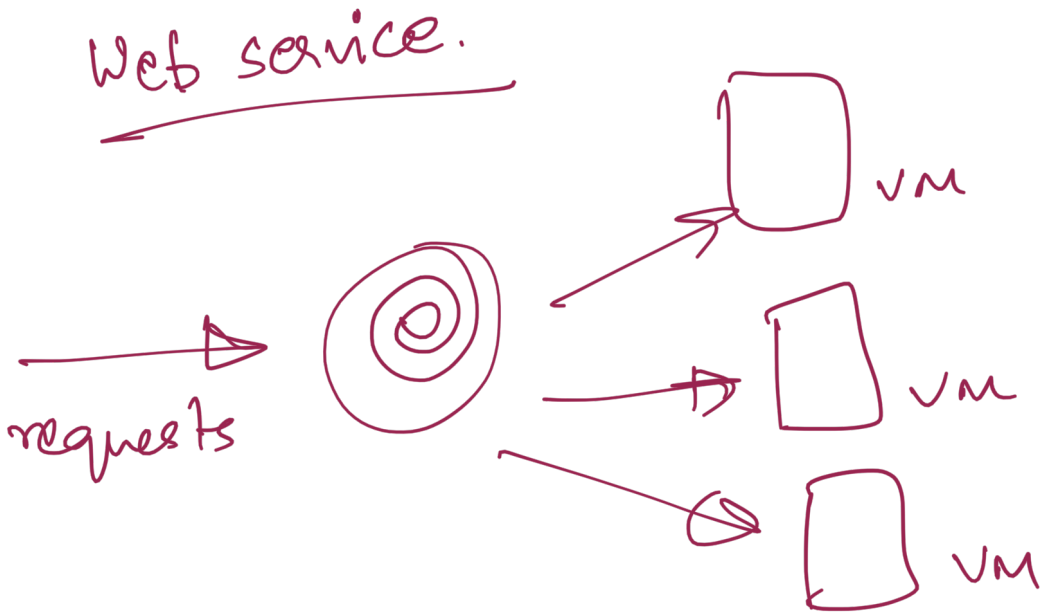
FaaS to FaaSter
via Scalable Serverless Infrastructure



Purushottam (Puru) Kulkarni

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

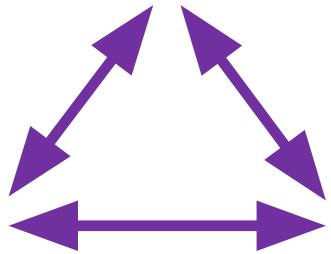
The Cloud Services Story

Infrastructure
IaaS

Platform
PaaS

Software
SaaS

Function
FaaS



Software-defined
techniques

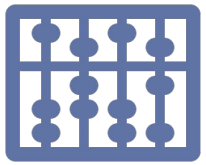
VMs/Containers, HCI,
SDWAN, SDN, VNF,
SAN, NAS, Data stores
User space protocol stacks
,,,

Hardware assistance
& capabilities

NVM, NVMeOF, RDMA,
SRIOV, GPUDirect,
Unified addressing
SmartNICs, VPP
…

The FaaS promise

Remove complexity of provisioning and management of cloud resources

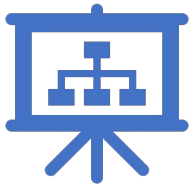


Function granularity

1-1 mapping of service, provisioning and billing

Serverless!

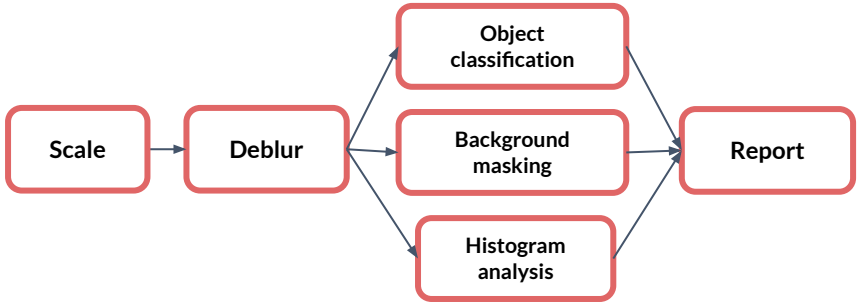
No infrastructure management (for clients)



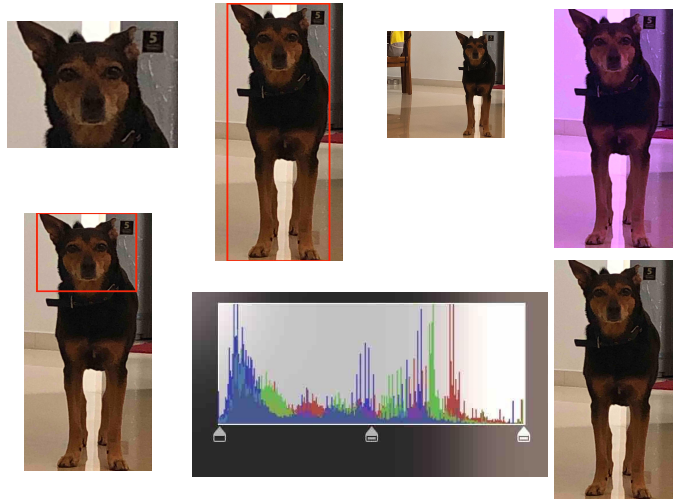
Agile development

Flexible and dynamic composition

FaaS in action



Composition pipeline
sequential, parallel, branching

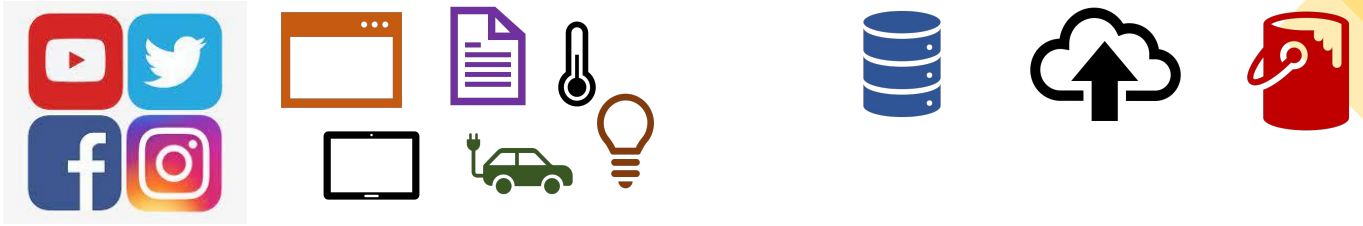


Applications	Functions
Social network Media service E-commerce website Banking system Video analytics ML tasks Network functions	urlshorten followers userinfo postsStorage userprofile ads thumbnail getrating getreviews recommender discounts catalogue castinfo wishlist payment balance contactinfo imagerecognition classification speed location luminosity resize histanalysis encrypt decrypt packetanalysis rulematching

An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems ASPLOS 2019 (aka the DeathStar Benchmark Suite)

