infaas

Automated *Model-less* Inference Serving

Francisco Romero[*], **Qian Li**[*],
Neeraja J. Yadwadkar, and Christos Kozyrakis

PLATFORMLAB
Stanford MAST

Model Registration

User

Register a model

Model Repository

Today's Inference Serving

Invoke a Model

User(s)

Today's Inference Serving

Inference

Select a model

∑-ResNet

∑-DenseNet

Framework

mxnet

Users must select the right *model-variant*

Hardware architecture

CPUs

GPUs

FPGAs

ASICs

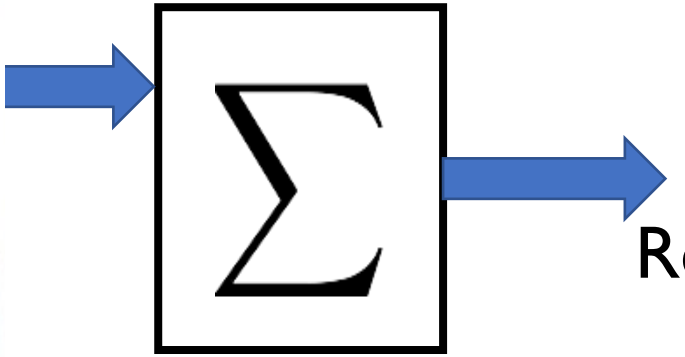
Optimizer

TensorRT

TVM

Diverse application requirements

Object Detection



Face Recognition

Example: Face Recognition

Accuracy Latency

Social Media

Latency
Accuracy

Cost

Navigation for visually impaired person

Applications evolve over time

The performance penalty can be up to 100x

Today's Inference serving

Invoke a Model

User(s) → Today's Inference Serving

Inference

Select a model

∑-ResNet
∑-DenseNet
∑-SqueezeNet
∑-Inception

Framework

Caffe2
mxnet
PYTORCH

Hardware architecture

CPUs
GPUs
FPGAs
ASICs

Optimizer

TensorRT
TVM

Model-less Inference serving

Prediction task,
App Reqs

User(s) → Model-less Inference Serving

Inference

Prediction task:

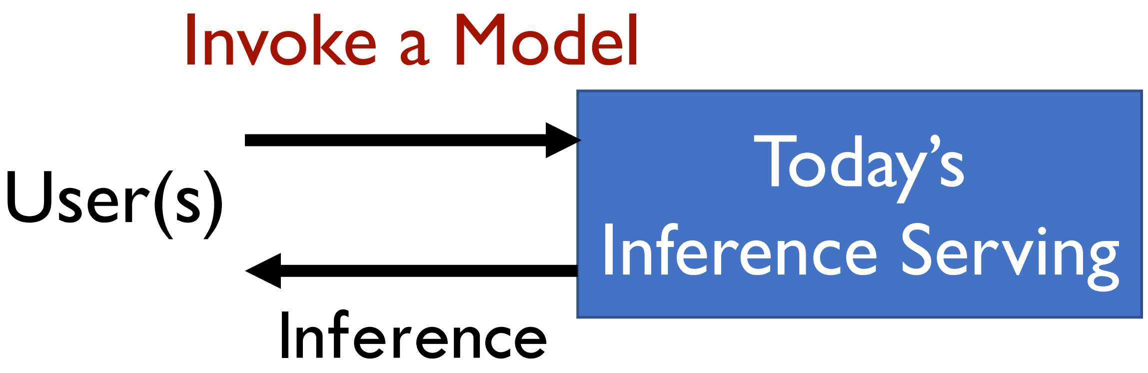
"Face Recognition"

App Reqs:

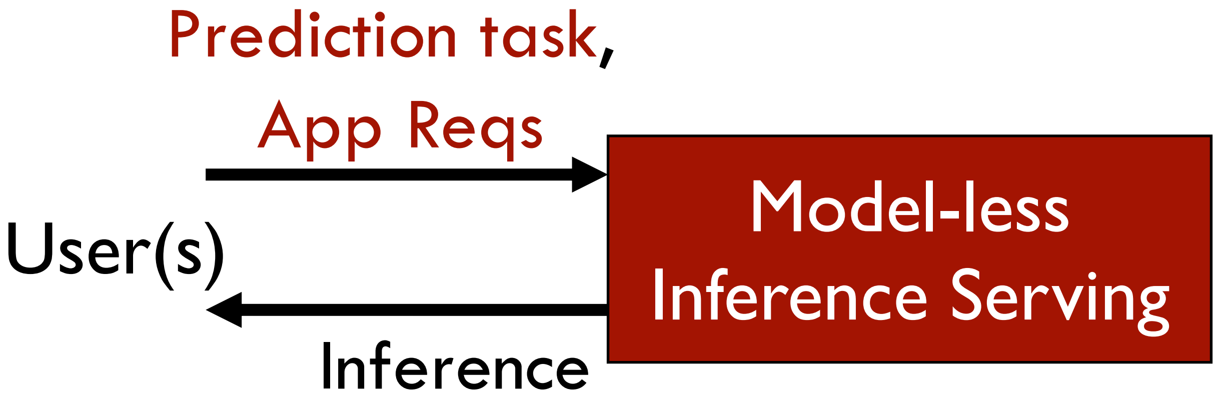
Target accuracy

Target latency

Today's Inference serving



Model-less Inference serving



Easy-to-use: Automatically and efficiently select a model and hardware

Cost Efficient: Share the hardware as well as models across users

INFerence-as-a-Service system

INFaaS provides a *model-less API* to inference queries that abstracts (a) Model Selection and (b) Resource Provisioning from users.

