



CSCE 585: Machine Learning Systems

Lecture 6: Understanding and Explaining the Root Causes of Performance Faults in ML Systems

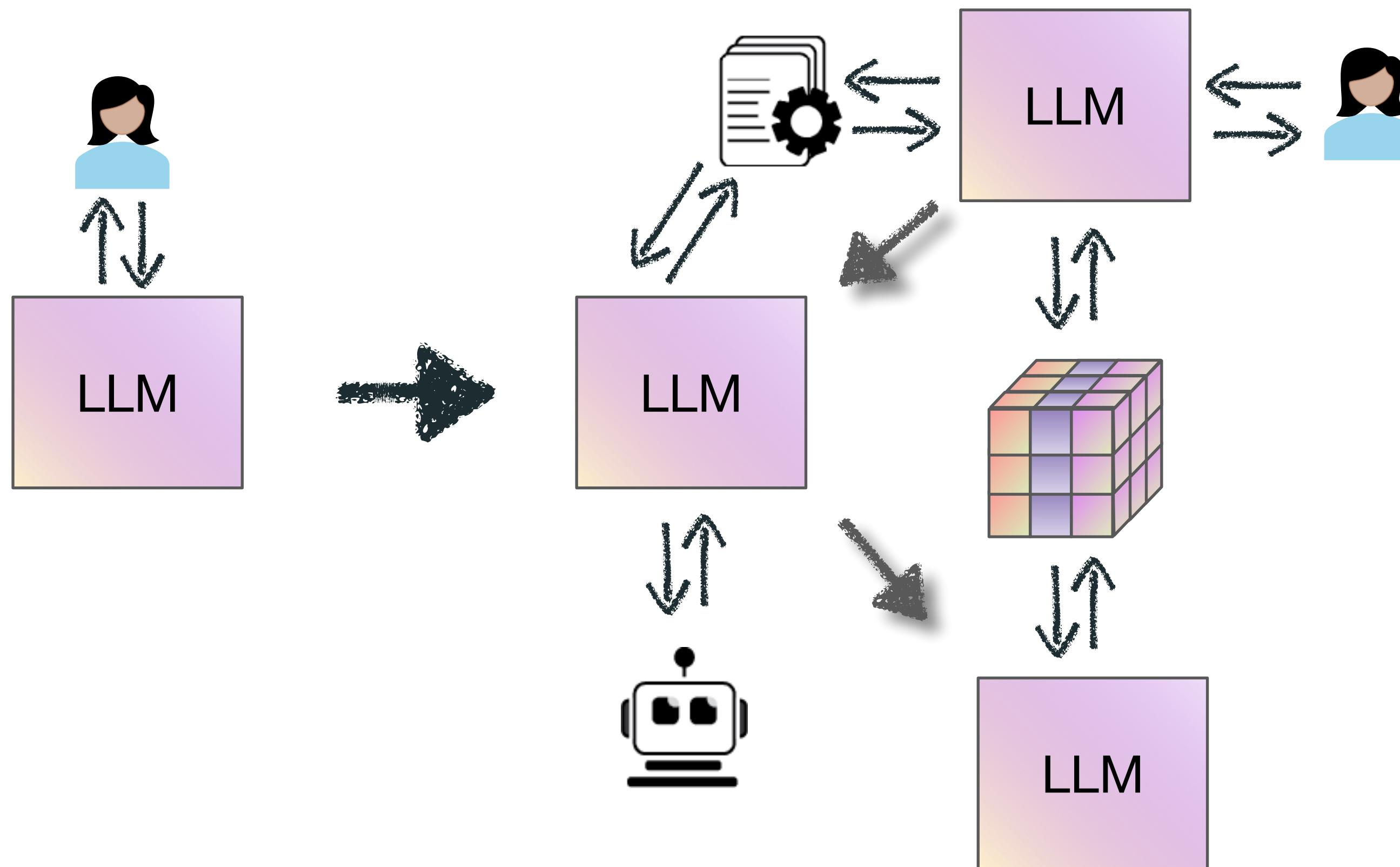
Pooyan Jamshidi

Case Study 1

LLM Systems

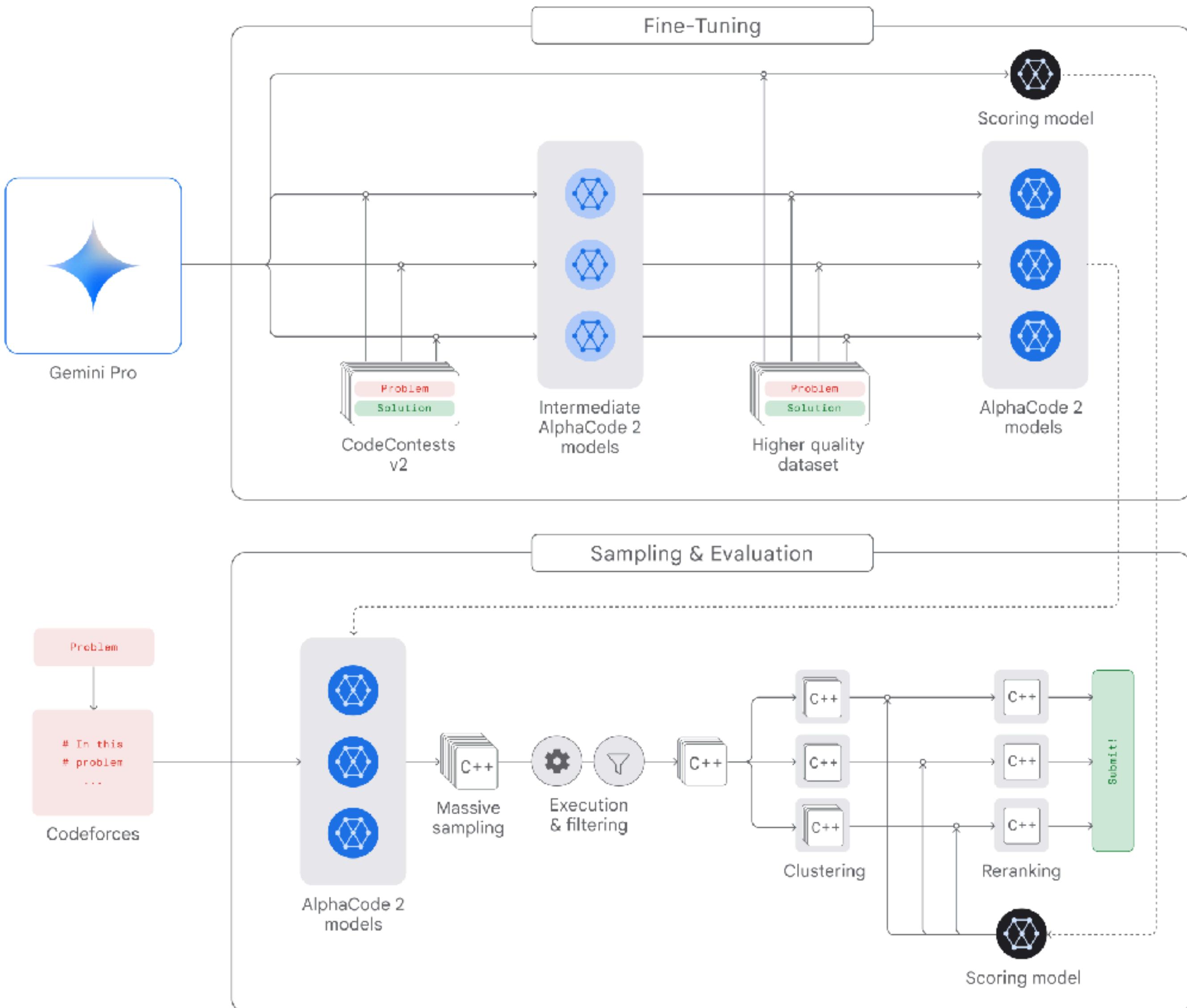


State-of-the-art AI results are increasingly obtained by systems composed of multiple components, not just monolithic models



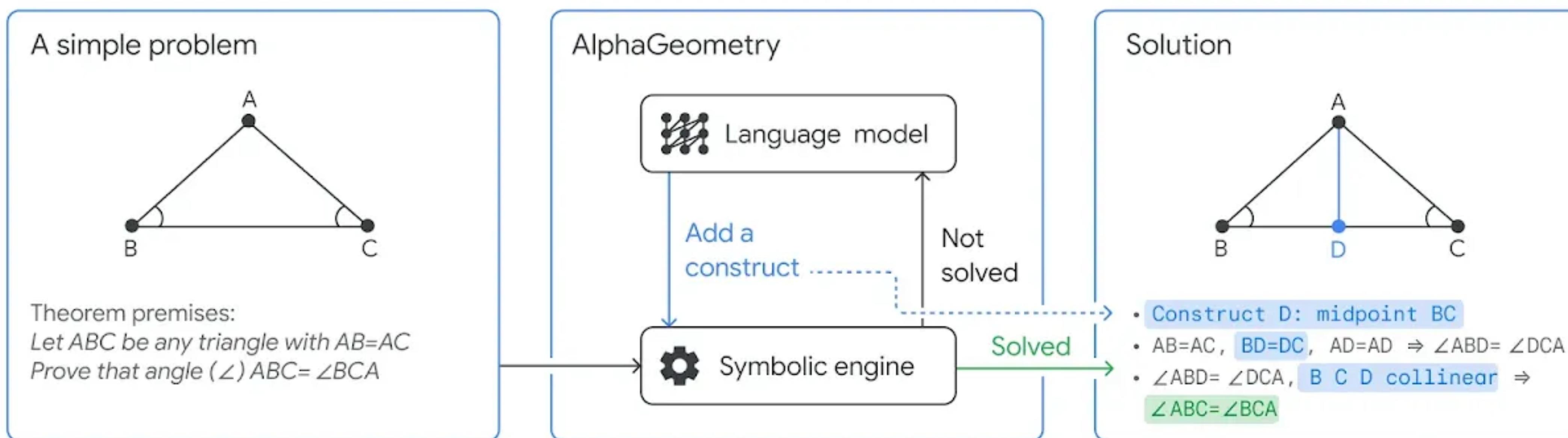
State-of-the-art AI results are increasingly obtained by systems composed of multiple components, not just monolithic models

Google's AlphaCode 2 set state-of-the-art results in programming through a carefully engineered system that uses LLMs to **generate** up to 1 million possible solutions for a task and then **filter** down the set.



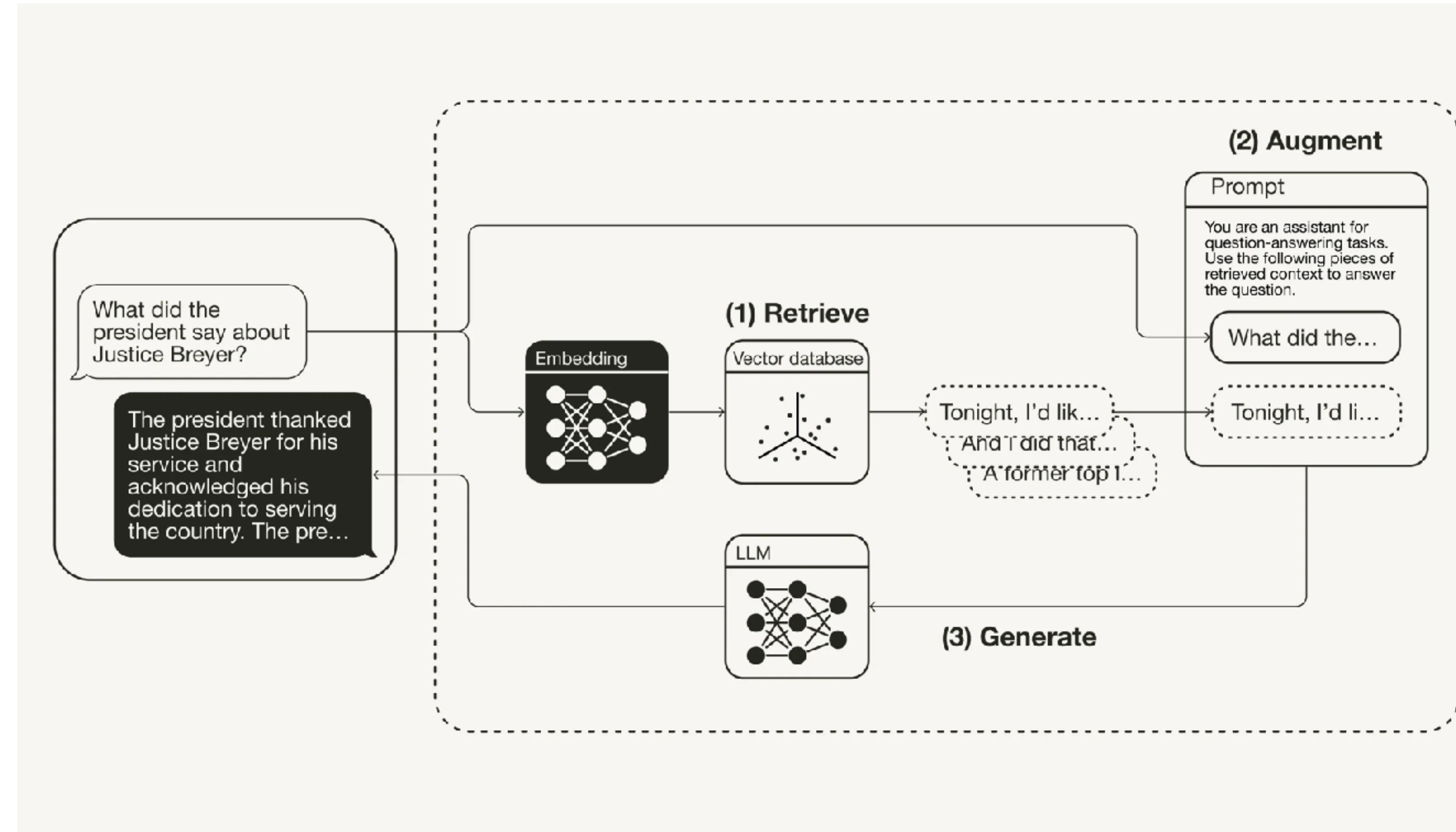
State-of-the-art AI results are increasingly obtained by systems composed of multiple components, not just monolithic models

AlphaGeometry combines an LLM with a traditional **symbolic solver** to tackle Olympiad problems.



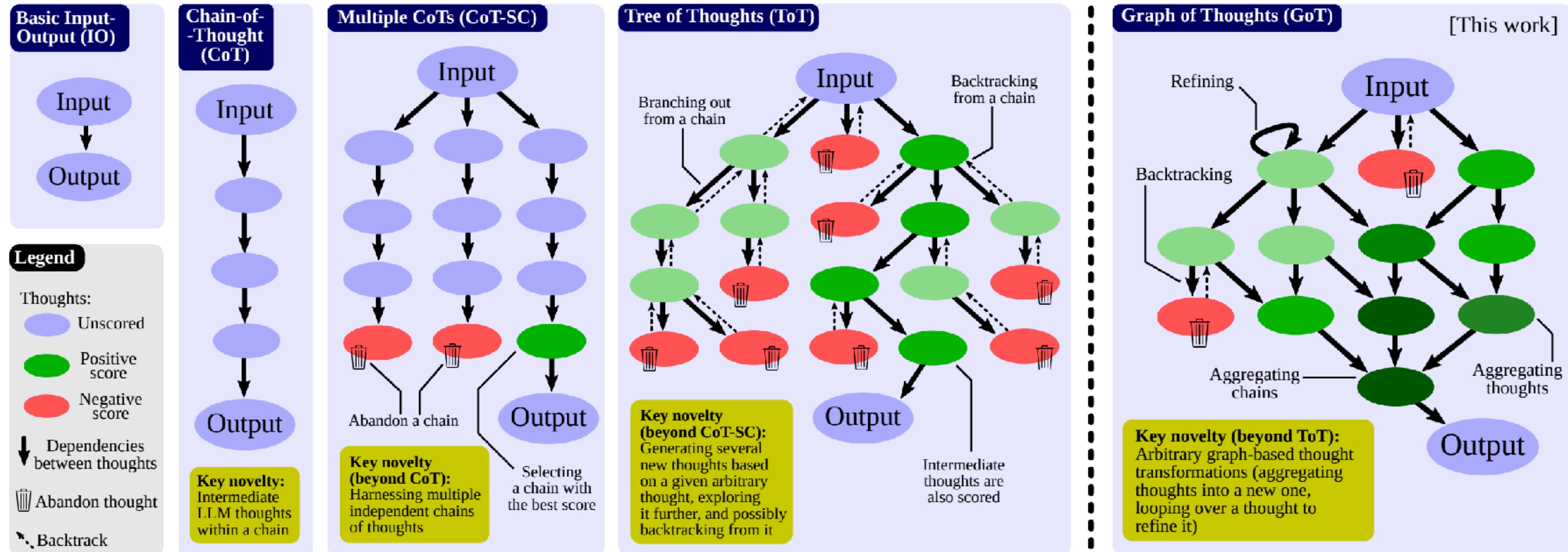
State-of-the-art AI results are increasingly obtained by systems composed of multiple components, not just monolithic models

~60% of LLM applications use some form of **retrieval-augmented generation (RAG)**



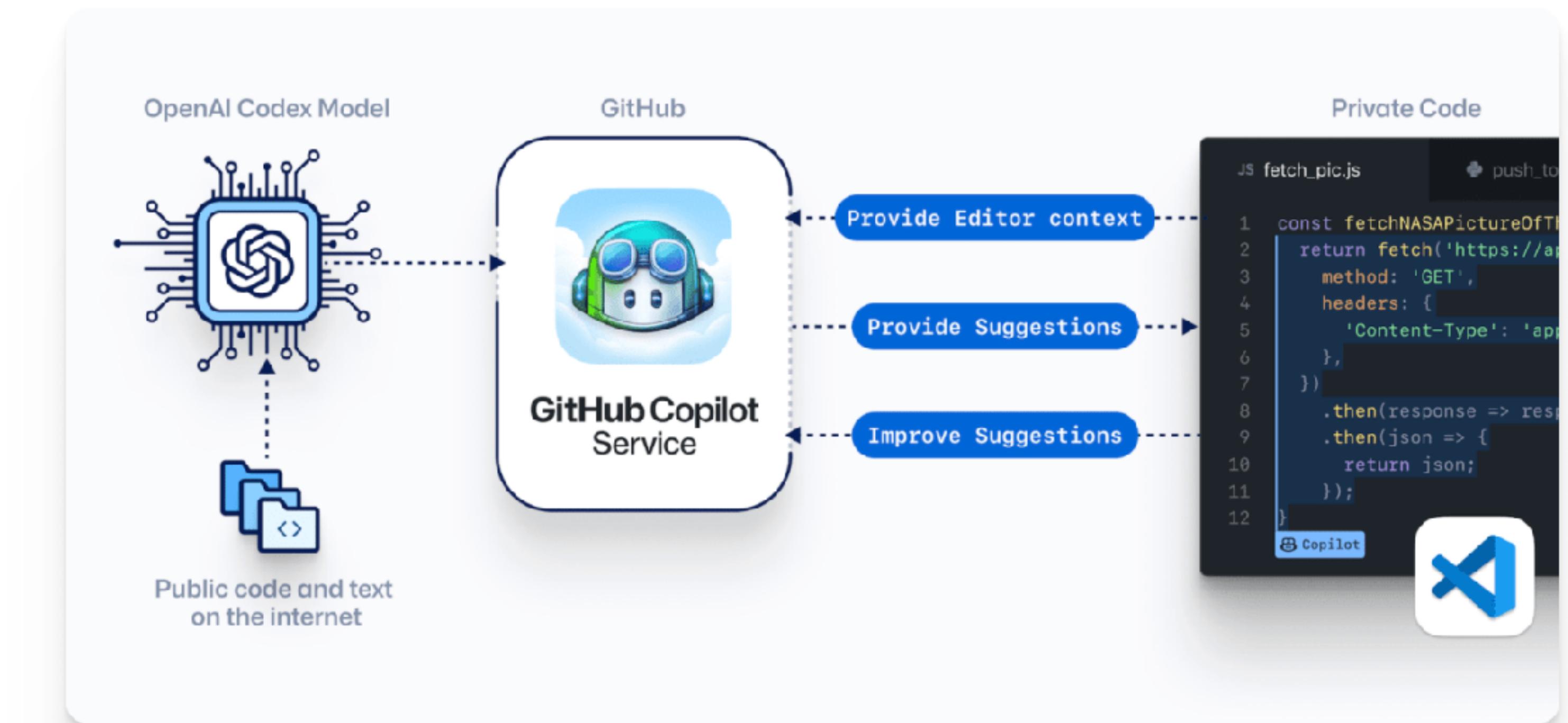
State-of-the-art AI results are increasingly obtained by systems composed of multiple components, not just monolithic models

...and 30% use multi-step chains.



State-of-the-art AI results are increasingly obtained by systems composed of multiple components, not just monolithic models

Github Copilot uses carefully tuned smaller models and various search heuristics to provide results.



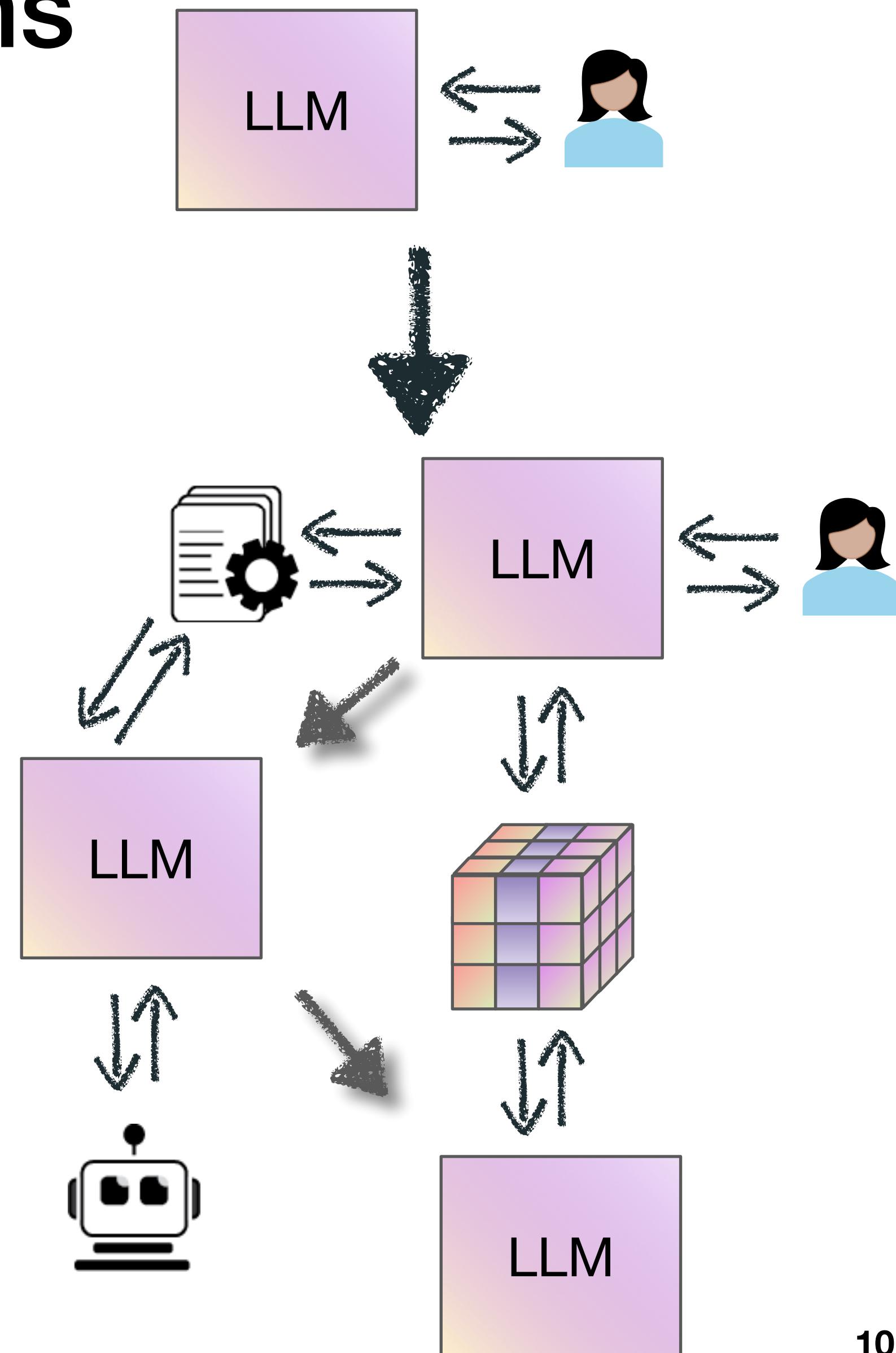
State-of-the-art AI results are increasingly obtained by systems composed of multiple components, not just monolithic models

Google's Gemini launch post measured its MMLU (Massive Multitask Language Understanding) benchmark results using a new CoT@32 inference strategy that calls the model 32 times.

	Gemini Ultra	Gemini Pro	GPT-4	GPT-3.5	PaLM 2-L	Claude 2	Inflection-2	Grok 1	LLAMA-2
MMLU Multiple-choice questions in 57 subjects (professional & academic) (Hendrycks et al., 2021a)	90.04% CoT@32*	79.13% CoT@8*	87.29% CoT@32 (via API**)	70% 5-shot	78.4% 5-shot	78.5% 5-shot CoT	79.6% 5-shot	73.0% 5-shot	68.0%***

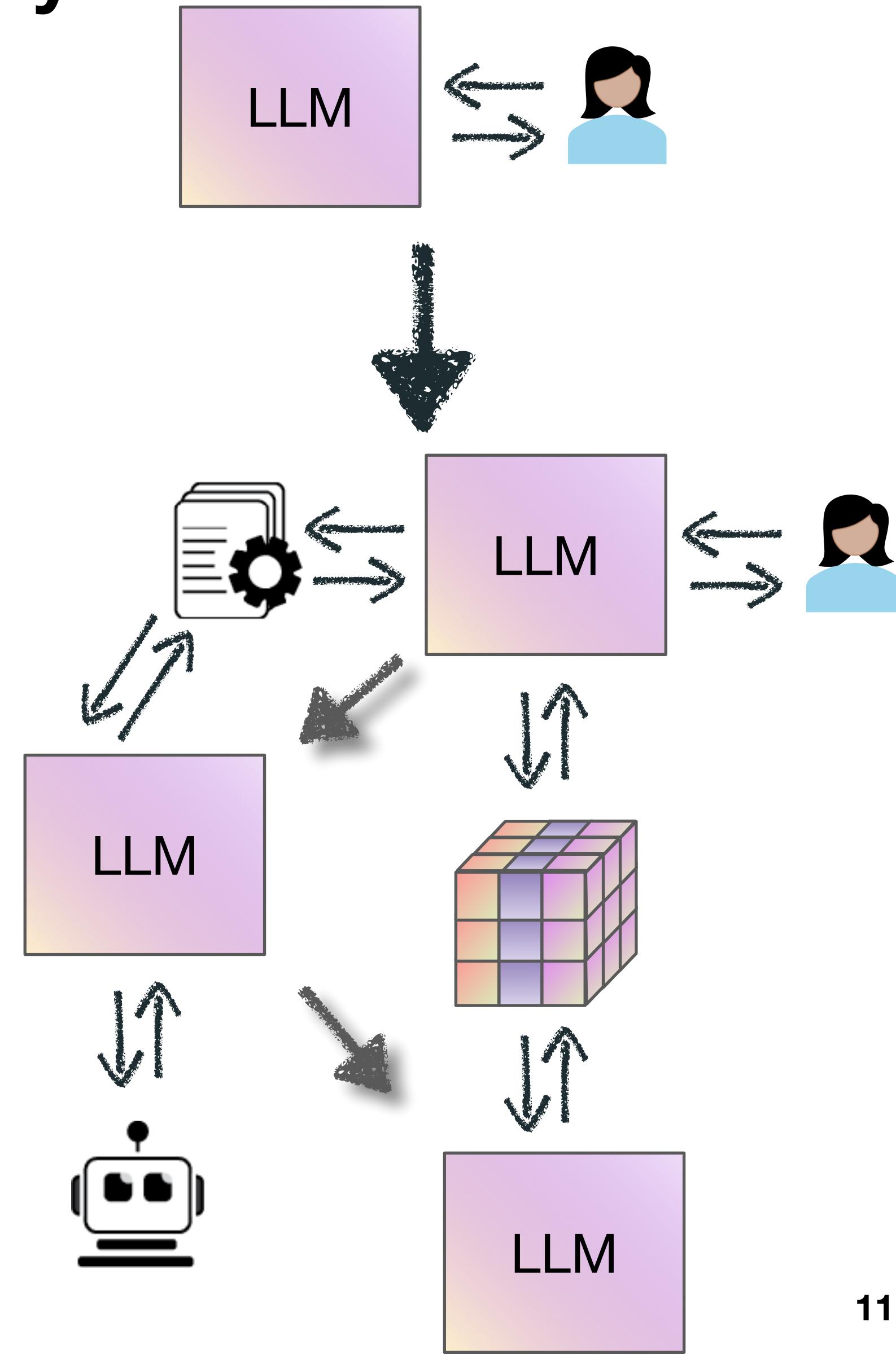
The paradigm shift from monolithic to modular-composed machine learning systems

- **Modular-composed ML Systems** are a class of modern computer systems that tackle AI/ML tasks using:
 - Multiple **interacting** and **interdependent components**,
 - including multiple calls to **models**, **search & retrieval** algorithms, and external **tools**.
- In contrast, **Monolithic ML Systems** are simply traditional ML Systems that call a **statistical model** at the backend.
 - e.g., a **Transformer** that predicts the next token in text.



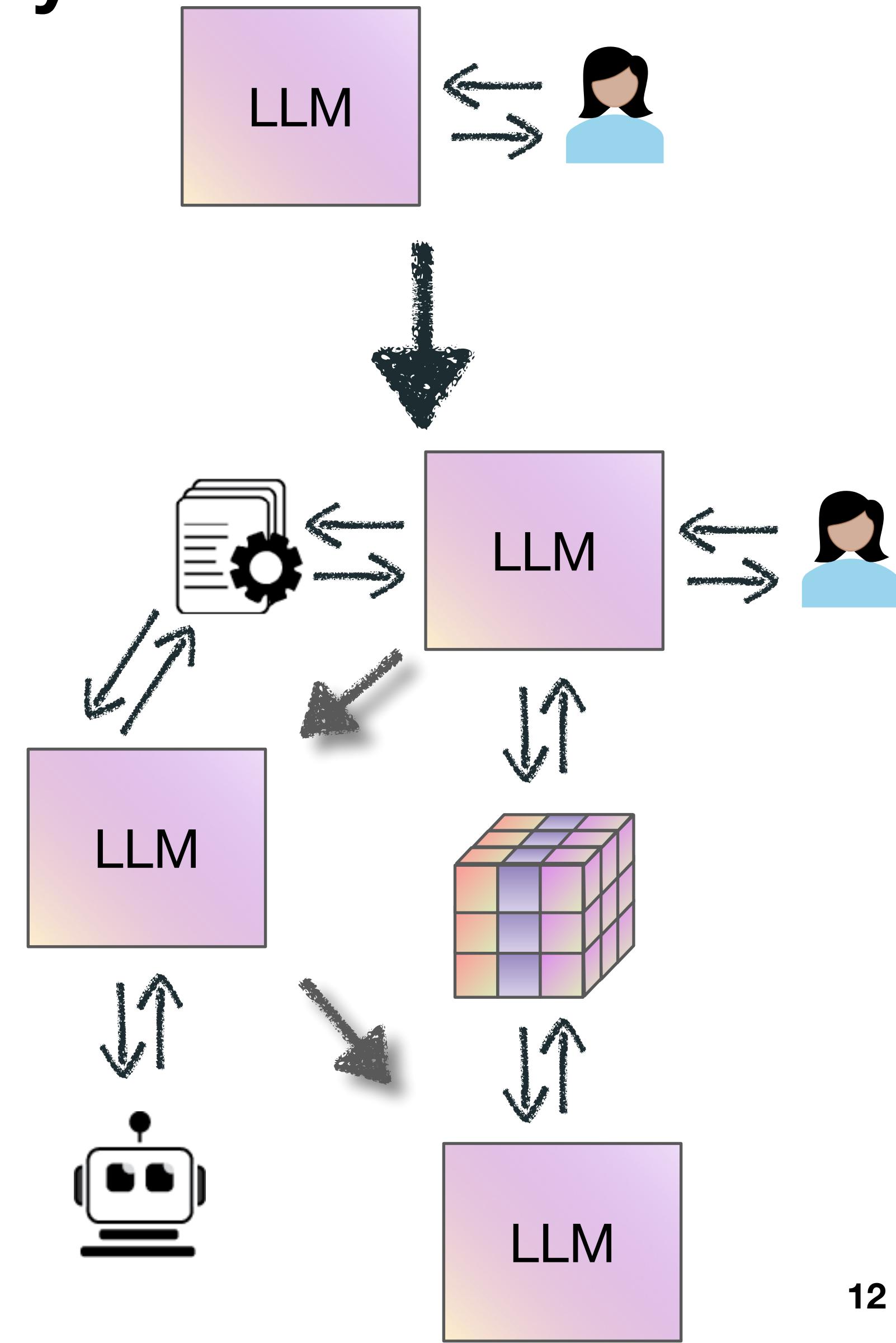
This paradigm shift to modular-composed ML systems opens up new opportunities for computer systems research

- Design space exploration
 - With an SLA of 100 milliseconds for RAG, should we budget to spend 20 ms on the retriever and 80 on the LLM, or the other way around?
- Performance tradeoff and optimization
 - Modular-composed systems contain non-differentiable components.
 - Performance optimization for pipelines of pretrained LLMs and other components.



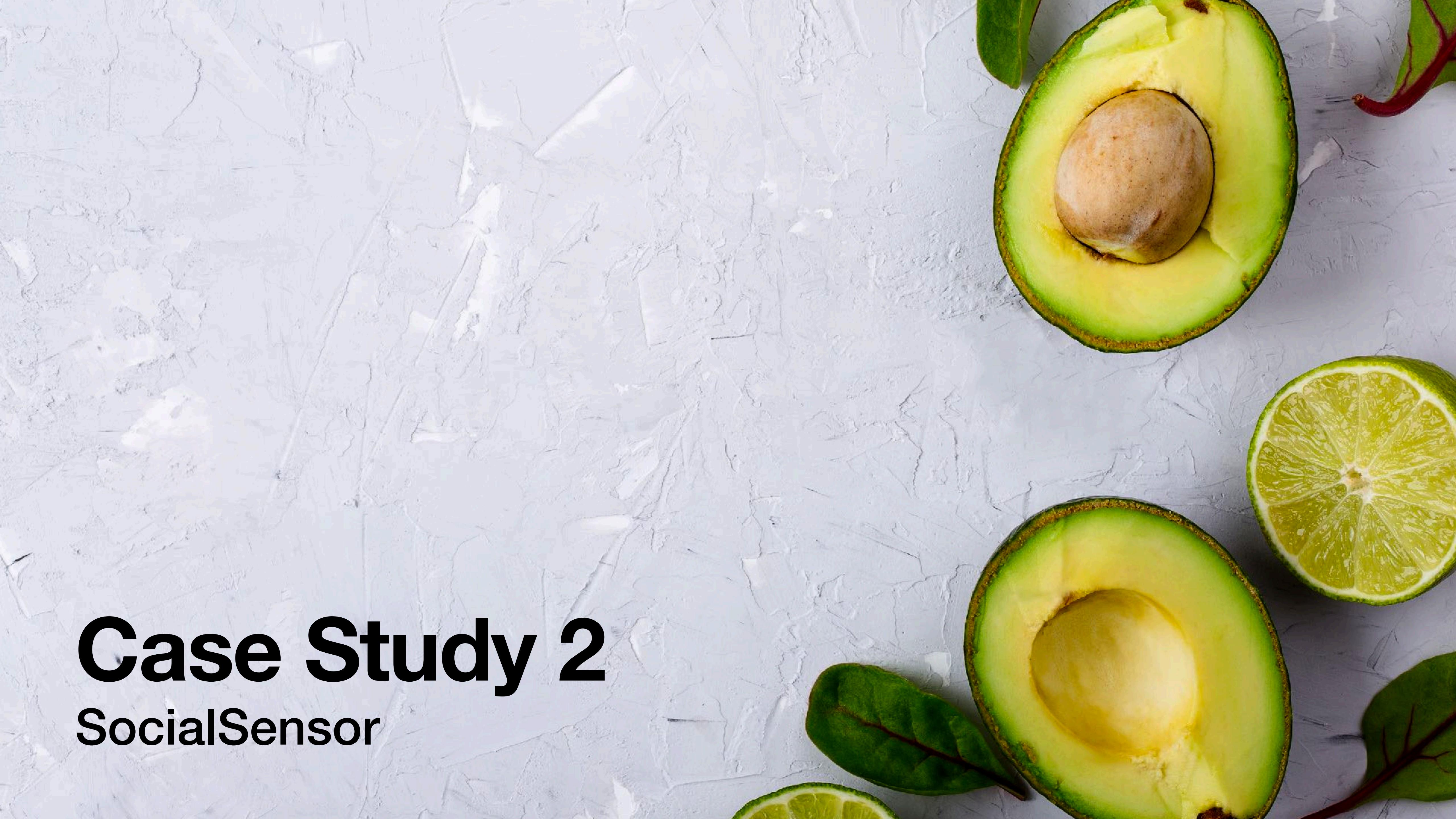
This paradigm shift to modular-composed ML systems opens up new opportunities for computer systems research

- This shift to modular-composed systems opens many **interesting systems questions**.
- It is also exciting because it means leading AI results can be achieved through clever **systems ideas**, not just scaling up training.

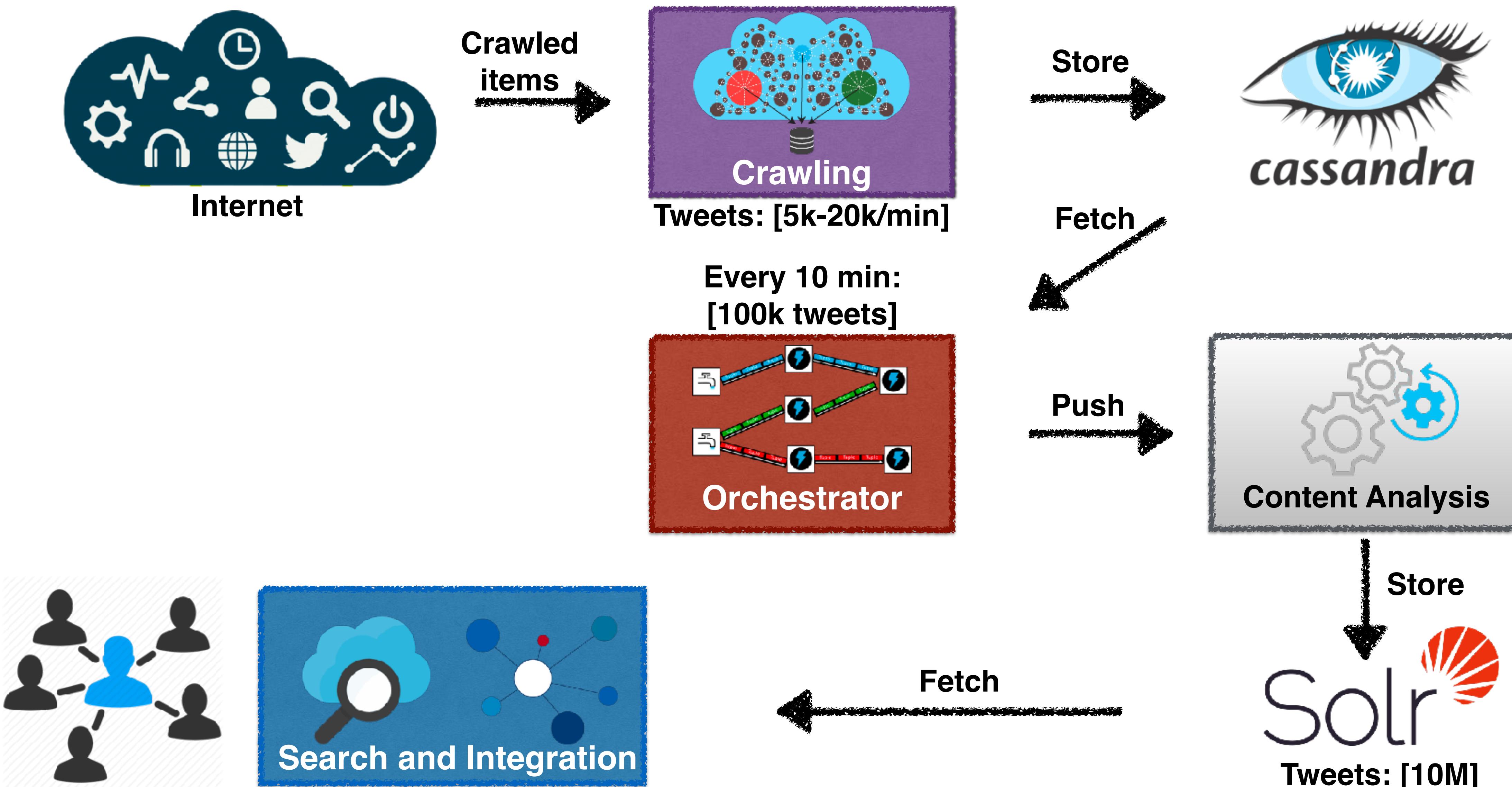


Case Study 2

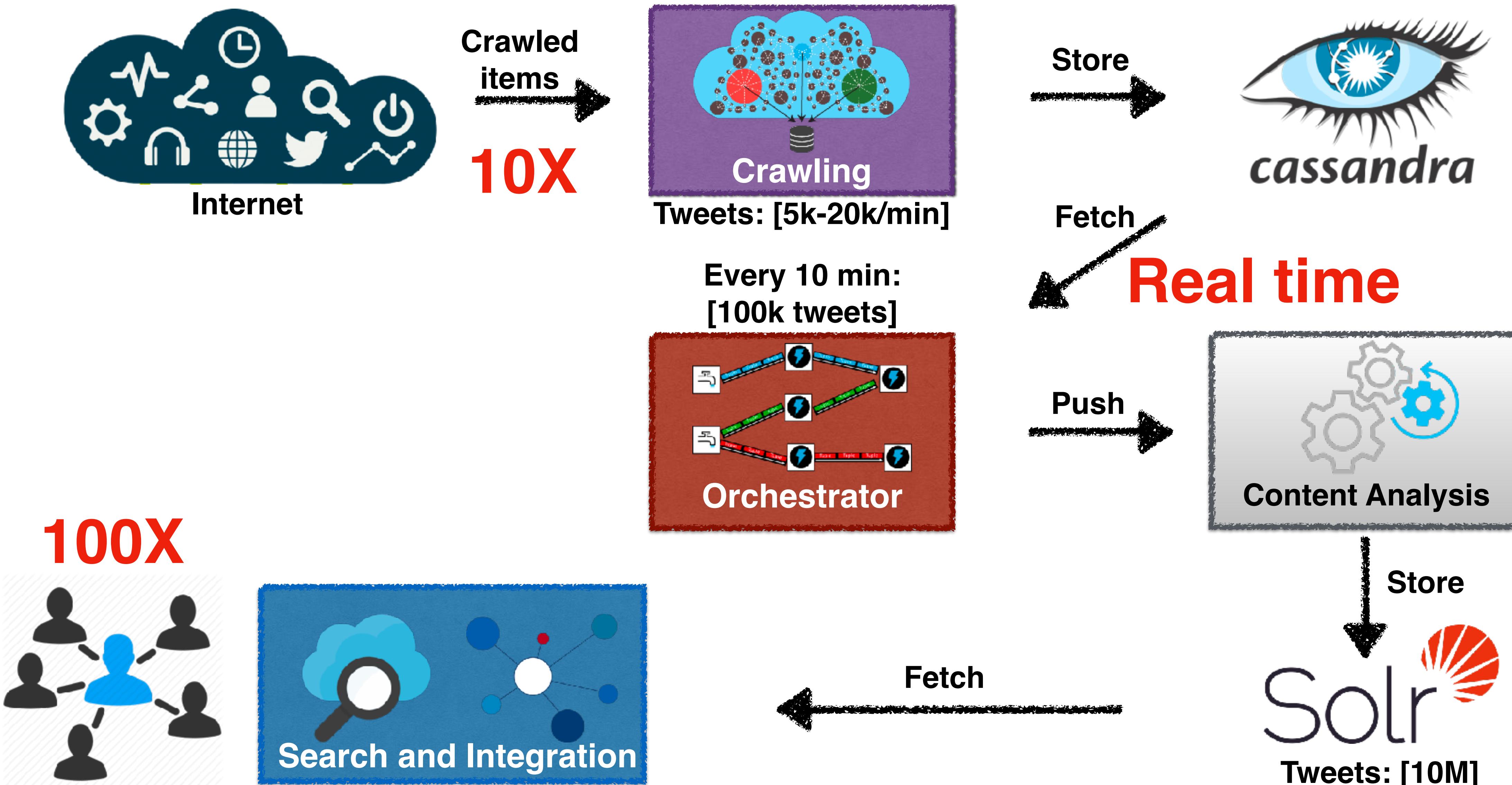
SocialSensor



SocialSensor



Challenges



How can we gain a better performance without using more resources?

Let's try out different system configurations!

Opportunity: Data processing engines in the pipeline were all configurable



> 100



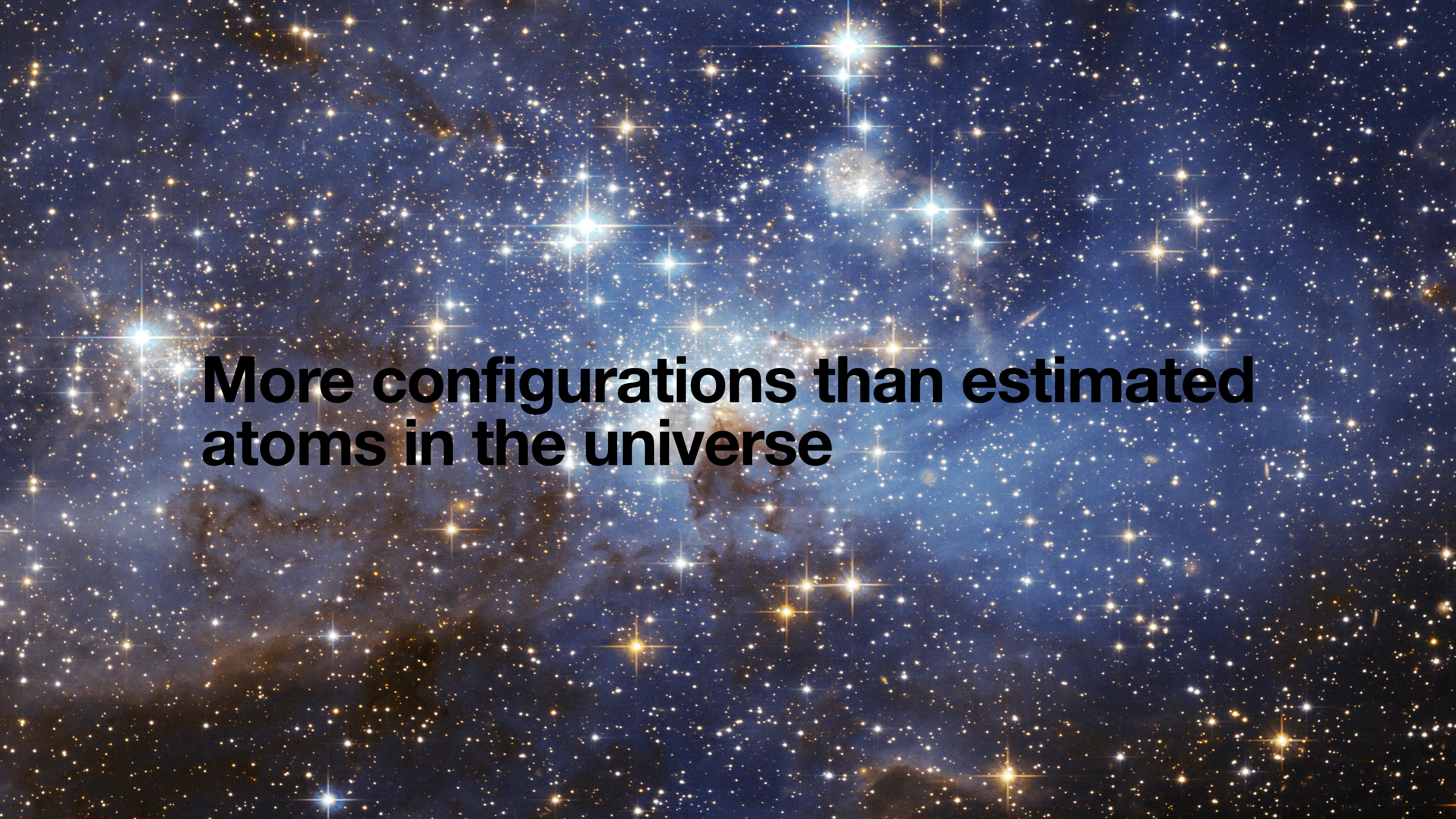
STORM

> 100



> 100

2^{300}



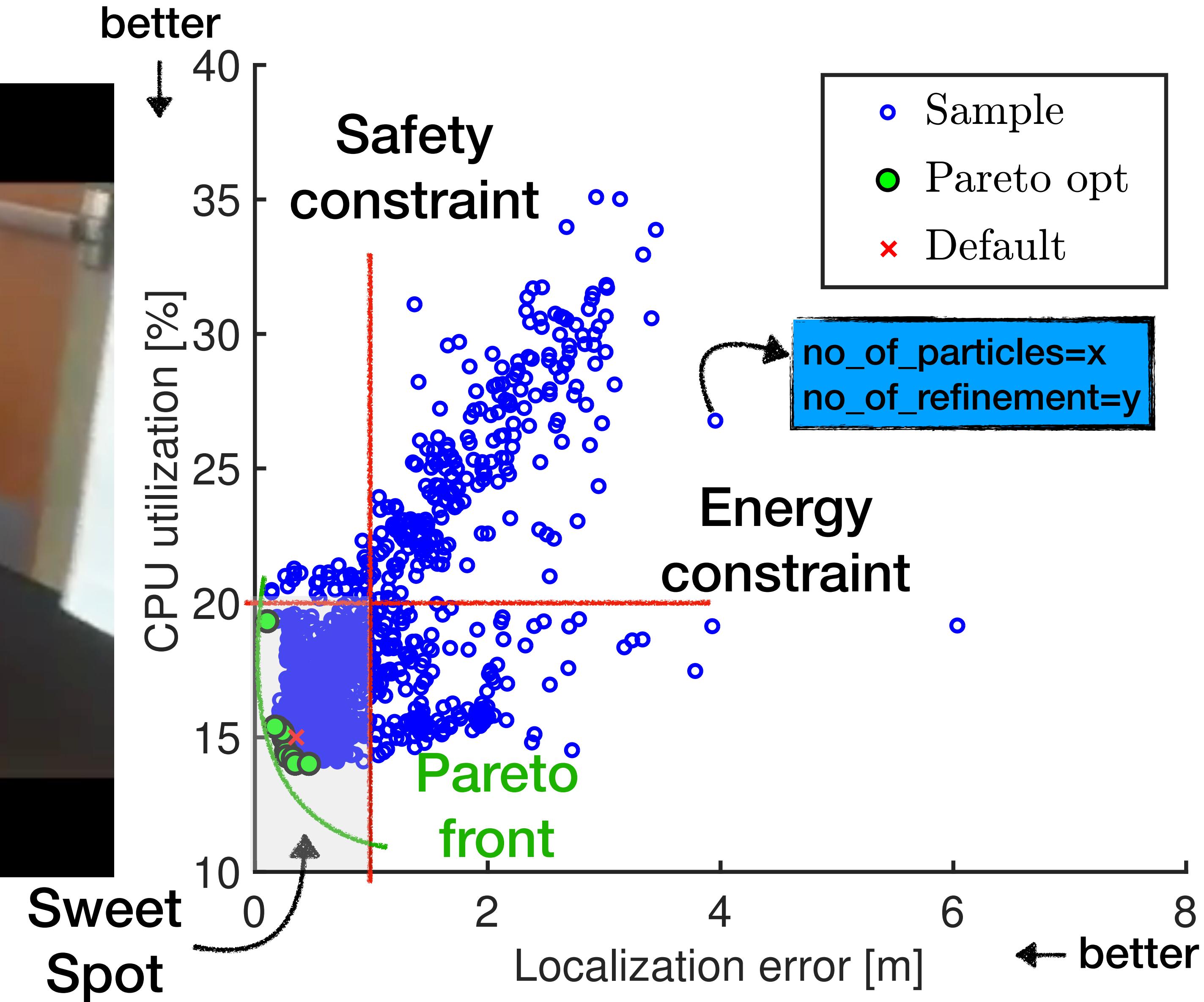
**More configurations than estimated
atoms in the universe**

Case Study 3

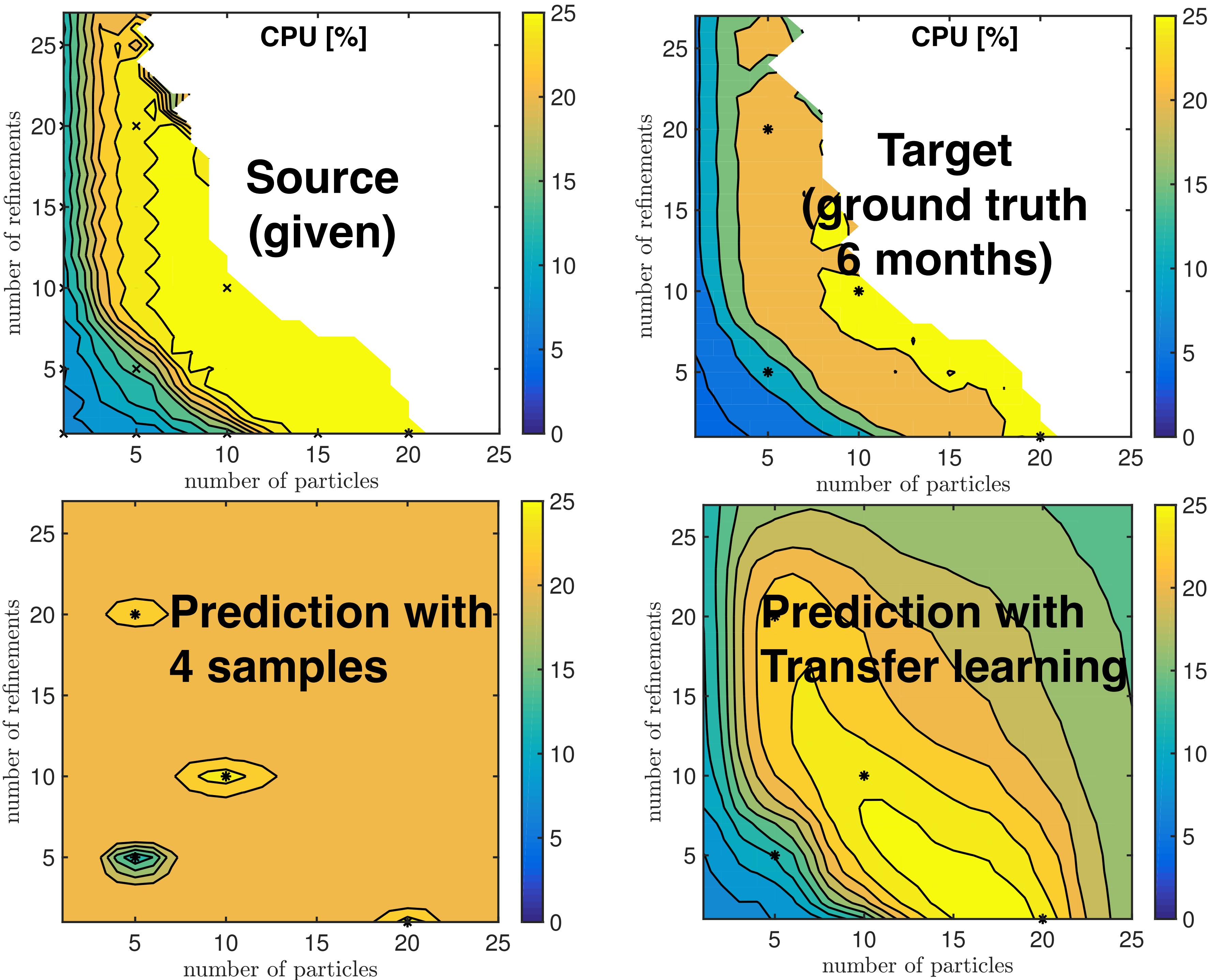
Robotics



CoBot experiment: DARPA BRASS



CoBot experiment



Details: [SEAMS '17]

Transfer Learning for Improving Model Predictions in Highly Configurable Software

Pooyan Jamshidi, Miguel Velez, Christian Kästner
Carnegie Mellon University, USA
{pjamshid,mvelezce,kaestner}@cs.cmu.edu

Norbert Siegmund
Bauhaus-University Weimar, Germany
norbert.siegmund@uni-weimar.de

Prasad Kawthekar
Stanford University, USA
pkawthek@stanford.edu

Abstract—Modern software systems are built to be used in dynamic environments using configuration capabilities to adapt to changes and external uncertainties. In a self-adaptation context, we are often interested in reasoning about the performance of the systems under different configurations. Usually, we learn a black-box model based on real measurements to predict the performance of the system given a specific configuration. However, as modern systems become more complex, there are many configuration parameters that may interact and we end up learning an exponentially large configuration space. Naturally, this does not scale when relying on real measurements in the actual changing environment. We propose a different solution:

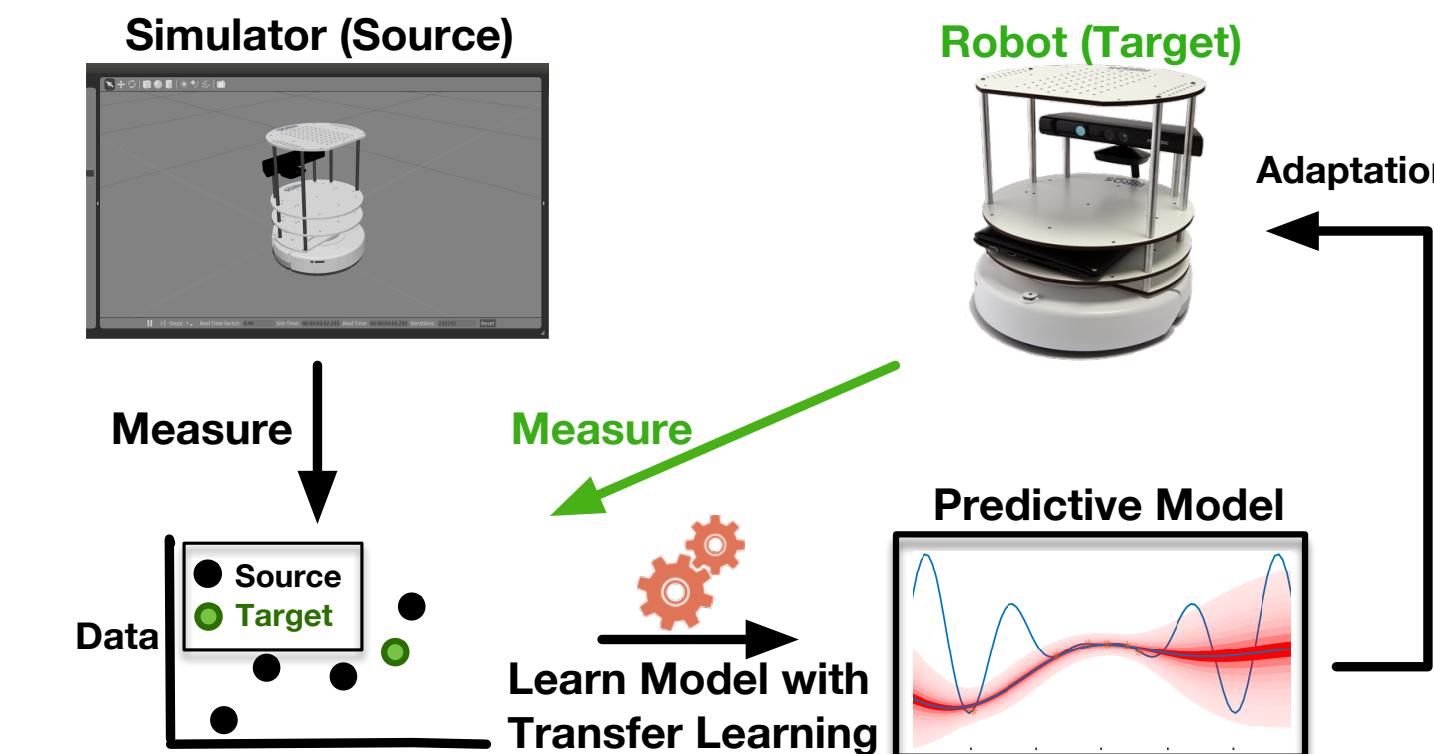


Fig. 1: Transfer learning for performance model learning.

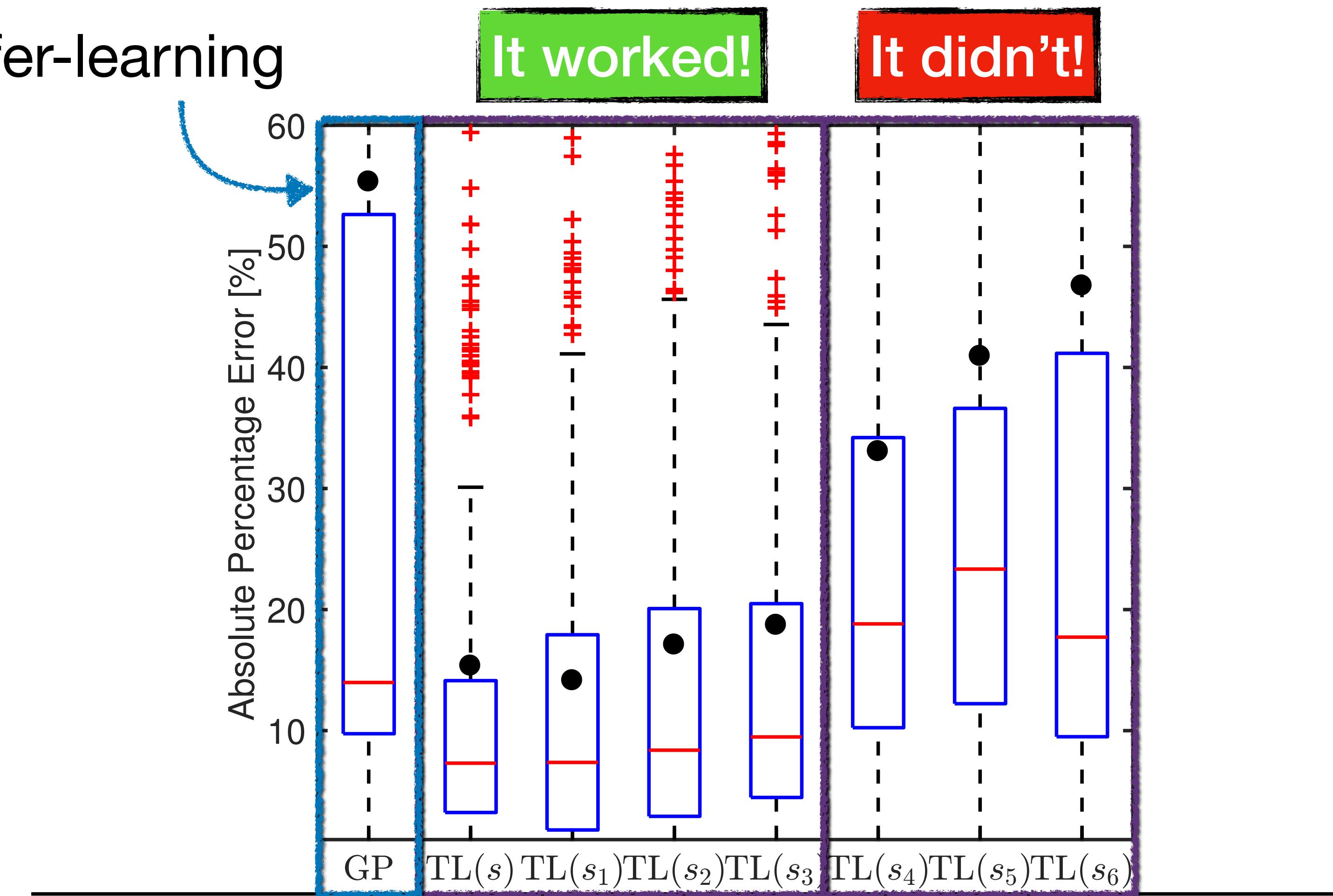
Looking further: When transfer learning goes wrong

Non-transfer-learning

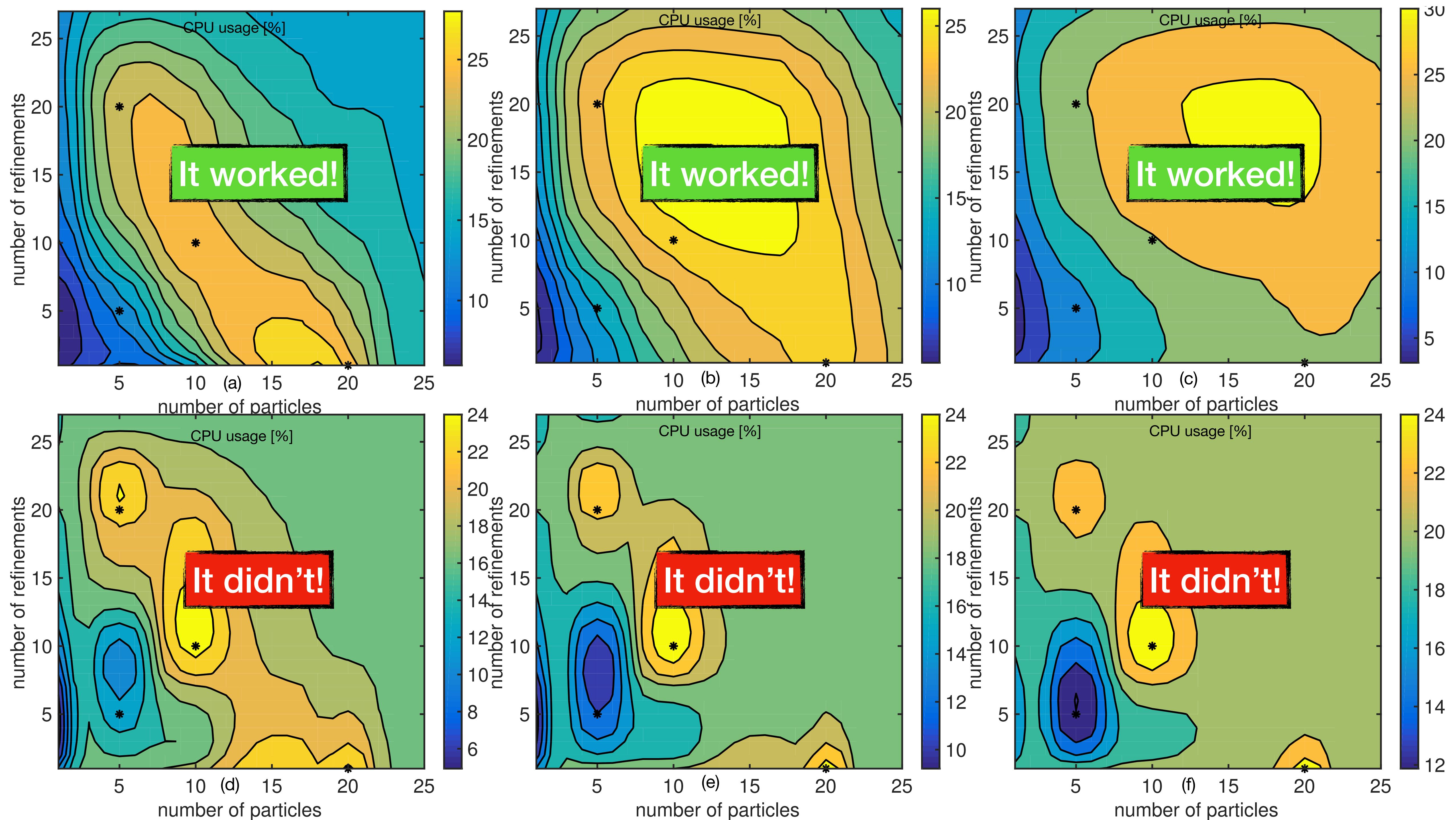
It worked!

