



RICE

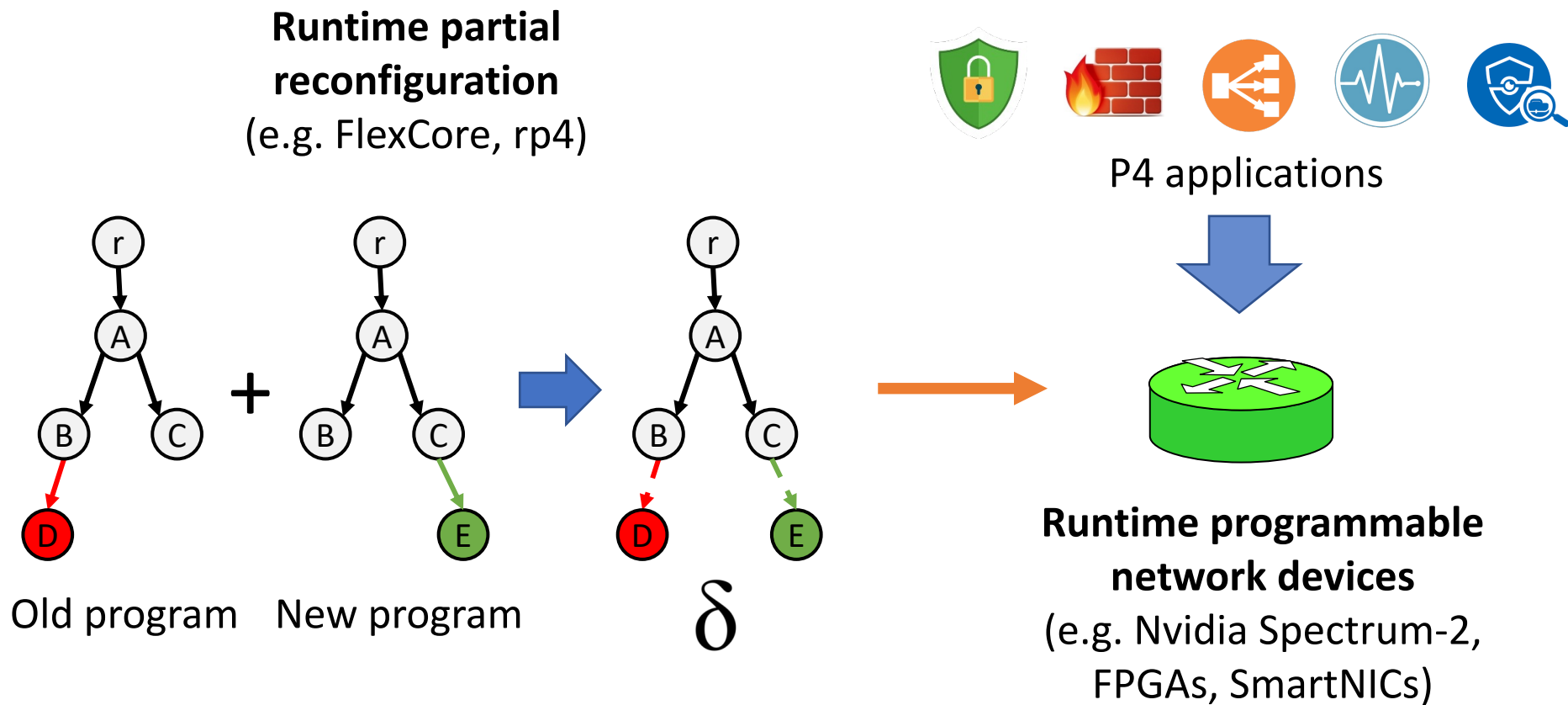


Microsoft

FlexPlan: Synthesizing Runtime Programmable Switch Updates

Yiming Qiu, Ryan Beckett, Ang Chen

Background: Runtime programmable devices

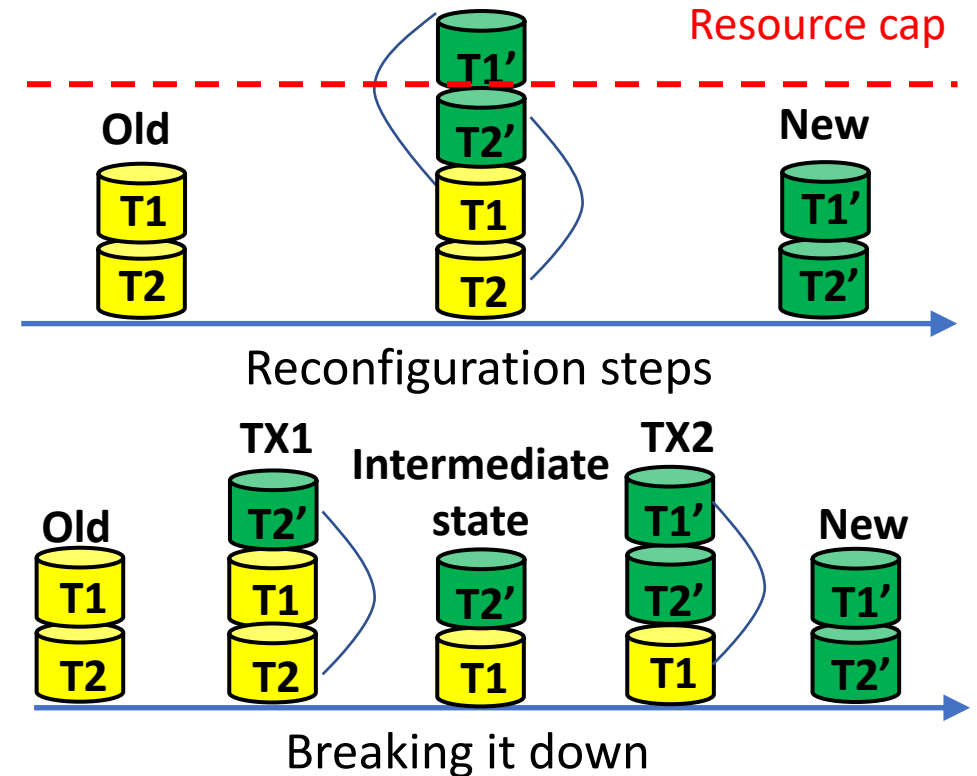


- Capable of runtime reconfiguration with **zero downtime**
 - FlexCore (NSDI'22) based on Nvidia Spectrum-2 switches
