Thread-Modular Shape Analysis

Josh Berdine	MSR Cambridge
Byron Cook	MSR Cambridge
Alexey Gotsman	University of Cambridge
Tal Lev-Ami	Tel Aviv University
Roman Manevich	UCLA
G. Ramalingam	MSR India
Mooly Sagiv	Tel Aviv University
	UC Berkeley
	Cadence Berkeley
Michal Segalov	Tel Aviv University

Programs and Properties

- Concurrent programs
- Unbounded number of threads
  - parametric systems
- Unbounded number of objects
- Pointers and destructive updates

- Memory safety
  - Absence of null dereferences
  - Absence of memory leaks
- Preservation of data structure invariants
- Linearizability
- User-specified invariants

Concurrent Set [M. Maged SPAA'02]

```
remove(key) {
 while (true) {
  <prev,cur,next,found> = locate(key)
  if (!found) return false;
  if (CAS(prev.next, <0,cur>, <0,next>))
        DeleteNode(curr2);
  if (!CAS(cur.next, <0,next>, <1,next>)
        continue;
  else  locate(key);
 }
}
```

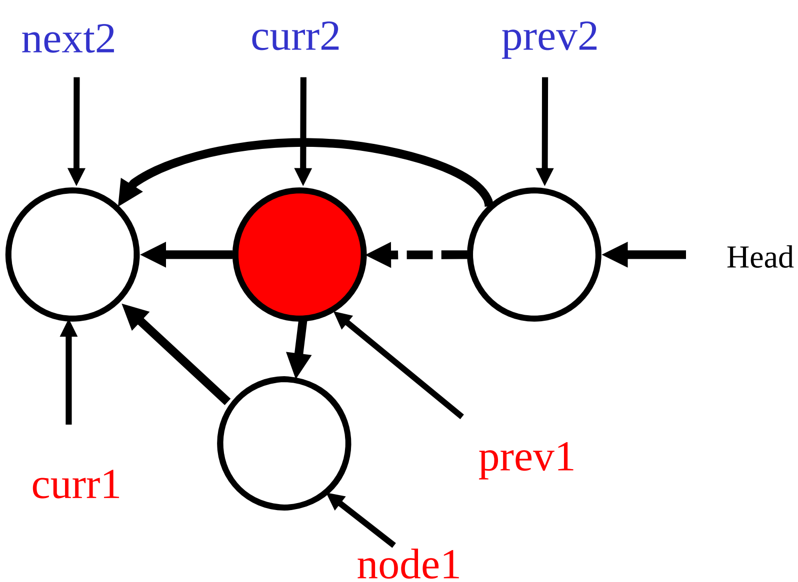
```
add(node) {
 while (true) {
 <prev,cur,next,found> = locate(node.key)
  if (found) return false;
  node.next = cur
  if (CAS(prev.next,  <0,cur>,  <0,node>))
      return true;
}
```

```
locate(key) {
restart: pred = Head ;
<tmp,curr> = pred.next;
 while (true) {
  if (curr == null) return <null, null, null, false>;
  <cmark, next> = curr.next;
  ckey = curr.key;
  if (pred.next != <0,curr>)  goto restart;
      if (!cmark) {
        if (ckey >= key) return <prev, curr, next, (key == ckey) >
        pred = curr;
      }
   else { if (CAS(pred.next, <0,curr>, <0,next>)) DeleteNode(curr);
           else goto restart; }
   curr = next; }
}
```

Concurrent Set [M. Maged SPAA'02]

```
remove(key) {
 while (true) {
  <prev2,cur2,next2,found> = locate(key)
  if (!found) return false;
  if (CAS(prev2.next, <0,curr2>, <0,next2>))
        DeleteNode(cur2);
  if (!CAS(cur2.next, <0,next2>, <1,next2>)
        continue;
  else  locate(key);
 }
}
```

```
add(node1) {
 while (true) {
 <prev,cur,next,found> = locate(node1.key)
  if (found) return false;
  node1.next = cur1
  if (CAS(prev1.next,<0,cur1>,<0,node1>))
      return true;
}
```



☒ A memory leak

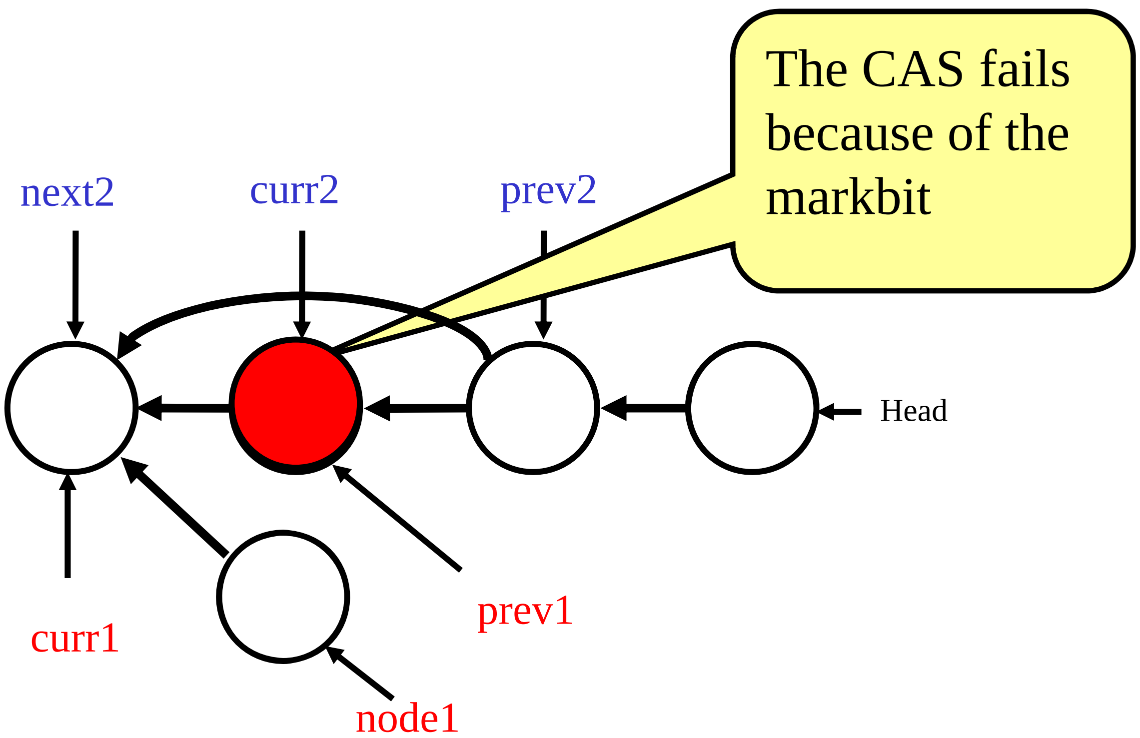
What is the bug?

- A node is removed before it is marked

```
remove(key) {
 while (true) {
 <prev,cur,next,found> = locate(key)
 if (!found) return false;
 if (!CAS(cur.next, <0,next>, <0,next>)
        continue;
 if (CAS(prev.next, <0,cur>, <1,next>))
        DeleteNode(cur);
 else  locate(key);
 }
}
```

```
remove(key) {
 while (true) {
  <prev2,cur2,next2,found> = locate(key)
  if (!found) return false;
  if (!CAS(cur2.next, <0,next2>, <1,next2>)
        continue;
  if (CAS(prev2.next,<0,cur2>, <0,next2>))
        DeleteNode(curr2);
  else  locate(key);
 }
}
```

```
add(node1) {
 while (true) {
 <prev1,cur1,next1,found>=locate(node1.key)
 if (found) return false;
 node1.next = curr1
 if (CAS(prev1.next, <0,curr1>, <0,node1>))
      return true;
}
```



Captured Invariants

- No memory leaks
  - Every “dangling” pointer is pointed-to by some thread reachable from **`Head`**, or has been returned by some remove method
- After a successful add, **`prev`** is reachable from **`Head`**, the node inserted is pointed-to by **`prev`** and it points to **`curr`**
- Only a single node can be added/removed by each operation
- An outgoing edge of a marked node is immutable

Challenges

- Develop an analysis which automatically proves interesting properties of concurrent heap-manipulating programs
  - Concurrency is challenging
  - The global nature of the heap
- Designing the right abstraction
- Developing effective transformers
  - Sound proof rules for atomic statements

A Singleton Buffer

```
Boolean empty = true;

Object b = null;
```

```
produce() {
1: Object p = new();
2: await (empty) then {
       b = p;
       empty = false;
   }
3:
}
```

```
consume() {
Object c;
4: await (!empty) then {
        c = b;
        empty = true;
   }
5: use(c);
6: dispose(c);
7:
}
```

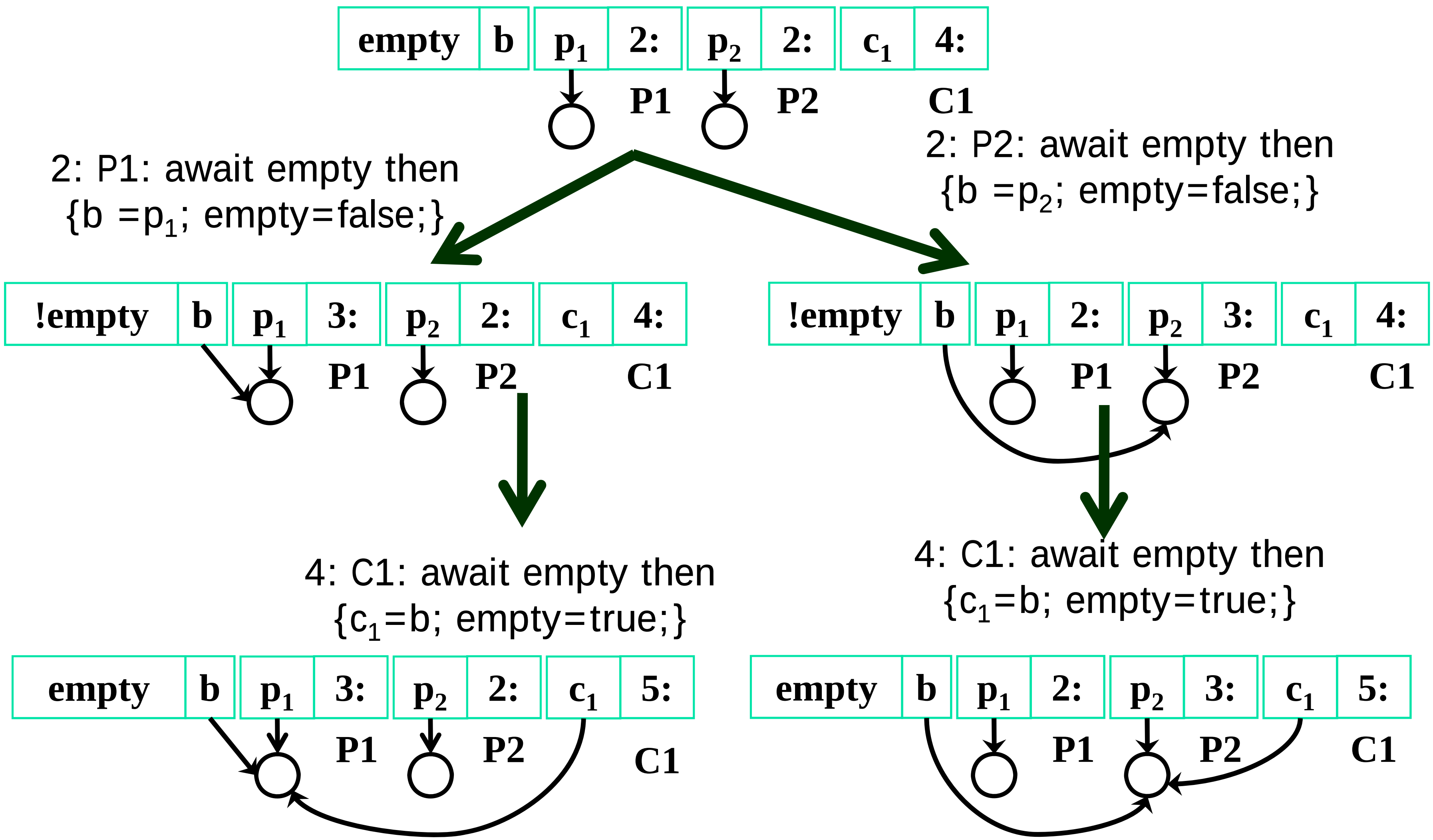
Safe Dereference

No Double free

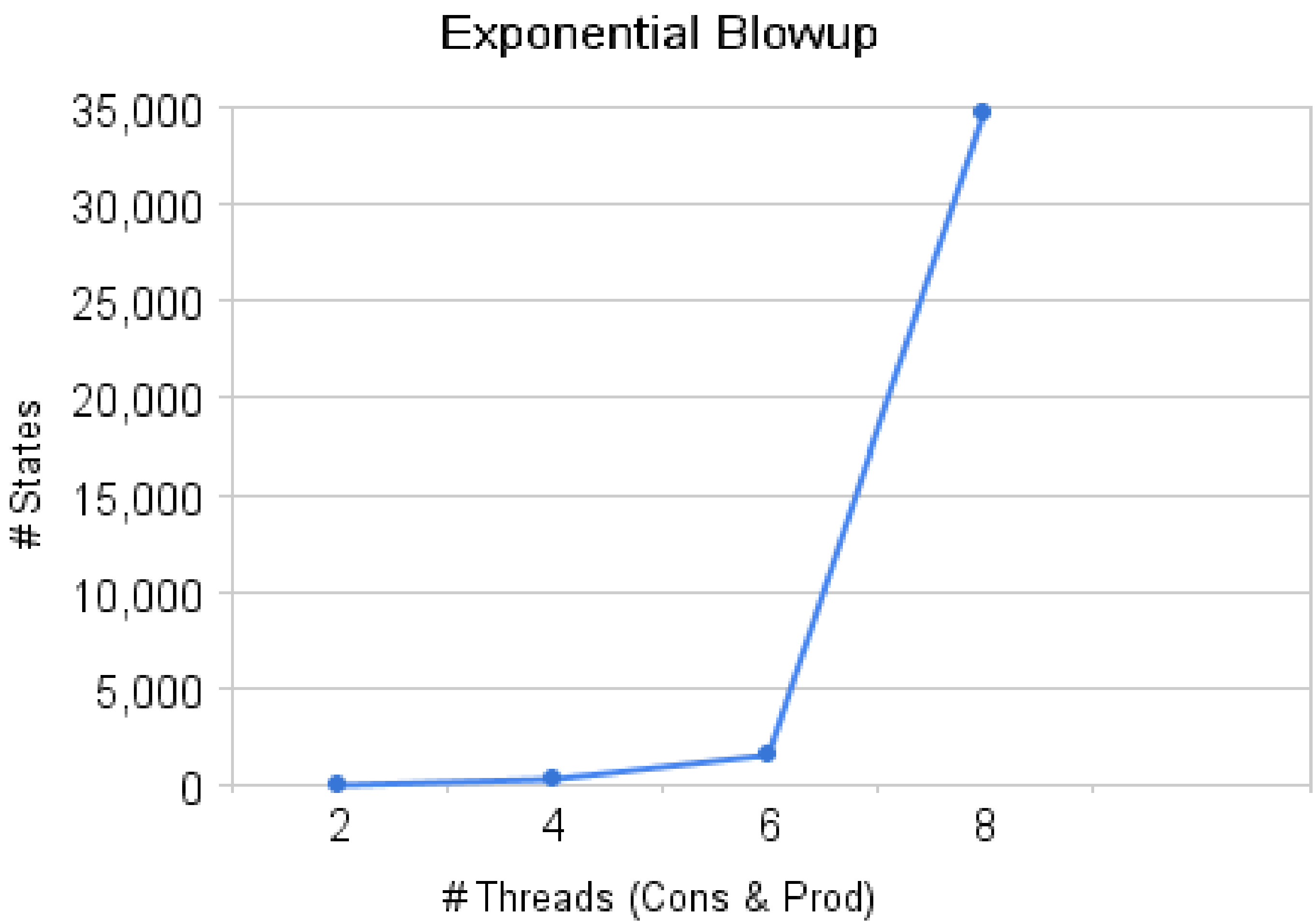
State Space Exploration

- Enumerate all interleavings
- Check the properties

Partial State Space Exploration 1 consumer / 2 producers



State Space Explosion (bounded number of threads)



