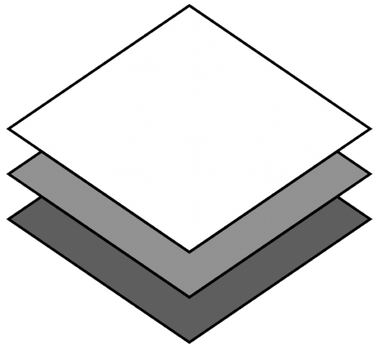


Testing & Deployment

Josh Tobin, **Sergey Karayev**, Pieter Abbeel

Amazon SageMaker · Determined AI · Neptune Machine Learning Lab · H2O · FLOYD · DOMINO DATA LAB

"All-in-one"

Data

Versioning: Quilt, DVC, Pachyderm, LIQUIDATA

Labeling: figure eight, scale, prodigy

Processing: PY, Luigi

Database

Storage: NFS, S3

or

Development

Distributed Training: PYTORCH, RAY, HOROVOD

Frameworks: PYTORCH, K, fast.ai

Software Engineering: jupyter, git

Training/Evaluation

Hyperparameter Tuning: SIGOPT, tune

Experiment Management: Weights & Biases, comet, TensorBoard, LOSSWISE

Resource Management: Polyaxon, PY, slurm workload manager

or

Deployment

Monitoring: ?

Web: ALGORITHMIA, SELDON

Hardware / Mobile: ML, NVIDIA TensorRT, TensorFlowLite

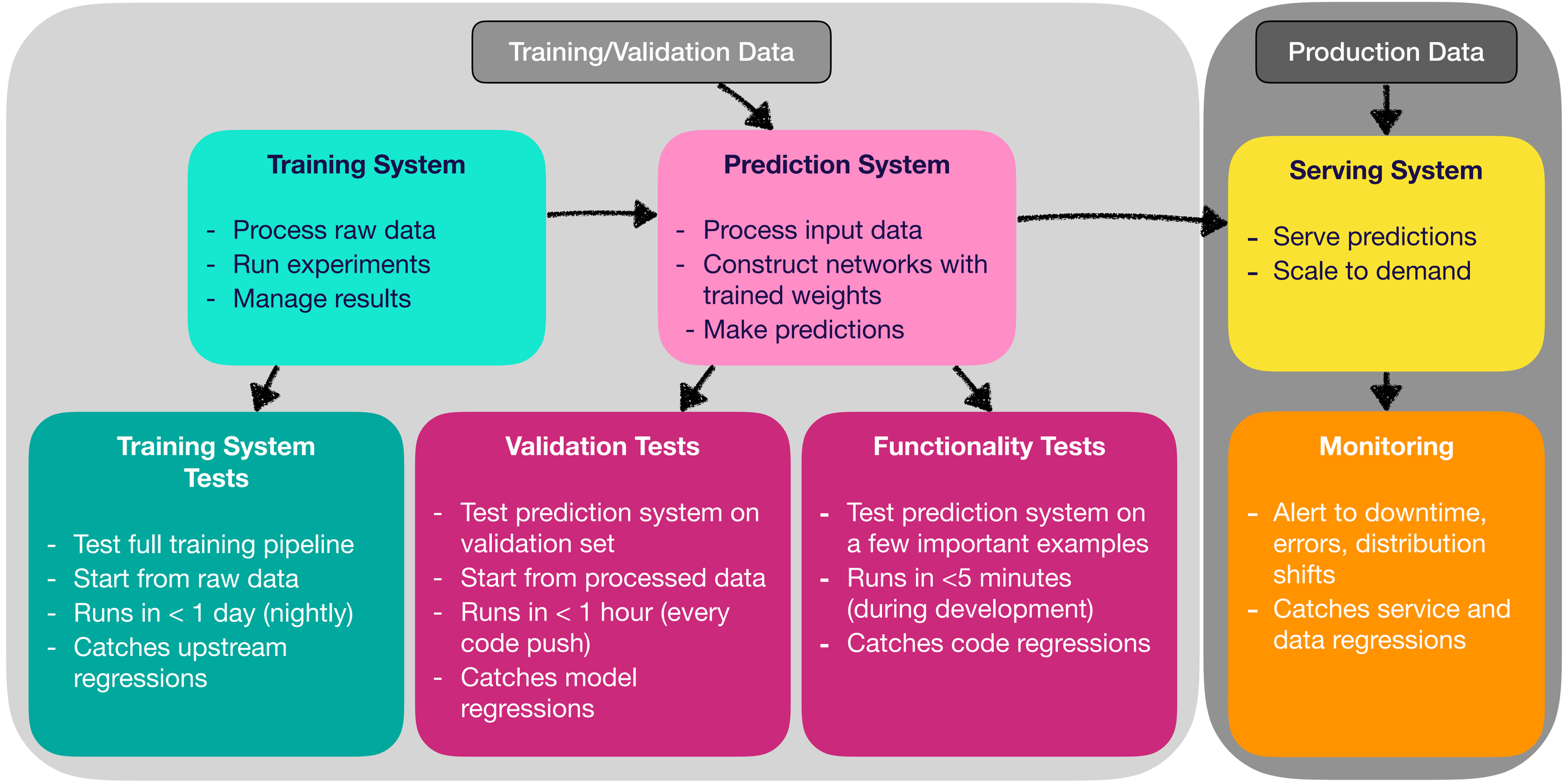
Interchange: ONNX

CI / Testing: Buildkite, ?

or or

Module Overview

- Concepts
  - Project structure
  - ML Test Score: a rubric for production readiness
- Infrastructure/Tooling
  - CI/Testing
  - Docker in more depth
  - Deploying to the web
  - Monitoring
  - Deploying to hardware/mobile


Training/Validation Data
Production Data
Training System
- Process raw data
- Run experiments
- Manage results
Prediction System
- Process input data
- Construct networks with trained weights
- Make predictions
Serving System
- Serve predictions
- Scale to demand
Training System Tests
- Test full training pipeline
- Start from raw data
- Runs in < 1 day (nightly)
- Catches upstream regressions
Validation Tests
- Test prediction system on validation set
- Start from processed data
- Runs in < 1 hour (every code push)
- Catches model regressions
Functionality Tests
- Test prediction system on a few important examples
- Runs in <5 minutes (during development)
- Catches code regressions
Monitoring
- Alert to downtime, errors, distribution shifts
- Catches service and data regressions

The ML Test Score:
A Rubric for ML Production Readiness and Technical Debt Reduction

IEEE Big Data 2017

Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley
Google, Inc.
ebreck, cais, nielsene, msalib, dsculley@google.com

An exhaustive framework/checklist from practitioners at Google

Follow-up to previous work from Google

Machine Learning: The High-Interest Credit Card of Technical Debt

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young
{dsculley,gholt,dgg,edavydov}@google.com
{toddphillips,ebner,vchaudhary,mwyoung}@google.com
Google, Inc

Rules of Machine Learning: Best Practices for ML Engineering

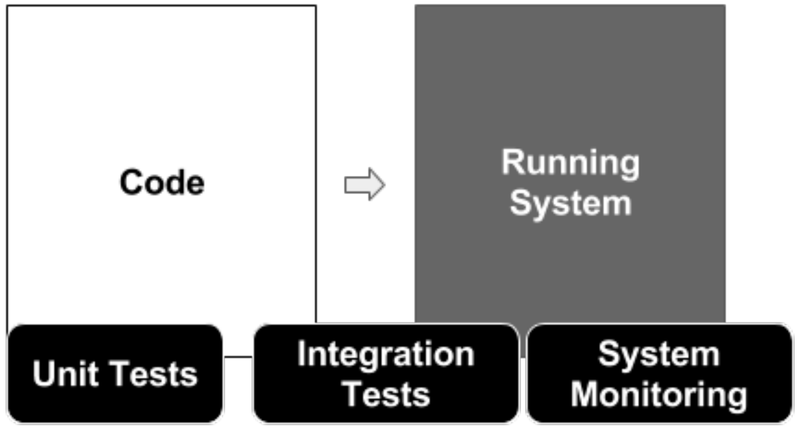
Martin Zinkevich

Hidden Technical Debt in Machine Learning Systems

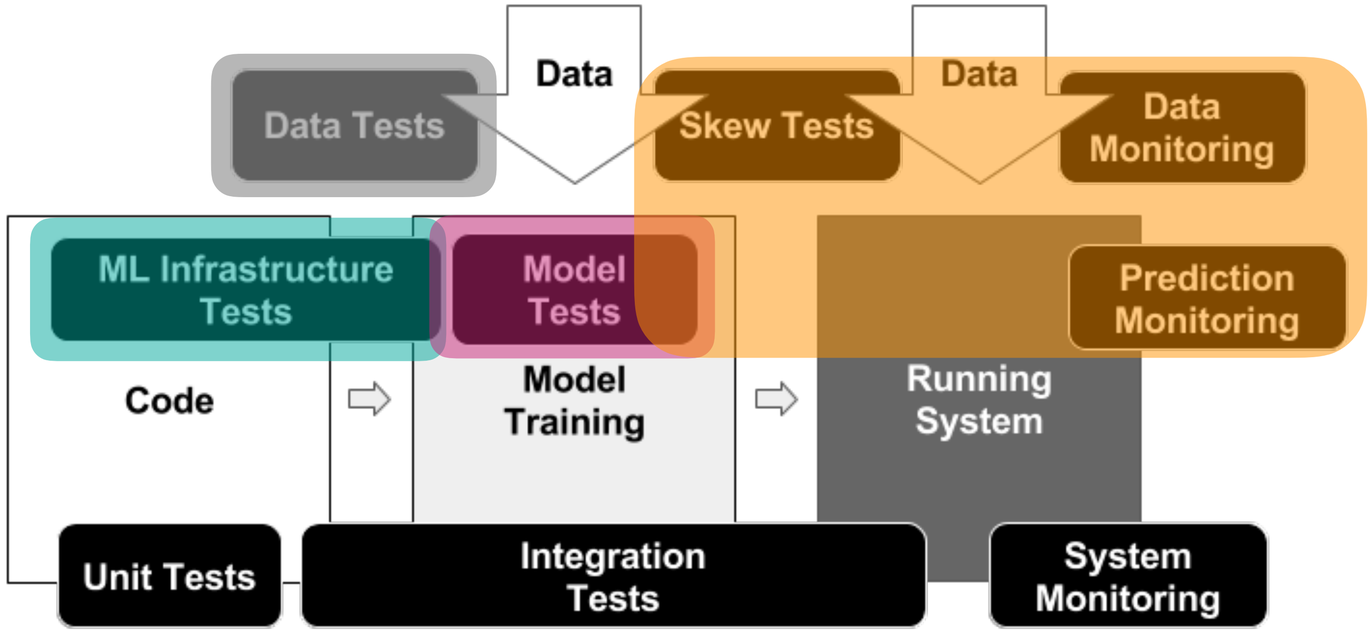
D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips
{dsculley,gholt,dgg,edavydov,toddphillips}@google.com
Google, Inc.

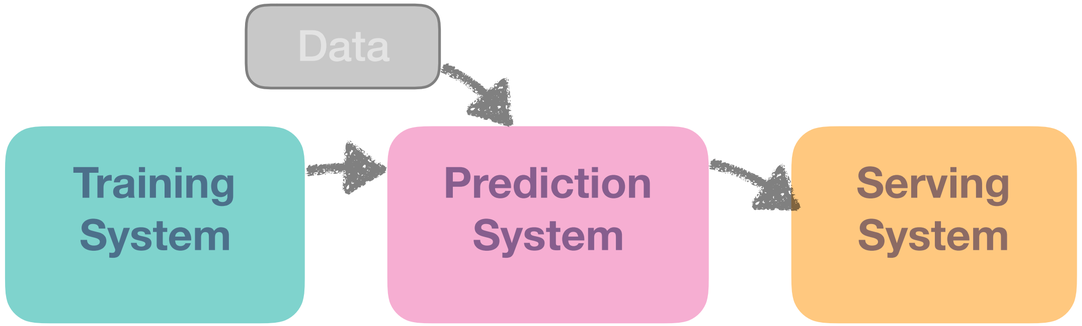
Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, Dan Dennison
{ebner,vchaudhary,mwyoung,jfcrespo,dennison}@google.com
Google, Inc.

Traditional Software



Machine Learning Software





1	Feature expectations are captured in a schema.
2	All features are beneficial.
3	No feature's cost is too much.
4	Features adhere to meta-level requirements.
5	The data pipeline has appropriate privacy controls.
6	New features can be added quickly.
7	All input feature code is tested.

Data Tests

1	Model specs are reviewed and submitted.
2	Offline and online metrics correlate.
3	All hyperparameters have been tuned.
4	The impact of model staleness is known.
5	A simpler model is not better.
6	Model quality is sufficient on important data slices.
7	The model is tested for considerations of inclusion.

Model Tests

1	Training is reproducible.
2	Model specs are unit tested.
3	The ML pipeline is Integration tested.
4	Model quality is validated before serving.
5	The model is debuggable.
6	Models are canaried before serving.
7	Serving models can be rolled back.

ML Infrastructure Tests

1	Dependency changes result in notification.
2	Data invariants hold for inputs.
3	Training and serving are not skewed.
4	Models are not too stale.
5	Models are numerically stable.
6	Computing performance has not regressed.
7	Prediction quality has not regressed.

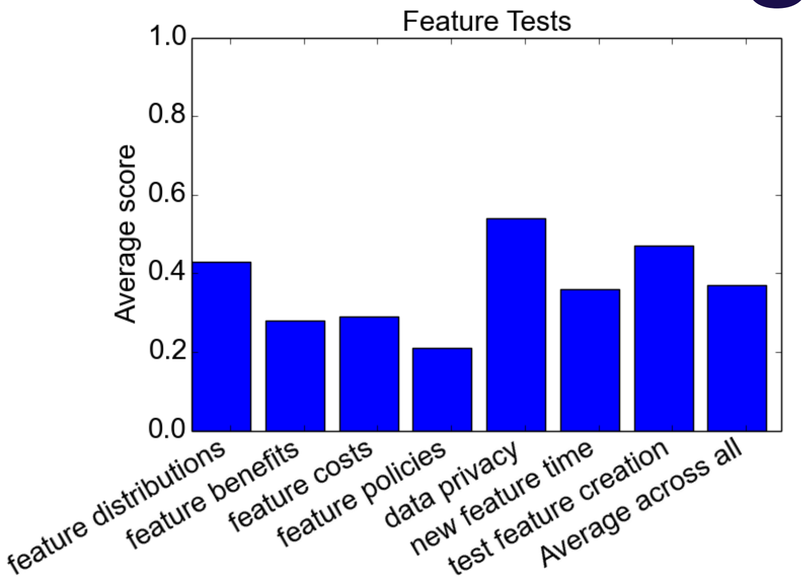
Monitoring Tests

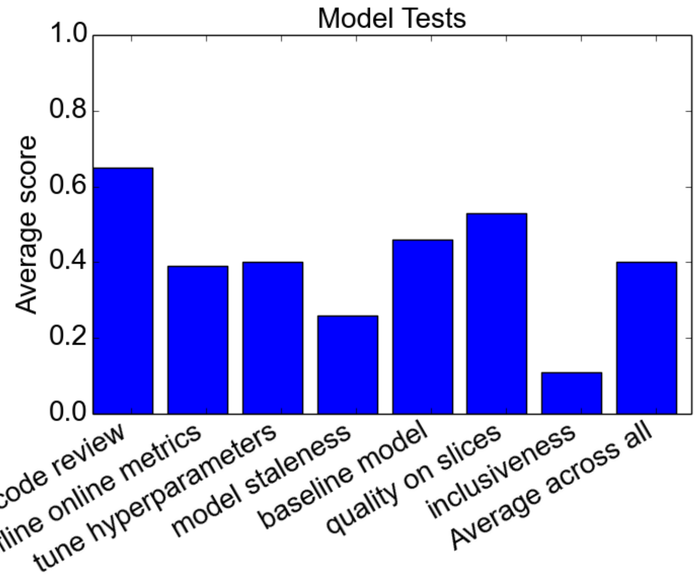
Scoring the Test

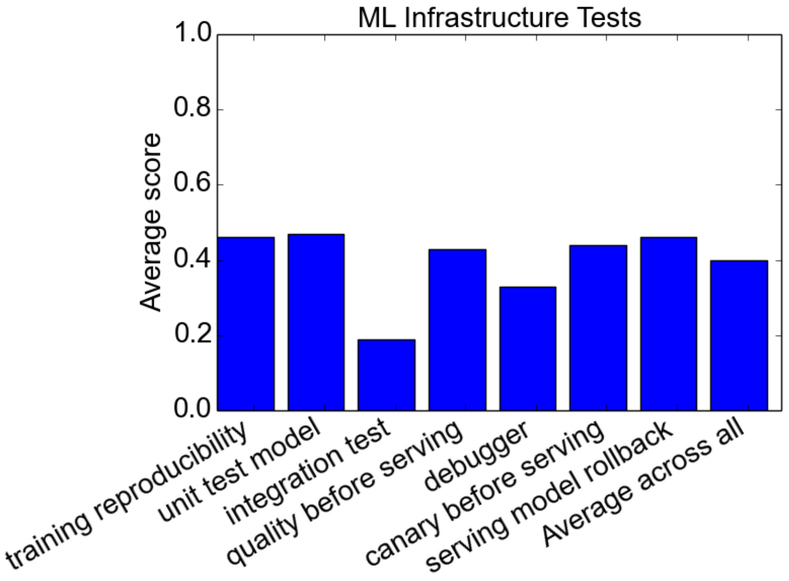
Points	Description
0	More of a research project than a productionized system.
(0,1]	Not totally untested, but it is worth considering the possibility of serious holes in reliability.
(1,2]	There's been first pass at basic productionization, but additional investment may be needed.
(2,3]	Reasonably tested, but it's possible that more of those tests and procedures may be automated.
(3,5]	Strong levels of automated testing and monitoring, appropriate for mission-critical systems.
> 5	Exceptional levels of automated testing and monitoring.

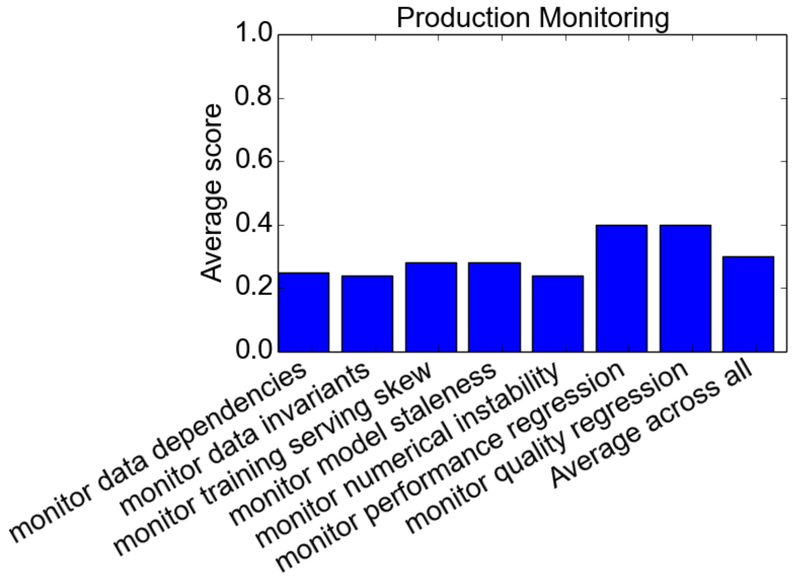
Results at Google

- 36 teams with ML systems in production at Google









Questions?

Amazon SageMaker · Determined AI · Neptune Machine Learning Lab · H2O · FLOYD · DOMINO DATA LAB

"All-in-one"

Data

Versioning: Quilt, DVC, Pachyderm, LIQUIDATA

Labeling: figure eight, scale, prodigy

Processing: PY, Luigi

Database

Storage: NFS, S3

or

Development

Distributed Training: PYTORCH, RAY, HOROVOD

Frameworks: PYTORCH, K, fast.ai

Software Engineering: jupyter, git

Training/Evaluation

Hyperparameter Tuning: SIGOPT, tune

Experiment Management: Weights & Biases, comet, TensorBoard, LOSSWISE

Resource Management: Polyaxon, PY, slurm workload manager

or

Deployment

Monitoring: ?