 - rP4 (NSDI'22) based on Xilinx Alevo FPGAs

Background: multi-step runtime update

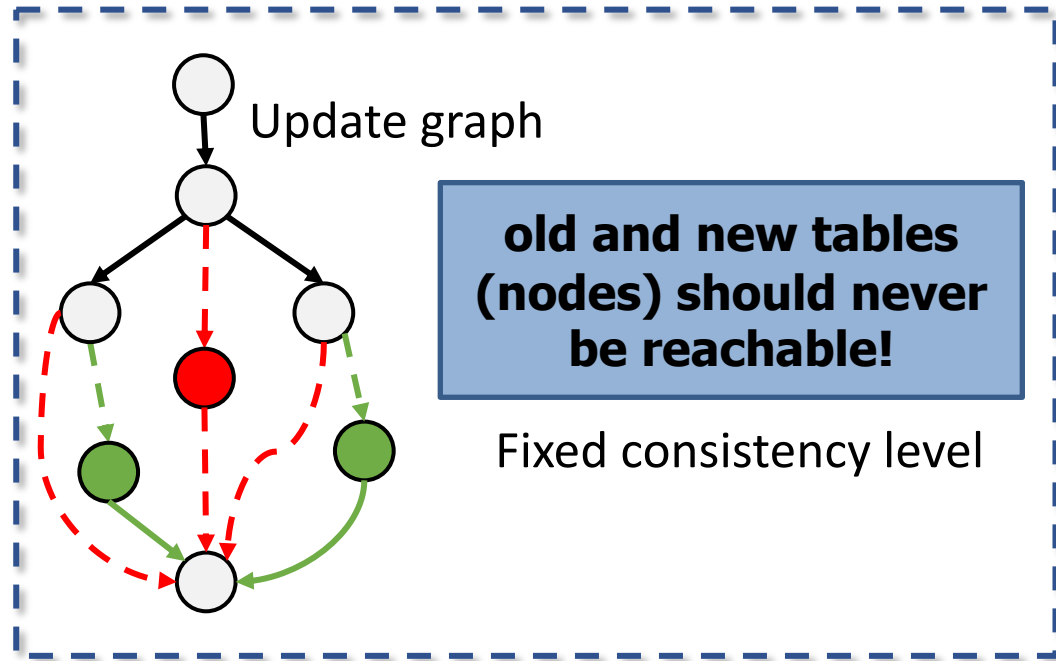
```
// Update acl tables to support nat
control ingress {
  apply {
    if (ipv4.isValid) {
      @del(T1) acl_ipv4.apply();
      @add(T1') nat_acl_ipv4.apply();
    } else if (ipv6.isValid) {
      @del(T2) acl_ipv6.apply();
      @add(T2') nat_acl_ipv6.apply();
    }
  }
}
```

