ACM Symposium on Cloud Computing 2024

Visit us at:
https://kknetsyslab.cs.ucr.edu/

SURE: Secure Unikernels Make Serverless Computing Rapid and Efficient

Federico Parola★ Shixiong Qi† Anvaya B. Narappa† K. K. Ramakrishnan† Fulvio Risso★

†University of California, Riverside ★Politecnico di Torino

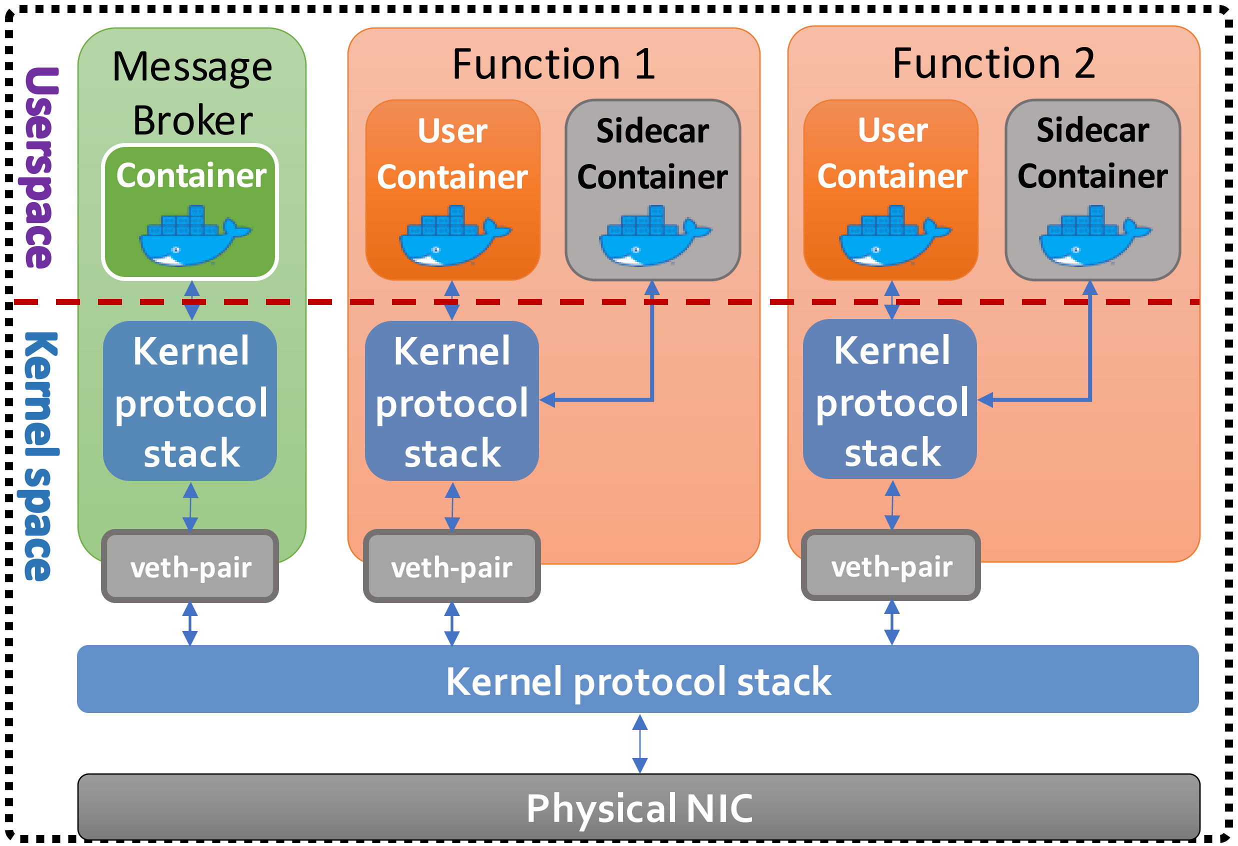
Serverless Computing

or Function-as-a-Service (FaaS)

- A **programming abstraction** that enables users to upload programs, run them at (**virtually**) any scale, and **pay only for the resources used**

- **Serverless support for loosely-coupled microservices**
  - ***Virtualized Runtime***
    - Fine-grained isolation at the individual function level
  - ***Inter-function networking***
    - Communication between decoupled functions
  - ***Service mesh and sidecar***
    - Facilitate orchestration of serverless functions in distributed environments

A Single Node View



Is today's serverless architecture enough support for loosely coupled microservices? – Isolation and Performance

State of the Landscape #1

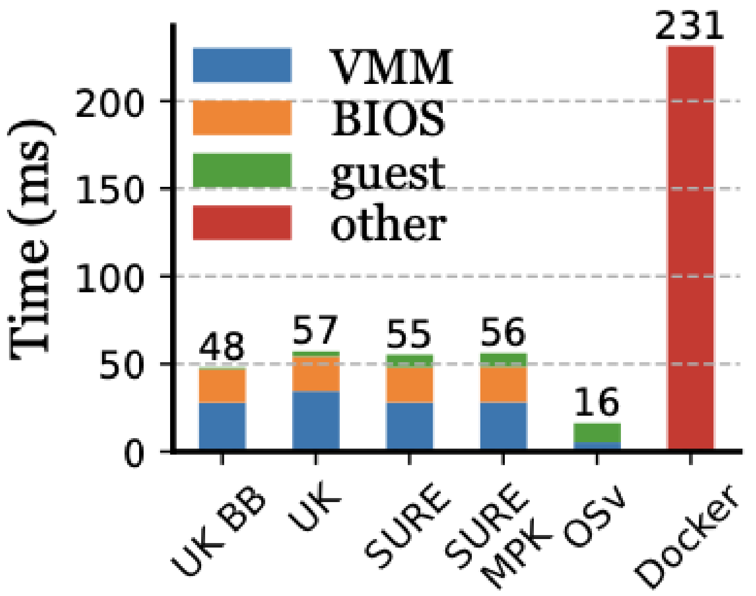
The tradeoff between isolation and agility in *virtualized runtime*

- Isolating serverless functions in open, shared cloud

Virtualized runtime	Isolation	Startup speed
Container	Weak ✗	Moderate ✓
Full-size VM	Strong ✓	Bad ✗
Unikernel	Strong ✓	Good ✓

- **Unikernel can make serverless functions agile and enable strong isolation**
  - *~4× faster startup compared to Docker containers*
- Unikernel offers **single address space**

Problem #1: Single-address-space unikernel is considered not safe



- UK **BB**: **B**are-**B**ones UniKraft
  - UK and SURE use **QEMU**
- OSv: OSv unikernel + **Firecracker**
- Docker: docker container

➢ Exploration of Unikernels for serverless and microservices

  ➢ USETL [**APSys'19**], UaaF [**IWQoS'20**], SEUSS [**EuroSys'20**], NanoVMs

  ➢ MirageOS [**ASPLOS'13**], OSv [**ATC'14**], LightVM [**SOSP'17**], Unikraft [**EuroSys'21**]

