

# ALPS:

## An **A**daptive **L**earning, **P**riority OS **S**cheduler for Serverless Functions

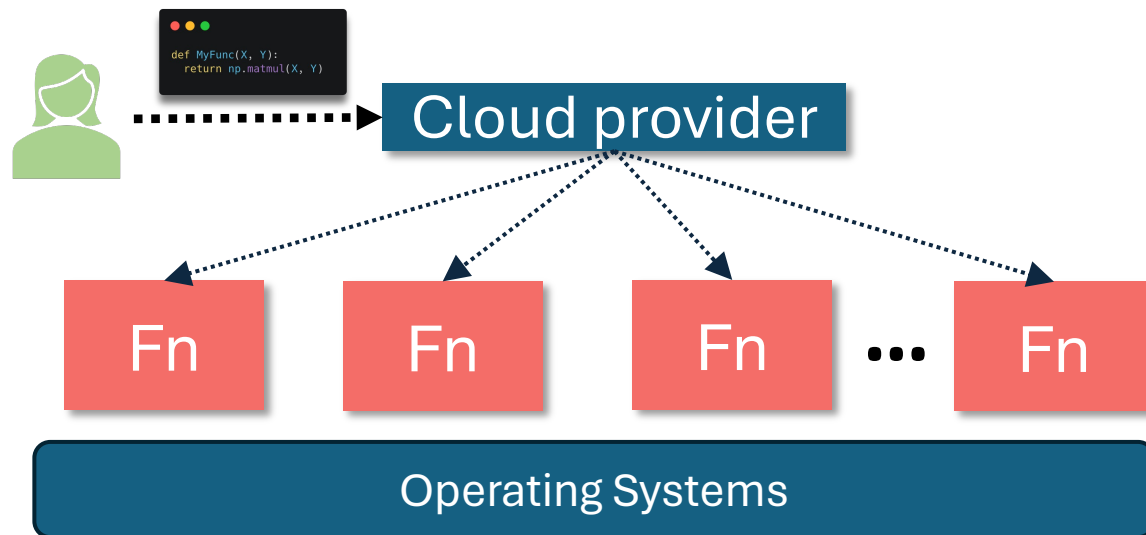
Yuqi Fu<sup>1</sup>, Ruizhe Shi<sup>2</sup>, Haoliang Wang<sup>3</sup>, Songqing Chen<sup>2</sup>, Yue Cheng<sup>1</sup>