INFaaS is open-source!
https://stanford-mast.github.io/INFaaS/

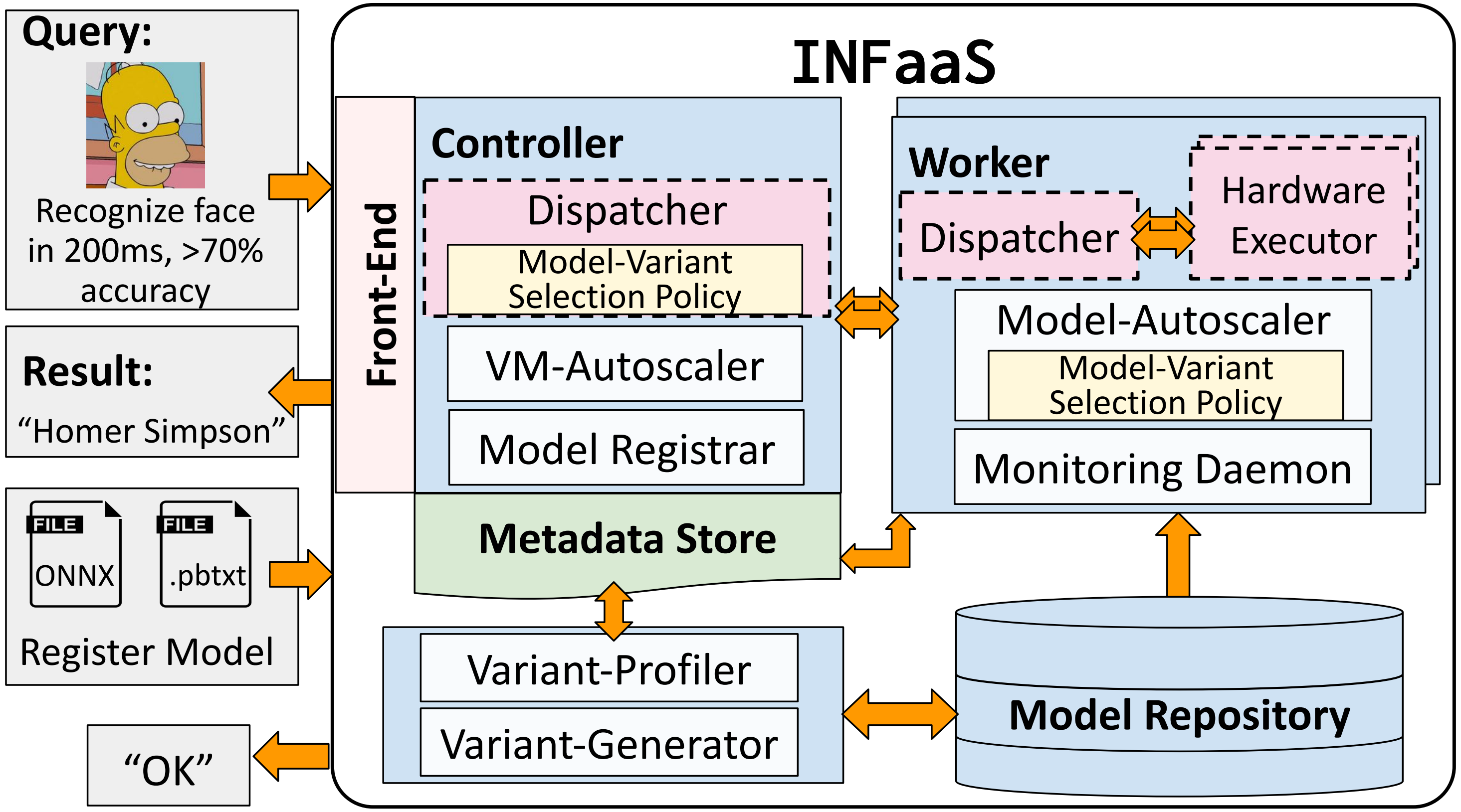
Goals & Requirements

Ease-of-use: Automatically select a model and hardware

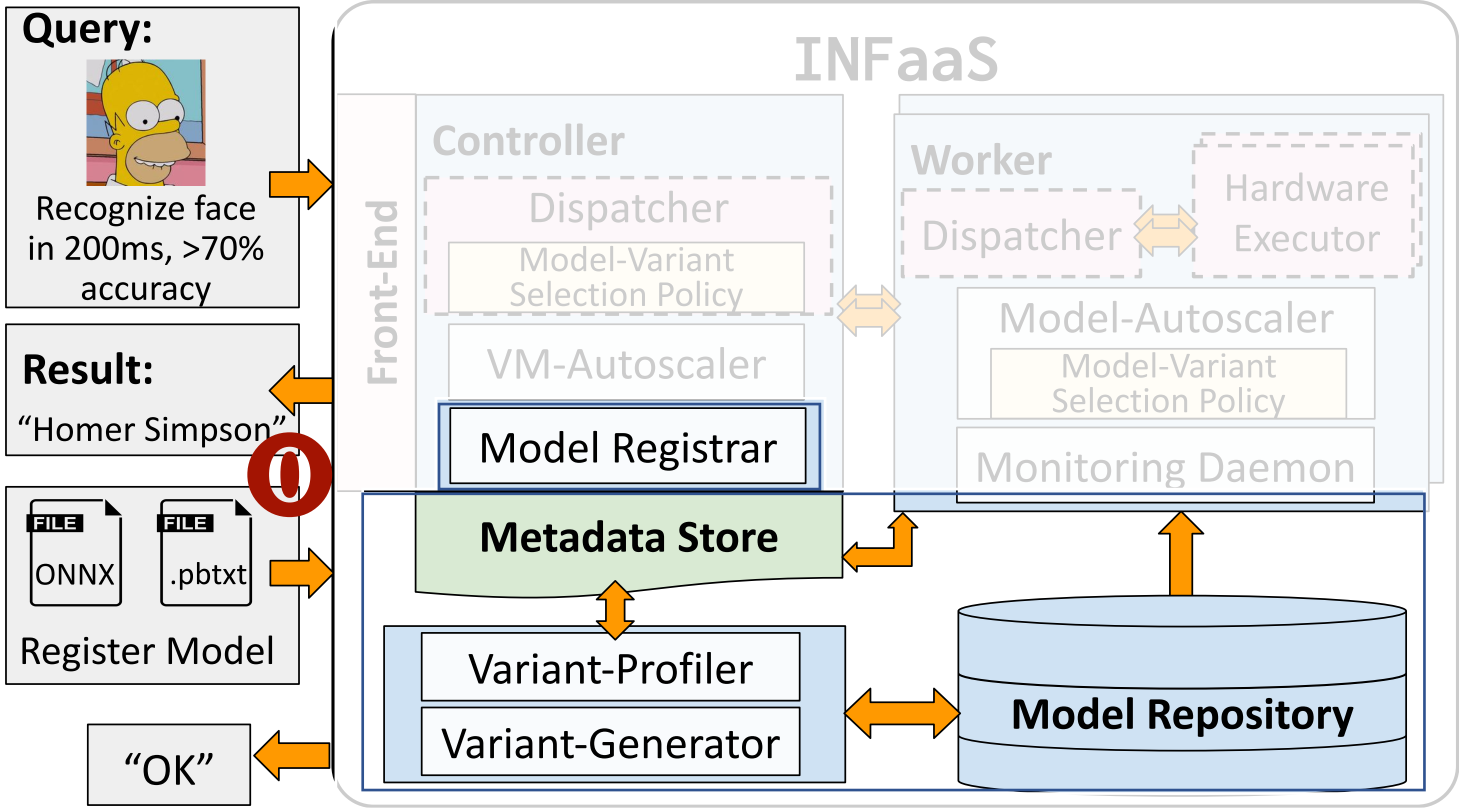
Challenges

- Novice and expert users
- Diverse user requirements
- Large search space
- Decision overhead

INFaaS overview

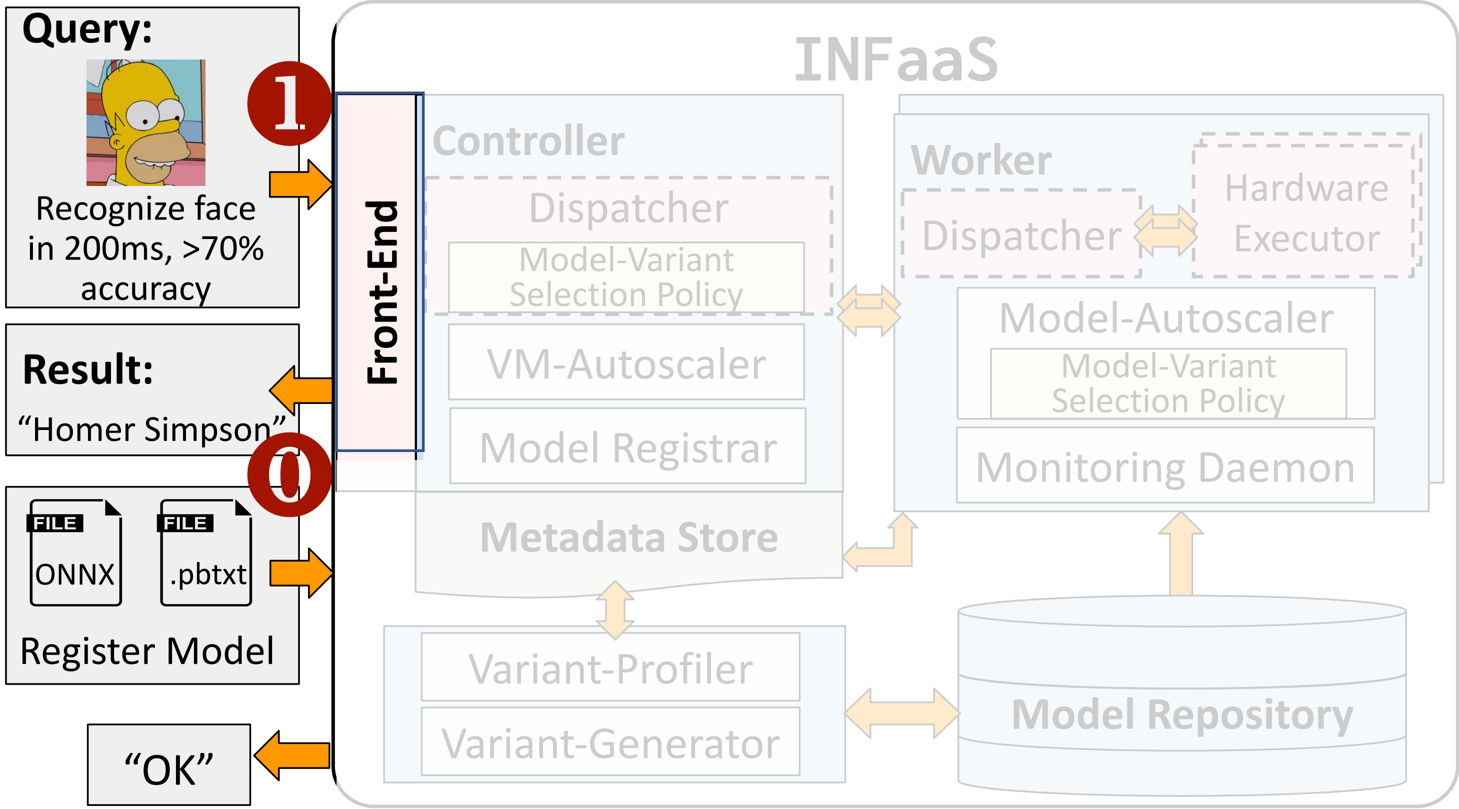


INFaaS overview



0 Users register models

INFaaS overview



❶ Users register models

❶ The user submits a query using INFaaS' user API

Front-end

- Goal: need to map query requirements to models and resources
    - Affects user API, metadata organization, model-variant selection, and autoscaling

