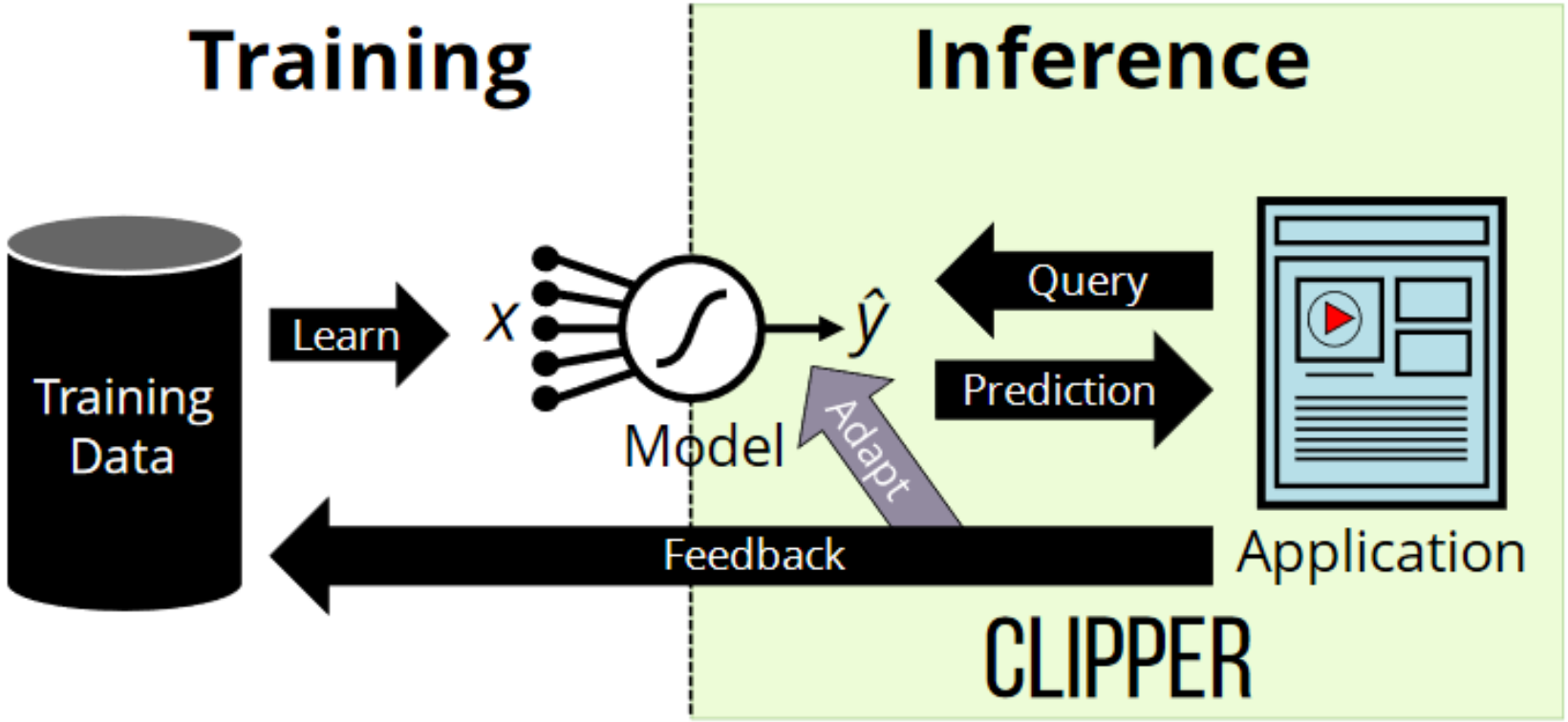
Clipper: A Low-Latency Online Prediction Serving System

Daniel Crankshaw*, Xin Wang*, Giulio Zhou*, Michael J. Franklin*†, Joseph E. Gonzalez*, Ion Stoica*

**UC Berkeley* *†The University of Chicago*

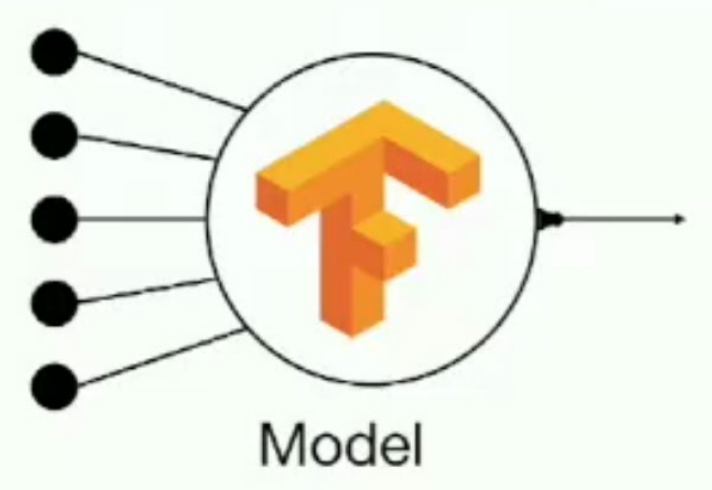
Machine Learning Lifecycle



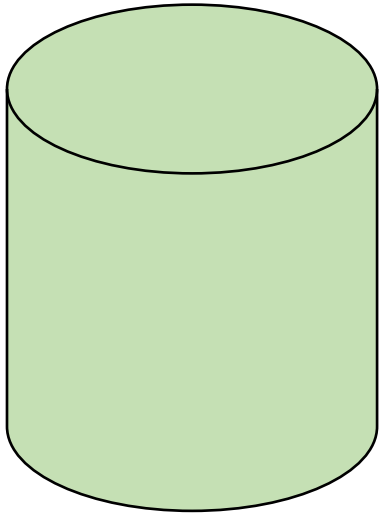
One-off Systems

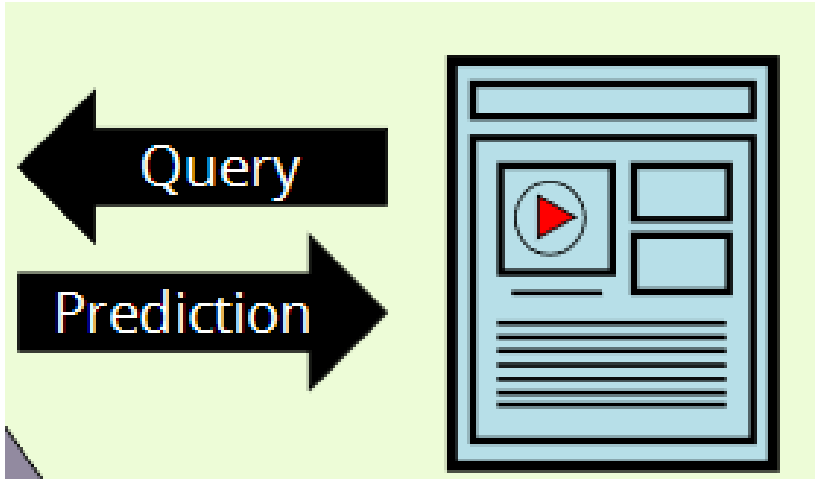
- Large companies invest a lot into building one-off systems to serve complex models
- Model and framework are tightly coupled to the application
- Can't easily change or update the model