The FaaS game plan

- **Decouple the what, the when and the where**
- Data generation & storage



- Triggers & events
-

AWS SNS

AWS SQS

HTTP

GCF PubSub

Storage trigger

- Execution platforms

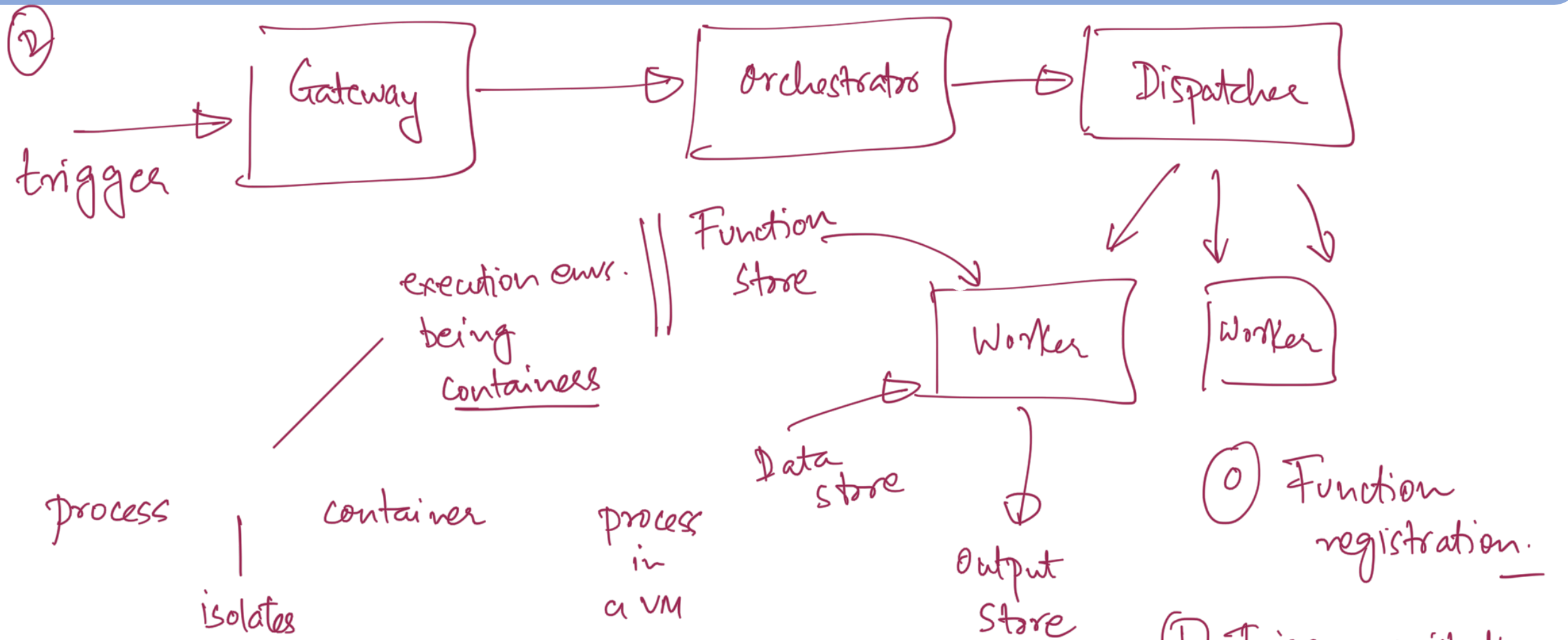
AWS Lambda

Azure Functions

Apache OpenWhisk

Google Cloud Functions

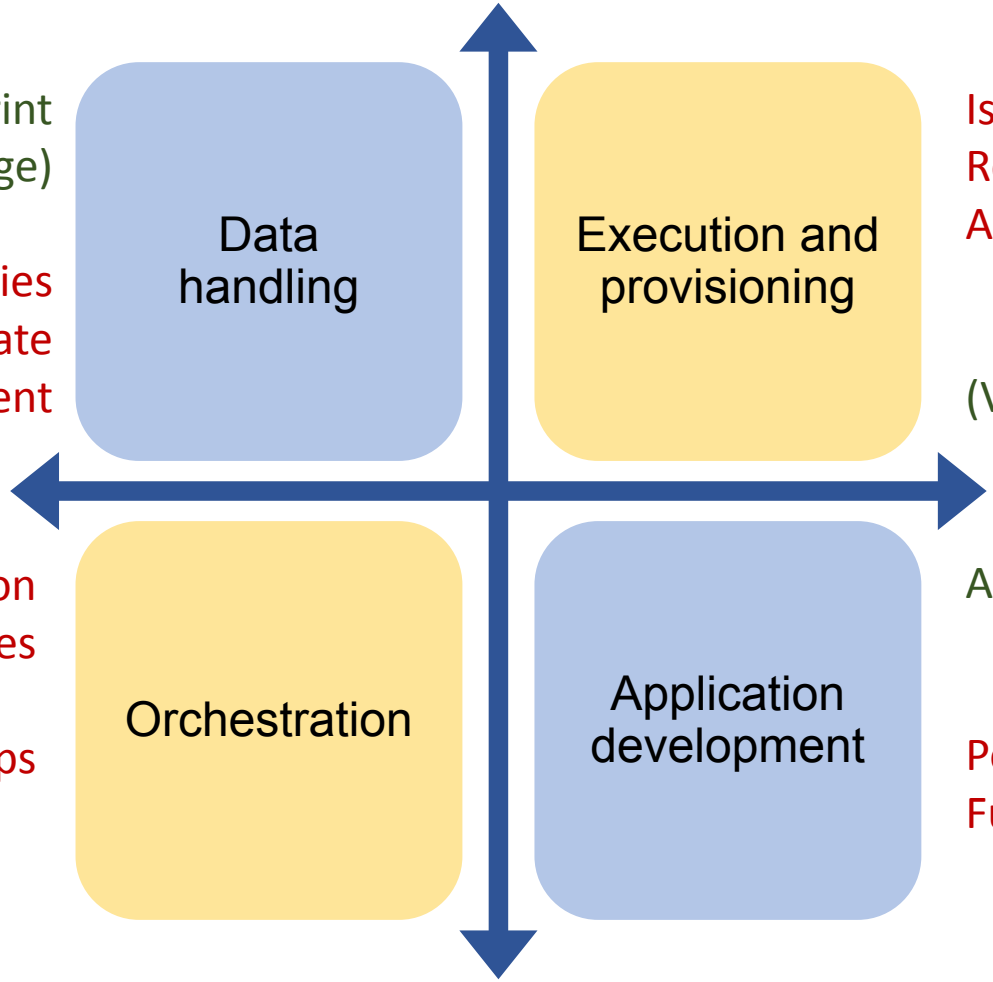
Ex. serverless platform / arch.



Challenges

Stateless and low memory footprint
(S3, Object storage)

Data dependencies
Global shared state
Data staging and movement



Isolation vs. Performance
Resources vs. Startup times
Abstractions vs. Resource efficiency

(VMs, Containers, Isolates, Lambdas)

Synchronization
Control flow dependencies

Multi-cloud setups

Architect applications from the ground up

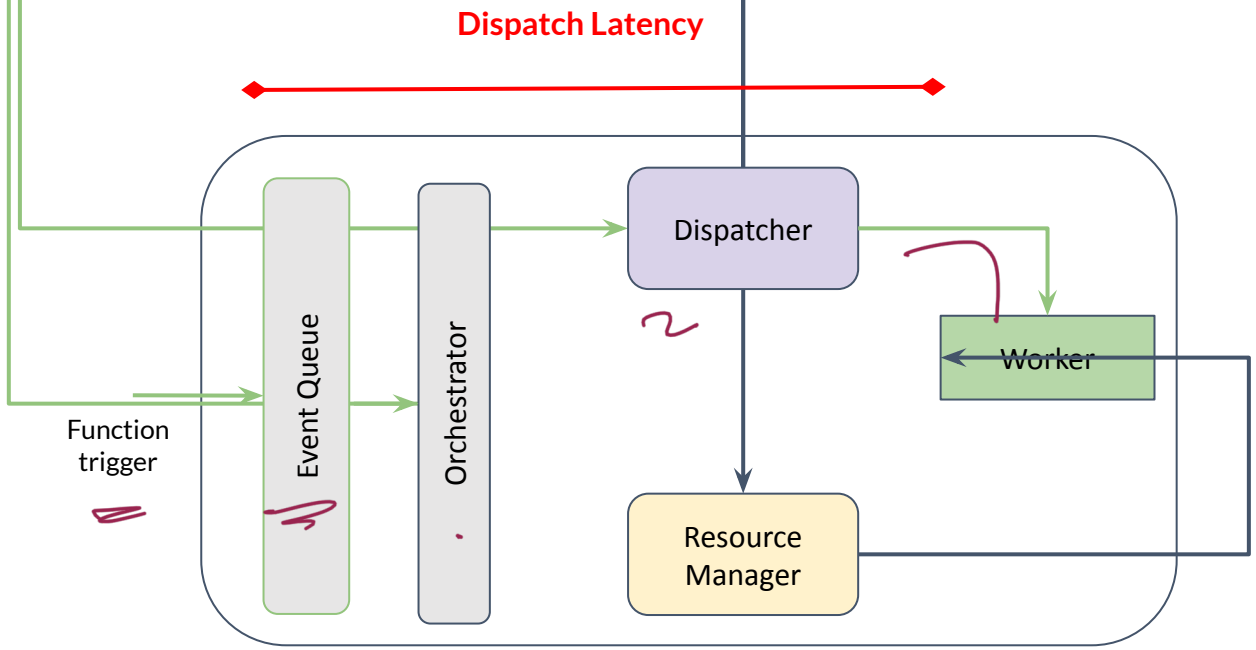
Porting of legacy systems
Function abstraction and interfaces

