CSCE 585: Machine Learning Systems

Lecture 7: Understanding and Explaining the Root Causes of (Performance) Faults in (ML) Systems with Causal AI

Pooyan Jamshidi

UNIVERSITY OF South Carolina

Reconciling Accuracy, Cost, and Latency of Inference Serving Systems

Pooyan Jamshidi

University of South Carolina

https://pooyanjamshidi.github.io/

Problem:

Multi-Objective Optimization with Known Constraints under Uncertainty

$$
\begin{aligned}
\max \quad & \alpha \cdot AA - (\beta \cdot RC + \gamma \cdot LC) \\
\text{subject to} \quad & \lambda \leq \sum_{m \in M} th_m(n_m), \\
& \lambda_m \leq th_m(n_m) \\
& p_m(n_m) \leq L, \forall m \in M, \\
& RC \leq B,
\end{aligned}
$$

Solutions:

Different Assumptions

InfAdapter [2023]: Autoscaling for **ML Inference**

IPA [2024]: Autoscaling for **ML Inference Pipeline**

Sponge [2024]: Autoscaling for ML Inference Pipeline **Dynamic SLO**

What is Causal AI?

Let’s start with a (fiction) story

- Zeus is a patient waiting for a heart transplant. On 1 January, he received a new heart. Five days later, he died.
- Imagine that we can somehow know, that had Zeus not received a heart transplant on 1 January then he would have been alive five days later.
  - All others things in his life being unchanged.
- Now, what do you think was the cause of Zeus’s death?!
- Most people would agree that the **transplant caused Zeus’ death**.
- The intervention had a causal effect.

Let’s start with a (fiction) story

- Hera, received a heart transplant on 1 January. Five days later she was alive.
- Again, imagine we can somehow know that had Hera not received the heart on 1 January then she would still have been alive five days later.
  - All others things in his life being unchanged.
- The transplant **did not have a causal effect on Hera’s five day survival**.

Let's collect some data!

Exposure variable A (1: exposed, 0: unexposed); Outcome variable Y (1: death, 0: survival)

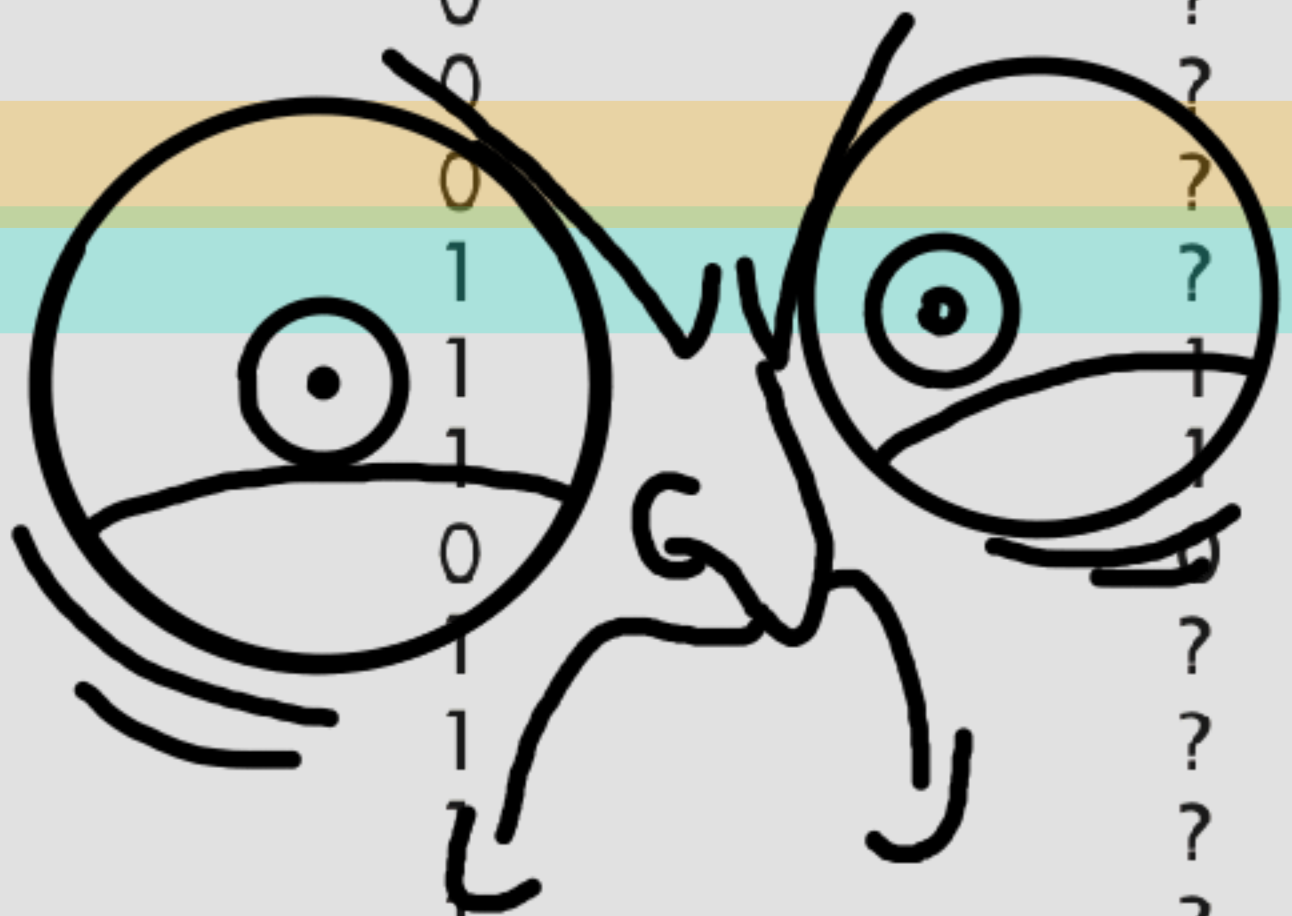
ID	A	Y
Rheia	0	0
Kronos	0	1
Demeter	0	0
Hades	0	0
Hestia	1	0
Poseidon	1	0
Hera	1	0
Zeus	1	1
Artemis	0	1
Apollo	0	1
Circe	0	0
Ares	1	1
Athene	1	1
Eros	1	1
Aphrodite	1	1
Prometheus	1	1
Selene	1	1
Hermes	1	0
Eos	1	0
Helios	1	0

ID	$Y_{a=0}$	$Y_{a=1}$
Rheia	0	1
Kronos	1	0
Demeter	0	0
Hades	0	0
Hestia	0	0
Poseidon	1	0
Hera	0	0
Zeus	0	1
Artemis	1	1
Apollo	1	0
Circe	0	1
Ares	1	1
Athene	1	1
Eros	0	1
Aphrodite	0	1
Prometheus	0	1
Selene	1	1
Hermes	1	0
Eos	1	0
Helios	1	0

Individual Causal Effect

contrast of the values of counterfactual outcomes, but only one of those values is observed.

ID	A	Y	$Y_{a=0}$	$Y_{a=1}$
Rheia	0	0	0	?
Kronos	0	1	1	?
Demeter	0	0	0	?
Hades	0	0	0	?
Hestia	1	0	?	0
Poseidon	1	0	?	0
Hera	1	0	?	0
Zeus	1	1	?	1
Artemis	0	1	1	?
Apollo	0	1	1	?
Circe	0	0	0	?
Ares	1	1	?	1
Athene	1	1	?	1
Eros	1	1	?	1
Aphrodite	1	1	?	1
Prometheus	1	1	?	1
Selene	1	1	?	1
Hermes	1	0	?	0
Eos	1	0	?	0
Helios	1	0	?	0

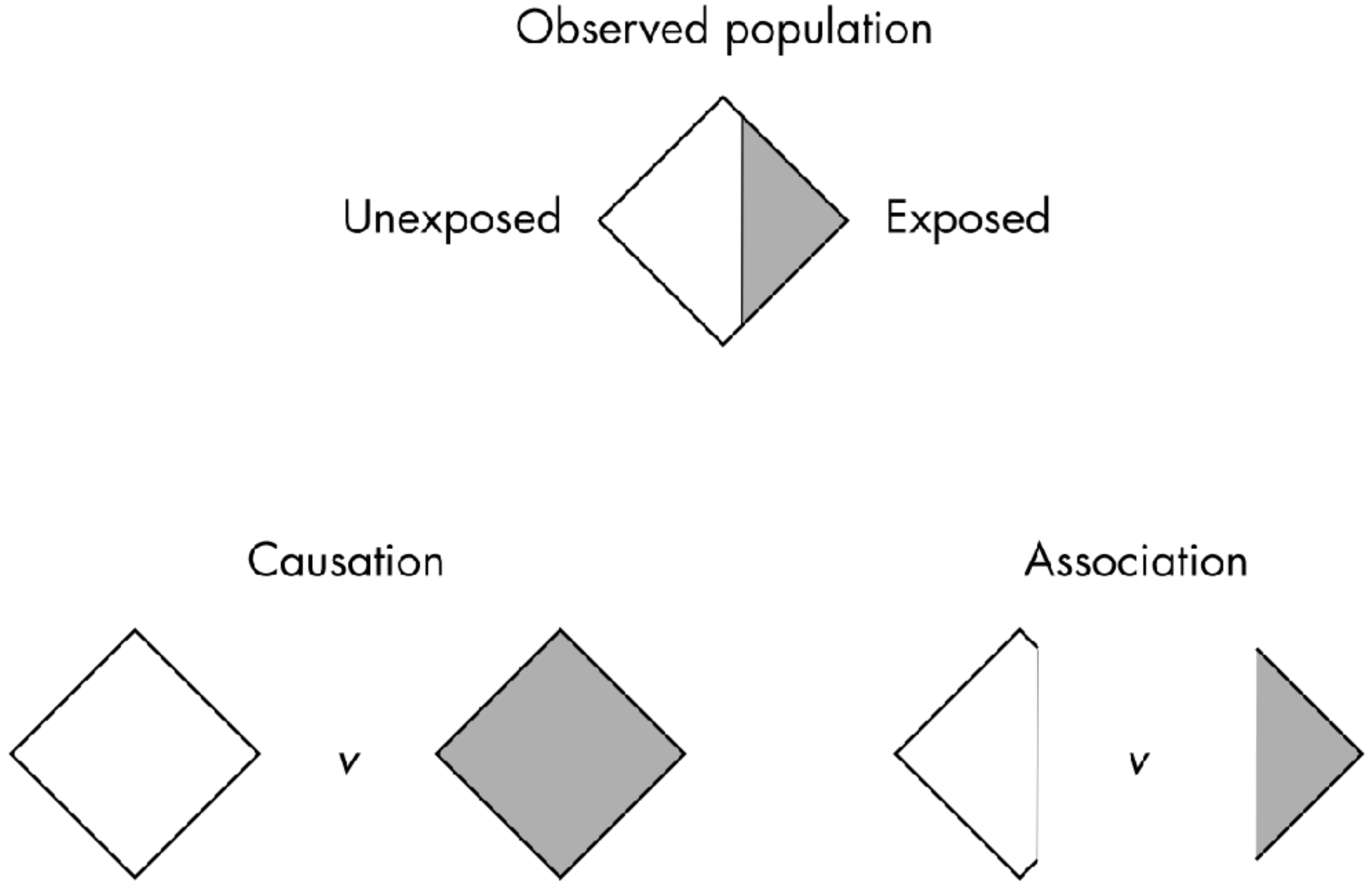


Population Causal Effects

- Pr[Ya = 1]: proportion of subjects that would have developed the outcome Y had all subjects in the population of interest received exposure value a.
- The exposure has a causal effect in the population if Pr[Ya=1=1] ≠ Pr[Ya=0=1].
- Unlike individual causal effects, population causal effects can sometimes be computed—or, more rigorously, consistently estimated.

$$Pr[Y_{a=1} = 1] - Pr[Y_{a=0} = 1] \neq 0$$

Association is not Causation!



Computing Causal Effects via Randomization

Unlike association measures, effect measures cannot be directly computed because of missing data. However, effect measures can be computed/estimated in randomized experiments!

- Suppose we have a (near-infinite) population and that we flip a coin for each subject in such population. We assign the subject to group 1 if the coin turns tails, and to group 2 if it turns heads.
- Next we administer the treatment or exposure of interest ($A = 1$) to subjects in group 1 and placebo ($A = 0$) to those in group 2. Five days later, at the end of the study, we compute the mortality risks in each group, $\Pr[Y = 1|A = 1]$ and $\Pr[Y = 1|A = 0]$.
- When subjects are randomly assigned to groups 1 and 2, the proportion of deaths among the exposed, $\Pr[Y = 1|A = 1]$, will be the same whether subjects in group 1 receive the exposure and subjects in group 2 receive placebo, or vice versa.
- Because group membership is randomised, both groups are "comparable": which particular group got the exposure is irrelevant for the value of $\Pr[Y = 1|A = 1]$. (The same reasoning applies to $\Pr[Y = 1|A = 0]$.)
- Formally, we say that both groups are **exchangeable**.

Let's do some math!

$$Pr[Y = 1 | A = 1] = Pr[Y = 1 | A = 0] = Pr[Y_a = 1]$$

$$Pr[Y_a = 1 | A = a] = Pr[Y = 1 | A = a]$$

$$Pr[Y = 1 | A = a] = Pr[Y_a = 1]$$

In ideal randomized experiments, Association is Causation!

But not in non-randomized observational studies

Still remember this?

$$Pr[Y = 1 | A = 1] = 7/13$$

$$Pr[Y = 1 | A = 0] = 3/7$$

CONTINUING PROFESSIONAL EDUCATION

A definition of causal effect for epidemiological research

M A Hernán

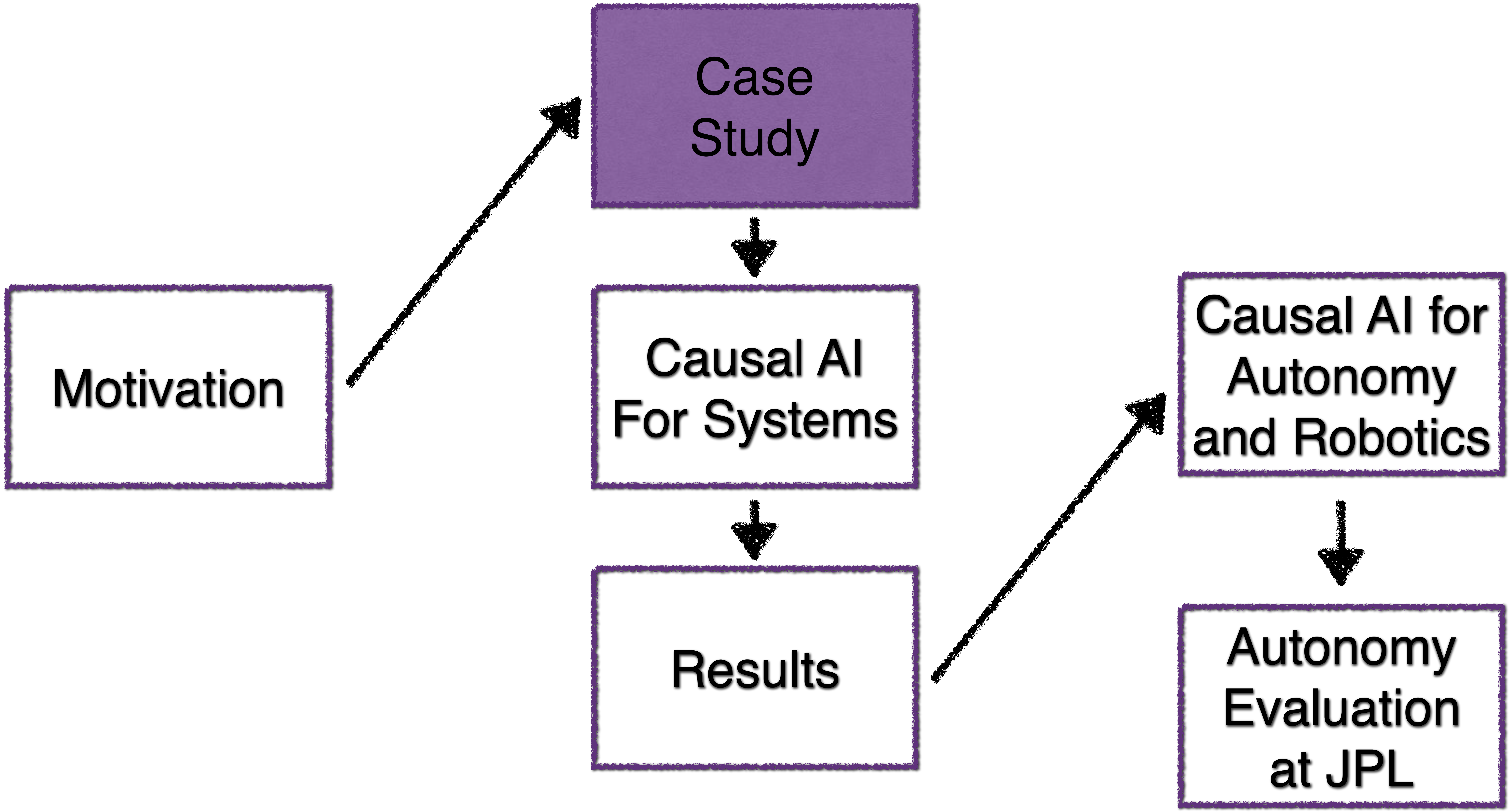
J Epidemiol Community Health 2004;**58**:265–271. doi: 10.1136/jech.2002.006361

Limitations!

- We have so far assumed that the counterfactual outcomes Y_a exist and are well defined.
- Loss to follow up
- Non-compliance
- Unblinding

Now, let's look at some applications of Causal AI in Computer Systems

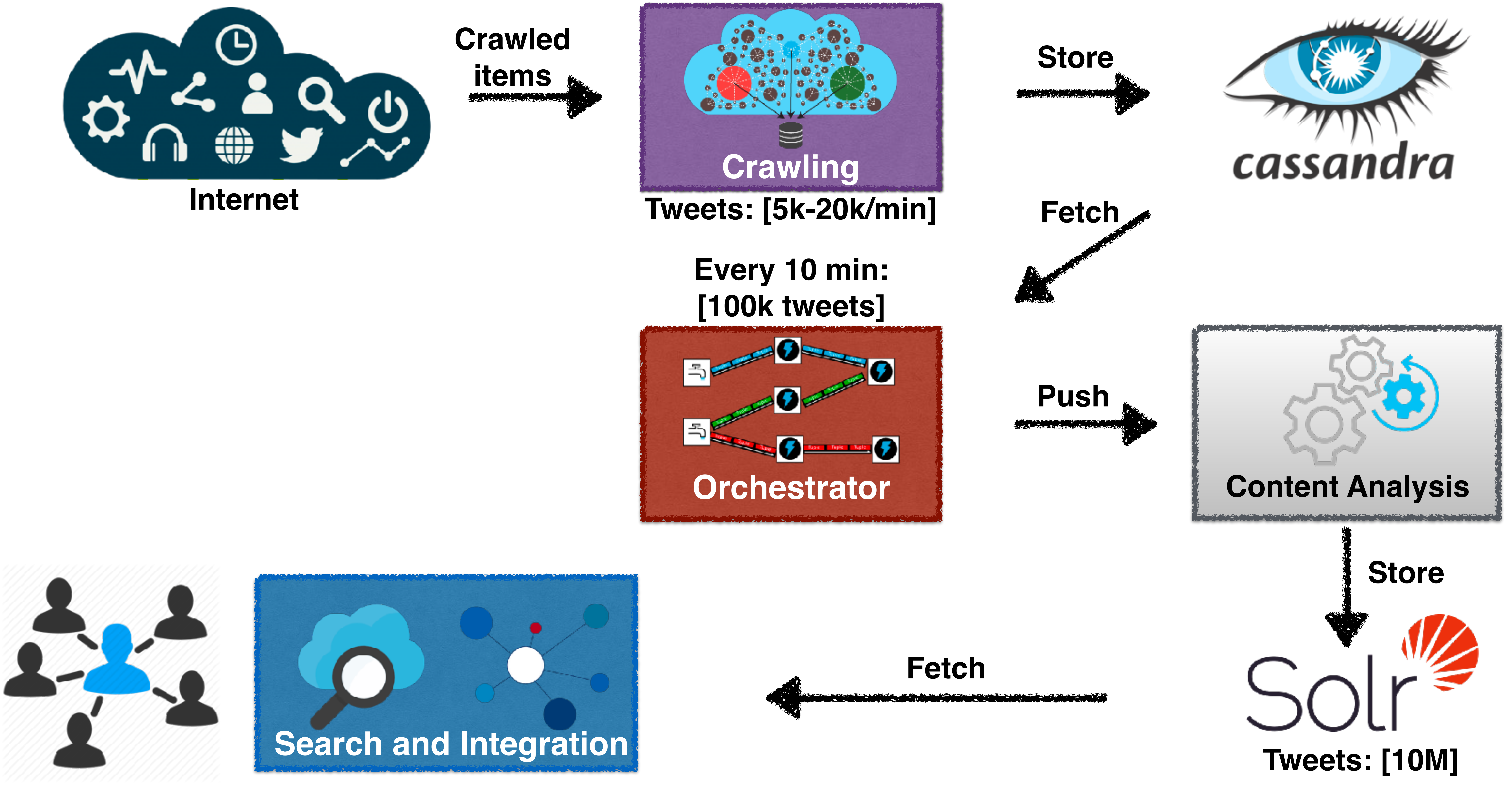
Outline


Motivation
Case Study
Causal AI For Systems
Results
Causal AI for Autonomy and Robotics
Autonomy Evaluation at JPL

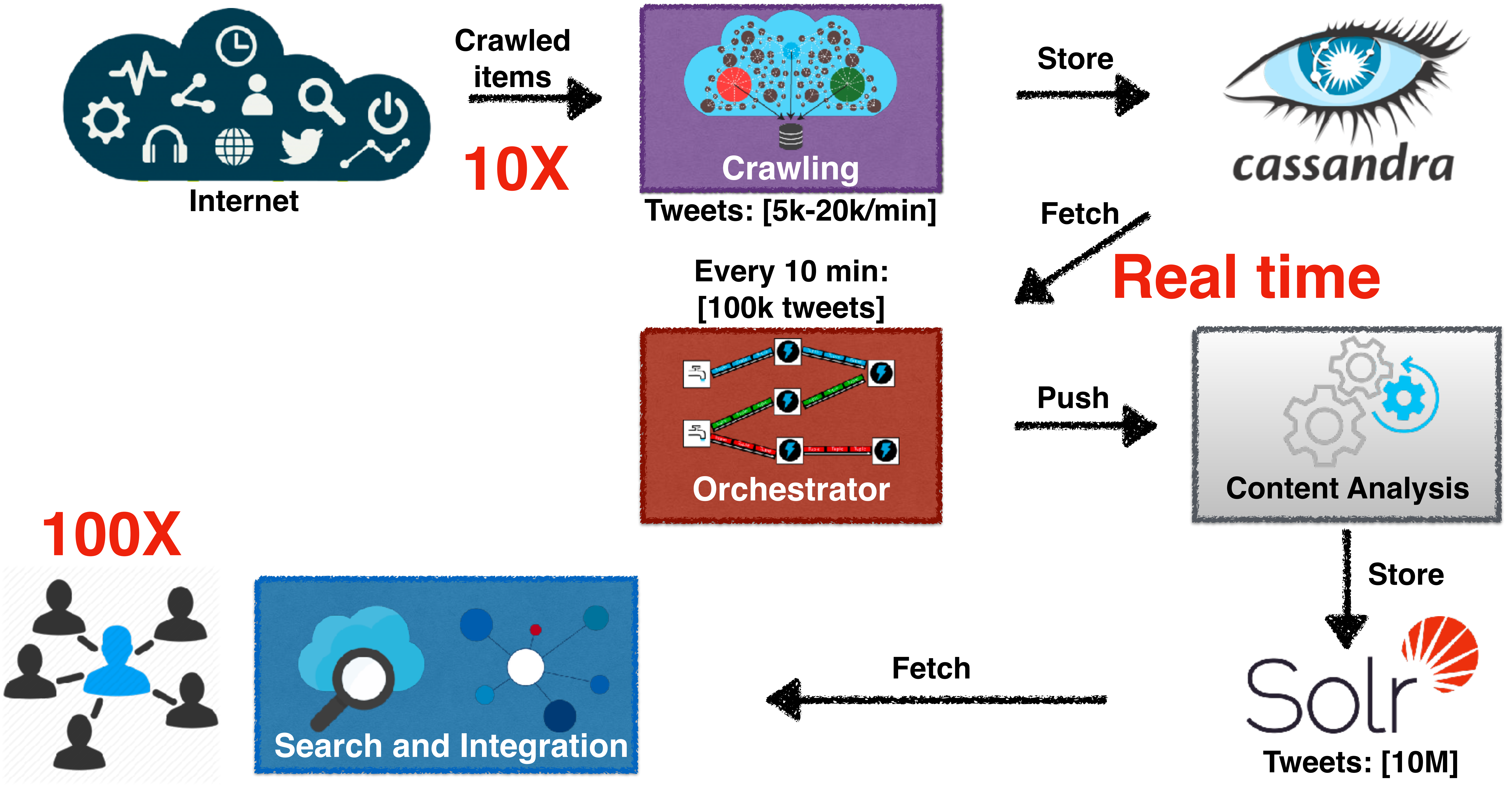
Case Study 1

SocialSensor

SocialSensor



Challenges



Internet
Crawled items
10X
Crawling
Tweets: [5k-20k/min]
Store
cassandra
Fetch
Real time
Every 10 min:
[100k tweets]
Orchestrator
Push
Content Analysis
Store
Solr
Tweets: [10M]
Fetch
100X
Search and Integration

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

cassandra STORM Solr

> 100 > 100 > 100

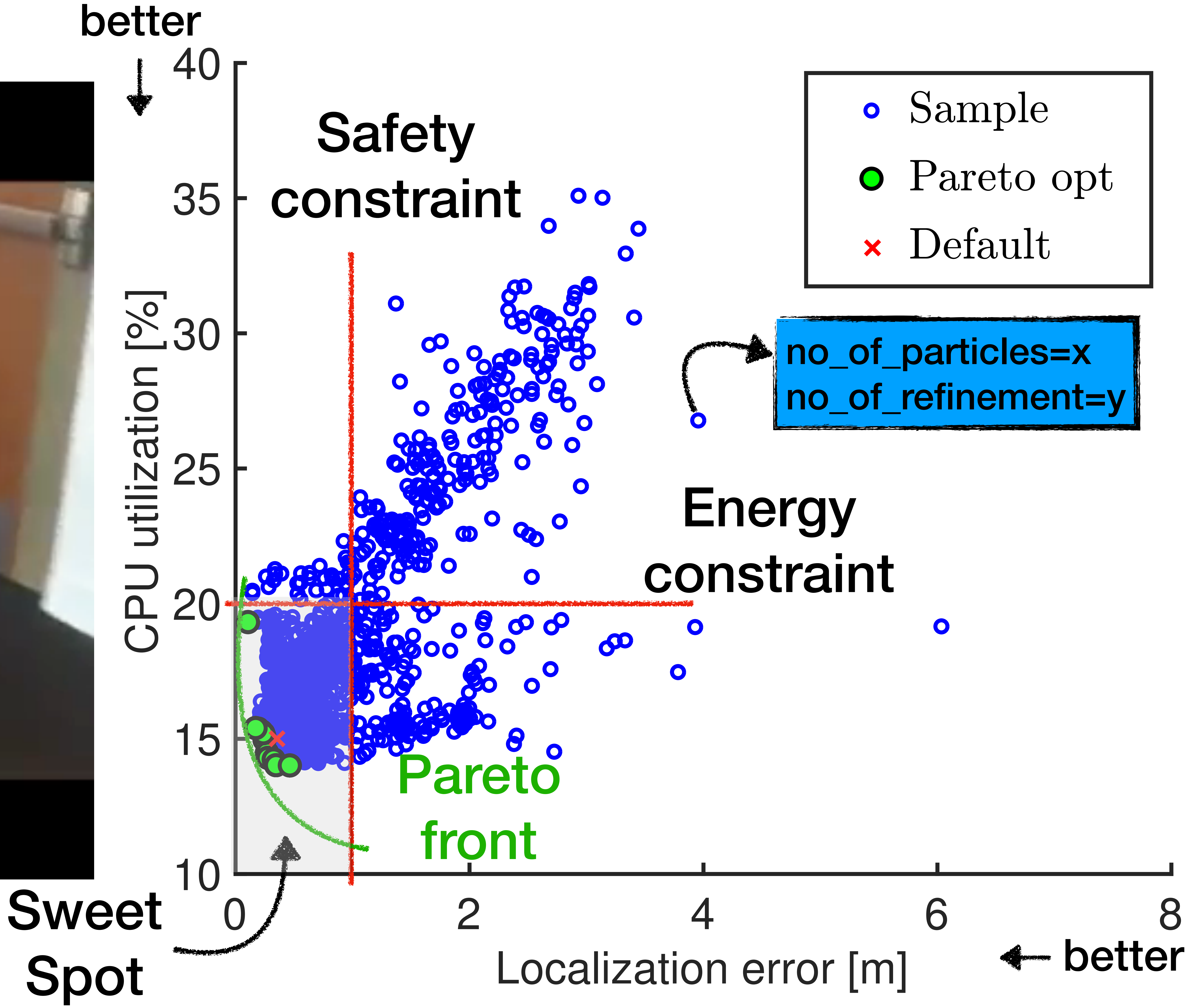
$$2^{300}$$

More configurations than estimated atoms in the universe

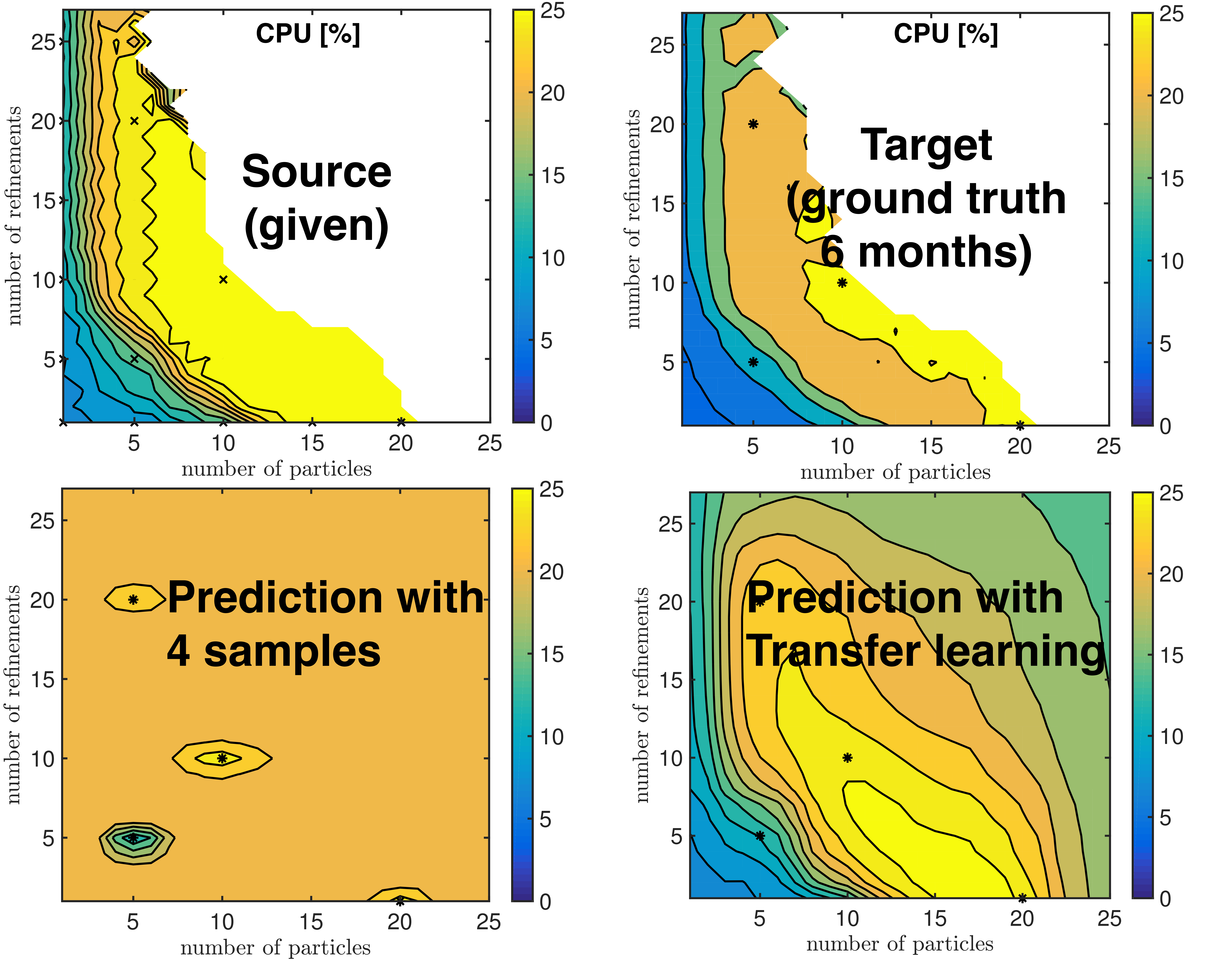
Case Study 2

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:**

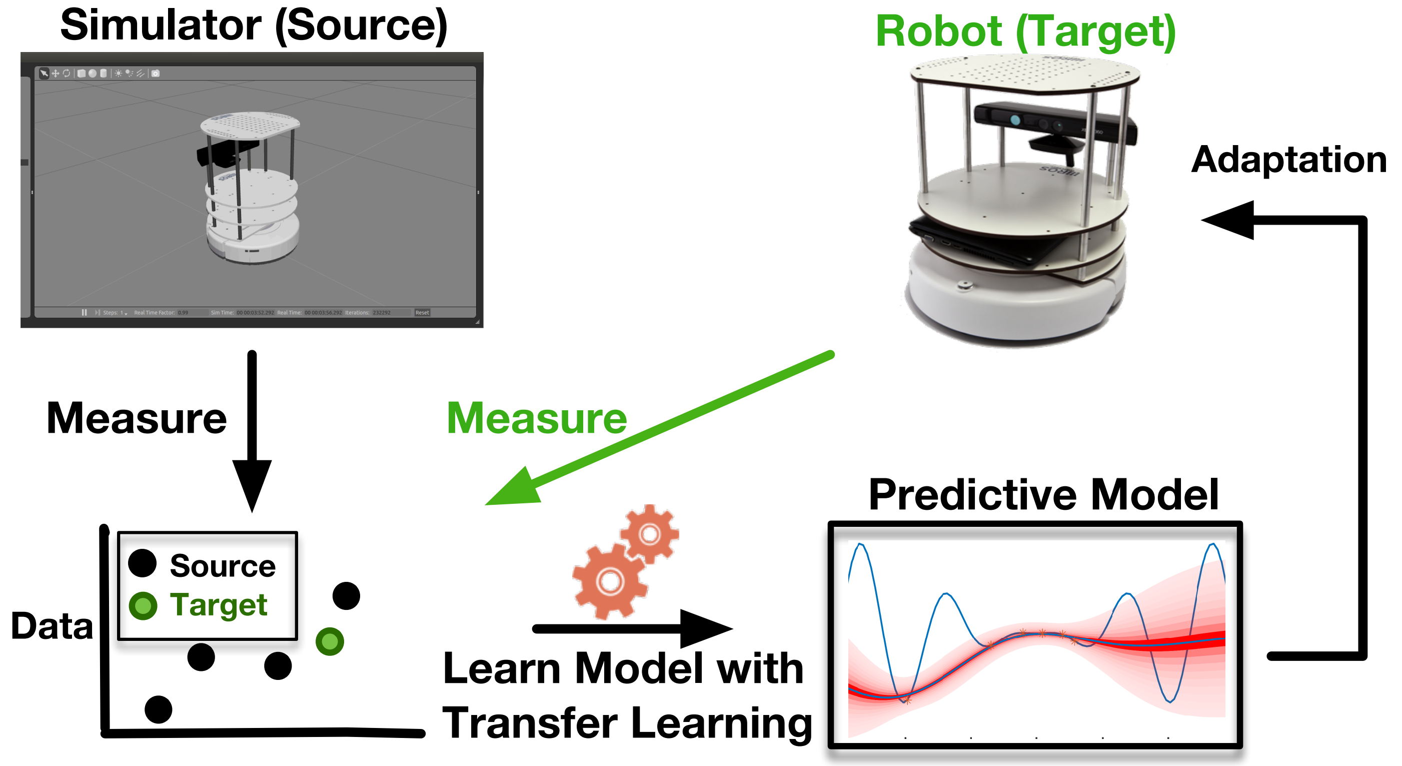
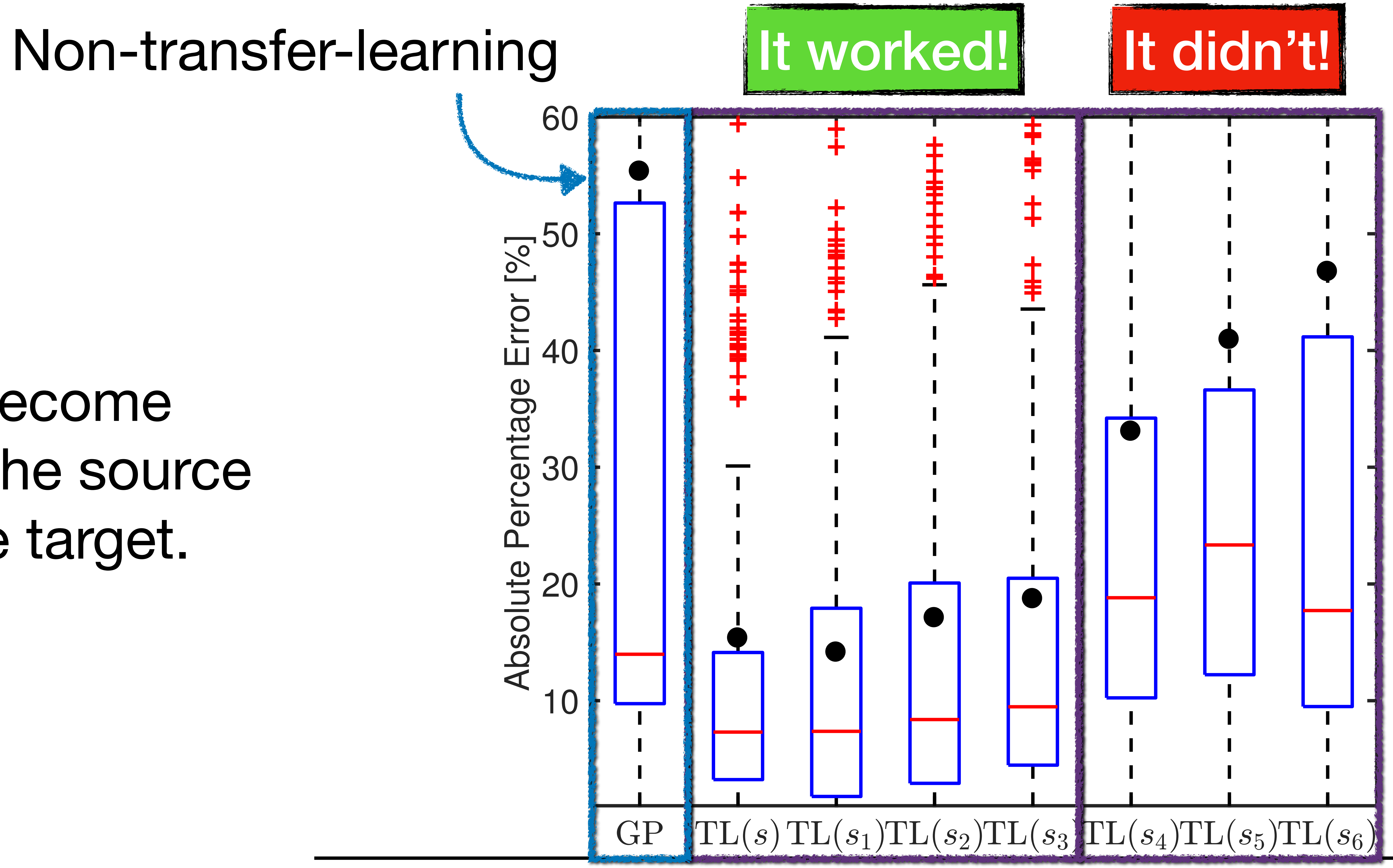


Fig. 1: Transfer learning for performance model learning.

Looking further: When transfer learning goes wrong

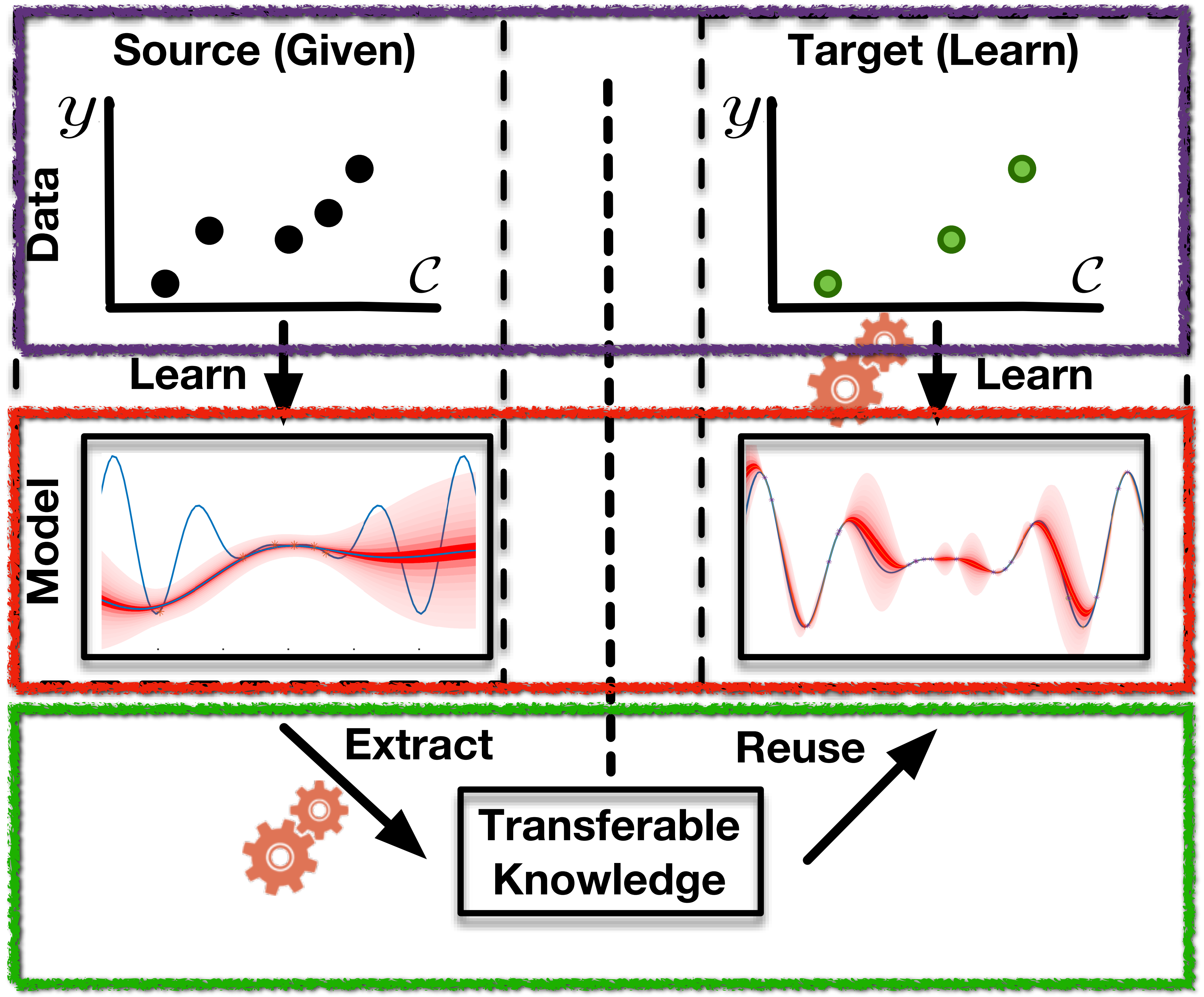


Insight: Predictions become more accurate when the source is **more related** to the target.

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

CPU usage [%]
It worked!
number of refinements
number of particles
(a)
(b)
(c)
It didn't!
(d)
(e)
(f)

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**

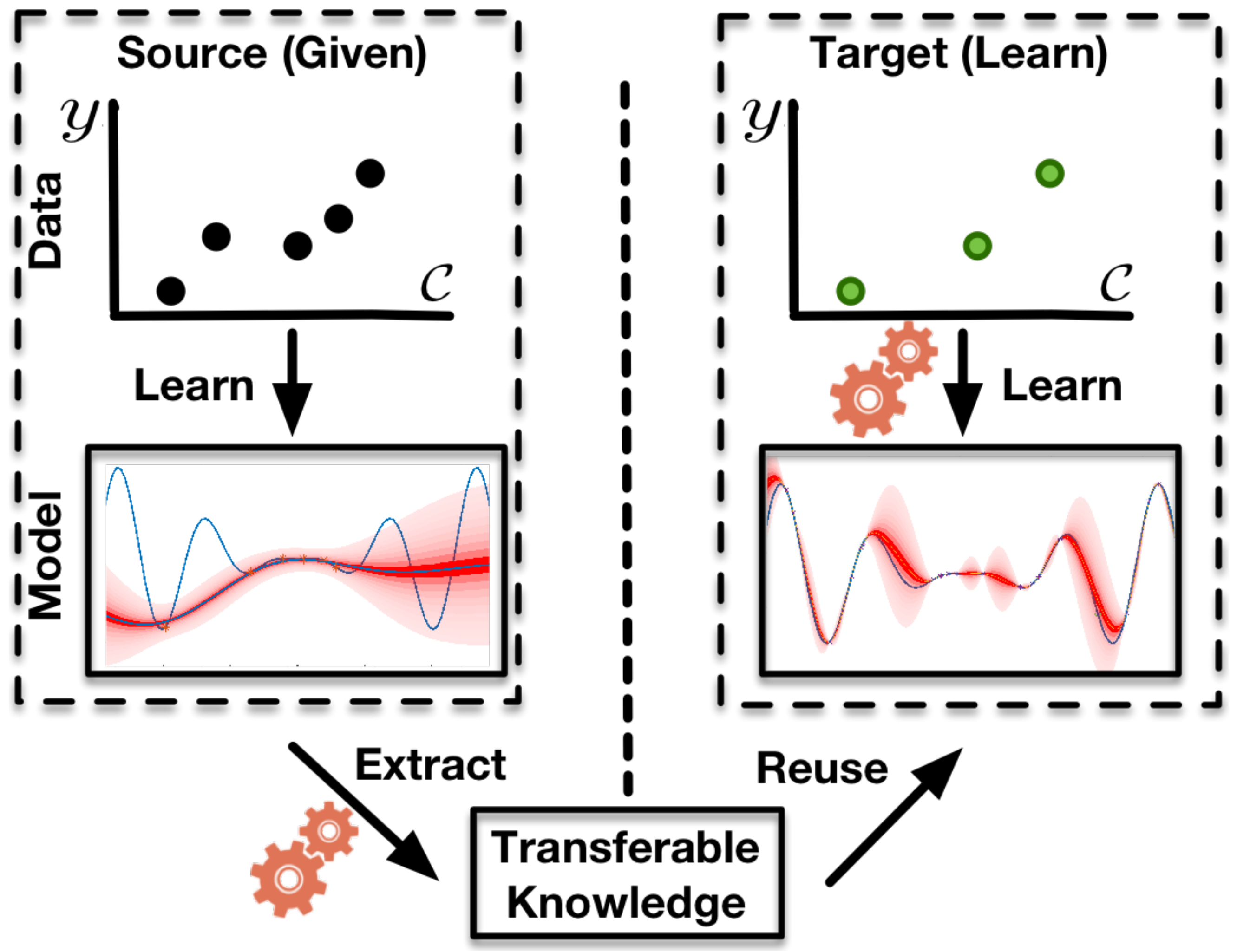


Fig. 1: Transfer learning is a form of machine learning that takes advantage of transferable knowledge from source to learn an accurate, reliable, and less costly model for the target environment.

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

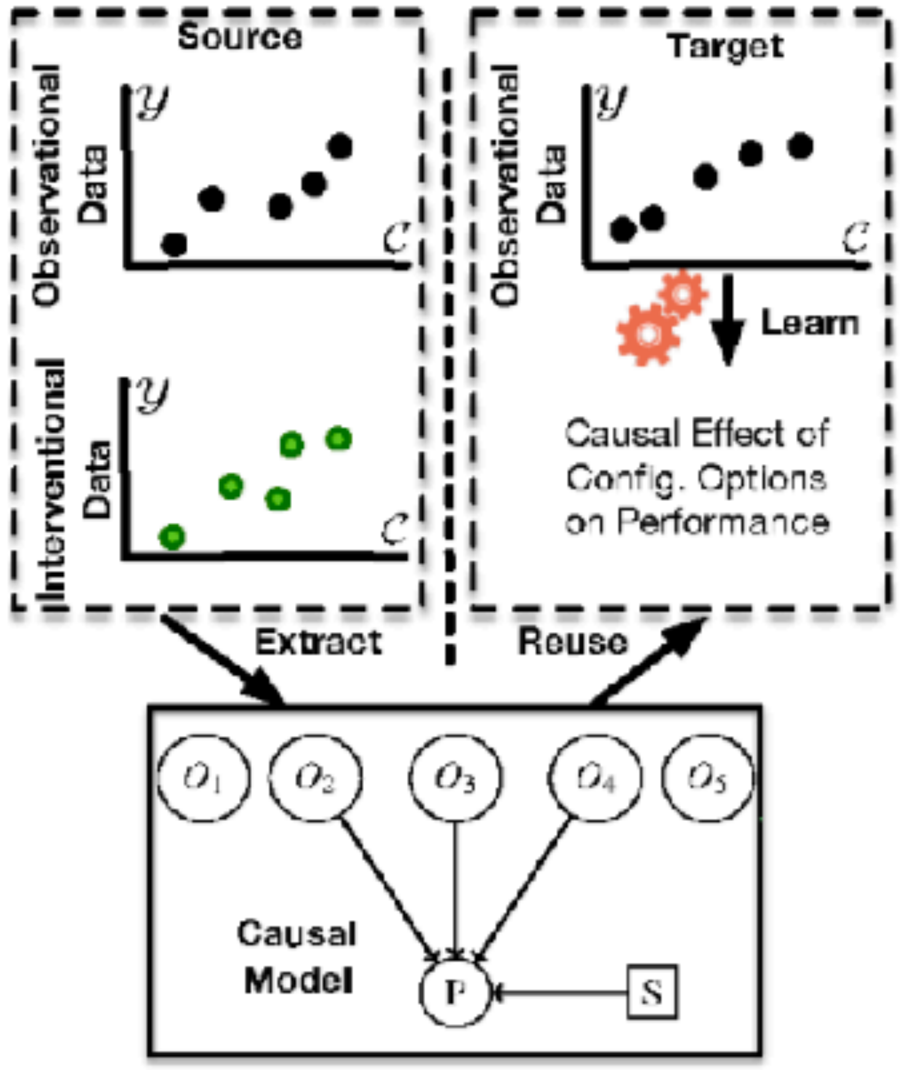
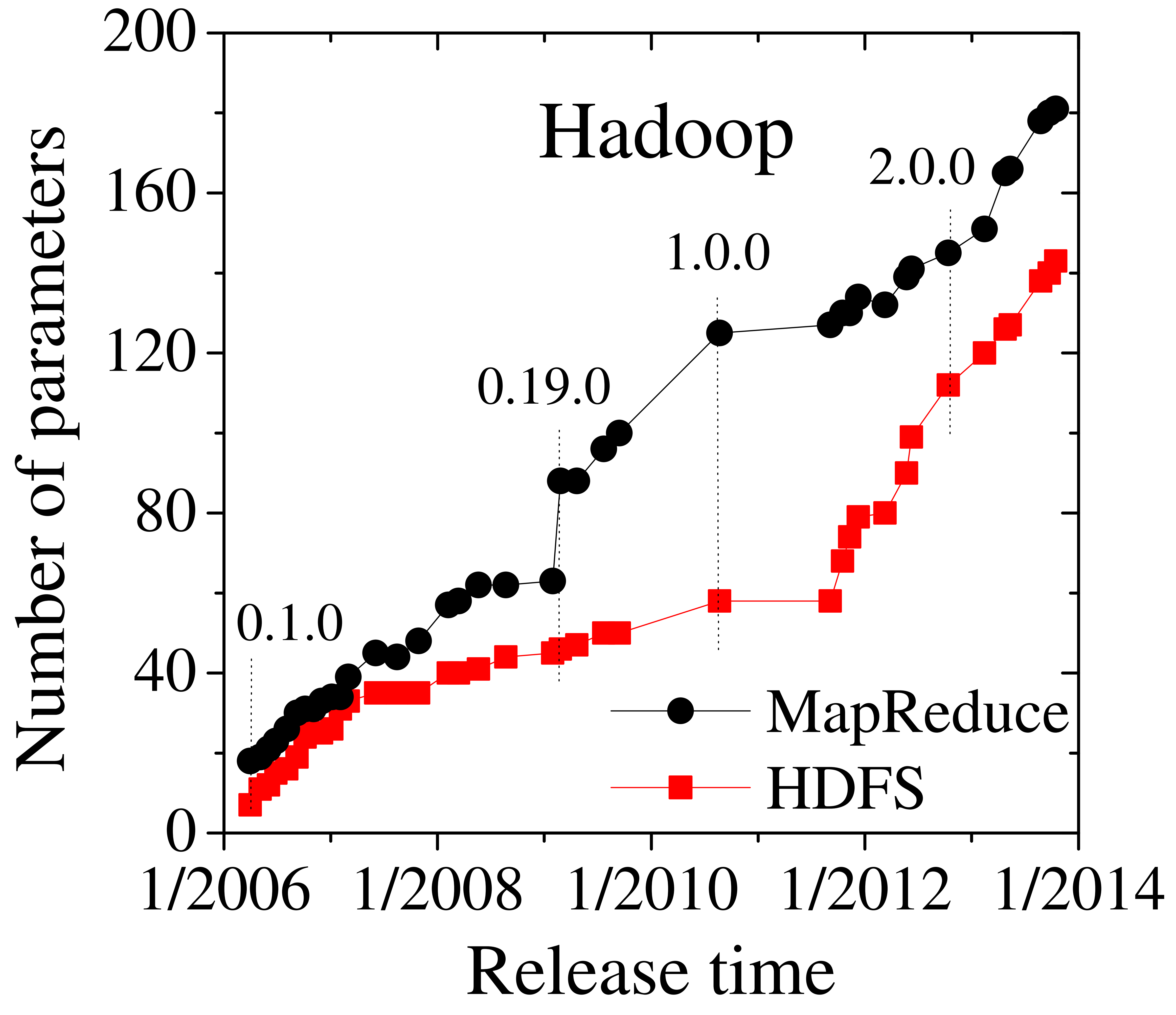


Figure 1: Exploiting causal inference for performance analysis.