It didn't!

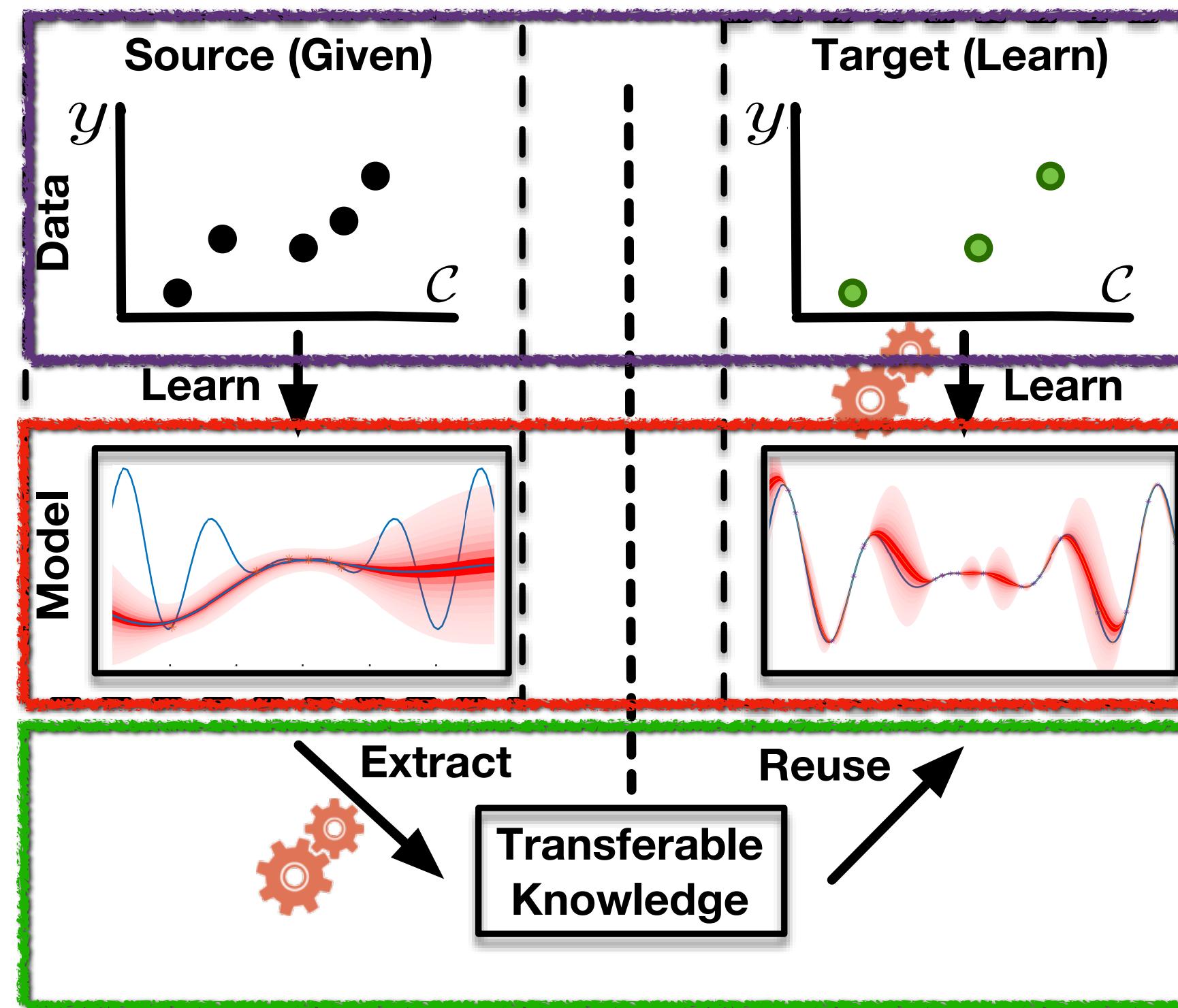
Insight: Predictions become more accurate when the source is **more related** to the target.



Sources	s	s_1	s_2	s_3	s_4	s_5	s_6
noise-level	0	5	10	15	20	25	30
corr. coeff.	0.98	0.95	0.89	0.75	0.54	0.34	0.19
$\mu(pe)$	15.34	14.14	17.09	18.71	33.06	40.93	46.75



Key question: Can we develop a theory to explain when transfer learning works?



Q1: How source and target are “related”?

Q2: What characteristics are preserved?

Q3: What are the actionable insights?

Details: [ASE '17]

Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis

Pooyan Jamshidi
Carnegie Mellon University, USA

Norbert Siegmund
Bauhaus-University Weimar, Germany

Miguel Velez, Christian Kästner
Akshay Patel, Yuvraj Agarwal
Carnegie Mellon University, USA

Abstract—Modern software systems provide many configuration options which significantly influence their non-functional properties. To understand and predict the effect of configuration options, several sampling and learning strategies have been proposed, albeit often with significant cost to cover the highly dimensional configuration space. Recently, transfer learning has been applied to reduce the effort of constructing performance models by transferring knowledge about performance behavior across environments. While this line of research is promising to learn more accurate models at a lower cost, it is unclear why and when transfer learning works for performance modeling. To shed light on when it is beneficial to apply transfer learning, we conducted an empirical study on four popular software systems, varying software configurations and environmental conditions, such as hardware, workload, and software versions, to identify the key knowledge pieces that can be exploited for transfer learning. Our results show that in small environmental changes (e.g., homogeneous workload change), by applying a linear transformation to the performance model, we can understand the performance behavior of the target environment, while for severe environmental changes (e.g., drastic workload change) we

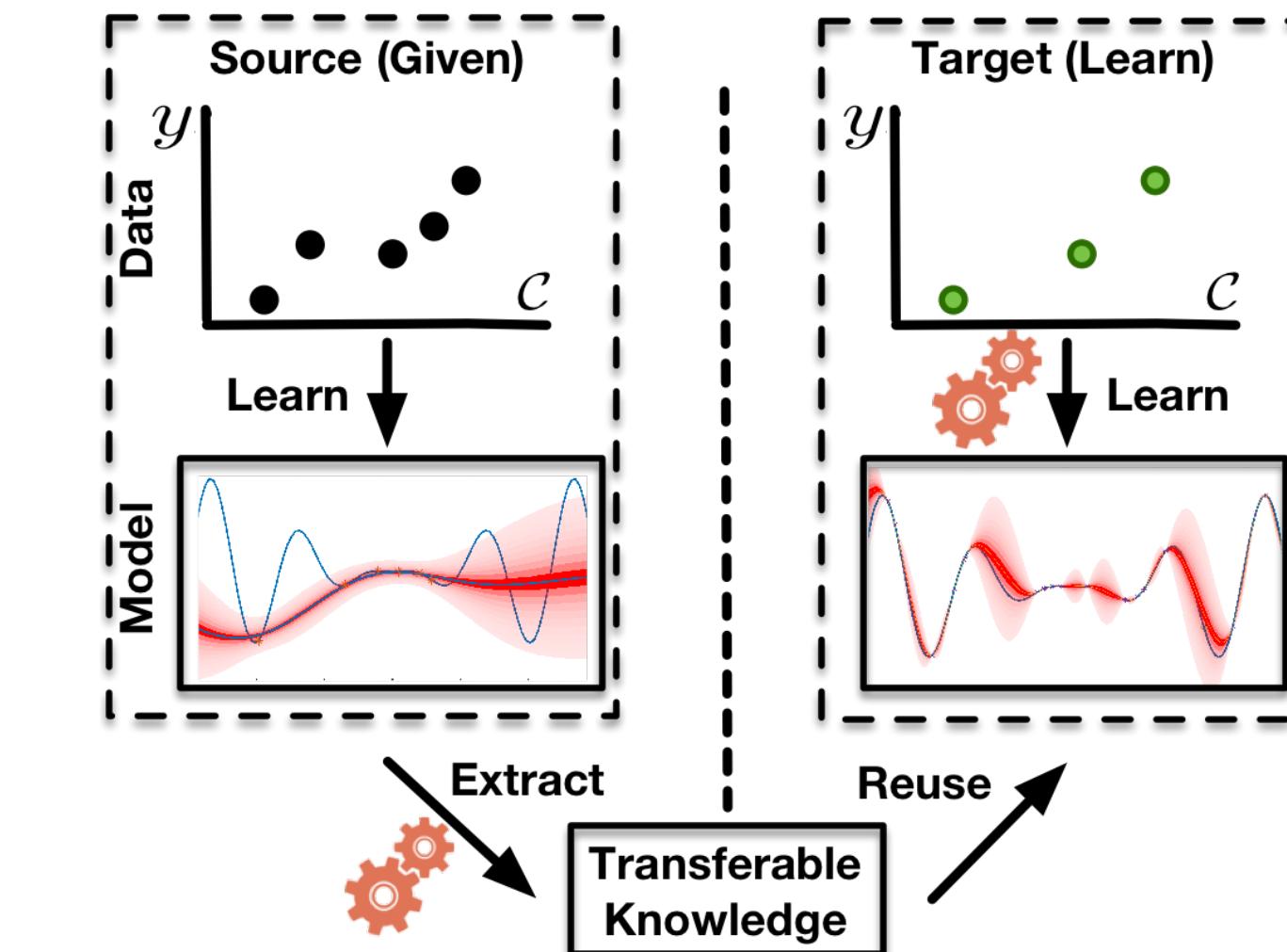


Fig. 1: Transfer learning is a form of machine learning that takes advantage of transferable knowledge from source to learn an accurate, reliable, and less costly model for the target environment.

Details: [AAAI Spring Symposium '19]

Transfer Learning for Performance Modeling of Configurable Systems: A Causal Analysis

Mohammad Ali Javidian, Pooyan Jamshidi, Marco Valtorta

Department of Computer Science and Engineering
University of South Carolina, Columbia, SC, USA

Abstract

Modern systems (e.g., deep neural networks, big data analytics, and compilers) are highly configurable, which means they expose different performance behavior under different configurations. The fundamental challenge is that one cannot simply measure all configurations due to the sheer size of the configuration space. Transfer learning has been used to reduce the measurement efforts by transferring knowledge about performance behavior of systems across environments. Previously, research has shown that statistical models are indeed transferable across environments. In this work, we investigate identifiability and transportability of causal effects and statistical relations in highly-configurable systems. Our causal analysis agrees with previous exploratory analysis (Jamshidi et al. 2017) and confirms that the causal effects of configuration options can be carried over across environments with high

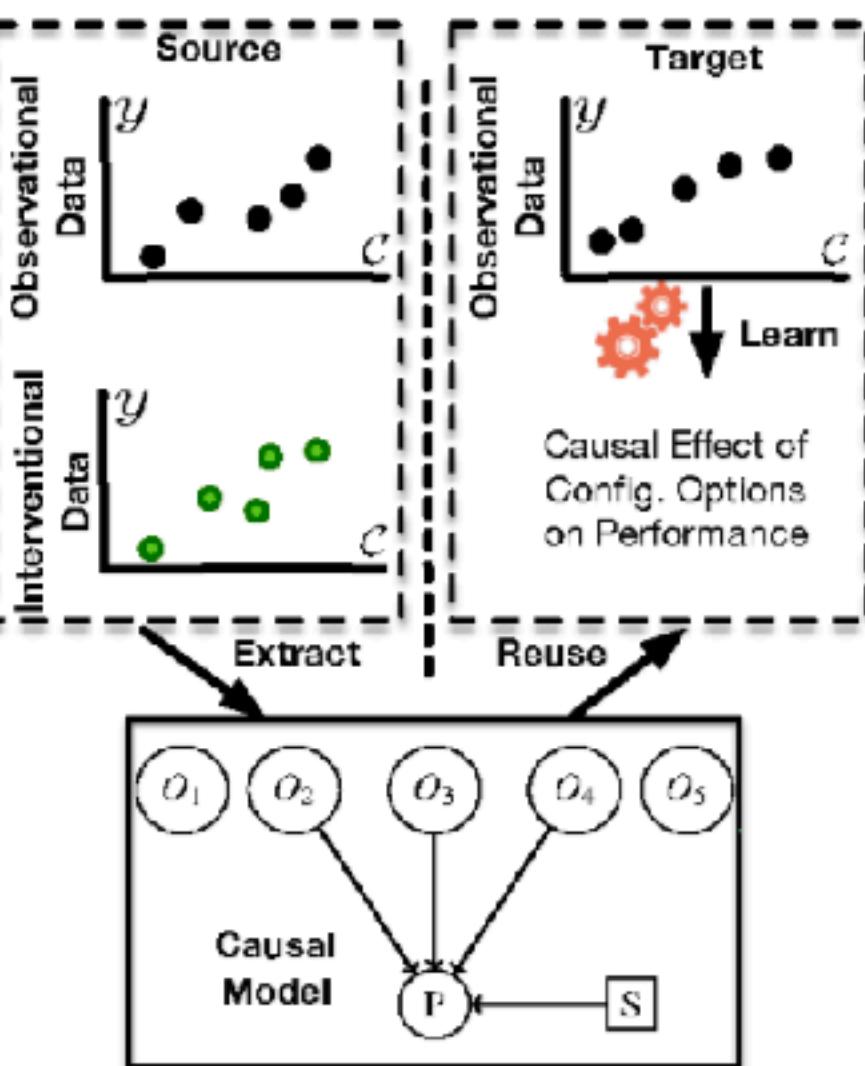
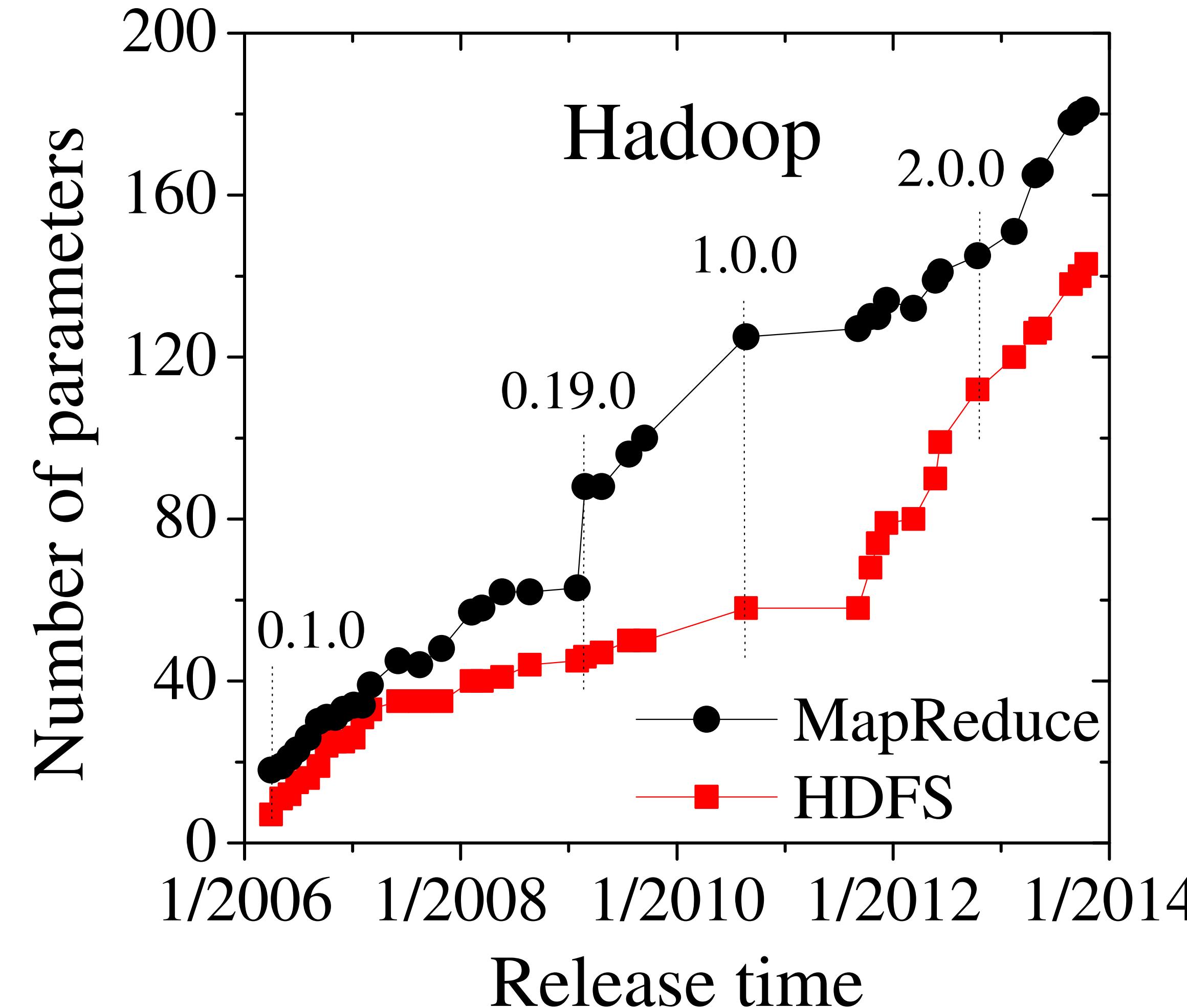


Figure 1: Exploiting causal inference for performance analysis.

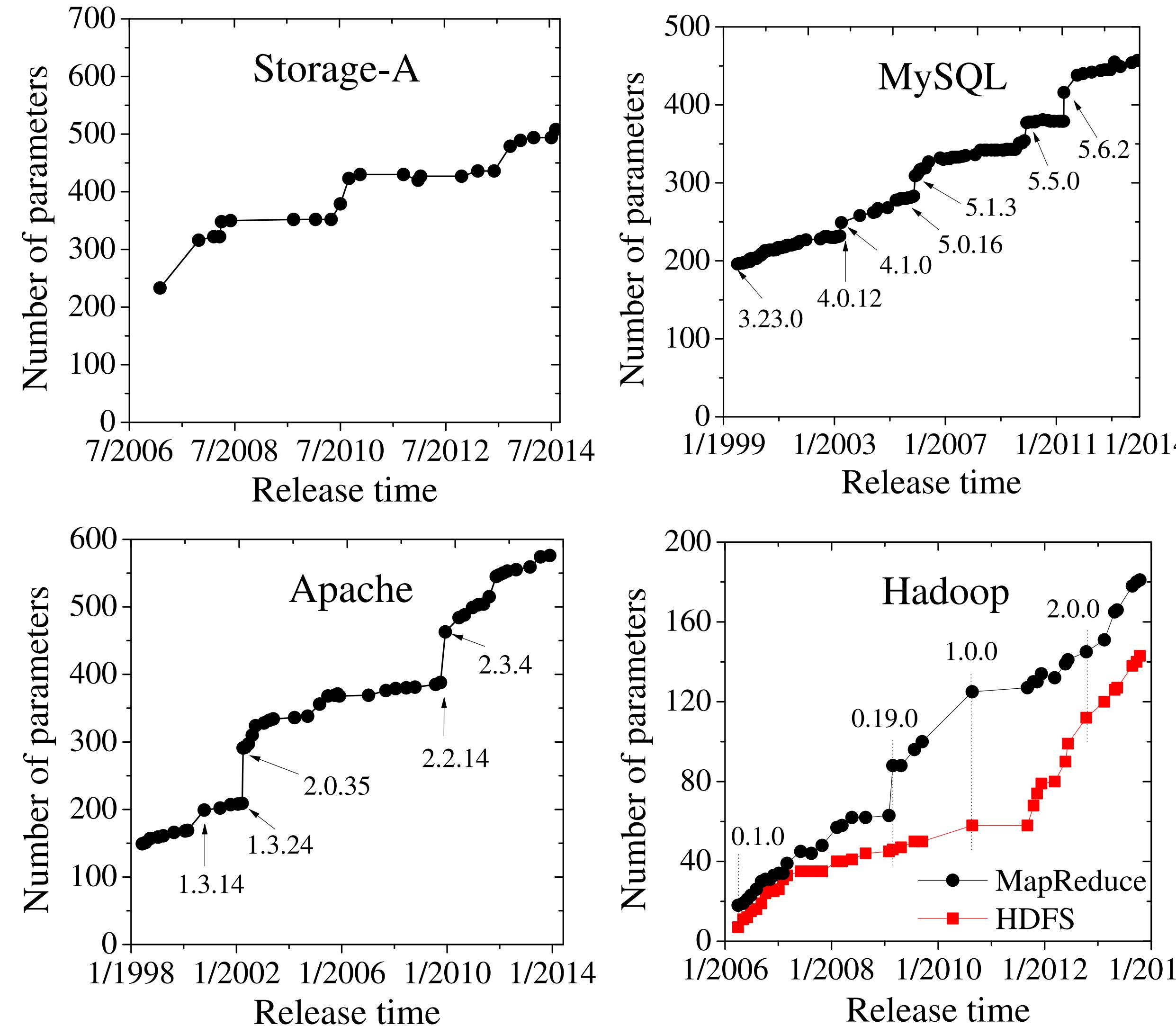


```
102  
103 drpc.port: 3772  
104 drpc.worker.threads: 64  
105 drpc.max_buffer_size: 1048576  
106 drpc.queue.size: 128  
107 drpc.invocations.port: 3773  
108 drpc.invocations.threads: 64  
109 drpc.request.timeout.secs: 600  
110 drpc.childopts: "-Xmx768m"  
111 drpc.http.port: 3774  
112 drpc.https.port: -1  
113 drpc.https.keystore.password: ""  
114 drpc.https.keystore.type: "JKS"  
115 drpc.http.creds.plugin: org.apache.storm.security.auth.DefaultHttpCredentialsPlugin  
116 drpc.authorizer.acl.filename: "drpc-auth-acl.yaml"  
117 drpc.authorizer.acl.strict: false  
118  
119 transactional.zookeeper.root: "/transactional"  
120 transactional.zookeeper.servers: null  
121 transactional.zookeeper.port: null  
122  
123 ## blobstore configs  
124 supervisor.blobstore.class: "org.apache.storm.blobstore.NimbusBlobStore"  
125 supervisor.blobstore.download.thread.count: 5  
126 supervisor.blobstore.download.max_retries: 3  
127 supervisor.localizer.cache.target.size.mb: 10240  
128 supervisor.localizer.cleanup.interval.ms: 600000  
129
```

Empirical observations confirm that systems are becoming increasingly configurable



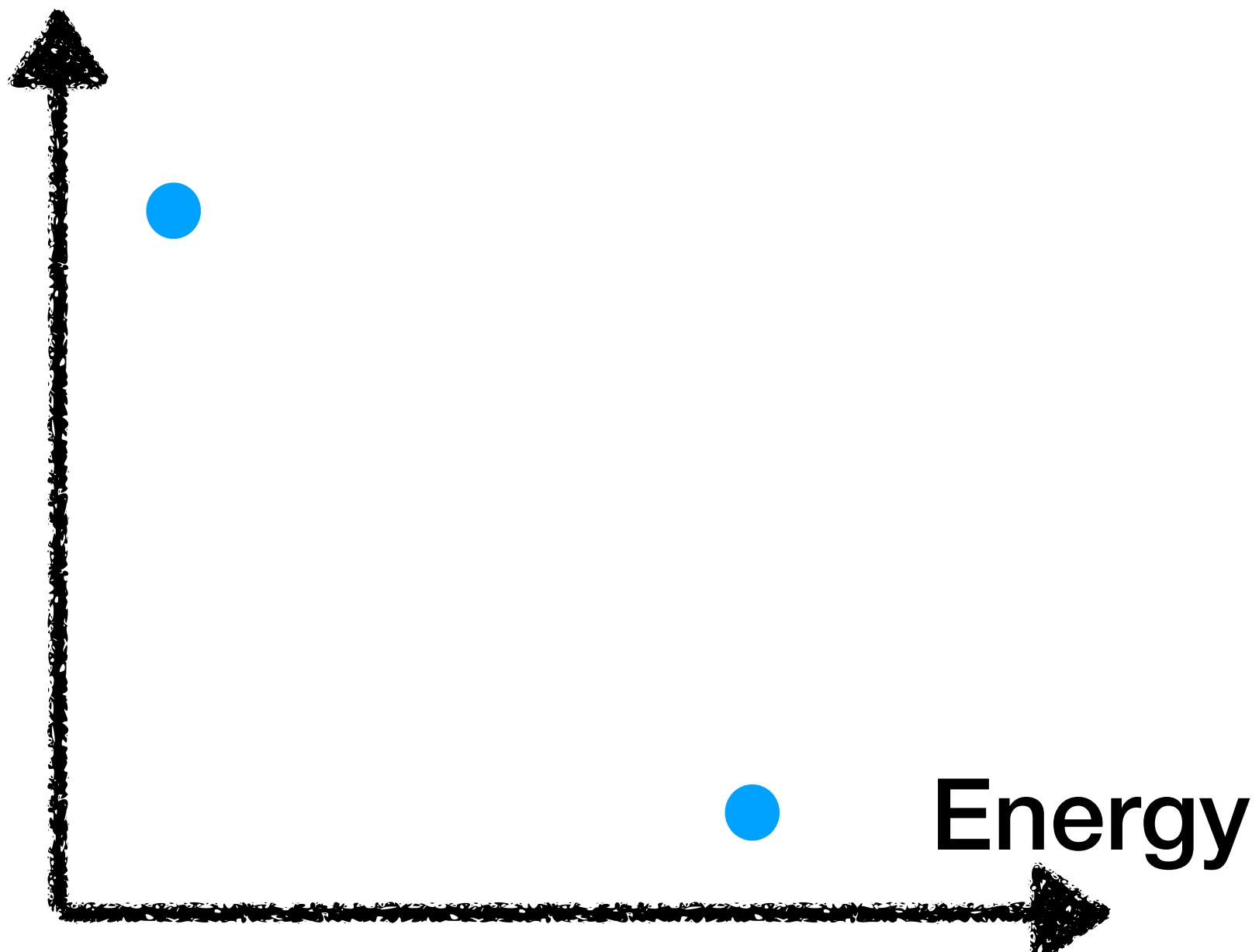
Empirical observations confirm that systems are becoming increasingly configurable



Configurations determine the performance behavior

```
void Parrot_setenv( . . . name, . . . value) {
    #ifdef PARROT HAS SETENV
        my_setenv(name, value, 1);
    #else
        int name_len=strlen(name);
        int val_len=strlen(value);
        char* envs=glob_env;
        if(envs==NULL){
            return;
        }
        strcpy(envs,name);
        strcpy(envs+name_len,"=");
        strcpy(envs+name_len + 1,value);
        putenv(envs);
    #endif
}
```

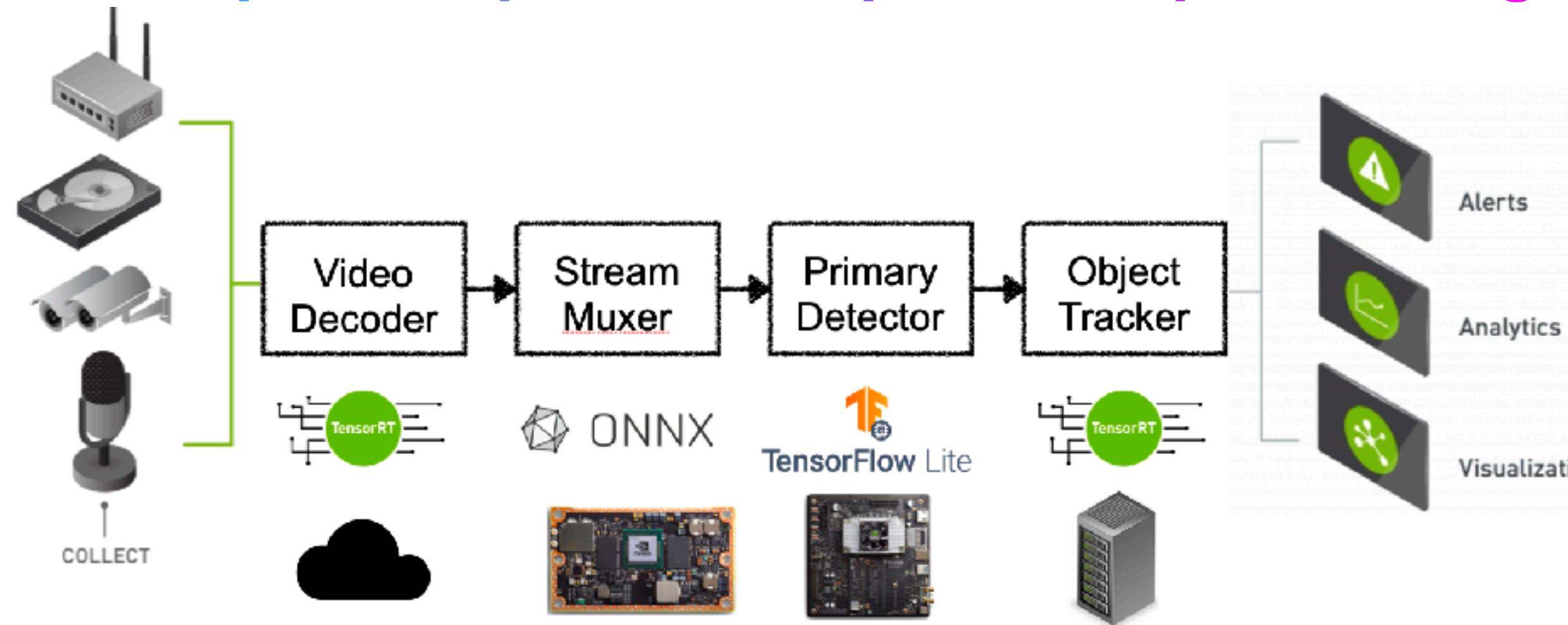
Speed



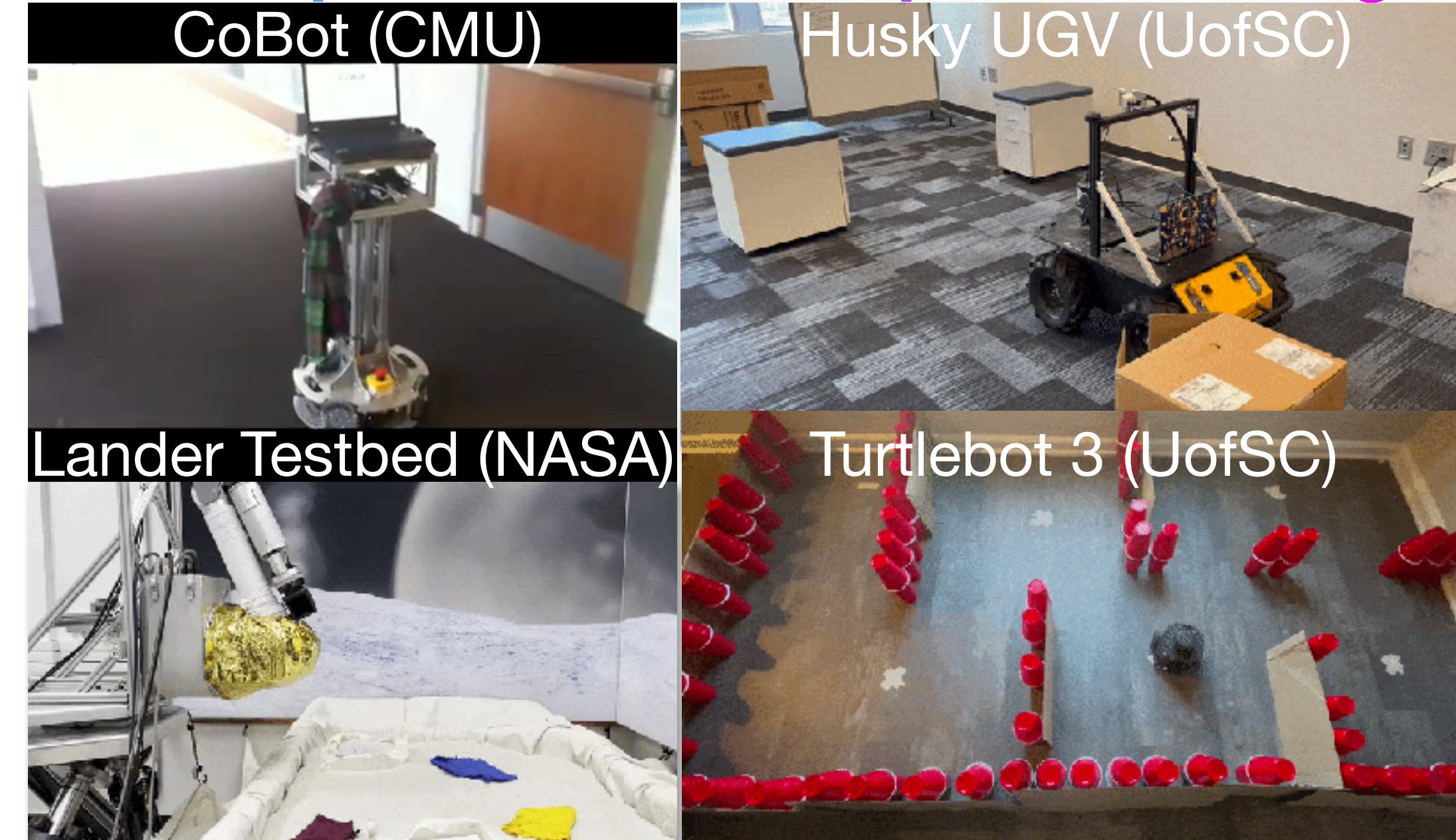
Challenges of configurations

- Difficulties in knowing **which parameters** should be set
- Setting the parameters to obtain the **intended behavior**
- Reasoning about **multiple objectives** (energy, speed)

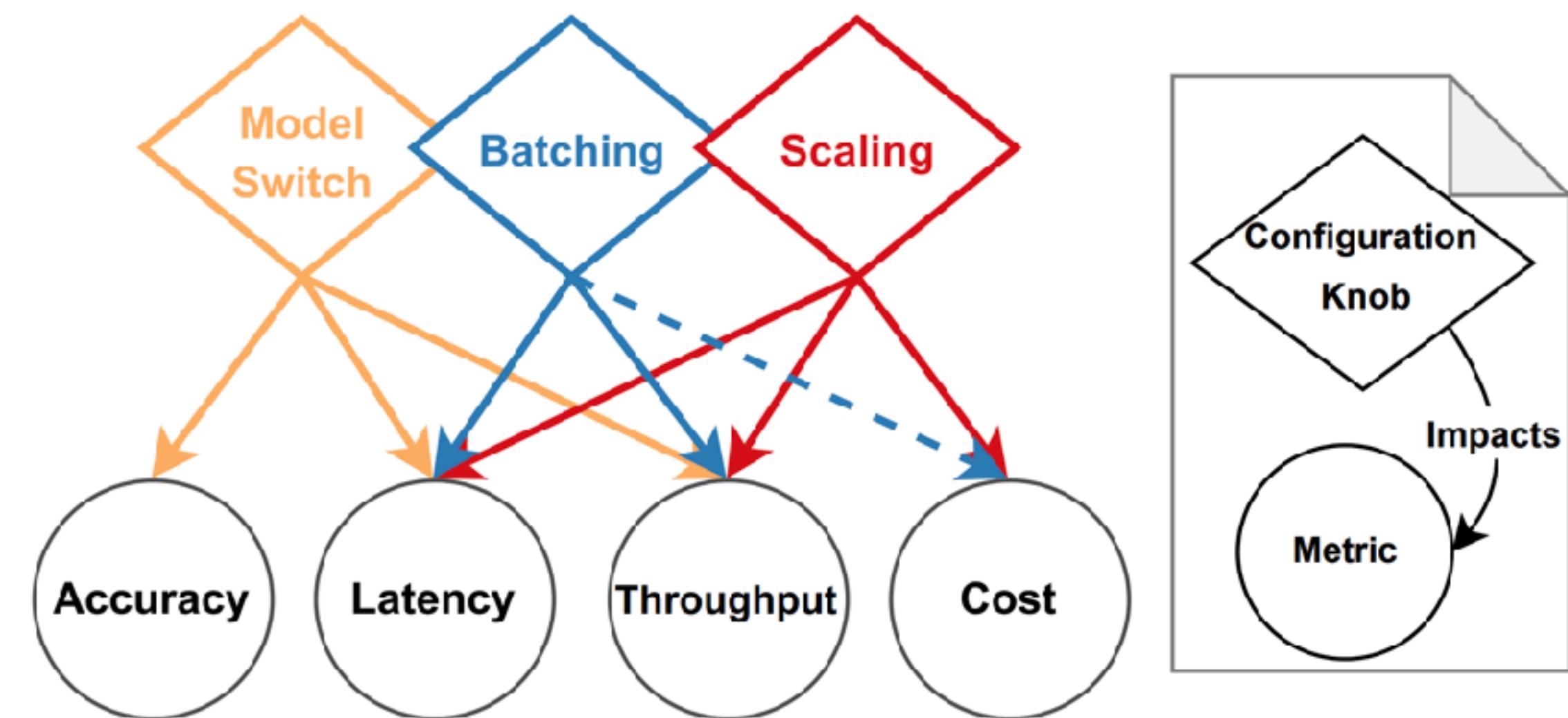
The variability space (design space) of (composed) systems is exponentially increasing



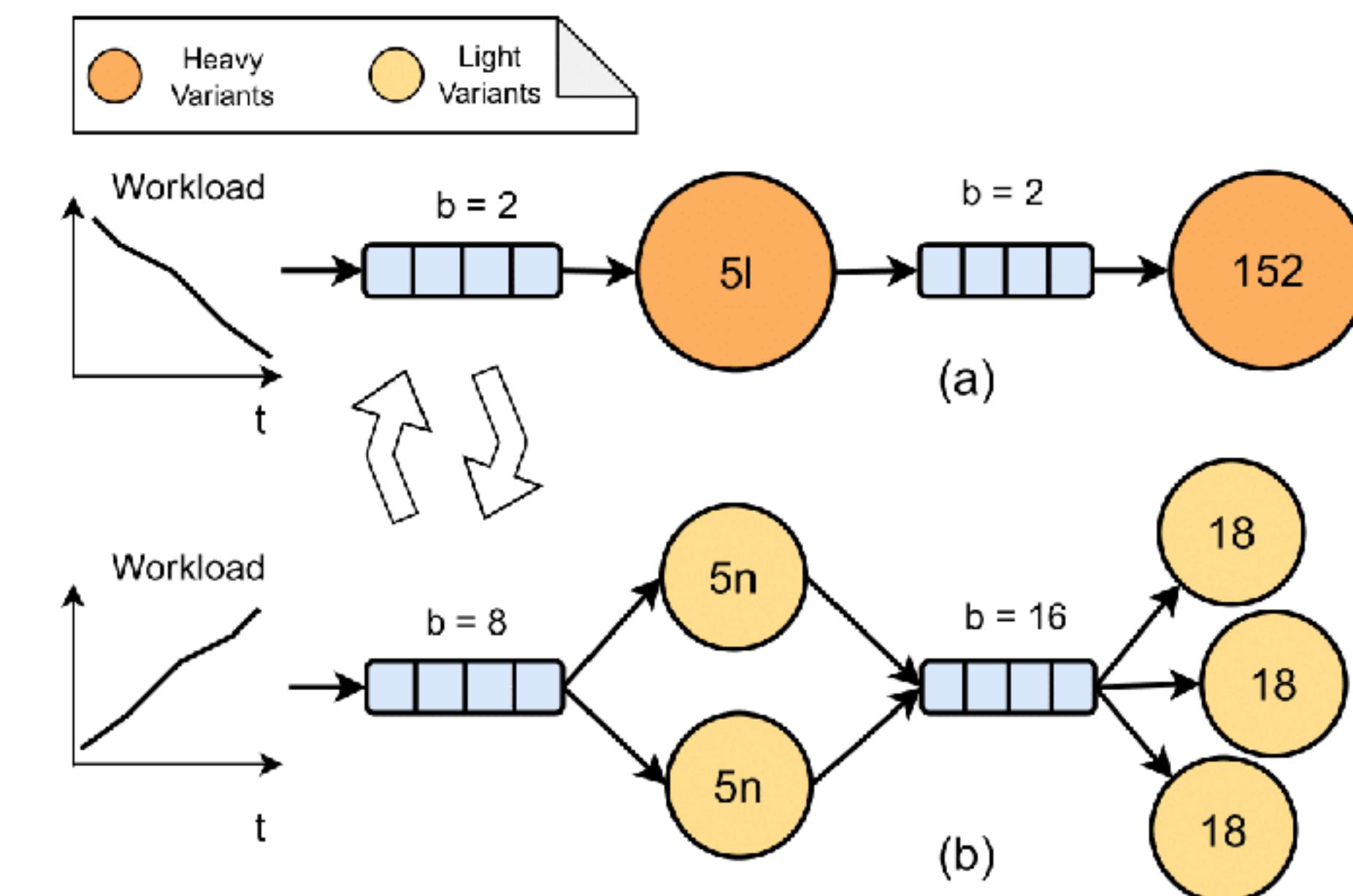
Systems operate in uncertain environments with imperfect and incomplete knowledge



Performance goals are competing and users have preferences over these goals



Goal: Enabling users to find the right quality tradeoff

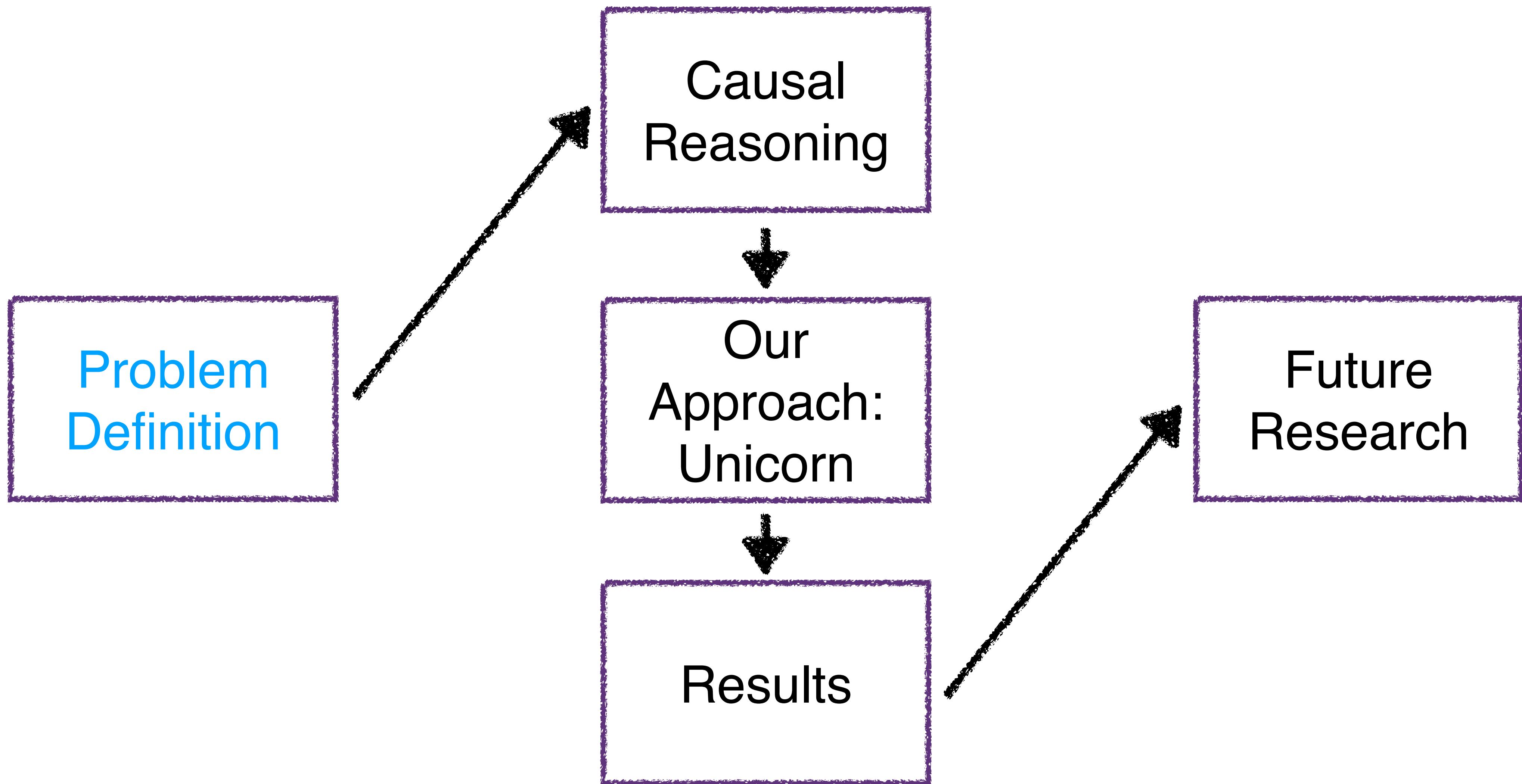


The goal of our research is...

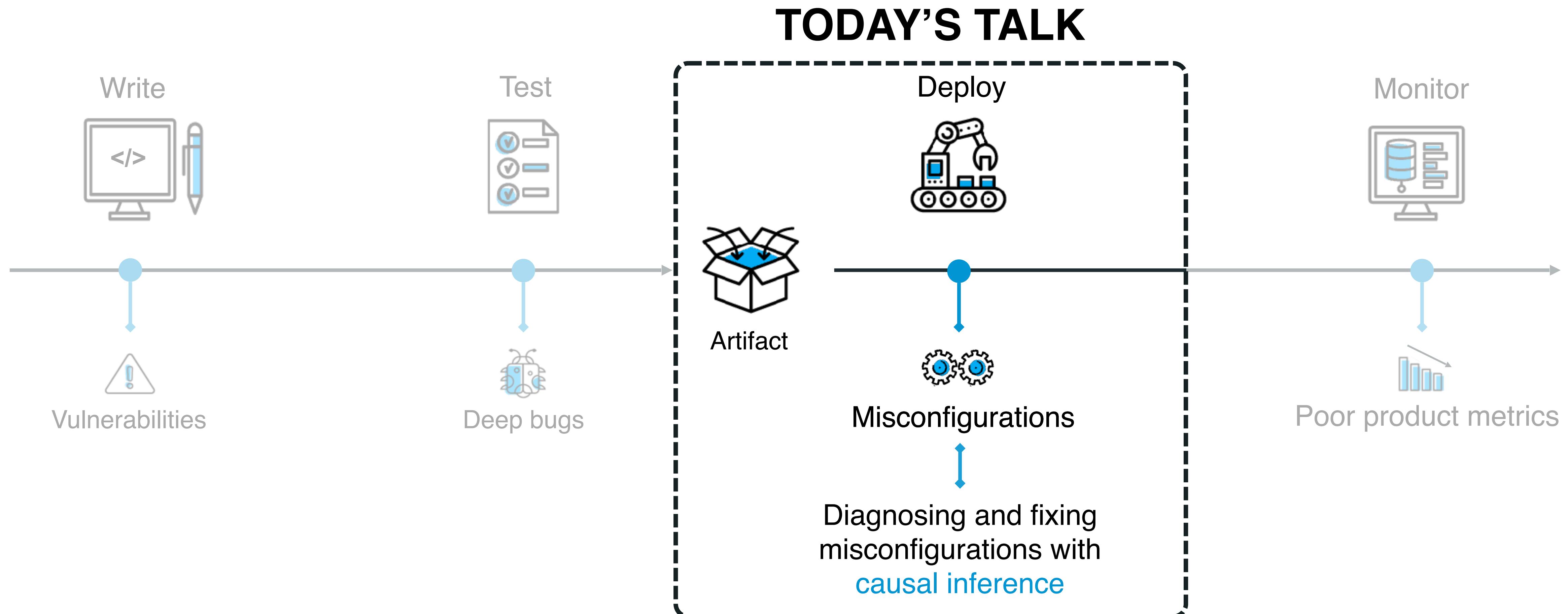
Understanding the performance behavior of
real-world highly-configurable systems that **scale** well...

... and enabling developers/users to **reason** about
qualities (performance, energy) and to make **tradeoffs**?

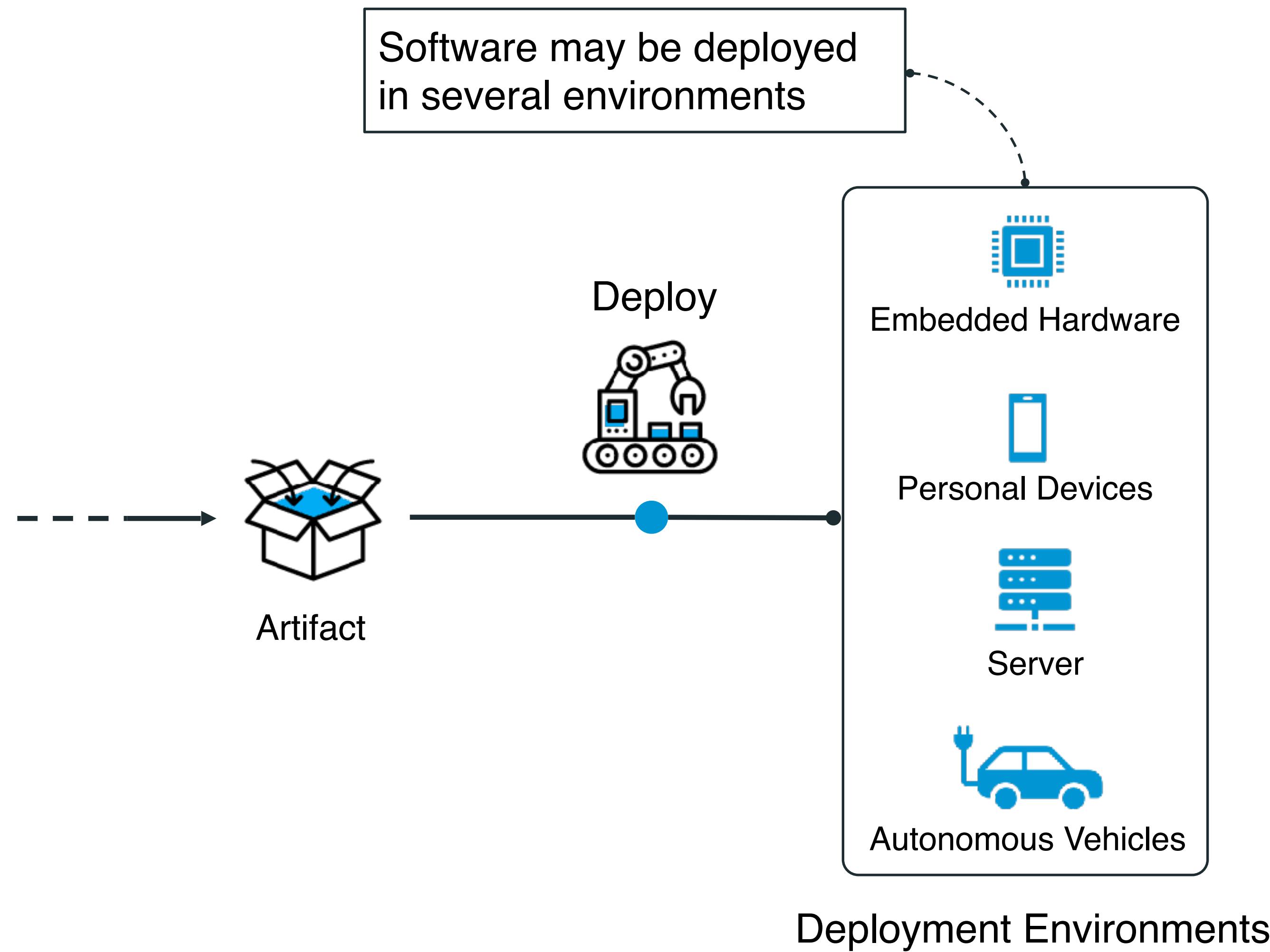
Outline



A Typical Software Lifecycle



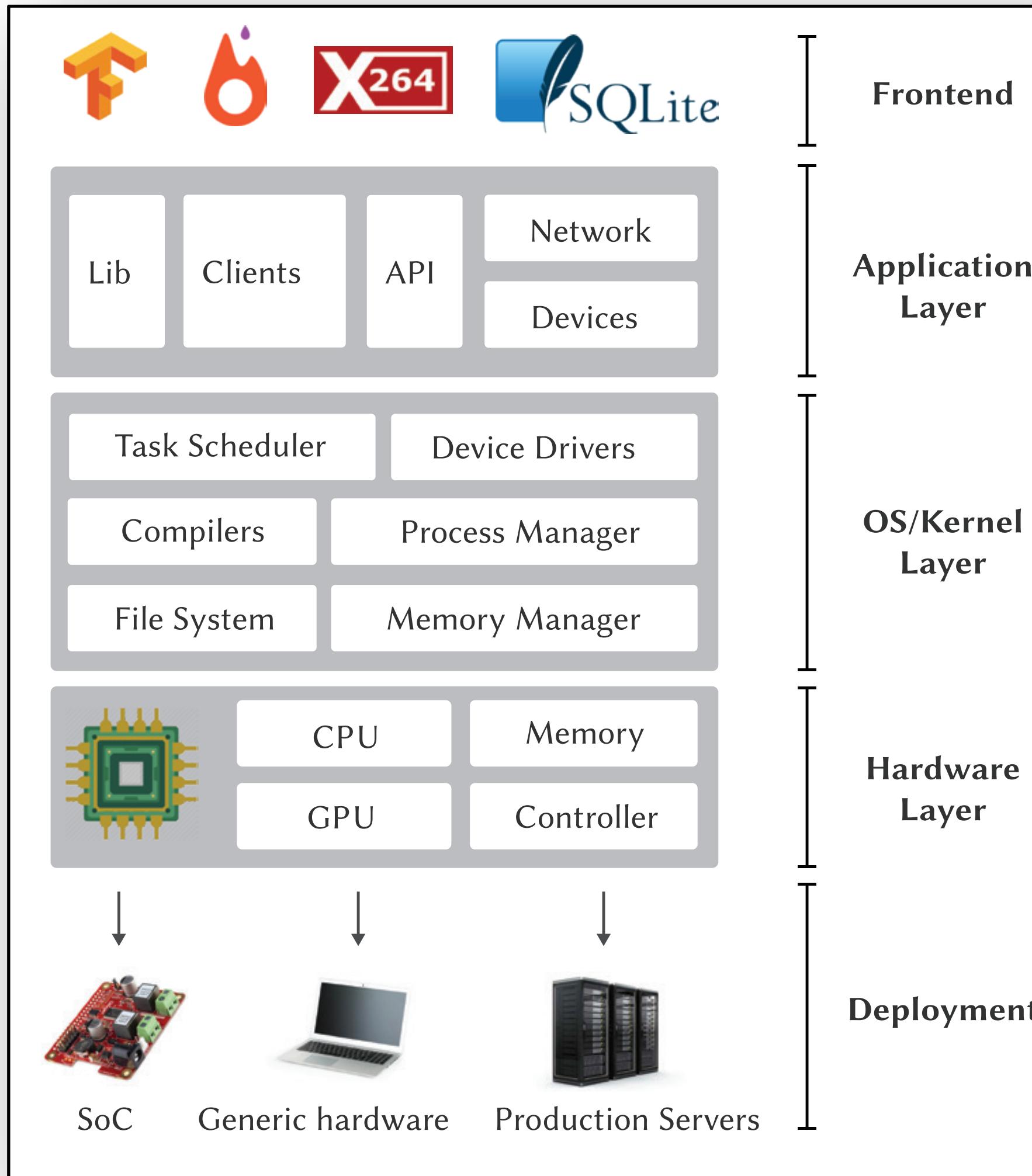
Today's Talk



Challenge

- ▷ Each deployment environment must be **configured correctly**
- ▷ This is challenging and prone to **misconfigurations**

Today's Talk



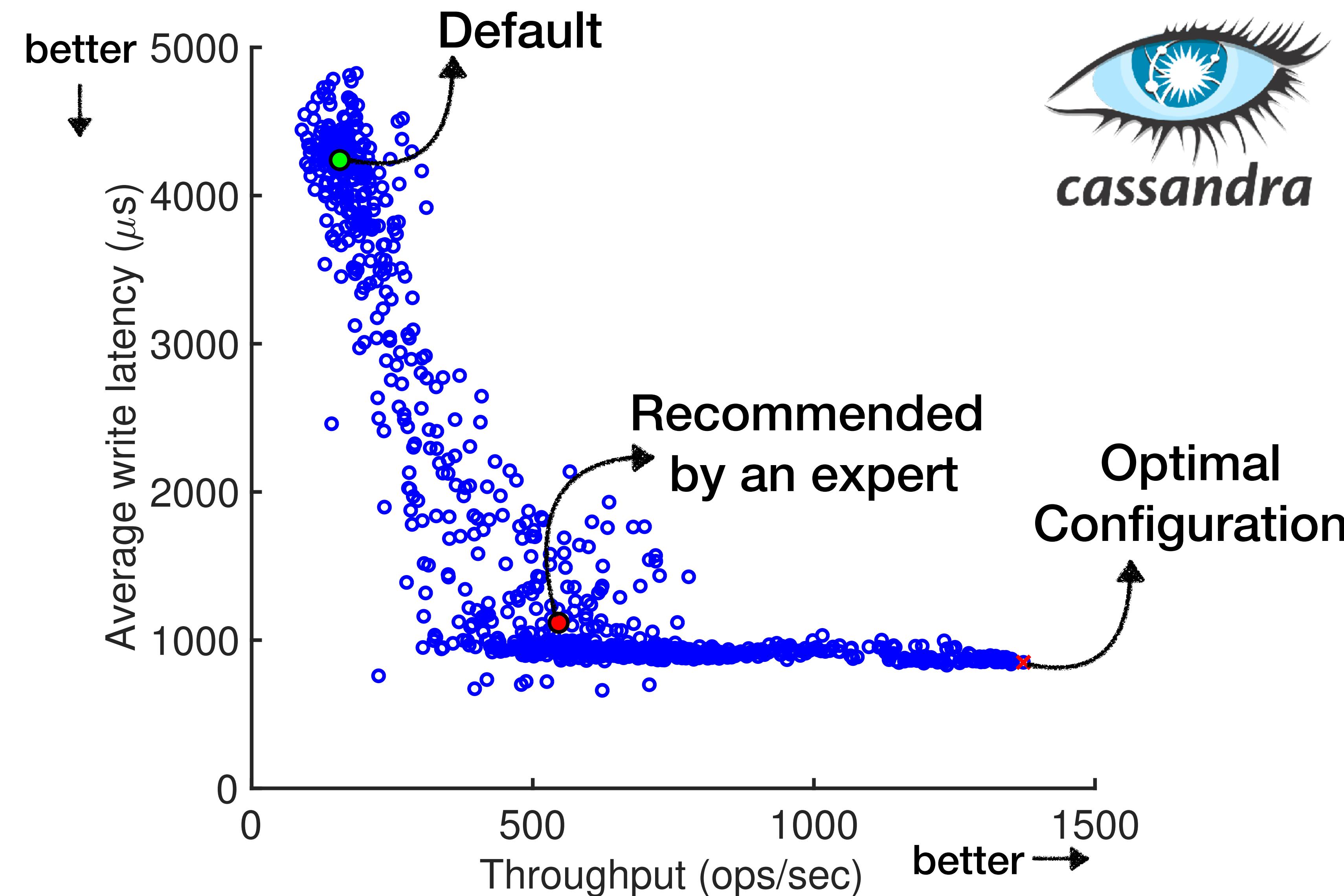
Problem

- ▷ Each deployment environment must be **configured correctly**
- ▷ This is challenging and prone to **misconfigurations**

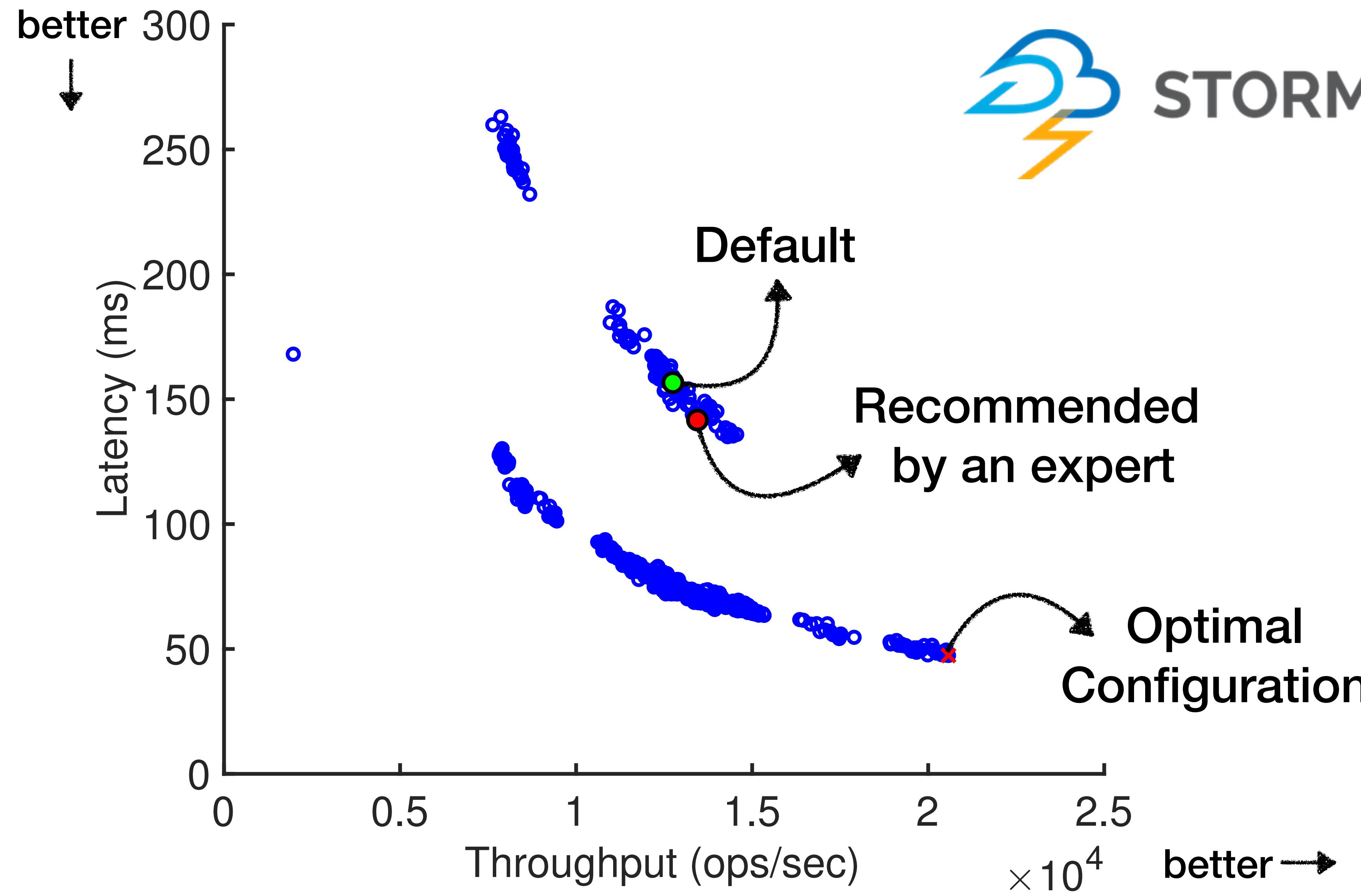
Why?