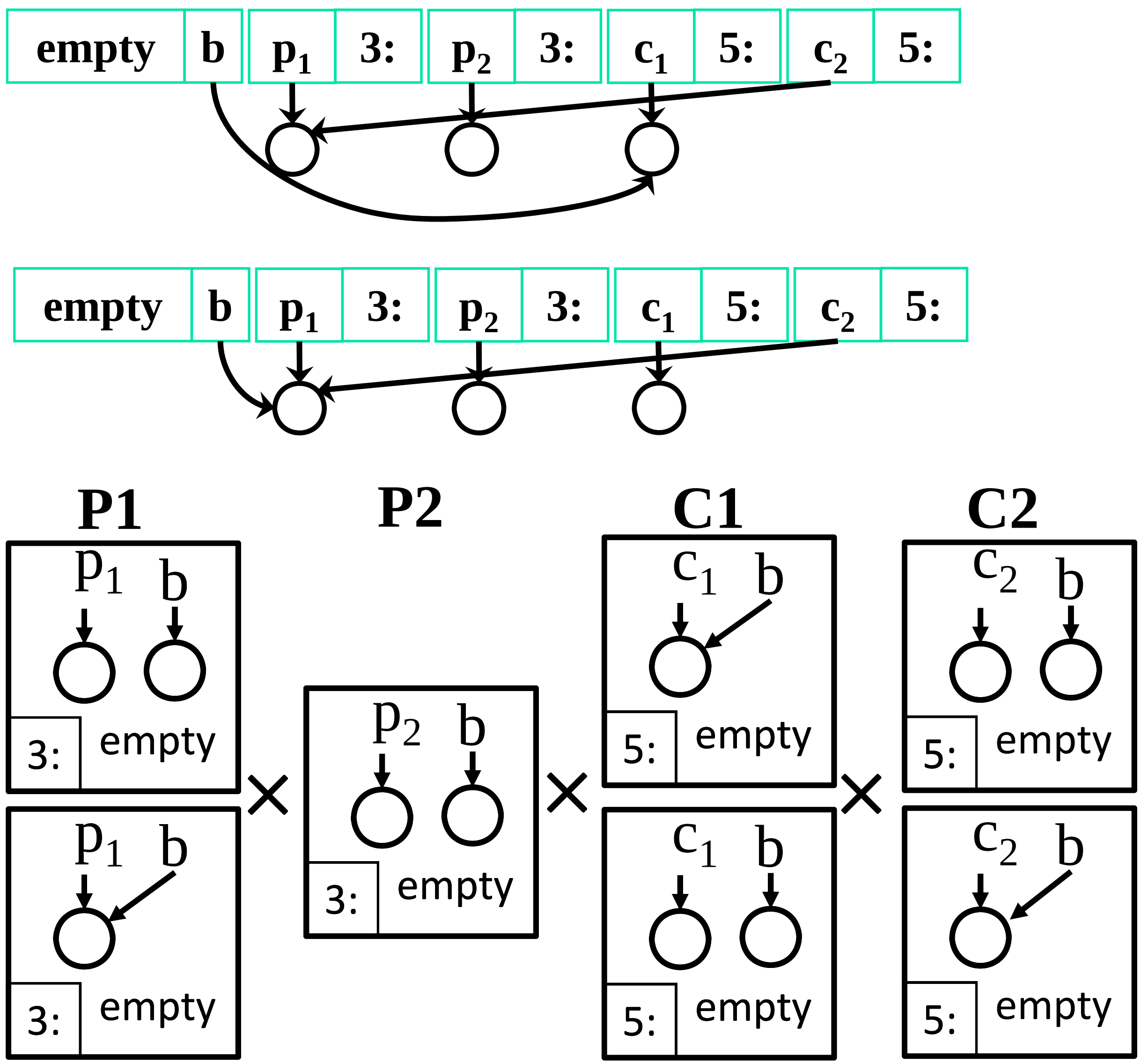
Plan

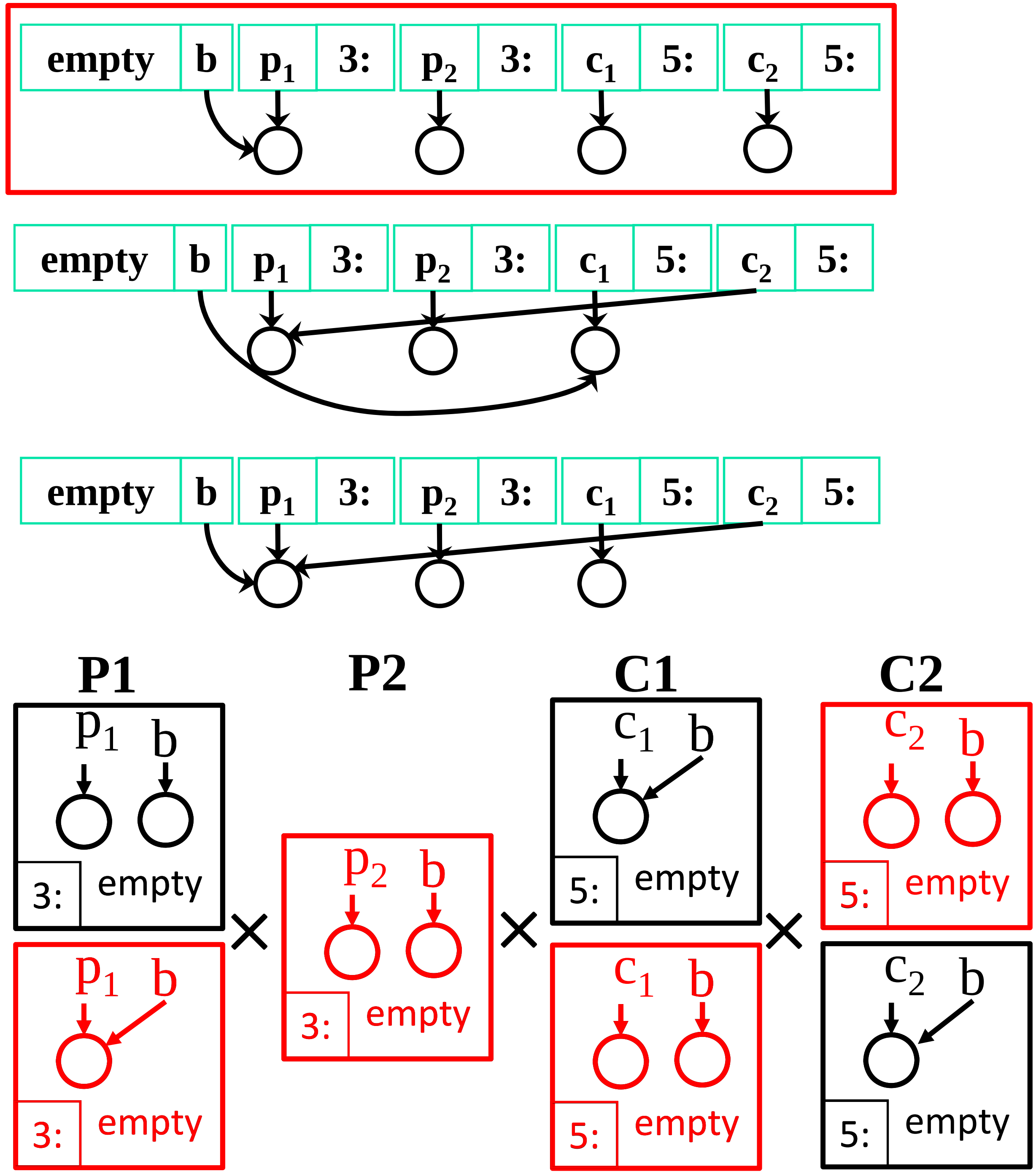
- Thread-modular analysis
- Semi-thread-modular analysis
- Unbounded number of threads
- Empirical results

Thread-Modular Analysis

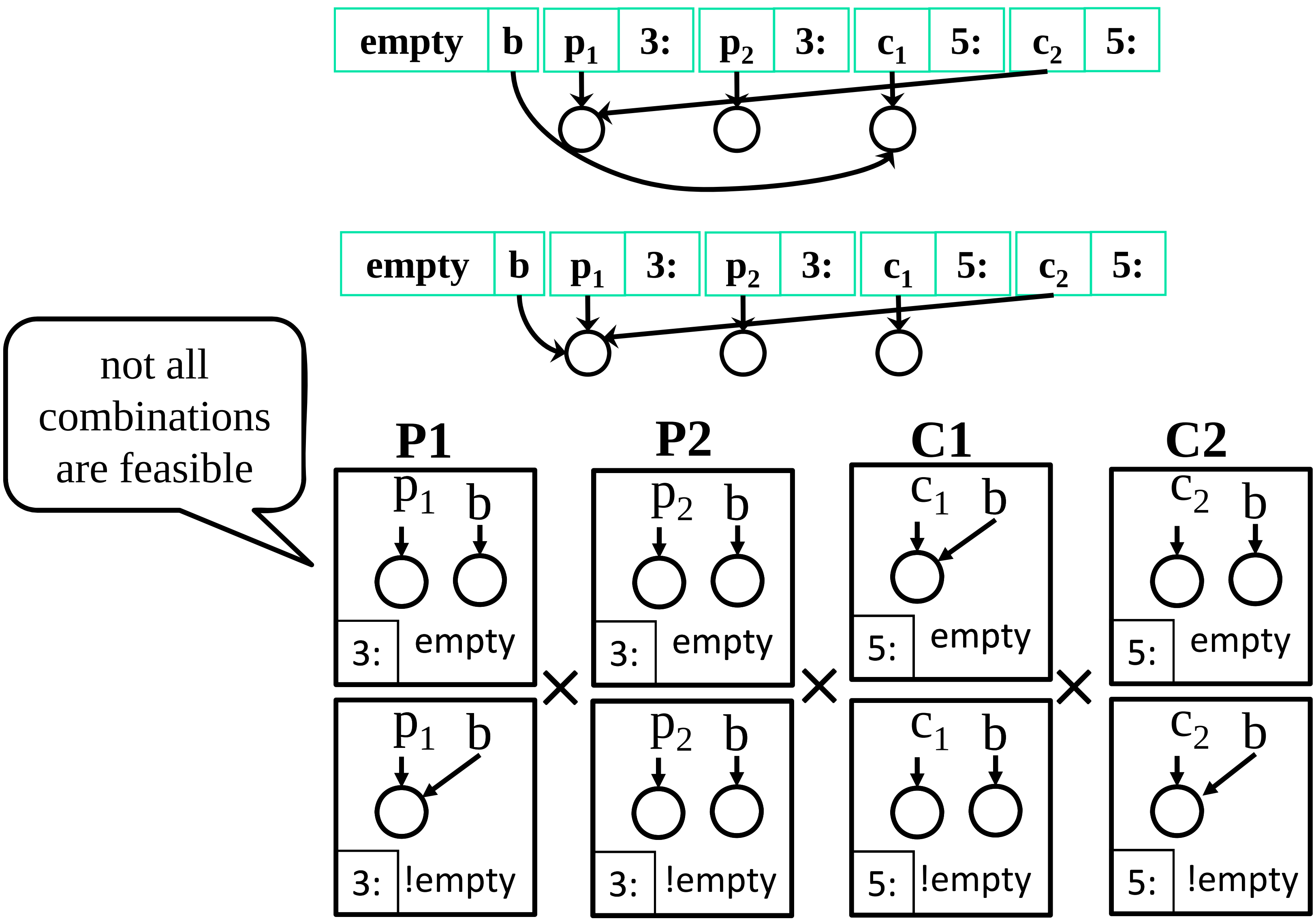
- Abstract away the correlations between local states of different threads
  - No correlations between program counters
  - Cartesian Abstraction
- Information maintained
  - Correlations between the local state and global state of each thread
- “The quadratic cost of computing transformers can be greatly reduced...” [Flanagan & Qadeer SPIN, 2003]
- Naturally handles unbounded number of threads

Thread-Modular Abstraction

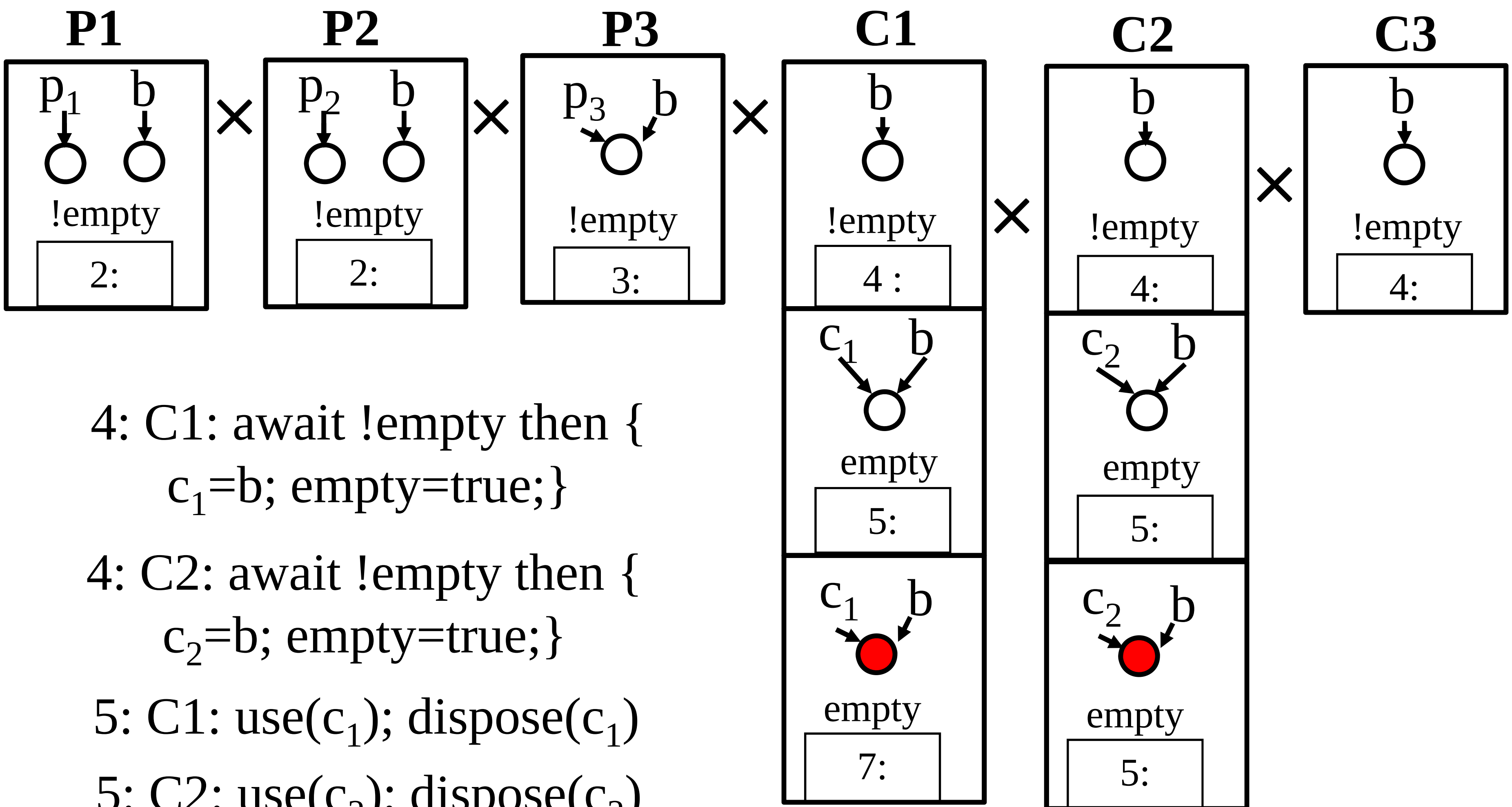



empty b p1 3: p2 3: c1 5: c2 5:
empty b p1 3: p2 3: c1 5: c2 5:
empty b p1 3: p2 3: c1 5: c2 5:
P1
P2
C1
C2
p1 b
3: empty
p1 b
3: empty
p2 b
3: empty
c1 b
5: empty
c1 b
5: empty
c2 b
5: empty
c2 b
5: empty

