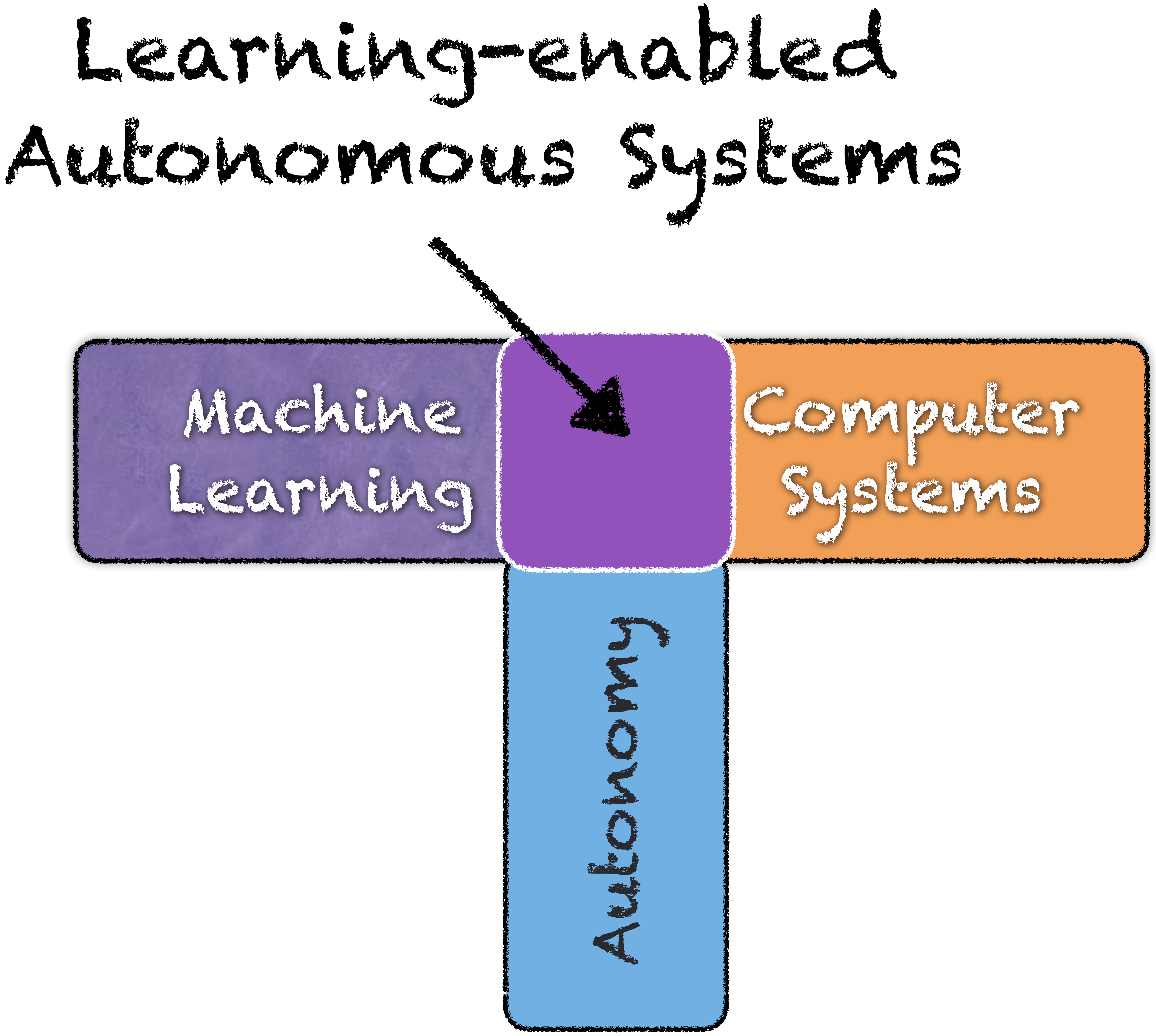
Performance Tradeoff in Machine Learning Systems

A journey from *optimization* to *transfer learning* all the way to *counterfactual causal inference*

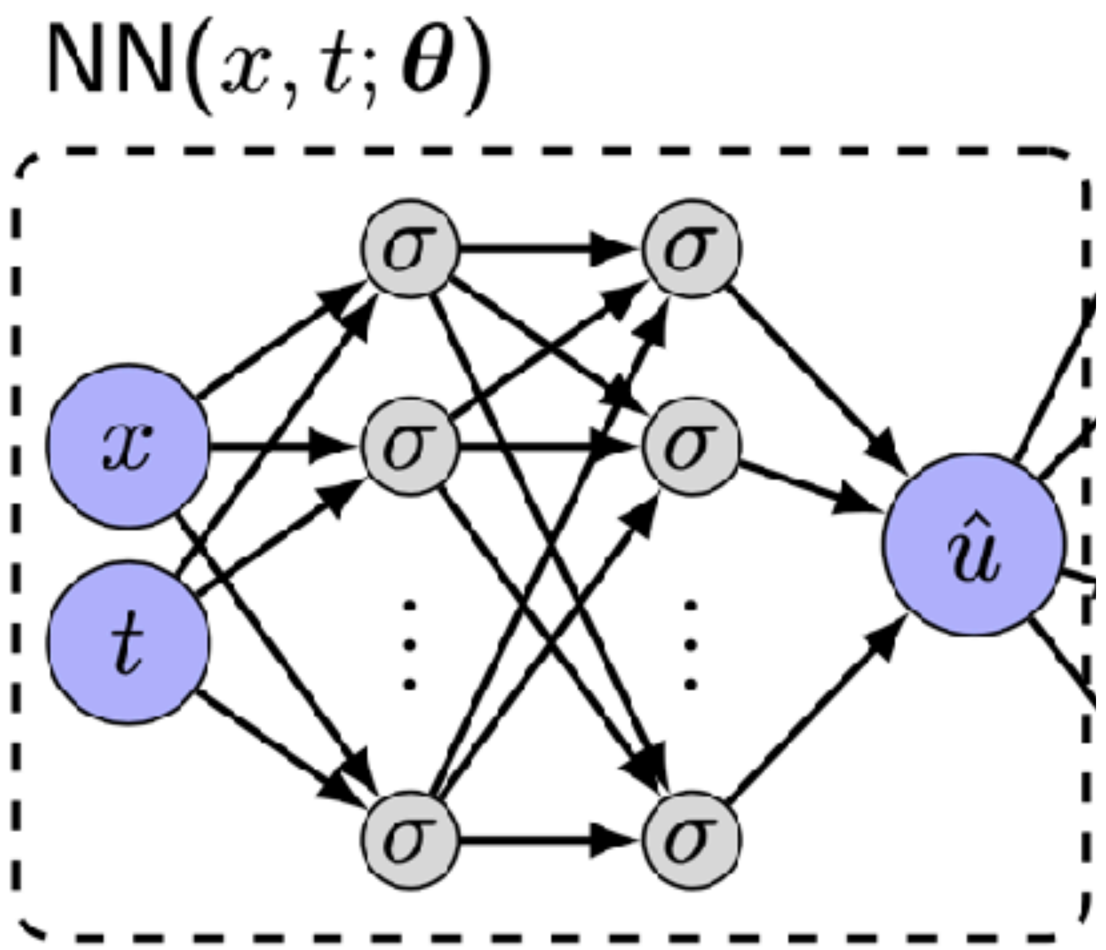
Pooyan Jamshidi
UofSC

Artificial Intelligence and Systems Laboratory
(AISys Lab)



https://pooyanjamshidi.github.io/AISys/

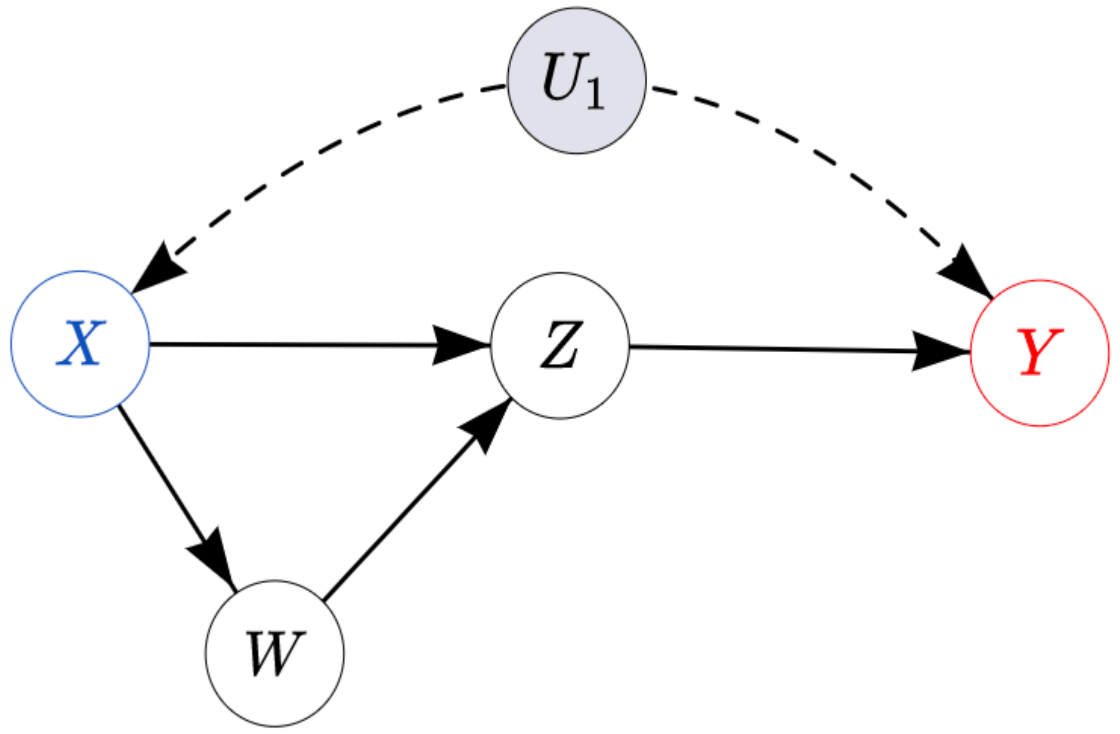
Research Directions at AISys



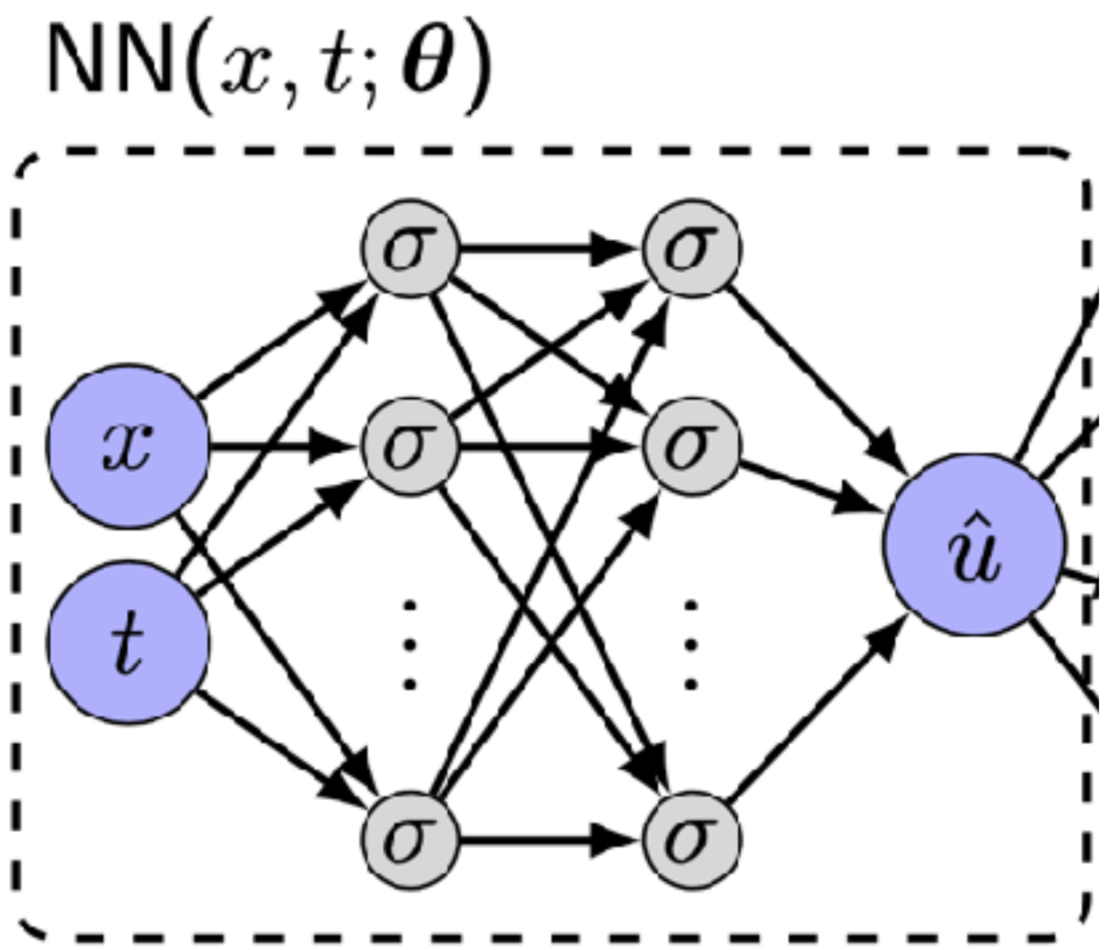
Well-known Physics
Big Data

Theory:
- Transfer Learning
- Causal Invariances
- Structure Learning
- Concept Learning
- Physics-Informed

Applications:
- Systems
- Autonomy
- Robotics



Causal AI



Limited known Physics
Small Data

Thanks NASA, NSF, and Google for supporting our research 😍

Team effort

Rahul Krishna
Columbia

Shahriar Iqbal
UofSC

M. A. Javidian
Purdue

Baishakhi Ray
Columbia

Christian Kästner
CMU

Miguel Velez
CMU

Norbert Siegmund
Leipzig

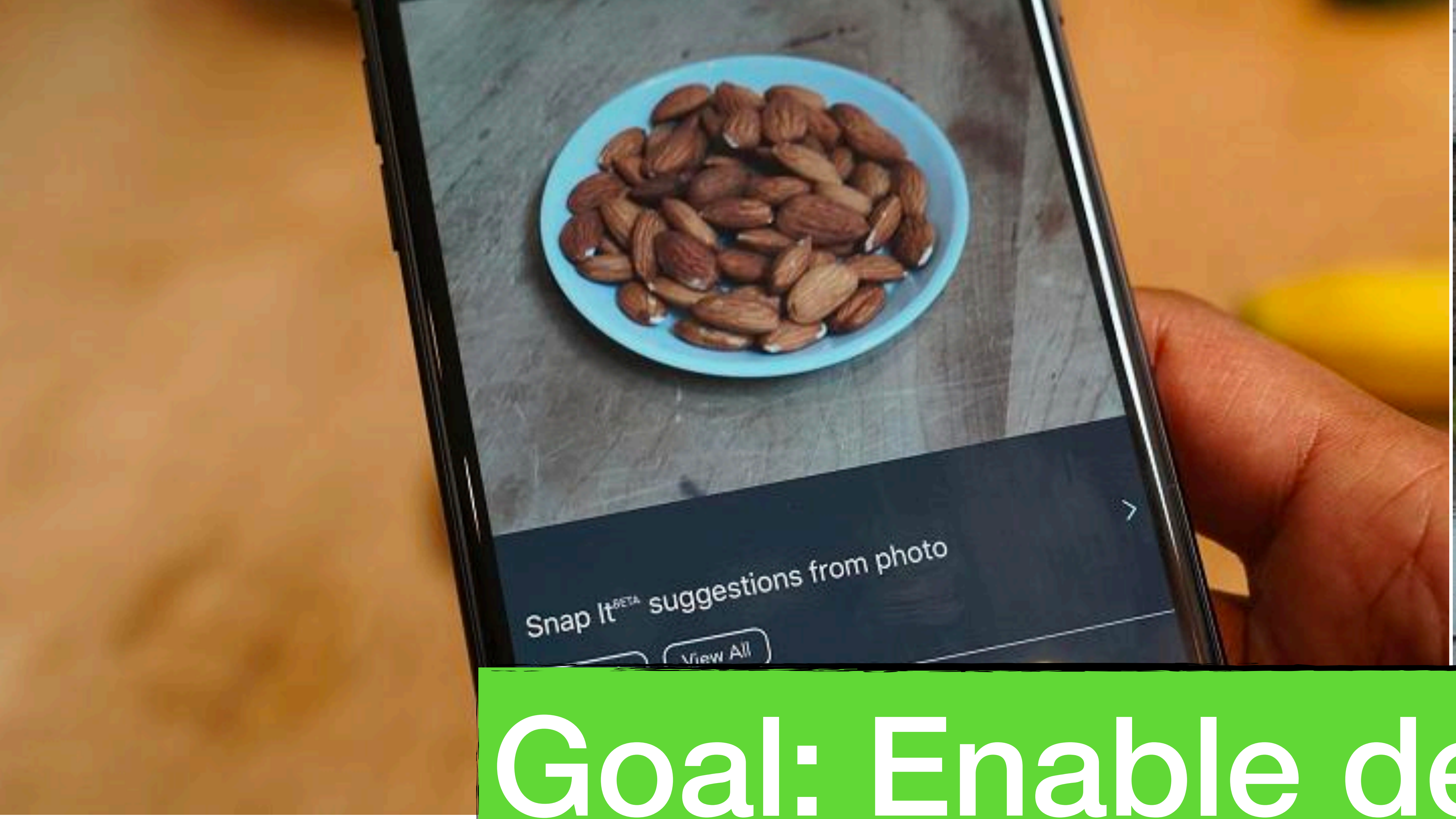
Sven Apel
Saarland

Lars Kotthoff
Wyoming

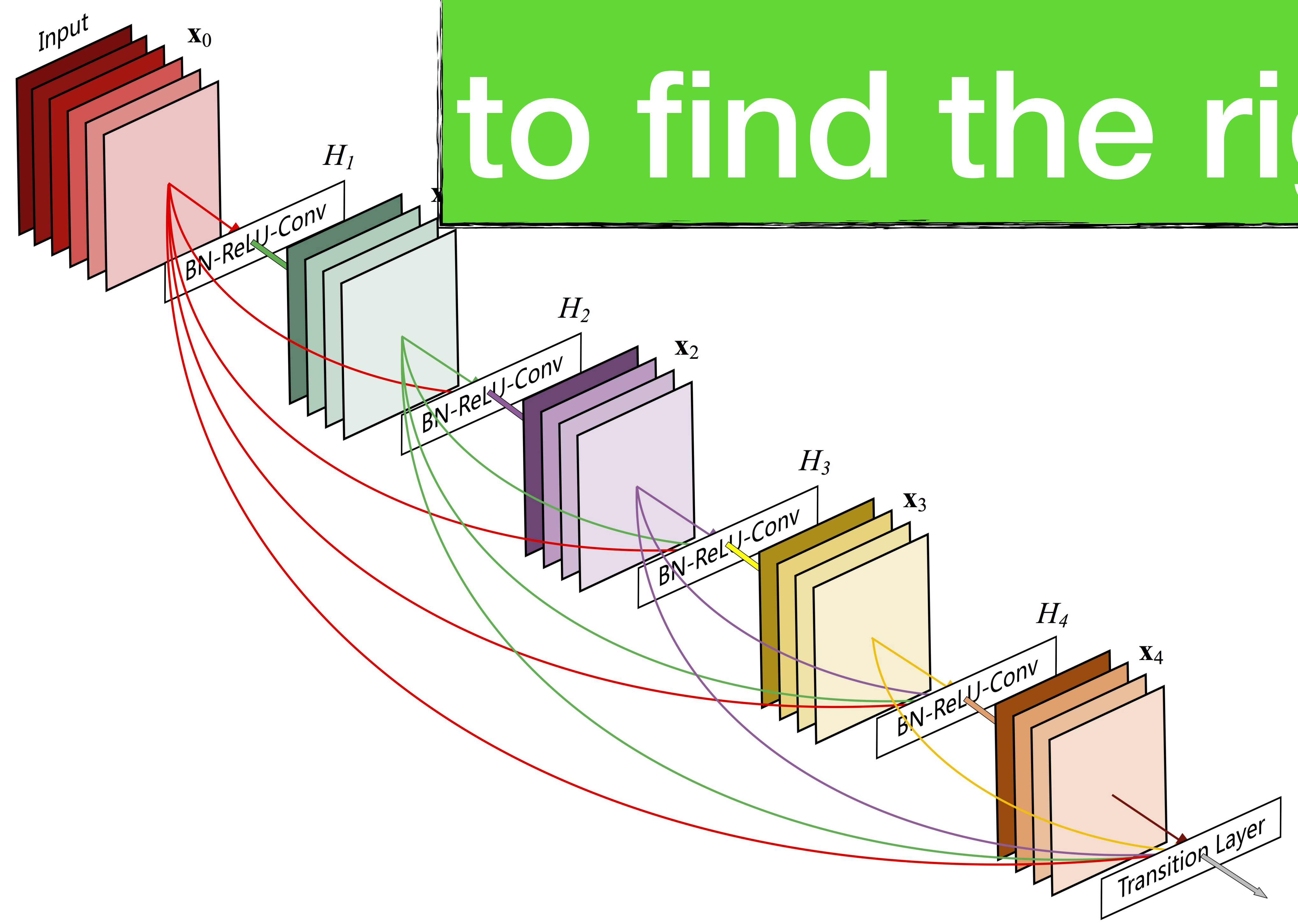
Marco Valtorta
UofSC

Vivek Nair
Facebook

Tim Menzies
NCSU

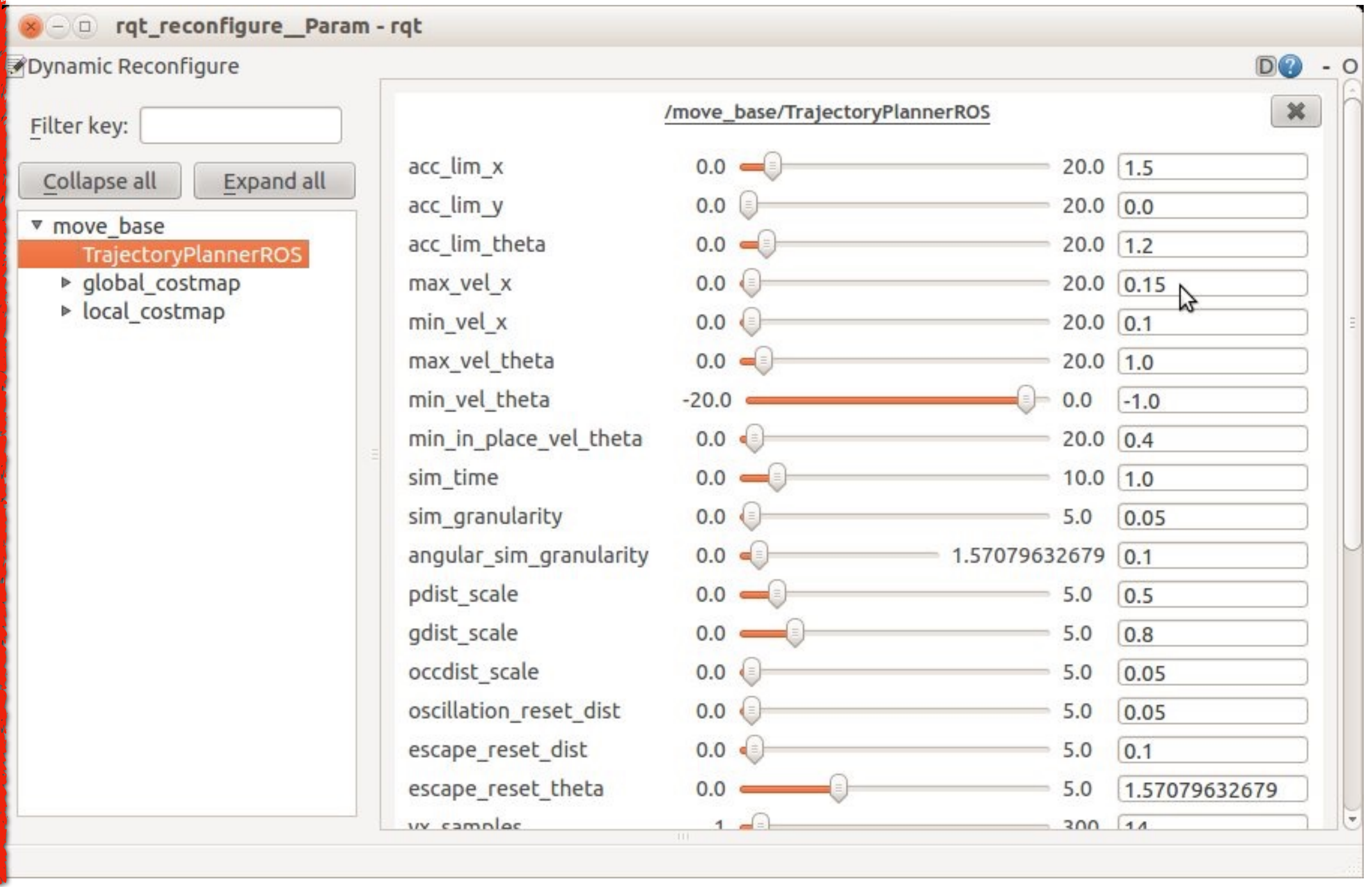


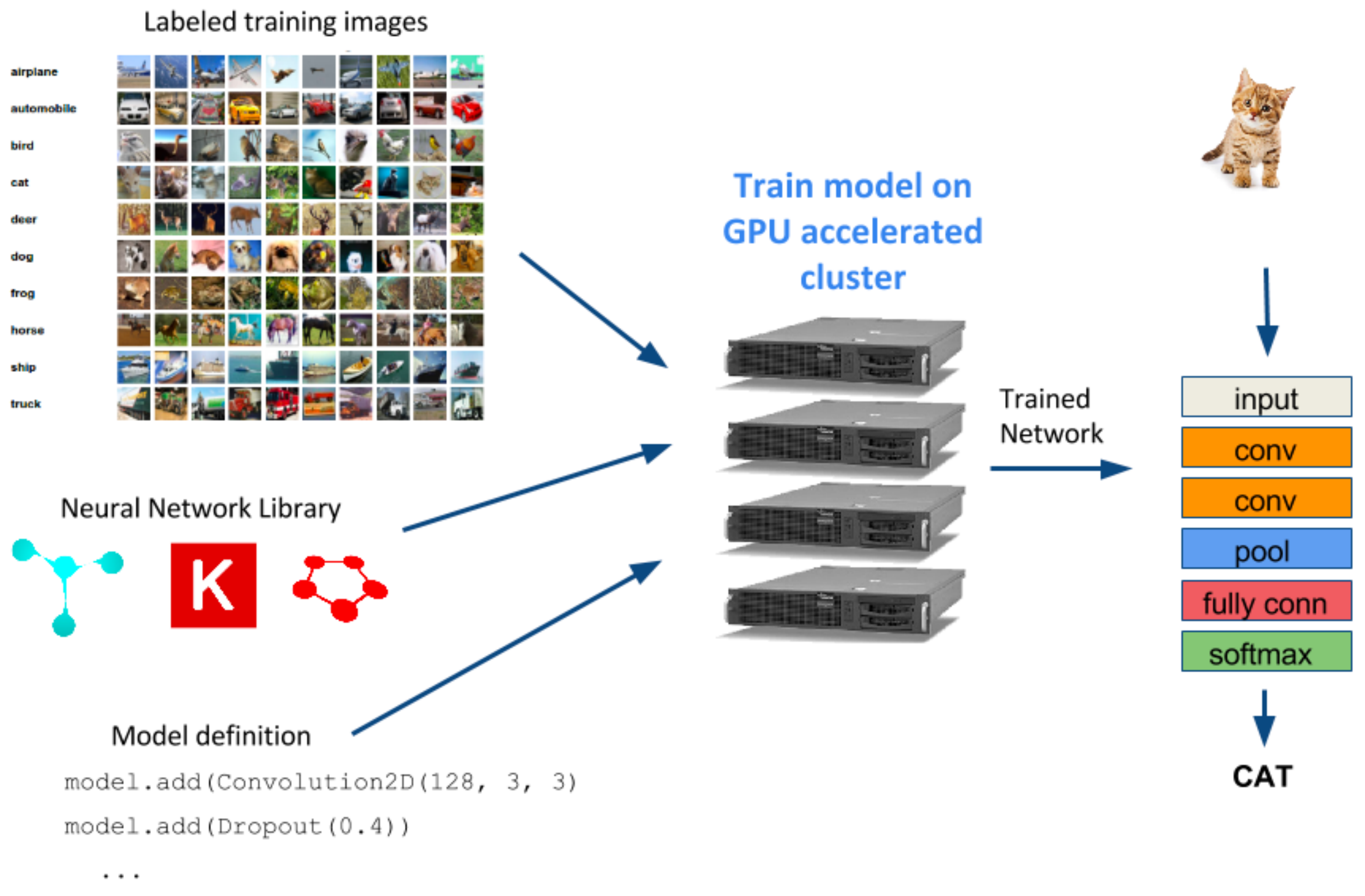
Goal: Enable developers/users to find the right quality tradeoff



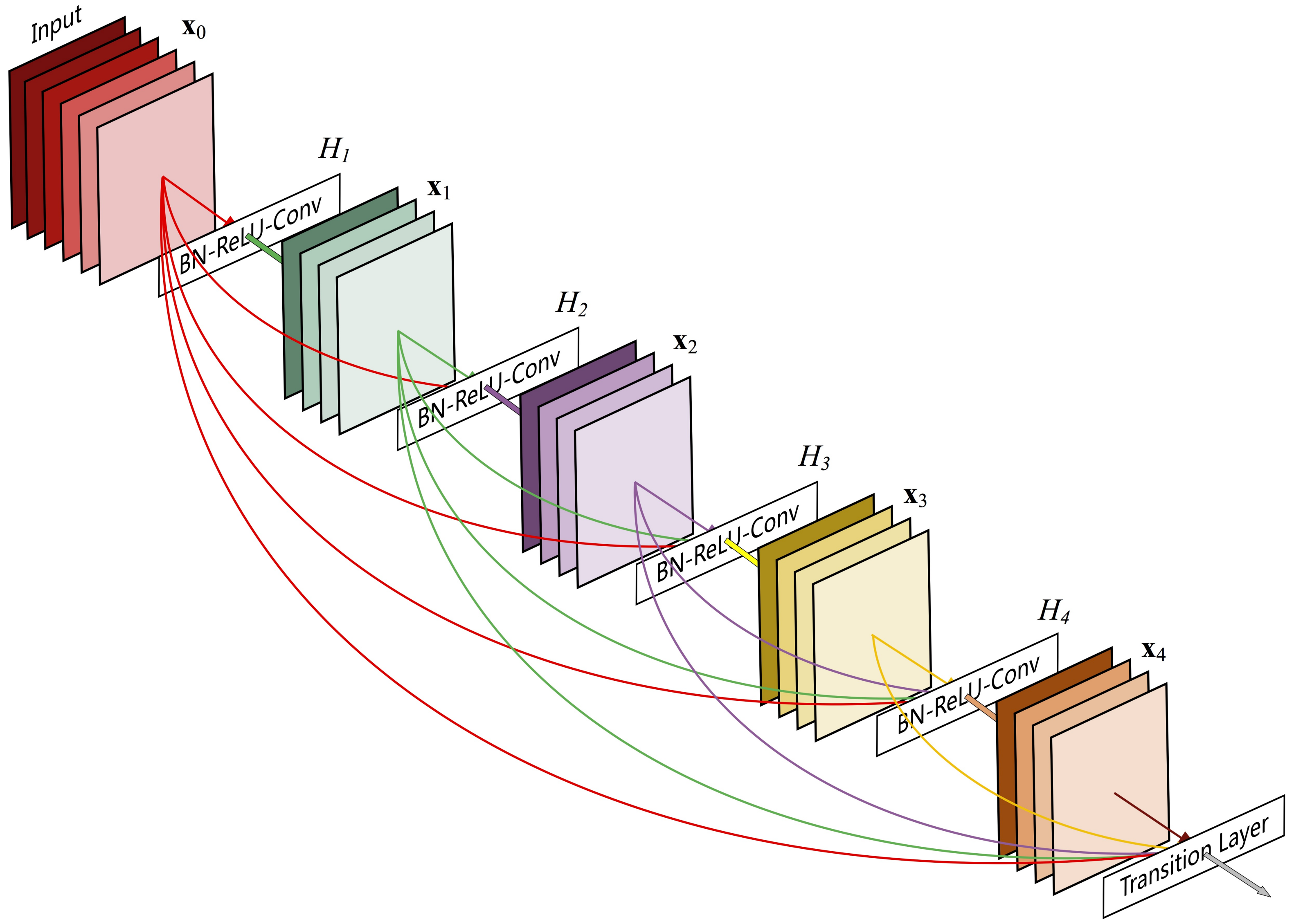
Today's most popular systems are ~~configurable~~ built configurable

```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
```



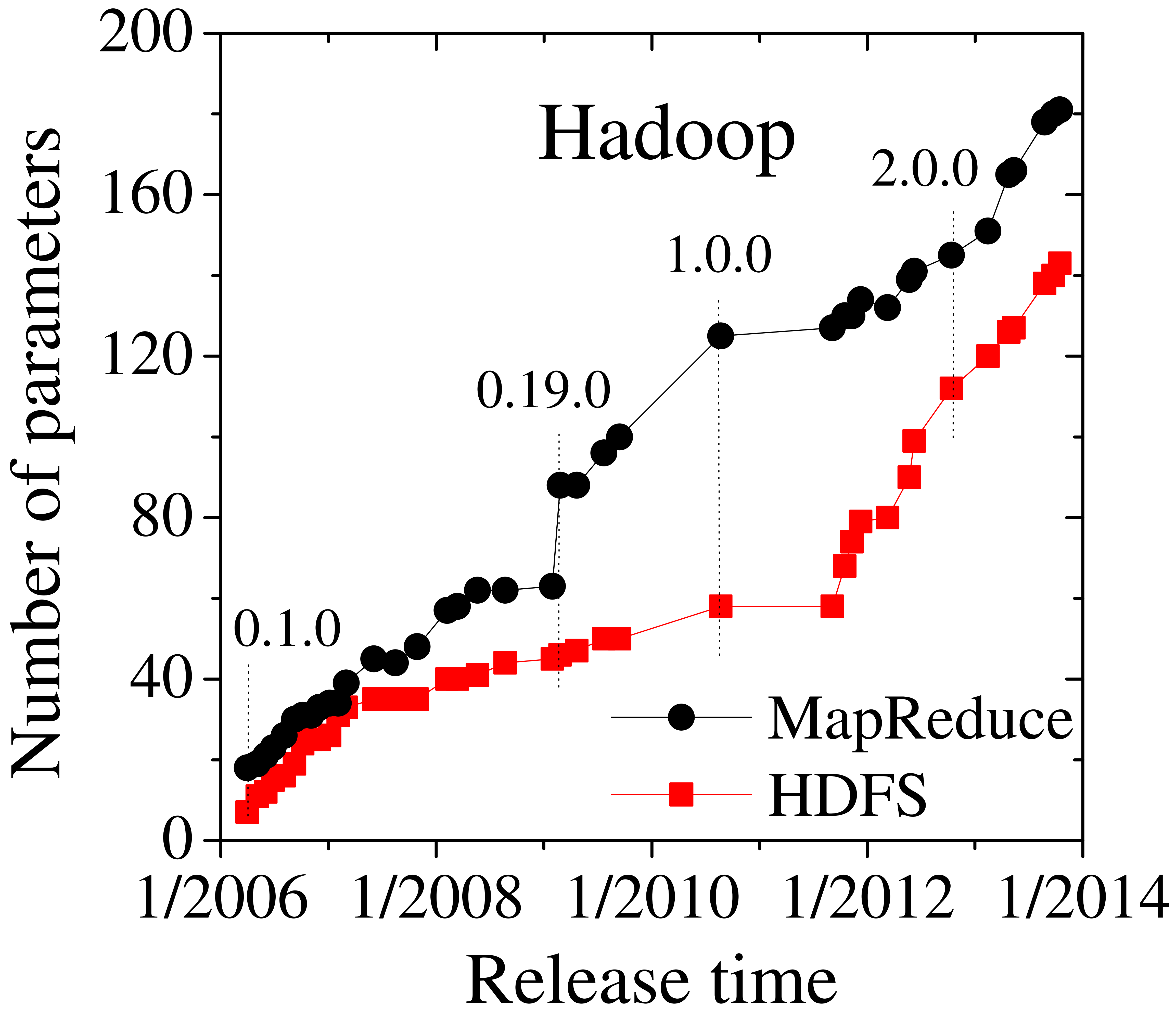


hadoop



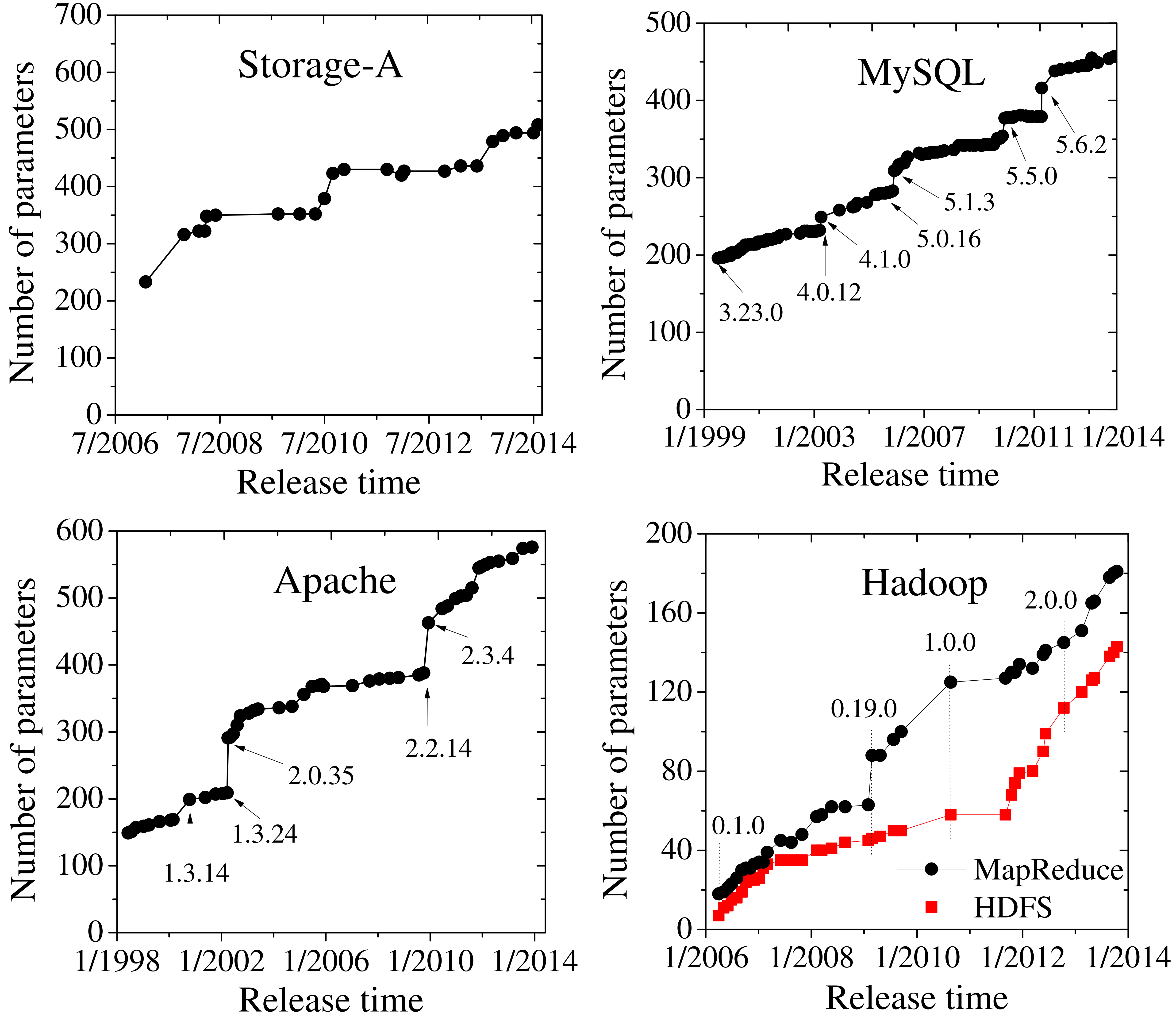
```yaml
drpc.port: 3772
drpc.worker.threads: 64
drpc.max_buffer_size: 1048576
drpc.queue.size: 128
drpc.invocations.port: 3773
drpc.invocations.threads: 64
drpc.request.timeout.secs: 600
drpc.childopts: "-Xmx768m"
drpc.http.port: 3774
drpc.https.port: -1
drpc.https.keystore.password: ""
drpc.https.keystore.type: "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]

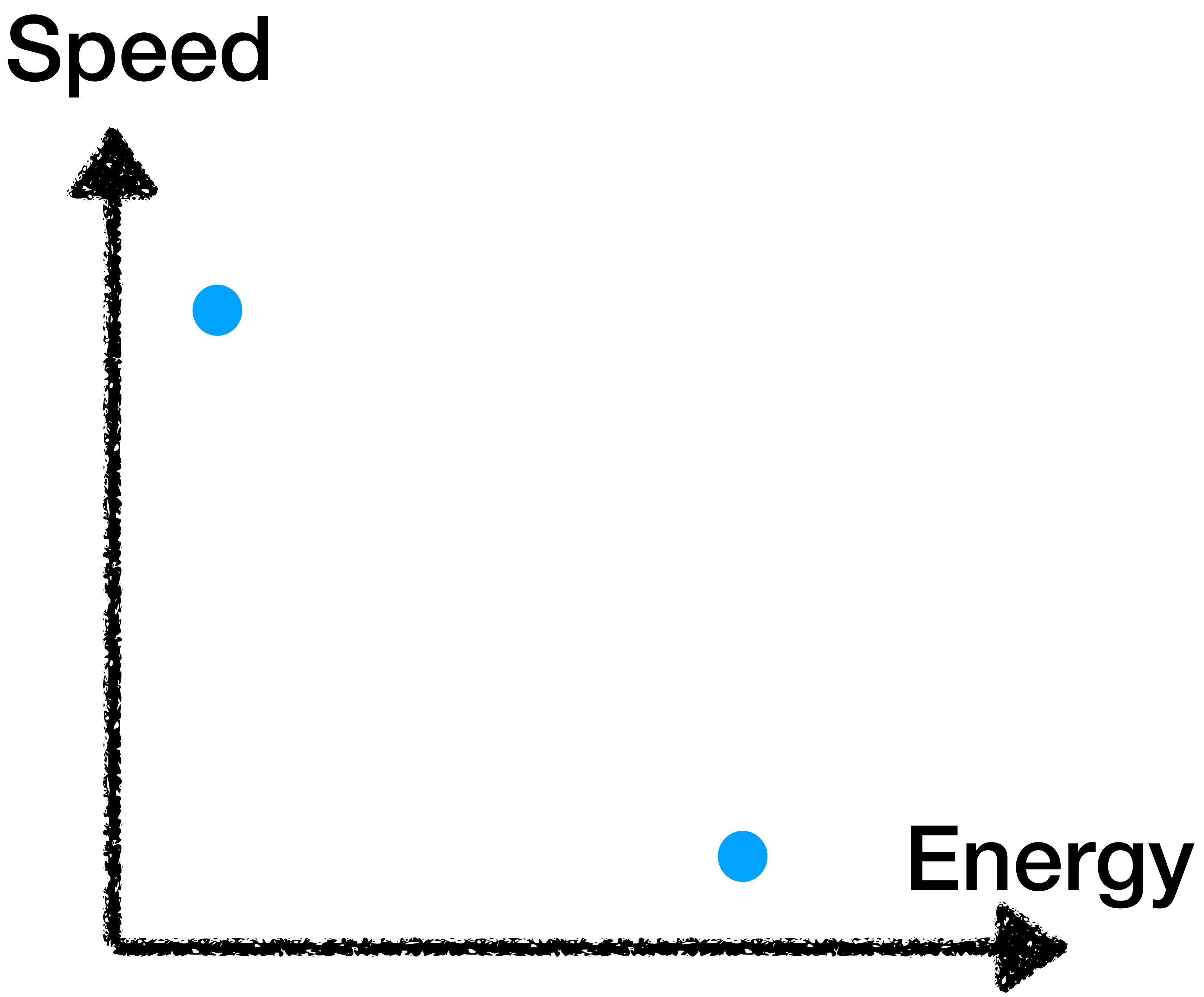
Empirical observations confirm that systems are becoming increasingly configurable



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

Configurations determine the performance behavior

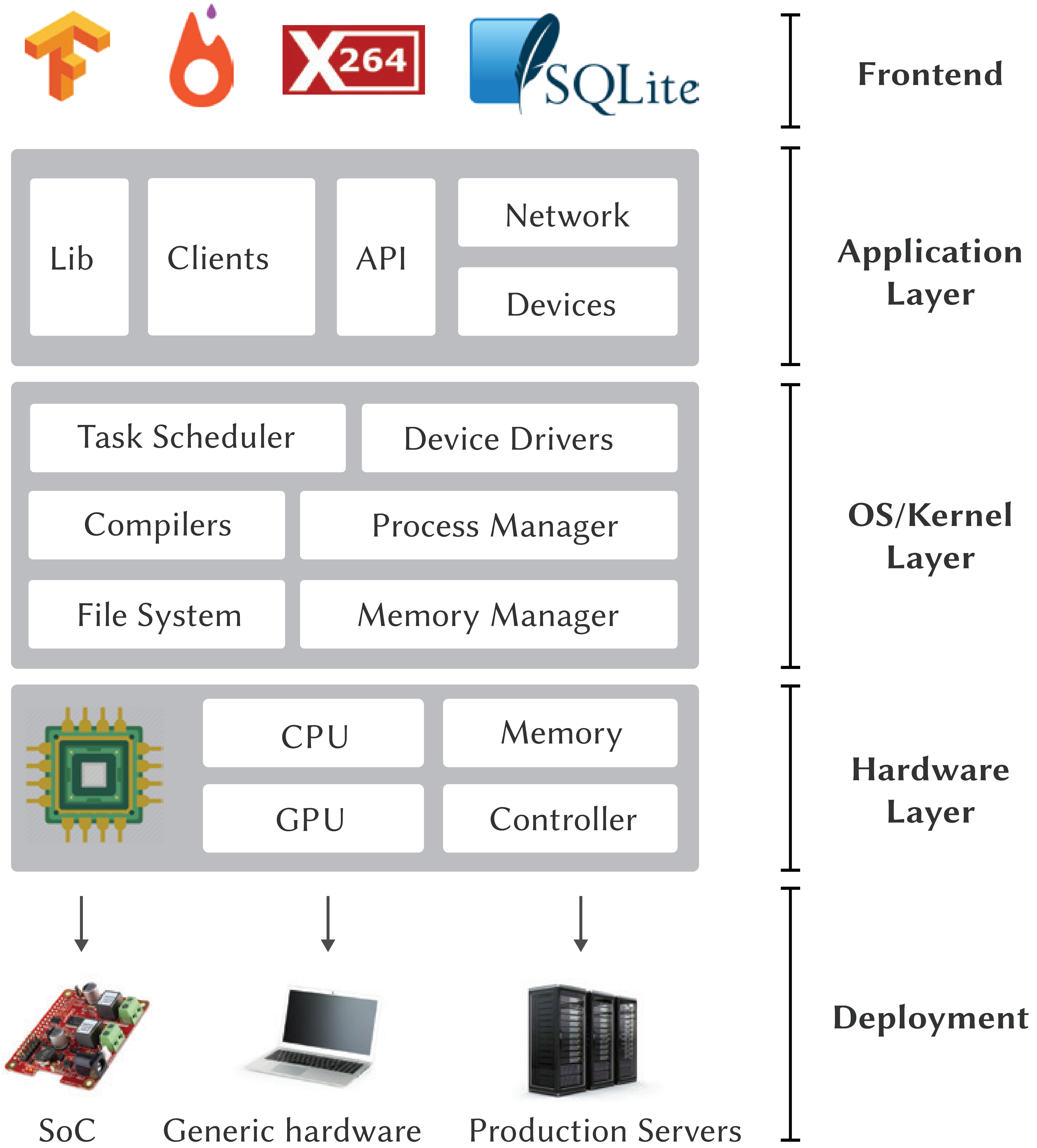
```
void Parrot_setenv(. . . name,. . . value){
#ifdef PARROT_HAS_SETENV
  my_setenv(name, value, 1);
#else
  int name_len=strlen(name);
  int val_len=strlen(value);
  char* envs=glob_env;
  if(envs==NULL){
    return;
  }
  strcpy(envs,name);
  strcpy(envs+name_len,"=");
  strcpy(envs+name_len + 1,value);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