Offline Scoring



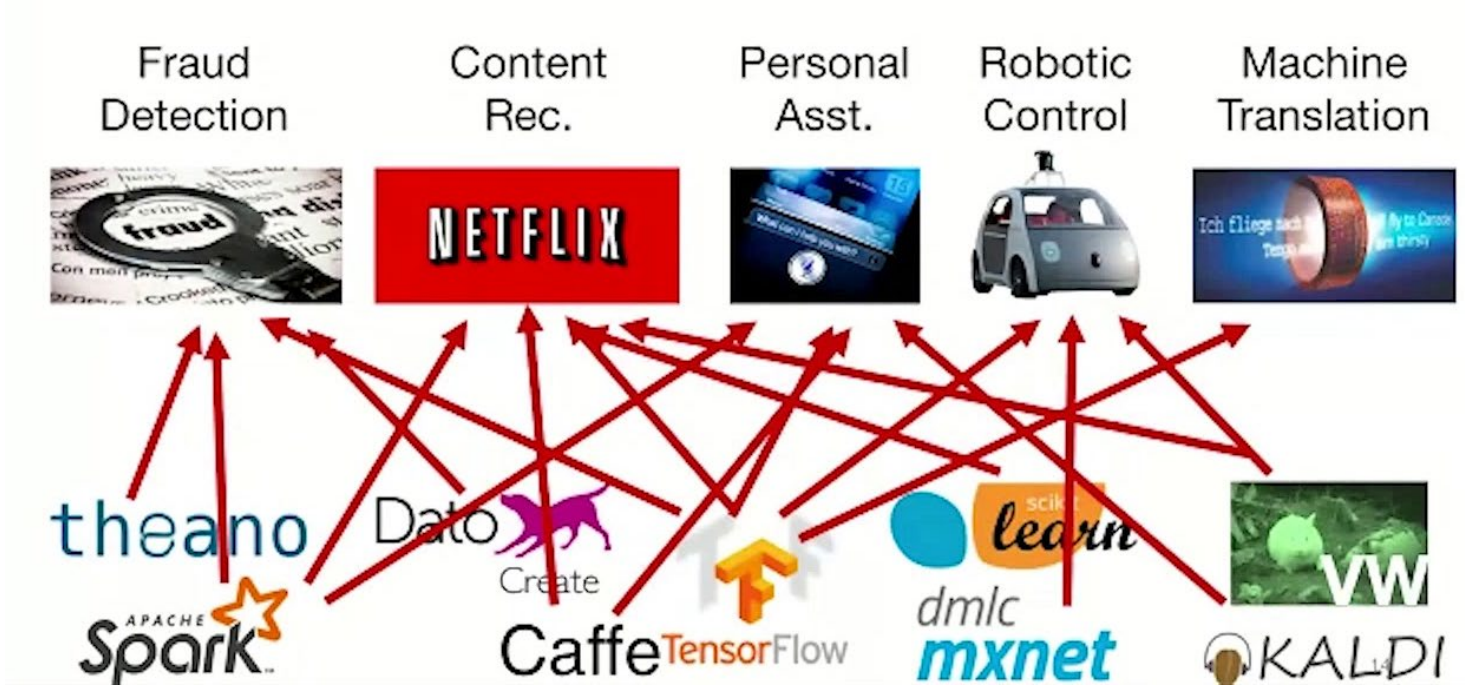
Data Store



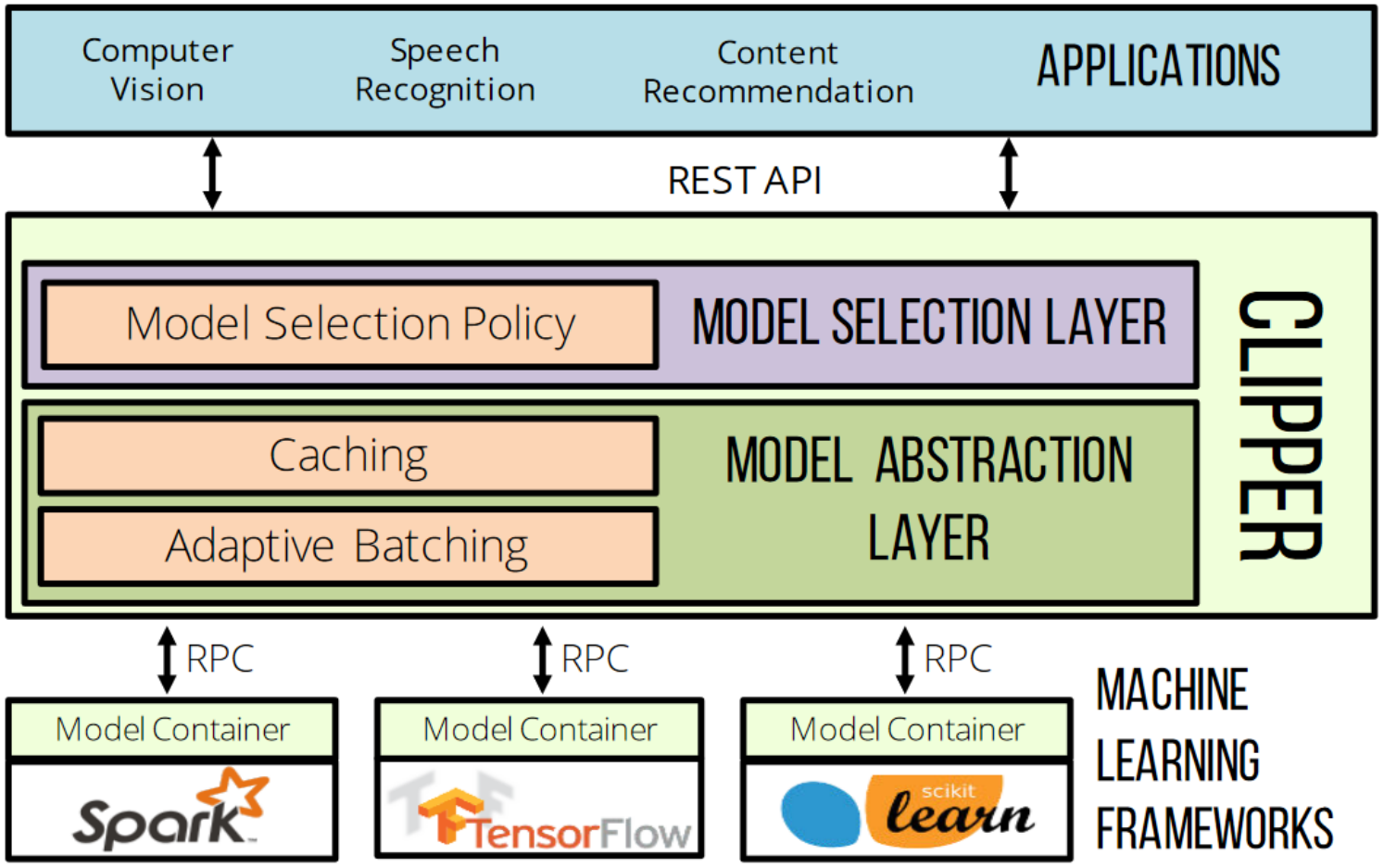


Clipper's Goals

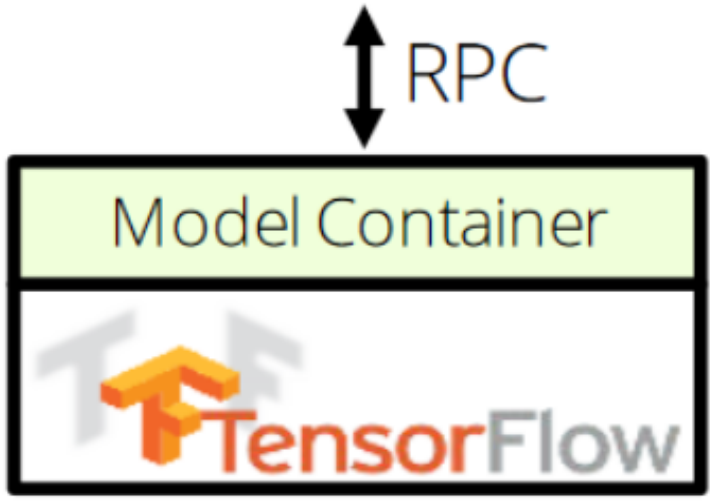
- Support high throughput and low latency serving
- Support many frameworks for many different applications



Clipper


Computer Vision
Speech Recognition
Content Recommendation
APPLICATIONS
REST API
Model Selection Policy
MODEL SELECTION LAYER
Caching
Adaptive Batching
MODEL ABSTRACTION LAYER
CLIPPER
RPC
RPC
RPC
Model Container
Model Container
Model Container
Spark
TensorFlow
scikit learn
MACHINE LEARNING FRAMEWORKS

Model Container



Model API

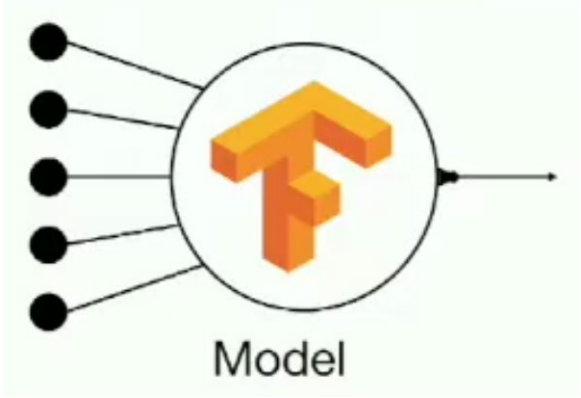
```
class ModelContainer:
  def __init__():
    """ Initialize the model """

  def predict_batch(inputs: list[X]) -> list[Y]:
    """ Compute predictions for a batch of inputs """
```

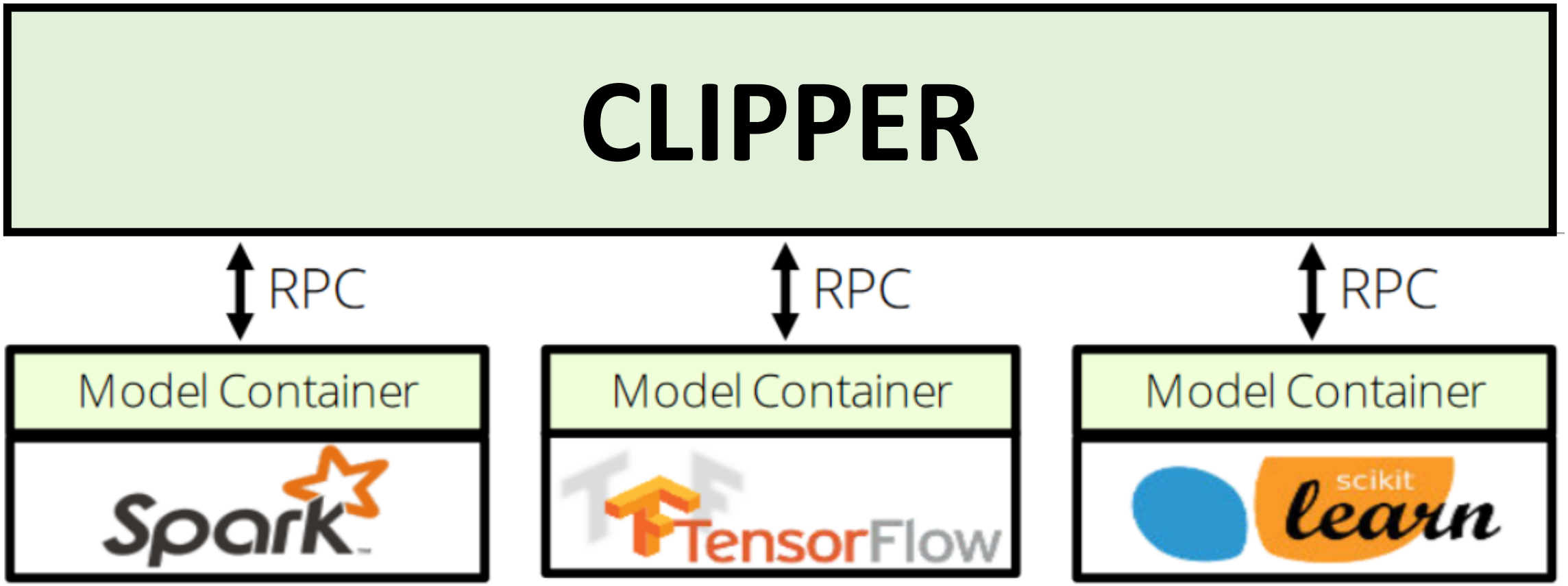
Model Container

```
class ModelContainer:
  def __init__():
    """ Initialize the model """

  def predict_batch(inputs: list[X]) -> list[Y]:
    """ Compute predictions for a batch of inputs """
```



TensorFlow


CLIPPER
RPC
RPC
RPC
Model Container
Model Container
Model Container
Spark
TensorFlow
scikit
learn

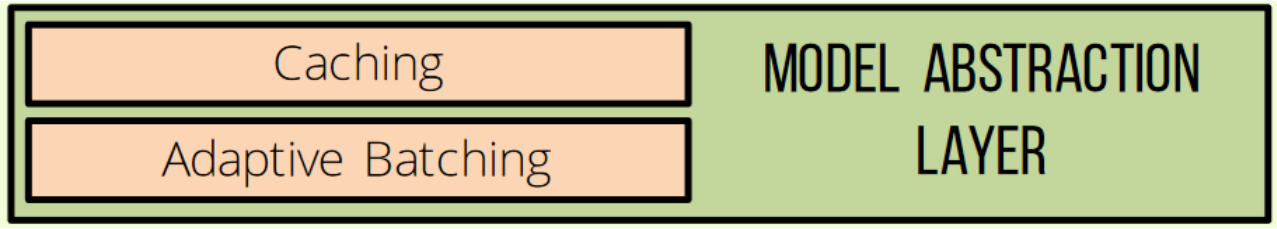
Batching

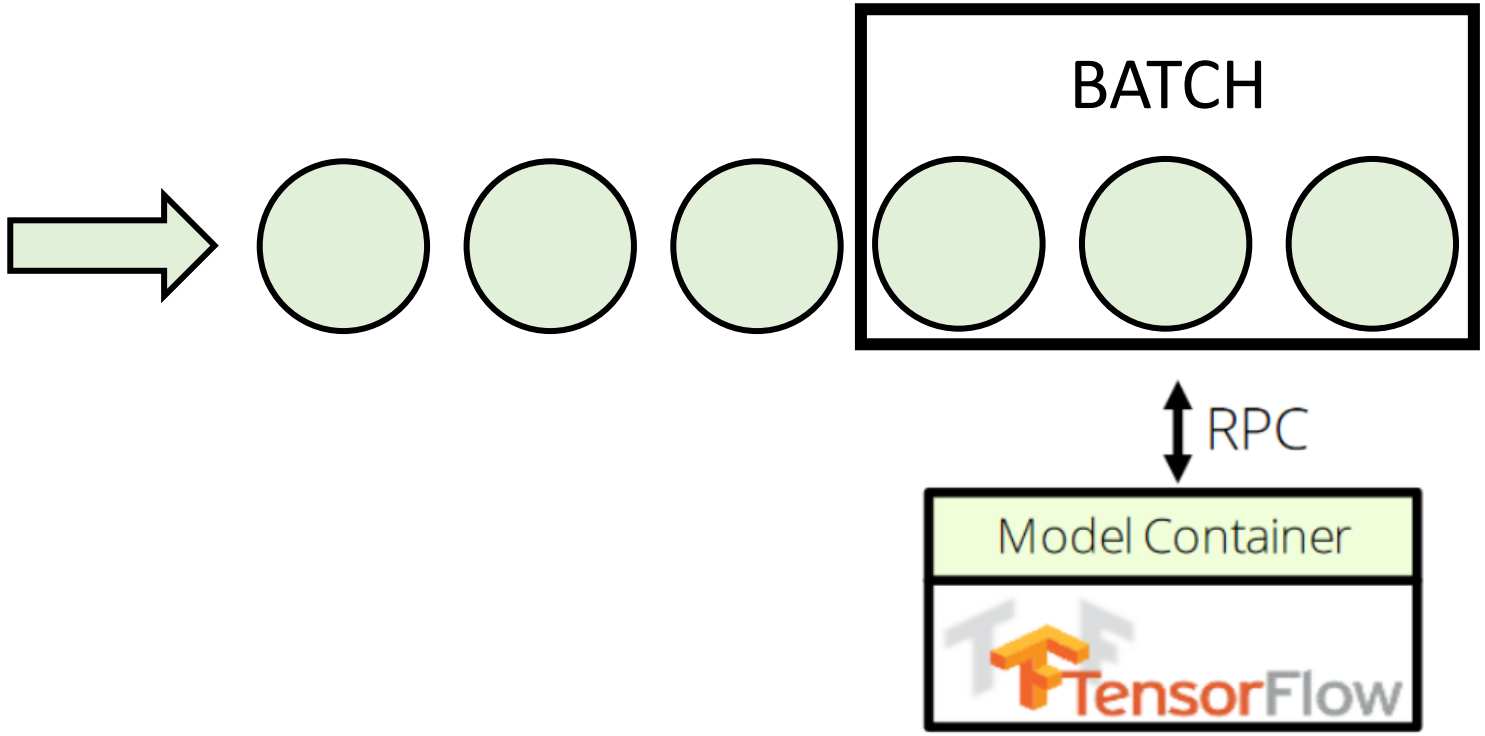
- Frameworks are designed for high throughput batch processing
- Concurrent users generate many simultaneous queries
- Hard to determine the optimal batch size