- Challenge: needs to be intuitive

The *model-less* abstraction

Registered model

bert-pytorch-cpu

resnet50-tf-cpu

resnet50-caffe2-gpu

The *model-less* abstraction

Prediction task

translation

face-detection

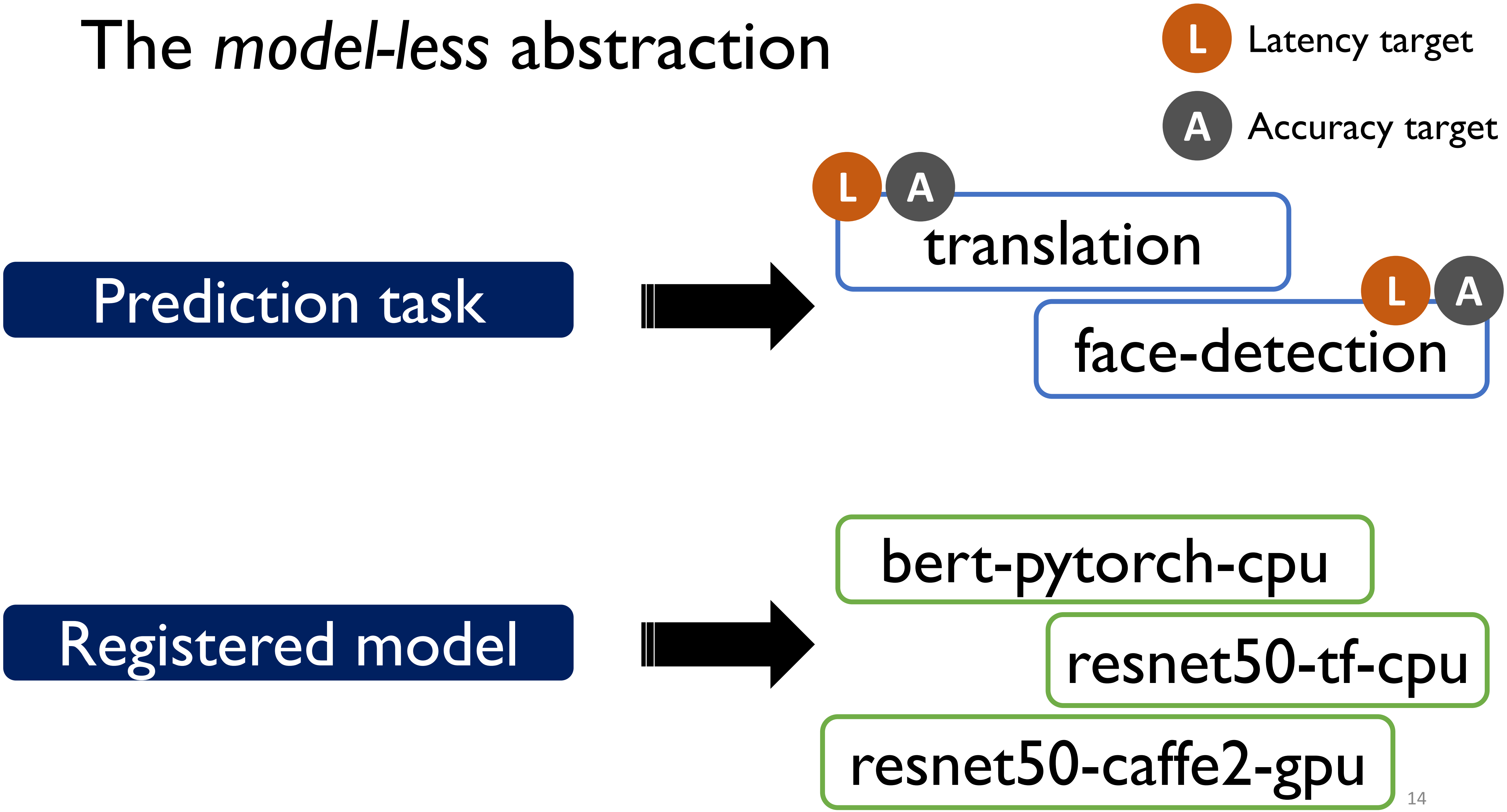
Registered model

bert-pytorch-cpu

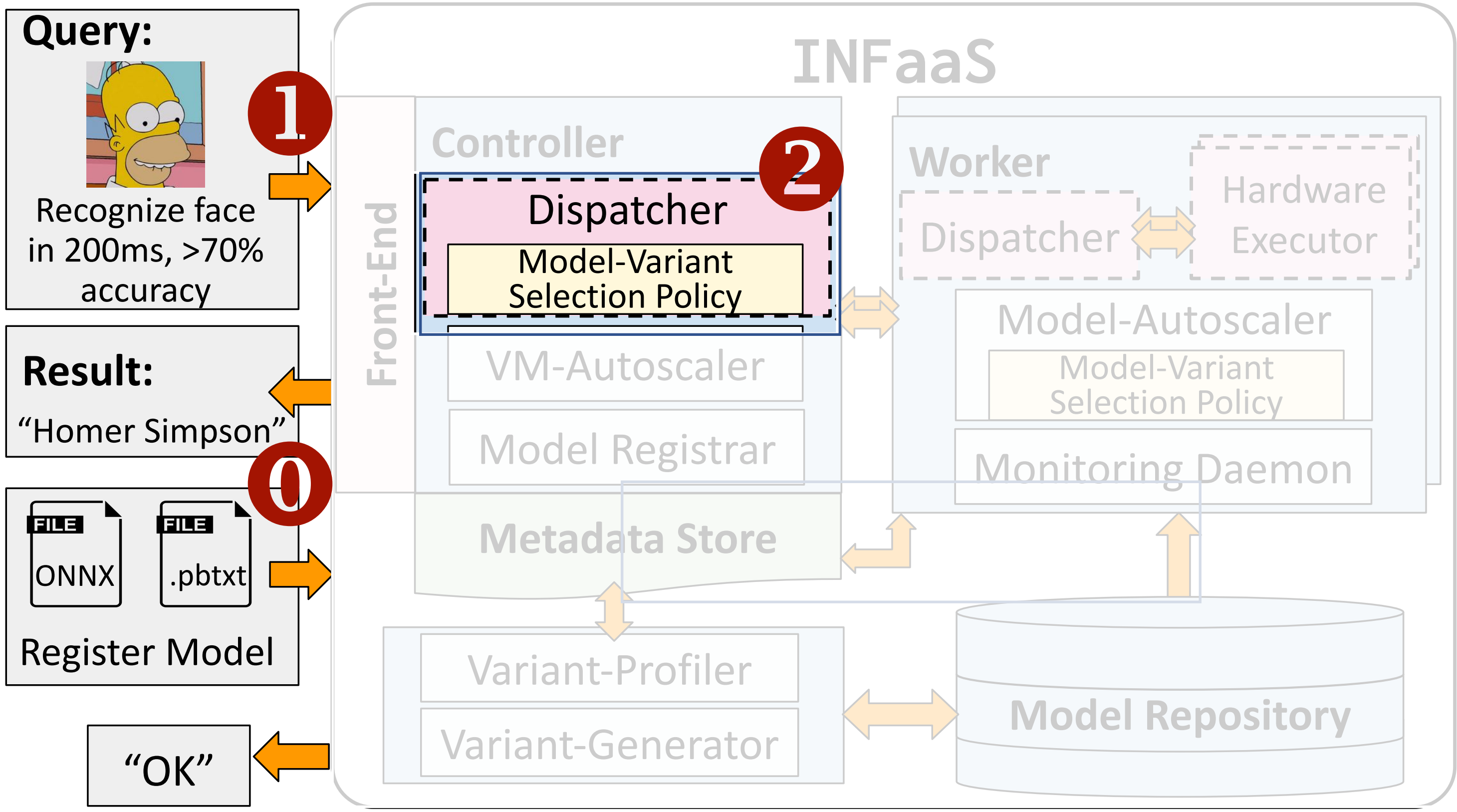
resnet50-tf-cpu

resnet50-caffe2-gpu

The *model-less* abstraction



INFaaS overview