How do we understand performance behavior of real-world highly-configurable systems that scale well…

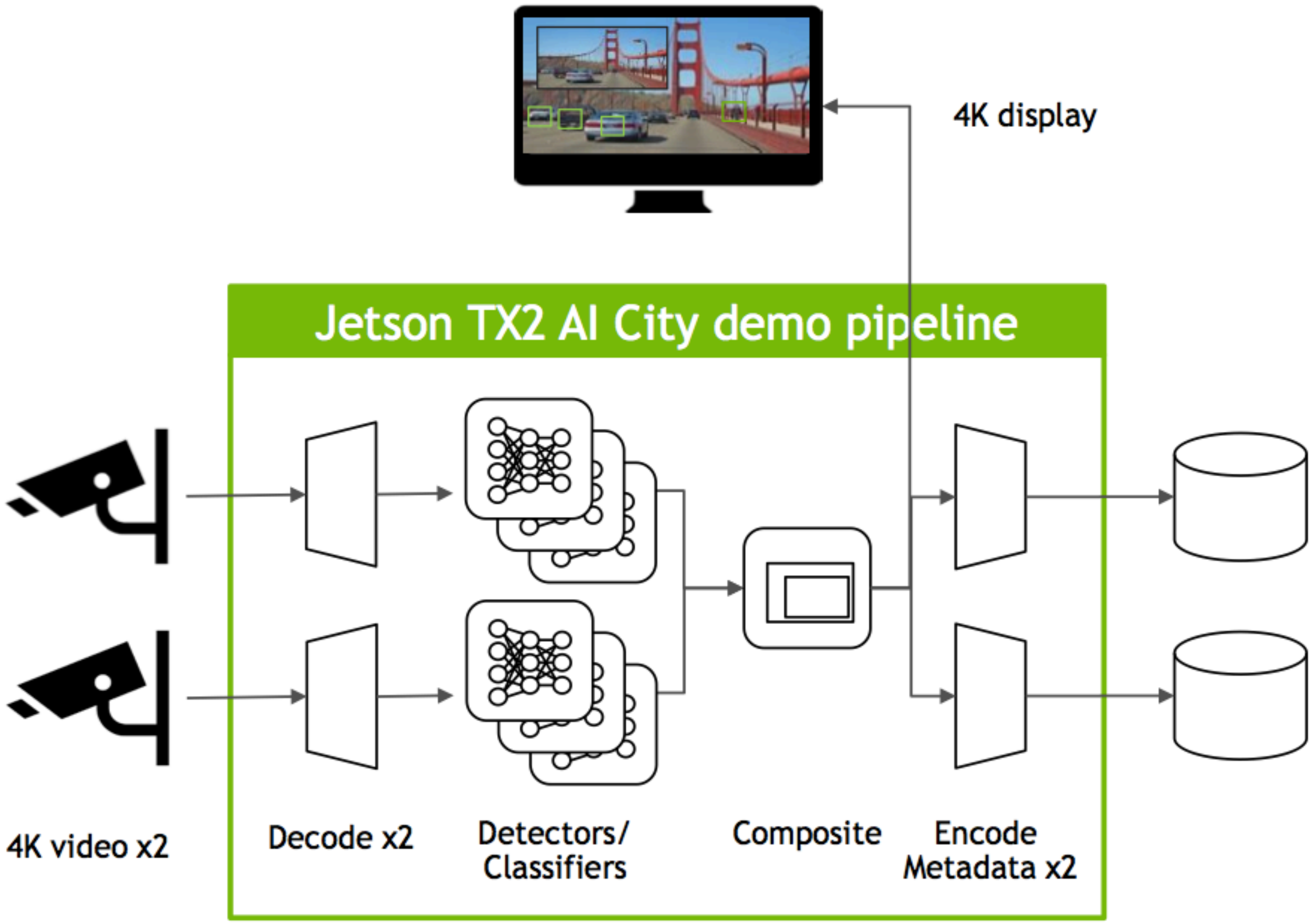
… and enable developers/users to reason about qualities (performance, energy) and to make tradeoff?

Scope: Configuration across stack

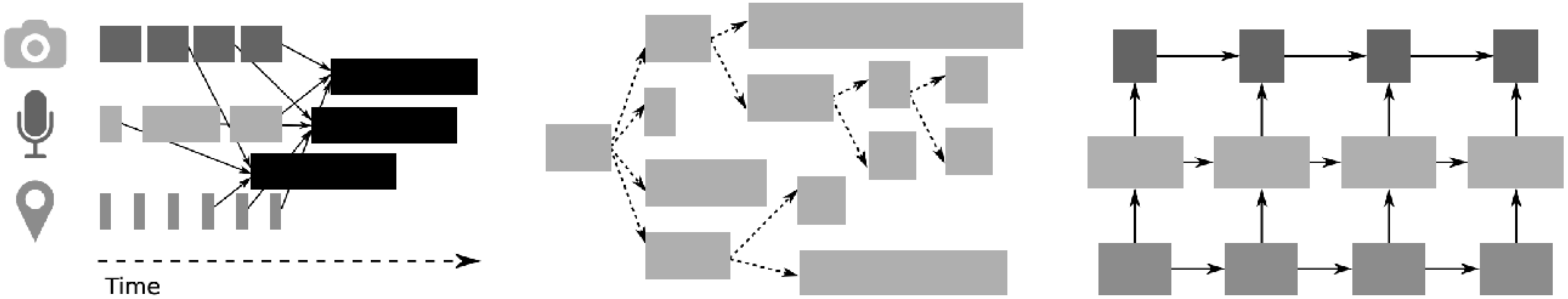


Composed Systems (Single-node)

- Online processing of sensory data
- Neural network models
- Homogeneous tasks



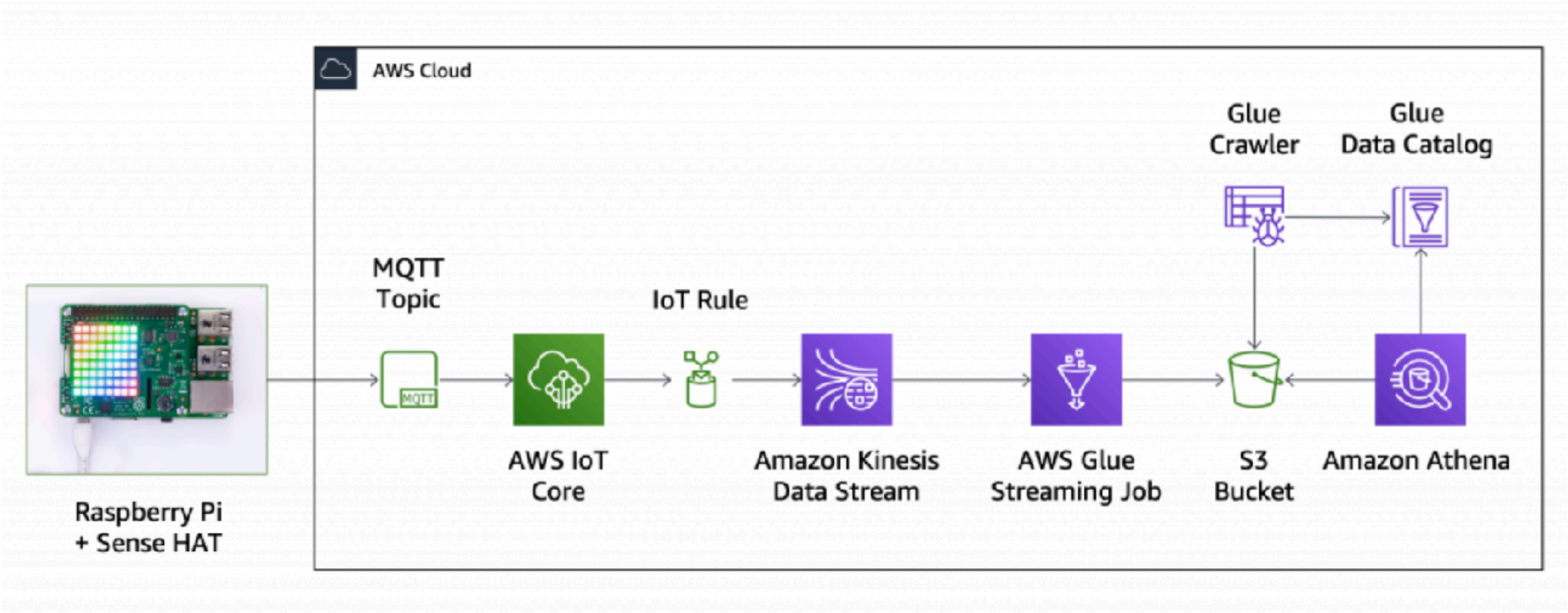
Composed Systems (Multi-node)



(a) multiple sensor inputs (b) Monte Carlo tree search (MCTS) (c) Recurrent Neural Network (RNN)

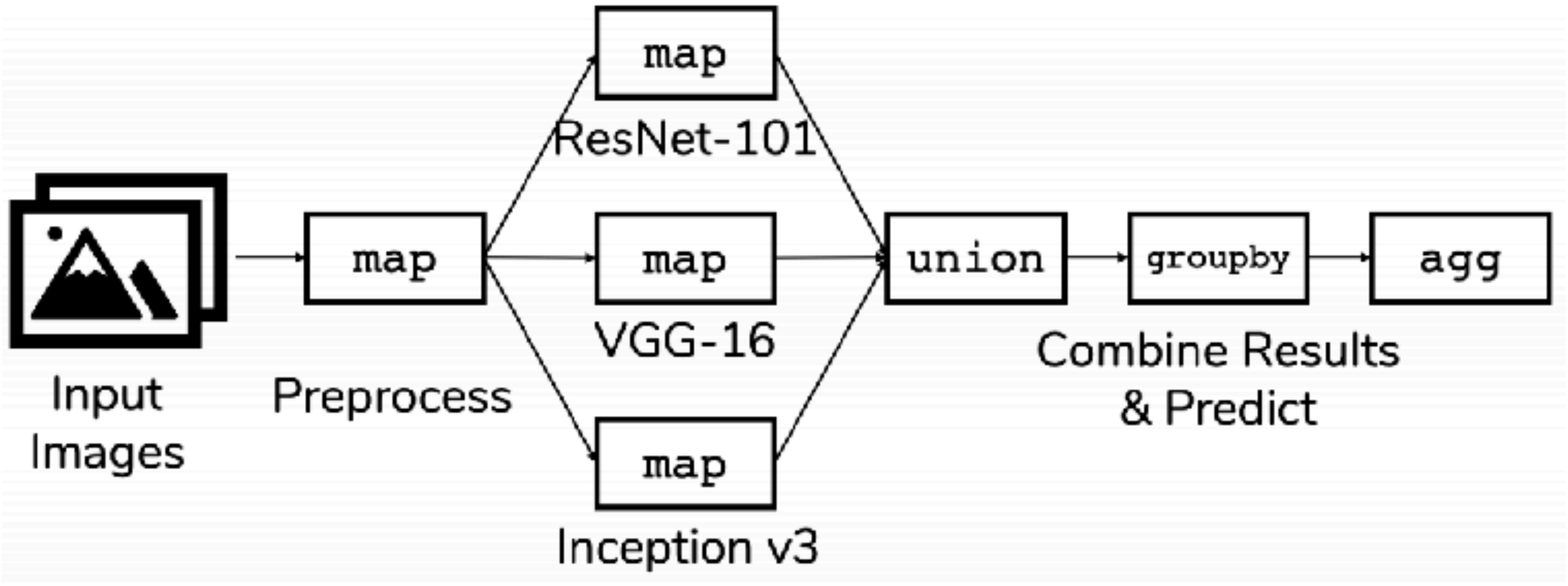
- Online processing of sensory data
- Graph types models
- Heterogeneous tasks

Composed Systems (IoT)



- They are integrated with cloud services and we do not have access to those system, we could configure them to some extent.

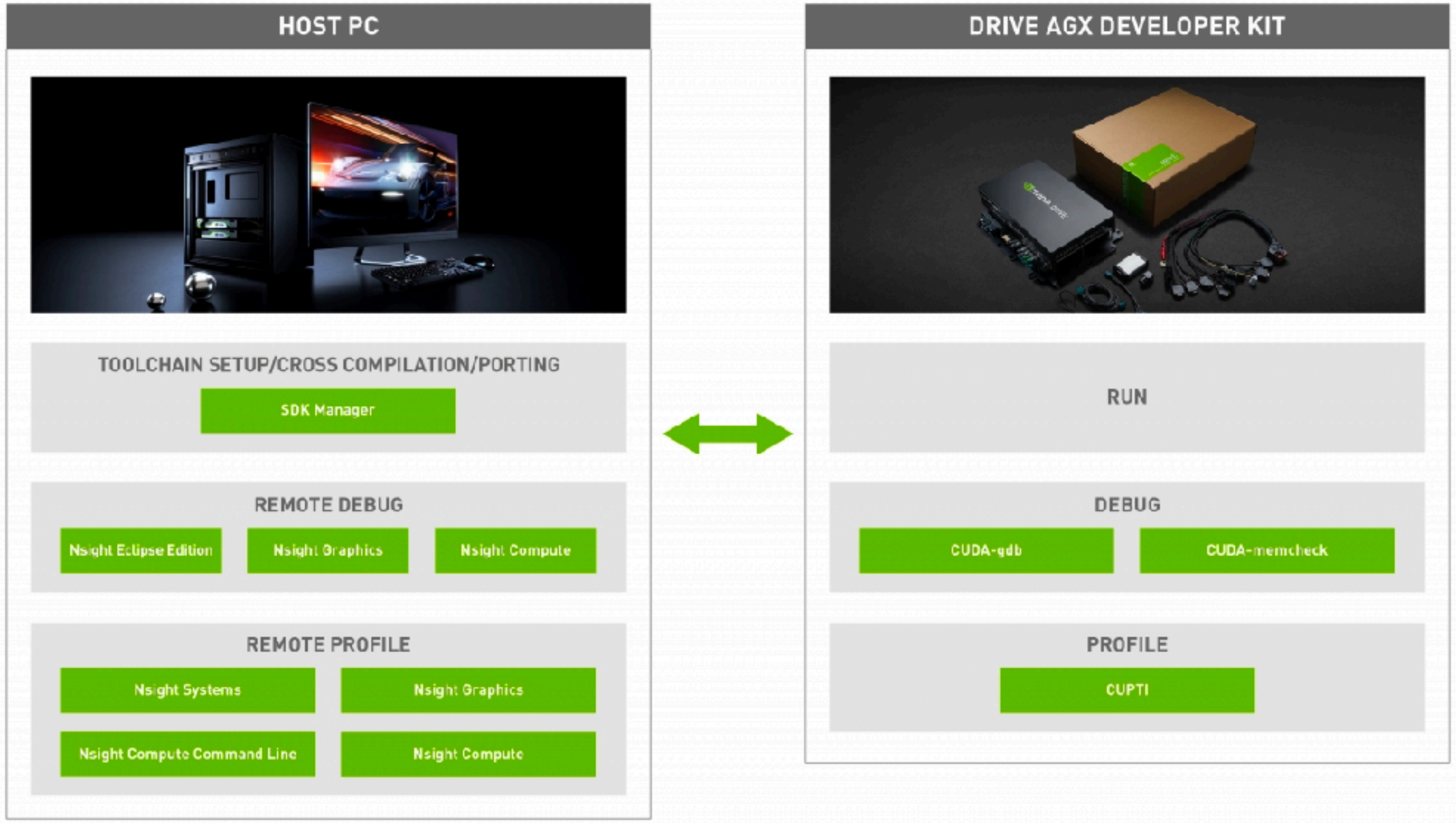
Distributed (big data)




```
fl = cloudflow.Dataflow([('url', str)])
img = fl.map(img_preproc)
p1 = img.map(resnet_101)
p2 = img.map(vgg_16)
p3 = img.map(inception_v3)
fl.output = p1.union(p2,p3).groupby(rowID).agg(max,'conf')
```

- The components may be assigned to different hardware nodes without direct control of the users
- These are typically configurable, but need expertise to find the best configuration

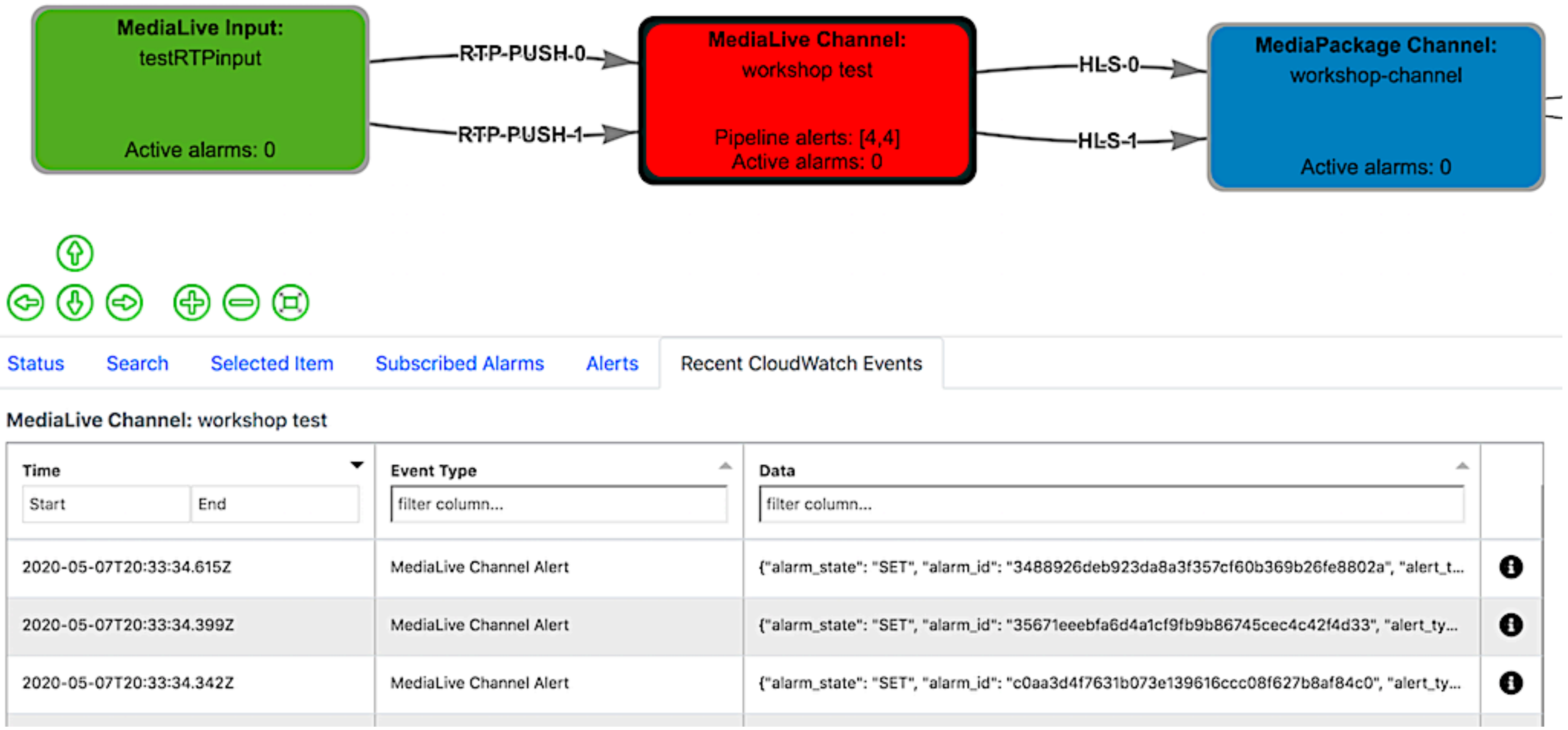
Cyber-physical systems



- We may not have direct access to the hardware directly, so remote debugging is needed.

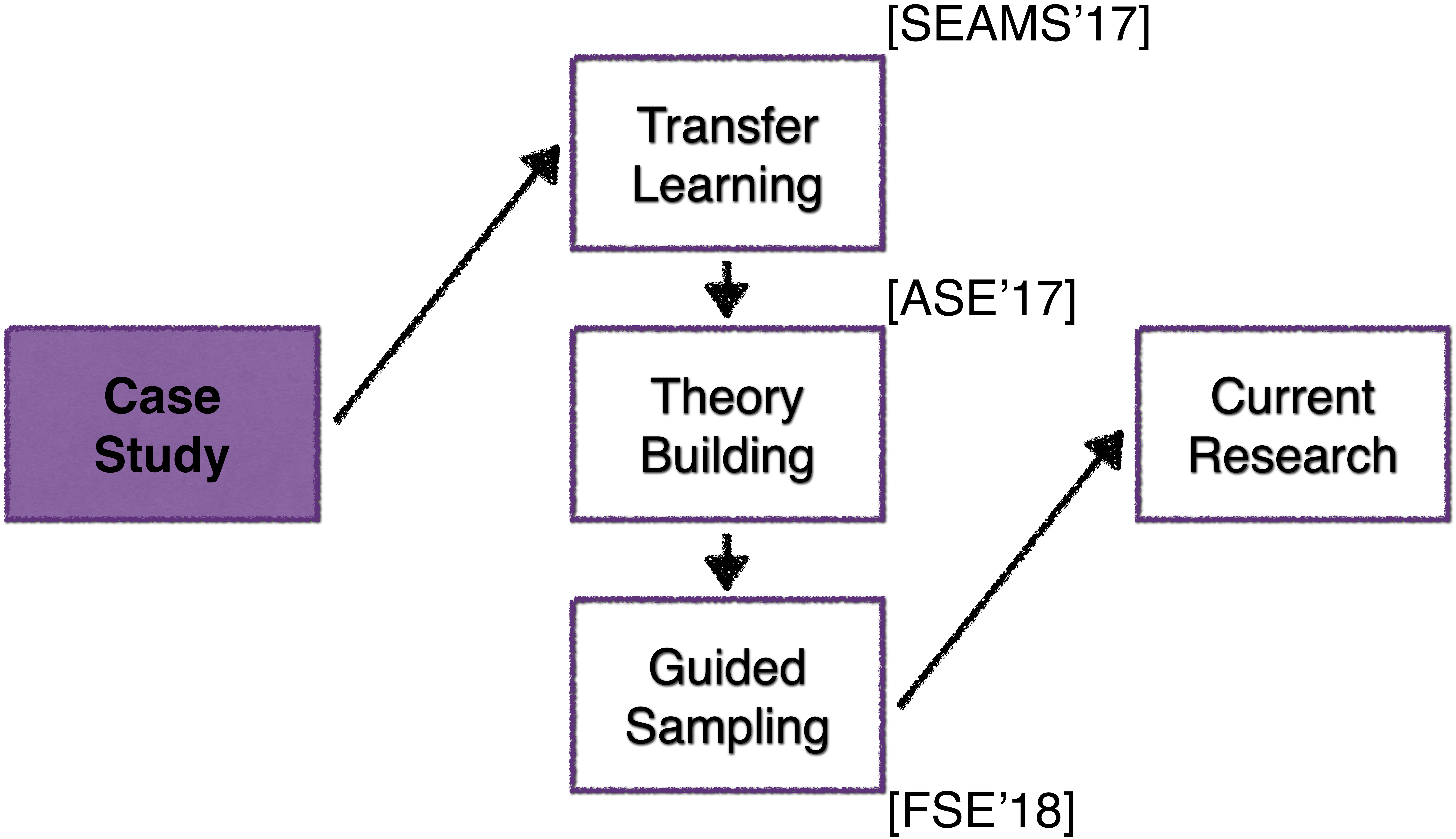
Cloud, Multi-cloud Systems

Event-driven systems



- Code migrate from one hardware to another (lots of interactions)

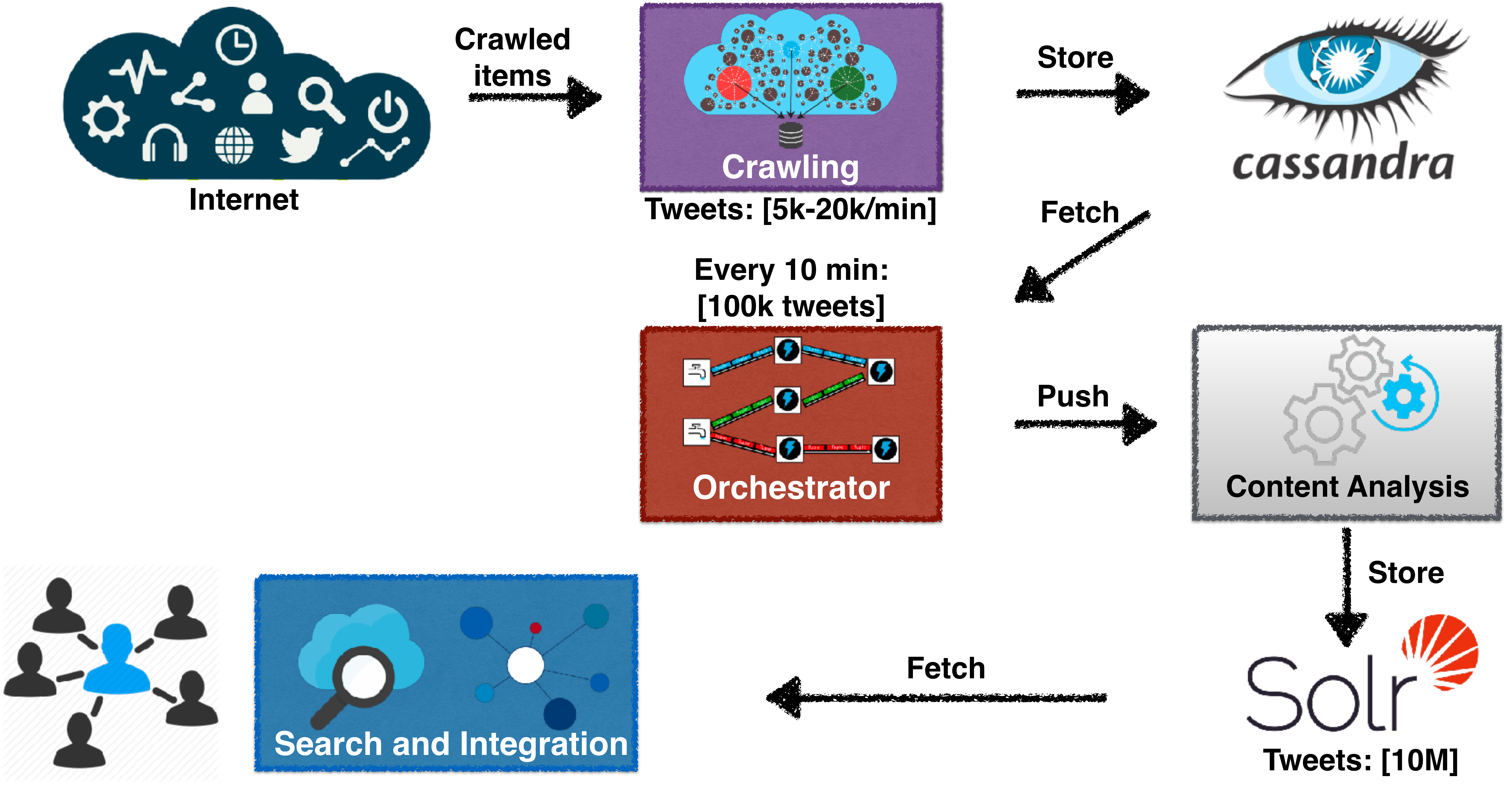
Outline


[SEAMS'17]
Transfer Learning
[ASE'17]
Case Study
Theory Building
Current Research
Guided Sampling
[FSE'18]

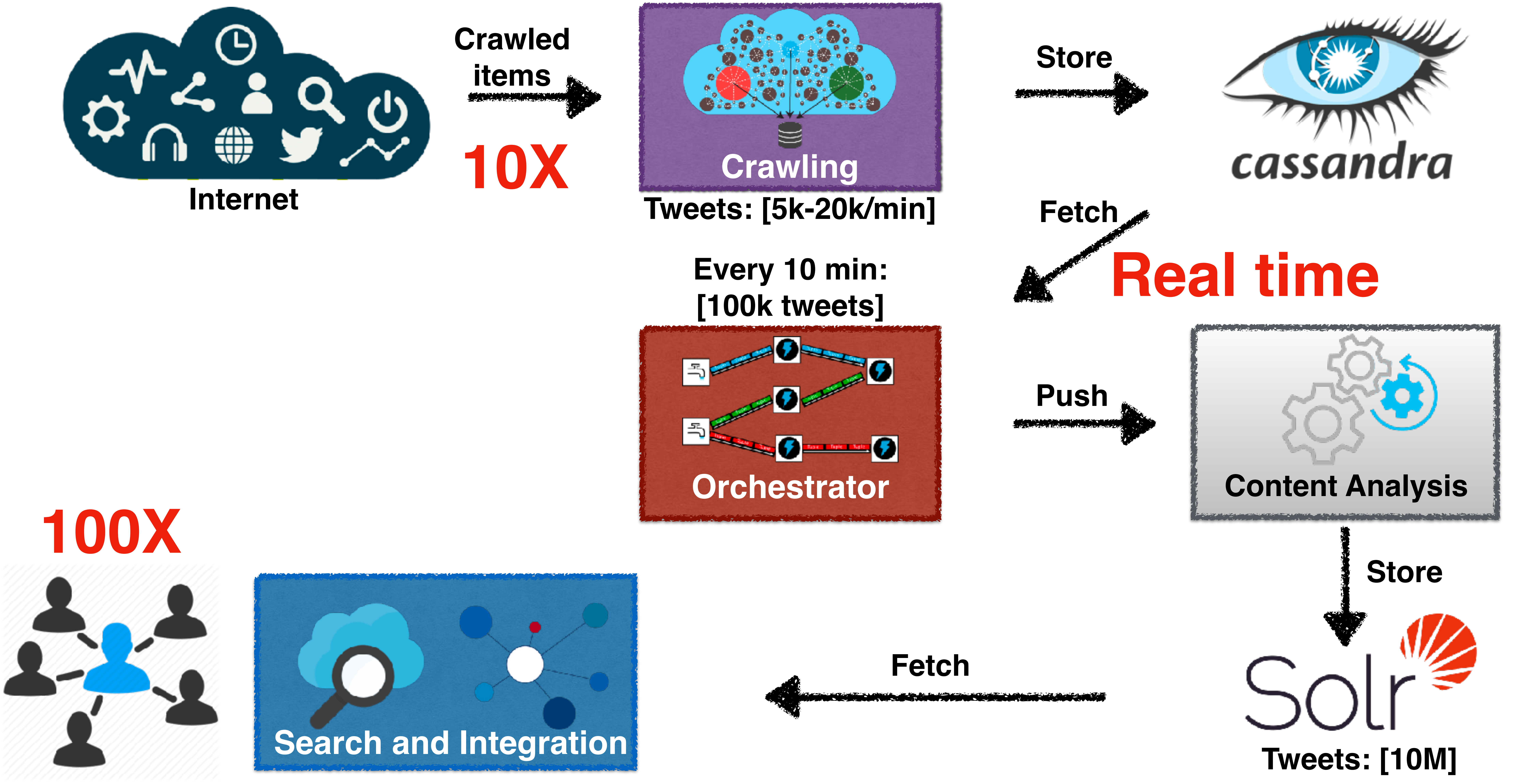
SocialSensor

- Identifying trending topics
- Identifying user defined topics
- Social media search

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

> 100

STORM

> 100

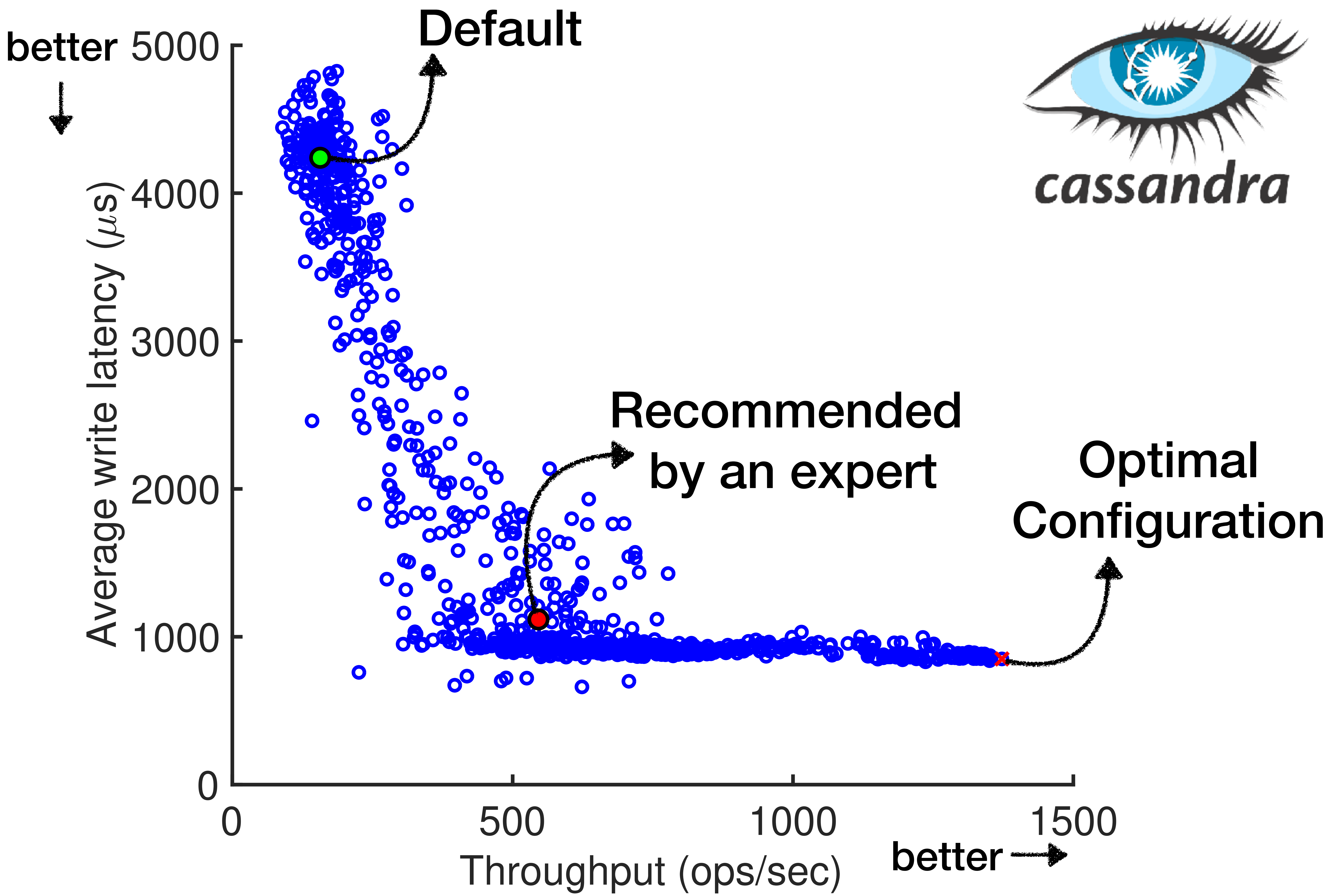
Solr

> 100

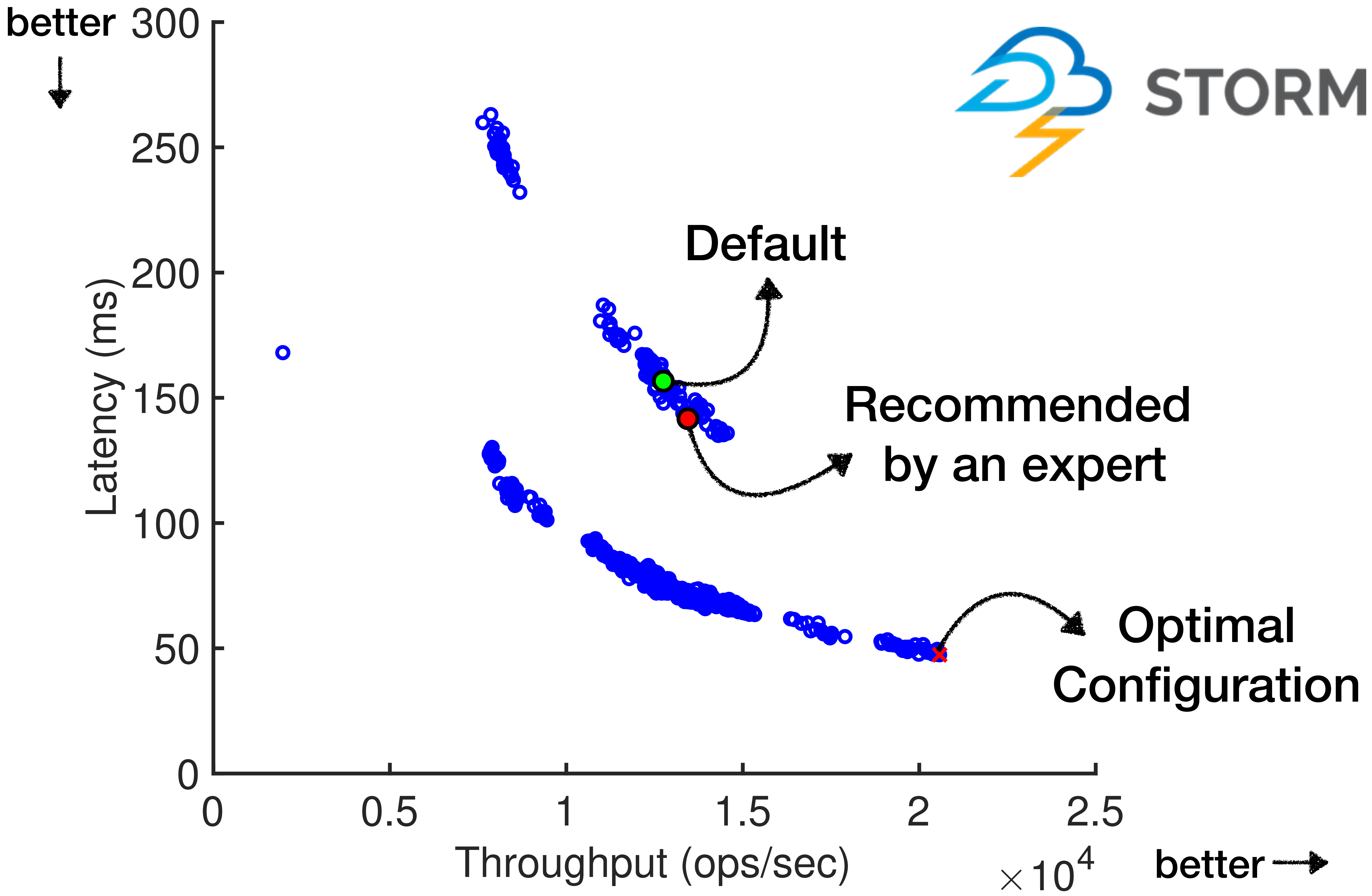
$$2^{300}$$

More combinations than estimated atoms in the universe

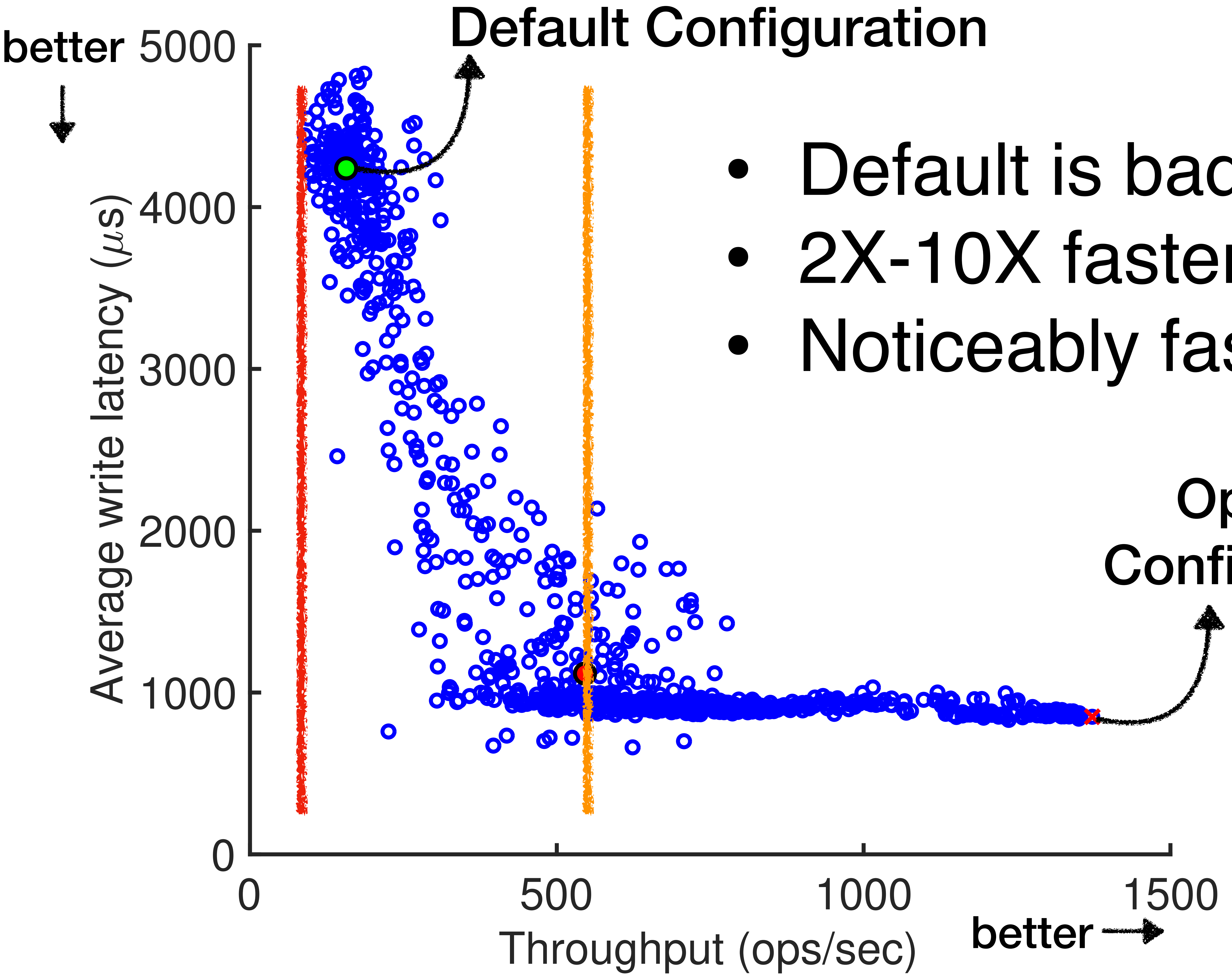
Default configuration was bad, so was the expert'



