USENIX ATC'24

ALPS:

An Adaptive Learning, Priority OS Scheduler for Serverless Functions

Yuqi Fu[1], Ruizhe Shi[2], Haoliang Wang[3], Songqing Chen[2], Yue Cheng[1]

1 University of Virginia

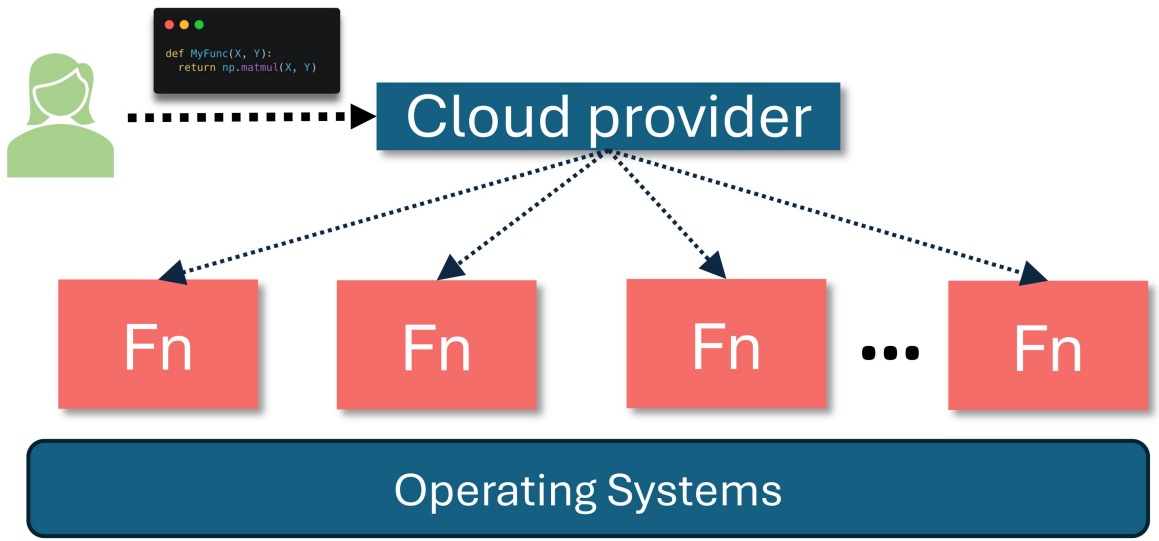
2 George Mason University

3 Adobe

Serverless Computing

Function-as-a-Service (FaaS): Cloud function as a basic deployment unit.

Main benefits: Scalability; Pay as you go; DevOps cost



AWS Lambda

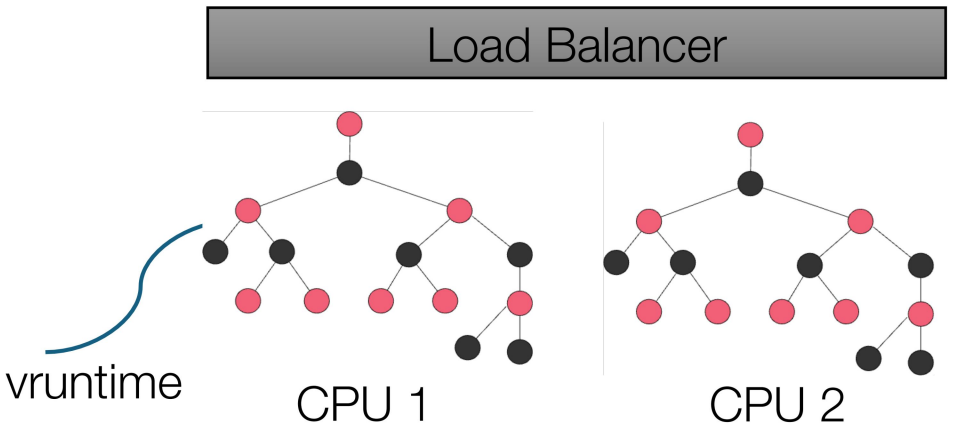
Azure Functions

Google Cloud Functions

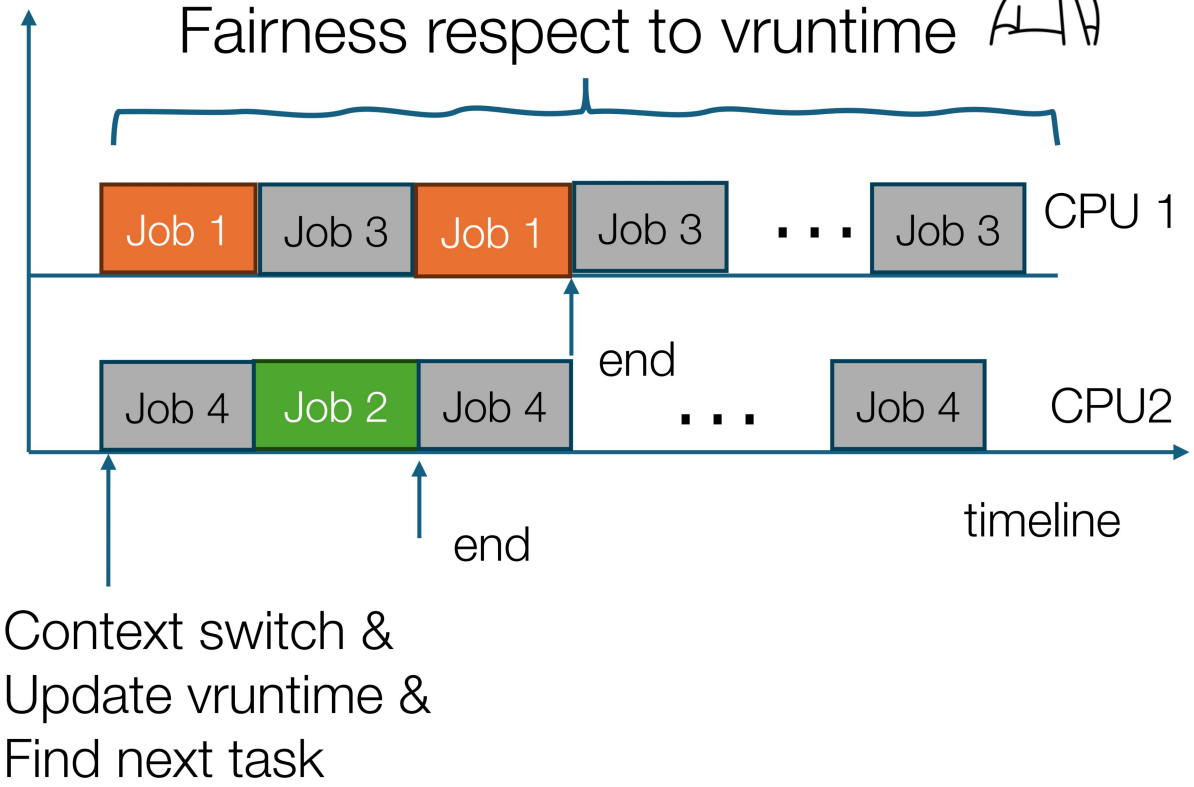
Alibaba Function Compute

Current Scheduler Limitations

CFS (Completely Fair Scheduler)
- Proportional-share time slice
- Default Linux scheduler



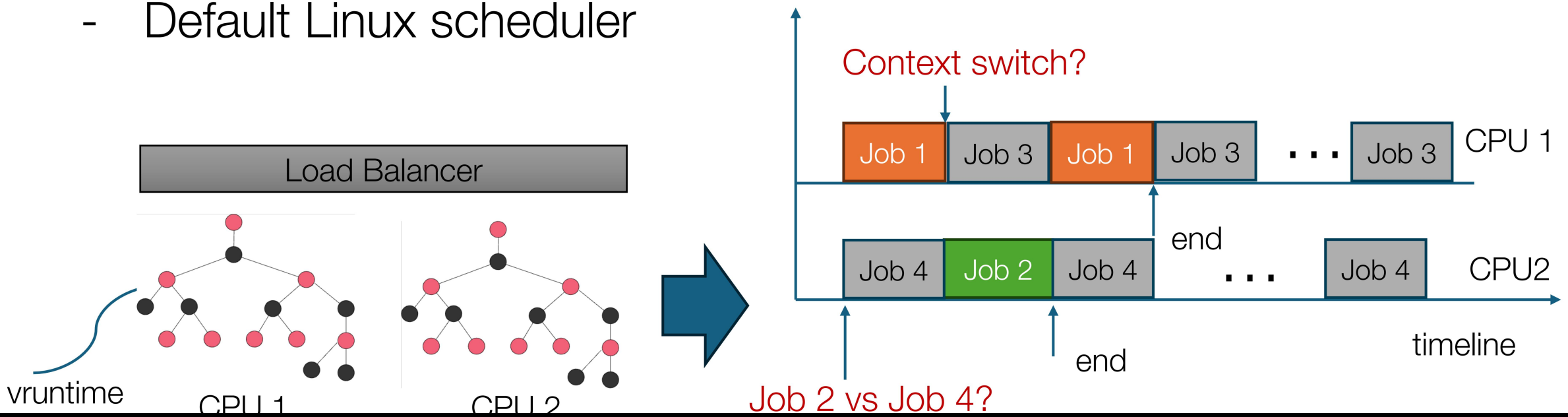
Short jobs: 1, 2
Long jobs: 3, 4



Current Scheduler Limitations

CFS (Completely Fair Scheduler)
- Proportional-share time slice
- Default Linux scheduler

Short jobs: 1, 2
Long jobs: 3, 4



- **Implication: CFS lacks application workload intelligence**

Approximating SRPT

SRPT (Shortest Remaining Processing Time)

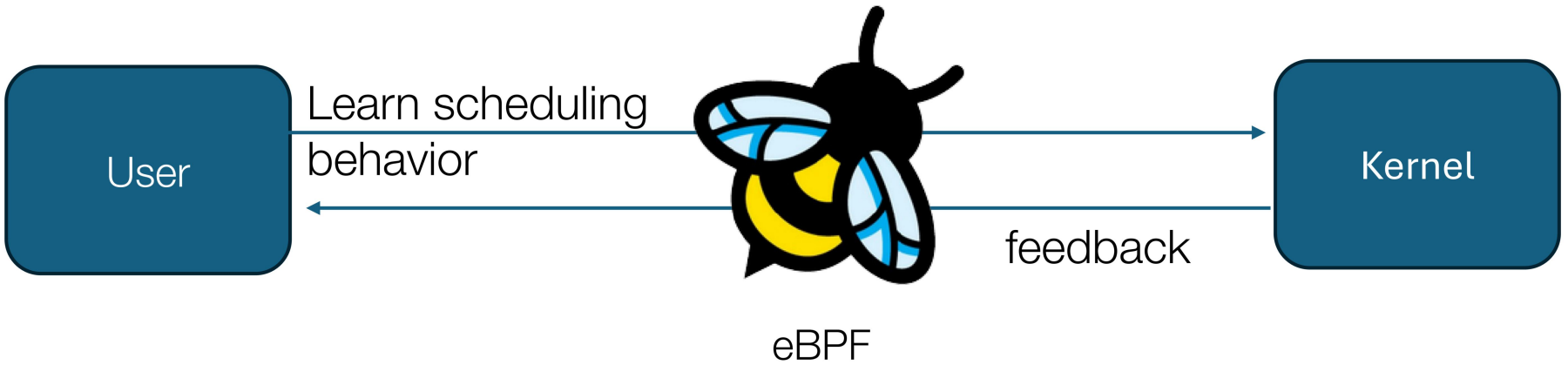
- provably optimal for average performance
- impractical in online scheduling
- cause CPU resource starvation for long-term jobs

- **Implication: SRPT is impractical and causes starvation for long jobs**

Outline

- Motivation
- ALPS design and implementation
- Evaluation

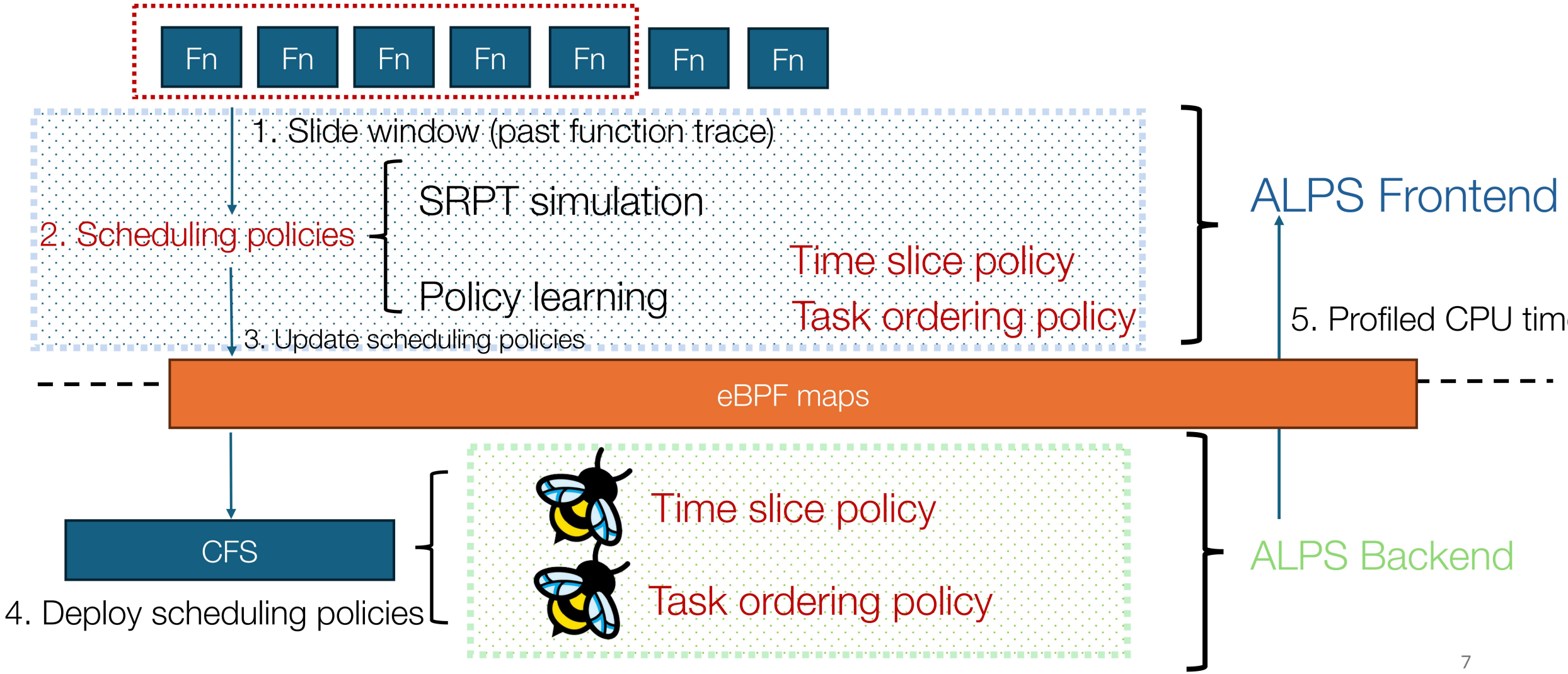
Birds’ eye view of ALPS design



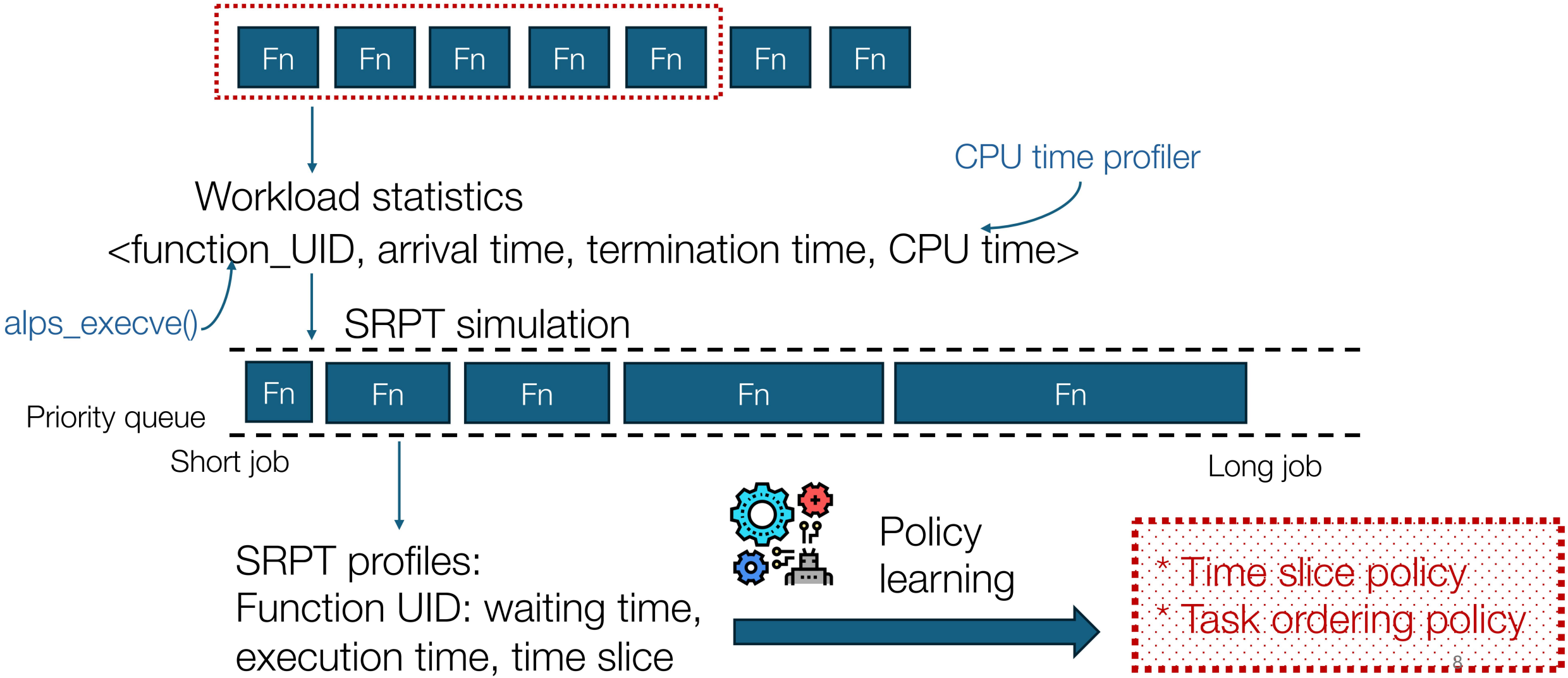
ALPS has a novel OS scheduler architecture that decouples a user-space frontend and kernel-space backend

- Policy: Time slice policy & time ordering policy
- Mechanism: Leveraging eBPF for user-kernel-space communication

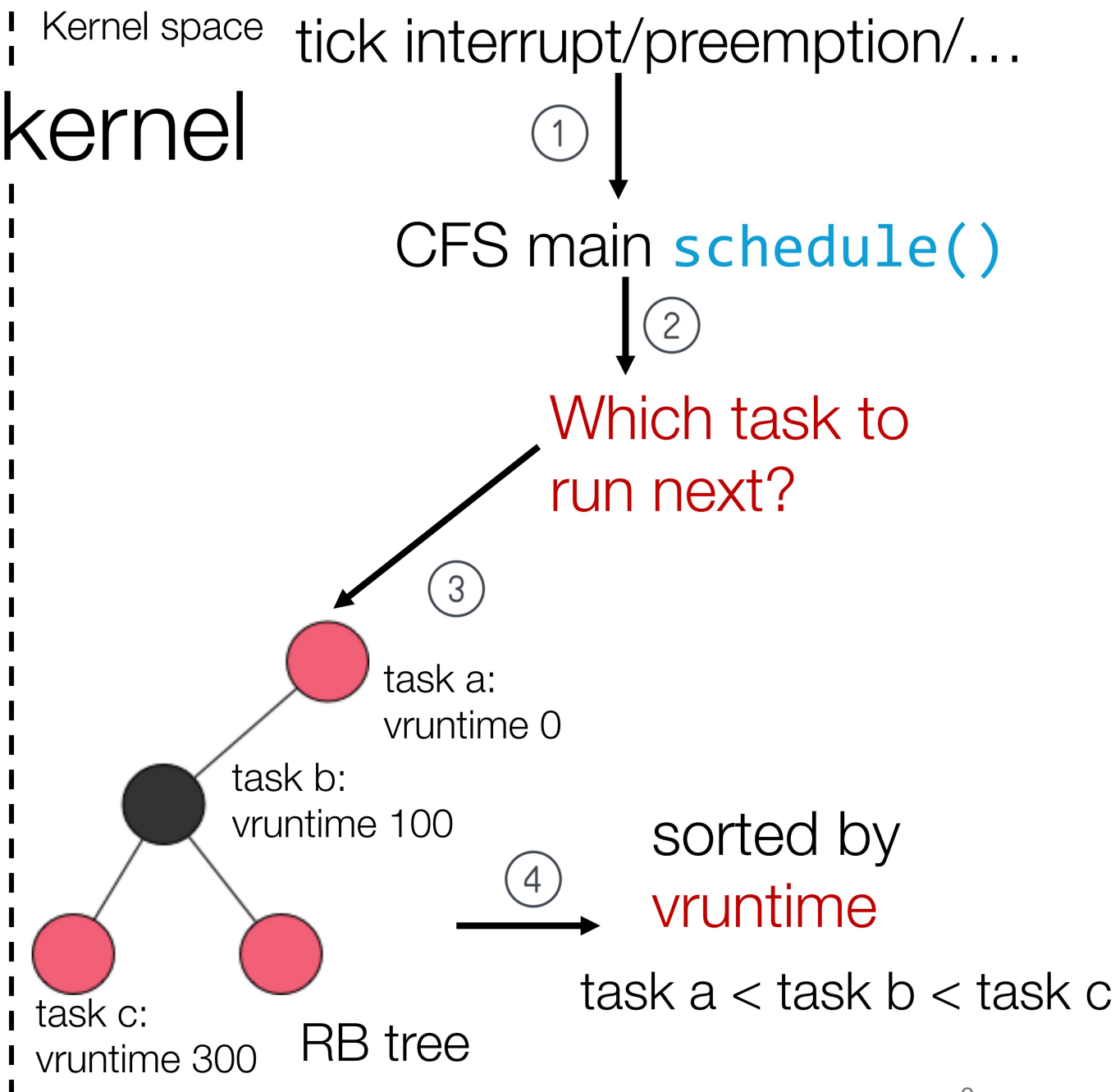
ALPS Design


Fn
Fn
Fn
Fn
Fn
Fn
Fn
1. Slide window (past function trace)
2. Scheduling policies
SRPT simulation
Policy learning
3. Update scheduling policies
Time slice policy
Task ordering policy
ALPS Frontend
5. Profiled CPU time
eBPF maps
CFS
4. Deploy scheduling policies
Time slice policy
Task ordering policy
ALPS Backend

ALPS frontend: SRPT simulation



CFS workflow in the kernel



ALPS' time slice policy

User space | Kernel space

tick interrupt/preemption/…

① CFS main `schedule()`

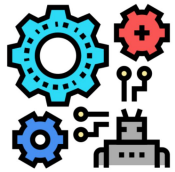
② Which task to run next?

③

④ sorted by vruntime

task a < task b < task c

SRPT simulation profiles



Policy learning

Expected time slice

upper bound for function

e.g. task a: 40ms; task b: 8ms …

deploy

eBPF maps

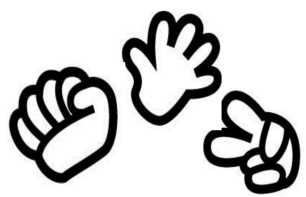
task a: vruntime 0

task b: vruntime 100

task c: vruntime 300

RB tree

Current time slice

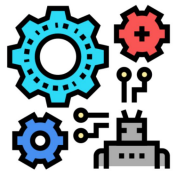


upper bound time slice

User space | Kernel space

ALPS' time slice policy

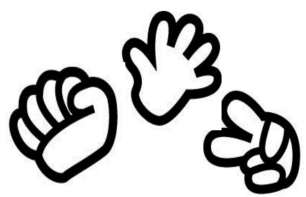
SRPT simulation profiles



Policy learning

Expected time slice
upper bound for function
e.g. task a: 40ms; task b: 8ms …

Current time slice: 40 ms



Greater than

upper bound time slice: 8ms

eBPF maps

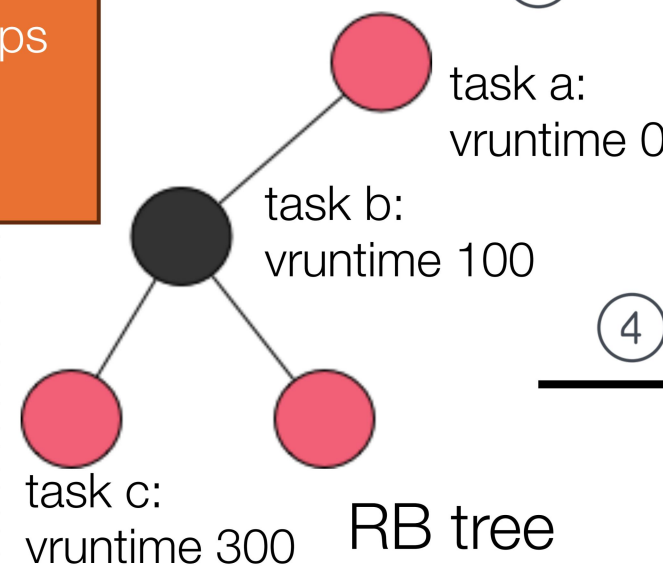
deploy

tick interrupt/preemption/…

① CFS main `schedule()`

② CFS branch:
pick the task with
minimal vruntime

③



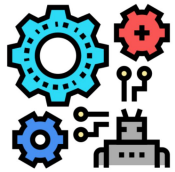
④ sorted by
vruntime

task a < task b < task c

ALPS' time slice policy

User space | Kernel space

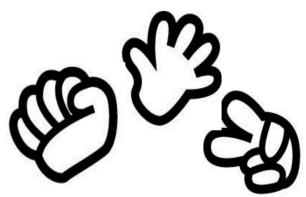
SRPT simulation profiles



Policy learning

Expected time slice
upper bound for function
e.g. task a: 40ms; task b: 8ms …

Current time slice: 8 ms



Less than

upper bound time slice: 40ms

eBPF maps

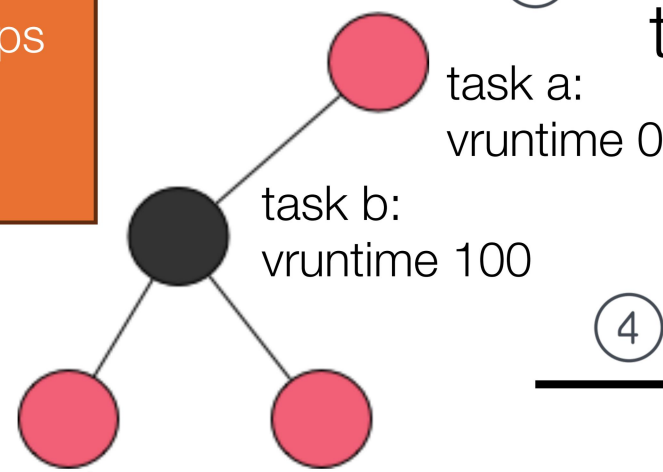
deploy

tick interrupt/preemption/…

① CFS main `schedule()`

② **ALPS branch:**
grant additional time slice to current task

③



RB tree

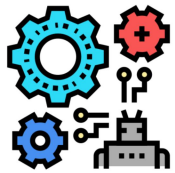
④ sorted by vruntime

task a < task b < task c

ALPS' task ordering policy

User space | Kernel space

SRPT simulation profiles



Policy learning

Expected ALPS task priority for function
e.g. task a: 2; task b: 1 …
ALPS priority overrides CFS vruntime

eBPF maps

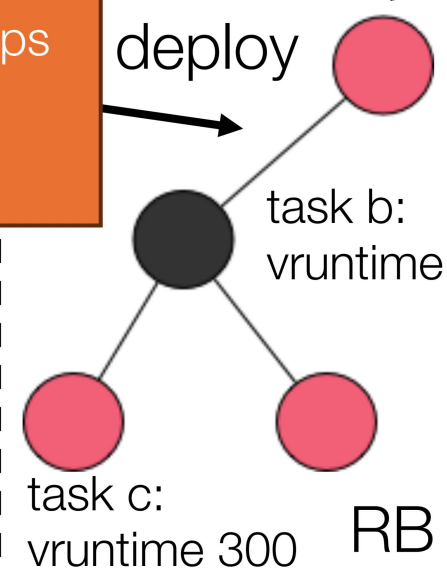
deploy

tick interrupt/preemption/…

① CFS main `schedule()`

② Which task to run next?

③



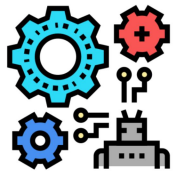
④ sorted by vruntime

task a < task b < task c

ALPS' task ordering policy

User space | Kernel space

SRPT simulation profiles



Policy learning

Expected ALPS task priority for function
e.g. task a: 2; task b: 1 …
ALPS priority overrides CFS vruntime

eBPF maps

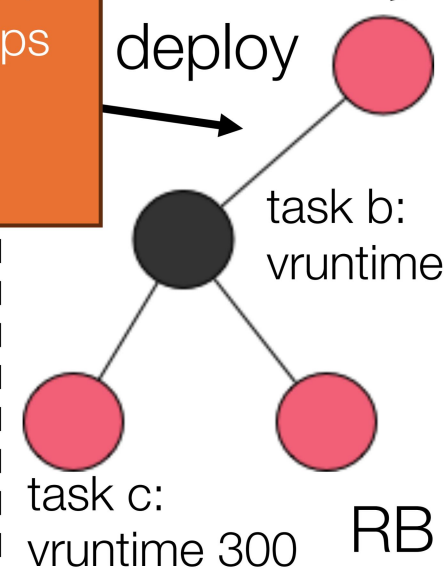
deploy

tick interrupt/preemption/…

① CFS main `schedule()`

② Which task to run next?

③



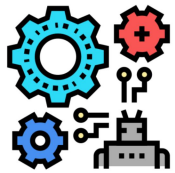
④ sorted by **ALPS priority**

task b < task a < task c

ALPS' task ordering policy

User space | Kernel space

SRPT simulation profiles



Policy learning

Expected ALPS task priority for function
e.g. task a: 2; task b: 1 …
ALPS priority overrides CFS vruntime

eBPF maps

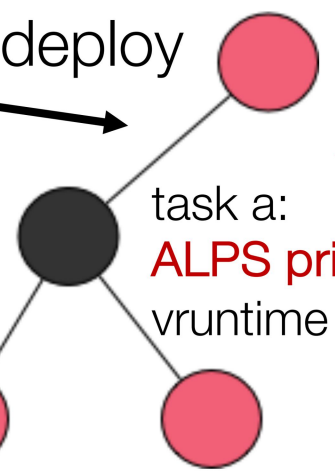
deploy

tick interrupt/preemption/…

① CFS main `schedule()`

② Which task to run next?

③



RB tree

④ sorted by **ALPS priority**

task b < task a < task c

Outline

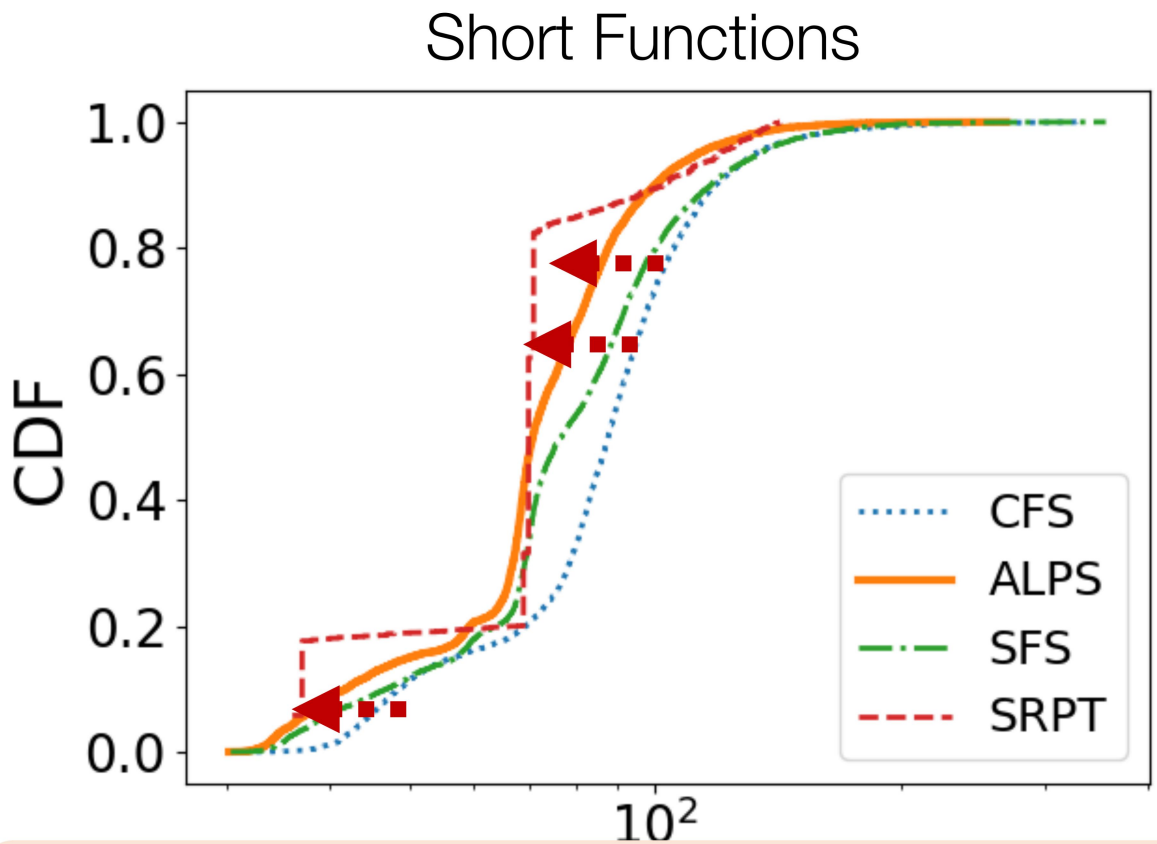
- Motivation
- ALPS design and implementation
- Evaluation

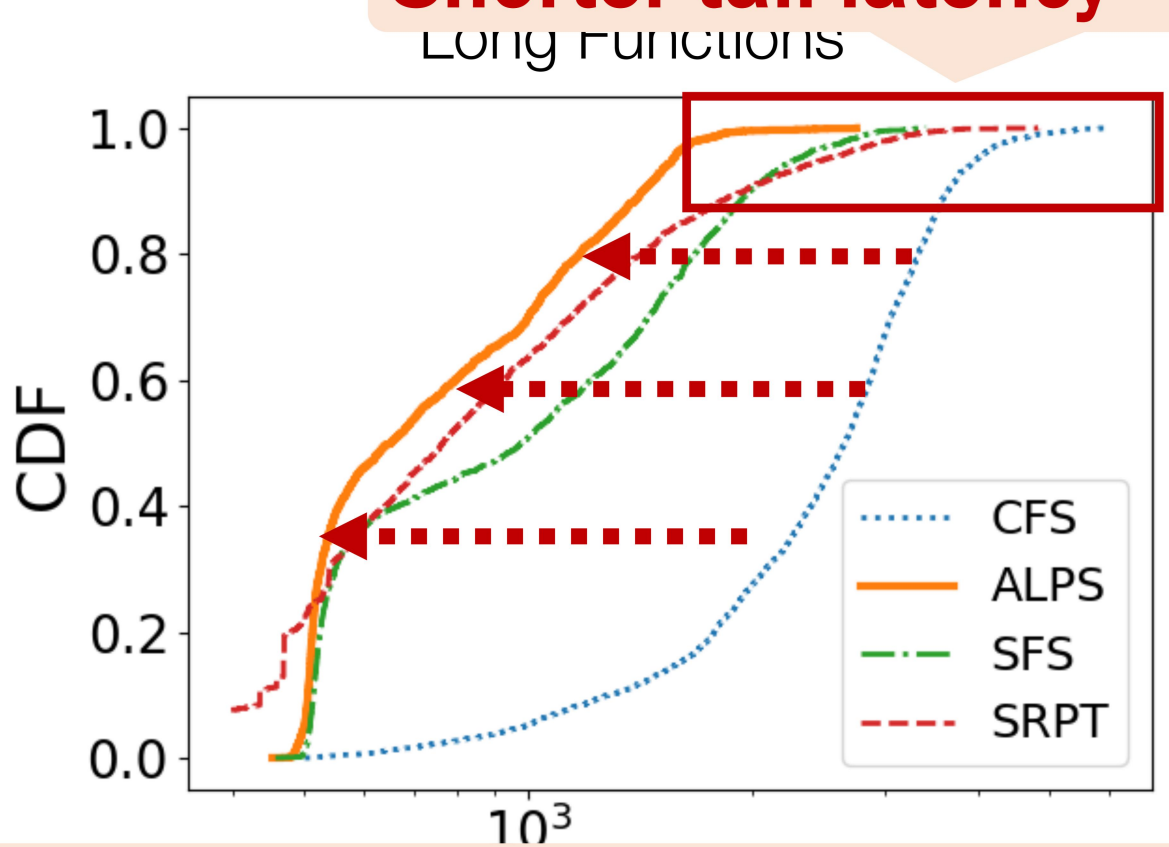
Experimental methodology

- Platform:
  - OpenLambda & Docker
  - We modified 135 LoC in OpenLambda and 223 LoC in Docker
- Host machine: bare-mental with 56 CPUs and 256 GB RAM
- OS: Ubuntu 22.04.1 LTS
- Production workload traces:
  - Huawei Cloud Functions trace [SoCC'23]
  - Azure Functions trace [ATC'20]

End-to-End Performance: Azure Workload

Shorter tail latency





ALPS achieves shorter execution durations compared to CFS

Conclusion

- ALPS continuously learns FaaS workload intelligence from user-space SRPT simulation.
- We built a prototype of ALPS into a user-space frontend and a kernel-space backend atop Linux CFS using customized eBPF functions and hooks.
- Extensive evaluation shows that ALPS improves the performance for both short functions and long functions compared to CFS.