Clipper's solution:

adaptive batching for best tradeoff between throughput and latency

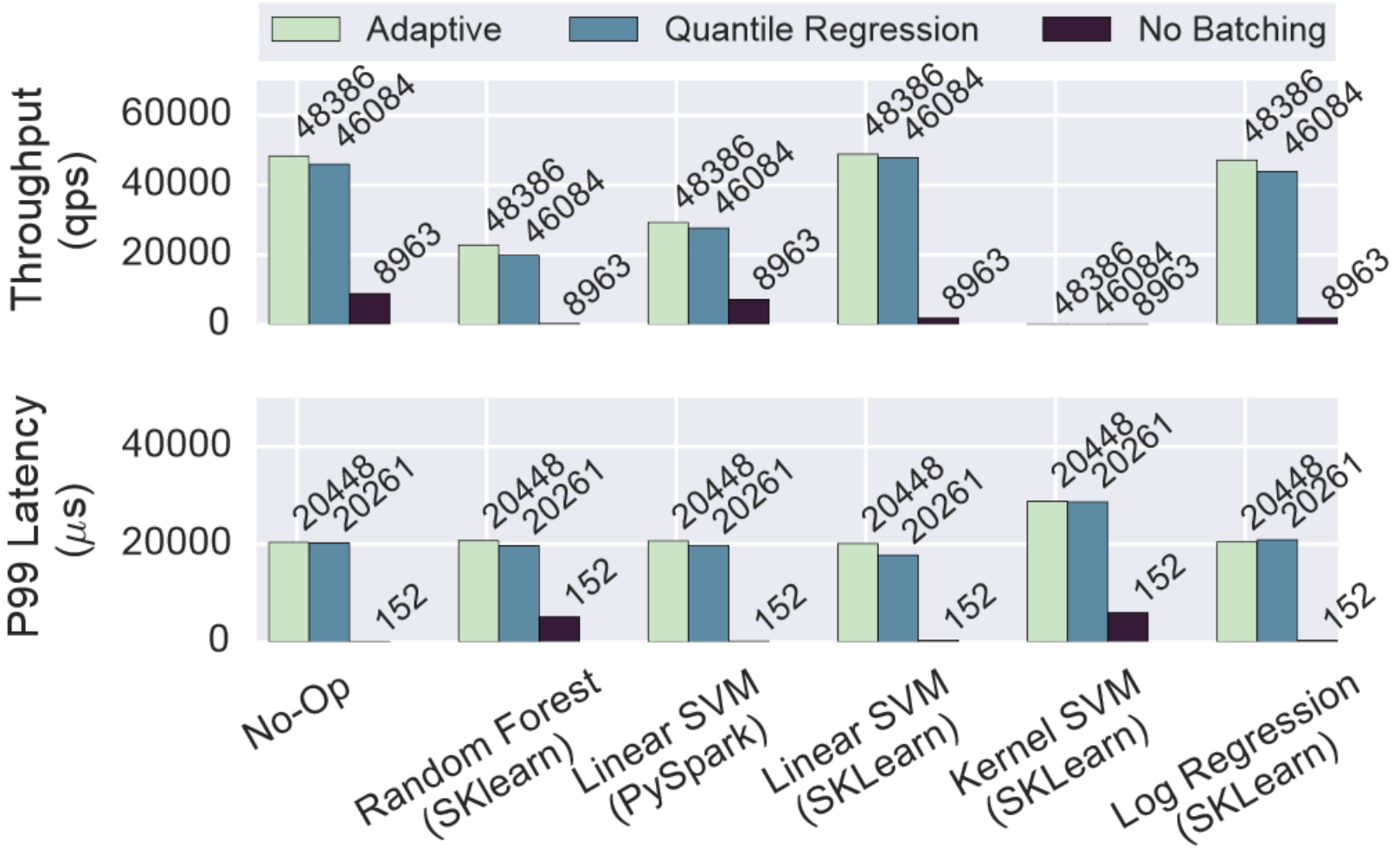
Adaptive Batching



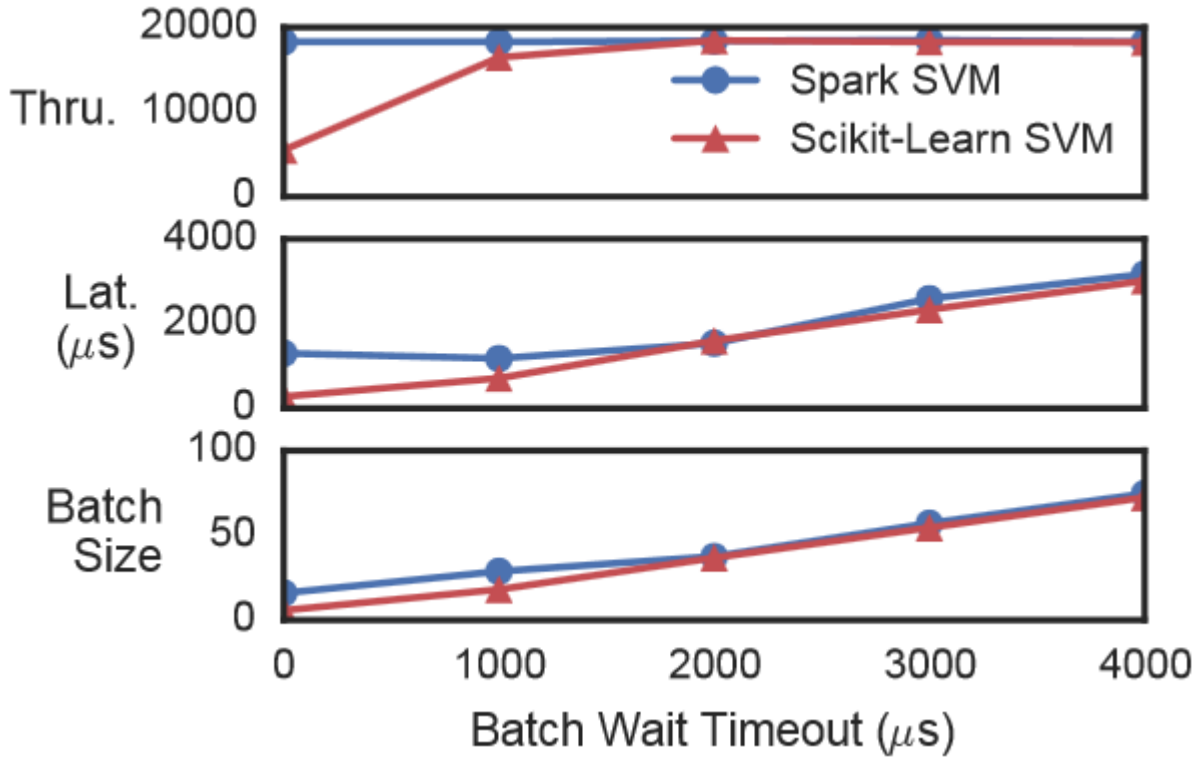


- Developer specifies latency objective
- Increase batch size until latency objective is exceeded
- When latency is exceeded, reduce batch size by a fraction