- ▷ The configuration options lie **across the software stack**
- ▷ There are **several non-trivial interactions** with one another
- ▷ The configuration space is **combinatorially large** with 100's of configuration options

Default configuration was bad, so was the expert'



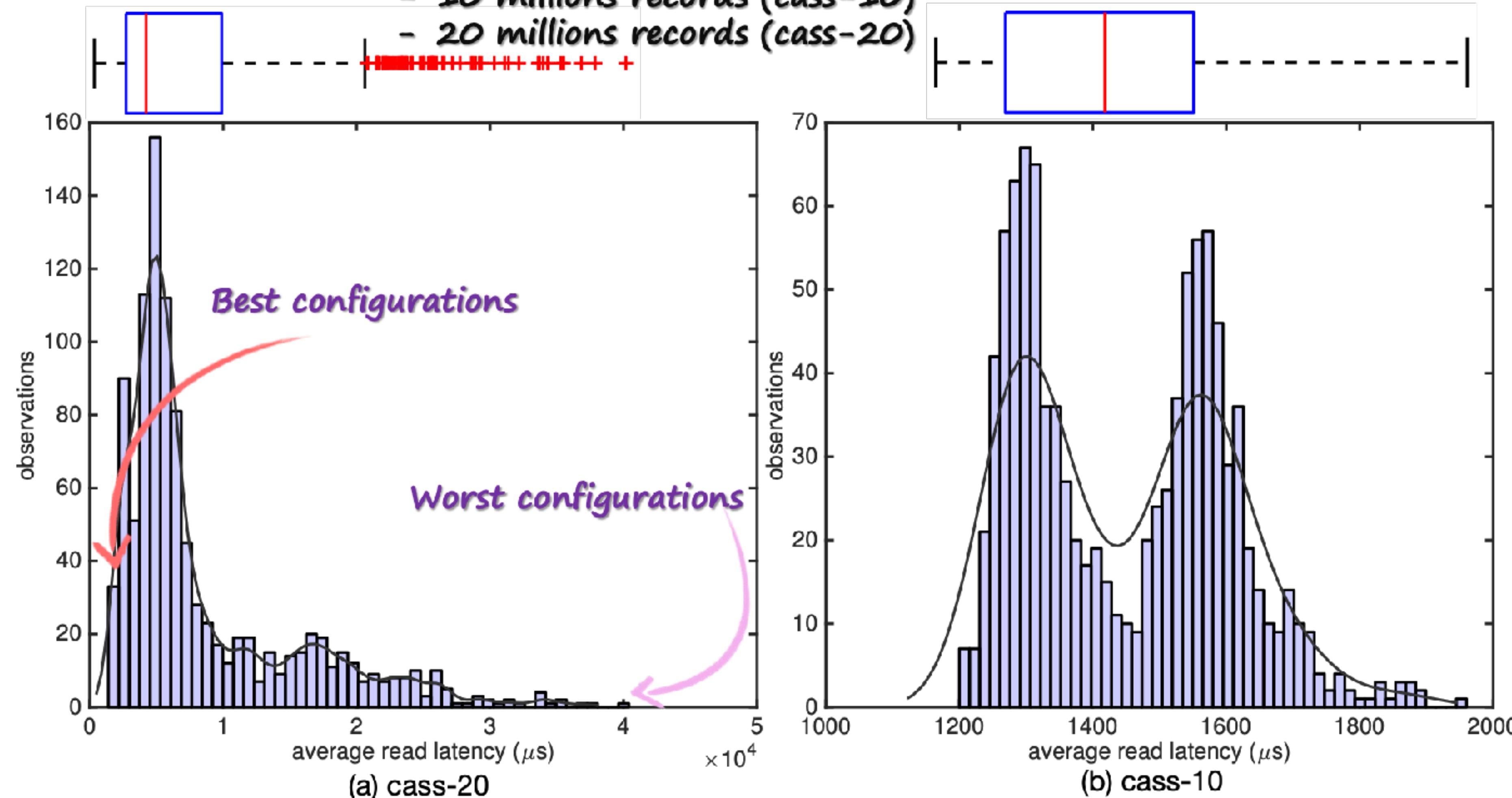
Default configuration was bad, so was the expert'



Performance behavior varies in different environments

Experiments on
Apache Cassandra:

- 6 parameters, 1024 configurations
- Average read latency
- 10 millions records (cass-10)
- 20 millions records (cass-20)



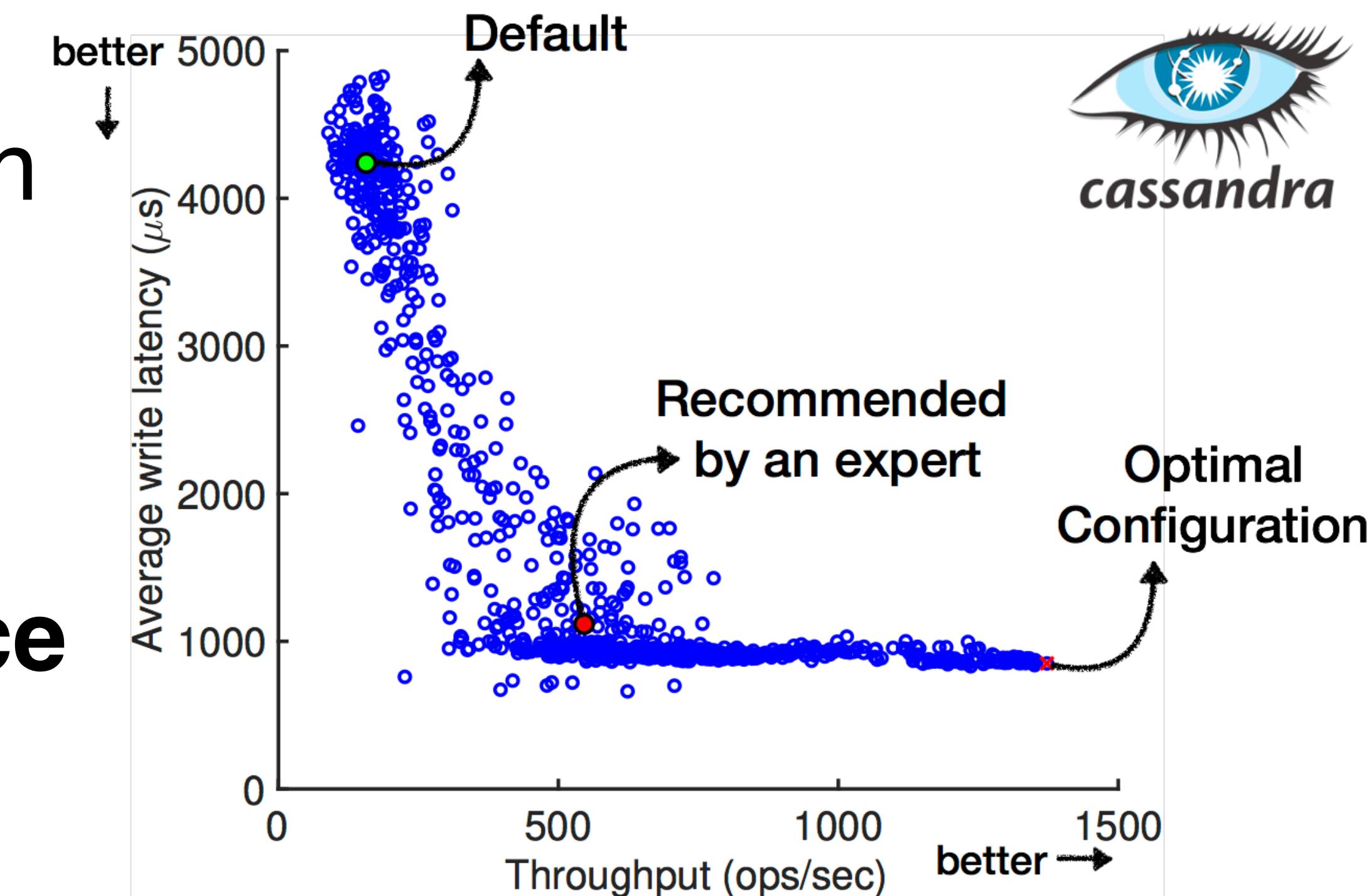
Why this is an important problem?

Optimal configuration

- 2X-10X faster than the worst
- Noticeably faster than the median
- Default is bad
- Expert's is not optimal

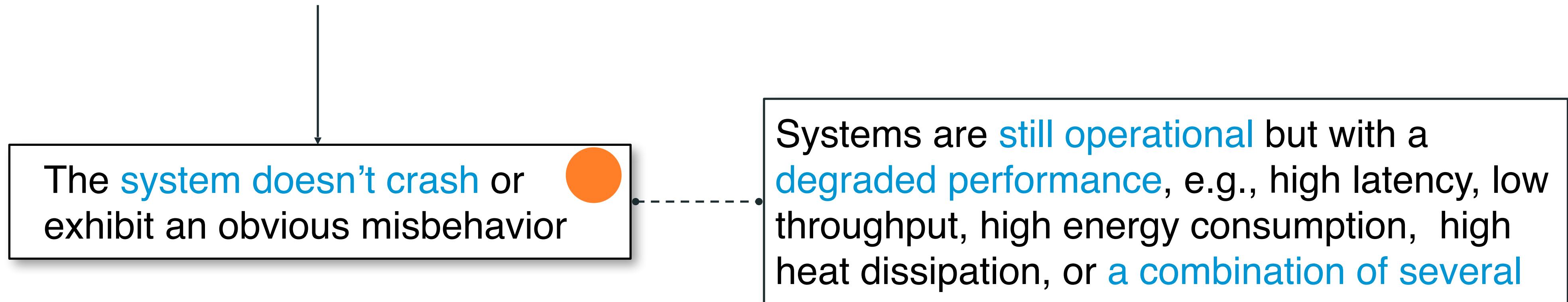
Exploring large configuration space

- Exhaustive search is expensive
- Specific to the environment
(hardware/workload/version)



Misconfiguration and its Effects

- Misconfigurations can elicit unexpected interactions between software and hardware
- These can result in non-functional faults
 - Affecting non-functional system properties like latency, throughput, energy consumption, etc.



Motivating Example



CUDA performance issue on tx2

Home > Autonomous Machines > Jetson & Embedded Systems > Jetson TX2



william_wu

When we are trying to [transplant our CUDA source code from TX1 to TX2](#), it behaved strange.

We noticed that [TX2 has twice computing-ability as TX1 in GPU](#), as expectation, [we think TX2 will 30% - 40% faster than TX1 at least](#).

Unfortunately, most of our code base spent twice the time as TX1, in other words, [TX2 only has 1/2 speed as TX1, mostly](#). We believe that TX2's CUDA API runs much slower than TX1 in many cases.

The user is transferring the code from [one hardware to another](#)

Jun '17

The [target hardware is faster](#) than the [the source hardware](#). User [expects the code to run at least 30-40% faster](#).

The [code ran 2x slower](#) on the more powerful hardware



[CUDA performance issue on tx2](#)

Motivating Example

June 3rd



william_wu

Any suggestions on how to improve my performance?
Thanks!

June 4th



AastaLLL Moderator

TX2 is pascal architecture. Please update your CMakeLists:

```
+ set(CUDA_STATIC_RUNTIME OFF)
...
+ -gencode=arch=compute_62,code=sm_62
```

June 4th



william_wu

We have already tried this. We still have high latency.
Any other suggestions?

June 5th



AastaLLL Moderator

Please do the following and let us know if it works

1. Install JetPack 3.0
2. Set nvmodel=MAX-N
3. Run jetson_clock.sh

The user had several **misconfigurations**

In Software:

- ✗ Wrong **compilation flags**
- ✗ Wrong **SDK version**

In Hardware:

- ✗ Wrong **power mode**
- ✗ Wrong **clock/fan** settings

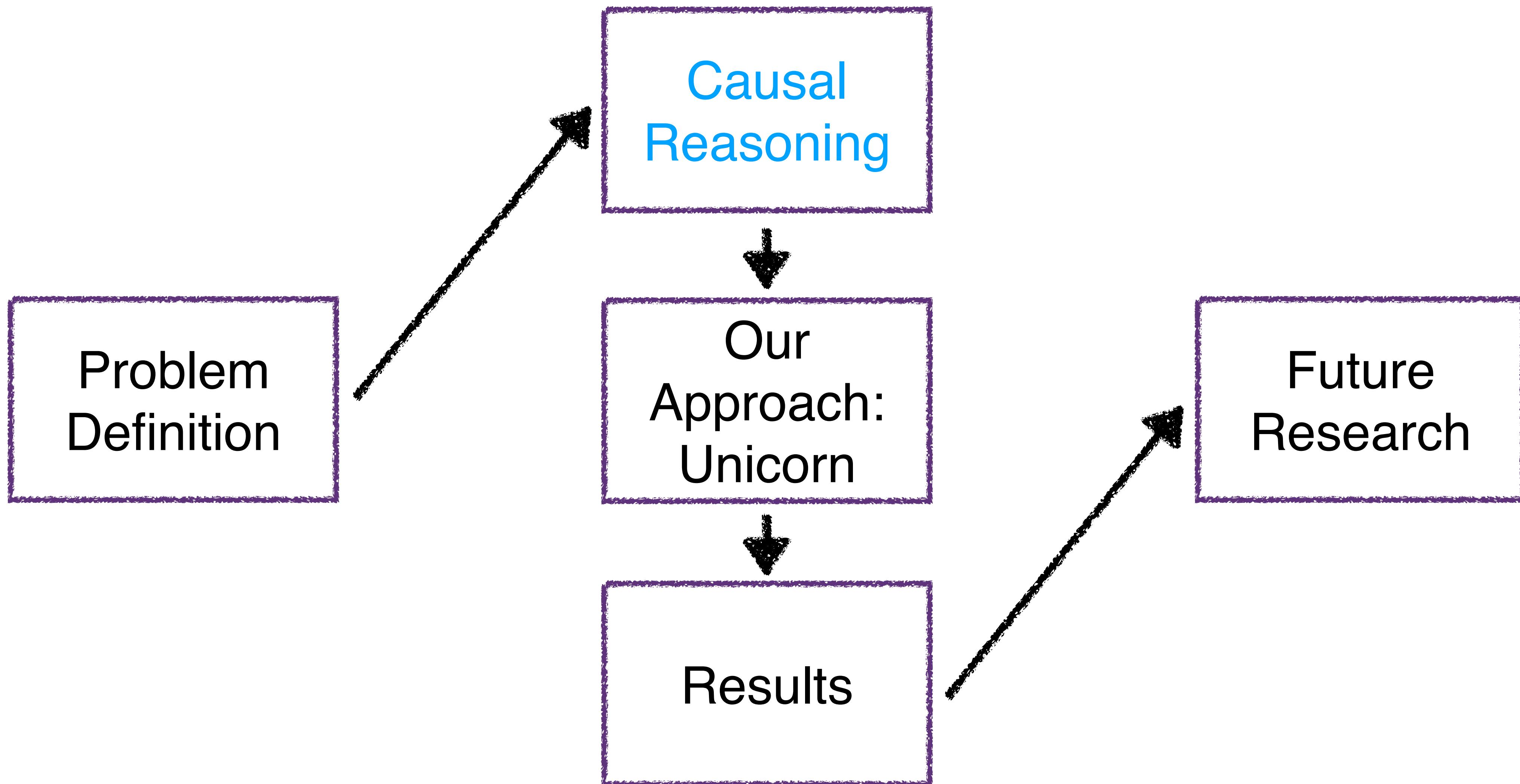
The discussions took **2 days**



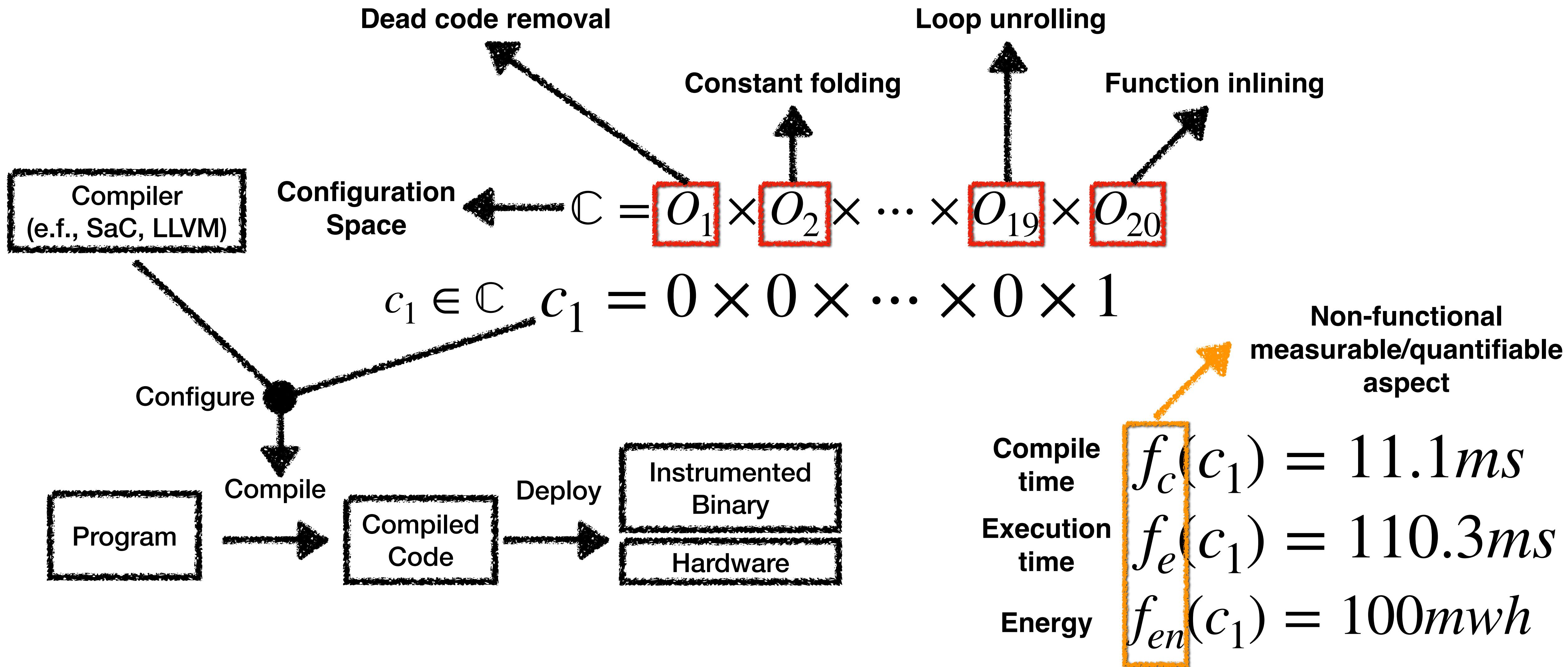
How to resolve such issues faster?

How to resolve these issues faster?

Outline



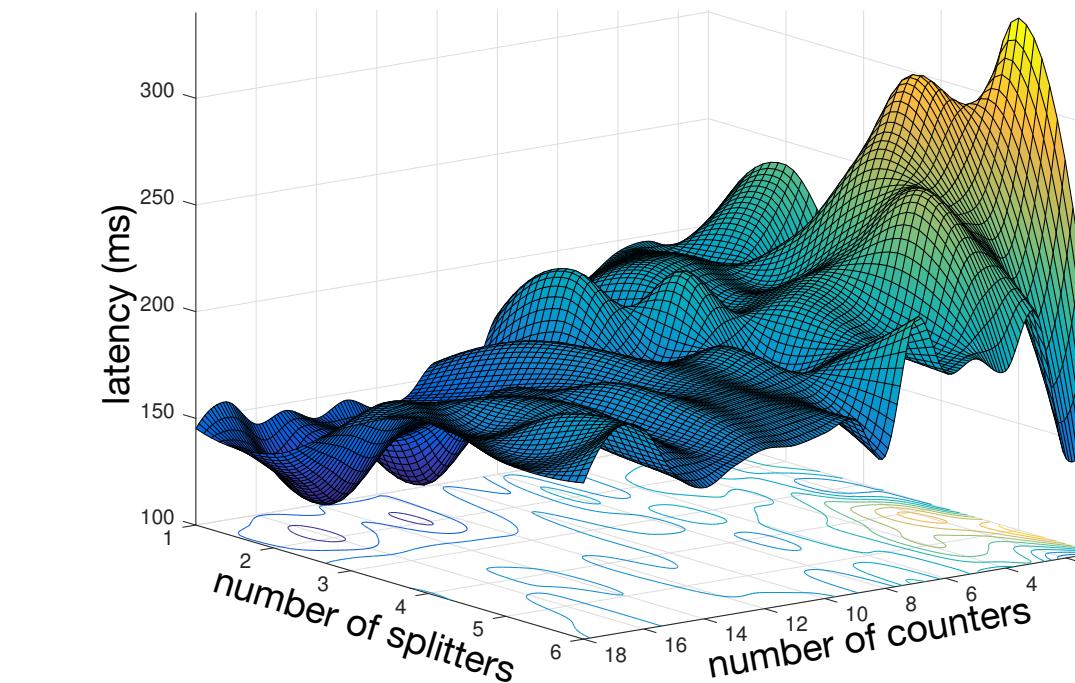
Performance measurement



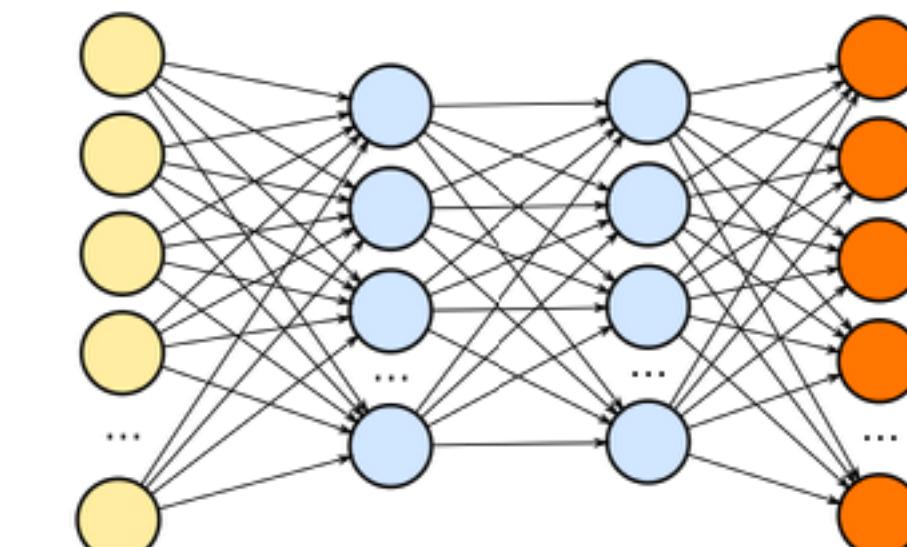
Blackbox Performance Modeling

	Bitrate (bits/s)	EnableP adding	...	Cache Misses	...	Throughput (fps)
c_1	1k	1	...	42m	...	7
c_2	2k	1	...	32m	...	22
...
c_n	5k	0	...	12m	...	25

Observational Data



Gaussian Process



Neural Network

$$\begin{aligned} \text{Throughput} = & \boxed{5.1 \times \text{Bitrate}} + \boxed{2.5 \times \text{BatchSize}} \\ & + \boxed{12.3 \times \text{Bitrate} \times \text{BatchSize}} \end{aligned}$$

Interactions

Polynomial Regression

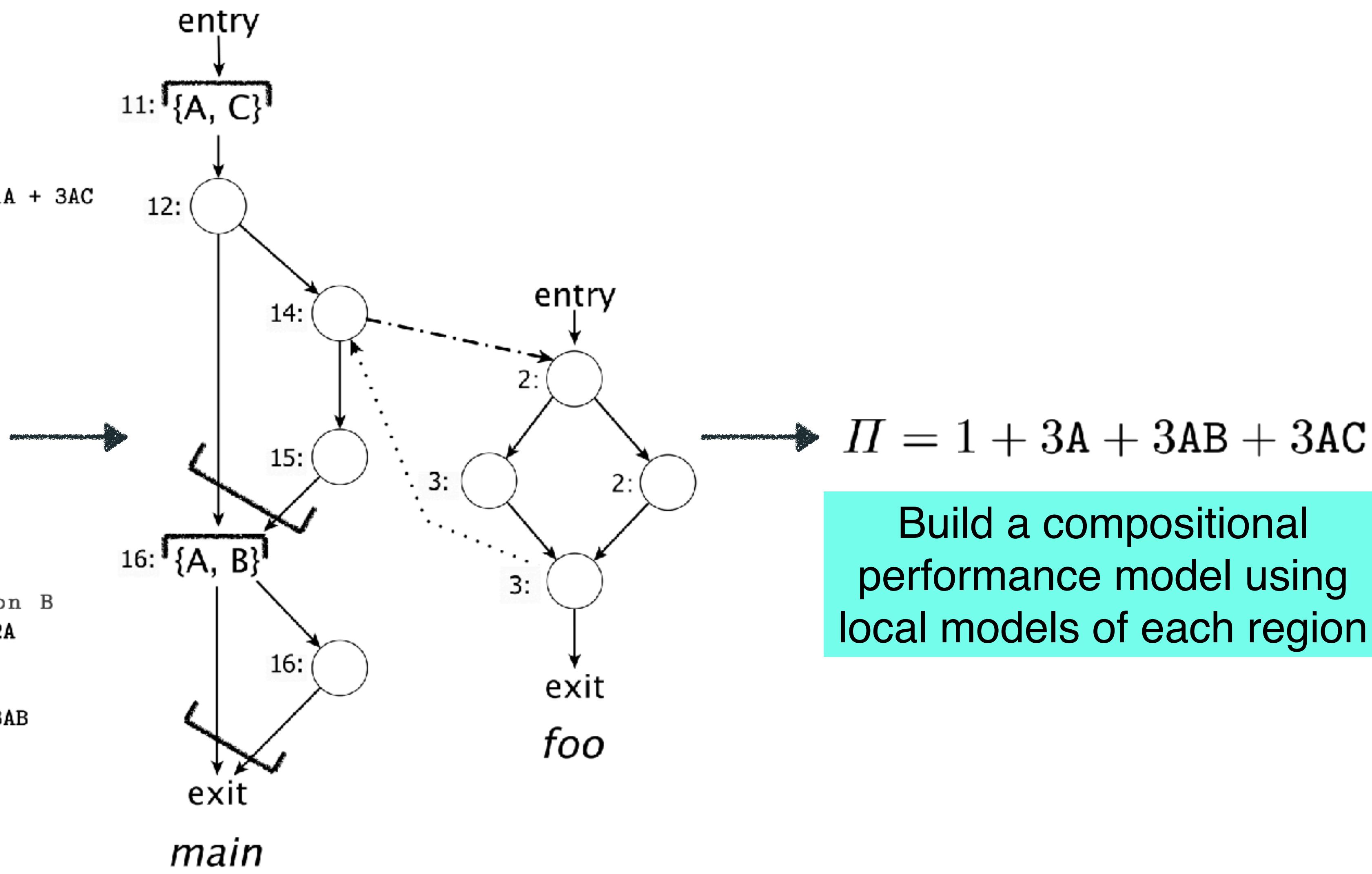
These methods rely on **statistical correlations** to **extract meaningful information** required for performance tasks.

Whitebox Performance Modeling

```

def foo(boolean x)
    // Begin region R1
    if(x) ... // execution: 4s       $\Pi_{R_1} = 1A + 3AC$ 
    else ... // execution 1s
    // End region R1
def main(List workload)
    a = getopt("A"); b = getopt("B");
    c = getopt("C"); d = getopt("D");
    e = getopt("E"); f = getopt("F");
    g = getopt("G"); h = getopt("H");
    i = getopt("I"); j = getopt("J");
    ... // execution: 1s
    boolean x = false;
    // Begin region R2
    if(a) // variable depends on option A
        ... // execution: 2s
        foo(c); // variable depends on option B
        x = true;
    // End region R2
    // Begin region R3
    if(b && x) ... // execution: 3s   $\Pi_{R_3} = 3AB$ 
    // End region R3

```



Identify configuration-dependent regions

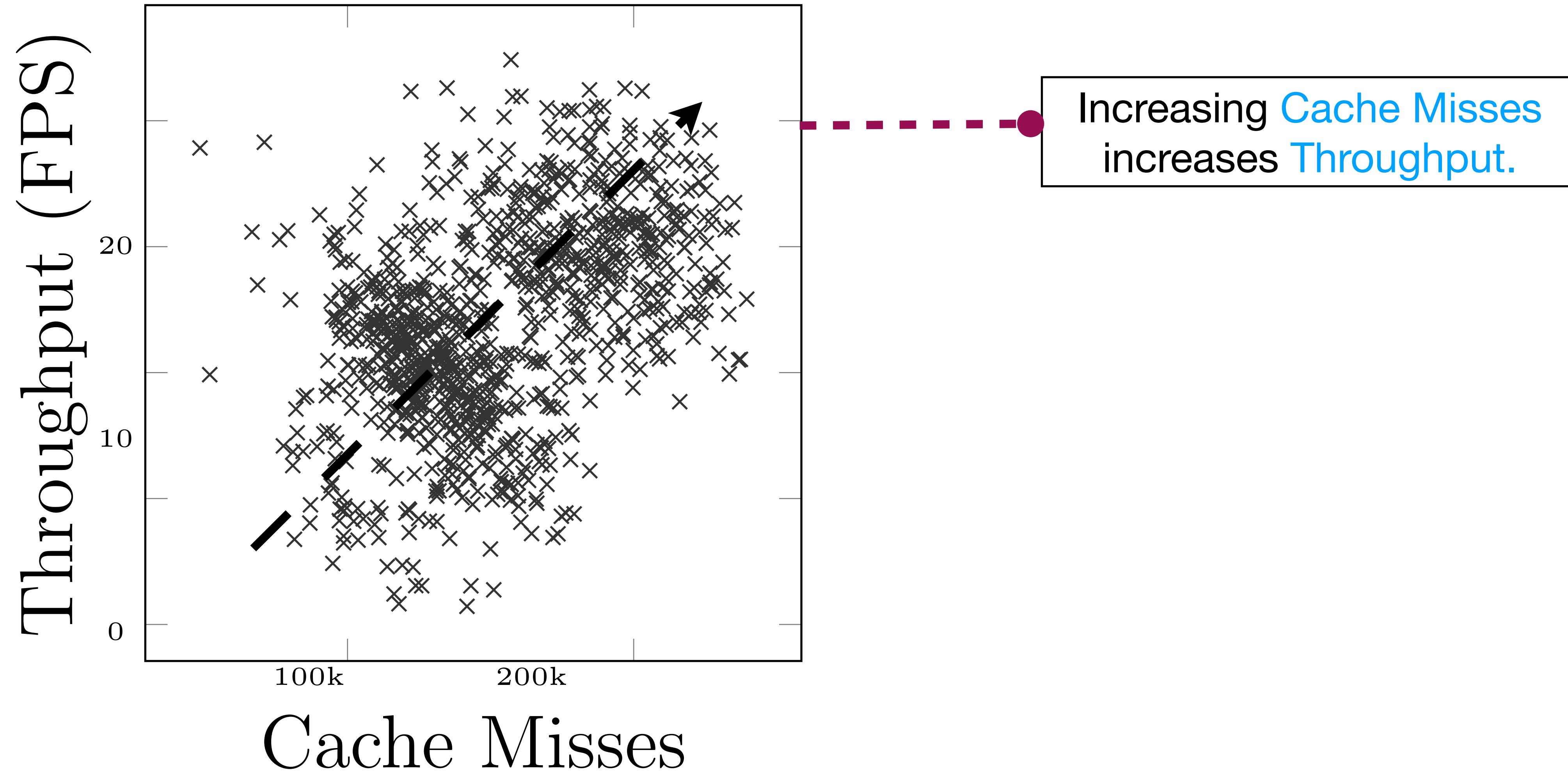
Instrumented control flow graph

These methods rely on program analysis techniques (static and dynamic analysis of the code) to extract meaningful information required for performance tasks.

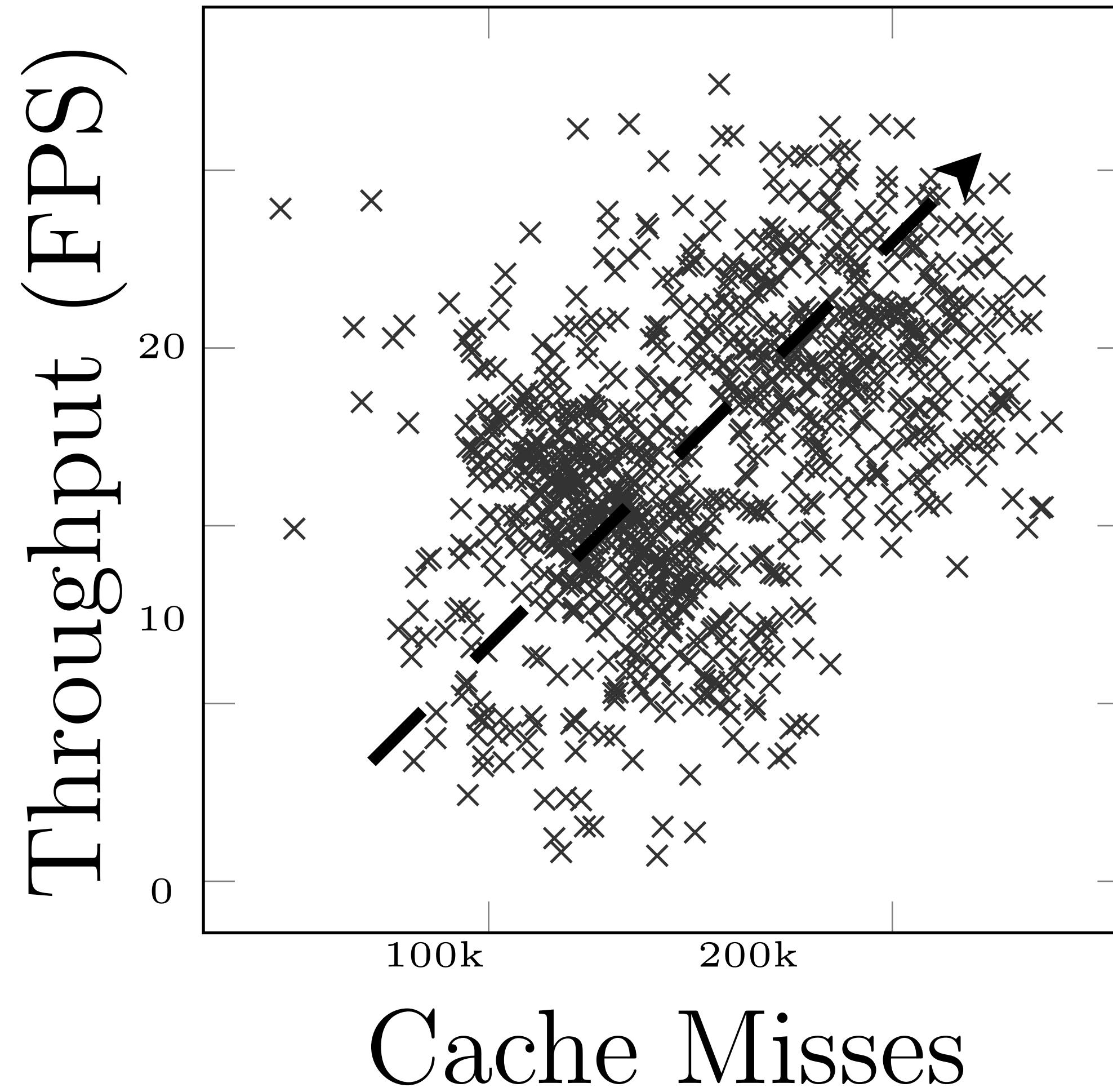
Performance models suffer from several shortcomings

- Blackbox performance models could produce **incorrect explanations and unreliable/unstable predictions** across environments and in the presence of measurement noise.
- Whitebox performance models **do not scale well** to real-world systems (with many configuration options and large code bases).

Incorrect explanation



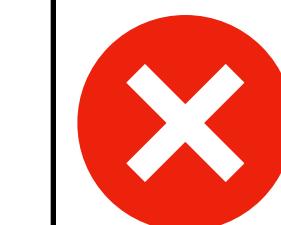
Incorrect explanation



Increasing Cache Misses increases Throughput.

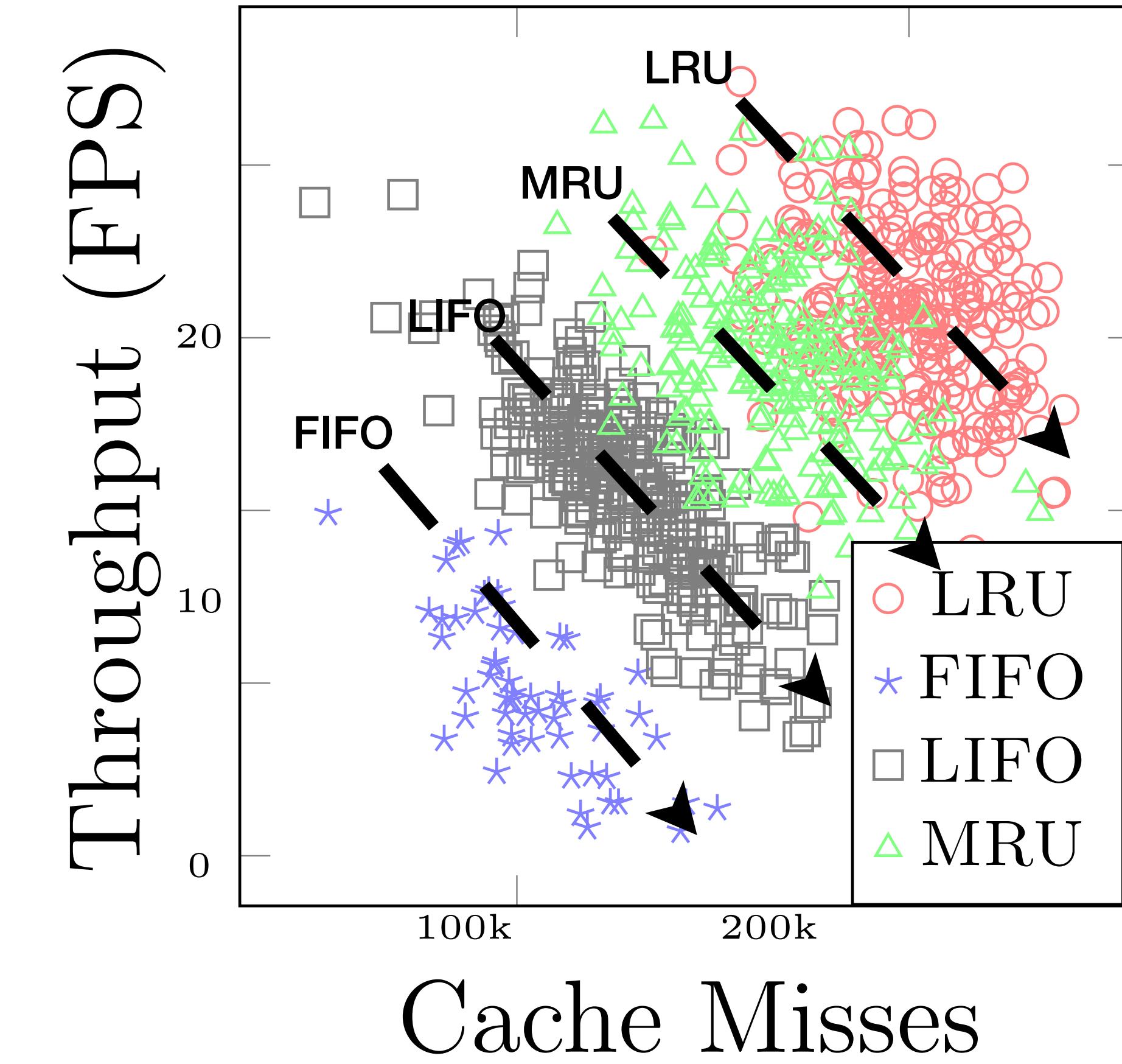
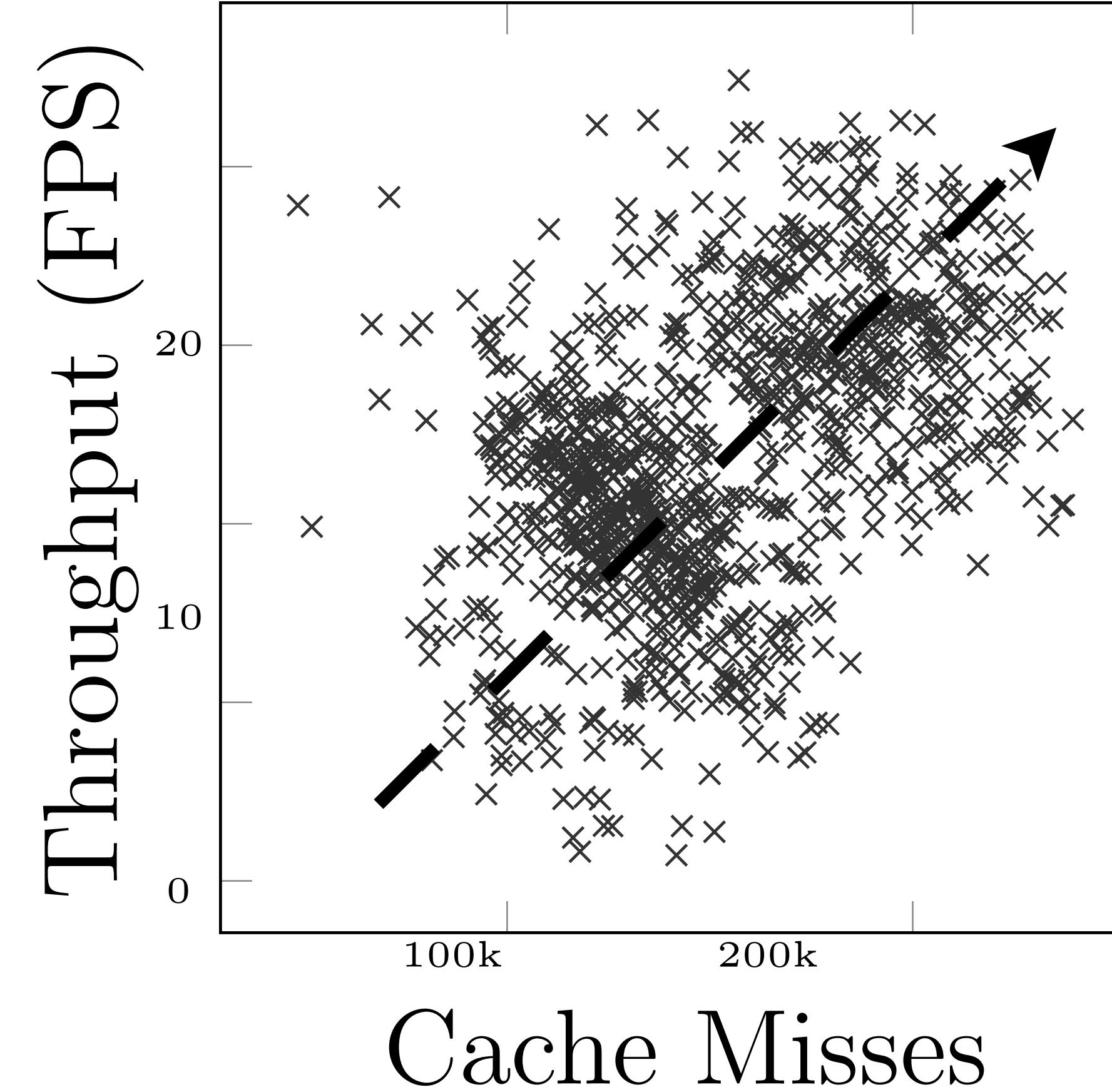
This is counter-intuitive

More Cache Misses should reduce Throughput not increase it



Any statistical models built on this data will be incorrect.

Incorrect explanation



Segregating data on **Cache Policy** indicates that within **each group**
Increase of Cache Misses result in a **decrease in Throughput**.

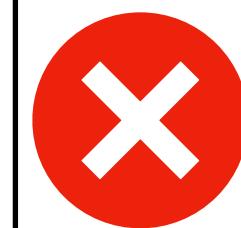
Unstable predictions

Performance influence model in TX2:

$$\begin{aligned} \text{Throughput} = & 2 \times \text{Bitrate} + 1.9 \times \text{BatchSize} + [1.8 \times \text{BufferSize}] + [0.5 \times \text{EnablePadding}] + [5.9 \times \text{Bitrate} \times \text{BufferSize}] \\ & + [6.2 \times \text{Bitrate} \times \text{EnablePadding}] + [4.1 \times \text{Bitrate} \times \text{BufferSize} \times \text{EnablePadding}] \end{aligned}$$

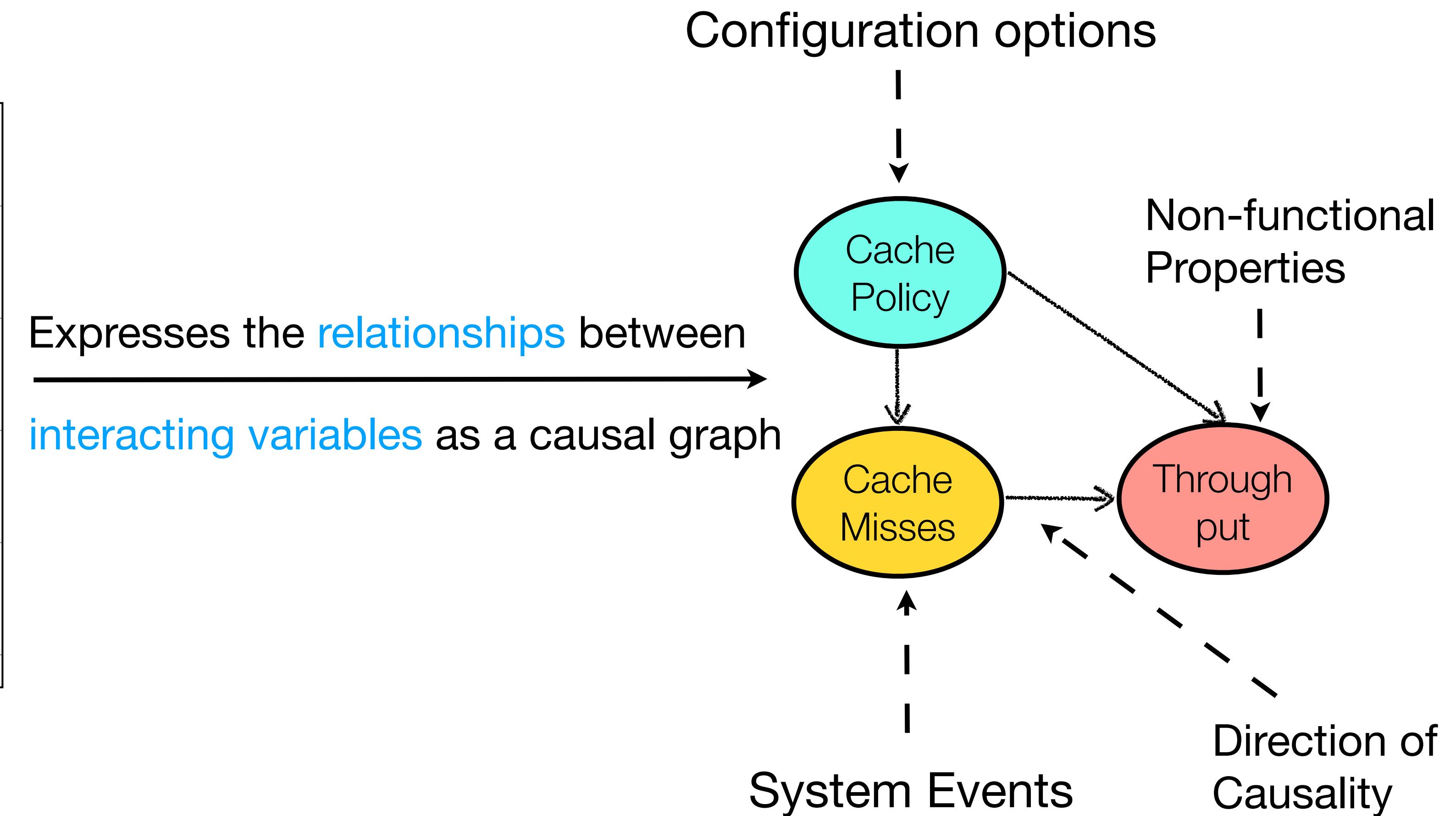
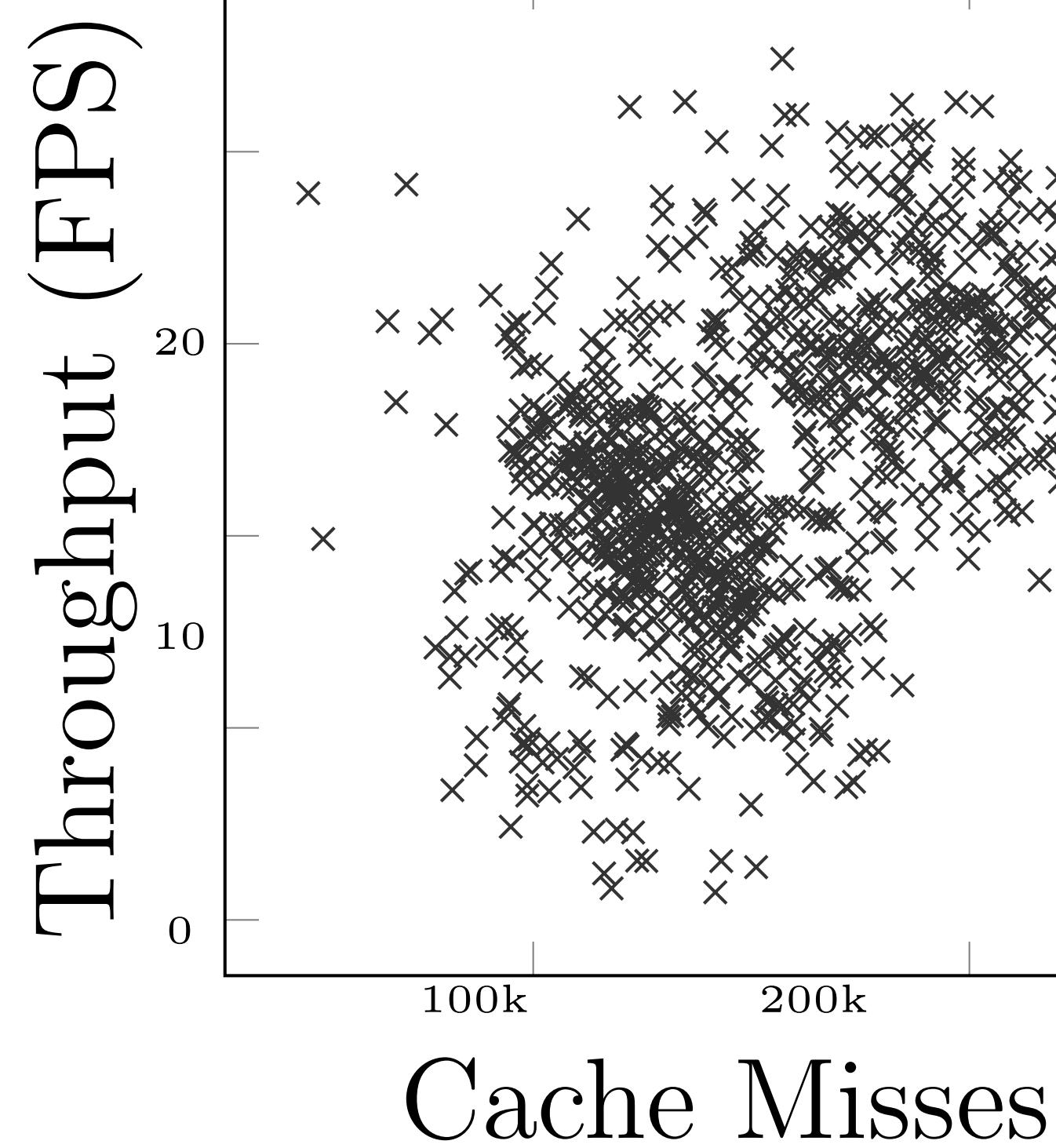
Performance influence model in Xavier:

$$\text{Throughput} = 5.1 \times \text{Bitrate} + 2.5 \times \text{BatchSize} + [12.3 \times \text{Bitrate} \times \text{BatchSize}]$$

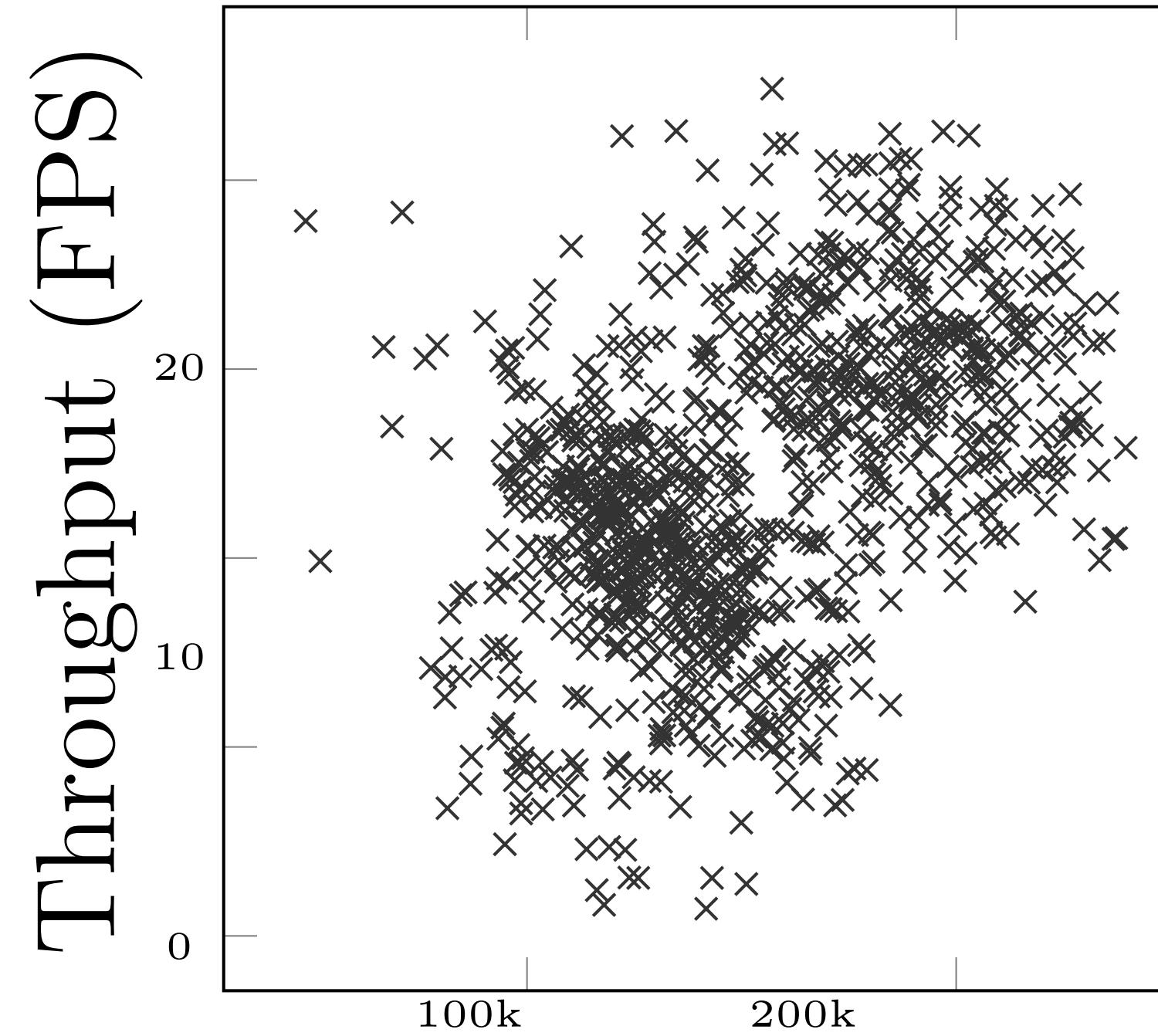


Performance influence models **change significantly** across environments, resulting in low accuracy in new environments.

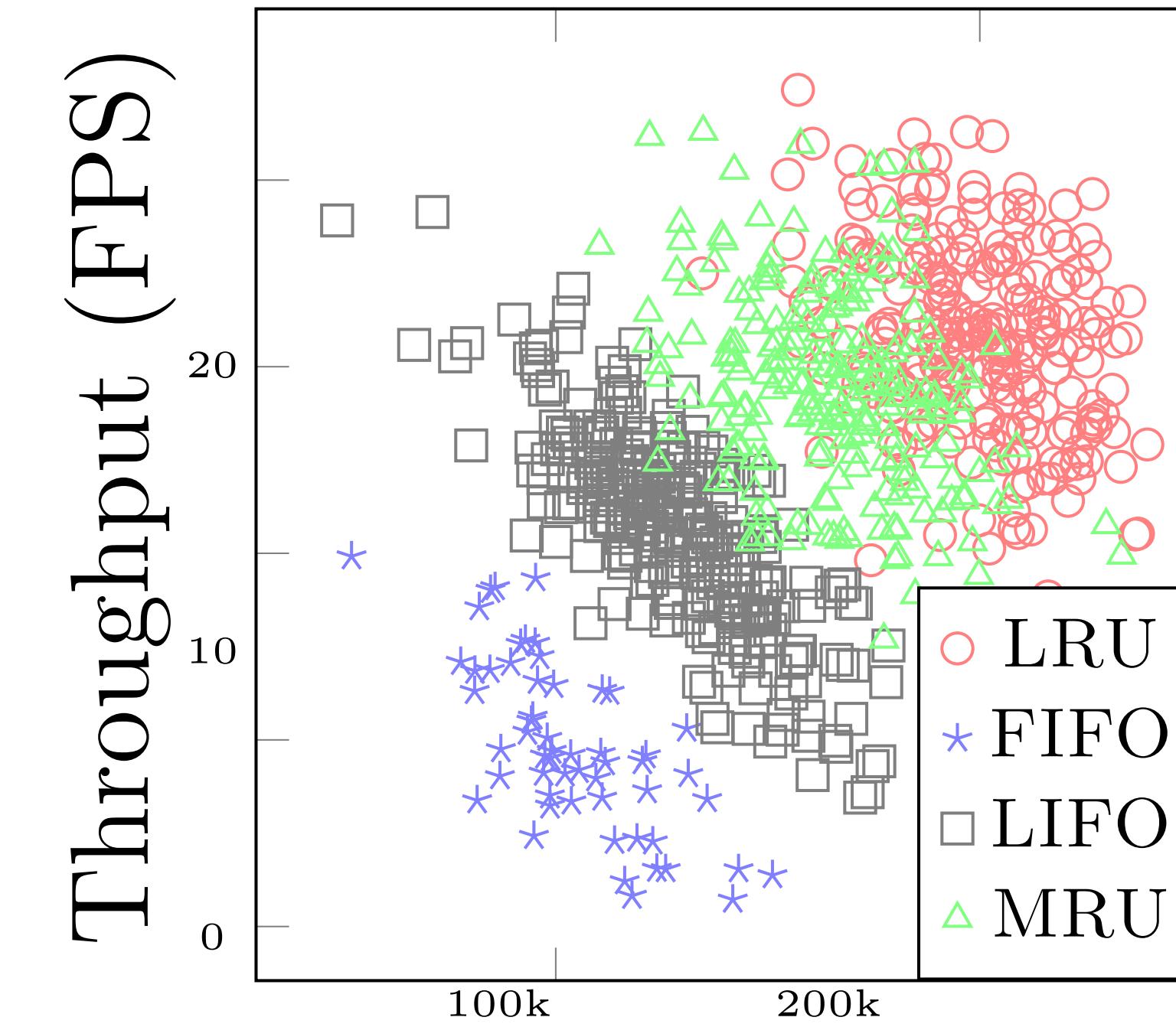
Causal performance modeling



Causal performance models produce correct explanations

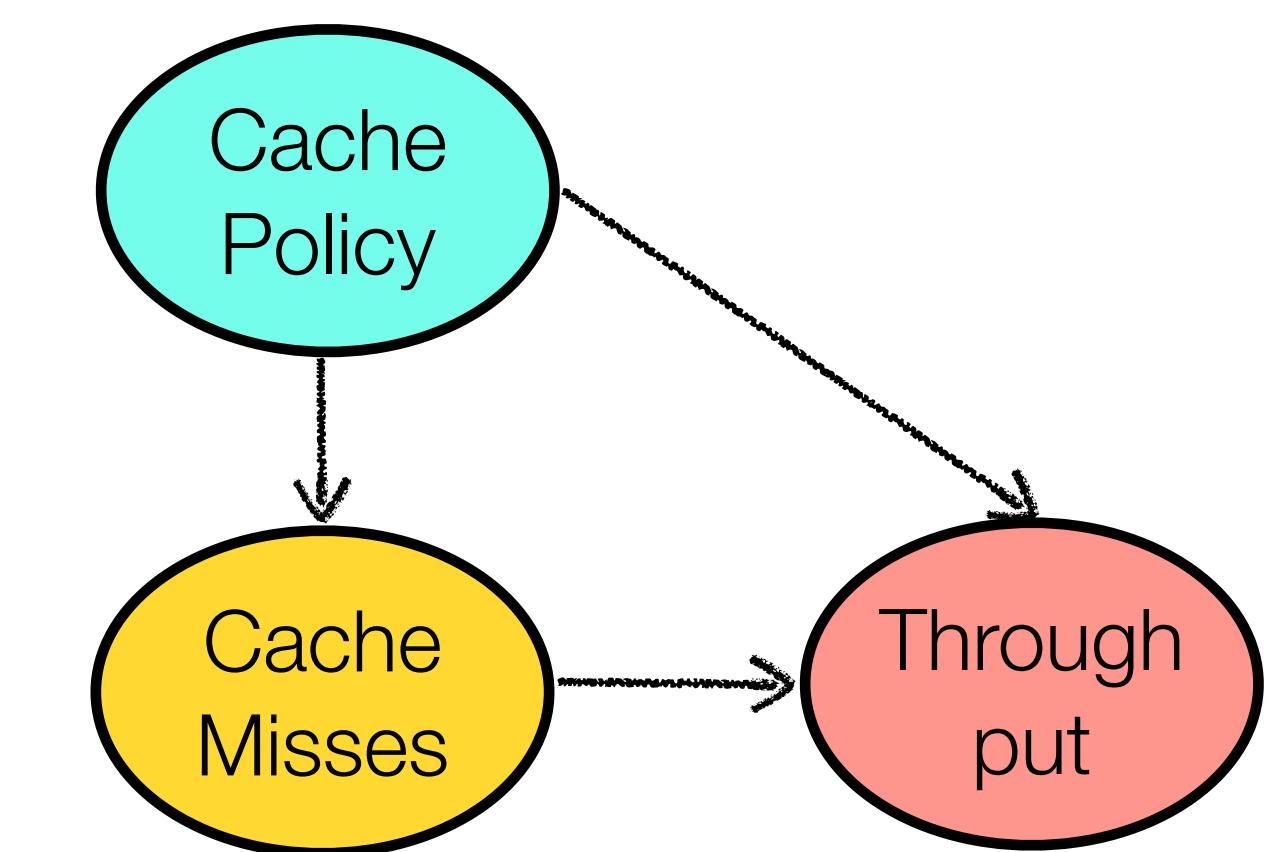


Cache Misses



Cache Misses

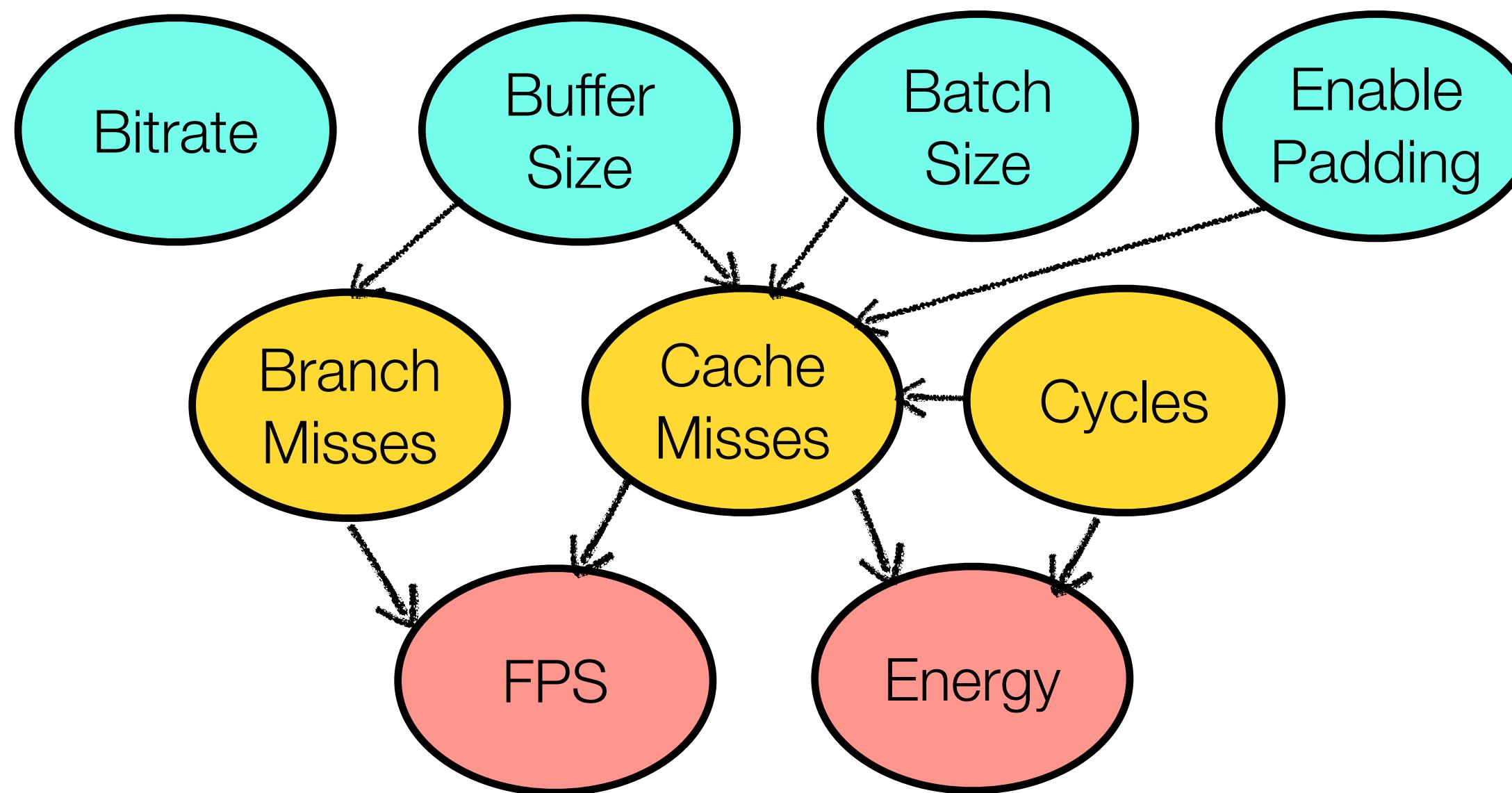
Cache Policy affects Throughput via Cache Misses.



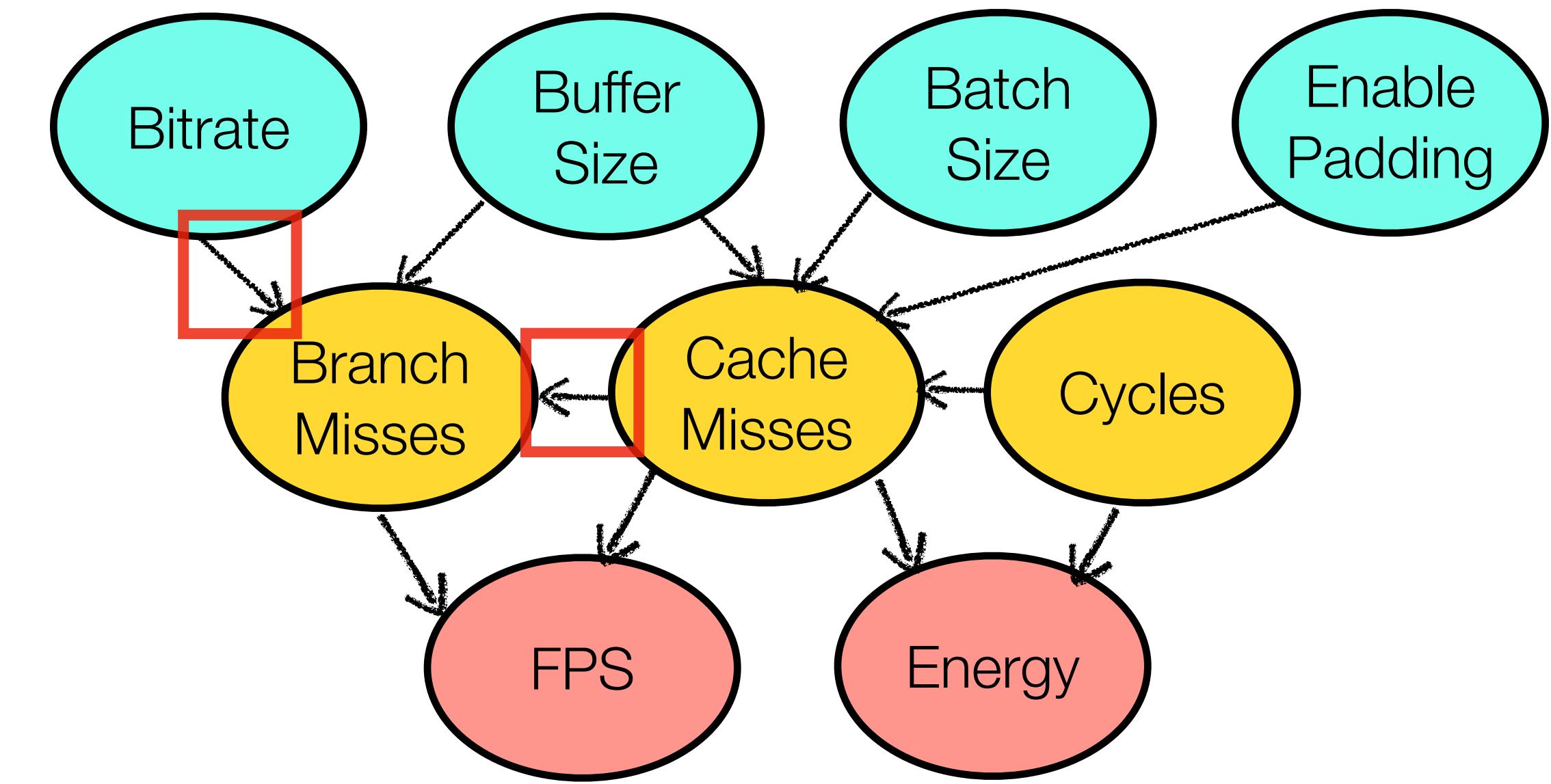
Causal performance models capture correct interactions.