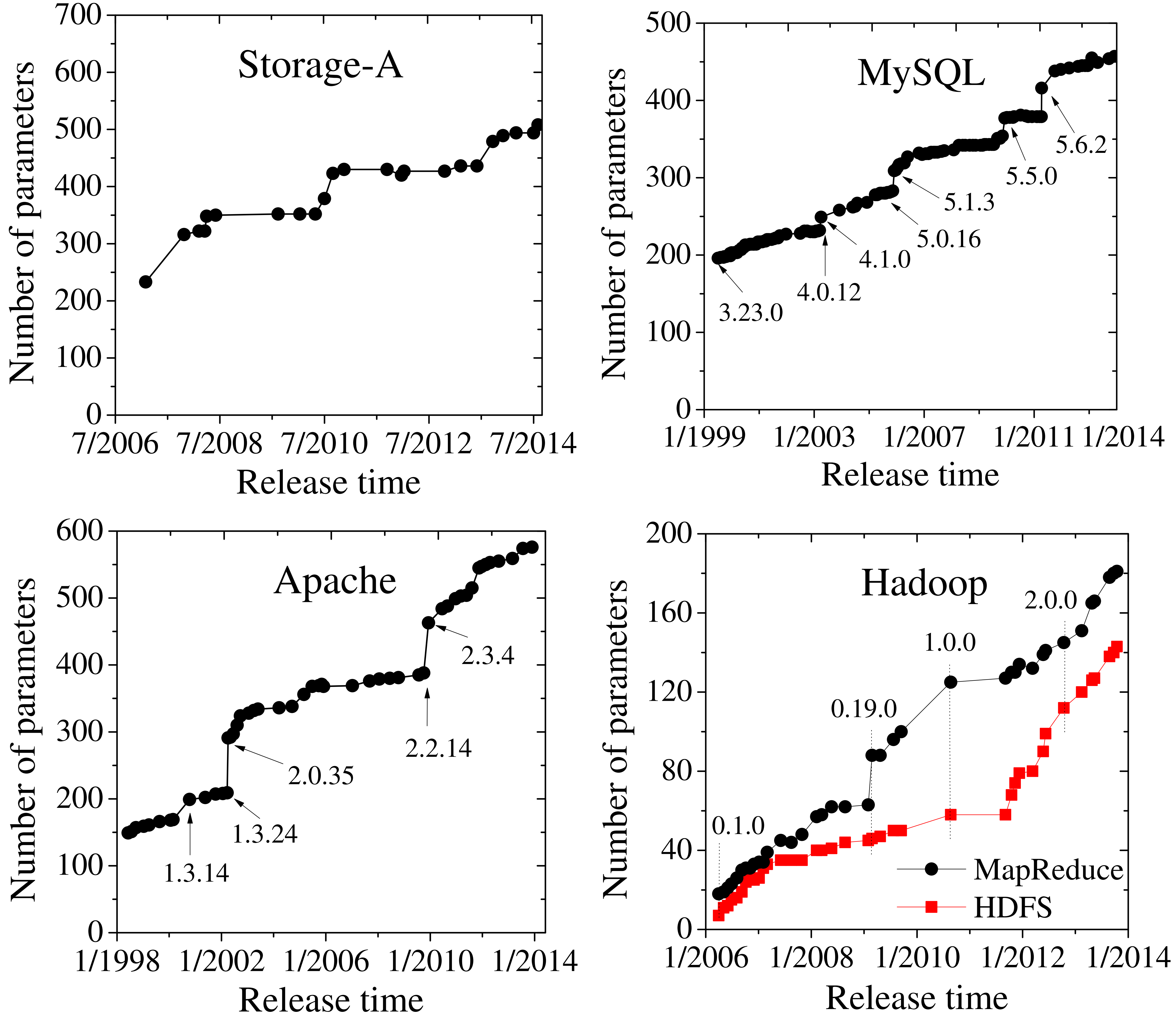
```yaml

drpc.port: 3772
drpc.worker.threads: 64
drpc.max_buffer_size: 1048576
drpc.queue.size: 128
drpc.invocations.port: 3773
drpc.invocations.threads: 64
drpc.request.timeout.secs: 600
drpc.childopts: "-Xmx768m"
drpc.http.port: 3774
drpc.https.port: -1
drpc.https.keystore.password: ""
drpc.https.keystore.type: "JKS"
drpc.http.creds.plugin: org.apache.storm.security.auth.DefaultHttpCredentialsPlugi
drpc.authorizer.acl.filename: "drpc-auth-acl.yaml"
drpc.authorizer.acl.strict: false

transactional.zookeeper.root: "/transactional"
transactional.zookeeper.servers: null
transactional.zookeeper.port: null

## blobstore configs
supervisor.blobstore.class: "org.apache.storm.blobstore.NimbusBlobStore"
supervisor.blobstore.download.thread.count: 5
supervisor.blobstore.download.max_retries: 3
supervisor.localizer.cache.target.size.mb: 10240
supervisor.localizer.cleanup.interval.ms: 600000
```

Empirical observations confirm that systems are becoming increasingly configurable



[Tianyin Xu, et al., “**Too Many Knobs**…”, FSE’15]

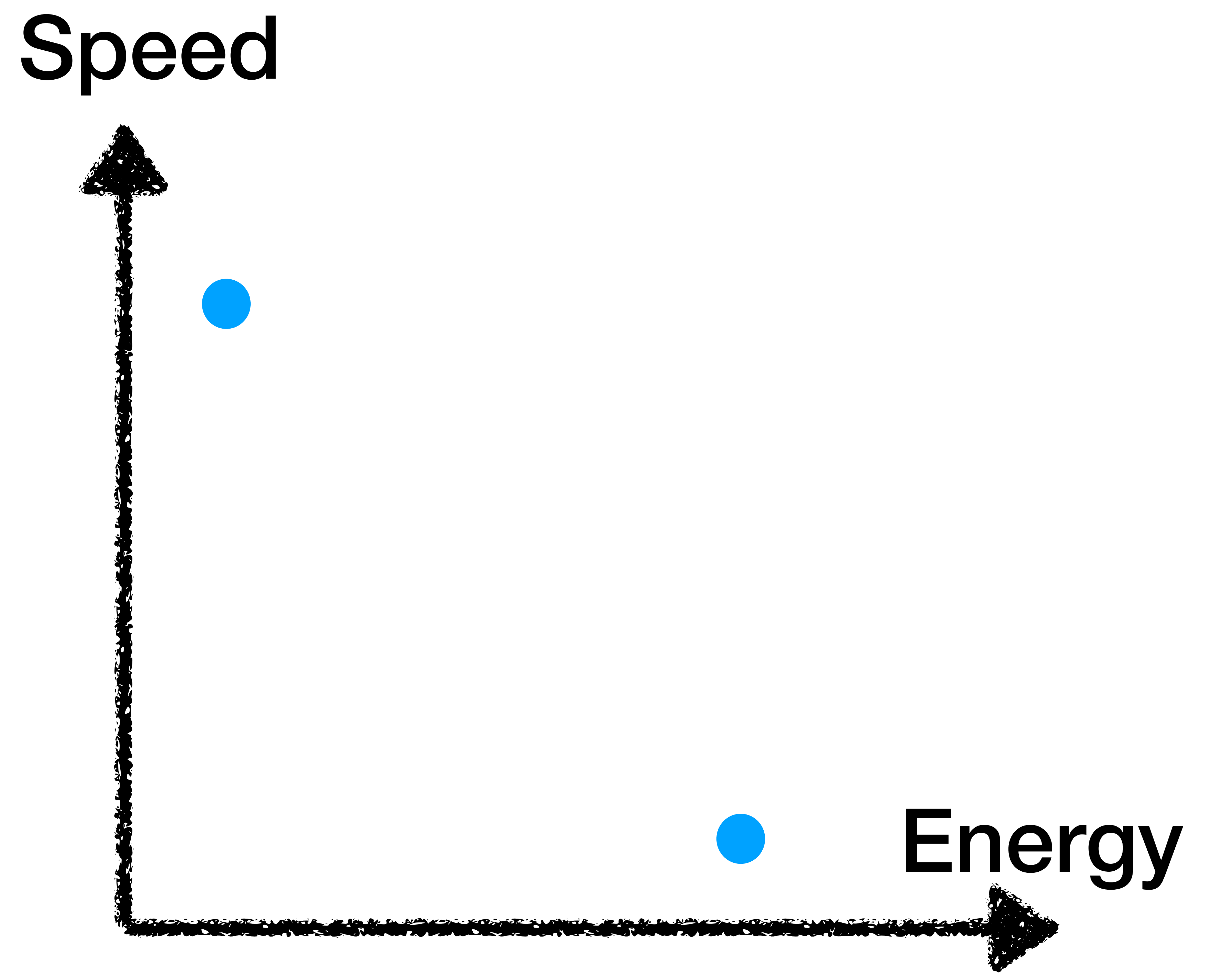
Empirical observations confirm that systems are becoming increasingly configurable



[Tianyin Xu, et al., “**Too Many Knobs**…”, FSE’15]

Configurations determine the performance behavior

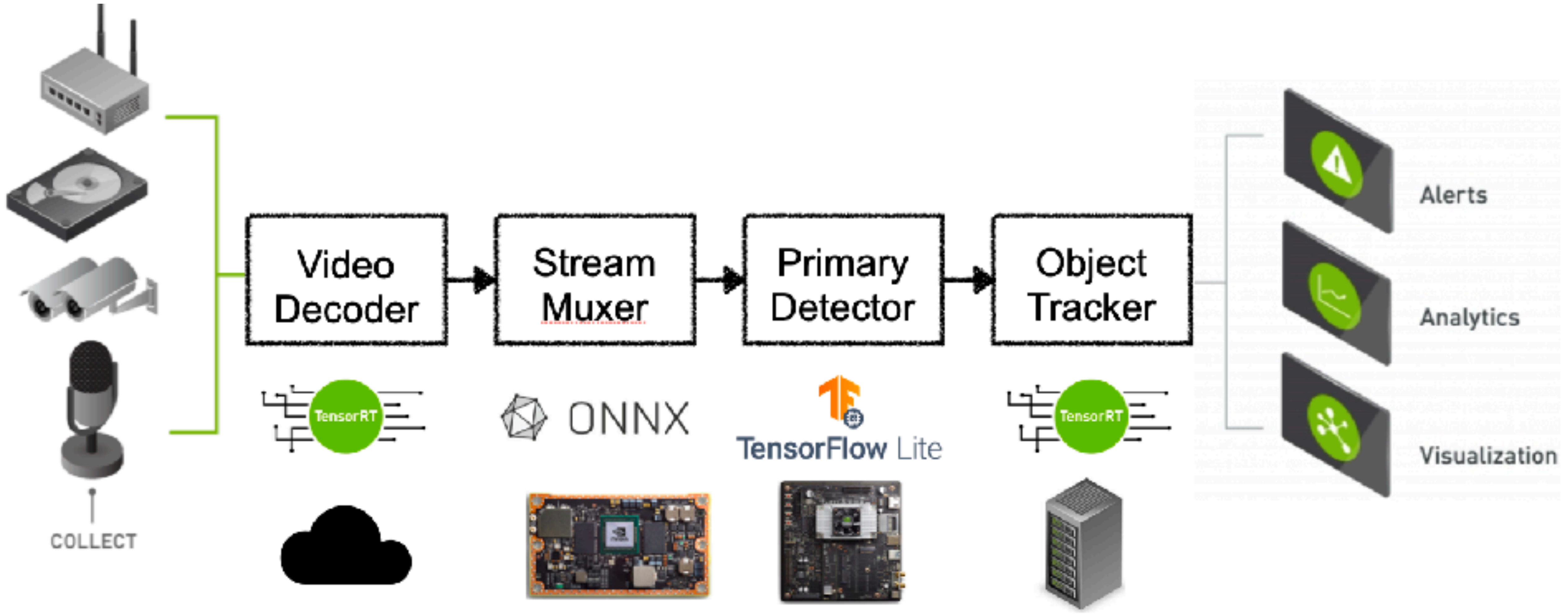
```
void Parrot_setenv(. . . name,. . . value){
#ifdef PARROT_HAS_SETENV
  my_setenv(name, value, 1);
#else
  int name_len=strlen(name);
  int val_len=strlen(value);
  char* envs=glob_env;
  if(envs==NULL){
    return;
  }
  strcpy(envs,name);
  strcpy(envs+name_len,"=");
  strcpy(envs+name_len + 1,value);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



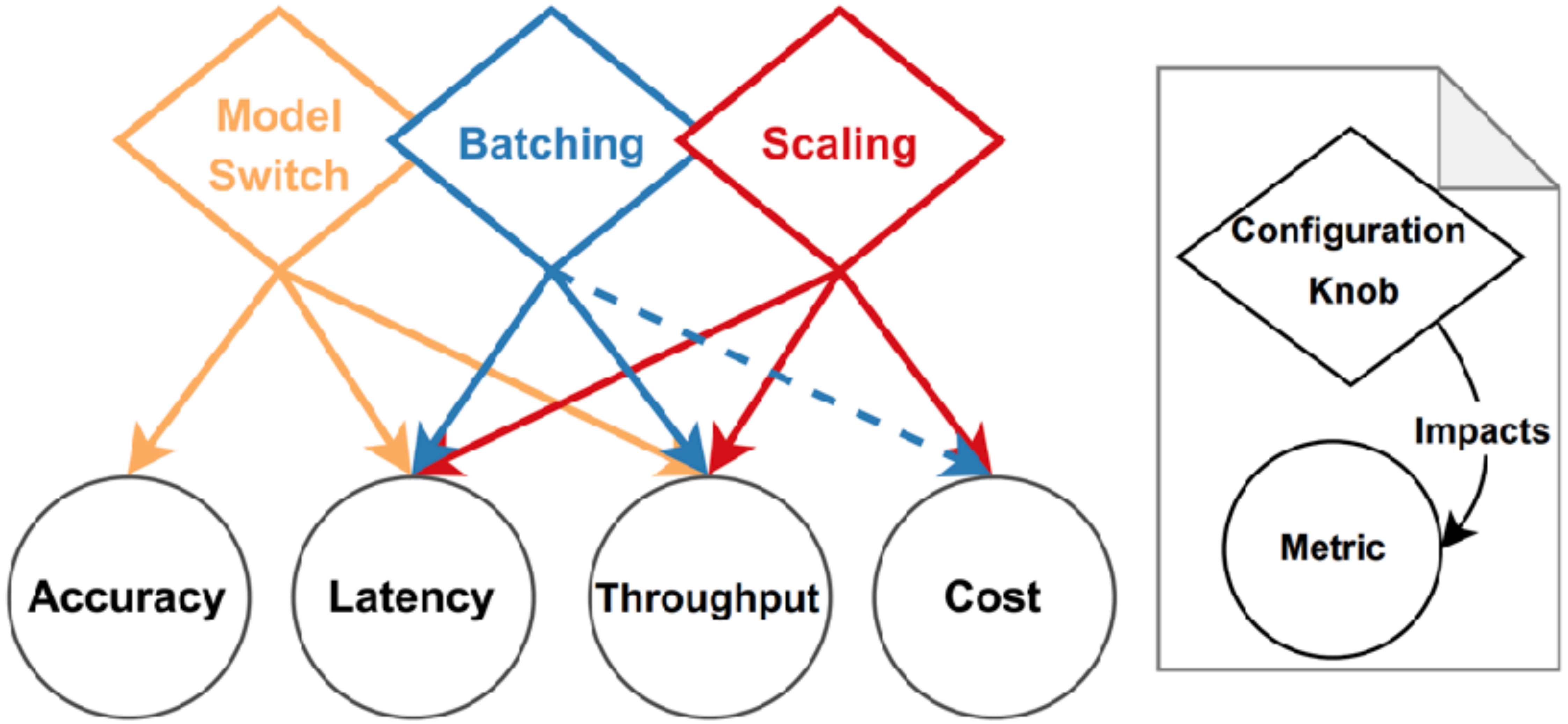
Challenges of configurations

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

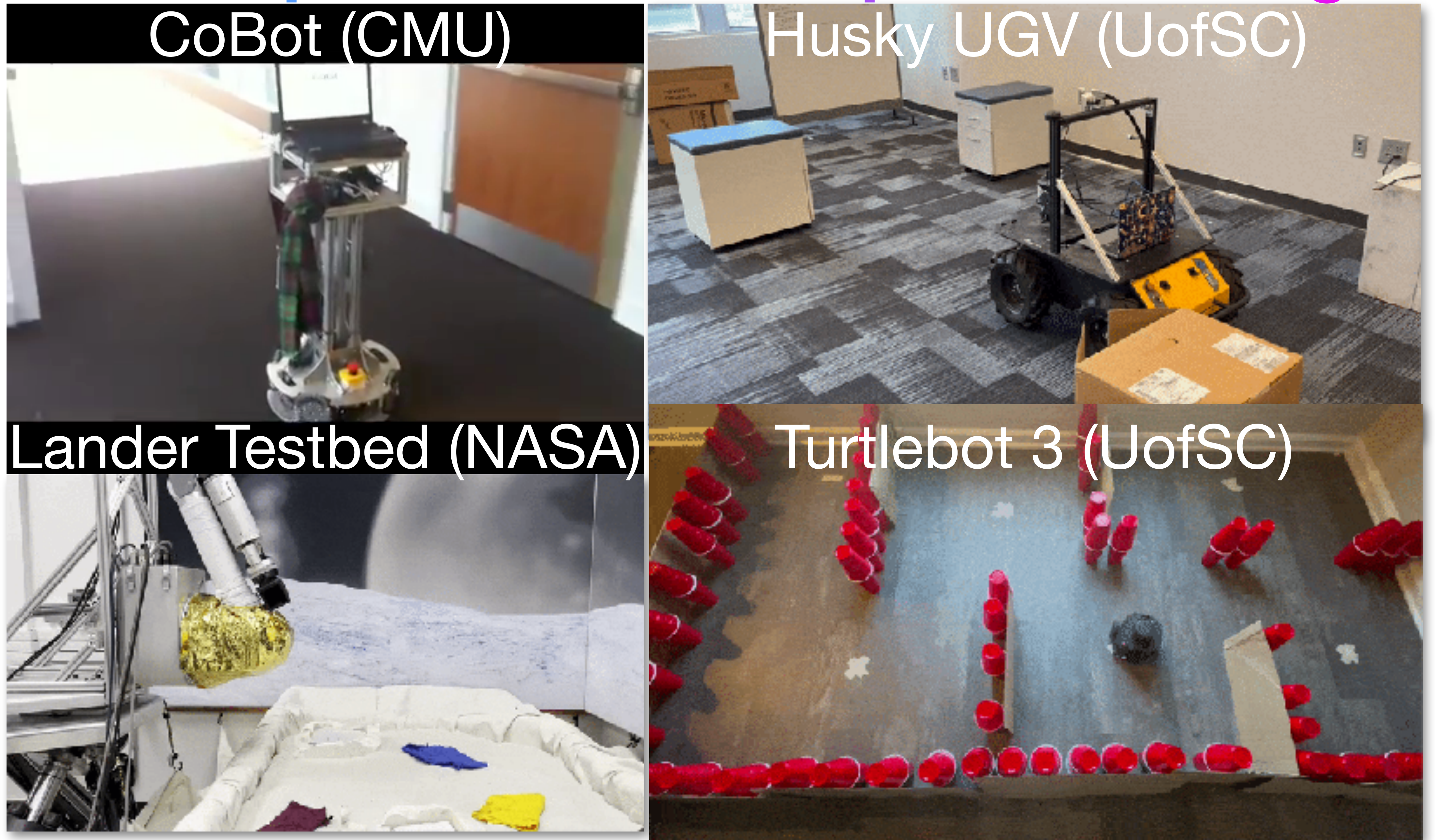
The variability space (design space) of (composed) systems is exponentially increasing



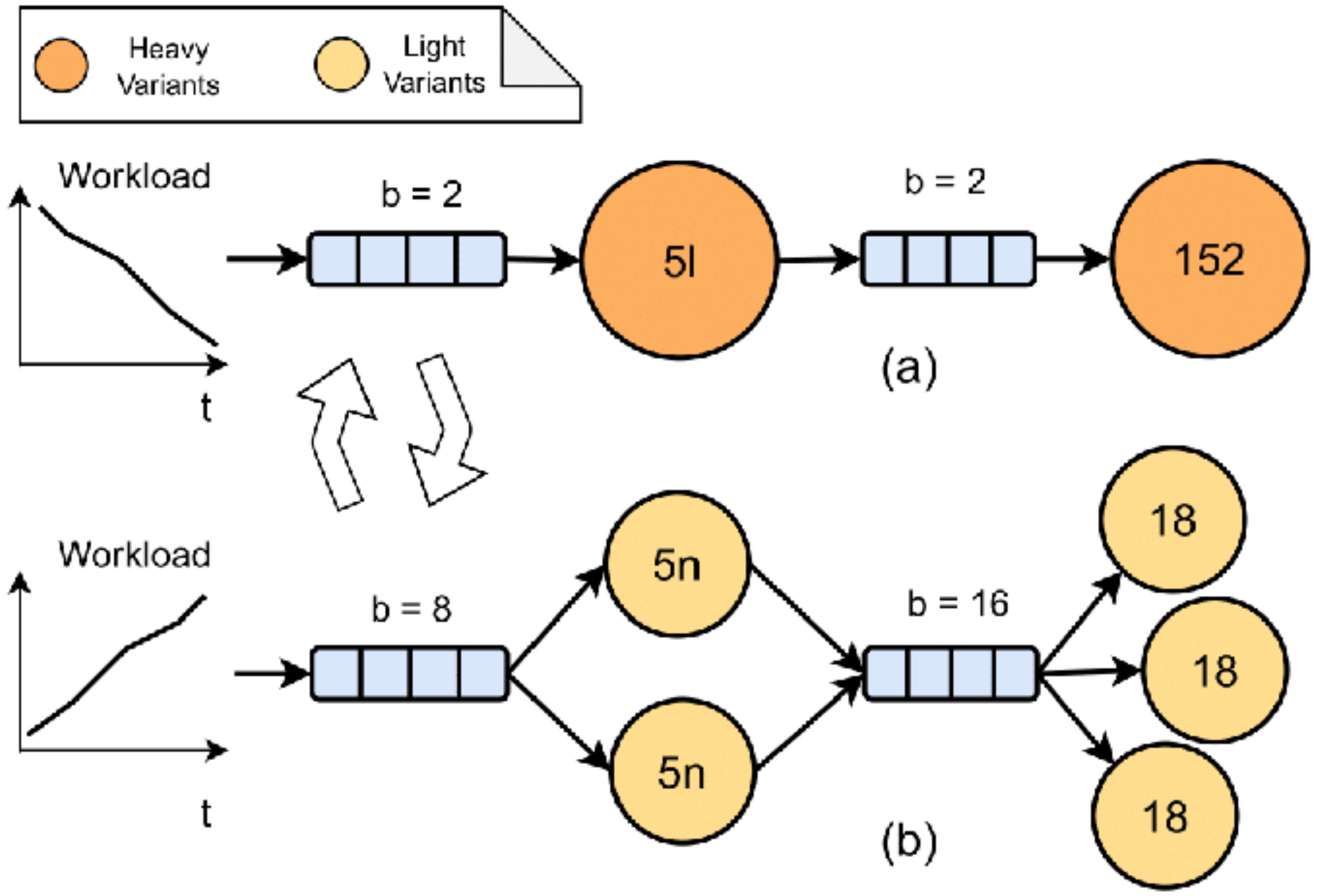
Performance goals are competing and users have preferences over these goals



Systems operate in uncertain environments with imperfect and incomplete knowledge



Goal: Enabling users to find the right quality tradeoff

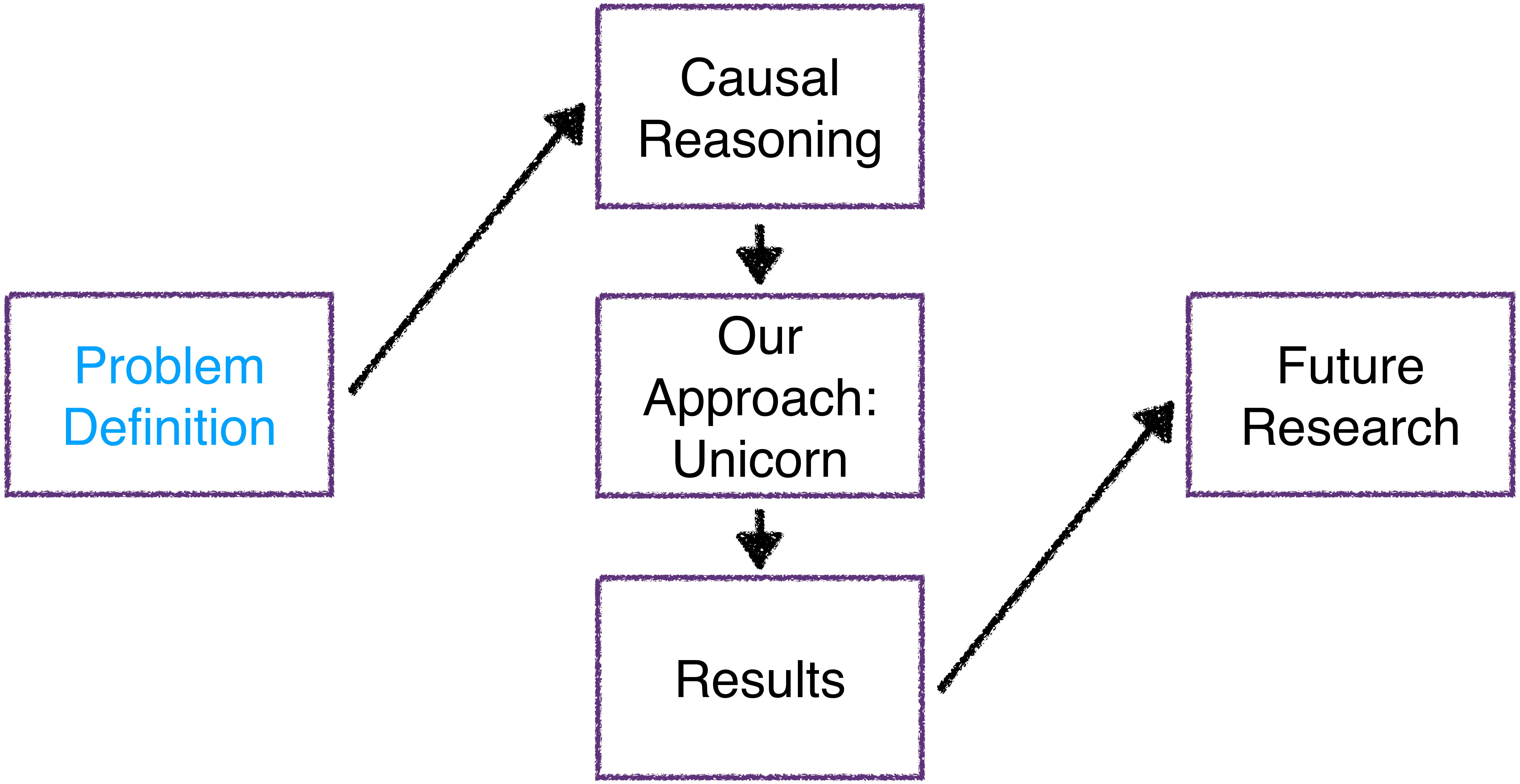


The goal of our research is…

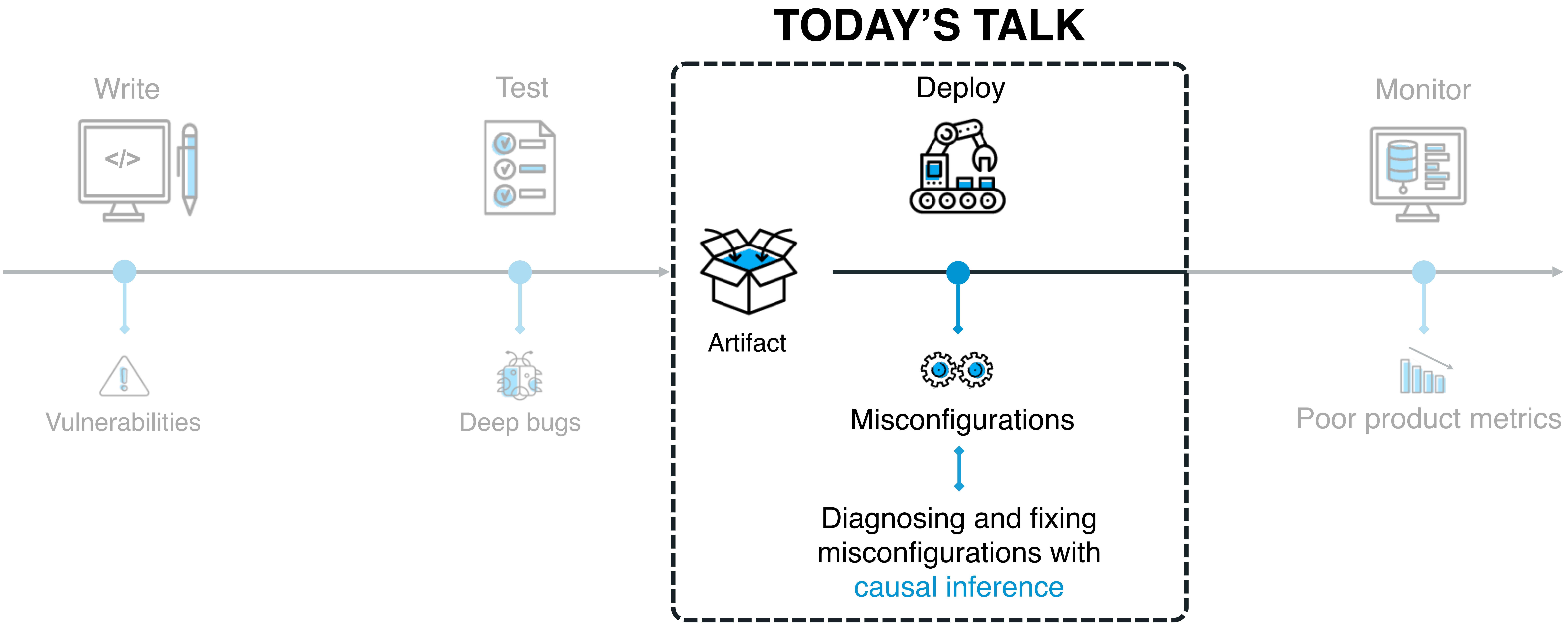
Understanding the performance behavior of real-world highly-configurable systems that scale well…

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

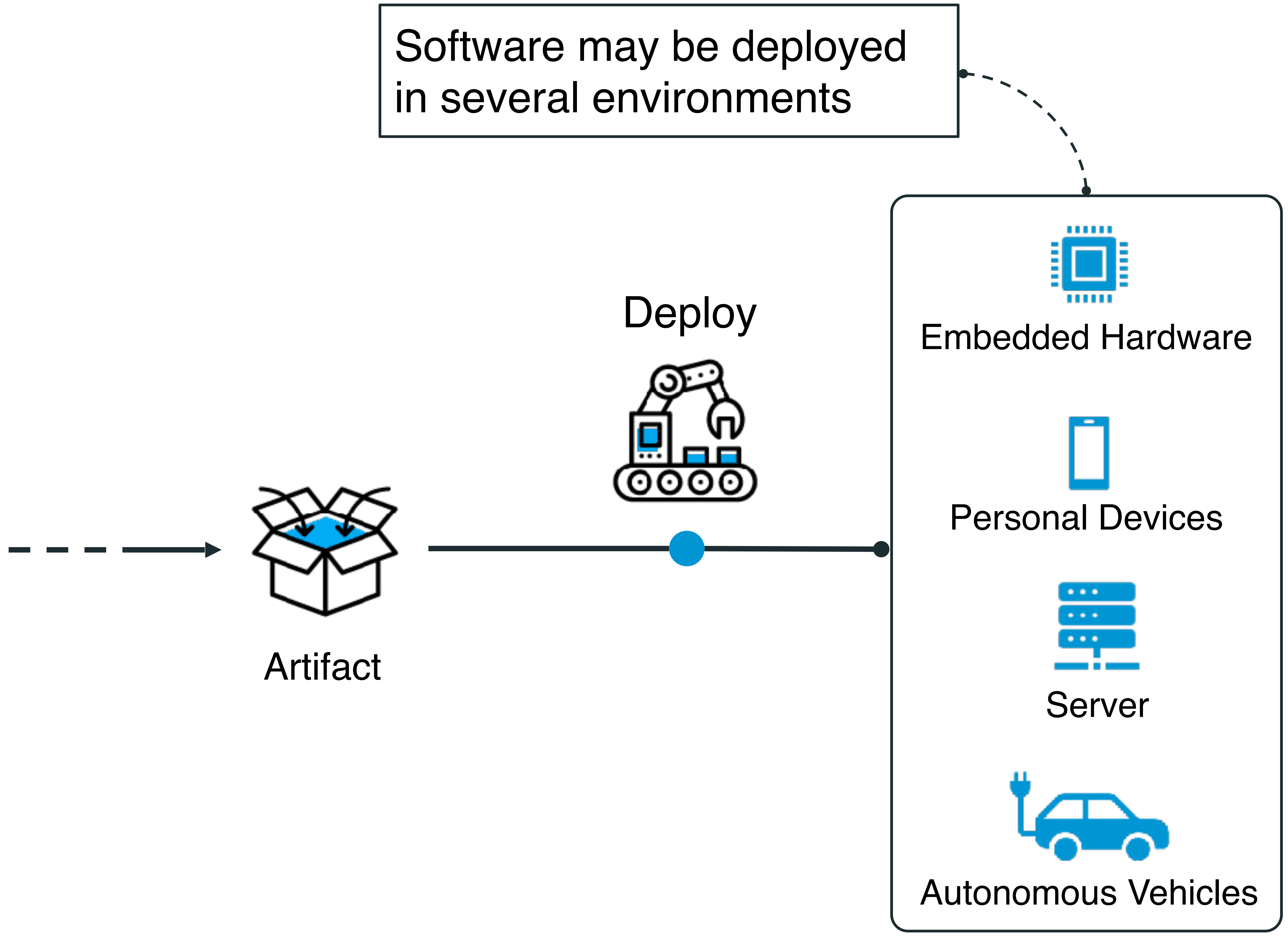
Outline


Problem Definition
Causal Reasoning
Our Approach: Unicorn
Results
Future Research

A Typical Software Lifecycle



Today’s Talk

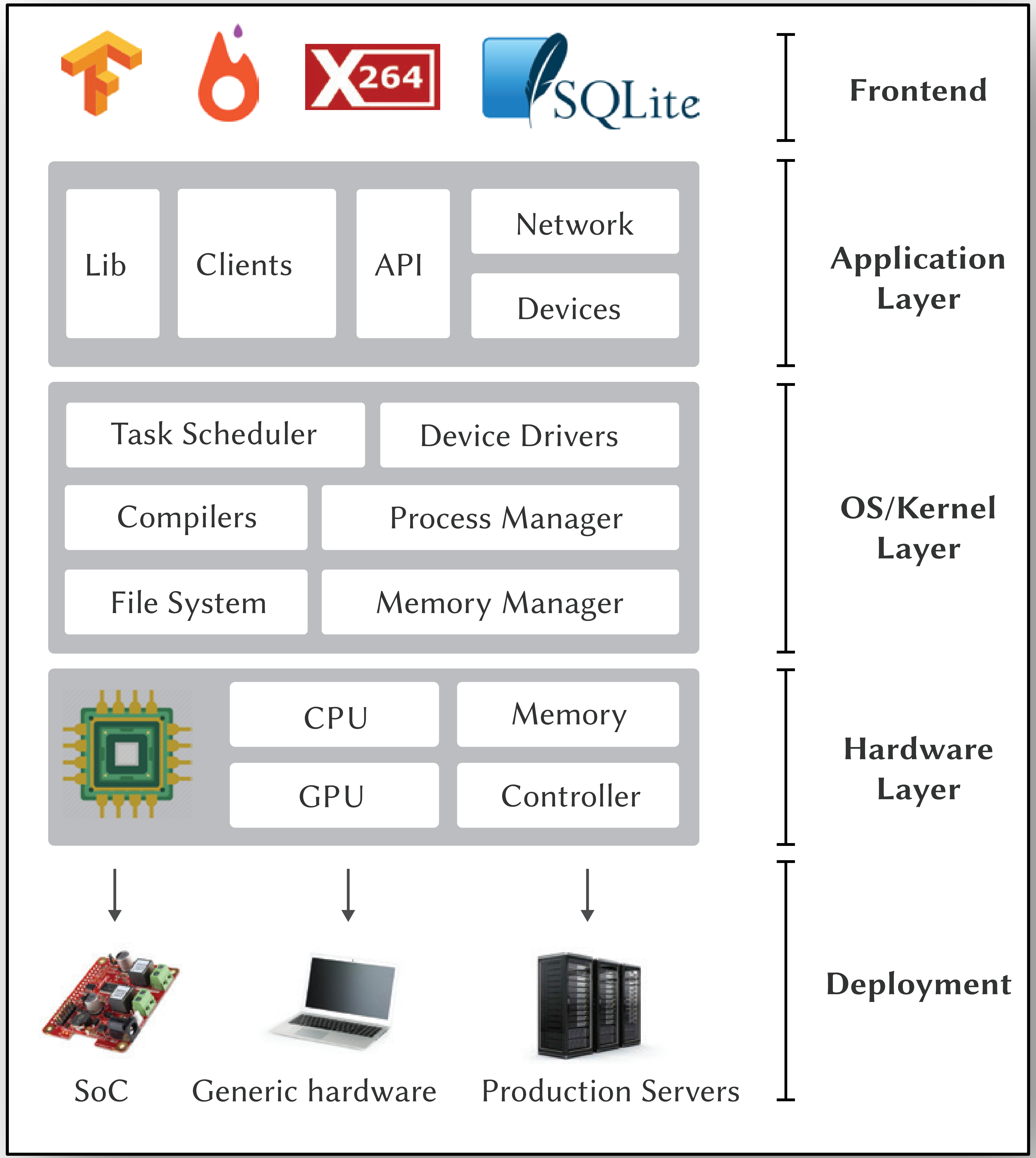


Deployment Environments

Challenge

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

Today's Talk



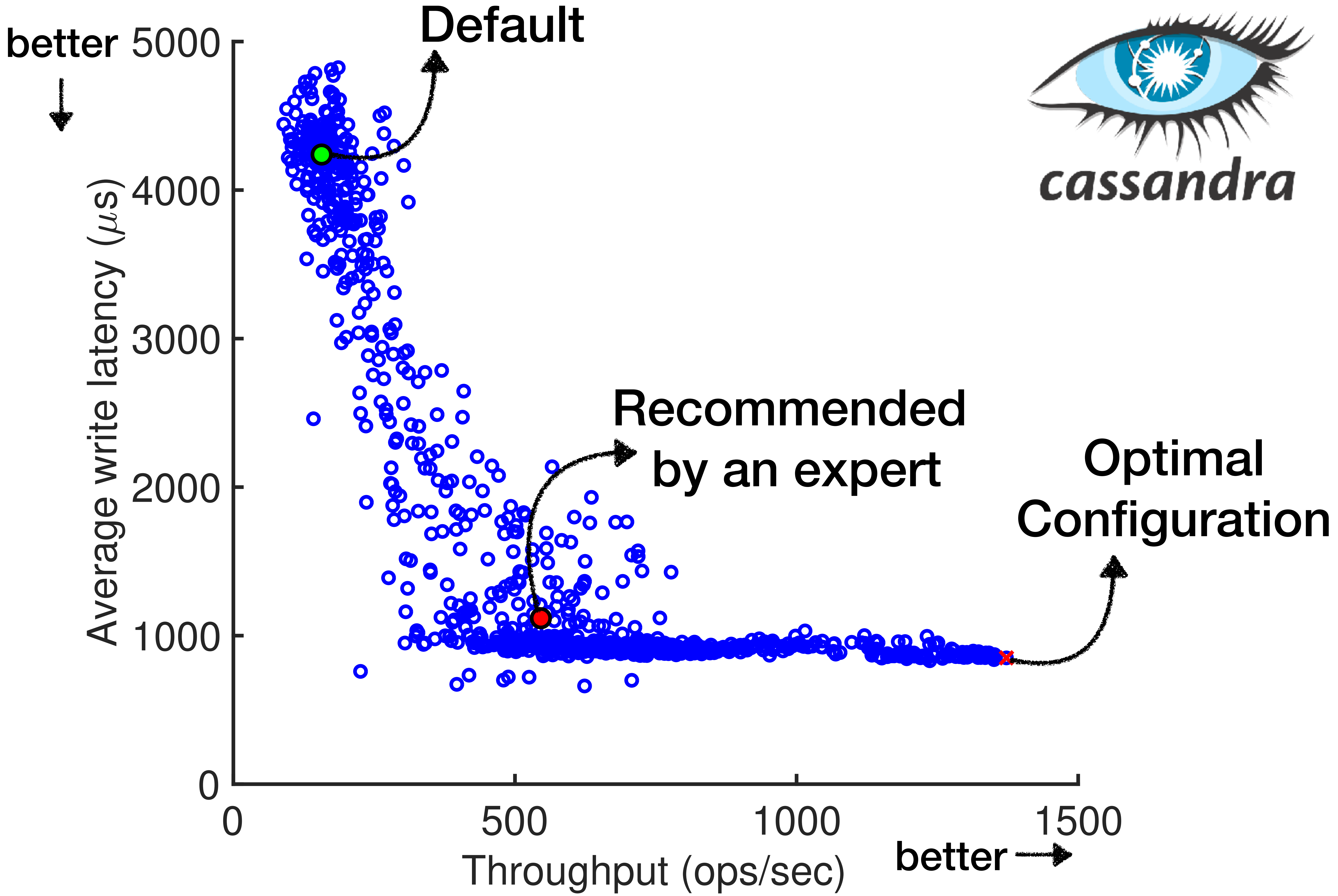
Problem

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

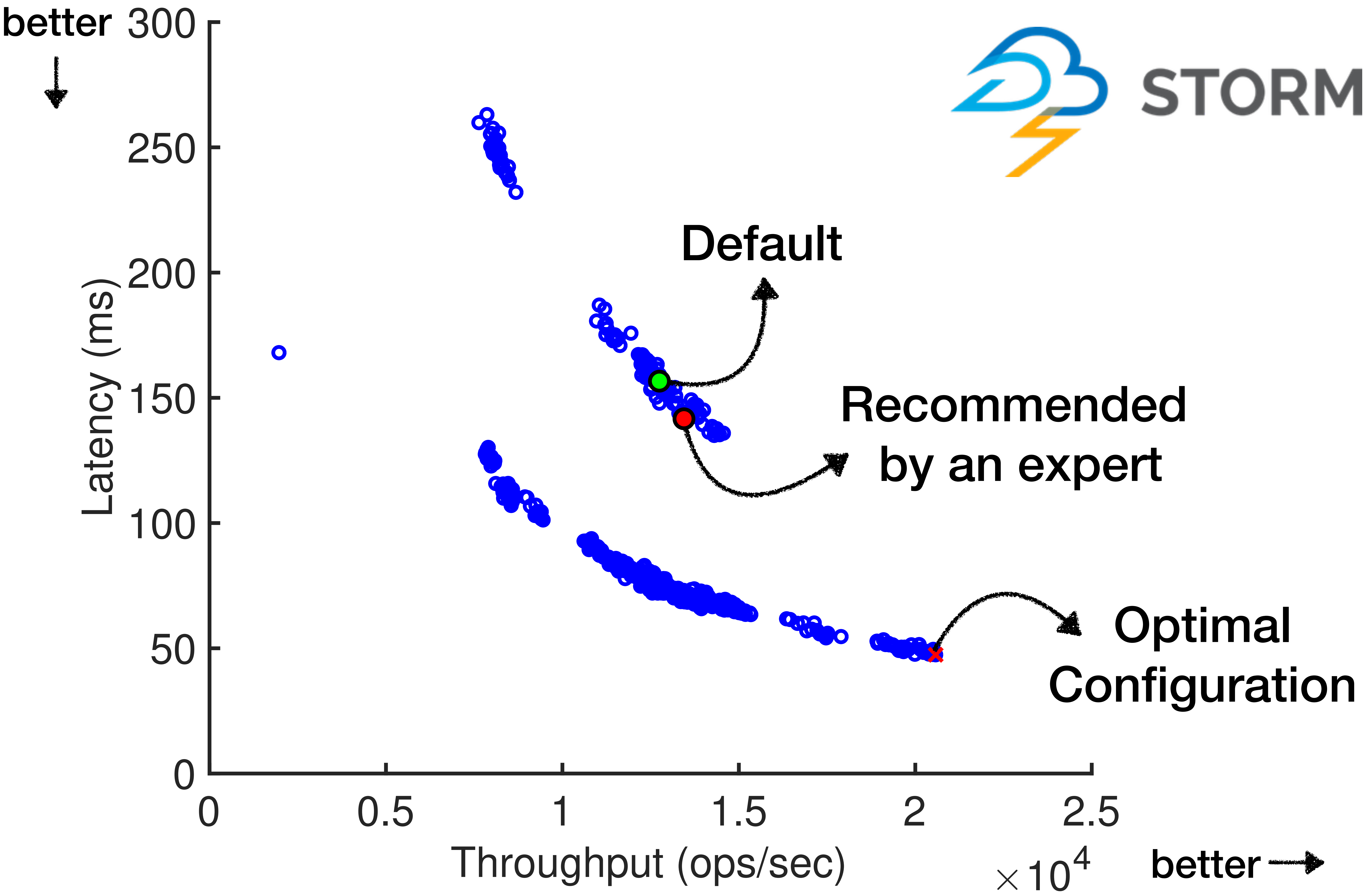
Why?

- 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'

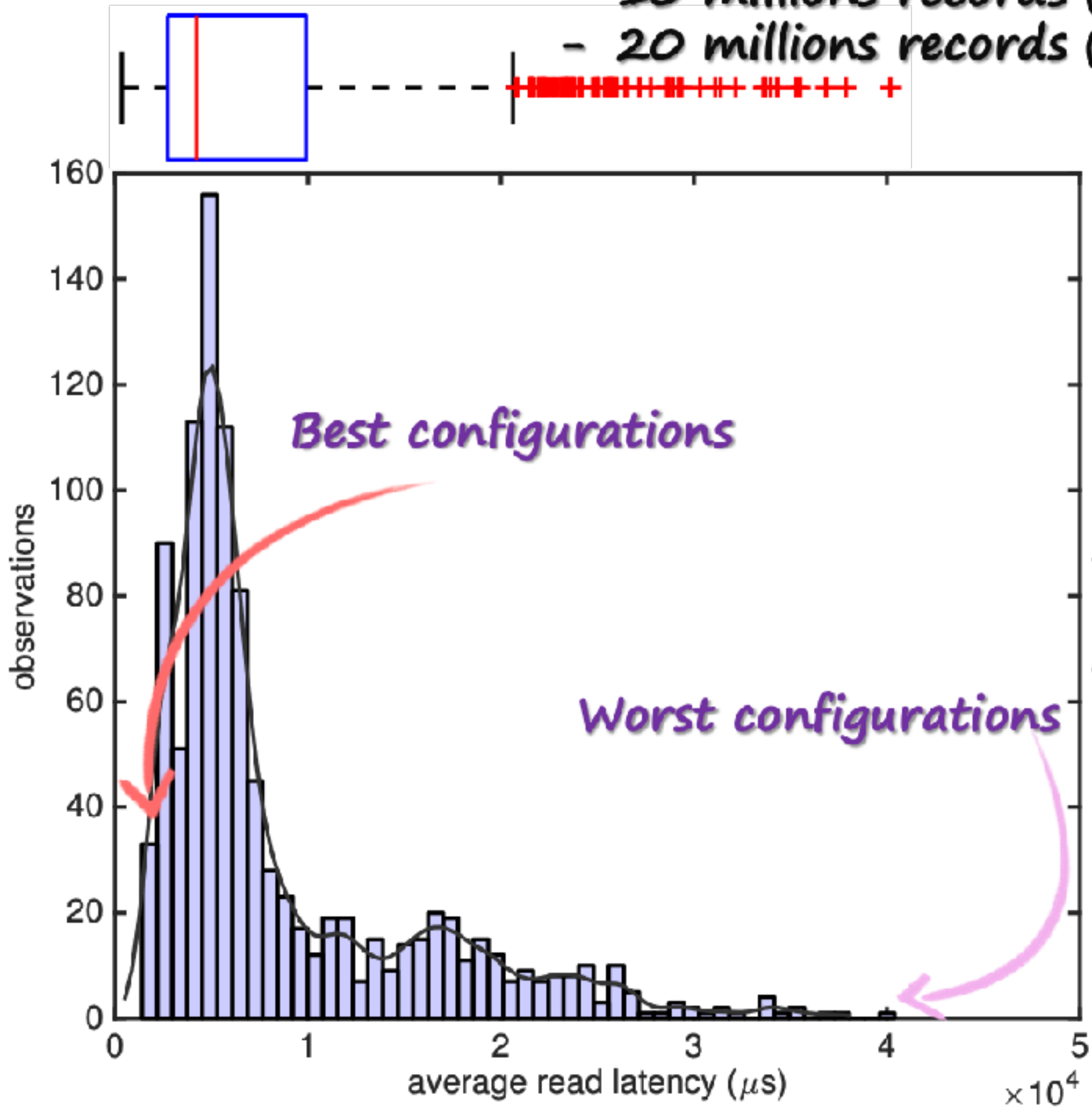


Default configuration was bad, so was the expert'

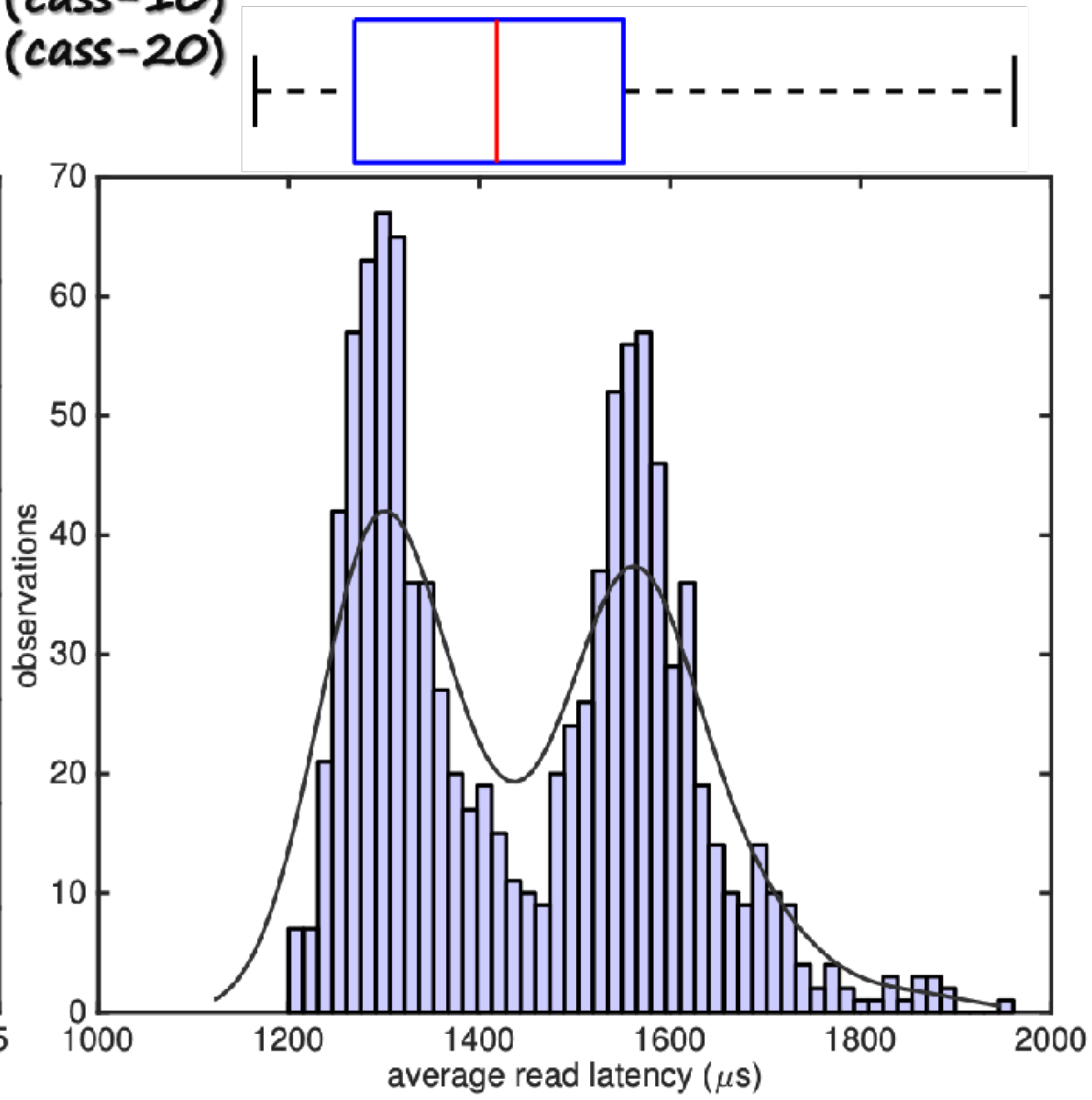


Performance behavior varies in different environments

Experiments on
Apache Cassandra:
- 6 parameters, 1024 configurations
- Average read latency
- 10 millions records (cass-10)
- 20 millions records (cass-20)



(a) cass-20



(b) cass-10

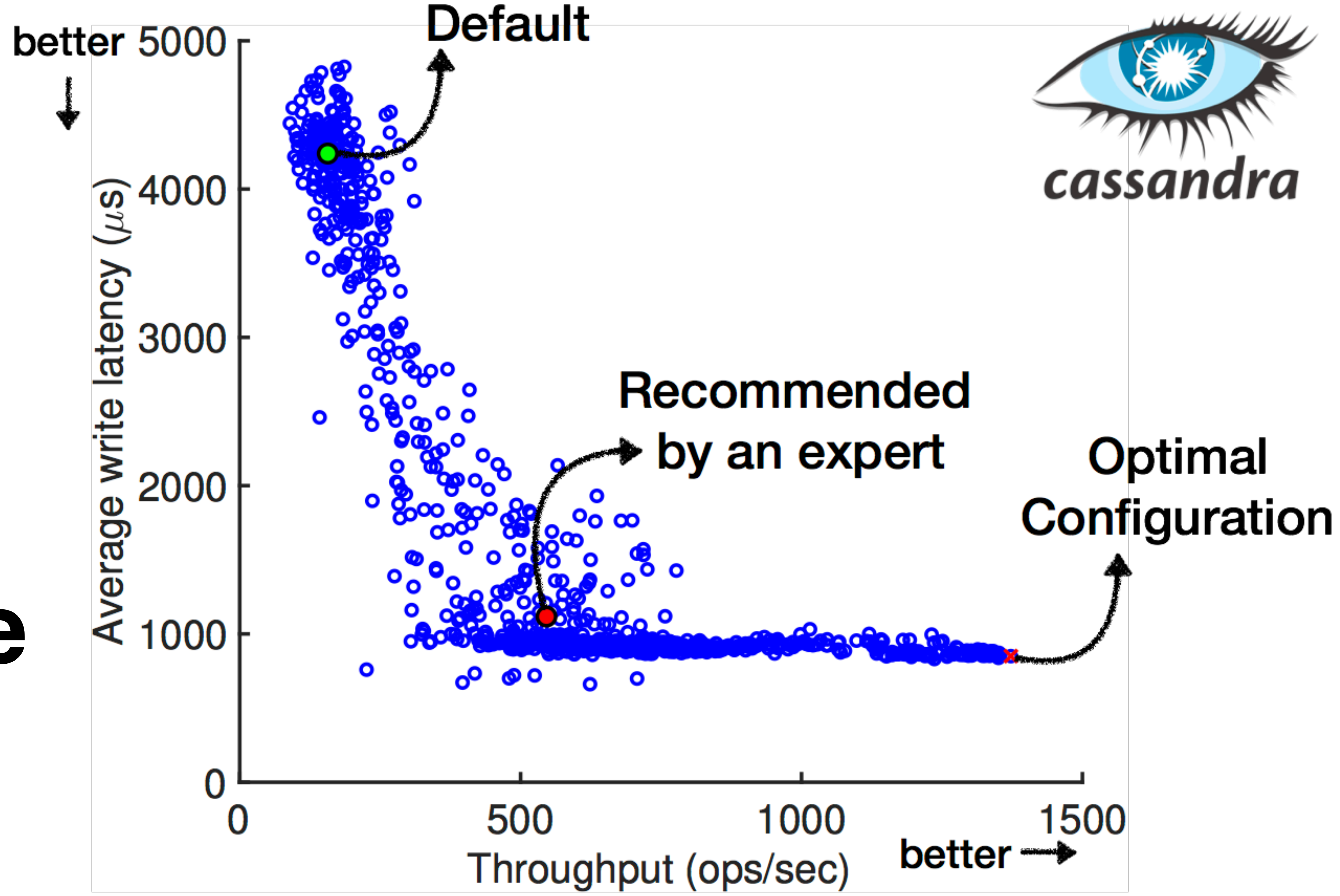
Why this is an important problem?

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.

The system doesn't crash or exhibit an obvious misbehavior

Systems are still operational but with a degraded performance, e.g., high latency, low throughput, high energy consumption, high heat dissipation, or a combination of several

Motivating Example

NVIDIA DEVELOPER

CUDA performance issue on tx2

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

william_wu Jun '17

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

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 nvpmodel=MAX-N
3. Run jetson_clock.sh

The user had several misconfigurations

In Software:

- ✖ Wrong compilation flags
- ✖ Wrong SDK version

In Hardware:

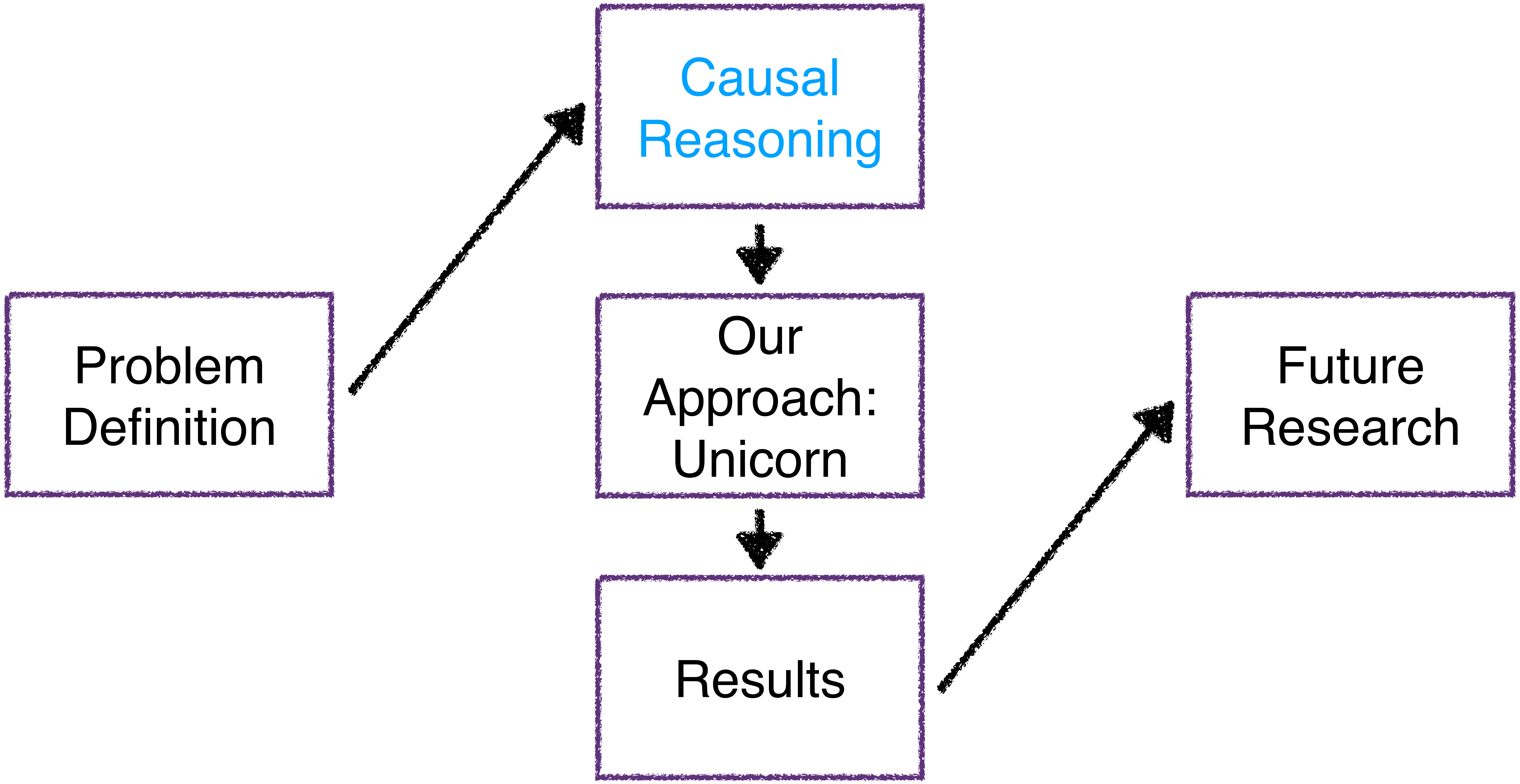
- ✖ Wrong power mode
- ✖ Wrong clock/fan settings

The discussions took 2 days

How to resolve such issues faster?

How to resolve these issues faster?

Outline

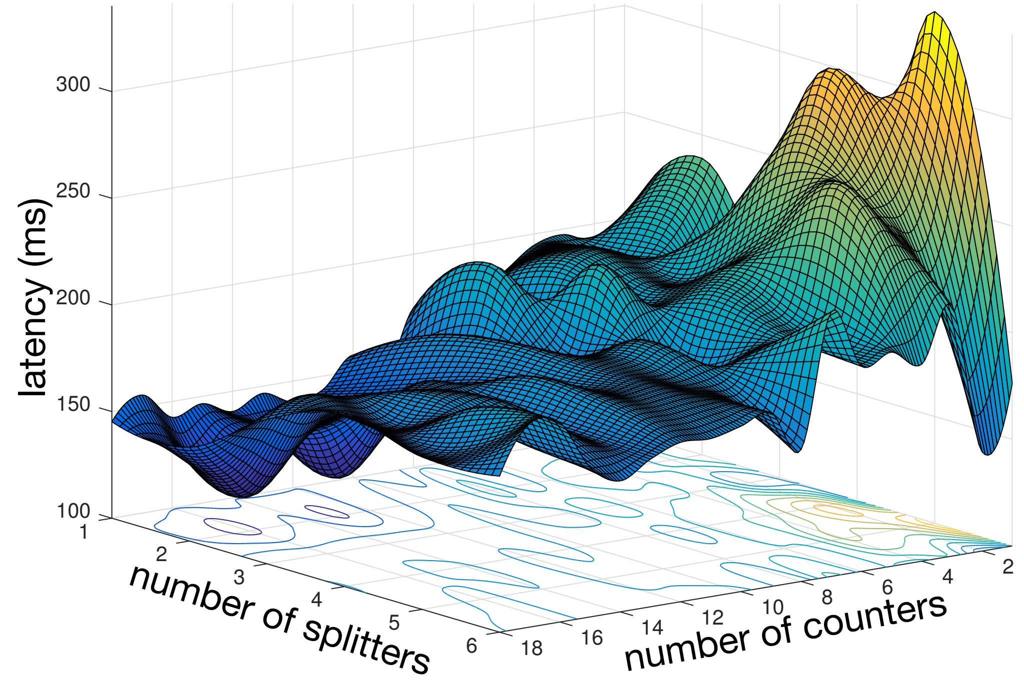

Causal Reasoning
Problem Definition
Our Approach: Unicorn
Future Research
Results

Performance measurement

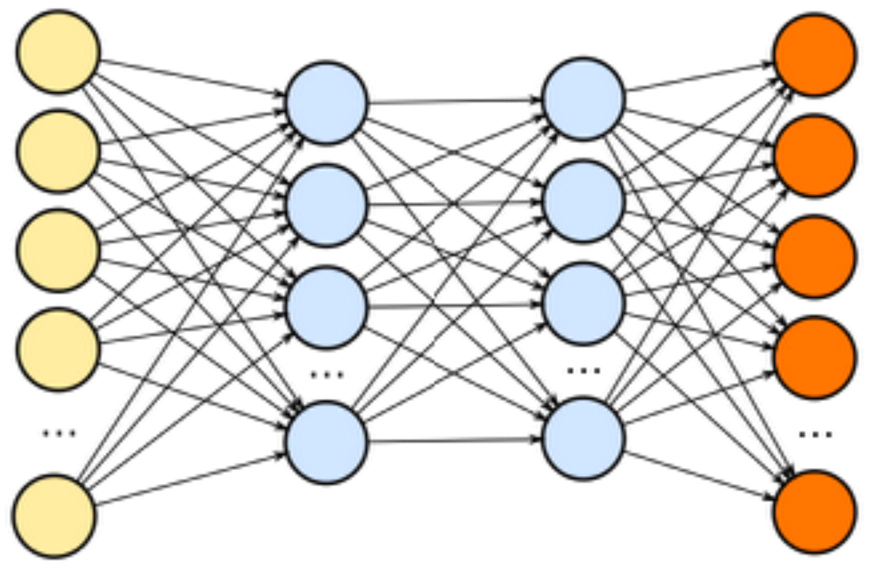
Blackbox Performance Modeling

	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

$$Throughput = \overbrace{\boxed{5.1 \times Bitrate}}^{\text{Options}} + \overbrace{\boxed{2.5 \times BatchSize}}^{\text{Options}}$$
$$+ \underbrace{\boxed{12.3 \times Bitrate \times BatchSize}}_{\text{Interactions}}$$

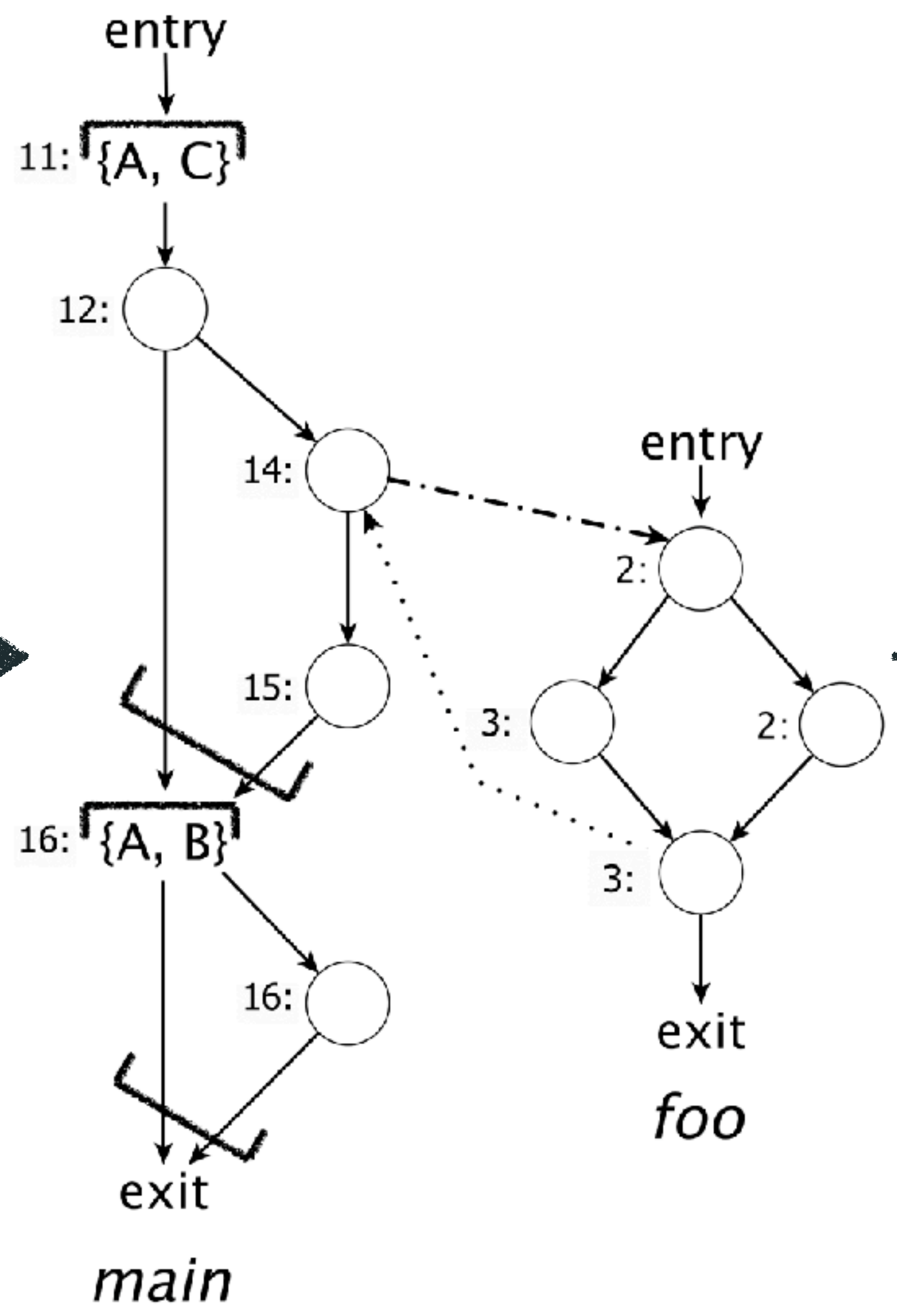
Polynomial Regression

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          ΠR1 = 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;                       ΠR2 = 2A
    // End region R2
    // Begin region R3
    if(b && x) ... // execution: 3s   ΠR3 = 3AB
    // End region R3
```

