Clipper

A Low-Latency Online Prediction Serving System

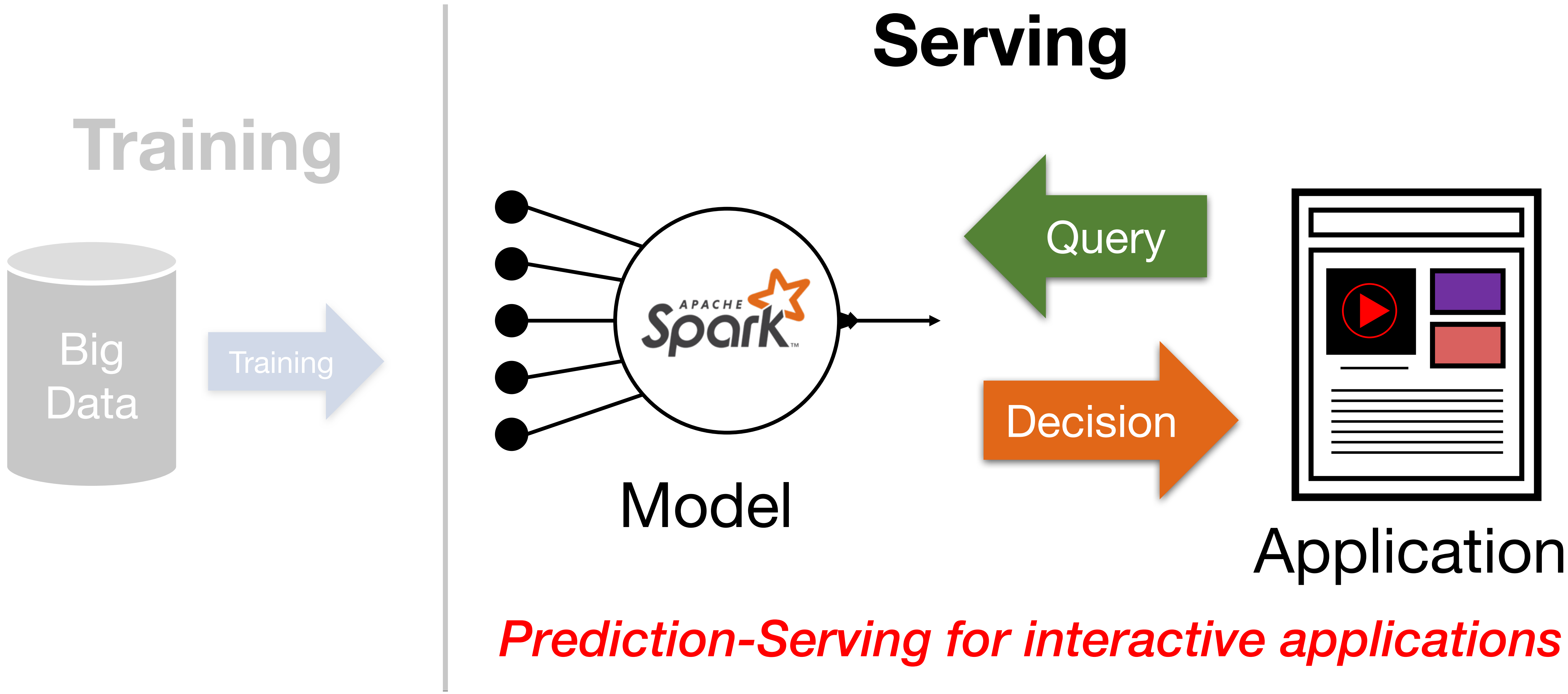
Dan Crankshaw

crankshaw@cs.berkeley.edu

http://clipper.ai