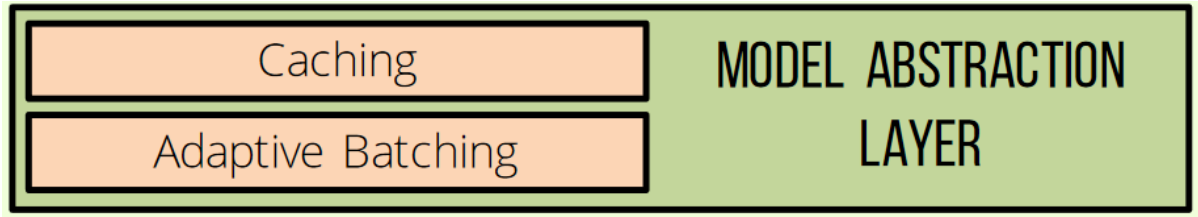
Adaptive Batching

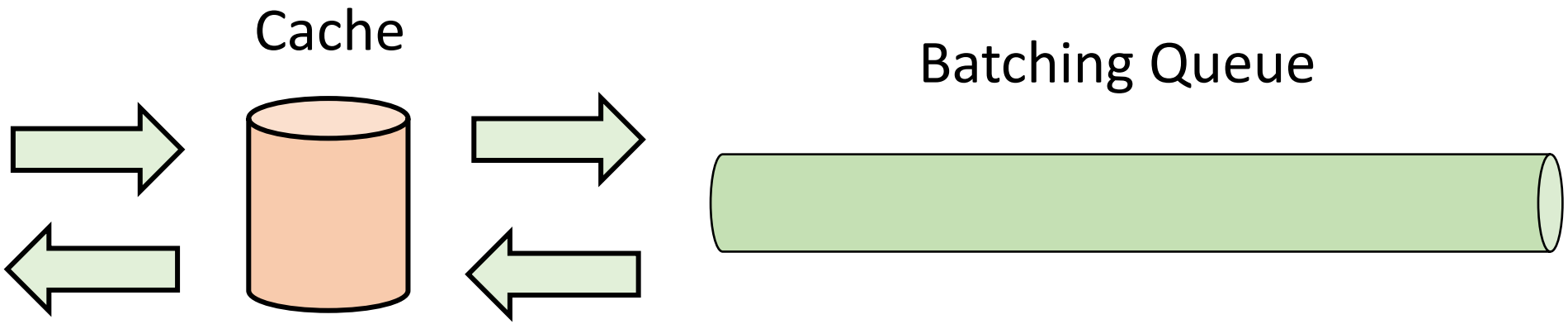


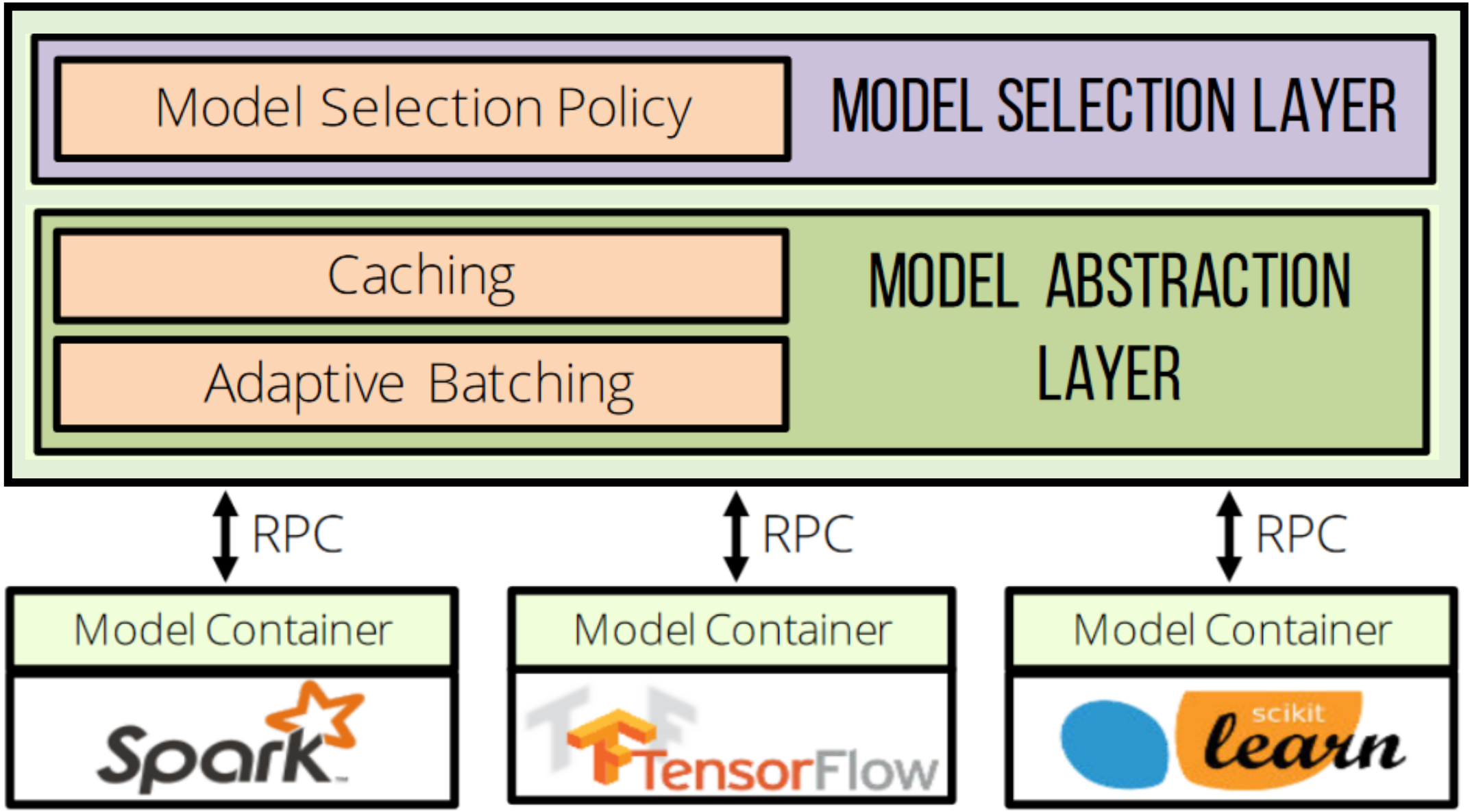
Delayed Batching



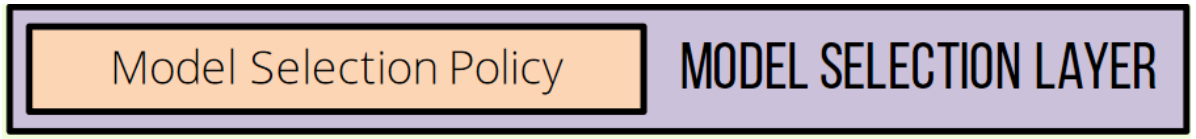
Caching


Caching
Adaptive Batching
MODEL ABSTRACTION LAYER


Cache
Batching Queue

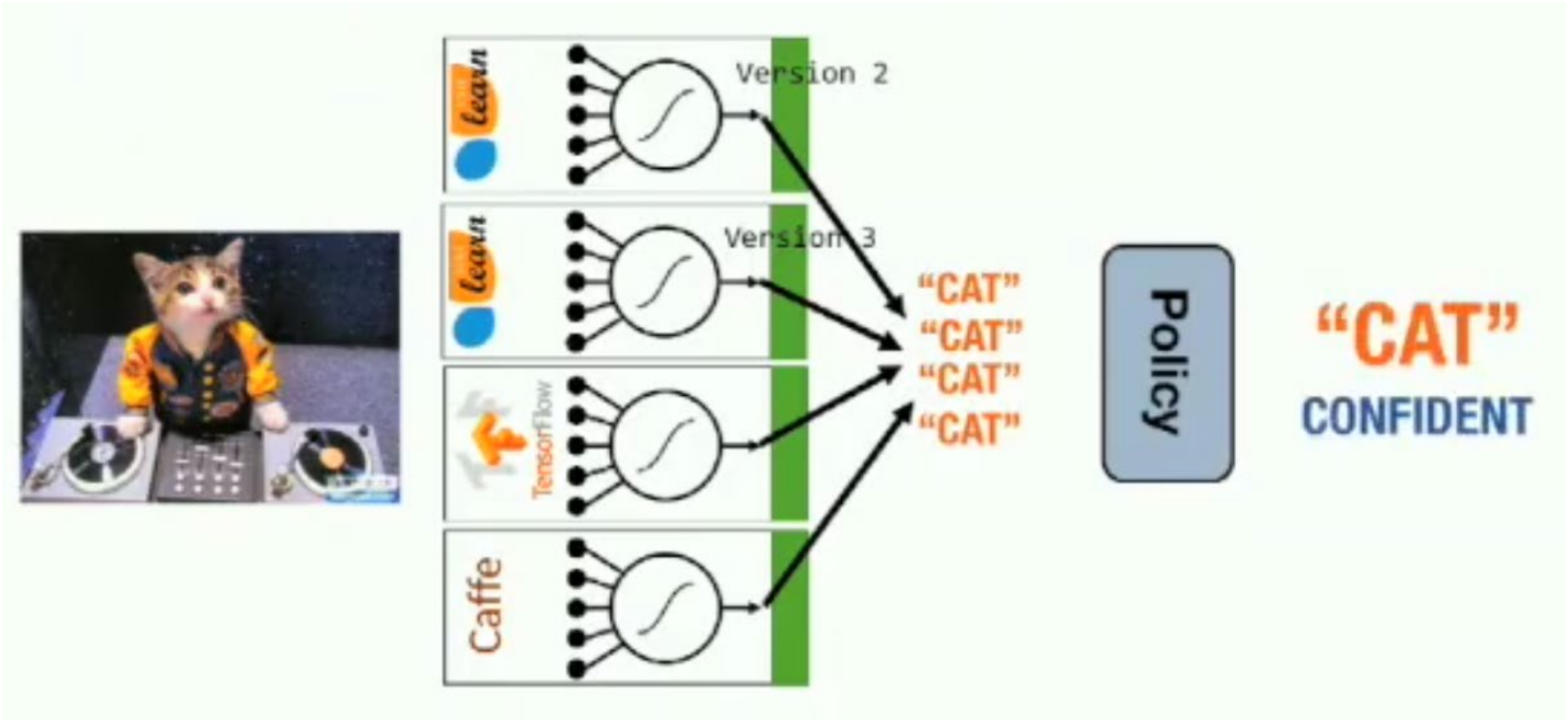

Model Selection Policy
MODEL SELECTION LAYER
Caching
Adaptive Batching
MODEL ABSTRACTION LAYER
RPC
RPC
RPC
Model Container
Model Container
Model Container
Spark
TensorFlow
scikit learn

Model Selection Layer

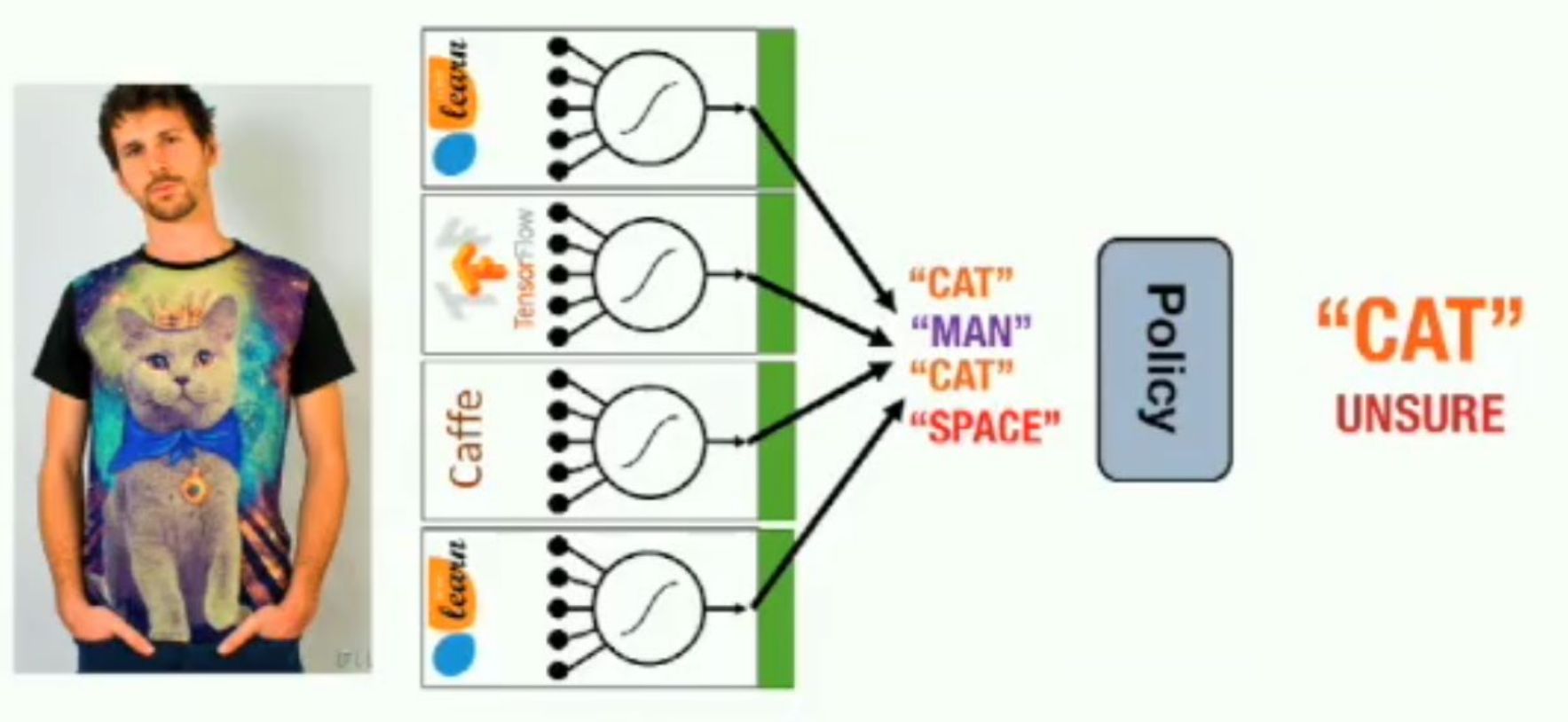


- Support for multiple models
- Ensemble models and return confidence scores
- Support feedback from users
- Abandon slow models, better to return a bad but on time prediction rather than a good but slow prediction

