Inferring Nondeterministic Sequential Specifications for Parallelism Correctness

Jacob Burnim, **Tayfun Elmas***
George Necula, Koushik Sen
University of California, Berkeley

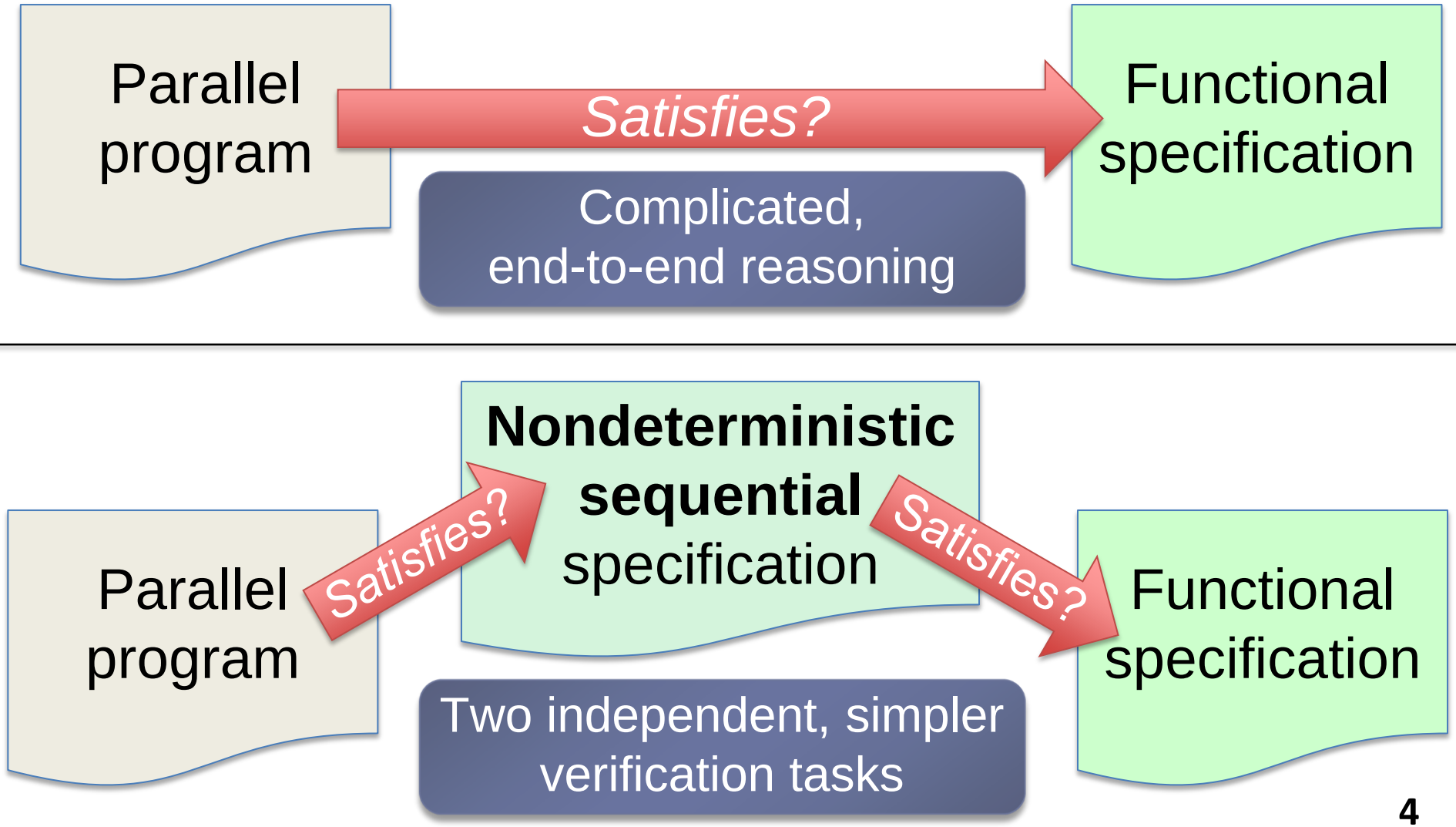
Correctness of parallel programs

```
parallel_integer_matrix_multiply(A, B) {

    // compute C = A * B  using your
    // favorite parallel algorithm

    return C
}
```

Functional specification: $\forall i, j.\ \mathtt{C}_{i,j} = \sum_k \left(\mathtt{A}_{i,k} * \mathtt{B}_{k,j}\right)$

Checking correctness of parallel programs

Our proposal: Separate reasoning about functional correctness and thread schedules



Our proposal: Separate reasoning about functional correct

$$\forall_{0 \le x < width} \cdot \forall_{0 \le y < height} \cdot \left(\left| f_{iter}^{\,maxiter}(0) \right| < 2 \Rightarrow img[x][y] = 0 \right) \wedge \left(\exists_{1 \le i < maxiter} \cdot \left| f_{iter}^{\,i}(0) \right| \ge 2 \wedge \forall_{1 \le j < i} \cdot \left| f_{iter}^{\,j}(0) \right| < 2 \Rightarrow img[x][y] = \mathrm{HSB}\left((i/maxiter)^g, 1, 1 \right) \right)$$

where $f_{iter}(c) = c^2 + \left(xcenter + (xoff + x)/res \right) + i\left(ycenter + (yoff - y)/res \right)$

Parallelism correctness.

Parallel fractal generation

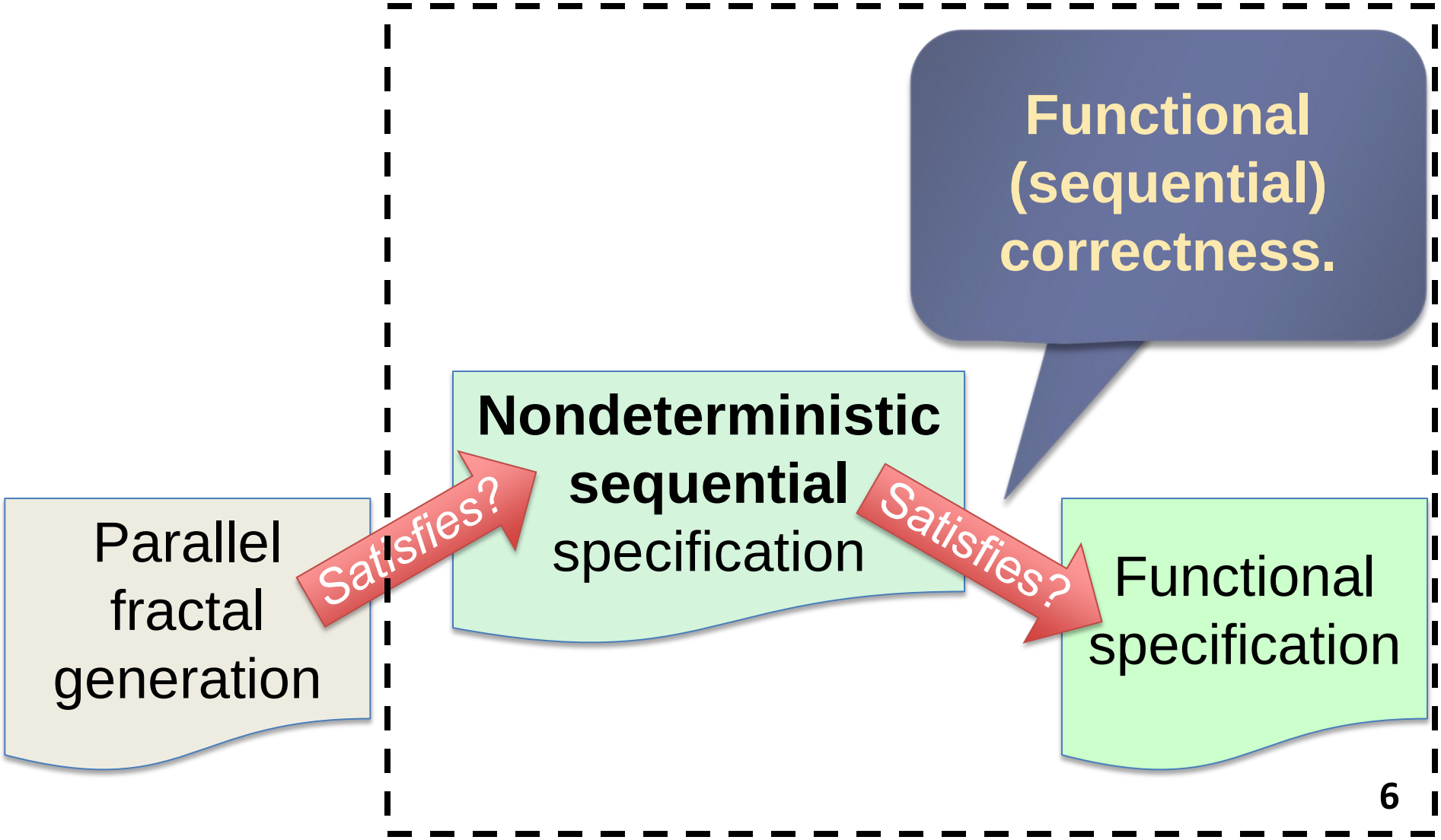
Satisfies?