https://github.com/ucbrise/clipper

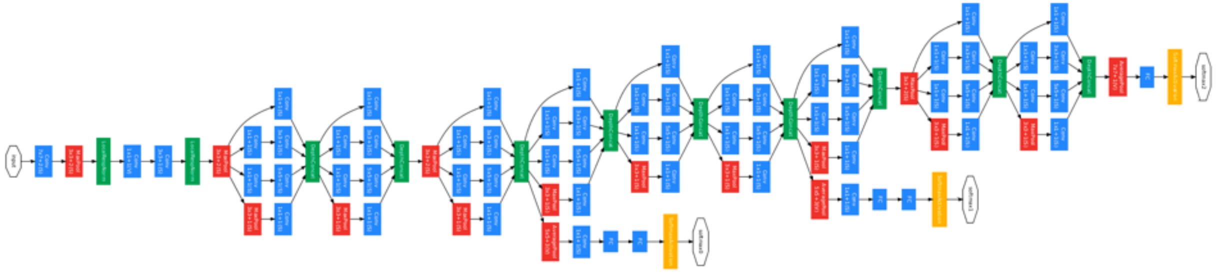
December 8, 2017



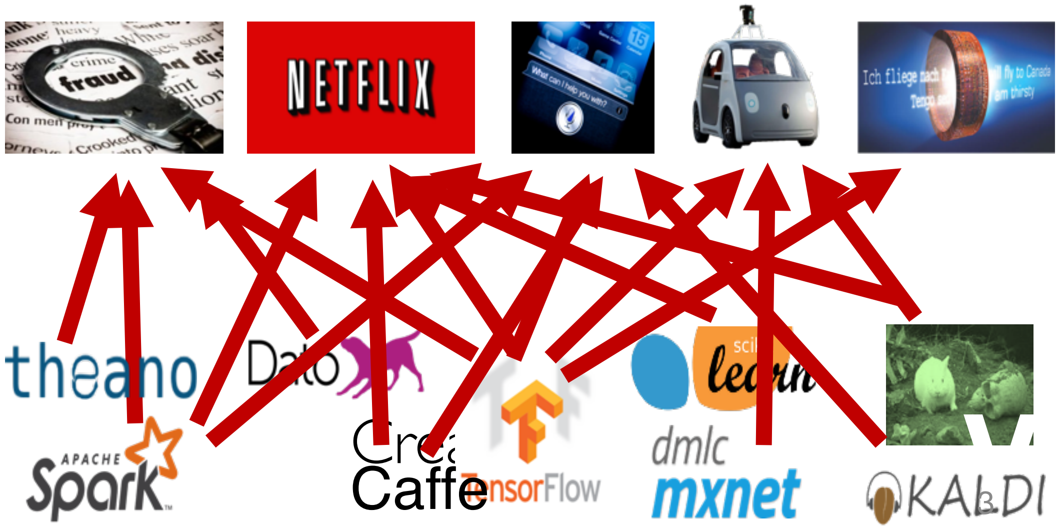
Prediction-Serving for interactive applications