Thread-Modular Abstraction



Partial Abstract Interpretation

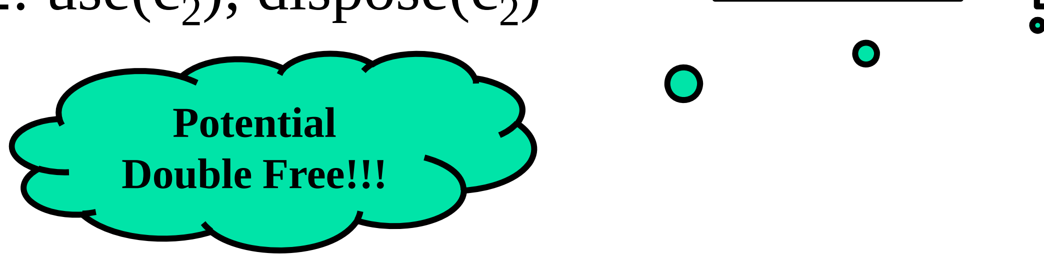


4: C1: await !empty then {
c_1=b; empty=true;}

4: C2: await !empty then {
c_2=b; empty=true;}

5: C1: use(c_1); dispose(c_1)

5: C2: use(c_2); dispose(c_2)



A Singleton Buffer

```
Boolean empty = true;

Object b = null;
```

```
produce() {
1: Object p = new();
2: await (empty) then {
        b = p;
        empty = false;
    }
3:
}
```

```
consume() {
Object c;
4: await (!empty) then {
        c = b;   b=null;
        empty = true;
    }
5: use(c);
6: dispose(c);
7:
}
```

Safe Dereference

No Double free

Thread-Modular Analysis

- Abstract away the correlations between local states of different threads
  - No correlations between program counters
  - Cartesian Abstraction
- Information maintained
  - Correlations between the local state of each thread and the global state
- Scales with the number of threads
- Handles unbounded number of threads
- But limited precision

Increasing Precision

- Enforce program restrictions
  - Limited aliasing
  - Ownership relations [Boyapati et. al. OOPSLA'02]
  - Limited concurrency
- Enhanced analysis
  - Global instrumentation
  - Separation Domains [Gotsman et. al. PLDI'07]
  - Semi-Thread Modular Analysis [Berdine et. al. CAV'08, Segalov et. al., TR]

Microsoft Development Environment [design] - kbdclass.c [Read Only]

File Edit View Debug Tools Window Help

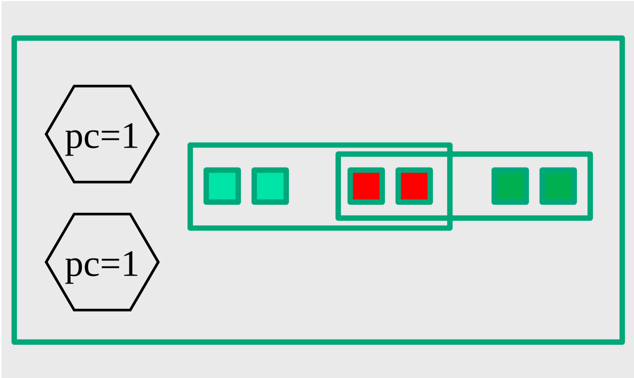
kbdclass.c

```c
{
    PIRP irp;
    LIST_ENTRY listHead, *entry;
    KIRQL irql;

    InitializeListHead(&listHead);

    KeAcquireSpinLock(&DeviceExtension->SpinLock, &irql);

    do {
        irp = KeyboardClassDequeueReadByFileObject(DeviceExtension, FileObject);
        if (irp) {
            irp->IoStatus.Status = STATUS_CANCELLED;
            irp->IoStatus.Information = 0;

            InsertTailList (&listHead, &irp->Tail.Overlay.ListEntry);
        }
    } while (irp != NULL);

    KeReleaseSpinLock(&DeviceExtension->SpinLock, irql);

    //
    // Complete these irps outside of the spin lock
    //
    while (! IsListEmpty (&listHead)) {
        entry = RemoveHeadList (&listHead);
        irp = CONTAINING_RECORD (entry, IRP, Tail.Overlay.ListEntry);
```

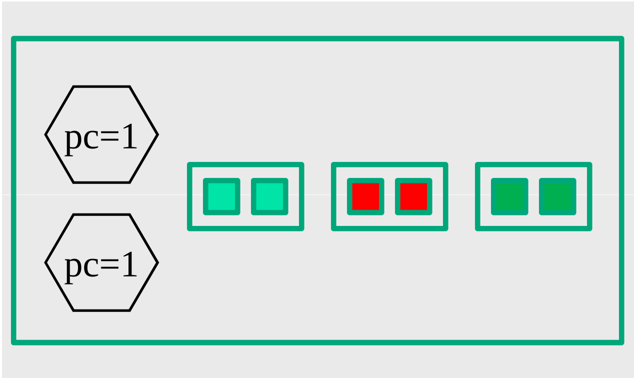
Ready | Ln 971 | Col 2 | Ch 2 | INS

Thread-Modular Analysis

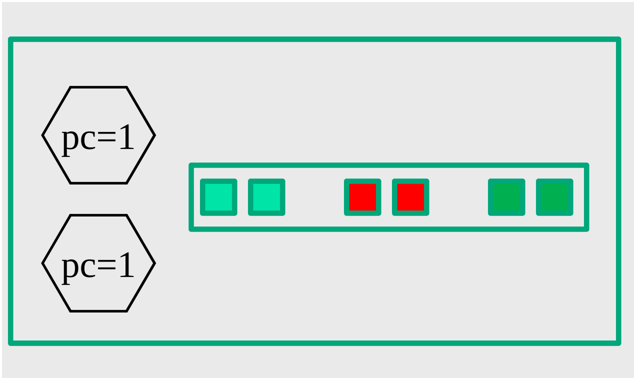
Non-disjoint resource invariants
[the rest of this talk]
Fine-grained concurrency



Separated resource invariants
[Gotsman et al., PLDI 07]
Coarse-grained concurrency



Single global resource invariant
[Flanagan & Qadeer, SPIN 03]



Thread Quantification for Concurrent Shape Analysis

J. Berdine, T. Lev-Ami, R. Manevich, G. Ramalingam, M. Sagiv

CAV'08

Semi-Thread-Modular Analysis

M. Segalov, T. Lev-Ami, R. Manevich, G. Ramalingam, M. Sagiv

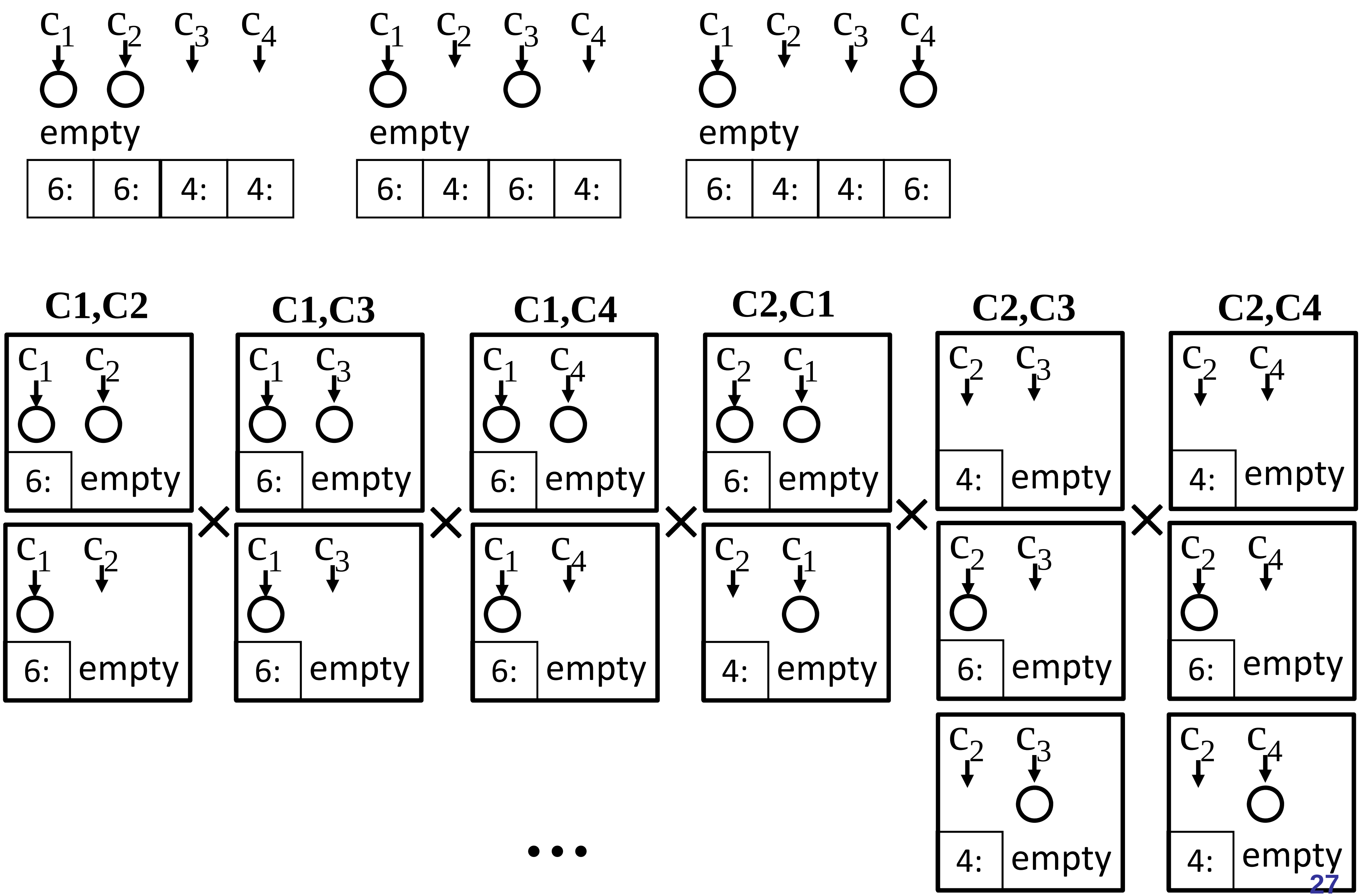
Main Results

- A refinement of thread-modular analysis
  - Not fully modular
- Precise enough to prove properties of fine-grained concurrent programs
  - Were not automatically proved before
- Two effective methods for efficiently computing transformers
  - Summarizing Effects
  - Summarizing Abstraction
  - On a concurrent set imp. speedup is x34!

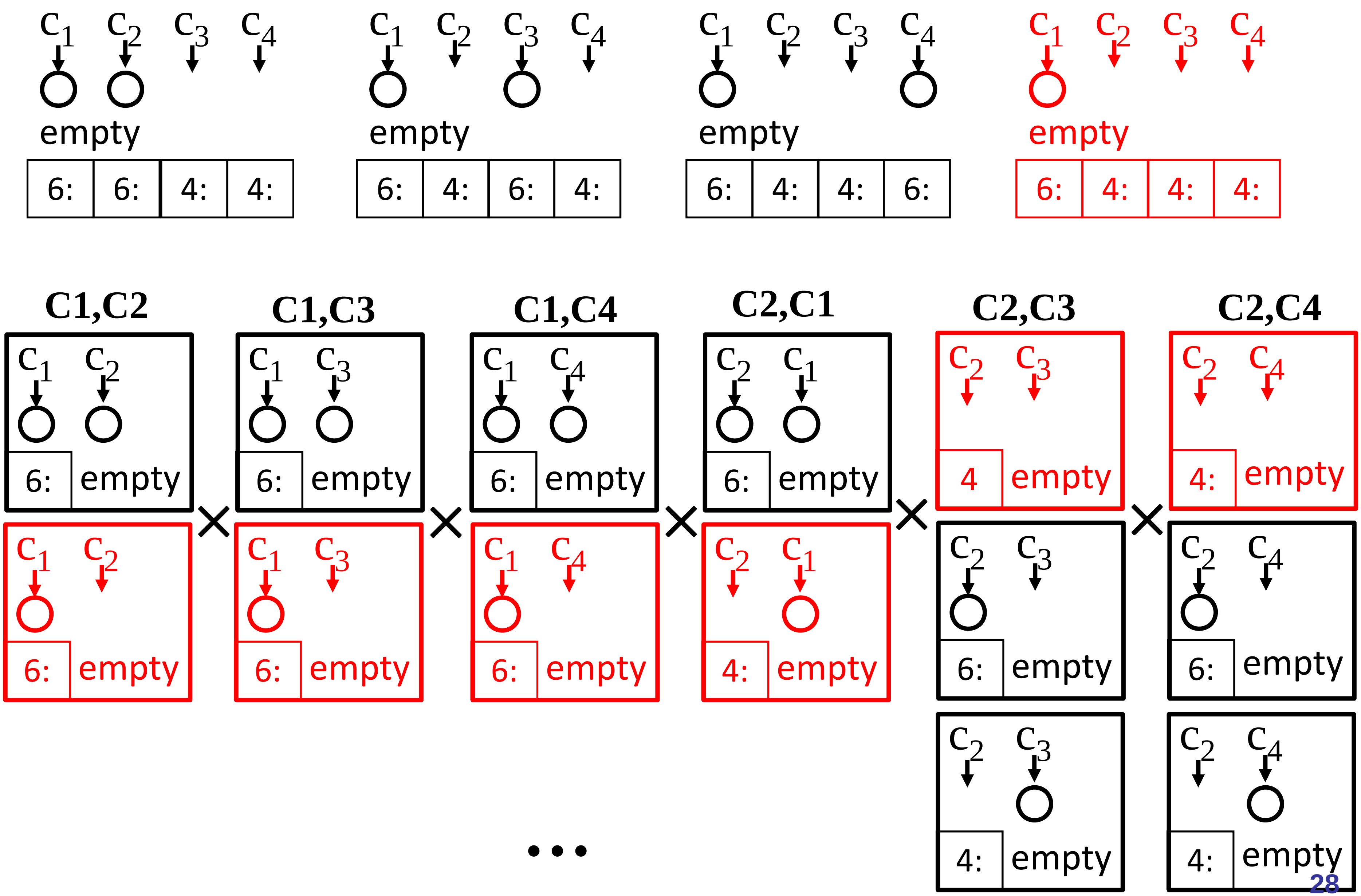
Semi-Thread-Modular Analysis

- Abstract away correlations between local states of more than two threads
- Information maintained
  - Correlations between the local state of each thread and the global state
  - May-correlations between local states of every pair of threads
    - Not necessarily symmetric

Semi-Thread-Modular Abstraction



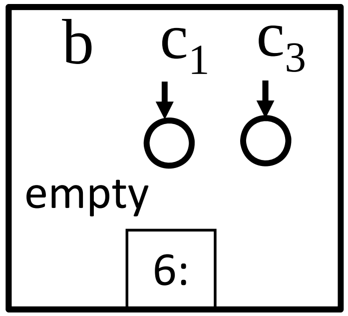
Semi-Thread-Modular Concretization



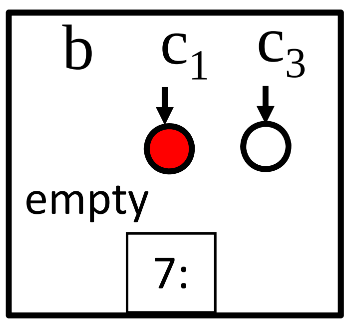
Worst-Case Complexity

- Full state analysis
  - Shared state – G, Local state – L_{tid}
  - State space = $\wp(G \times L_1 \times \ldots \times L_n)$
  - #states: $O(|G| \cdot |L|^n)$
- Thread-modular analysis
  - State space = $\wp(G \times L_1) \times \ldots \times \wp(G \times L_n)$
  - #states: $O(n \cdot |G| \cdot |L|)$
- Semi-thread-modular analysis
  - State space = $\wp(G \times L_1 \times L_2) \times \ldots \times \wp(G \times L_{n-1} \times L_n)$
  - #states: $O(n \cdot |G| \cdot |L|^2)$

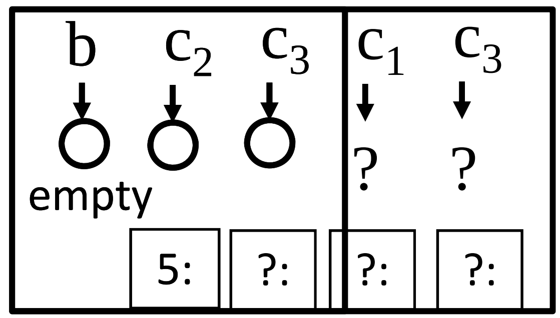
Point-wise Transformer 6: C1: dispose(c_1)