# Serverless Computing

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

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

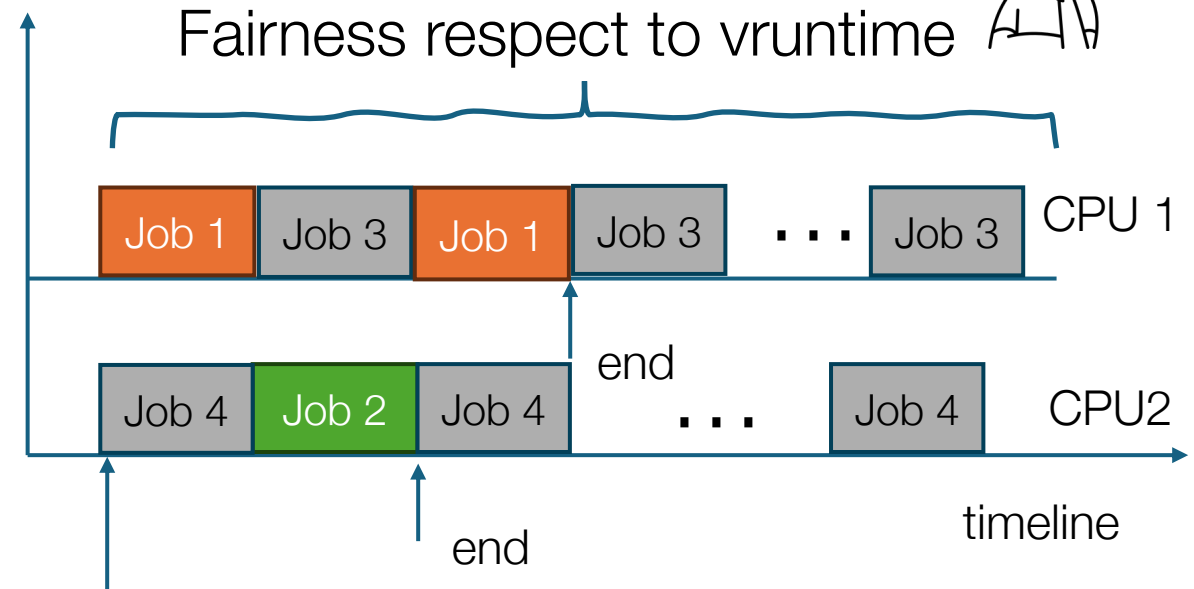
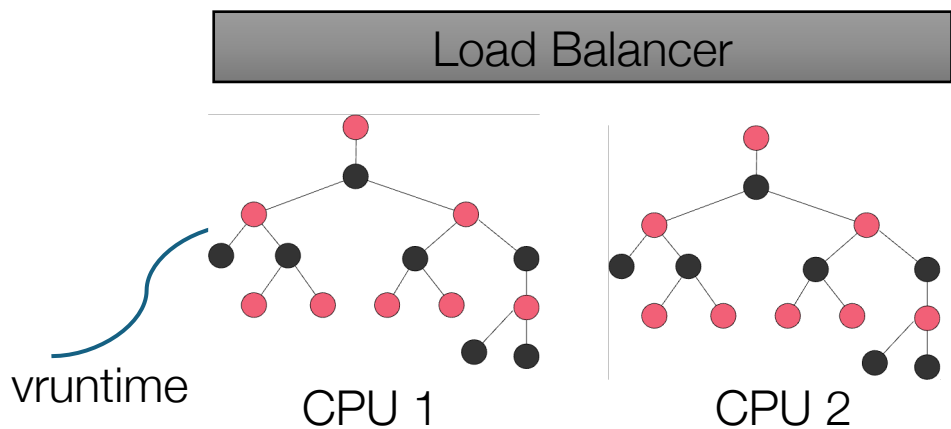


# Current Scheduler Limitations

CFS (Completely Fair Scheduler)

- Proportional-share time slice
- Default Linux scheduler

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



Context switch &  
Update vruntime &  
Find next task

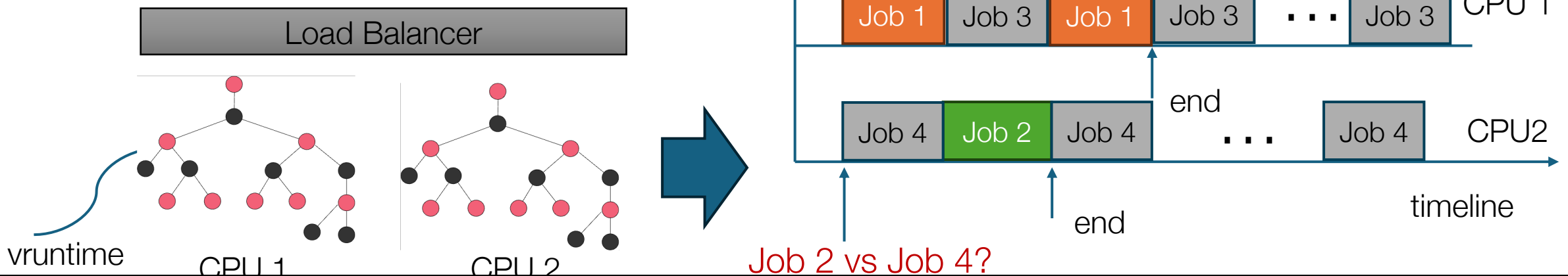
# Current Scheduler Limitations

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)