Causal performance models are transferable across environments

A partial causal performance model in Jetson TX2

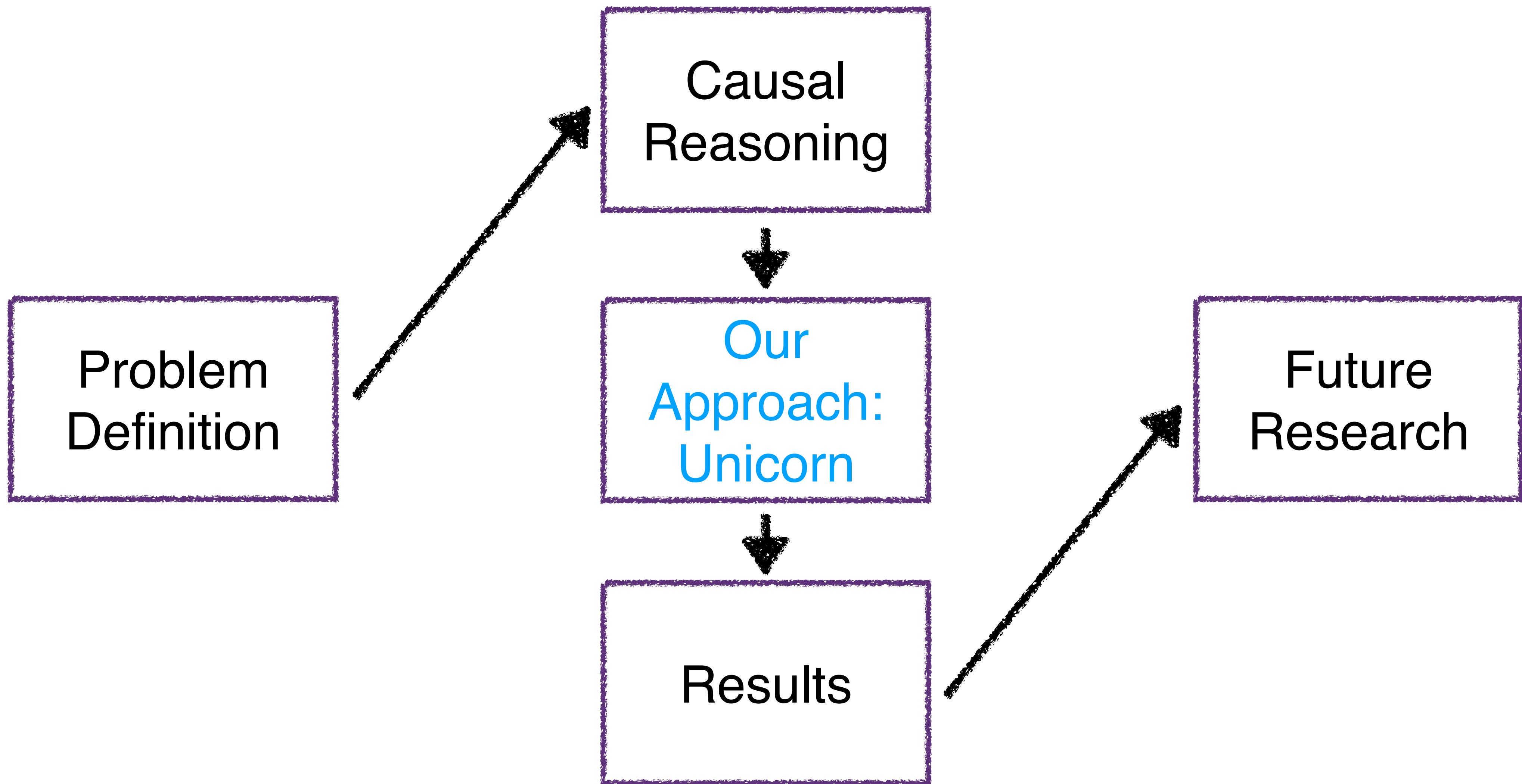


A partial causal performance model in Jetson Xavier

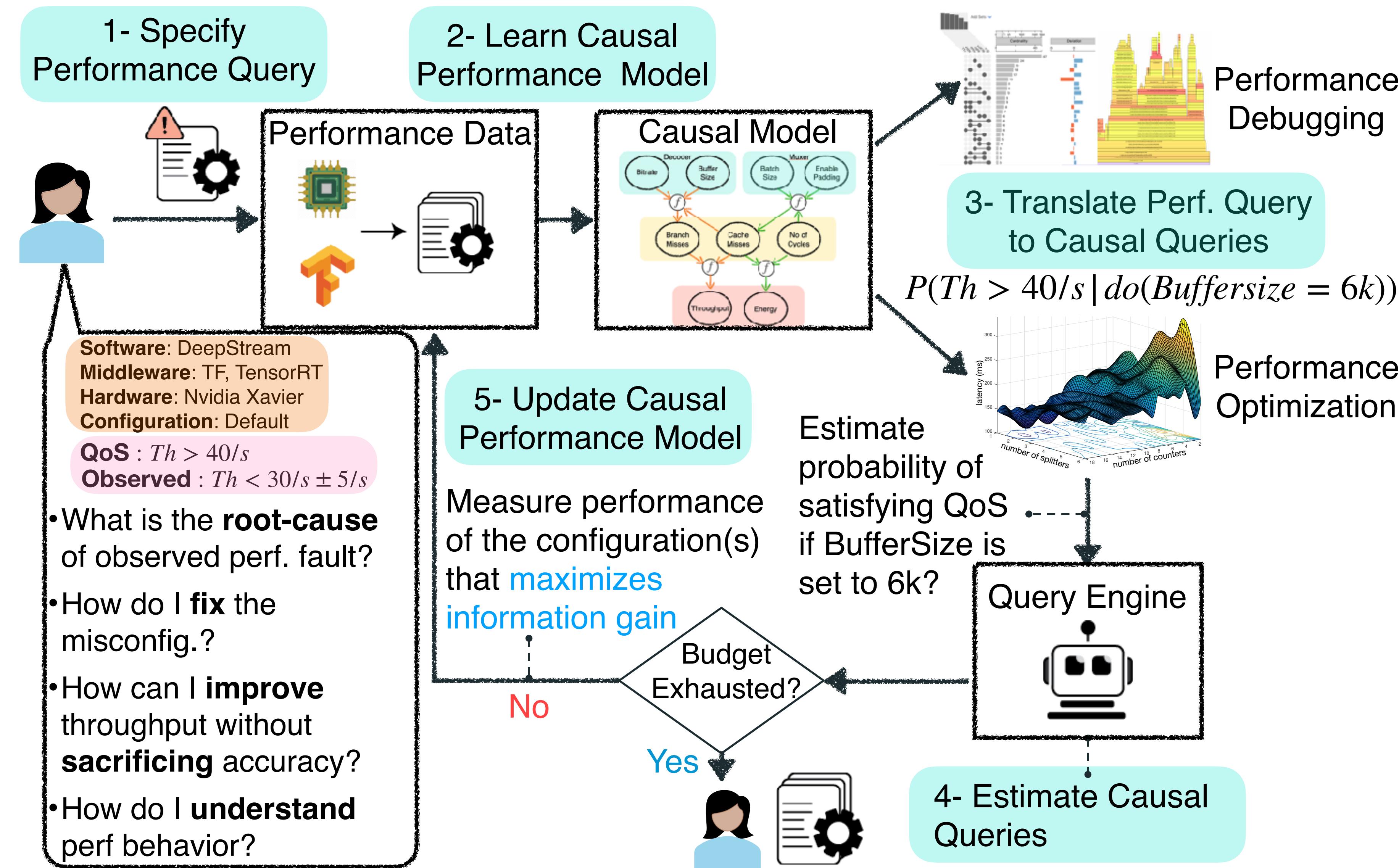


Causal performance models remain relatively **stable** across environments.

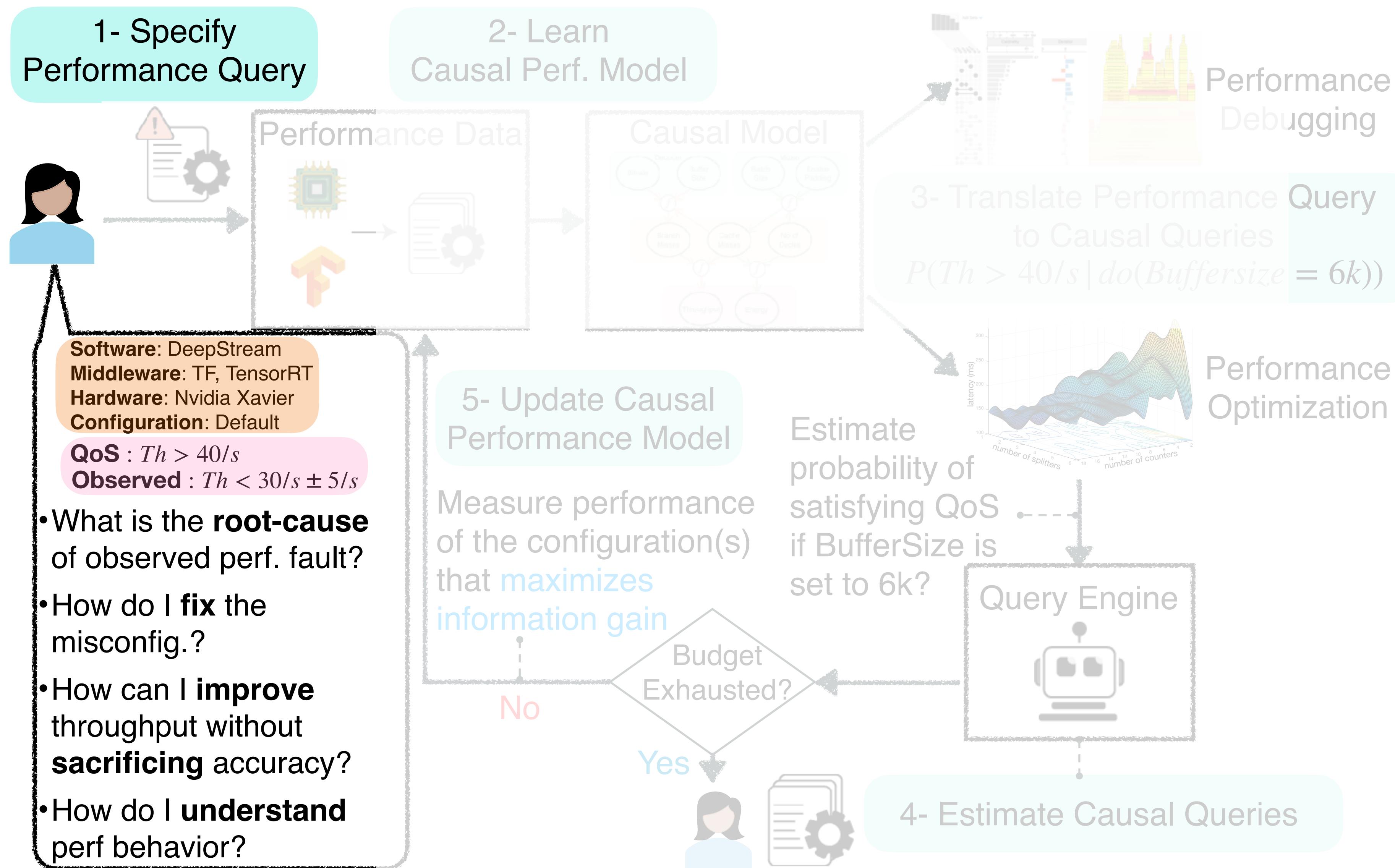
Outline



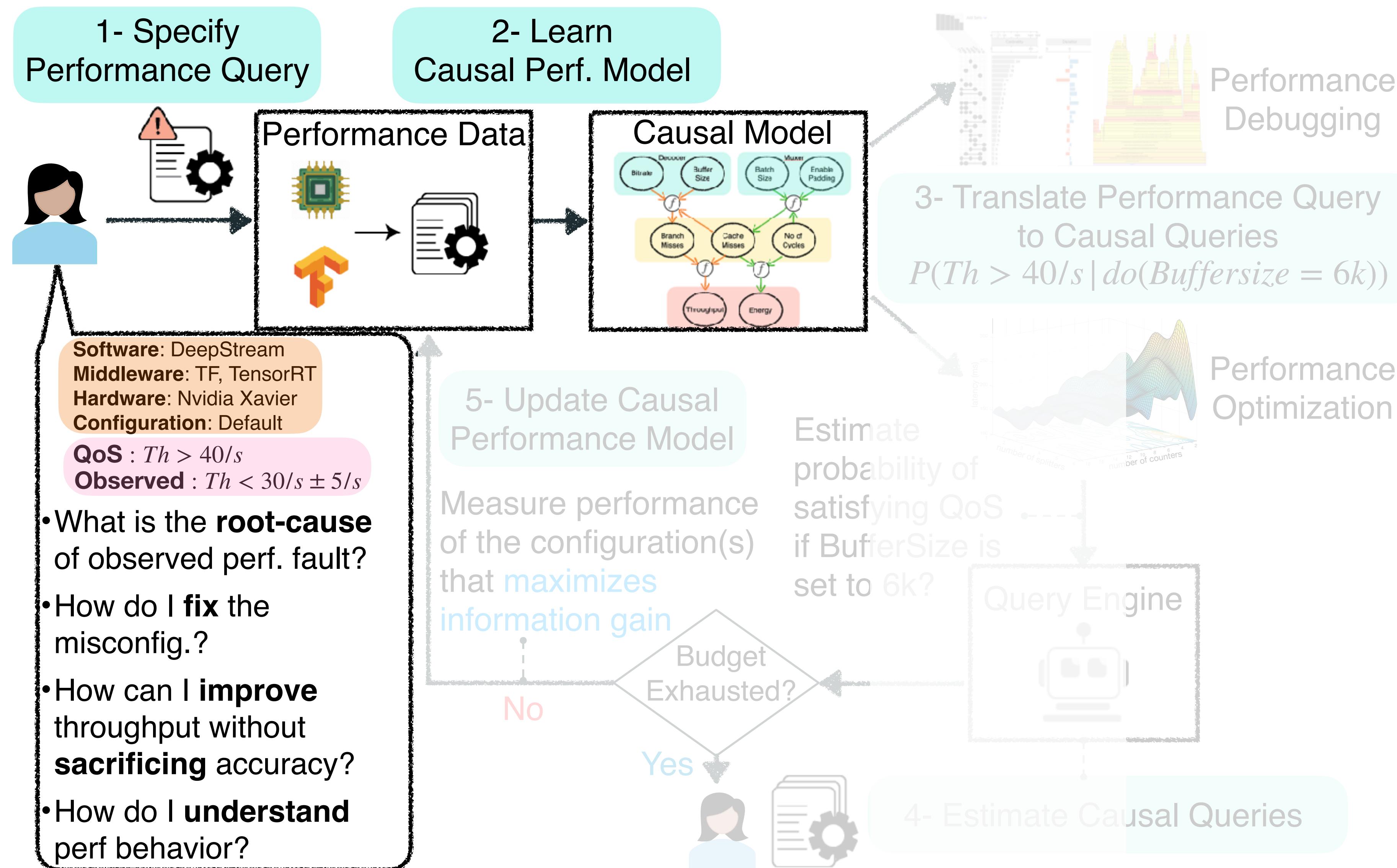
UNICORN: Performance Reasoning through the Lens of Causality



UNICORN: Performance Reasoning through the Lens of Causality



UNICORN: Performance Reasoning through the Lens of Causality



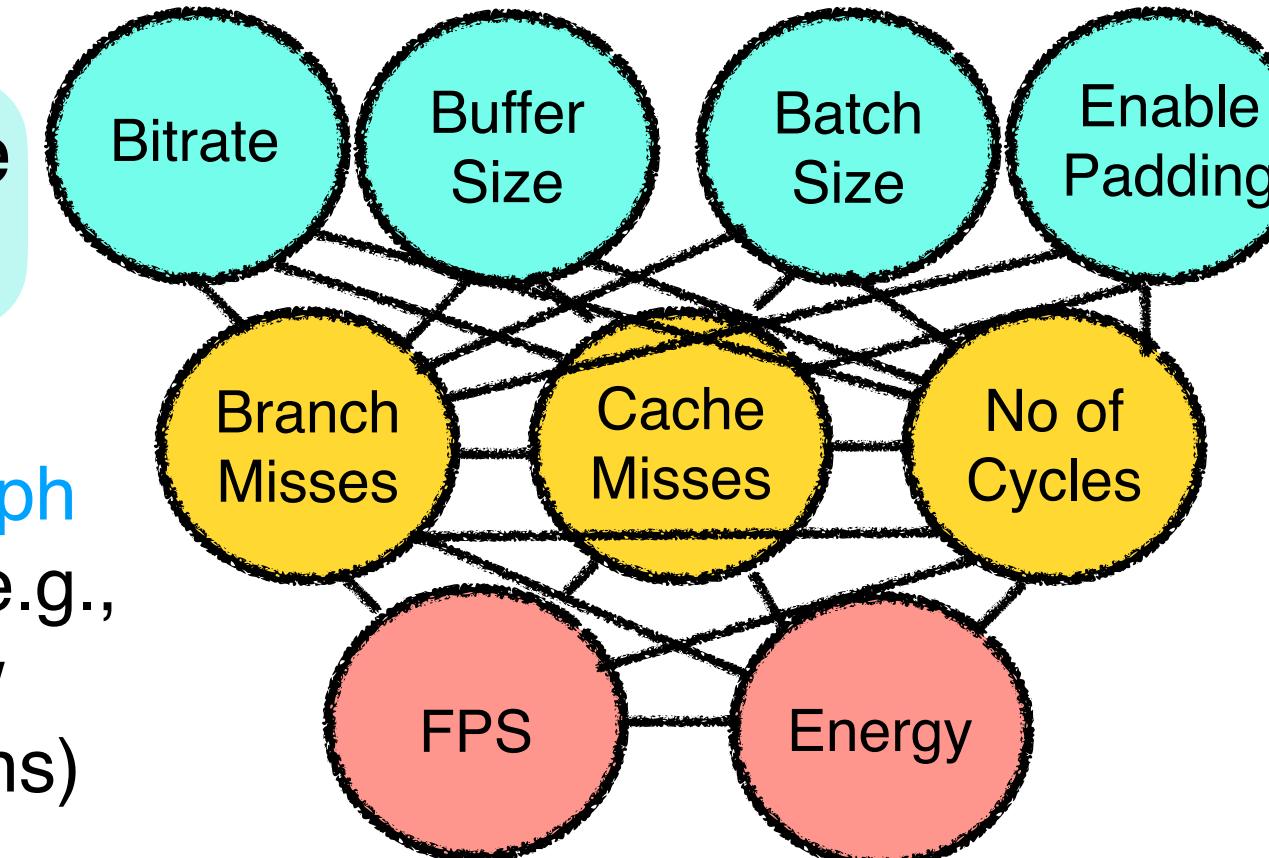
Learning Causal Performance Model

	Bitrate (bits/s)	Enable Padding	...	Cache Misses	...	Through put (fps)
c ₁	1k	1	...	42m	...	7
c ₂	2k	1	...	32m	...	22
...
c _n	5k	0	...	12m	...	25

1- Recovering the Skelton



fully connected graph given constraints (e.g., no connections btw configuration options)



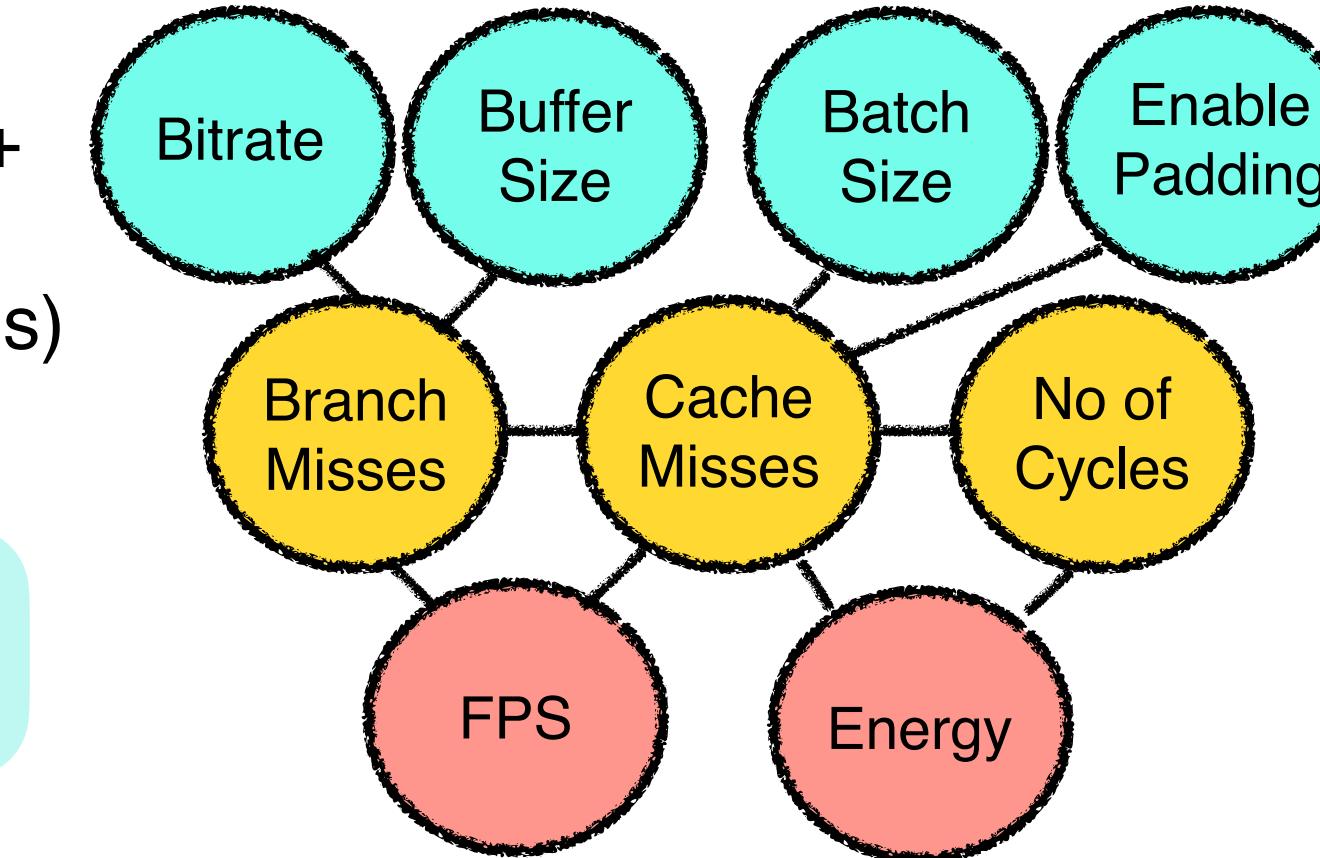
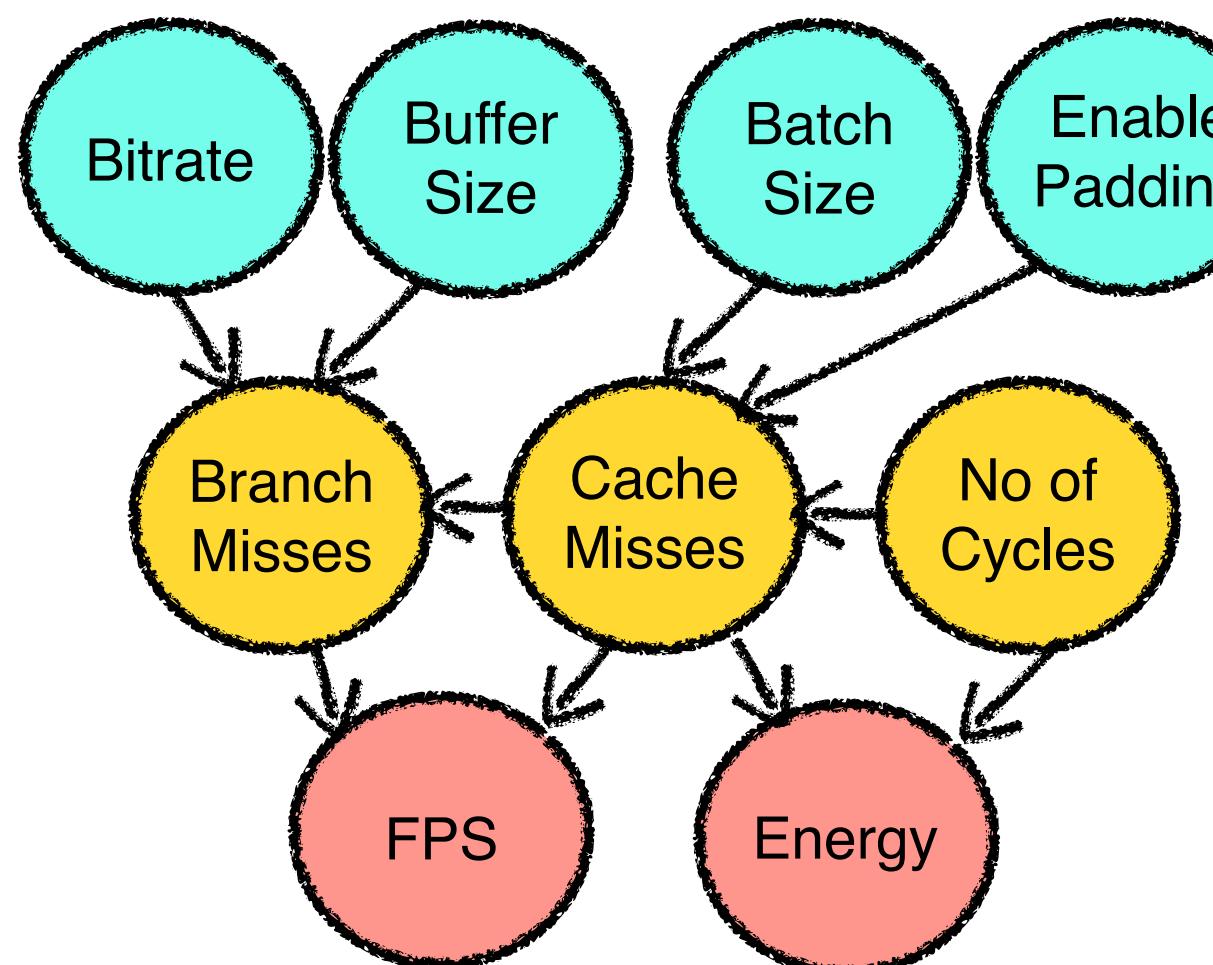
statistical independence tests

2- Pruning Causal Structure

orientation rules & measures (entropy) + structural constraints (colliders, v-structures)



3- Orienting Causal Relations



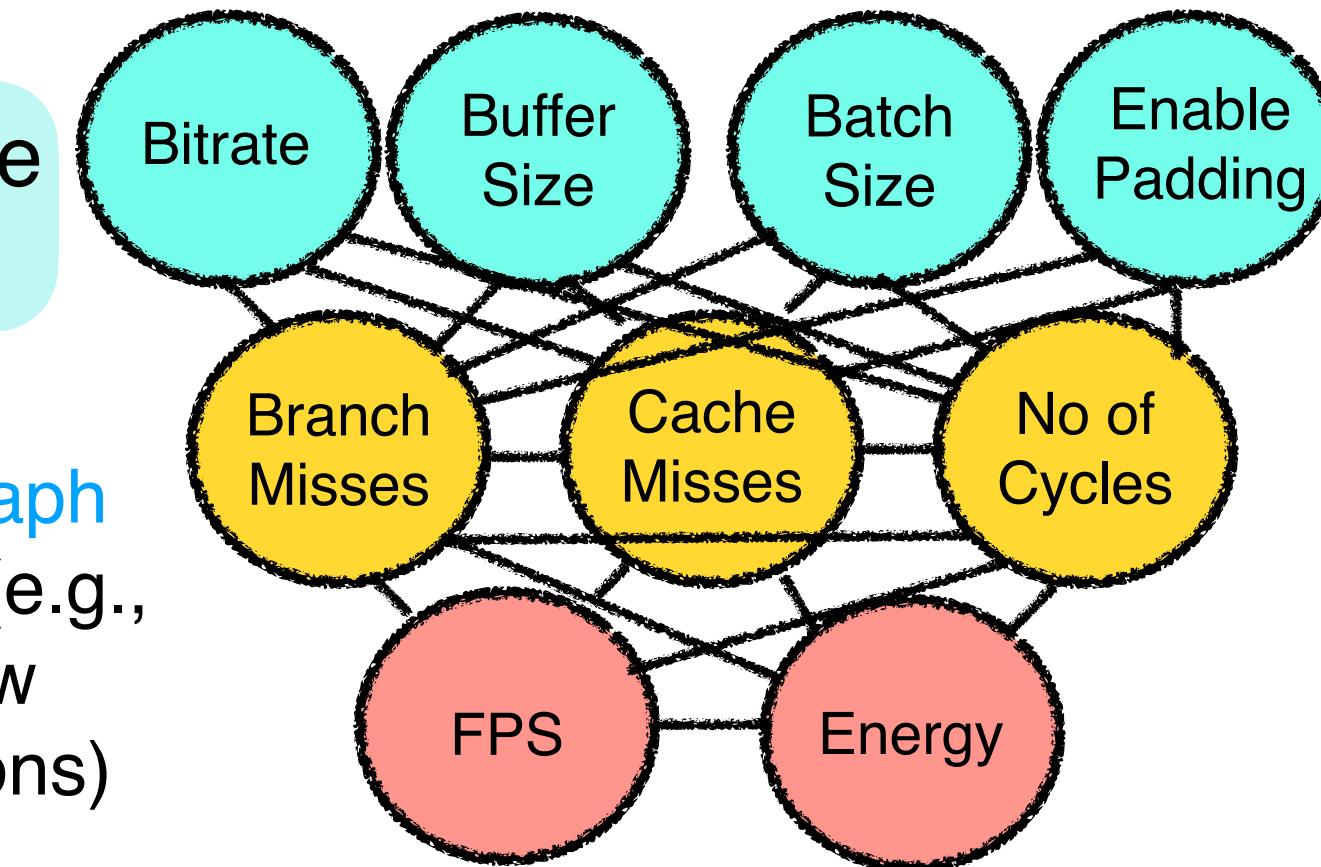
Learning Causal Performance Model

	Bitrate (bits/s)	Enable Padding	...	Cache Misses	...	Through put (fps)
c ₁	1k	1	...	42m	...	7
c ₂	2k	1	...	32m	...	22
...
c _n	5k	0	...	12m	...	25

1- Recovering the Skelton



fully connected graph given constraints (e.g., no connections btw configuration options)

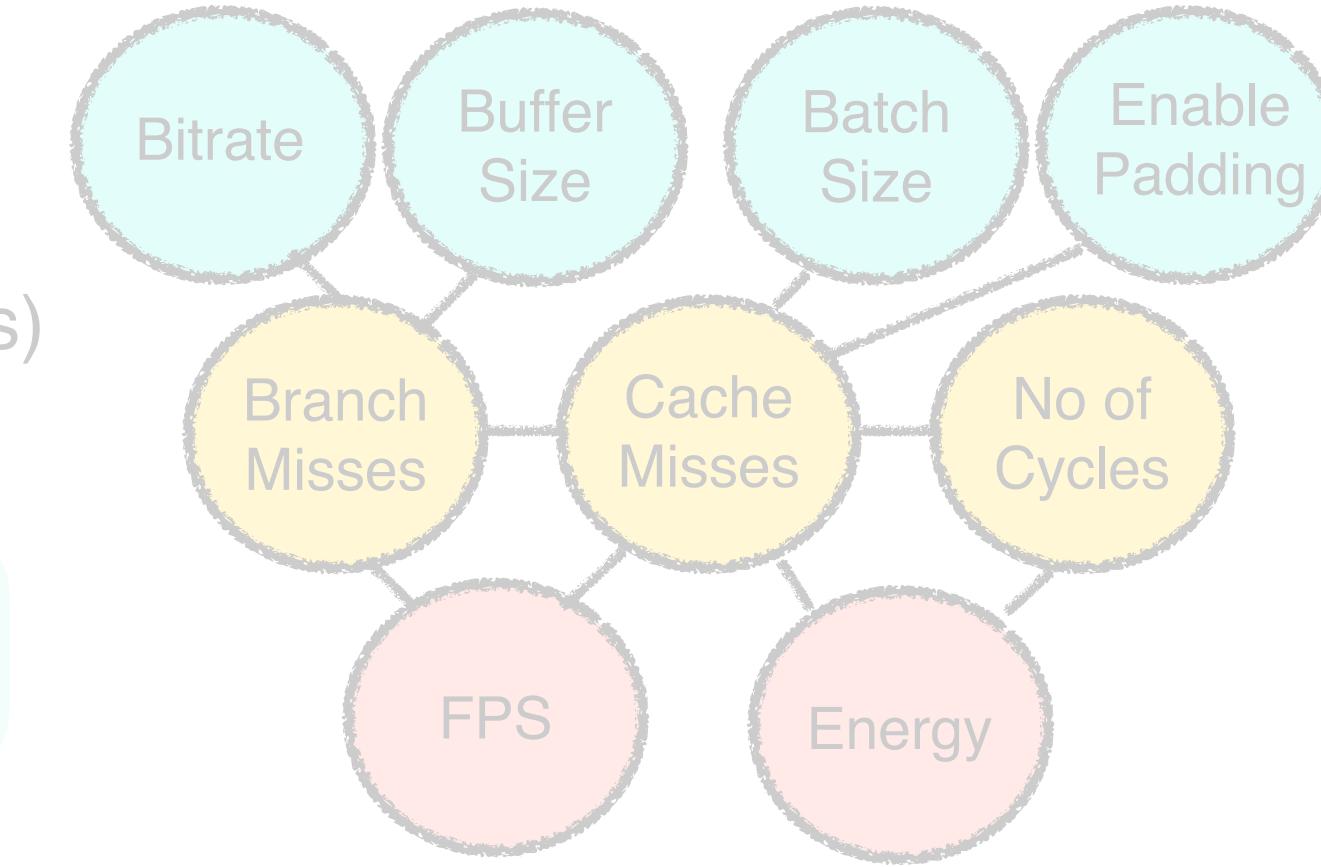
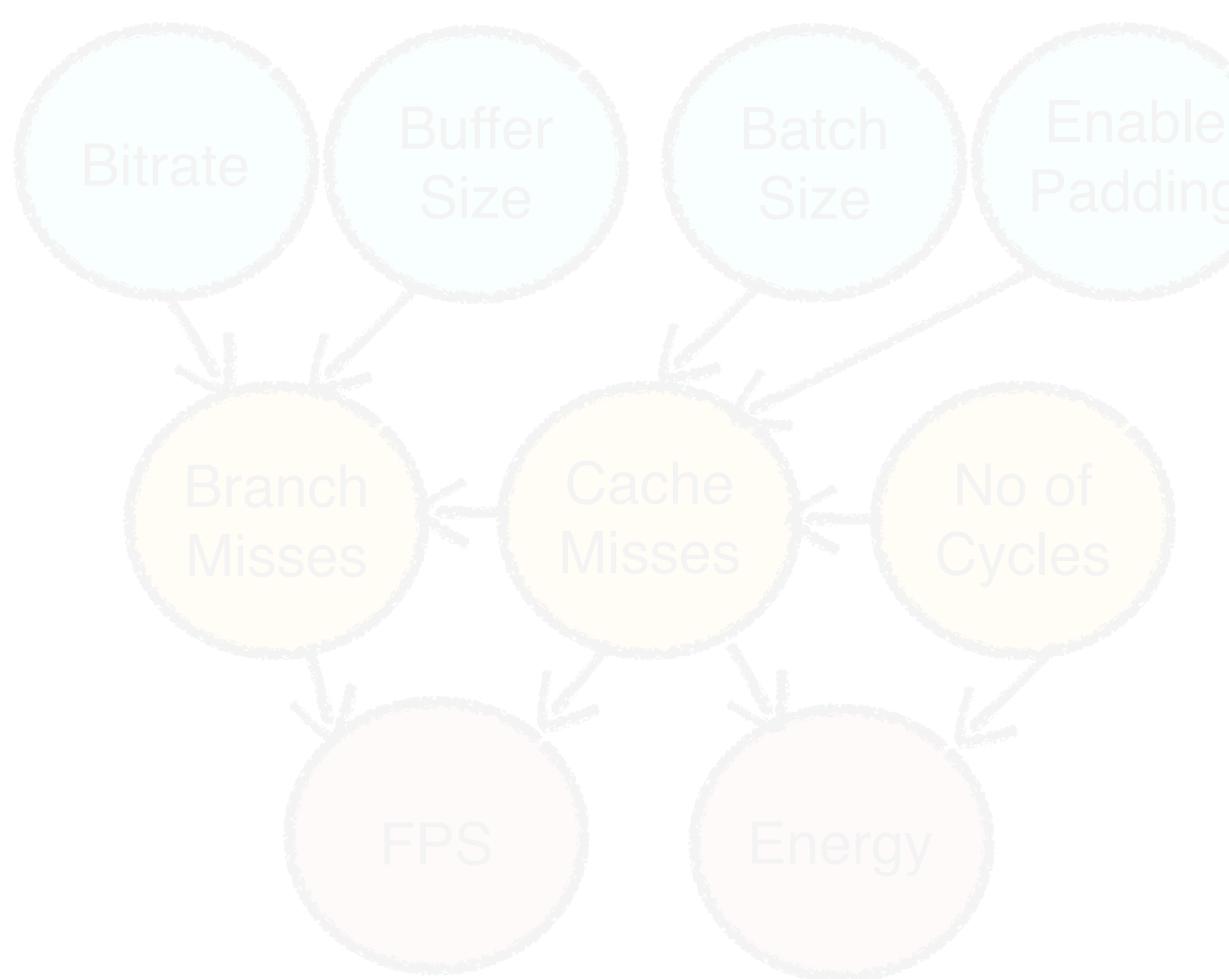


statistical independence

2- Pruning Causal Structure

orientation rules & measures (entropy) + structural constraints (colliders, v-structures)

3- Orienting Causal Relations



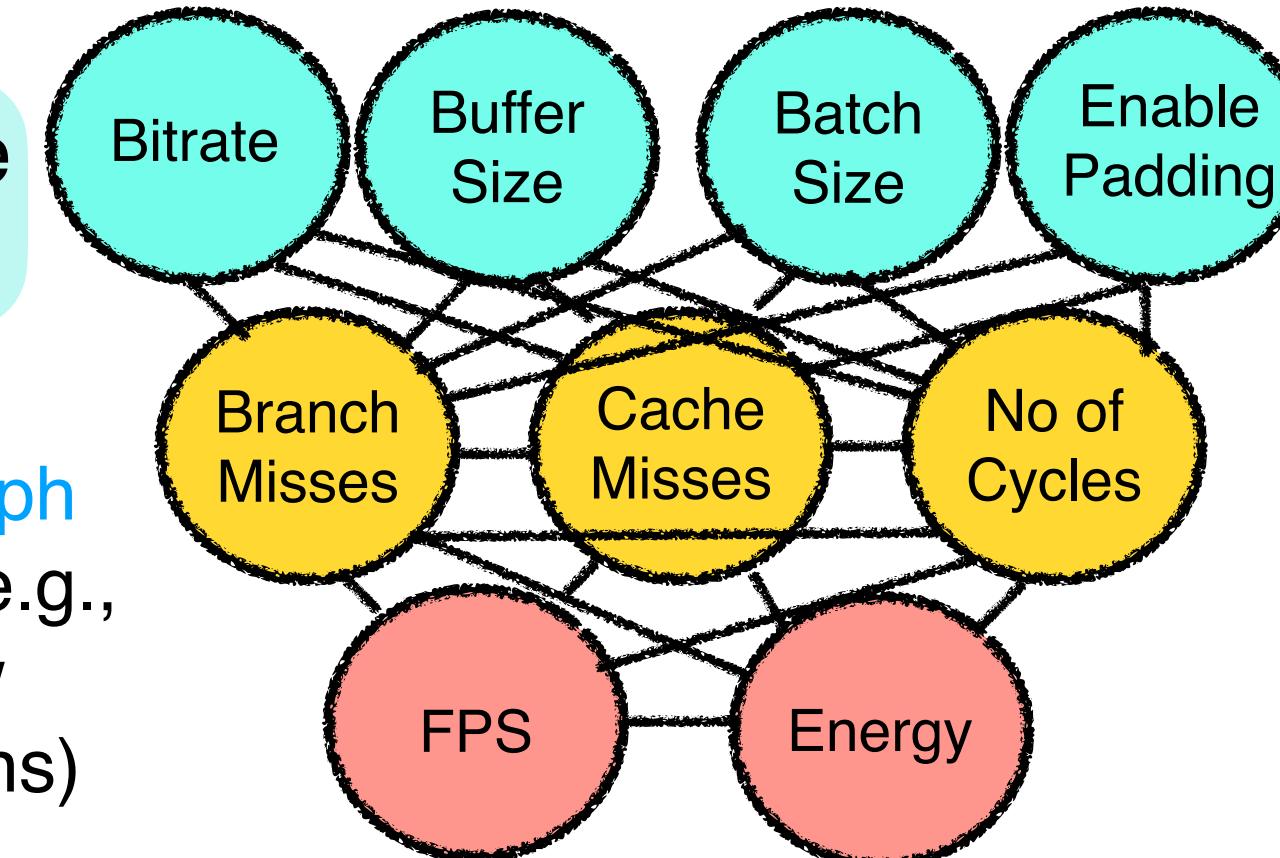
Learning Causal Performance Model

	Bitrate (bits/s)	Enable Padding	...	Cache Misses	...	Through put (fps)
c ₁	1k	1	...	42m	...	7
c ₂	2k	1	...	32m	...	22
...
c _n	5k	0	...	12m	...	25

1- Recovering the Skelton



fully connected graph given constraints (e.g., no connections btw configuration options)



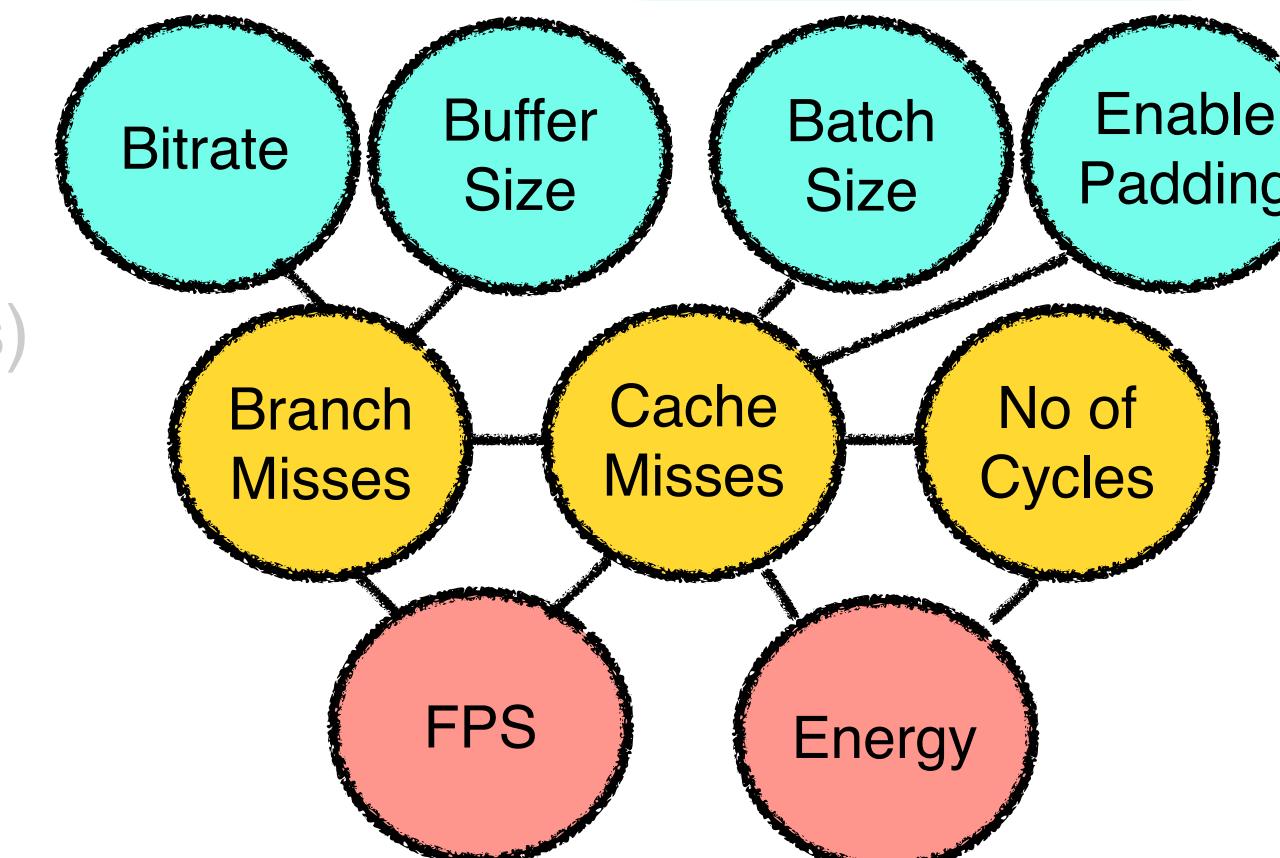
statistical independence

2- Pruning Causal Structure

orientation rules & measures (entropy) + structural constraints (colliders, v-structures)



3- Orienting Causal Relations



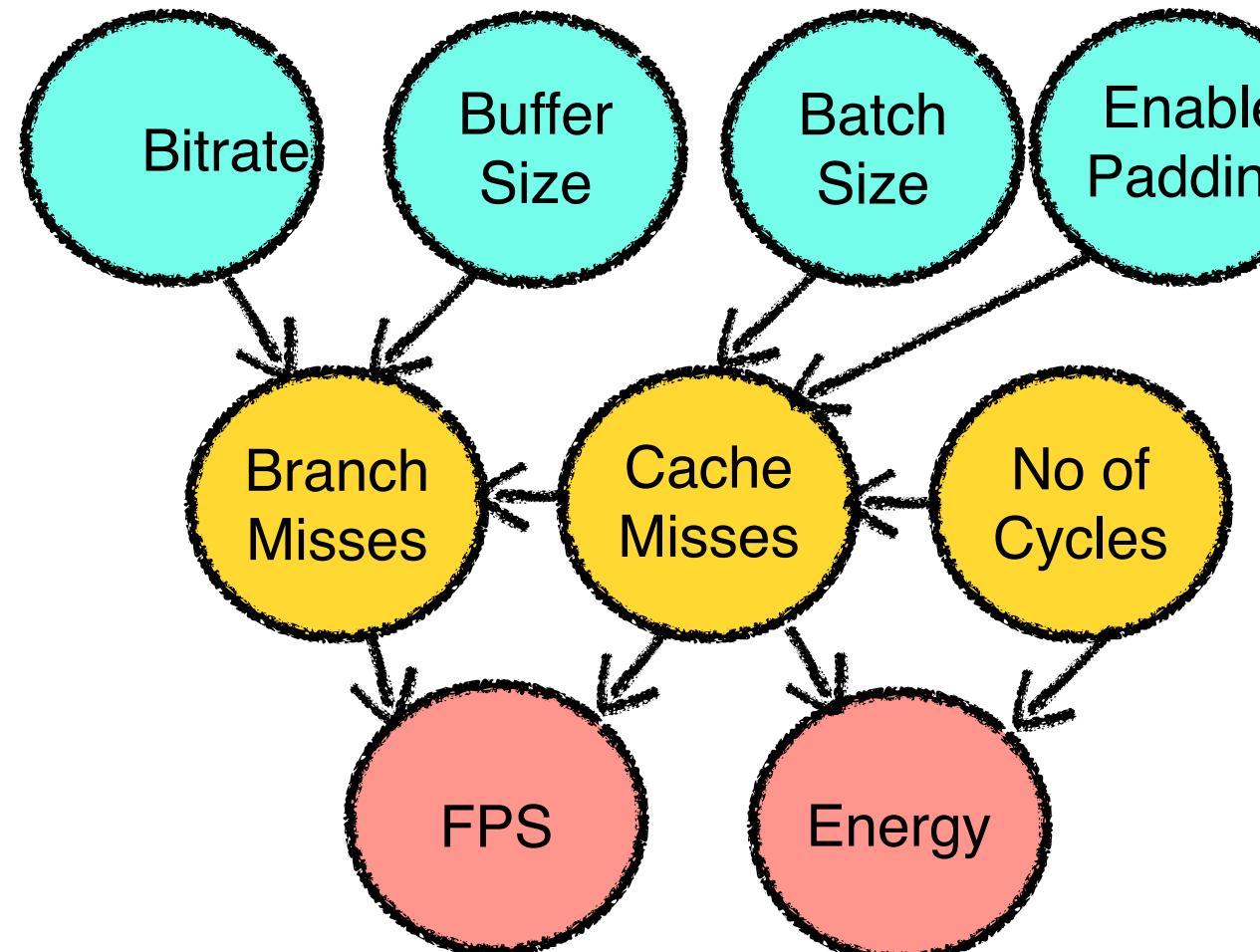
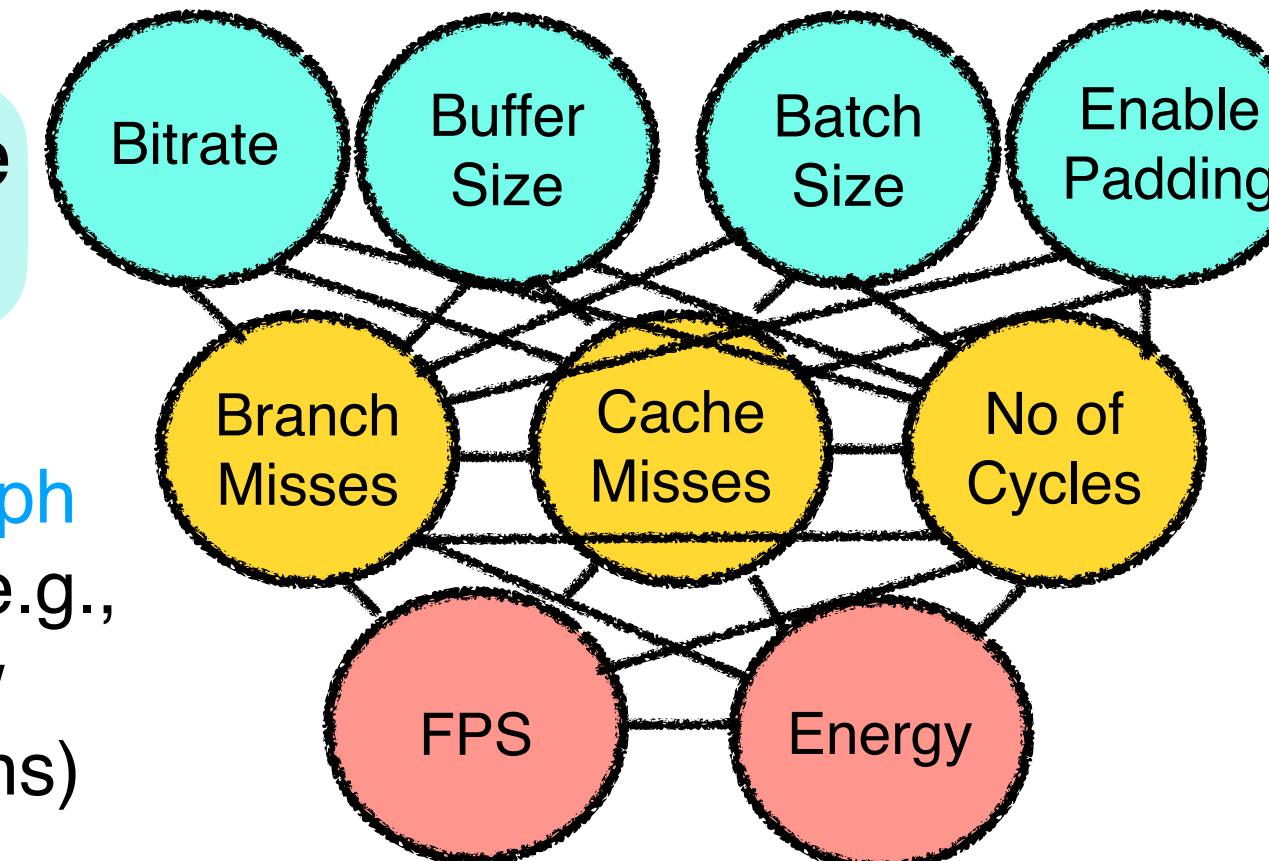
Learning Causal Performance Model

	Bitrate (bits/s)	Enable Padding	...	Cache Misses	...	Through put (fps)
c ₁	1k	1	...	42m	...	7
c ₂	2k	1	...	32m	...	22
...
c _n	5k	0	...	12m	...	25

1- Recovering the Skelton



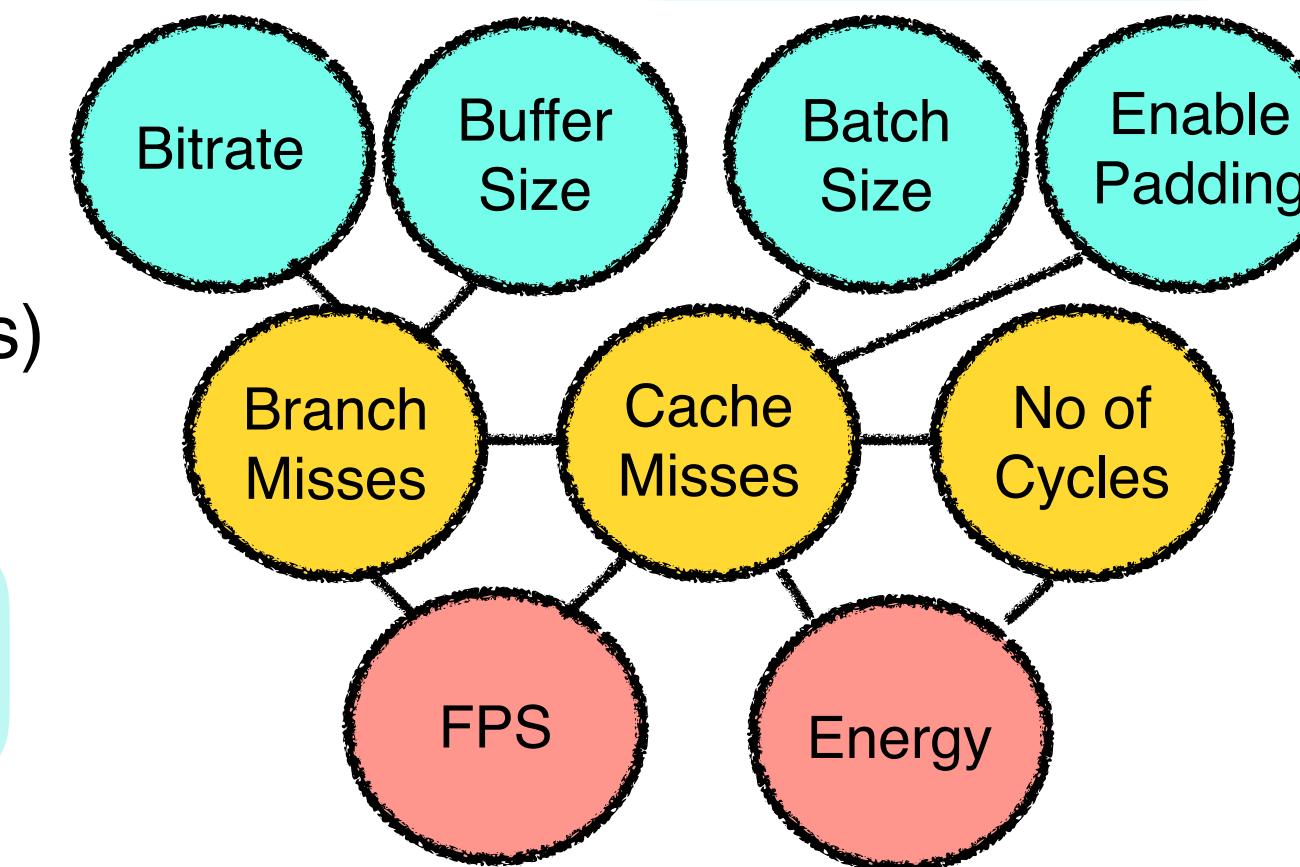
fully connected graph given constraints (e.g., no connections btw configuration options)



2- Pruning Causal Structure

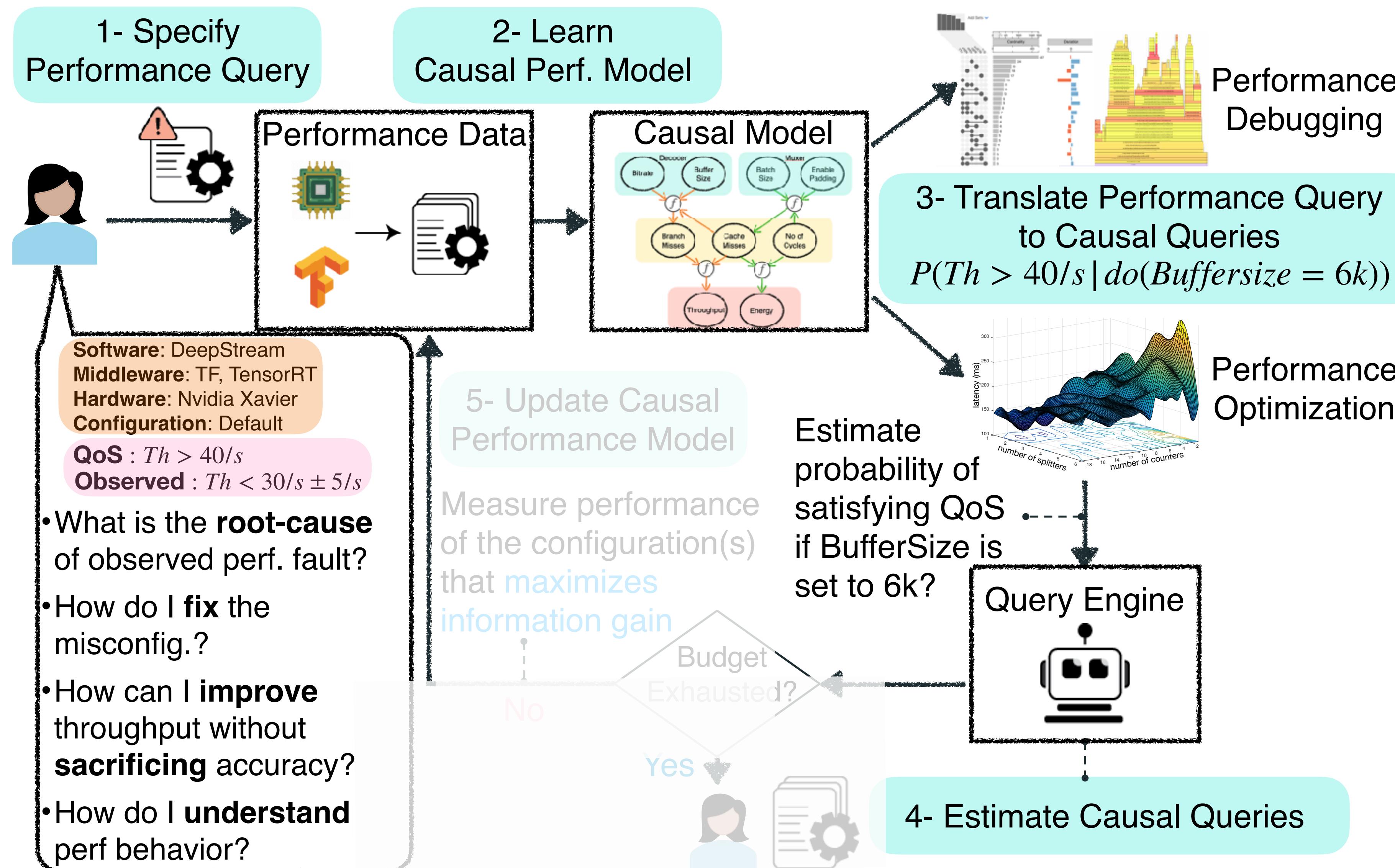
statistical independence tests

orientation rules & measures (entropy) + structural constraints (colliders, v-structures)



3- Orienting Causal Relations

UNICORN: Performance Reasoning through the Lens of Causality



Diagnosing and Fixing the Faults

- Counterfactual inference asks “what if” questions about changes to the misconfigurations



Example

“Given that my current swap memory is 2 Gb, and I have high latency. What is the probability of having low latency if swap memory was increased to 4 Gb?”



