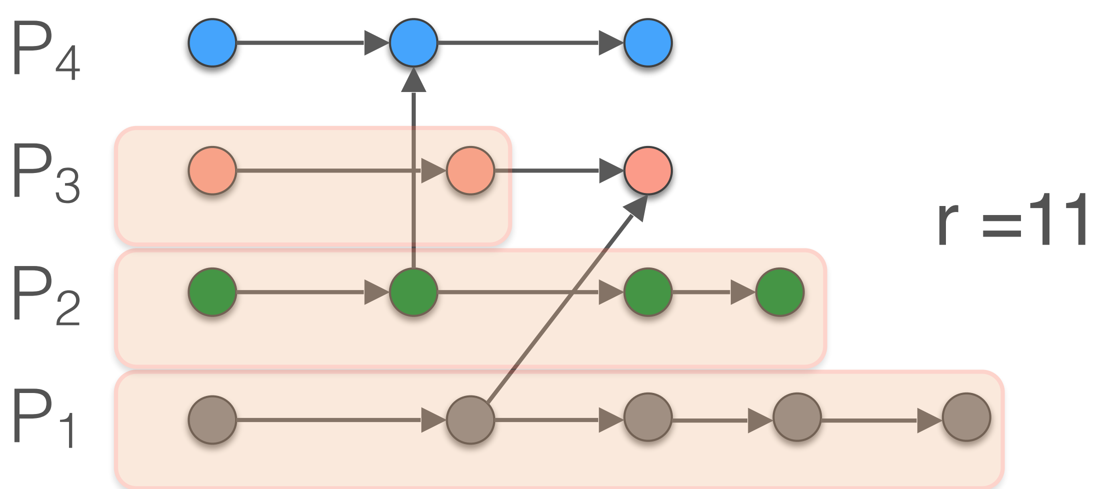


n_u is polynomial in input size: $n_u \leqslant E$; where E is # of events.

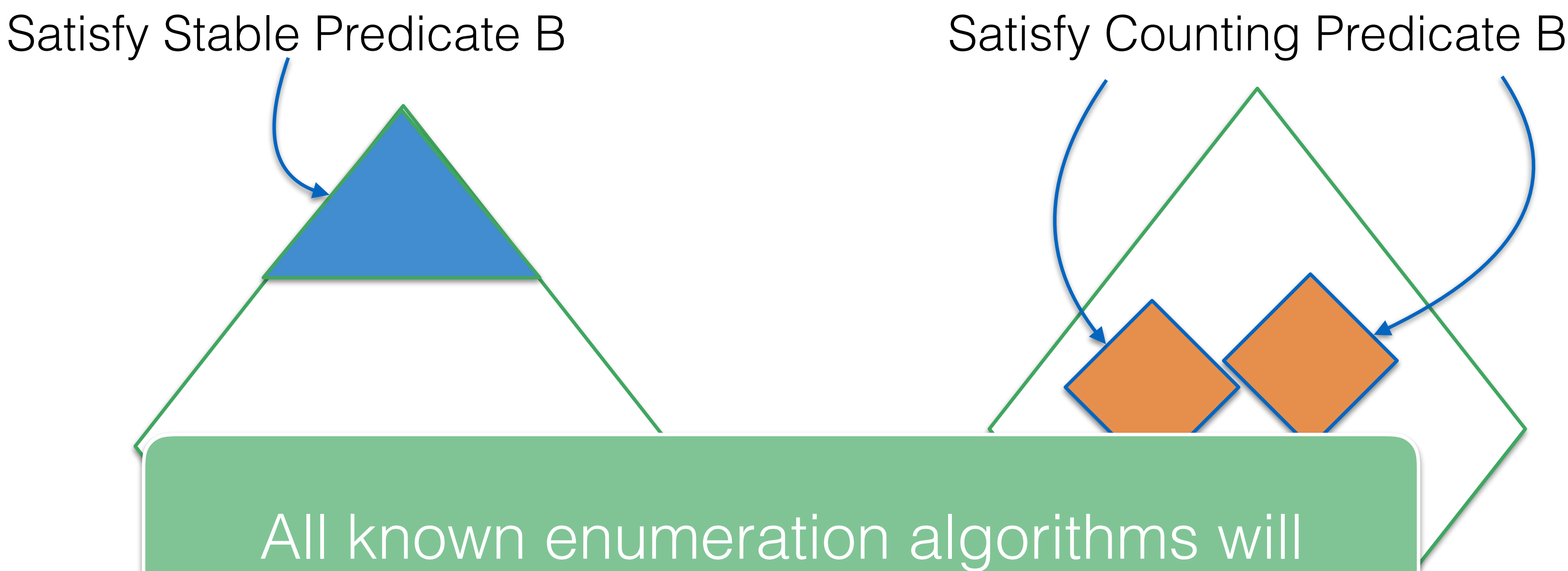
Finding the optimal uniflow chain partition is NP-Hard (jump number of a poset)