State of the Landscape #2

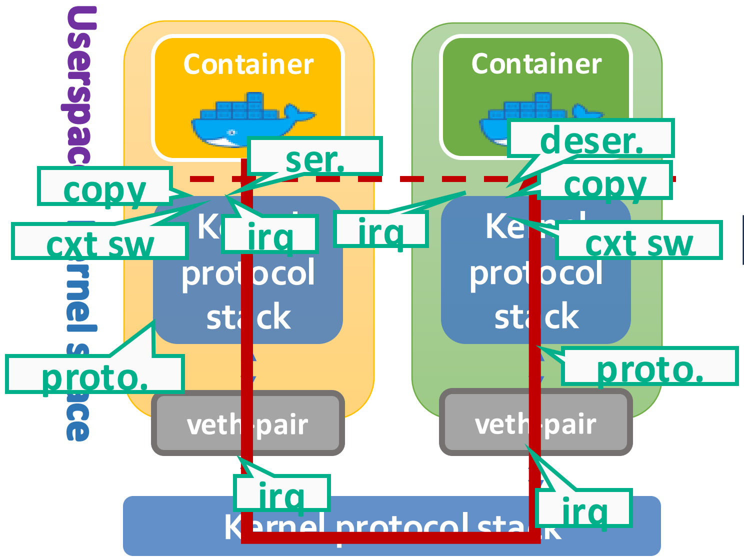
Inter-function networking in serverless computing

- Cost of kernel-based inter-function networking
  - Context switch, interrupt, copy, protocol processing, serialization/de-serialization
- Solution: shared memory processing
  - Faasm [ATC'20], SPRIGHT [SIGCOMM'22], Pheromone [NSDI'23], Ditto [SIGCOMM'23], YuanRong [SIGCOMM'24]
  - ***Pass-by-reference instead of pass-by-value***

Problem #1: Single-address-space unikernel is considered not safe

Problem #2: Shared memory processing is considered not safe

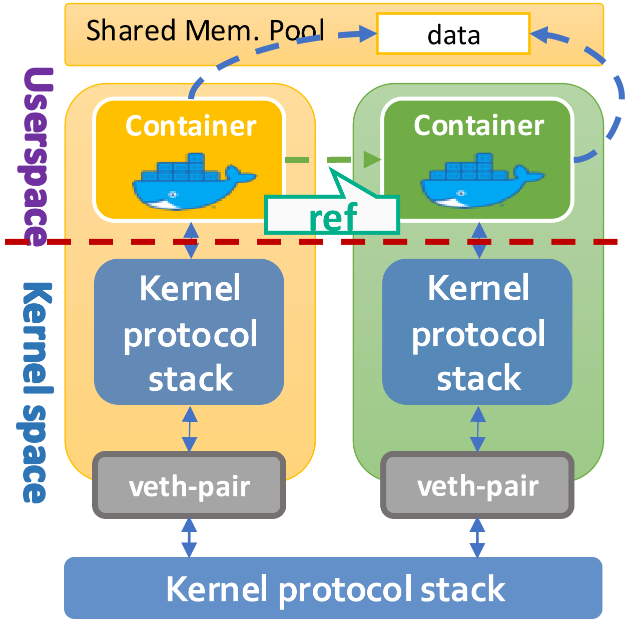
Problem#3: Shared memory processing is limited to a single node



Kernel-based Networking

Bypass the kernel!

Pass by reference!

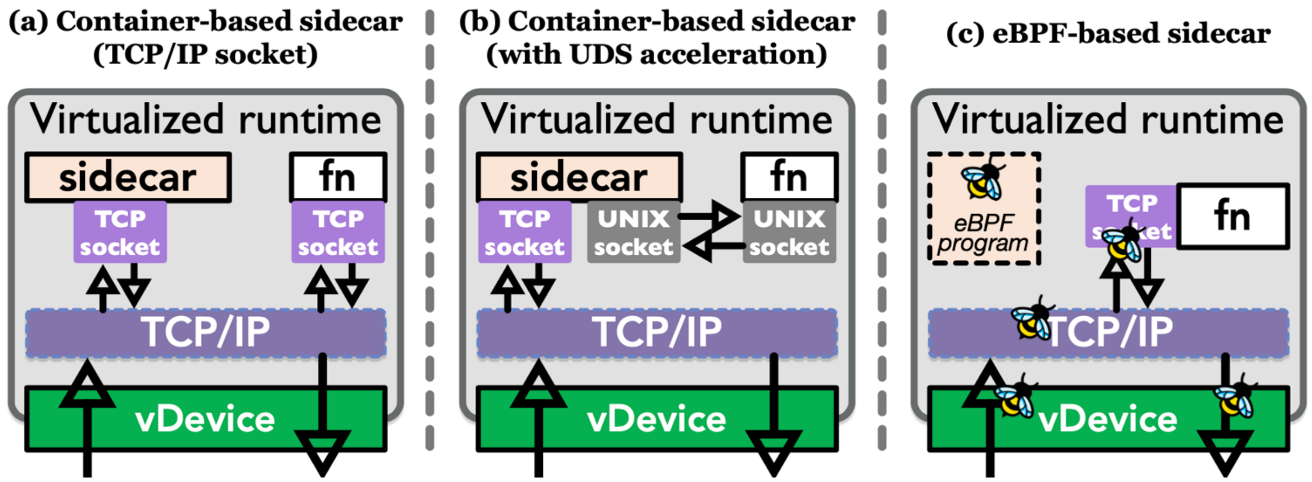


Shared Memory Processing

State of the Landscape #3

Service mesh support in serverless computing

- Existing design: sidecar is an ***individual*** component (e.g., container) independent of the function
  - Intermediated by the **TCP/IP stack** or by **Unix Domain Socket**
  - Incur unnecessary networking overheads
- Optimization: **eBPF-based sidecar**
  - SPRIGHT [SIGCOMM'22], Cilium
  - Attached to in-kernel eBPF hooks
  - No additional the userspace-kernel boundary crossing
  - No additional container networking overhead