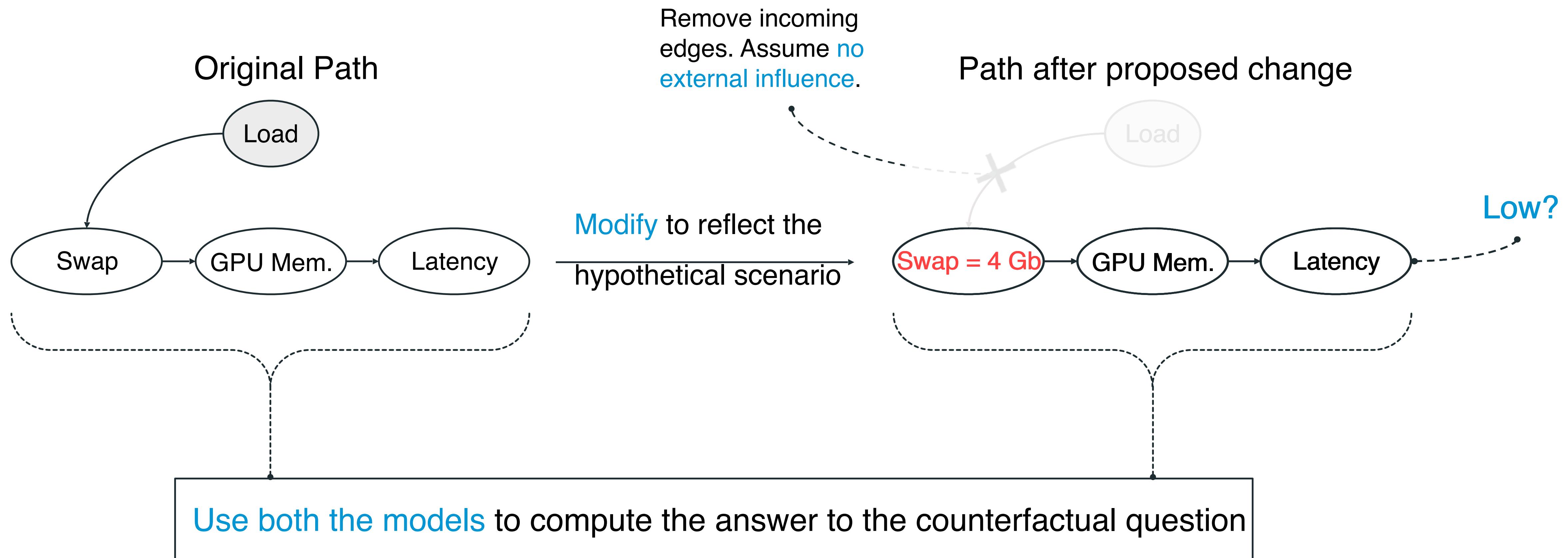
We are interested in the scenario where:

- We hypothetically have low latency;

Conditioned on the following events:

- We hypothetically set the new Swap memory to 4 Gb
- Swap Memory was initially set to 2 Gb
- We observed high latency when Swap was set to 2 Gb
- Everything else remains the same

Diagnosing and Fixing the Faults



Diagnosing and Fixing the Faults



$Potential = P(\hat{Latency} = \text{low} \mid \hat{Swap} = 4 \text{ Gb}, Swap = 2 \text{ Gb}, Latency_{swap=2Gb} = \text{high}, U)$

We expect a low latency

The Swap was initially 2 Gb

The Swap is now 4 Gb

The latency was high

Everything else stays the same

Diagnosing and Fixing the Faults

$$\text{Potential} = P\left(\text{outcome} = \text{good} \mid \text{change}, \text{outcome}_{\neg\text{change}} = \text{bad}, \neg\text{change}, U \right)$$

Probability that the **outcome is good** after a change, conditioned on the past

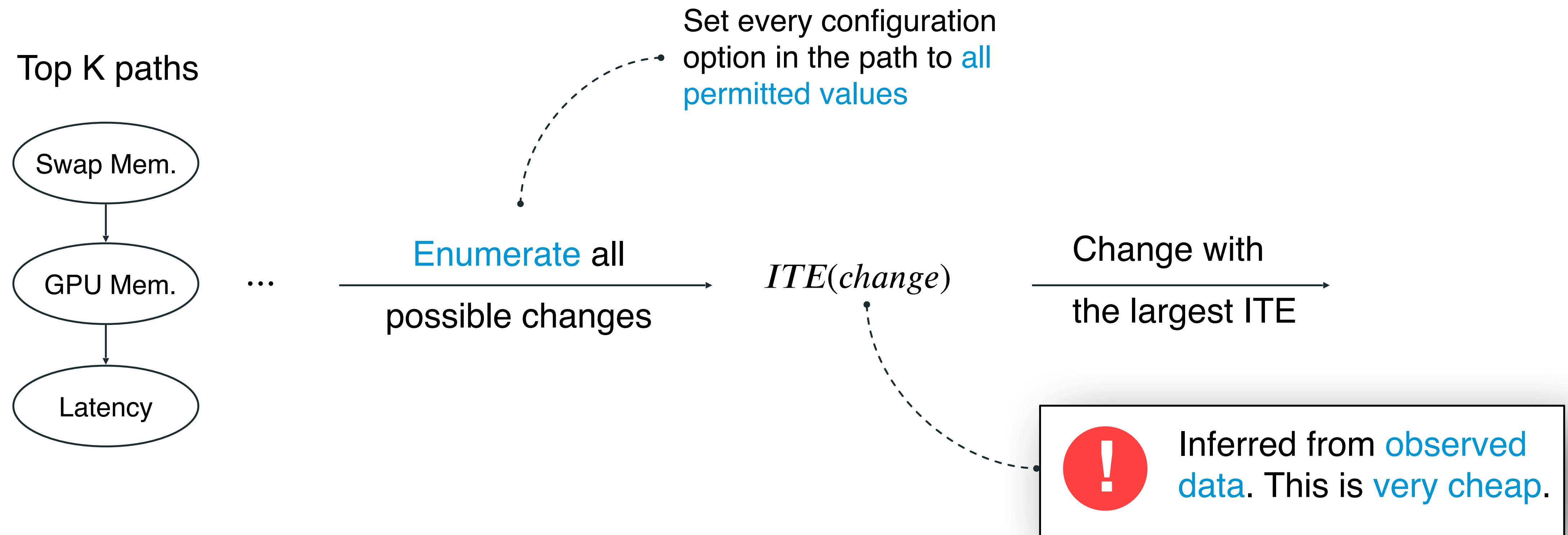
$$\text{Control} = P\left(\text{outcome} = \text{bad} \mid \neg\text{change}, U \right)$$

Probability that the **outcome was bad** before the change

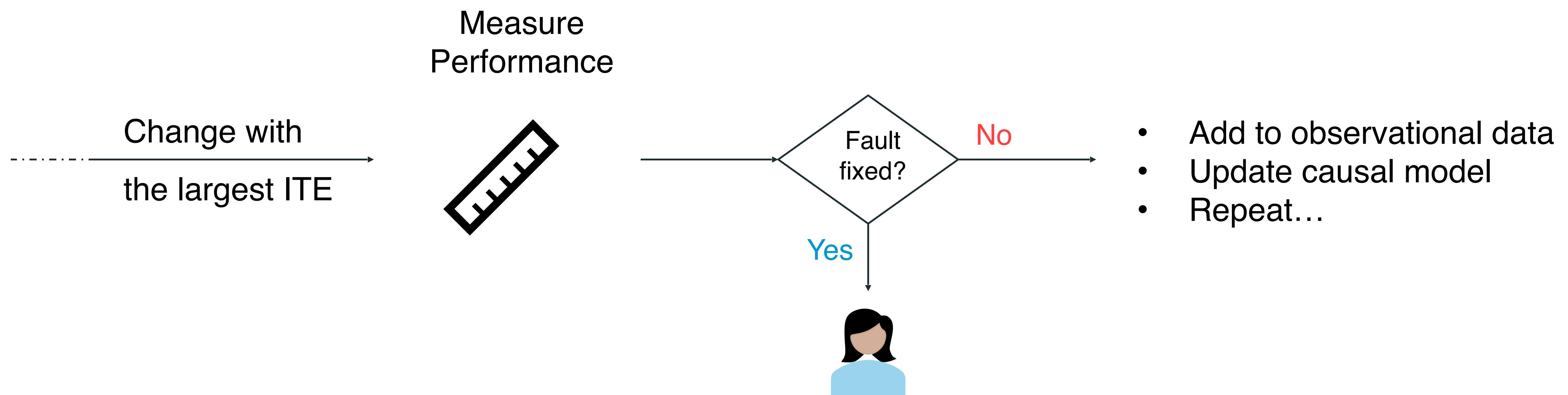
Individual Treatment Effect = Potential – Outcome

If this difference is **large**, then our change is **useful**

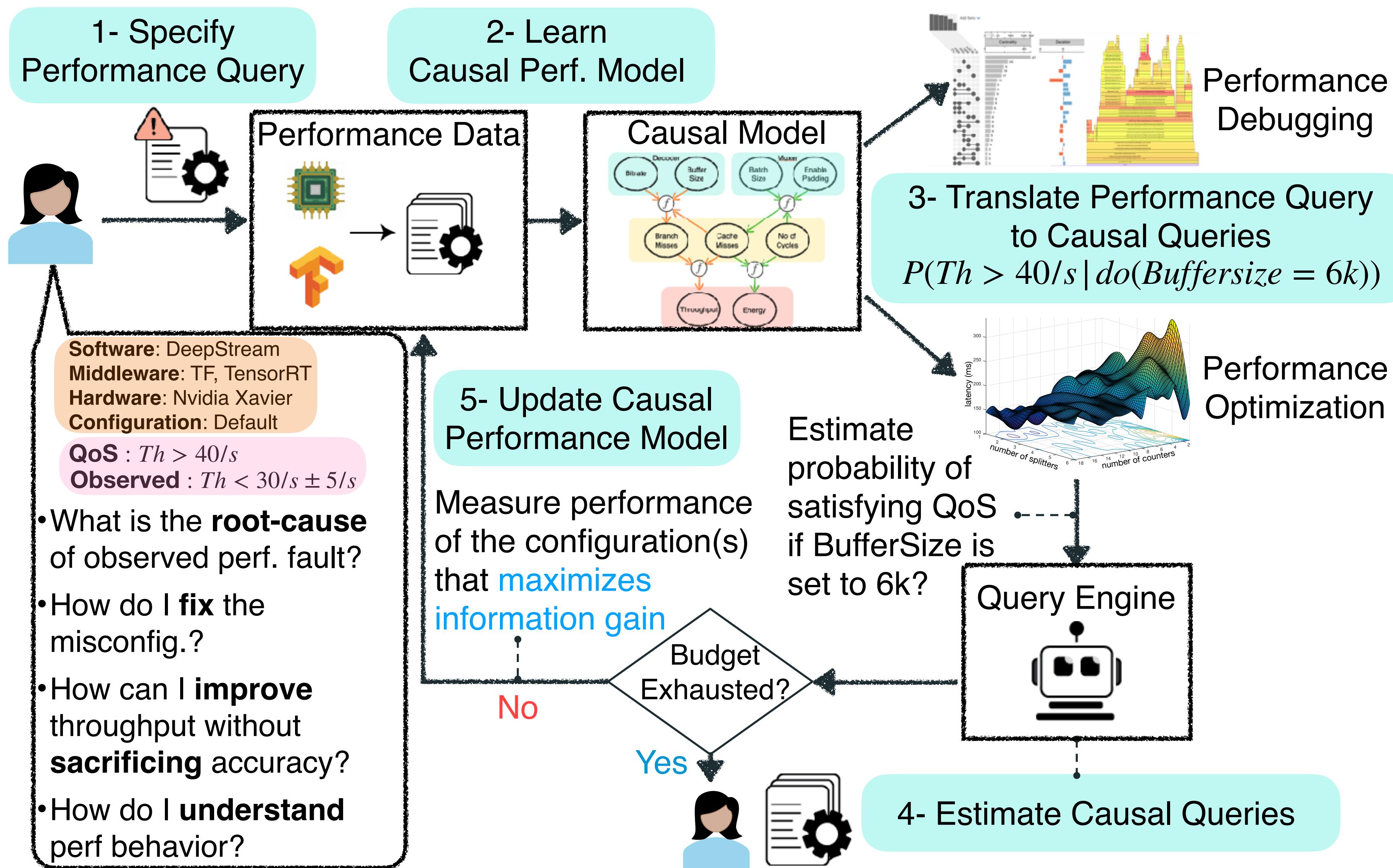
Diagnosing and Fixing the Faults



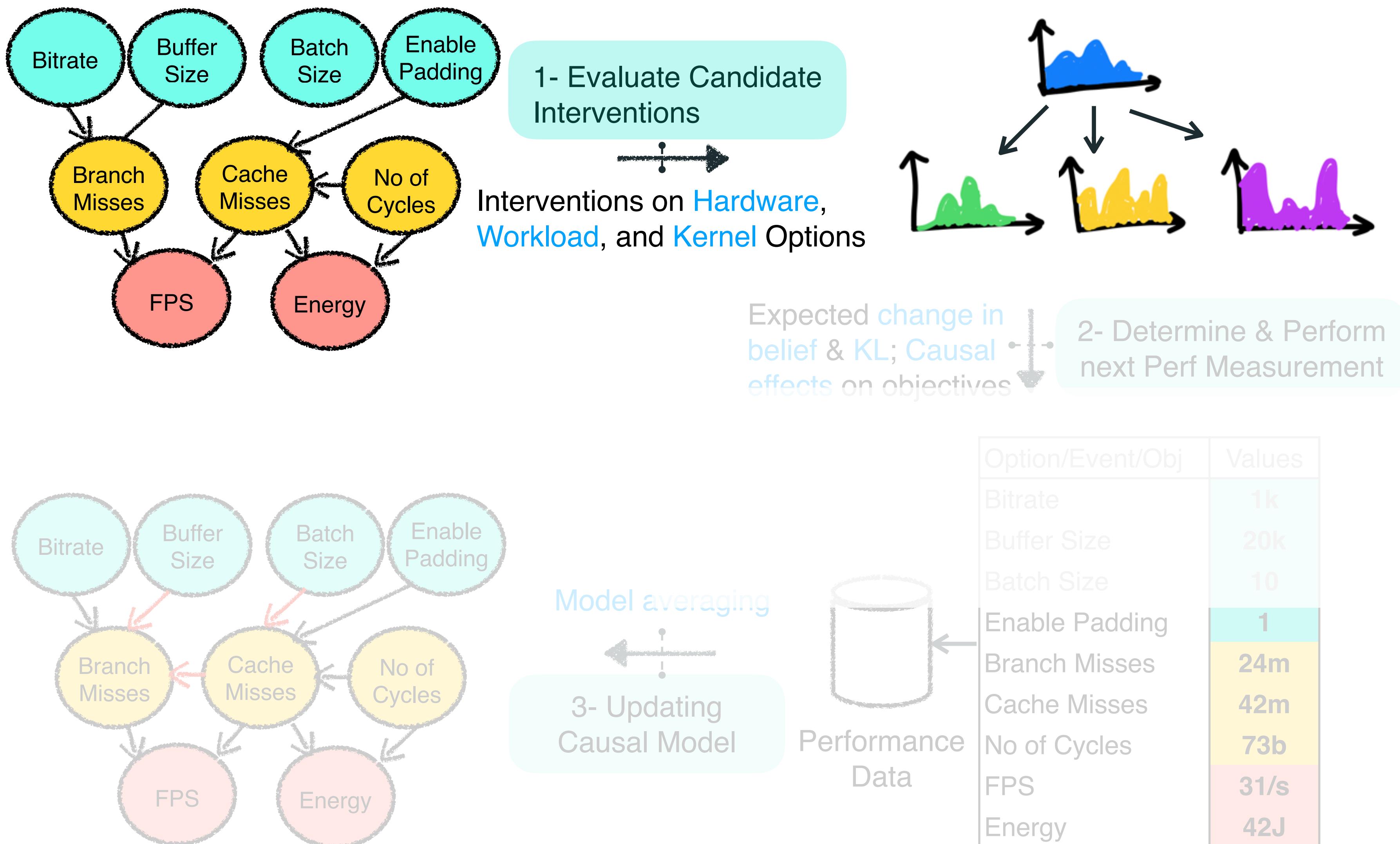
Diagnosing and Fixing the Faults



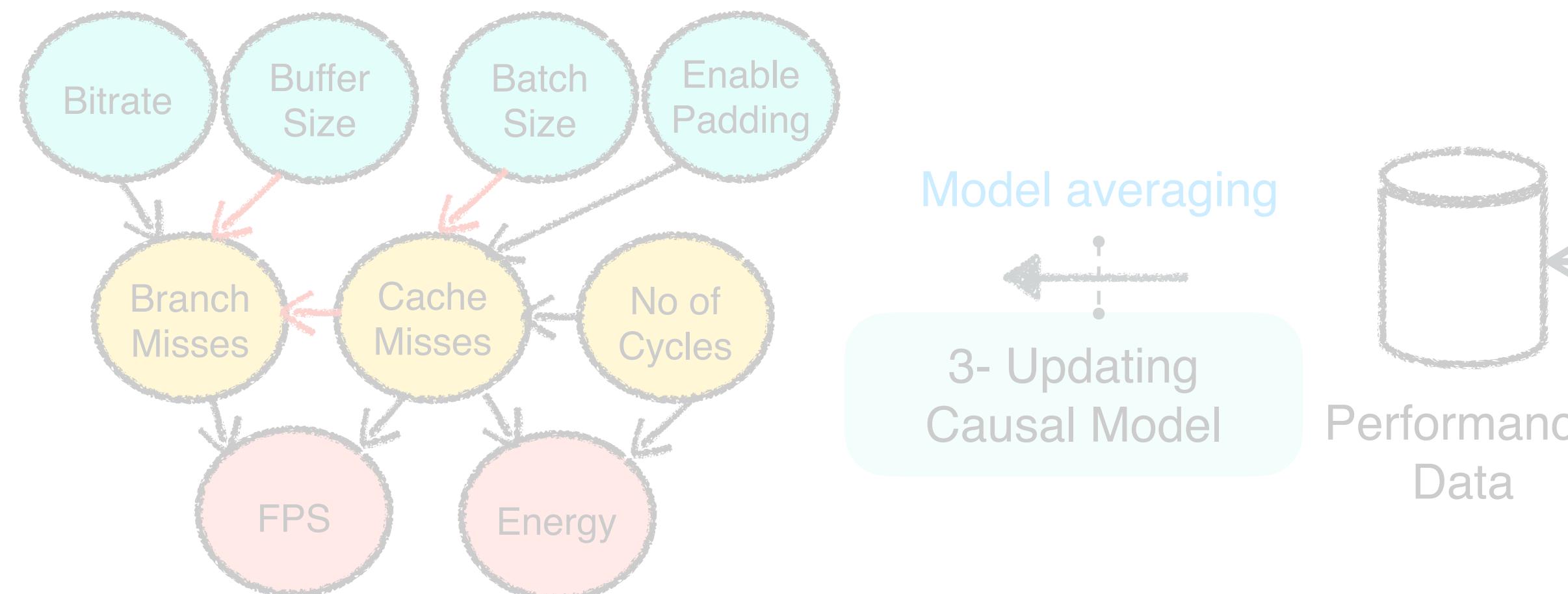
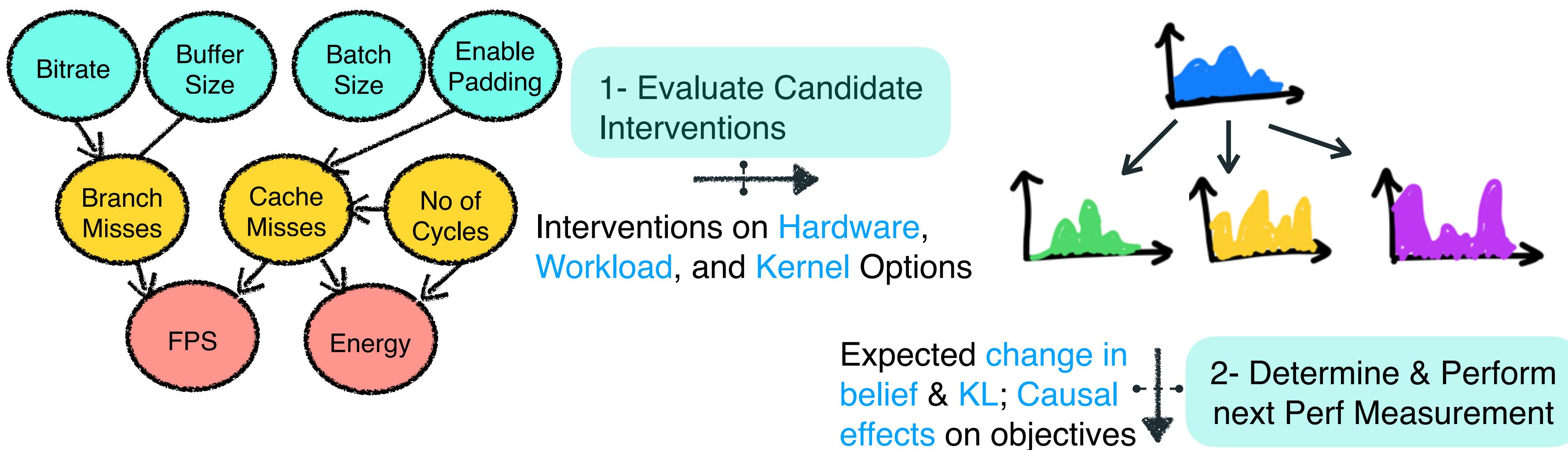
UNICORN: Our Causal AI for Systems Method



Active Learning for Updating Causal Performance Model

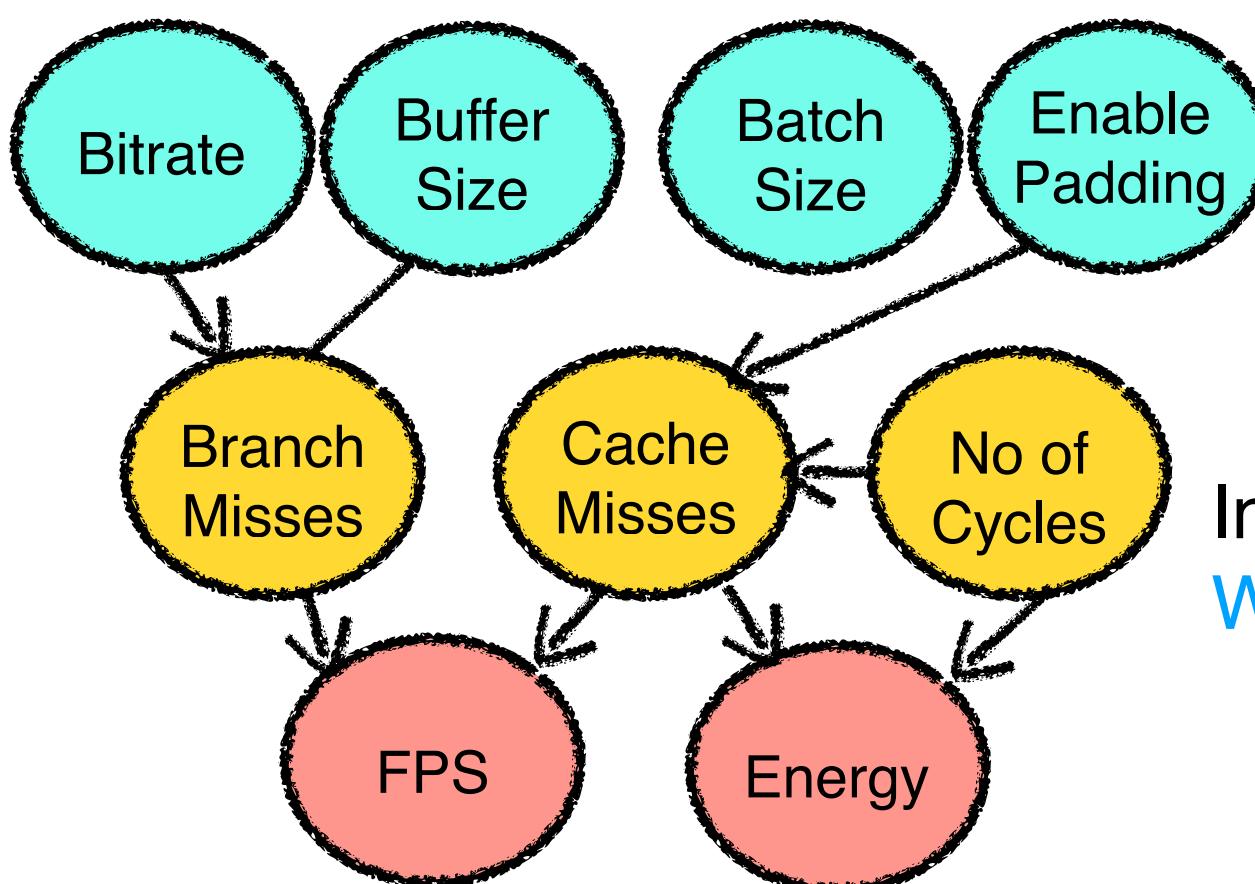


Active Learning for Updating Causal Performance Model



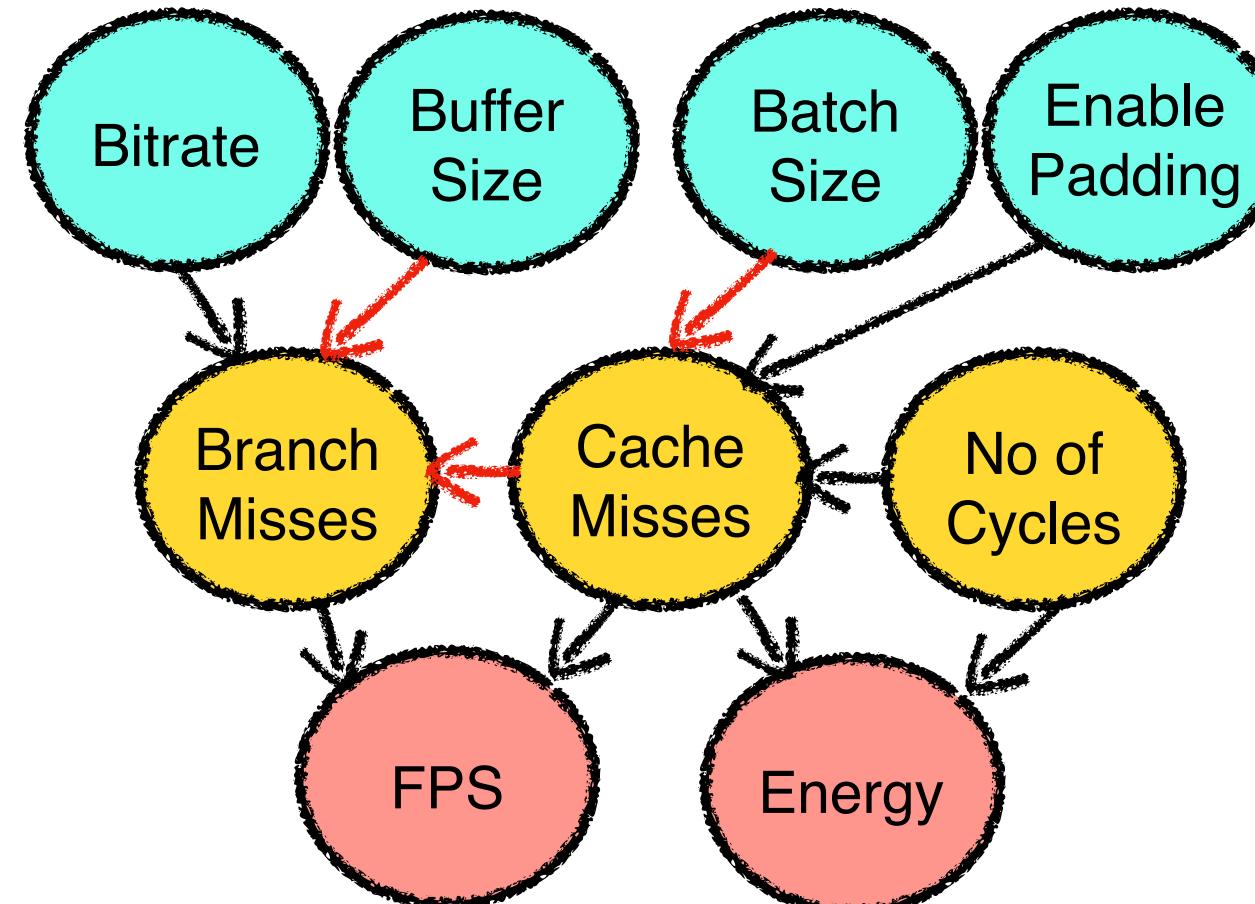
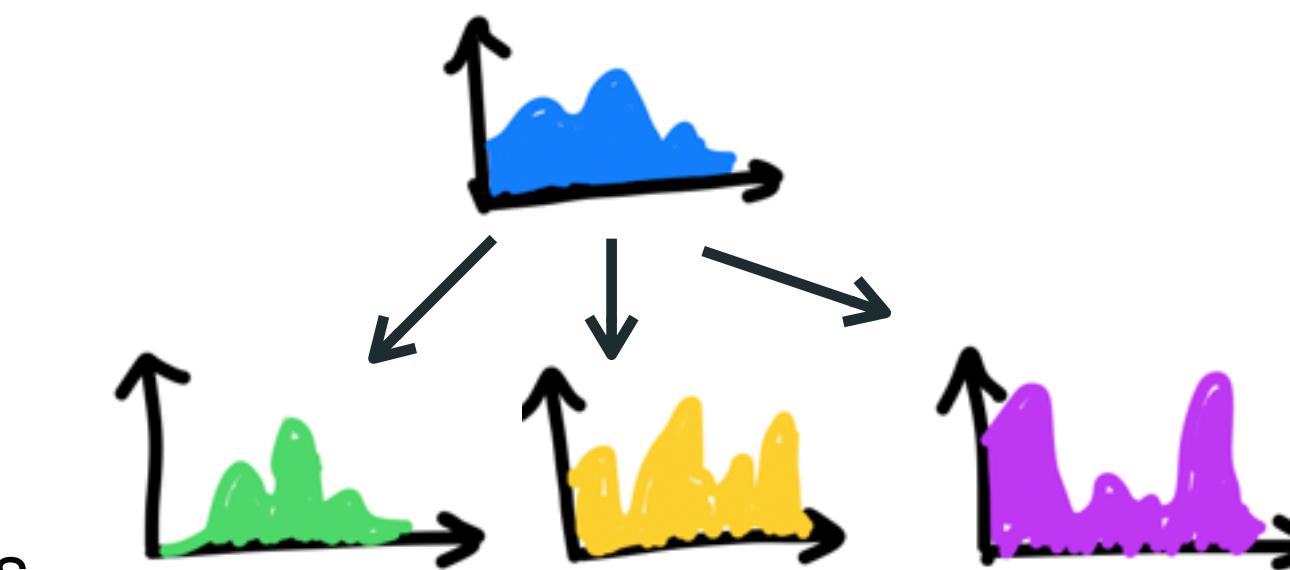
Option/Event/Obj	Values
Bitrate	1k
Buffer Size	20k
Batch Size	10
Enable Padding	1
Branch Misses	24m
Cache Misses	42m
No of Cycles	73b
FPS	31/s
Energy	42J

Active Learning for Updating Causal Performance Model



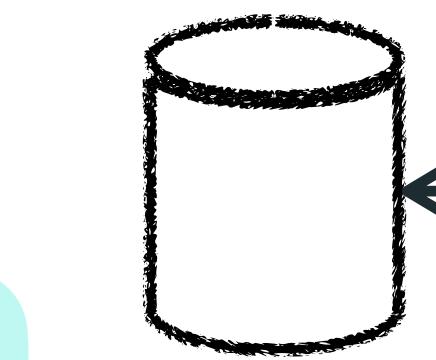
1- Evaluate Candidate Interventions

Interventions on **Hardware**,
Workload, and **Kernel Options**



Model averaging

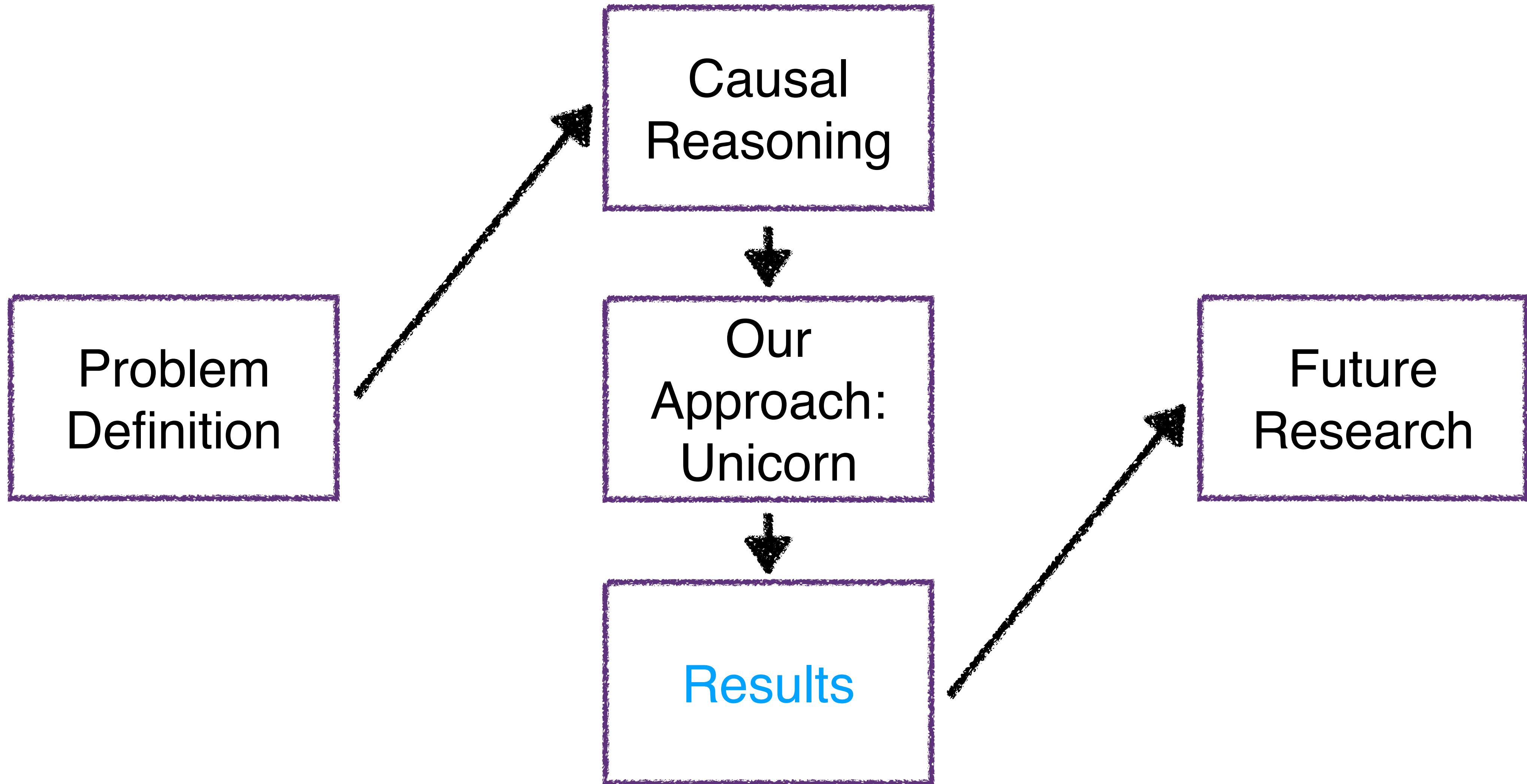
3- Updating
Causal Model



Performance
Data

Option/Event/Obj	Values
Bitrate	1k
Buffer Size	20k
Batch Size	10
Enable Padding	1
Branch Misses	24m
Cache Misses	42m
No of Cycles	73b
FPS	31/s
Energy	42J

Outline



Results: Case Study



CUDA performance issue on tx2

Home > Autonomous Machines > Jetson & Embedded Systems > Jetson TX2



william_wu

When we are trying to [transplant our CUDA source code from TX1 to TX2](#), it behaved strange.

We noticed that [TX2 has twice computing-ability as TX1 in GPU](#), as expectation, [we think TX2 will 30% - 40% faster than TX1 at least](#).

Unfortunately, most of our code base spent twice the time as TX1, in other words, [TX2 only has 1/2 speed as TX1, mostly](#). We believe that TX2's CUDA API runs much slower than TX1 in many cases.

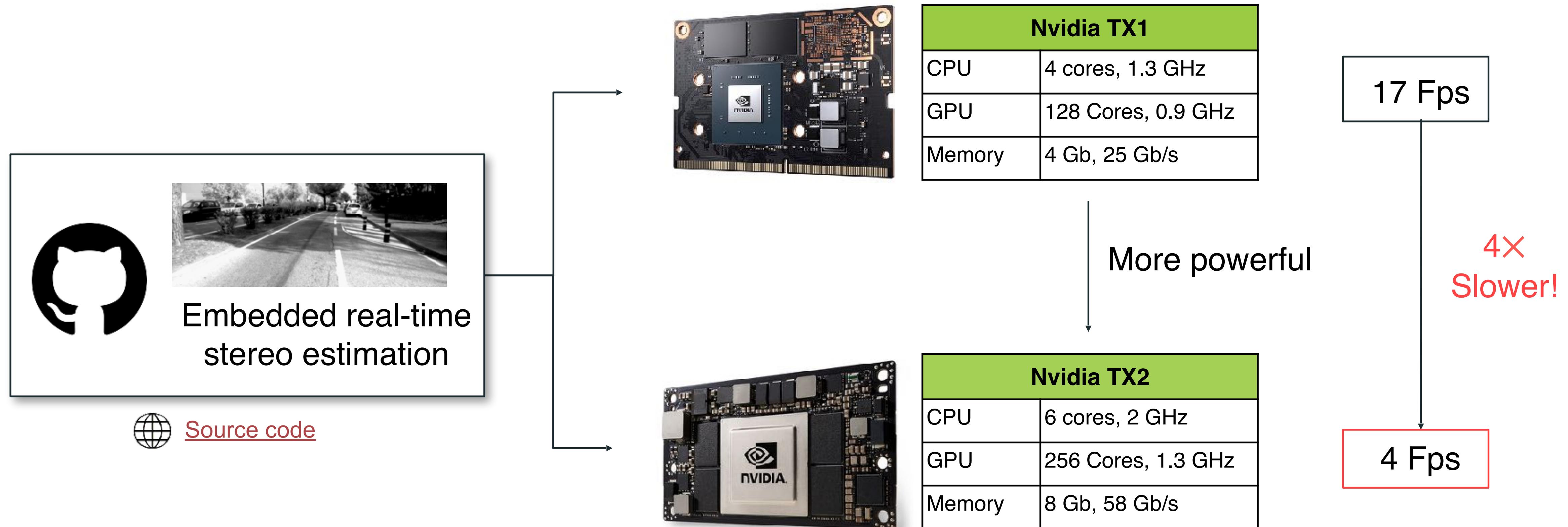
The user is transferring the code from [one hardware to another](#)

Jun '17

The [target hardware is faster](#) than the [the source hardware](#). User [expects the code to run at least 30-40% faster](#).

The [code ran 2x slower](#) on the more powerful hardware

Results: Case Study



Results: Case Study

Configuration	UNICO RN	Decision Tree	Forum
CPU Cores	✓	✓	✓
CPU Freq.	✓	✓	✓
EMC Freq.	✓	✓	✓
GPU Freq.	✓	✓	✓
Sched. Policy		✓	
Sched. Runtime		✓	
Sched. Child Proc		✓	
Dirty Bg. Ratio		✓	
Drop Caches		✓	
CUDA_STATIC_RT	✓	✓	✓
Swap Memory		✓	

	UNICORN	Decision Tree	Forum
Throughput (on TX2)	26 FPS	20 FPS	23 FPS
Throughput Gain (over TX1)	53 %	21 %	39 %
Time to resolve	24 min.	3½ Hrs.	2 days

- The user expected 30-40% gain

Results

- X Finds the root-causes accurately
- X No unnecessary changes
- X Better improvements than forum's recommendation
- X Much faster

Evaluation: Experimental Setup

Hardware

Nvidia TX1	
CPU	4 cores, 1.3 GHz
GPU	128 Cores, 0.9 GHz
Memory	4 Gb, 25 GB/s

Nvidia TX2	
CPU	6 cores, 2 GHz
GPU	256 Cores, 1.3 GHz
Memory	8 Gb, 58 GB/s

Nvidia Xavier	
CPU	8 cores, 2.26 GHz
GPU	512 cores, 1.3 GHz
Memory	32 Gb, 137 GB/s

Systems

Xception

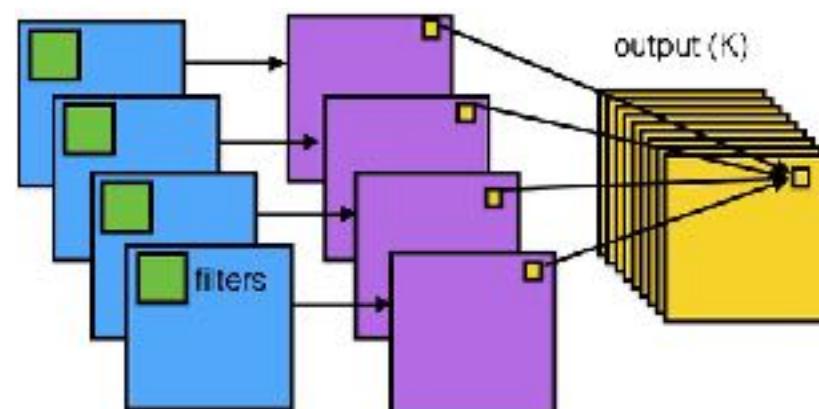
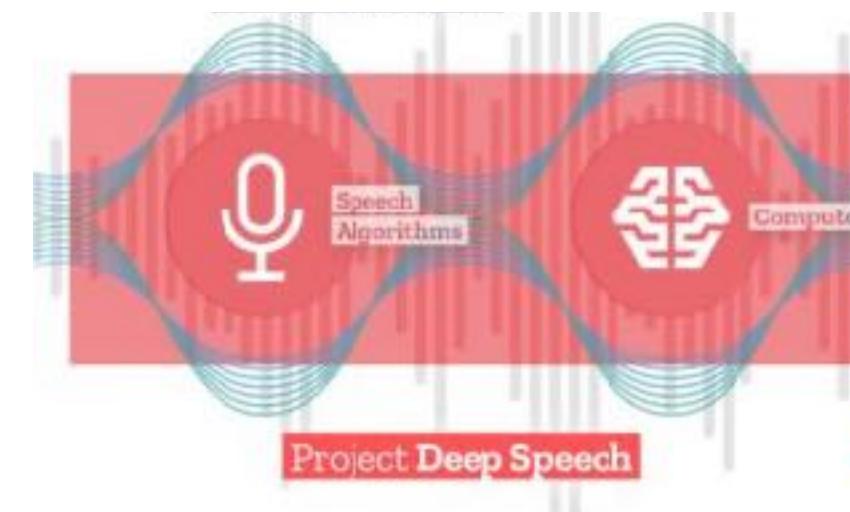


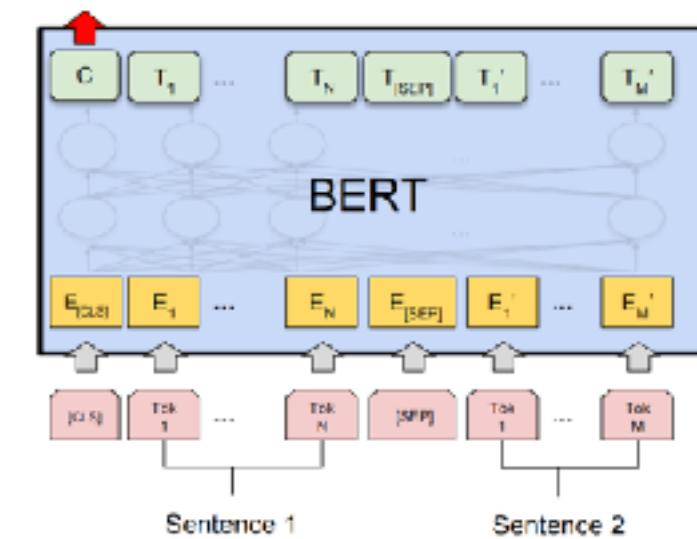
Image recognition
(50,000 test images)

DeepSpeech



Voice recognition
(5 sec. audio clip)

BERT



Sentiment Analysis
(10000 IMDb reviews)

x264



Video Encoder
(11 Mb, 1080p video)

Configuration Space

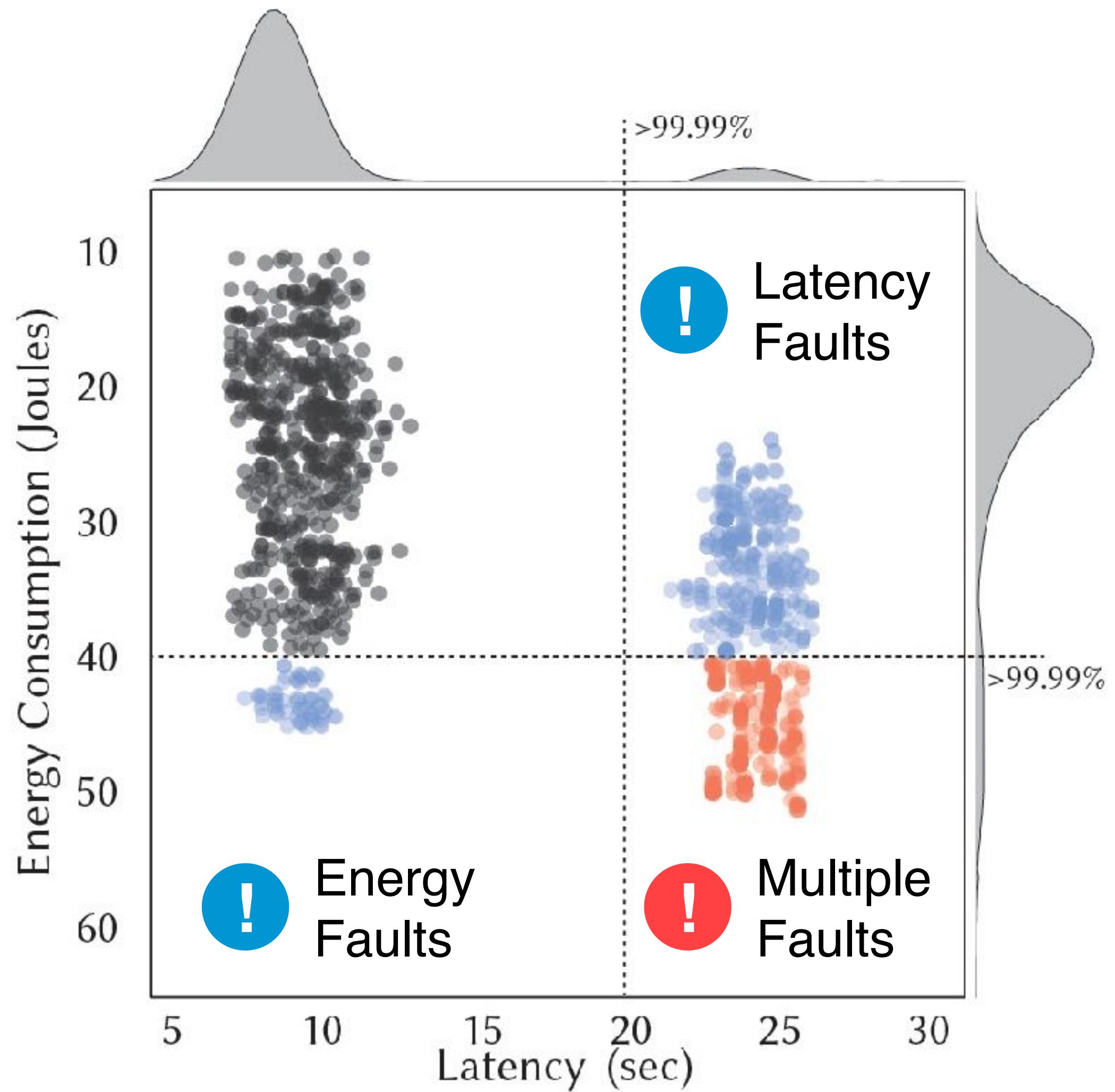
X 30 Configurations

- 10 software
- 10 OS/Kernel
- 10 hardware

X 17 System Events

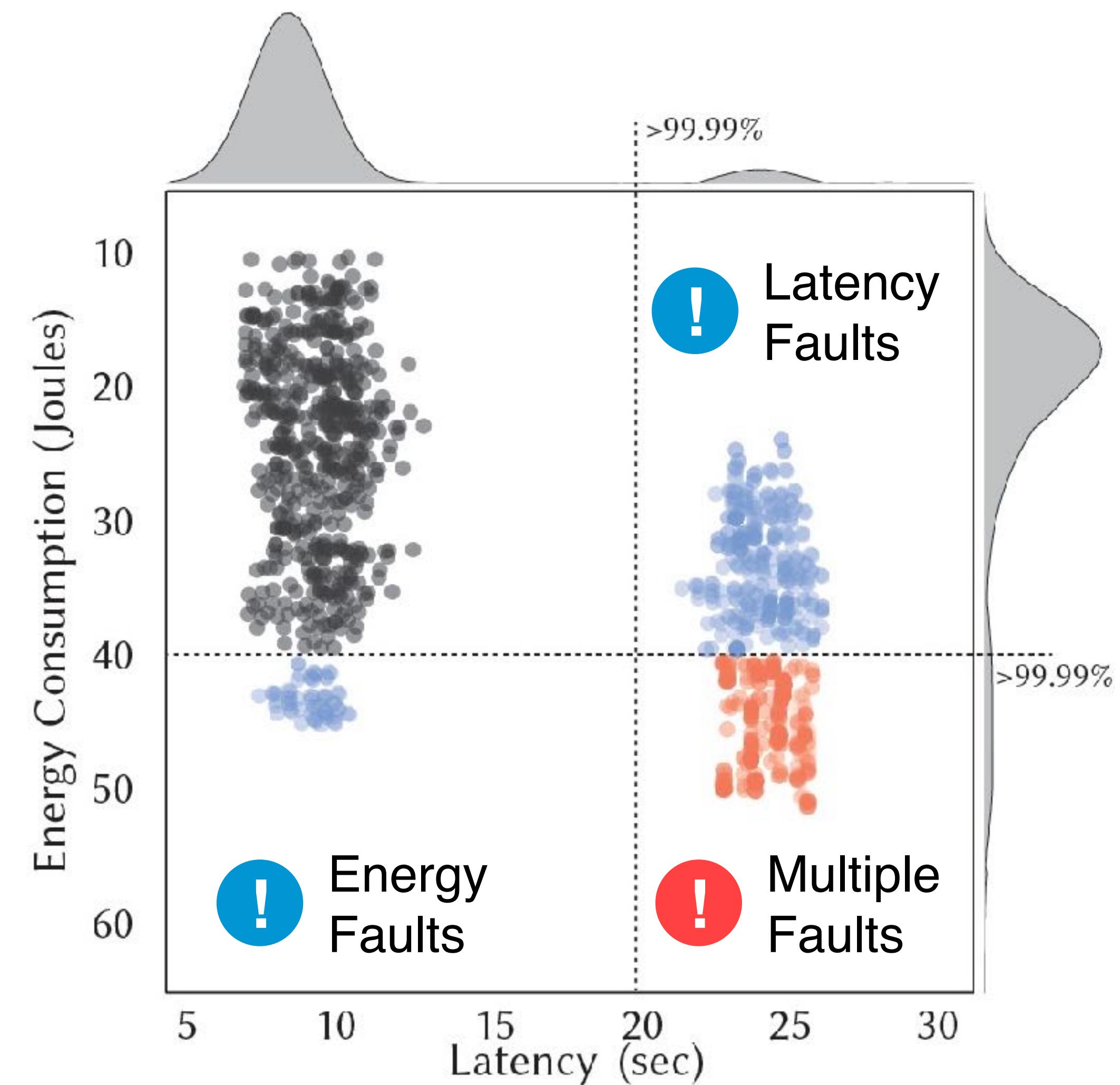
Evaluation: Data Collection

- For each software/hardware combination create a **benchmark dataset**
 - Exhaustively set each of configuration option to **all permitted values**.
 - For continuous options (e.g., GPU memory Mem.), sample **10 equally spaced values** between [min, max]
- Measure the latency, energy consumption, and heat dissipation
 - Repeat **5x** and **average**



Evaluation: Ground Truth

- For each performance fault:
 - Manually investigate the root-cause
 - “Fix” the misconfigurations
- A “fix” implies the configuration **no longer has tail performance**
 - User defined benchmark (i.e., 10th percentile)
 - Or some QoS/SLA benchmark
- **Record** the configurations that were changed



Experimental Setup: Baselines

Debugging

BugDoc: A System for Debugging Computational Pipelines

Raoni Lourenço
New York University
raoni@nyu.edu

Juliana Freire
New York University
juliana.freire@nyu.edu

Dennis Shasha
New York University
shasha@courant.nyu.edu

Statistical Debugging for Real-World Performance Problems

Linhai Song Shan Lu *
University of Wisconsin–Madison
{songlh, shanlu}@cs.wisc.edu

Optimization

Sequential Model-Based Optimization for General Algorithm Configuration (extended version)

Frank Hutter, Holger H. Hoos and Kevin Leyton-Brown

University of British Columbia, 2366 Main Mall, Vancouver BC, V6T 1Z4, Canada
{hutter, hoos, kevinlb}@cs.ubc.ca

EnCore: Exploiting System Environment and Correlation Information for Misconfiguration Detection

Jiaqi Zhang[†], Lakshminarayanan Renganarayana[§], Xiaolan Zhang[§], Niyu Ge[§],
Vasanth Bala[§], Tianyin Xu[†], Yuanyuan Zhou[†]

[†]University of California San Diego
{jiz013, tixu, yyzhou}@cs.ucsd.edu

[§]IBM Watson Research Center
{lrengan, cxzhang, niyuge, vbala}@us.ibm.com

Iterative Delta Debugging

Cyrille Artho

Research Center for Information Security (RCIS), AIST, Tokyo, Japan

Sequential Model-Based Optimization for General Algorithm Configuration (extended version)

Frank Hutter, Holger H. Hoos and Kevin Leyton-Brown

University of British Columbia, 2366 Main Mall, Vancouver BC, V6T 1Z4, Canada
{hutter, hoos, kevinlb}@cs.ubc.ca

Predictive Entropy Search for Multi-objective Bayesian Optimization

Daniel Hernández-Lobato

DANIEL.HERNANDEZ@UAM.ES

Universidad Autónoma de Madrid, Francisco Tomás y Valiente 11, 28049, Madrid, Spain.

José Miguel Hernández-Lobato

JMHL@SEAS.HARVARD.EDU

Harvard University, 33 Oxford street, Cambridge, MA 02138, USA.

Amar Shah

AS793@CAM.AC.UK

Cambridge University, Trumpington Street, Cambridge CB2 1PZ, United Kingdom.

Ryan P. Adams

RPA@SEAS.HARVARD.EDU

Harvard University and Twitter, 33 Oxford street Cambridge, MA 02138, USA.

Results: Efficiency (Debugging; Single objective)

			Accuracy				Precision				Recall				Gain				Time [†]					
			UNICORN		CBI	DD	ENCORE	BugDoc	UNICORN		CBI	DD	ENCORE	BugDoc	UNICORN		CBI	DD	ENCORE	BugDoc	UNICORN			
TX2	Latency	DEEPSTREAM	87	61	62	65	81	83	66	59	60	71	80	61	65	60	70	88	66	67	68	79	0.8	4
		XCEPTION	86	53	42	62	65	86	67	61	63	67	83	64	68	69	62	82	48	42	57	59	0.6	4
		BERT	81	56	59	60	57	76	57	55	61	73	71	74	68	67	65	74	54	59	62	58	0.4	4
		DEEPSPEECH	81	61	59	60	72	76	58	69	61	71	81	73	61	63	69	76	59	53	55	66	0.7	4
		x264	83	59	63	62	62	82	69	58	65	66	78	64	67	63	72	85	69	72	68	71	1.4	4
XAVIER	Energy	DEEPSTREAM	91	81	79	77	87	81	61	62	64	73	85	63	61	62	75	86	68	62	61	78	0.7	4
		XCEPTION	84	66	63	63	81	78	56	58	66	65	80	69	55	63	68	83	59	50	51	62	0.4	4
		BERT	66	59	53	63	72	70	62	64	64	65	79	61	54	63	66	62	49	36	49	53	0.5	4
		DEEPSPEECH	73	68	63	72	71	75	55	59	54	68	78	53	52	59	71	78	64	48	65	63	1.2	4
		x264	77	71	70	74	74	83	63	53	61	66	78	67	53	54	72	87	73	71	76	76	0.3	4

Find root causes
more accurately than
ML-based methods

Better gain
Up to 20x
faster

Results: Efficiency (Debugging; Multi-objective)

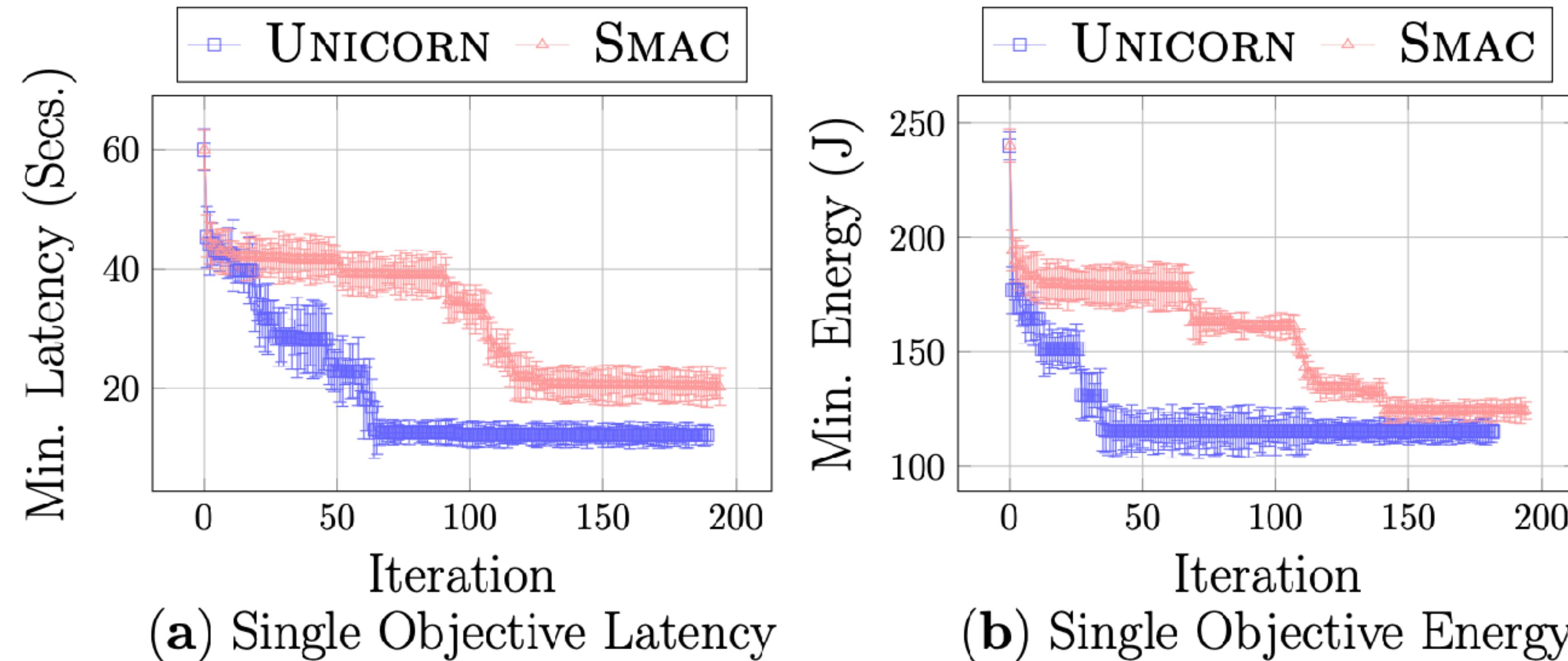
	Energy + Latency	Accuracy				Precision				Recall				Gain (Latency)				Gain (Energy)				Time [†]	
		UNICORN	CBI	ENCORE	BugDoc	UNICORN	CBI	ENCORE	BugDoc	UNICORN	CBI	ENCORE	BugDoc	UNICORN	CBI	ENCORE	BugDoc	UNICORN	CBI	ENCORE	BugDoc	UNICORN	Others
Energy + Latency	XCEPTION	89	76	81	79	77	53	54	62	81	59	59	62	84	53	61	65	75	38	46	44	0.9	4
	BERT	71	72	73	71	77	42	56	63	79	59	62	65	84	53	59	61	67	41	27	48	0.5	4
	DEEPSPEECH	86	69	71	72	80	44	53	62	81	51	59	64	88	55	55	62	77	43	43	41	1.1	4
	x264	85	73	83	81	83	50	54	67	80	63	62	61	75	62	64	66	76	64	66	64	1	4

[†] Wallclock time in hours

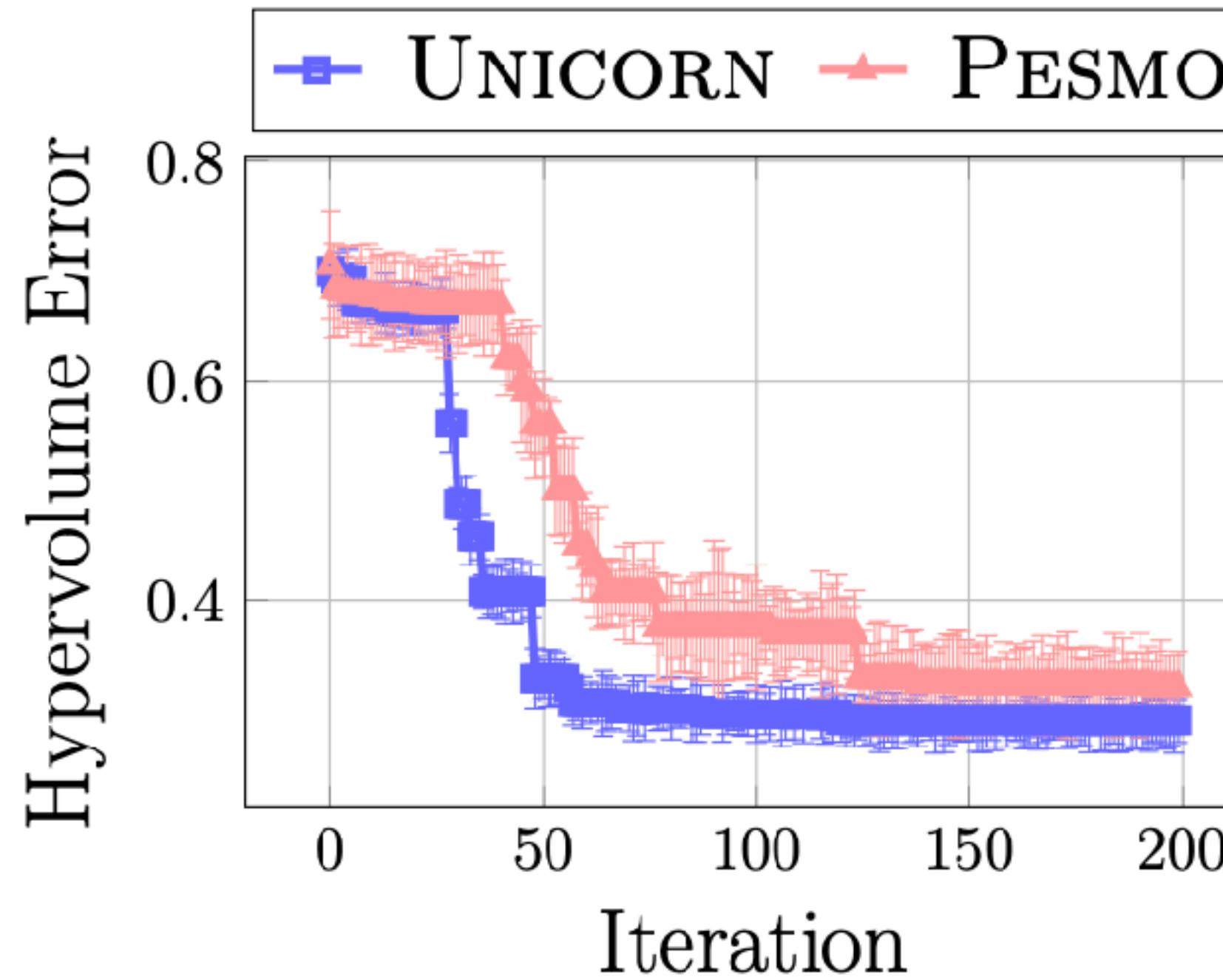
Multiple Faults
in Latency &
Energy usage

Better gain across both objectives

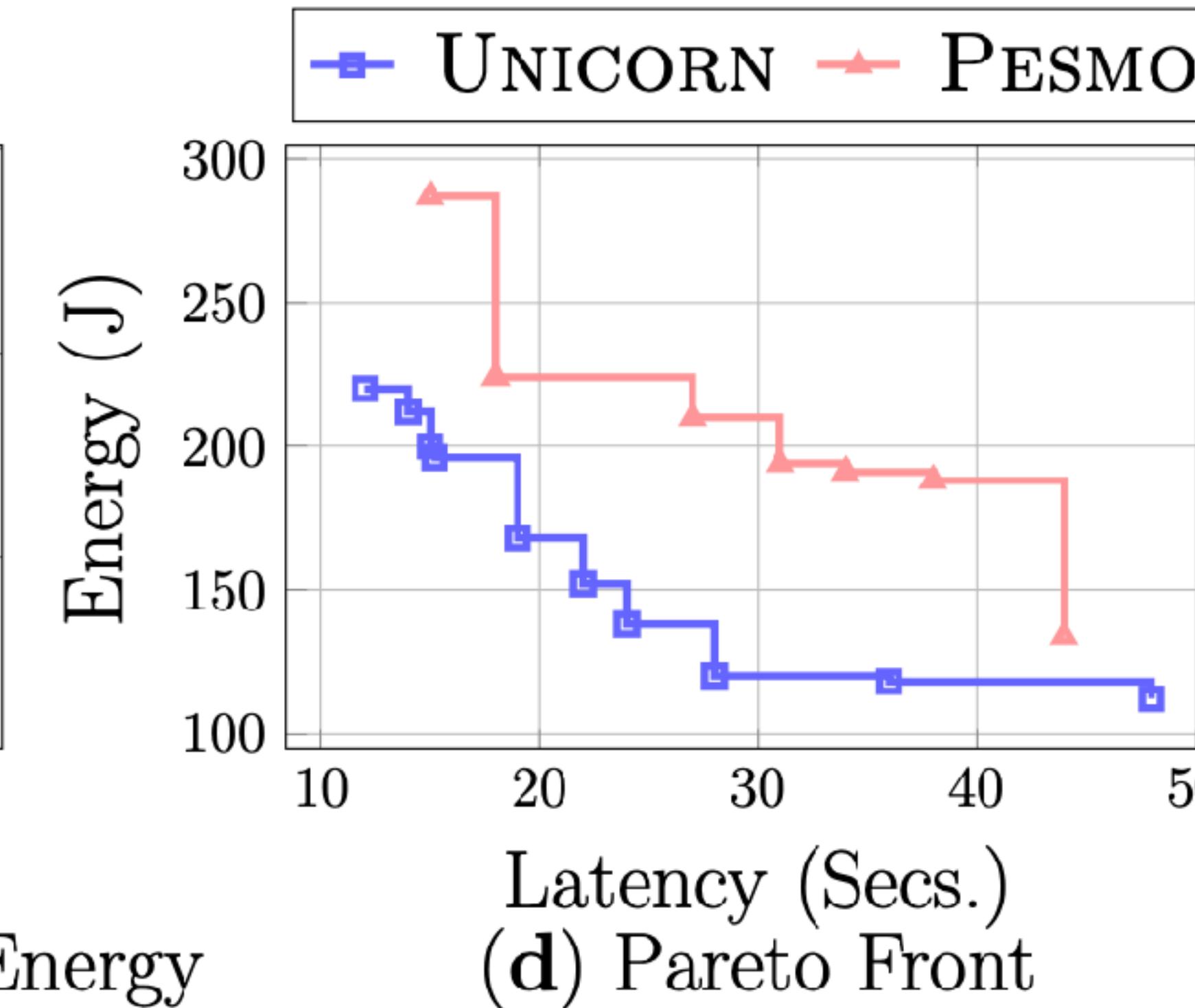
Results: Efficiency (Optimization; Single objective)



Results: Efficiency (Optimization; Multi-objective)

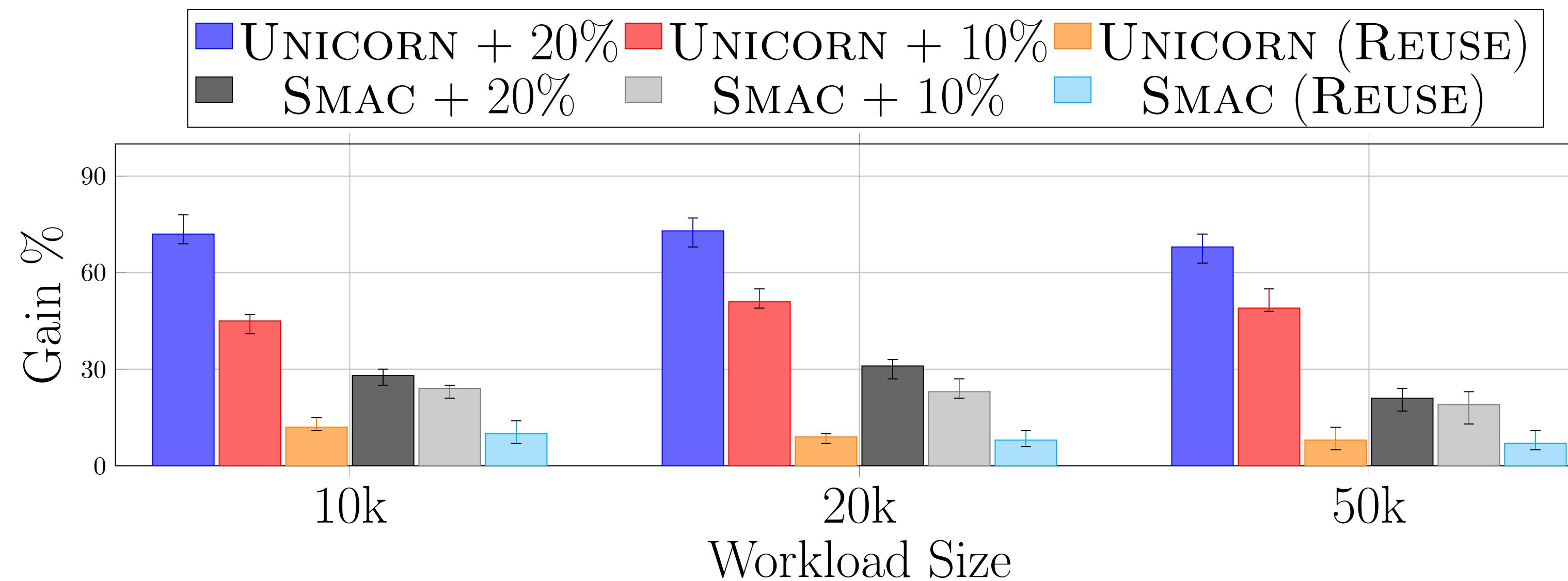


(c) Multi-Objective Latency and Energy



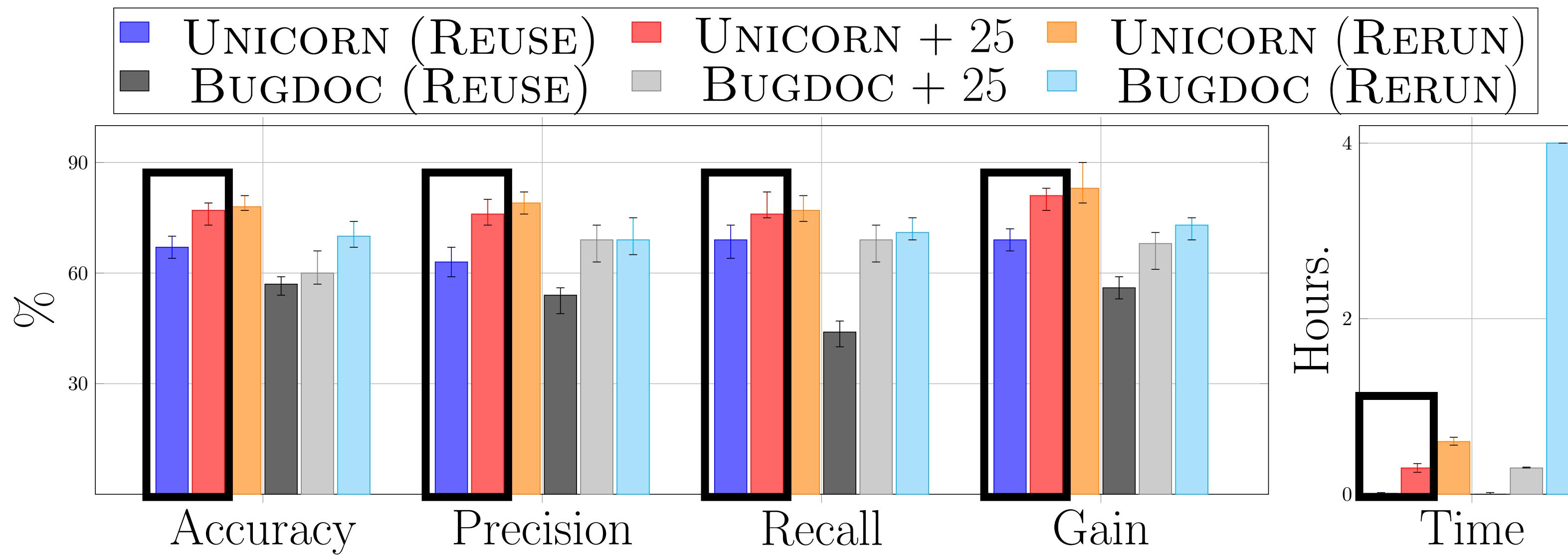
(d) Pareto Front

Results: Transferability



UNICORN finds configuration with higher gain when workload changes.