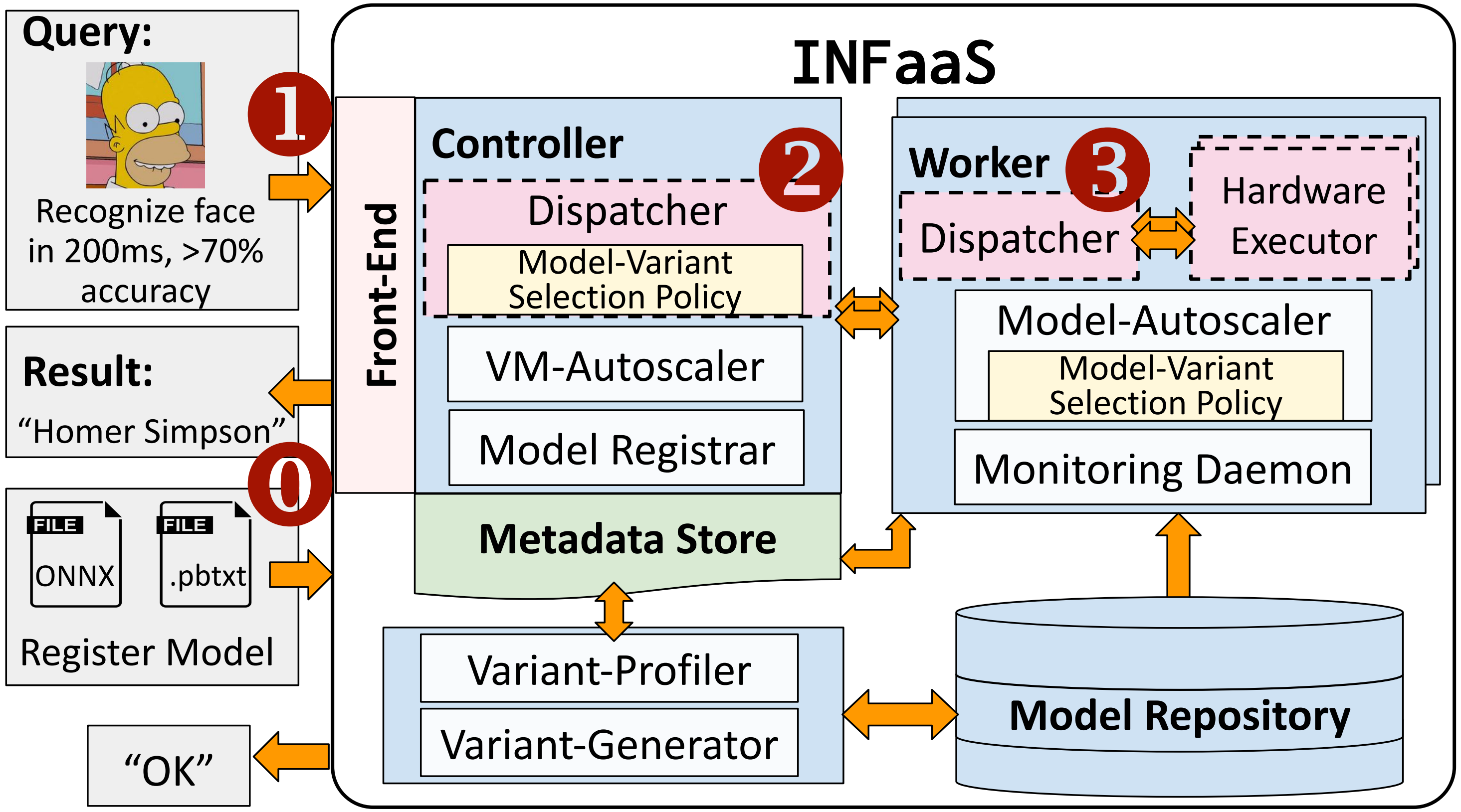


0 Users register models

1 The user submits a query using INFaaS' user API

2 The Controller selects a model-variant, then selects a worker to process the query

INFaaS overview



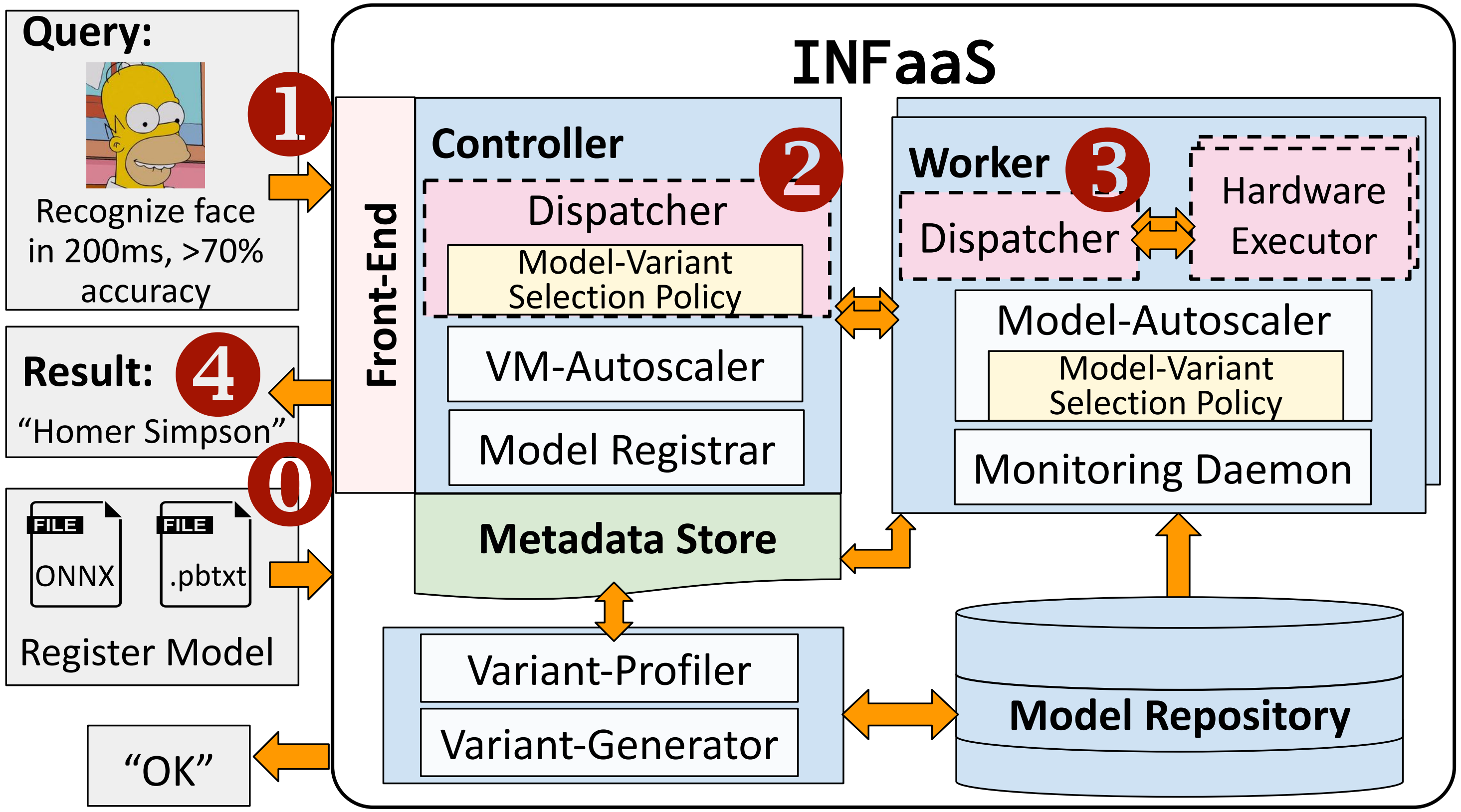
⓿ Users register models

❶ The user submits a query using INFaaS' user API

❷ The Controller selects a model-variant, then selects a worker to process the query