***Problem #1:** Single-address-space unikernel is considered not safe*

***Problem #2:** Shared memory processing is considered not safe*

***Problem#3:** Shared memory processing is limited to a single node*

***Problem#4:** eBPF is not suitable for unikernels*

Our solution SURE

Secure Unikernels Make Serverless Computing Rapid and Efficient

Unikernels

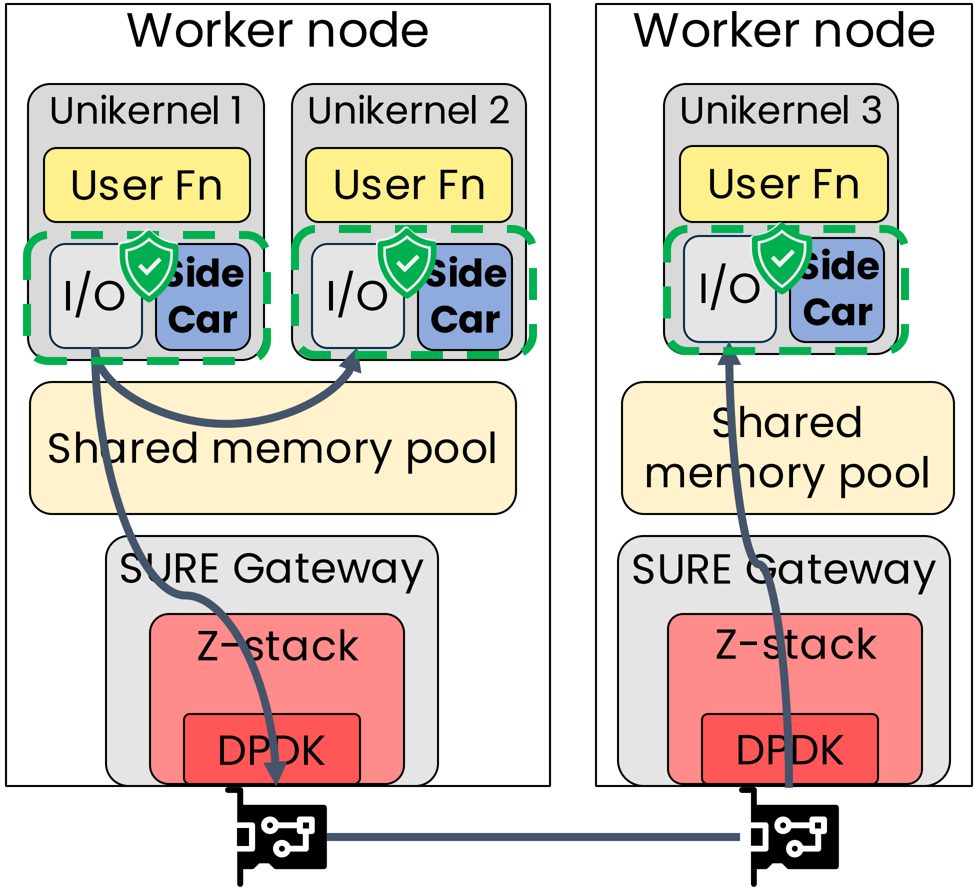
Shared-memory intra-node data plane

Zero-copy inter-node TCP/IP stack (**Z-stack**)

Consolidated proto. processing by SURE Gateway

Library-based sidecar

MPK-based call gate



***Design#1:** Secure shared memory while retaining its high performance*

***Design#2:** Enhance intra-unikernel isolation to sandbox user code*

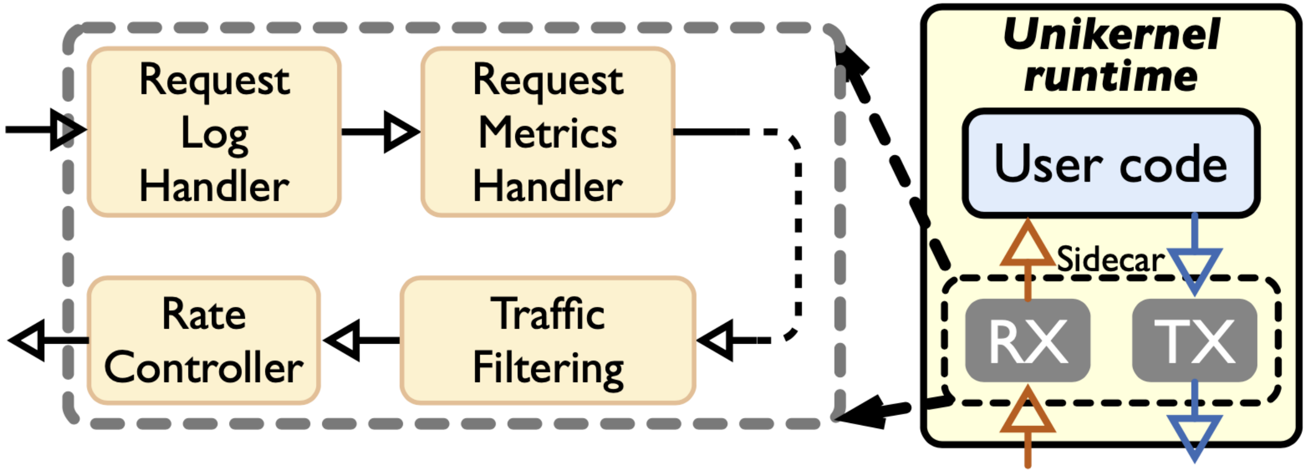
***Design#3:** Extended zero-copy networking to be distributed*

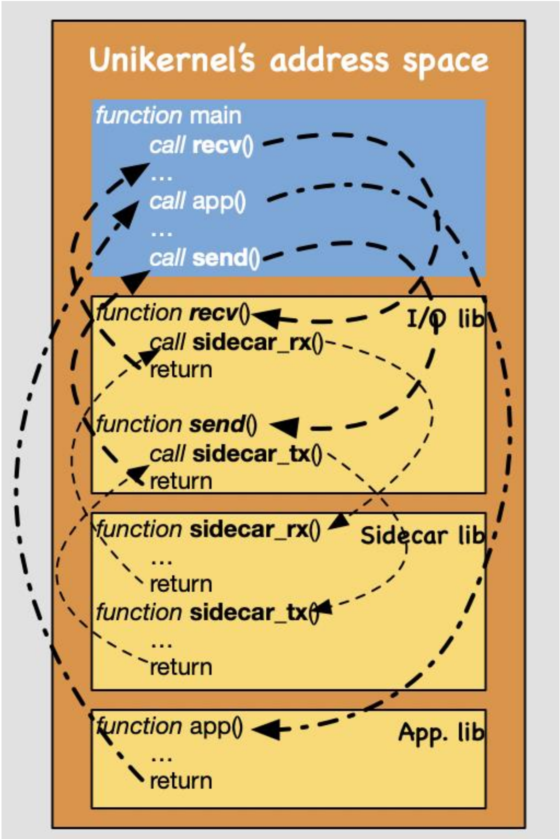
***Design#4:** eBPF-like sidecar with L7 visibility in unikernels*

Library-based SURE Sidecar

Based on the LibOS design of unikernels

- Deploy the sidecar as a **library** linked into the function code within the unikernel
  - The sidecar contains a sequence of handlers that perform certain sidecar functionalities



