

# **Transfer Learning for Performance Analysis of Machine Learning Systems**

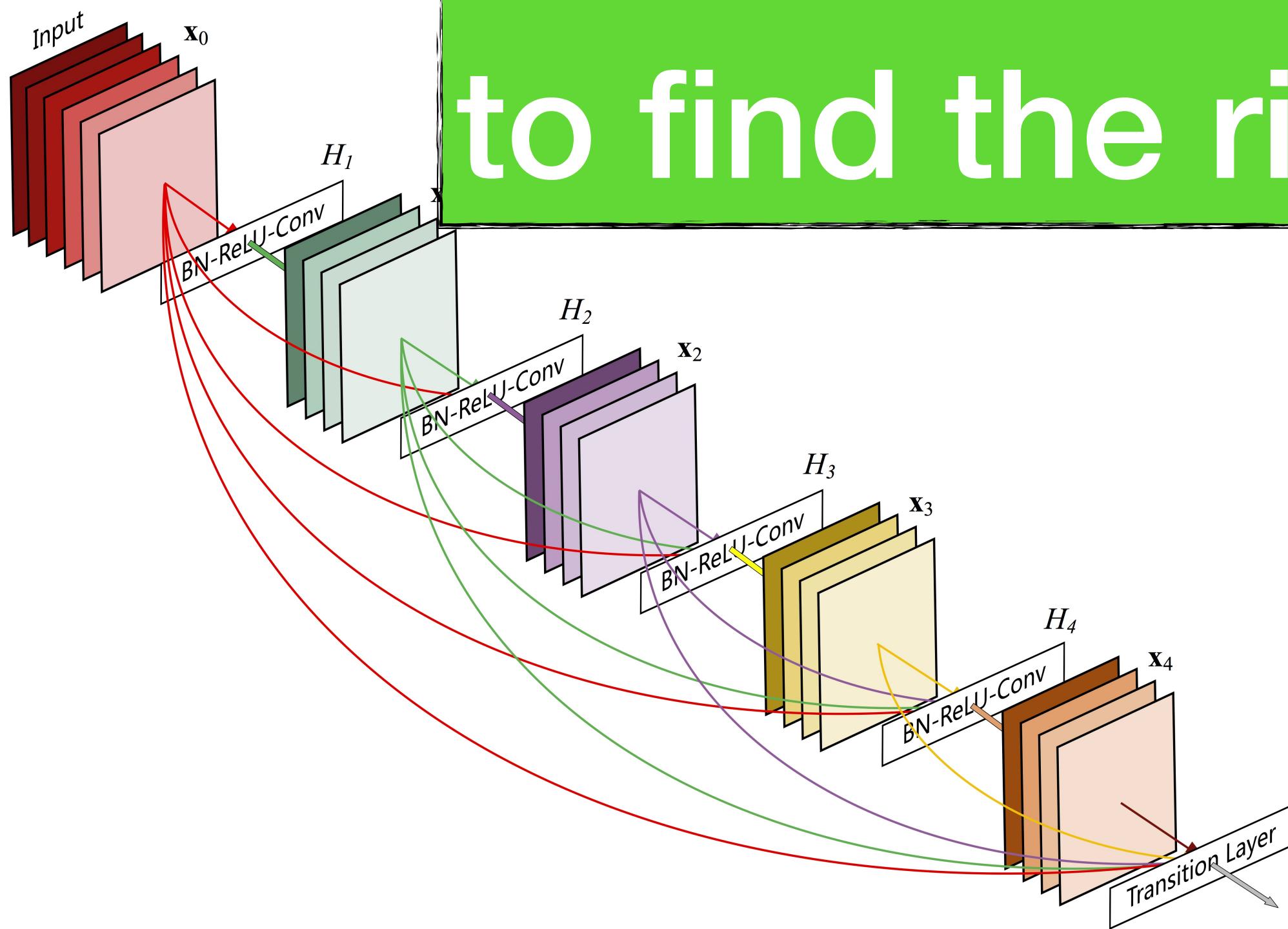
Pooyan Jamshidi  
University of South Carolina



[@pooyanjamshidi](https://twitter.com/pooyanjamshidi)



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



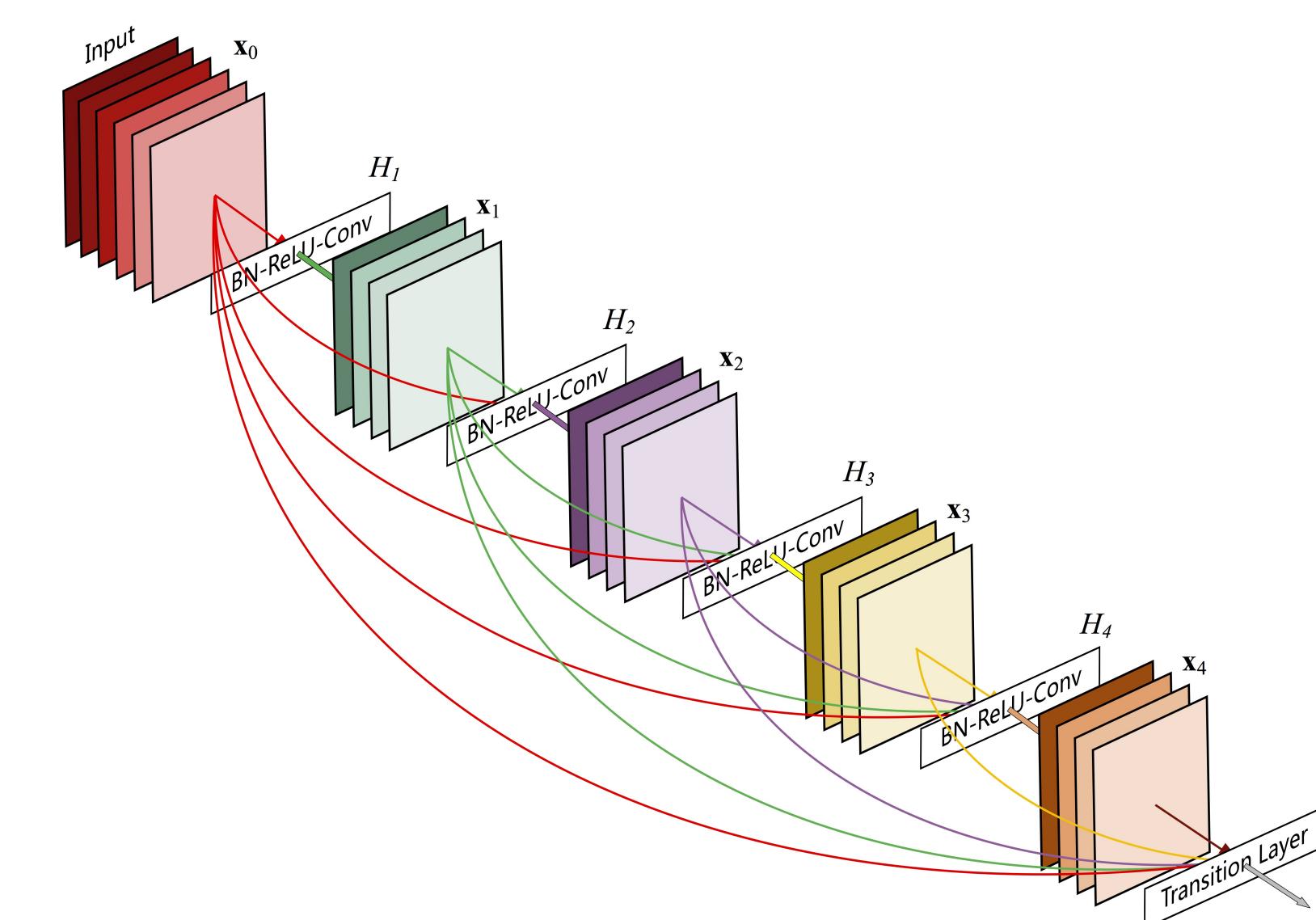
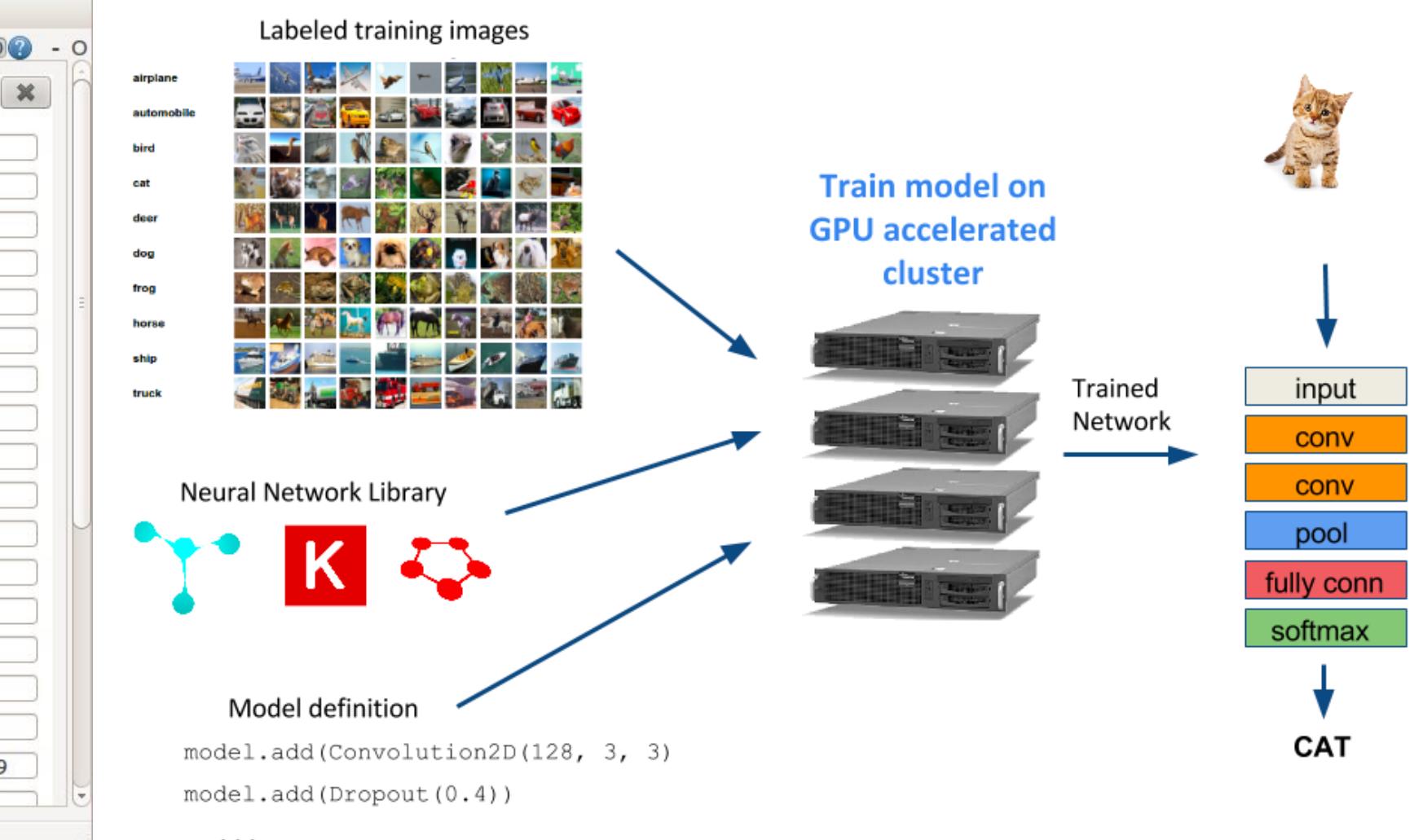
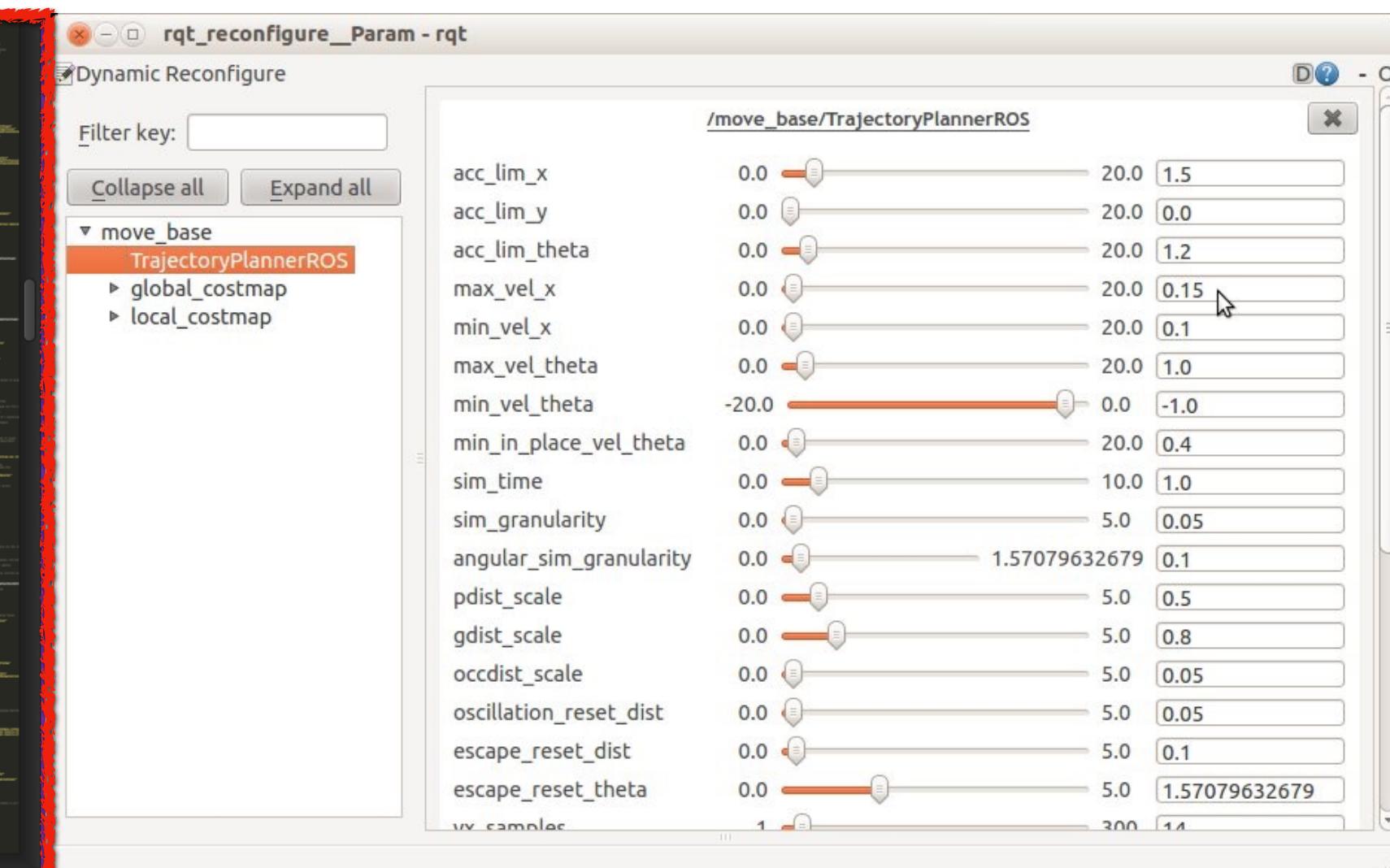
*built*

# Today's most popular systems are configurable

```

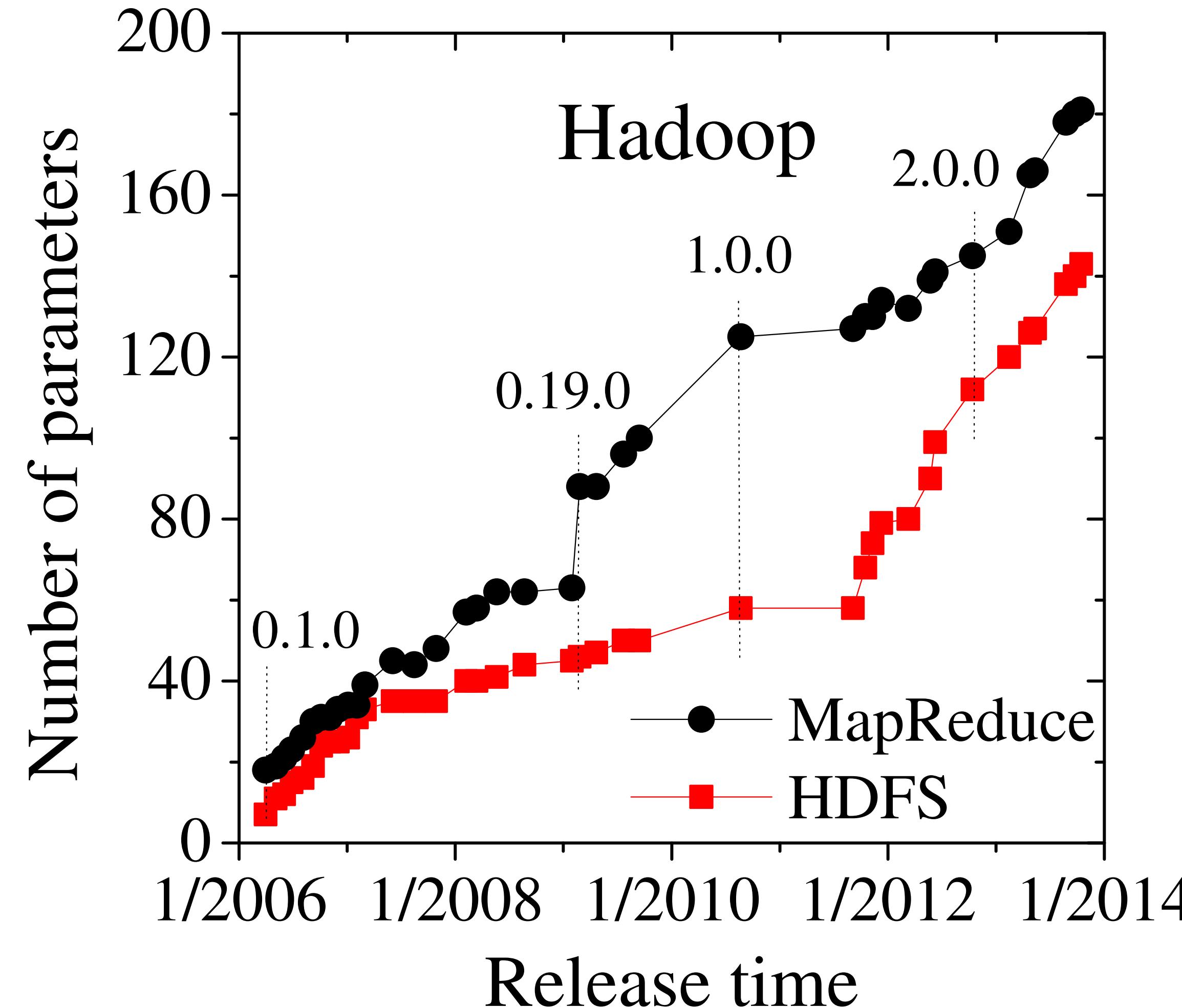
102
103 drpc.port: 3772
104 drpc.worker.threads: 64
105 drpc.max_buffer_size: 1048576
106 drpc.queue.size: 128
107 drpc.invocations.port: 3773
108 drpc.invocations.threads: 64
109 drpc.request.timeout.secs: 600
110 drpc.childopts: "-Xmx768m"
111 drpc.http.port: 3774
112 drpc.https.port: -1
113 drpc.https.keystore.password: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

```

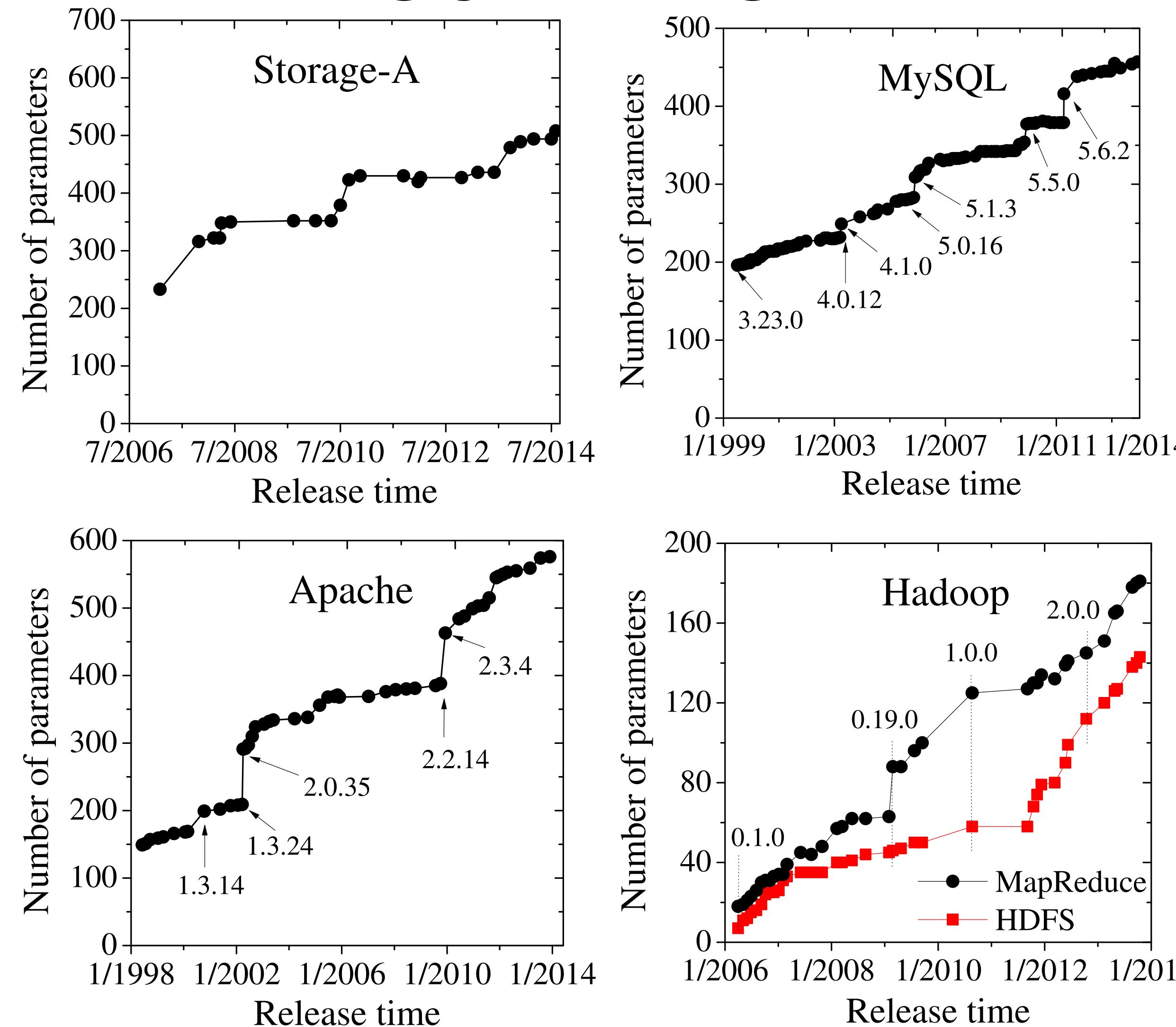


```
102  
103 drpc.port: 3772  
104 drpc.worker.threads: 64  
105 drpc.max_buffer_size: 1048576  
106 drpc.queue.size: 128  
107 drpc.invocations.port: 3773  
108 drpc.invocations.threads: 64  
109 drpc.request.timeout.secs: 600  
110 drpc.childopts: "-Xmx768m"  
111 drpc.http.port: 3774  
112 drpc.https.port: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

# Empirical observations confirm that systems are becoming increasingly configurable



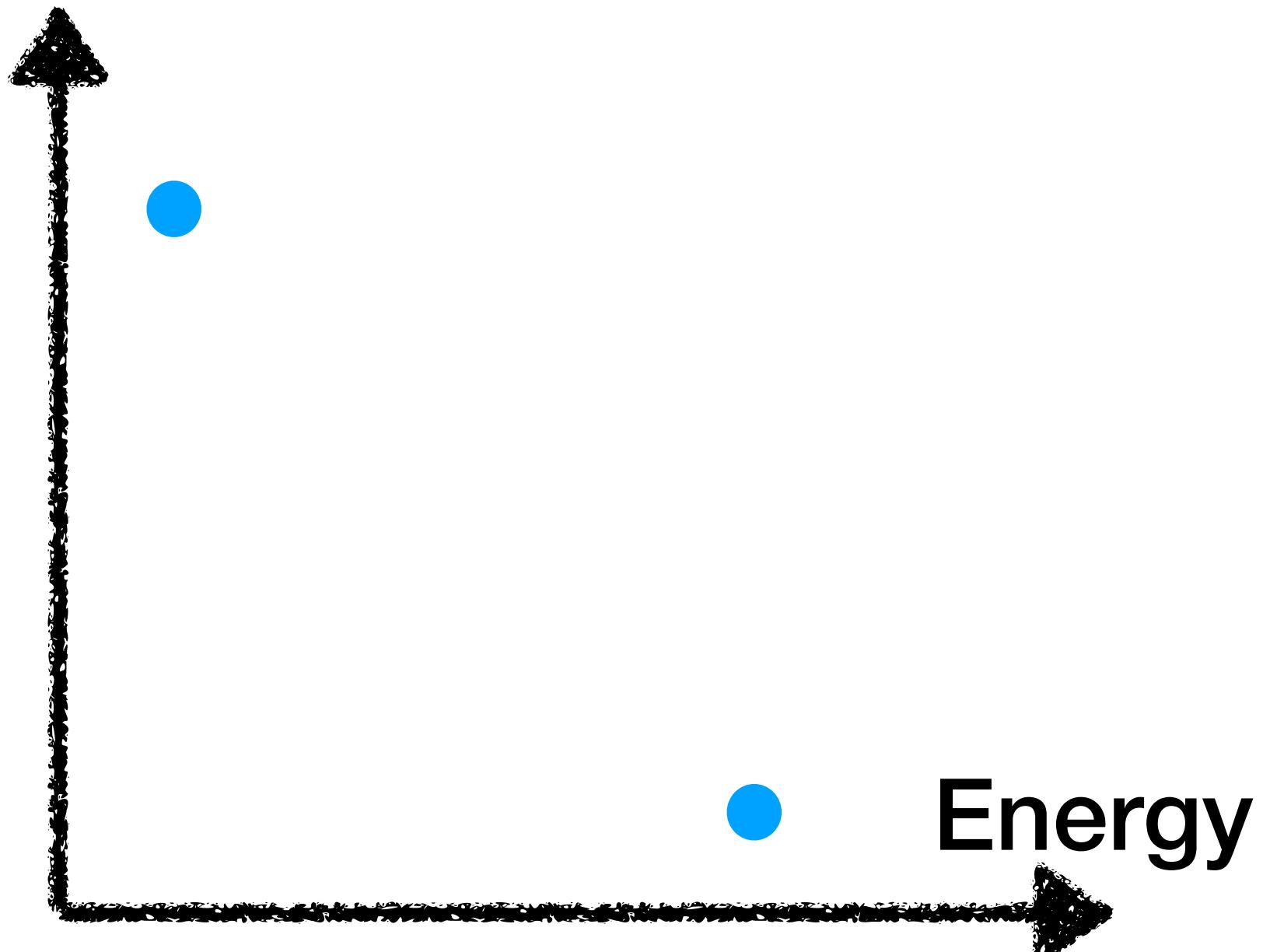
# Empirical observations confirm that systems are becoming increasingly configurable



# Configurations determine the performance behavior

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

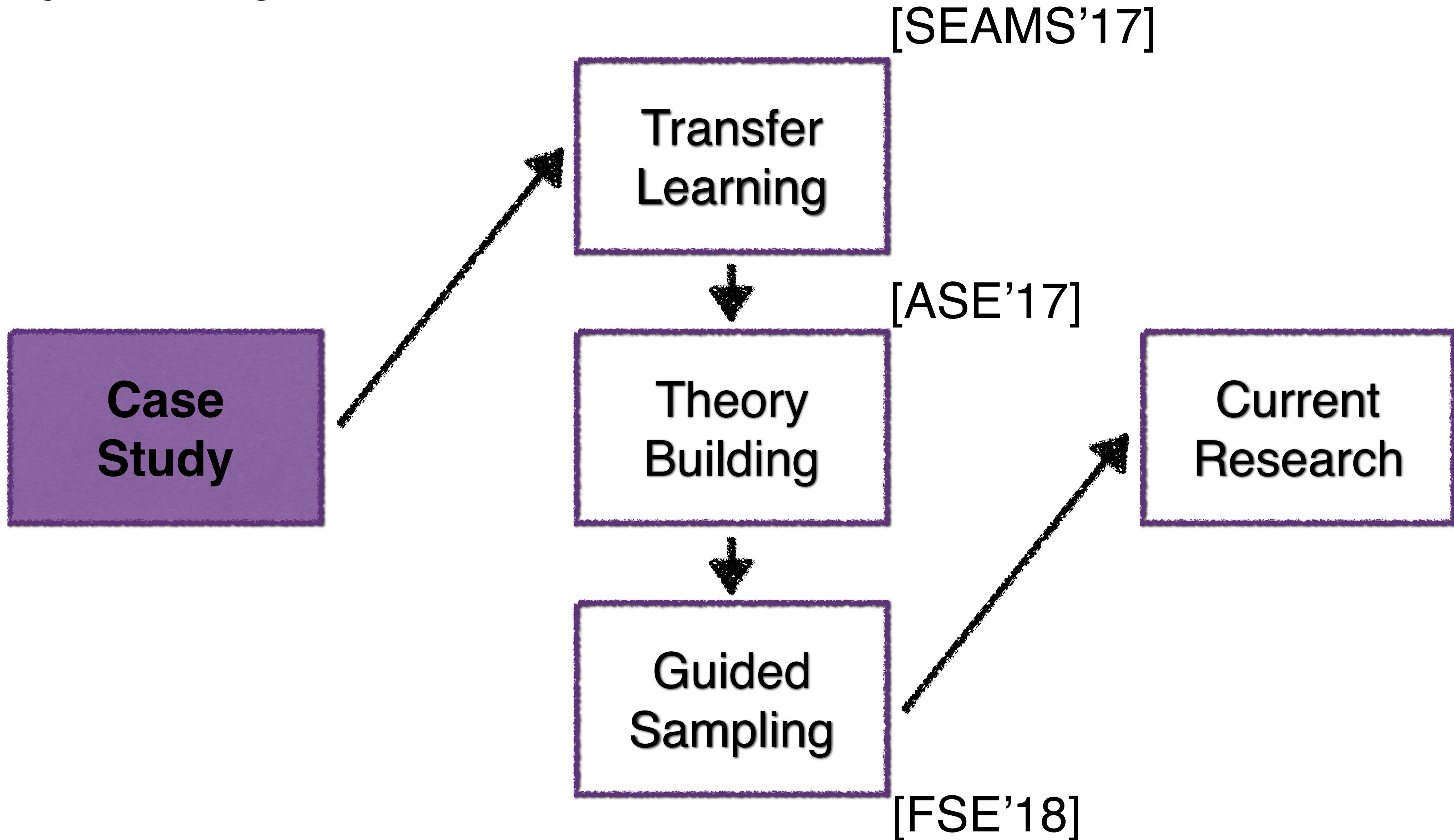
Speed



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?**

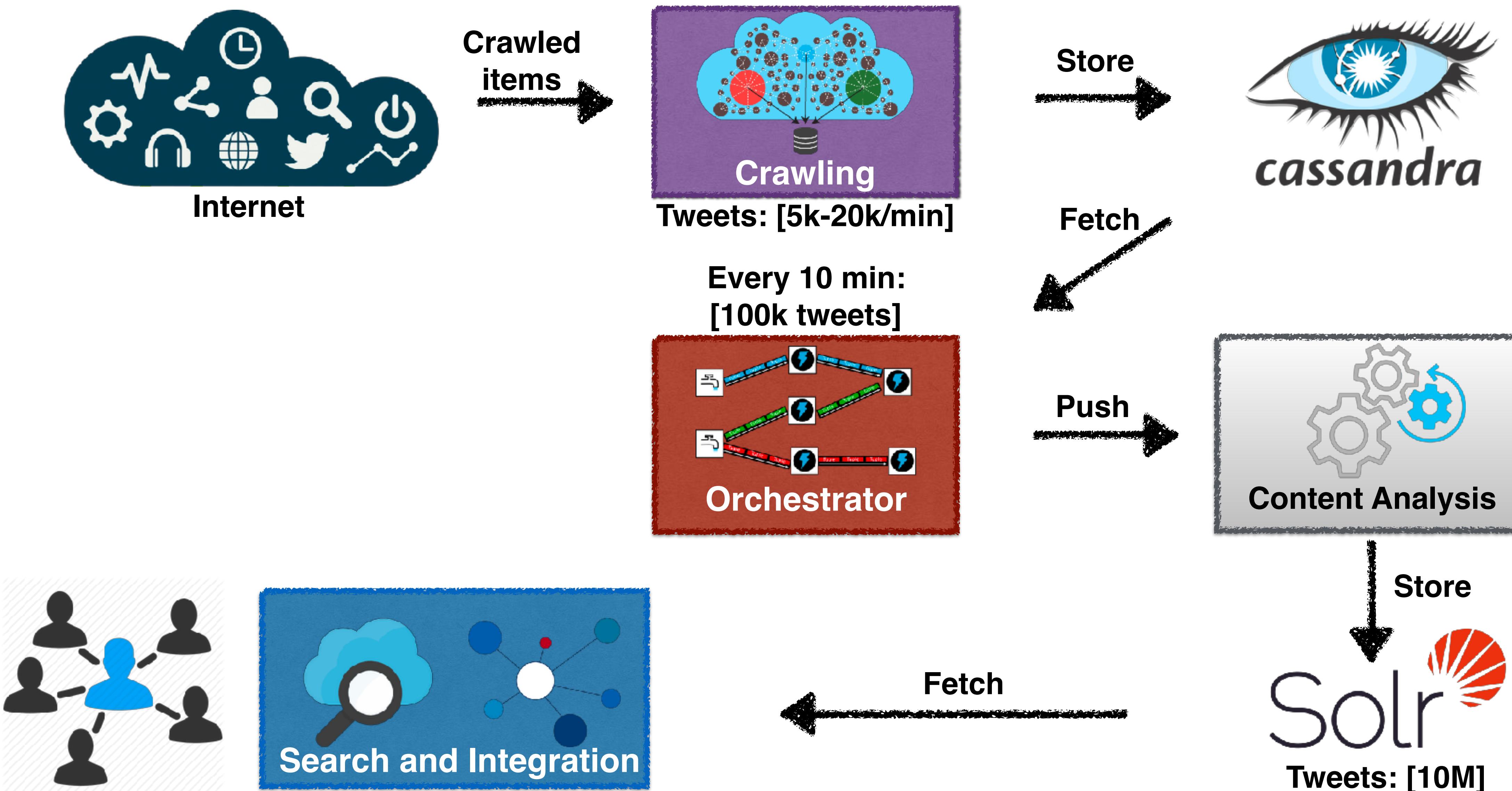
# Outline



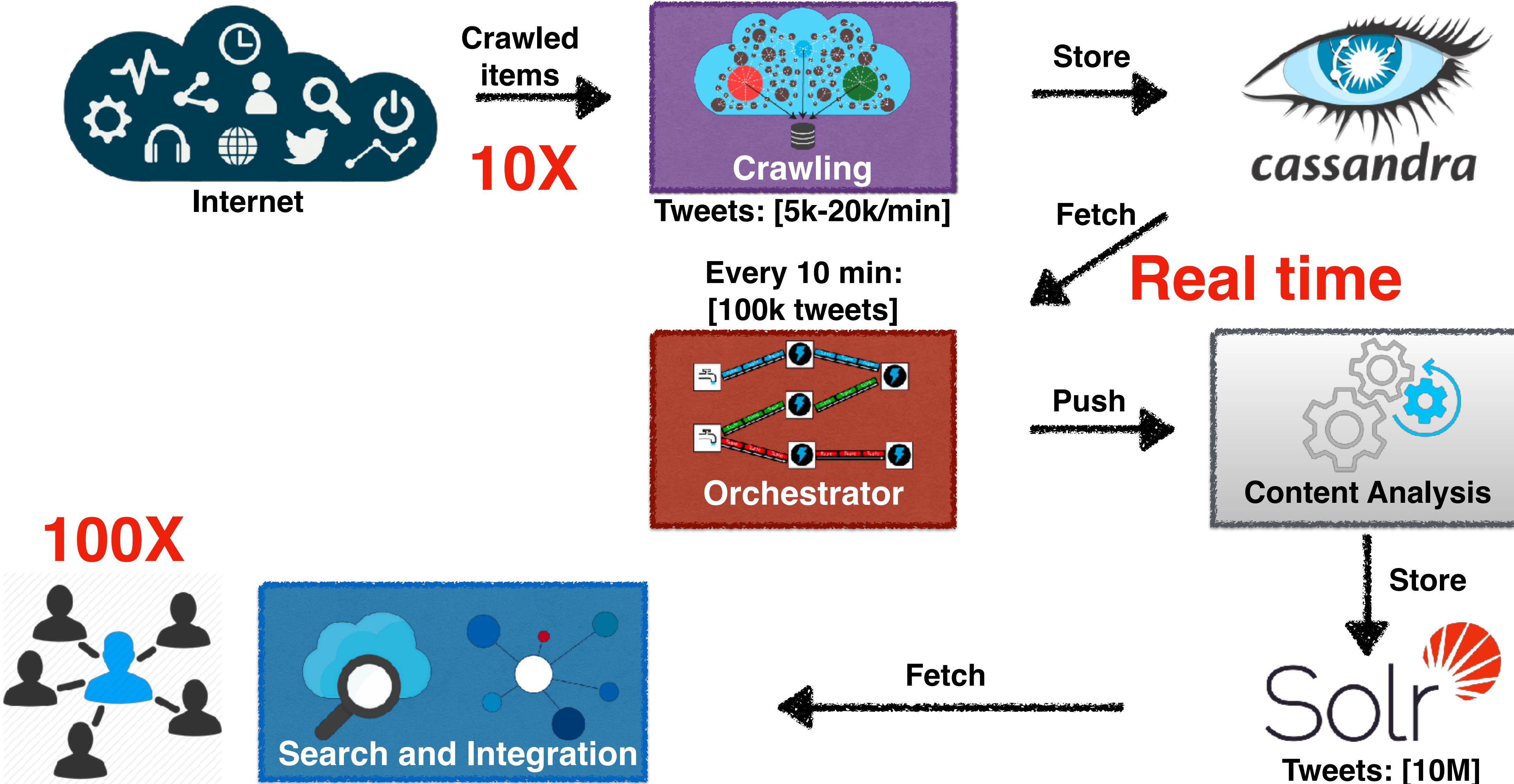
# SocialSensor

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

# SocialSensor



# Challenges



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

**Let's try out different system configurations!**

# Opportunity: Data processing engines in the pipeline were all configurable



> 100



> 100

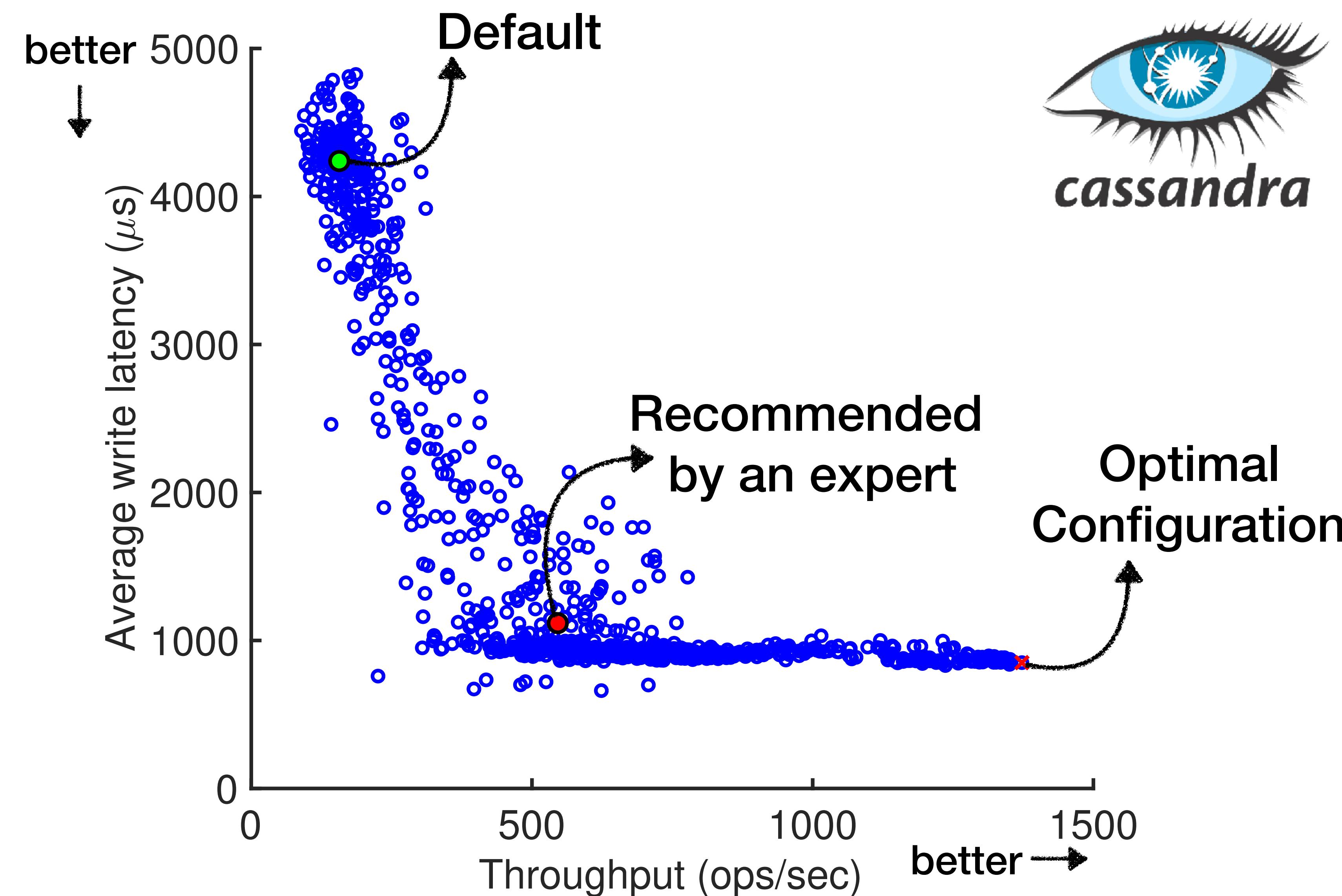


> 100

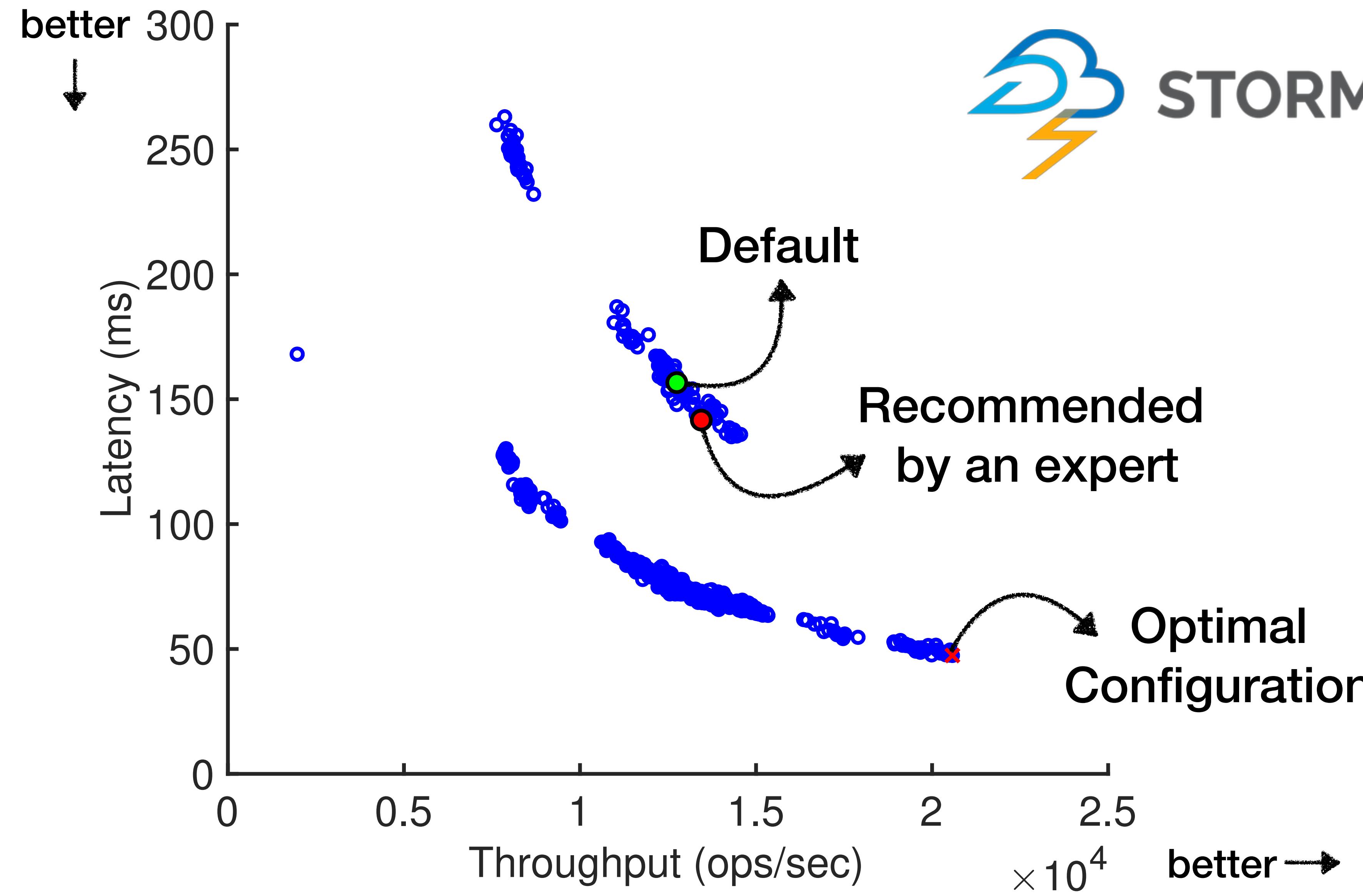
$2^{300}$



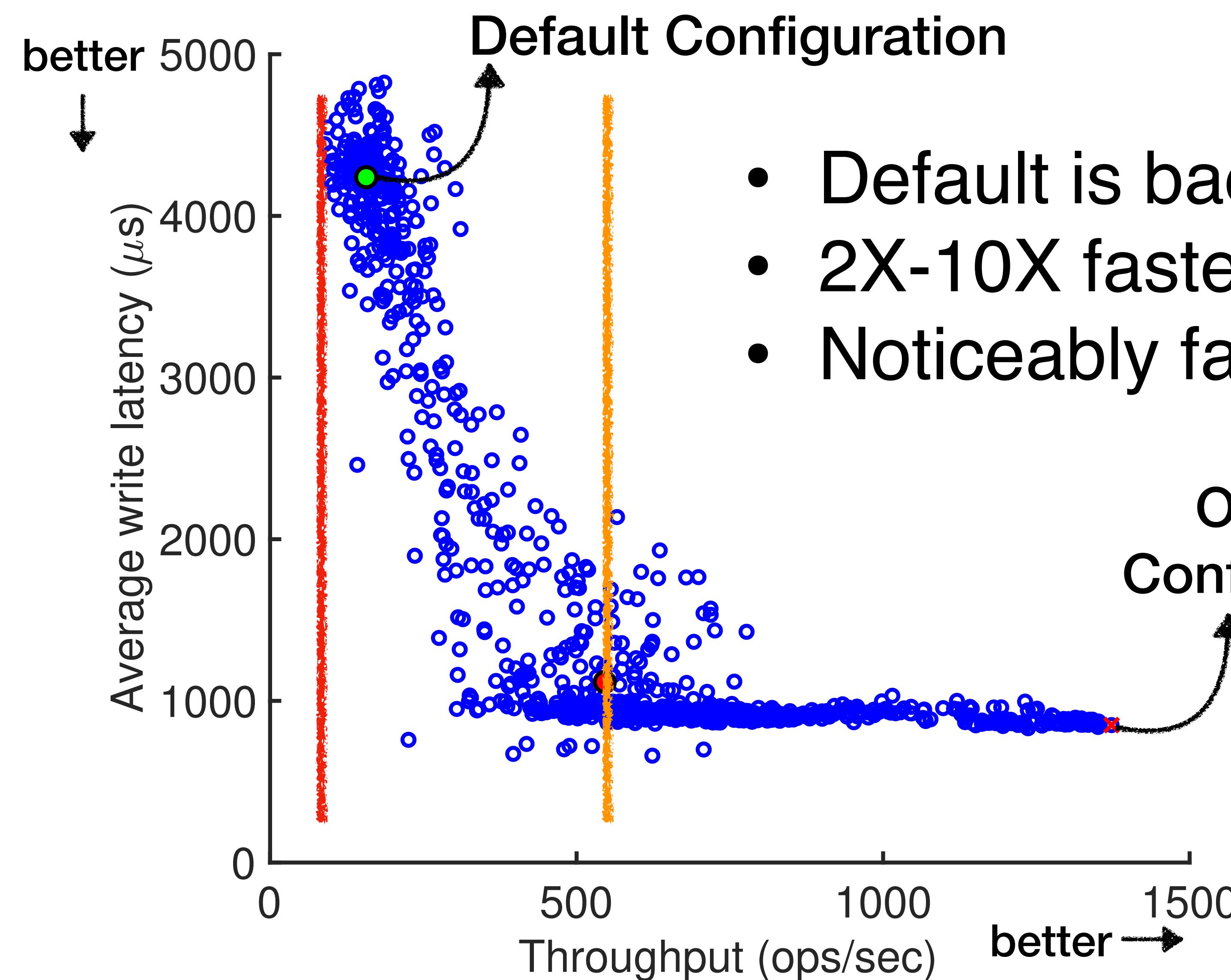
# Default configuration was bad, so was the expert'



# Default configuration was bad, so was the expert'



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

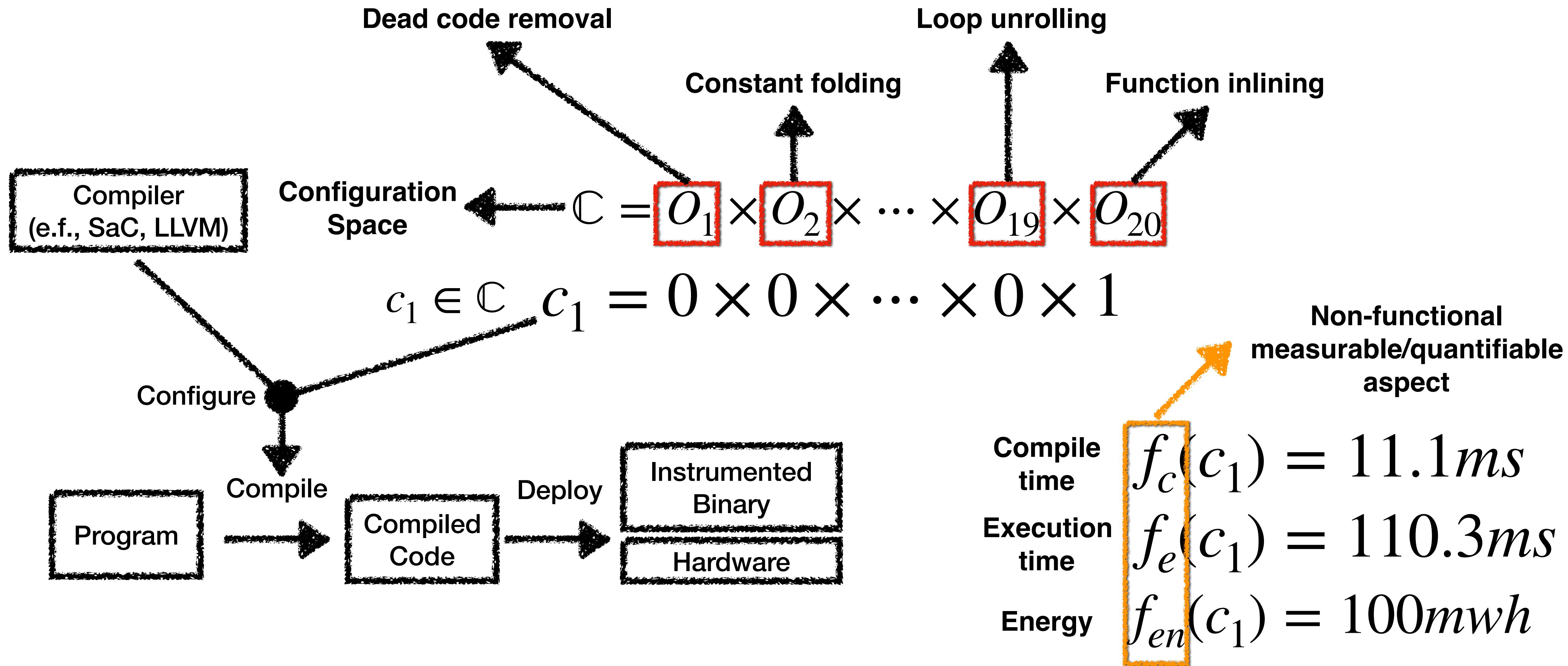


- Default is bad
- 2X-10X faster than worst
- Noticeably faster than median

# What did happen at the end?

- Achieved the objectives (**100X** user, same experience)
- Saved money by reducing cloud resources up to **20%**
- Our tool was able to identify configurations that was consistently better than expert recommendation

# Setting the scene



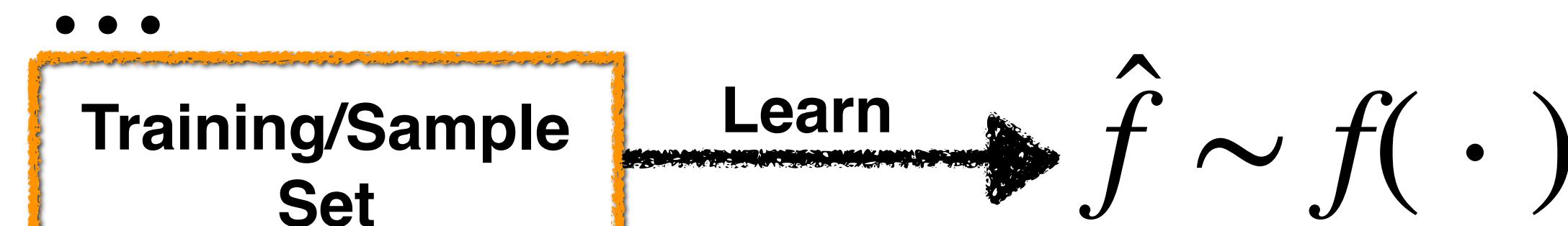
# A typical approach for understanding the performance behavior is sensitivity analysis

$O_1 \times O_2 \times \cdots \times O_{19} \times O_{20}$	
$c_1$	$0 \times 0 \times \cdots \times 0 \times 1$
$c_2$	$0 \times 0 \times \cdots \times 1 \times 0$
$c_3$	$0 \times 0 \times \cdots \times 1 \times 1$
	$\cdots$
	$1 \times 1 \times \cdots \times 1 \times 0$
$c_n$	$1 \times 1 \times \cdots \times 1 \times 1$

$$y_1 = f(c_1)$$

$$y_2 = f(c_2)$$

$$y_3 = f(c_3)$$



$$y_n = f(c_n)$$

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

	$O_1 \times O_2 \times \dots \times O_{19} \times O_{20}$
$c_1$	$0 \times 0 \times \dots \times 0 \times 1$
$c_2$	$0 \times 0 \times \dots \times 1 \times 0$
$c_3$	$0 \times 0 \times \dots \times 1 \times 1$
$\dots$	
$c_n$	$1 \times 1 \times \dots \times 1 \times 0$
	$1 \times 1 \times \dots \times 1 \times 1$

$$y_1 = f(c_1)$$

$$y_2 = f(c_2)$$

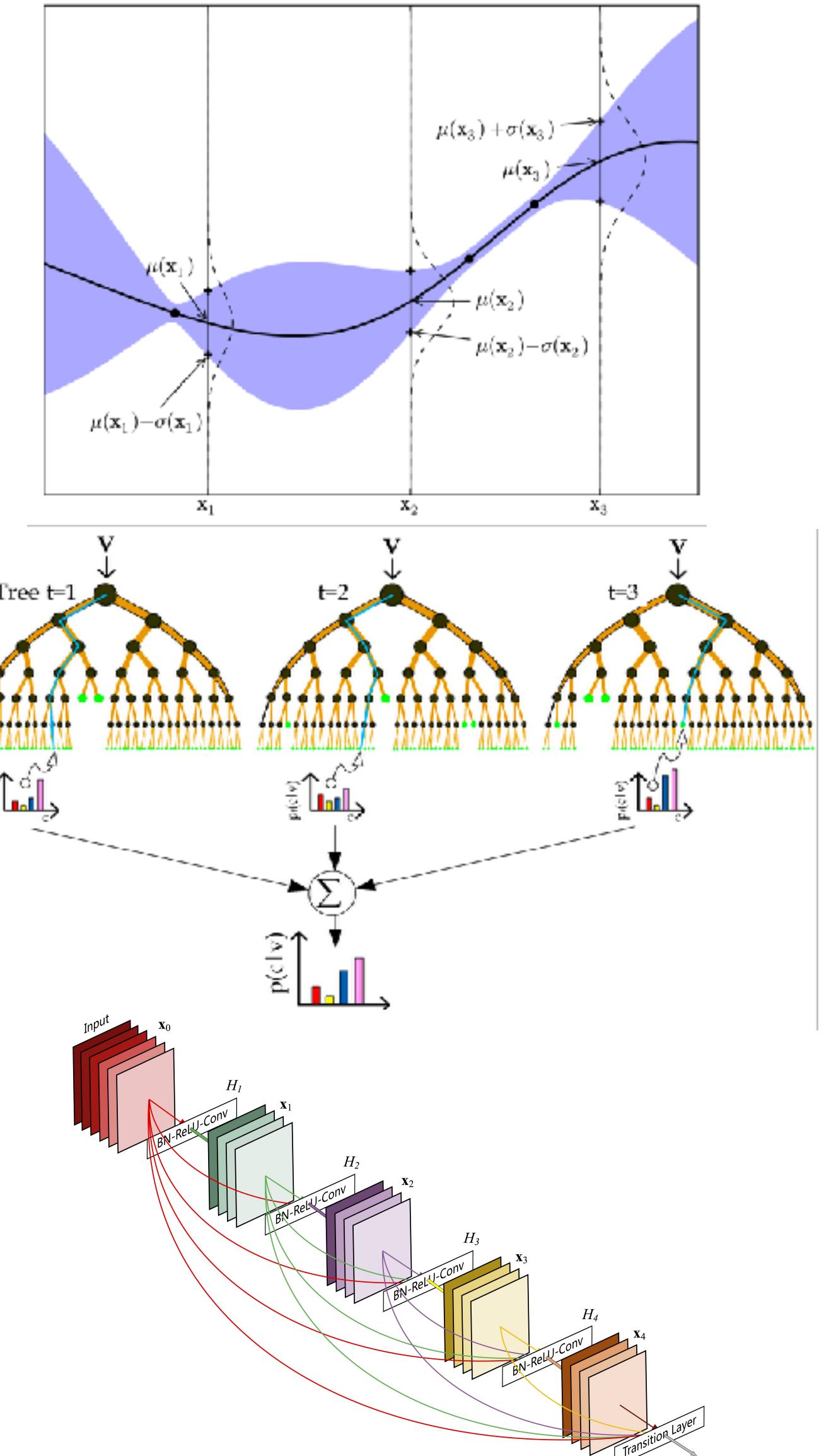
$$y_3 = f(c_3)$$

...

**Training/Sample Set**

$$y_n = f(c_n)$$

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



# Evaluating a performance model

$O_1 \times O_2 \times \cdots \times O_{19} \times O_{20}$	
$c_1$	$0 \times 0 \times \cdots \times 0 \times 1$
$c_2$	$0 \times 0 \times \cdots \times 1 \times 0$
$c_3$	$0 \times 0 \times \cdots \times 1 \times 1$
$\dots$	
$c_n$	$1 \times 1 \times \cdots \times 1 \times 0$
	$1 \times 1 \times \cdots \times 1 \times 1$

$y_1 = f(c_1)$
$y_2 = f(c_2)$
$y_3 = f(c_3)$
$\dots$
<b>Training/Sample Set</b>
$y_n = f(c_n)$

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

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

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

# 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  
}
```

