Google Cloud

NetInfra

P4-BAR

Scaling P4-Based Automated Reasoning

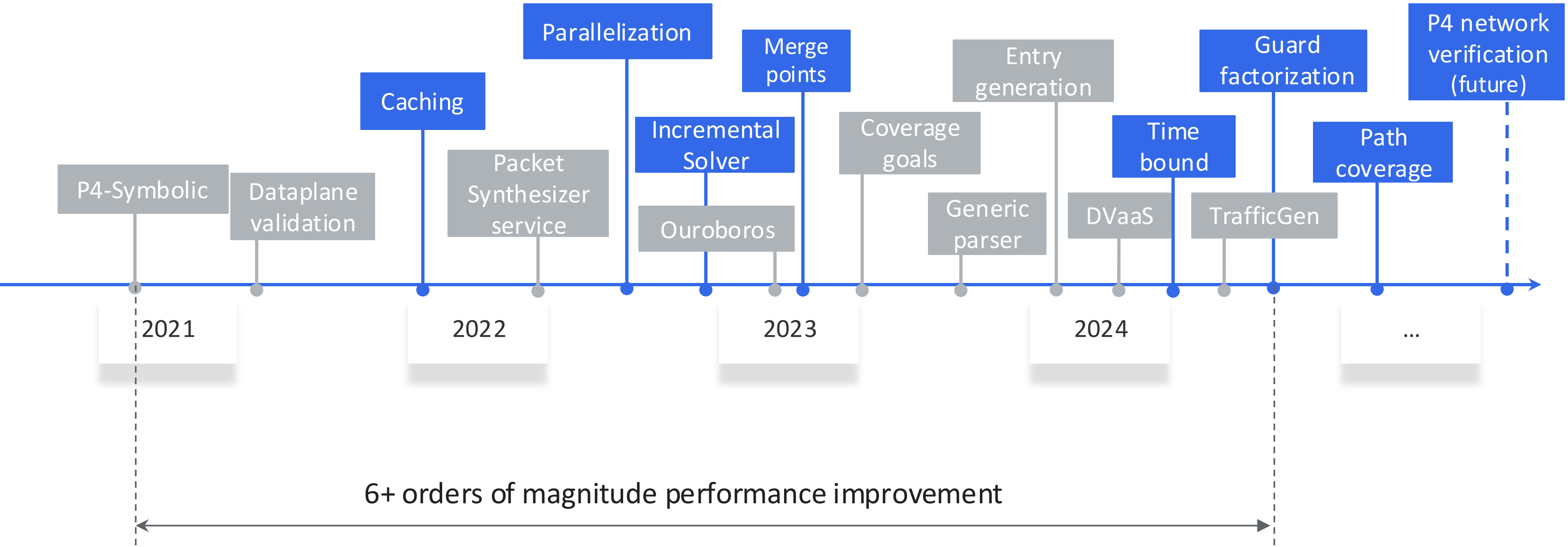
(Performance and Coverage)

P4 Workshop, Oct. 3, 2024

Ali Kheradmand (Google), Meghana Sistla (UT Austin*)

* Work done while at Google

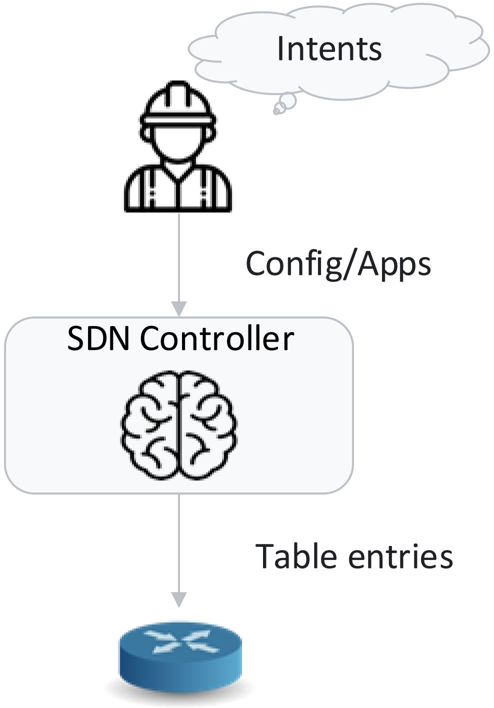
Highlights of Our Journey Developing P4-Symbolic



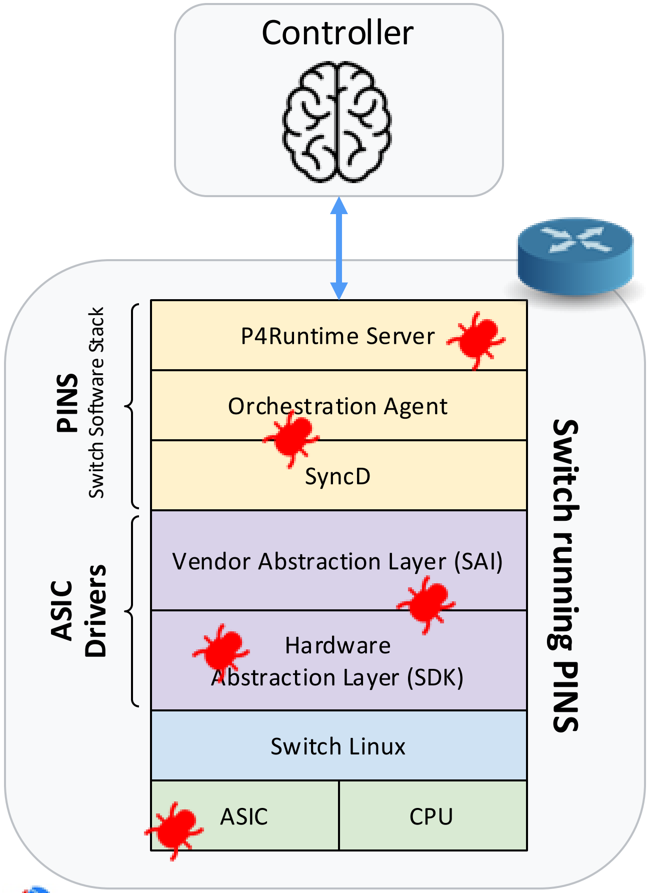
Overall goal: Ensure network works as intended

Subgoal 1: Ensure controller produces correct table entries (according to intents)

Subgoal 2: Ensure switch works as expected (according to table entries)



Focus: Ensure switch stack works as expected



Controller

P4Runtime Server

Orchestration Agent

SyncD

Vendor Abstraction Layer (SAI)

Hardware Abstraction Layer (SDK)

Switch Linux

ASIC

CPU

PINS Switch Software Stack

ASIC Drivers

Switch running PINS

Traditional: manually write tests

- Exponentially large space to cover
  - Labor intensive
- Hard to evolve

Our way: **Automatically** derive tests from a **formal specification** of how the switch should work

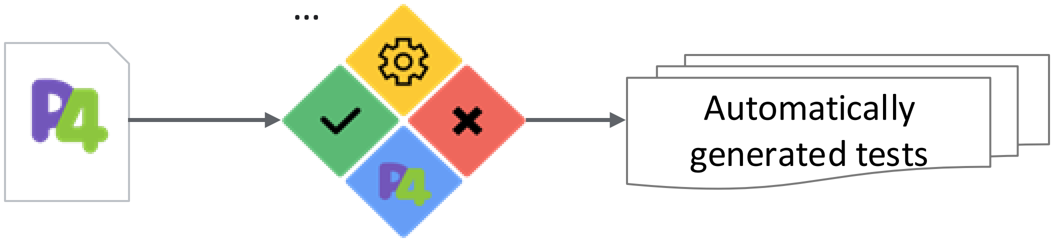
- Comprehensive coverage
- Effortless evolution

Need:

1. Specification language
2. Test generation tools

P4-Based Automated Reasoning (P4-BAR)

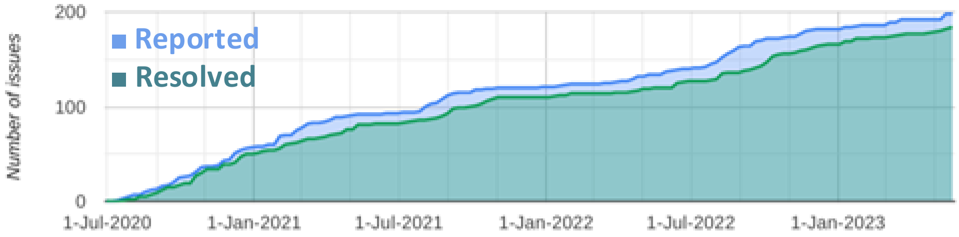
P4-Symbolic
P4-Fuzzer
...



Automatically generated tests

300+ (unique) bugs found so far

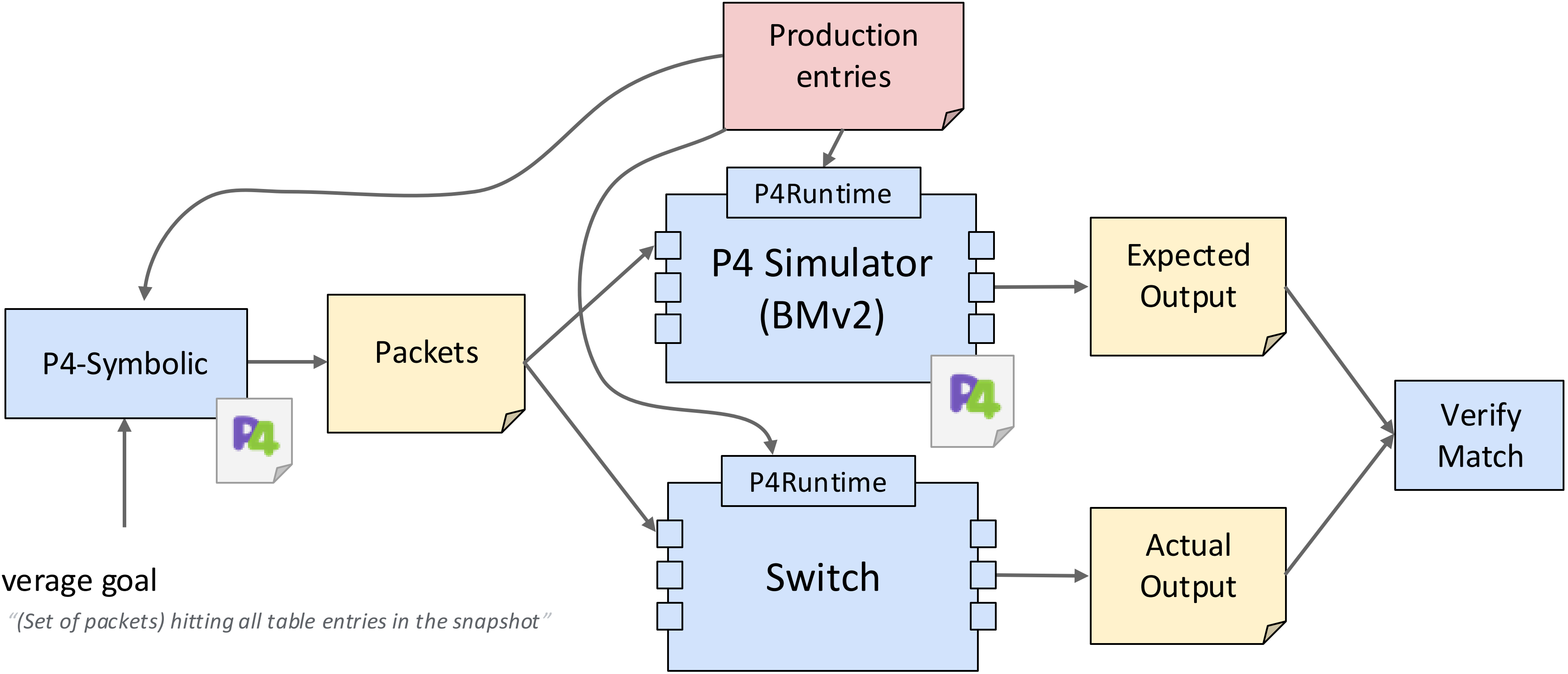
(and many more bugs prevented)



SwitchV: Automated SDN Switch Validation with P4 Models (SIGCOMM’22)

Kinan Dak Albab, Jonathan Dilorenzo, Stefan Heule, Ali Kheradmand, Steffen Smolka, Konstantin Weitz, Muhammad Tirmazi, Jiaqi Gao, Minlan Yu

Dataplane Validation

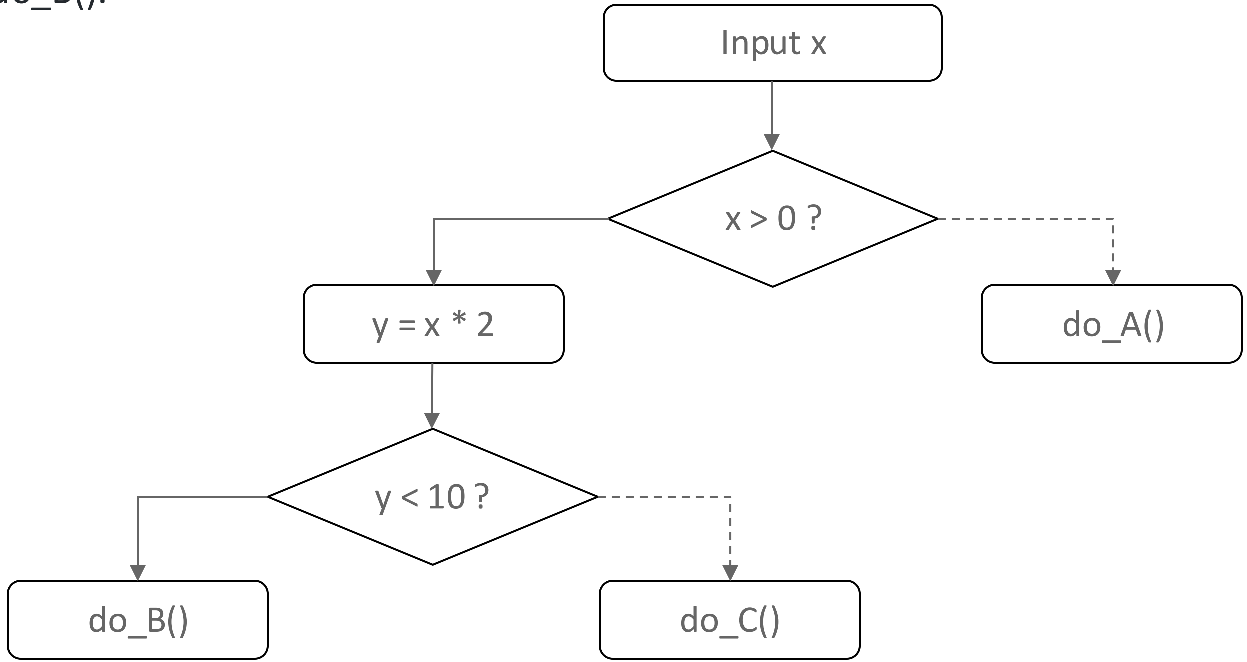


Coverage goal

E.g. "(Set of packets) hitting all table entries in the snapshot"

Symbolic Execution

“Give me the input that triggers “do_B().”



Cover: do_B( )

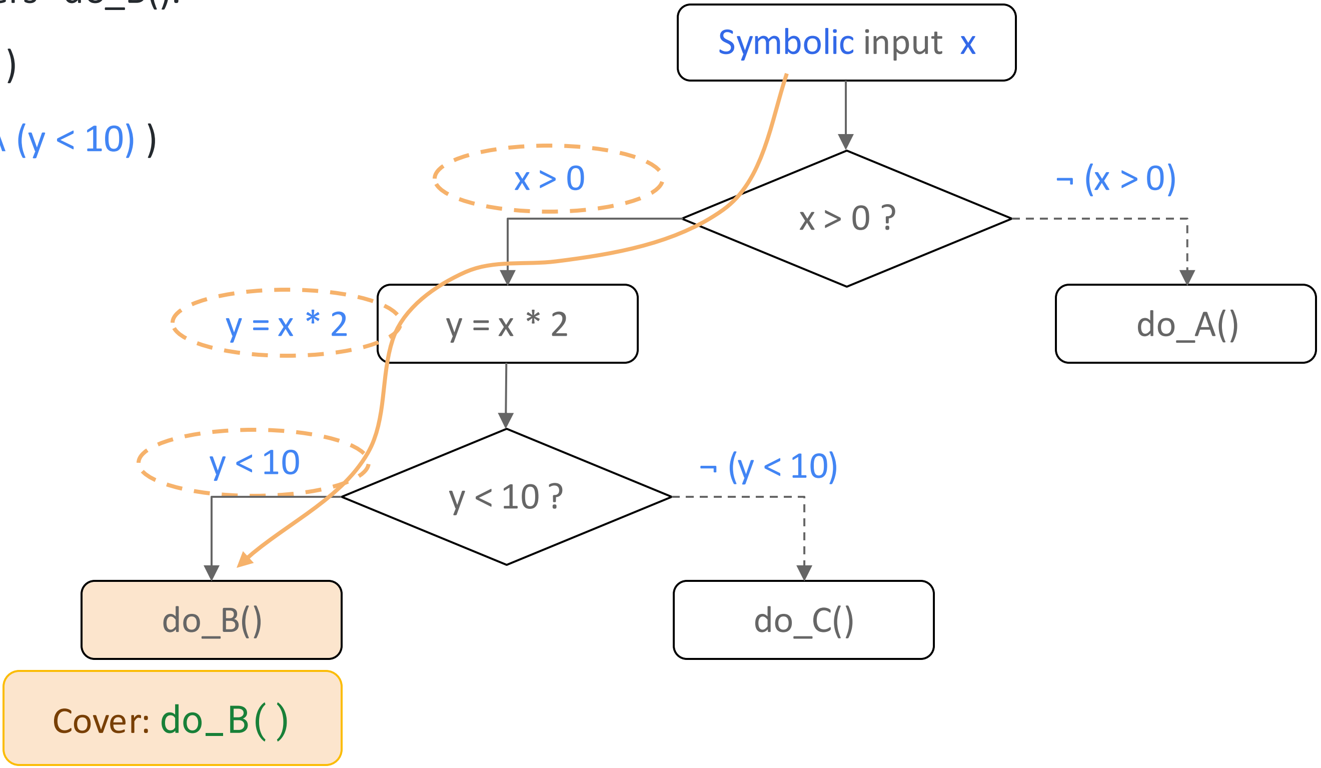
Symbolic Execution

“Give me the input that triggers “do_B().”