Identify configuration-dependent regions



Instrumented control flow graph

$$\Pi = 1 + 3A + 3AB + 3AC$$

Build a compositional performance model using local models of each region

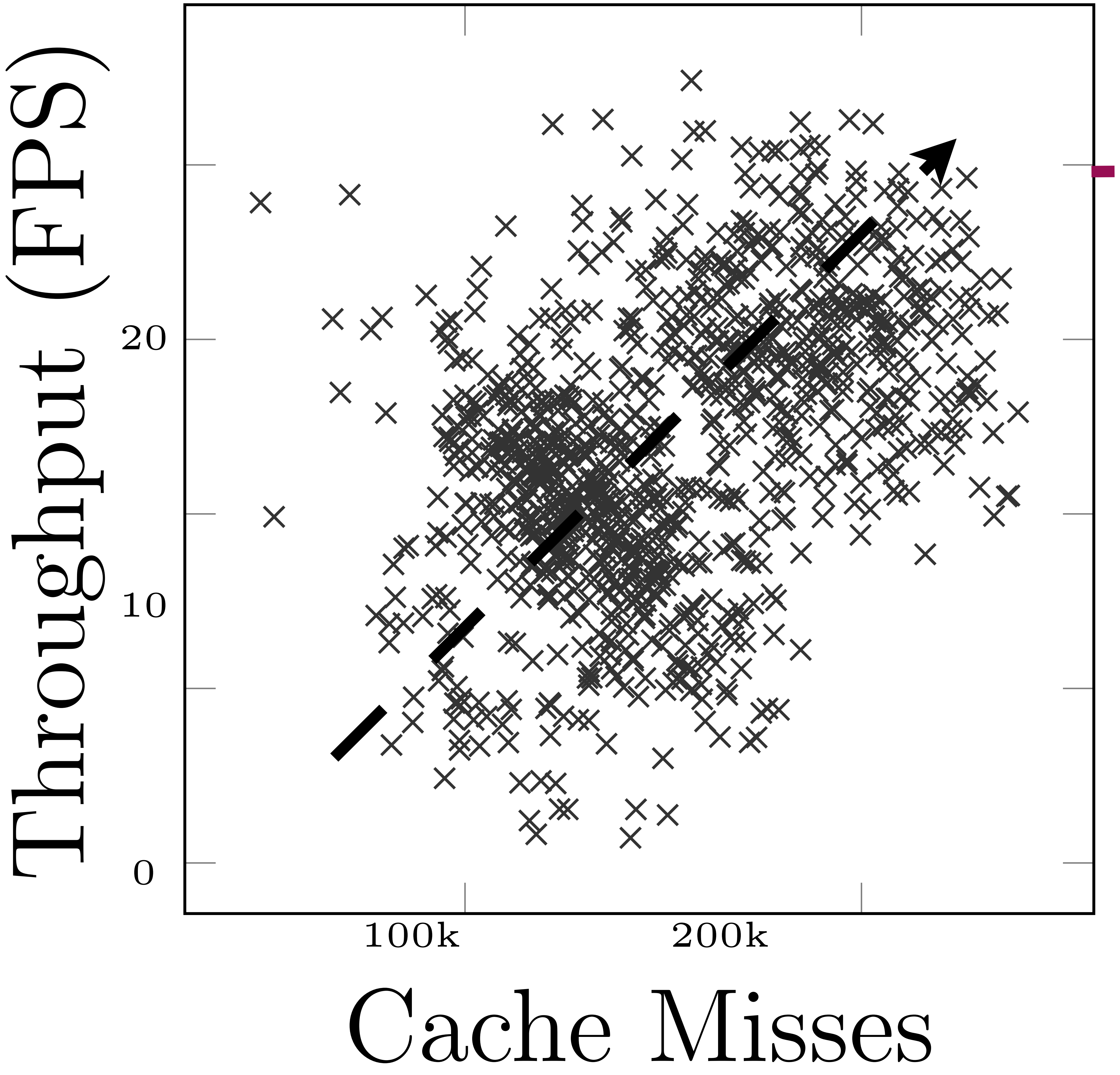
These methods rely on program analysis techniques (static and dynamic analysis of the code) to extract meaningful information required for performance tasks.

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.

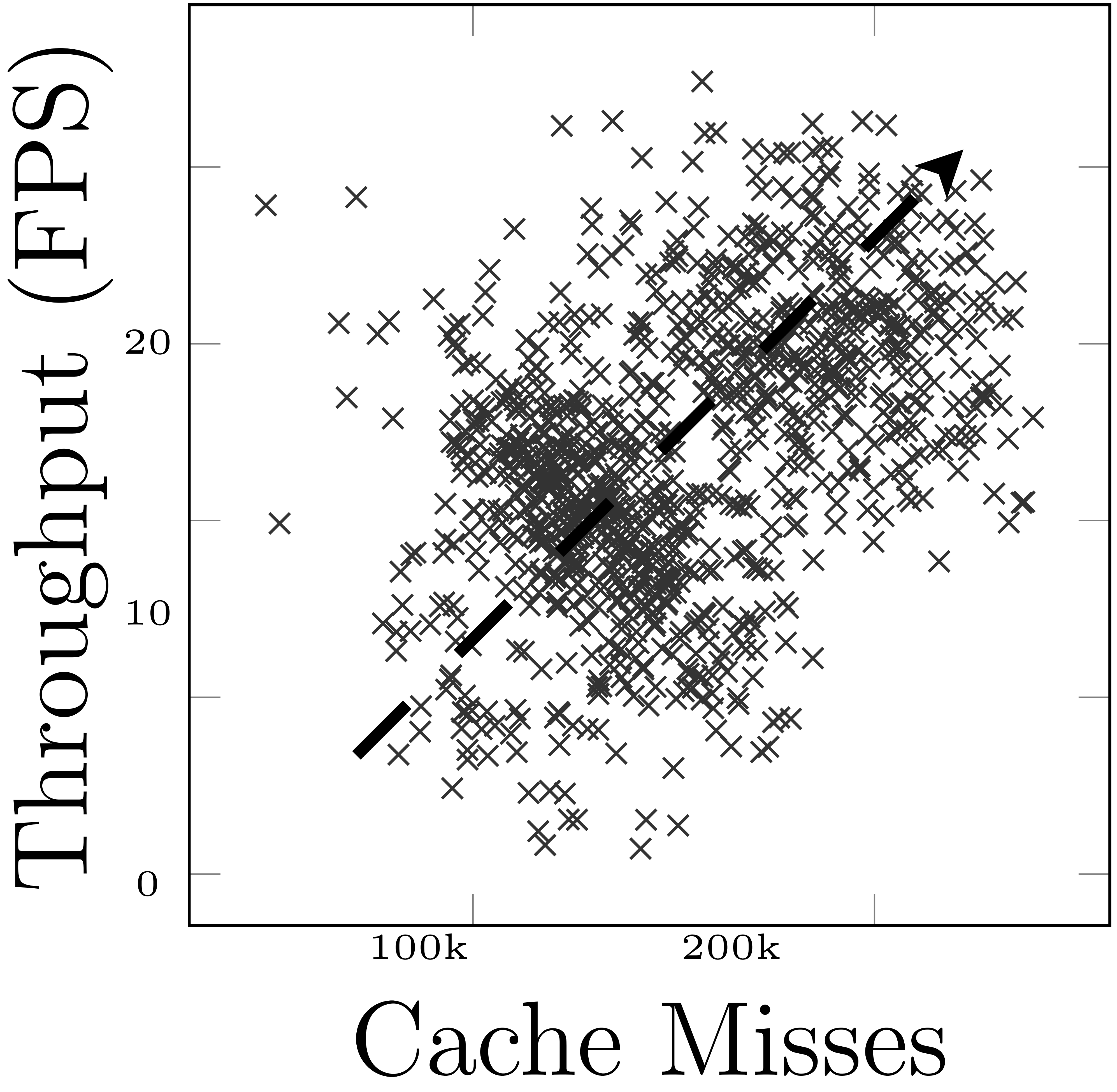
- Whitebox performance models do not scale well to real-world systems (with many configuration options and large code bases.

Incorrect explanation



Increasing Cache Misses increases Throughput.

Incorrect explanation



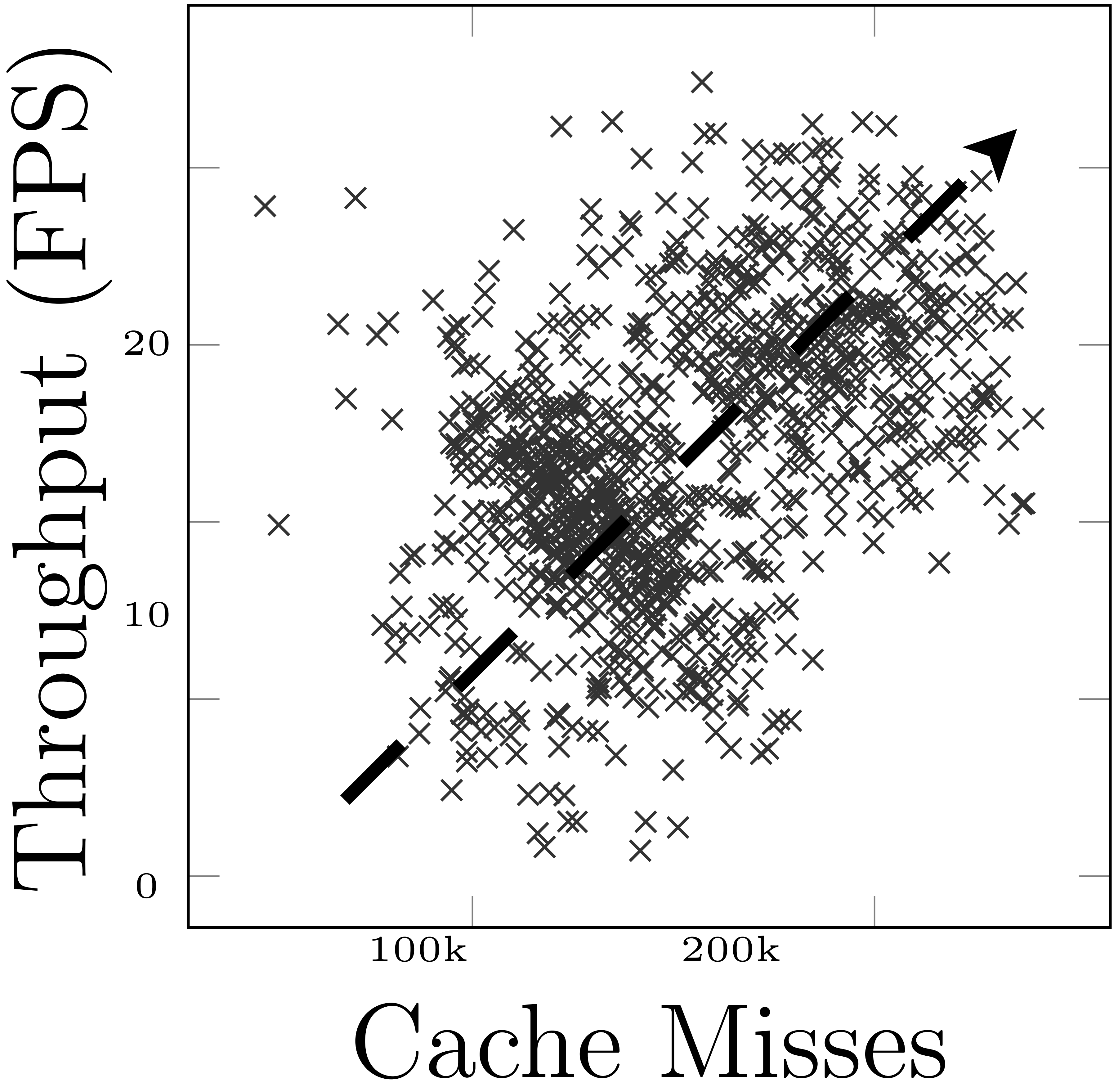
Increasing Cache Misses increases Throughput.

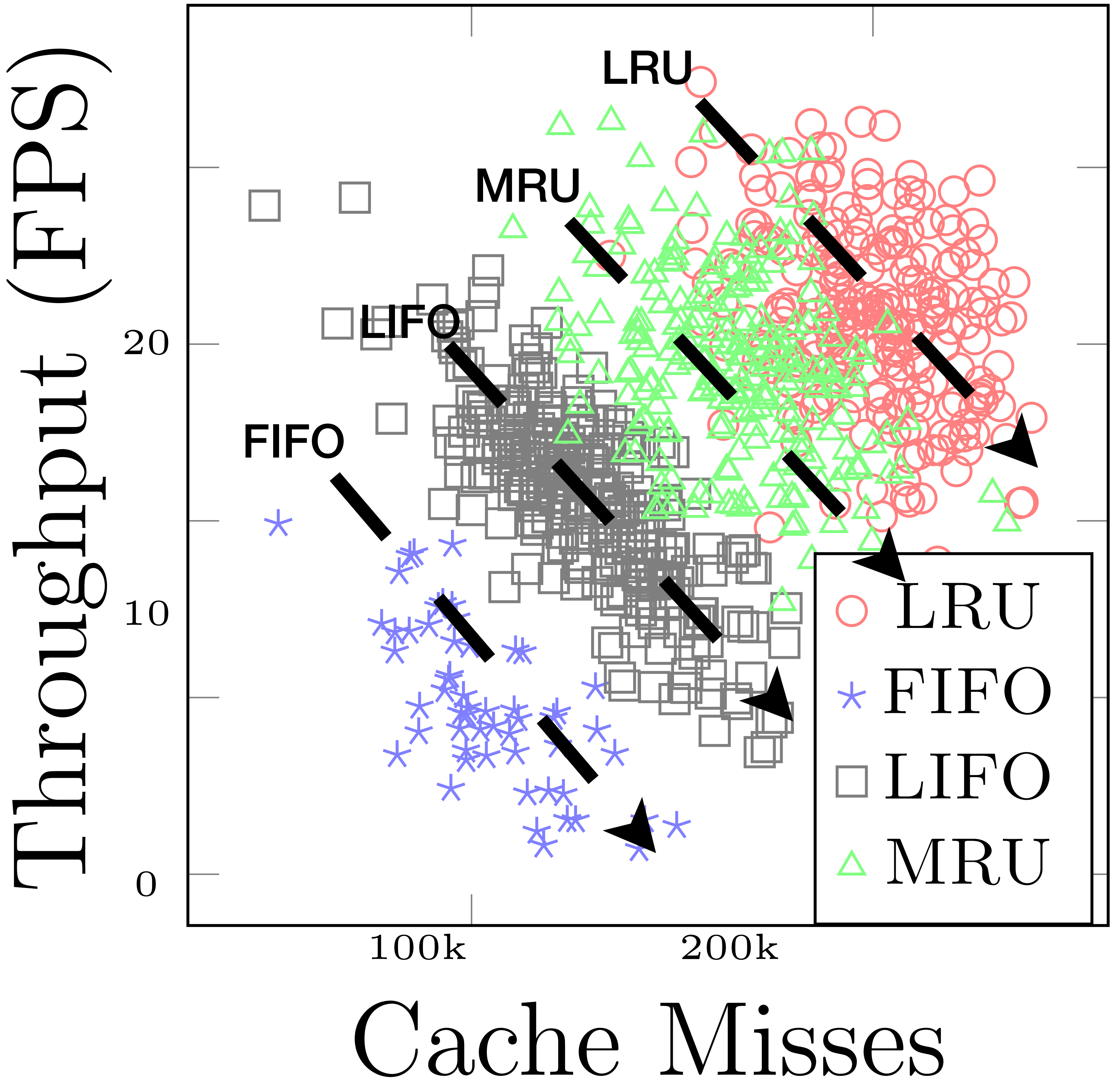
This is counter-intuitive

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:

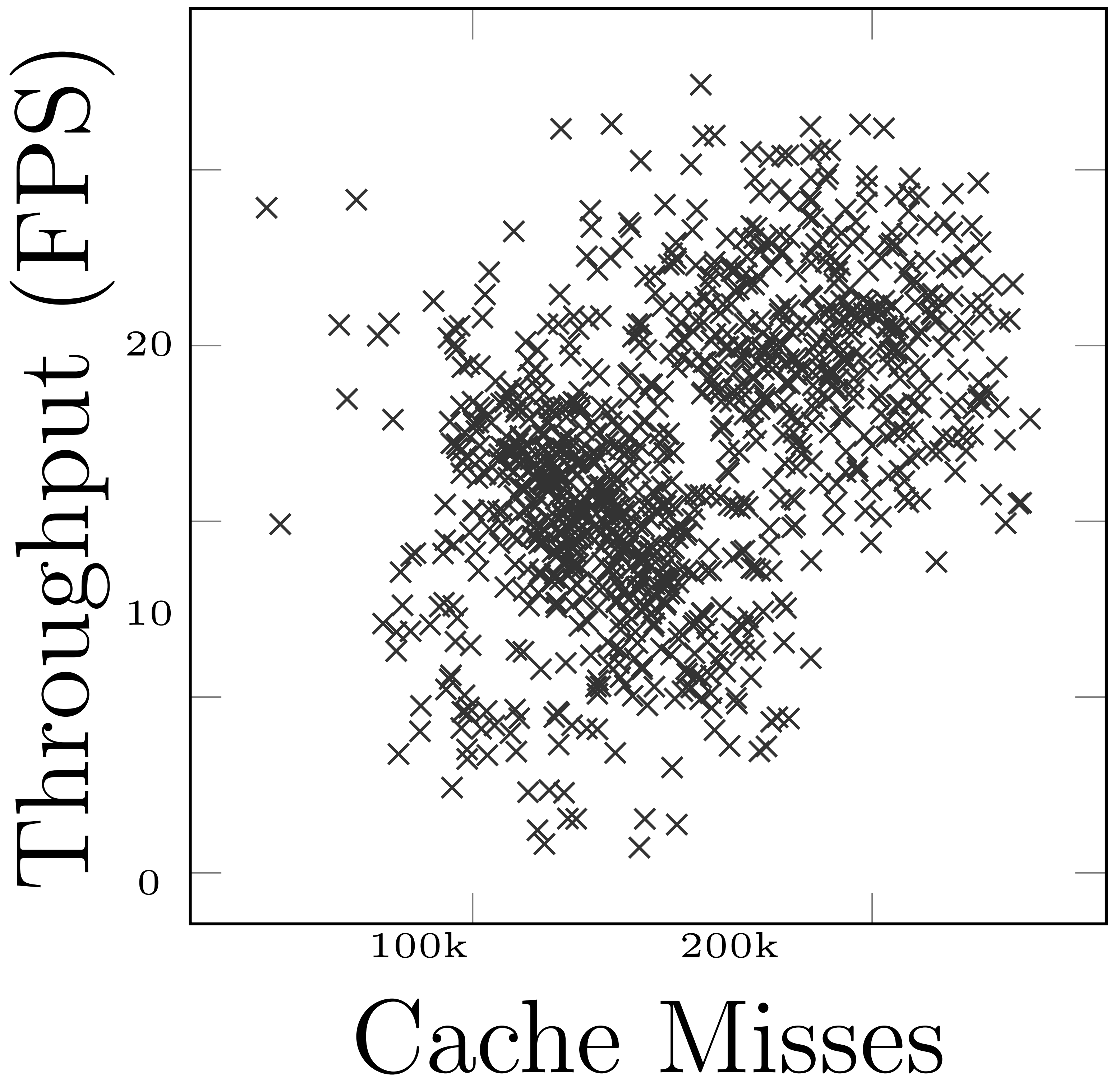
$$Throughput = 2 \times Bitrate + 1.9 \times BatchSize + \boxed{1.8 \times BufferSize} + \boxed{0.5 \times EnablePadding} + \boxed{5.9 \times Bitrate \times BufferSize}$$
$$+ \boxed{6.2 \times Bitrate \times EnablePadding} + \boxed{4.1 \times Bitrate \times BufferSize \times EnablePadding}$$

Performance influence model in Xavier:

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

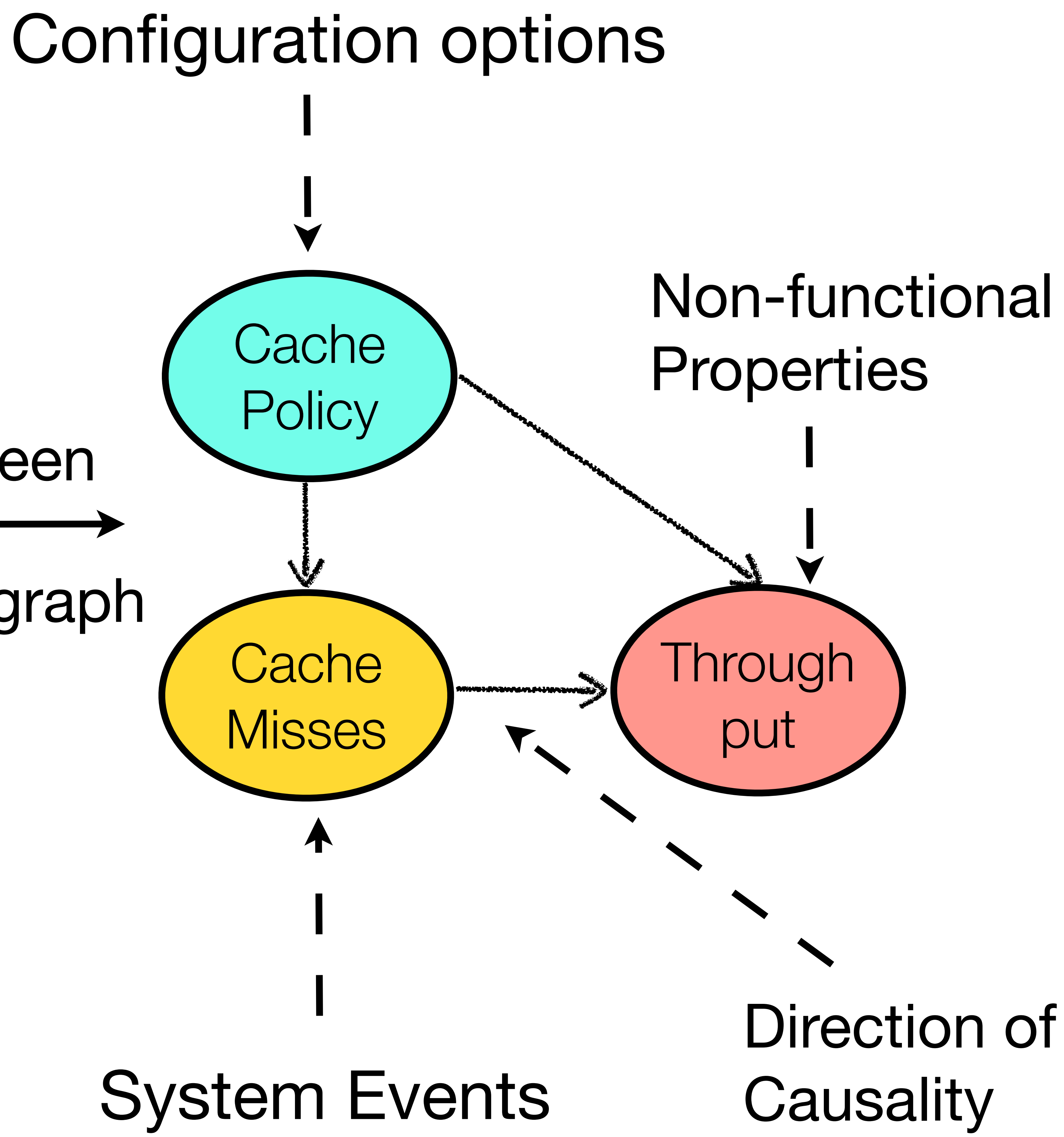
Performance influence models change significantly across environments, resulting in low accuracy in new environments.

Causal performance modeling

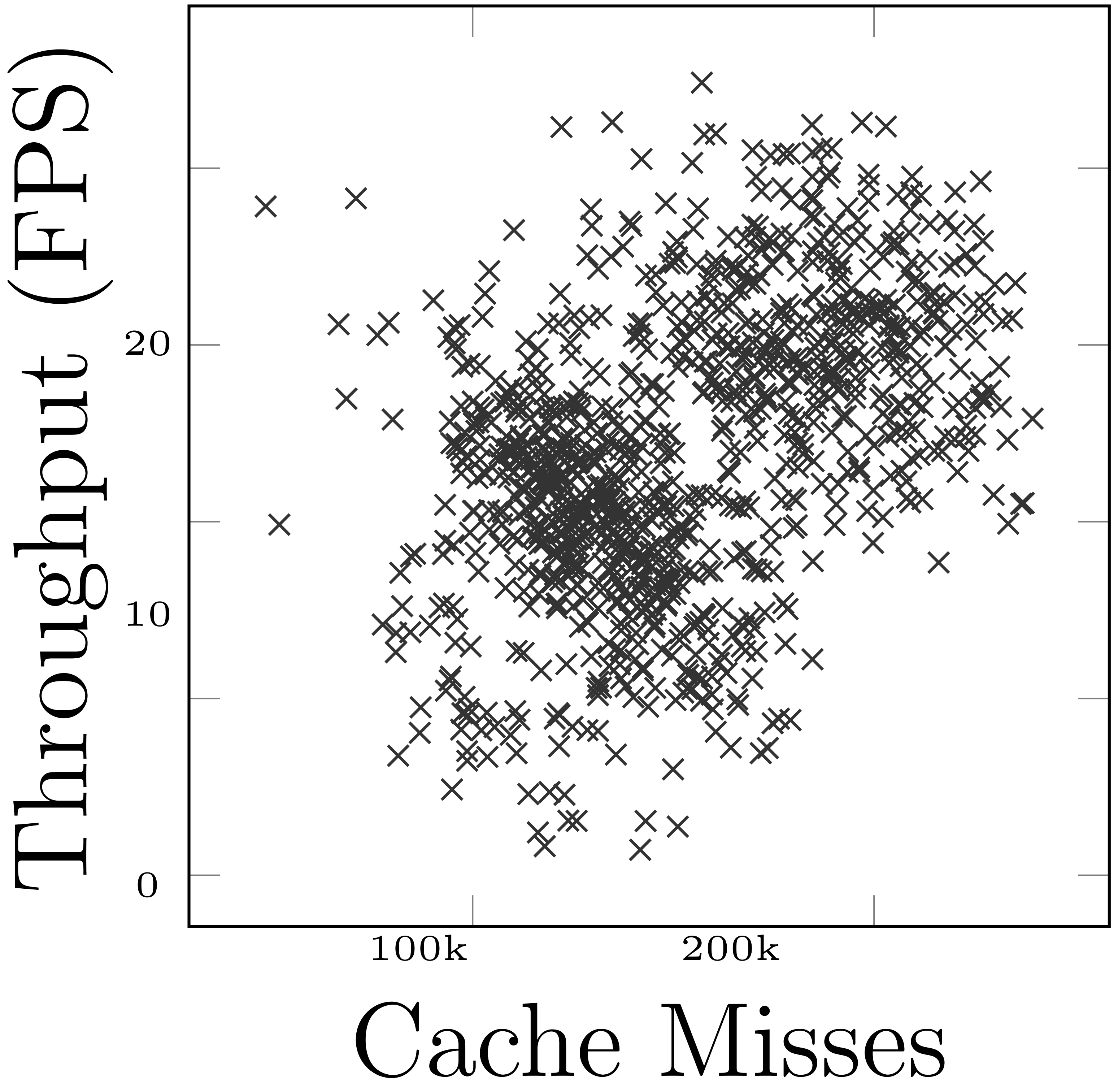


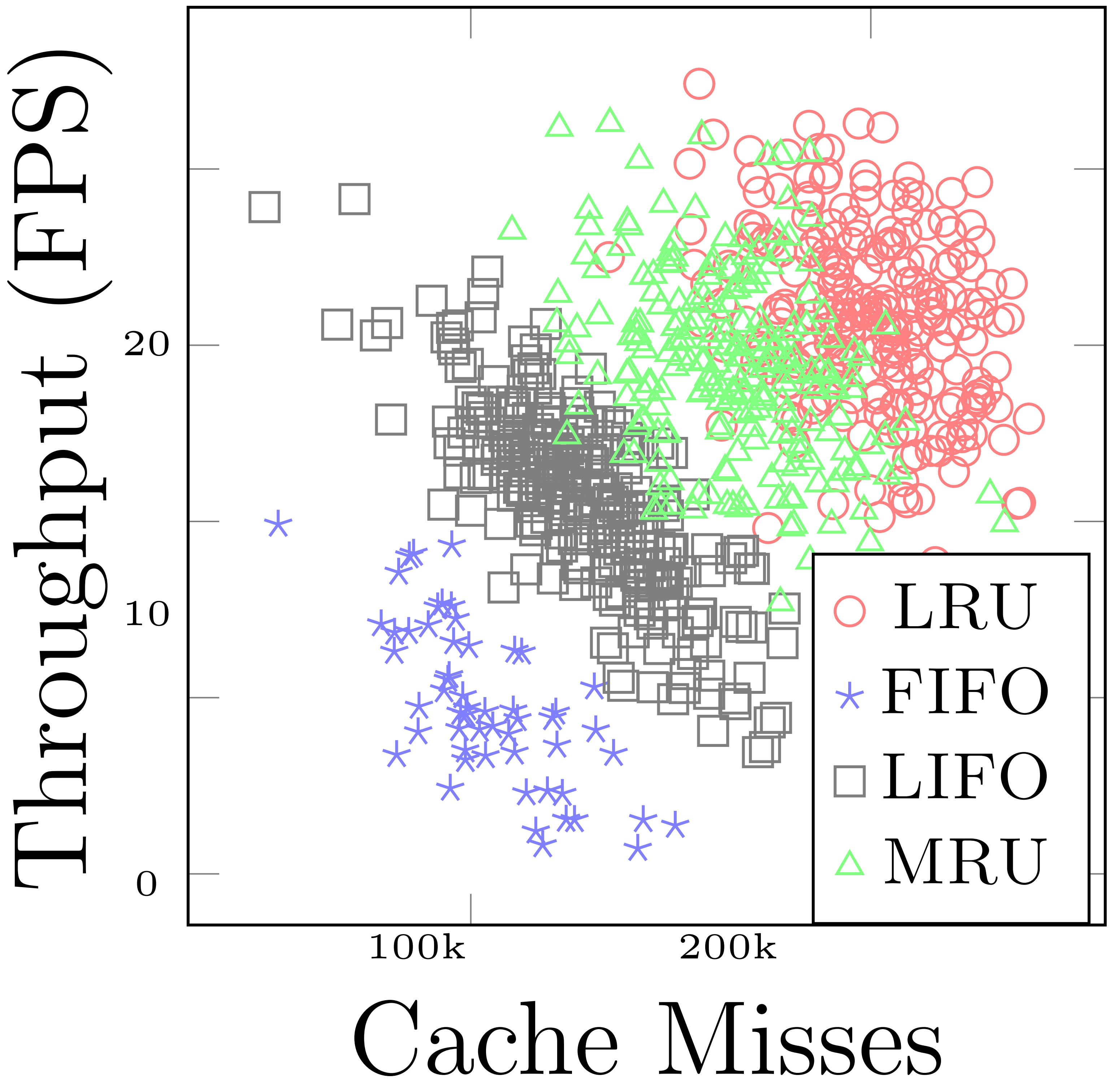
Expresses the relationships between

interacting variables as a causal graph

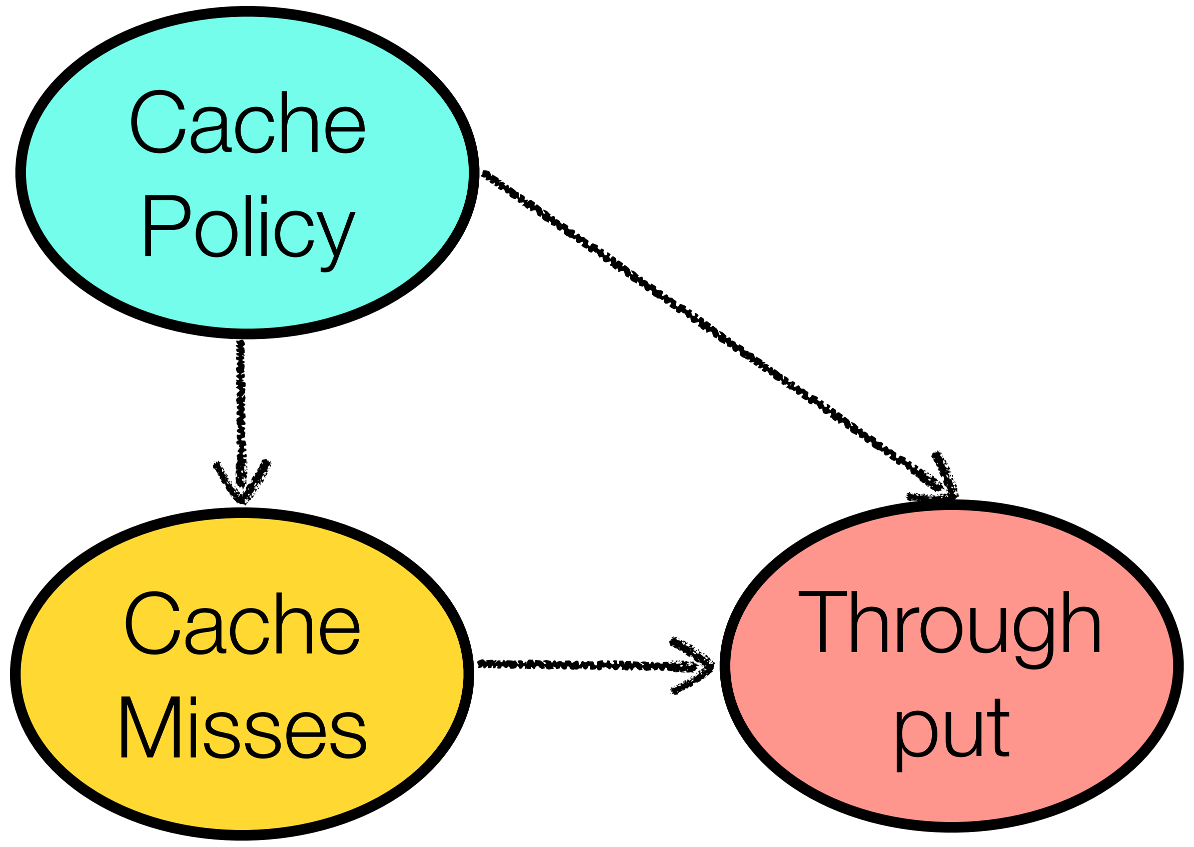


Causal performance models produce correct explanations



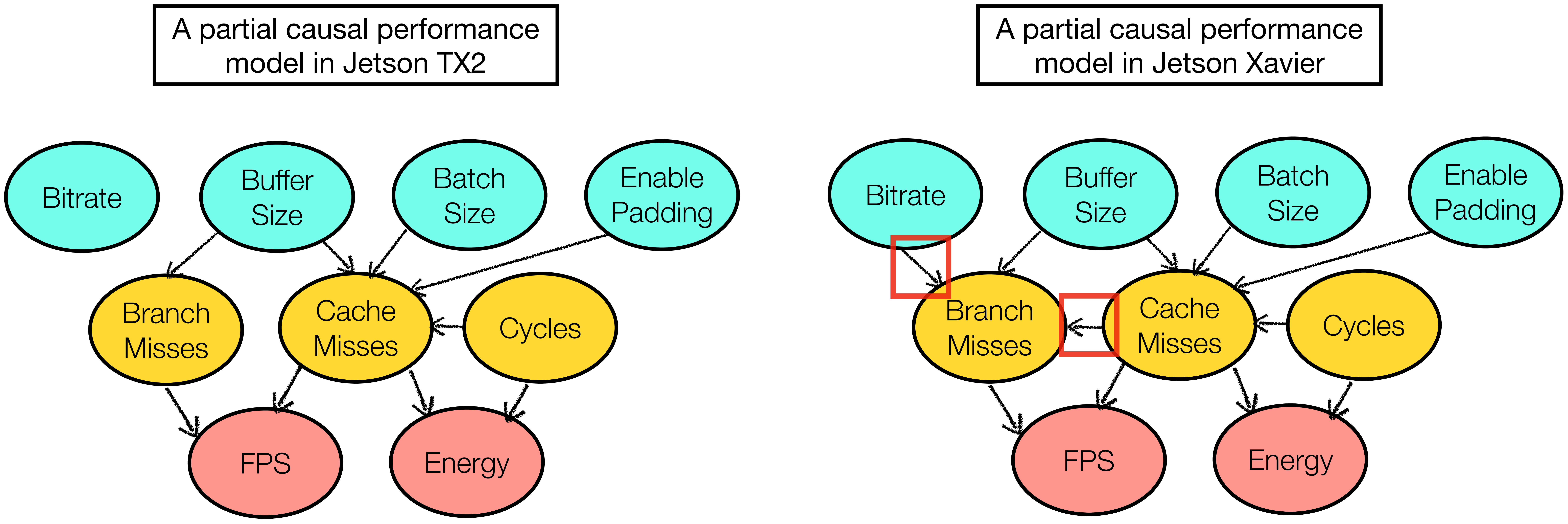


Cache Policy affects Throughput via Cache Misses.



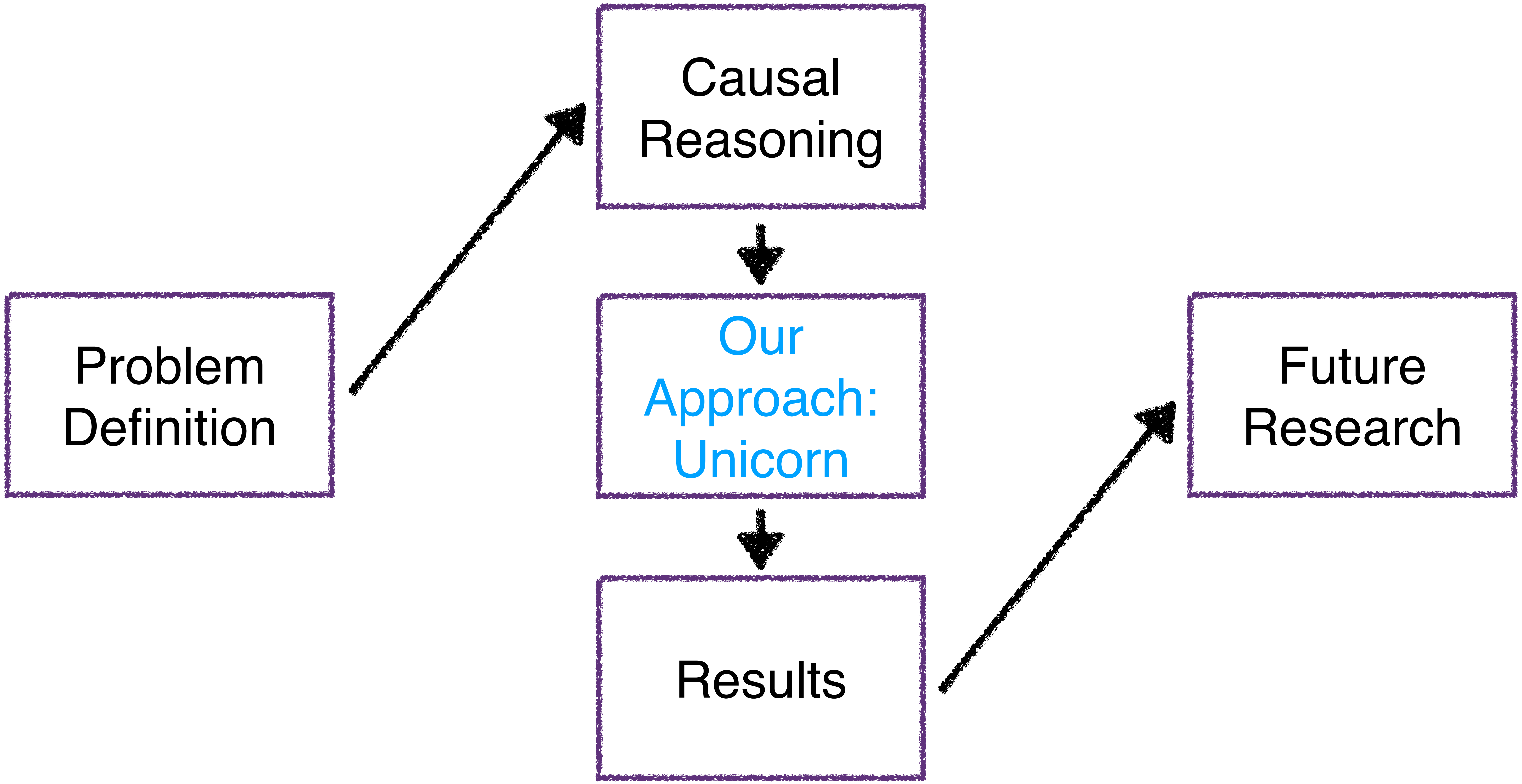
Causal performance models **capture correct interactions**.

Causal performance models are transferable across environments



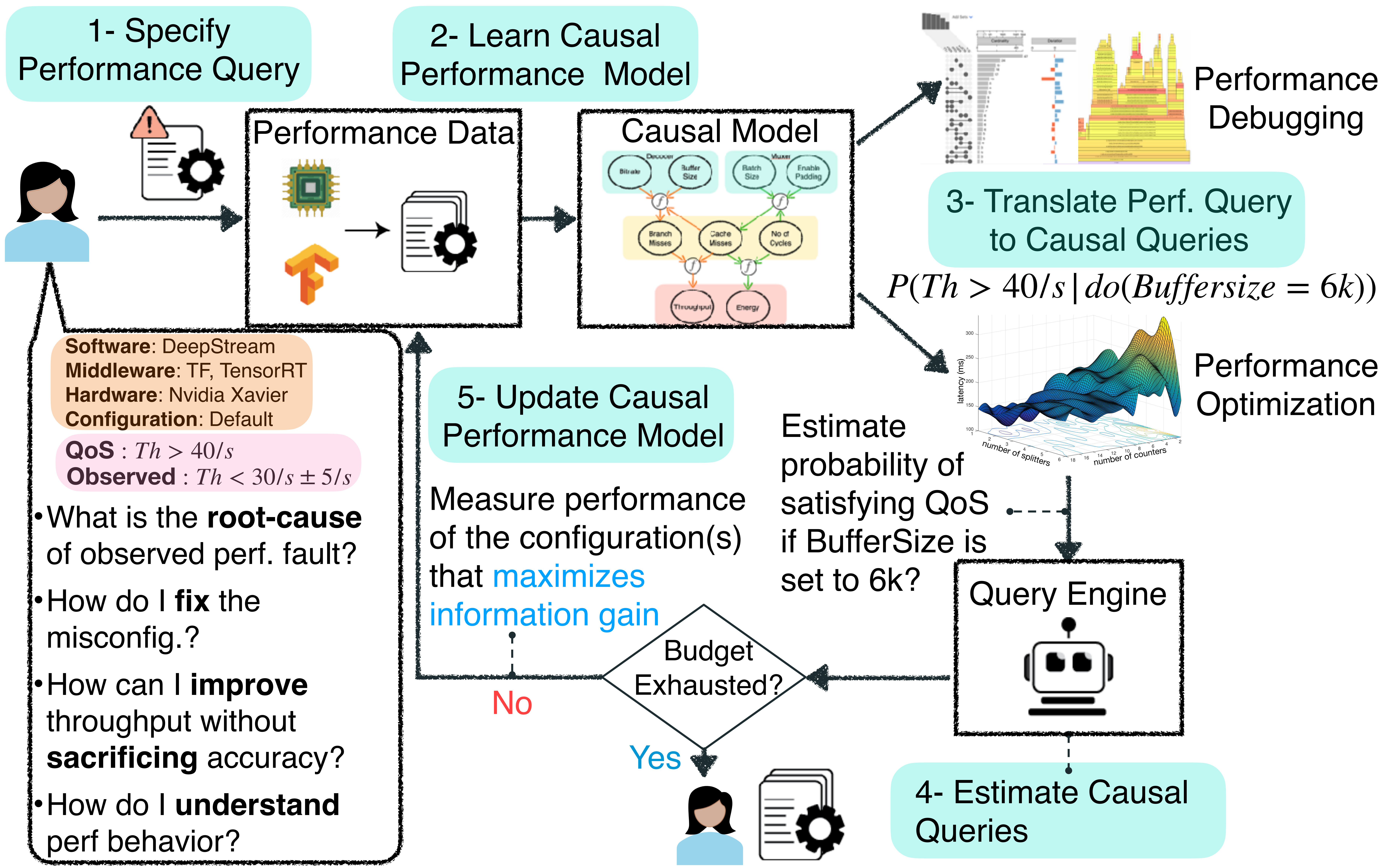
Causal performance models remain relatively stable across environments.