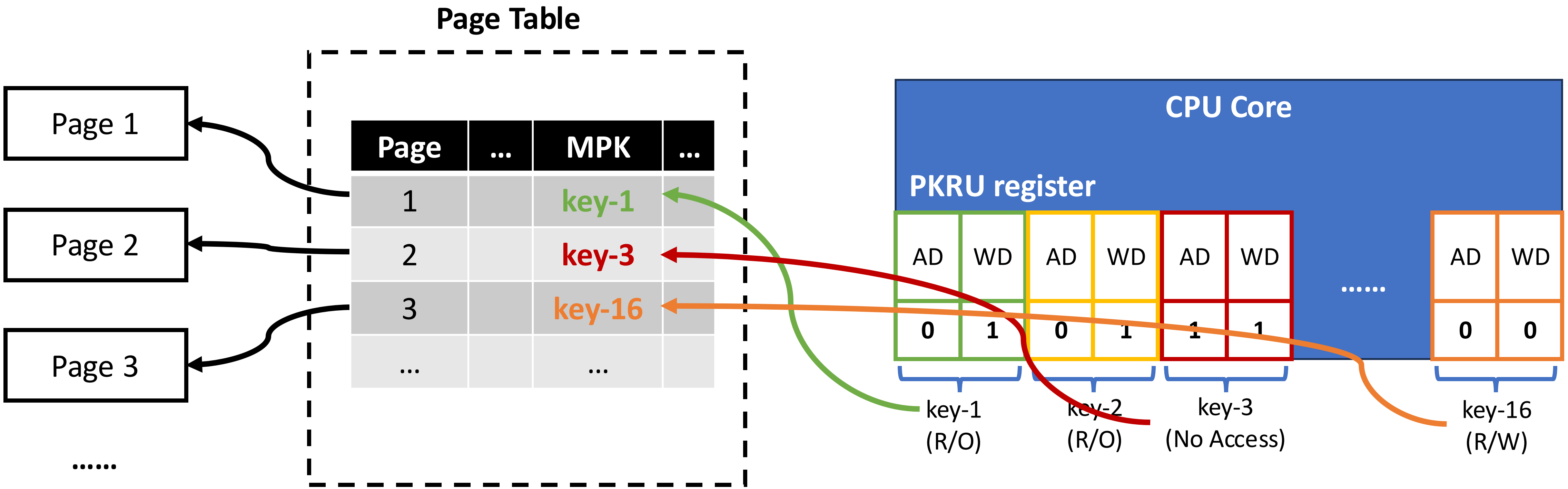


- The unikernel's **single-address-space** simplifies data exchange between sidecar and user code
  - Invocation is made by procedure call
  - overcome the shortcomings of an individual userspace sidecar.

Memory-level isolation in SURE

A Primer on MPK (Memory Protection Key)

- MPK is a hardware-level, intra-process memory isolation feature in Intel's server CPUs (since 2019)
- **PKRU (Protection Key Register User)**
  - A per-core, 32-bit CPU register defines the access privilege of MPK, described by 2 bits
    - "Access Disable" (**AD**) and "Write Disable" (**WD**)
  - A total of **16** keys available within a SURE function
  - **Read/Write (0, 0), Read-Only (0, 1), or No-Access (1, ×)**

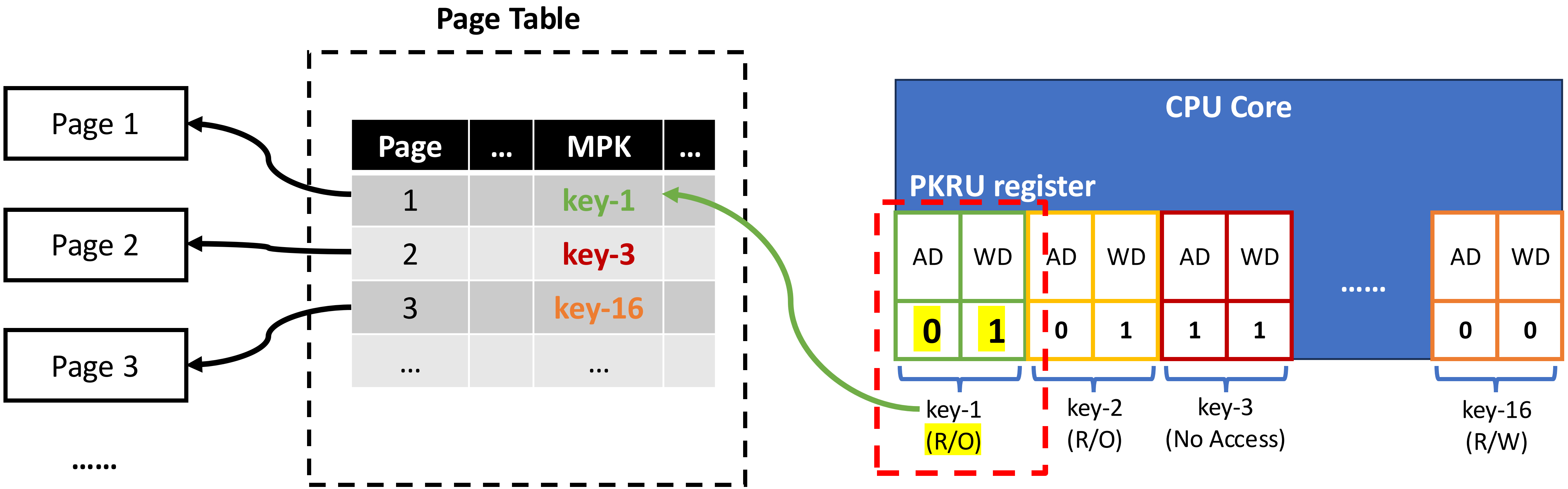


Memory-level isolation in SURE

SURE uses *two* approaches to switch the access privilege of a memory page

- **#1 WRPKRU** (Write Data to PKRU)
  - x86 instruction to change the access privilege of the MPK by modifying PKRU
  - ***A SURE function may access more than 16 pages!***
  - ***Not feasible to tag each page with a distinct key***
- **Memory related to Unikernel TCB components is managed by WRPKRU** Coarse-grained but faster

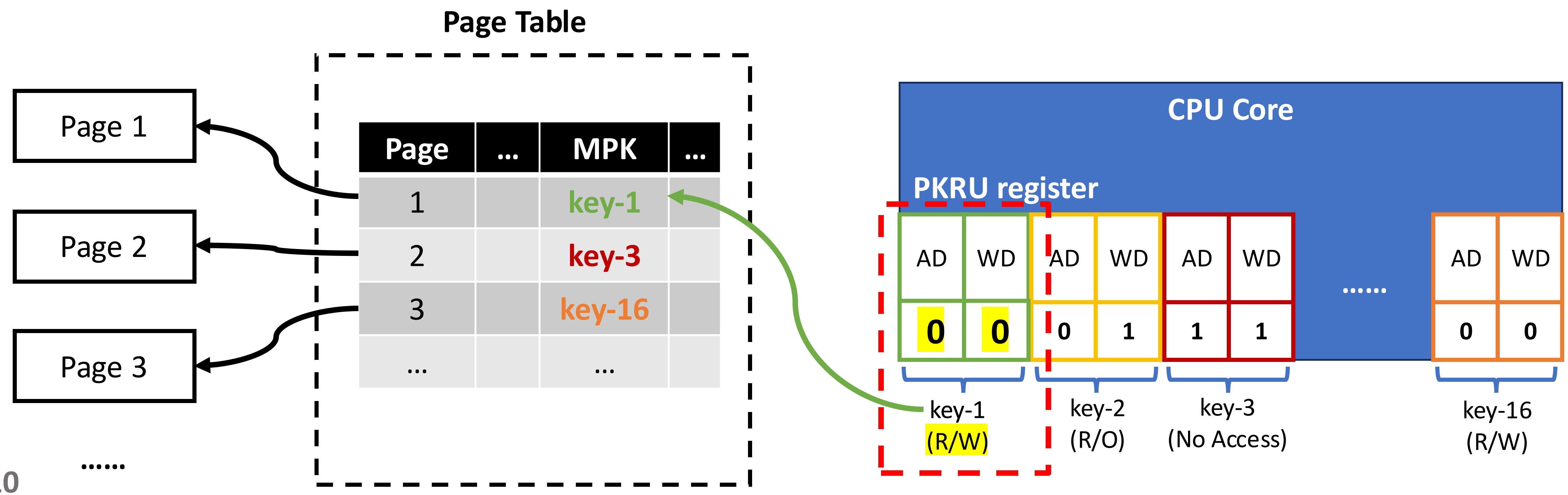
- **#2 "PTE Update"**
  - Update the 4 bits reserved for the MPK key ID in the PTE
  - Then flush the corresponding TLB entry
  - Allow for more ***scalable*** access management
- **Shared memory buffers are managed by "PTE Update"** Fine-grained but slower