Hardware / Mobile: ML, NVIDIA TensorRT, TensorFlow Lite

Web: ALGORITHMIA, SELDON

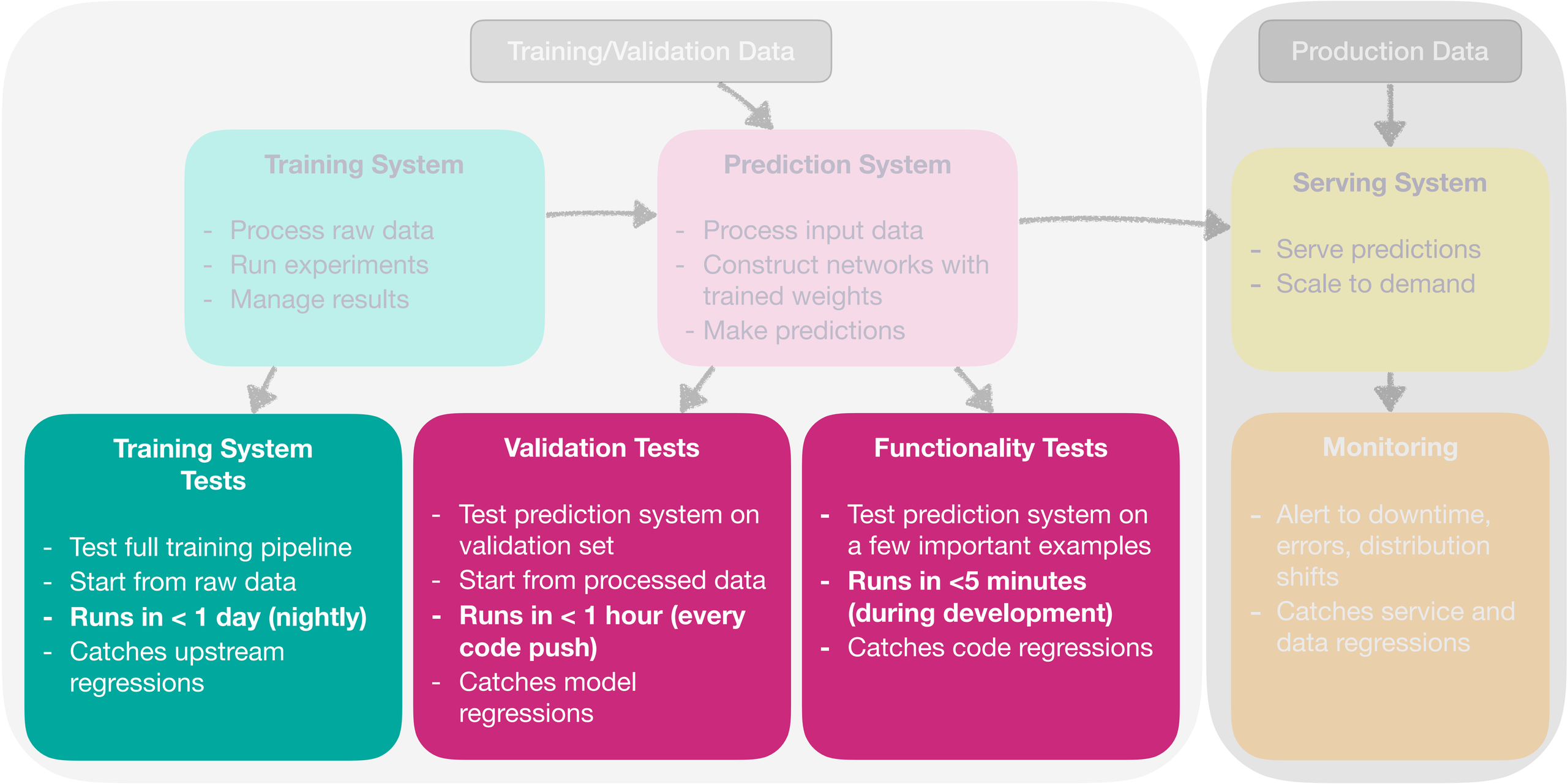
Interchange: ONNX

CI / Testing: Buildkite, ?

or or

Testing / CI

- **Unit / Integration Tests**
  - Tests for individual module functionality and for the whole system
- **Continuous Integration**
  - Tests are run every time new code is pushed to the repository, before updated model is deployed.
- **SaaS for CI**
  - CircleCI, Travis, Jenkins, Buildkite
- **Containerization (via Docker)**
  - A self-enclosed environment for running the tests


Training/Validation Data
Production Data
Training System
- Process raw data
- Run experiments
- Manage results
Prediction System
- Process input data
- Construct networks with trained weights
- Make predictions
Serving System
- Serve predictions
- Scale to demand
Training System Tests
- Test full training pipeline
- Start from raw data
- Runs in < 1 day (nightly)
- Catches upstream regressions
Validation Tests
- Test prediction system on validation set
- Start from processed data
- Runs in < 1 hour (every code push)
- Catches model regressions
Functionality Tests
- Test prediction system on a few important examples
- Runs in <5 minutes (during development)
- Catches code regressions
Monitoring
- Alert to downtime, errors, distribution shifts
- Catches service and data regressions

1	Feature expectations are captured in a schema.
2	All features are beneficial.
3	No feature's cost is too much.
4	Features adhere to meta-level requirements.
5	The data pipeline has appropriate privacy controls.
6	New features can be added quickly.
7	All input feature code is tested.

Data Tests

1	Model specs are reviewed and submitted.
2	Offline and online metrics correlate.
3	All hyperparameters have been tuned.
4	The impact of model staleness is known.
5	A simpler model is not better.
6	Model quality is sufficient on important data slices.
7	The model is tested for considerations of inclusion.

Model Tests

1	Training is reproducible.
2	Model specs are unit tested.
3	The ML pipeline is Integration tested.
4	Model quality is validated before serving.
5	The model is debuggable.
6	Models are canaried before serving.
7	Serving models can be rolled back.

ML Infrastructure Tests

1	Dependency changes result in notification.
2	Data invariants hold for inputs.
3	Training and serving are not skewed.
4	Models are not too stale.
5	Models are numerically stable.
6	Computing performance has not regressed.
7	Prediction quality has not regressed.

Monitoring Tests

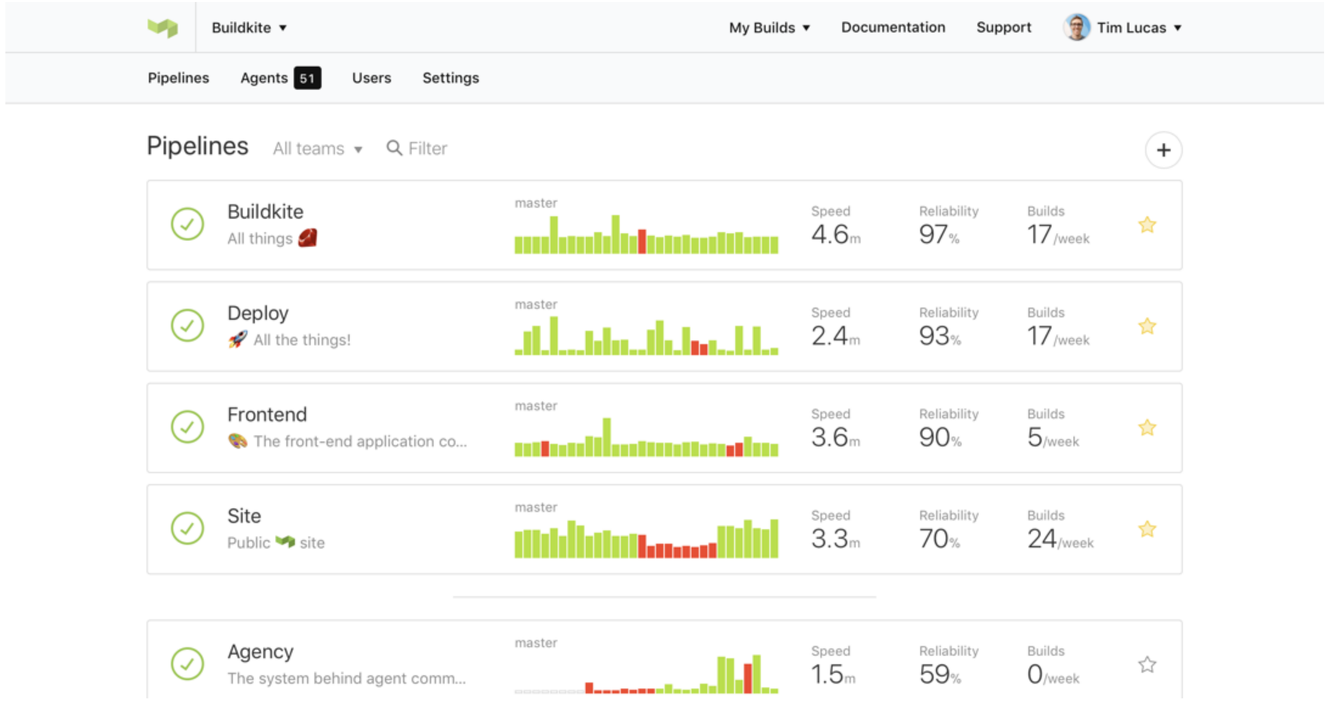
CircleCI / Travis / etc

- SaaS for continuous integration
- Integrate with your repository: every push kicks off a cloud job
- Job can be defined as commands in a Docker container, and can store results for later review
- No GPUs
- CircleCI has a free plan

Jenkins / Buildkite

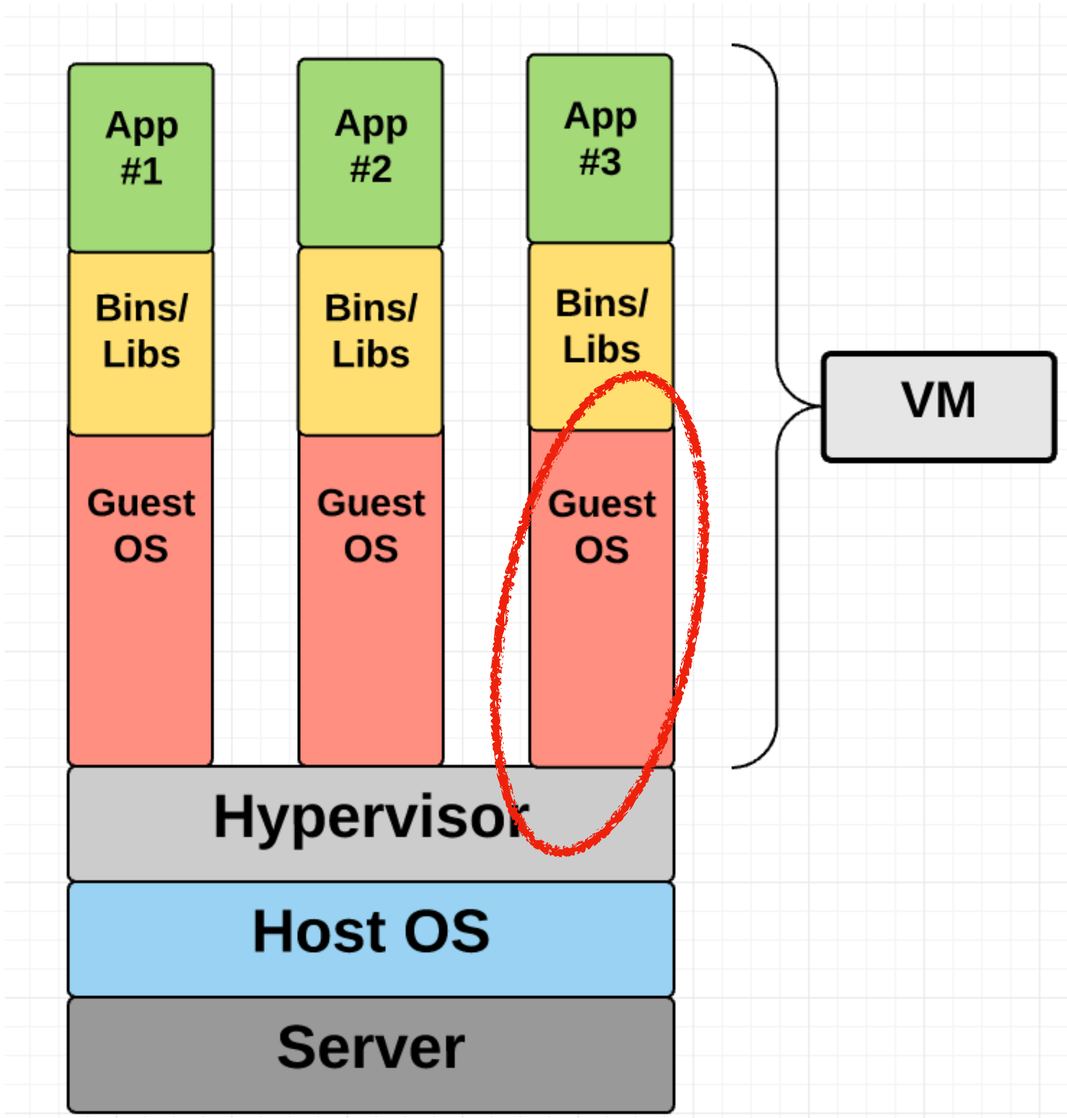
- Nice option for running CI on your own hardware, in the cloud, or mixed
- Good option for a scheduled training test (can use your GPUs)
- Very flexible

```
$ buildkite-agent start
Starting buildkite-agent with PID: 11301
Registering agent with Buildkite...
Successfully registered agent "docker-builder-1
Waiting for work...
Assigned job 69f662c7-b990-489e-8e77-8e02550
```

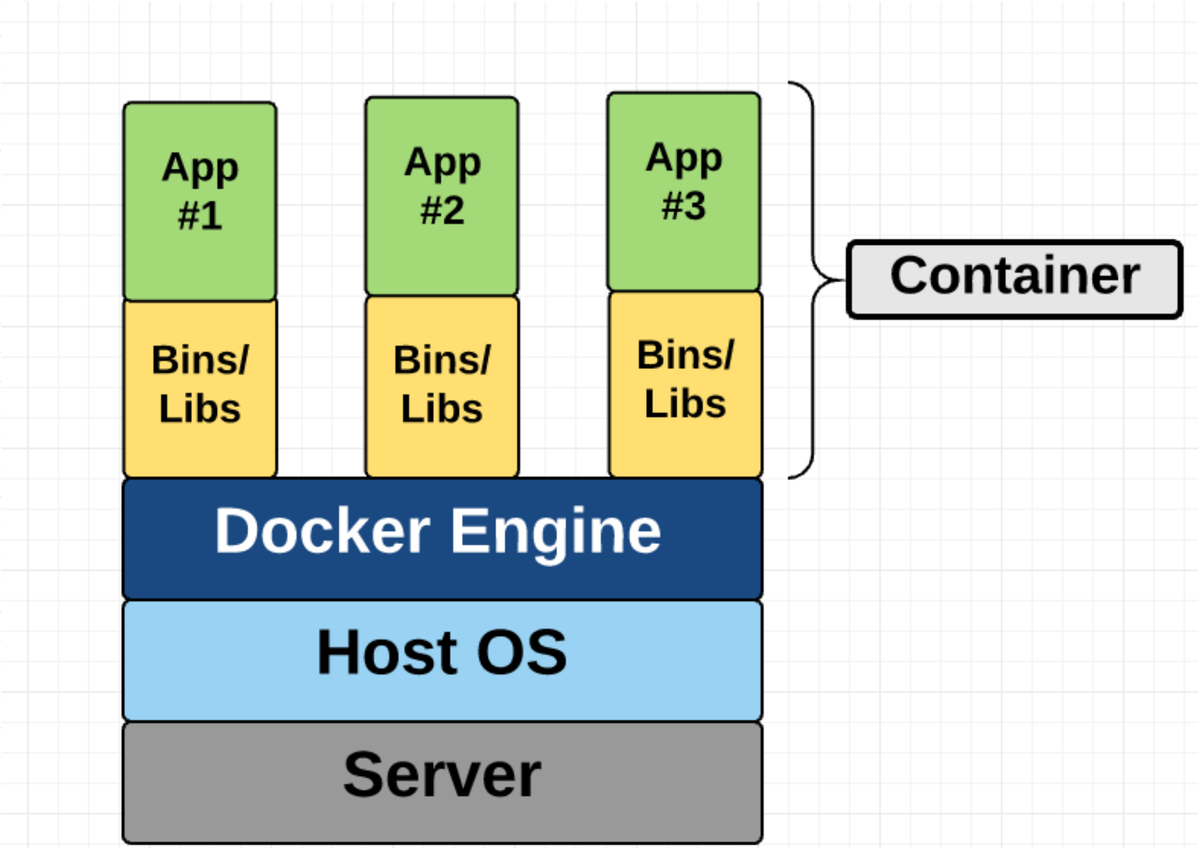


Docker

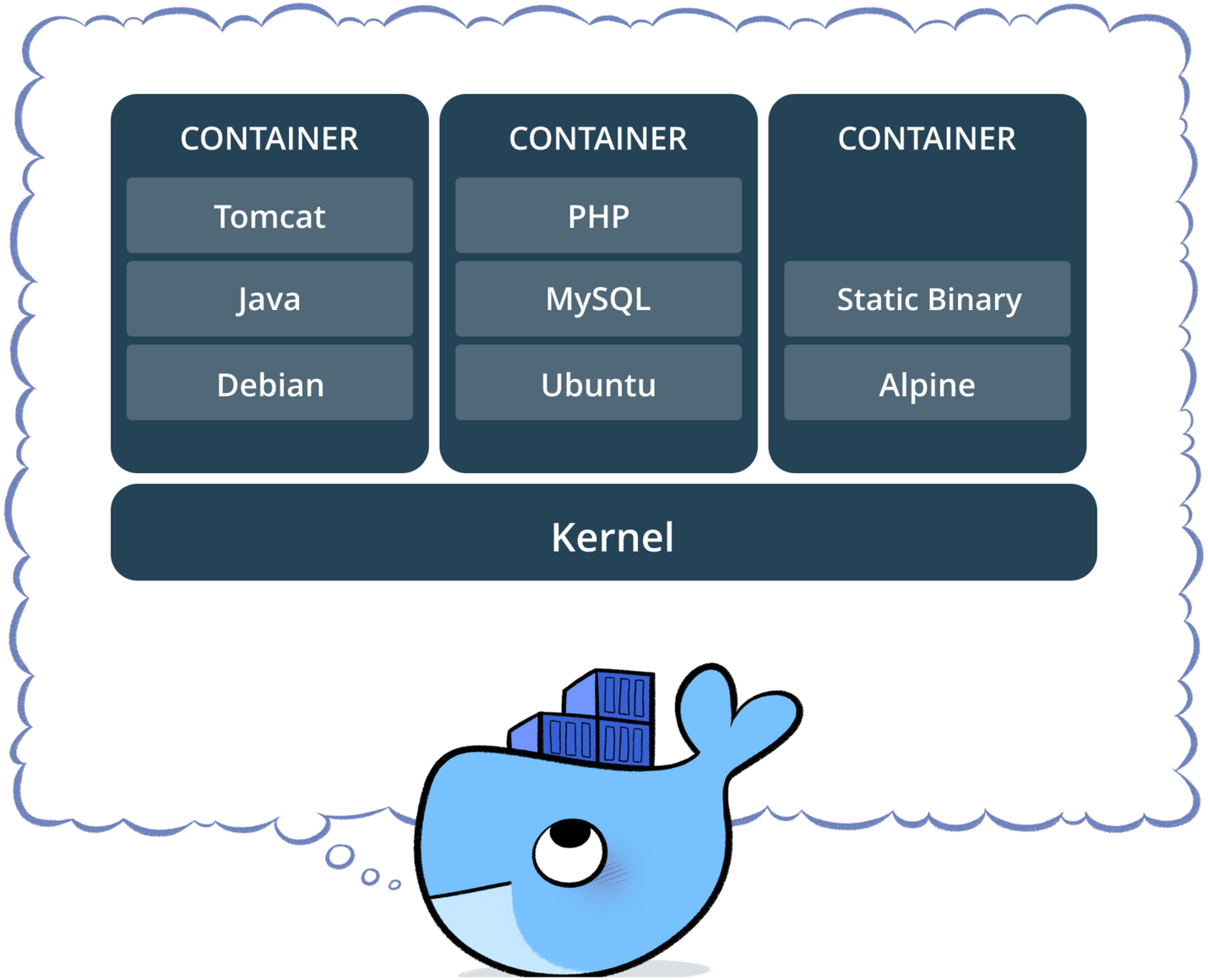
- Before proceeding, we have to understand Docker better.
  - Docker vs VM
  - Dockerfile and layer-based images
  - DockerHub and the ecosystem
  - Kubernetes and other orchestrators



No OS -> Light weight



https://medium.freecodecamp.org/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b