6: C1: dispose(c_1)



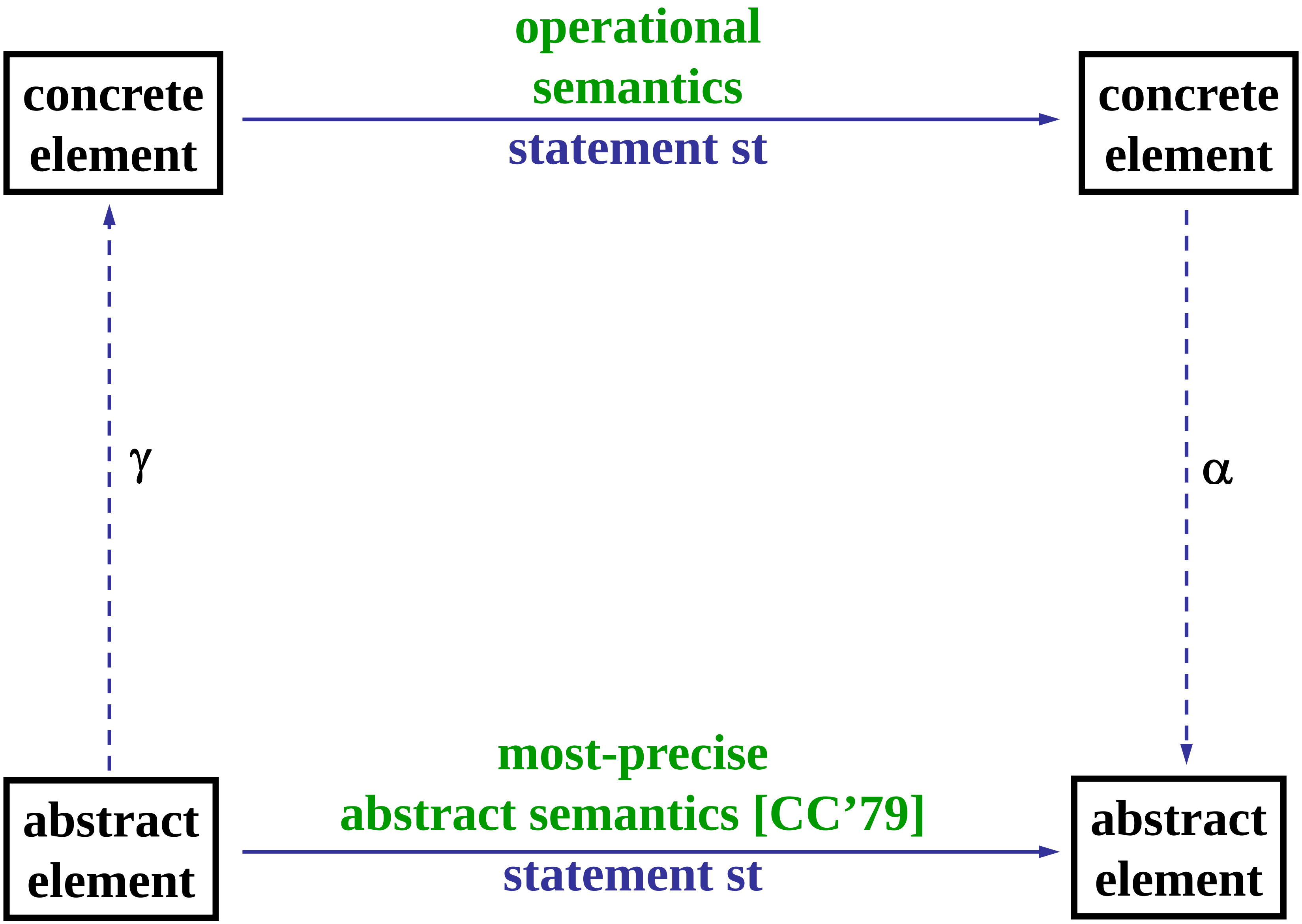
Point-wise Transformer
6: C1: dispose(c_1)



6: C1: dispose(c_1)

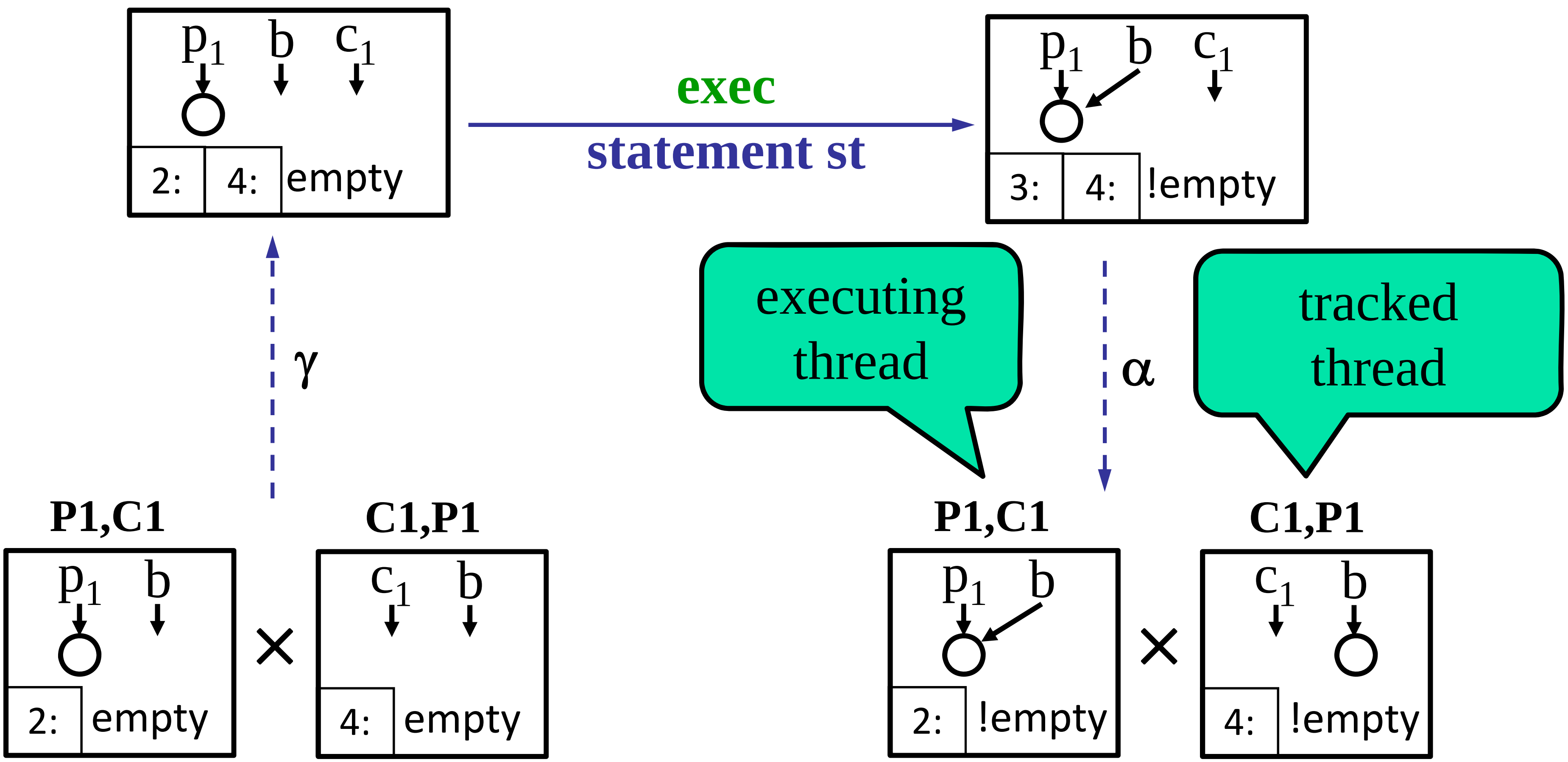
Is this command safe in this configuration?
Missing information on c_1
Unknown effect on b

Most-Precise Transformer

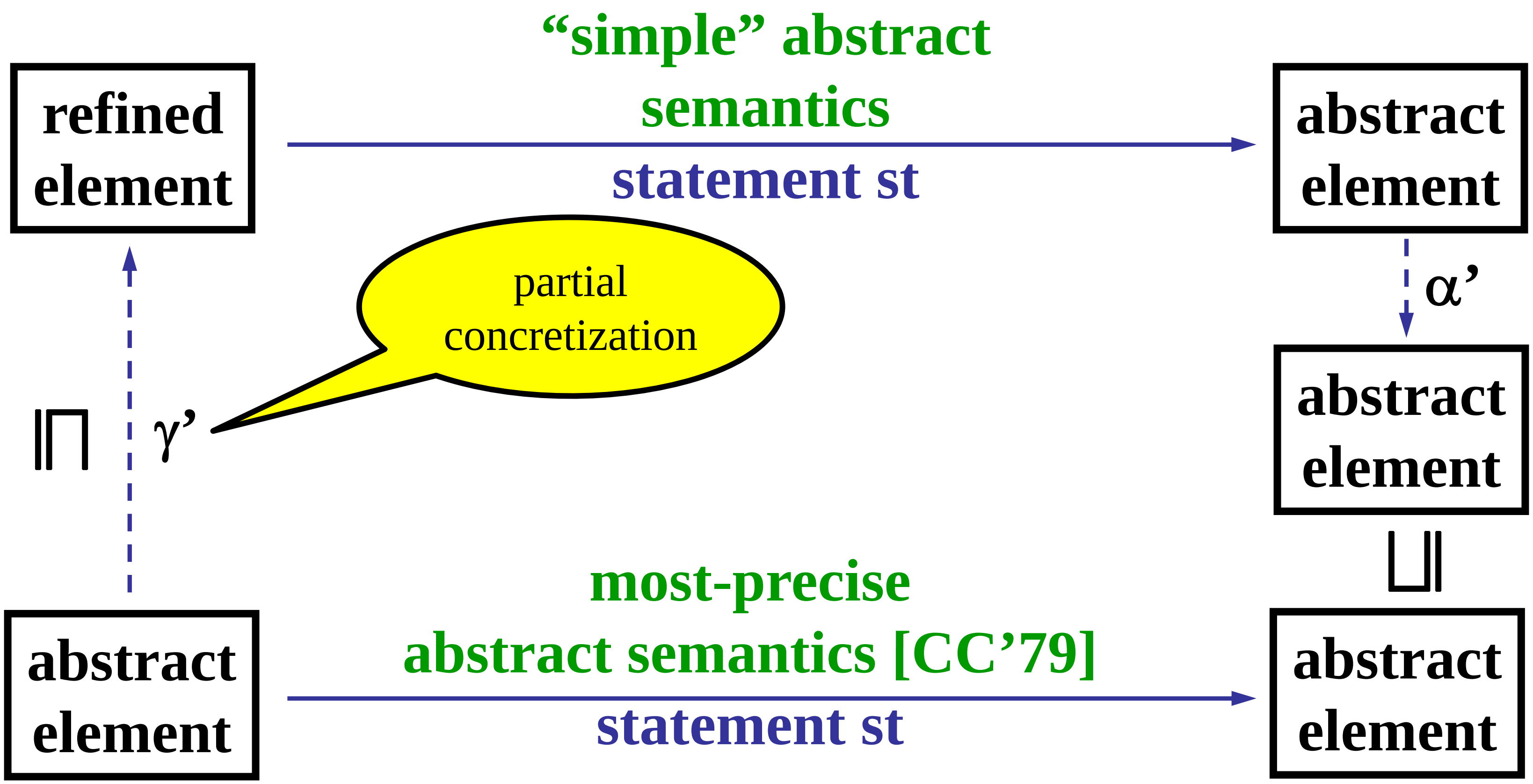


Most-precise transformer

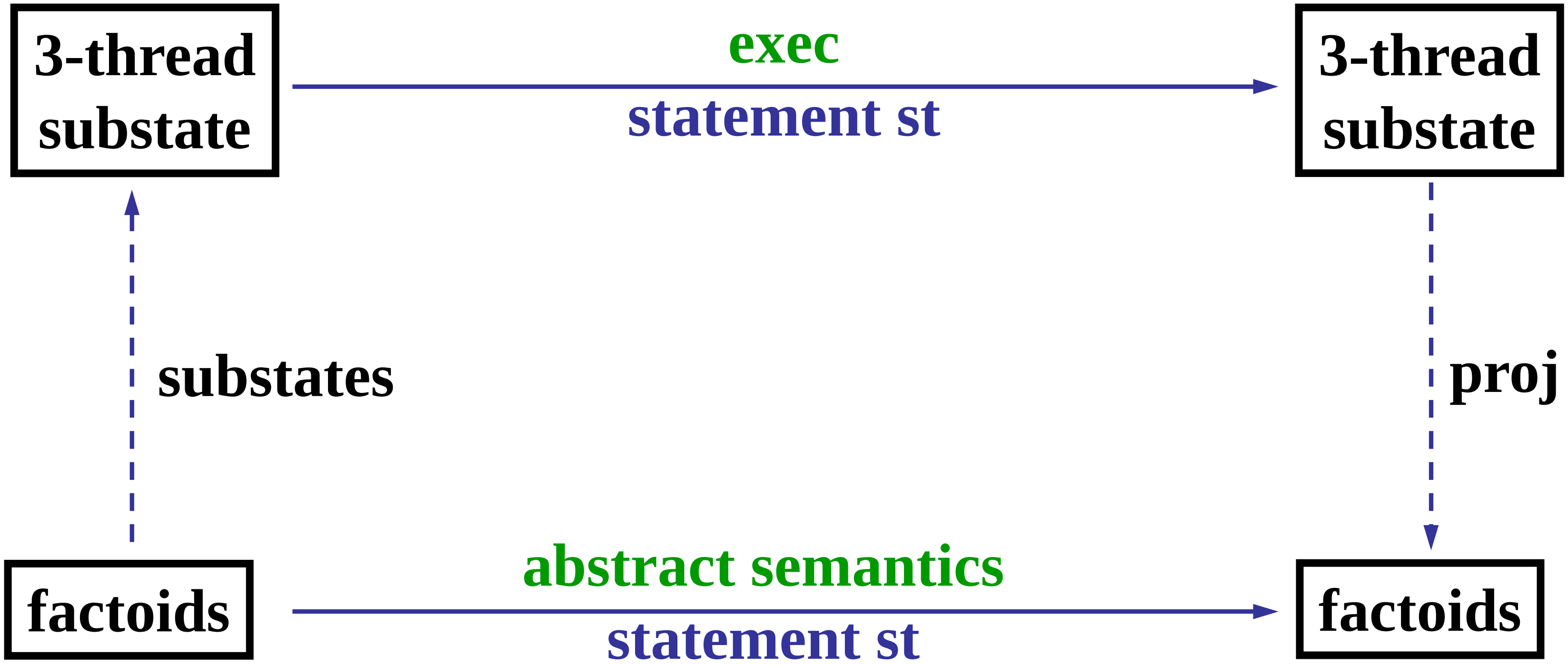
2: P1: await (empty) then { b=p_1; empty=false; }