Nondeterministic sequential specification

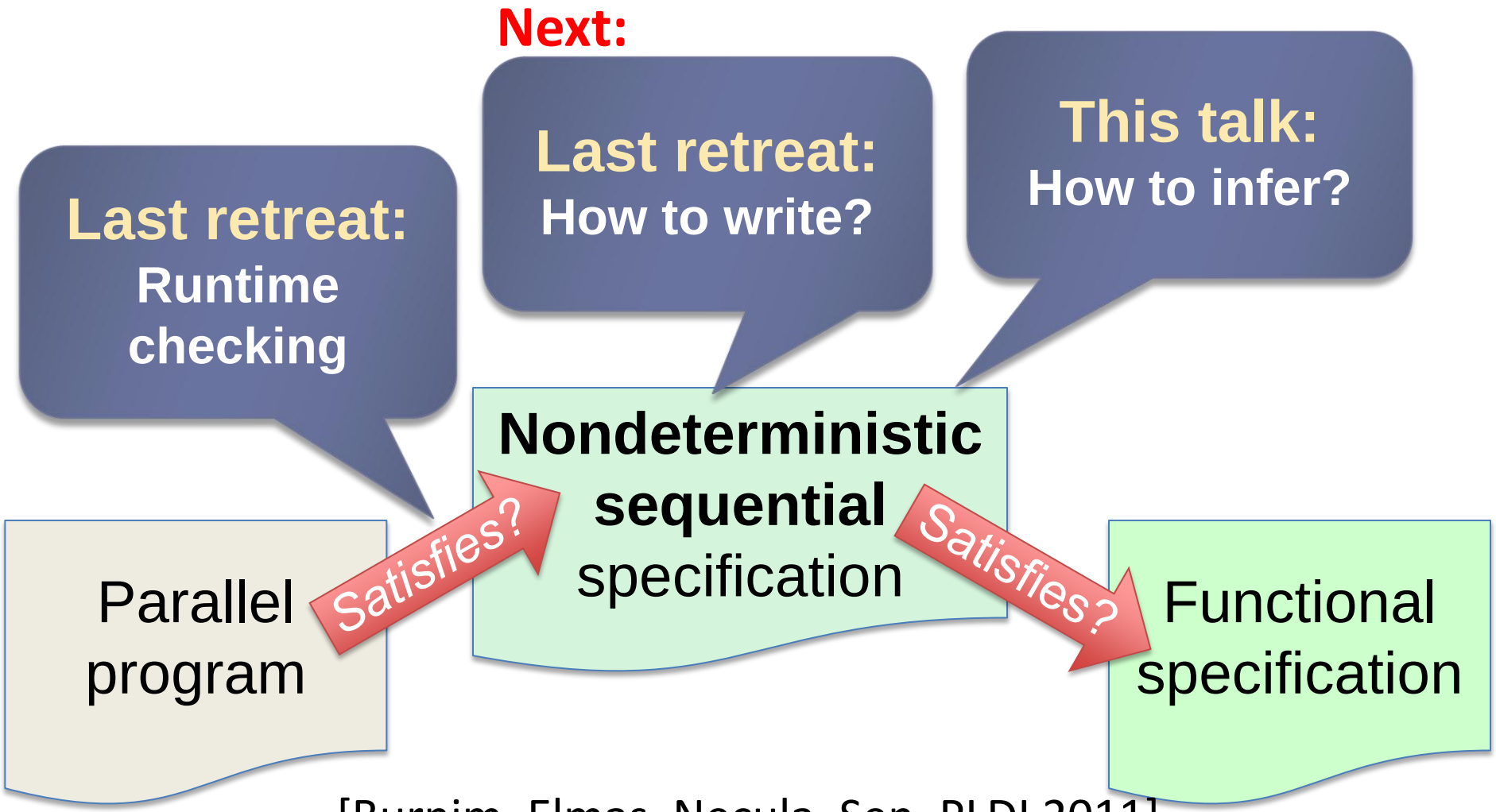
Satisfies?

Functional specification

Our proposal: Separate reasoning about functional correctness and thread schedules



Outline

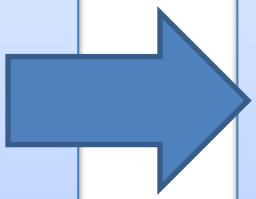


[Burnim, Elmas, Necula, Sen, PLDI 2011]

NDSeq - 101

Parallel program

```
.....
parallel-for(...) {
    .....
  S
    .....
}
.....
```



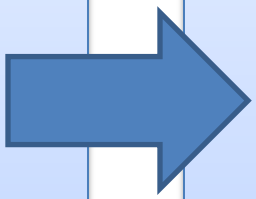
NDSeq spec

```
.....
nd-for(...) {
    .....
  if(*) S
    .....
}
.....
```

Example 1: Parallel reduction

Parallel program

```
parallel-for(i = 1 to N){
  while(true){
    t1 = x
    t2 = (t1 + i)^i

    if(CAS(x, t1, t2))
       end_iteration
    print “retrying”
  }
}
print x
```



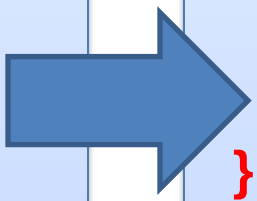
NDSeq spec

```
nd-for(i = 1 to N){
  while(true){
    t1 = x
    t2 = (t1 + i)^i
    if(*)
      if(CAS(x, t1, t2))
         end_iteration
    print “retrying”
  }
}
print x
```