Cut formed with bottom *r* events

Lemma: Any cut formed with bottom *r* ($1 \leqslant r \leqslant |E|$) events of uniflow partition **is consistent.**



Enumerating Cuts satisfying Counting or Stable Predicates



All known enumeration algorithms will

Our algorithms only enumerate the cuts of the lattice that satisfy the predicate.

- Predicate Detection
- Uniflow Chain Partition
- Stable and Counting Predicates

> Enumerating cuts: Stable Predicates

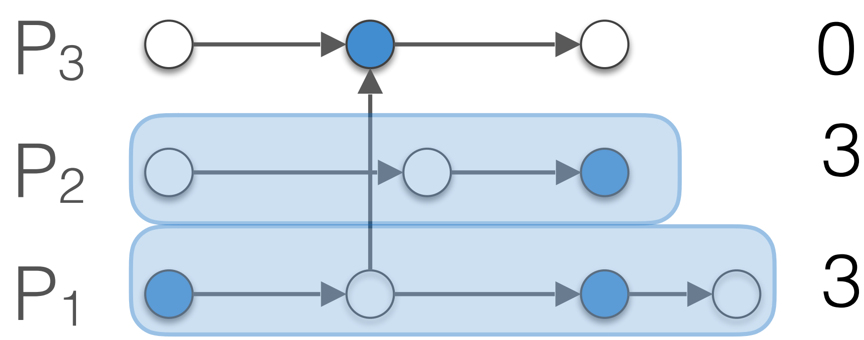
- Enumerating cuts: Counting Predicates

Enumerating Stable Predicates using Uniflow Partition

B = "3 or more blue events have been executed"

Step 1: G_B = FindSmallestCut(B, {})

//smallest cut that satisfies B and is bigger than {}.

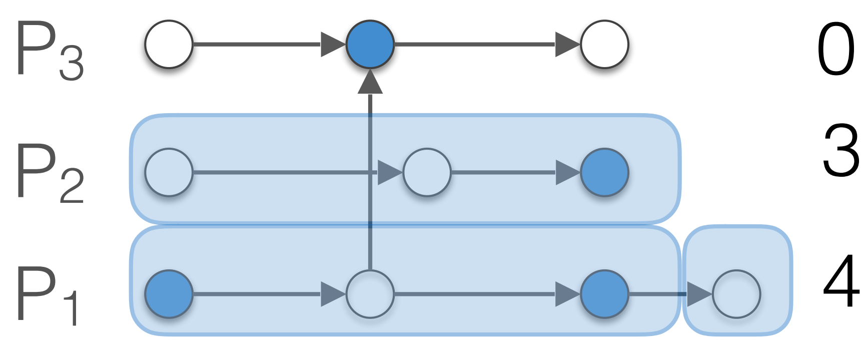


Enumerating Stable Predicates using Uniflow Partition

B = "3 or more blue events have been executed"

Step 2: G_B = FindSmallestCut(B, G_B)