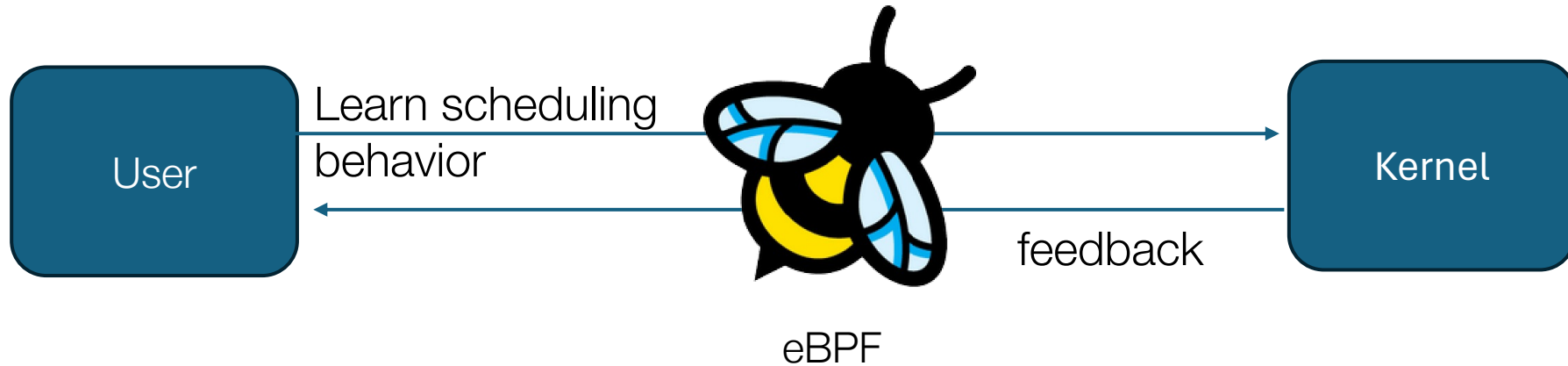
- provably **optimal** for average performance
- **impractical** in online scheduling
- cause **CPU resource starvation** for long-term jobs