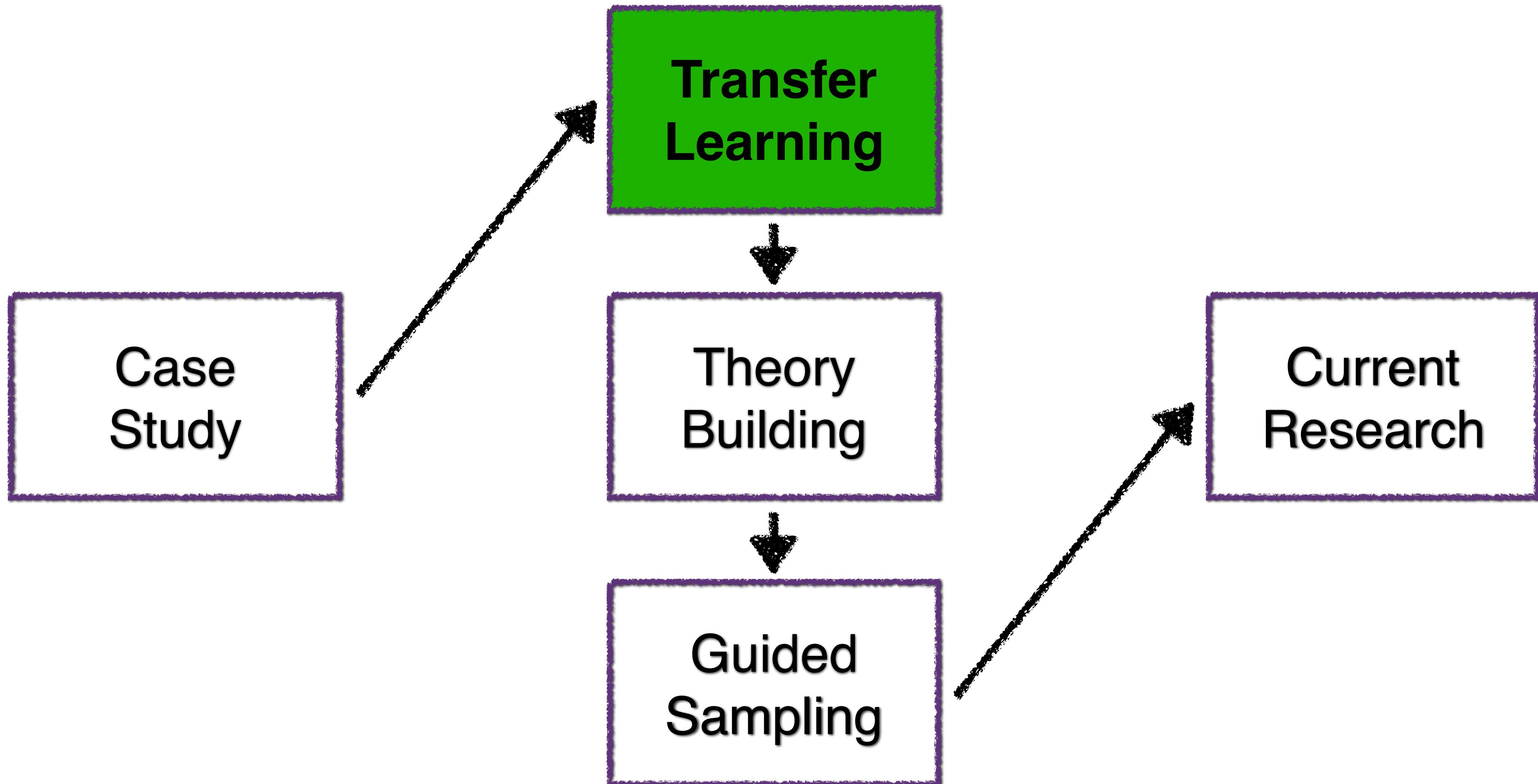
Execution time (s)

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

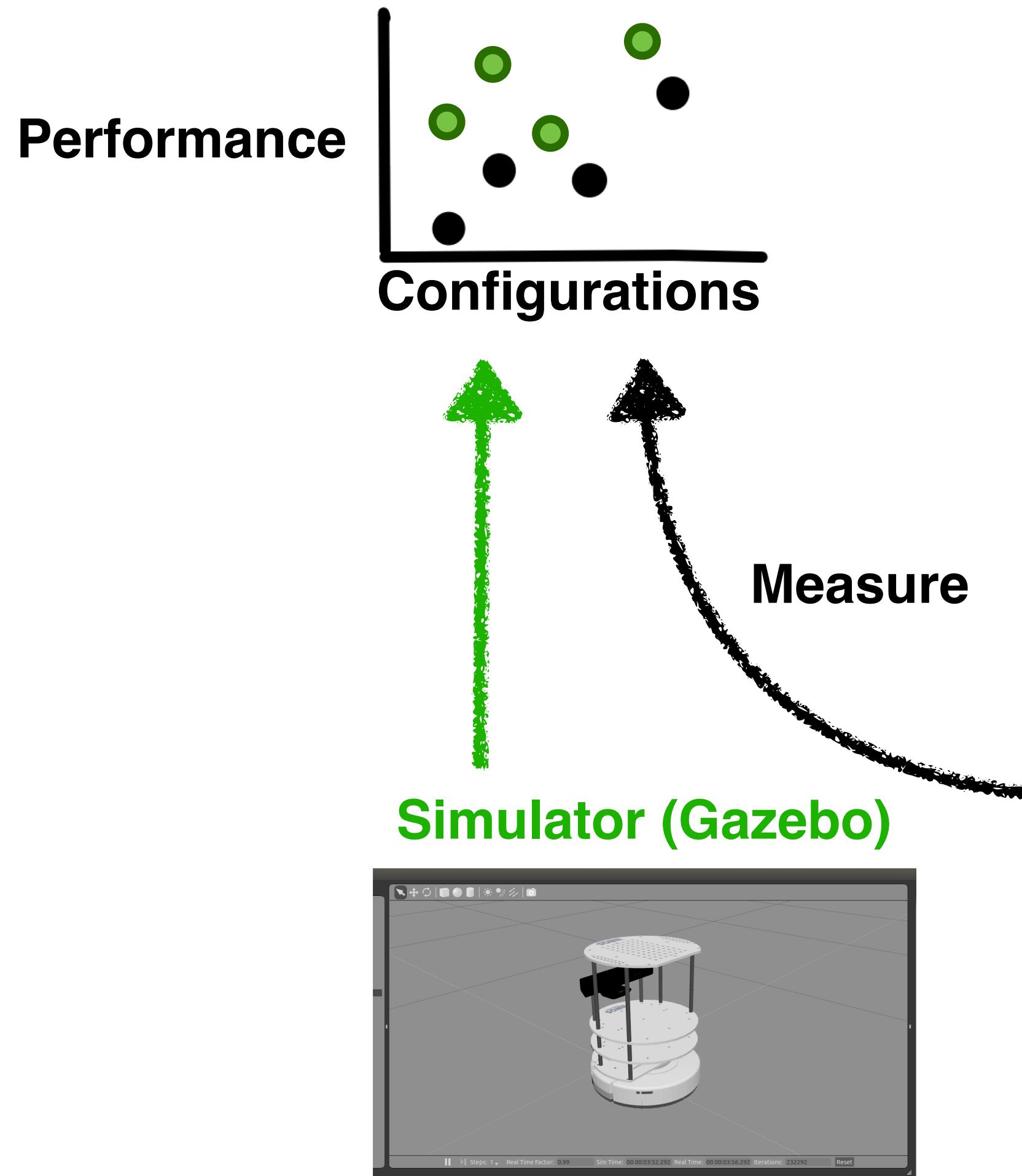
$$f(o_1 := 1) = 8$$

$$f(o_1 := 0) = 5$$

# Outline



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

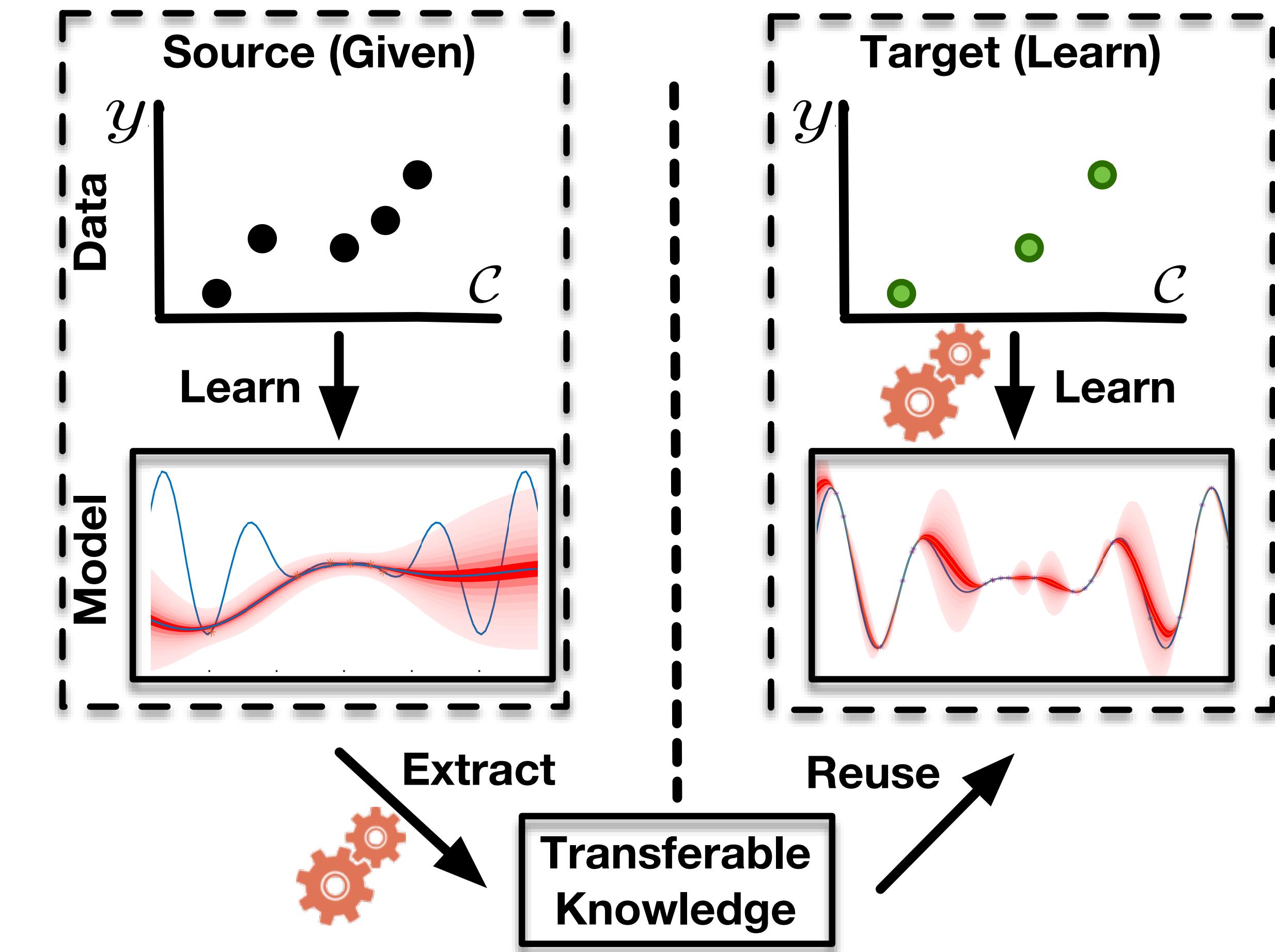


So why not reuse these data,  
instead of measuring on real robot?



# We developed methods to make learning cheaper via transfer learning

**Goal:** Gain strength by transferring information across environments



# What is transfer learning?



---

# What is transfer learning?

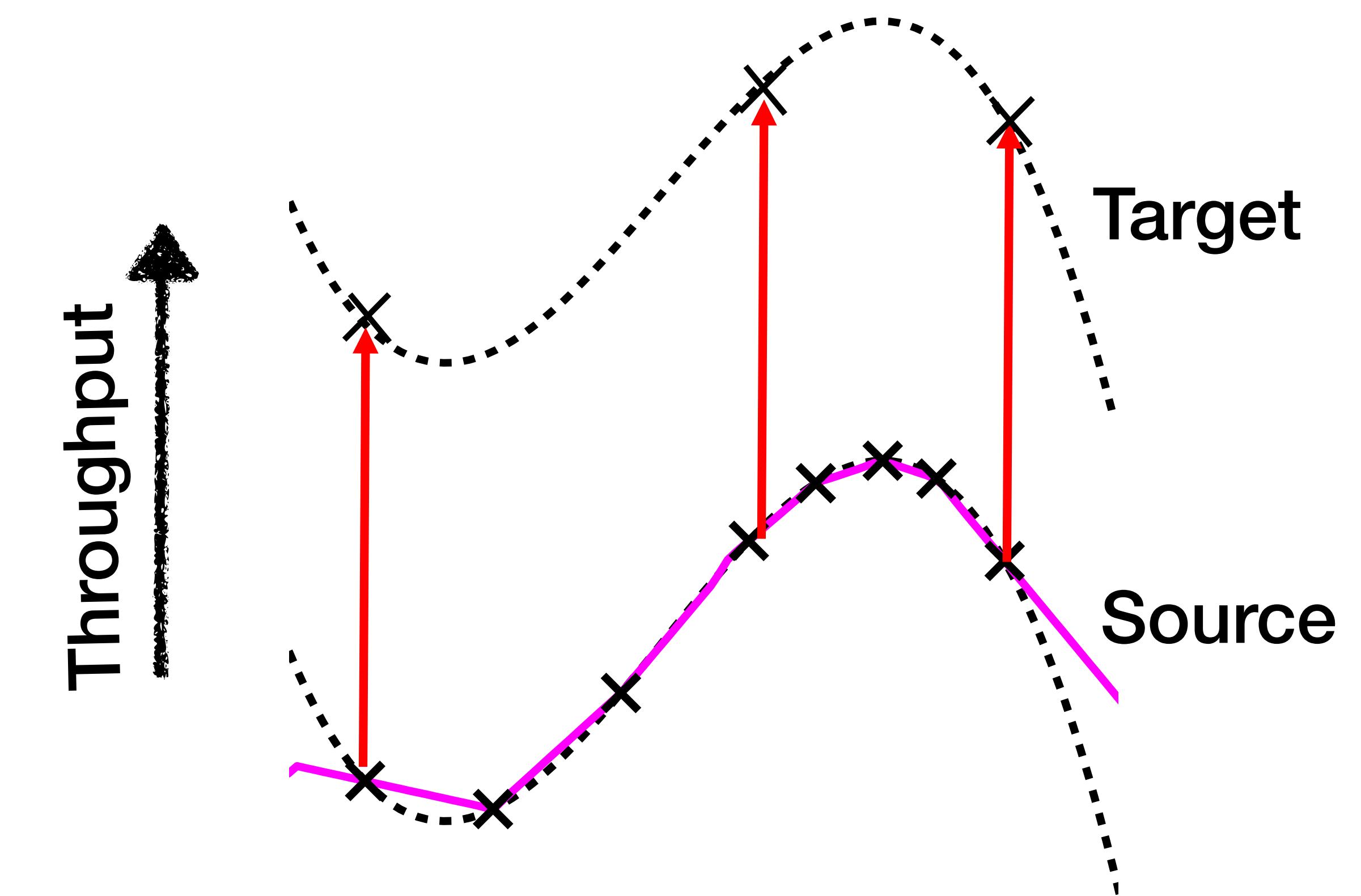
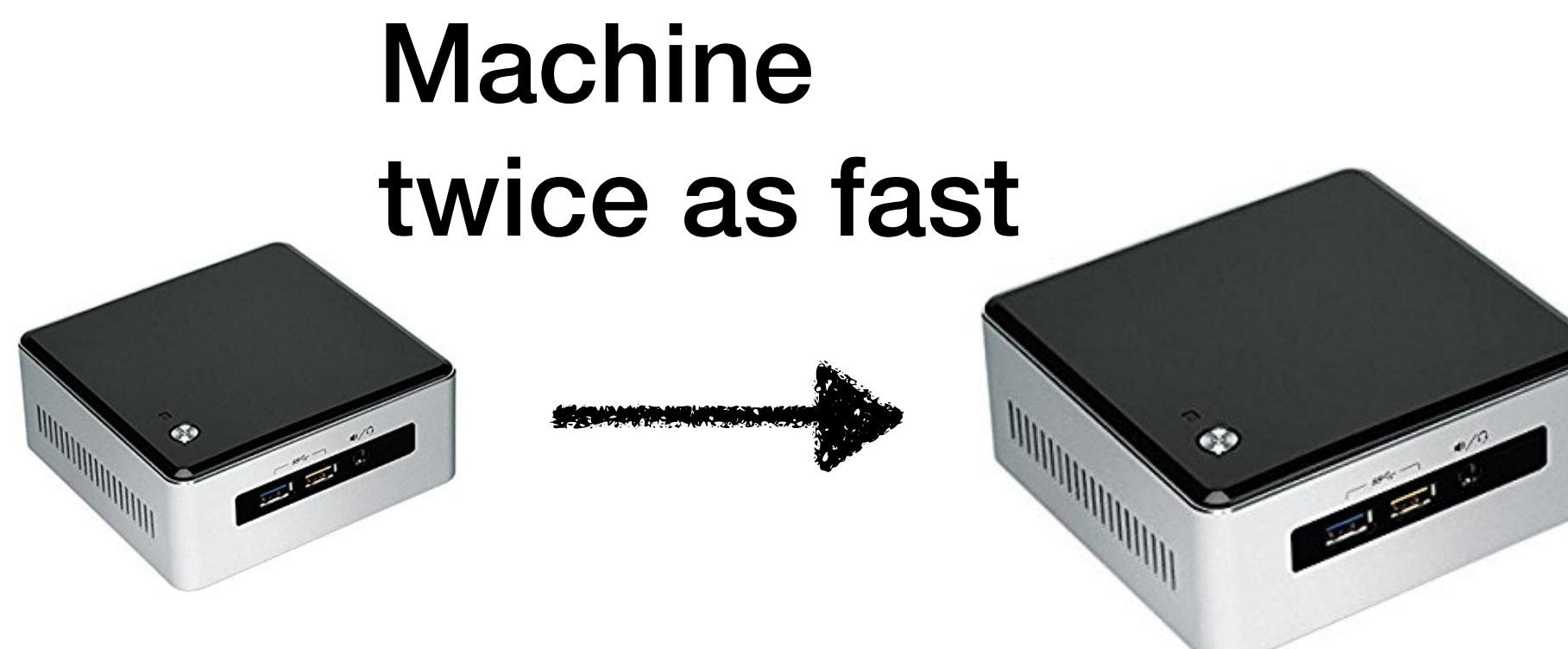


Transfer learning is a machine learning technique, where knowledge gain during training in one type of problem is used to train in other similar type of problem

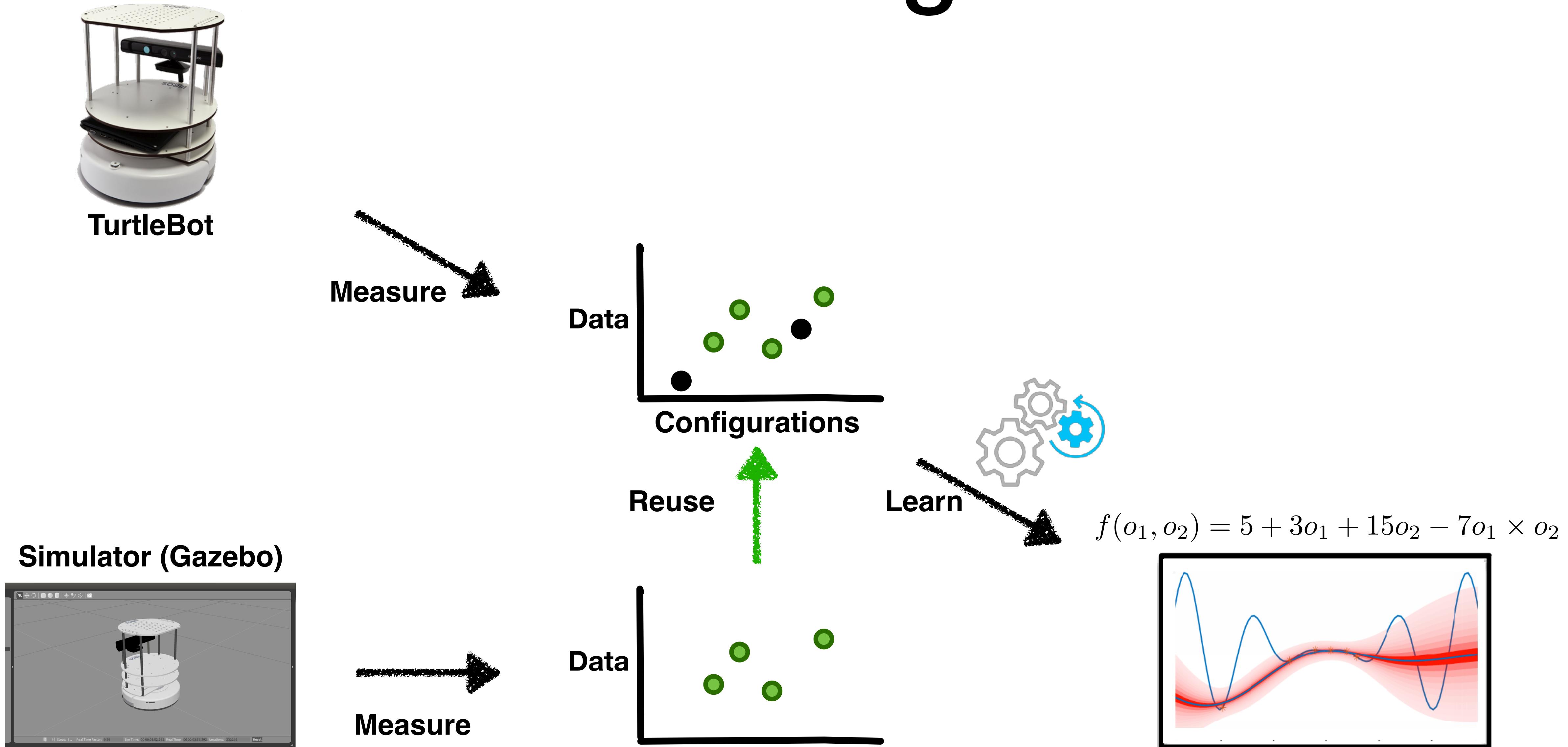
# What is the advantage of transfer learning?

- 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**.