❸ The query proceeds to run on the variant's target hardware platform

INFaaS overview



⓿Users register models

❶The user submits a query using INFaaS' user API

❷The Controller selects a model-variant, then selects a worker to process the query

❸The query proceeds to run on the variant's target hardware platform

❹Upon completion, the result is returned to the user

Ease-of-use and cost efficiency

Select a model

- ∑-ResNet
- ∑-DenseNet
- ∑-SqueezeNet
- ∑-Inception

Framework

Caffe2 mxnet PYTORCH

Hardware architecture

CPUs GPUs FPGAs ASICs

Optimizer

TensorRT TVM

INFaaS removes the system configuration burden and improves ease-of-use

Goals & Requirements

Challenges

Ease-of-use: Automatically select a model and hardware

- Novice and expert users
- Diverse user requirements
- Large search space
- Decision overhead

Autoscaling: allocate just enough resources, meet SLOs, minimize cost

- Query load and pattern changes
- Heterogeneous hardware & models
- Scalability

Existing systems

- Static provisioning (TensorFlow Serving, Triton Inference Server)

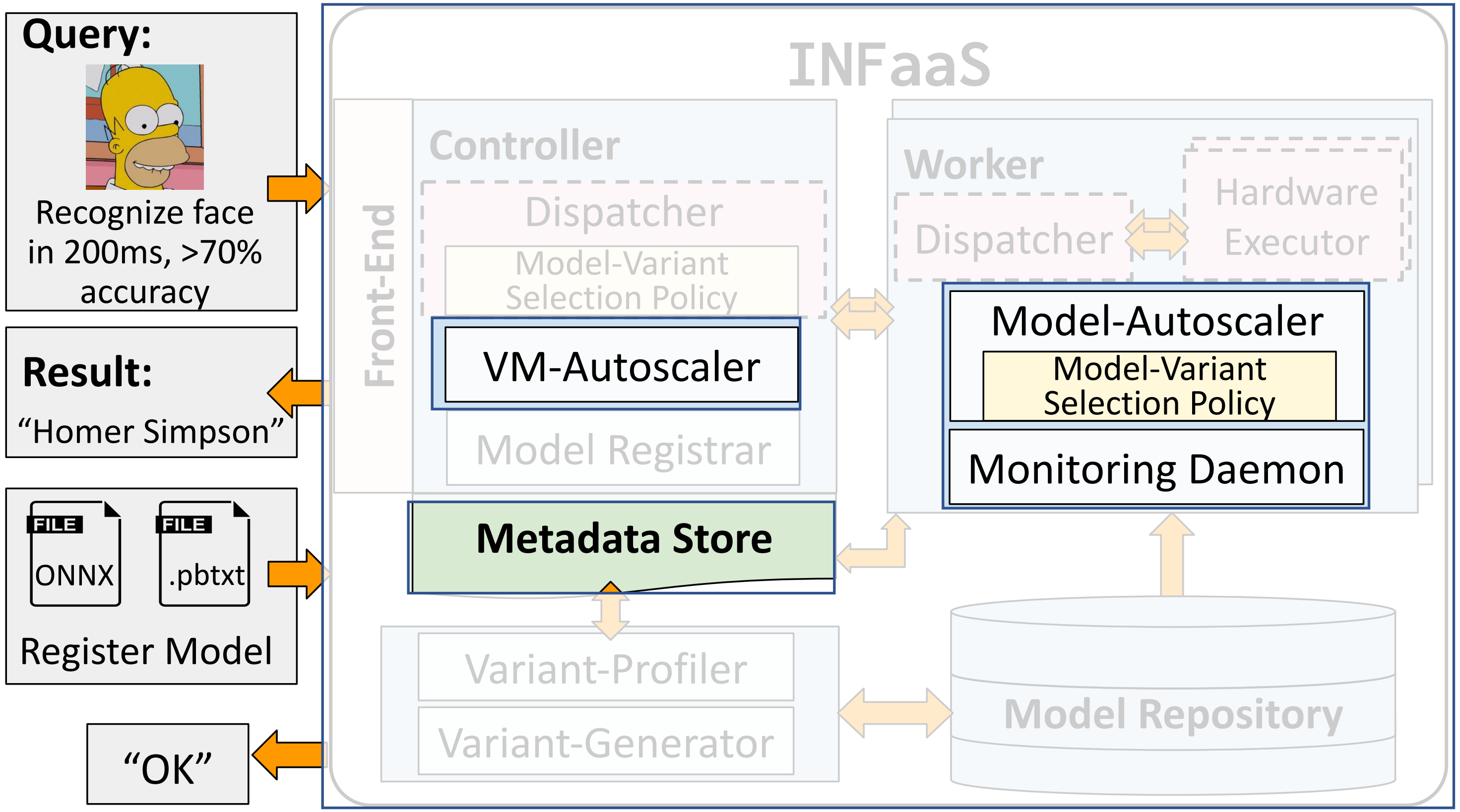
  - based on peak load

  - Meet SLOs but expensive
  - Waste resources at low load

Existing systems

- <u>Static provisioning</u> (TensorFlow Serving, Triton Inference Server) - based on peak load
  - Meet SLOs but expensive
  - Waste resources at low load
- <u>Replica-only</u> (Clipper, SageMaker, AI Platform) - replicate individual variants
  - Lower costs but high start-up latency
  - Fail to leverage heterogeneous resources / variants

Autoscaling

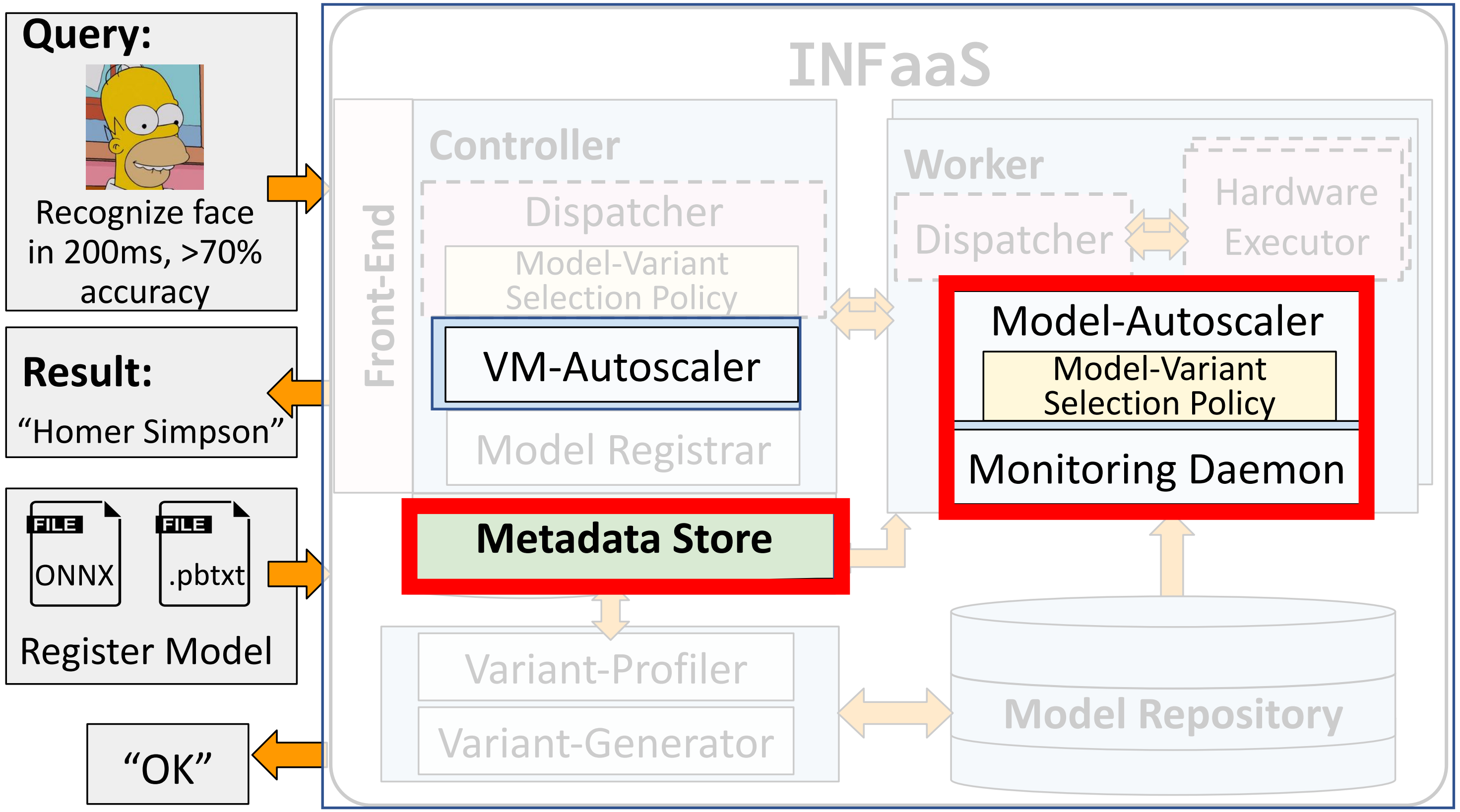


<u>3 types of scaling</u>

- Model-horizontal scaling
- Model-vertical scaling
  -> *(Our contribution)*
- VM-autoscaling