Default configuration was bad, so was the expert'



The default configuration is typically bad and the optimal configuration is noticeably better than median



cassandra

- Default is bad
- 2X-10X faster than worst
- Noticeably faster than median


100X more user
cloud resources reduced **20%**
outperform expert recommendation

Identifying the root cause of performance faults is difficult

- Code was transplanted from TX1 to TX2
- TX2 is more powerful, but software was 2x slower than TX1
- Three misconfigurations:
  - Wrong compilation flags for compiling CUDA (didn't use 'dynamic' flag)
  - Wrong CPU/GPU modes (didn't use TX2 optimized cores)
  - Wrong Fan mode (didn't change to handle thermal throttling)

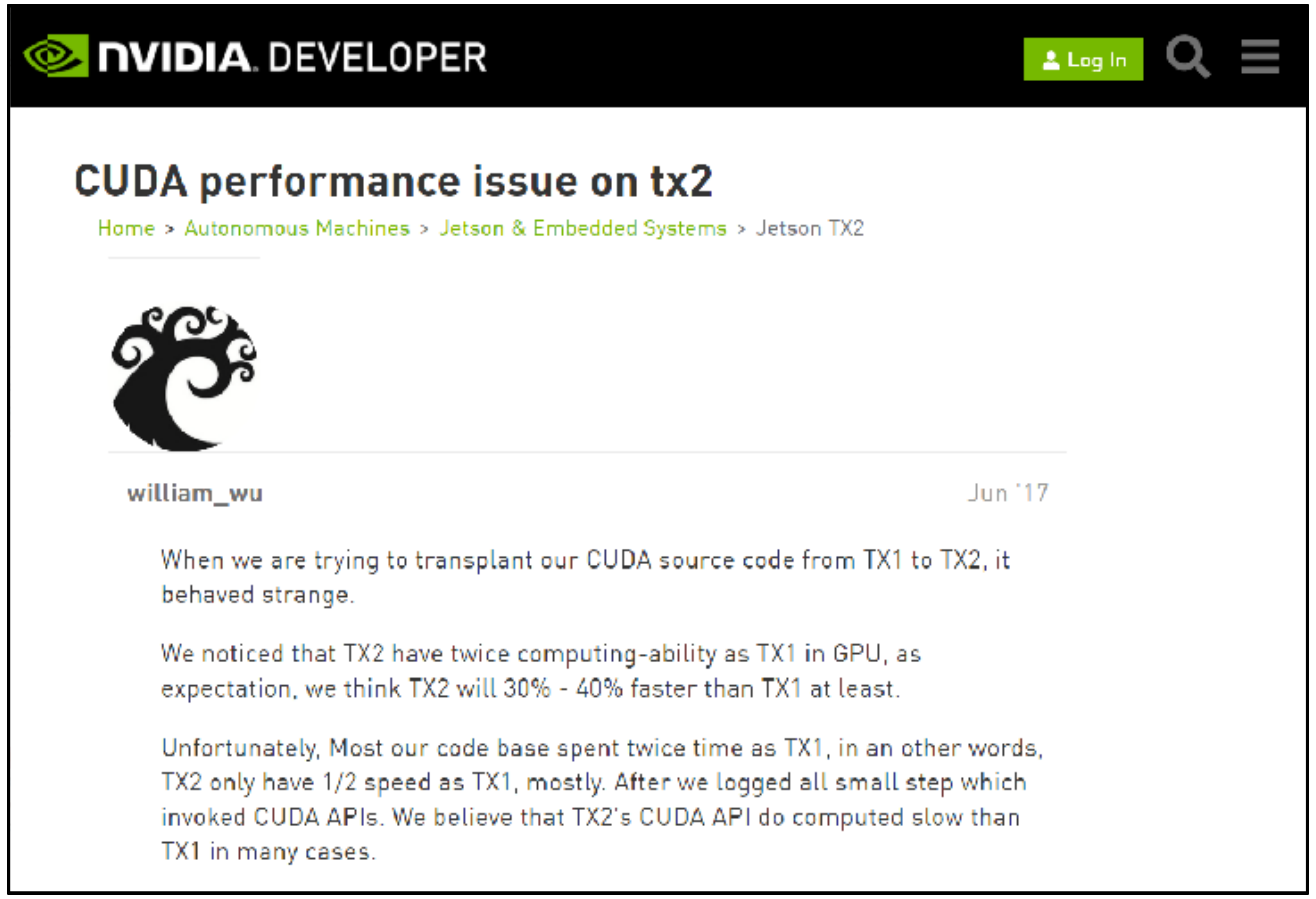


Fig 1. Performance fault on NVIDIA TX2

https://forums.developer.nvidia.com/t/50477

Fixing performance faults is difficult

- These were not in the default settings
- Took 1 month to fix in the end...
- We need to do this better

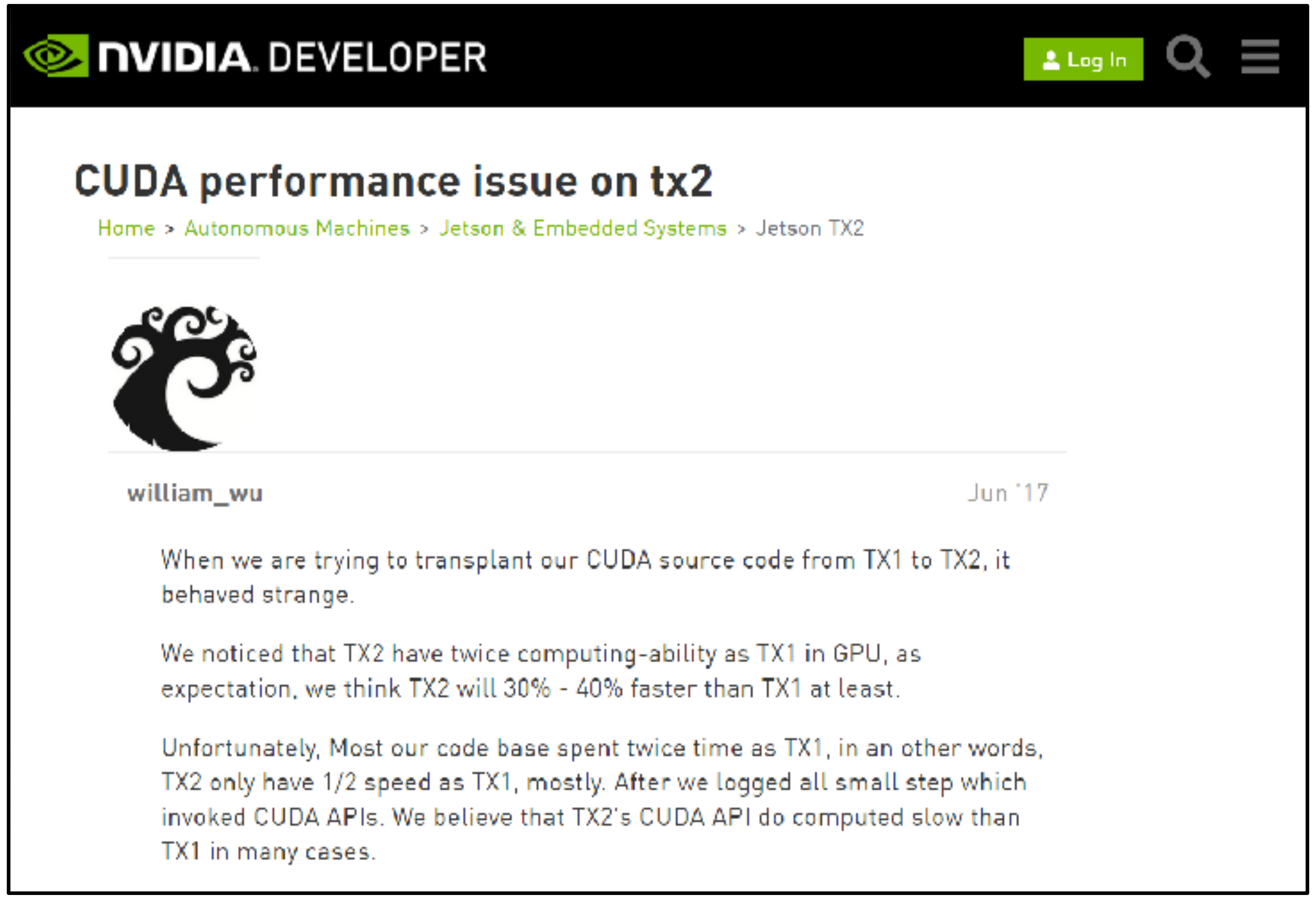


Fig 1. Performance fault on NVIDIA TX2

https://forums.developer.nvidia.com/t/50477

Identifying the root cause of performance faults is difficult

slow image classification with tensorflow on TX2

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

iandow Oct '17

I'm trying to see how much faster the TX2 can classify images with Tensorflow than a Raspberry Pi model 3. My tensorflow model was developed with transfer learning from the InceptionV3 CNN. My RPi takes about 20 seconds to classify an image, and to my surprise the TX2 also takes about 20 seconds. Here is the python script I'm using to classify the image:

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/label_image/label_image.py
2

Here's what the output looks like when I run that script:

https://gist.github.com/iandow/28b5581ab908cf145709f342b460ff31
2

Am I doing something wrong here? I was expecting the TX2 to drastically outperform the RPi.

> Solved by AastaLLL in post #2
>
> Hi, Please remember to maximize CPU/GPU frequency to have better performance. sudo ./jetson_clocks.sh To accelerate the performance of Tensorflow, you can inference a TF model with our fast TensorRT engine. More information about TensorRT can be found here: Please remember to export a UFF m...

Reply

Oct 2017

1 / 8
Oct 2017

Dec 2017

Identifying the root cause of performance faults is difficult

AastaLLL Moderator Oct '17

Hi,

Please remember to maximize CPU/GPU frequency to have better performance.

```
sudo ./jetson_clocks.sh
```

To accelerate the performance of Tensorflow, you can inference a TF model with our fast TensorRT engine.
More information about TensorRT can be found here:

> NVIDIA Developer – 5 Apr 16
>
> **NVIDIA TensorRT**
>
> NVIDIA TensorRT™ is an SDK for high-performance deep learning inference. It includes a deep learning inference optimizer and runtime that delivers low latency and high-throughput for deep learning inference applications. TensorRT-based applications...

Please remember to export a UFF model on x86-based Linux machine first, and run the UFF model with tensorRT on TX2.
Thanks.

Identifying the root cause of performance faults is difficult

iandow Oct '17

The output from my tensorflow script indicates that it's running on the GPU (see gist). I doubt the default GPU frequency is the limiting factor here. Nvidia's TensorRT image classification examples run screaming fast (like, 20 image classifications per second, fast). I think there's something wrong with how I installed tensorflow if it can only classify 1 image every 20 seconds.

Reply

AastaLLL Moderator Oct '17

Hi,

Please check if TensorFlow uses the swap memory.
Thanks.

Reply

Identifying the root cause of performance faults is difficult

25 DAYS LATER

1006045366h Nov '17

Hi

What do u mean by check if tensorflow uses swap memory ?
Whats the effect of swap memory ?

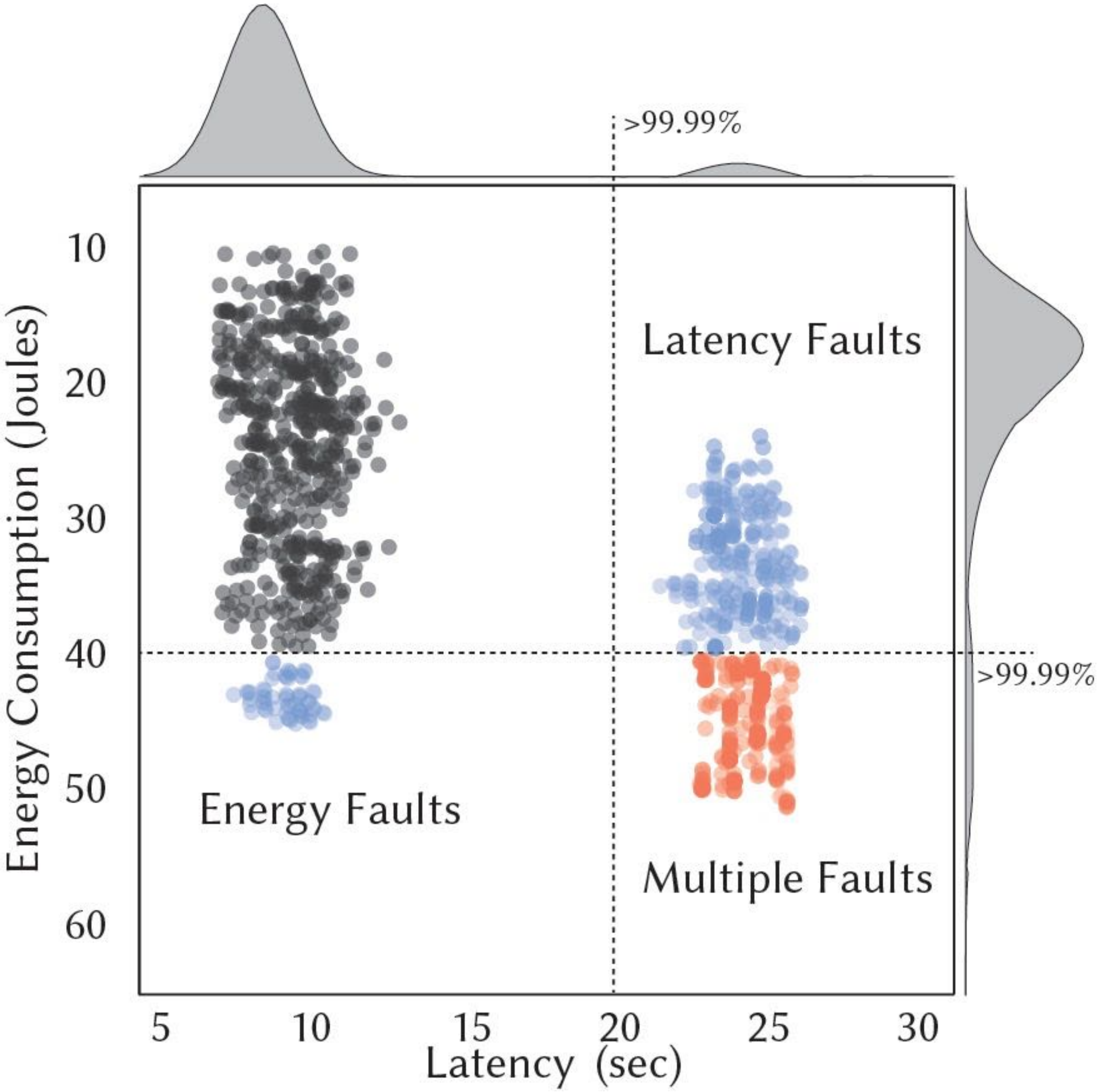
Reply

Performance distributions are multi-modal and have long tails

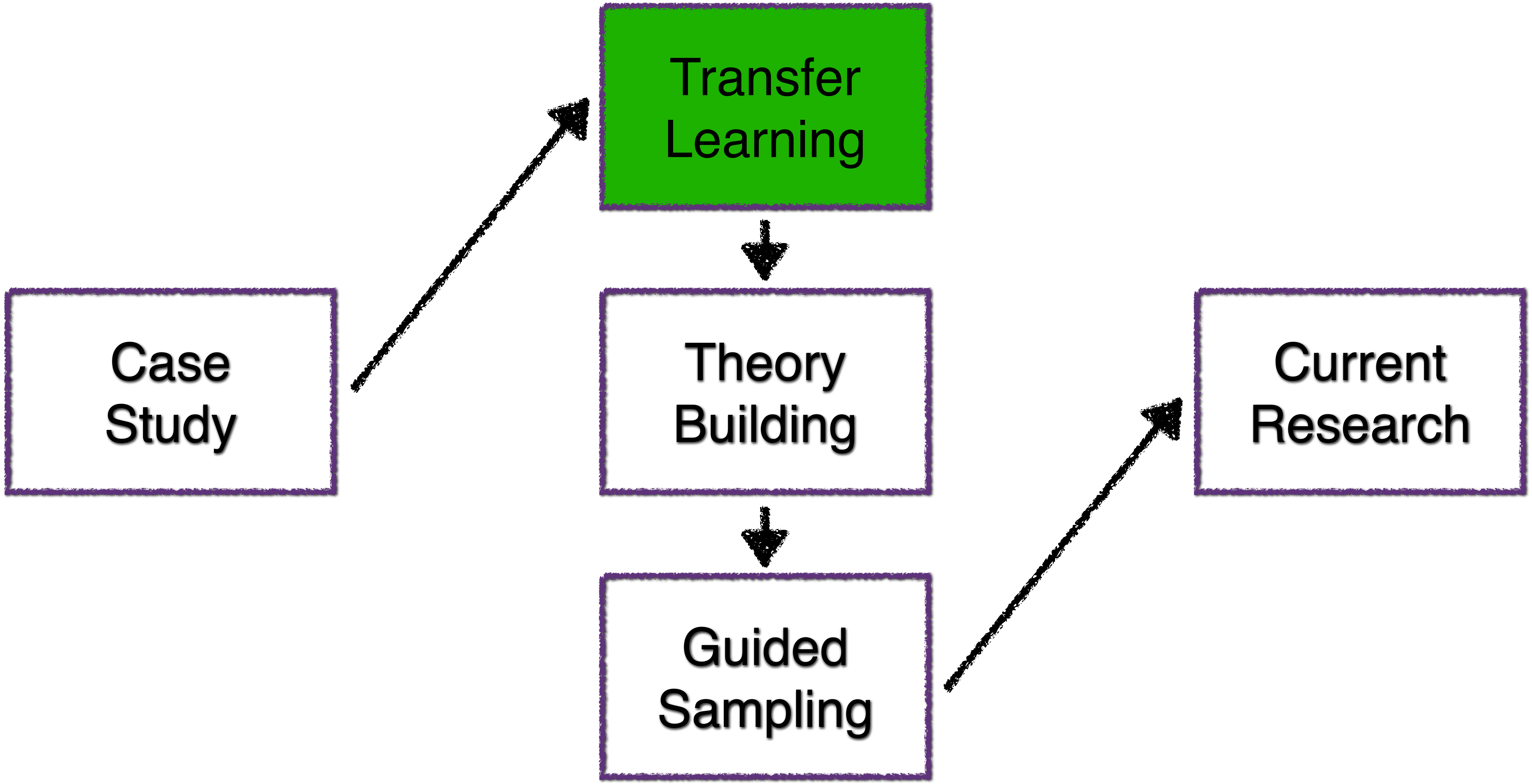
- Certain configurations can cause performance to take abnormally large values

- Faulty configurations take the tail values (worse than 99.99th percentile)

- Certain configurations can cause faults on multiple performance objectives.



Outline


Transfer Learning
Case Study
Theory Building
Current Research
Guided Sampling

Setting the scene

A typical approach for understanding the performance behavior is sensitivity analysis

$$\begin{array}{ccc}
 & O_1 \times O_2 \times \cdots \times O_{19} \times O_{20} & \\
c_1 & 0 \times 0 \times \cdots \times 0 \times 1 & y_1 = f(c_1) \\
c_2 & 0 \times 0 \times \cdots \times 1 \times 0 & y_2 = f(c_2) \\
c_3 & 0 \times 0 \times \cdots \times 1 \times 1 & y_3 = f(c_3) \\
 & \cdots & \cdots \\
 & 1 \times 1 \times \cdots \times 1 \times 0 & \\
c_n & 1 \times 1 \times \cdots \times 1 \times 1 & y_n = f(c_n)
\end{array}$$

$$\boxed{\text{Training/Sample Set}} \xrightarrow{\text{Learn}} \hat{f} \sim f(\cdot)$$

Performance model could be in any appropriate form of black-box models

$$
\begin{array}{cc}
 & O_1 \times O_2 \times \cdots \times O_{19} \times O_{20} \\
c_1 & 0 \times 0 \times \cdots \times 0 \times 1 \\
c_2 & 0 \times 0 \times \cdots \times 1 \times 0 \\
c_3 & 0 \times 0 \times \cdots \times 1 \times 1 \\
 & \cdots \\
 & 1 \times 1 \times \cdots \times 1 \times 0 \\
c_n & 1 \times 1 \times \cdots \times 1 \times 1
\end{array}
$$

$$y_1 = f(c_1)$$

$$y_2 = f(c_2)$$

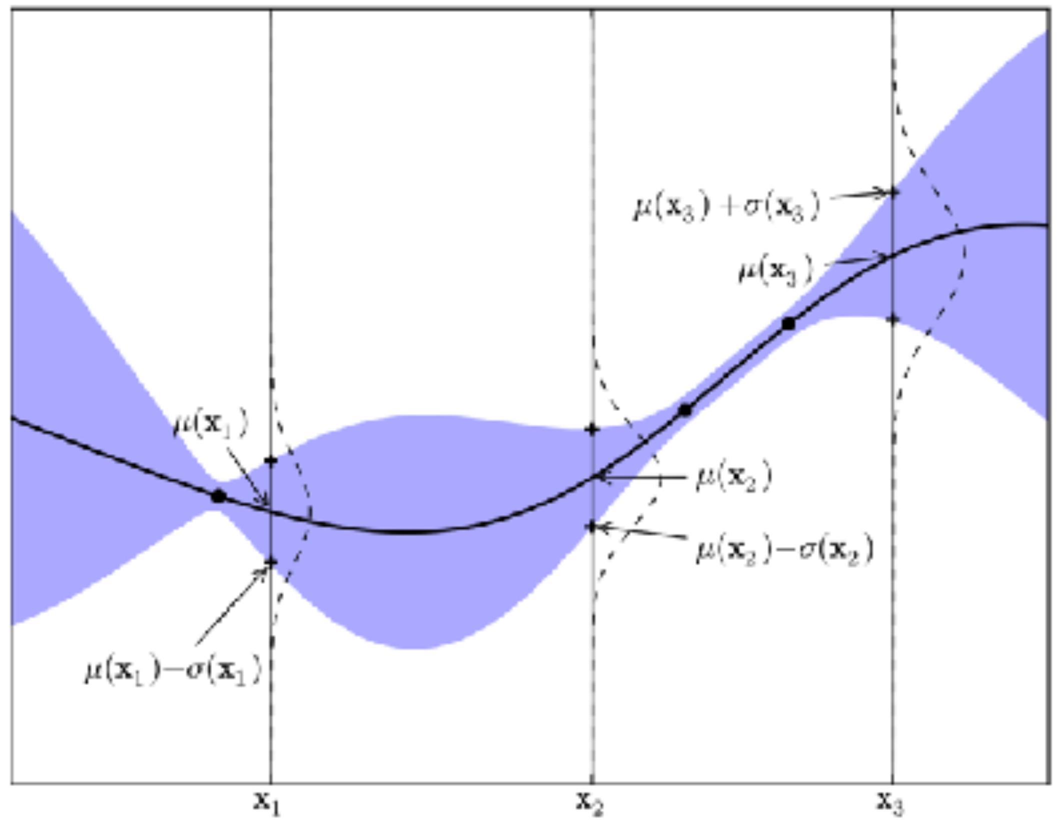
$$y_3 = f(c_3)$$

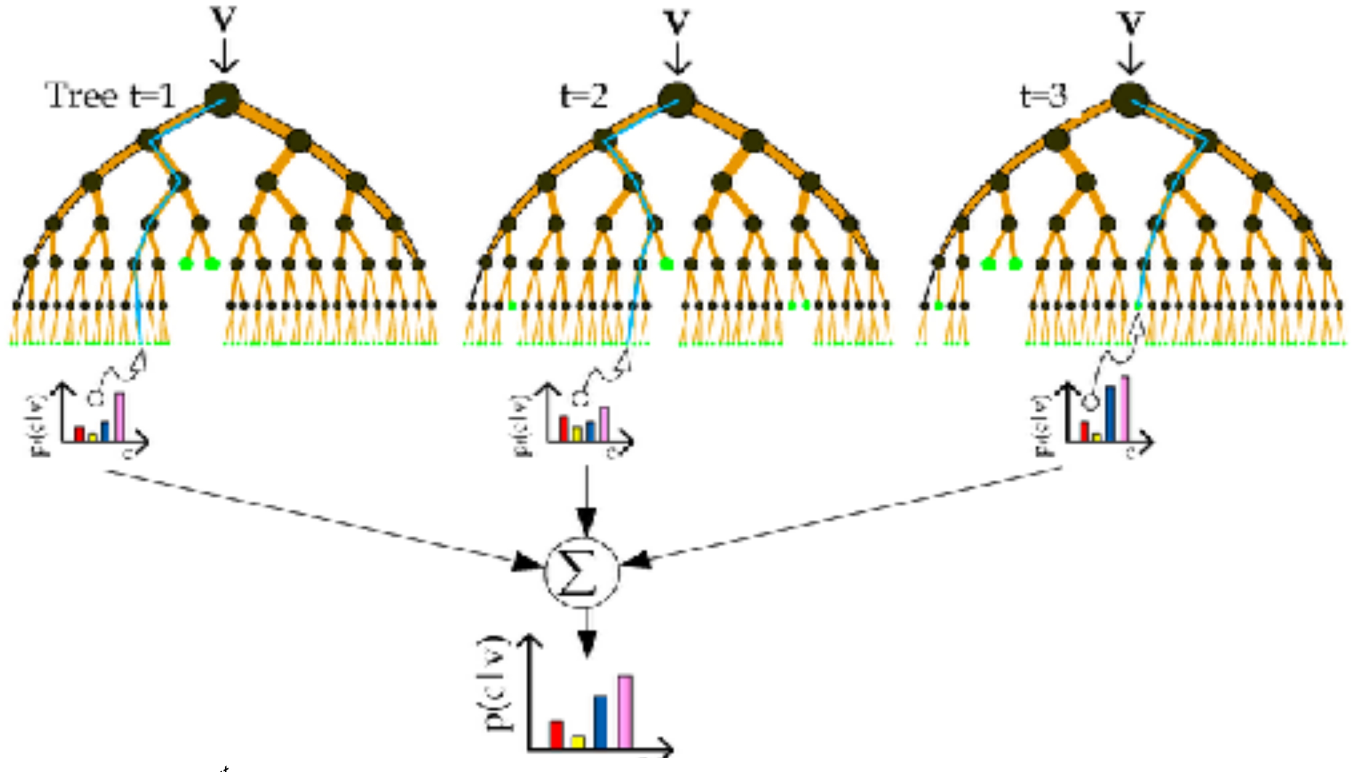
$$\cdots$$

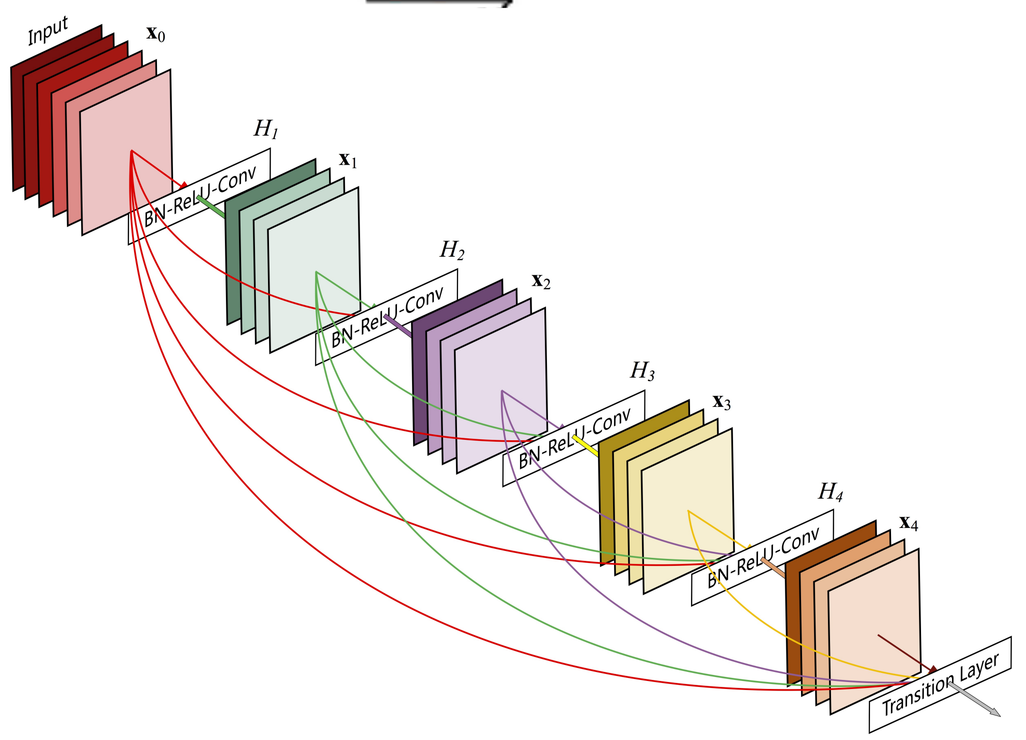
Training/Sample Set

$$\hat{f} \sim f(\cdot)$$

$$y_n = f(c_n)$$







Evaluating a performance model

$$
\begin{array}{l|l|}
 & O_1 \times O_2 \times \cdots \times O_{19} \times O_{20} \\
c_1 & 0 \times 0 \times \cdots \times 0 \times 1 \\
c_2 & 0 \times 0 \times \cdots \times 1 \times 0 \\
c_3 & 0 \times 0 \times \cdots \times 1 \times 1 \\
 & \cdots \\
 & 1 \times 1 \times \cdots \times 1 \times 0 \\
c_n & 1 \times 1 \times \cdots \times 1 \times 1
\end{array}
$$

$y_1 = f(c_1)$

$y_2 = f(c_2)$

$y_3 = f(c_3)$

$\ldots$

Training/Sample Set

$y_n = f(c_n)$

Evaluate →

Accuracy

$$APE(\hat{f}, f) = \frac{|\hat{f}(c) - f(c)|}{f(c)} \times 100$$

Learn →

$$\hat{f} \sim f(\cdot)$$

A performance model contain useful information about influential options and interactions

$$f : \mathbb{C} \to \mathbb{R}$$

$$f(\cdot) = 1.2 + 3\boxed{o_1} + 5\boxed{o_3} + 0.9\boxed{o_7} + 0.8\boxed{o_3 o_7} + 4\boxed{o_1 o_3 o_7}$$

Performance model can then be used to reason about qualities

```
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
}
```

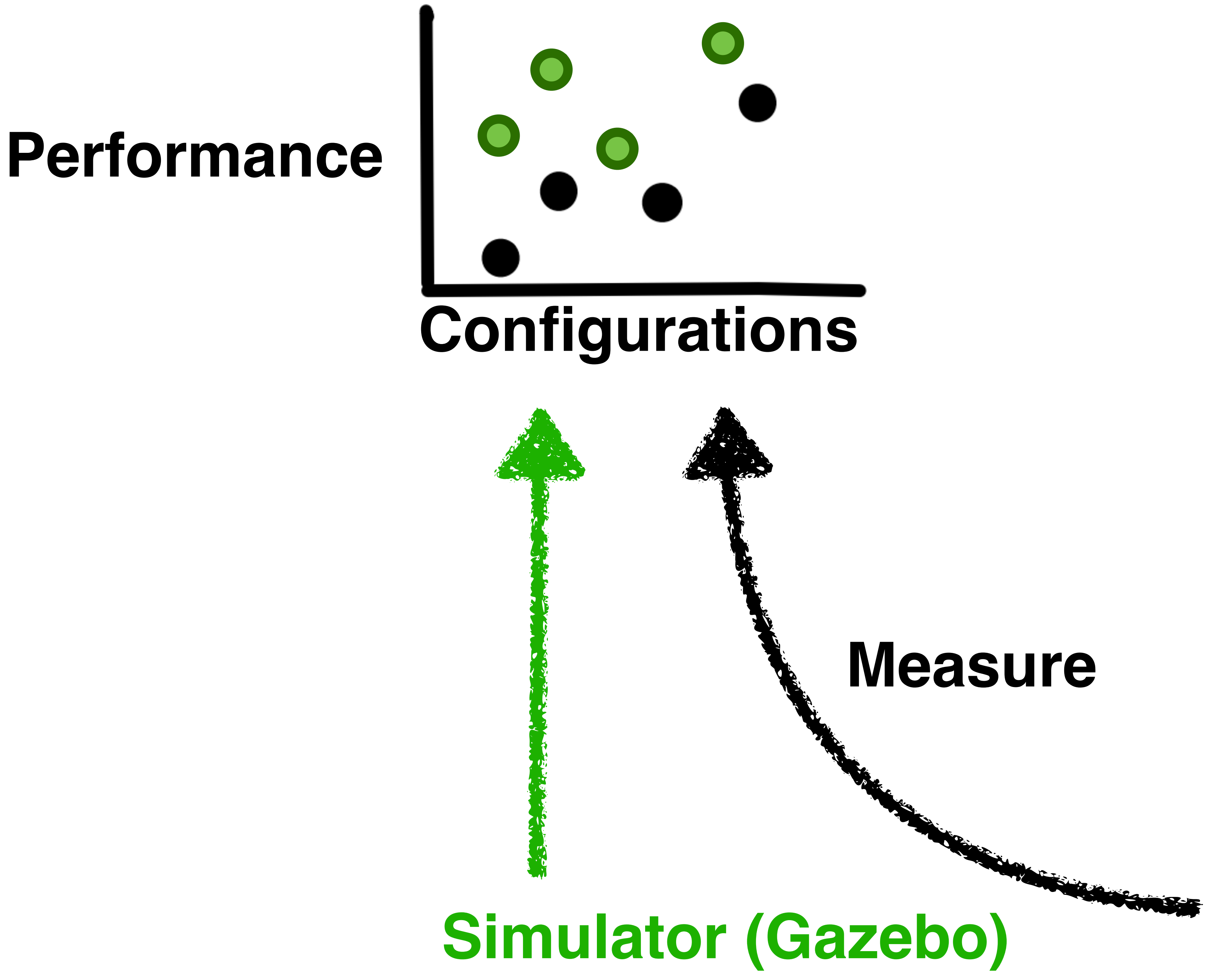
Execution time (s)

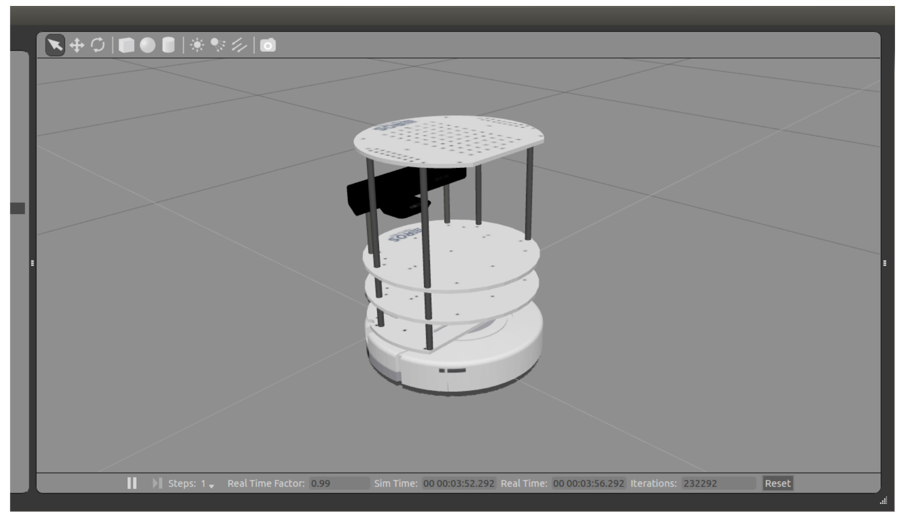
$$f(\cdot) = 5 + 3 \times o_1$$

$$\boxed{f(o_1 := 1) = 8}$$

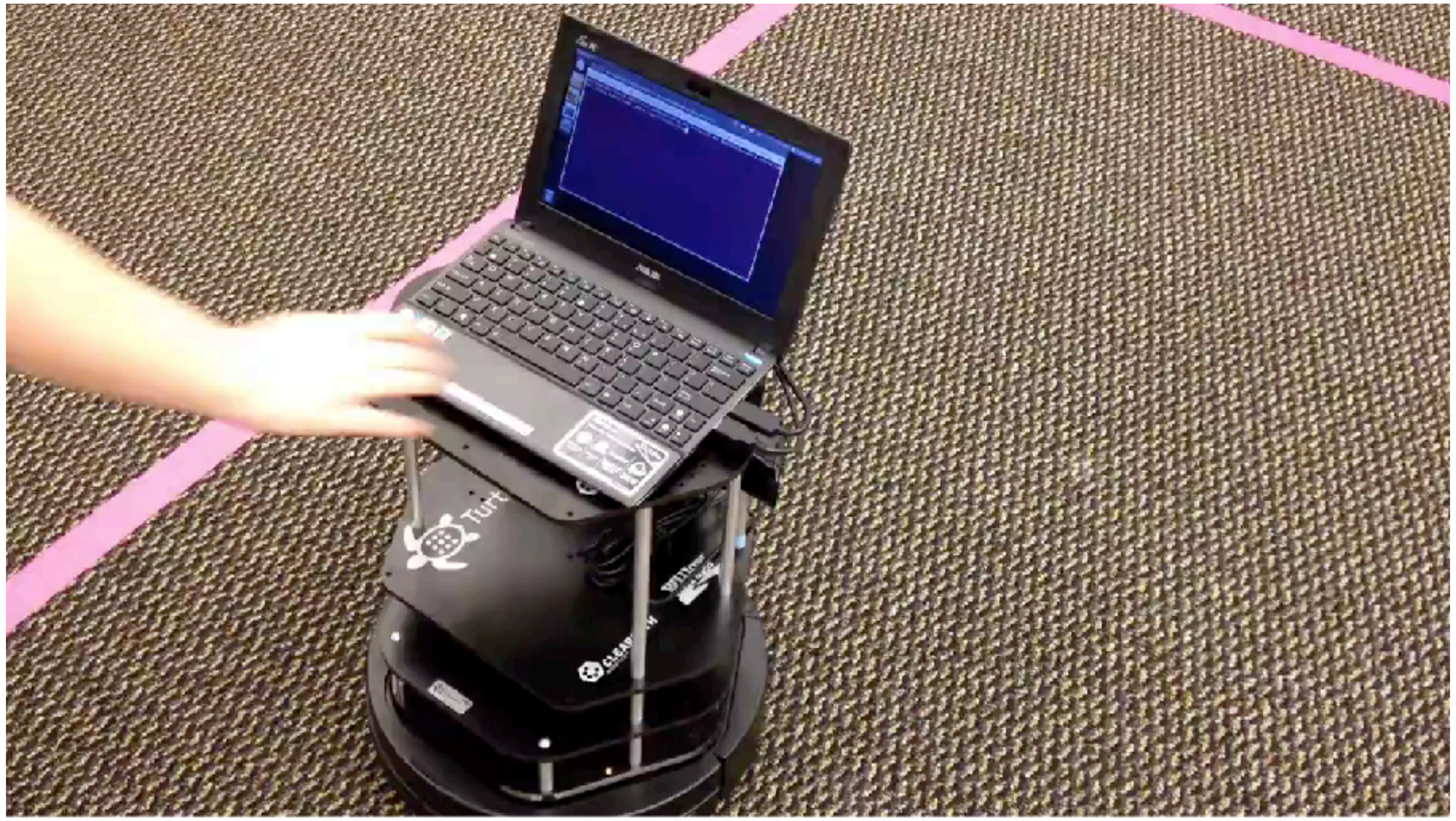
$$\boxed{f(o_1 := 0) = 5}$$

Insight: Performance measurements of the real system is “**similar**” to the ones from the simulators



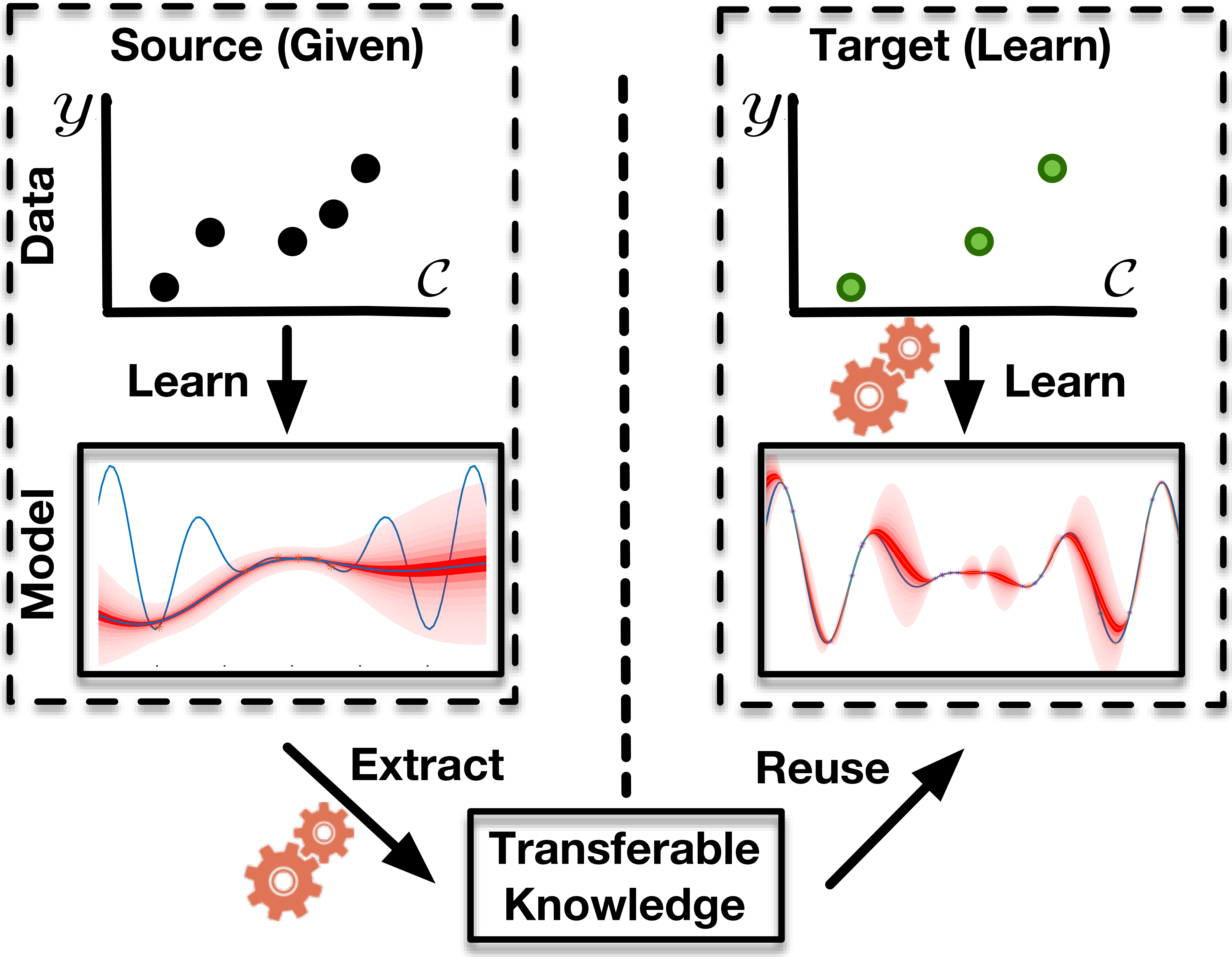


So why not reuse these data, instead of measuring on real robot?



We developed methods to make learning cheaper via **transfer learning**

Goal: Gain strength by transferring information across environments



What is the advantage of transfer learning?

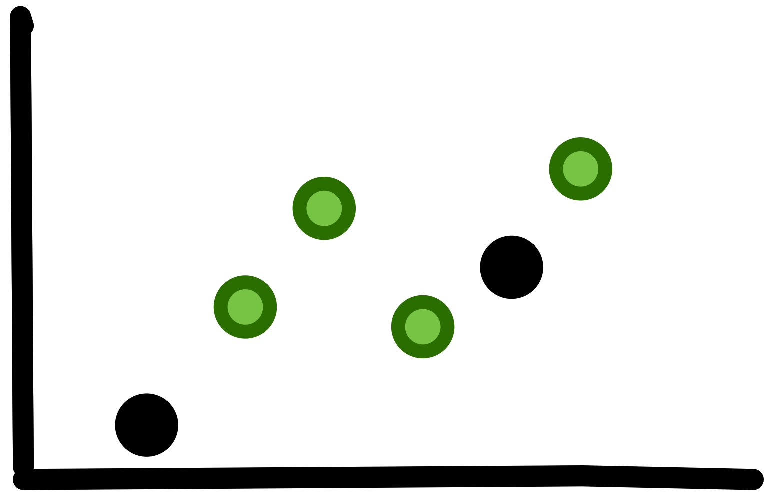
- During learning you may need thousands of rotten and fresh potato and hours of training to learn.
- But now using the same knowledge of rotten features you can identify rotten tomato with **less samples and training time**.
- You may have learned during **daytime** with enough light and exposure; but your present tomato identification job is at **night**.
- You may have learned sitting very **close**, just beside the box of potato; but now for tomato identification you are in the **other side of the glass**.