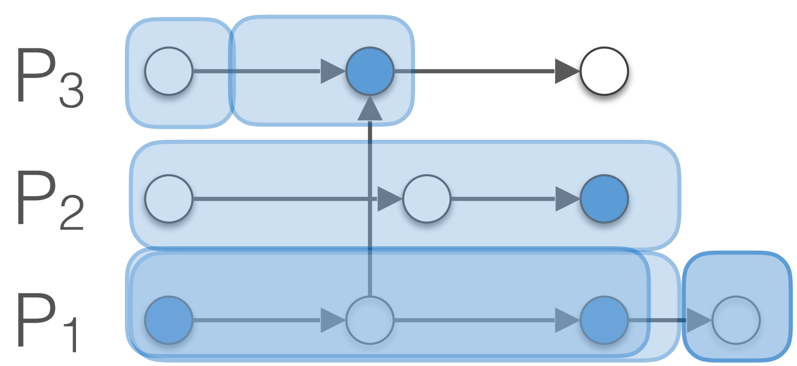
//smallest cut that satisfies B and is bigger than G_B.



Enumerating Stable Predicates using Uniflow Partition

G_B = FindSmallestCut(B, {})
while(true):
    enumerate(G_B)
    G_B = FindSmallestCut(B, G_B)
    if not expanded: break

B = "3 or more blue events have been executed"



0	1	1	2
3	3	3	0
4	3	4	3

…
so on

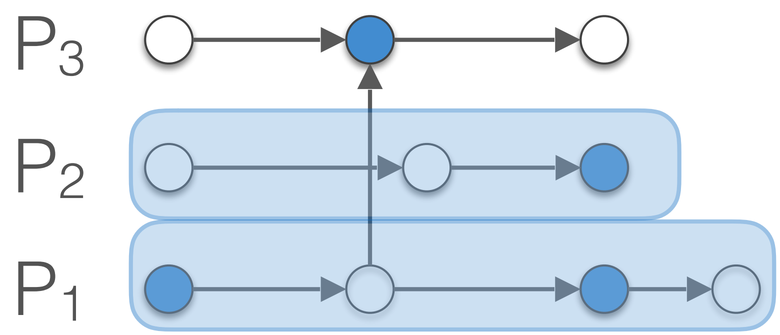
- Predicate Detection
- Uniflow Chain Partition
- Stable and Counting Predicates
- Enumerating cuts: Stable Predicates

> Enumerating cuts: Counting Predicates

Enumerating Counting Predicates using Uniflow Partition

B = "exactly 3 blue events have been executed"

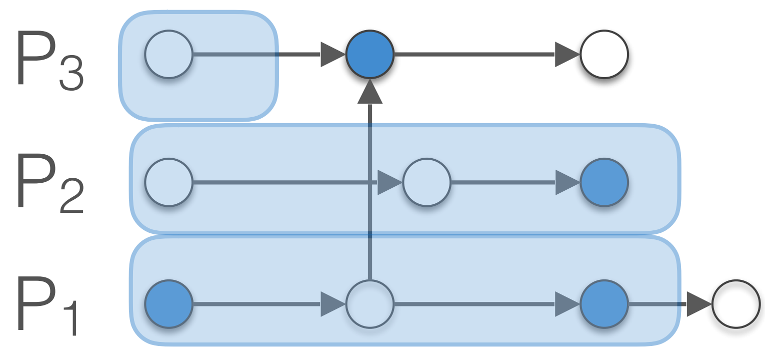
Step 1: G = FindSmallestCut(B)



Enumerating Counting Predicates using Uniflow Partition

B = "exactly 3 blue events have been executed"

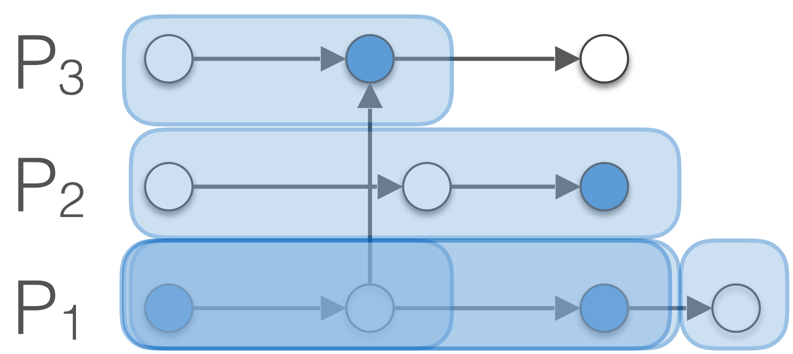
Step 2: EnumerateSameView(G,B)