Timescale: ~10s of milliseconds

Prediction-Serving Challenges

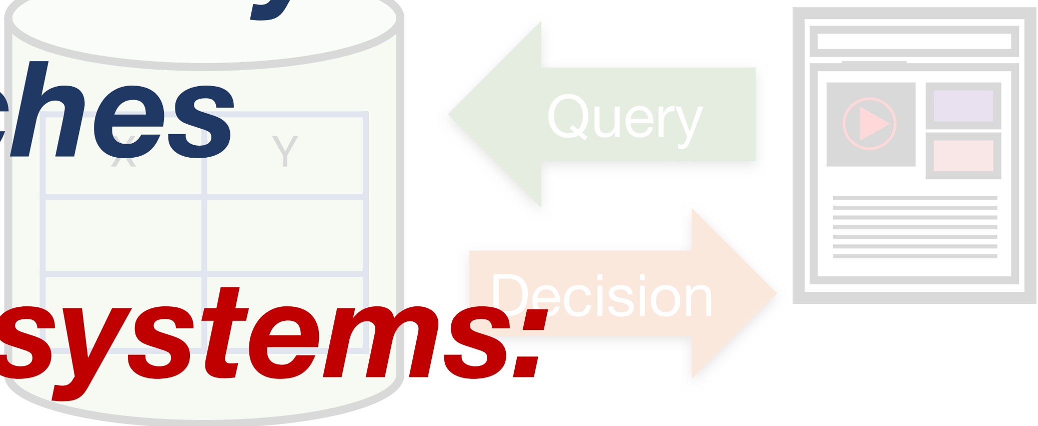


Support low-latency, high-throughput serving workloads



Large and growing ecosystem of ML models and frameworks

