MLSys 24

LIFL: A Lightweight, Event-driven Serverless Platform for Federated Learning

Shixiong Qi† K. K. Ramakrishnan† Myungjin Lee★

†University of California, Riverside ★Cisco Research

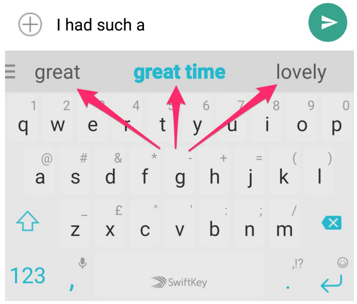
Networked Systems Group

UC RIVERSIDE

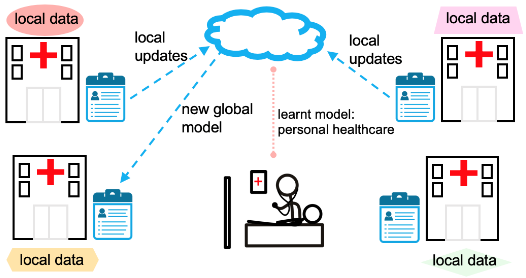
Visit us at:
https://kknetsyslab.cs.ucr.edu/
https://research.cisco.com/

A Quick Primer on Federated Learning

- **Federated Learning (FL) helps**
  - Learn on <u>fresh</u> real-world data
  - Reduce data <u>privacy</u> leakage

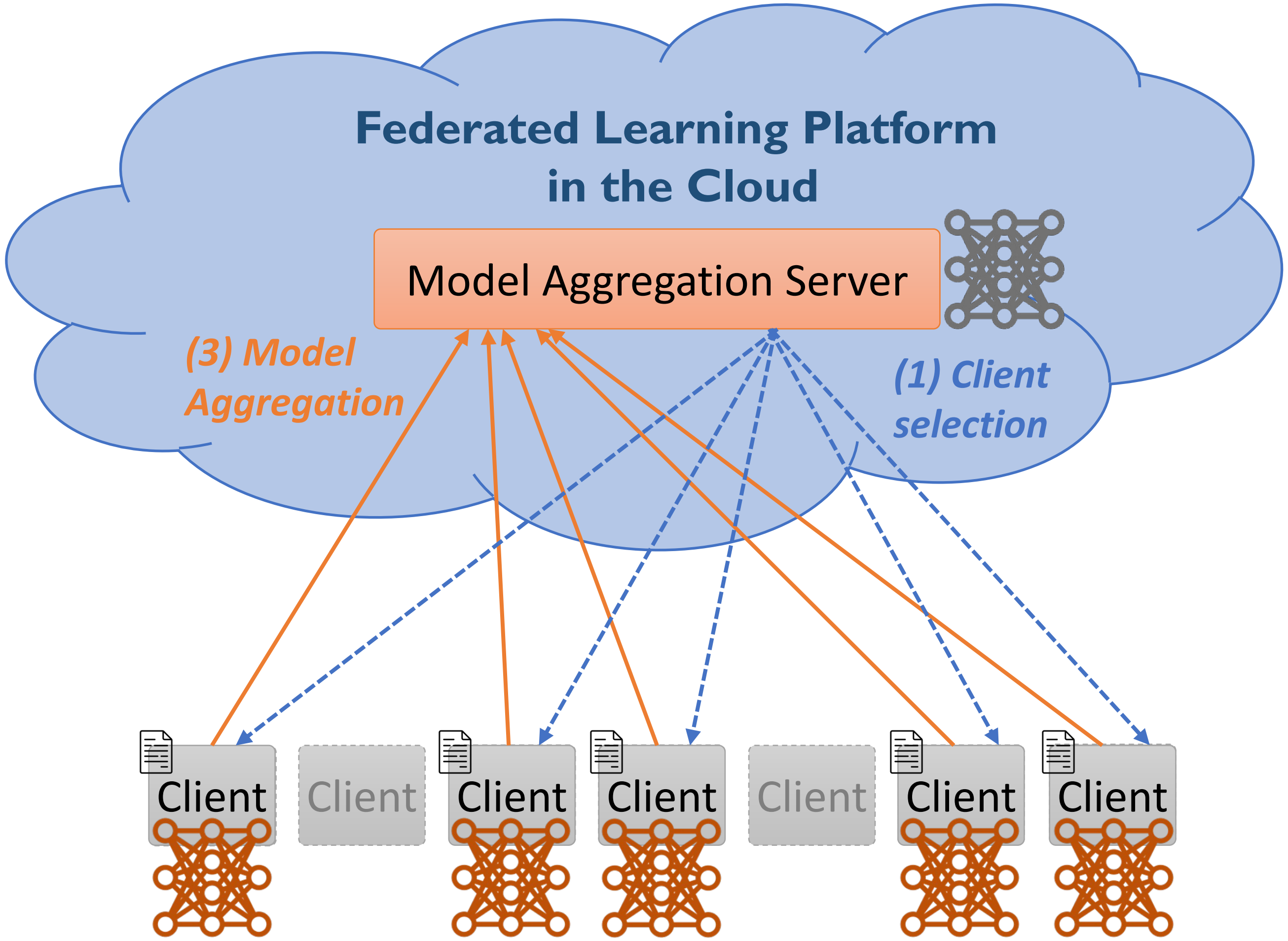


Keyboard prediction



Healthcare

- **Execution of FL**
  - Client selection
  - On-device Training
  - Model Aggregation

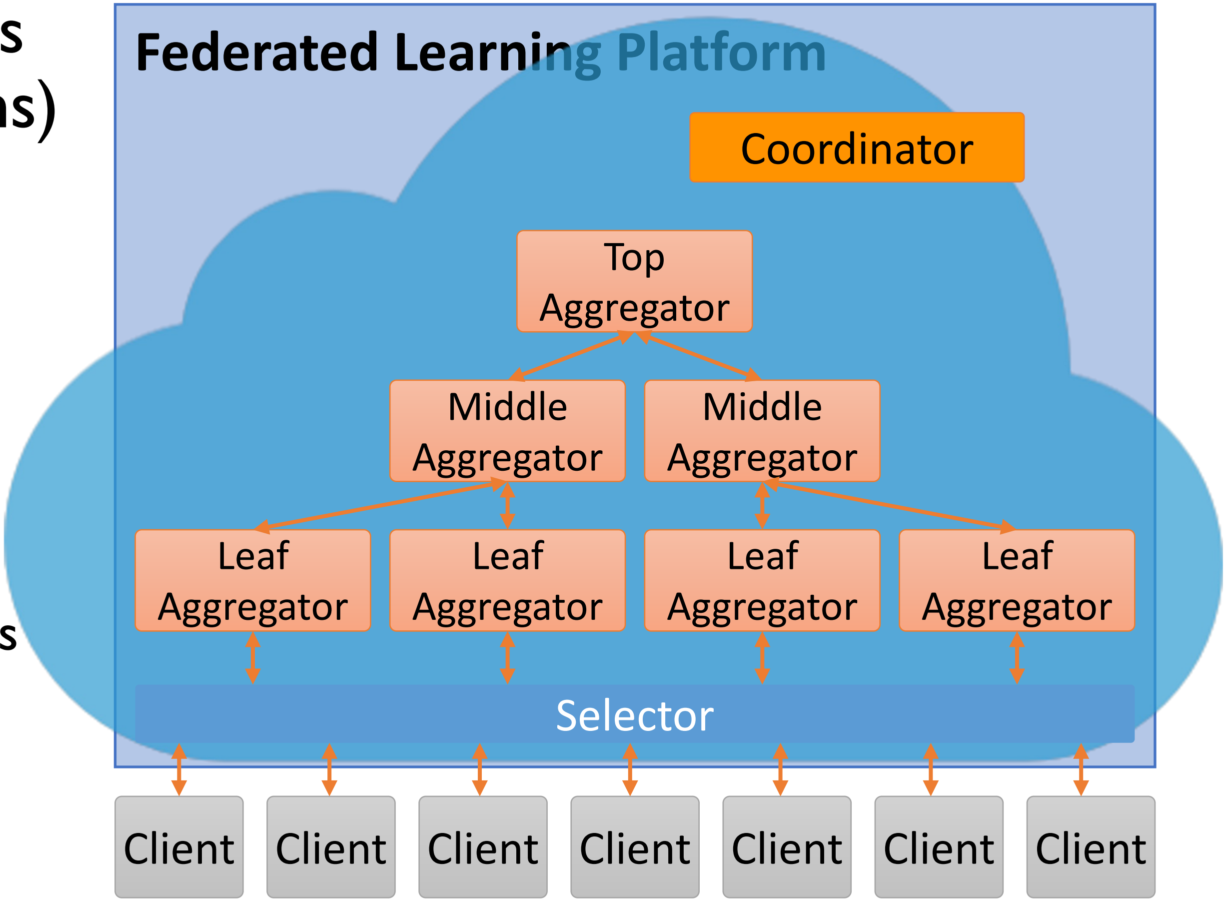


Networked Systems Group
UC RIVERSIDE

Existing System Design for Federated Learning

- Model aggregation server (based on various commercial[1,2] and open-source[3] platforms)
  - **Coordinator**:
    - Orchestrating interactions among aggregators, selectors, and clients
  - **Aggregator**:
    - Hierarchical aggregation
  - **Selector**:
    - Selecting clients to participate in the FL process
    - Client-aggregator mapping

- Use *Cloud* to scale FL training to many clients

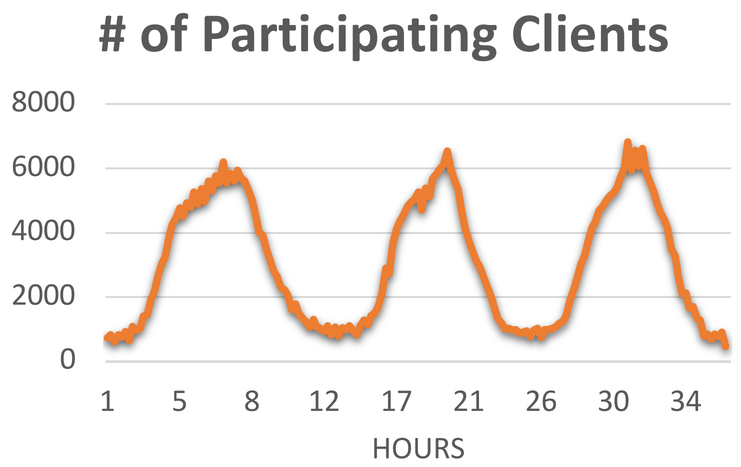


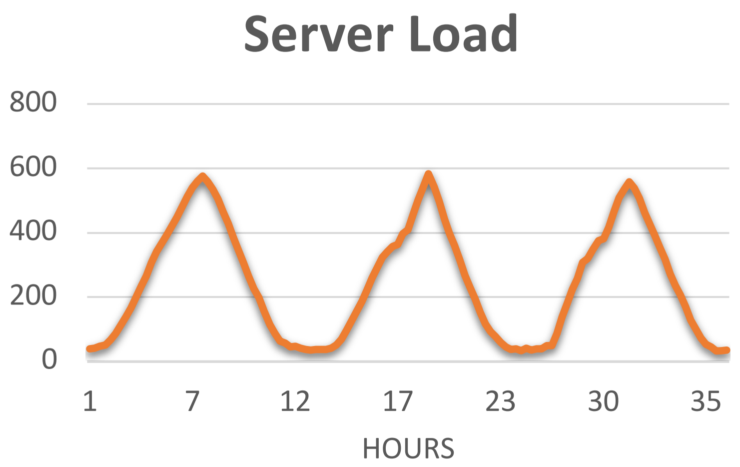
[1] Bonawitz, Keith, et al. "Towards federated learning at scale: System design." MLsys 2019.
[2] Huba, Dzmitry, et al. "Papaya: Practical, private, and scalable federated learning." MLsys 2022.
[3] Daga, Harshit, et al. "Flame: Simplifying Topology Extension in Federated Learning." ACM SoCC'23.