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

# Outline

- Motivation
- **ALPS design and implementation**
- Evaluation

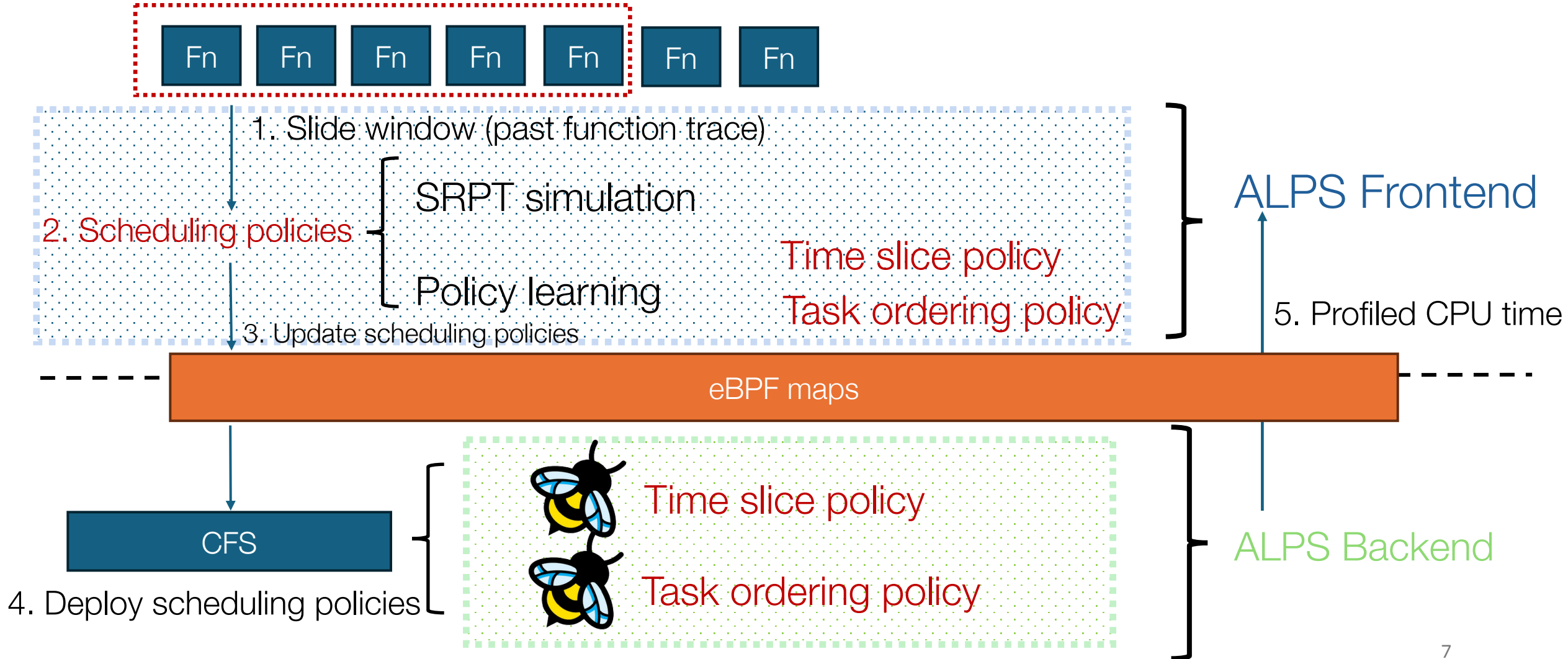
# Birds' eye view of ALPS design



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





# ALPS frontend: SRPT simulation



Workload statistics

<function\_UID, arrival time, termination time, CPU time>

CPU time profiler

alps\_execve()

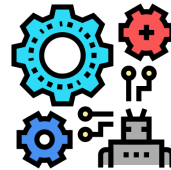
SRPT simulation

Priority queue

Short job

Long job

SRPT profiles:  
Function UID: waiting time,  
execution time, time slice



Policy learning

- \* Time slice policy
- \* Task ordering policy

# CFS workflow in the kernel

User space | Kernel space

tick interrupt/preemption/...

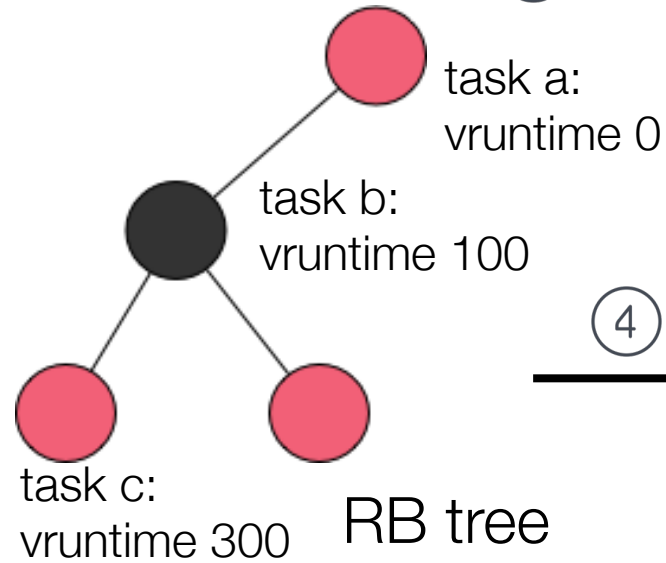
①

CFS main `schedule()`

②

Which task to run next?

