Swayam

Distributed Autoscaling for Machine Learning as a Service

Arpan Gujarati,
Björn B. Brandenburg

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

Sameh Elnikety,
Yuxiong He

Microsoft Research

Kathryn S. McKinley

Google

Machine Learning as a Service (MLaaS)

Microsoft Azure Machine Learning

powered by aws Amazon Machine Learning

IBM® Data Science & Machine Learning

Google Cloud AI

Machine Learning as a Service (MLaaS)

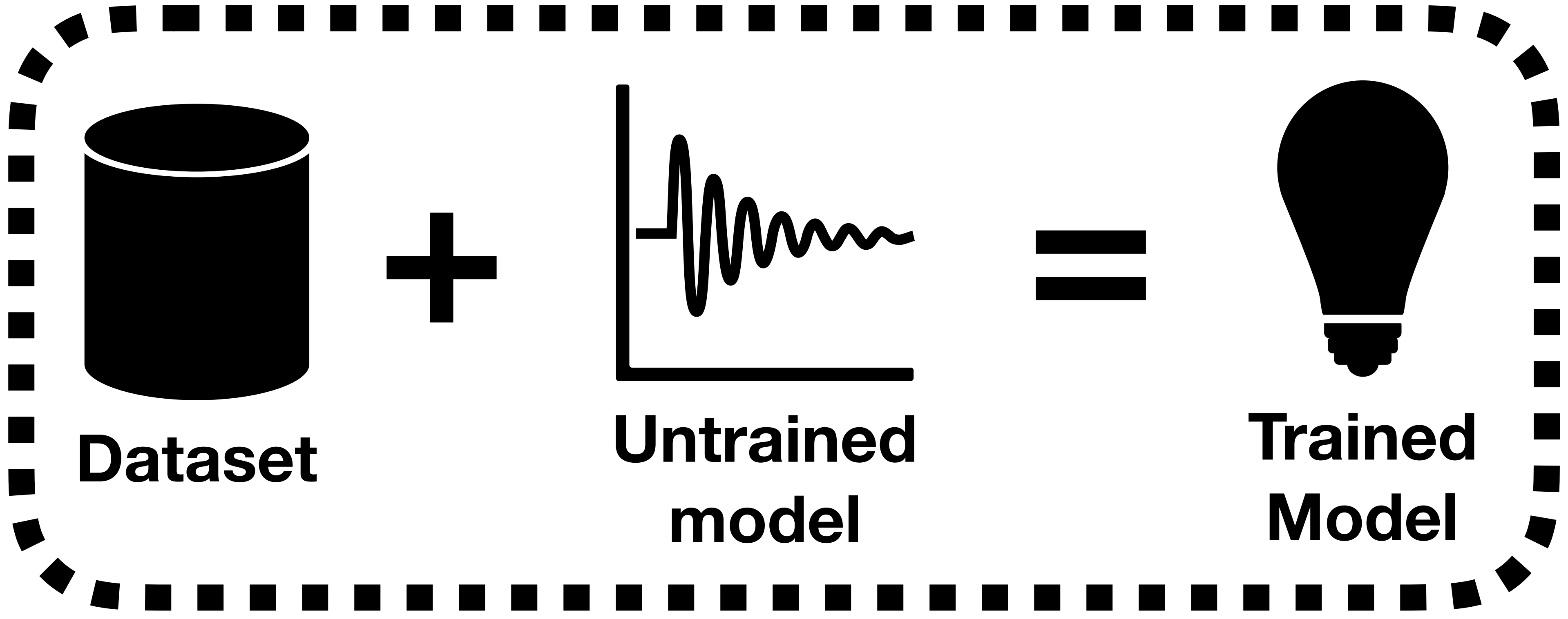
Microsoft Azure Machine Learning

powered by aws Amazon Machine Learning

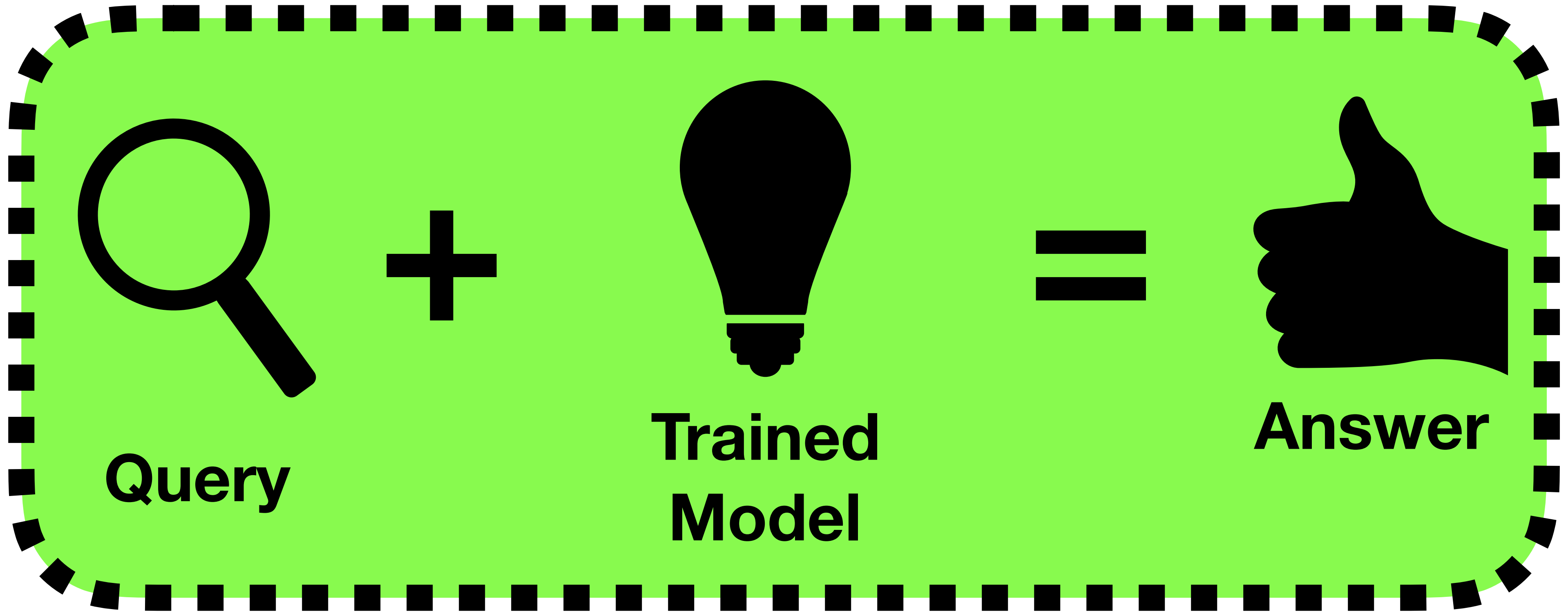
IBM® Data Science & Machine Learning

Google Cloud AI

1. Training



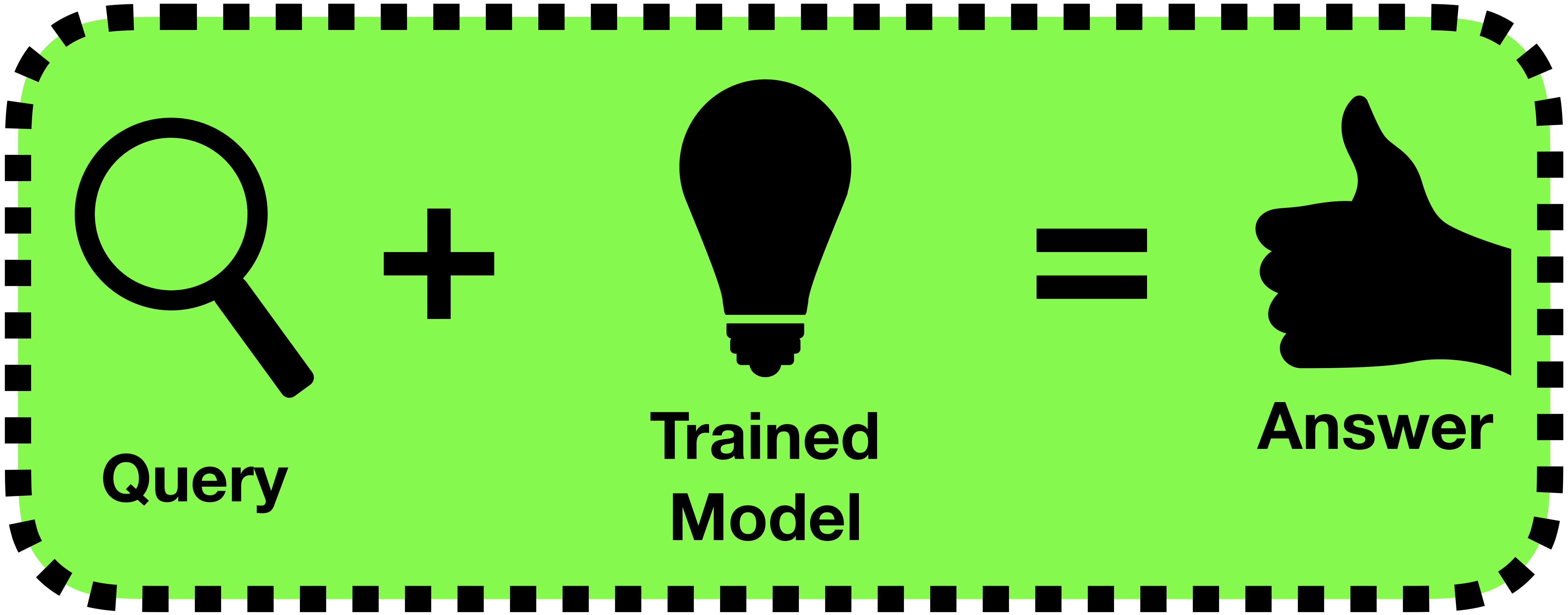
2. Prediction



Machine Learning as a Service (MLaaS)

This work

2. Prediction



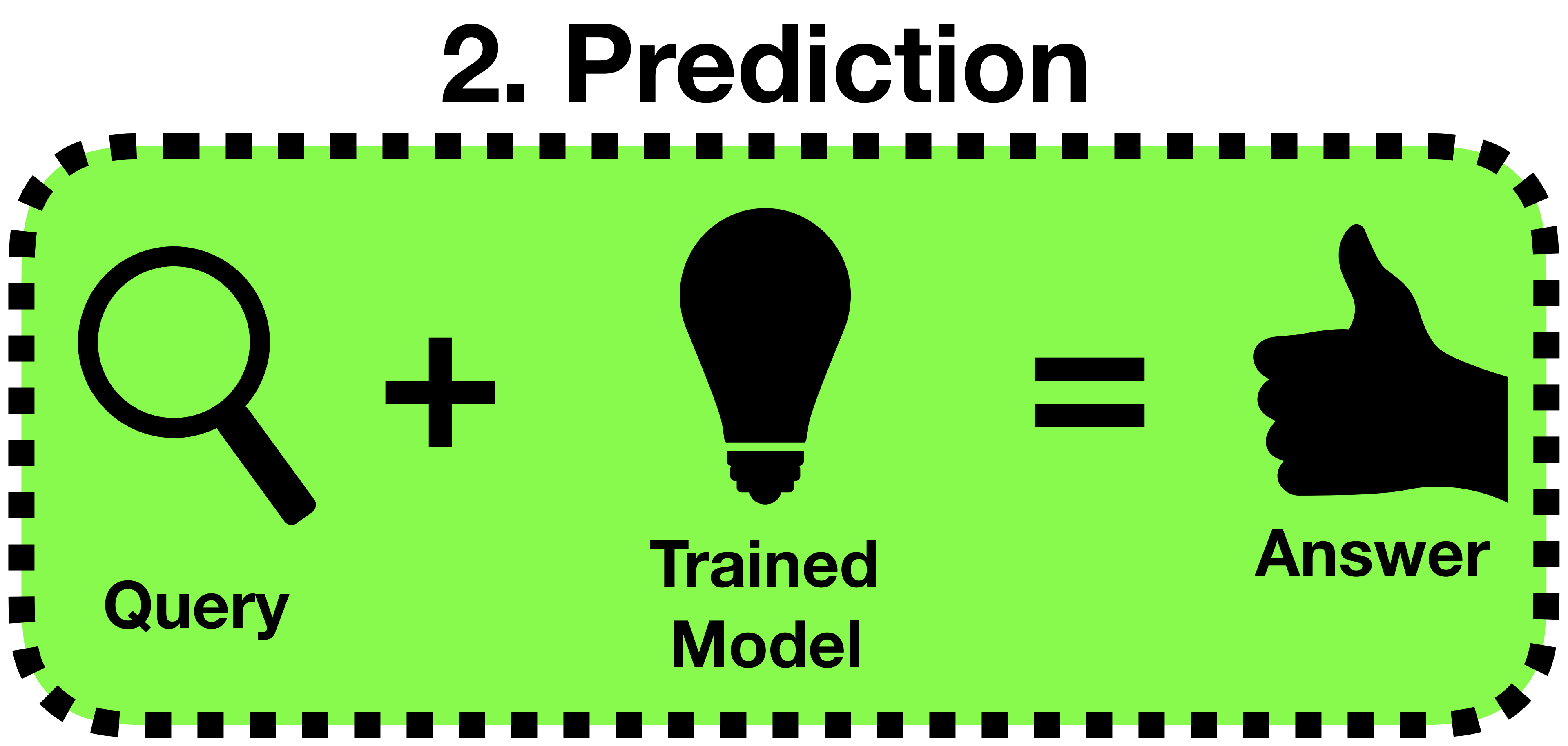
*Models are **already trained** and available for prediction*

Swayam

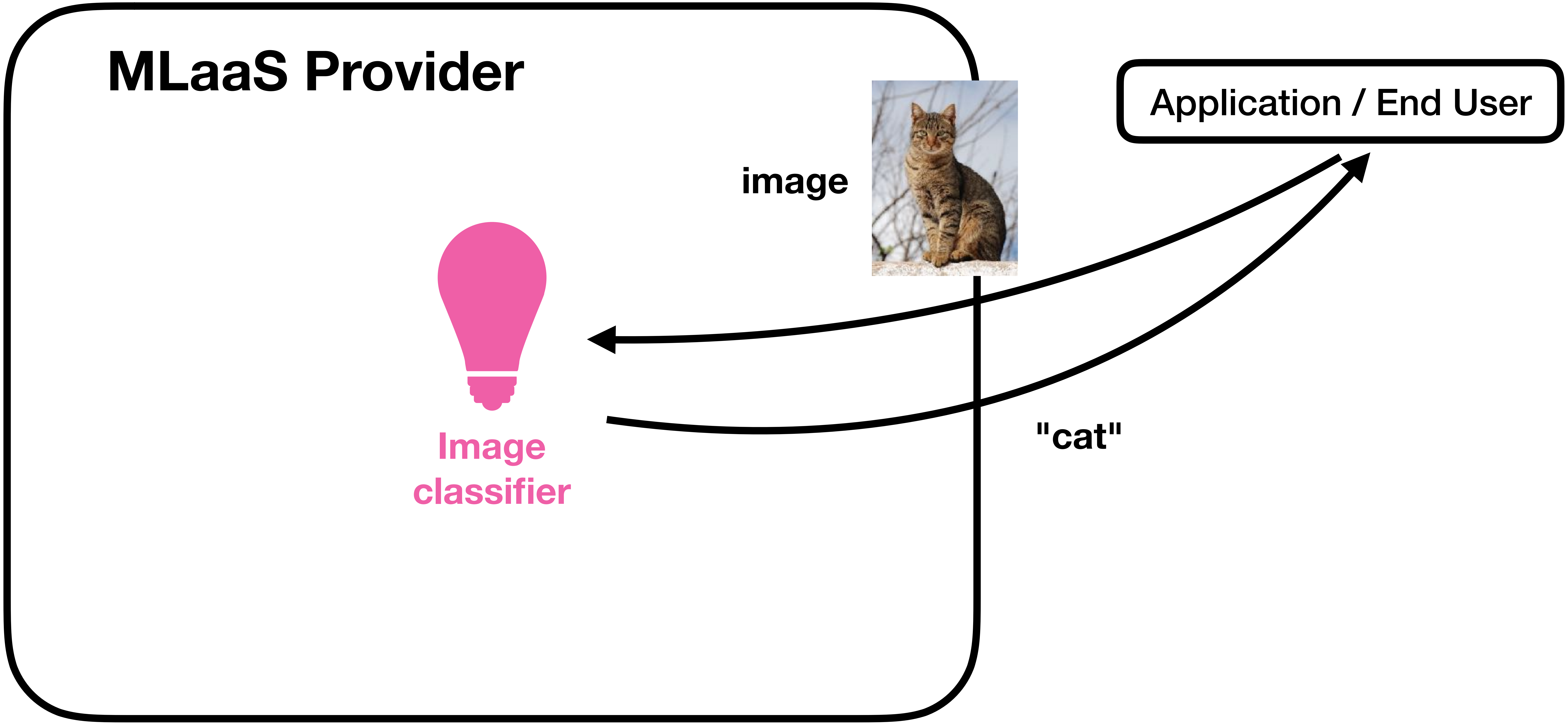
Distributed autoscaling

of the compute resources
needed for **prediction serving**

inside the MLaaS infrastructure



Prediction serving (application perspective)

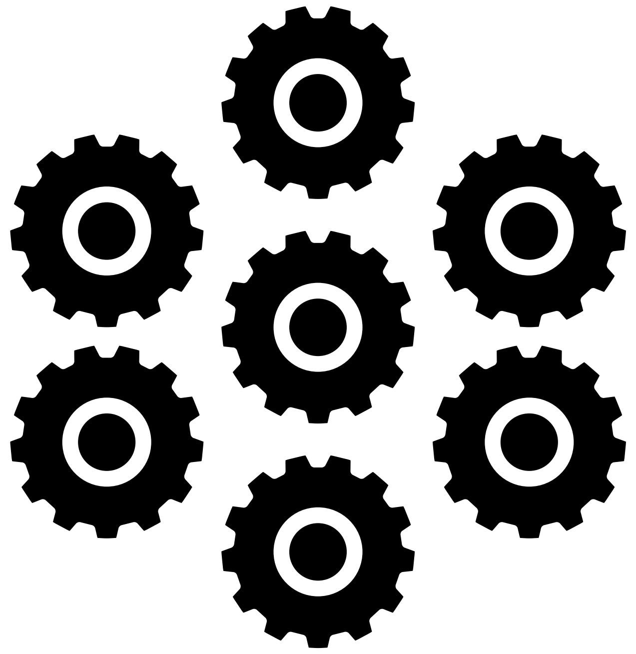


Prediction serving (provider perspective)

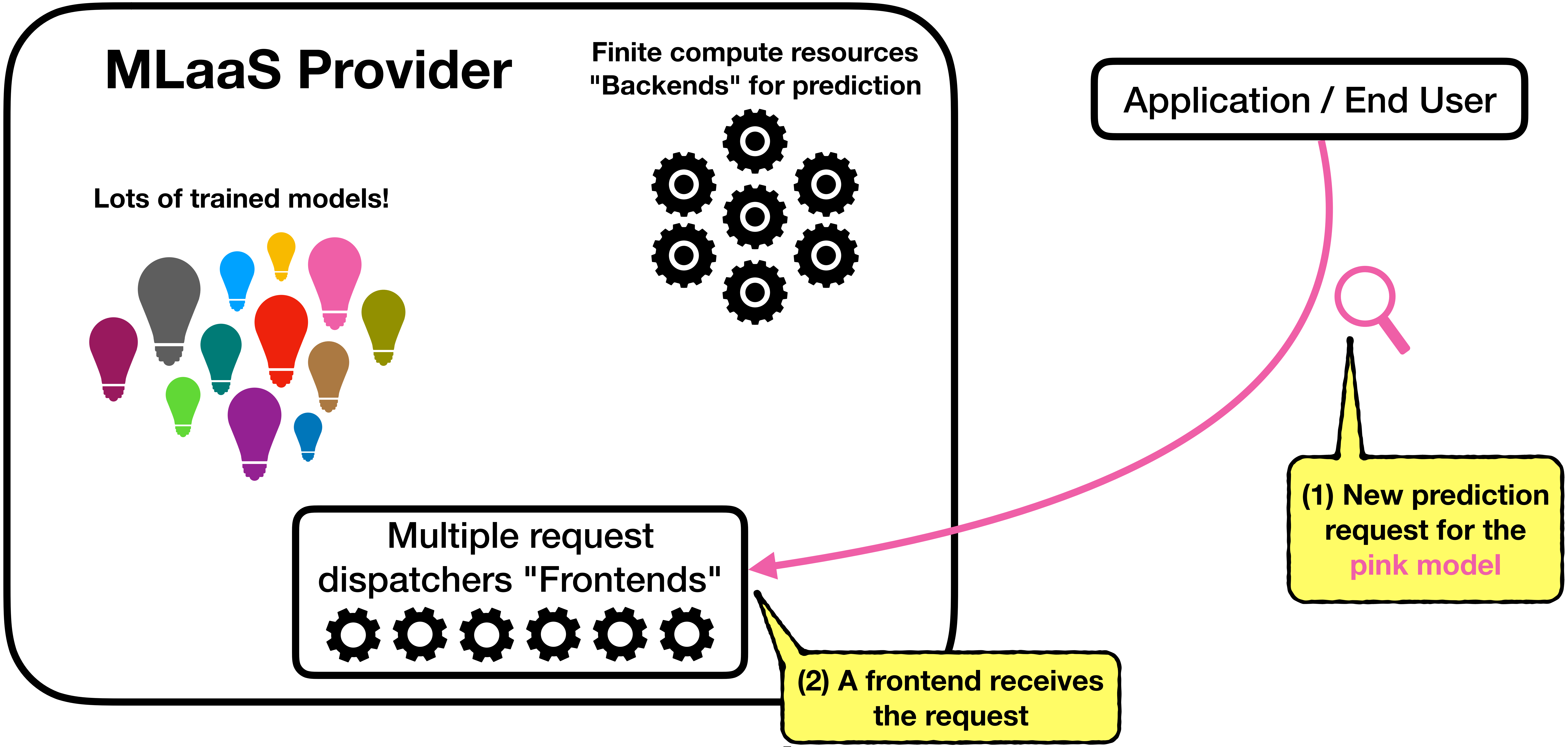
MLaaS Provider

Lots of trained models!

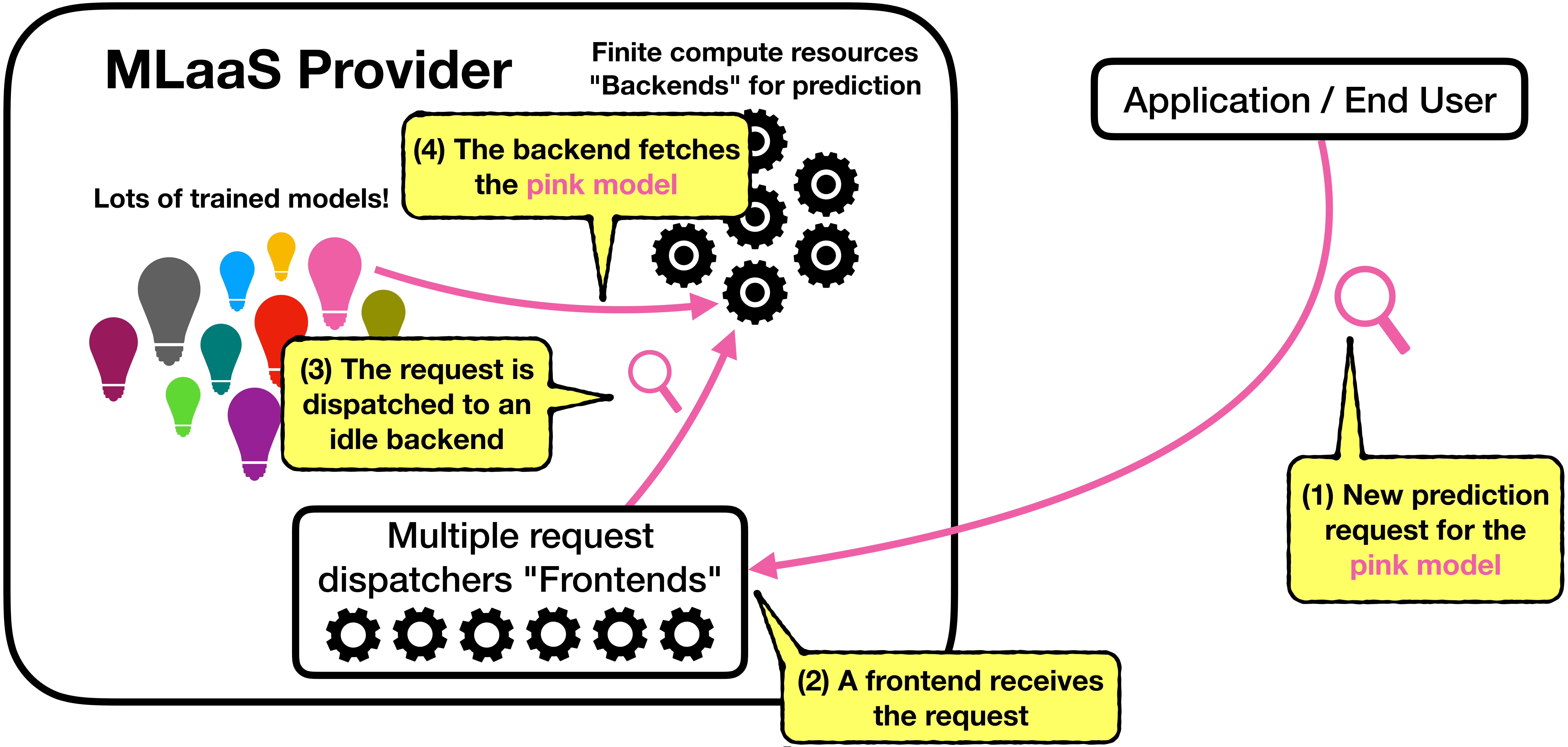
Finite compute resources
"Backends" for prediction



