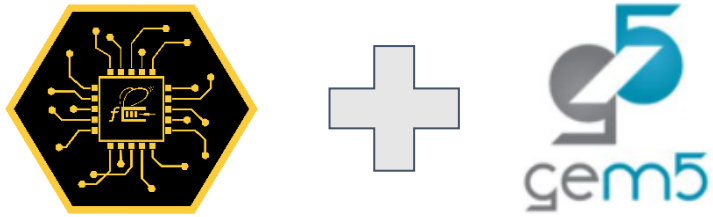
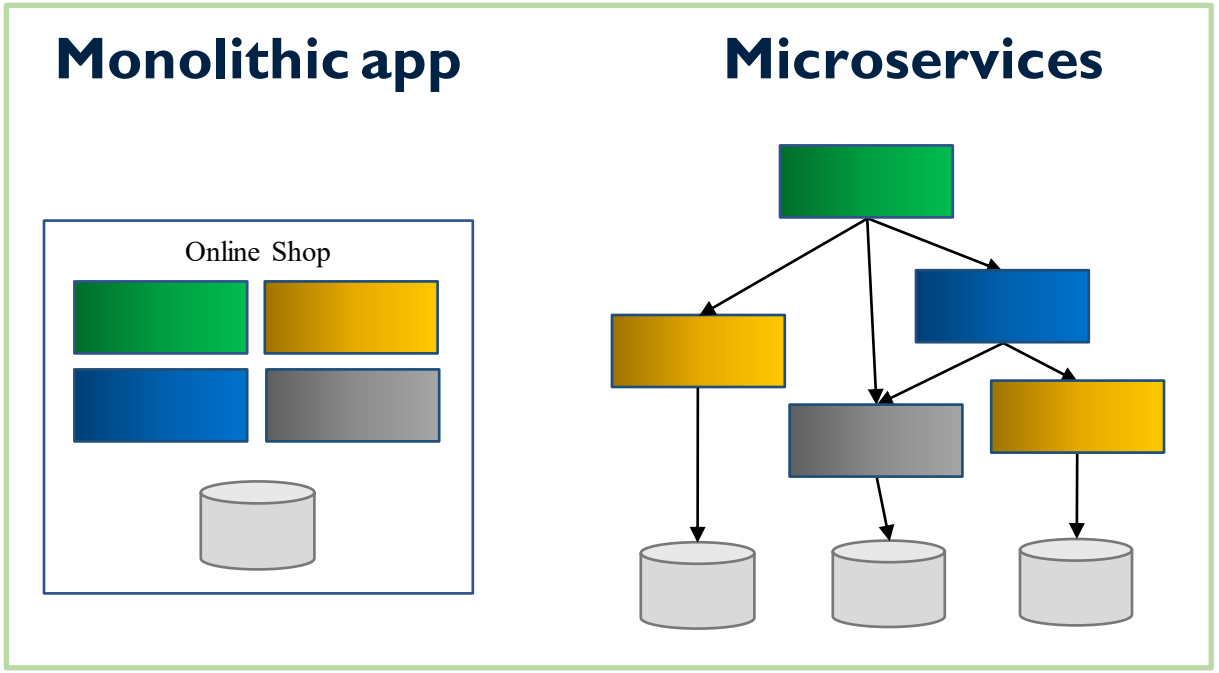
vSwarm-μ: Microarchitectural Research for Serverless