Results: Transferability



UNICORN quickly fixes the bug and achieves higher gain, accuracy, precision and recall when hardware changes

Results: Scalability

System	Configs	Events	Paths	Queries	Degree	Gain (%)	Time/Fault (in sec.)		
							Discovery	Query Eval	Total
SQLITE	34	19	32	191	3.6	93	9	14	291
	242	19	111	2234	1.9	94	57	129	1345
	242	288	441	22372	1.6	92	111	854	5312
DEEPSTREAM	53	19	43	497	3.1	86	16	32	1509
	53	288	219	5008	2.3	85	97	168	3113

Discovery time, query evaluation time and total time **do not increase exponentially** as the number of configuration options and systems events are increased

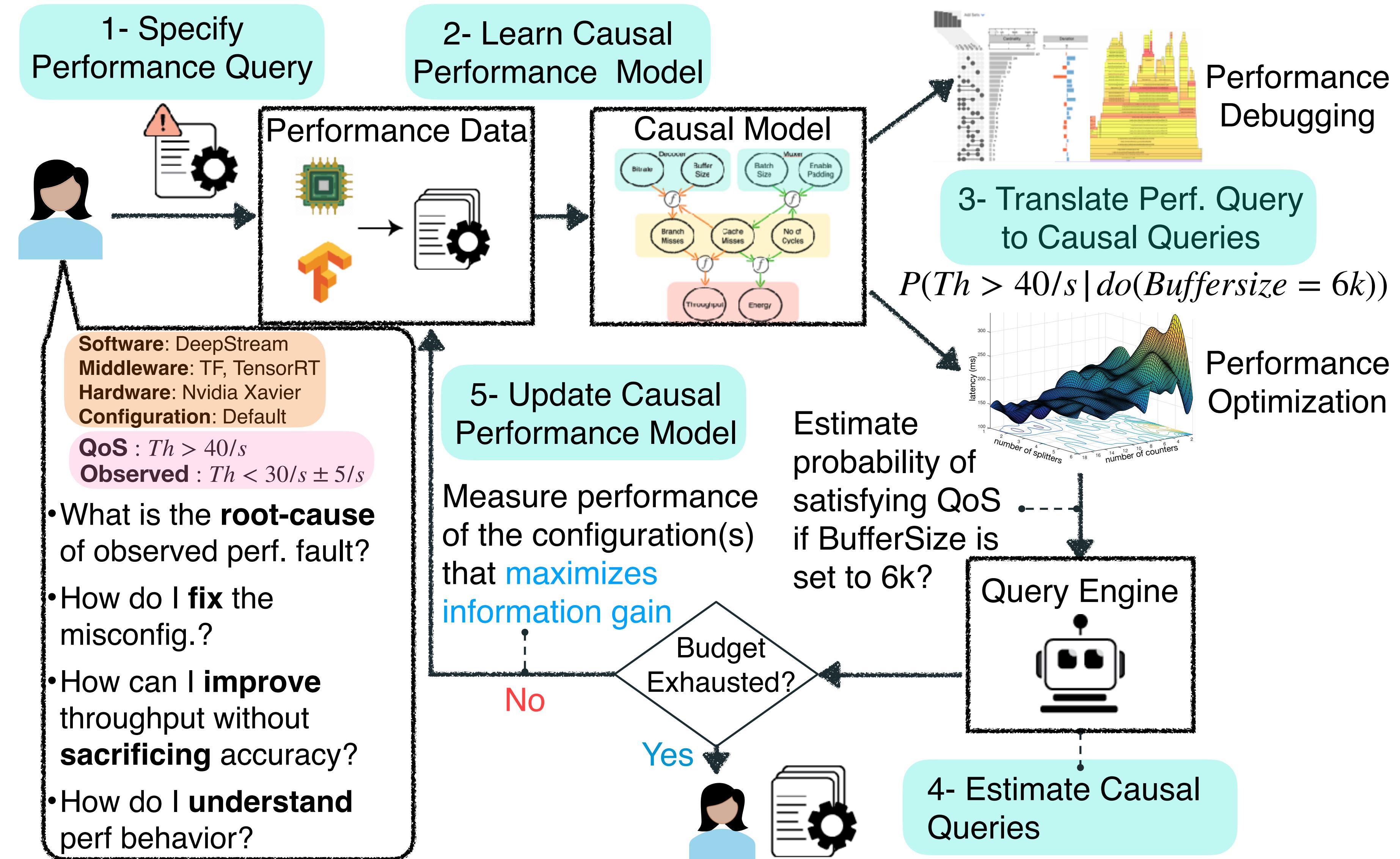
Results: Scalability

System	Configs	Events	Paths	Queries	Degree	Gain (%)	Time/Fault (in sec.)		
							Discovery	Query Eval	Total
SQLITE	34	19	32	191	3.6	93	9	14	291
	242	19	111	2234	1.9	94	57	129	1345
	242	288	441	22372	1.6	92	111	854	5312
DEEPSTREAM	53	19	43	497	3.1	86	16	32	1509
	53	288	219	5008	2.3	85	97	168	3113

Causal graphs are
sparse

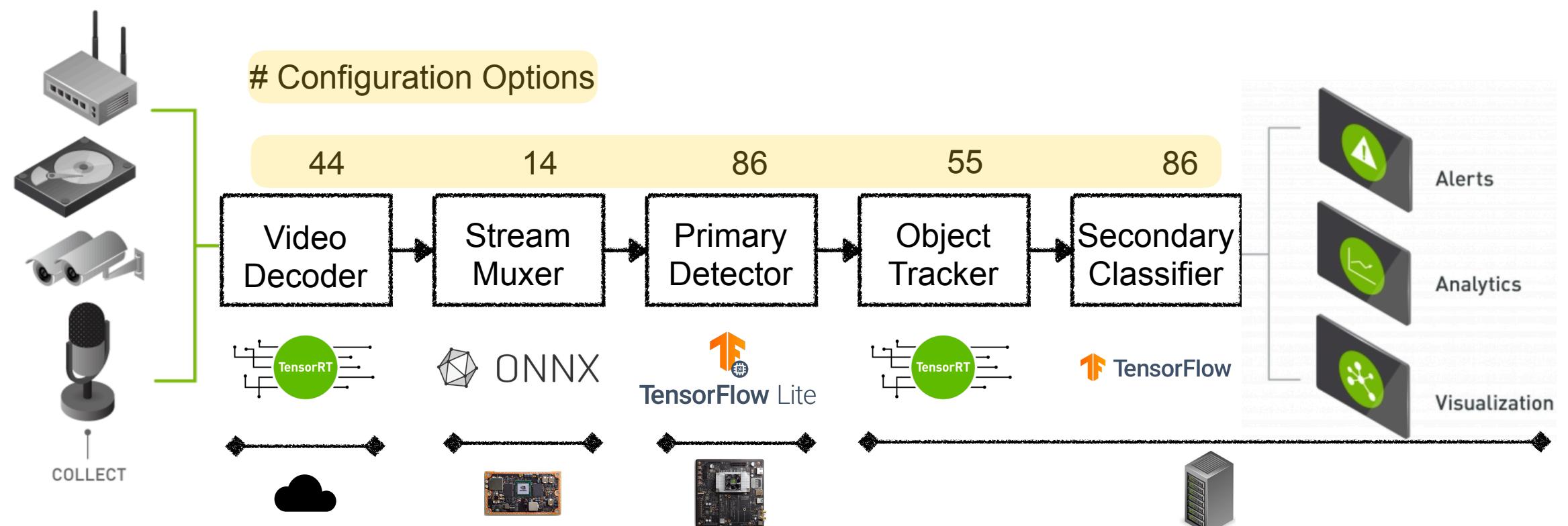
Summary: Causal AI for Systems

1. Learning a **causal performance model** for different downstream systems tasks.
2. The learned causal model is **transferable** across different environments.
3. The causal reasoning approach is **scalable** to large-scale systems.



The variability space of today's systems is exponentially increasing

Systems are heterogeneous, multiscale, multi-modal, and multi-stream



Variability Space =
Algorithm Selection +
Configuration Space +
System Architecture +
Deployment Environment

Evaluation: Experimental Setup

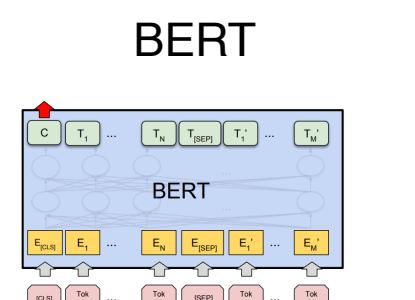
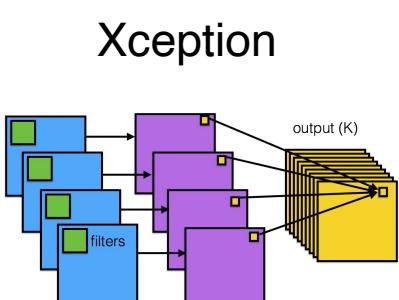
Hardware

Nvidia TX1	
CPU	4 cores, 1.3 GHz
GPU	128 Cores, 0.9 GHz
Memory	4 Gb, 25 GB/s

Nvidia TX2	
CPU	6 cores, 2 GHz
GPU	256 Cores, 1.3 GHz
Memory	8 Gb, 58 GB/s

Nvidia Xavier	
CPU	8 cores, 2.26 GHz
GPU	512 cores, 1.3 GHz
Memory	32 Gb, 137 GB/s

Systems

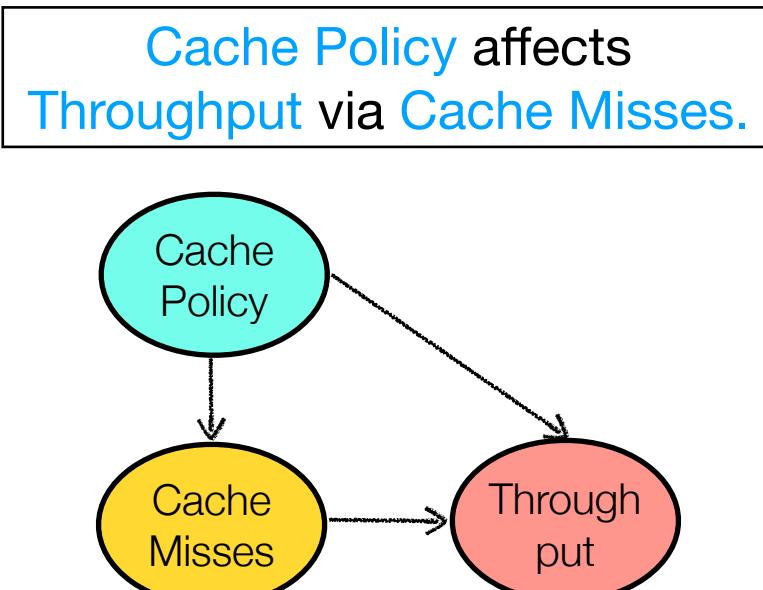
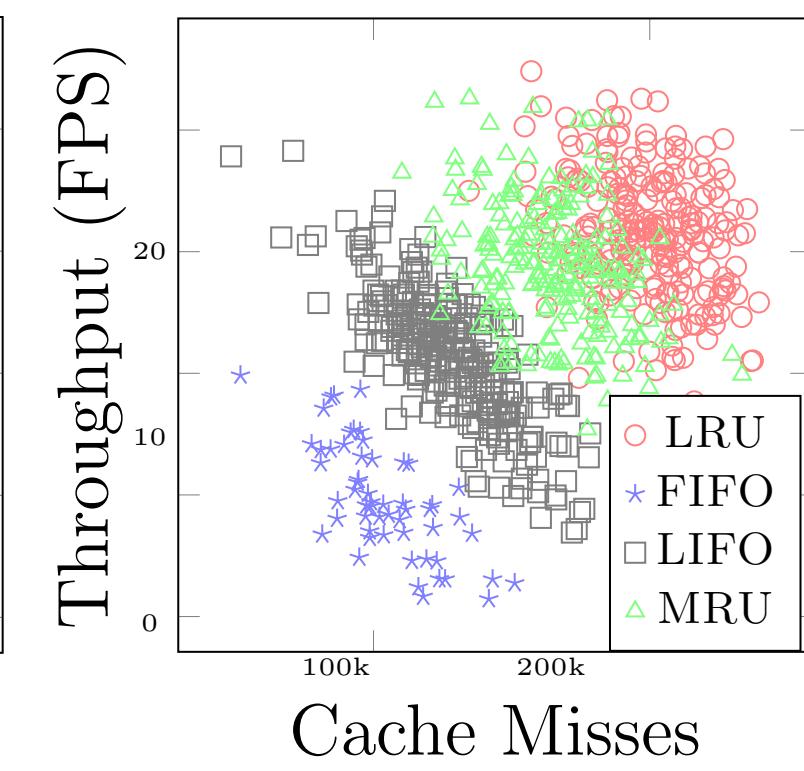
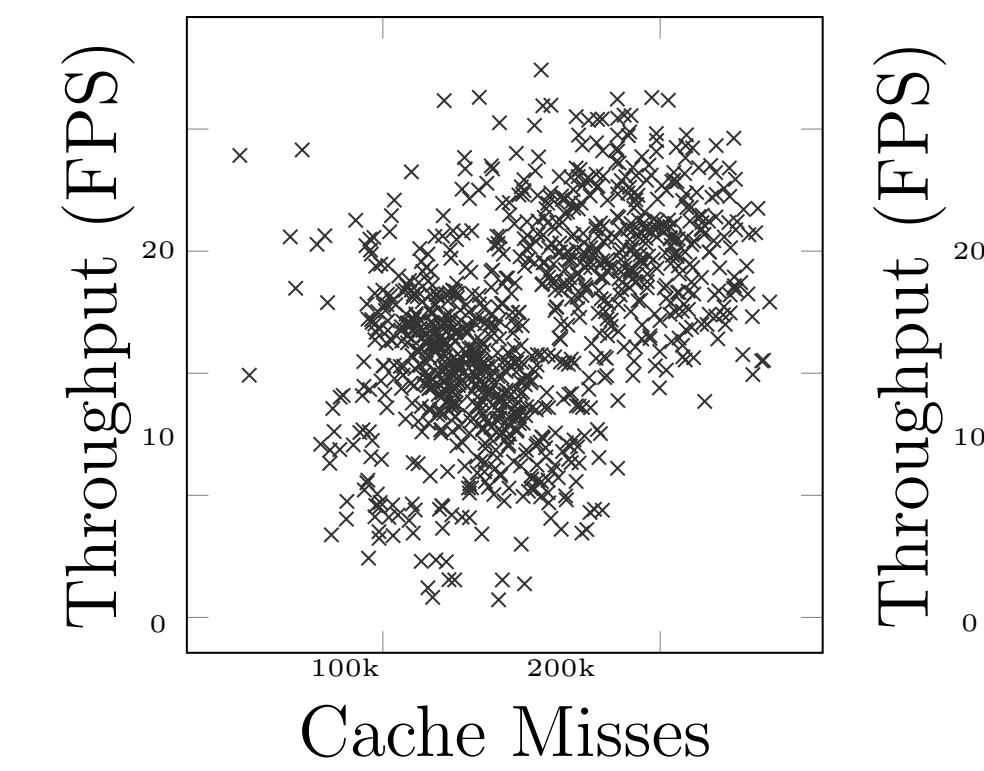


Configuration Space

- X 30 Configurations
 - 10 software
 - 10 OS/Kernel
 - 10 hardware
- X 17 System Events



Causal performance models produce correct explanations



Causal performance models capture correct interactions.

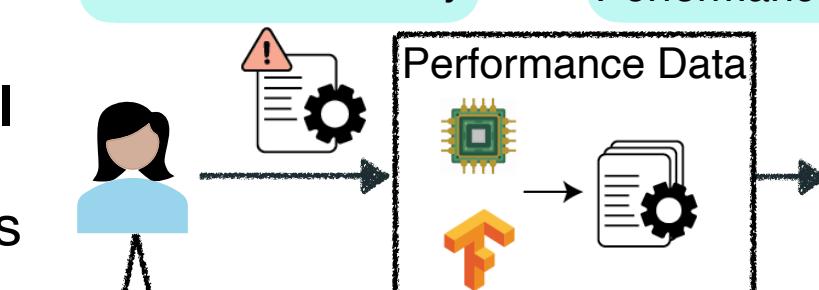
Summary: Causal AI for Systems

1. Learning a **causal performance model** for different downstream systems tasks.

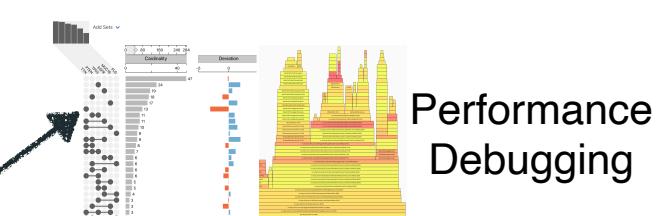
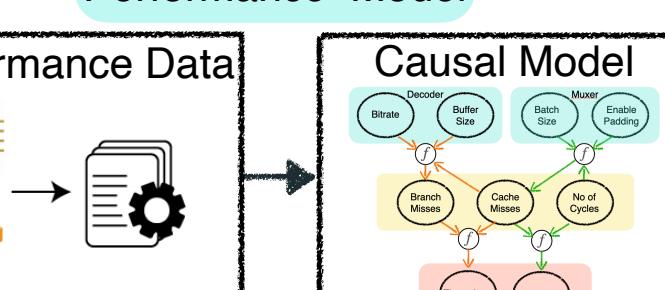
2. The learned causal model is **transferable** across different environments.

3. The causal reasoning approach is **scalable** to large-scale systems.

1- Specify Performance Query



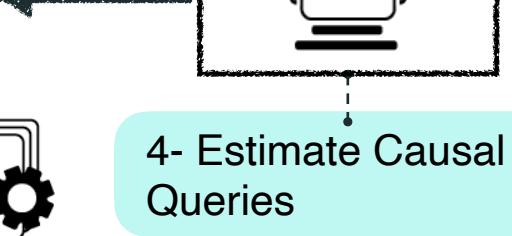
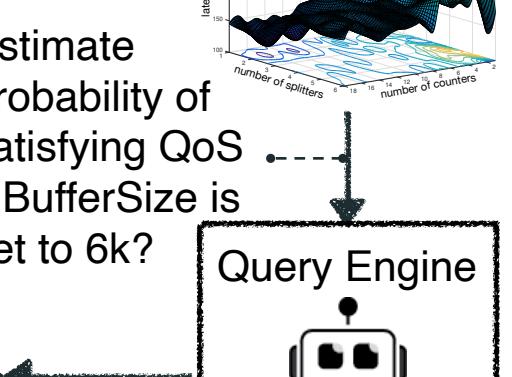
2- Learn Causal Performance Model



3- Translate Perf. Query to Causal Queries
 $P(Th > 40/s | do(Buffer size = 6k))$

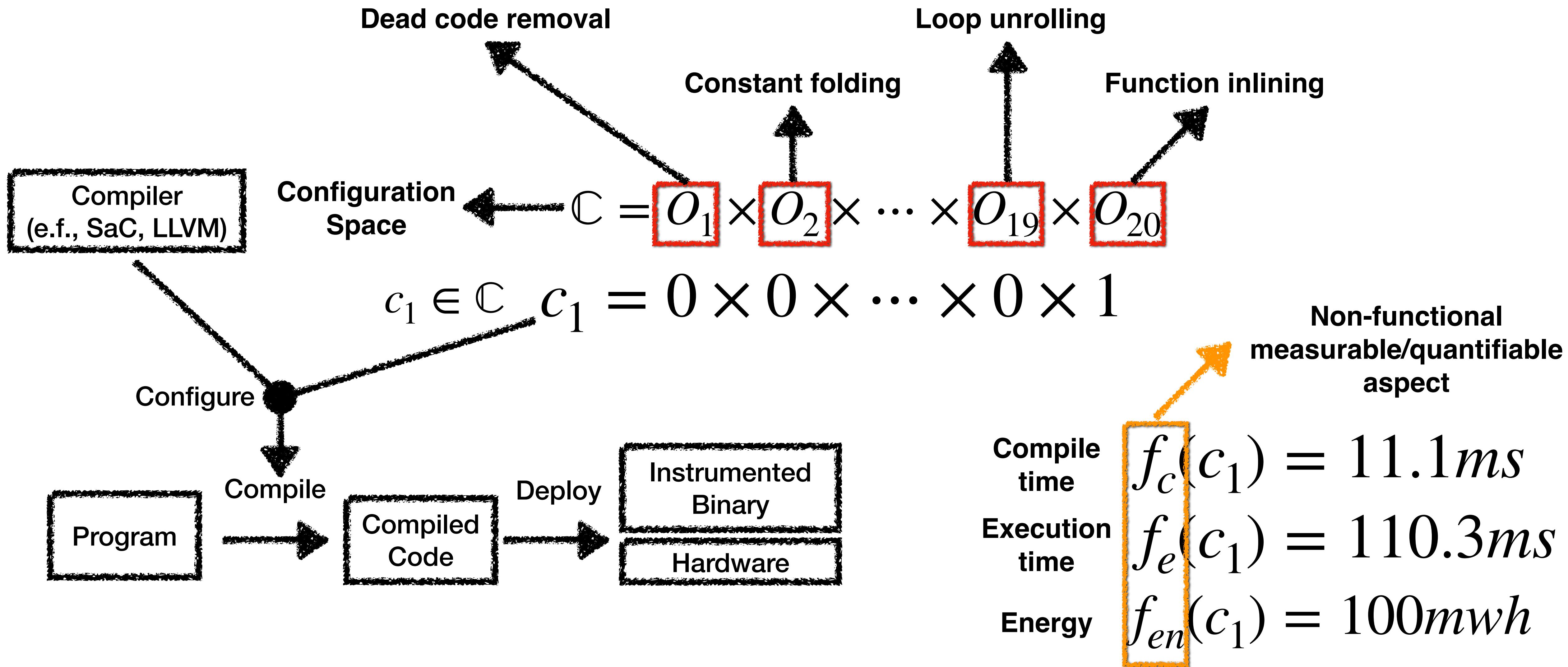
5- Update Causal Performance Model

Measure performance of the configuration(s) that maximizes information gain



How to resolve these issues faster?

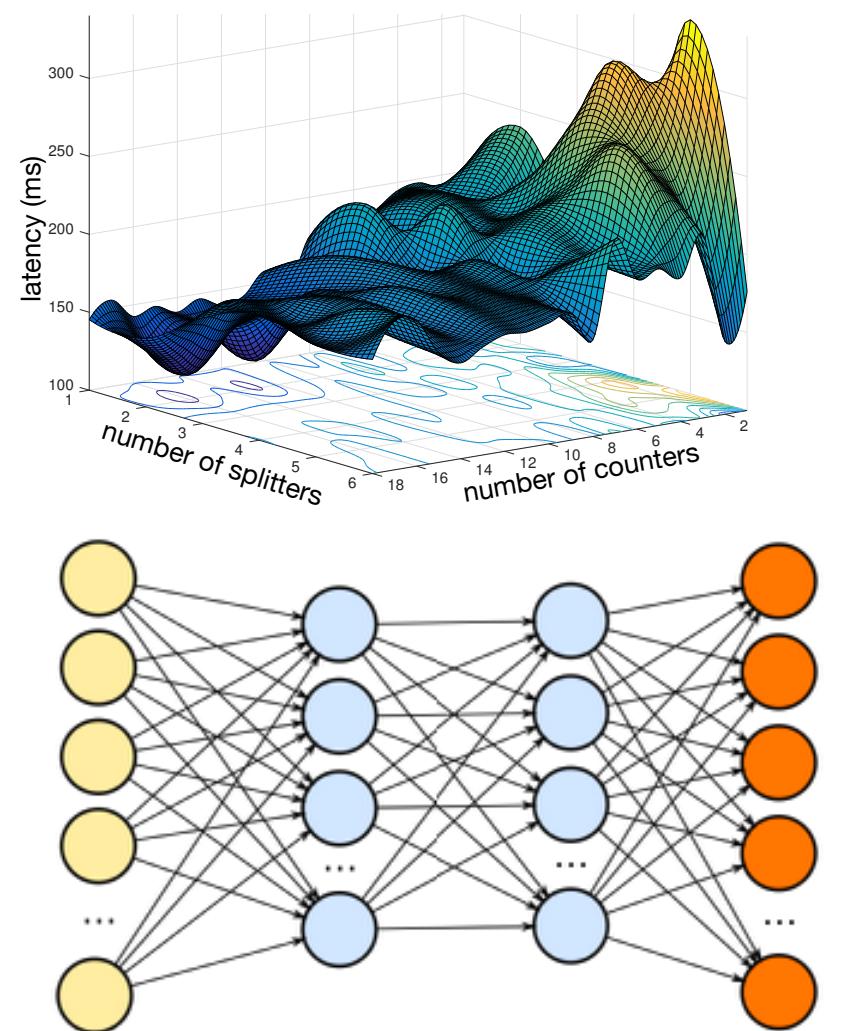
Performance measurement



Performance Influence Models

	Bitrate (bits/s)	Enable Padding	...	Cache Misses	...	Through put (fps)
c ₁	1k	1	...	42m	...	7
c ₂	2k	1	...	32m	...	22
...
c _n	5k	0	...	12m	...	25

Observational Data



Black-box models



Throughput

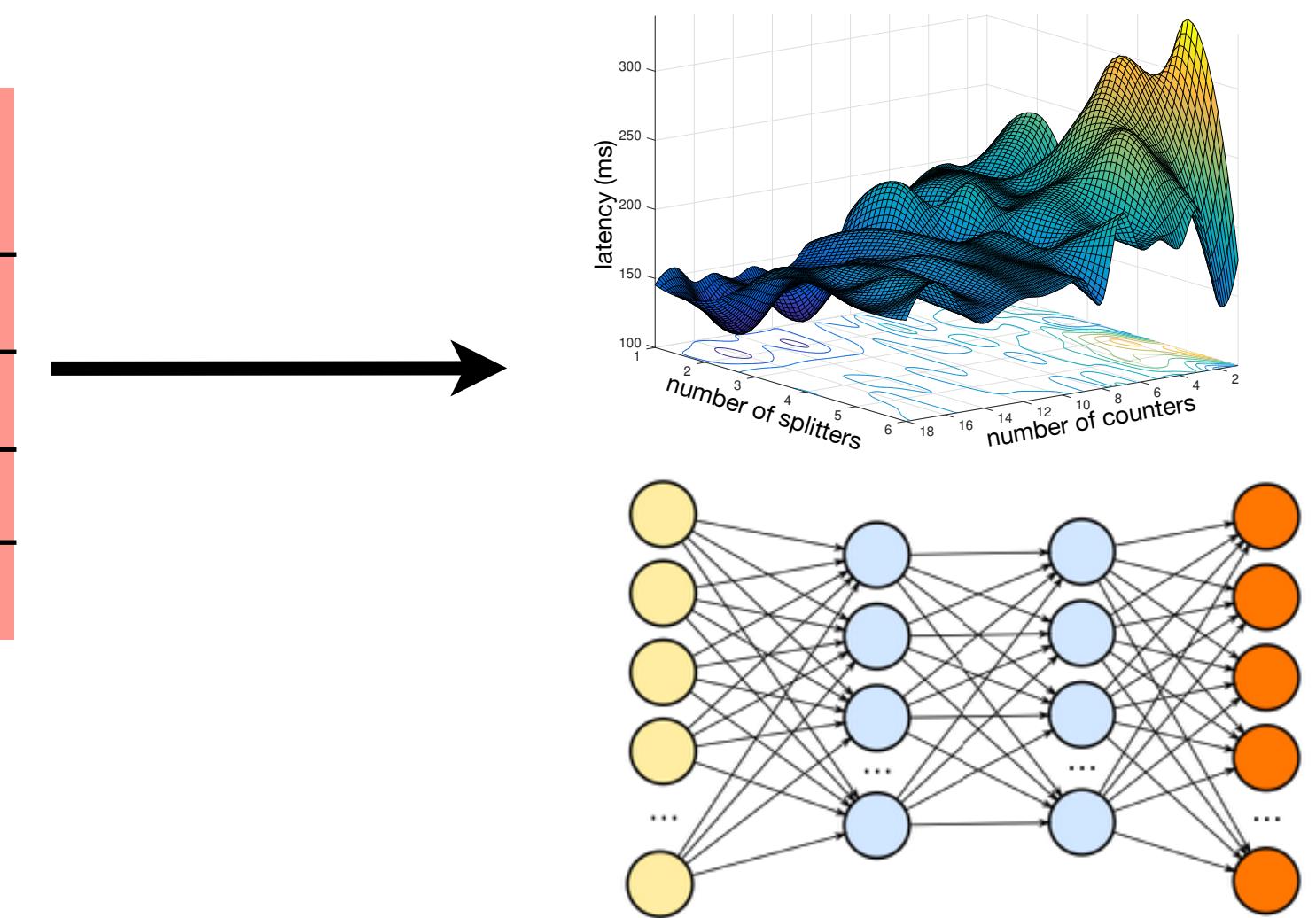
$$\begin{aligned} \text{Options} &\downarrow \\ \text{Throughput} = & 5.1 \times \text{Bitrate} + 2.5 \times \text{BatchSize} \\ &+ 12.3 \times \text{Bitrate} \times \text{BatchSize} \\ \text{Options} &\downarrow \\ \text{Discovered} &\text{Interactions} \end{aligned}$$

Regression Equation

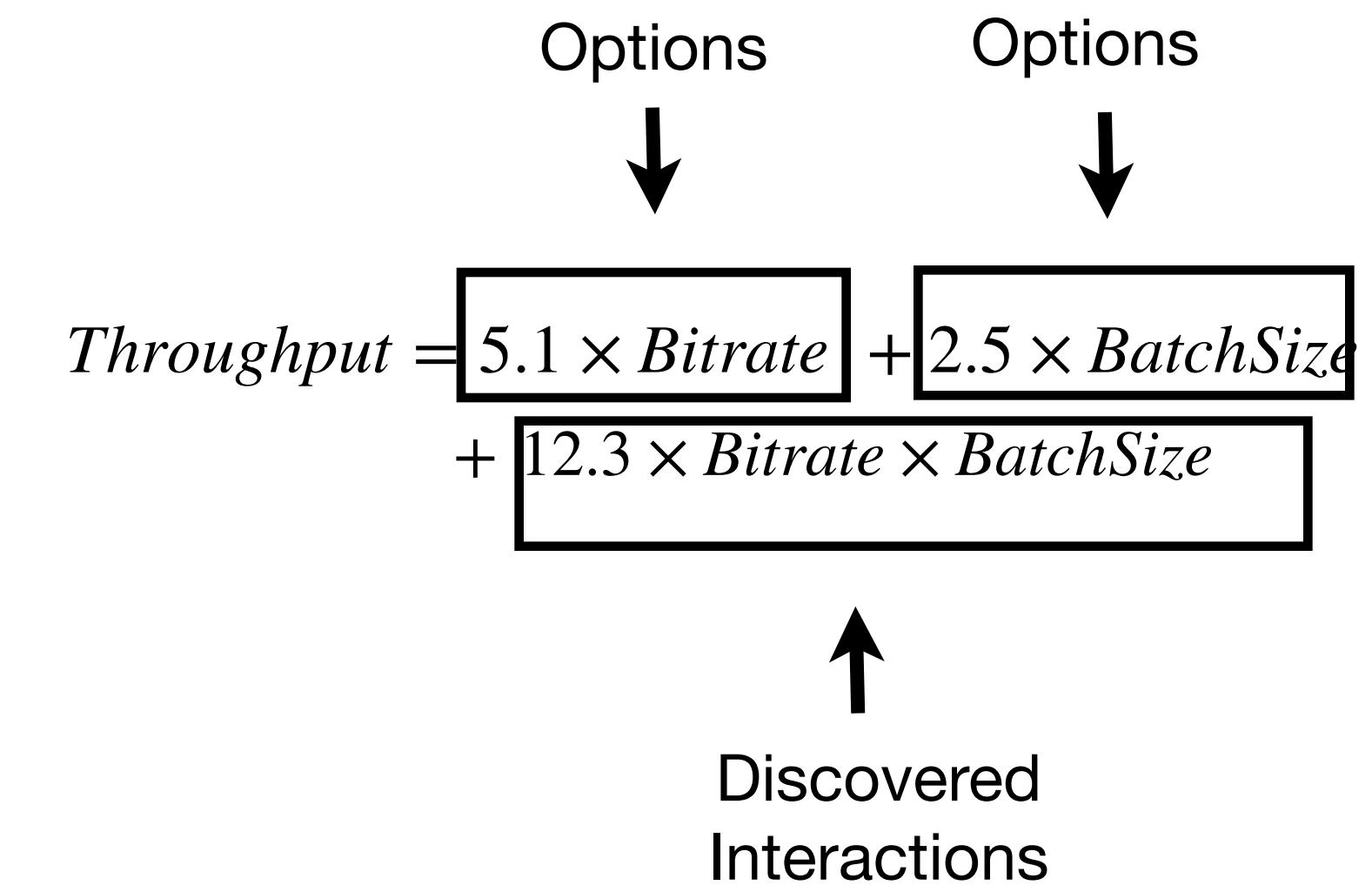
Performance Influence Models

	Bitrate (bits/s)	Enable Padding	...	Cache Misses	...	Through put (fps)
c ₁	1k	1	...	42m	...	7
c ₂	2k	1	...	32m	...	22
...
c _n	5k	0	...	12m	...	25

Observational Data



Black-box models



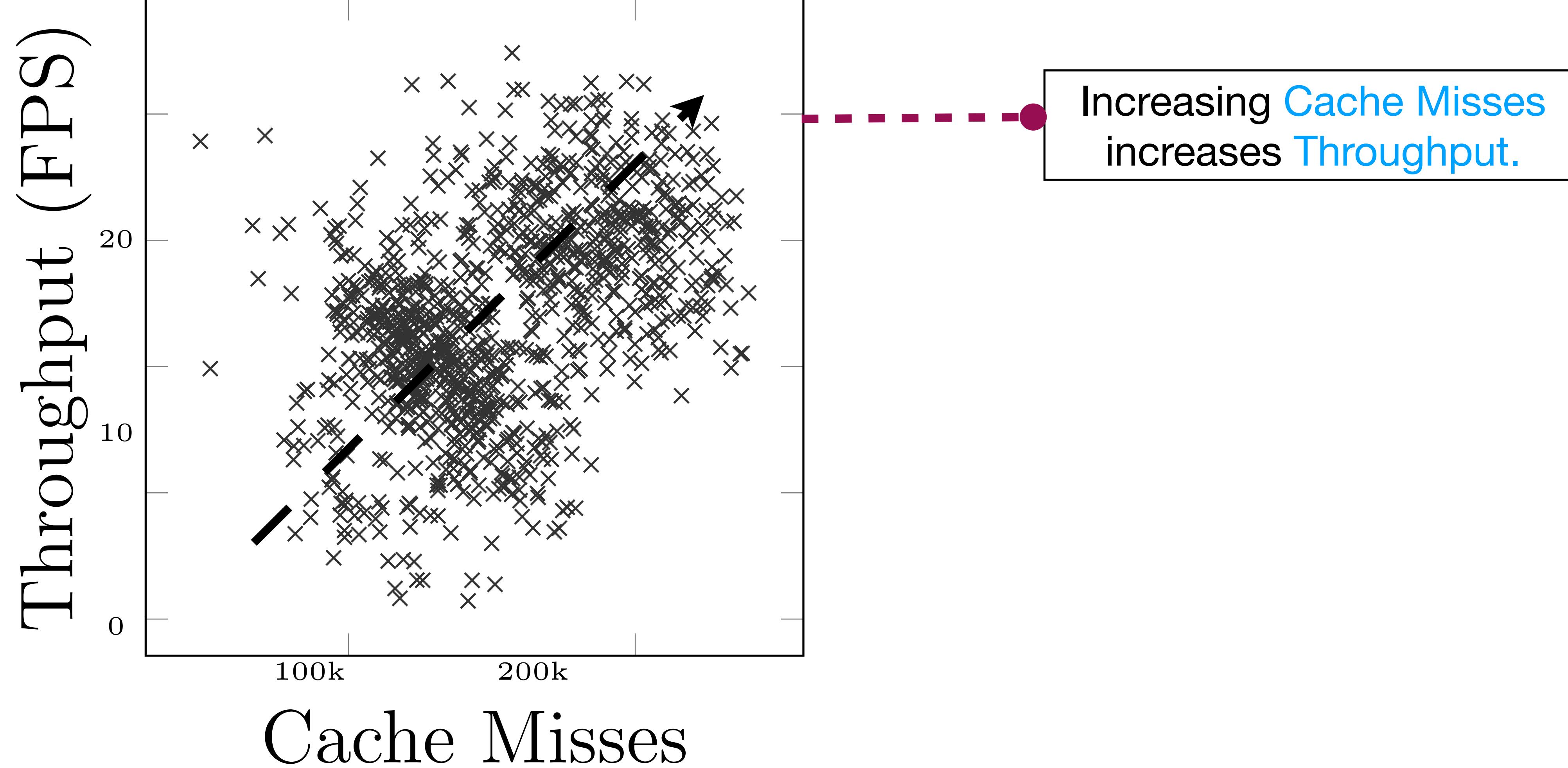
Regression Equation

These methods rely on **statistical correlations** to extract **meaningful information** required for performance tasks.

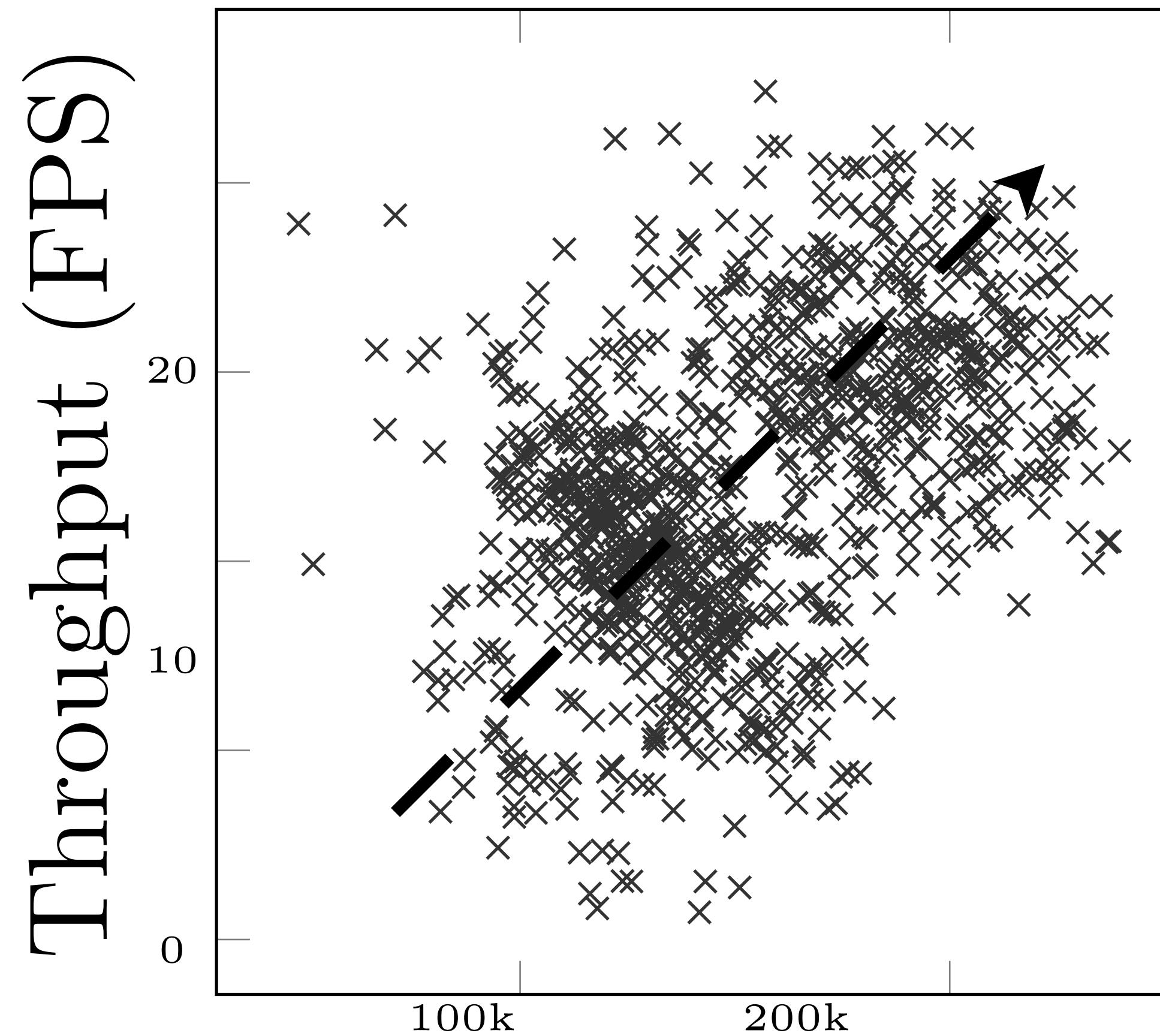
Performance Influence Models suffer from several shortcomings

- Performance influence models could produce **incorrect explanations**
- Performance influence models could produce **unreliable predictions.**
- Performance influence models could produce **unstable predictions** across environments and in the presence of measurement noise.

Performance Influence Models Issue: Incorrect Explanation



Performance Influence Models Issue: Incorrect Explanation



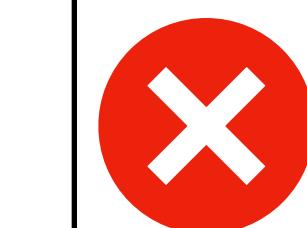
Cache Misses

Throughput (FPS)

Increasing Cache Misses increases Throughput.

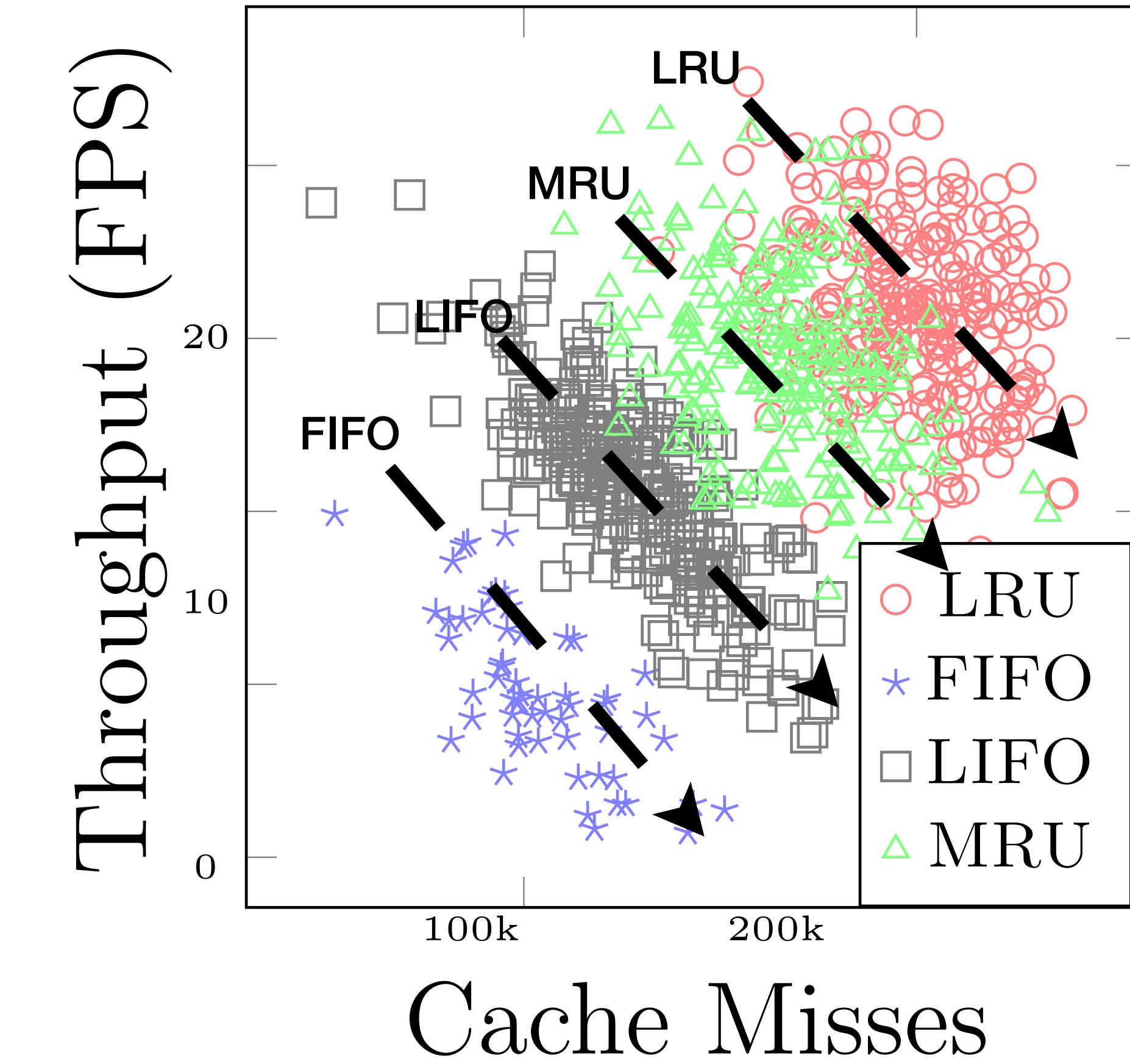
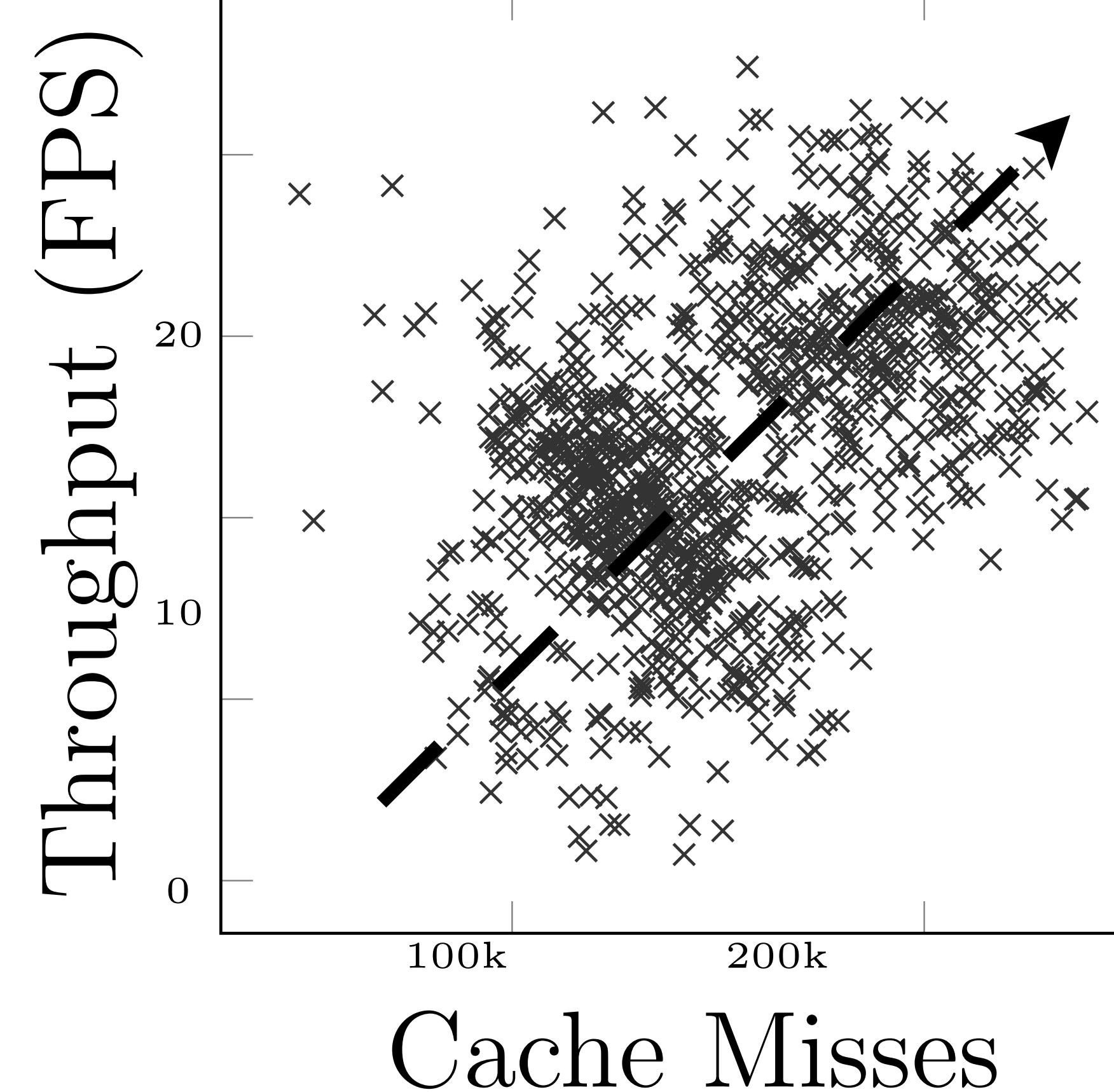
This is counter-intuitive

More Cache Misses should reduce Throughput not increase it



Any ML/statistical models built on this data will be incorrect.

Performance Influence Models Issue: Incorrect Explanation



Segregating data on **Cache Policy** indicates that within **each group**
Increase of Cache Misses result in a **decrease in Throughput**.

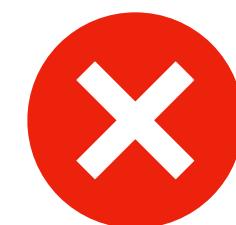
Performance Influence Models Issue: Unstable Predictors

Performance influence model in TX2.

$$\begin{aligned} \text{Throughput} = & 2 \times \text{Bitrate} + 1.9 \times \text{BatchSize} + 1.8 \times \text{BufferSize} + 0.5 \times \text{EnablePadding} + 5.9 \times \text{Bitrate} \times \text{BufferSize} \\ & + 6.2 \times \text{Bitrate} \times \text{EnablePadding} + 4.1 \times \text{Bitrate} \times \text{BufferSize} \times \text{EnablePadding} \end{aligned}$$

Performance influence model in Xavier.

$$\text{Throughput} = 5.1 \times \text{Bitrate} + 2.5 \times \text{BatchSize} + 12.3 \times \text{Bitrate} \times \text{BatchSize}$$



Performance Influence Models **change significantly** in new environments resulting in less accuracy.

Performance Influence Models Issue: Unstable Predictors

Performance influence model in TX2.

$$\begin{aligned} \text{Throughput} = & [2 \times \text{Bitrate}] + [1.9 \times \text{BatchSize}] + 1.8 \times \text{BufferSize} + 0.5 \times \text{EnablePadding} + 5.9 \times \text{Bitrate} \times \text{BufferSize} \\ & + 6.2 \times \text{Bitrate} \times \text{EnablePadding} + 4.1 \times \text{Bitrate} \times \text{BufferSize} \times \text{EnablePadding} \end{aligned}$$

Performance influence model in Xavier.

$$\text{Throughput} = [5.1 \times \text{Bitrate}] + [2.5 \times \text{BatchSize}] + 12.3 \times \text{Bitrate} \times \text{BatchSize}$$



Performance influence are cannot be **reliably**
used across environments.

Performance Influence Models Issue: Non-generalizability

Performance influence model in TX2

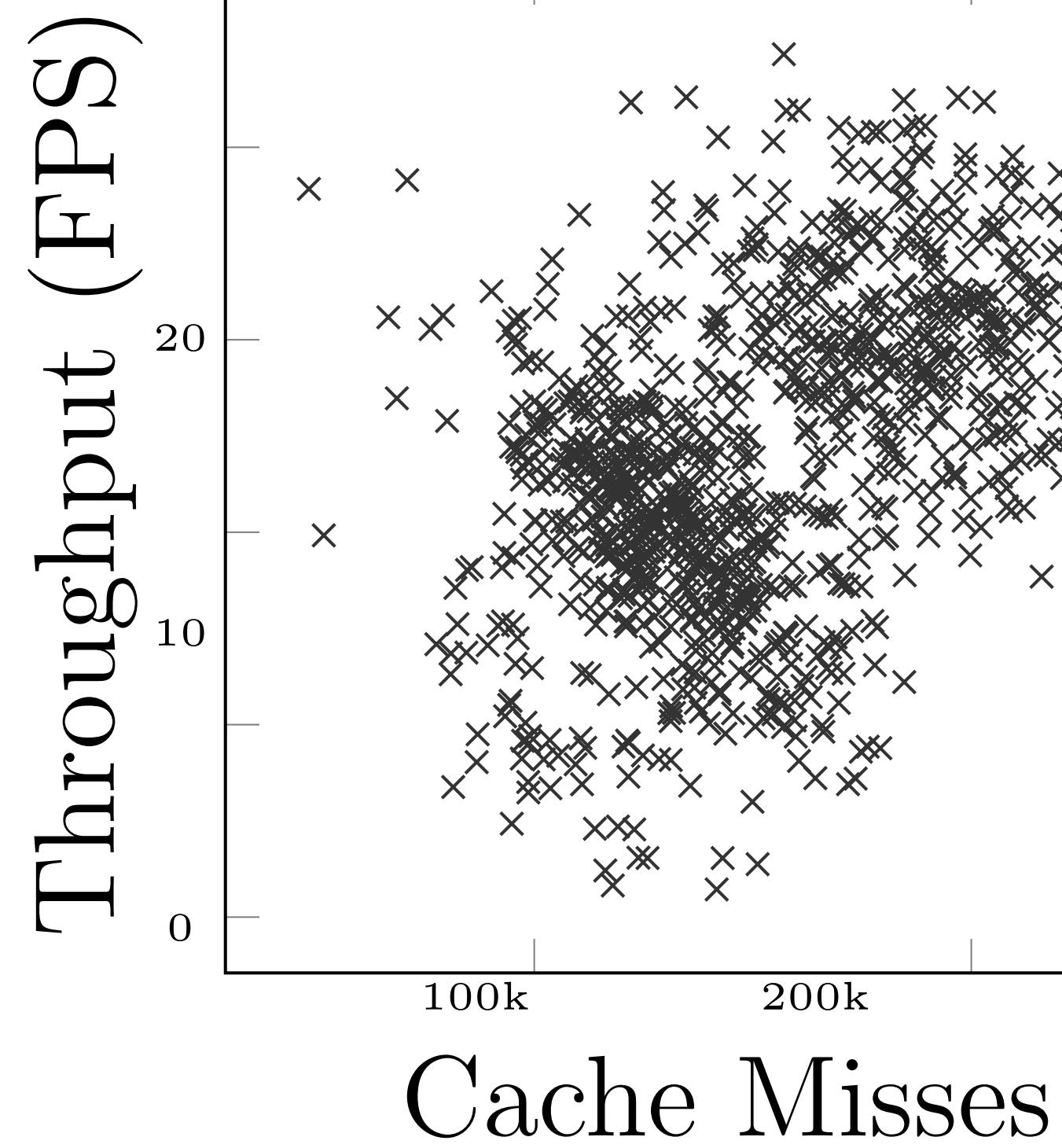
$$\begin{aligned} \text{Throughput} = & 2 \times \text{Bitrate} + 1.9 \times \text{BatchSize} + 1.8 \times \text{BufferSize} + 0.5 \times \text{EnablePadding} + 5.9 \times \text{Bitrate} \times \text{BufferSize} \\ & + 6.2 \times \text{Bitrate} \times \text{EnablePadding} + 4.1 \times \text{Bitrate} \times \text{BufferSize} \times \text{EnablePadding} \end{aligned}$$

Performance influence model in Xavier.

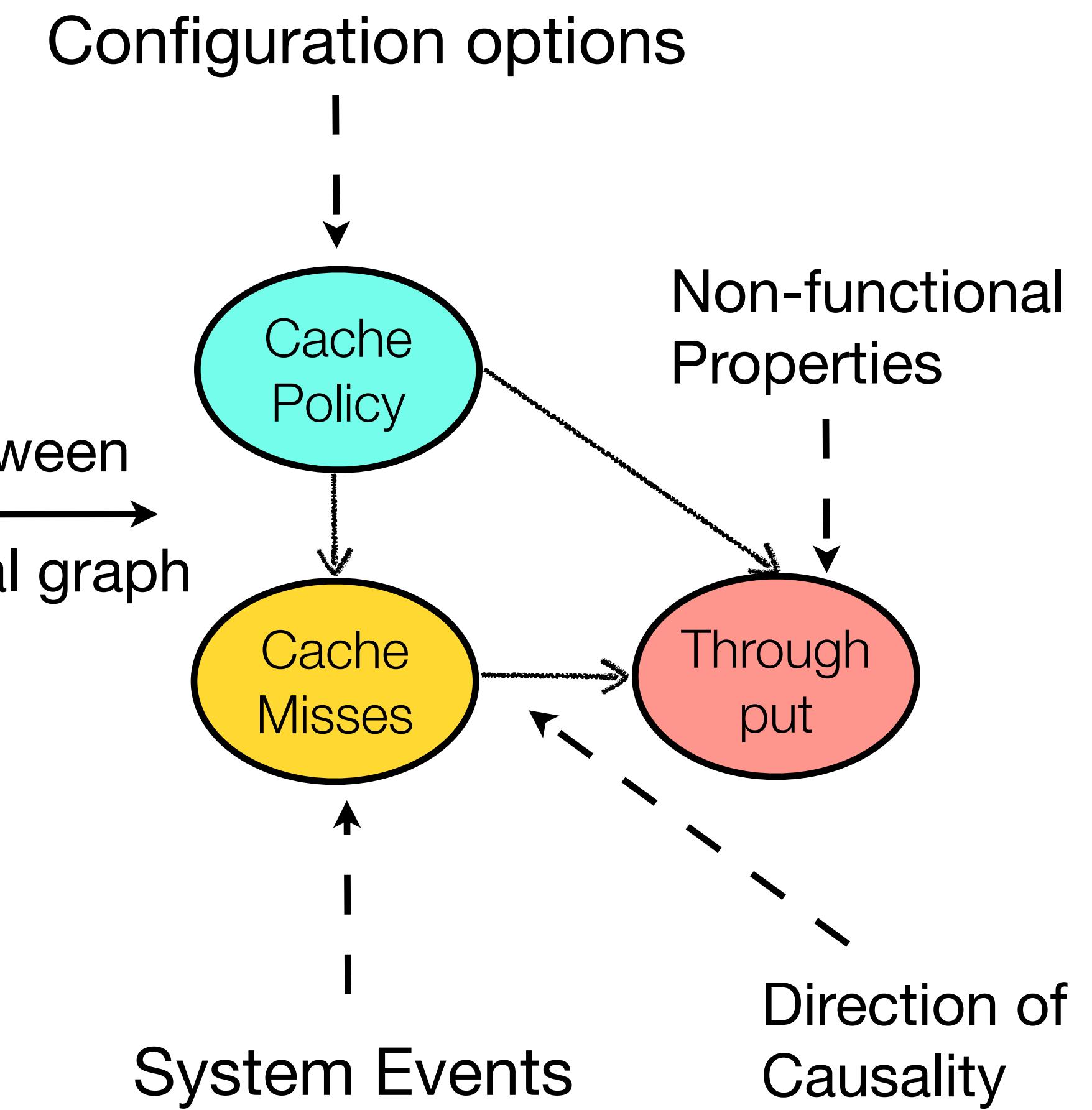
$$\text{Throughput} = 5.1 \times \text{Bitrate} + 2.5 \times \text{BatchSize} + 12.3 \times \text{Bitrate} \times \text{BatchSize}$$


Performance influence models do not **generalize**
well across deployment environments.