Enumerating Counting Predicates using Uniflow Partition

B = "exactly 3 blue events have been executed"

Step 3: G = Successor(G,B)



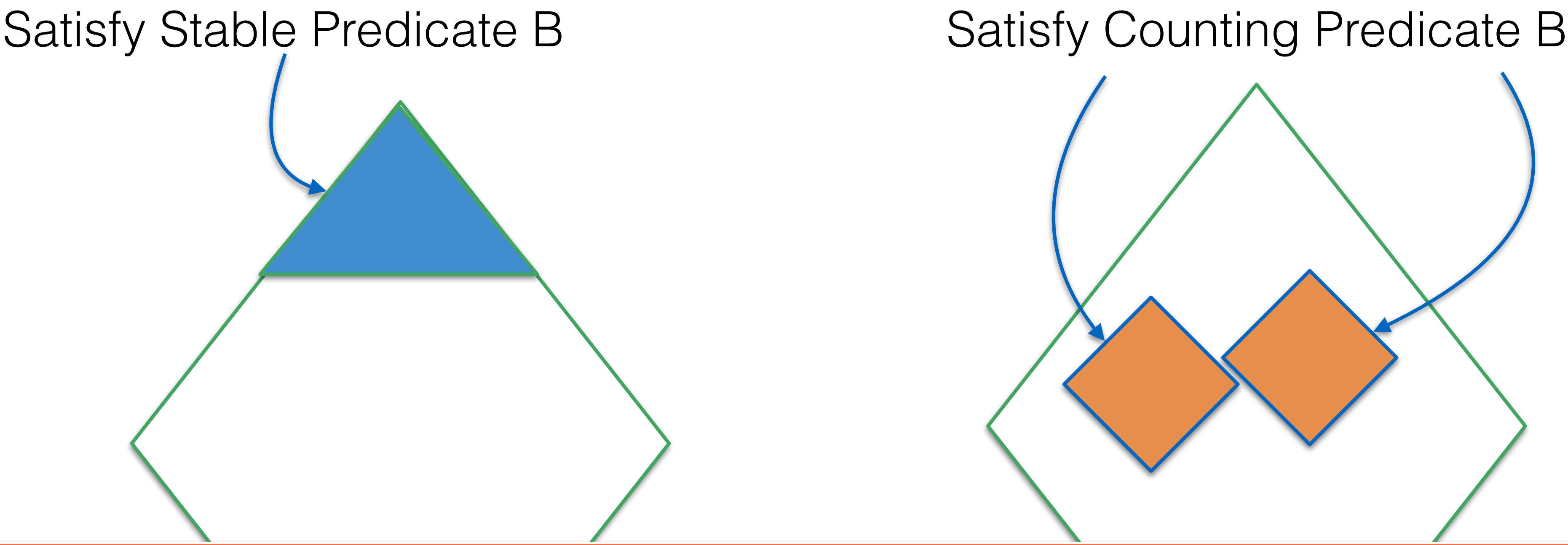
Enumerating Counting Predicates using Uniflow Partition

```
G = FindSmallestCut(B)
while(G != null)
        EnumerateSameView(G,B)
        G = Successor(G,B)
```

Theorem: Let $S_B \in \mathcal{C}(E)$ denote the set of consistent cuts that satisfy the stable or counting predicate B. Then, enumerating all consistent cuts in S_B takes $\mathcal{O}(f \cdot |S_B|)$ time using the algorithms given in this paper; where f is a polynomial function of $|E|$ (the number of events) and n (the number of processes).

In comparison, enumerating all the cuts of S_B using the existing algorithms such as BFS, DFS, Lex (or QuickLex) may take $\mathcal{O}(|\mathcal{C}(E)|)$ time in the worst case. Note that the $|\mathcal{C}(E)|$ can be exponentially bigger than $|S_B|$.

Summary



Our algorithms only enumerate the cuts of the lattice that satisfy the predicate.

Future Work

- Lower bounds on algorithms that enumerate global states satisfying stable and counting predicates

- Other interesting classes of predicates that can be efficiently enumerated.

Thanks.

Questions?