Example 2: Branch and bound

Parallel program

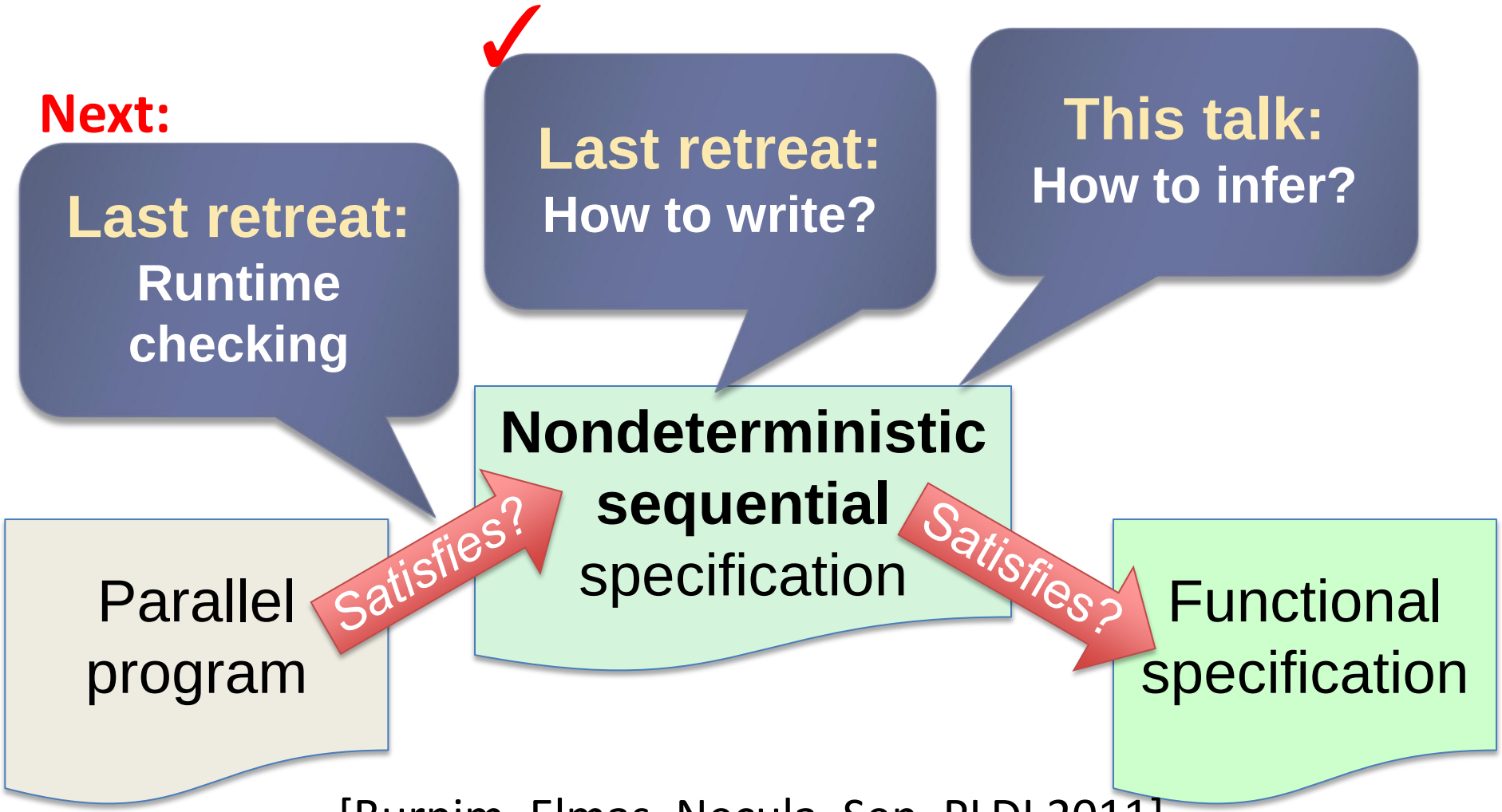
```
parallel-for(i = 1 to N){

  b = lower_bound(i)
  if(b >= lowest_cost)
     end_iteration

  c = compute_cost(i)
  synchronized_by(lock){
    if(c < lowest_cost){
       lowest_cost = c
      best_soln = s
  } }
}
print best_soln
```



NDSeq spec

```
nd-for(i = 1 to N){
  if(*){
    b = lower_bound(i)
    if(b >= lowest_cost)
       end_iteration
  }
  c = compute_cost(i)
  synchronized_by(lock){
    if(c < lowest_cost){
       lowest_cost = c
      best_soln = s
  } }
}
print best_soln
```

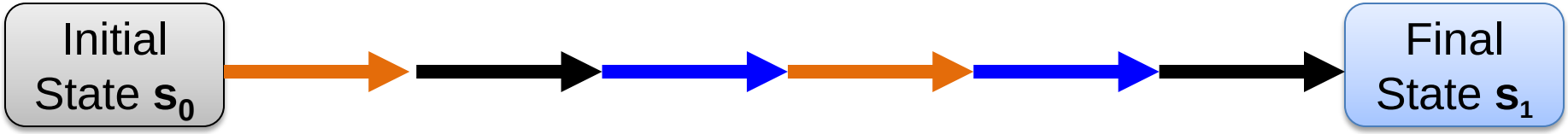
Outline



[Burnim, Elmas, Necula, Sen, PLDI 2011]

Parallelism correctness: Semantic definition

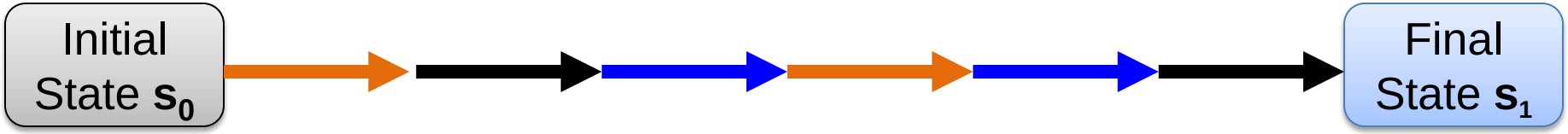
For each parallel execution



there exists an equivalent NDSeq execution

Q: What does checking mean?

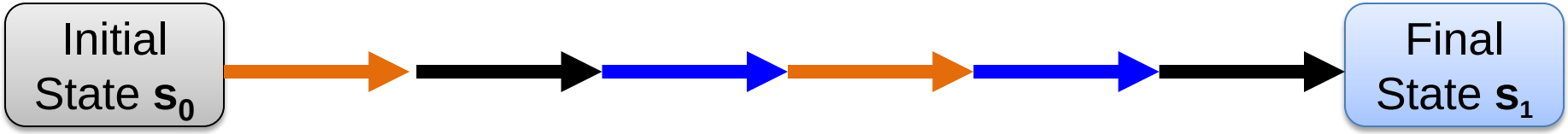
For each parallel execution (generated by testing)



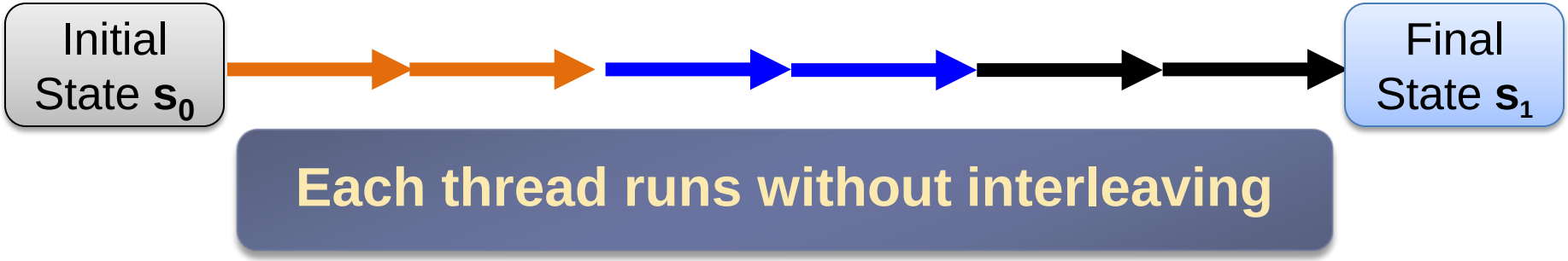
can we show that there exists an equivalent NDSeq execution?

