Equality Saturation: Term Extraction and an Application to Network Synthesis

General Exam: Deyuan (Mike) He

April 17, 2024

Examination Committee

Prof.Aarti Gupta (Advisor)   Prof.Andrew Appel   Prof.Mae Milano

Outline

1. Brief introduction to equality saturation
2. Term Extraction for equality saturation **(Part A)**
3. Applying equality saturation for network resource synthesis **(Part B)**
4. (If time permits) Ongoing project of invariant synthesis for distributed systems

Compiler optimizations are hard to design

Vectorize loop

Inlining

Code motion

Load store forwarding

Constant folding

Compiler optimizations are hard to design

Constant folding	Vectorize loop	Load store forwarding
Vectorize loop	Inlining	Code motion
Inlining	Constant folding	Constant folding
Code motion	Load store forwarding	Vectorize loop
Load store forwarding	Code motion	Inlining

Which order to choose?

Phase Ordering Problem

Compiler optimizations are hard to design

GCC's *passes.def*

500+ LoC to define the order

https://github.com/gcc-mirror/gcc/blob/master/gcc/passes.def

Compiler optimizations are hard to design

Observation: program transformations are *destructive*

$?V \times 2$

$(X \times 2) \div 2$

$?V \times 2 \rightarrow ?V << 1$

$?V \mapsto X$

$(X << 1) \div 2$

$(?X \times ?Y) \div ?Z \rightarrow ?X \times (?Y \div ?Z)$

$?X \div ?X \rightarrow 1$

$?X \times 1 \rightarrow ?X$

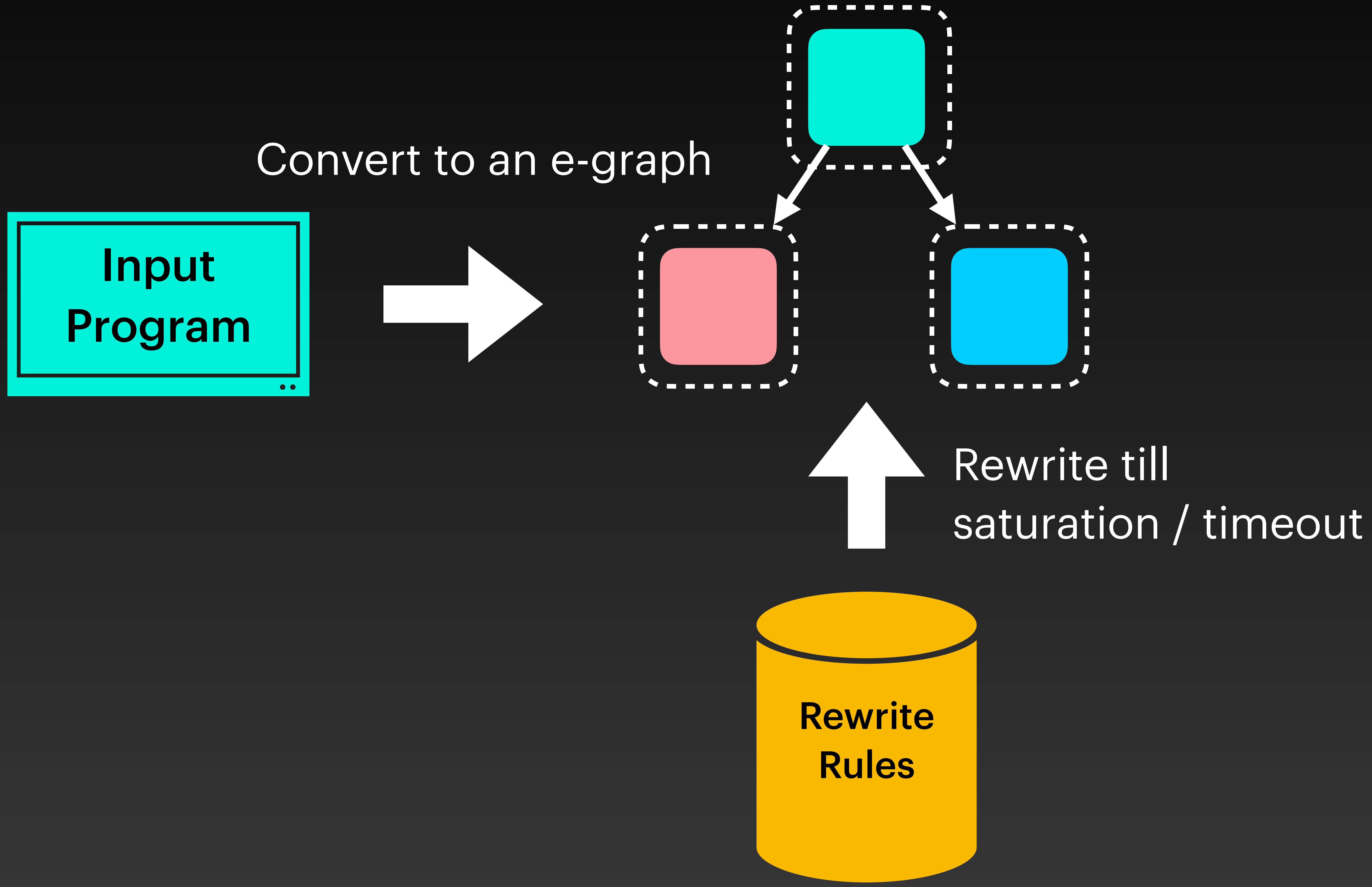
$(X \times 2) \div 2$

Equality Saturation

Non-destructive rewriting

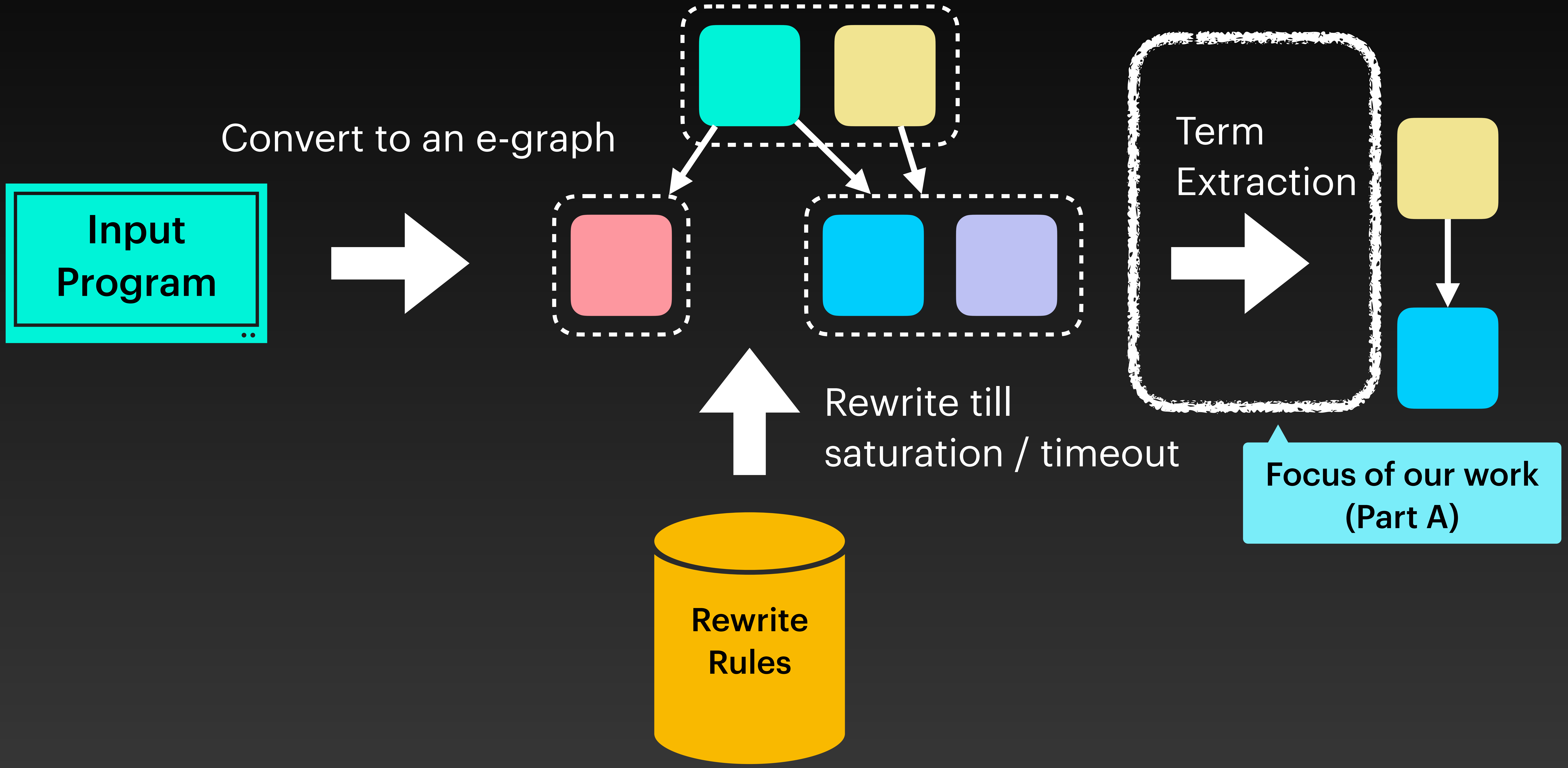
Tate, R., et al, "Equality Saturation: a New Approach to Optimization," in *Logical Methods in Computer Science*, 2011.
Willsey, M., et al. "egg: Fast and extensible equality saturation," in *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–29, 2021.

Equality Saturation



Tate, R., et al, "Equality Saturation: a New Approach to Optimization," in *Logical Methods in Computer Science*, 2011.
Willsey, M., et al. "egg: Fast and extensible equality saturation," in *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–29, 2021.

Equality Saturation

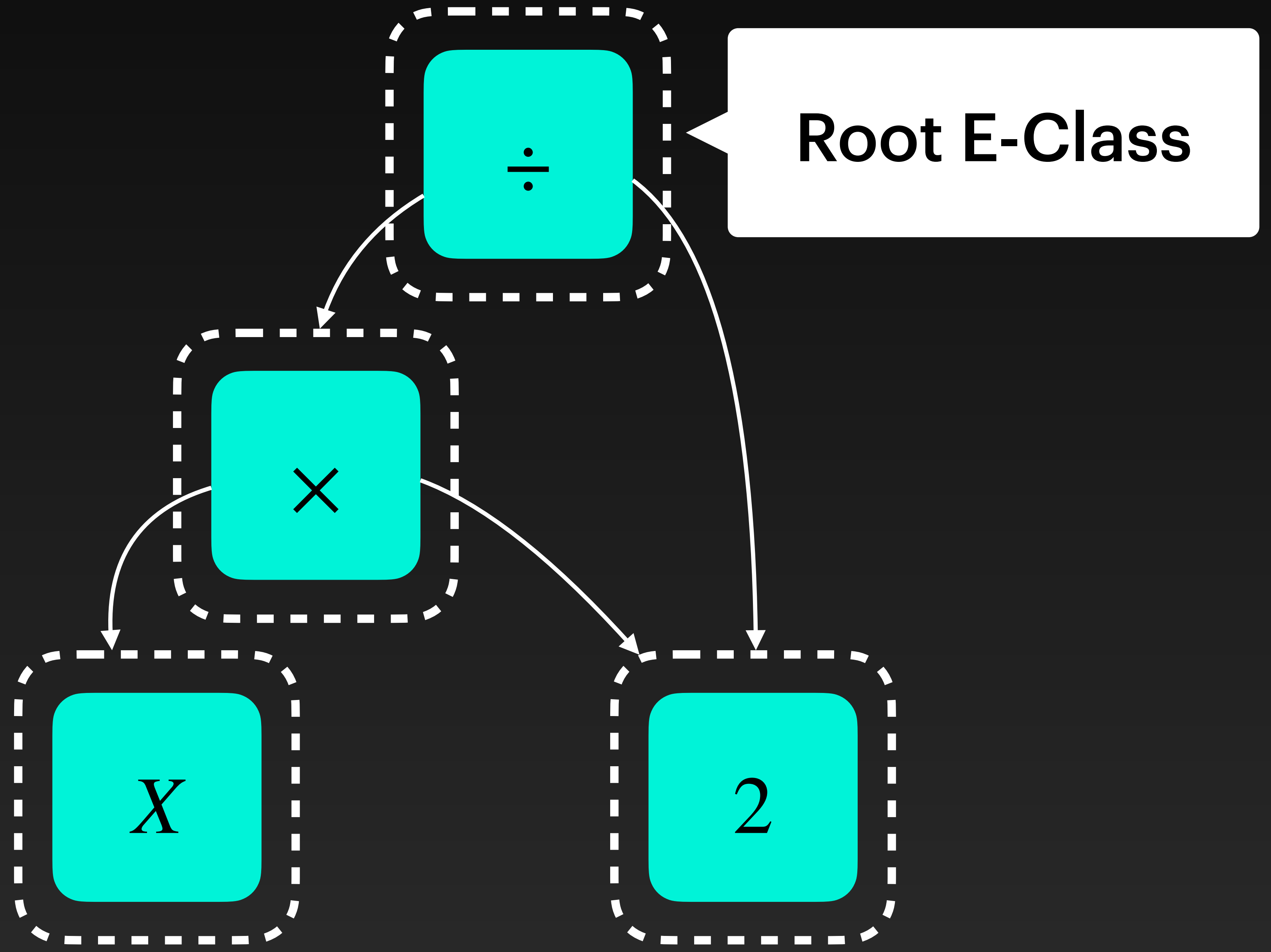


Tate, R., et al, "Equality Saturation: a New Approach to Optimization," in *Logical Methods in Computer Science*, 2011.
Willsey, M., et al. "egg: Fast and extensible equality saturation," in *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–29, 2021.

Equality Saturation and E-Graphs

Converting terms to E-Graphs

$(X \times 2) \div 2$



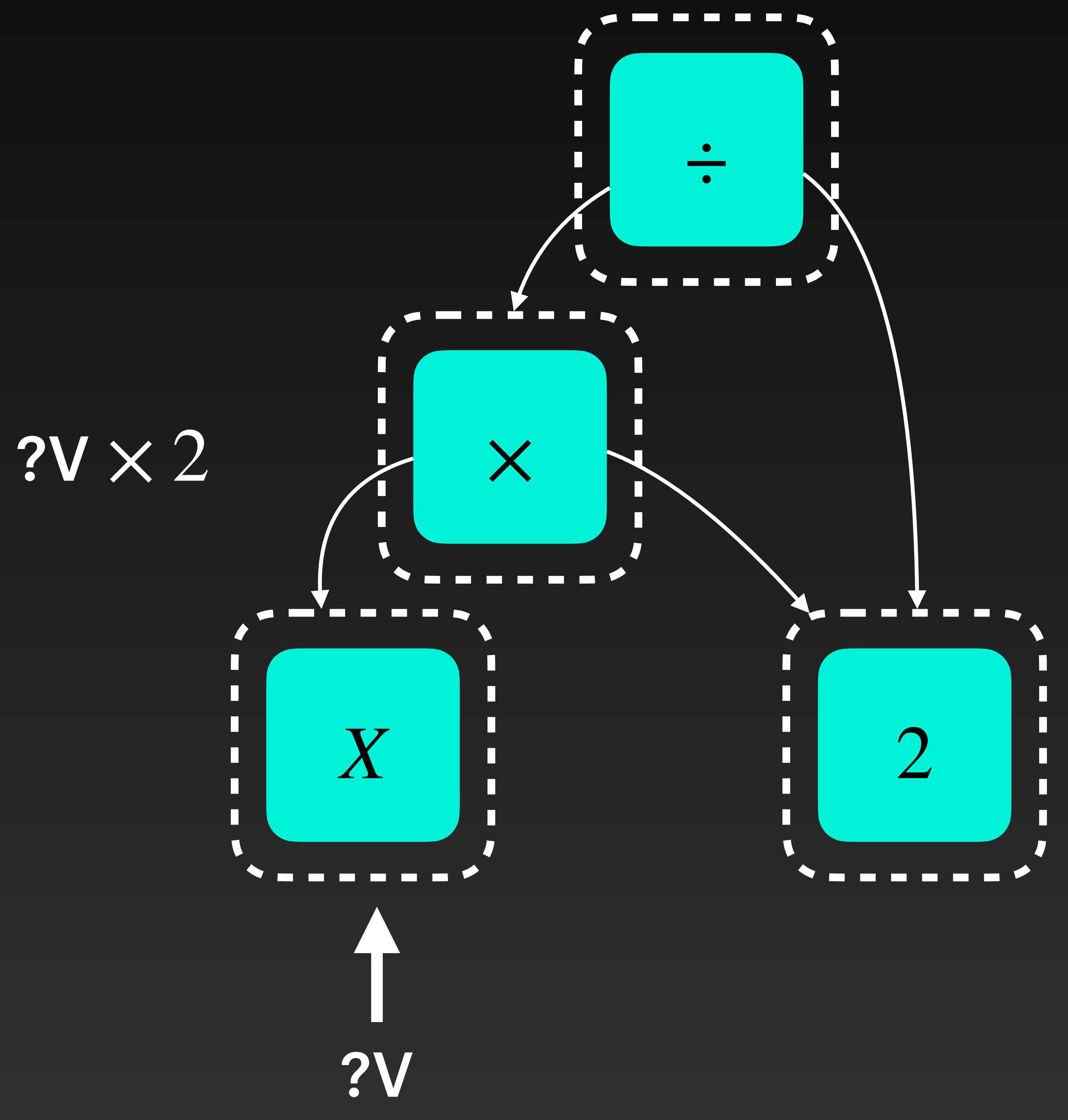
E-Nodes: Function symbols, literals

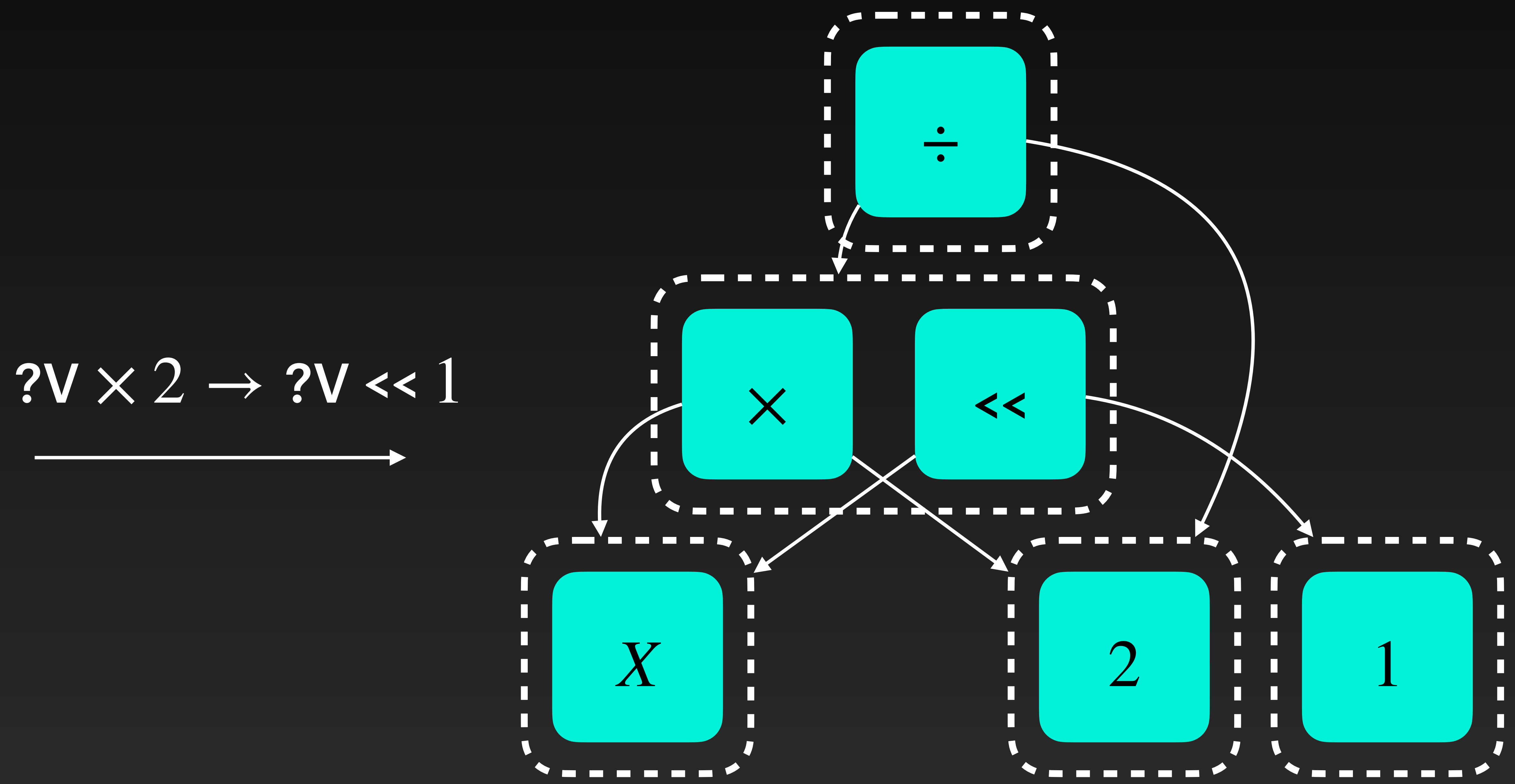
E-Classes: Equivalent terms

Tate, R., et al, "Equality Saturation: a New Approach to Optimization," in *Logical Methods in Computer Science*, 2011.
Willsey, M., et al. "egg: Fast and extensible equality saturation," in *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–29, 2021.