Outline

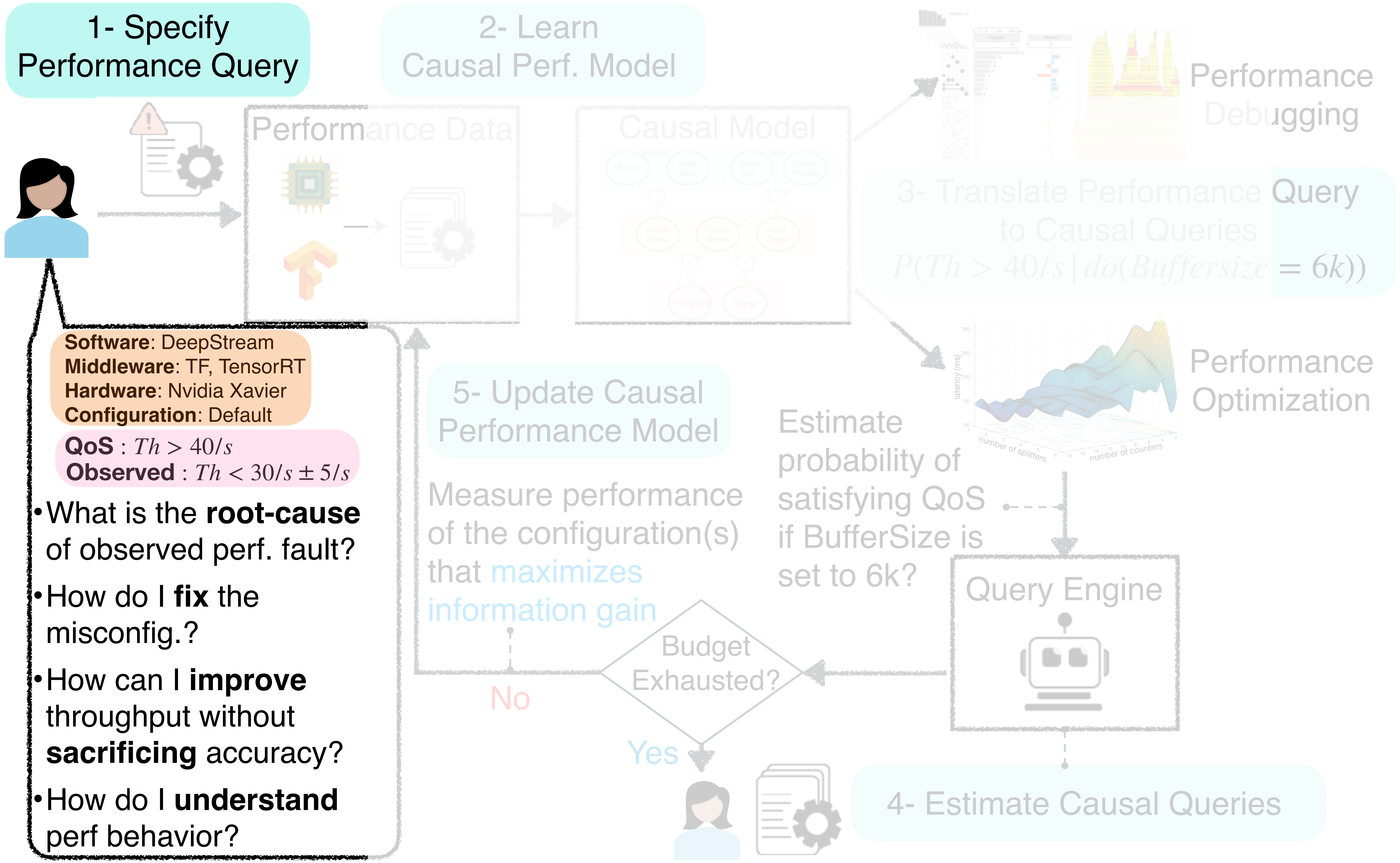


Problem Definition
Causal Reasoning
Our Approach: Unicorn
Results
Future Research

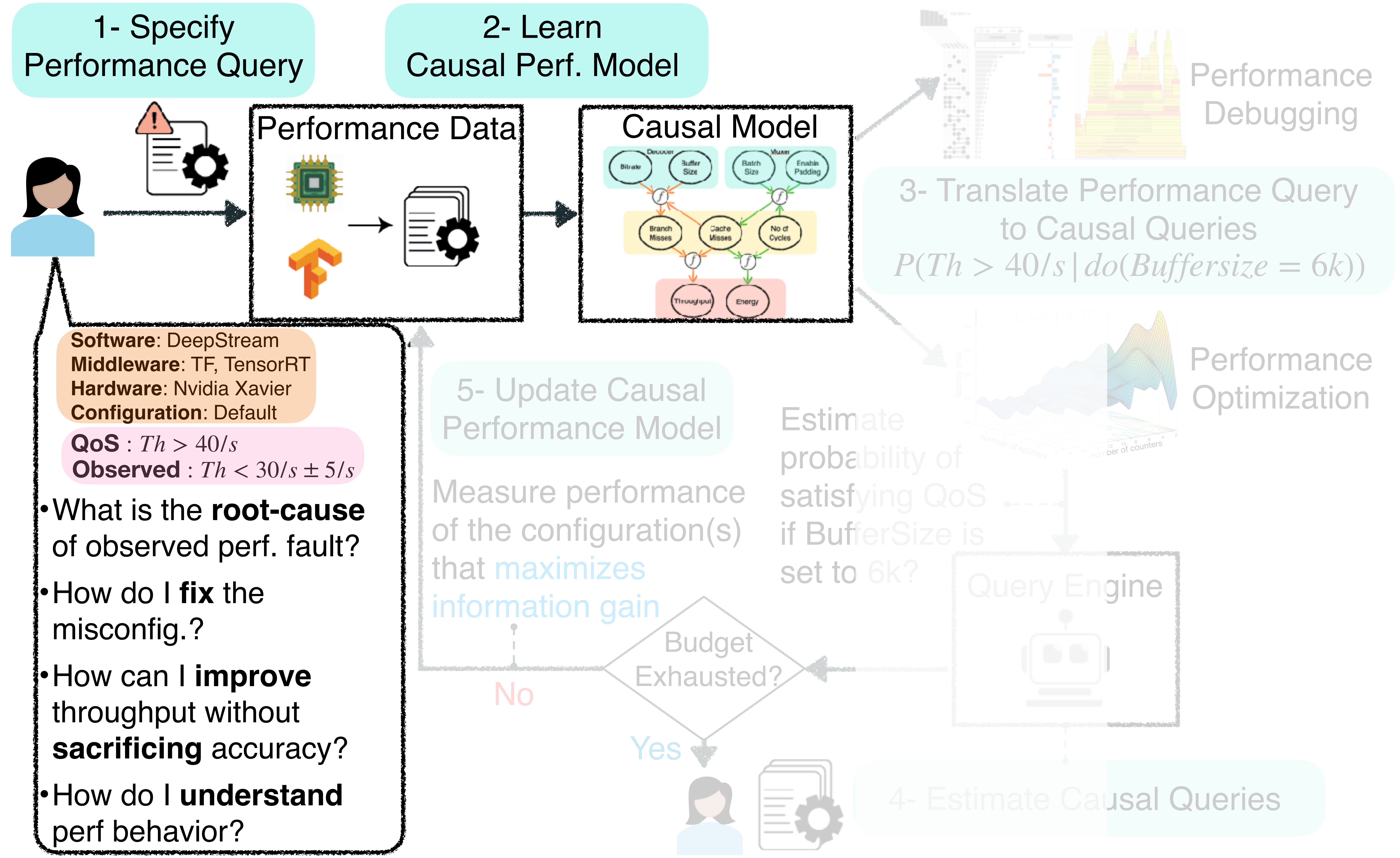
UNICORN: Performance Reasoning through the Lens of Causality



UNICORN: Performance Reasoning through the Lens of Causality



UNICORN: Performance Reasoning through the Lens of Causality

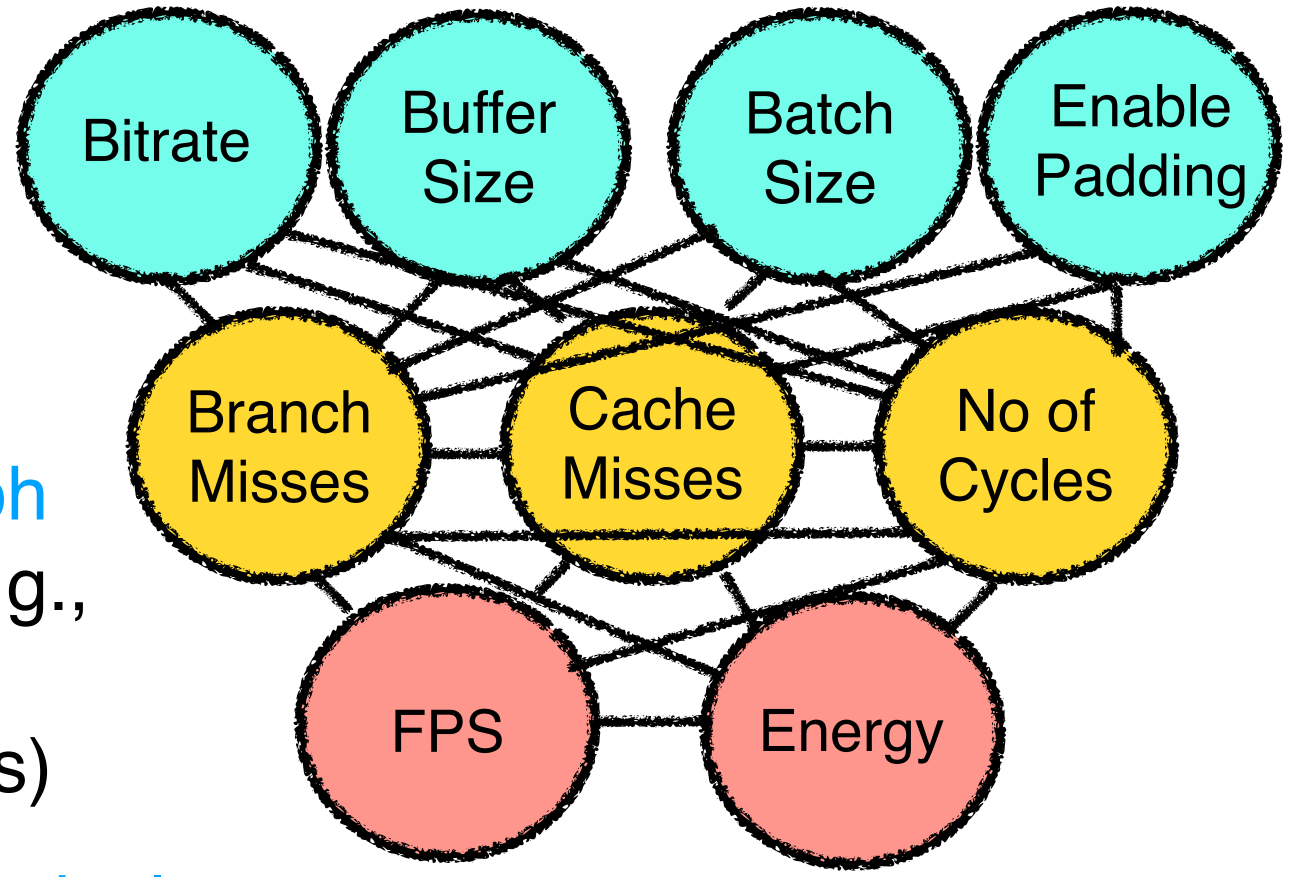


Learning Causal Performance Model

	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

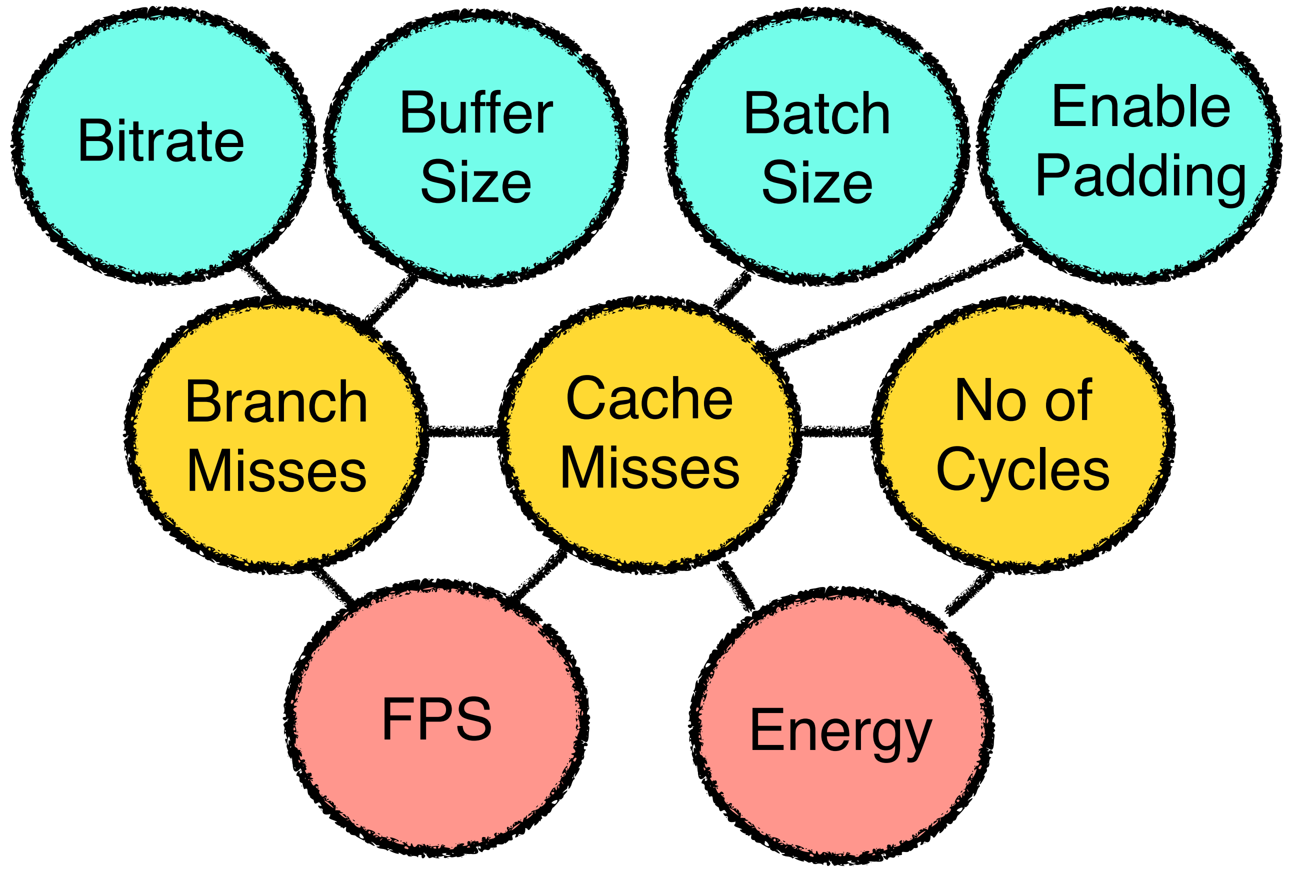
1- Recovering the Skelton

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



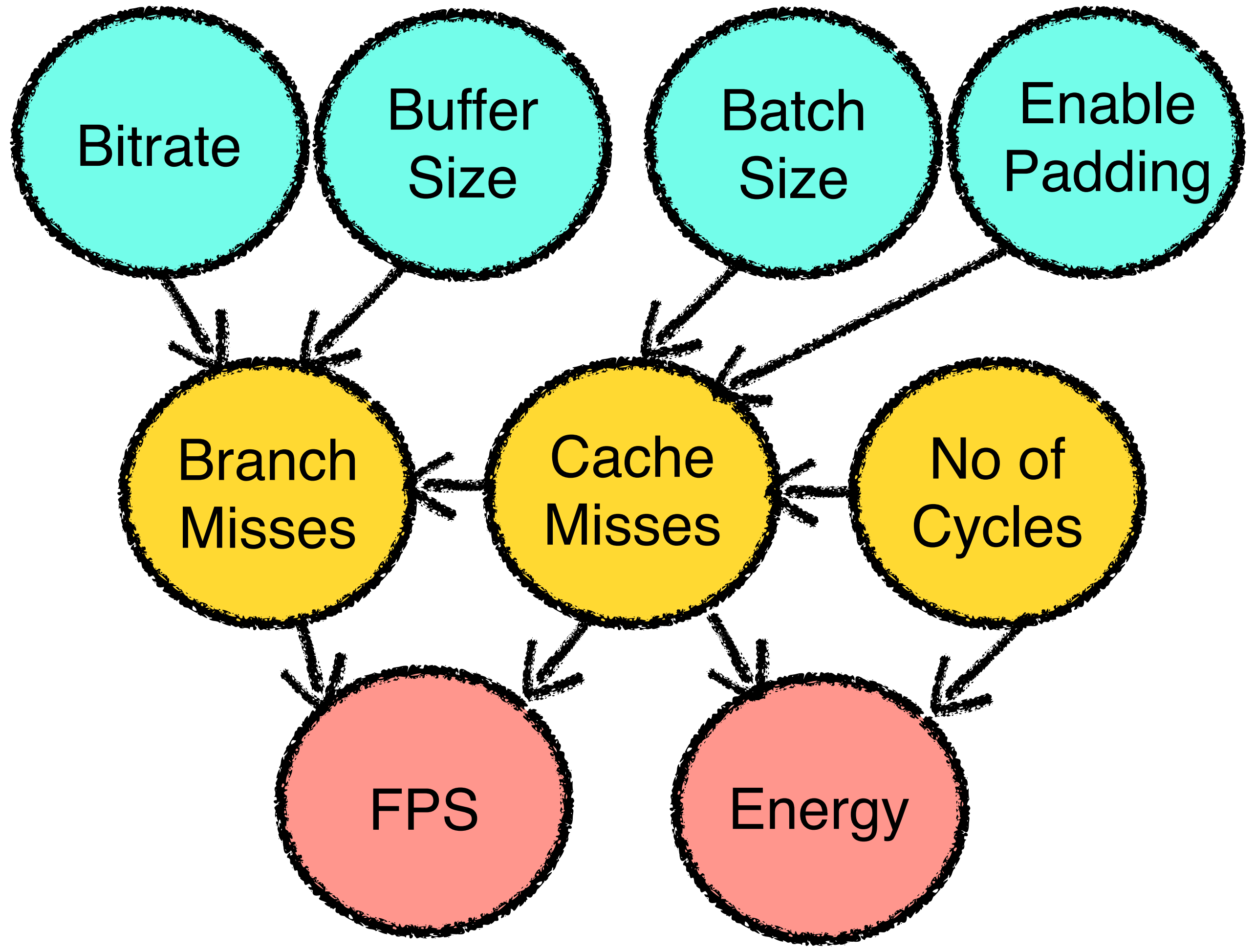
statistical independence tests

2- Pruning Causal Structure



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

3- Orienting Causal Relations

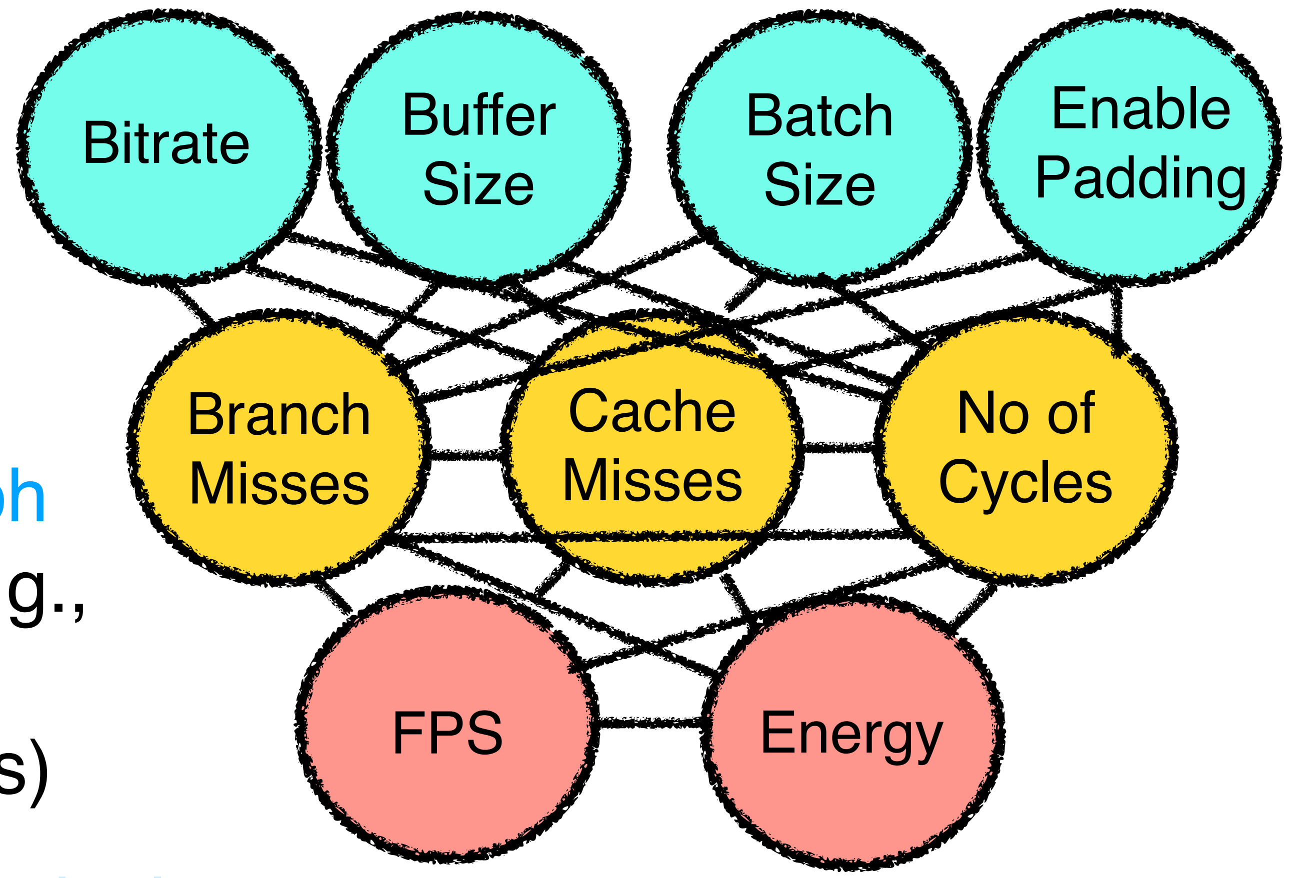


Learning Causal Performance Model

	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 tests

2- Pruning Causal Structure

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

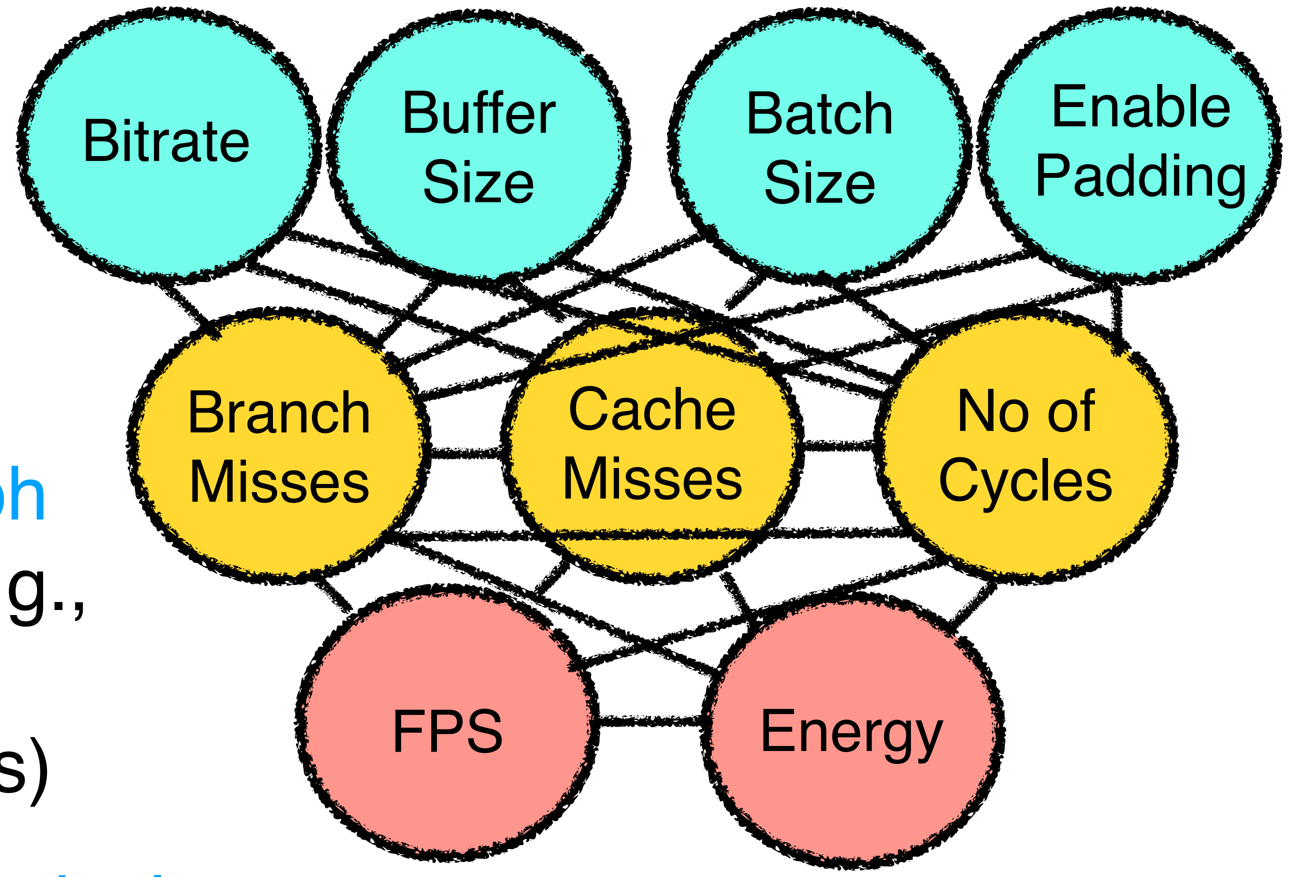
3- Orienting Causal Relations

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

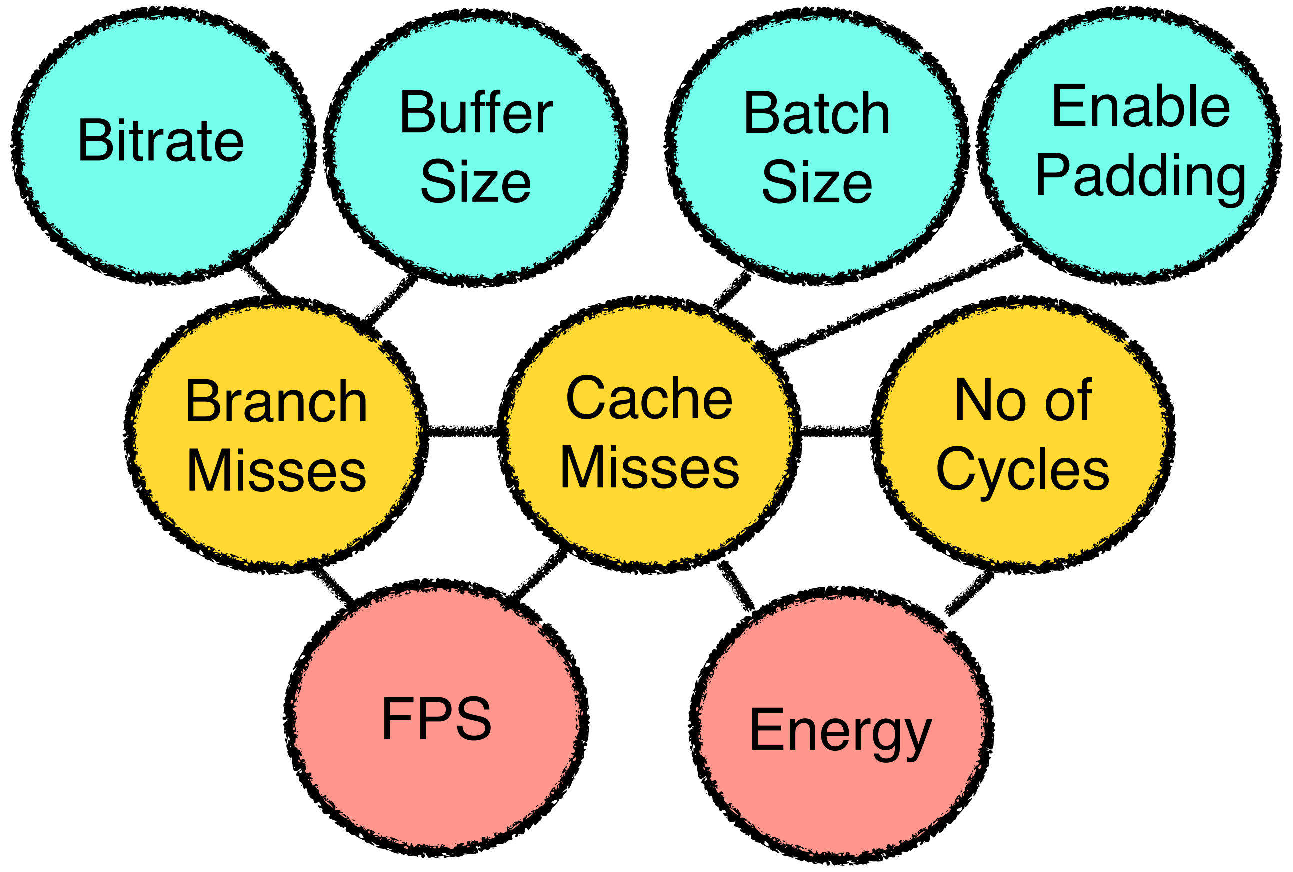
1- Recovering the Skelton

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



statistical independence tests

2- Pruning Causal Structure



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

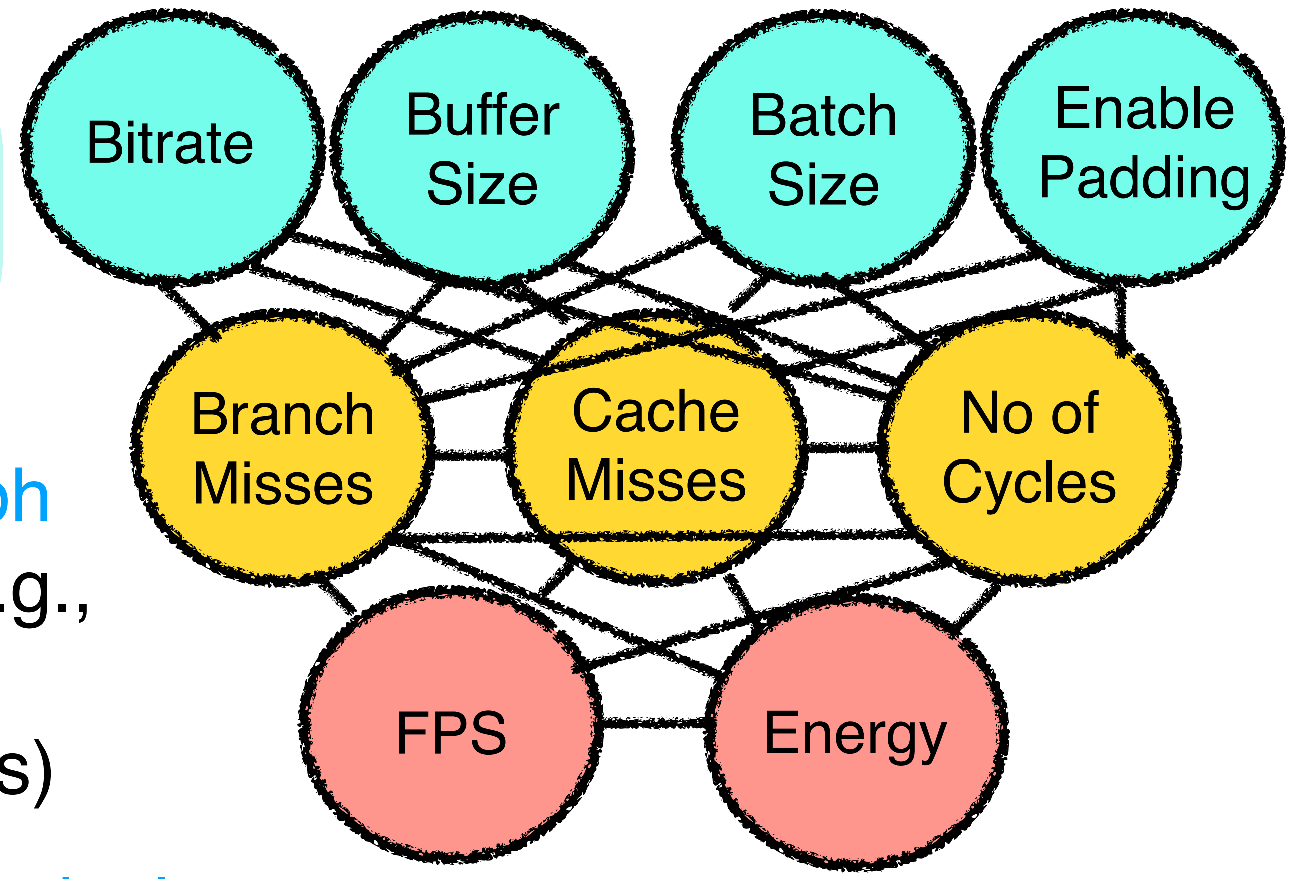
3- Orienting Causal Relations

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

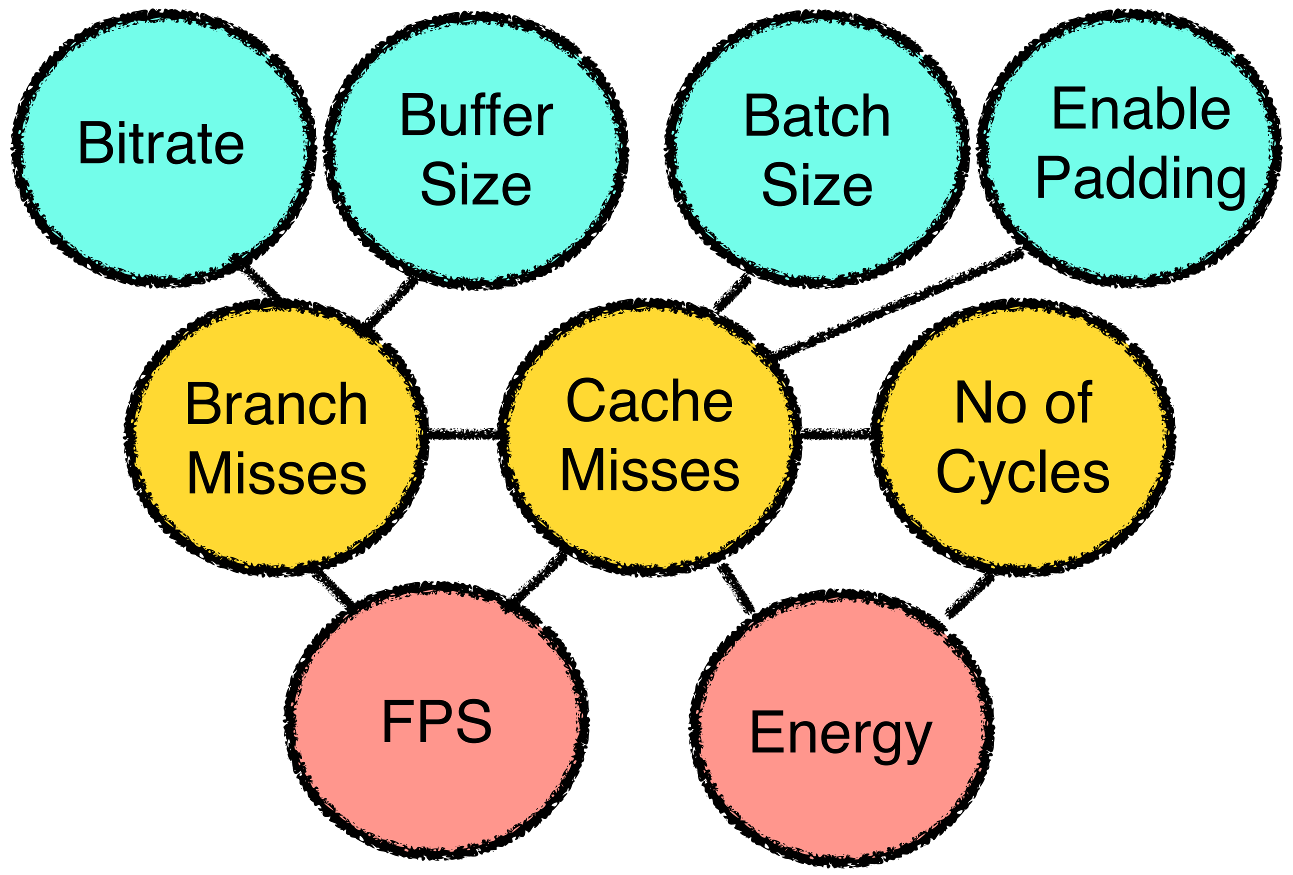
1- Recovering the Skelton

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



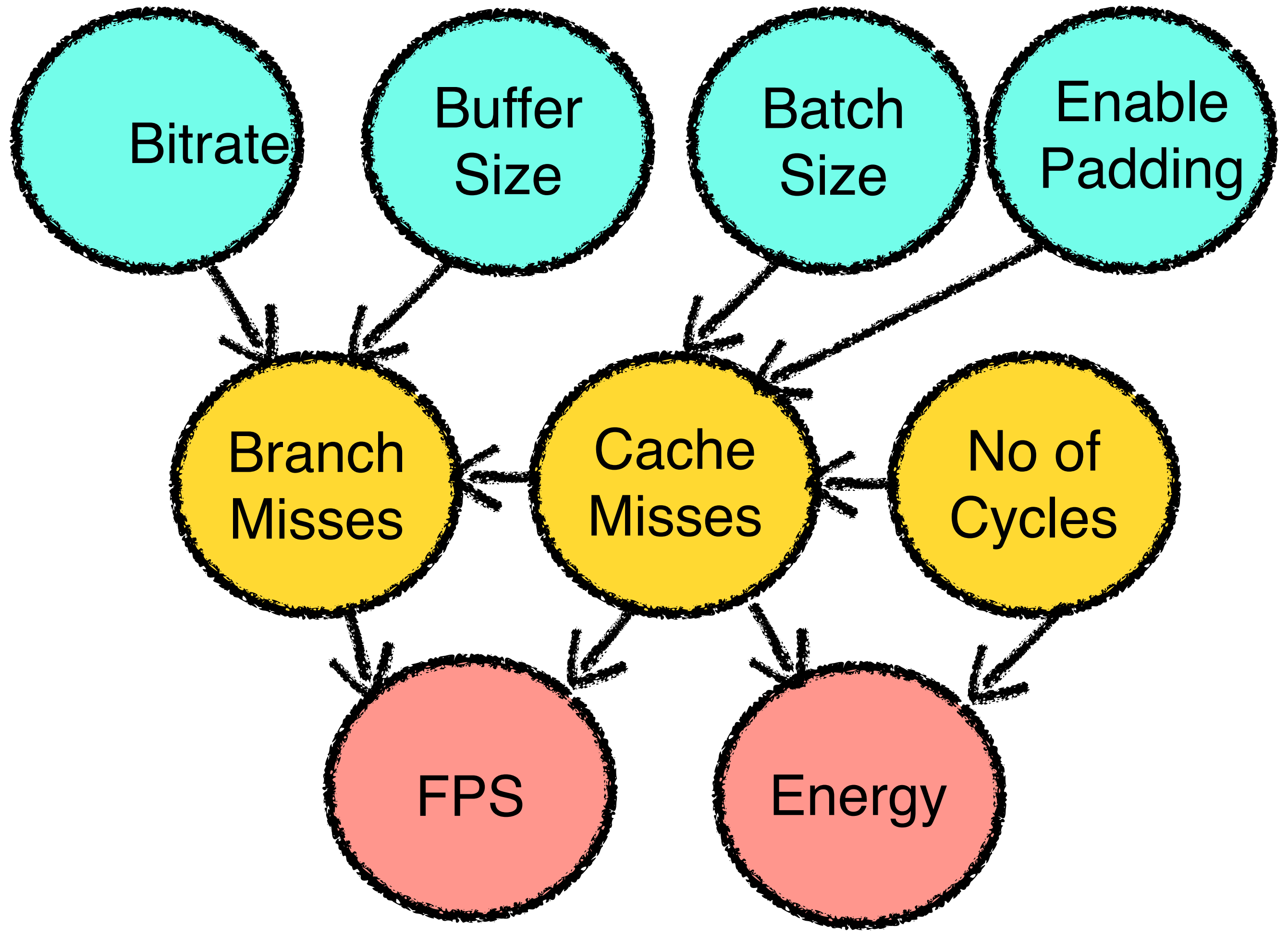
statistical independence tests

2- Pruning Causal Structure

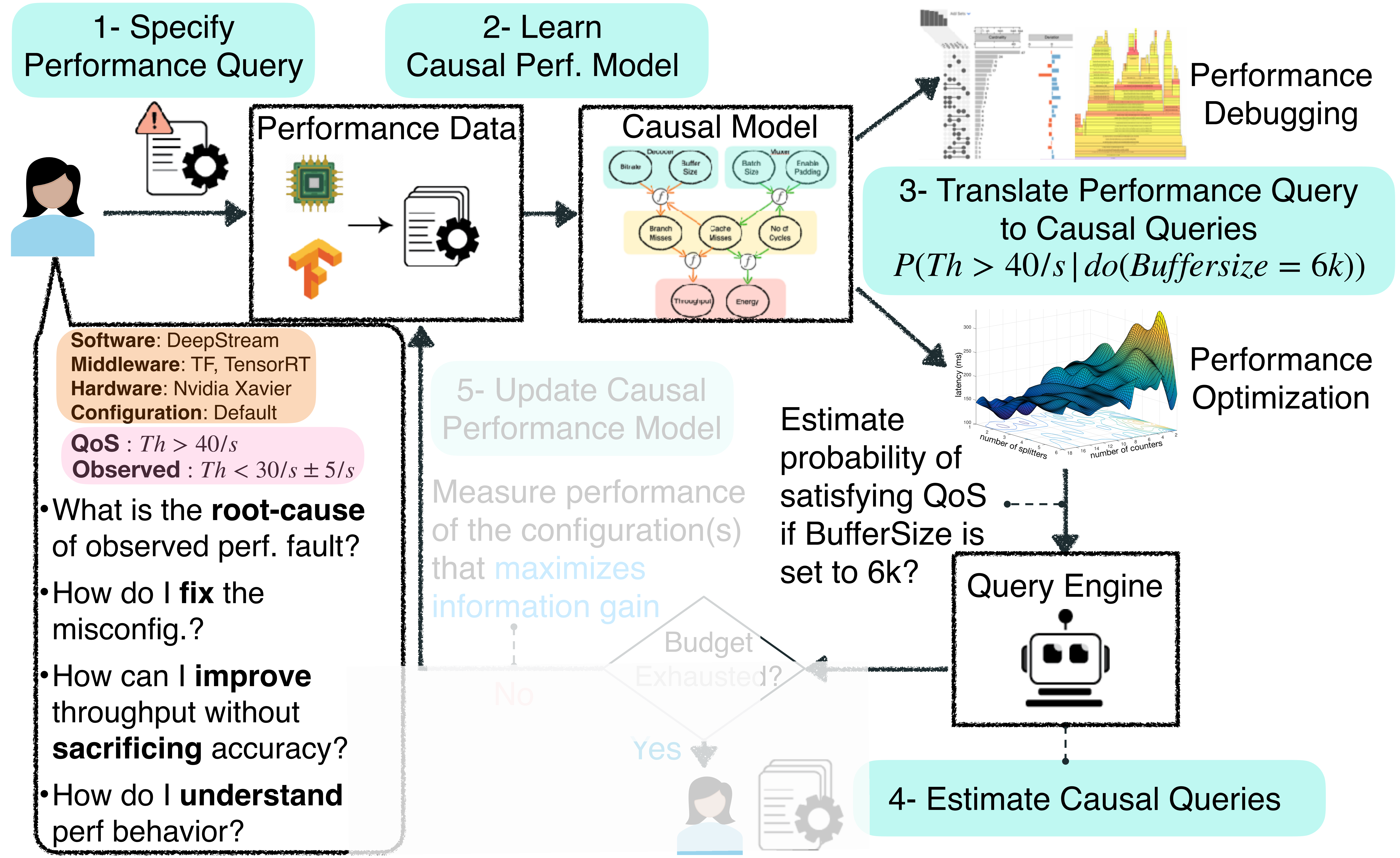


orientation rules & measures (entropy) + structural constraints (colliders, v-structures)

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"?

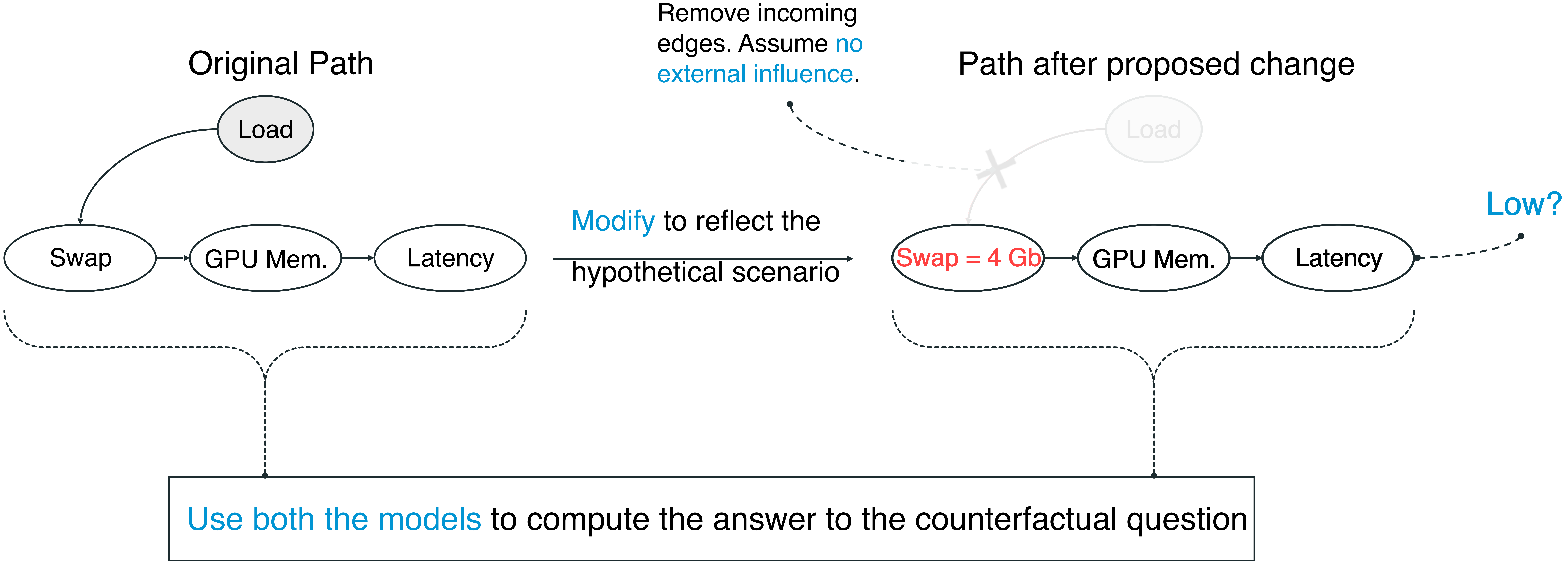
We are interested in the scenario where:

- We hypothetically have low latency;

Conditioned on the following events:

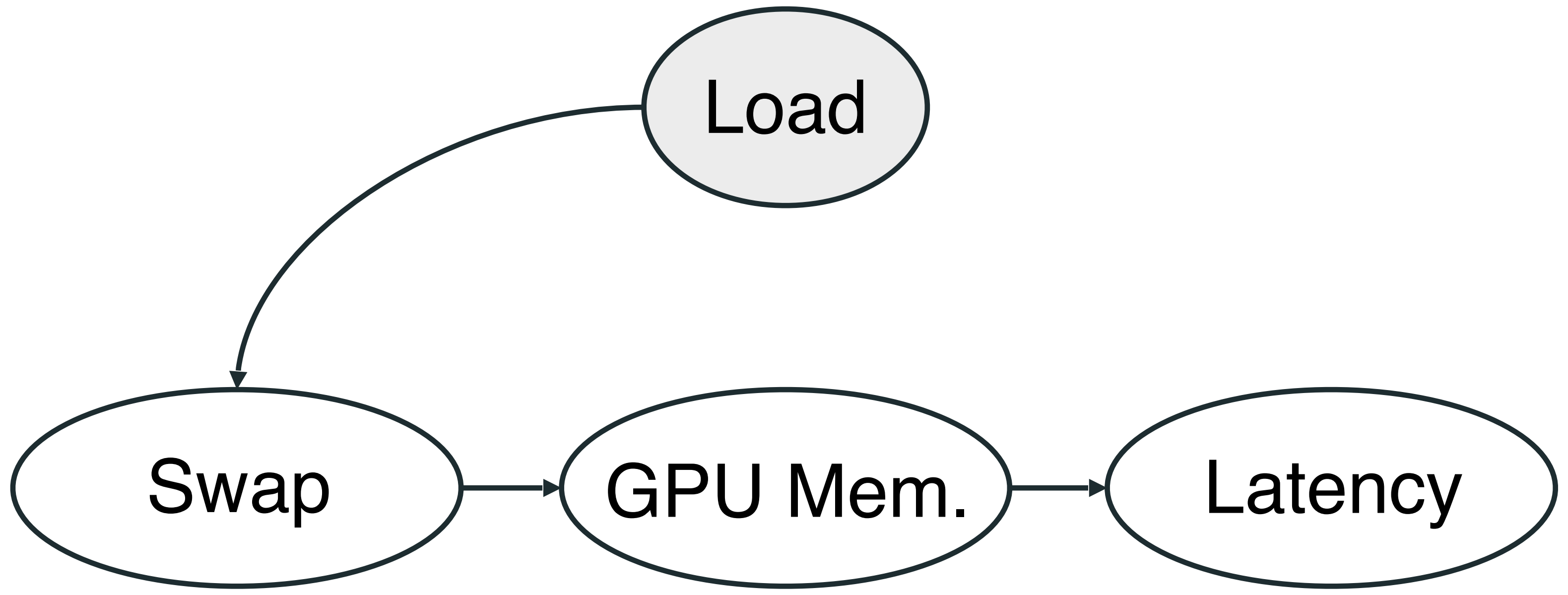
- We hypothetically set the new Swap memory to 4 Gb
- Swap Memory was initially set to 2 Gb
- We observed high latency when Swap was set to 2 Gb
- Everything else remains the same

Diagnosing and Fixing the Faults

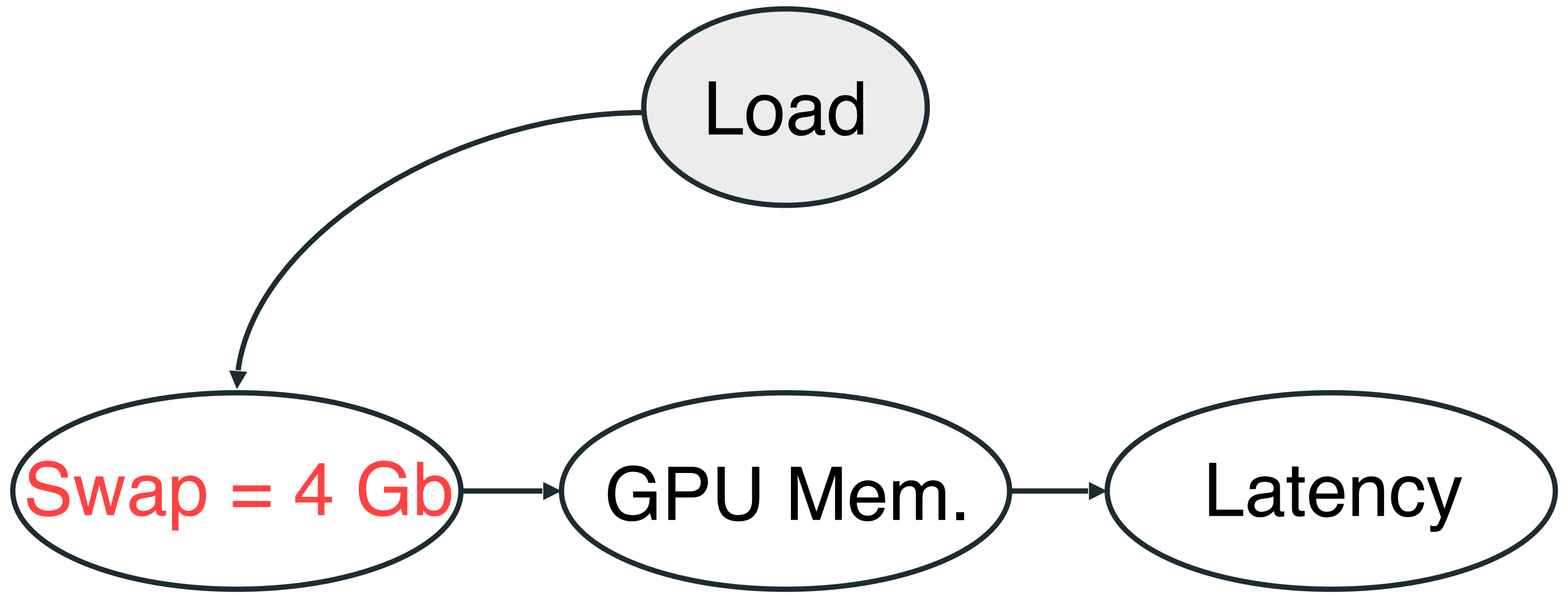


Diagnosing and Fixing the Faults

Original Path



Path after proposed change



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

We expect a low latency

The Swap is now 4 Gb

The Swap was initially 2 Gb

The latency was high

Everything else stays the same

Diagnosing and Fixing the Faults

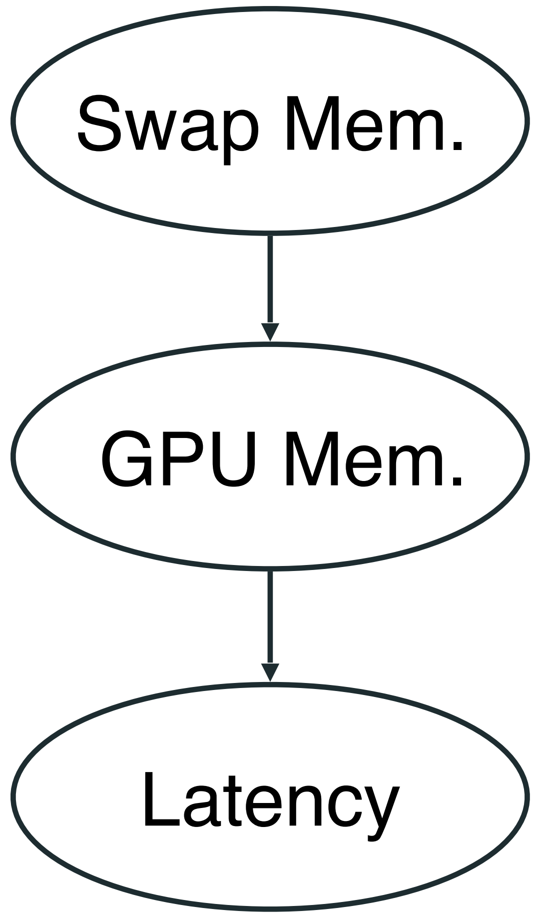
$$\text{Potential} = \underbrace{P\Big(\, \widehat{outcome} = good \;\Big|\; change,\;\; outcome_{\neg change} = bad,\;\; \neg change,\; U\,\Big)}_{\text{Probability that the outcome is good after a change, conditioned on the past}}$$

$$\text{Control} = \underbrace{P\Big(\, \widehat{outcome} = bad \;\Big|\; \neg change,\; U\,\Big)}_{\text{Probability that the outcome was bad before the change}}$$

$$\text{Individual Treatment Effect} = \underbrace{\text{Potential} - \text{Outcome}}_{\text{If this difference is large, then our change is useful}}$$

Diagnosing and Fixing the Faults

Top K paths



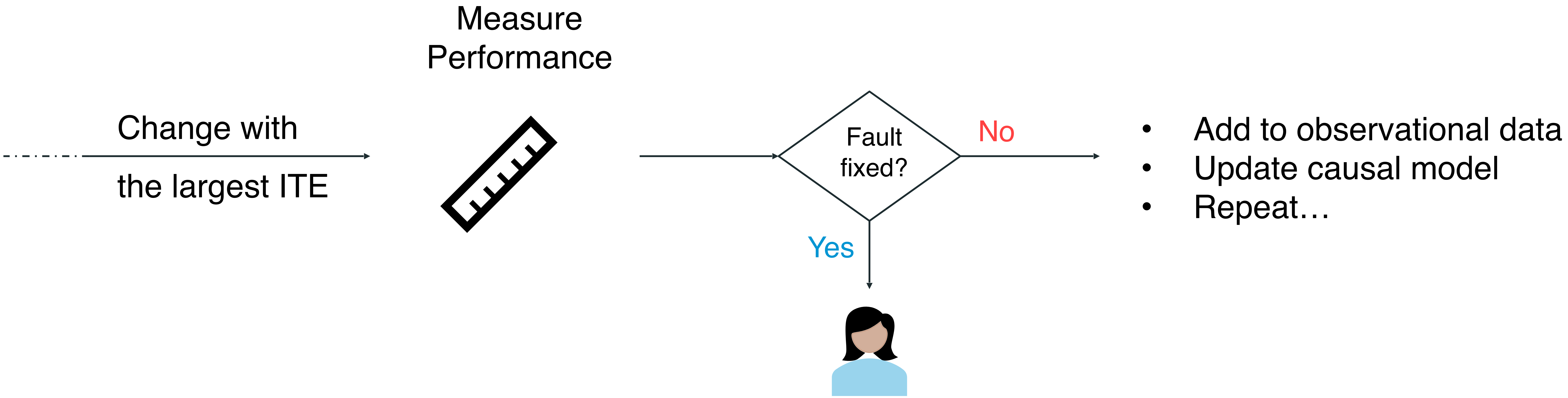
…

Set every configuration option in the path to all permitted values

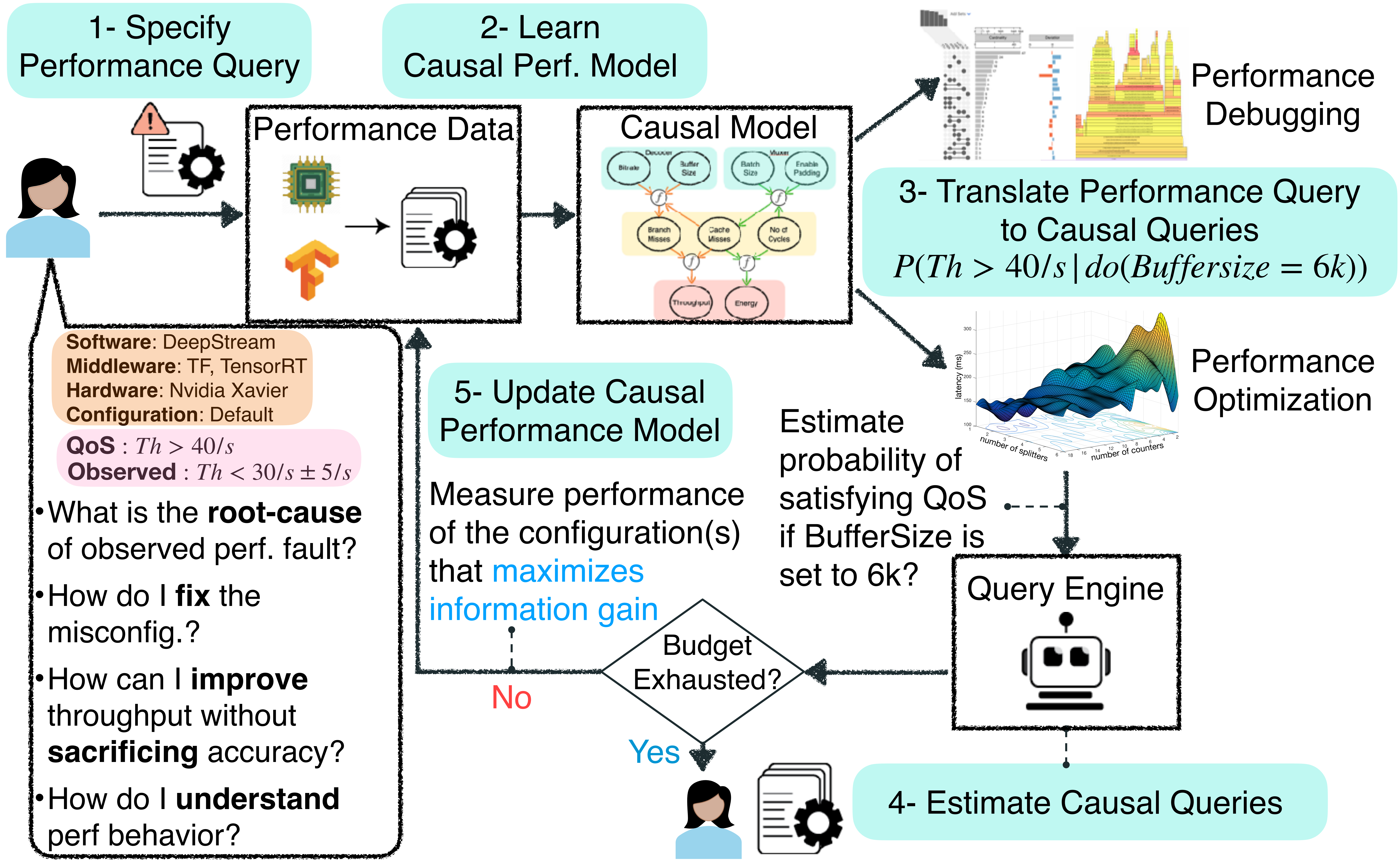
$$\xrightarrow[\text{possible changes}]{\text{Enumerate all}} ITE(change) \xrightarrow[\text{the largest ITE}]{\text{Change with}}$$

! Inferred from observed data. This is very cheap.