Two example problems

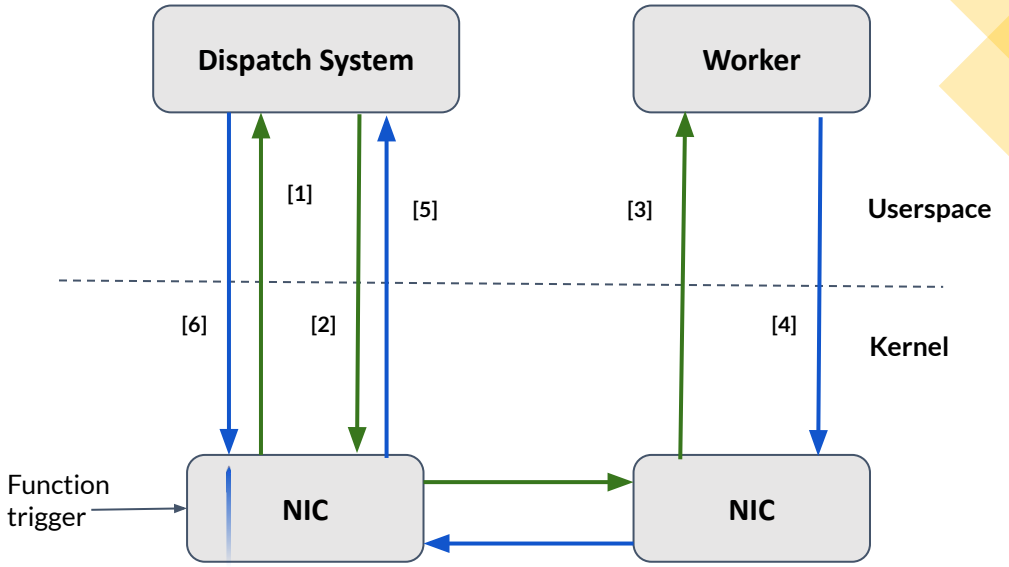
Scalable orchestration of serverless workflows

Acceleration Functions -as-a-Service

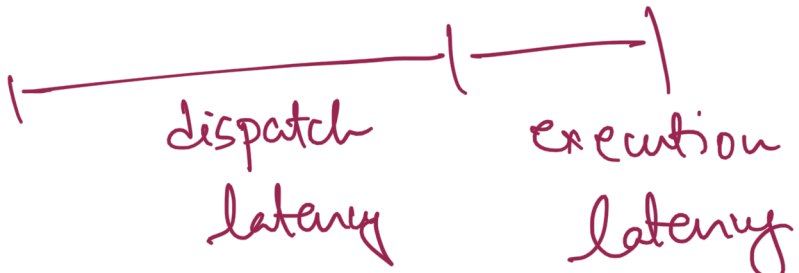
Scalable orchestration of serverless workflows



Latency aggregates as FaaS components are executed for each function in workflow



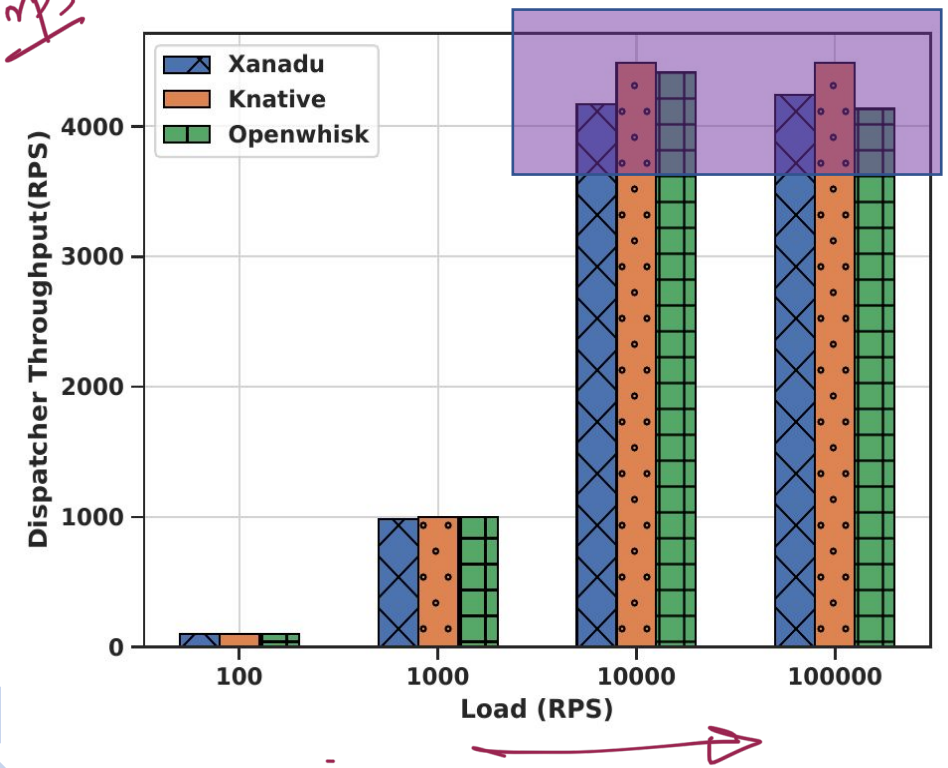
~6 network stack traversals per trigger!

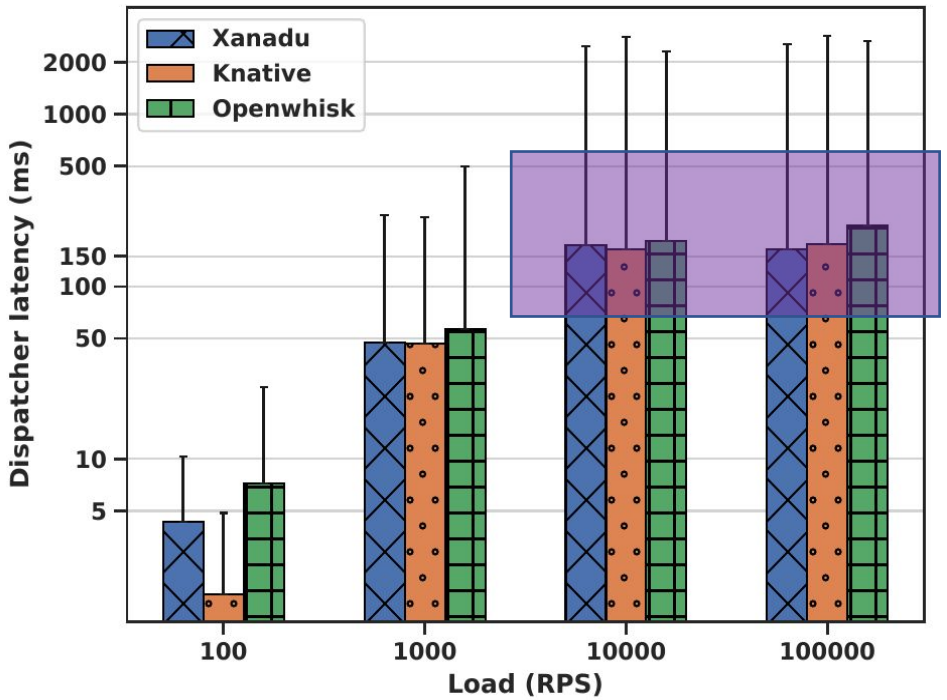


Speedo: Fast dispatch and orchestration of serverless workflows
Nilanjan Daw, Umesh Bellur, Purushottam Kulkarni
ACM Symposium on Cloud Computing (SoCC '21), November 1–4, 2021

The dispatch bottleneck

- Social network workflow: **~6K posts** per second => **~70K** triggers per second