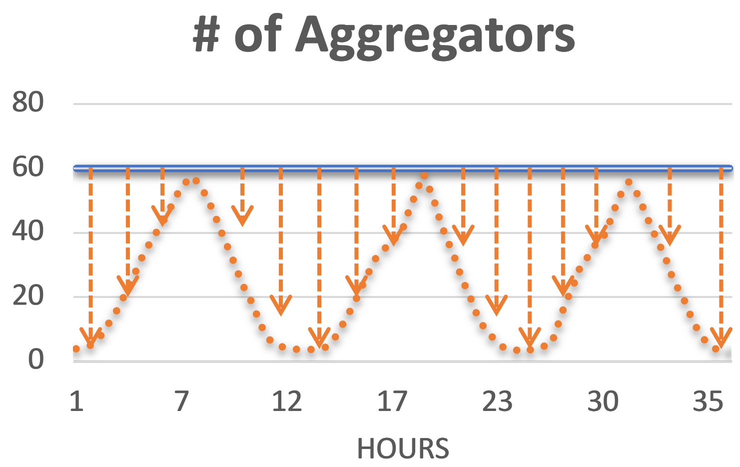
Existing System Design for Federated Learning

Problem statement

- ***High variability of # of FL clients***
  - Real-world trace from GBoard[1]
- ***Serverful*** FL systems lack elasticity







[1] Bonawitz, Keith, et al. "Towards federated learning at scale: System design." MLsys 2019.

Networked Systems Group
UC RIVERSIDE

A “Serverless” Cloud for Federated Learning

Basics of Serverless Computing

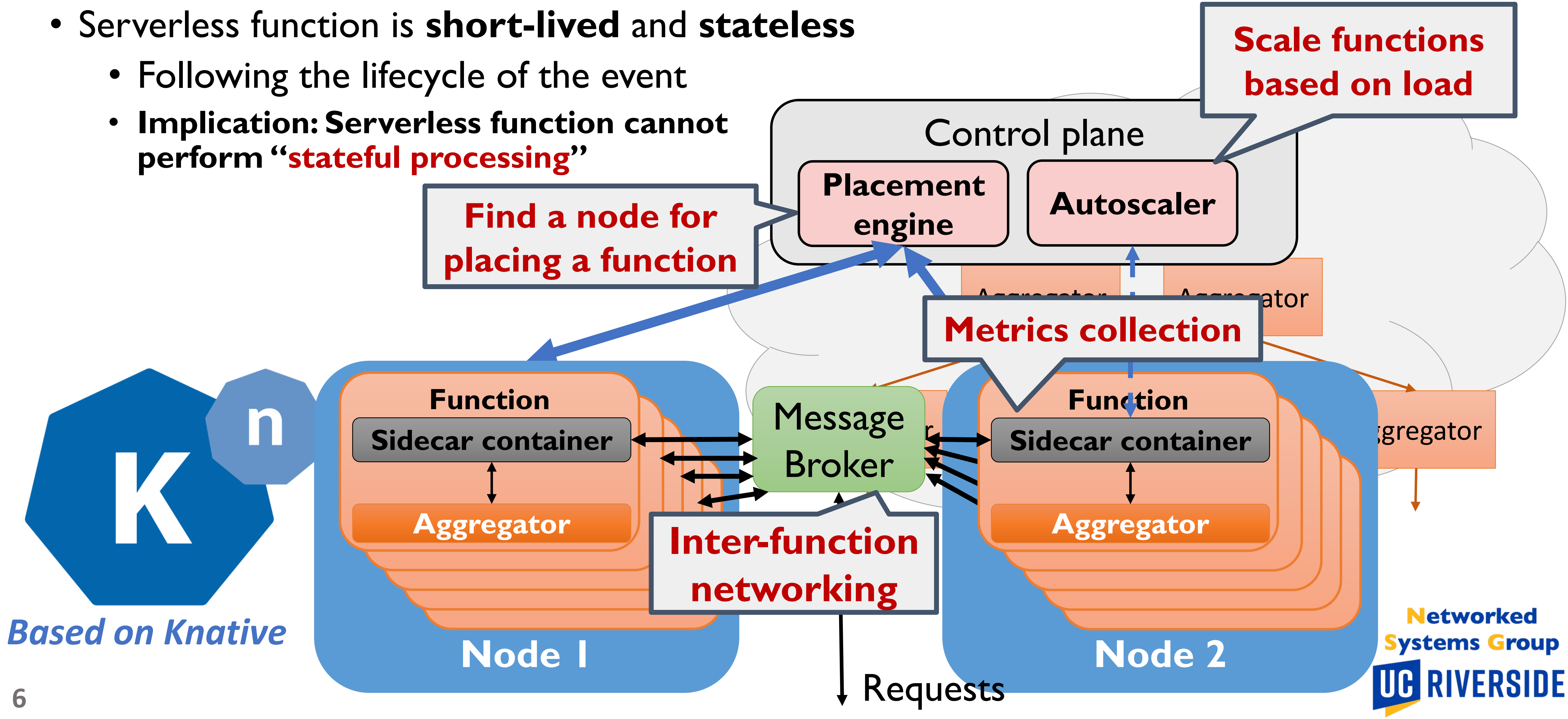
- **“Event-driven” Execution**: Applications are triggered based on events, terminated upon event completion
  - **Fine-grained resource elasticity**
- **True “Pay-as-you-go” Billing**: Pay only for the duration of execution of an application. No charge when the application is idle
  - **Fine-grained billing**

Networked Systems Group
UC RIVERSIDE

A "Serverless" Cloud for Federated Learning

An abstract functional view

- Serverless function is **short-lived** and **stateless**
  - Following the lifecycle of the event
  - **Implication: Serverless function cannot perform "stateful processing"**

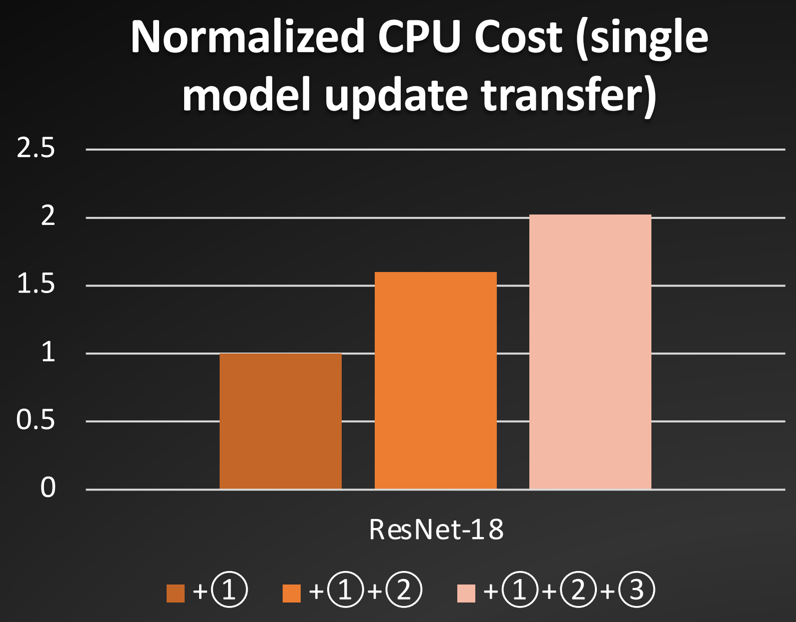


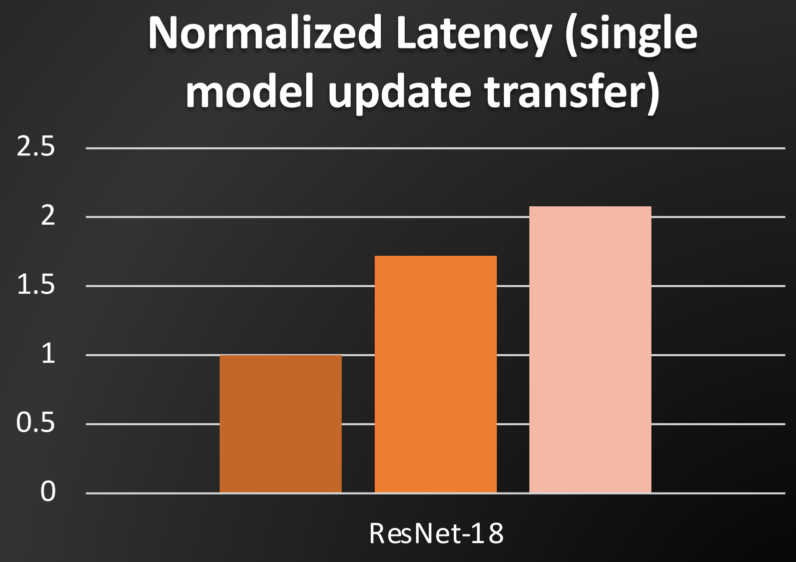
A "Serverless" Cloud for FL

Event-driven model aggregation

- Existing ***serverless*** FL systems[1,2] offer elasticity, but data plane is **heavyweight**[3]
  - +① Kernel-based networking
  - +② Container-based sidecar
  - +③ Message broker
- Control plane is suboptimal
  - Primary designed for web applications

*Unable to support **efficient** FL aggregation*





[1] Jayaram, K. R., et al. "Just-in-Time Aggregation for Federated Learning." IEEE MASCOTS 2022.