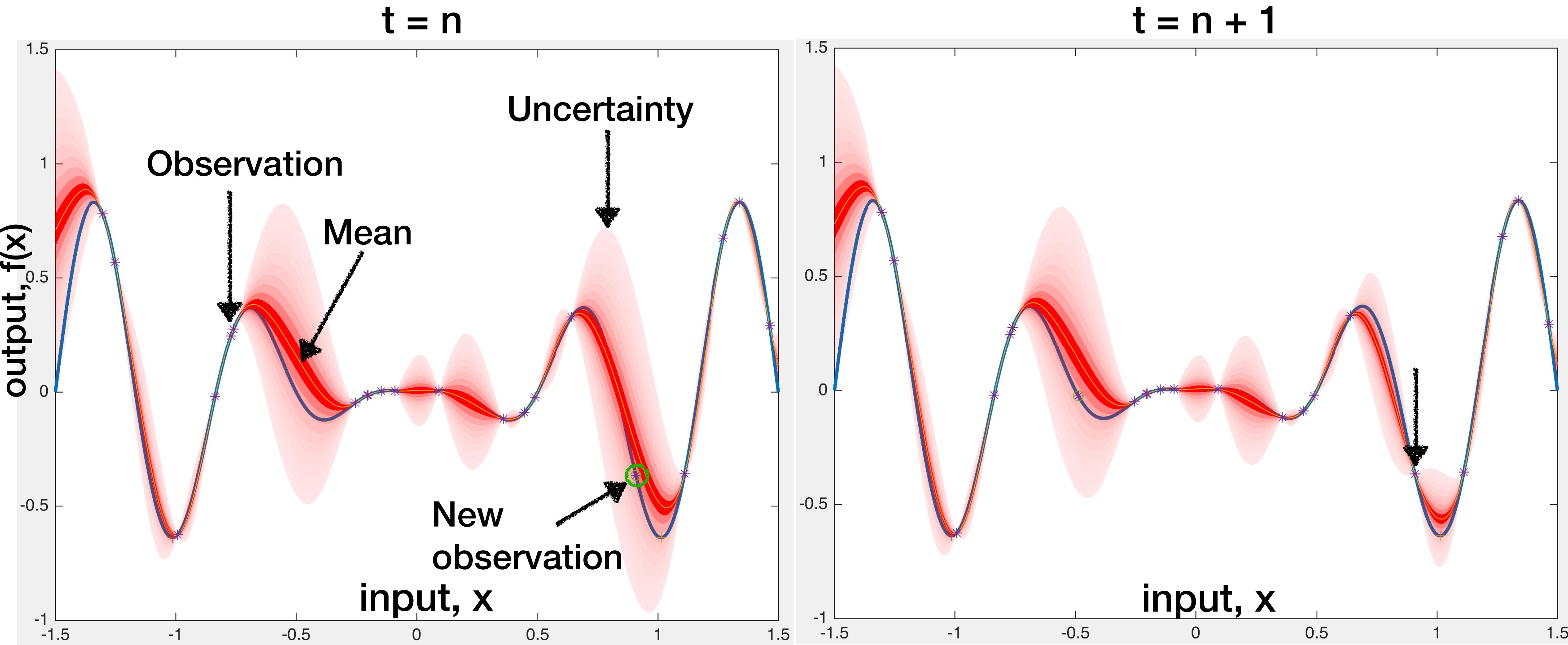
# A simple transfer learning via model shift



# Our transfer learning solution



# Gaussian processes for performance modeling



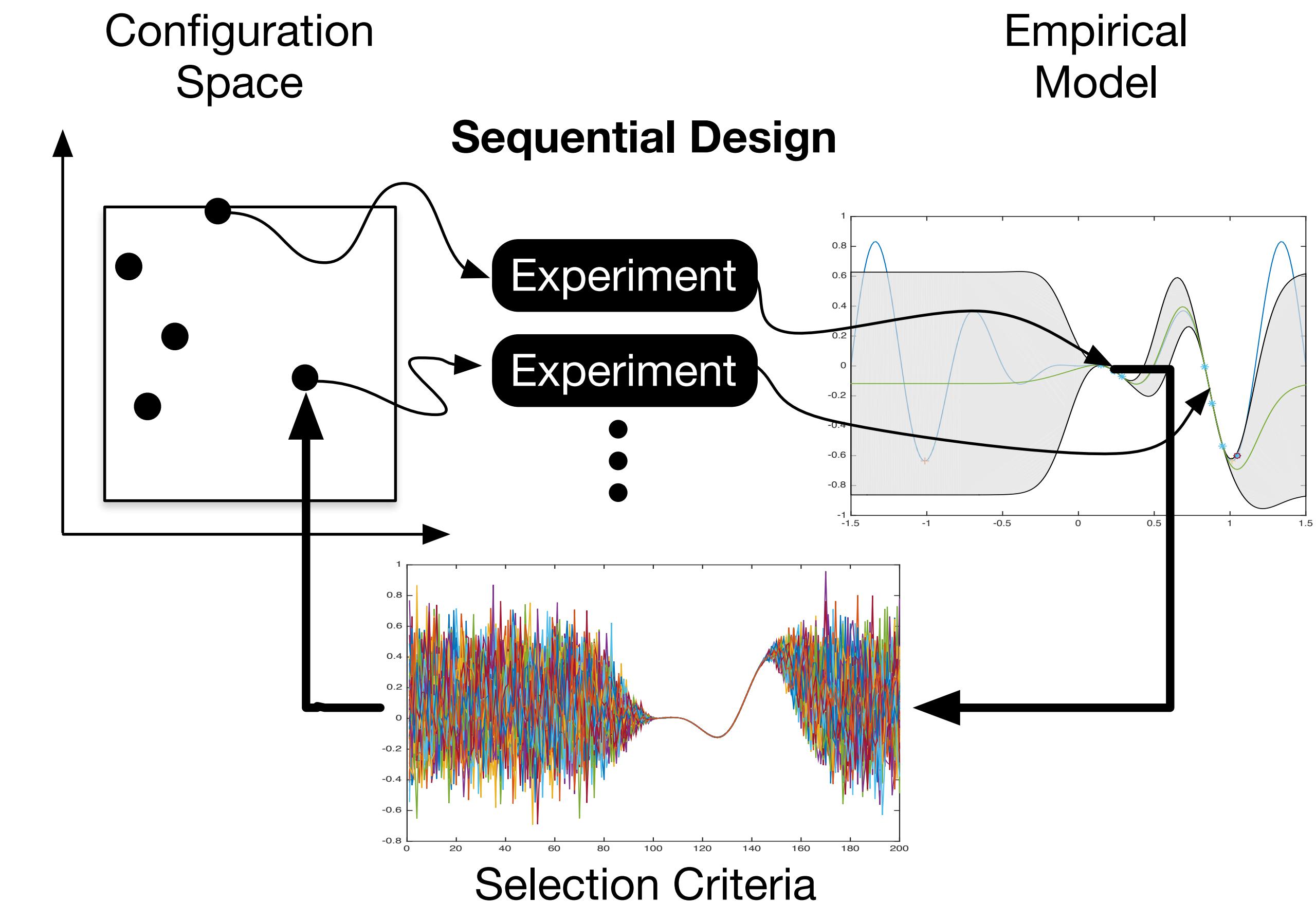
# Gaussian Processes enables reasoning about performance

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

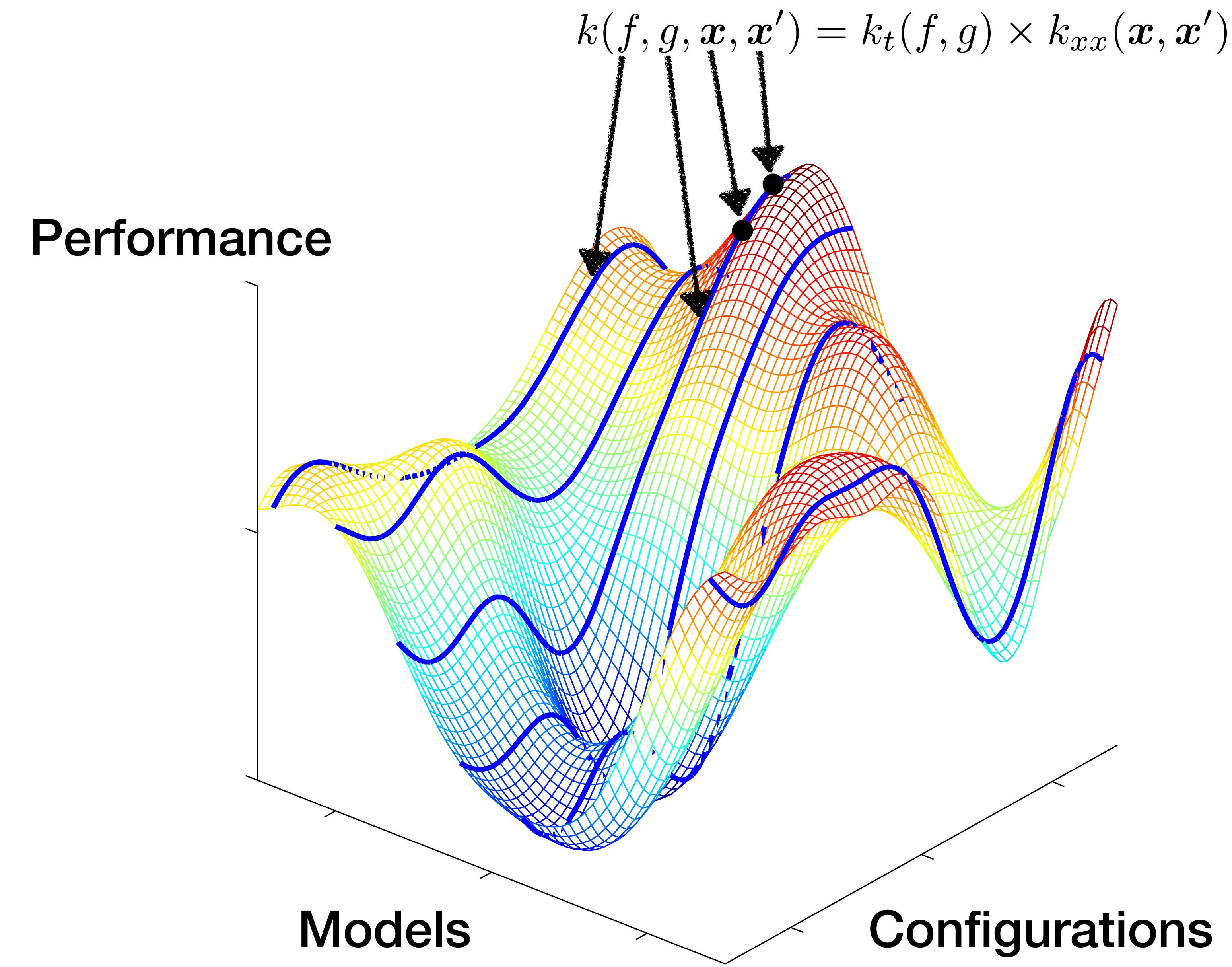
Step 4: Repeat



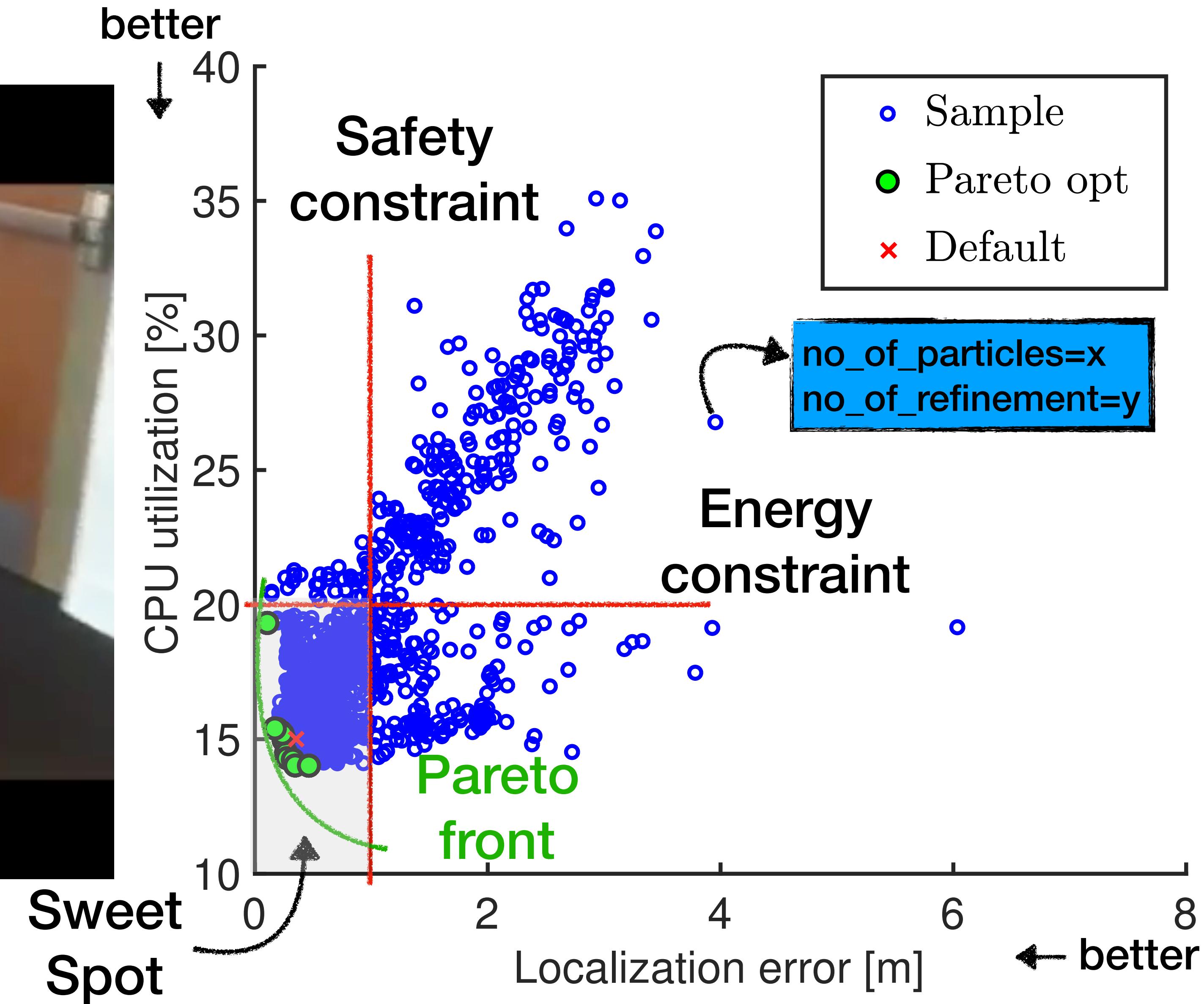
# The intuition behind our transfer learning approach

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

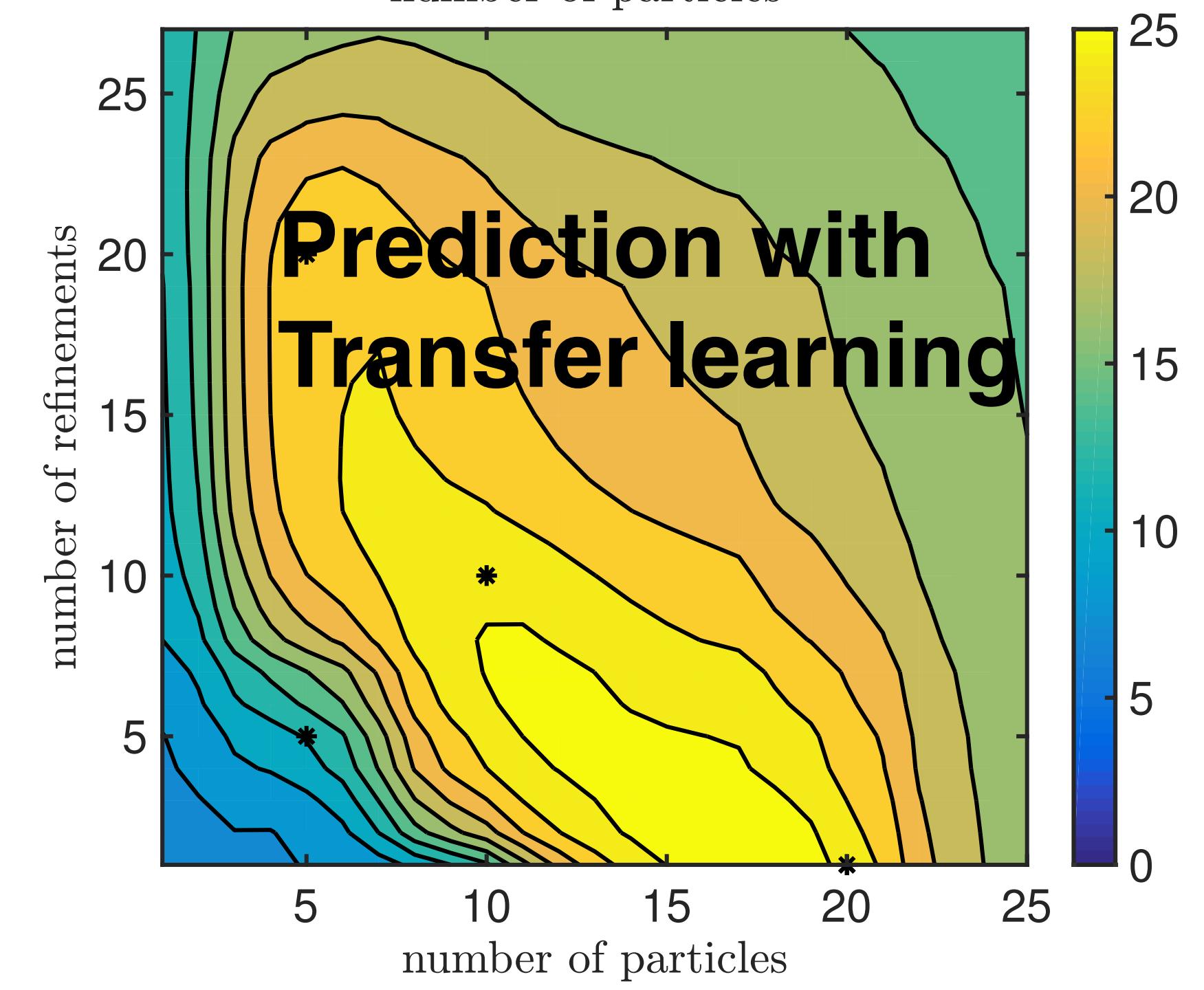
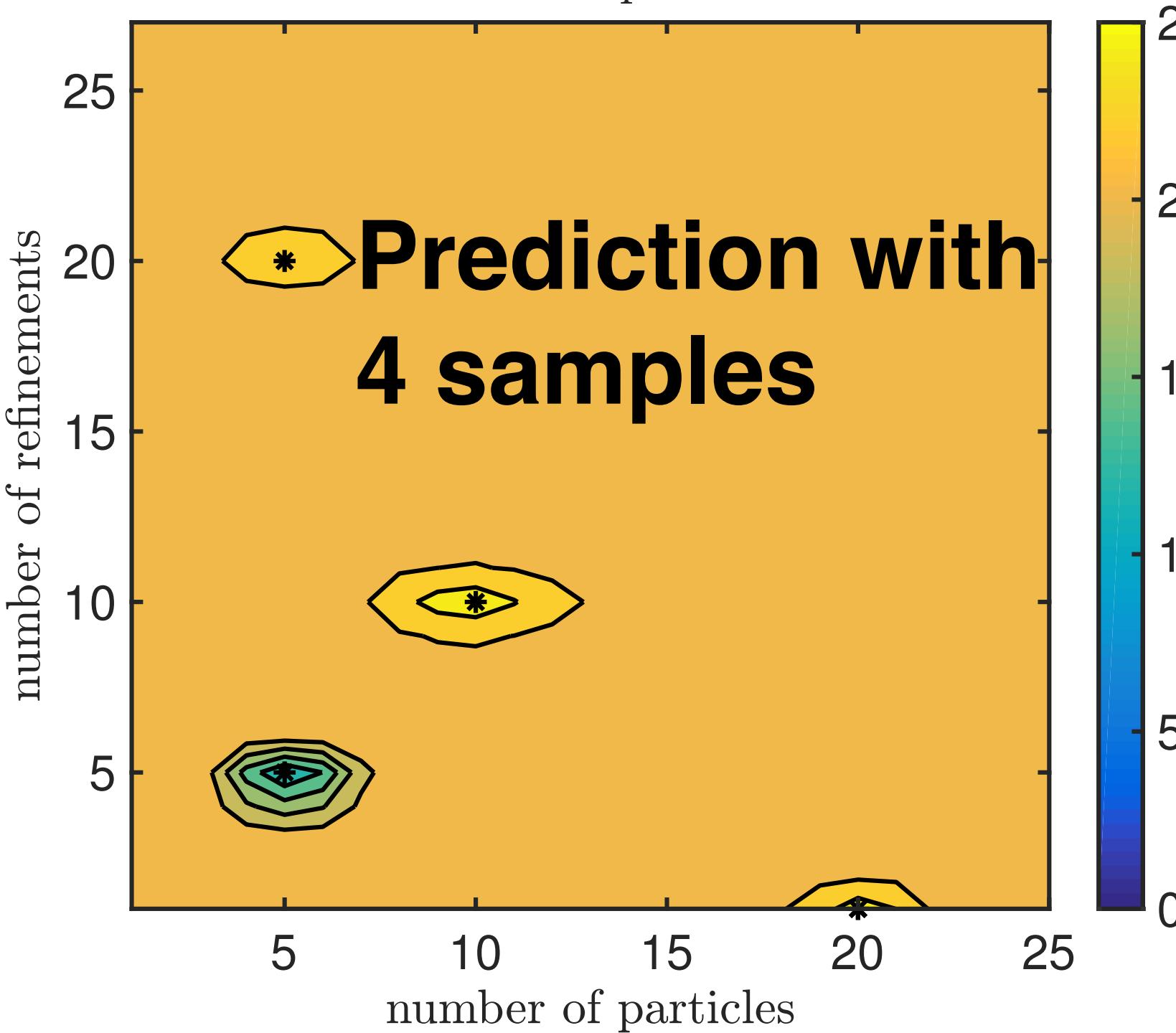
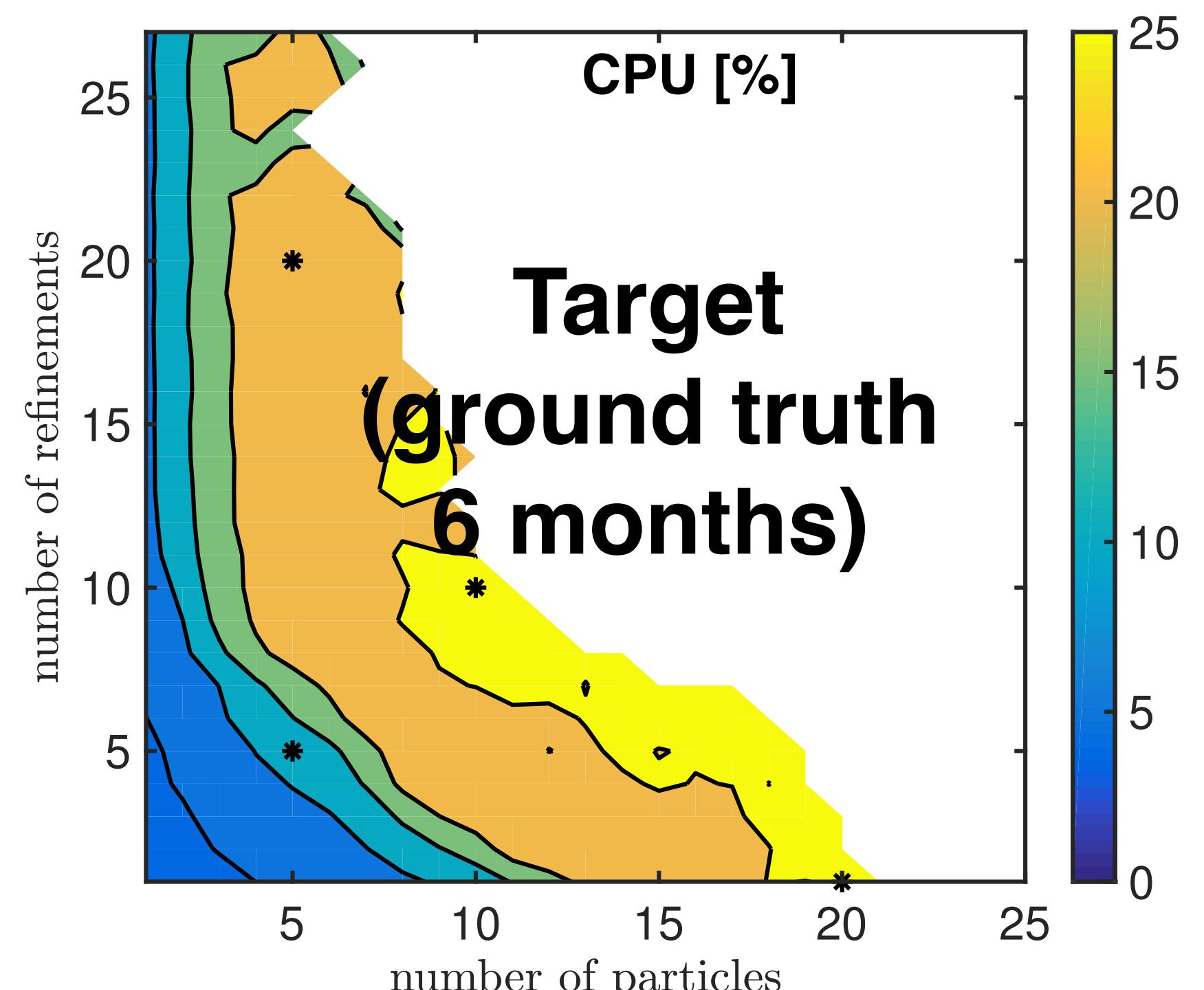
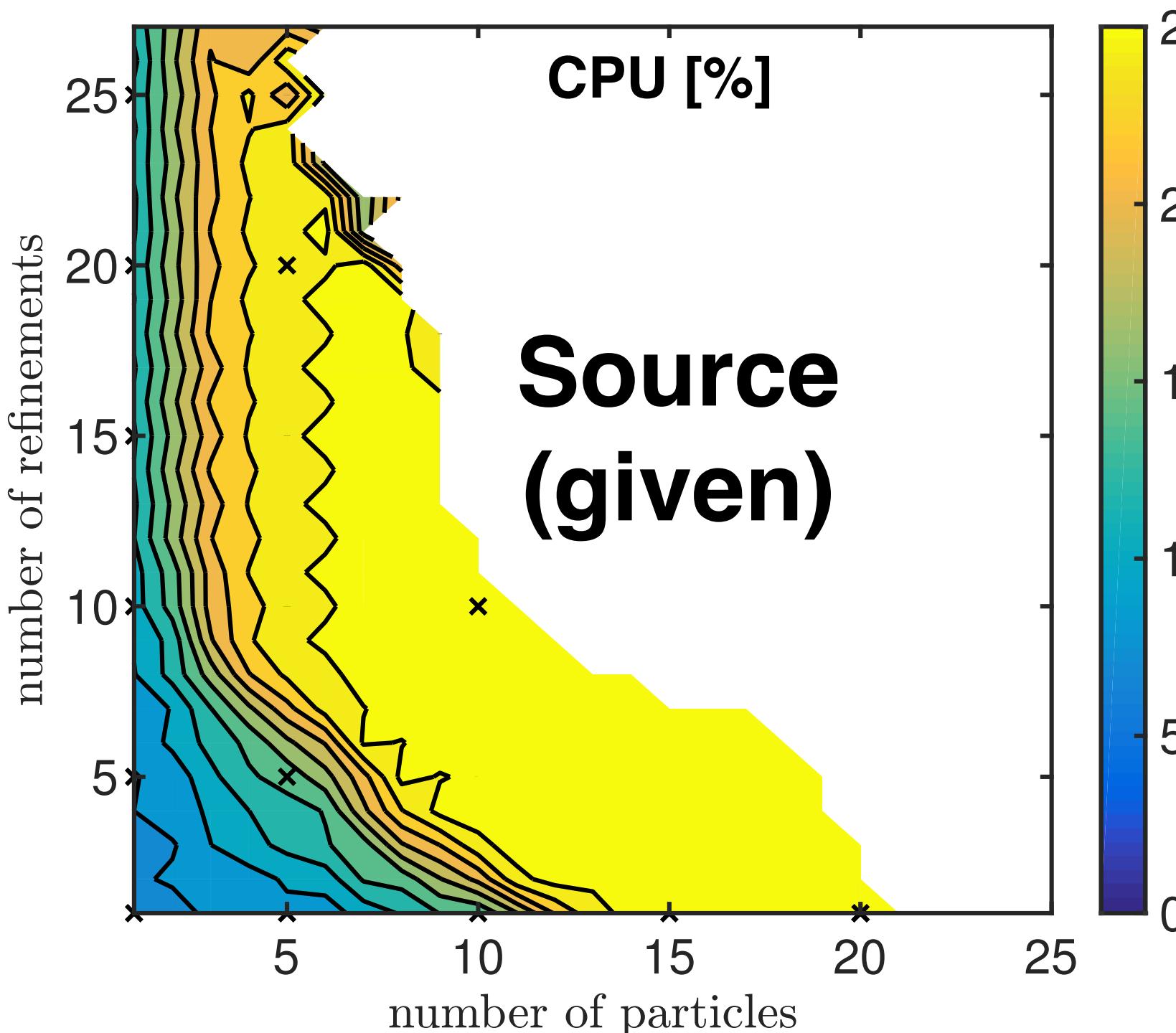
**Example:** Learning the chess game make learning the Go game a lot easier!



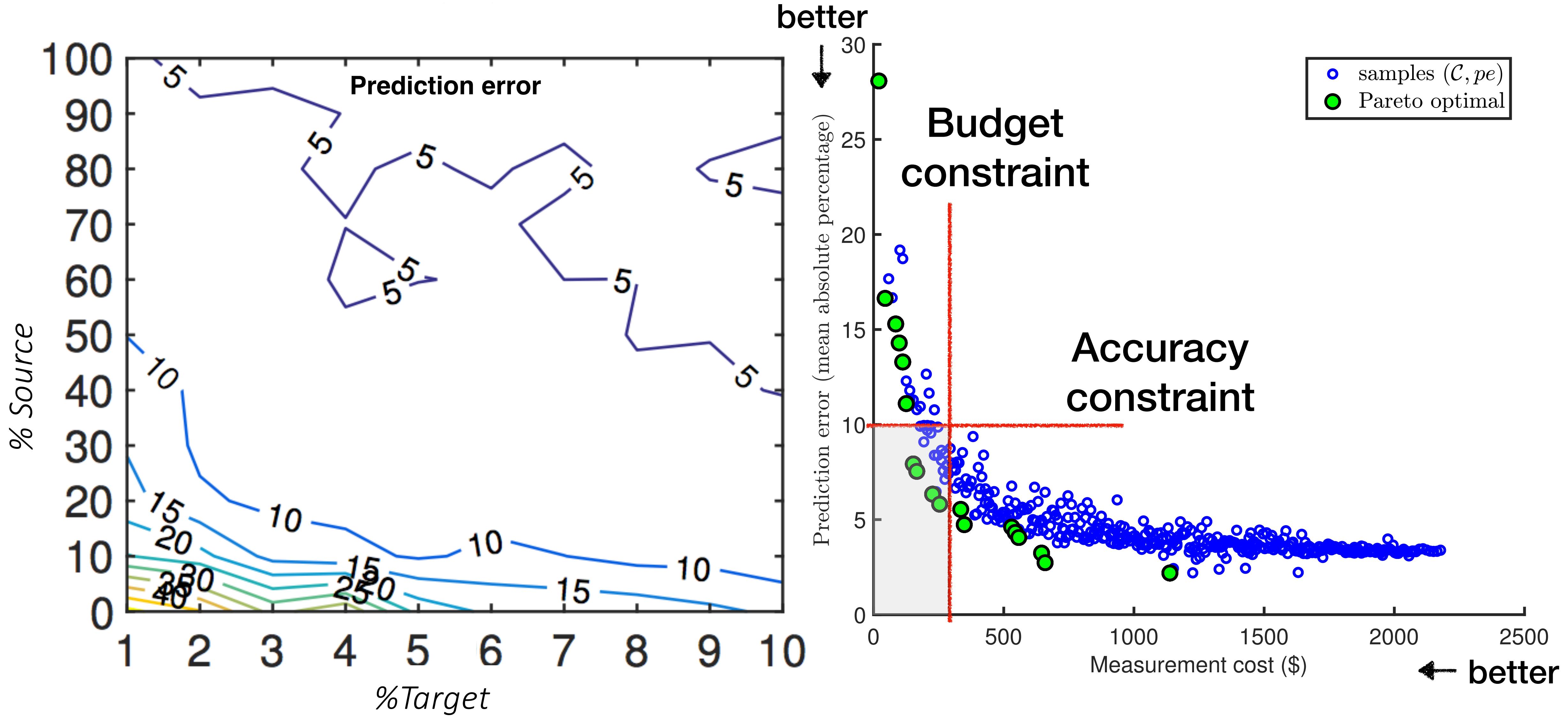
# CoBot experiment: DARPA BRASS



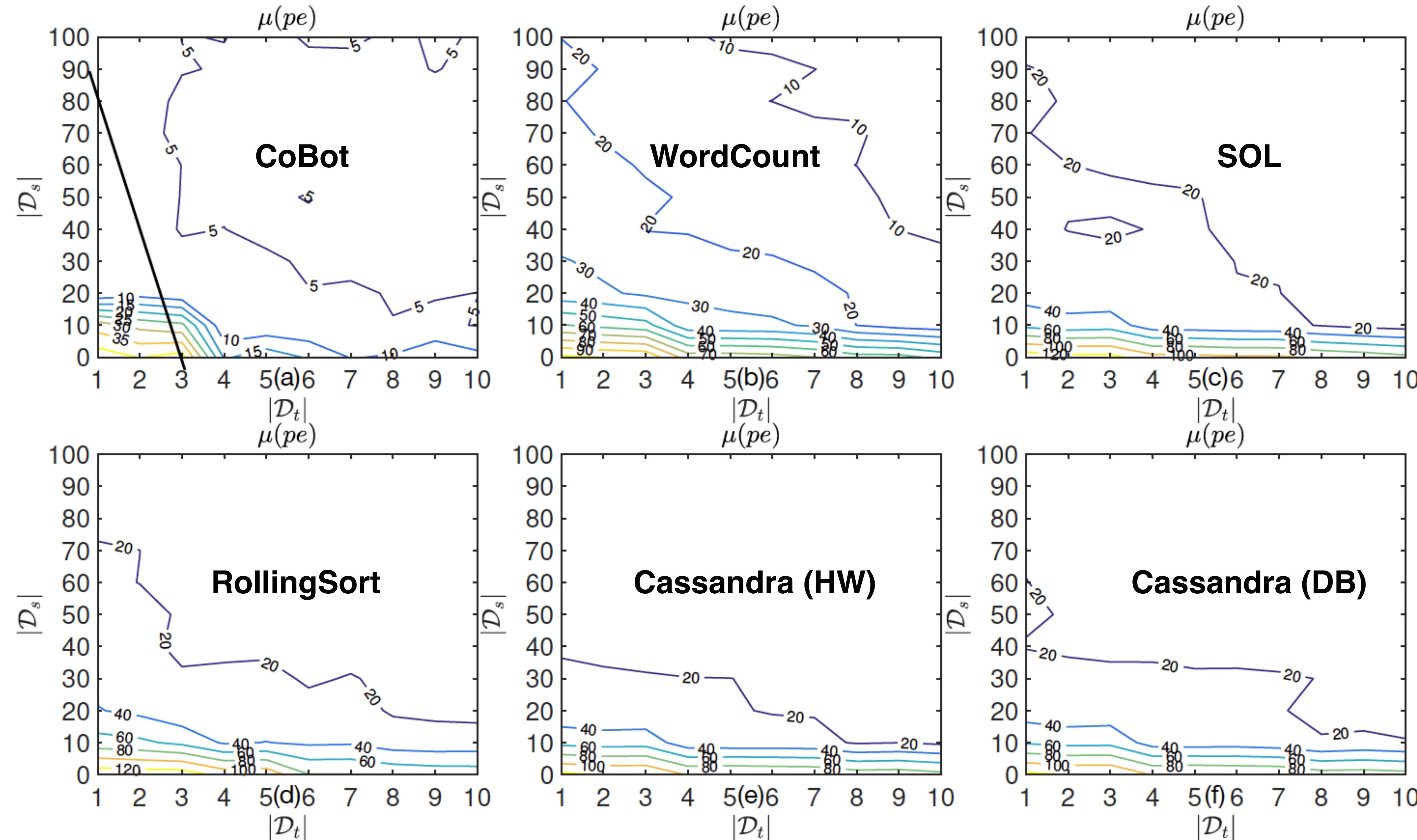
# CoBot experiment



# Result: CoBot experiment



# Results: Other configurable systems



# 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](mailto:{pjamshid,mvelezce,kaestner}@cs.cmu.edu)

Norbert Siegmund  
Bauhaus-University Weimar, Germany  
[norbert.siegmund@uni-weimar.de](mailto:norbert.siegmund@uni-weimar.de)

Prasad Kawthekar  
Stanford University, USA  
[pkawthek@stanford.edu](mailto: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:

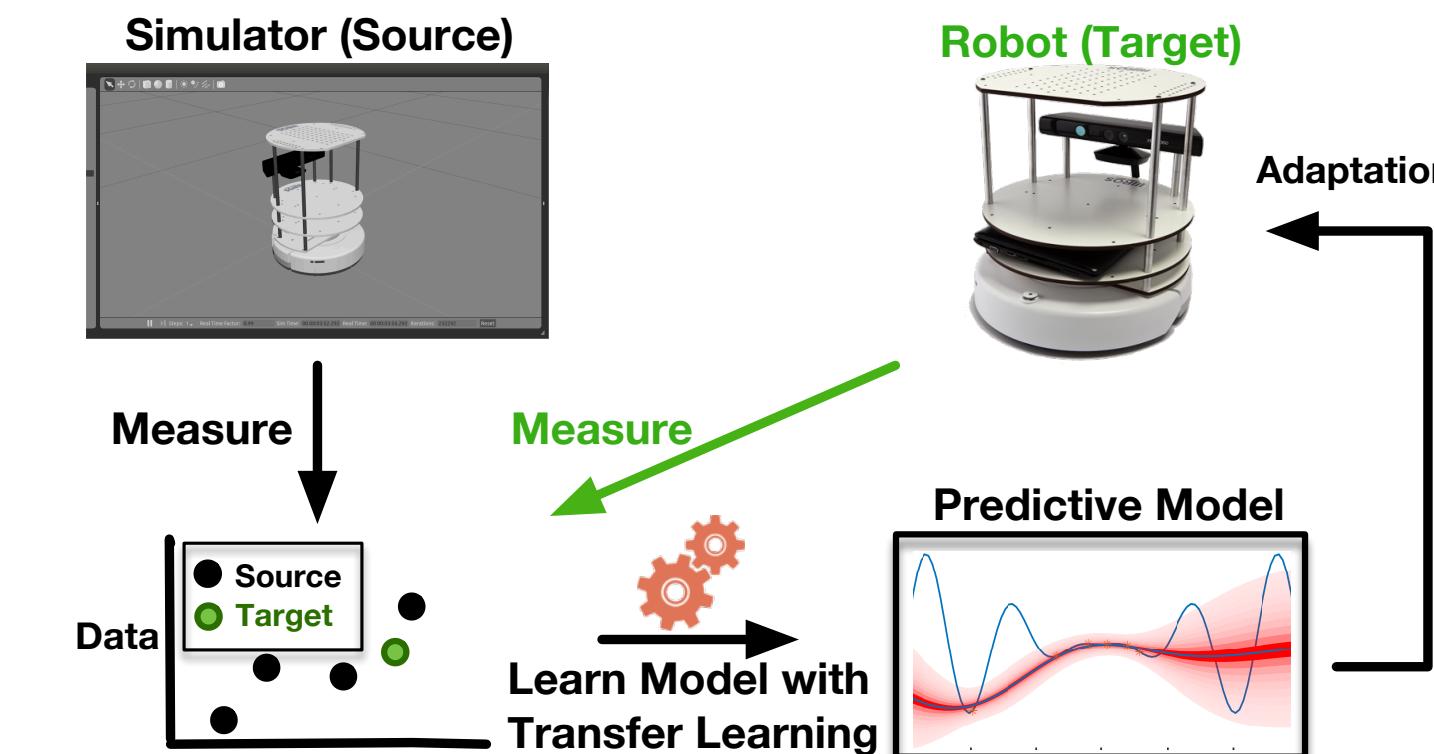
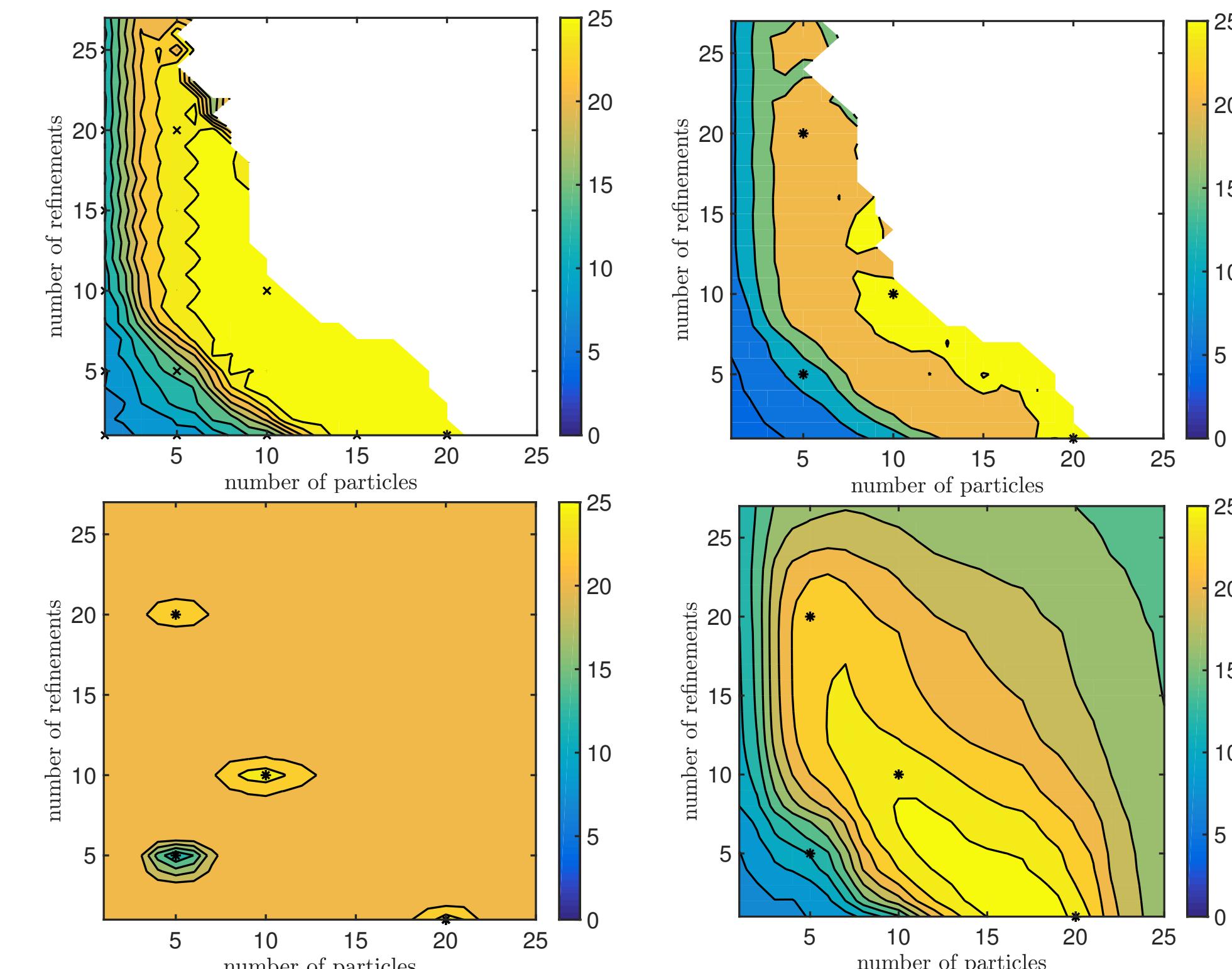


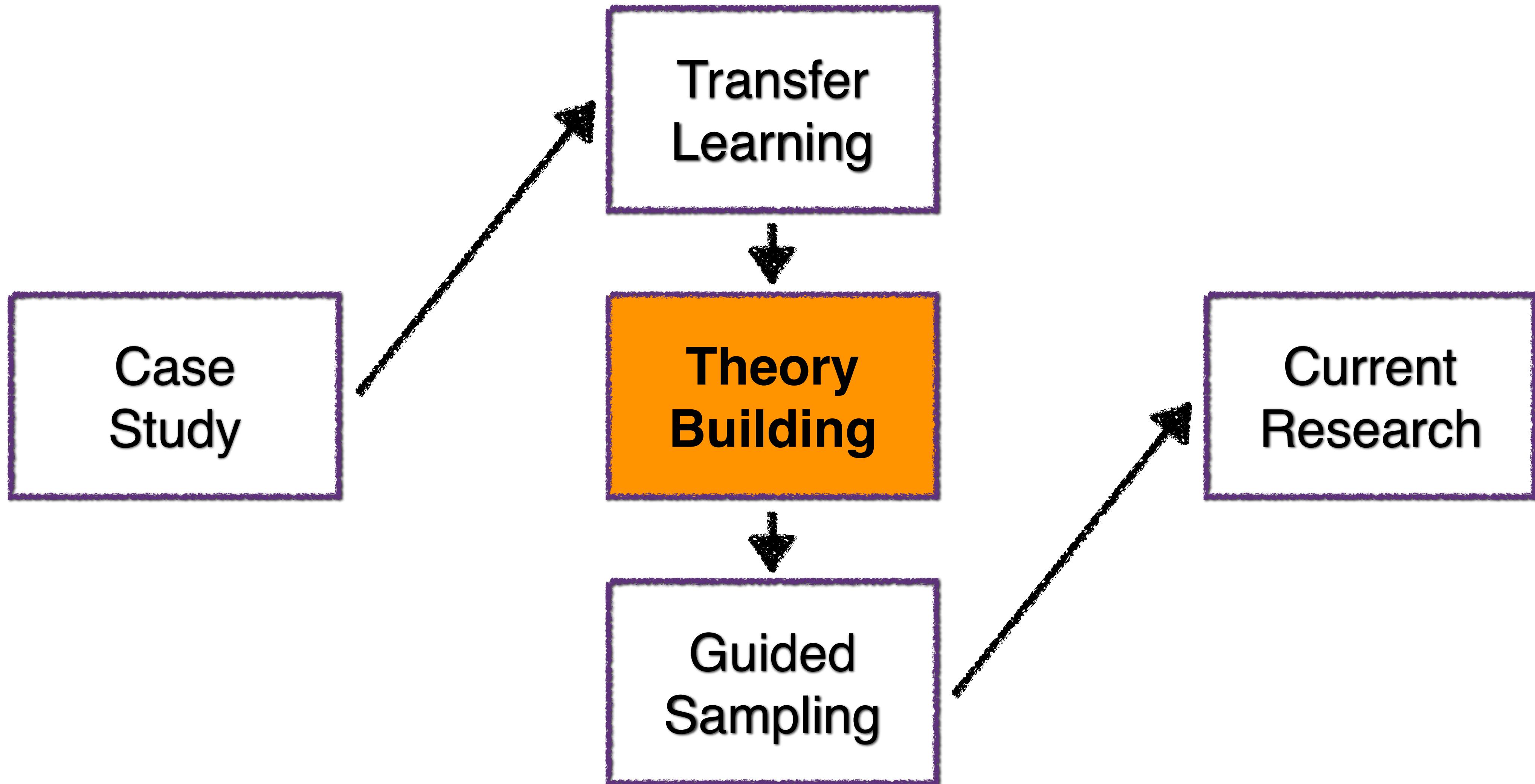
Fig. 1: Transfer learning for performance model learning.