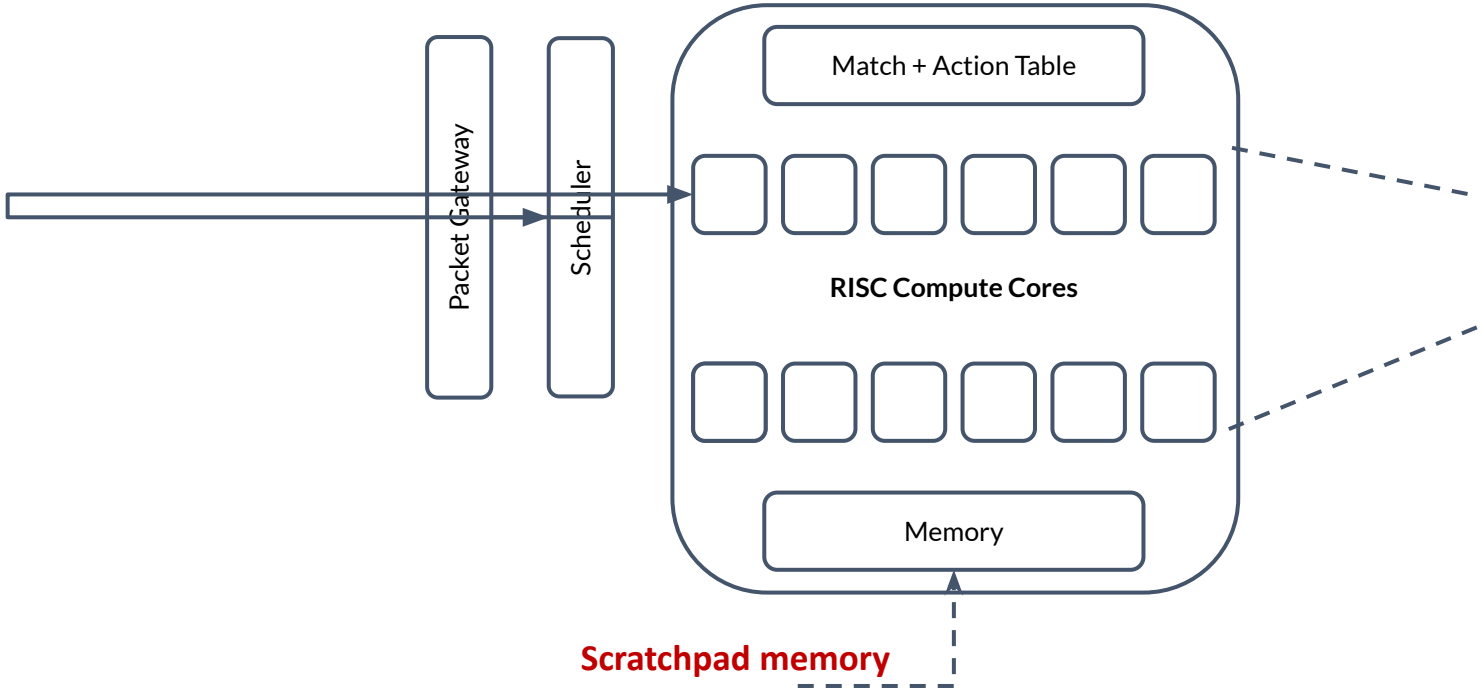




Userspace dispatchers saturate at moderate loads (with high dispatch latency)

Compute at the Edge!

Modern networking hardware (SmartNICs)



Programmable in-network compute

High degree of parallelism

Offload FaaS dispatch system onto SmartNICs for decisions closer to the network !