Equality Saturation and E-Graphs

Program Transformations with Syntactic Rewrites





$$(?X \times ?Y) \div ?Z \rightarrow ?X \times (?Y \div ?Z)$$
$$?X \div ?X \rightarrow 1$$
$$?X \times 1 \rightarrow ?X$$

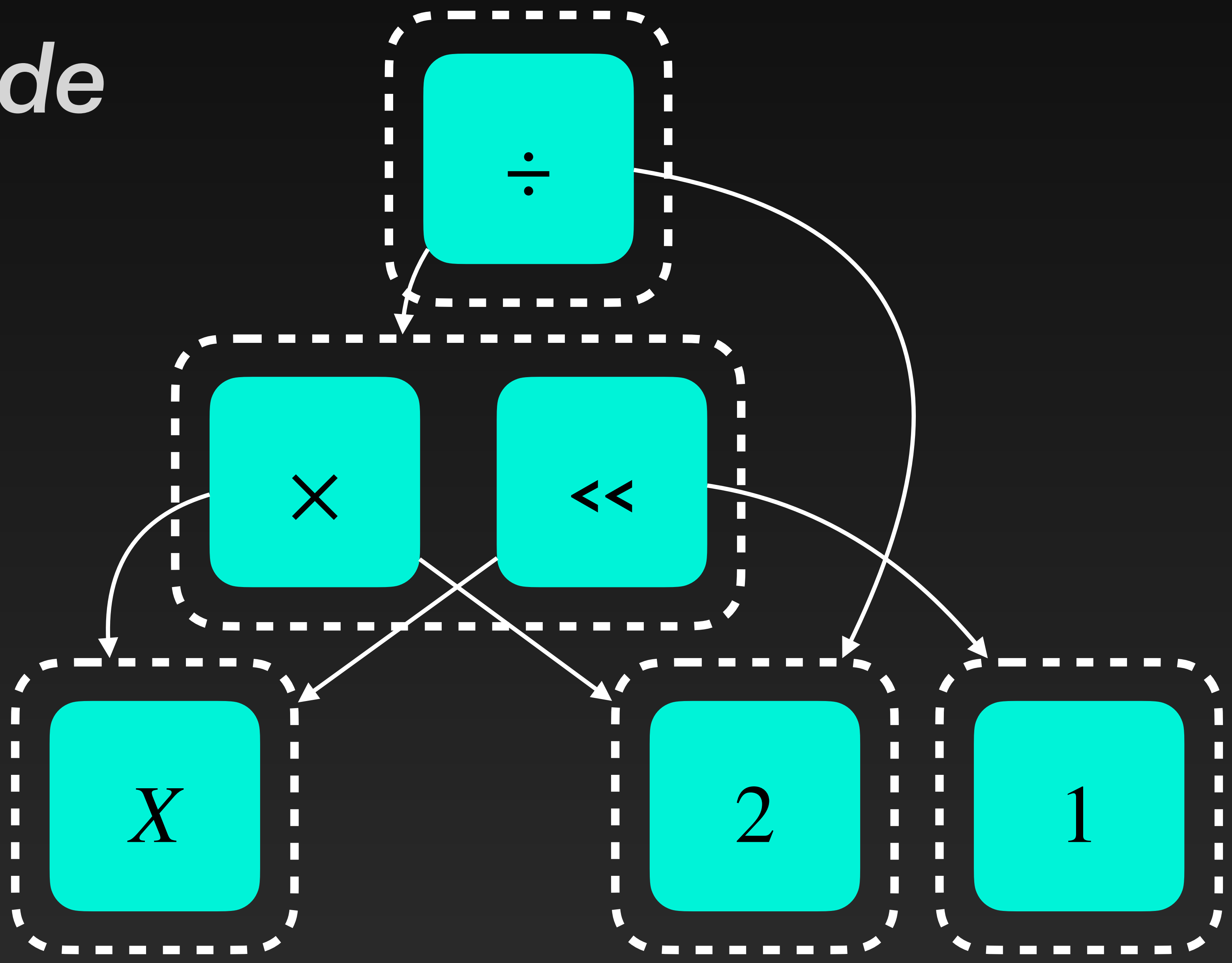
Non-destructive rewriting

Tate, R., et al, "Equality Saturation: a New Approach to Optimization," in *Logical Methods in Computer Science*, 2011.
Willsey, M., et al. "egg: Fast and extensible equality saturation," in *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–29, 2021.

Equality Saturation and E-Graphs

Term Extraction

1. Assign a *cost for each E-Node*

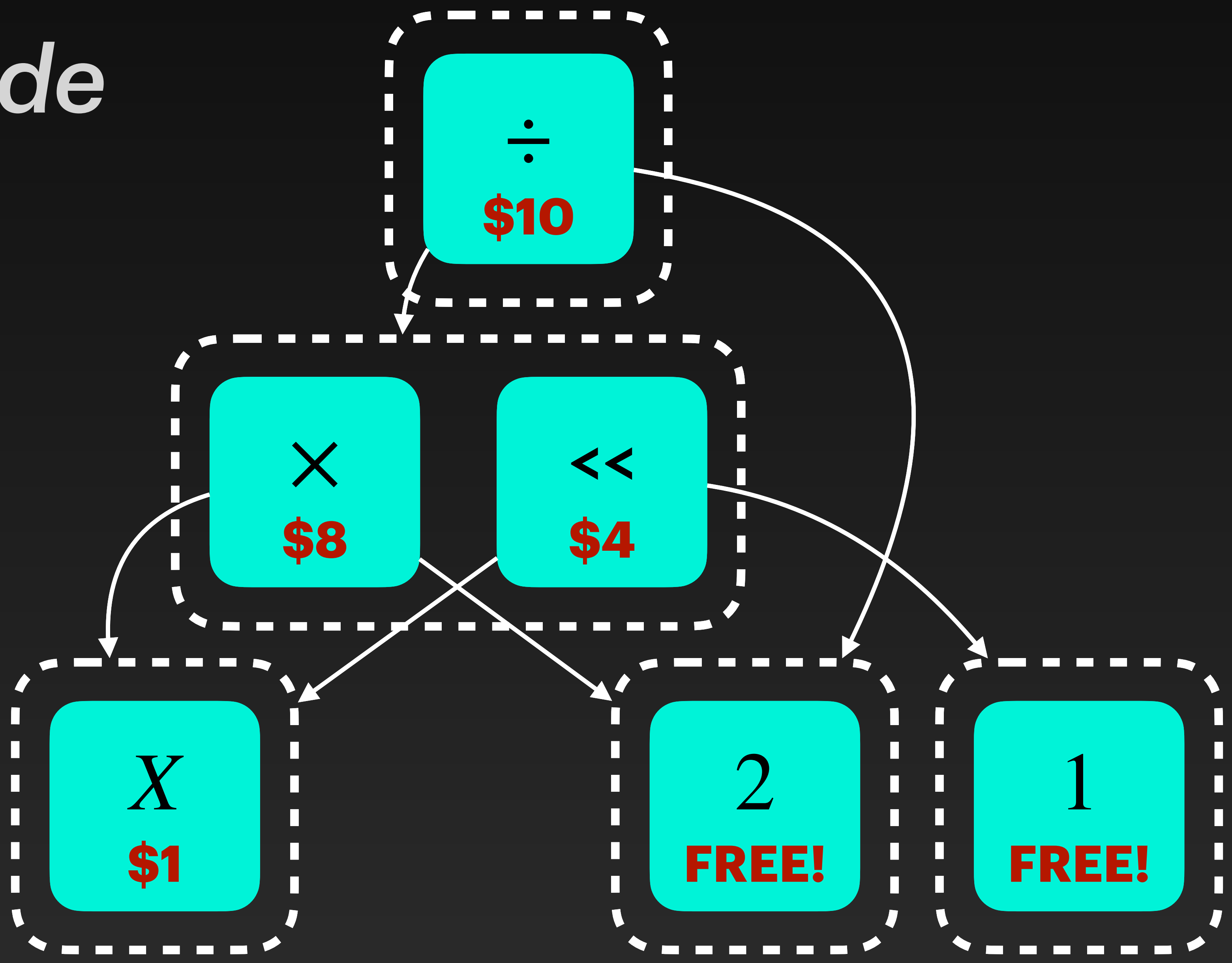


Tate, R., et al, "Equality Saturation: a New Approach to Optimization," in *Logical Methods in Computer Science*, 2011.
Willsey, M., et al. "egg: Fast and extensible equality saturation," in *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–29, 2021.

Equality Saturation and E-Graphs

Term Extraction

1. Assign a *cost for each E-Node*



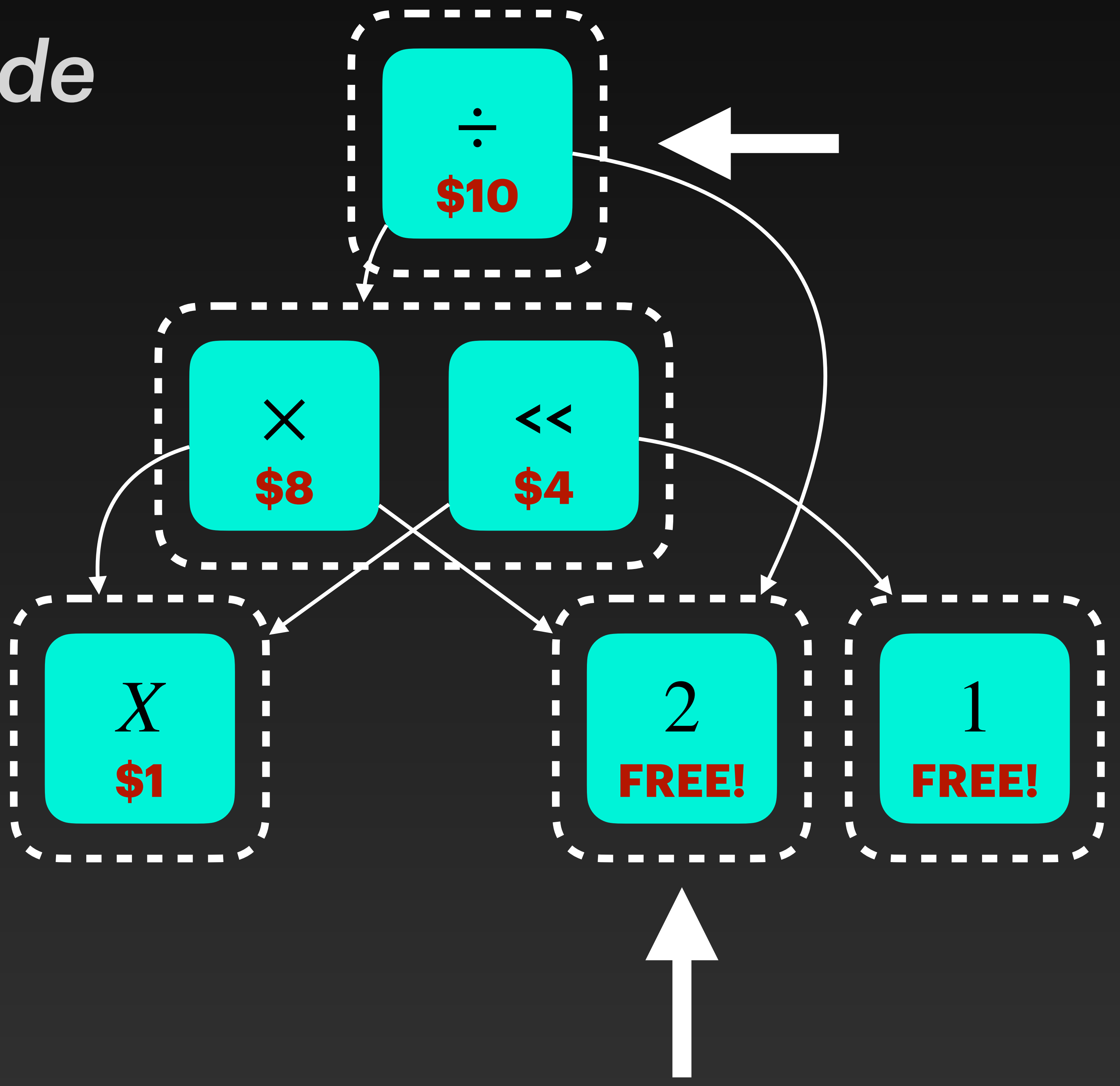
Tate, R., et al, "Equality Saturation: a New Approach to Optimization," in *Logical Methods in Computer Science*, 2011.
Willsey, M., et al. "egg: Fast and extensible equality saturation," in *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–29, 2021.

Equality Saturation and E-Graphs

Term Extraction

1. Assign a *cost for each E-Node*

2. Pick the min-cost term
    Attempt: Greedy



$(X \ll 1) \div 2$

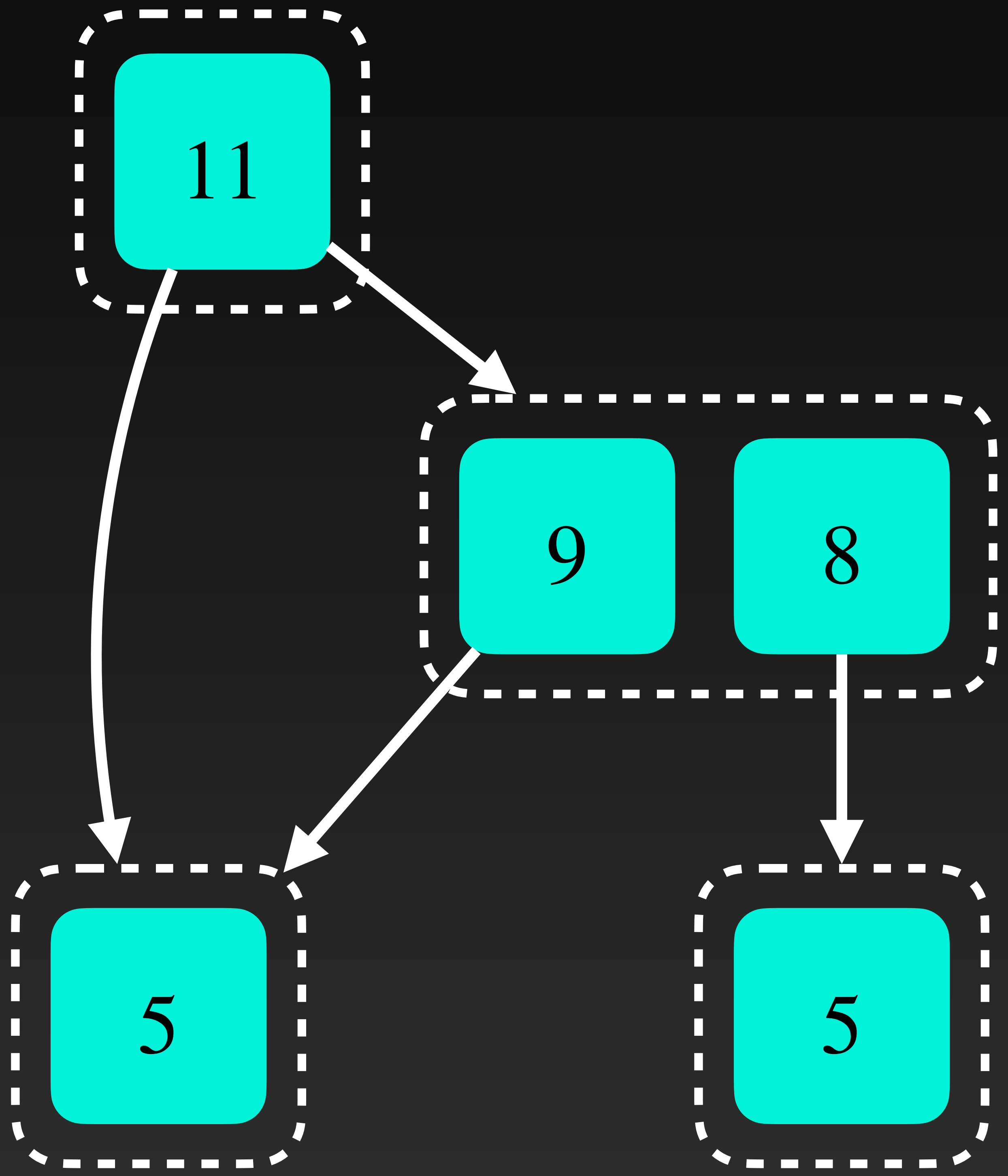
Cost = 10 + 4 + 1 = 15

Is That It?

Tate, R., et al, "Equality Saturation: a New Approach to Optimization," in *Logical Methods in Computer Science*, 2011.
Willsey, M., et al. "egg: Fast and extensible equality saturation," in *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–29, 2021.

Term Extraction
When Greedy Fails

Optimal:
11 + 9 + 5 = 25



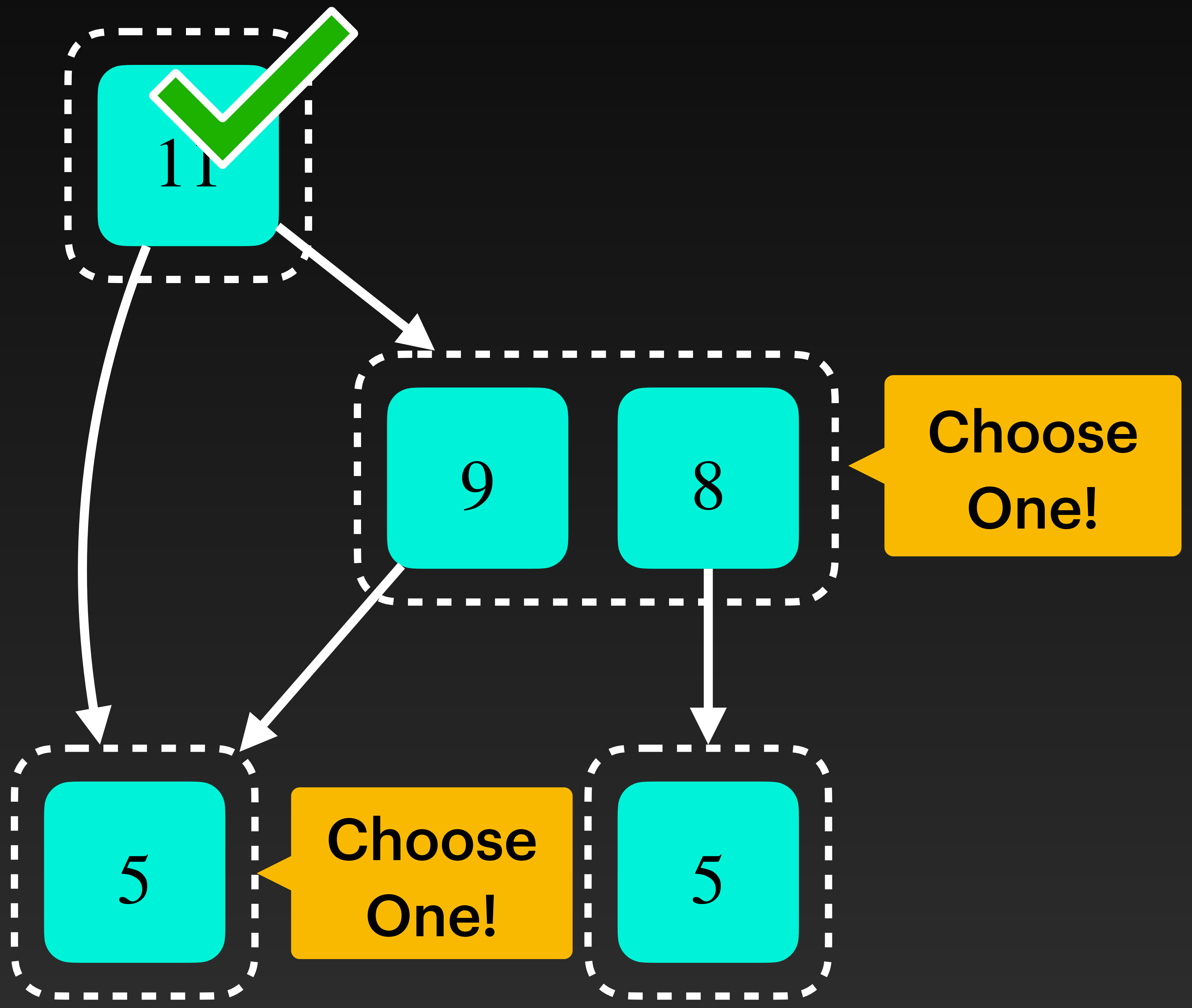
Greedy:
11 + 8 + 5 + 5 = 29

Previous work: ILP-based extraction

Root Constraint:
Extract at least one E-Node from the Root E-Class

Children Constraints:
If an E-Node n is extracted, then for all E-Class C, if C is a child of n, then extract at least one E-Node from C

Objective:
Minimize the sum of costs of extracted E-Node



Yichen Yang, Phitchaya Mangpo Phothilimtha, Yisu Remy Wang, Max Willsey, Sudip Roy, Jacques Pienaar. "Equality Saturation for Tensor Graph Superoptimization." (2021).

Previous work: ILP-based extraction

Variables: v_x for each e-node x

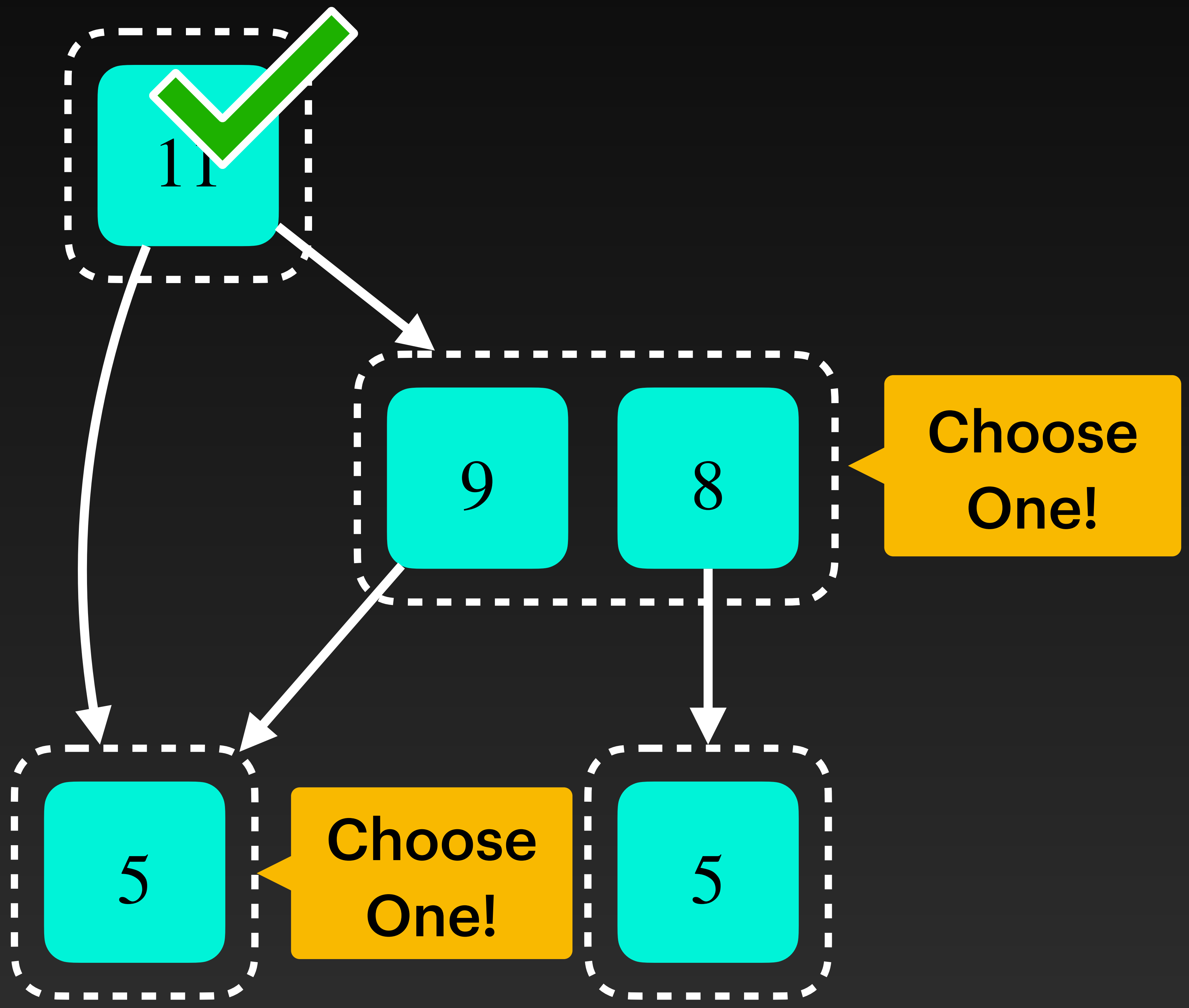
Objective:

$$\min \sum_x \text{cost}(x) \cdot v_x$$

Root Constraint: $\sum_{x \in \textbf{Root}} v_x \geq 1$

Children Constraints: $-v_x + \sum_{y \in C_i} v_y \geq 1$

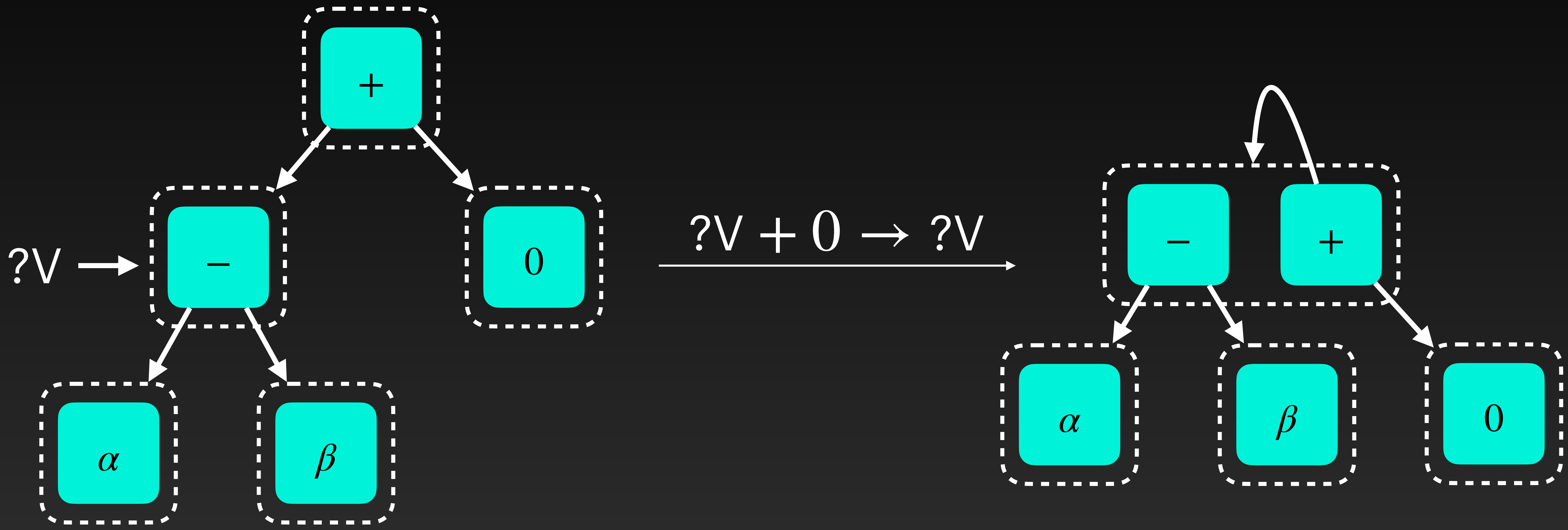
for each child C_i of x



Yichen Yang, Phitchaya Mangpo Phothilimtha, Yisu Remy Wang, Max Willsey, Sudip Roy, Jacques Pienaar. "Equality Saturation for Tensor Graph Superoptimization." (2021).