Our transfer learning solution

TurtleBot

Measure

Data

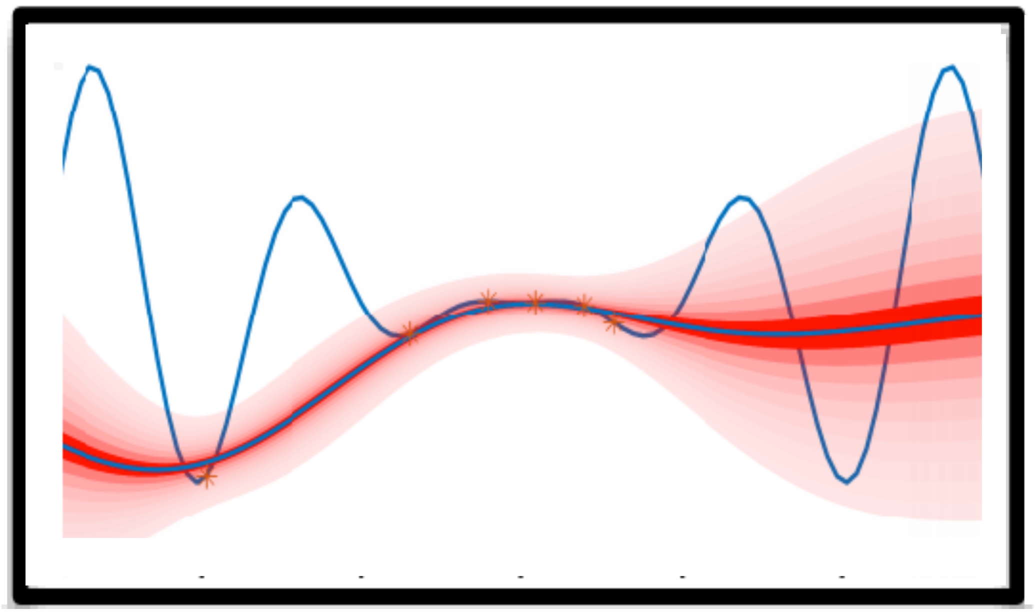


Configurations

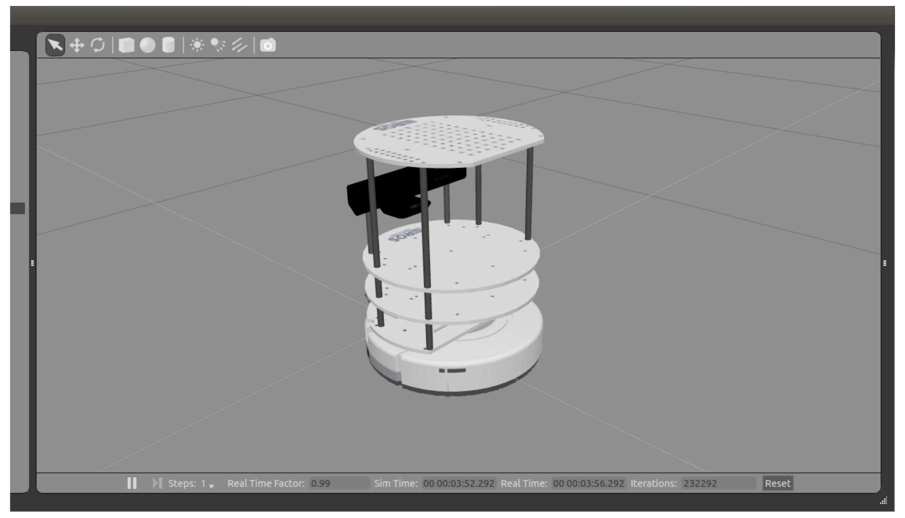
Reuse

Learn

$$f(o_1, o_2) = 5 + 3o_1 + 15o_2 - 7o_1 \times o_2$$

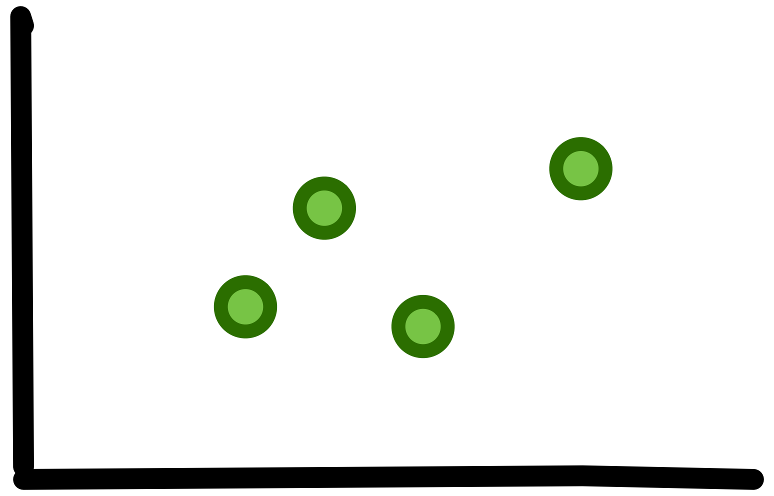


Simulator (Gazebo)



Measure

Data



[P. Jamshidi, et al., “**Transfer learning for improving model predictions** ….”, SEAMS’17]

Gaussian processes for performance modeling

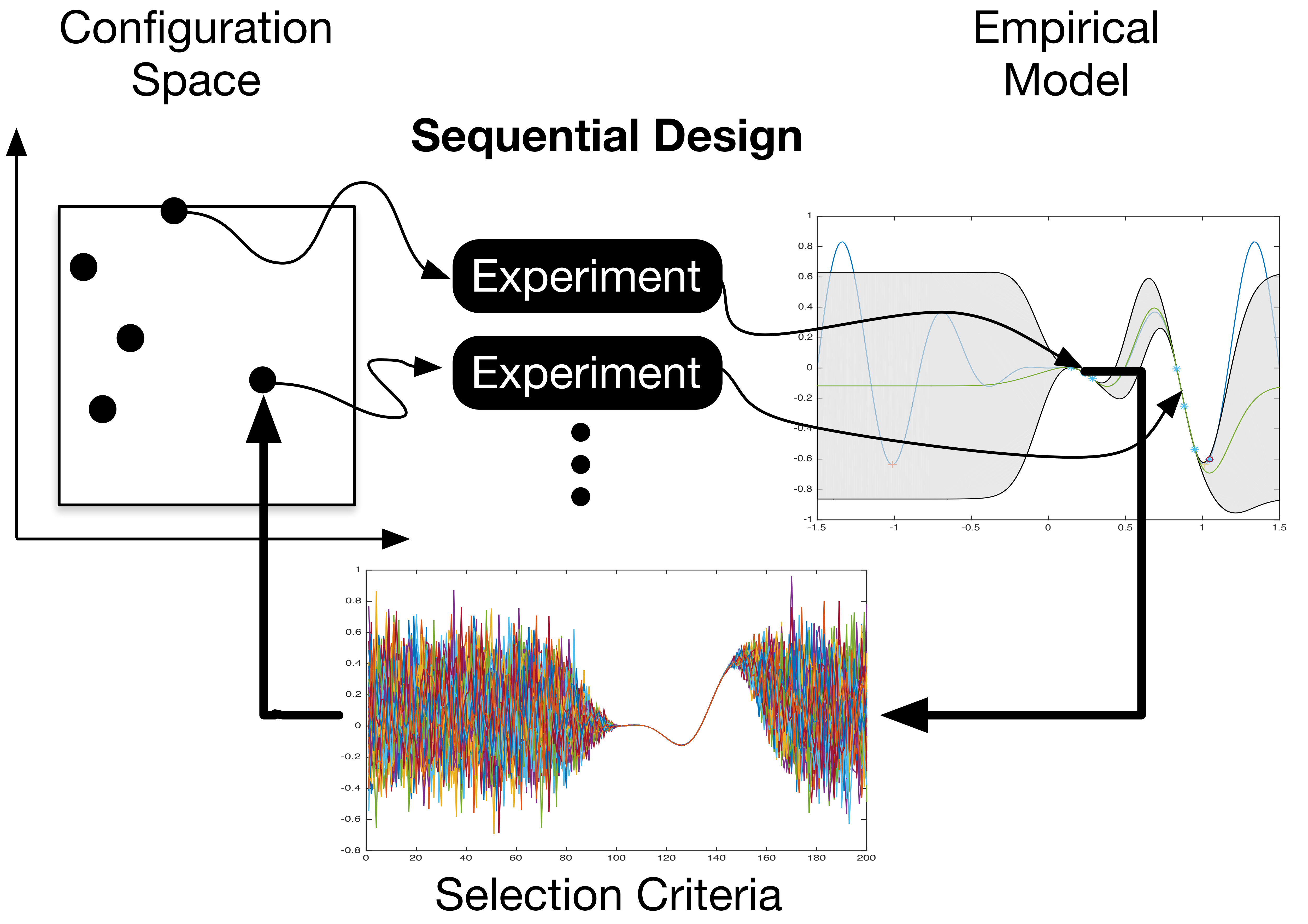
Gaussian Processes enables reasoning about performance

Step 1: Fit GP to the data seen so far

Step 2: Explore the model for regions of most variance

Step 3: Sample that region

Step 4: Repeat

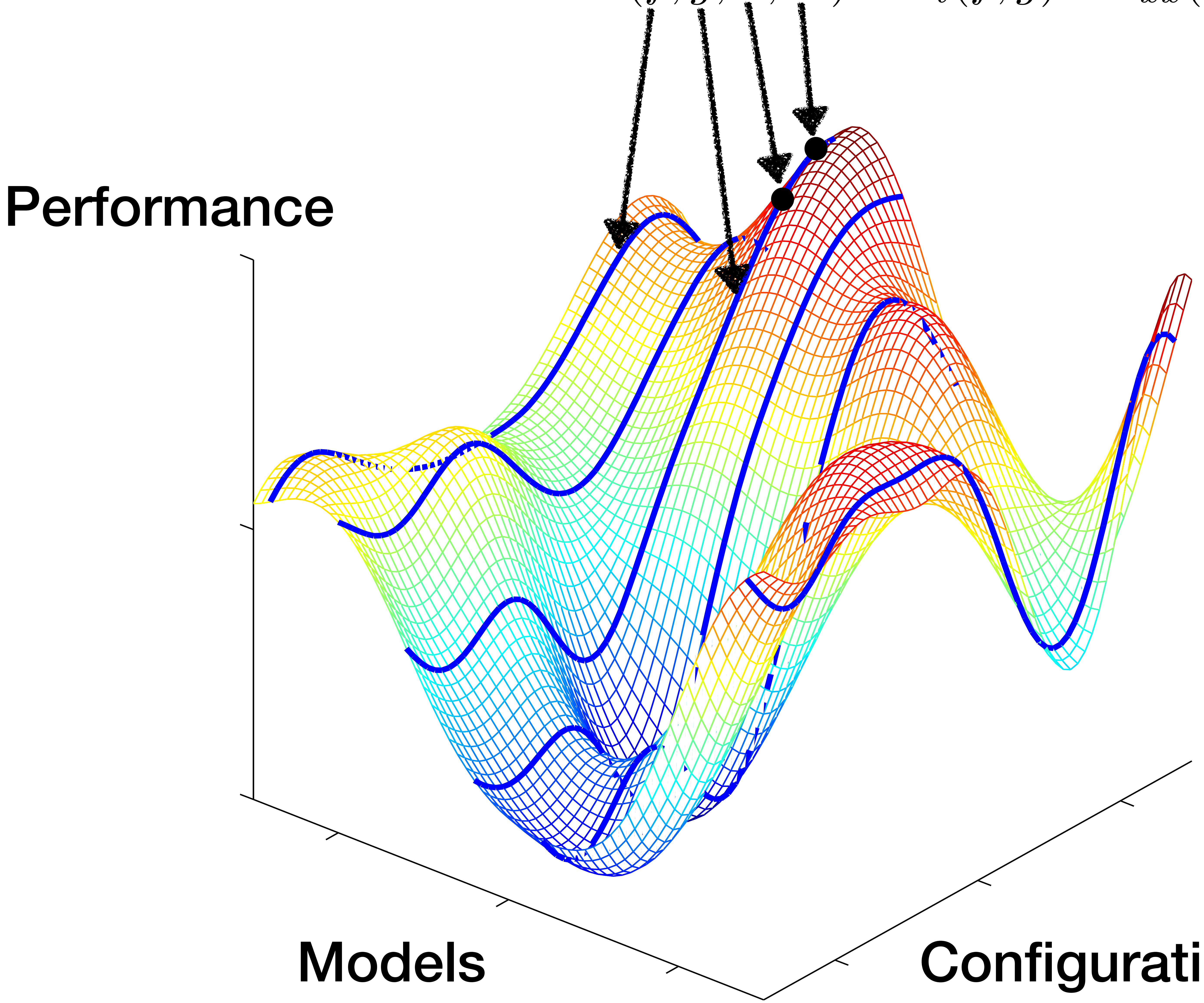


The intuition behind our transfer learning approach

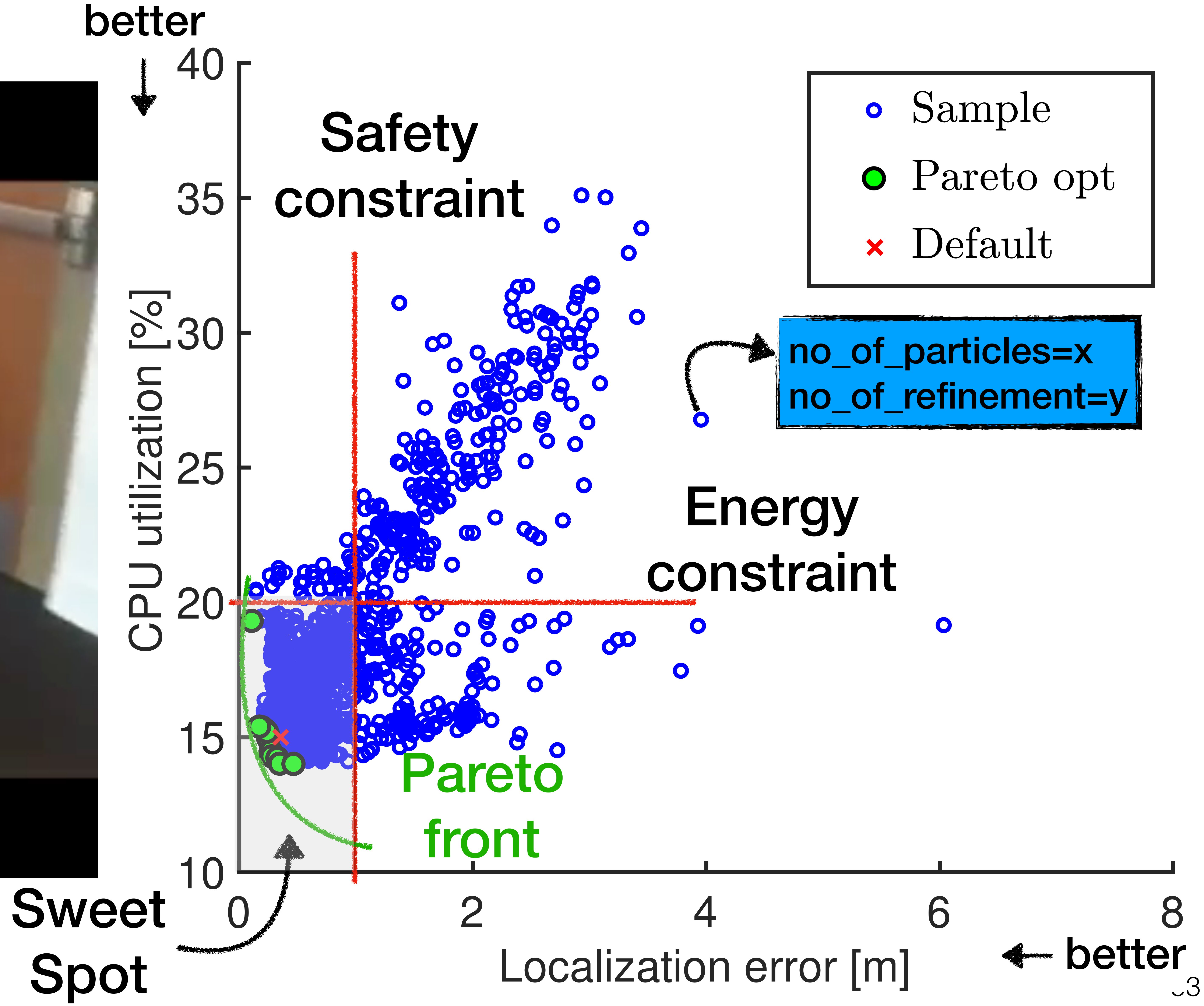
Intuition: Observations on the source(s) can affect predictions on the target

Example: Learning the chess game make learning the Go game a lot easier!

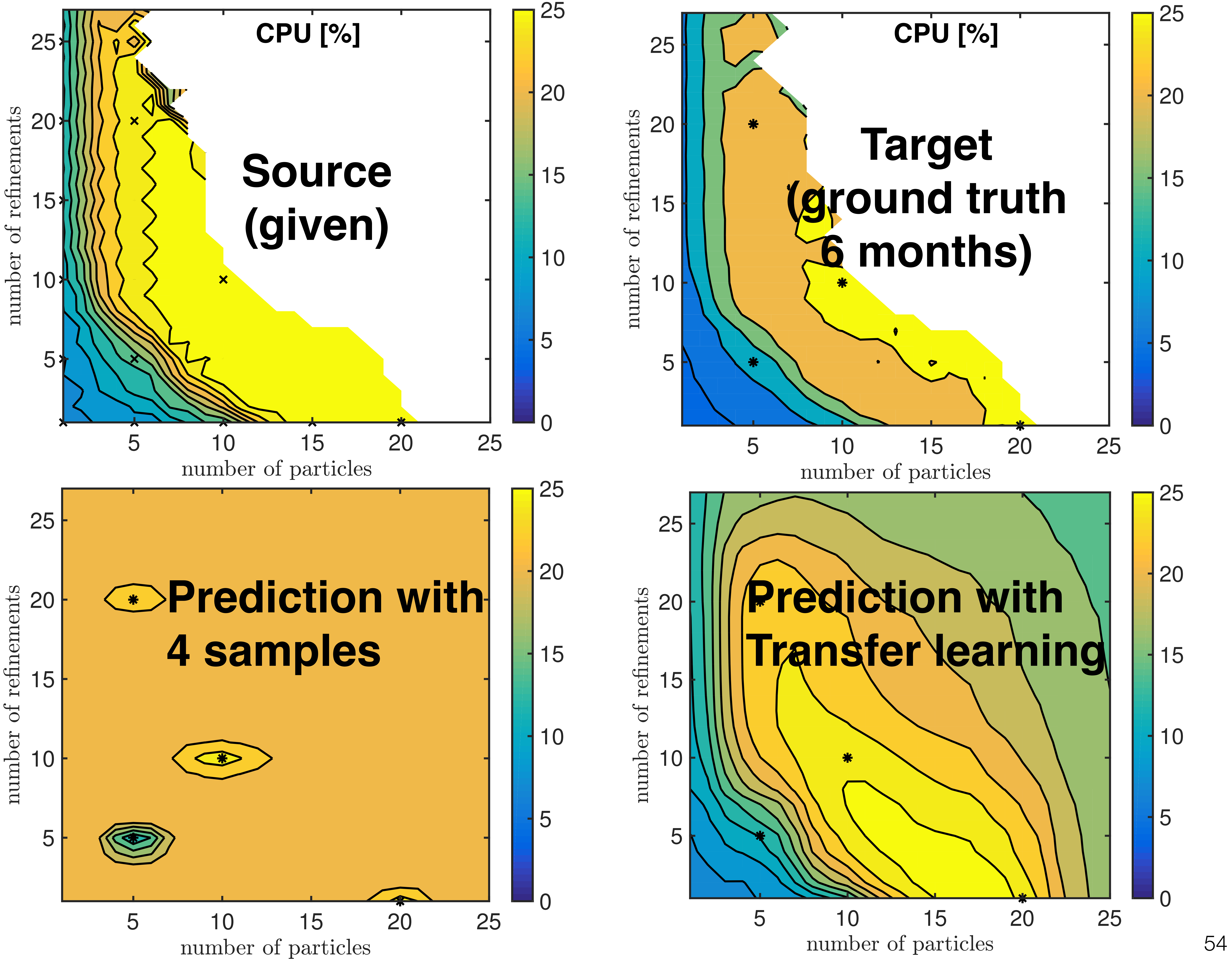
$$k(f, g, \boldsymbol{x}, \boldsymbol{x}') = k_t(f, g) \times k_{xx}(\boldsymbol{x}, \boldsymbol{x}')$$



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:**

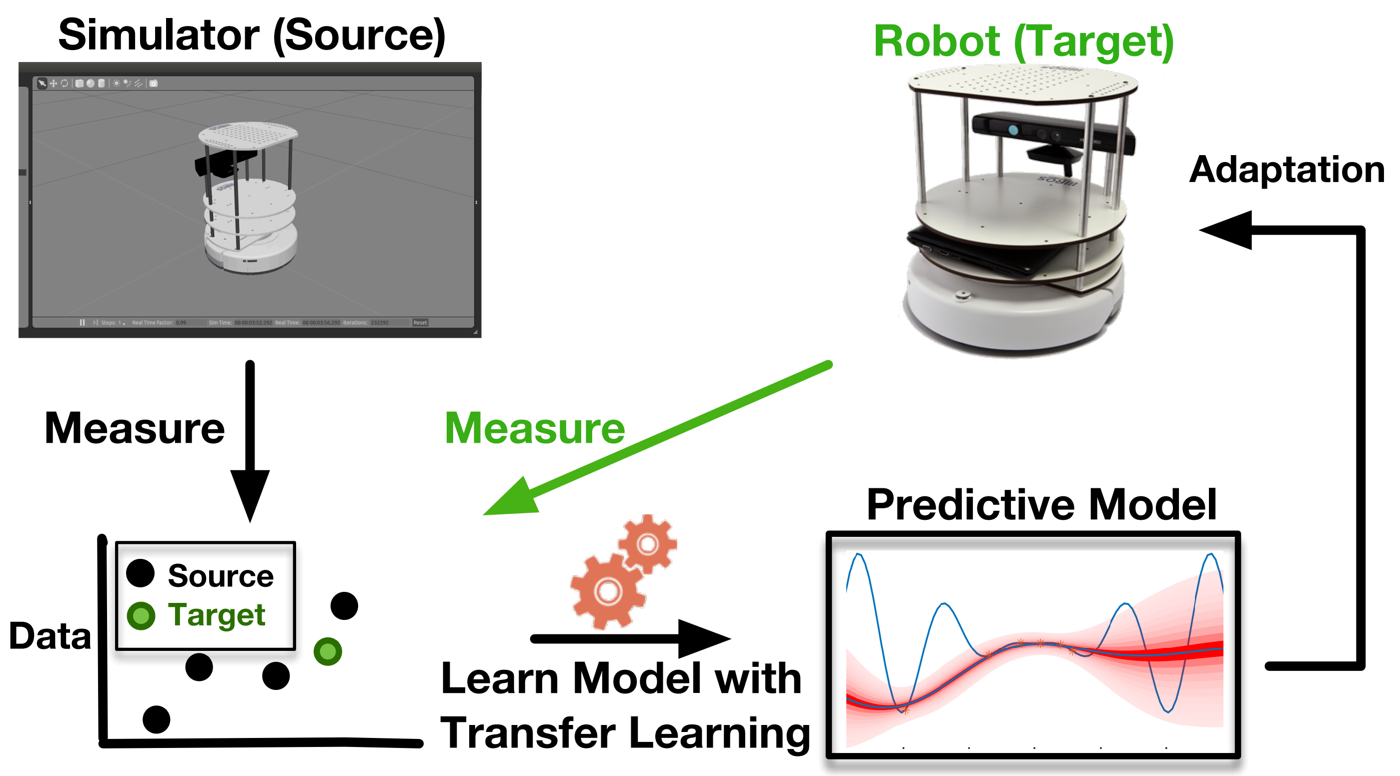
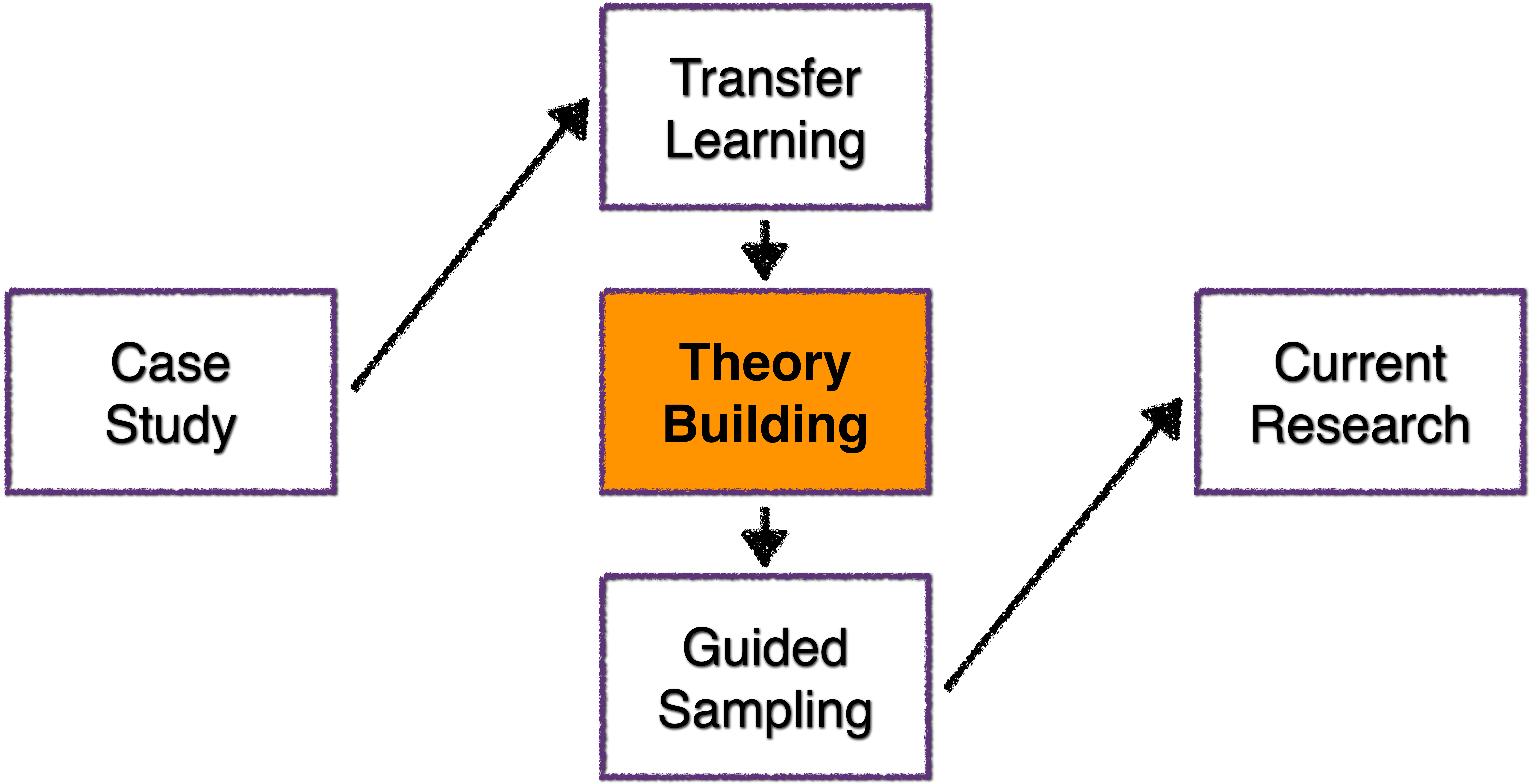


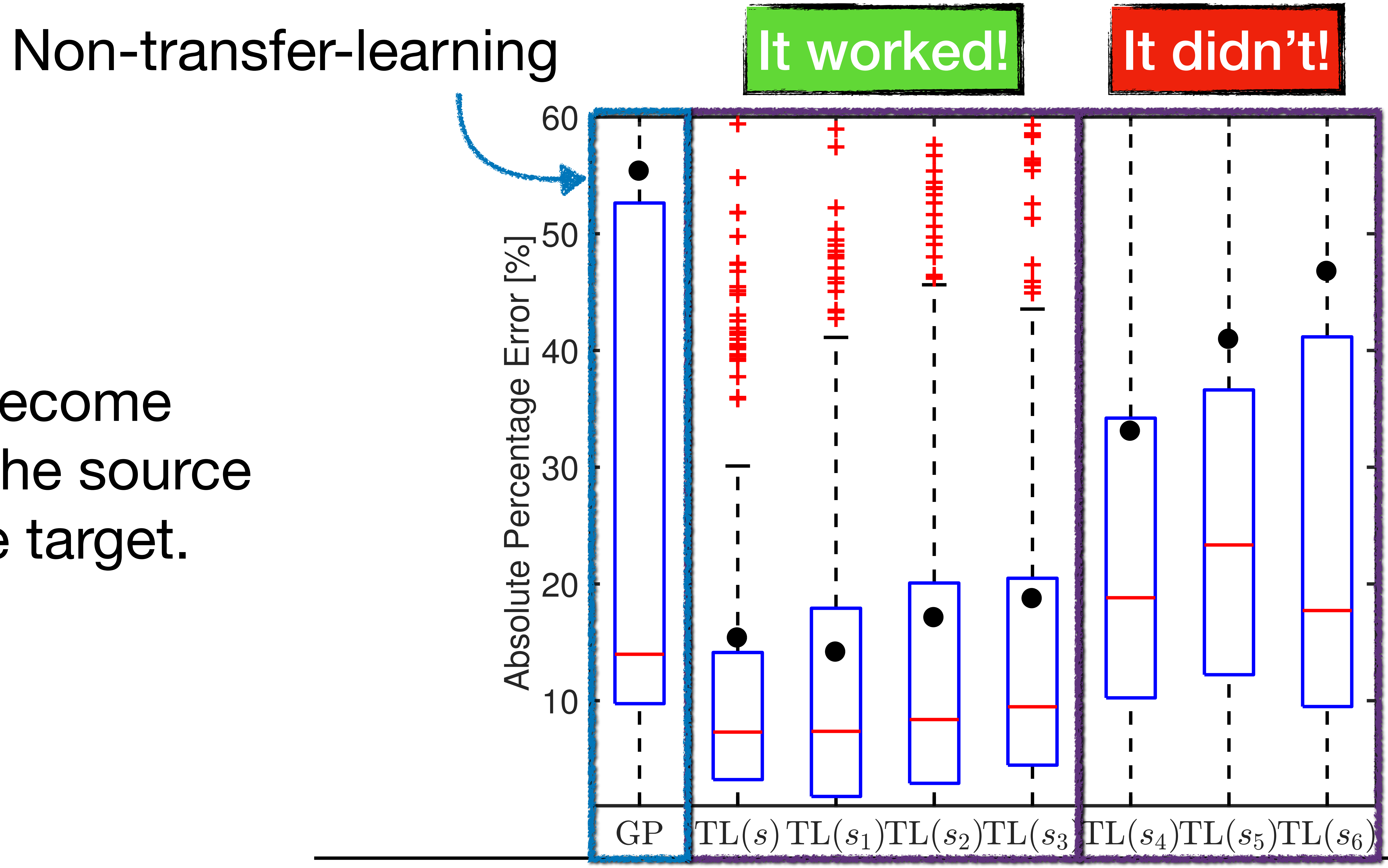
Fig. 1: Transfer learning for performance model learning.

Outline


Case Study
Transfer Learning
Theory Building
Guided Sampling
Current Research

Looking further: When transfer learning goes wrong

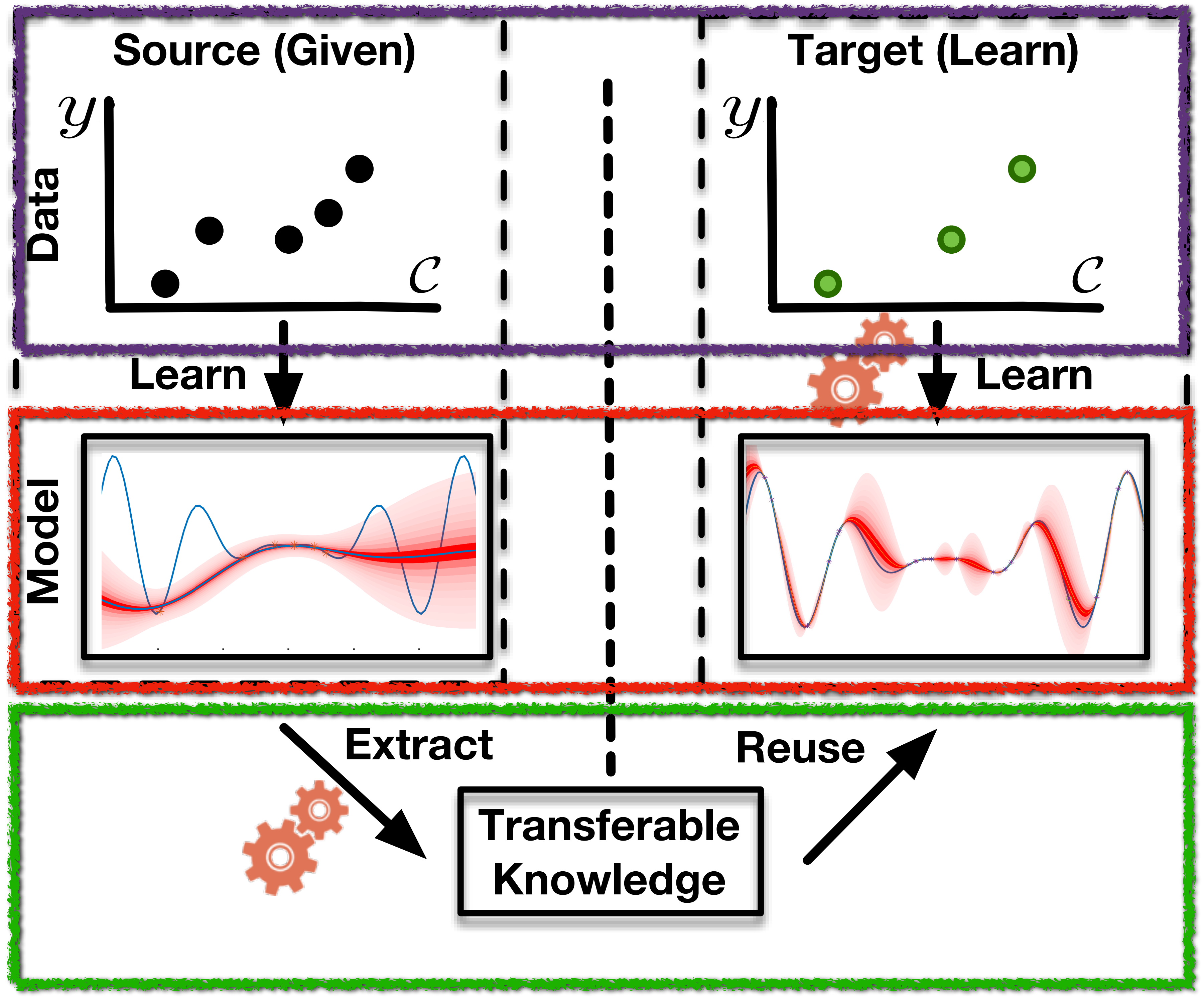
Insight: Predictions become more accurate when the source is **more related** to the target.



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!
It didn't!
number of refinements
number of particles
(a)
(b)
(c)
(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?

We hypothesized that we can exploit similarities across environments to learn "cheaper" performance models

Source Environment (Execution time of Program X)

$O_1 \times O_2 \times \cdots \times O_{19} \times O_{20}$

$$
\begin{array}{c}
c_1 \\ c_2 \\ c_3 \\ \\ \\ c_n
\end{array}
\boxed{\begin{array}{c}
0 \times 0 \times \cdots \times 0 \times 1 \\
0 \times 0 \times \cdots \times 1 \times 0 \\
0 \times 0 \times \cdots \times 1 \times 1 \\
\cdots \\
1 \times 1 \times \cdots \times 1 \times 0 \\
1 \times 1 \times \cdots \times 1 \times 1
\end{array}}
\boxed{\begin{array}{c}
y_{s1} \\ y_{s2} \\ y_{s3} \\ \\ \\ y_{sn}
\end{array}}
\begin{array}{l}
= f_s(c_1) \\ = f_s(c_2) \\ = f_s(c_3) \\ \\ \\ = f_s(c_n)
\end{array}
$$

Similarity

Target Environment (Execution time of Program Y)

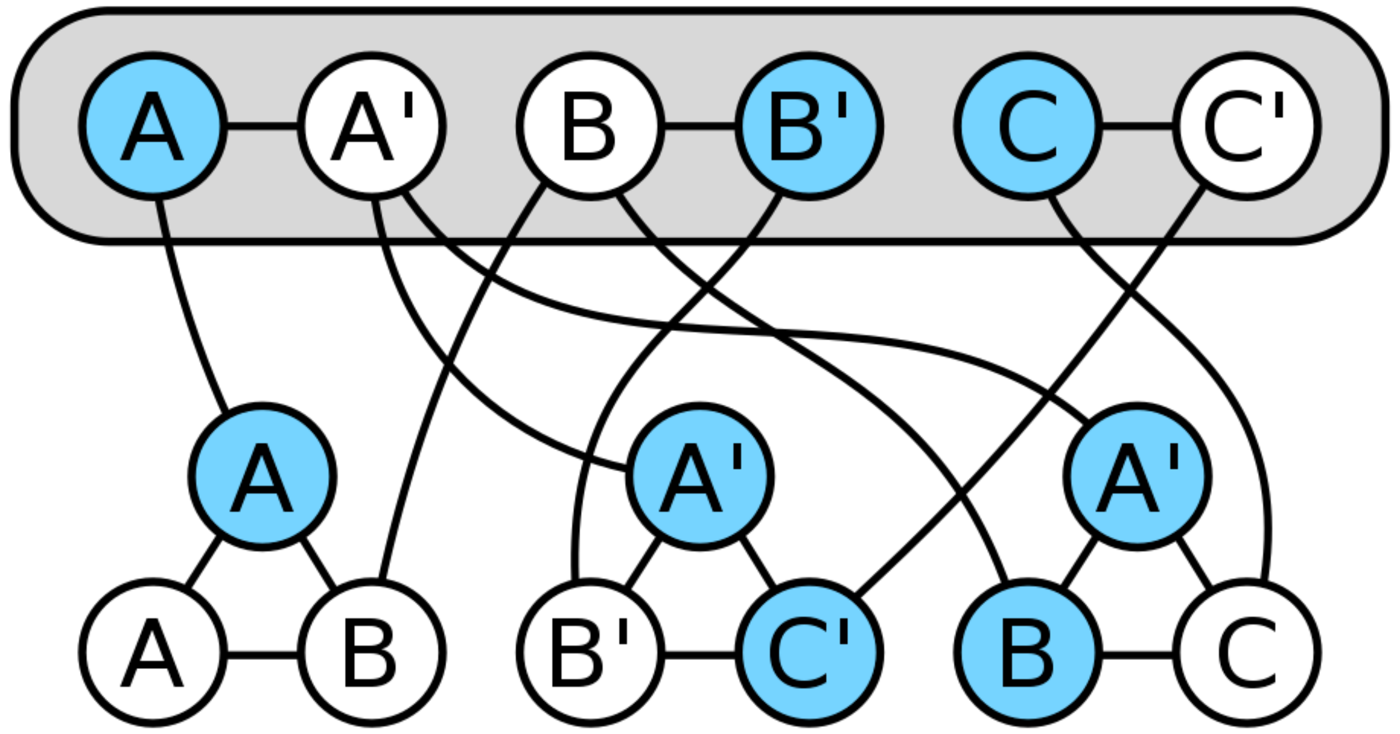
$O_1 \times O_2 \times \cdots \times O_{19} \times O_{20}$

$$
\boxed{\begin{array}{c}
0 \times 0 \times \cdots \times 0 \times 1 \\
0 \times 0 \times \cdots \times 1 \times 0 \\
0 \times 0 \times \cdots \times 1 \times 1 \\
\cdots \\
1 \times 1 \times \cdots \times 1 \times 0 \\
1 \times 1 \times \cdots \times 1 \times 1
\end{array}}
\boxed{\begin{array}{c}
y_{t1} \\ y_{t2} \\ y_{t3} \\ \\ \\ y_{tn}
\end{array}}
\begin{array}{l}
= f_t(c_1) \\ = f_t(c_2) \\ = f_t(c_3) \\ \\ \\ = f_t(c_n)
\end{array}
$$

[P. Jamshidi, et al., "**Transfer learning for performance modeling of configurable systems**….", ASE'17]

Our empirical study: We looked at different highly-configurable systems to gain insights

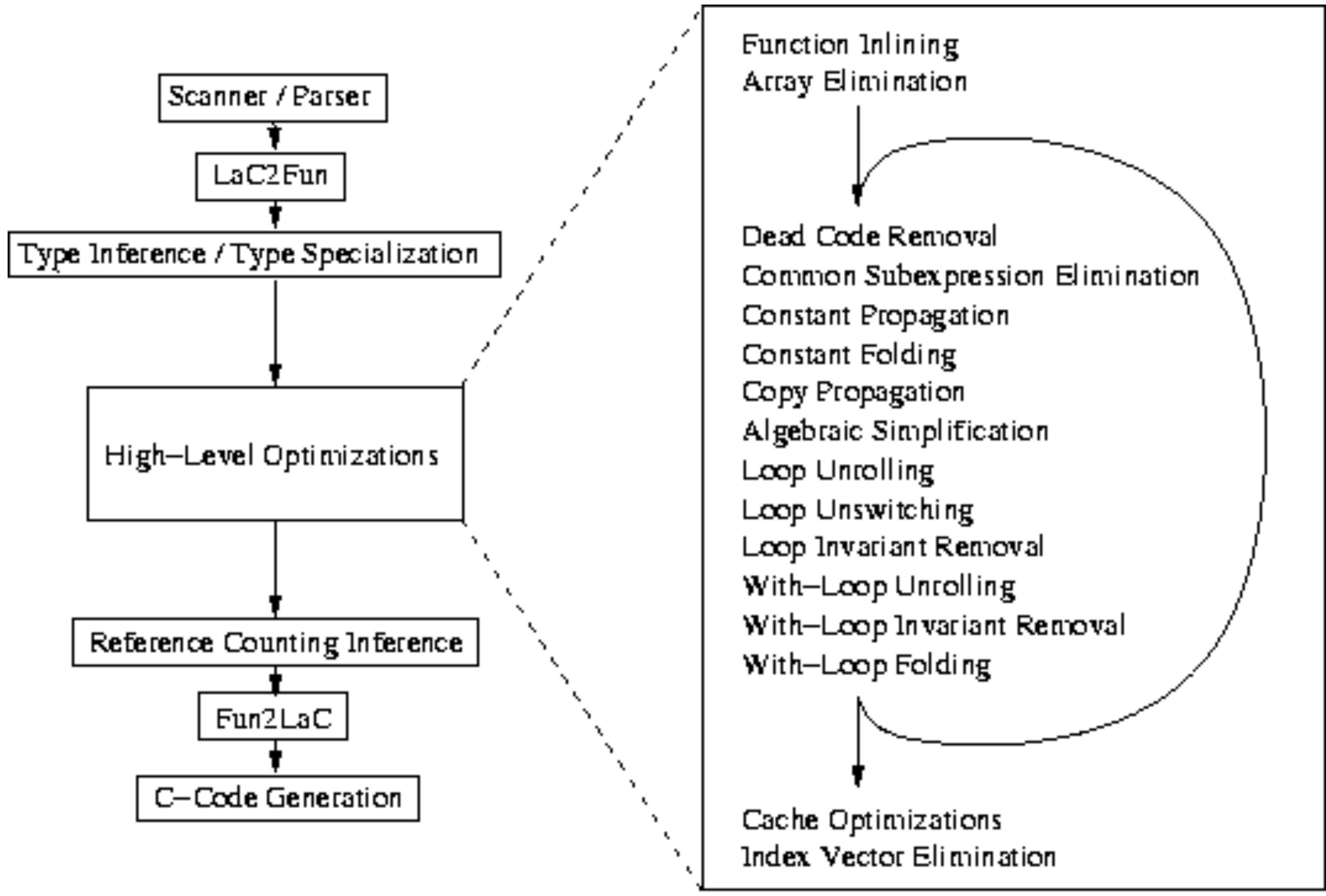
$(A \vee B) \wedge (\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee B \vee C)$



SPEAR (SAT Solver)
Analysis time
14 options

X264 (video encoder)
Encoding time
16 options

SQLite (DB engine)
Query time
14 options

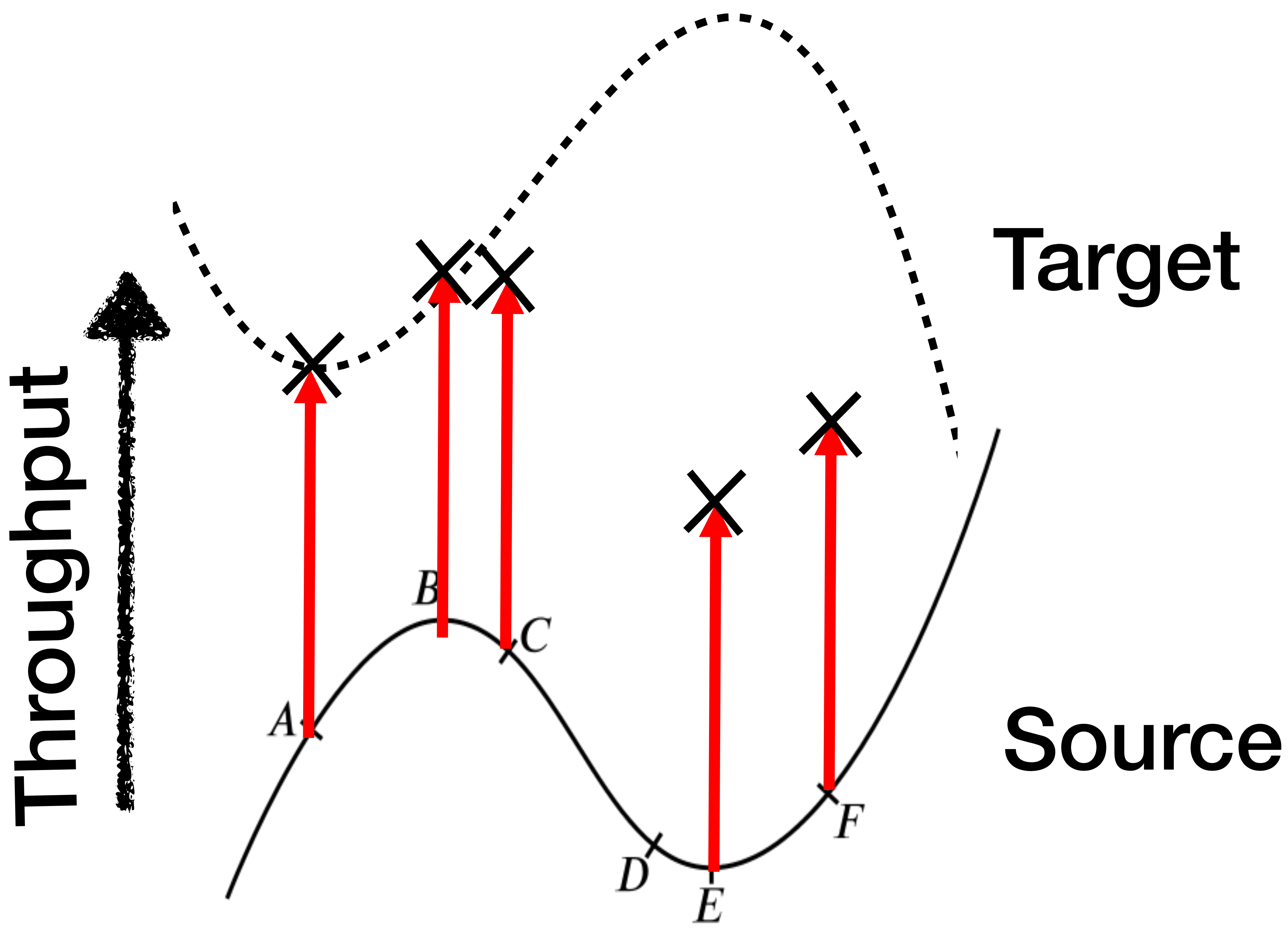


SaC (Compiler)
Execution time
50 options

[P. Jamshidi, et al., “**Transfer learning for performance modeling of configurable systems**….”, ASE’17]

Linear shift happens only in limited environmental changes

Soft	Environmental change	Severity	Corr.
SPEAR	NUC/2 -> NUC/4	Small	1.00
	Amazon_nano -> NUC	Large	0.59
	Hardware/workload/version	V Large	-0.10
x264	Version	Large	0.06
	Workload	Medium	0.65
SQLite	write-seq -> write-batch	Small	0.96
	read-rand -> read-seq	Medium	0.50



Implication: Simple transfer learning is limited to hardware changes in practice

Influential options and interactions are preserved across environments

Soft	Environmental change	Severity	Dim	t-test	
x264	Version	Large	16	12	10
	Hardware/workload/ver	V Large		8	9
SQLite	write-seq -> write-batch	V Large	14	3	4
	read-rand -> read-seq	Medium		1	1
SaC	Workload	V Large	50	16	10

We only need to explore part of the space:

$$\frac{2^{16}}{2^{50}} = 0.000000000058$$

Implication: Avoid wasting budget on non-informative part of configuration space and focusing where it matters.

