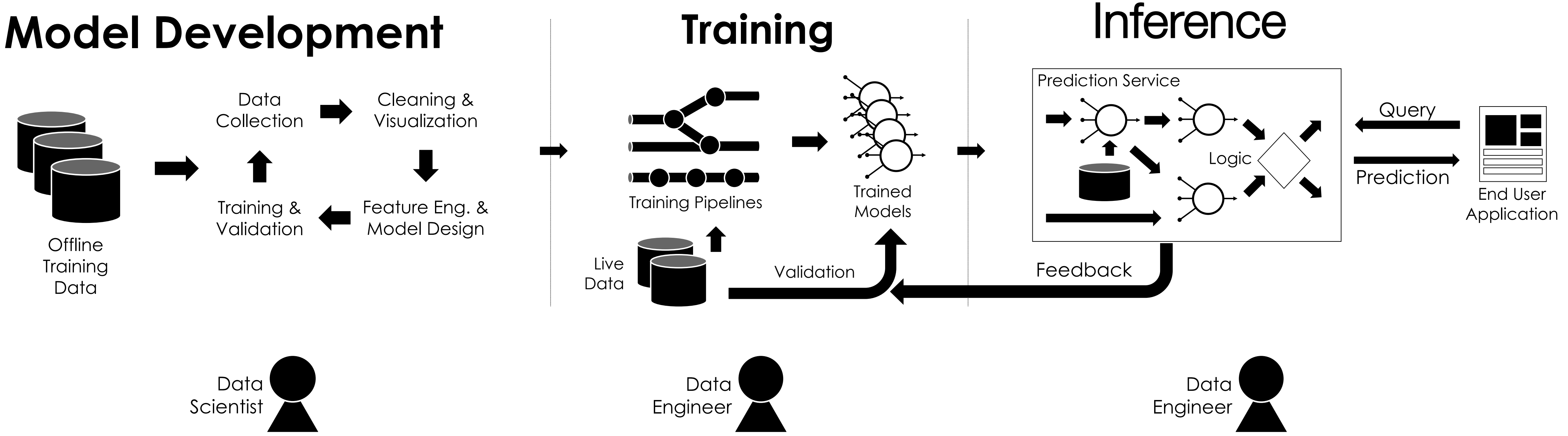
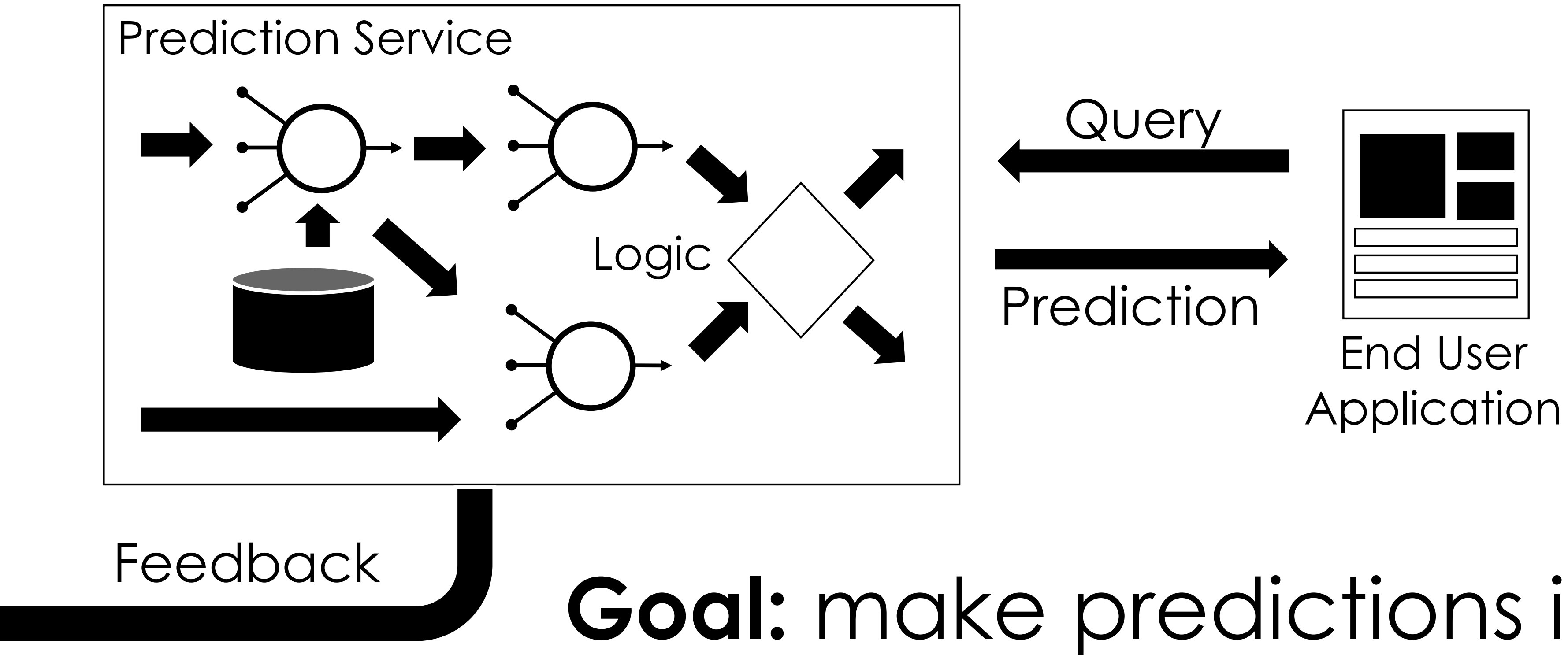
AI-Systems
Prediction Serving

Joseph E. Gonzalez
Co-director of the RISE Lab
jegonzal@cs.berkeley.edu

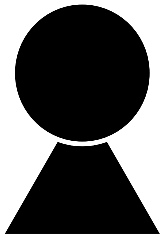
Machine Learning Lifecycle



Inference



Goal: make predictions in ~10ms under **bursty** load

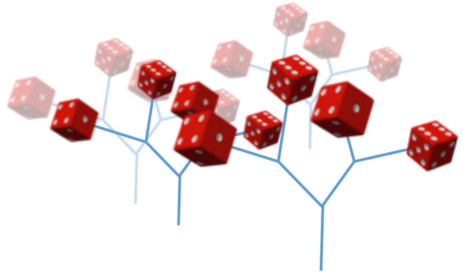


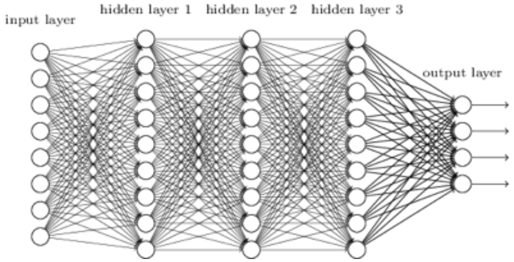
Complicated by **Deep Neural Networks**
➔ New **ML Algorithms** and **Systems**

why is **Inference** challenging?

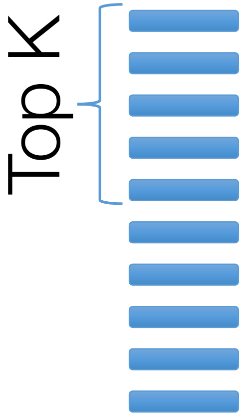
Need to render **low latency** (< 10ms) predictions for **complex**

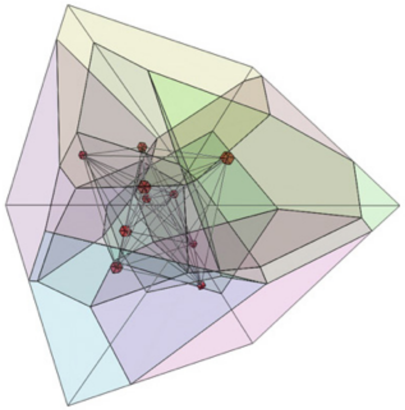
Models





Queries





Features

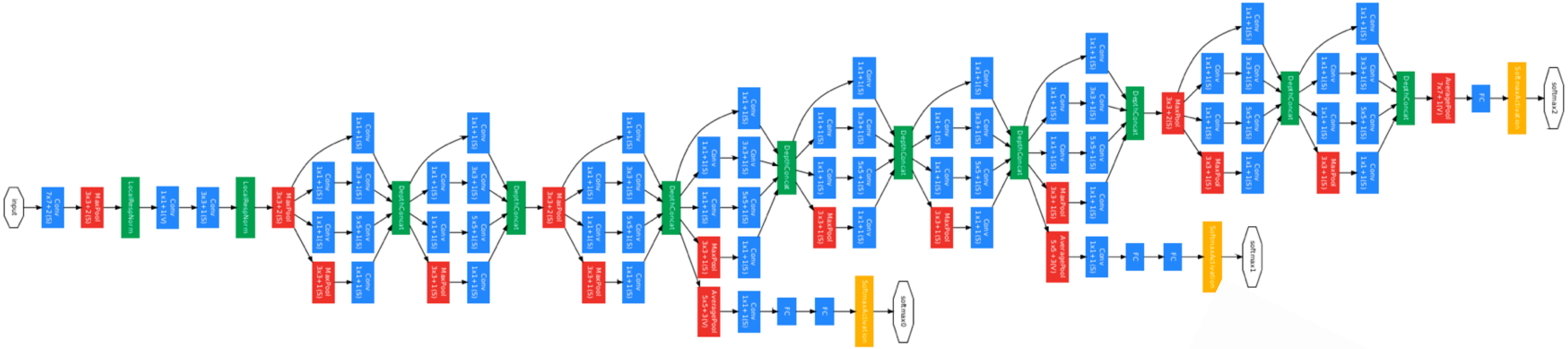
SELECT * FROM
users JOIN items,
click_logs, pages
WHERE …

under **heavy load** with system **failures**.

Basic Linear Models (Often High Dimensional)

- Common for **click prediction** and **text filter models** (spam)
- Query x encoded in sparse Bag-of-Words:
  - x = “The quick brown” = {(”brown”, 1), (”the”, 1), (“quick”, 1)}
- Rendering a prediction: $$\mathbf{Predict}(x) = \sigma\left(\sum_{(w,c)\in x} \theta_w c\right)$$
- θ is a large vector of weights for each possible word
  - or word combination (n-gram models) …
- Optimizations?

Support low-latency, high-throughput serving workloads



Models getting more complex

- 10s of GFLOPs [1]

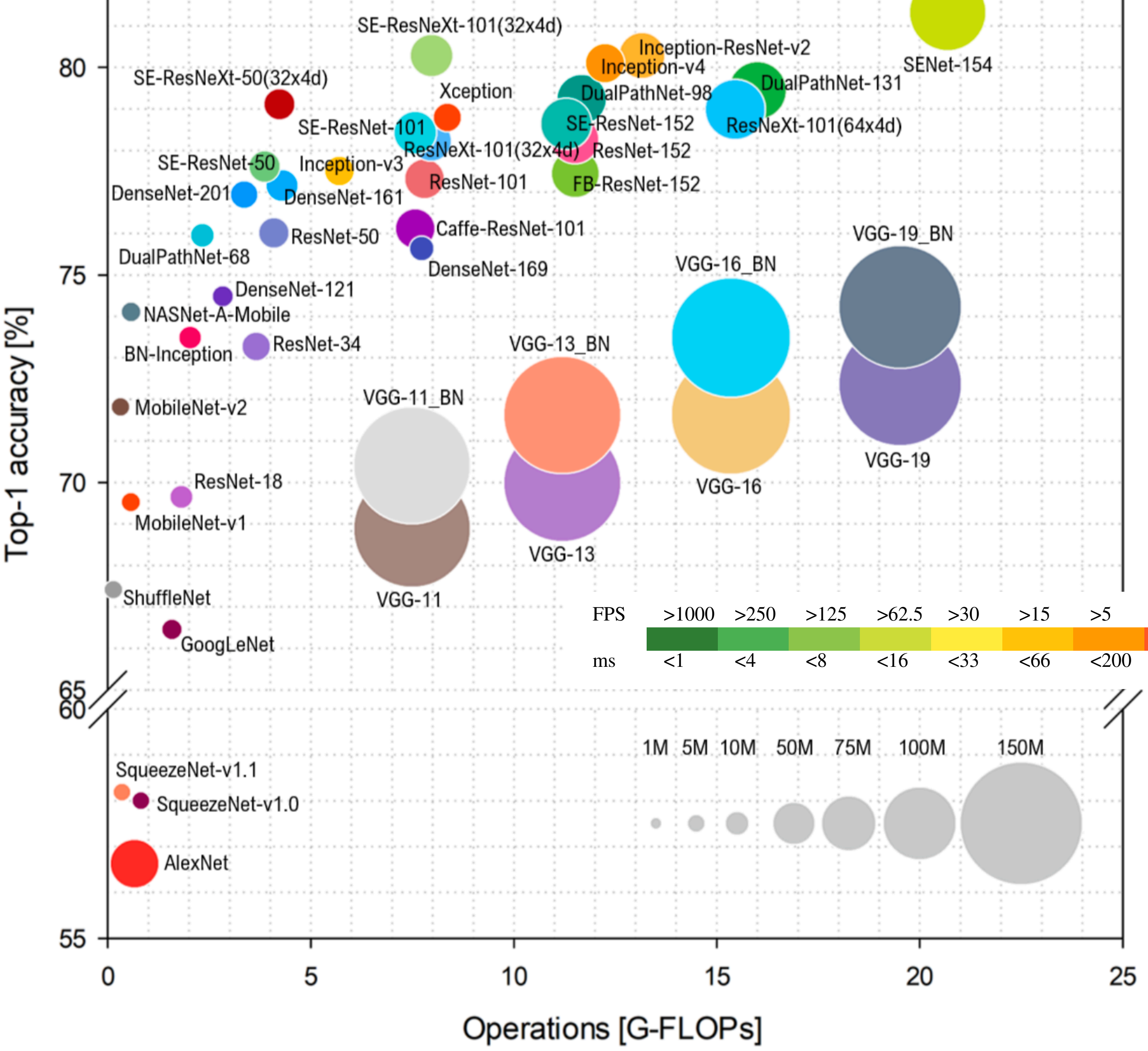
Deployed on critical path

- Maintain SLOs under heavy load

Using specialized hardware for predictions

[1] Deep Residual Learning for Image Recognition. He et al. CVPR 2015.

[Benchmark Analysis of Representative Deep Neural Network Architectures]



FPS	>1000	>250	>125	>62.5	>30	>15	>5	<=5
ms	<1	<4	<8	<16	<33	<66	<200	>=200

TitanXP

DNN \ Batch Size	1	2	4	8	16	32	64
AlexNet	1.28	0.70	0.48	0.27	0.18	0.14	0.15
BN-Inception	5.79	3.00	1.64	1.10	0.87	0.77	0.71
CaffeResNet-101	8.20	4.82	3.32	2.54	2.27	2.16	2.08
DenseNet-121 (k=32)	8.93	4.41	2.64	1.96	1.64	1.44	1.39
DenseNet-169 (k=32)	13.03	6.72	3.97	2.73	2.14	1.87	1.75
DenseNet-201 (k=32)	17.15	9.25	5.36	3.66	2.84	2.41	2.27
DenseNet-161 (k=48)	15.50	9.10	5.89	4.45	3.66	3.43	3.24
DPN-68	10.68	5.36	3.24	2.47	1.80	1.59	1.52
DPN-98	22.31	13.84	8.97	6.77	5.59	4.96	4.72
DPN-131	29.70	18.29	11.96	9.12	7.57	6.72	6.37
FBResNet-152	14.55	7.79	5.15	4.31	3.96	3.76	3.65
GoogLeNet	4.54	2.44	1.65	1.06	0.86	0.76	0.72
Inception-ResNet-v2	25.94	14.36	8.82	6.43	5.19	4.88	4.59
Inception-v3	10.10	5.70	3.65	2.54	2.05	1.89	1.80
Inception-v4	18.96	10.61	6.53	4.85	4.10	3.77	3.61
MobileNet-v1	2.45	0.89	0.68	0.60	0.55	0.53	0.53
MobileNet-v2	3.34	1.63	0.95	0.78	0.72	0.63	0.61
NASNet-A-Large	32.30	23.00	19.75	18.49	18.11	17.73	17.77
NASNet-A-Mobile	22.36	11.44	5.60	2.81	1.61	1.75	1.51
ResNet-101	8.90	5.16	3.32	2.69	2.42	2.29	2.21
ResNet-152	14.31	7.36	4.68	3.83	3.50	3.30	3.17
ResNet-18	1.79	1.01	0.70	0.56	0.51	0.41	0.38
ResNet-34	3.11	1.80	1.20	0.96	0.82	0.71	0.67
ResNet-50	5.10	2.87	1.99	1.65	1.49	1.37	1.34
ResNeXt-101 (32x4d)	17.05	9.02	6.27	4.62	3.71	3.25	3.11
ResNeXt-101 (64x4d)	21.05	15.54	10.39	7.80	6.39	5.62	5.29
SE-ResNet-101	15.10	9.26	6.17	4.72	4.03	3.62	3.42
SE-ResNet-152	23.43	13.08	8.74	6.55	5.51	5.06	4.85
SE-ResNet-50	8.32	5.16	3.36	2.62	2.22	2.01	2.06
SE-ResNeXt-101 (32x4d)	24.96	13.86	9.16	6.55	5.29	4.53	4.29
SE-ResNeXt-50 (32x4d)	12.06	7.41	5.12	3.64	2.97	3.01	2.56
SENet-154	53.80	30.30	19.32	13.27	10.45	9.41	8.91
ShuffleNet	5.40	2.67	1.37	0.82	0.66	0.59	0.56
SqueezeNet-v1.0	1.53	0.84	0.66	0.59	0.54	0.52	0.53
SqueezeNet-v1.1	1.60	0.77	0.44	0.37	0.32	0.31	0.30
VGG-11	3.57	4.40	2.89	1.56	1.19	1.10	1.13
VGG-11_BN	3.49	4.60	2.99	1.71	1.33	1.24	1.27
VGG-13	3.88	5.03	3.44	2.25	1.83	1.75	1.79
VGG-13_BN	4.40	5.37	3.71	2.42	2.05	1.97	2.00
VGG-16	5.17	5.91	4.01	2.84	2.20	2.12	2.15
VGG-16_BN	5.04	5.95	4.27	3.06	2.45	2.36	2.41
VGG-19	5.50	6.26	4.71	3.29	2.59	2.52	2.50
VGG-19_BN	6.17	6.67	4.86	3.56	2.88	2.74	2.76
Xception	6.44	5.35	4.90	4.47	4.41	4.41	4.36

[Benchmark Analysis of Representative Deep Neural Network Architectures]

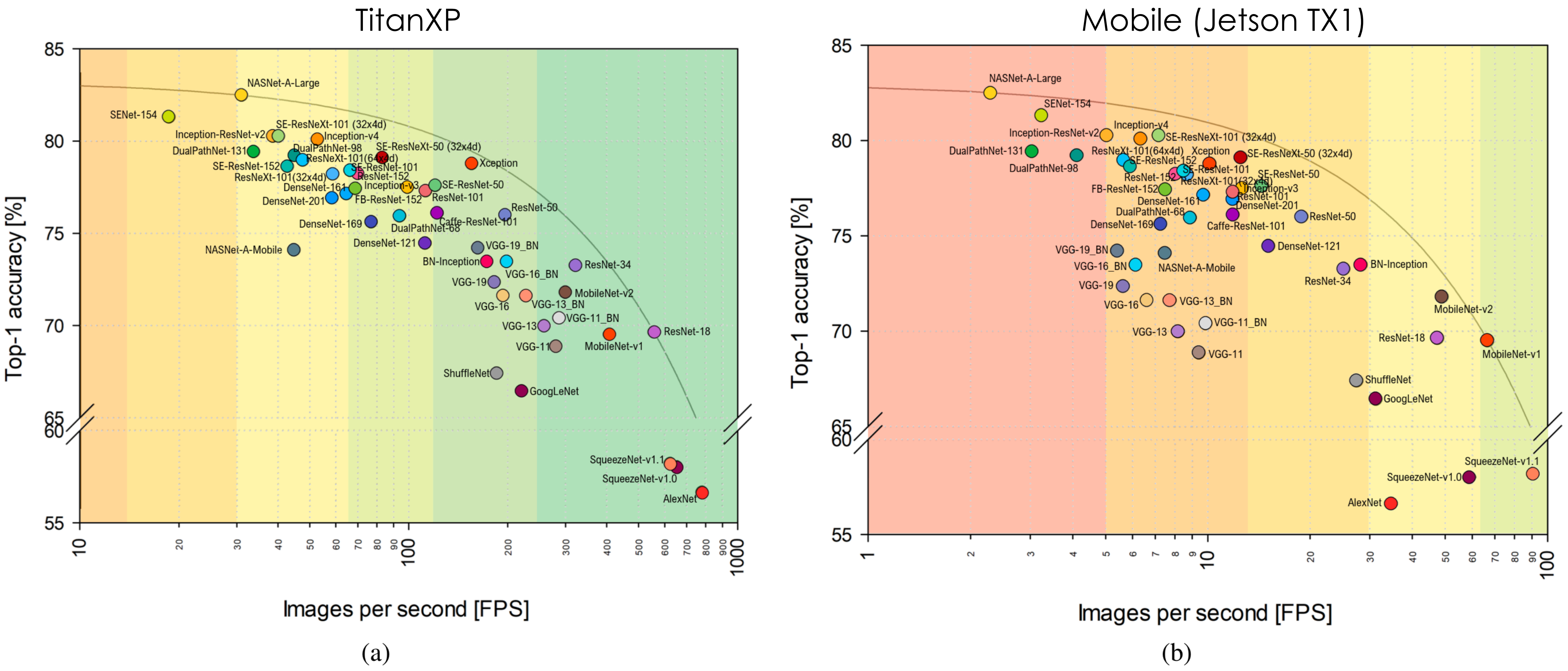
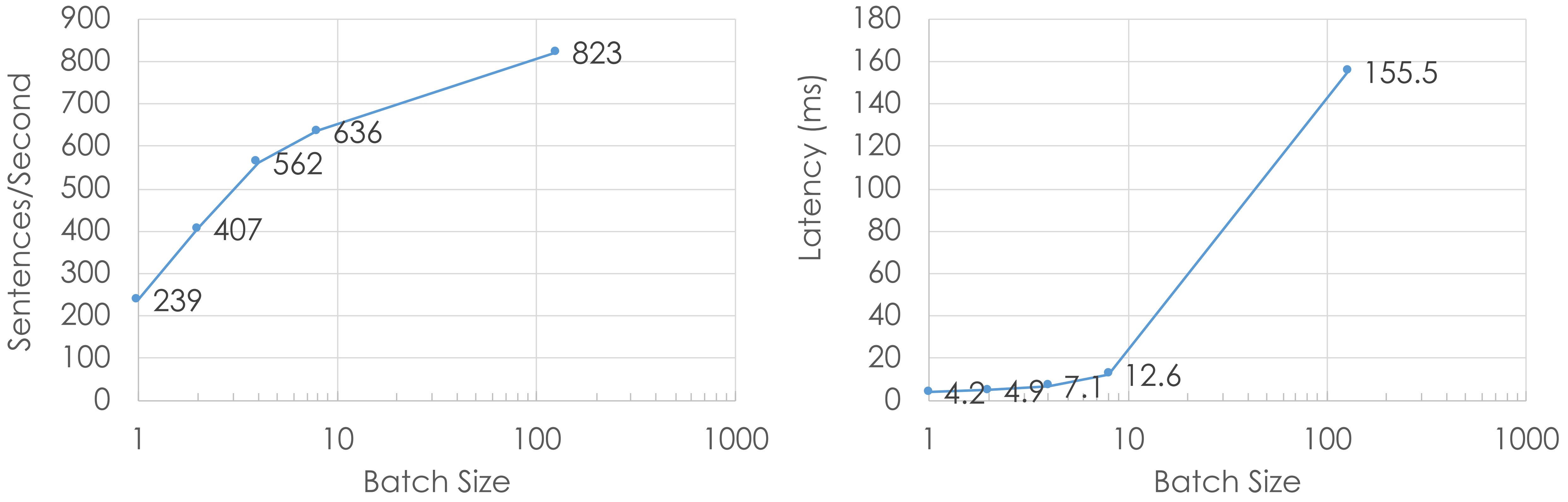


FIGURE 3: Top-1 accuracy *vs.* number of images processed per second (with batch size 1) using the Titan Xp (a) and Jetson TX1 (b).

BERT-Large on a V100 (~$10K)



Results included Mixed precision optimizations!

Numbers obtained from: https://developer.nvidia.com/deep-learning-performance-training-inference

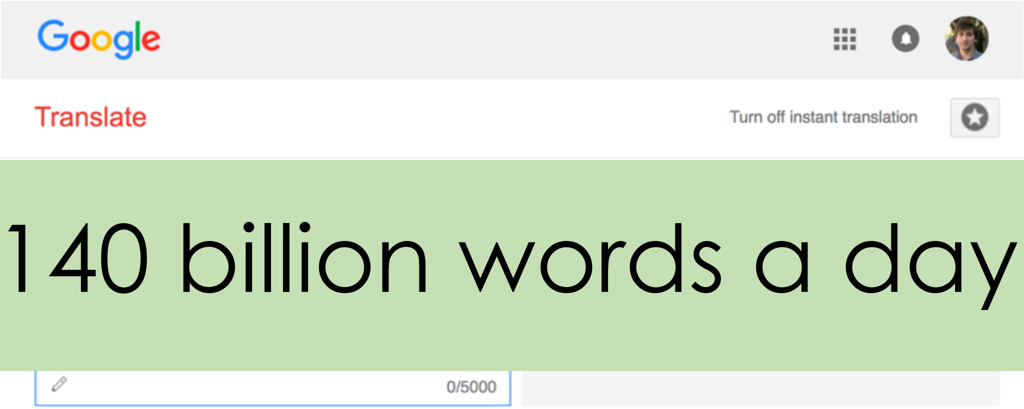
Google Translate

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi
yonghui,schuster,zhifengc,qvl,mnorouzi@google.com

Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean

Serving



140 billion words a day[1]

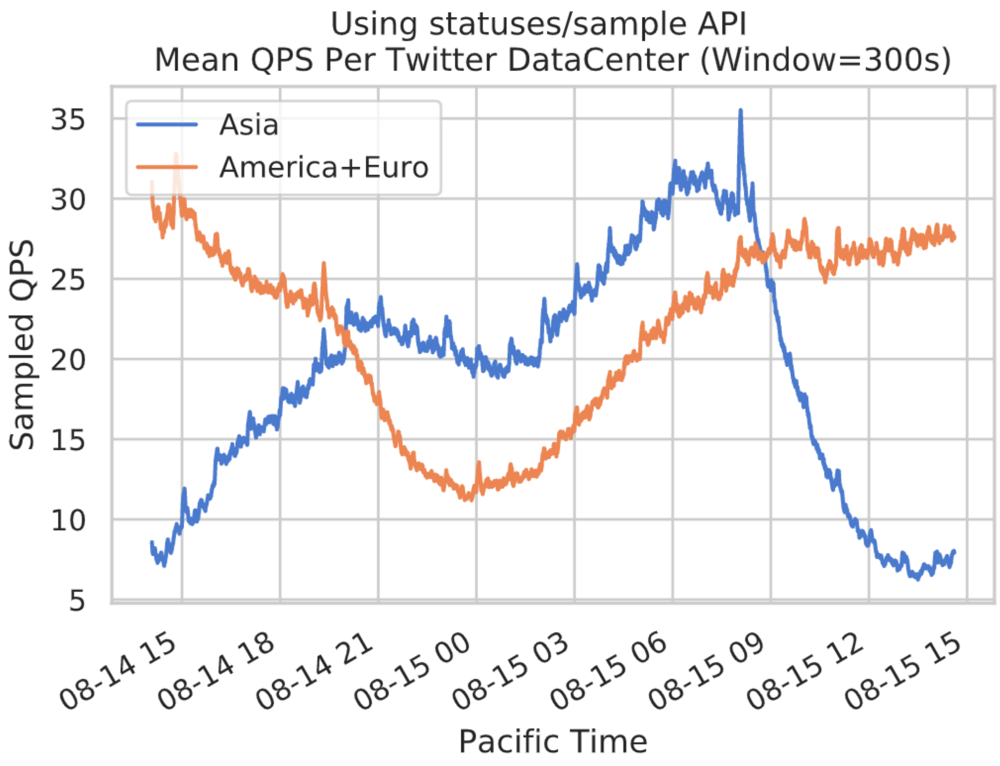
82,000 GPUs
running 24/7

"If each of the ***world's Android phones*** *used the new Google voice search for just* ***three minutes a day****, these engineers realized, the company would* ***need twice as many data centers.****"*
– Wired

Designed New Hardware!
Tensor Processing Unit (TPU)

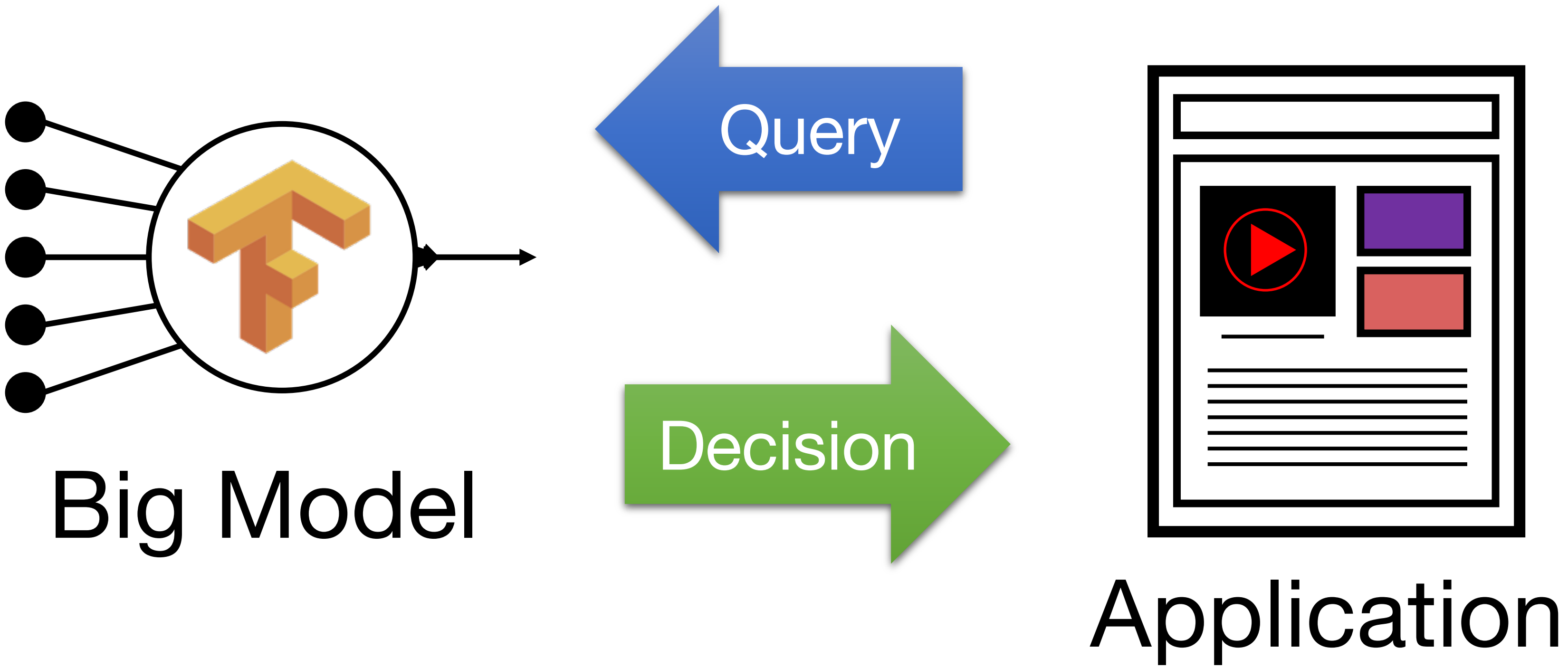
[1] https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html

Other Challenges?