<u>Division of responsibility</u>

Autoscaling



<u>3 types of scaling</u>

- Model-horizontal scaling
- Model-vertical scaling
  -> *(Our contribution)*
- VM-autoscaling

<u>Division of responsibility</u>

Model-Autoscaler at each worker

- <u>Goal:</u> Decide the *type* and *number* of model-variants to meet the load and requirements, while minimizing cost
- Formulate as an Integer Programming problem

$$\min \mathrm{Cost}(\underset{\substack{\Downarrow \\ \text{\{load, unload\} variant instances}}}{\mathrm{action}}) = \textcolor{red}{\text{Hardware Cost}} + \lambda\, \textcolor{blue}{\text{Loading Latency}}$$

Model-Autoscaler at each worker

- Goal: Decide the *type* and *number* of model-variants to meet the load and requirements, while minimizing cost
- Formulate as an Integer Programming problem

$$\min \mathrm{Cost}(\underbrace{\mathrm{action}}_{\{\text{load, unload}\}\text{ variant instances}}) = \color{red}{\text{Hardware Cost}} + \lambda\, \color{blue}{\text{Loading Latency}}$$

Constraints:

(1) With the chosen scaling action, INFaaS supports the incoming query load.

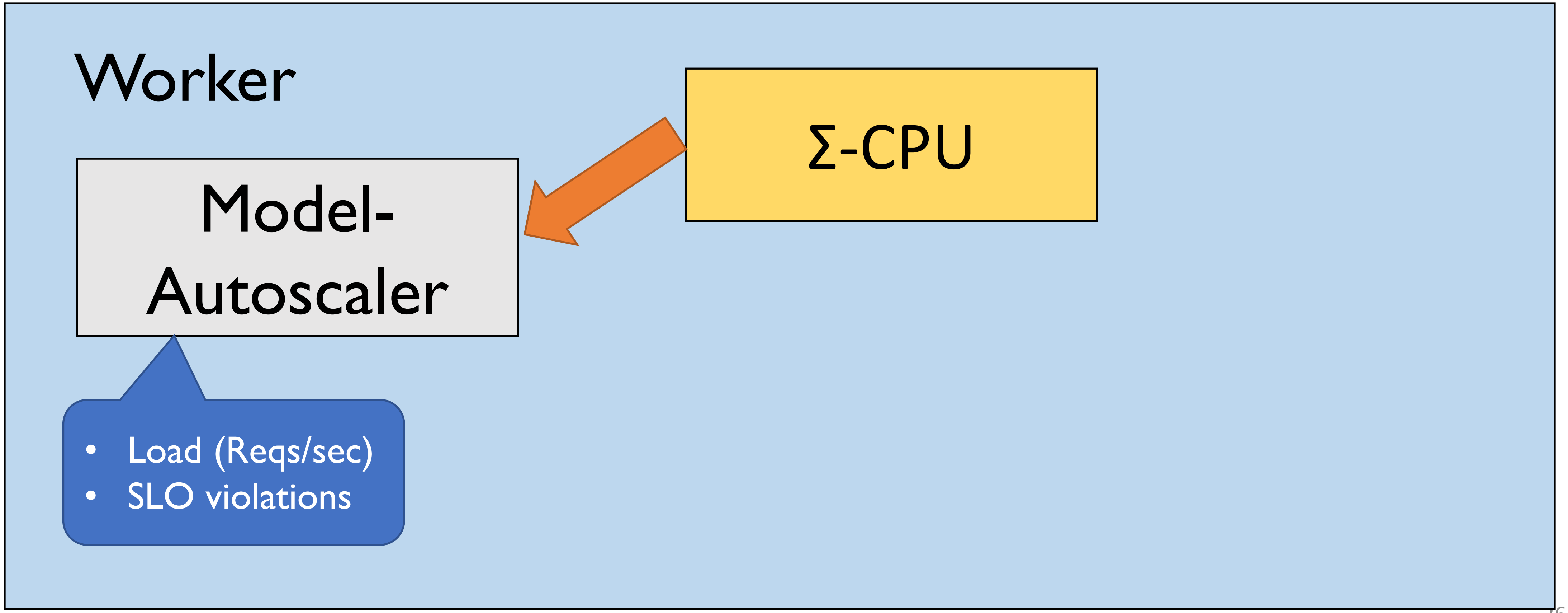
(2) The newly-loaded instances satisfy applications' SLOs.

(3) Do not exceed the total system resources.

(4) The number of running instances is non-negative.

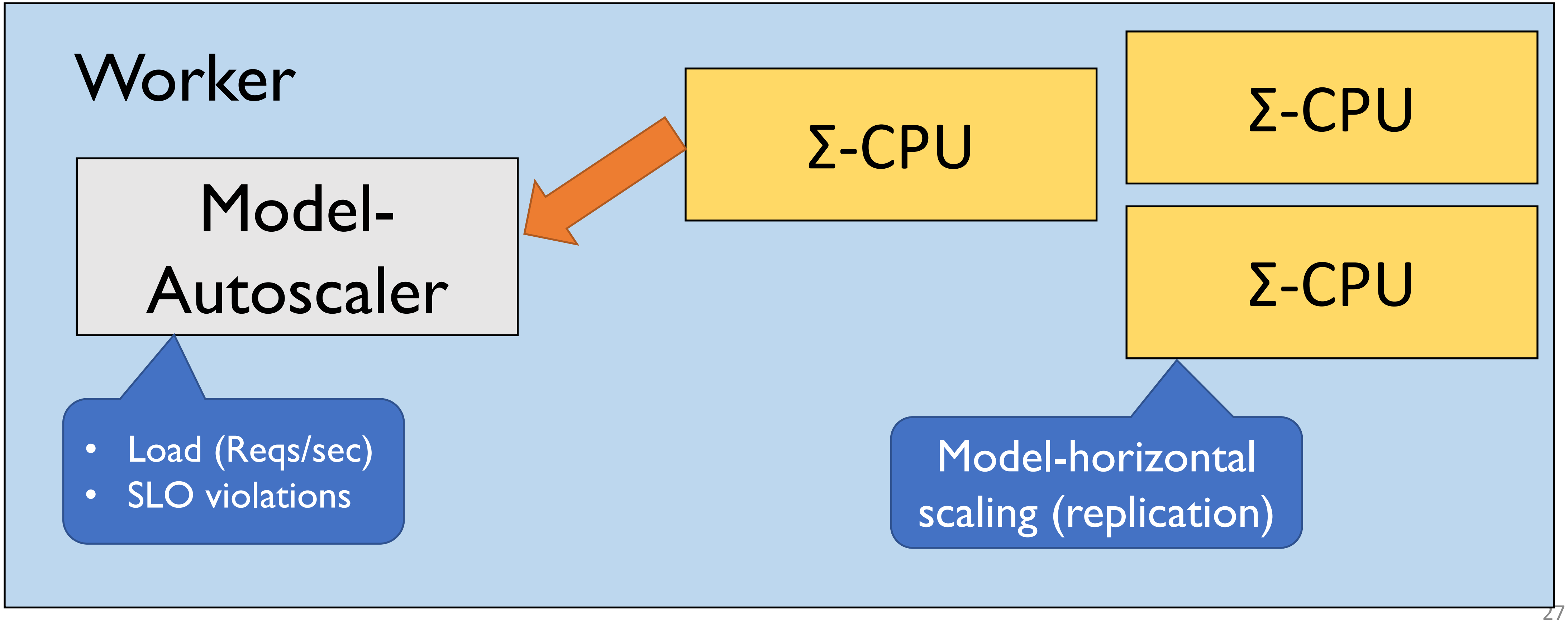
Model-Autoscaler at each worker

- Respond to changes in load and meet SLOs by: 1) model-horizontal scaling and 2) model-vertical scaling