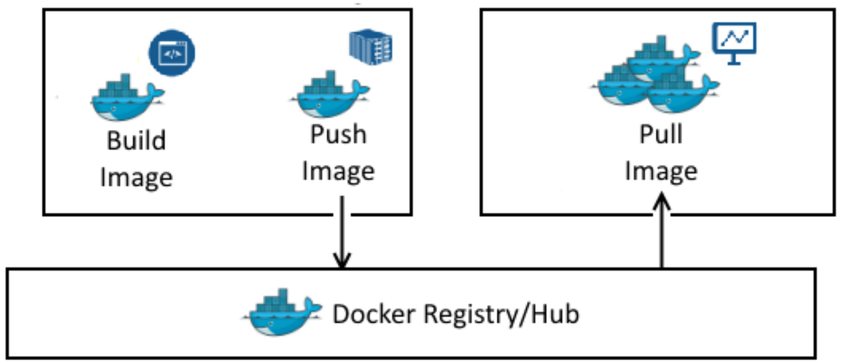
Light weight -> Heavy use

- Spin up a container for every discrete task
- For example, a web app might have four containers:
  - Web server
  - Database
  - Job queue
  - Worker

https://www.docker.com/what-container

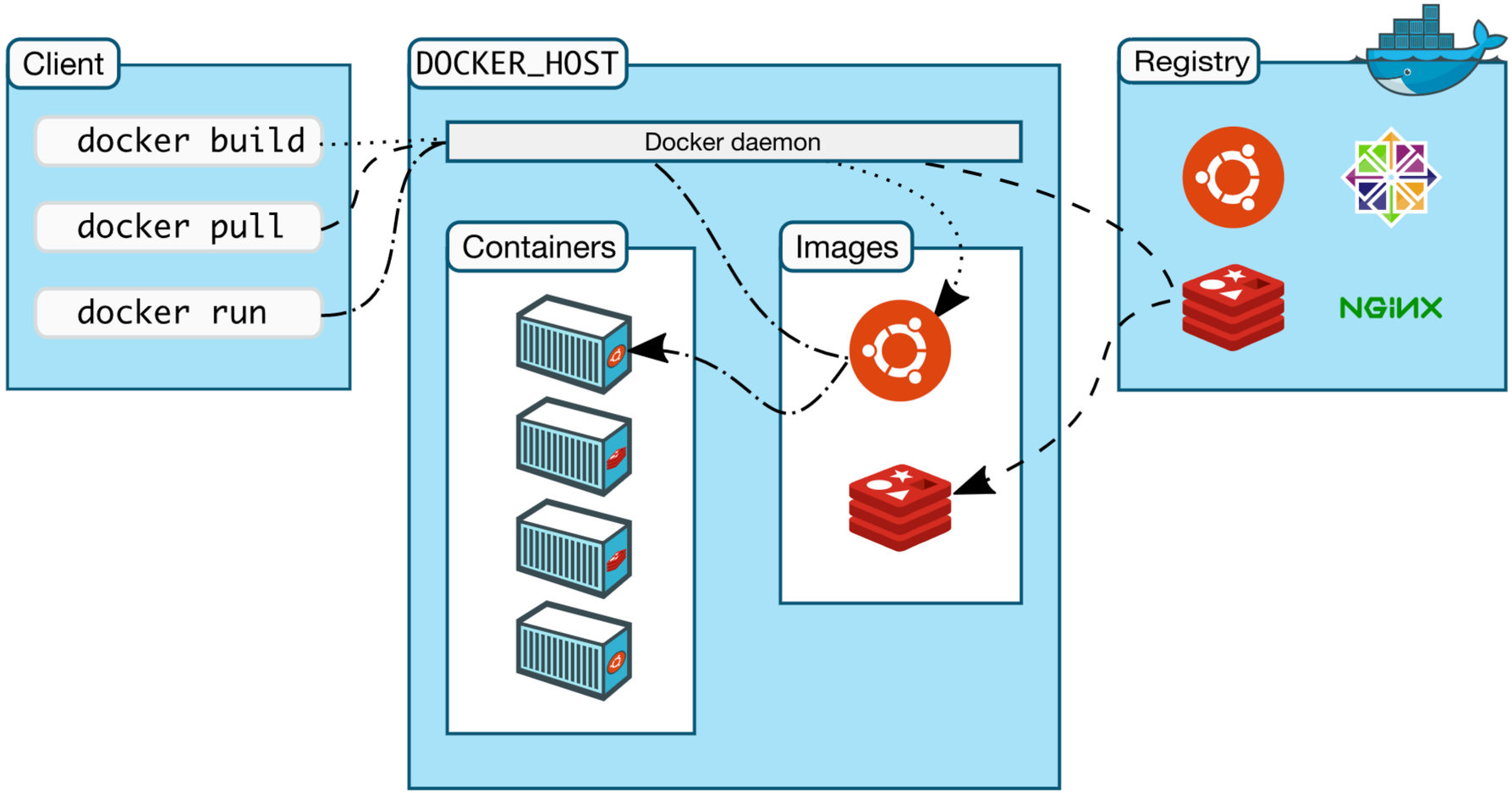
Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this contain
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```



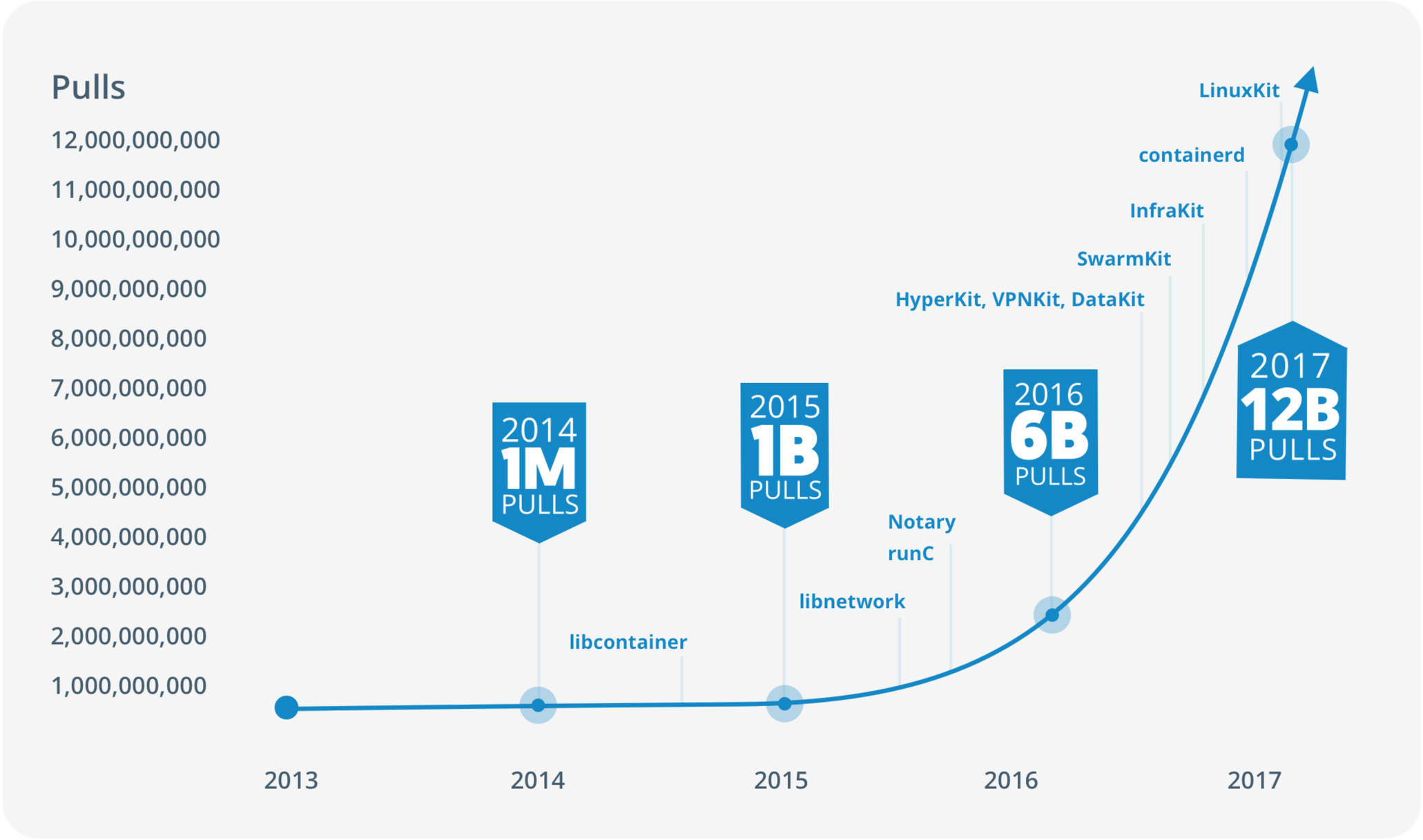

```
$ docker run -p 4000:80 gordon/get-started:part2
Unable to find image 'gordon/get-started:part2' locally
part2: Pulling from gordon/get-started
10a267c67f42: Already exists
f68a39a6a5e4: Already exists
9beaffc0cf19: Already exists
3c1fe835fb6b: Already exists
4c9f1fa8fcb8: Already exists
ee7d8f576a14: Already exists
fbccdcced46e: Already exists
Digest: sha256:0601c866aab2adcc6498200efd0f754037e909e5fd42069adeff72d1e2439068
Status: Downloaded newer image for gordon/get-started:part2
 * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
```

Strong Ecosystem



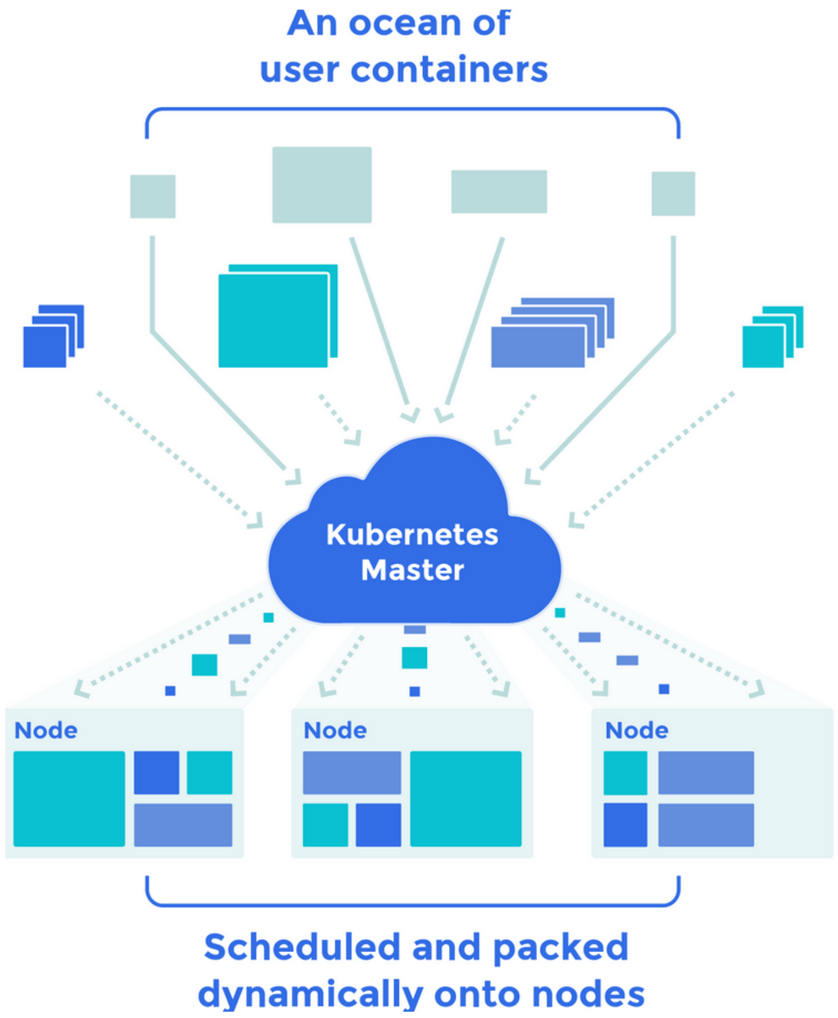
- Images are easy to find, modify, and contribute back to DockerHub
- Private images easy to store in same place

https://docs.docker.com/engine/docker-overview



https://www.docker.com/what-container#/package_software

Container Orchestration



- Distributing containers onto underlying VMs or bare metal
- Kubernetes is the open-source winner, but cloud providers have good offerings, too.

kubernetes
by Google

Questions?

Amazon SageMaker · Determined AI · Neptune Machine Learning Lab · H2O · FLOYD · DOMINO DATA LAB

"All-in-one"

Data

Versioning: Quilt, DVC, Pachyderm, LIQUIDATA

Labeling: figure eight, scale, prodigy

Processing: PY, Luigi

Database

Storage: NFS, S3

or

Development

Distributed Training: PYTORCH, RAY, HOROVOD

Frameworks: PYTORCH, K, fast.ai

Software Engineering: jupyter, git

Training/Evaluation

Hyperparameter Tuning: SIGOPT, tune

Experiment Management: Weights & Biases, comet, TensorBoard, LOSSWISE

Resource Management: Polyaxon, PY, slurm workload manager

or

Deployment

Monitoring: ?

Hardware / Mobile: ML, NVIDIA TensorRT, TensorFlow Lite

Web: ALGORITHMIA, SELDON

Interchange: ONNX

CI / Testing: Buildkite, ?

or

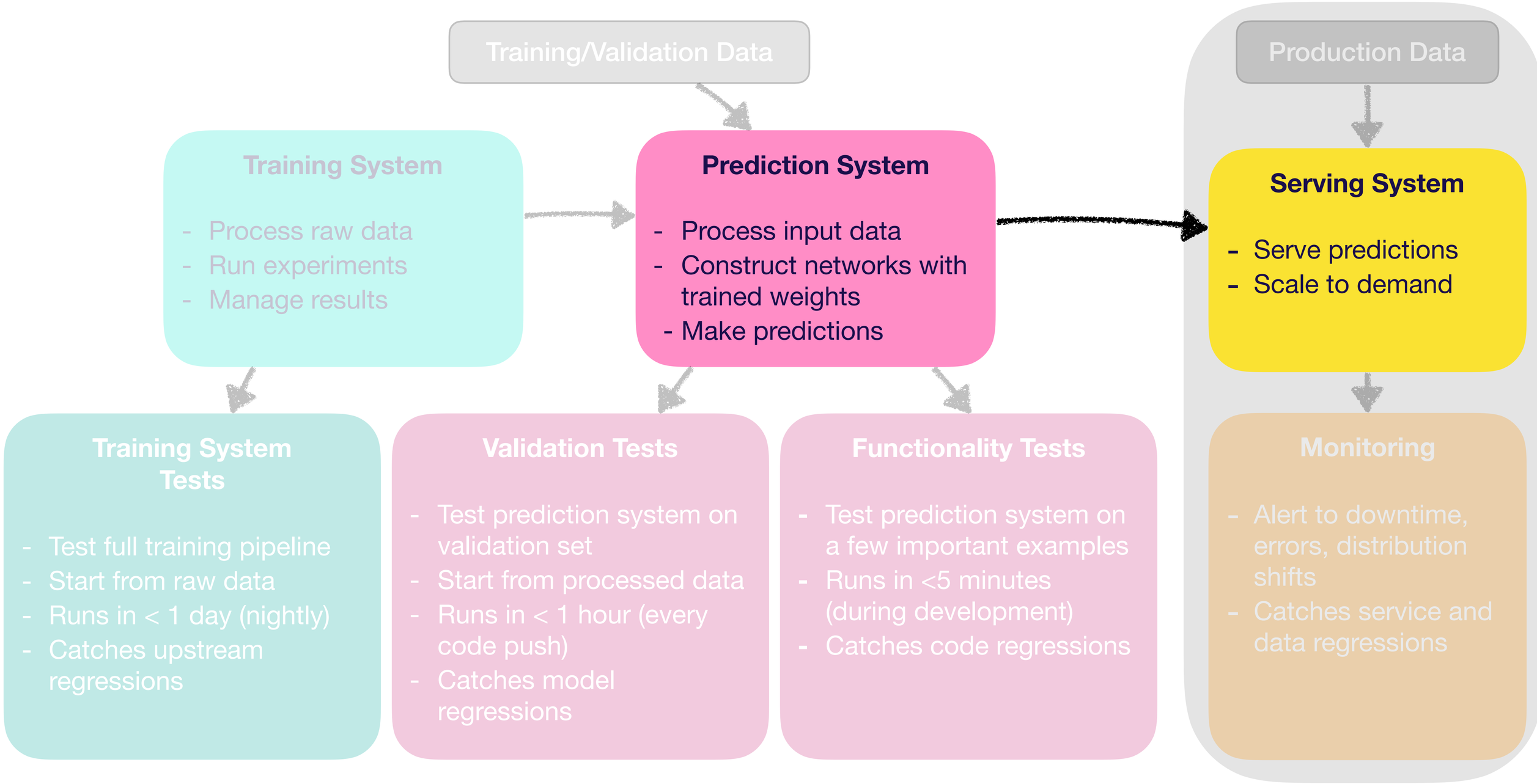
or

Terminology

- **Inference**: making predictions (not training)
- **Model Serving**: ML-specific deployment, usually focused on optimizing GPU usage

Web Deployment

- **REST API**
  - Serving predictions in response to canonically-formatted HTTP requests
  - The web server is running and calling the prediction system
- **Options**
  - Deploying code to VMs, scale by adding instances
  - Deploy code as containers, scale via orchestration
  - Deploy code as a “serverless function”
  - Deploy via a model serving solution


Training/Validation Data
Production Data
Training System
- Process raw data
- Run experiments
- Manage results
Prediction System
- Process input data
- Construct networks with trained weights
- Make predictions
Serving System
- Serve predictions
- Scale to demand
Training System Tests
- Test full training pipeline
- Start from raw data
- Runs in < 1 day (nightly)
- Catches upstream regressions
Validation Tests
- Test prediction system on validation set
- Start from processed data
- Runs in < 1 hour (every code push)
- Catches model regressions
Functionality Tests
- Test prediction system on a few important examples
- Runs in <5 minutes (during development)
- Catches code regressions
Monitoring
- Alert to downtime, errors, distribution shifts
- Catches service and data regressions

1	Feature expectations are captured in a schema.
2	All features are beneficial.
3	No feature's cost is too much.
4	Features adhere to meta-level requirements.
5	The data pipeline has appropriate privacy controls.
6	New features can be added quickly.
7	All input feature code is tested.

Data Tests

1	Model specs are reviewed and submitted.
2	Offline and online metrics correlate.
3	All hyperparameters have been tuned.
4	The impact of model staleness is known.
5	A simpler model is not better.
6	Model quality is sufficient on important data slices.
7	The model is tested for considerations of inclusion.

Model Tests

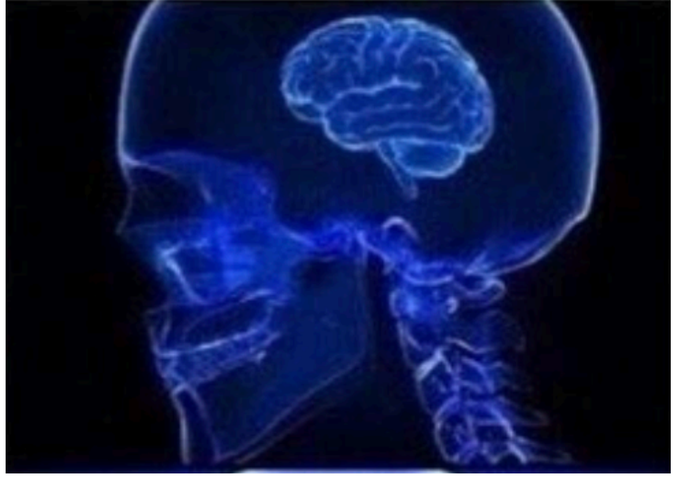
1	Training is reproducible.
2	Model specs are unit tested.
3	The ML pipeline is Integration tested.
4	Model quality is validated before serving.
5	The model is debuggable.
6	Models are canaried before serving.
7	Serving models can be rolled back.

ML Infrastructure Tests

1	Dependency changes result in notification.
2	Data invariants hold for inputs.
3	Training and serving are not skewed.
4	Models are not too stale.
5	Models are numerically stable.
6	Computing performance has not regressed.
7	Prediction quality has not regressed.