- **Bursty load →**
  - overprovision resources →
    - **expensive**
  - TPU reports 28% utilization of vector units in production
  - Solutions
    - statistical multiplexing → difficult → why?
    - could try to predict arrival process → difficult (impossible?)!
- **Versioning** and **testing models**
- **Prediction pipelines →** more on this soon

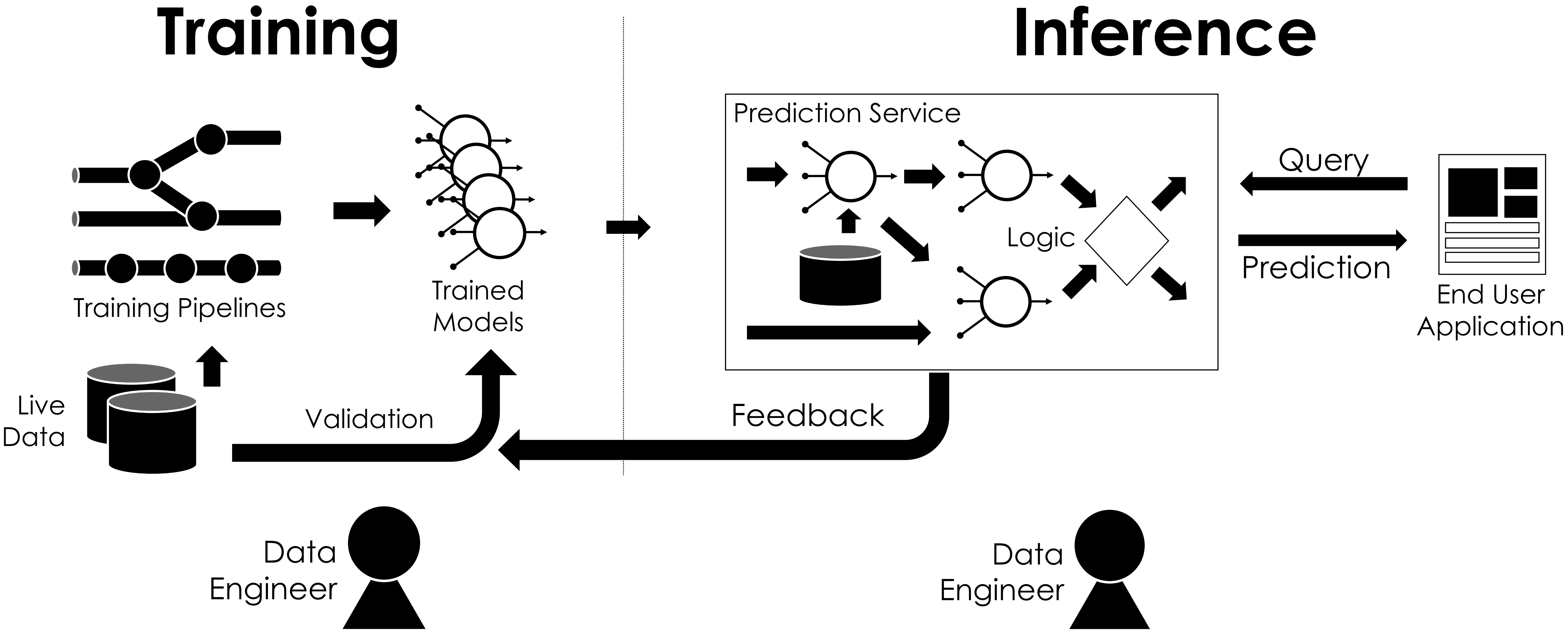
Inference



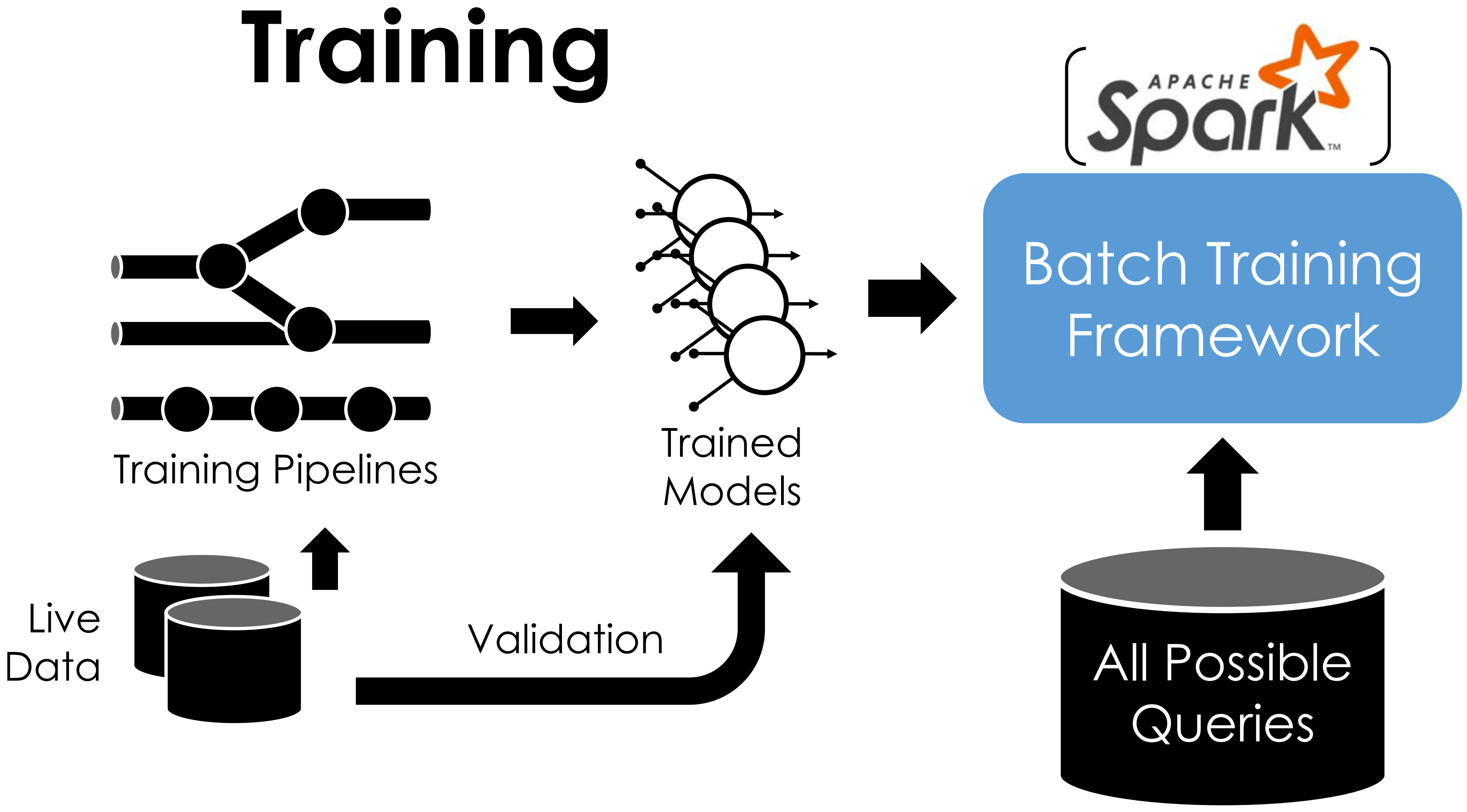
Two Approaches

- **_Offline:_** Pre-Materialize Predictions
- **_Online:_** Compute Predictions on the fly

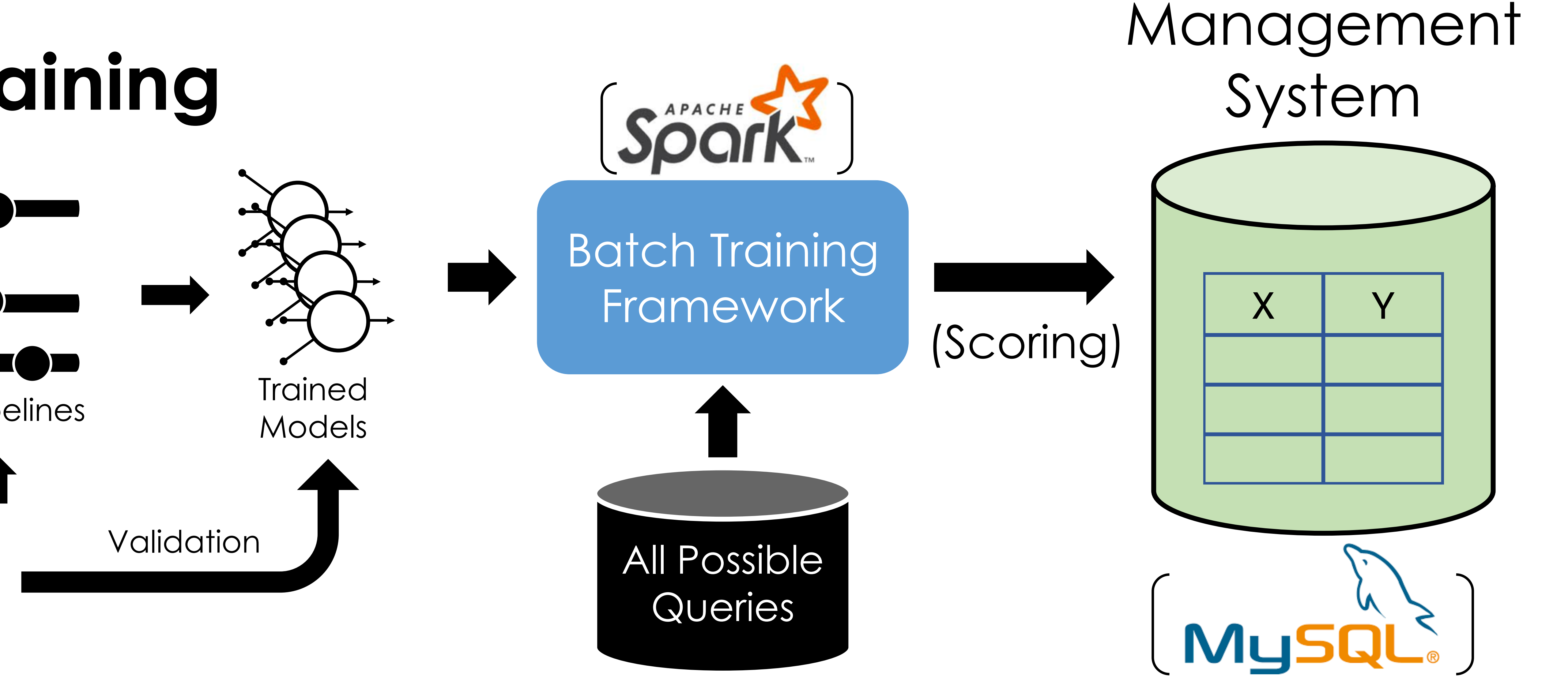
Pre-materialized Predictions



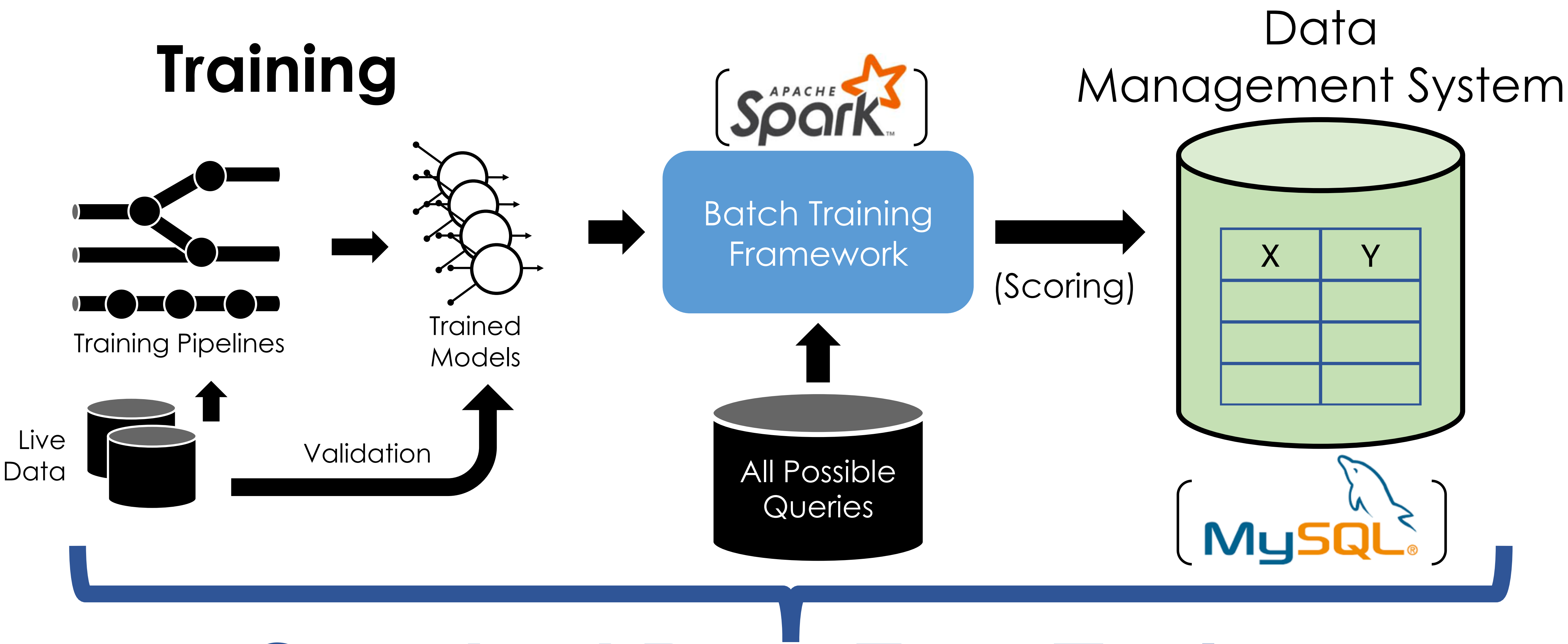
Pre-materialized Predictions



Pre-materialized Predictions

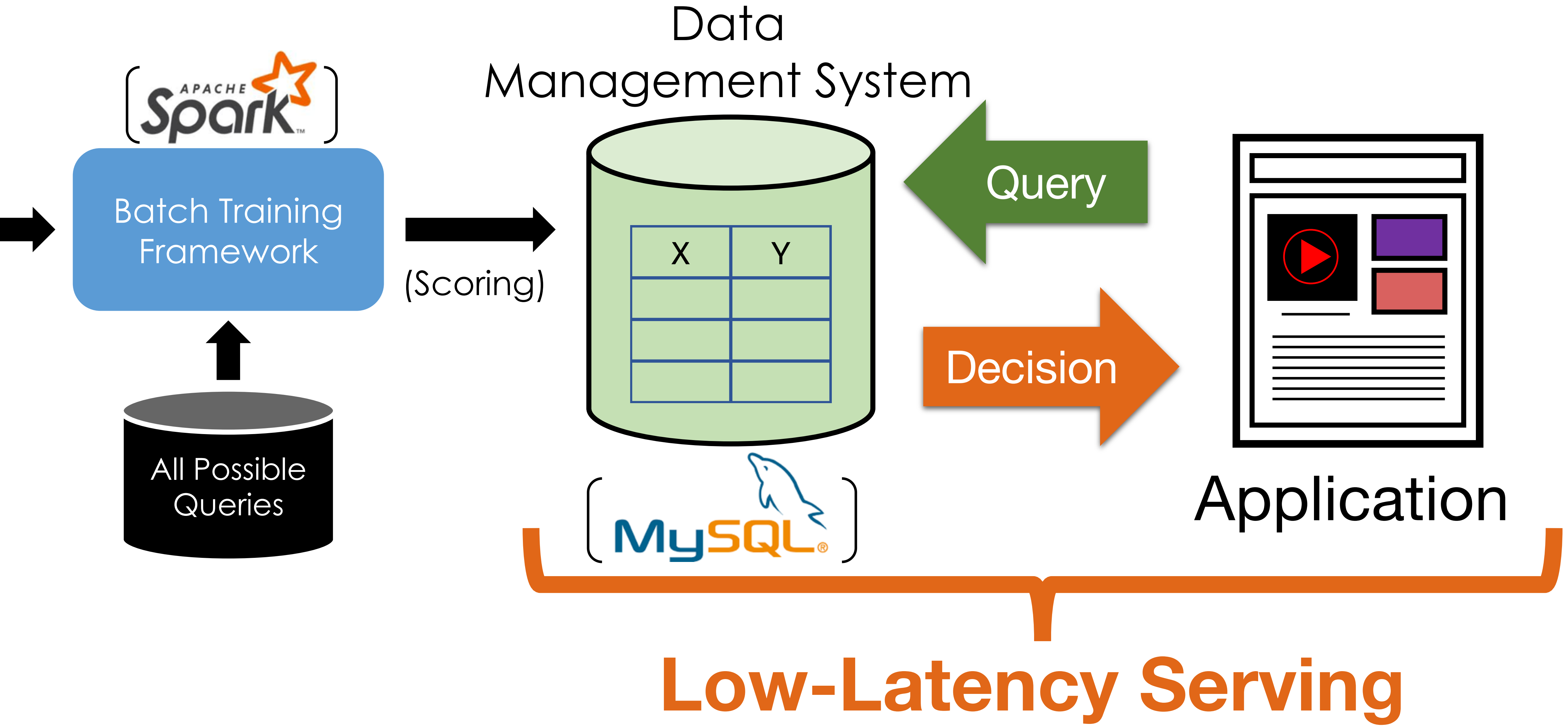


Pre-materialized Predictions



Standard Data Eng. Tools

Serving **Pre-materialized** Predictions



Serving **Pre-materialized** Predictions

Advantages:

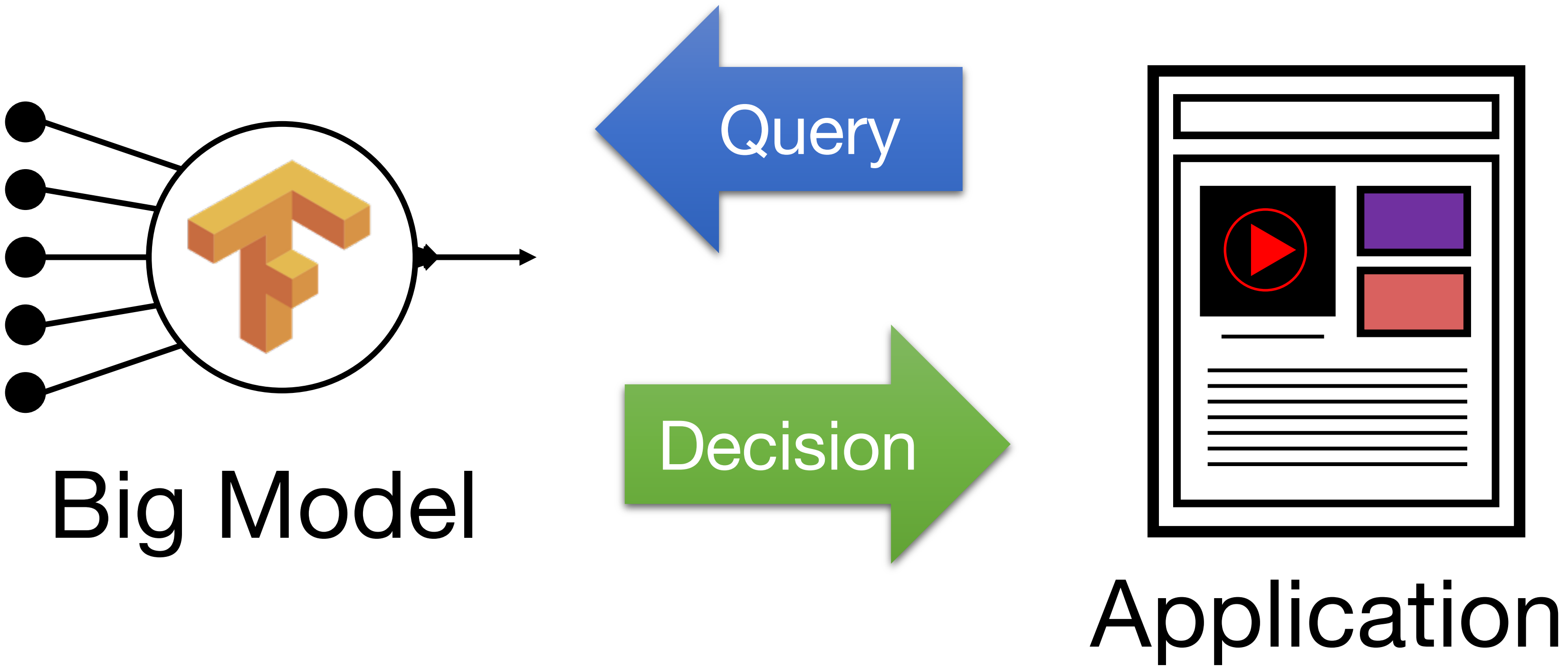
- Leverage existing **data serving** and **model training** infrastructure
- **Batch processing** improves hardware perf.
- **Indexing** support for complex queries
  - Find all ***Pr("cute")*** dresses **where price < $20**
- More **predictable** performance

Serving **Pre-materialized** Predictions

Problems:

- Requires full set of **queries ahead of time**
  - Small and **bounded input domain**
- Requires substantial **computation** and **space**
  - Example: *scoring all content for all customers!*
- Costly update → rescore everything!

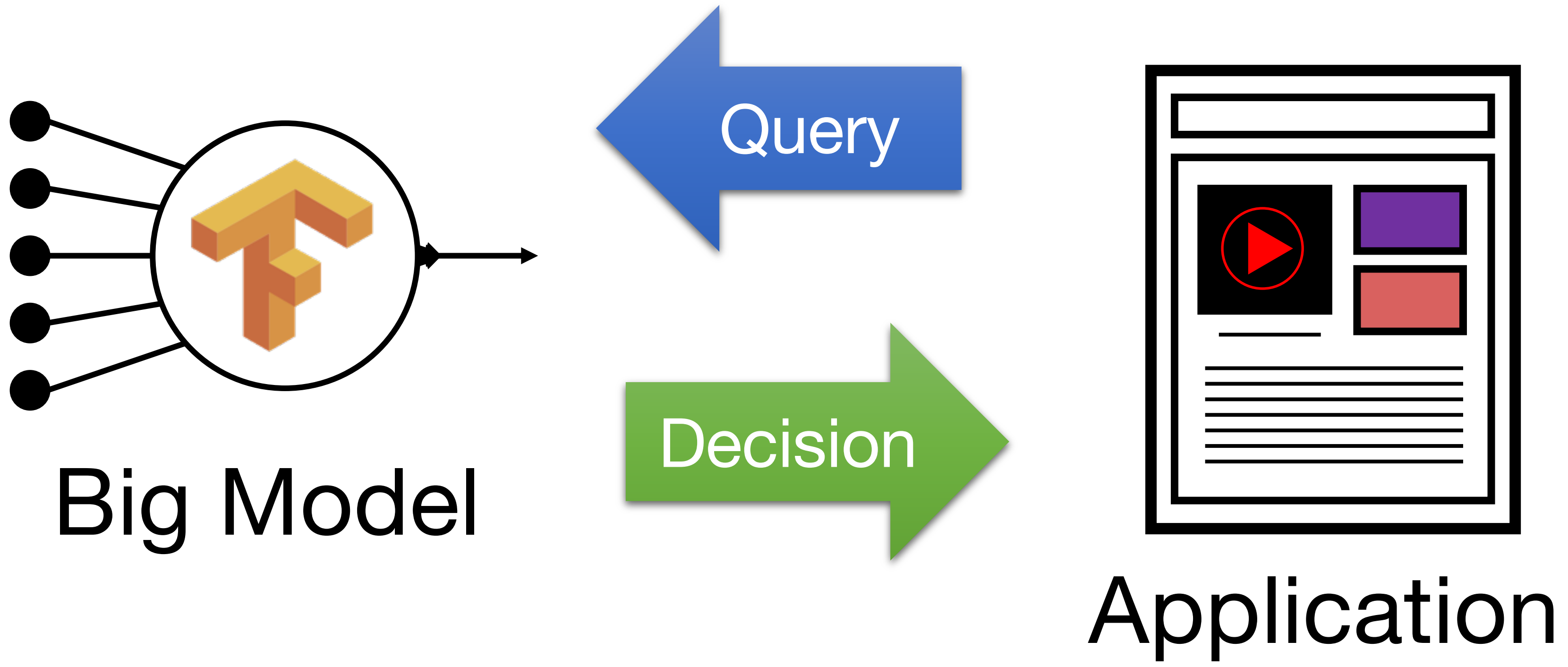
Inference



Two Approaches

- ***Offline:*** Pre-Materialize Predictions
- ***Online:*** Compute Predictions on the fly

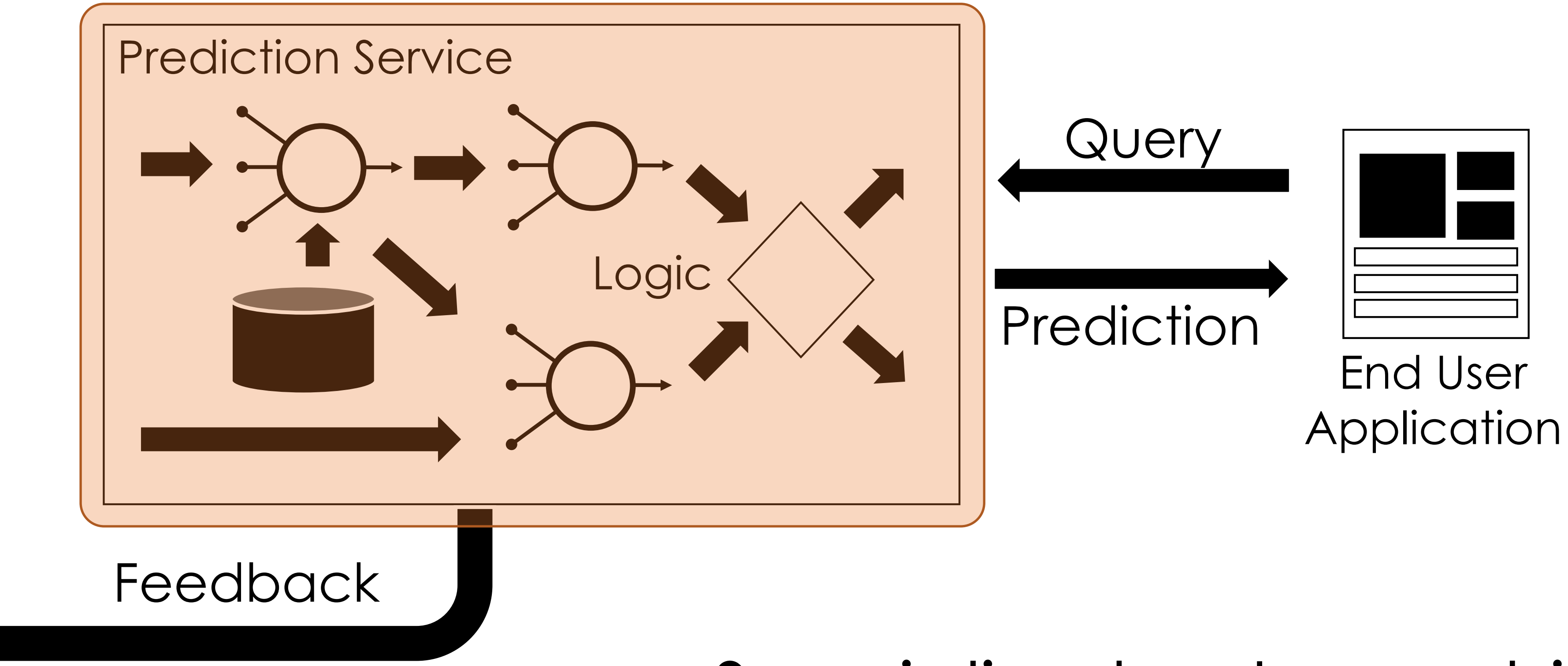
Inference



Two Approaches

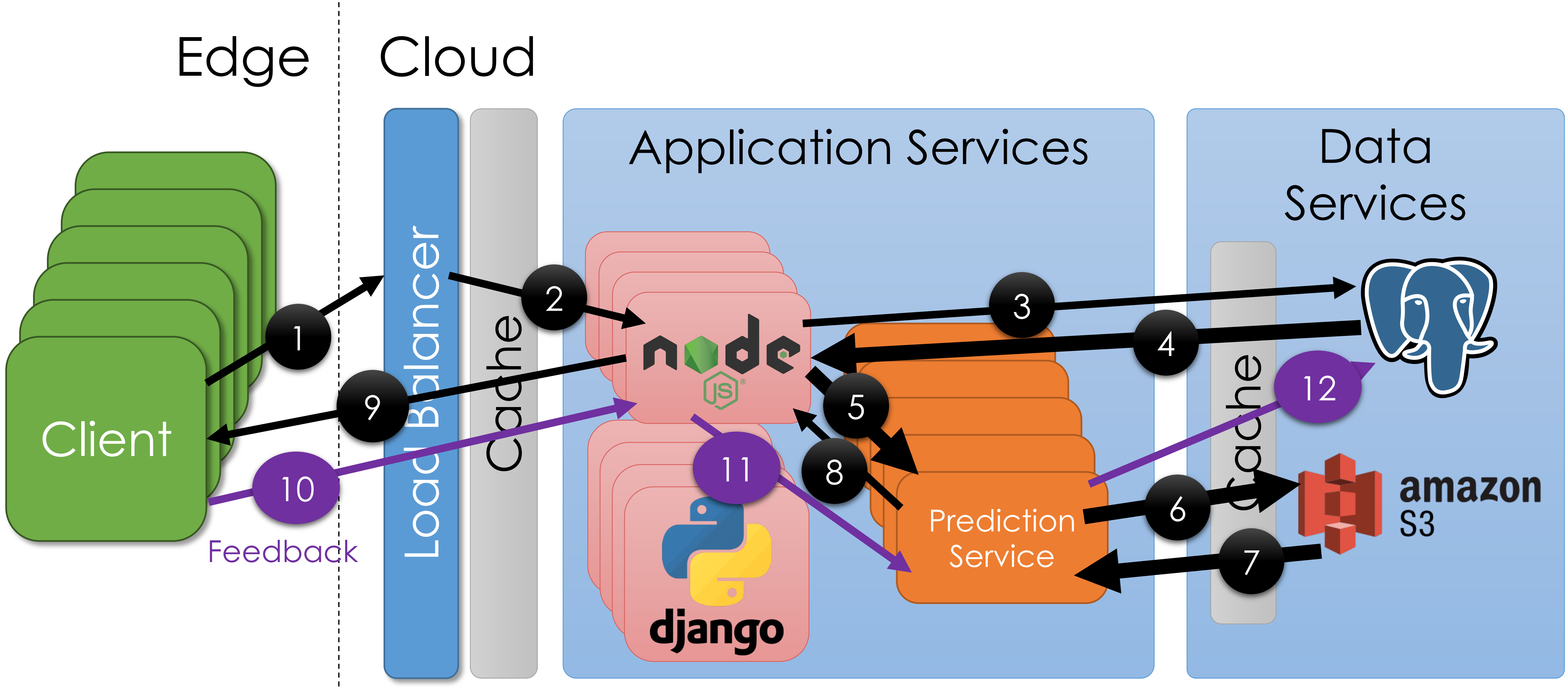
- ➤ ***Offline:*** Pre-Materialize Predictions
- ➤ ***Online:*** Compute Predictions on the fly

Prediction Services



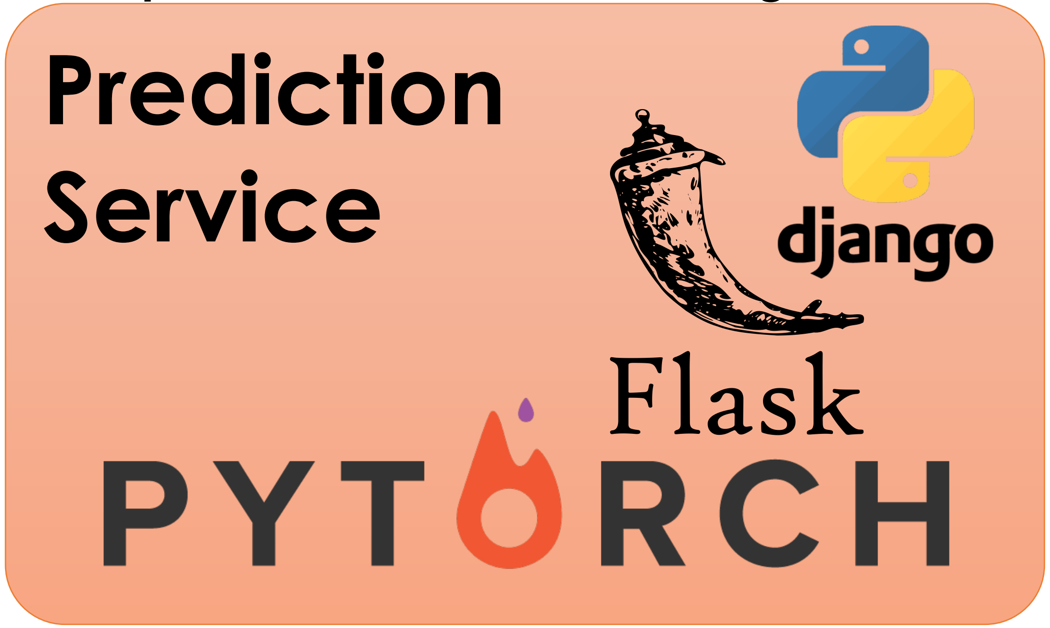
Specialized systems which render predictions at **query time**.

Architecture of a Prediction Service



Architecture of a Prediction Service

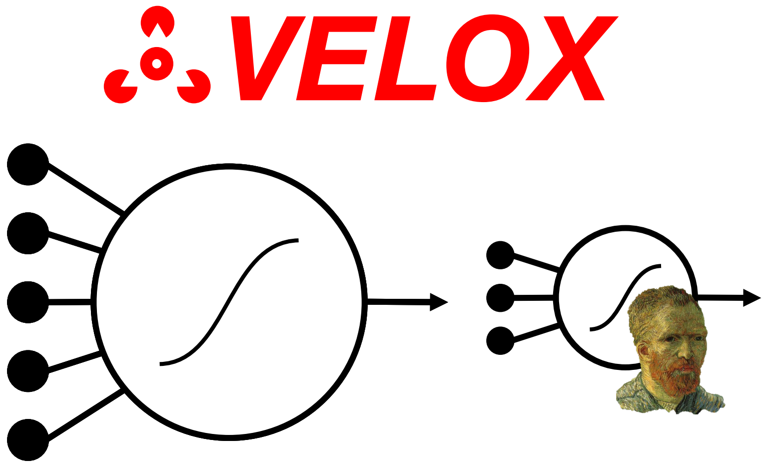
Simple Prediction Service Design



Use existing web technologies.