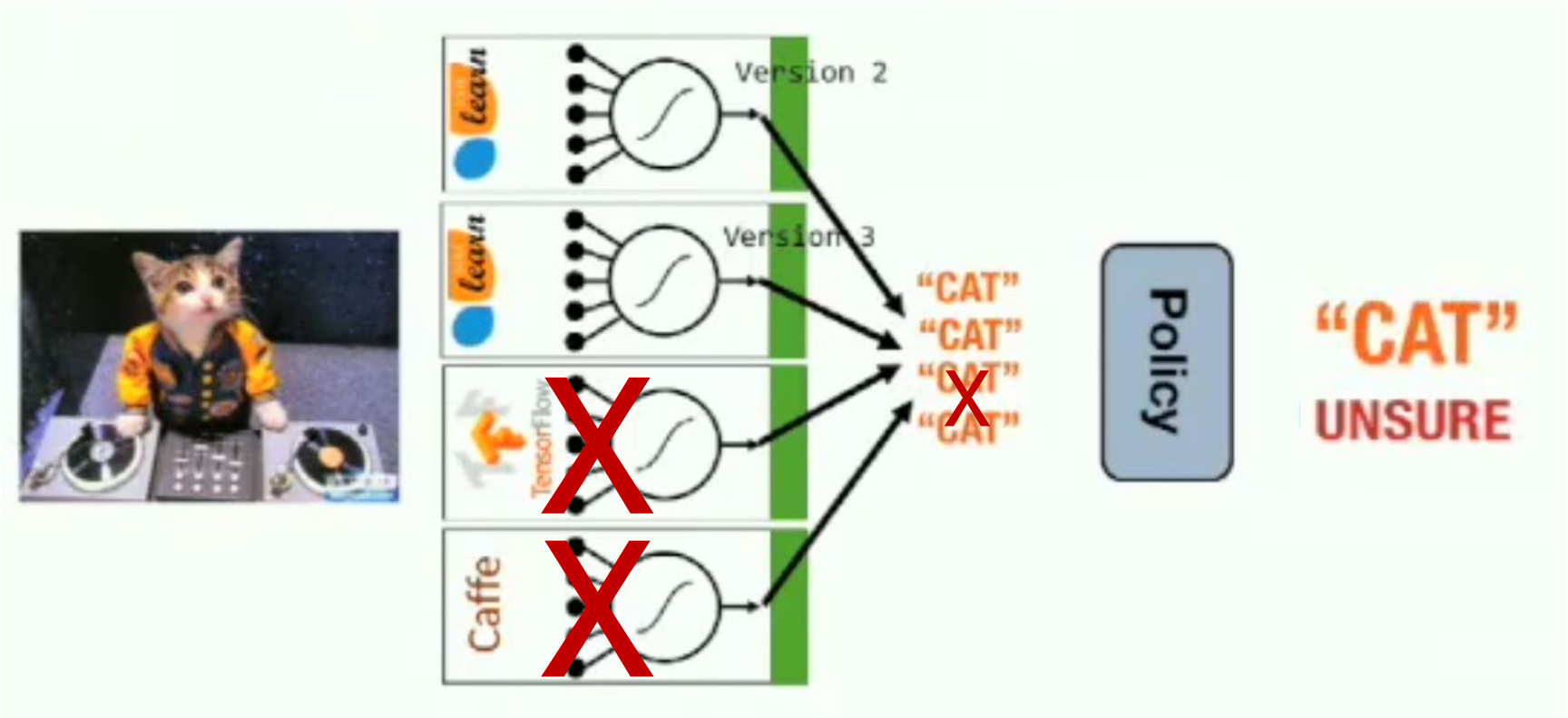
Ensembling

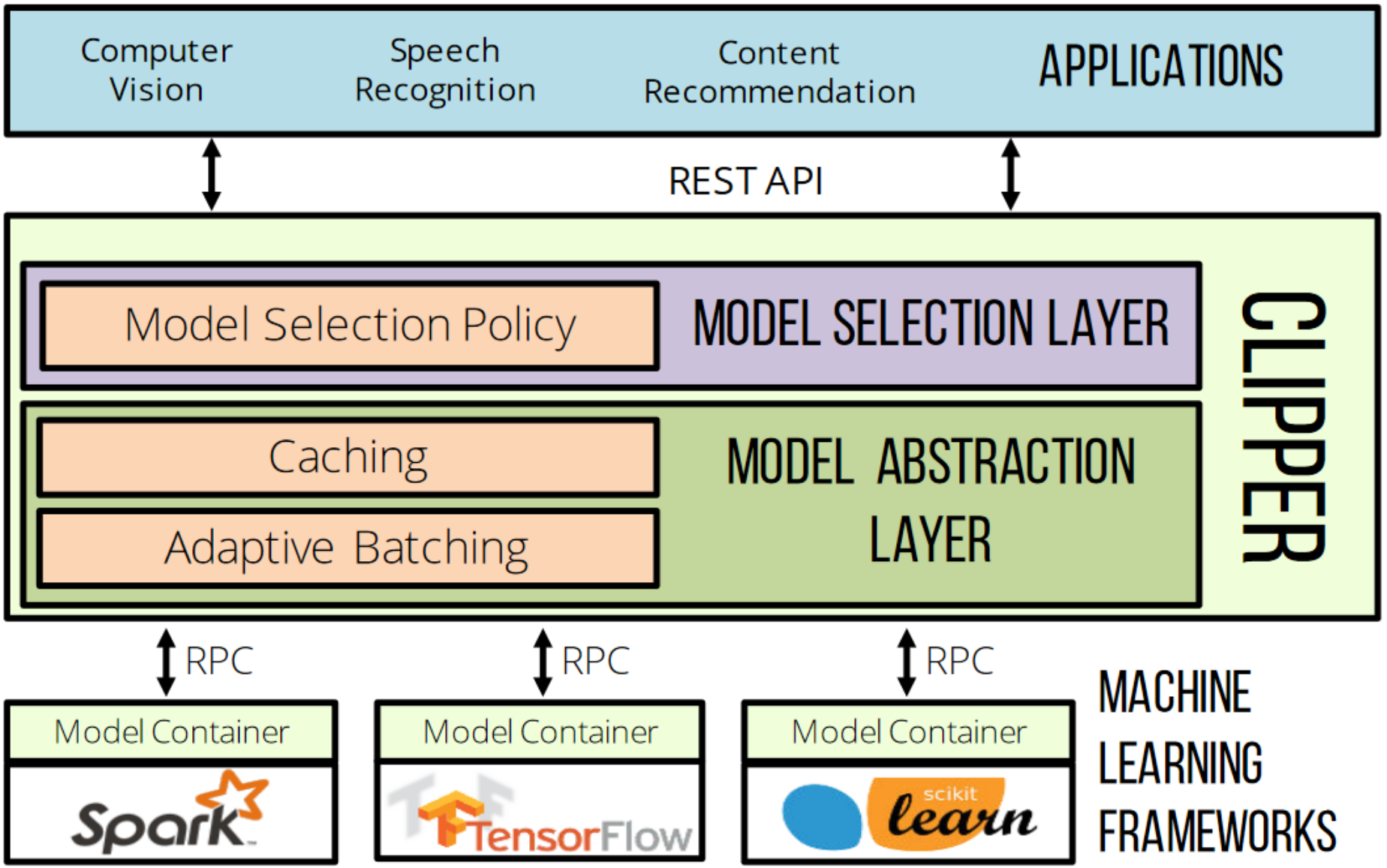

Version 2
Version 3
Tensorflow
Caffe
"CAT"
"CAT"
"CAT"
"CAT"
Policy
"CAT"
CONFIDENT

Ensembling


learn
TensorFlow
Caffe
learn
"CAT"
"MAN"
"CAT"
"SPACE"
Policy
"CAT"
UNSURE

Straggler Mitigation

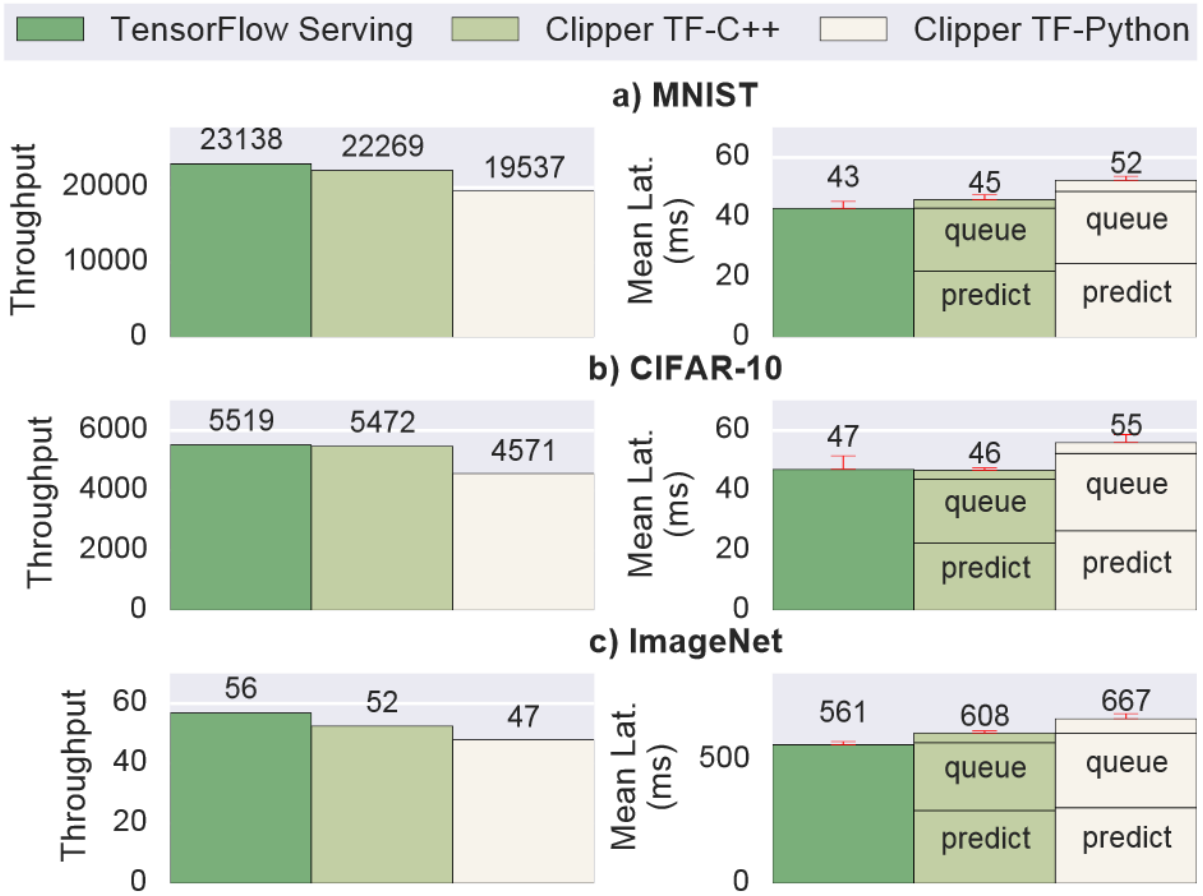



Computer Vision
Speech Recognition
Content Recommendation
APPLICATIONS
REST API
Model Selection Policy
MODEL SELECTION LAYER
Caching
Adaptive Batching
MODEL ABSTRACTION LAYER
CLIPPER
RPC
RPC
RPC
Model Container
Model Container
Model Container
Spark
TensorFlow
scikit learn
MACHINE LEARNING FRAMEWORKS

Compared to one-off system

- TensorFlow serving
  - Built specifically to serve TensorFlow models
  - Optimized for TensorFlow, tightly coupled
  - Doesn't support multiple frameworks
  - Fewer features than Clipper

Compared to one-off system



Compared to one-off systems

Clipper:

- Supports multiple frameworks and models
- Dynamically selects the optimal batch sizes
- Dynamically selects the best models via ensembling
- And is as performant as one-off systems

Downsides

- Treats each model like a black-box
- No low-level framework optimizations
- Can’t make assumptions about how frameworks execute

Serving DNNs like **Clockwork**: Performance Predictability from the Bottom Up

Arpan Gujarati*, Reza Karimi†, Safya Alzayat*, Wei Hao*, Antoine Kaufmann*, Ymir Vigfusson†, Jonathan Mace*

**Max Planck Institute for Software Systems* *†Emory University*

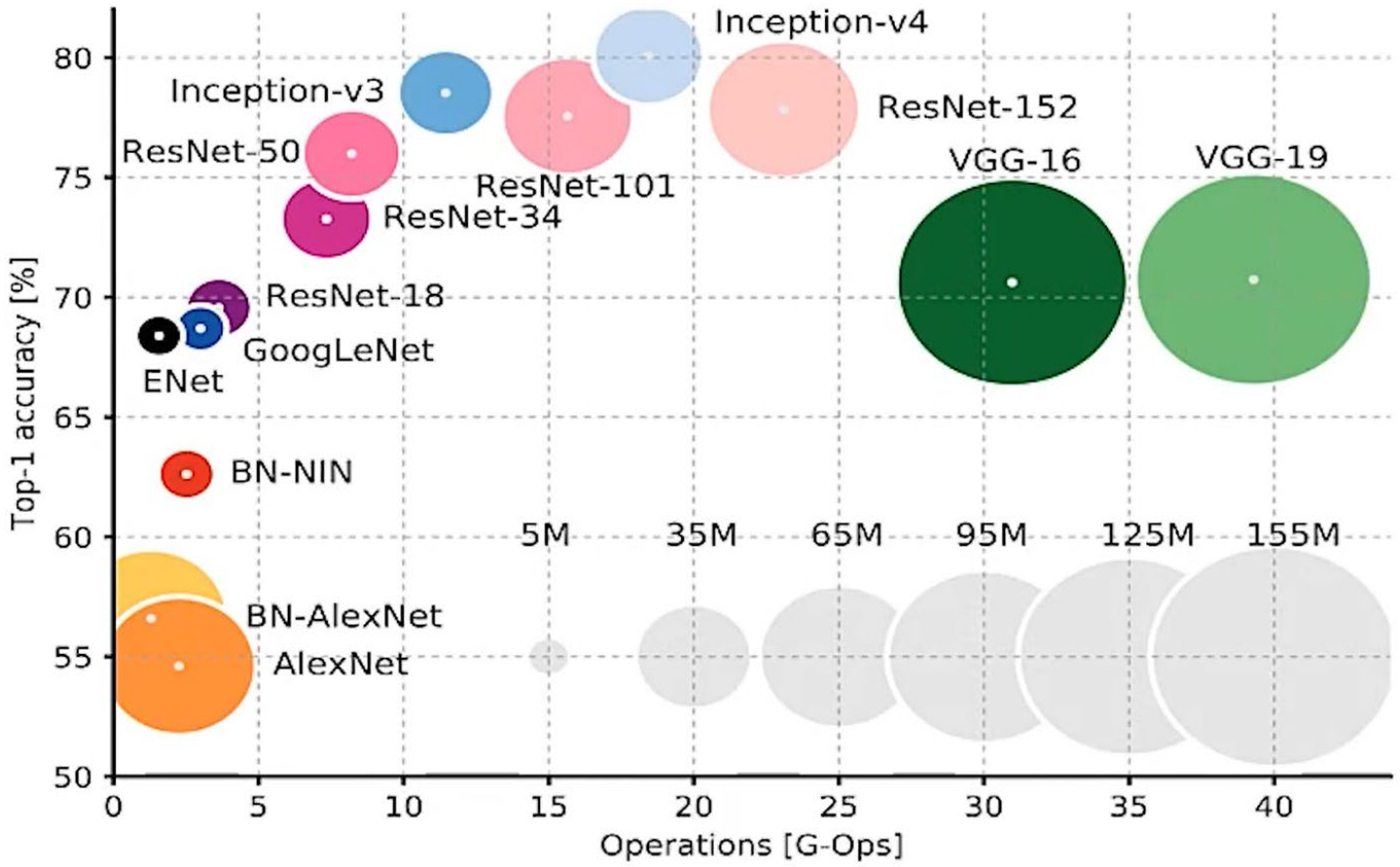
Inferences

- Why is inference important?
  - Model inference now on critical path of web requests
  - Facebook serves over 200,000,000,000,000+ inferences/day
  - 100+ companies developing hardware for accelerated ML inference
- Why is inference complex?
  - Too many models for many different tasks (with vastly different requirements)
- Solution – model serving systems
  - Hosted services that take pre-trained ML models and serve inference requests made to the models