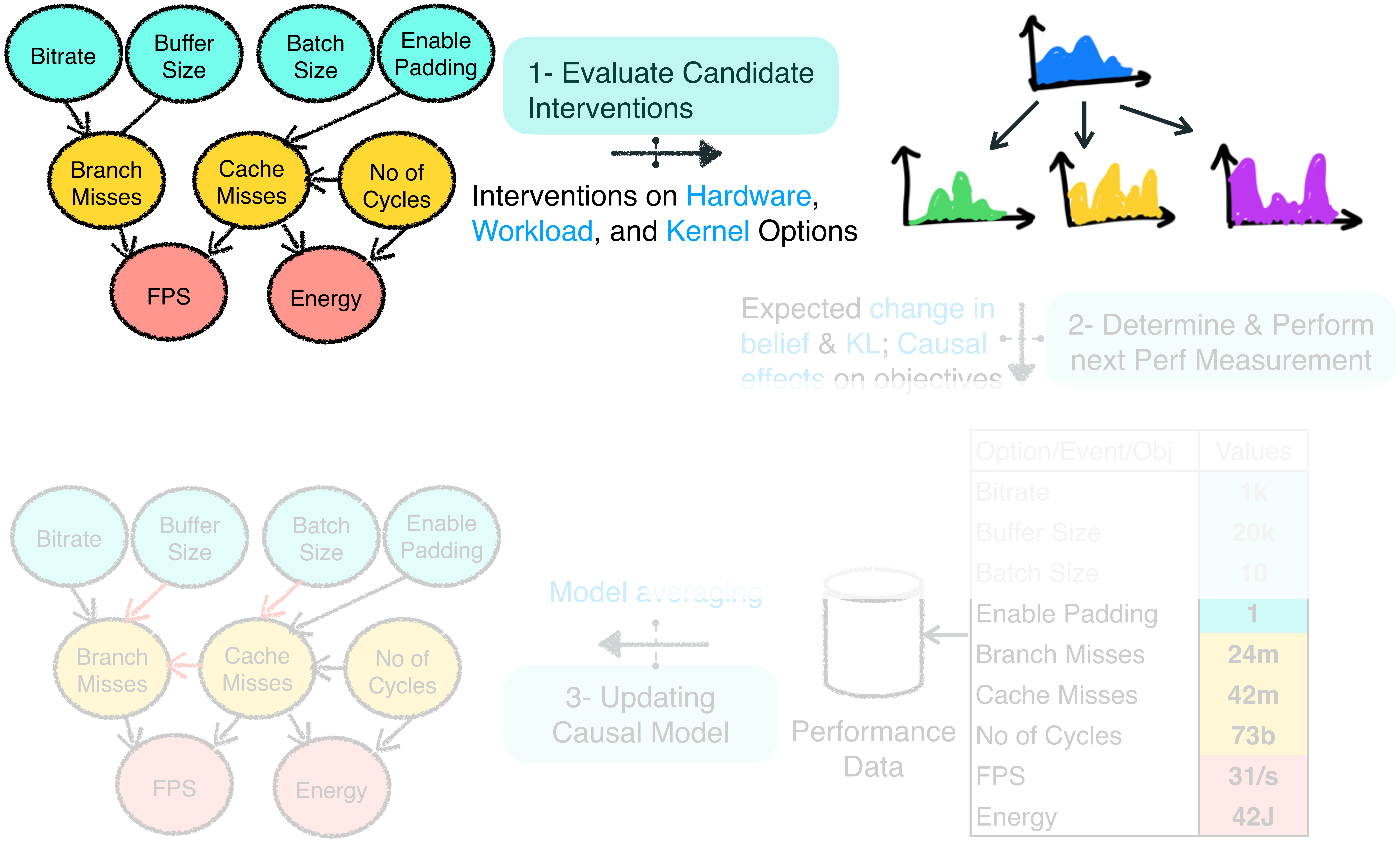
Diagnosing and Fixing the Faults



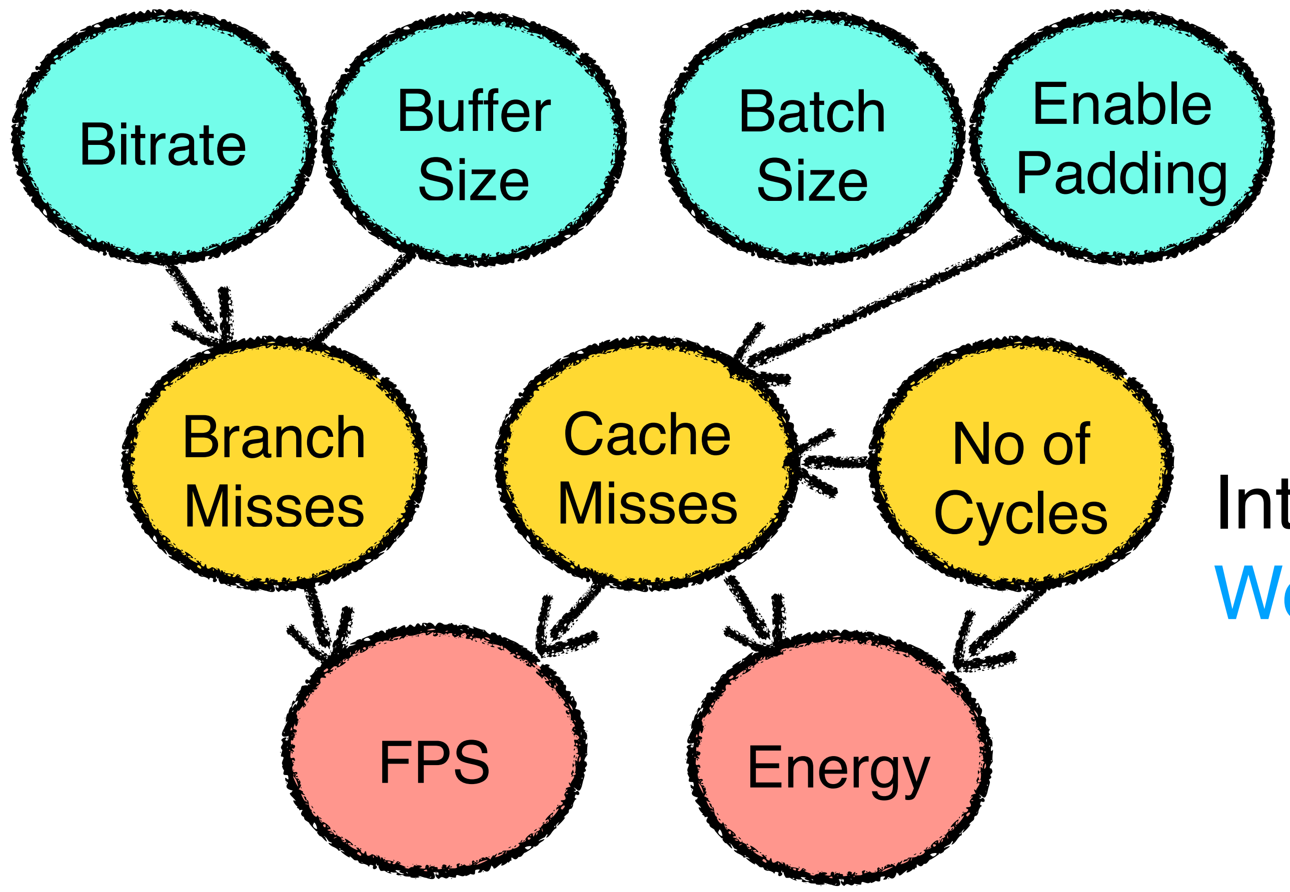
UNICORN: Our Causal AI for Systems Method



Active Learning for Updating Causal Performance Model

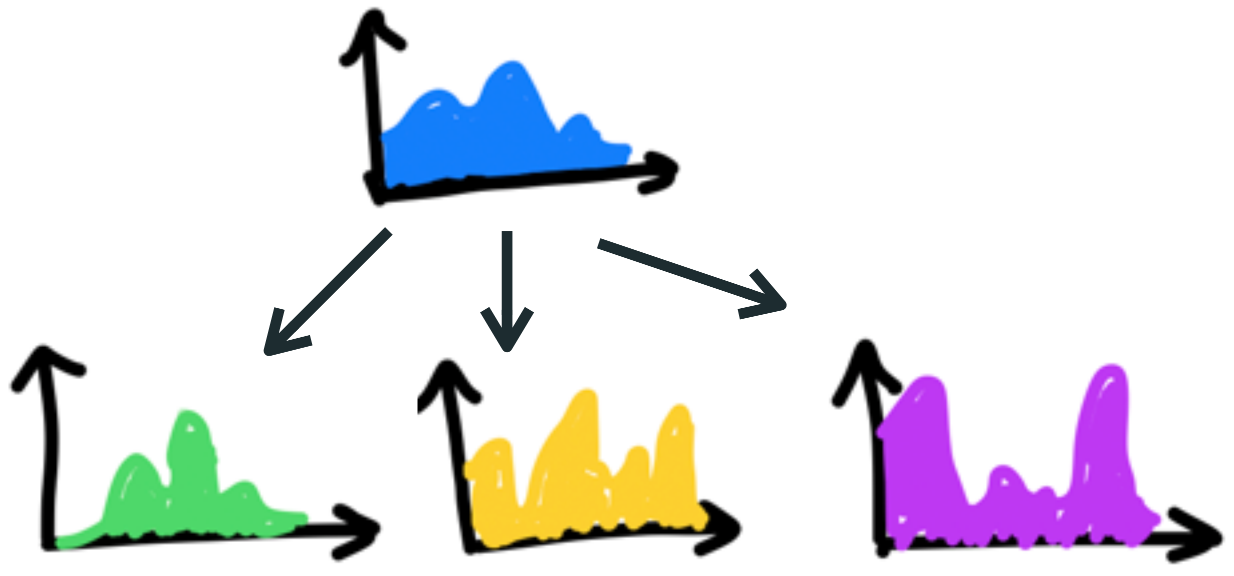


Active Learning for Updating Causal Performance Model



1- Evaluate Candidate Interventions

Interventions on Hardware, Workload, and Kernel Options



Expected change in belief & KL; Causal effects on objectives

2- Determine & Perform next Perf Measurement

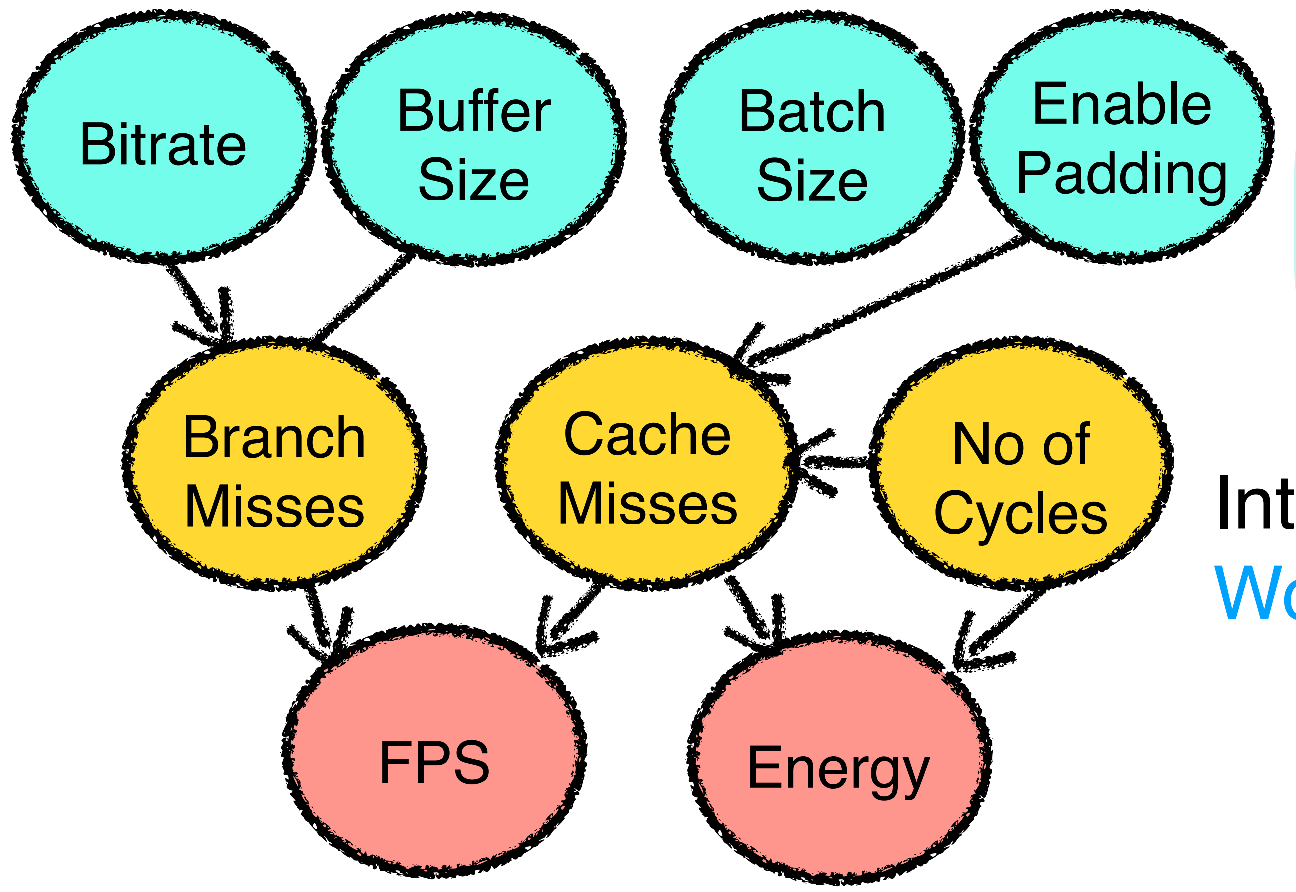
Option/Event/Obj	Values
Bitrate	1k
Buffer Size	20k
Batch Size	10
Enable Padding	1
Branch Misses	24m
Cache Misses	42m
No of Cycles	73b
FPS	31/s
Energy	42J

Performance Data

Model averaging

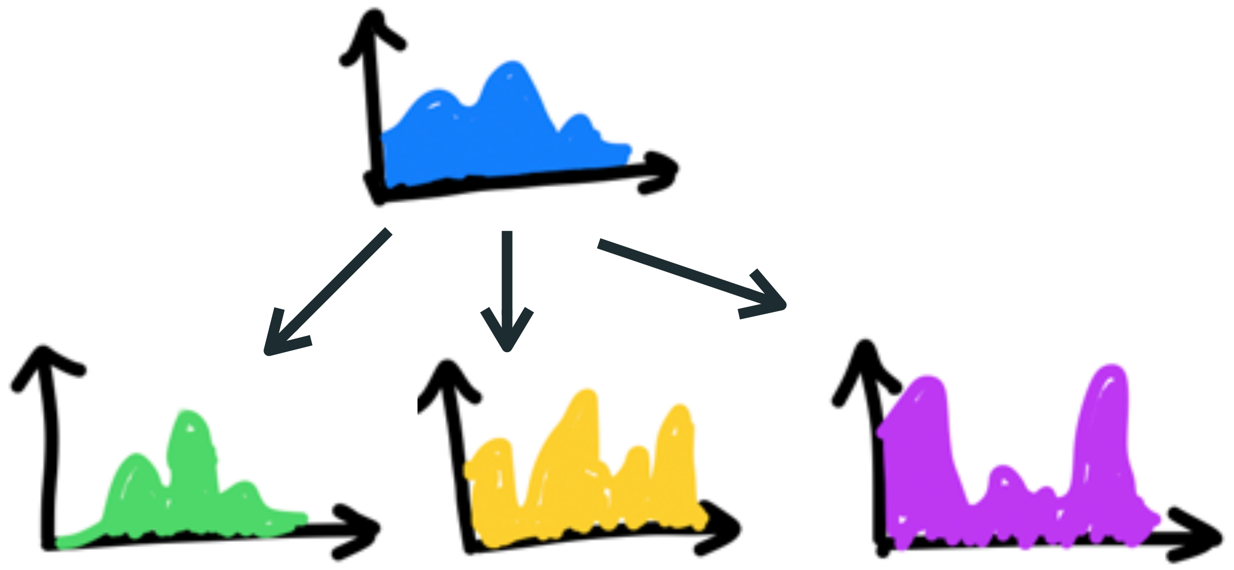
3- Updating Causal Model

Active Learning for Updating Causal Performance Model



1- Evaluate Candidate Interventions

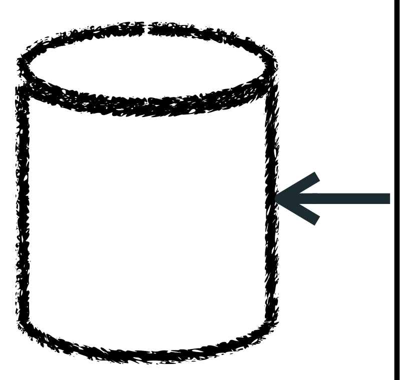
Interventions on Hardware, Workload, and Kernel Options



Expected change in belief & KL; Causal effects on objectives

2- Determine & Perform next Perf Measurement

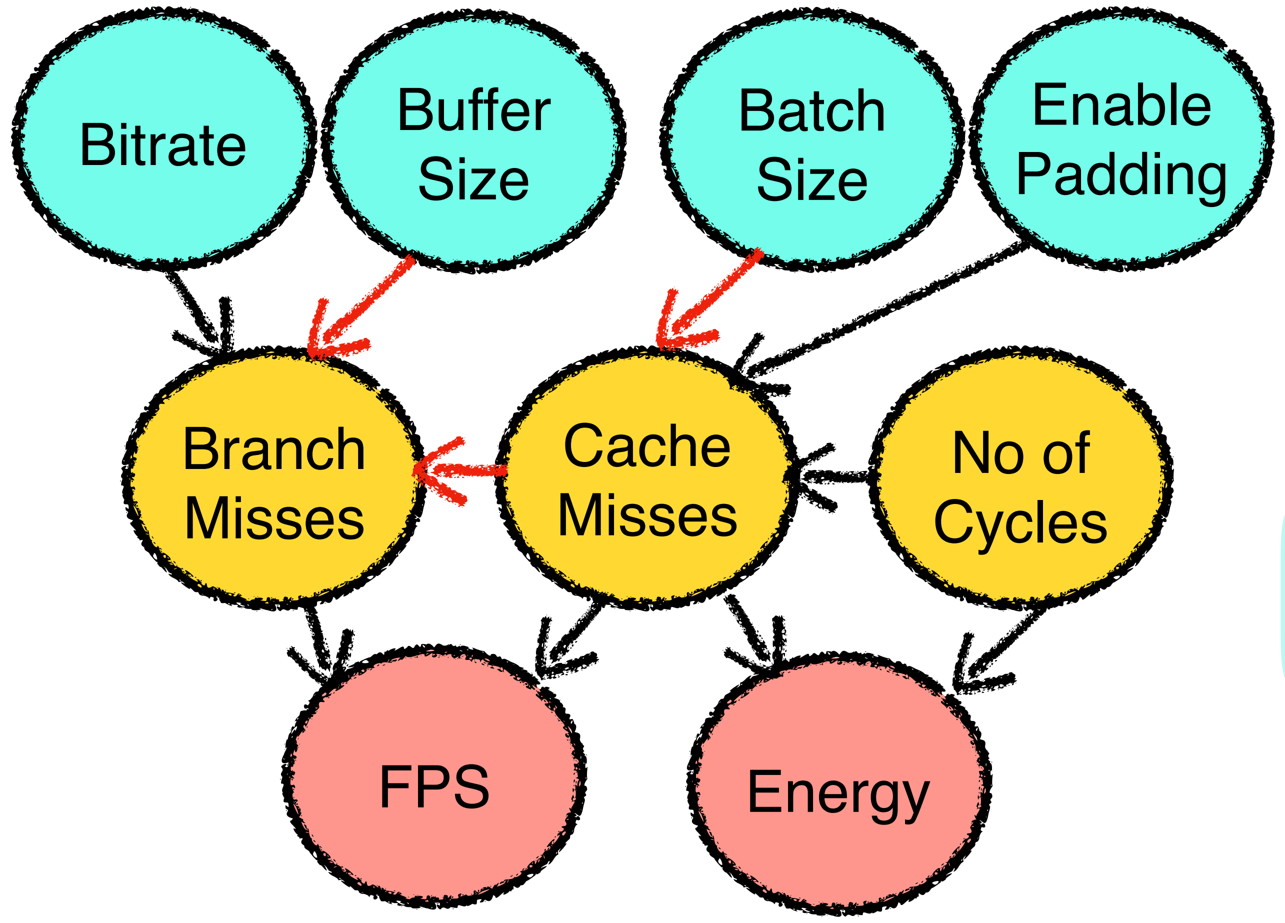
Option/Event/Obj	Values
Bitrate	**1k**
Buffer Size	**20k**
Batch Size	**10**
Enable Padding	**1**
Branch Misses	**24m**
Cache Misses	**42m**
No of Cycles	**73b**
FPS	**31/s**
Energy	**42J**



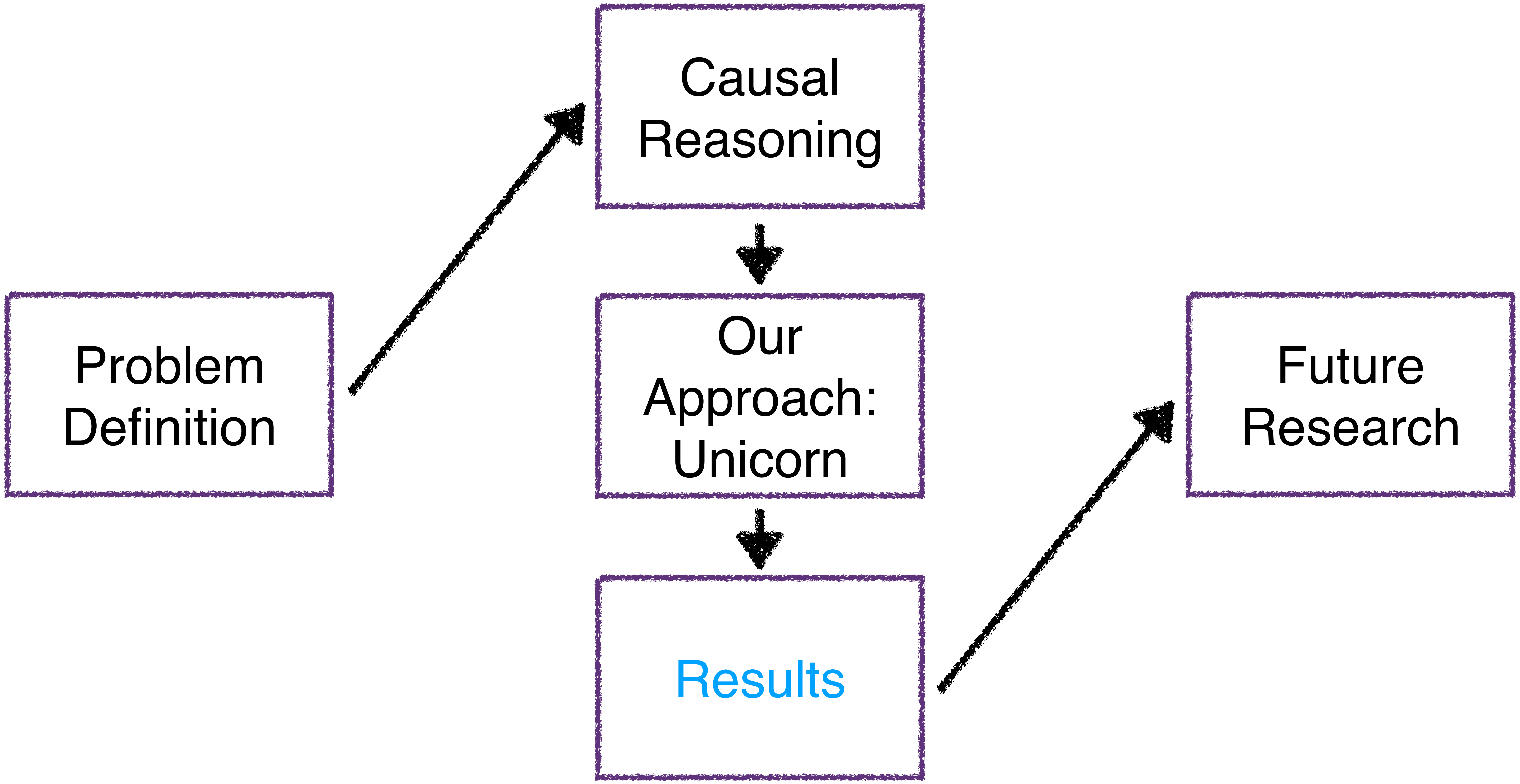
Performance Data

Model averaging

3- Updating Causal Model



Outline



Causal Reasoning
Problem Definition
Our Approach: Unicorn
Future Research
Results

Results: Case Study

NVIDIA. DEVELOPER

CUDA performance issue on tx2

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

william_wu Jun '17

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

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

Embedded real-time stereo estimation

Source code

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

More powerful

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

17 Fps

4× Slower!

4 Fps

Results: Case Study

Configuration	UNICORN	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.	$3^1/_2$ 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

Configuration Space

X 30 Configurations

- 10 software
- 10 OS/Kernel
- 10 hardware

X 17 System Events

Systems

Xception

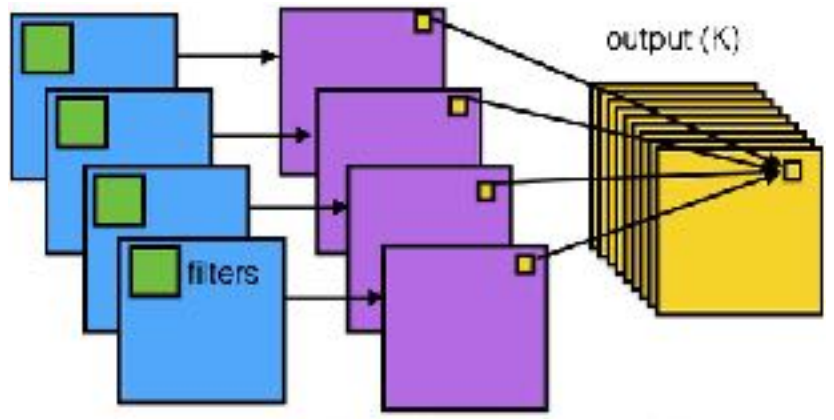
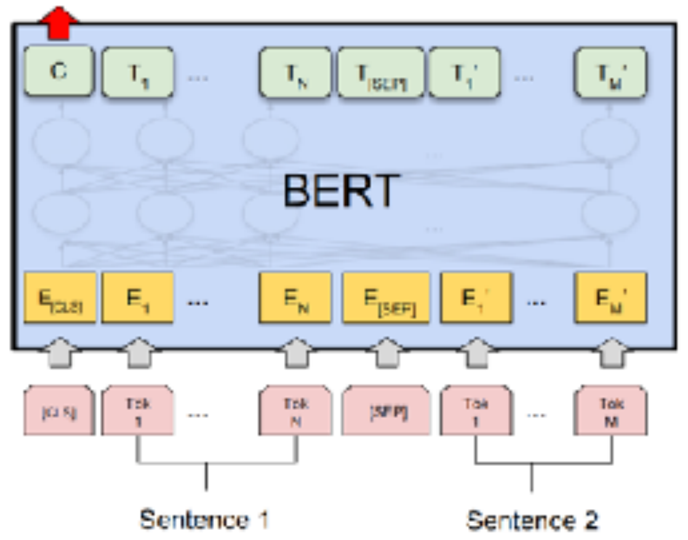


Image recognition
(50,000 test images)

DeepSpeech

Voice recognition
(5 sec. audio clip)

BERT



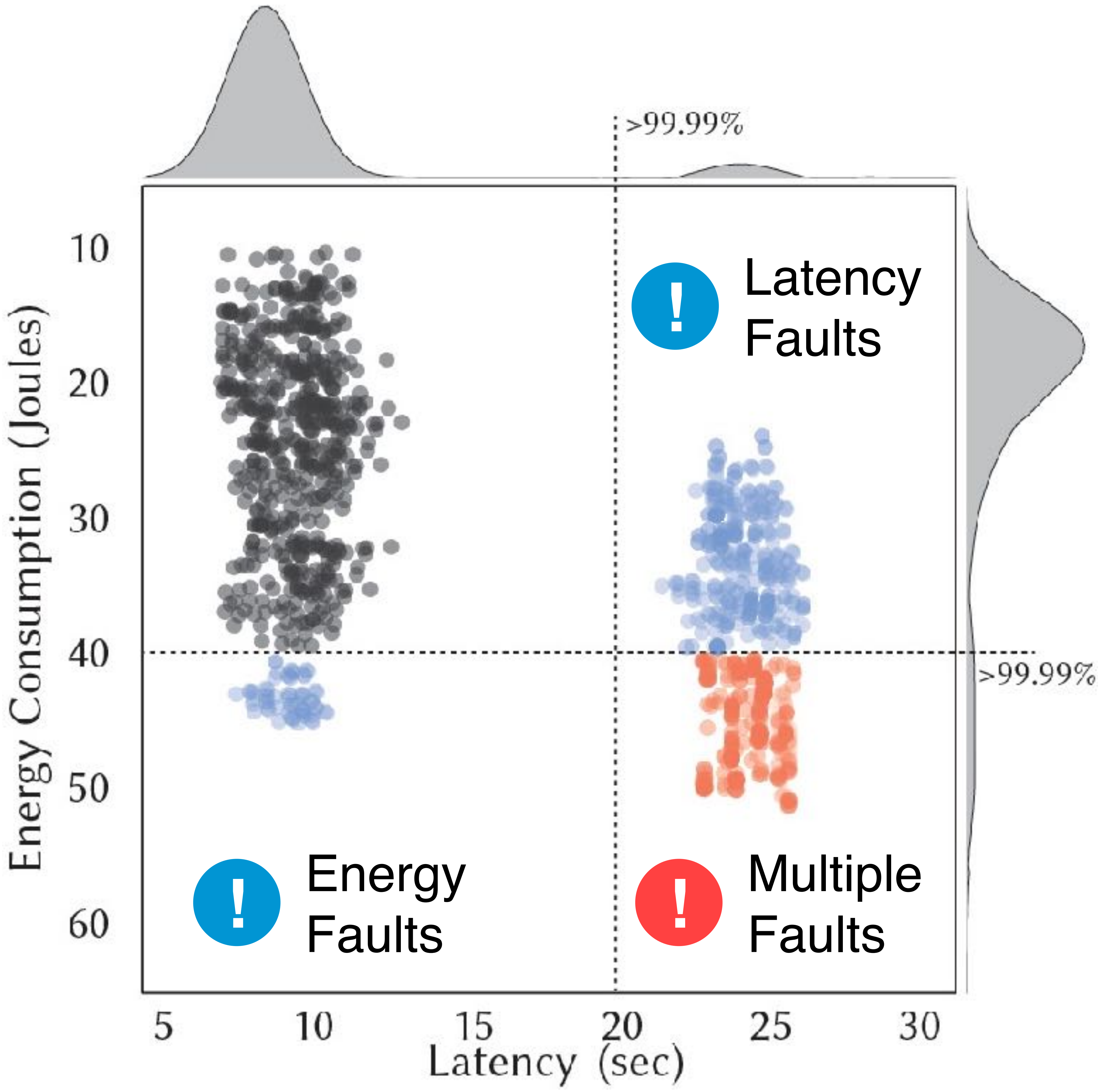
Sentiment Analysis
(10000 IMDb reviews)

x264

Video Encoder
(11 Mb, 1080p video)

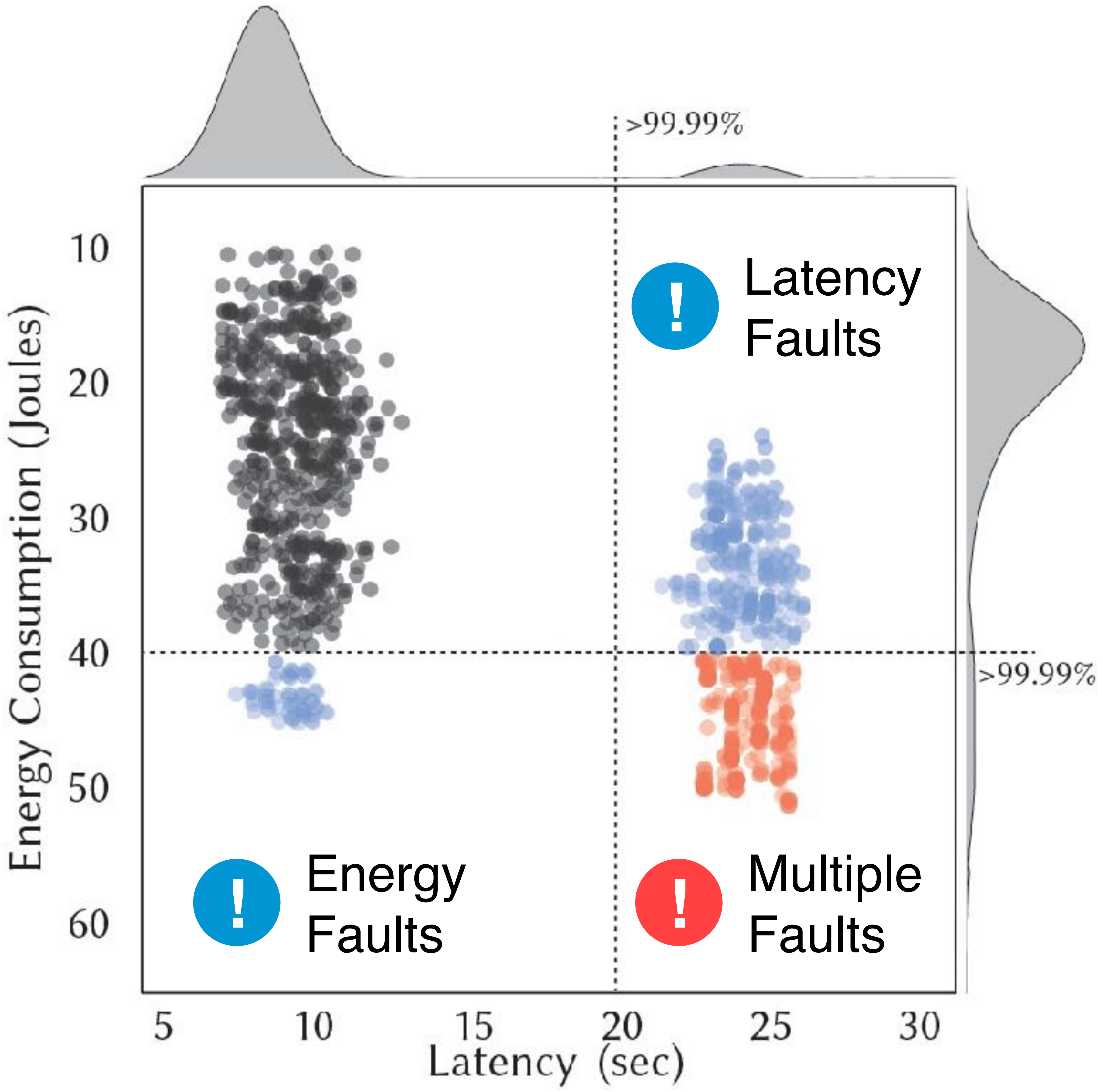
Evaluation: Data Collection

- 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

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

Statistical Debugging for Real-World Performance Problems

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

Iterative Delta Debugging

Cyrille Artho

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

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

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	CBI	DD	EnCore	BugDoc	Unicorn	Others
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)

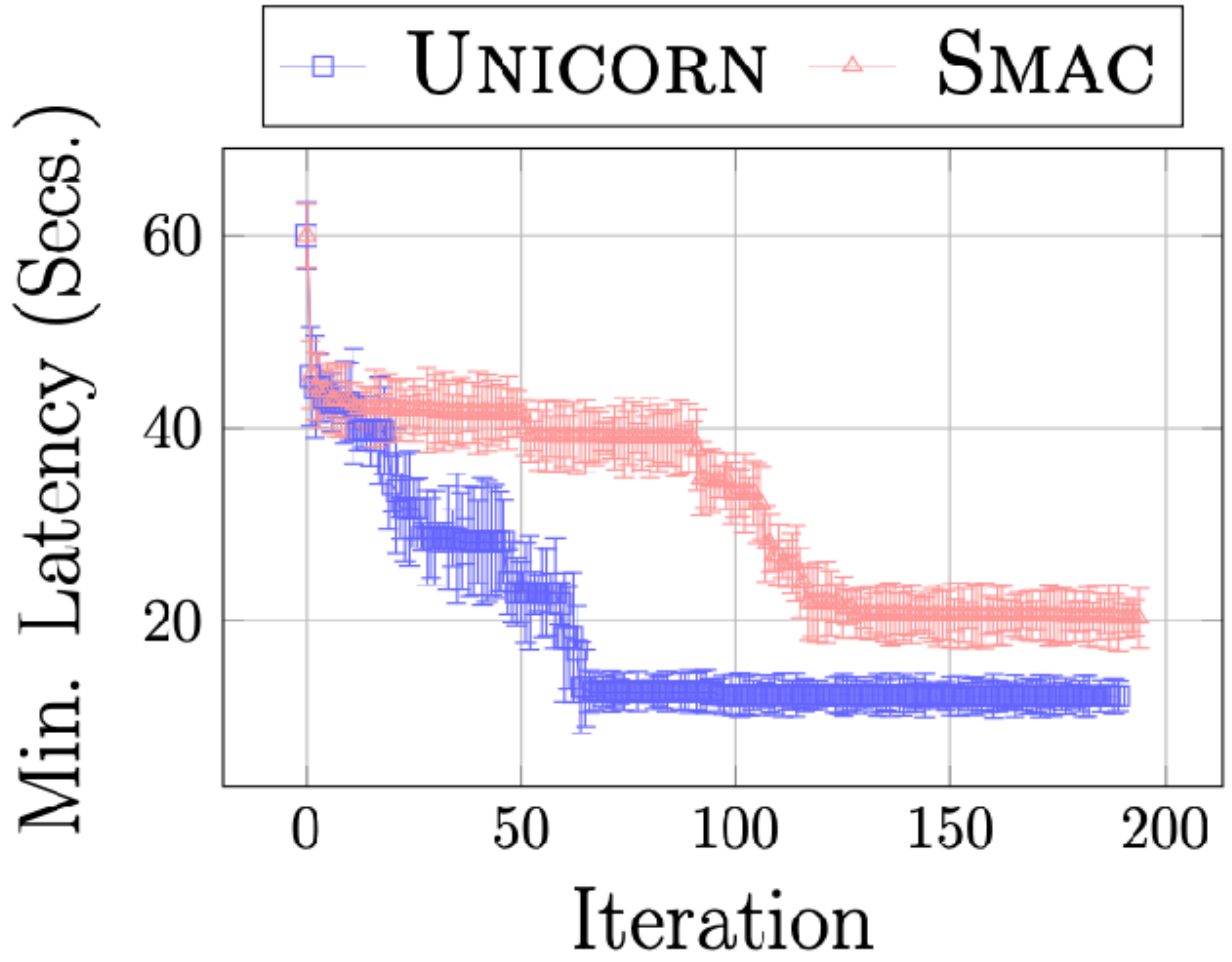
		Accuracy				Precision				Recall				Gain (Latency)				Gain (Energy)				Time†	
		Unicorn	CBI	EnCore	BugDoc	Unicorn	CBI	EnCore	BugDoc	Unicorn	CBI	EnCore	BugDoc	Unicorn	CBI	EnCore	BugDoc	Unicorn	CBI	EnCore	BugDoc	Unicorn	Others
Energy + Latency	Xception	**89**	76	81	79	**77**	53	54	62	**81**	59	59	62	**84**	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

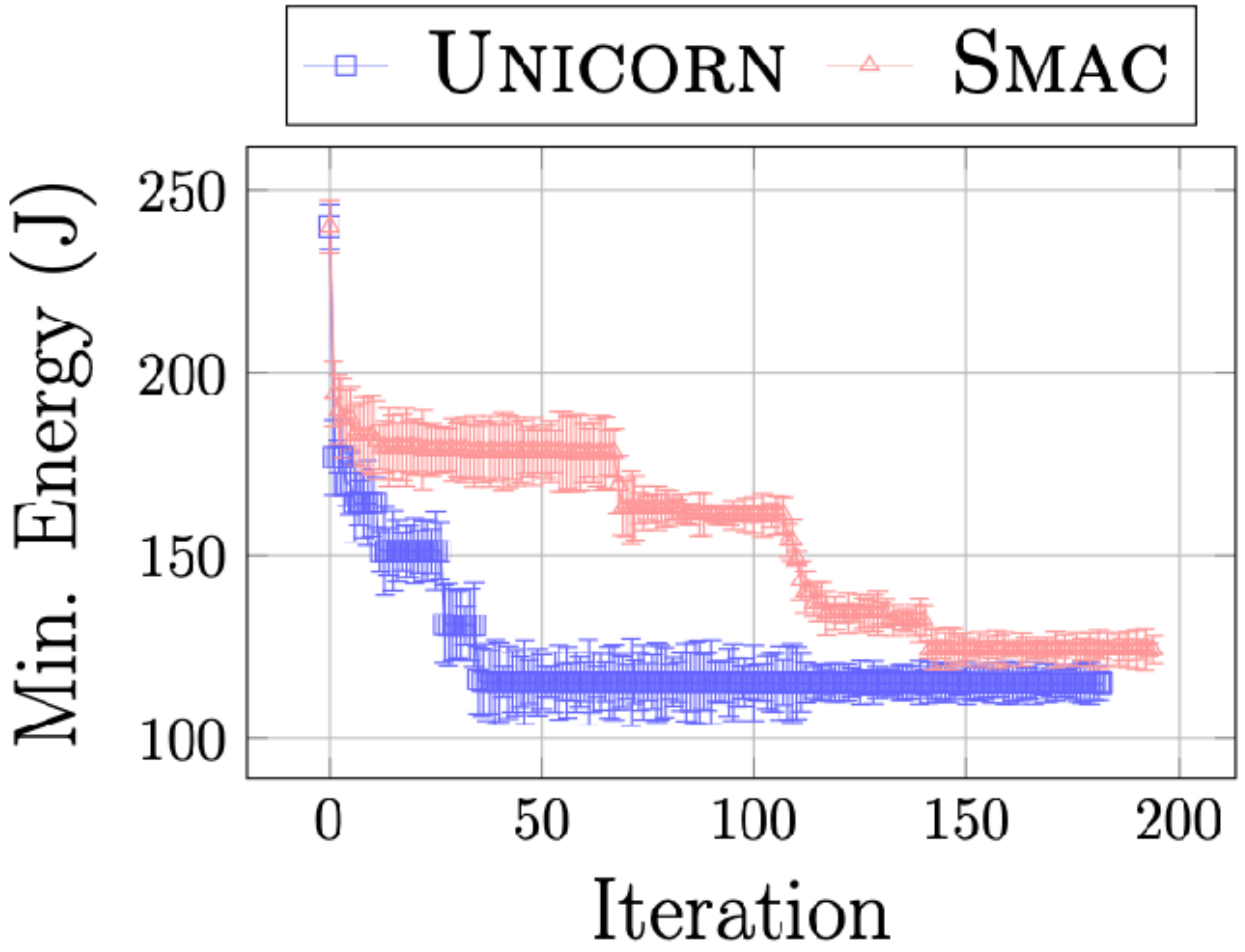
Multiple Faults in Latency & Energy usage

Better gain across both objectives

Results: Efficiency (Optimization; Single objective)

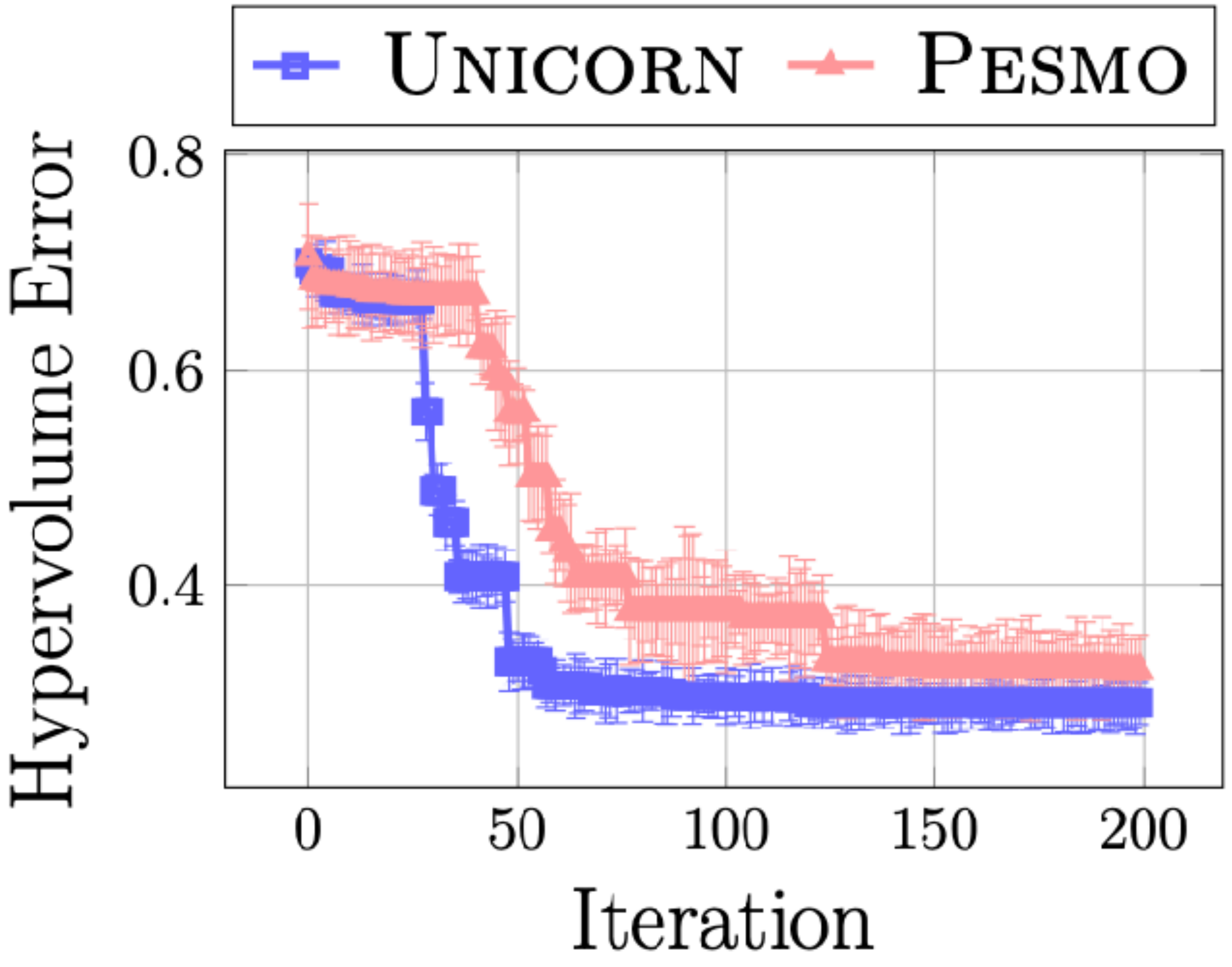


(a) Single Objective Latency

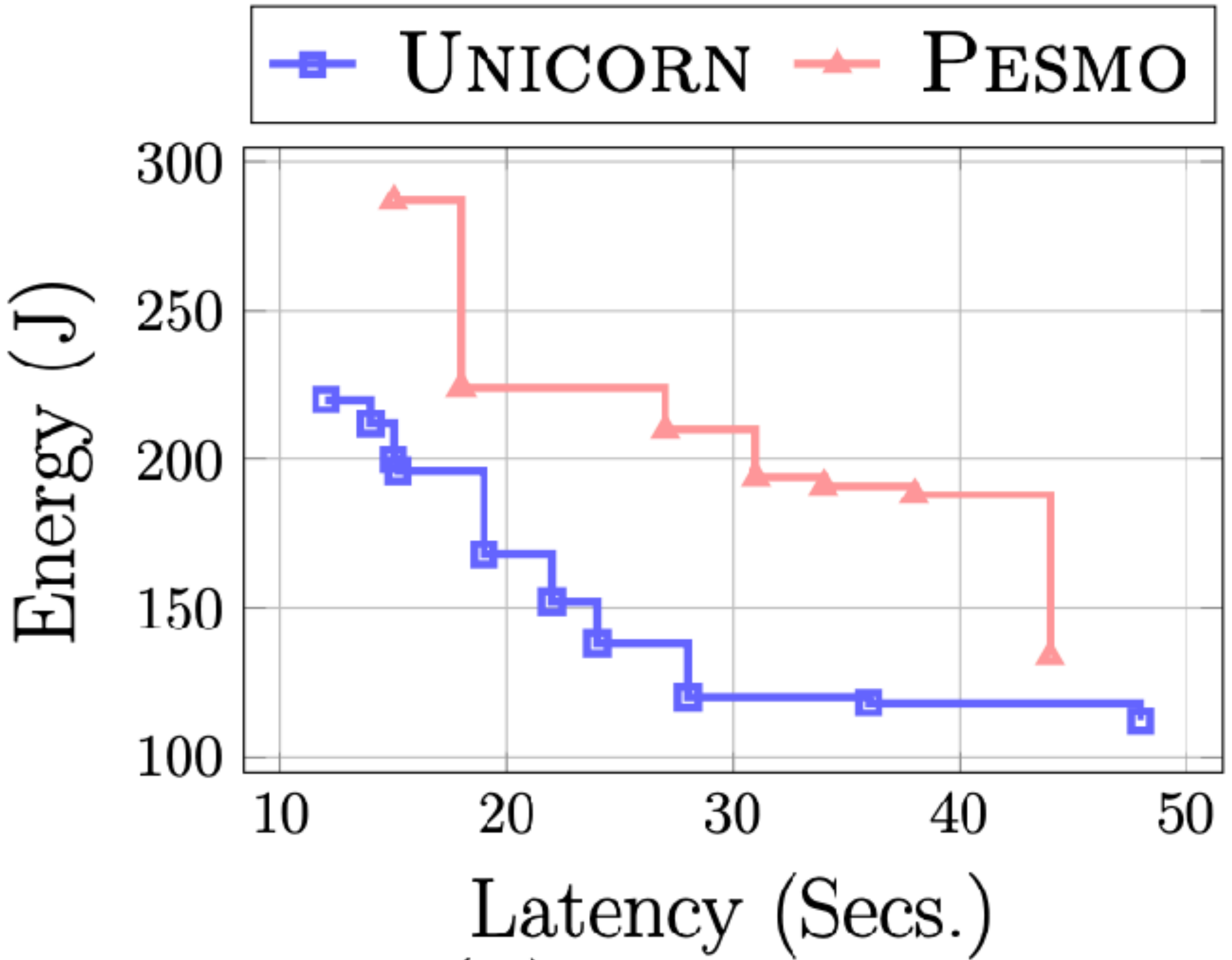


(b) Single Objective Energy

Results: Efficiency (Optimization; Multi-objective)

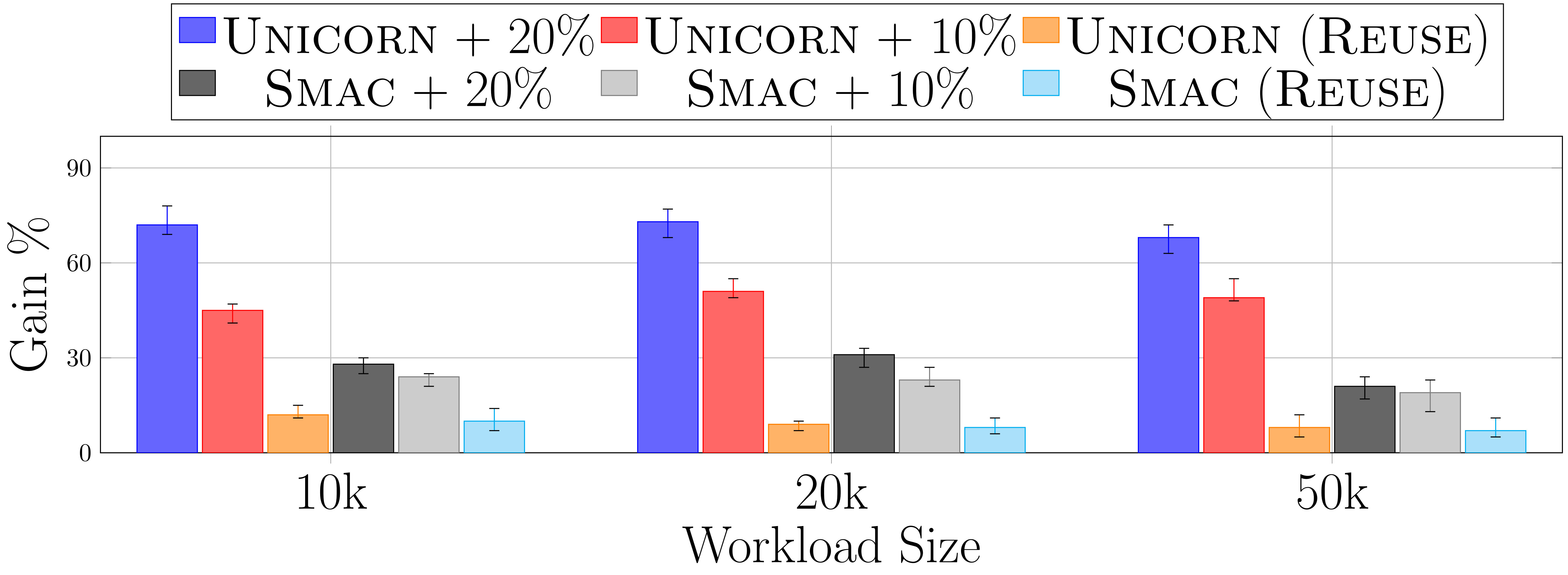


(**c**) Multi-Objective Latency and Energy



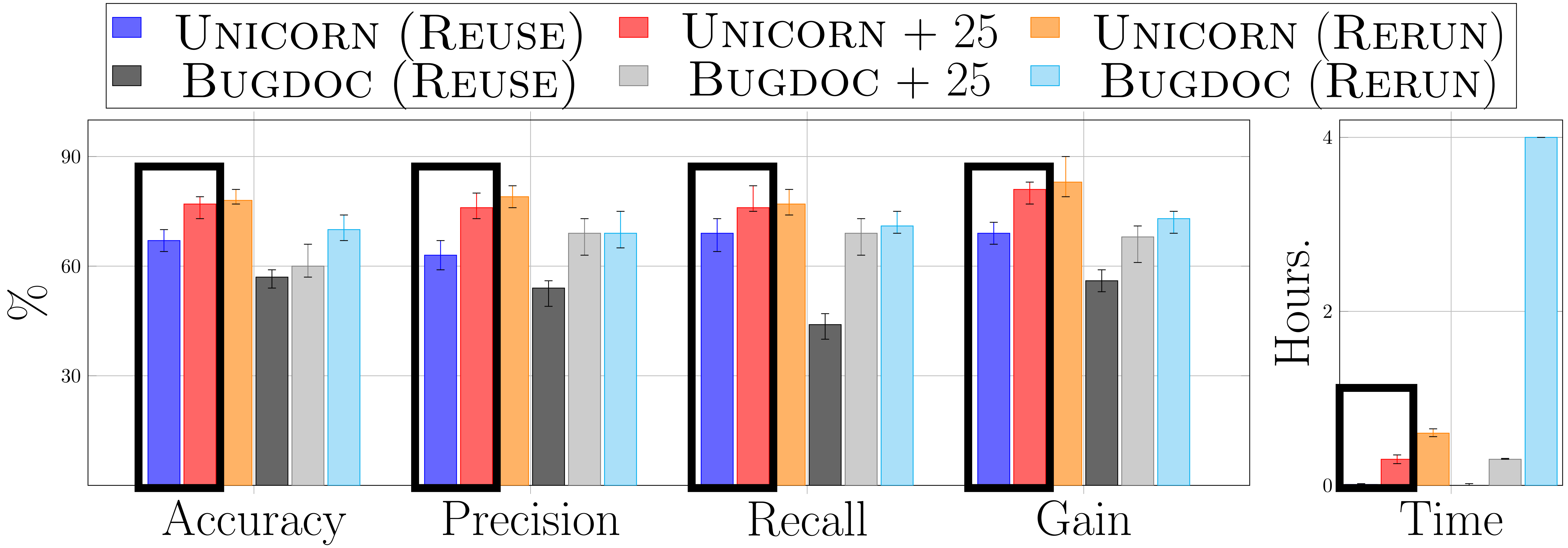
(**d**) Pareto Front

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**

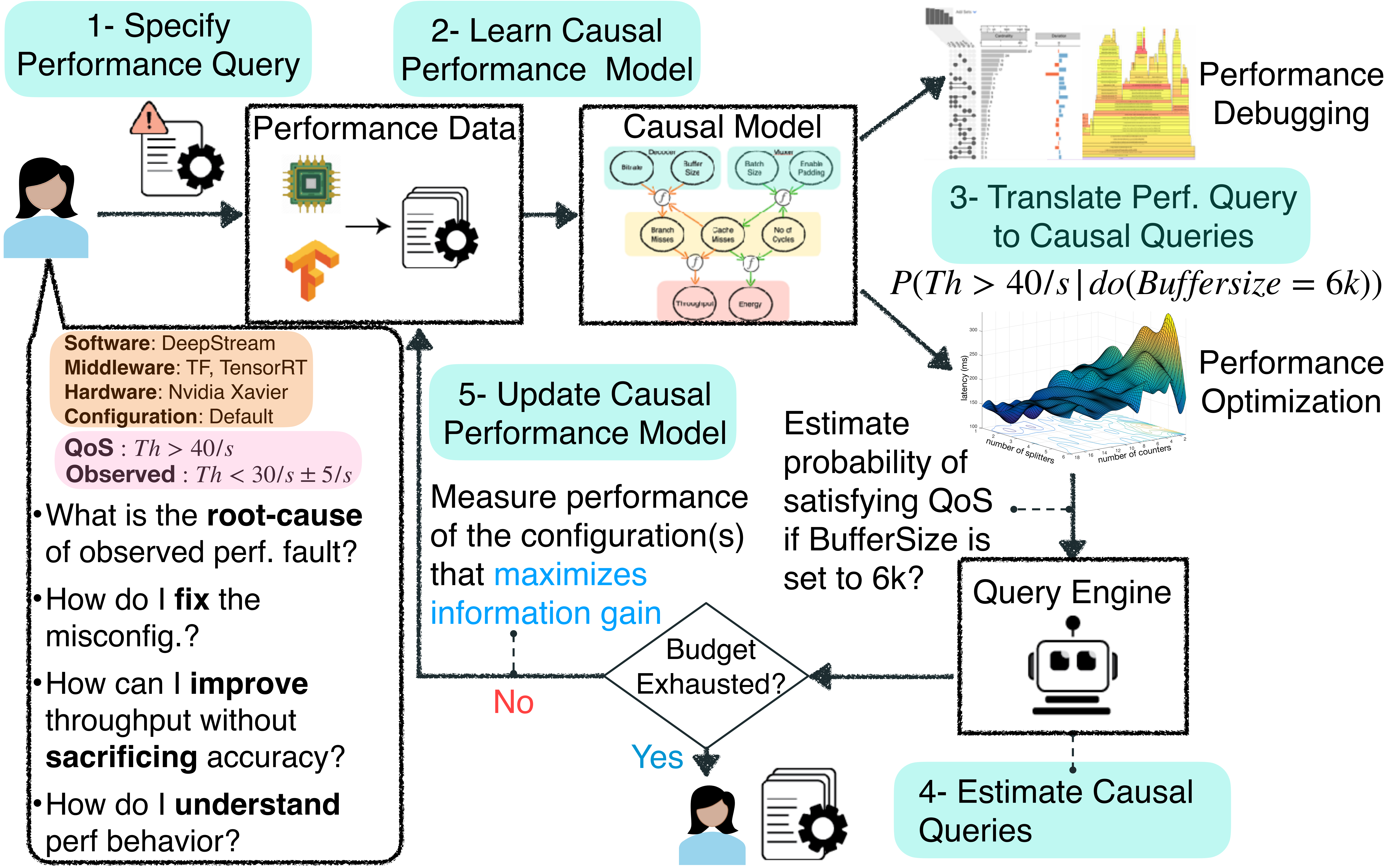
Causal graphs are sparse

Summary: Causal AI for Systems

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

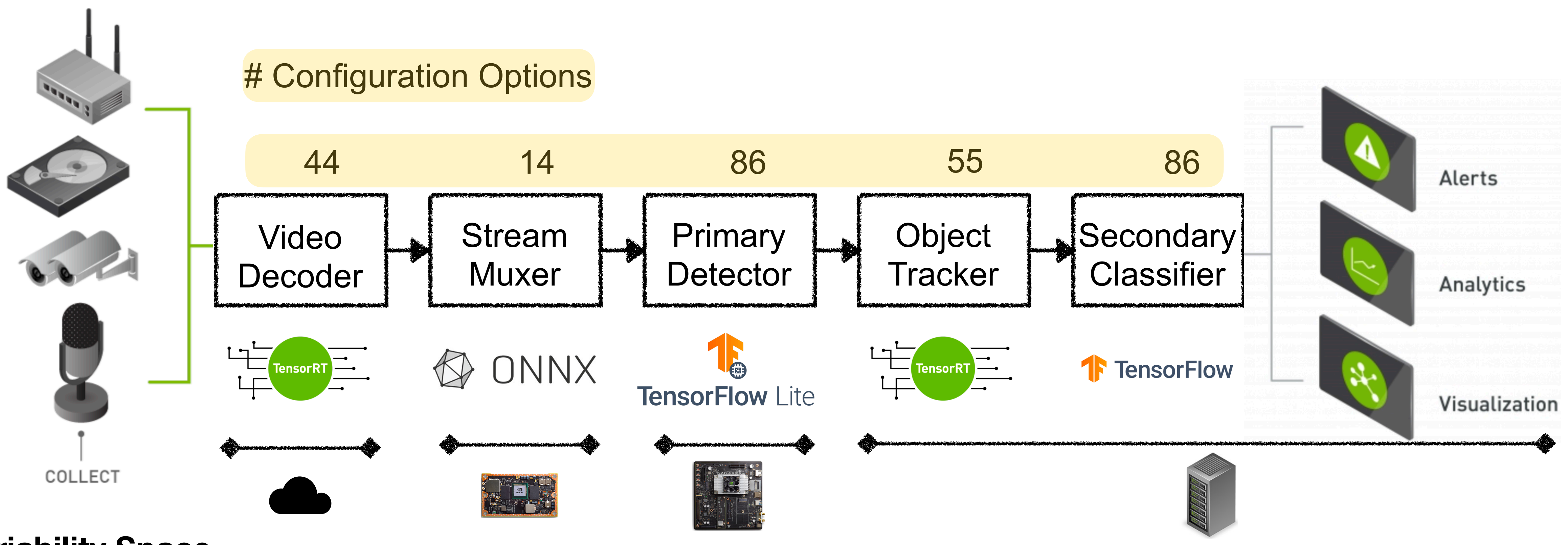
2. The learned causal model is **transferable** across different environments.