Monitoring Tests

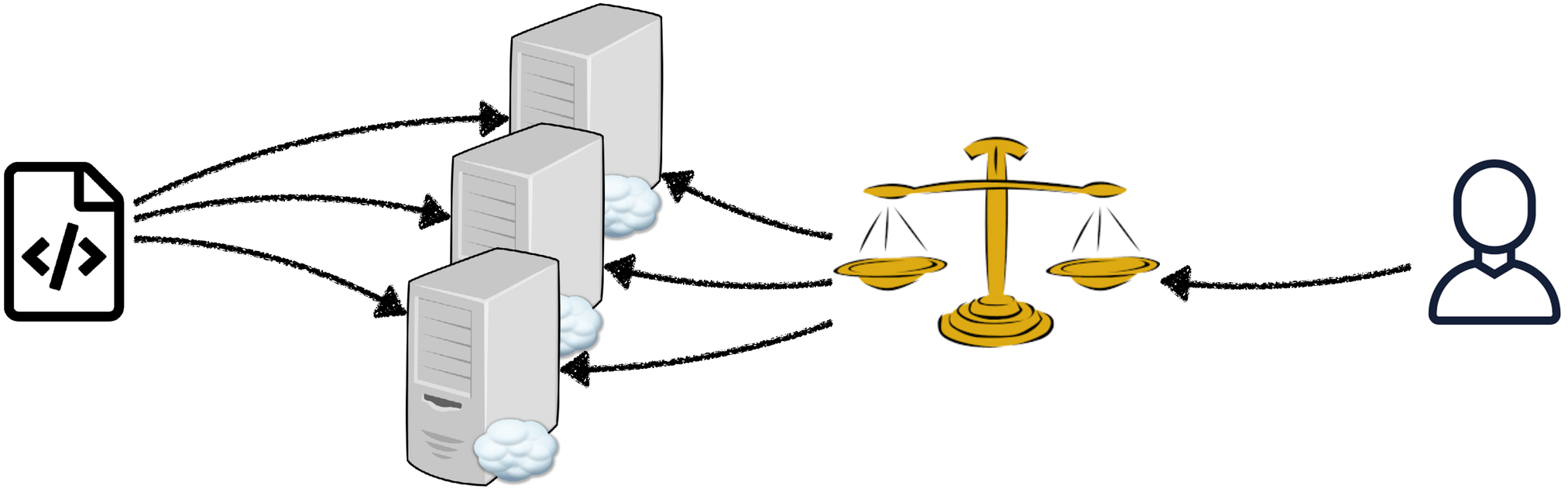
DEPLOY CODE TO YOUR OWN BARE METAL



DEPLOY CODE TO YOUR OWN BARE METAL

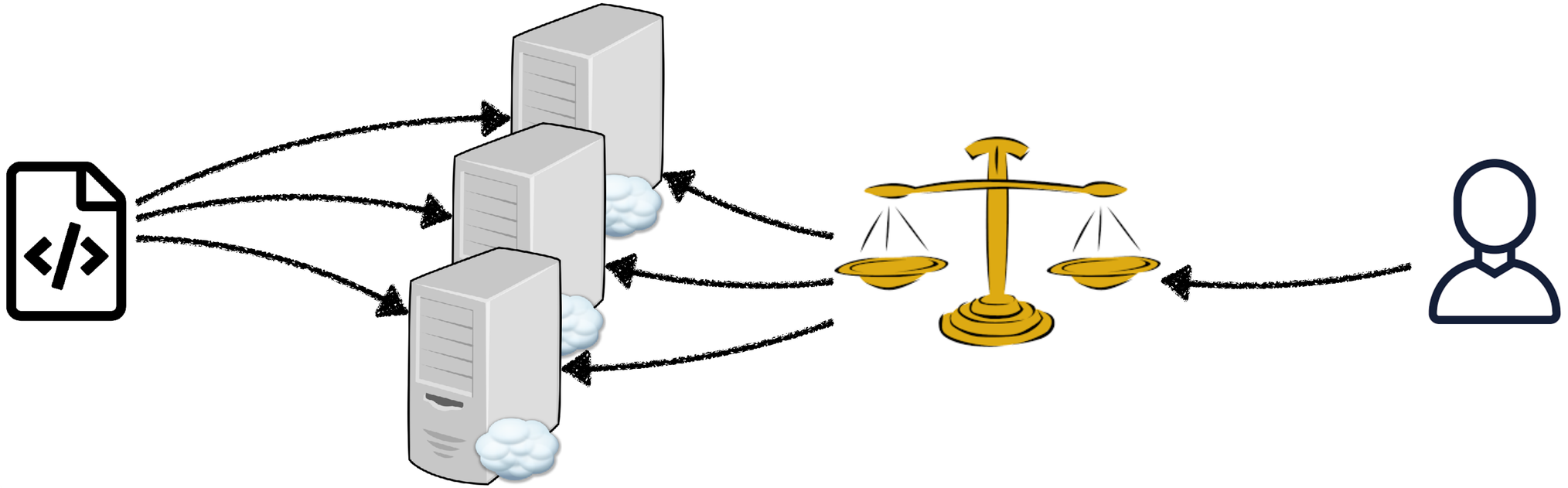
DEPLOY CODE TO CLOUD INSTANCES

Deploying code to cloud instances



- Each instance has to be provisioned with dependencies and app code
- Number of instances is managed manually or via auto-scaling
- Load balancer sends user traffic to the instances

Deploying code to cloud instances



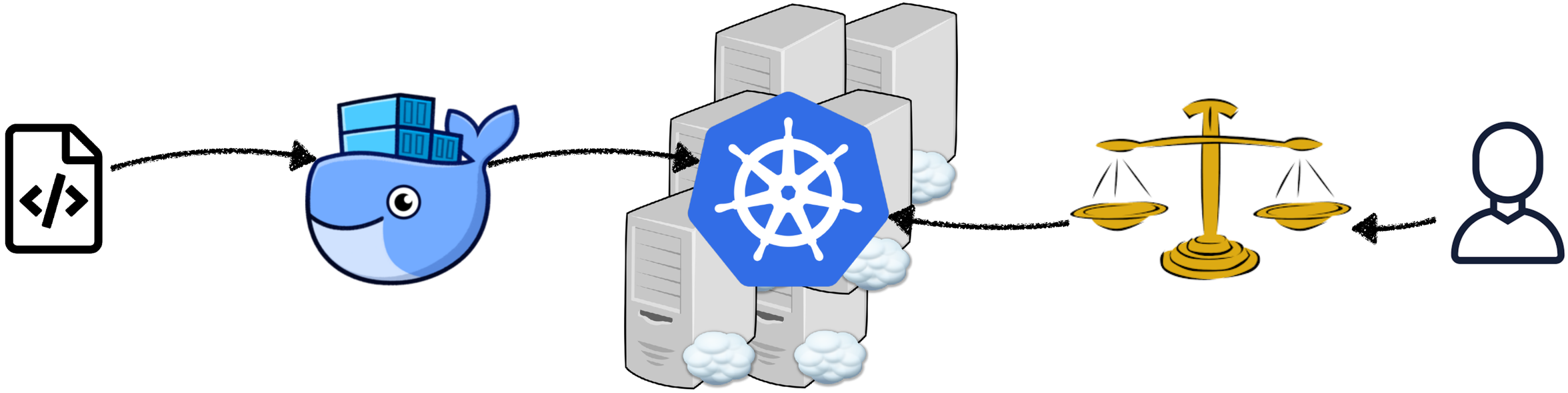
- **Cons:**
  - Provisioning can be brittle
  - Paying for instances even when not using them (auto-scaling does help)

DEPLOY CODE TO YOUR OWN BARE METAL

DEPLOY CODE TO CLOUD INSTANCES

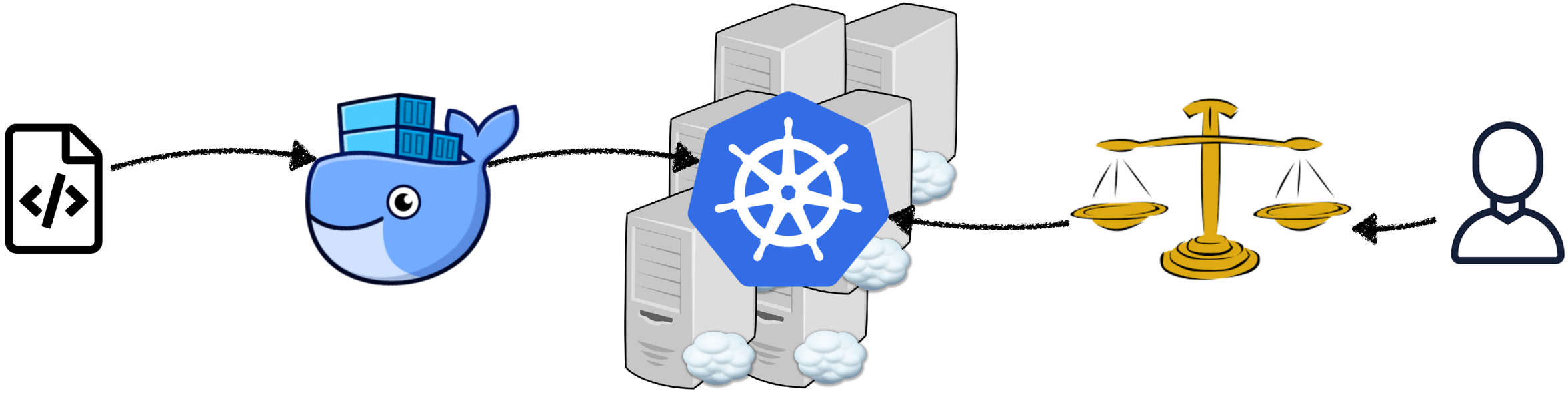
DEPLOY DOCKER CONTAINERS TO CLOUD INSTANCES

Deploying containers



- App code and dependencies are packaged into Docker containers
- Kubernetes or alternative (AWS Fargate) orchestrates containers (DB / workers / web servers / etc.

Deploying containers



- **Cons:**
  - Still managing your own servers and paying for uptime, not compute-time

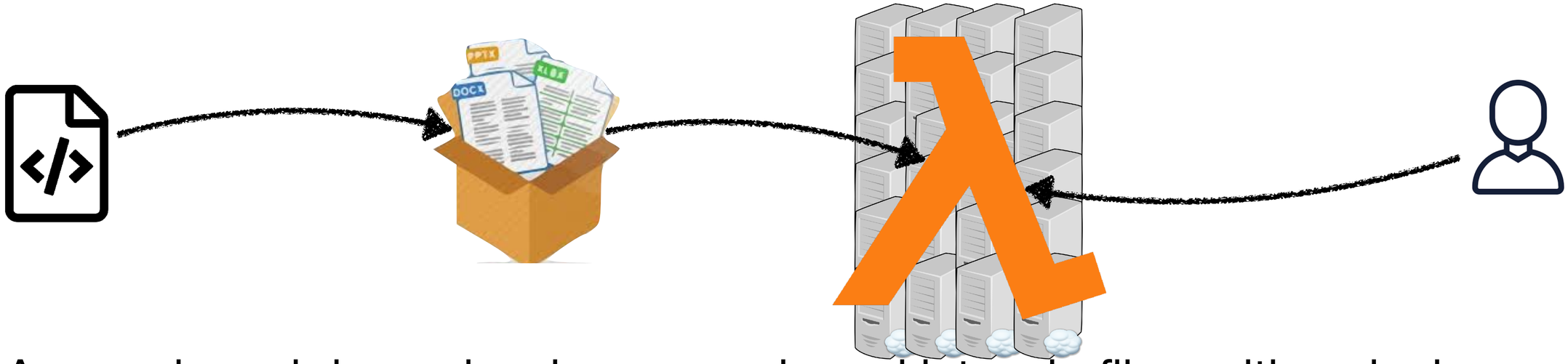
DEPLOY CODE TO YOUR OWN BARE METAL

DEPLOY CODE TO CLOUD INSTANCES

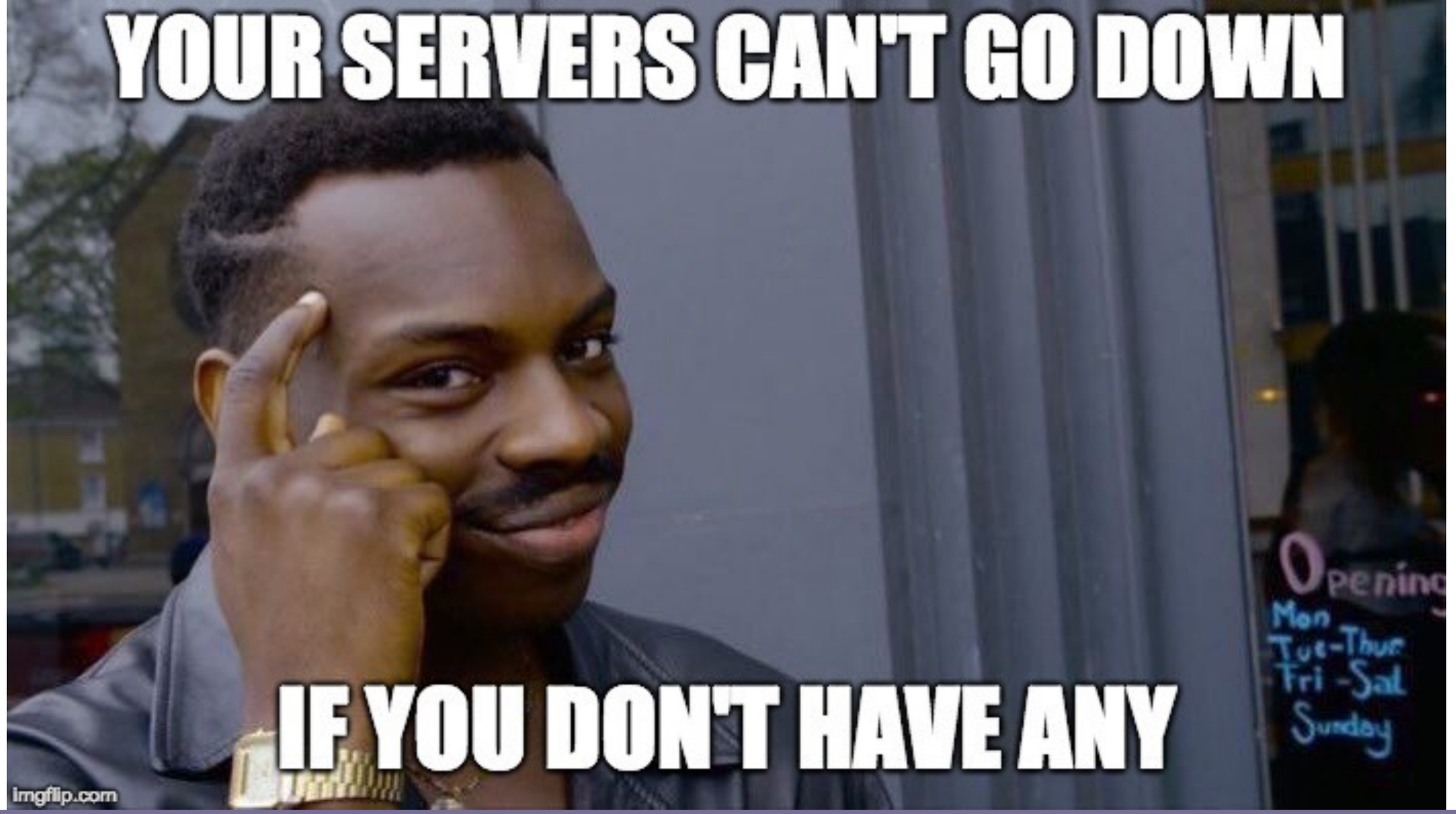
DEPLOY DOCKER CONTAINERS TO CLOUD INSTANCES

DEPLOY SERVERLESS FUNCTIONS

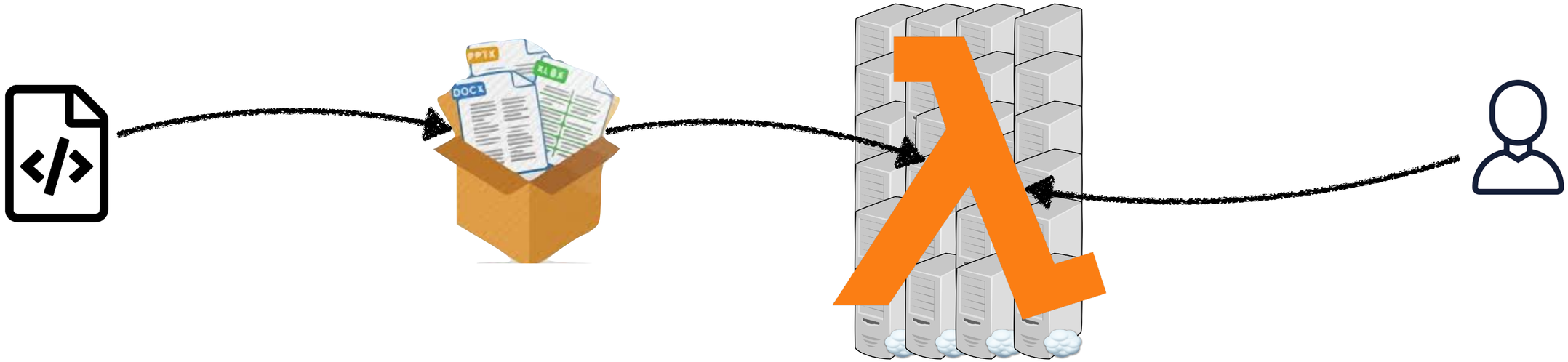
Deploying code as serverless functions



- App code and dependencies are packaged into .zip files, with a single entry point function
- AWS Lambda (or Google Cloud Functions, or Azure Functions) manages everything else: instant scaling to 10,000+ requests per second, load balancing, etc.
- Only pay for compute-time.


YOUR SERVERS CAN'T GO DOWN
IF YOU DON'T HAVE ANY
Opening
Mon
Tue-Thur
Fri -Sat
Sunday
imgflip.com

Deploying code as serverless functions



- **Cons:**
  - Entire deployment package has to fit within 500MB, <5 min execution, <3GB memory (on AWS Lambda)
  - Only CPU execution

1	Feature expectations are captured in a schema.
2	All features are beneficial.
3	No feature's cost is too much.
4	Features adhere to meta-level requirements.
5	The data pipeline has appropriate privacy controls.
6	New features can be added quickly.
7	All input feature code is tested.

Data Tests

1	Model specs are reviewed and submitted.
2	Offline and online metrics correlate.
3	All hyperparameters have been tuned.
4	The impact of model staleness is known.
5	A simpler model is not better.
6	Model quality is sufficient on important data slices.
7	The model is tested for considerations of inclusion.

Model Tests

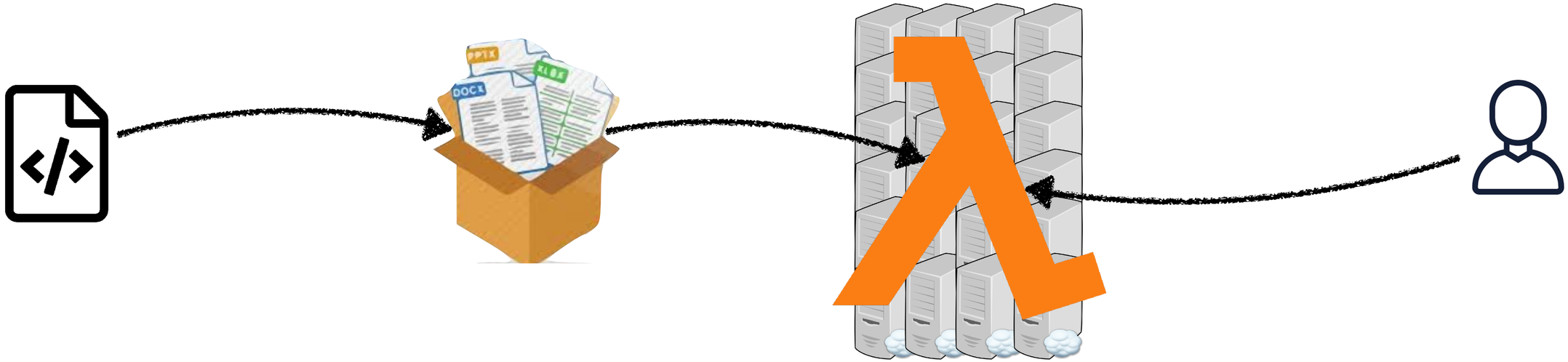
1	Training is reproducible.
2	Model specs are unit tested.
3	The ML pipeline is Integration tested.
4	Model quality is validated before serving.
5	The model is debuggable.
6	Models are canaried before serving.
7	Serving models can be rolled back.

ML Infrastructure Tests

1	Dependency changes result in notification.
2	Data invariants hold for inputs.
3	Training and serving are not skewed.
4	Models are not too stale.
5	Models are numerically stable.
6	Computing performance has not regressed.
7	Prediction quality has not regressed.

Monitoring Tests

Deploying code as serverless functions



- **Canarying**
  - Easy to have two versions of Lambda functions in production, and start sending low volume traffic to one.
- **Rollback**
  - Easy to switch back to the previous version of the function.