P4 program update

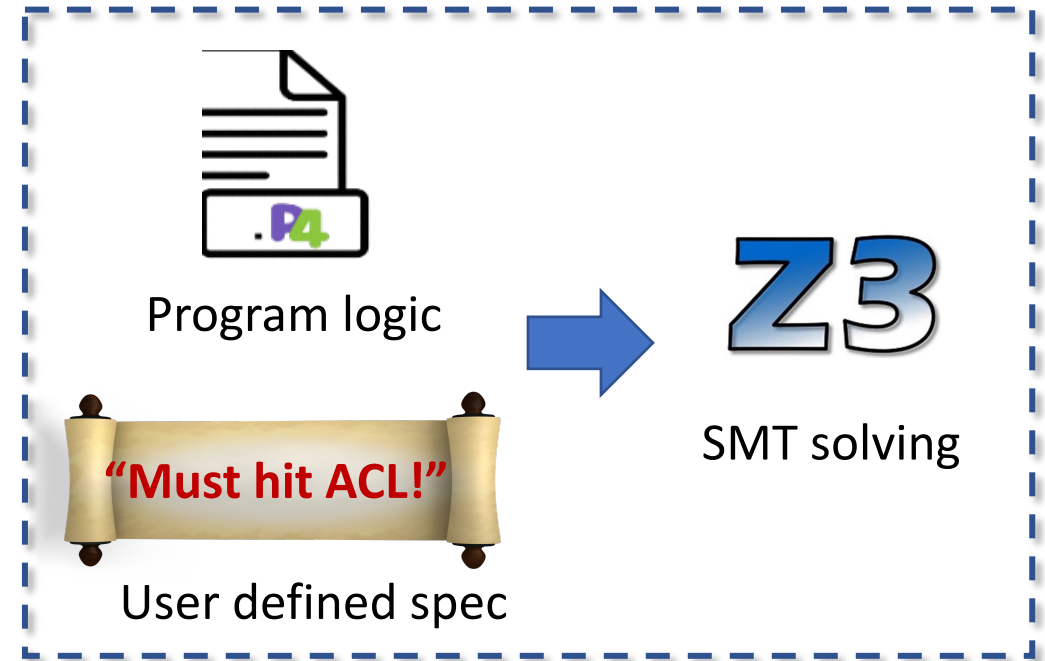


- Updating everything all at once not always feasible
 - Update transaction (TX) could cause significant resource spike
 - Multi-step update possible, but causes intermediate states
- Regulate intermediate states with consistency guarantees!

SOTA: graph analysis and P4 verification



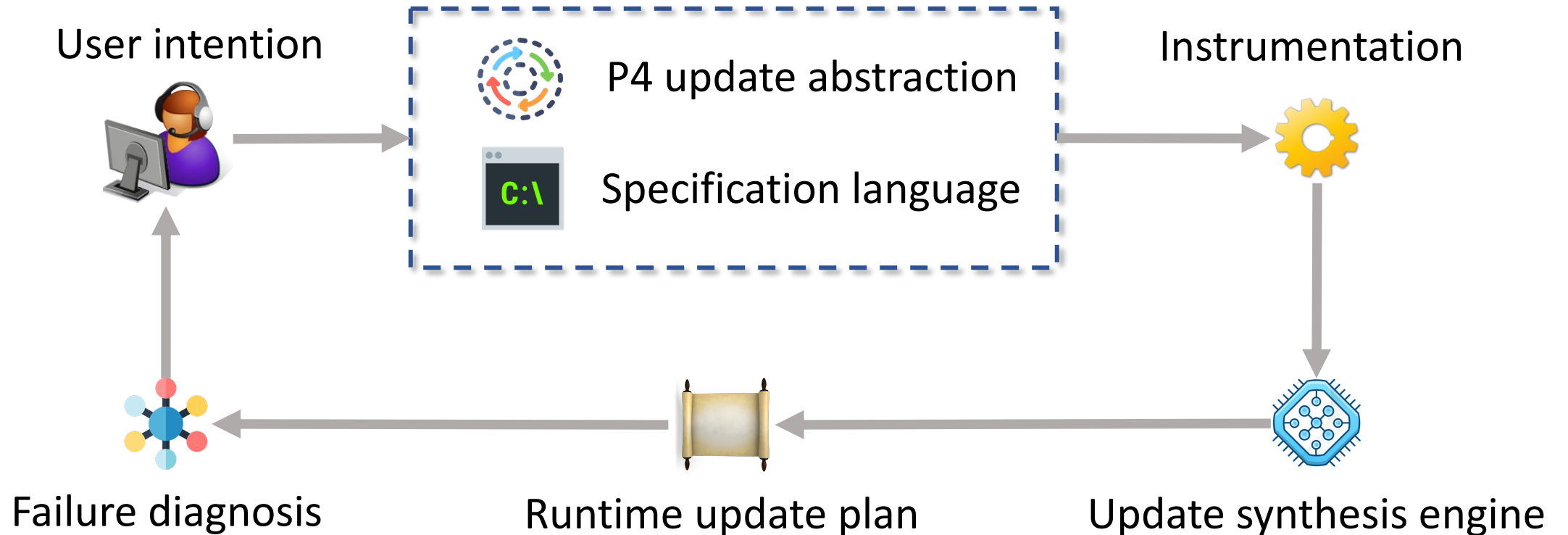
Update plan generation with
Graph analysis (FlexCore)