Sound Transformer



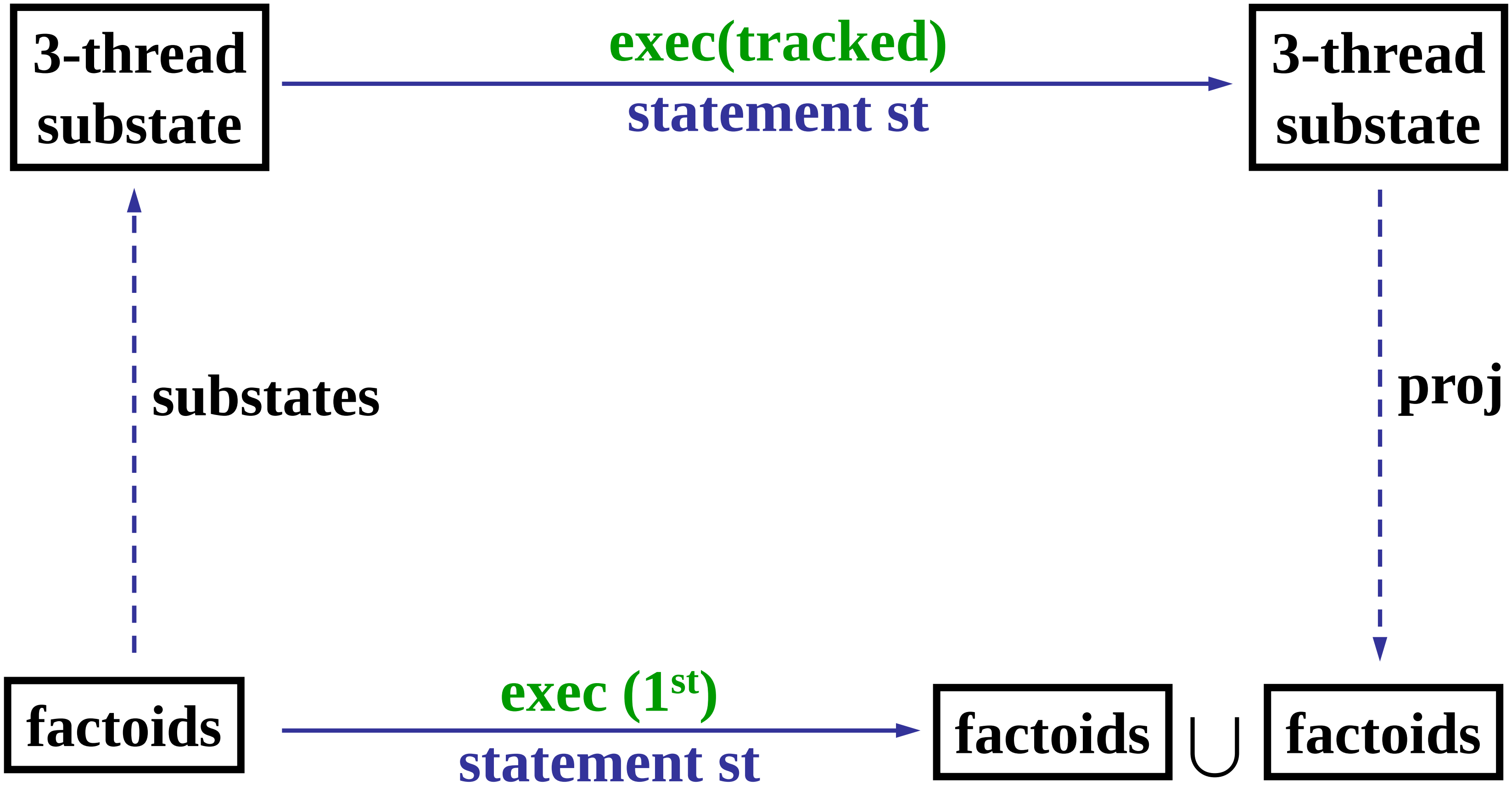
Partial Concretization-based Transformer



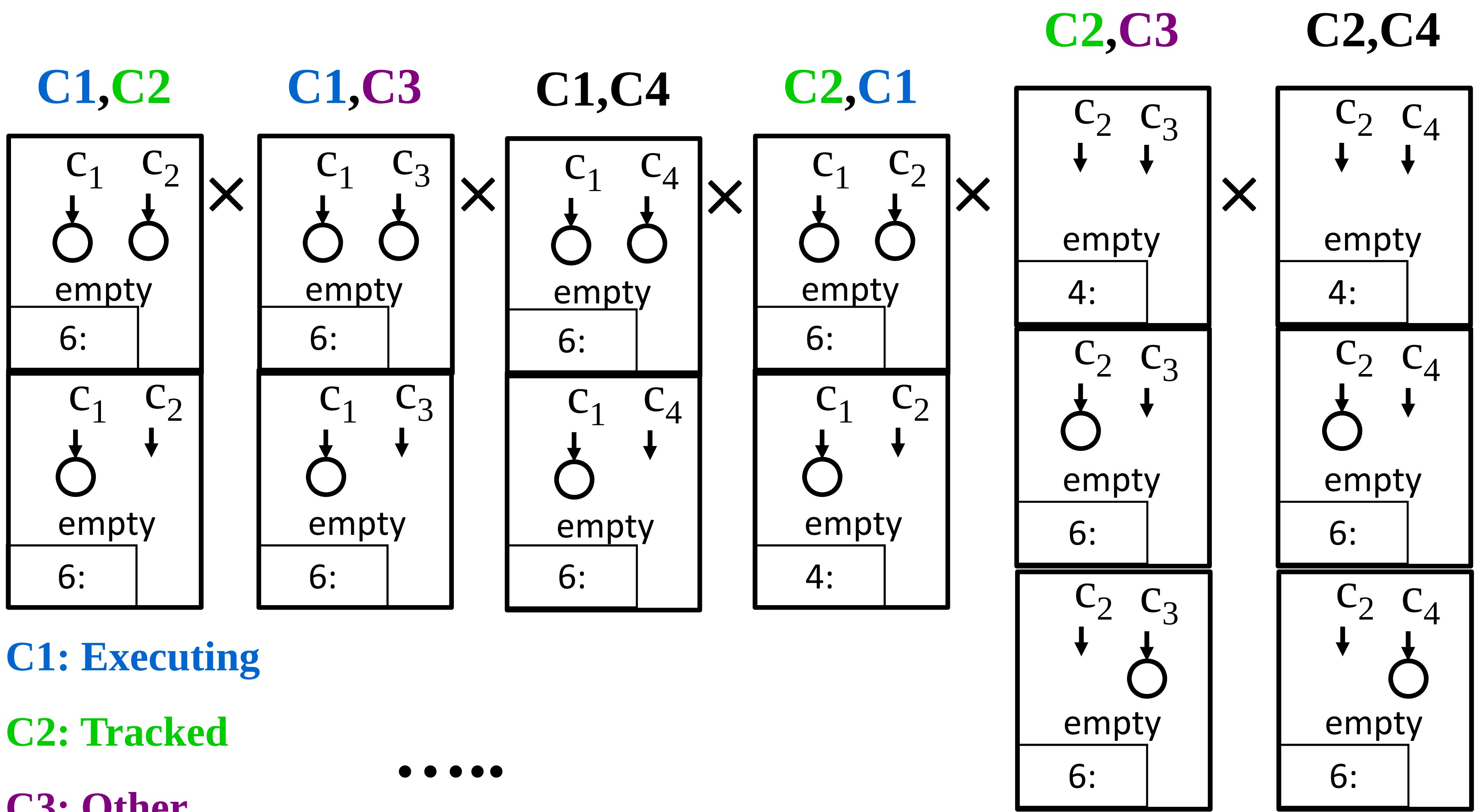
Transformer for Concurrent Systems

$$TR(F) = \{\langle l', g', o\rangle : \langle l, g, o\rangle \in F, \langle l, g\rangle\ \tau\ \langle l', g'\rangle\}$$

$$\cup \left\{ \begin{array}{l} \langle l_2, g', o\rangle, \langle l_2, g', \alpha(l_1')\rangle : f_1, f_2, f_3, f_4 \in F: \\ \quad \langle l_1, g, l_2, o\rangle \in substates(f_1, f_2, f_3, f_4), \\ \quad \langle l_1, g\rangle\ \tau\ \langle l_1', g'\rangle \end{array} \right\}$$