Previous work: ILP-based extraction

Cycles



How to avoid infinite expansions?

Yichen Yang, Phitchaya Mangpo Phothilimtha, Yisu Remy Wang, Max Willsey, Sudip Roy, Jacques Pienaar. "Equality Saturation for Tensor Graph Superoptimization." (2021).

Previous work: ILP-based extraction

Topological Order Constraints

Variables: v_x for each e-node x

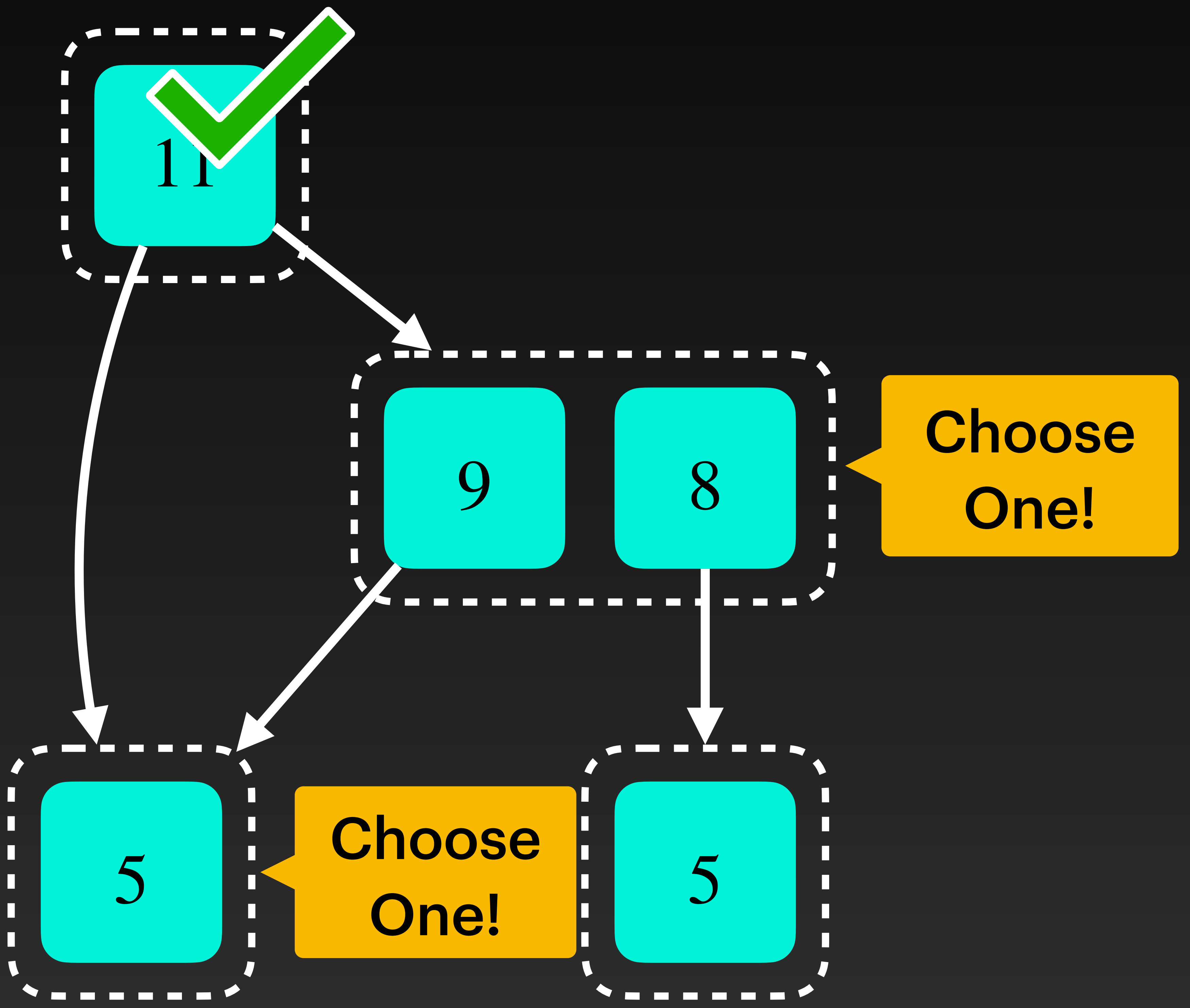
Objective:

$$\min \sum_x \text{cost}(x) \cdot v_x$$

Root Constraint: $\sum_{x \in \textbf{Root}} v_x \geq 1$

Children Constraints: $-v_x + \sum_{y \in C_i} v_y \geq 1$

for each child C_i of x



Yichen Yang, Phitchaya Mangpo Phothilimtha, Yisu Remy Wang, Max Willsey, Sudip Roy, Jacques Pienaar. "Equality Saturation for Tensor Graph Superoptimization." (2021).

Previous work: ILP-based extraction

Topological Order Constraints

Variables: v_x , o_x for each e-node x

Topological order

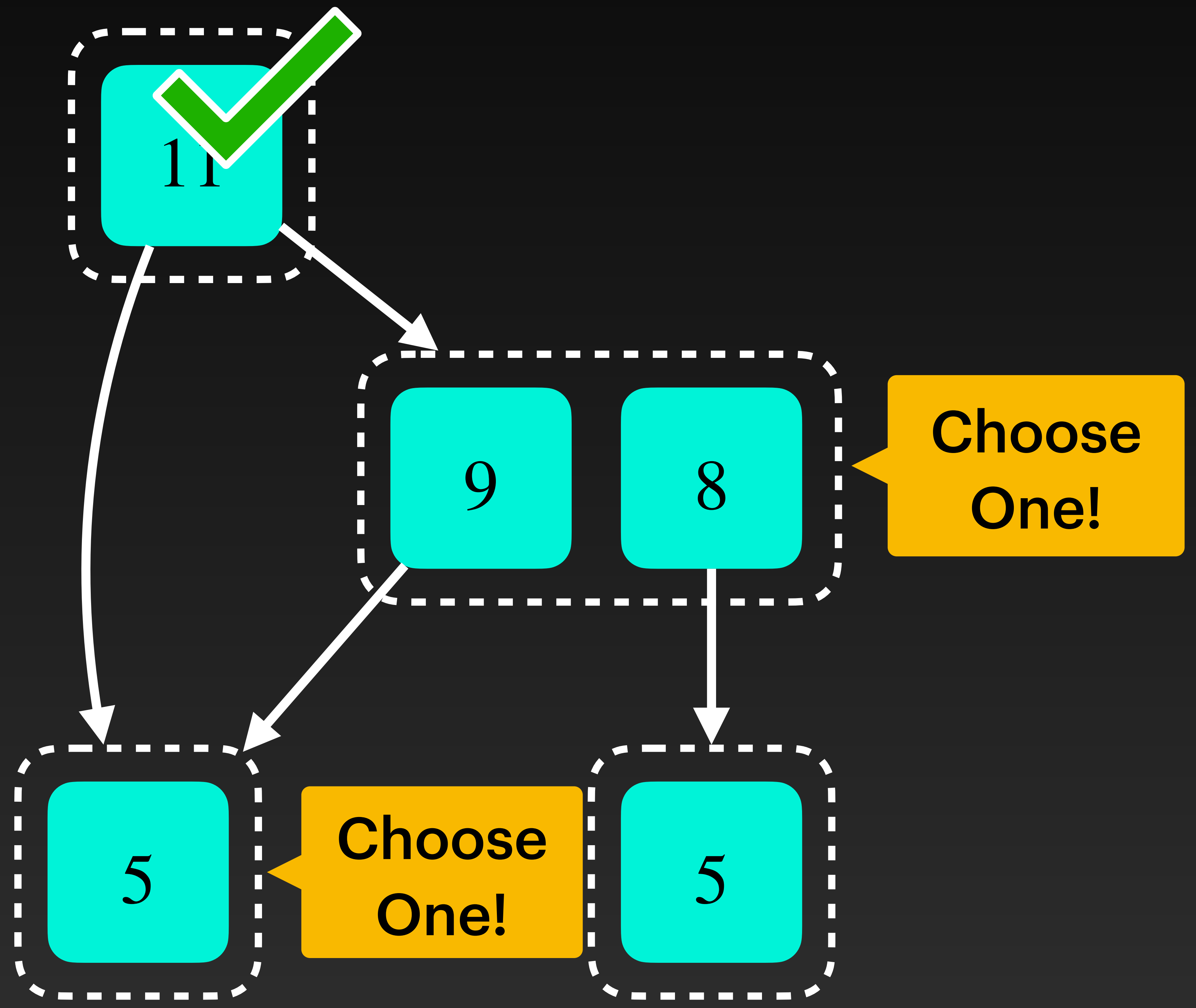
Objective:

$$\min \sum_x \text{cost}(x) \cdot v_x$$

Root Constraint: $\sum_{x \in \textbf{Root}} v_x \geq 1$

Children Constraints: $-v_x + \sum_{y \in C_i} v_y \geq 1$

for each child C_i of x



Topological order constraints: $o_y \geq o_x + 1$ $\left(\textbf{if } v_x = 1\right)$, ($y$ is in some children of x)

Yichen Yang, Phitchaya Mangpo Phothilimtha, Yisu Remy Wang, Max Willsey, Sudip Roy, Jacques Pienaar. "Equality Saturation for Tensor Graph Superoptimization." (2021).

Previous work: ILP-based extraction

Topological Order Constraints

Variables: v_x , o_x for each e-node x

Topological order

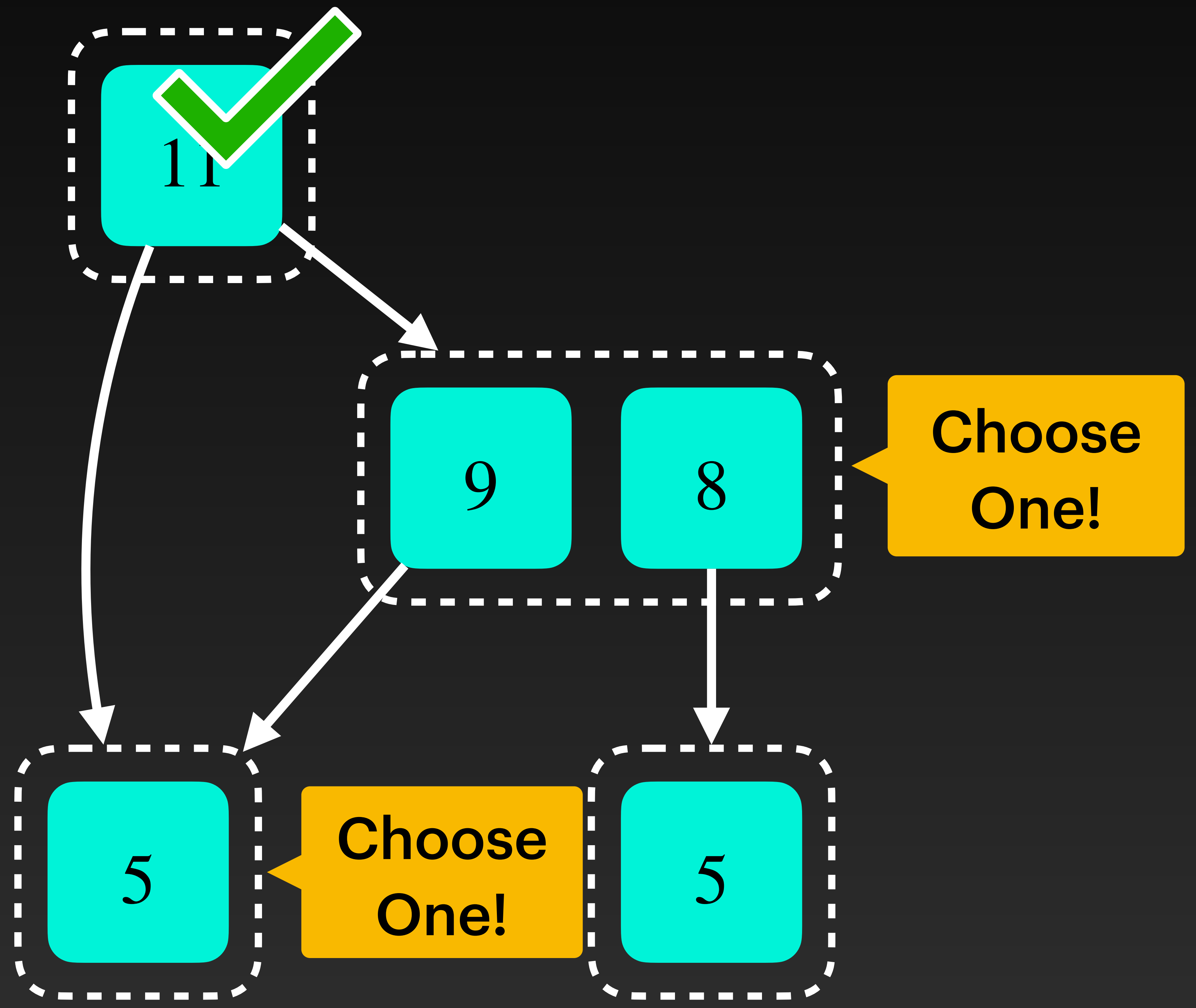
Objective:

$$\min \sum_x \text{cost}(x) \cdot v_x$$

Root Constraint: $\sum_{x \in \textbf{Root}} v_x \geq 1$

Children Constraints: $-v_x + \sum_{y \in C_i} v_y \geq 1$

for each child C_i of x



Topological order constraints: $o_y + (1 - v_x) \cdot L \geq o_x + 1$ (y is in some children of x)

L is a large enough constant

 # Variables: $O(n)$ **# Constraints:** $O(n)$ **Search Space:** $O(2^n + n^n)$

Yichen Yang, Phitchaya Mangpo Phothilimtha, Yisu Remy Wang, Max Willsey, Sudip Roy, Jacques Pienaar. "Equality Saturation for Tensor Graph Superoptimization." (2021).

Our solution 1: ILP + Acyclicity constraints

Variables: v_x for each e-node x

Objective:

$$\min \sum_x \text{cost}(x) \cdot v_x$$

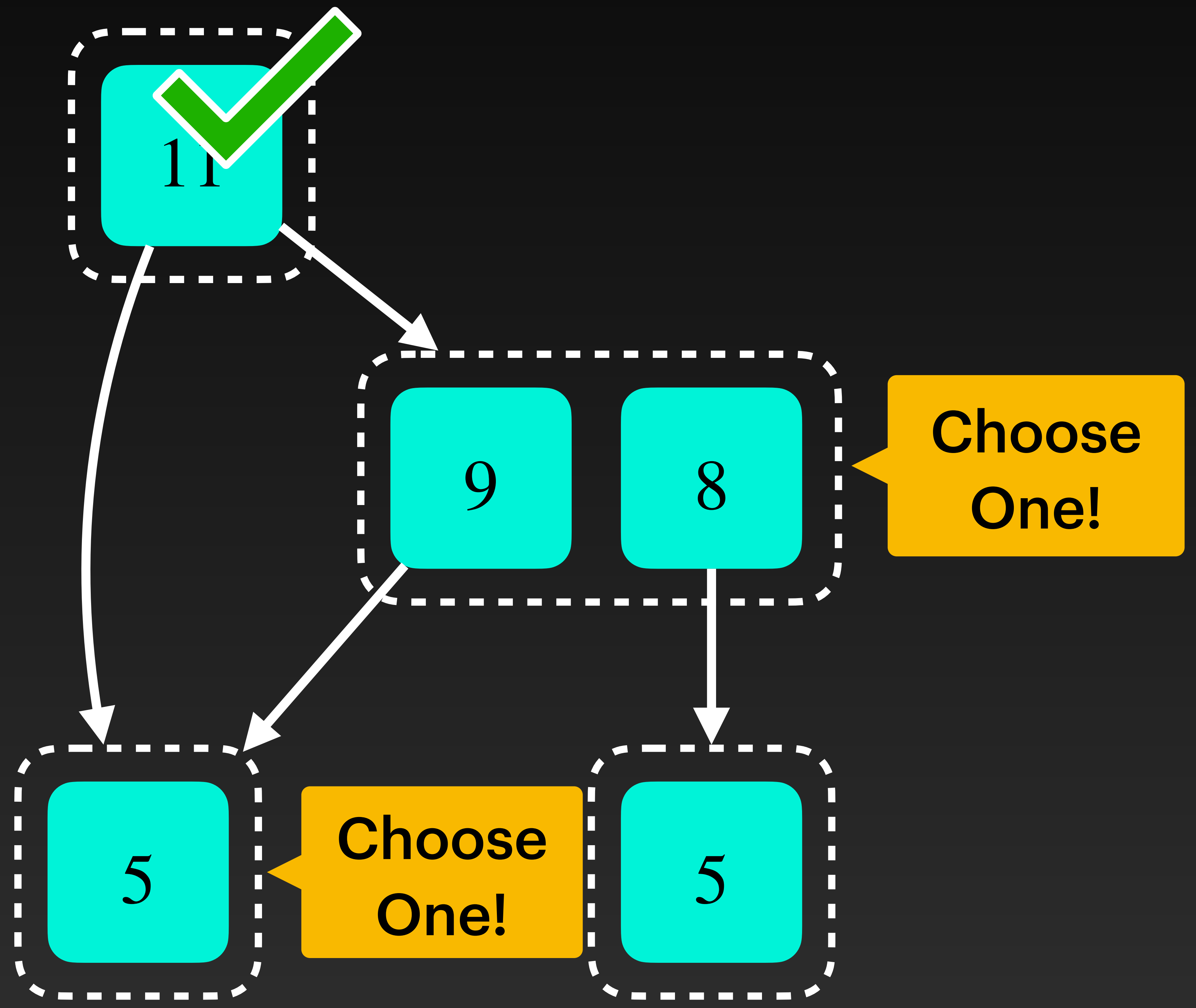
Root Constraint: $\sum_{x \in \textbf{Root}} v_x \geq 1$