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



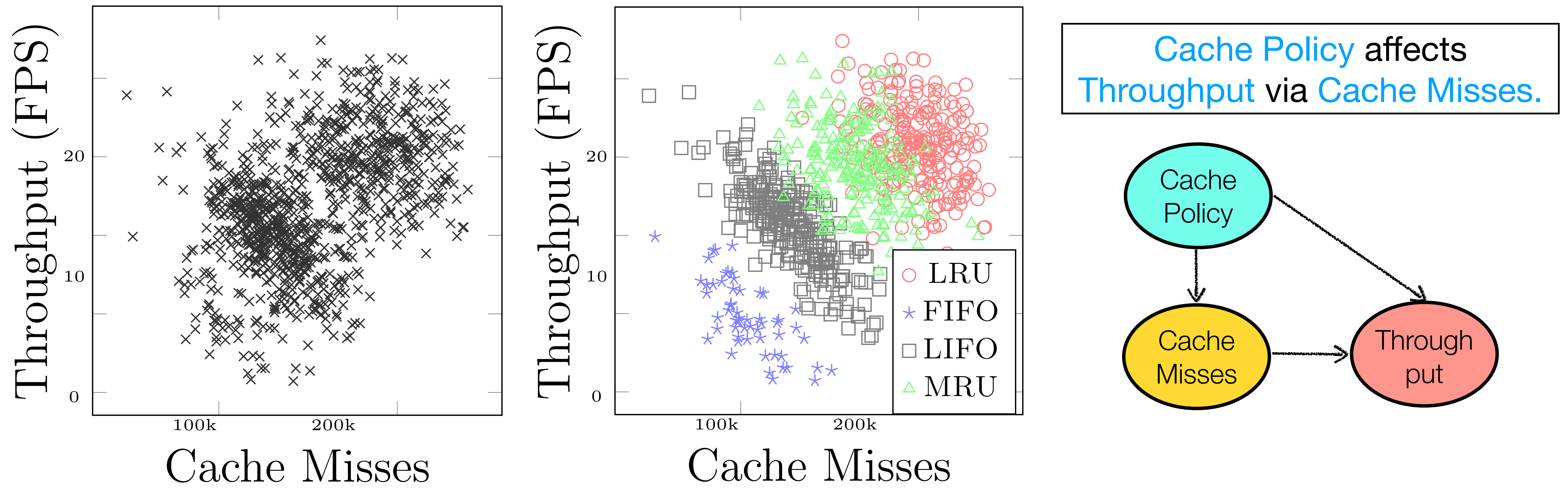
The variability space of today's systems is exponentially increasing

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



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

Causal performance models produce correct explanations



Causal performance models capture correct interactions.

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

Configuration Space

X 30 Configurations
- 10 software
- 10 OS/Kernel
- 10 hardware

X 17 System Events

Systems

Xception

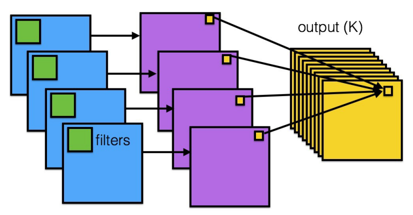
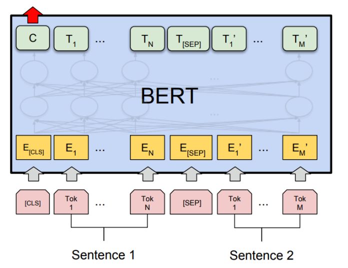


Image recognition (50,000 test images)

DeepSpeech

Voice recognition (5 sec. audio clip)

BERT



Sentiment Analysis (10000 IMDb reviews)

x264

Video Encoder (11 Mb, 1080p video)

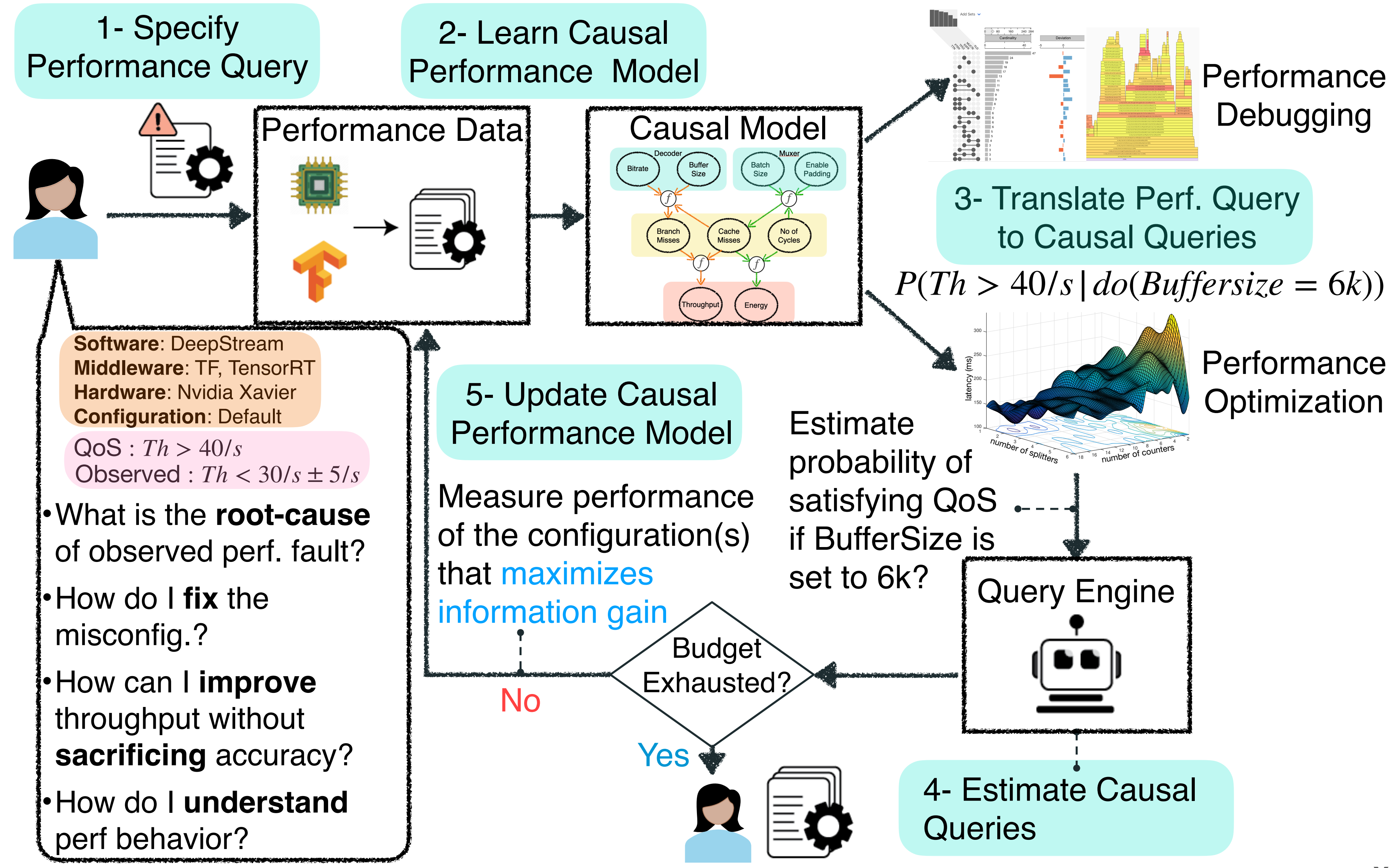
x

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.



x

How to resolve these issues faster?

Performance measurement

Dead code removal

Constant folding

Loop unrolling

Function inlining

Compiler
(e.f., SaC, LLVM)

Configuration
Space

$$\mathbb{C} = \boxed{O_1} \times \boxed{O_2} \times \cdots \times \boxed{O_{19}} \times \boxed{O_{20}}$$

$c_1 \in \mathbb{C}$

$$c_1 = 0 \times 0 \times \cdots \times 0 \times 1$$

Configure

Program

Compile

Compiled
Code

Deploy

Instrumented
Binary

Hardware

Non-functional
measurable/quantifiable
aspect

Compile
time

$$f_c(c_1) = 11.1ms$$

Execution
time

$$f_e(c_1) = 110.3ms$$

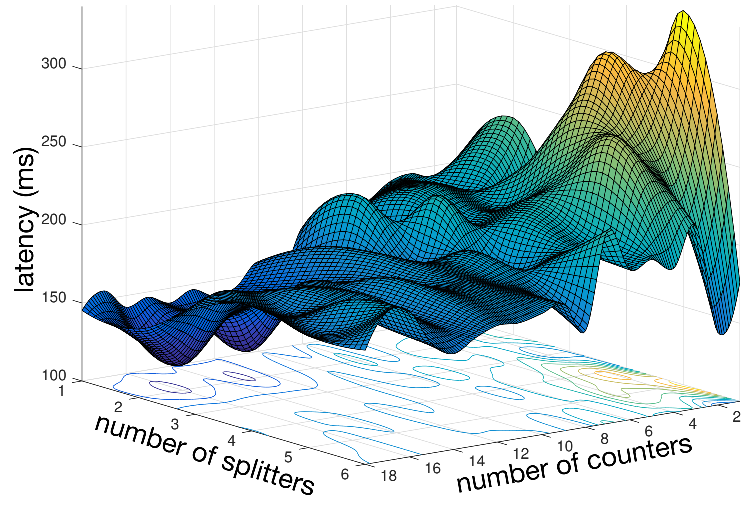
Energy

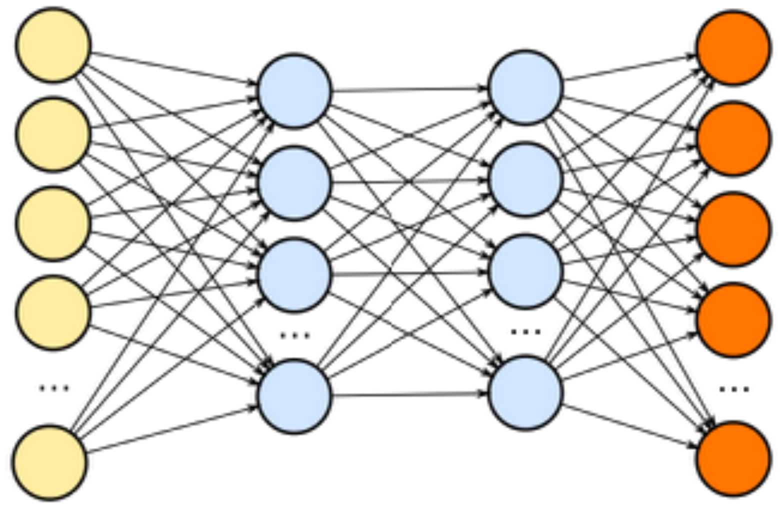
$$f_{en}(c_1) = 100mwh$$

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

Options

Options

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

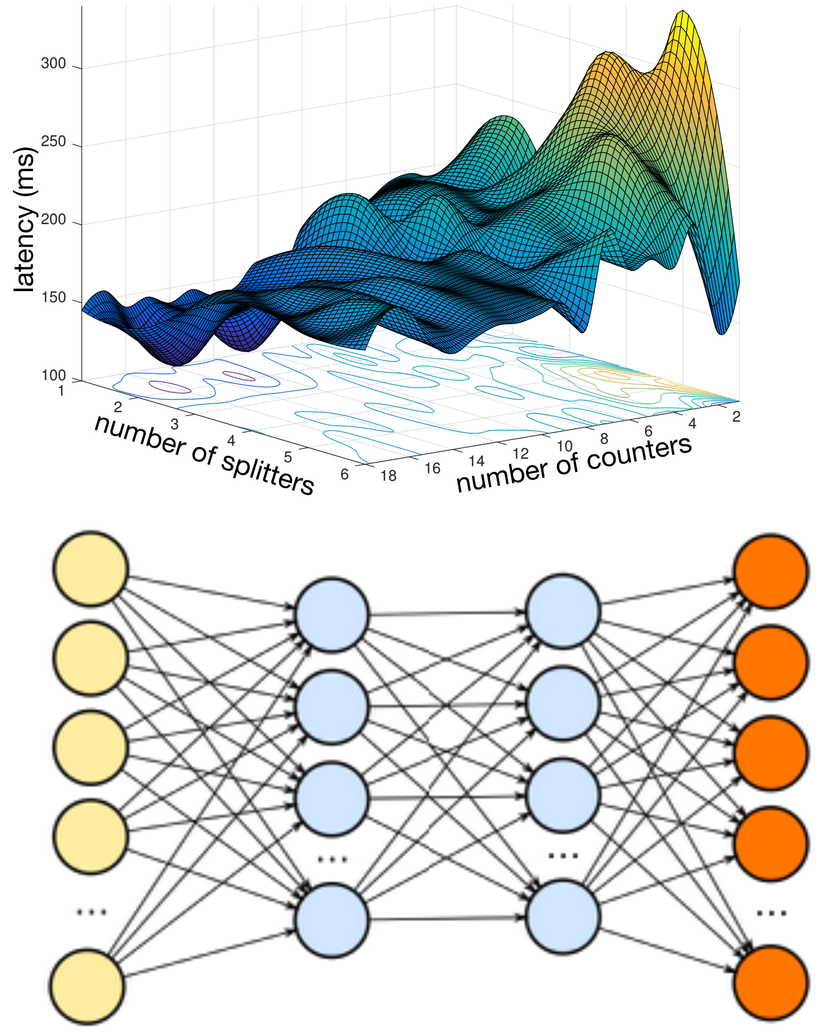
Discovered Interactions

Regression Equation

Performance Influence Models

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

Observational Data



Black-box models

Options

Options

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

Discovered Interactions

Regression Equation

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

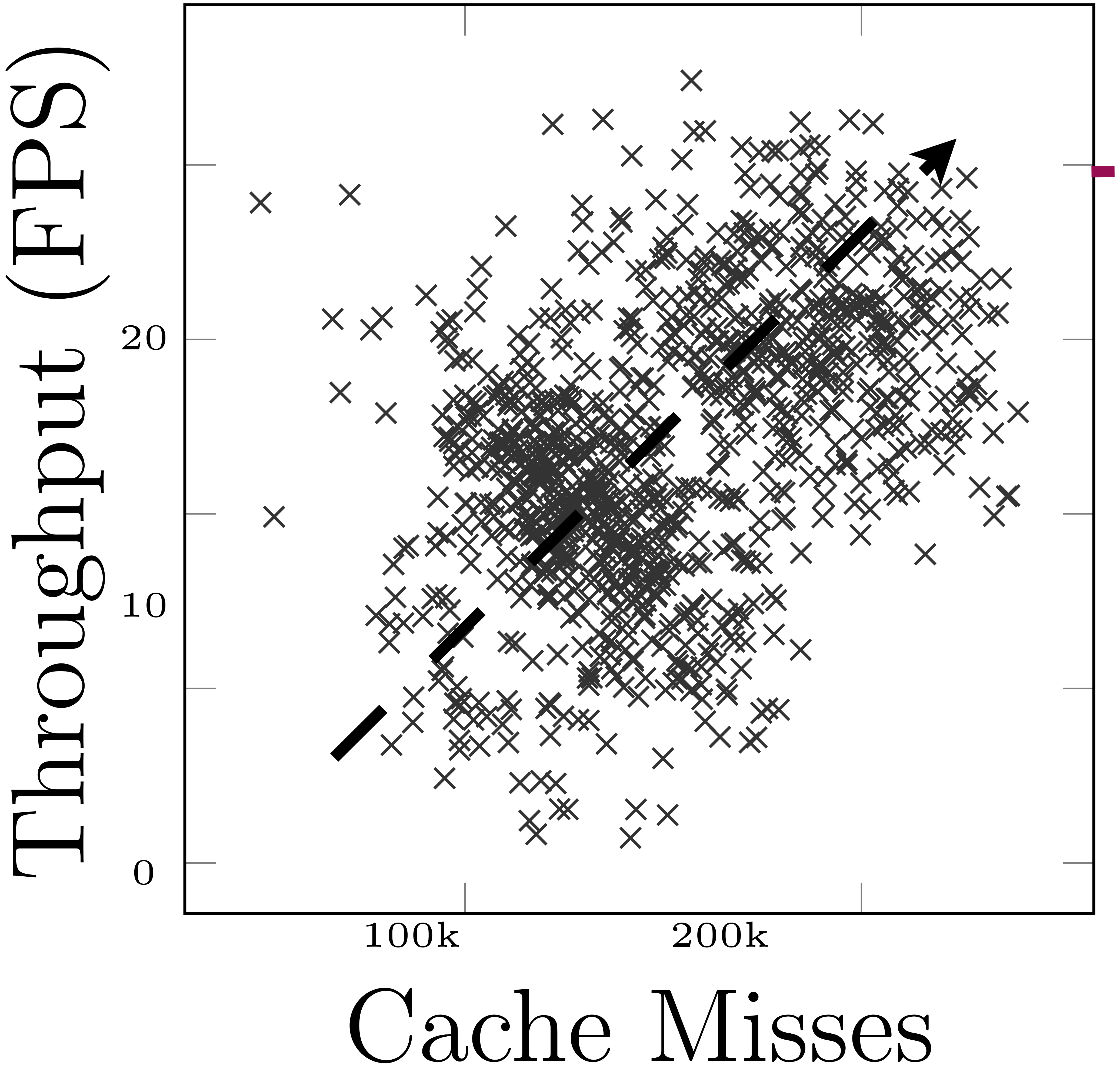
Performance Influence Models suffer from several shortcomings

- Performance influence models could produce **incorrect explanations**

- Performance influence models could produce **unreliable predictions.**

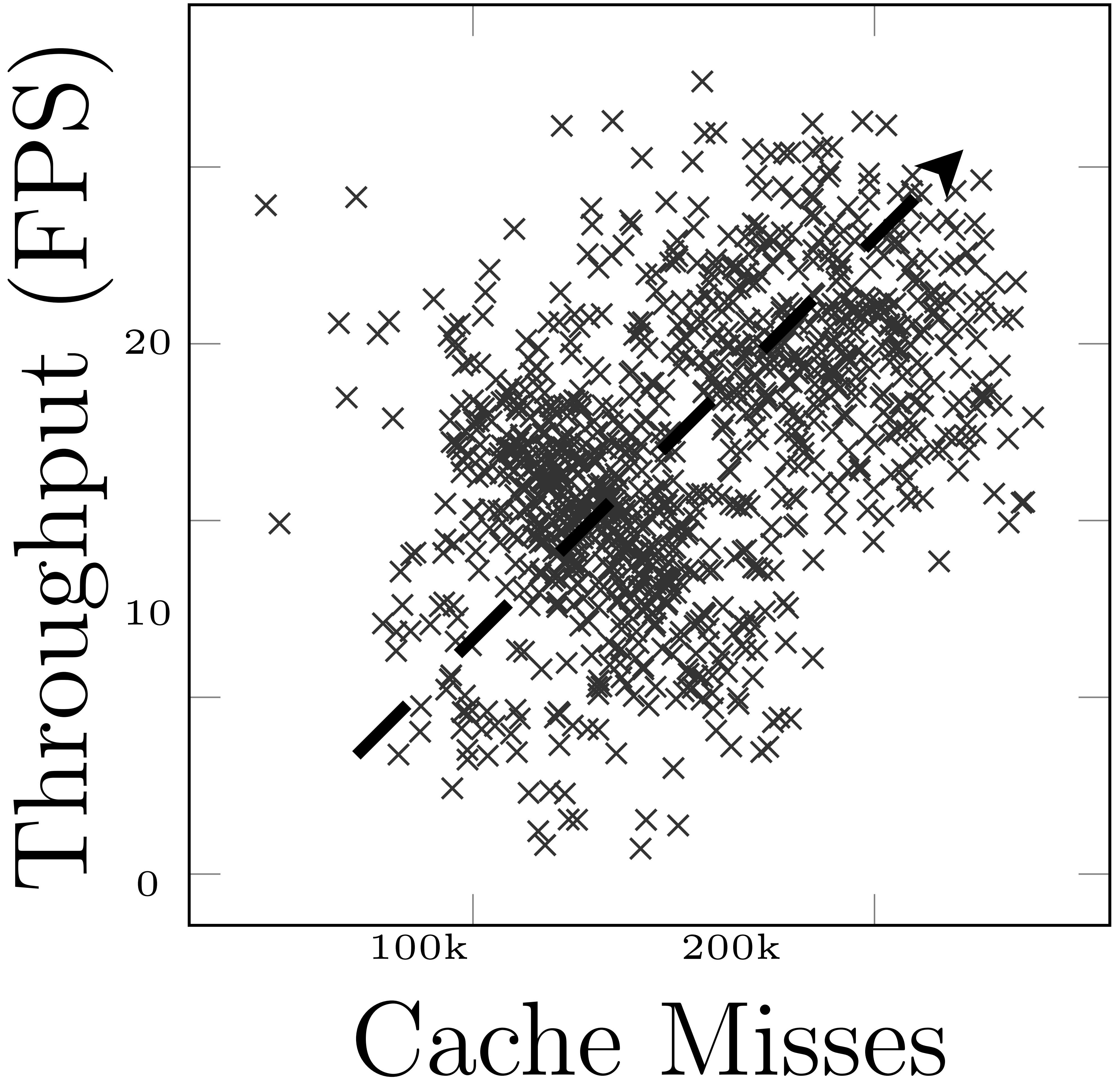
- Performance influence models could produce **unstable predictions** across environments and in the presence of measurement noise.

Performance Influence Models Issue: Incorrect Explanation



Increasing Cache Misses increases Throughput.

Performance Influence Models Issue: Incorrect Explanation



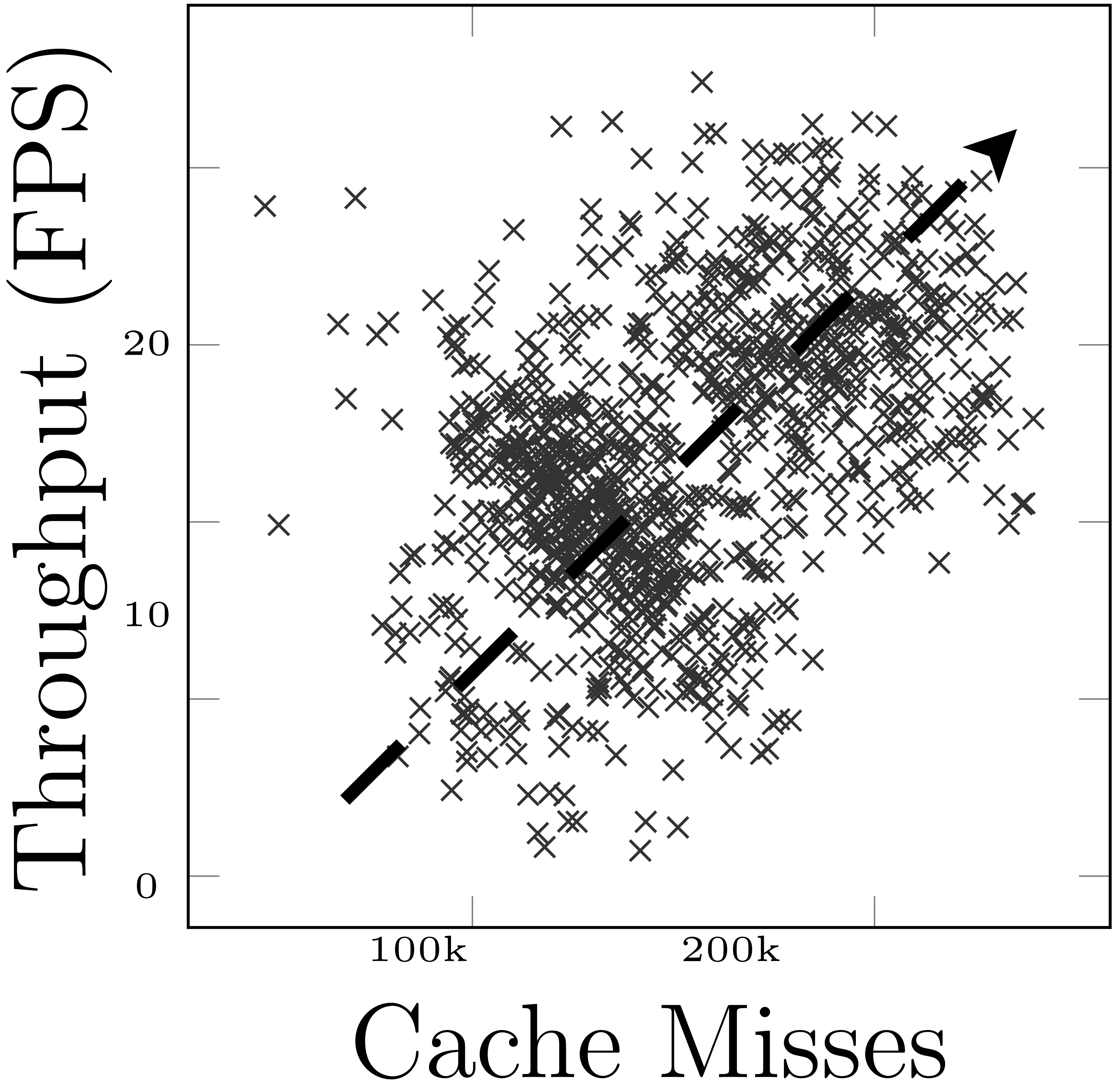
Increasing Cache Misses increases Throughput.

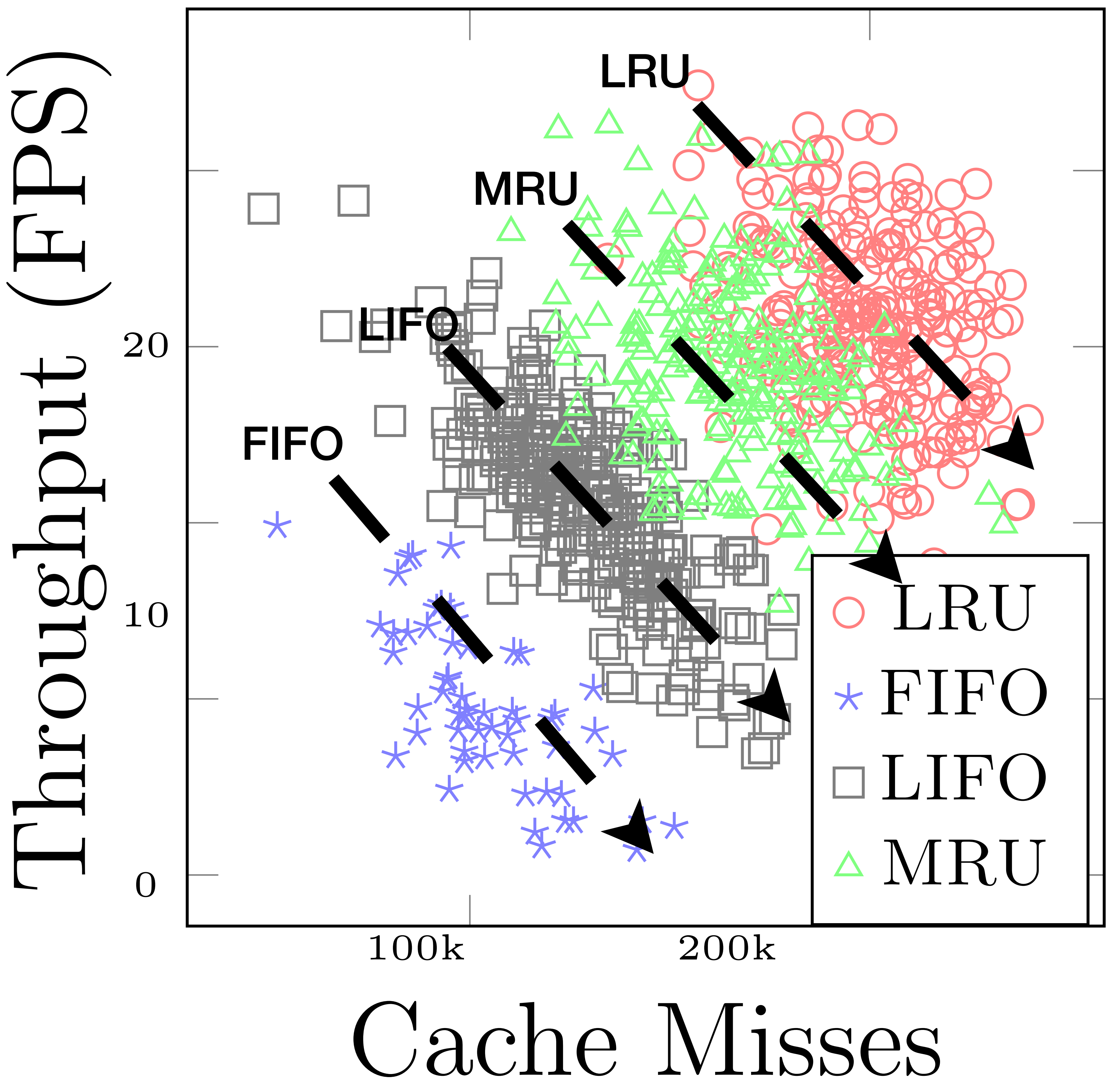
This is counter-intuitive

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.

$$Throughput = 2 \times Bitrate + 1.9 \times BatchSize + \boxed{1.8 \times BufferSize} + \boxed{0.5 \times EnablePadding} + \boxed{5.9 \times Bitrate \times BufferSize} + \boxed{6.2 \times Bitrate \times EnablePadding} + \boxed{4.1 \times Bitrate \times BufferSize \times EnablePadding}$$

Performance influence model in Xavier.

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

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

Performance Influence Models Issue: Unstable Predictors

Performance influence model in TX2.

$$Throughput = \boxed{2 \times Bitrate} + \boxed{1.9 \times BatchSize} + 1.8 \times BufferSize + 0.5 \times EnablePadding + 5.9 \times Bitrate \times BufferSize$$
$$+ 6.2 \times Bitrate \times EnablePadding + 4.1 \times Bitrate \times BufferSize \times EnablePadding$$

Performance influence model in Xavier.

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

Performance influence are cannot be reliably used across environments.

Performance Influence Models Issue: Non-generalizability

Performance influence model in TX2

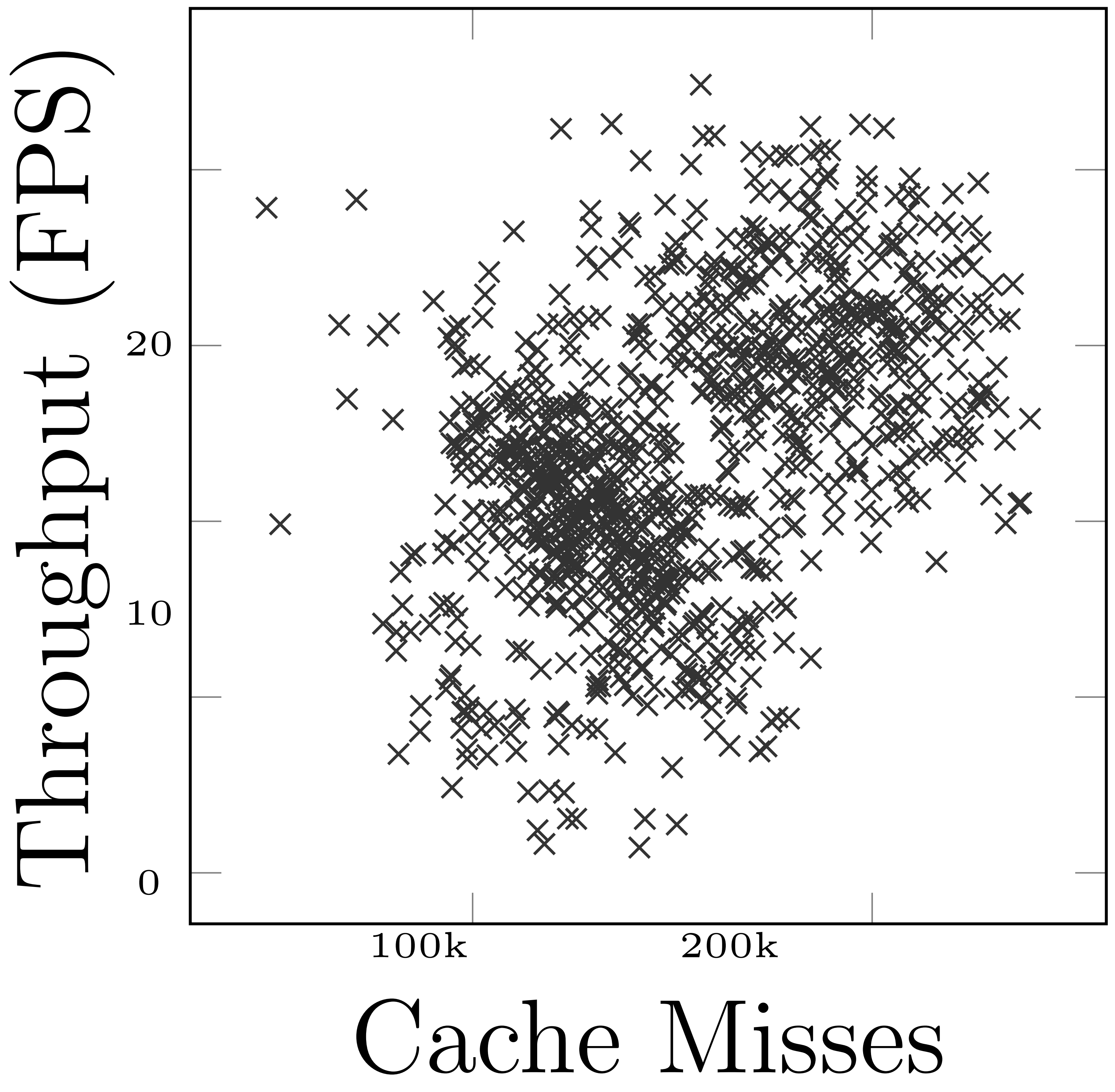
$$Throughput = 2 \times Bitrate + 1.9 \times BatchSize + 1.8 \times BufferSize + 0.5 \times EnablePadding + 5.9 \times Bitrate \times BufferSize$$
$$+ 6.2 \times Bitrate \times EnablePadding + 4.1 \times Bitrate \times BufferSize \times EnablePadding$$

Performance influence model in Xavier.

$$Throughput = 5.1 \times Bitrate + 2.5 \times BatchSize + 12.3 \times Bitrate \times BatchSize$$

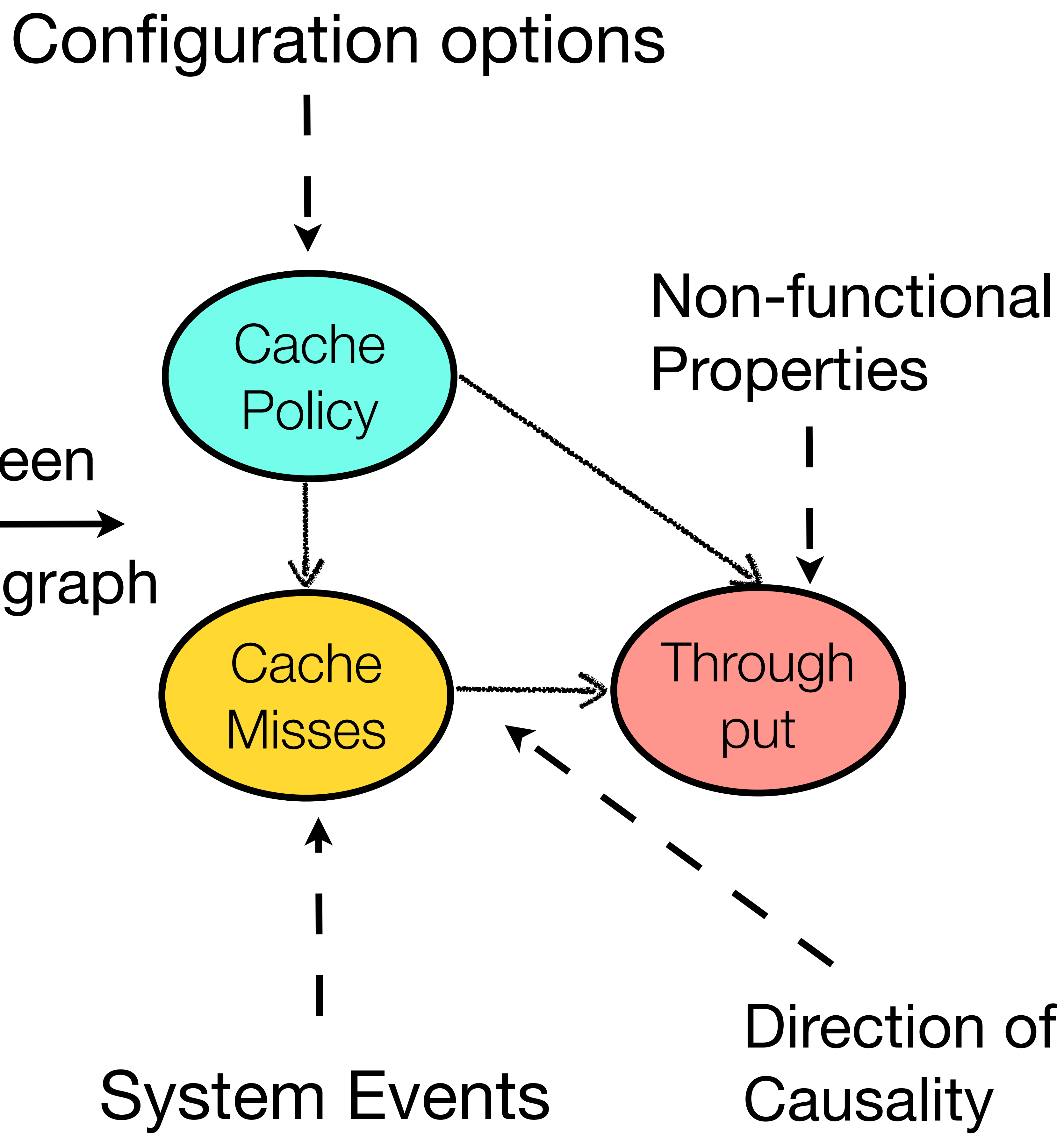
Performance influence models do not generalize well across deployment environments.

Causal Performance Model

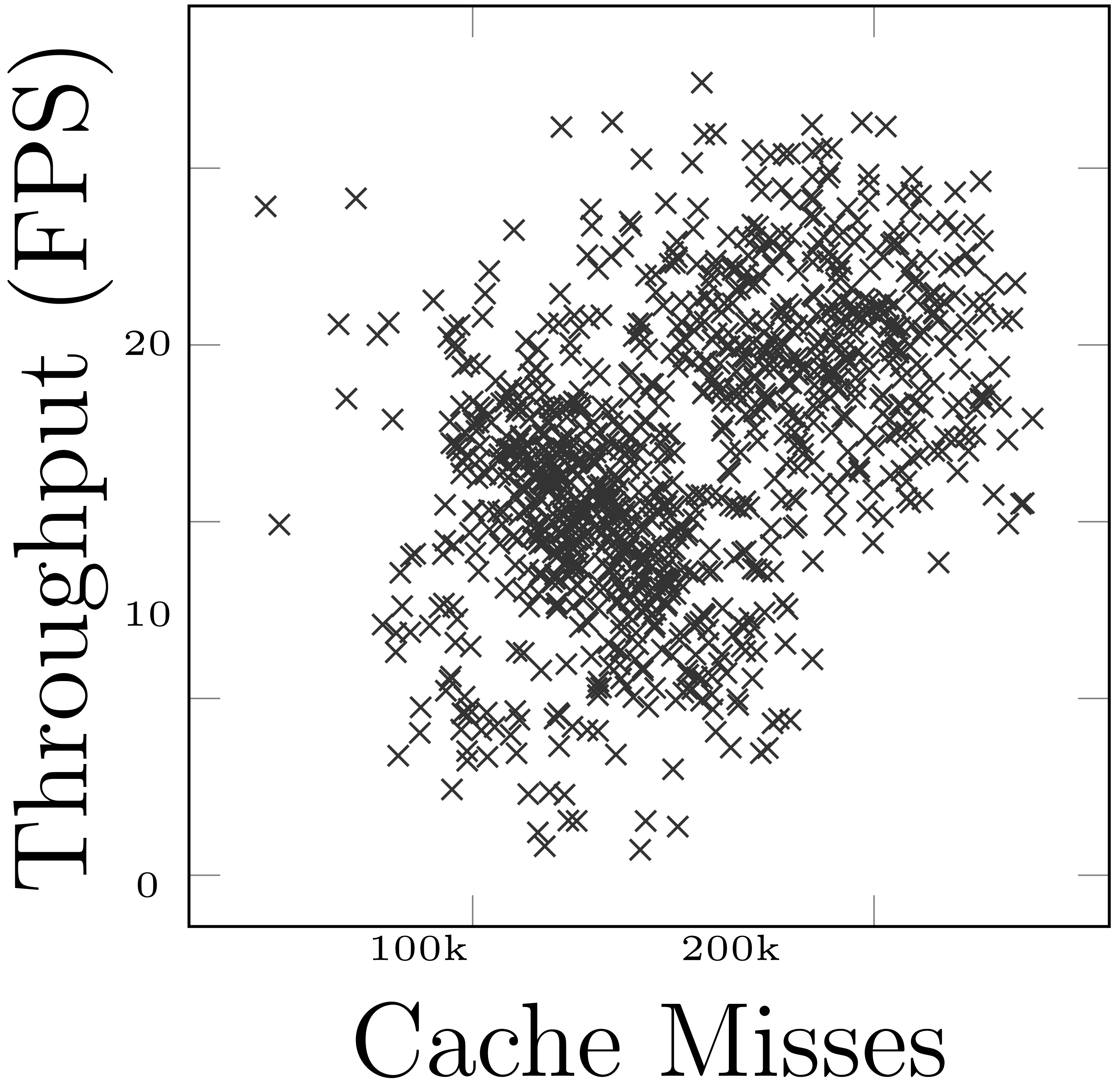


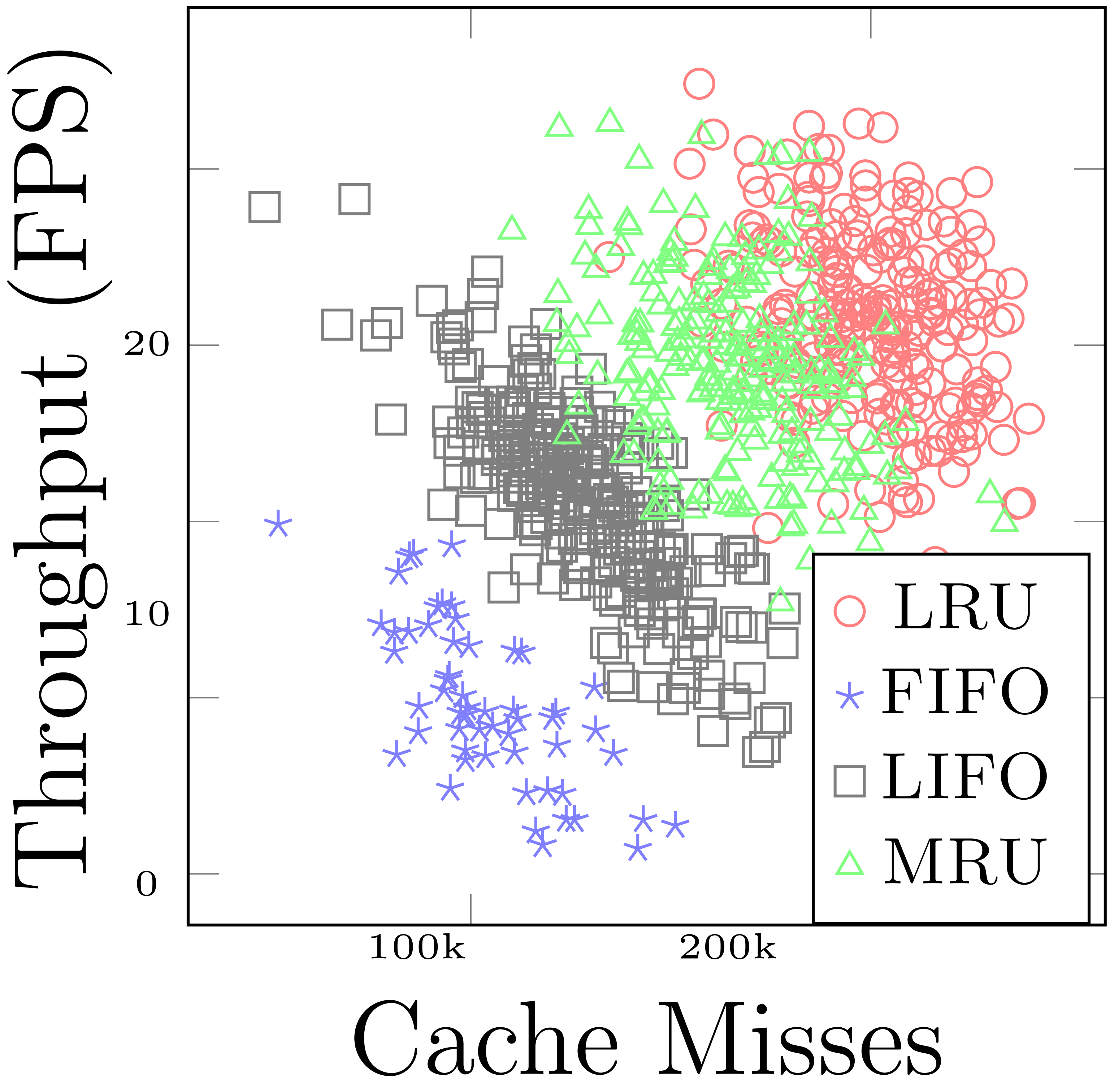
Expresses the relationships between

interacting variables as a causal graph

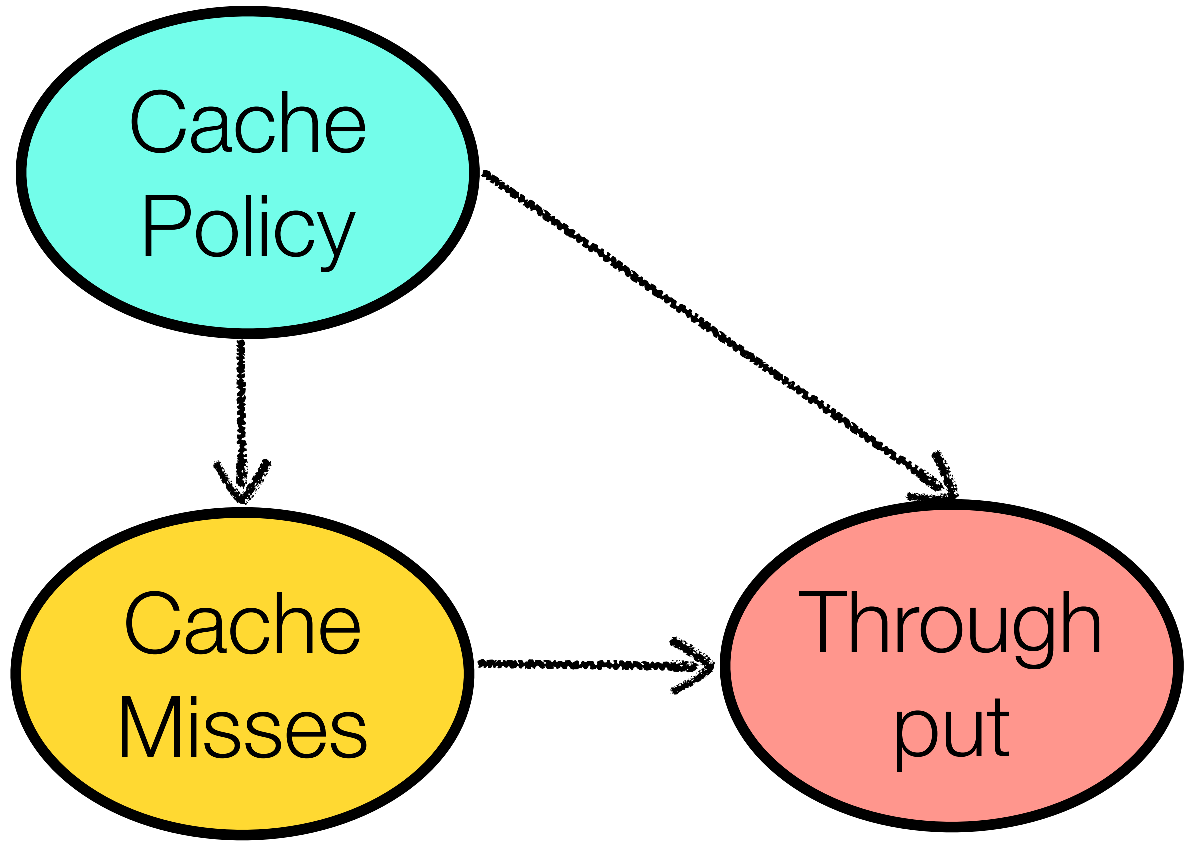


Why Causal Inference? - Produces Correct Explanations



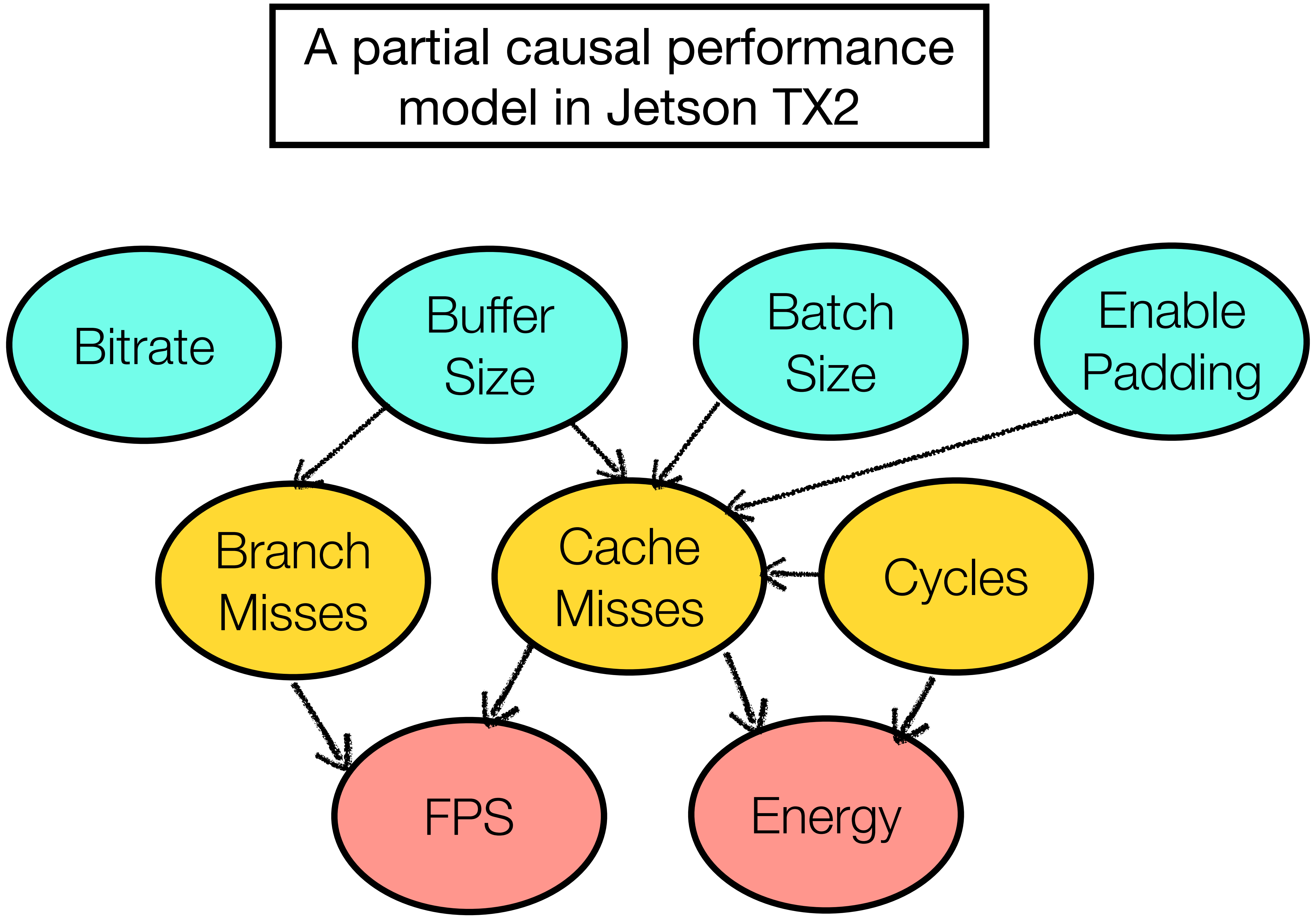


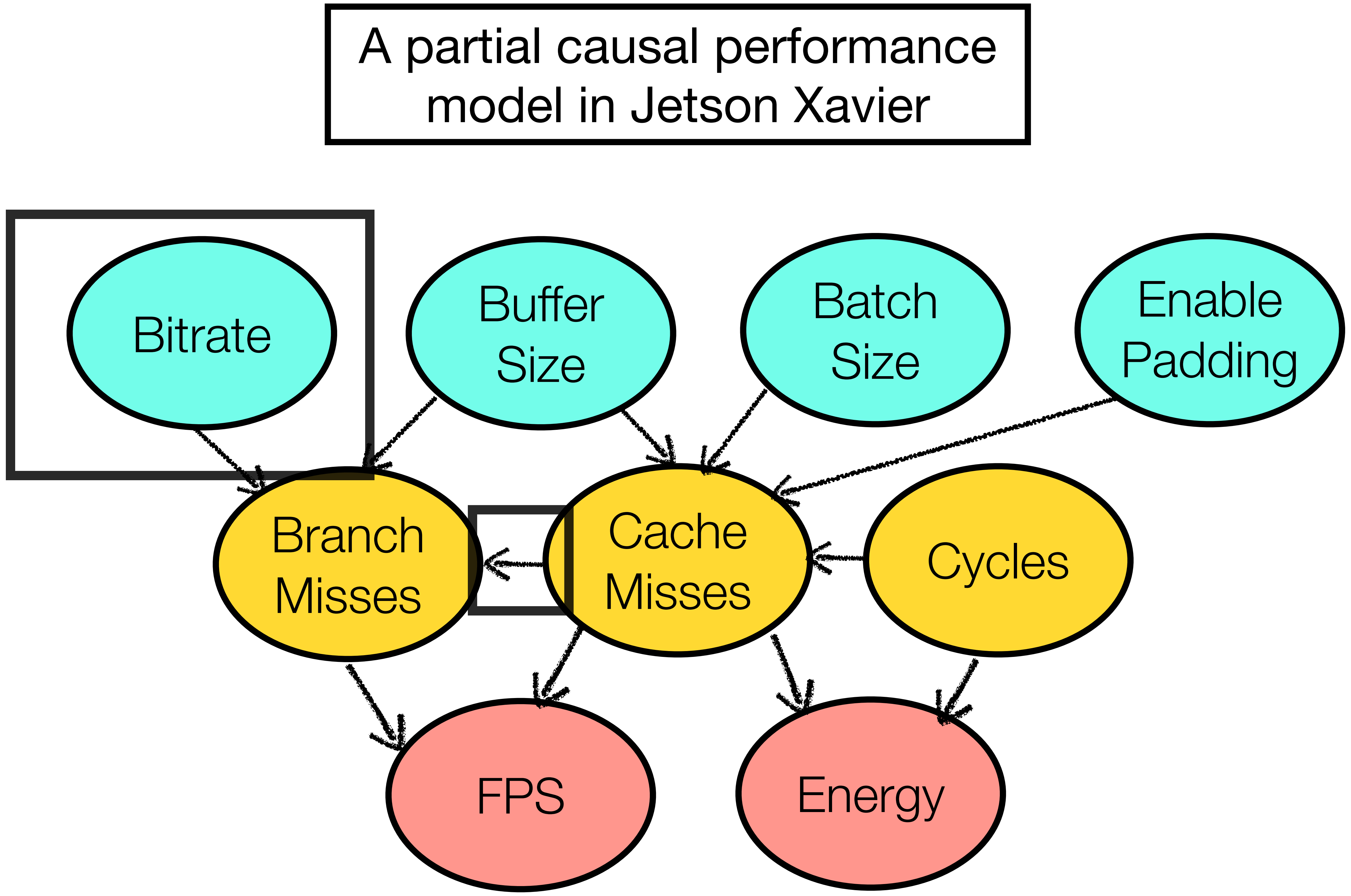
Cache Policy affects Throughput via Cache Misses.