Memory-level isolation in SURE

SURE uses *two* approaches to switch the access privilege of a memory page

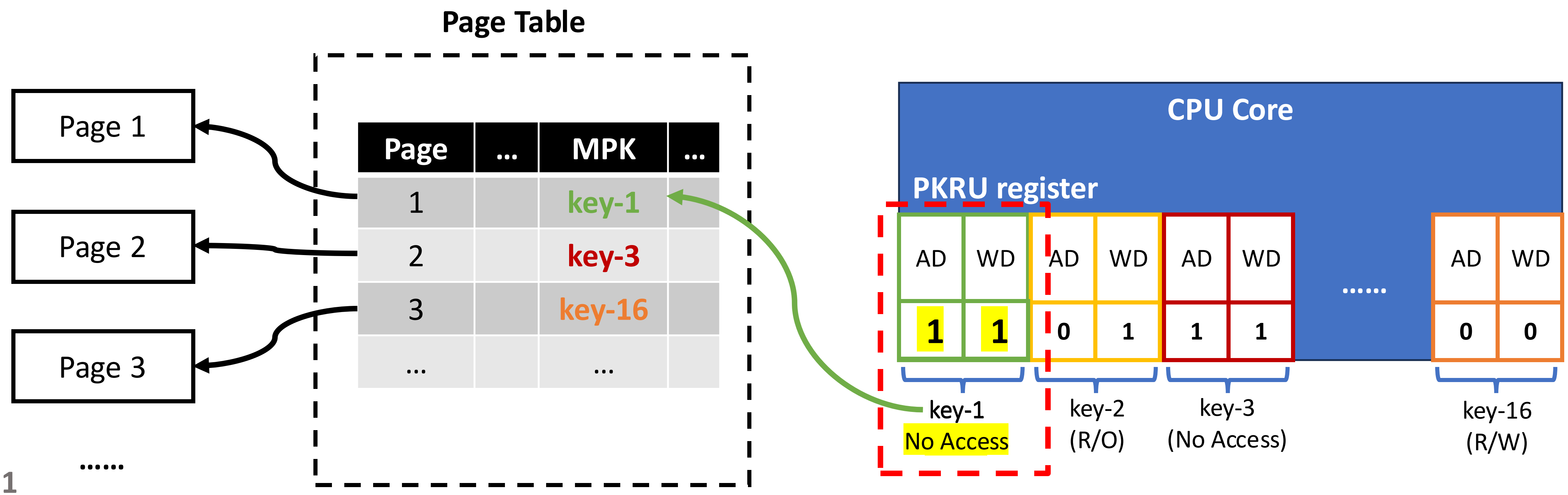
- **#1 WRPKRU** (Write Data to PKRU)
  - x86 instruction to change the access privilege of the MPK by modifying PKRU
  - ***A SURE function may access more than 16 pages!***
  - ***Not feasible to tag each page with a distinct key***
- **Memory related to Unikernel TCB components is managed by WRPKRU** Coarse-grained but faster
- **#2 "PTE Update"**
  - Update the 4 bits reserved for the MPK key ID in the PTE
  - Then flush the corresponding TLB entry
  - Allow for more ***scalable*** access management
- **Shared memory buffers are managed by "PTE Update"** Fine-grained but slower



Memory-level isolation in SURE

SURE uses *two* approaches to switch the access privilege of a memory page

- **#1 WRPKRU** (Write Data to PKRU)
  - x86 instruction to change the access privilege of the MPK by modifying PKRU
  - *A SURE function may access more than 16 pages!*
  - *Not feasible to tag each page with a distinct key*
- **Memory related to Unikernel TCB components is managed by WRPKRU** Coarse-grained but faster
- **#2 "PTE Update"**
  - Update the 4 bits reserved for the MPK key ID in the PTE
  - Then flush the corresponding TLB entry
  - Allow for more ***scalable*** access management
- **Shared memory buffers are managed by "PTE Update"** Fine-grained but slower

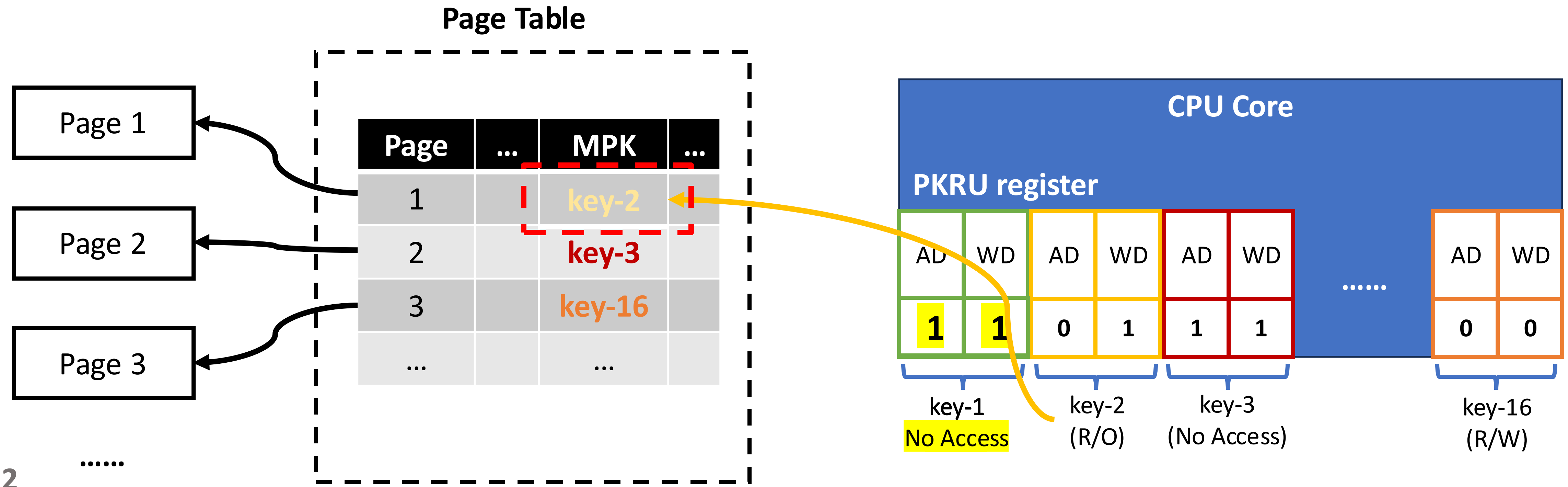


Memory-level isolation in SURE

SURE uses *two* approaches to switch the access privilege of a memory page

- **#1 WRPKRU** (Write Data to PKRU)
  - x86 instruction to change the access privilege of the MPK by modifying PKRU
  - *A SURE function may access more than 16 pages!*
  - *Not feasible to tag each page with a distinct key*
- **Memory related to Unikernel TCB components is managed by WRPKRU** Coarse-grained but faster

- **#2 "PTE Update"**
  - Update the 4 bits reserved for the MPK key ID in the PTE
  - Then flush the corresponding TLB entry
  - Allow for more ***scalable*** access management
- **Shared memory buffers are managed by "PTE Update"** Fine-grained but slower