Children Constraints: $-v_x + \sum_{y \in C_i} v_y \geq 1$

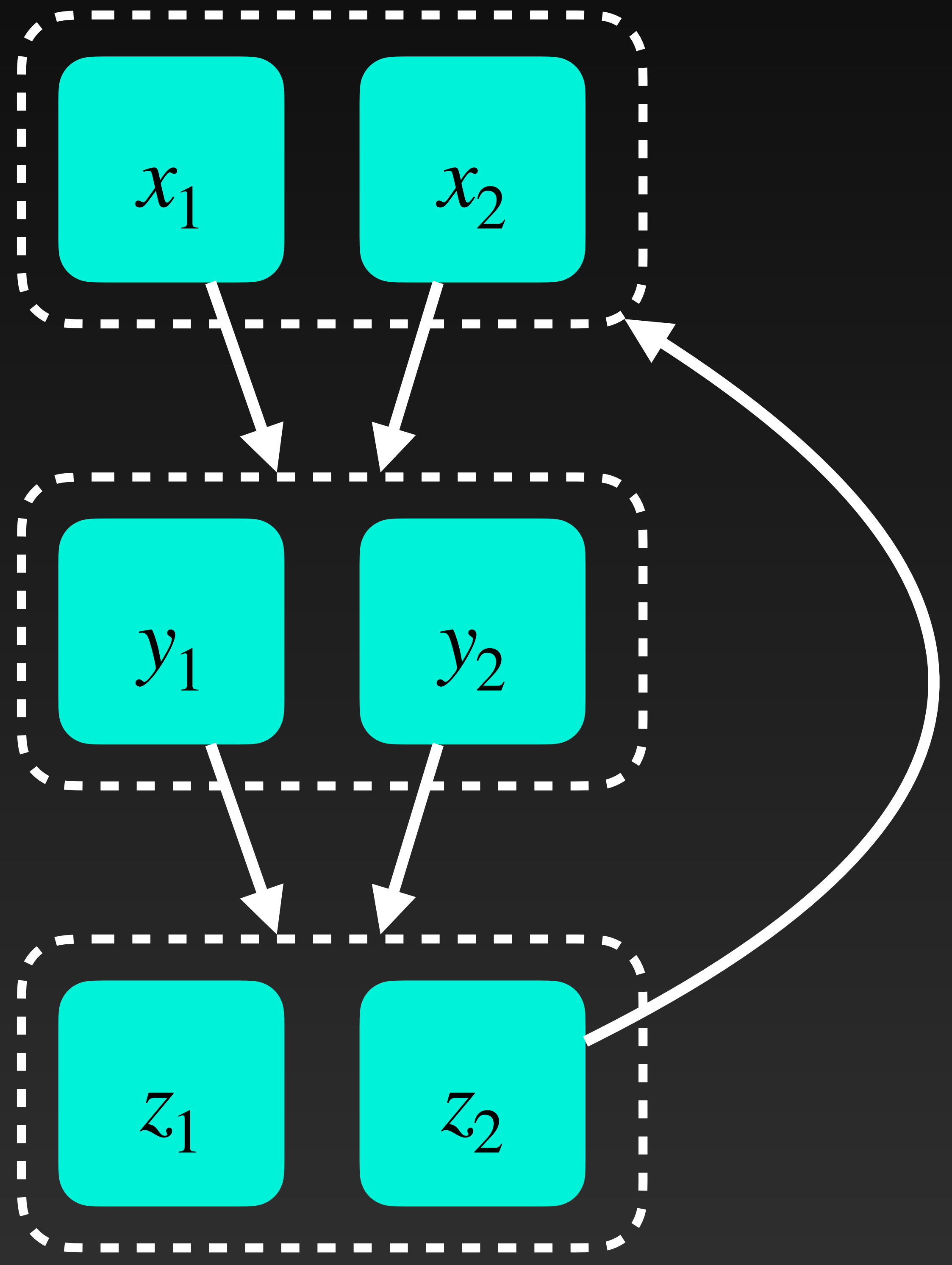
for each child C_i of x

Acyclicity Constraints: Do not extract any cycle

Works well when number of cycles is reasonable



Acyclicity constraints



$$Tseitin\left(\bigvee \begin{matrix} (\neg x_1 \wedge \neg x_2) \\ (\neg y_1 \wedge \neg y_2) \\ \neg z_2 \end{matrix} \right)$$

$$\Leftrightarrow$$

$$\bigwedge \begin{pmatrix} O_1 \leftrightarrow (\neg x_1 \wedge \neg x_2) \\ O_2 \leftrightarrow (\neg y_1 \wedge \neg y_2) \\ O_1 \vee O_2 \vee \neg z_2 \end{pmatrix}$$

Acyclicity constraints in ILP formulation

Solution 1: ILP + Acyclicity constraints

Variables: v_x for each e-node x

Objective:

$$\min \sum_x \text{cost}(x) \cdot v_x$$

Root Constraint: $\sum_{x \in \textbf{Root}} v_x \geq 1$

Children Constraints: $-v_x + \sum_{y \in C_i} v_y \geq 1$

for each child C_i of x

Acyclicity Constraints: Acyclicity constraints in ILP formulation

 # Variables: $O(n)$ **# Constraints:** $O(n \cdot \#\text{cycles})$ **Search Space:** $O(2^n)$

Solution 2: Weighted Partial MaxSAT

For each E-Node x, create a boolean variable v_x

v_x is $\top \Leftrightarrow x$ is in the extracted term

Must always be satisfied

Hard Clauses

Root Constraint:

$$\bigvee_{x \in \text{Root}} v_x$$

Children Constraints:

$$v_x \rightarrow \bigwedge_{C \in \text{children}(x)} \bigvee_{x' \in C} v_{x'}$$

Acyclicity Constraints:

$$\text{Tesitin}\left(\bigvee_{C_i} \bigwedge_{x \in C_i \wedge \text{in\_cycle}(x)} v_x\right)$$

SAT / UNSAT

Soft Clauses

$\neg v_x$ with weight cost(x)

Objective:
Maximizing weight of unextracted E-Nodes

 # Variables: $O(n)$ **# Constraints:** $O(n \cdot \#\text{cycles})$ **Search Space:** $O(2^n)$

Term extraction

Complexity

Solution 1 (***ILP-ACyc***): ILP formulation with acyclic constraints

Solution 2 (***WPMAXSAT***): Weighted partial MaxSAT formulation with acyclic constraints

Previous work (***ILP-Topo***): ILP with topological order constraints

Encoding	# Variables	# Constraints	Search Space Complexity
ILP-ACyc WPMAXSAT	$O(n)$	$O(nk)$	$O(2^n)$
ILP-Topo	$O(n)$	$O(n)$	$O(2^n + n^n)$

Same solution space

n: number of E-Nodes

k: number of E-Class cycles

Potentially Exponential

Term extraction
Evaluation benchmarks

Empirically

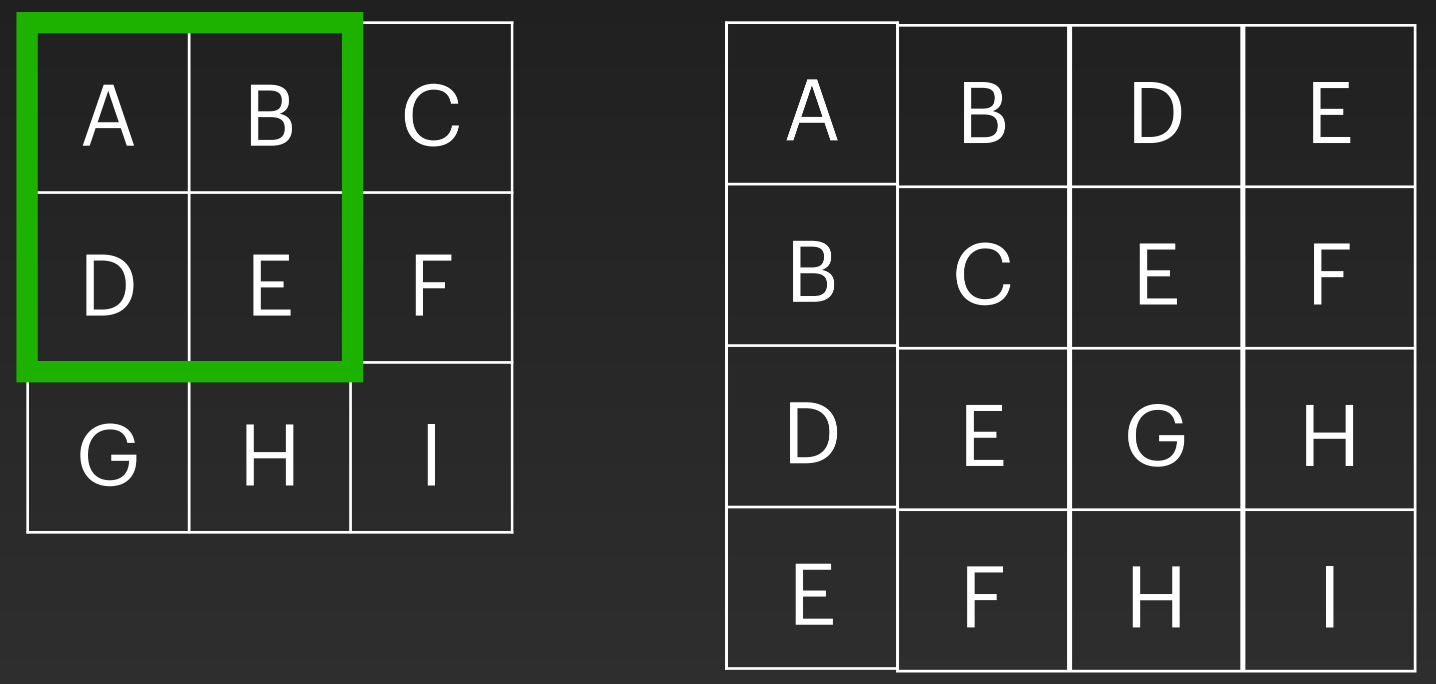
Implemented a prototype in the egg [1] framework

Workload: term extraction after equality saturation on tensor programs (DNNs) including

MobileNetV2, ResMLP, ResNet-18, ResNet-50, EfficientNet

Rewrite rules from Glenside [2]

- *Image-to-column (im2col) only*
- *Image-to-column (im2col) + simplifications (operator fusion, reordering, etc.)*



Im2col of a 3x3 input for a 2x2 kernel

[1] Willsey, M., et al. "egg: Fast and extensible equality saturation," in *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–29, 2021.
[2] Smith, Gus Henry, Andrew, Liu, Steven, Lyubomirsky, Scott, Davidson, Joseph, McMahan, Michael, Taylor, Luis, Ceze, Zachary, Tatlock. "Pure tensor program rewriting via access patterns (representation pearl)." *Proceedings of the 5th ACM SIGPLAN International Symposium on Machine Programming*. ACM, 2021.

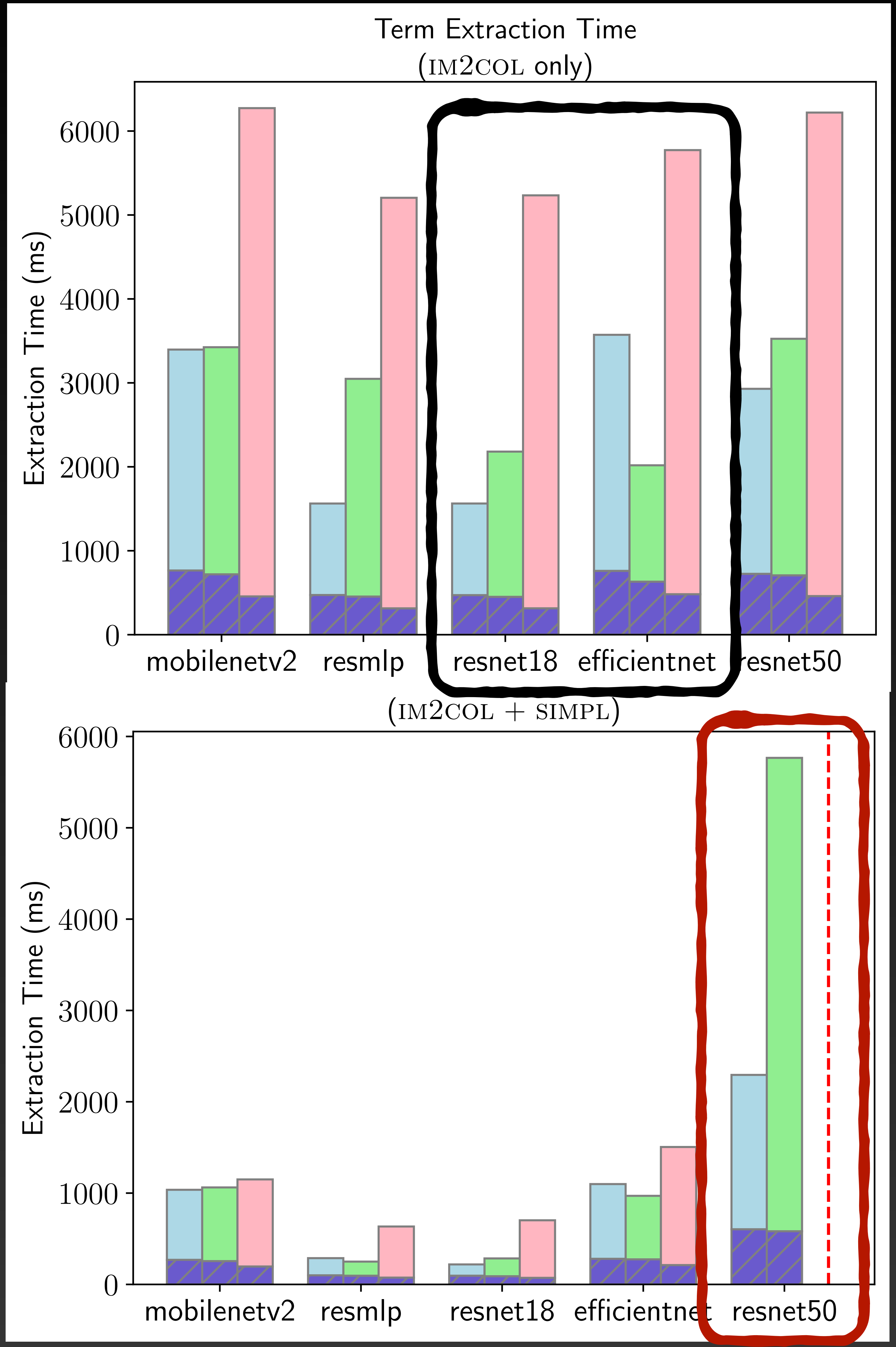
Term extraction
Benchmark statistics

Unit: 1,000	MobileNetV2		ResMLP		ResNet-18		ResNet-50		EfficientNet	
	Im2Col	Im2Col+ SIMPL	Im2Col	Im2Col+ SIMPL	Im2Col	Im2Col+ SIMPL	Im2Col	Im2Col+ SIMPL	Im2Col	Im2Col+ SIMPL
# E-Nodes	50	20	40	8	35	8	45	40	50	20
# E-Classes	25	6	20	2.5	25	3	22	20	20	7
# Cycles	17	17	15	4	14	4	21	10	16	20

Statistics of saturated E-Graphs (Unit: 1k)

Term extraction

Evaluation results



X: Models

Y: End-to-End extraction time (milliseconds)

Upper: Image-to-column rewrite rule only

Lower: Image-to-column + simplifications including Operator fusion, reordering, etc.

WPMAXSAT | ILP-ACyc | ILP-Topo | Overhead

ILP-Topo timeouts (300s)

Solving WPMAXSAT and ILP-ACyc is ***~3x faster*** than solving ILP-Topo

For a larger input, solving ILP-Topo (previous work) timeouts after 300s while solving WPMAXSAT and ILP-ACyc takes a few seconds

Optimality is guaranteed by all encodings

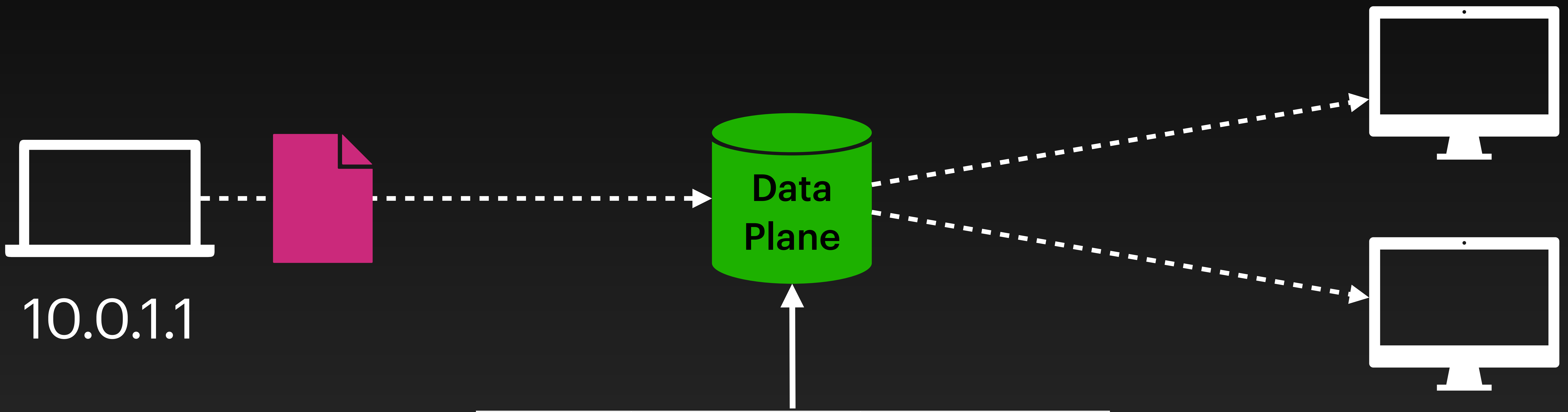
Mike H., et al, "Improving Term Extraction with Acyclic Constraints," in *E-Graph Research, Applications, Practices, and Human-factors Symposium (EGRAPHS'23)*, 2023.

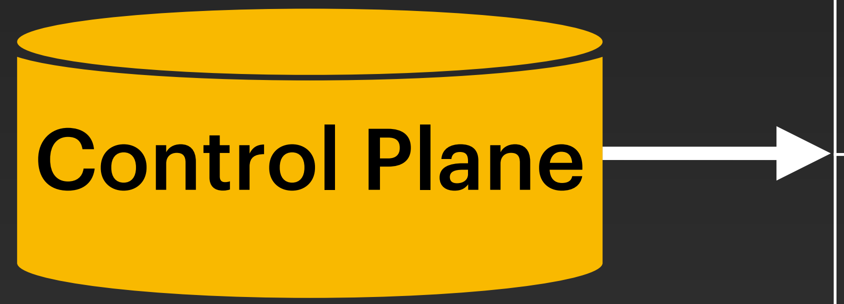
EGRAPHS'23 Workshop paper

https://www.cs.princeton.edu/~dh7120/assets/papers/EGRAPHS2023.pdf

CatsTail: P4 Resource Synthesis using Equality Saturation

Programmable switches



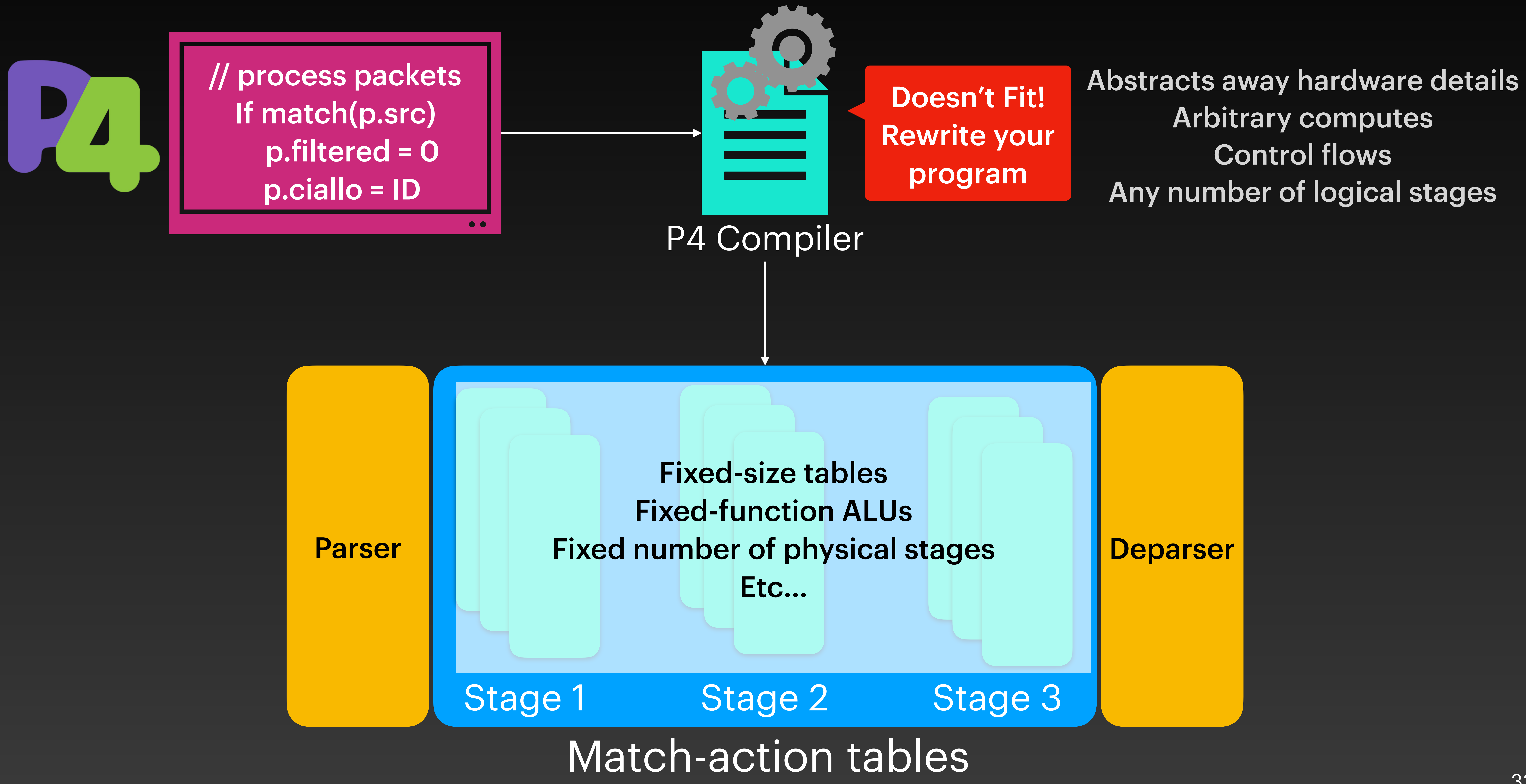


Key	Value	Action
src_ip	10.0.1.1	DROP
count	1	INCR(num_pkt)
...	...	...

Match-action tables

Bosshart, Pat, Dan, Daly, Glen, Gibb, Martin, Izzard, Nick, McKeown, Jennifer, Rexford, Cole, Schlesinger, Dan, Talayco, Amin, Vahdat, George, Varghese, David, Walker. "P4: programming protocol-independent packet processors". *SIGCOMM Comput. Commun. Rev.* 44. 3(2014): 87–95.