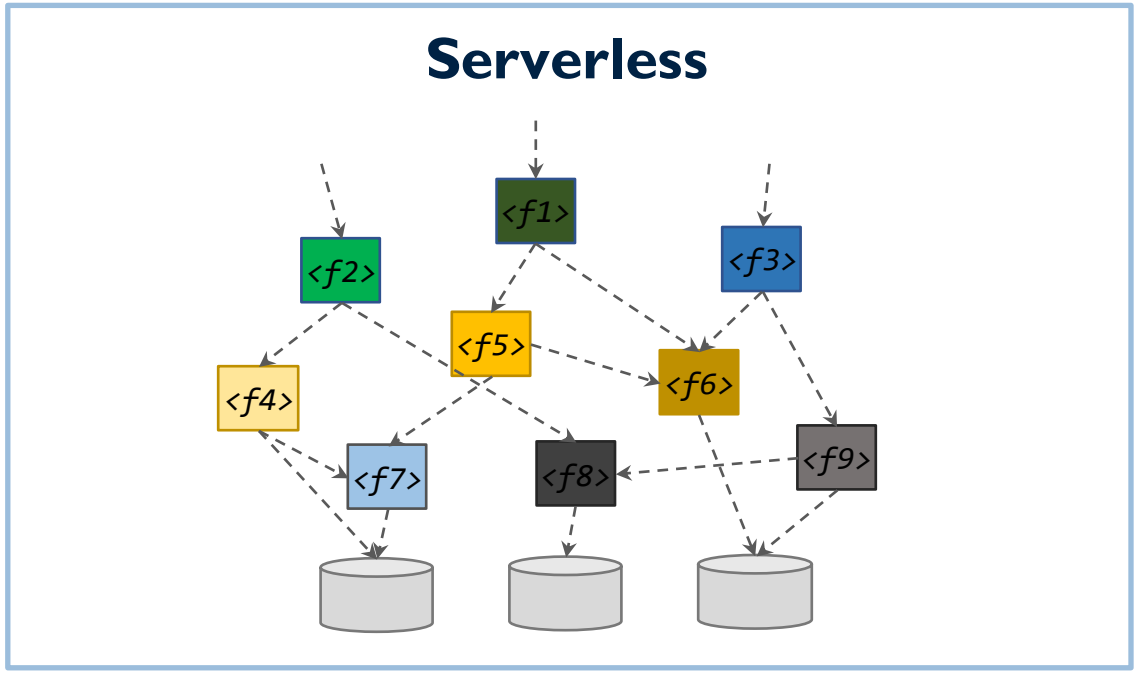
David Schall

University of Edinburgh



Cloud Applications: from Monoliths to Serverless





Conventional cloud deployments:

- Virtual machines that stay up for long periods of time
- User is billed even when the service is idle

Serverless cloud deployments:

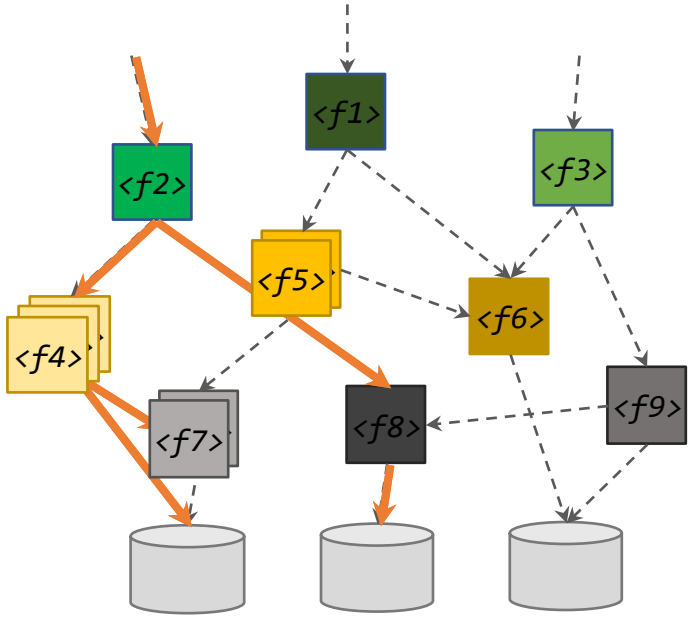
- Functions are invoked on-demand
- No invocations → no cost
- > 50% of cloud customers use serverless [Datadog 2022]

Serverless is big … and growing!