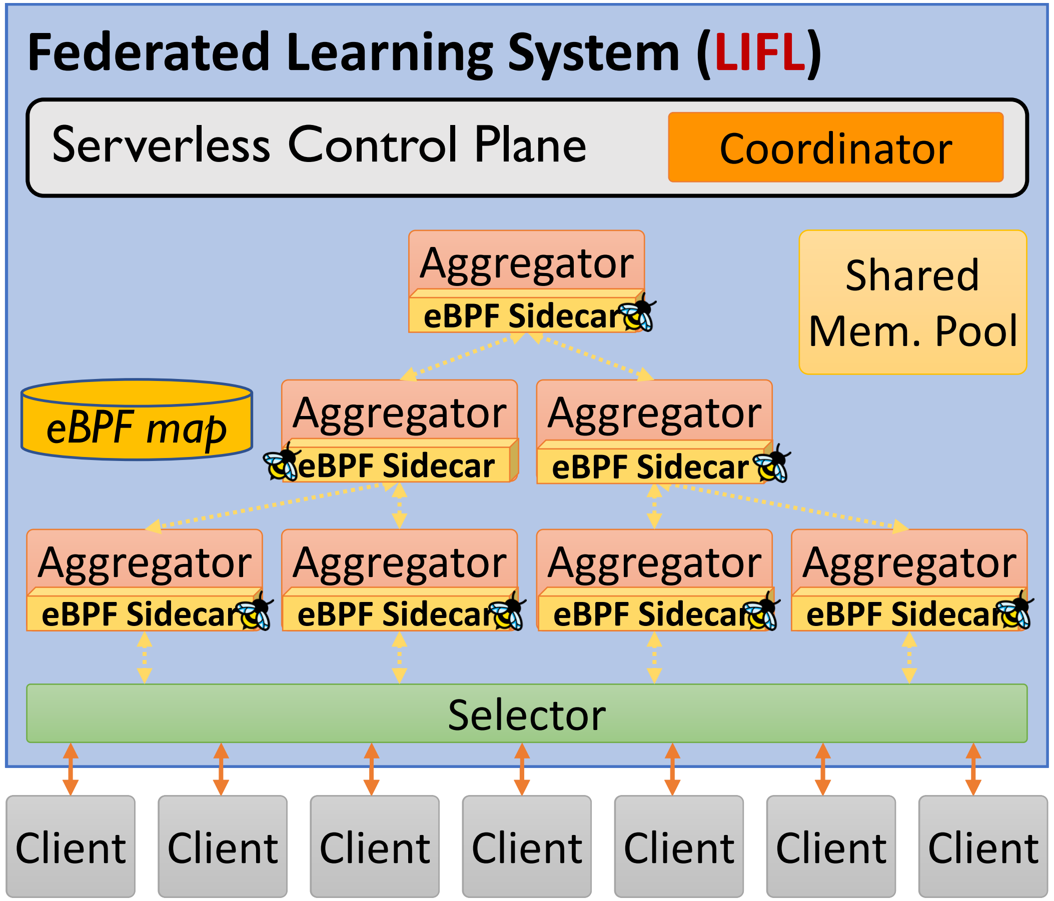
[2] Grafberger, Andreas, et al. "Fedless: Secure and scalable federated learning using serverless computing." IEEE Big Data 2021.

[3] Qi, Shixiong, et al. "SPRIGHT: extracting the server from serverless computing! high-performance eBPF-based event-driven, shared-memory processing." ACM SIGCOMM 2022.

LIFL: Lightweight FL with an optimized serverless design

Two primary focus in LIFL

- **Streamline the serverless data plane**
  - Use eBPF to offload sidecar and message broker
  - Use shared memory processing to speed up hierarchical aggregation
- **Control plane optimization**
  - Locality-aware placement
  - Hierarchy-aware scaling
  - Aggregator reusing
  - Eager aggregation

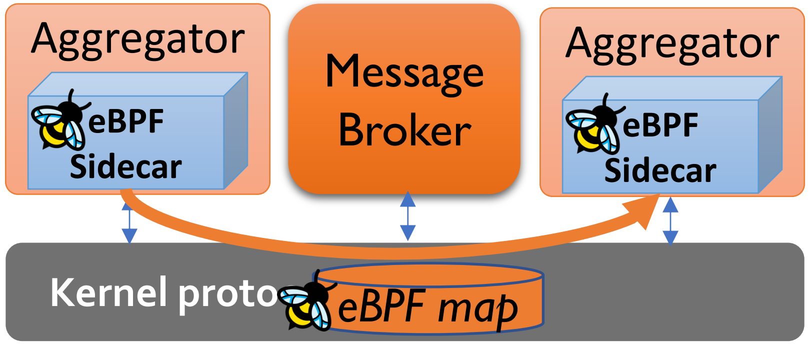


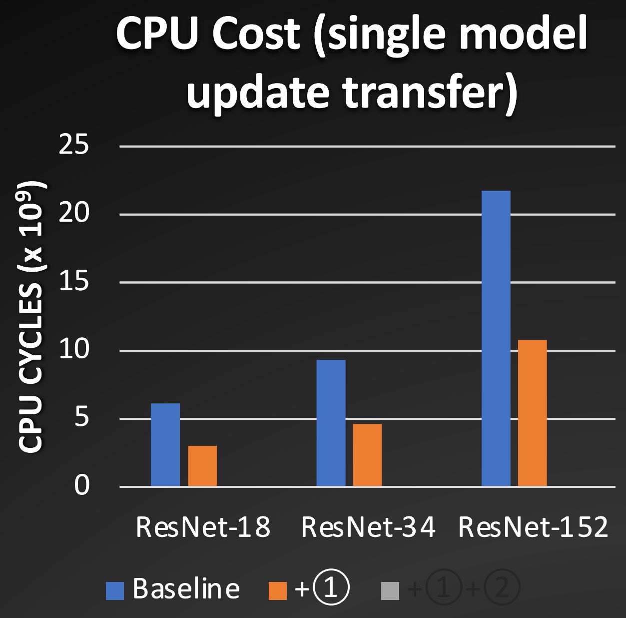
Networked Systems Group
UC RIVERSIDE

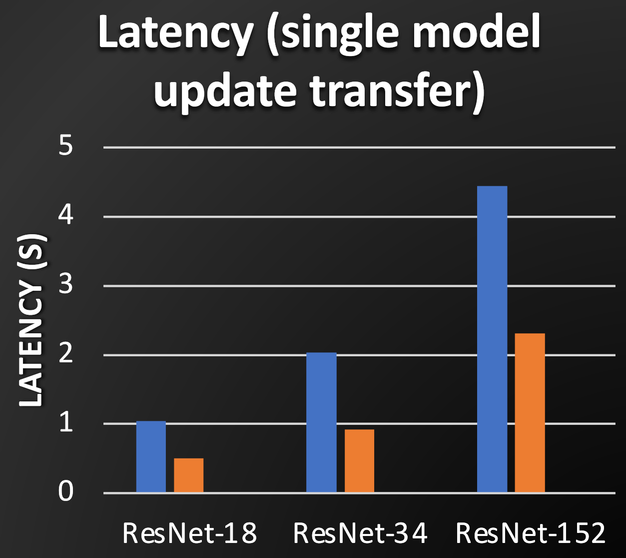
Data Plane Optimizations in LIFL

+① eBPF-based stateful processing

- **eBPF:** a code snippet attached to a specific "hookpoint" in the kernel
- **eBPF supports event-driven execution**
  - NO cost when idle
- **eBPF's stateful processing**
  - *In-kernel eBPF map*
  - Metrics collection, Routing between aggregators



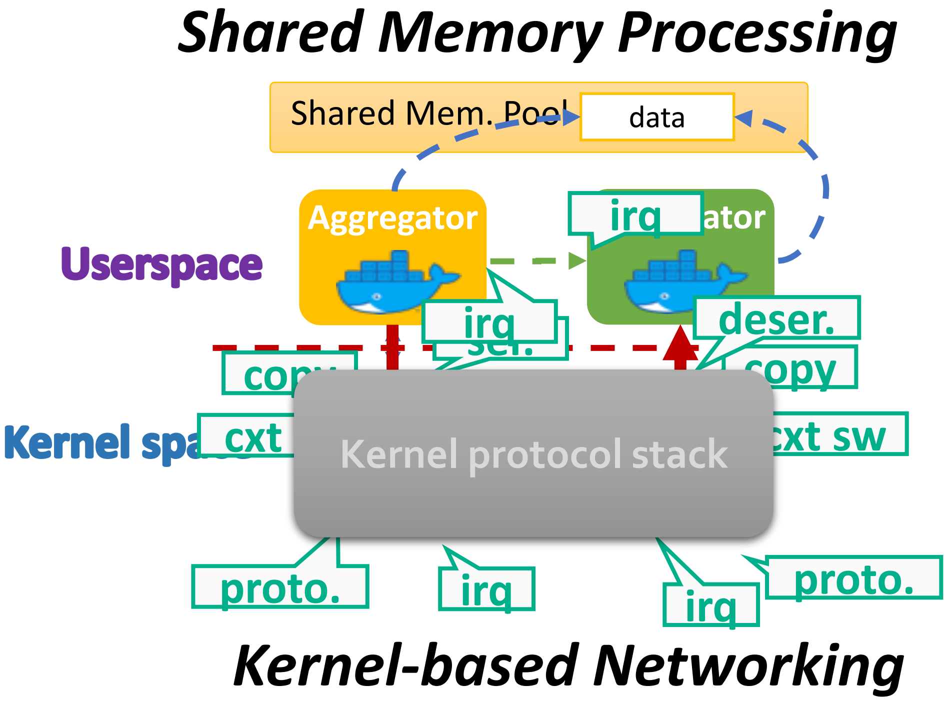


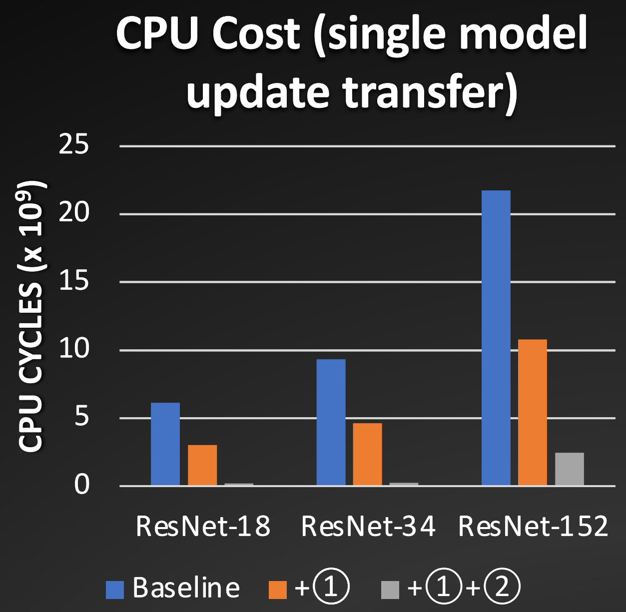


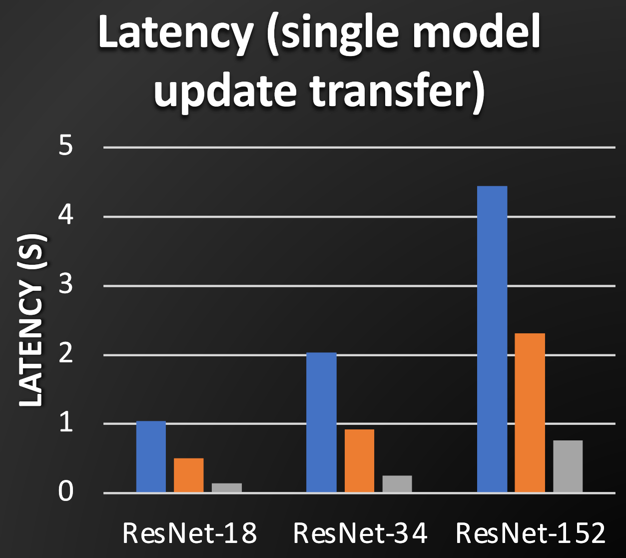
Data Plane Optimizations in LIFL

+② Shared memory processing for hierarchical aggregation

- **Bypass the kernel**
- **Overhead saving by shared memory processing**
  - Context switch, interrupt, copy, protocol processing, serialization/de-serialization
- **Pass by reference**