Partial Concretization

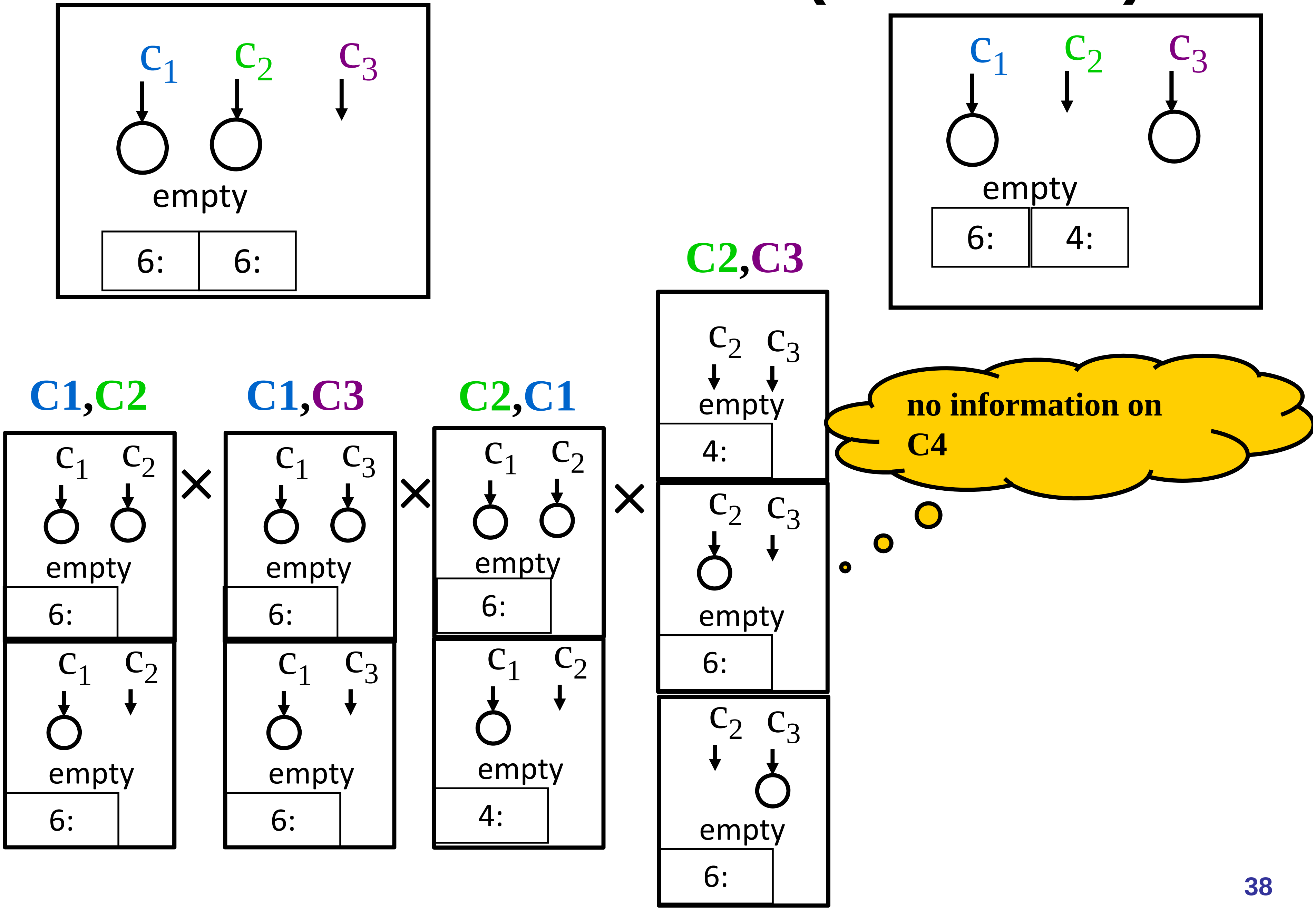


C1: Executing

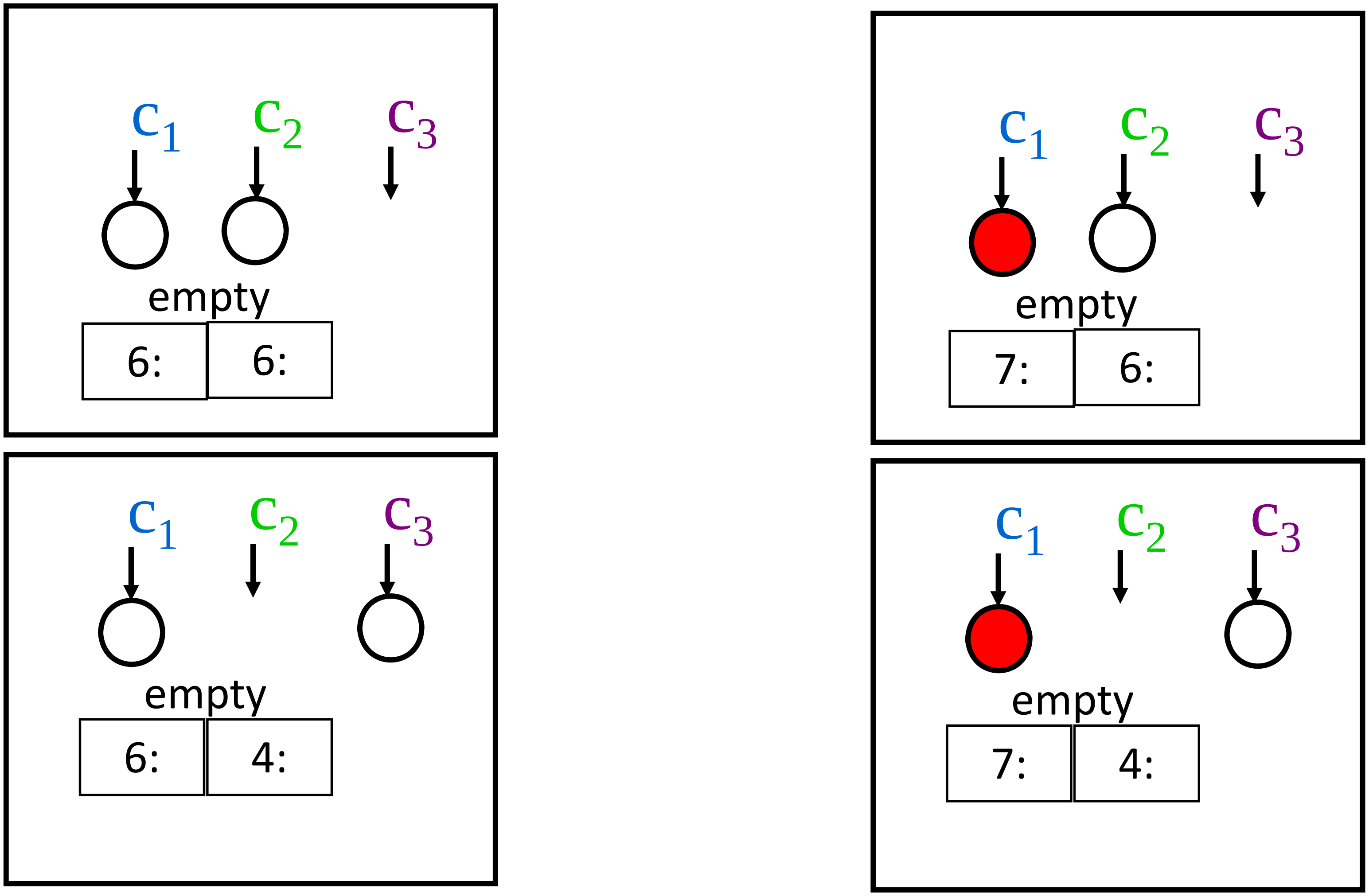
C2: Tracked

C3: Other

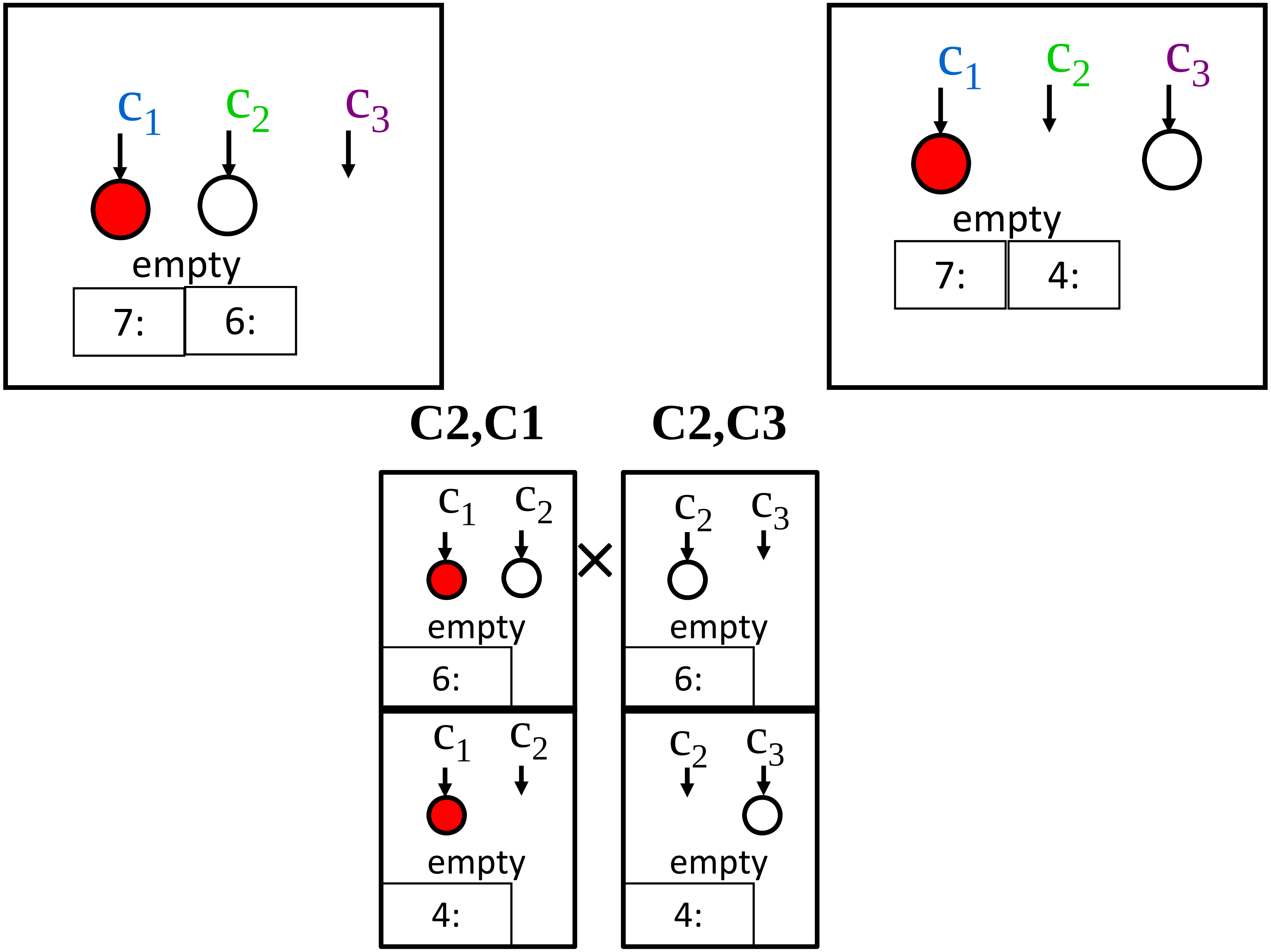
Partial Concretization(Substates)



6: C1: dispose(c) (exec)



6: C1: dispose(c) (project)



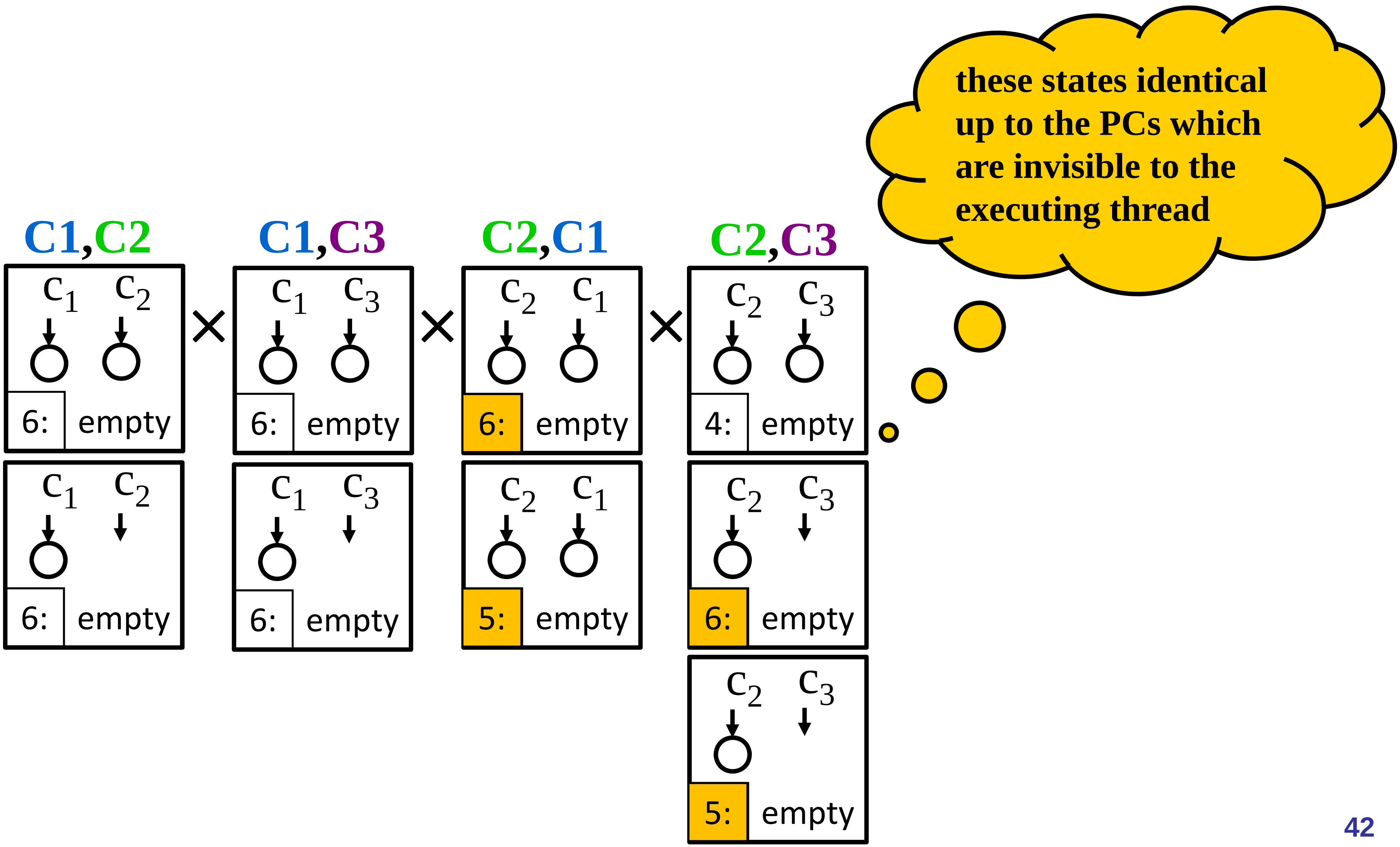
Reducing Quadratic Factors

$$TR(F) = \{\langle l', g', o\rangle : \langle l, g, o\rangle \in F, \langle l, g\rangle \tau \langle l', g'\rangle\} \cup$$

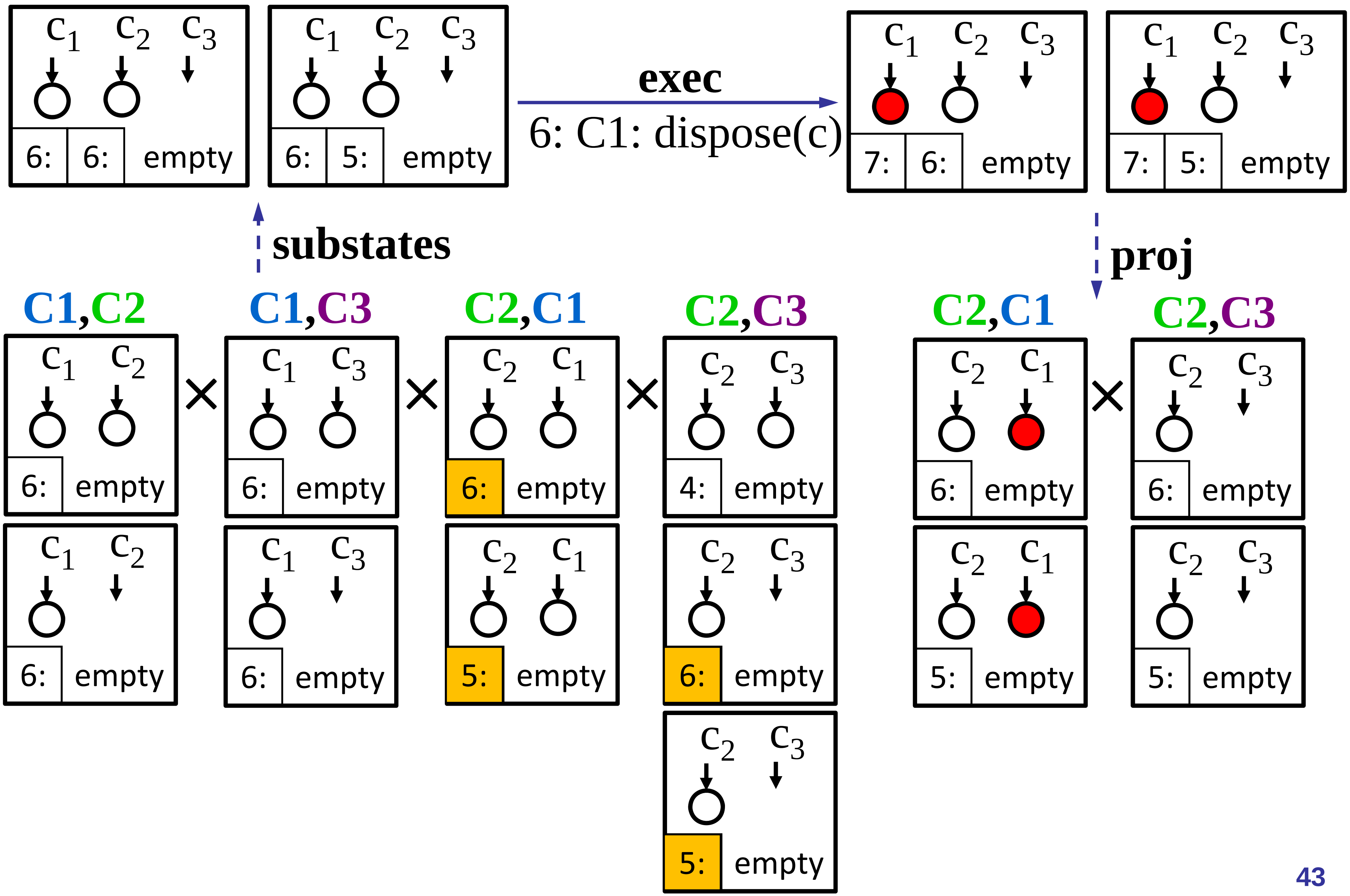
$$\left\{ \begin{array}{l} \langle l_2, g', o\rangle, \langle l_2, g', \alpha(l_1')\rangle : f_1, f_2, f_3, f_4 \in F: \\ \qquad \langle l_1, g, l_2, o\rangle \in substates(f_1, f_2, f_3, f_4), \\ \qquad \langle l_1, g\rangle \tau \langle l_1', g'\rangle \end{array} \right\}$$

- Exploit redundancies in the action
  - Cannot affect locals of other threads
  - Use asymmetry between the two abstractions
  - Can prove no loss of information
  - Summarizing Effects
- Apply aggressive abstraction to the executing threads
  - Potential loss of precision
  - Summarizing Abstraction

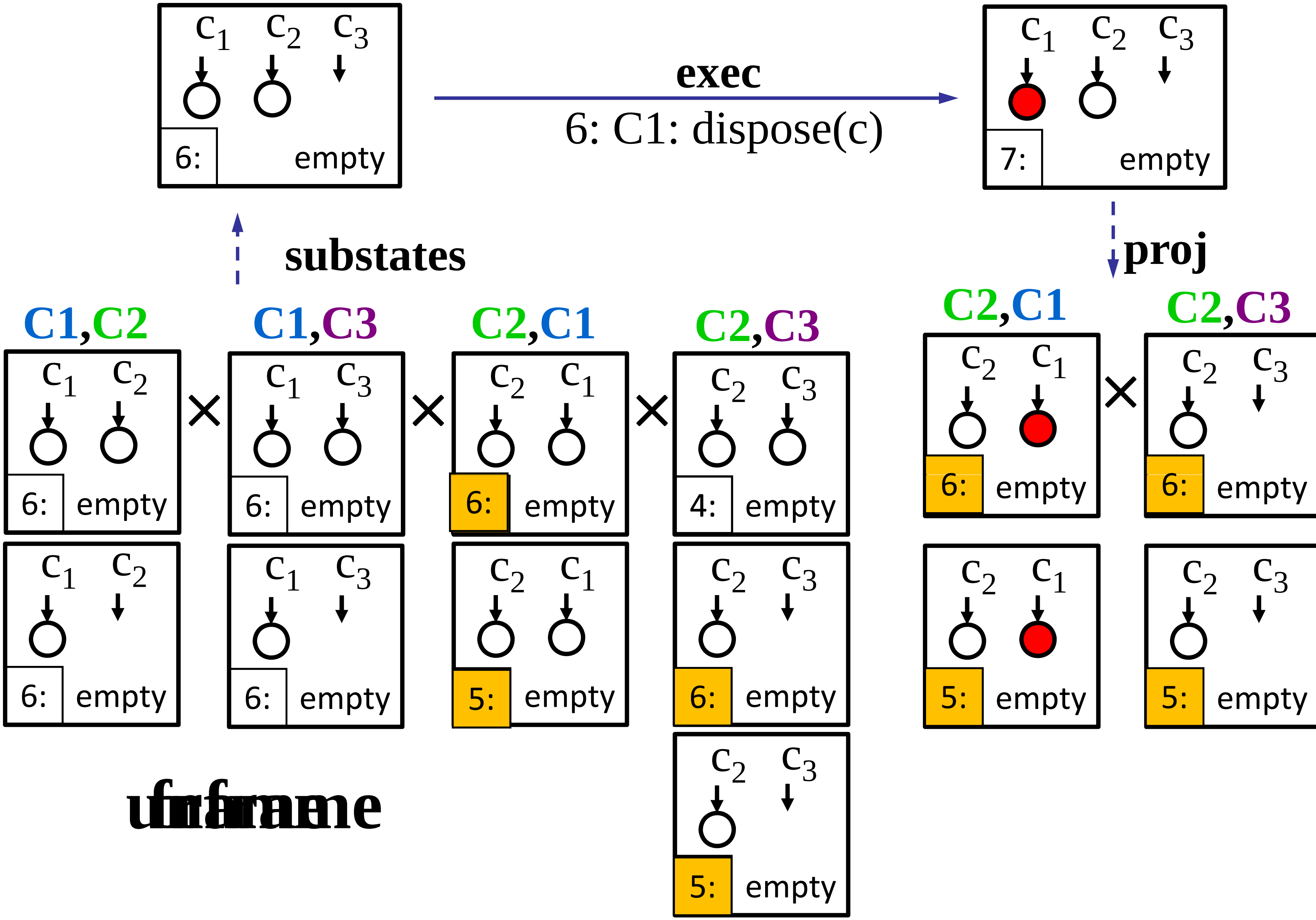
Exploiting Redundancies 6: C1: dispose(c)



Exploiting Redundancies 6: C1: dispose(c)



Exploiting Redundancies 6: C1: dispose(c)



ufrframe

Summarizing Abstraction

$$TR(F) = \{\langle l', g', o\rangle : \langle l, g, o\rangle \in F, \langle l, g\rangle\ \tau\ \langle l', g'\rangle\} \cup$$

$$\left\{ \begin{array}{l} \langle l_2, g', o\rangle, \langle l_2, g', \alpha(l_1')\rangle : f_1, f_2, f_3, f_4 \in F: \\ \qquad \langle l_1, g, l_2, o\rangle \in substates(f_1, f_2, f_3, f_4), \\ \qquad \langle l_1, g\rangle\ \tau\ \langle l_1', g'\rangle \end{array} \right\}$$

- Summarizing Effects reduces the tracked thread's number of states
- Summarizing Abstraction reduces state of executing thread
  - Our heuristic – keep only information accessed by statement
- Significant reduction in size of partial concretization
  - Especially in heap-manipulating programs
  - Precise enough in our benchmarks