P4 verification

- Graph analysis does not support program semantics/user defined specs
- P4 verification does not support update reasoning/task of synthesis


FlexPlan: a formal synthesis approach



- **FlexPlan**: a formal approach to generate update plans that are both **safe** (consistency guarantee) and **feasible** (resource usage)

Specification language

```
specification {  
  // create new ghost variables for the program  
  ghost bit sawOld = false;  
  ghost bit sawNew = false;  
  // update variables when annotations encountered  
  @old => { sawOld = true; }  
  @new => { sawNew = true; }  
  // define no ipv4 packet mixes old and new logic  
  execution_consistency = {  
    $pkt.ingress.ipv4.isValid() =>  
      !($pkt.egress.sawOld && $pkt.egress.sawNew);  
  }  
  assert execution_consistency;  
}
```



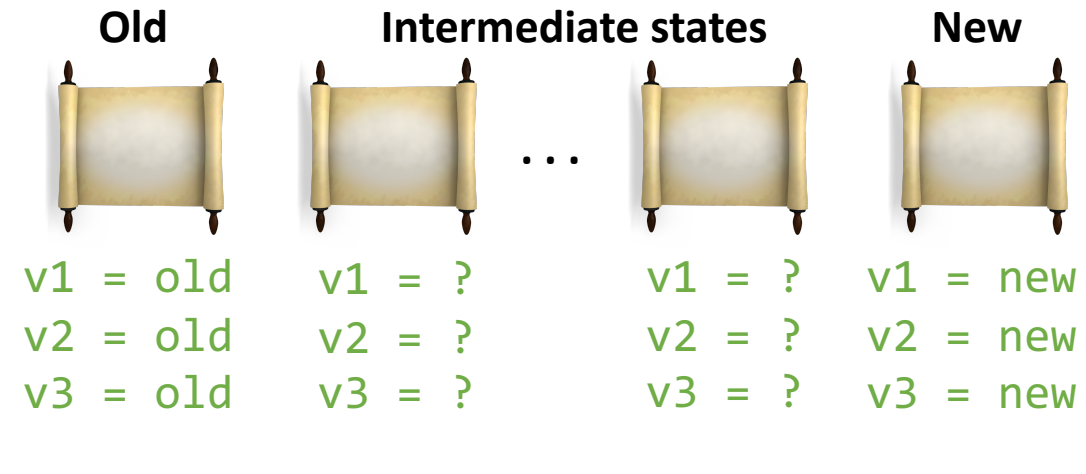
Specifications	LoC
Execution consistency for IPv4	13
Field consistency for egress spec	8
Program consistency for TCP	13
Element consistency for ACL	15
Table consistency for ECMP	10
Correct VLAN table access	8
Correct TTL decrement	6

- Flexible consistency/safety specification based on user intentions

Program update sketching

```
def func:  
  a = ? * 2  
  b = ? + a  
  c = ? % 4  
  return c - ?  
  
assert (func > 0)
```

```
v1, v2, v3 = ?, ?, ?  
if (v1 == new) {  
  // apply new table  
} else if (v1 == old) {  
  // apply old table  
}  
if (v2 == new) . . .  
if (v3 == new) . . .
```



Program sketch [1]:

Representing possible programs

Version sketch:

Representing possible intermediate states

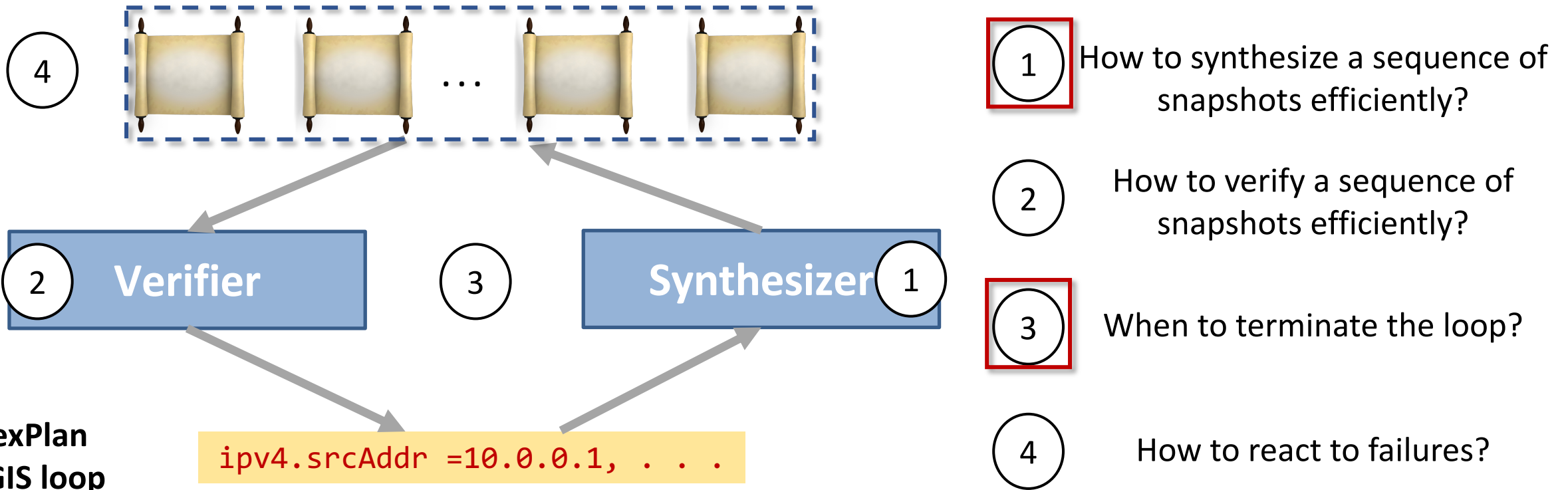
Sequence sketch:

Representing possible update plans (sequence of version sketches)