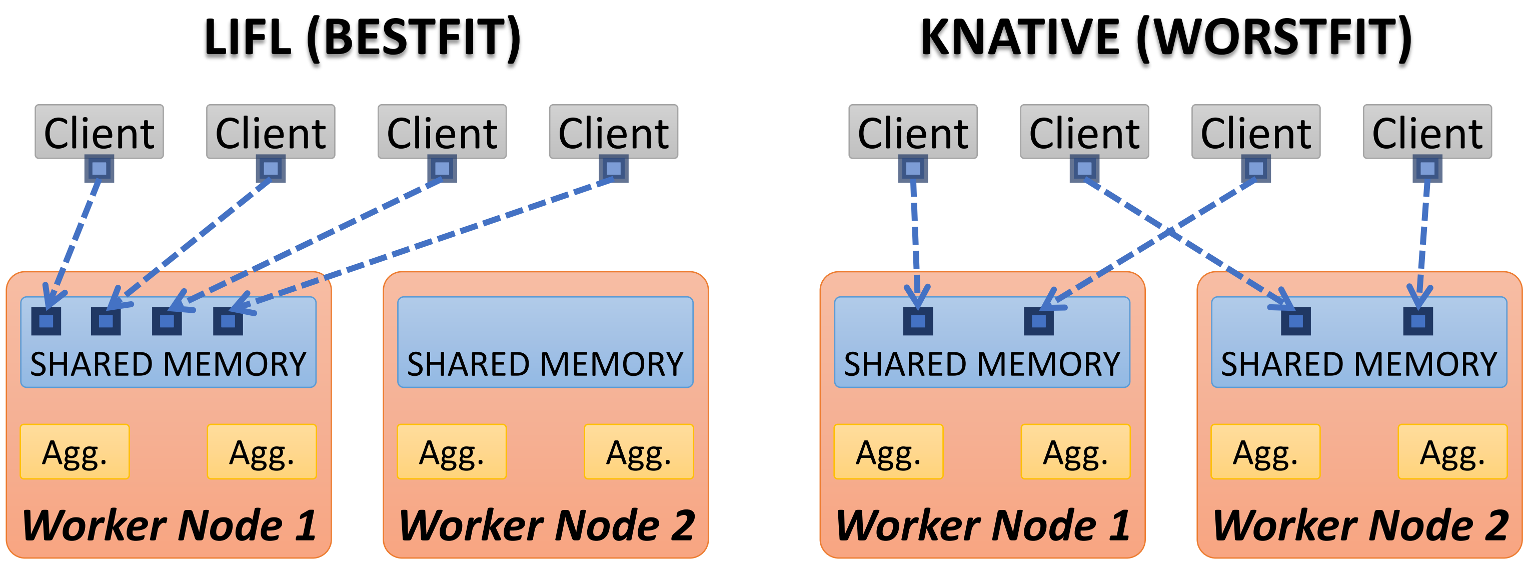


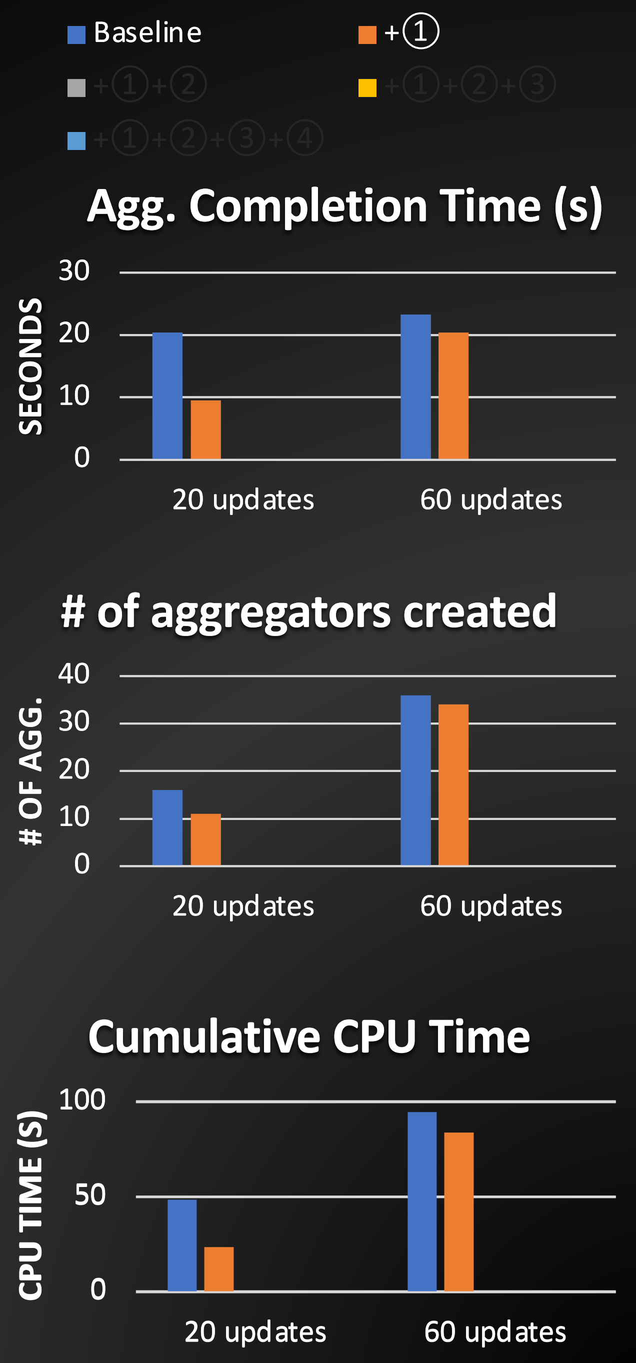


Control Plane Optimizations in LIFL

+① Locality-aware Placement

- Inter-node communication still uses kernel networking
  - Maximize shared memory processing
- Approached as a **bin-packing** problem
  - We choose ***BestFit*** for LIFL
    - Concentrates load onto the fewest nodes possible
  - Existing serverless design (e.g., Knative) use WorstFit
    - Spread the load across more nodes

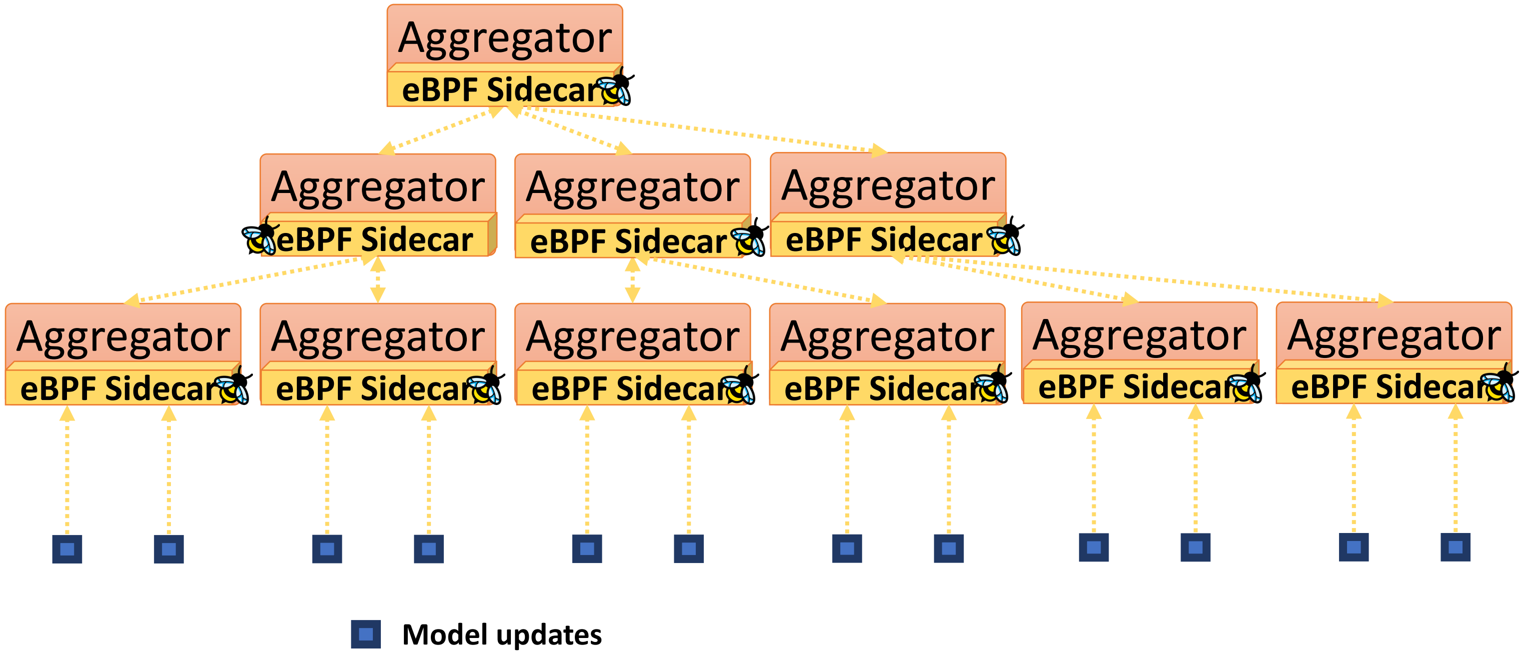


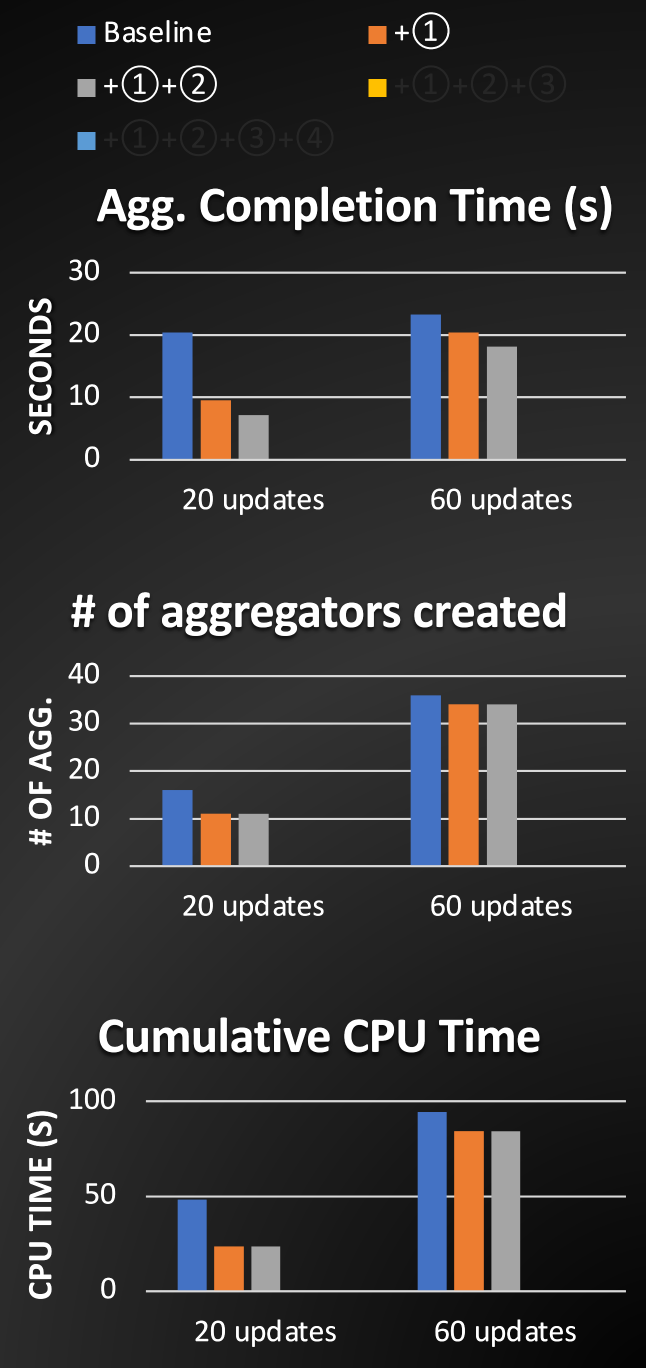


Control Plane Optimizations in LIFL

+② Hierarchy-aware Scaling

- We use **Exponentially Weighted Moving Average** to estimate **arrival rate** of model updates on each node
- Maximize the parallelism of aggregation **at each level**