Serverless Basics

Datacenter application organized as a collection of stateless functions

- Functions invoked on-demand
  - via triggers (e.g., user click) or by another function
- Functions are stateless: facilitates on-demand **scaling down to zero**
  - Zero is not possible for monoliths & microservices
- Developers: pay only per invocation (CPU + memory), not idle time
  - Key difference from monoliths & microservices!
  - Financial incentive to reduce function footprint
- Cloud providers: high density and utilization at the server level



Workloads are changing

Serverless on a Server

Serverless represents a new class of workloads

- **Unique characteristics** and **challenges**
  - Short function execution times: a few ms or less
  - Small memory footprint
  - Sporadically invoked (seconds or minutes)

Serverless runs on CPUs designed for **conventional** workloads

- Problematic: **Inefficient (lukewarm) execution**

  Session 9A: Lukewarm Serverless Functions: Characterization and Optimization

→ High demand of **microarchitectural support** for emerging workloads

gem5 is designed to study **conventional** workloads

- Limited support for serverless software stack in the reference setup
- Workload highly depend on server-client communication

→ Incomplete and misleading simulation results

Lukewarm Serverless Functions: Characterization and Optimization

David Schall
d.h.schall@sms.ed.ac.uk
University of Edinburgh
Edinburgh, United Kingdom

Artemiy Margaritov*
artemiy.margaritov@huawei.com
Turing Core, Huawei 2012 Labs
Edinburgh, United Kingdom

Dmitrii Ustiugov*
dmitrii.ustiugov@ed.ac.uk
ETH Zurich
Zurich, Switzerland

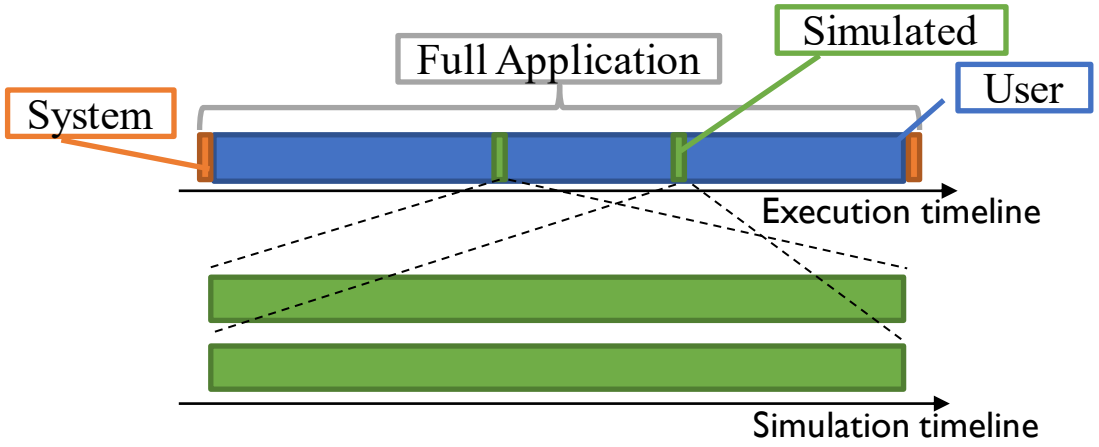
Andreas Sandberg
andreas.sandberg@arm.com
Arm Research
Cambridge, United Kingdom

Boris Grot
boris.grot@ed.ac.uk
University of Edinburgh
Edinburgh, United Kingdom

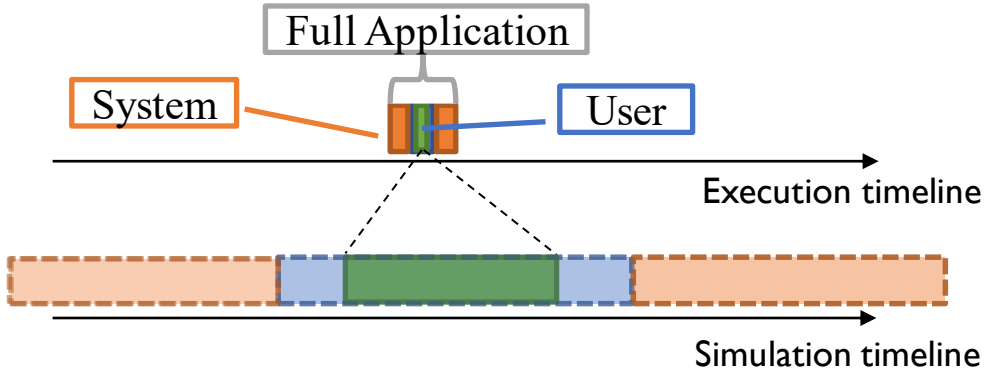
What are the limitations of gem5?