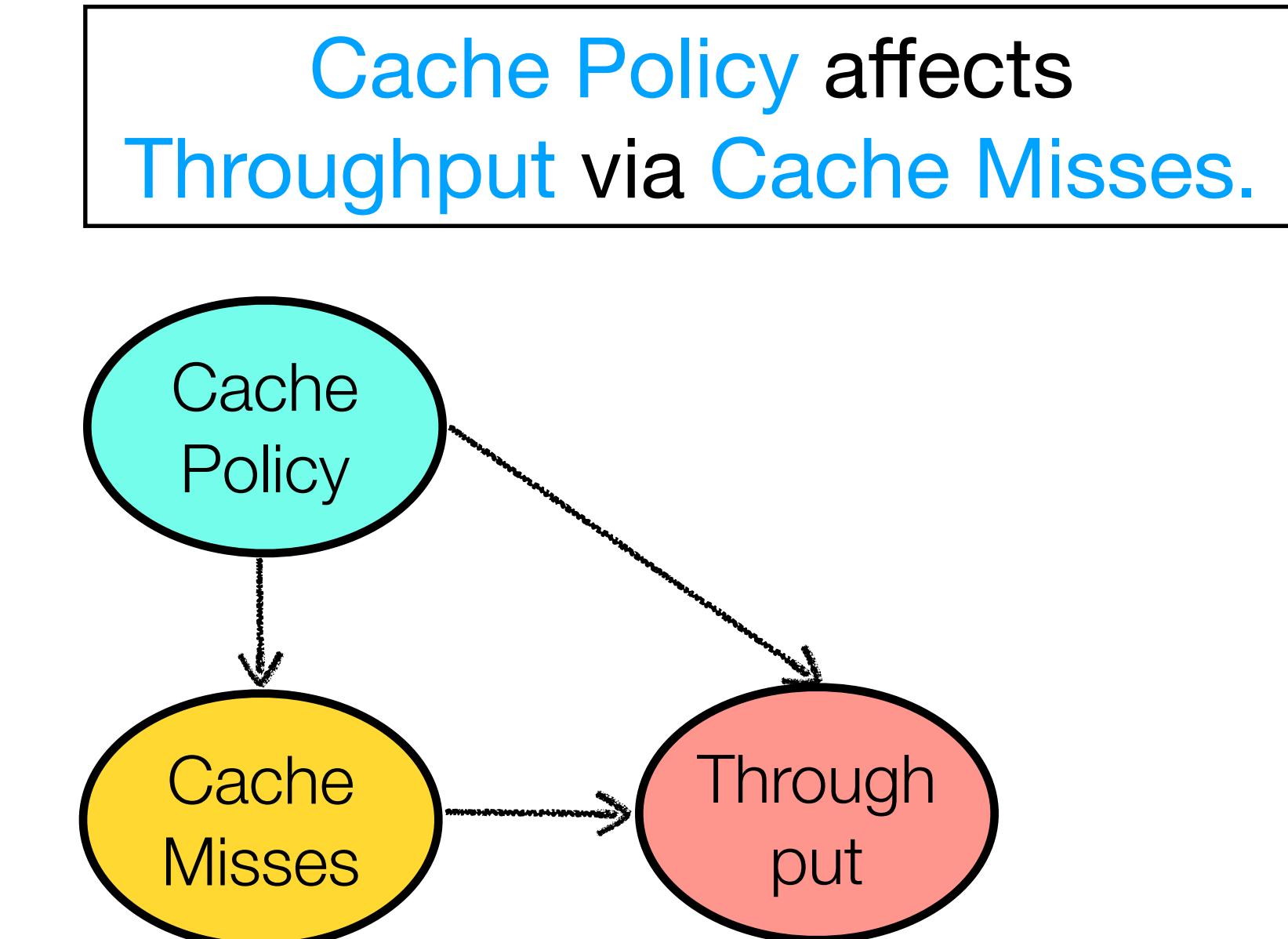
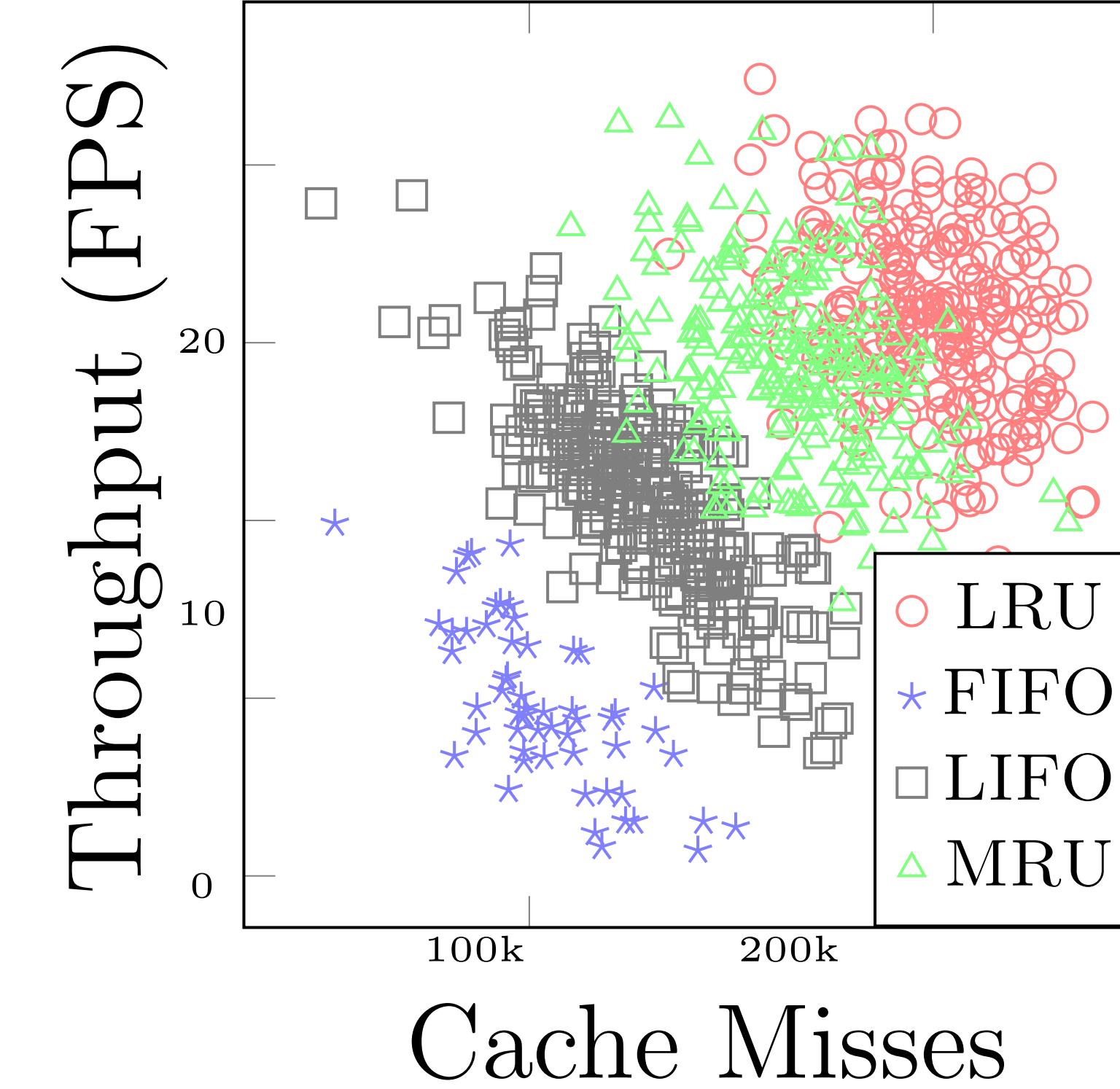
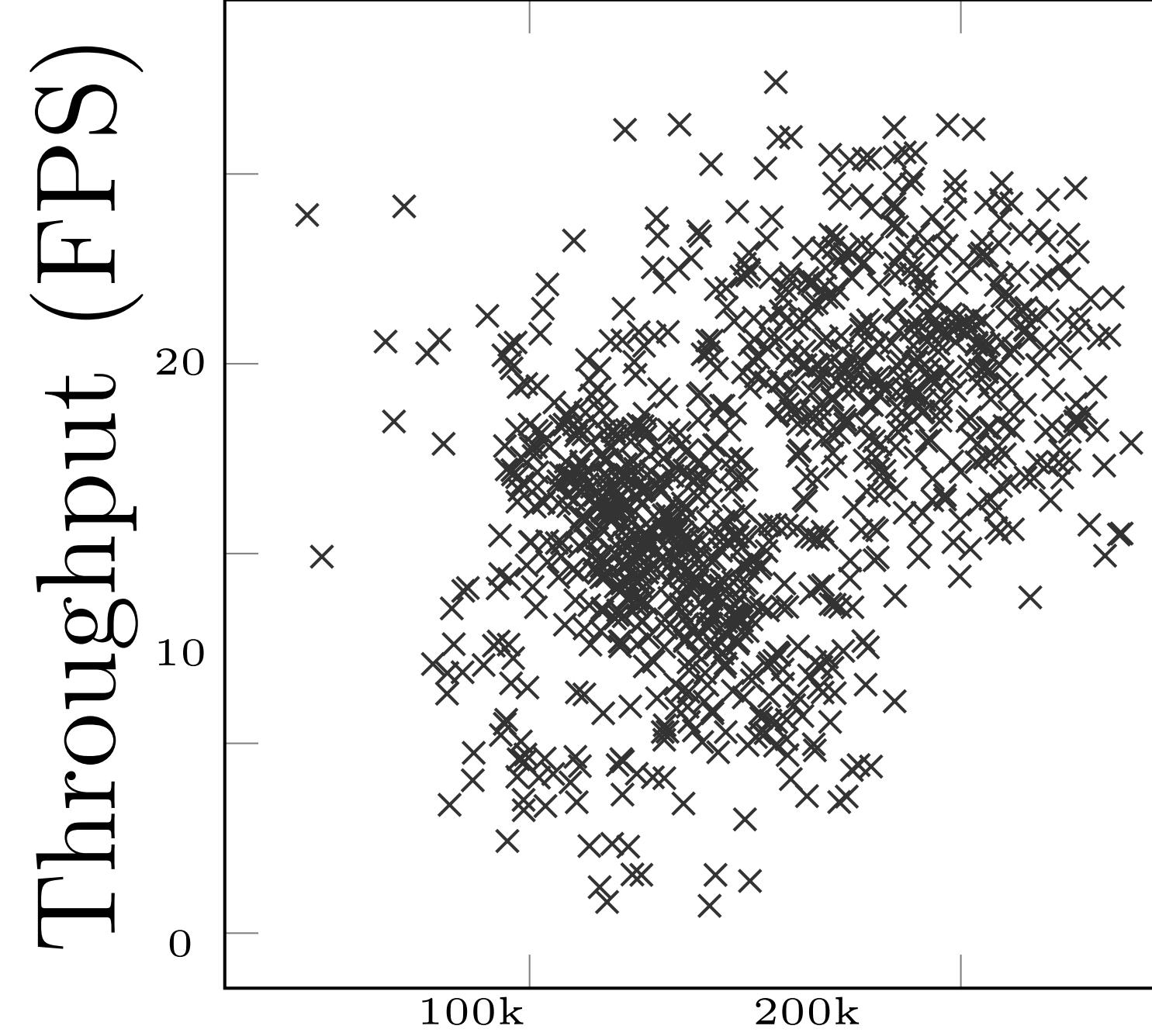
Causal Performance Model



Expresses the **relationships** between
interacting variables as a causal graph



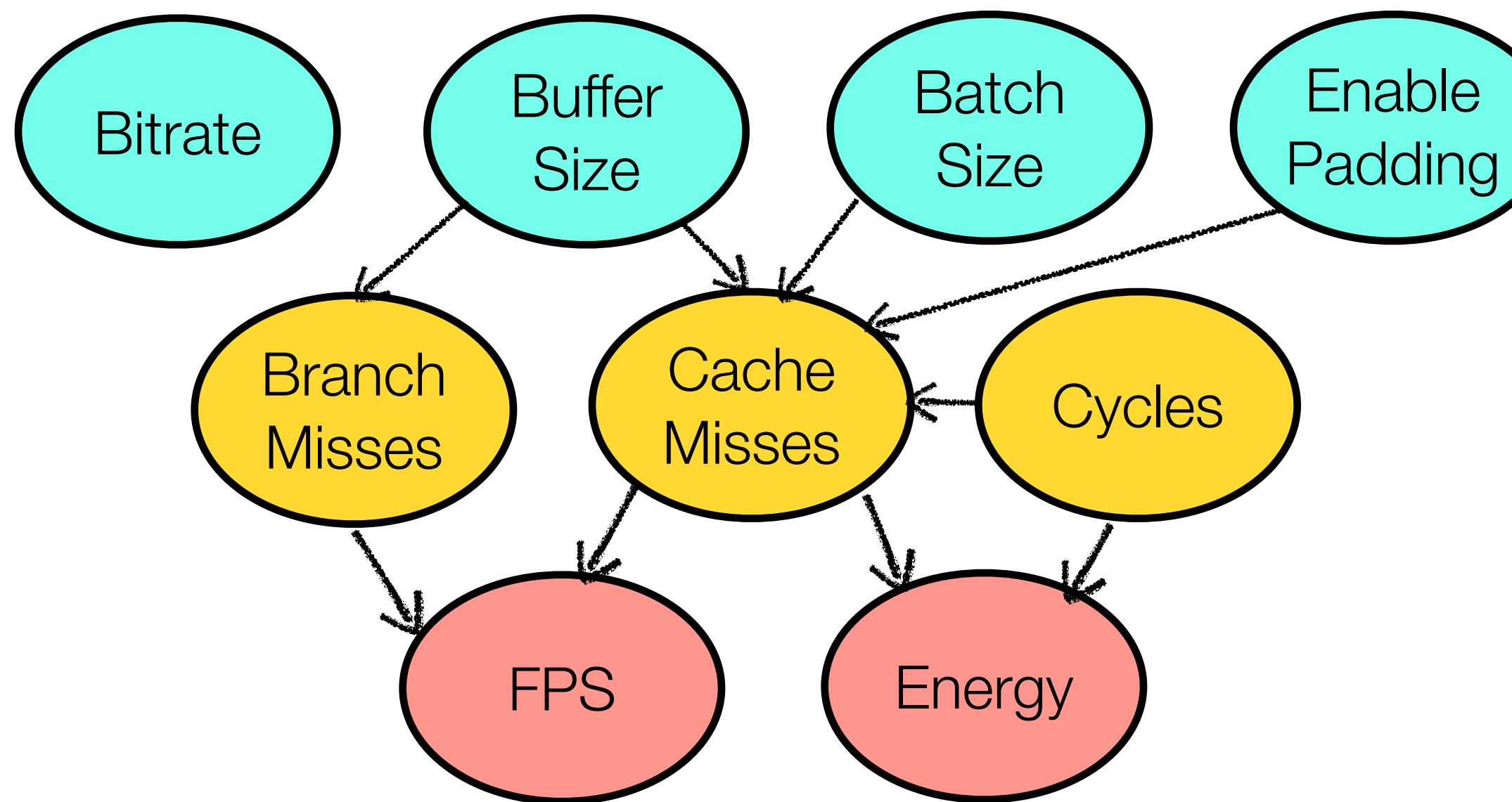
Why Causal Inference? - Produces Correct Explanations



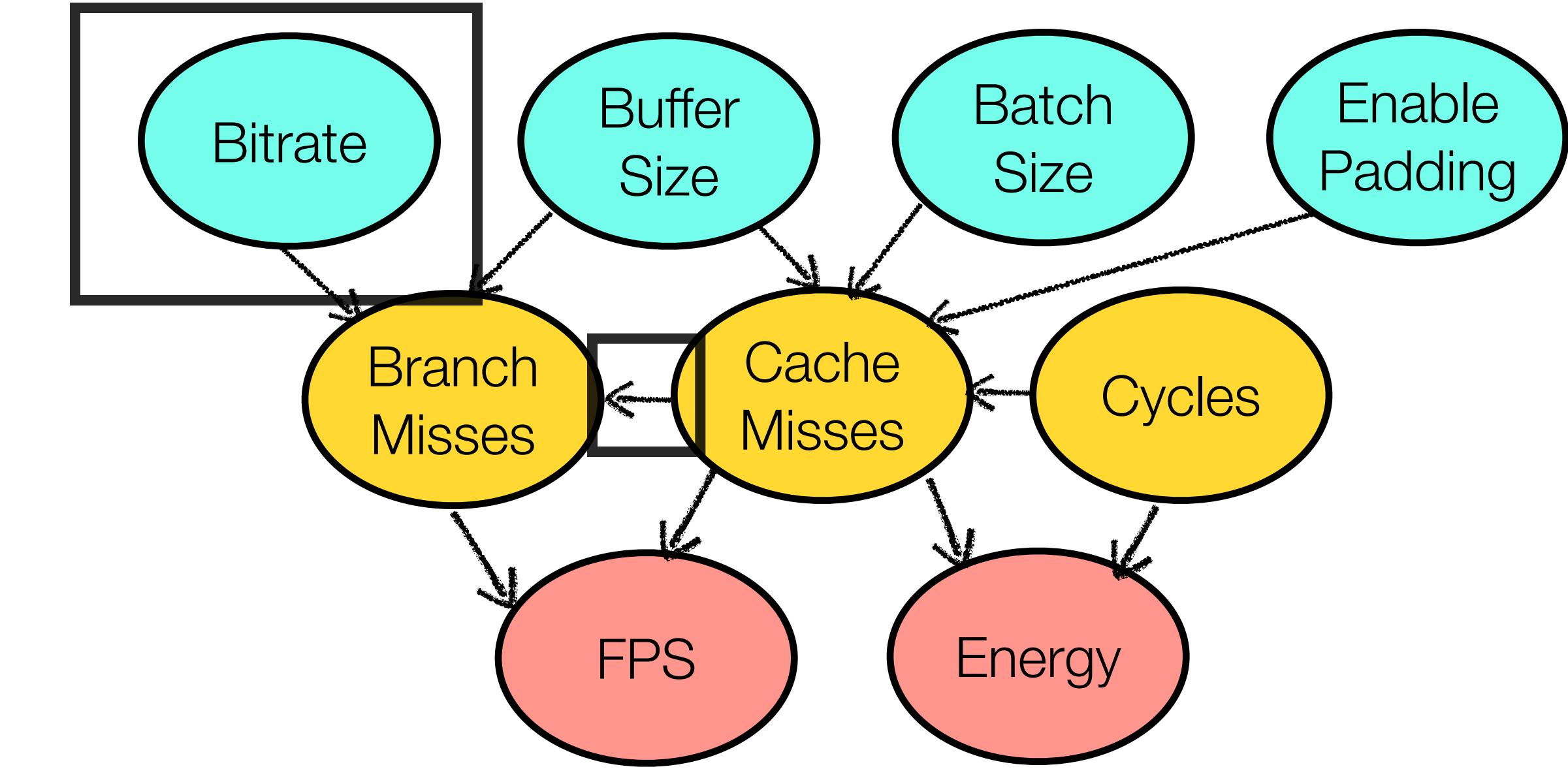
Causal Performance Models recovers the correct interactions.

Why Causal Inference? - Minimal Structure Change

A partial causal performance model in Jetson TX2

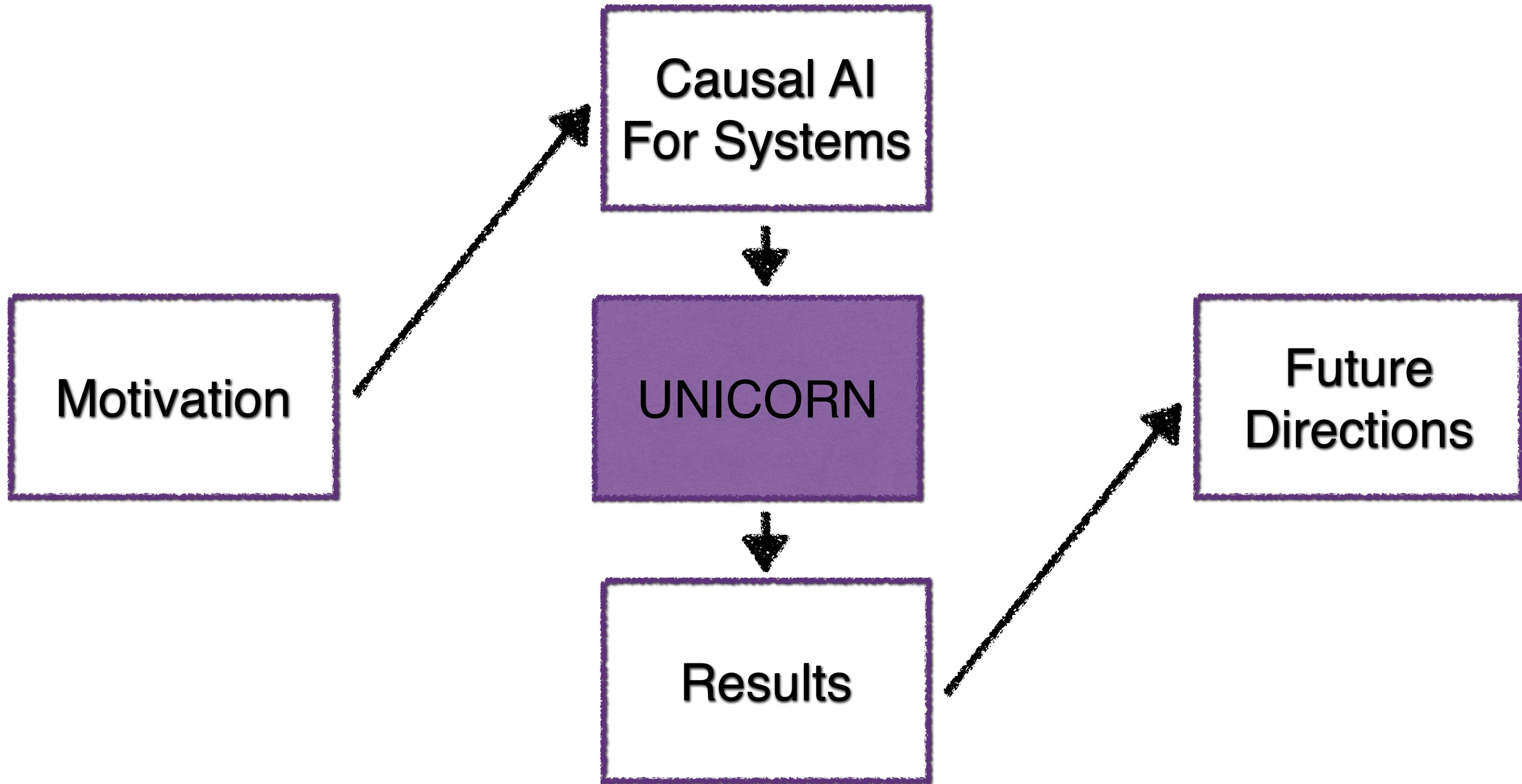


A partial causal performance model in Jetson Xavier



Causal models remain
relatively **stable**

Outline

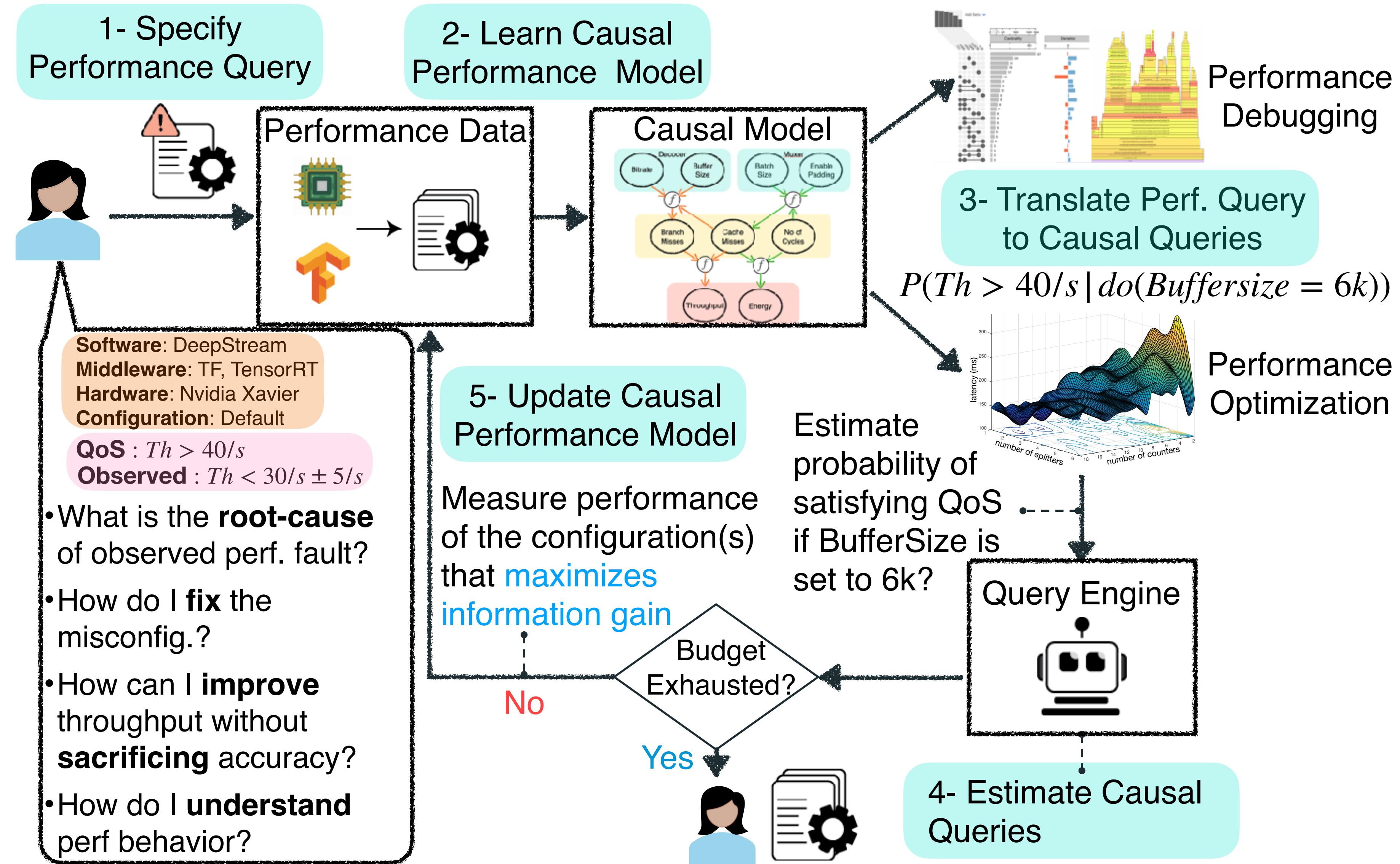


UNICORN: Our Causal AI for Systems Method

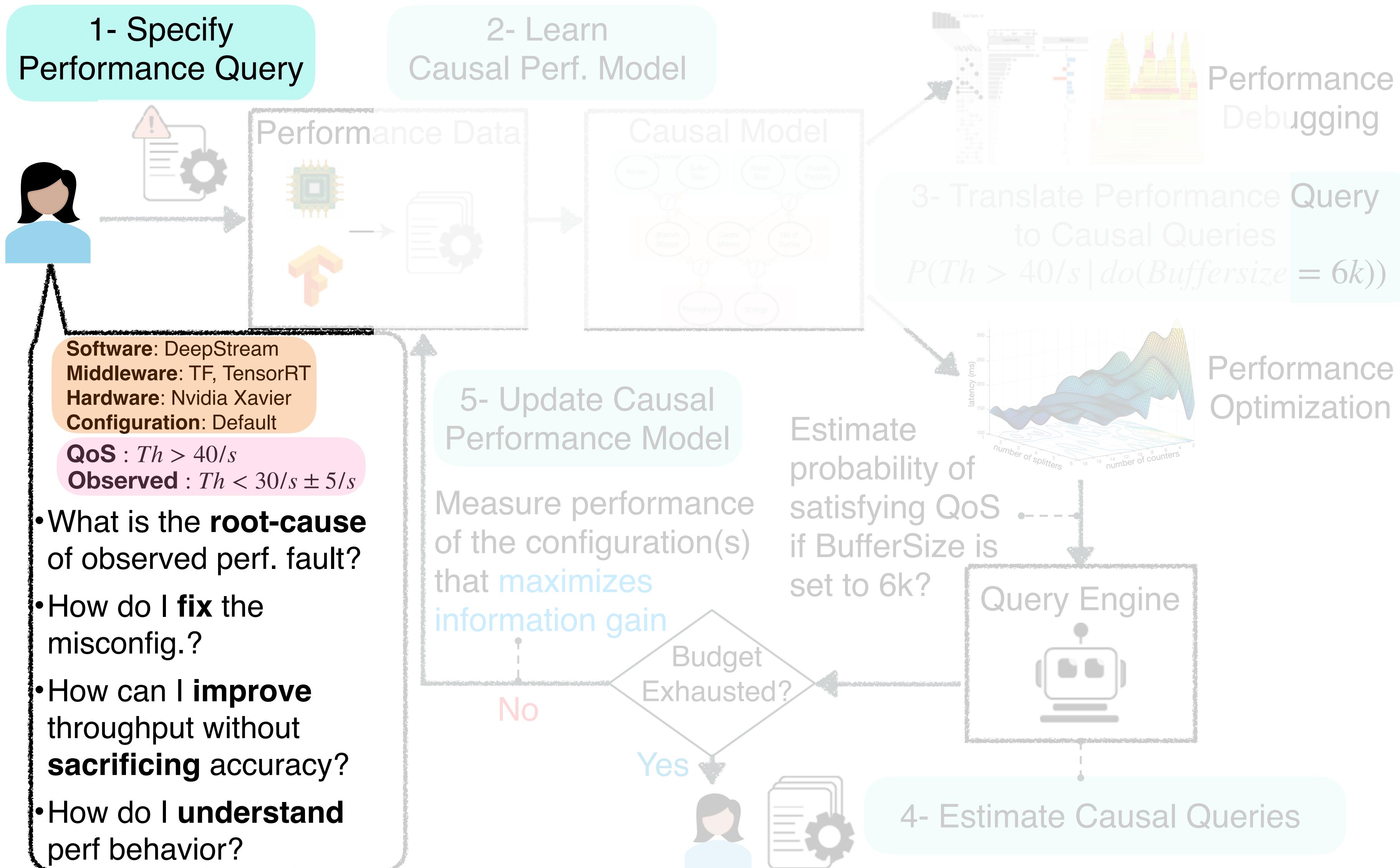
- **Build a Causal Performance Model** that capture the interactions options in the variability space using the observation performance data.
- Iterative causal performance **model evaluation** and **model update**
- Perform downstream performance tasks such as performance debugging & optimization using **Causal Reasoning**



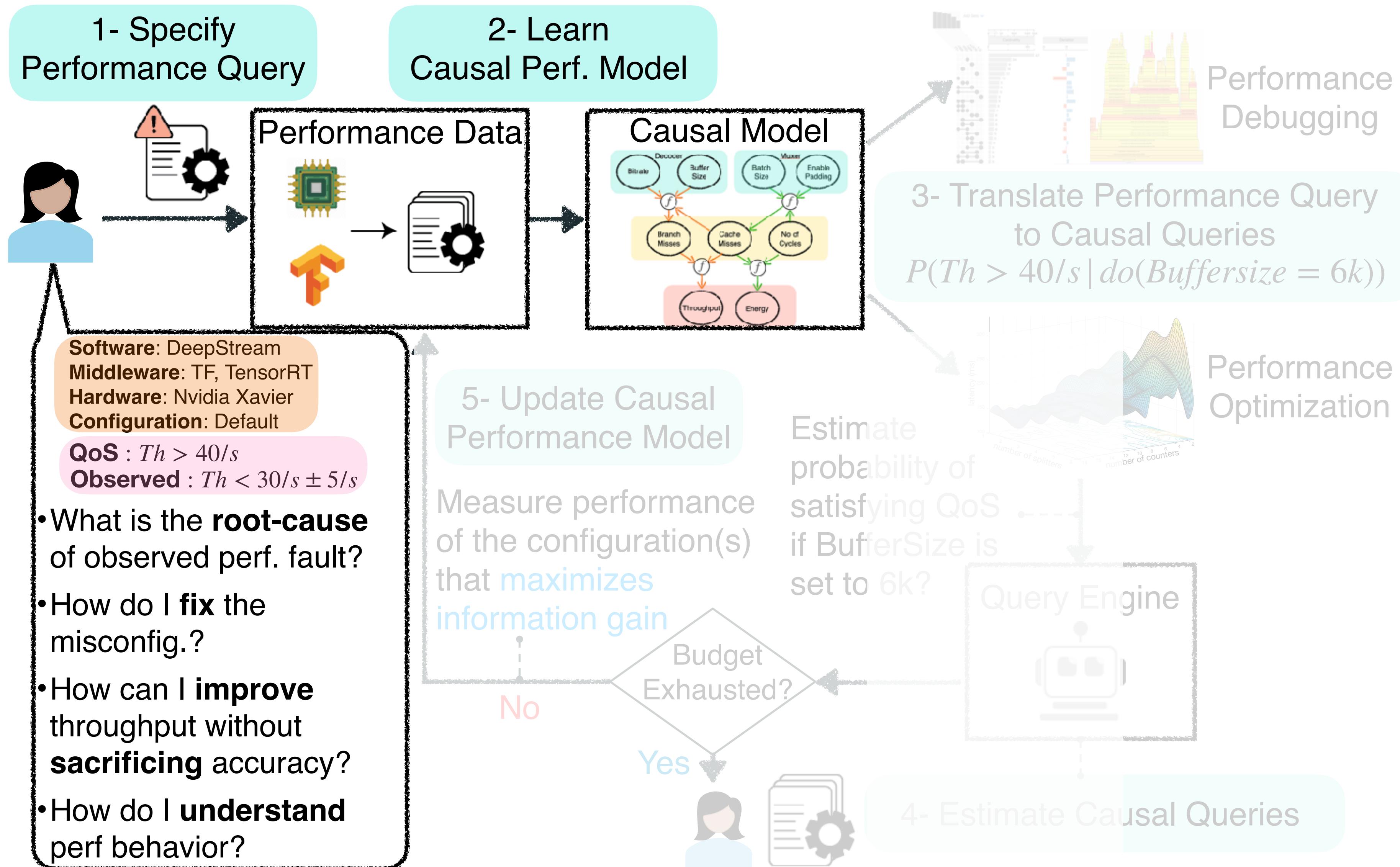
UNICORN: Our Causal AI for Systems Method



UNICORN: Our Causal AI for Systems Method



UNICORN: Our Causal AI for Systems Method



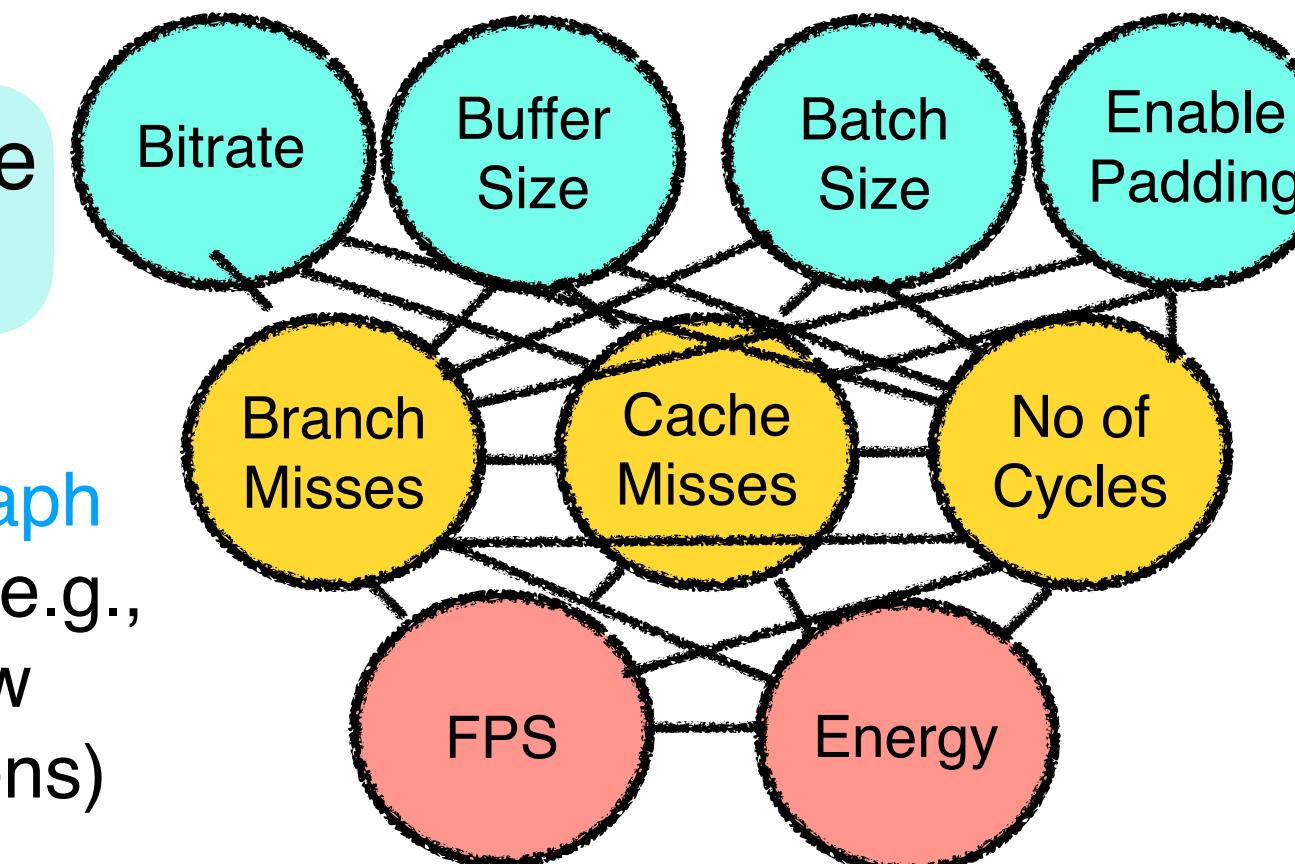
Learning Causal Performance Model

	Bitrate (bits/s)	Enable Padding	...	Cache Misses	...	Through put (fps)
c ₁	1k	1	...	42m	...	7
c ₂	2k	1	...	32m	...	22
...
c _n	5k	0	...	12m	...	25

1- Recovering the Skelton



fully connected graph given constraints (e.g., no connections btw configuration options)

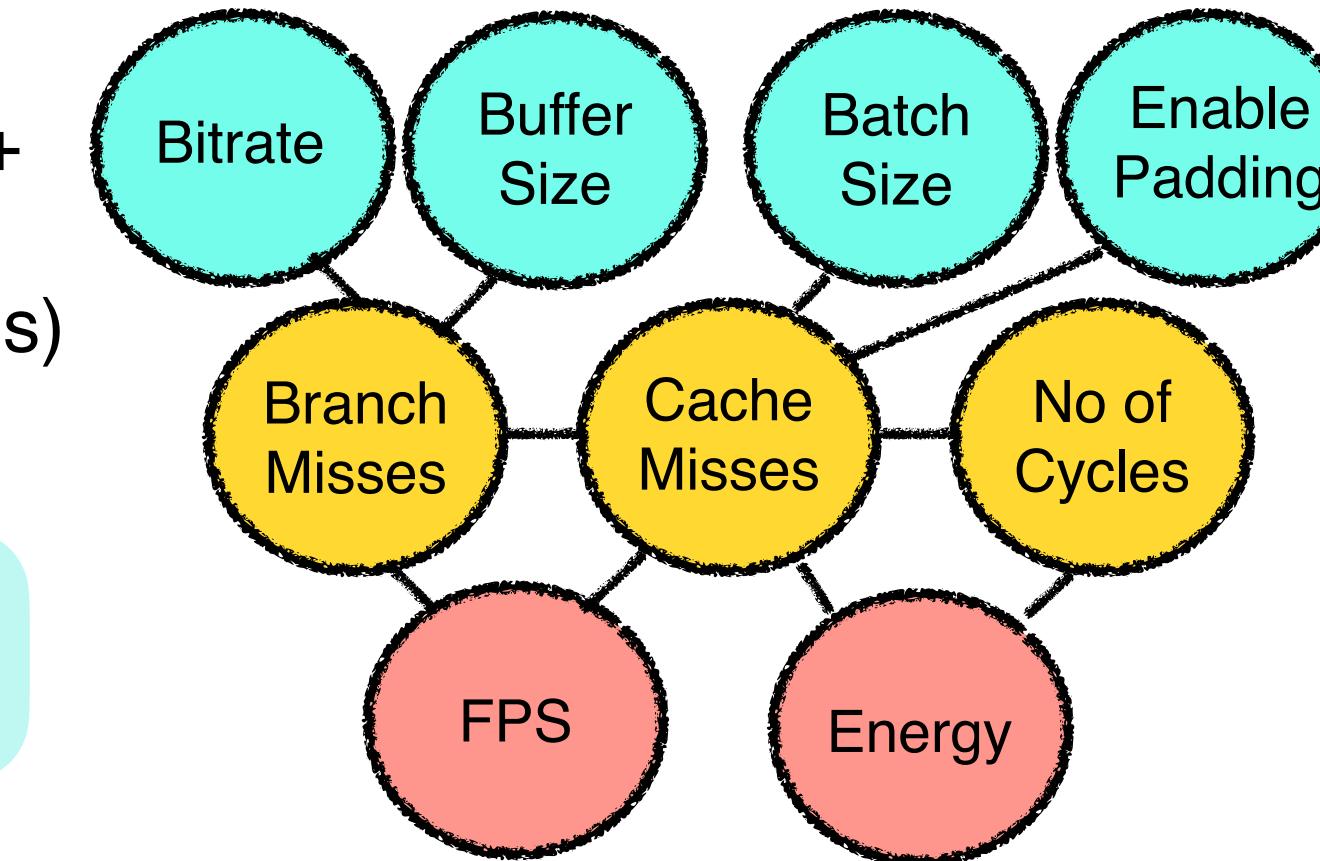
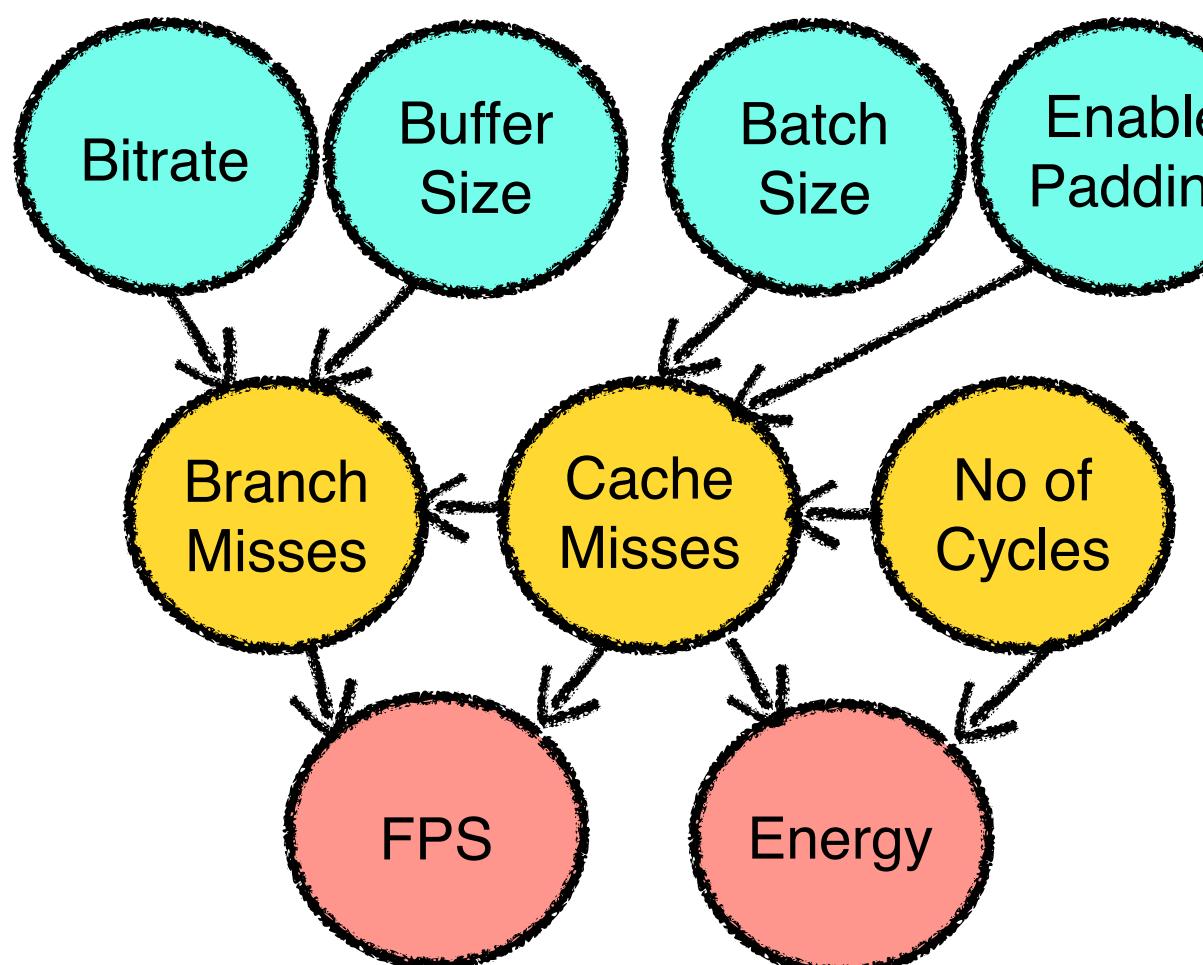


statistical independence tests

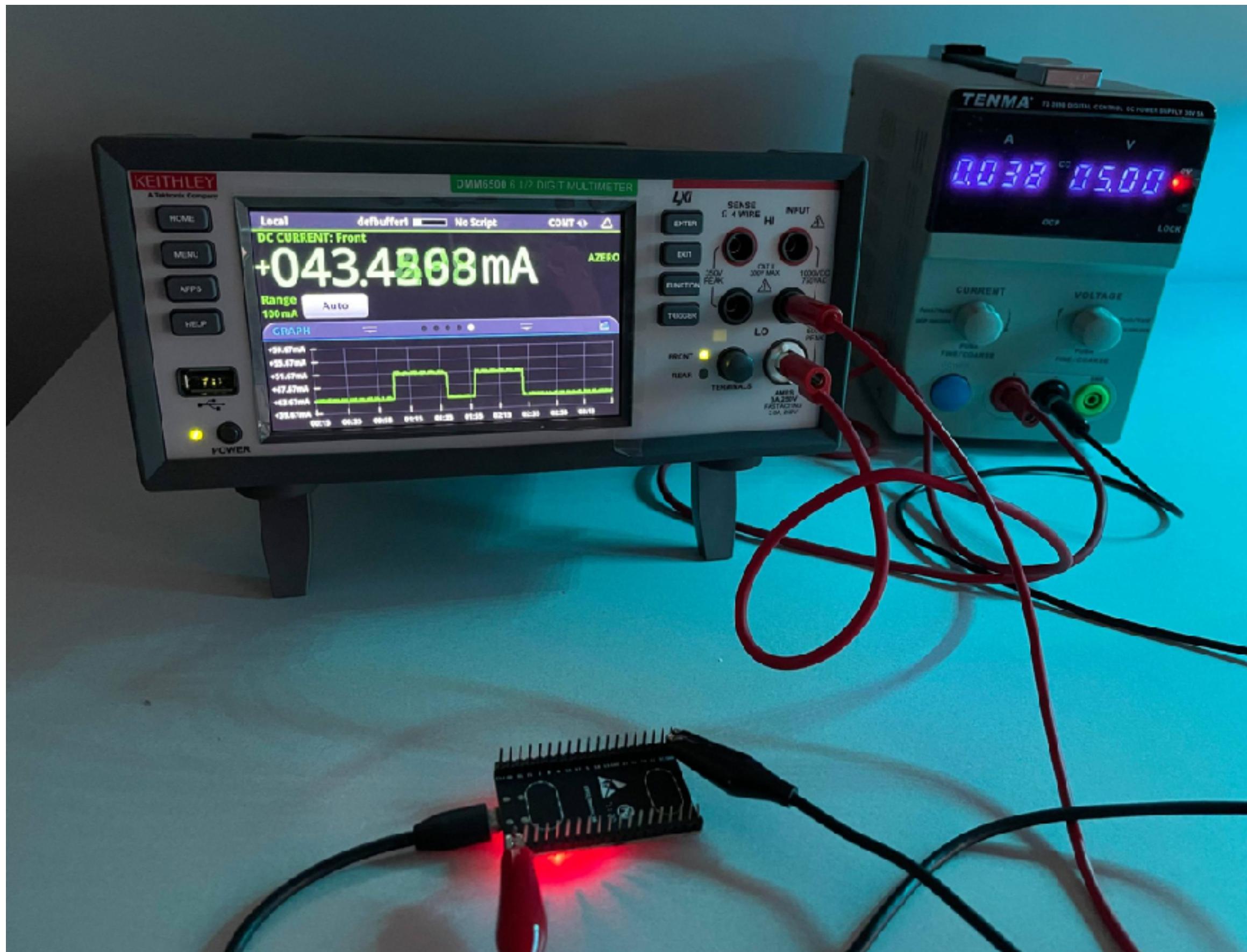
2- Pruning Causal Structure

orientation rules & measures (entropy) + structural constraints (colliders, v-structures)

3- Orienting Causal Relations



Our setup for performance measurements



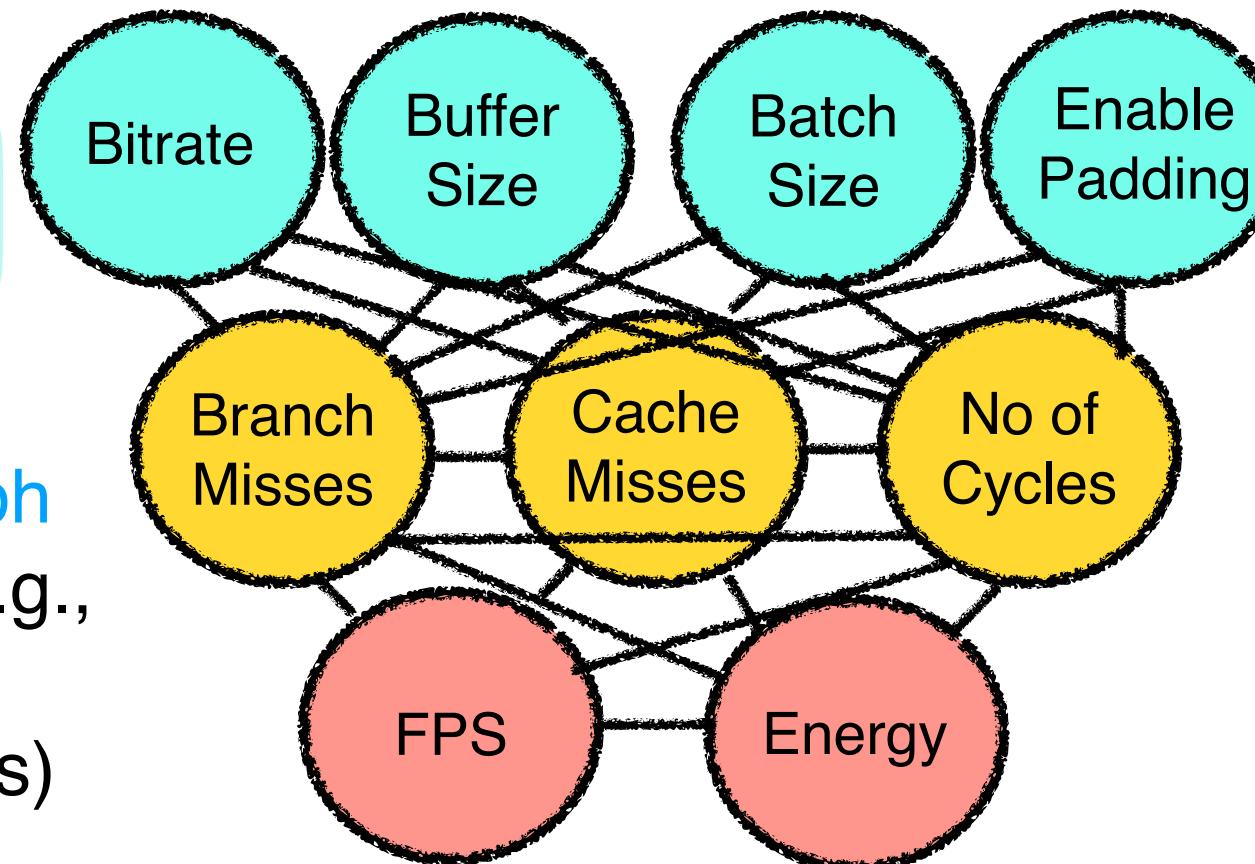
Learning Causal Performance Model

	Bitrate (bits/s)	Enable Padding	...	Cache Misses	...	Through put (fps)
c ₁	1k	1	...	42m	...	7
c ₂	2k	1	...	32m	...	22
...
c _n	5k	0	...	12m	...	25

1- Recovering the Skelton



fully connected graph given constraints (e.g., no connections btw configuration options)



statistical

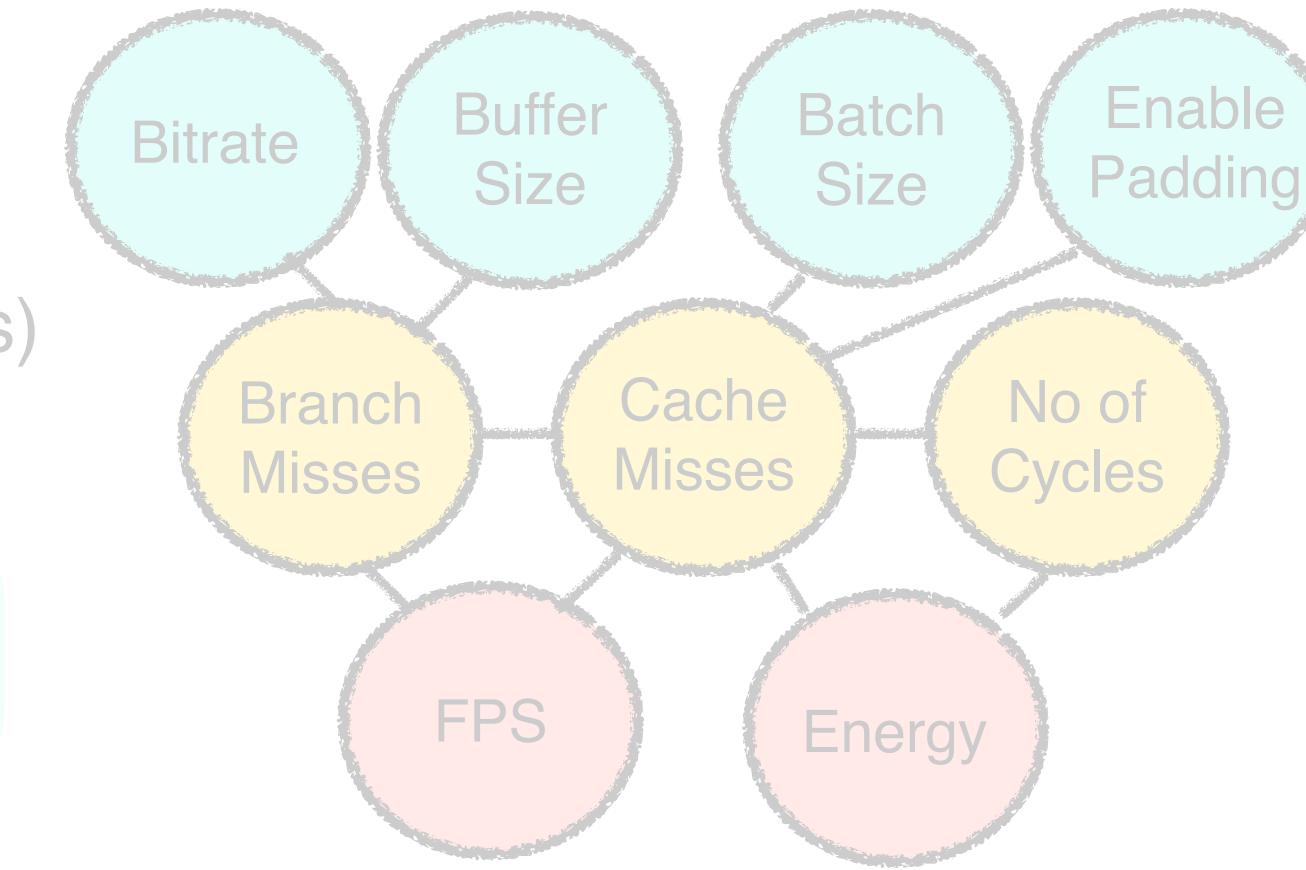
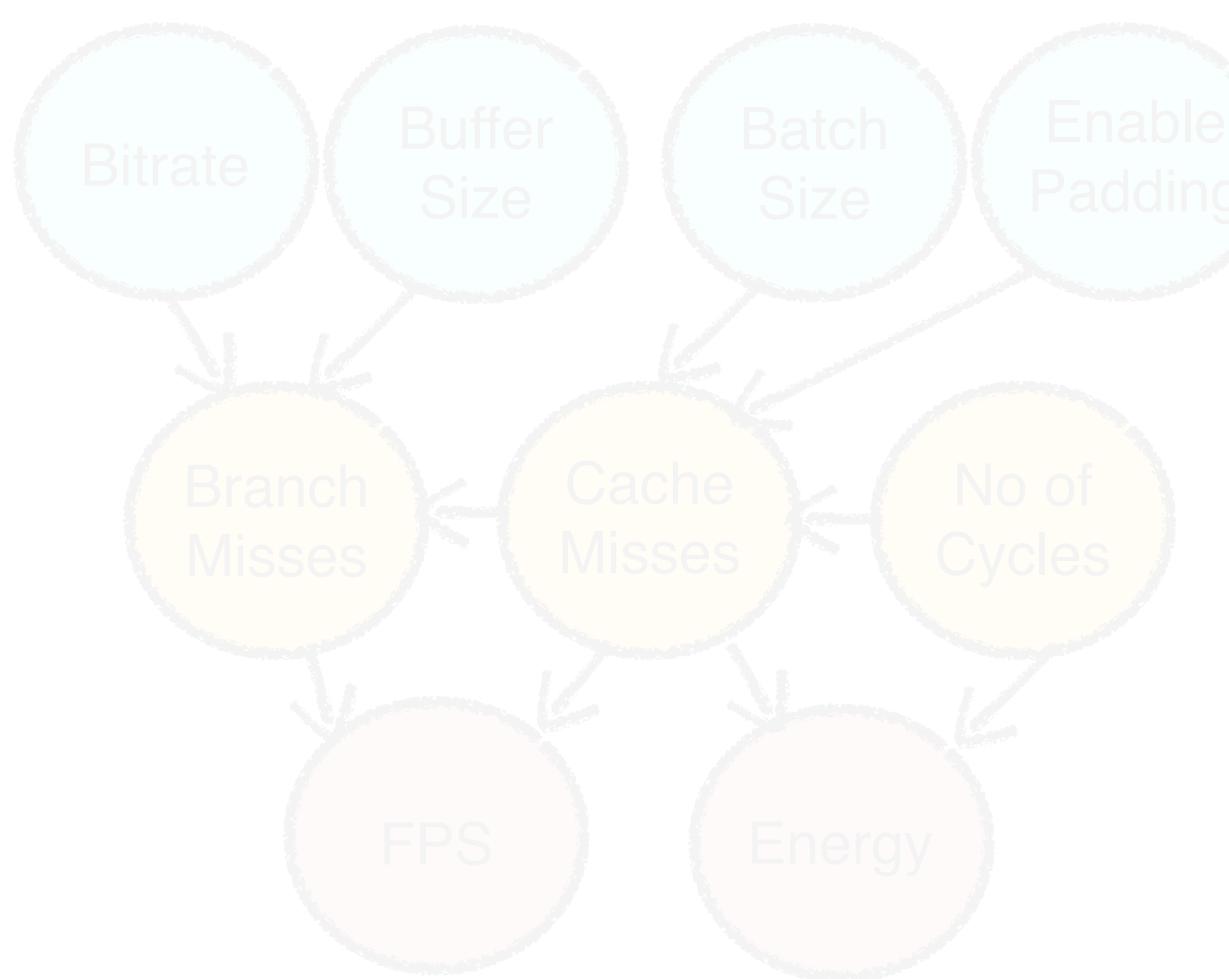
independence

tests

2- Pruning
Causal Structure

orientation rules &
measures (entropy) +
structural constraints
(colliders, v-structures)

3- Orienting
Causal Relations



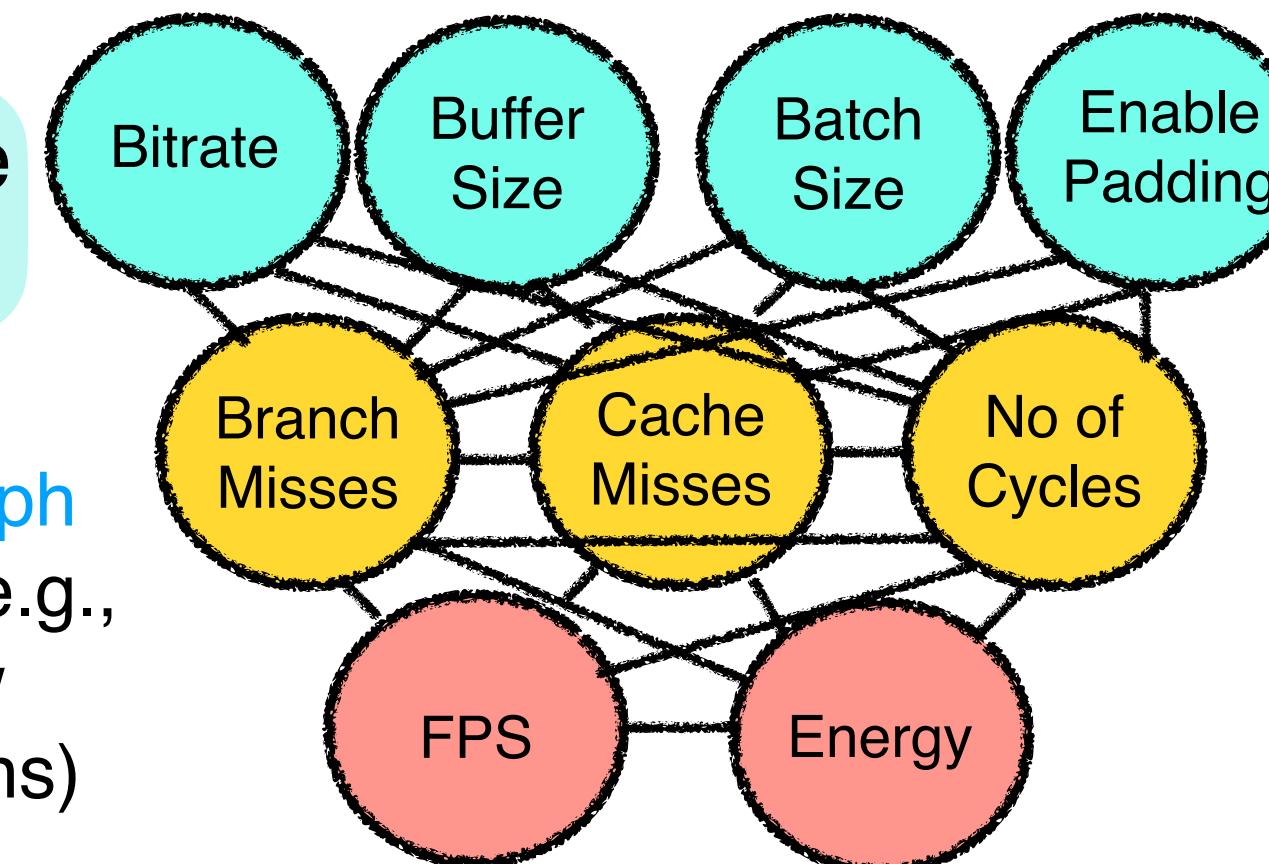
Learning Causal Performance Model

	Bitrate (bits/s)	Enable Padding	...	Cache Misses	...	Through put (fps)
c ₁	1k	1	...	42m	...	7
c ₂	2k	1	...	32m	...	22
...
c _n	5k	0	...	12m	...	25

1- Recovering the Skelton



fully connected graph given constraints (e.g., no connections btw configuration options)

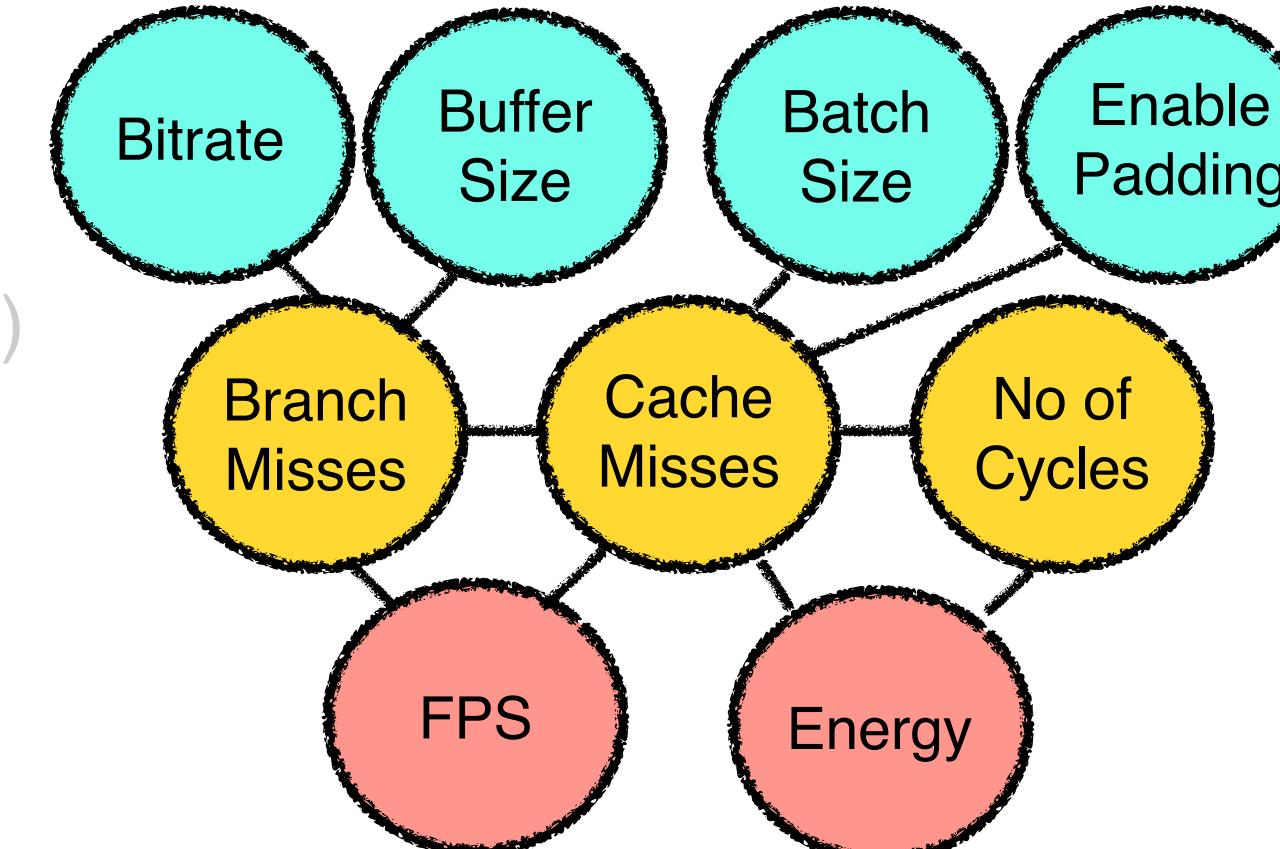


statistical independence

2- Pruning Causal Structure

orientation rules & measures (entropy) + structural constraints (colliders, v-structures)

3- Orienting Causal Relations



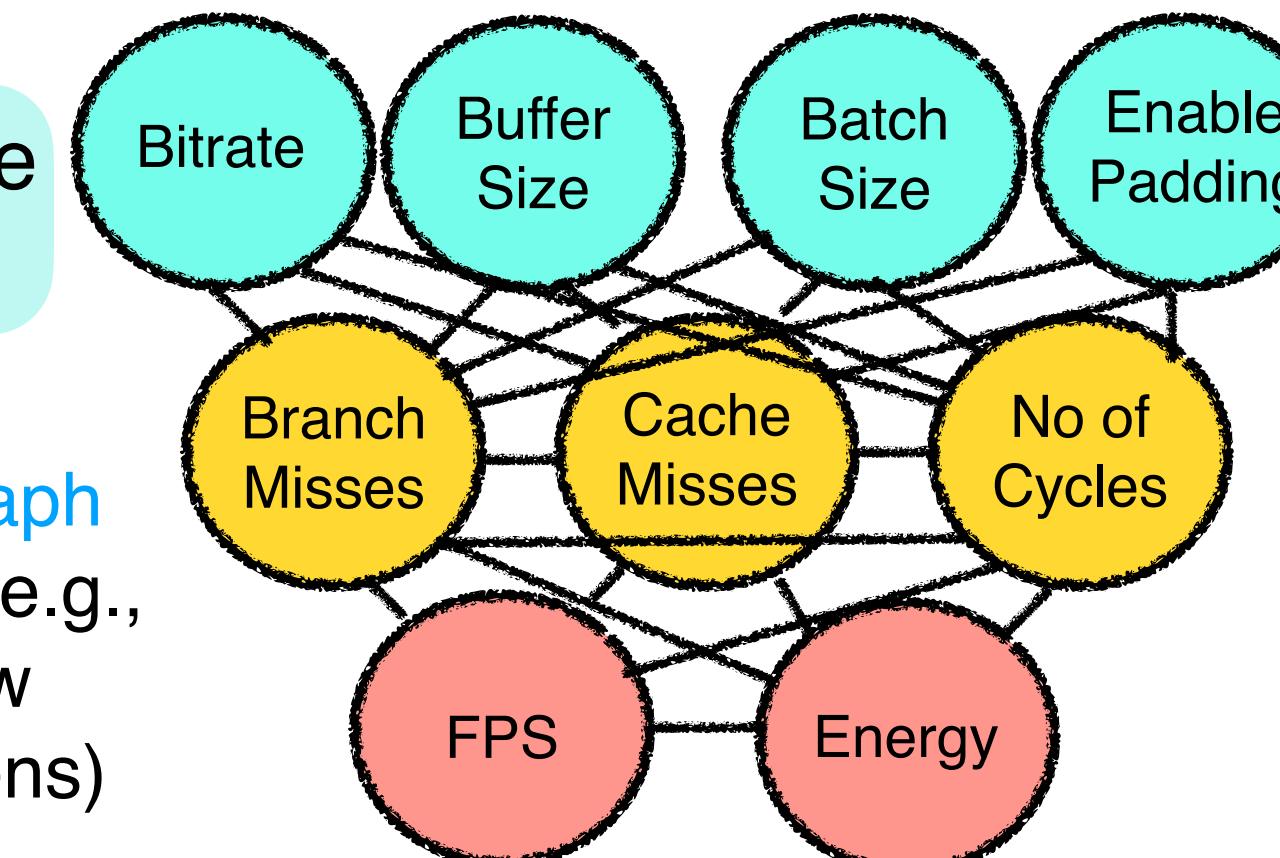
Learning Causal Performance Model

	Bitrate (bits/s)	Enable Padding	...	Cache Misses	...	Through put (fps)
c ₁	1k	1	...	42m	...	7
c ₂	2k	1	...	32m	...	22
...
c _n	5k	0	...	12m	...	25

1- Recovering the Skelton



fully connected graph given constraints (e.g., no connections btw configuration options)