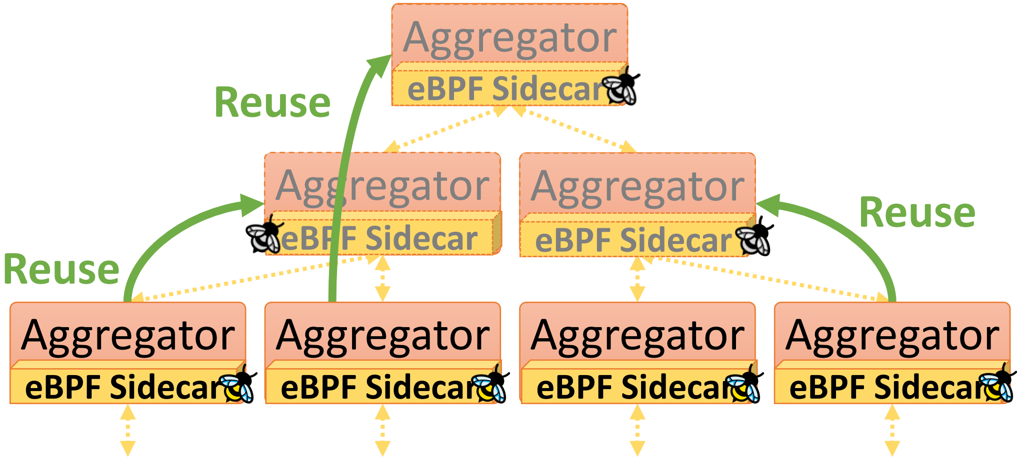


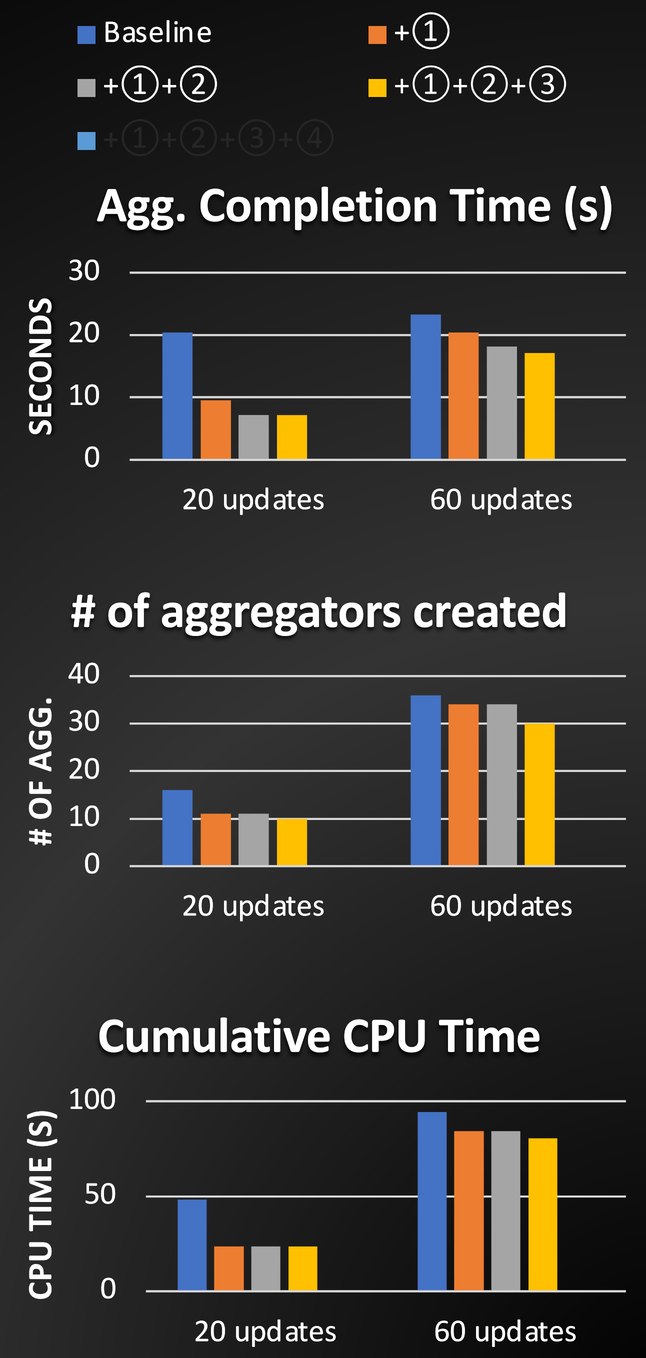


Control Plane Optimizations in LIFL

+③ Aggregator Reusing

- Aggregators at the higher level are often idle
  - While the leaf aggregators are working
  - Vice versa
- Aggregators in LIFL are homogeneous
  - Same simple function of summation



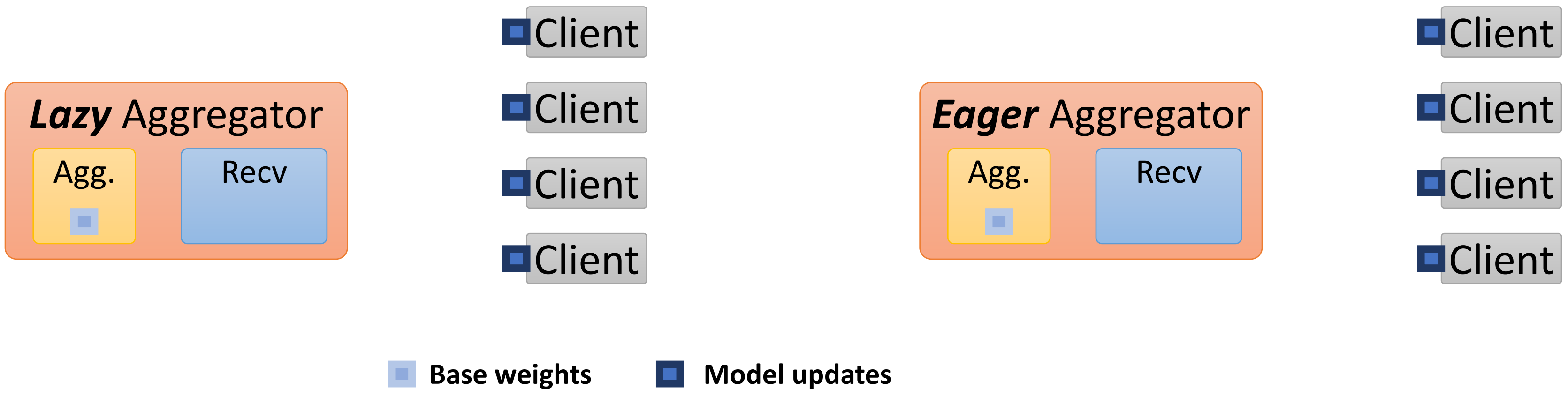


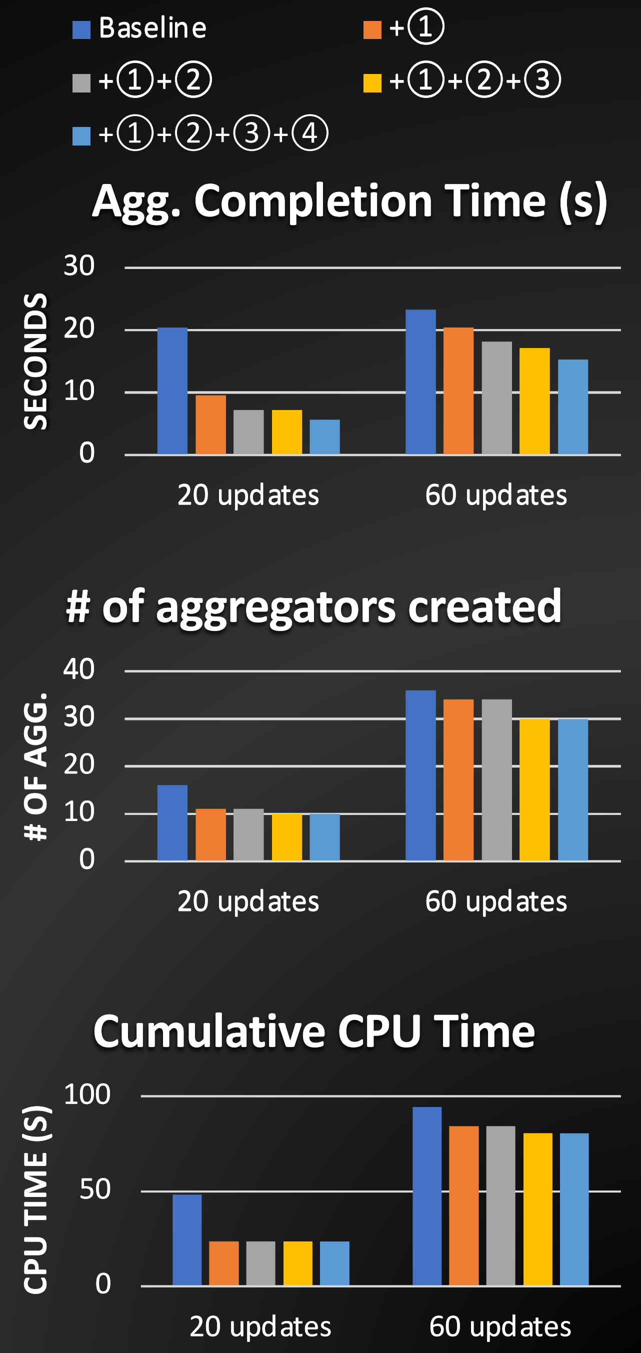
Control Plane Optimizations in LIFL

+④ Eager Aggregation

Key idea: aggregate the arriving updates **immediately**

- Leverage the overlap between the start-up delay and transfers of model updates, allowing eager aggregation to mask cold starts up until the last model update