Mapping to programmable switches is hard



Mapping to programmable switches is hard

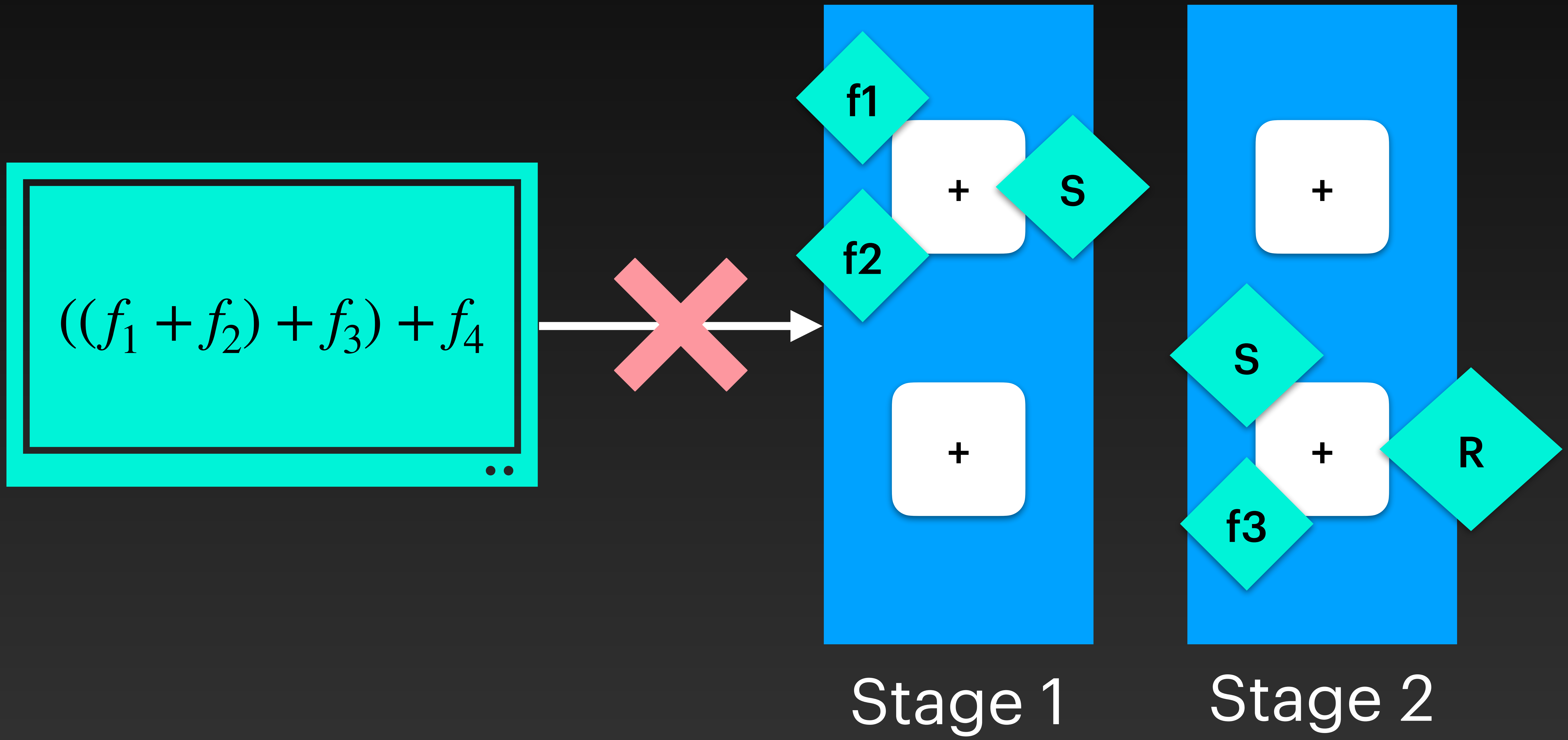
Challenge 1: Limited # of Stages

Challenge 2: Table Dependencies

Challenge 3: Targeting different backends

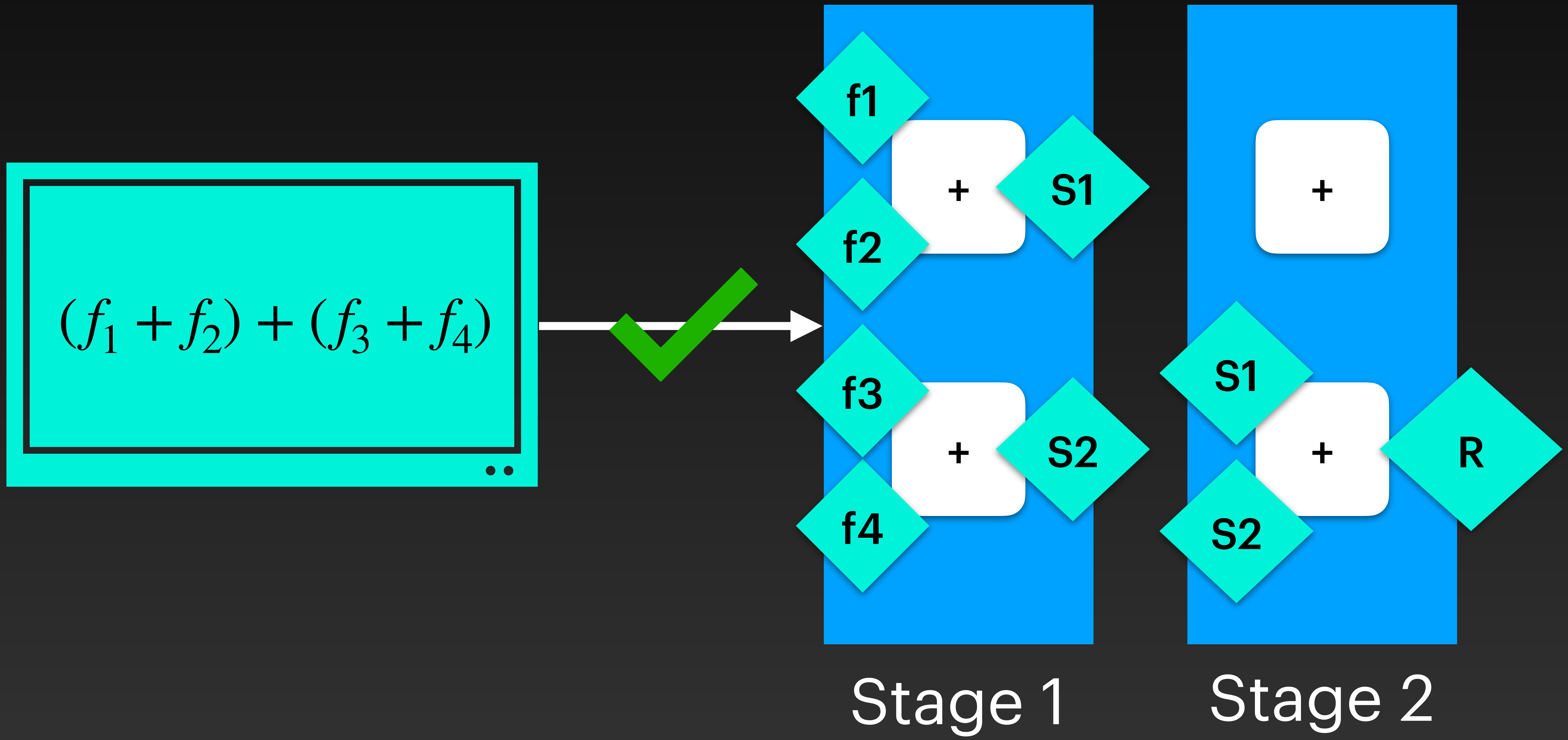
Mapping to programmable switches is hard

Challenge 1: Limited # of Stages



Mapping to programmable switches is hard

Challenge 1: Limited # of Stages



Mapping to programmable switches is hard

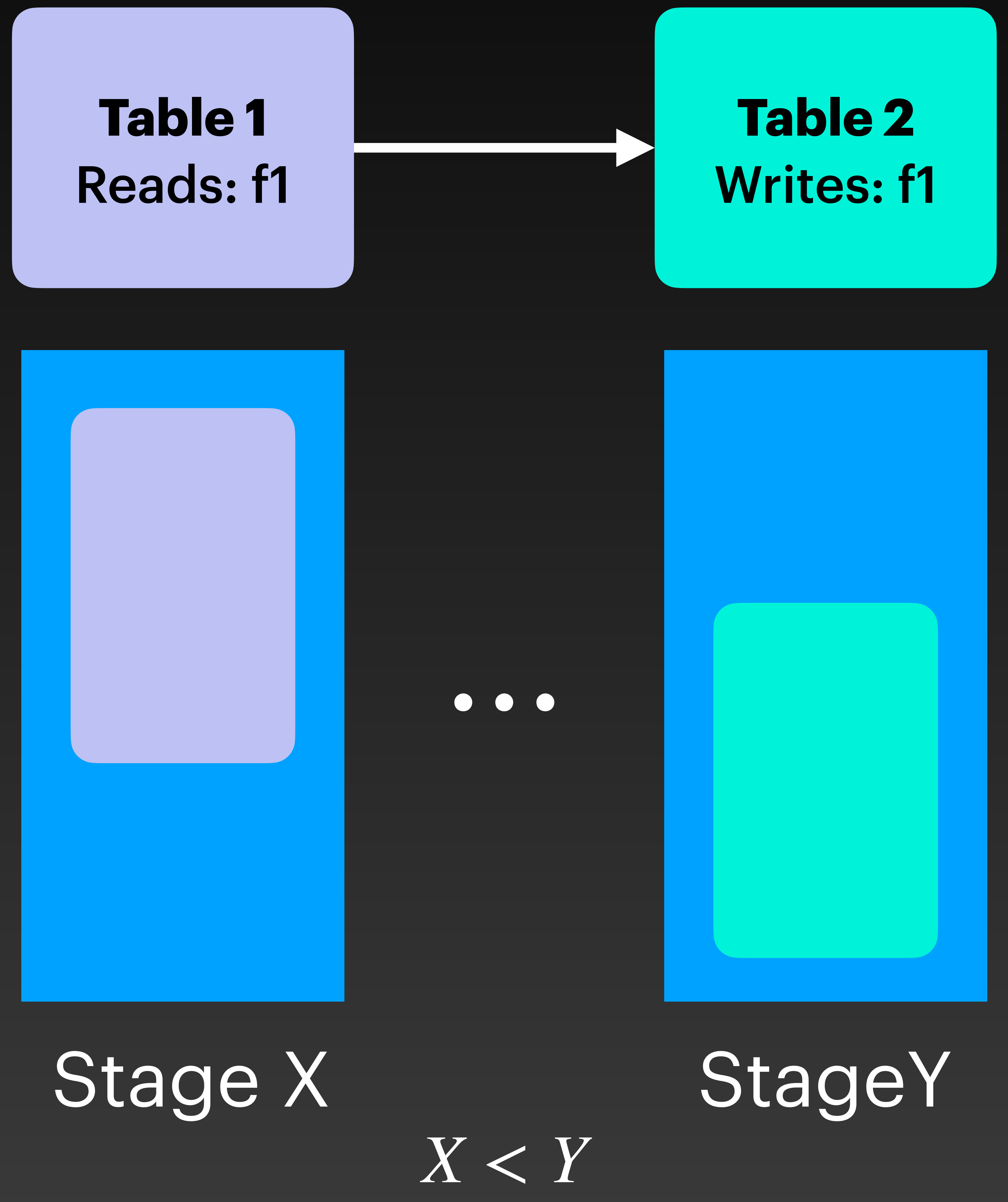
Challenge 2: Table Dependencies

R/W Dependencies

(this example) Write-after-Read

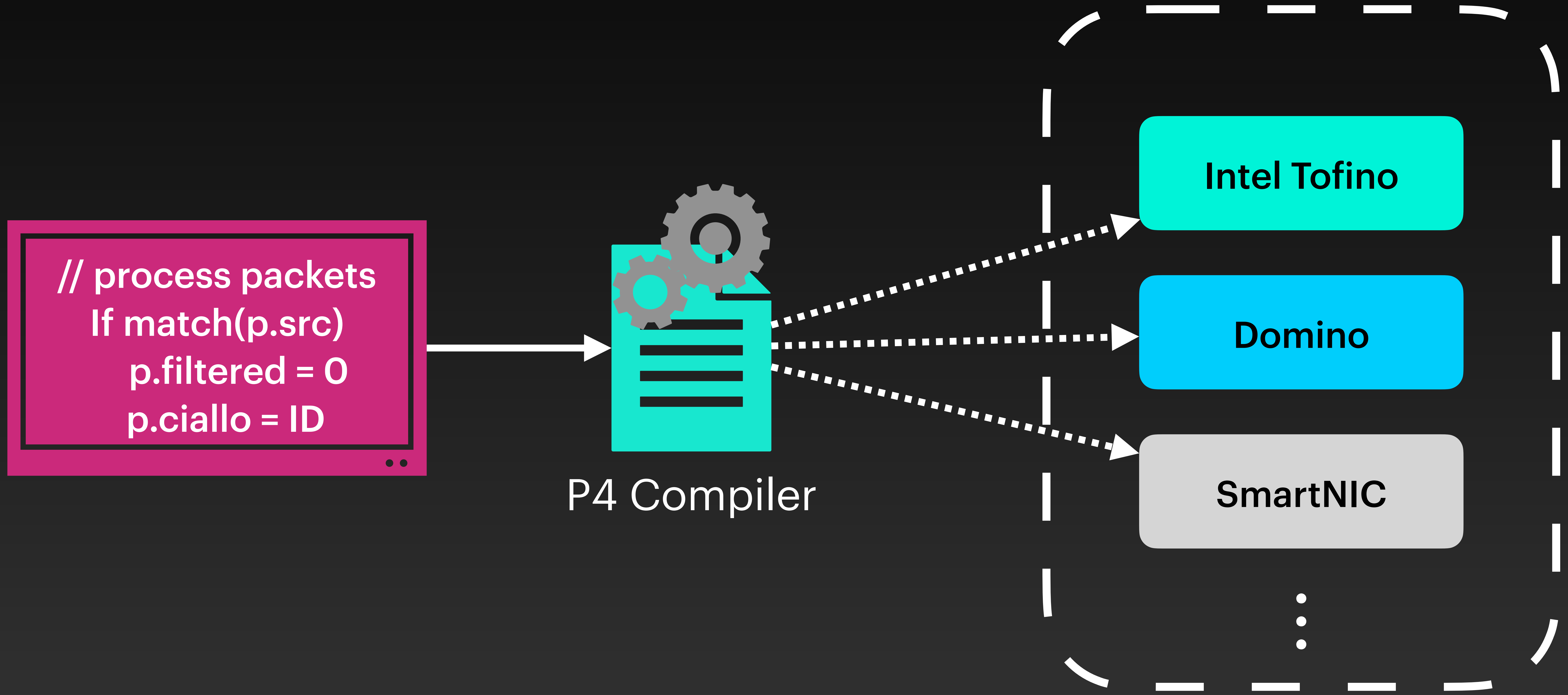
Read-after-Write

Write-after-Write



Mapping to programmable switches is hard

Challenge 3: Targeting different backends



Previous work: CaT

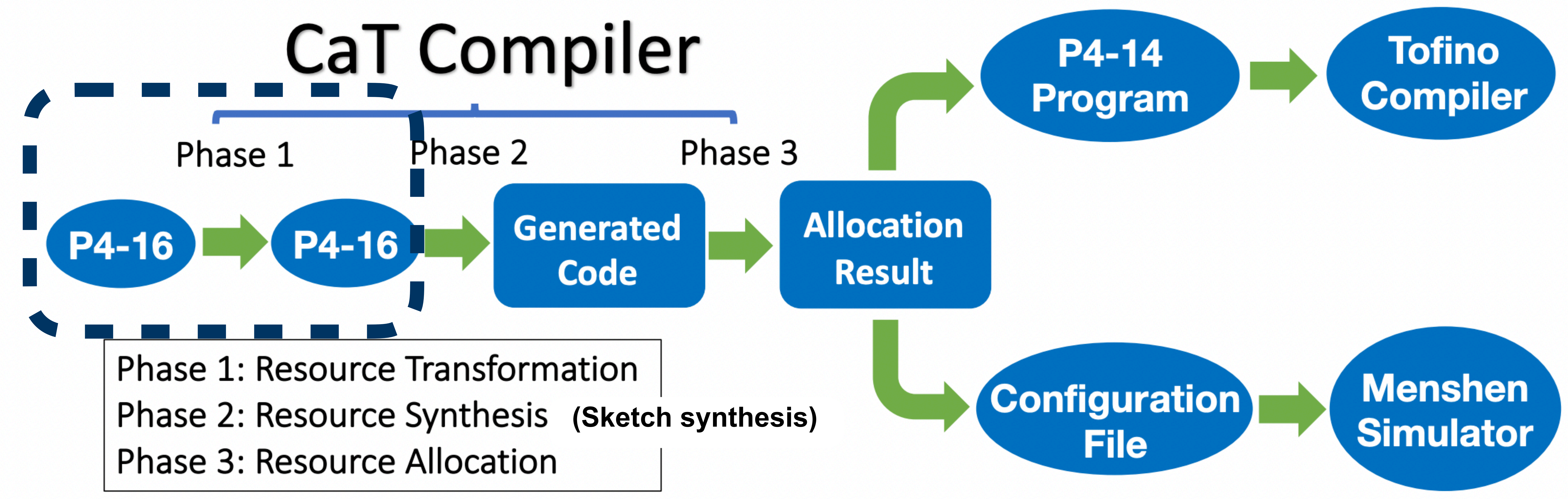


Figure 1: The workflow of the CaT compiler.

Gao, Xiangyu, Divya, Raghunathan, Ruijie, Fang, Tao, Wang, Xiaotong, Zhu, Anirudh, Sivaraman, Srinivas, Narayana, Aarti, Gupta. "CaT: A Solver-Aided Compiler for Packet-Processing Pipelines." Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3. Association for Computing Machinery, 2023.

Previous work: CaT

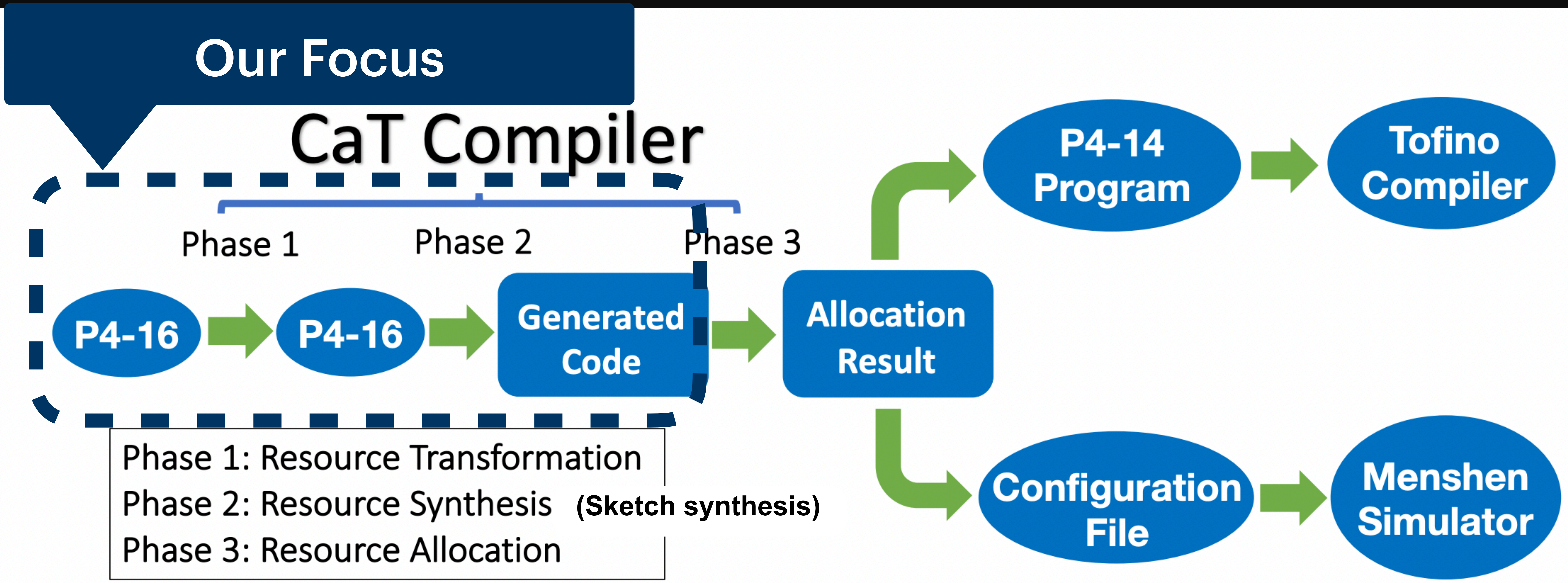
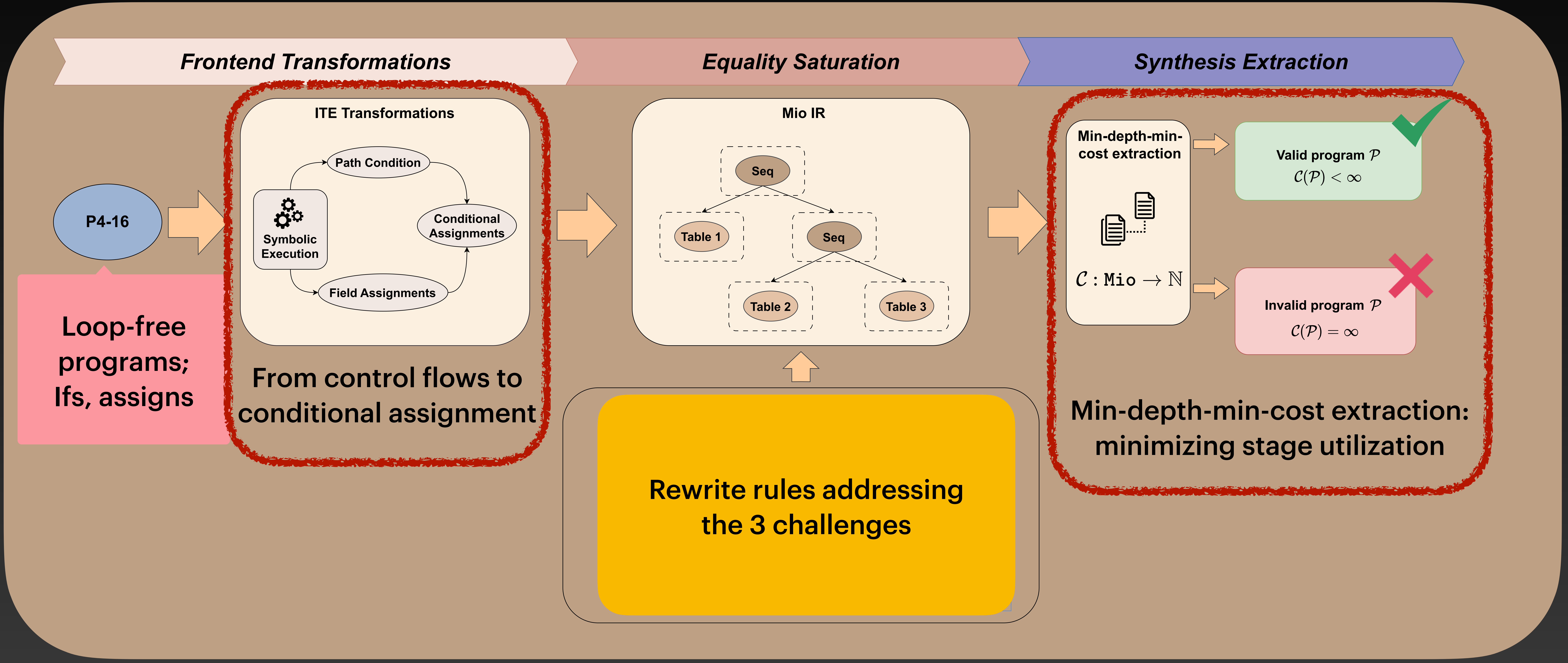


Figure 1: The workflow of the CaT compiler.