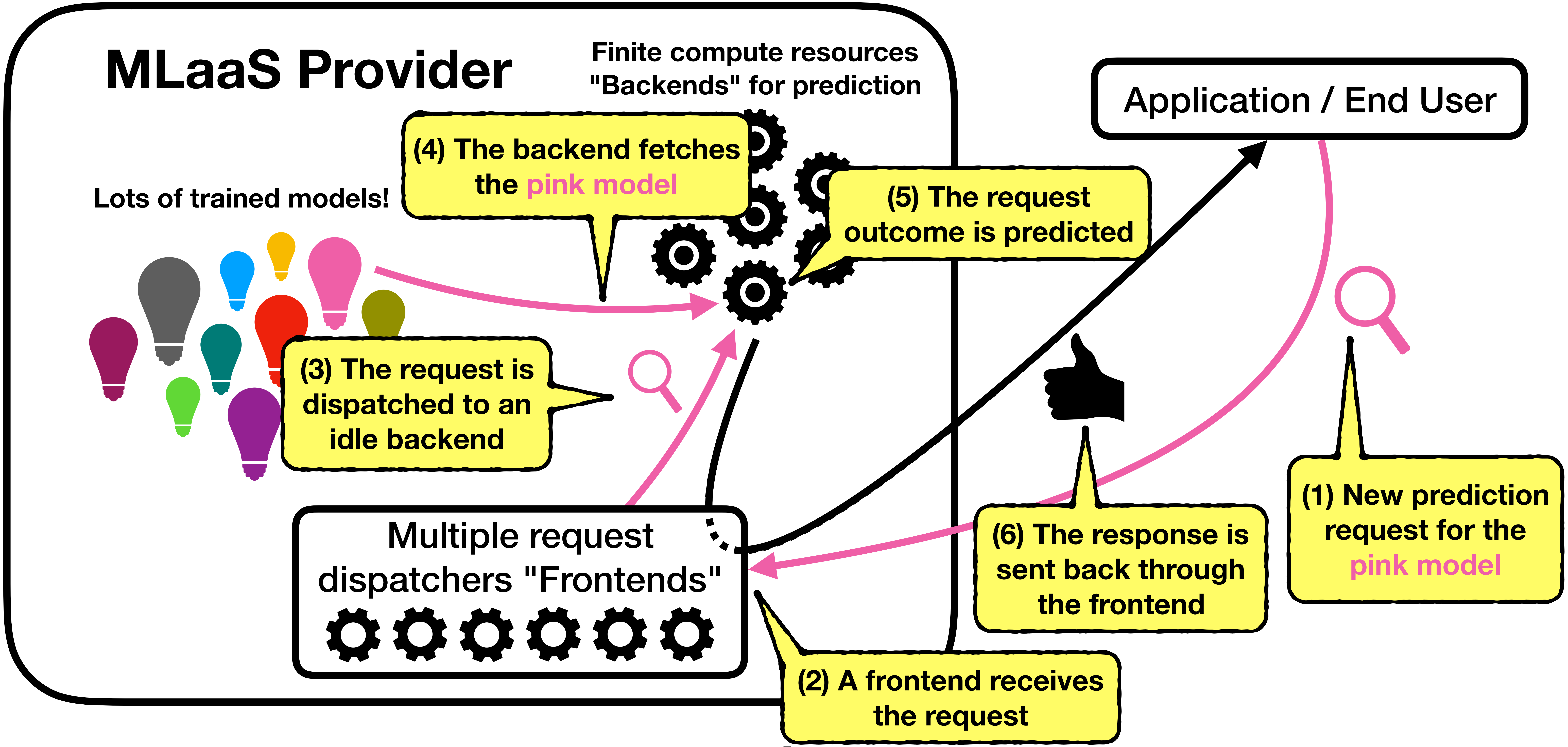
Prediction serving (provider perspective)



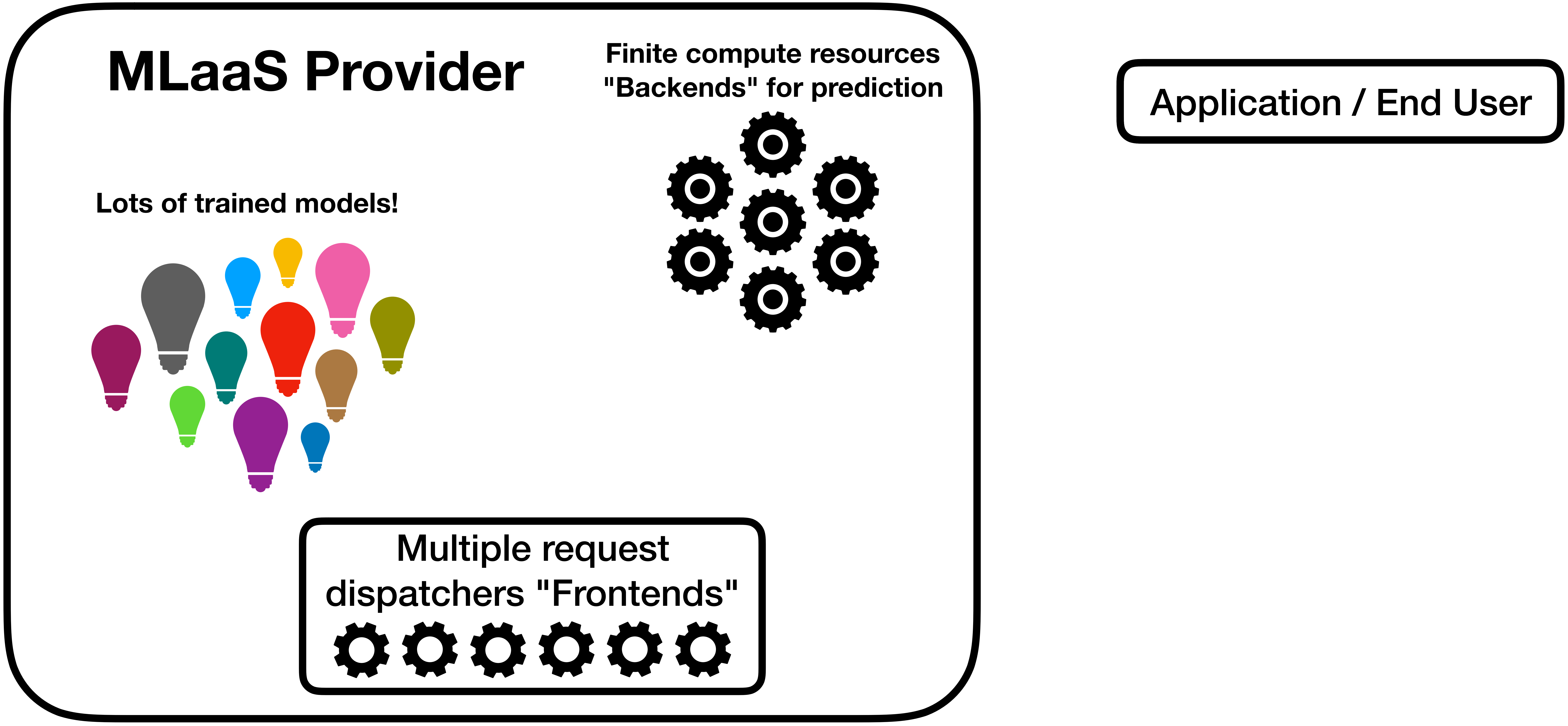
Prediction serving (provider perspective)



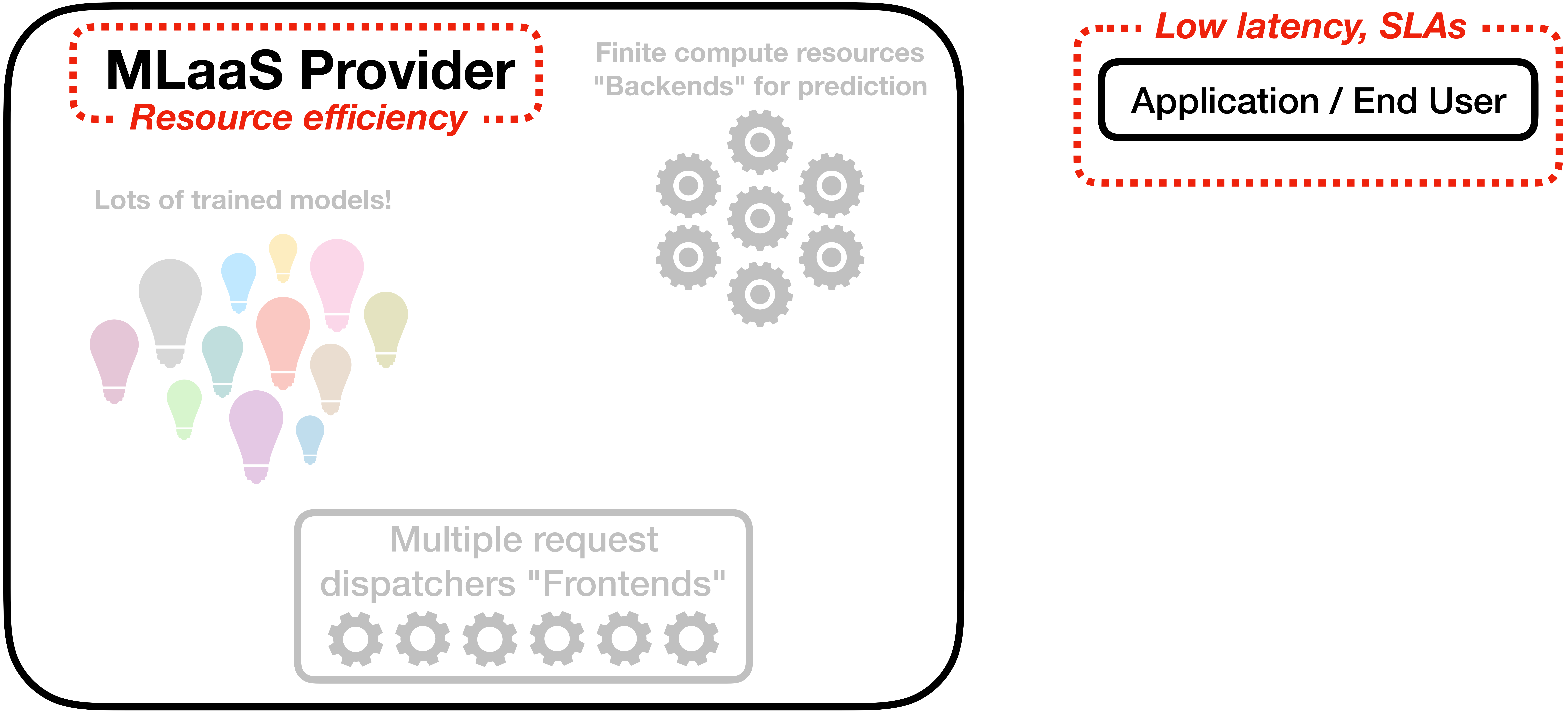
Prediction serving (provider perspective)



Prediction serving (objectives)

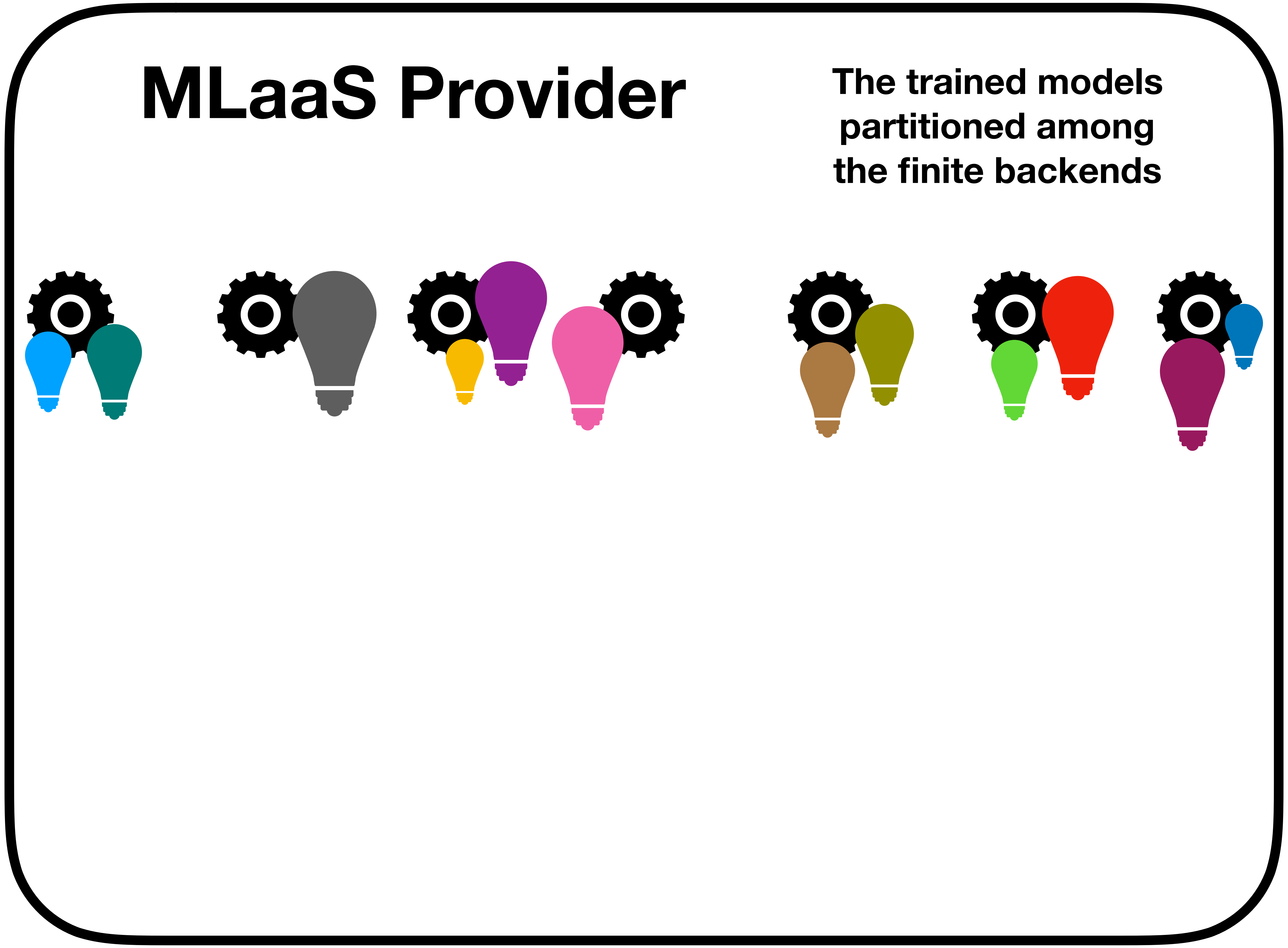


Prediction serving (objectives)