Q: What does checking mean?
A: Serialize all threads in execution

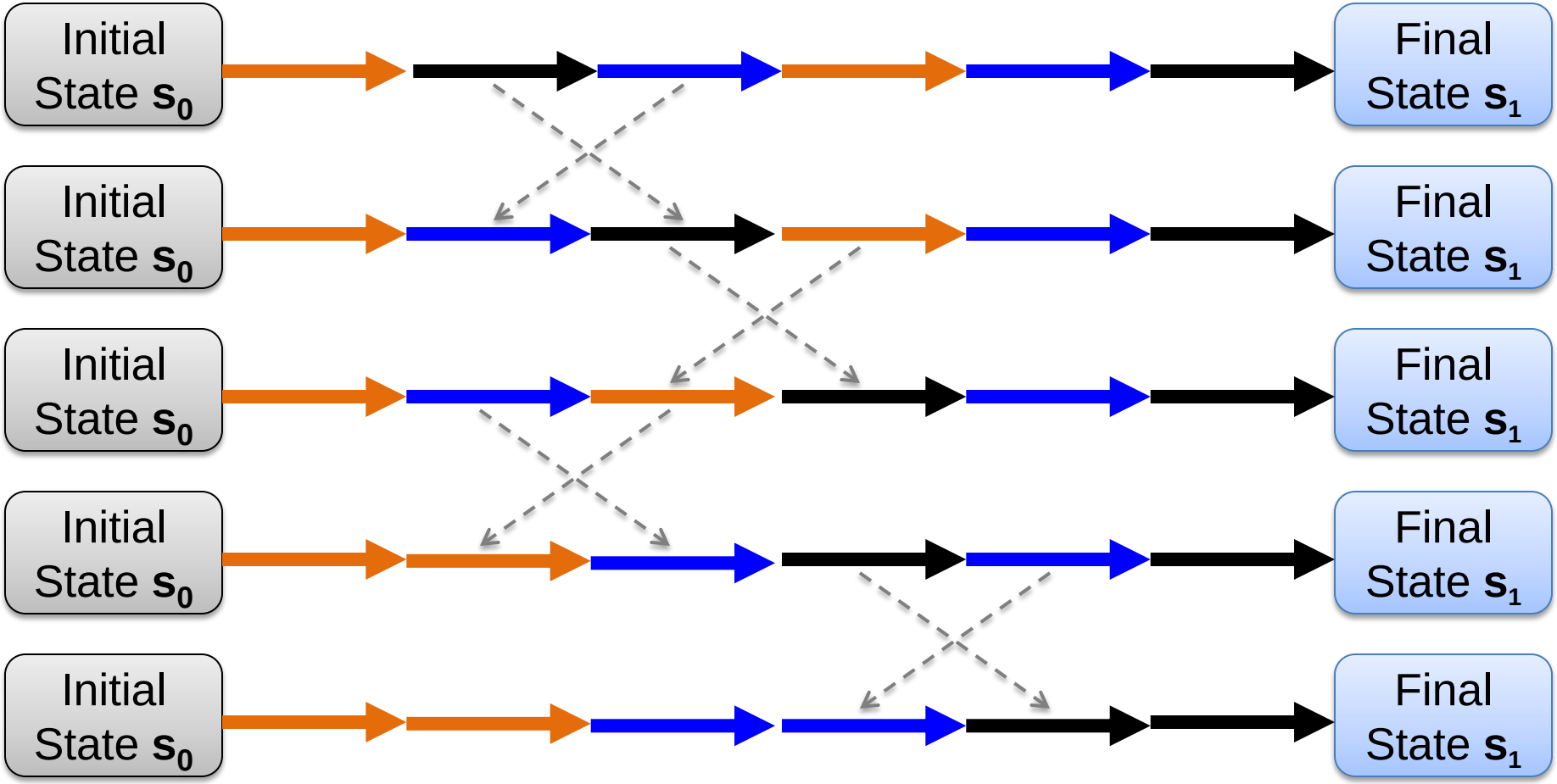
For each parallel execution (generated by testing)



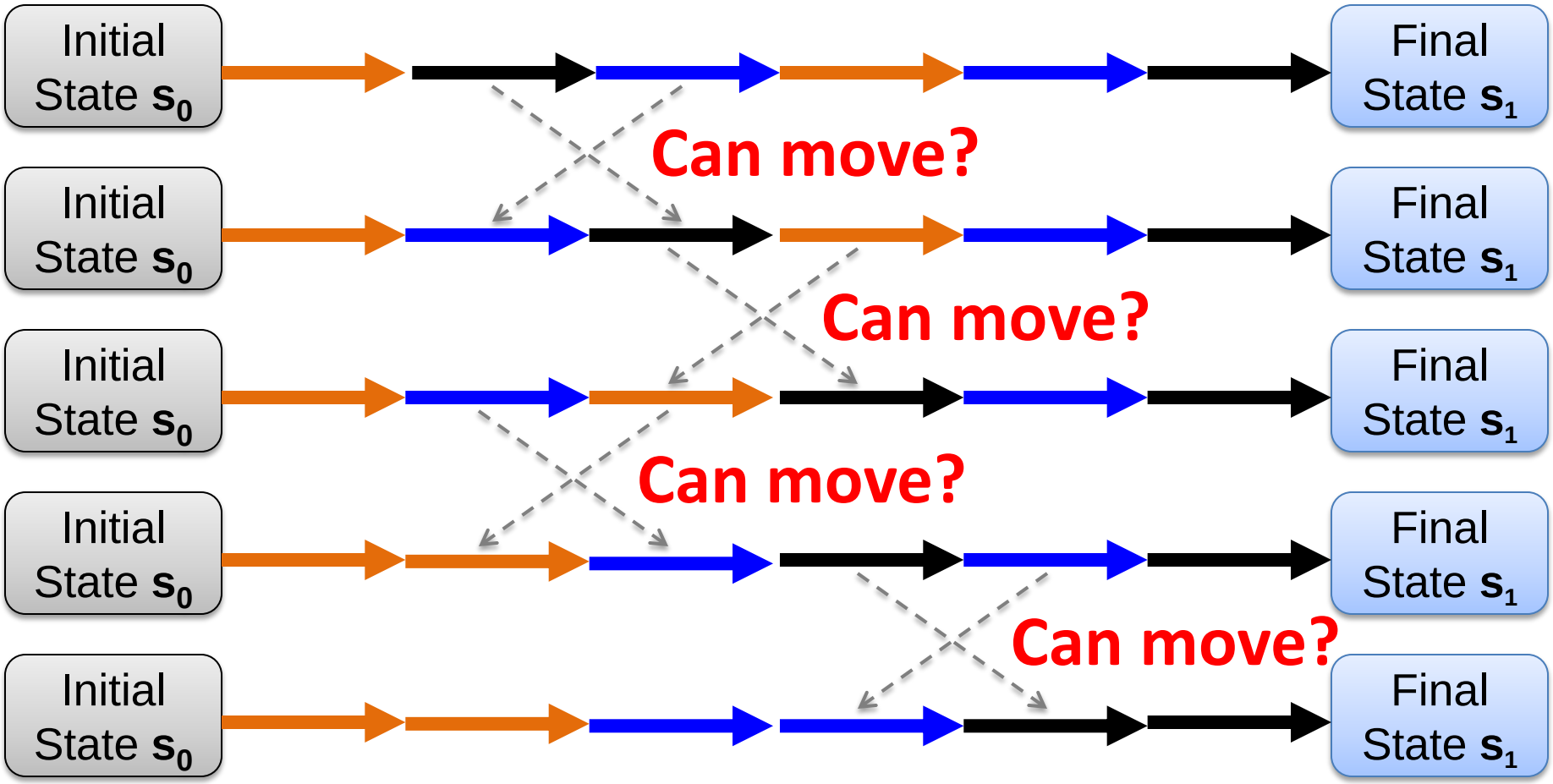
can we show that there exists an equivalent NDSeq execution?



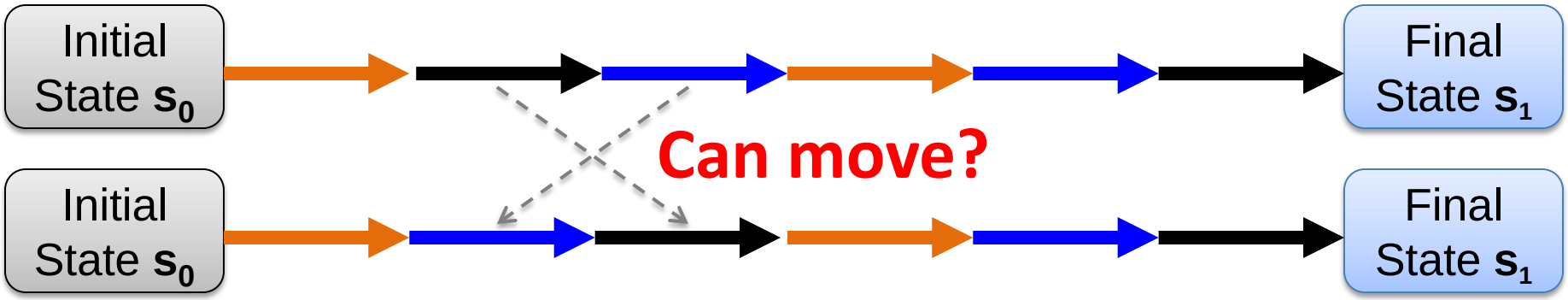
Q: What does checking mean?
A: Serialize all threads in execution