Put it all together

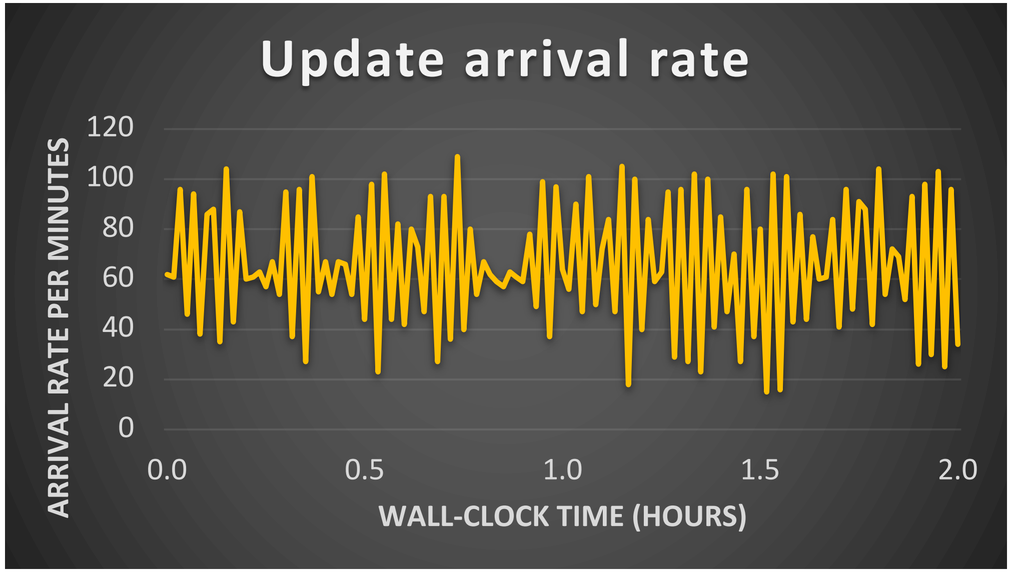
Evaluating LIFL's data and control plane

Alternatives

- LIFL vs. Knative-based Serverless vs. Always-on Serverful

Workload (from FedScale[1]):

- ResNet-18 FL clients (a total of 2,800 clients used)
- FEMNIST dataset
- ***Varying load***



Implementation of LIFL is based on **FLAME**[2] – an extensible framework that eases support for new FL training topologies and their workloads

[1] Lai, Fan, et al. "Fedscale: Benchmarking model and system performance of federated learning at scale." International conference on machine learning. PMLR, 2022.
[2] Daga, Harshit, et al. "Flame: Simplifying Topology Extension in Federated Learning." ACM SoCC'23.

Networked Systems Group
UC RIVERSIDE

Put it all together

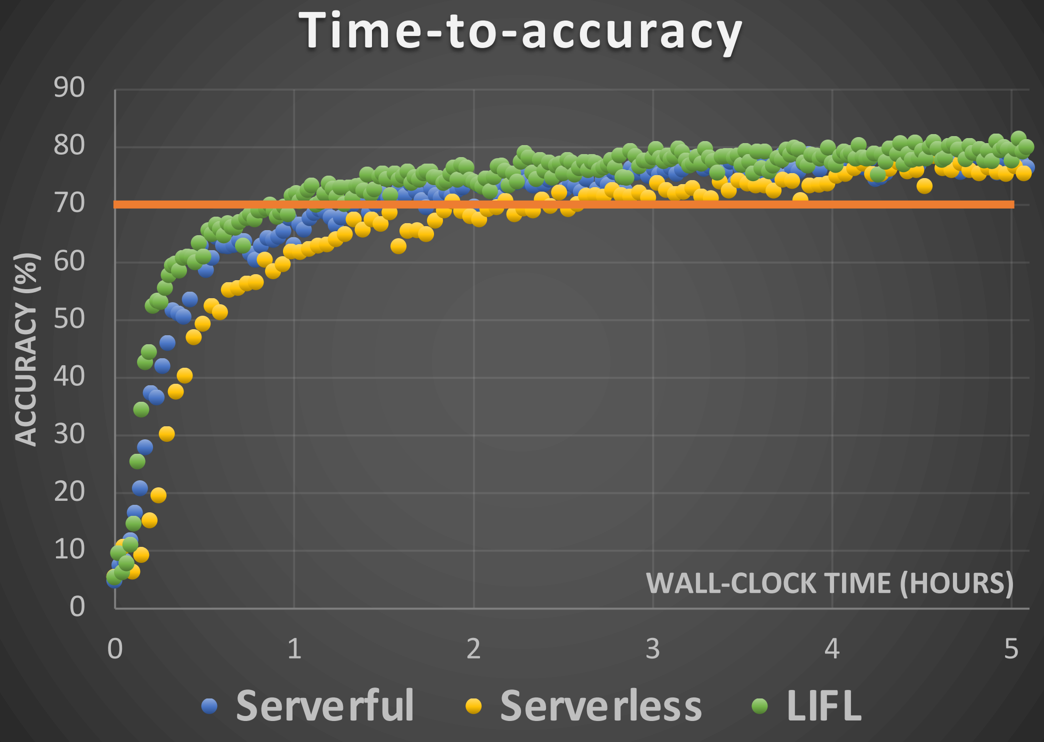
Serverful vs. Serverless vs. LIFL

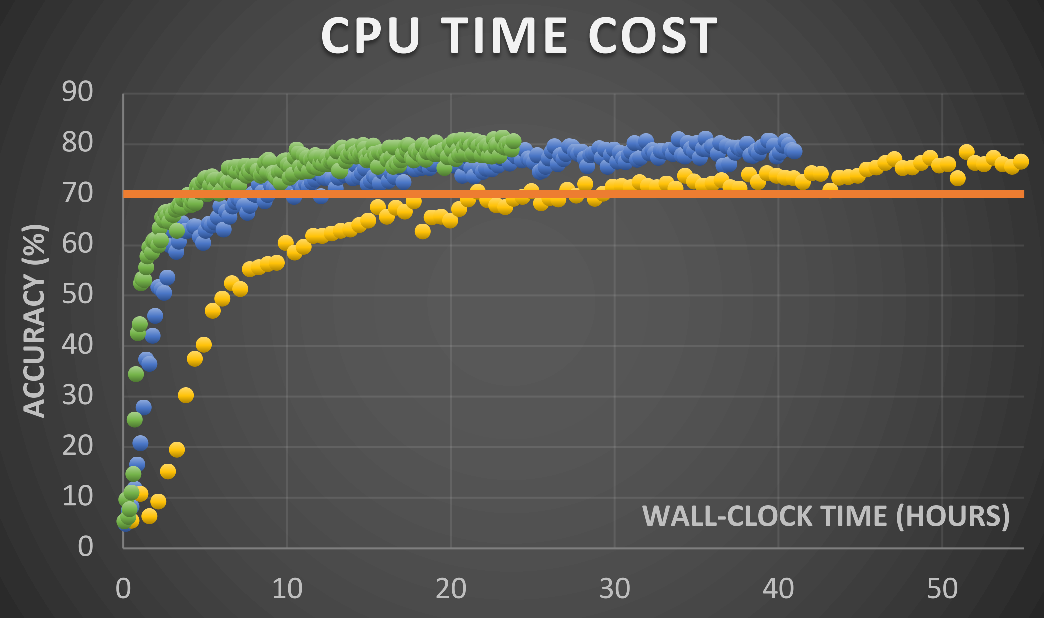
Overall outcomes:

- **LIFL** achieves **1.6X** faster time-to-accuracy than **Serverful** and **2.7X** faster than **Serverless**
- **LIFL** has **1.8X** less CPU time cost than **Serverful** and **5.7X** less than **Serverless**

Our design makes FL aggregation more efficient and faster!

For how LIFL trains a more heavyweight ResNet-152 model, please refer to our paper





Conclusion

- Dependency on cloud to scale FL training to many clients
  - Serverless computing is an ideal fit for varying FL aggregation workload

- Existing serverless designs involve heavyweight data plane and suboptimal control plane

- LIFL incorporates the control and data plane optimizations in serverless computing
  - Truly deliver the promise of serverless
  - Make the FL aggregation more efficient and faster

- LIFL is open-sourced as part of Flame
  - ☞ Find LIFL at: https://github.com/cisco-open/flame.git
  - If you have any questions or comments, please feel free to email us (flame-github-owners@cisco.com and sqi009@ucr.edu)

Networked Systems Group

UC RIVERSIDE

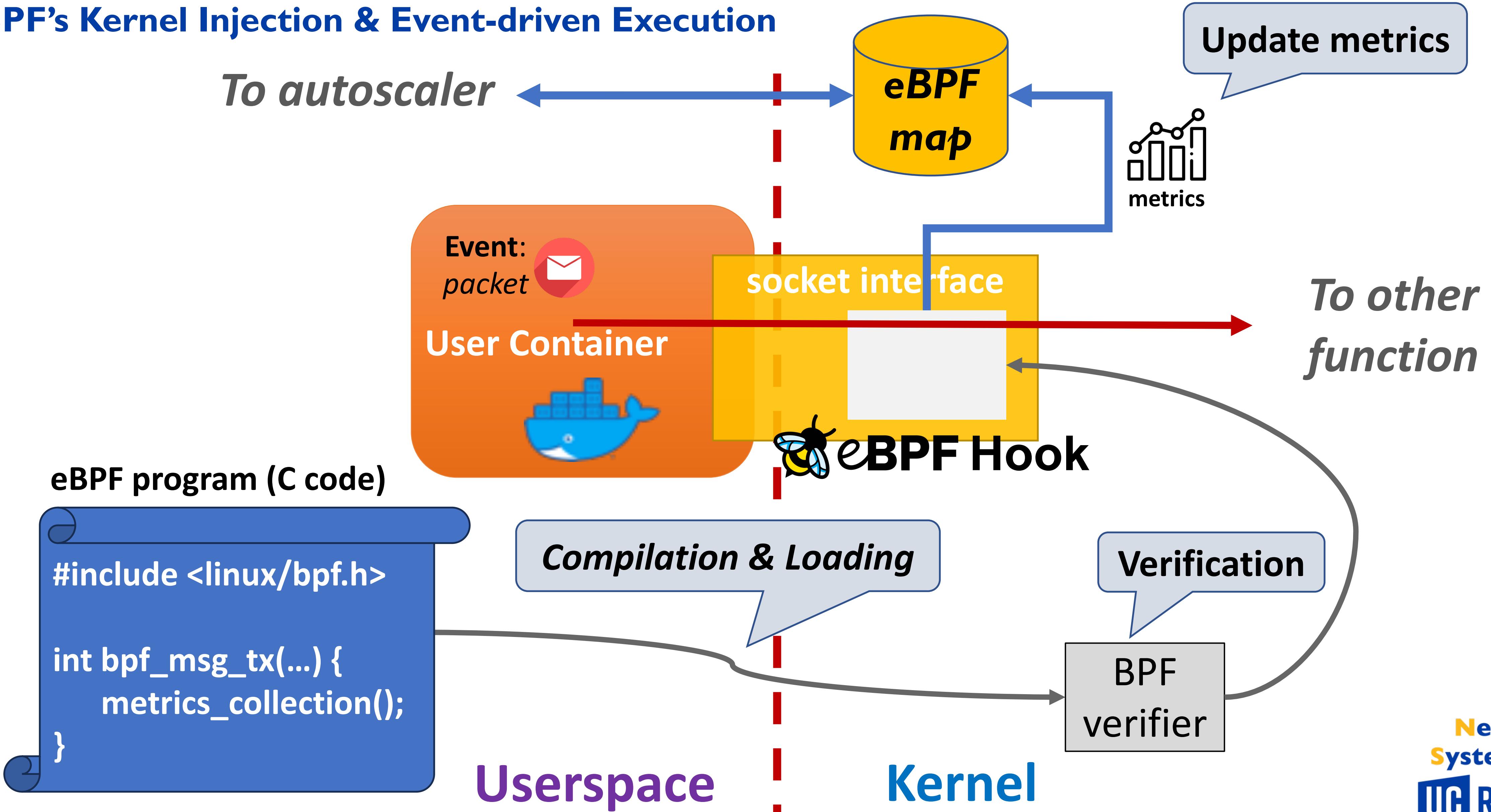
Backup Slides

Networked Systems Group

UC RIVERSIDE

Basics of extended Berkeley Packet Filter (eBPF)