Causal Performance Models recovers the correct interactions.

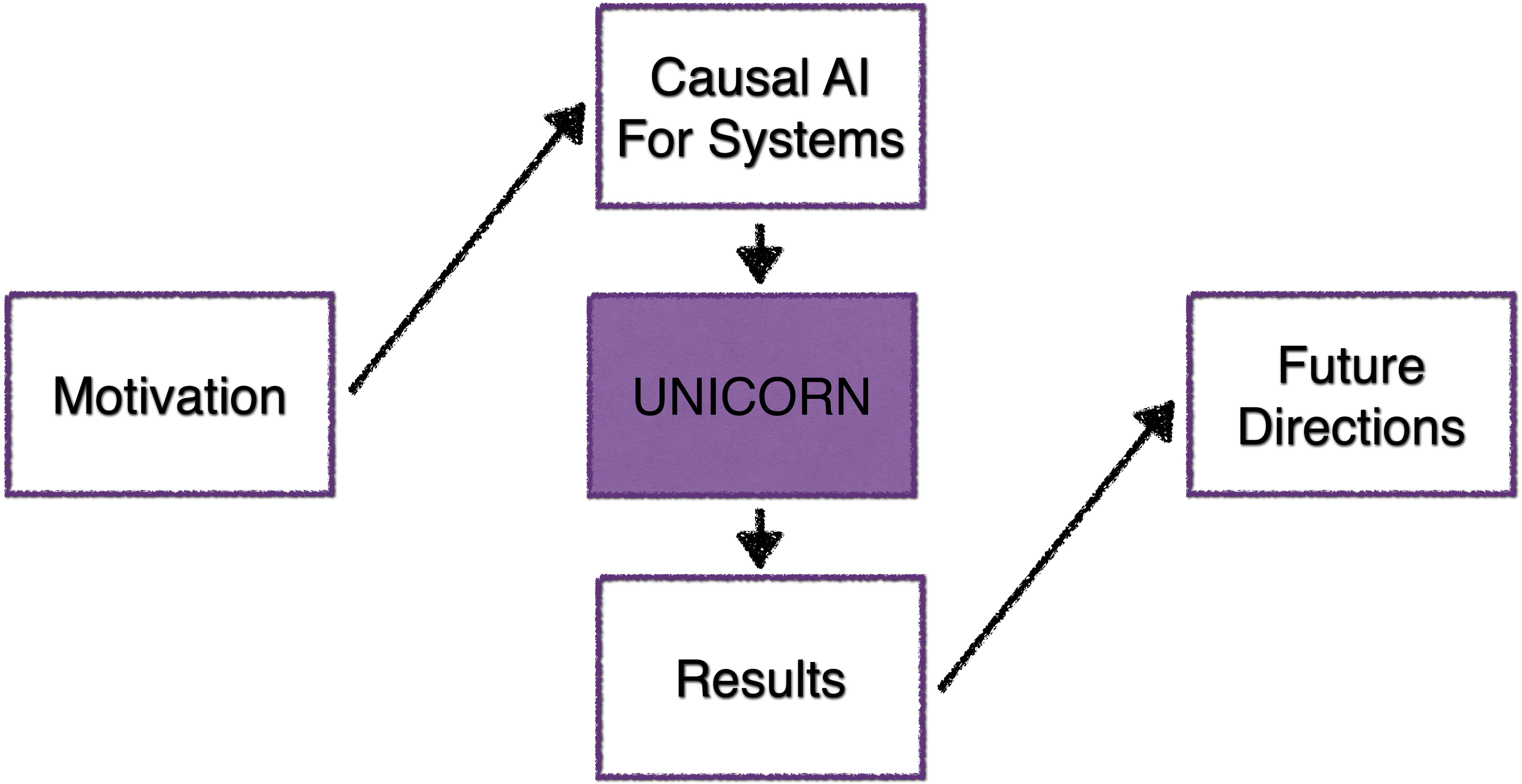
Why Causal Inference? - Minimal Structure Change





Causal models remain relatively stable

Outline


Causal AI
For Systems
Motivation
UNICORN
Future
Directions
Results

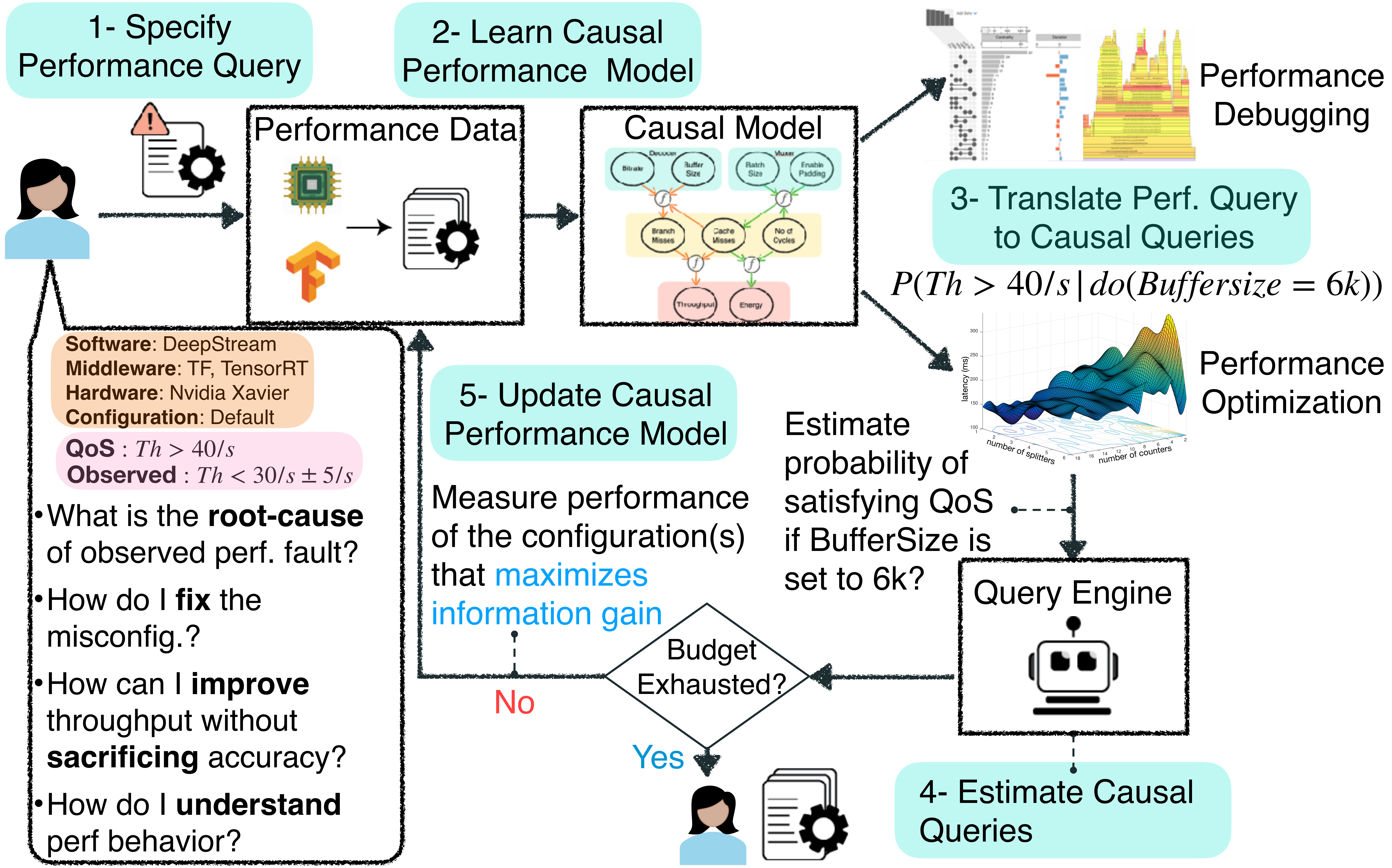
UNICORN: Our Causal AI for Systems Method

- **Build a Causal Performance Model** that capture the interactions options in the variability space using the observation performance data.

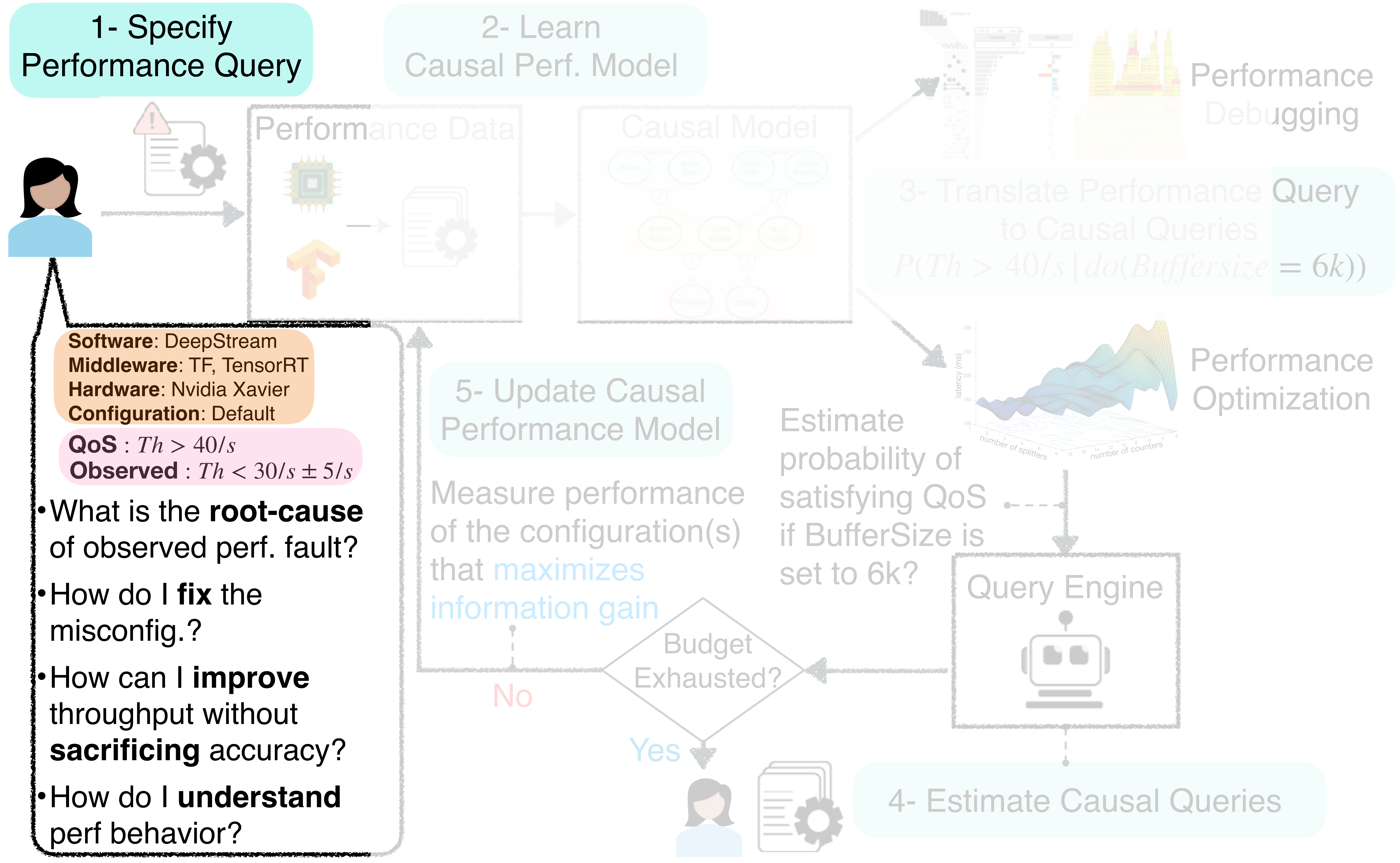
- Iterative causal performance **model evaluation** and **model update**

- Perform downstream performance tasks such as performance debugging & optimization using **Causal Reasoning**

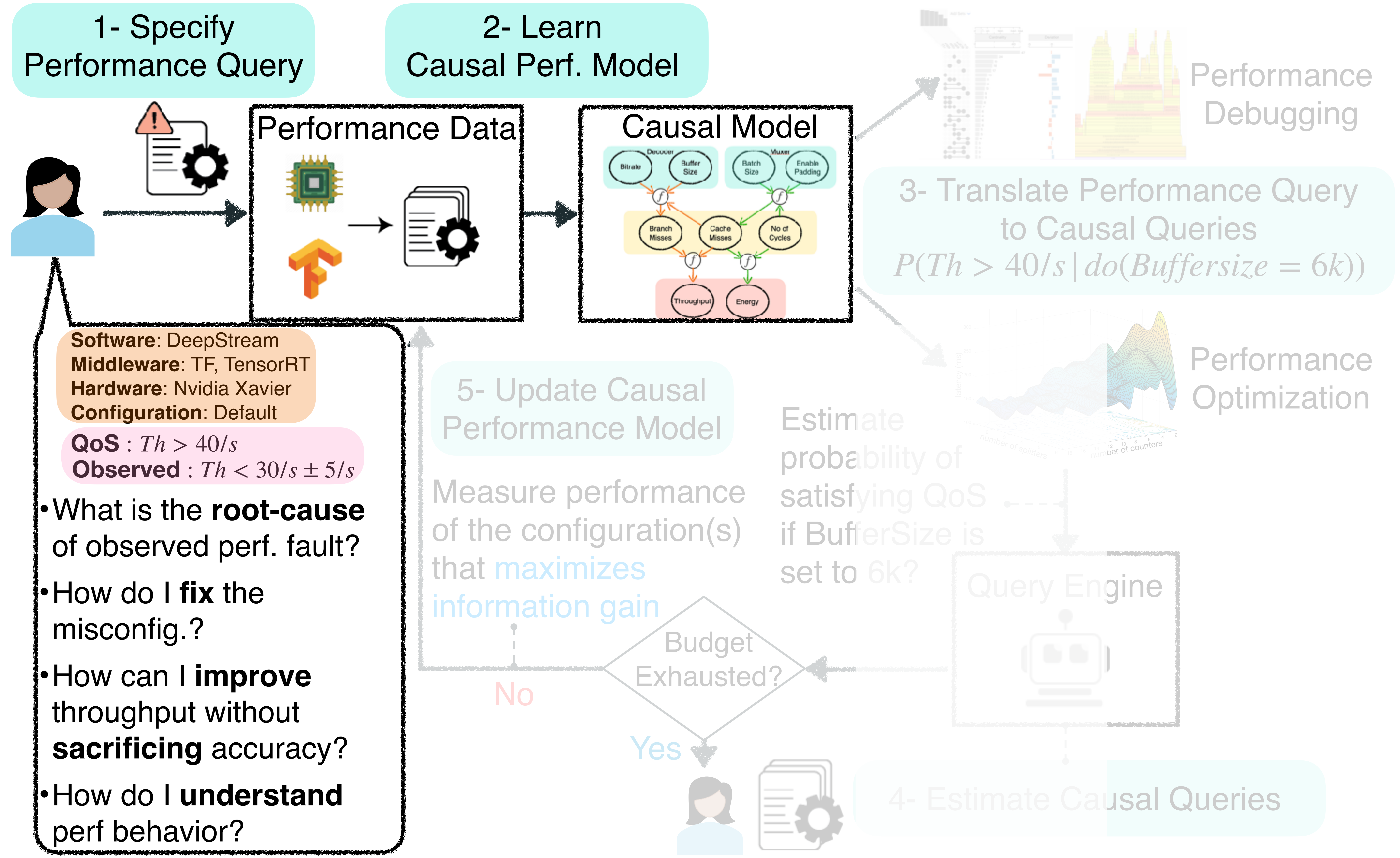
UNICORN: Our Causal AI for Systems Method



UNICORN: Our Causal AI for Systems Method



UNICORN: Our Causal AI for Systems Method

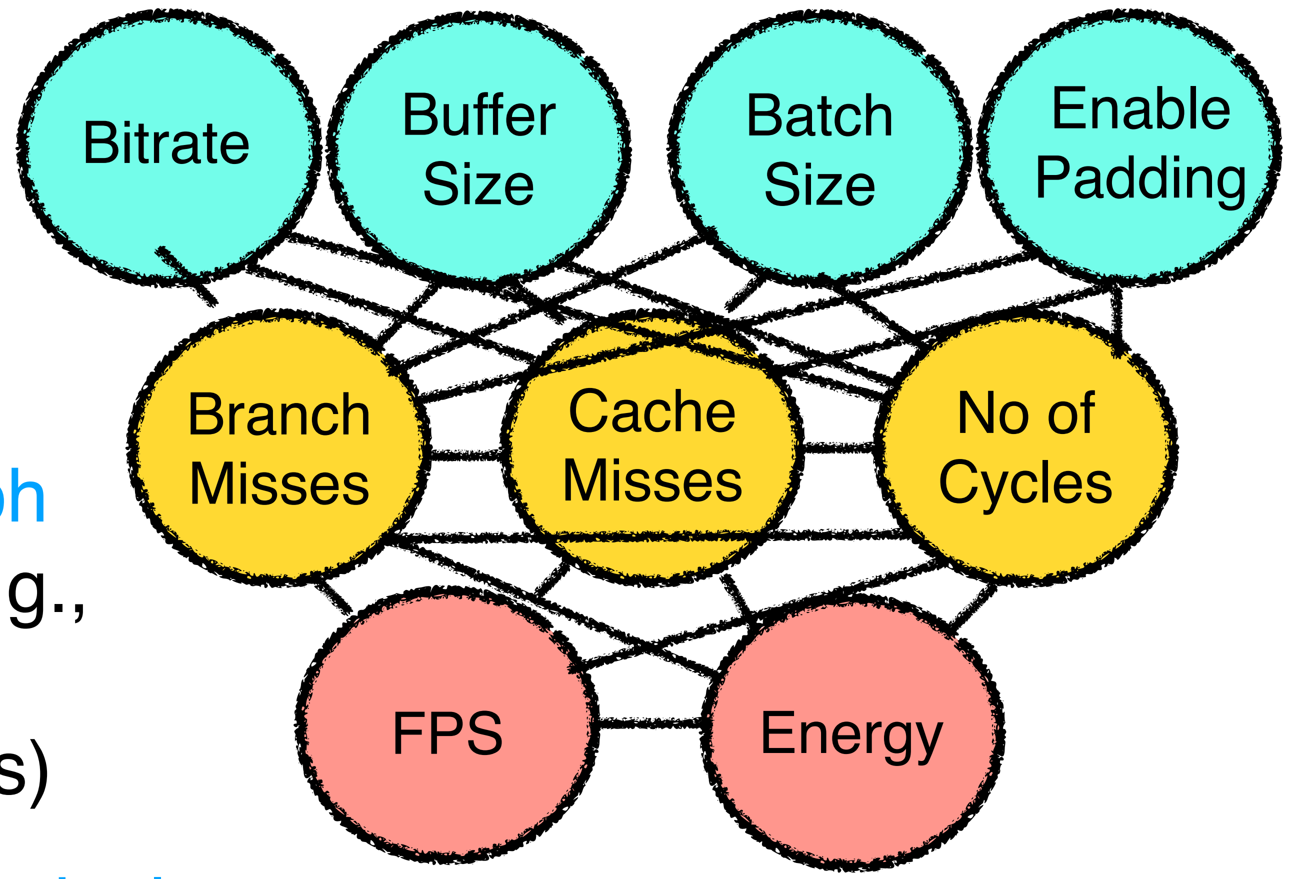


Learning Causal Performance Model

	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

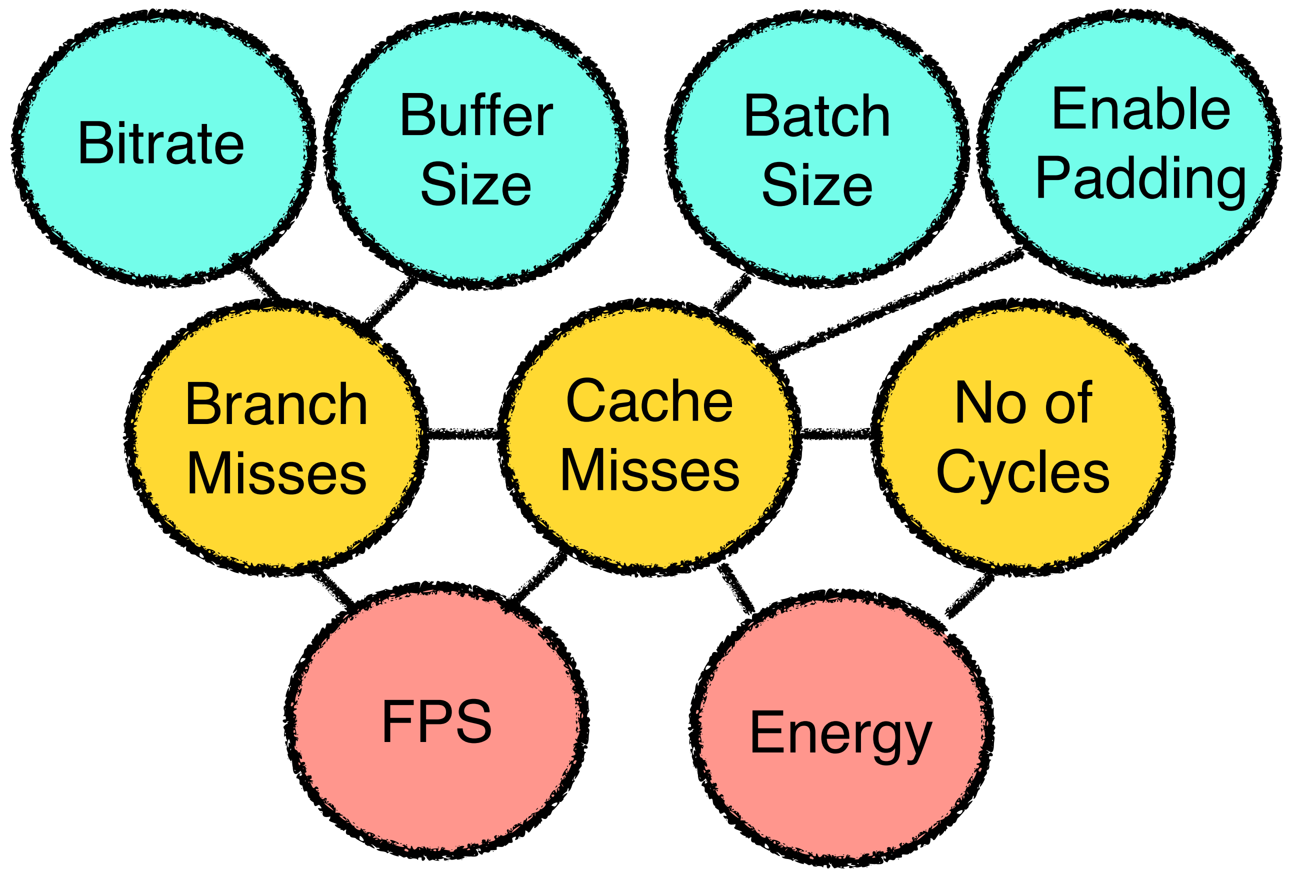
1- Recovering the Skelton

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



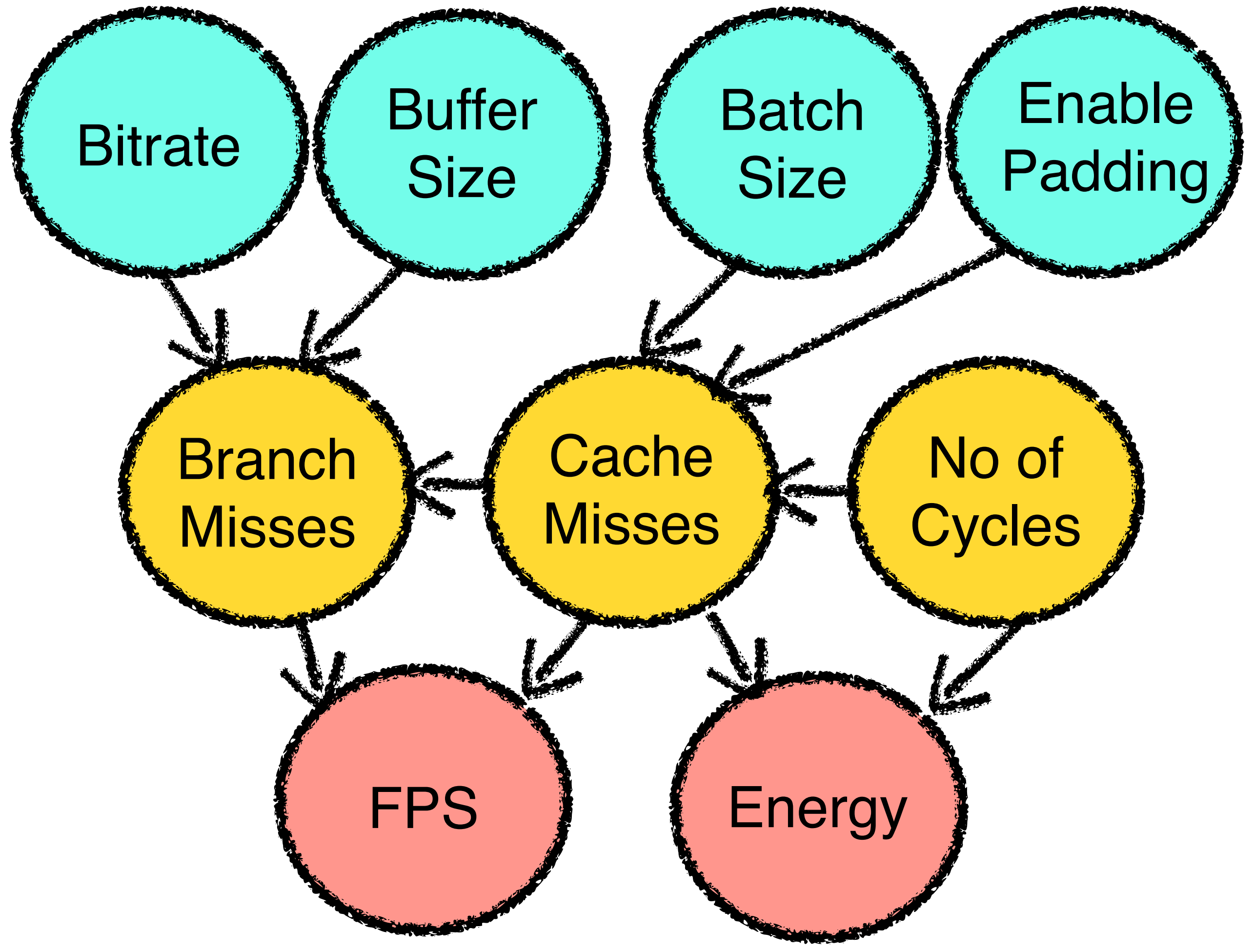
statistical independence tests

2- Pruning Causal Structure

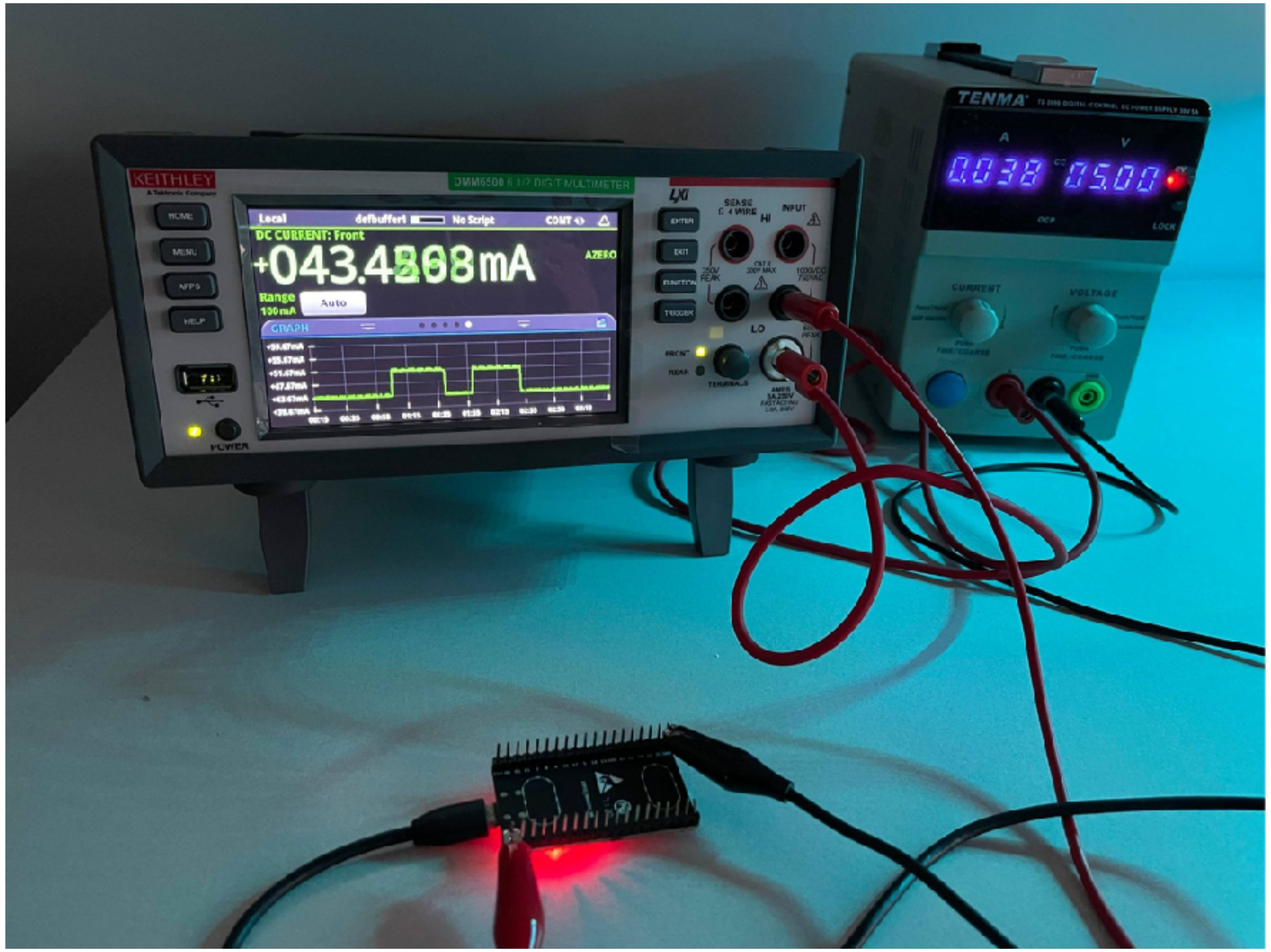


orientation rules & measures (entropy) + structural constraints (colliders, v-structures)

3- Orienting Causal Relations



Our setup for performance measurements

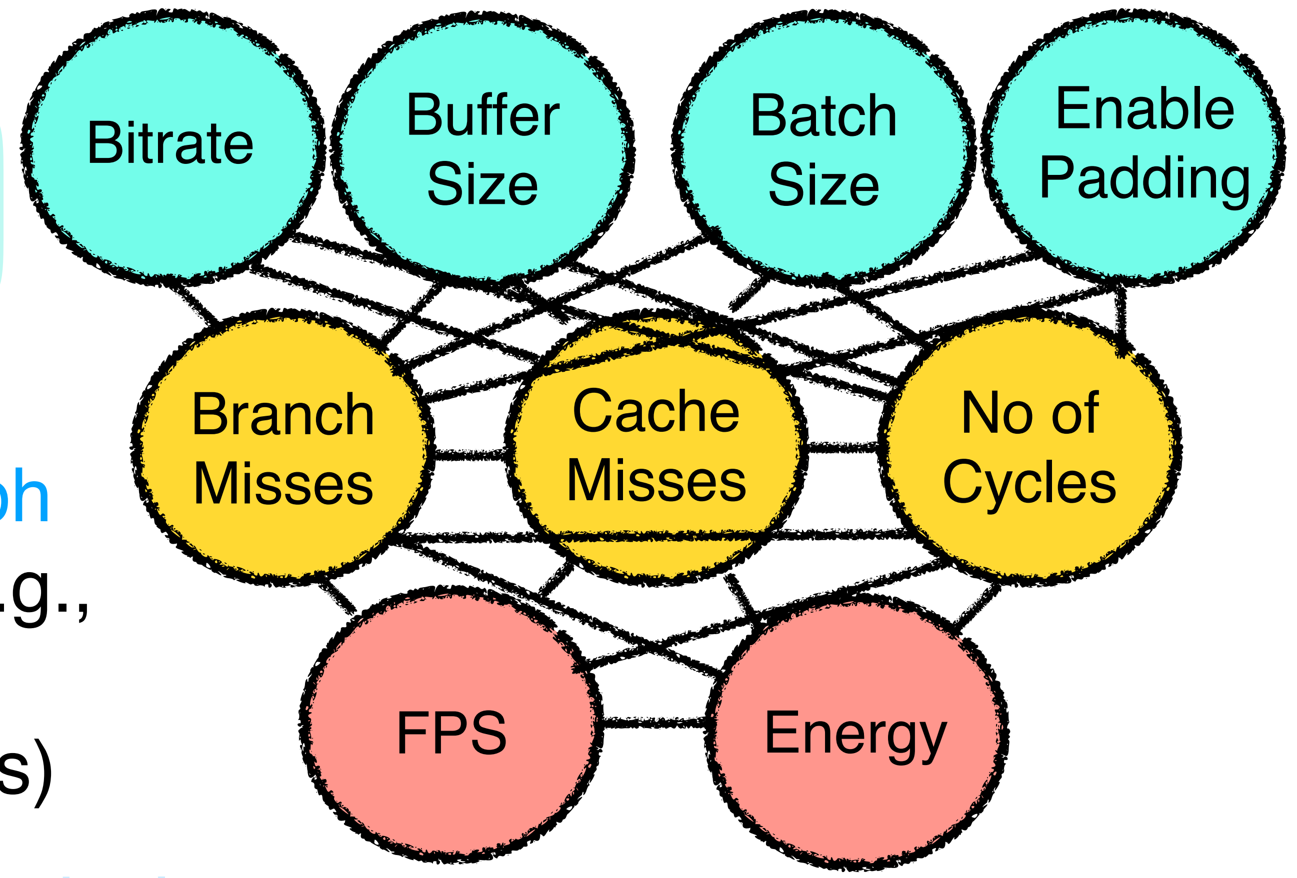


Learning Causal Performance Model

	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)

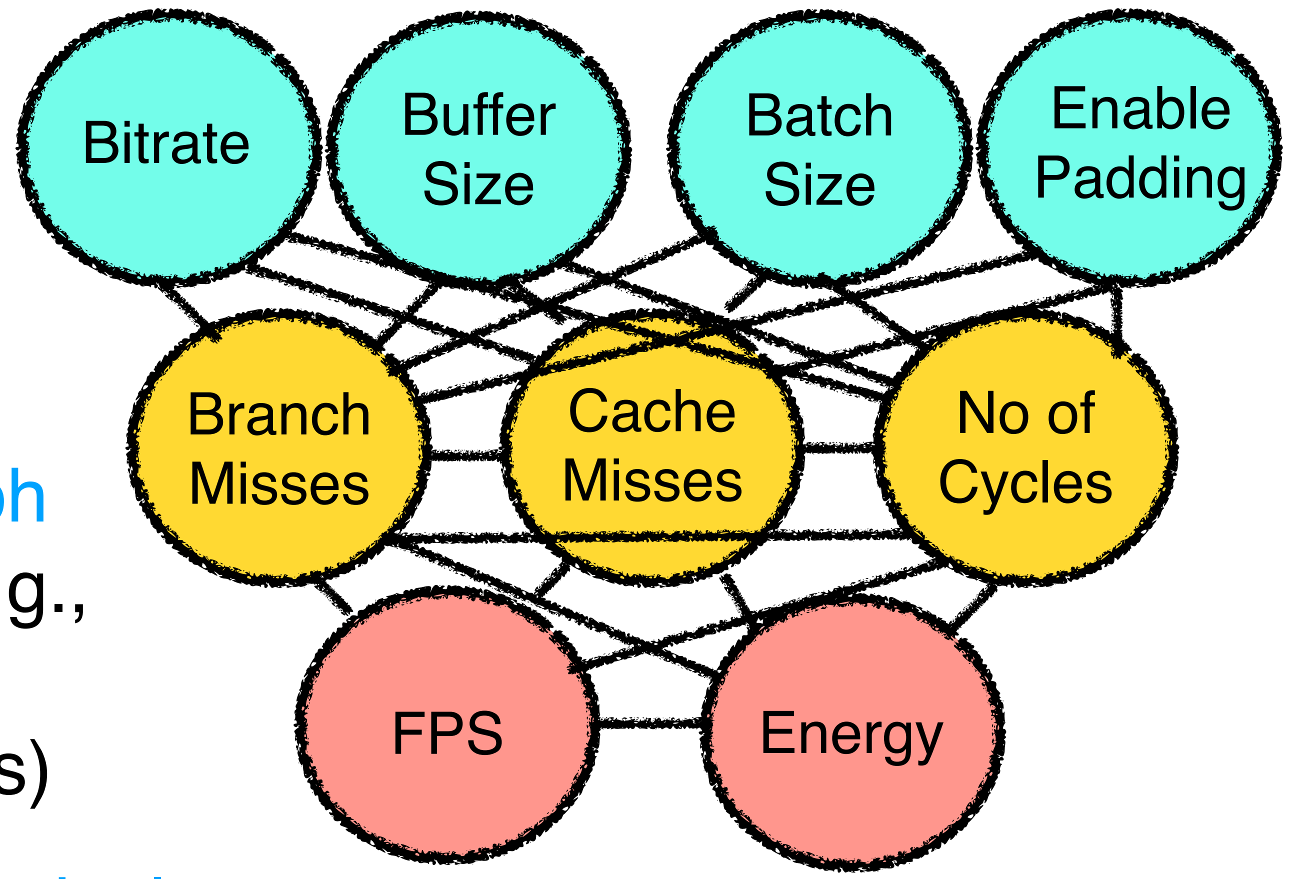
3- Orienting Causal Relations

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

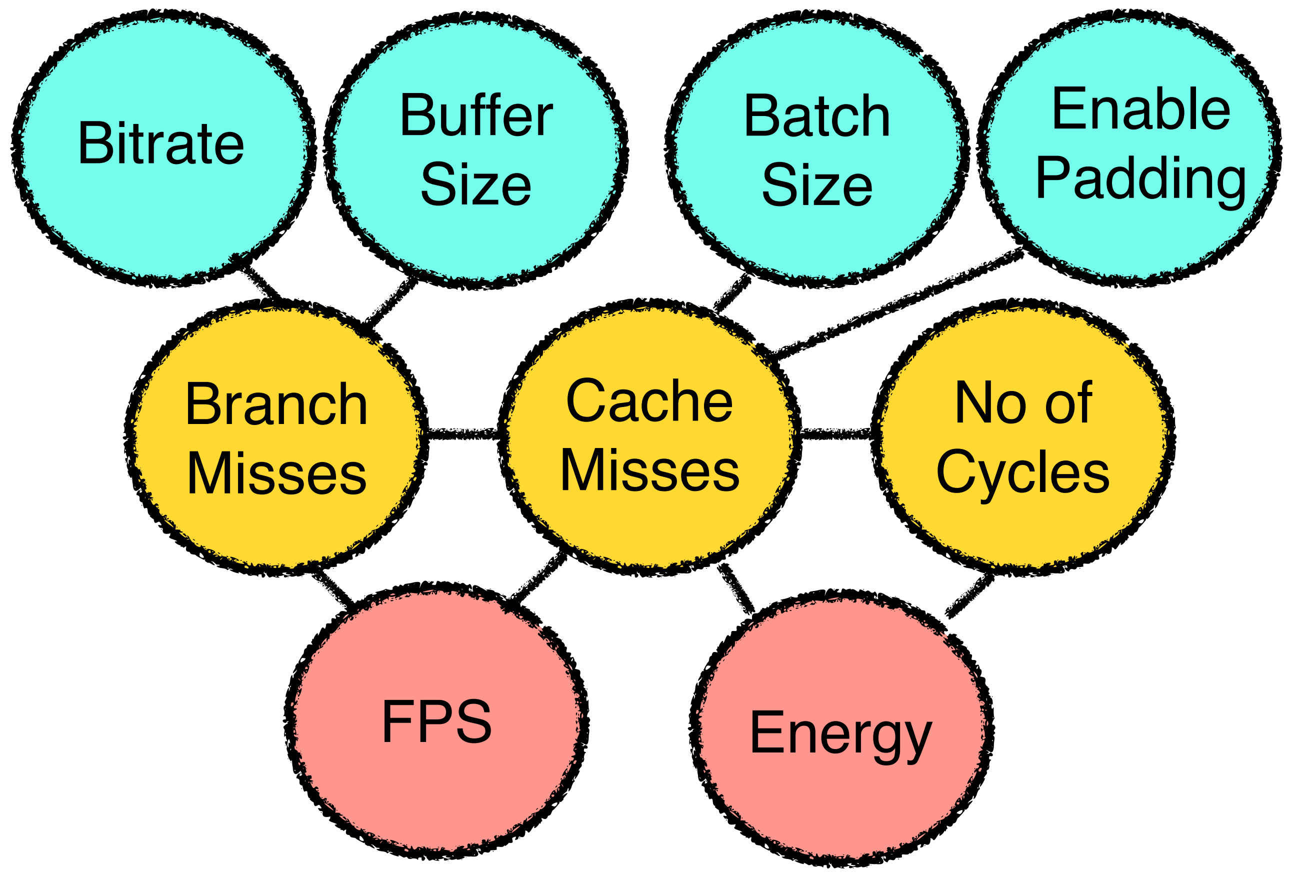
1- Recovering the Skelton

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



statistical independence tests

2- Pruning Causal Structure



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

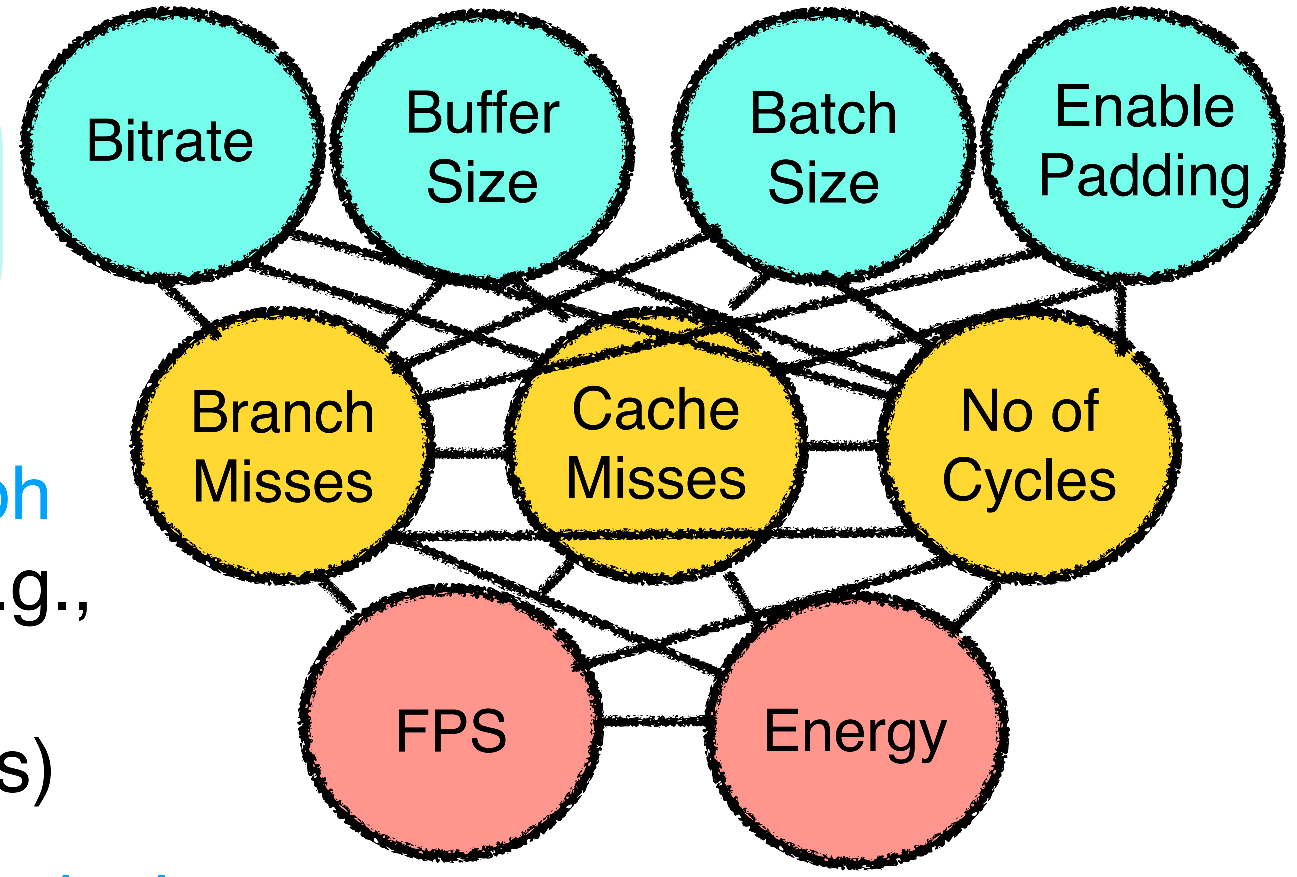
3- Orienting Causal Relations

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

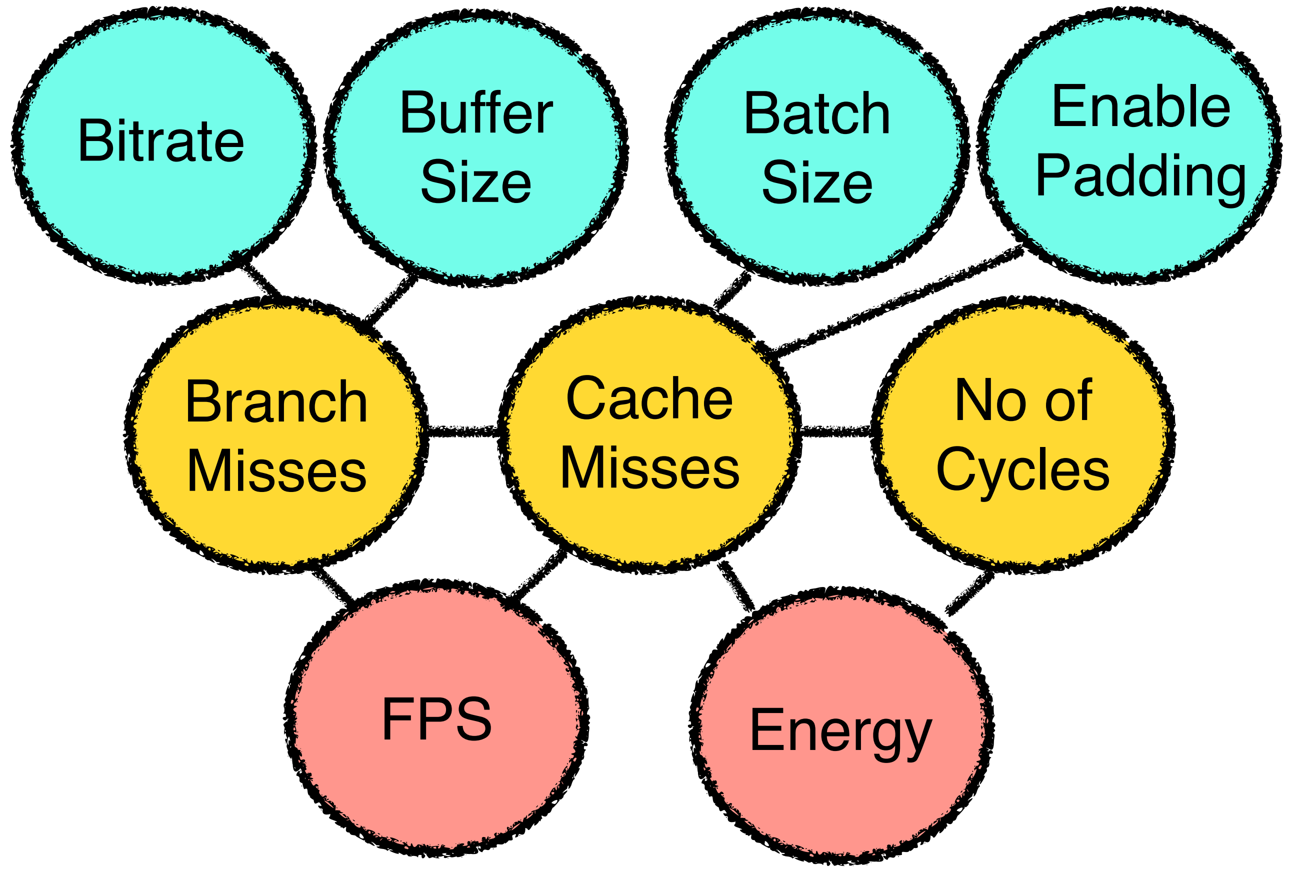
1- Recovering the Skelton

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



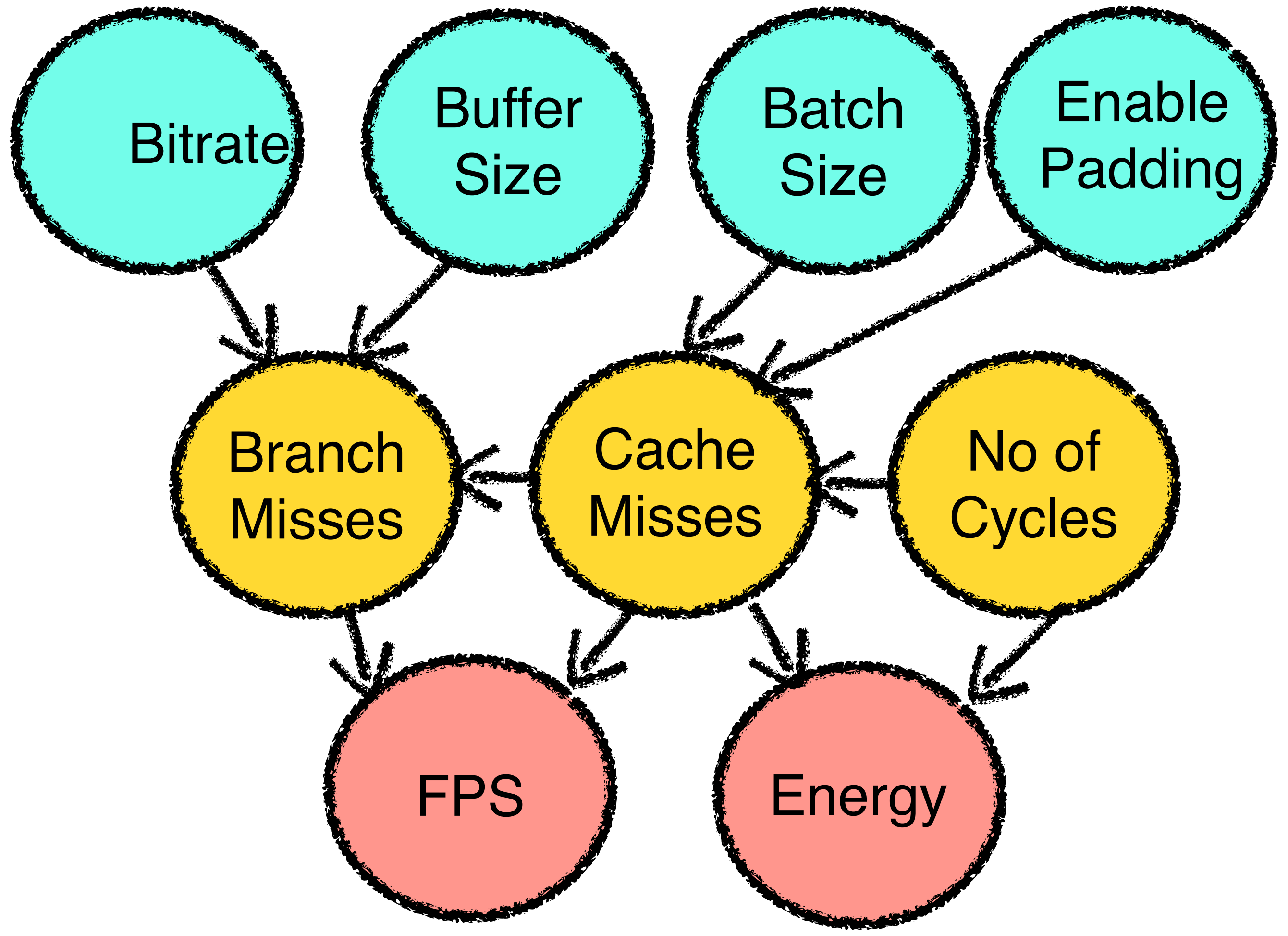
statistical independence tests

2- Pruning Causal Structure



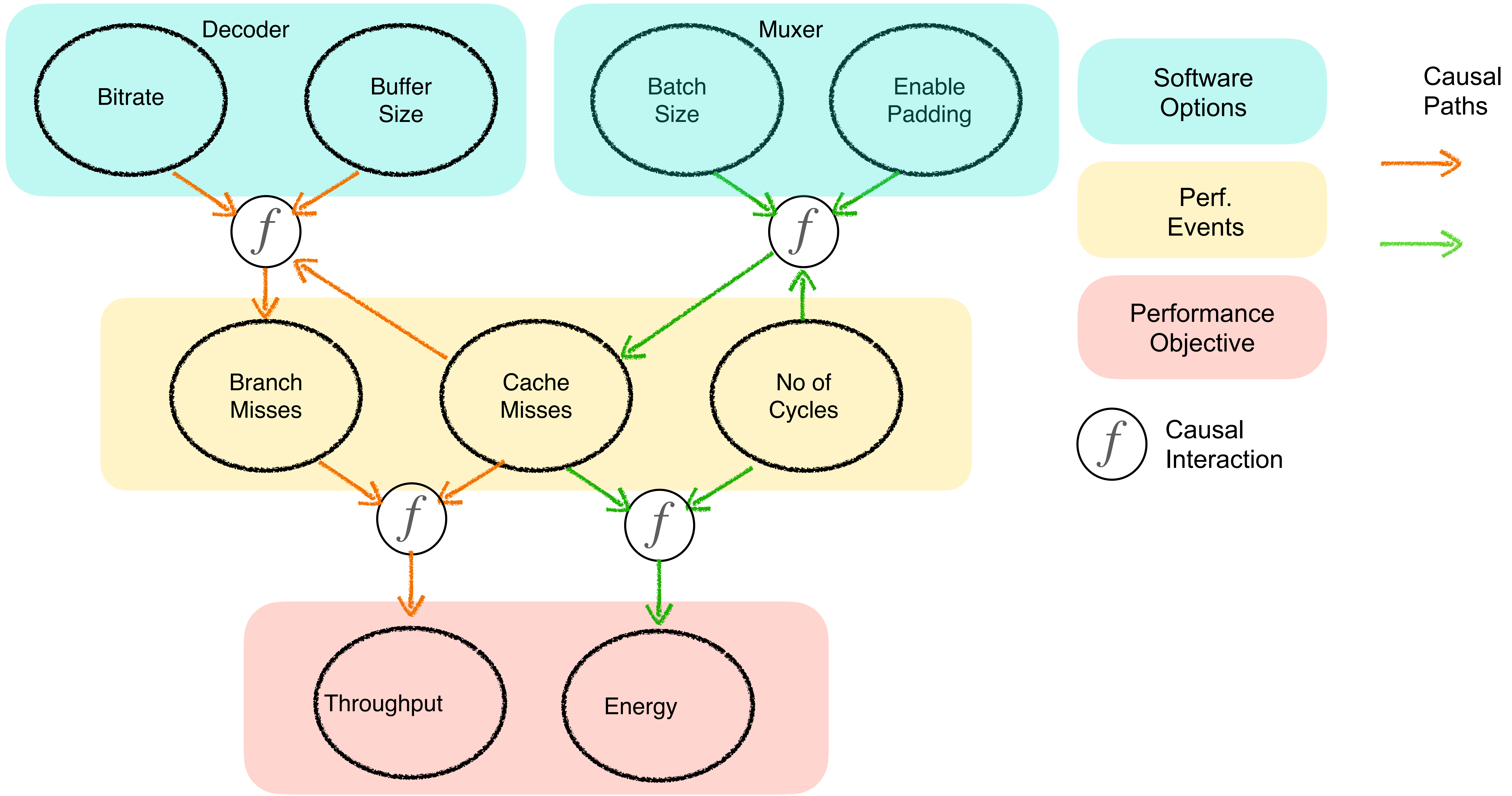
orientation rules & measures (entropy) + structural constraints (colliders, v-structures)