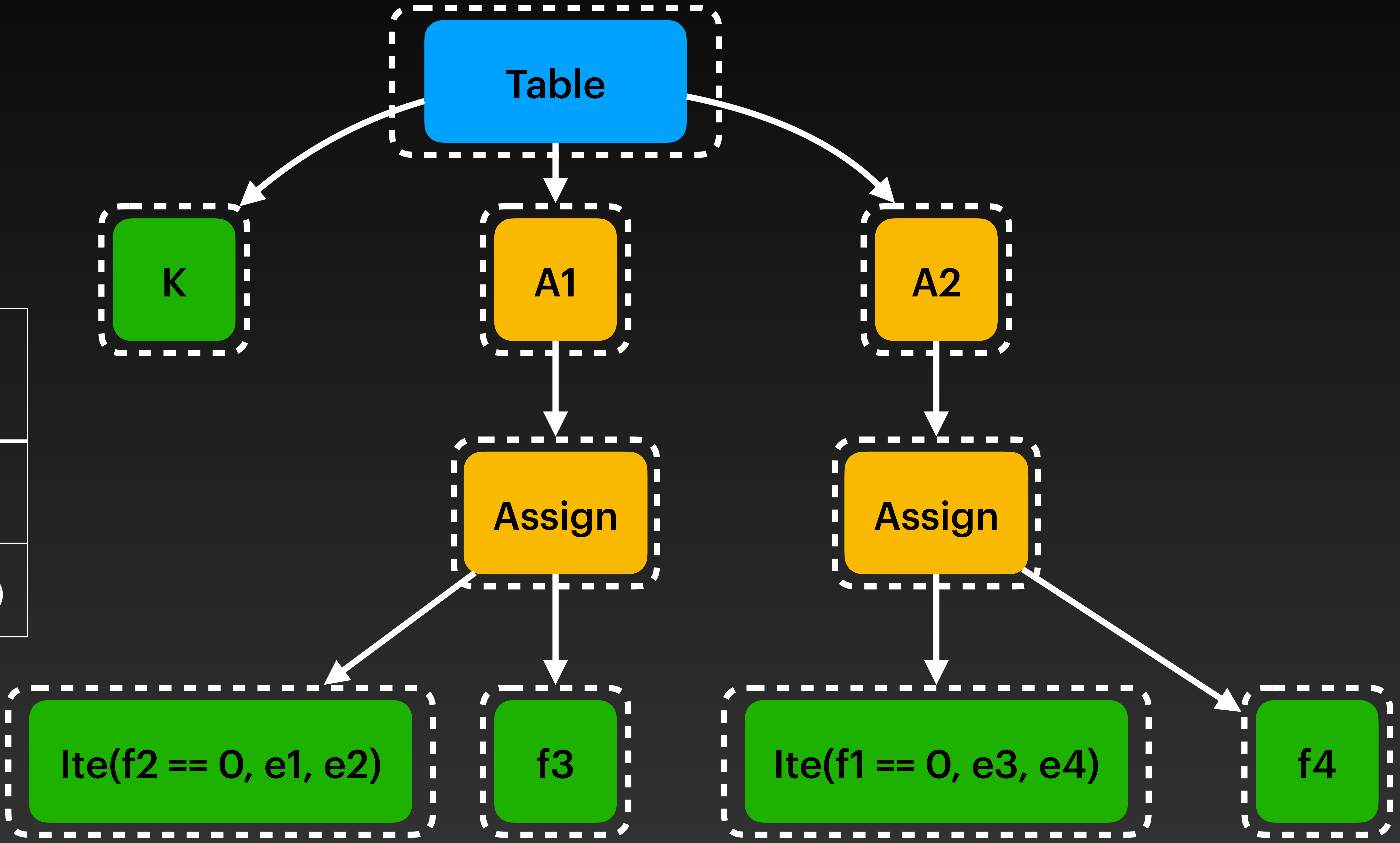
Gao, Xiangyu, Divya, Raghunathan, Ruijie, Fang, Tao, Wang, Xiaotong, Zhu, Anirudh, Sivaraman, Srinivas, Narayana, Aarti, Gupta. "CaT: A Solver-Aided Compiler for Packet-Processing Pipelines." Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3. Association for Computing Machinery, 2023.

Resource synthesis via Equality Saturation



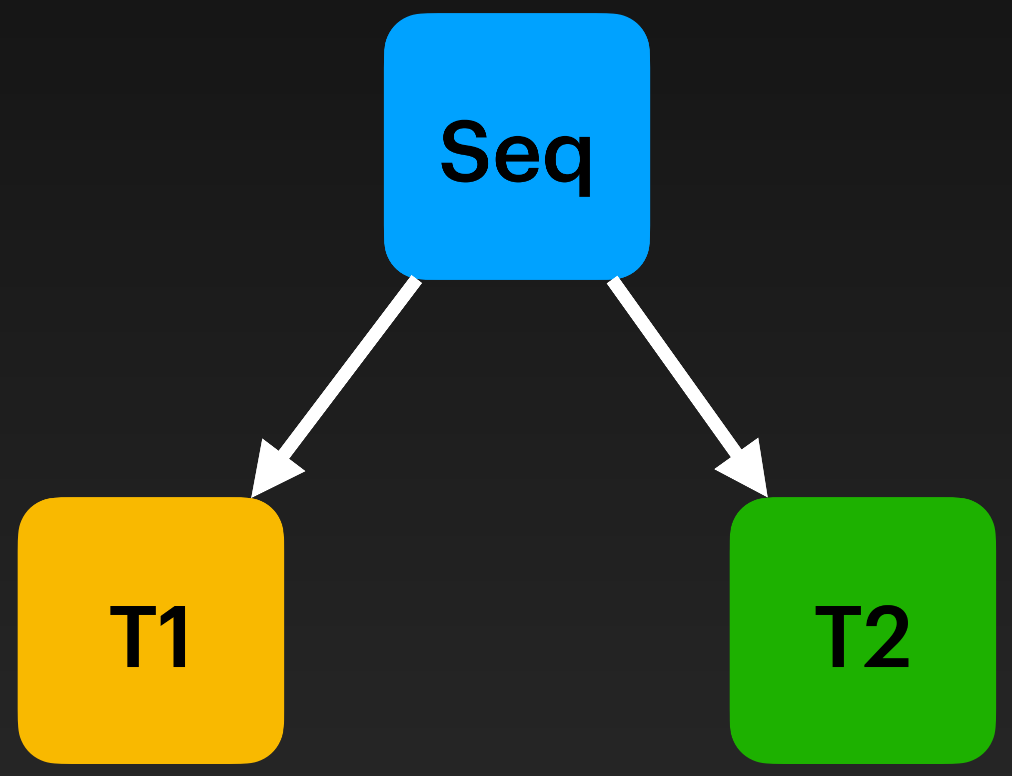
Frontend transformation

Match	Action
K	hdr.f3 = ite(hdr.f2 == 0, e1, e2)
	hdr.f4 = ite(hdr.f1 == 0, e3, e4)

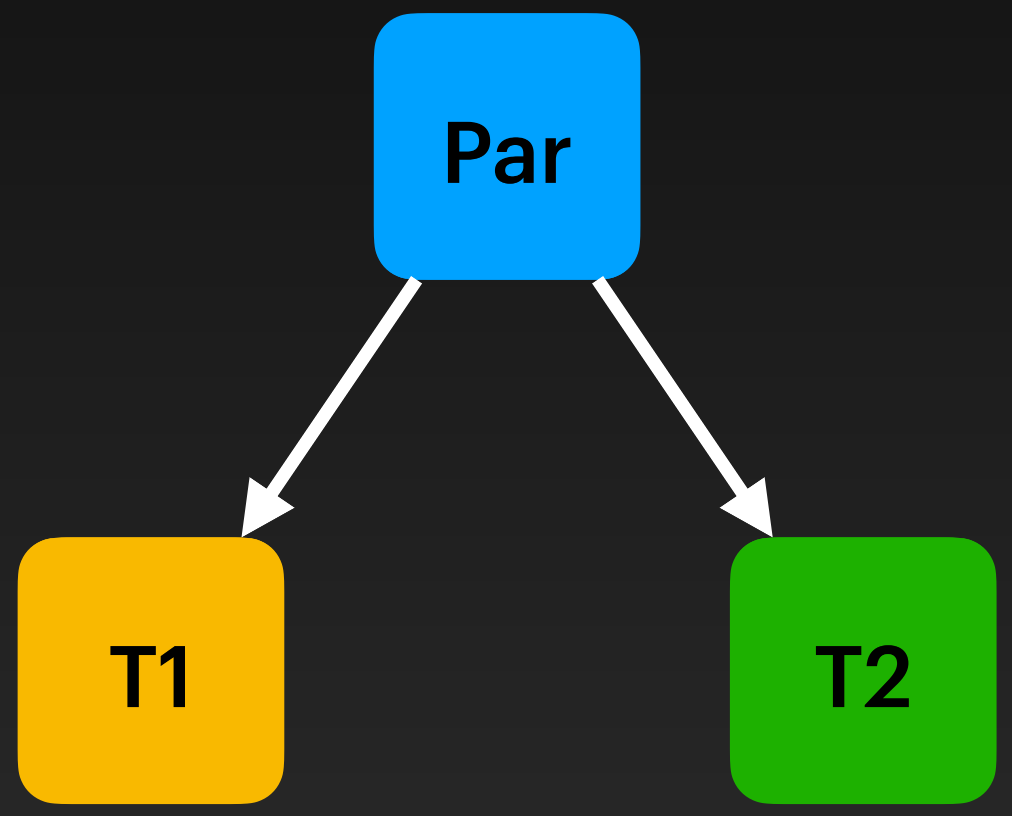


Frontend transformation

Introduce Table operators to allow table transformations



T1 must be placed before T2



T1 and T2 are put in the same stage

Rewrite rules

Challenge 1: Limited resource

General-purpose program transformations

?x + ?y => ?y + ?x
(?x + ?y) + ?z <=> ?x + (?y + ?z)
?x + 0 => ?x
~(?x & ?y) => ~?x | ~?y
?x & ?x => ?x
ite(true, ?x, ?y) => ?x
ite(false, ?x, ?y) => ?y
Etc…

52 Rules

Challenge 2: Table Dependencies

Table Transformations

Table parallelization
Subexpression lifting
Table merging
Etc…

10 Rules

Challenge 3: Different backends

Synthesis rewrites
1-1 to sketch grammars in CaT (Gao et al.)

?x + ?y => **alu_add** ?x ?y
if **mapped**(?x) & **mapped**(?y)

?V = **ite**(?x == ?y, ?x + ?z, ?x) =>
stateful_alu(if, ?V, ?x == ?y, ?x + z, ?x)
if …

Tofino: 11 Rules   Domino: 21 Rules

Sivaraman, Anirudh, Alvin, Cheung, Mihai, Budiu, Changhoon, Kim, Mohammad, Alizadeh, Hari, Balakrishnan, George, Varghese, Nick, McKeown, Steve, Licking. "Packet Transactions: High-Level Programming for Line-Rate Switches." *Proceedings of the 2016 ACM SIGCOMM Conference*. Association for Computing Machinery, 2016.

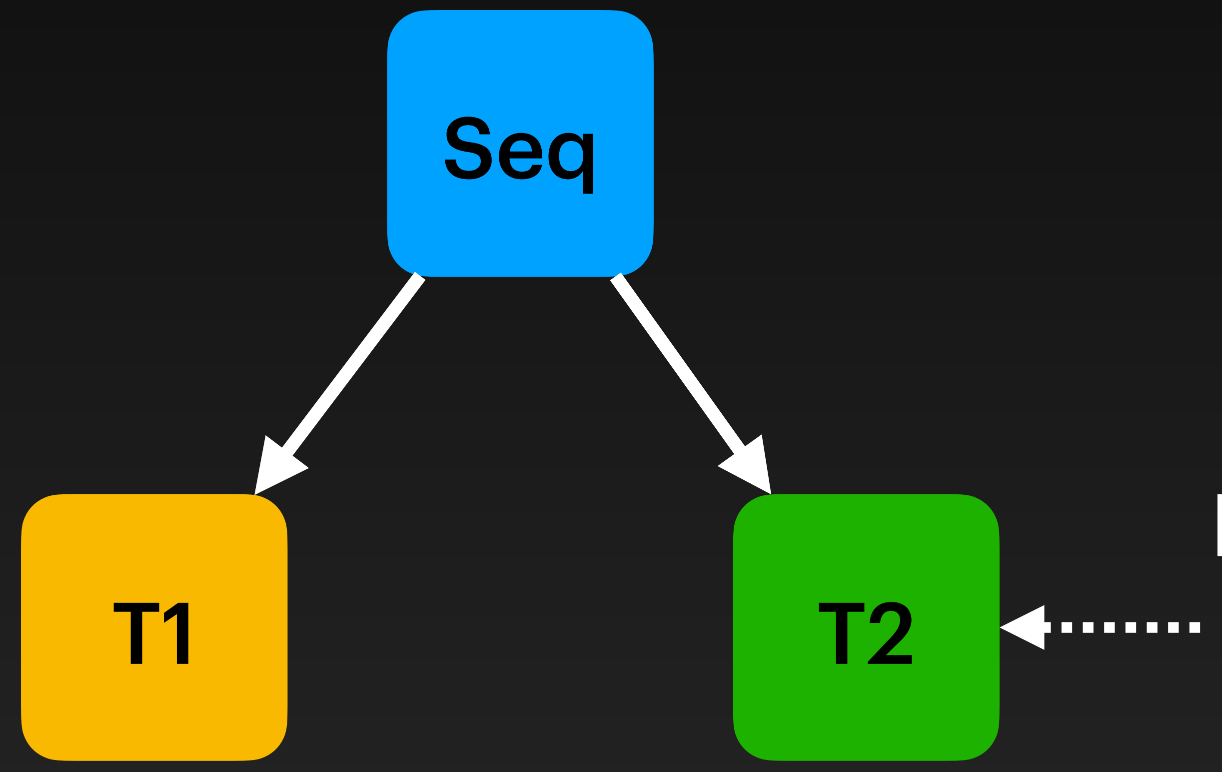
Table transformations

Goals:

- Explores different topological orders of applying tables
- Parallelizing table placements
- Decomposing computations
- Eliminate table dependencies

Table transformations

Decomposing computations

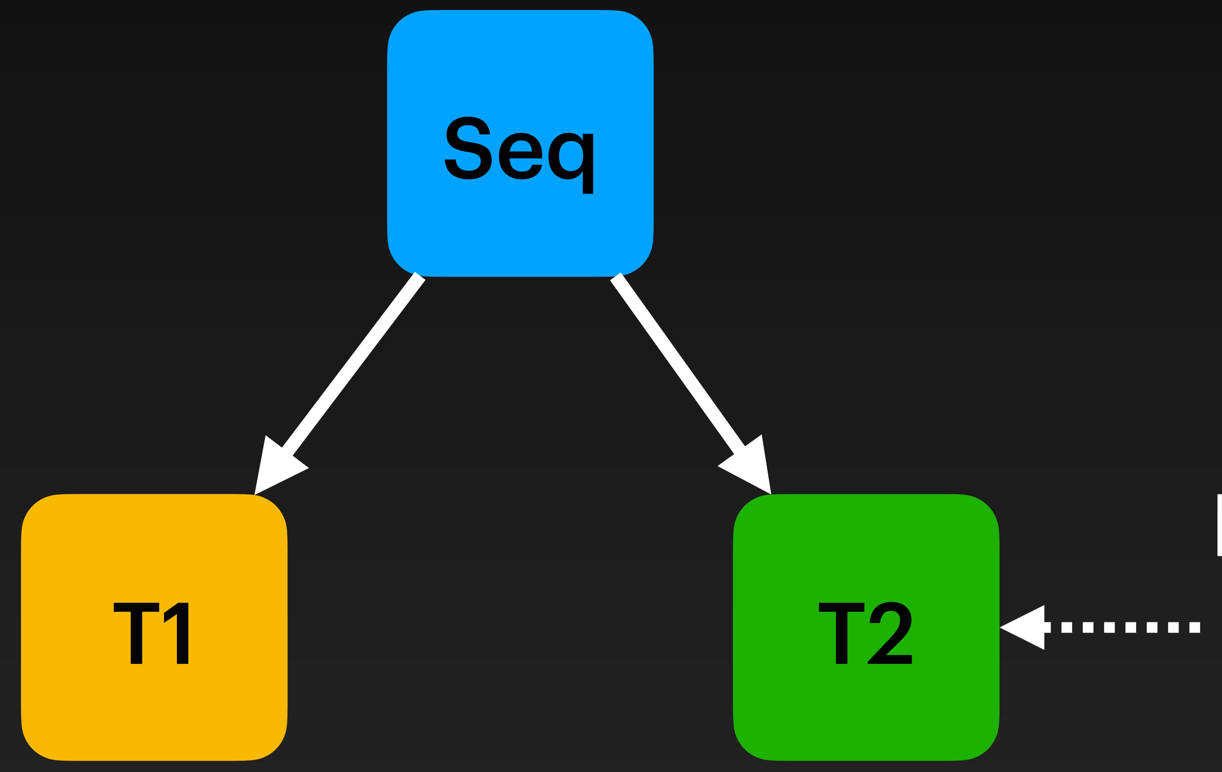


hdr.value = **ite**(hdr.f1 + hdr.f2 + C > hdr.f3,
e1,
e2)

Lift computes with depth > 3

Table transformations

Decomposing computations

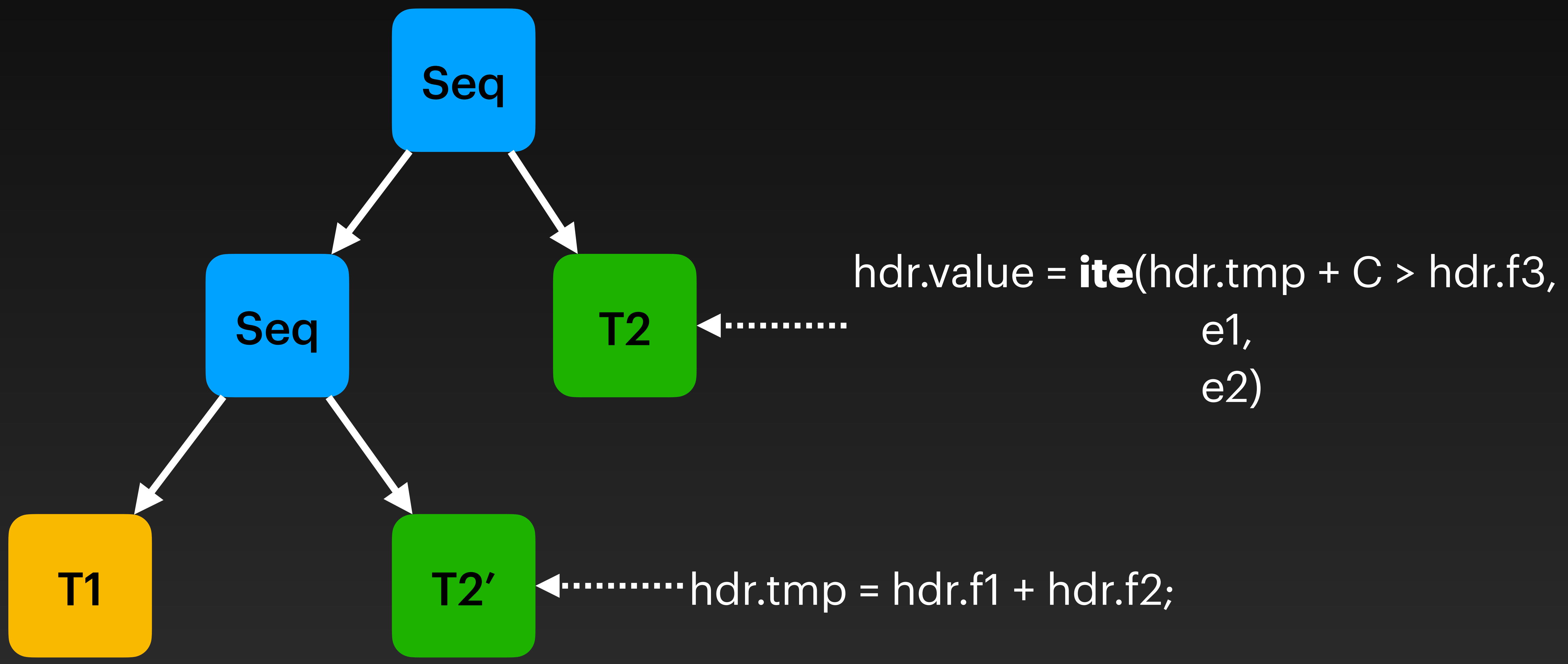



```
hdr.tmp =  hdr.f1 + hdr.f2
hdr.value = ite( hdr.tmp + C > hdr.f3,
                 e1,
                 e2)
```

Lift computes with depth > 3

Table transformations

Decomposing computations



Can be done if split computation does not involve global variables

Synthesis rewrites

Target-dependent rewrite rules

Based on ALU Grammars used for Sketch-guided synthesis in CaT (Gao et al.)

Stateless ALUs

Pure computations

Stateful ALUs

May modify a register file in the ALU (representing global variables)

SKETCH: a Syntax-guided Synthesis-based technique; Program sketches with holes

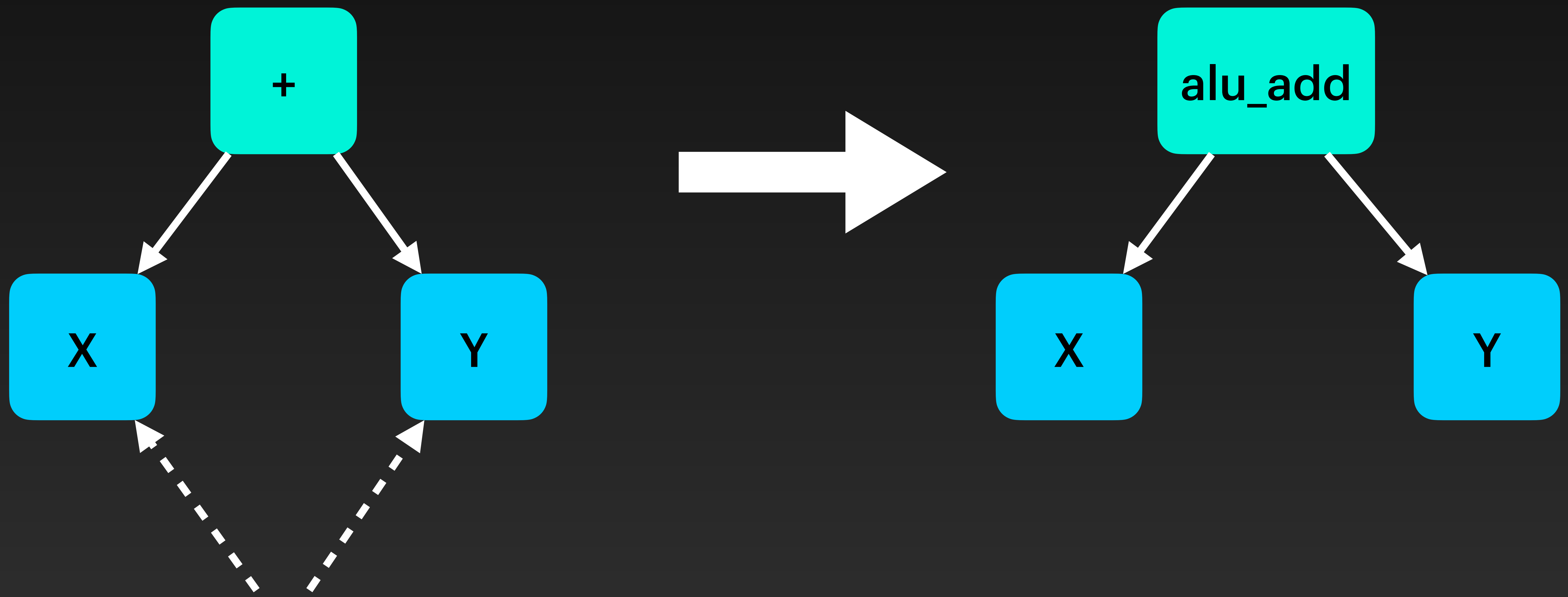
R. Alur *et al.*, "Syntax-guided synthesis," *2013 Formal Methods in Computer-Aided Design*, Portland, OR, USA, 2013, pp. 1-8, doi: 10.1109/FMCAD.2013.6679385.