Prediction-Serving Today



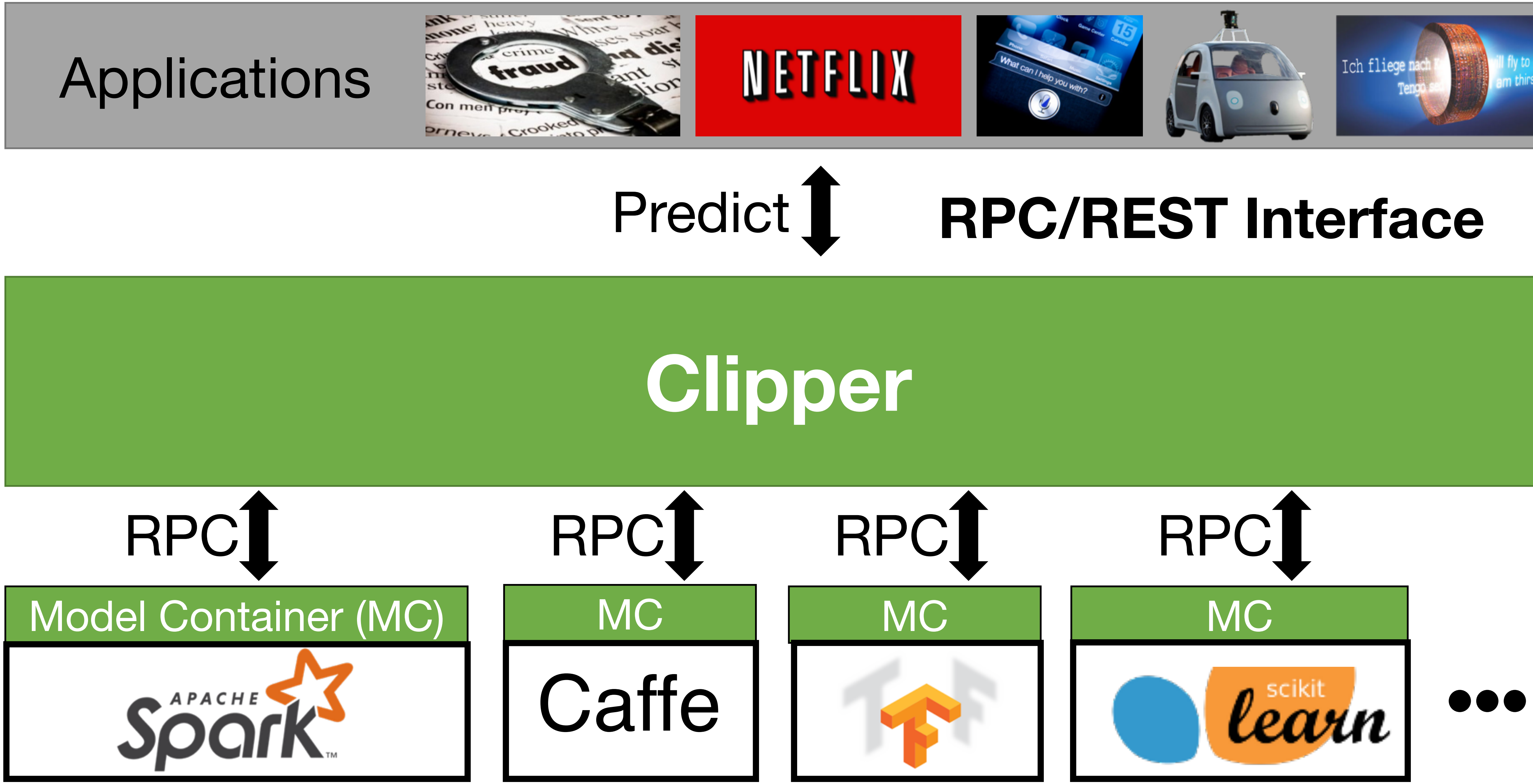
Clipper aims to unify these approaches

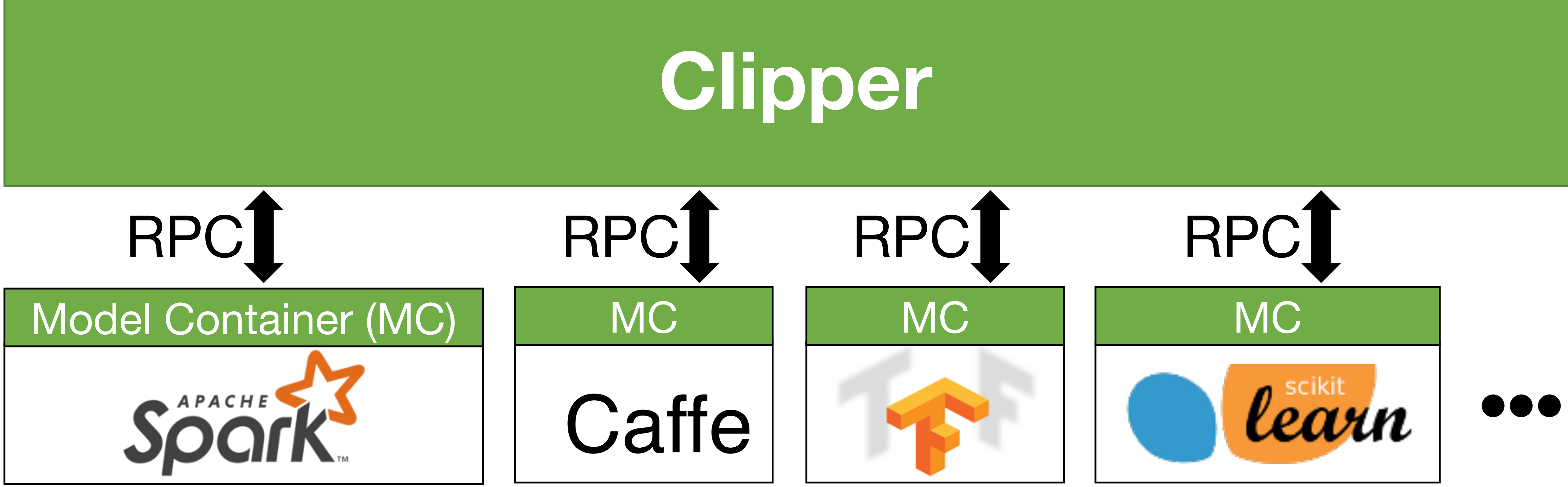
New class of systems: Prediction-Serving Systems