1. Challenge of Serverless Host Simulations

Conventional workloads



Serverless workloads



- **Long running** applications
  - Challenge: Infeasible simulation times
    - Cannot simulate full application in cycle accurate mode.
  - Solution: Focus on application (what matters)
    - Simplified software stack in favour of simulation speed
    - System stack components are negligible 🙂
- **Good support** from gem5
  - SimPoints, regions of interest, snapshots,…

- **Short execution times** of serverless functions
  - Function can be simulated in full 🙂
  - **Significant fraction of execution** spent in system stack
- **Key layers** are **not supported** by gem5 out of the box
  - Isolation: Containerization, virtualization ☹
  - Key layers in the communication stack are simplified

gem5 must support the full system stack

Enabling Serverless on gem5

Serverless function consist of:

- Function code ✓
- Dependent libraries ✓
- Remote Procedure Call (RPC) communication framework ✗
- Everything packaged as container image. ✗

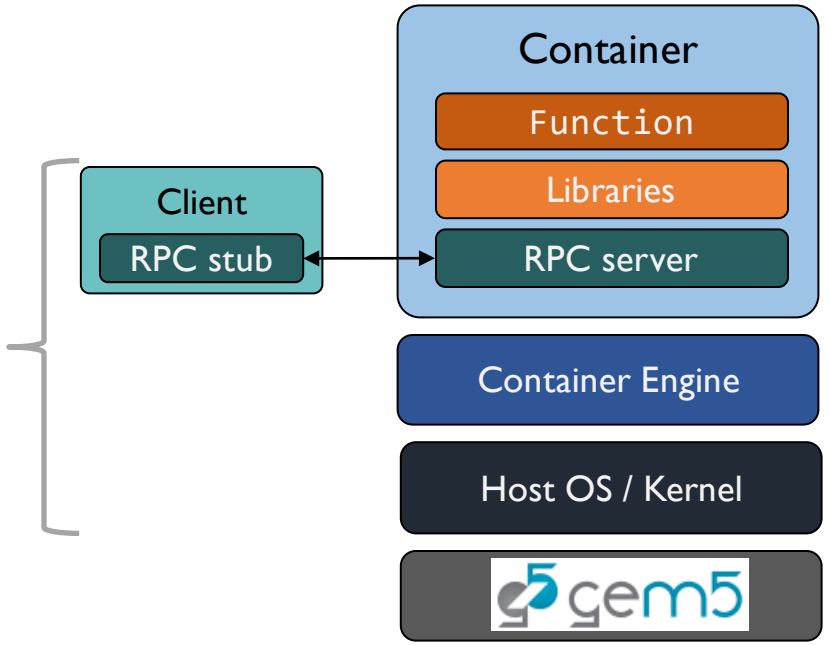
vSwarm-u: Serverless stack for gem5

- Kernel compatible with gem5 and containerization
  - Modules necessary: cgroups, overlays, bridge and net filtering,…
  - gem5 cannot load modules dynamically → need to be compiled.
- Modern OS with container engine installed
- Publicly available https://github.com/ease-lab/vSwarm-u

Bonus: vSwarm: Serverless benchmark suite

- 20 ready to use containerized functions that run on gem5.
- Publicly available https://github.com/ease-lab/vSwarm