Solar-Lezama, A. (2009). The Sketching Approach to Program Synthesis. In: Hu, Z. (eds) Programming Languages and Systems. APLAS 2009. Lecture Notes in Computer Science, vol 5904. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-10672-9_3

Synthesis rewrites

Stateless ALUs

Inductively defined based on Sketch grammars



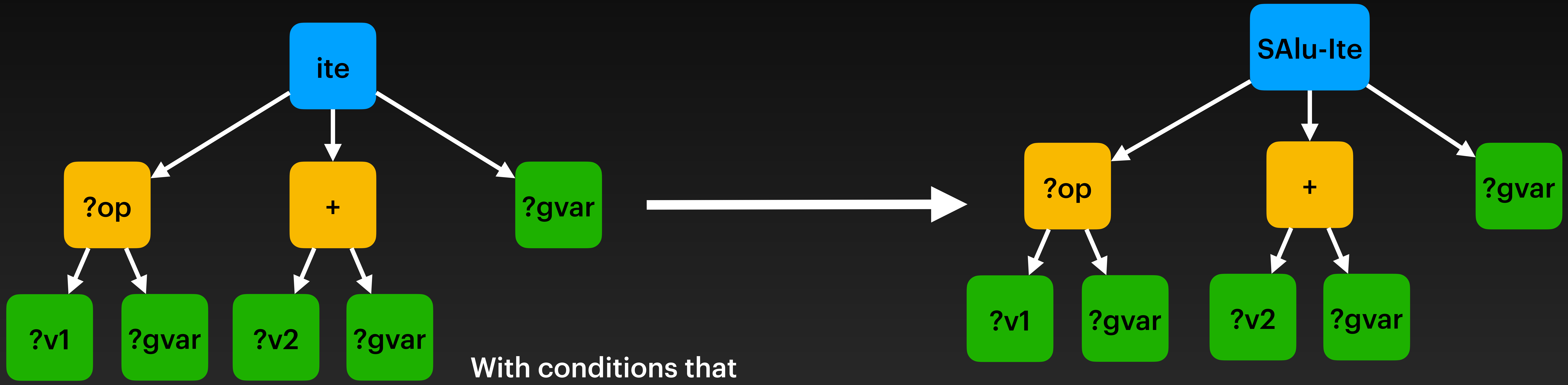
Base Case: X and Y are literals or PHV field variable

Induction Step: X and Y represent stateless ALU computations

Synthesis rewrites

Stateful ALUs

Based on Sketch grammars



With conditions that

1. ?gvar is a global variable or 0
2. ?v1 and ?v2 are PHV fields or constants

Limitations: a global variable is not read/written by two different tables

Rewrite rules

Efficiently explores the space of candidate mappings by composing the rewrite rules via Equality Saturation

General-purpose program transformations

?x + ?y => ?y + ?x
(?x + ?y) + ?z <=> ?x + (?y + ?z)
?x + 0 => ?x
~(?x & ?y) => ~?x | ~?y
?x & ?x => ?x
ite(true, ?x, ?y) => ?x
ite(false, ?x, ?y) => ?y
Etc...

52 Rules

Table Transformations

Table parallelization
Subexpression lifting
Table merging
Etc...

10 Rules

Synthesis rewrites
1-1 to sketch grammars in CaT (Gao et al.)

?x + ?y => **alu_add** ?x ?y
if **mapped**(?x) & **mapped**(?y)

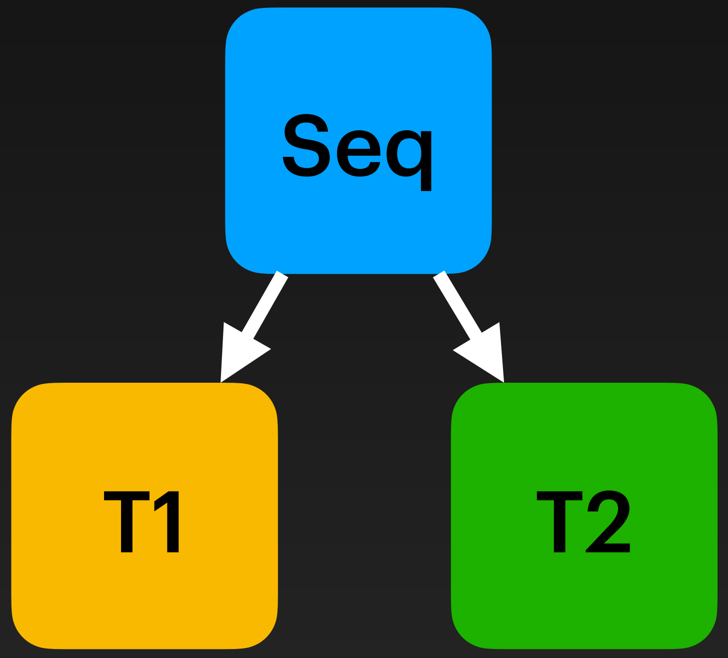
?V = **ite**(?x == ?y, ?x + ?z, ?x) =>
stateful_alu(if, ?V, ?x == ?y, ?x + z, ?x)
if ...

Tofino: 11 Rules   Domino: 21 Rules

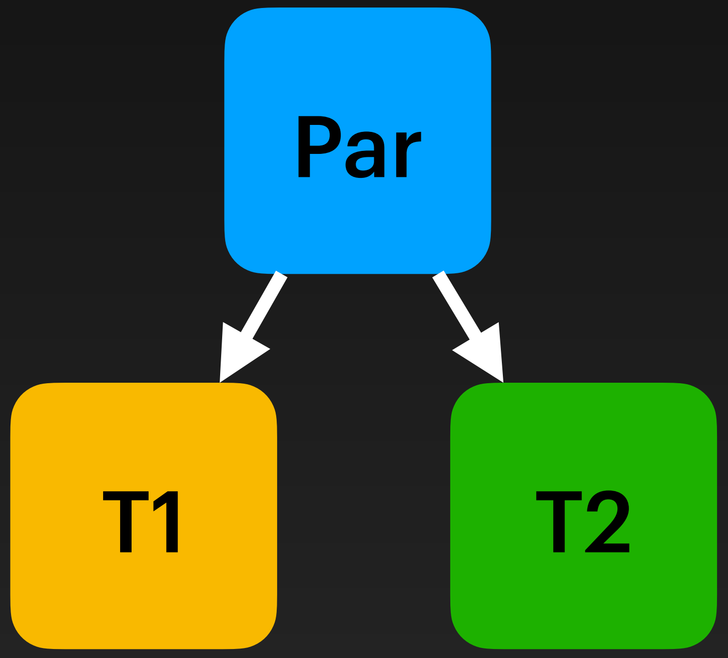
Sivaraman, Anirudh, Alvin, Cheung, Mihai, Budiu, Changhoon, Kim, Mohammad, Alizadeh, Hari, Balakrishnan, George, Varghese, Nick, McKeown, Steve, Licking. "Packet Transactions: High-Level Programming for Line-Rate Switches." *Proceedings of the 2016 ACM SIGCOMM Conference*. Association for Computing Machinery, 2016.

Extraction

Goal: Extract min-depth computation tree



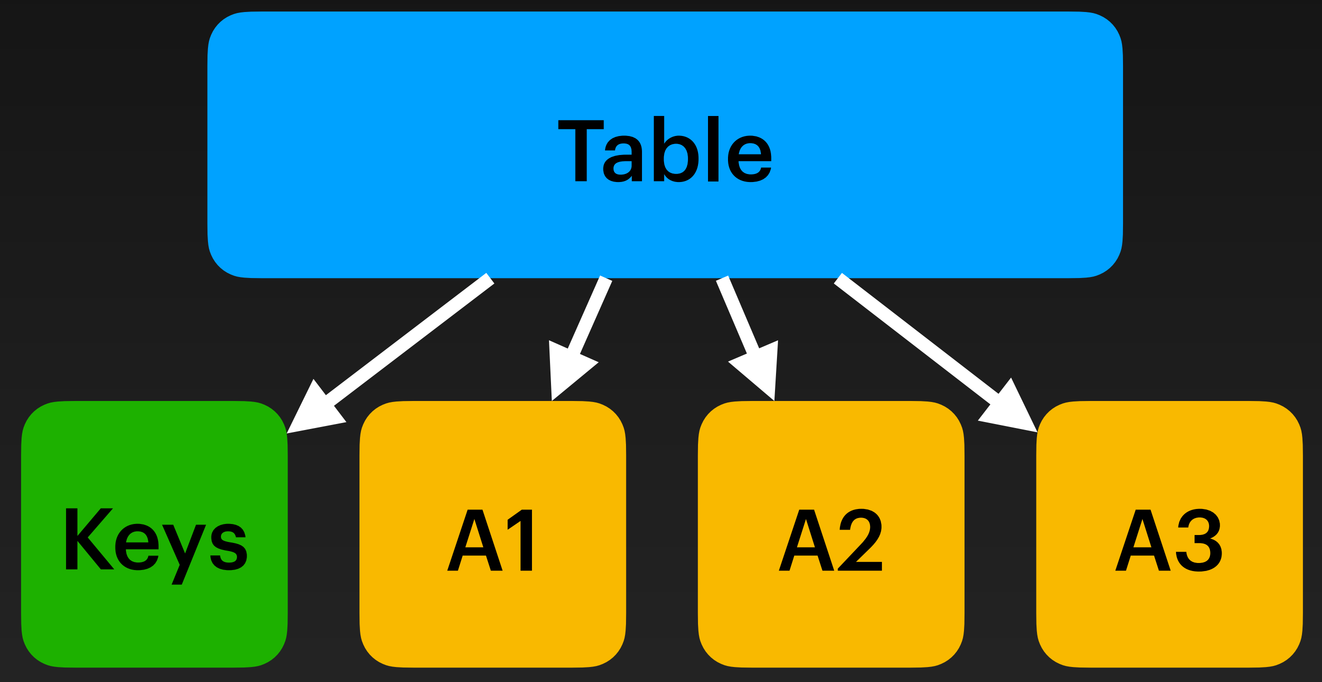
$\text{cost}(T_1) + \text{cost}(T_2)$



$\max\left(\text{cost}(T_1), \text{cost}(T_2)\right)$

Extraction

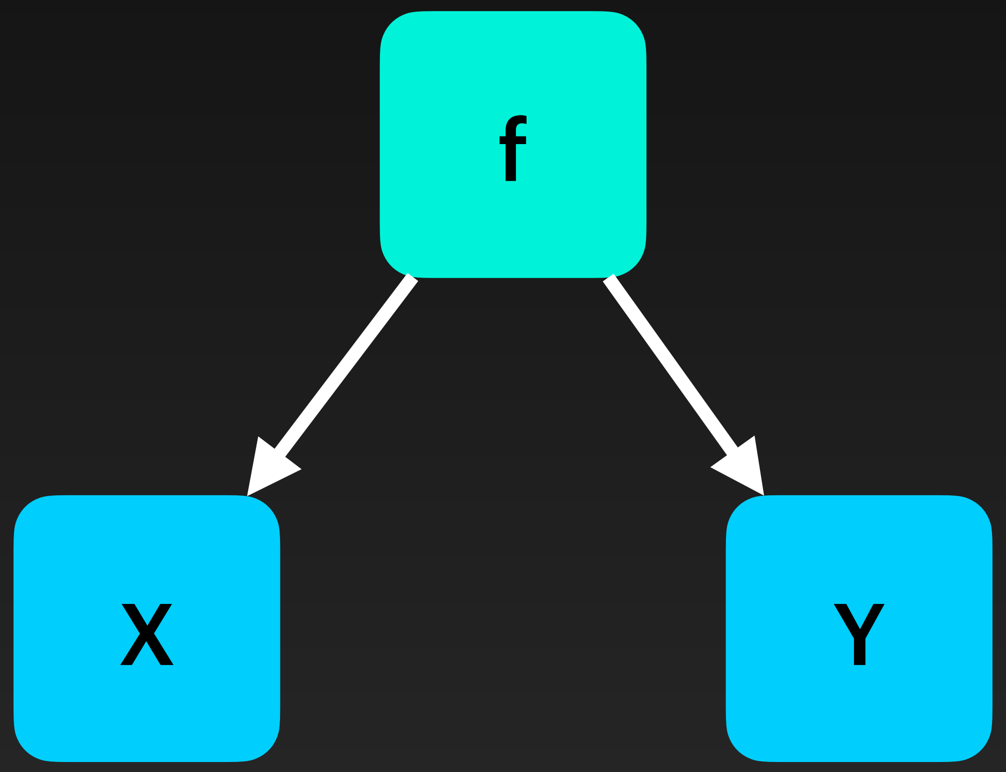
Goal: Extract min-depth computation tree



$$\max_i \left(\text{Cost}(A_i)\right)$$

Extraction

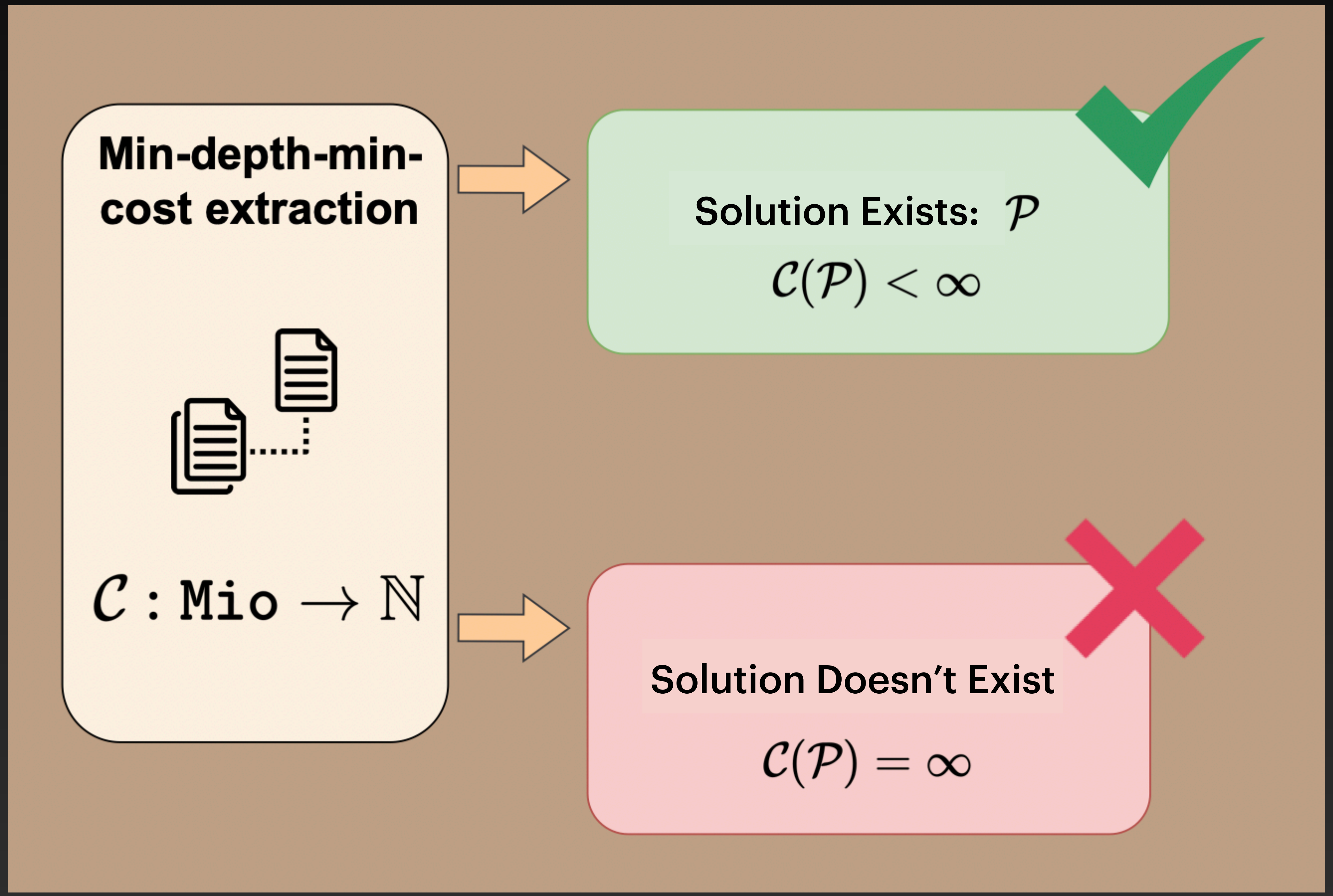
Goal: Extract min-depth computation tree



$$\text{ite}\left(\text{mapped}(f(X, Y)), \max\left(\text{Cost}(X), \text{Cost(Y)}\right) + 1, \infty\right)$$

Only allow extracting computations that are already mapped to target backends

Extraction



$\mathscr{C}(\mathscr{P})$ = Minimum number of stages required to map $\mathscr{P}$

Evaluations

RQ1: Efficiency of CatsTail: synthesis time compared with the previous work CaT (Gao et al.)

RQ2: Efficacy of CatsTail: stage utilization compared with CaT

RQ3: Does the extraction always succeed?

Evaluations

RQ1: Efficiency of CatsTail: synthesis time compared with the previous work CaT (Gao et al.)

Experiments setup:

Target Backends: Intel Tofino and Domino (Banzai) ALUs

Input programs: 8 P4 programs with real-word applications, including:

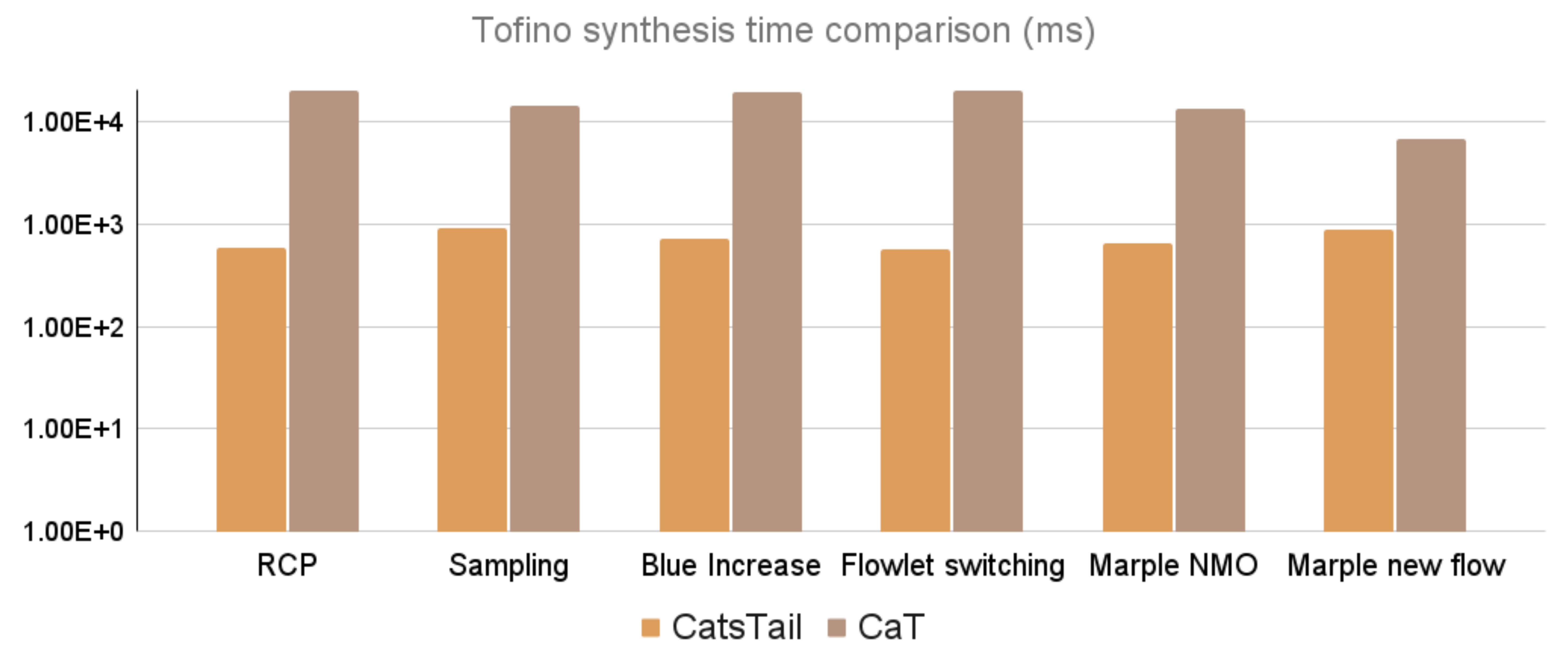
Rate control protocol, Packet sampling, Flowlet Switching, Stateful firewall,Blue increase/decrease, Marple flow

Rewrite Rules:
For the Tofino backend, we enable all the synthesis rewrite
For the Domino backend, we ran two sets of experiments:

1. Full: All synthesis rewrite rules
2. Sk: synthesis rewrite rules corresponding to the sketch grammar CaT used in their benchmark

Evaluations

RQ1: Efficiency of CatsTail: synthesis time compared with the previous work CaT (Gao et al.)