Highly specialized systems for specific problems

Offline scoring with existing frameworks and systems

Clipper Decouples Applications and Models

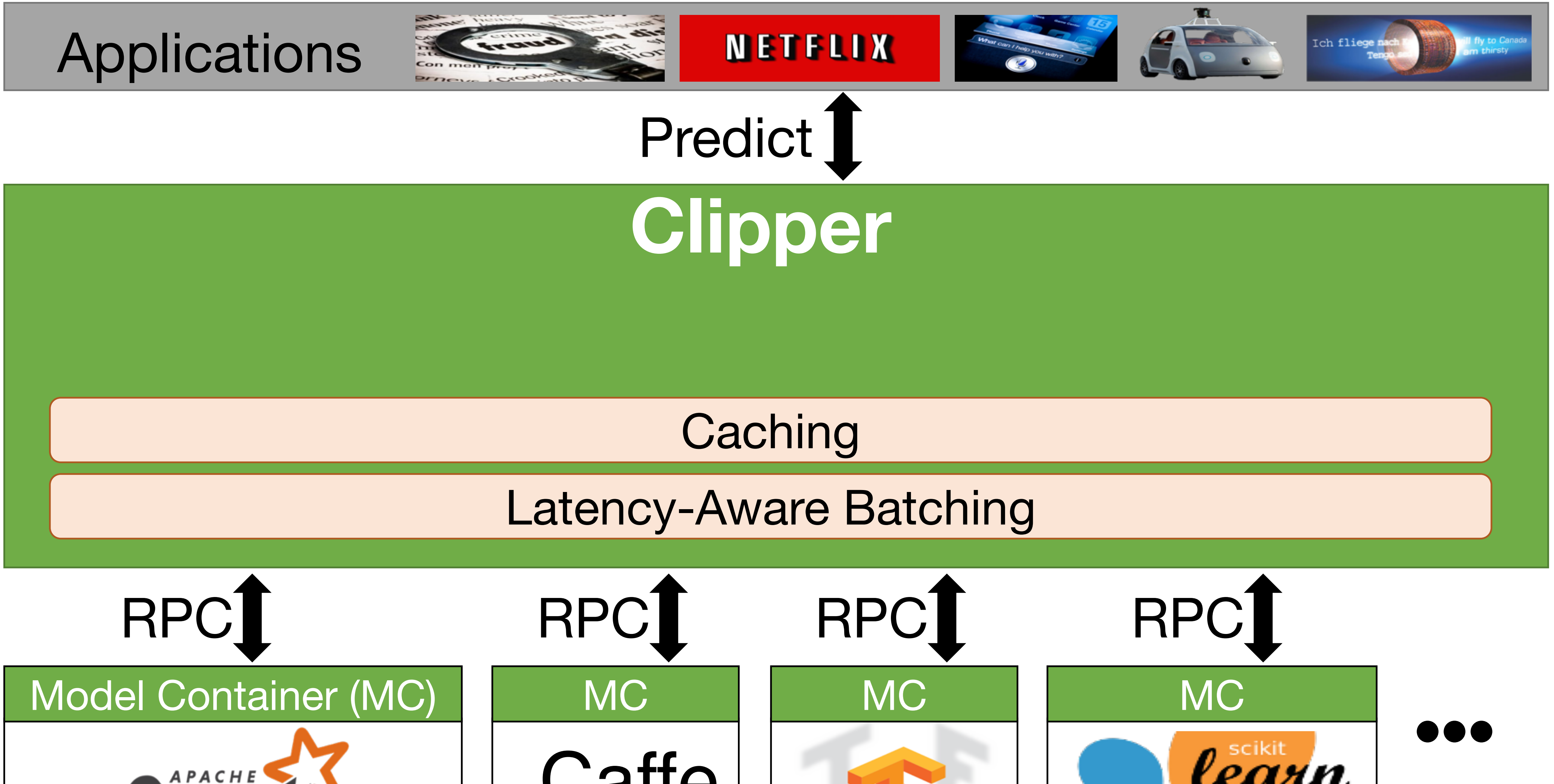




Common Interface → Simplifies Deployment:

- Evaluate models using original code & systems
- Models run in separate processes as Docker containers
  - Resource isolation: Cutting edge ML frameworks can be buggy
  - Scale-out and deployment on Kubernetes

Clipper Architecture



Status of the project

https://github.com/ucbrise/clipper

- First released in May 2017 with a focus on usability
- Currently working towards 0.3 release and actively working with early users
  - Focused on performance improvements and better monitoring and stability
- Supports native deployments on Kubernetes and a local Docker mode
- Goal: Community-owned platform for model deployment and serving
  - Post issues and questions on GitHub and subscribe to our mailing list clipper-dev@googlegroups.com