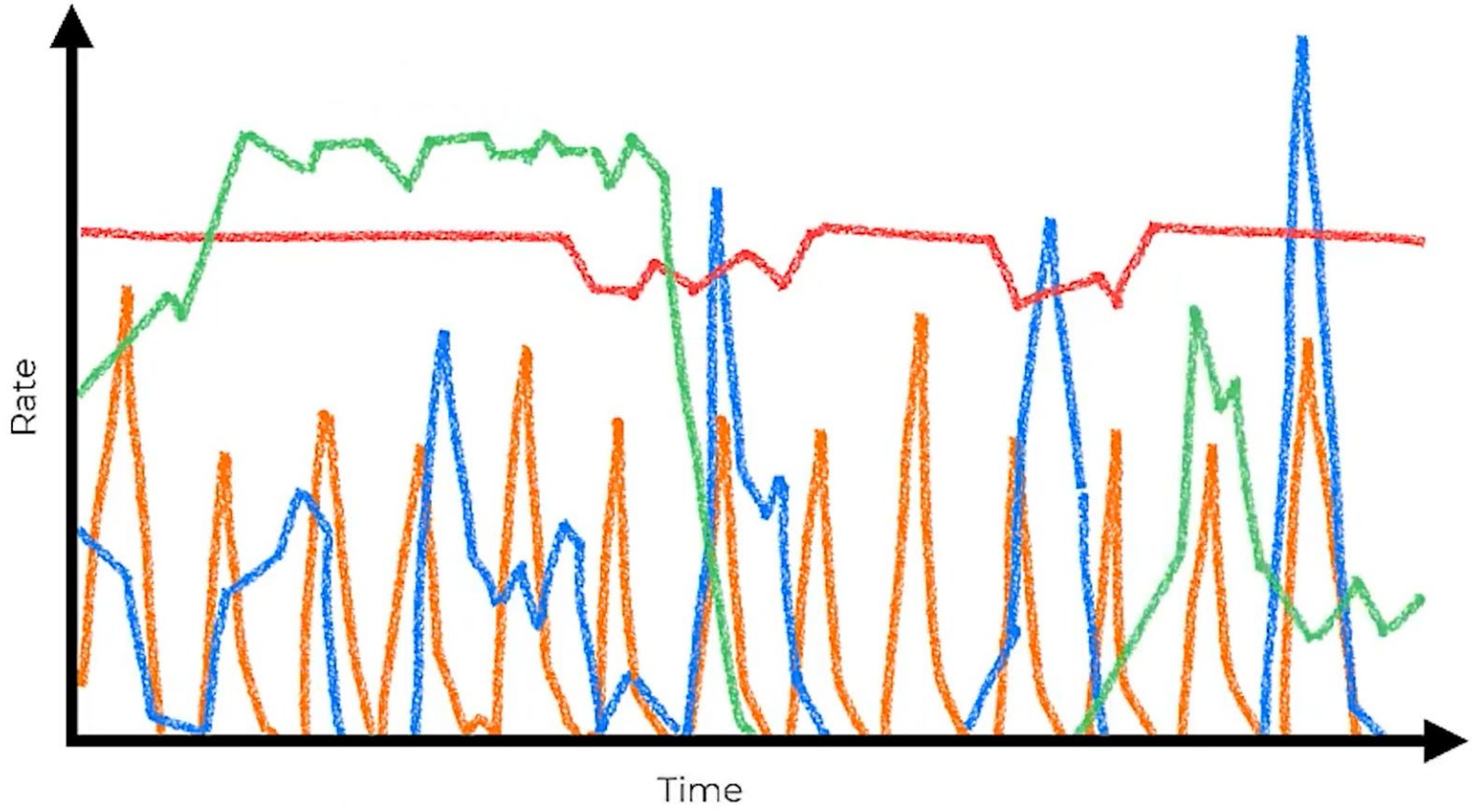
Inferences

- Inference requests have an inherent **deadline**
- Seek to bound latency, ideally provide SLOs
- Need to curtail tail latency

Model Serving Systems – Model Ecosystem



Model Serving Systems – Request Patterns



Model Serving Systems – Current Status

- Existing model serving architectures
  - Fundamentally assume constituent systems have unpredictable latency performance
  - Best effort techniques to curtail variability cascade unpredictability to other components and propagate tail latency to higher layers
  - Variability stems from internal design decisions (caching, concurrency, OS…)
- Current approaches to latency
  - Dedicate GPUs to models, lots of batching
  - Add more hardware, trading efficiency and utilization for throughput

Model Serving Systems – Hardware Costs

ResNet-50	Latency	Throughput	Cost
CPU	175.0 ms	6 req/s	$
GPU	2.8 ms	350 req/s	**$$$**

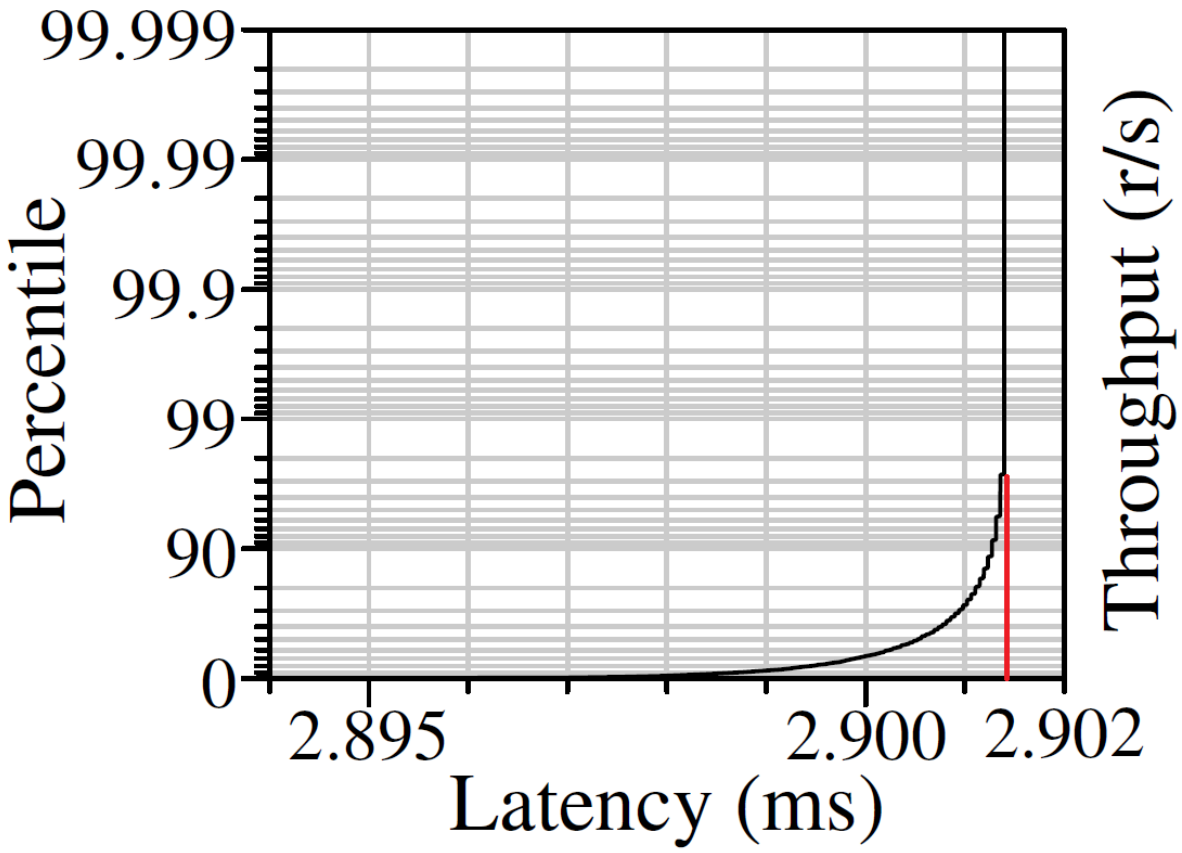
Clockwork – What is it?

- Fully distributed model serving system with predictable end-to-end performance
- Achieves 100ms latency targets for 99.9999% of requests simultaneously on thousands of models
- Mitigates tail latency by reducing unpredictability at the underlying layers and components

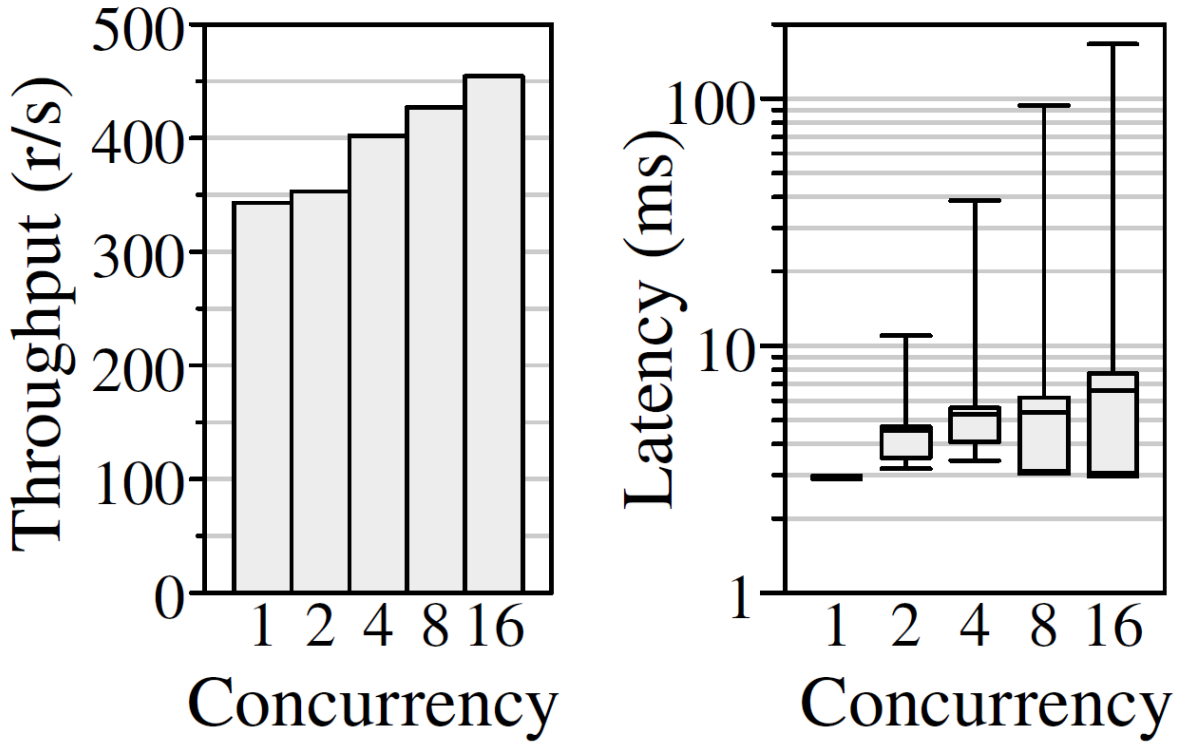
Clockwork – Core Assumption

- Inference using DNN models has deterministic performance
  - Inference request carries a fixed-size input tensor
  - Input copied to GPU over PCIe interconnect
  - DNN itself is a pre-defined sequence of tensor multiplications and activation functions (and lacks any conditional branching)
  - These operations are applied to the input tensor one-at-a-time to convert the input into an output tensor
  - Output is also a fixed-size tensor, which is copied back to the main memory over the PCIe interconnect