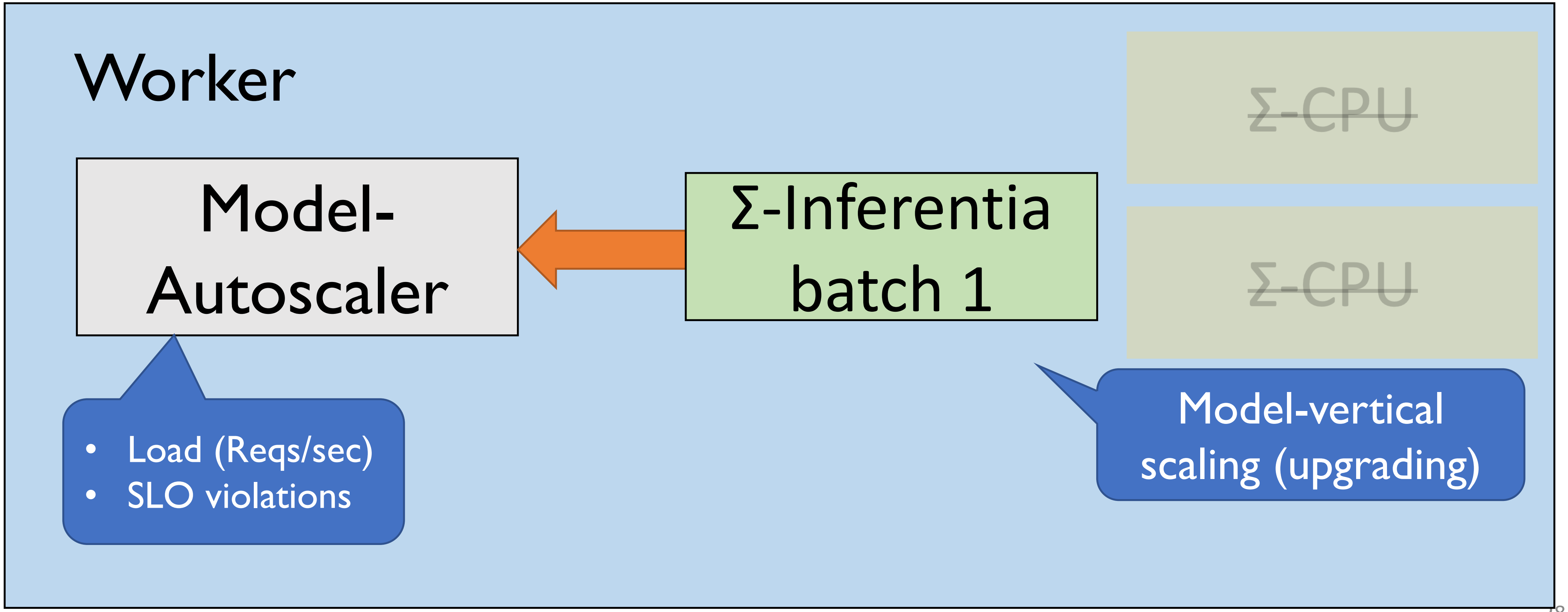
Model-Autoscaler at each worker

- Respond to changes in load and meet SLOs by: 1) model-horizontal scaling and 2) model-vertical scaling



Model-Autoscaler at each worker

- Respond to changes in load and meet SLOs by: 1) model-horizontal scaling and 2) model-vertical scaling



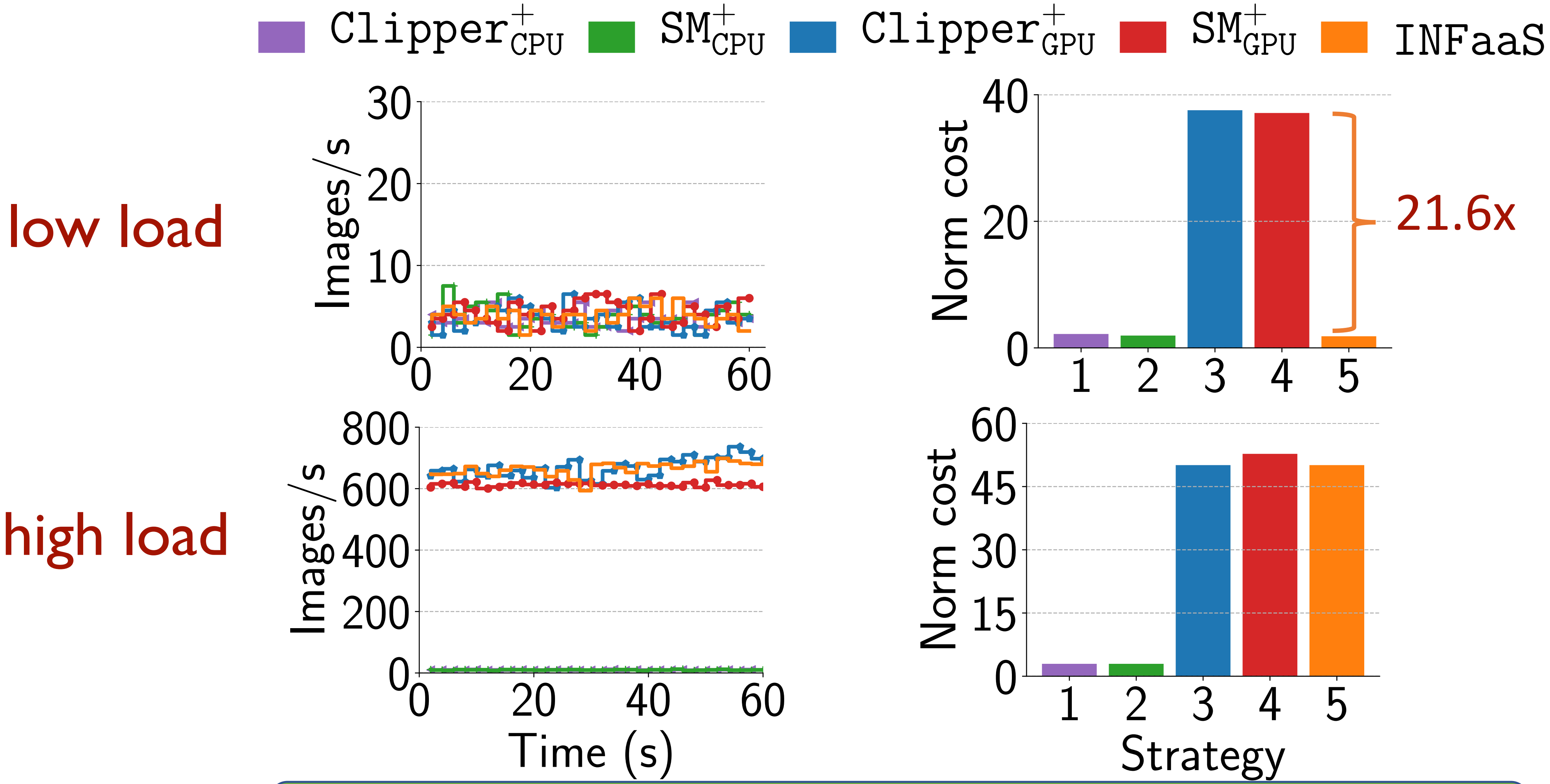
Evaluation

- Baselines:
  - CLIPPER$^+$ (Clipper, TIS and TFS): preloaded and persisted *beefy* variants
    - CLIPPER$^+_{GPU}$ and CLIPPER$^+_{CPU}$
  - SM$^+$ (InferLine, SageMaker, and AI Platform): model-horizontal scaling, replicated *light-weight* variants on/across worker
    - SM$^+_{GPU}$ and SM$^+_{CPU}$

Evaluation

- We deployed INFaaS on AWS EC2
  - GPU worker has 1 NVIDIA V100 GPU
  - Inferentia worker has 1 AWS Inferentia accelerator
  - Controller / CPU worker / client are CPU-only machines