# Summary (transfer learning)

- Model for making tradeoff between qualities
- Scale to large space and environmental changes
- Transfer learning can help
  - Increase prediction accuracy
  - Increase model reliability
  - Decrease model building cost



# Outline



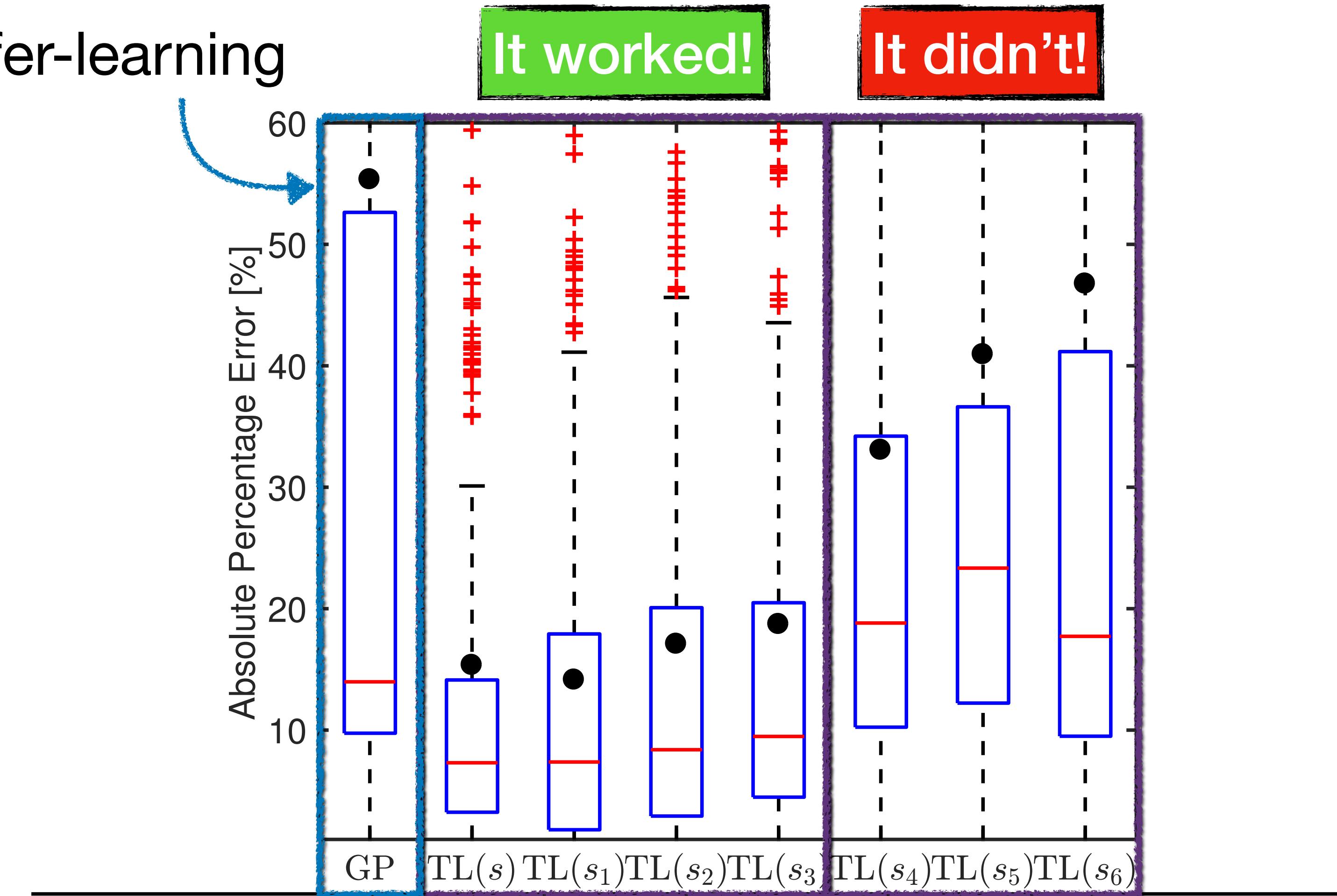
# Looking further: When transfer learning goes wrong

Non-transfer-learning

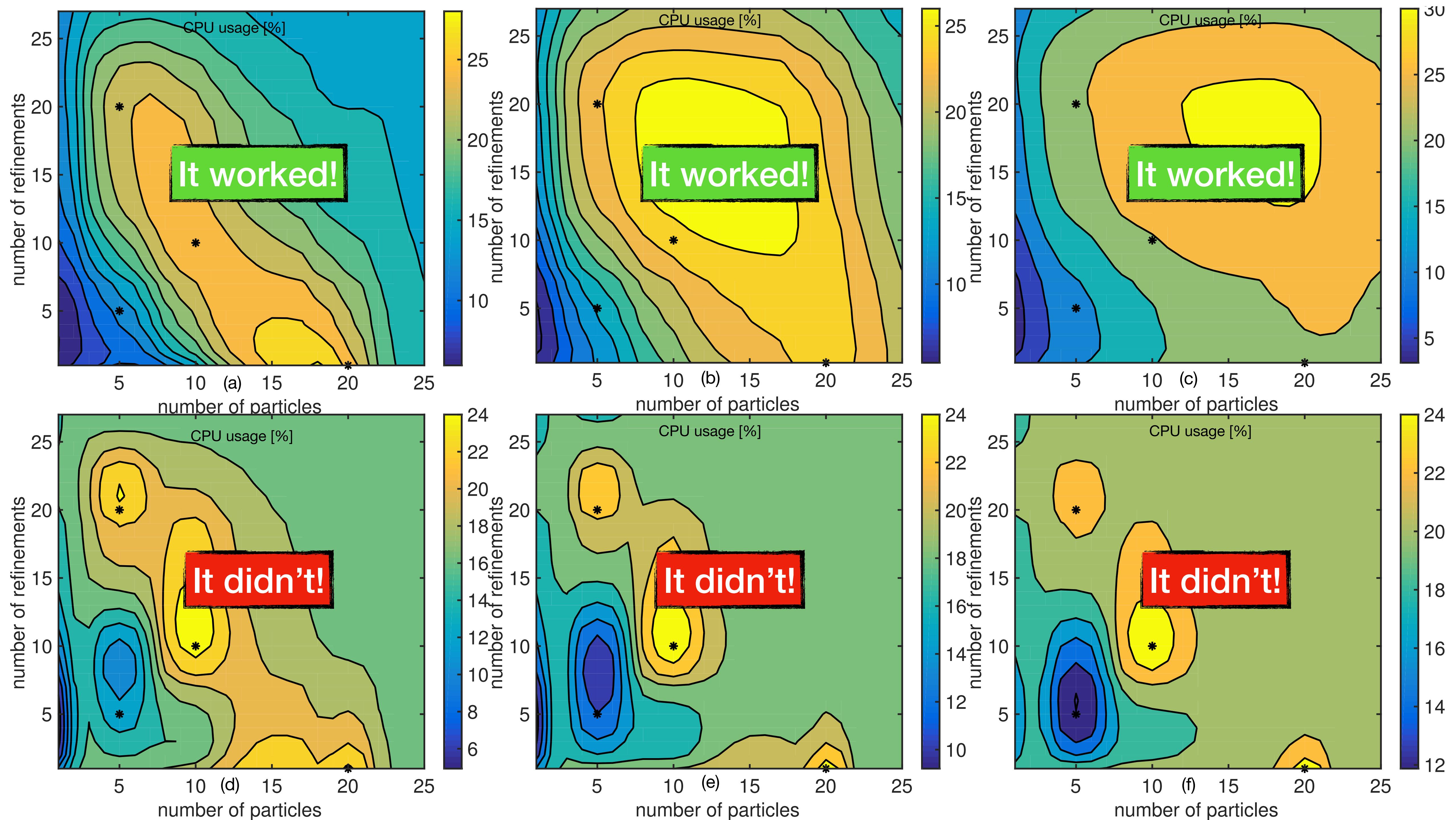
It worked!

It didn't!

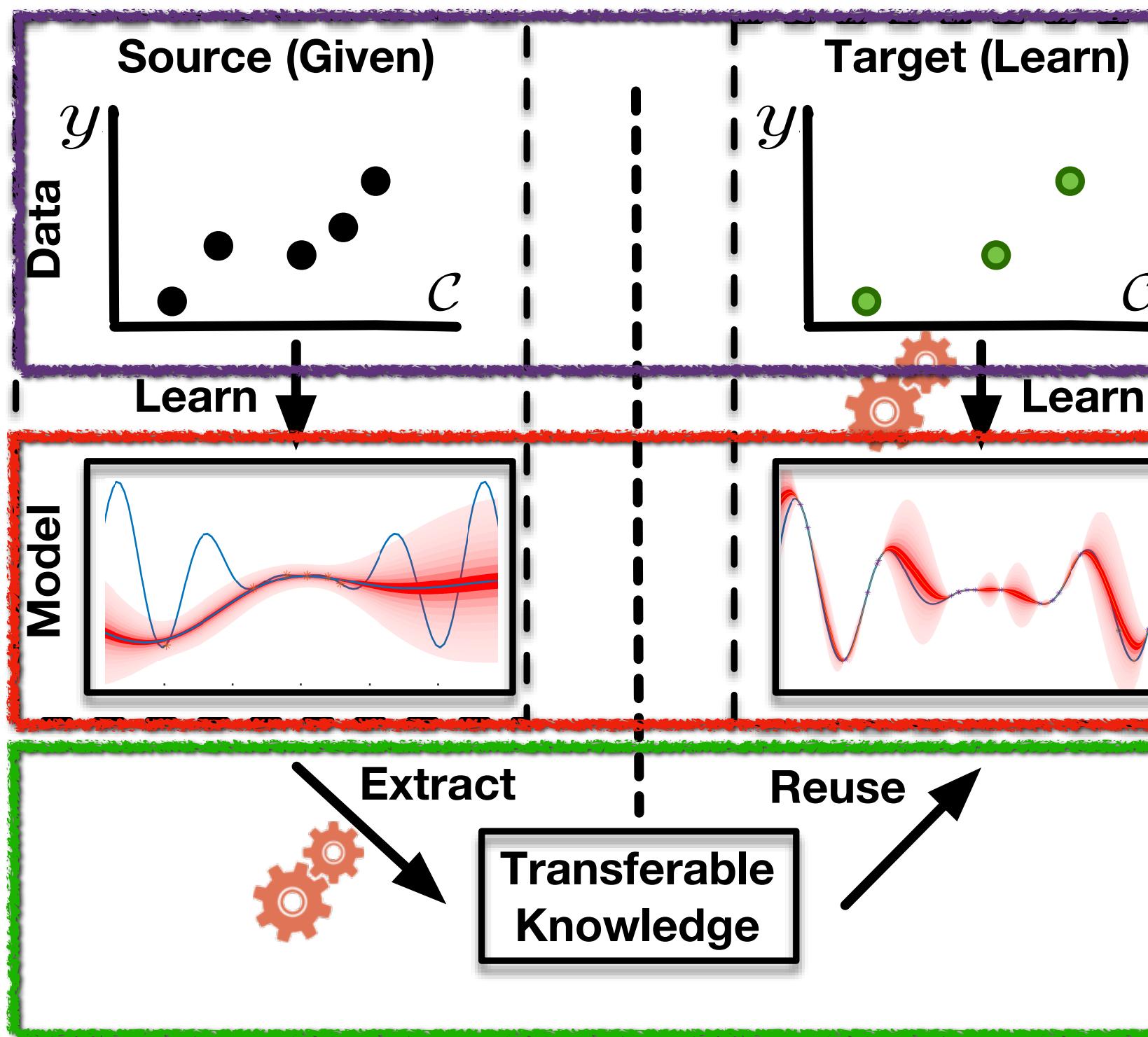
**Insight:** Predictions become more accurate when the source is **more related** to the target.



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



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



**Q1: How source and target are “related”?**

**Q2: What characteristics are preserved?**

**Q3: What are the actionable insights?**

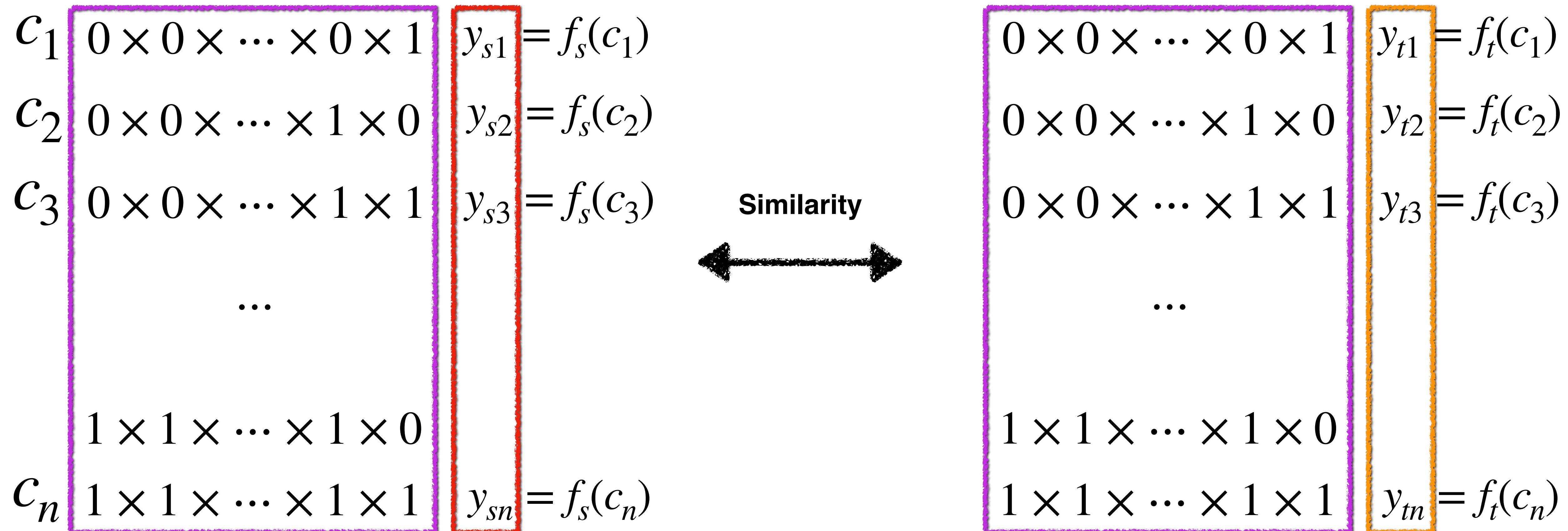
We hypothesized that we can exploit similarities across environments to learn “cheaper” performance models

# Source Environment (Execution time of Program X)

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

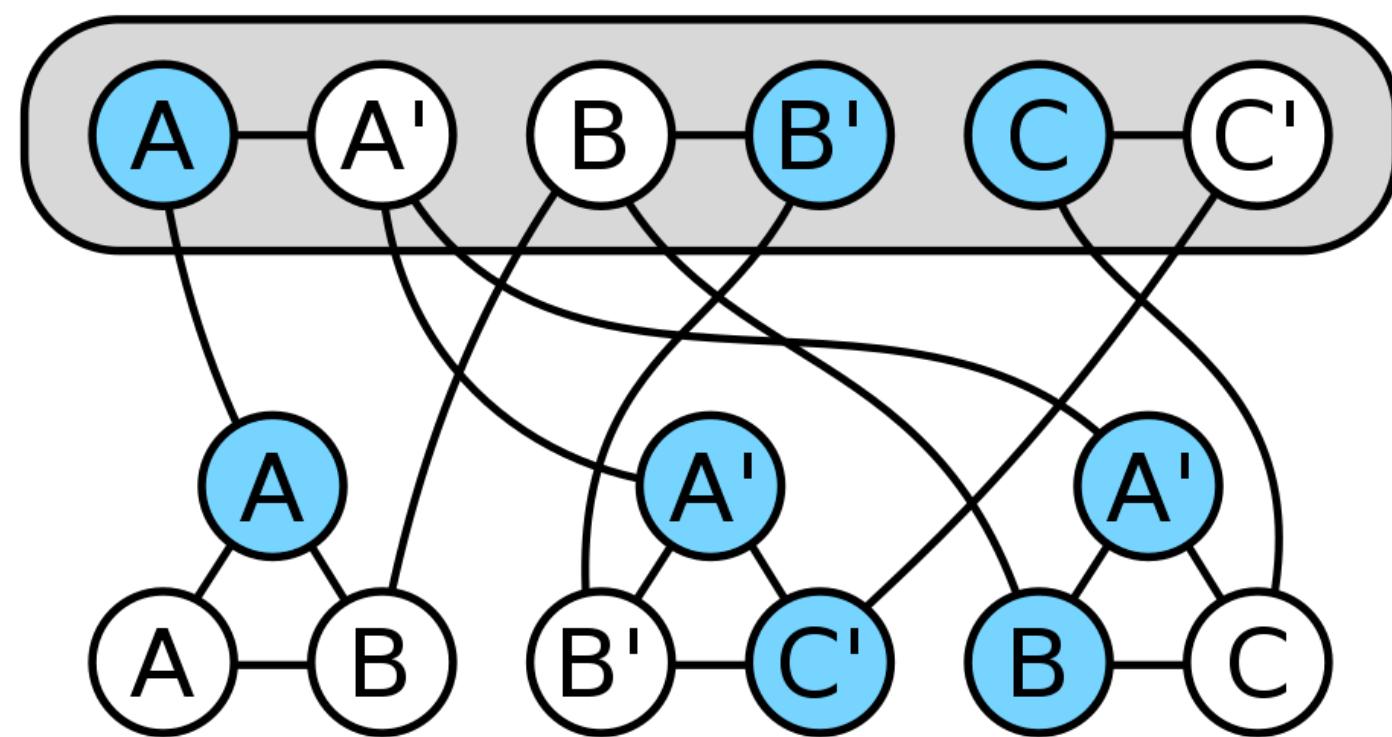
# Target Environment (Execution time of Program Y)

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



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

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



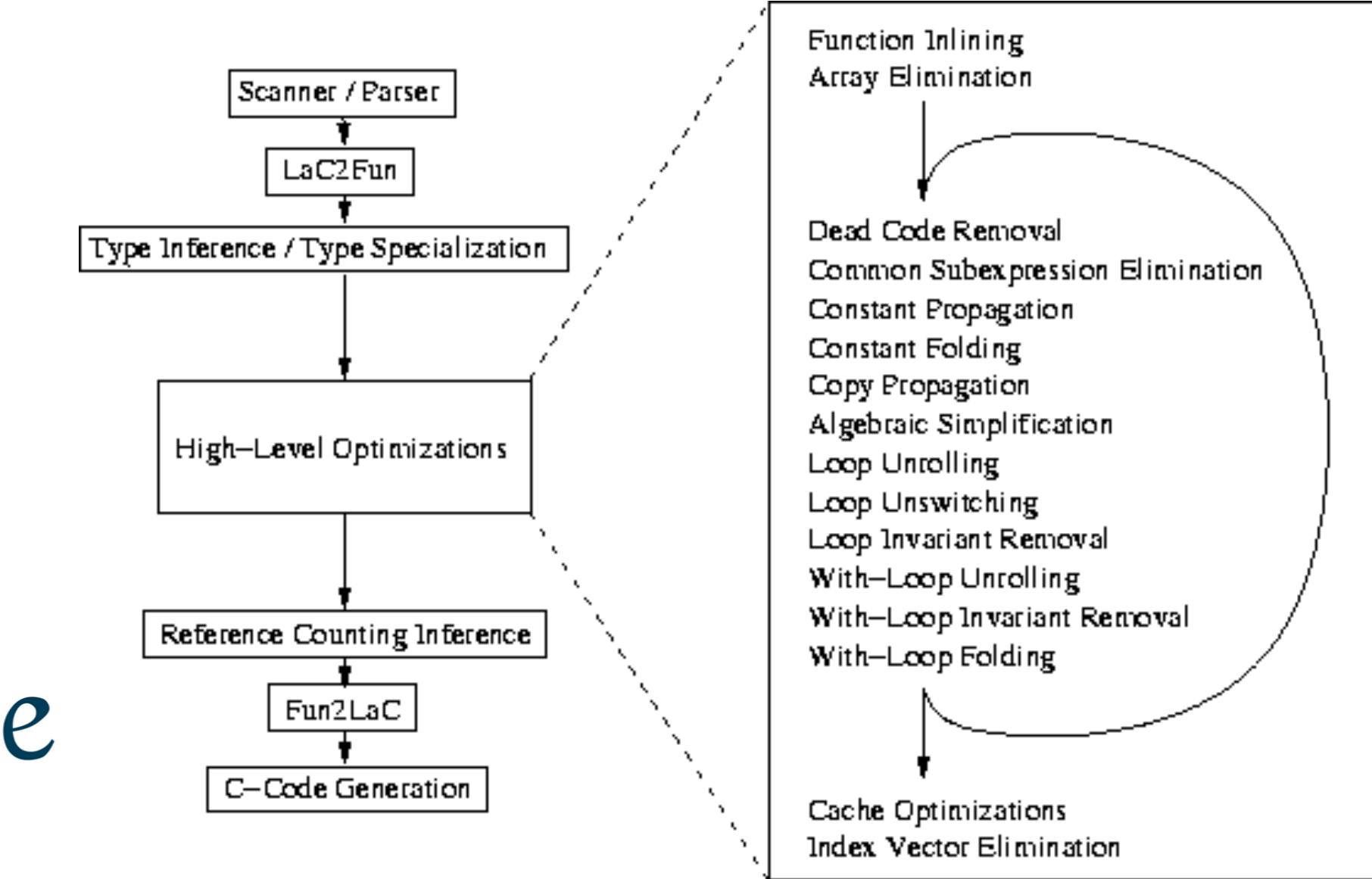
**SPEAR (SAT Solver)**  
**Analysis time**  
14 options



**X264 (video encoder)**  
**Encoding time**  
16 options



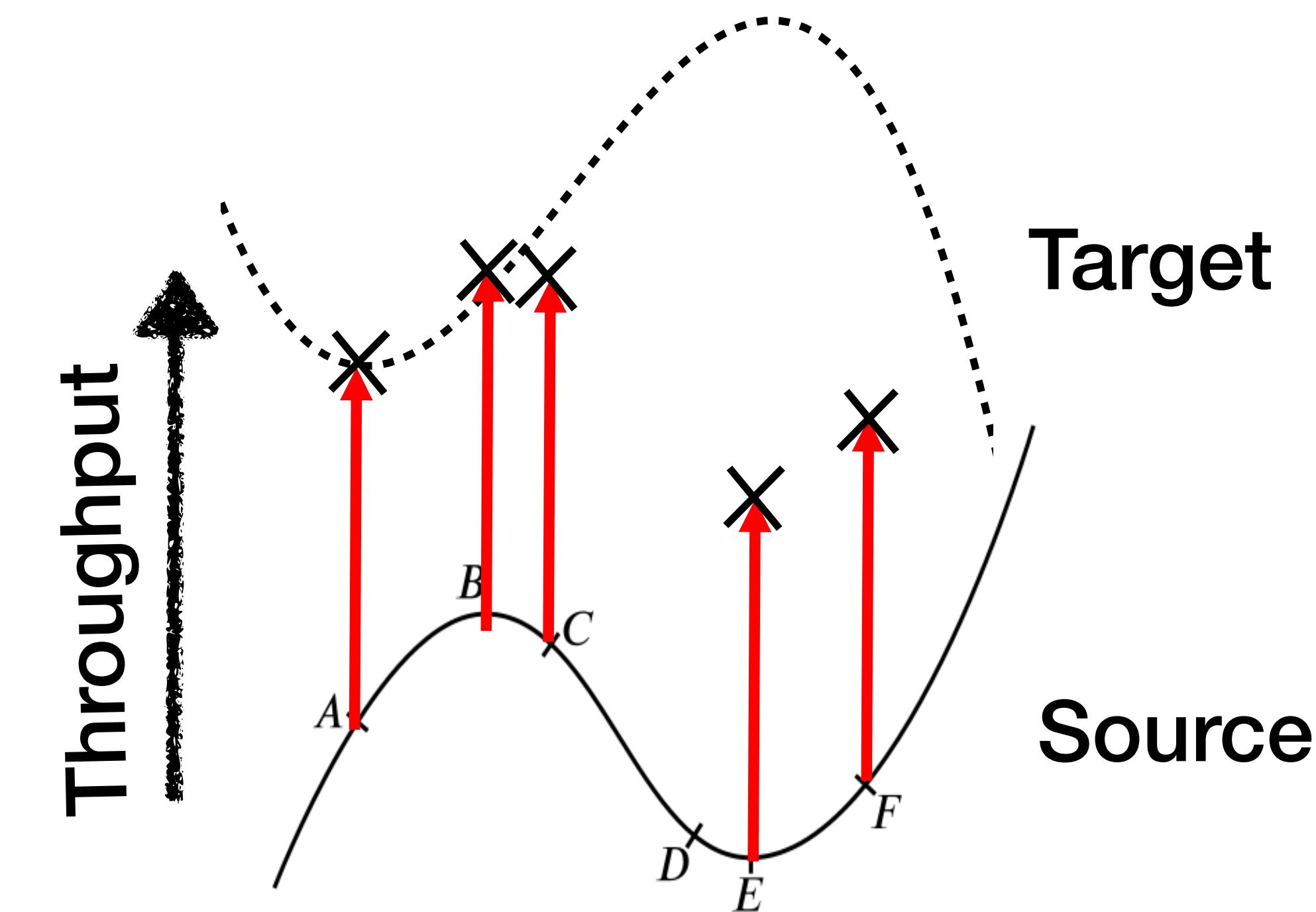
**SQLite (DB engine)**  
**Query time**  
14 options



**SaC (Compiler)**  
**Execution time**  
50 options

# Observation 1: 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

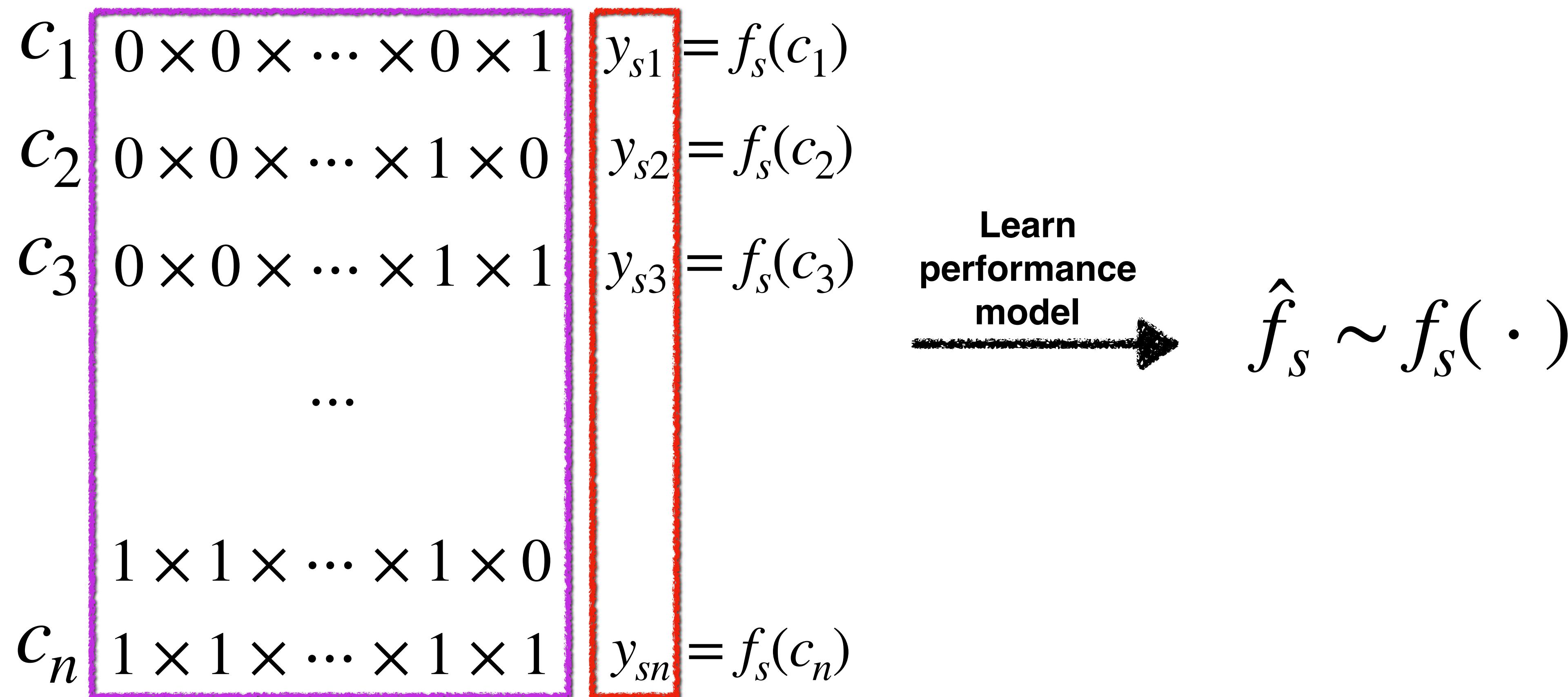
Soft	Environmental change	Severity	Dim	t-test	
x264	Version	Large	16	12	10
	Hardware/workload/ver	V Large		8	9
SQLite	write-seq 📁⌚ write-batch	V Large	14	3	4
	read-rand 📁⌚ read-seq	Medium		1	1
SaC	Workload	V Large	50	16	10

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)  
 $O_1 \times O_2 \times \dots \times O_{19} \times O_{20}$



**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

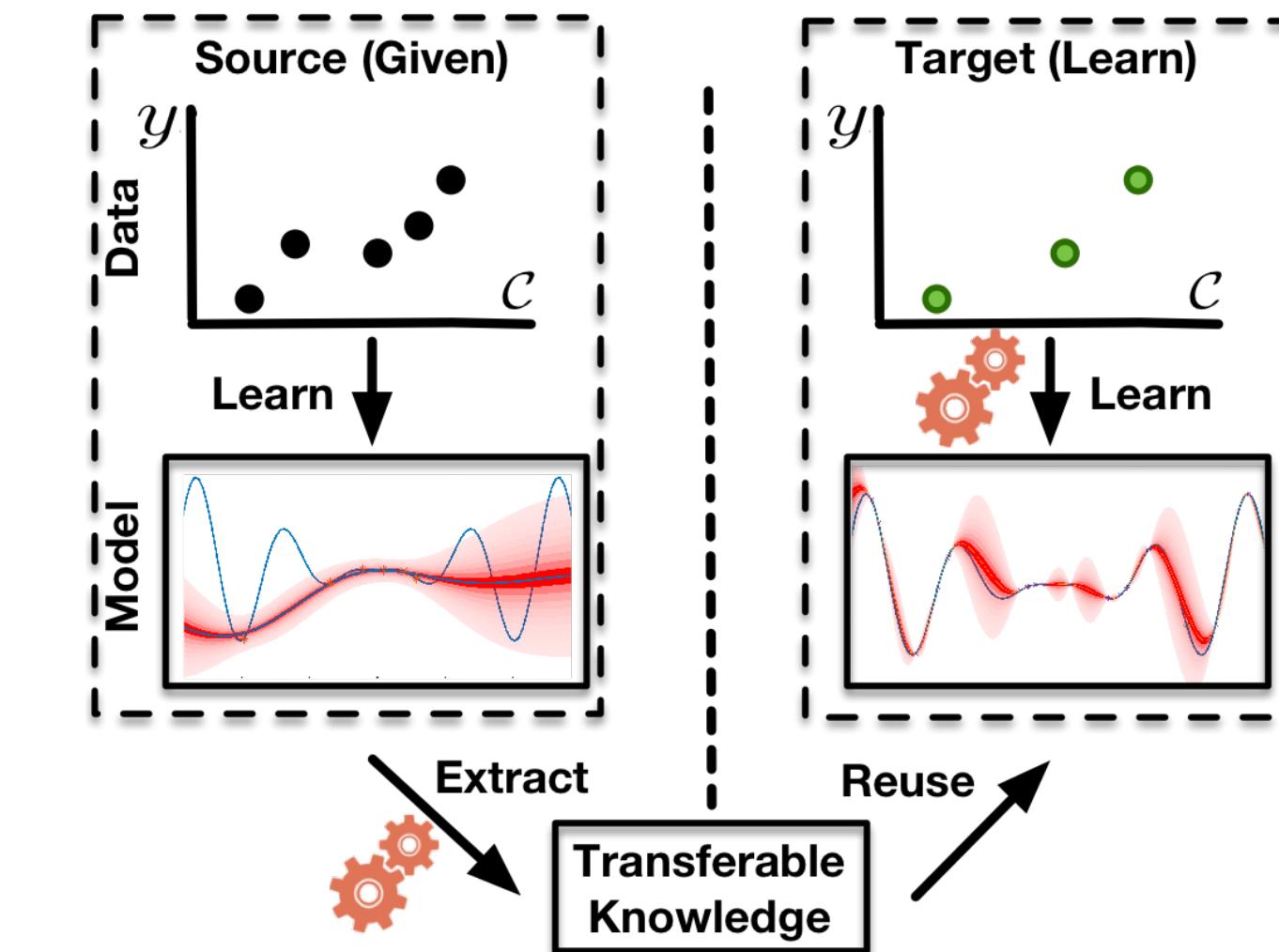
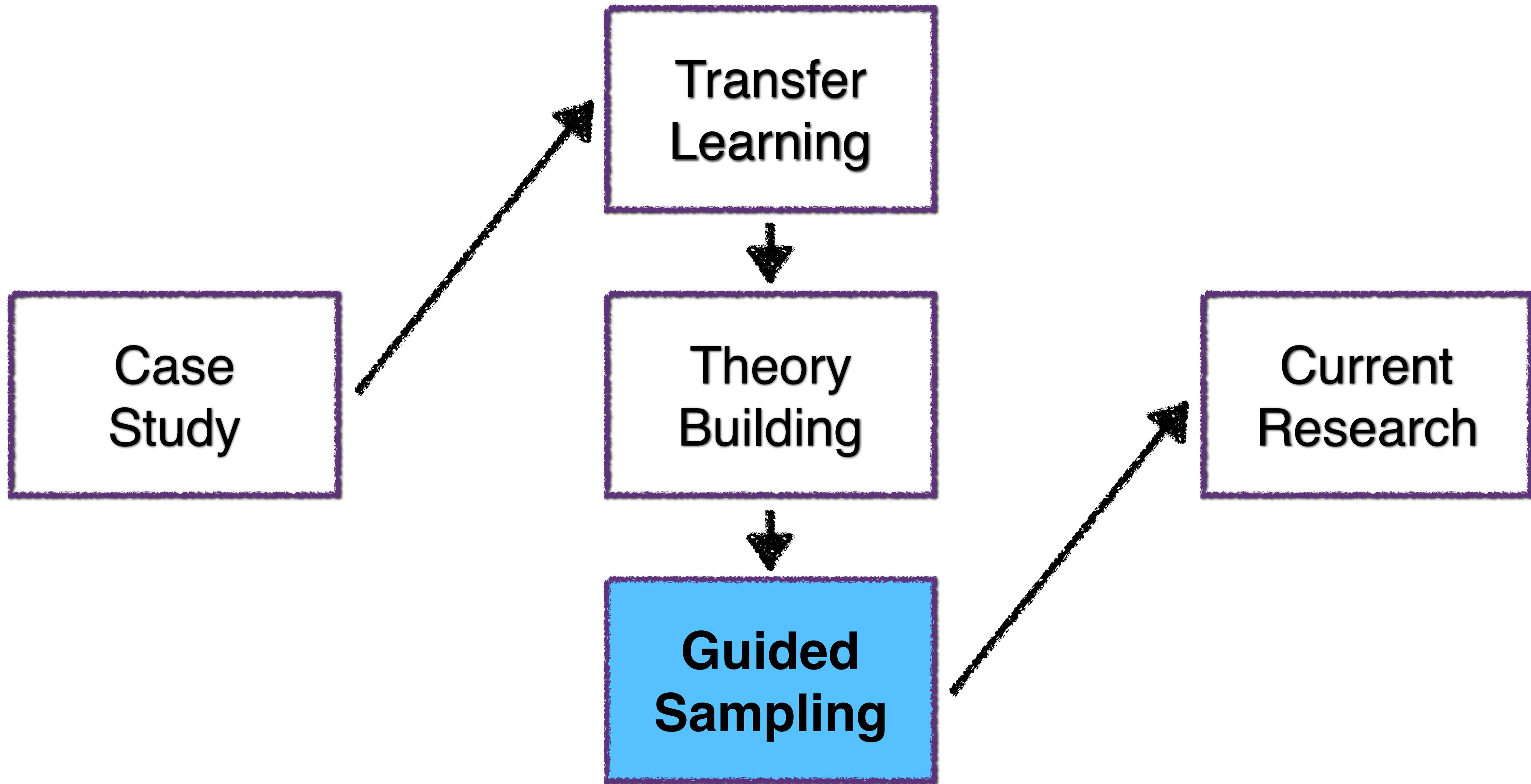


Fig. 1: Transfer learning is a form of machine learning that takes advantage of transferable knowledge from source to learn an accurate, reliable, and less costly model for the target environment.

# Outline



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

**How to select the most informative configurations?**



The similarity across environments 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

$O_1 \times O_2 \times \dots \times O_{19} \times O_{20}$	
$c_1$	$0 \times 0 \times \dots \times 0 \times 1$
$c_2$	$0 \times 0 \times \dots \times 1 \times 0$
$c_3$	$0 \times 0 \times \dots \times 1 \times 1$
	$\dots$
	$1 \times 1 \times \dots \times 1 \times 0$
$c_n$	$1 \times 1 \times \dots \times 1 \times 1$

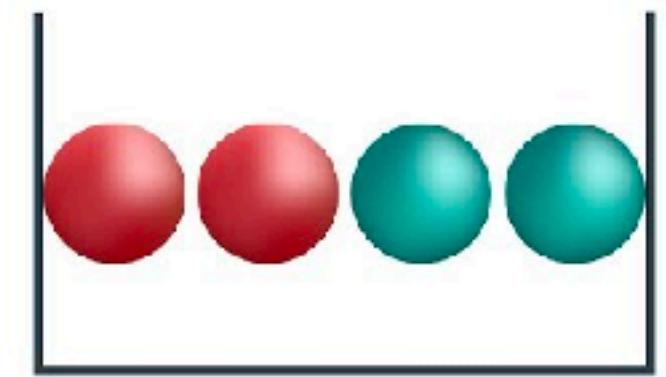
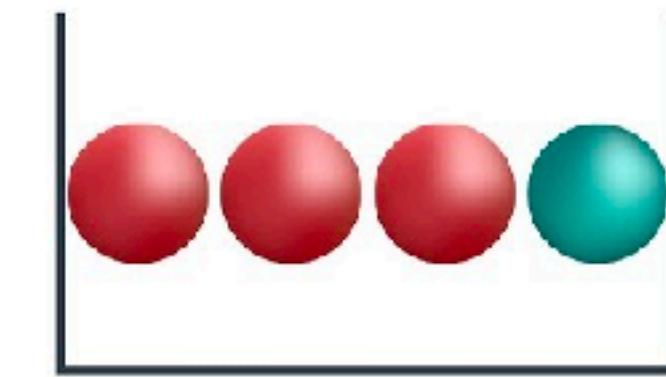
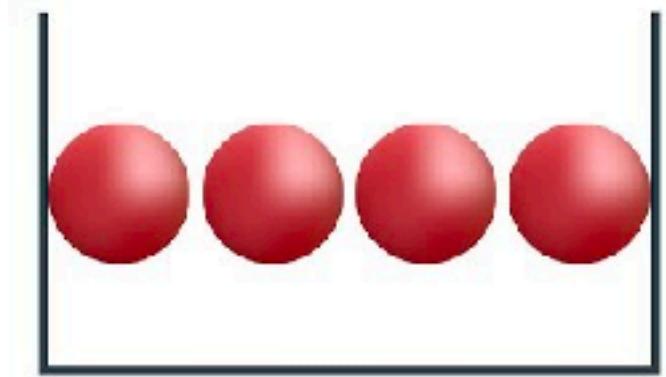
- We therefore end up blindly explore the configuration space
- That is essentially the key reason why “most” work in this area consider random sampling.