Clockwork – 1-thread latency



Clockwork – Throughput vs Latency



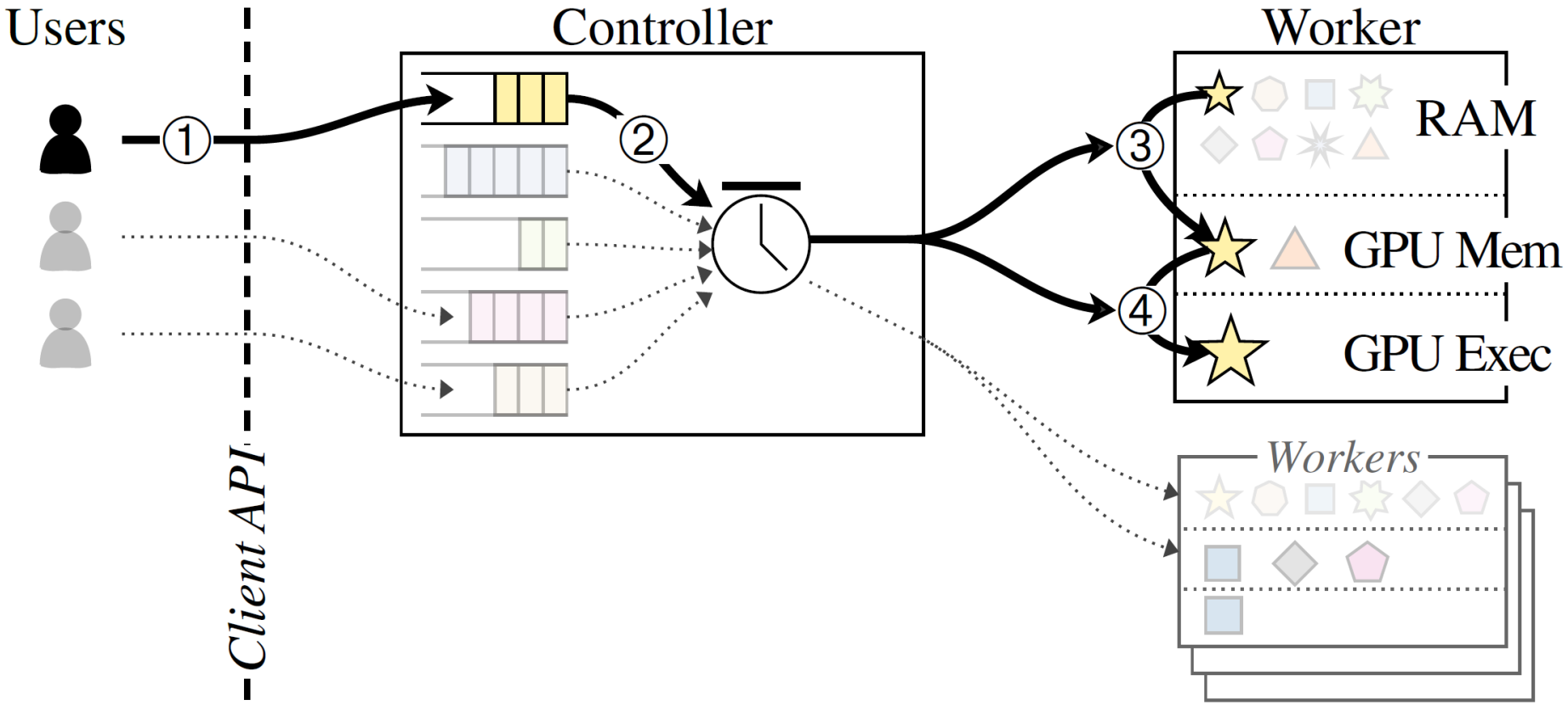
Latency

- Current approaches to latency
  - Ignore the problem; allow volatility to propagate to later requests
  - Match worst case latency for all requests
  - Scale resources
  - Add a feedback mechanism

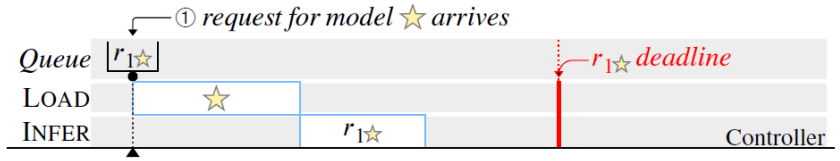
Clockwork – Predictability

- A predictable system from the bottom up
- Restrict choices available to the lower system layers
  - Hardware level
  - OS level
  - Application level
- *Consolidate choices* in the upper layers; force the lower layers to follow a narrow (and nearly deterministic) path of possible choices

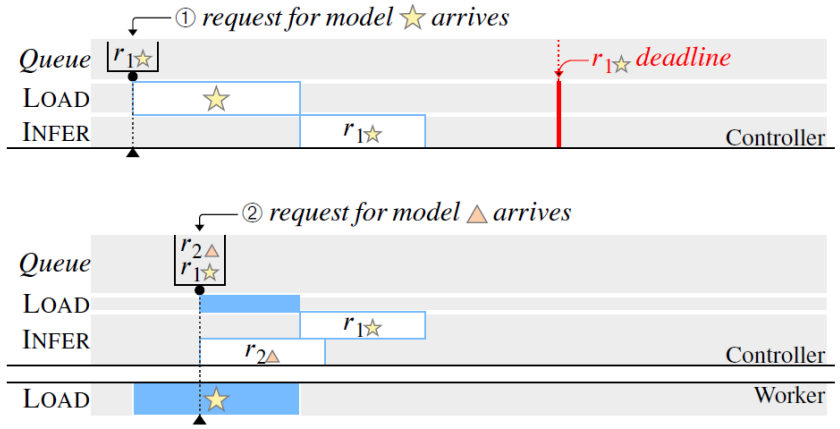
Clockwork – Design Overview



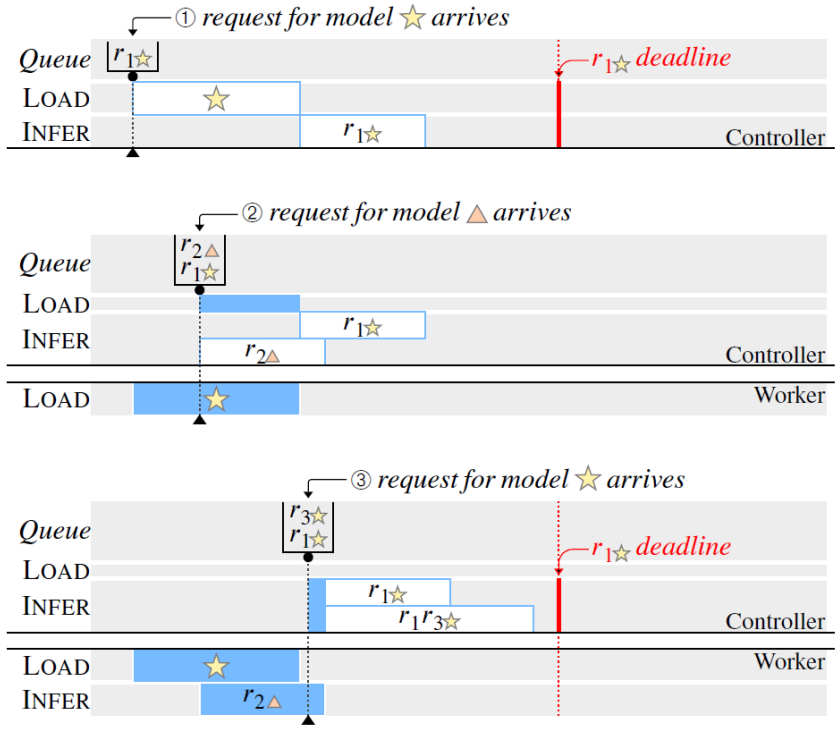
Clockwork – Example



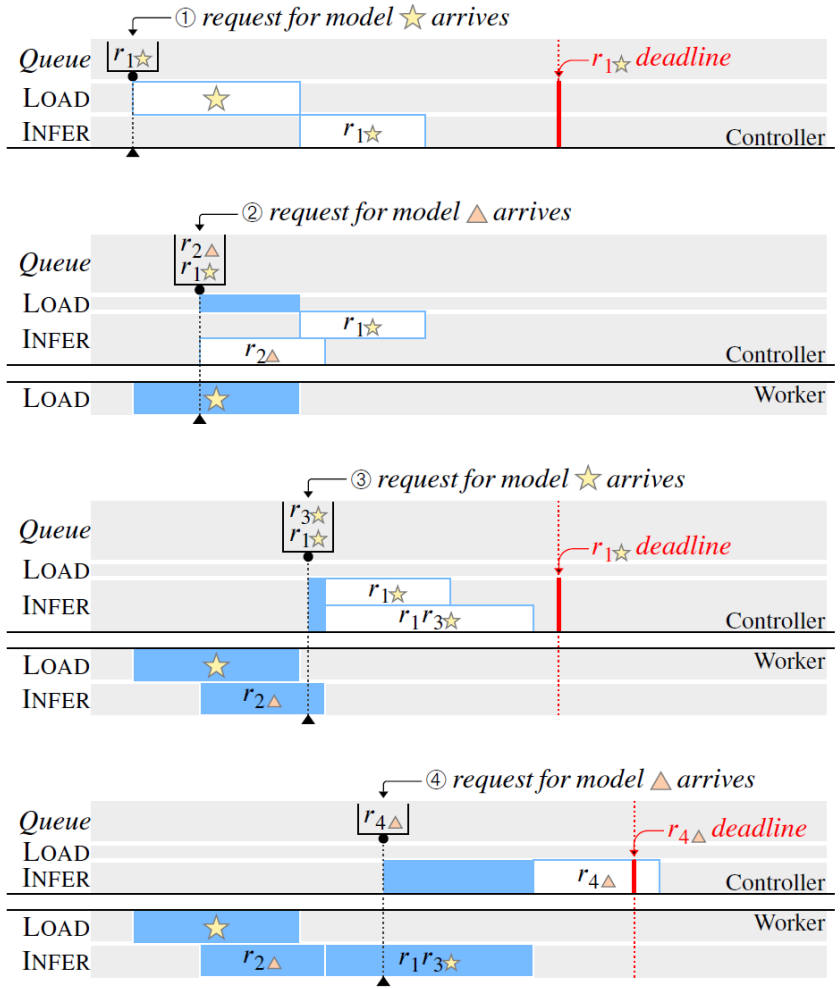
Clockwork – Example



Clockwork – Example



Clockwork – Example



Clockwork – Transfer & Execution Times

Model Family	Model	IO Size (kB)		Weights		GPU Execution Latency (ms)				
		Input	Output	Size (MB)	Transfer (ms)	B1	B2	B4	B8	B16
DenseNet	densenet169	602	4	56.5	4.50	5.18	6.29	8.57	12.82	21.85
Inception v3	inceptionv3	1073	4	95.3	7.77	4.46	6.85	10.99	16.45	26.17
Mobile Pose	mobile_pose_mobilenetv3	590	209	19.0	1.55	1.29	1.92	3.13	5.71	11.62
ResNet	resnet18	602	4	46.7	3.81	1.27	1.86	2.73	4.06	7.02
	resnet50	602	4	102.3	8.33	2.61	3.78	5.61	9.13	15.67
	resnet152	602	4	240.9	19.58	7.71	11.14	16.21	26.48	44.60

Clockwork – Predictable DNN Worker

- Memory management
  - Managed memory and caches can be unpredictable
  - Memory used as cache introduces variability between cache hits and misses
  - GPU memory not large enough to hold all models
  - Host-to-GPU memory transfer slower than DNN inference execution
  - Clockwork treats GPU memory as cache
  - Workers expose LOAD and UNLOAD actions to controller for copying models to and removing models from the GPU memory with deterministic latency

Clockwork – Predictable DNN Worker

- Inference Execution
  - Hardware interactions can be unpredictable
  - Controller only sends INFER request when model is loaded in GPU
  - INFER broken down into 3 steps:
    - INPUT
    - EXEC
    - OUTPUT
  - While INPUT and OUTPUT might coincide, only a single EXEC is run at a time
  - Avoids unpredictable behavior caused by multiple concurrent EXECs

Clockwork – Predictable DNN Worker

- Interface with controller
  - External factors can trigger performance variance
  - Controller sends detailed timing instructions with each command

`type`	INFER, LOAD, or UNLOAD
`earliest`	the time when this action may begin executing
`latest`	when this action will be rejected