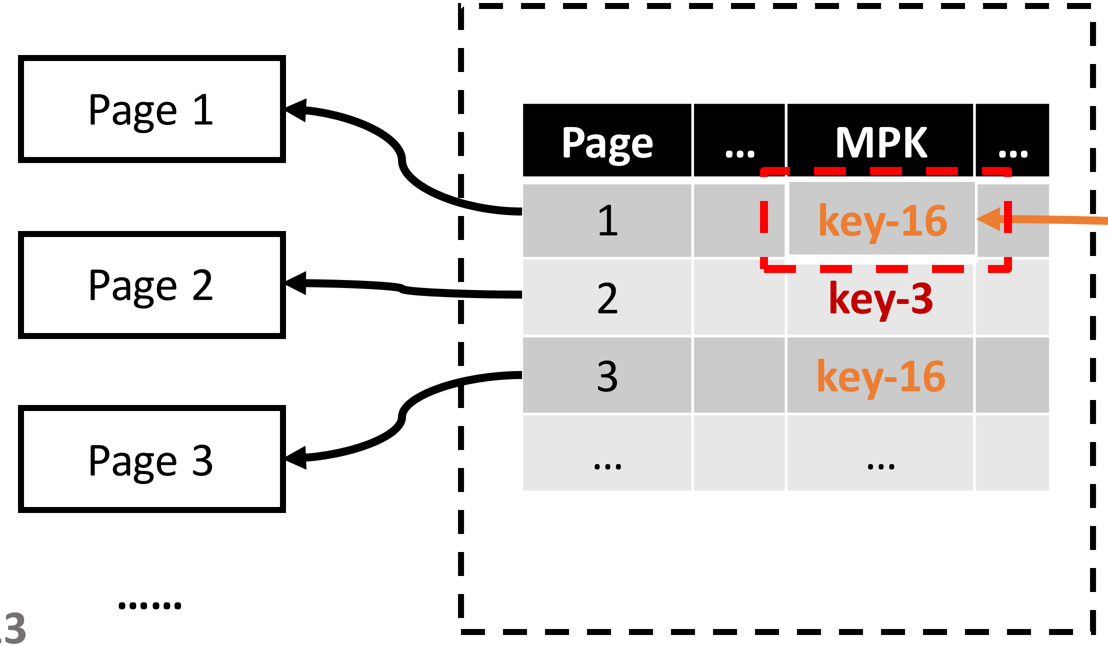
Memory-level isolation in SURE

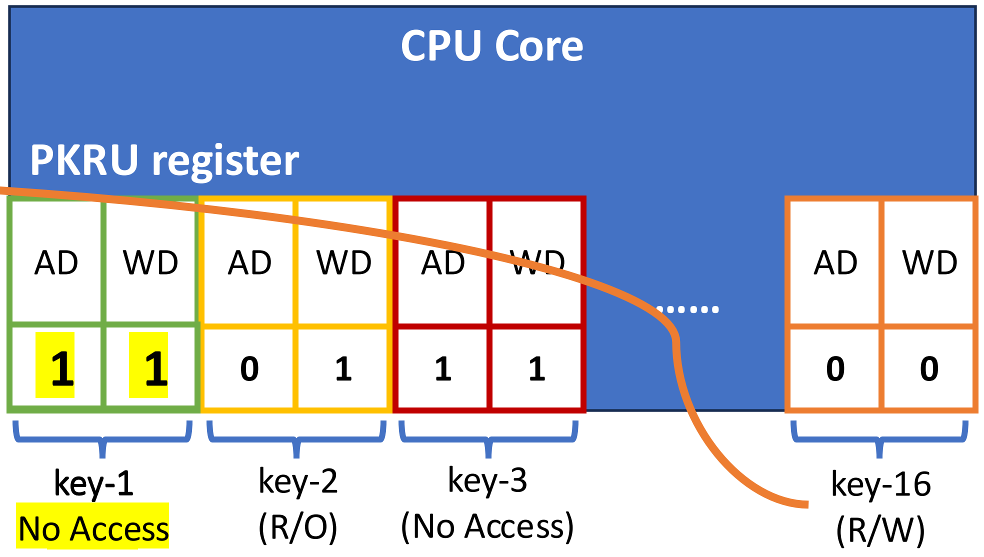
SURE uses *two* approaches to switch the access privilege of a memory page

- **#1 WRPKRU** (Write Data to PKRU)
  - x86 instruction to change the access privilege of the MPK by modifying PKRU
  - ***A SURE function may access more than 16 pages!***
  - ***Not feasible to tag each page with a distinct key***
- **Memory related to Unikernel TCB components is managed by WRPKRU** Coarse-grained but faster

- **#2 "PTE Update"**
  - Update the 4 bits reserved for the MPK key ID in the PTE
  - Then flush the corresponding TLB entry
  - Allow for more ***scalable*** access management
- **Shared memory buffers are managed by "PTE Update"** Fine-grained but slower

Page Table

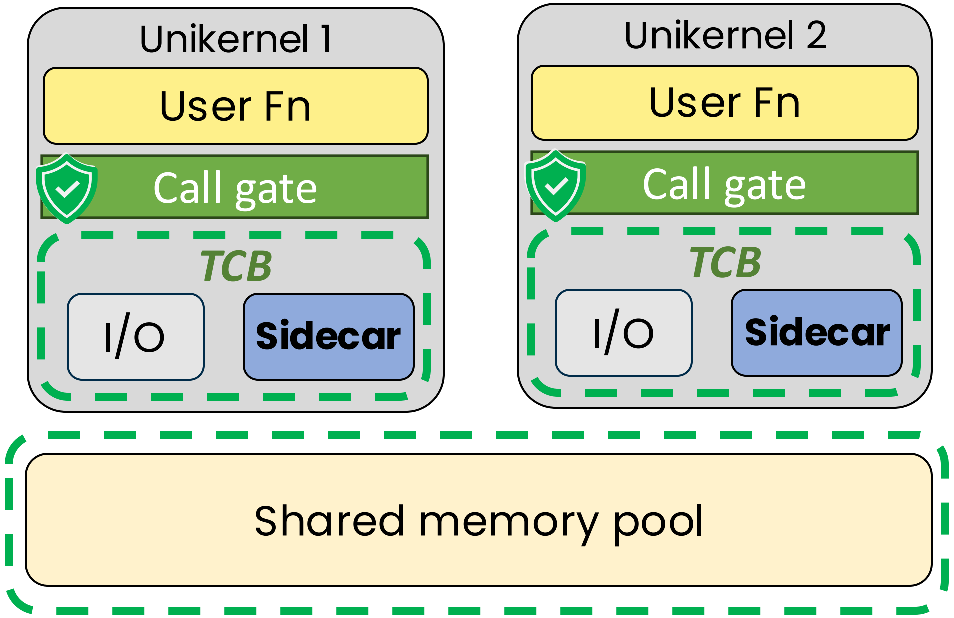




Secure APIs based on SURE call gates

A "call gate" abstraction for user code to safely interact with protected pages

- **Only call gate can update access privilege**
  - Via WRPKRU or PTE Update
  - Easier to work with ***binary inspection*** to prohibit illegal updates to access privilege
- **Enhanced unikernel TCB (from Unikraft) in SURE**
  - Prevent unwanted update or access to PKRU register and PTEs of protected pages
  - Avoid ***Privilege Escalation*** of MPK in a single address space
  - ***Refer to the paper***