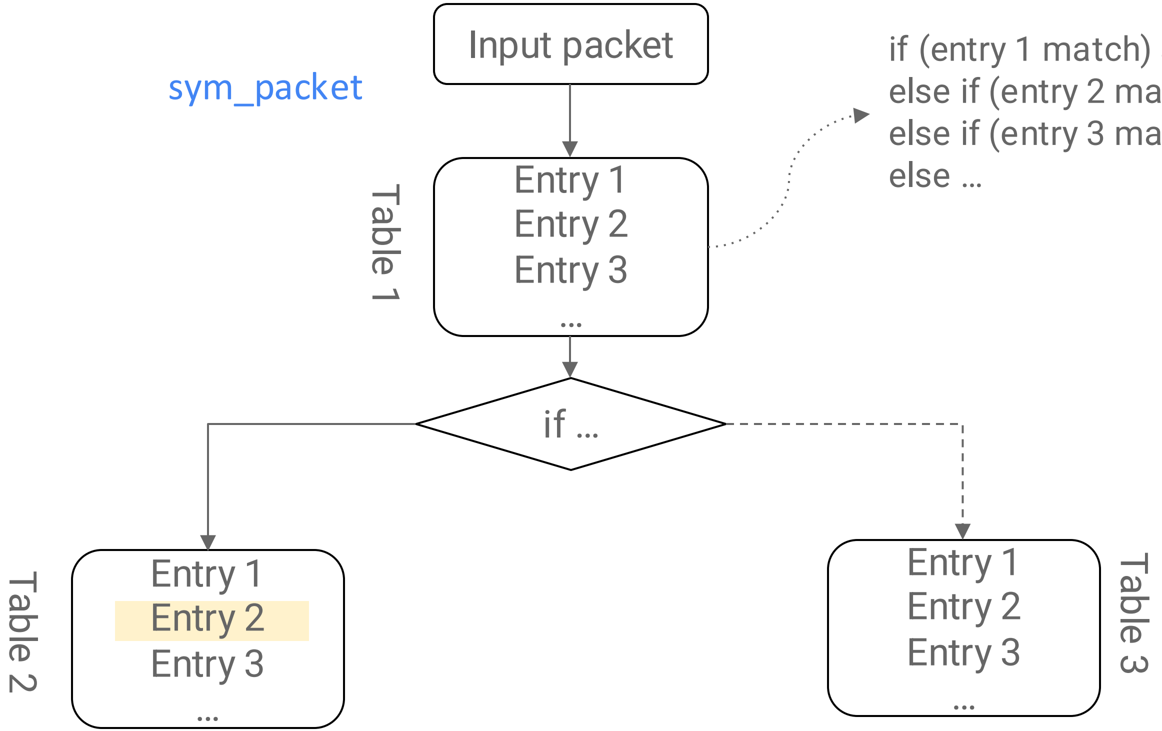
⇒ Solve(“do_B() is reached”)

⇒ Solve($(x > 0) \wedge (y = x * 2) \wedge (y < 10)$)

⇒ A solution: {**x = 1**, y = 2}
(with Z3 solver)



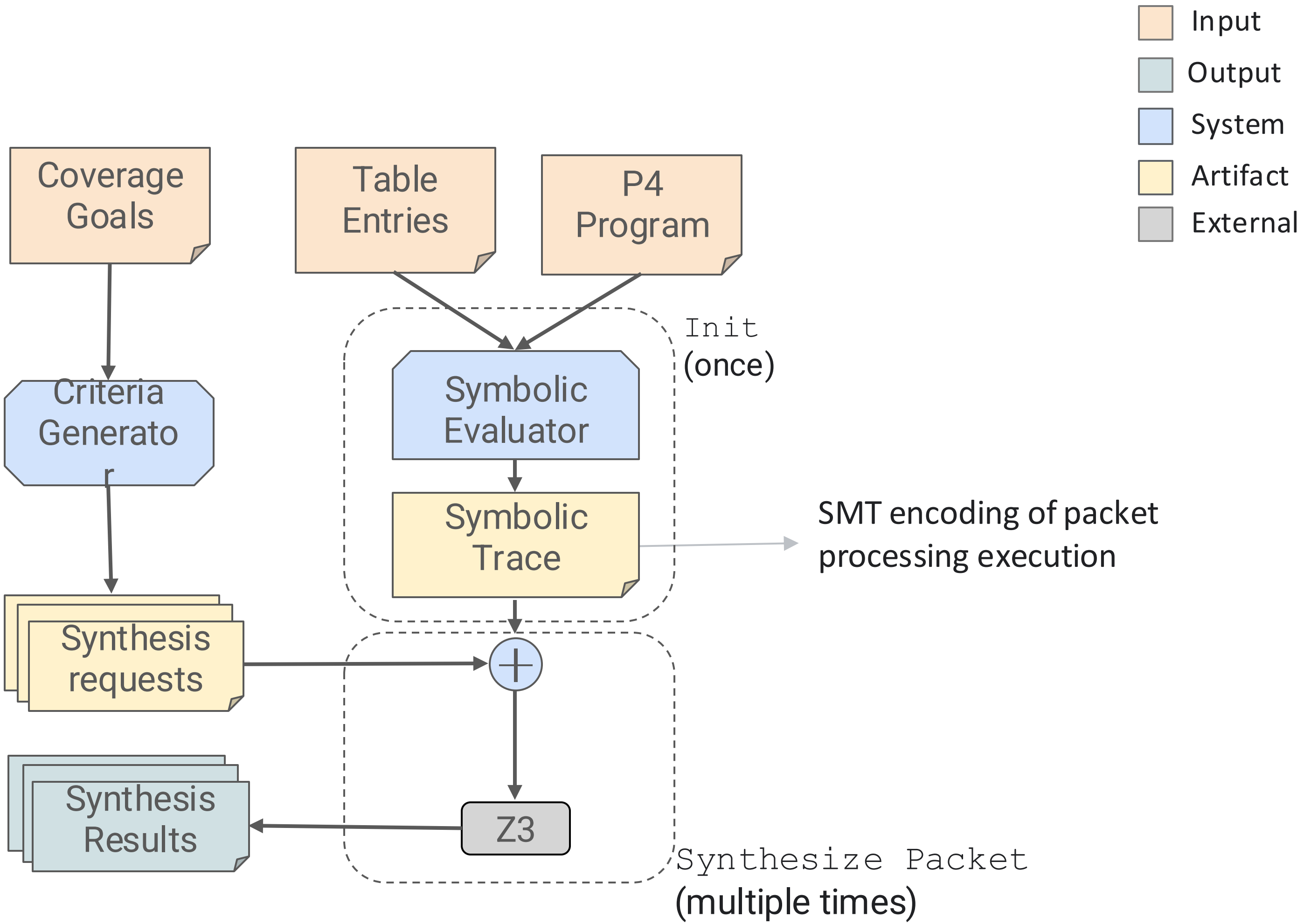
Symbolic Execution in P4



if (entry 1 match) action 1
else if (entry 2 match) action 2
else if (entry 3 match) action 3
else ...

P4-Symbolic

e.g.
“Hitting all table entries”

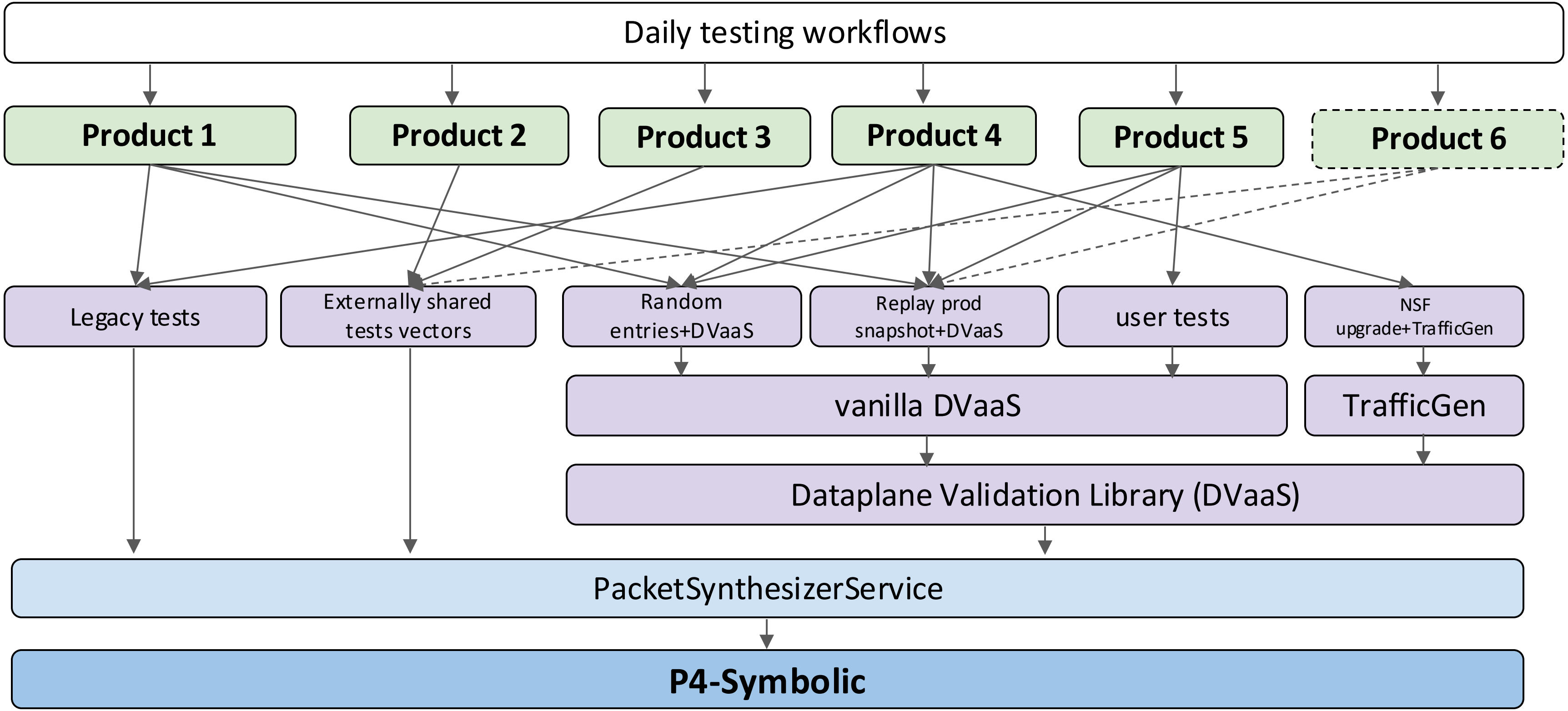


SMT encoding of packet processing execution

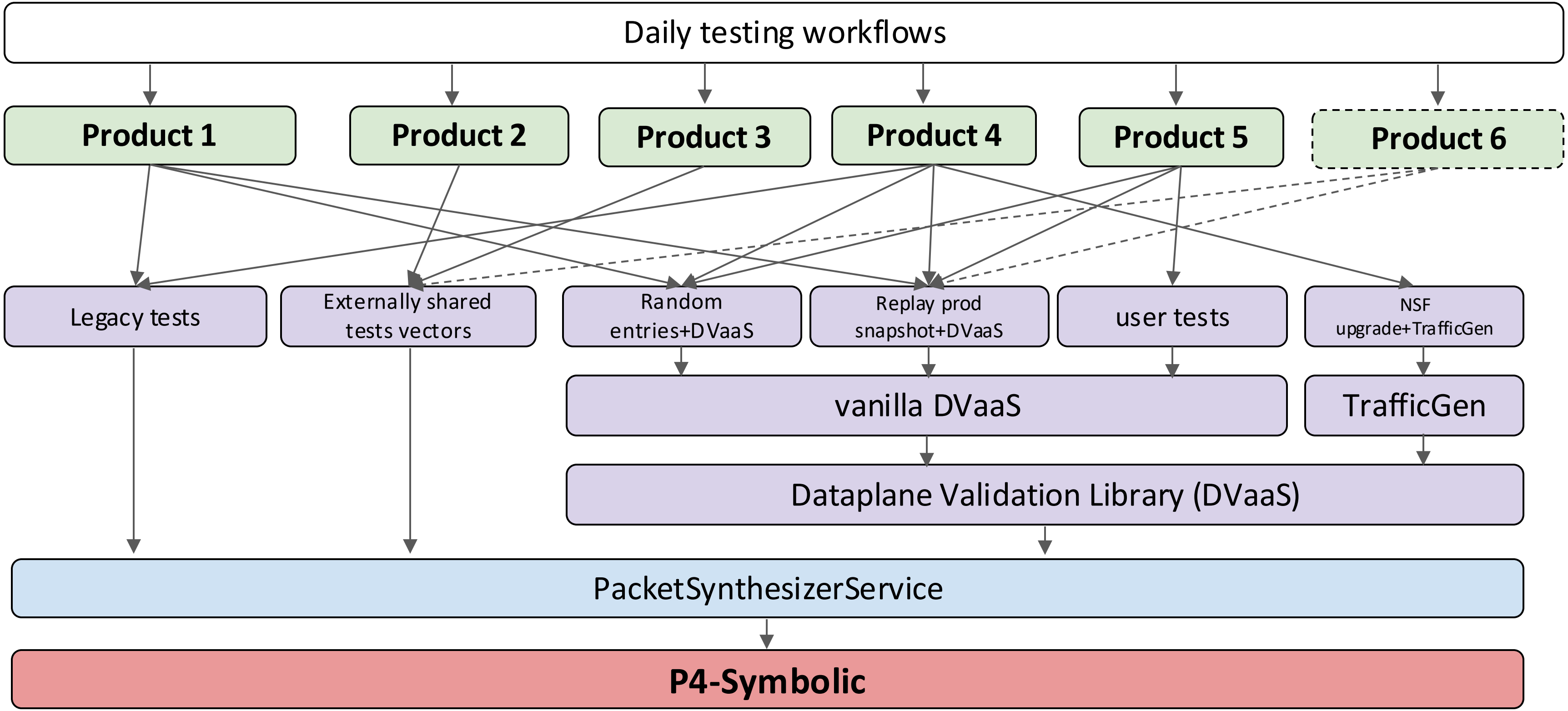
e.g.
Request 1: packet must hit *12th* entry in table *acl_ingress*
Request 2: …
…

e.g.
Result 1: packet = …
Result 2: unsatisfiable
…

P4-Symbolic in use



Problem



NP-hard problem => computationally expensive => bottleneck

Outline

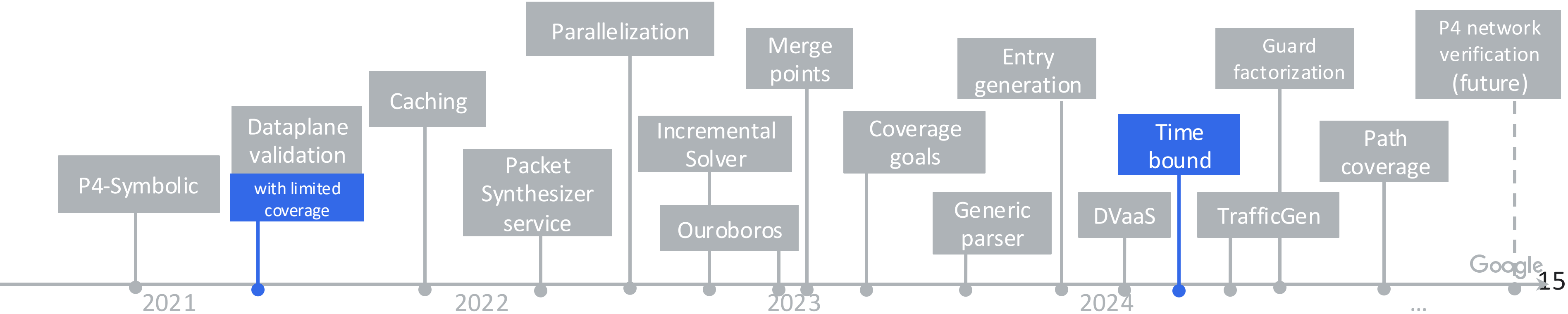
- Background and Context
- P4-Symbolic
- Performance Improvements
- Coverage Improvements
- Future

Undesirable “Solution”

Reduce coverage

- Smaller coverage goals
  - E.g. Ignore expensive tables, entries
- Time bound coverage:
  - Stop execution after a certain “time limit” (even if coverage goal not achieved)

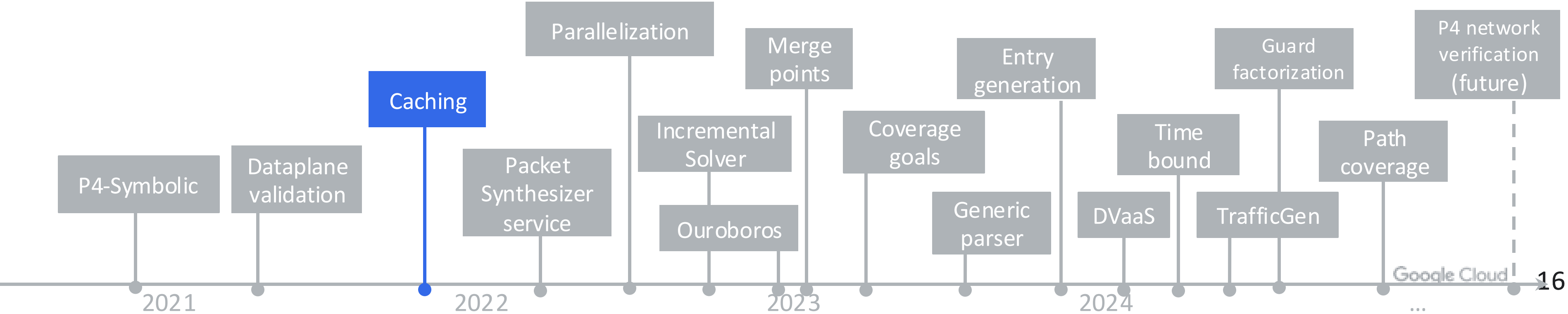
Undesirable, but at times necessary as a last resort



1. Offline packet synthesis (caching)

No need to regenerate packets unless inputs (P4 model, entries, goals) change

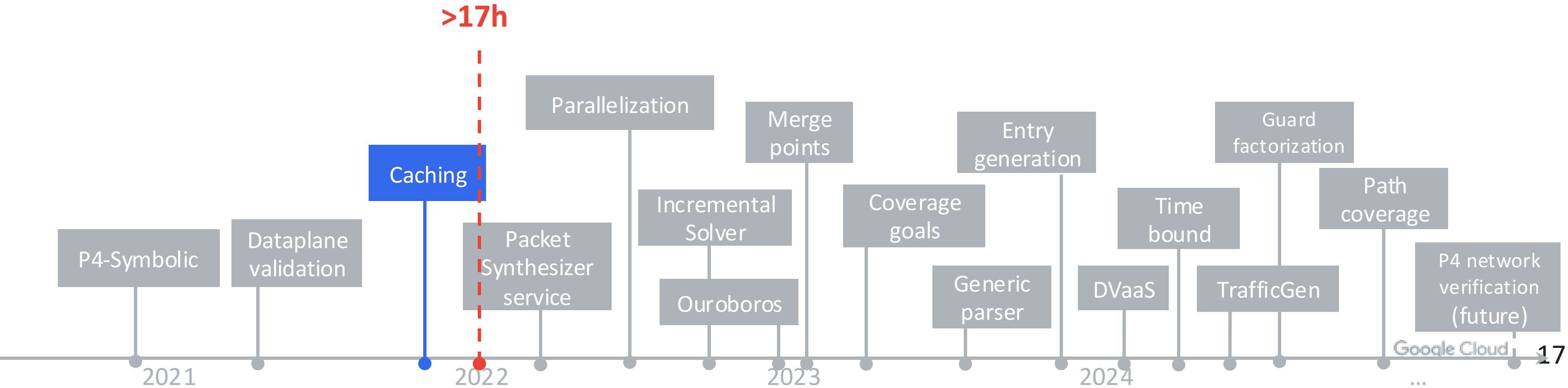
Do not allow code merge until cache is populated