Transfer learning across environment

Source
(Execution time of Program X)

$$O_1 \times O_2 \times \cdots \times O_{19} \times O_{20}$$

$$\begin{array}{l|l|l}
c_1 & 0 \times 0 \times \cdots \times 0 \times 1 & y_{s1} = f_s(c_1) \\
c_2 & 0 \times 0 \times \cdots \times 1 \times 0 & y_{s2} = f_s(c_2) \\
c_3 & 0 \times 0 \times \cdots \times 1 \times 1 & y_{s3} = f_s(c_3) \\
 & \cdots & \\
 & 1 \times 1 \times \cdots \times 1 \times 0 & \\
c_n & 1 \times 1 \times \cdots \times 1 \times 1 & y_{sn} = f_s(c_n)
\end{array}$$

$$\xrightarrow{\text{Learn performance model}} \quad \hat{f}_s \sim f_s(\cdot)$$

Observation 1: Not all options and interactions are influential and interactions degree between options are not high

$$\mathbb{C} = O_1 \times O_2 \times O_3 \times O_4 \times O_5 \times O_6 \times O_7 \times O_8 \times O_9 \times O_{10}$$

$$\hat{f}_s(\cdot) = 1.2 + 3\boxed{o_1} + 5\boxed{o_3} + 0.9\boxed{o_7} + 0.8\boxed{o_3 o_7} + 4\boxed{o_1 o_3 o_7}$$

Observation 2: Influential options and interactions are preserved across environments

$$\hat{f}_s(\cdot) = 1.2 + 3o_1 + 5o_3 + 0.9o_7 + 0.8o_3o_7 + 4o_1o_3o_7$$

$$\hat{f}_t(\cdot) = 10.4 - 2.1o_1 + 1.2o_3 + 2.2o_7 + 0.1o_1o_3 - 2.1o_3o_7 + 14o_1o_3o_7$$

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**

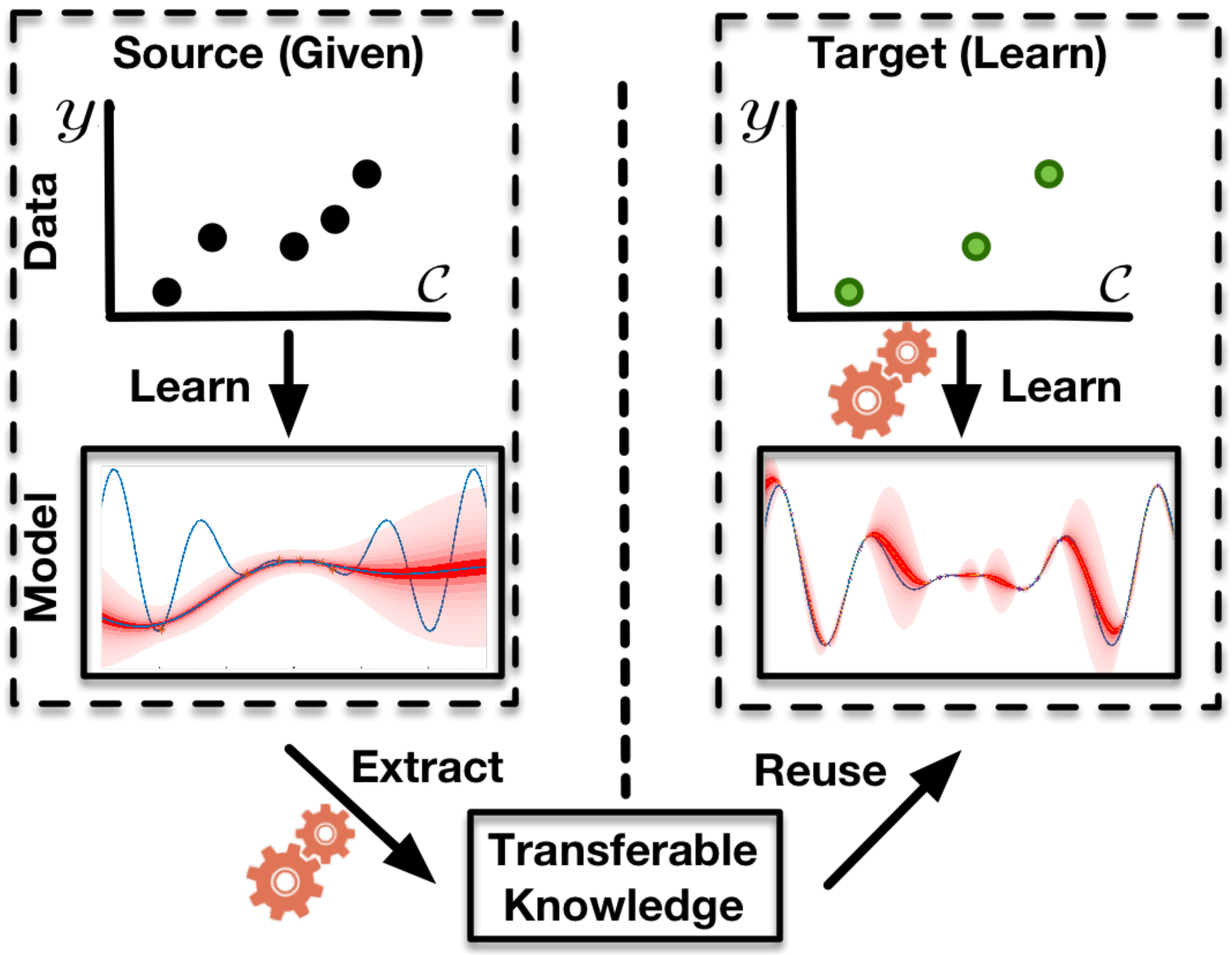


Fig. 1: Transfer learning is a form of machine learning that takes advantage of transferable knowledge from source to learn an accurate, reliable, and less costly model for the target environment.

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

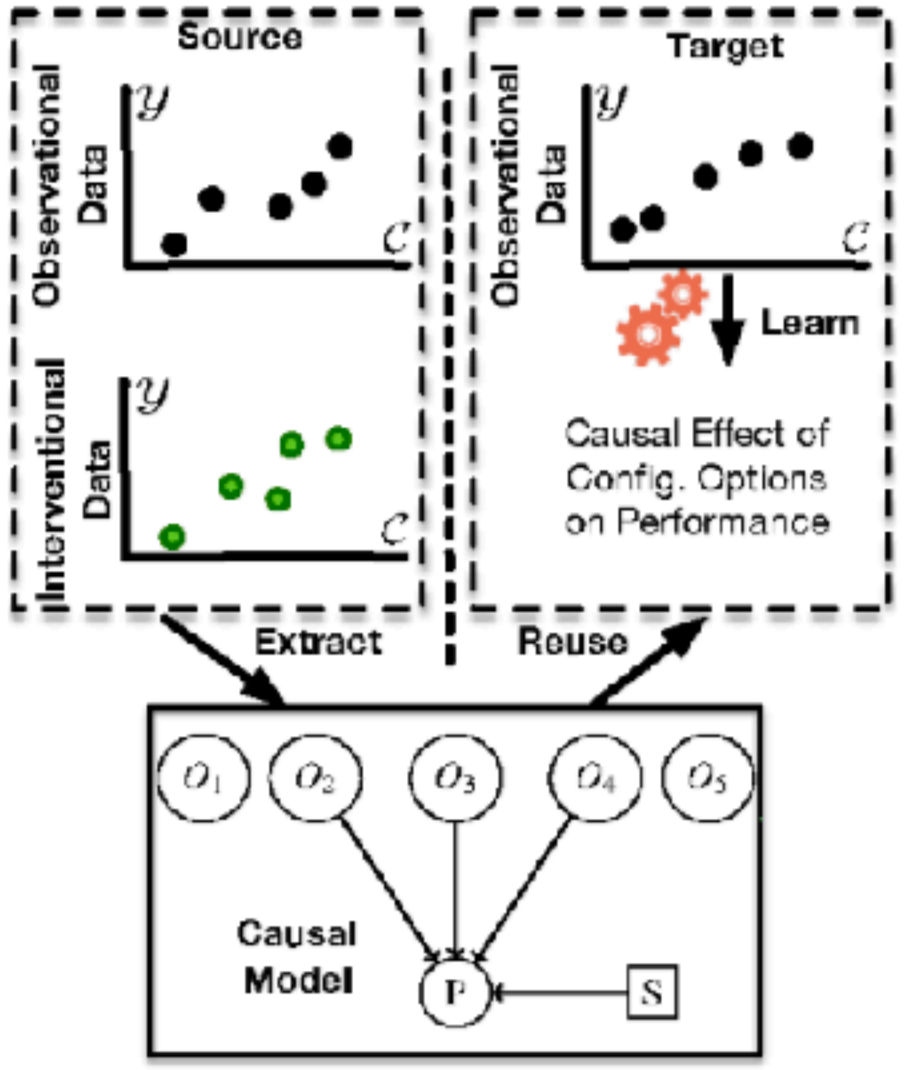
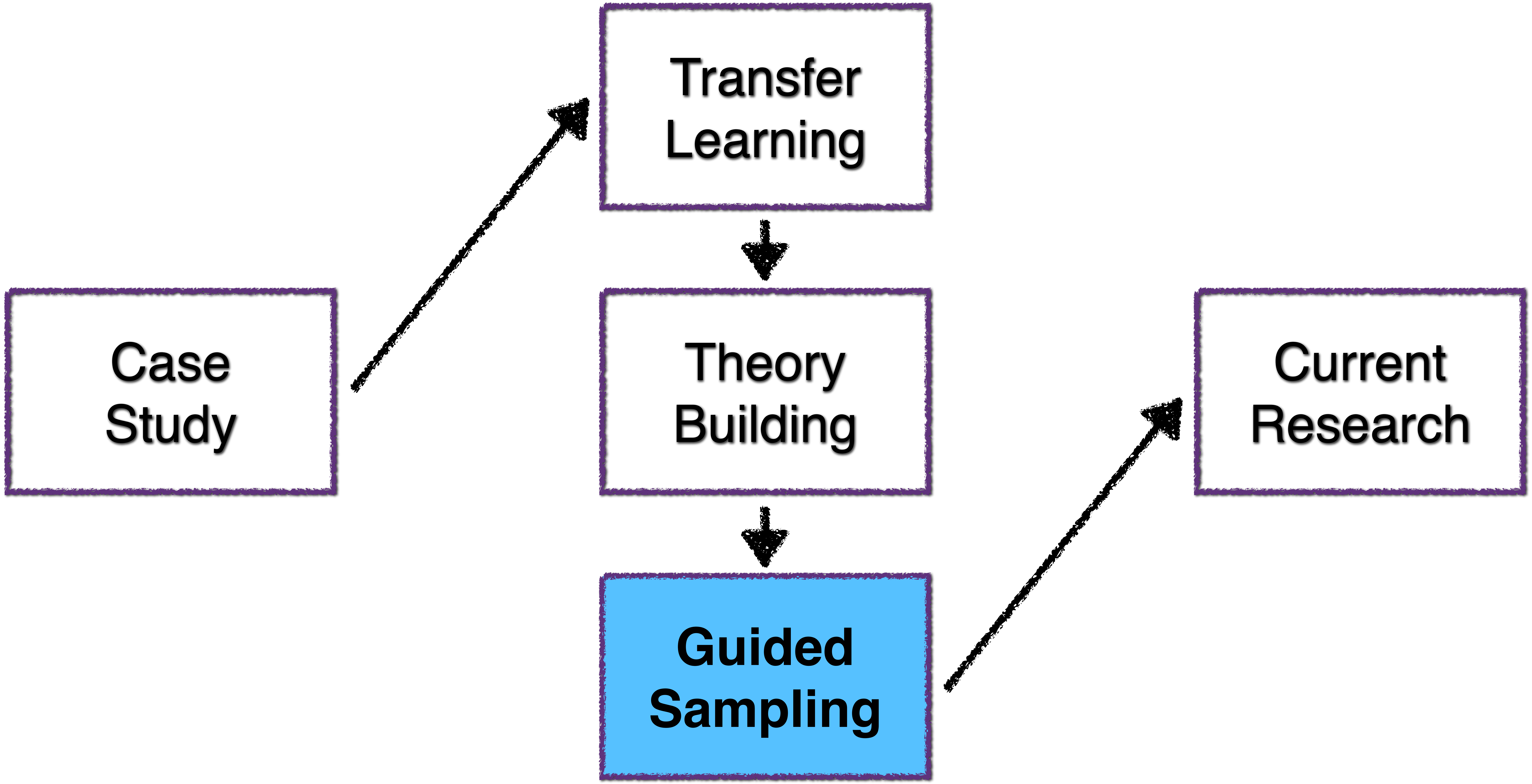


Figure 1: Exploiting causal inference for performance analysis.

Outline


Transfer Learning
Case Study
Theory Building
Current Research
Guided Sampling

How to sample the configuration space to learn a “better” performance behavior?

How to select the most informative configurations?

The similarity across environment is a rich source of knowledge for exploration of the configuration space

When we treat the system as black boxes, we cannot typically distinguish between different configurations

$$
\begin{array}{cc}
 & O_1 \times O_2 \times \cdots \times O_{19} \times O_{20} \\
c_1 & \boxed{\begin{array}{c} 0 \times 0 \times \cdots \times 0 \times 1 \\ 0 \times 0 \times \cdots \times 1 \times 0 \\ 0 \times 0 \times \cdots \times 1 \times 1 \\ \cdots \\ 1 \times 1 \times \cdots \times 1 \times 0 \\ 1 \times 1 \times \cdots \times 1 \times 1 \end{array}}
\end{array}
$$

c_1: $0 \times 0 \times \cdots \times 0 \times 1$

c_2: $0 \times 0 \times \cdots \times 1 \times 0$

c_3: $0 \times 0 \times \cdots \times 1 \times 1$

$\cdots$

$1 \times 1 \times \cdots \times 1 \times 0$

c_n: $1 \times 1 \times \cdots \times 1 \times 1$

- We therefore end up blindly explore the configuration space

- That is essentially the key reason why “most” work in this area consider random sampling.

Without considering this knowledge, many samples may not provide new information

$$\hat{f}_s(\cdot) = 1.2 + 3\boxed{o_1} + 5\boxed{o_3} + 0.9\boxed{o_7} + 0.8\boxed{o_3 o_7} + 4\boxed{o_1 o_3 o_7}$$

$$\begin{array}{ll}
 & O_1 \times O_2 \times O_3 \times O_4 \times O_5 \times O_6 \times O_7 \times O_8 \times O_9 \times O_{10} \\
c_1 & 1 \times 0 \times 1 \times 0 \times 0 \times 0 \times 1 \times 0 \times 0 \times 0 \\
c_2 & 1 \times 0 \times 1 \times 0 \times 0 \times 0 \times 1 \times 0 \times 0 \times 1 \\
c_3 & 1 \times 0 \times 1 \times 0 \times 0 \times 0 \times 1 \times 0 \times 1 \times 0 \\
 & \cdots \\
c_{128} & 1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 1
\end{array}$$

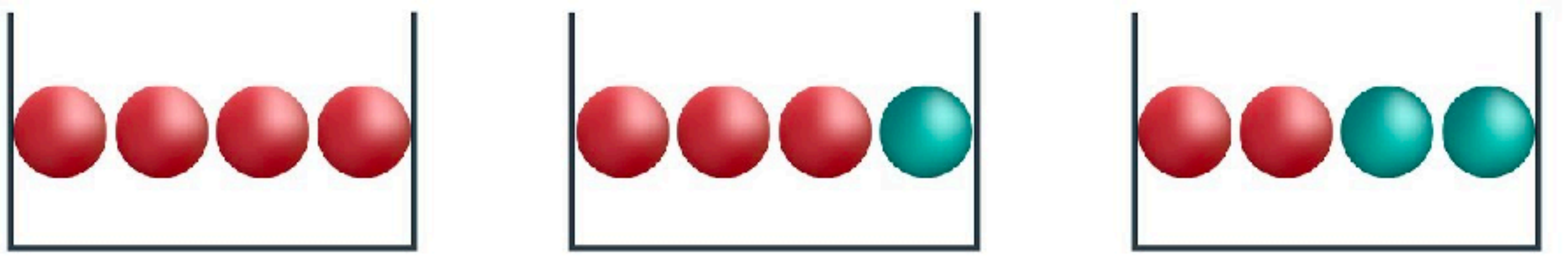
$\hat{f}_s(c_1) = 14.9$

$\hat{f}_s(c_2) = 14.9$

$\hat{f}_s(c_3) = 14.9$

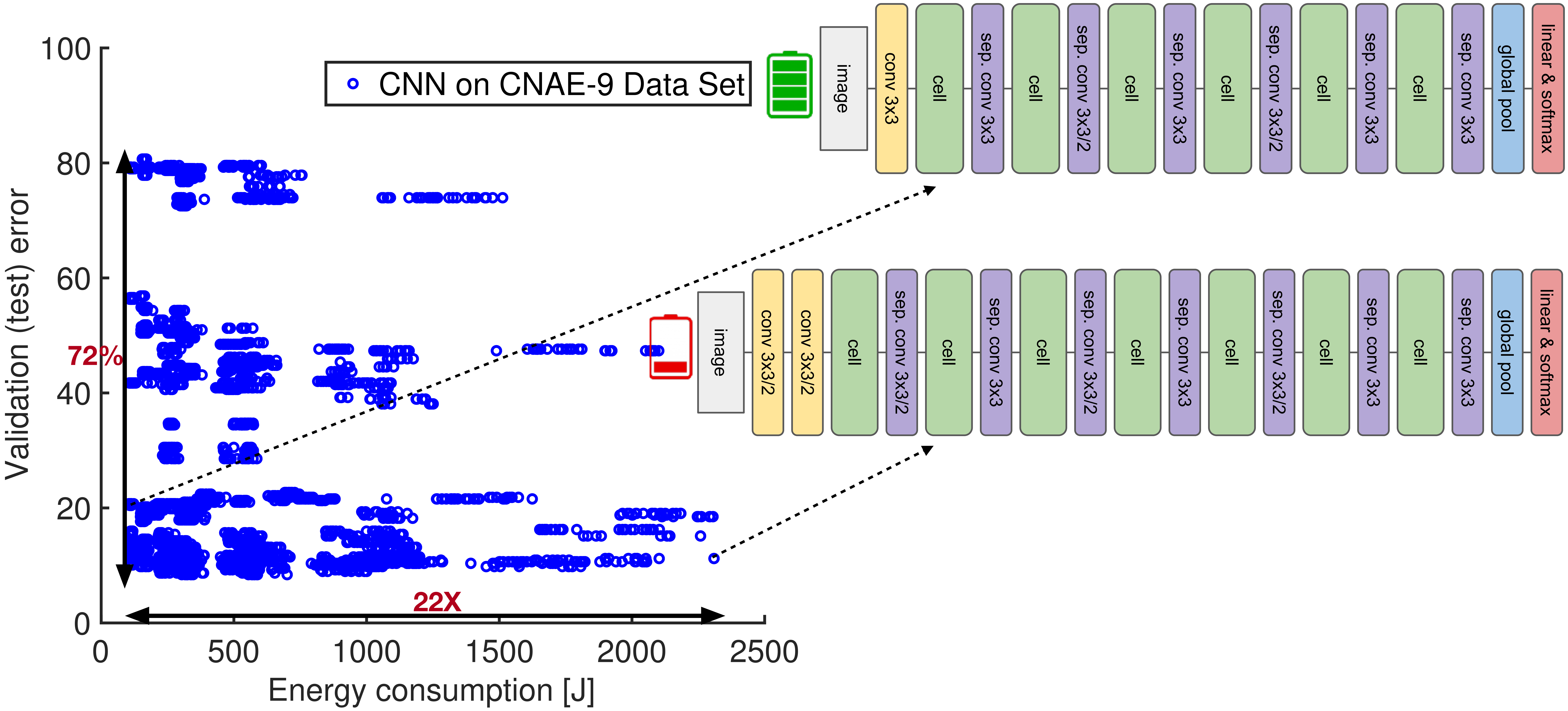
$\hat{f}_s(c_{128}) = 14.9$

Without knowing this knowledge, many blind/ random samples may not provide any additional information about performance of the system



Evaluation:
Learning performance behavior of Machine Learning Systems

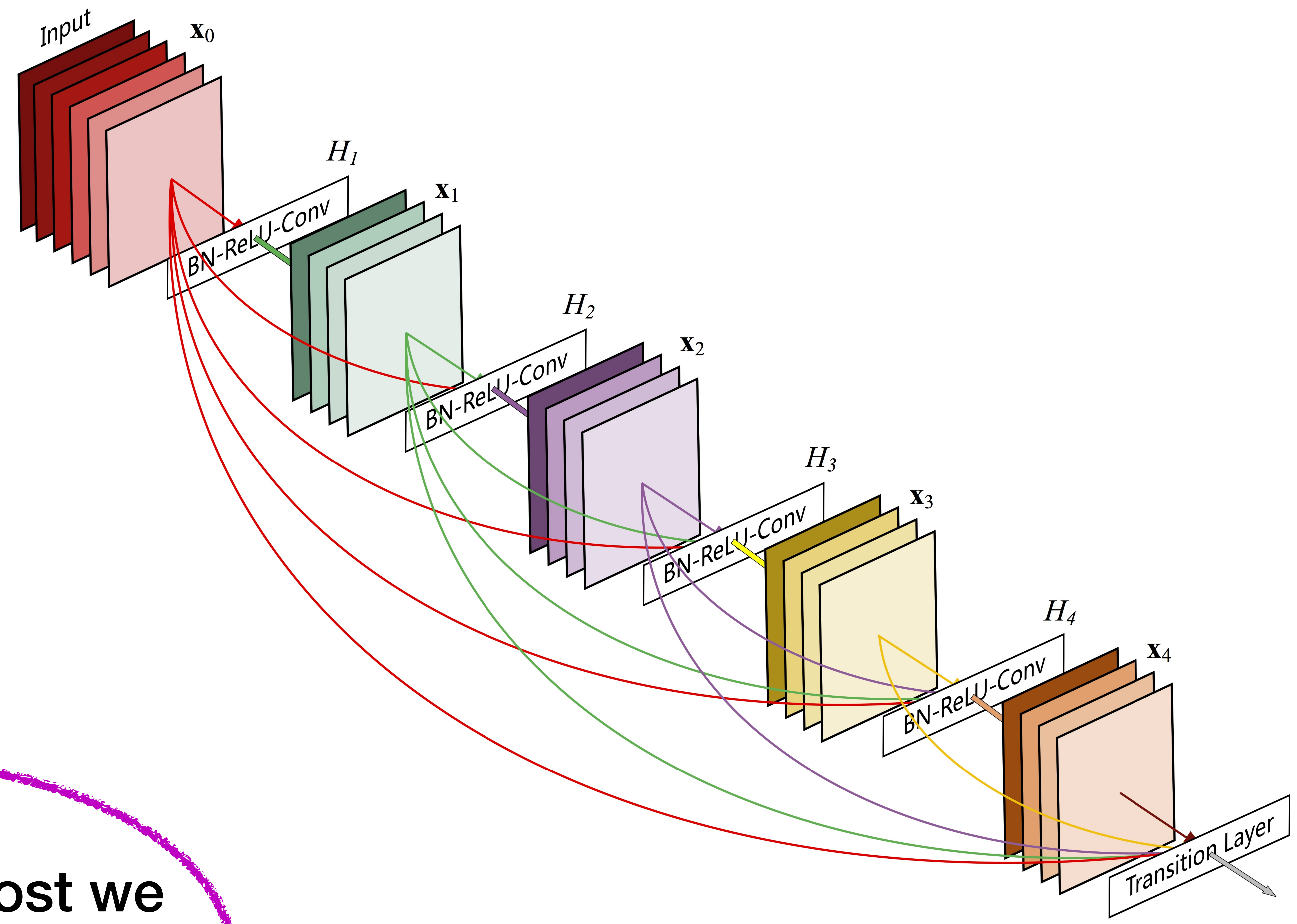
Configurations of deep neural networks affect accuracy and energy consumption



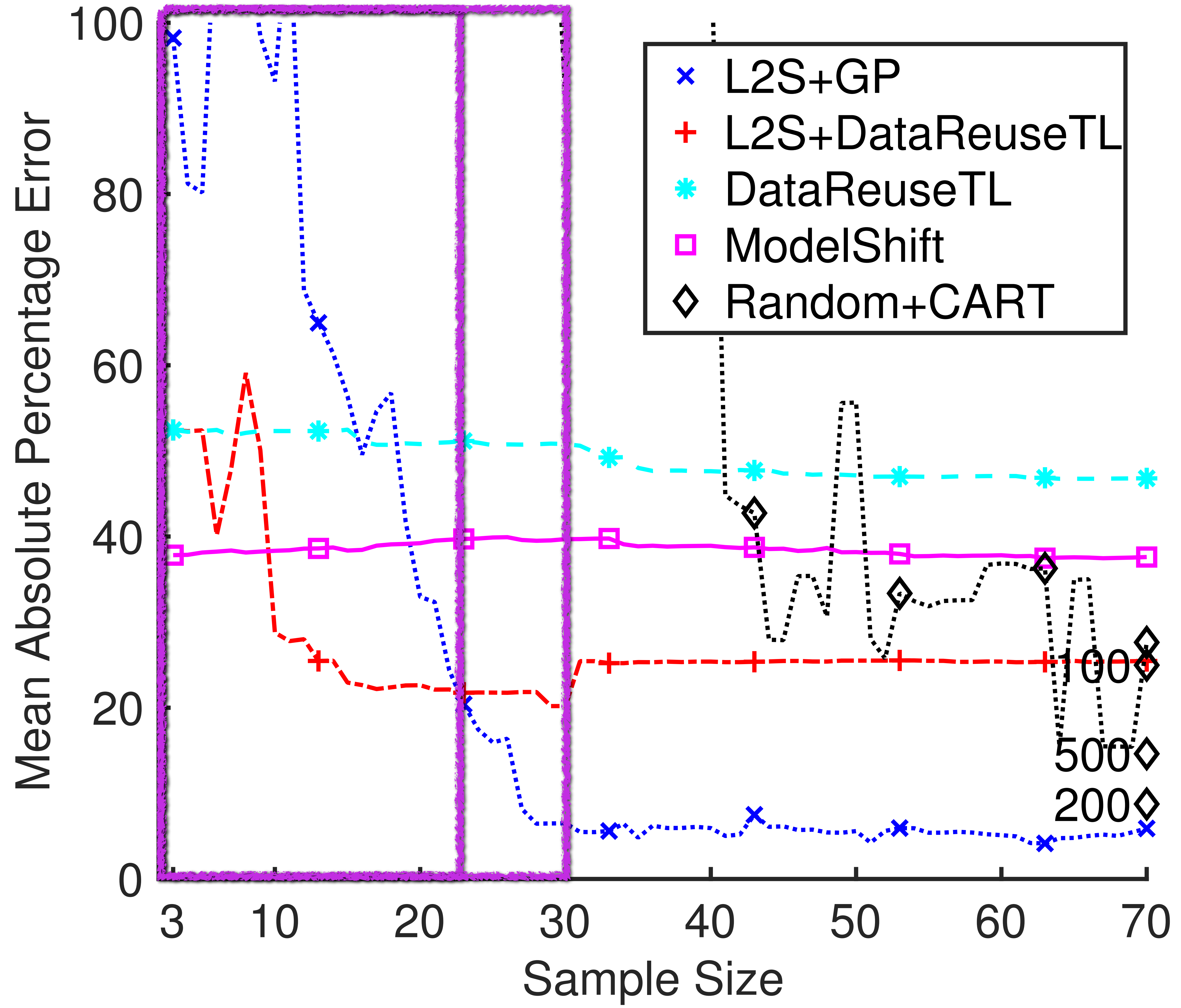
DNN measurements are costly

Each sample cost ~1h
4000 * 1h ~= 6 months

Yes, that's the cost we paid for conducting our measurements!

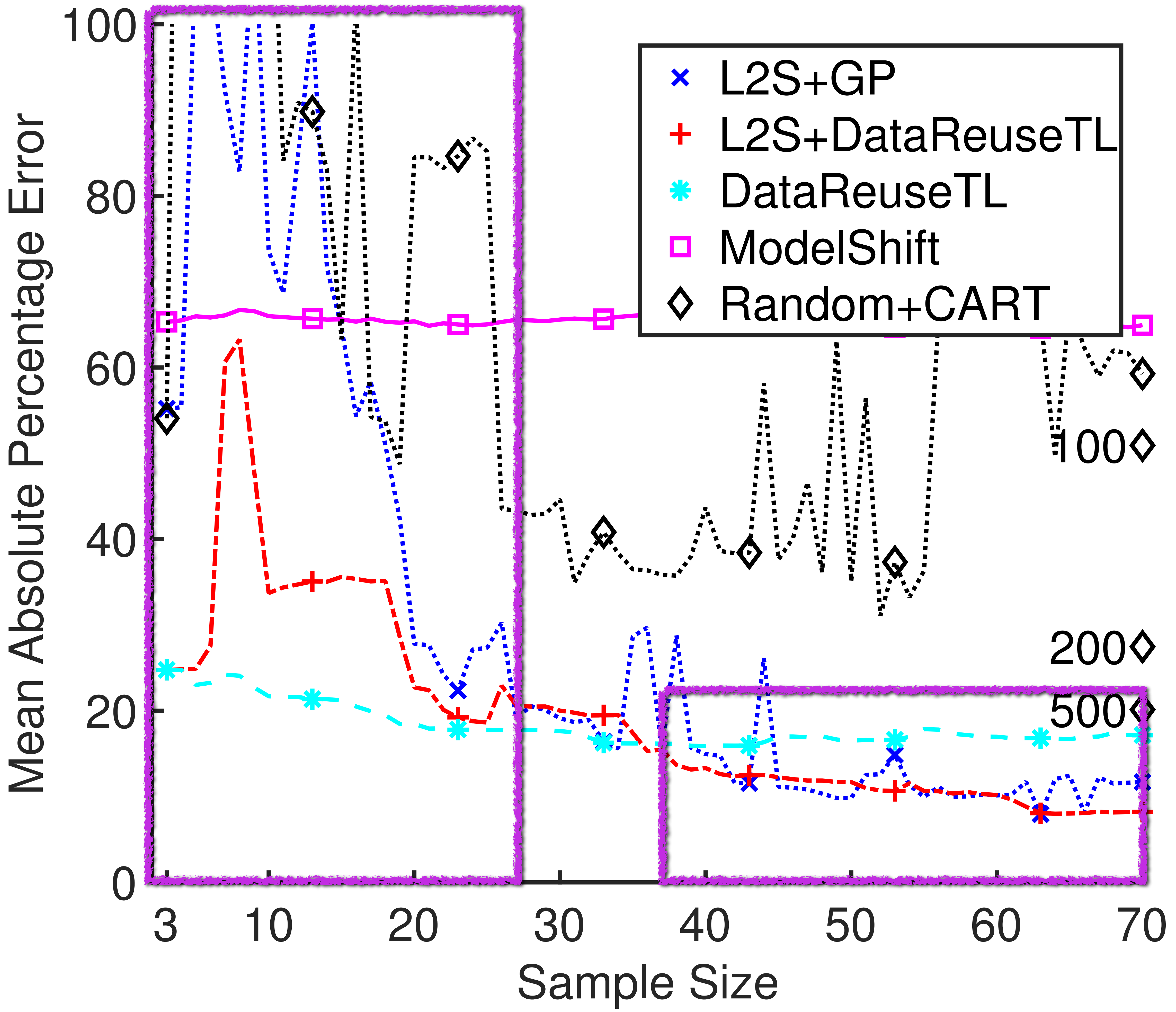


L2S enables learning a more accurate model with less samples exploiting the knowledge from the source



Convolutional Neural Network

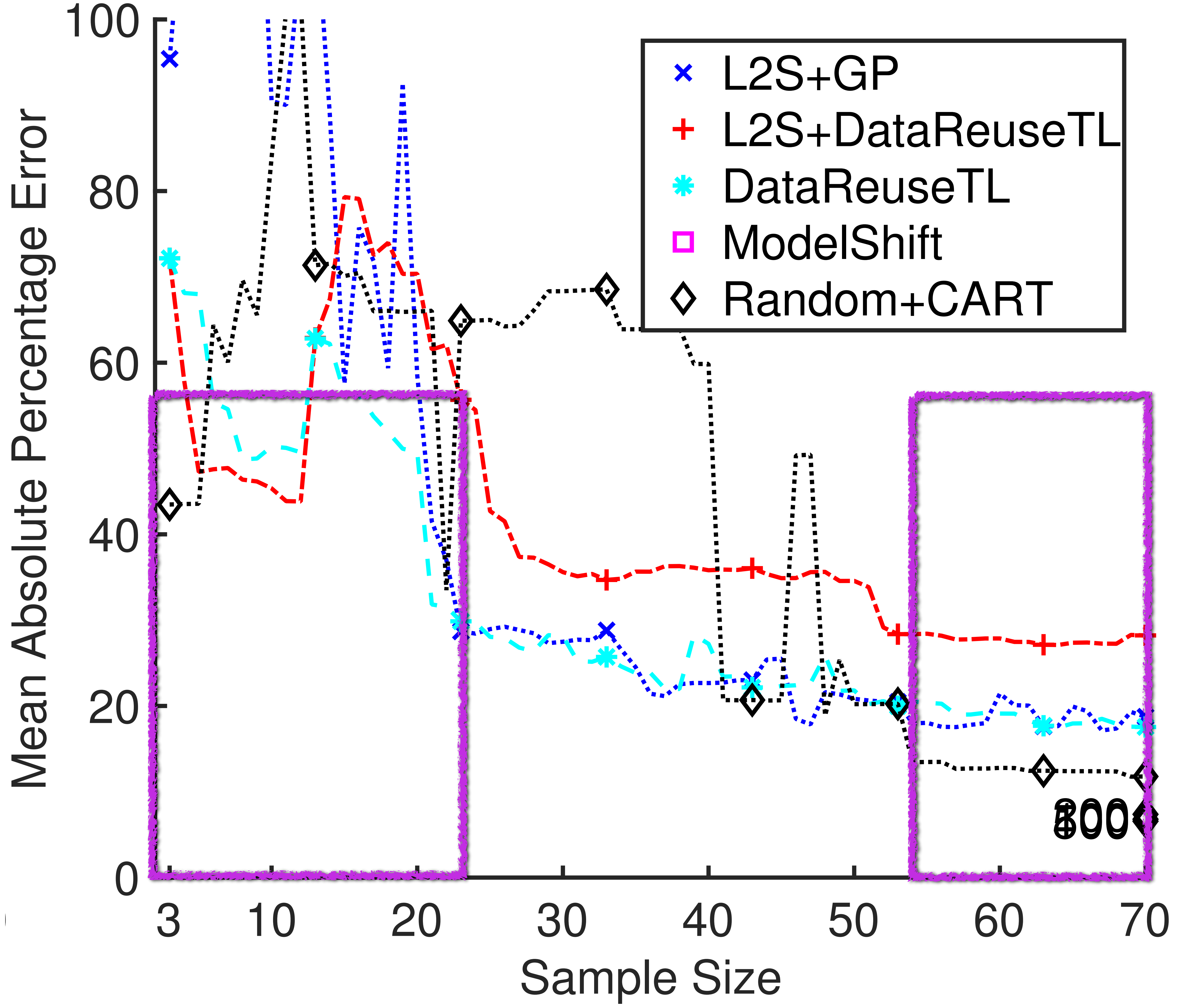
L2S may also help data-reuse approach to learn faster



XGBoost

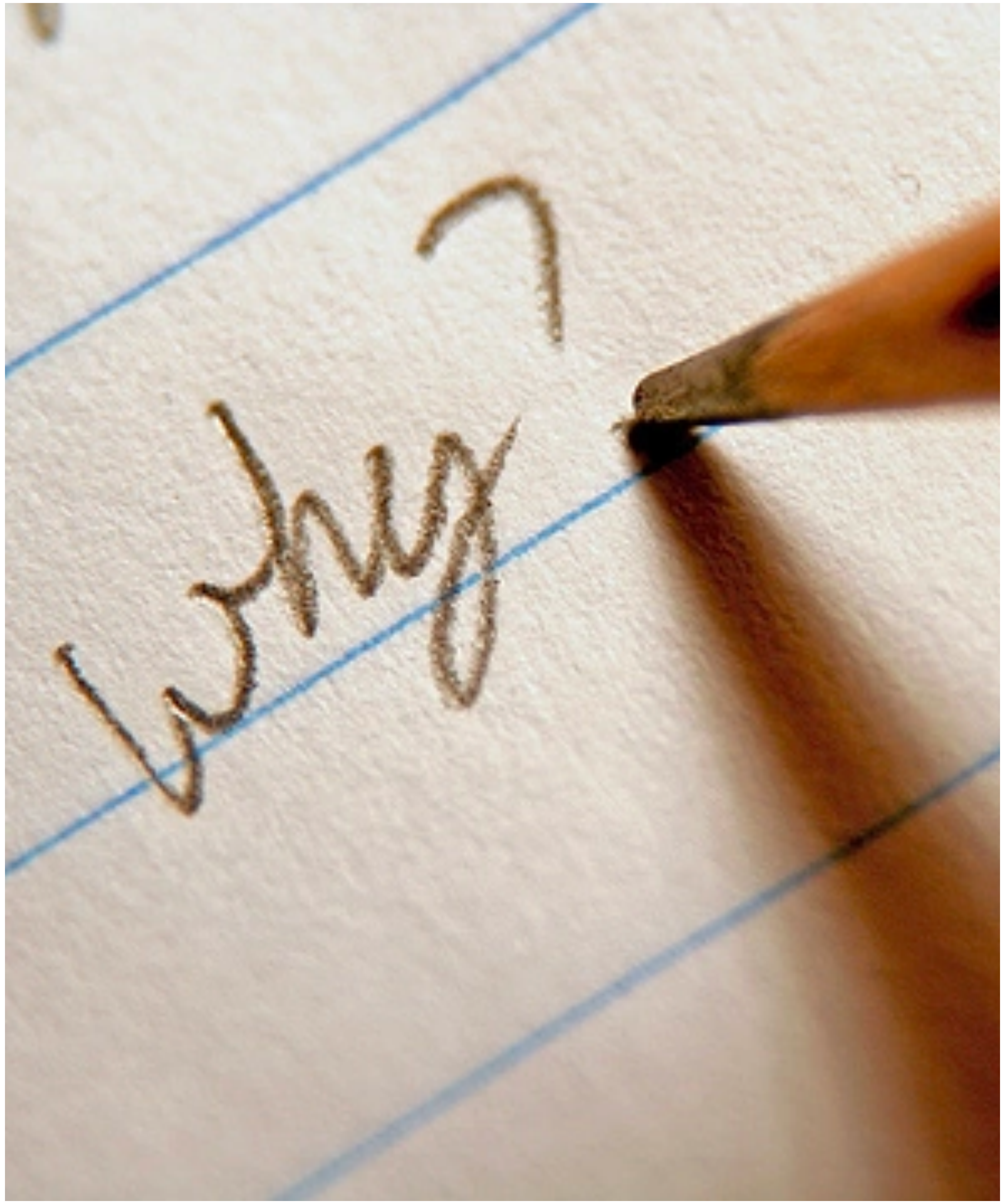
Evaluation:
Learning performance behavior of Big Data Systems

Some environments the similarities across environments may be too low and this results in “negative transfer”

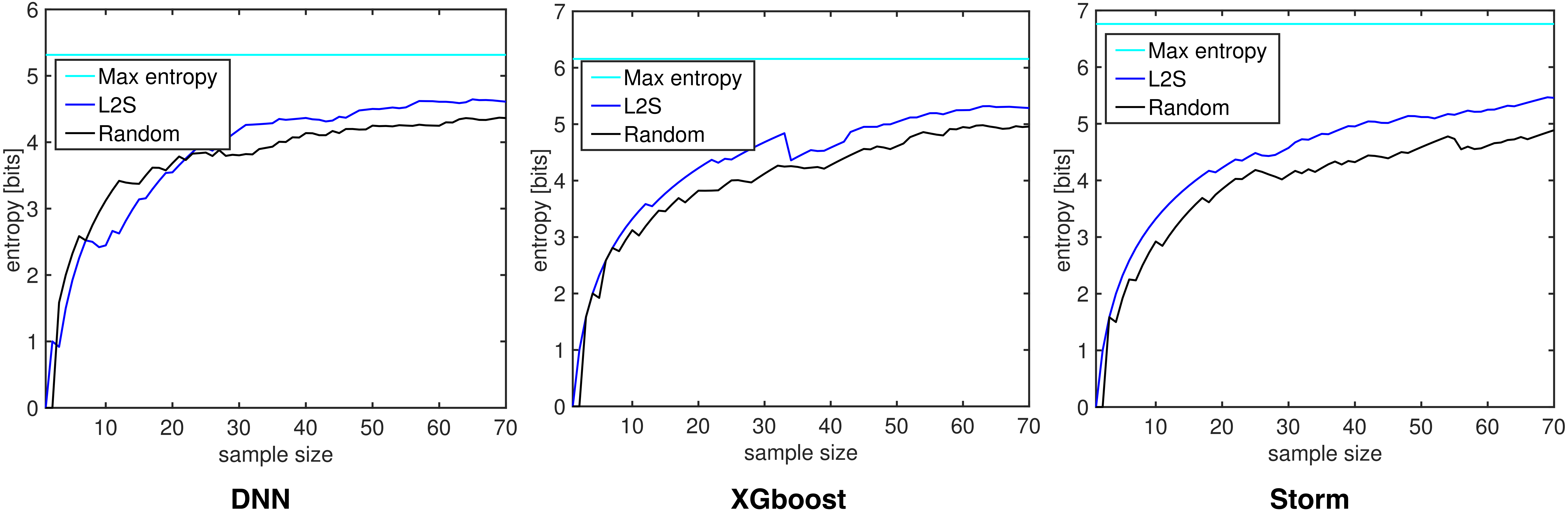


Apache Storm

Why performance models using L2S sample are more accurate?