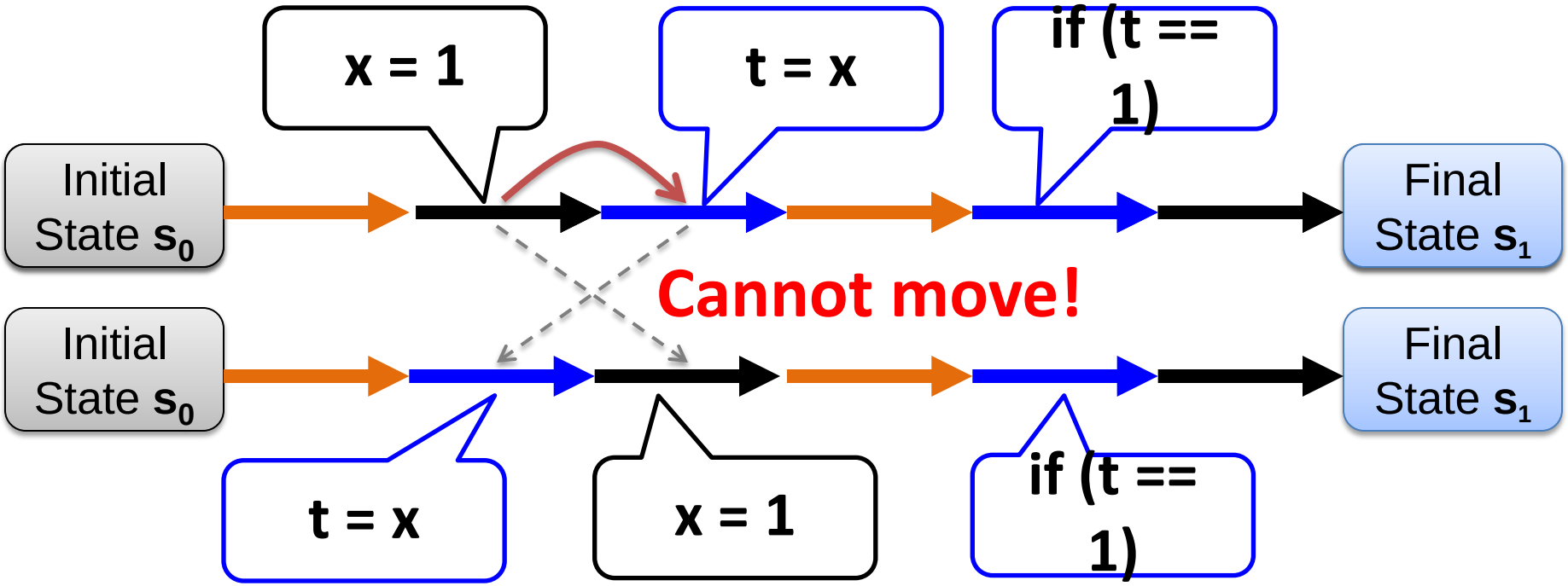
When is it safe to move actions?



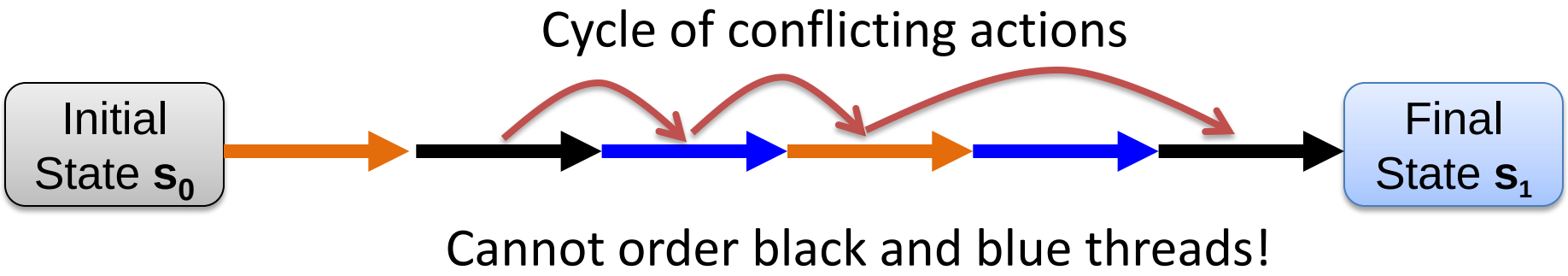
When is it safe to move actions?



Conflicting actions: Cannot move!



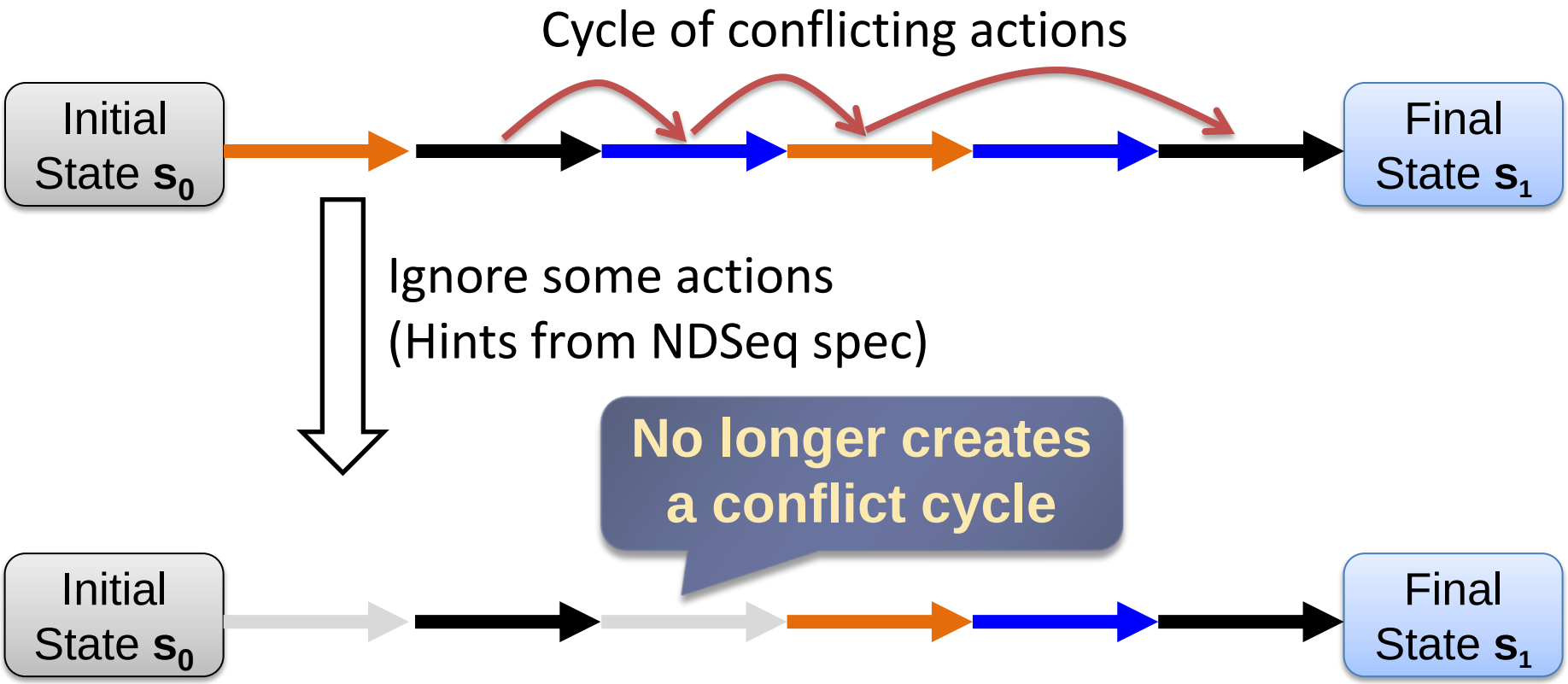
Conflict cycles



Conflict serializability:

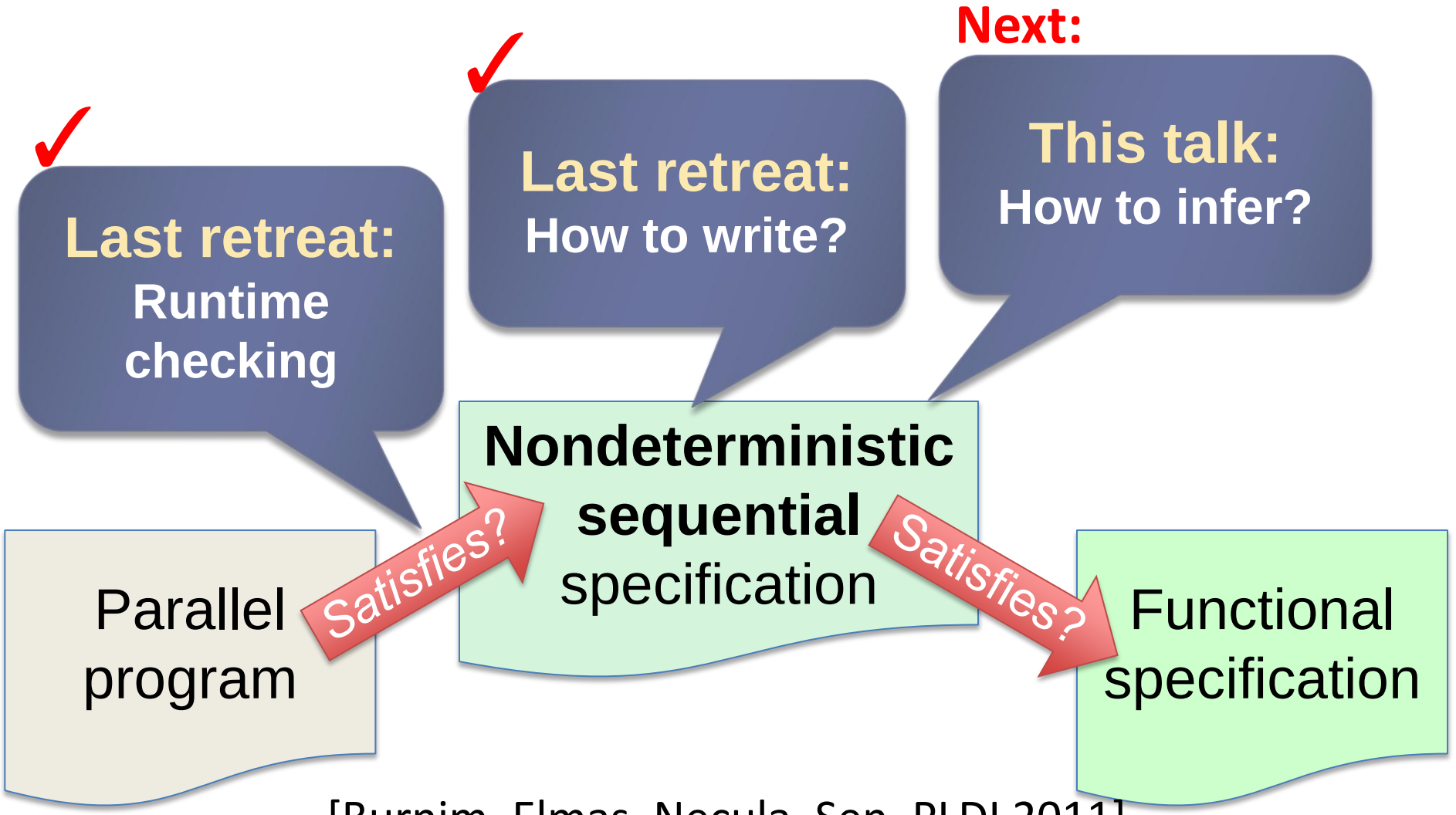
Classic Theorem: If no conflict cycle, then there exists an equivalent NDSeq execution.

Can we ignore some conflict?



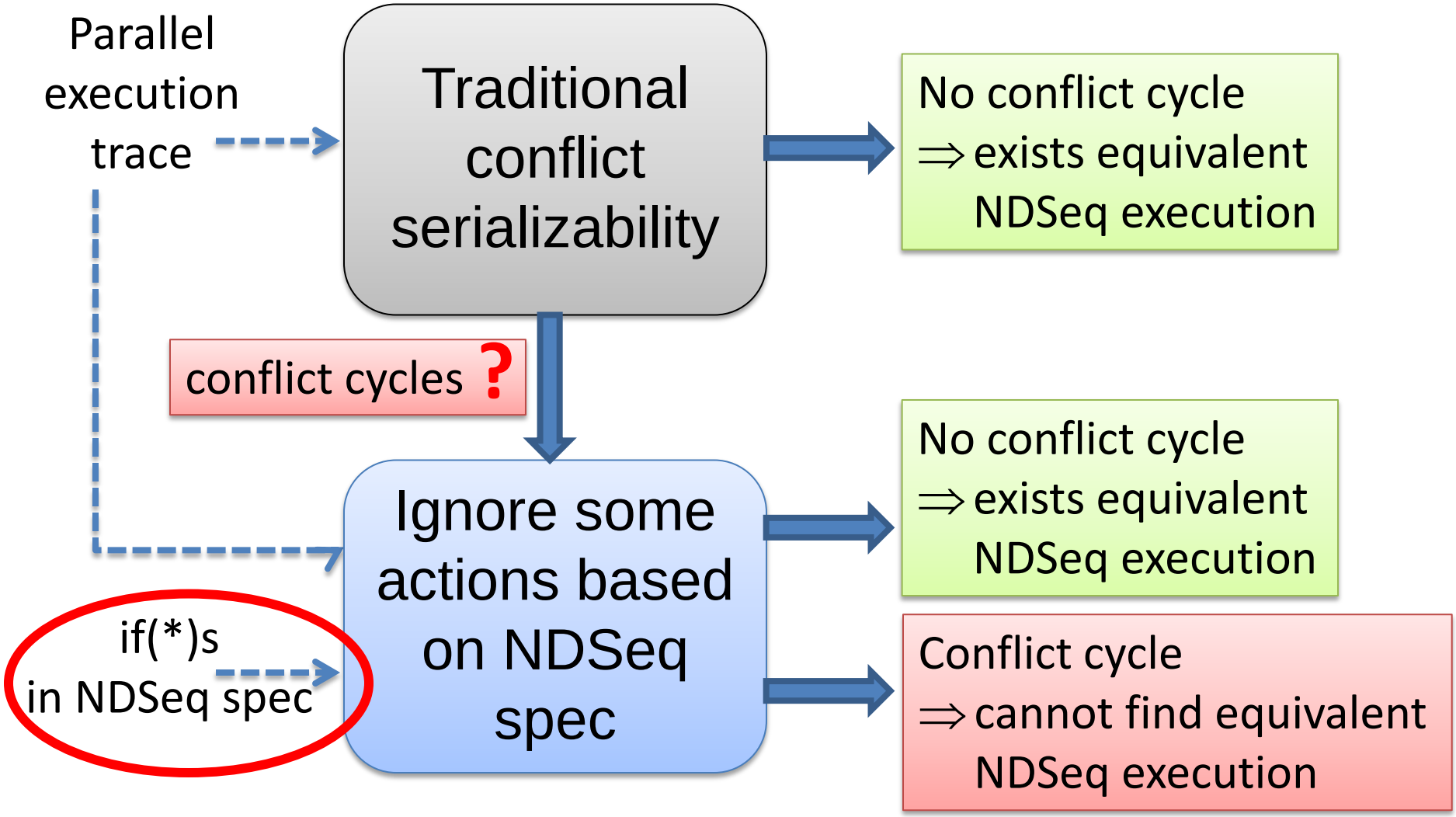
Theorem: If no cycle remains after ignoring actions, then there exists an equivalent NDSeq execution.