1. Offline packet synthesis (caching)

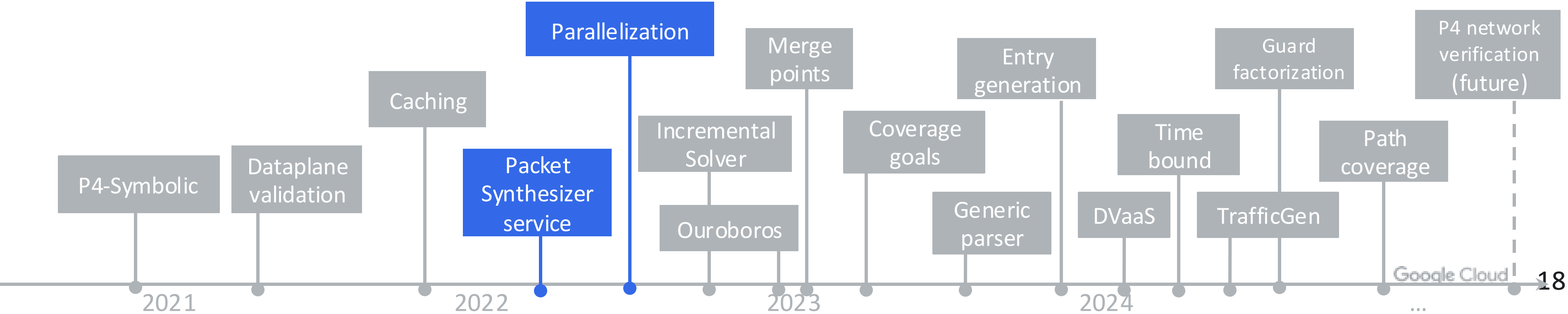
Caveats

- Time to populate cache
- Frequent P4 model updates
  - Headache with concurrent development
- Ineffective in tests that frequently update entries,



Google Cloud

2. Parallelization



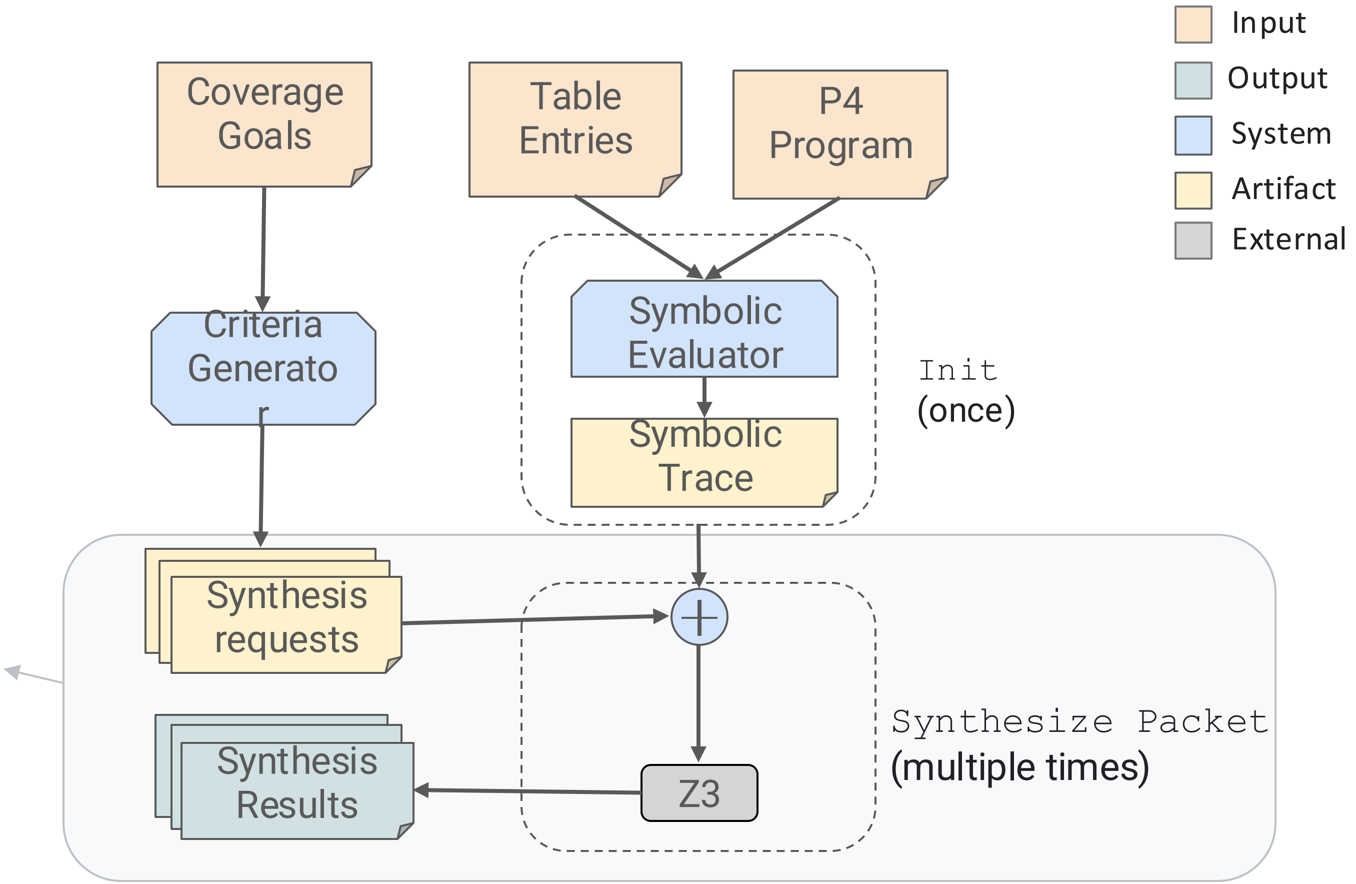
Google Cloud

2. Parallelization

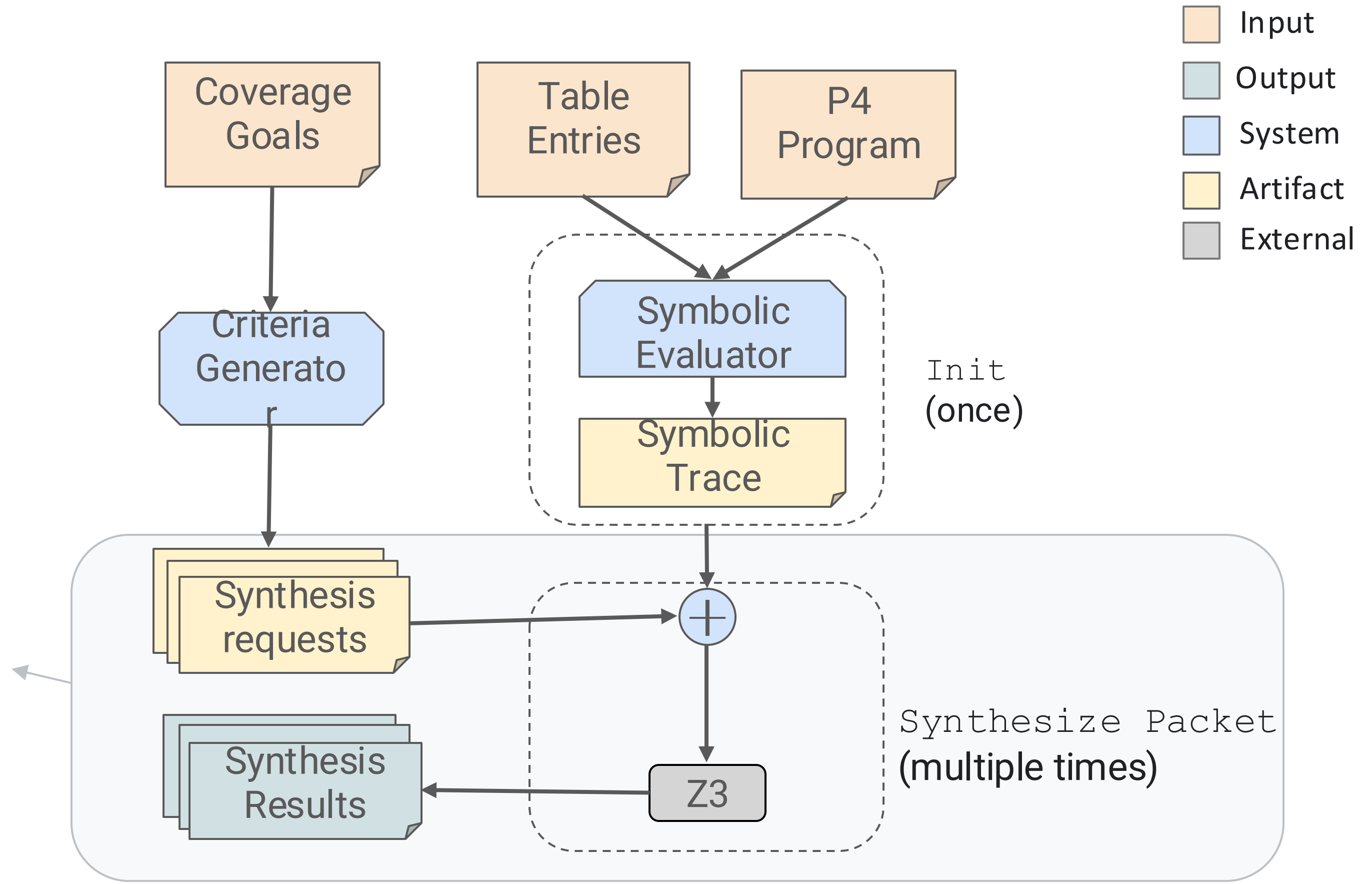
Independent of each other

=>

embarrassingly parallelizable



2. Parallelization



Independent of each other

⇓

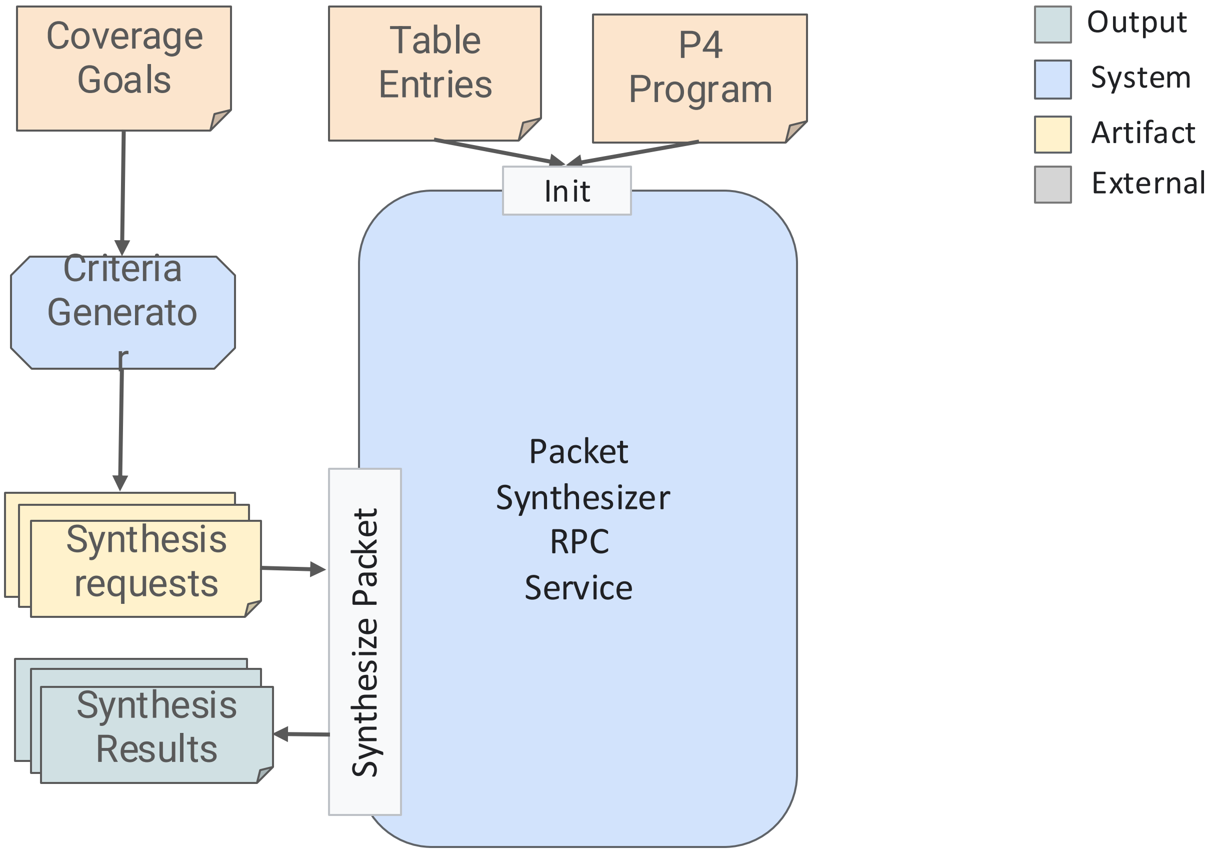
embarrassingly parallelizable

2. Parallelization

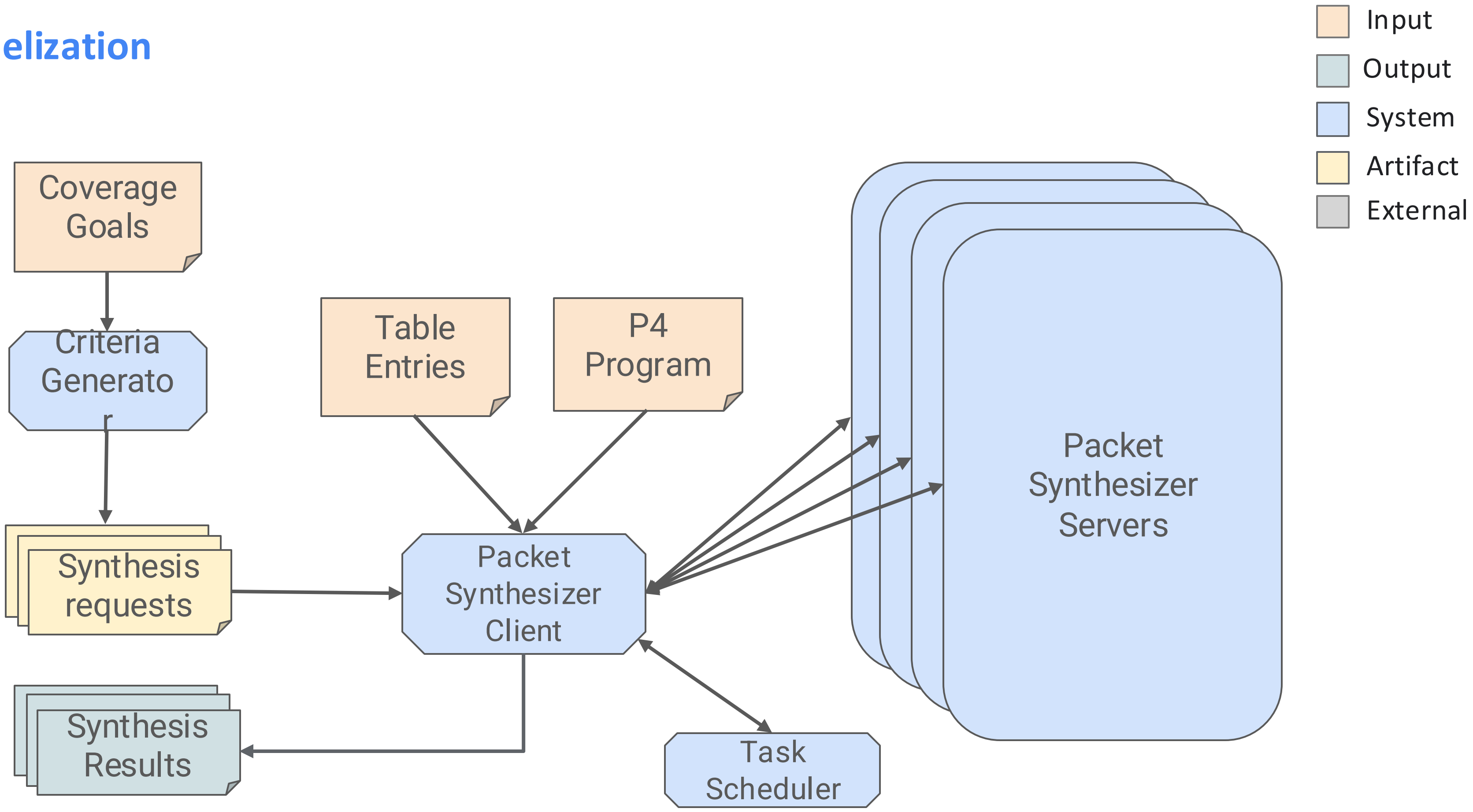
Independent of each other

⇓

embarrassingly parallelizable

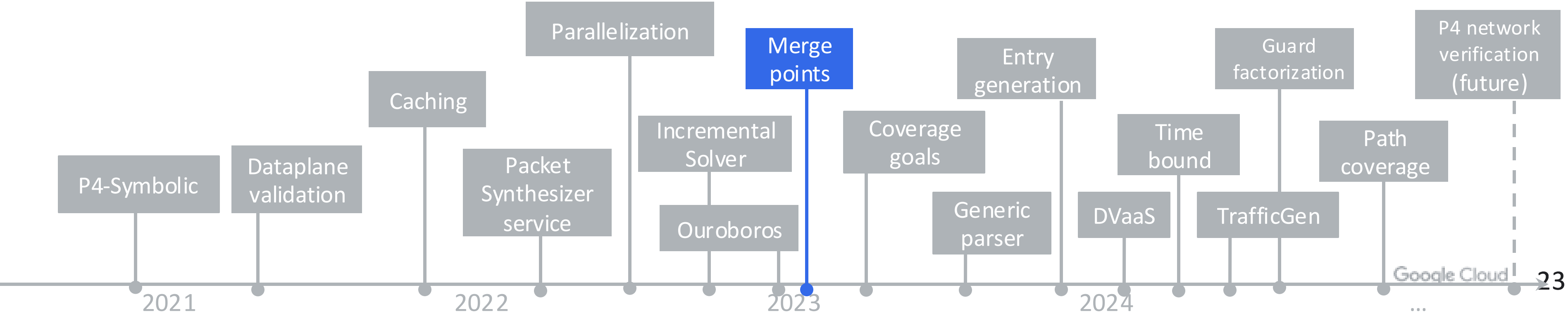


2. Parallelization



Google Cloud

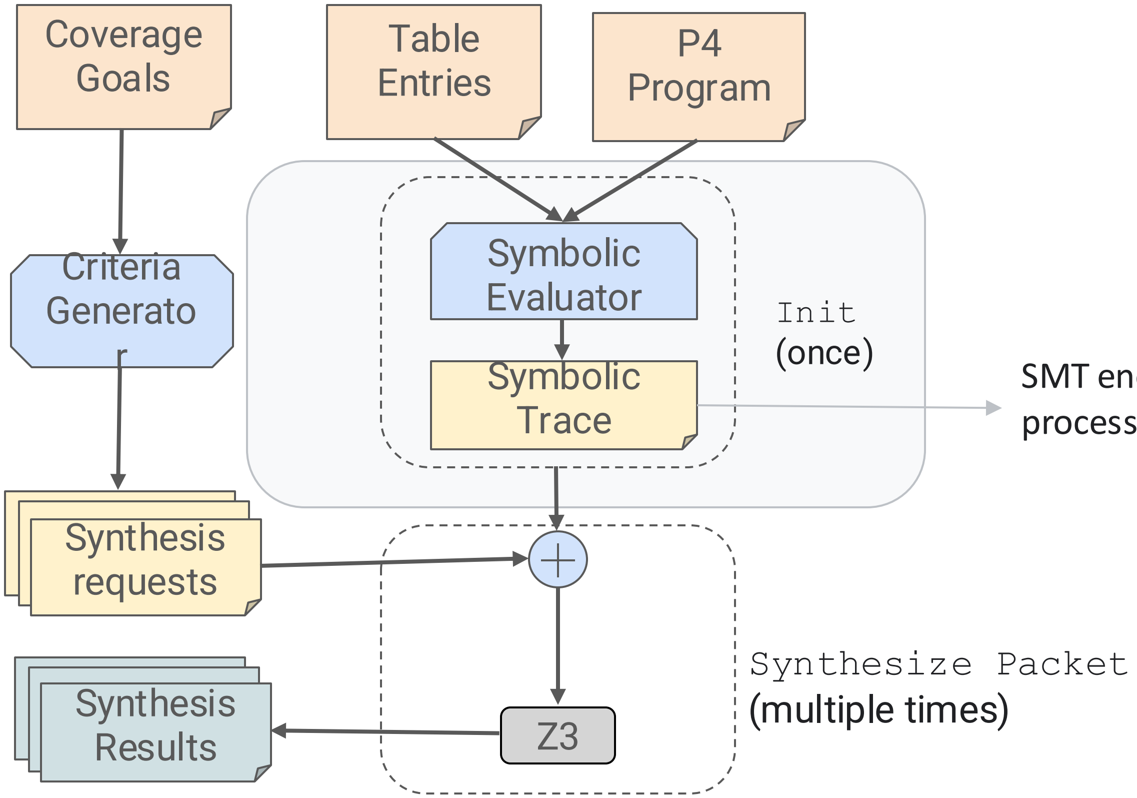
3. Symbolic Execution Merge Points



Google Cloud

3. Symbolic Execution Merge Points

■ Input
■ Output
■ System
■ Artifact
■ External



Coverage Goals

Table Entries

P4 Program

Criteria Generato r

Symbolic Evaluator

Symbolic Trace

Init (once)