The samples generated by L2S contains more information… “entropy <-> information gain”



Limitations

- Limited number of systems and environmental changes
    - Synthetic models
        - https://github.com/pooyanjamshidi/GenPerf
- Binary options
    - Non-binary options -> binary
- Negative transfer

Details: [FSE '18]

Learning to Sample: Exploiting Similarities across Environments to Learn Performance Models for Configurable Systems

Pooyan Jamshidi
University of South Carolina
USA

Miguel Velez
Christian Kästner
Carnegie Mellon University
USA

Norbert Siegmund
Bauhaus-University Weimar
Germany

ABSTRACT

Most software systems provide options that allow users to tailor the system in terms of functionality and qualities. The increased flexibility raises challenges for understanding the configuration space and the effects of options and their interactions on performance and other non-functional properties. To identify how options and interactions affect the performance of a system, several sampling and learning strategies have been recently proposed. However, existing approaches usually assume a fixed environment (hardware, workload, software release) such that learning has to be repeated once the environment changes. Repeating learning and measurement for each environment is expensive and often practically infeasible. Instead, we pursue a strategy that transfers knowledge across environments but sidesteps heavyweight and expensive transfer-

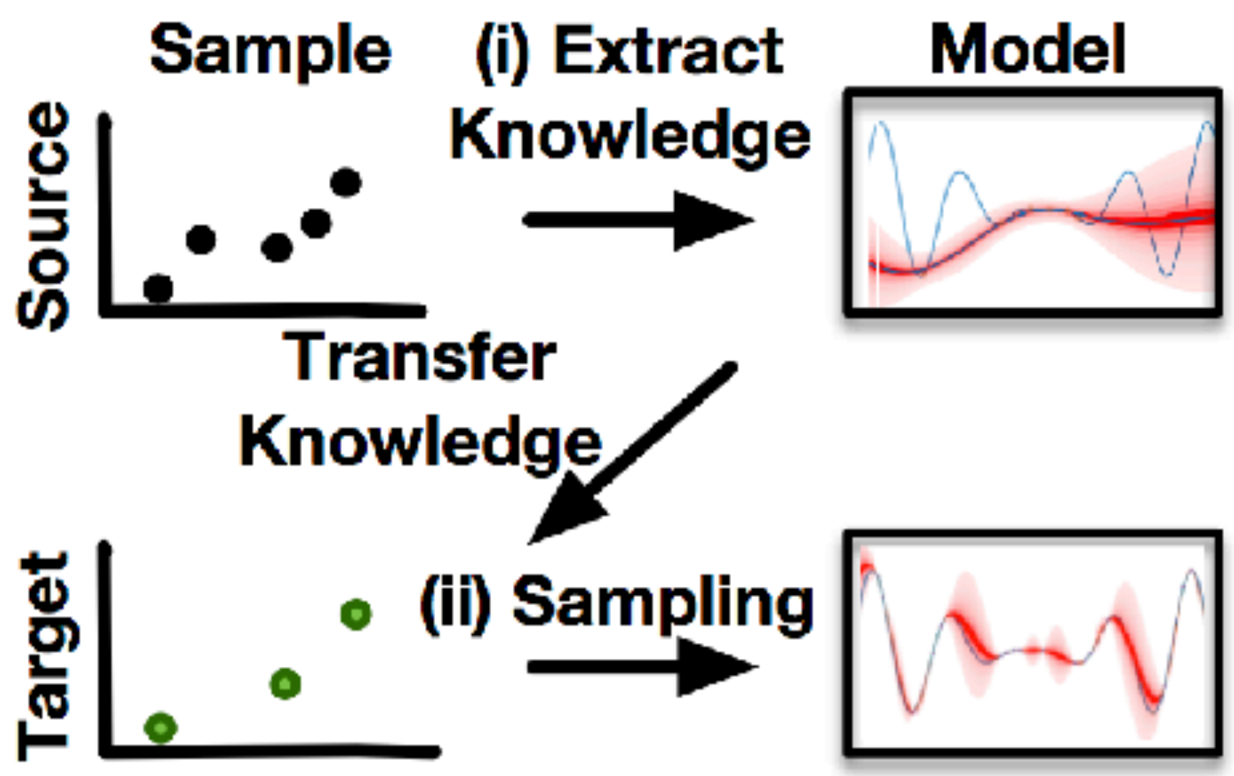
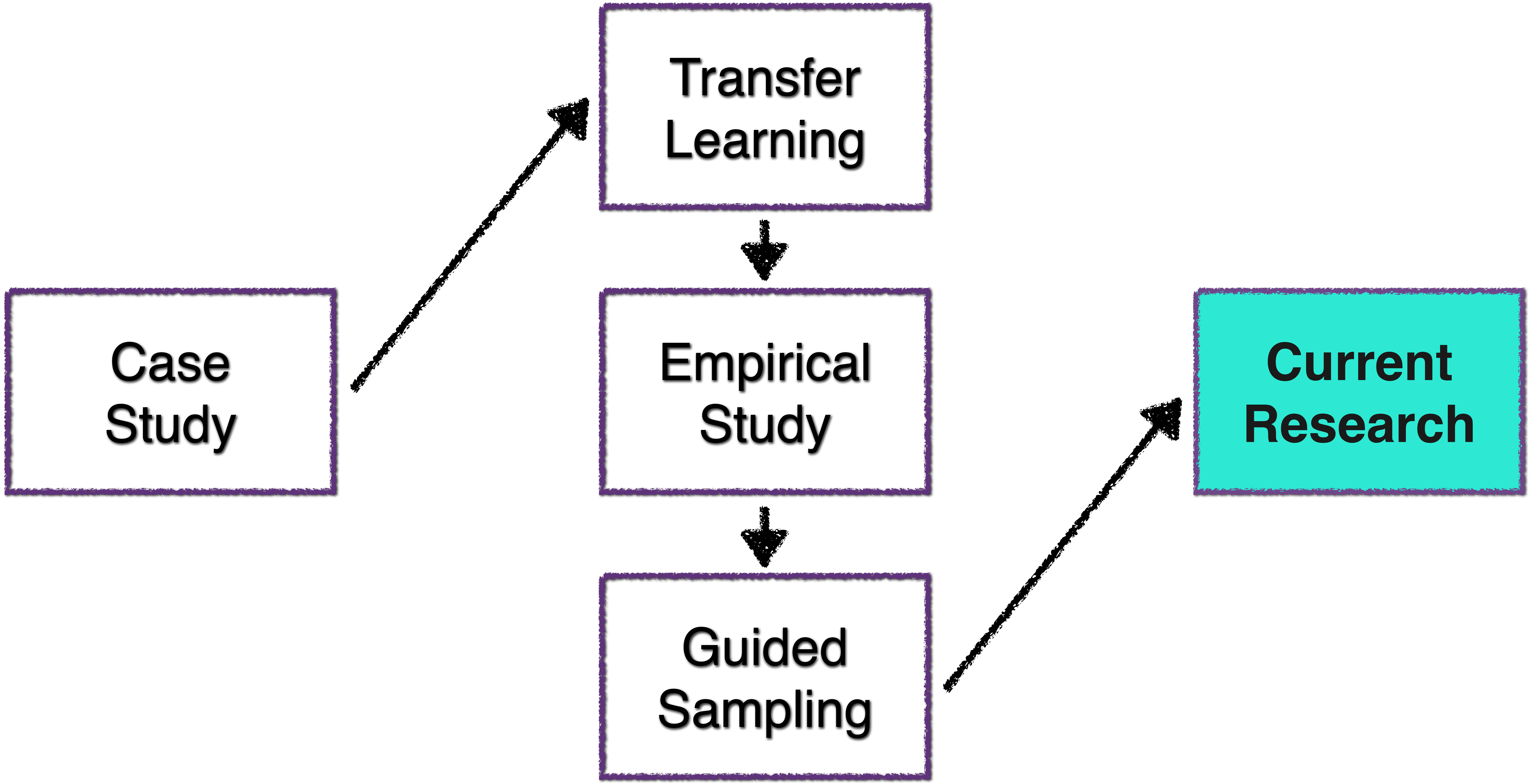


Figure 1: L2S performs guided sampling employing the knowledge extracted from a source environment.

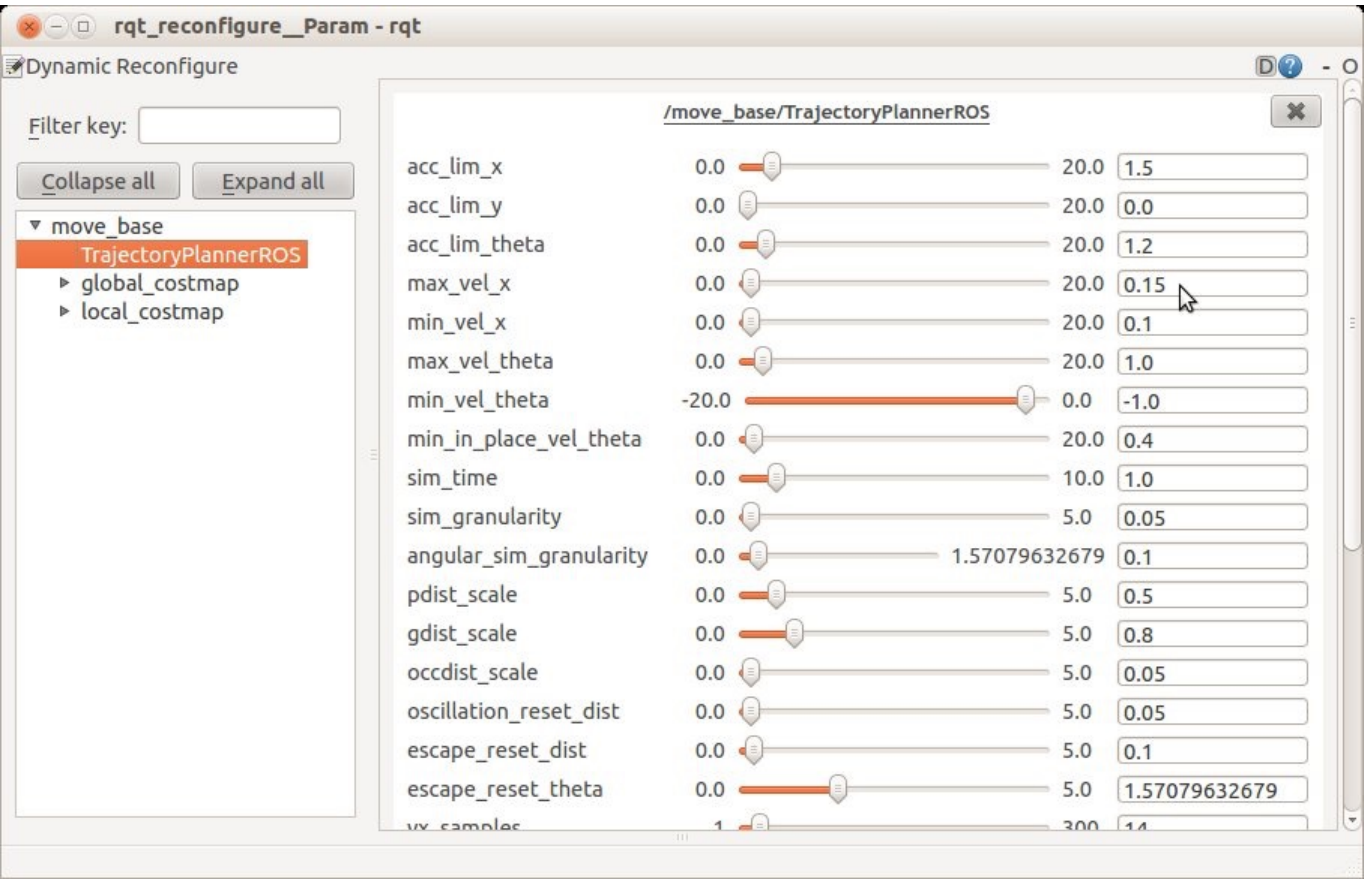
Outline

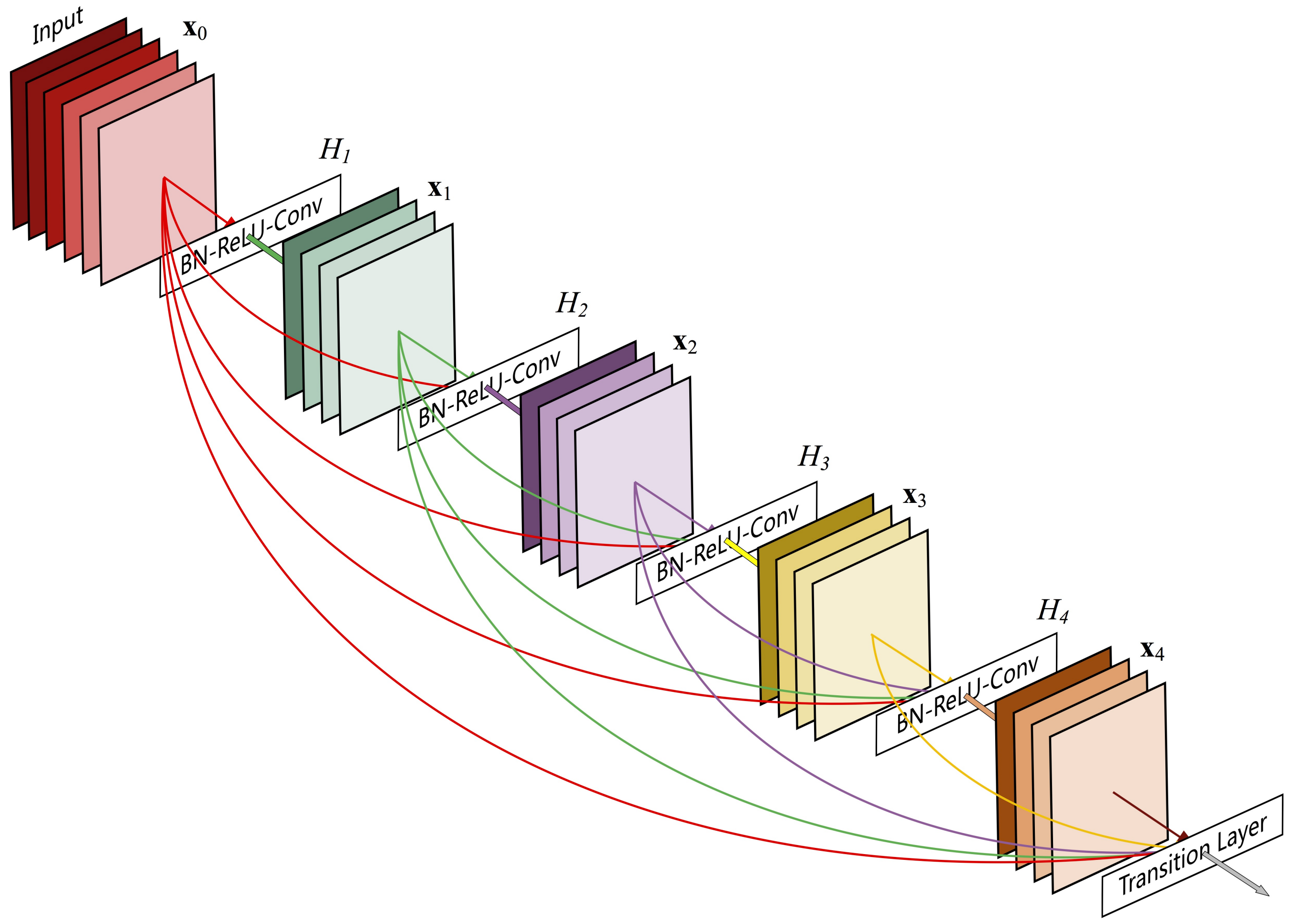

Transfer Learning
Case Study
Empirical Study
Current Research
Guided Sampling

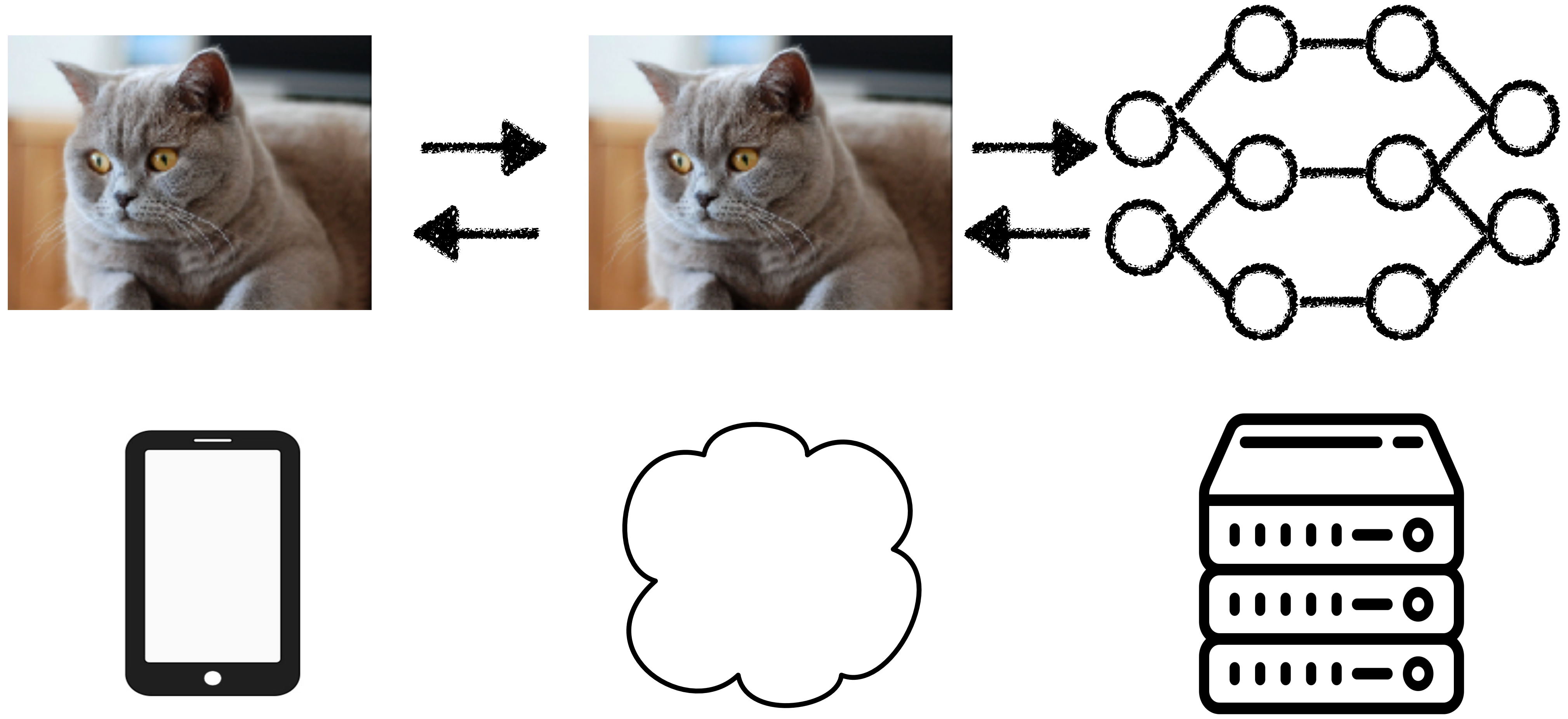

What will the software systems
of the future look like?

Software 2.0

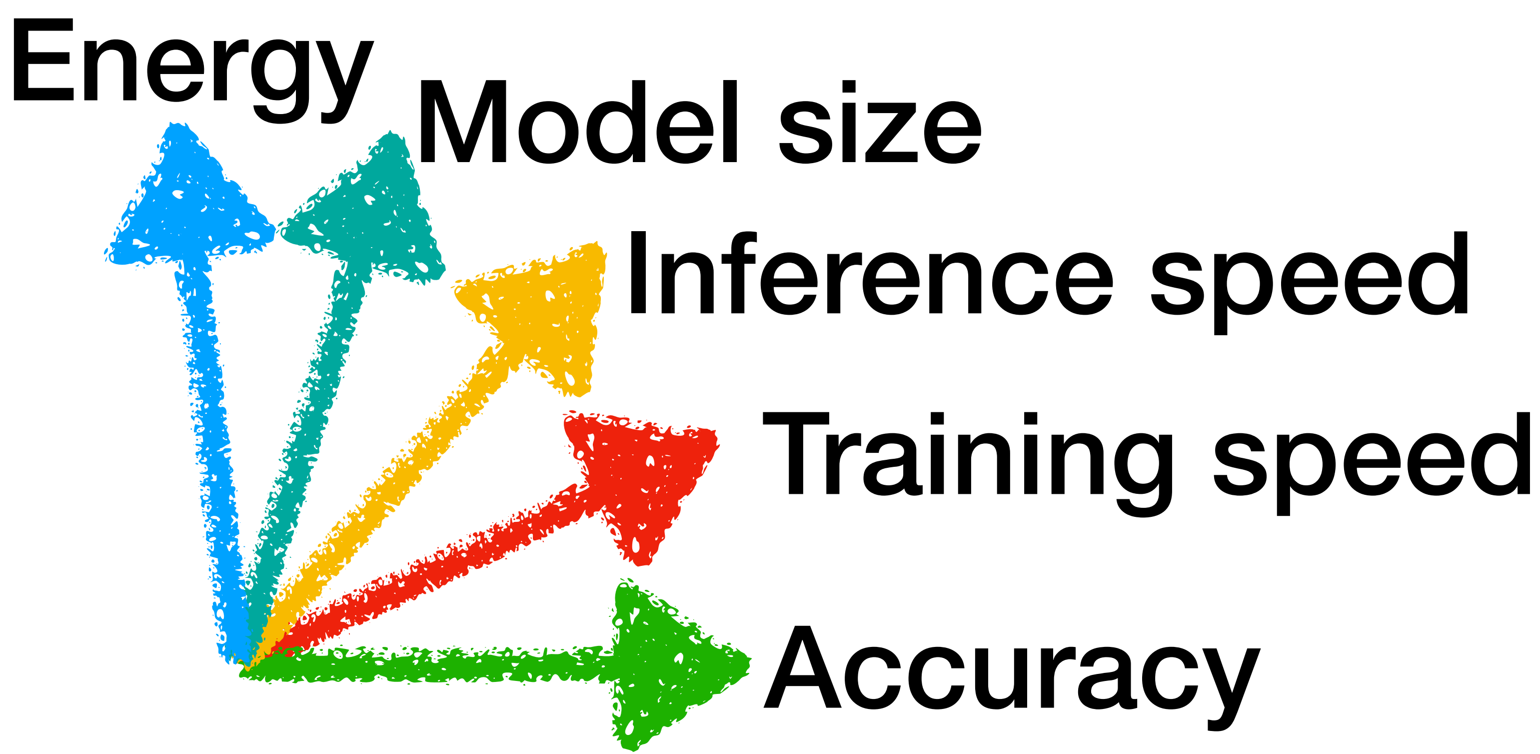
Increasingly **customized** and **configurable**



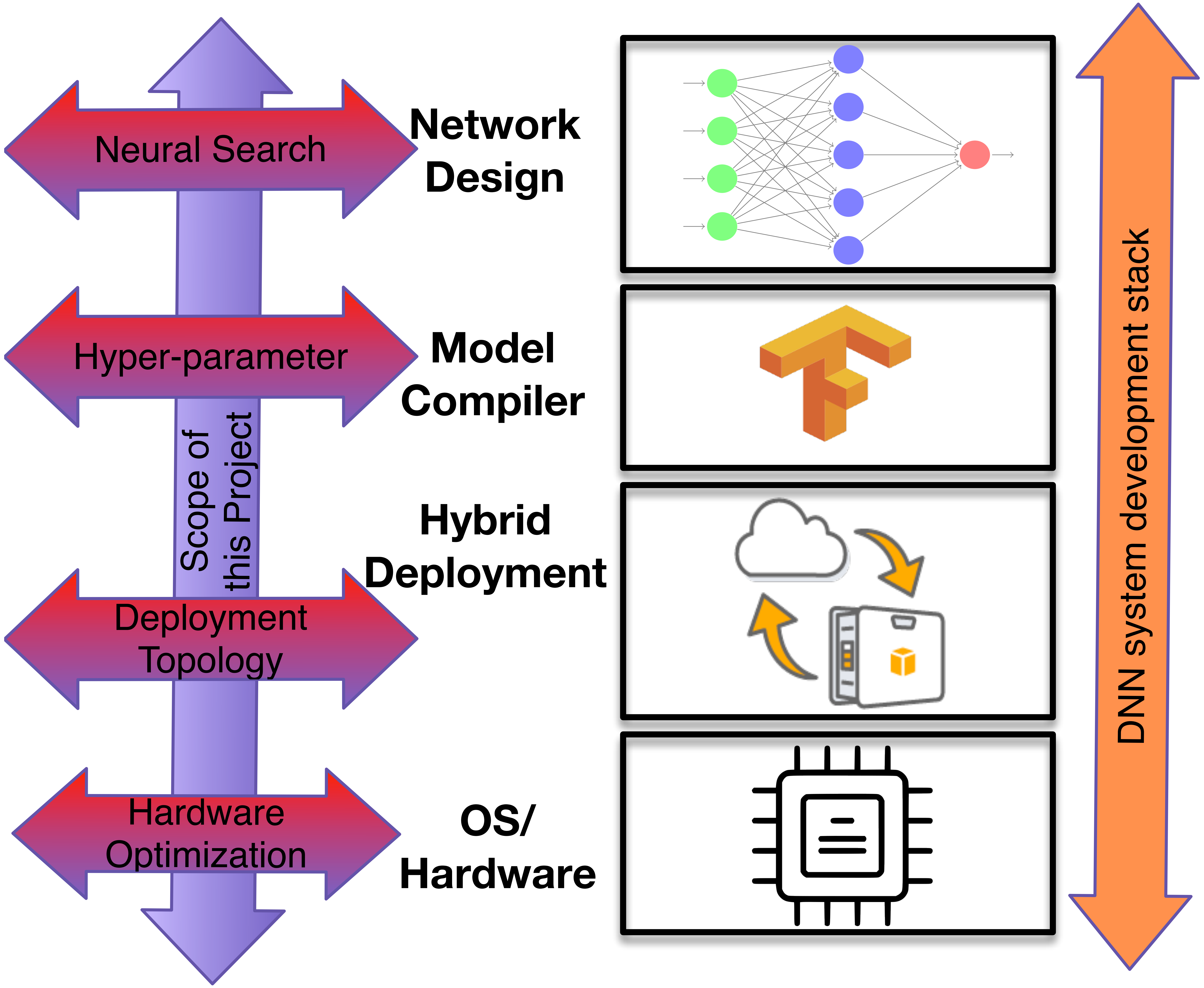




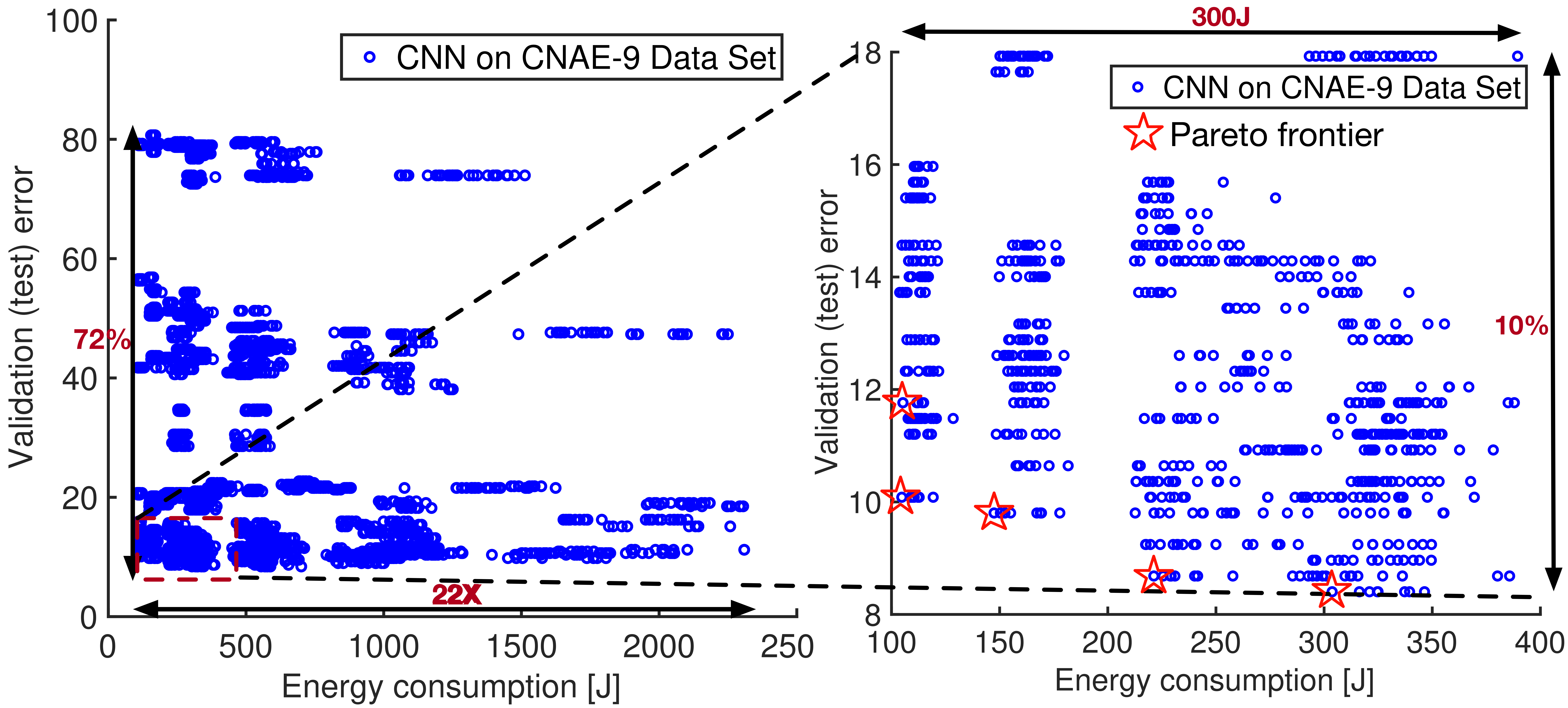
Increasingly **competing objectives**



Deep neural network as a highly configurable system



We found many configuration with the same accuracy while having drastically different energy demand



Details: [FlexiBO]

FLEXIBO: Cost-Aware Multi-Objective Optimization of Deep Neural Networks

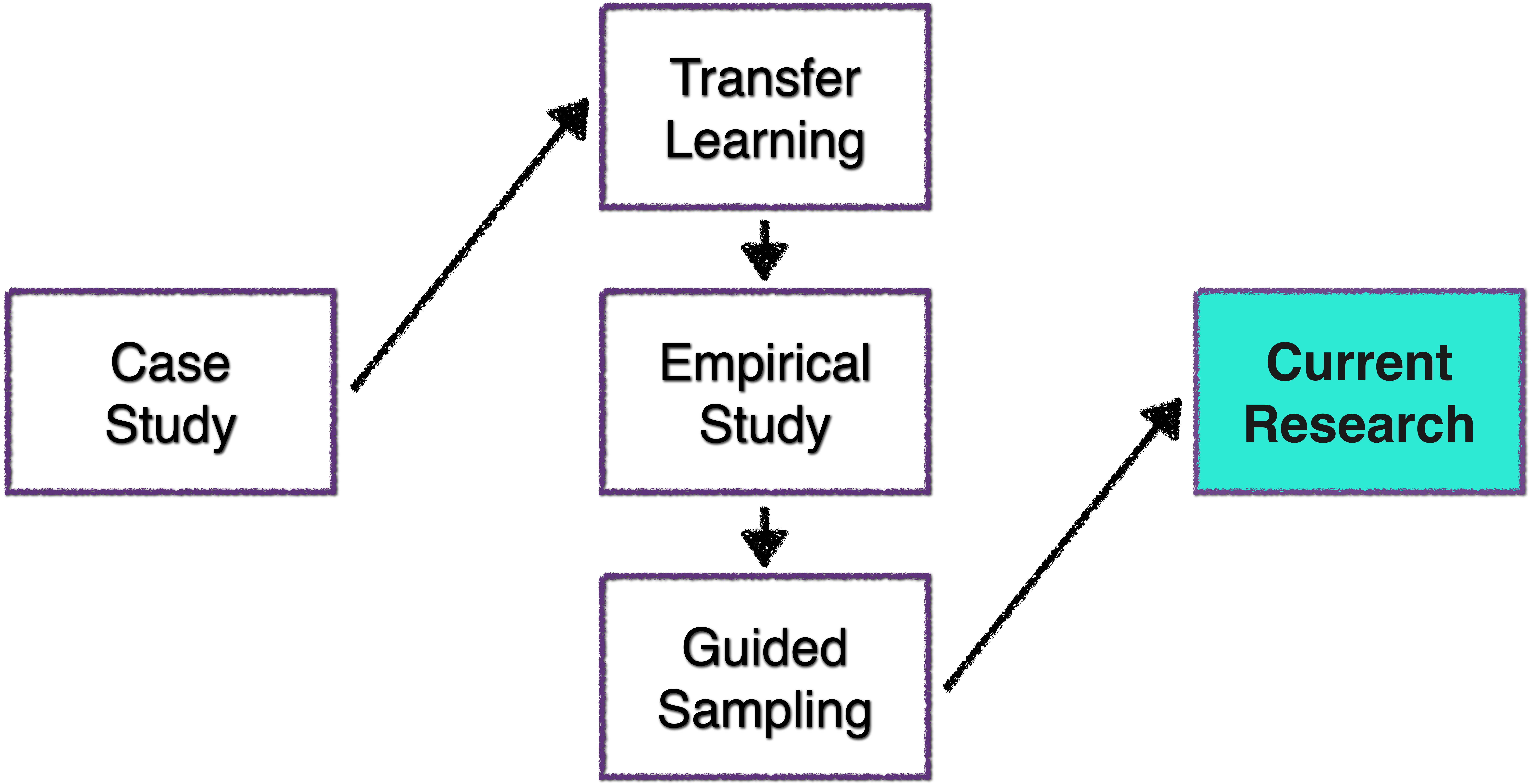
Md Shahriar Iqbal MIQBAL@EMAIL.SC.EDU
University of South Carolina, Columbia SC 29201 USA

Jianhai Su SUJ@EMAIL.SC.EDU
University of South Carolina, Columbia SC 29201 USA

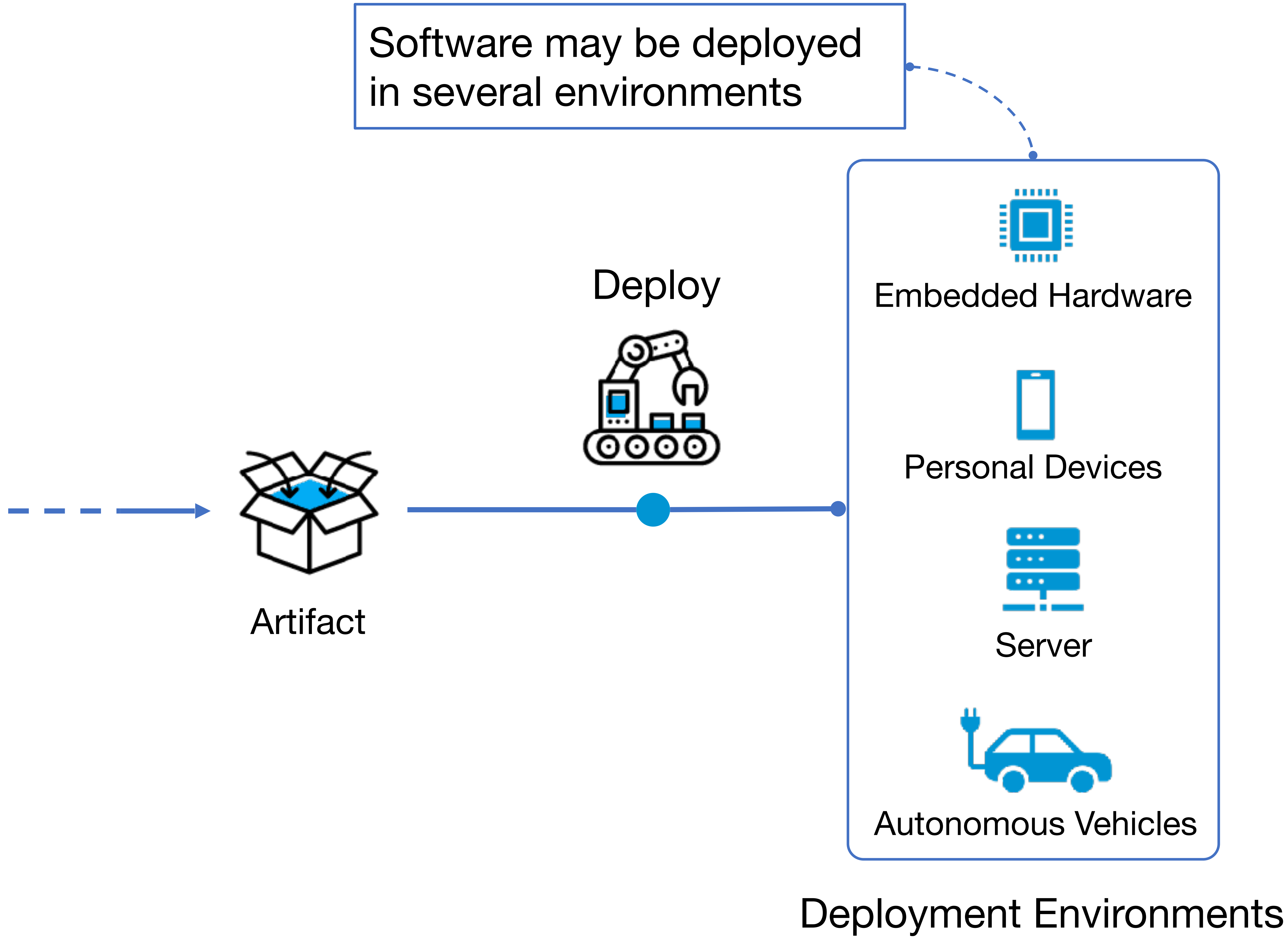
Lars Kotthoff LARSKO@UWYO.EDU
University of Wyoming, Laramie WY 82071 USA

Pooyan Jamshidi PJAMSHID@CSE.SC.EDU
University of South Carolina, Columbia SC 29201 USA

Outline


Case Study
Transfer Learning
Empirical Study
Guided Sampling
Current Research

Overview



Problem

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

Why?

- The configuration space is combinatorially large with 1000’s of configuration options
- There are several non-trivial interactions between the software and the hardware

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, and/or high heat dissipation

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?