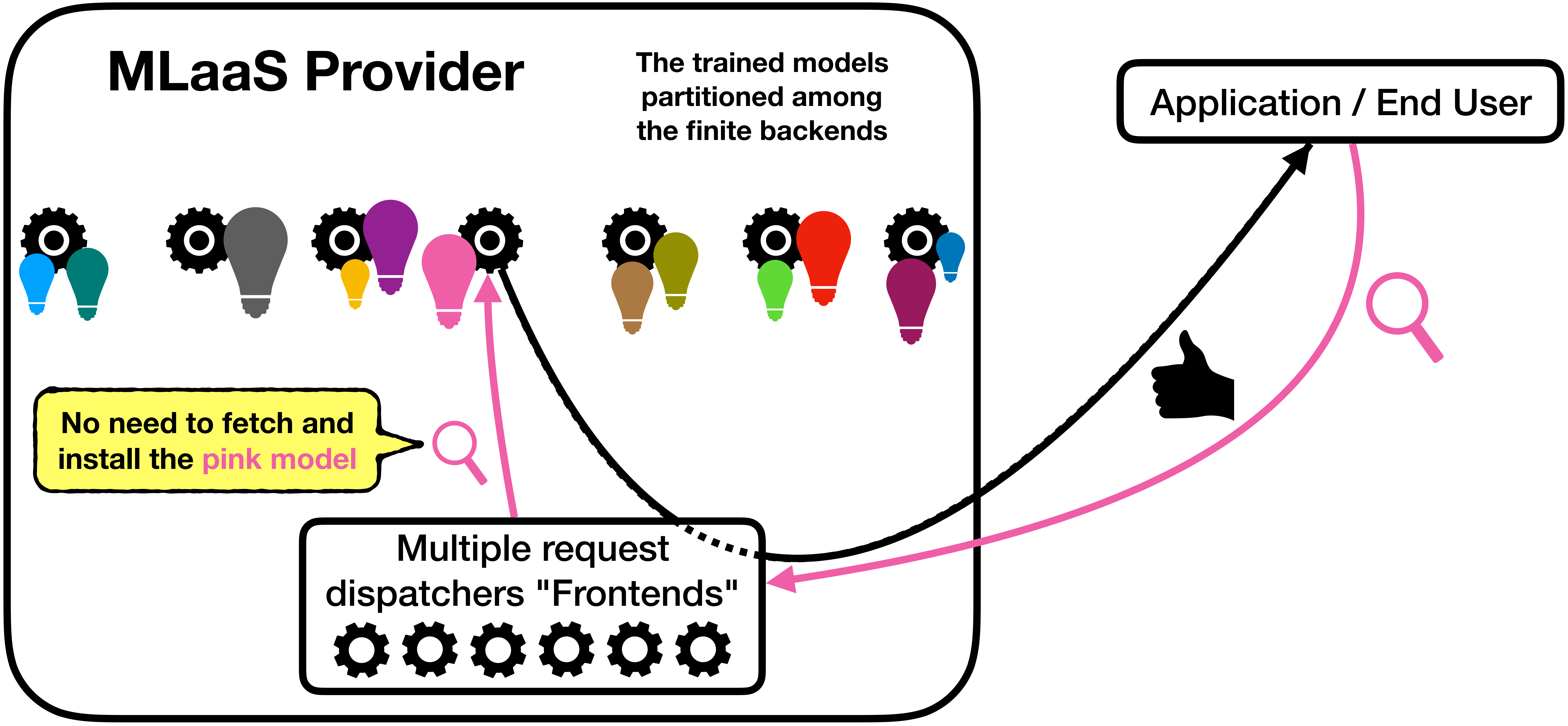


Static partitioning of trained models

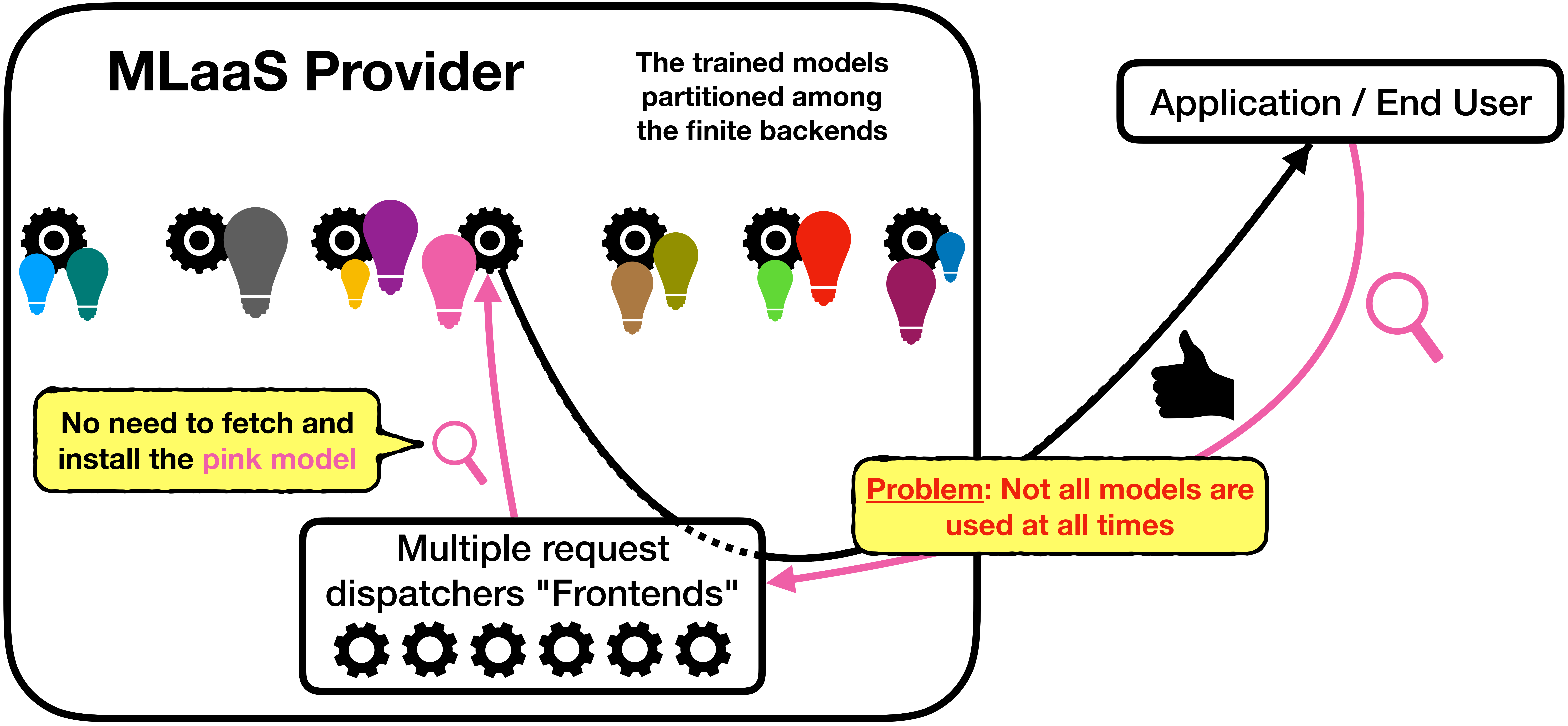
Static partitioning of trained models



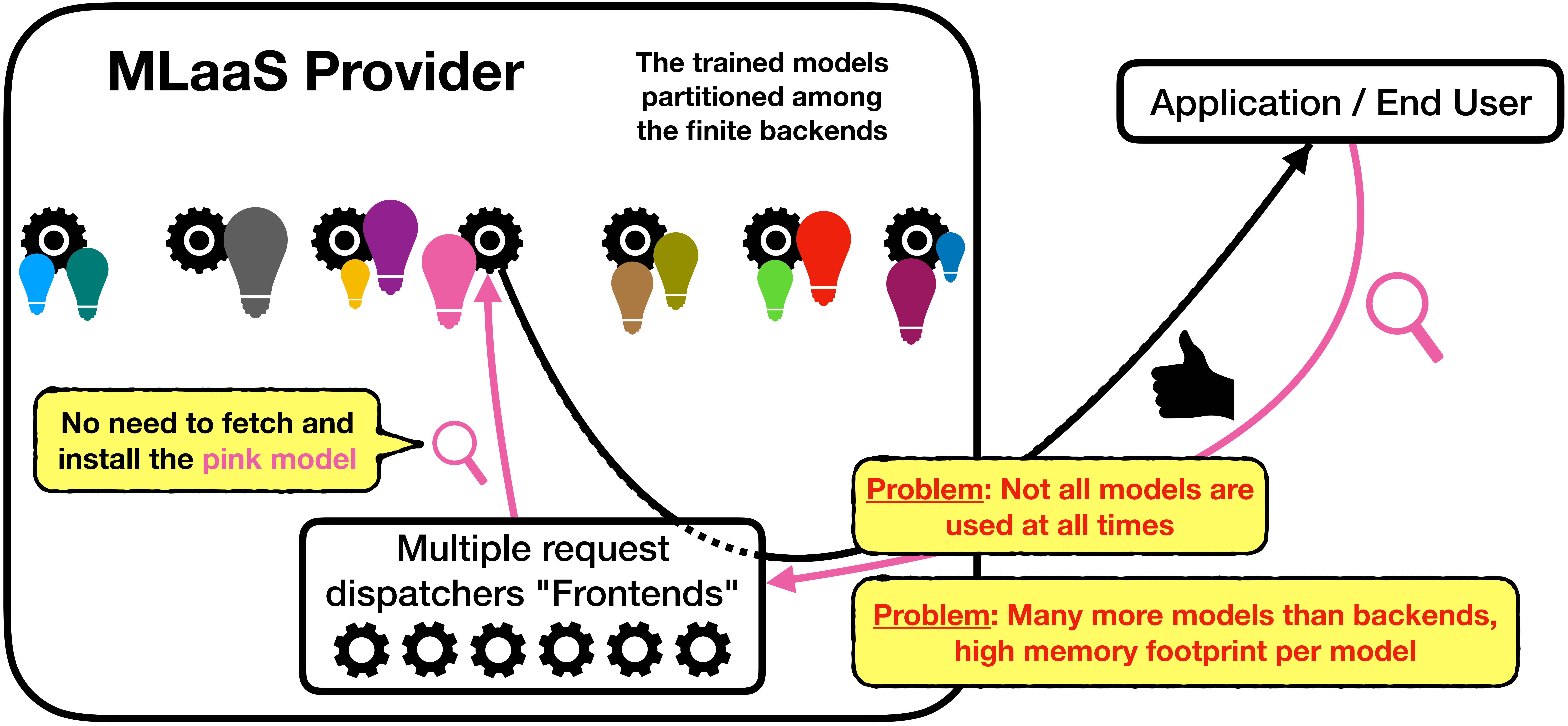
Static partitioning of trained models



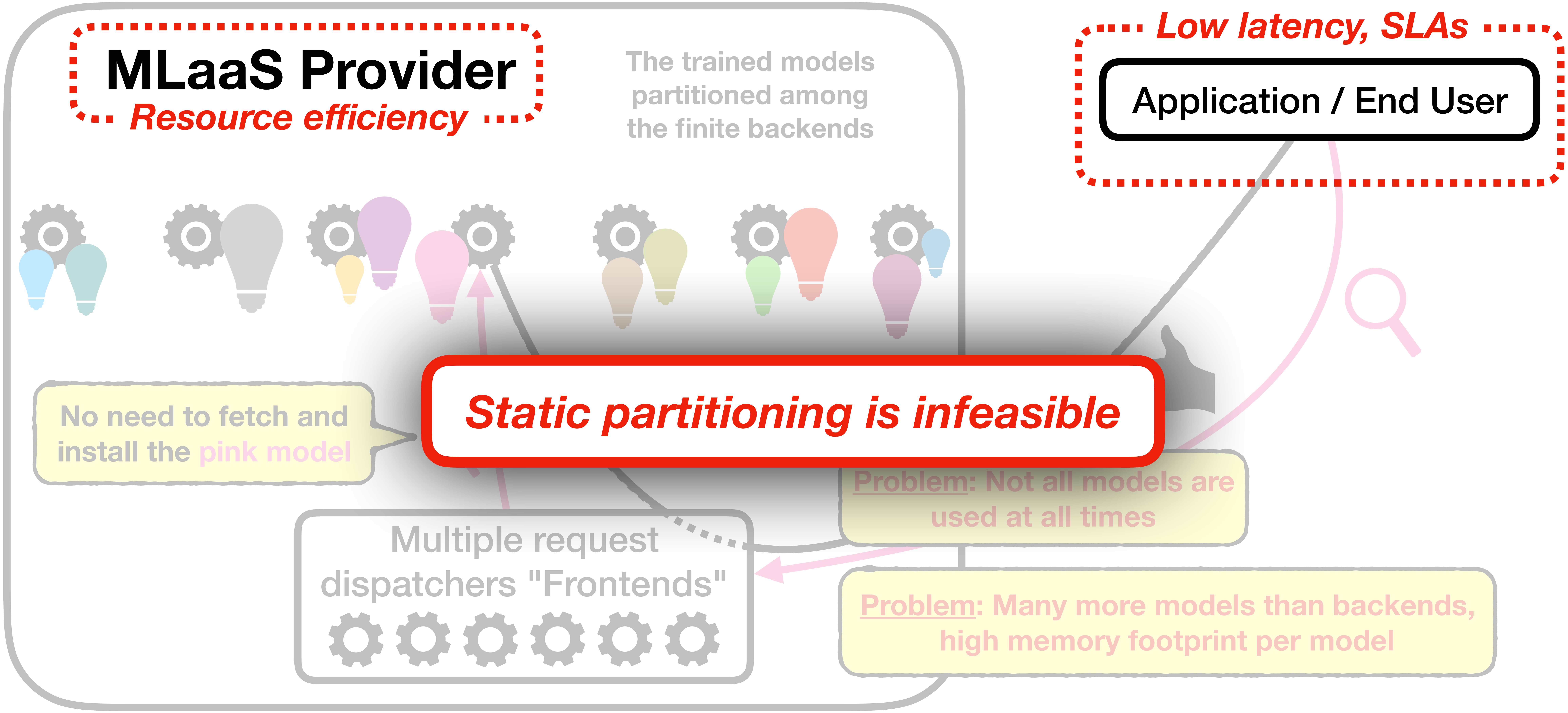
Static partitioning of trained models



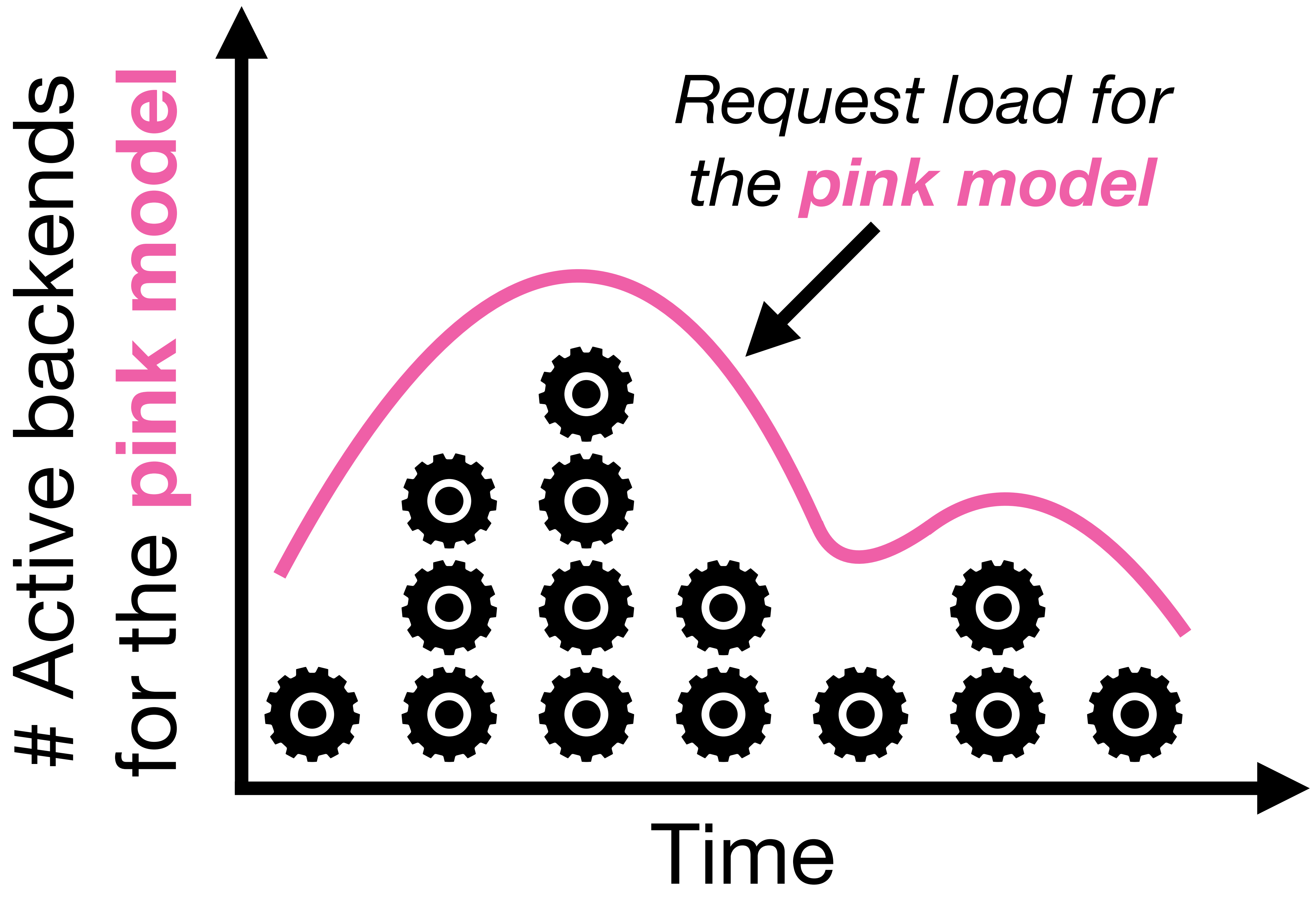
Static partitioning of trained models



Static partitioning of trained models

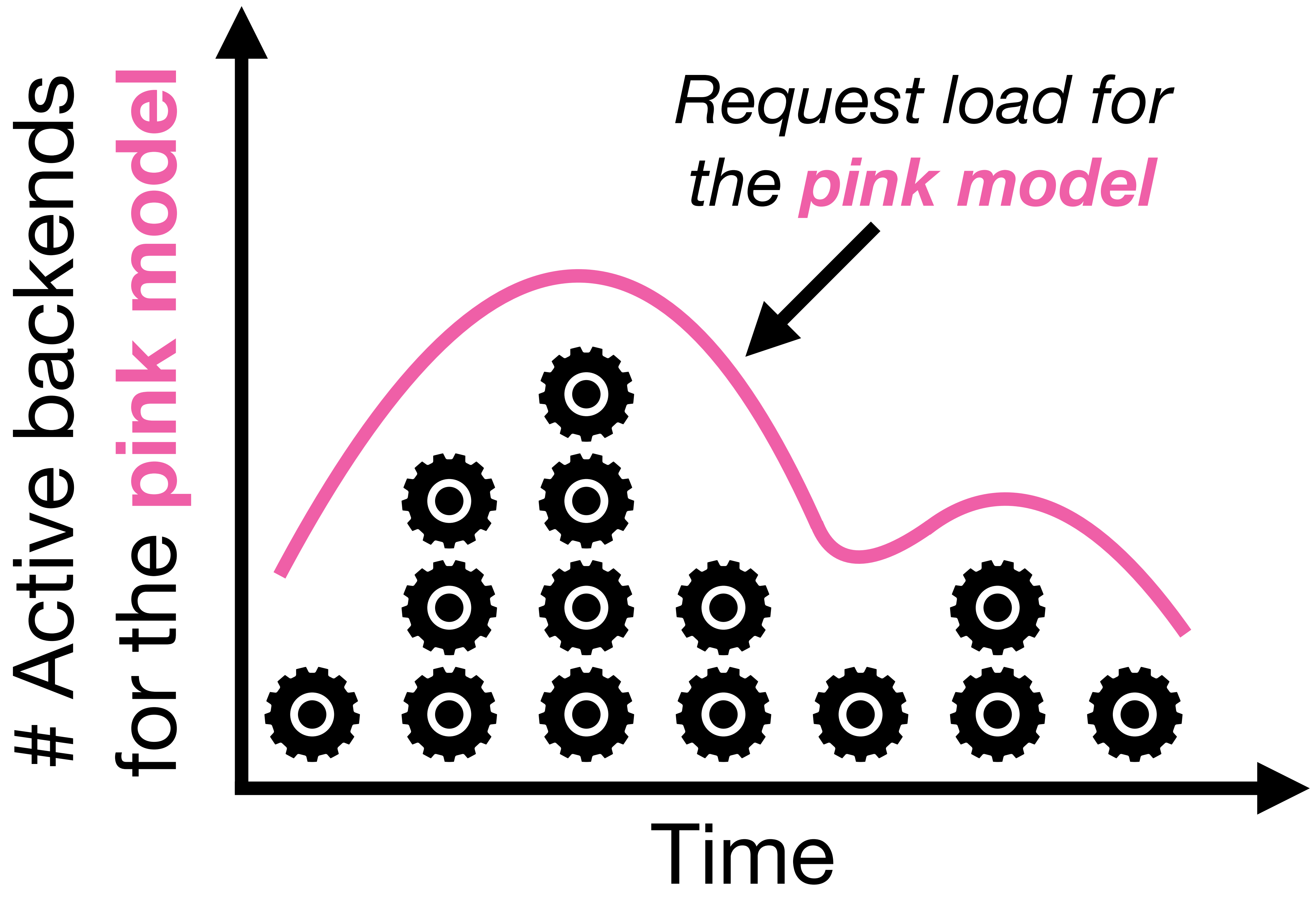


Classical approach: autoscaling



The number of active backends are automatically scaled up or down based on load

Classical approach: autoscaling



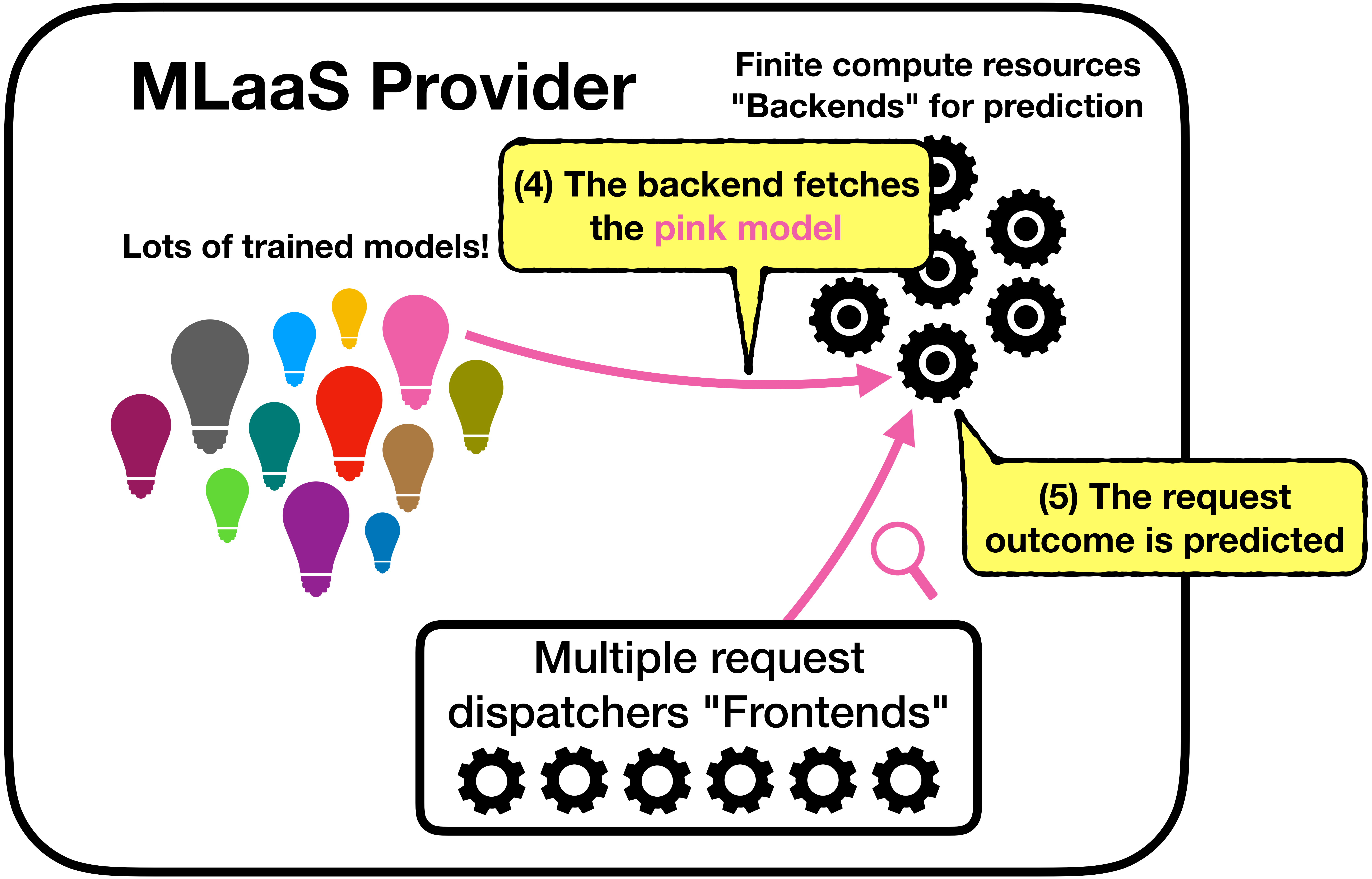
The number of active backends are automatically scaled up or down based on load

With **ideal autoscaling ...**

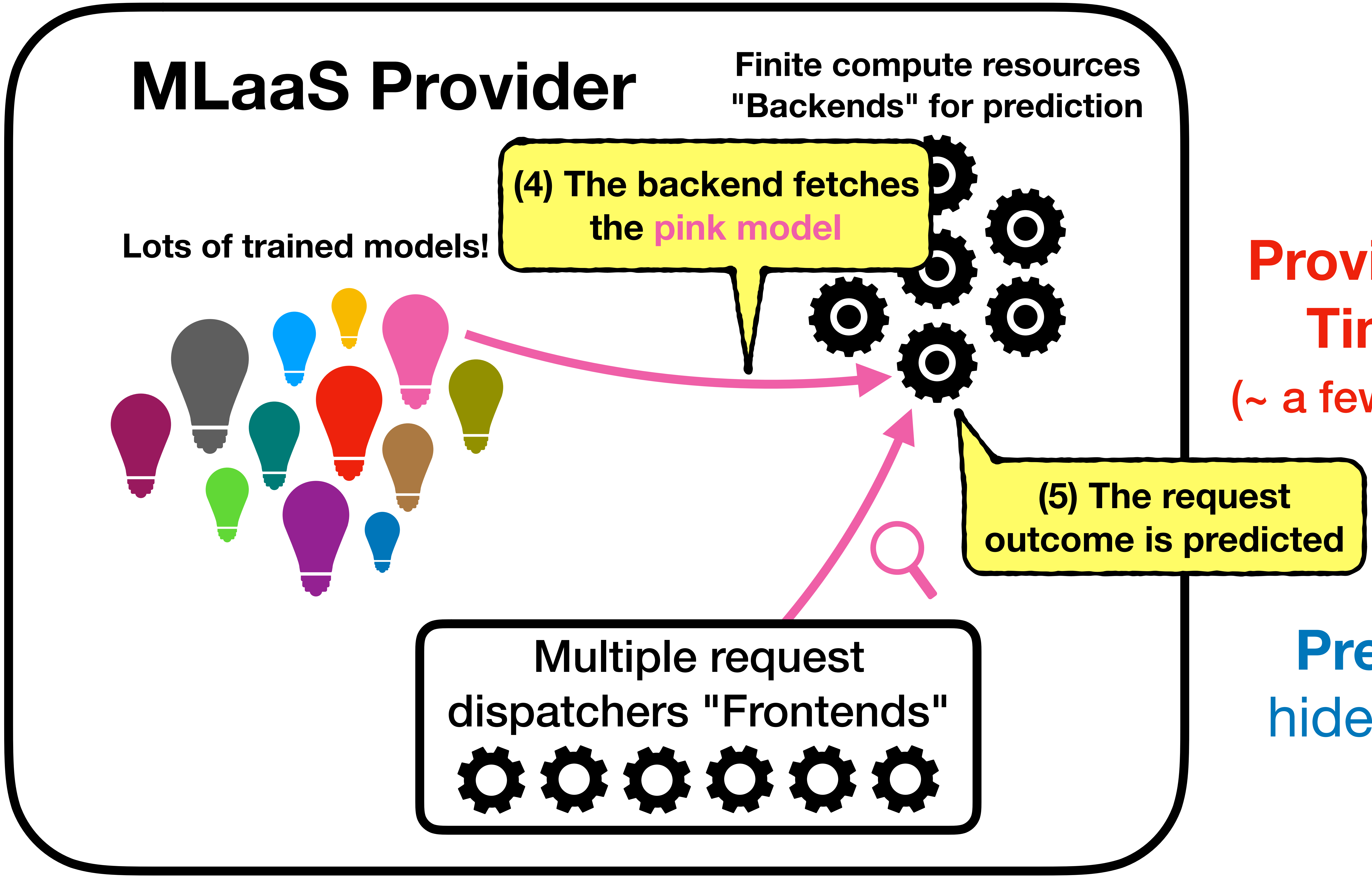
- ✔ Enough backends to guarantee **low latency**
- ✔ # Active backends over time is **minimized for resource efficiency**

Autoscaling for MLaaS is challenging [1/3]

Autoscaling for MLaaS is challenging [1/3]



Autoscaling for MLaaS is challenging [1/3]



Challenge

Provisioning Time (4) >> Execution Time (5)

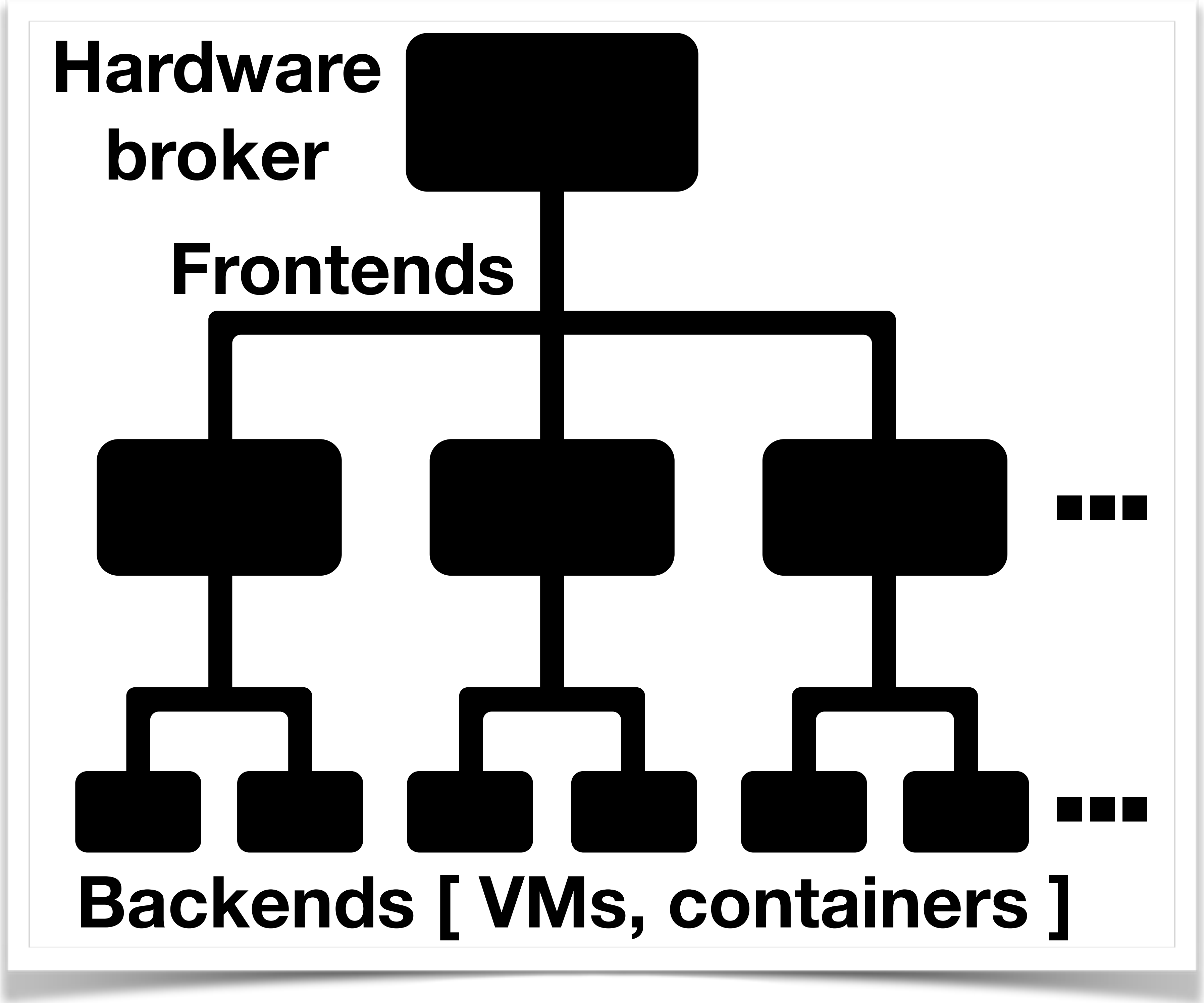
(~ a few seconds) (~ 10ms to 500ms)

Requirement

Predictive autoscaling to hide the provisioning latency

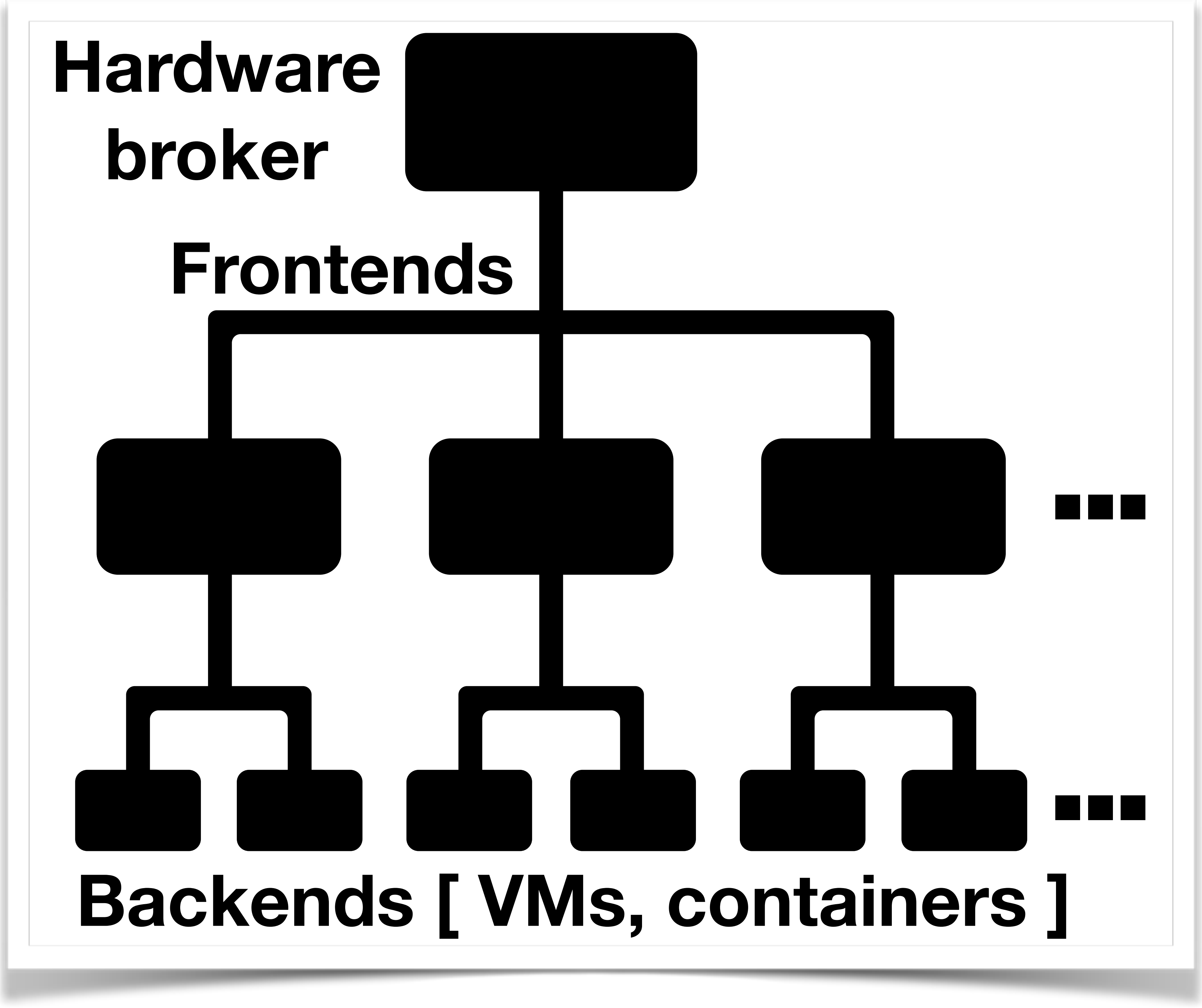
Autoscaling for MLaaS is challenging [2/3]

MLaaS architecture is
large-scale, multi-tiered



Autoscaling for MLaaS is challenging [2/3]

MLaaS architecture is
large-scale, multi-tiered



Challenge

Multiple frontends with **partial information** about the workload

Requirement

Fast, coordination-free, globally-consistent autoscaling decisions on the frontends

Autoscaling for MLaaS is challenging [3/3]

Strict, model-specific SLAs
on response times

"99% of requests must complete under 500ms"

"99.9% of requests must complete under 1s"

"[A] 95% of requests must complete under 850ms"

"[B] Tolerate up to 25% increase in request rates without violating [A]"

Autoscaling for MLaaS is challenging [3/3]

Strict, model-specific SLAs
on response times

"99% of requests must complete under 500ms"

"99.9% of requests must complete under 1s"

"[A] 95% of requests must complete under 850ms"

"[B] Tolerate up to 25% increase in request rates without violating [A]"

Challenge

No **closed-form solutions** to get response-time distributions for SLA-aware autoscaling