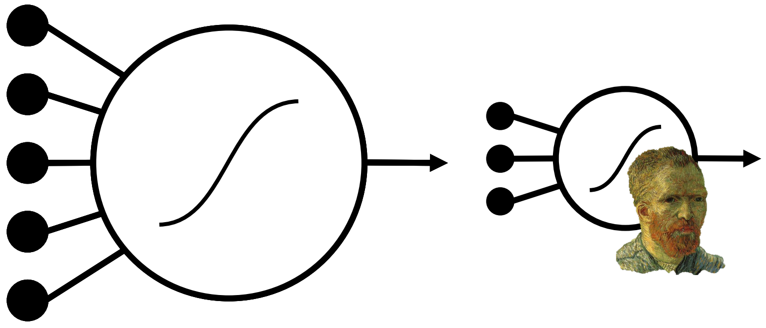
- **Strengths**
  - Leverages existing technologies
  - Easy to setup
- **Limitations**
  - Need to address common issues
    - batching, monitoring, etc…
  - Limited isolation between models
  - Missed opportunity for common abstraction

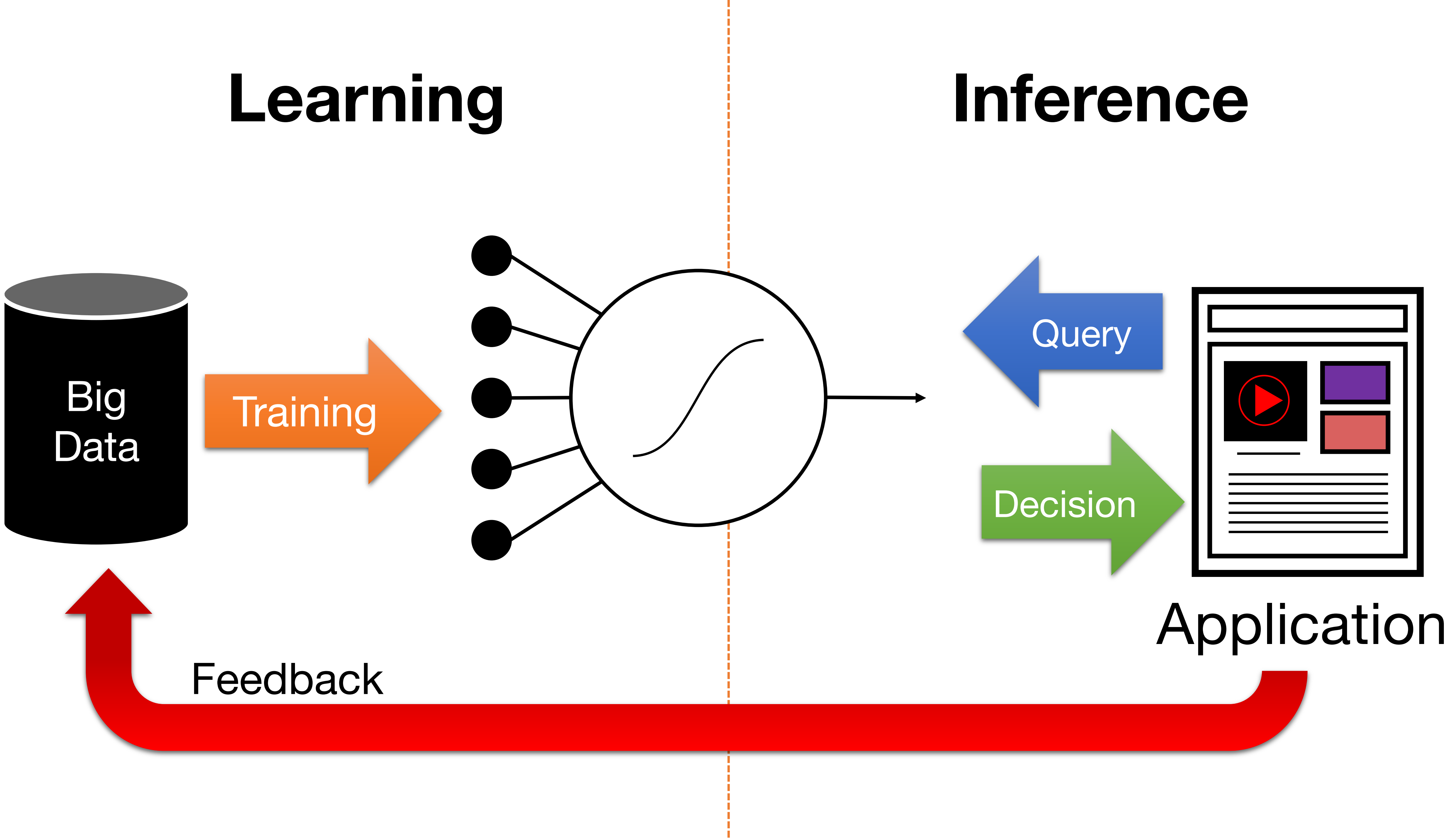
Two Approaches to
Prediction Service Design

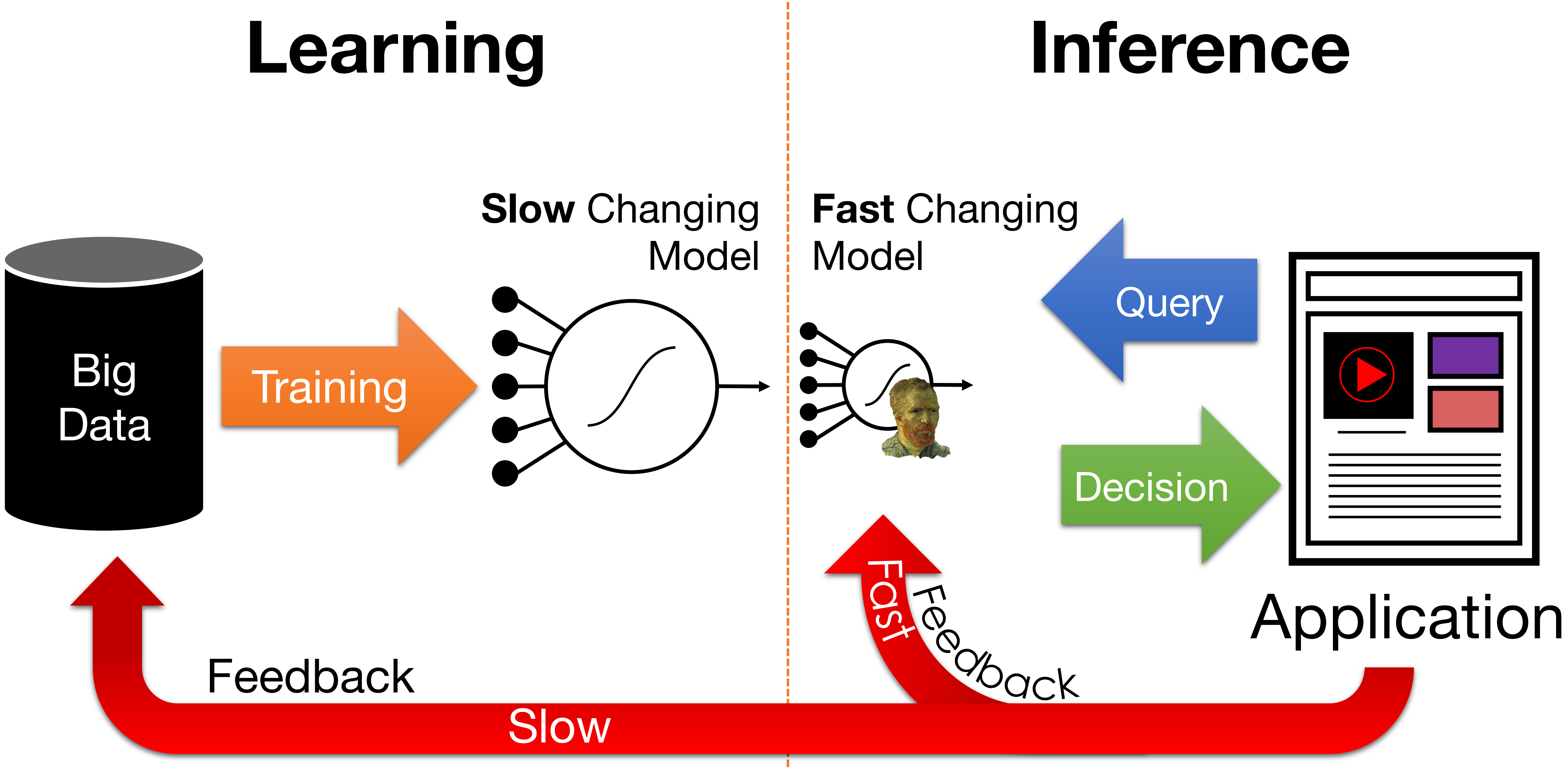


Addressing Feedback by
Learning at Different Speeds

VELOX




Learning
Inference
Big Data
Training
Query
Decision
Application
Feedback


Learning
Inference
Slow Changing Model
Fast Changing Model
Big Data
Training
Query
Decision
Application
Feedback
Slow
Fast
Feedback

Hybrid Offline + Online Learning

Update feature functions offline using batch solvers

- Leverage high-throughput systems (Tensor Flow)
- Exploit slow change in population statistics

$$f(x;\theta)^T w_u$$

Update the user weights online:

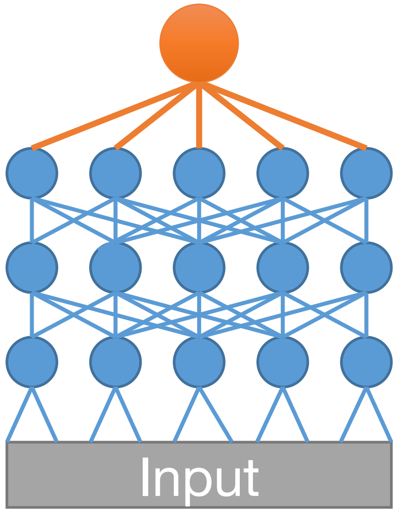
- Simple to train + more robust model
- Address rapidly changing user statistics

Common modeling structure

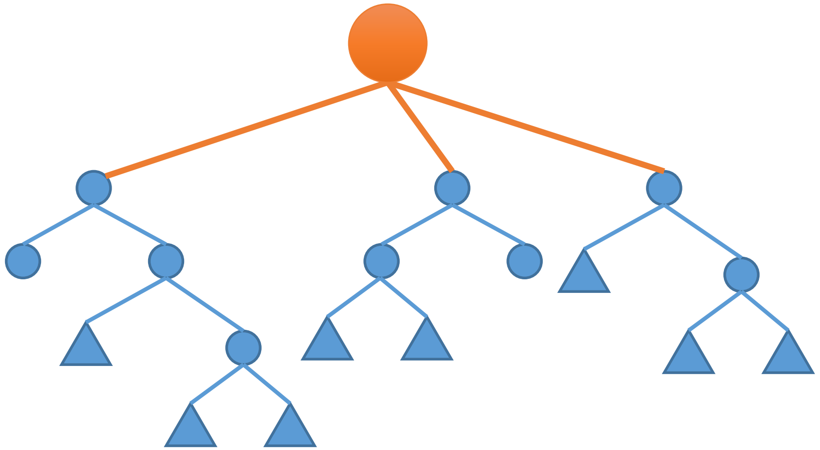
$$f(x;\theta)^T w_u$$

Matrix Factorization

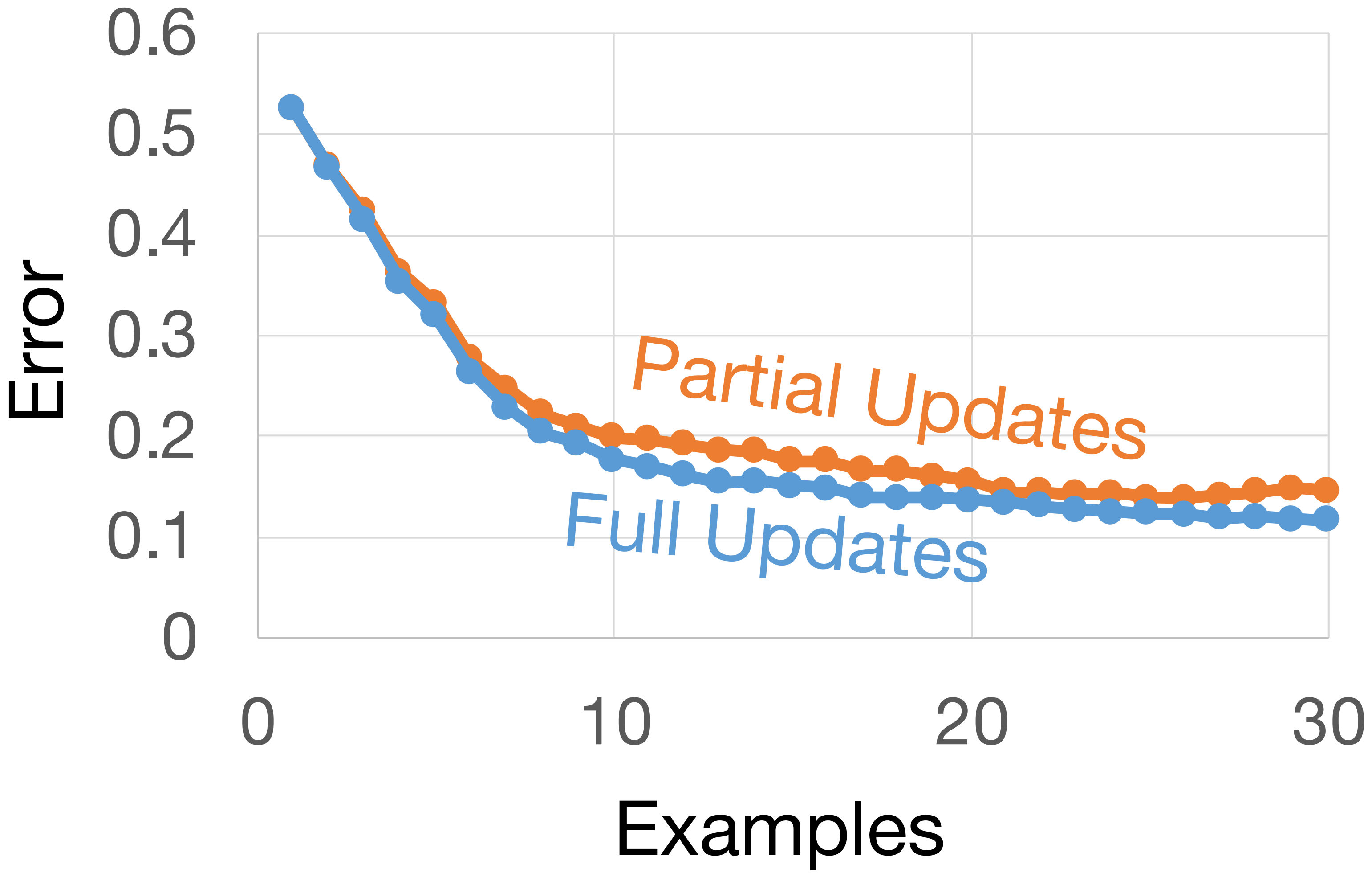
Deep Learning



Ensemble Methods

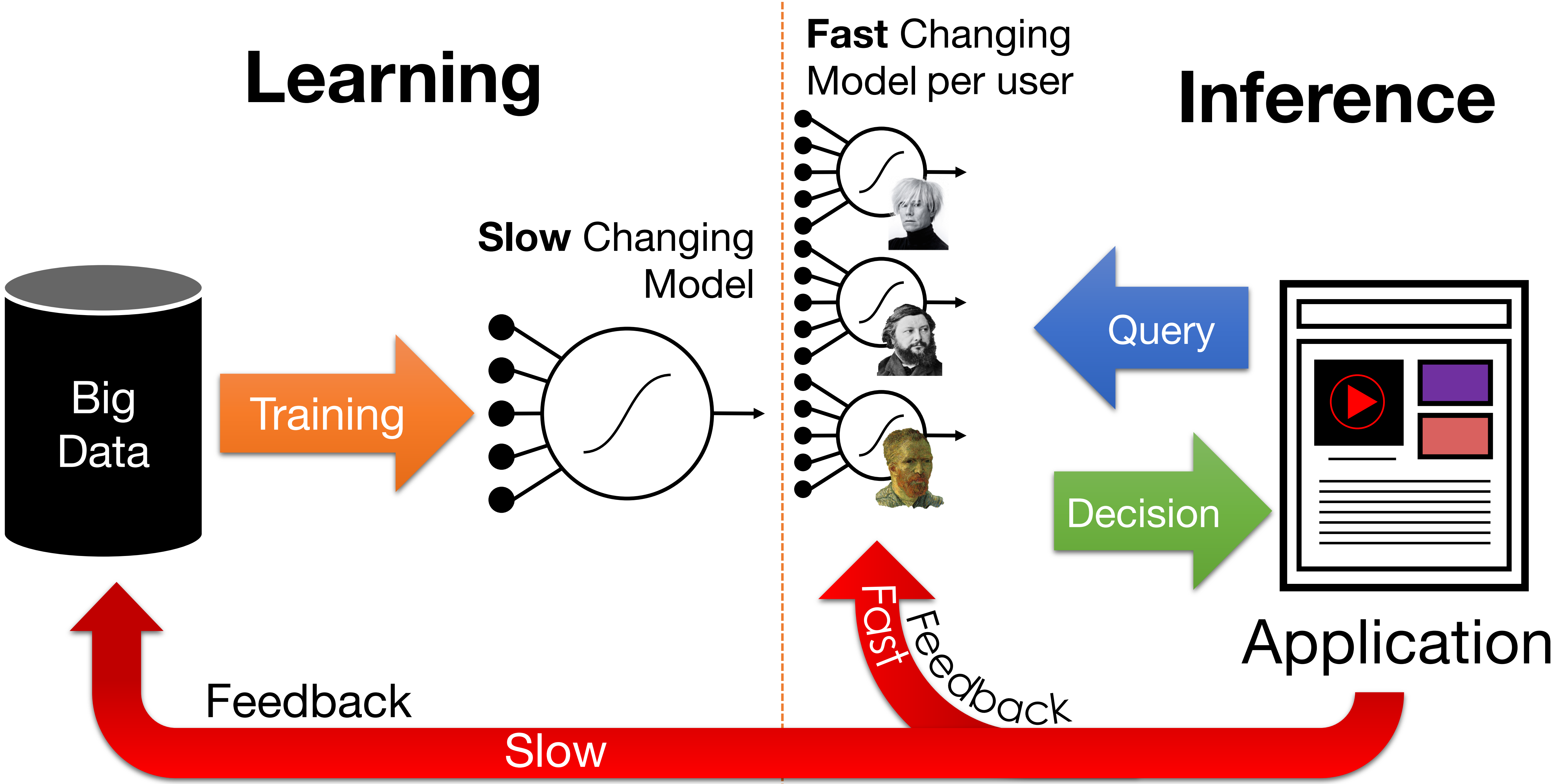


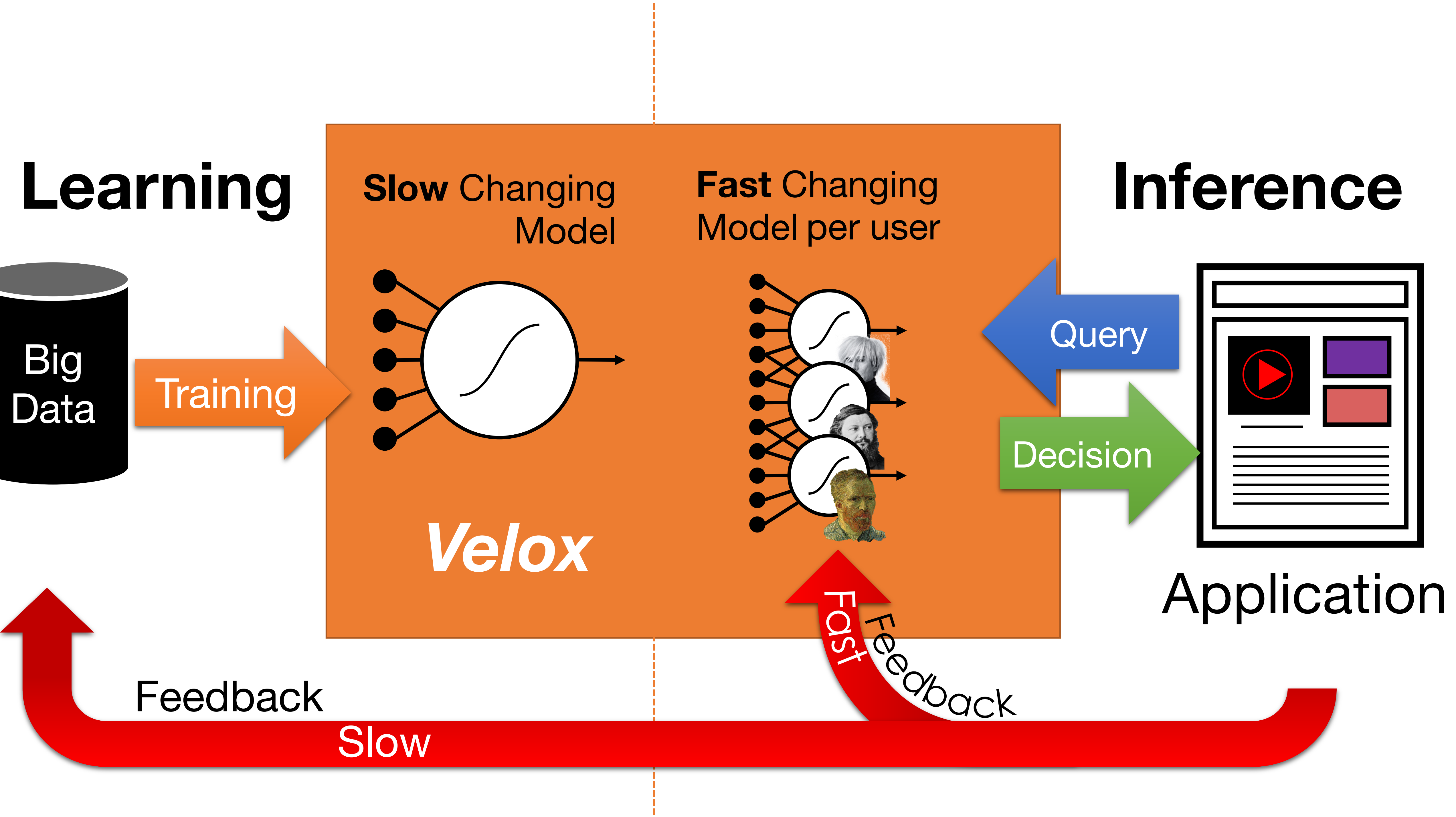
Velox Online Learning for Recommendations
(20-News Groups)



Partial Updates: *0.4 ms*
Retraining: *7.1 seconds*

>4 orders-of-magnitude ***faster adaptation***


Learning
Inference
Fast Changing
Model per user
Slow Changing
Model
Big
Data
Training
Query
Decision
Application
Feedback
Slow
Fast
Feedback


Learning
Inference
Slow Changing Model
Fast Changing Model per user
Big Data
Training
Query
Decision
Velox
Application
Fast Feedback
Feedback
Slow

VELOX Architecture

Fraud Detection

Content Rec.

NETFLIX

Keystone ML

MLLib

Spark

Velox

Single JVM Instance

VELOX Architecture

Fraud Detection	Content Rec.	Personal Asst.	Robotic Control	Machine Translation

NETFLIX

Keystone ML

MLLib

Spark

Velox

Single JVM Instance

scikit learn

dmlc mxnet

VW

Dato Create

TensorFlow

KALDI

Caffe

theano

VELOX as a Middle Layer Arch?

Fraud Detection	Content Rec.	Personal Asst.	Robotic Control	Machine Translation
	NETFLIX			

Generalize *Velox*?

theano
Dato Create
KeystoneML
Caffe
TensorFlow
scikit learn
dmlc mxnet
VW
KALDI

Clipper **Generalizes** Velox Across ML Frameworks

Fraud Detection

Content Rec.

NETFLIX

Personal Asst.

Robotic Control

Machine Translation

Clipper

theano

Dato Create

KeystoneML

Caffe

TensorFlow

scikit learn

dmlc mxnet

VW

KALDI

NETFLIX

Middle layer for prediction serving.

Common Abstraction

System Optimizations

theano

scikit learn

Dato

Apache Spark

Caffe

TensorFlow

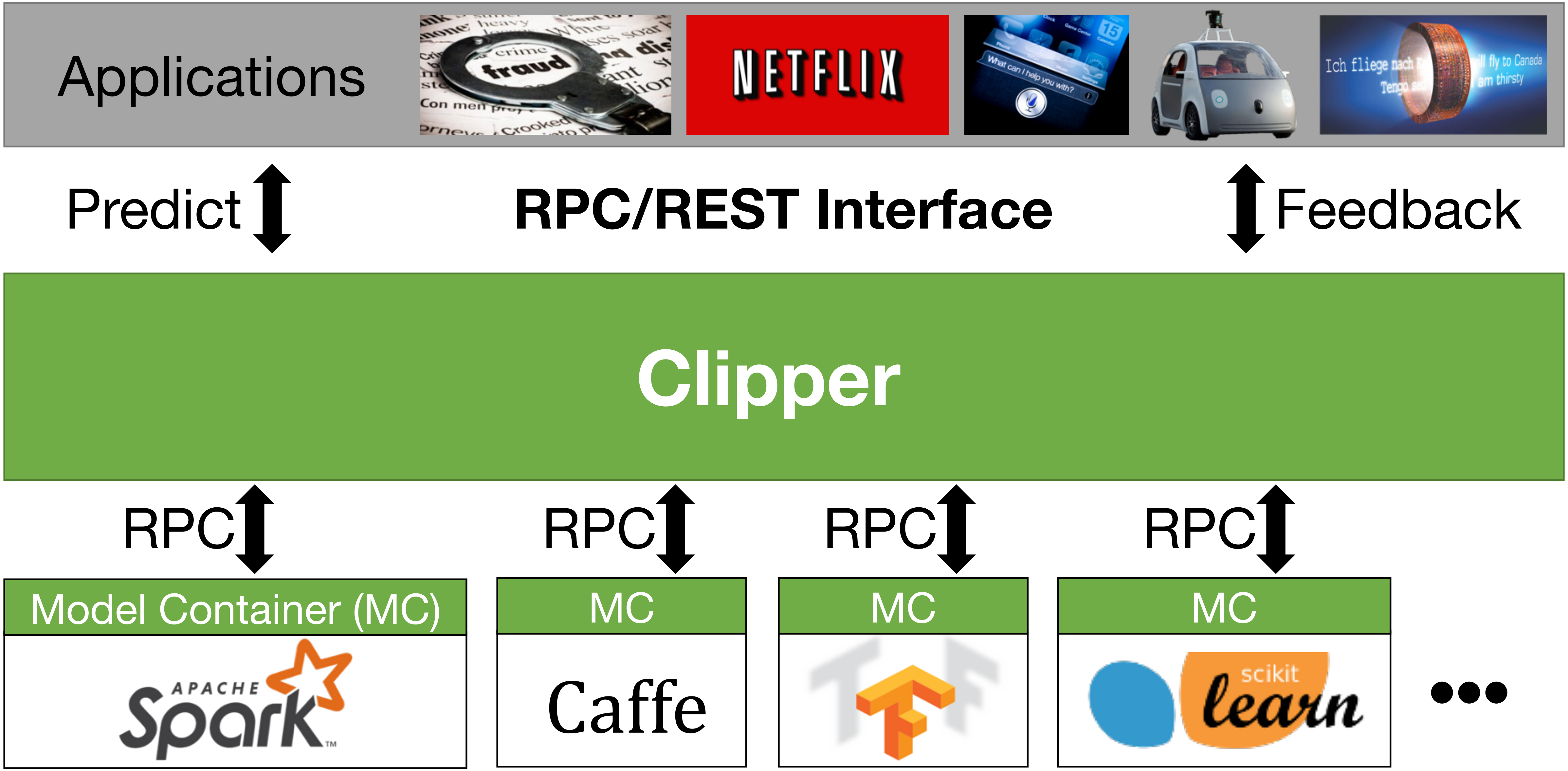
Create

dmlc

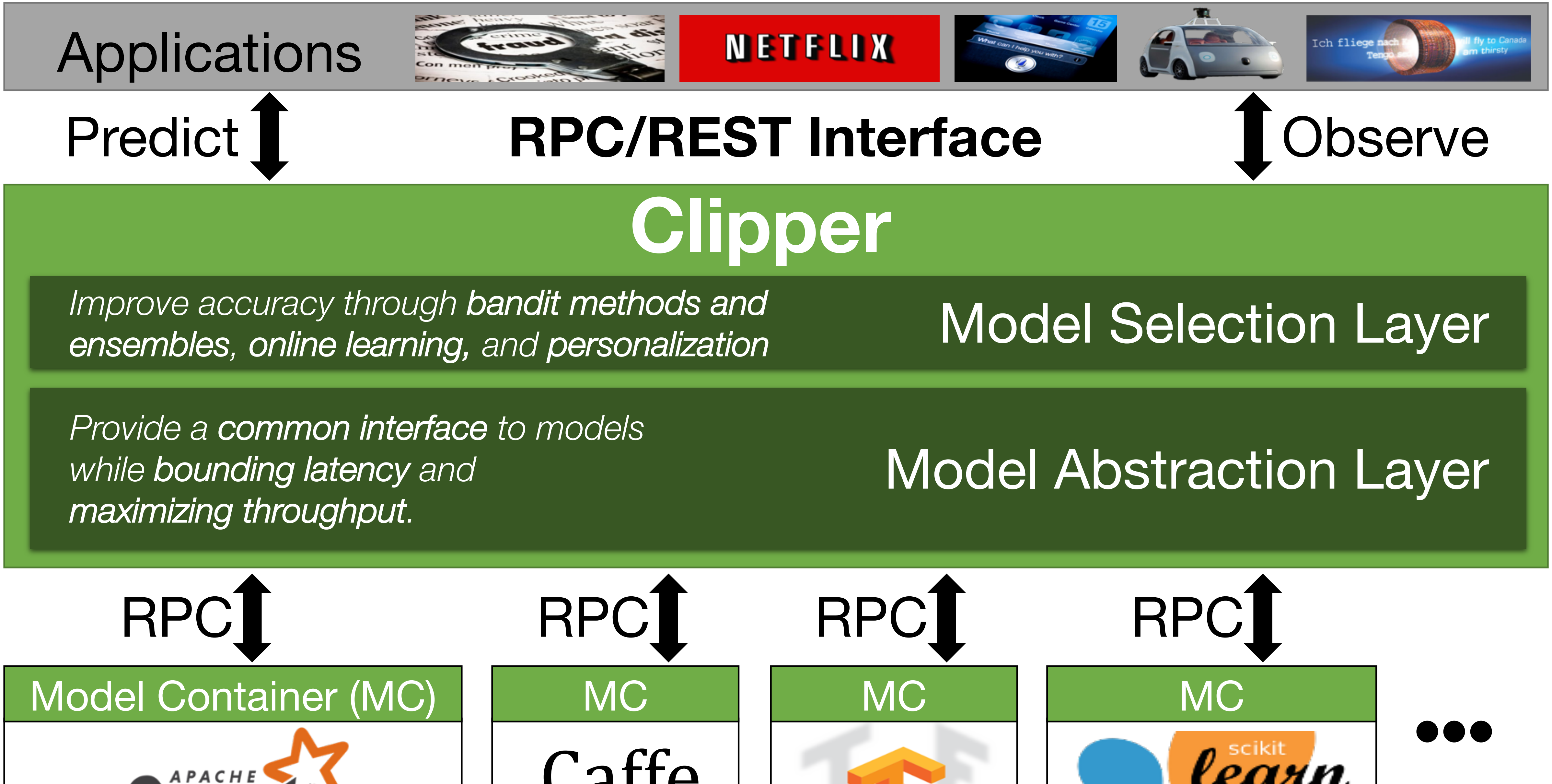
mxnet

KALDI

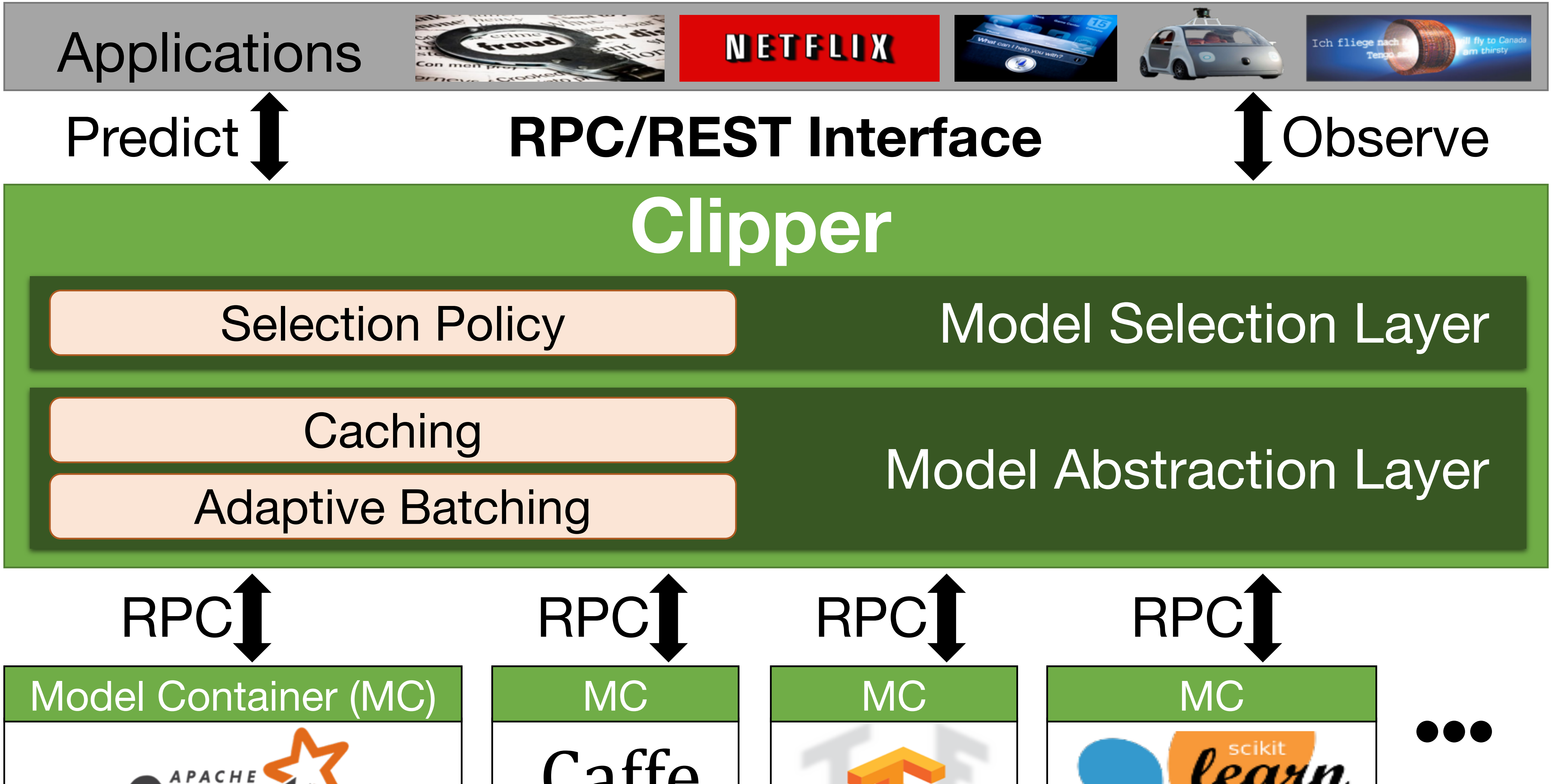
Clipper Decouples Applications and Models