Model Serving

- Web deployment options specialized for machine learning models
  - Tensorflow Serving (Google)
  - Model Server for MXNet (Amazon)
  - Clipper (Berkeley RISE Lab)
  - SaaS solutions like Algorithmia
- The most important feature is batching requests for GPU inference

Tensorflow Serving

- Able to serve Tensorflow predictions at high throughput.
- Used by Google and their “all-in-one” machine learning offerings
- Overkill unless you know you need to use GPU for inference

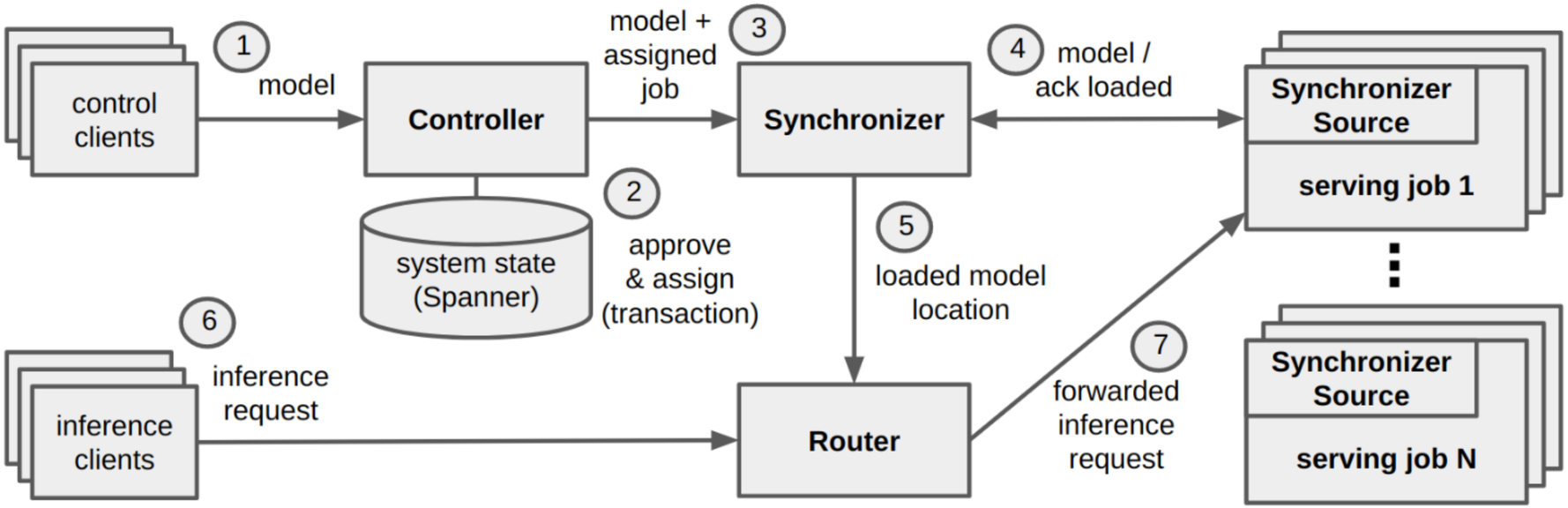
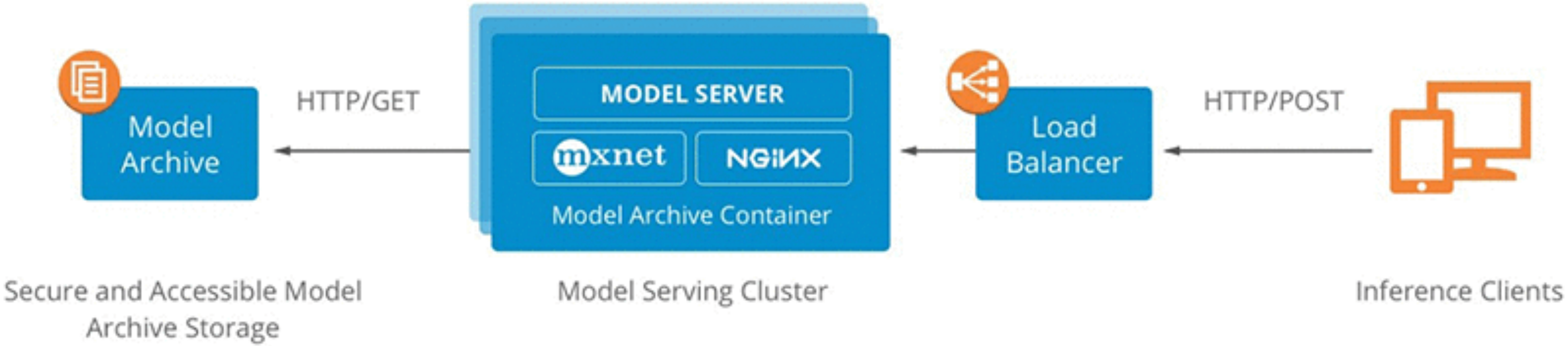


Figure 2: TFS^2 (hosted model serving) architecture.

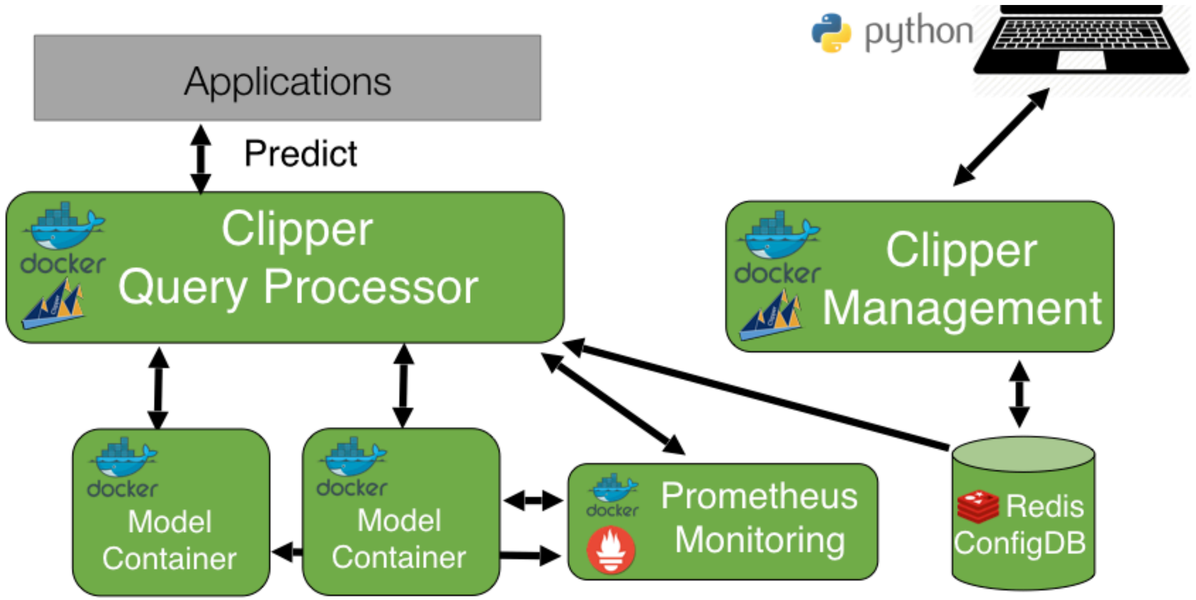
MXNet Model Server



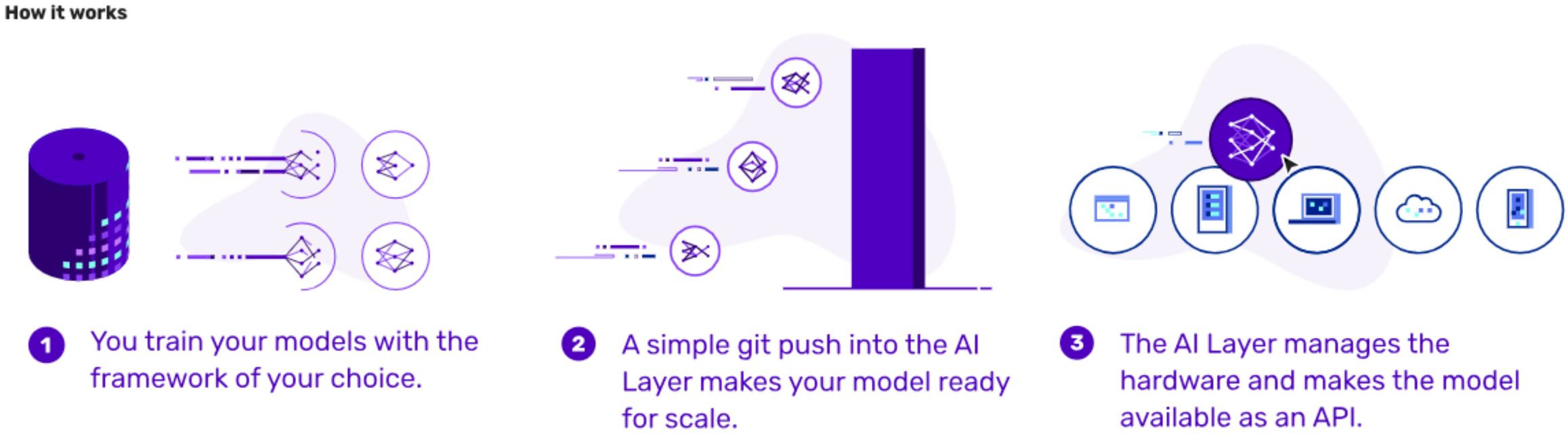
- Amazon's answer to Tensorflow Serving
- Part of their SageMaker "all-in-one" machine learning offering.

Clipper

- Open-source model serving via REST, using Docker.
- Nice-to-haves on top of the basics
  - e.g “state-of-the-art bandit and ensemble methods to intelligently select and combine predictions”
- Probably too early to be actually useful (unless you know why you need it)



Algorithmia

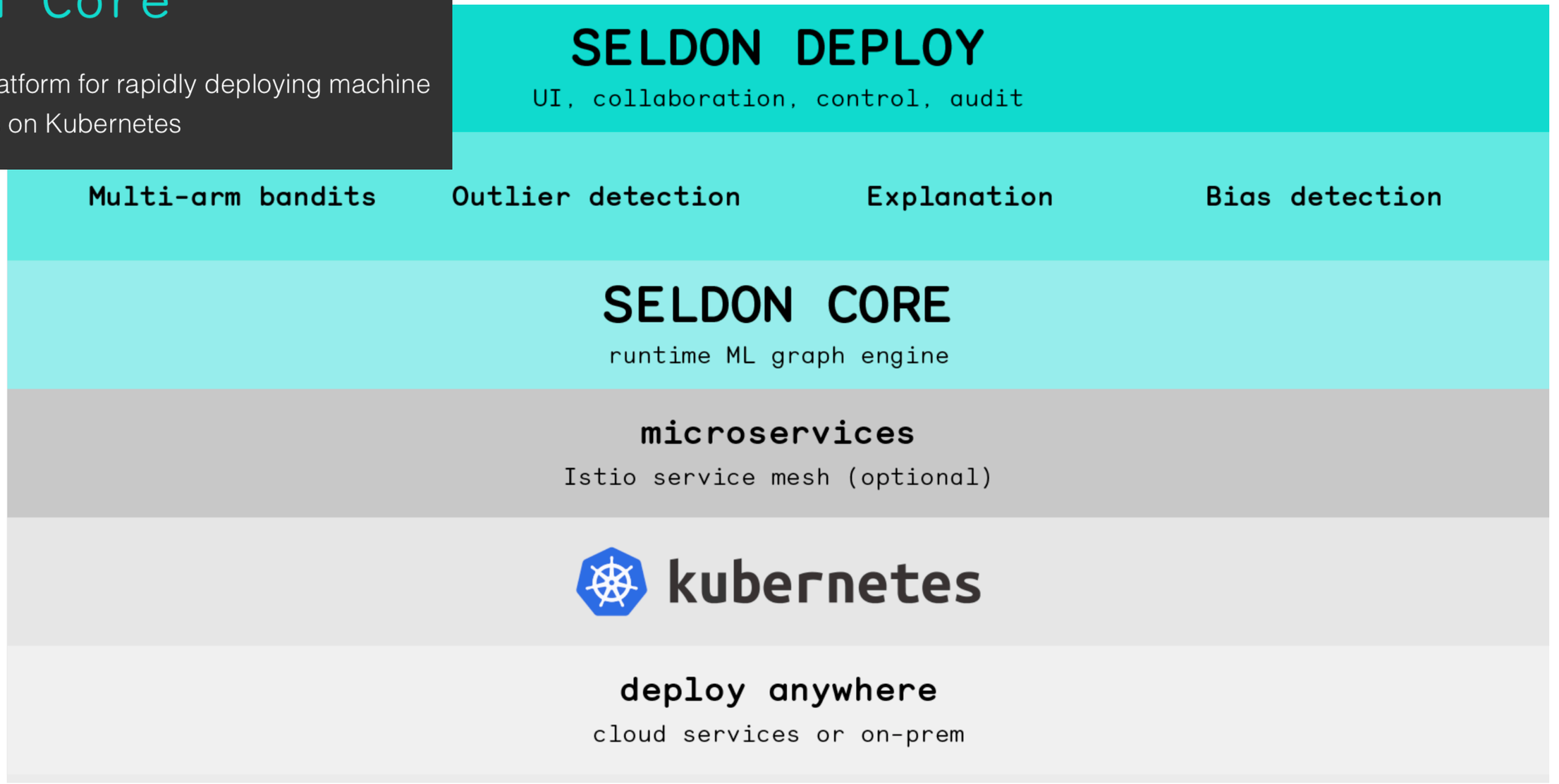


- Some startups promise effortless model serving

Seldon

Seldon Core

Open-source platform for rapidly deploying machine learning models on Kubernetes



https://www.seldon.io/tech/products/core/

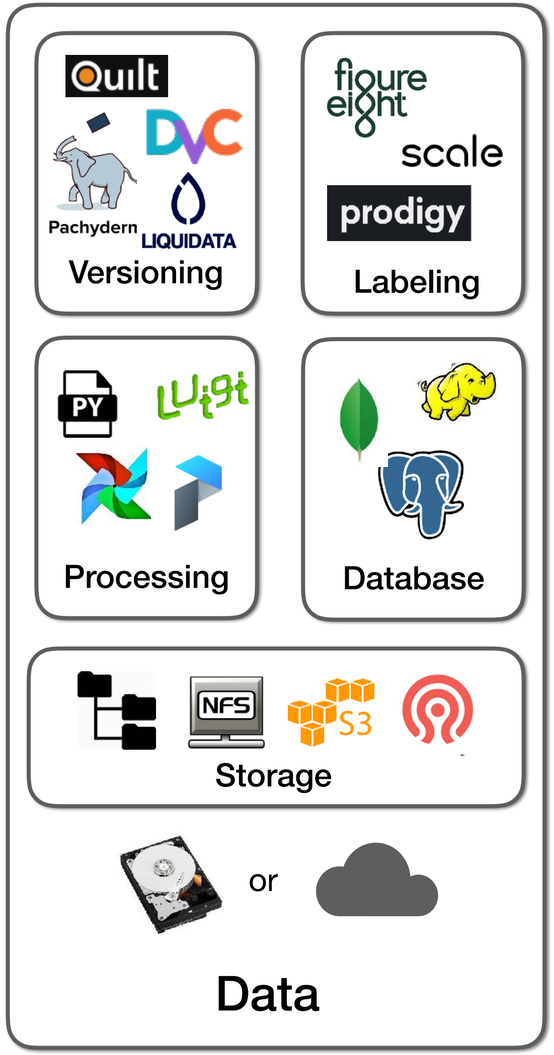
Takeaway

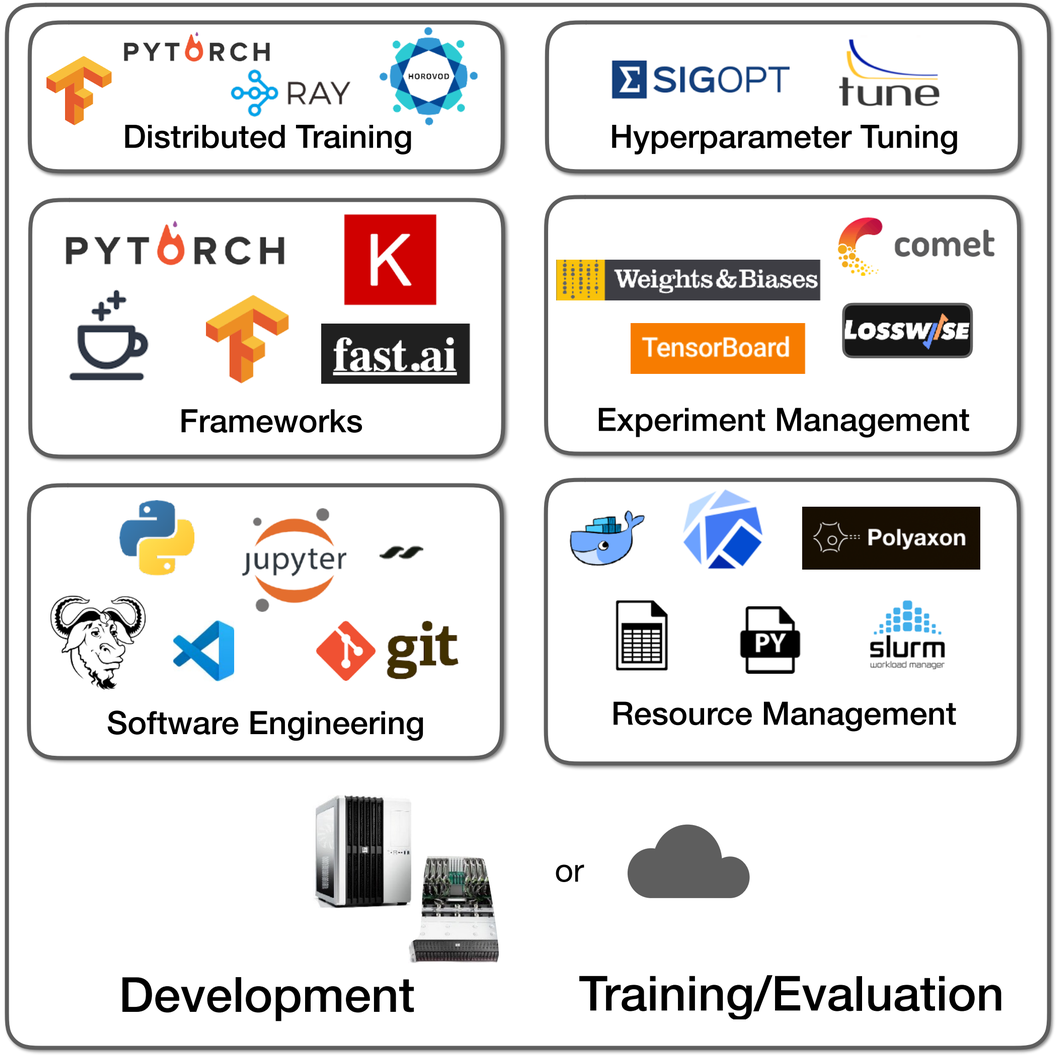
- If you are doing CPU inference, can get away with scaling by launching more servers, or going serverless.
  - The dream is deploying Docker as easily as Lambda
  - Next best thing is either Lambda or Docker, depending on your model's needs.
- If using GPU inference, things like TF Serving and Clipper become useful by adaptive batching, etc.

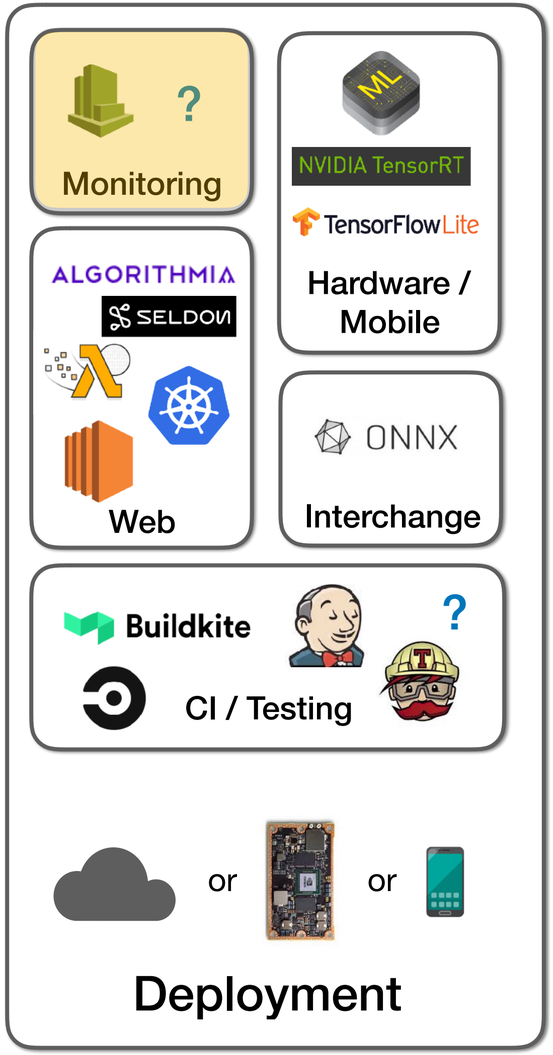
Questions?