  - Actions that can not start within window are cancelled
  - Allows workers to quickly get back on schedule after unexpected delay
  - Worker sends the result of each action back to the controller

Clockwork – Central Controller

- Modelling worker performance
  - Controller maintains per-worker per-model performance profile
  - Profiles updated continuously based to response from worker
  - Also tracks actions and memory state at every worker

Clockwork – Central Controller

- Action scheduler
  - Proactively manage action schedules for workers
  - Estimate tight but realistic ‘earliest’ and ‘latest’ time intervals
  - Balance load by mixing ‘hot’ and ‘cold’ inferences among workers

Clockwork – Implementation

- Models
- DNN workers
- Central controller

Clockwork – Implementation

- Models
  - Weights and kernels
  - Memory metadata
  - Profiling models

Clockwork – Implementation

- DNN workers
  - Managing model weights in memory
    - Workspace, IOCache and PageCache
    - Actions and results

Clockwork – Implementation

- Central controller
  - Managing worker state
  - Scheduling INFER requests
  - Scheduling LOAD requests

Clockwork – Implementation

- Central controller
  - Managing worker state
    - Memory state
    - Action profiles
    - Pending actions

Clockwork – Implementation

- Central controller
  - Scheduling INFER requests
    - Each model has a request queue per batch size
      - New requests queued into every batch queue

Model Family	Model	IO Size (kB)		Weights		GPU Execution Latency (ms)				
		Input	Output	Size (MB)	Transfer (ms)	B1	B2	B4	B8	B16
DenseNet	densenet169	602	4	56.5	4.50	5.18	6.29	8.57	12.82	21.85
Inception v3	inceptionv3	1073	4	95.3	7.77	4.46	6.85	10.99	16.45	26.17
Mobile Pose	mobile_pose_mobilenetv3	590	209	19.0	1.55	1.29	1.92	3.13	5.71	11.62
ResNet	resnet18	602	4	46.7	3.81	1.27	1.86	2.73	4.06	7.02
	resnet50	602	4	102.3	8.33	2.61	3.78	5.61	9.13	15.67
	resnet152	602	4	240.9	19.58	7.71	11.14	16.21	26.48	44.60

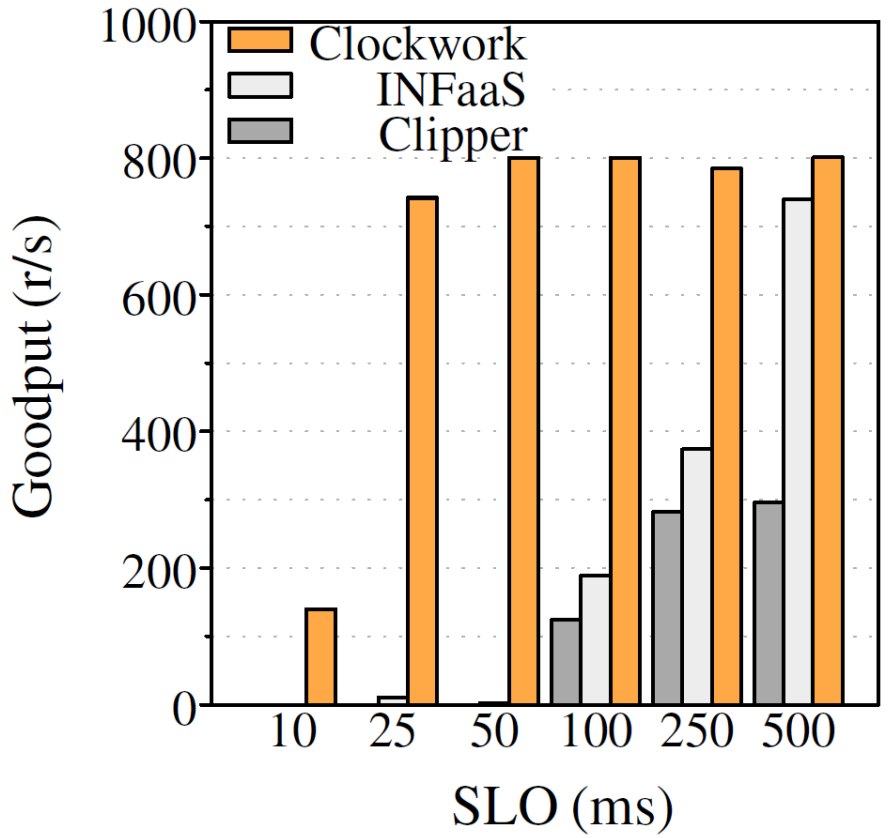
Clockwork – Implementation

- Central controller
  - Scheduling LOAD requests
    - Maintains *load* and *demand* statistics for each model and GPU

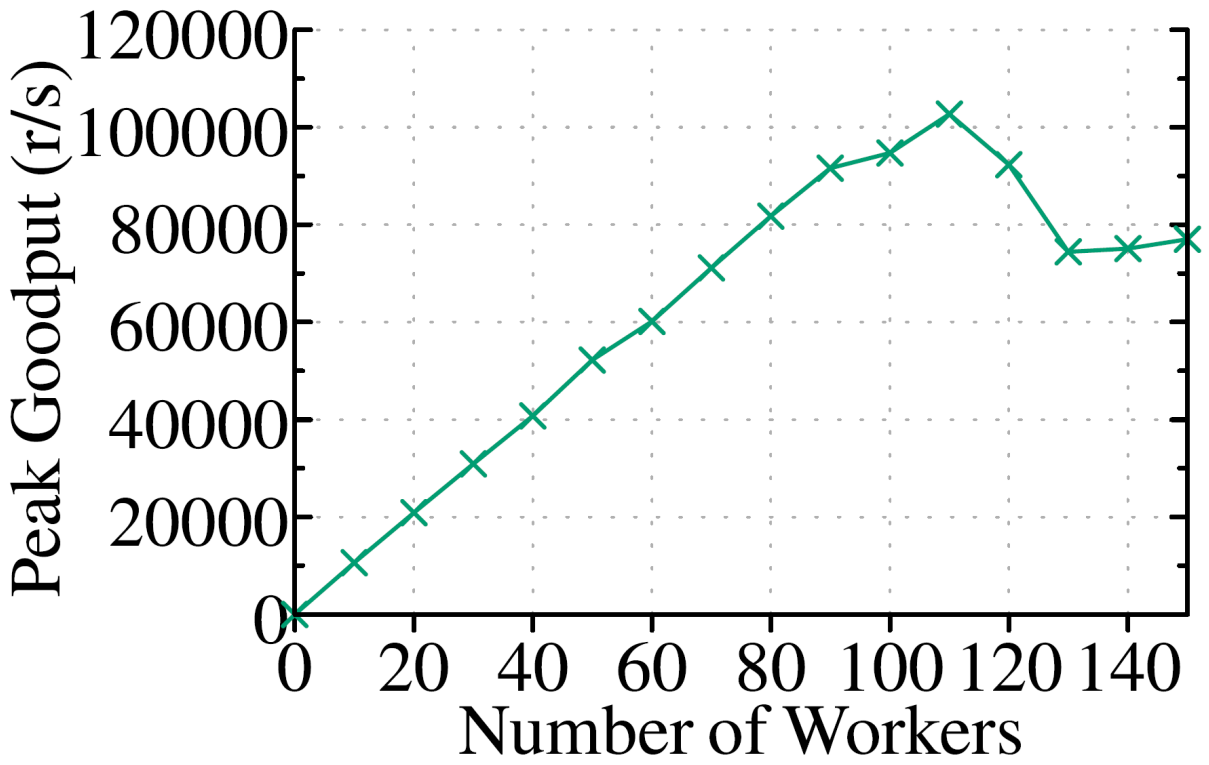
d_m the total demand for each model m
$a_{m,g}$ the demand allocation of model m on GPU g
$\ell_g = \sum_m a_{m,g}$ the total load on each GPU g

$$p_{m,g} = d_m - \sum_g a_{m,g} \cdot \frac{\text{capacity}_g}{\ell_g}$$

Clockwork – Comparison



Clockwork – Scaling



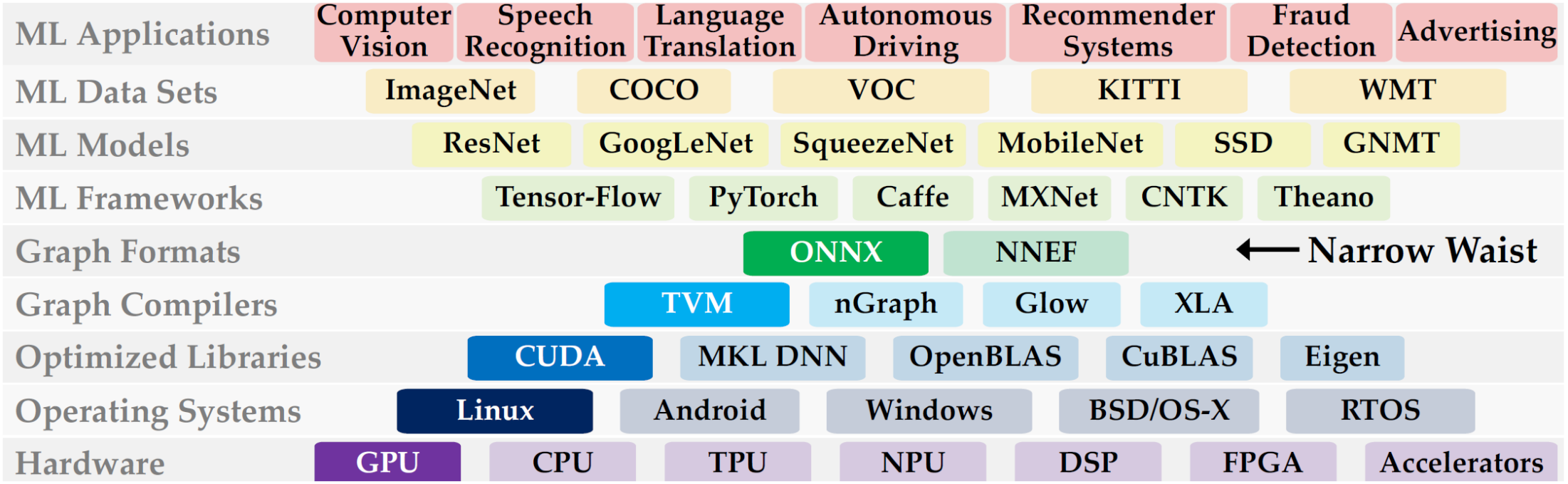
Clockwork – Performance

- Replay workload trace of Microsoft Azure Functions (MAF)
- SLO target: 100 ms
- Goodput average: 9638 r/s
- Total requests: 208,000,000
- Failed requests: 58

Model Family	Count	Model Variants
DenseNet [36]	4	121, 161, 198, 201
DLA [75]	1	34
GoogLeNet [67]	1	
Inception [68]	1	v3
Xception [68]	1	
MobilePose [33]	4	SPRN18, MNv3, RN18, RN50
ResNeSt [78]	4	14, 26, 40, 101
ResNet [30]	22	18, 18b, 34, 34b, 50, 50b, 50c, 50d, 50s, 50-1.8x, 101, 101b, 101c, 101d, 101s, 101-1.9x, 101-2.2x, 152, 152b,152c, 152d, 152s
ResNet-v2 [31]	5	18, 34, 50, 101, 152
ResNeXt [73]	3	50-32, 101-32, 101-64
SENet [35]	2	50-32, 101-32
TSN [70]	7	iv1, iv3, r18, r34, r50, r101, r152
Wide ResNet [76]	3	16-10, 28-10, 40-8
Winograd [45]	3	RN18, RN50, RN101

List of models used in experiments.

Model Serving Systems – Narrow Waist Stack



Clockwork – HummingBird

- Scikit-learn is used about 5 times more than PyTorch and TensorFlow combined
- Clockwork is limited to DNNs
- HummingBird converts traditional ML models into ONNX format
- Scope of Clockwork can be extended to cover traditional ML models

Clockwork – Piazza

- Batching inferences

> Hi, I have some questions for the Clockwork paper. I am a little confused about how batching inference requests works. To my understanding, the batch size of a model is pre-defined but when scheduling INFER actions, the paper mentions that 'each model has a request queue per batch size'. Does that mean we can change the batch size of a model? How does that happen?

  - The batch size is the number of inputs appended for parallel execution
  - Models don’t explicitly have a batch size
  - Clockwork predicts the best batching strategy which allows for maximum parallel execution without violating any SLO constraints