③



④

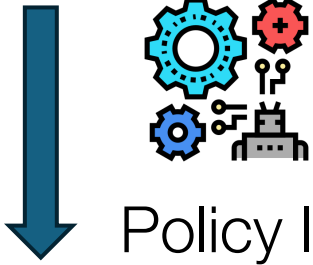
sorted by vruntime

task a < task b < task c

# ALPS' time slice policy

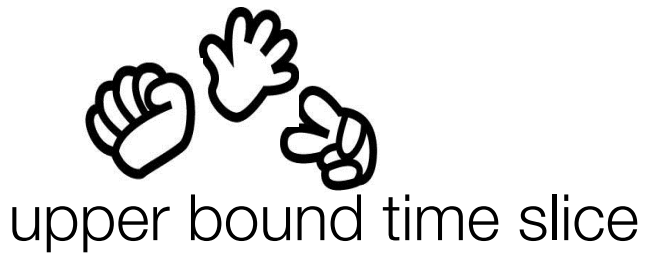
User space | Kernel space

SRPT simulation profiles



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

Current time slice

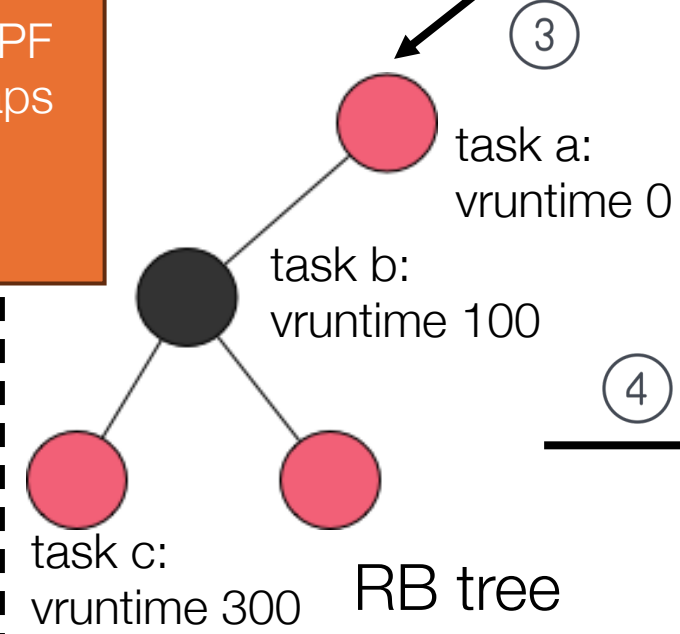


deploy

CFS main `schedule()`



Which task to run next?



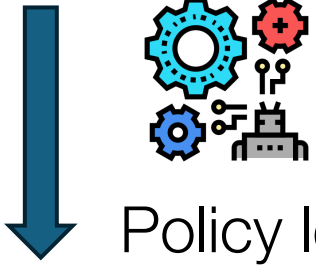
sorted by **vruntime**

task a < task b < task c

# ALPS' time slice policy

User space | Kernel space

SRPT simulation profiles



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



deploy

tick interrupt/preemption/...  
① ↓  
CFS main `schedule()`



② ↓  
**CFS branch:**  
pick the task with minimal vruntime

③ ↓  
task a:  
vruntime 0

task b:  
vruntime 100

task c:  
vruntime 300

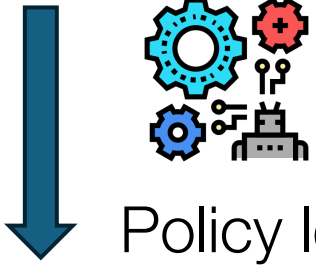
④ →  
sorted by  
**vruntime**  
task a < task b < task c

RB tree

# ALPS' time slice policy

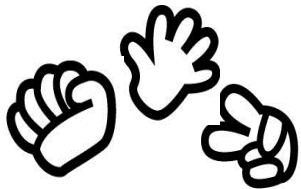
User space | Kernel space

SRPT simulation profiles



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

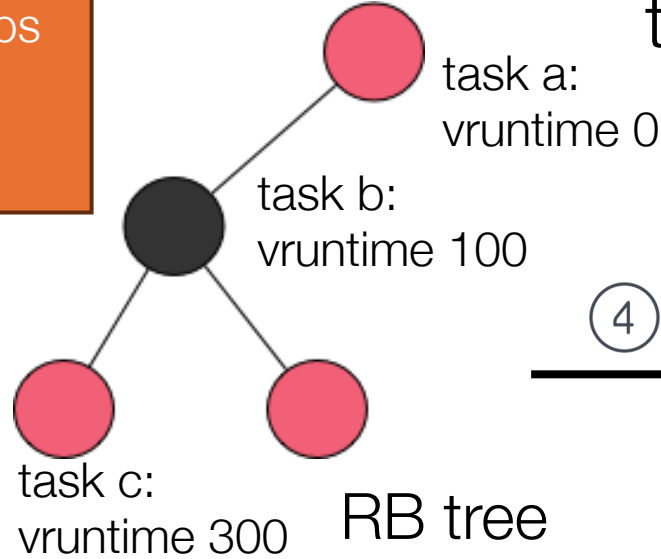


deploy

CFS main `schedule()`



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



RB tree

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

tick interrupt/preemption/...

①

CFS main `schedule()`

②

Which task to run next?