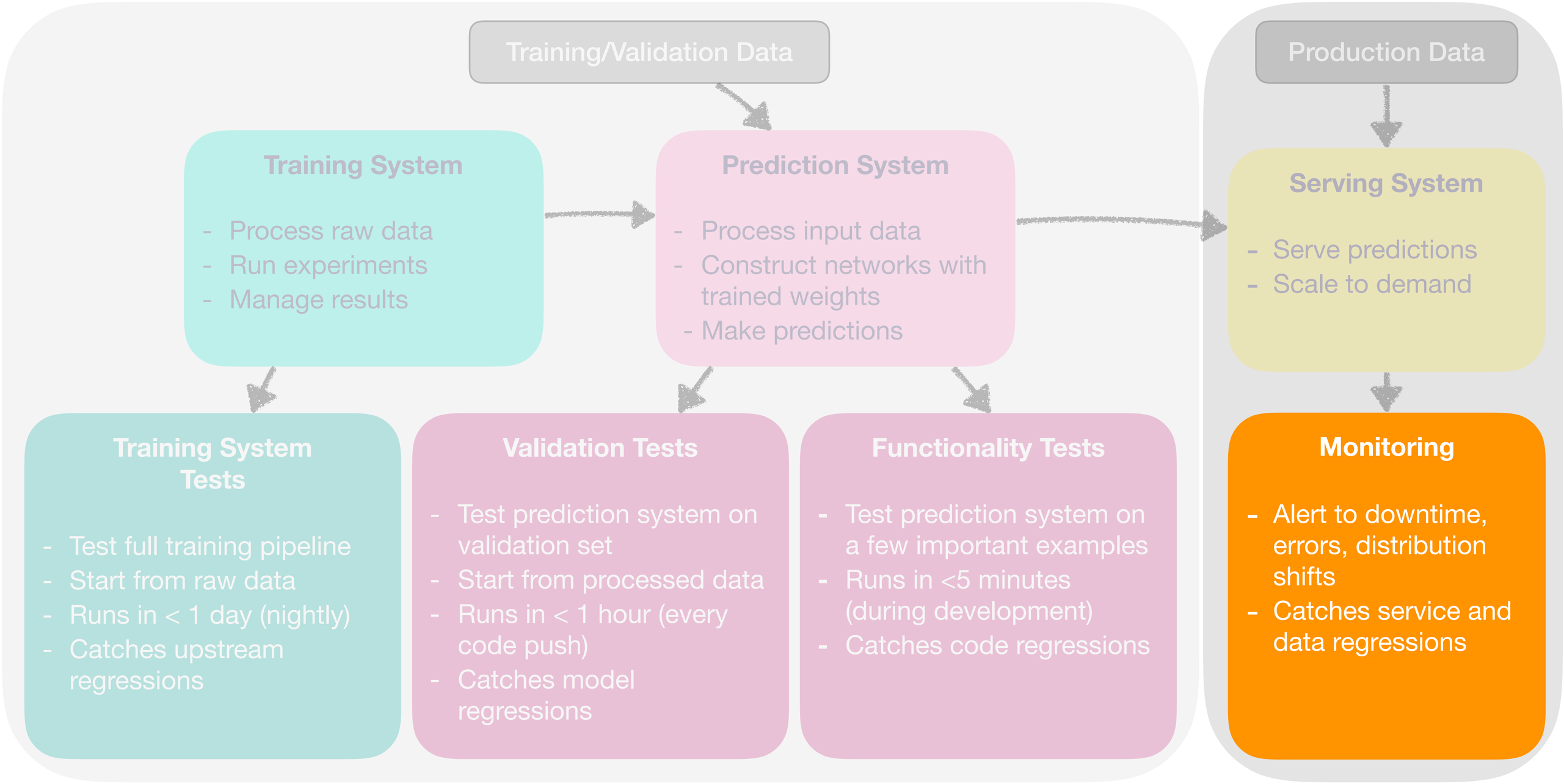

Amazon SageMaker
Determined AI
Neptune
Machine Learning Lab
H2O
FLOYD
DOMINO
DATA LAB

"All-in-one"


Quilt
DVC
Pachydern
LIQUIDATA
Versioning
figure eight
scale
prodigy
Labeling
PY
Processing
Database
NFS
S3
Storage
or
Data


PYTORCH
RAY
HOROVOD
Distributed Training
SIGOPT
tune
Hyperparameter Tuning
PYTORCH
K
fast.ai
Frameworks
Weights & Biases
comet
TensorBoard
LOSSWISE
Experiment Management
jupyter
git
Software Engineering
Polyaxon
PY
slurm
workload manager
Resource Management
Development
or
Training/Evaluation


Monitoring
ML
NVIDIA TensorRT
TensorFlowLite
Hardware / Mobile
ALGORITHMIA
SELDON
Web
ONNX
Interchange
Buildkite
CI / Testing
or
or
Deployment


Training/Validation Data
Production Data
Training System
- Process raw data
- Run experiments
- Manage results
Prediction System
- Process input data
- Construct networks with trained weights
- Make predictions
Serving System
- Serve predictions
- Scale to demand
Training System Tests
- Test full training pipeline
- Start from raw data
- Runs in < 1 day (nightly)
- Catches upstream regressions
Validation Tests
- Test prediction system on validation set
- Start from processed data
- Runs in < 1 hour (every code push)
- Catches model regressions
Functionality Tests
- Test prediction system on a few important examples
- Runs in <5 minutes (during development)
- Catches code regressions
Monitoring
- Alert to downtime, errors, distribution shifts
- Catches service and data regressions

1	Feature expectations are captured in a schema.
2	All features are beneficial.
3	No feature's cost is too much.
4	Features adhere to meta-level requirements.
5	The data pipeline has appropriate privacy controls.
6	New features can be added quickly.
7	All input feature code is tested.

Data Tests

1	Model specs are reviewed and submitted.
2	Offline and online metrics correlate.
3	All hyperparameters have been tuned.
4	The impact of model staleness is known.
5	A simpler model is not better.
6	Model quality is sufficient on important data slices.
7	The model is tested for considerations of inclusion.

Model Tests

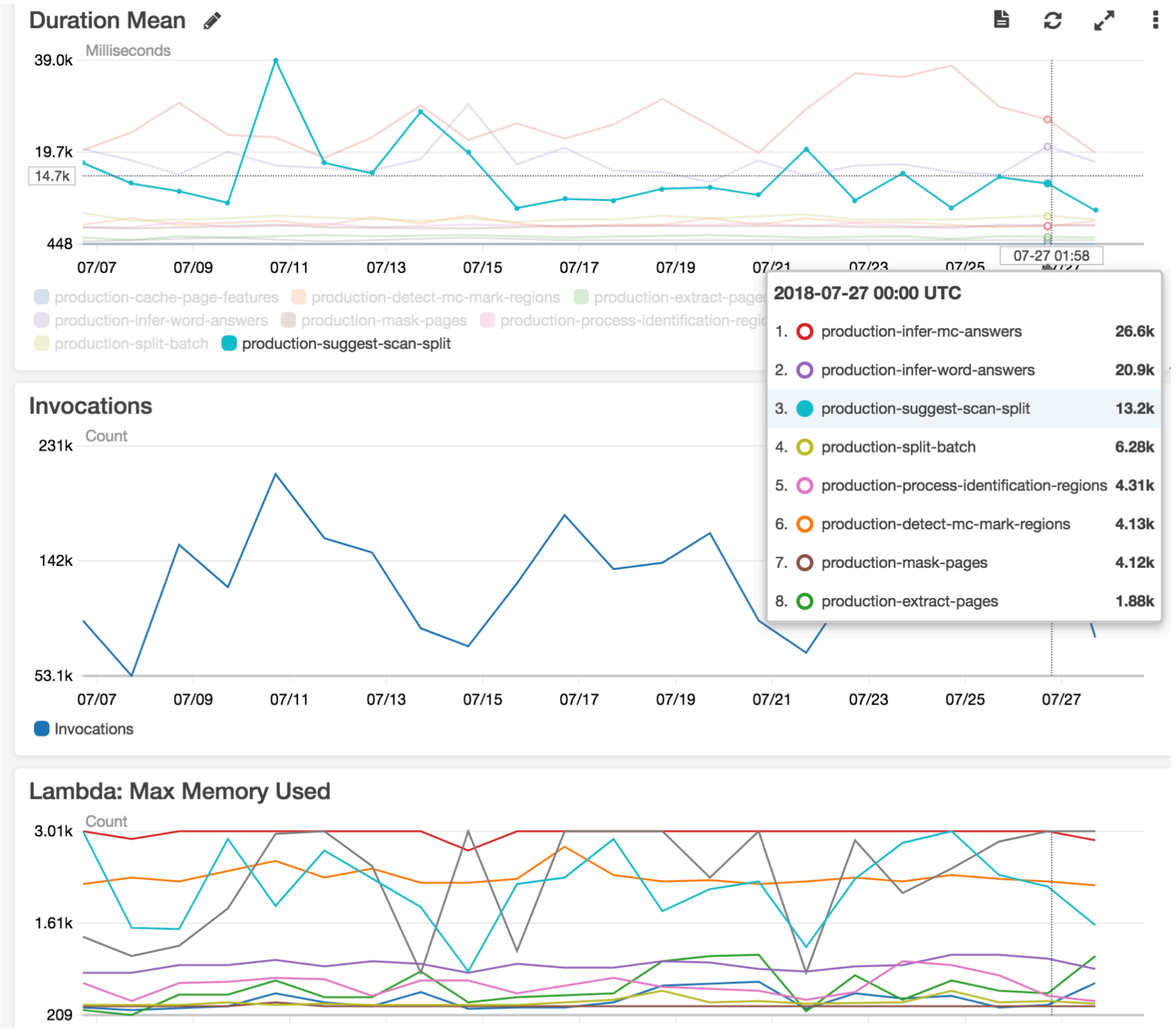
1	Training is reproducible.
2	Model specs are unit tested.
3	The ML pipeline is Integration tested.
4	Model quality is validated before serving.
5	The model is debuggable.
6	Models are canaried before serving.
7	Serving models can be rolled back.

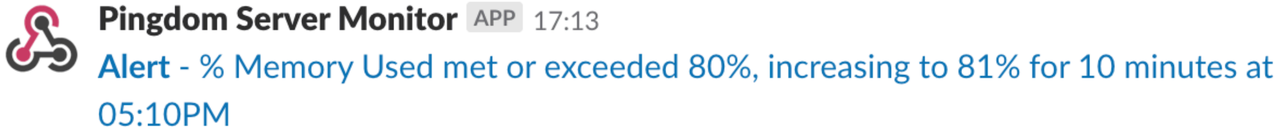
ML Infrastructure Tests

1	Dependency changes result in notification.
2	Data invariants hold for inputs.
3	Training and serving are not skewed.
4	Models are not too stale.
5	Models are numerically stable.
6	Computing performance has not regressed.
7	Prediction quality has not regressed.

Monitoring Tests

Monitoring

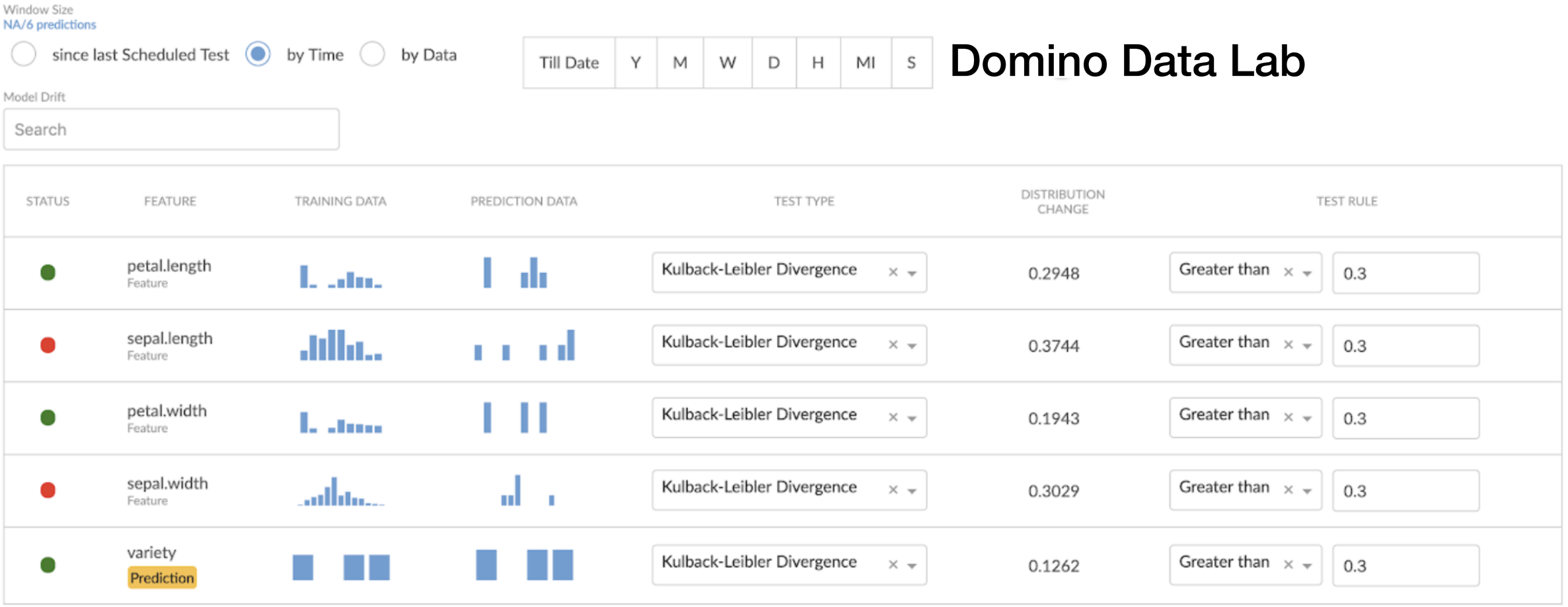




- Alarms for when things go wrong, and records for tuning things.
- Cloud providers have decent monitoring solutions.
- Anything that can be logged can be monitored (i.e. data skew)
- Will explore in lab

Data Distribution Monitoring

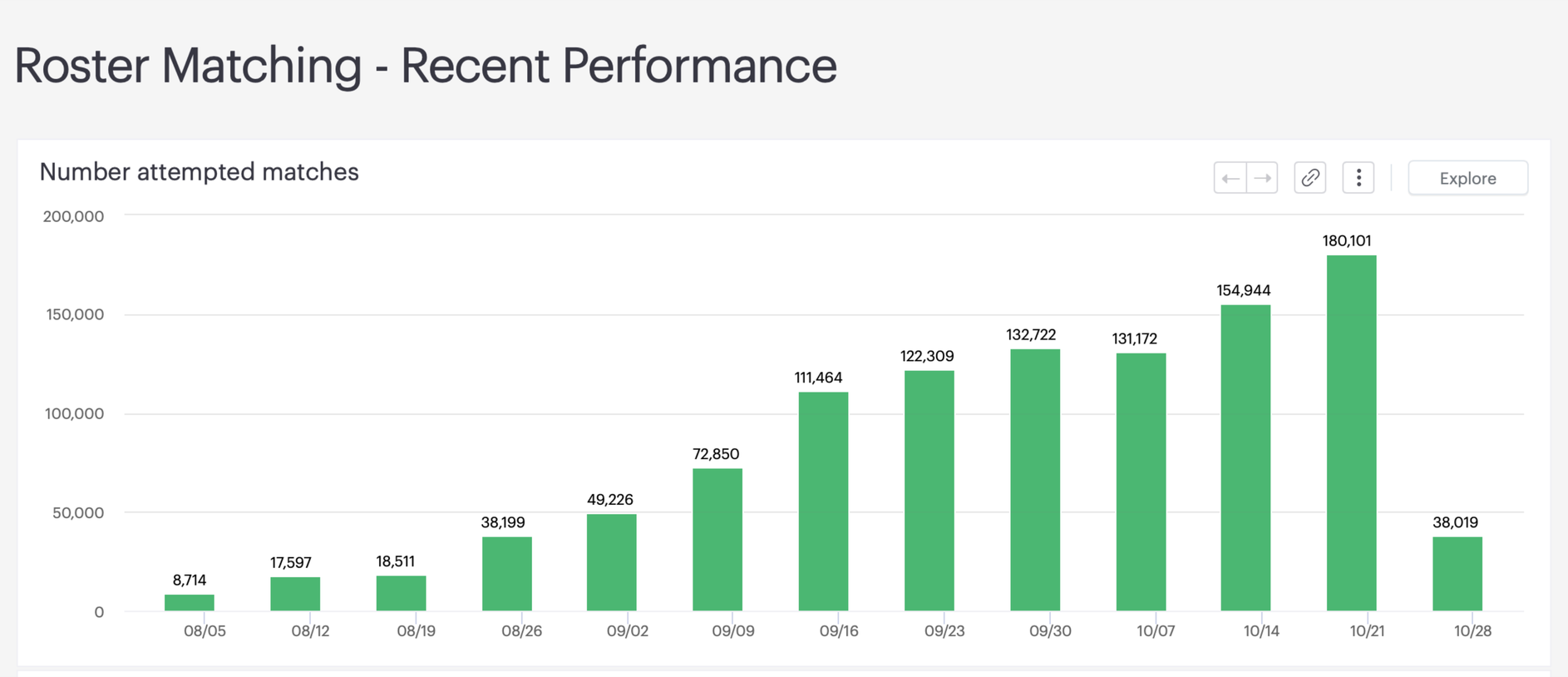
- Underserved need!



Closing the Flywheel

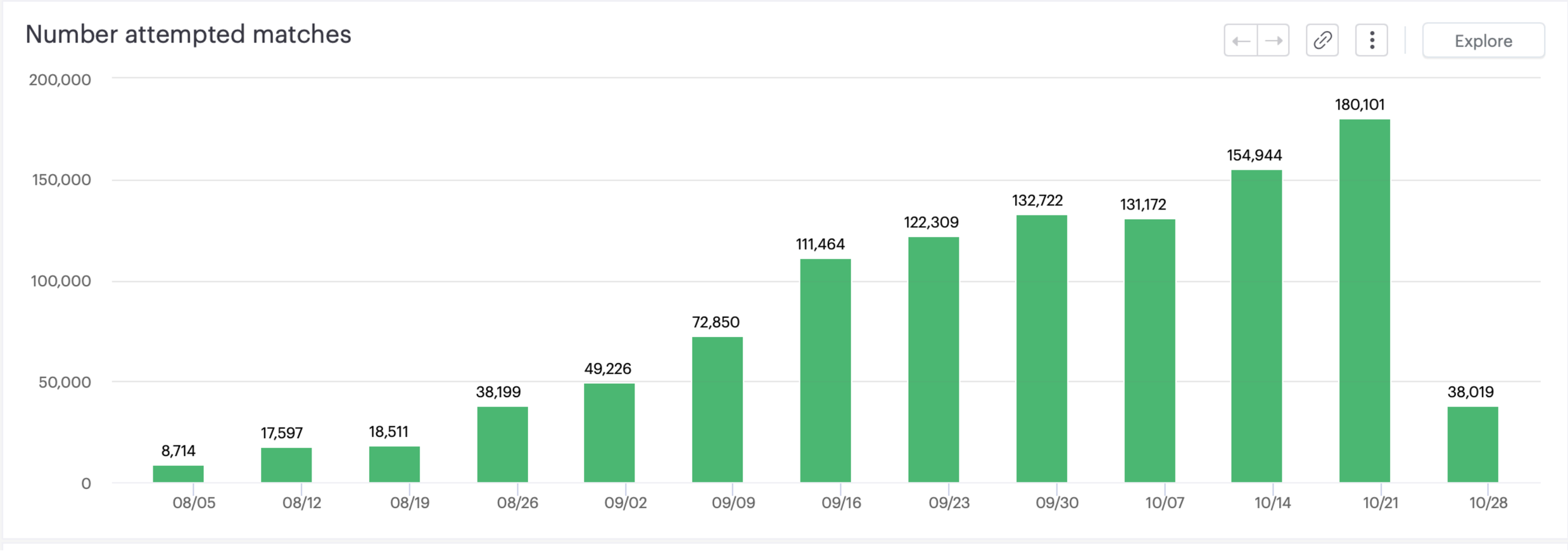
- Important to monitor the **business uses** of the model, not just its own statistics