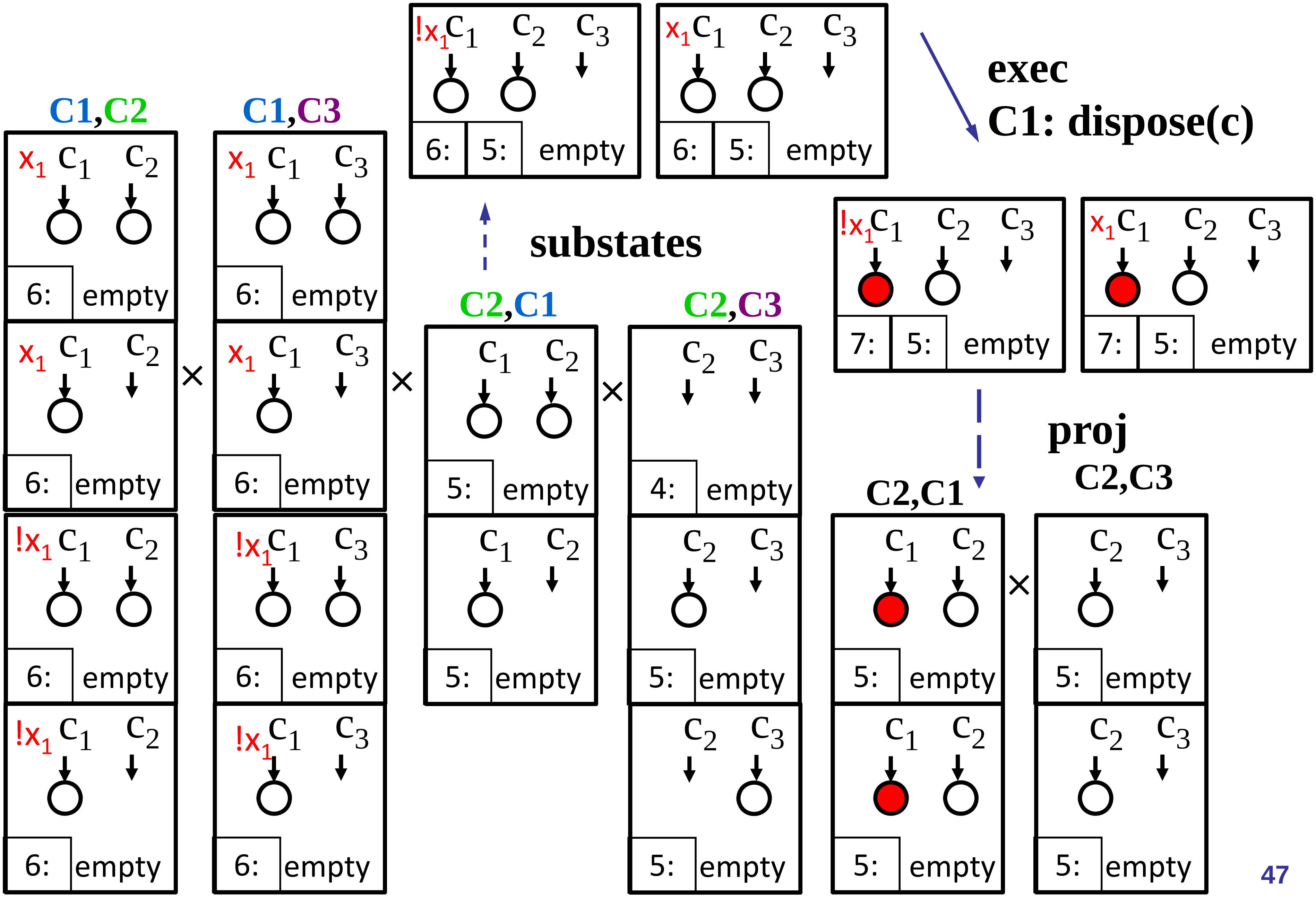
A Singleton Buffer - Modified

```
Boolean empty = true;
Object b = null;
```

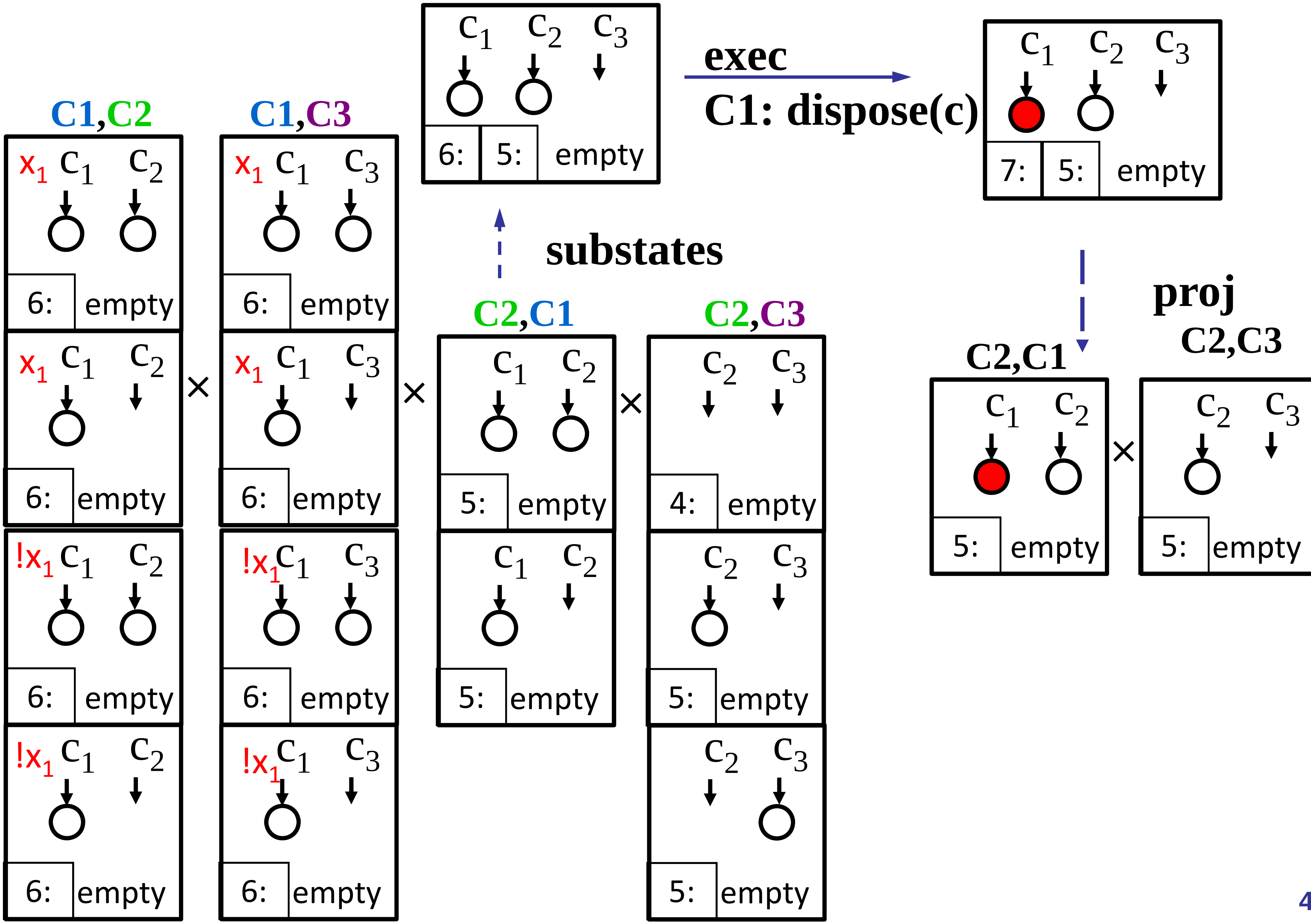
```
produce() {
1: Object p = new();
2: await (empty) then {
        b = p;
        empty = false;
    }
3:
}
```

```
consume() {
Object c;
Boolean x;
4: await (!empty) then {
        c = b;
        empty = true;
    }
5: x = f(c);
6: dispose(c);
7: use(x);
8:
}
```

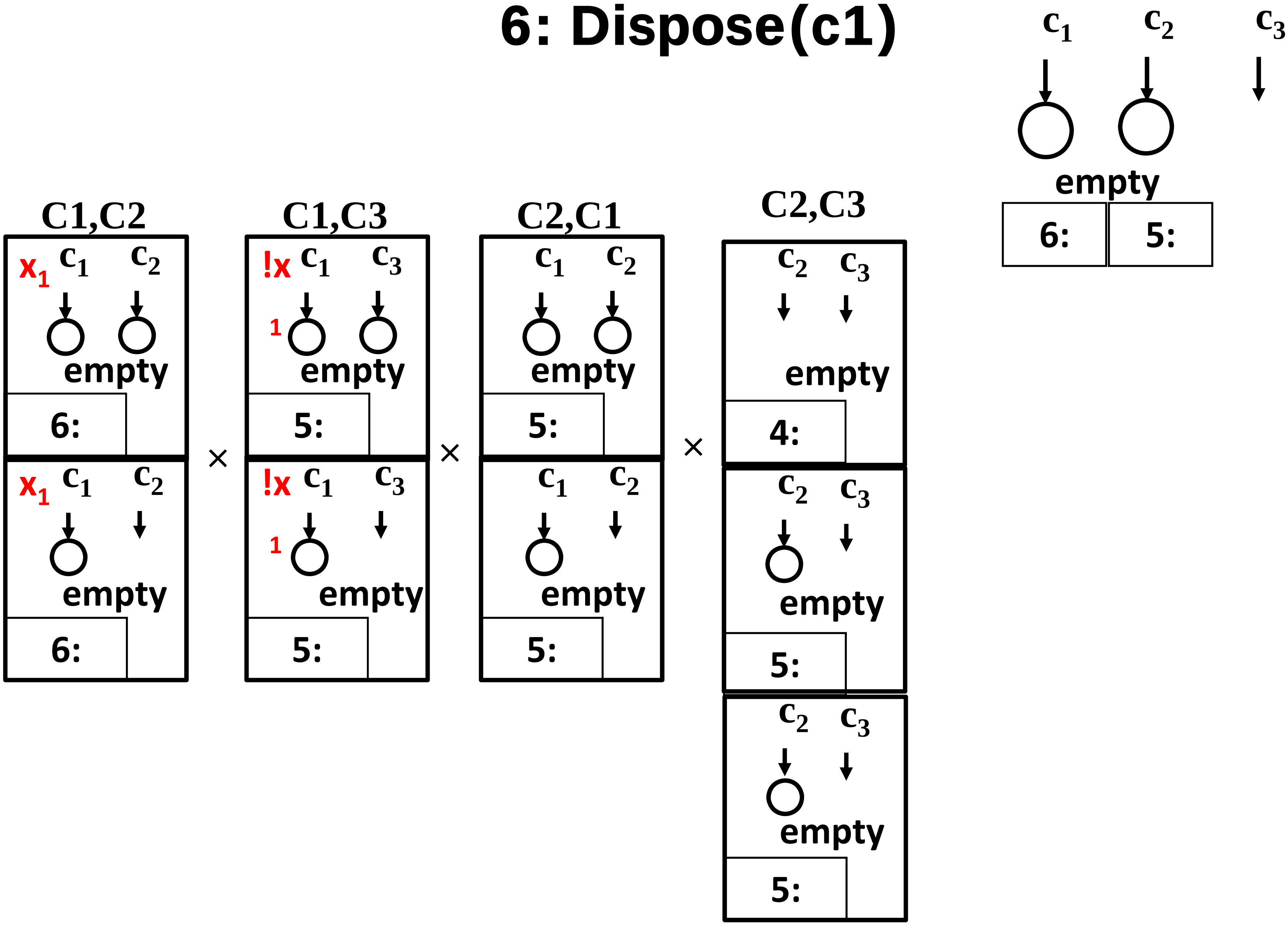
Example 6: C1: dispose(c)



Example 6: C1: dispose(c)



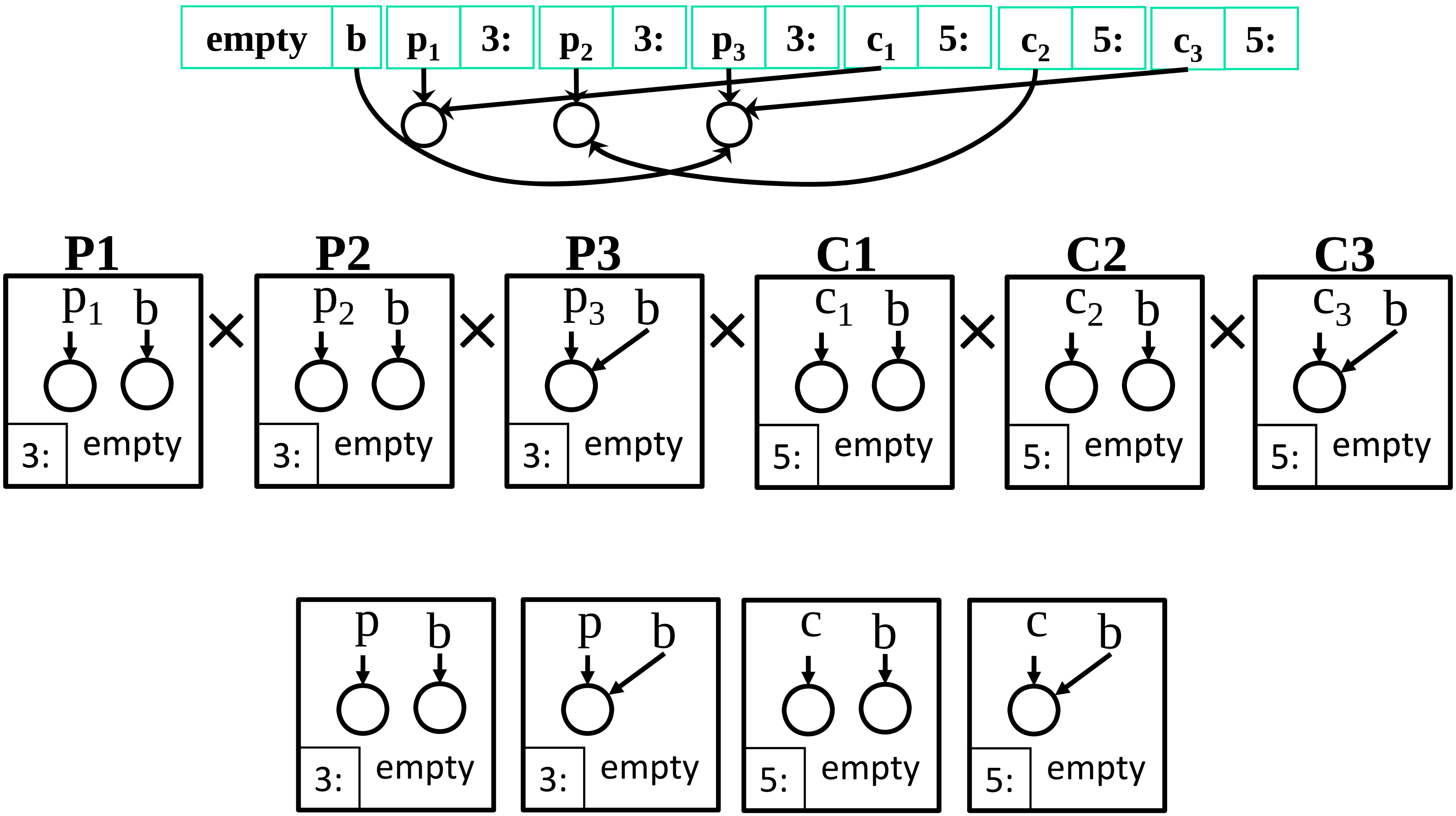
Loss of Precision in Summarizing Abstractions
6: Dispose(c1)



Unbounded Number of Threads

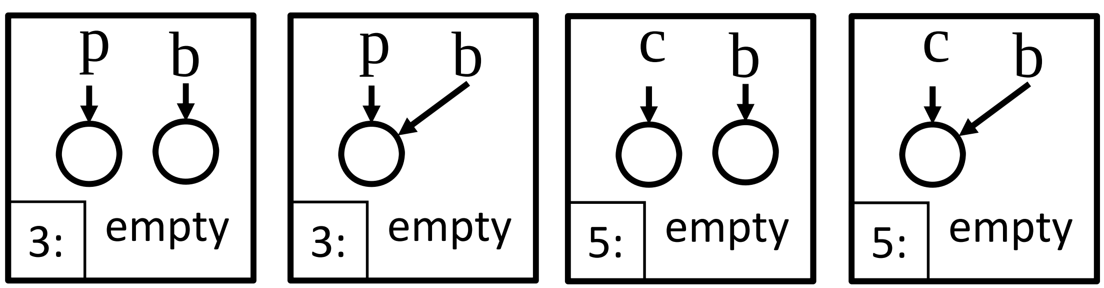
- Abstract thread identifiers
- Usually no extra loss of precision
- Universally quantity over threads