Simplifying Model Deployment with Clipper

Getting Started with Clipper is Easy

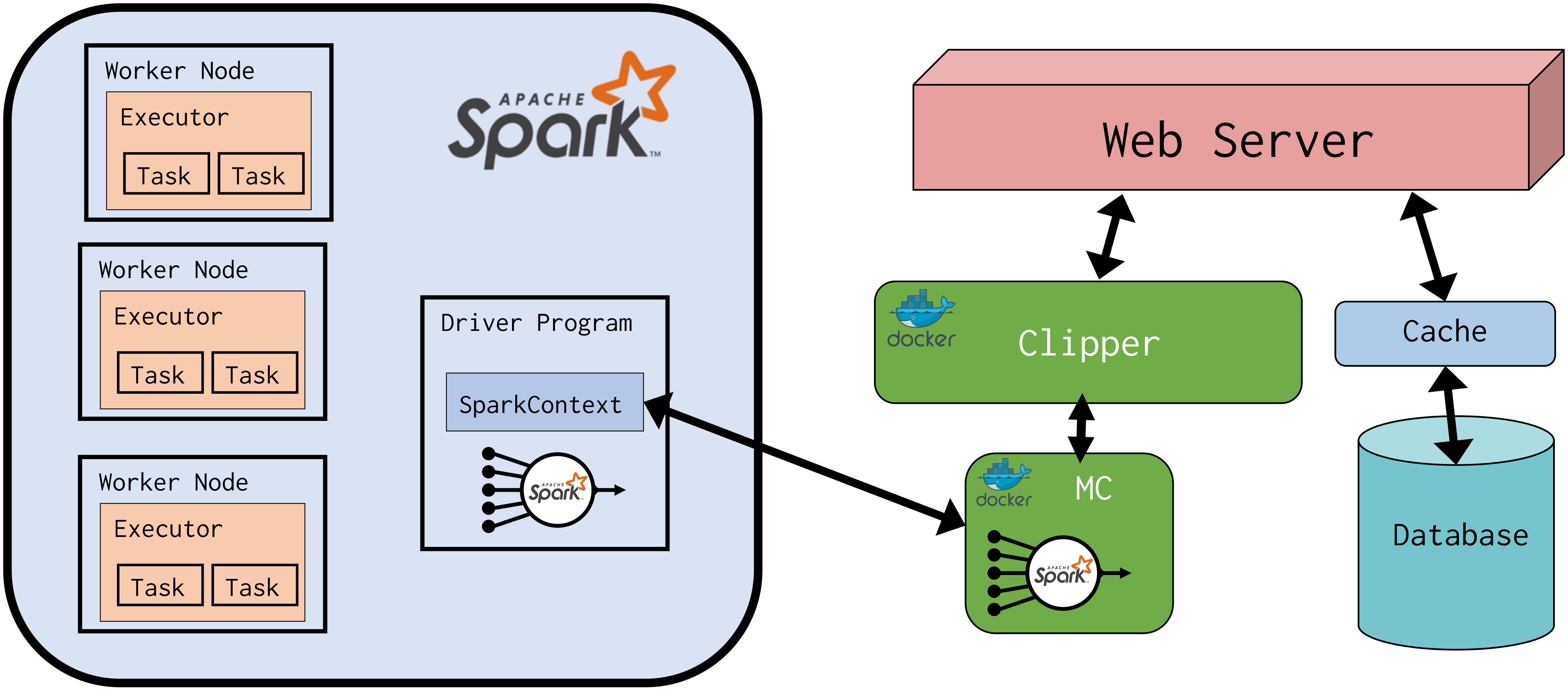
Docker images available on DockerHub

Clipper admin is distributed as pip package:

```
pip install clipper_admin
```

Get up and running without cloning or compiling!

Clipper Connects Training and Serving



Problem: Models don’t run in isolation

Must extract model plus pre- and post-processing logic

Clipper provides a library of model deployers

- ***Deployer automatically and intelligently saves all prediction code***
  - *Captures both framework-specific models and arbitrary serializable code*

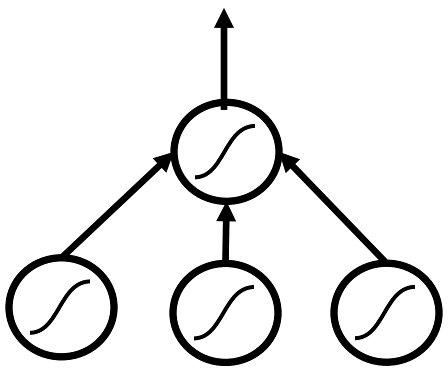
- ***Replicates required subset of training environment and loads prediction code in a Clipper model container***

Clipper provides a (growing) library of model deployers

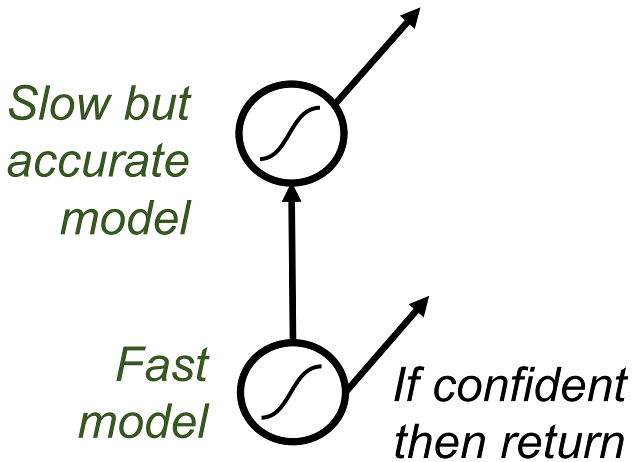
- ***Python***
  - Combine framework specific models with external featurization, post-processing, business logic
  - Currently support Scikit-Learn, PySpark, TensorFlow
  - PyTorch, Caffe2, XGBoost coming soon
- ***Scala and Java with Spark:***
  - both MLLib and Pipelines APIs
- ***Arbitrary R functions***