Outline

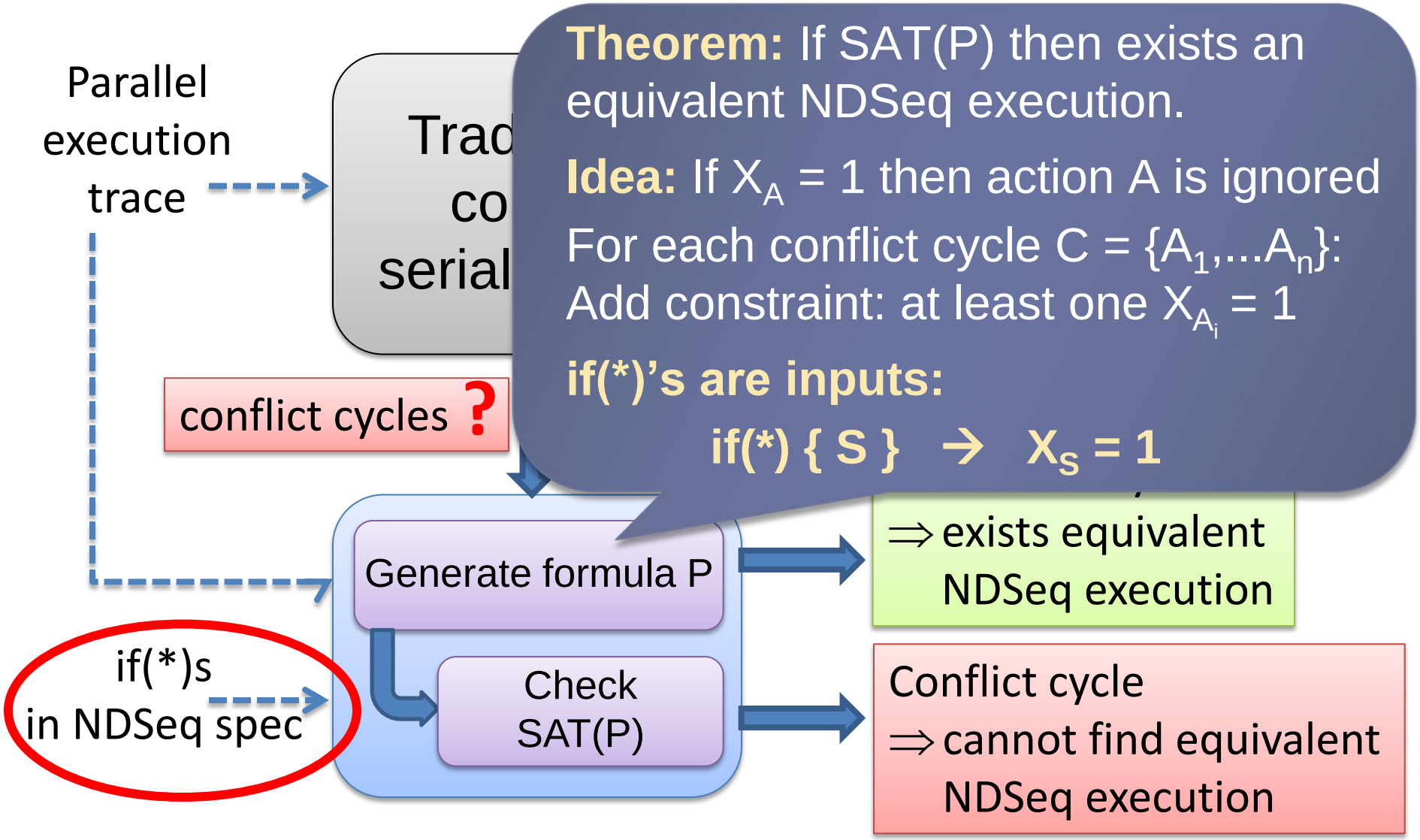


[Burnim, Elmas, Necula, Sen, PLDI 2011]

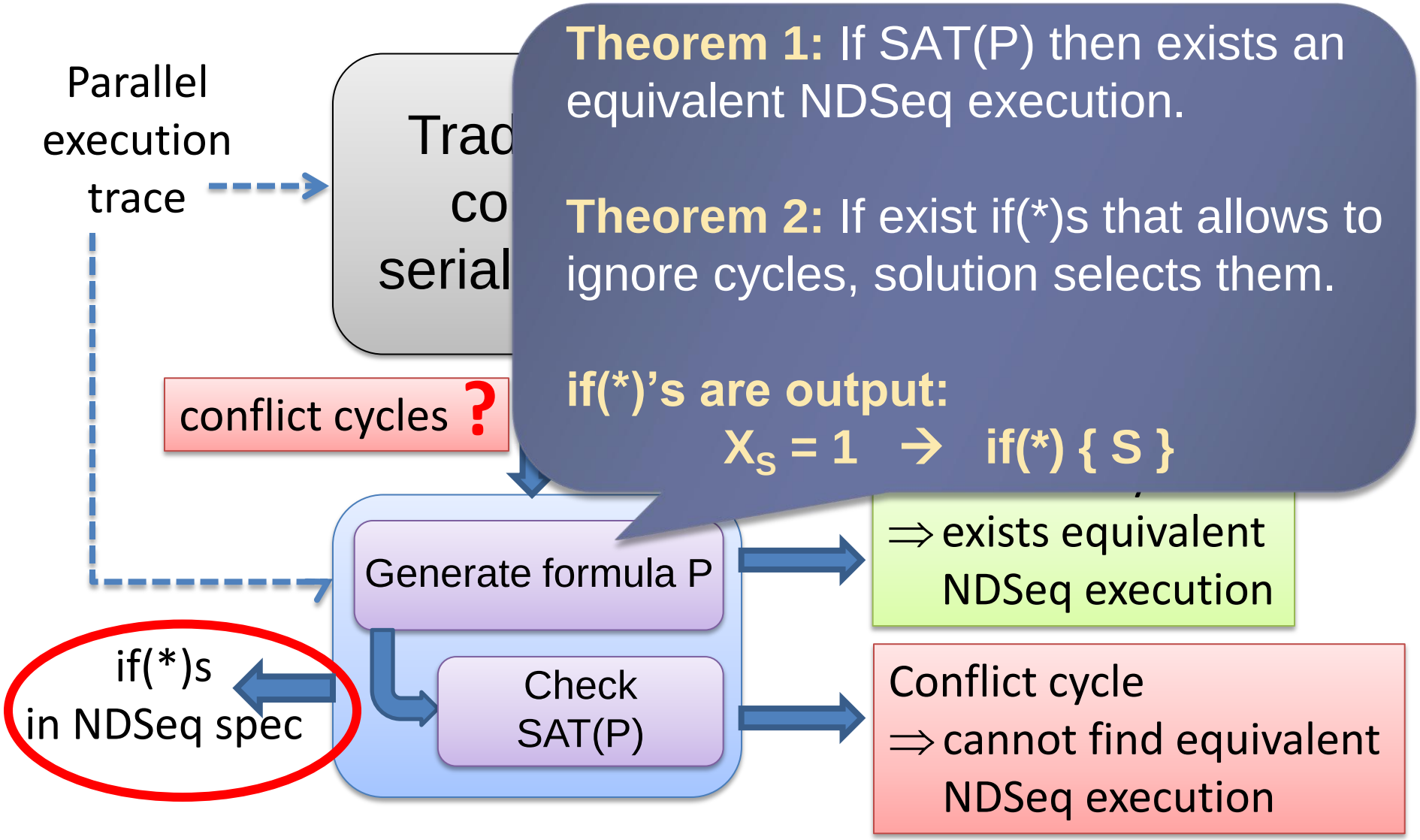
Our approach: Check if there exists an equivalent NDSeq execution