Thread-Modular Abstraction for Unbounded Number of threads

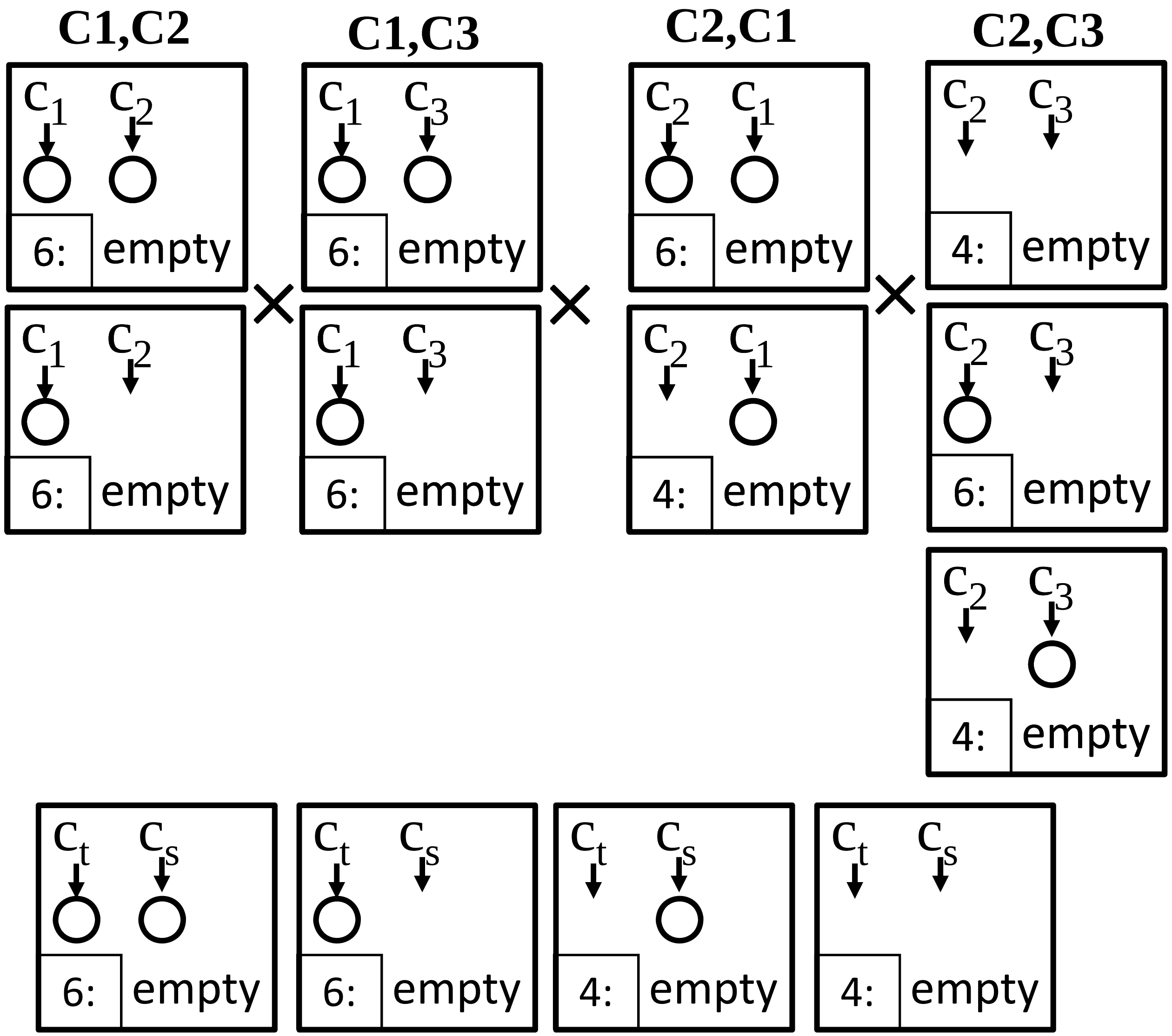


Thread-Modular Abstraction for Unbounded Number of threads

$$
\begin{aligned}
&\forall t: \\
&\quad (pc(t)=3 \wedge p(t) \neq b \wedge valid(p(t)) \wedge valid(b) \wedge empty) \\
&\vee \\
&\quad (pc(t)=3 \wedge p(t) = b \wedge valid(p(t)) \wedge valid(b) \wedge empty) \\
&\vee \\
&\quad (pc(t)=5 \wedge c(t) \neq b \wedge valid(c(t)) \wedge valid(b) \wedge empty) \\
&\vee \\
&\quad (pc(t)=5 \wedge c(t) = b \wedge valid(c(t)) \wedge valid(b) \wedge empty)
\end{aligned}
$$



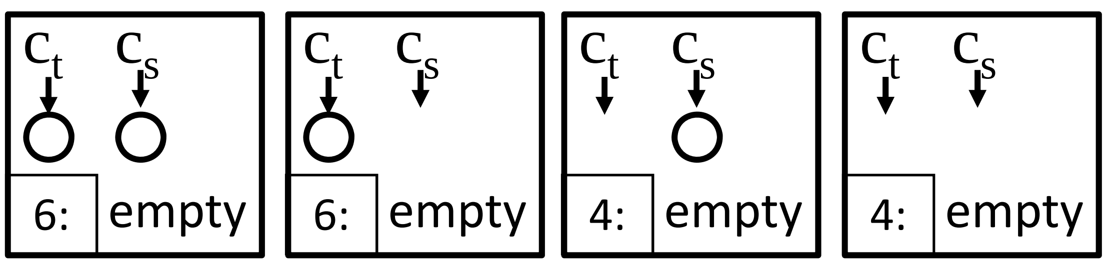
Semi-Thread-Modular Abstraction for Unbounded Number of threads



Semi-Thread-Modular Abstraction for Unbounded Number of threads

$$\forall t, s: s \neq t \Rightarrow$$

$$(pc(t)=6 \wedge c(t) \neq c(s) \wedge valid(c(t)) \wedge valid(c(s)) \wedge empty)$$
$$\vee$$
$$(pc(t)=6 \wedge valid(c(t)) \wedge c(s)=null \wedge empty)$$
$$\vee$$
$$(pc(t)=4 \wedge c(t)=null \wedge valid(c(s)) \wedge empty)$$
$$\vee$$
$$(pc(t)=4 \wedge c(t)=null \wedge c(s)=null \wedge empty)$$



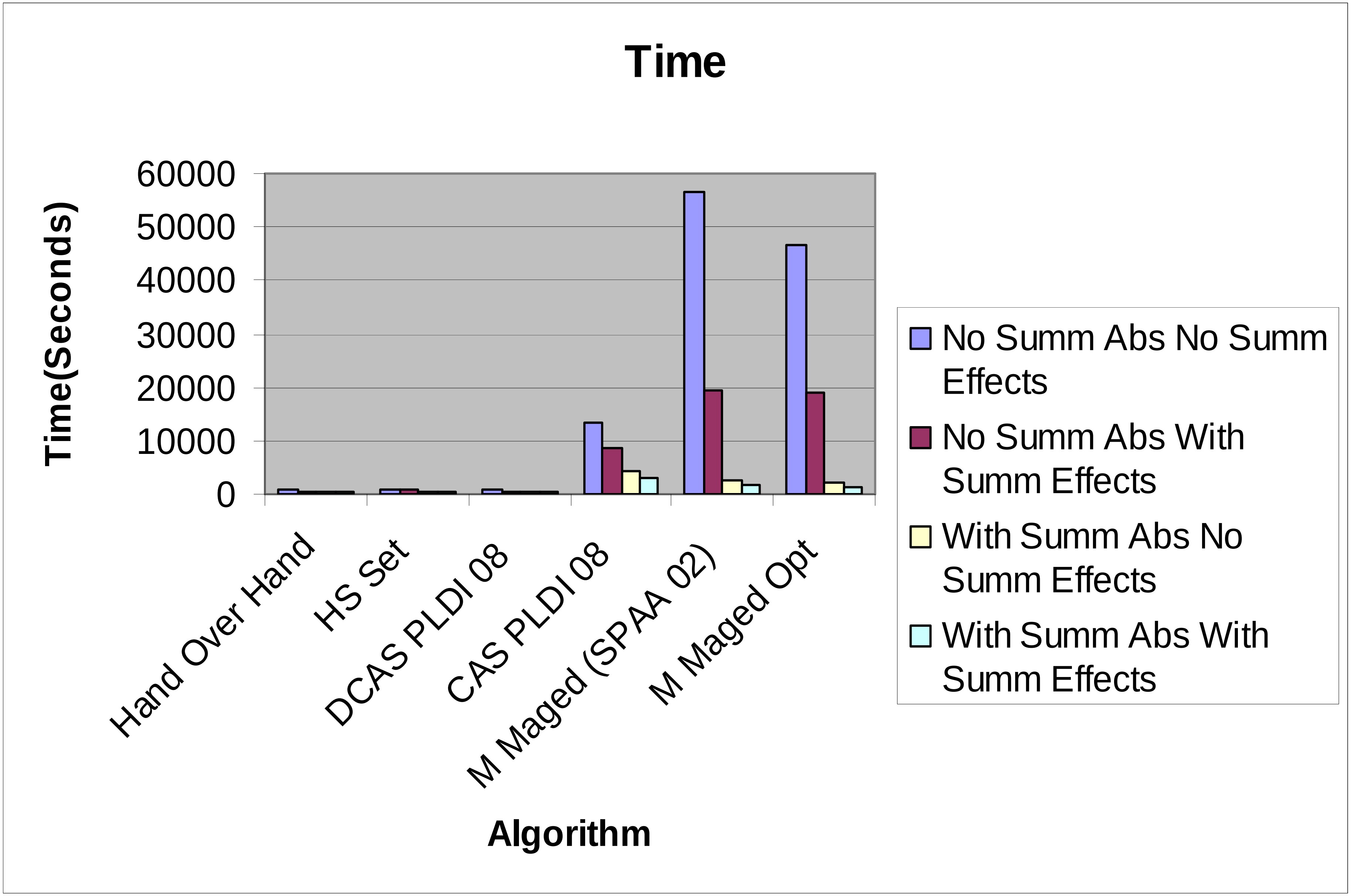
In the TR

- Proofs of soundness
- No loss of precision from summarizing effects
- Combination with heap abstraction
  - Meet is important

Evaluation

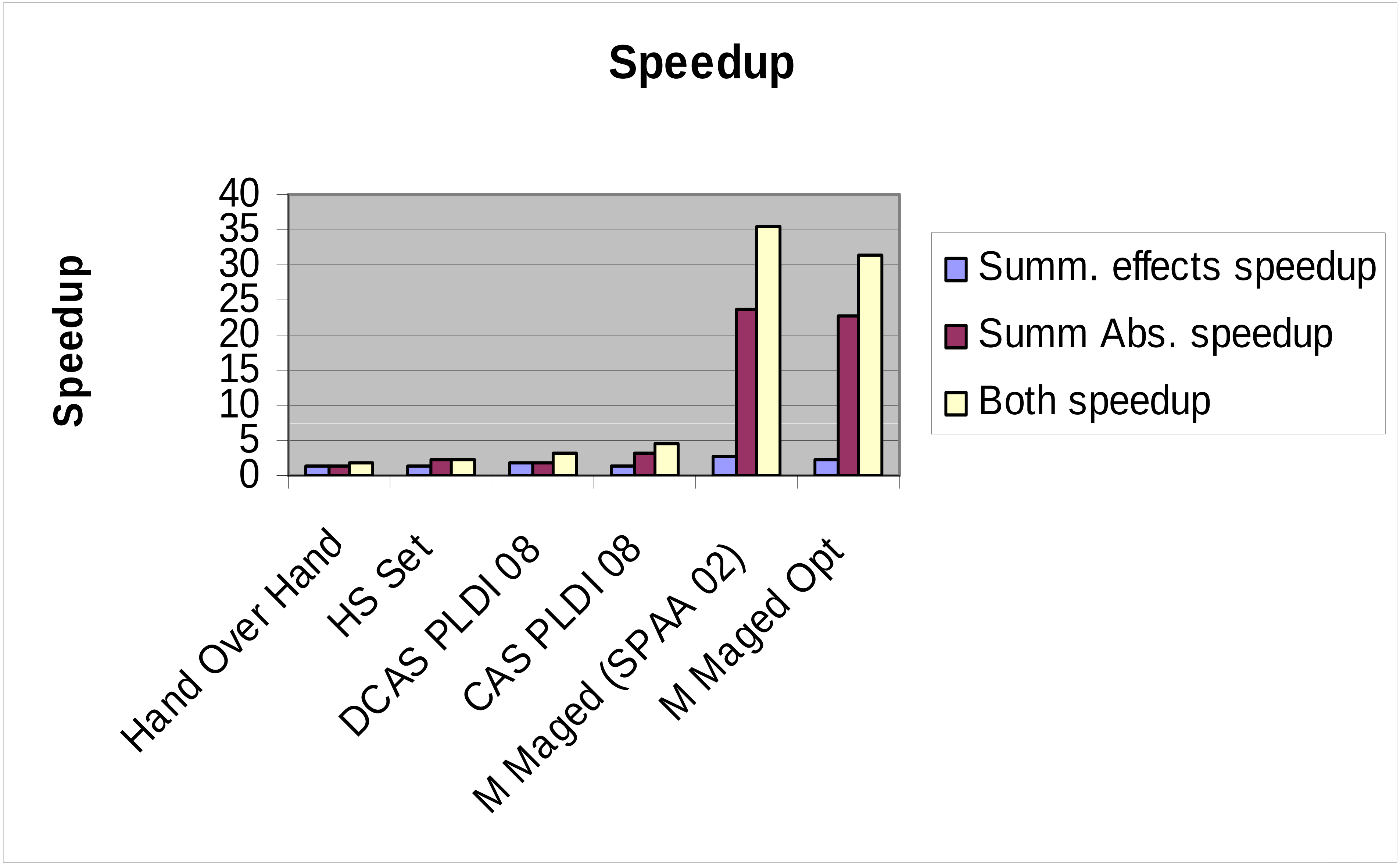
- Implemented (semi-)thread-modular shape analysis using HEDEC/TVLA
  - Unbounded number of threads
  - Unbounded number of objects
  - Call strings for procedures
- Thread-modular unable to prove properties without additional (global) instrumentation
- Semi-thread-modular analysis proves required properties
- Reproduce the injected errors

Evaluation



Time
60000
50000
40000
30000
20000
10000
0
Time(Seconds)
Hand Over Hand
HS Set
DCAS PLDI 08
CAS PLDI 08
M Maged (SPAA 02)
M Maged Opt
Algorithm
No Summ Abs No Summ Effects
No Summ Abs With Summ Effects
With Summ Abs No Summ Effects
With Summ Abs With Summ Effects

Evaluation


Speedup
Speedup
40
35
30
25
20
15
10
5
0
Hand Over Hand
HS Set
DCAS PLDI 08
CAS PLDI 08
M Maged (SPAA 02)
M Maged Opt
Summ. effects speedup
Summ Abs. speedup
Both speedup

Related Work

- Process centric abstractions
  - [C. A. R. Hoare ’72] [Owicki & Gries ’76] [E. Clarke TOPLAS’80] [Talupur et al. VMCAI’06] [Flanagan & Qadeer, SPIN’03] many more...
  - [Malkis, Podelski, Rybalchenco, SAS’07]
- Thread-modular shape analysis
  - [Gotsman et al. PLDI’07]
  - [Manevich et al. SAS’08]
  - [Calcagno et al. SAS’07]
  - R-G reasoning [Vafeiadis et al. ’06-’09]

Summary

- A new abstraction for concurrent systems
  - Scalable in the number of threads
  - Handles unbounded number of threads
  - Semi-thread-modular program analysis
- Provably sound analysis
- Potential loss of precision
  - Abstraction
  - Transformers
  - But precise enough – 0 false alarms
- Reducing quadratic factors