eBPF's Kernel Injection & Event-driven Execution

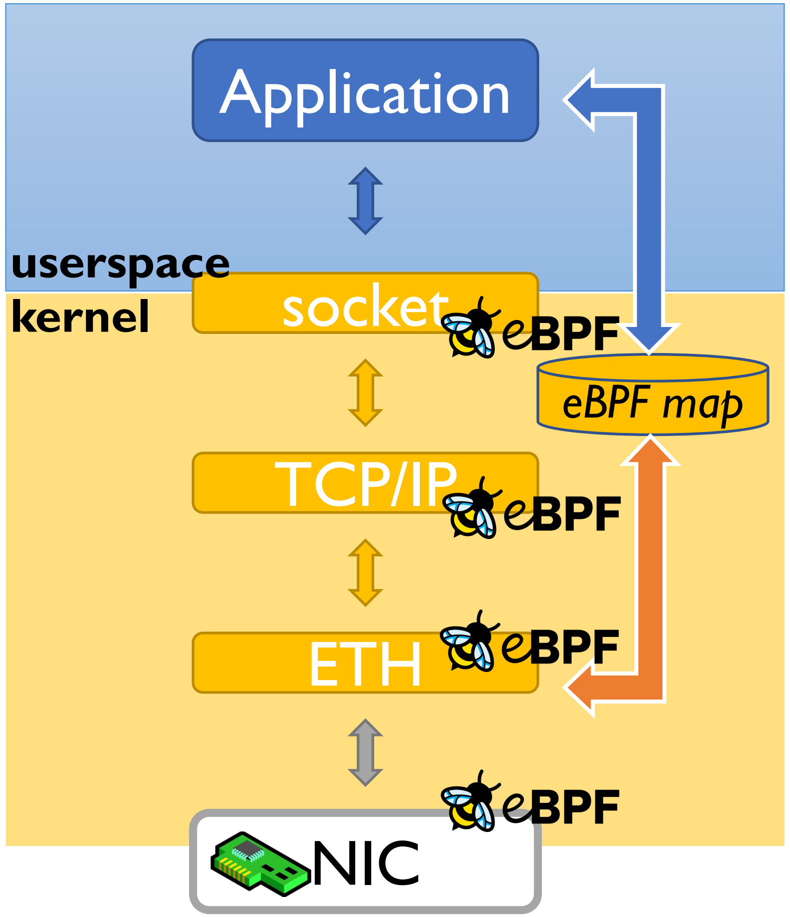


Networked Systems Group
UC RIVERSIDE

Basics of extended Berkeley Packet Filter (eBPF)

Features of eBPF

- **Various hook points in kernel**
  - Sockets, protocol stack, network device drivers, …
- **Programmability**
  - Dynamic Loading; No change to the kernel
  - Transparent to the user function
- **Stateful** processing offloaded to **kernel**
  - ***eBPF Map***
  - Help in keeping ***states***, e.g., routes, metrics

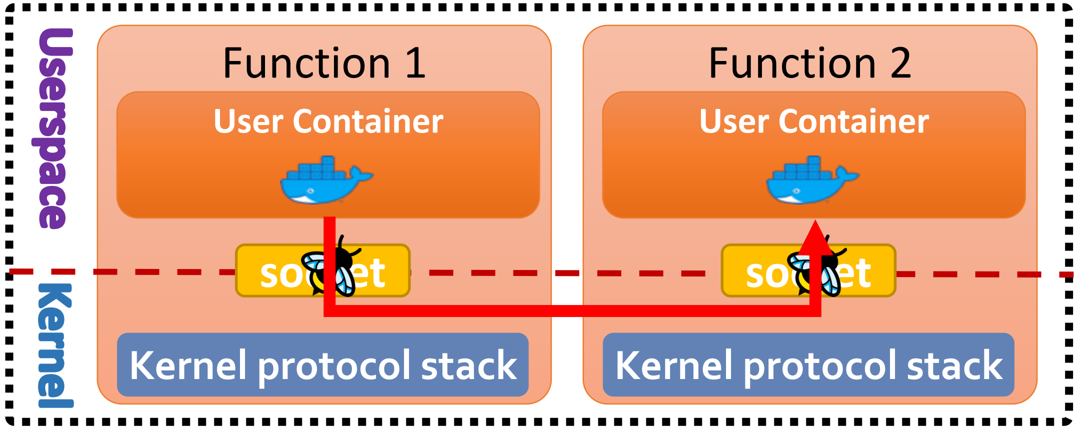


Networked Systems Group
UC RIVERSIDE

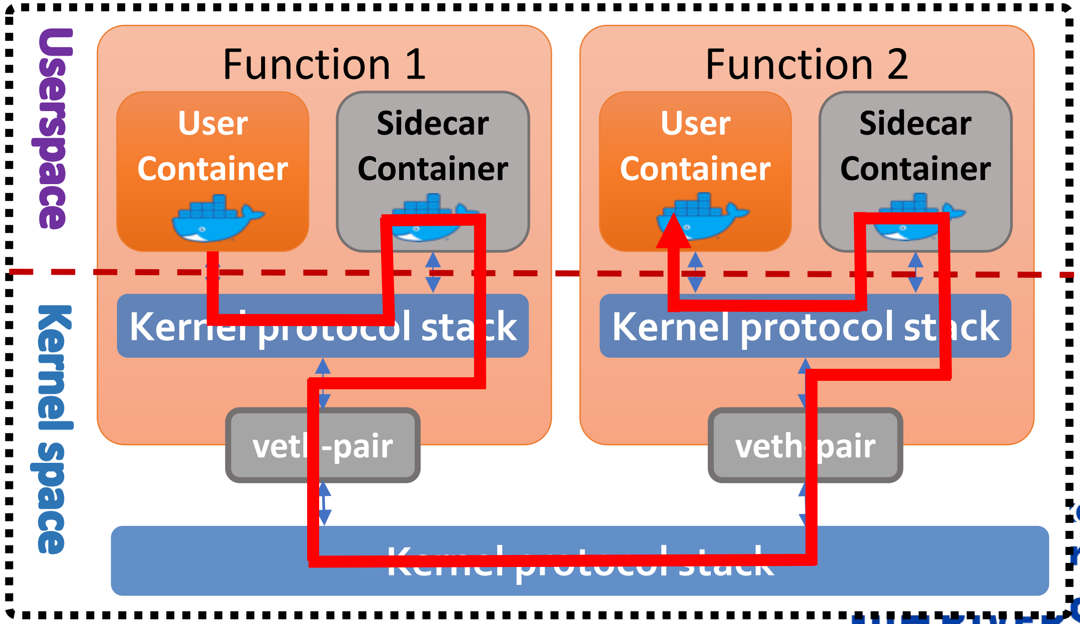
eBPF-based Event-driven Sidecar in LIFL

- In-kernel eBPF-based "sidecar"
  - Sidecar being injected at the socket interface
    - Metric Collection
    - Traffic Filtering
    - *Routing*
- **All in the kernel**
  - Avoid ***extra*** user-kernel boundary crossings
- **Purely event-driven**
  - **No** CPU overhead when there are no requests

eBPF-based sidecar

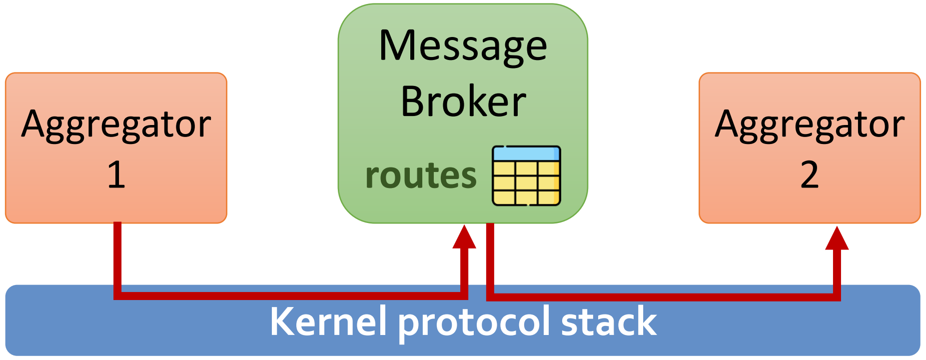


Container-based sidecar

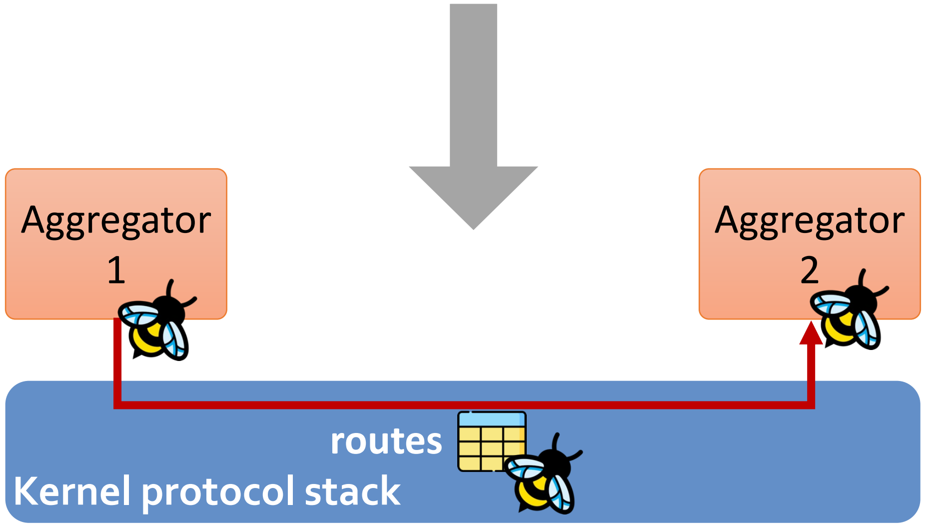


Direct Function Routing in LIFL

- Serverless aggregators are ***stateless***
  - Offloaded stateful processing (***routing***) to message broker
- Having the broker perform invocations between aggregators is unnecessary
  - Routing overhead
- *Direct Function Routing* in LIFL
  - Offloading routes to ***in-kernel eBPF map***
  - Bypassing the userspace broker