Synthesis requests

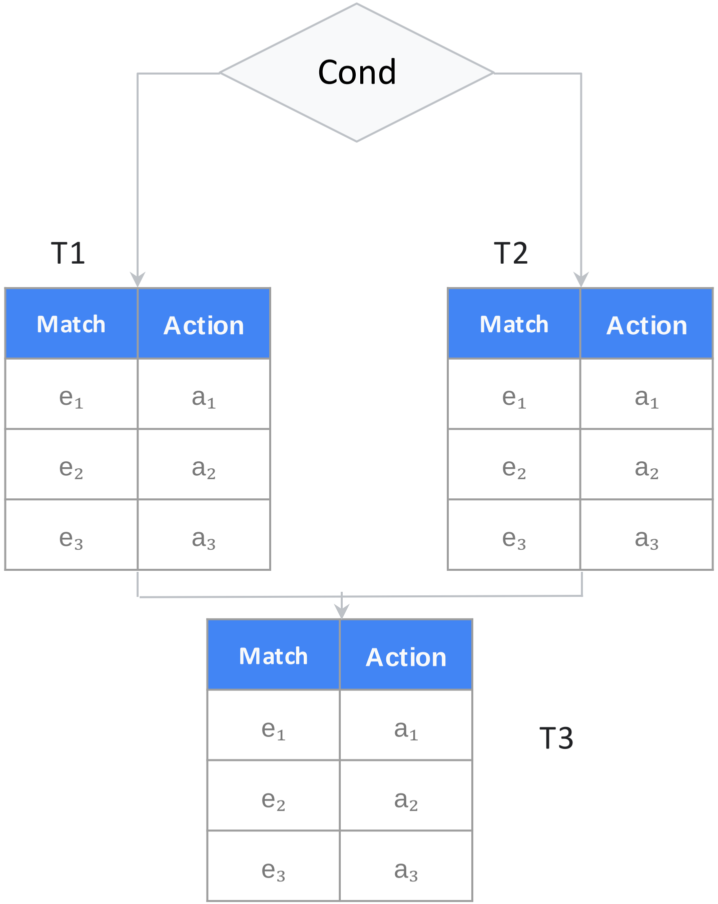
Z3

Synthesis Results

Synthesize Packet (multiple times)

SMT encoding of packet processing execution

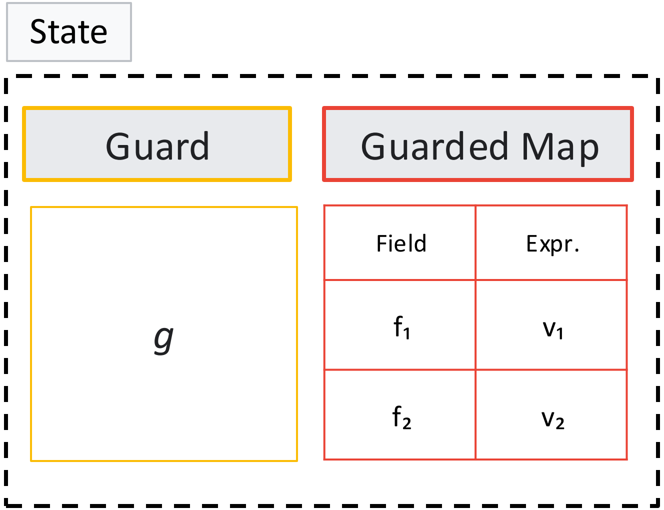
3. Symbolic Execution Merge Points



Match	Action
e_1	a_1
e_2	a_2
e_3	a_3

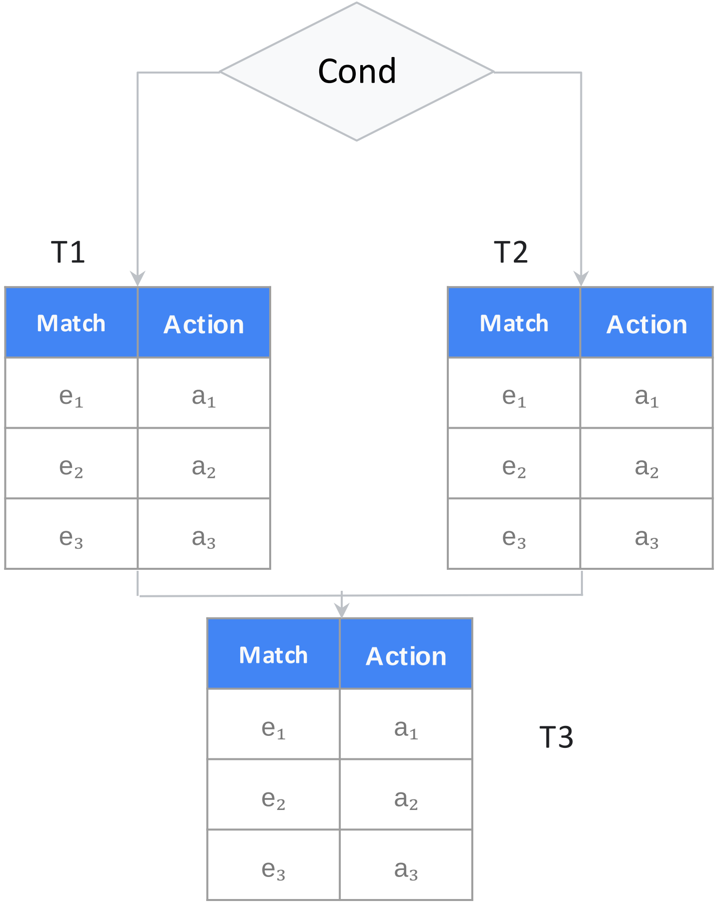
Match	Action
e_1	a_1
e_2	a_2
e_3	a_3

Match	Action
e_1	a_1
e_2	a_2
e_3	a_3



Field	Expr.
f_1	v_1
f_2	v_2

3. Symbolic Execution Merge Points



T1

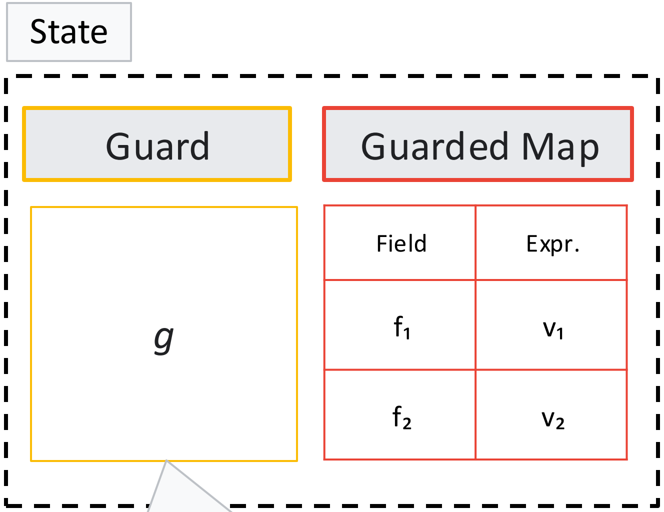
Match	Action
e_1	a_1
e_2	a_2
e_3	a_3

T2

Match	Action
e_1	a_1
e_2	a_2
e_3	a_3

T3

Match	Action
e_1	a_1
e_2	a_2
e_3	a_3

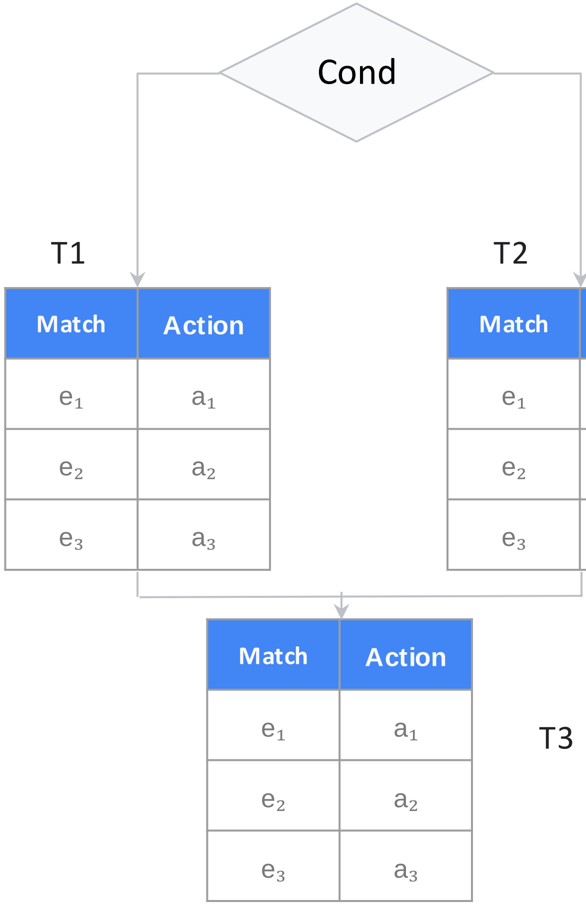


Field	Expr.
f_1	v_1
f_2	v_2

Guard: conditions that allow execution to reach the current point

When is it updated: (1) Conditionals, (2) Table entry **match** condition

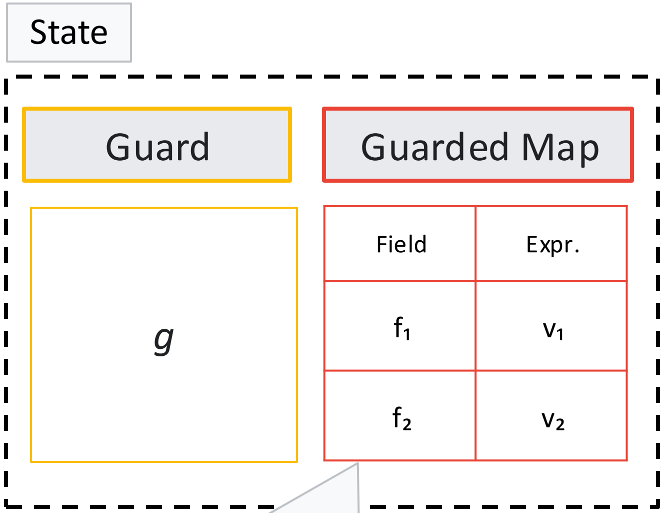
3. Symbolic Execution Merge Points



Match	Action
e_1	a_1
e_2	a_2
e_3	a_3

Match	Action
e_1	a_1
e_2	a_2
e_3	a_3

Match	Action
e_1	a_1
e_2	a_2
e_3	a_3



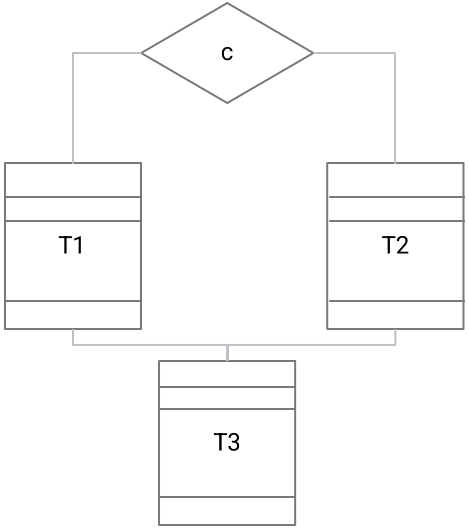
Field	Expr.
f_1	v_1
f_2	v_2

Guarded Map: *fields in header + metadata -> SMT expression* (value of the field at the current execution point)

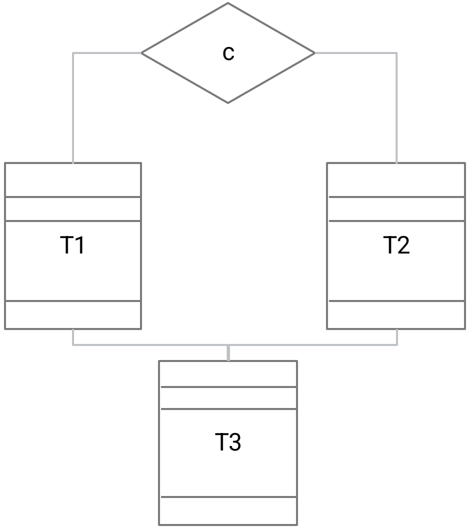
When is it updated: Table entry **action**

3. Symbolic Execution Merge Points

No Merge Points

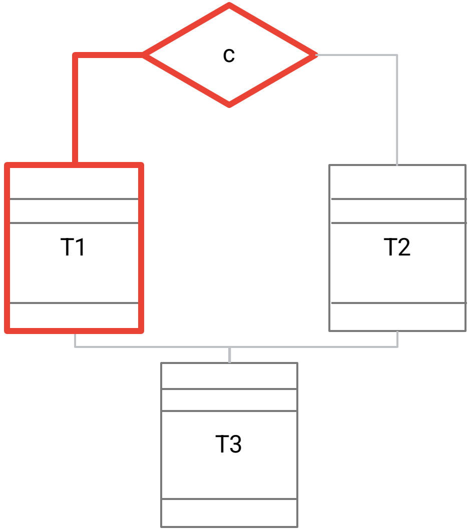


With Merge Points

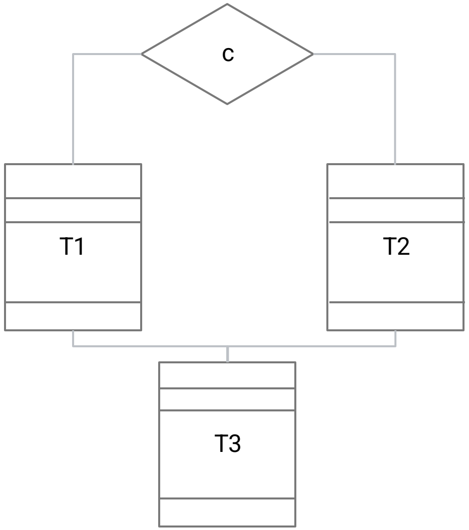


3. Symbolic Execution Merge Points

No Merge Points

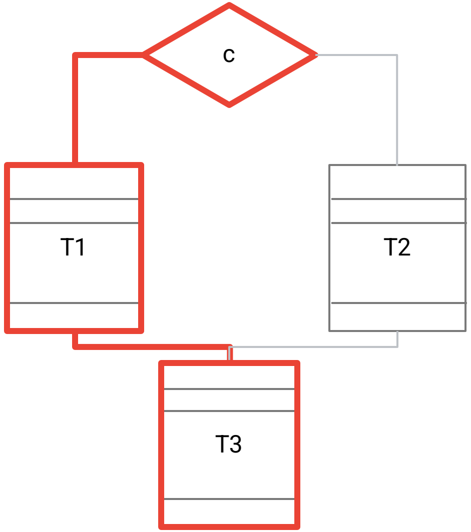


With Merge Points

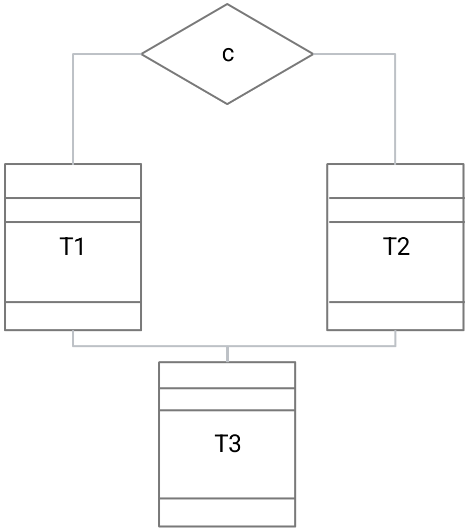


3. Symbolic Execution Merge Points

No Merge Points

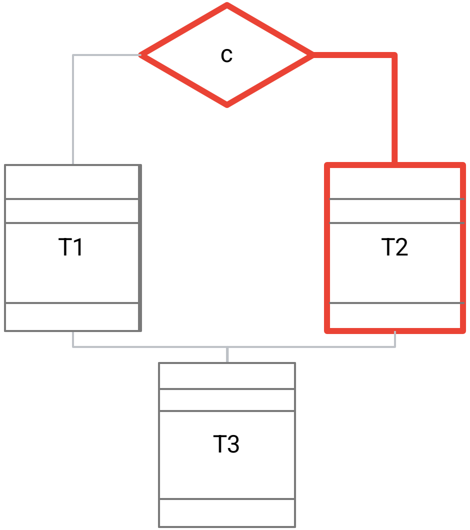


With Merge Points

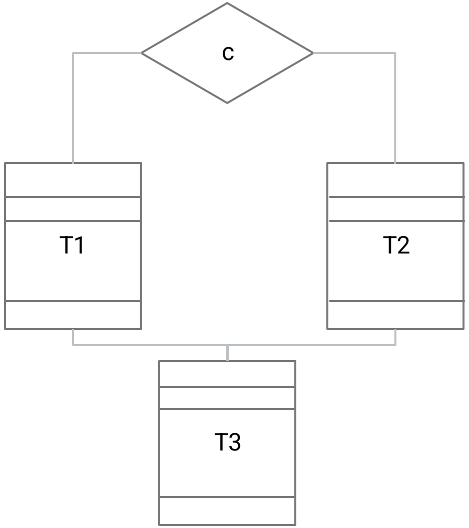


3. Symbolic Execution Merge Points

No Merge Points

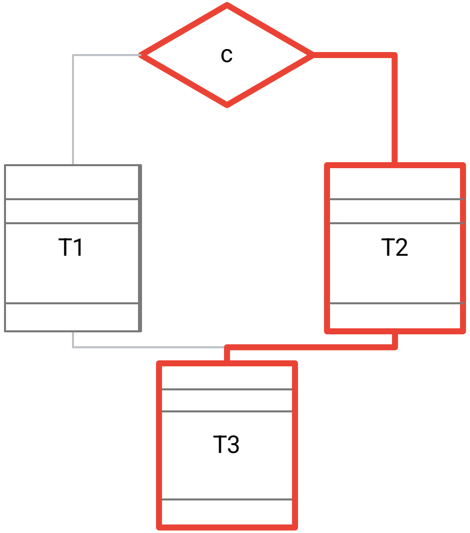


With Merge Points

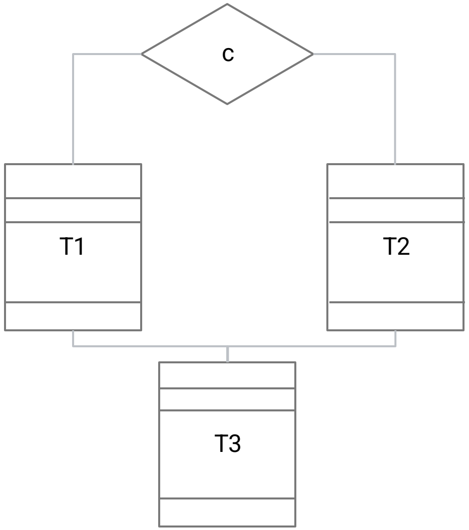


3. Symbolic Execution Merge Points

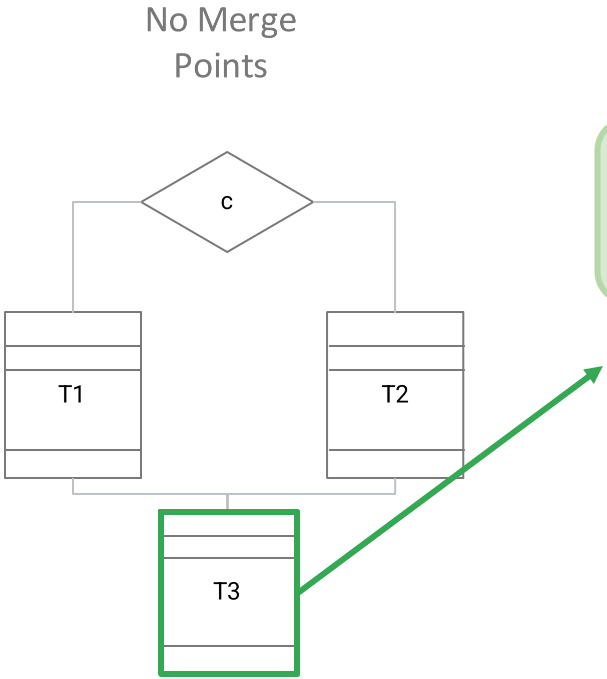
No Merge Points



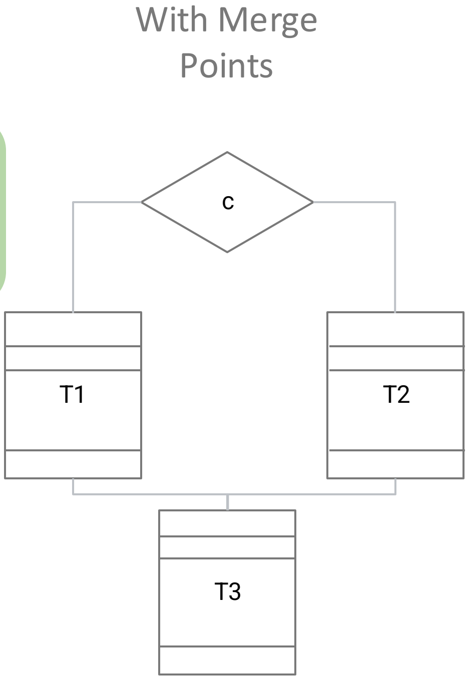
With Merge Points



3. Symbolic Execution Merge Points

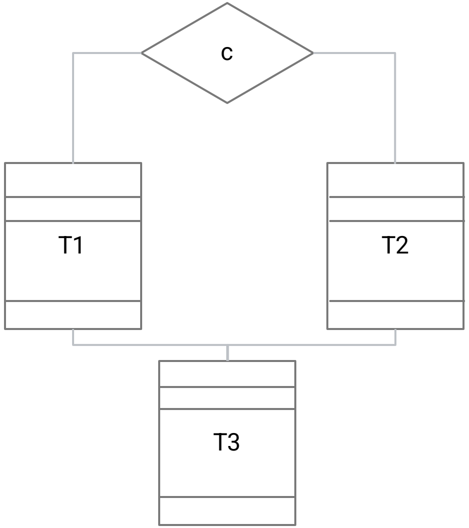


Visited twice!
Larger expressions in state!

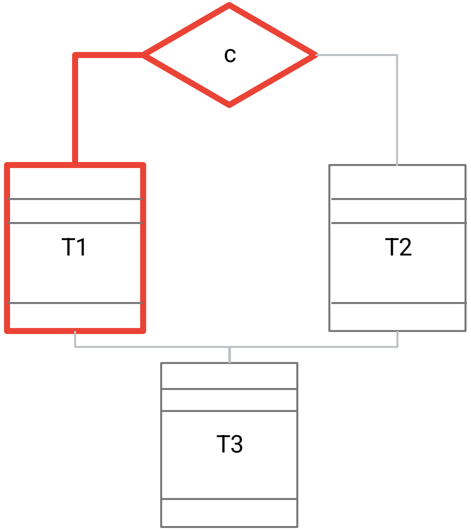


3. Symbolic Execution Merge Points

No Merge Points

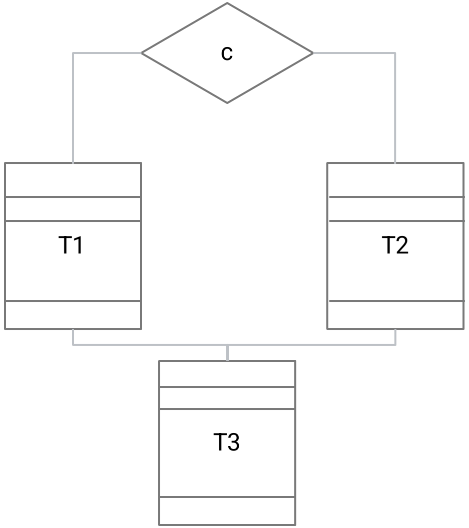


With Merge Points

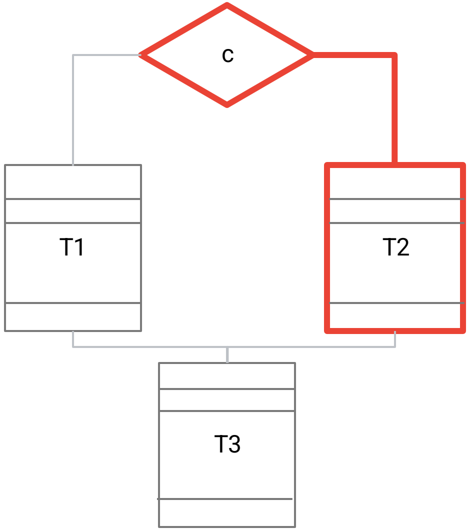


3. Symbolic Execution Merge Points

No Merge
Points

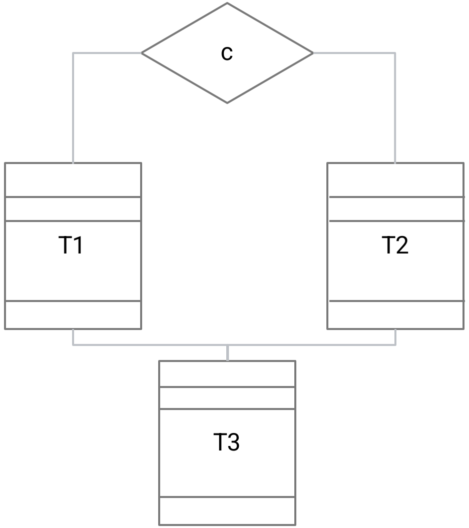


With Merge
Points

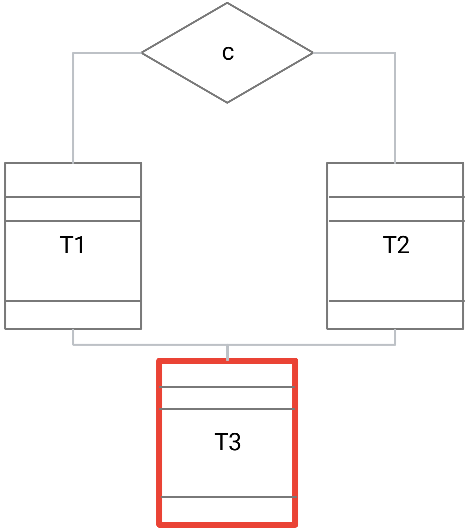


3. Symbolic Execution Merge Points

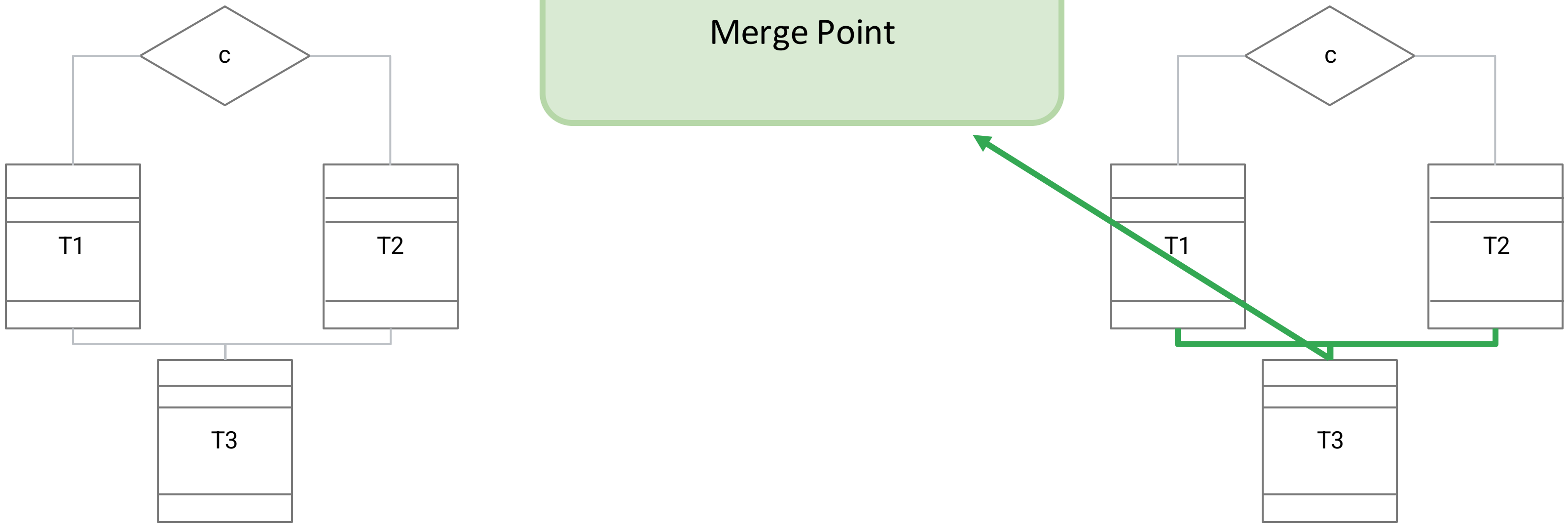
No Merge
Points



With Merge
Points



3. Symbolic Execution Merge Points



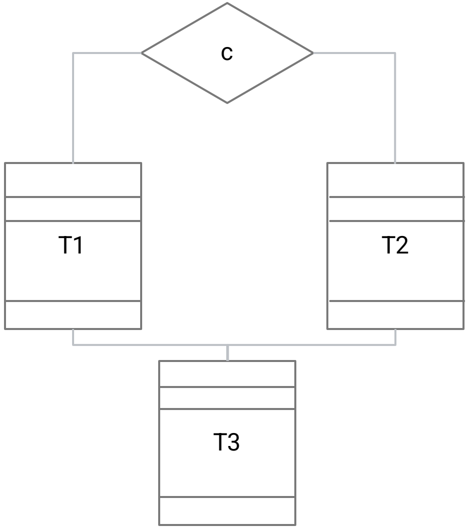
No Merge Points

With Merge Points

Merge Point

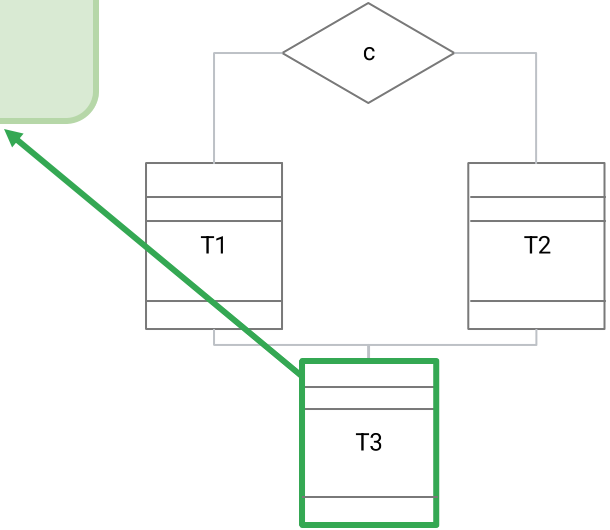
3. Symbolic Execution Merge Points

No Merge Points



Visited only **1** time!
Smaller expressions!

With Merge Points



Results so far

	Product 1		Product 2	
	Clos stage 2	Clos stage 3	Clos stage 2	Clos stage 3
# Packet synthesis requests ≈ {entries}x{packet fate}	~1000	~1000	~1500	~3500
Runtime (before improvements)	~10 mins	~10 mins	~40 mins	**~7 hours ↑** **(17 hours at some point)**
Runtime **(with improvements)** parallelization, merge points	**<5s**	**<5s**	**<30s**	**~1m**

Did we solve the problem?

YES!

but only temporarily! :(

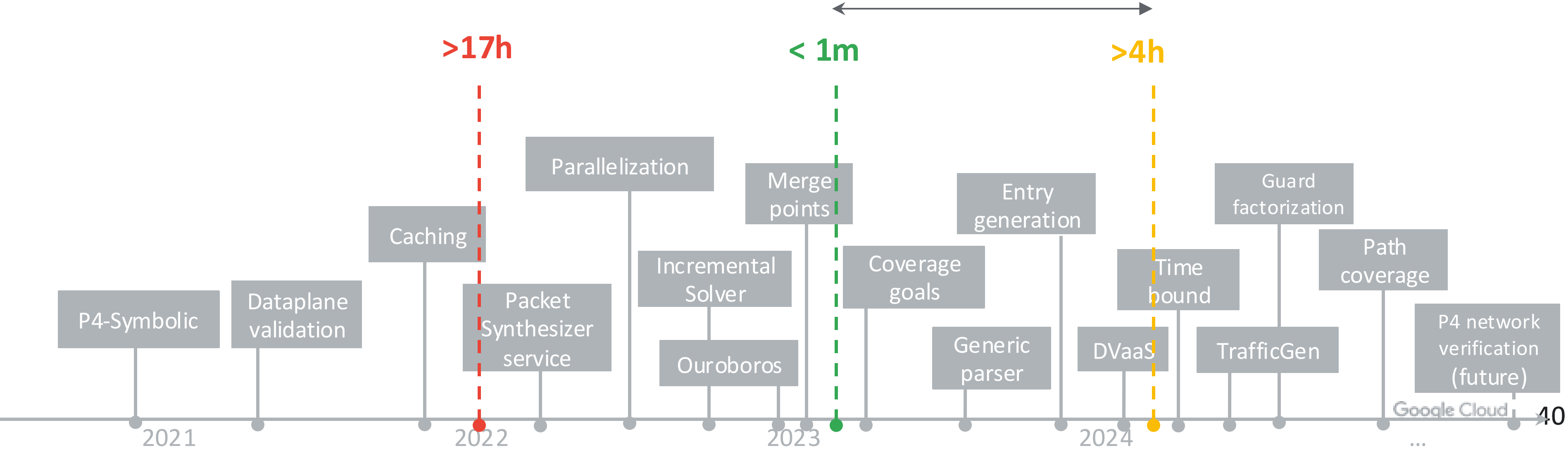
Last resorts

- Reduced coverage
- Re-enabled time-bound synthesis
- Relied on offline synthesis

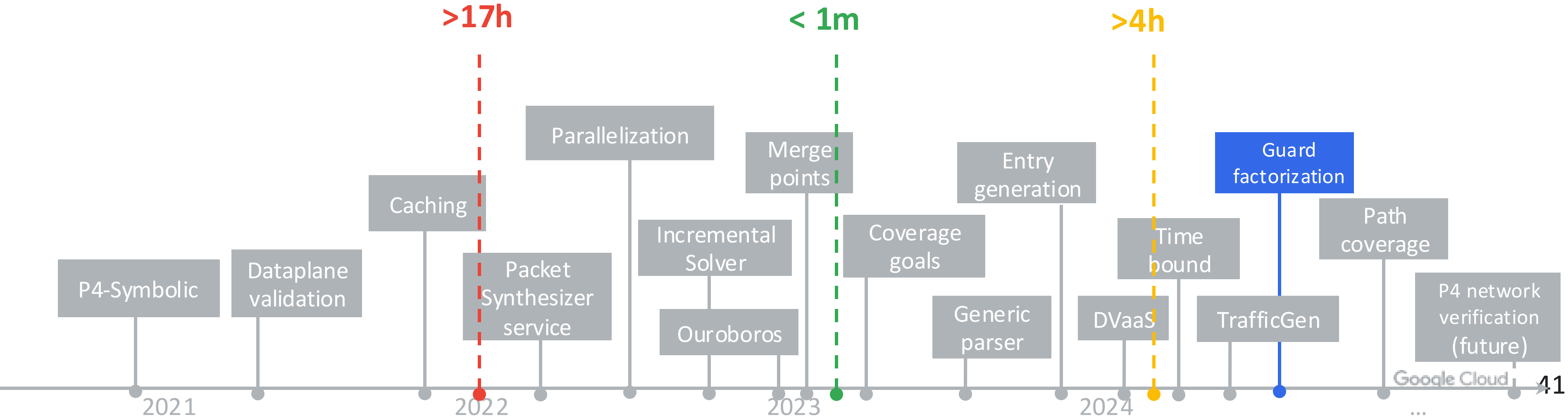
New products and use cases

- More complex pipelines
- Significantly larger (5x) snapshots

More compute (servers) did not help



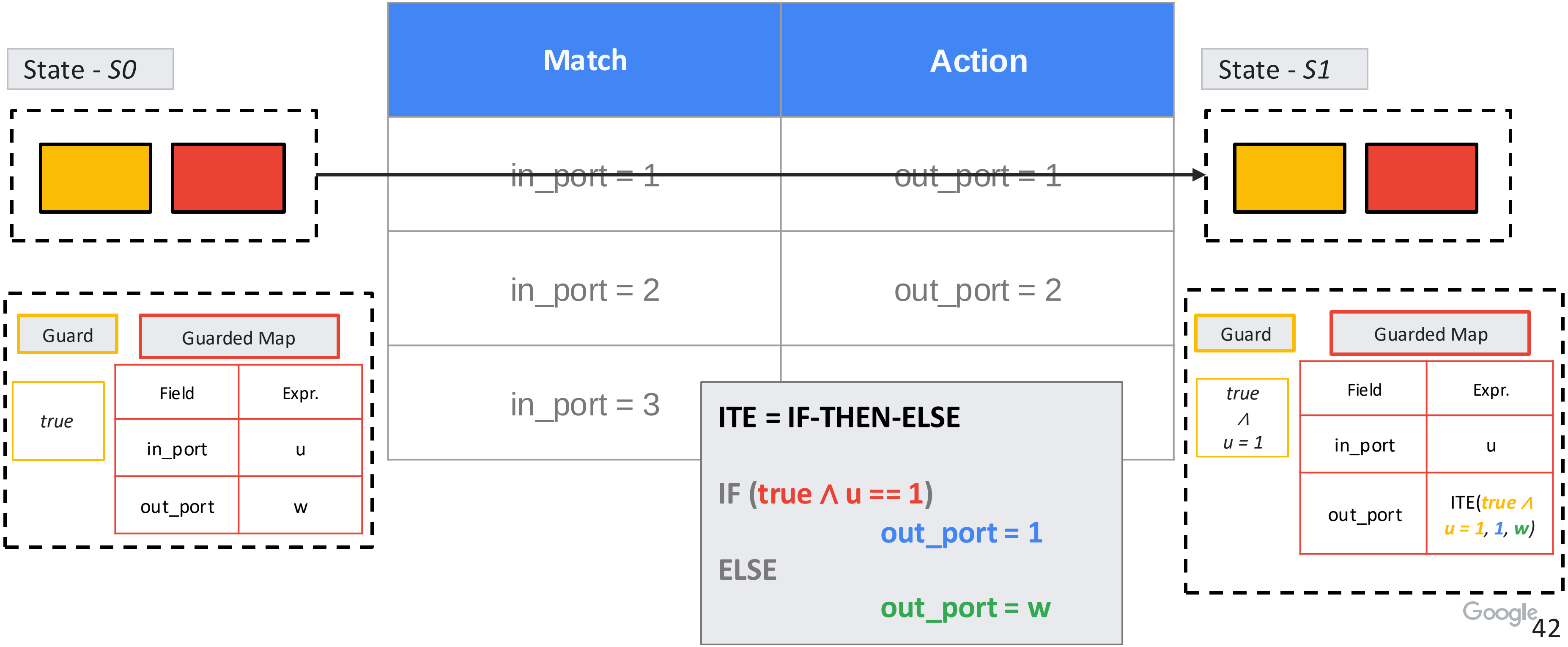
4. Guard Factorization



Google Cloud

4. Guard Factorization

Current Symbolic Execution Flow



State - *S0*

Match	Action
in_port = 1	out_port = 1
in_port = 2	out_port = 2
in_port = 3	

State - *S1*

S0:

Guard	Guarded Map	
	Field	Expr.
true	in_port	u
	out_port	w

S1:

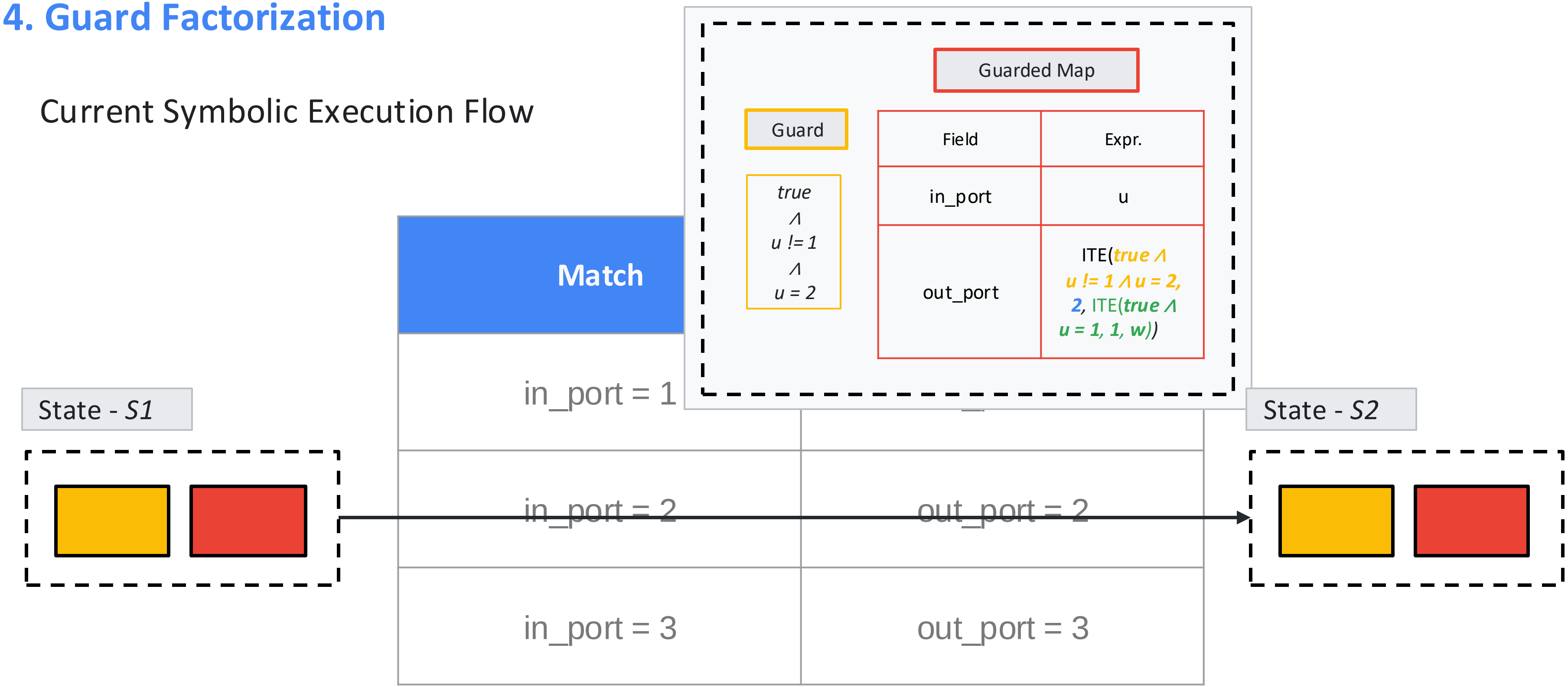
Guard	Guarded Map	
	Field	Expr.
true *∧* *u = 1*	in_port	u
	out_port	ITE(*true ∧ u = 1*, *1*, *w*)

ITE = IF-THEN-ELSE

```
IF (true ∧ u == 1)
            out_port = 1
ELSE
            out_port = w
```

4. Guard Factorization

Current Symbolic Execution Flow



Match	
in_port = 1	
in_port = 2	out_port = 2
in_port = 3	out_port = 3

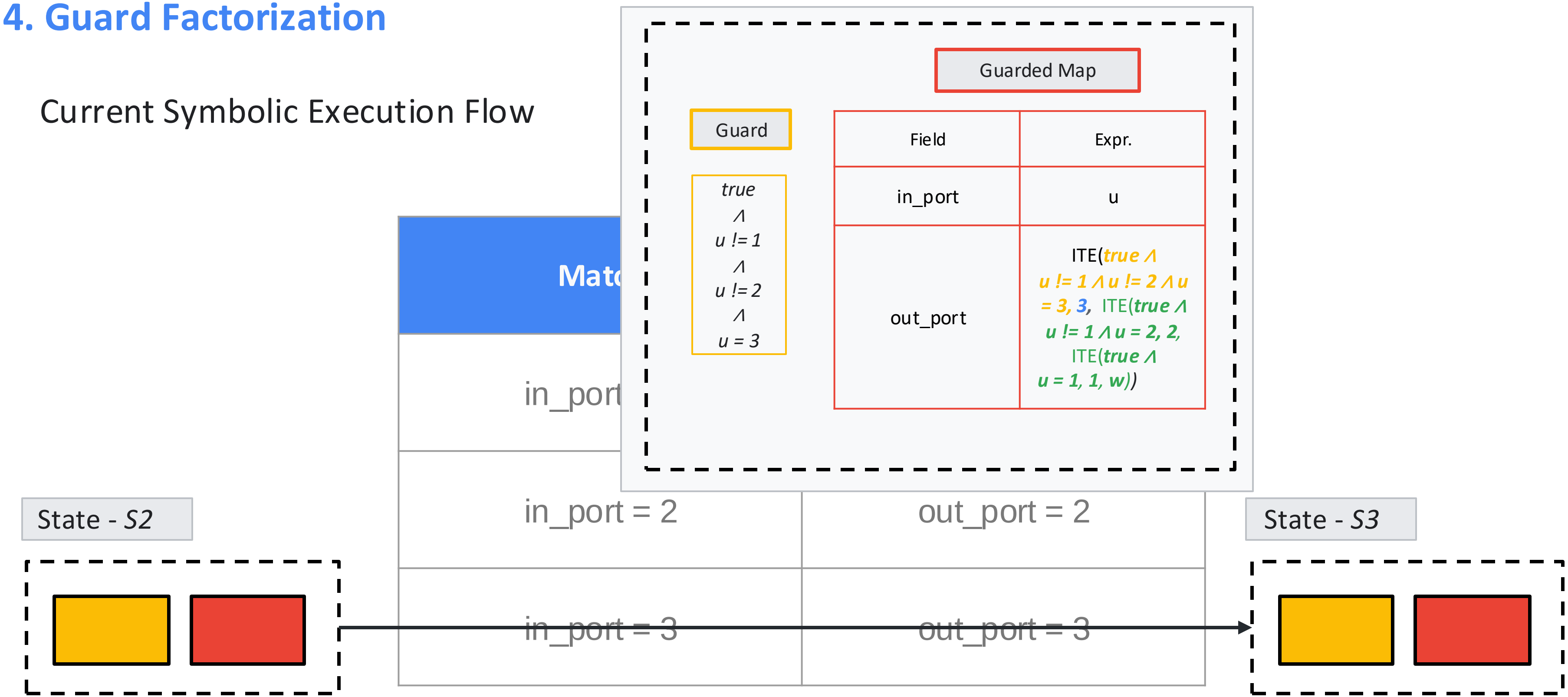
Guarded Map

Guard
true ∧ *u != 1* ∧ *u = 2*

Field	Expr.
in_port	u
out_port	ITE(*true ∧ u != 1 ∧ u = 2*, *2*, ITE(*true ∧ u = 1, 1, w*))

4. Guard Factorization

Current Symbolic Execution Flow



State - *S2*

Guard

true
∧
u != 1
∧
u != 2
∧
u = 3

Guarded Map

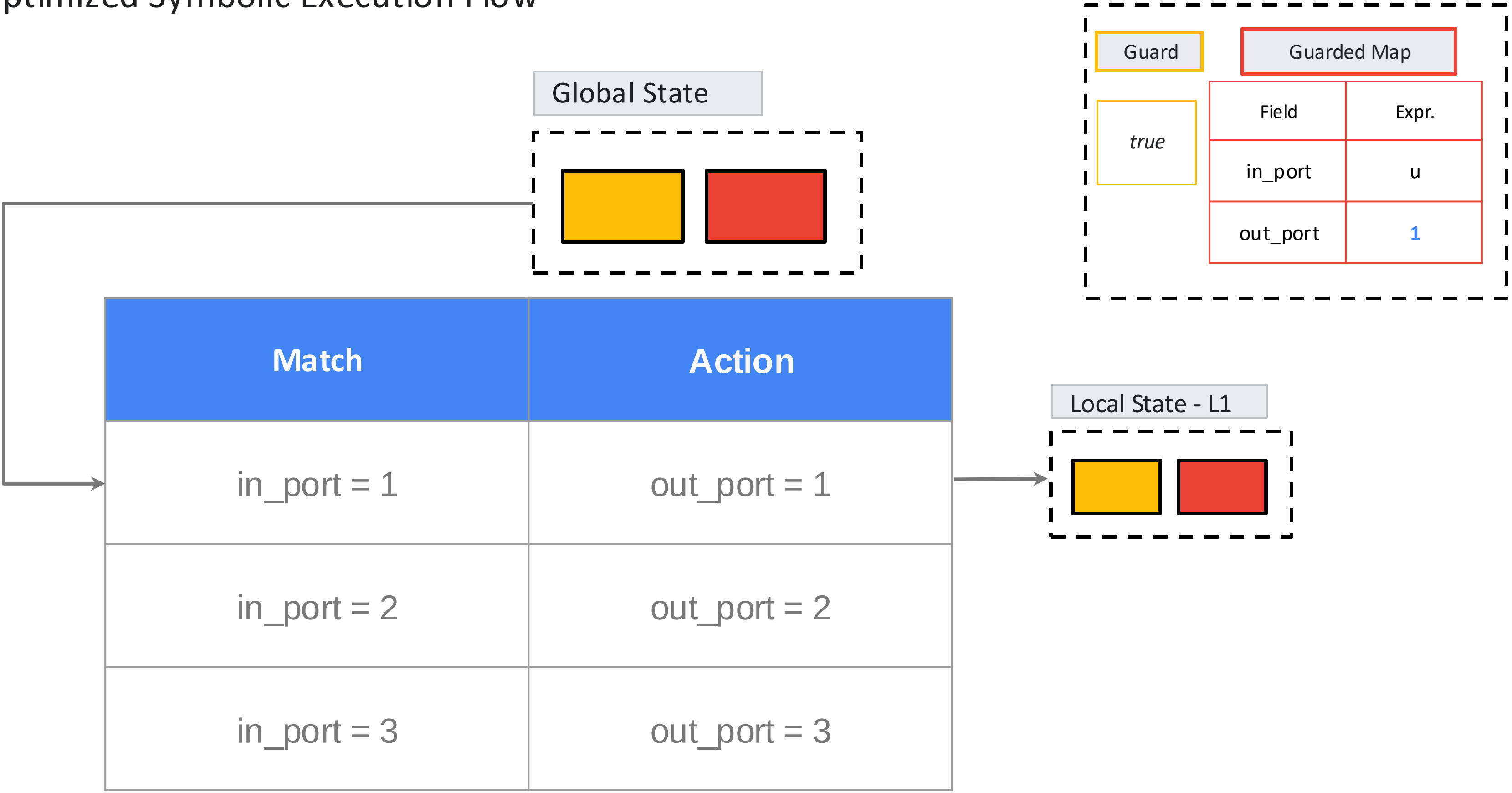
Field	Expr.
in_port	u
out_port	ITE(***true ∧ u != 1 ∧ u != 2 ∧ u = 3, 3***, ITE(***true ∧ u != 1 ∧ u = 2, 2,*** ITE(***true ∧ u = 1, 1, w***))

Matc	
in_port	
in_port = 2	out_port = 2
~~in_port = 3~~	~~out_port = 3~~

State - *S3*

4. Guard Factorization

Optimized Symbolic Execution Flow

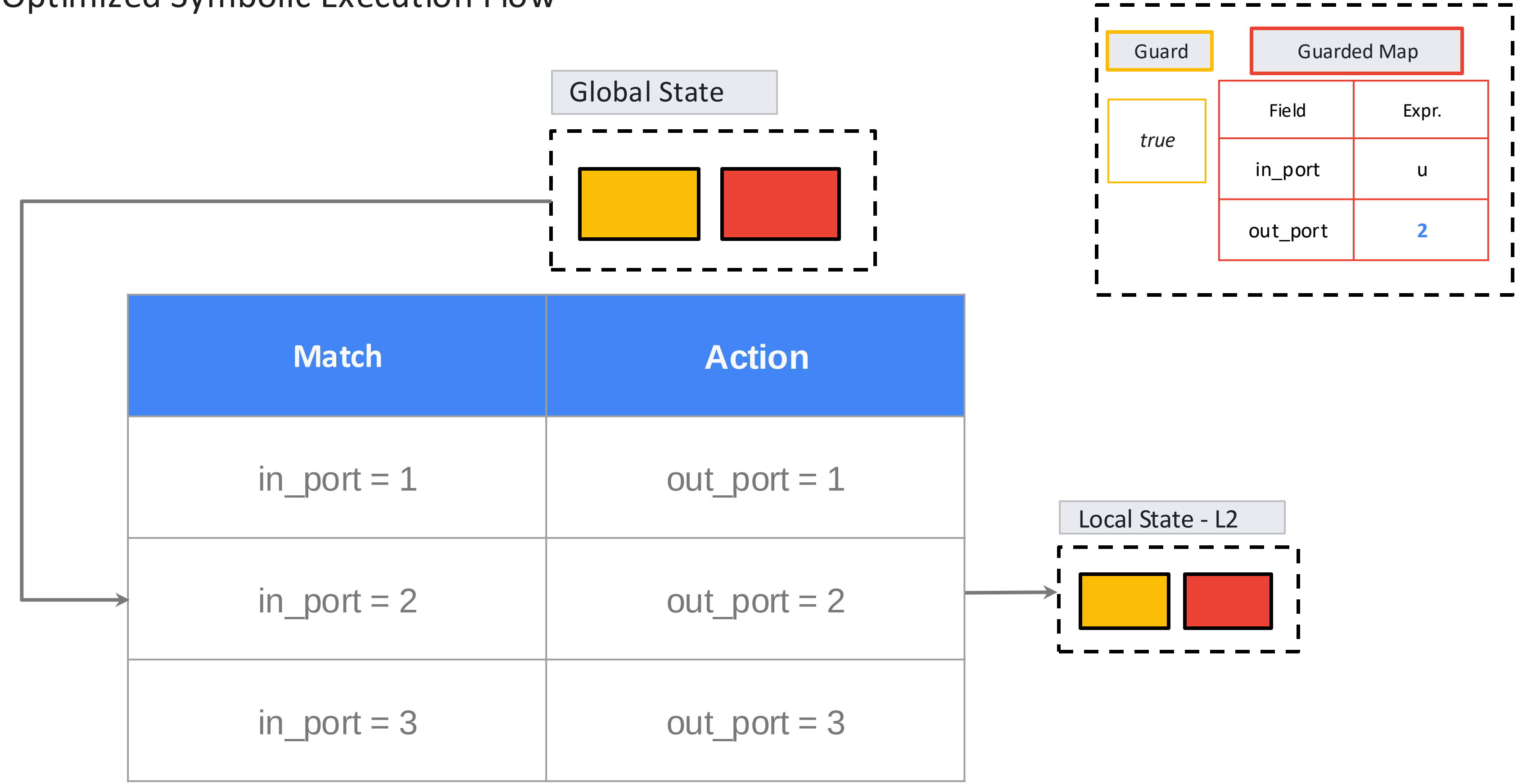


Match	Action
in_port = 1	out_port = 1
in_port = 2	out_port = 2
in_port = 3	out_port = 3

Guard	Field	Expr.
true	in_port	u
	out_port	1

4. Guard Factorization

Optimized Symbolic Execution Flow

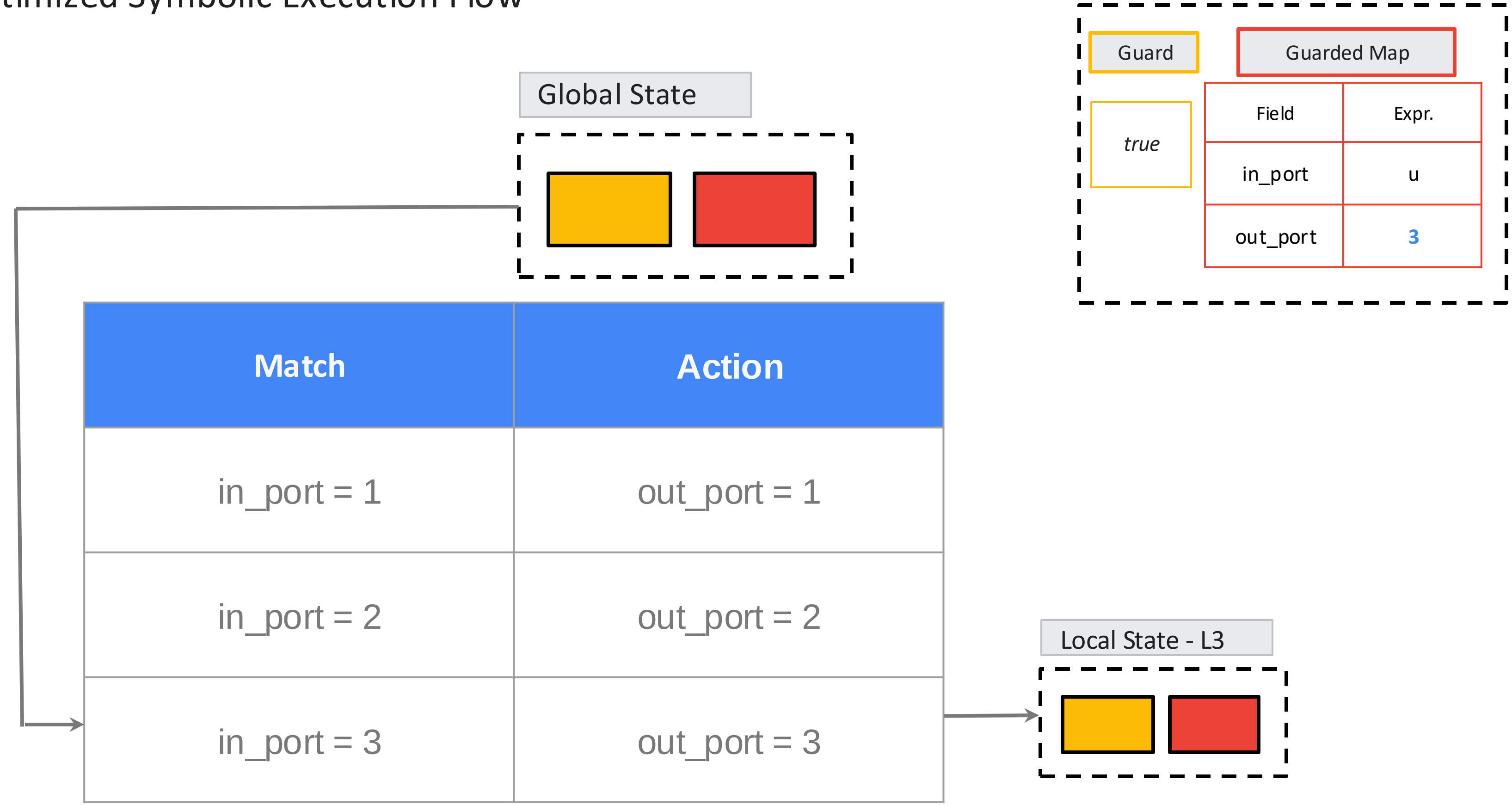


Match	Action
in_port = 1	out_port = 1
in_port = 2	out_port = 2
in_port = 3	out_port = 3

Field	Expr.
in_port	u
out_port	**2**

4. Guard Factorization

Optimized Symbolic Execution Flow



Global State

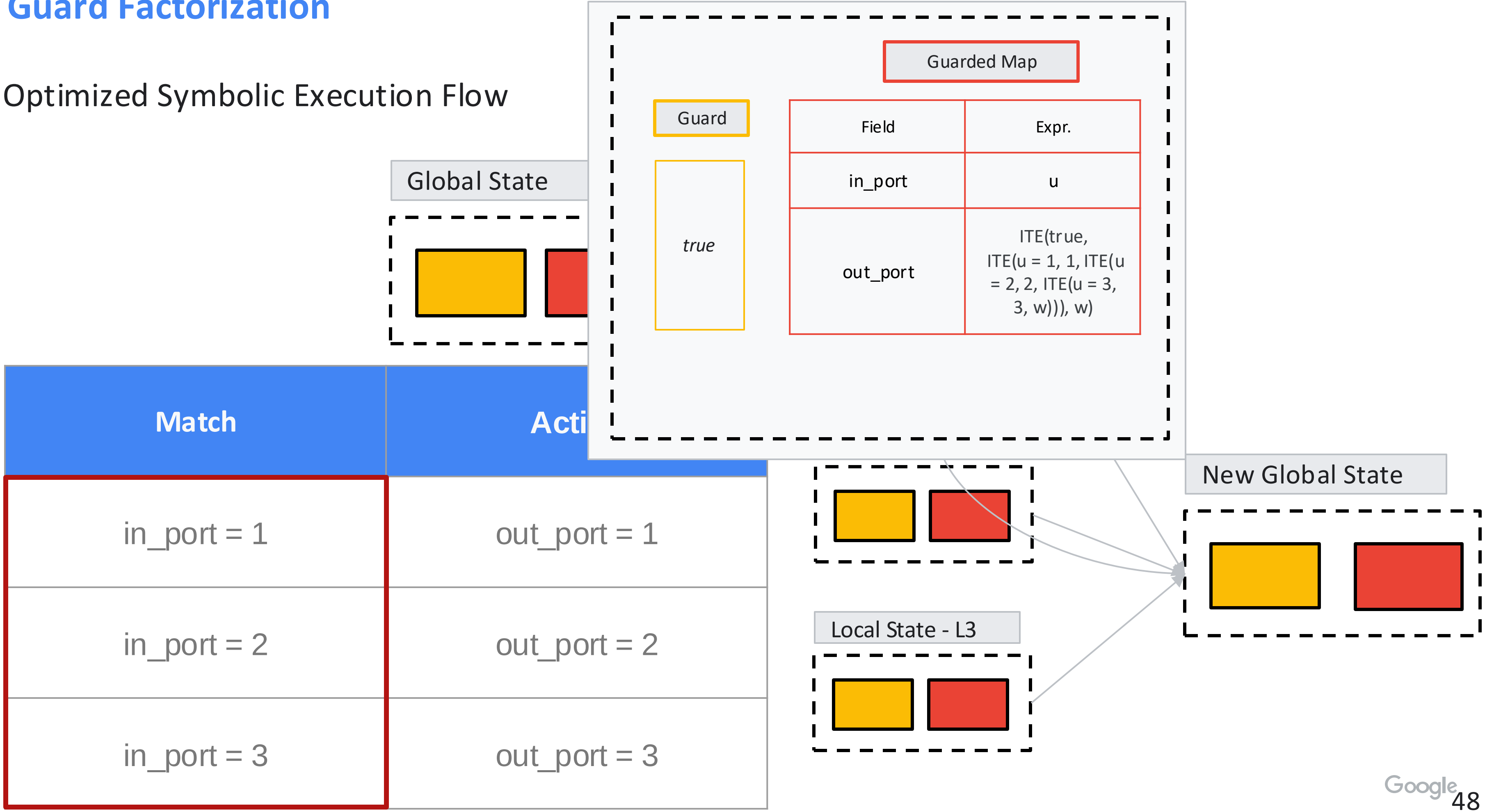
Match	Action
in_port = 1	out_port = 1
in_port = 2	out_port = 2
in_port = 3	out_port = 3

Local State - L3

Guard	Guarded Map	
	Field	Expr.
true	in_port	u
	out_port	**3**

4. Guard Factorization

Optimized Symbolic Execution Flow



Guarded Map

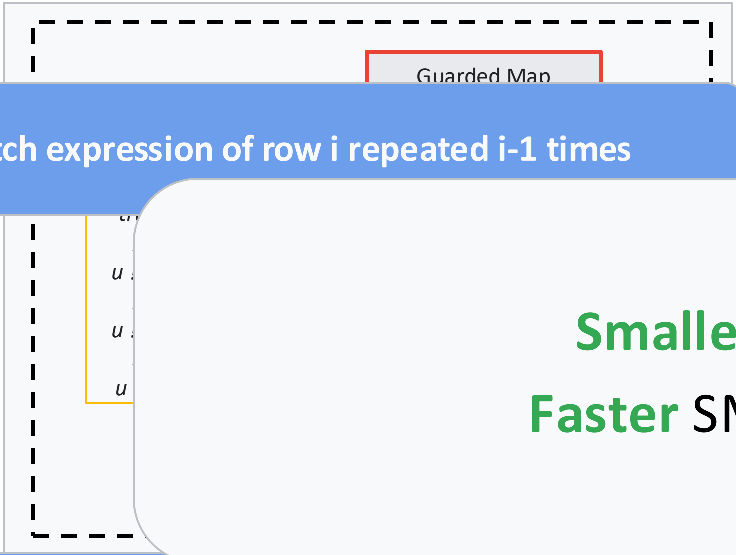
Guard
true

Field	Expr.
in_port	u
out_port	ITE(true, ITE(u = 1, 1, ITE(u = 2, 2, ITE(u = 3, 3, w))), w)

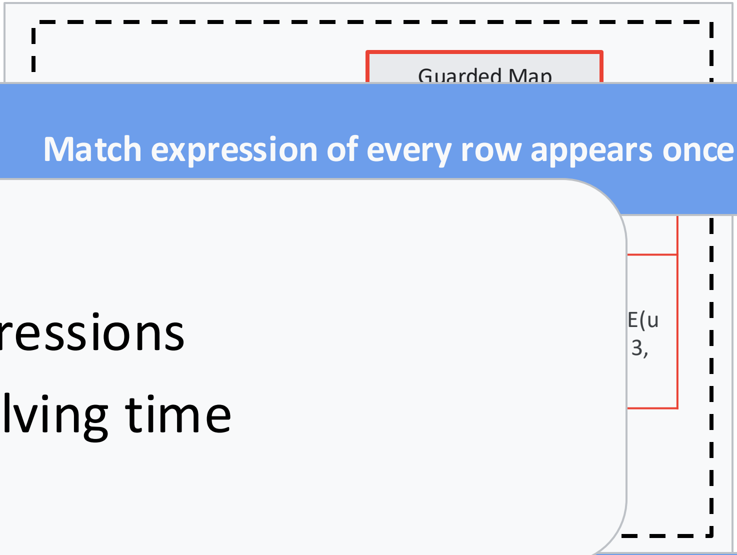
Match	Acti
in_port = 1	out_port = 1
in_port = 2	out_port = 2
in_port = 3	out_port = 3

4. Guard Factorization

Optimized Symbolic Execution Flow



Match expression of row i repeated i-1 times



Match expression of every row appears once

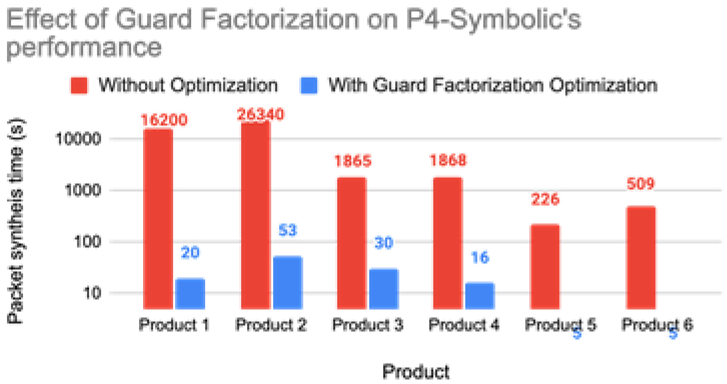
Smaller expressions

Faster SMT solving time

Complexity: O(n^2)

Complexity: O(n)

Results



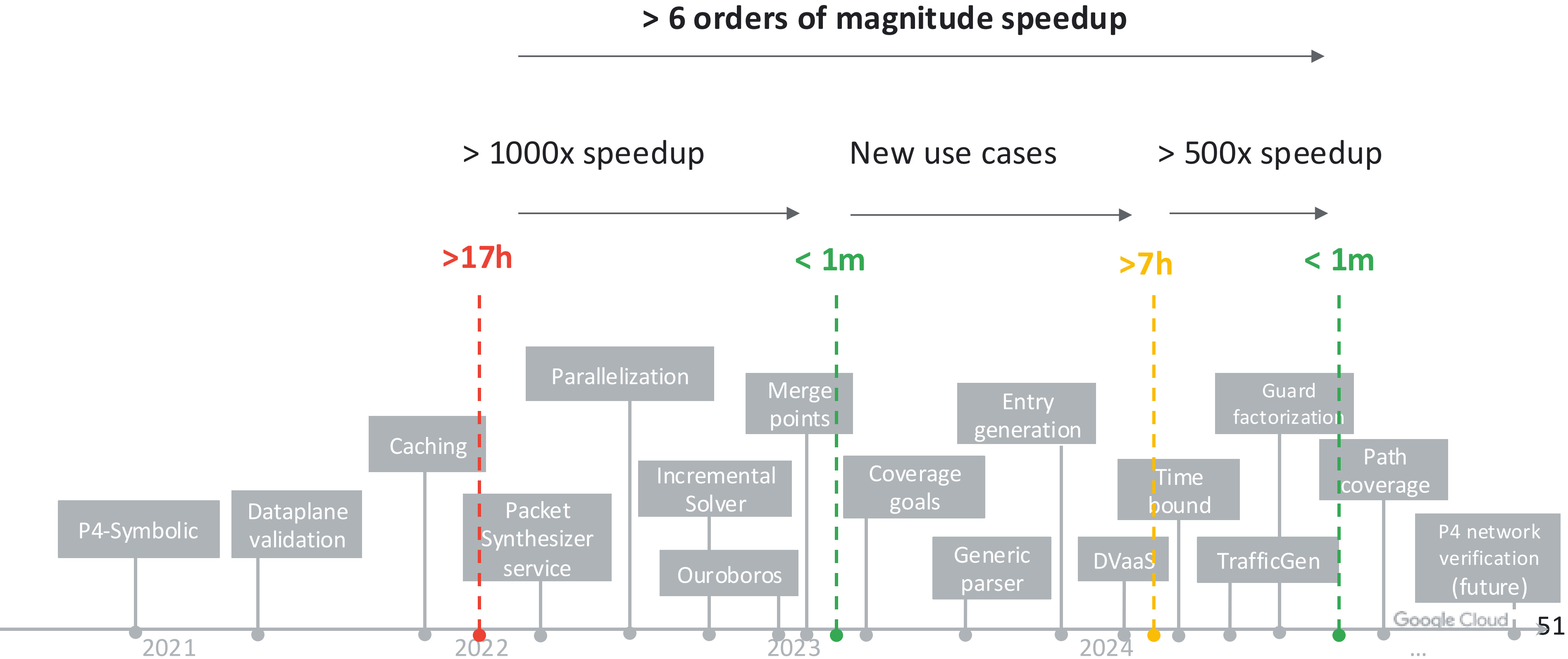
800x speedup!
(2.2hrs -> ~20sec)

Increased developer velocity

Helps testing of the switch better:

- **Test with larger snapshots**
- **Expand coverage goals:** Re-enable more coverage goals (e.g. header coverage)

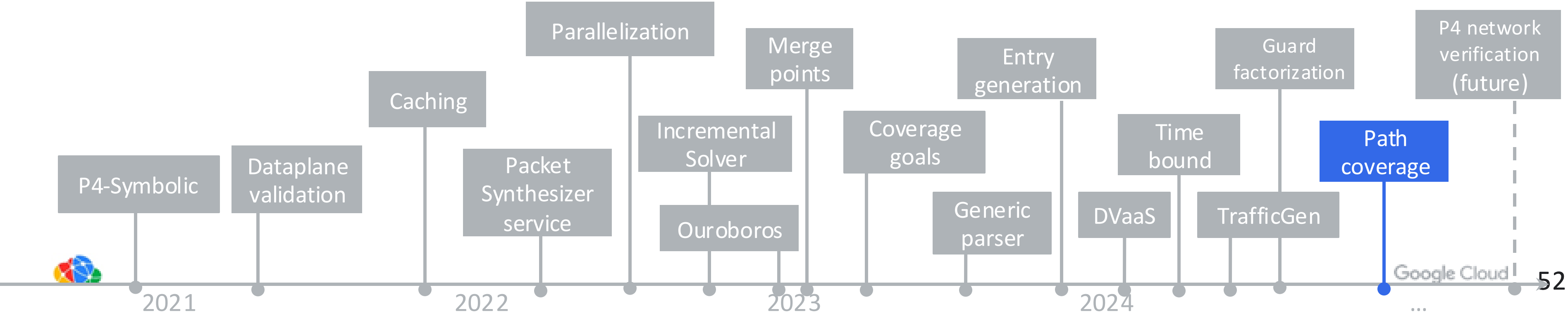
Current status



Google Cloud

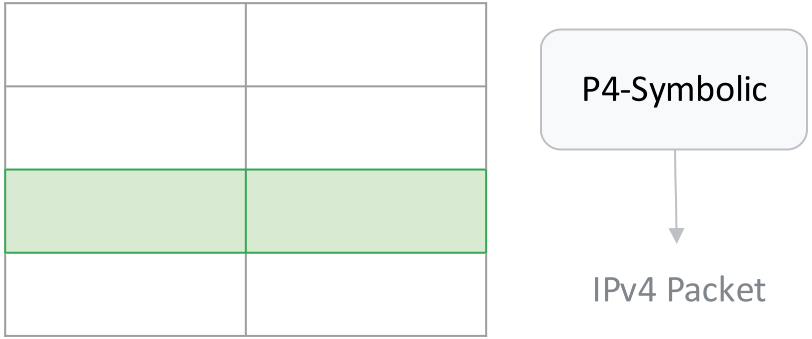
Outline

- Background and Context
- P4-Symbolic
- Performance Improvements
- Coverage Improvements
- Future



Coverage caveat

ACL Table



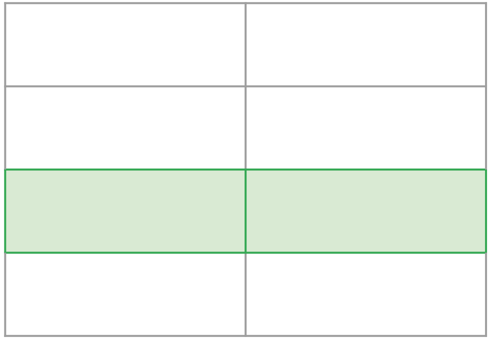
P4-Symbolic

IPv4 Packet

Coverage Goal:
“hitting all table entries”

Coverage caveat

ACL Table



Coverage Goal:
“hitting all table entries”

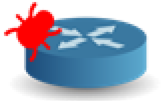
P4-Symbolic

IPv4 Packet

Production Scenario

ACL Table

IPv6 Packet



Failure

Could miss the bug!

Nearly missed bug

Good packet: “IPv4 packet hitting ACL table”

Bad packet: “IPv6 packet hitting ACL table”

Coverage

```
for e in entries:
    generate a packet hitting e
```

```
for e in entries:
    for h in headers:
        generate a packet containing . header h and hitting e
```

Solution 1: Manually expand coverage goals!