3- Orienting Causal Relations

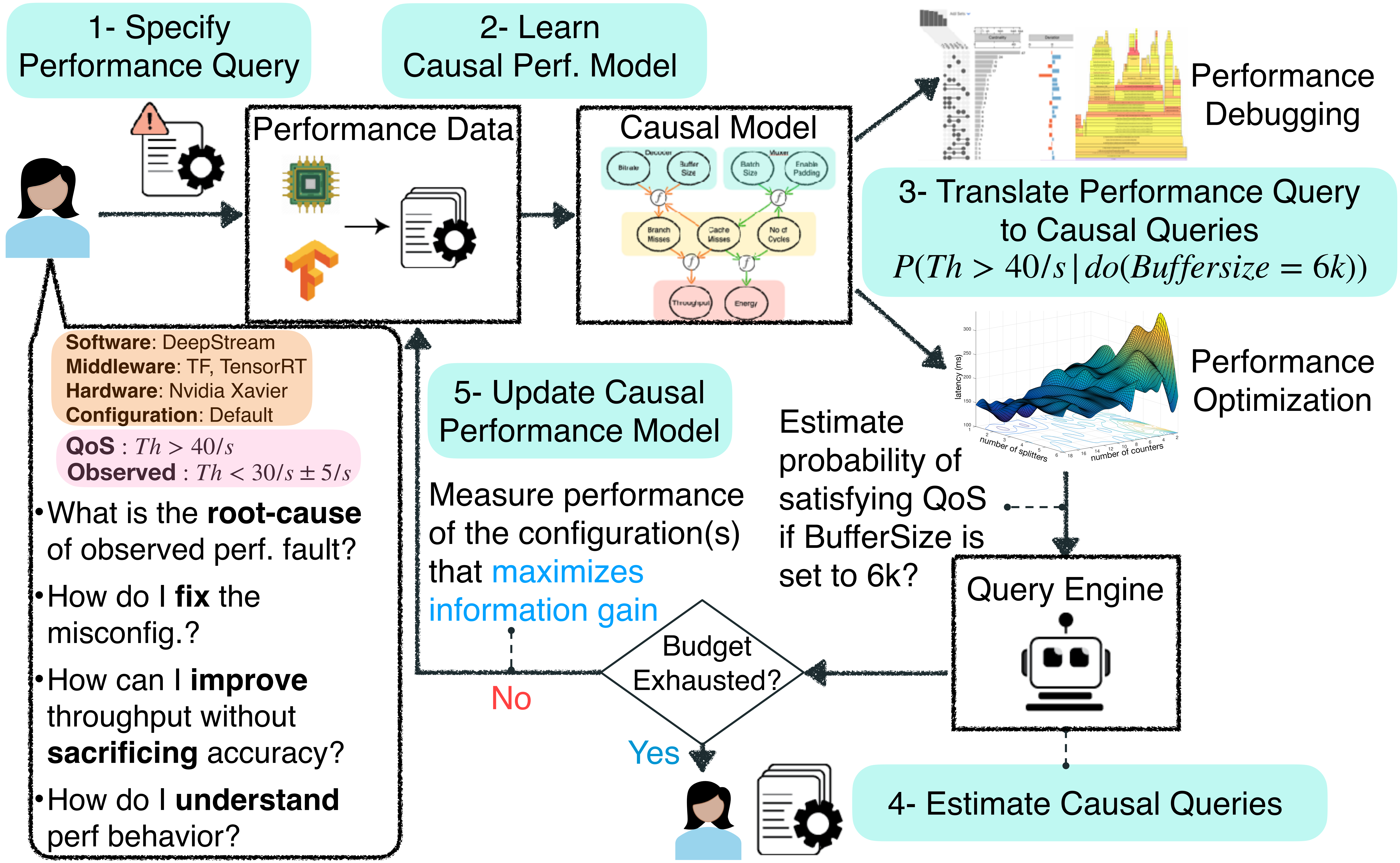


Causal Performance Model

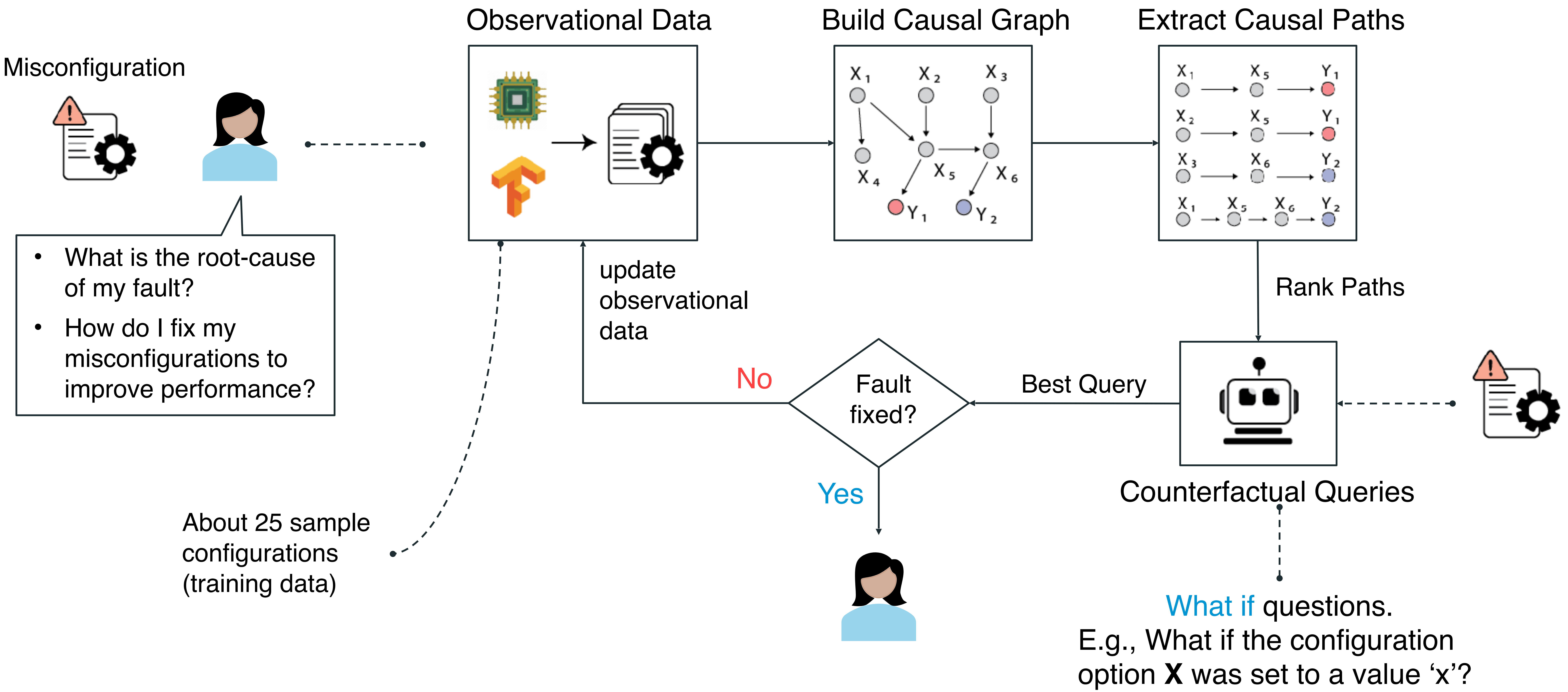
$$Branchmisses = 2 \times Bitrate + 8.1 \times Buffersize + 4.1 \times Bitrate \times Buffersize \times Cachemisses$$



UNICORN: Our Causal AI for Systems Method



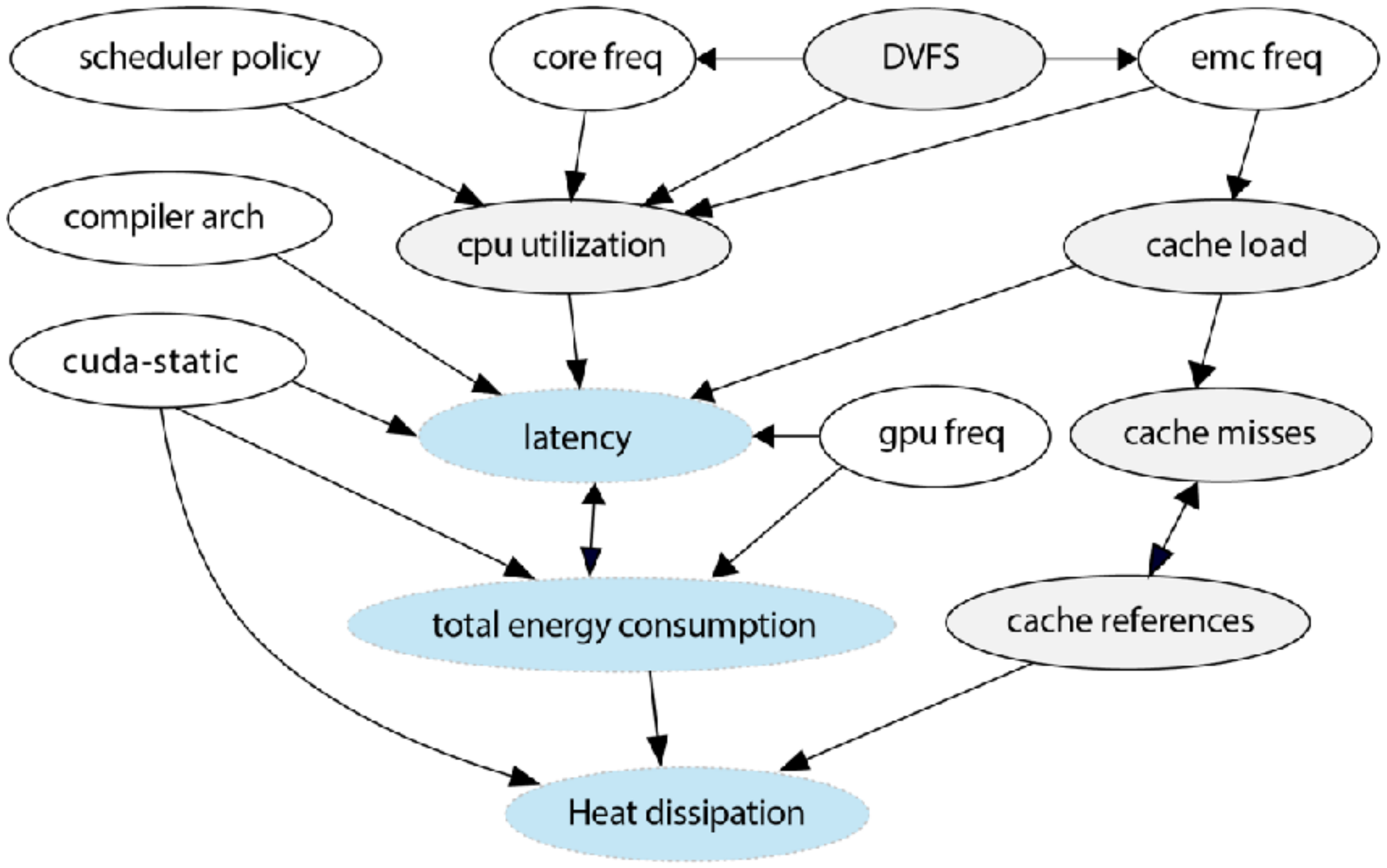
Causal Debugging



Extracting Causal Paths from the Causal Model

Problem

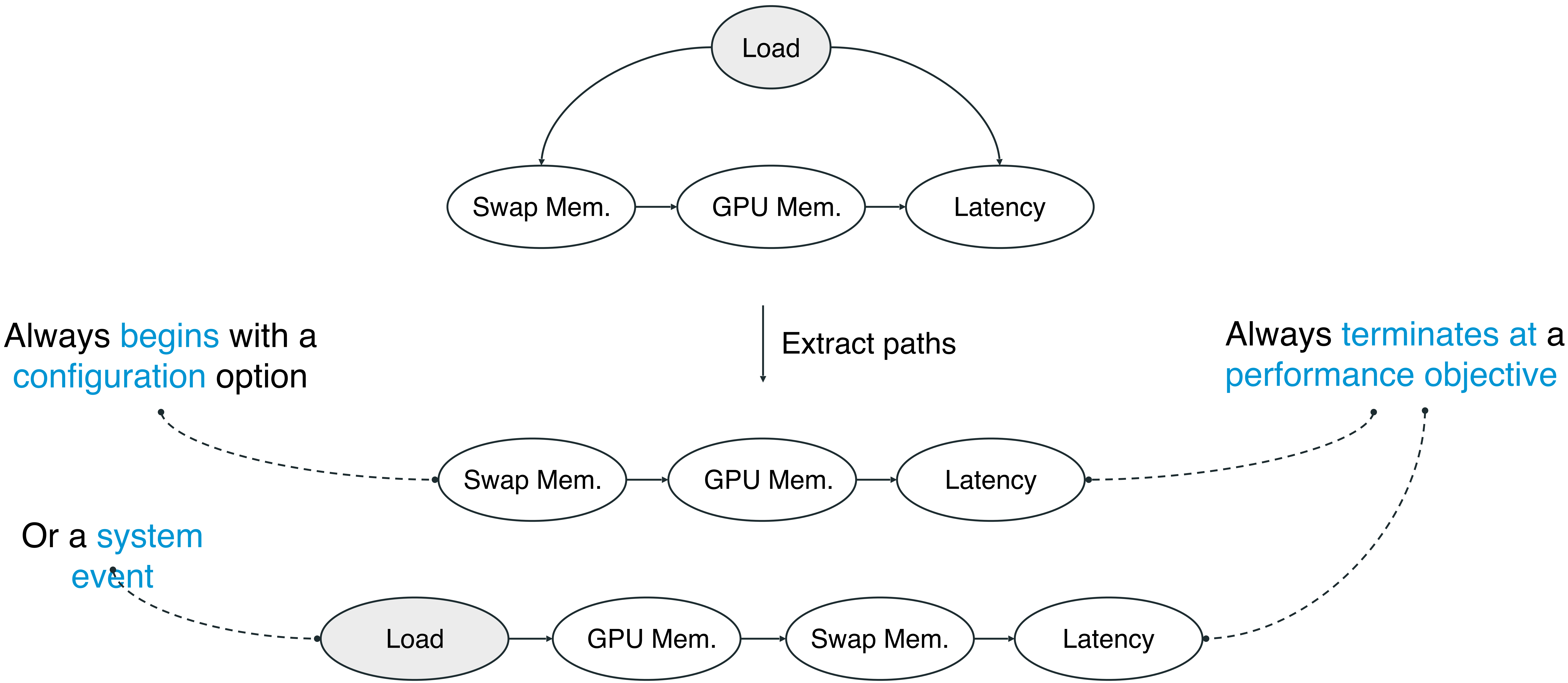
- ✗ In real world cases, this causal graph can be very complex
- ✗ It may be intractable to reason over the entire graph directly



Solution

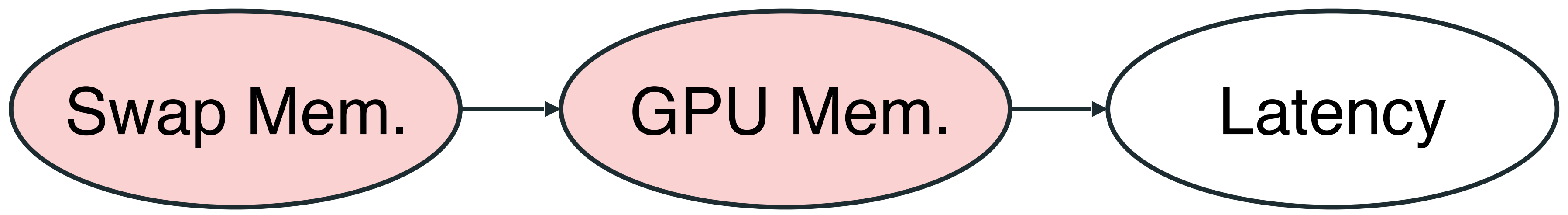
- ✓ Extract paths from the causal graph
- ✓ Rank them based on their Average Causal Effect on latency, etc.
- ✓ Reason over the top K paths

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



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

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.

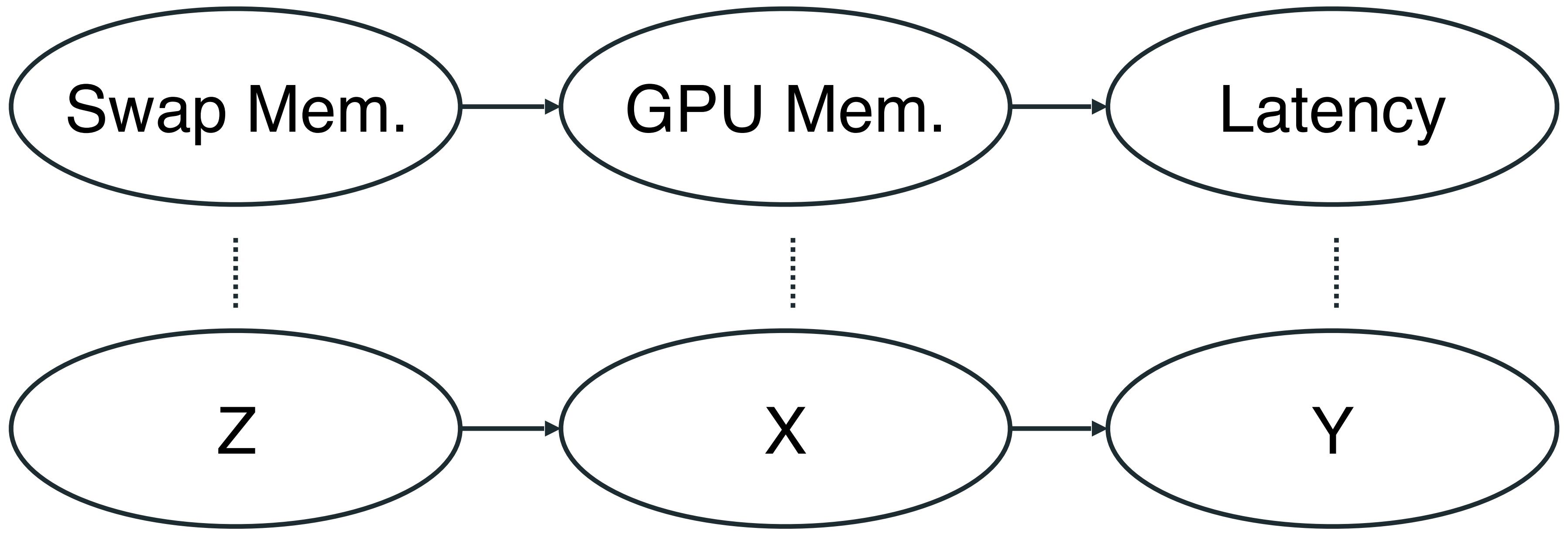
Expected value of GPU Mem. when we artificially intervene by setting Swap to the value a

Ranking Causal Paths from the Causal Model

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

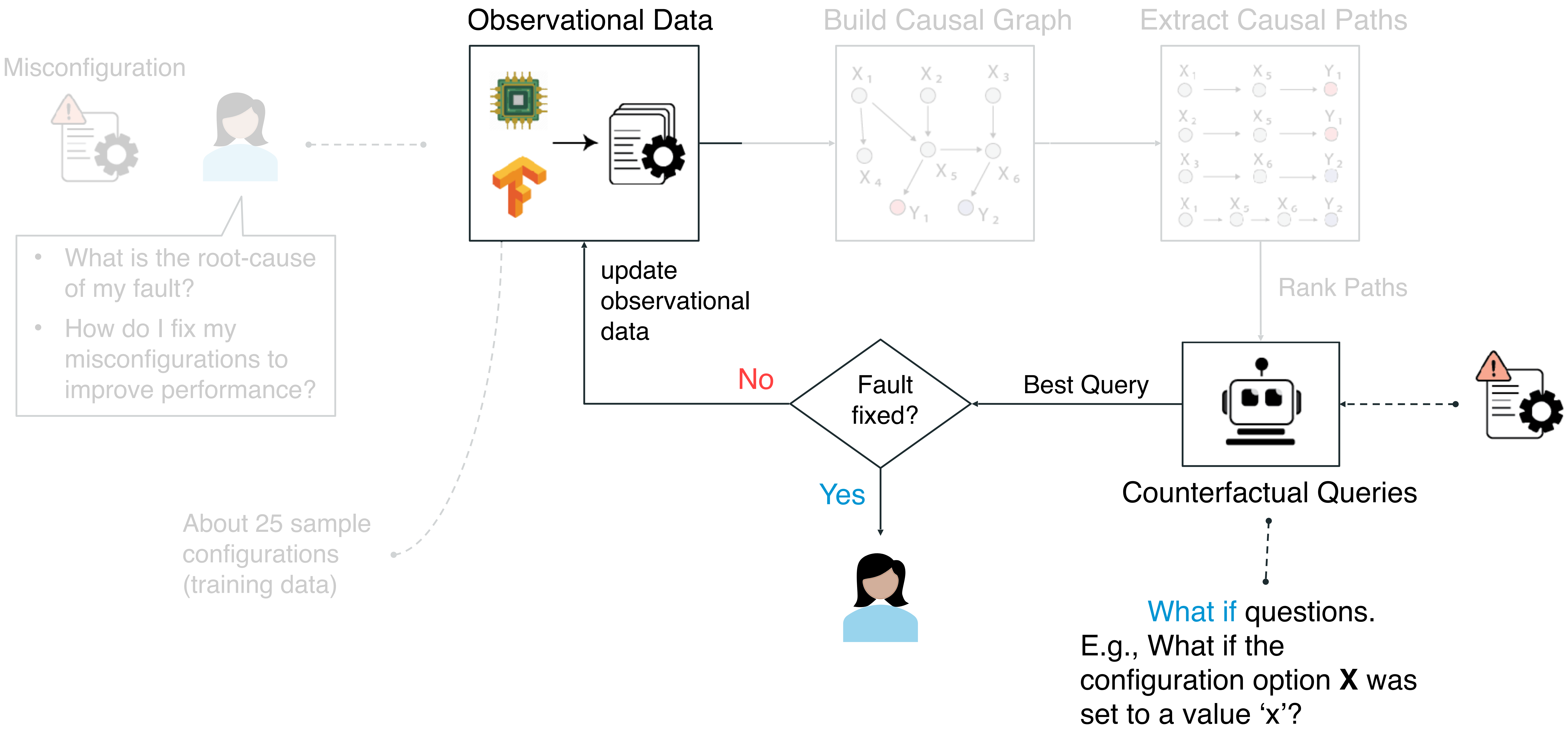
$$PACE(Z, Y) = \frac{1}{2}(ACE(Z,\ X) + ACE(X,\ Y))$$

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"?

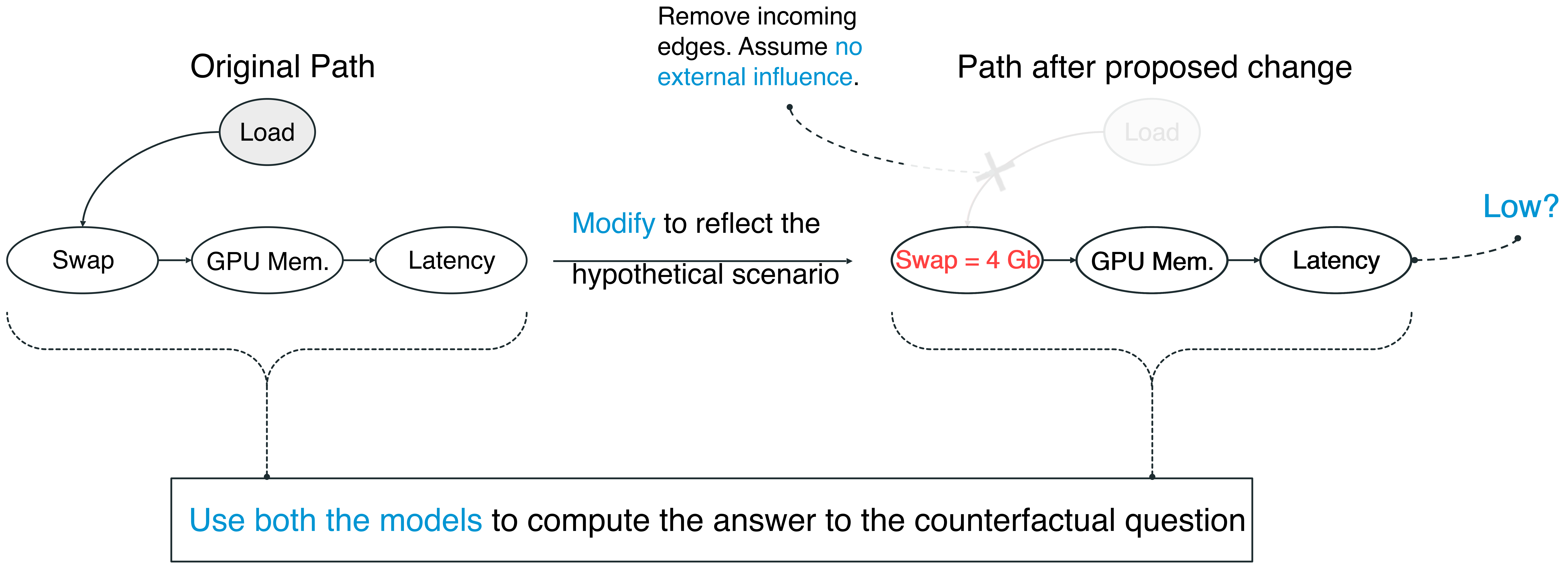
We are interested in the scenario where:

- We hypothetically have low latency;

Conditioned on the following events:

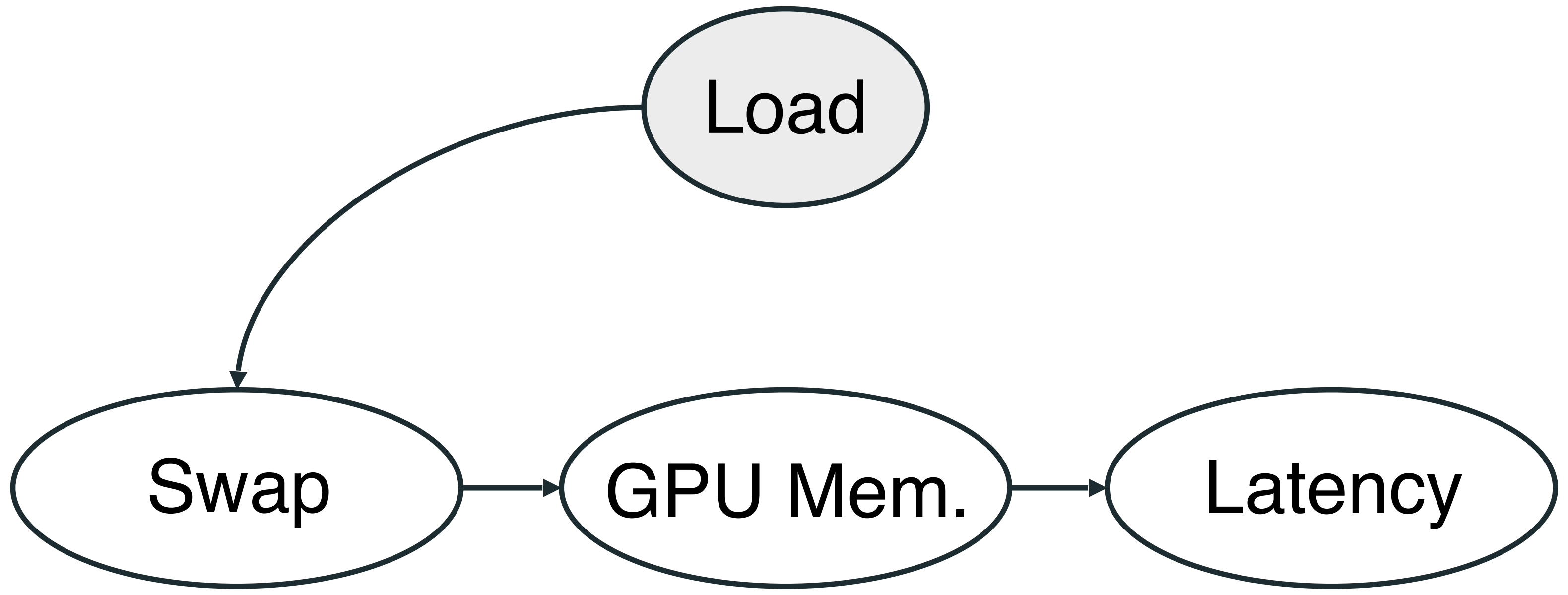
- We hypothetically set the new Swap memory to 4 Gb
- Swap Memory was initially set to 2 Gb
- We observed high latency when Swap was set to 2 Gb
- Everything else remains the same

Diagnosing and Fixing the Faults

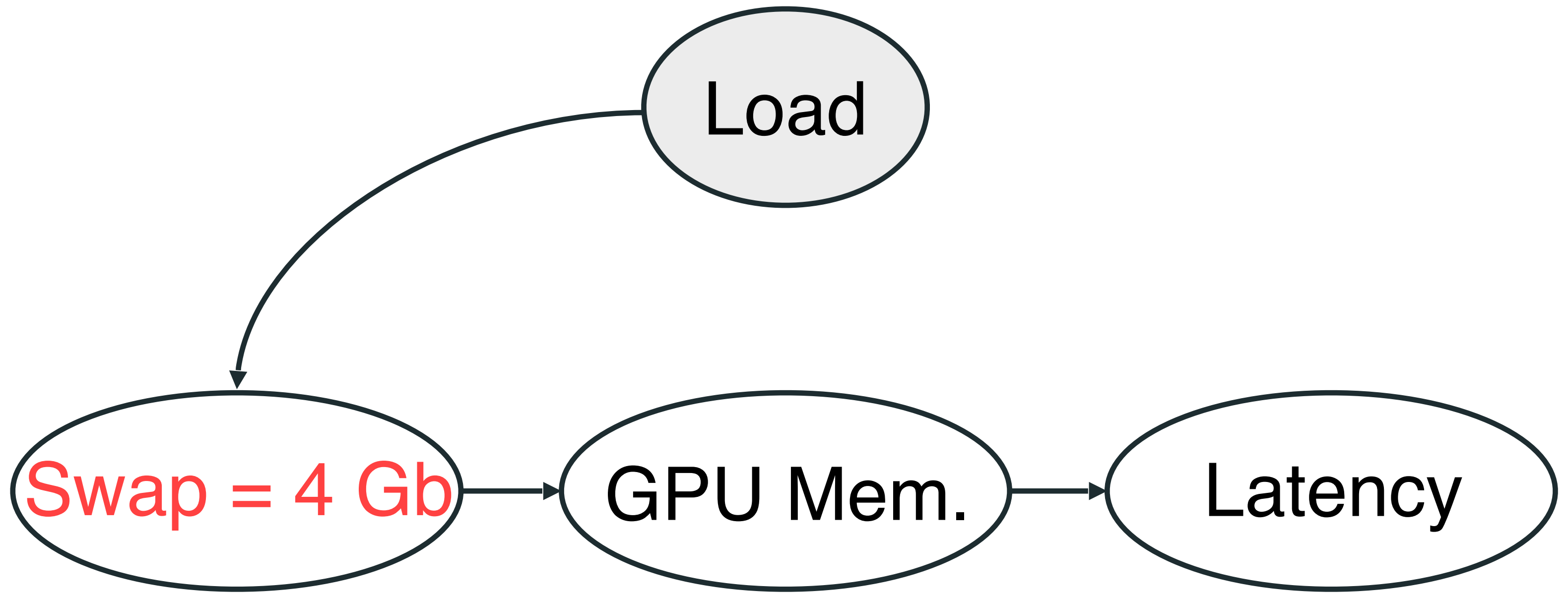


Diagnosing and Fixing the Faults

Original Path



Path after proposed change



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

We expect a low latency

The Swap is now 4 Gb

The Swap was initially 2 Gb

The latency was high

Everything else stays the same

Diagnosing and Fixing the Faults

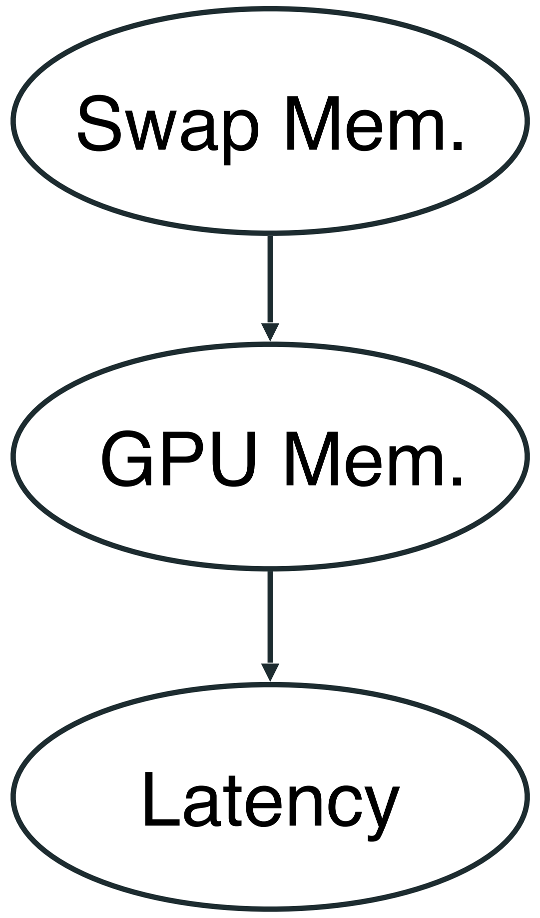
$$\text{Potential} = \underbrace{P\left(\hat{outcome} = good \;\middle|\; change, \quad outcome_{\neg change} = bad, \quad \neg change, \; U \right)}_{\text{Probability that the outcome is good after a change, conditioned on the past}}$$

$$\text{Control} = \underbrace{P\left(\hat{outcome} = bad \;\middle|\; \neg change, \; U \right)}_{\text{Probability that the outcome was bad before the change}}$$

$$\boxed{\text{Individual Treatment Effect} = \underbrace{\text{Potential} - \text{Outcome}}_{\text{If this difference is large, then our change is useful}}}$$

Diagnosing and Fixing the Faults

Top K paths



...

Set every configuration option in the path to all permitted values

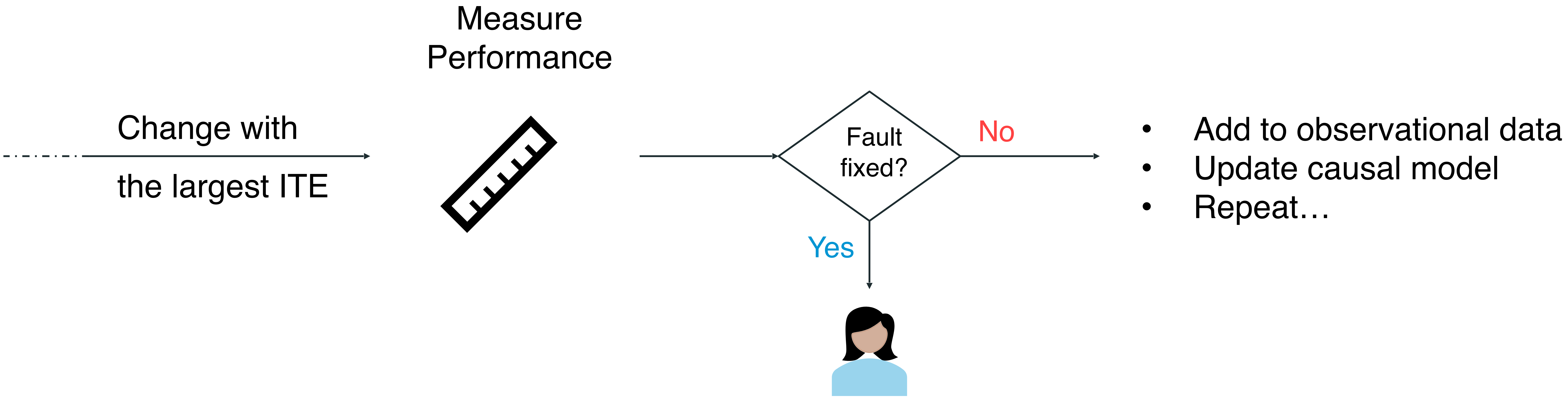
Enumerate all possible changes

$ITE(change)$

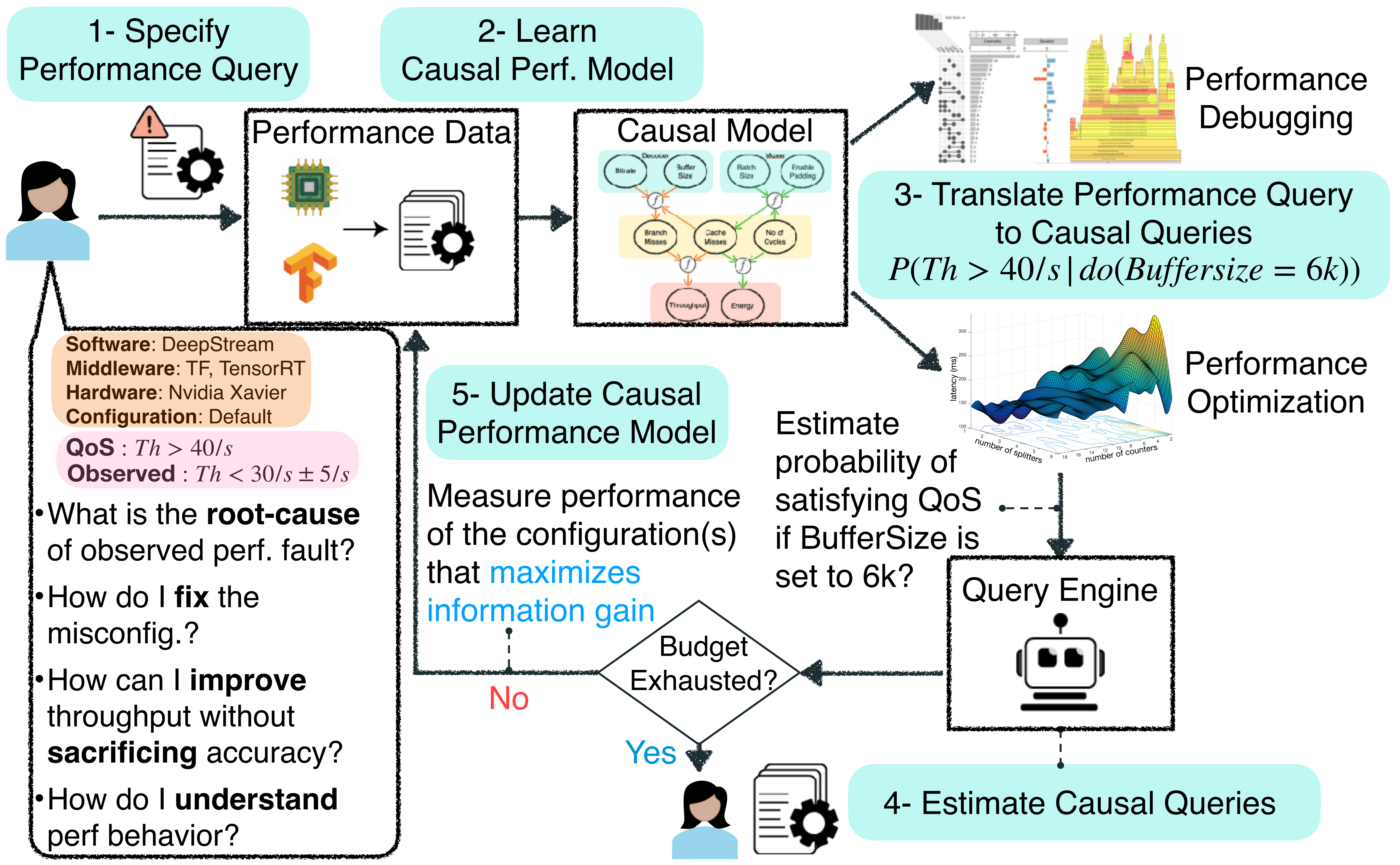
Change with the largest ITE

Inferred from observed data. This is very cheap.

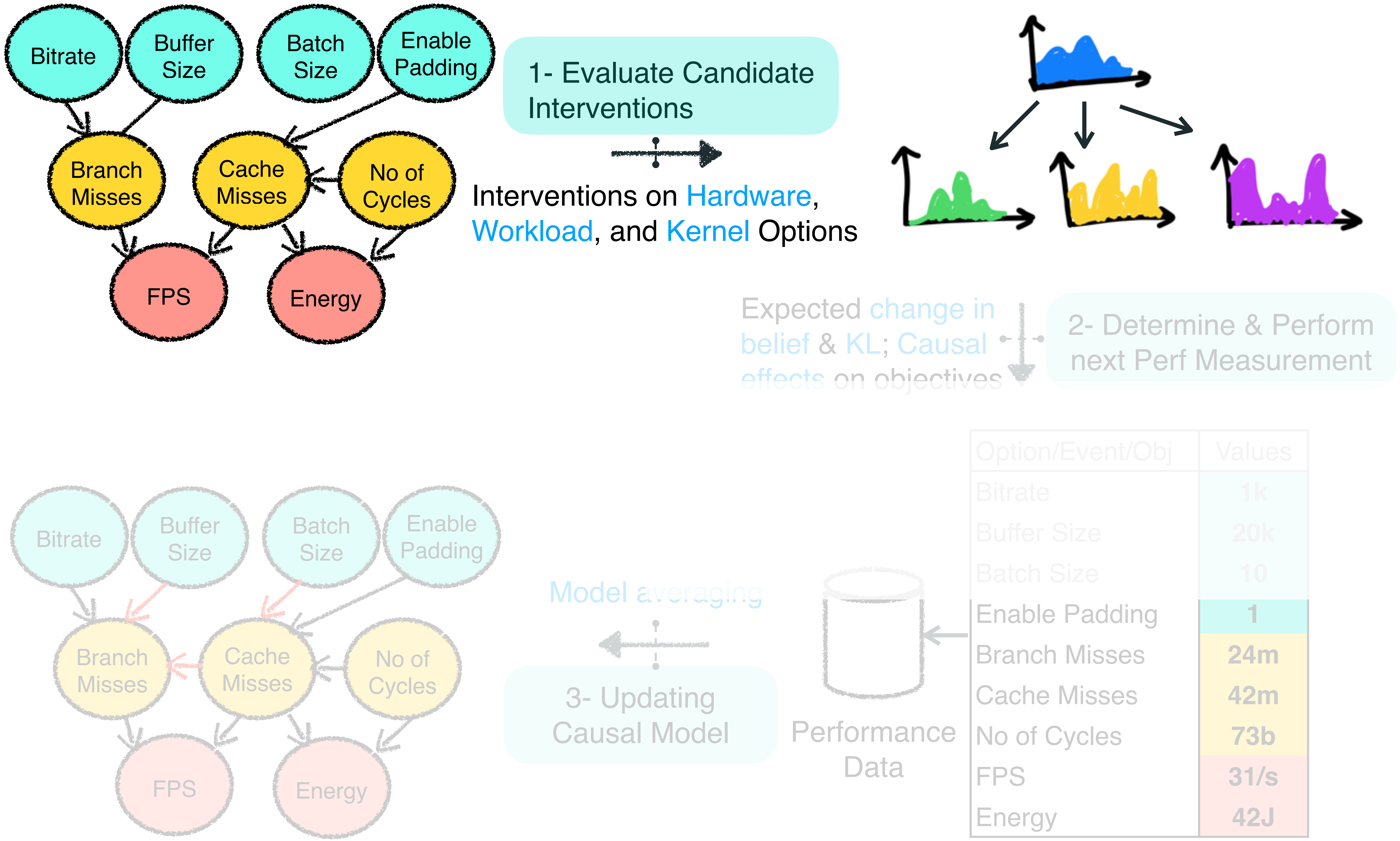
Diagnosing and Fixing the Faults



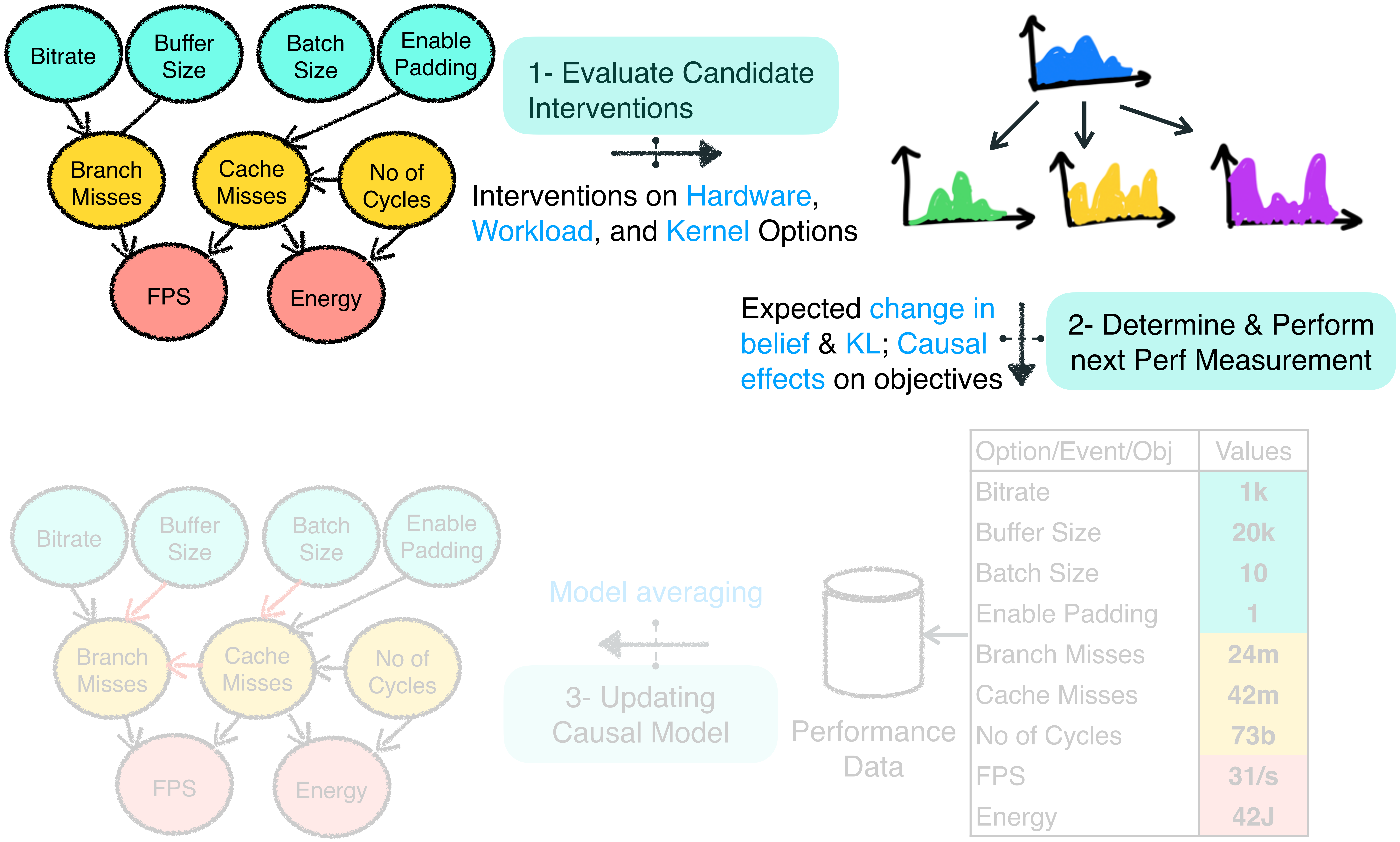
UNICORN: Our Causal AI for Systems Method



Active Learning for Updating Causal Performance Model

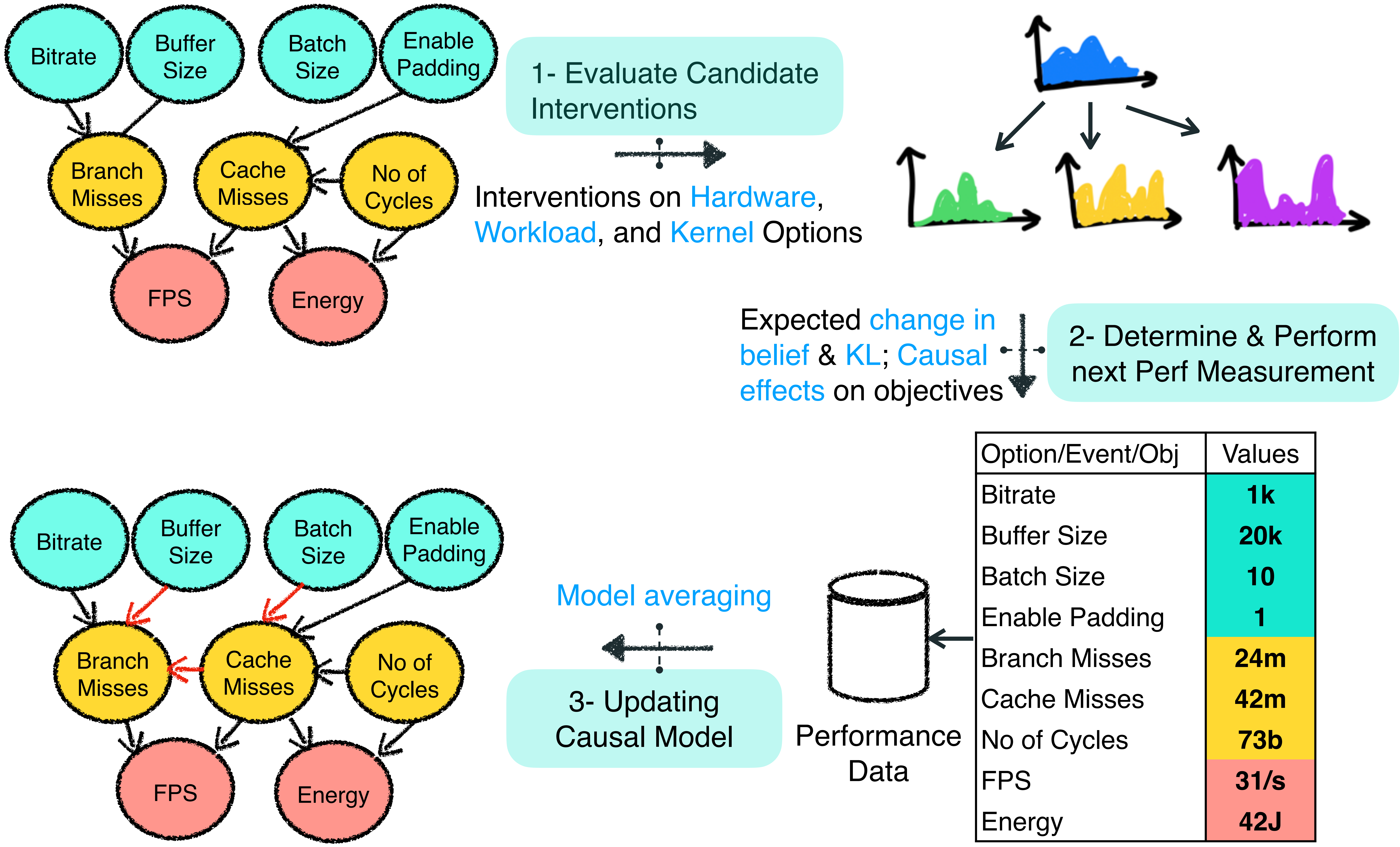


Active Learning for Updating Causal Performance Model



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



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.

EuroSys 2022

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

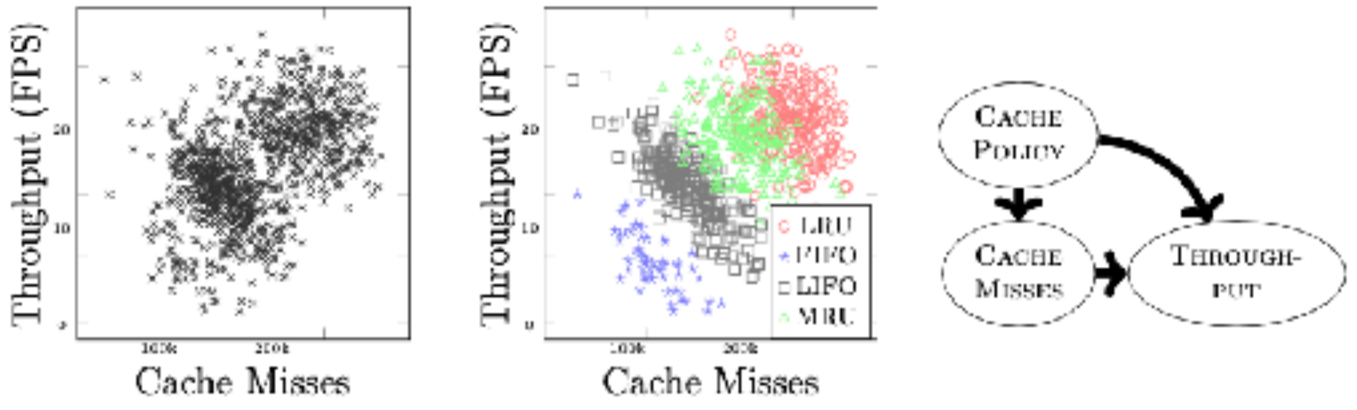
Mohammad Ali Javidian
Purdue University
mjavidia@purdue.edu

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,



(a) (b) (c)