Indirect Function Routing

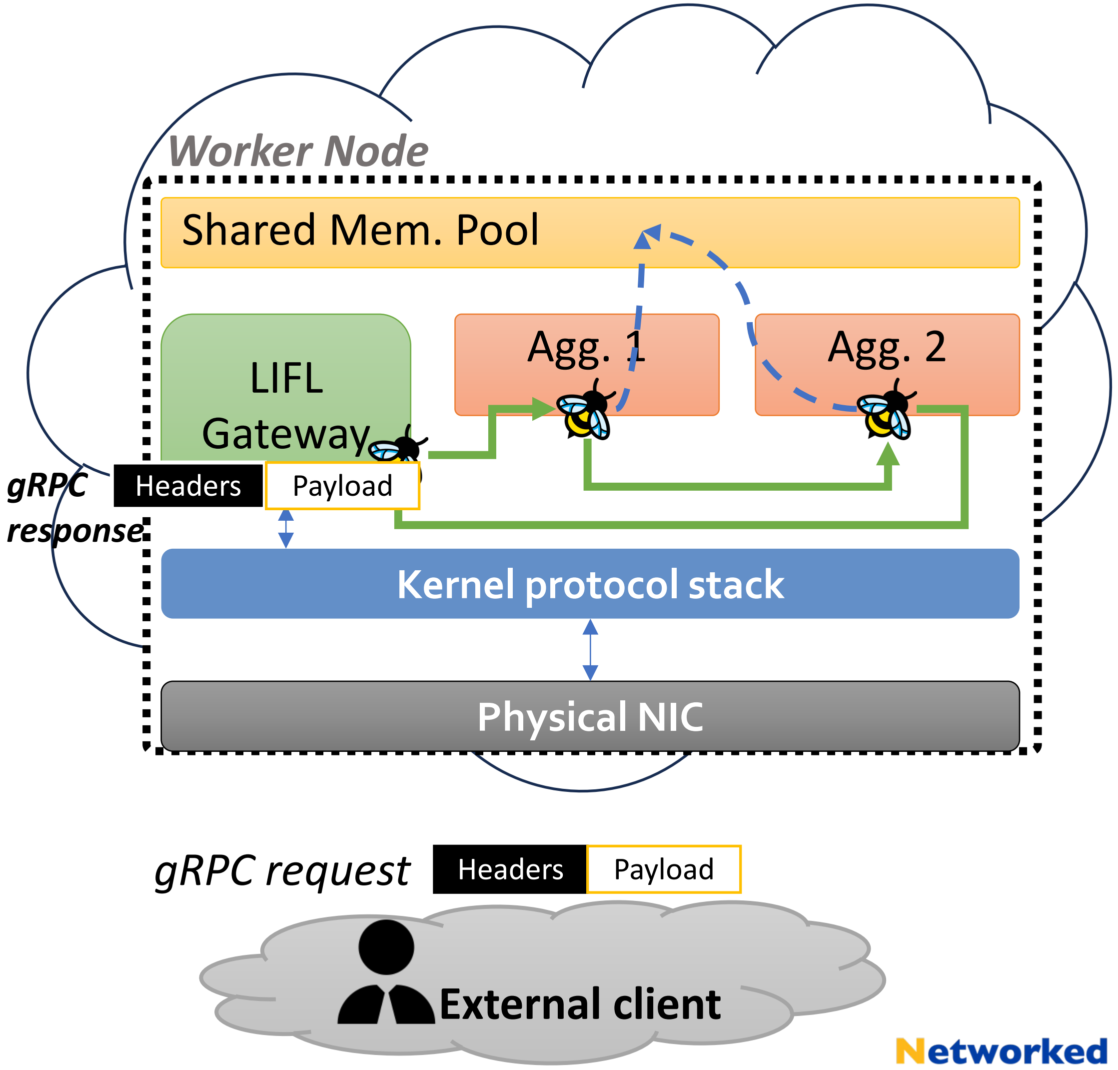


Direct Function Routing

Networked Systems Group
UC RIVERSIDE

Shared Memory Processing

- How to handle protocol processing?
  - **LIFL Gateway**: Entry-point of a local hierarchy
    - **Consolidate** kernel protocol processing
    - Move model updates into shared memory
- Shared memory processing **between** aggregators
  - "Pass-by-reference" instead of "Pass-by-value"
  - We use **eBPF** to deliver ***references***
    - Socket-to-socket transfer

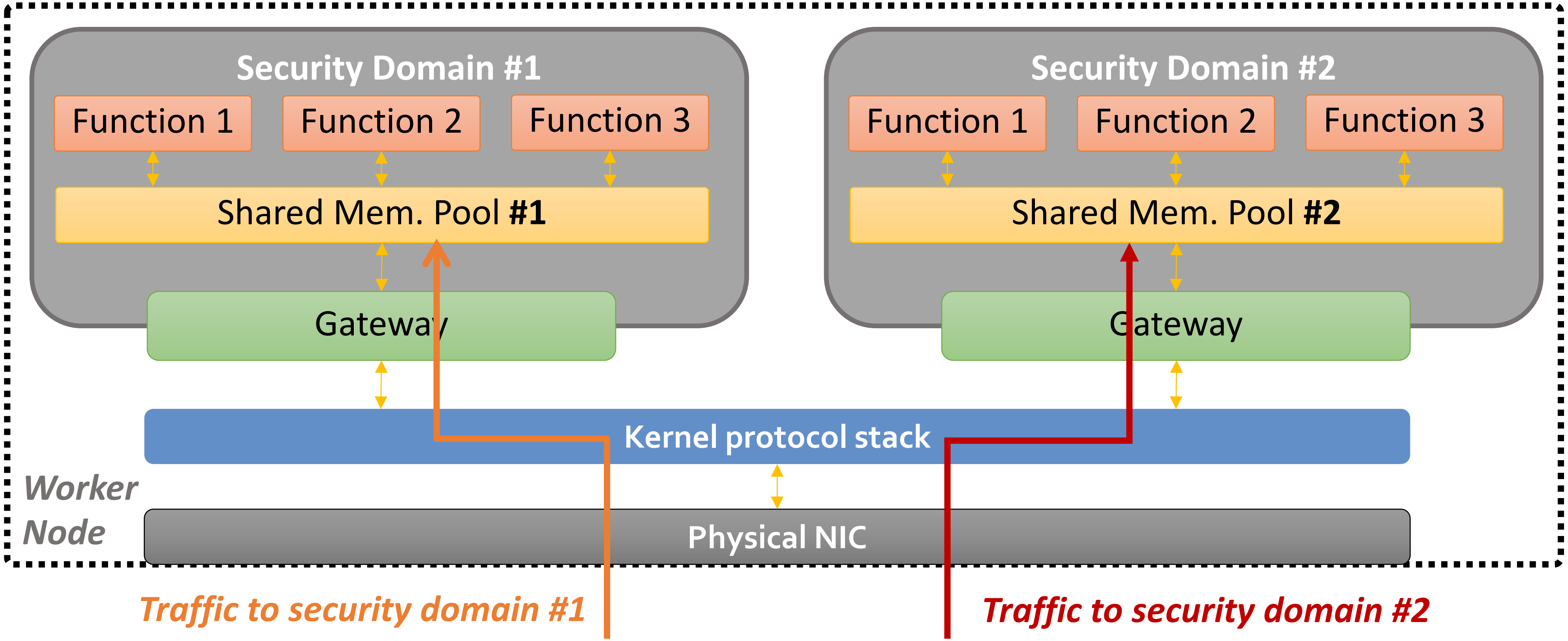


Networked Systems Group
UC RIVERSIDE

How to secure shared memory processing

Our Solution: Security domain

- **Trust model**: functions within a chain trust each other, functions in different chains may not
- We construct a security domain for each function chain
  - a **private** shared memory pool for each chain

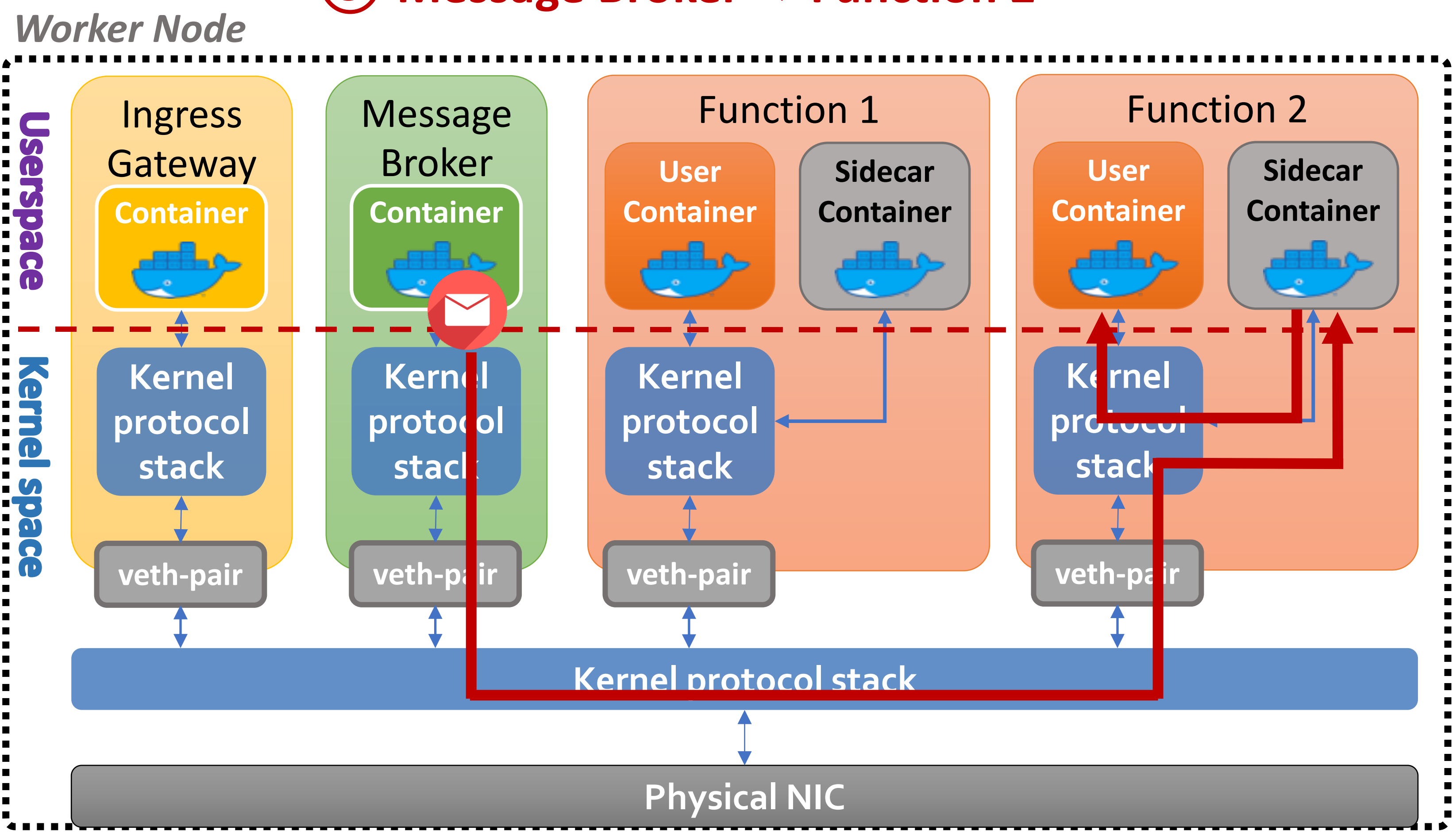


Networked Systems Group
UC RIVERSIDE

Auditing the Overheads of Serverless Data Plane

Processing involved in a typical serverless function chain setup

⑤ Message Broker ⇒ Function 2



Data Pipeline No.	External			Within chain				Total
	①	②	total	③	④	⑤	total	
# of copies	1	2	3	4	4	4	12	15
# of ctxt switches	1	2	3	4	4	4	12	15
# of irqs	3	4	7	6	6	6	18	25
# of proto. processing	1	2	3	3	3	3	9	12
# of serialization	0	1	1	2	2	2	6	7
# of deserialization	1	1	2	2	2	2	6	8

Networked Systems Group
UC RIVERSIDE