Add entry coverage, header coverage, ….. , and so on

- Cannot cover every case
- Very complex coverage goals -> more requests -> more time to solve -> slower packet synthesis

Solution 2: Path Coverage! *(Ultimate coverage)*

Path Coverage

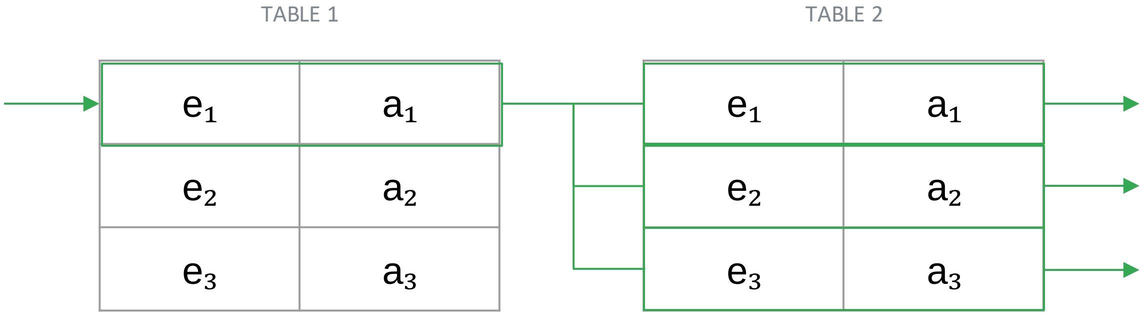


TABLE 1

e_1	a_1
e_2	a_2
e_3	a_3

TABLE 2

e_1	a_1
e_2	a_2
e_3	a_3

Path Coverage

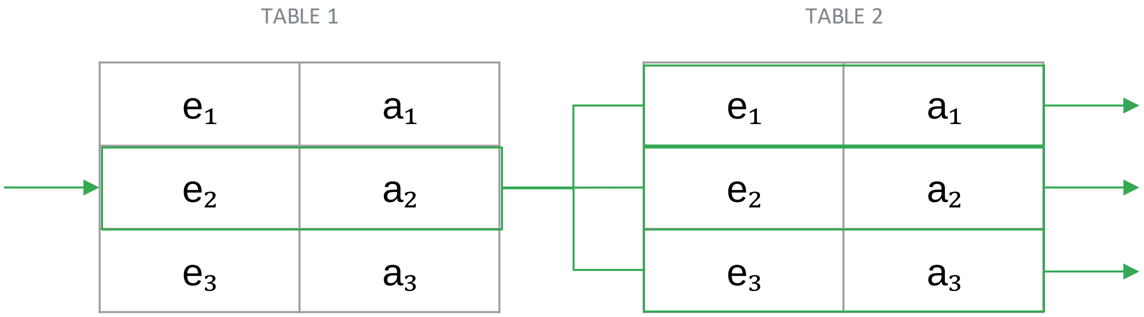


TABLE 1

e_1	a_1
e_2	a_2
e_3	a_3

TABLE 2

e_1	a_1
e_2	a_2
e_3	a_3

Path Coverage

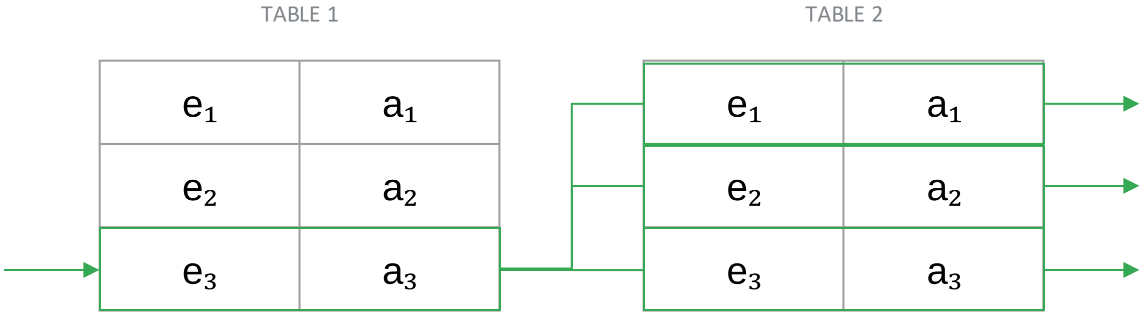


TABLE 1

e_1	a_1
e_2	a_2
e_3	a_3

TABLE 2

e_1	a_1
e_2	a_2
e_3	a_3

TOTAL PATHS COVERED: 9

Every possible scenario of a packet flow tested!

Path Coverage

Table 1: 1000 entries

Table 2: 1000 entries

Table 3: 1000 entries

Total Paths: **10^9 = 1B**

Covering all paths is **exponential**!

PROBLEM: **PATH EXPLOSION!**

Is there hope? - Yes

- Observation: Not all paths are valid
- Prune paths as you go!

Path Pruning

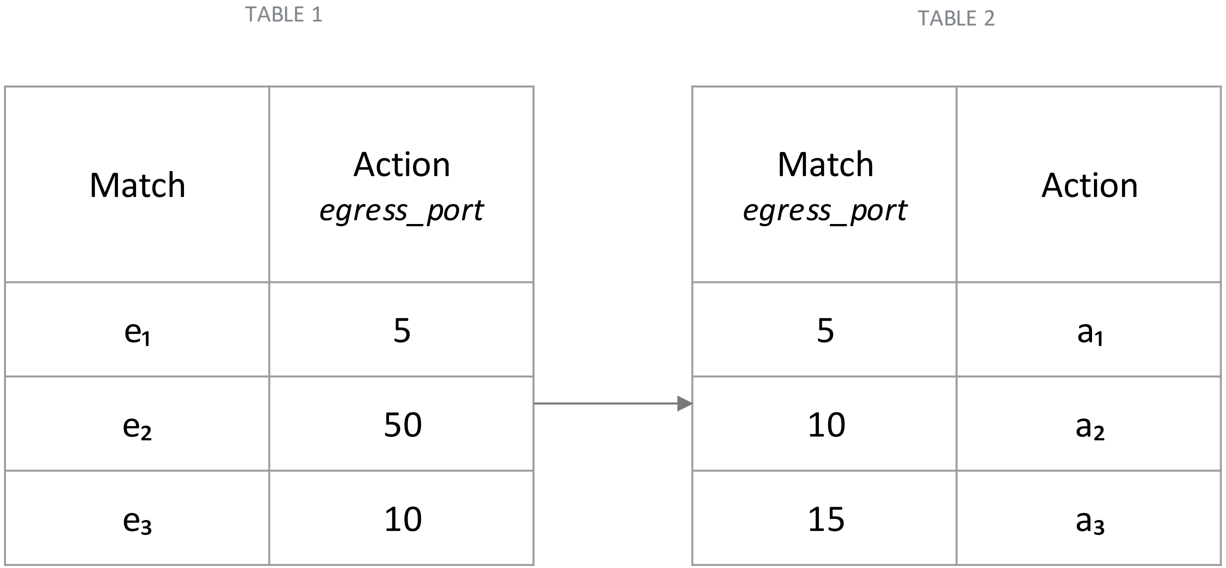


TABLE 1

Match	Action *egress_port*
e_1	5
e_2	50
e_3	10

TABLE 2

Match *egress_port*	Action
5	a_1
10	a_2
15	a_3

Path Pruning

✓ Valid Path

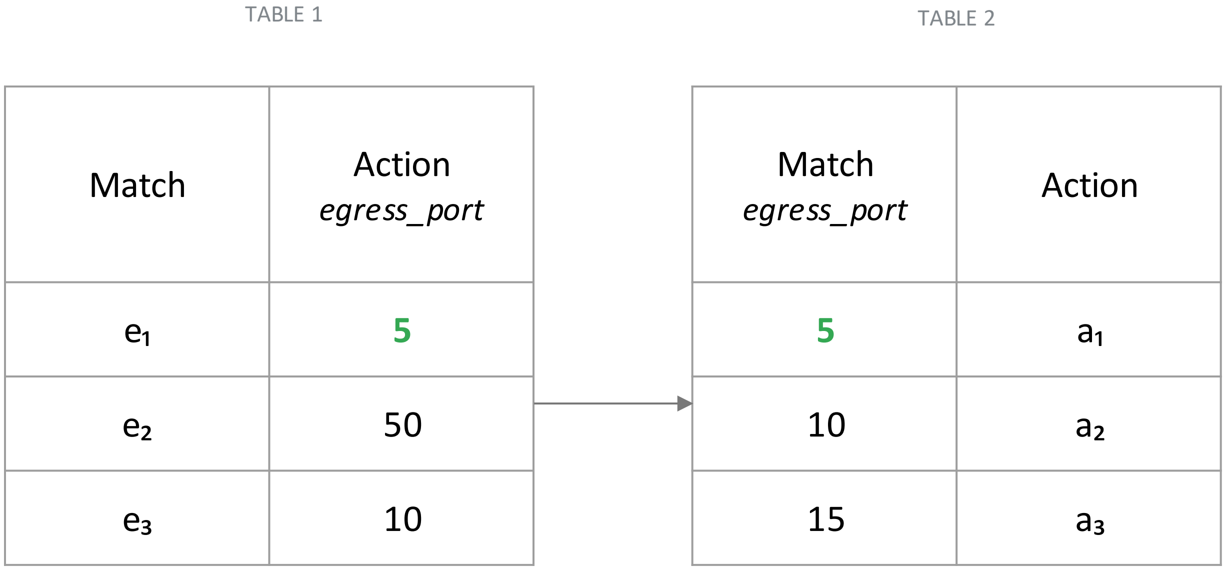


TABLE 1

Match	Action *egress_port*
e_1	**5**
e_2	50
e_3	10

TABLE 2

Match *egress_port*	Action
5	a_1
10	a_2
15	a_3

**Valid Path -> An actual packet would take this path*

Path Pruning

Invalid Path

TABLE 1

Match	Action *egress_port*
e_1	**5**
e_2	50
e_3	10

TABLE 2

Match *egress_port*	Action
5	a_1
10	a_2

TOTAL PATHS: 9

VALID PATHS: 2

**Invalid Path -> An actual packet would not take this path*

Initial Results

Switch: Product 1 Clos Stage 2

#Paths: > 10^14

#Valid Paths: ~2M (~10^6)

Time taken: 8hrs

Initial Results

Switch: Product 1 Clos Stage 2

#Paths: > 10^14

#Valid Paths: ~2M (~10^6)

1 representative packet per valid path
#Packets Synthesized = ~2M

Time taken: 8hrs

Can we do better?

Observation: Lesser calls to solver => faster execution times.

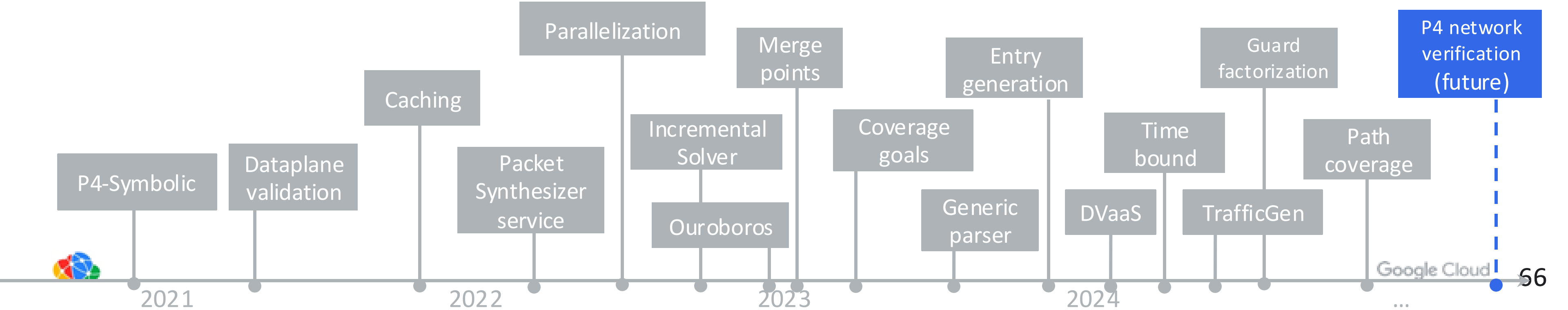
- Only 12.9% of calls to solver are satisfiable

Can we make fewer calls to solver somehow?

We plan to explore ideas from literature that address this problem

Outline

- Background and Context
- P4-Symbolic
- Performance Improvements
- Coverage Improvements
- Future



P4-Symbolic

Dataplane validation

Caching

Packet Synthesizer service

Parallelization

Incremental Solver

Ouroboros

Merge points

Coverage goals

Generic parser

Entry generation

DVaaS

Time bound

TrafficGen

Guard factorization

Path coverage

P4 network verification (future)

2021

2022

2023

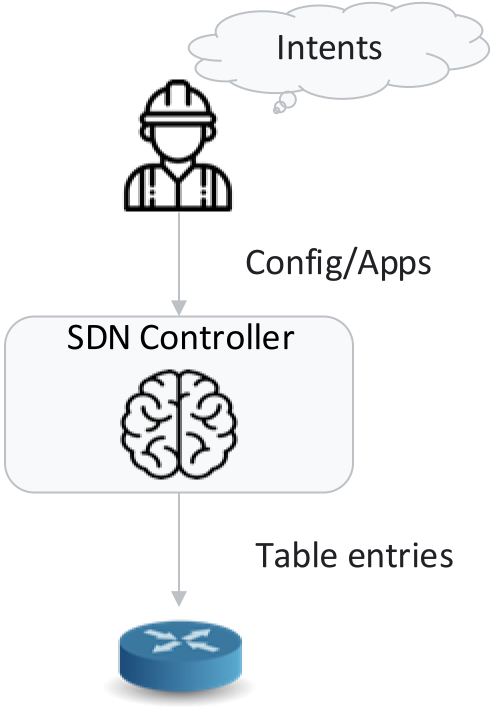
2024

...

Overall goal: Ensure network works as intended

Subgoal 1: Ensure controller produces correct table entries (according to intents)

Subgoal 2: Ensure switch works as expected (according to table entries)



Network Verification

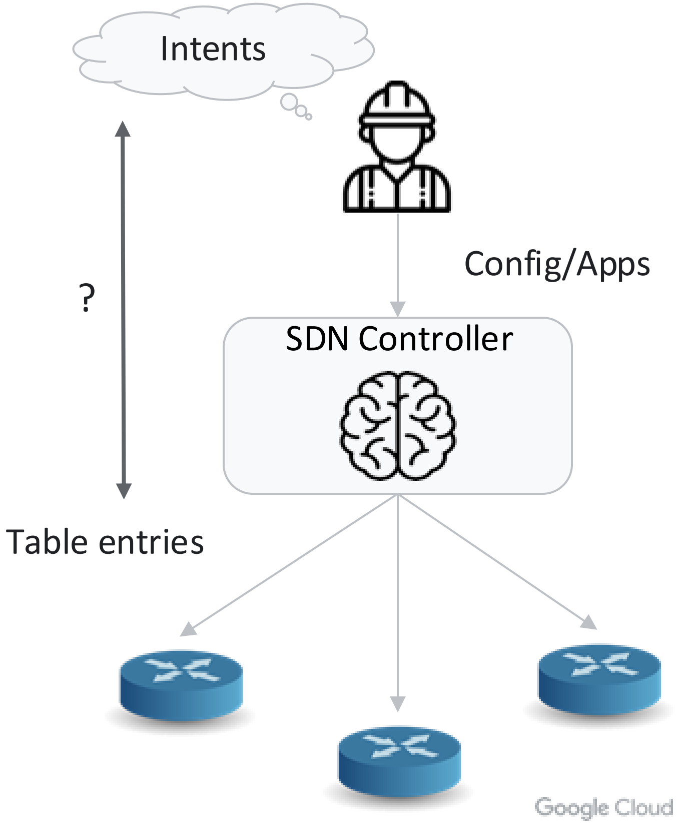
Existing system

- Hardcoded, incomplete model of switch
  - No guarantees on fidelity
  - Hard to evolve

In essence: symbolic execution at network level

Idea

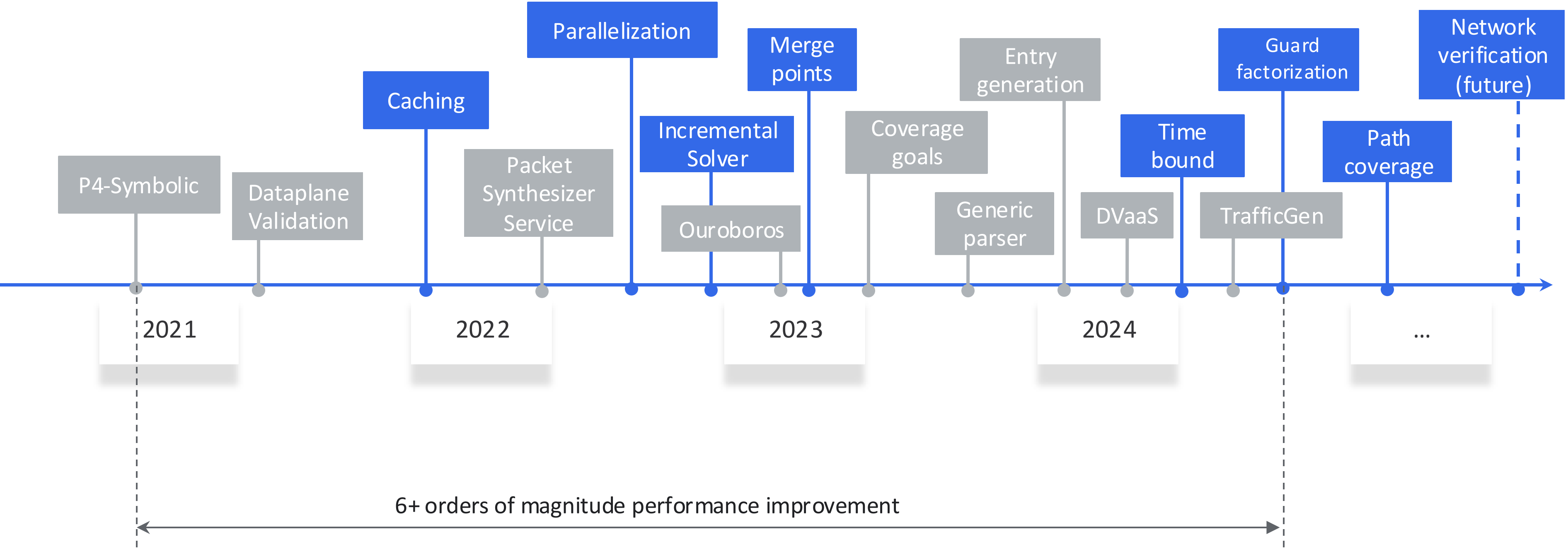
- Extend P4 based symbolic execution to network level
  - Guaranteed high fidelity
    - Due to dataplane validation
  - Effortless evolution



Thank you.

Google Cloud

Milestones and highlights



6+ orders of magnitude performance improvement

Relevant work

- HSA (NSDI'12), APV (ICNP'13), ddNF (HVC'16), #PEC (ICNP'19), etc.
  - Domain optimized "solvers" for network verification
  - Better performance, but more limitations (e.g. in packet rewrites)
- P4-Symbolic
  - Generic SMT solver (Z3)
  - More flexibility
  - Less performant
    - Good enough (for now)
    - Can employ ideas from above if needed