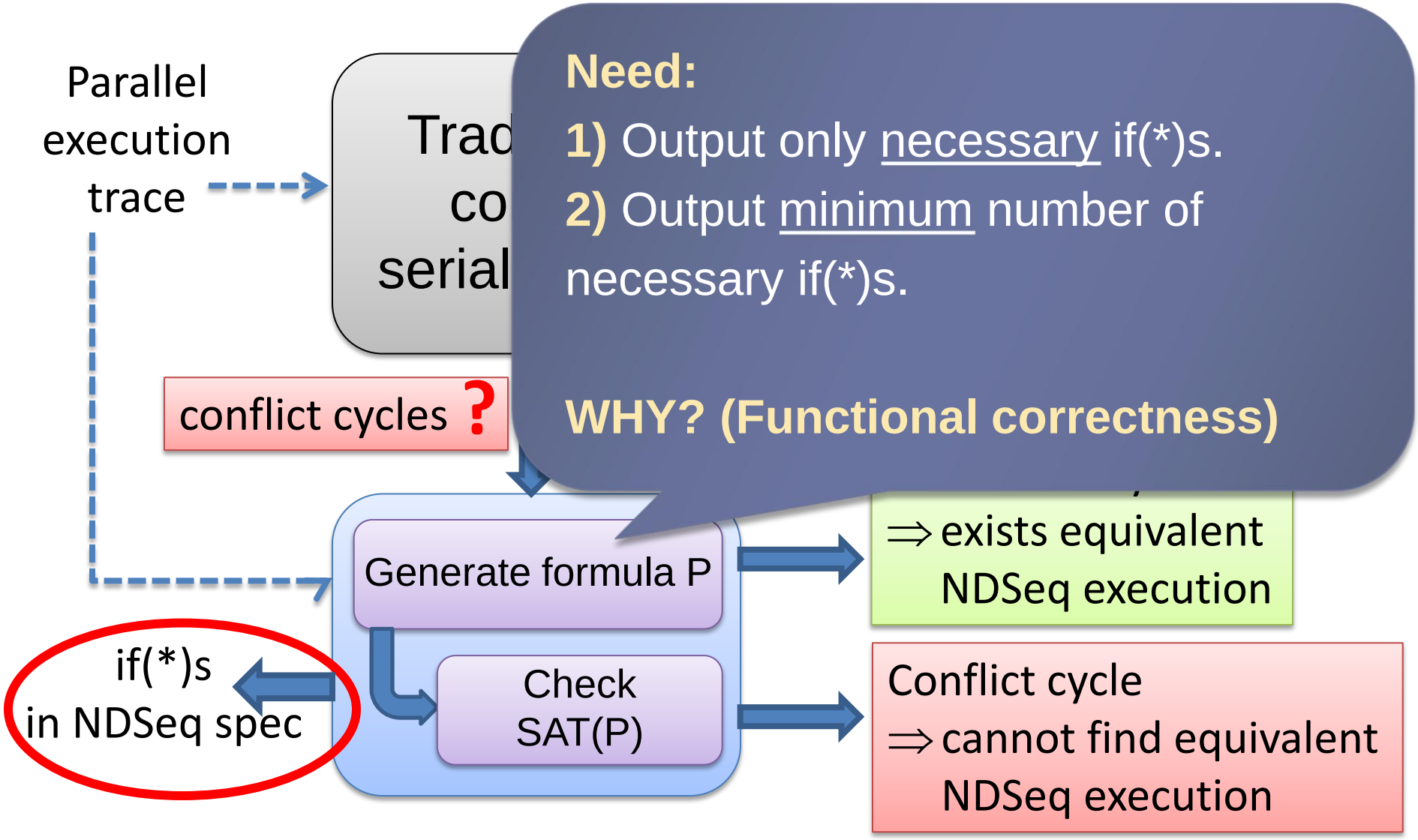
Encoding parallelism correctness as SAT



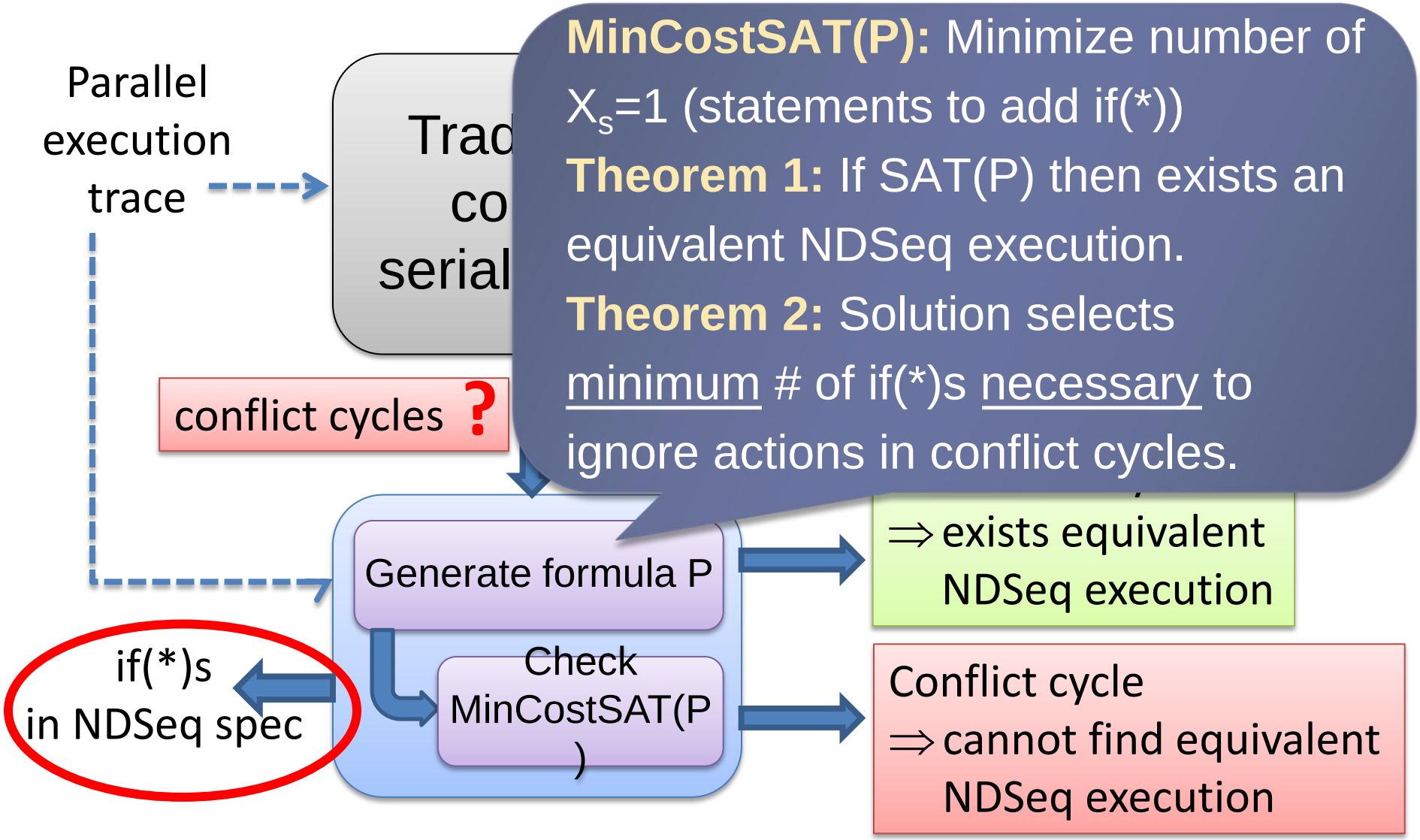
Obtaining if(*)s from solution to SAT



BUT, which if(*)s are necessary?



Minimal NDSeq spec as MinCostSAT



Experimental results

Benchmark	Line of code	Size of trace	Irrelevant events	Number of parallel constructs	Number of if(*)s(manual)	Number of if(*)s (inferred)	Inferred if(*)s right?
phylogeny(fixed)	4.4K	29K	24K	2	3	3	yes
concurrent stack	40	1K	350	1	2	2	yes
concurrent queue	60	320	110	1	2	2	yes
mesh refinement	1K	930K	845K	1	2	2	yes
sor	300	905K	561K	1	0	0	yes
matmult	700	962K	8K	1	0	0	yes
series	800	2008K	1.2K	1	0	0	yes
crypt	1.1K	493K	100K	2	0	0	yes
moldyn	1.3K	4517K	4300K	4	0	0	yes
lufact	1.5K	1048K	792K	1	0	0	yes
raytracer (fixed)	1.9K	9125K	8960K	1	0	0	yes
montecarlo	3.6K	1723K	731K	1	0	0	yes
pi3	150	1062K	141	1	0	0	yes
keysearch3	200	1050K	1049K	2	0	0	yes
mandelbrot	250	576K	330K	1	0	0	yes
raytracer (buggy)	1.9K	9125K	8960K	1	0	UNSAT	-
phylogeny (buggy)	4.4K	29K	24K	2	3	UNSAT	-

Conclusion

- **Problem:** Adding if(*) most difficult part of writing NDSeq specs

- **Solution:** Sound, automated inference for if(*)s; formulated as MinCostSAT
  - Reasonable starting point for writing NDSeq specs

- Promising evidence for fully-automated testing and verification of parallel programs using NDSeq specs