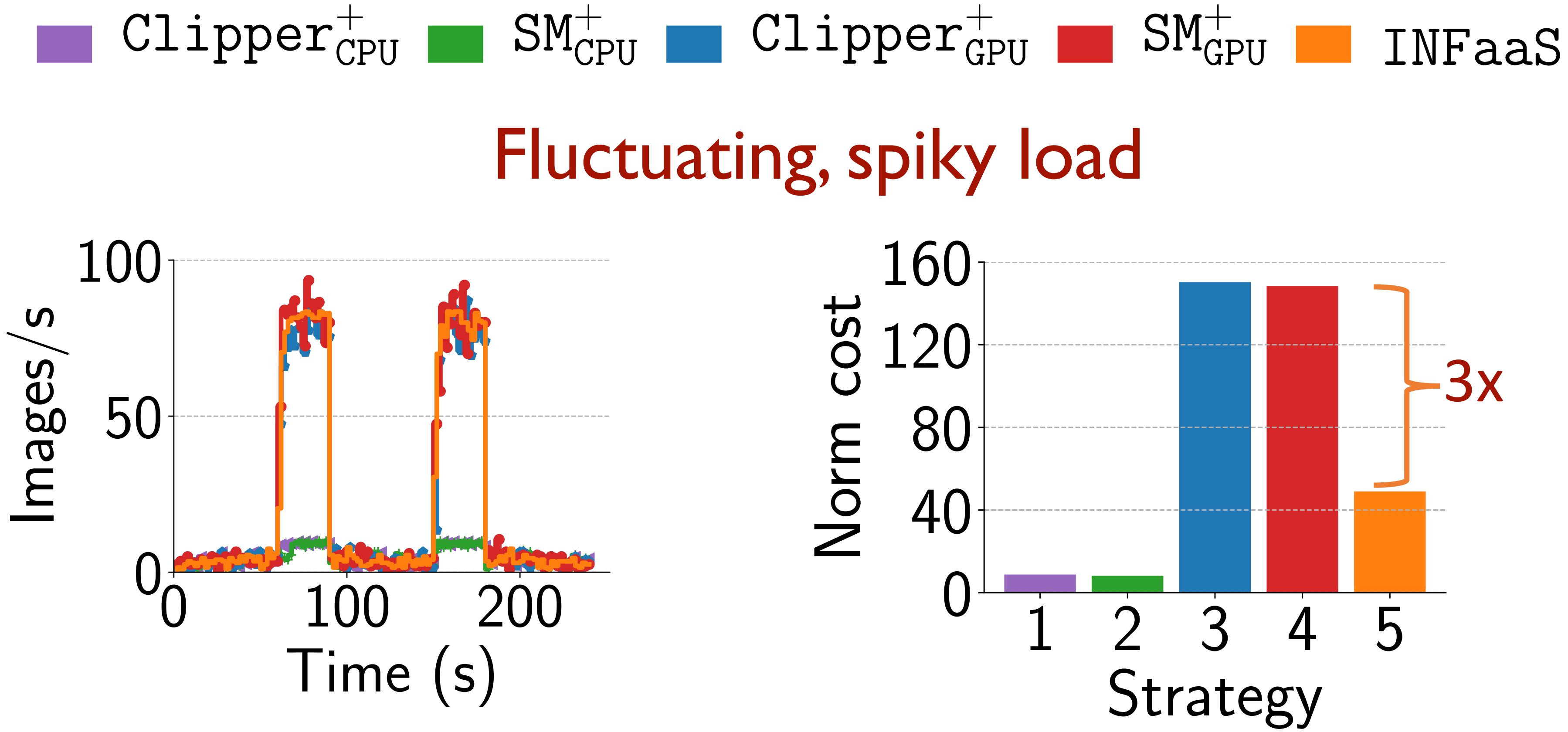
How well does INFaaS scale with load?



INFaaS achieved load while minimizing cost

How well does INFaaS scale with load?

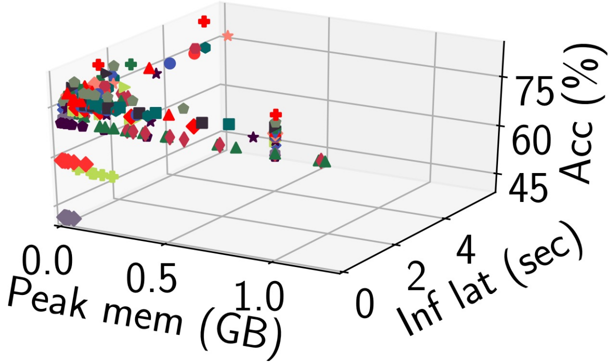
Fluctuating, spiky load



INFaaS reduced cost by 3x by leveraging CPU/Inferentia variant;

if limited to CPU/GPU variants, still 1.7x cheaper

Putting it all together

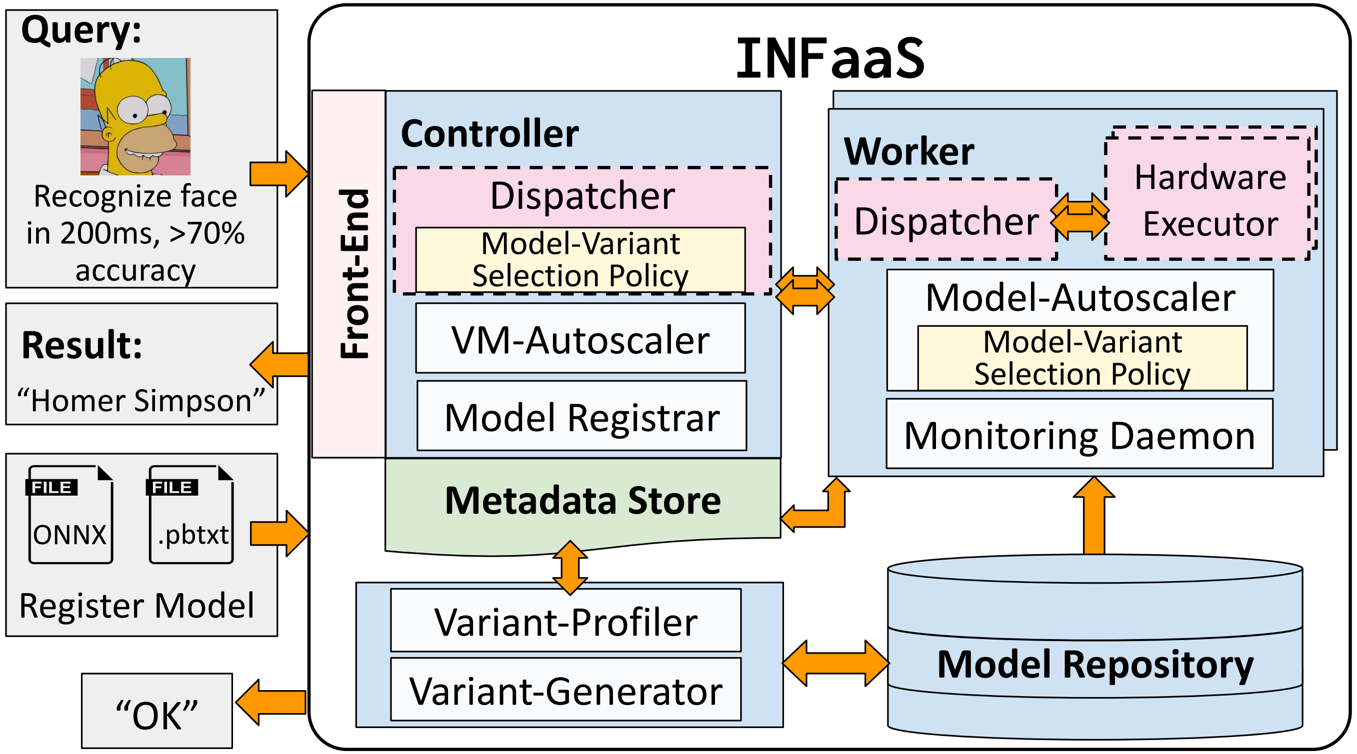


175 model-variants

- Real workload: Twitter trace (diurnal pattern + spikes)
- Compared to CLIPPER+ and SM+:
  - 1.1x, 1.3x higher throughput versus CLIPPER+, SM+
  - 1.6x, 2.5x fewer SLO violations compared to CLIPPER+, SM+
  - 1.23x lower cost by leveraging CPU, GPU, Inferentia machines

INFaaS achieved high performance, better resource utilization, lower SLO violations, and reduced cost

Conclusion



https://stanford-mast.github.io/INFaaS/

Contact us:

{faromero,qianl15,neerajay,kozyraki}@stanford.edu