Ongoing Research

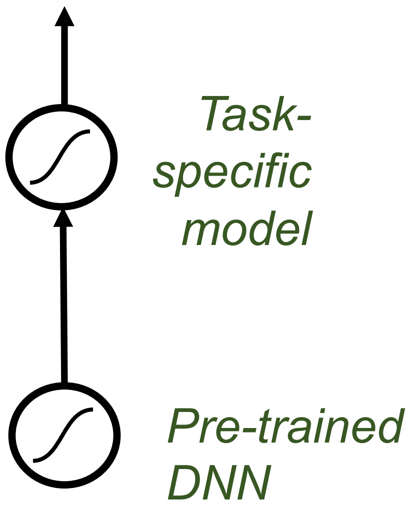
Supporting Modular Multi-Model Pipelines



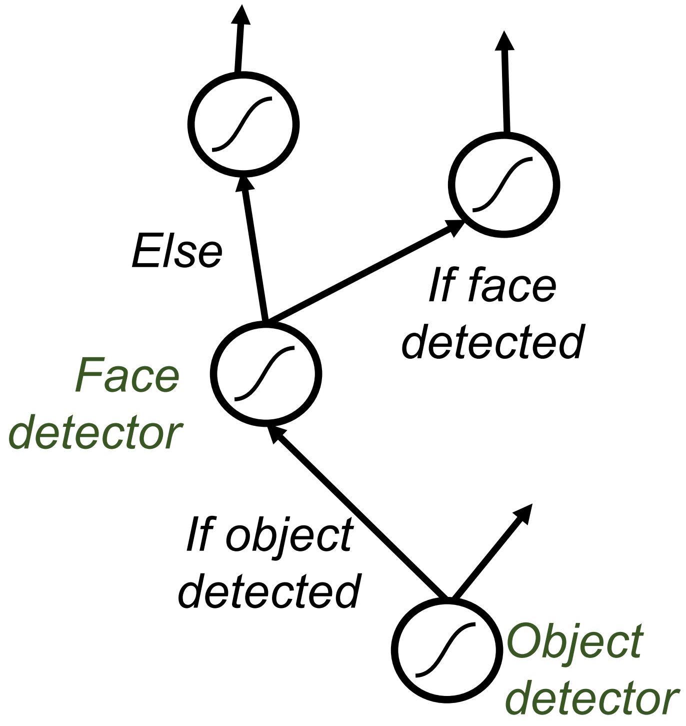
Ensembles can improve accuracy



Faster inference with prediction cascades



Faster development through model-reuse



Model specialization

How to efficiently support serving arbitrary model pipelines?

Challenges of Serving Model Pipelines

- Complex tradeoff space of latency, throughput, and monetary cost
  - Many serving workloads are interactive and highly latency-sensitive
  - Performance and cost depend on model, workload, and physical resources available
- Model composition leads to combinatorial explosion in the size of the tradeoff space
  - Developers must make decisions about how to configure individual models while ***reasoning about end-to-end pipeline performance***

Solution: Workload-Aware Optimizer

- Exploit structure and properties of inference computation
  - Immutable state
  - Query-level parallelism
  - Compute-intensive
- Pipeline definition
  - Intermingle arbitrary application code and Clipper-hosted model evaluation for maximum flexibility
- Optimizer input
  - Pipeline, sample workload, and performance or cost constraints
- Optimizer output
  - Optimal pipeline configuration that meets constraints
- Deployed models use Clipper as physical execution engine for serving

Conclusion

- Challenges of serving increasingly *complex models* trained in *variety of frameworks* while meeting *strict performance demands*
- Clipper adopts a *container-based architecture* and employs prediction caching and *latency-aware batching*
- Clipper's model deployer library makes it easy to deploy both *framework-specific models* and *arbitrary processing code*
- Ongoing efforts on a *workload-aware optimizer* to optimize the deployment of complex, *multi-model pipelines*

`http://clipper.ai`