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

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

	$O_1 \times O_2 \times O_3 \times O_4 \times O_5 \times O_6 \times O_7 \times O_8 \times O_9 \times O_{10}$	
$c_1$	$1 \times 0 \times 1 \times 0 \times 0 \times 0 \times 1 \times 0 \times 0 \times 0$	$\hat{f}_s(c_1) = 14.9$
$c_2$	$1 \times 0 \times 1 \times 0 \times 0 \times 0 \times 1 \times 0 \times 0 \times 1$	$\hat{f}_s(c_2) = 14.9$
$c_3$	$1 \times 0 \times 1 \times 0 \times 0 \times 0 \times 1 \times 0 \times 1 \times 0$	$\hat{f}_s(c_3) = 14.9$
...		
$c_{128}$	$1 \times 1 \times 1$	$\hat{f}_s(c_{128}) = 14.9$

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



In higher dimensional spaces, the blind samples even become less informative/effective



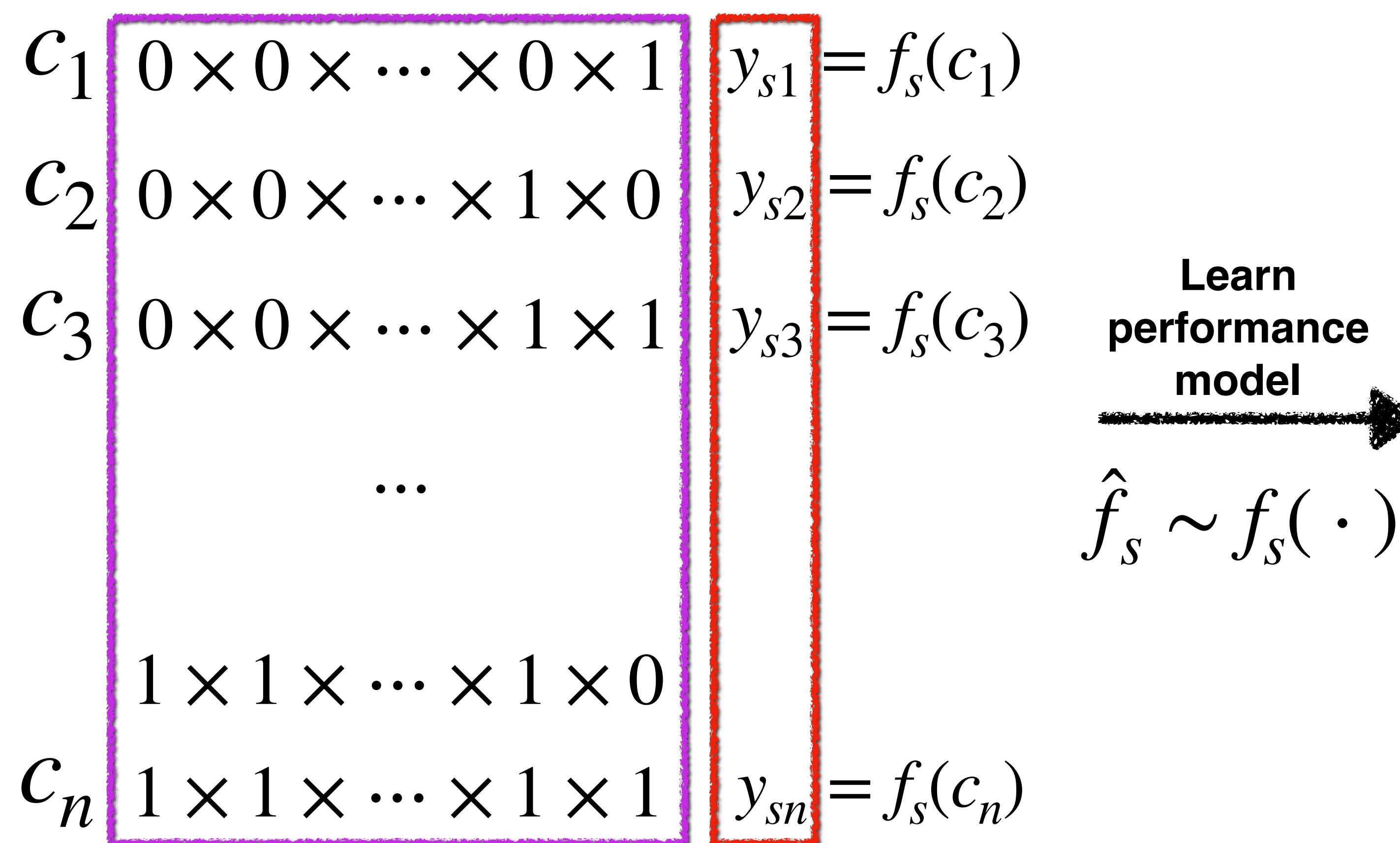
# **Learning to Sample (L2S)**

# Extracting the knowledge about influential options and interactions: Step-wise linear regression

Source

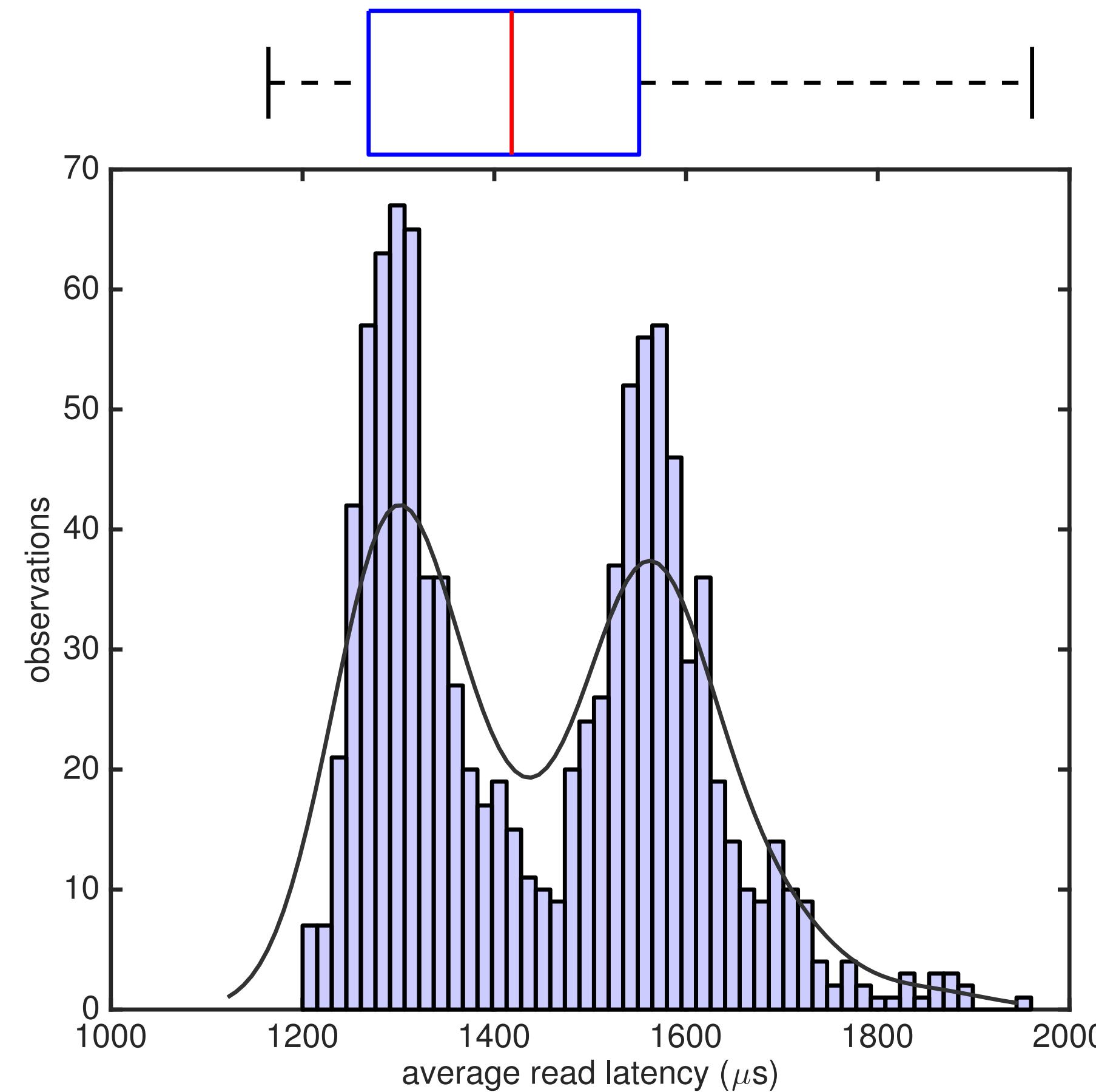
(Execution time of Program X)

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



1. Fit an **initial model**
2. **Forward selection:** Add terms iteratively
3. **Backward elimination:** Removes terms iteratively
4. **Terminate:** When neither (2) or (3) improve the model

# Build a performance distribution using kernel density estimation using the source data



L2S extracts the knowledge about influential options and interactions via performance models

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

L2S exploits the knowledge it gained from the source to sample the target environment

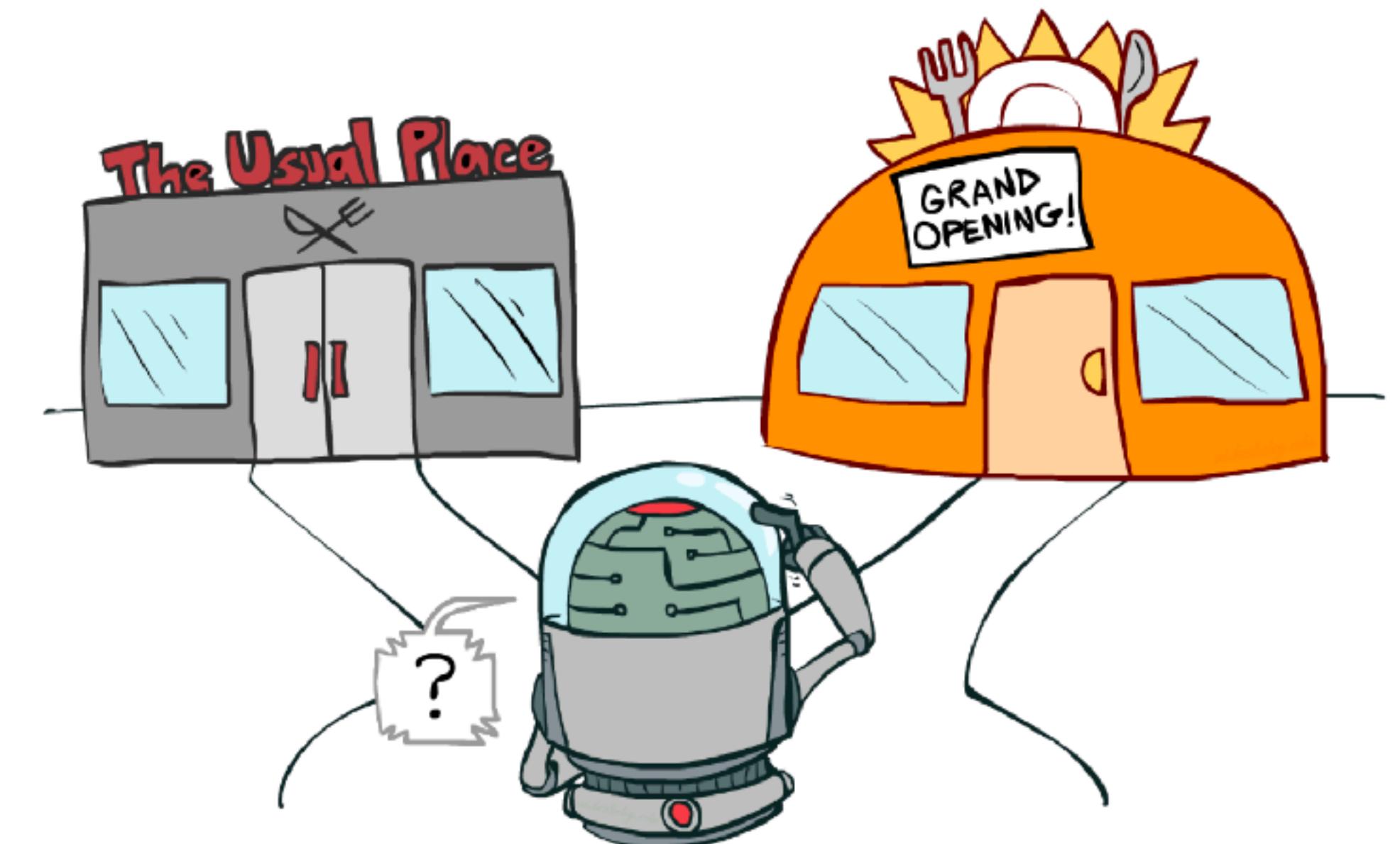
$$\hat{f}_s(\cdot) = 1.2 + 3o_1 + 5o_3 + 0.9o_7 + 0.8o_3o_7 + 4o_1o_3o_7$$

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

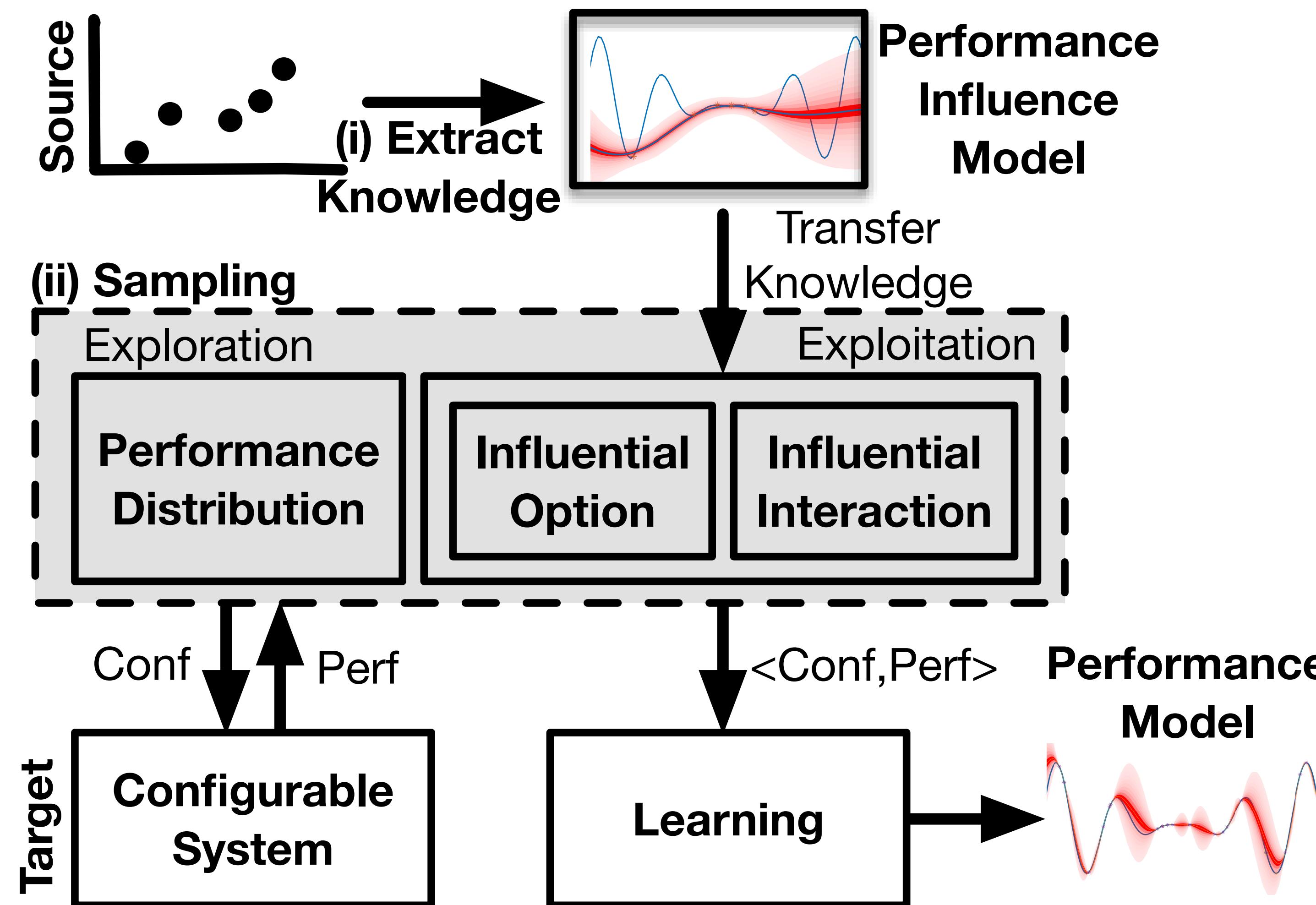
	$O_1 \times O_2 \times O_3 \times O_4 \times O_5 \times O_6 \times O_7 \times O_8 \times O_9 \times O_{10}$	
$c_1$	$0 \times 0 \times 0$	$\hat{f}_t(c_1) = 10.4$
$c_2$	$1 \times 0 \times 0$	$\hat{f}_t(c_2) = 8.1$
$c_3$	$0 \times 0 \times 1 \times 0 \times 0 \times 0 \times 0 \times 0 \times 0 \times 0$	$\hat{f}_t(c_3) = 11.6$
$c_4$	$0 \times 0 \times 0 \times 0 \times 0 \times 0 \times 1 \times 0 \times 0 \times 0$	$\hat{f}_t(c_4) = 12.6$
$c_5$	$0 \times 0 \times 1 \times 0 \times 0 \times 0 \times 1 \times 0 \times 0 \times 0$	$\hat{f}_t(c_5) = 11.7$
$c_6$	$1 \times 0 \times 1 \times 0 \times 0 \times 0 \times 1 \times 0 \times 0 \times 0$	$\hat{f}_t(c_6) = 23.7$
$c_x$	$1 \times 0 \times 1 \times 0 \times 0 \times 0 \times 1 \times 0 \times 0 \times 0$	$\hat{f}_t(c_x) = 9.6$

# Exploration vs Exploitation

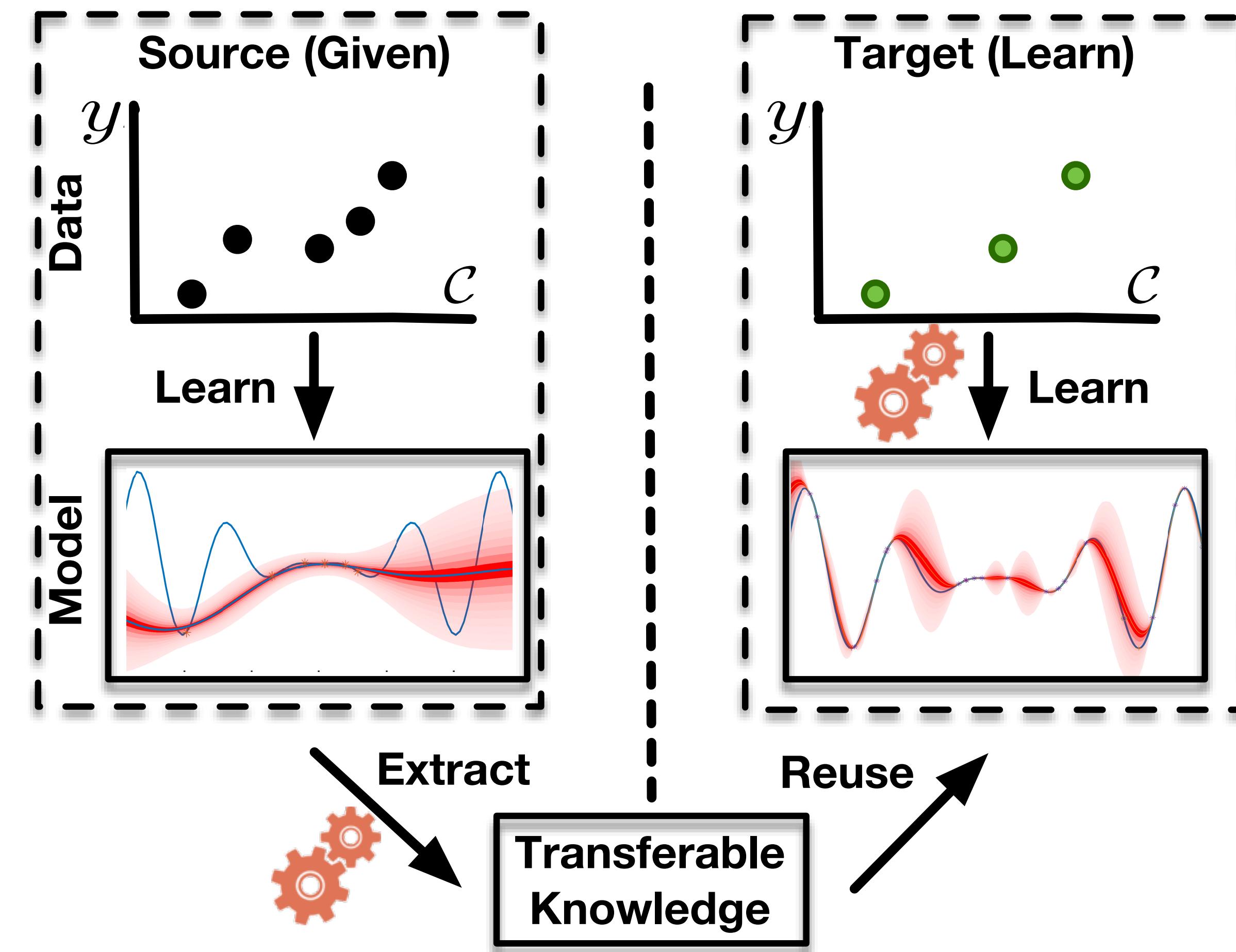
We also explore the configuration space using pseudo-random sampling to detect missing interactions



For capturing options and interactions that only appears in the target, L2S relies on exploration (random sampling)



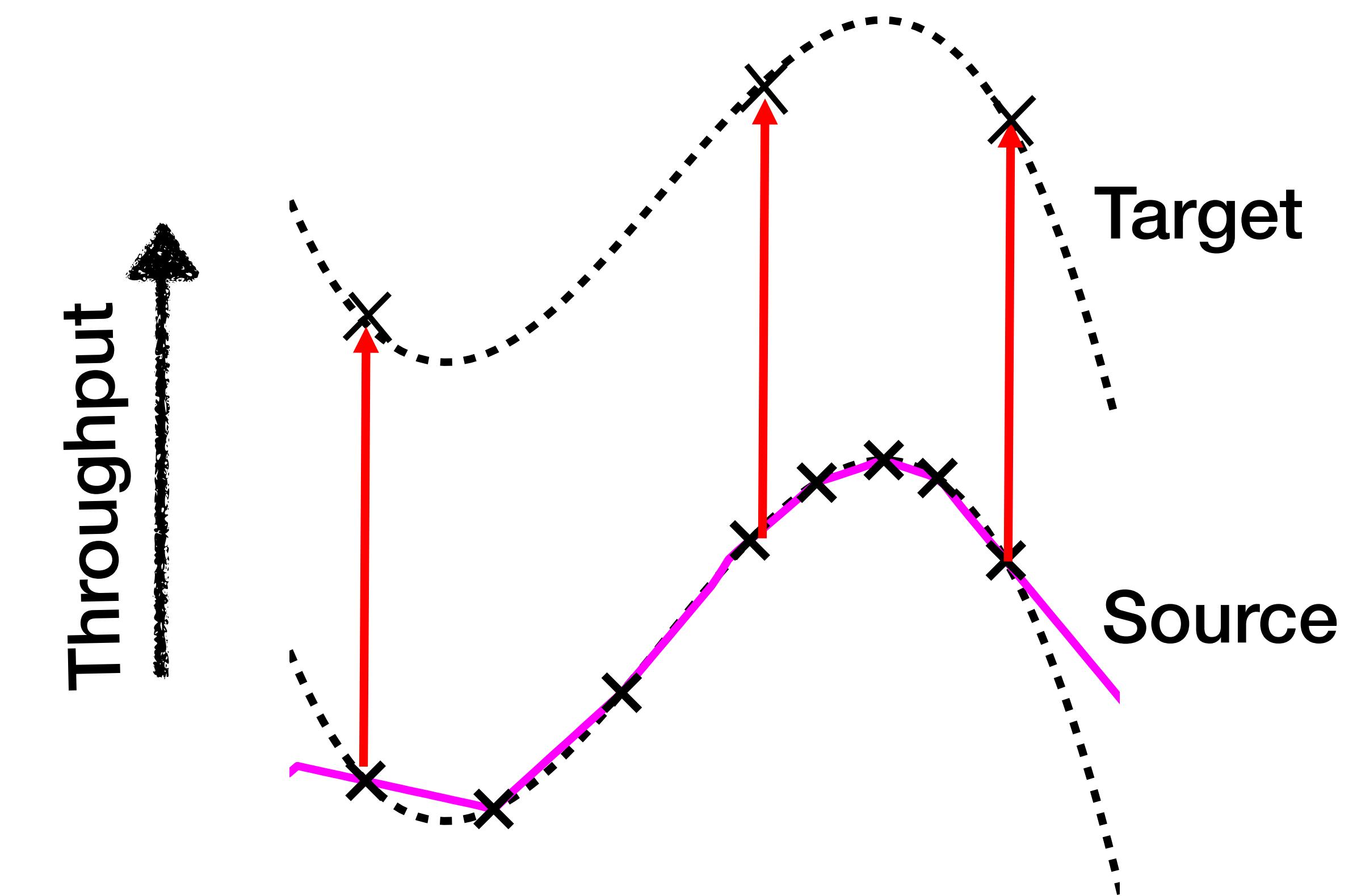
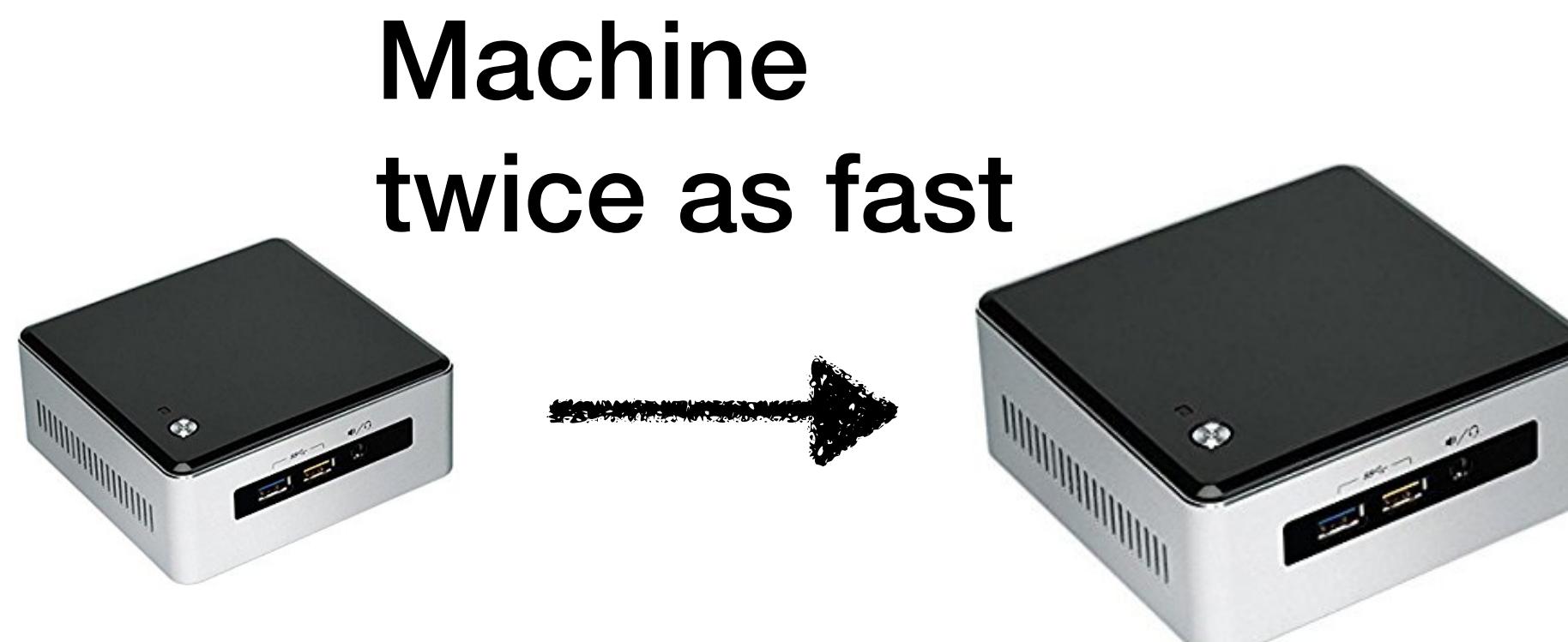
L2S transfers knowledge about the structure of performance models from the source to guide the sampling in the target environment



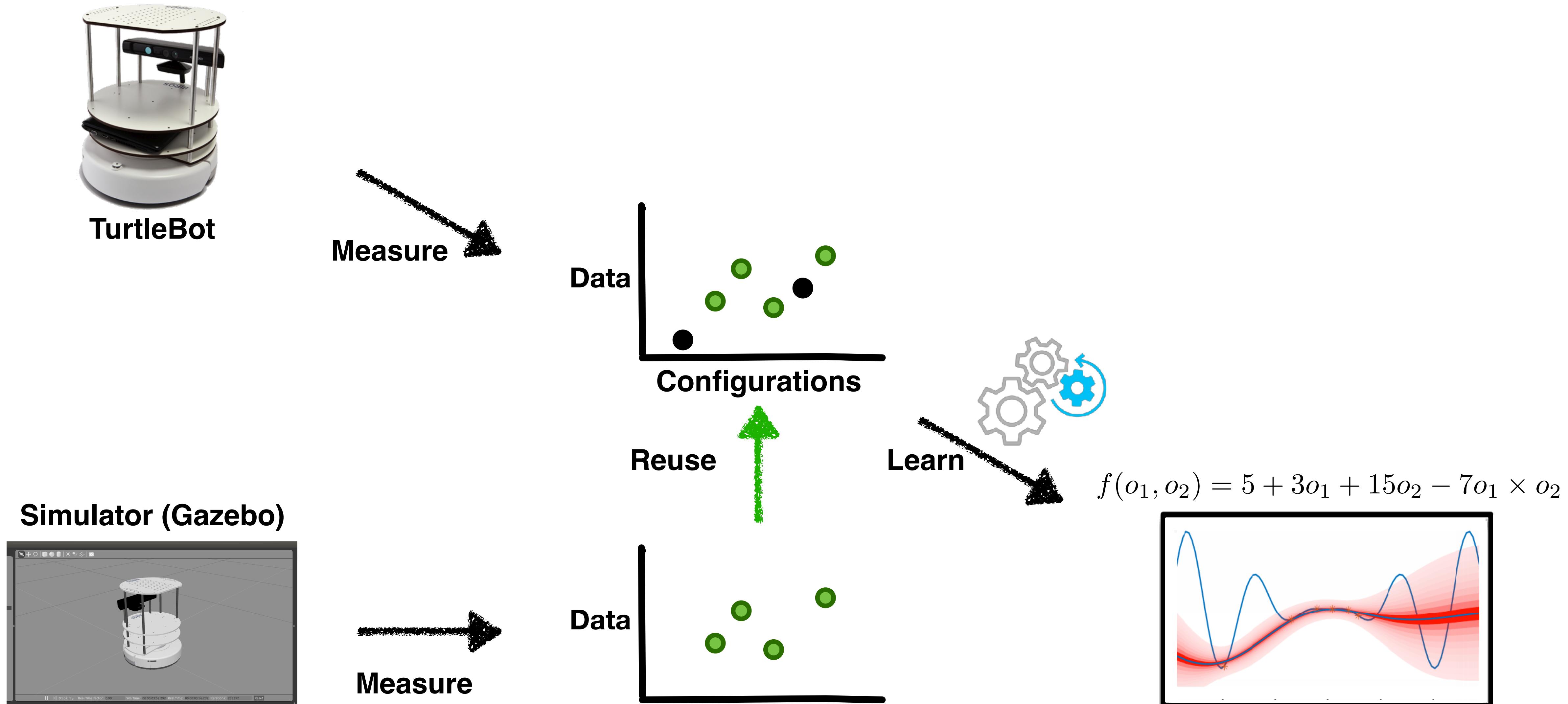
$$\hat{f}_s(\cdot) = 1.2 + 3o_1 + 5o_3 + 0.9o_7 + 0.8o_3o_7 + 4o_1o_3o_7$$

**Evaluation:**  
Other transfer learning approaches  
also exists

**“Model shift”** builds a model in the source and uses the shifted model “directly” for predicting the target



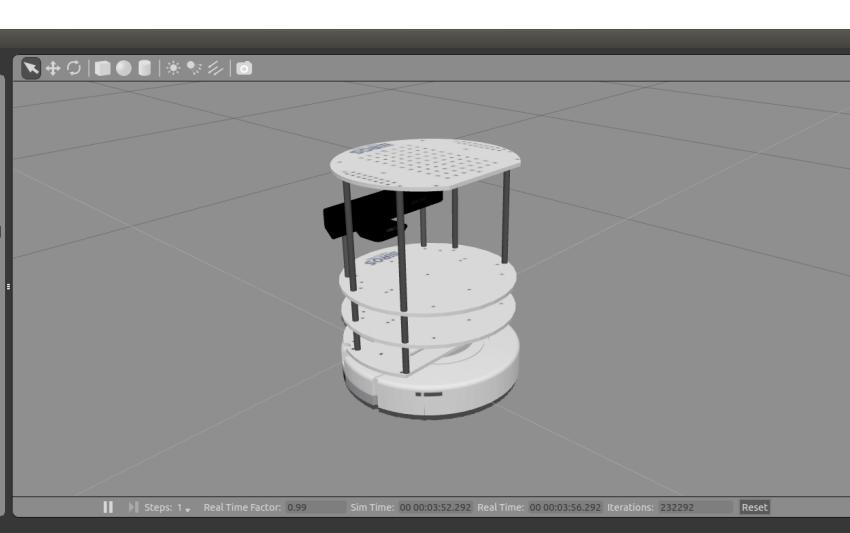
“Data reuse” combines the data collected in the source with the ones in the target in a “multi-task” setting to predict the target



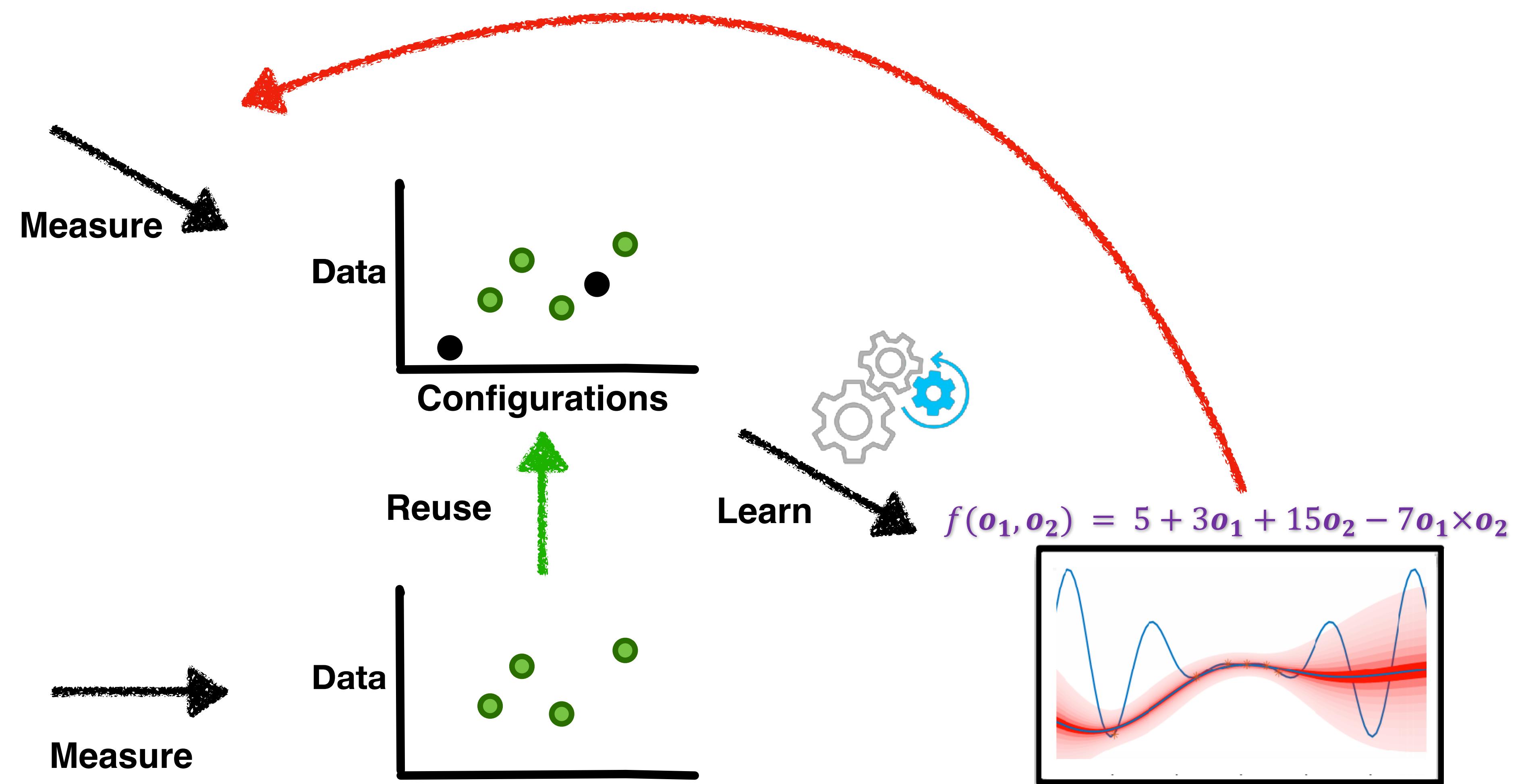
# “Data reuse” with guided sampling



TurtleBot

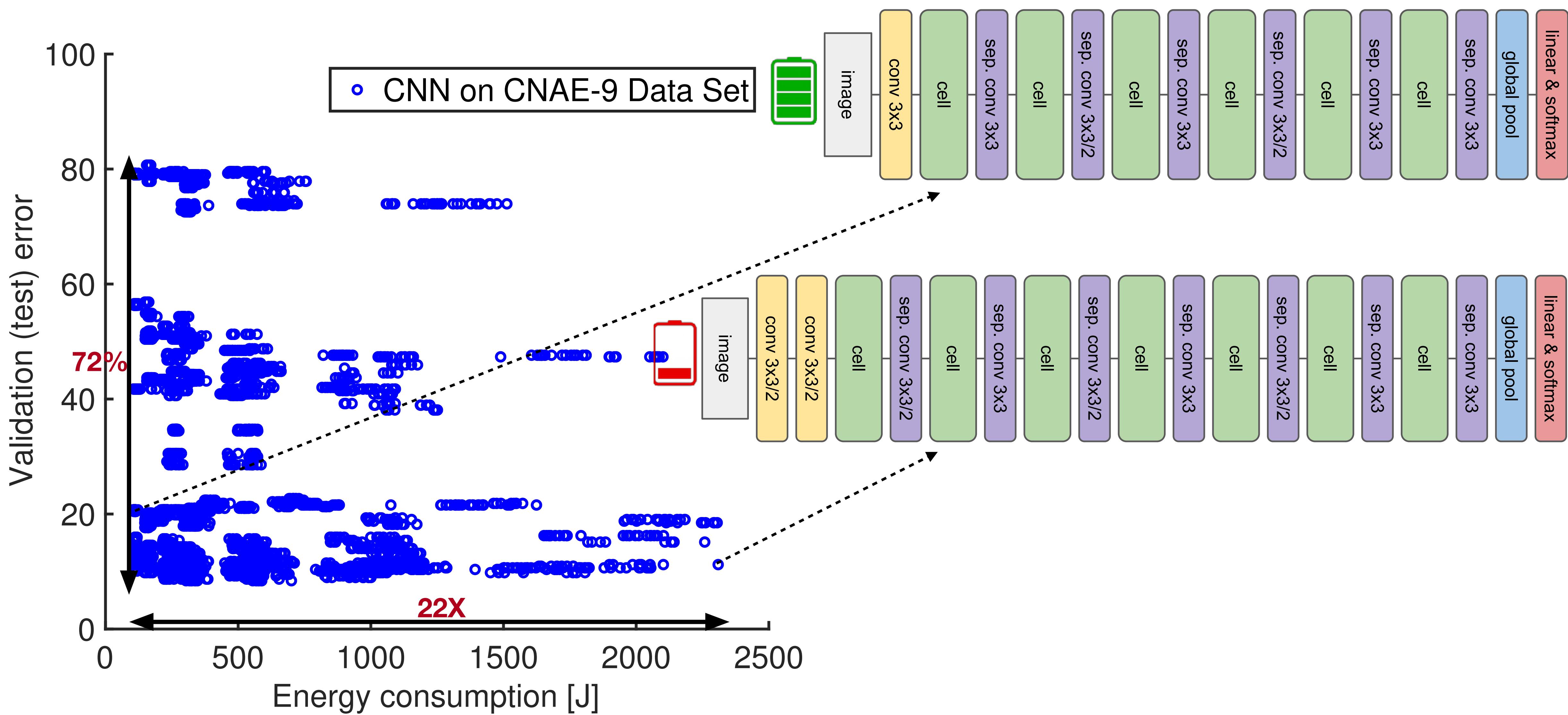


Simulator (Gazebo)

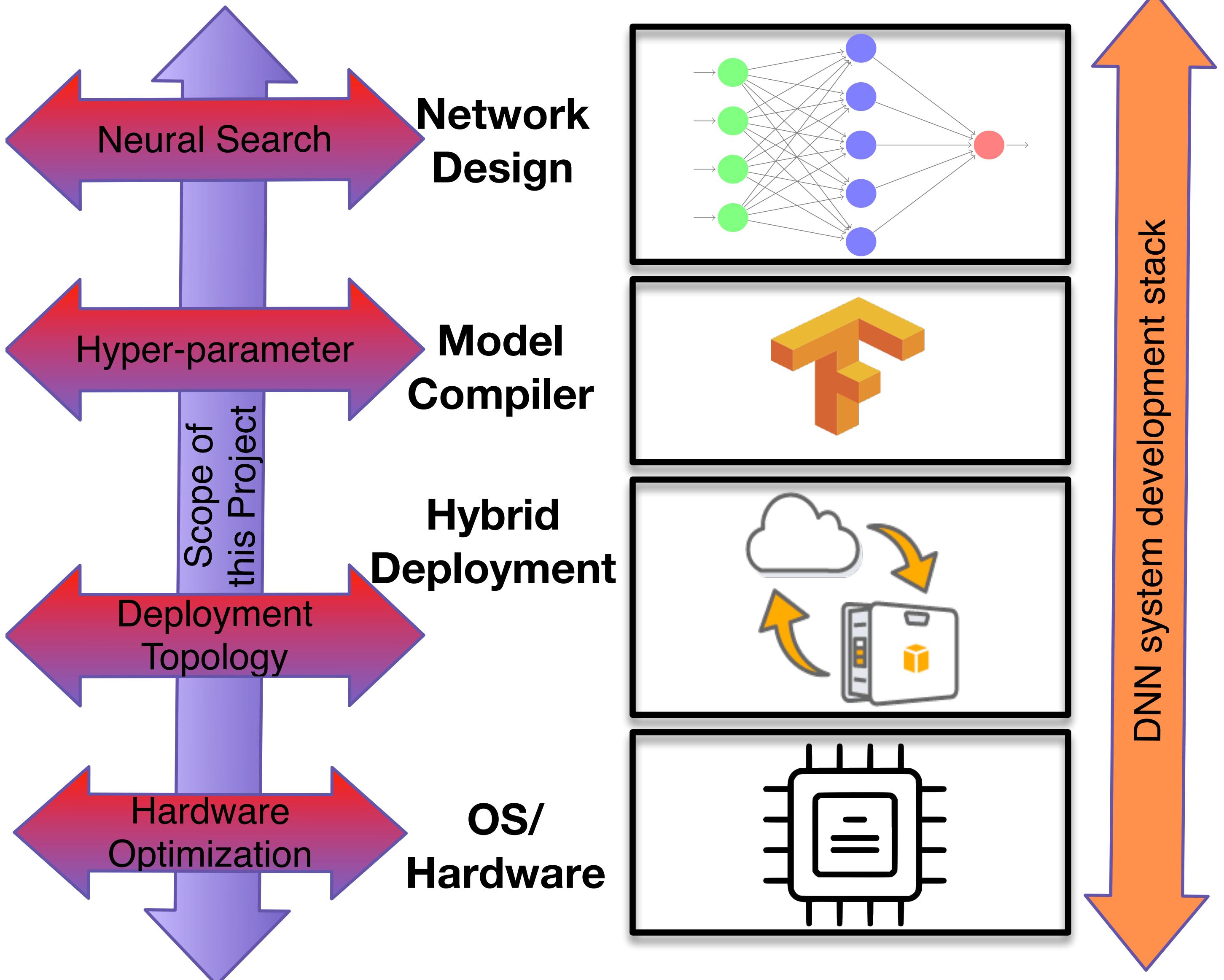


# Evaluation: Learning performance behavior of Machine Learning Systems

# Configurations of deep neural networks affect accuracy and energy consumption



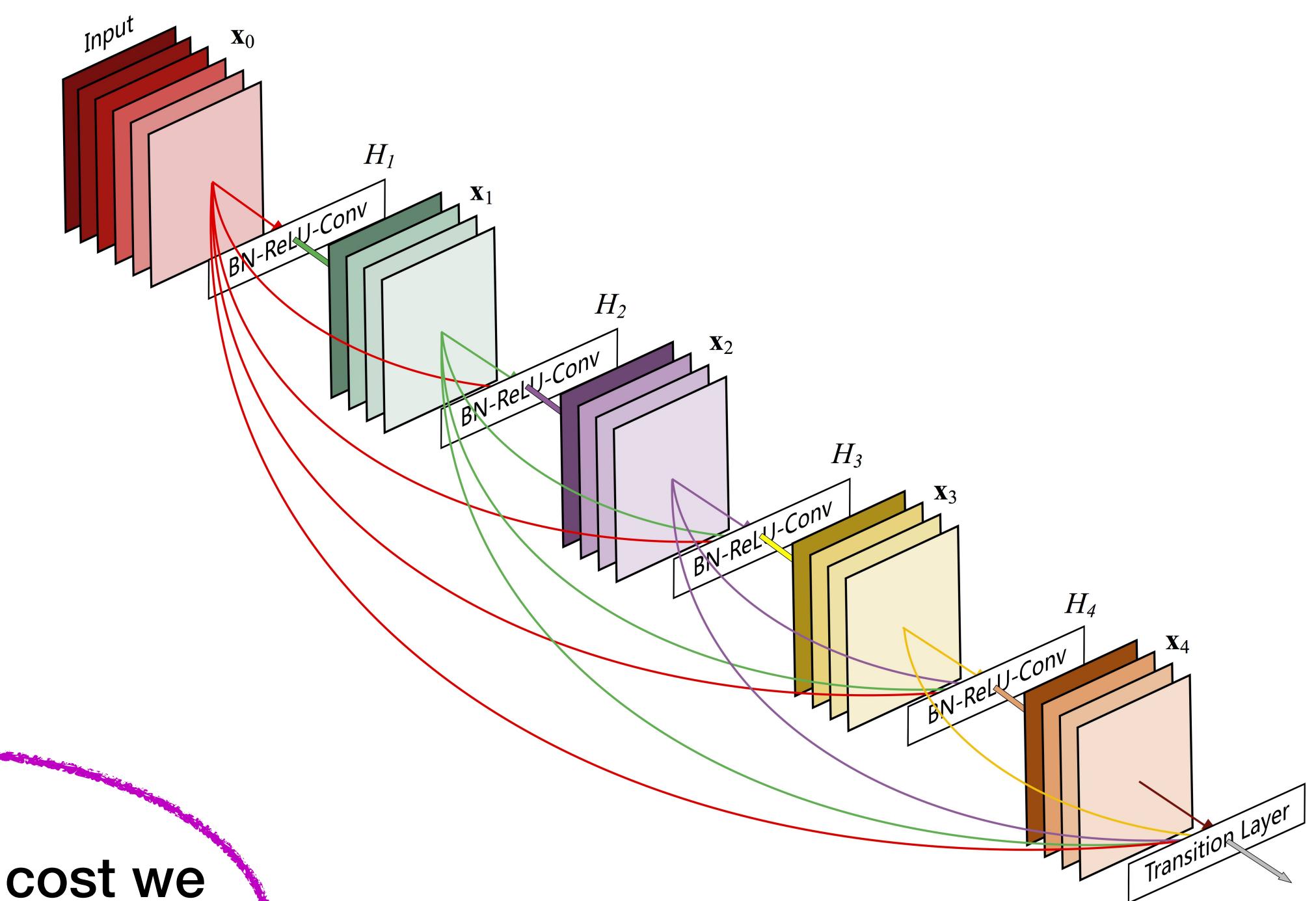
# Deep neural network as a highly configurable system