- Important to be able to contribute failures back to dataset

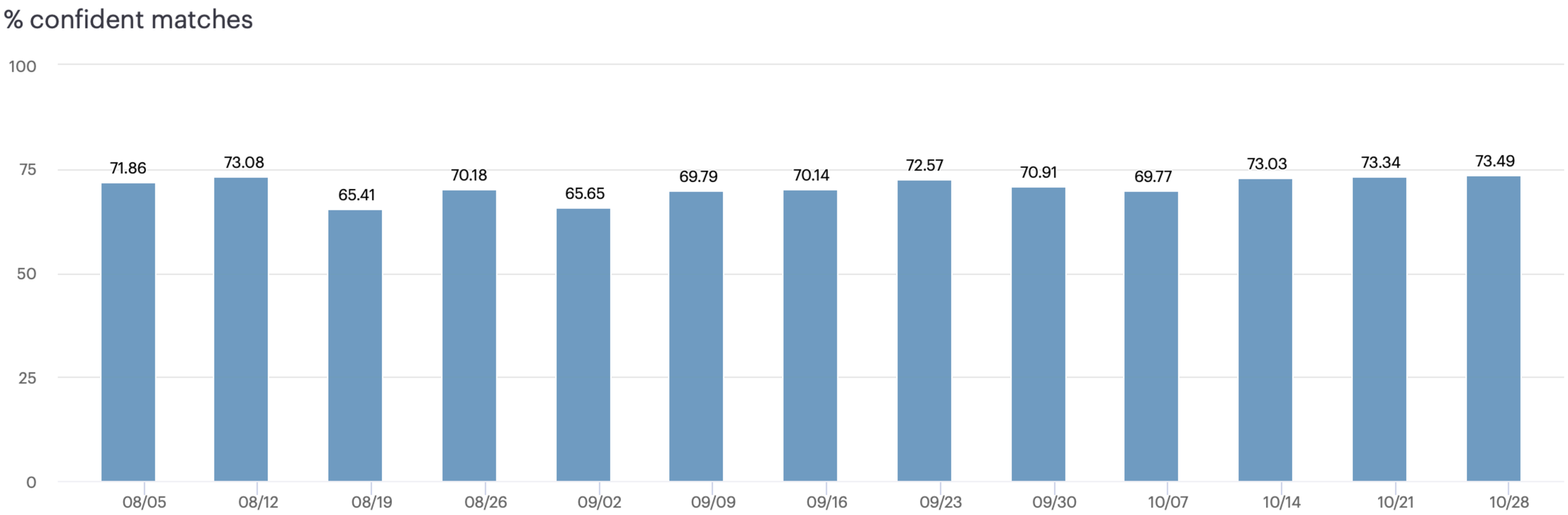


Closing the Flywheel

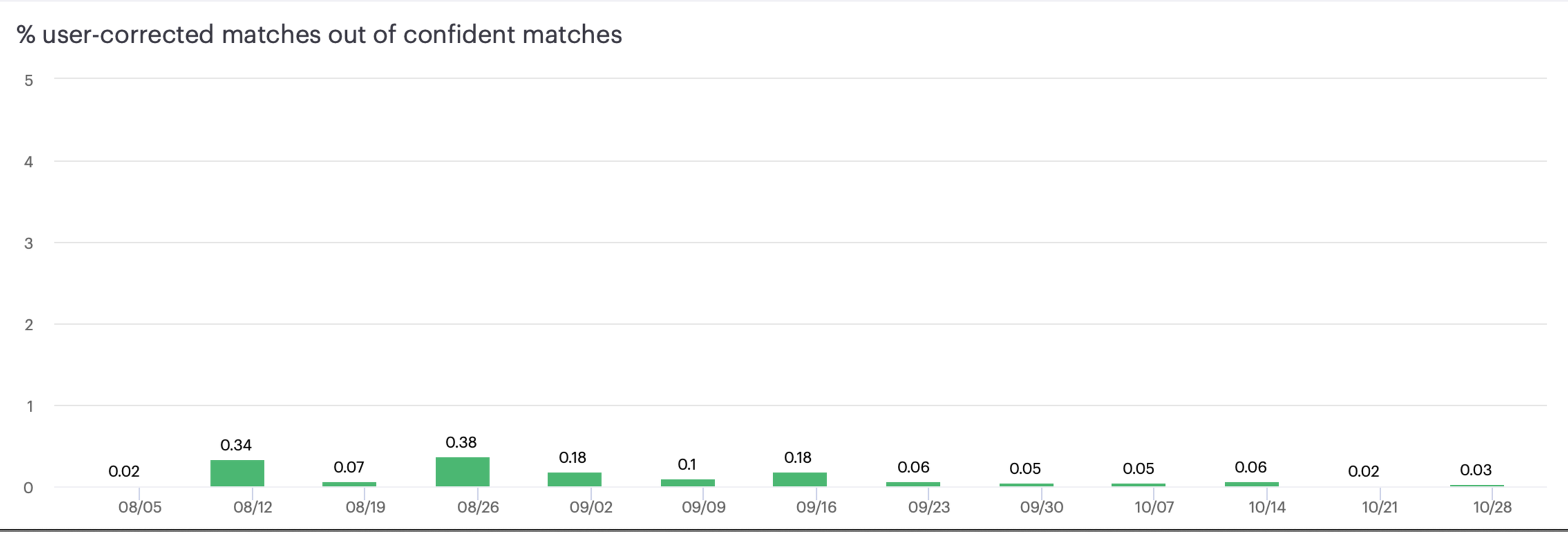
Roster Matching - Recent Performance



Closing the Flywheel

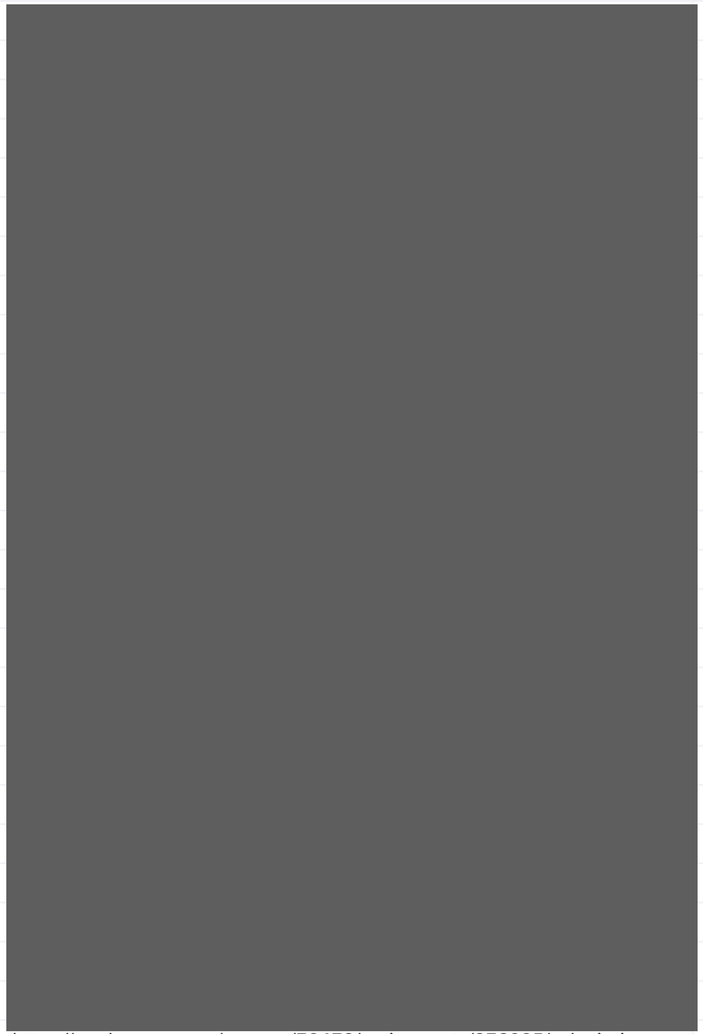


Closing the Flywheel

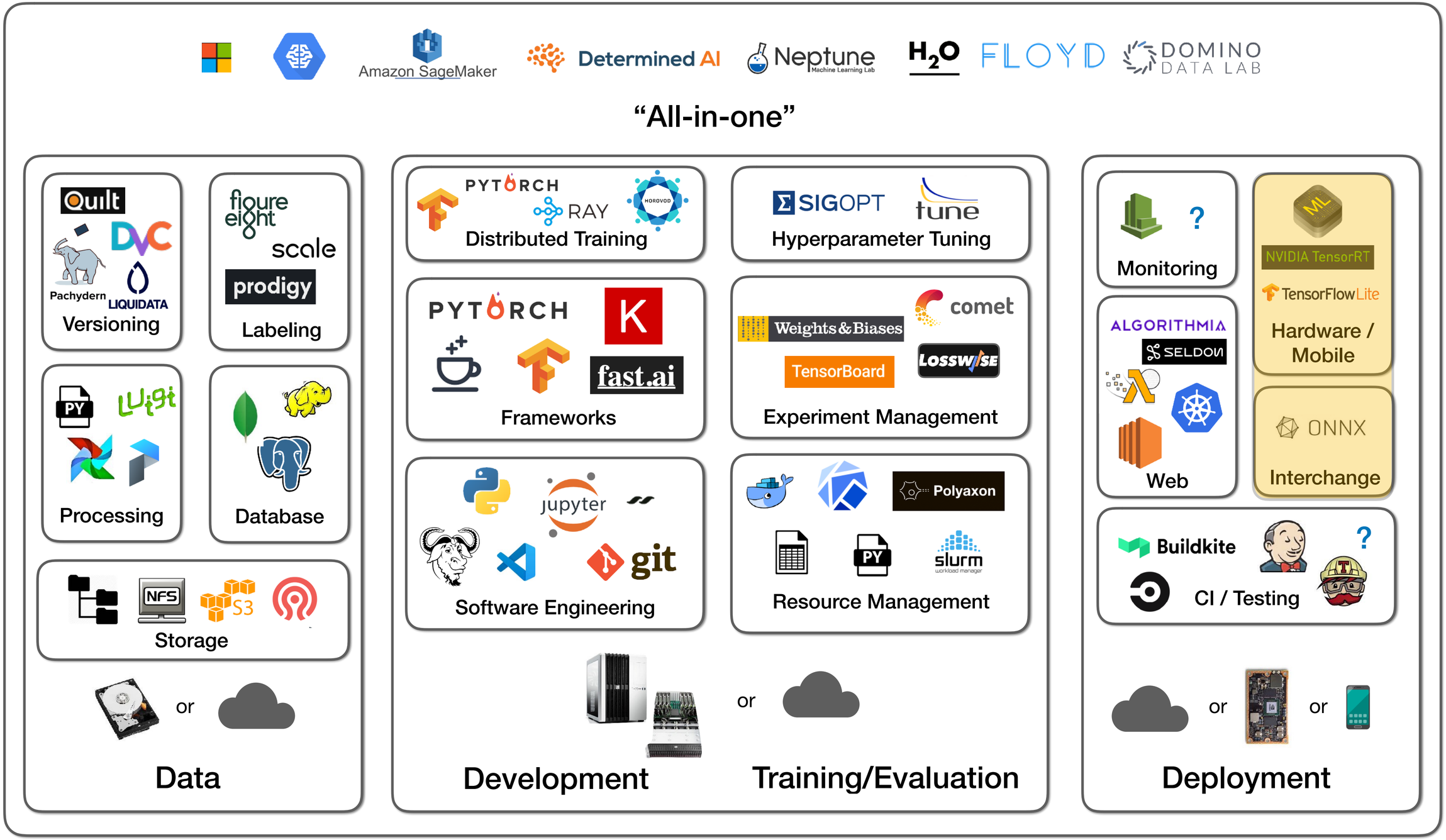


Closing the Flywheel

assignments with mistakes

	format	created_at	name_region	sid_region	num_submissions	num_confident	num_mistakes
1		2019-10-29 14:43:...	true	true	30	29	6
2		2019-10-28 19:16:...	true		5	2	1
3		2019-10-28 17:43:...	true	true	53	44	1
4		2019-10-26 20:53:...	true	true	182	148	1
5		2019-10-24 20:44:...	true		219	10	6
6		2019-10-24 18:18:...	true	true	185	171	1
7		2019-10-24 17:30:...	true	true	243	212	1
8		2019-10-24 17:14:...	true	true	80	64	1
9		2019-10-24 02:45:...	true	true	106	97	1
10		2019-10-23 18:53:...	true	true	209	203	3
11		2019-10-23 18:11:...	true	true	118	97	2
12		2019-10-23 17:55:...	true	true	595	555	1
13		2019-10-23 17:42:...	true	true	49	38	1
14		2019-10-23 15:18:...	true	true	87	86	1
15		2019-10-23 03:36:...	true		99	12	5
16		2019-10-22 21:21:...	true		220	8	5
17		2019-10-22 20:15:...	true	true	22	22	1
18		2019-10-22 04:37:...	true	true	98	64	1
19		2019-10-22 02:14:...	true	true	300	276	1
20		2019-10-22 01:26:...	true		252	112	2
21		2019-10-21 21:24:...	true		21	19	1
22		2019-10-21 20:24:...	true	true	210	202	1
23		2019-10-21 18:32:...	true	true	33	19	1
24		2019-10-21 16:51:...	true		194	20	7
25		2019-10-21 04:37:...	true	true	41	9	5
26		2019-10-19 00:32:...	true	true	1298	1095	1
27	https://gradescope.com/courses/70472/assignments/276905/submissions	2019-10-18 23:16:...	true	true	154	140	1

Questions?


Amazon SageMaker
Determined AI
Neptune
Machine Learning Lab
H2O
FLOYD
DOMINO DATA LAB
"All-in-one"
Quilt
DVC
Pachydern
LIQUIDATA
Versioning
figure eight
scale
prodigy
Labeling
PY
Processing
Database
NFS
S3
Storage
or
Data
PYTORCH
RAY
HOROVOD
Distributed Training
SIGOPT
tune
Hyperparameter Tuning
PYTORCH
K
fast.ai
Frameworks
Weights & Biases
comet
TensorBoard
LOSSWISE
Experiment Management
jupyter
git
Software Engineering
Polyaxon
PY
slurm
workload manager
Resource Management
or
Development
Training/Evaluation
Monitoring
ML
NVIDIA TensorRT
TensorFlowLite
Hardware / Mobile
ALGORITHMIA
SELDON
Web
ONNX
Interchange
Buildkite
CI / Testing
or
or
Deployment

Problems

- Embedded and mobile frameworks are less fully featured than full PyTorch/Tensorflow
  - Have to be careful with architecture
  - Interchange format
- Embedded and mobile devices have little memory and slow/expensive compute
  - Have to reduce network size / quantize weights / distill knowledge

Problems

- Embedded and **mobile frameworks** are less fully featured than full PyTorch/Tensorflow
  - Have to be careful with architecture
  - Interchange format
- Embedded and mobile devices have little memory and slow/expensive compute
  - Have to reduce network size / **quantize weights** / distill knowledge

Tensorflow Lite

- Not all models will work, but much better than "Tensorflow Mobile" used to be

Pick a model

Pick a new model or retrain an existing one.

Convert

Convert a TensorFlow model into a compressed flat buffer with the TensorFlow Lite Converter.

Deploy

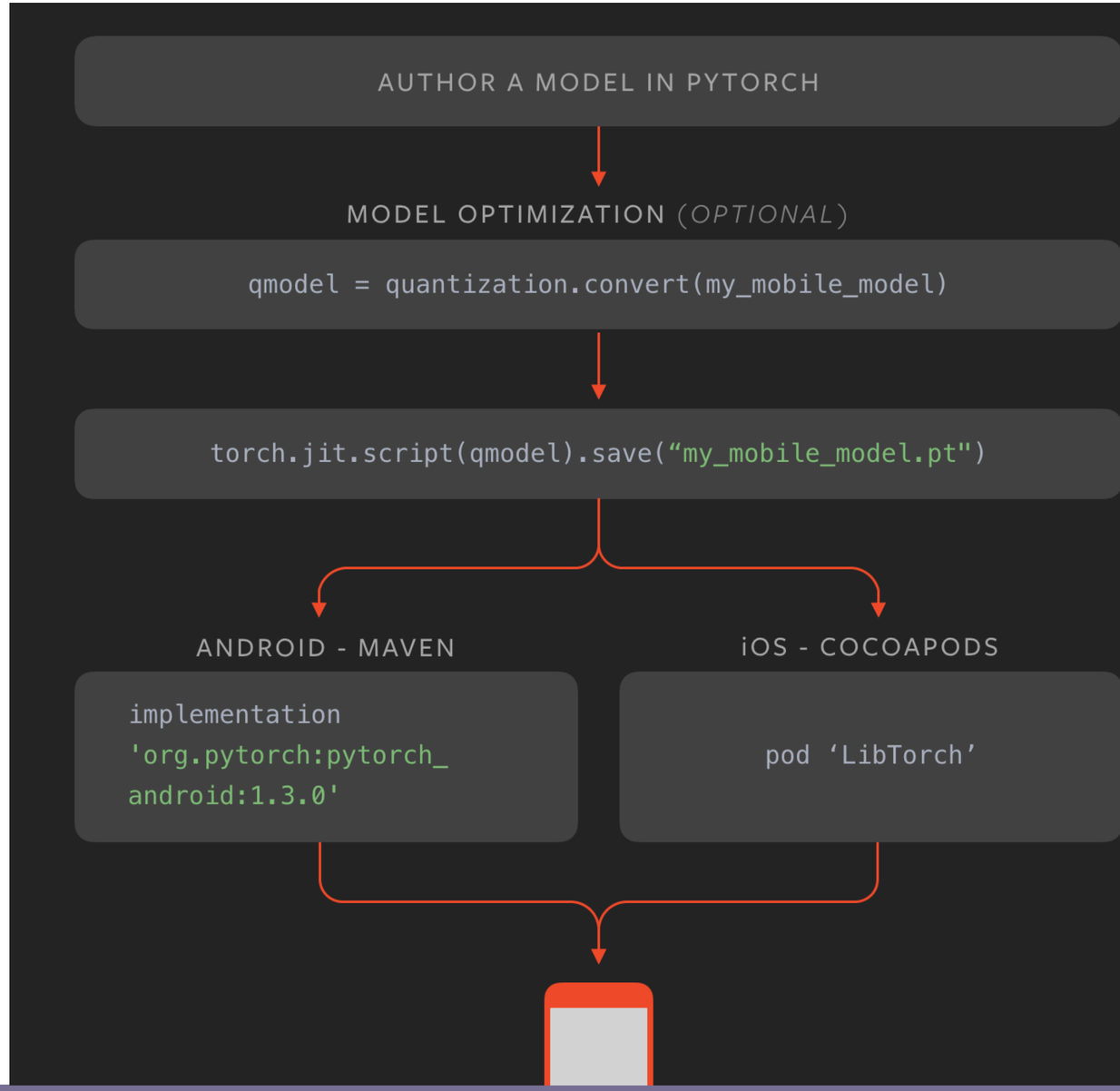
Take the compressed .tflite file and load it into a mobile or embedded device.

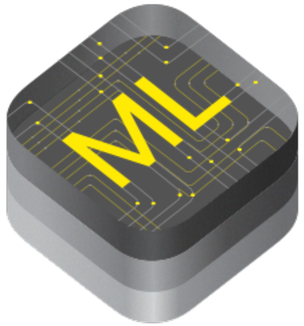
Optimize

Quantize by converting 32-bit floats to more efficient 8-bit integers or run on GPU.

PyTorch Mobile

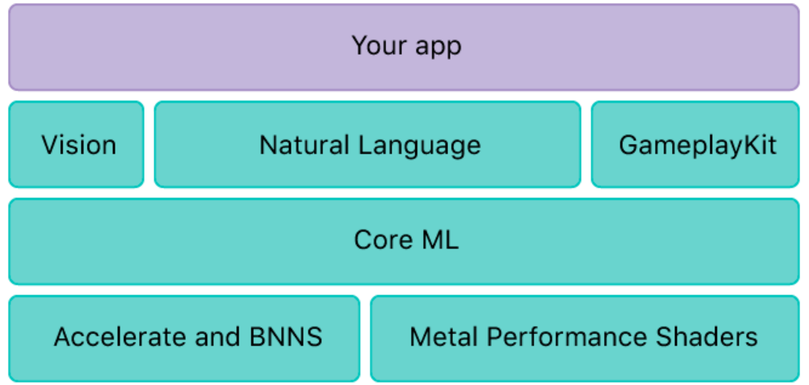
- Optional quantization
- TorchScript: static execution graph
- Libraries for Android and iOS





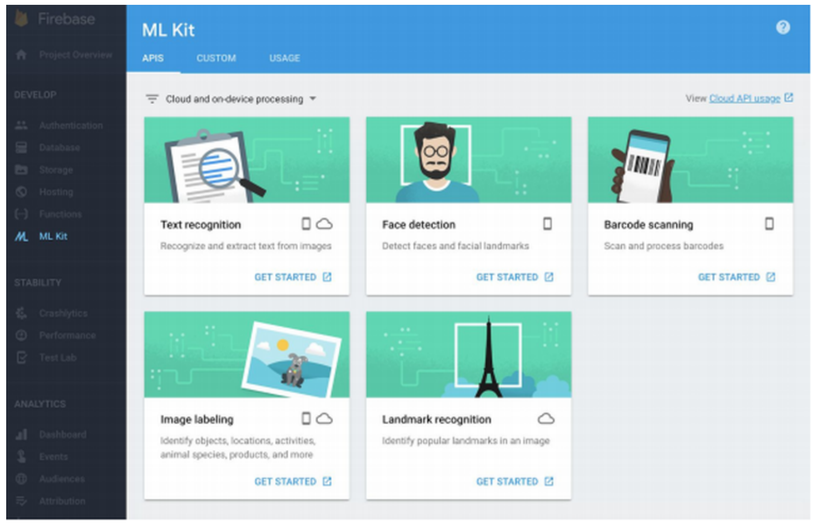
CoreML

- Released at Apple WWDC 2017
- Inference only
- https://coreml.store/



ML Kit

- Announced at Google I/O 2018
- Either via API or on-device
- Offers pre-trained models, or can upload Tensorflow Lite model



- Supposed to work with both CoreML and MLKit

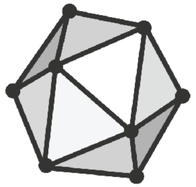
https://heartbeat.fritz.ai/core-ml-vs-ml-kit-which-mobile-machine-learning-framework-is-right-for-you-e25c5d34c765

Problems

- Embedded and mobile frameworks are less fully featured than full PyTorch/Tensorflow
  - Have to be careful with architecture
  - **Interchange format**
- Embedded and mobile devices have little memory and slow/expensive compute
  - Have to reduce network size / quantize weights / distill knowledge

Interchange Format

- Open Neural Network Exchange: open-source format for deep learning models
- The dream is to mix different frameworks, such that frameworks that are good for development (PyTorch) don't also have to be good at inference (Caffe2)