Requirement

Accurate **waiting-time** and **execution-time** distributions

Swayam: model-driven distributed autoscaling

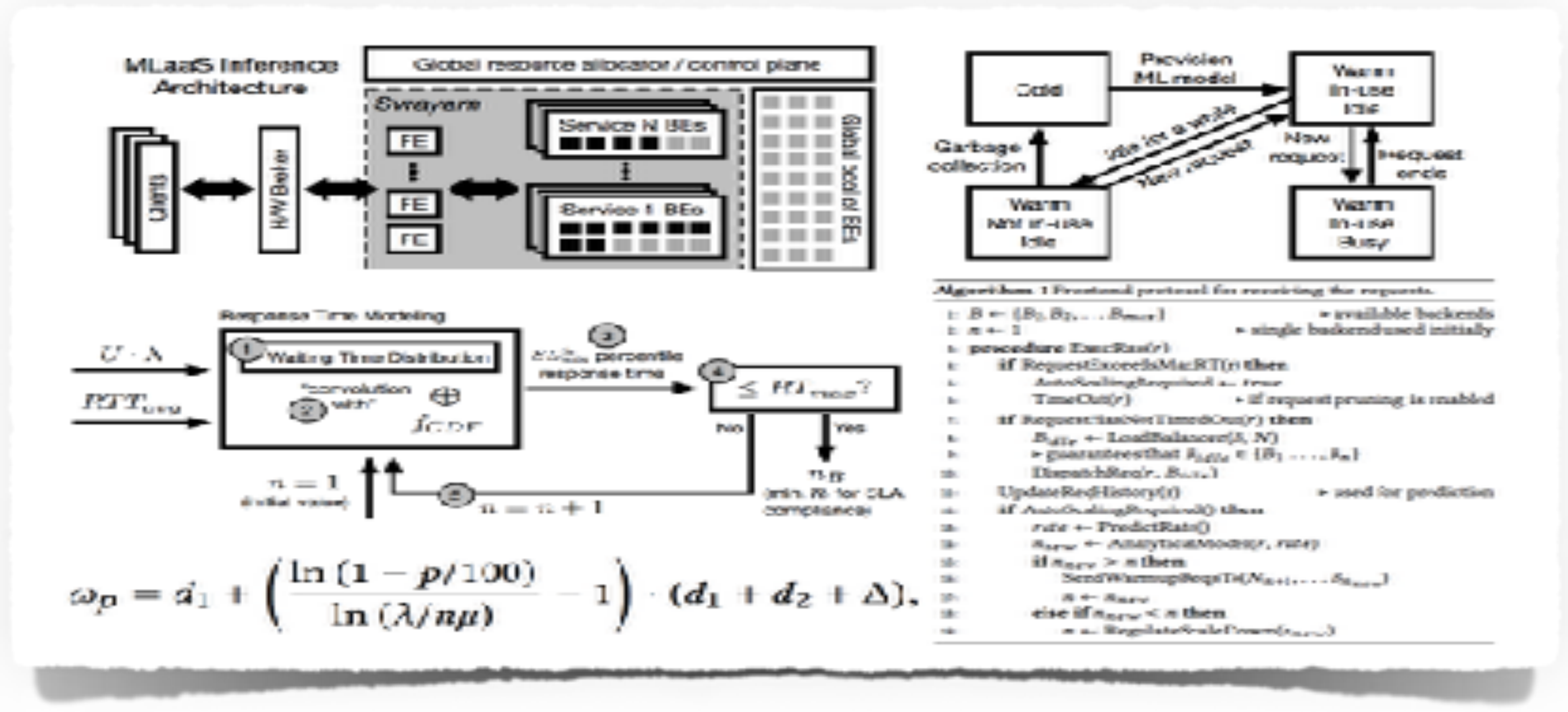
Challenges

Provisioning Time (4) **>>** **Execution Time (5)**

(~ a few seconds) (~ 10ms to 500ms)

Multiple frontends with **partial information** about the workload

No **closed-form solutions** to get response-time distributions for SLA-aware autoscaling



We address these **challenges**

by leveraging specific **ML workload characteristics**

and design an **analytical model** for resource estimation

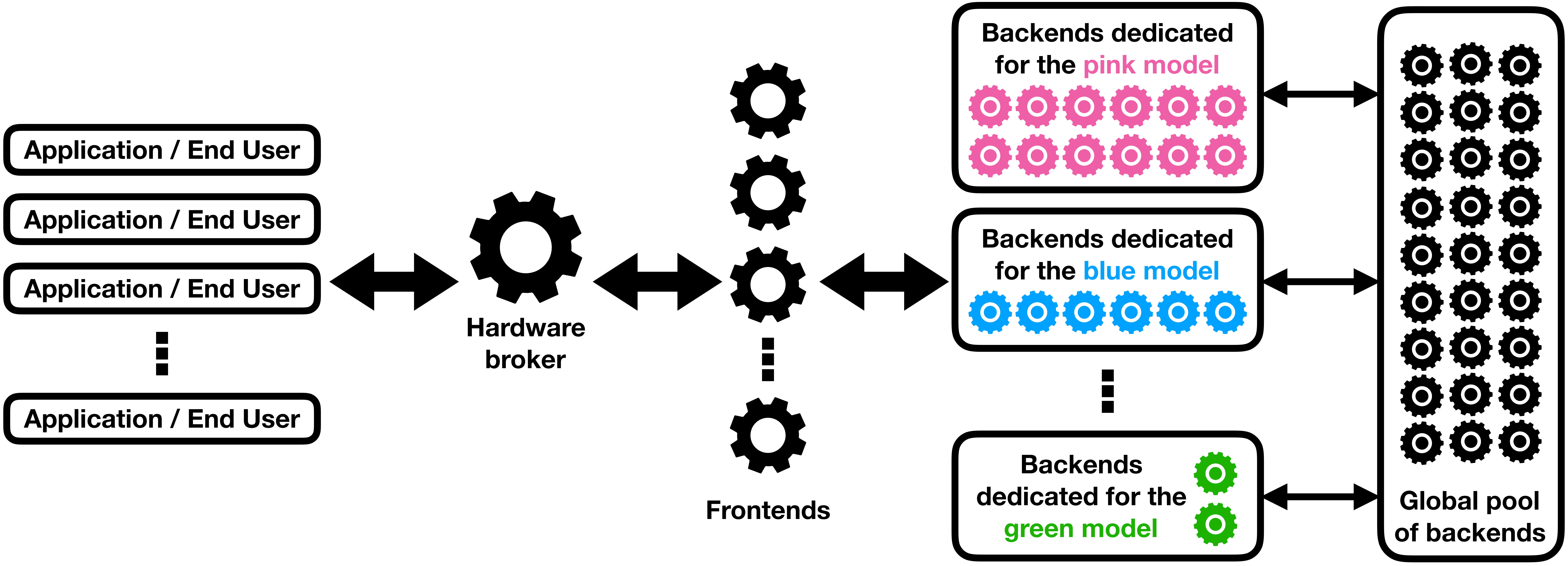
that allows **distributed** and **predictive** autoscaling

Outline

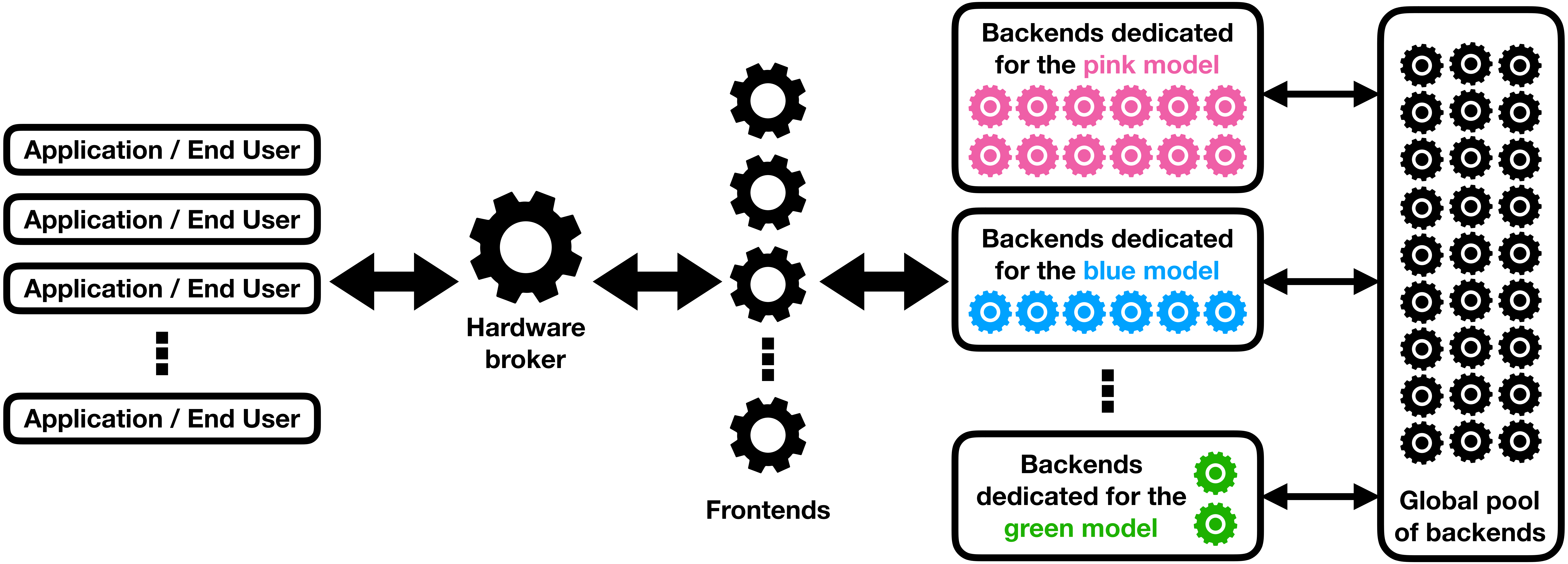
1. **System architecture, key ideas**
2. Analytical model for resource estimation
3. Evaluation results

System architecture

System architecture



System architecture



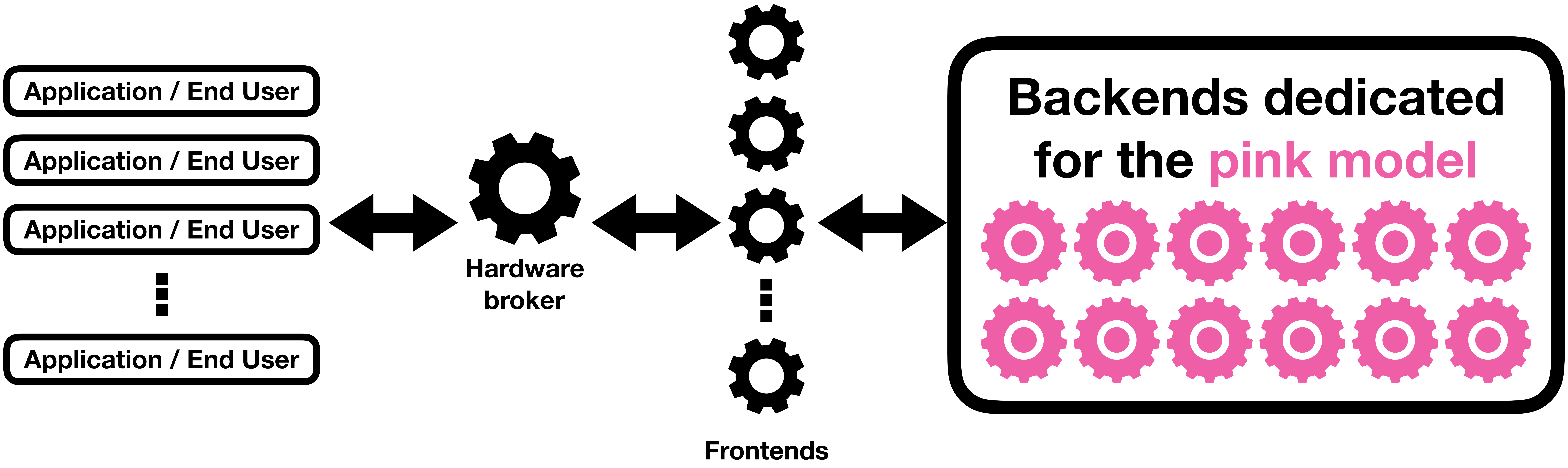
Objective: dedicated set of backends should dynamically scale

1. If load decreases, extra backends go back to the global pool ***(for resource efficiency)***

2. If load increases, new backends are set up in advance ***(for SLA compliance)***

System architecture

Let's focus on the pink model



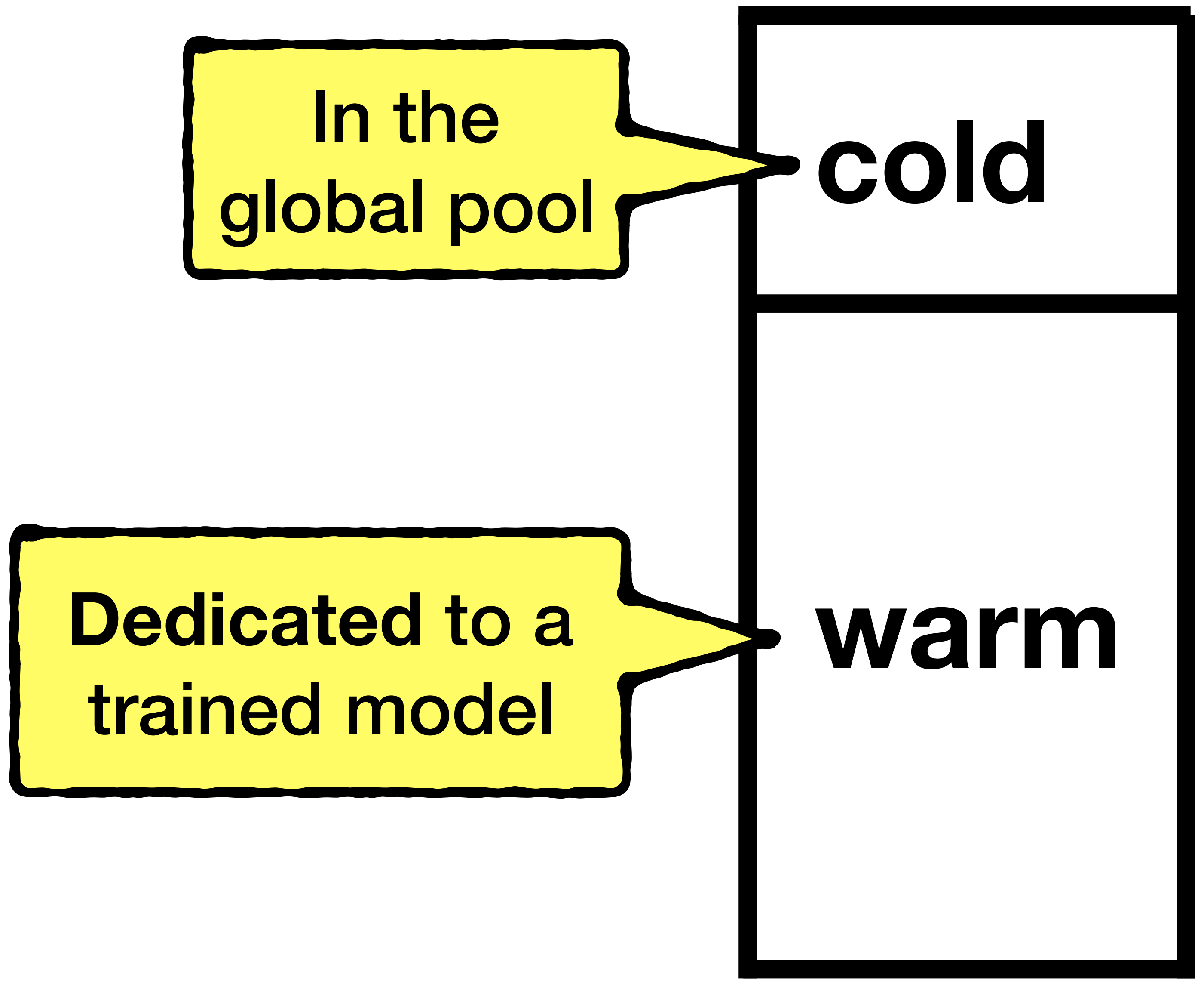
Objective: dedicated set of backends should dynamically scale

1. If load decreases, extra backends go back to the global pool ***(for resource efficiency)***

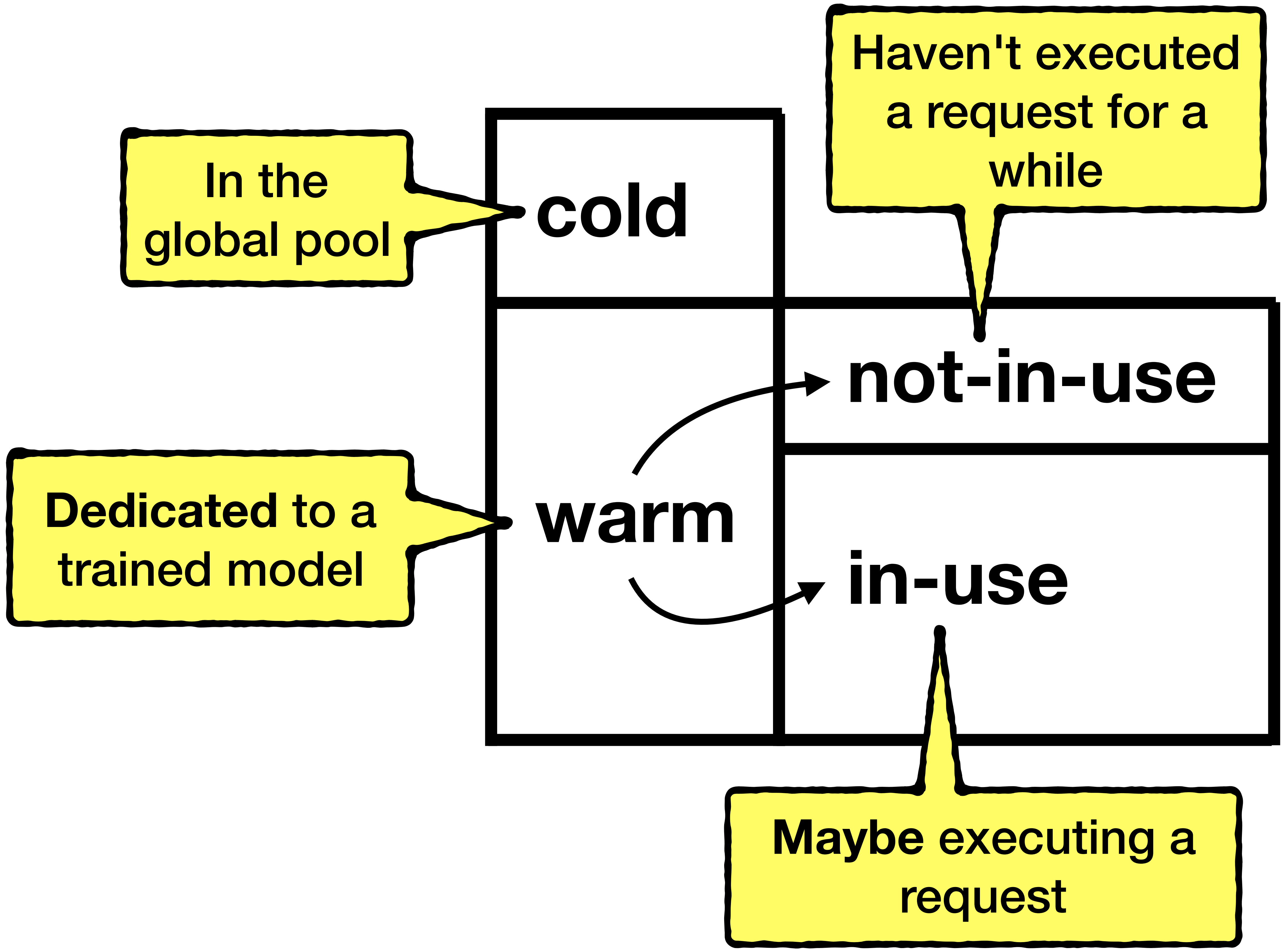
2. If load increases, new backends are set up in advance ***(for SLA compliance)***

Key idea 1: Assign states to each backend

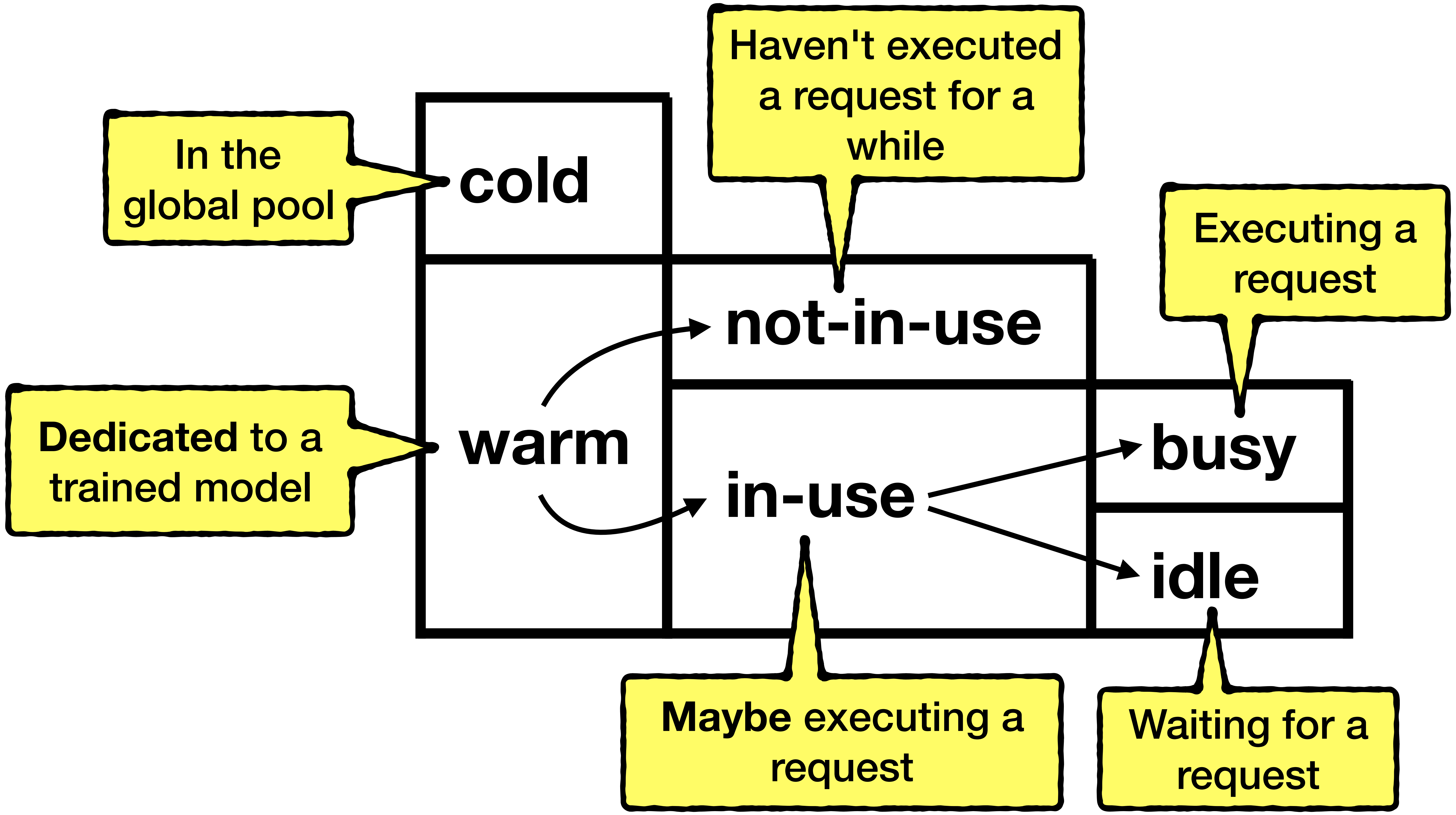
Key idea 1: Assign states to each backend



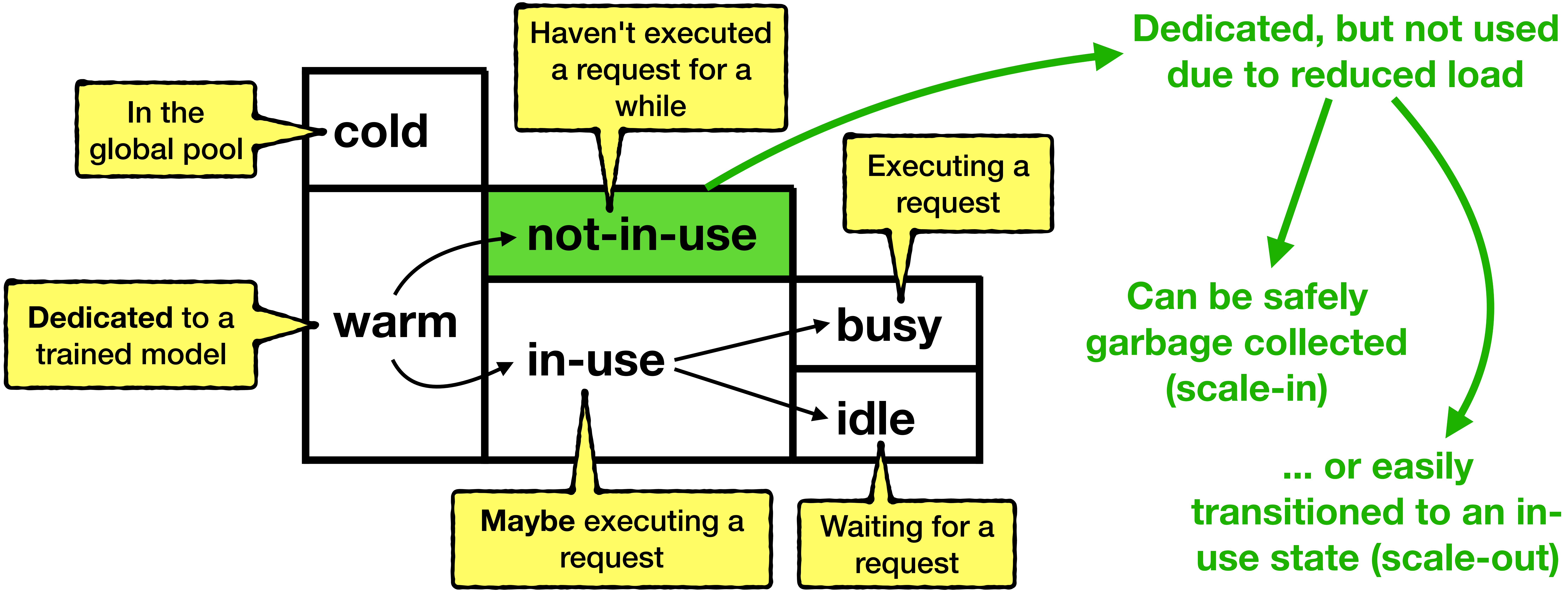
Key idea 1: Assign states to each backend



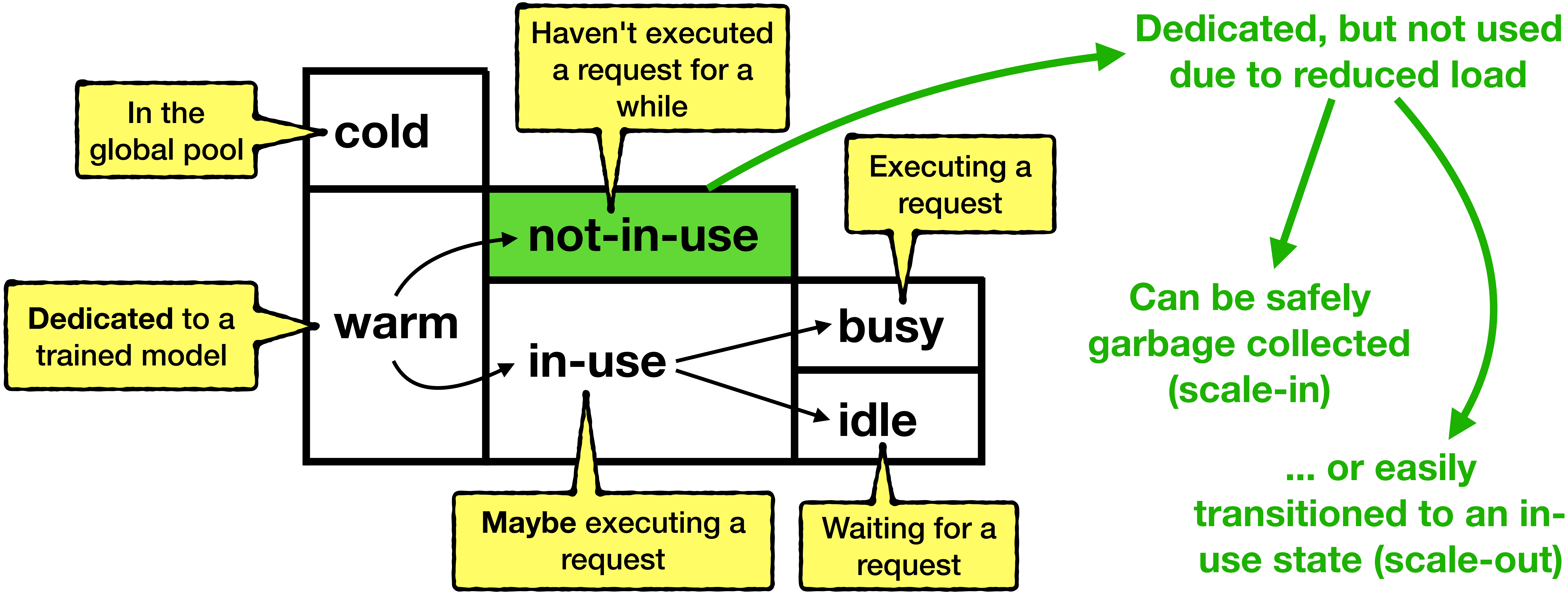
Key idea 1: Assign states to each backend



Key idea 1: Assign states to each backend

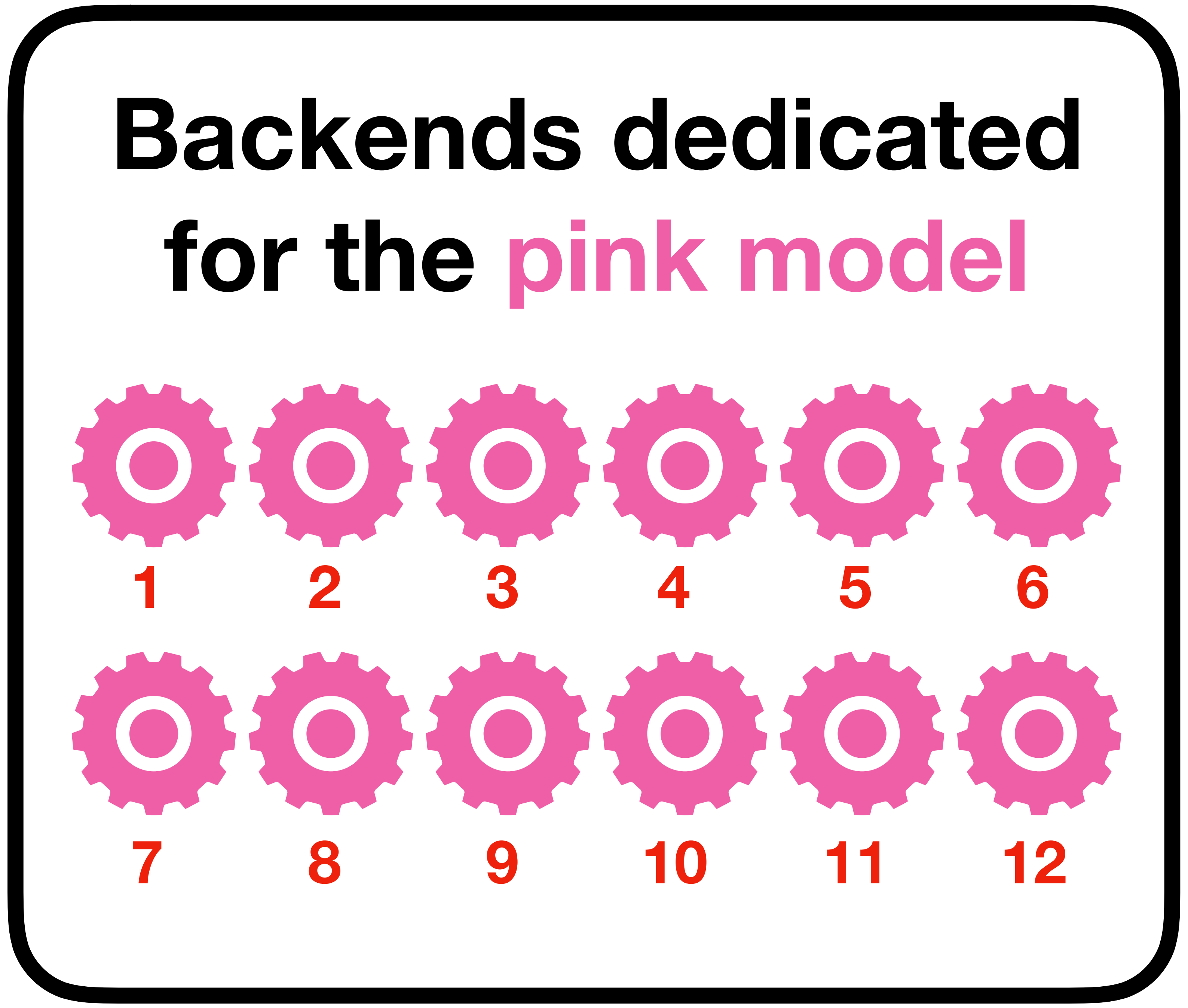


Key idea 1: Assign states to each backend

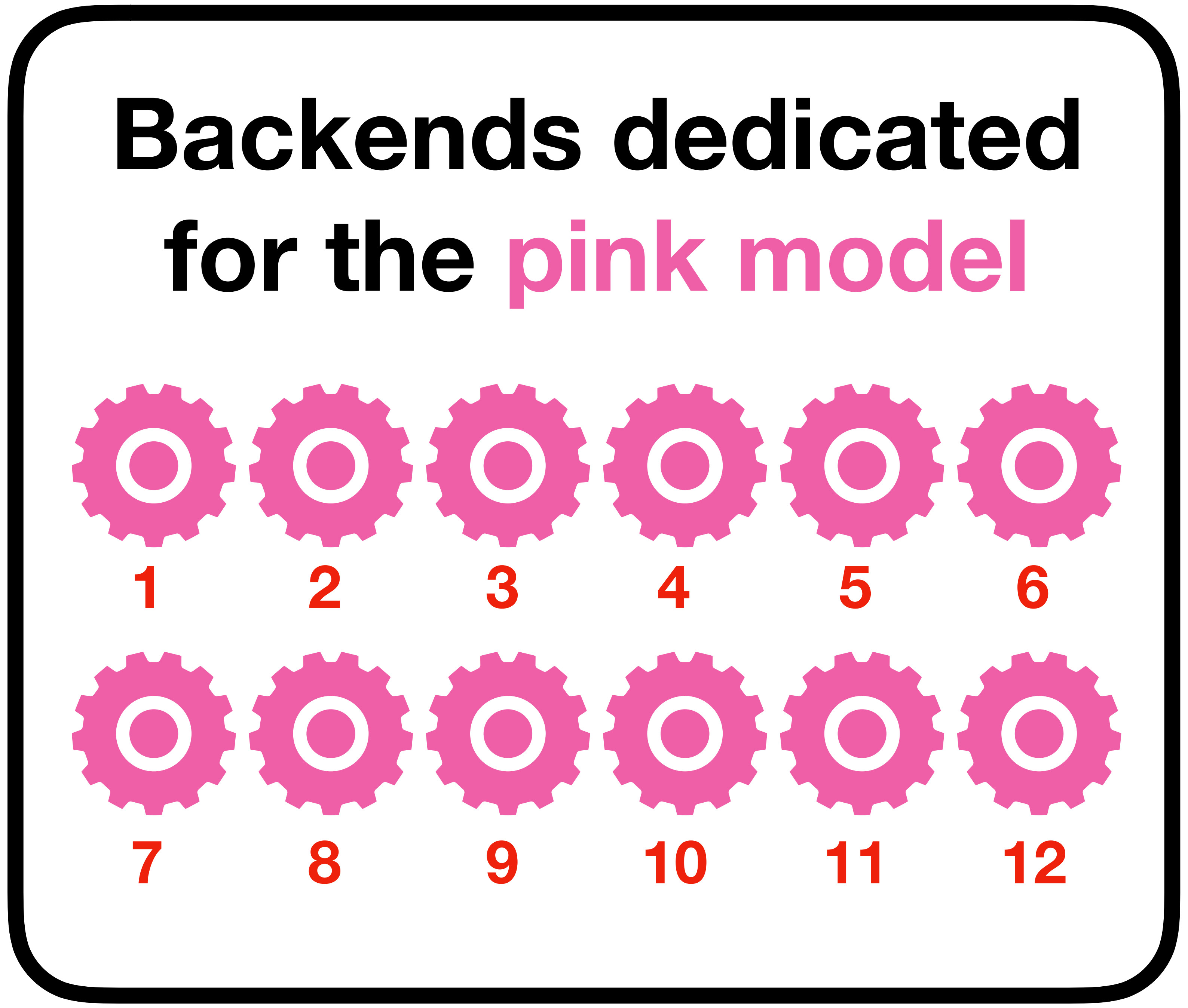


How do frontends know which dedicated backends to use, and which to not use?

Key idea 2: Order the dedicated set of backends

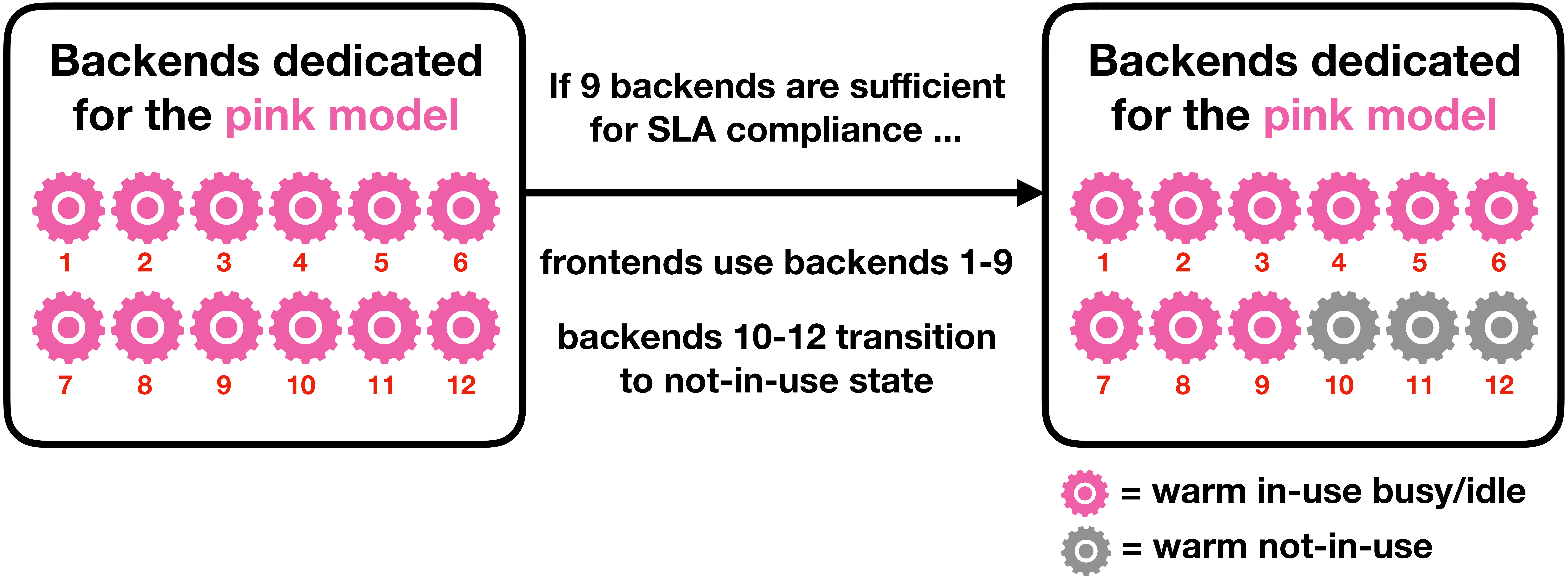


Key idea 2: Order the dedicated set of backends

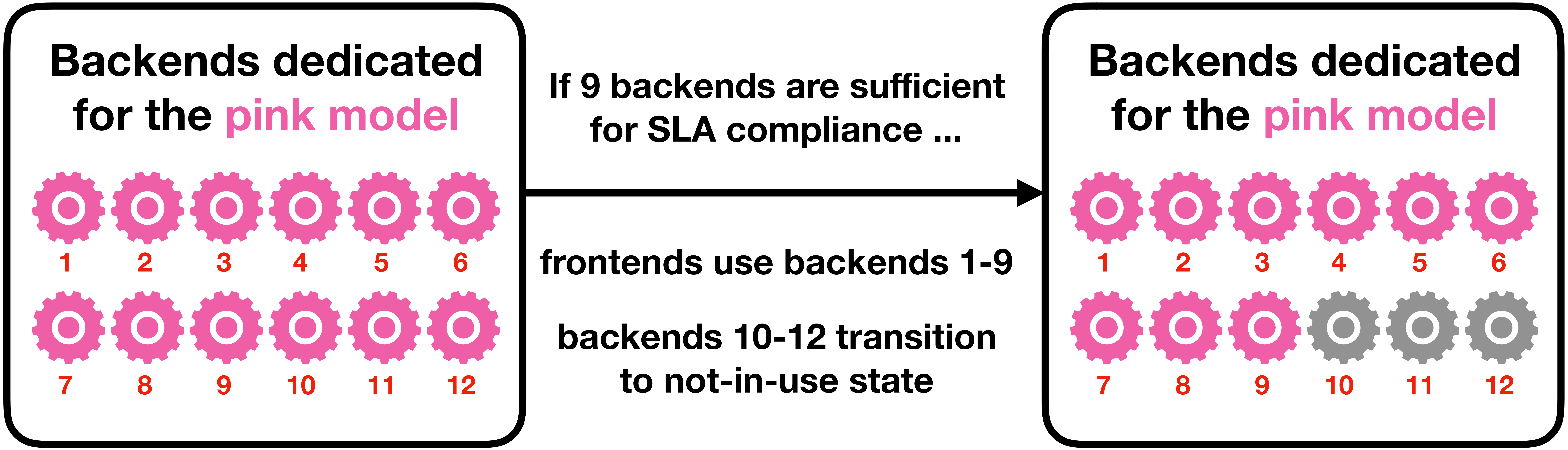


If 9 backends are sufficient for SLA compliance ...

Key idea 2: Order the dedicated set of backends

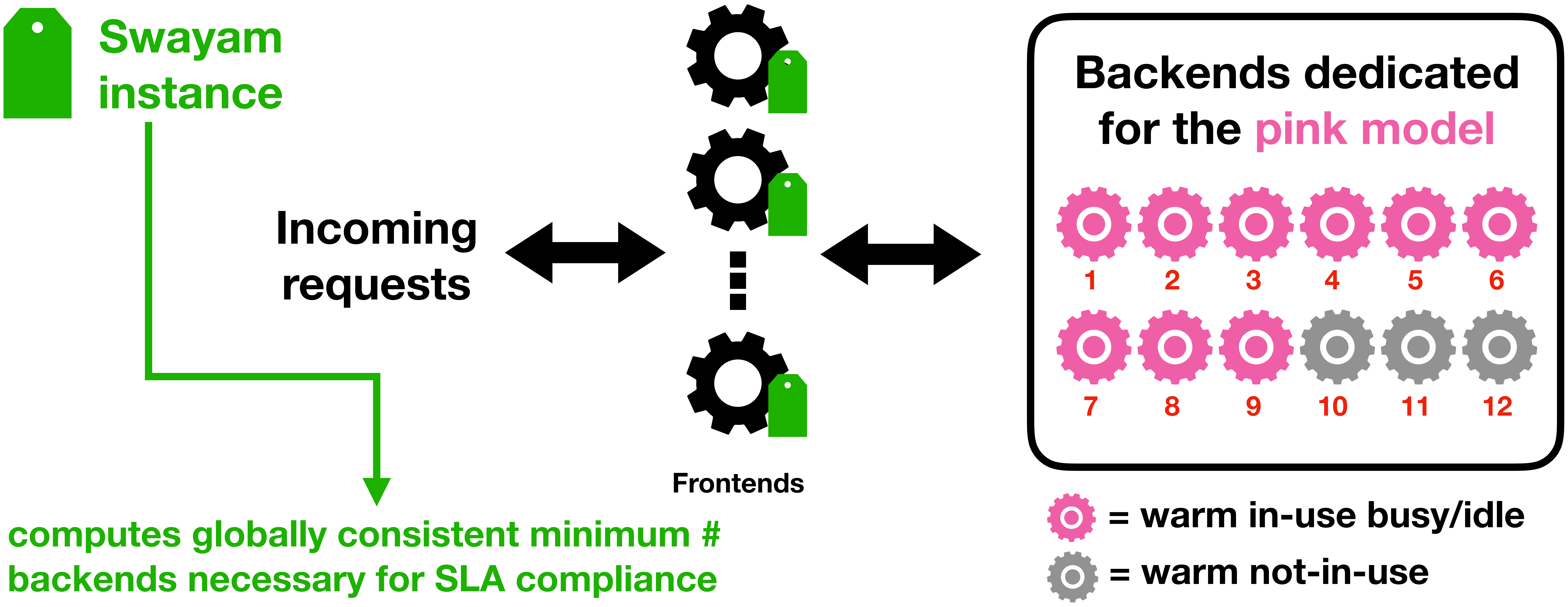


Key idea 2: Order the dedicated set of backends



How do frontends know how many backends are sufficient?

Key idea 3: Swayam instance on every frontend



Outline

1. System architecture, key ideas

2. Analytical model for resource estimation

3. Evaluation results

Making globally-consistent decisions

at each frontend (Swayam instance)

What is the minimum # backends required for SLA compliance?

Making globally-consistent decisions

at each frontend (Swayam instance)

What is the minimum # backends required for SLA compliance?

1. Expected request execution time
2. Expected request waiting time
3. Total request load

Making globally-consistent decisions

at each frontend (Swayam instance)

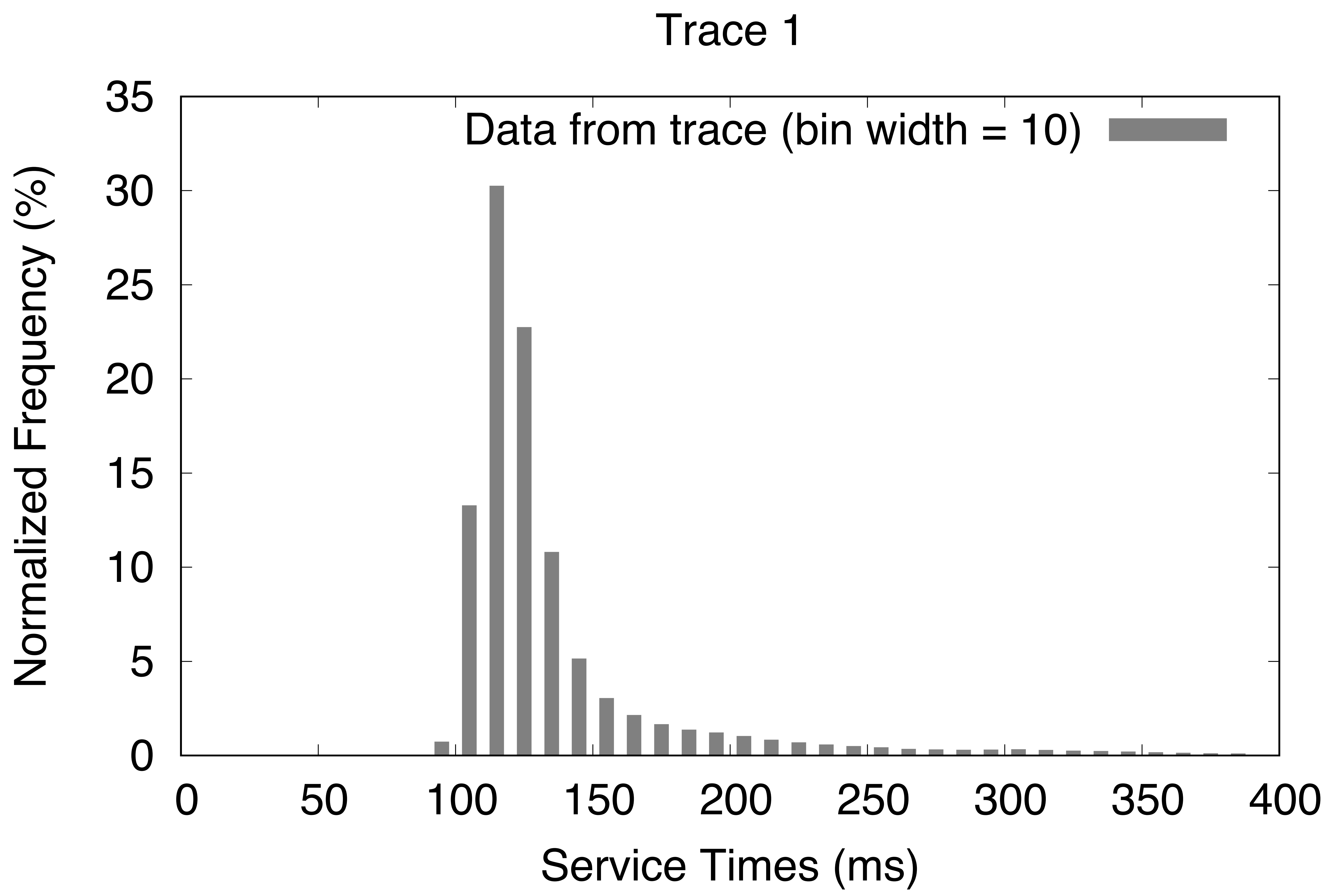
What is the minimum # backends required for SLA compliance?

1. Expected request execution time
2. Expected request waiting time
3. Total request load

leverage ML workload characteristics

Determining expected request execution times

Studied execution traces of 15 popular services hosted on Microsoft Azure's MLaaS platform

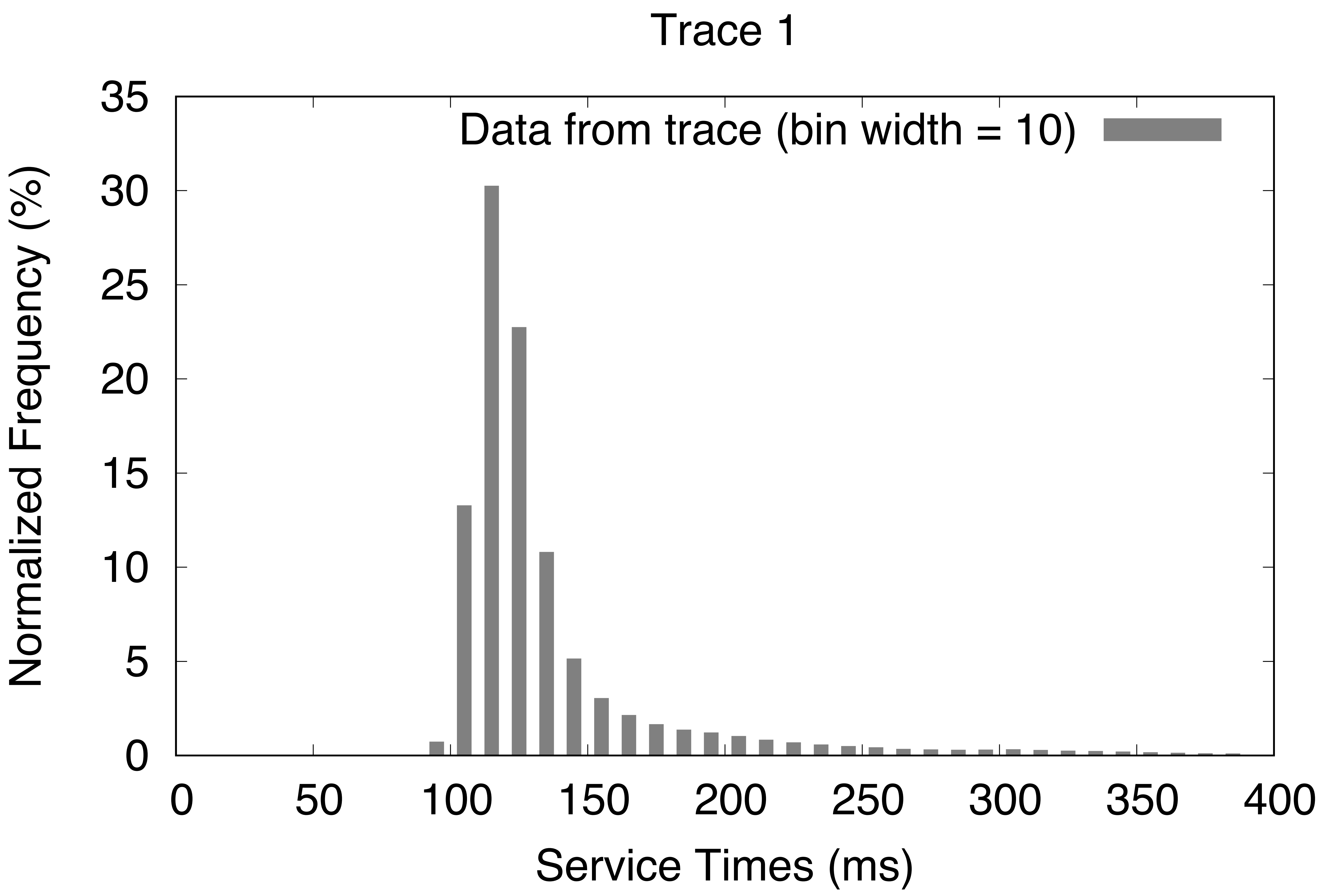


Determining expected request execution times

Studied execution traces of 15 popular services hosted on Microsoft Azure's MLaaS platform

Variation is low

- Fixed-sized feature vectors
- Input-independent control flow
- Non-deterministic machine & OS events main sources of variability

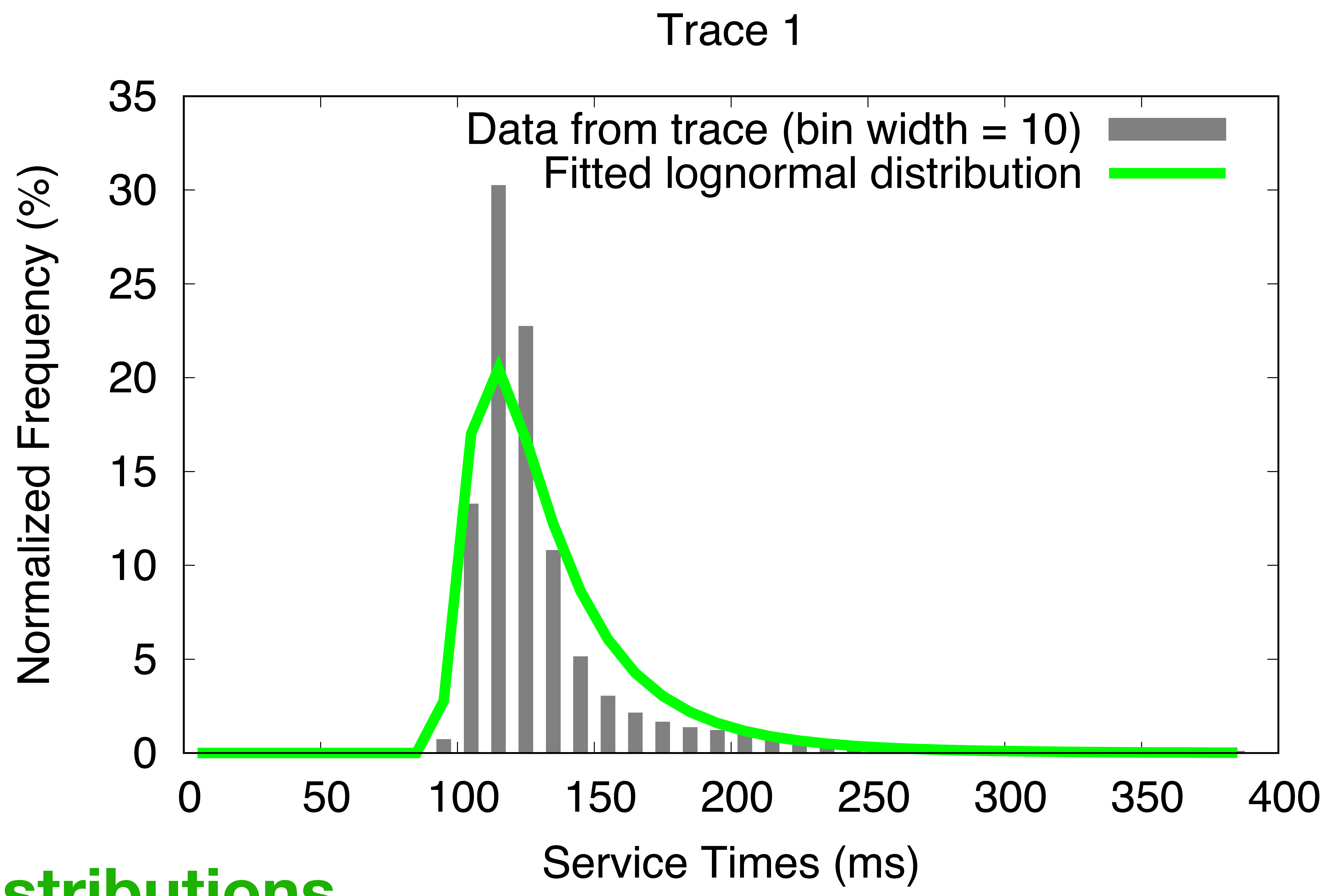


Determining expected request execution times

Studied execution traces of 15 popular services hosted on Microsoft Azure's MLaaS platform

Variation is low

- Fixed-sized feature vectors
- Input-independent control flow
- Non-deterministic machine & OS events main sources of variability



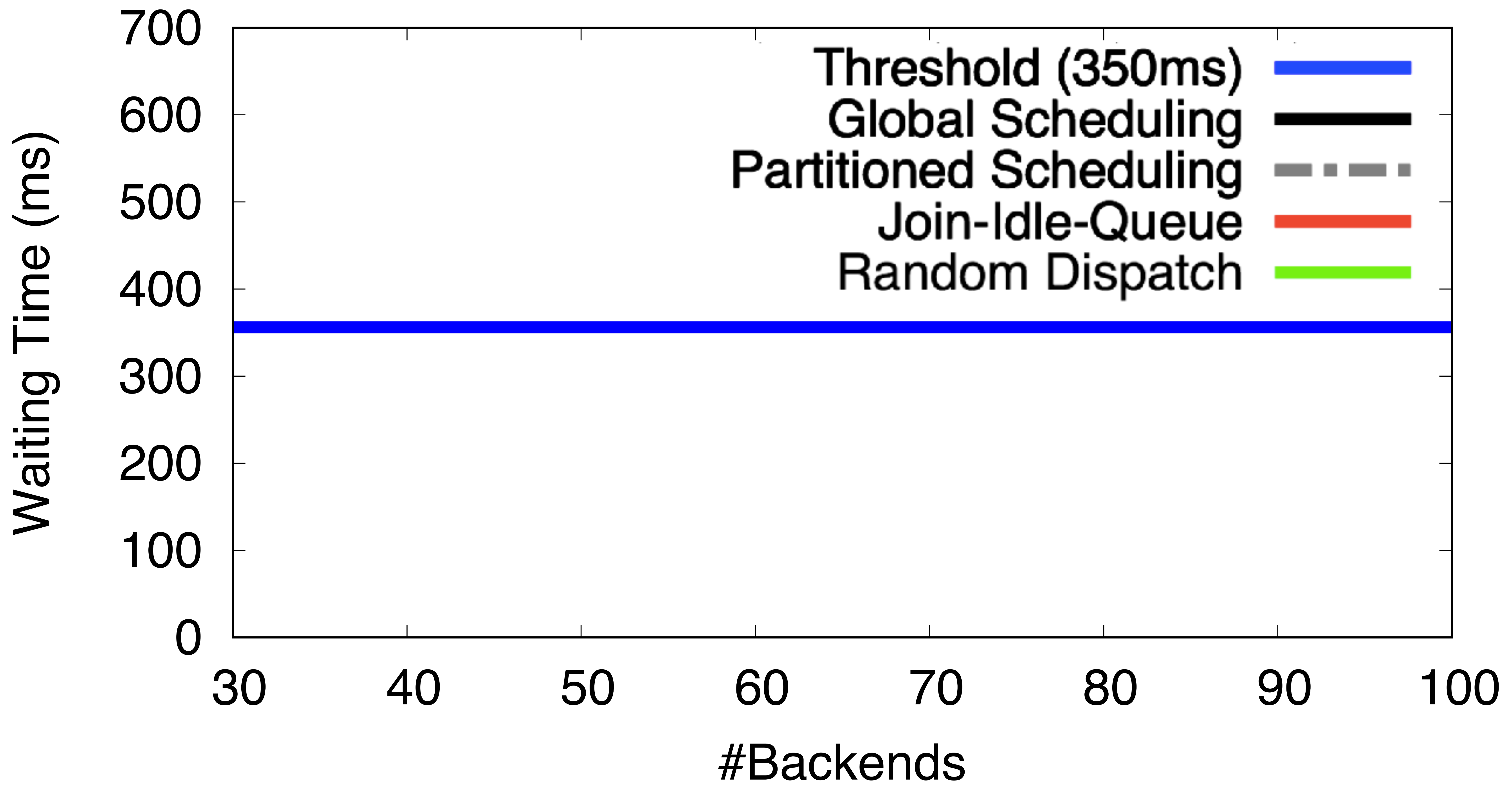
Modeled using log-normal distributions

Determining expected request waiting times

load balancing (LB)

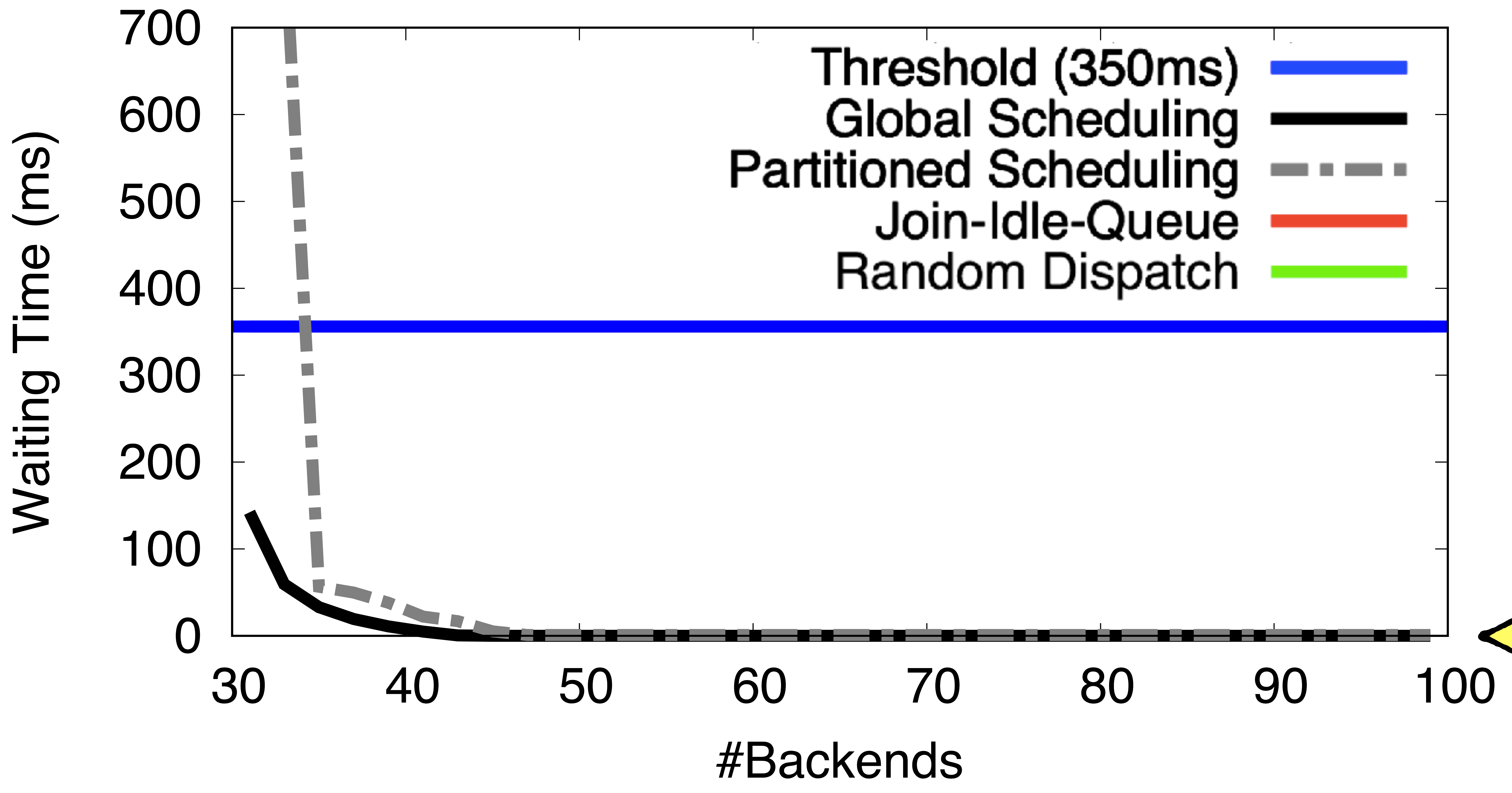
Determining expected request waiting times

load balancing (LB)



Determining expected request waiting times

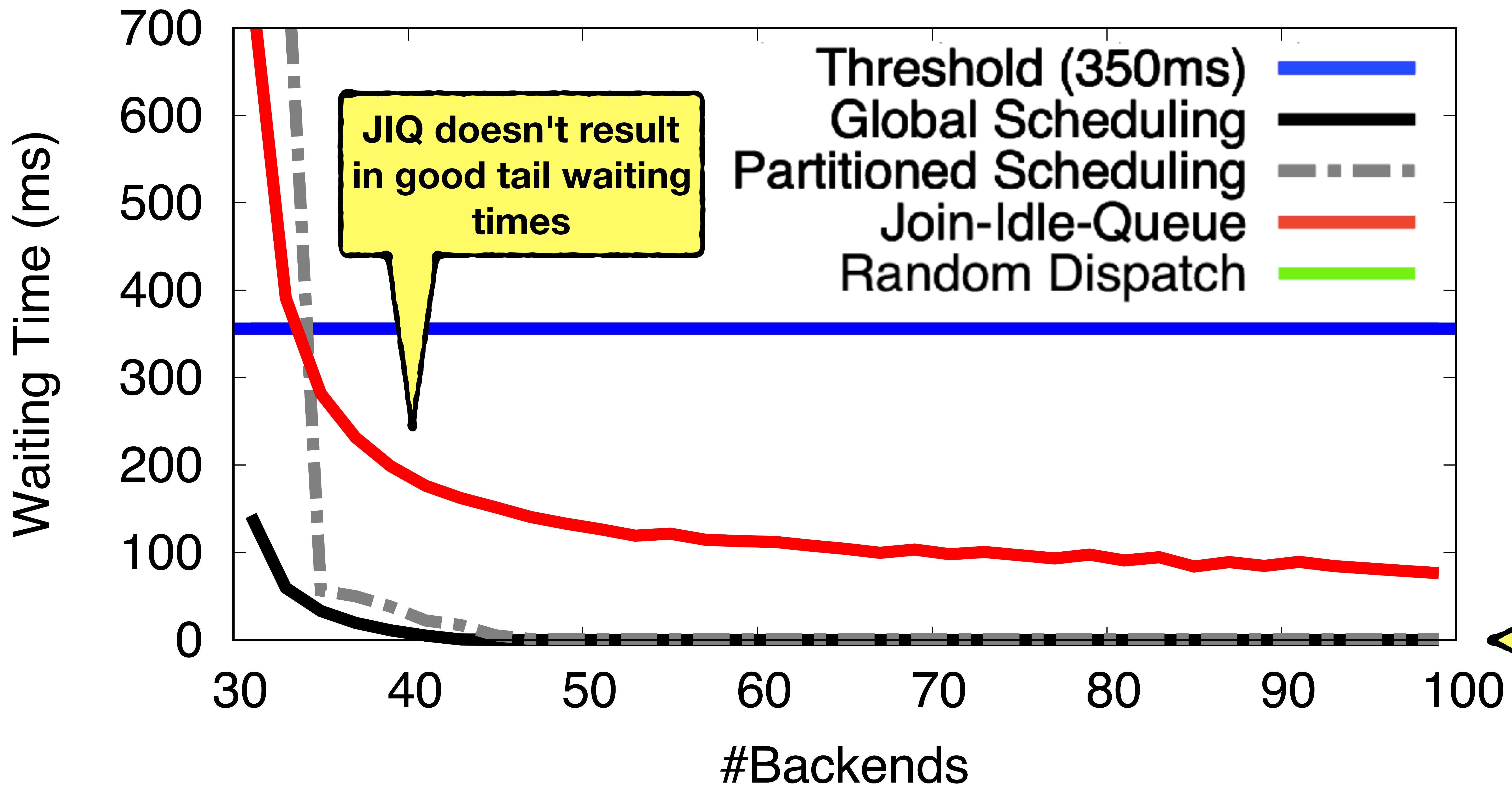
load balancing (LB)



Global and partitioned perform well, but there are implementation tradeoffs

Determining expected request waiting times

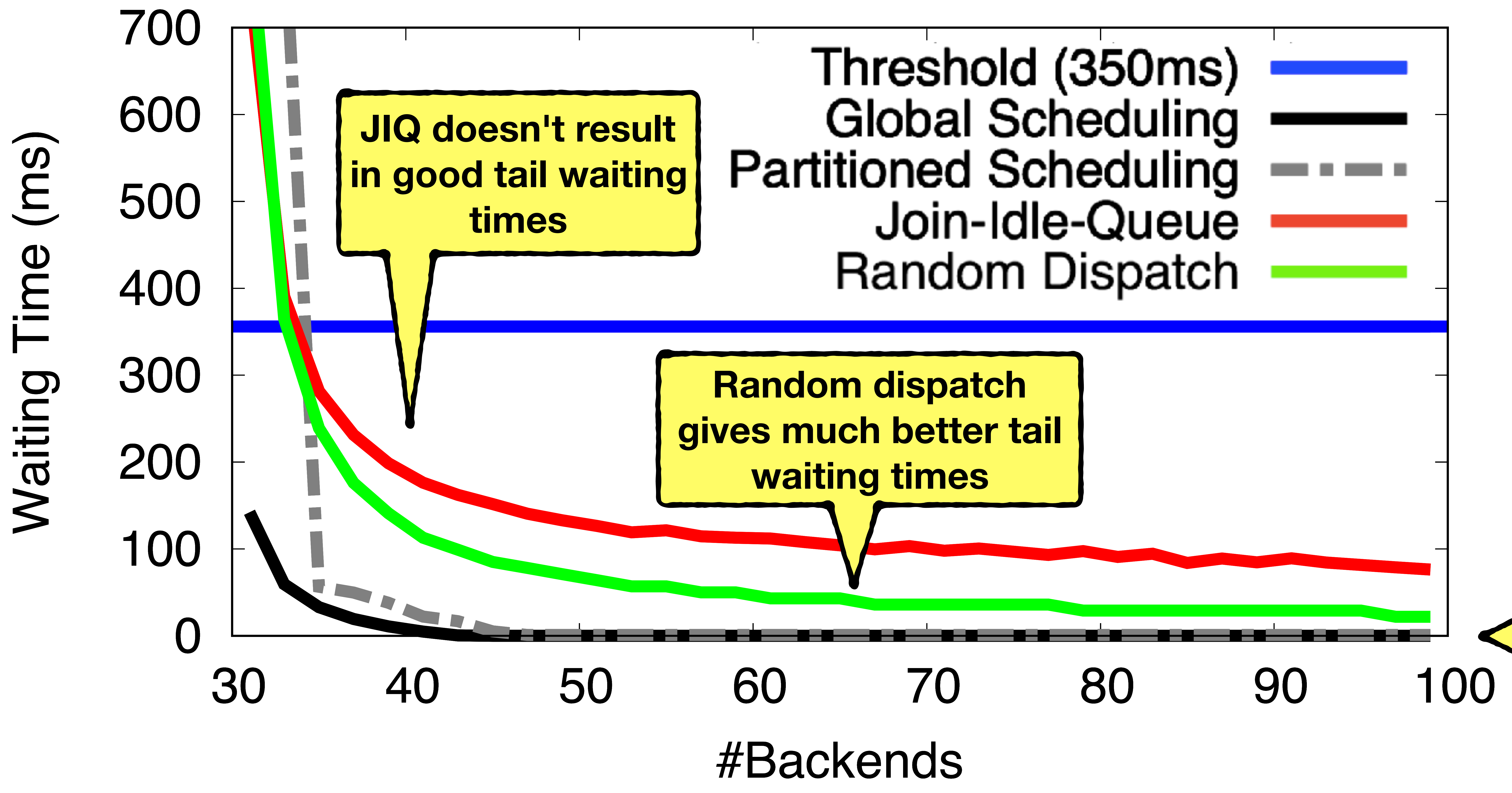
load balancing (LB)



Global and partitioned perform well, but there are implementation tradeoffs

Determining expected request waiting times

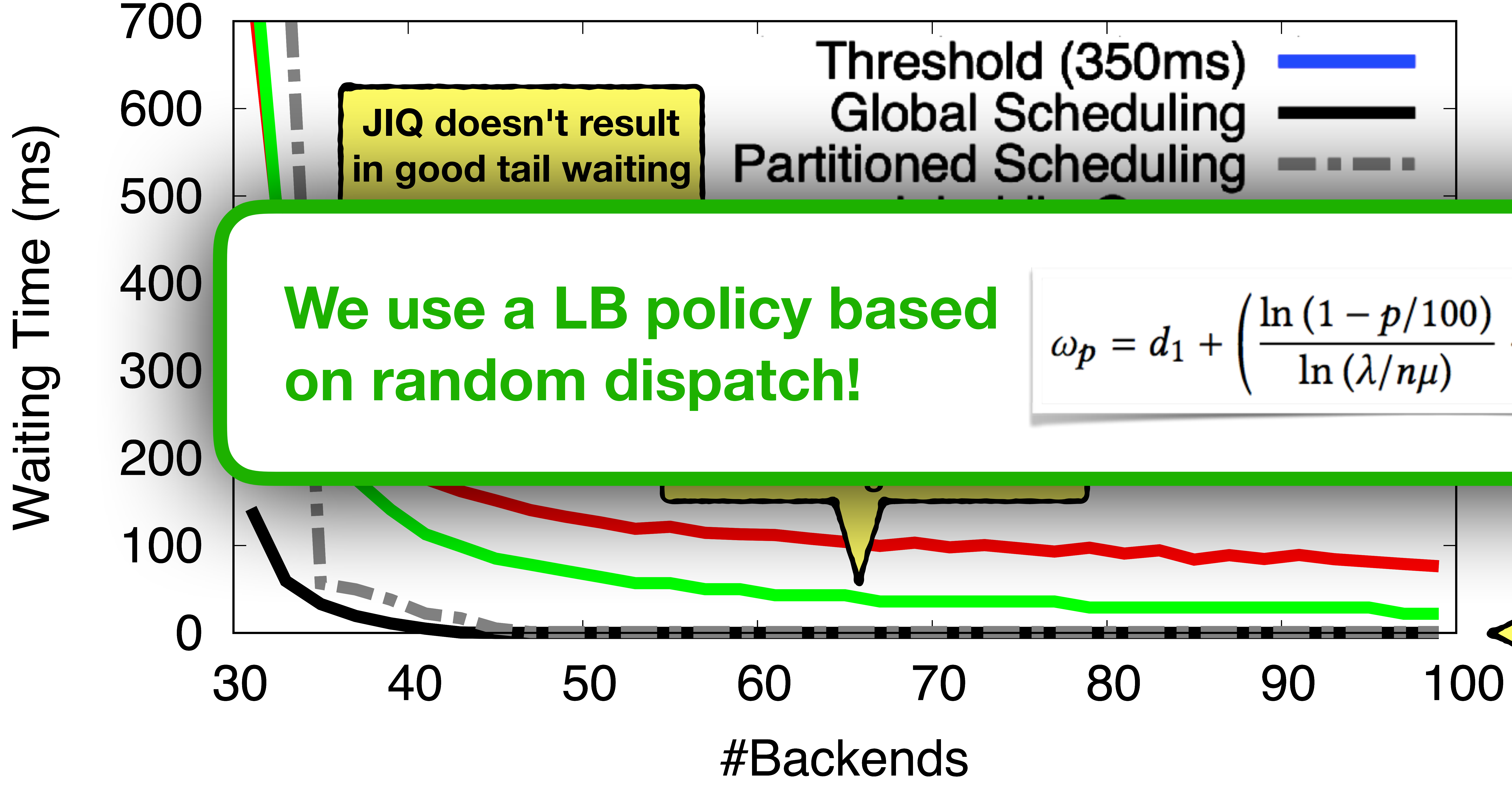
load balancing (LB)



Global and partitioned perform well, but there are implementation tradeoffs

Determining expected request waiting times

load balancing (LB)



We use a LB policy based on random dispatch!

$$\omega_p = d_1 + \left(\frac{\ln(1 - p/100)}{\ln(\lambda/n\mu)} - 1 \right) \cdot (d_1 + d_2 + \Delta),$$

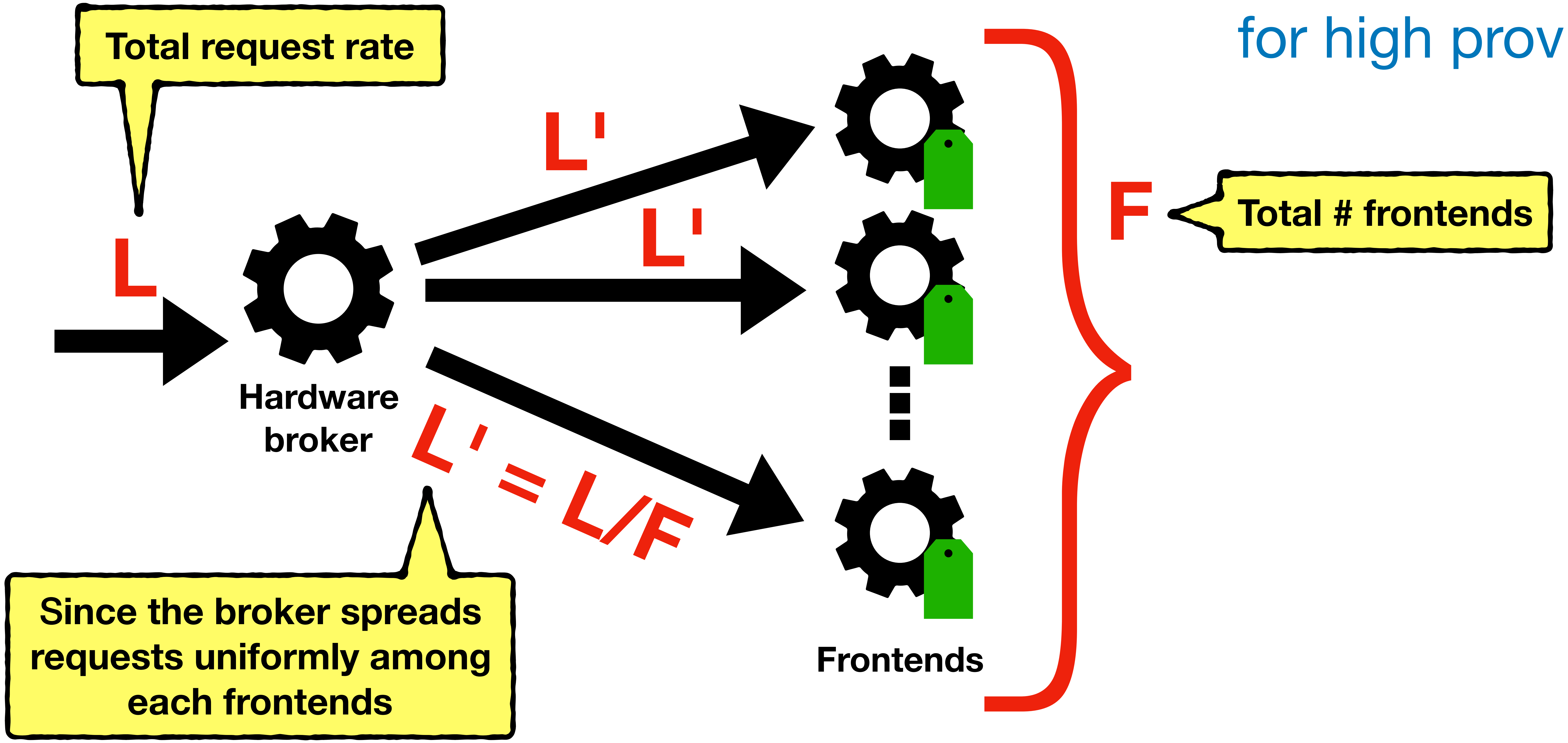
Global and partitioned perform well, but there are implementation tradeoffs

Determining the total request load

in the near future, to account
for high provisioning times

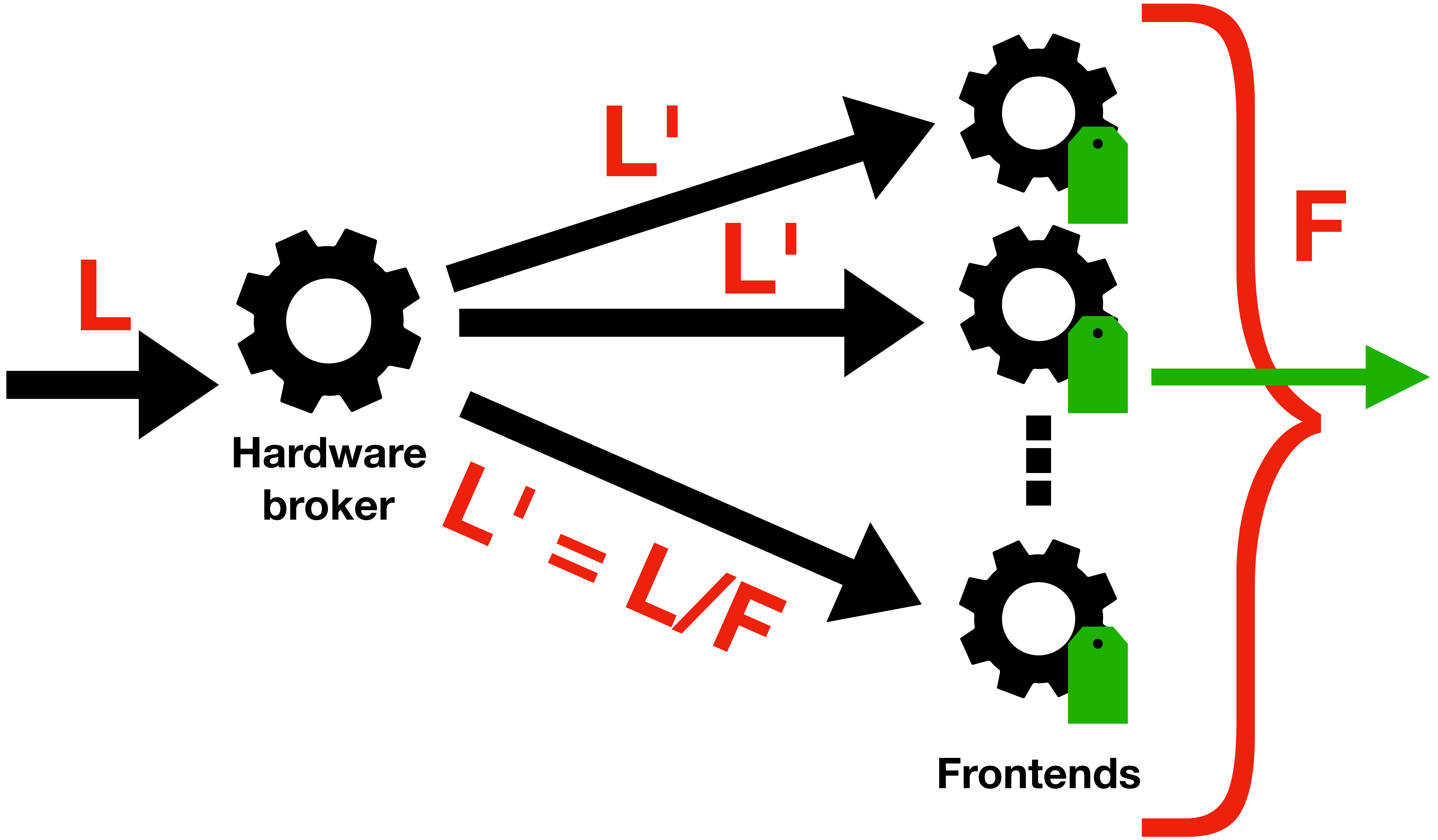
Determining the total request load

in the near future, to account for high provisioning times



Determining the total request load

in the near future, to account for high provisioning times



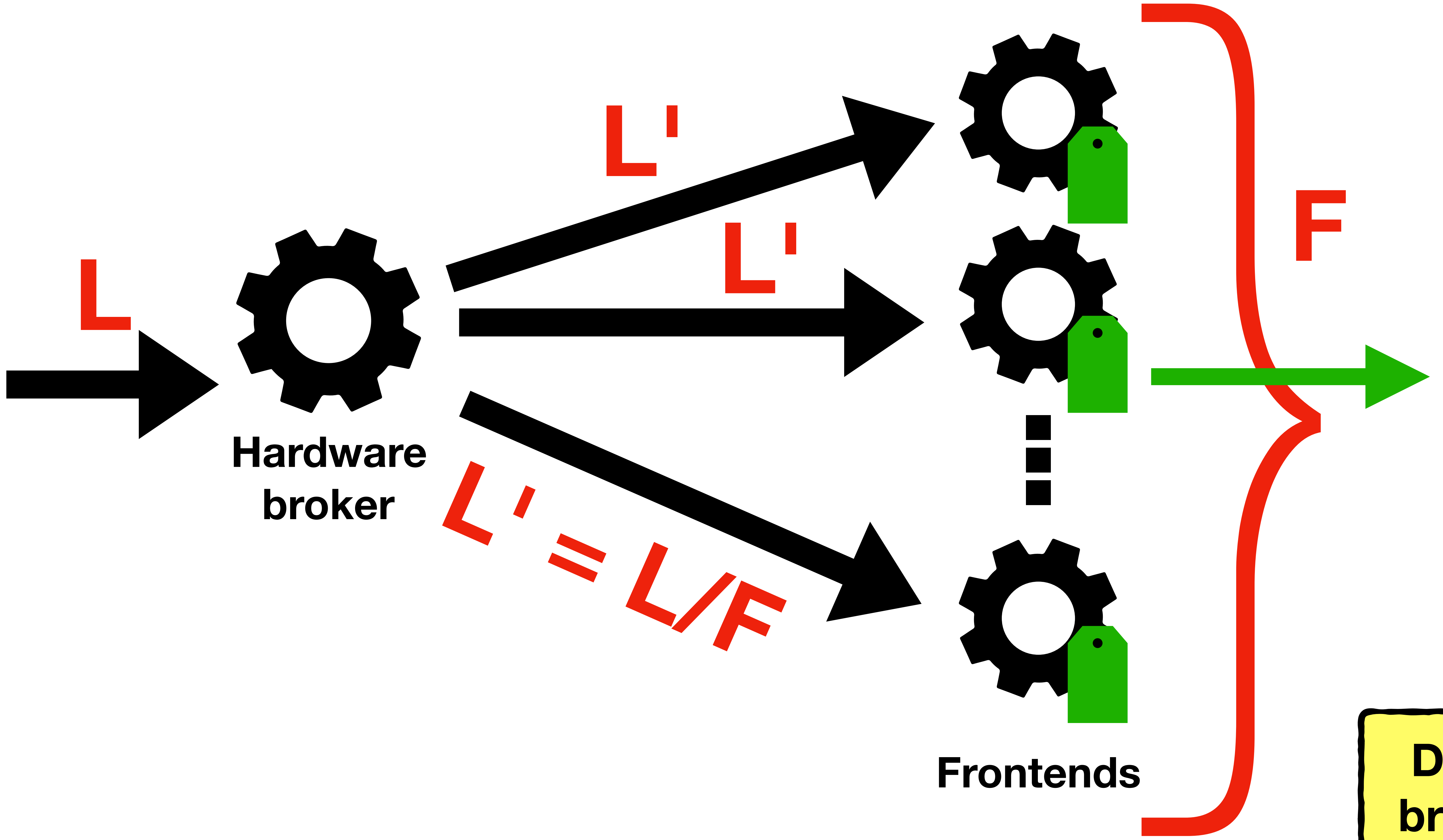
Each Swayam instance

- **Predicts L' for <u>near future</u>**

Depends on the time to setup a new backend

Determining the total request load

in the near future, to account for high provisioning times



Each Swayam instance

- Predicts L' for near future
- Given F, computes L = F x L'

Determined from broker / through a gossip protocol

Making globally-consistent decisions

at each frontend (Swayam instance)

What is the minimum # backends required for SLA compliance?

1. Expected request execution time ✓
2. Expected request waiting time ✓
3. Total request load ✓

SLA-aware resource estimation

For each trained model

n = min # backends

Response-Time Threshold

RT_{max}

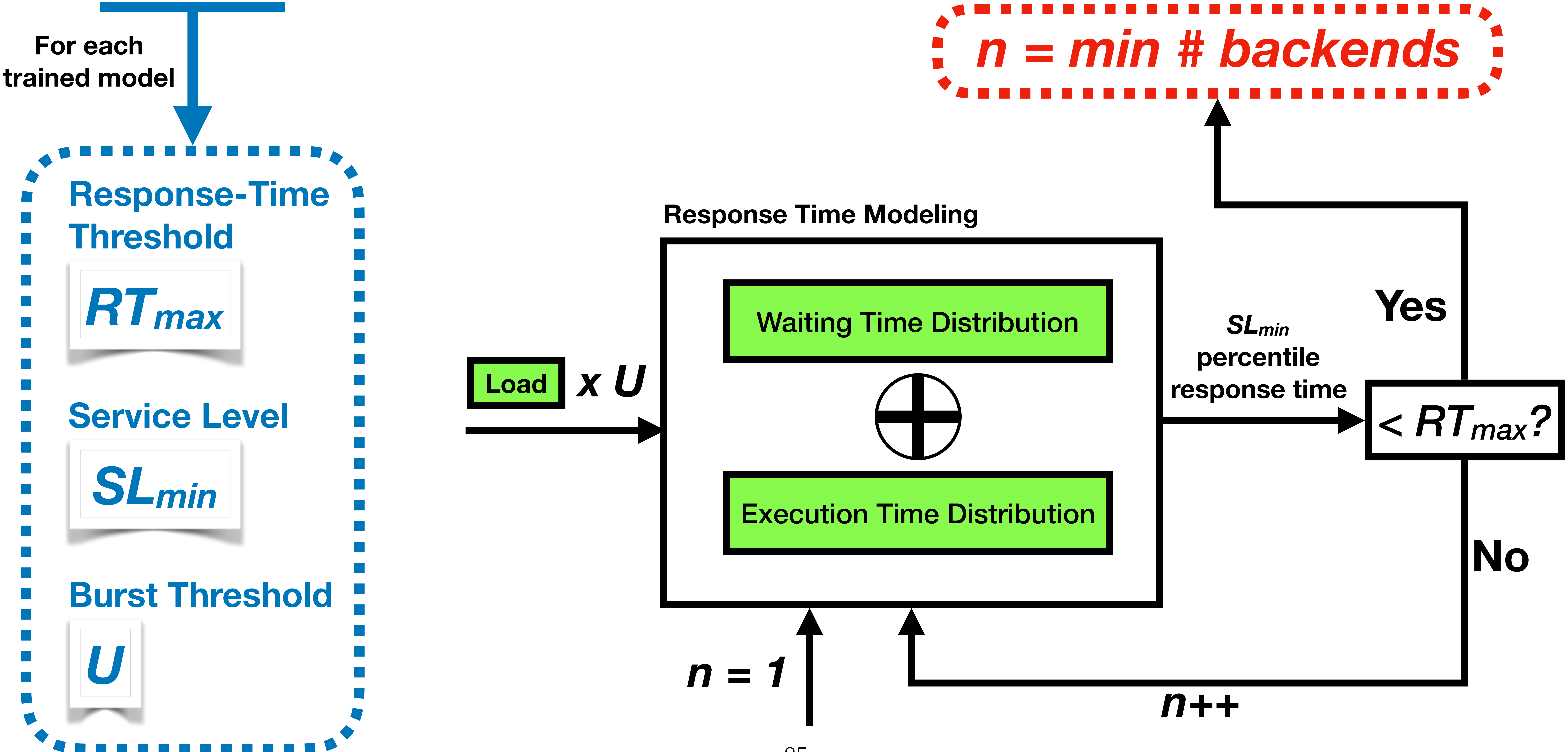
Service Level

SL_{min}

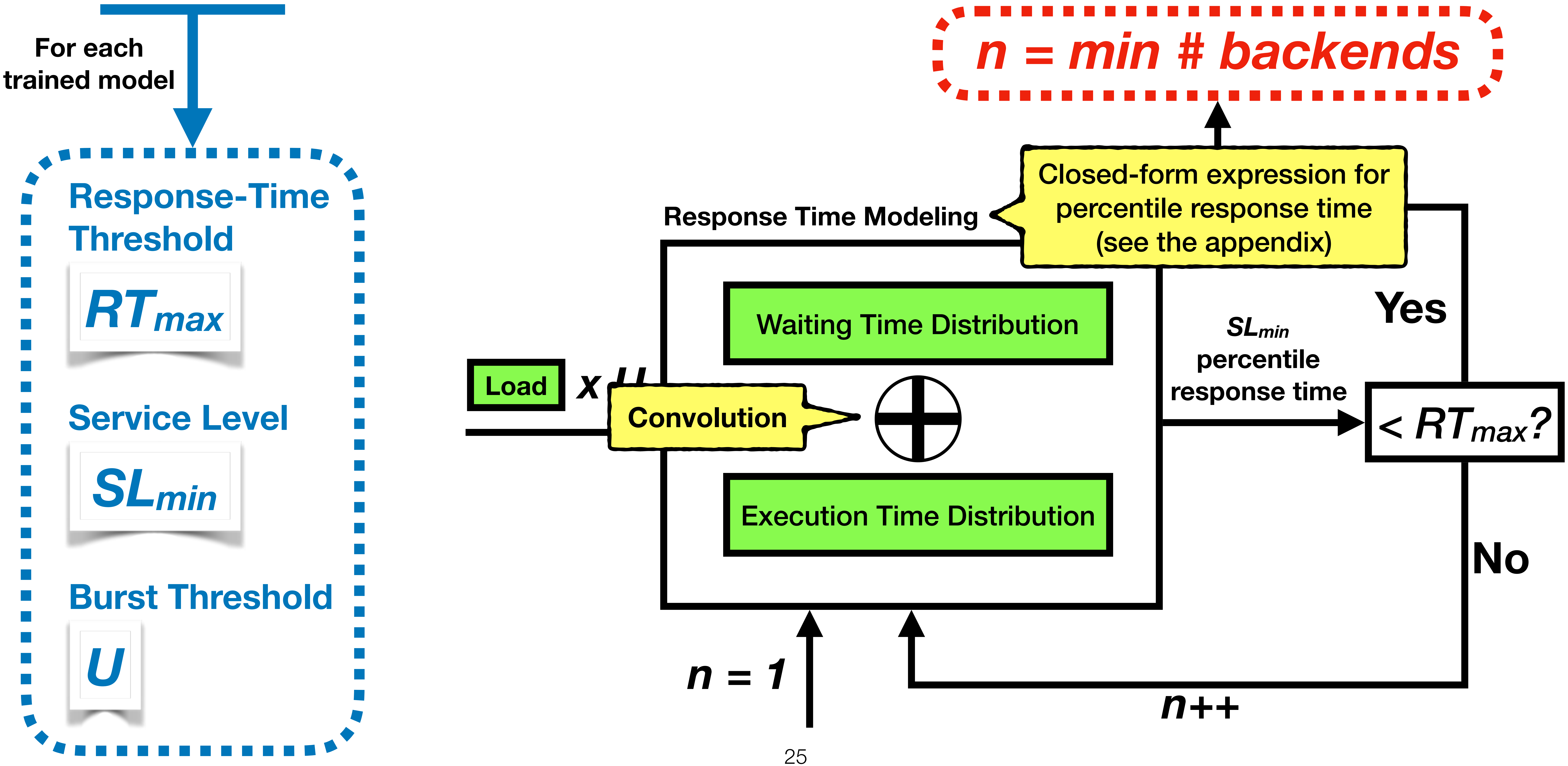
Burst Threshold

U

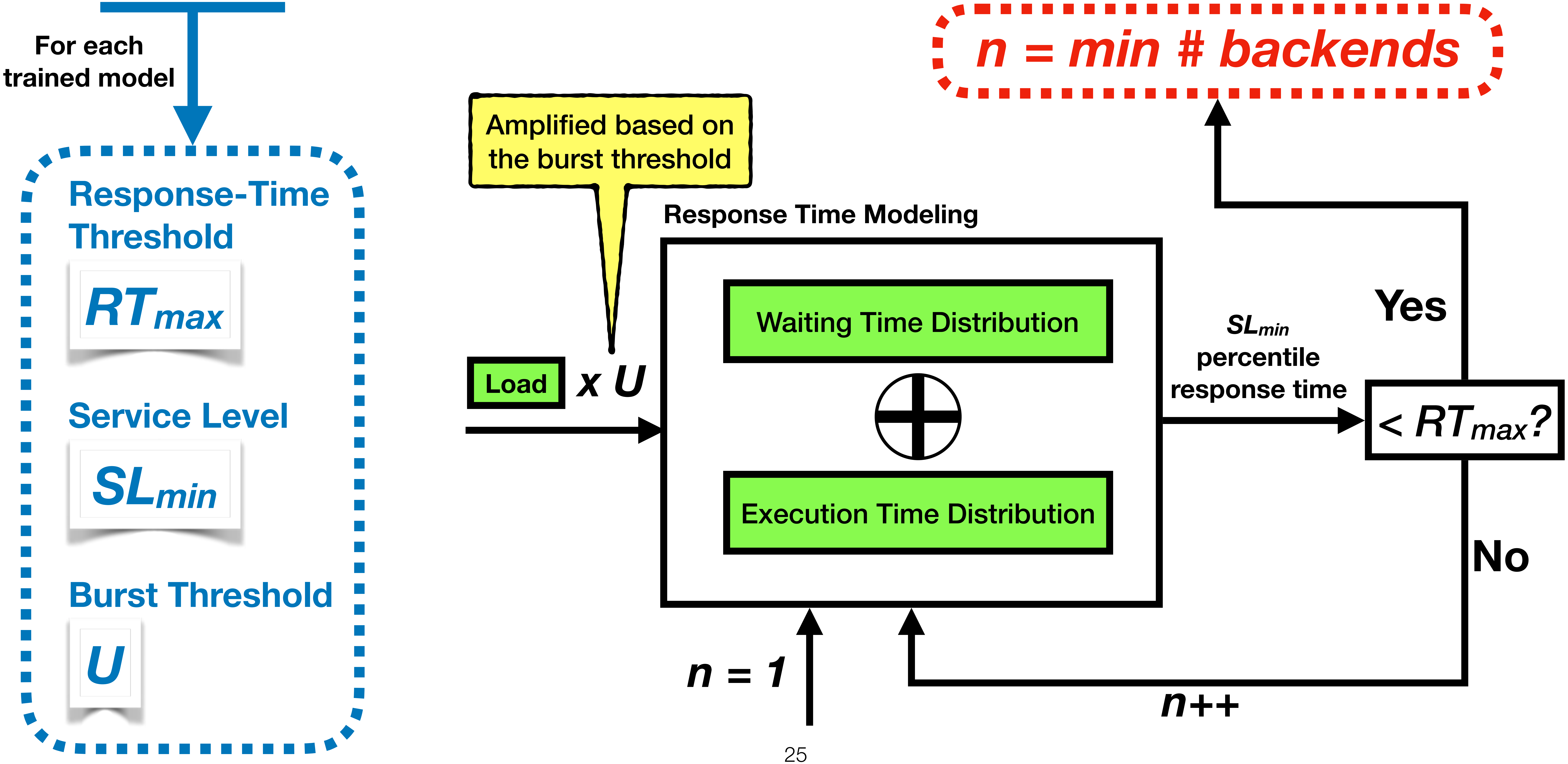
SLA-aware resource estimation



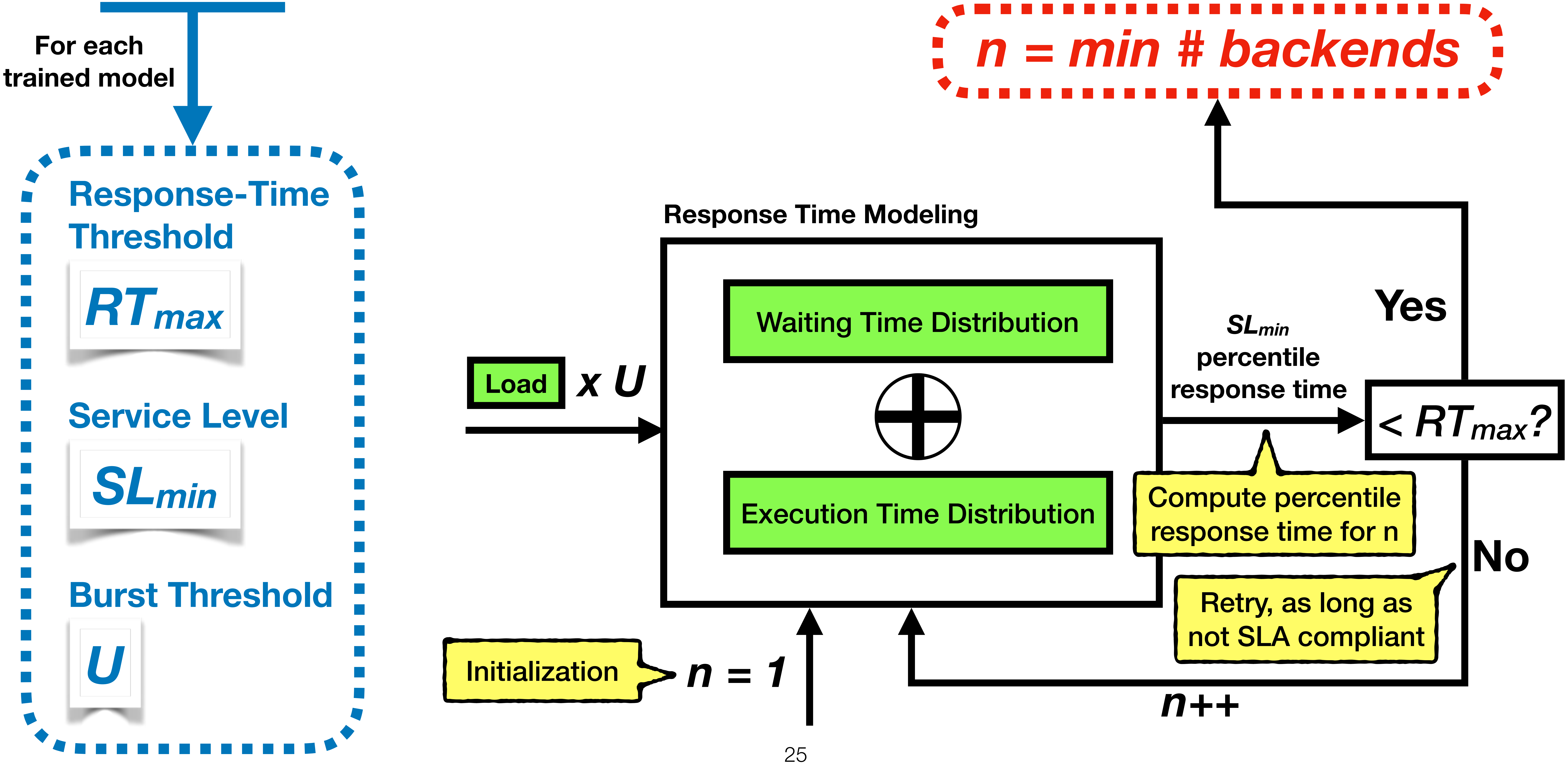
SLA-aware resource estimation



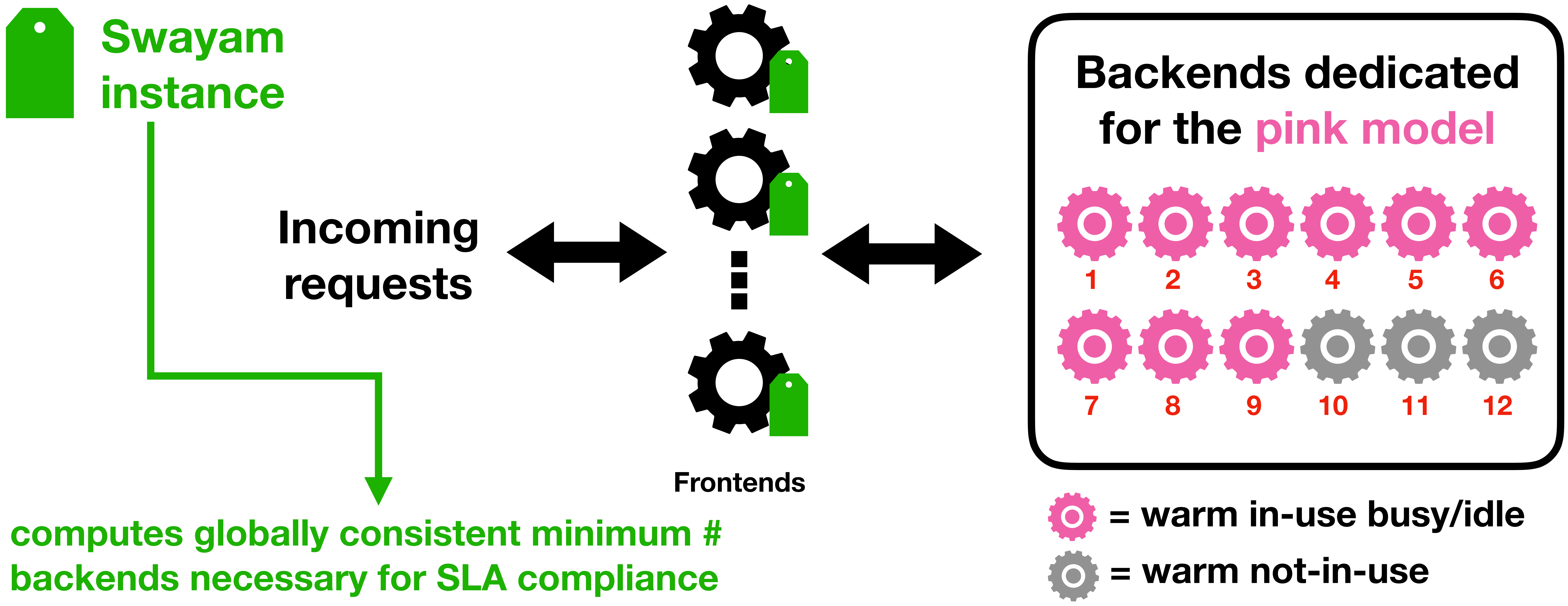
SLA-aware resource estimation



SLA-aware resource estimation



Swayam Framework



Outline

1. System architecture, key ideas
2. Analytical model for resource estimation
3. **Evaluation results**

Evaluation setup

- Prototype in C++ on top of Apache Thrift
  ➡ 100 backends per service
  ➡ 8 frontends
  ➡ 1 broker
  ➡ 1 server (for simulating the clients)

Evaluation setup

- Prototype in C++ on top of Apache Thrift
  - ➡ 100 backends per service
  - ➡ 8 frontends
  - ➡ 1 broker
  - ➡ 1 server (for simulating the clients)
- Workload
  - ➡ 15 production service traces (Microsoft Azure MLaaS)
  - ➡ Three-hour traces (request arrival times and computation times)
  - ➡ Query computation & model setup times emulated by spinning

SLA configuration for each model

- Response-time threshold RT_{max} = $5C$

  ➡ C denotes the mean computation time for the model

- Desired service level SL_{min} = 99%

  ➡ 99% of the requests must have response times under RT_{max}

- Burst threshold $U = 2x$

  ➡ Tolerate increase in request rate by up to 100%

- Initially, 5 pre-provisioned backends

Baseline: Clairvoyant Autoscaler (ClairA)

➡ It knows the processing time of each request beforehand

➡ It can travel back in time to provision a backend

➡ "Deadline-driven" approach to minimize resource waste

Baseline: Clairvoyant Autoscaler (ClairA)

➡ **It knows the processing time of each request beforehand**

➡ **It can travel back in time to provision a backend**

➡ **"Deadline-driven" approach to minimize resource waste**

- ClairA1 assumes zero setup times, immediate scale-ins

  ➡ Reflects the size of the workload

Baseline: Clairvoyant Autoscaler (ClairA)

➡ **It knows the processing time of each request beforehand**

➡ **It can travel back in time to provision a backend**

➡ **"Deadline-driven" approach to minimize resource waste**

- ClairA1 assumes zero setup times, immediate scale-ins

  ➡ Reflects the size of the workload

- ClairA2 assumes non-zero setup times, lazy scale-ins

  ➡ Swayam-like

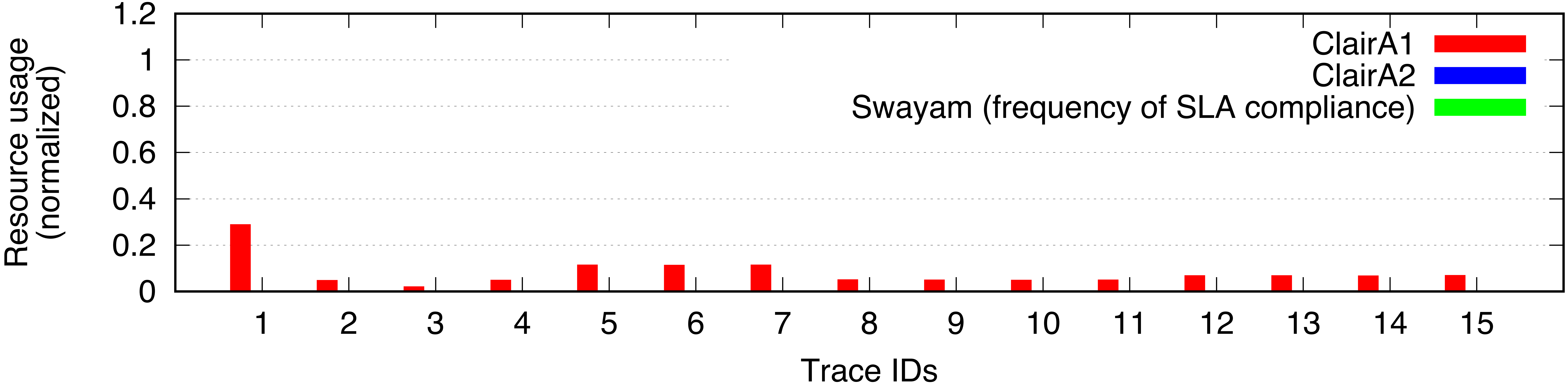
Baseline: Clairvoyant Autoscaler (ClairA)

➡ It knows the processing time of each request beforehand
➡ It can travel back in time to provision a backend
➡ "Deadline-driven" approach to minimize resource waste

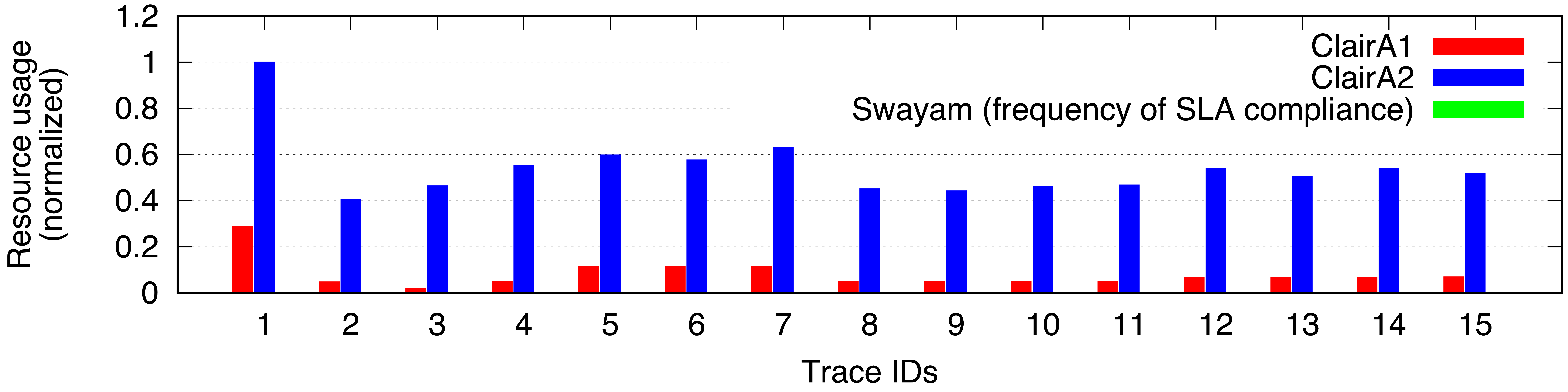
- ClairA1 assumes zero setup times, immediate scale-ins
  ➡ Reflects the size of the workload
- ClairA2 assumes non-zero setup times, lazy scale-ins
  ➡ Swayam-like
- Both ClairA1 and ClairA2 depend on RT_{max}, but not on SL_{min} and U

Resource usage vs. SLA compliance

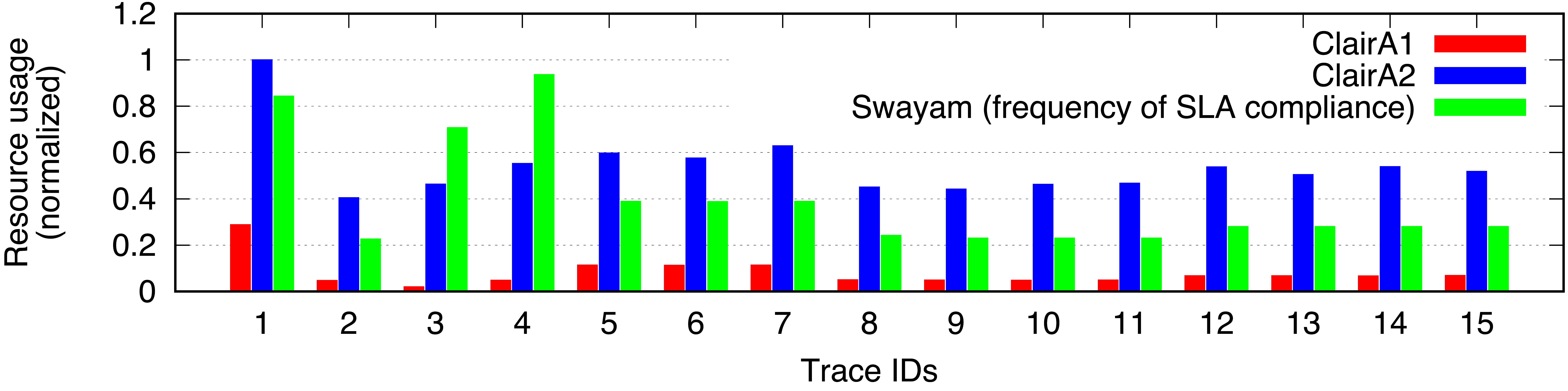
Resource usage vs. SLA compliance



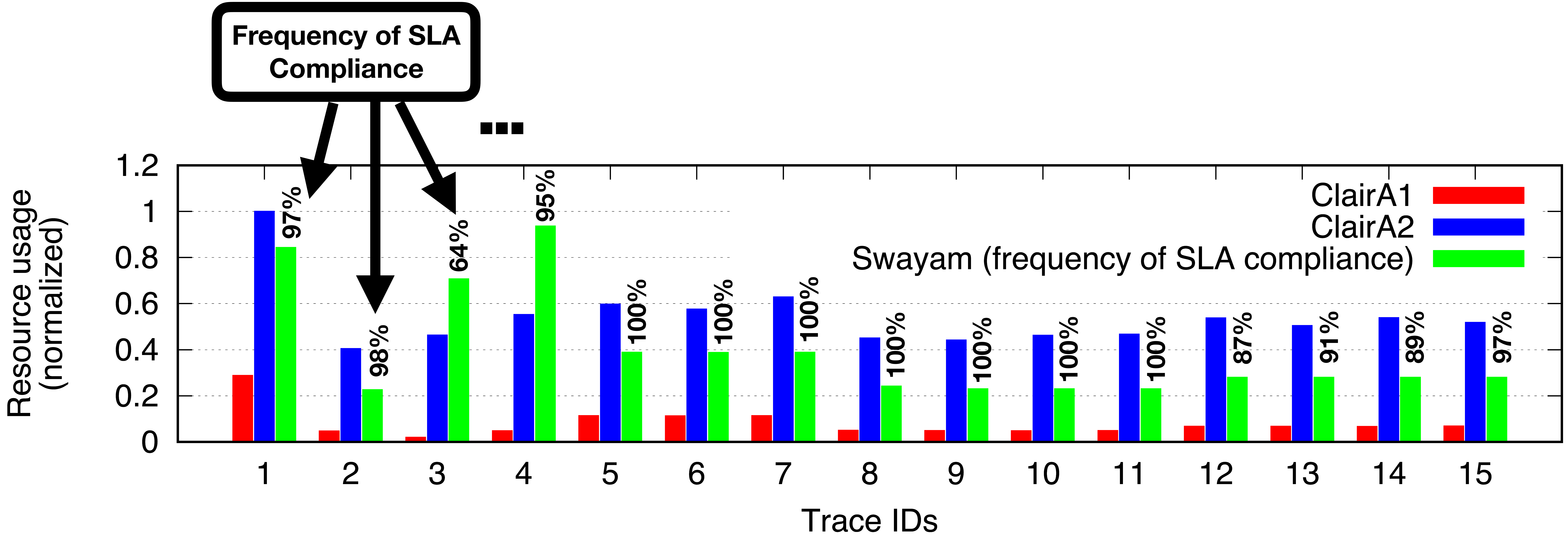
Resource usage vs. SLA compliance



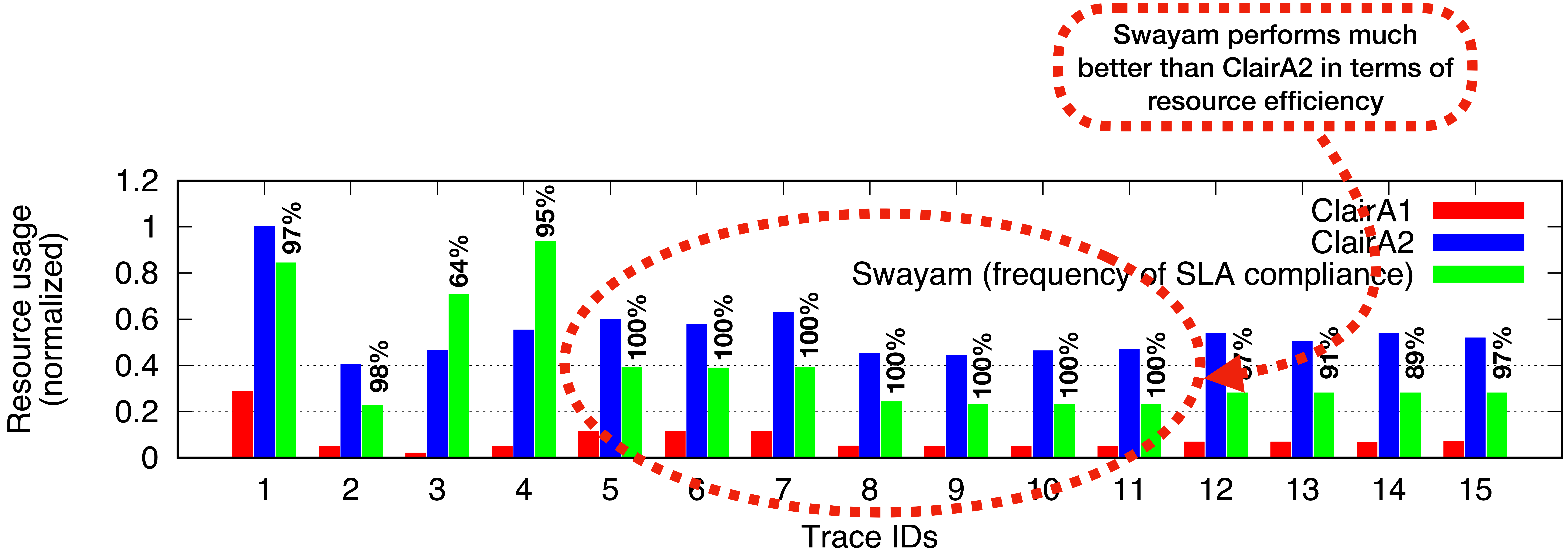
Resource usage vs. SLA compliance



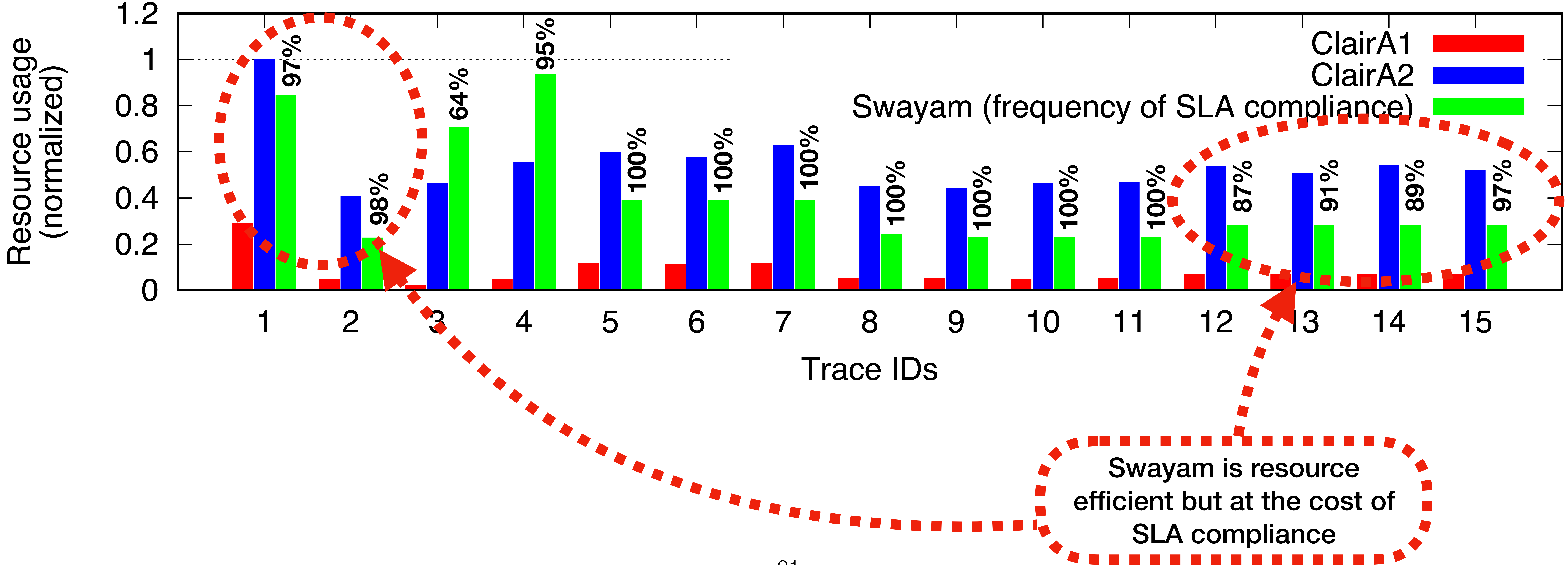
Resource usage vs. SLA compliance



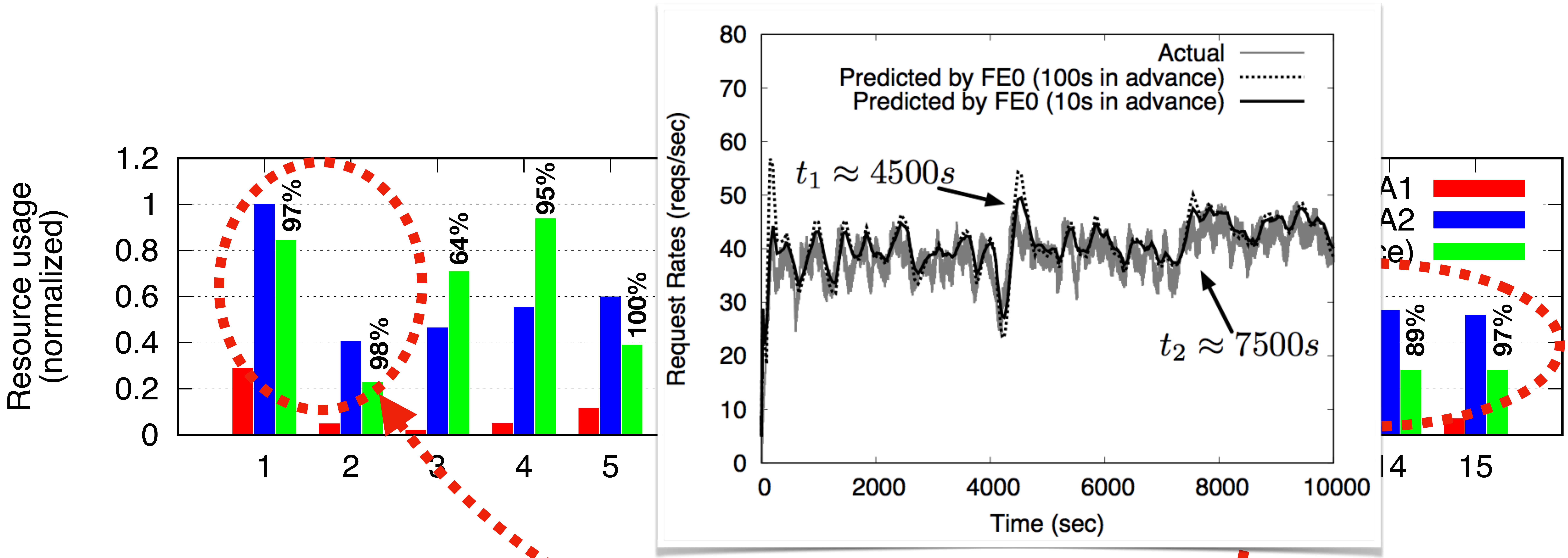
Resource usage vs. SLA compliance



Resource usage vs. SLA compliance



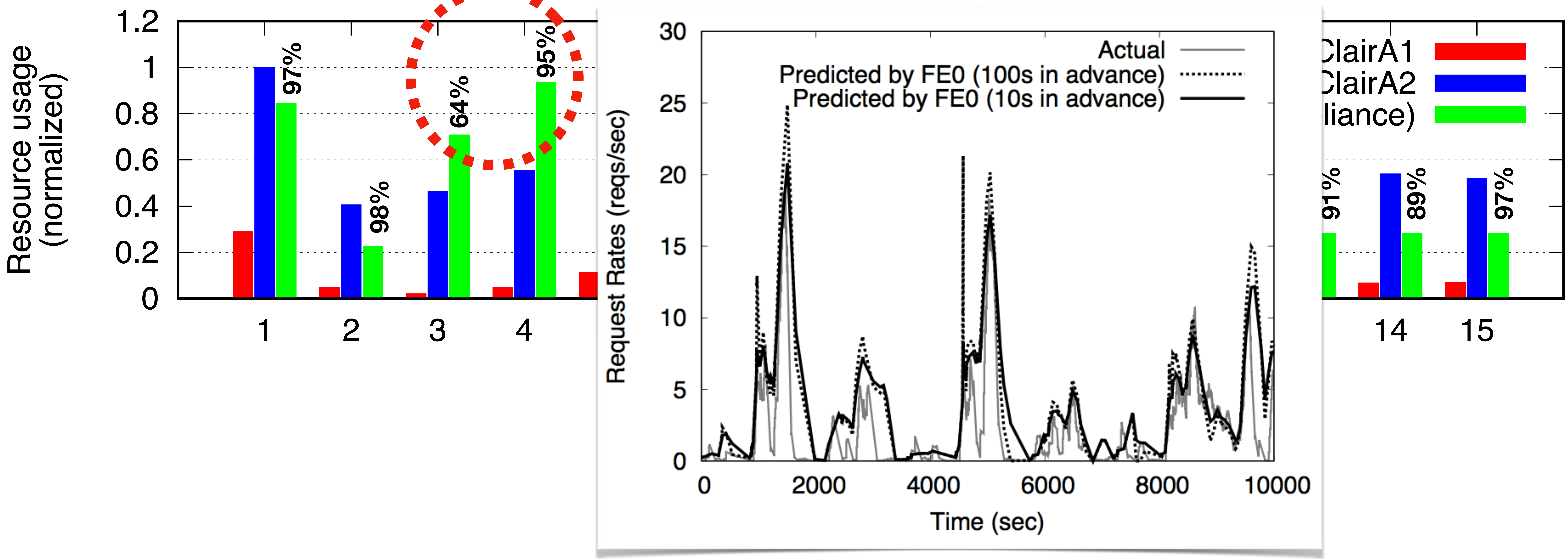
Resource usage vs. SLA compliance



Swayam is resource efficient but at the cost of SLA compliance

Resource usage vs. SLA compliance

Swayam seems to perform poorly
because of a very bursty trace



Summary

- Perfect SLA, irrespective of the input workload, is too expensive
  - ➡ in terms of resource usage (as modeled by ClairA)

Summary

- Perfect SLA, irrespective of the input workload, is too expensive
  ➡ in terms of resource usage (as modeled by ClairA)

- To ensure resource efficiency, practical systems
  ➡ need to trade off some SLA compliance
  ➡ while managing client expectations

Summary

- Perfect SLA, irrespective of the input workload, is too expensive
  ➡ in terms of resource usage (as modeled by ClairA)

- To ensure resource efficiency, practical systems
  ➡ need to trade off some SLA compliance
  ➡ while managing client expectations

- Swayam strikes a good balance, for MLaaS prediction serving
  ➡ by realizing significant resource savings
  ➡ at the cost of occasional SLA violations

Summary

- Perfect SLA, irrespective of the input workload, is too expensive
  ➡ in terms of resource usage (as modeled by ClairA)
- To ensure resource efficiency, practical systems
  ➡ need to trade off some SLA compliance
  ➡ while managing client expectations
- Swayam strikes a good balance, for MLaaS prediction serving
  ➡ by realizing significant resource savings
  ➡ at the cost of occasional SLA violations
- Easy integration into any existing request-response architecture

Thank you. Questions?