Centralized Core-granular Scheduling for Serverless Functions

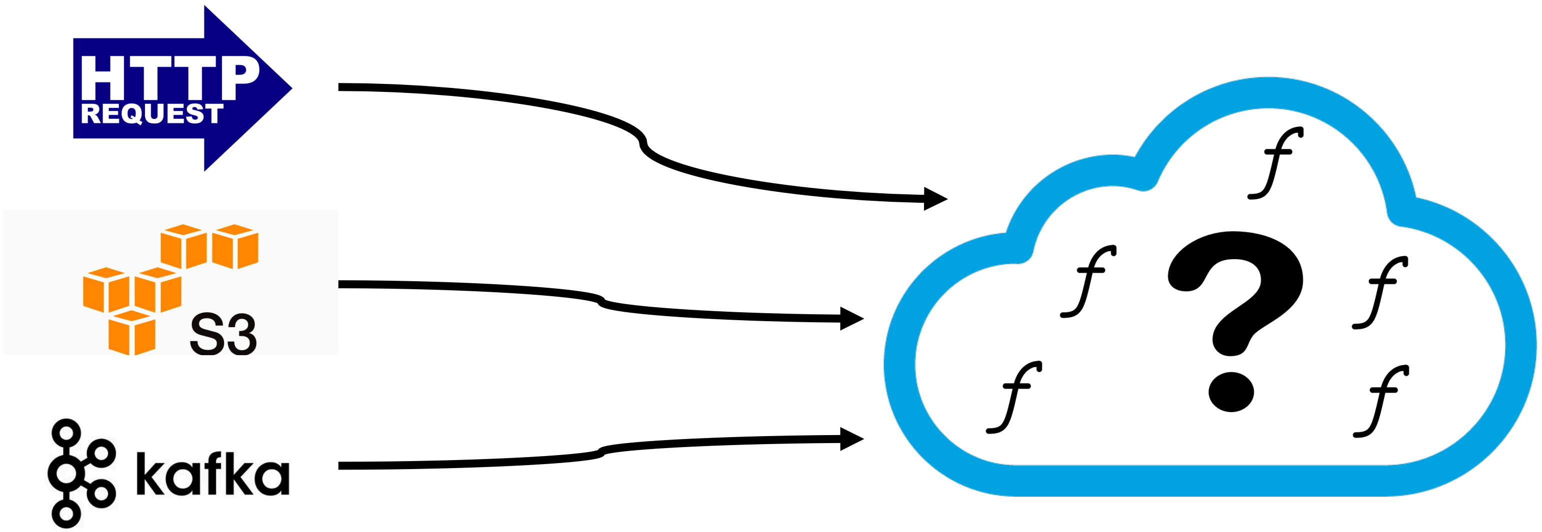
Kostis Kaffes, Neeraja J. Yadwadkar, Christos Kozyrakis

Stanford MAST

Serverless Computing is **Convenient** for **Users**

Users:

- Define a function
- Specify events as execution triggers
- Pay only for the actual runtime of the function activation



Ease-of-use has made Serverless Prevalent

User-facing apps

Data Analytics

PyWren
(SoCC '17)

ExCamera
(NSDI '17)

Sprocket
(SoCC '18)

Exotic Workloads

Compilation
gg (ATC '19)

NFV
(HotNets '18)

Serverless Functions' Characteristics

- Burstiness
  → Degree of parallelism can fluctuate wildly
- Short but highly-variable execution times
  → Execution times vary from ms to minutes
- Low or no intra-function parallelism
  → Each function runs on at most a couple of CPUs

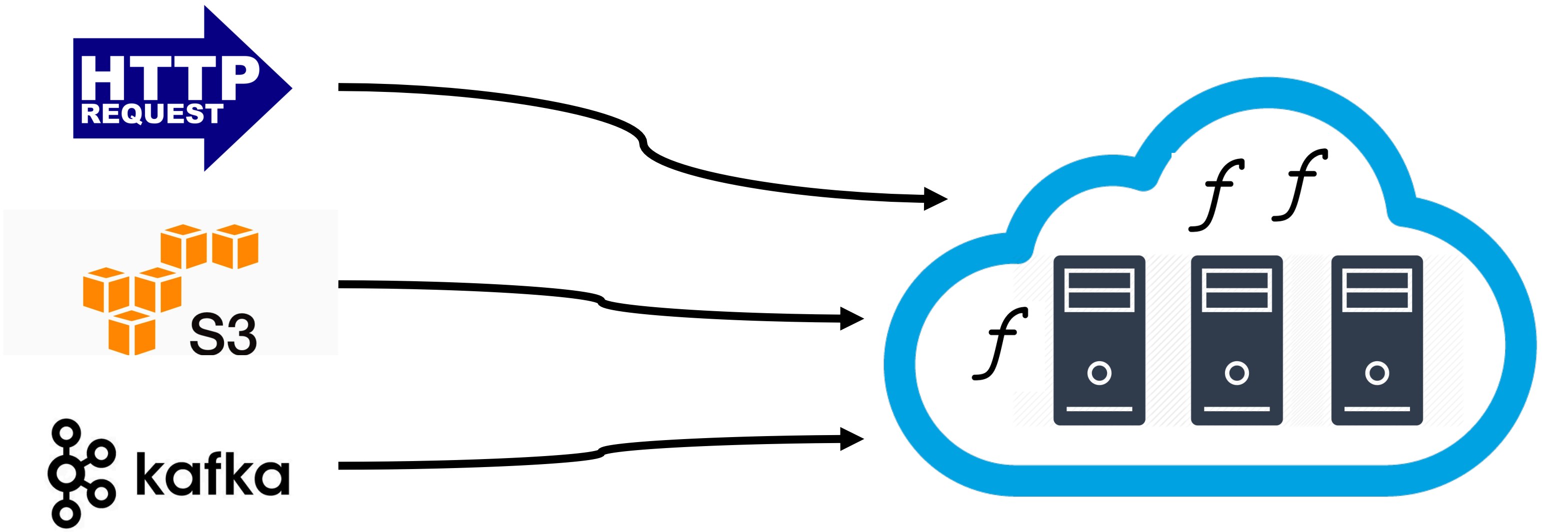
Serverless Systems' Performance Metrics

- Elasticity
  → Spawn a large number of functions in a short period of time
- Average and Tail Latency
  → User-facing workloads
  → High fan-out workloads
- Cost Efficiency

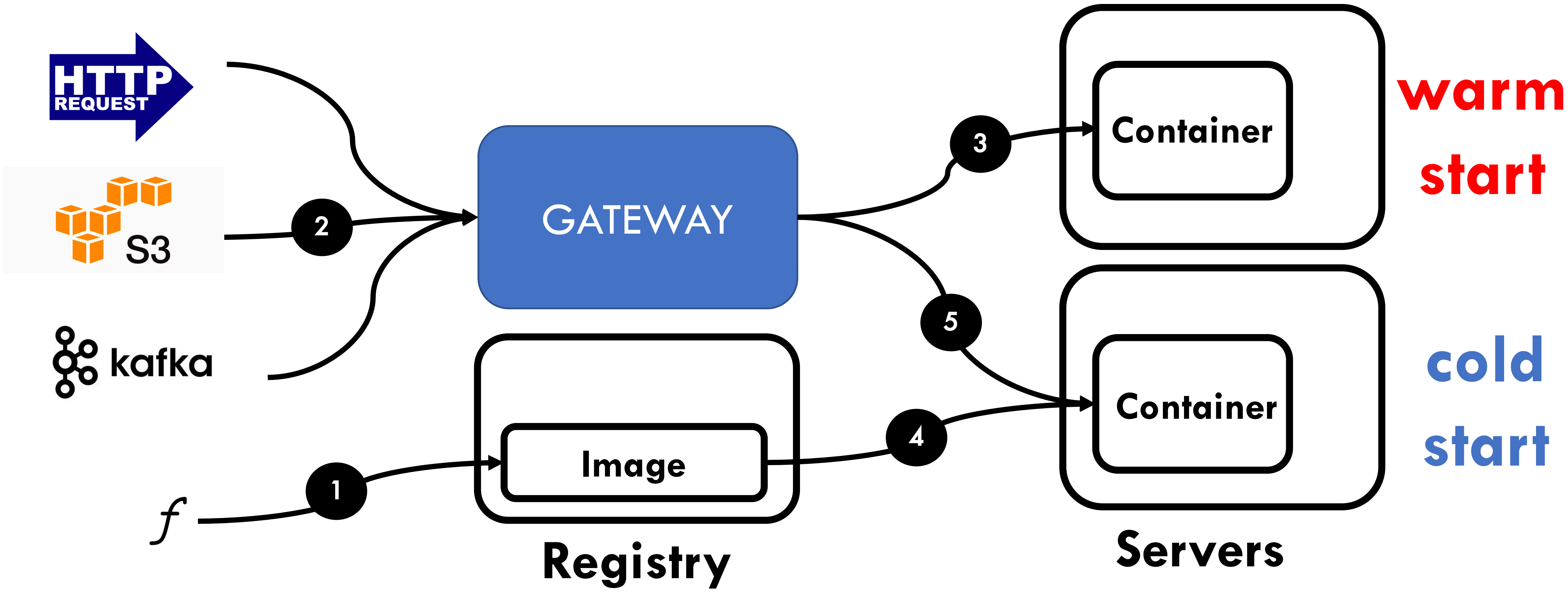
Serverless Computing is **Challenging** for **Providers**

Providers need to manage:

- Function placement
- Scaling
- Runtime Environment



Serverless Function Lifecycle



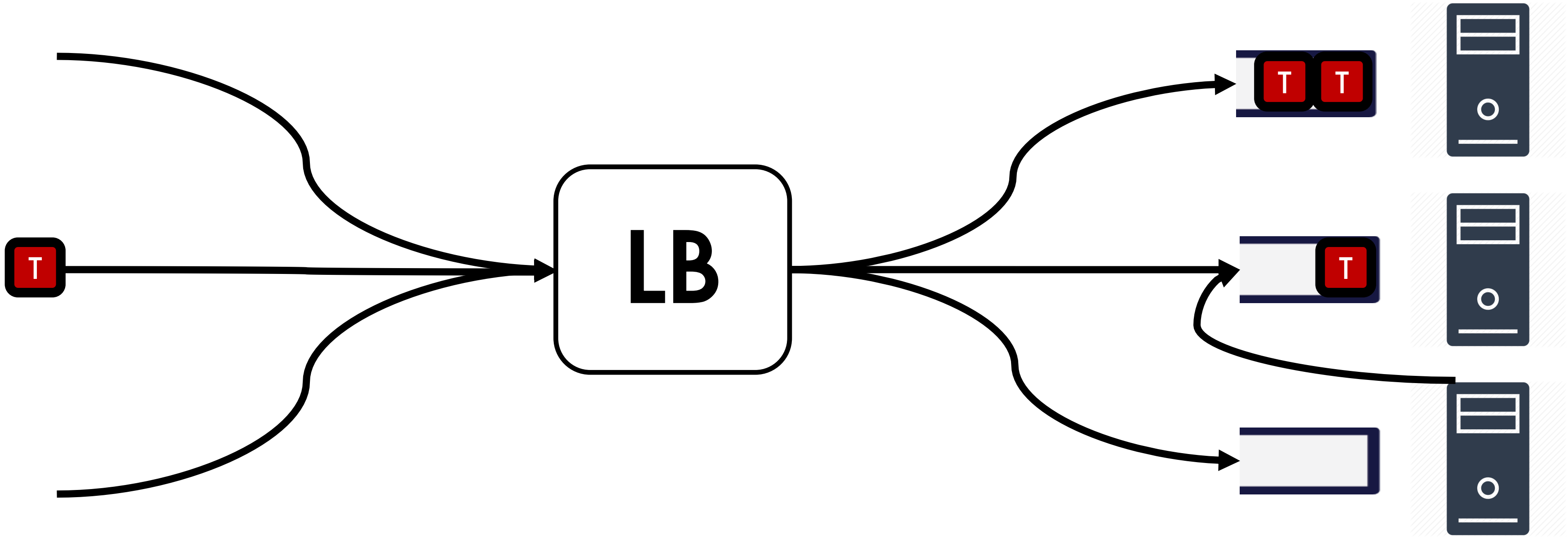
Different Approaches on Serverless Scheduling

- Task scheduling frameworks (Sparrow, Canary)
- Open-source serverless platforms (OpenFaas, Kubeless)
- Commercial serverless platforms (AWS Lambda, Azure Functions, Google Cloud Functions)

Option 1: Task Scheduling Frameworks

Two-level Scheduling:

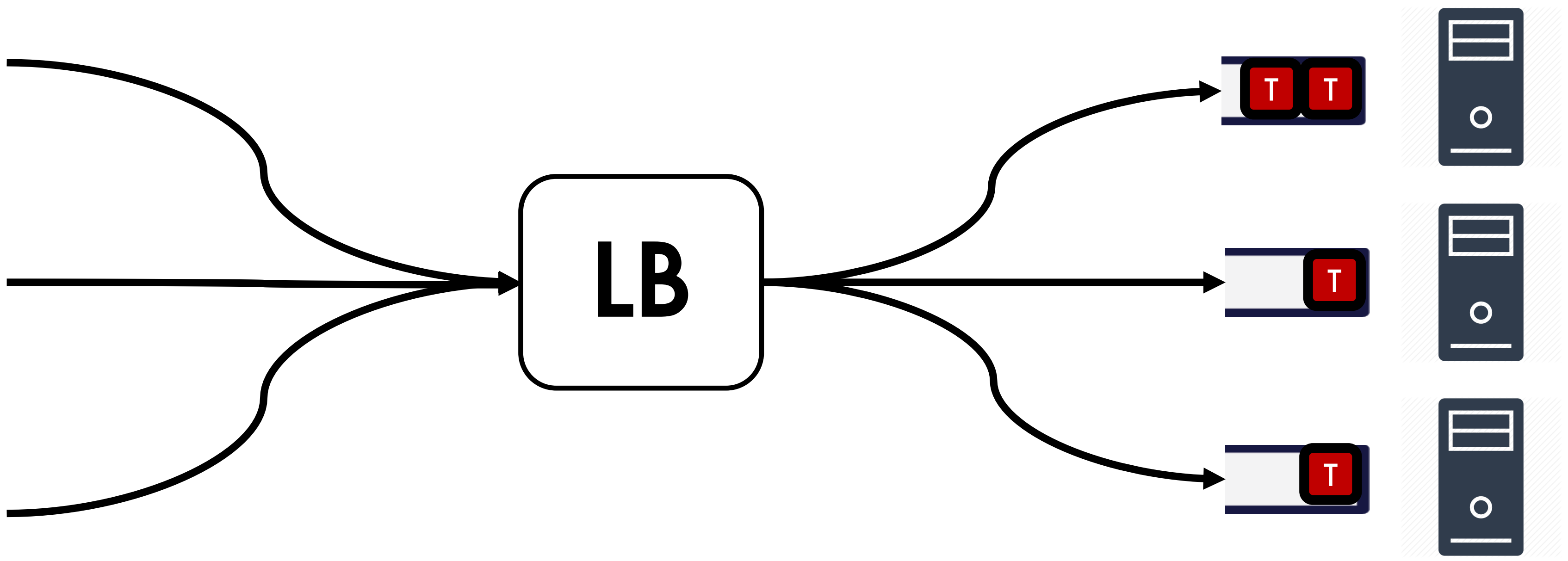
- Simple load-balancer assigns tasks to servers
- Per-machine agent detects imbalances and migrates tasks away from busy servers



Task Scheduling Frameworks' Problems

Such a design is unsuitable for serverless functions

- High variability → Queue imbalances → Frequent migrations
- High cold-start cost → Increased latency



Option 2: Open-source Serverless Schedulers

- Gateway receives functions invocations
- All container management is done by Kubernetes
- No migrations

→ Gateway Parameters

-- Scaling policy

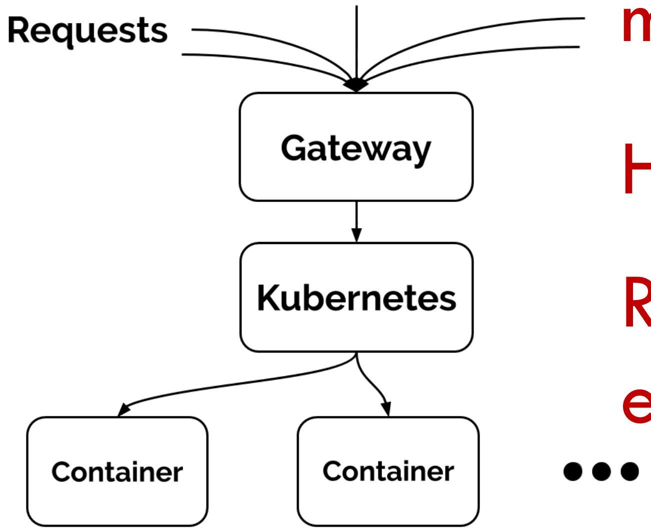
-- Max/min # instances

-- Timeouts

→ Kubernetes parameters

-- Container placement

-- ...



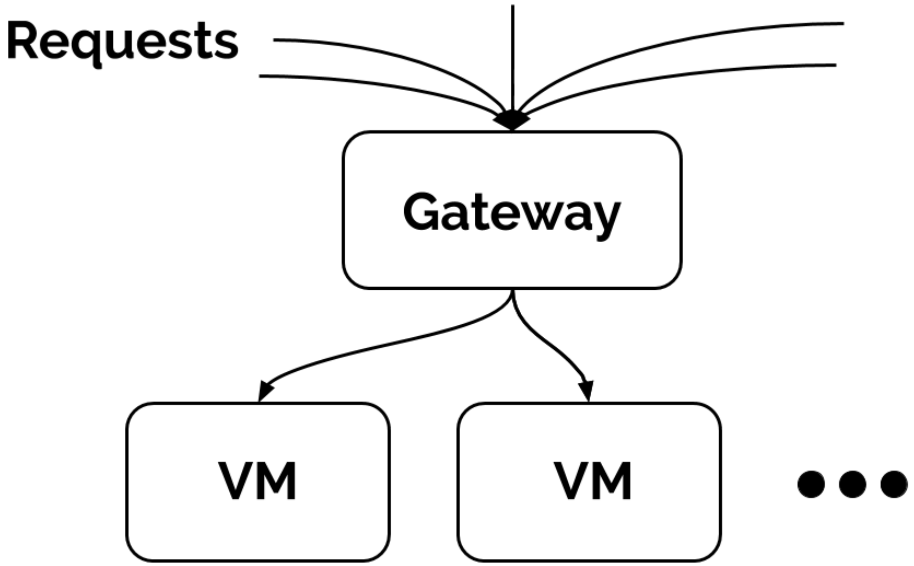
Scheduling split across multiple points

Hard to configure

Reduced elasticity and efficiency

Option 3: Commercial Serverless Schedulers

- Gateway packs containers running function invocations in VMs to improve utilization
- Once VM utilization exceeds some threshold, it spins up more VMs in different servers



Opaque policies and decisions

+

Function packing

=

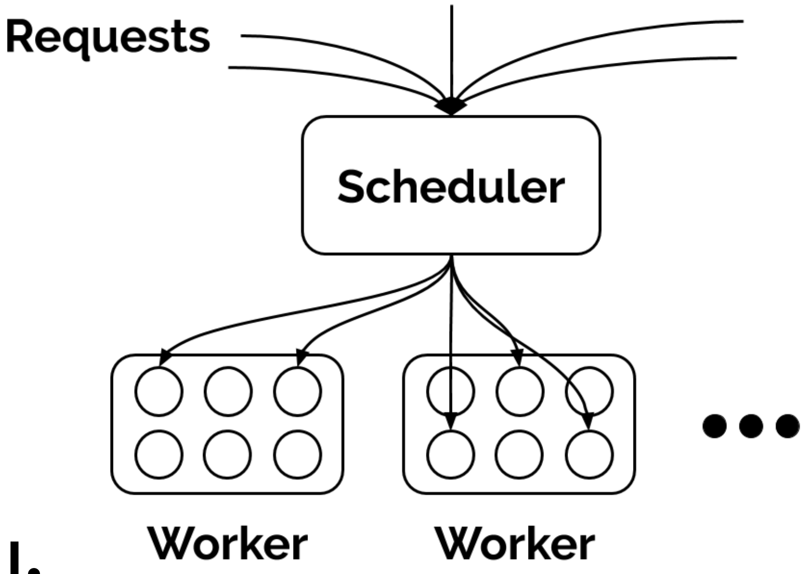
Unpredictable performance

How can we avoid existing schedulers' problems?

Problem: High variability leading to imbalances and queueing
Solution: Centralized Scheduling and Queueing

Problem: Hard or impossible to configure
Problem: Coarse-scale scheduling can cause interference
Solution: Core-Granular Scheduling

Centralized and Core-granular Scheduling



Visibility of all available cores:

- Less queueing
- Lower latency
- Higher elasticity

Fine-grain interference/utilization control:

- Pack many function instances together to maximize efficiency
- Reduce interference by placing one function per core

Opportunity 1: Inter-function Communication

Serverless workloads create data that need to be transferred between function instances

Now: Data shared through a common data store

Ideal: Direct function-to-function communication
→ Naming, addressing, and discovery through the centralized scheduler
→ Avoids an unnecessary data transfer and reduces cost

Opportunity 2: Core Specialization

Centralized scheduler can keep a list of “warm” cores for:

- Specific functions
- Different language runtimes (Python, Javascript, etc.)
- Different libraries and frameworks (numpy, scikit-learn)

and reduce cold start time

Opportunity 3: "Smarter" Policies

The scheduler has full visibility on the cluster state

It can use or **learn** better policies regarding:

- Container re-use
- Scaling
- Function packing
- …

Conclusion

Centralized and **core-granular** scheduling can enable:
→Better elasticity
→Lower latency
→Higher efficiency

It also provides exciting opportunities for future research:
→Inter-function communication
→Core Specialization
→"Smarter" Policies

Backup

Detailed Implementation

i. **Request arrives to a scheduler core**
ii. **Dequeue worker core**
iii. **Schedule request to worker core**
iv. **Enqueue worker core**
v. **Request arrives to scheduler core with empty worker core list**
vi. **Steal worker core from different queue**
vii. **Schedule request to worker core**