- Translating update plan generation into a sketching problem

FlexPlan CEGIS loop

Candidate plan:
A "guess" on feasible solution

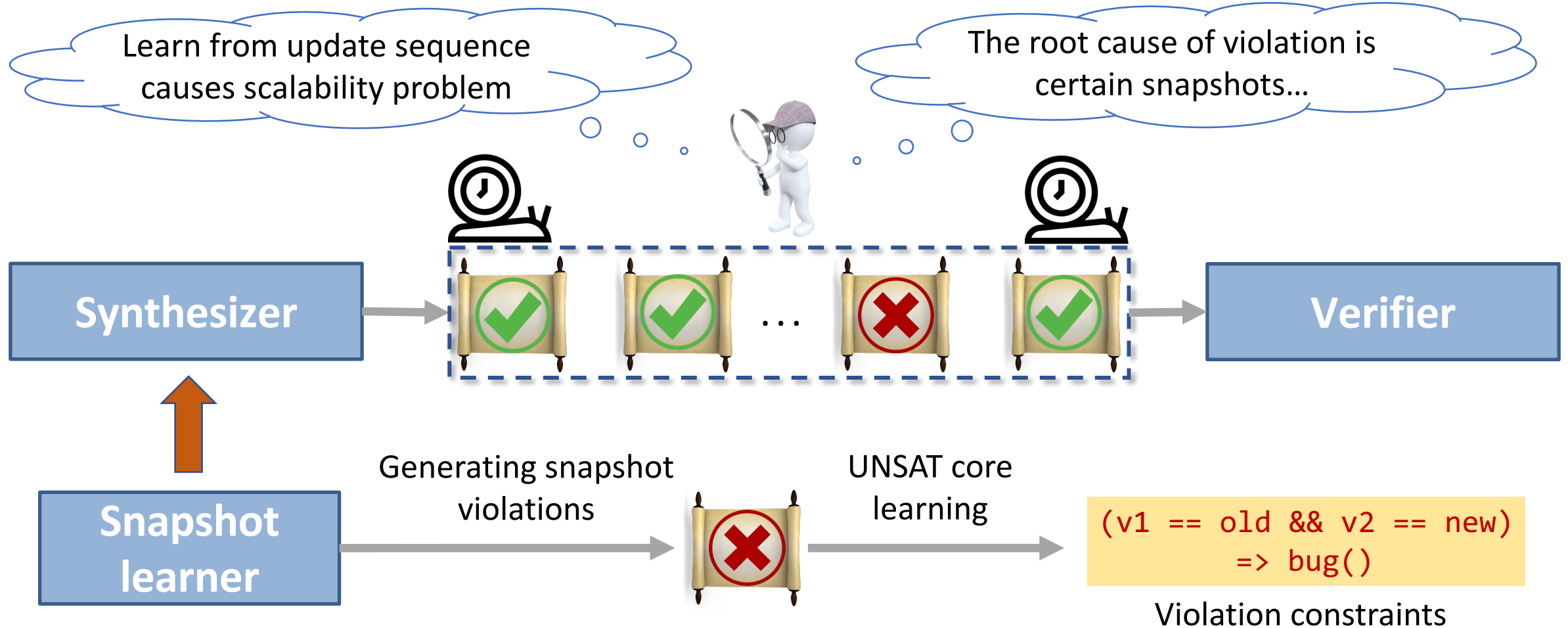


Counter example:

Concrete packet that triggers violation

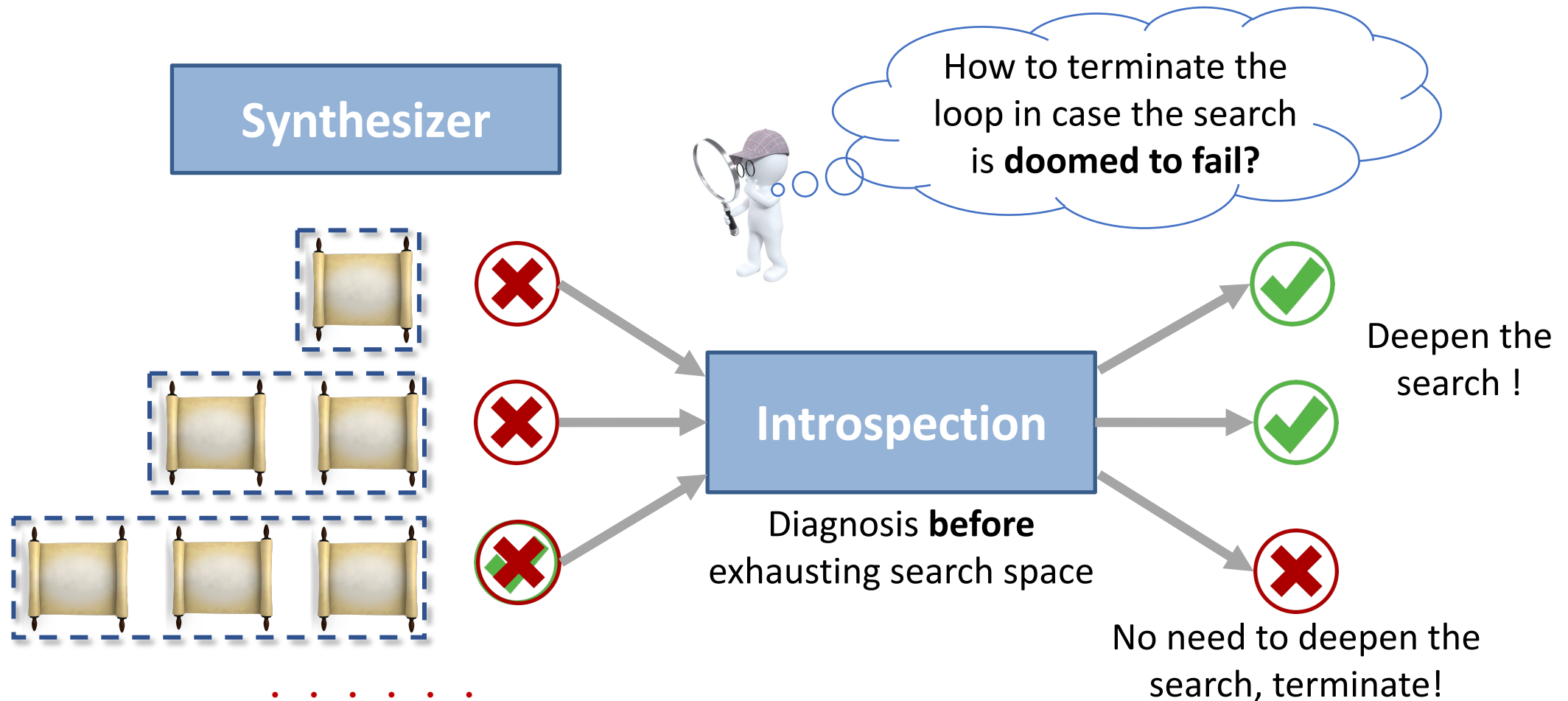
- Counter Example Guided Inductive Synthesis (CEGIS)