Speedo key ideas

Exploit in-network compute and programmability for FaaS dispatch

#1

Network stack traversal is costly

Observation

Dispatch decisions closer to the wire improves latency

Solution

Offload the dispatcher to the NIC

#2

Userspace parallelism is non-deterministic and costly

Observation

RISC cores on SmartNICs provide scalability

Solution

Exploit hardware parallelism on SmartNICs for scalability

#3

Dispatcher is not stateless

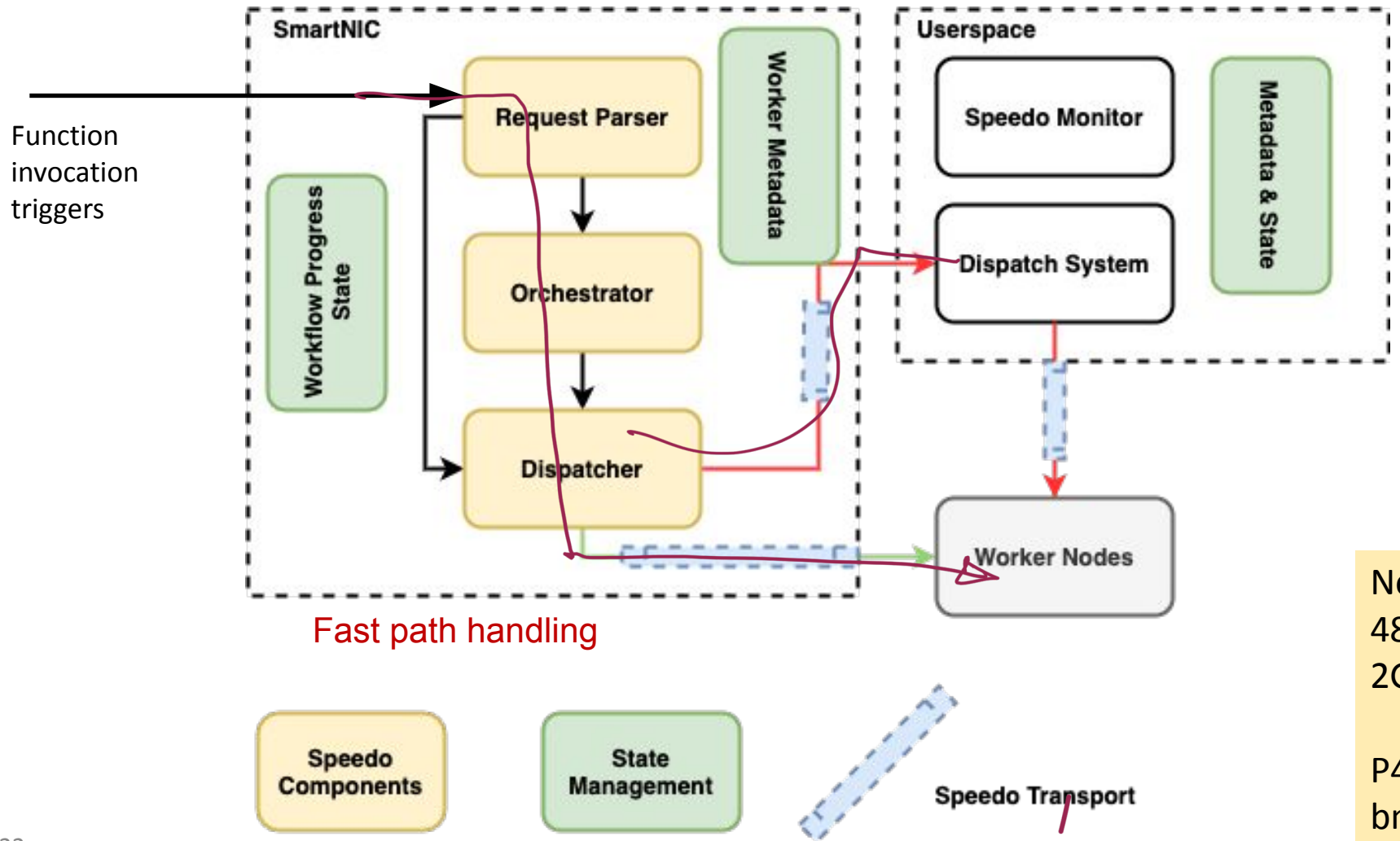
Observation

Lifetime of workflow state is ephemeral and short lived

Solution

Use NIC onboard memory to store workflow state

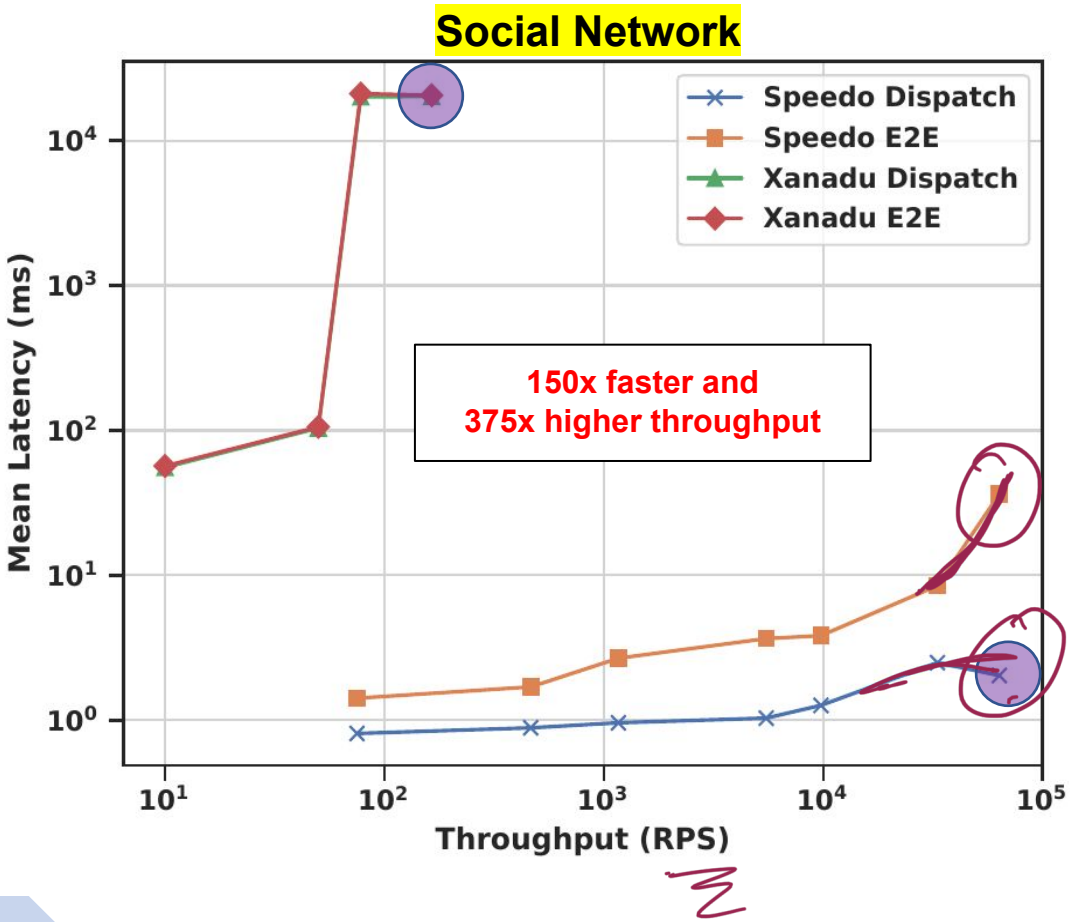
Speedo Architecture

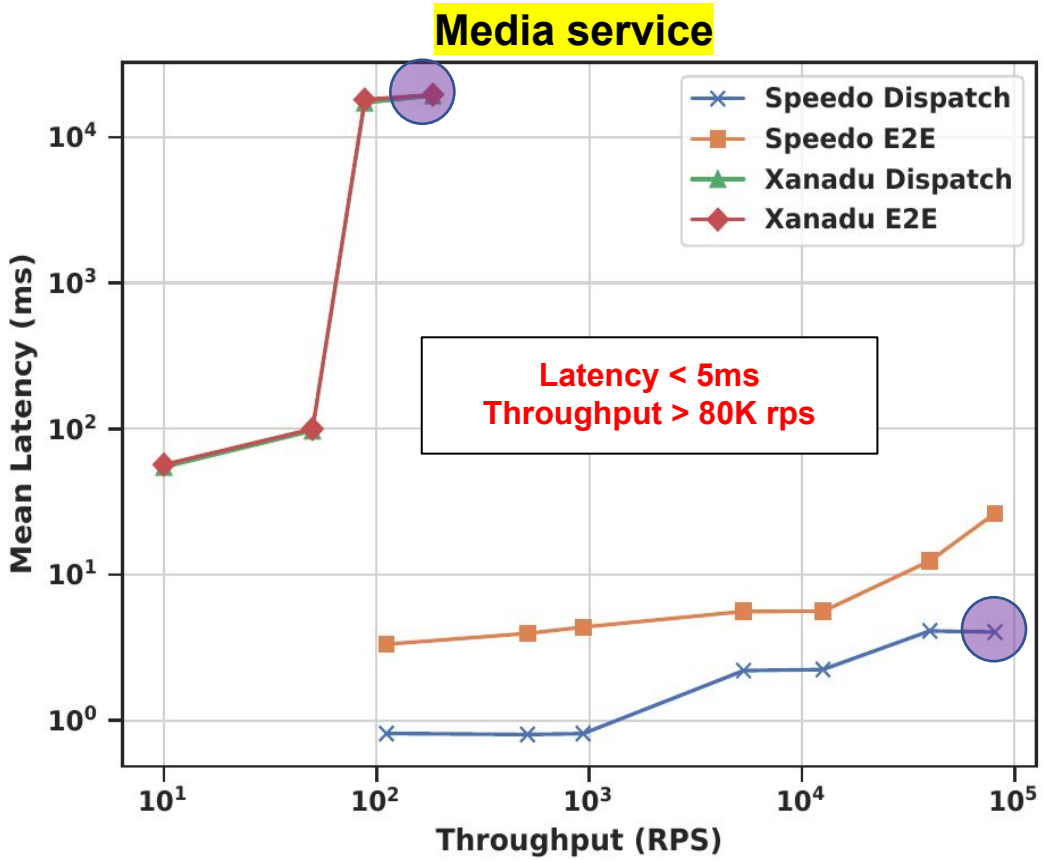


Netronome Agilio NIC
48 x 8 execution instances
2GB on-board memory

P4-compliant
bmv2 software switch

Speedo Performance





- Speedo saturates server worker capacity without saturating dispatch system

Speedo Summary

- Speedo exploited compute and programmability of SmartNICs

  to improve FaaS management components --- dispatcher and orchestrator

  177x improvement in dispatch throughput (compared to userspace dispatch)

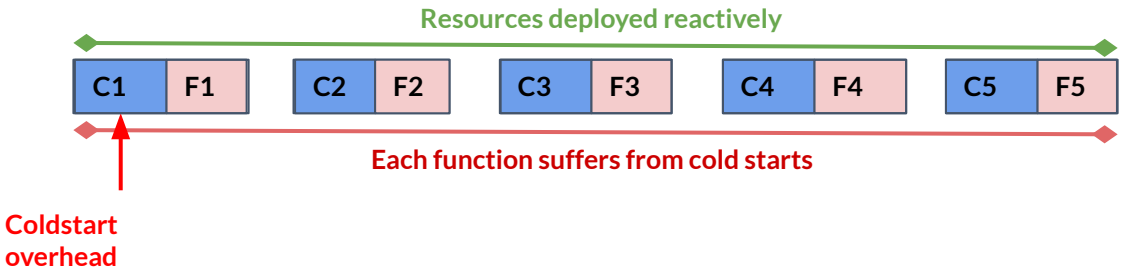
  50x average reduction in dispatch latency