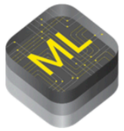
ONNX

Frameworks

Caffe2 Chainer Cognitive Toolkit mxnet PyTorch PaddlePaddle

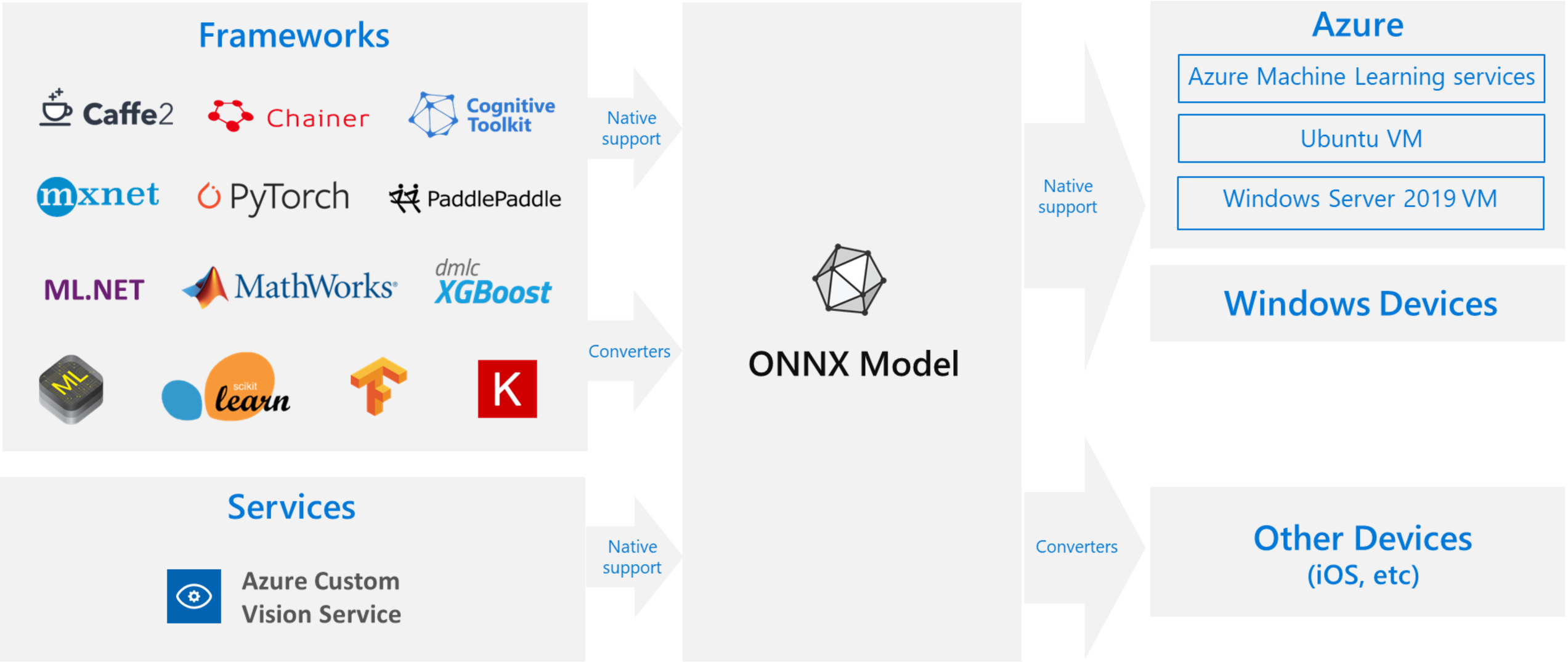
Converters

Runtimes



MathWorks  NVIDIA Qualcomm SOPHON vespa Windows

Interchange Format

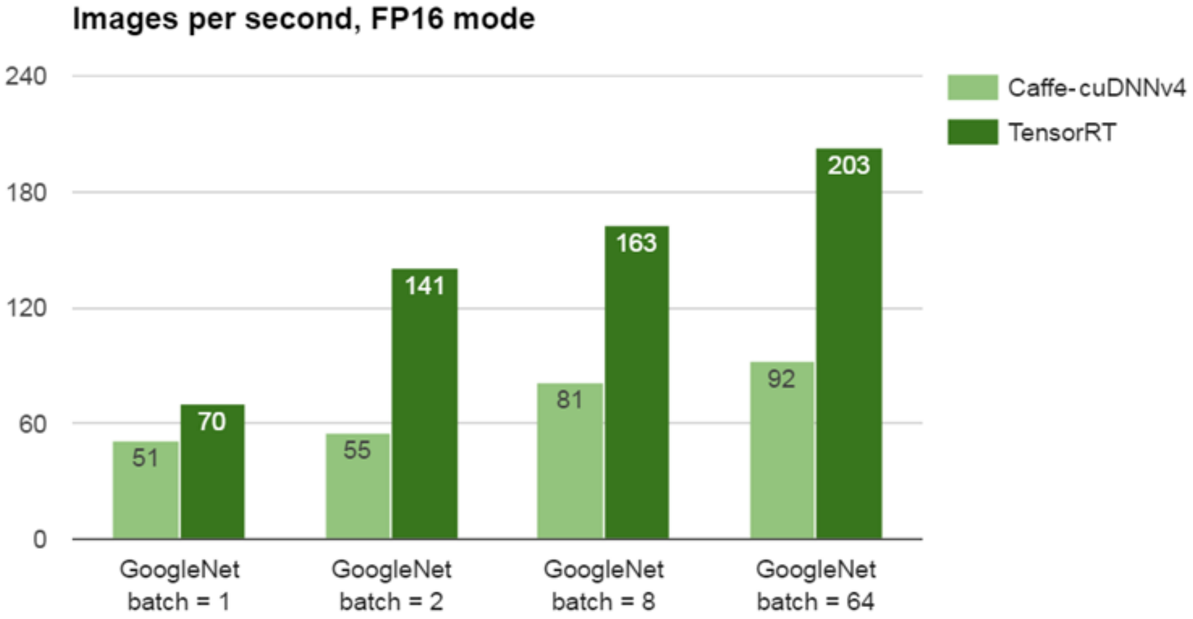


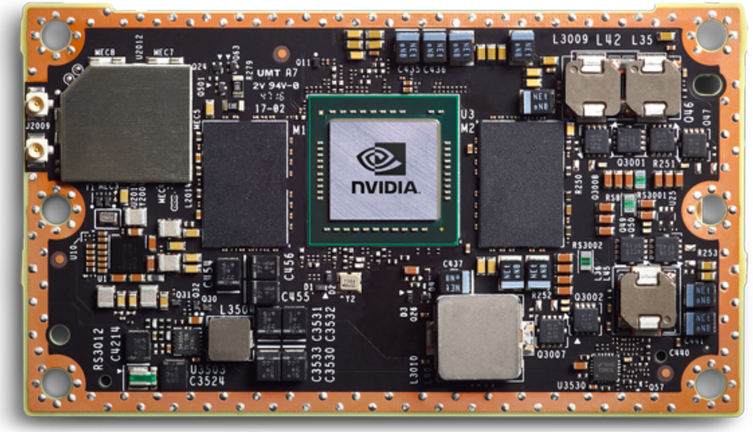
https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-onnx

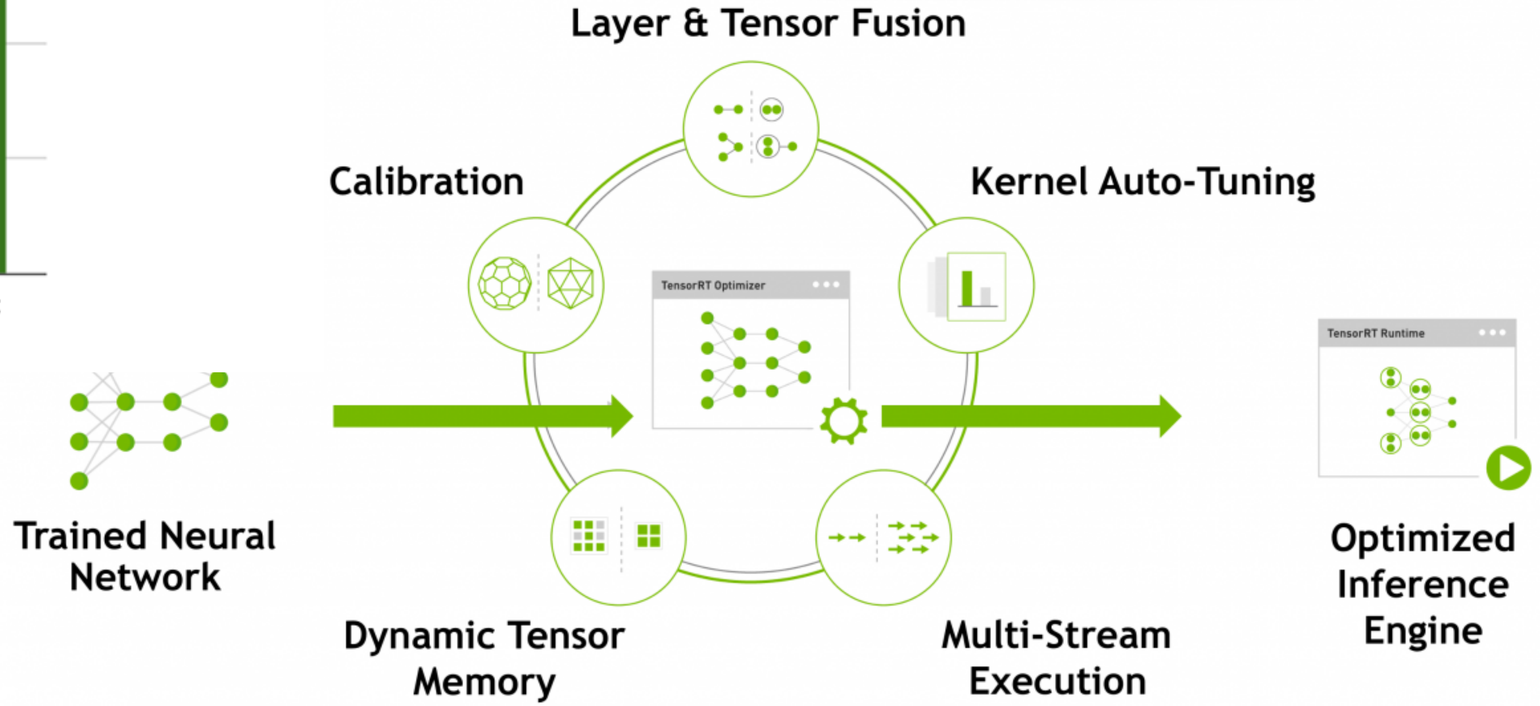
Problems

- **Embedded** and mobile frameworks are less fully featured than full PyTorch/Tensorflow
  - Have to be careful with architecture
  - Interchange format
- Embedded and mobile devices have little memory and slow/expensive compute
  - Have to reduce network size / **quantize weights** / distill knowledge

NVIDIA for Embedded



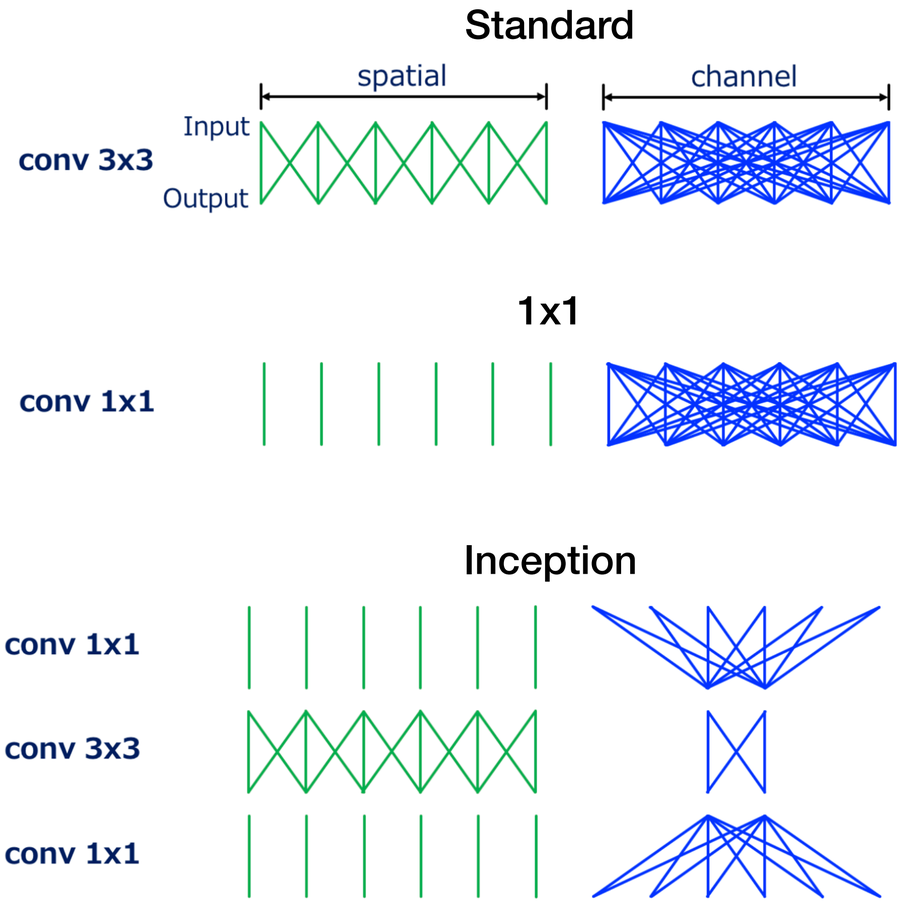


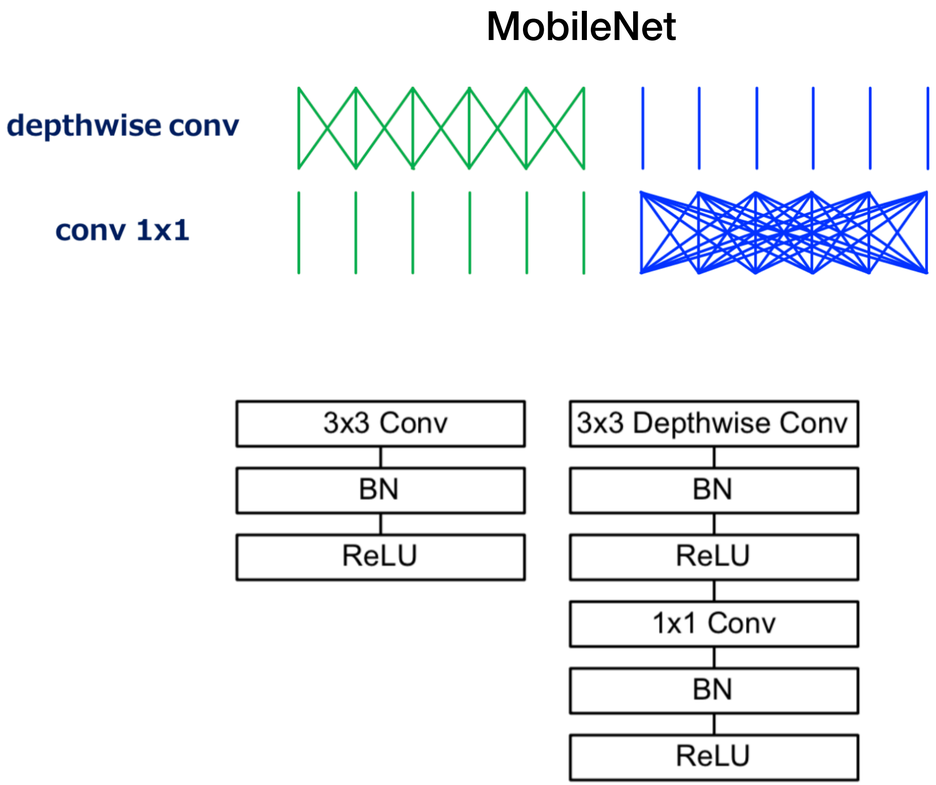


Problems

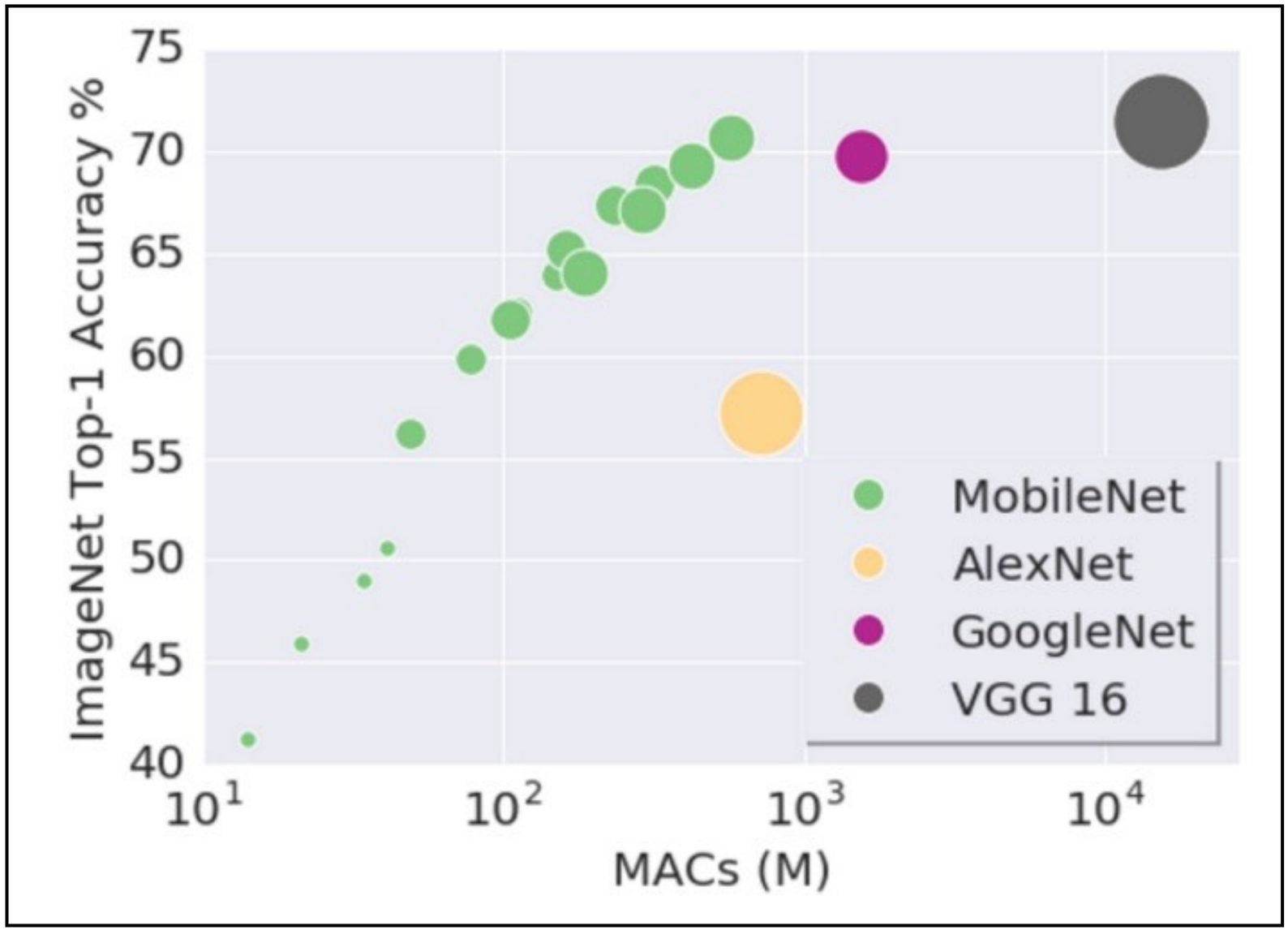
- Embedded and mobile frameworks are less fully featured than full PyTorch/Tensorflow
  - Have to be careful with architecture
  - Interchange format
- Embedded and mobile devices have little memory and slow/expensive compute
  - Have to **reduce network size** / quantize weights / distill knowledge

MobileNets





https://medium.com/@yu4u/why-mobilenet-and-its-variants-e-g-shufflenet-are-fast-1c7048b9618d



https://medium.com/@yu4u/why-mobilenet-and-its-variants-e-g-shufflenet-are-fast-1c7048b9618d

Problems

- Embedded and mobile frameworks are less fully featured than full PyTorch/Tensorflow
  - Have to be careful with architecture
  - Interchange format
- Embedded and mobile devices have little memory and slow/expensive compute
  - Have to reduce network size / quantize weights / **distill knowledge**

Knowledge distillation

- First, train a large "teacher" network on the data directly
- Next, train a smaller "student" network to match the "teacher" network's prediction distribution

Recommended case study: DistilBERT

Using the teacher signal, we are able to train a **smaller language model,** we call **DistilBERT,** from the **supervision of BERT** 🤱 (we used the English `bert-base-uncased` version of BERT).

Following Hinton et al., the training loss is a linear combination of the *distillation loss* and the masked *language modeling loss*. Our student is a small version of BERT in which we *removed the token-type embeddings and the pooler* (used for the next sentence classification task) and kept the rest of the architecture identical while reducing the numbers of layers by a factor of two.

Overall, our distilled model, DistilBERT, has about half the total number of parameters of BERT base and retains 95% of BERT's performances on the language understanding benchmark GLUE.

https://medium.com/huggingface/distilbert-8cf3380435b5

Questions?

Thank you!