(b) Synthesis time comparison for Intel Tofino ALUs

CatsTail | CaT

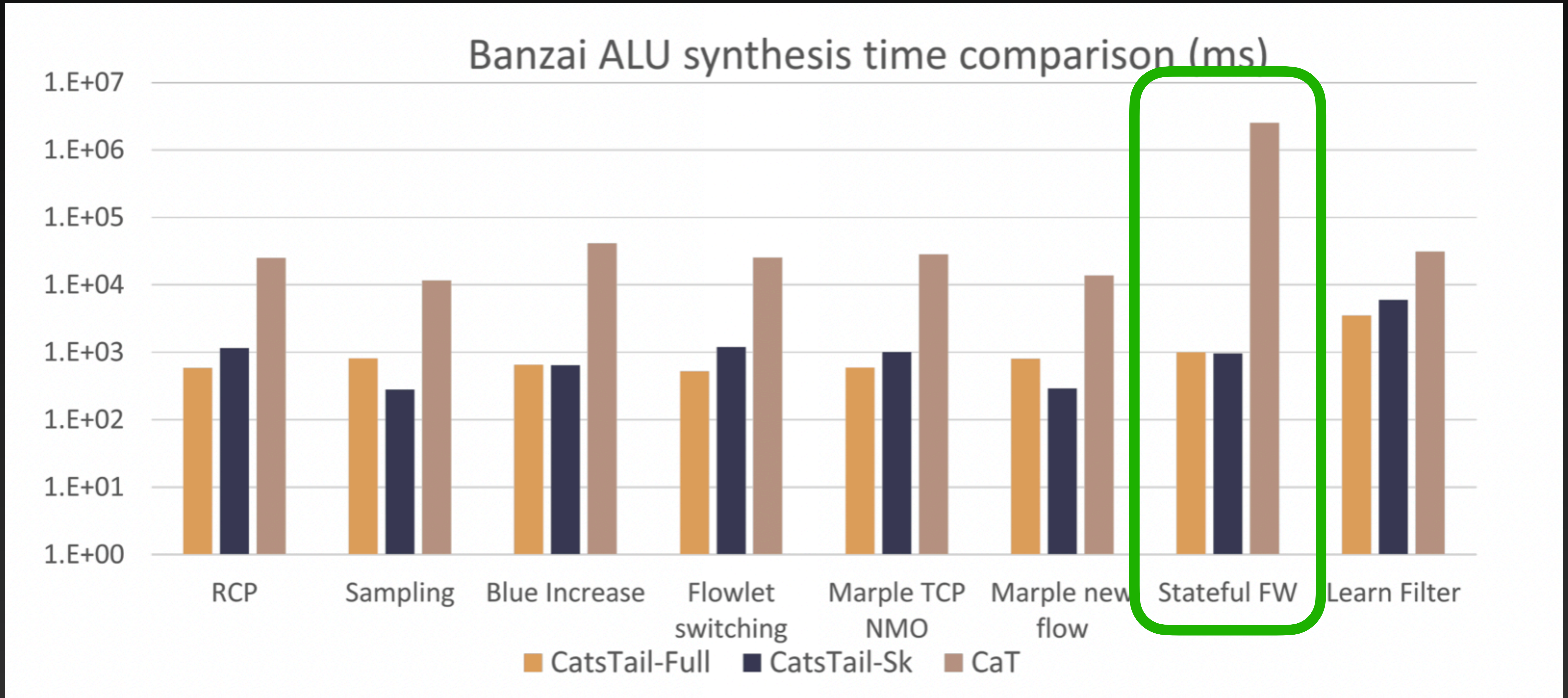
~an order of magnitude faster in synthesis

X: Benchmark cases.
Y: Synthesis time (ms), in log-scale

Successfully synthesized

Evaluations

RQ1: Efficiency of CatsTail: synthesis time compared with the previous work CaT (Gao et al.)



Successfully synthesized

Orders of magnitude faster

CatsTail-Full

CatsTail ran with all rewrite rules

CatsTail-Sk

Similar to CatsTail-Full except the synthesis rules only include those corresponds to sketches used in CaT

CaT

CaT synthesis time

Evaluations

RQ2: Efficacy of CatsTail: stage utilization compared with CaT

Table 1. Comparison of the number of stages required to map the synthesized program given by CatsTail and CaT [Gao et al. 2023] to Intel Tofino switches and Domino switches.

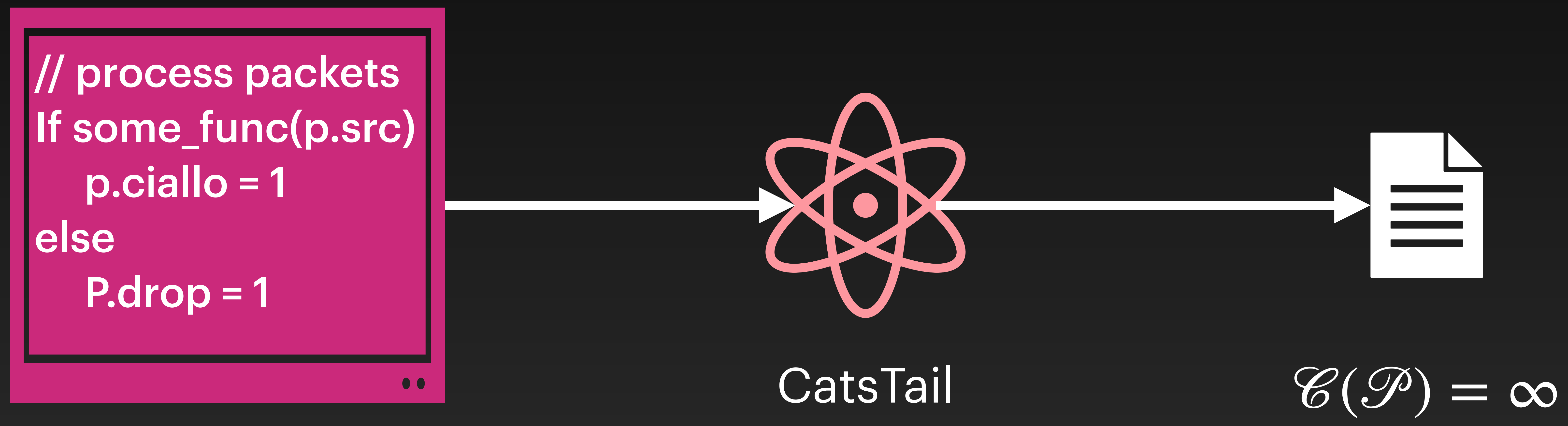
Benchmark	# Stages on Domino		# Stages on Tofino	
	CatsTail	CaT	CatsTail	CaT
RCP	2	2	1	1
Sampling	2	2	1	1
Blue Increase	4	4	1	1
Flowlet Switching	3	3	2	2
Marple Flow NMO	2	3	2	2
Marple New Flow	2	2	1	1
Stateful Firewall	4	4	-	-
Learn Filter	3	3	-	-

Same numbers of stage utilization

Nested ifs not supported by Tofino switch

Evaluations

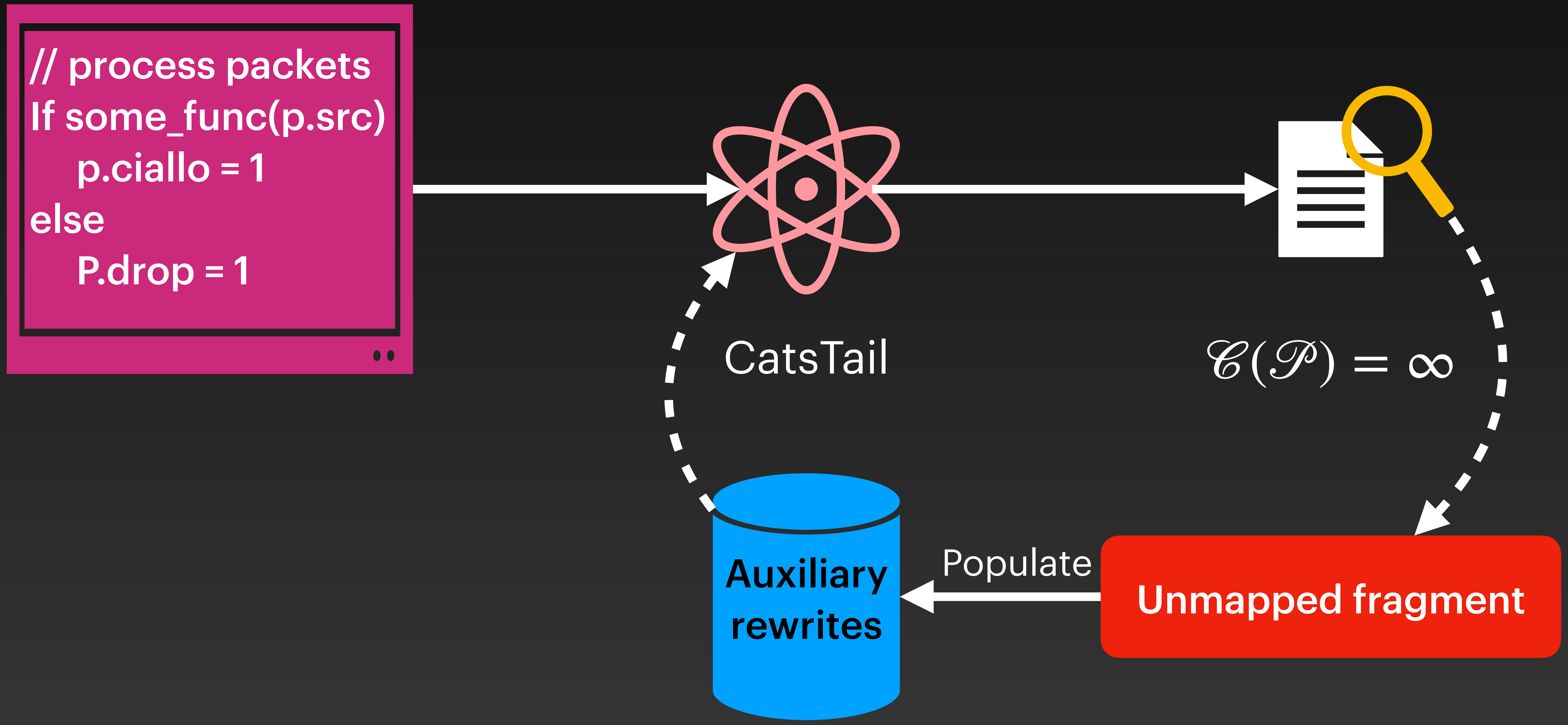
RQ3: Does the extraction always succeed?



Incompleteness of general purpose / table transformation rules

Evaluations

RQ3: Does the extraction always succeed?



Evaluations

RQ1: Efficiency of CatsTail: synthesis time compared with the previous work CaT (Gao et al.)

Orders of magnitude faster compared with CaT, thanks to the scalability of egg

RQ2: Efficacy of CatsTail: stage utilization compared with CaT

Stage utilization is as good as CaT

RQ3: Does the extraction always succeed?

No, but we can work around

Report

https://www.cs.princeton.edu/~dh7120/assets/papers/COS539Report.pdf

Prototype

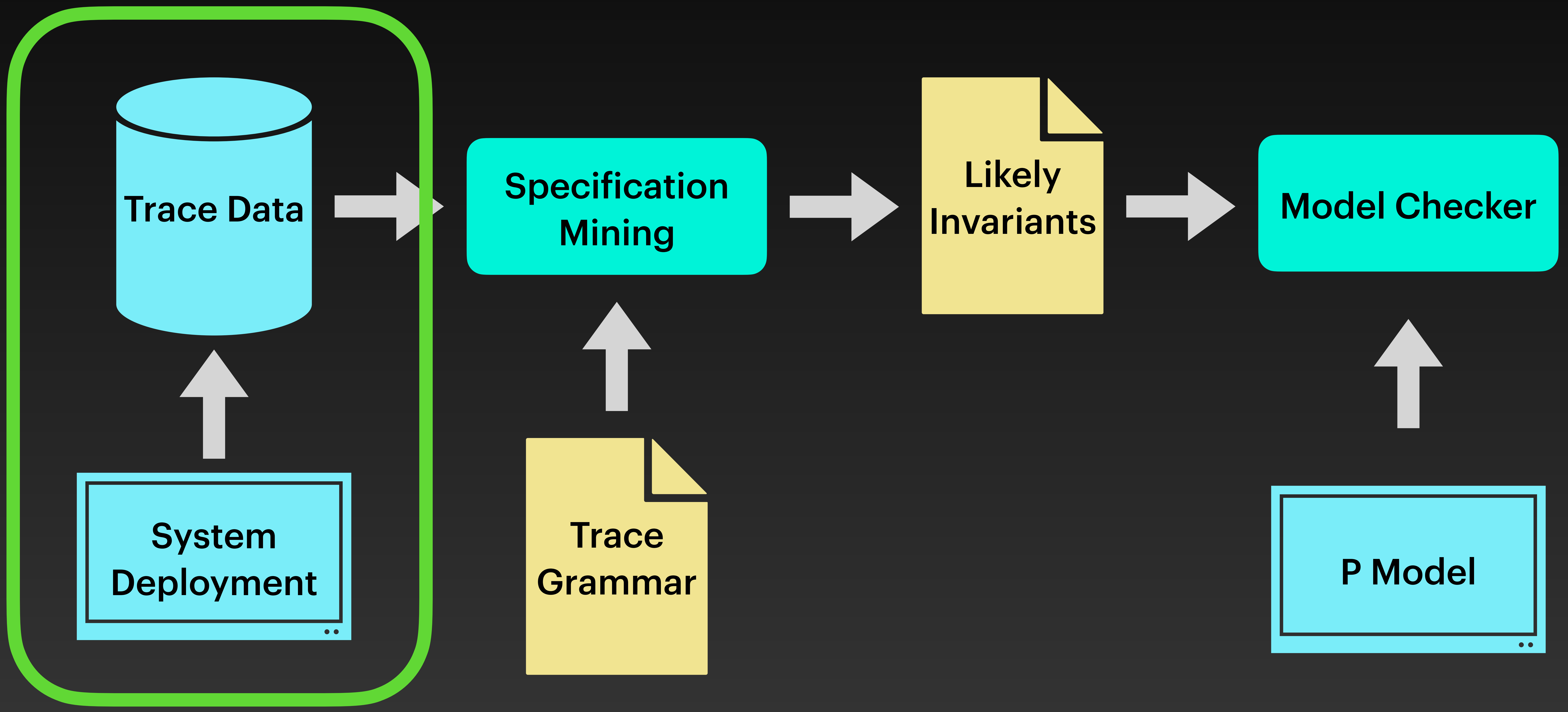
https://github.com/AD1024/CatsTail/

Outline

1. Brief introduction to equality saturation
2. Term Extraction for equality saturation **(Part A)**
3. Applying equality saturation for network resource synthesis **(Part B)**
4. (If time permits) Ongoing project of invariant synthesis for distributed systems

Recent project: PInfer

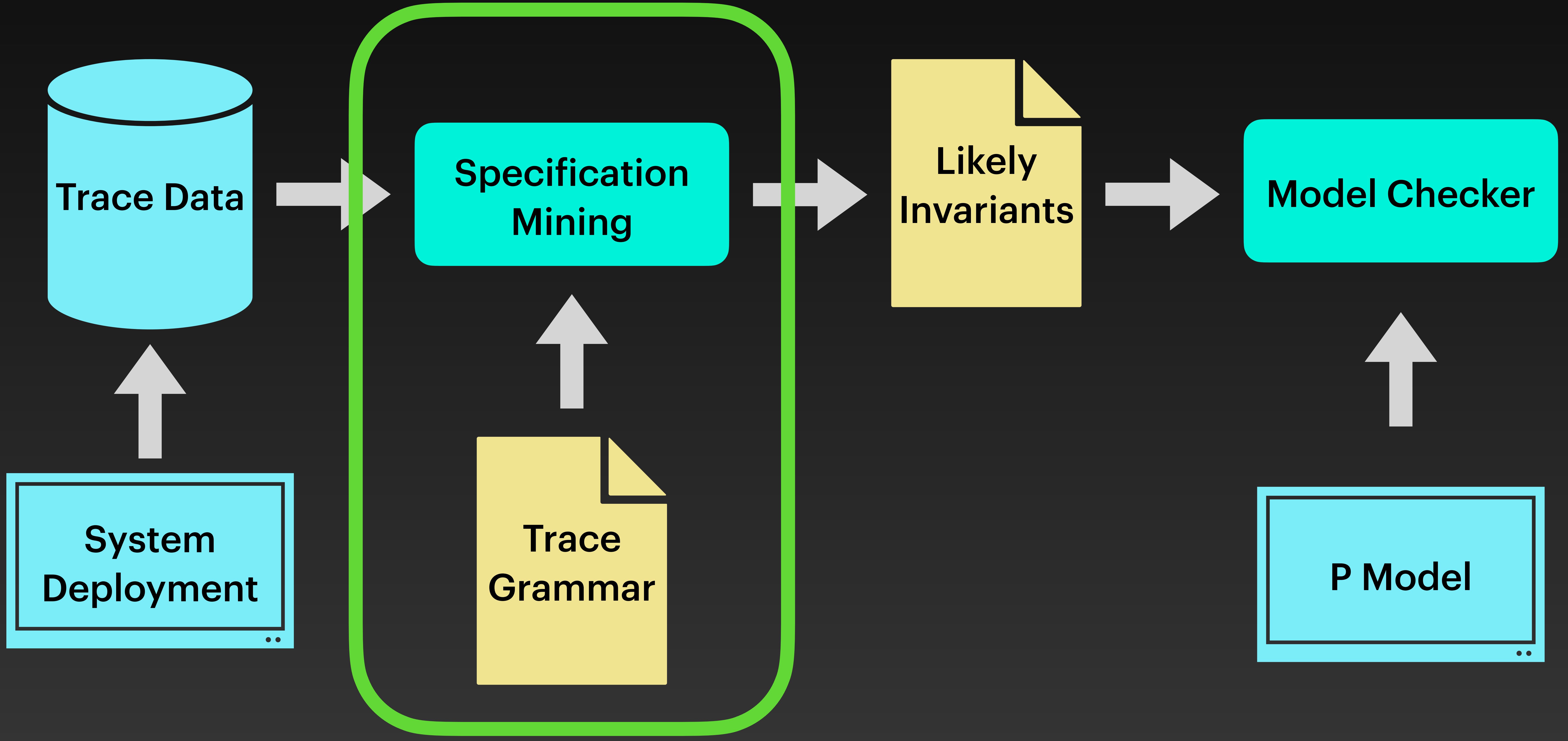
Learning invariants for distributed systems from traces



Desai, Ankush, Vivek, Gupta, Ethan, Jackson, Shaz, Qadeer, Sriram, Rajamani, Damien, Zufferey. "P: safe asynchronous event-driven programming." *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. Association for Computing Machinery, 2013.

Recent project: PInfer

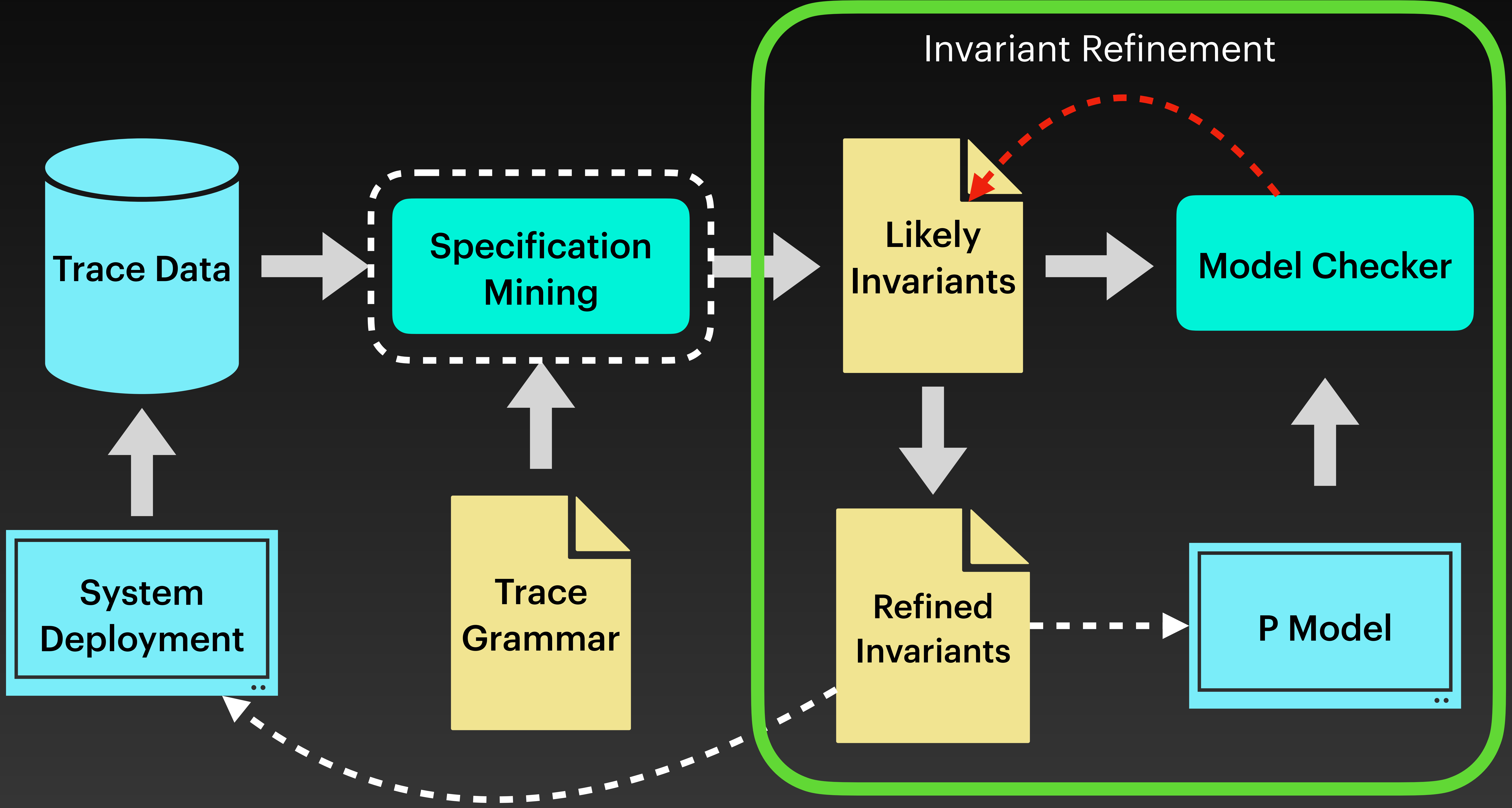
Learning invariants for distributed systems from traces



Desai, Ankush, Vivek, Gupta, Ethan, Jackson, Shaz, Qadeer, Sriram, Rajamani, Damien, Zufferey. "P: safe asynchronous event-driven programming." *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. Association for Computing Machinery, 2013.

Recent project: PInfer

Learning invariants for distributed systems from traces



Desai, Ankush, Vivek, Gupta, Ethan, Jackson, Shaz, Qadeer, Sriram, Rajamani, Damien, Zufferey. "P: safe asynchronous event-driven programming." *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. Association for Computing Machinery, 2013.

Invariant learning: Related works

Protocol Definition +
Invariants checking

Enumerate combinations of
predicates and connectives

Ivy

DistAI

DuoAI

SWISS

PInfer

Invariants about
messages / events

Invariants
about states (Ivy-style)

K. McMillan, O. Padon, "Ivy: A Multi-modal Verification Tool for Distributed Algorithms," in Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part II, 2020, pp. 190–202.

Travis Hance, Marijn Heule, Ruben Martins, Bryan Parno. "Finding Invariants of Distributed Systems: It's a Small (Enough) World After All." 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21). USENIX Association, 2021.

Jianan Yao, Runzhou Tao, Ronghui Gu, Jason Nieh. "DuoAI: Fast, Automated Inference of Inductive Invariants for Verifying Distributed Protocols." 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). USENIX Association, 2022.

Jianan Yao, Runzhou Tao, Ronghui Gu, Jason Nieh, Suman Jana, Gabriel Ryan. "DistAI: Data-Driven Automated Invariant Learning for Distributed Protocols." 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21). USENIX Association, 2021.

Invariant learning

PInfer

Challenges:

1. Huge search space: many valid predicates over events and payloads

Brute-force enumeration leads to vacuously true/false invariants, which are not useful for production systems

Trace Grammar that focuses useful predicates

2. Efficiency: enumerating logical connectives is computationally computationally intractable

Formulate invariant learning as a boolean function learning problem

Q & A