Optimization: Snapshot learning



- Learn from single snapshots, not entire sequences

Optimization: Loop termination

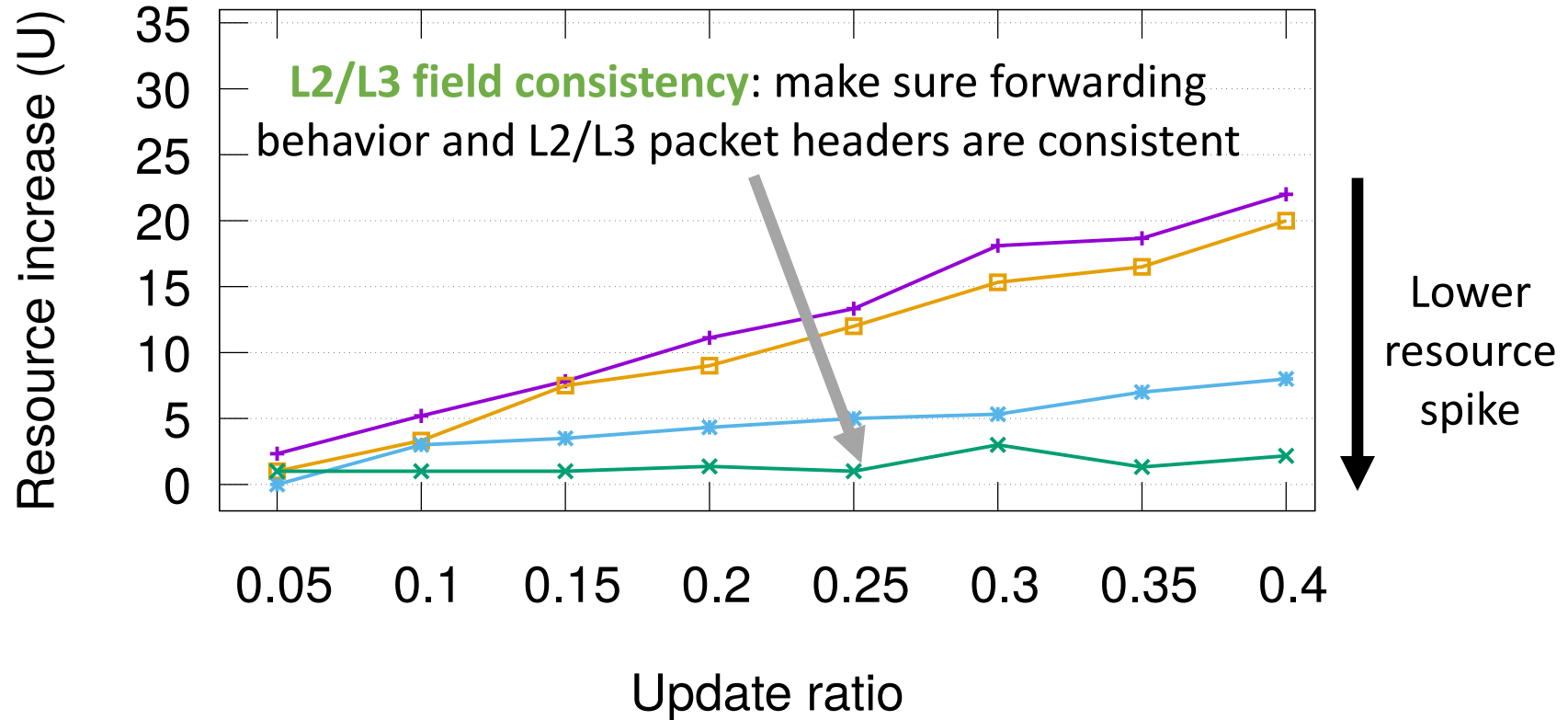


- Novel early termination technique (more in the paper)