Causal Debugging (with CADET)

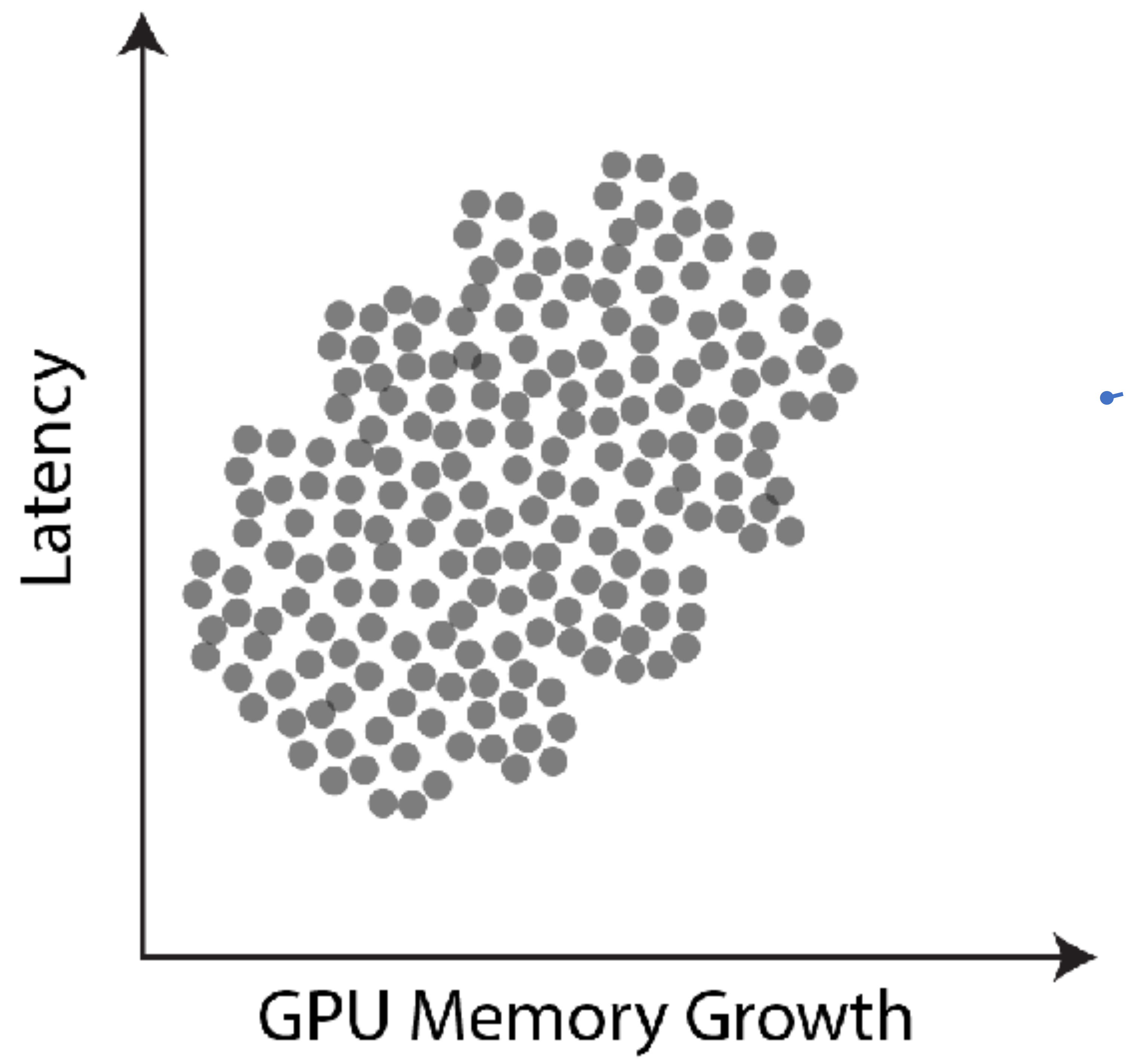
Objective

Diagnose and fix the root-cause of misconfigurations that cause non-functional faults

Approach

Use Causal Models and Counterfactual reasoning to diagnose and fix misconfigurations

Why Causal Inference? (Simpson's Paradox)



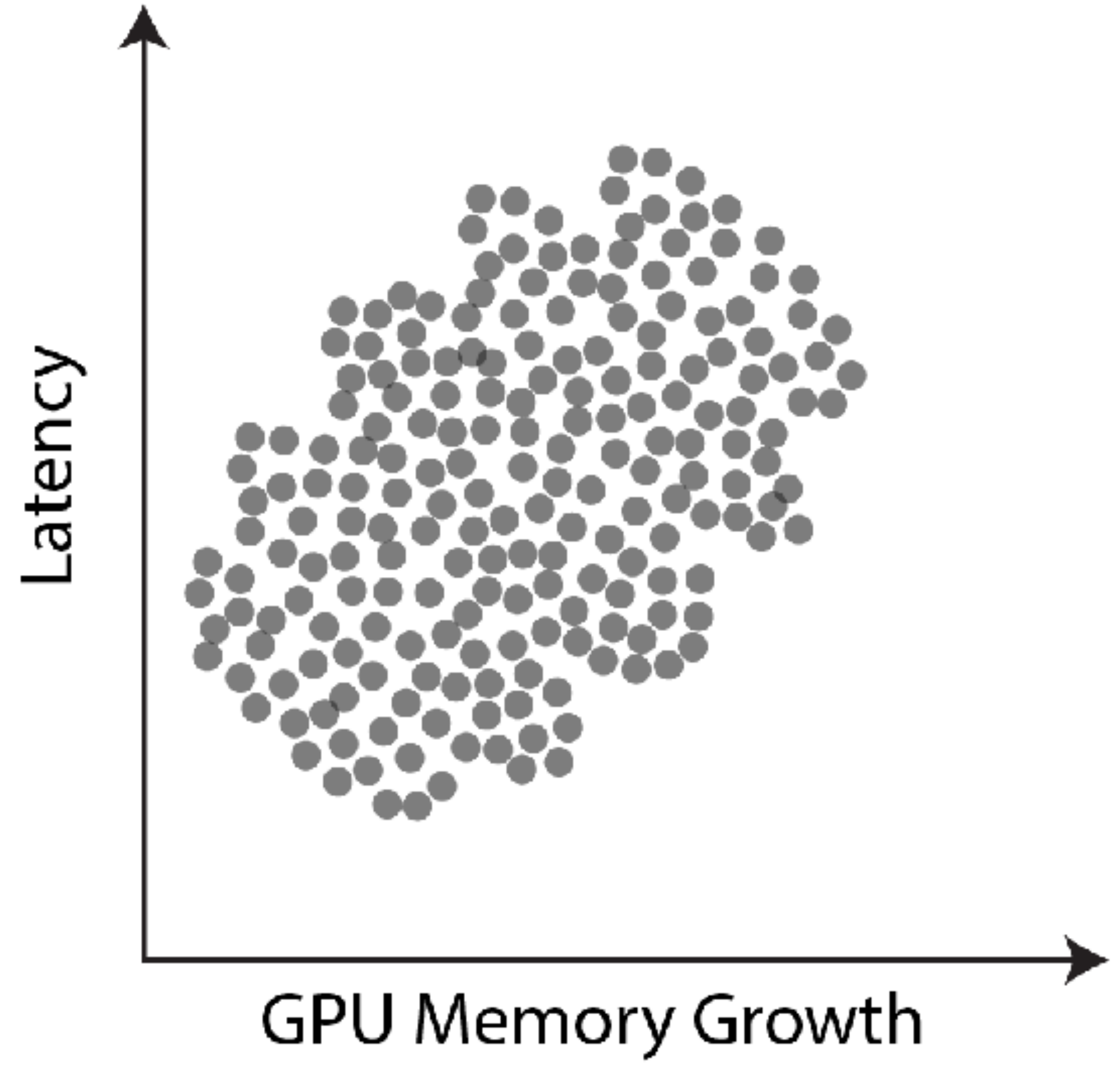
Increasing GPU memory increases Latency

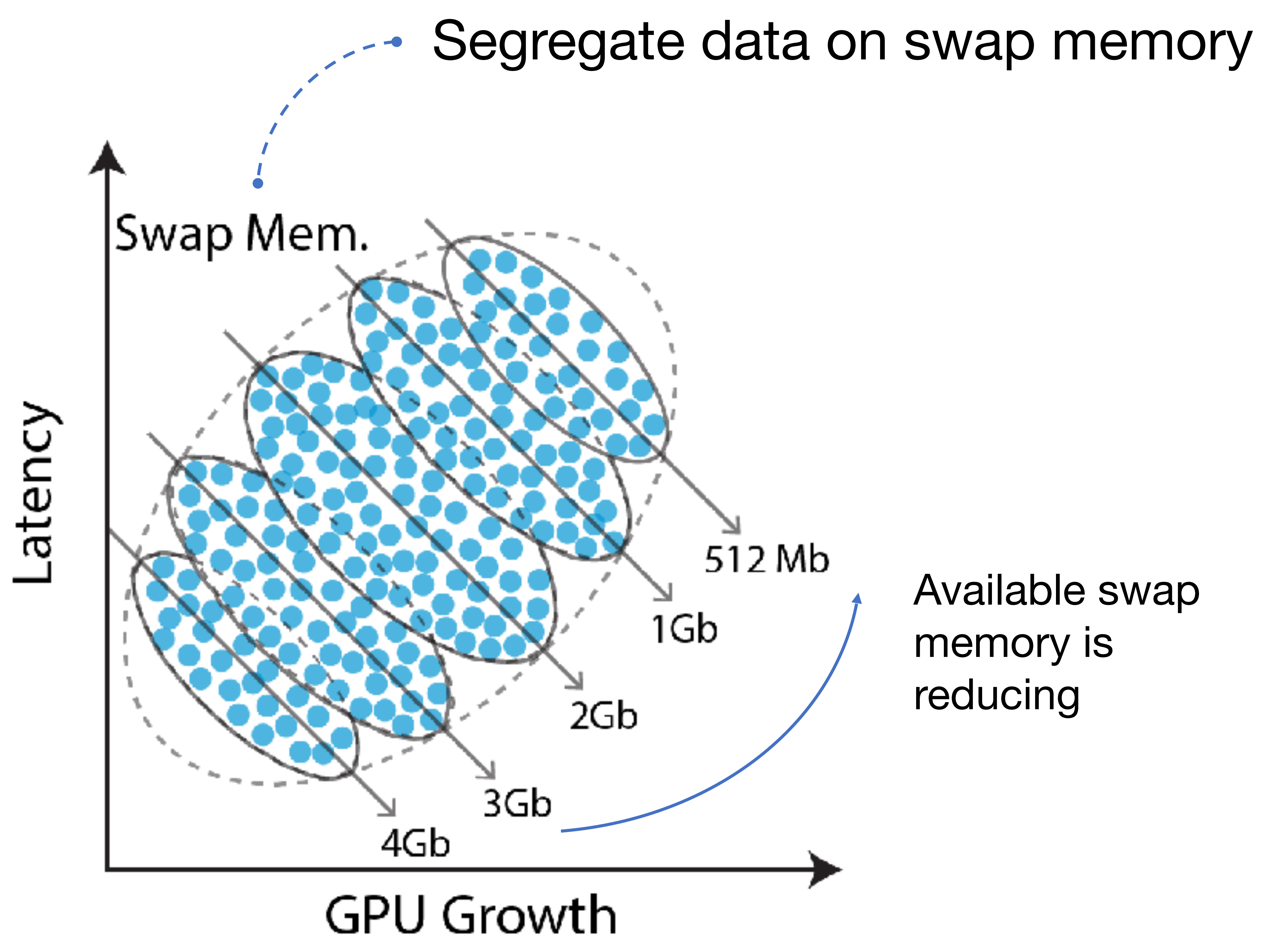
Counterintuitive!

More GPU memory usage should reduce latency not increase it.

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

Why Causal Inference? (Simpson's Paradox)



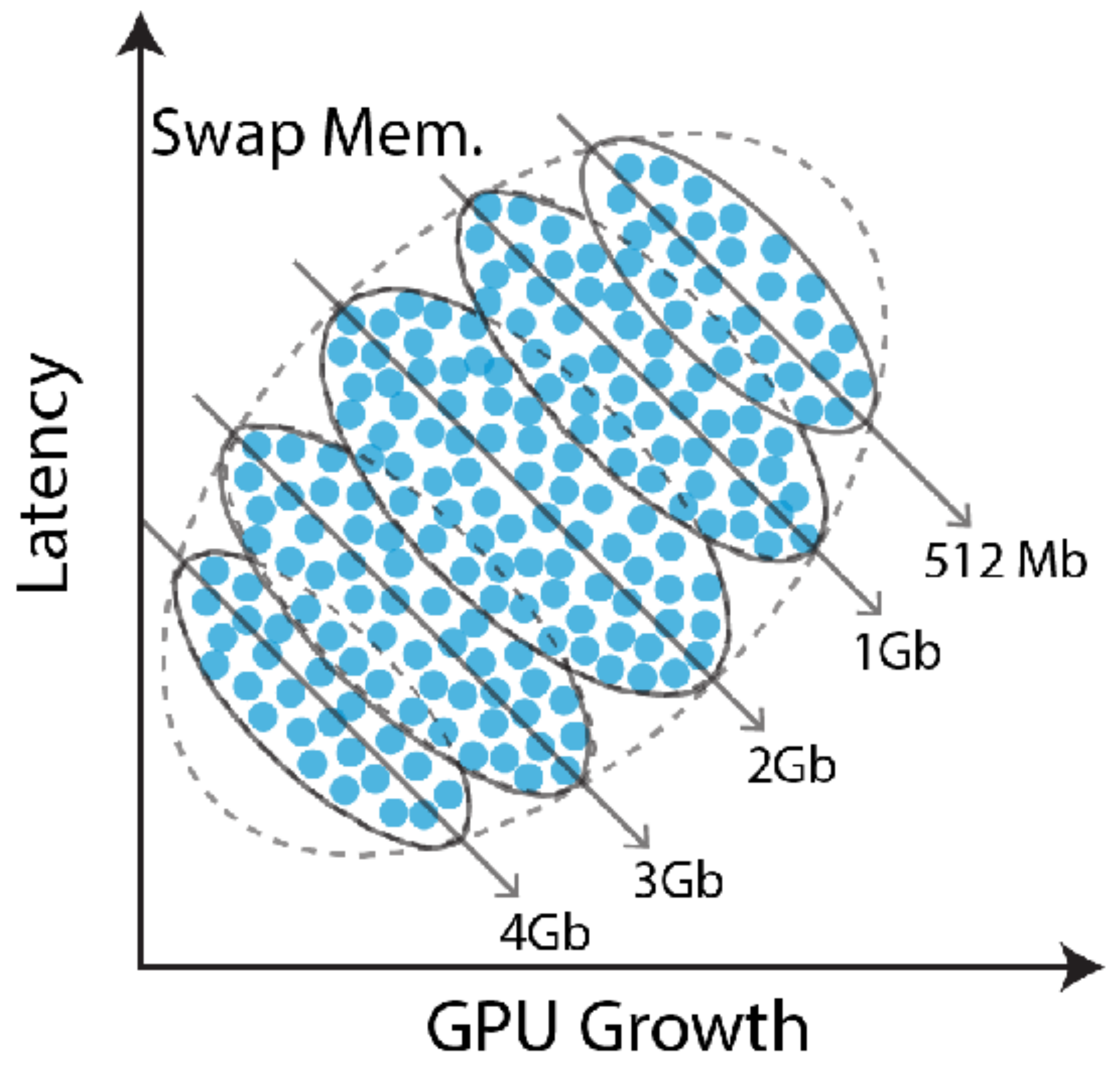


GPU memory growth borrows memory from the swap for some intensive workloads. Other host processes may reduce the available swap. Little will be left for the GPU to use.

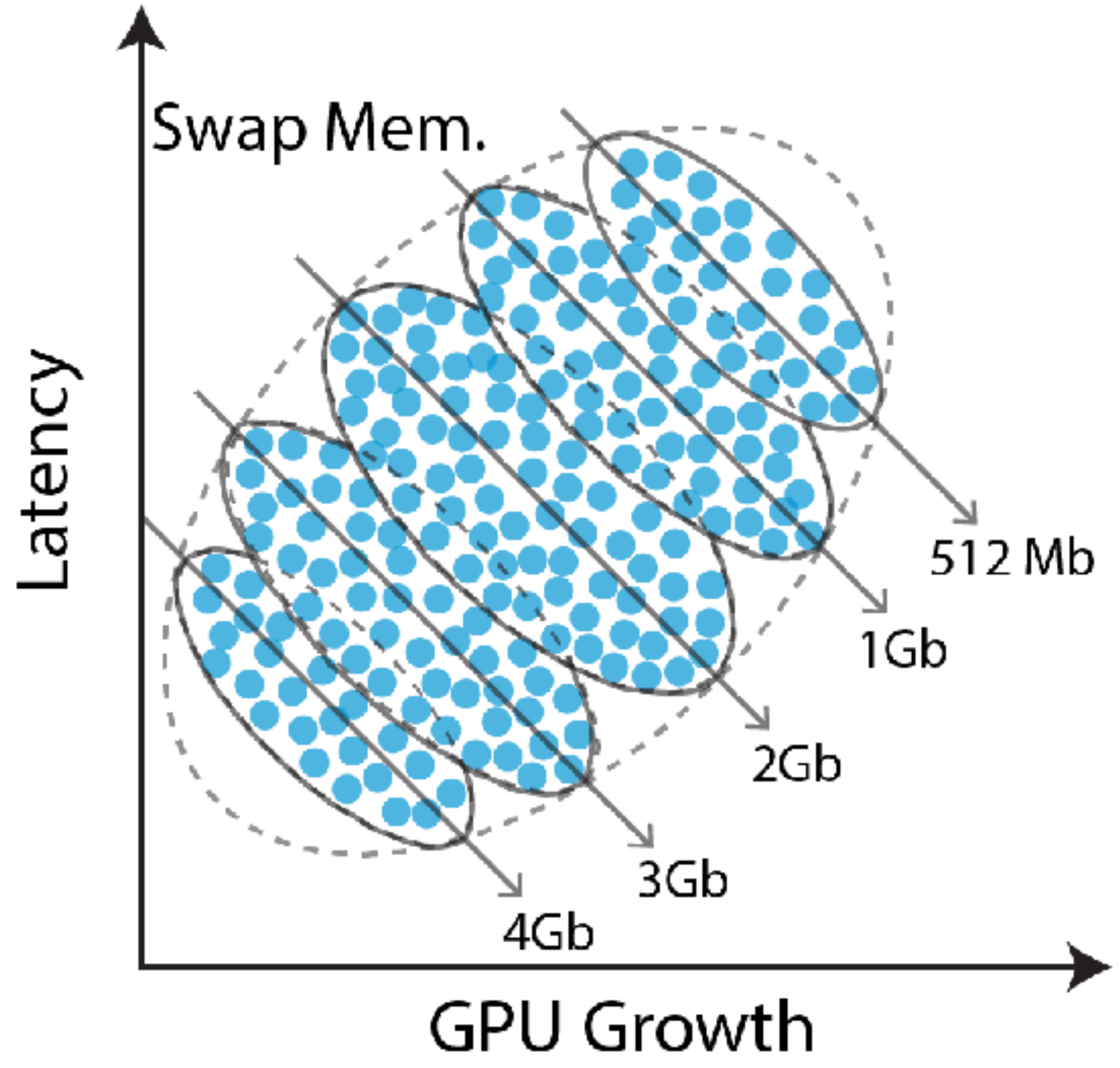
Why Causal Inference?

Real world problems can have 100s if not 1000s of interacting configuration options

Manually understanding and evaluating each combination is impractical, if not impossible.



Causal Models

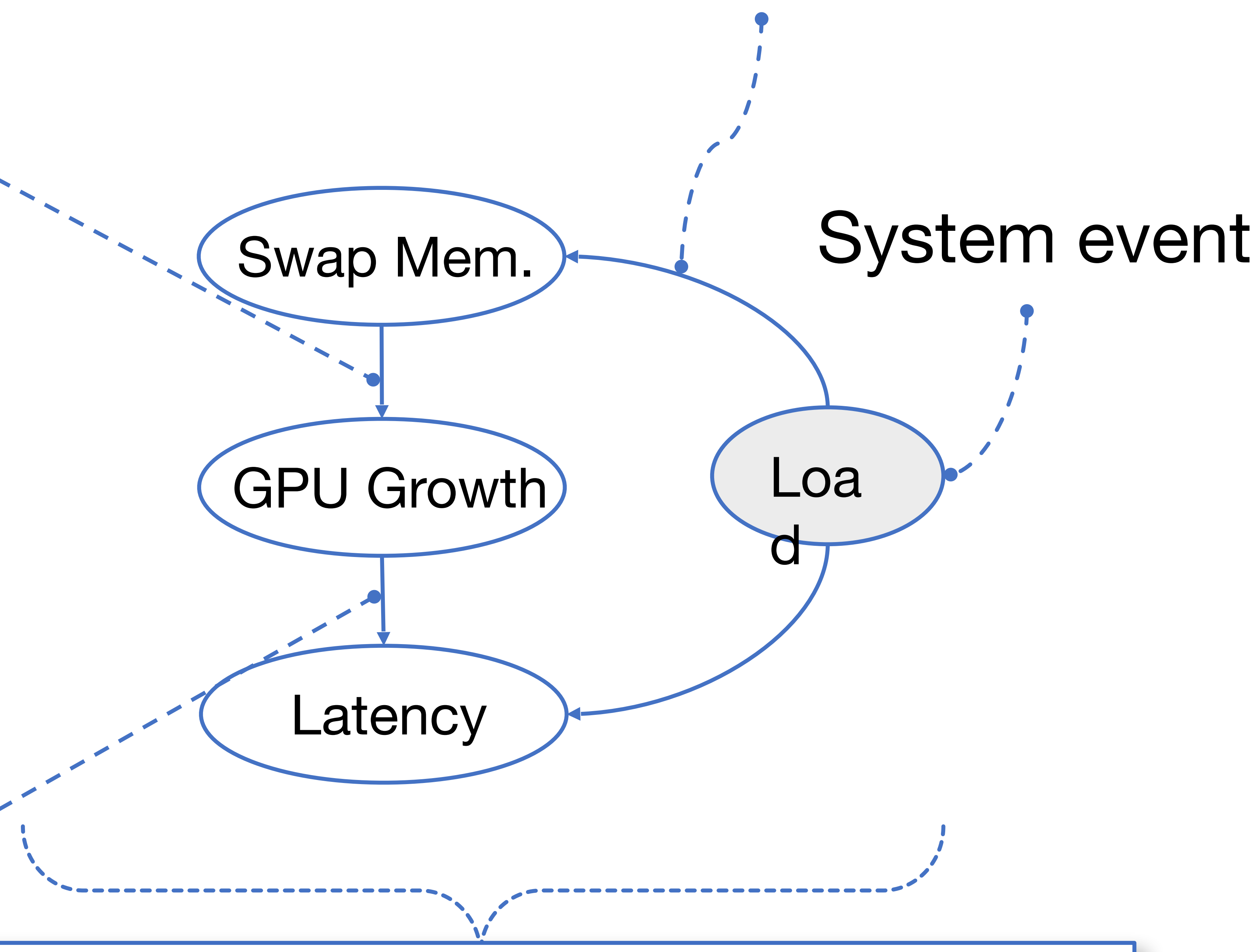


Configurations parameters

Direction(s) of the causality

System event

Express the relationships between interacting variables as a causal graph

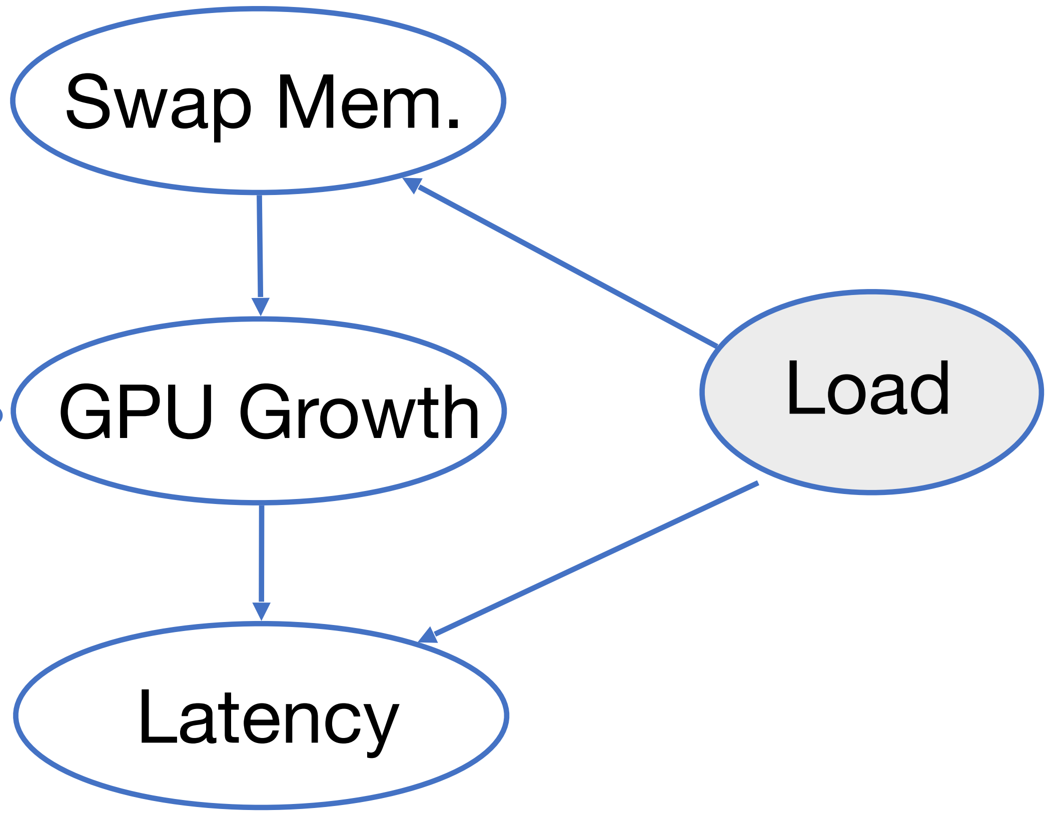


Non-functional property

- GPU growth affects latency via swap memory
- External factors like resource pressure also affects swap memory

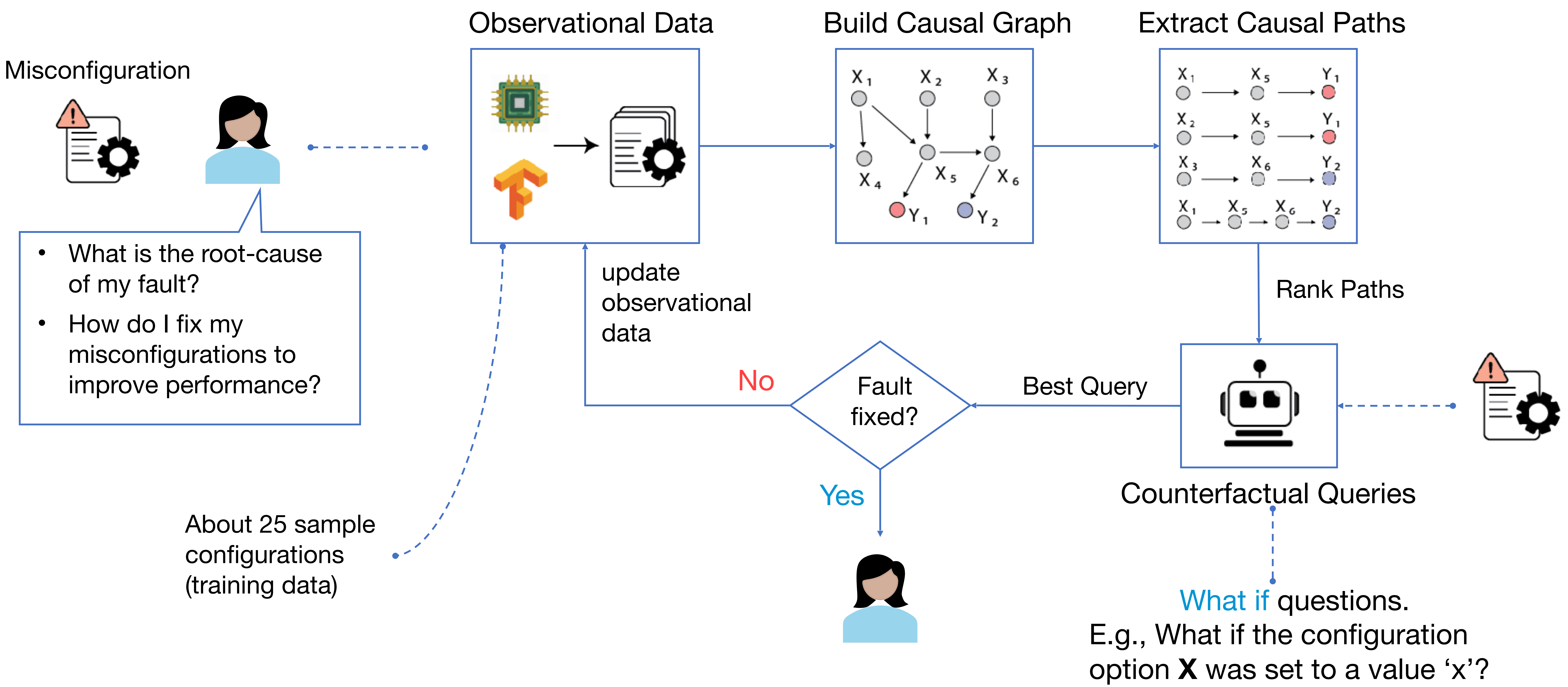
Causal Models

How to construct this causal graph?

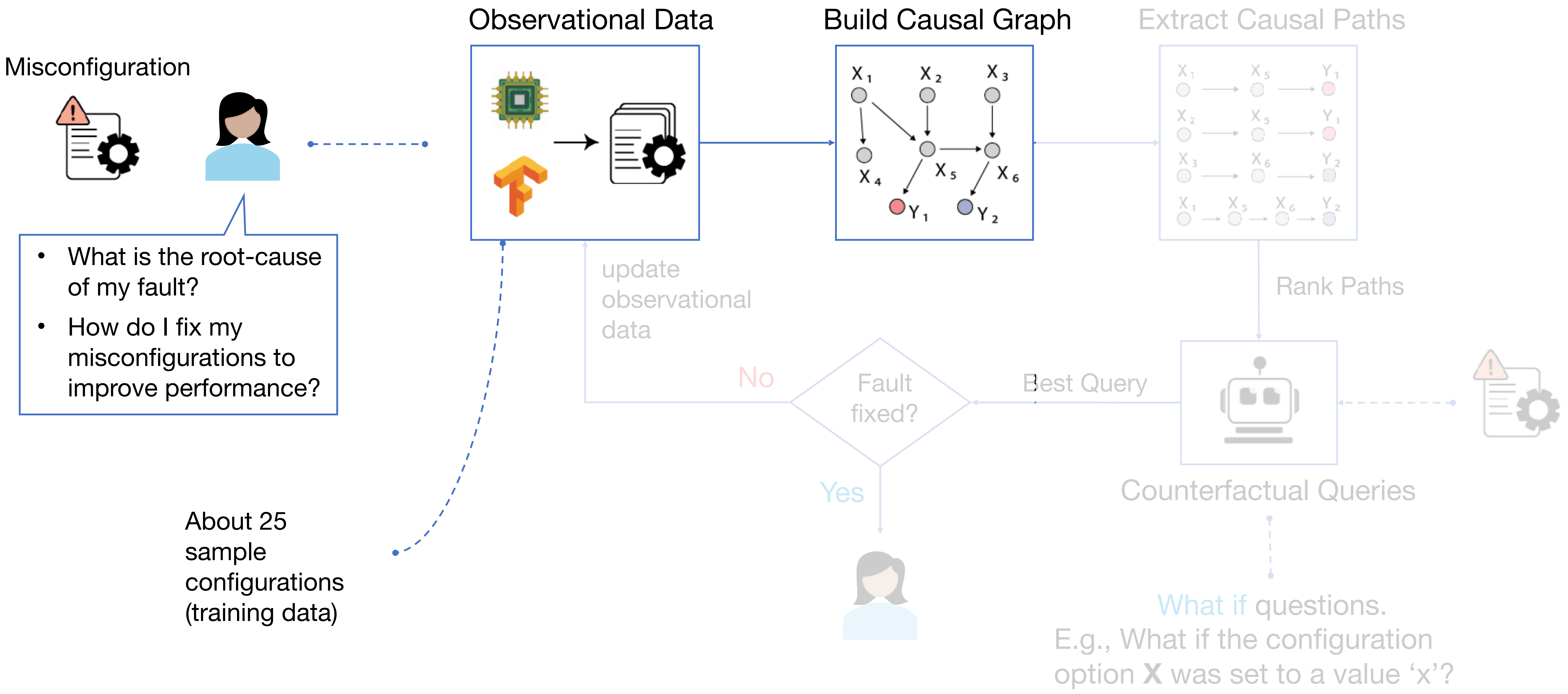


If there is a fault in latency, how to diagnose and fix it?

CADET: Causal Debugging Tool



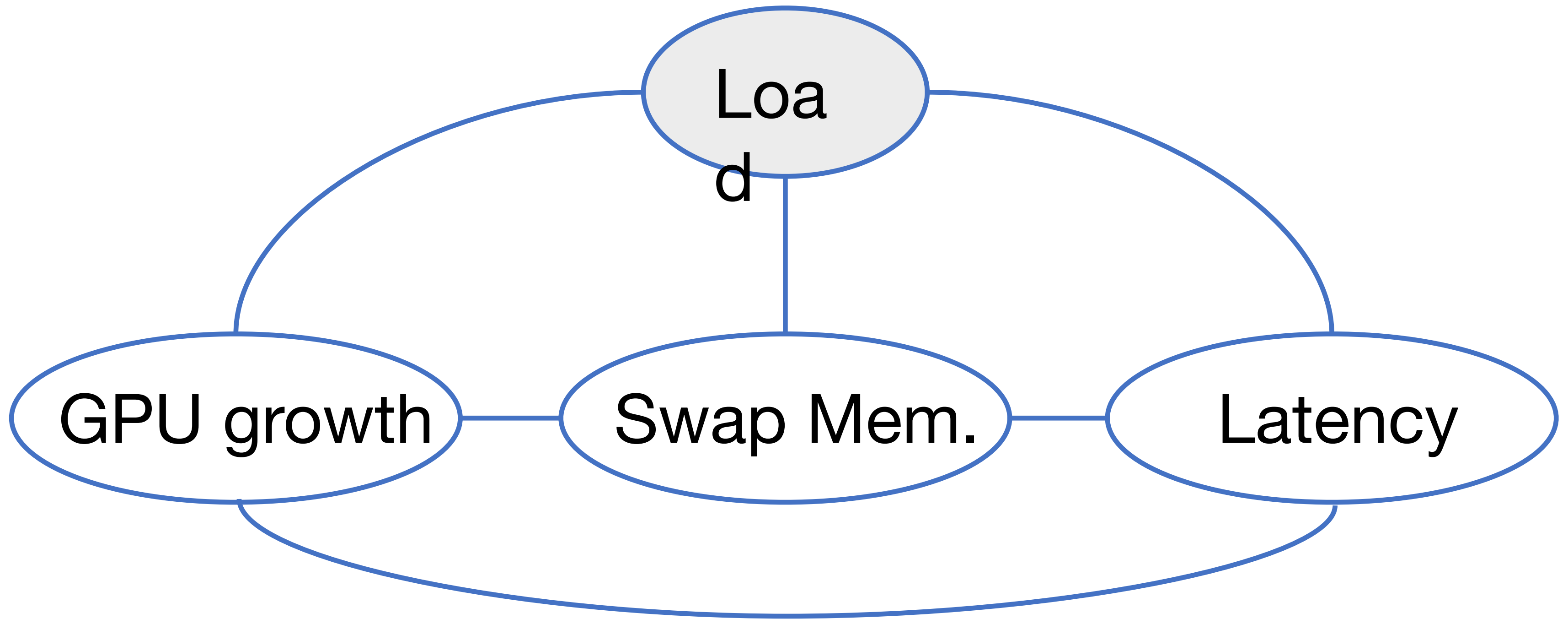
STEP 1: Generating a Causal Graph



Generating a Causal Graph (with FCI)

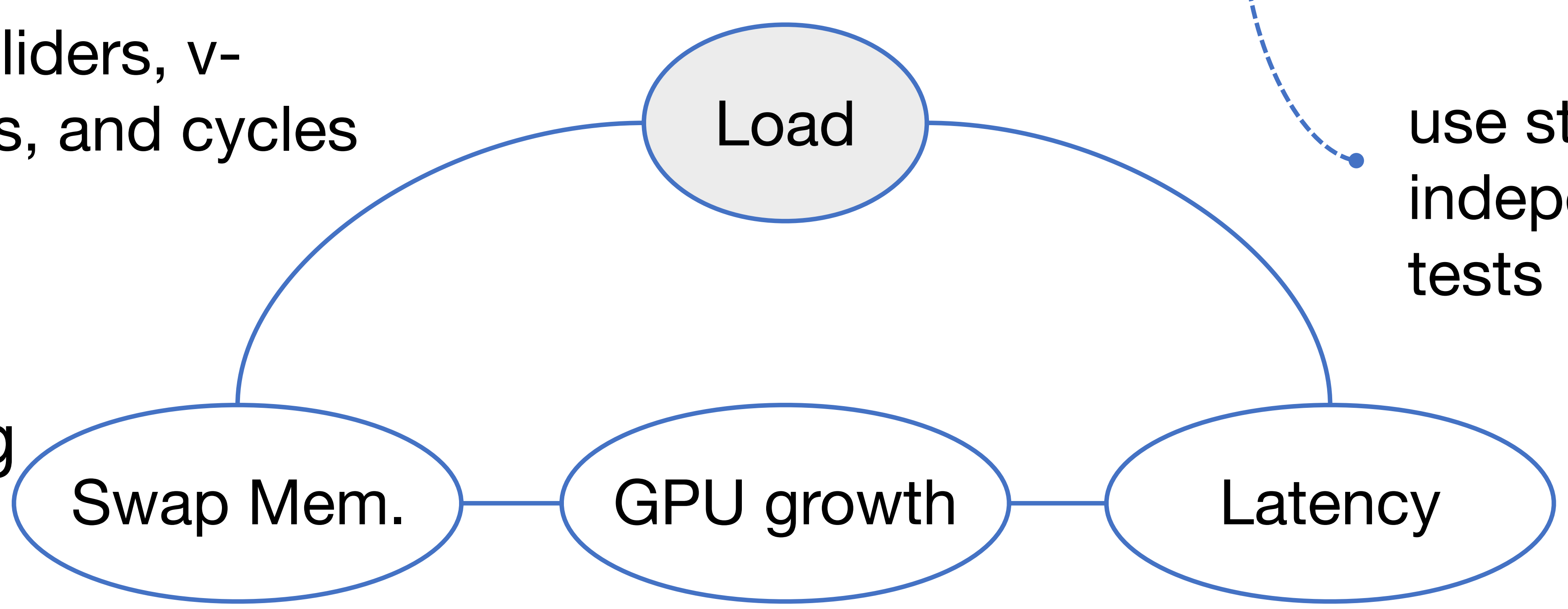
	GPU Growth	Swap Mem.	Load	Latency
c_1	0.2	2 Gb	10%	1 sec
c_2	0.5	1 Gb	20%	2 sec
⋮	⋮	⋮	⋮	⋮
c_n	1.0	4 Gb	40%	0.1 sec

Fully connected skeleton



Prune away edges between independent variables

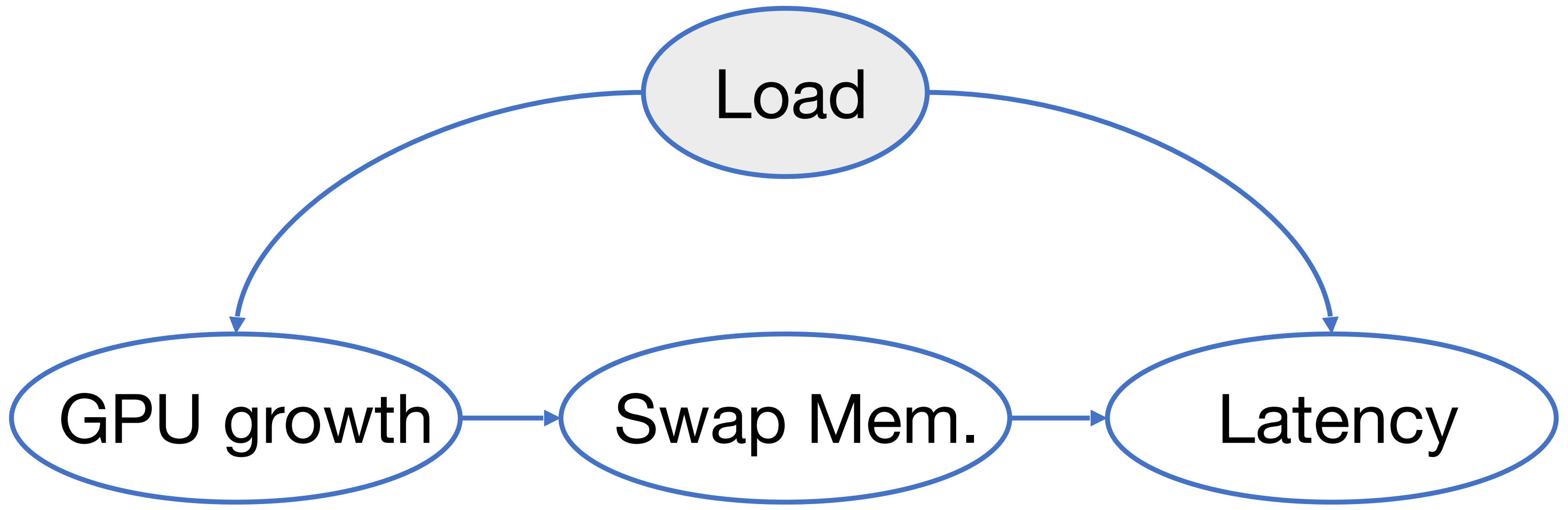
use statistical independence tests



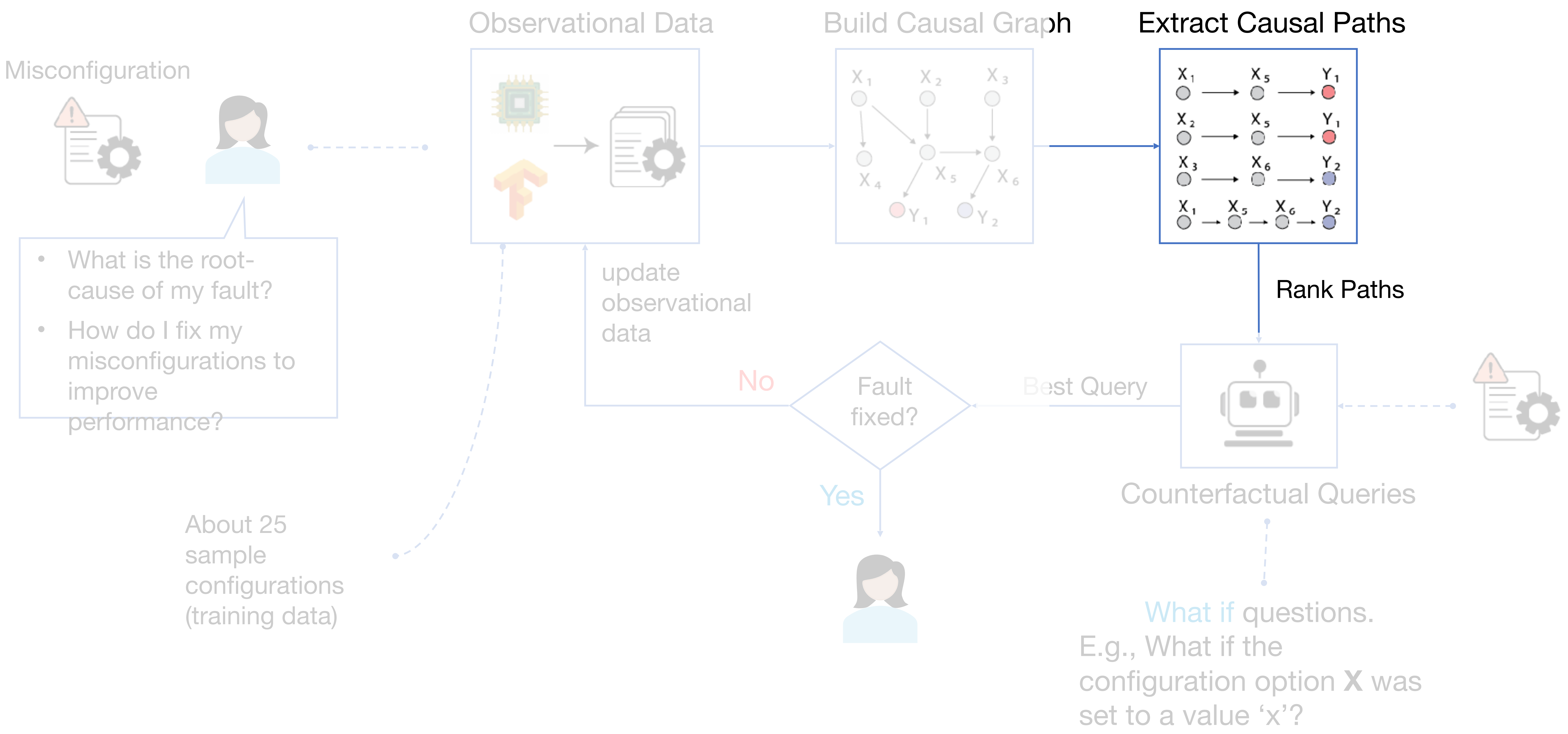
Use standard orientation rules for forks, colliders, v-structures, and cycles

orient remaining edges

Acyclic Directed Mixed Graph



STEP 2: Extracting Paths from the Graph



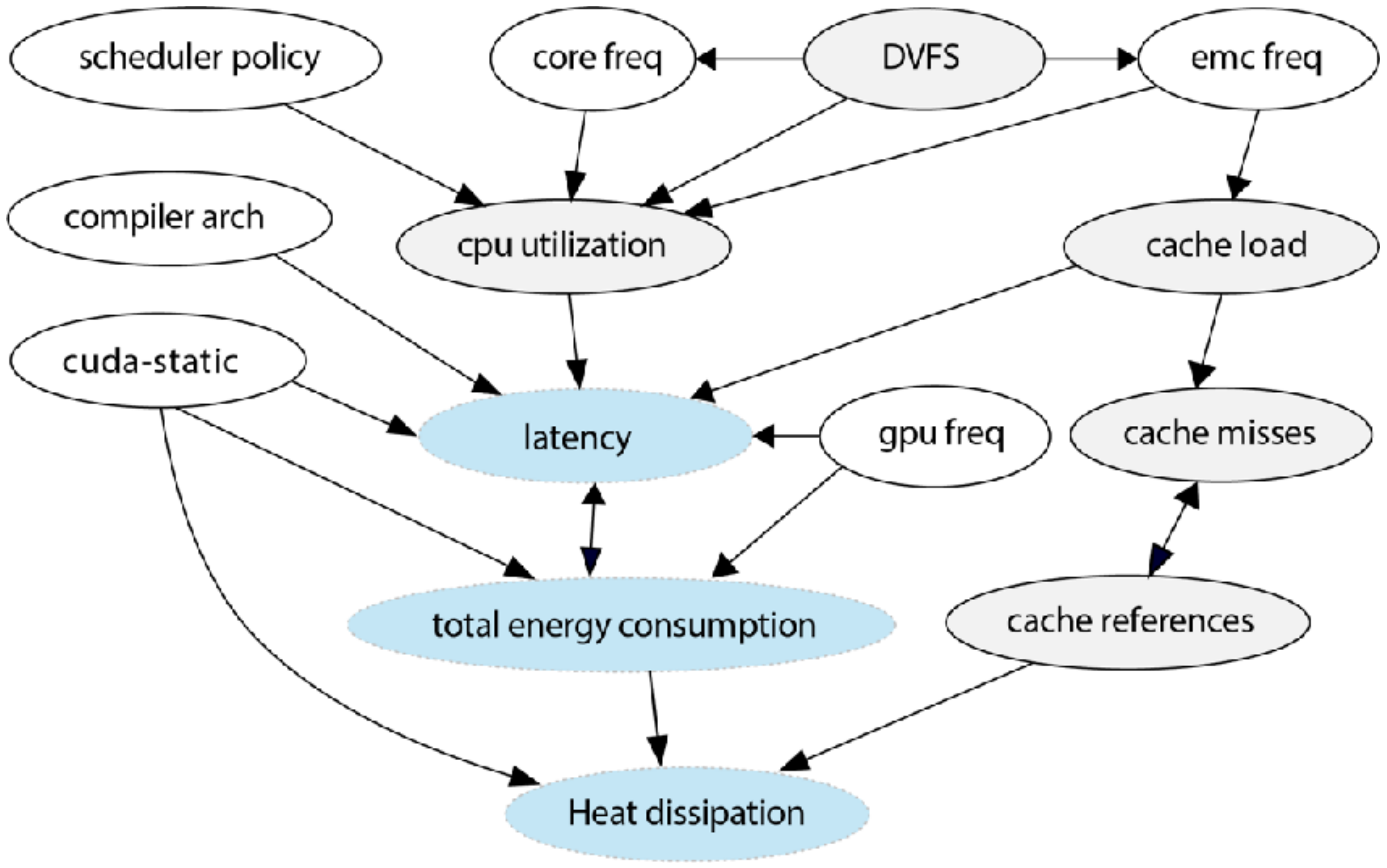
Extracting Paths from the Causal Graph

Problem

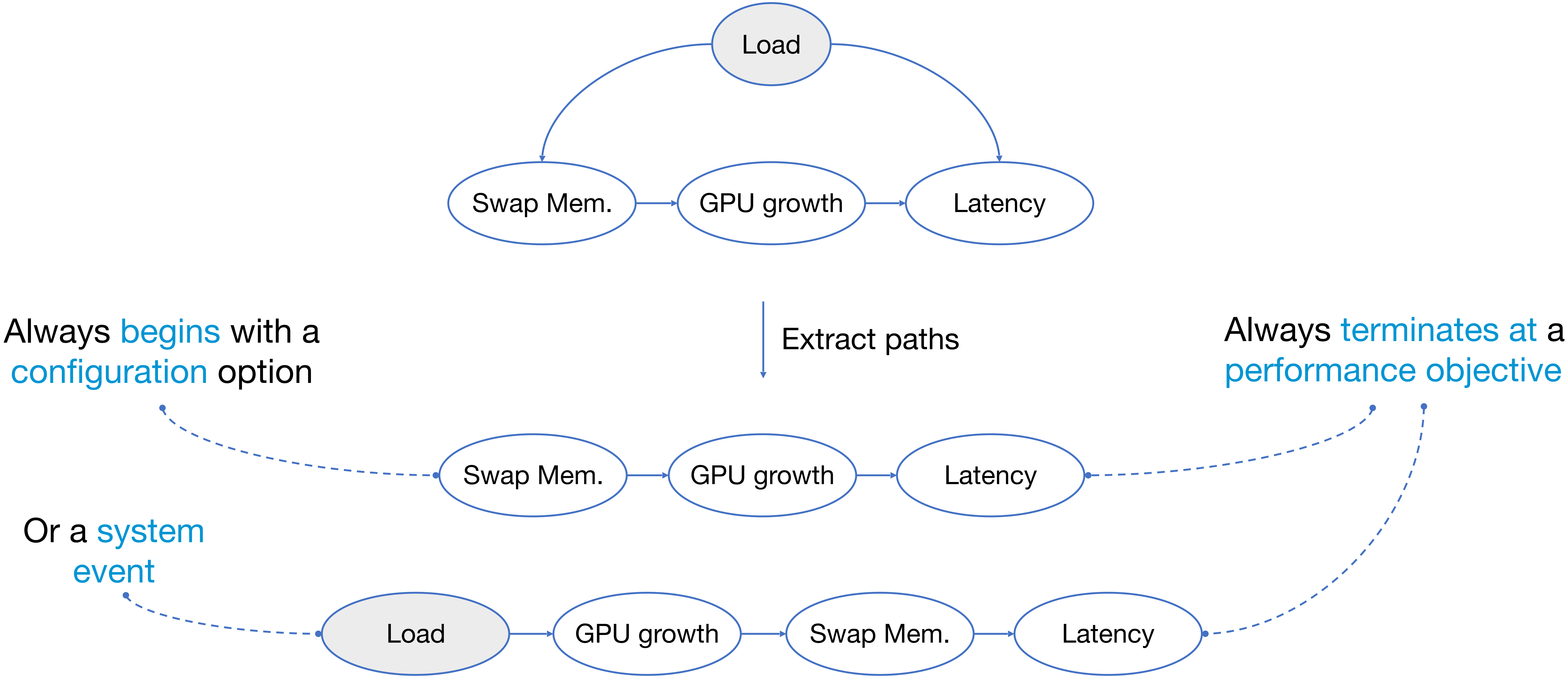
- In real world cases, this causal graph can be very complex
- It may be intractable to reason over the entire graph directly

Solution

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

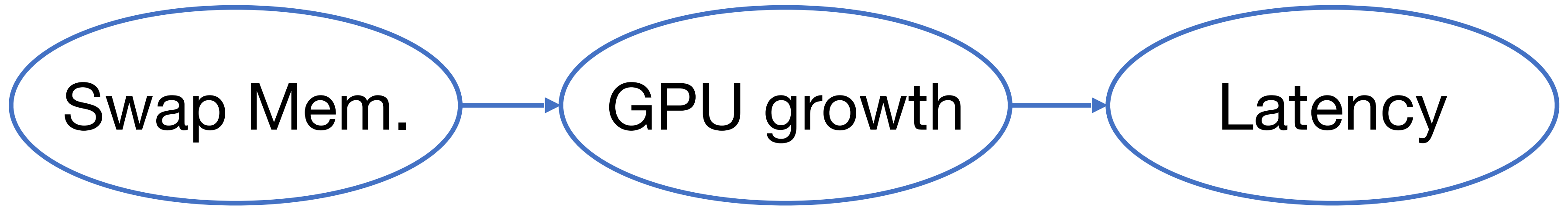


Extracting Paths from the Causal Graph



Ranking Paths from the Causal Graph

- Compute the Average Causal Effect (ACE) of each pair of neighbors in a path



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

Sum over all permitted values of Swap memory (Z).