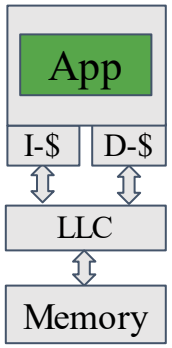
Serverless software stack



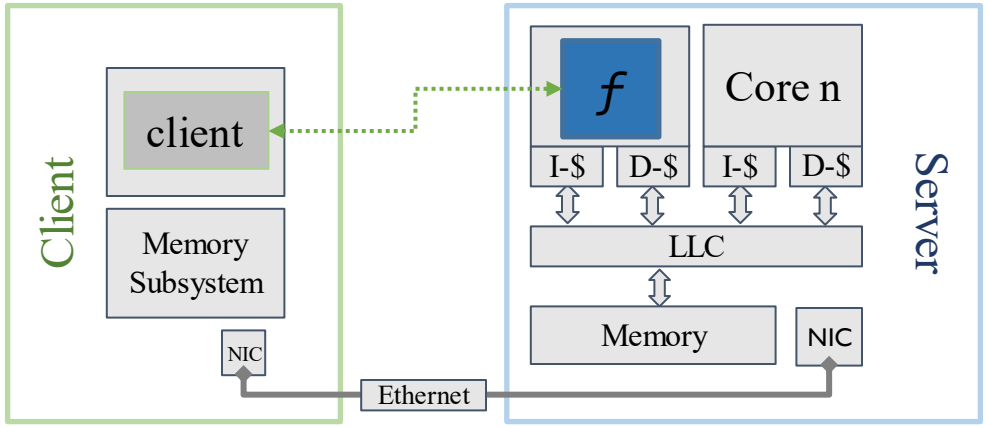
vSwarm-u allows simulation of containers with gem5

2. Challenge of Serverless Host Simulations

Conventional workloads



Serverless workloads



Long execution times:

- Modelling only **core components** is sufficient
  - CPU core and memory subsystem
- Long execution times eliminate side effects of simplified test environment.
  - Simple script for starting the application is sufficient
  - Taking the mean is representative

Short execution times:

- Communication is significant of execution time
  - affected by **many components**
  - Other cores, network,…
- Require sophisticated test environment
  - Isolation between client and server node
  - Precise trigger points for taking measurements

Representative infrastructure for detailed evaluation is required

Requirements on Test Infrastructure

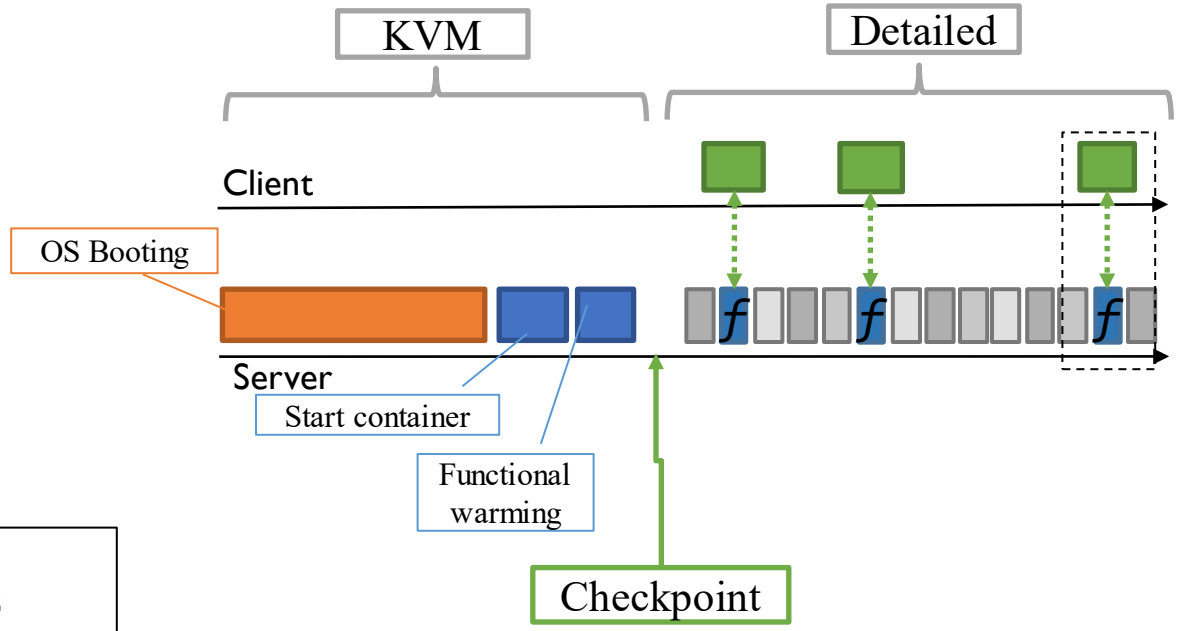
Before measurement:

1. OS booting
2. Start function container
3. Establish connection
4. Functional warming
   - I.e. for warming JIT engine in NodeJS code.