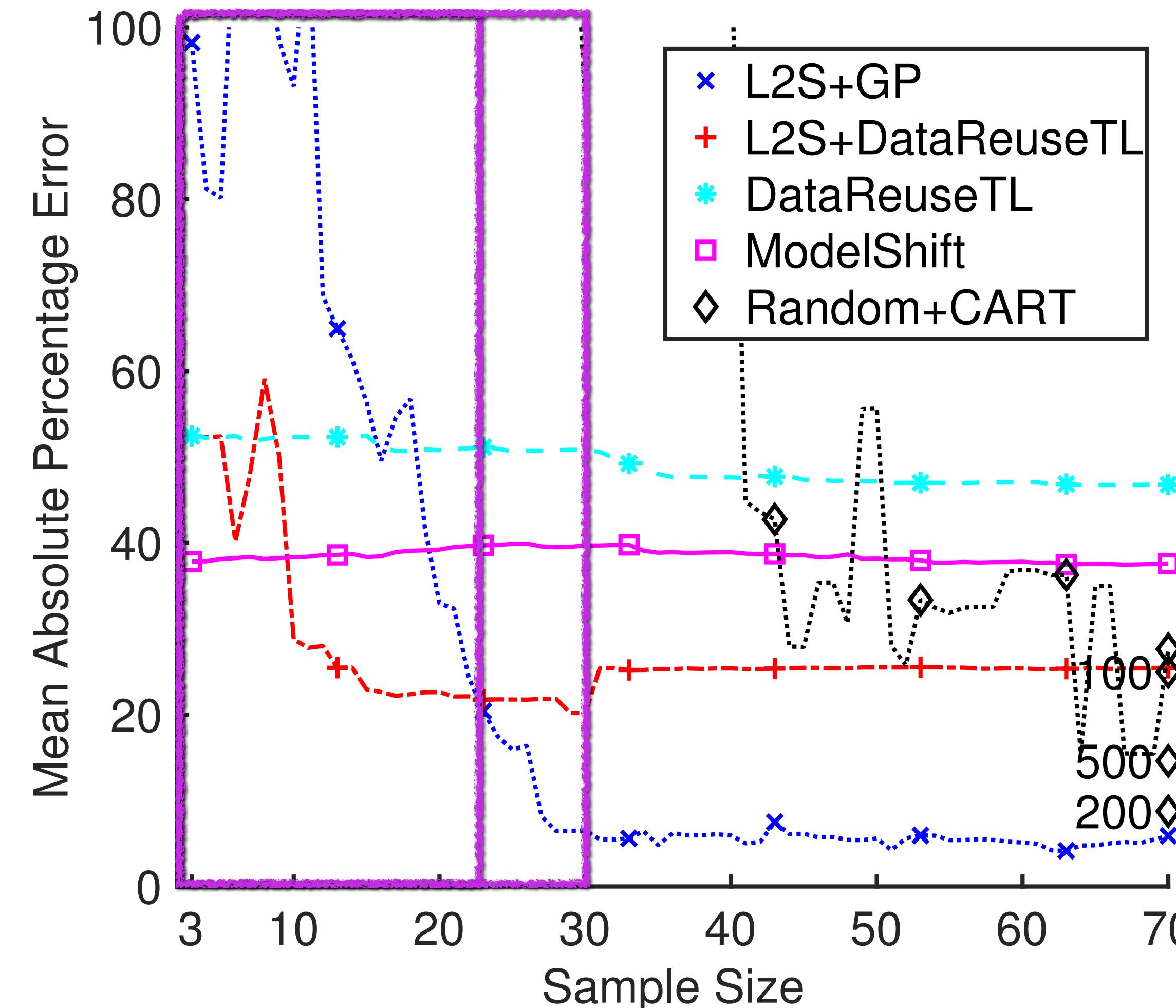
# DNN measurements are costly

Each sample cost ~1h  
 $4000 * 1\text{h} \approx 6 \text{ months}$

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

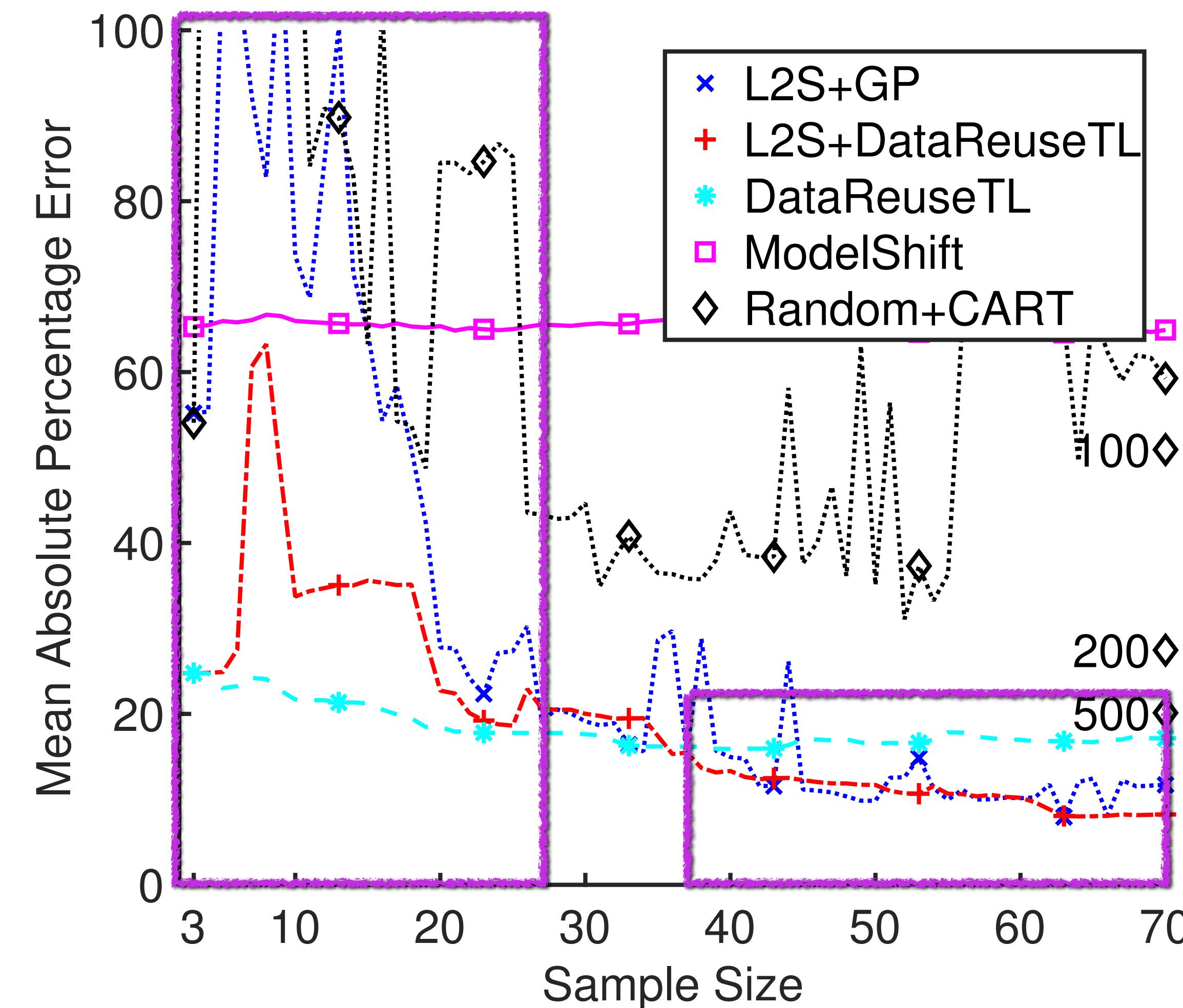


# L2S enables learning a more accurate model with less samples exploiting the knowledge from the source



## Convolutional Neural Network

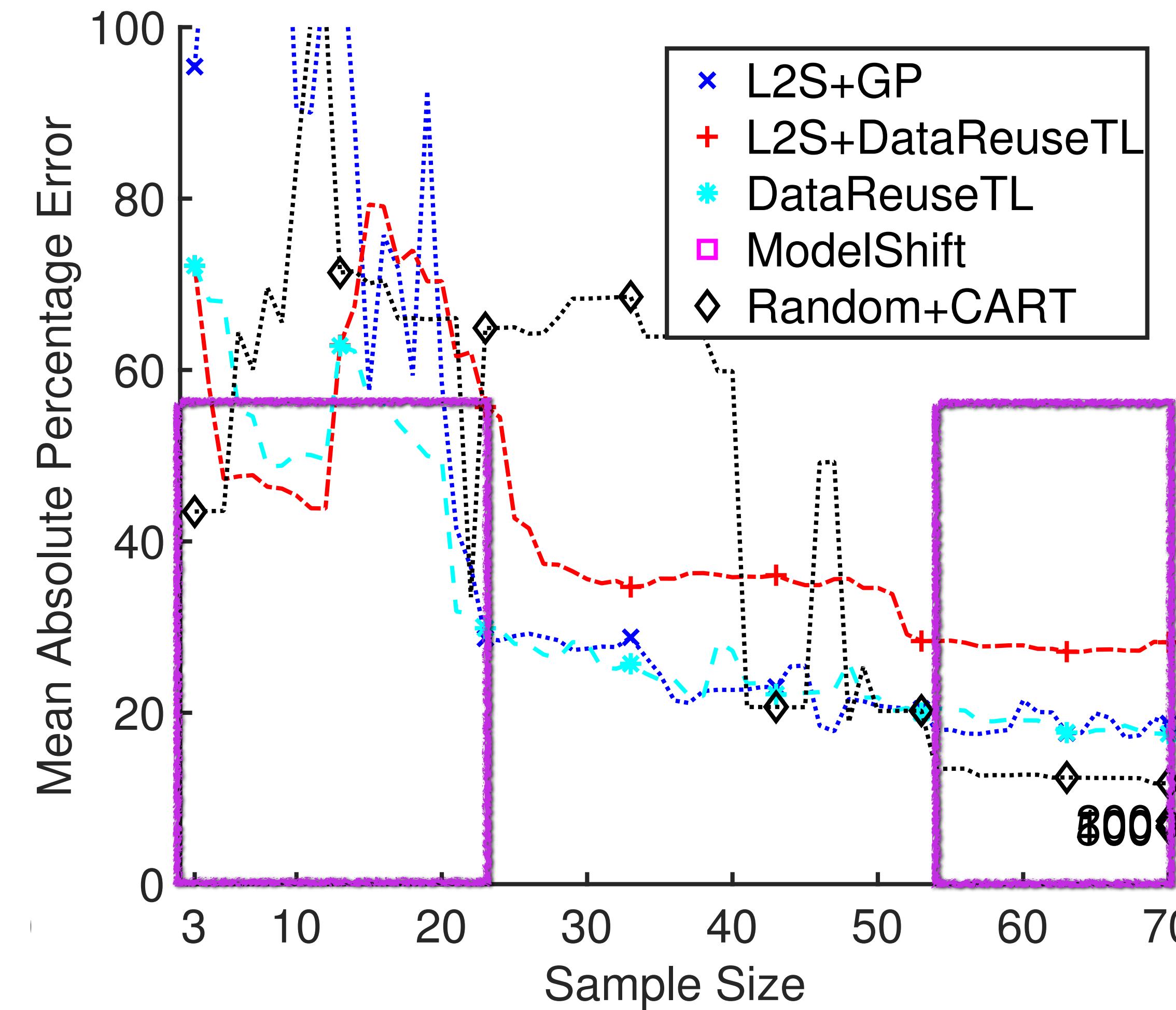
# L2S may also help data-reuse approach to learn faster



XGBoost

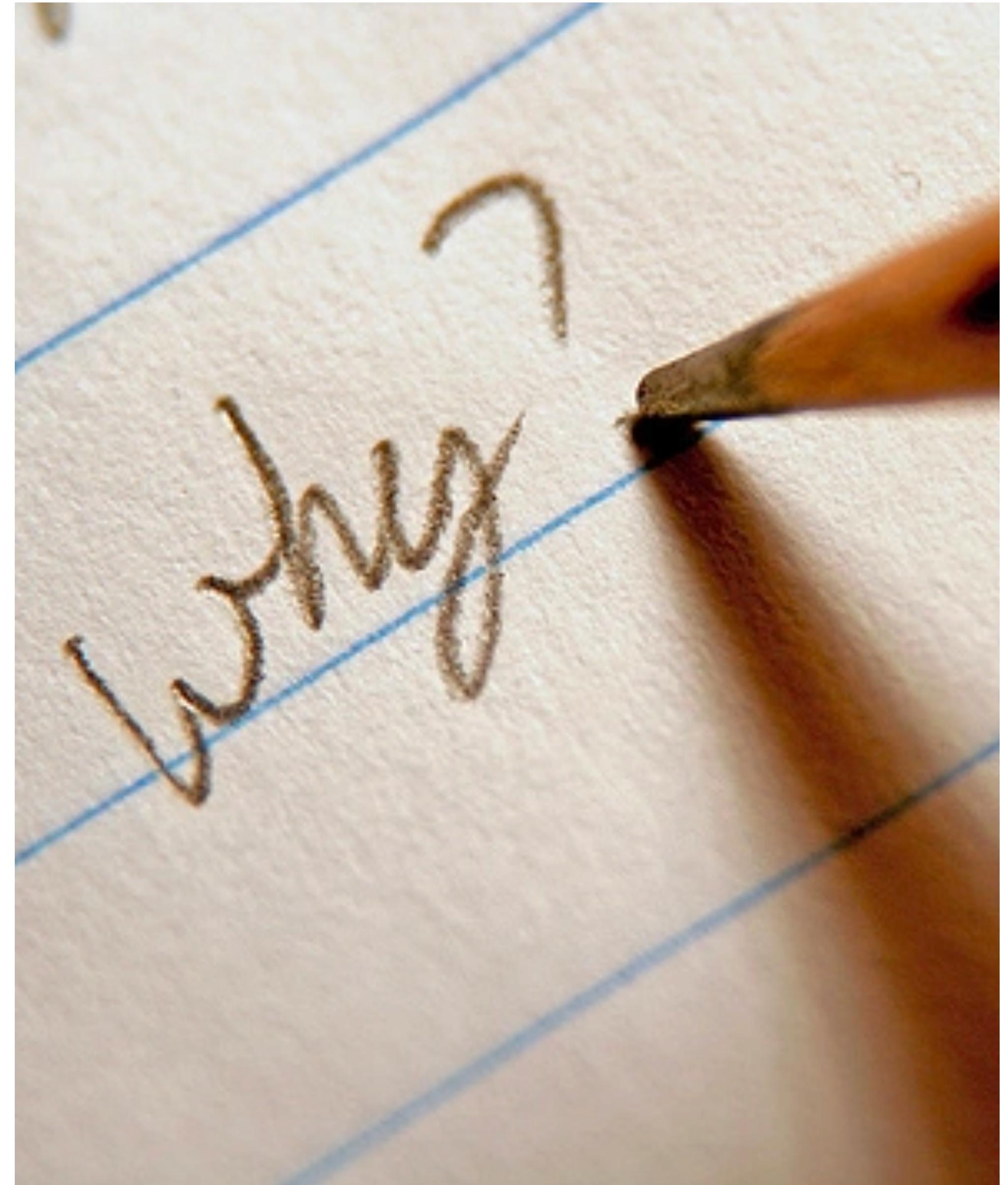
# **Evaluation: Learning performance behavior of Big Data Systems**

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

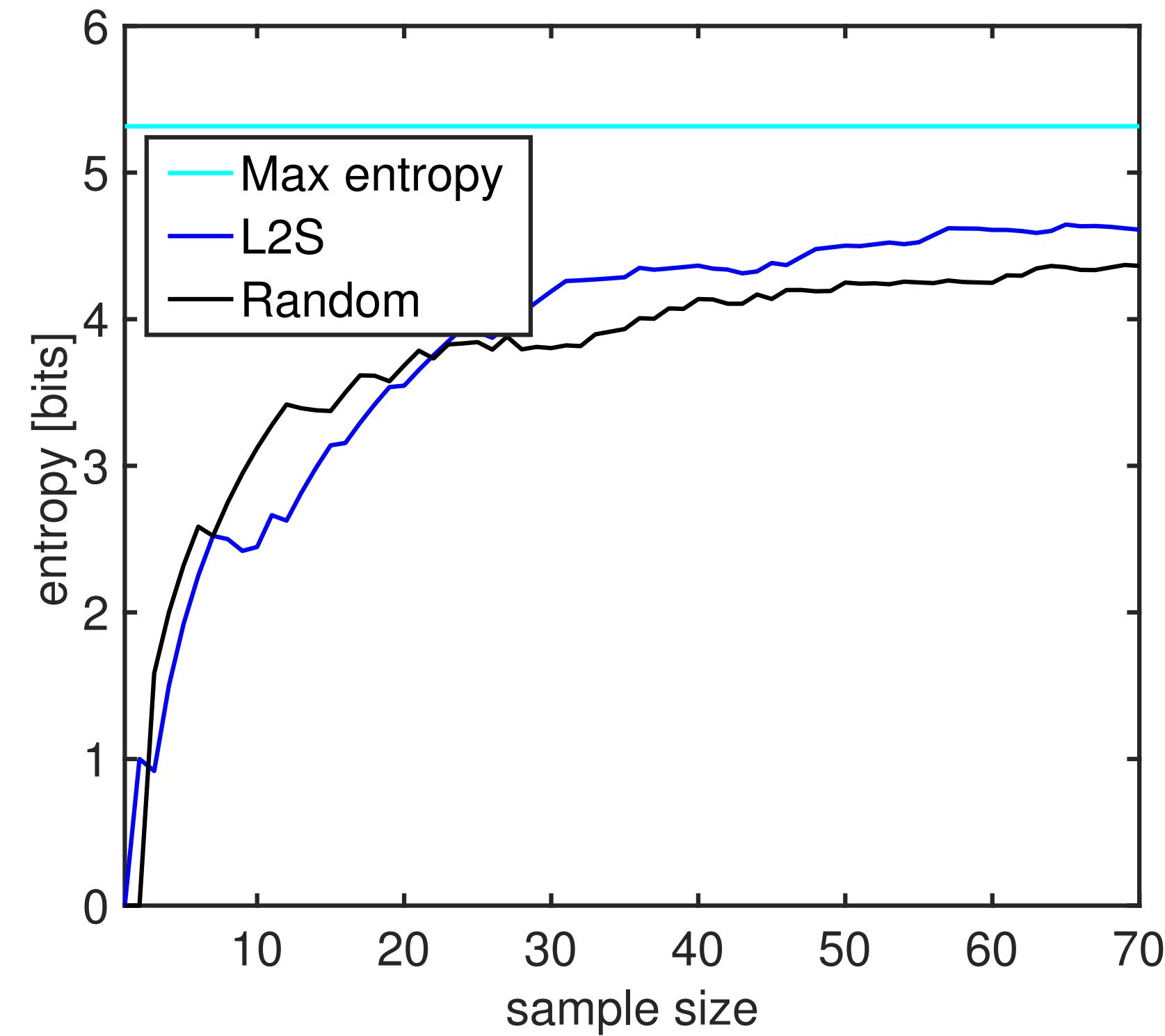


Apache Storm

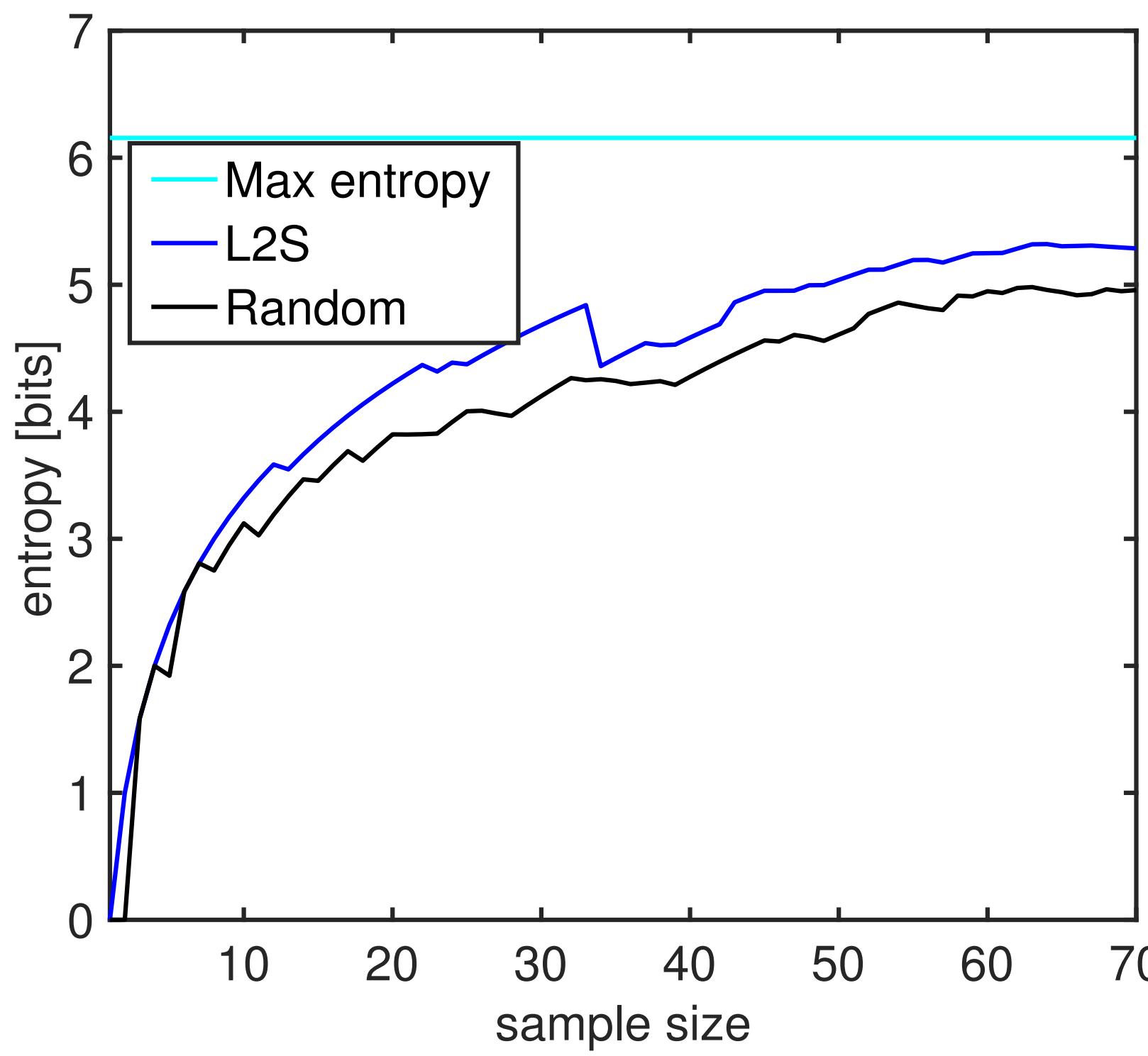
# Why performance models using L2S sample are more accurate?



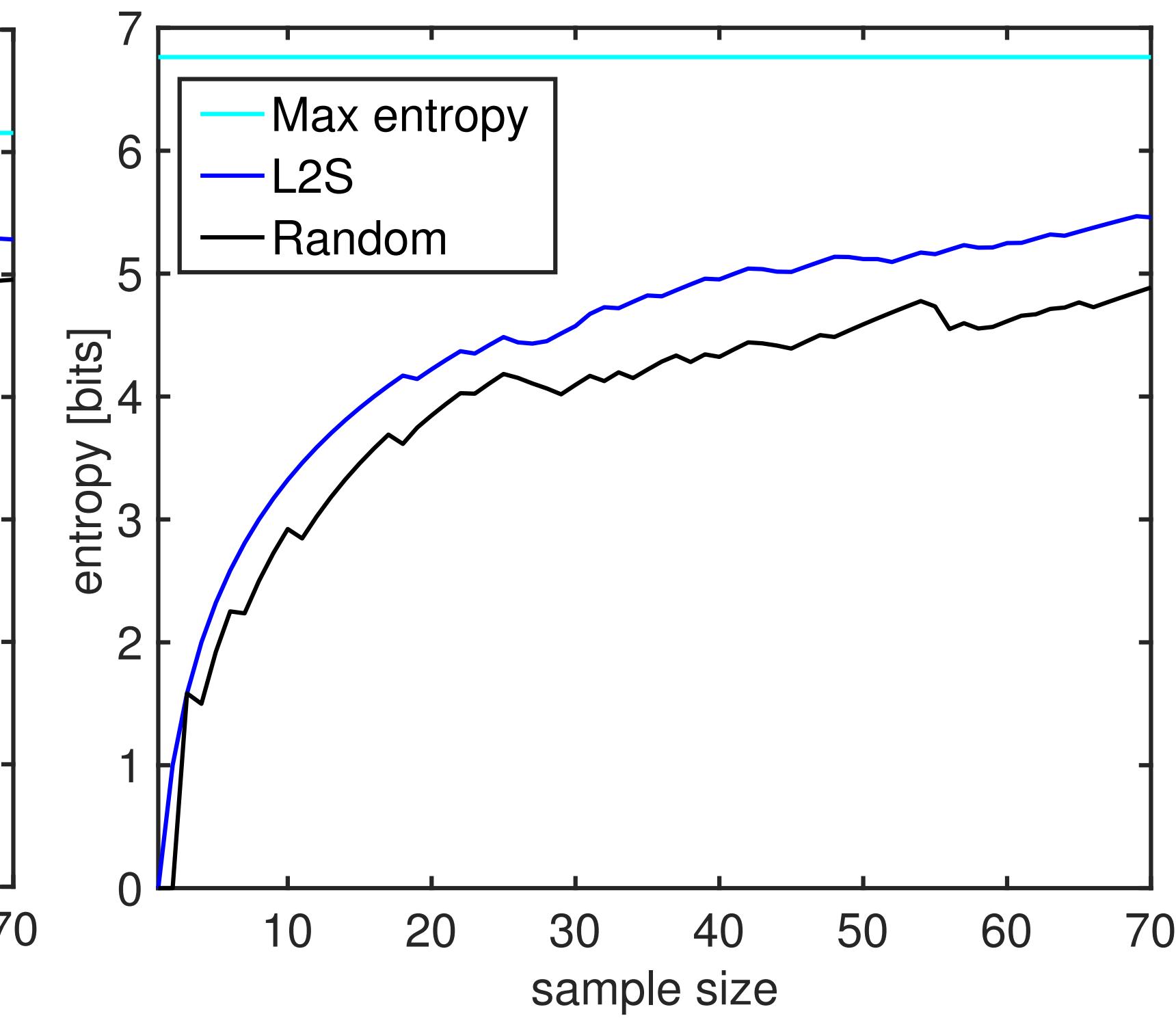
# The samples generated by L2S contains more information... “entropy $\leftrightarrow$ information gain”



DNN



XGboost



Storm

# 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-

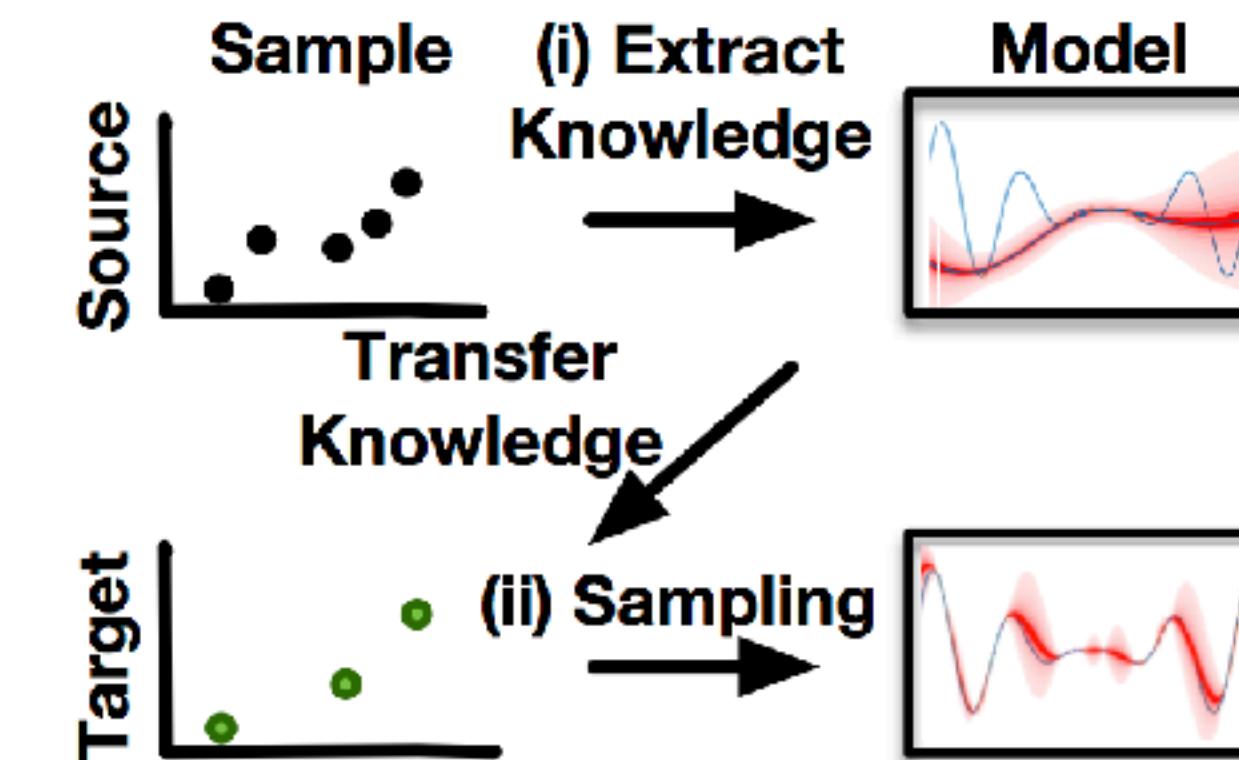
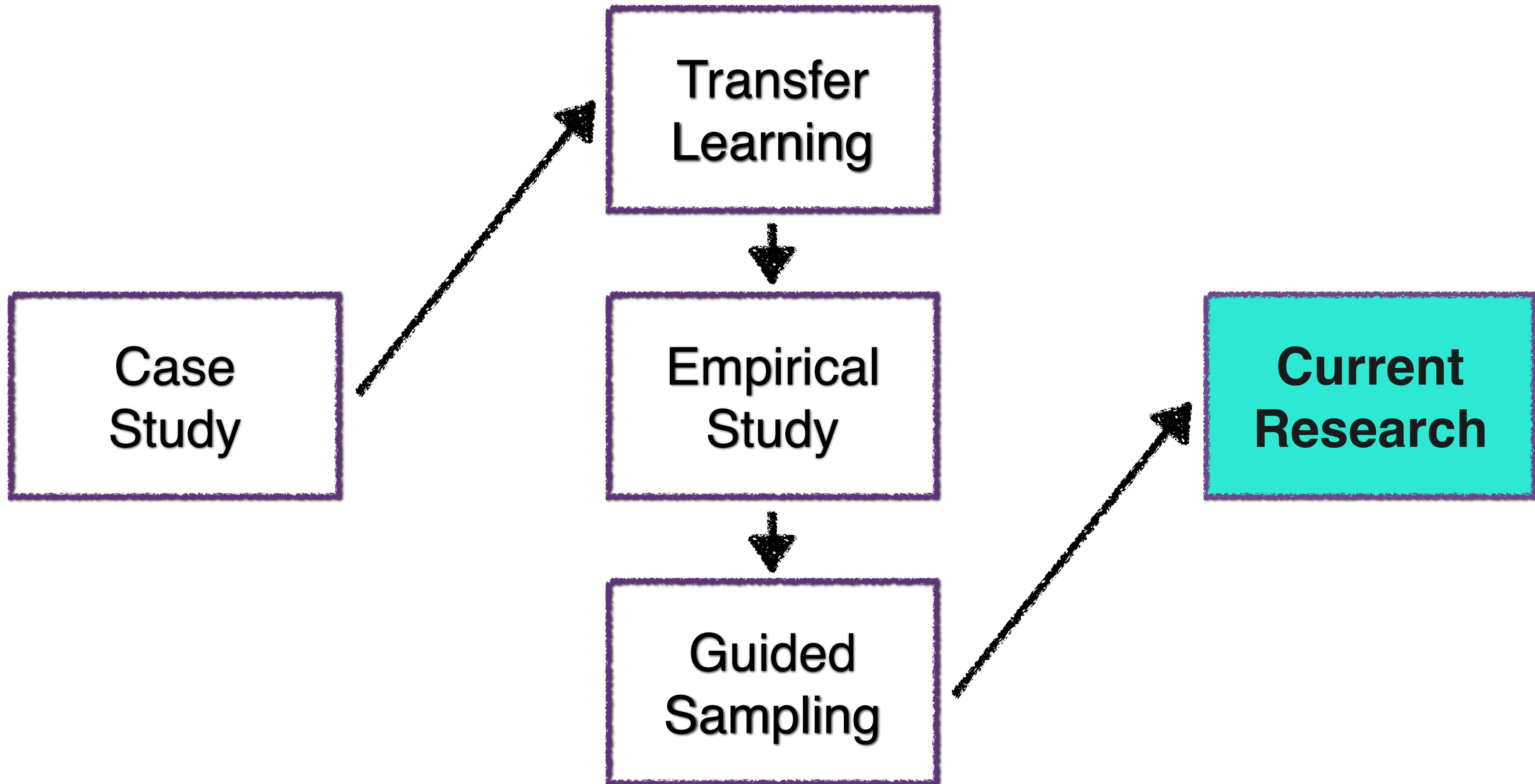


Figure 1: L2S performs guided sampling employing the knowledge extracted from a source environment.

# Outline

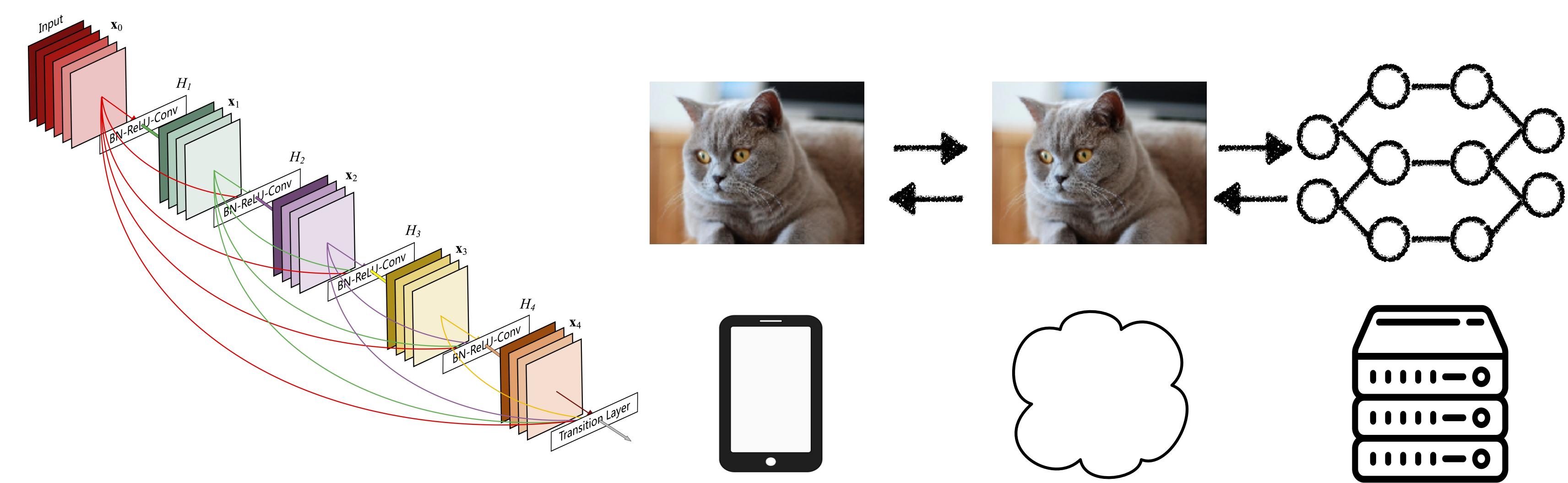
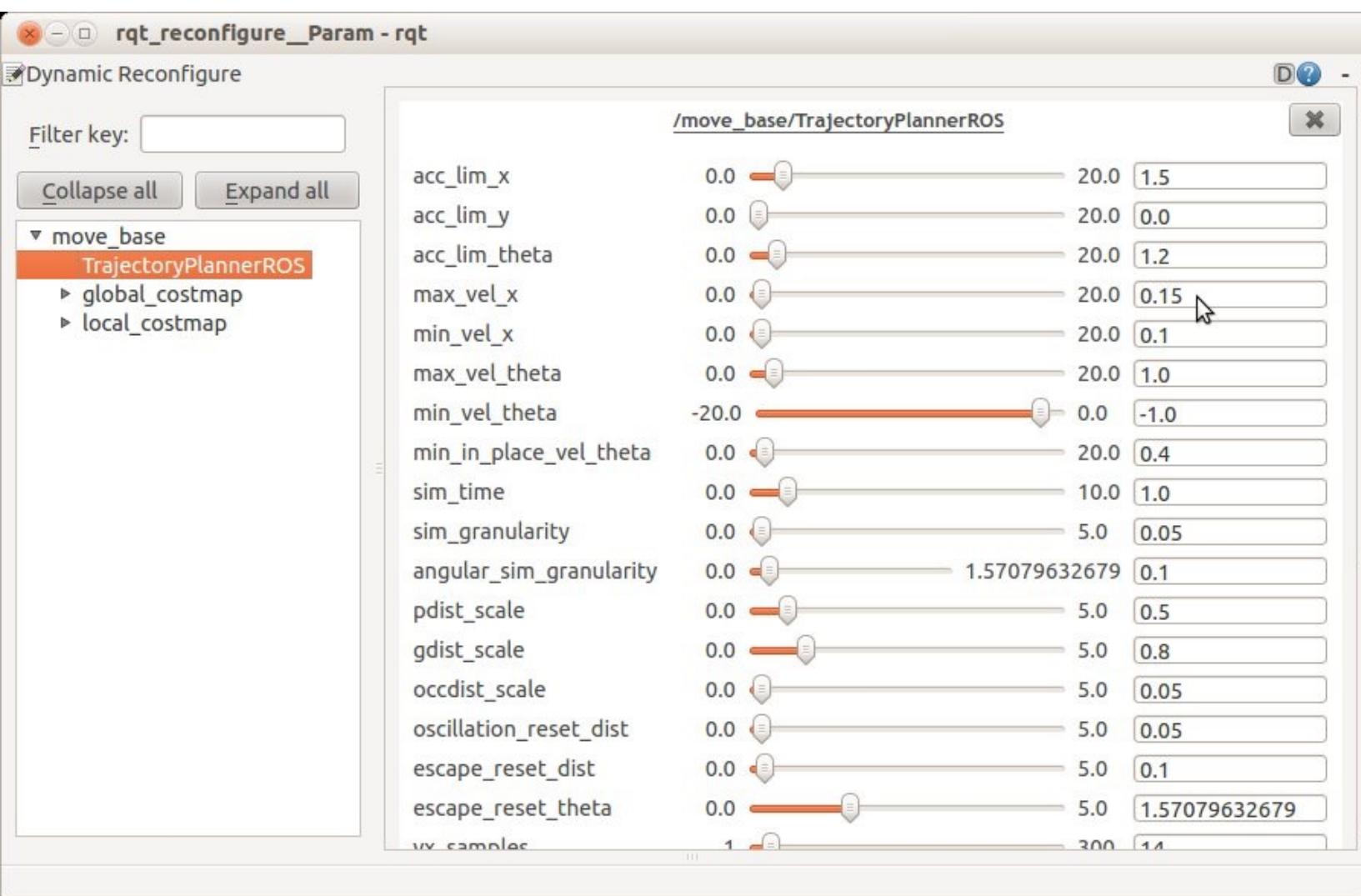




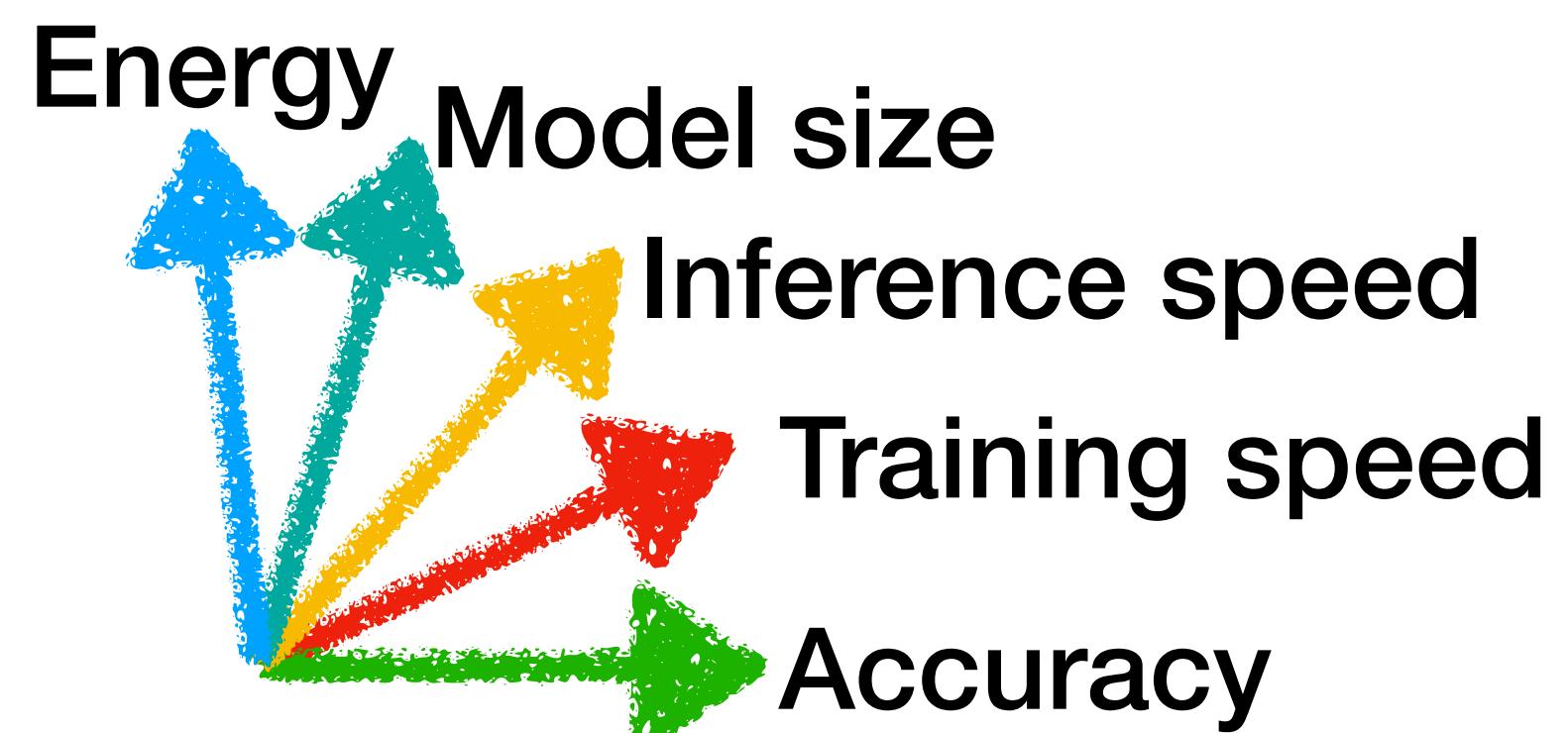
What will the software systems  
of the future look like?

# Software 2.0

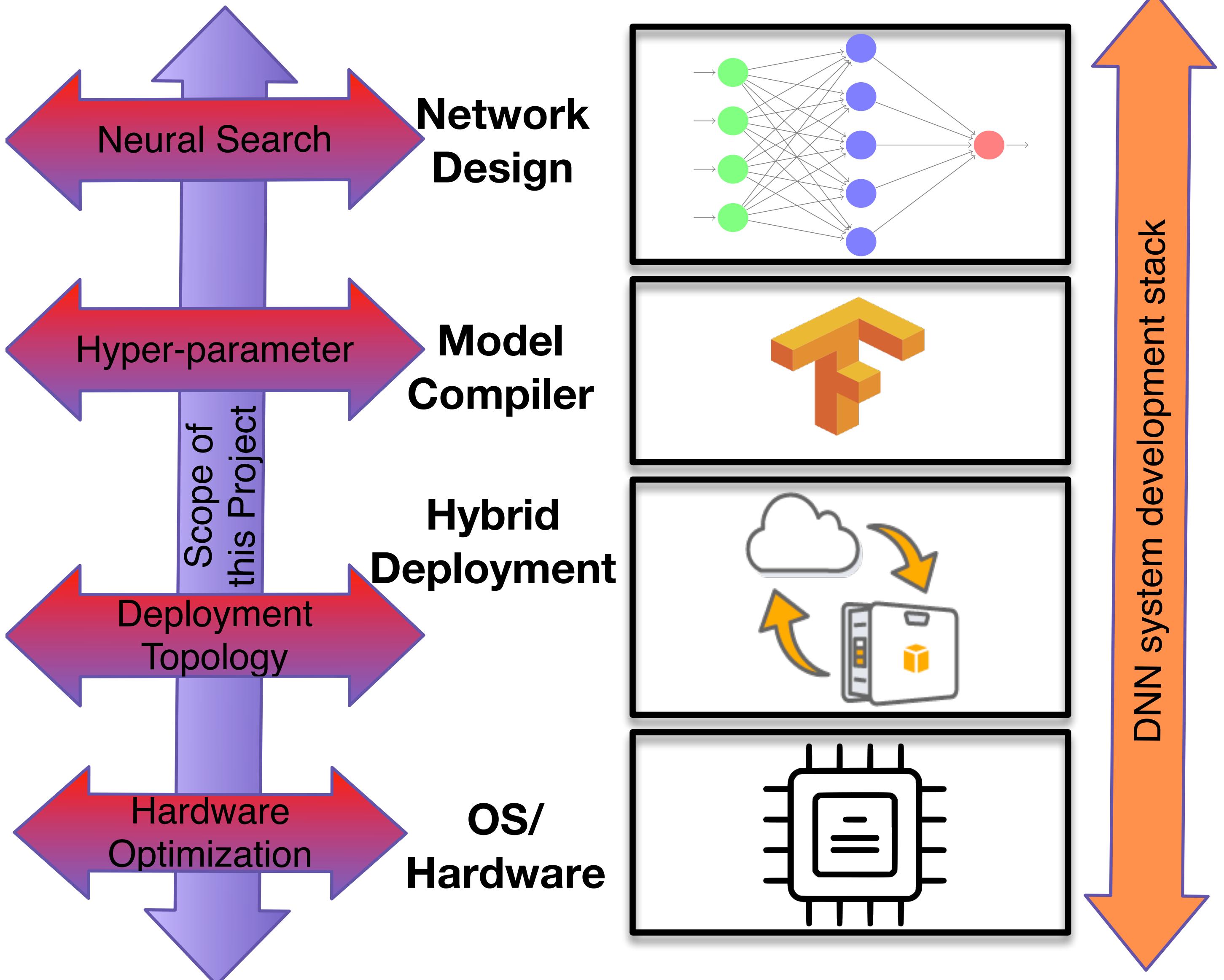
Increasingly **customized** and **configurable**



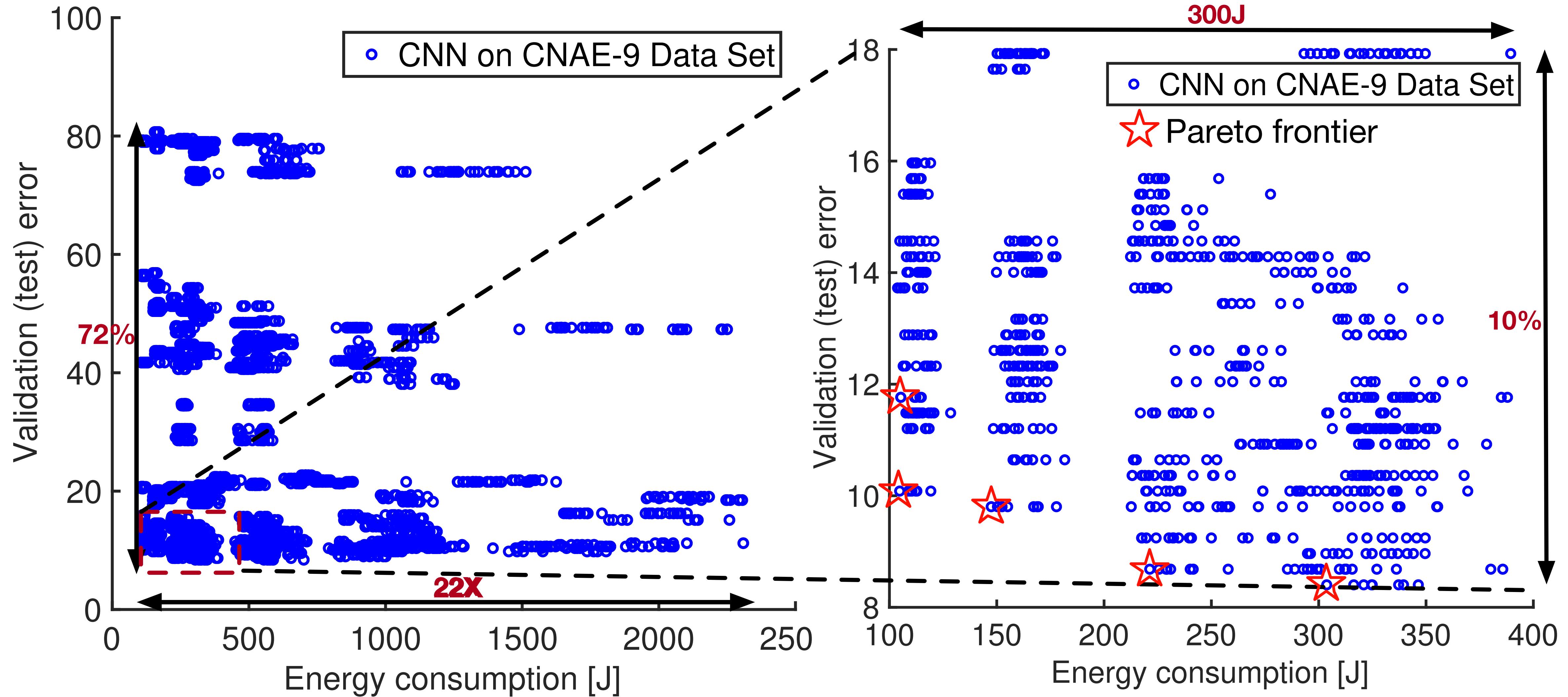
Increasingly **competing objectives**



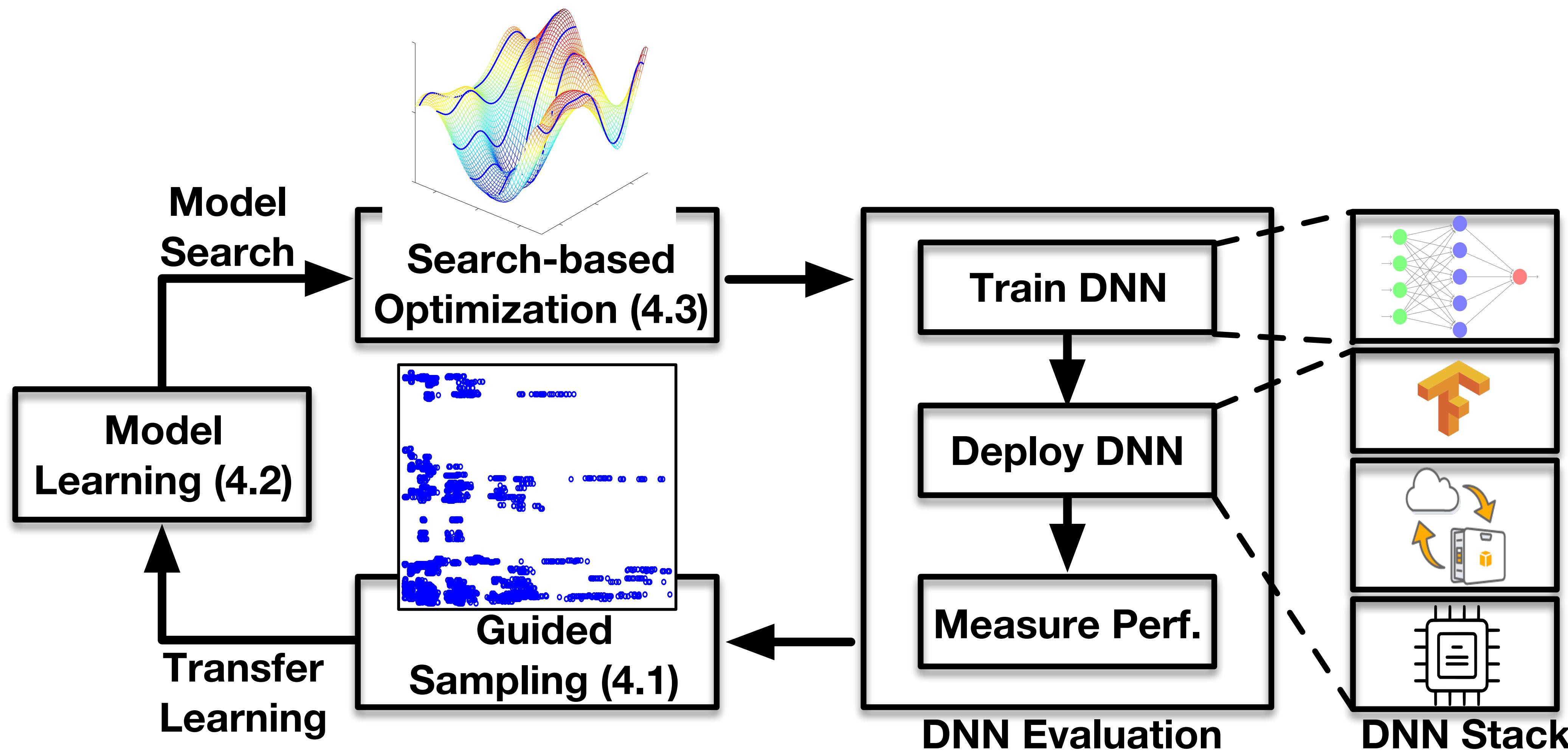
# Deep neural network as a highly configurable system



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



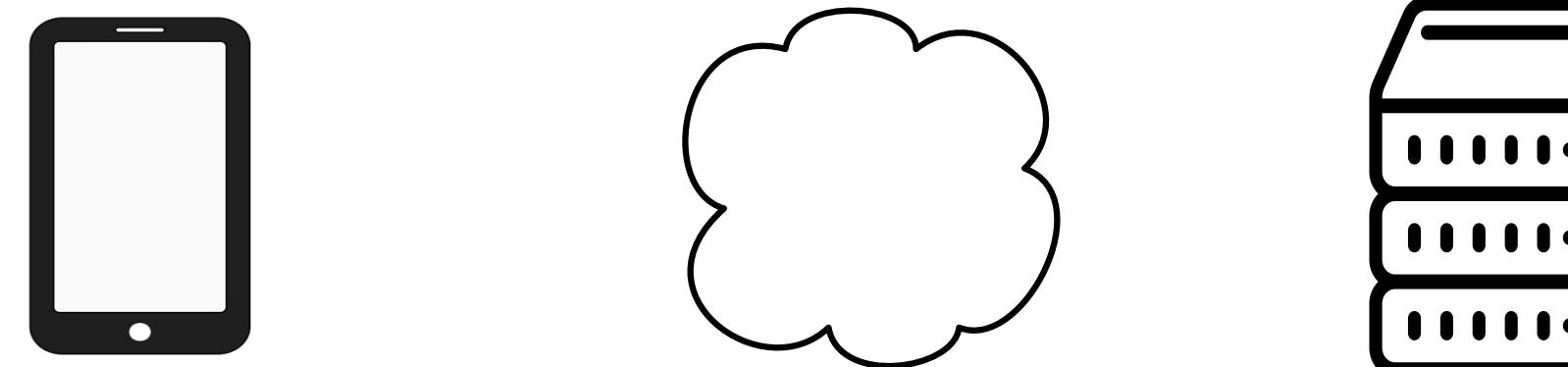
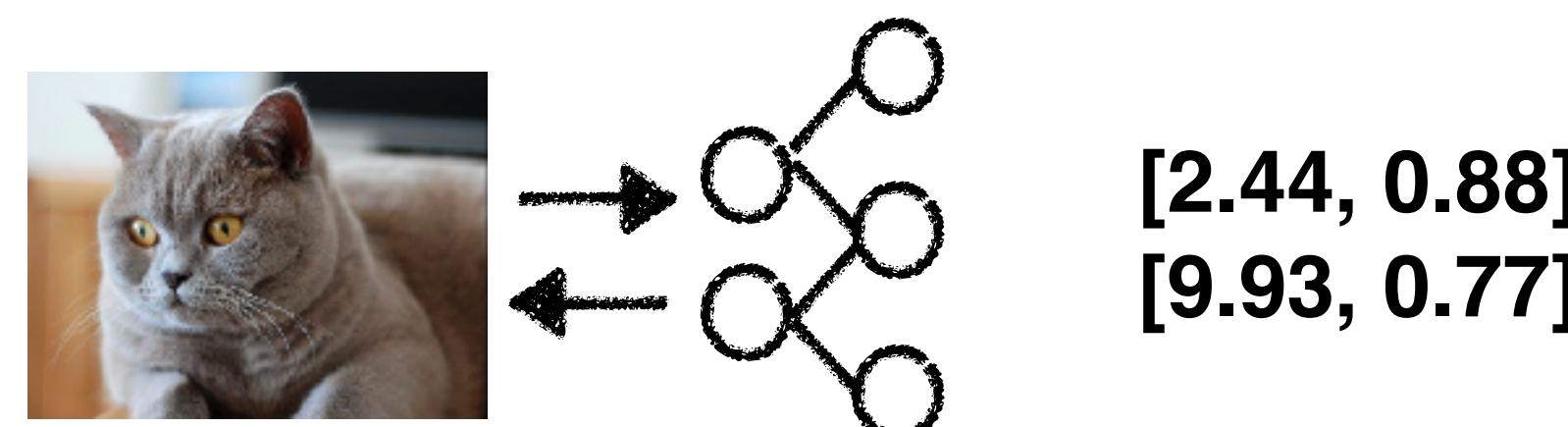
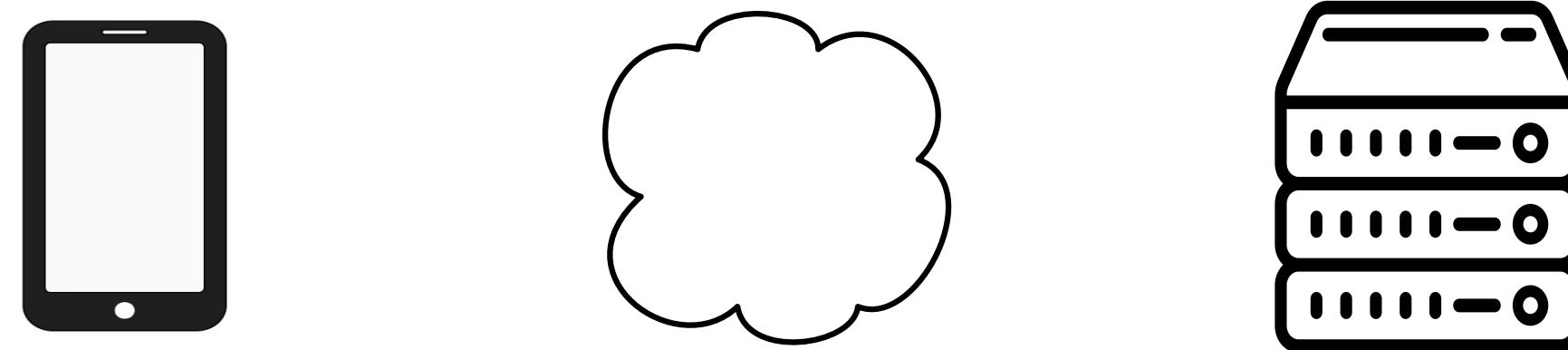
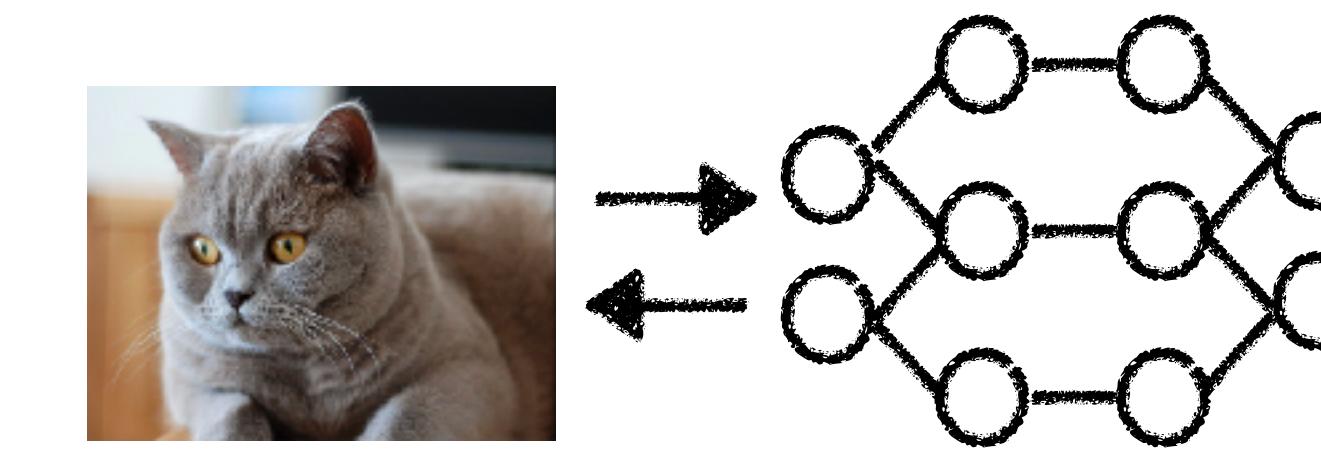
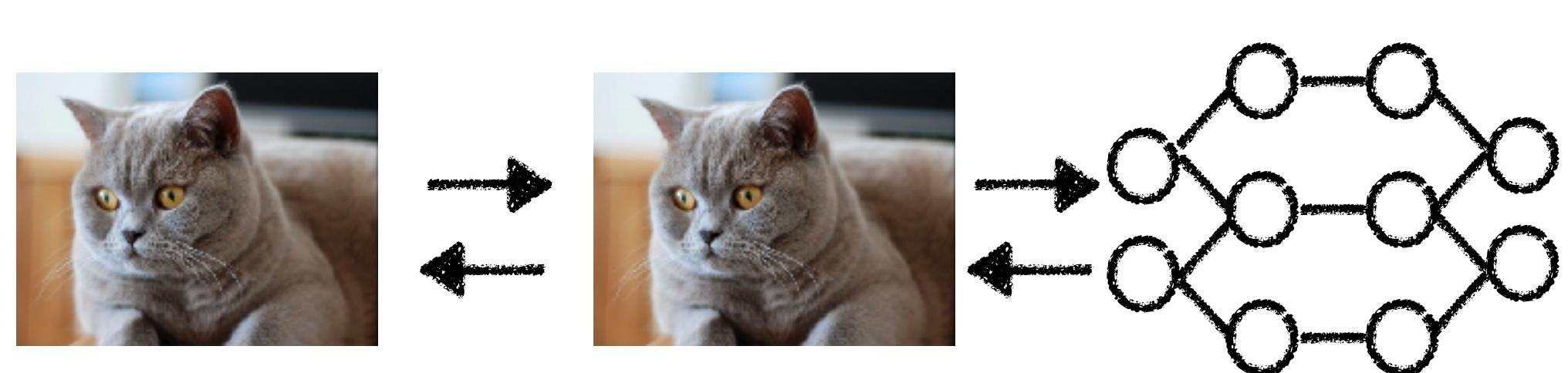
# Optimizing Energy Consumption of Deep Neural Networks



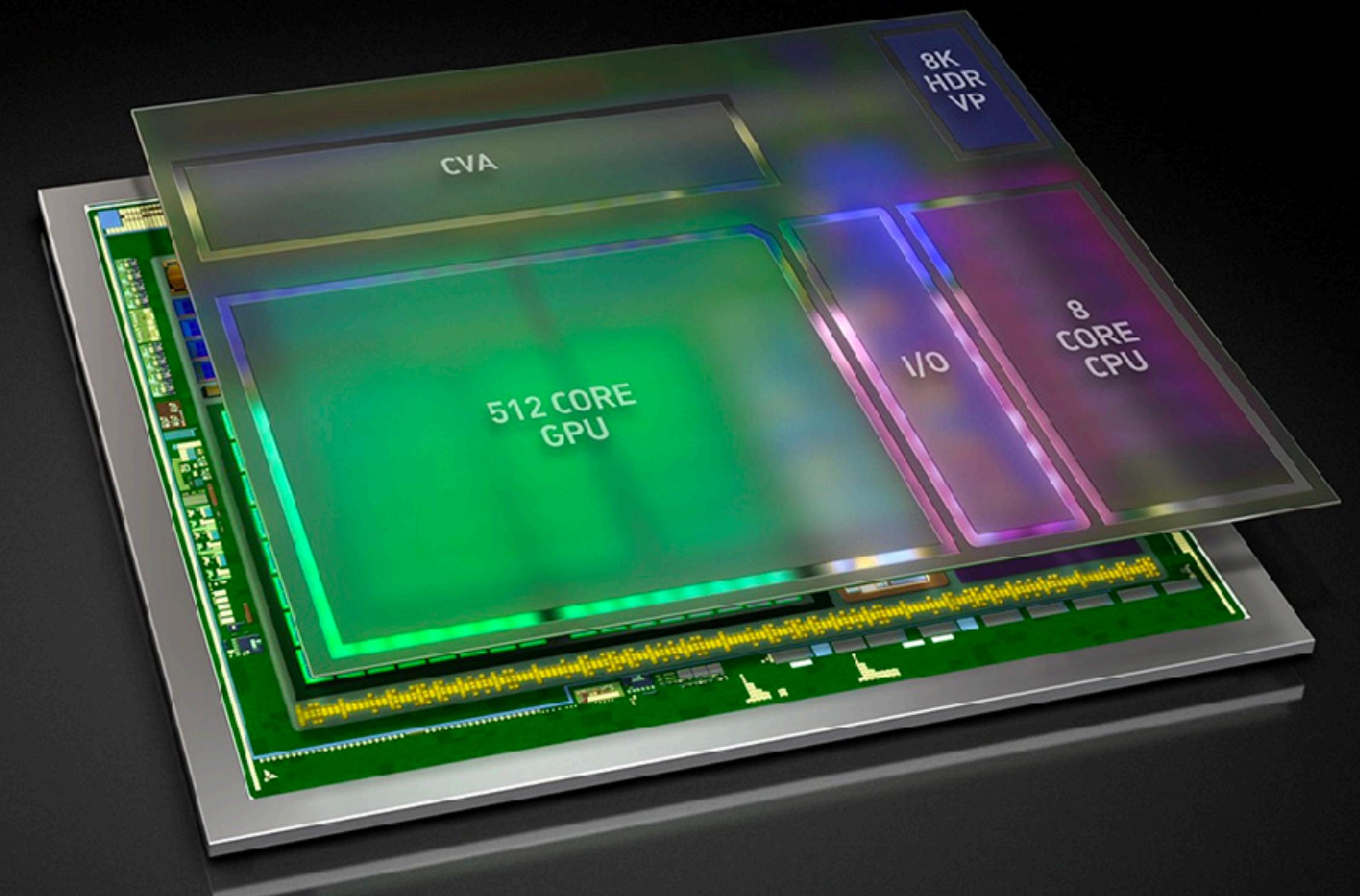
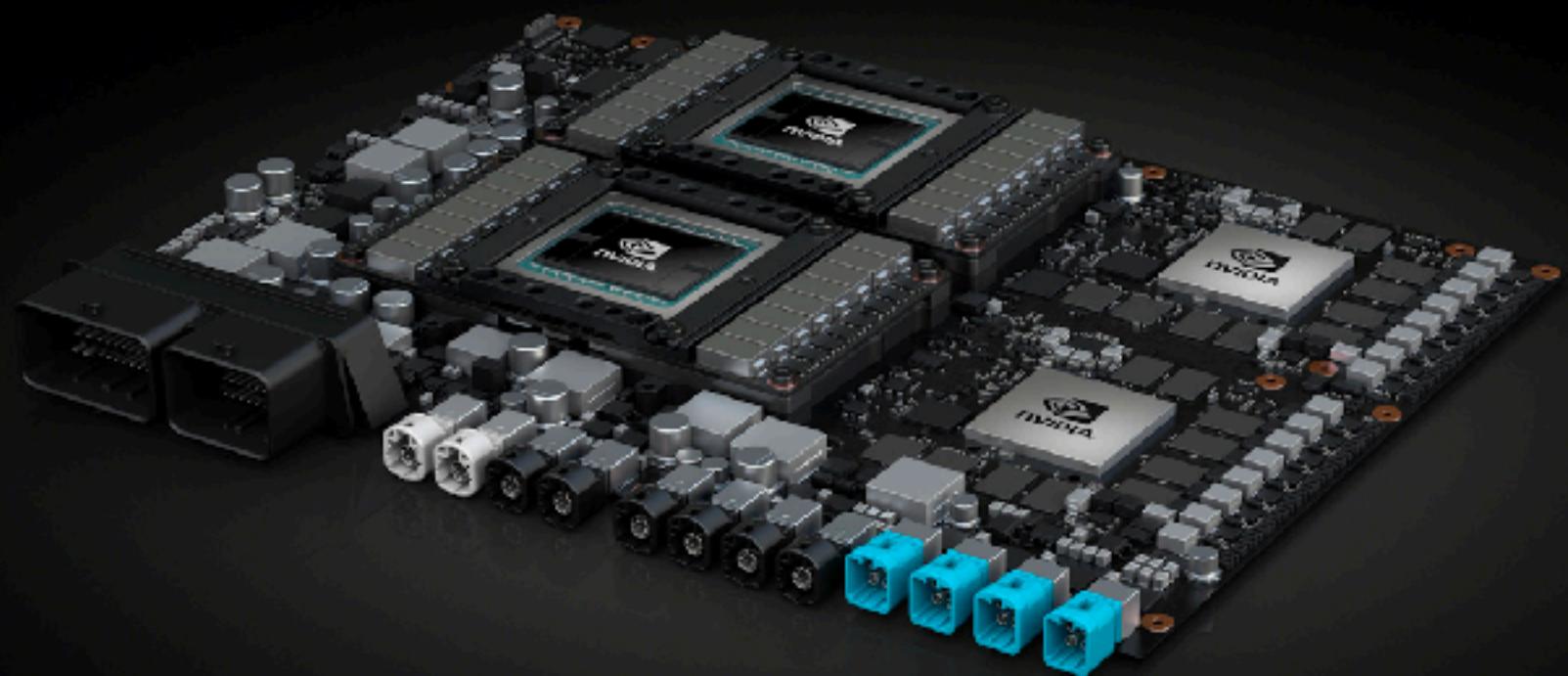
# Deep architecture design level variations

NAME	DESCRIPTION	TYPE	MIN	MAX
batch_size	SGD parameter	int	8	32
conv_1_filter_size	Architecture parameter	int	2	10
conv_1_num_filters	Architecture parameter	int	32	256
conv_2_filter_size	Architecture parameter	int	2	10
conv_2_num_filters	Architecture parameter	int	32	256
conv_3_filter_size	Architecture parameter	int	2	10
conv_3_num_filters	Architecture parameter	int	32	256
log_beta_1	Adam SGD parameter	real	-4.6	-0.7
log_beta_2	Adam SGD parameter	real	-13.8	-0.7
log_decay	Adam SGD parameter	real	-23	-2.3
log_epsilon	Adam SGD parameter	real	-23	-13.8
log_lr	Adam SGD parameter	real	-23	0

# Deep architecture deployment variations

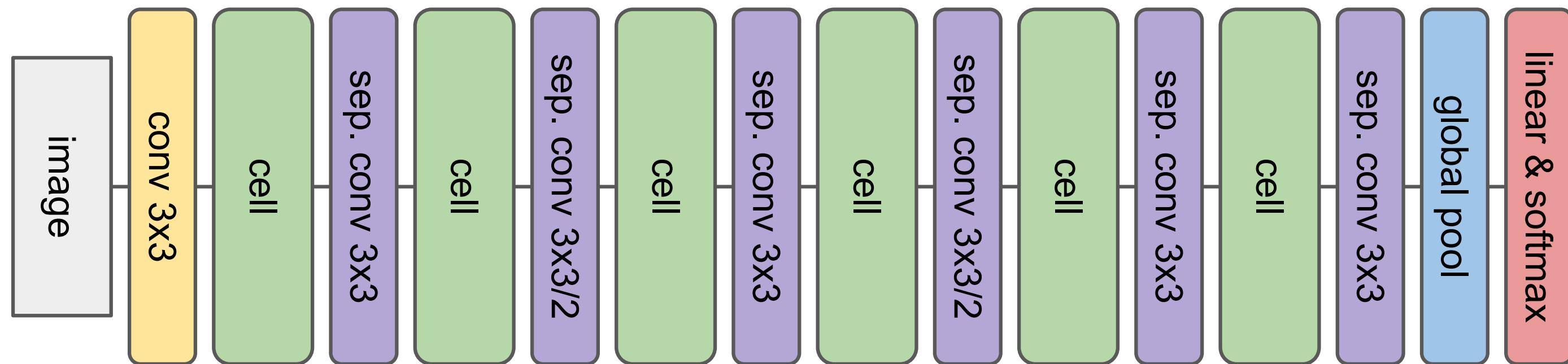


# Deep architecture hardware-level variations



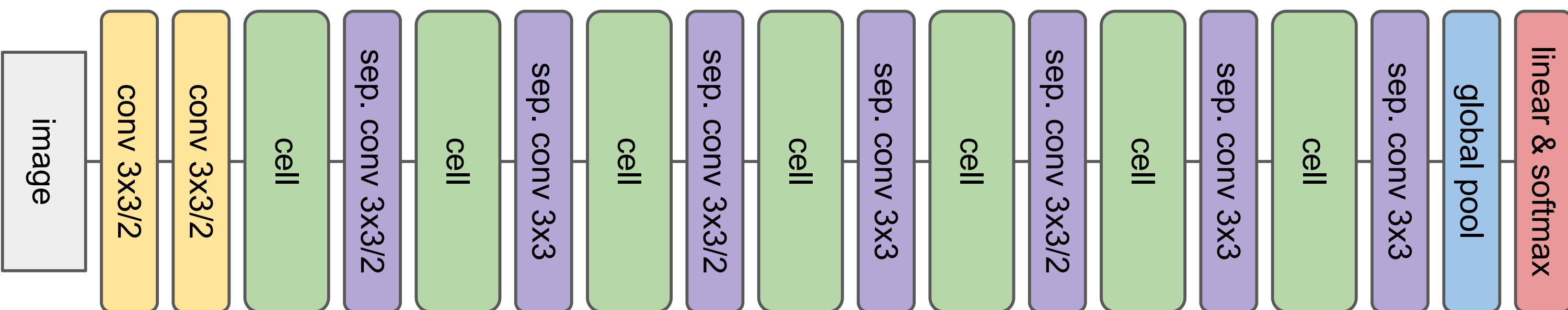
NVIDIA Xavier

# Exploring the design space of deep networks



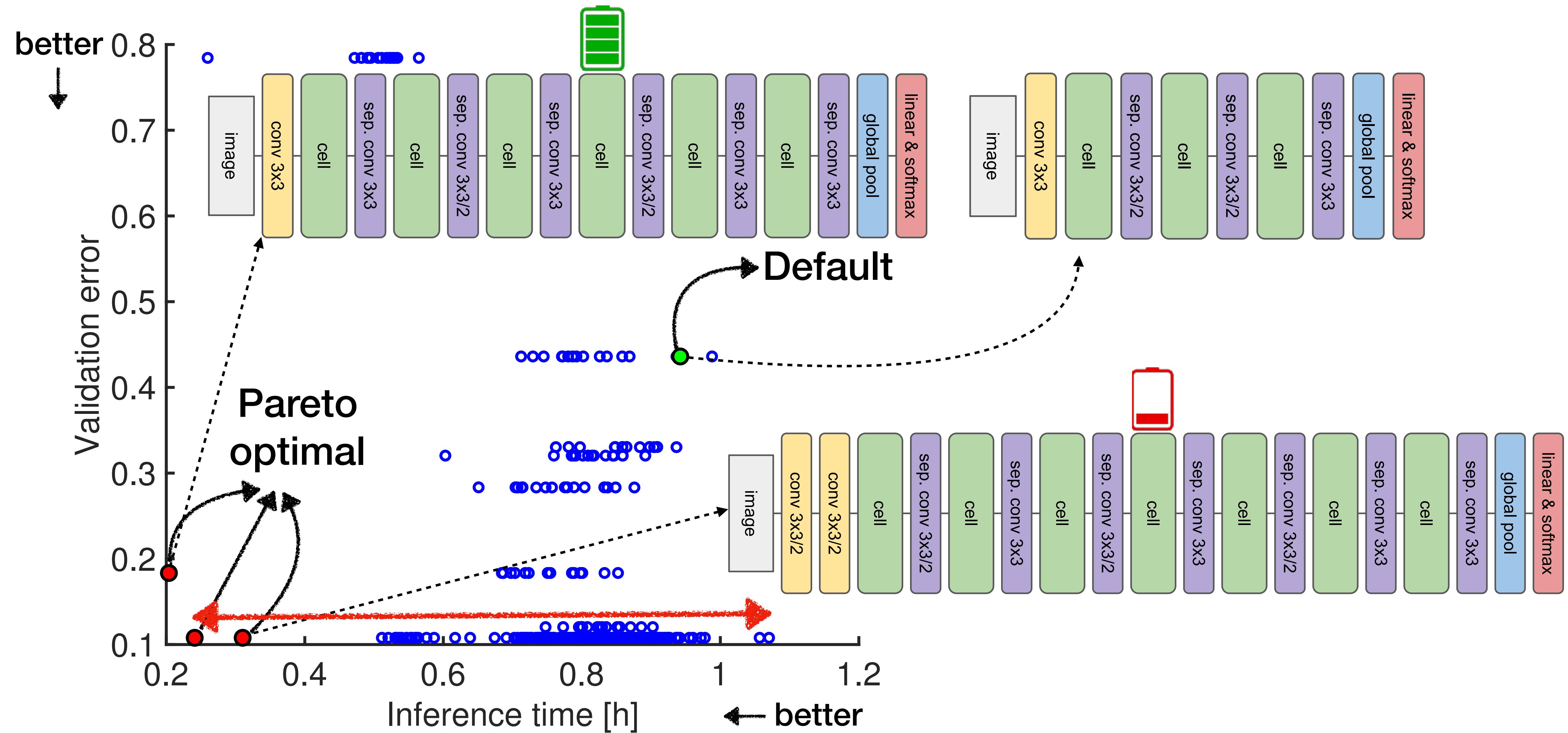
# Optimal Architecture (Yesterday)

# New Fraud Pattern

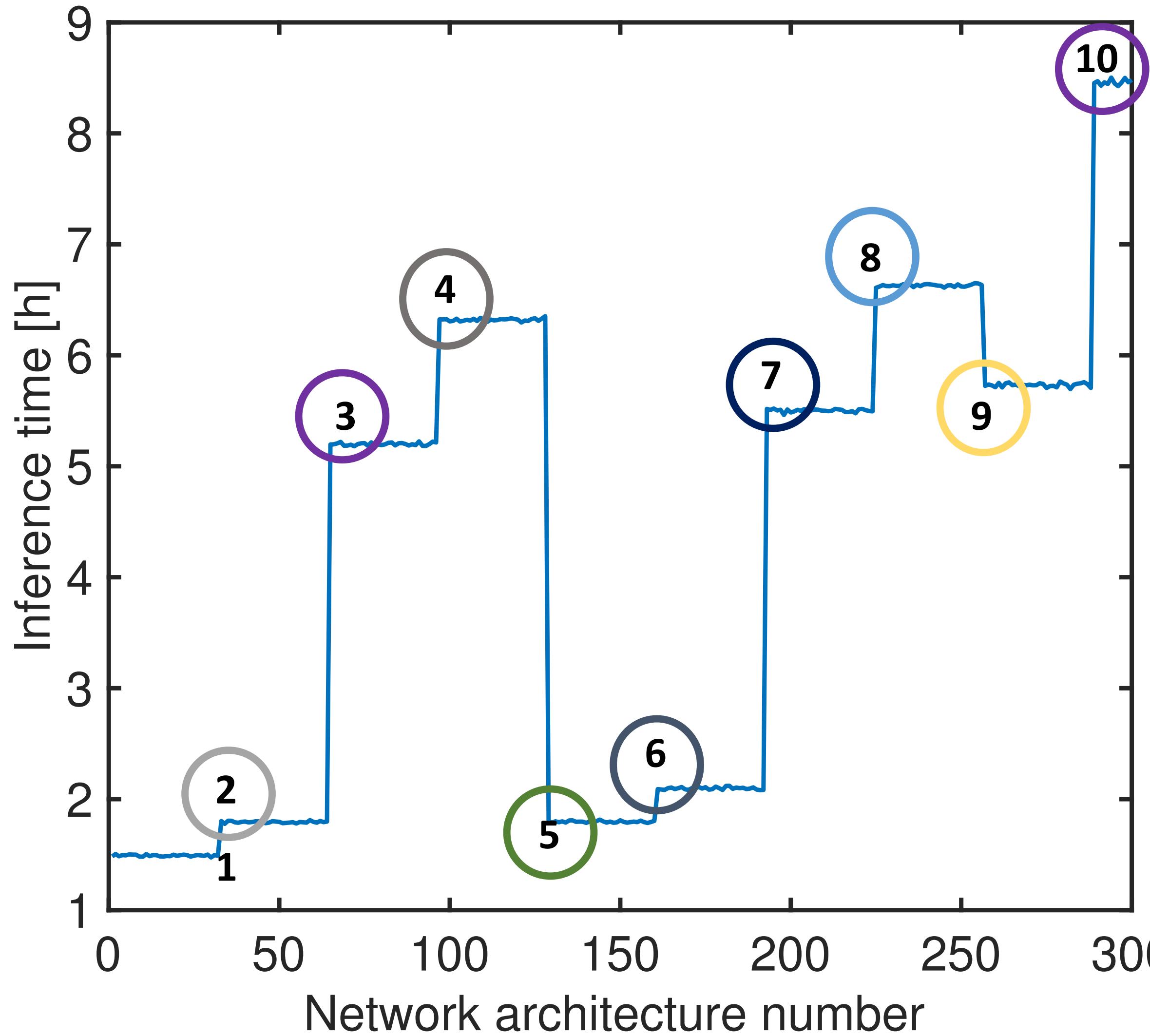


# Optimal Architecture (Today)

# Exploring the design space of deep networks

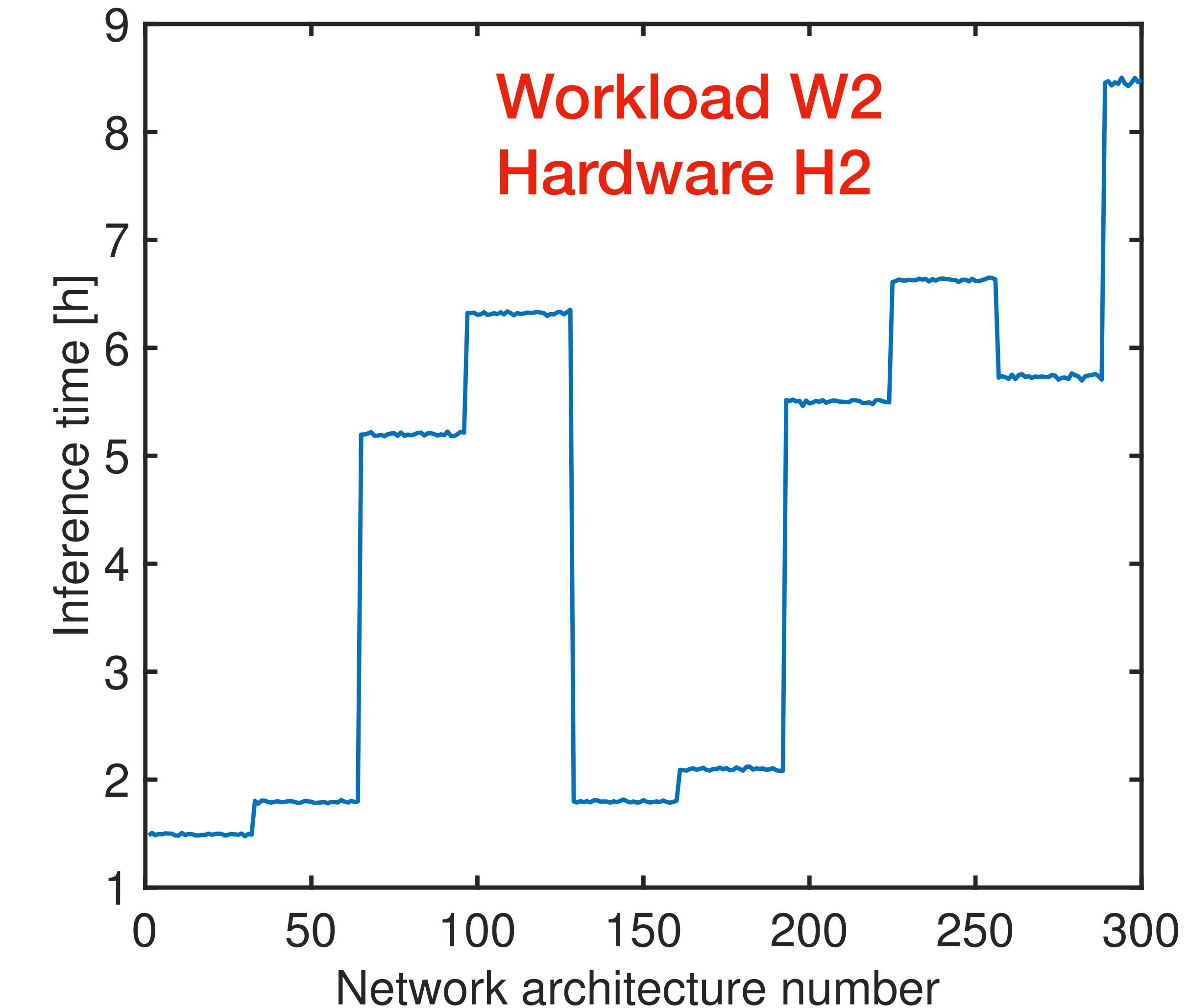
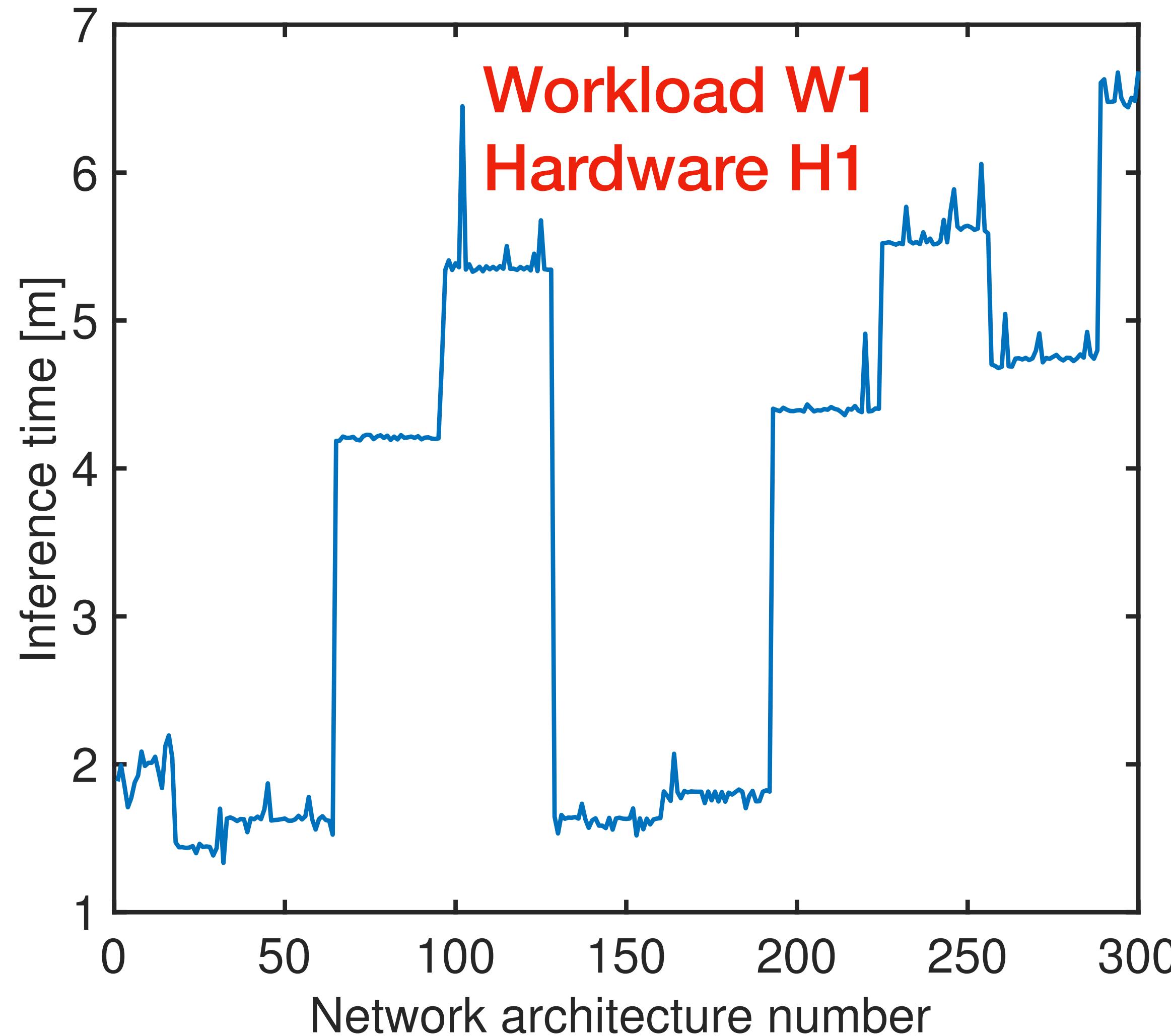


# Configuration options and interactions influence performance of DNNs

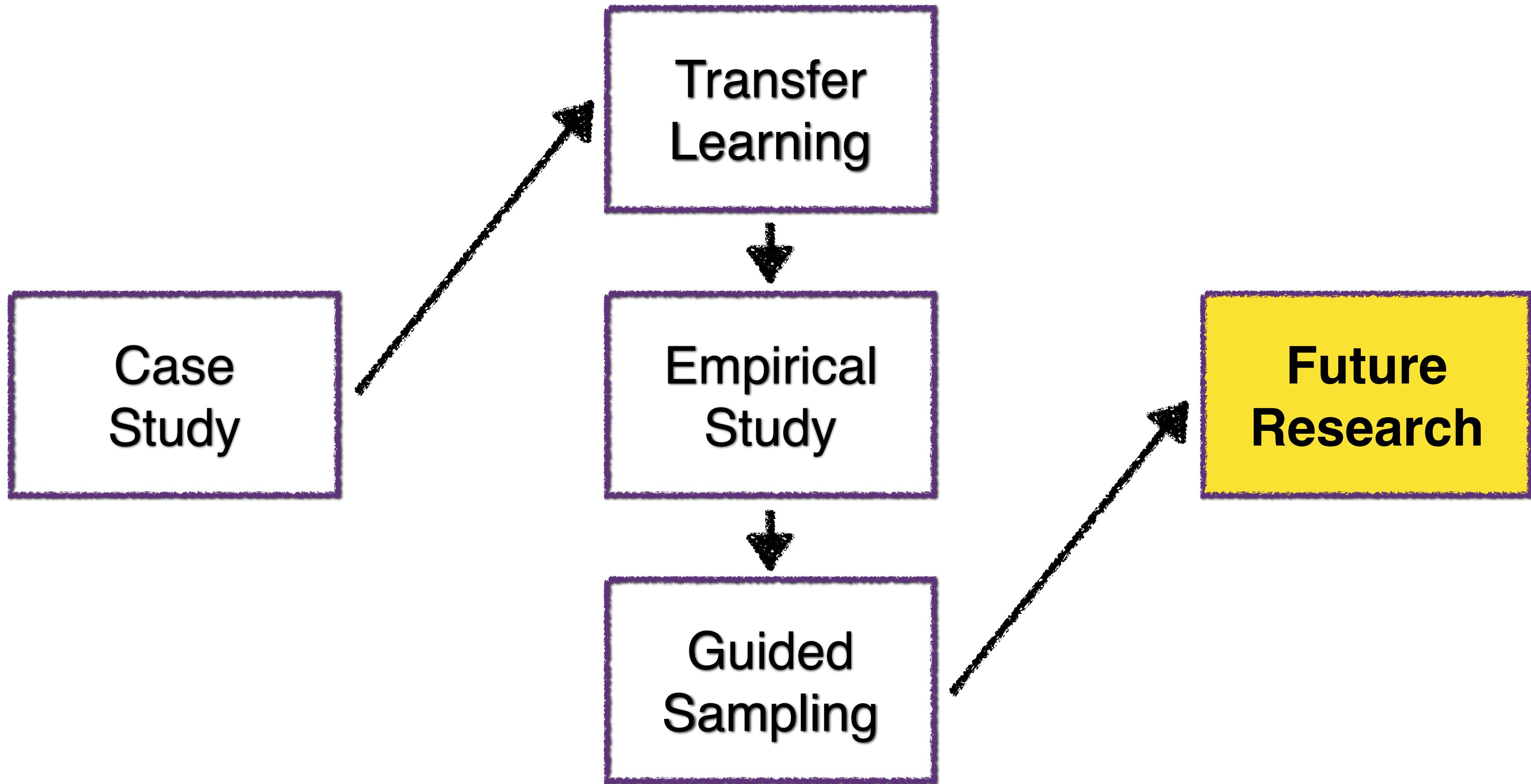


- 1- <0,0,0,0,0,0,**1,1,1,1,1**>
- 2- <0,0,0,0,0,**1**0,0,0,0>
- 3- <0,0,0,0,1,0,0,0,0,0>
- 4- <0,0,0,0,0**1,1**0,0,0,0>
- 5- <0,0,0,1,0,0,0,0,0,0>
- 6- <0,0,0,1,0,1,0,0,0,0>
- 7- <0,0,0,1,1,0,0,0,0,0>
- 8- <0,0,0,0,**1,1,1**0,0,0,0>
- 9- <0,0,0,1,0,0,0,0,0,0>
- 10- <0,0,0,1,0,0,1,0,0,0>

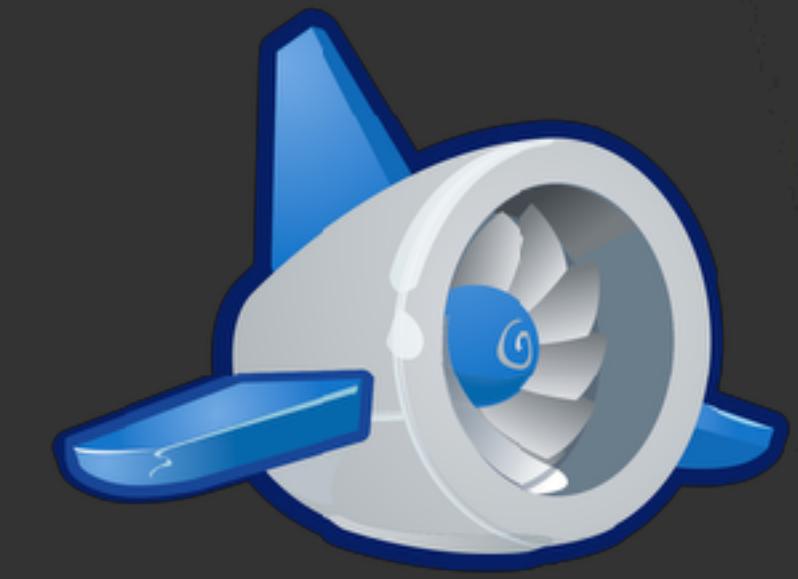
# Insight: Learn a model on a cheaper workload to explore the expensive workload faster



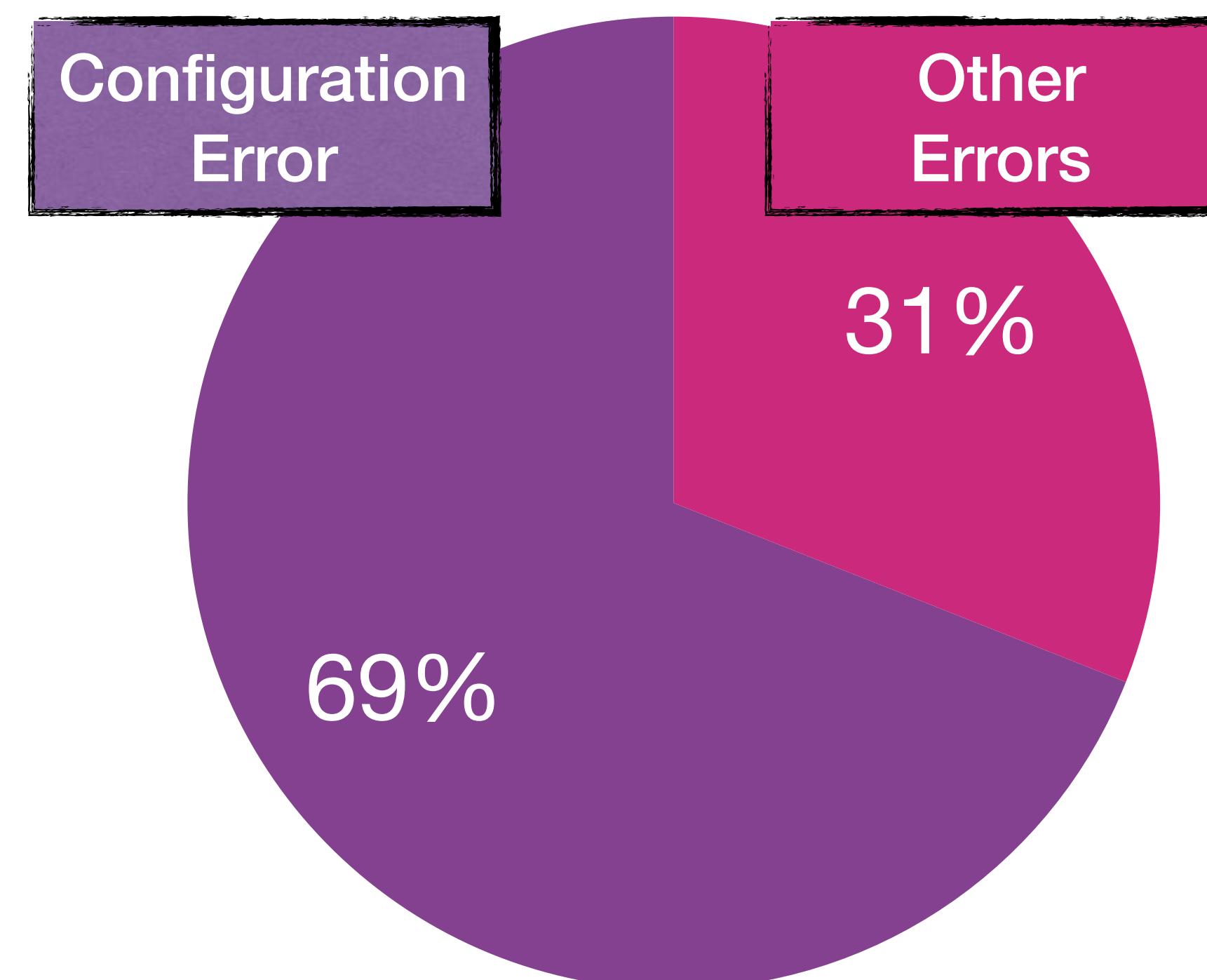
# Outline



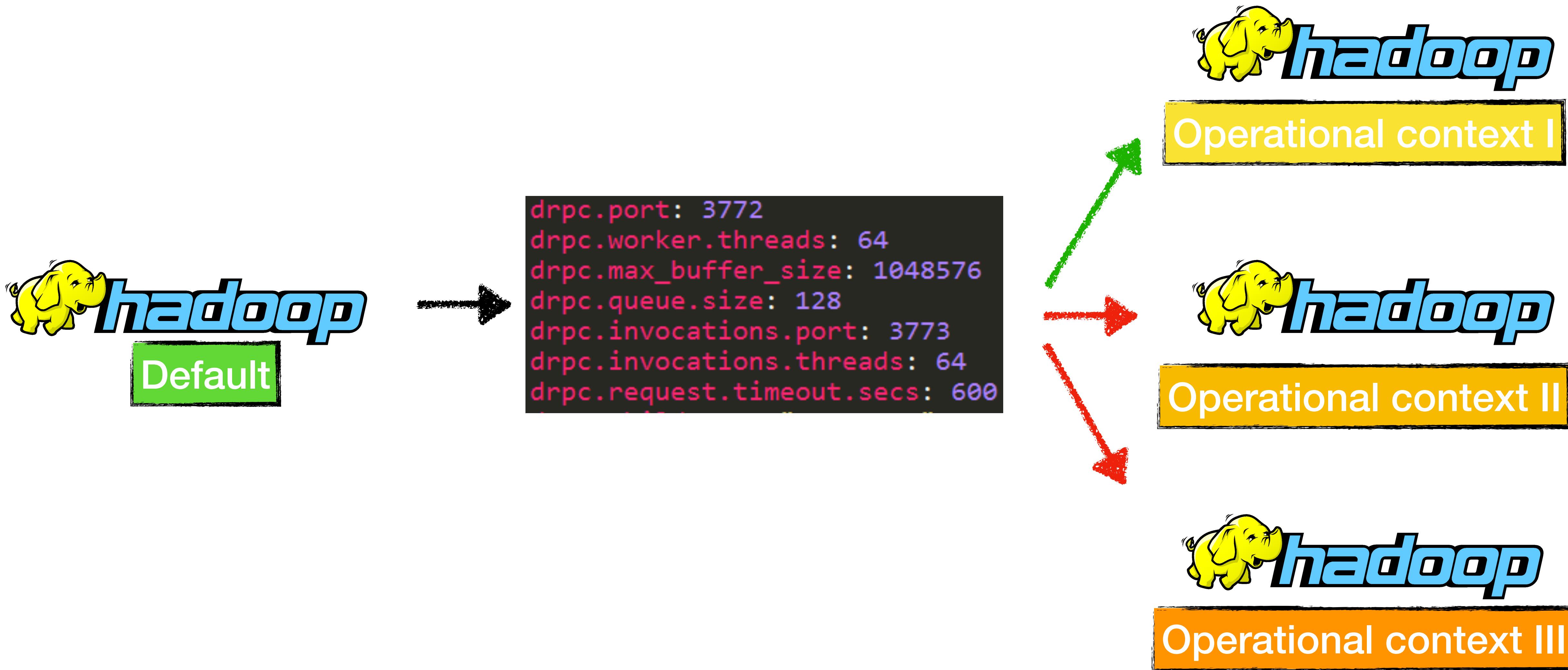
# Configuration errors are prevalent



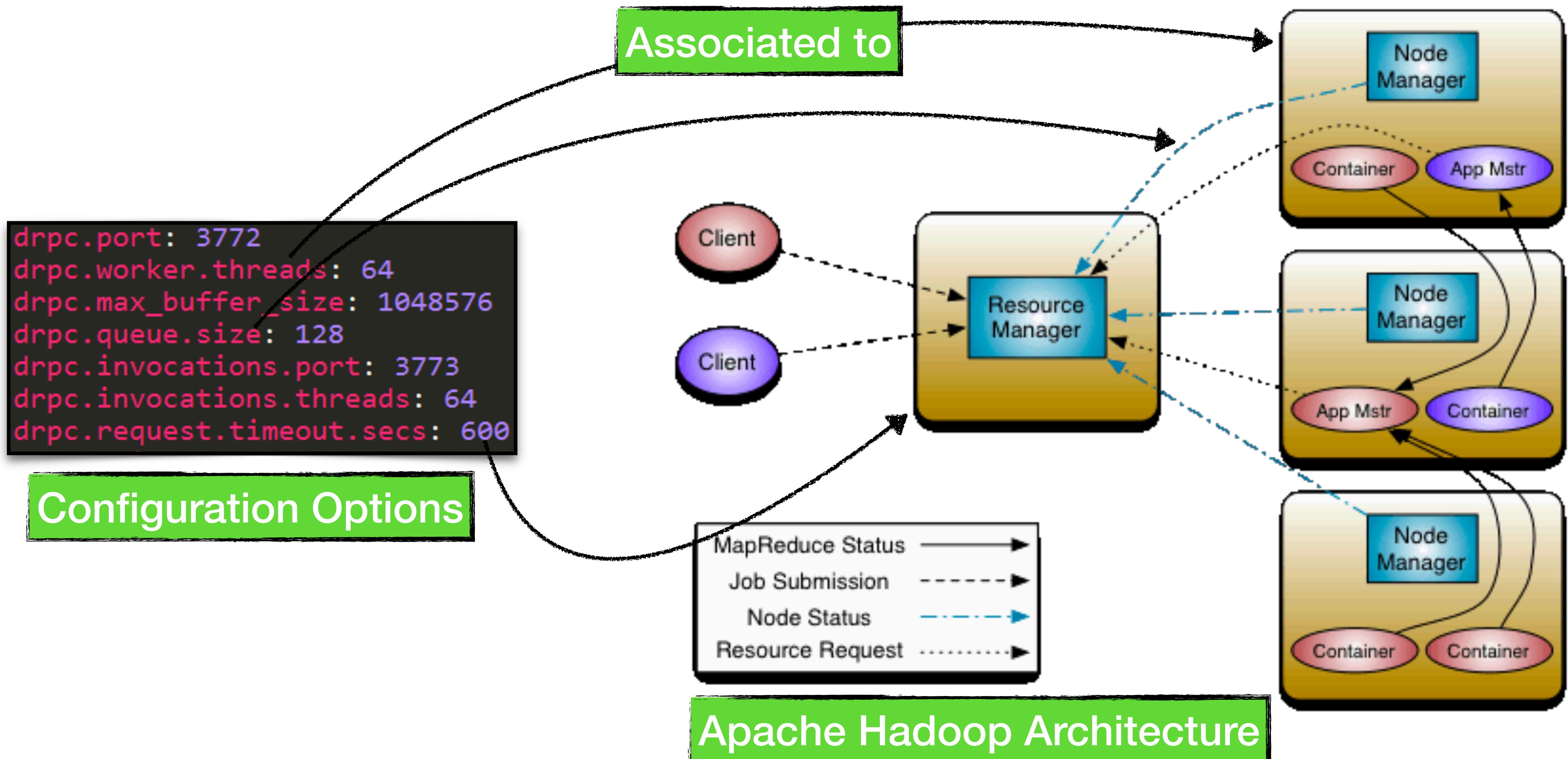
# Configuration errors are common



# Configuration complexity and dependencies between options is a major source of configuration errors



# Configuration complexity and dependencies between options is a major source of configuration errors



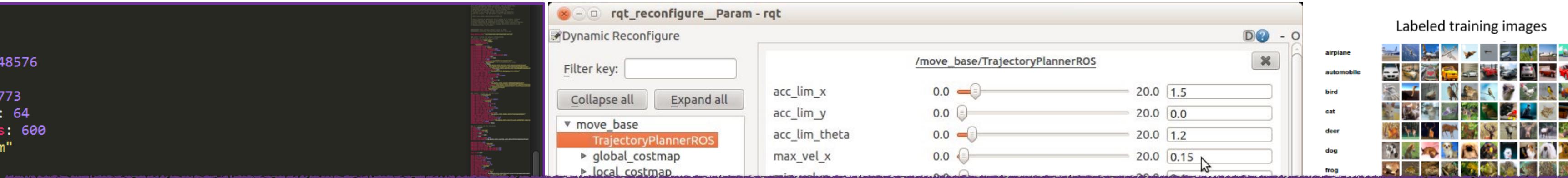
# **Configurations are software too**

# We can find the repair patches faster with a lighter workload

- **[localization]**: Using transfer learning to derive the **most likely configurations** that manifest the bugs
- **[repair]**: Automatically prepare **patches** to fix the configuration bug

# built Many systems are now configurable

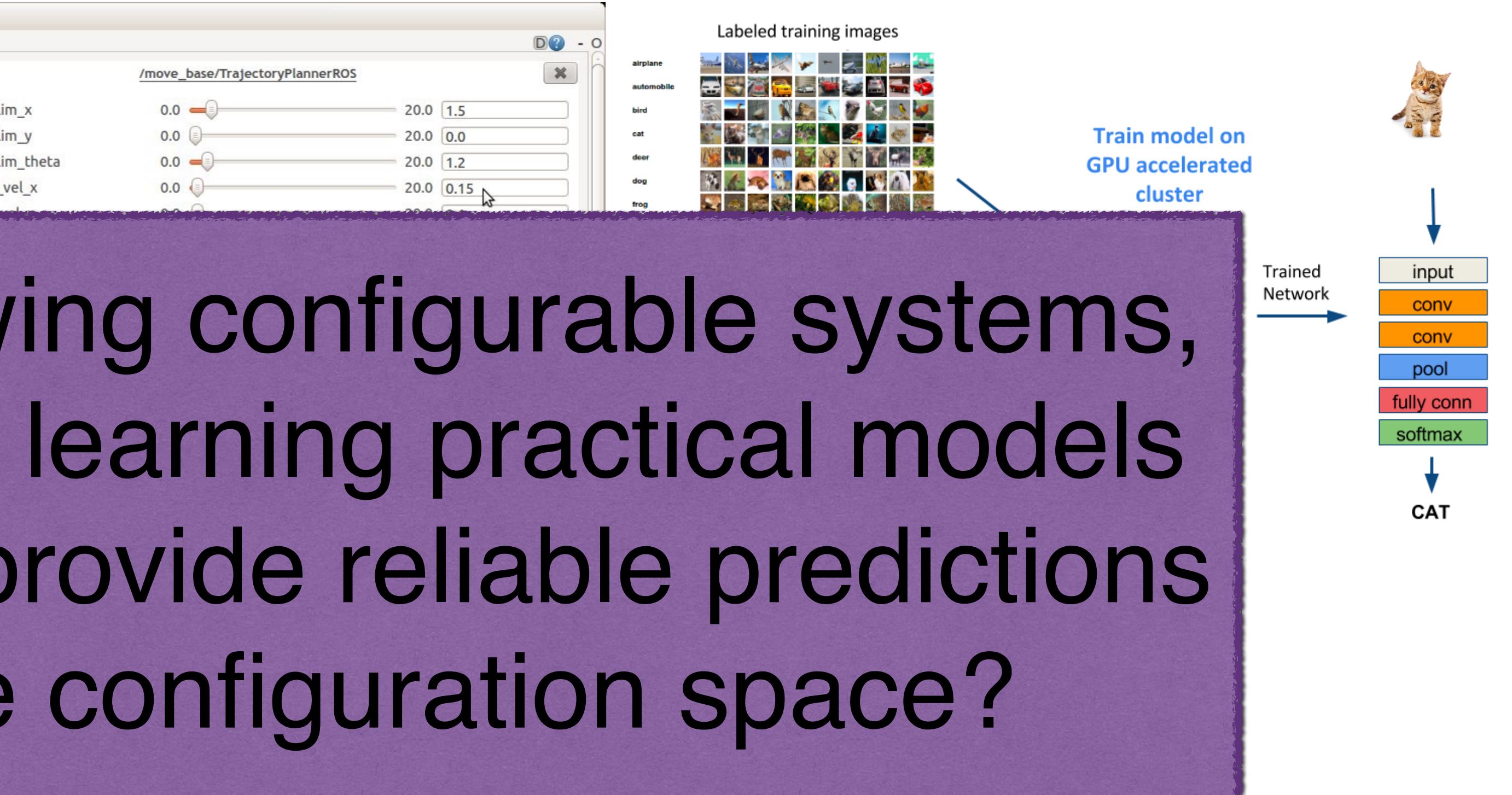
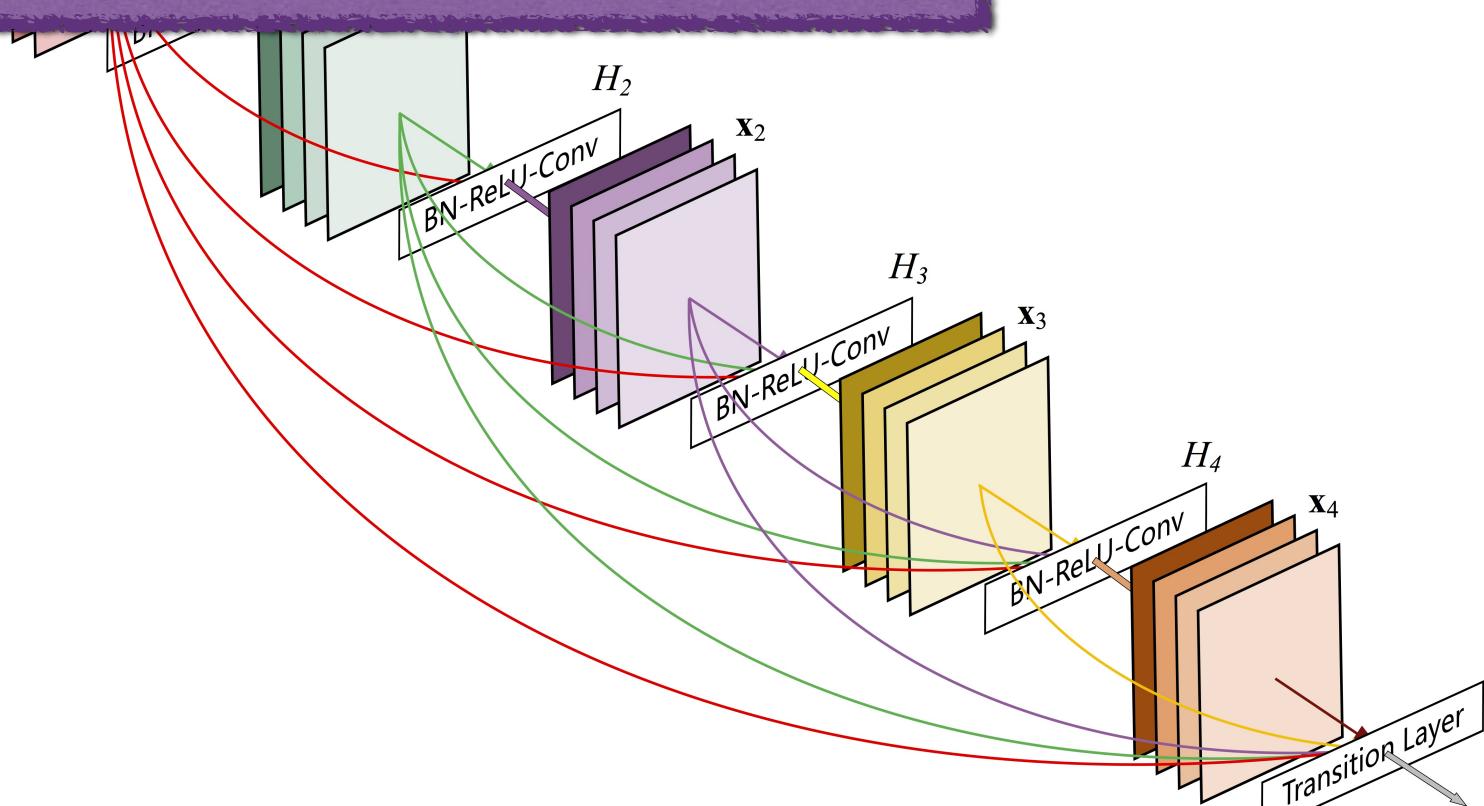
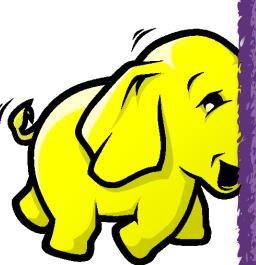
```
102  
103 drpc.port: 3772  
104 drpc.worker.threads: 64  
105 drpc.max_buffer_size: 1048576  
106 drpc.queue.size: 128  
107 drpc.invocations.port: 3773  
108 drpc.invocations.threads: 64  
109 drpc.request.timeout.secs: 600  
110 drpc.childopts: "-Xmx768m"  
111 drpc.http.port: 3774  
112 drpc.https.port: -1  
113 drpc.https.keystore.pass:  
114 drpc.https.keystore.type:  
115 drpc.http.creds.plugin:  
116 drpc.authorizer.acl.file:  
117 drpc.authorizer.acl.strict:  
118  
119 transactional.zookeeper.  
transactional.zookeeper.  
transactional.zookeeper.  
120  
121  
122  
123 ## blobstore configs  
supervisor.blobstore.class:  
supervisor.blobstore.download_dir:  
supervisor.blobstore.download_dir:  
supervisor.localizer.class:  
supervisor.localizer.clear:  
124  
125  
126  
127  
128  
129
```



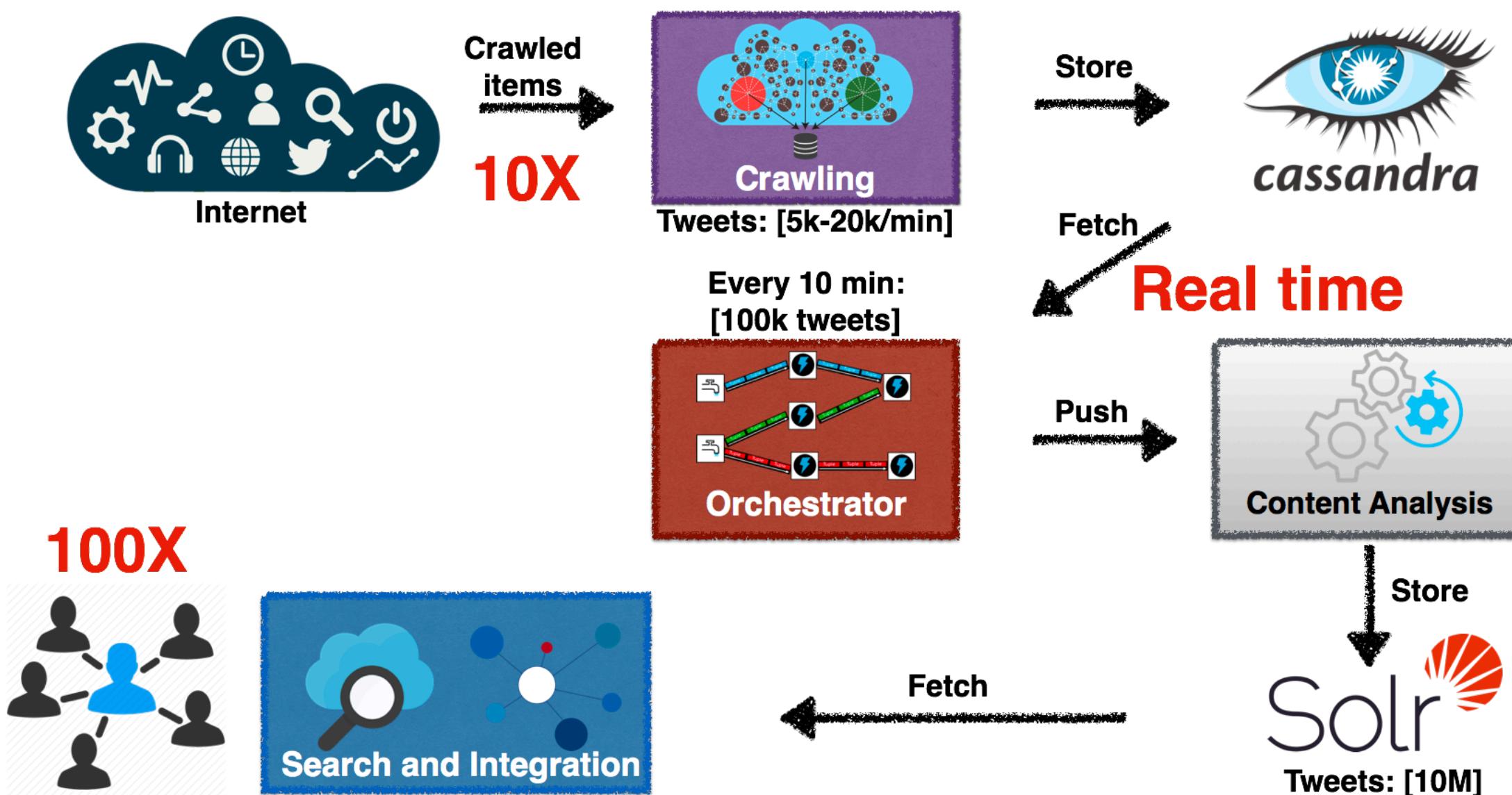
Train model on  
GPU accelerated  
cluster



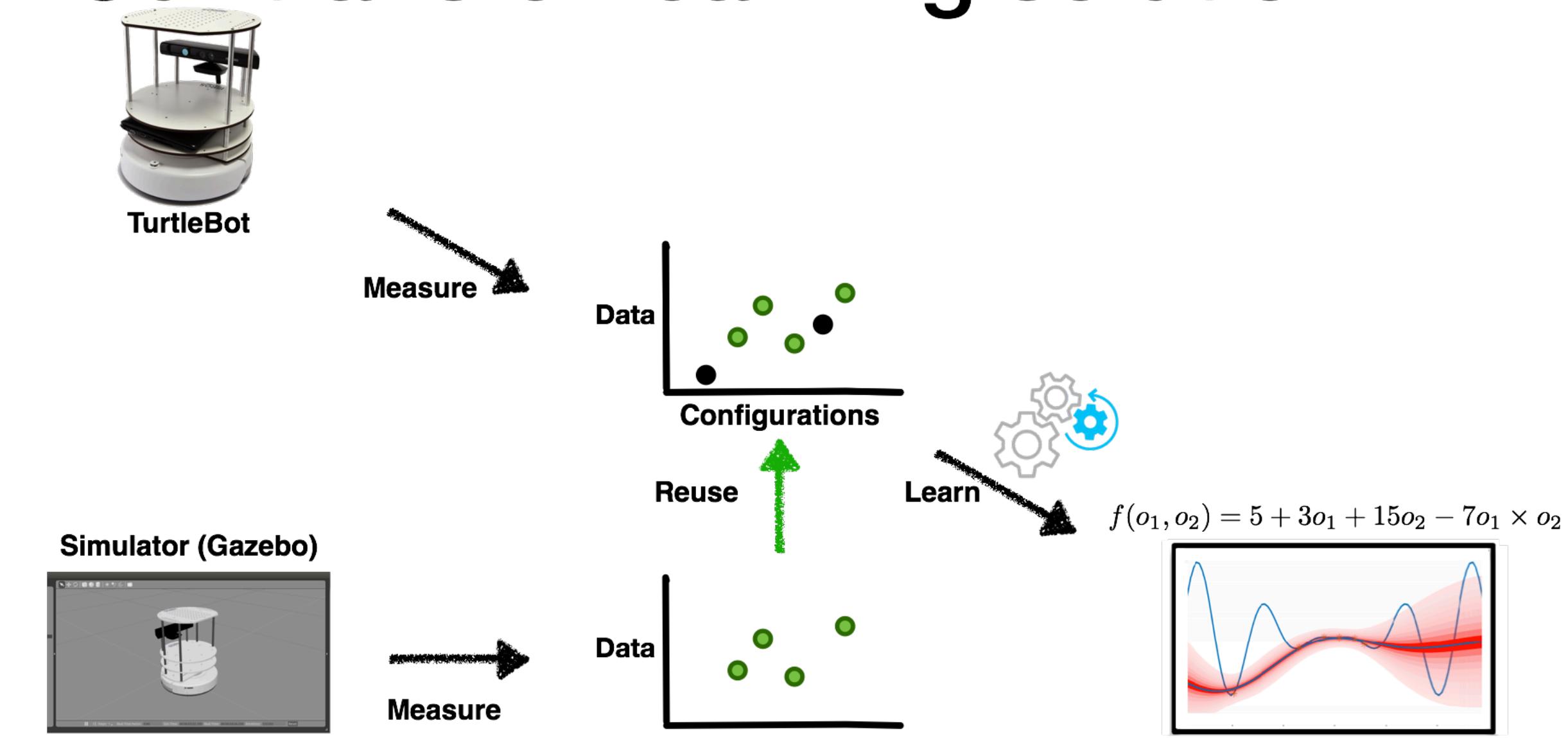
Given the ever growing configurable systems,  
how can we enable learning practical models  
that scale well and provide reliable predictions  
for exploring the configuration space?



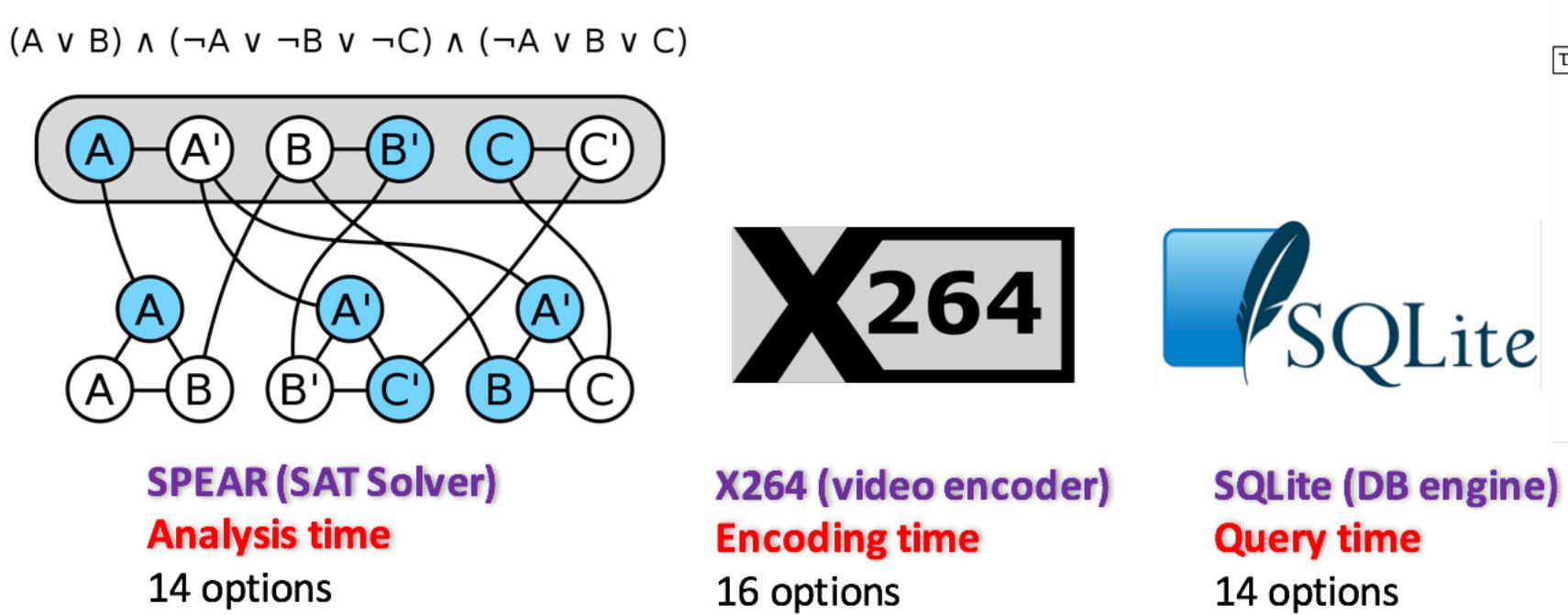
# Challenges



# Our transfer learning solution



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



Exploring the design space of deep networks