③



deploy

task a:  
vruntime 0

task b:  
vruntime 100

task c:  
vruntime 300

RB tree

④

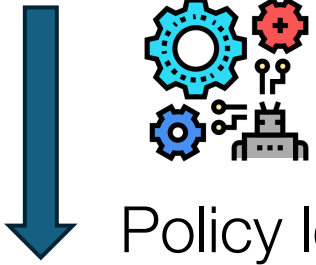
sorted by **vruntime**

task a < task b < task c

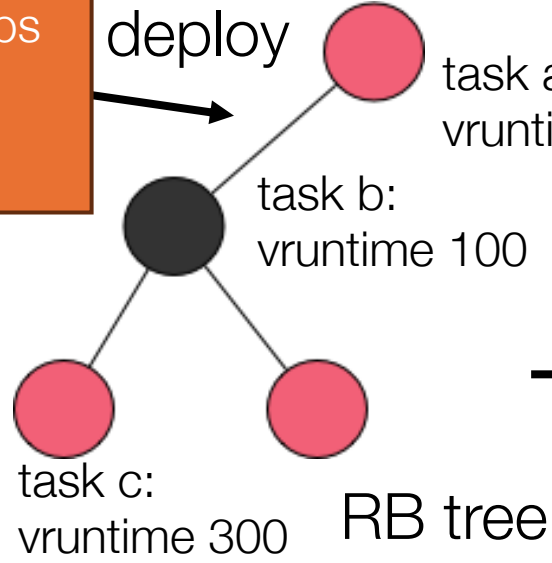
User space | Kernel space

# ALPS' task ordering policy

SRPT simulation profiles



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



tick interrupt/preemption/...

① ↓  
 CFS main **schedule()**

② ↓  
**Which task to run next?**

③ ↓  
 task a: vruntime 0

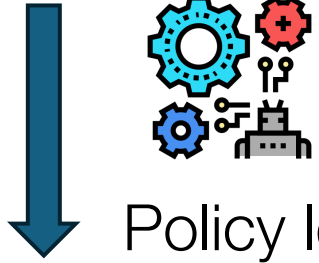
④ → sorted by **ALPS priority**

**task b < task a < task c**

User space | Kernel space

# ALPS' task ordering policy

SRPT simulation profiles



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

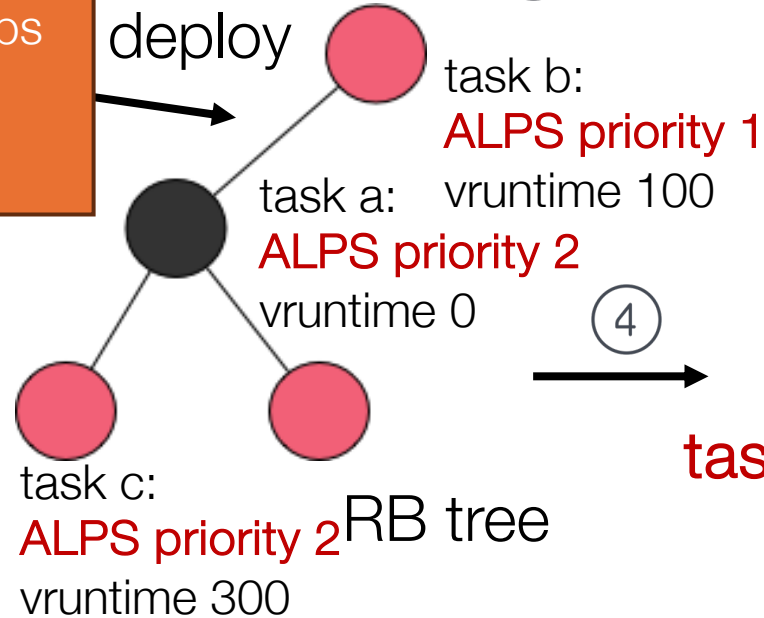
tick interrupt/preemption/...

① ↓  
 CFS main `schedule()`

② ↓  
 Which task to run next?



deploy



③

④

sorted by **ALPS priority**

task b < task a < task c



# Outline

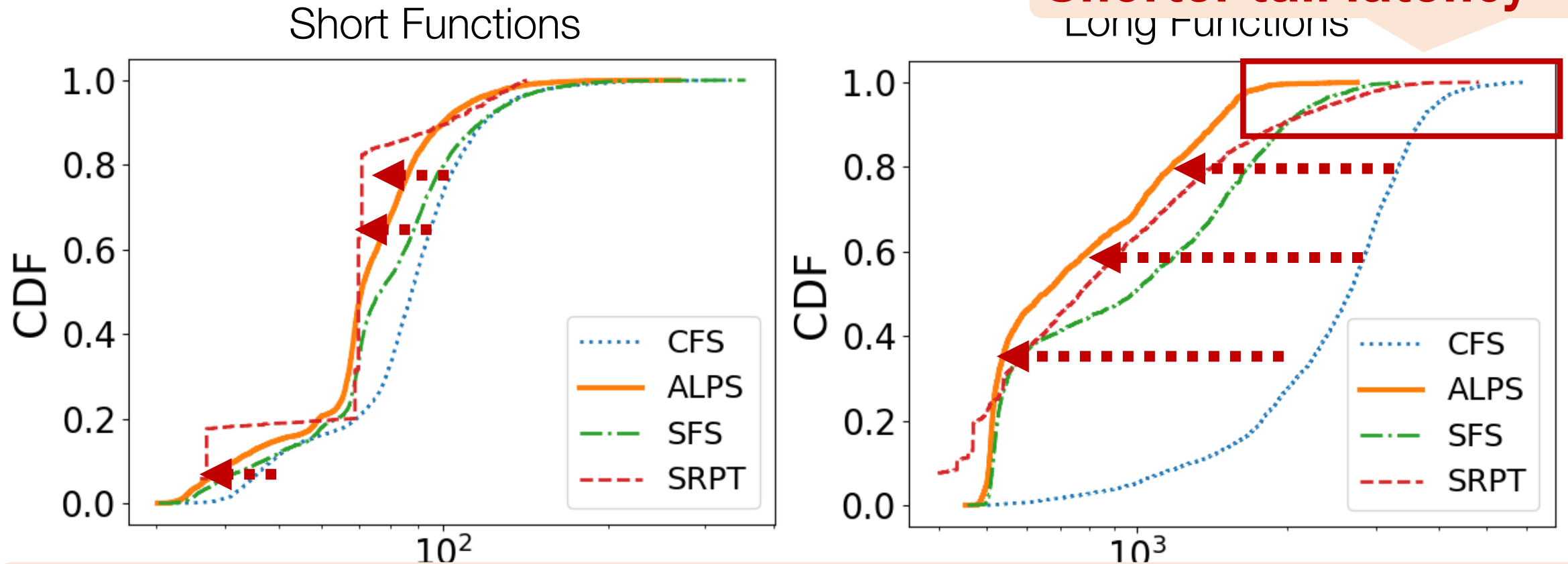
- Motivation
- ALPS design and implementation
- Evaluation

# Experimental methodology

- Platform:
  - OpenLambda & Docker
  - We modified **135** LoC in OpenLambda and **223** LoC in Docker
- Host machine: bare-metal 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

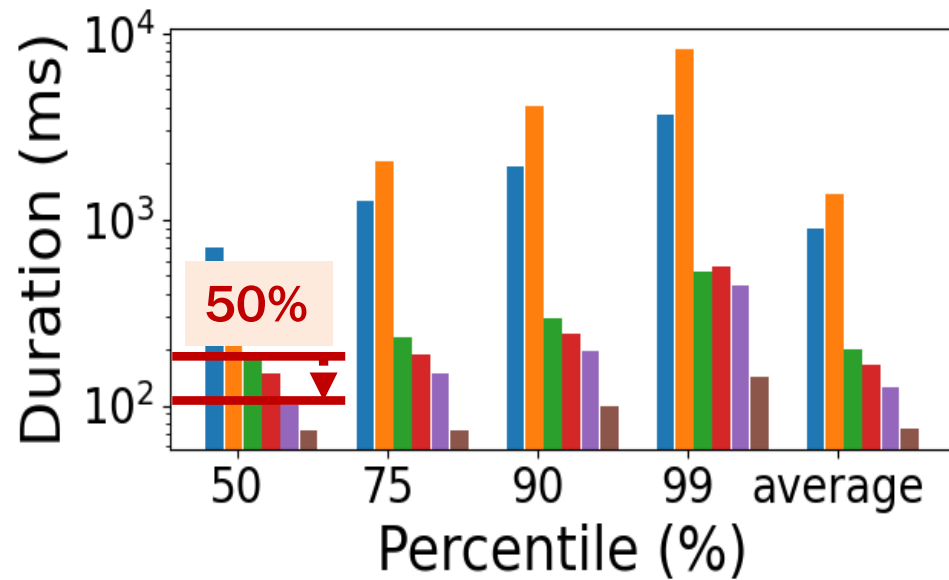
**Shorter tail latency**



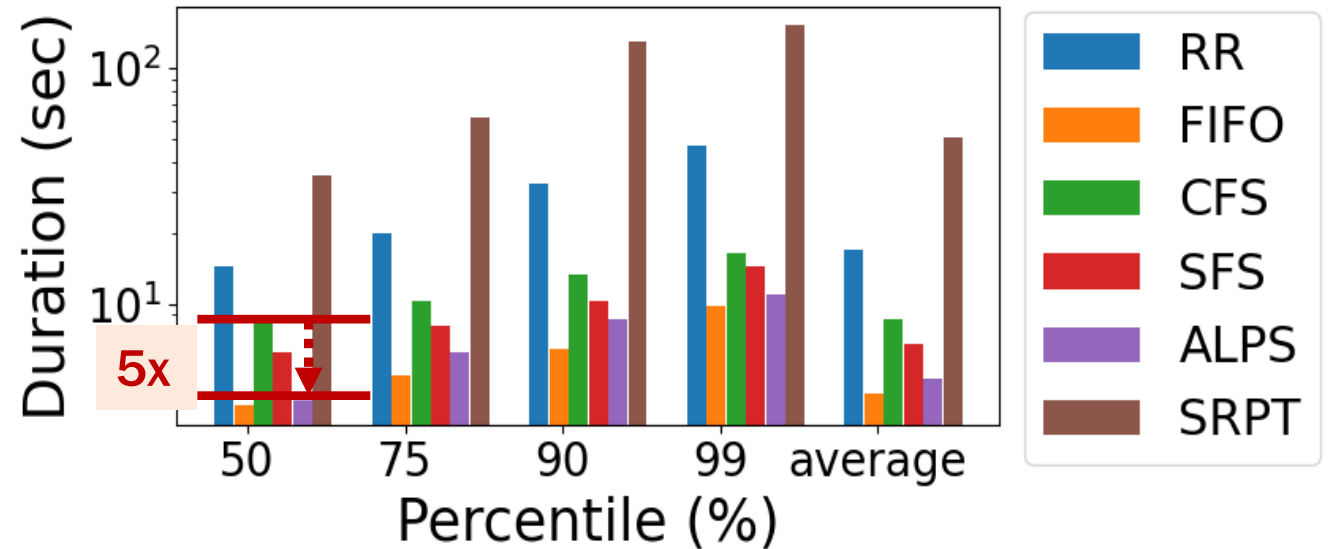
**ALPS achieves shorter execution durations compared to CFS**

# End-to-End Performance: Huawei Workload

## Short Functions



## Long Functions

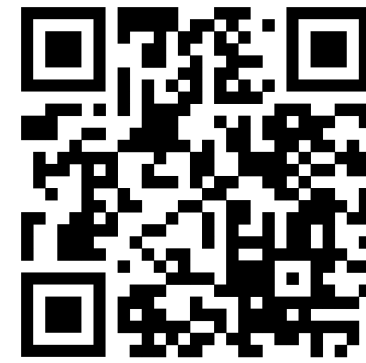


**ALPS outperforms CFS across all function execution duration percentiles**

# 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.

# Thank You! Questions?



Yuqi Fu, Ruizhe Shi, Haoliang Wang, Songqing Chen, Yue Cheng

ALPS source code: <https://github.com/ds2-lab/ALPS>



National  
Science  
Foundation