- **Future work**

  Dynamic offloading of compute between host CPU and SmartNIC

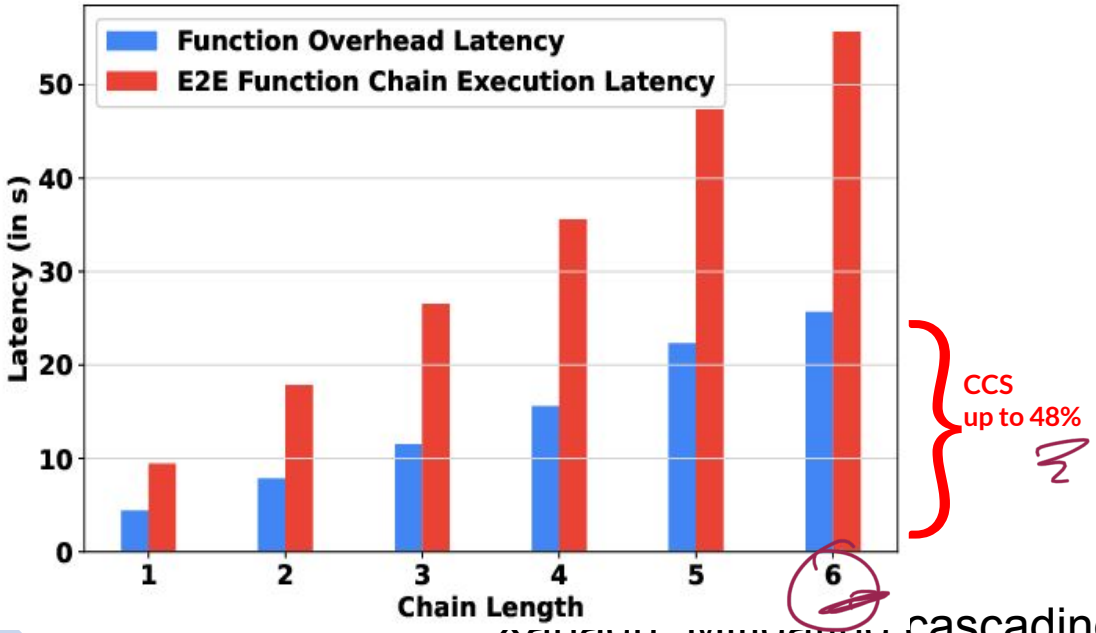
  SmartNIC resource partitioning across applications

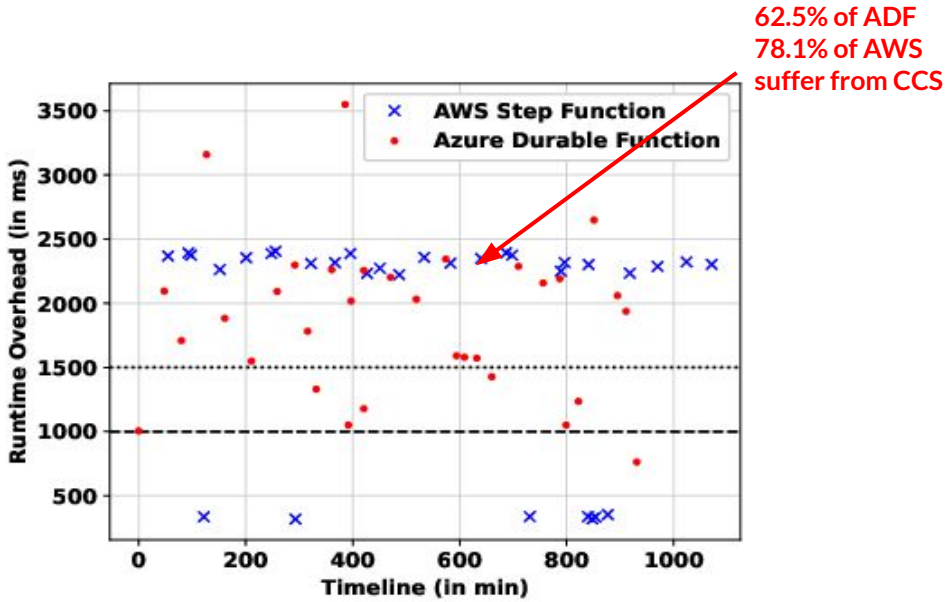
Addressing the Cascading Cold start (CCS) problem



E2E latency breakup

Cold start overhead C1+C2+C3+C4+C5	Function execution times





Xanadu: Mitigating cascading cold starts in serverless function chain deployments

Nilanjan Daw, Umesh Bellur, Purushottam Kulkarni

21st ACM/IFIP International Middleware Conference, December 2020

Xanadu key ideas

#1
Cascading cold starts cause performance degradation

Observation
Pre-deploying resources can prevent cascading overheads

Solution
Speculatively pre-deploy resources

#2
Cost of speculative deployment still significant

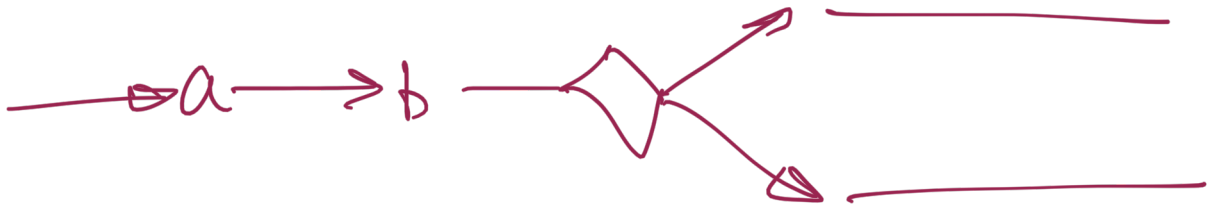
Observation
Application runtime tends to remain stable

Solution
Deploy resources Just-in-time to reduce resource costs

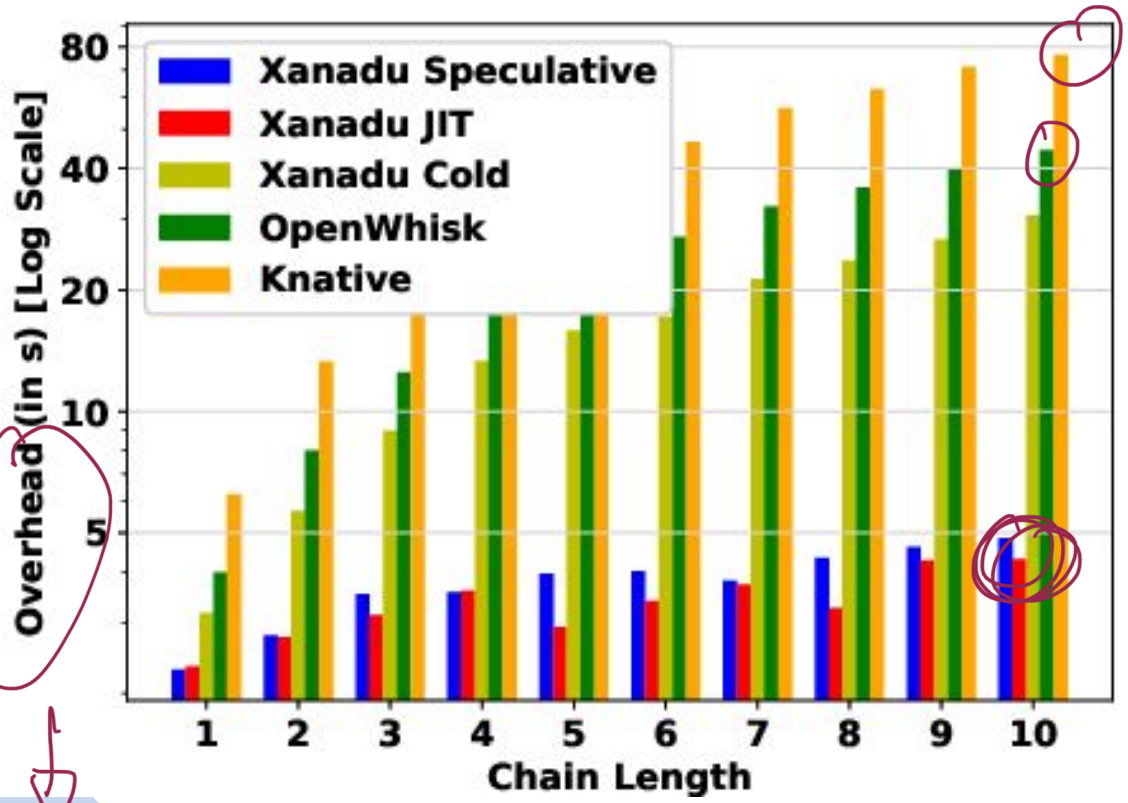
#3
Naive pre-deployment is resource expensive