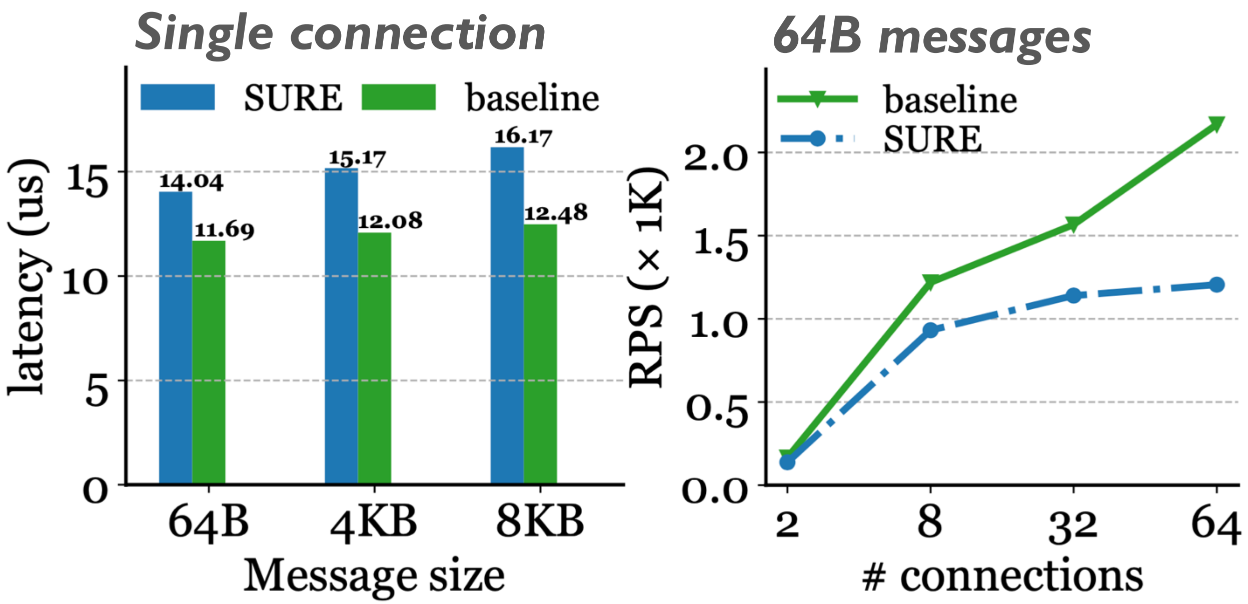


Microbenchmark Analysis

Cost of Memory Isolation with SURE

Baseline: a variant of SURE with MPK ***disabled***

- **MPK in SURE has limited penalty**
  - **With a single connection**: SURE shows 1.2-1.3× increased delay compared to the baseline.
  - **With increasing concurrent connection**: SURE's RPS decreases (e.g., 1.8× reduction at 64 connections)
- ***Relatively small overhead for the reward of robust memory-level isolation***



 Testbed: *One sm110p node on Cloudlab (with 100 Gbps NIC); Ubuntu 22.04; kernel 5.15*

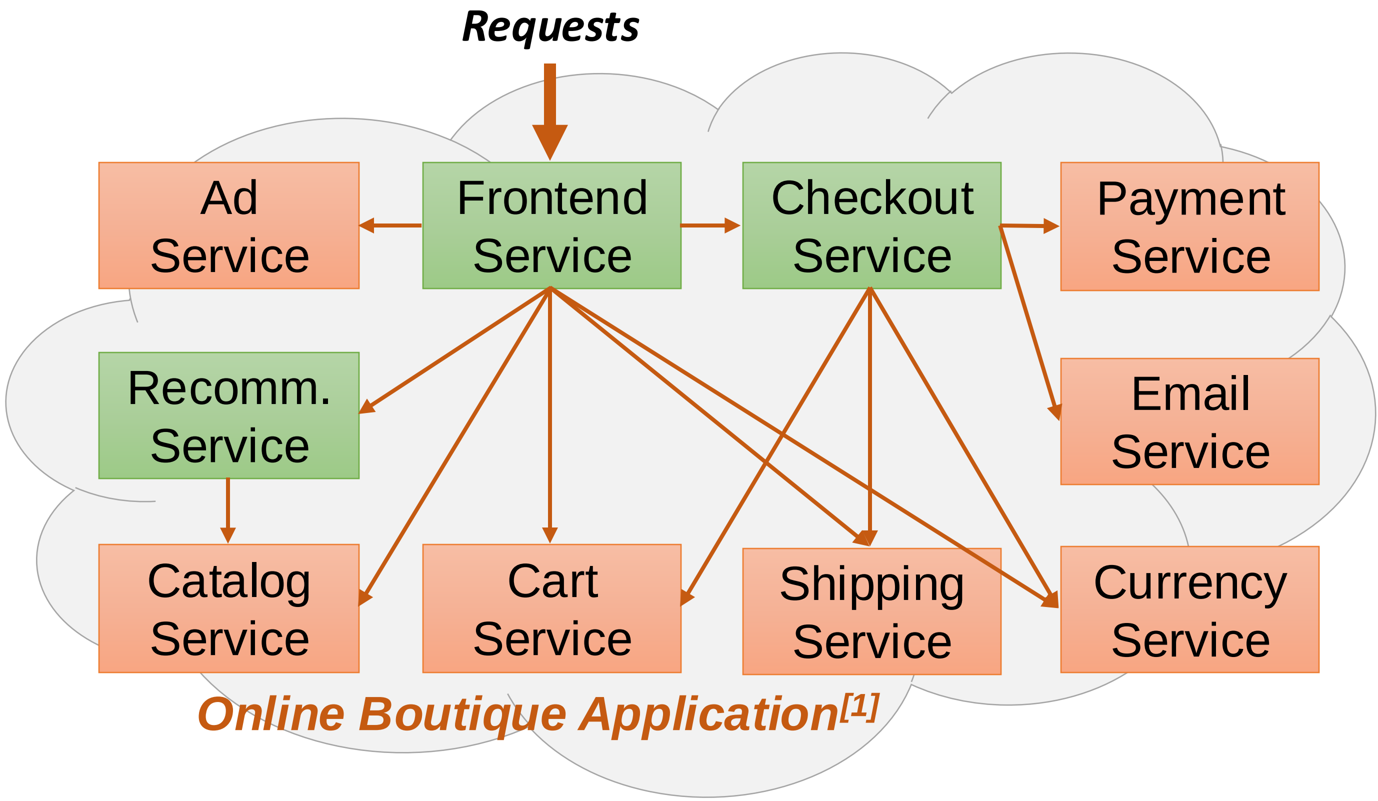
[1] https://github.com/GoogleCloudPlatform/microservices-demo

Realistic Workload Evaluation

Experiment setting

Online Boutique Microservice Chain [1]