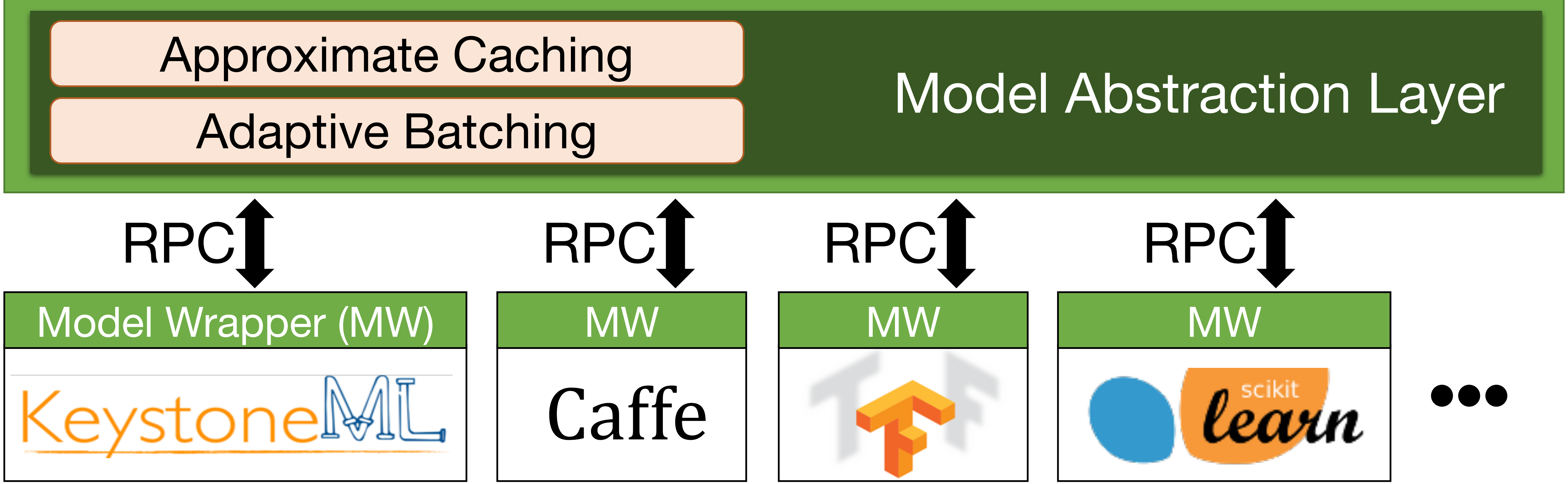


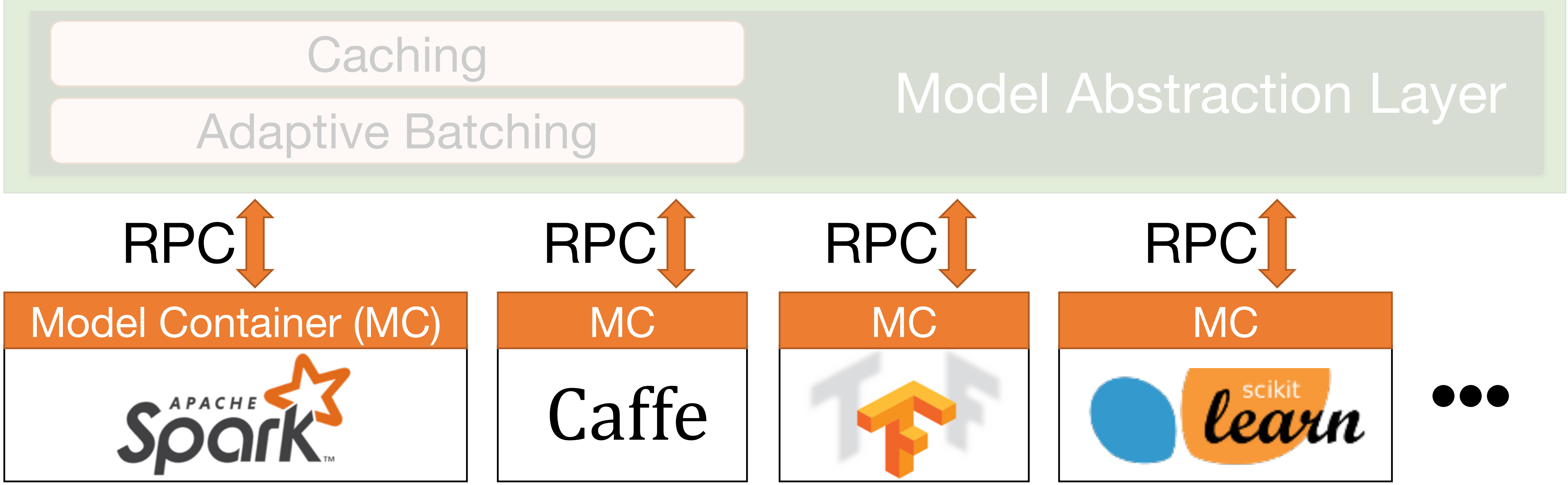
Clipper Architecture



Clipper Architecture




Approximate Caching
Adaptive Batching
Model Abstraction Layer
RPC
RPC
RPC
RPC
Model Wrapper (MW)
MW
MW
MW
KeystoneML
Caffe
scikit learn
•••



Common Interface → Simplifies Deployment:

➢ Evaluate models using original code & systems

Container-based Model Deployment

Implement Model API:

```
class ModelContainer:
    def __init__(model_data)
    def predict_batch(inputs)
```

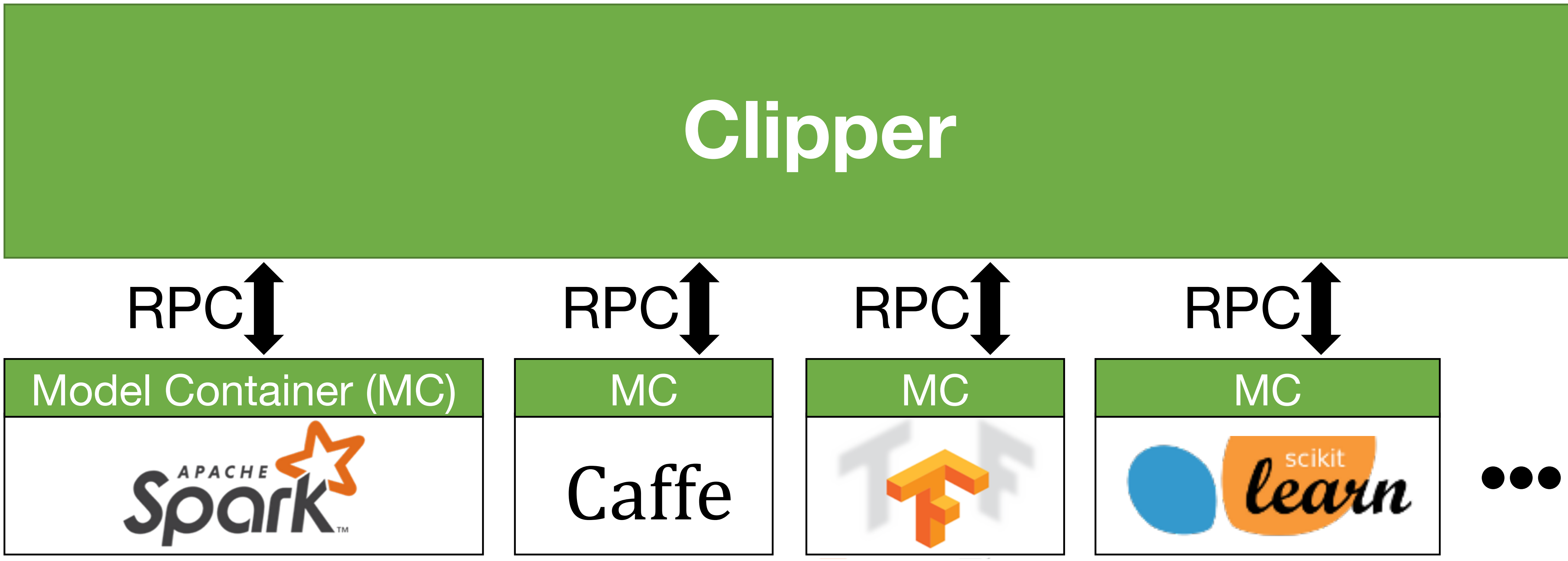
Container-based Model Deployment

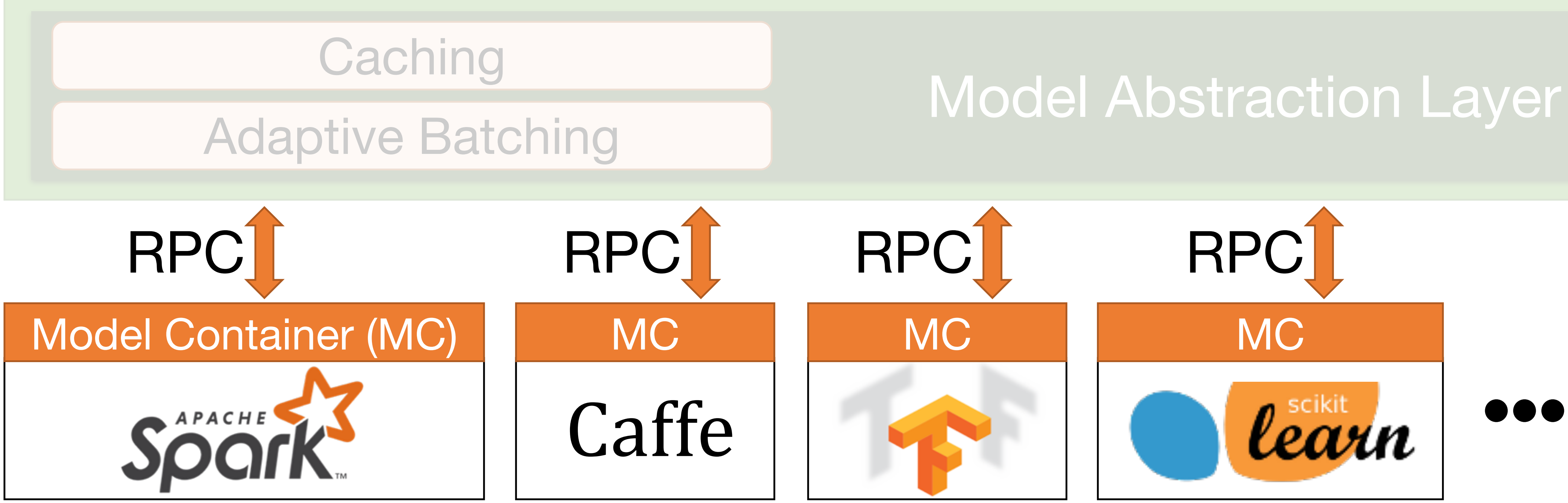
Model implementation packaged in container

Model Container (MC)

```
class ModelContainer:
    def __init__(model_data)
    def predict_batch(inputs)
```

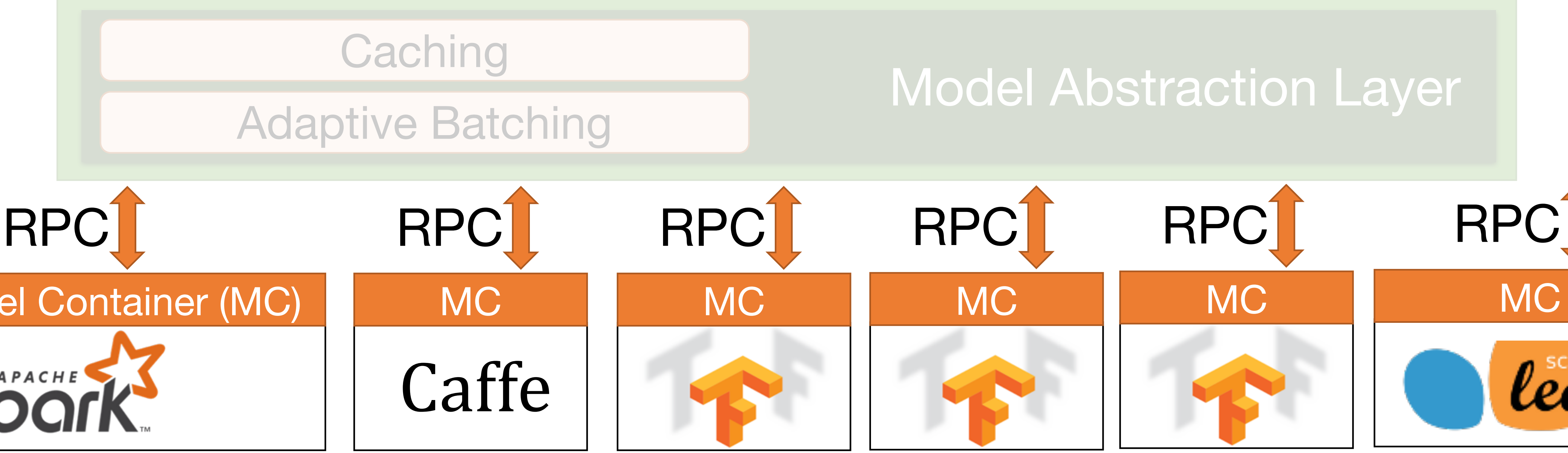
Container-based Model Deployment





Common Interface → Simplifies Deployment:

- ➢ Evaluate models using original code & systems
- ➢ Models run in separate processes as Docker containers
  - ➢ Resource isolation



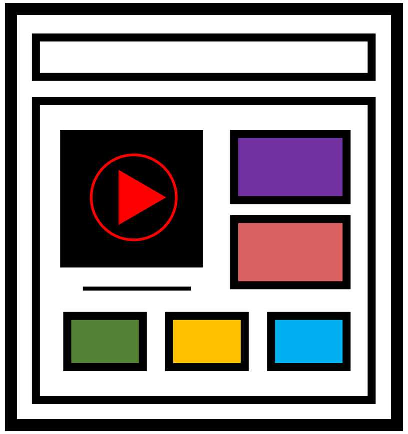
Common Interface → Simplifies Deployment:

- Evaluate models using original code & systems
- Models run in separate processes as Docker containers
  - Resource isolation
  - Scale-out

Problem: frameworks optimized for **batch processing** not **latency**

Batching to Improve Throughput

- Why batching helps:



A single page load may generate many queries

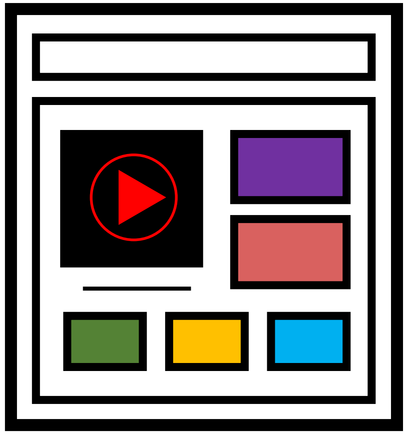
Hardware Acceleration

Helps amortize system overhead

- Optimal batch depends on:
  - hardware configuration
  - model and framework
  - system load

Adaptive Batching to Improve Throughput

- Why batching helps:



A single page load may generate many queries

Hardware Acceleration

Helps amortize system overhead

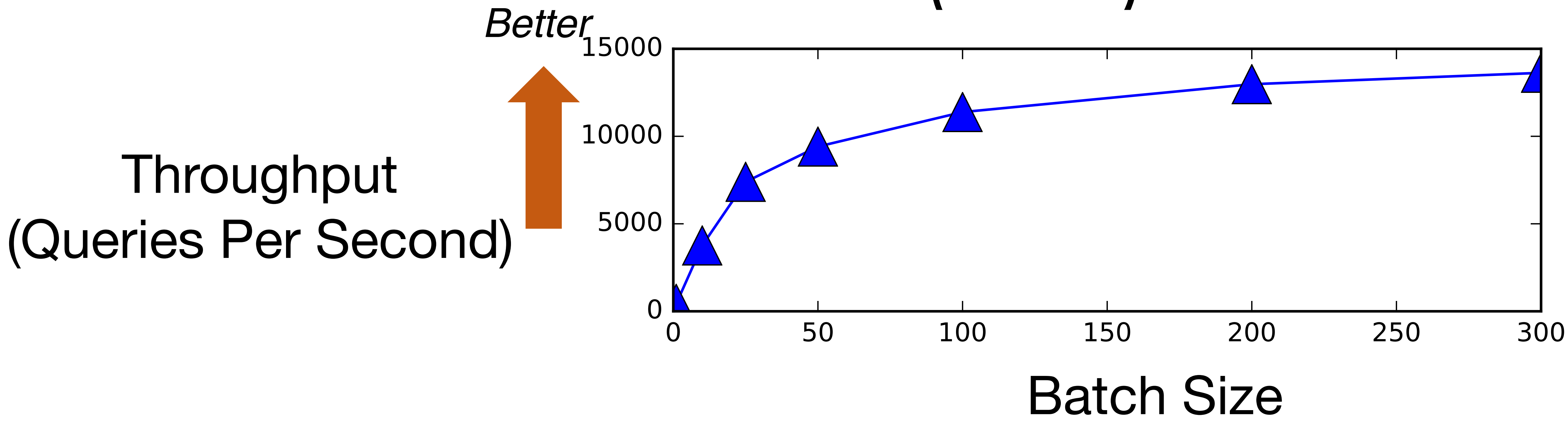
- Optimal batch depends on:
  - hardware configuration
  - model and framework
  - system load

Clipper Solution:

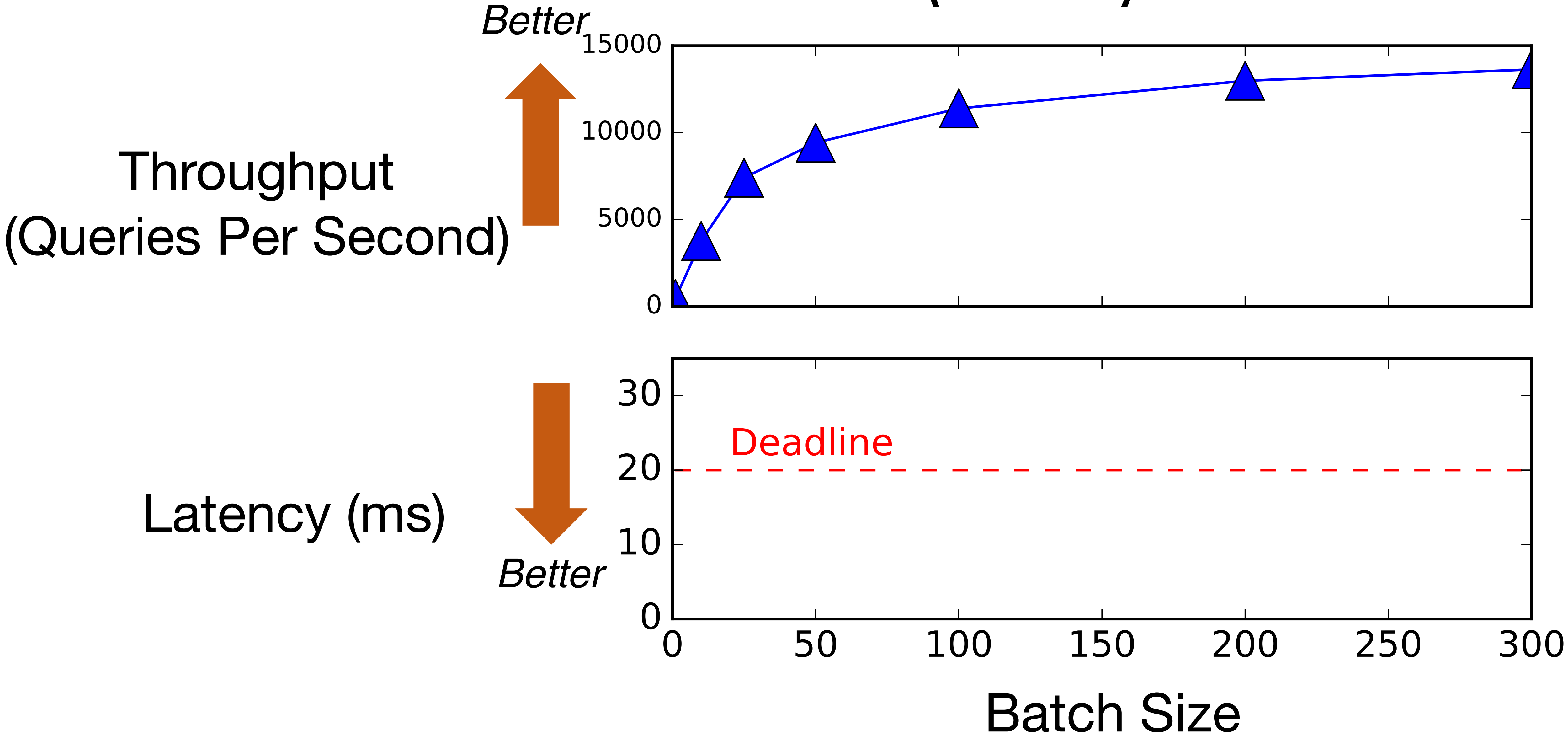
Adaptively tradeoff latency and throughput…

- Inc. batch size *until the latency objective is exceeded* (**Additive Increase**)
- If latency exceeds SLO cut batch size by a fraction (**Multiplicative Decrease**)

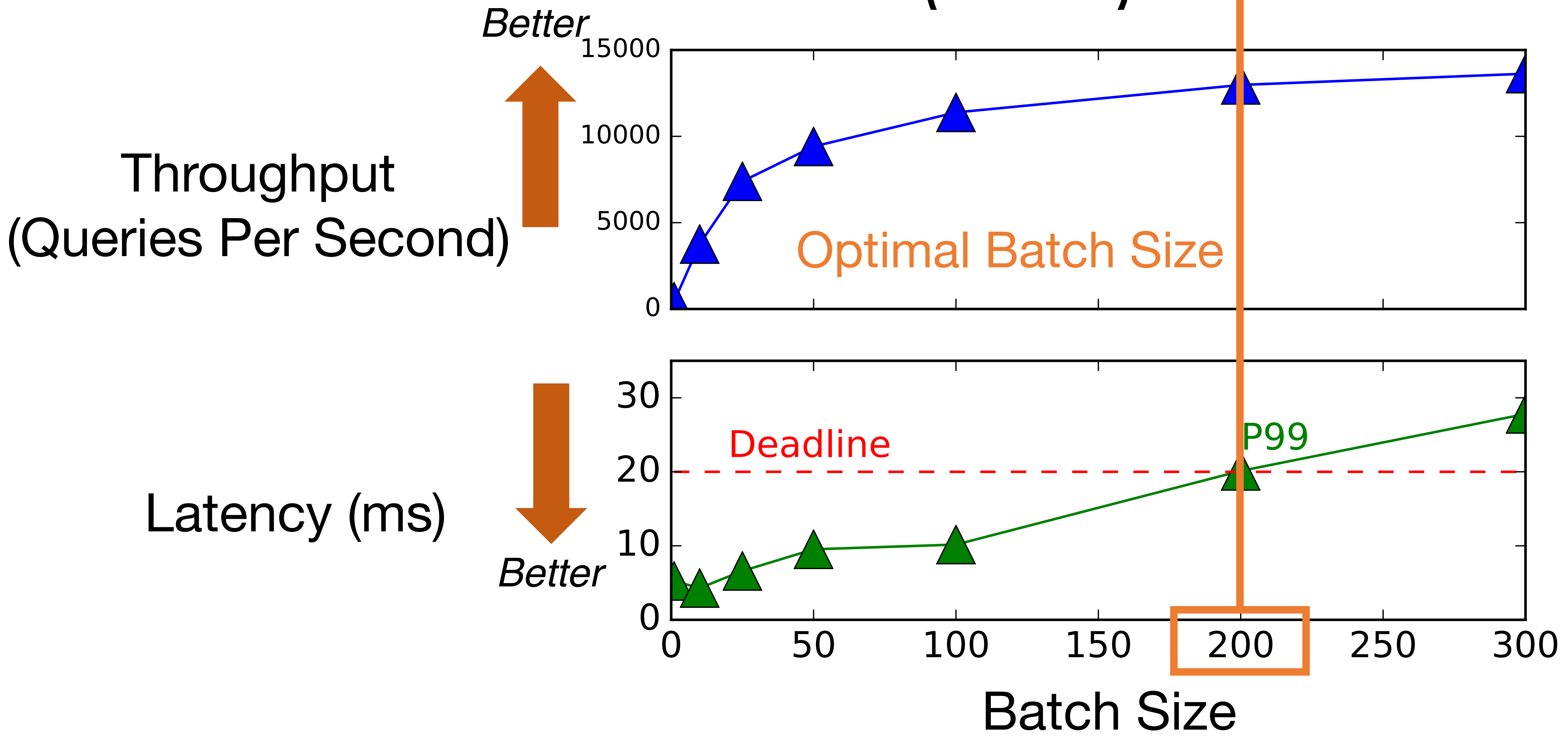
Tensor Flow Conv. Net (GPU)

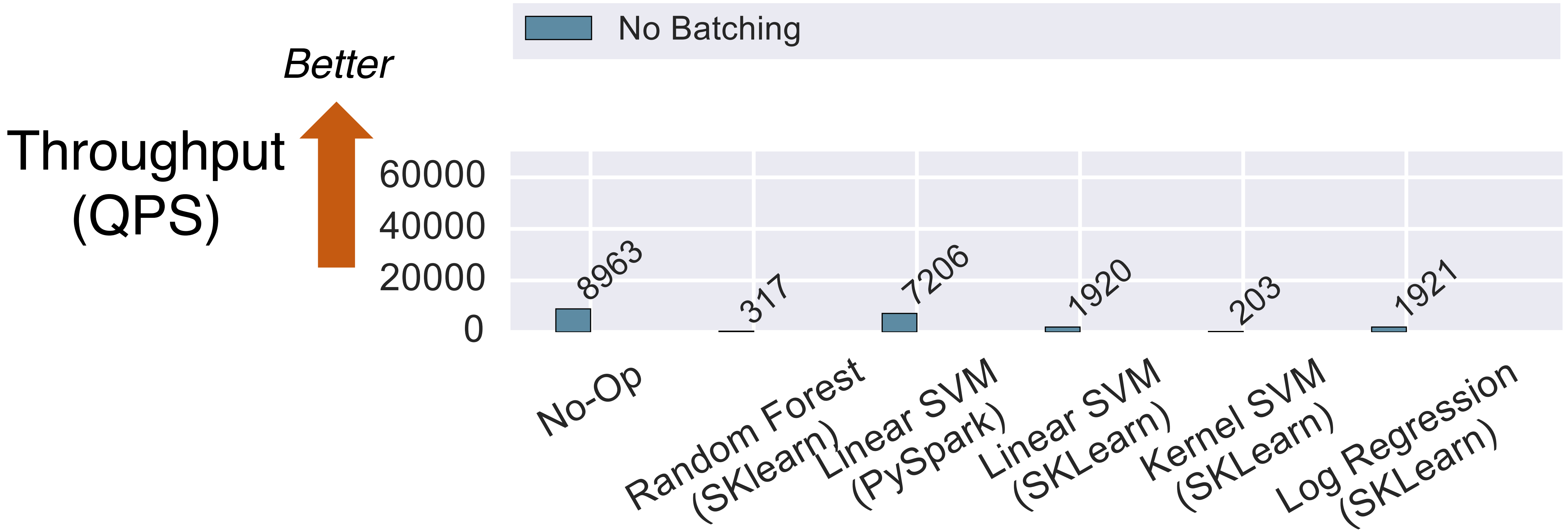


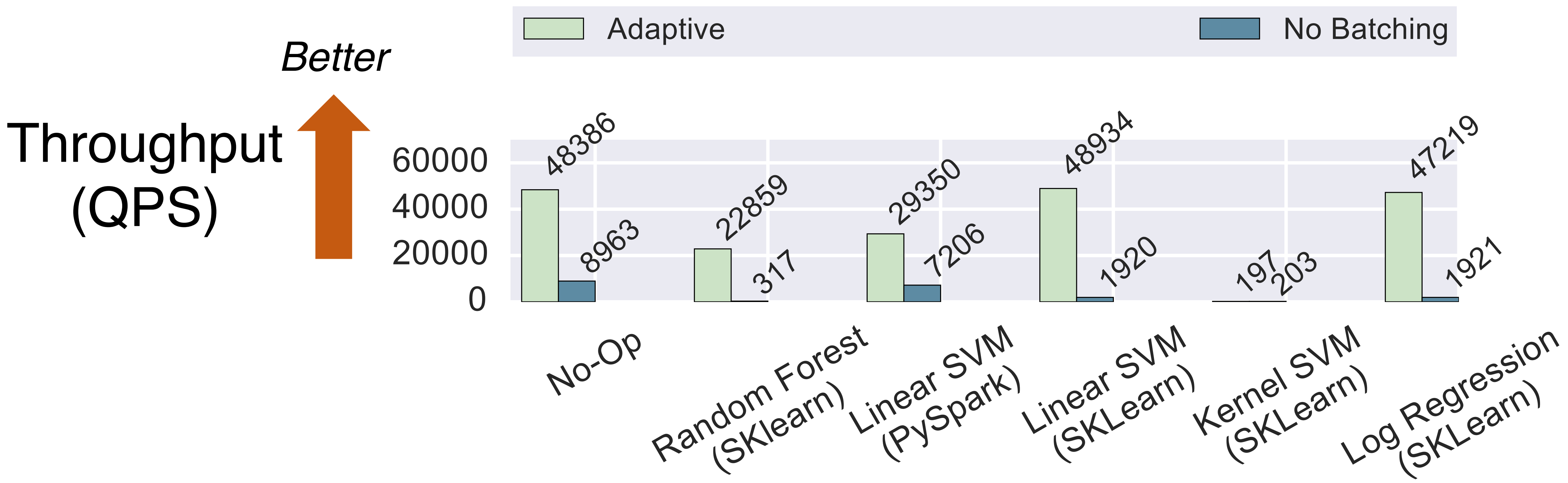
Tensor Flow Conv. Net (GPU)



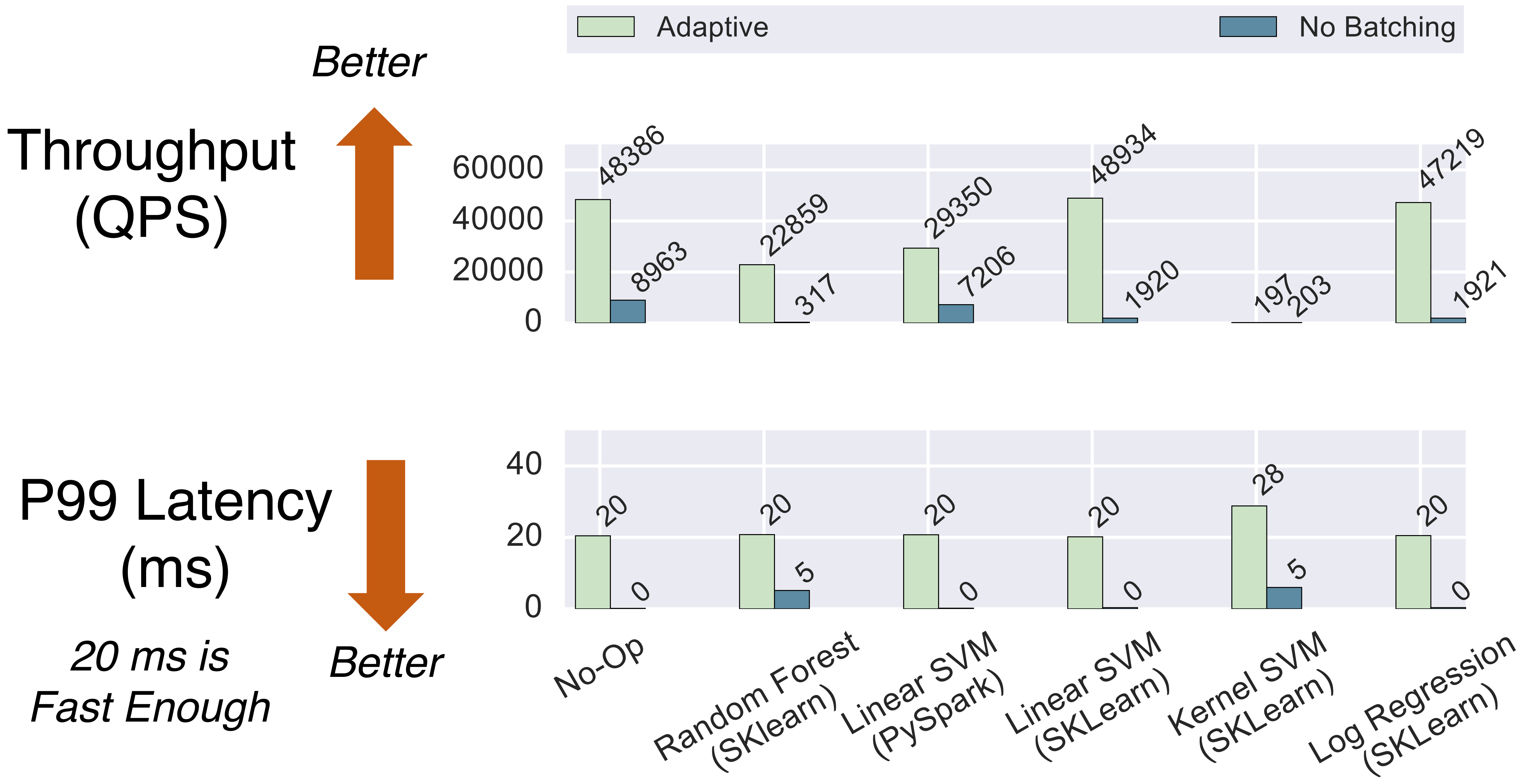
Tensor Flow Conv. Net (GPU)

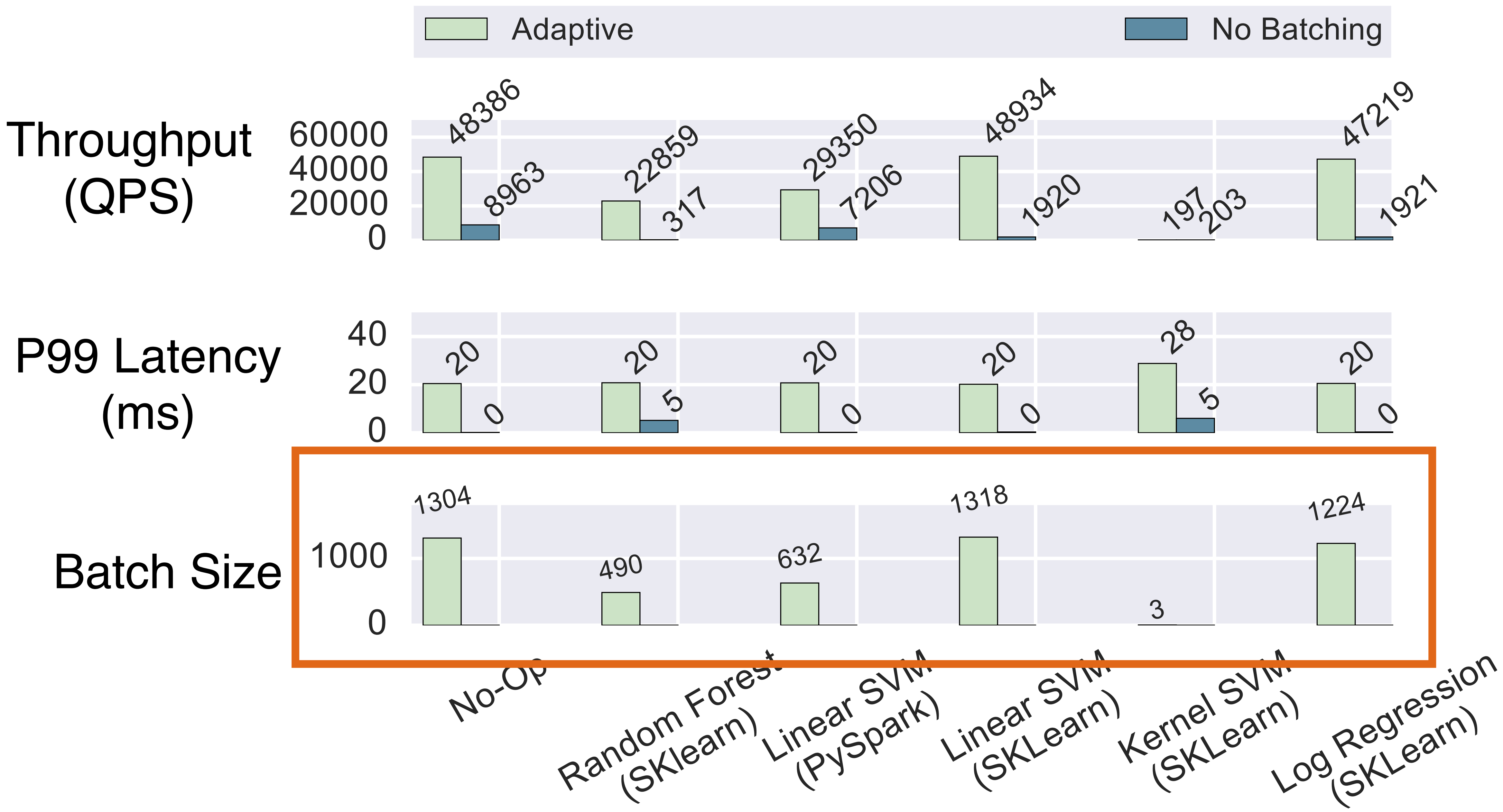



No Batching
Better
Throughput (QPS)
60000
40000
20000
0
8963
317
7206
1920
203
1921
No-Op
Random Forest (SKlearn)
Linear SVM (PySpark)
Linear SVM (SKLearn)
Kernel SVM (SKLearn)
Log Regression (SKLearn)



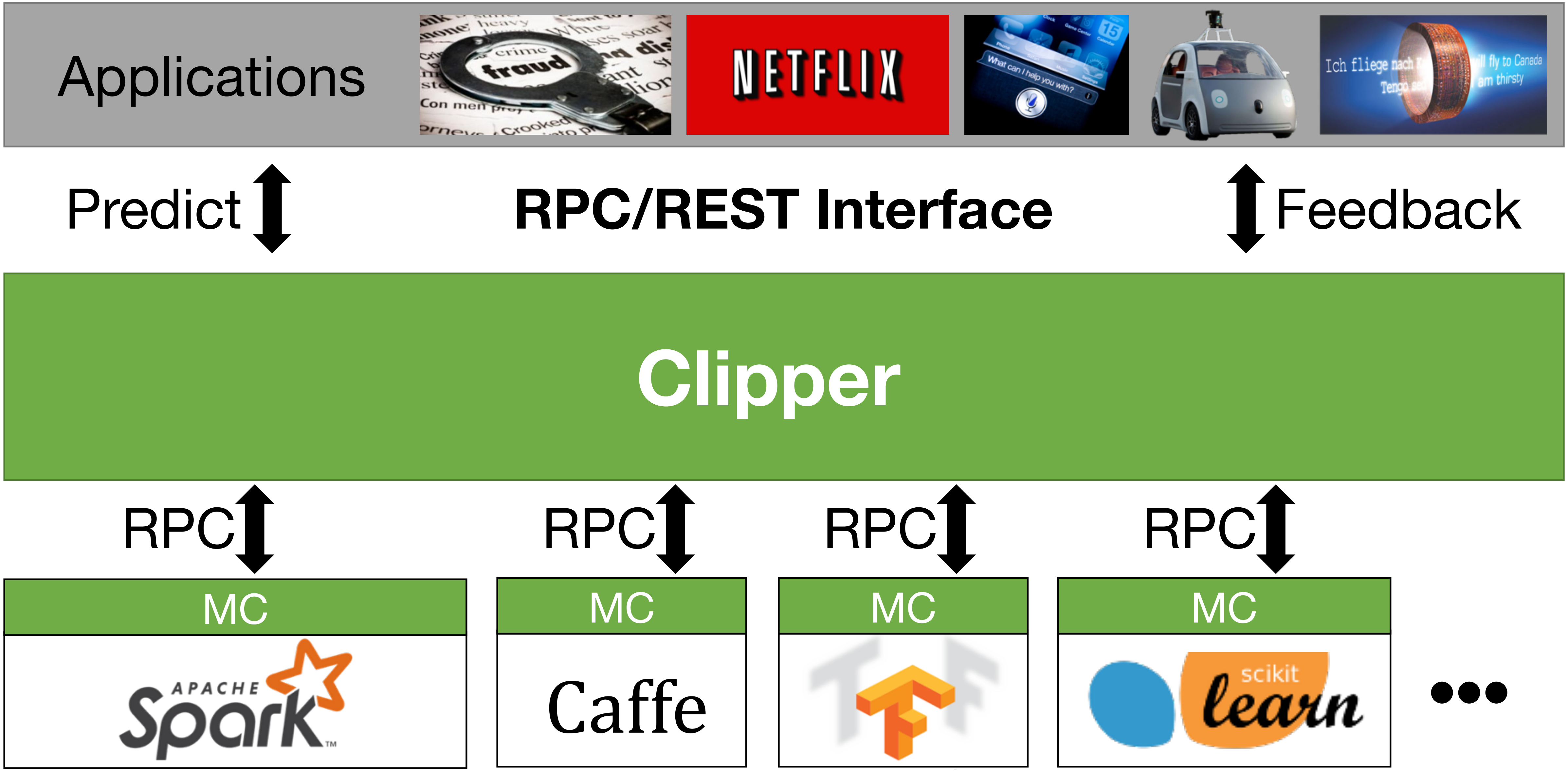
Adaptive
No Batching
Better
Throughput (QPS)
60000
40000
20000
0
48386
8963
22859
317
29350
7206
48934
1920
197
203
47219
1921
No-Op
Random Forest (SKlearn)
Linear SVM (PySpark)
Linear SVM (SKLearn)
Kernel SVM (SKLearn)
Log Regression (SKLearn)


Adaptive
No Batching
Better
Throughput
(QPS)
60000
40000
20000
0
48386
8963
22859
317
29350
7206
48934
1920
197
203
47219
1921
P99 Latency
(ms)
40
20
0
20
0
20
5
20
0
20
0
28
5
20
0
20 ms is
Fast Enough
Better
No-Op
Random Forest
(SKlearn)
Linear SVM
(PySpark)
Linear SVM
(SKLearn)
Kernel SVM
(SKLearn)
Log Regression
(SKLearn)

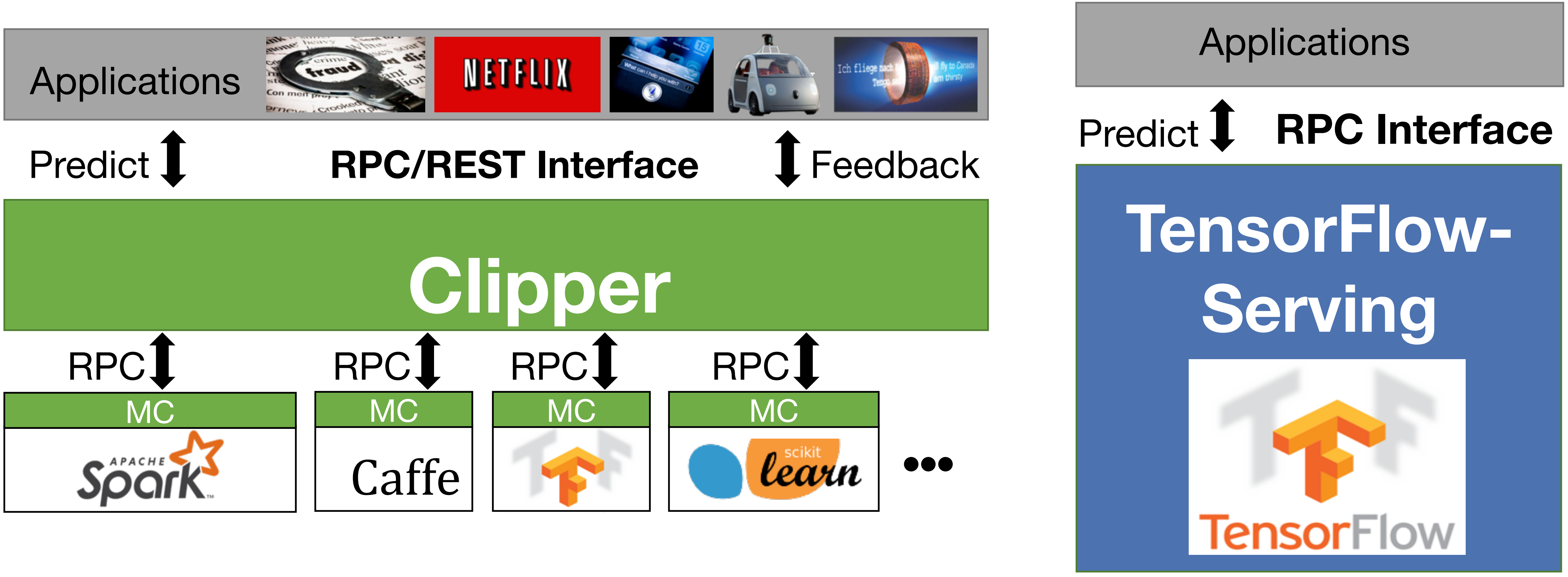


Adaptive
No Batching
Throughput (QPS)
60000
40000
20000
0
48386
8963
22859
317
29350
7206
48934
1920
197
203
47219
1921
P99 Latency (ms)
40
20
0
20
0
20
5
20
0
20
0
28
5
20
0
Batch Size
1000
0
1304
490
632
1318
3
1224
No-Op
Random Forest (SKlearn)
Linear SVM (PySpark)
Linear SVM (SKLearn)
Kernel SVM (SKLearn)
Log Regression (SKLearn)

Overhead of decoupled architecture

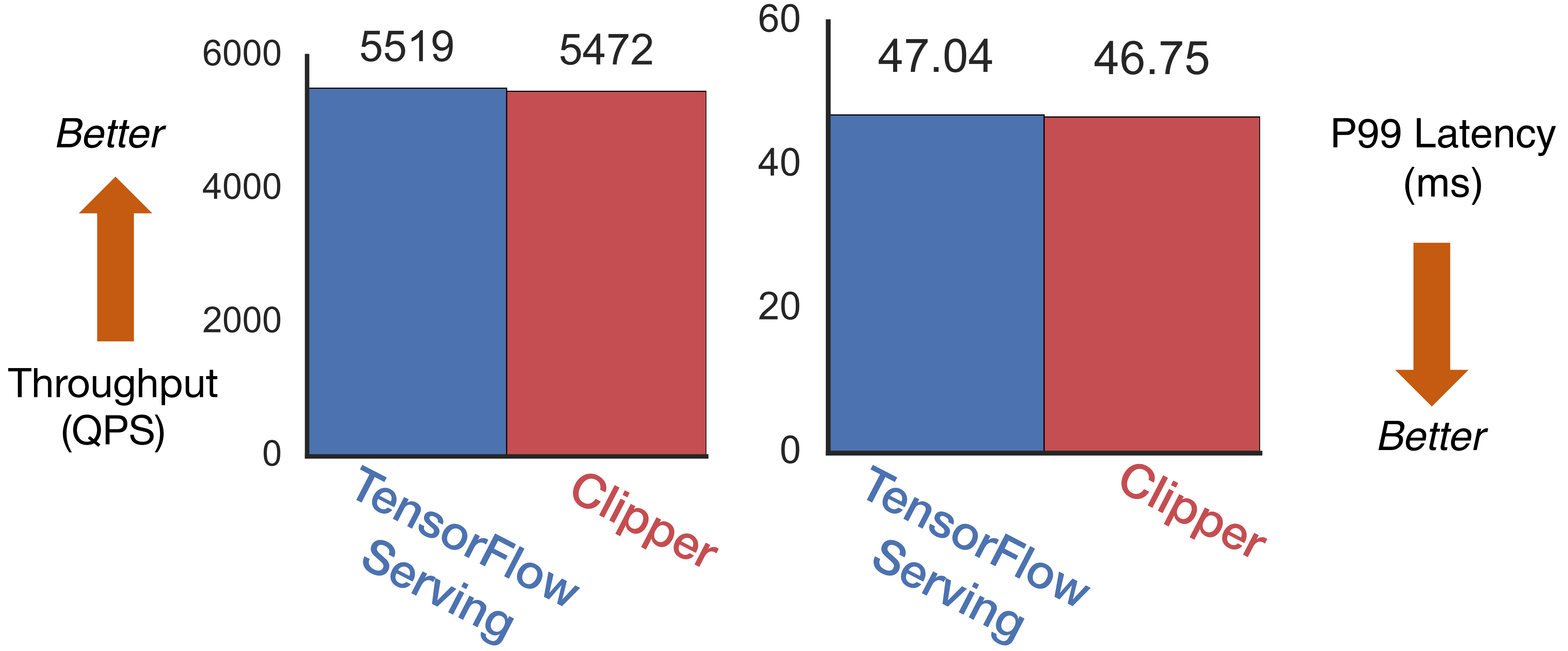


Overhead of decoupled architecture

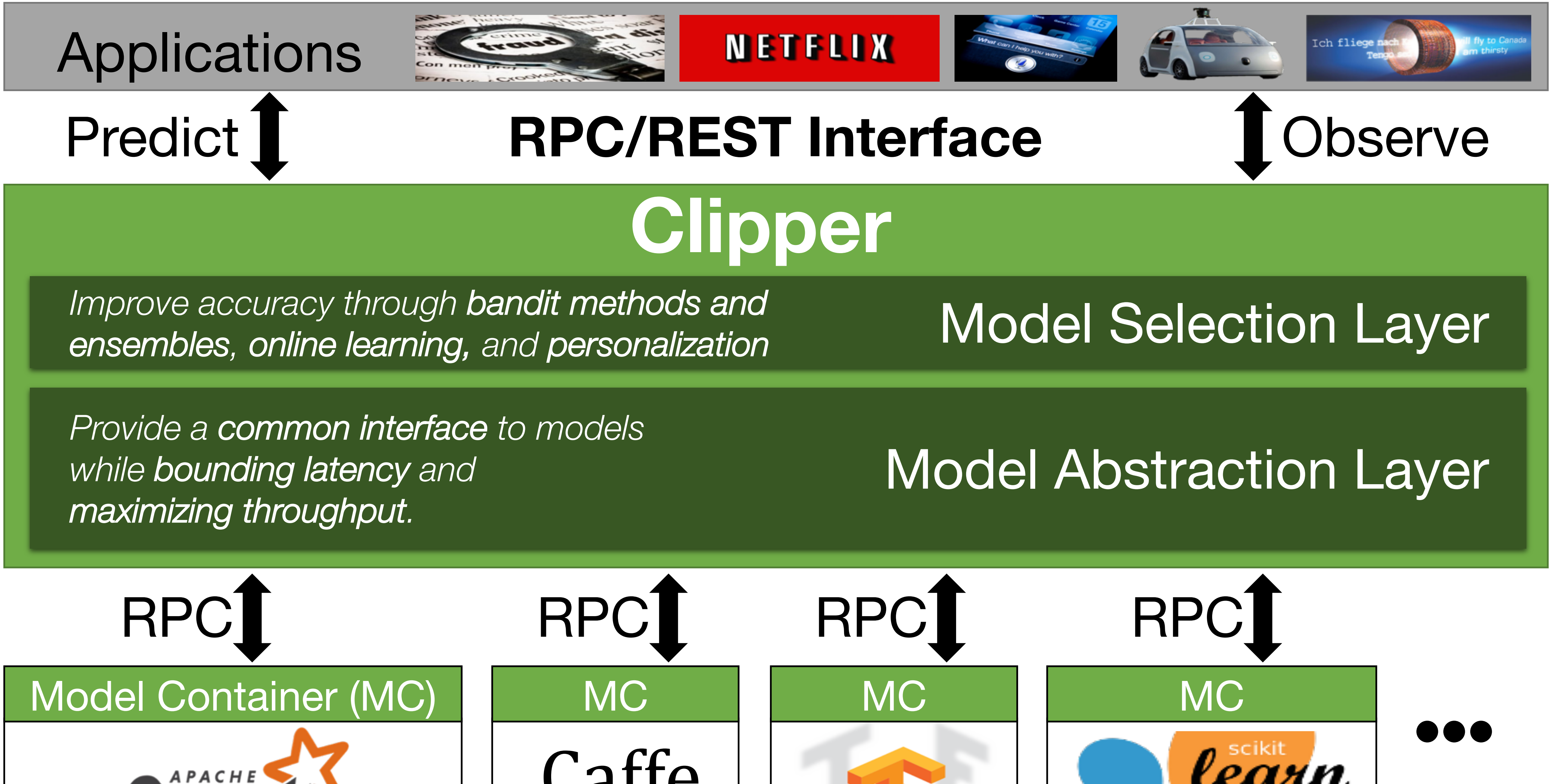


Overhead of decoupled architecture

Model: AlexNet trained on CIFAR-10

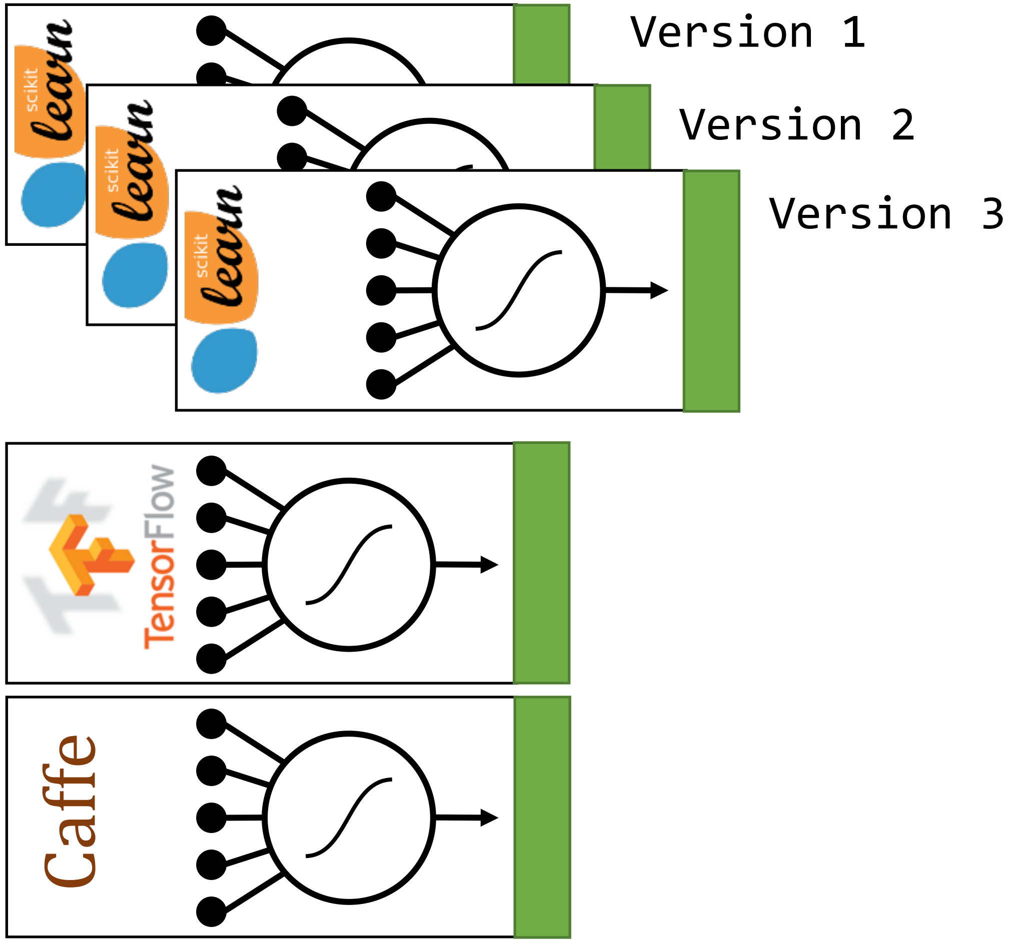


Clipper Architecture



*Improve accuracy through **bandit methods and ensembles, online learning,** and **personalization***

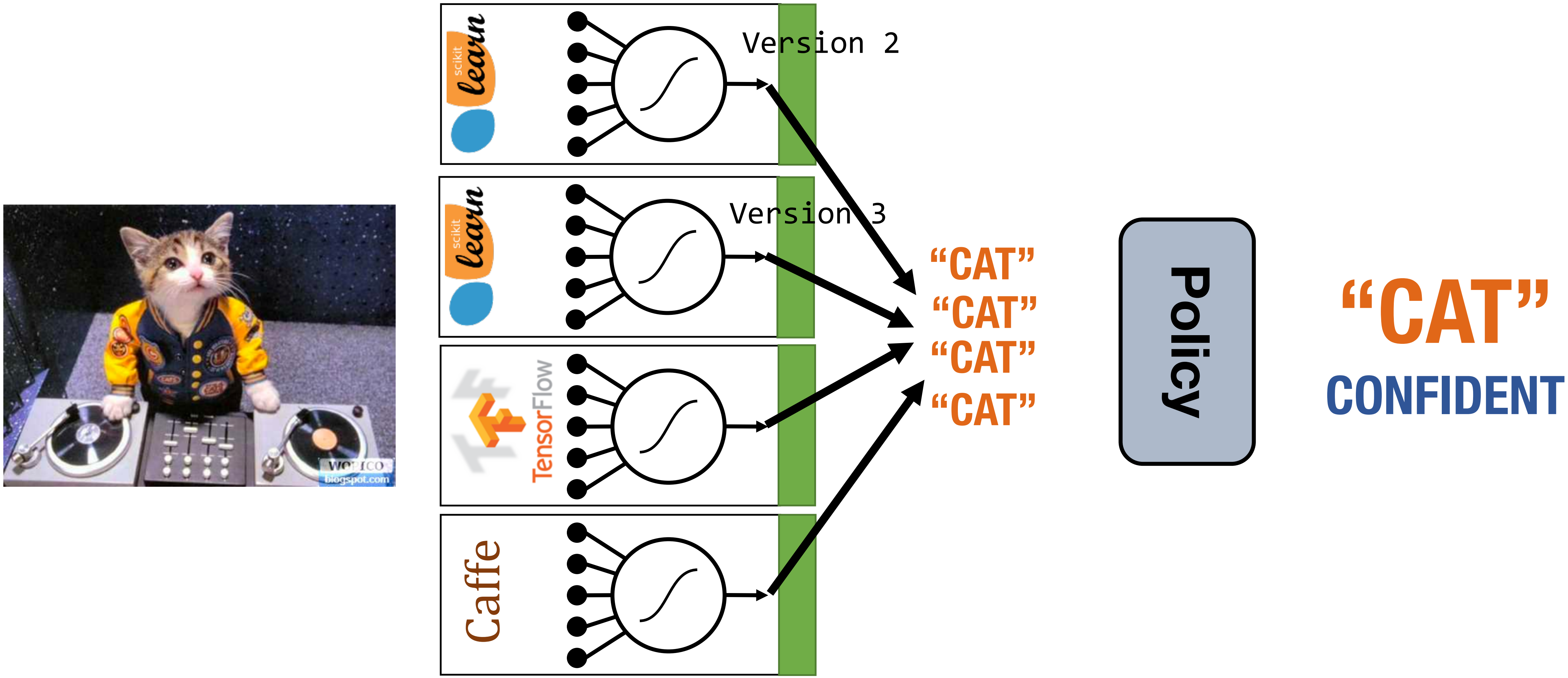
Model Selection Layer



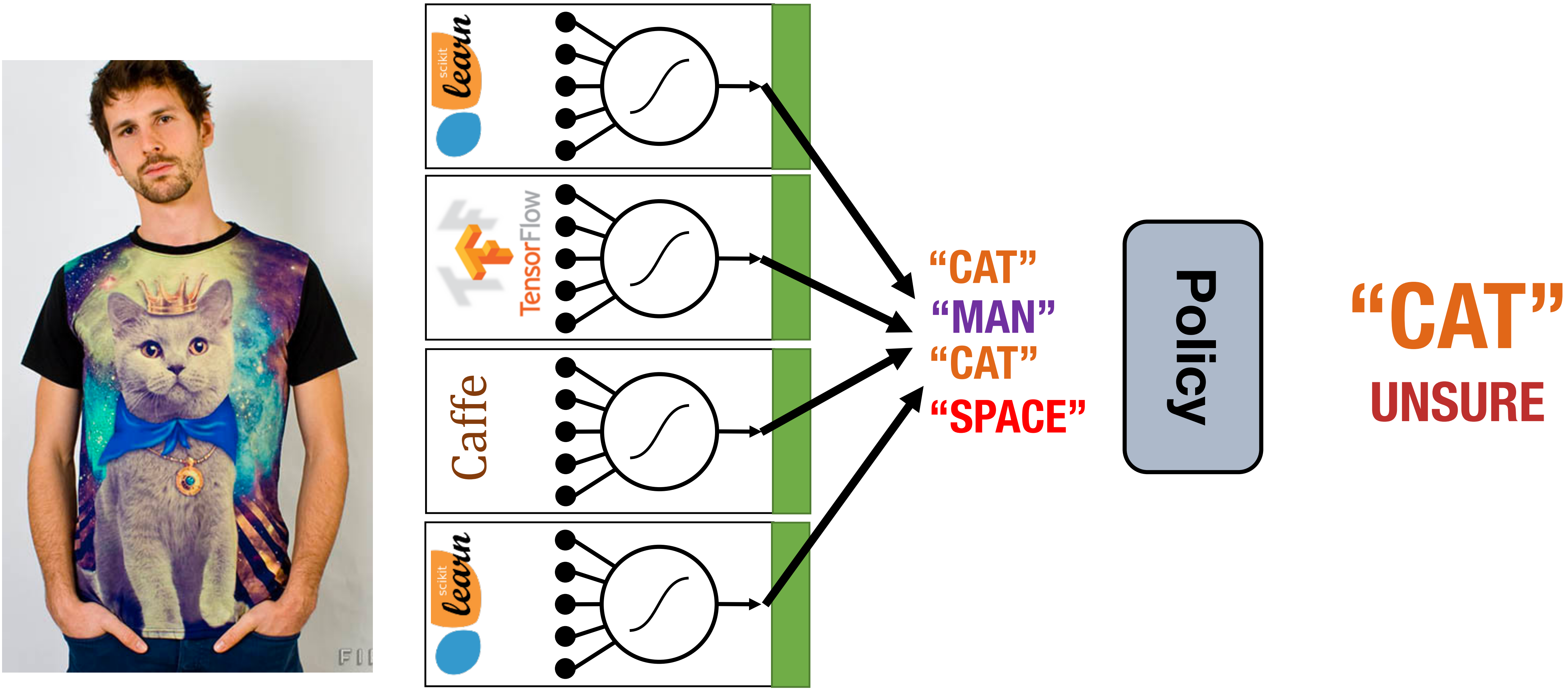
Periodic retraining

Experiment with new models and frameworks

Selection Policy: Estimate confidence

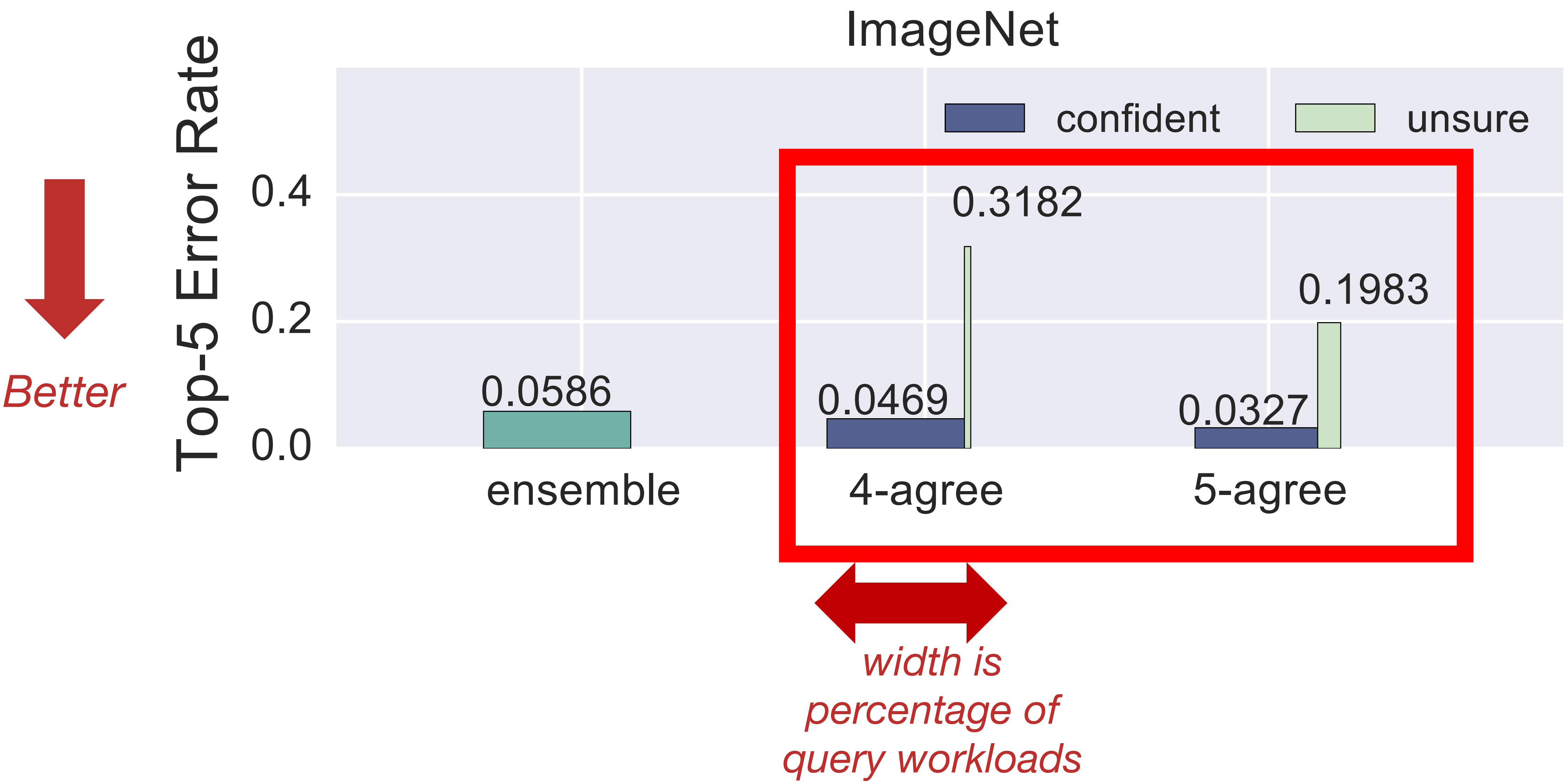


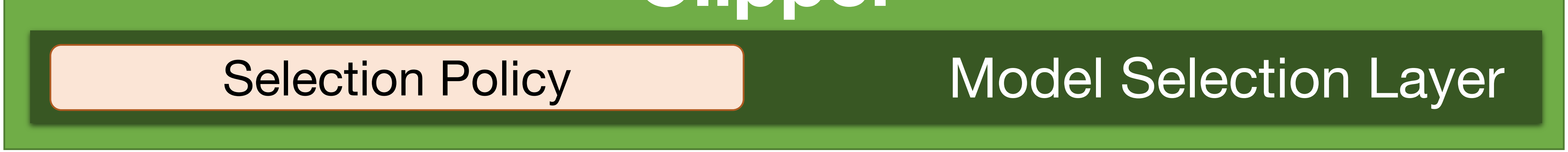
Selection Policy: Estimate confidence



Selection Policy: Estimate confidence

Selection Policy: Estimate confidence





Selection policies supported by Clipper

- Exploit multiple models to estimate confidence
- Use multi-armed bandit algorithms to learn optimal model-selection online
- Online personalization across ML frameworks

Online: Compute Predictions at Query Time

- **Examples**
  - Speech recognition, image tagging
  - Ad-targeting based on search terms, available ads, user features
- **Advantages**
  - Compute only necessary queries
  - Enables models to be changed rapidly and bandit exploration
  - Queries do not need to be from small ground set
- **Disadvantages**
  - Increases complexity and computation overhead of serving system
  - Requires low and predictable latency from models

Prediction Pipelines

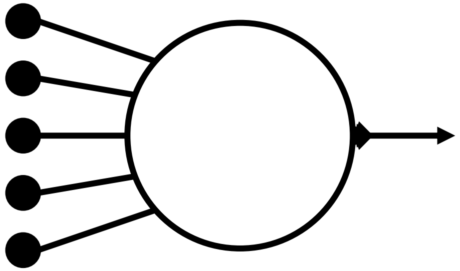
<u>Example</u>

Query Image

This is my daughter!

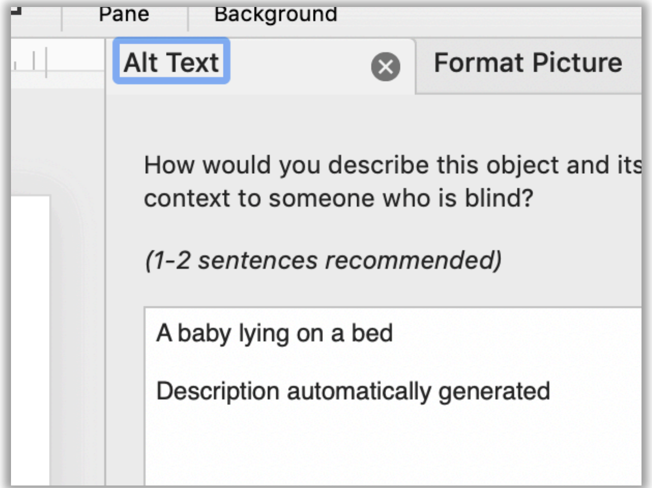
Query Image

Machine Learning Model



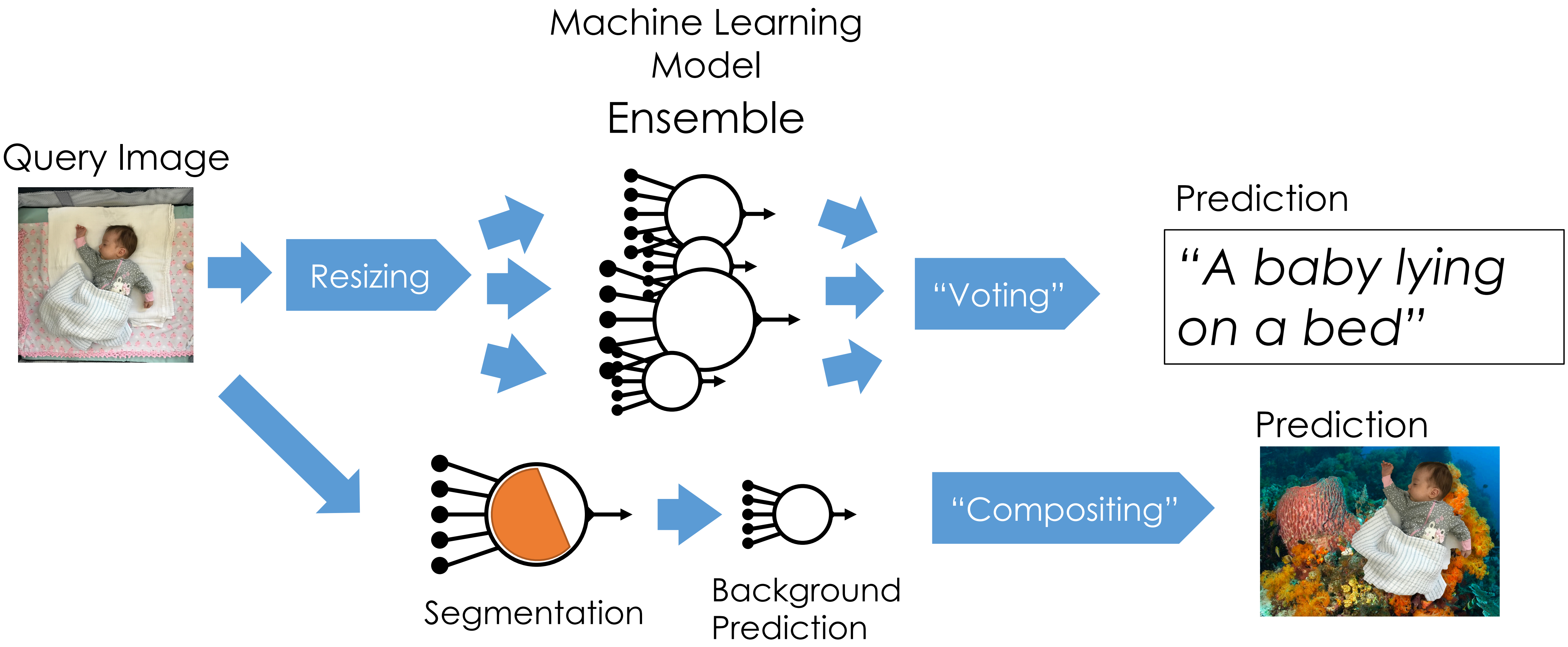
Prediction

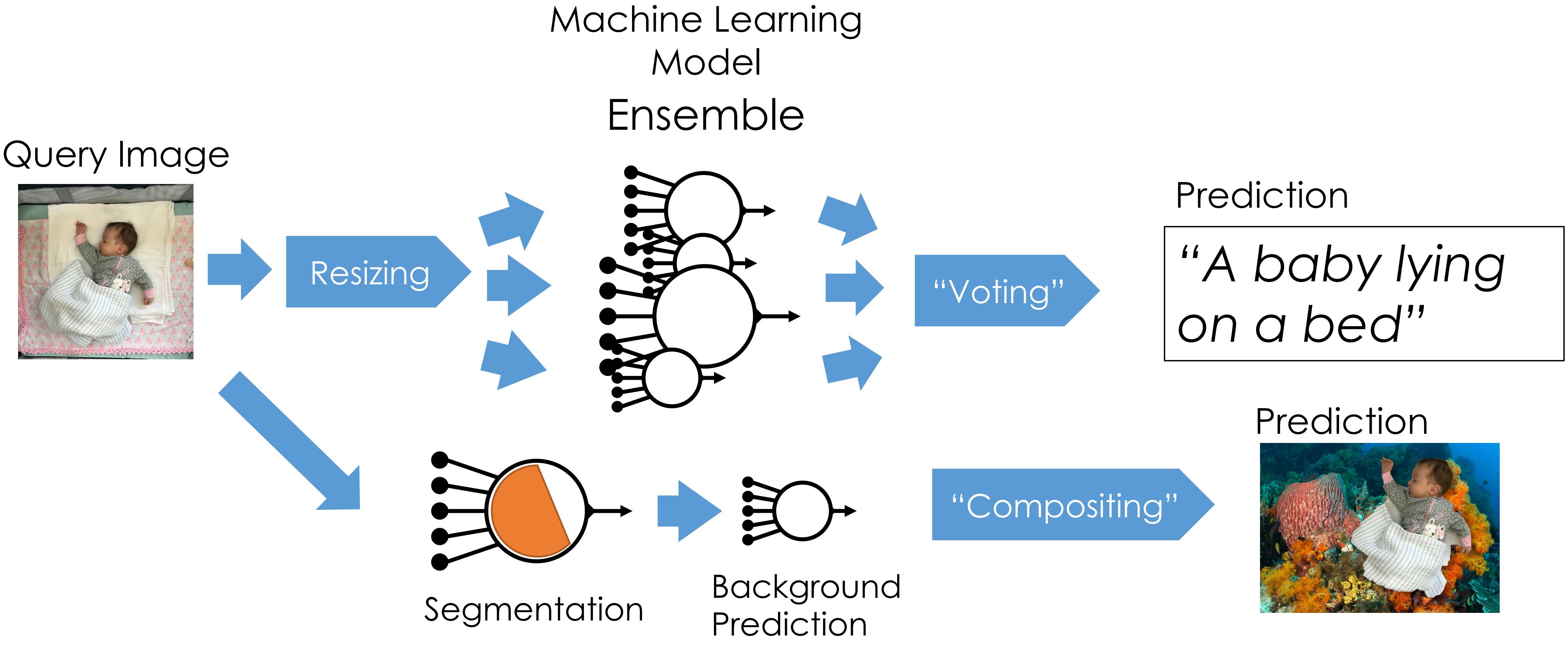
“A baby lying on a bed”



This caption was generated automatically
in the <u>cloud</u>
by Microsoft PowerPoint

More realistic → Prediction Pipeline



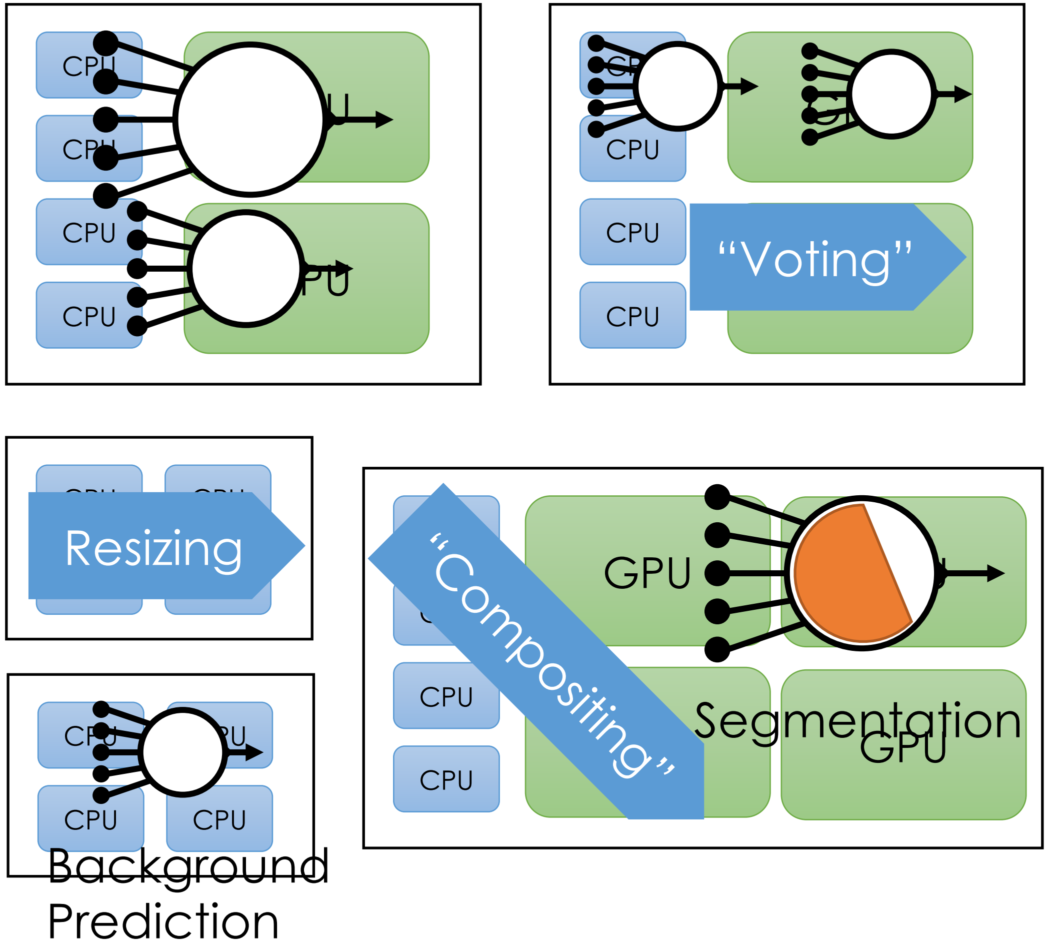


How do we provision resources for these pipelines?

Latency vs. **Throughput** vs. **Cost**

How do we provision resources for these pipelines?

Latency vs. Throughput vs. Cost



Two readings this week will address this problem.

- Pretzel
- InferLine

Cloud -- Edge

Example

KUNA

Home video security systems

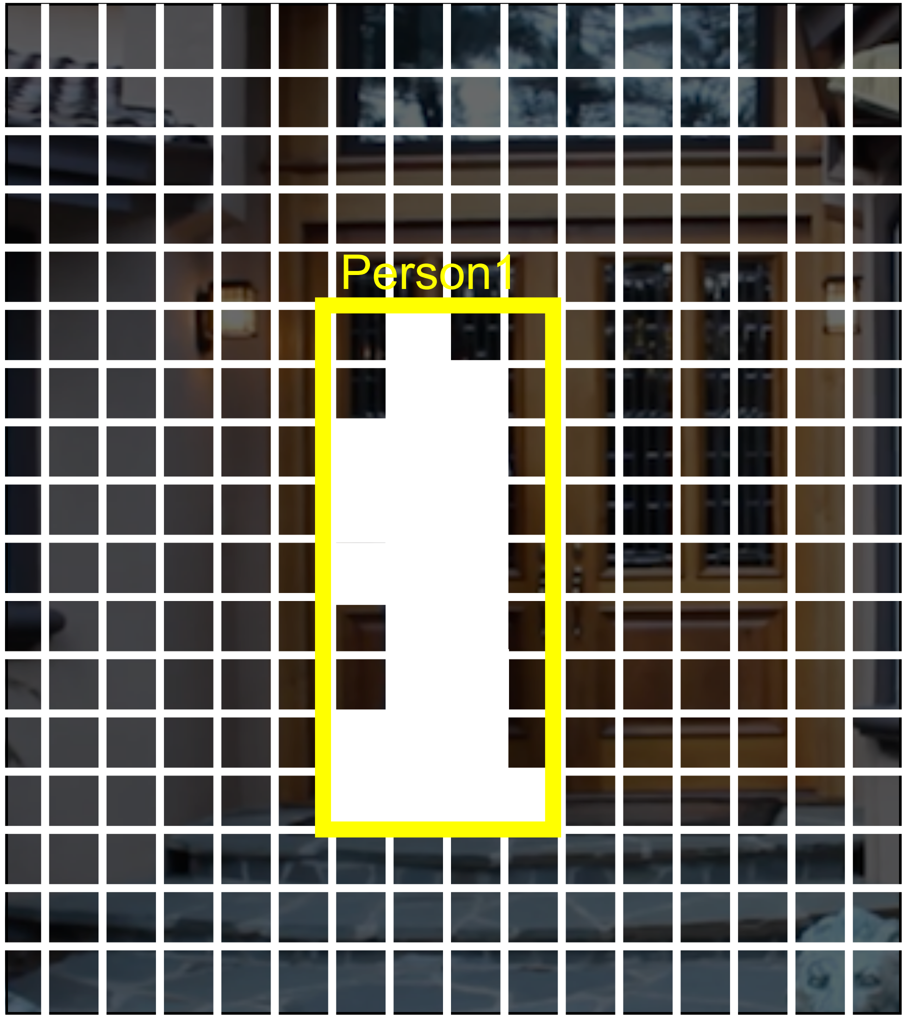
Technology

- AC Powered Lamp
- Commodity ARM proc.
- 720HD Video
- Microphone & Speaker
- Infrared Motion Sensors

Goals:

- Detect, identify, and record people
- Notify homeowner and open channel of comm.

How does KUNA work?



Fast onboard pixel-level filter identifies suspicious change

Key frames are sent to EC2 for further processing

More sophisticated processing to reduce false positives (**costly GPU time**)

KUNA technology challenges

- Splitting classification across **device** and **cloud**.
- **Shared learning** to identify common patterns
  - e.g., traffic in urban environments
- More **efficient prediction rendering** on cloud + edge
  - Running full CV pipeline on all images is very costly

Desired Capabilities

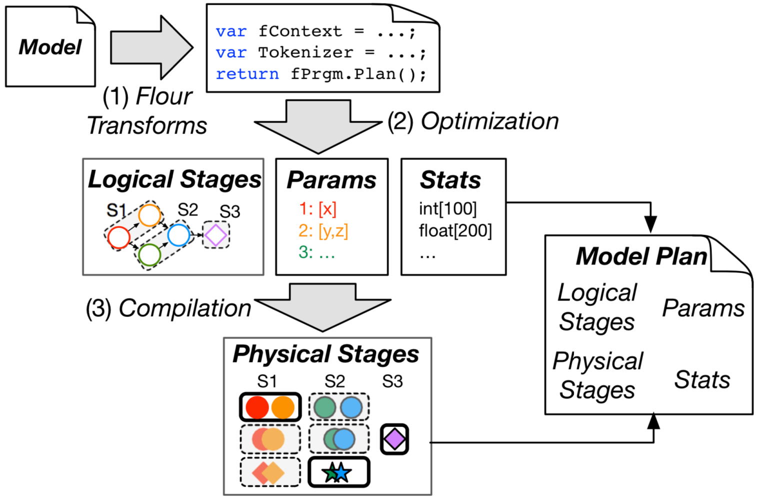
- Event characterization: "*Package delivery at 1:33 PM*"
- Automatic user interaction: "*I would be happy to digitally sign for the package …*"

Reading This Week

Reading for the Week

- Pretzel: Opening the Black Box of Machine Learning Prediction Serving Systems (OSDI'18)
  - Optimizing prediction serving pipeline using compiler and database system techniques
- InferLine: ML Inference Pipeline Composition Framework (pre-print)
  - Optimizing prediction serving pipeline configurations for deep learning on heterogenous hardware
- Focus: Querying Large Video Datasets with Low Latency and Low Cost (OSDI'18)
  - Enabling real-time queries on video data with offline pre-processing

Pretzel: Opening the Black Box of Machine Learning Prediction Serving Systems



- Addresses a range of **practical issues:**
  - Dealing with infrequently used models
    - Need to **"page-out"** infrequently used models → **Cold starts**
    - Need to **pack** many models in same machine
  - **Sharing model stages** across prediction pipelines
    - Eliminate **redundant computation** and **memory requirements**
  - Pushing **computation (reuse)** through feature concatenation
  - Generating efficient **binary executables** from high-level DSLs
- **Setting:** Focused on **non-deep learning** pipelines
  - CPU Intensive
  - ML.Net Infrastructure and production workloads

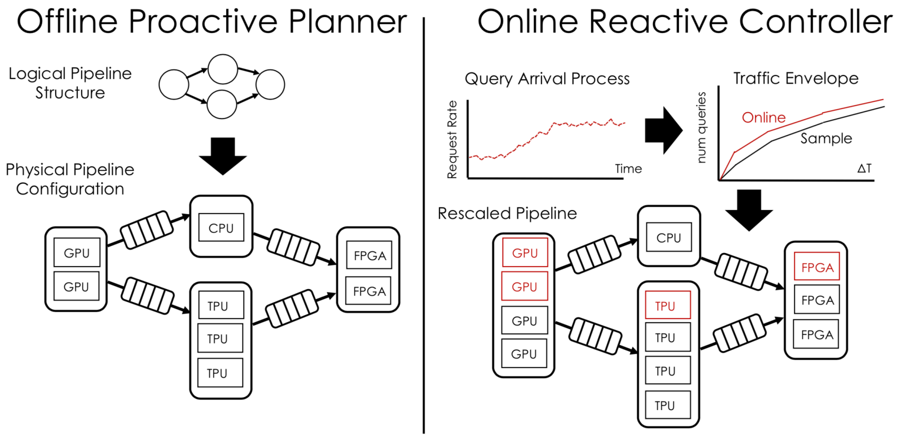
- **Sharing model stages** across prediction pipelines
  - Eliminate **redundant computation** and **memory requirements**
- Pushing **computation (reuse)** through feature concatenation
- Generating efficient **binary executables** from high-level DSLs

- **Setting:** Focused on **non-deep learning** pipelines
  - CPU Intensive
  - ML.Net Infrastructure and production workloads

- **Big Idea:** Leverage visibility into pipeline achieved by high-level pipeline DSL to optimize execution across pipelines and stages within a pipeline

- **What to look for in reading**
  - Motivations driven by real-world **workload profiling**
  - Combination of **offline** and **online** optimization
  - Decomposition of problem into **logical** and physical plans and **runtime scheduling**

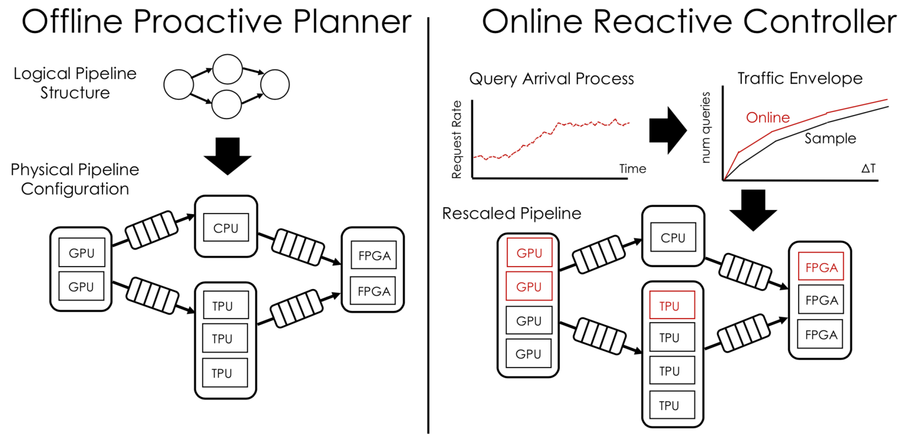
InferLine: Prediction Pipeline Provisioning and Management for Tight Latency Objectives



- **Context:** This is a pre-print paper from **my group**
  - Submitted OSDI'18 and SOSP'19 → rejected ☹
  - Issues with presentation and contributions
  - How can we review a professor's paper?
    - Your honest feedback is very helpful!

- Why choose this paper?
  - Discusses a range of challenges in **black-box pipeline** management
  - Presents interesting **configuration space**
  - **We need feedback!** *(It is ok to be negative.)*

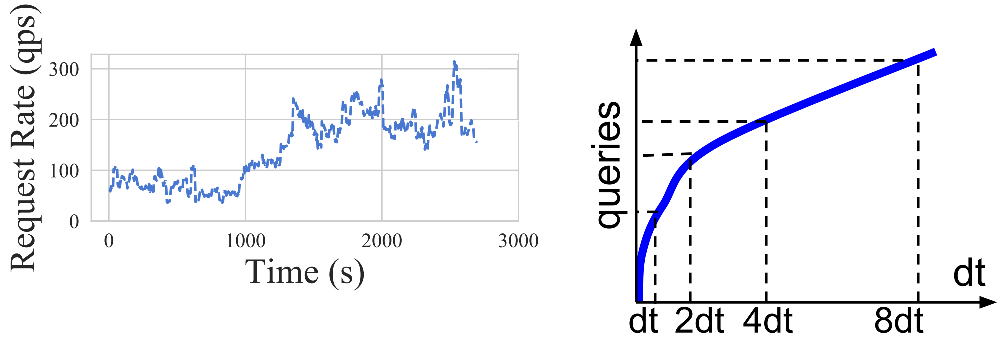
InferLine: Prediction Pipeline Provisioning and Management for Tight Latency Objectives



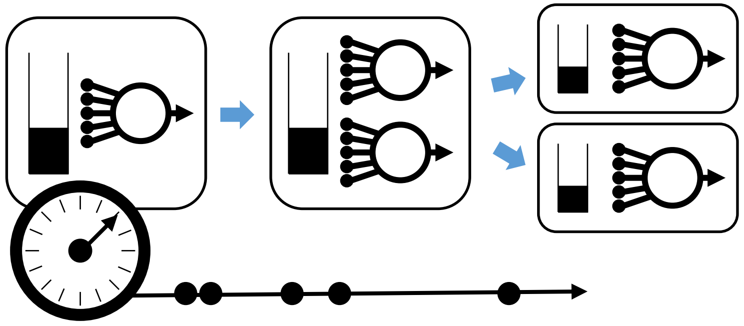
- **Big Idea:** Optimally configure per-model parameters in a prediction pipeline to achieve probabilistically bounded tail latency at minimal cost.

Technical Ideas in the InferLine Project

Arrival Process Characterization

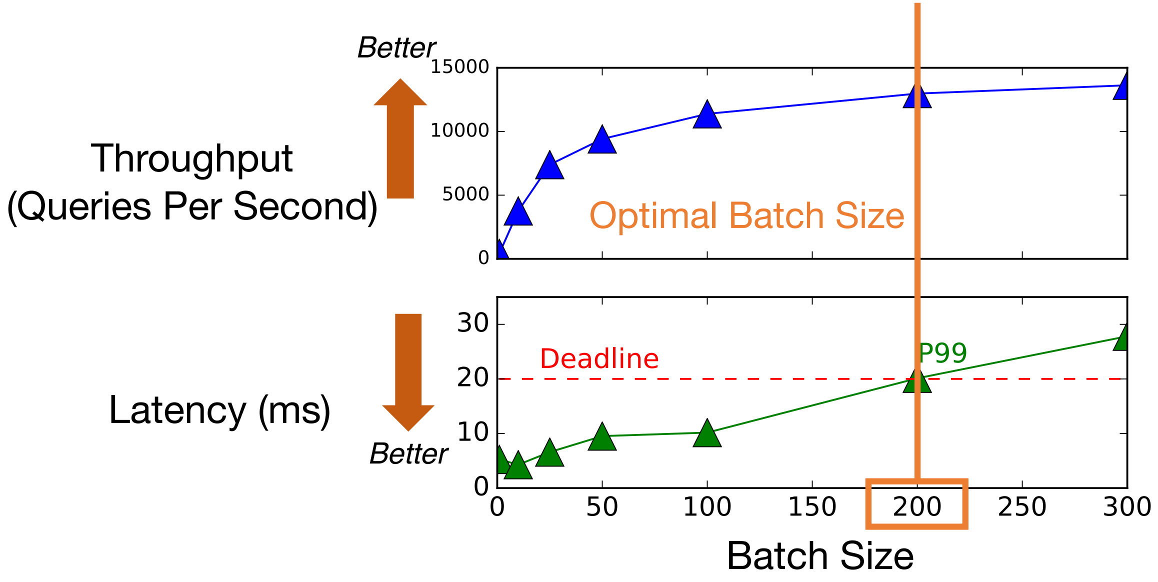


Discrete Event Continuous Time Simulator

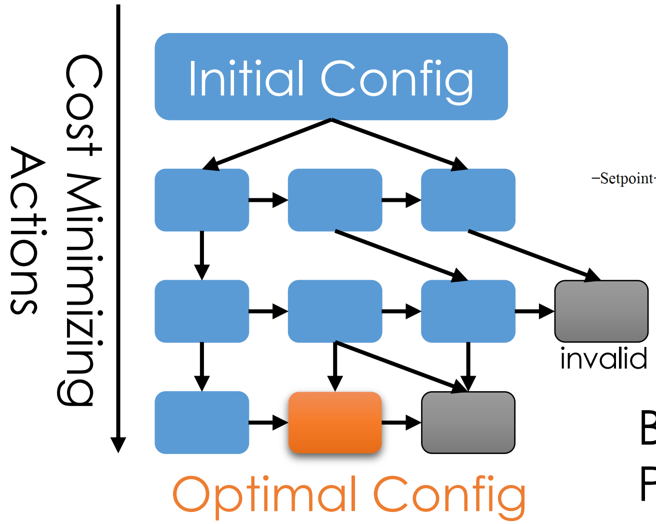


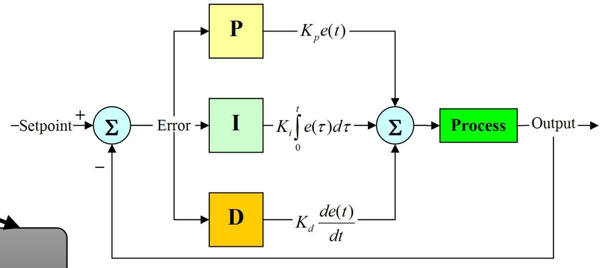
+ Network Calculus

Individual Model Profiles



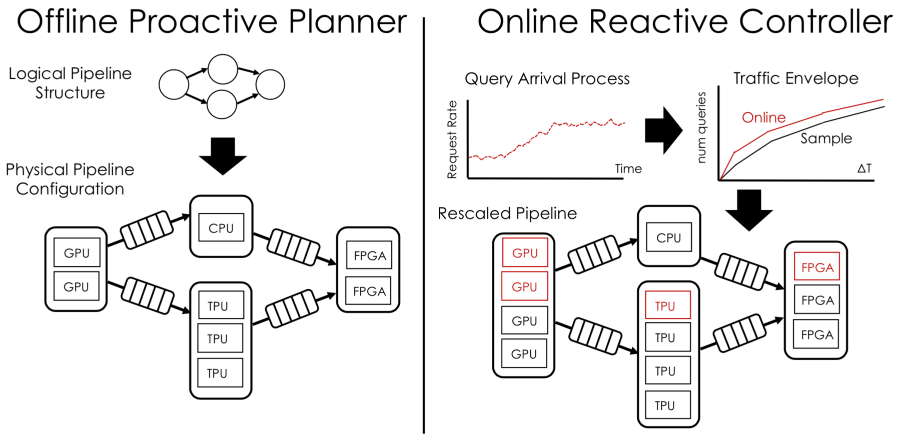
Proactive and Reactive Optimizer





Bang Bang &
PID Reactive Controller

InferLine: Prediction Pipeline Provisioning and Management for Tight Latency Objectives



- **Technical Ideas (summary)**
  - Individual model **performance profiles** + **discrete event simulation** → reason about **end-to-end latency** in the presence of complex **queuing behavior**
  - Simple **greedy search heuristic** to configure for each model:
    - Hardware type, number of copies, and batching parameters
  - Online re-provisioning using **network calculus** to **optimally resize**

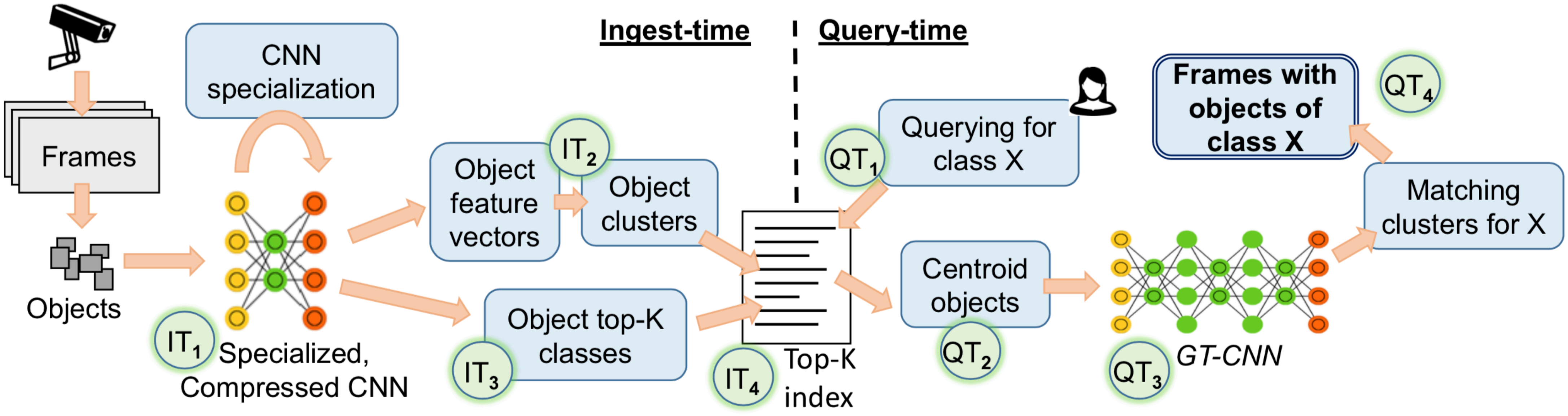
- **What to look for:**
  - **Too many ideas …. not enough contribution?**
  - Clarity of presentation

Focus: Querying Large Video Datasets with Low Latency and Low Cost

- **Context:** builds on a line of earlier work
  - Live Video Analytics at Scale with Approximation and Delay-Tolerance (NSDI'17)
  - Chameleon: Scalable Adaptation of Video Analytics (SIGCOMM'18)
- **Big Ideas in the Line of Work:**
  - **Trade-off accuracy and latency** in video processing tasks
  - Schedule resources according to **acc.** and **latency requirements**
- **Different "Serving Model":**
  - Large queries on historical or video streams:
    - *Find all the times where a car and a bike are in a frame.*

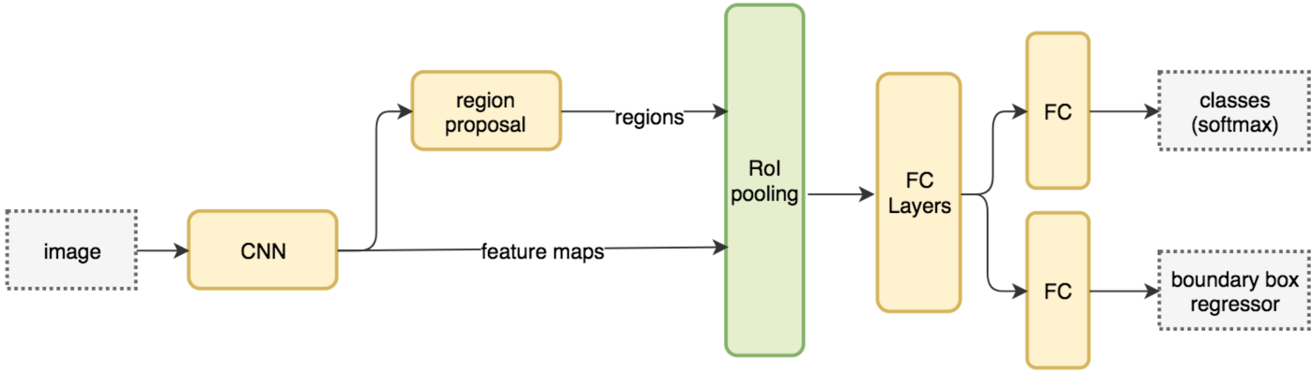
Focus: Querying Large Video Datasets with Low Latency and Low Cost

➢ **Big Idea(s)**



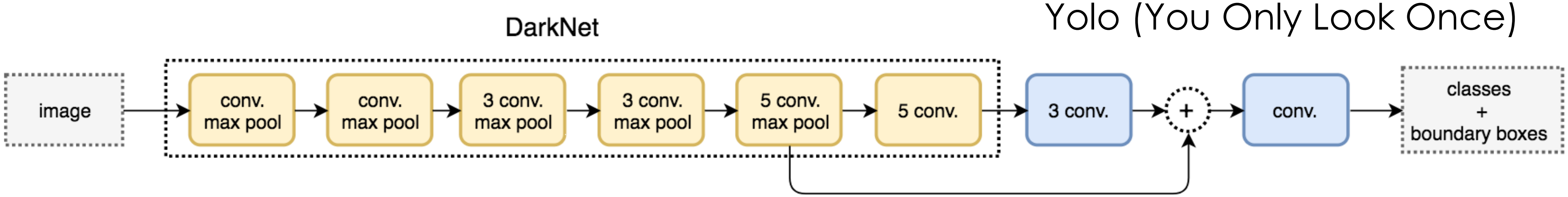
Focus: Querying Large Video Datasets with Low Latency and Low Cost

- **What to look for:**
  - Framing of relationships between object detection models and image classification



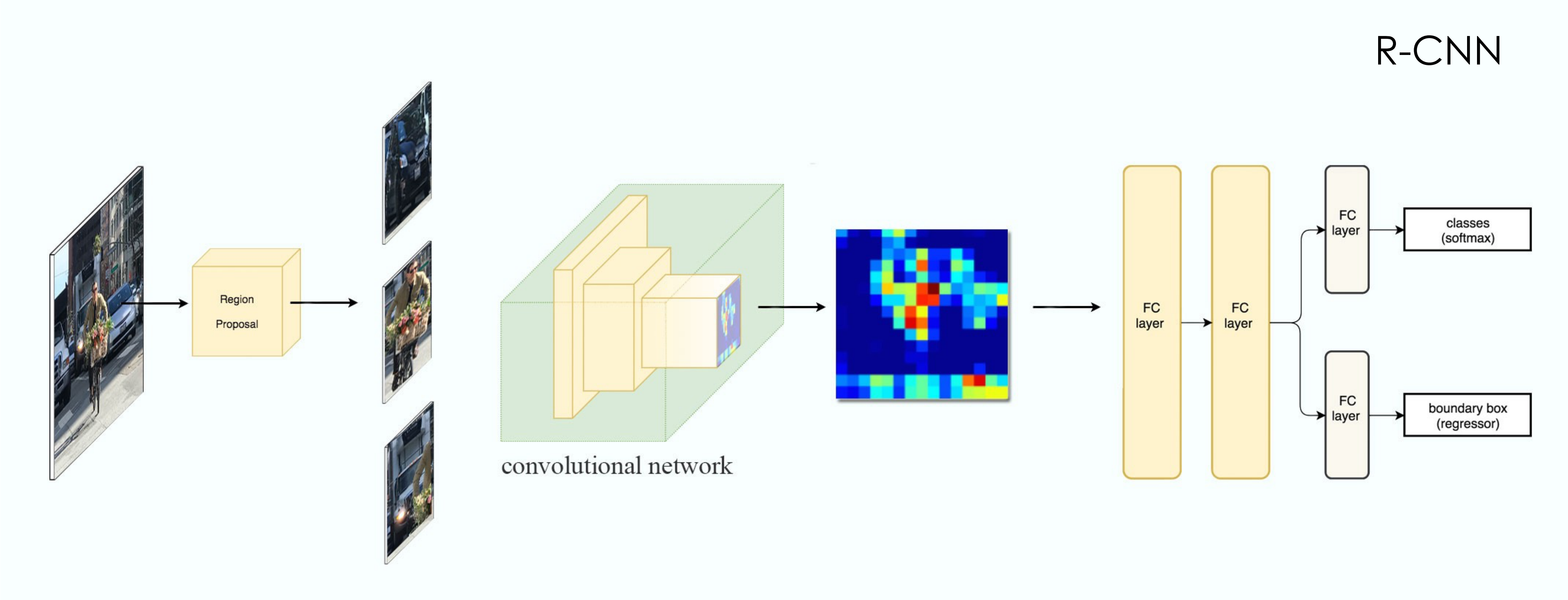
Faster R-CNN

Yolo (You Only Look Once)



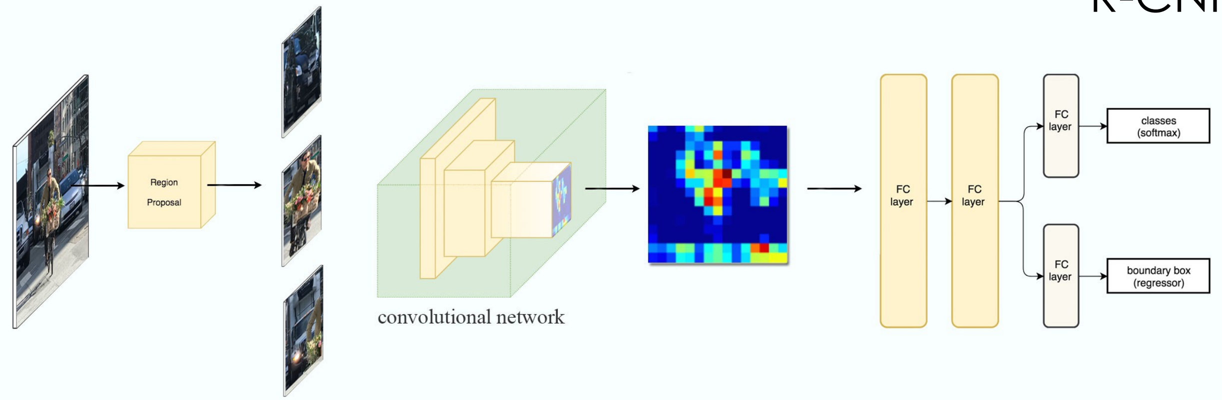
Focus: Querying Large Video Datasets with Low Latency and Low Cost

- **What to look for:**
  - Framing of relationships between object detection models and image classification



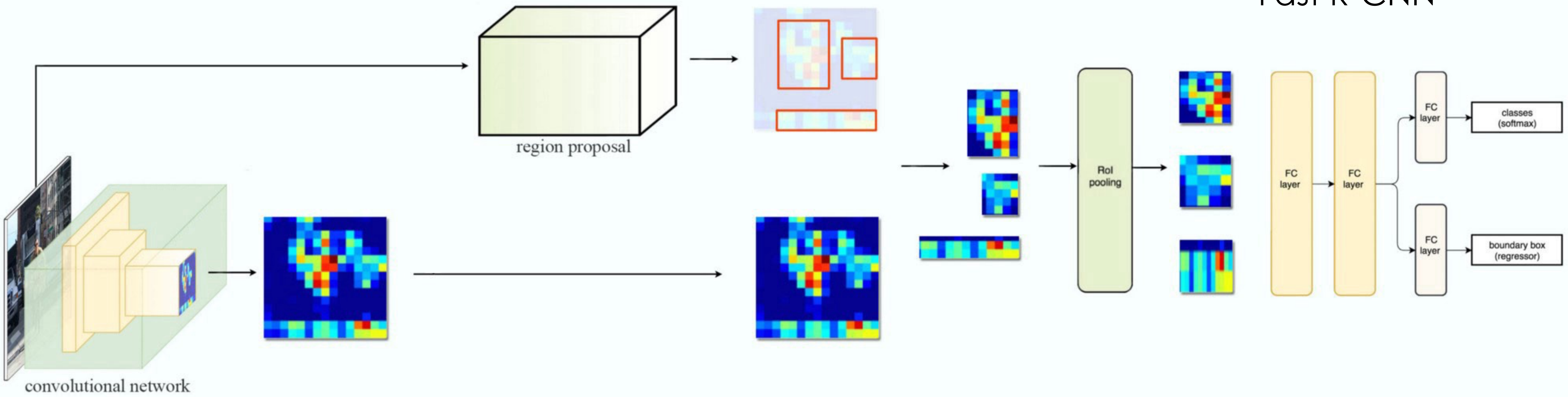
https://medium.com/@jonathan_hui/what-do-we-learn-from-region-based-object-detectors-faster-r-cnn-r-fcn-fpn-7e354377a7c9

R-CNN



Identify regions of interest and run feature network on each.

Fast R-CNN

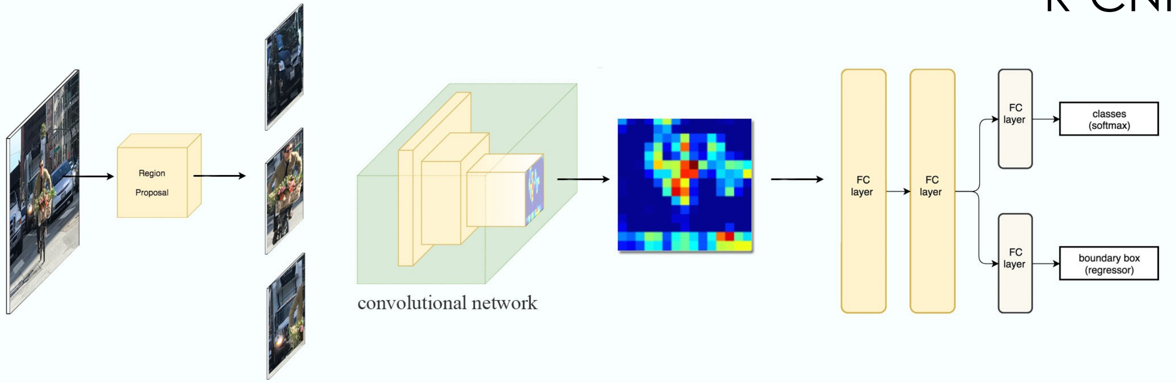


Reuse feature outputs for all externally proposed regions.

https://medium.com/@jonathan_hui/what-do-we-learn-from-region-based-object-detectors-faster-r-cnn-r-fcn-fpn-7e354377a7c9

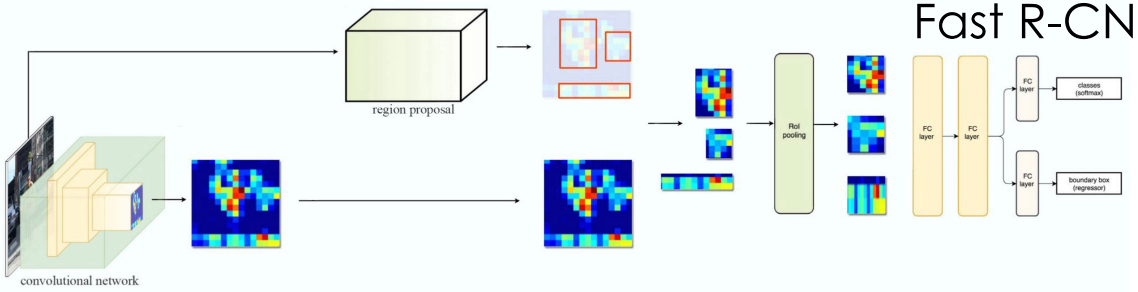
Identify regions of interest and run feature network on each.

R-CNN



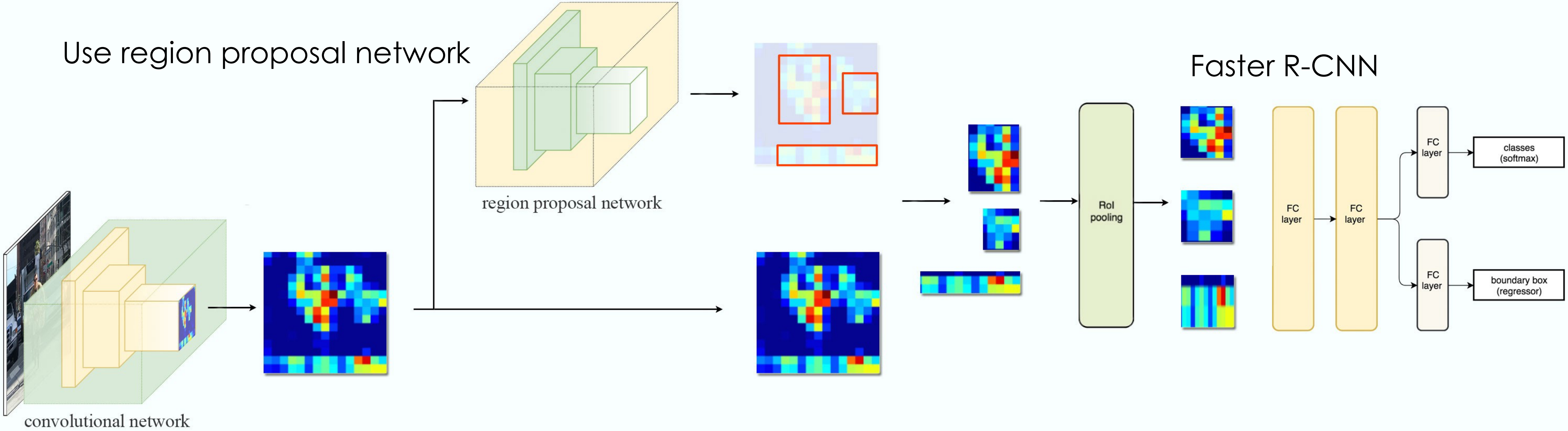
Reuse feature outputs for all externally proposed regions.

Fast R-CNN



Use region proposal network

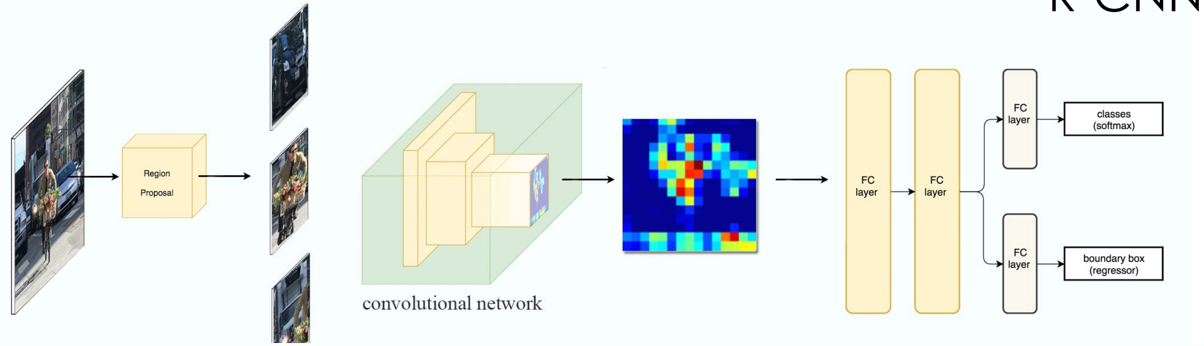
Faster R-CNN



https://medium.com/@jonathan_hui/what-do-we-learn-from-region-based-object-detectors-faster-r-cnn-r-fcn-fpn-7e354377a7c9

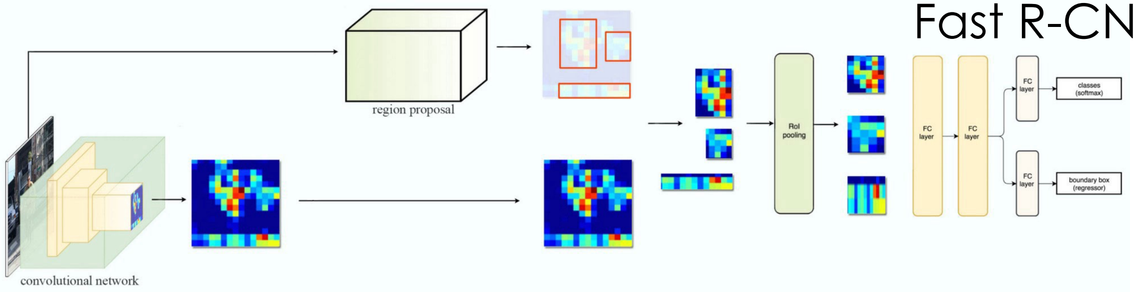
Identify regions of interest and run feature network on each.

R-CNN



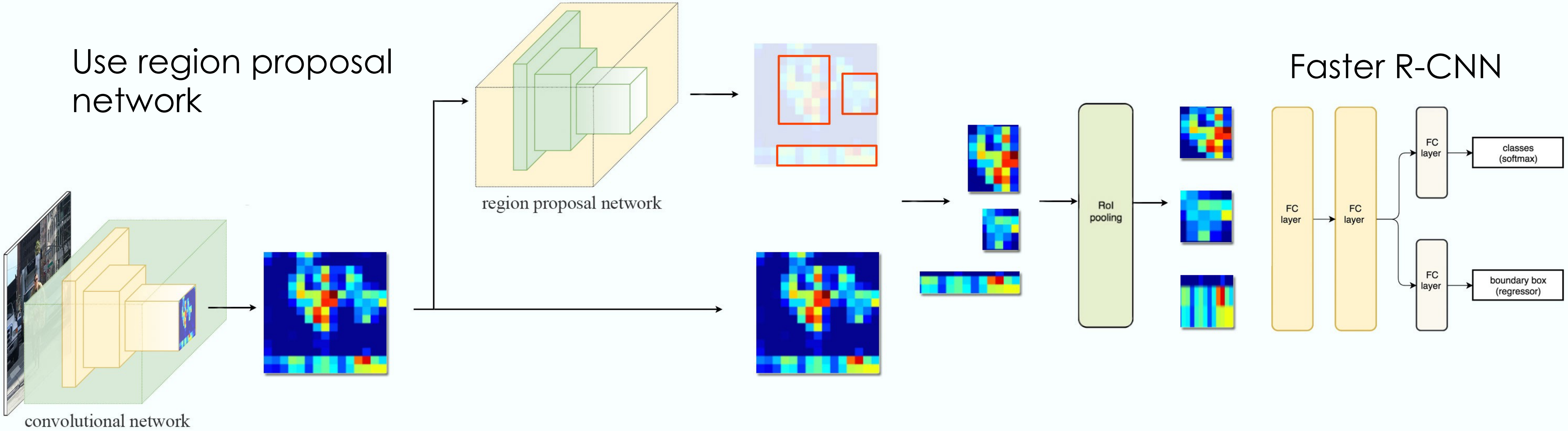
Reuse feature outputs for all externally proposed regions.

Fast R-CNN



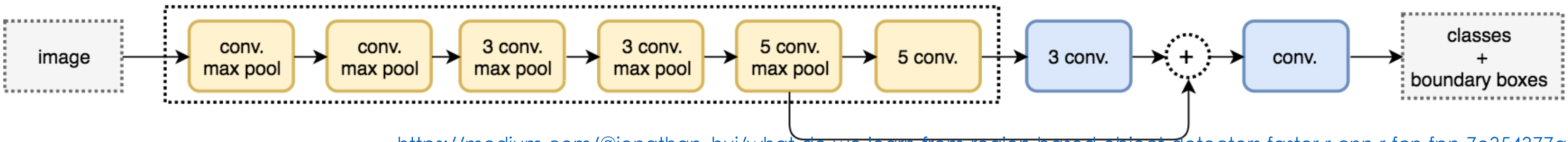
Use region proposal network

Faster R-CNN



DarkNet

Yolo (you only look once)



https://medium.com/@jonathan_hui/what-do-we-learn-from-region-based-object-detectors-faster-r-cnn-r-fcn-fpn-7e354377a7c9

Focus: Querying Large Video Datasets with Low Latency and Low Cost

- **What to look for:**
  - Framing of relationships between object detection models and image classification
    - Do they leverage the structure of object detection models?
  - Split between ingest and query time computation
  - Tradeoff between accuracy and latency → how is it evaluated?
  - Presentation → many optimizations explored, do they fit together?

Done!

Cascaded Predictions

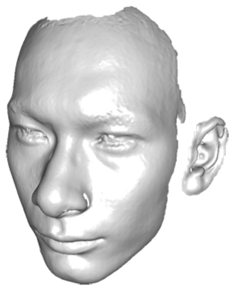
IDK Prediction Cascades

Simple models for simple tasks

Xin Wang

Yika Luo

Zi-Yi Duo



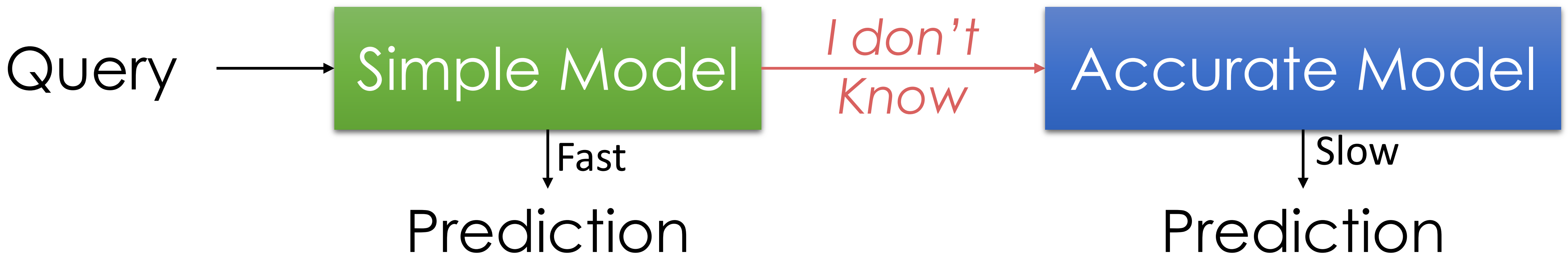
Fisher Yu

Query → Simple Model → *I don't Know* → Accurate Model

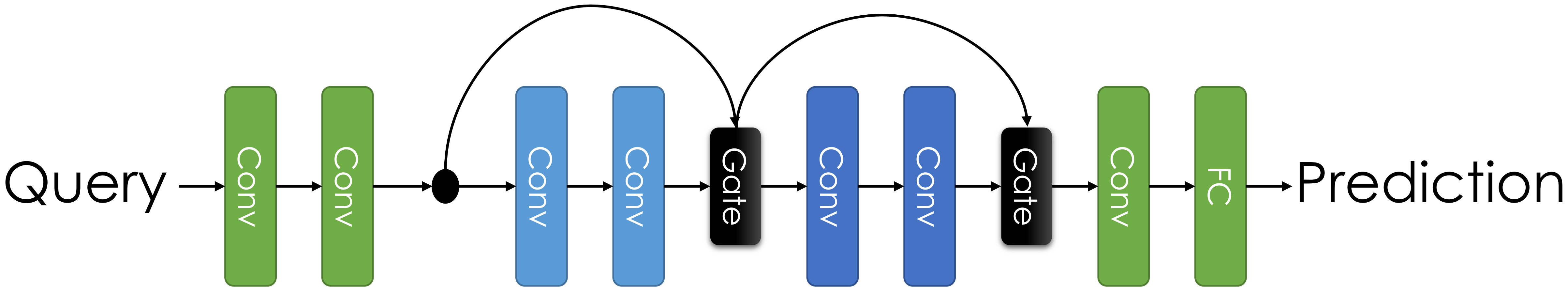
Simple Model → Fast → Prediction

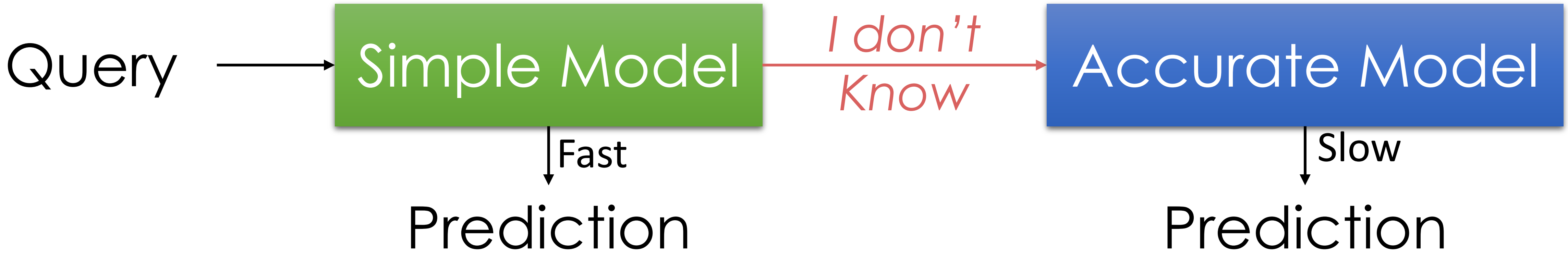
Accurate Model → Slow → Prediction

Learn to combine **fast (inaccurate) models** with **slow (accurate) models** to maximize accuracy while reducing computational costs.

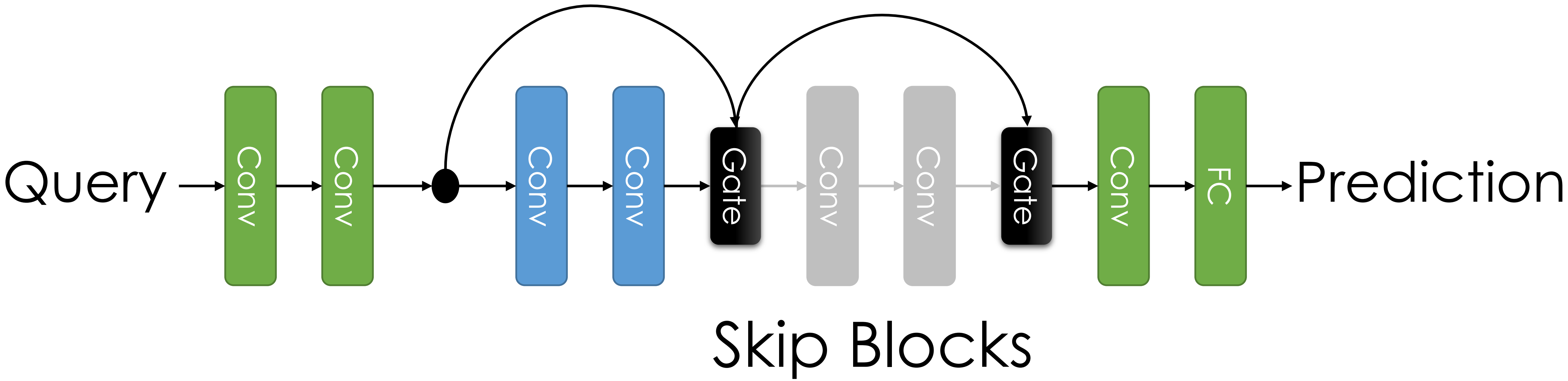


SkipNet: dynamic execution within a model

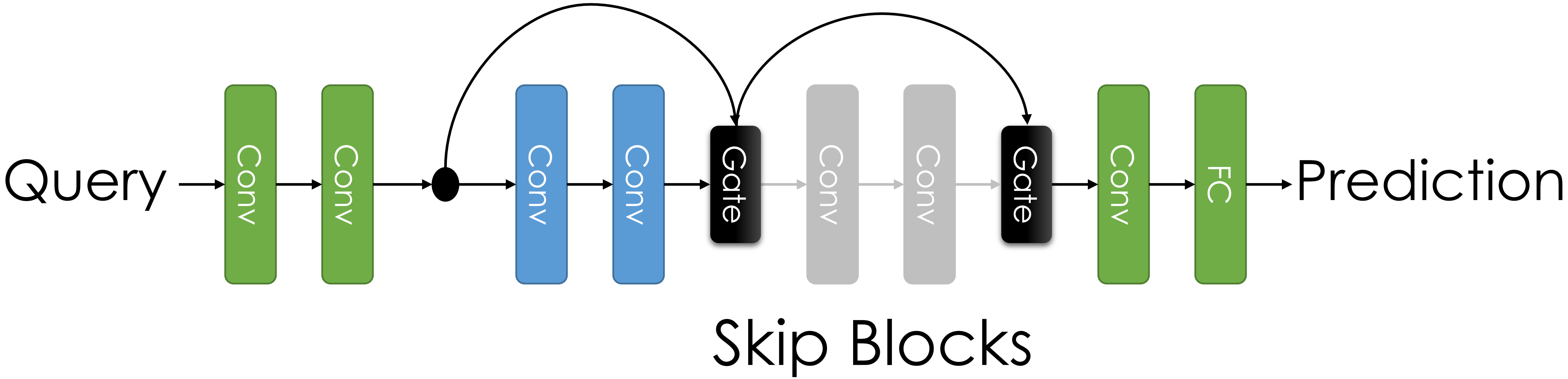




SkipNet: dynamic execution within a model

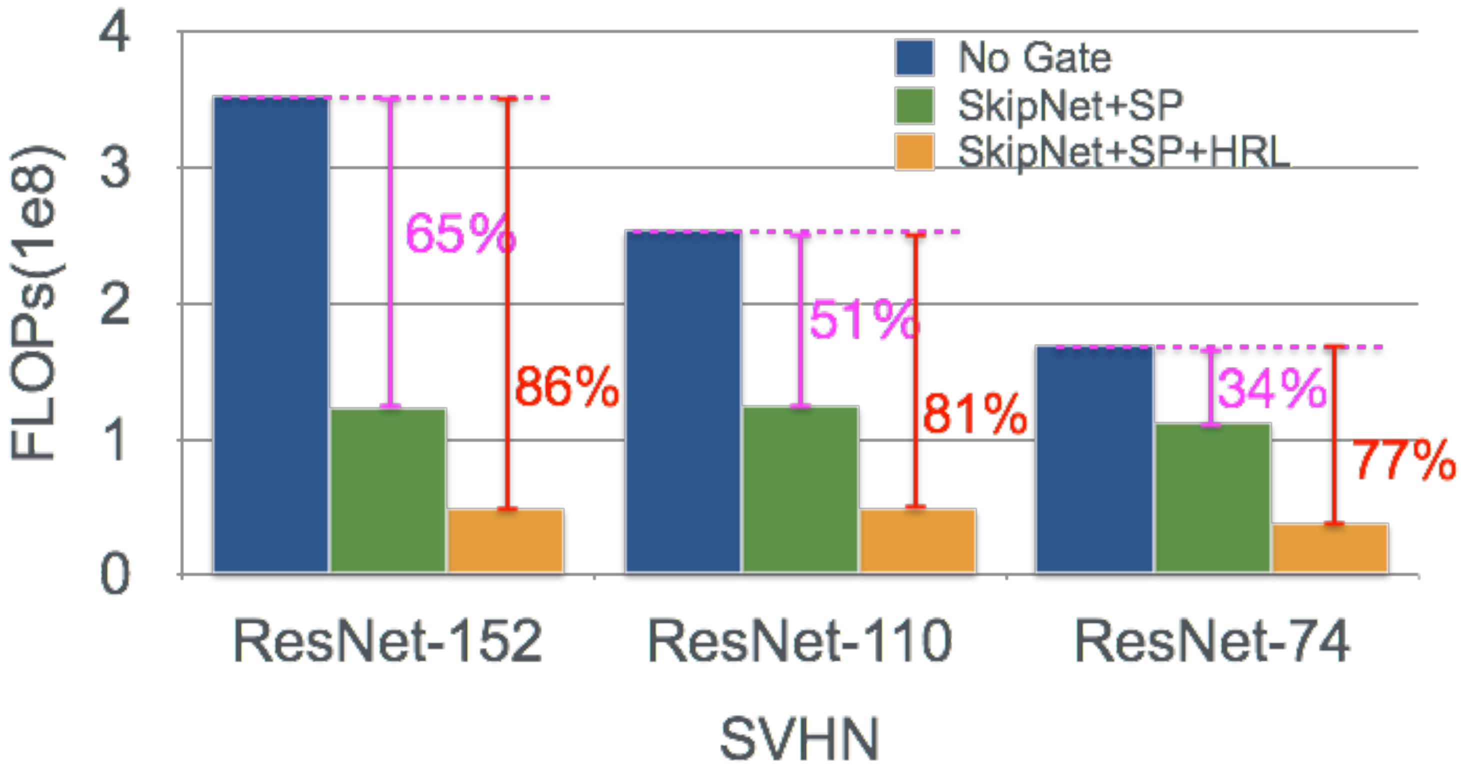


SkipNet: dynamic execution within a model



- Combine **reinforcement learning** with **supervised pre-training** to learn a gating policy

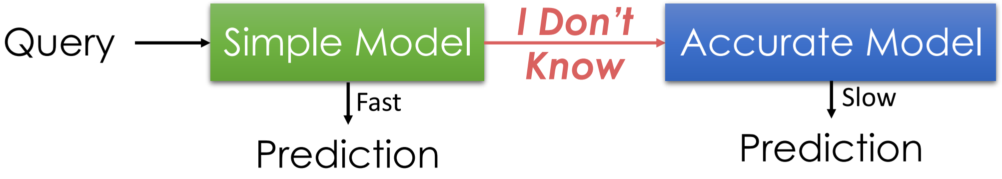
SkipNet Performance

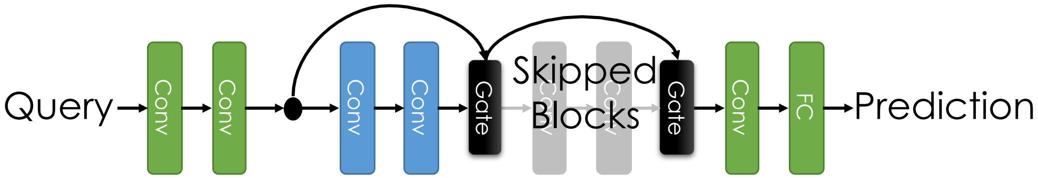


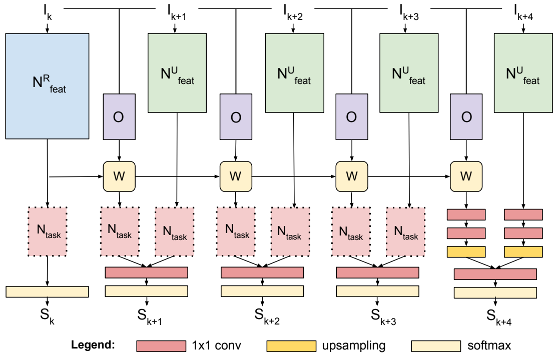
Easy Images
Skip **Many** Layers

Hard Images
Skip **Few** Layers

Efficient Neural Networks

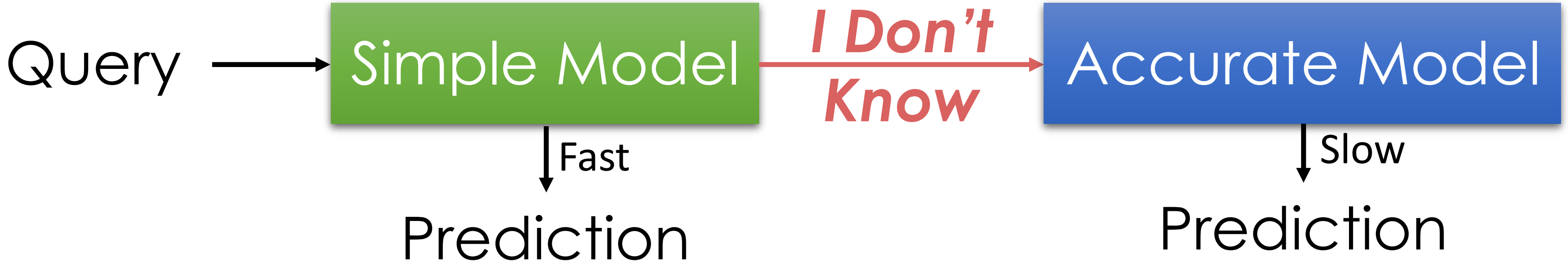




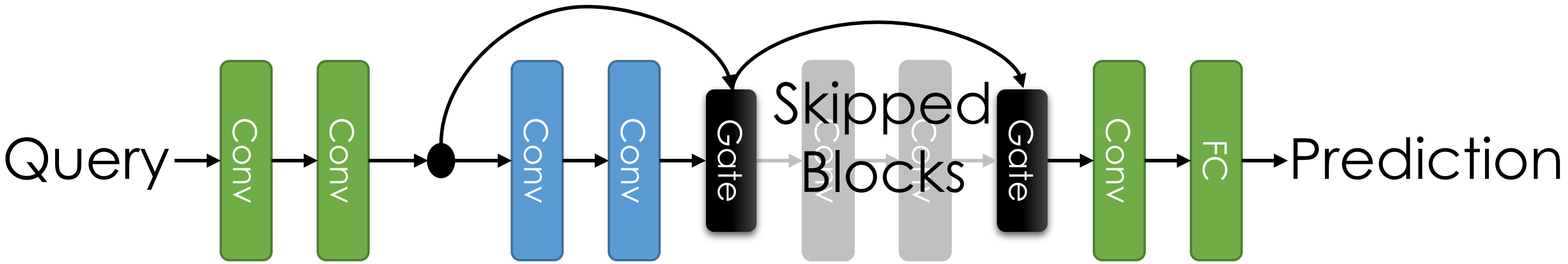


Dynamic Networks for **fast** and **accurate** inference

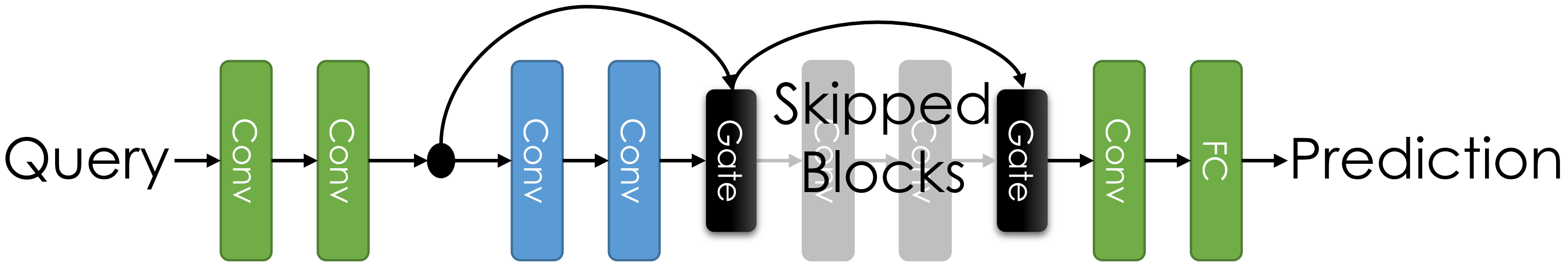
IDK Cascades: Using the fastest model possible [UAI'18]



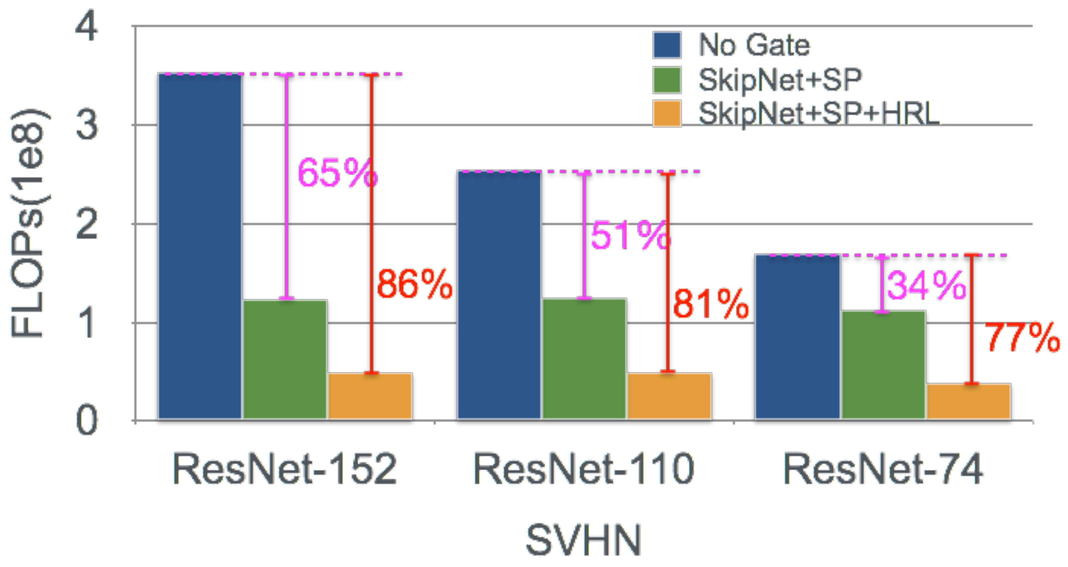
SkipNet: dynamic execution within a model [ECCV'18]



SkipNet: dynamic execution within a model [ECCV'18]

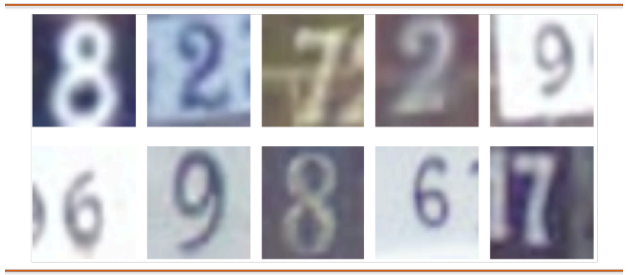


Large Reductions in FLOPS



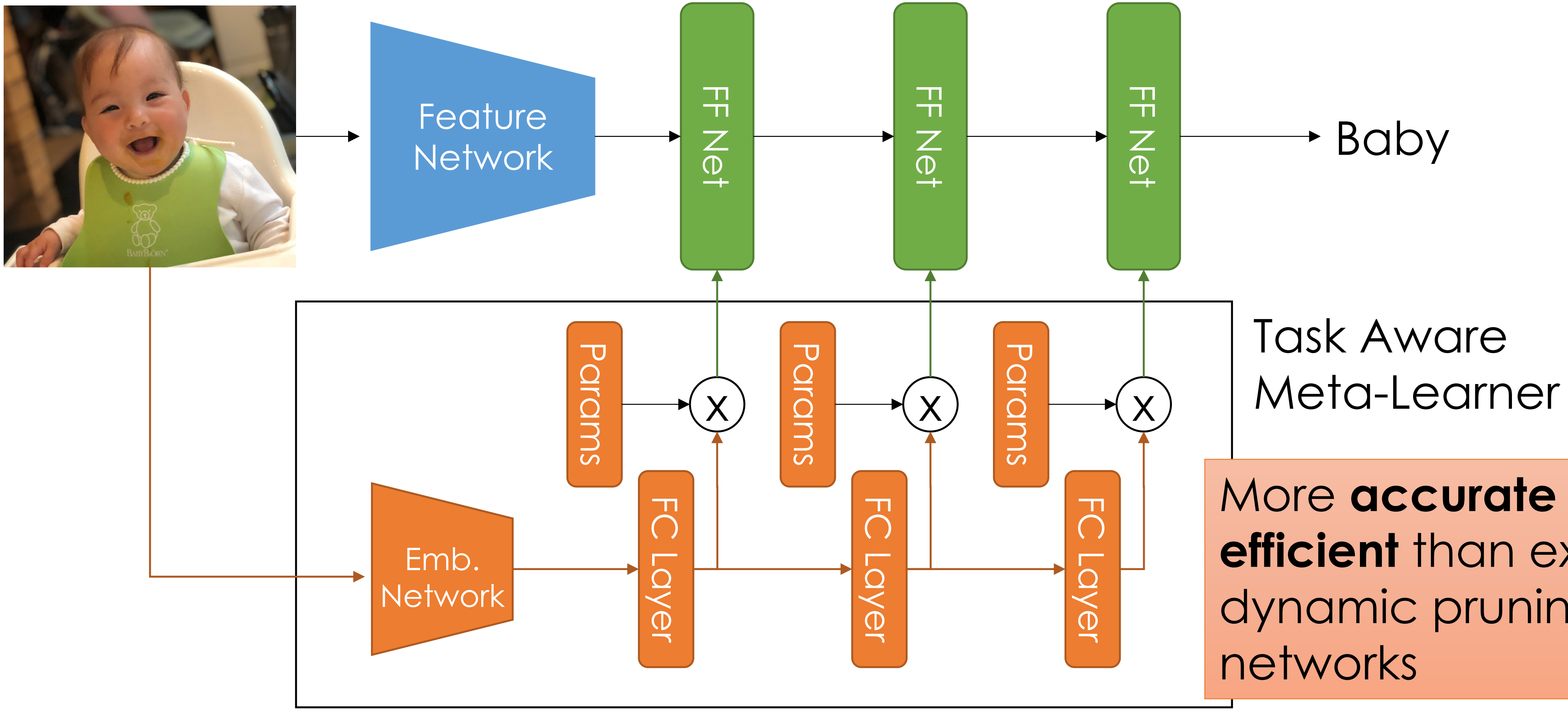
Skip more layers on clear images

Easy Images
Skip **Many** Layers



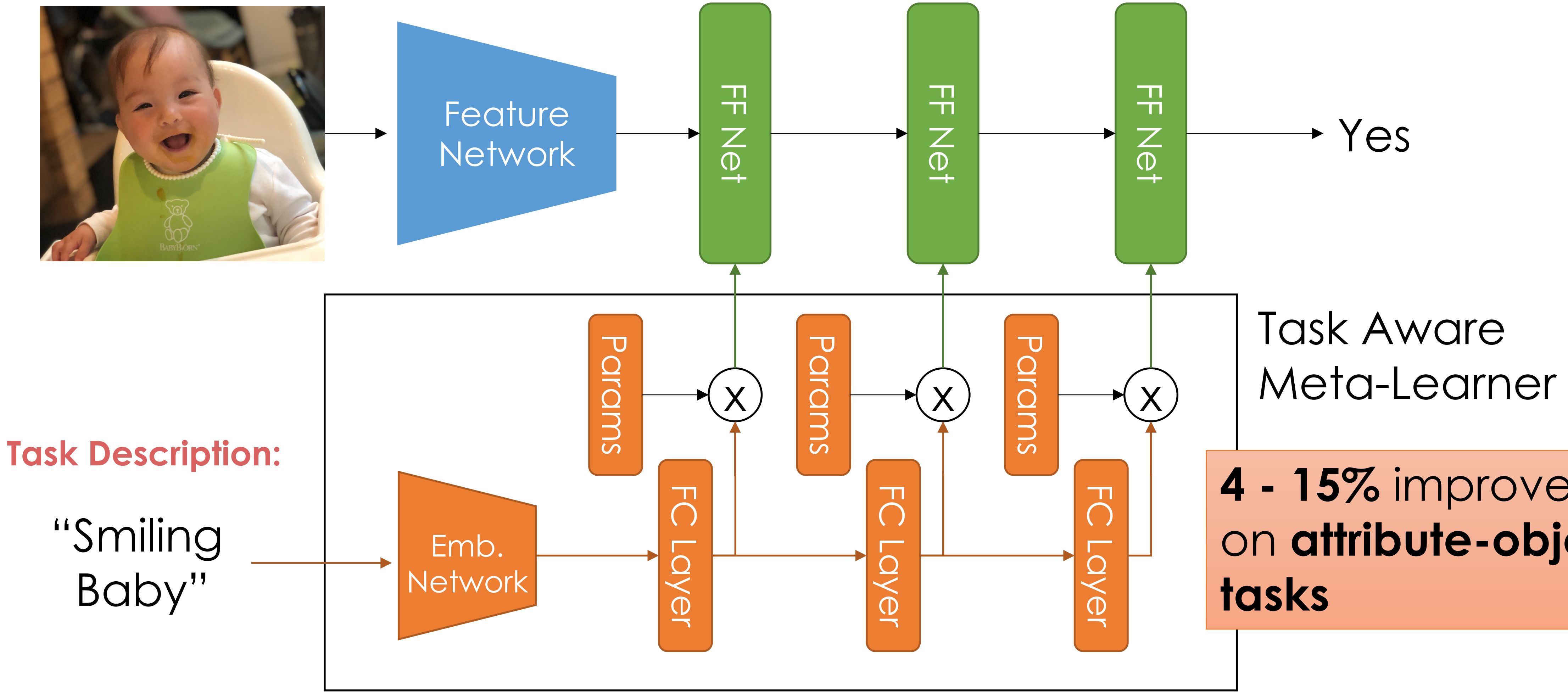
Hard Images
Skip **Few** Layers

Task Aware Feature Embeddings
[CVPR'19]



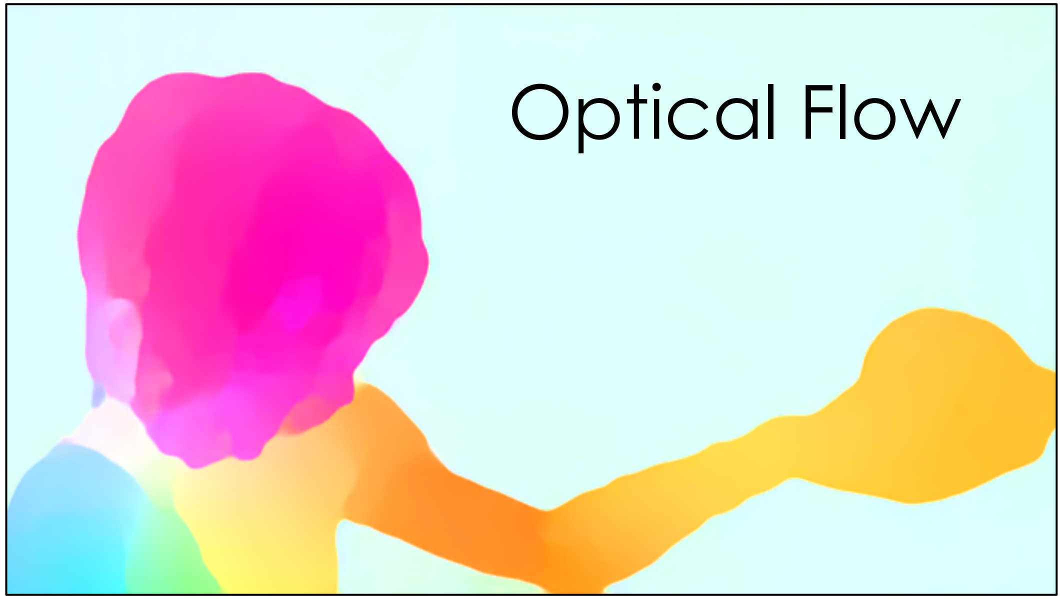
More **accurate** and **efficient** than existing dynamic pruning networks

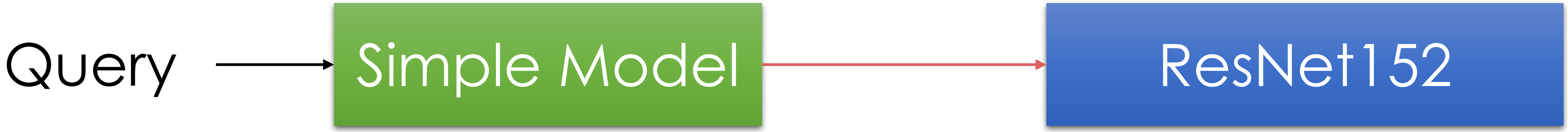
Task Aware Feature Embeddings
[CVPR'19]

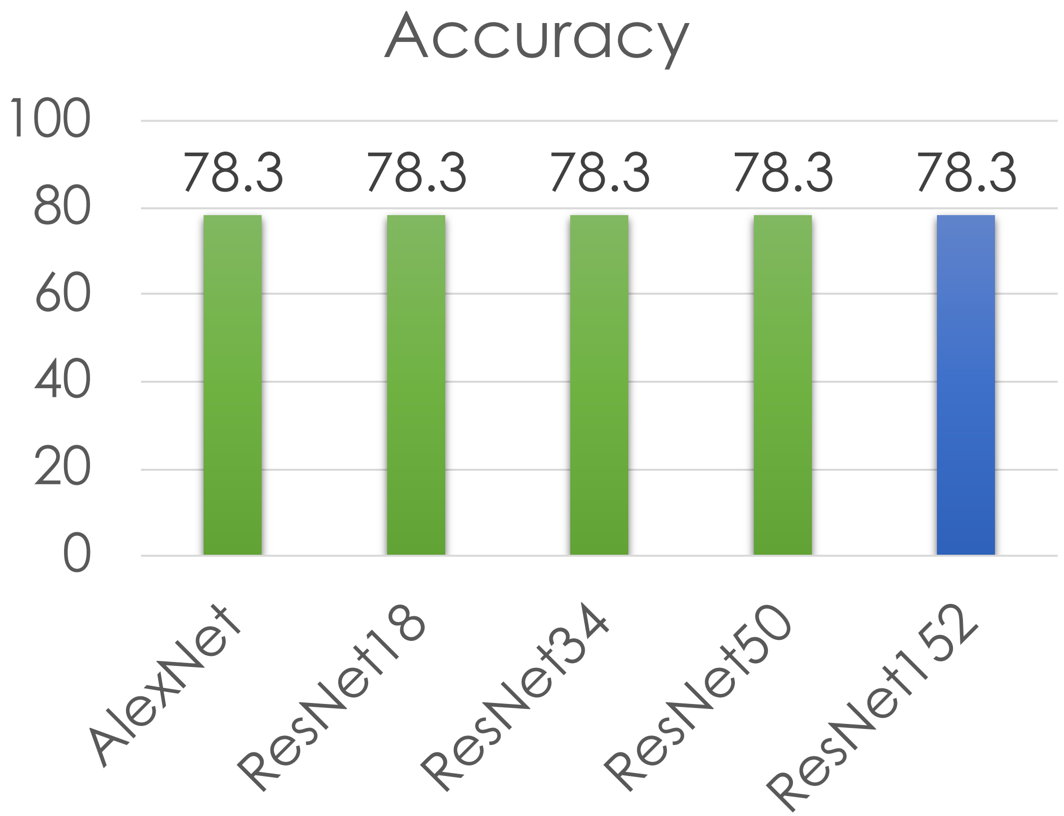


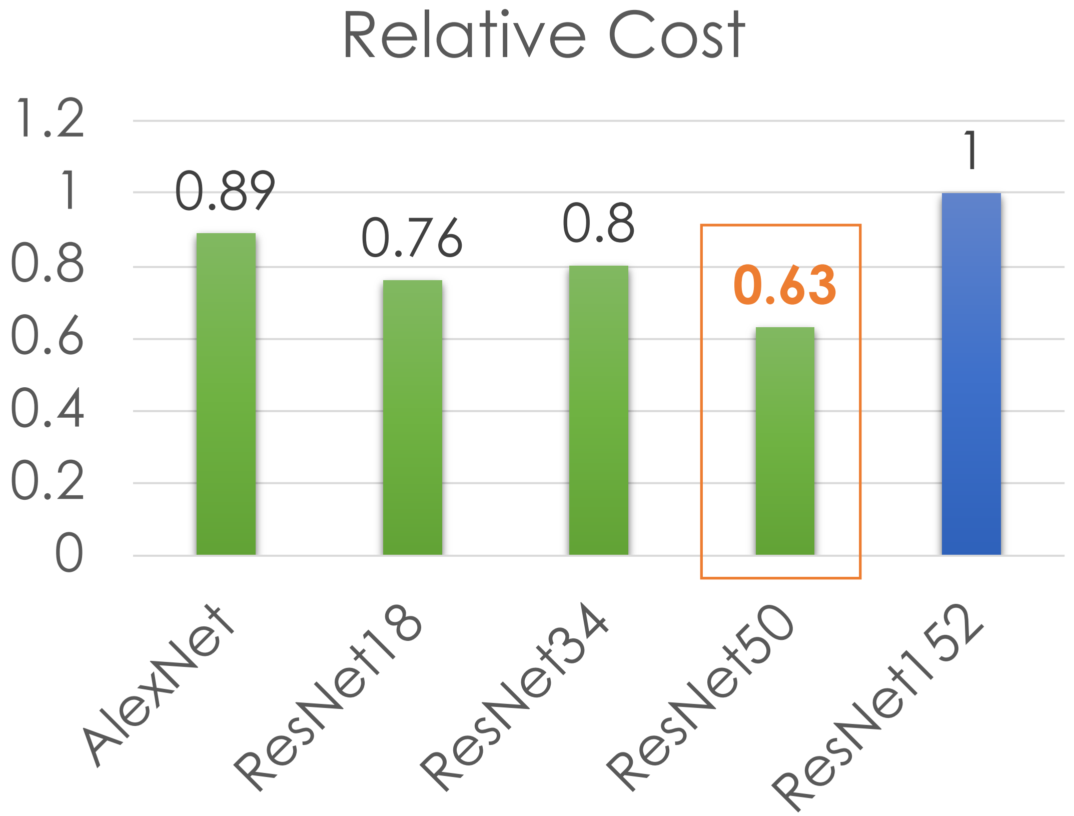
4 - 15% improvement on **attribute-object tasks**

Leverage **motion** to improve the **speed** and **accuracy** of **semantic segmentation**



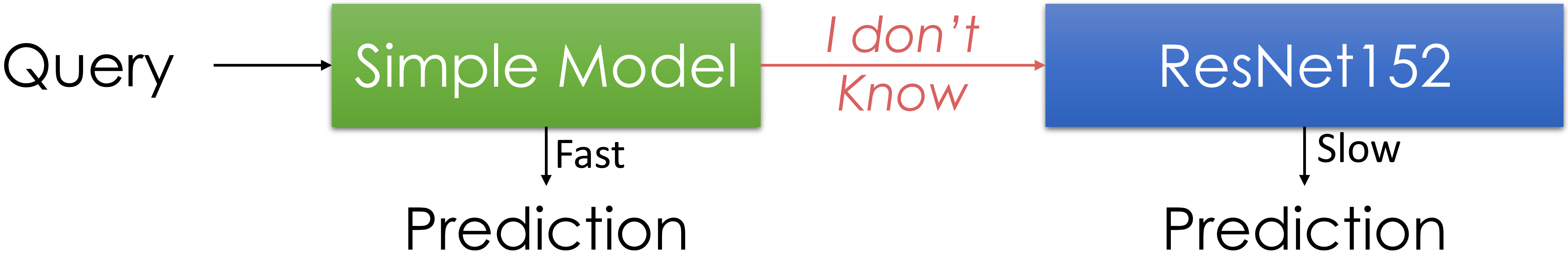




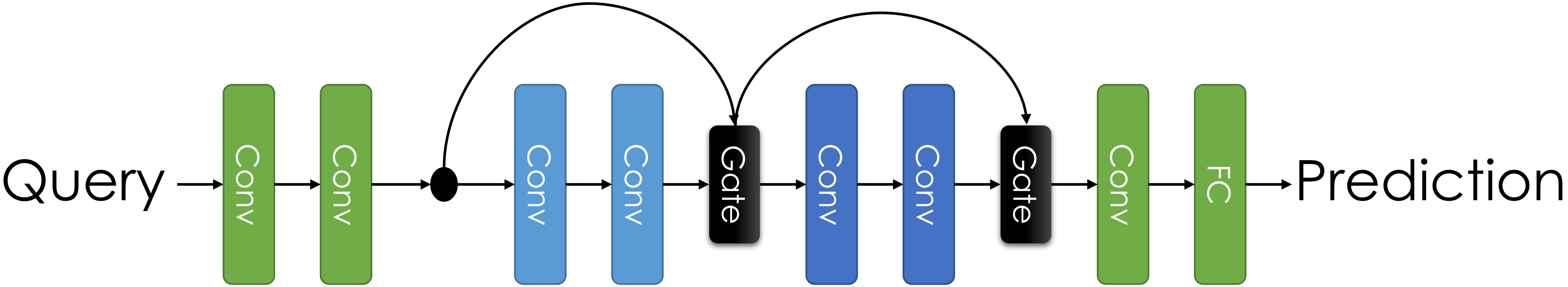


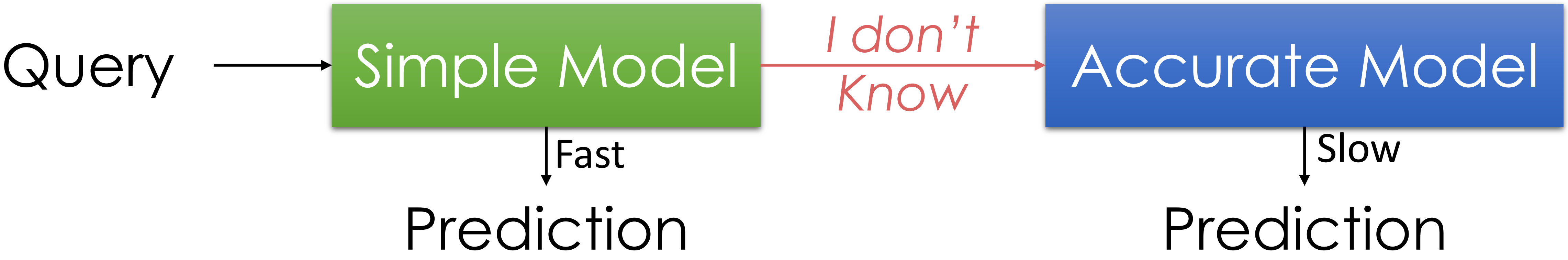
37% reduction in runtime

@ **no loss** in accuracy

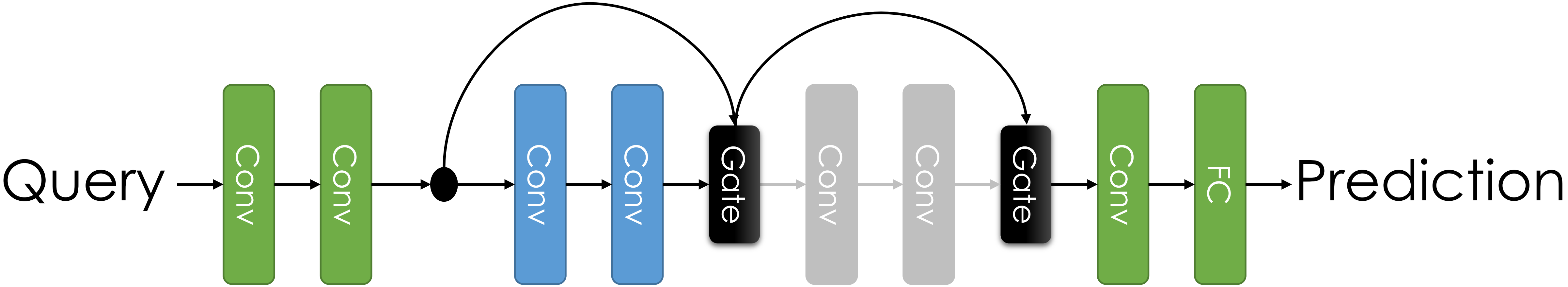


- Cascades within a Model



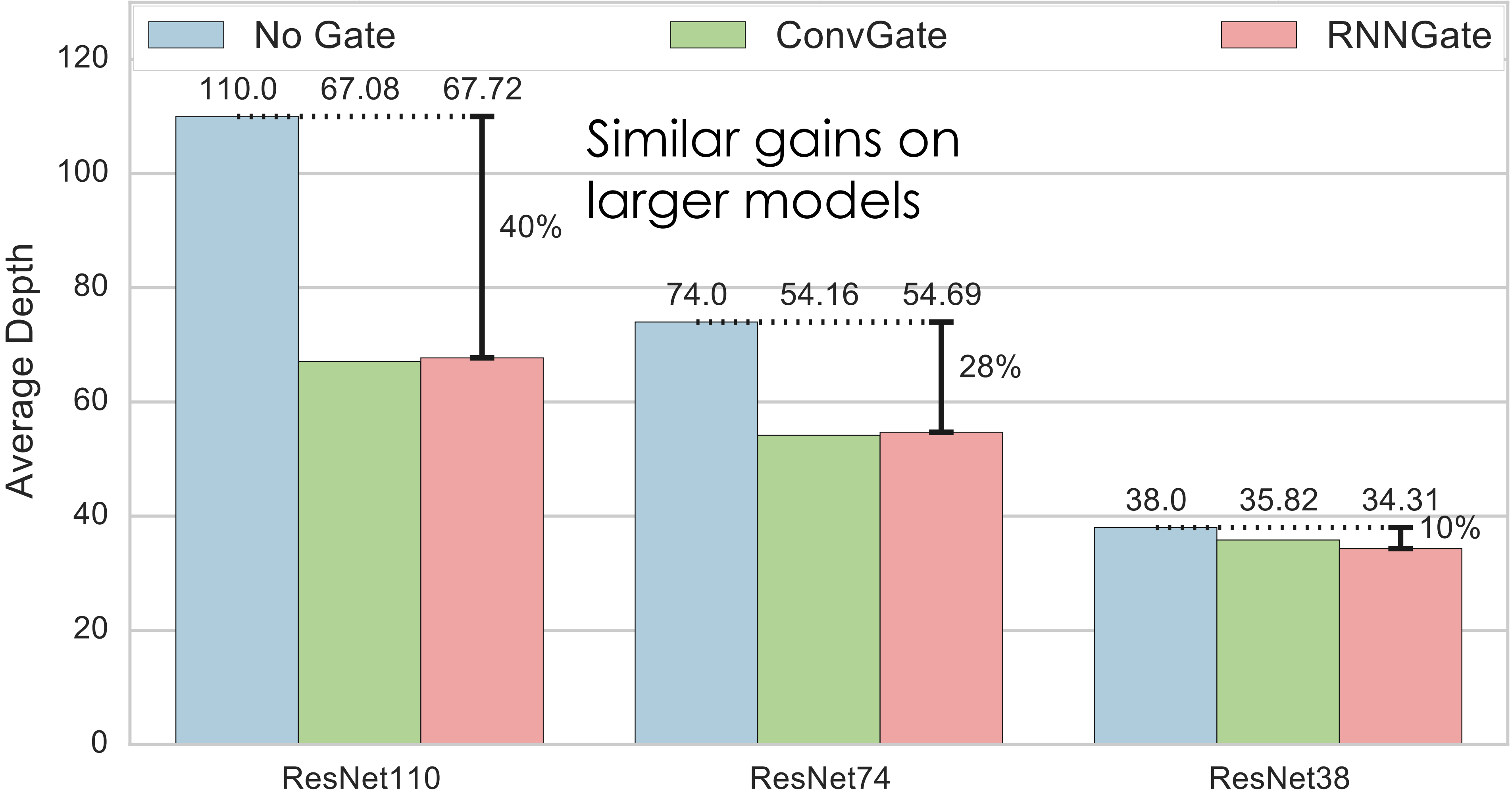


- Cascades within a Model

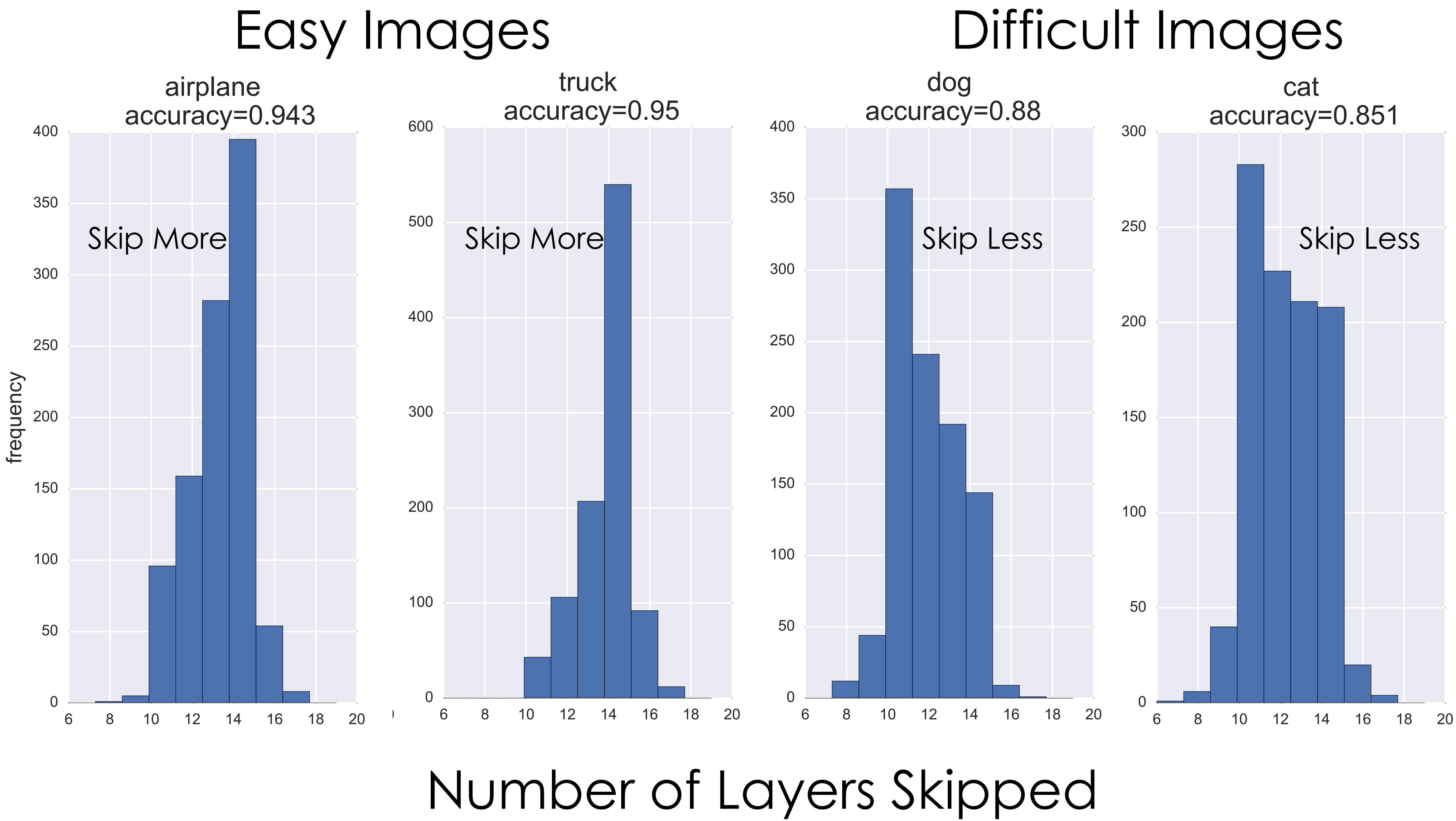


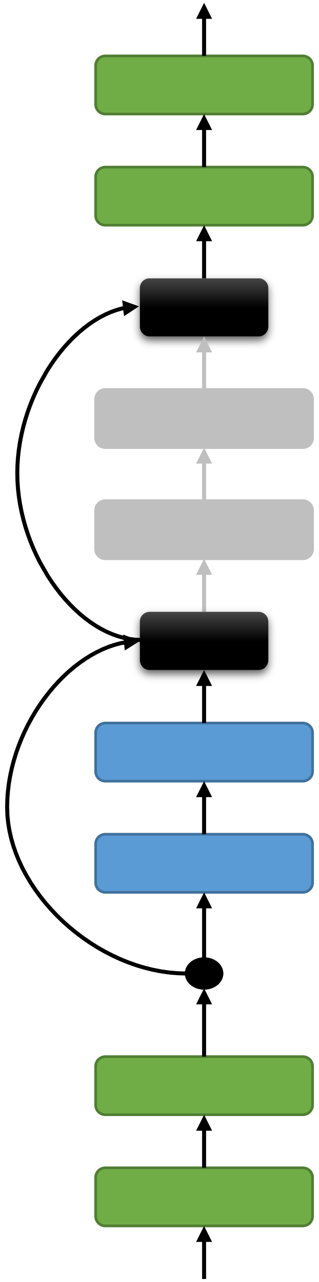
Skip Blocks

Cascading reduces computational cost



Similar gains on larger models


Easy Images
Difficult Images
airplane
accuracy=0.943
Skip More
truck
accuracy=0.95
Skip More
dog
accuracy=0.88
Skip Less
cat
accuracy=0.851
Skip Less
frequency
Number of Layers Skipped



Future Directions for Cascades

- Using **reinforcement learning** techniques to reduce gating costs
- **Query triage** during **load spikes** → forcing fractions of the network to go dark
- **Irregular execution** →
  - complicates batching
  - Issues for parallel execution