Implementation and setup

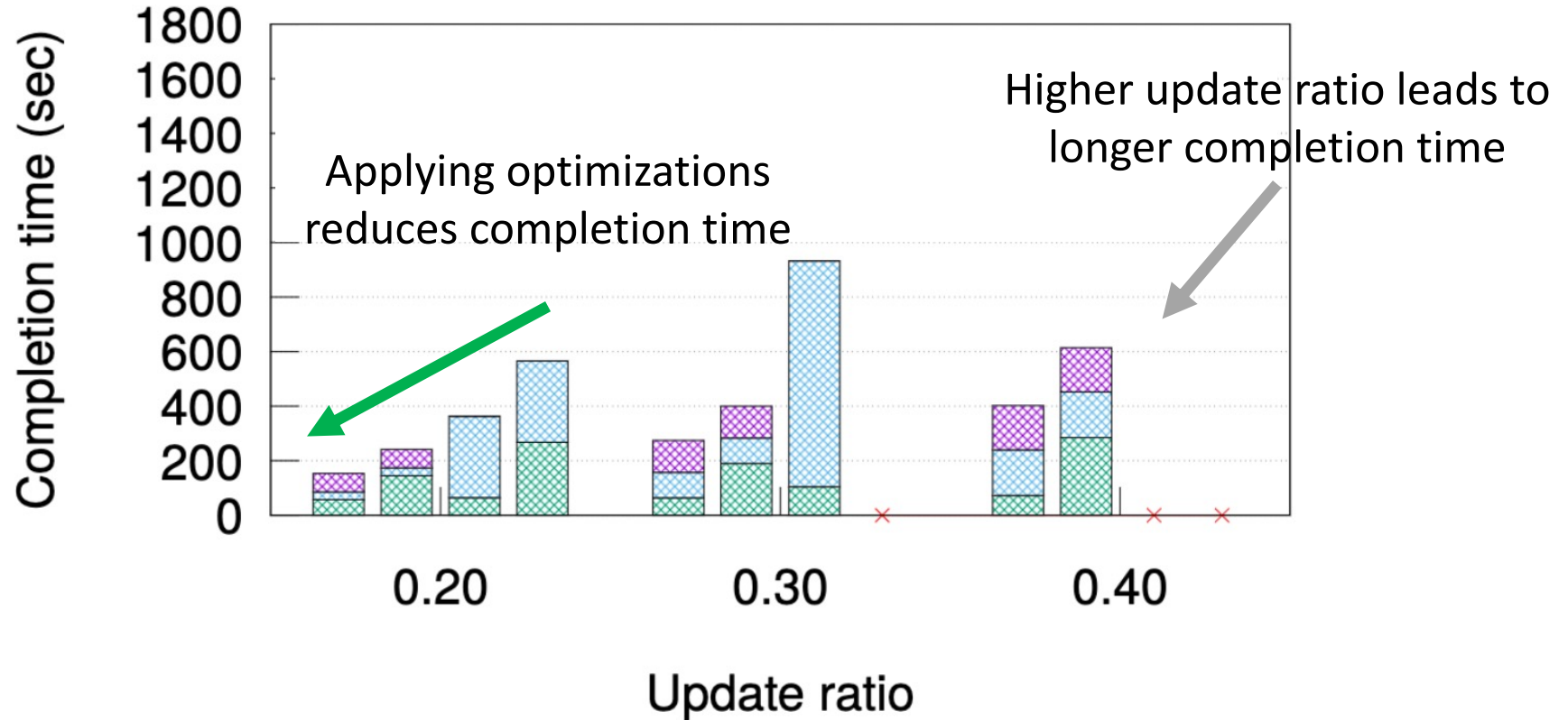
- FlexPlan prototype
 - <https://github.com/824728350/FlexPlan>
- Case study setup
 - Various P4 programs such as **switch.p4** (~6000LoC)
 - Both random and manually crafted program updates
- Evaluation questions (more in the paper!)
 - Do the specs generate granular update plans?
 - Do optimizations tackle scalability problem?

Expressive specs save resources



- FlexPlan supports various user defined consistency levels
- FlexPlan spec language provides significant resource saving

Optimizations improve scalability



- Higher update ratio results in longer completion time
- Snapshot learning/verification improve scalability significantly

Summary

- **FlexPlan**: a formal approach to generate update plans that are both **safe** (consistency guarantee) and **feasible** (resource usage)
- Hardware agnostic update plan generation framework
- Define your own runtime consistency requirements
- If there is a valid update plan, we'll find it.
- If we find a update plan, then it is correct!