- **Mechanism: `m5.switchCpus(from, to)`**
  - Use kvm core for booting, container start and warming
  - Detailed core for measurement
  - Use checkpoints after functional warming
    - **`m5.checkpoint(path/to/checkpoint)`**

- Use `m5` binary tool to control gem5

```
## Spin up Container
docker run -d --name func-container -p 50051:50051 $IMAGE_NAME
## Pause simulation and switch CPU
m5 exit   ## Use 'm5 fail <code>' to also send a code to gem5
```



KVM accelerates, m5 controls booting and warming

Requirements on Test Infrastructure

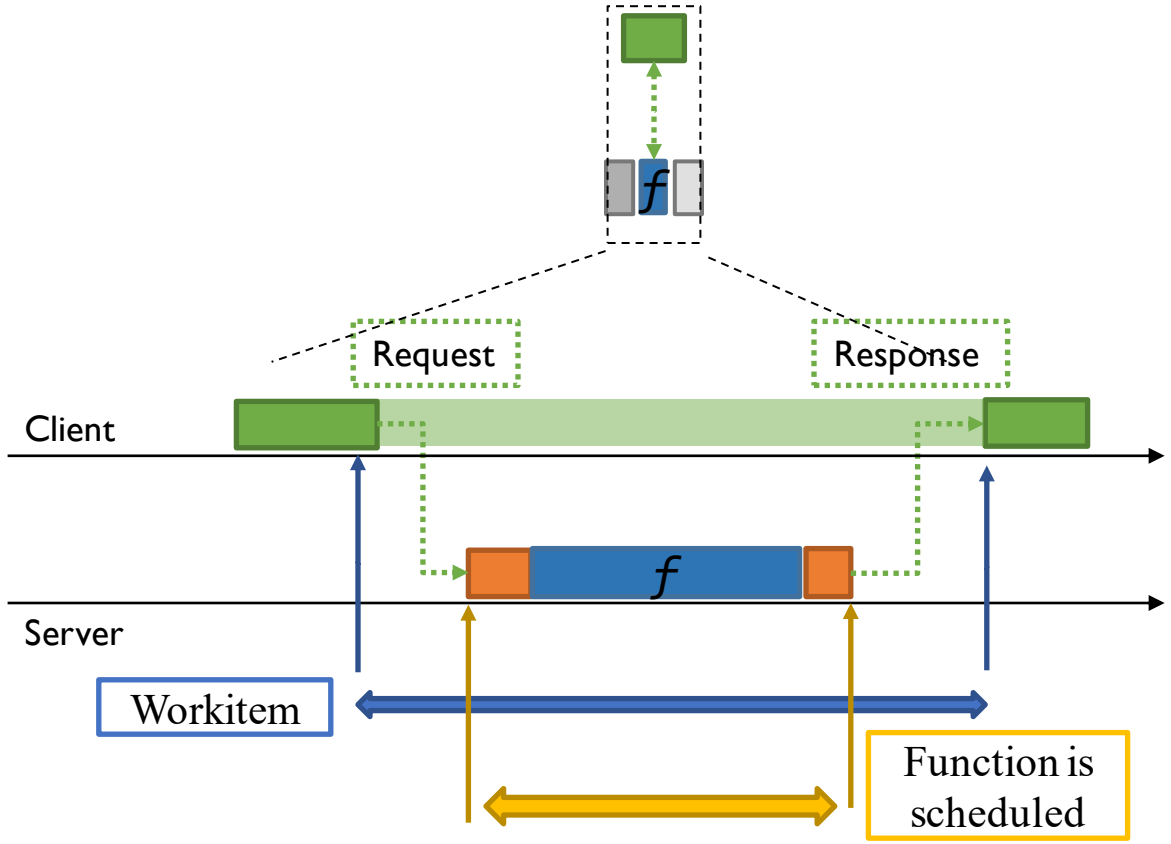
During measurement
- Want to simulate several invocations, different inputs
- Precise trigger points are required

- **Client side:** Instrumented go client
  - **m5ops**

```
m5.WorkBegin(n, 0)   // 21: Send Request
client.Request(pkt) // Invoke function (nth invocation)
m5.WorkEnd(n, 0)     // 22: Received Response
```

- **Server side:** Instrumented Scheduler
  - Attach **FuncEvent** to the symbol table
  - Use **ThreadInfo** to get previous and next pid
    - Kernel need to be build for it.
  - React on certain scheduling events

```
if (prev_pid == 0) ppSwitchIdleActive->notify(true);
…
exitSimLoop(<Exit message>,0, curTick(),0);
```



vSwarm-u offers the test infrastructure for serverless

vSwarm-u Roadmap

What is available already

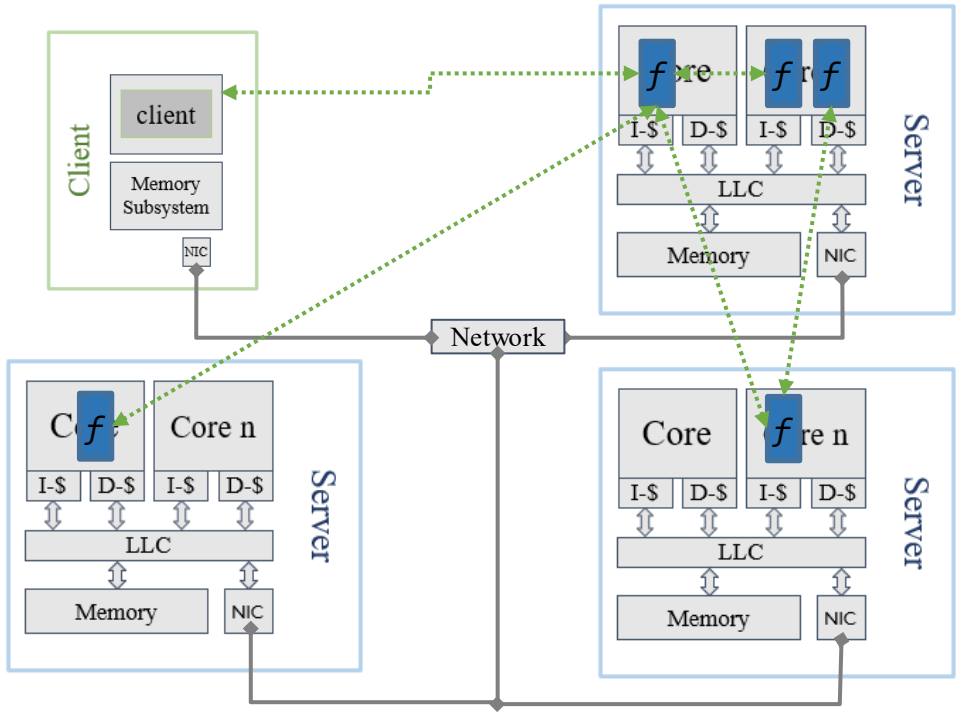
- Support for containerized workloads
- Ready to use standalone kernels from vSwarm
  - 20 Functions, three runtimes
- Server-client test infrastructure
  - One and two machines setups with several cores
  - Trigger point for client side (server side will be rolled out soon)

Actively used for research

- *ISCA'22 Session 9A: Lukewarm Serverless Functions: Characterization and Optimization*

What we plan to do

- Support for *Knative* in gem5
  - Container orchestrator with autoscaling.
- Support distributed workloads
- Support for Arm Instruction set
- Virtualization
  - Arm already supports virtualization



Call for contribution

Takeaways

Serverless functions present new challenges for modern CPUs

→ Unique workload characteristics have limited support in conventional microarchitecture.

→ **Urgent need for more research in hardware for serverless**

Unique workload characteristics have new requirements on research platforms

→ Short execution time require to model the full system / software stack

→ Simplifications in the software stack and test setup result incomplete results

vSwarm-u: Framework for microarchitectural research of serverless workloads.

→ Modern software stack to simulate **containerized workloads** with gem5

→ Robust test infrastructure to do detailed evaluation

Enabled Microarchitectural research for serverless

Thank you!

Questions?

Start Microarchitectural Research for Serverless:

https://github.com/ease-lab/vSwarm-u