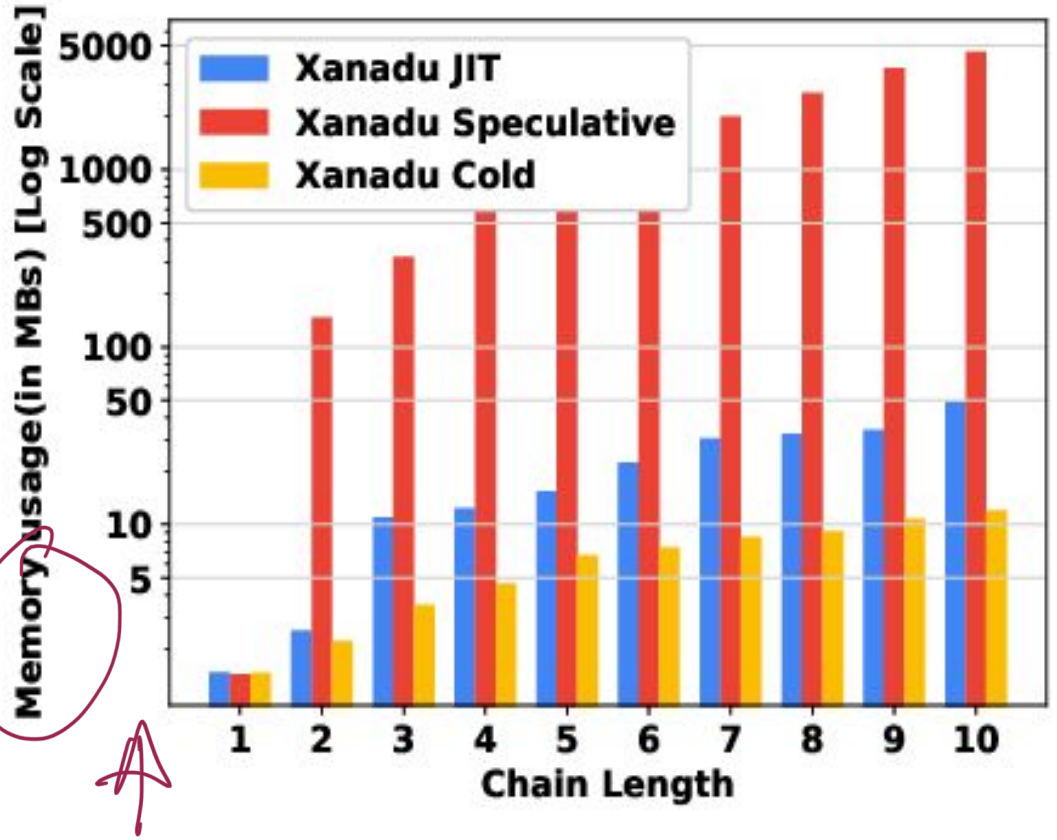
Observation
Application behavior tends to follow a stable path

Solution
Detect the Most Likely Path (MLP) for speculative deployment



JIT + MLE advantages





Xanadu Summary

- Cold start latency mitigation intersects several dimensions
  - Execution platforms (VMs, containers, isolates, lamdas)
  - Resources reservation vs. Costs
  - Proactive vs. Reactive management

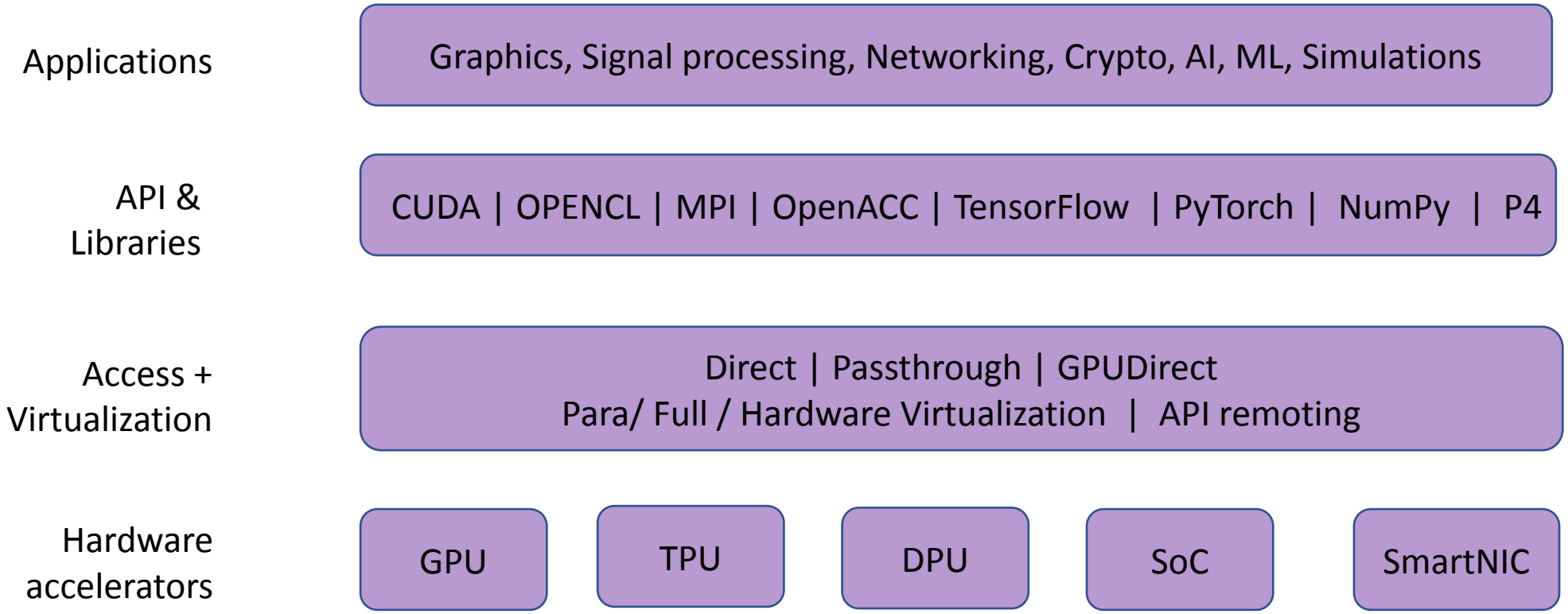
  - **Data provisioning (on the network)**
    - Standardized data pipelines need of the hour!