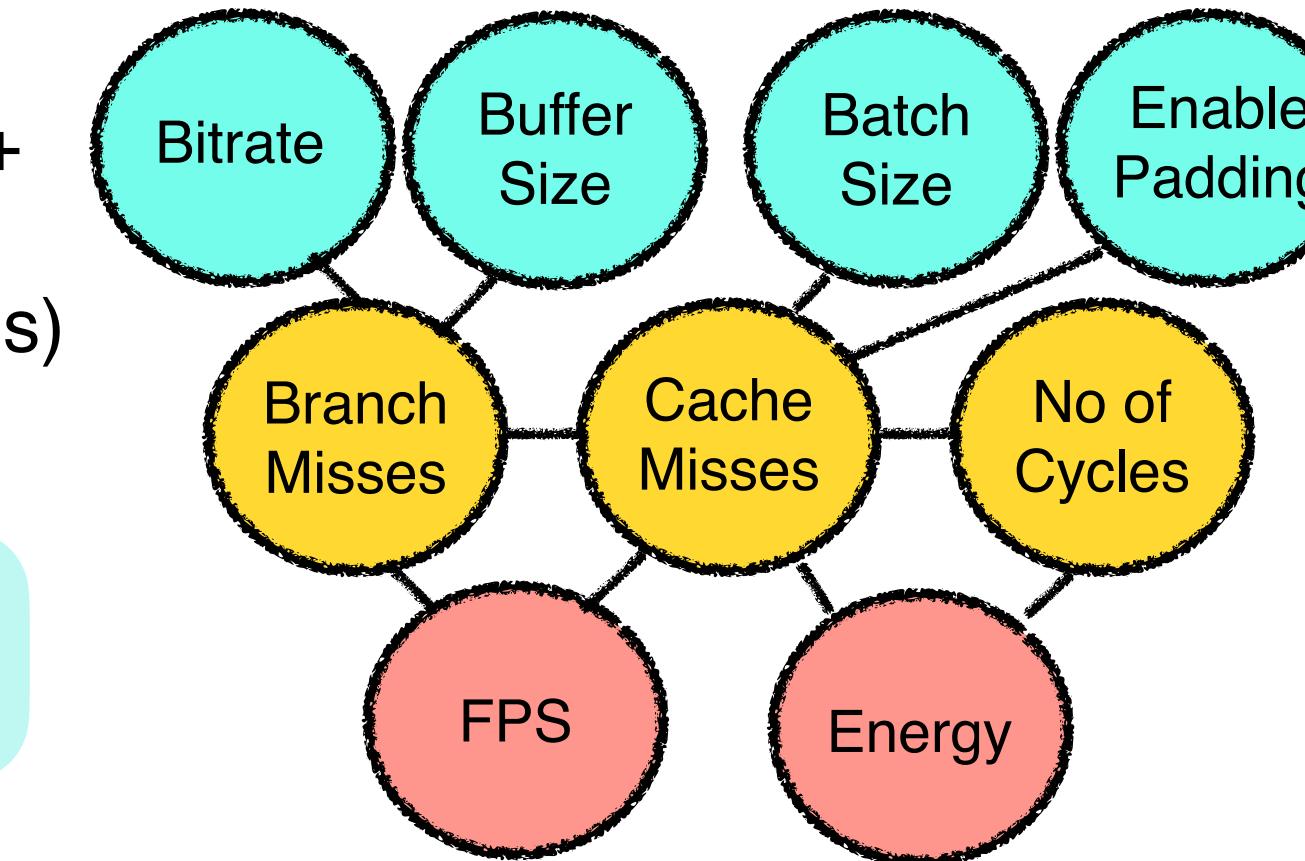
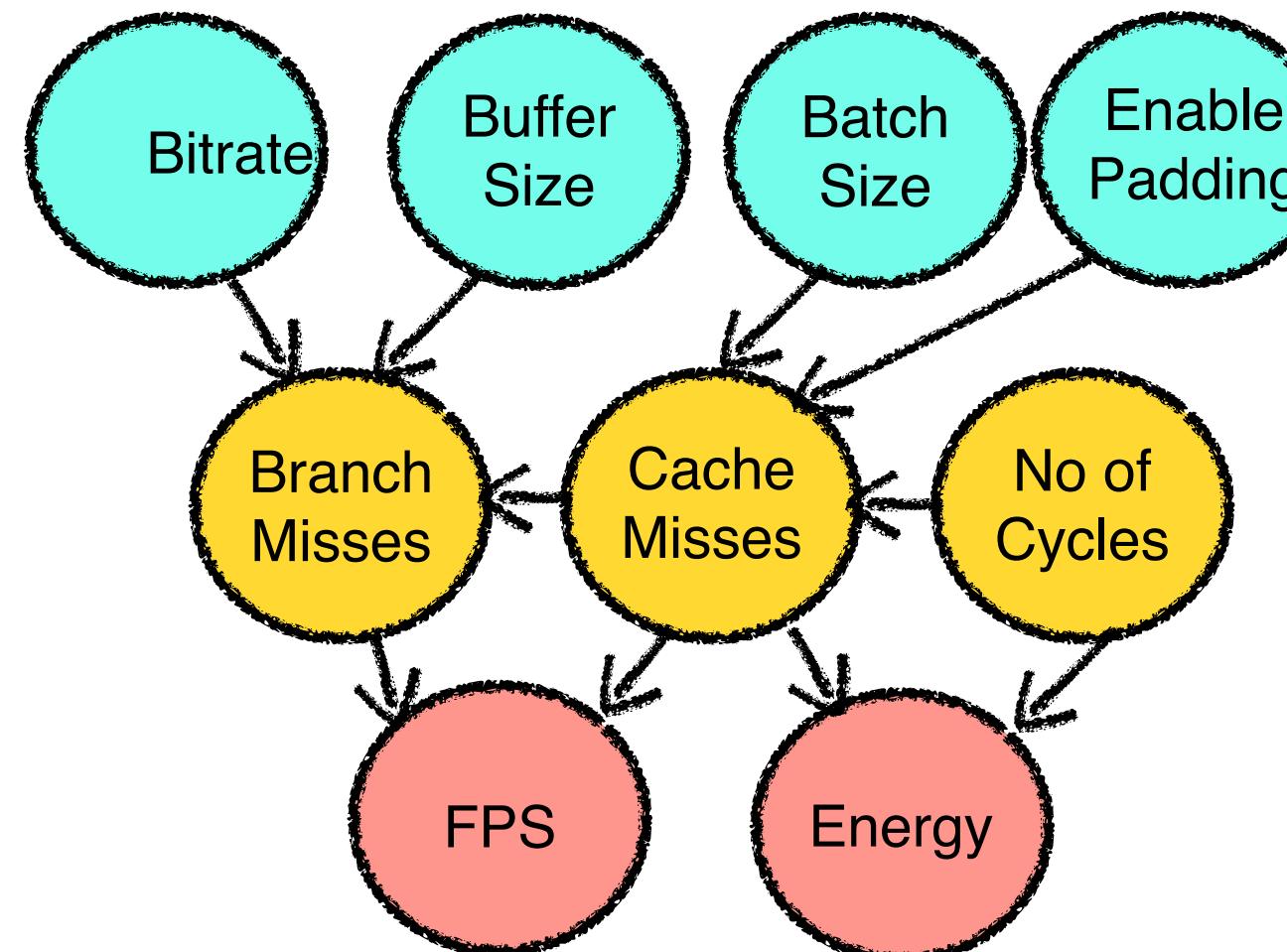
statistical independence tests

2- Pruning Causal Structure

orientation rules & measures (entropy) + structural constraints (colliders, v-structures)

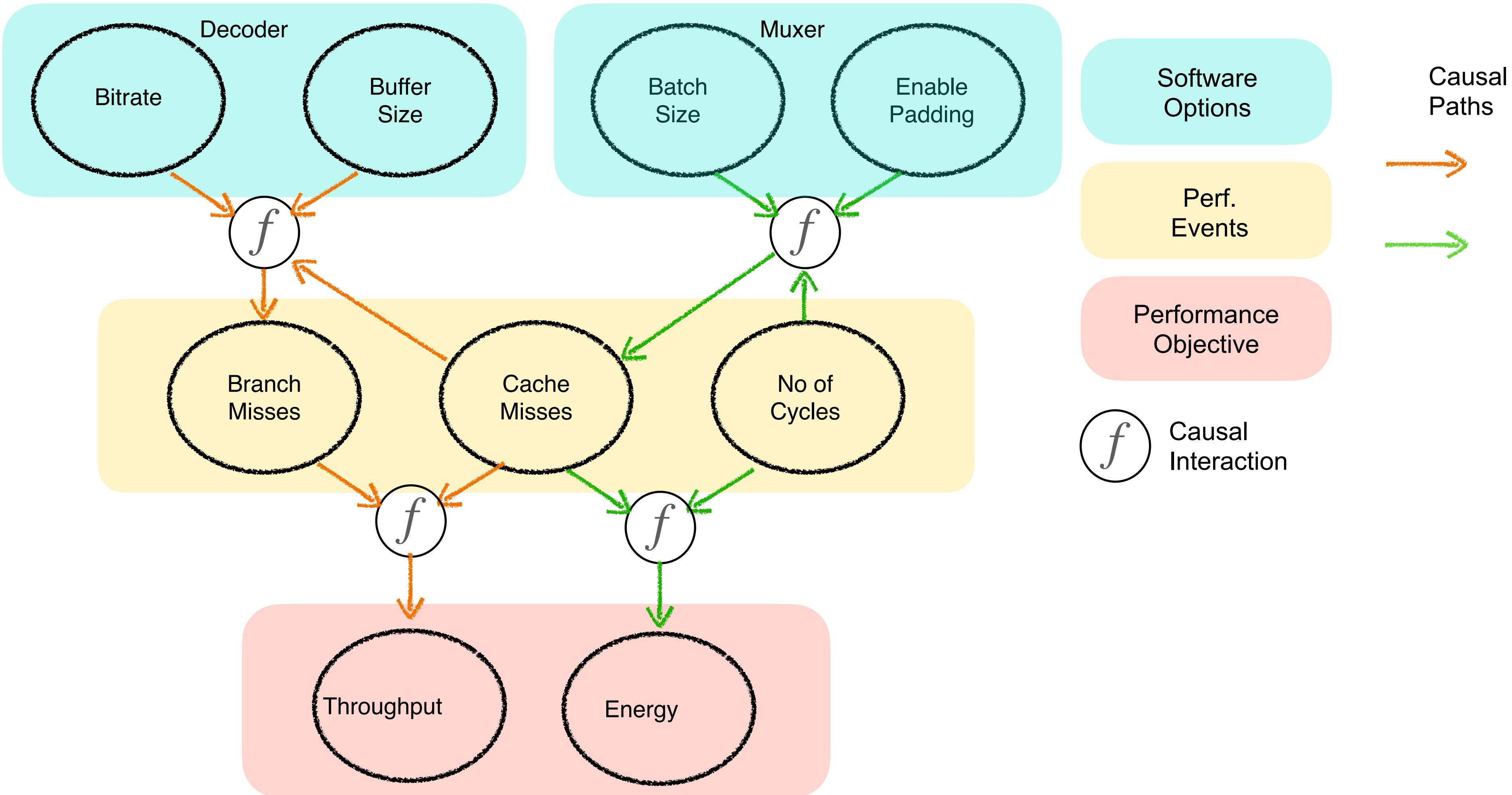


3- Orienting Causal Relations

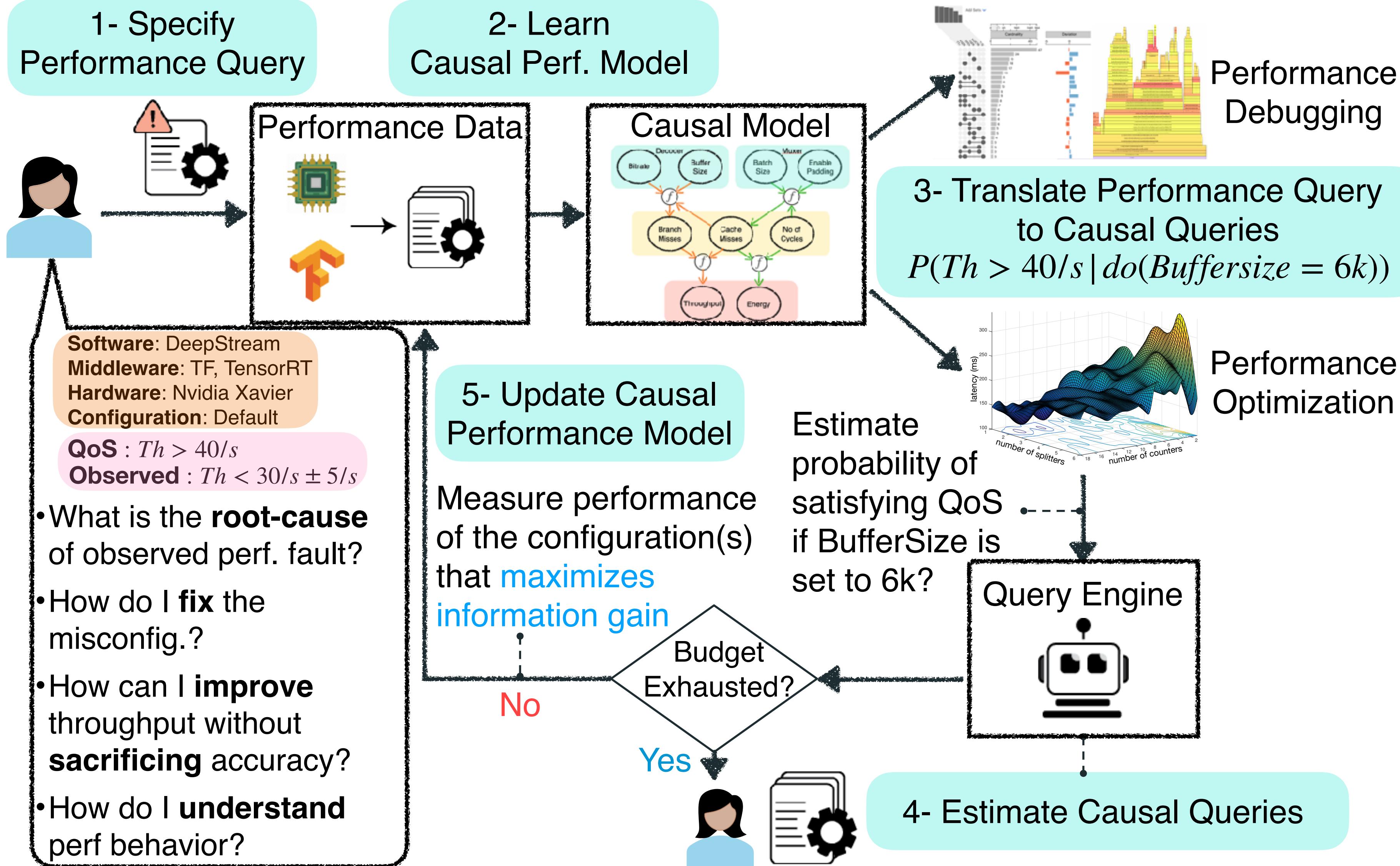


Causal Performance Model

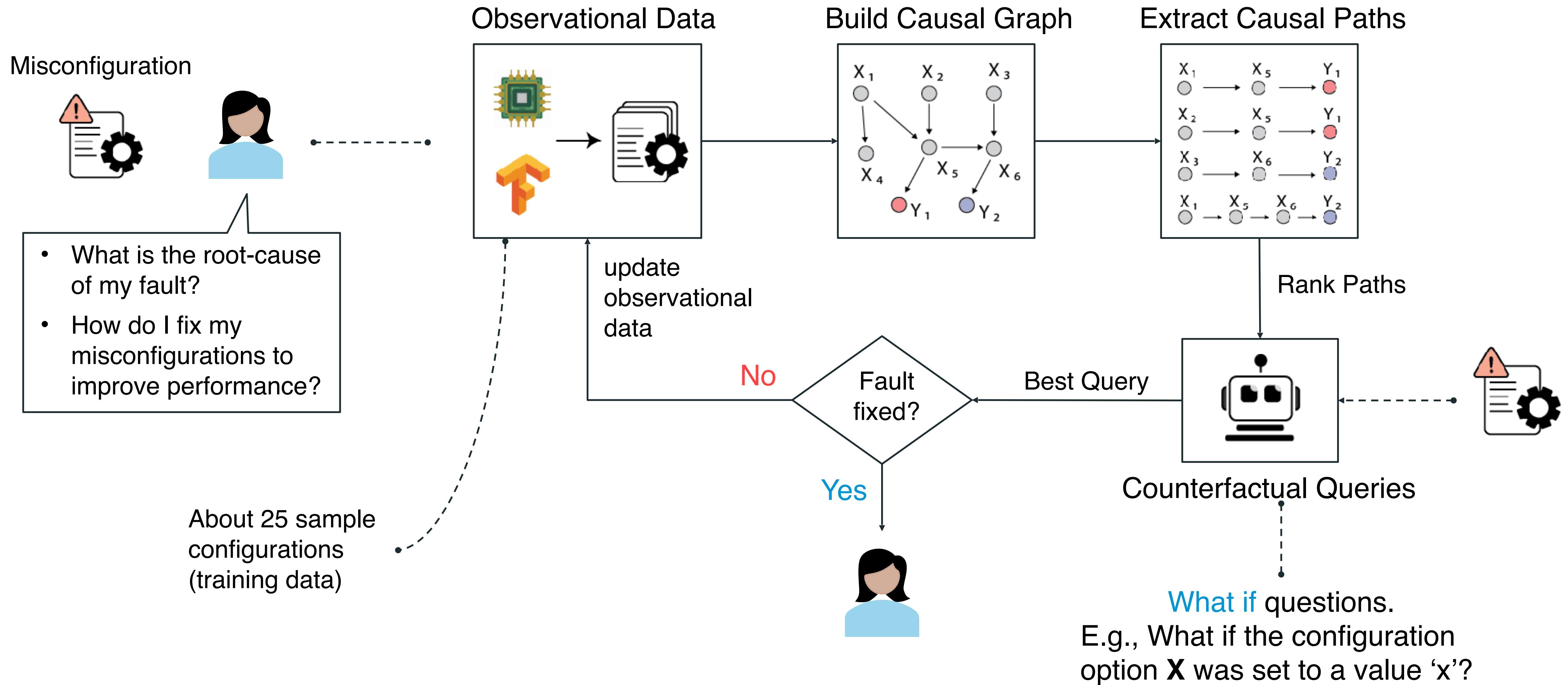
$$\text{Branchmisses} = 2 \times \text{Bitrate} + 8.1 \times \text{Buffer size} + 4.1 \times \text{Bitrate} \times \text{Buffer size} \times \text{Cache misses}$$



UNICORN: Our Causal AI for Systems Method



Causal Debugging



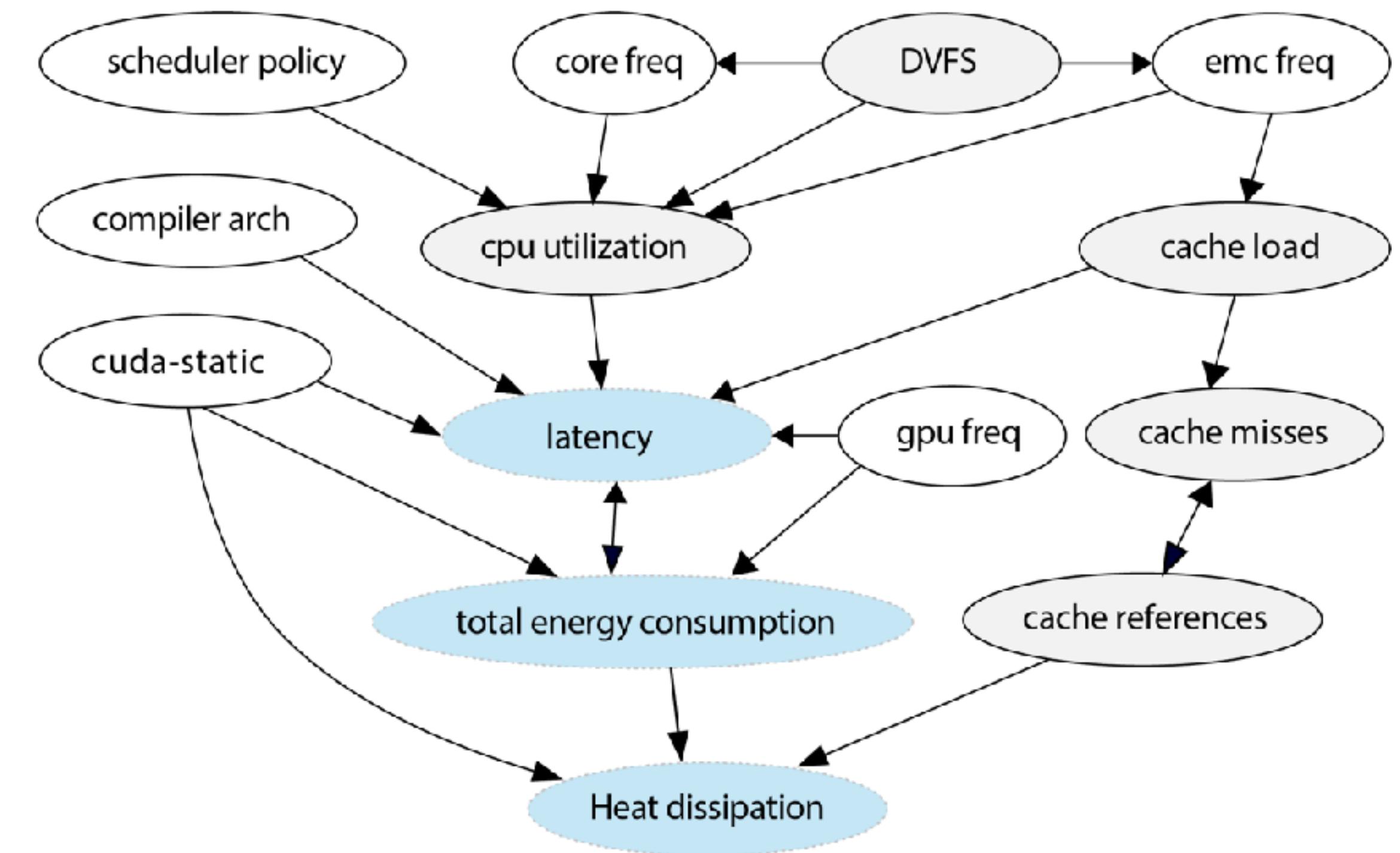
Extracting Causal Paths from the Causal Model

Problem

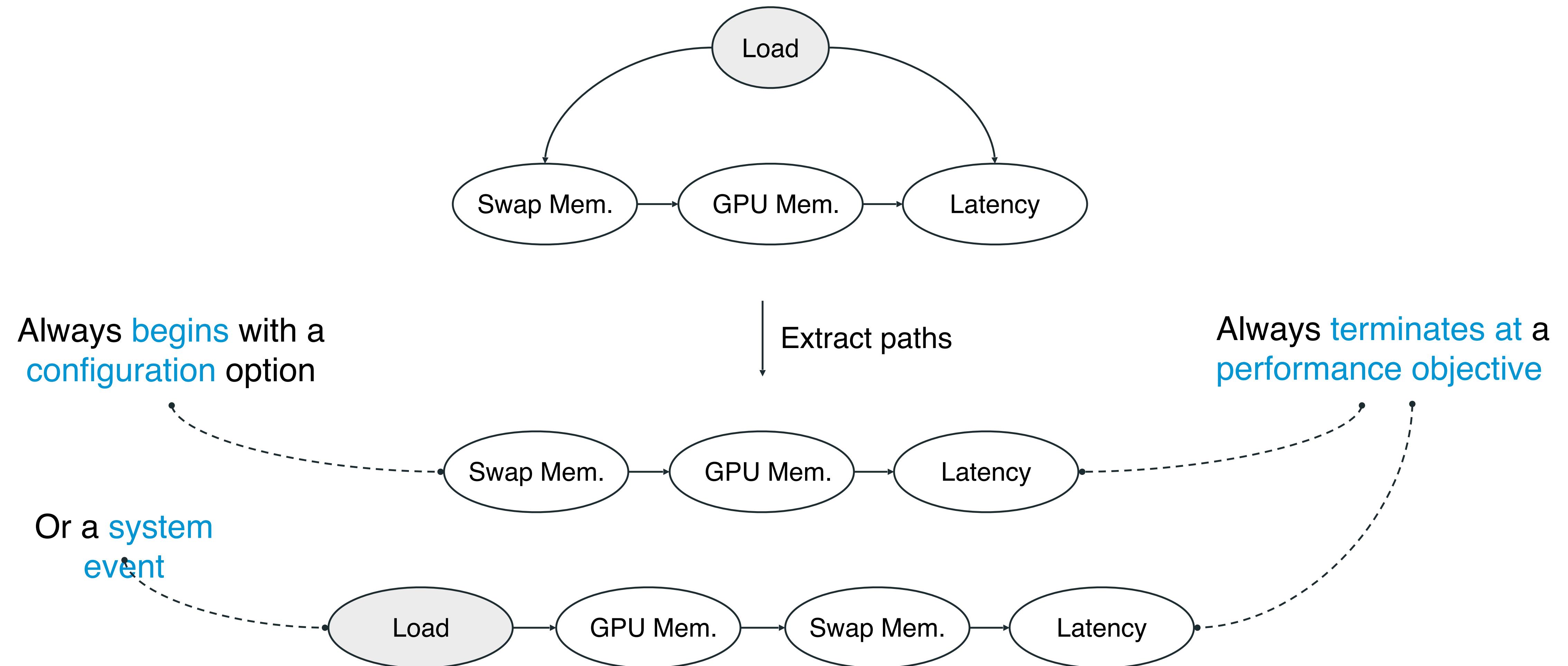
- ✗ In real world cases, this causal graph can be **very complex**
- ✗ It may be **intractable** to reason over the entire graph directly

Solution

- ✓ Extract paths from the causal graph
- ✓ Rank them based on their **Average Causal Effect** on latency, etc.
- ✓ Reason over the **top K** paths

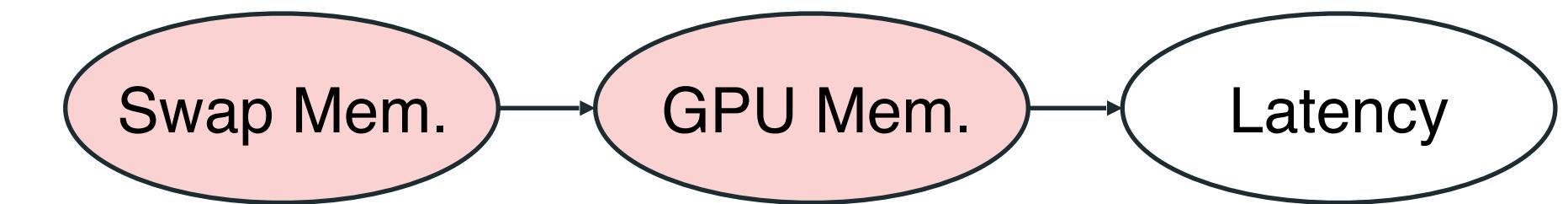


Extracting Causal Paths from the Causal Model



Ranking Causal Paths from the Causal Model

- They may be **too many** causal paths
- We need to select the **most useful ones**
- Compute the **Average Causal Effect (ACE)** of each pair of neighbors in a path



$$ACE(\text{GPU Mem.}, \text{Swap}) = \frac{1}{N} \sum_{a,b \in Z} \mathbb{E}\left(\text{GPU Mem.} \mid do(\text{Swap} = b)\right) - \mathbb{E}\left(\text{GPU Mem.} \mid do(\text{Swap} = a)\right)$$

Average over all permitted values of Swap memory.

Expected value of GPU Mem. when we **artificially intervene** by setting Swap to the value **b**

If this difference is large, then **small changes to Swap Mem. will cause large changes to GPU Mem.**

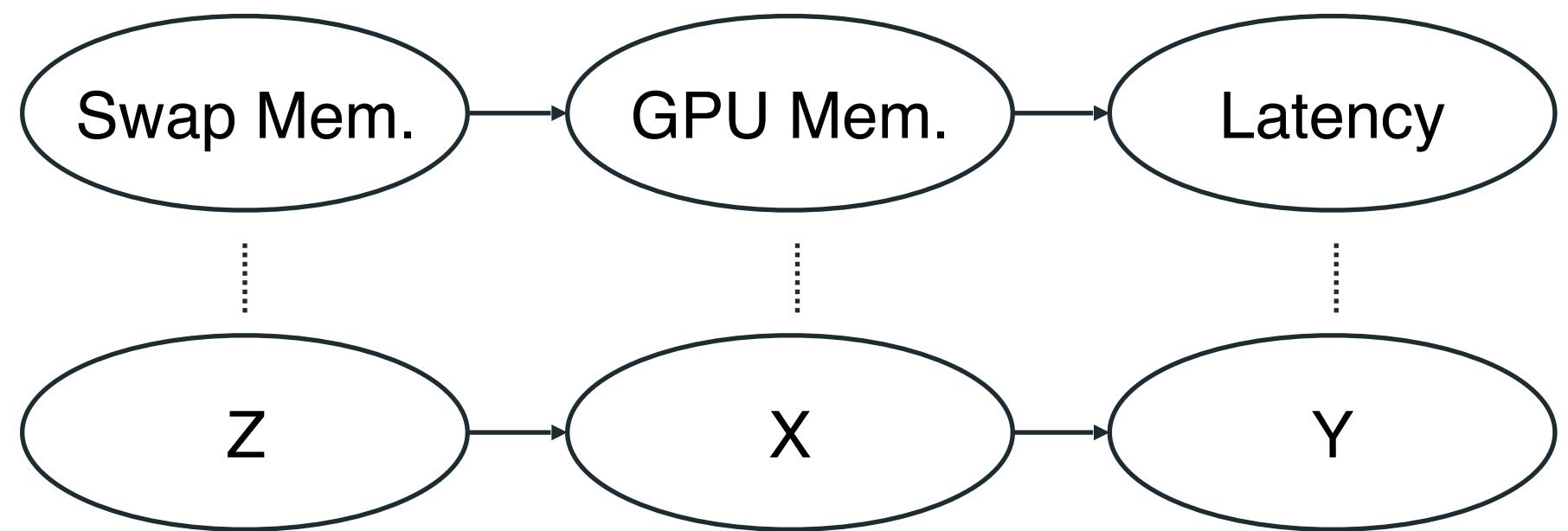
Expected value of GPU Mem. when we **artificially intervene** by setting Swap to the value **a**

Ranking Causal Paths from the Causal Model

- Average the ACE of **all pairs of adjacent nodes** in the path

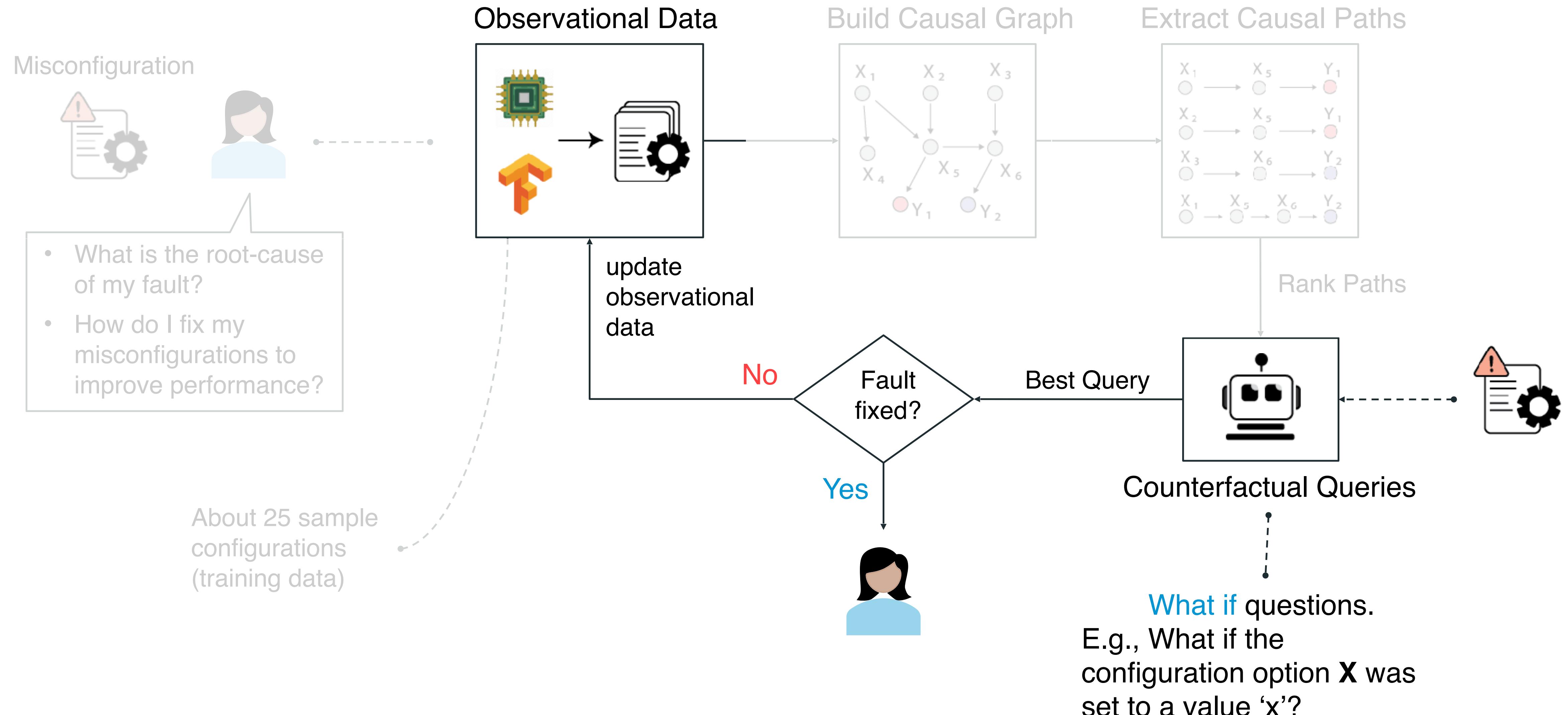
$$PACE(Z, Y) = \frac{1}{2}(ACE(Z, X) + ACE(X, Y))$$

Sum over all pairs of nodes in the causal path.



- Rank paths from **highest path ACE (PACE) score** to the lowest
- Use the **top K paths** for subsequent analysis

Diagnosing and Fixing the Faults



Diagnosing and Fixing the Faults

- Counterfactual inference asks “what if” questions about changes to the misconfigurations



Example

“Given that my current swap memory is 2 Gb, and I have high latency. What is the probability of having low latency if swap memory was increased to 4 Gb?”

⋮

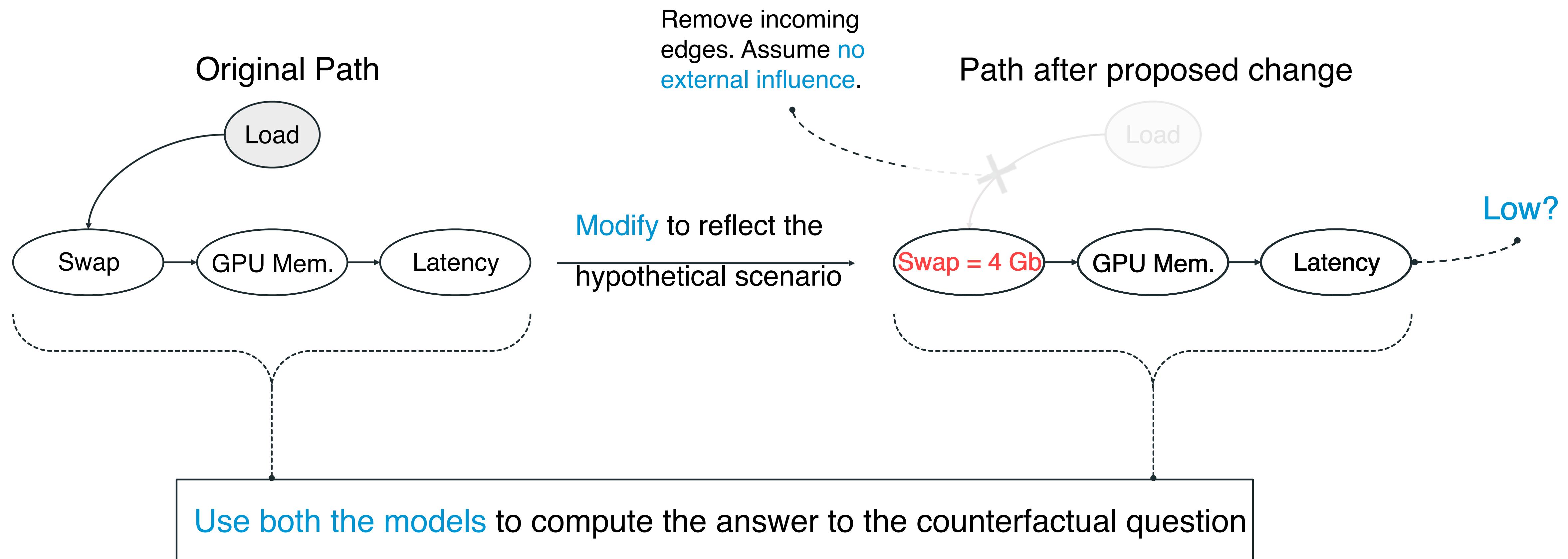
We are interested in the scenario where:

- We hypothetically have low latency;

Conditioned on the following events:

- We hypothetically set the new Swap memory to 4 Gb
- Swap Memory was initially set to 2 Gb
- We observed high latency when Swap was set to 2 Gb
- Everything else remains the same

Diagnosing and Fixing the Faults



Diagnosing and Fixing the Faults



$Potential = P(\hat{Latency} = \text{low} \mid \hat{Swap} = 4 \text{ Gb}, Swap = 2 \text{ Gb}, Latency_{swap=2Gb} = \text{high}, U)$

We expect a low latency

The Swap was initially 2 Gb

The Swap is now 4 Gb

The latency was high

Everything else stays the same

Diagnosing and Fixing the Faults

$$\text{Potential} = P\left(\text{outcome} = \text{good} \mid \text{change}, \text{outcome}_{\neg\text{change}} = \text{bad}, \neg\text{change}, U \right)$$

Probability that the **outcome is good** after a change, conditioned on the past

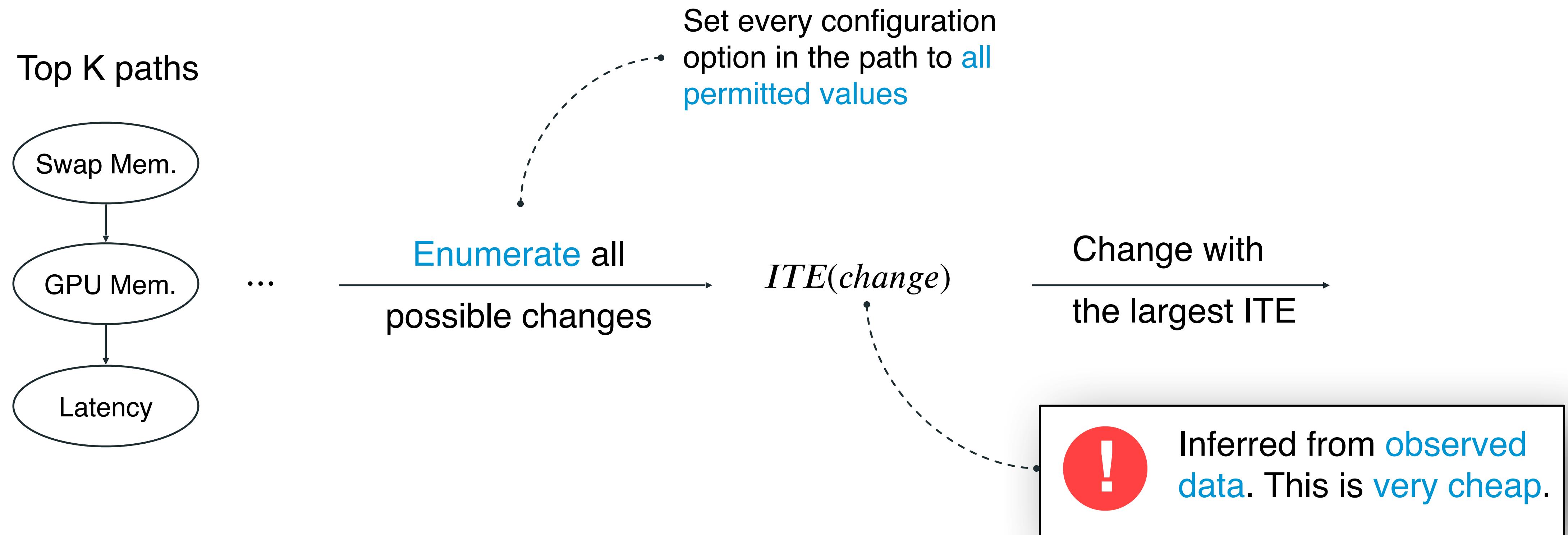
$$\text{Control} = P\left(\text{outcome} = \text{bad} \mid \neg\text{change}, U \right)$$

Probability that the **outcome was bad** before the change

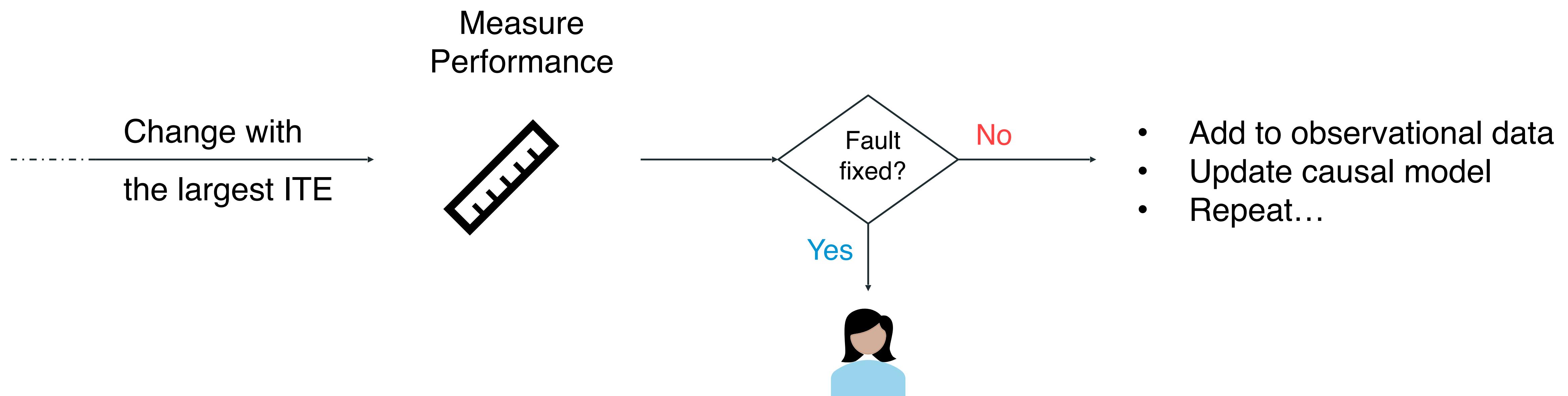
Individual Treatment Effect = Potential – Outcome

If this difference is **large**, then our change is **useful**

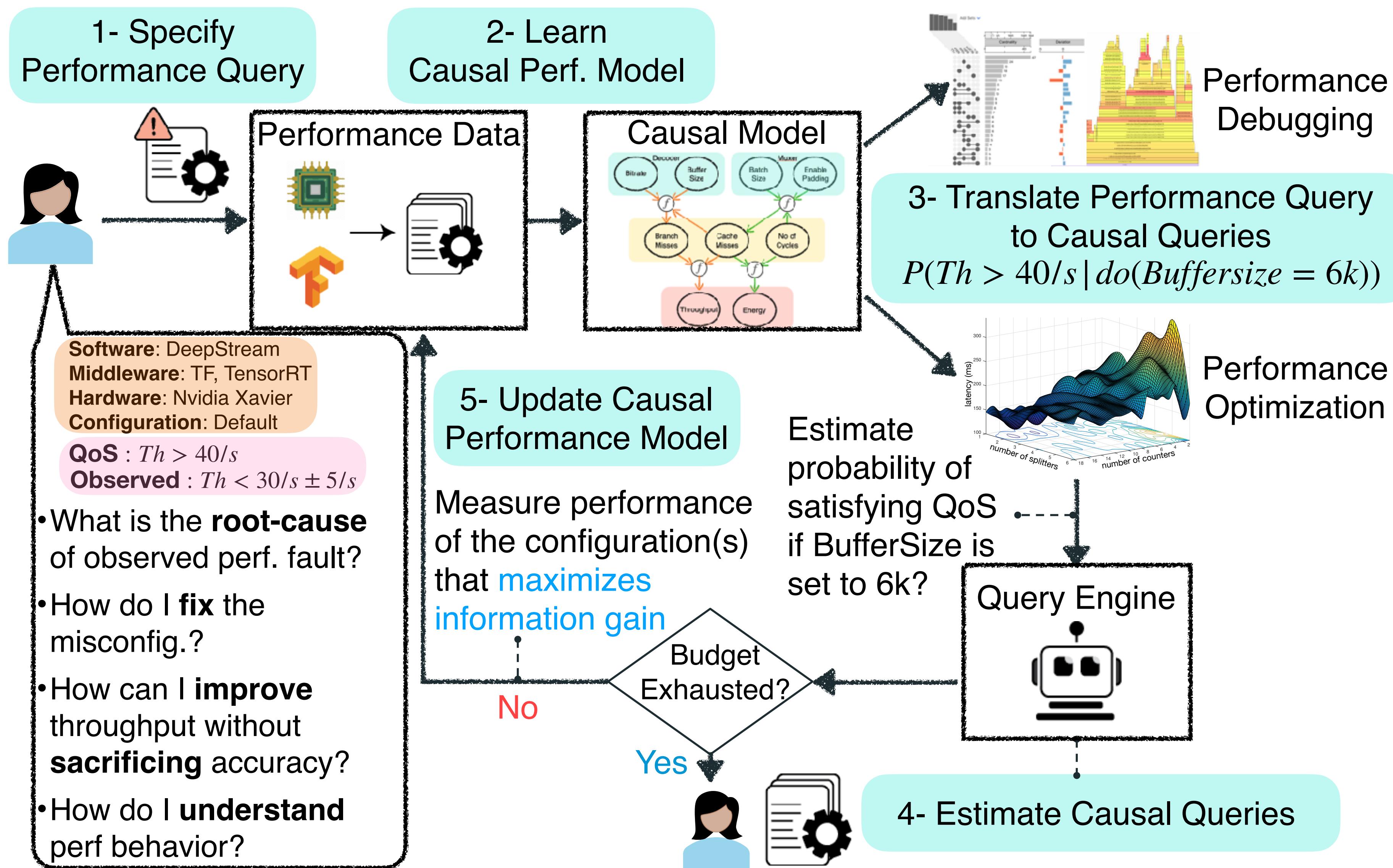
Diagnosing and Fixing the Faults



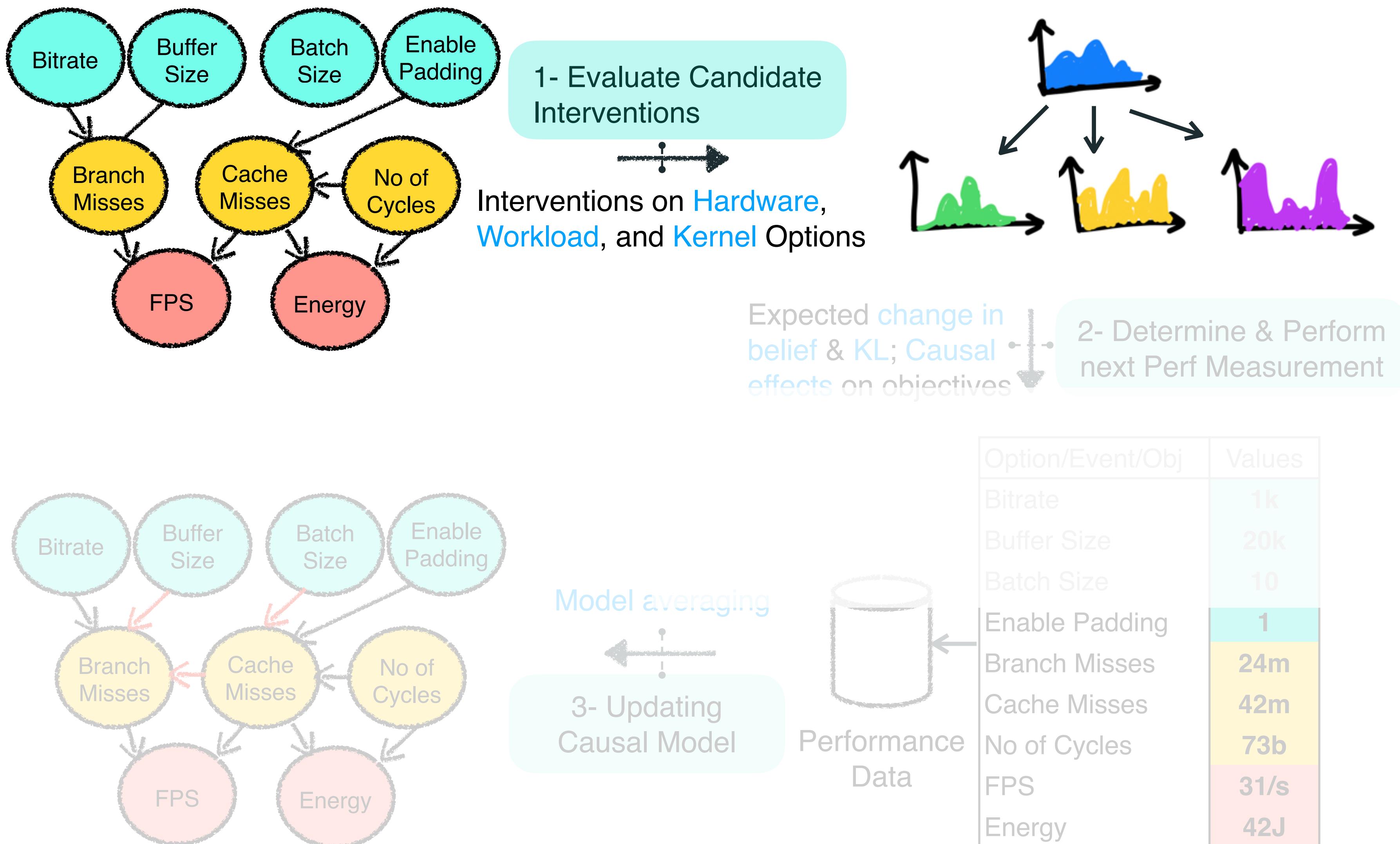
Diagnosing and Fixing the Faults



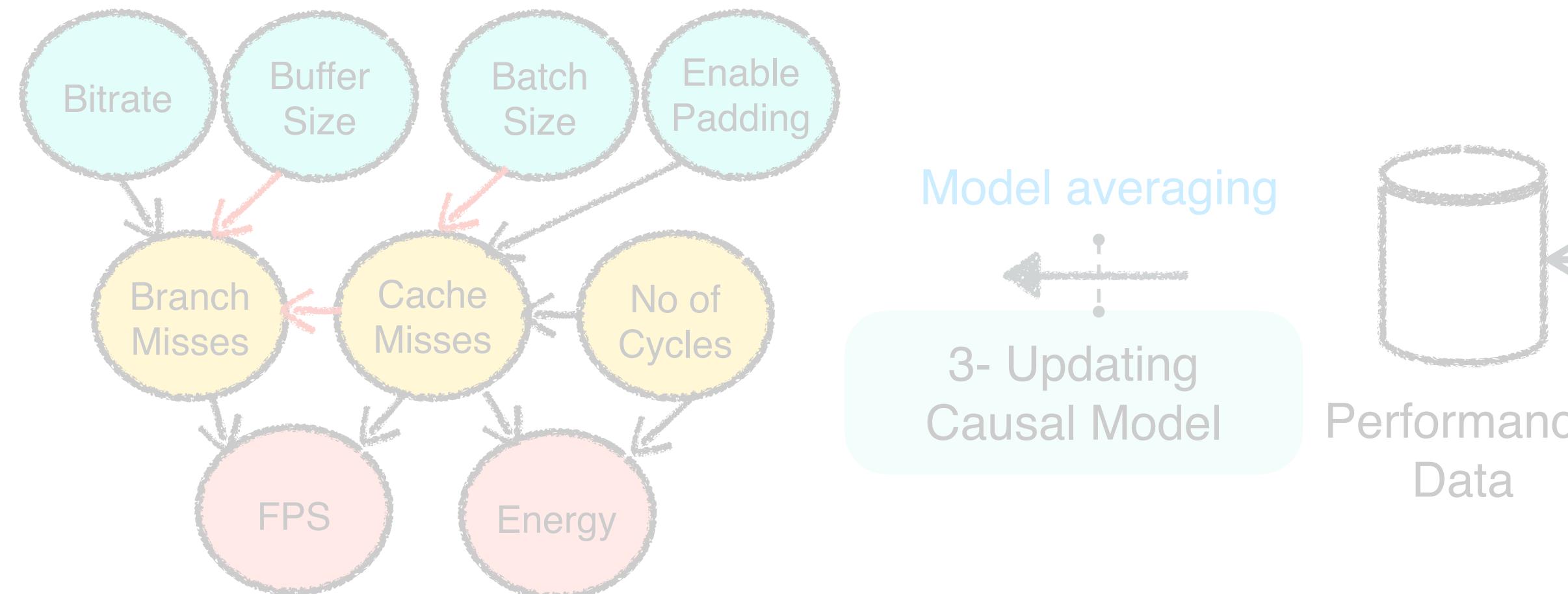
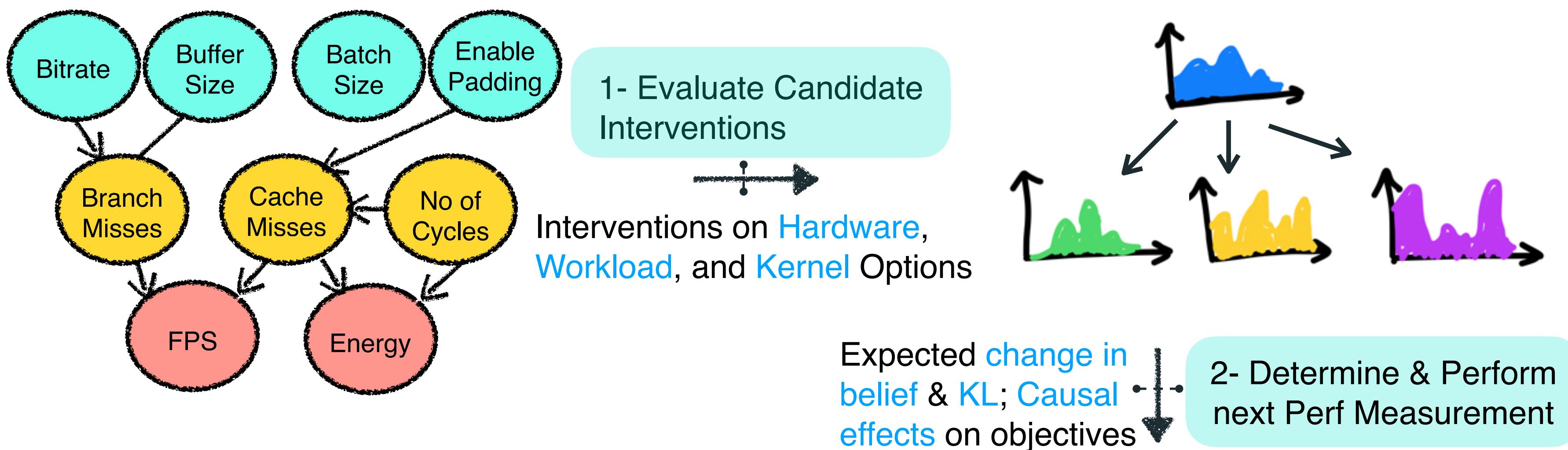
UNICORN: Our Causal AI for Systems Method



Active Learning for Updating Causal Performance Model

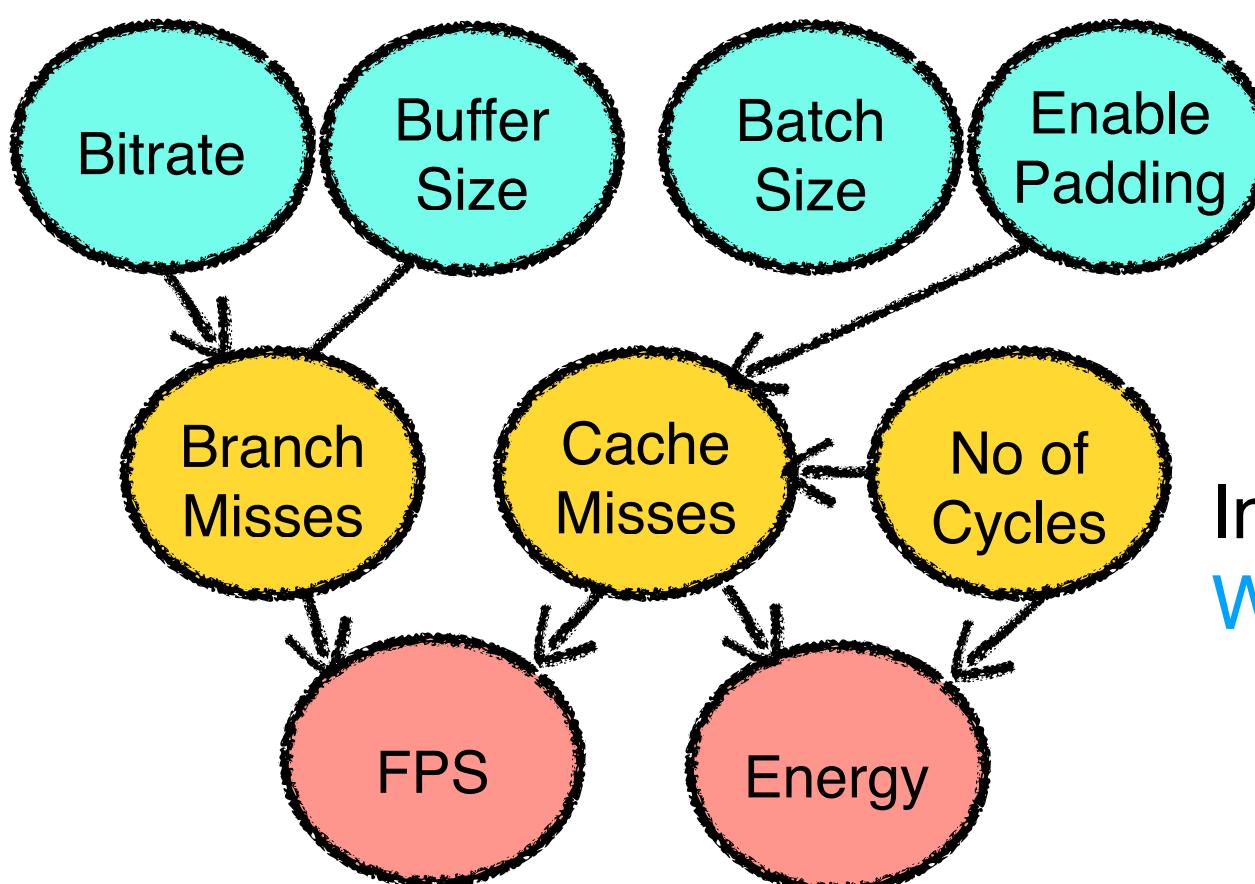


Active Learning for Updating Causal Performance Model



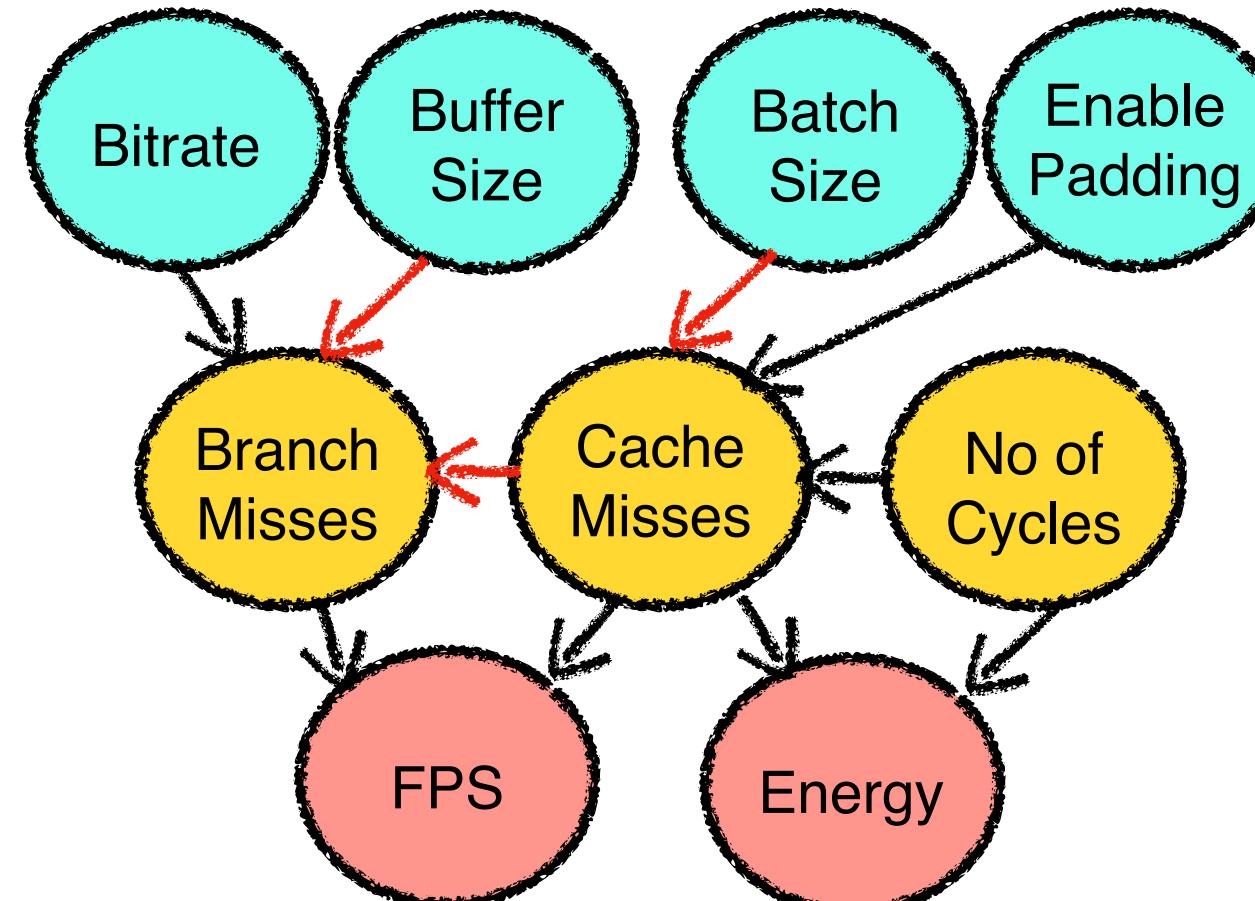
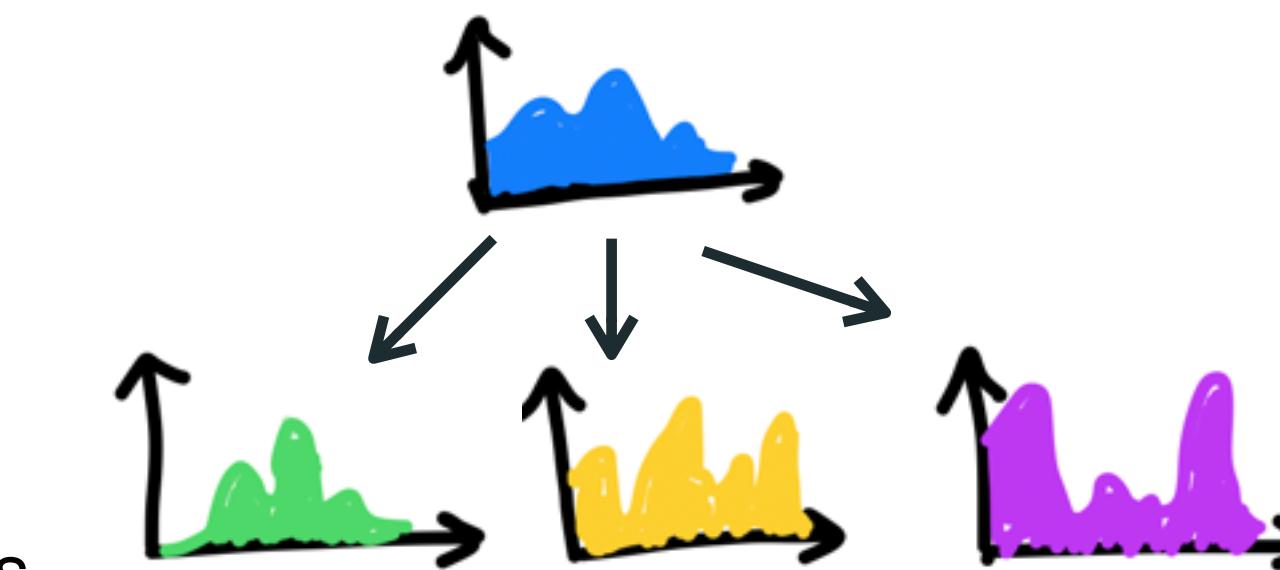
Option/Event/Obj	Values
Bitrate	1k
Buffer Size	20k
Batch Size	10
Enable Padding	1
Branch Misses	24m
Cache Misses	42m
No of Cycles	73b
FPS	31/s
Energy	42J

Active Learning for Updating Causal Performance Model



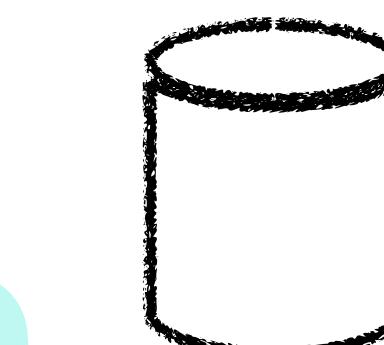
1- Evaluate Candidate Interventions

Interventions on **Hardware**,
Workload, and **Kernel Options**



Model averaging

3- Updating
Causal Model



Performance
Data

Option/Event/Obj	Values
Bitrate	1k
Buffer Size	20k
Batch Size	10
Enable Padding	1
Branch Misses	24m
Cache Misses	42m
No of Cycles	73b
FPS	31/s
Energy	42J

Benefits of Causal Reasoning for System Performance Analysis



There are two fundamental benefits that we get by our “Causal AI for Systems” methodology

1. We learn one central (causal) performance model from the data across different performance tasks:

- Performance **understanding**
- Performance **optimization**
- Performance **debugging** and repair
- Performance **prediction** for different environments (e.g., canary-> production)

2. The causal model is transferable across environments.

- We observed Sparse Mechanism Shift in systems too!
- Alternative non-causal models (e.g., regression-based models for performance tasks) are not transferable as they rely on i.i.d. setting.



UNICORN: Reasoning about Configurable System Performance through the Lens of Causality

Md Shahriar Iqbal
University of South Carolina
miqbal@email.sc.edu

Rahul Krishna
IBM Research
rkrsn@ibm.com

Mohammad Ali Javidian
Purdue University
mjavidia@purdue.edu

Baishakhi Ray
Columbia University
rayb@cs.columbia.edu

Pooyan Jamshidi
University of South Carolina
pjamshid@cse.sc.edu

Abstract

Modern computer systems are highly configurable, with the total variability space sometimes larger than the number of atoms in the universe. Understanding and reasoning about the performance behavior of highly configurable systems, over a vast and variable space, is challenging. State-of-the-art methods for performance modeling and analyses rely on predictive machine learning models, therefore, they become (i) *unreliable in unseen environments* (e.g., different hardware,