Expected value of GPU growth (X) when we artificially intervene by setting Swap (Z) to the value b

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

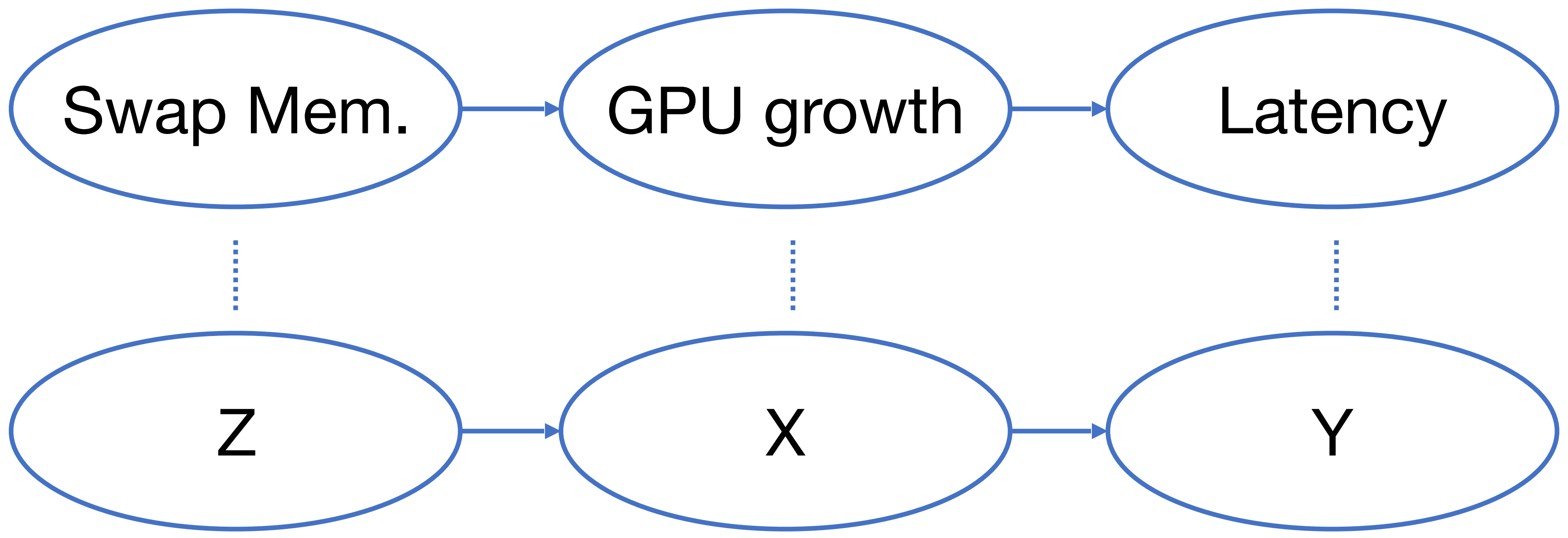
Expected value of GPU growth (X) when we artificially intervene by setting Swap (Z) to the value a

Ranking Paths from the Causal Graph

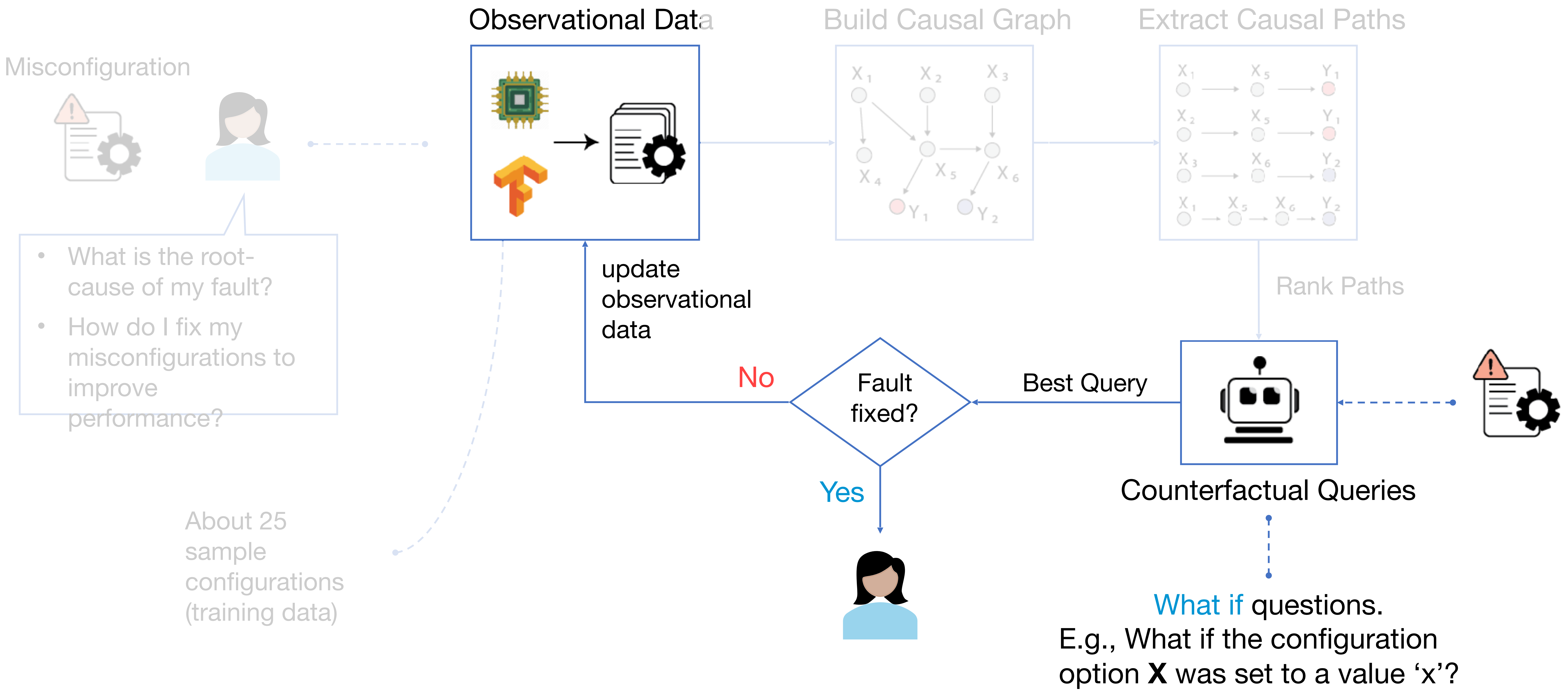
- Sum the ACE of all pairs of adjacent nodes in the path

$$PACE(X, Z) = \frac{1}{N} \sum_{Z,X \in Path} ACE(Z,\ X)$$

Sum over all pairs of nodes in the causal path.



STEP 3: 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?"

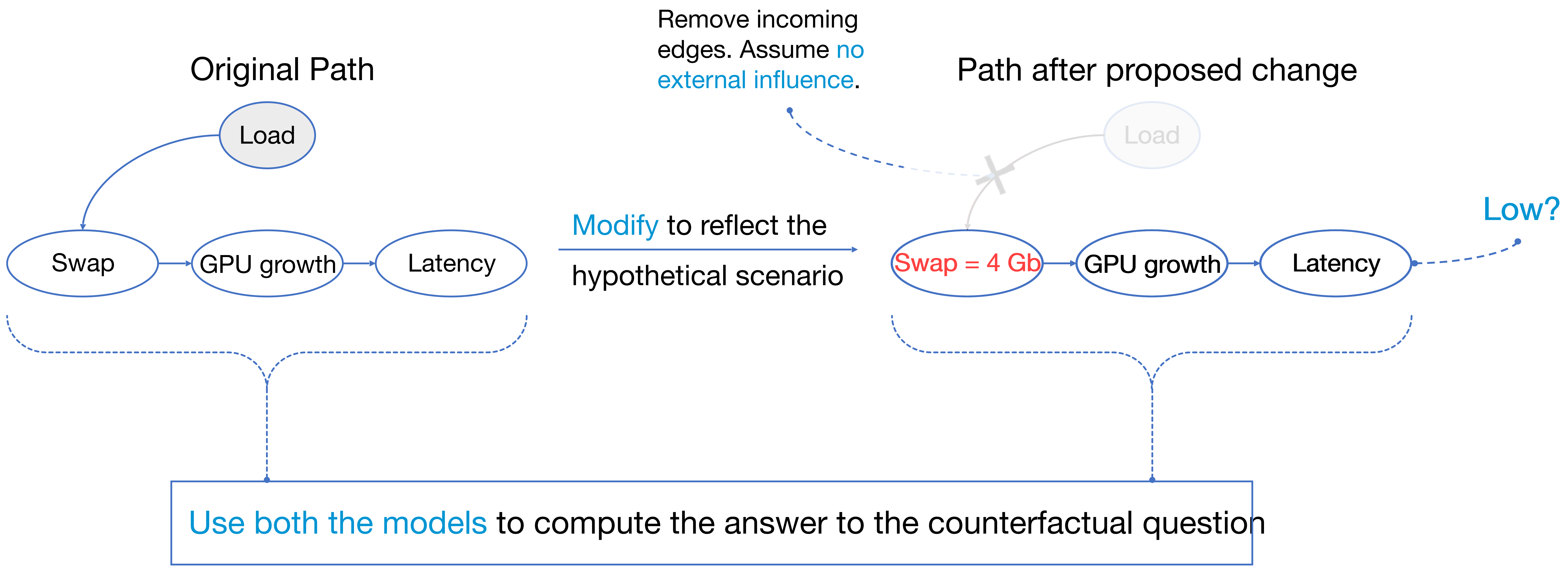
We are interested in the scenario where:

- We hypothetically have low latency;

Conditioned on the following events:

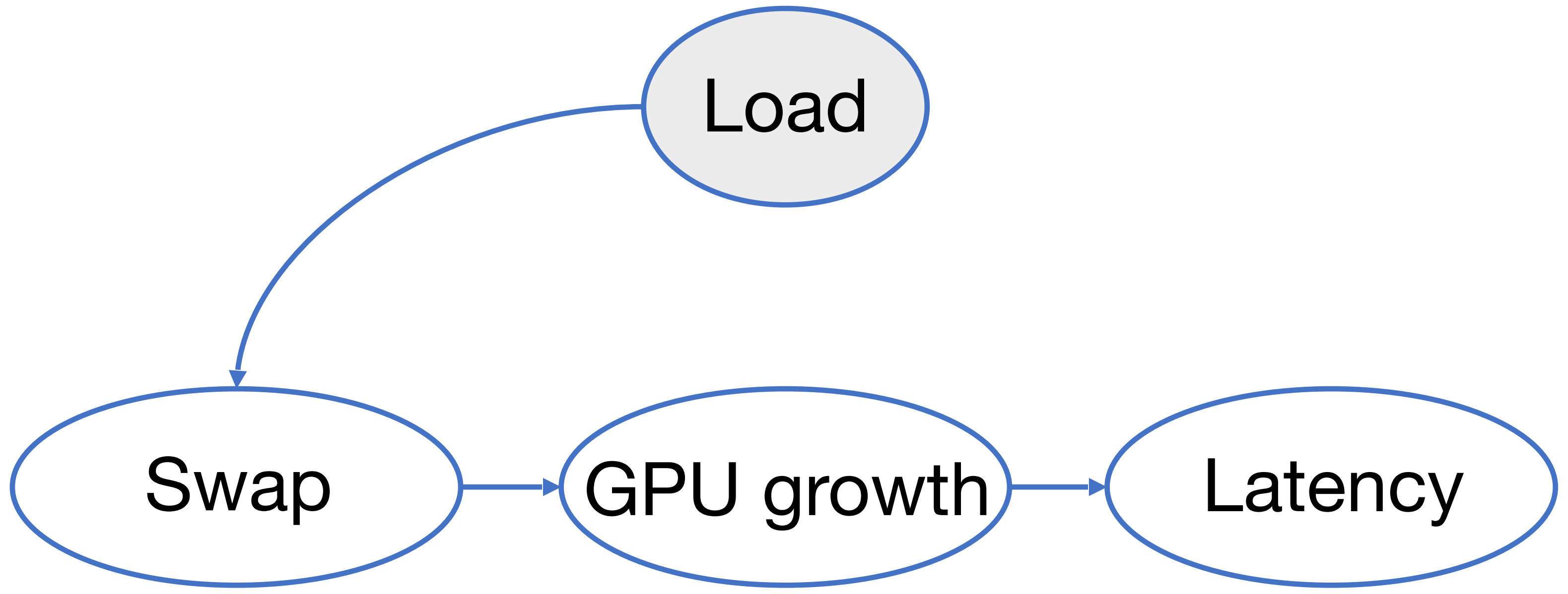
- We hypothetically set the new Swap memory to 4 Gb
- Swap Memory was initially set to 2 Gb
- We actually observed high latency when Swap was set to 2 Gb
- Everything else remains the same

Diagnosing and Fixing the Faults

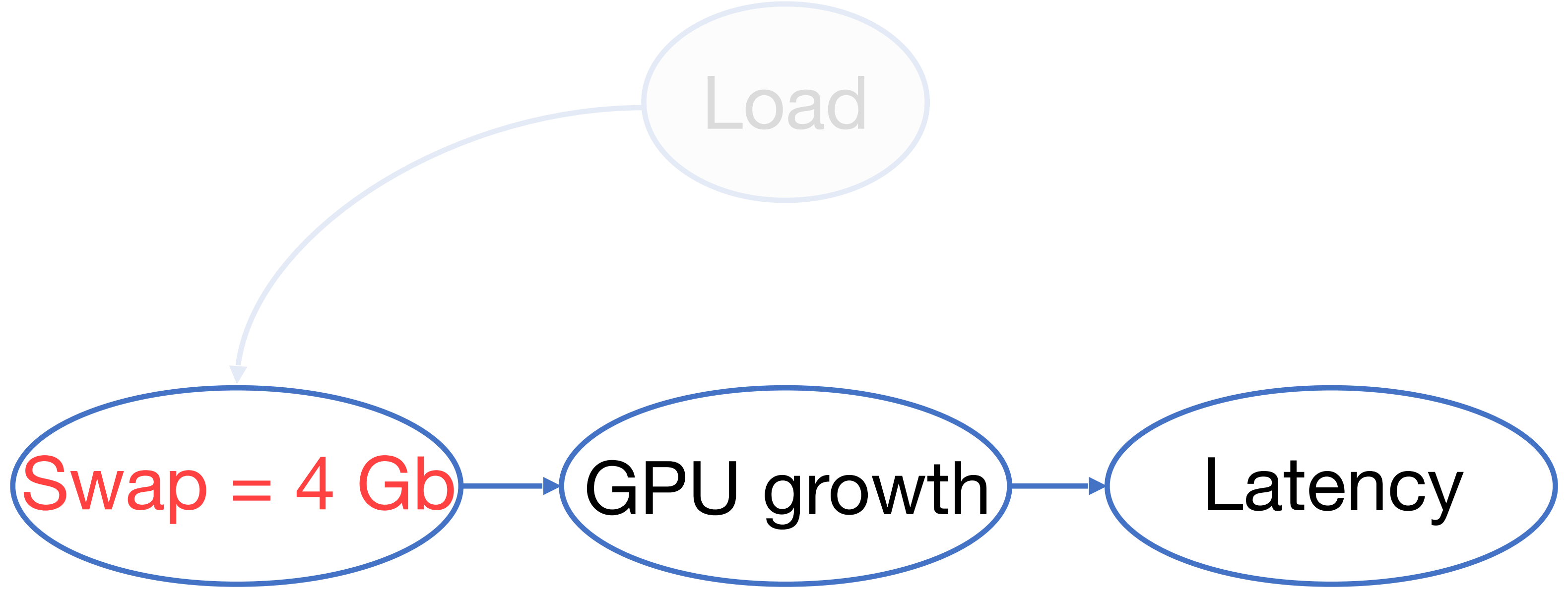


Diagnosing and Fixing the Faults

Original Path



Path after proposed change



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

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

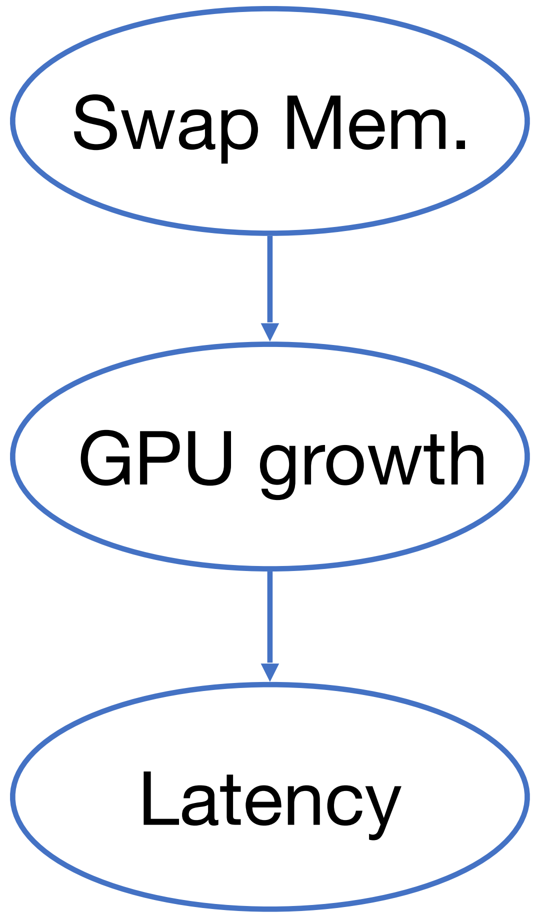
$$\text{Potential} = \underbrace{P\left(\hat{outcome} = \color{red}{good} \;\middle|\; change,\quad outcome_{\neg change} = bad,\quad \neg change, \sim U \right)}_{\text{Probability that the outcome is good after a change}}$$

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

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

Diagnosing and Fixing the Faults

Top K paths



…

Set every configuration option in the path to all permitted values

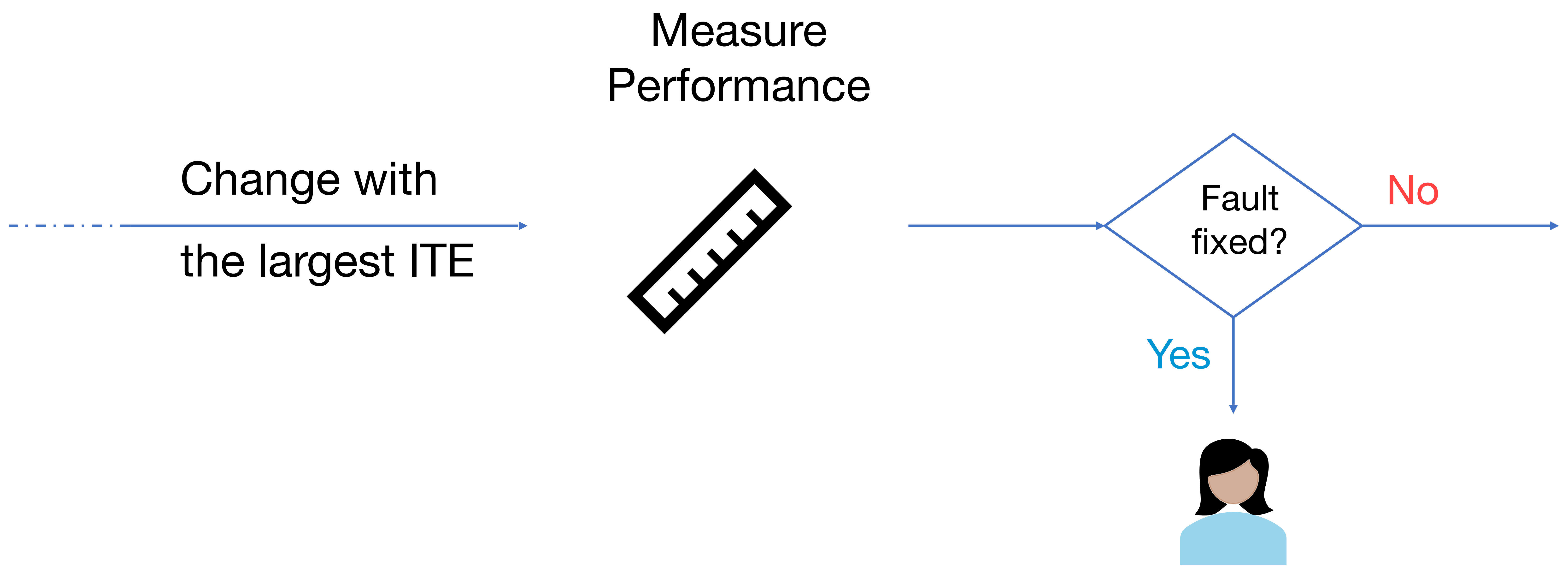
Enumerate all possible changes

$ITE(change)$

Change with the largest ITE

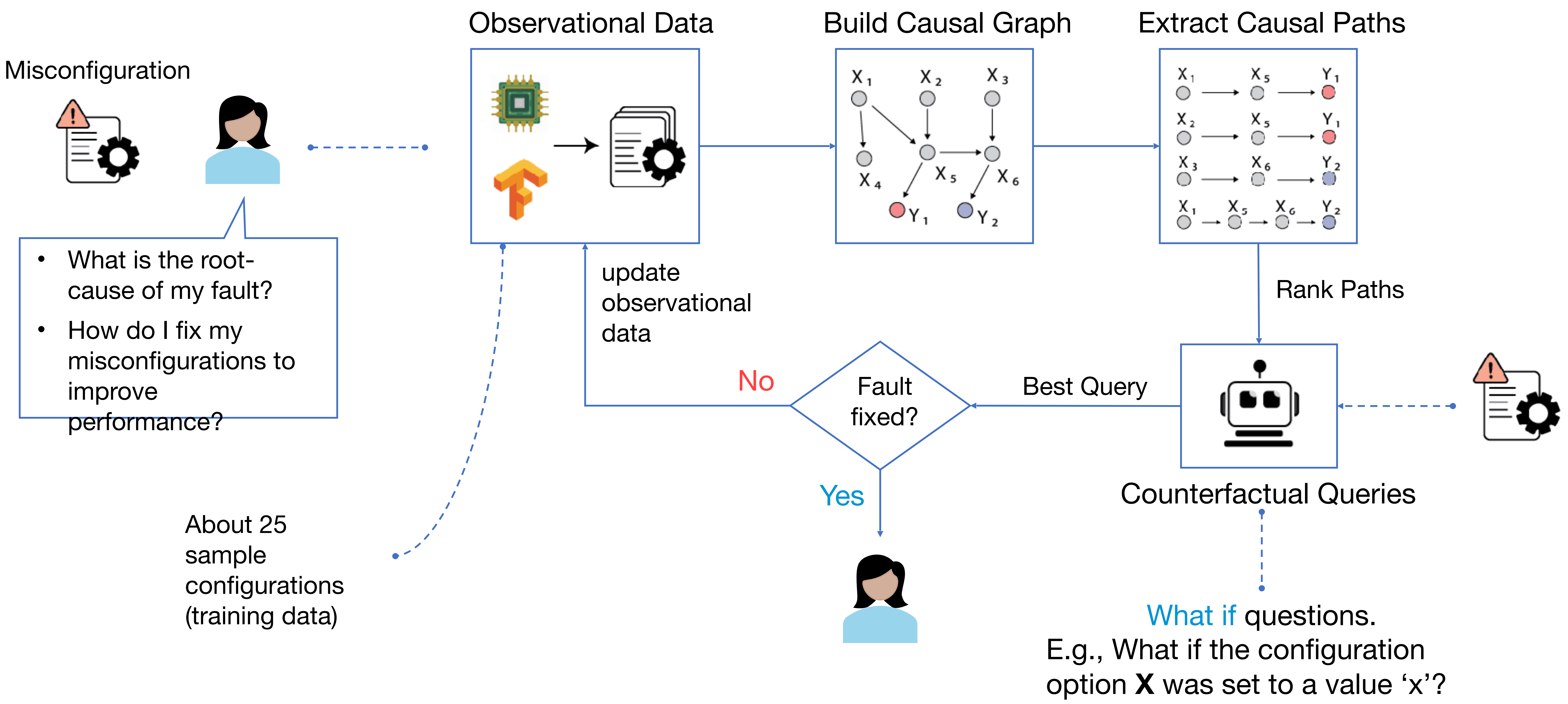
Inferred from available data. This is very cheap.

Diagnosing and Fixing the Faults



- Add to observational data
- Update causal model
- Repeat…

CADET: End-to-End Pipeline



Results: 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

Results: Motivating Example

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: Motivating Example

Configuration	CADET	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		✓	

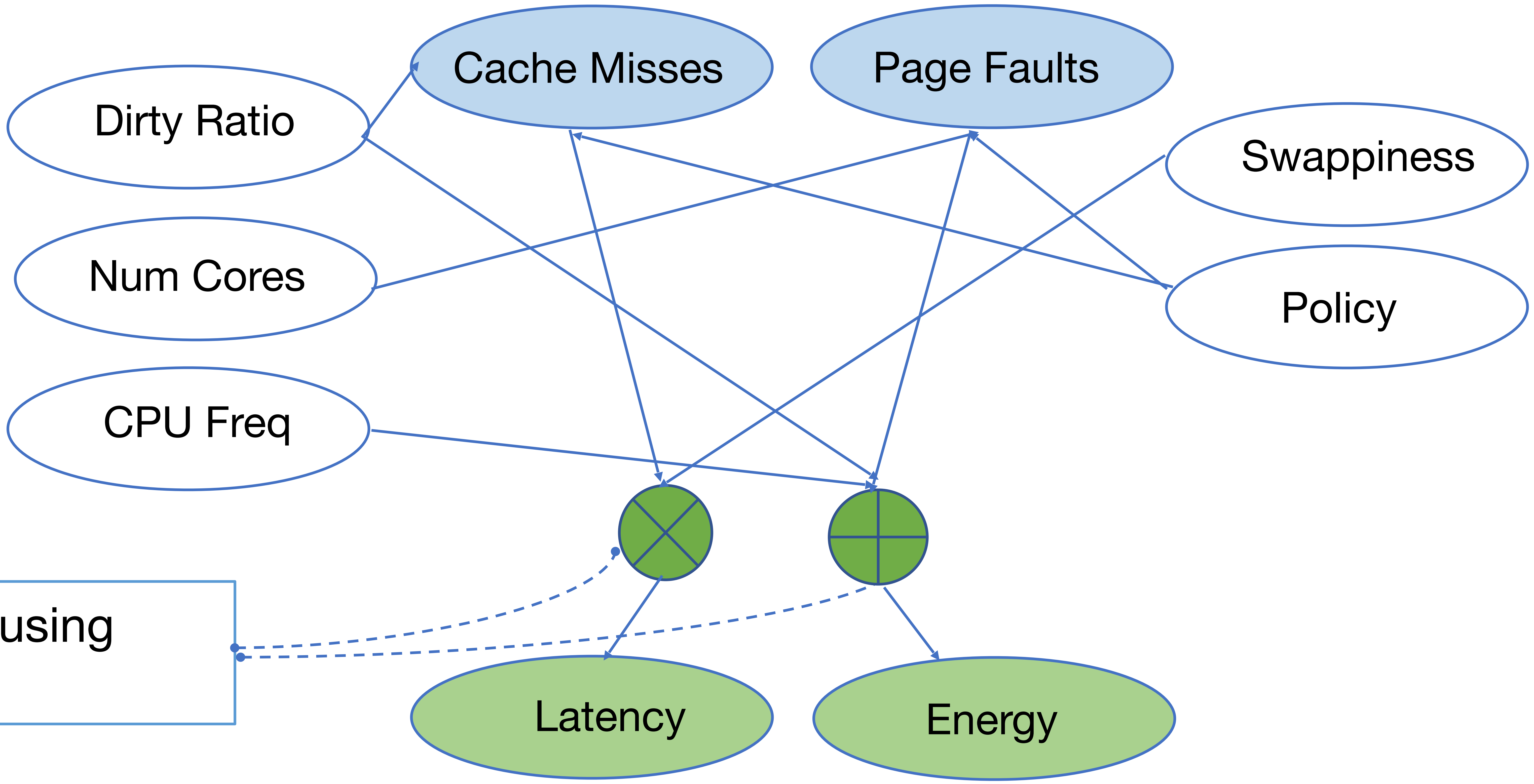
	CADET	Decision Tree	Forum
Throughput (on TX2)	26 FPS	20 FPS	23 FPS
Throughput Gain (over TX1)	53 %	21 %	39 %
Time to resolve	24 min.	$3^1/_2$ Hrs.	2 days

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

Future Work : Update Causal Graph with Functional Nodes



Discover relationships using functional nodes

Improves accuracy of the causal graph as it learns options interactions better

Future Work: Finding Resource Aware Fixes

- Finding resource aware ranked fixes for performance faults

"When I test the TX2 board, use the model that comes with yolo-v2-tiny to recognize a single video plays normally, but it has a stutter if playing 4 or more videos. How to solve this problem?"

Fix for TX2:

- Set sync to 0
- Set interval to 1 to apply inference periodically

Fix for Xavier:

- **Run on DLA instead of GPU**
- **More energy efficient**

Better fix as it solves the latency issue and improves energy that the developer is not actively concern of

CADET: A Systematic Method For Debugging Misconfigurations using Counterfactual Reasoning

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

Rahul Krishna*
Columbia University
rahul.krishna@columbia.edu

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 computing platforms are highly-configurable with thousands of interacting configuration options. However, configuring these systems is challenging. Erroneous configurations can cause unexpected non-functional faults. This paper proposes CADET (short for Causal Performance Debugging) that enables users to *identify*, *explain*, and *fix* the root cause of non-functional faults early and in a

GitHub https://github.com/softsys4ai/CADET

Automated Software Engineering
https://doi.org/10.1007/s10515-020-00273-8

Check for updates

ConfigCrusher: towards white-box performance analysis for configurable systems

Miguel Velez[1] · Pooyan Jamshidi[2] · Florian Sattler[3] · Norbert Siegmund[4] · Sven Apel[3] · Christian Kästner[1]

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, SUBMITTED MAY '19

Whence to Learn? Transferring Knowledge in Configurable Systems using BEETLE

Rahul Krishna, Vivek Nair, Pooyan Jamshidi, Tim Menzies, *IEEE Fellow*

An Uncertainty-Aware Approach to Optimal Configuration of Stream Processing Systems

Pooyan Jamshidi
Imperial College London, UK
Email: p.jamshidi@imperial.ac.uk

Giuliano Casale
Imperial College London, UK
Email: g.casale@imperial.ac.uk

Team effort


Rahul Krishna
Columbia


Shahriar Iqbal
UofSC


M. A. Javidian
Purdue


Baishakhi Ray
Columbia


Christian Kästner
CMU


Miguel Velez
CMU


Norbert Siegmund
Leipzig


Sven Apel
Saarland

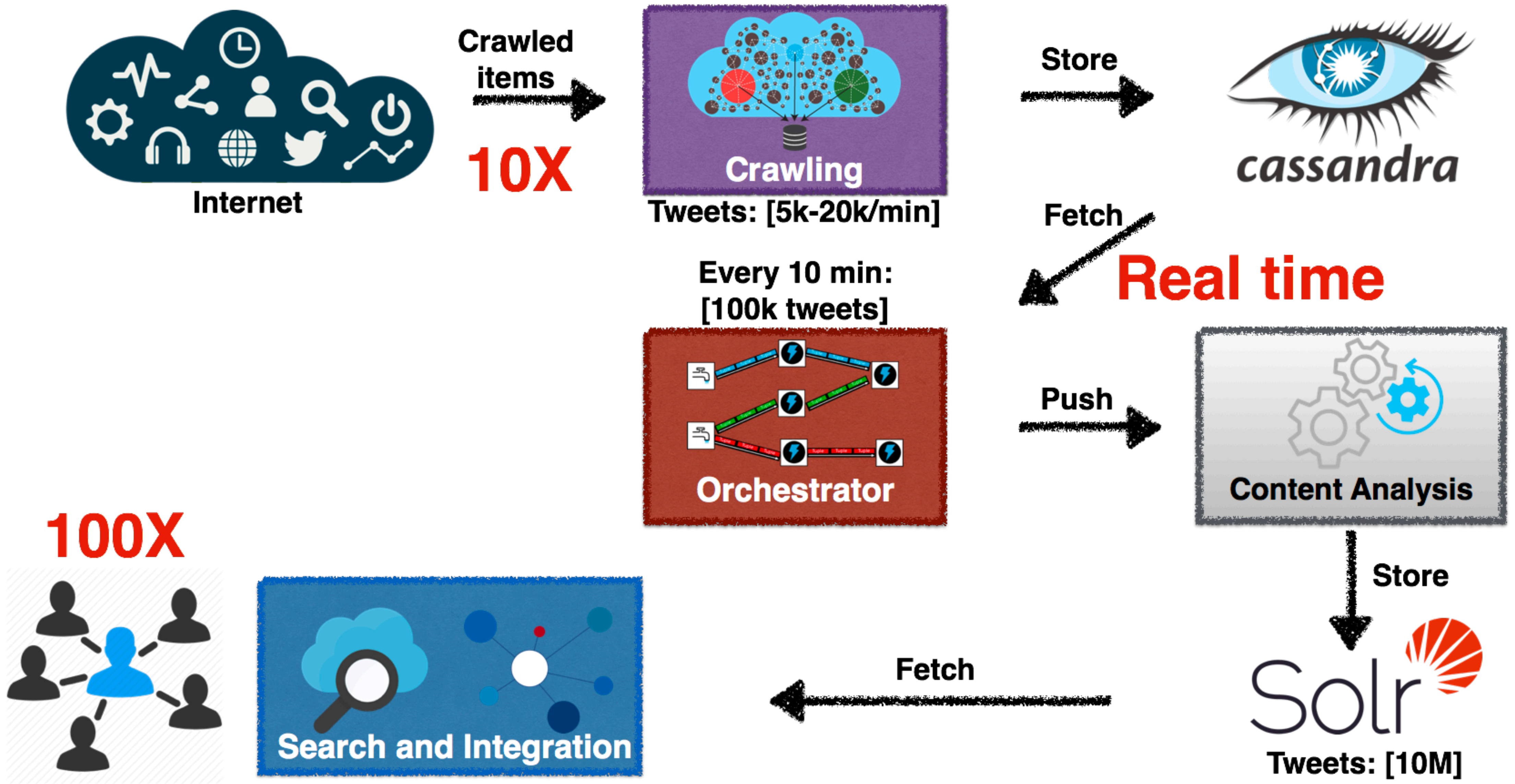

Lars Kotthoff
Wyoming


Marco Valtorta
UofSC

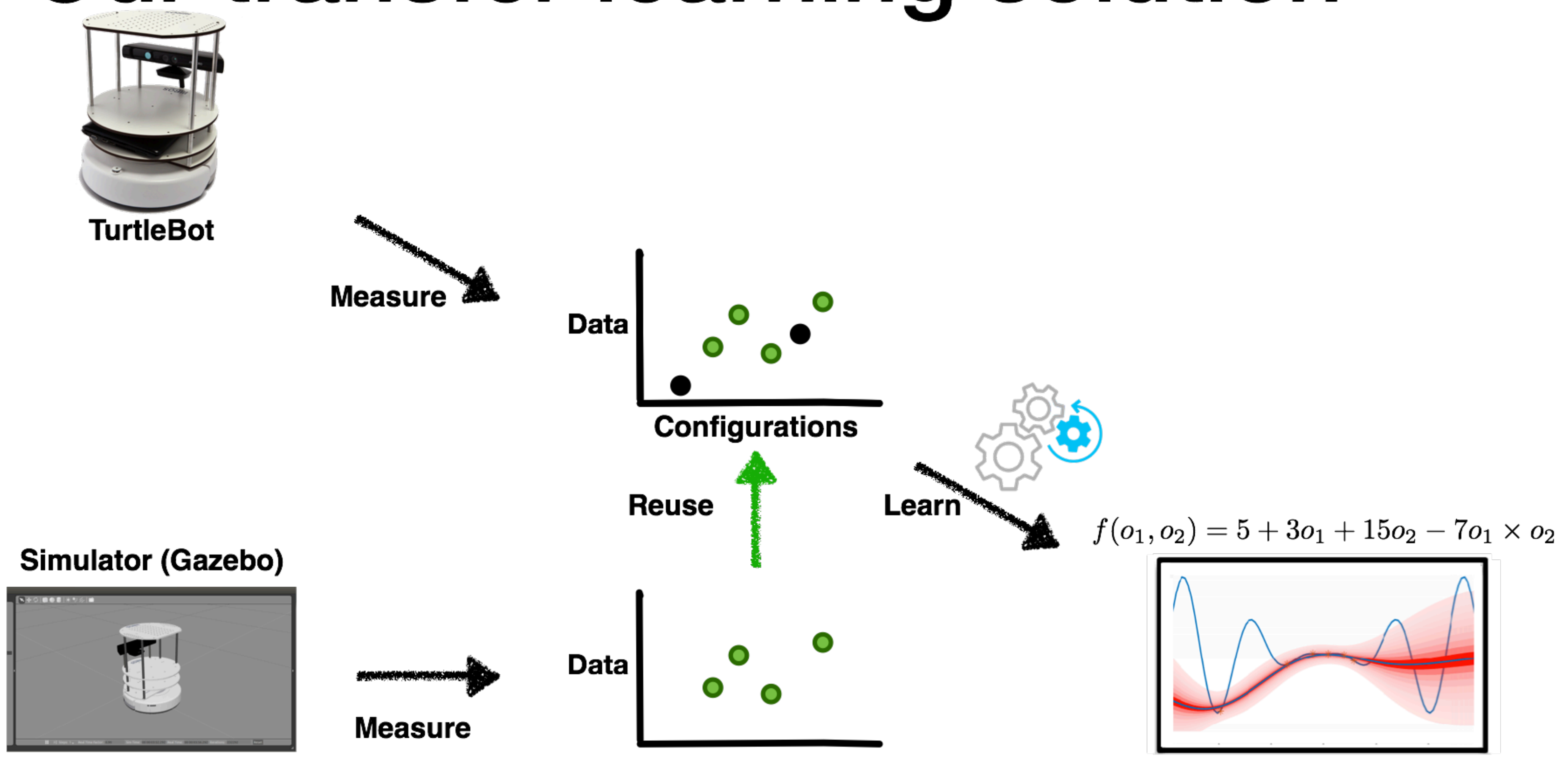

Vivek Nair
Facebook


Tim Menzies
NCSU

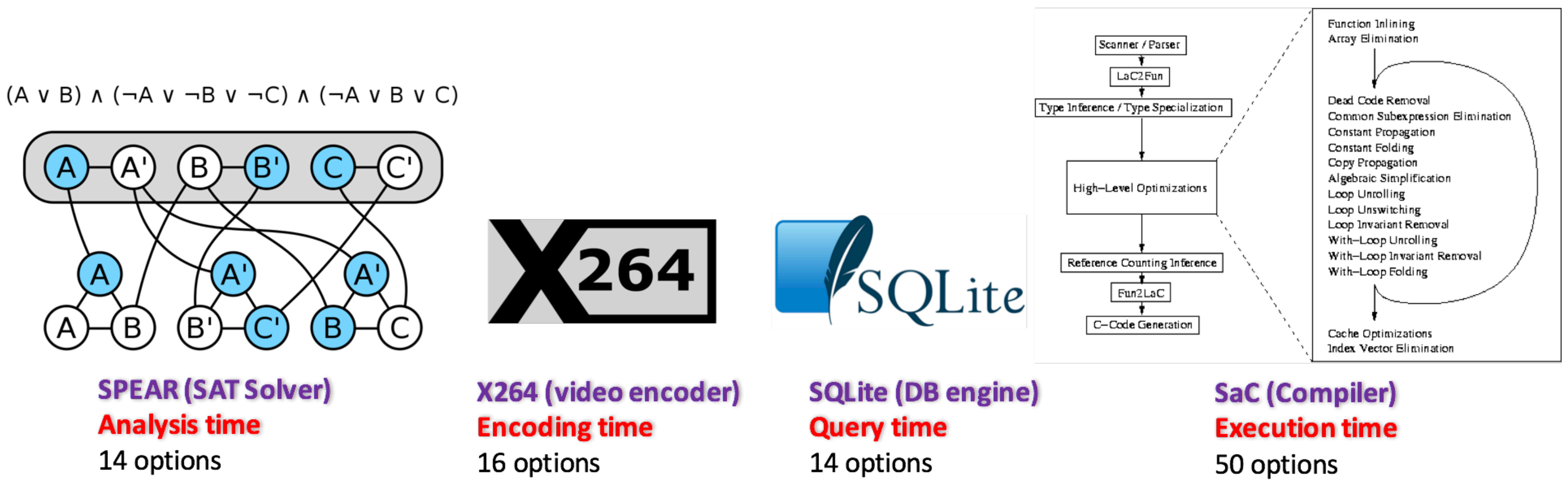
Challenges



Our transfer learning solution



Our empirical study: We looked at different highly-configurable systems to gain insights



Exploring the design space of deep networks