Two example problems

Scalable orchestration of serverless workflows

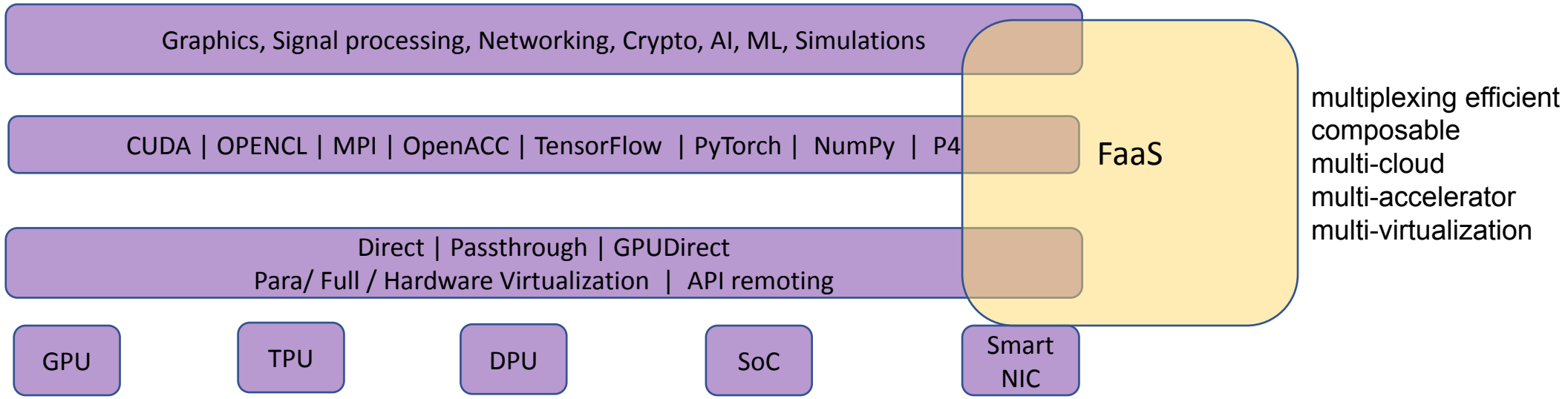
Acceleration Functions -as-a-Service

The acceleration spectrum



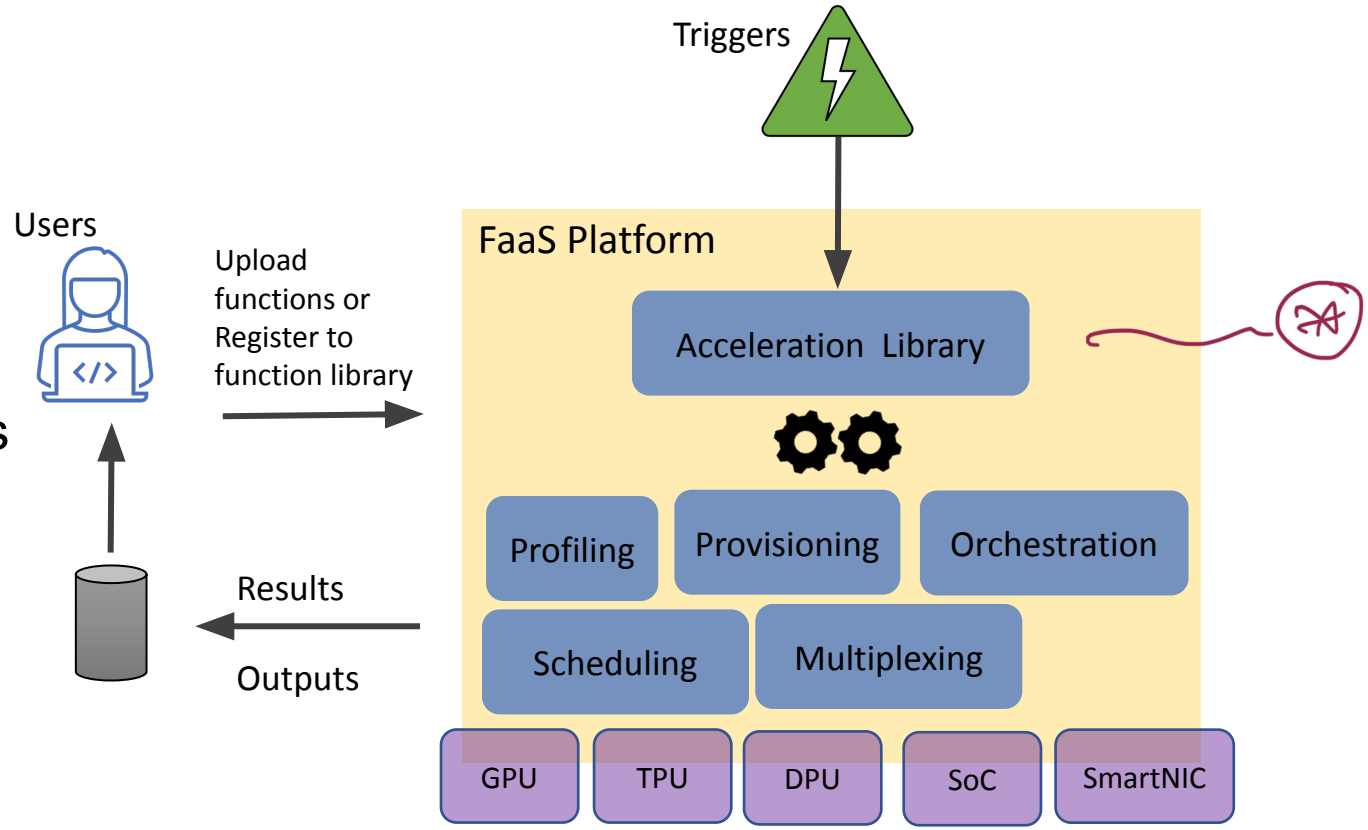
Challenges with Accelerators

- Exposure as a resource, needs programmatic control and usage
- Reserve and use models are not efficient
- Preemption and multiplexing not first-class primitives
- Accelerator interfaces not generic and lead to accelerator stack lock-in



Accelerated Functions-as-a-Service

- The ***function*** abstraction
  - Serverless, composition, …
- Multi-accelerator backends
- Transparent elastic resource pools
- Suite of optimizations for improved resource usage efficiency

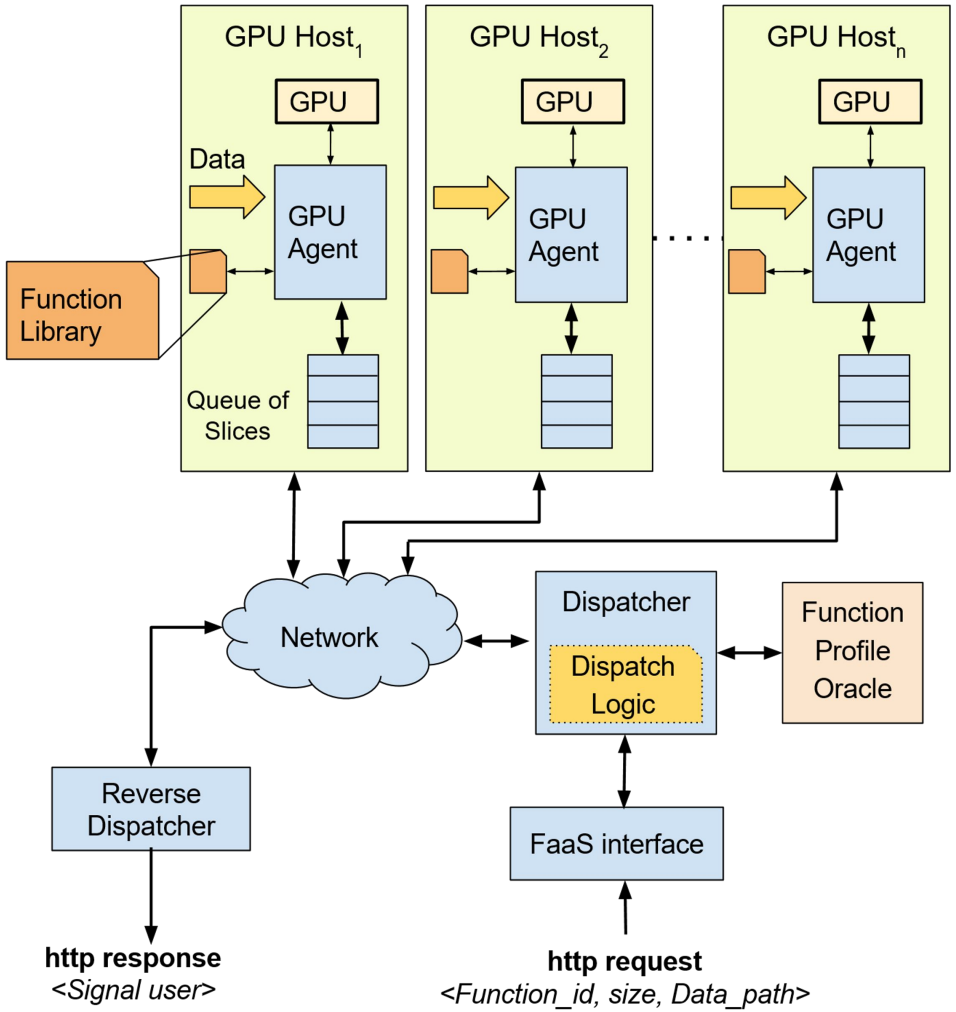


AFaaS with Heterogeneous GPUs

- **Issues**
  - GPU functions run to completion
  - Very limited/no control on scheduling
  - Multi-GPU parallelism per task missing
- **Key enabler**
  - GPU kernel slicing
- **Solution components**
  - #slices and #GPUs for each function
  - which types of GPUs, and #GPUs per type?
  - online heterogeneous GPU scheduling

Goals:

Matching task types with appropriate accelerators
Minimize average turn-around time of functions
Priority based allocation of acceleration services



Does it all add up?

Slicing-based and accelerator appropriate mapping improved resource efficiency and performance

Better than or equal to No slicing approach under various conditions