- *Intense* web workload with 10 functions
- 6 different function chains



Serverless Alternatives

- **Knative**
- **SPRIGHT [SIGCOMM'22]**
- **NightCore [ASPLOS'21]**

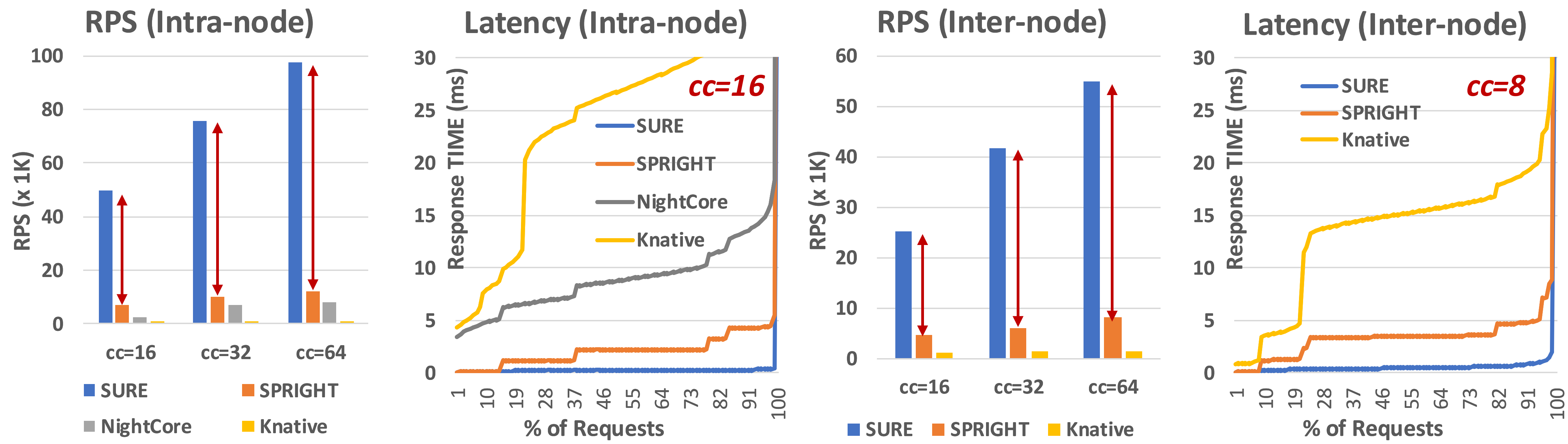
Two distinct deployment settings:

1) **Intra-node**
2) **Inter-node: Orange and Green functions deployed on distinct nodes**

 ***Testbed**: Three sm110p nodes on Cloudlab (with 100 Gbps NIC); Ubuntu 22.04; kernel 5.15*

Realistic Workload Evaluation

Requests per second & Tail latency



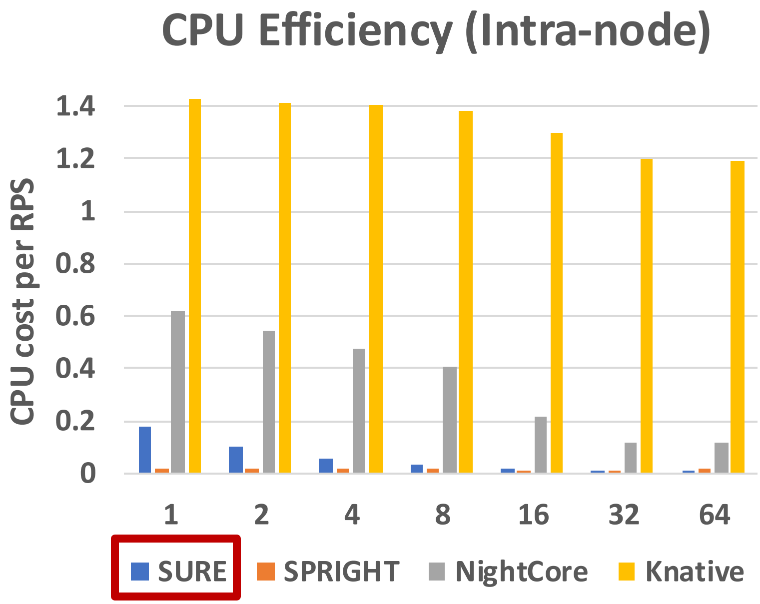
- **SURE is an order of magnitude better than any alternatives we evaluated**
- **Performance improvement attributed to the use of *distributed zero-copy data plane* and lightweight *library-based sidecar***

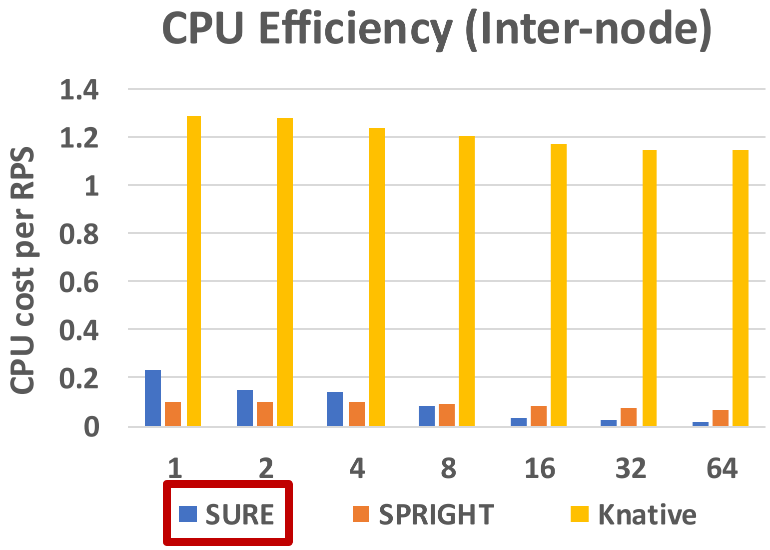
Realistic Workload Evaluation

CPU efficiency

- Our metric - "CPU Cost Per RPS" (**CCR**)
  - Defined as $\frac{Average\ CPU\ utilization}{RPS}$
  - ***Lower*** values of **CCR** suggest that each request requires ***fewer*** CPU cycles
    - ***A more efficient use of the CPU***

- **SURE** is more efficient than **NightCore** and **Knative**
  - No kernel networking; More lightweight sidecar; etc
- **SURE** is less efficient than **SPRIGHT** at a low concurrency (≤ 16 for intra-node and ≤ 4 for inter-node)
  - Comes from polling cost
- **SURE** is more efficient than **SPRIGHT** under high concurrency levels
  - **SPRIGHT** uses kernel for inter-node traffic, CPU usage grows substantially under high concurrency levels
  - More concurrent processing amortizes the polling cost





Worse efficiency

Better efficiency

Horizontal axis: concurrency

Conclusion

- **SURE is a unikernel-based, lightweight serverless framework**
  - Unikernel-based runtime brings **4× faster startup** VS. docker containers
  - MPK-based call gates to enable **fine-grained memory access management**
    - Mitigate the vulnerabilities of **memory space sharing**
    - While retaining high performance and efficiency
  - Offer **zero-copy** inter-function networking and lightweight **library**-based sidecars
    - ***Yield up to 8× RPS improvement compared to SPRIGHT in a distributed environment***
    - ***While being more secure***

- **SURE is open-sourced**
  - ☞ **Find SURE at: https://github.com/ucr-serverless/sure**
  - **If you have any questions or comments, please feel free to email us (federico.parola@polito.it and shixiong.qi@uky.edu)**

Paper:

Code: