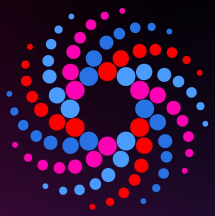


SC22
Dallas, TX | hpc accelerates.

SFS: Smart OS Scheduling for Serverless Functions

Yuqi Fu[1], Li Liu[2], Haoliang Wang[3], Yue Cheng[1], Songqing Chen[2]
[1]University of Virginia, [2]George Mason University, [3]Adobe Research

Serverless computing

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

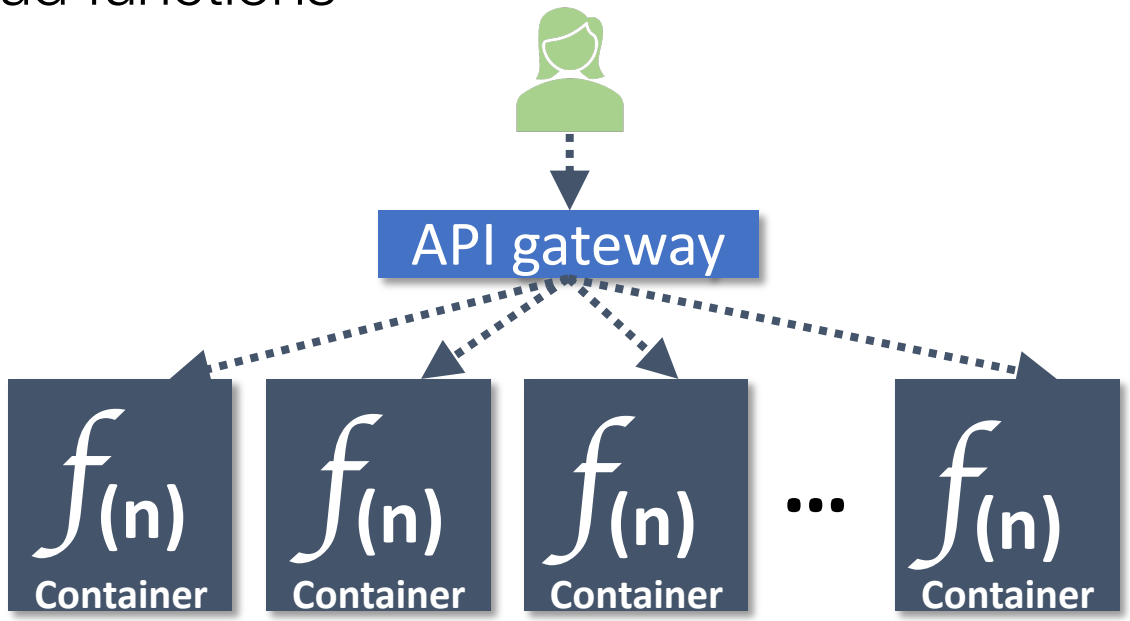
Function-as-a-Service (FaaS): Cloud functions as a basic deployment unit

AWS Lambda

Azure Functions

Google Cloud Functions

Alibaba Function Compute

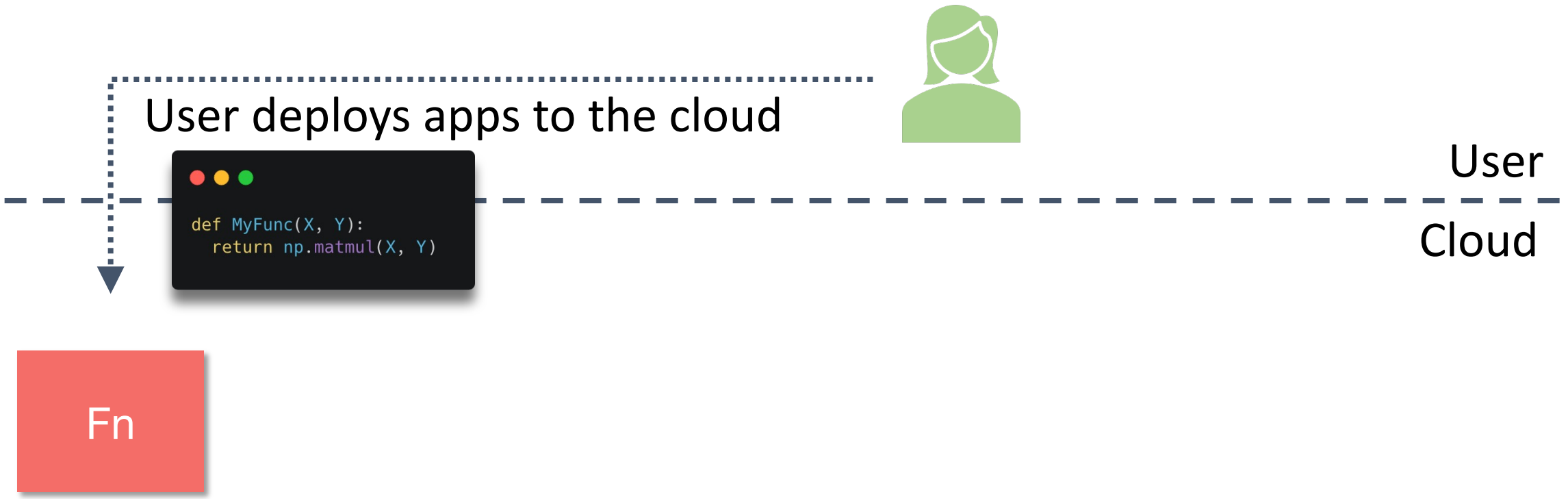


Function-as-a-Service (FaaS)

User

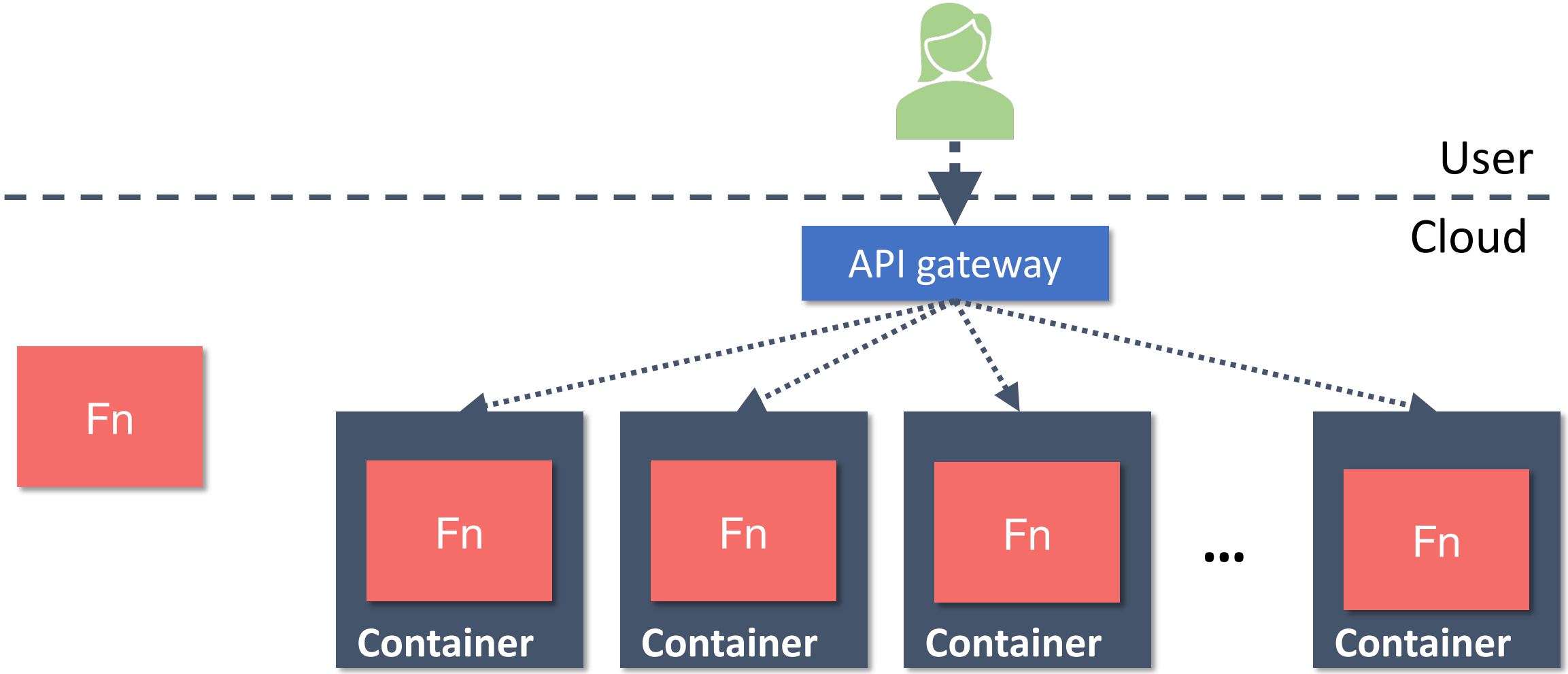
Cloud

Function-as-a-Service (FaaS)

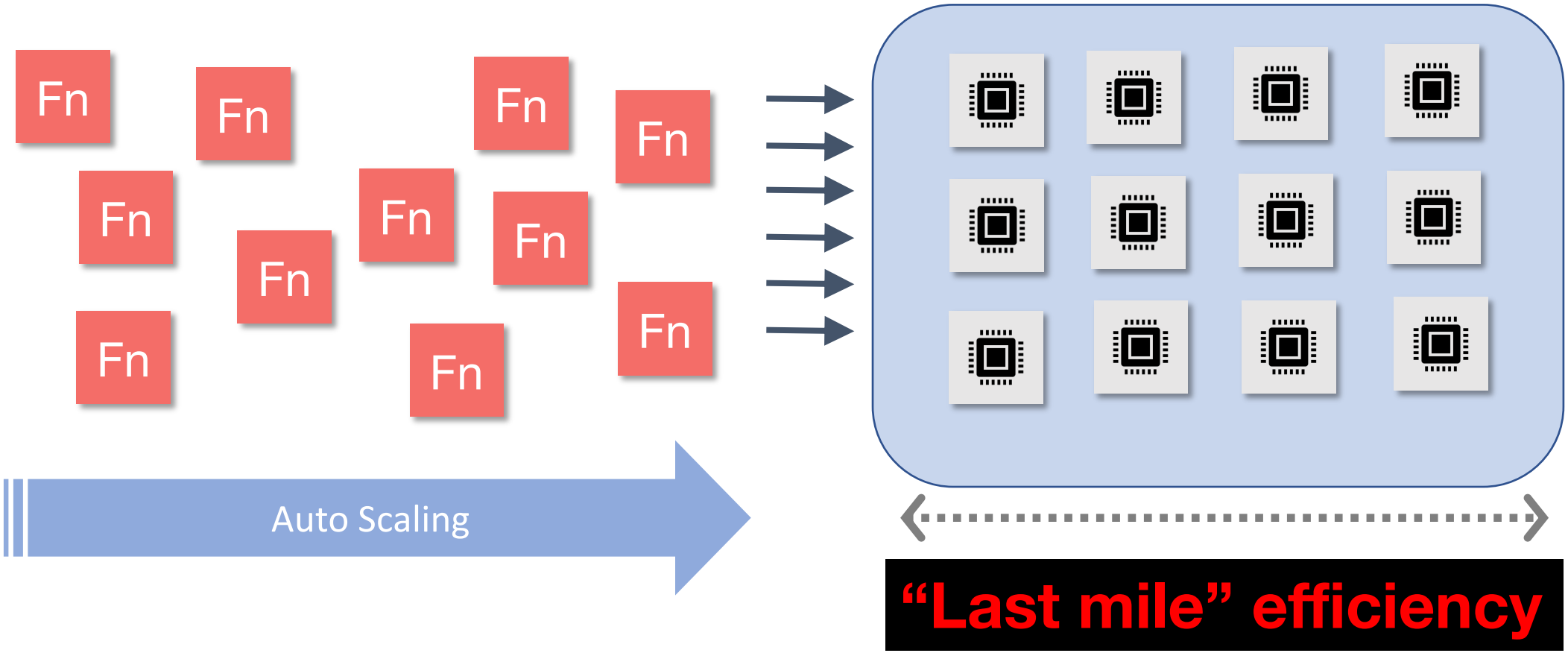


Cloud function

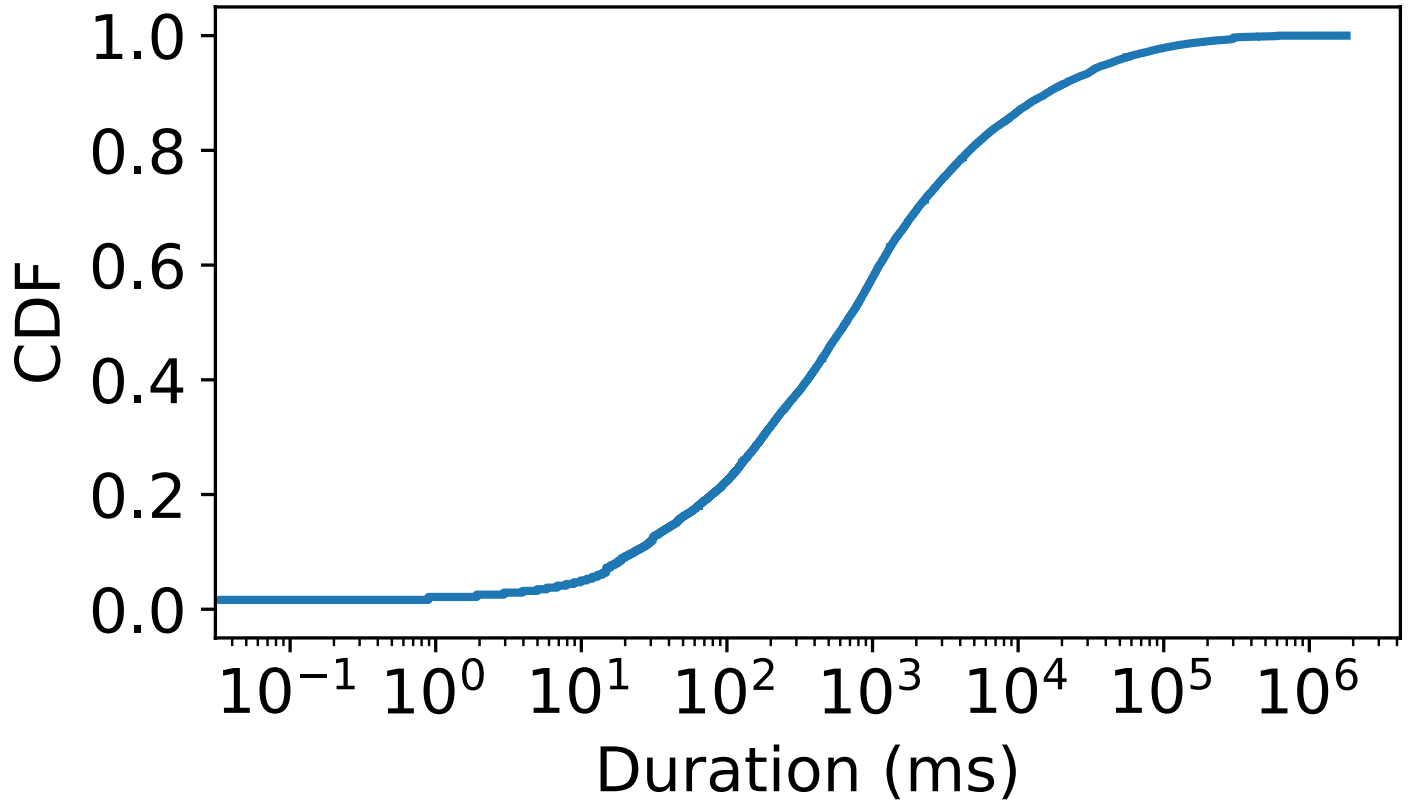
Function-as-a-Service (FaaS)



Serverless functions eventually run in OSes

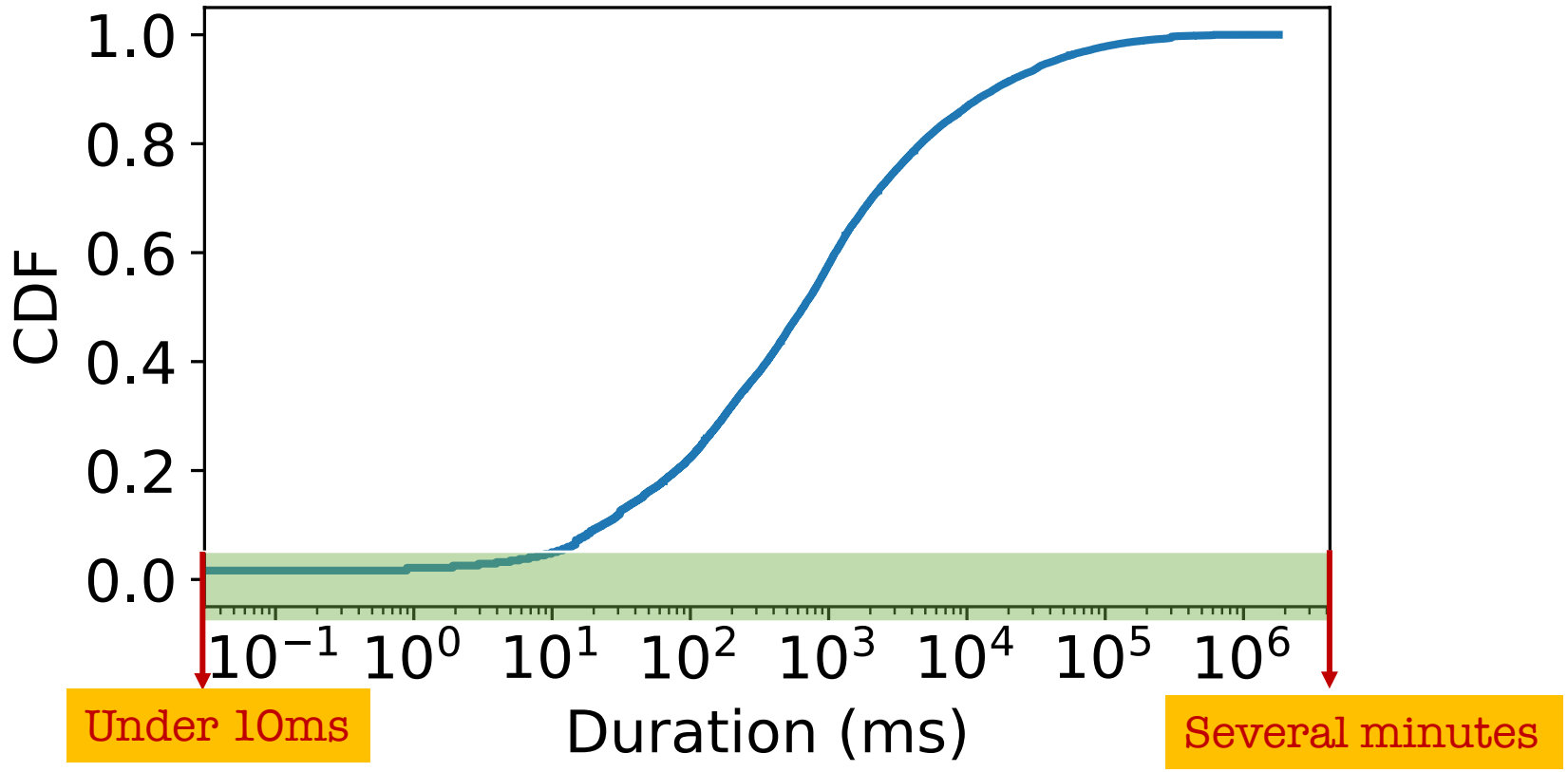


Production FaaS workloads are highly heterogeneous



A 14-day production FaaS workloads from Azure Function (ATC'20)

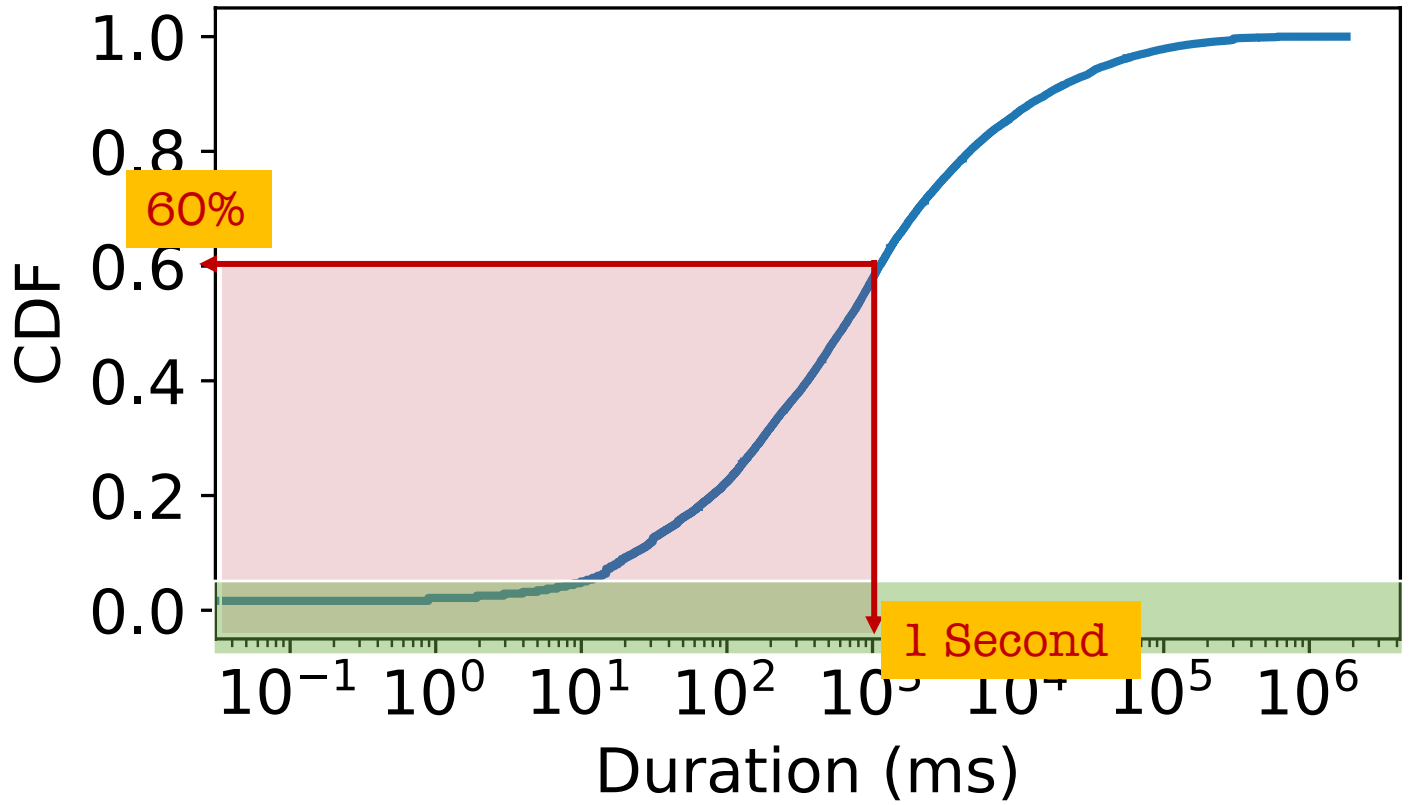
Production FaaS workloads are highly heterogeneous



A 14-day production FaaS workloads from Azure Function (ATC'20)

- Features a mixture of short and long functions

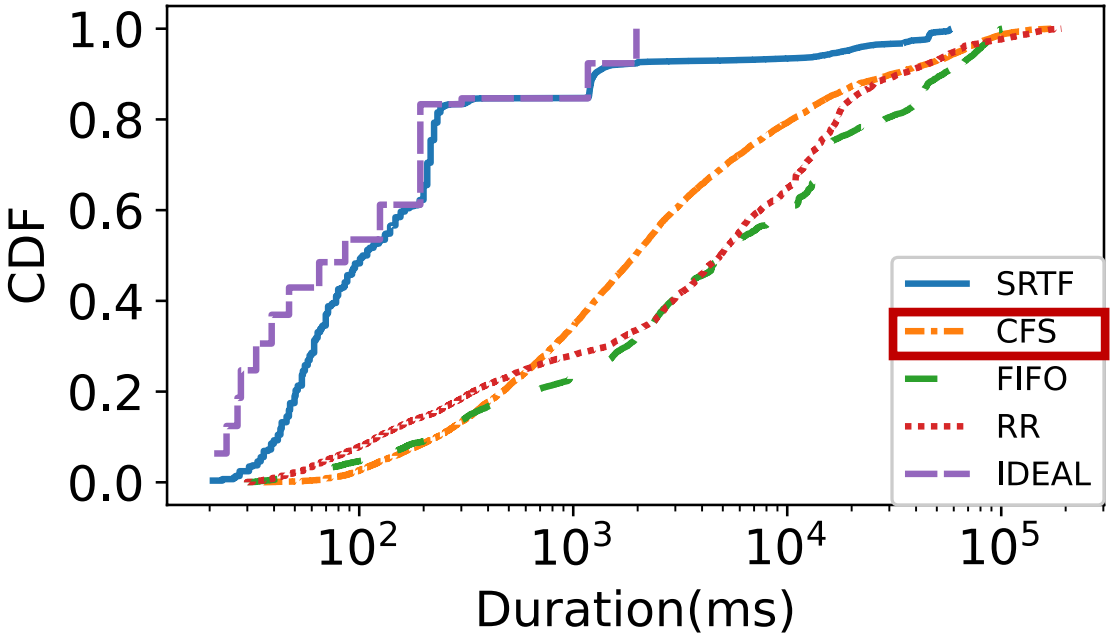
Production FaaS workloads are highly heterogeneous



A 14-day production FaaS workloads from Azure Function (ATC'20)

- Features a mixture of short and long functions
- A majority (60%) of functions finish in one second

Poor performance under existing OS scheduling

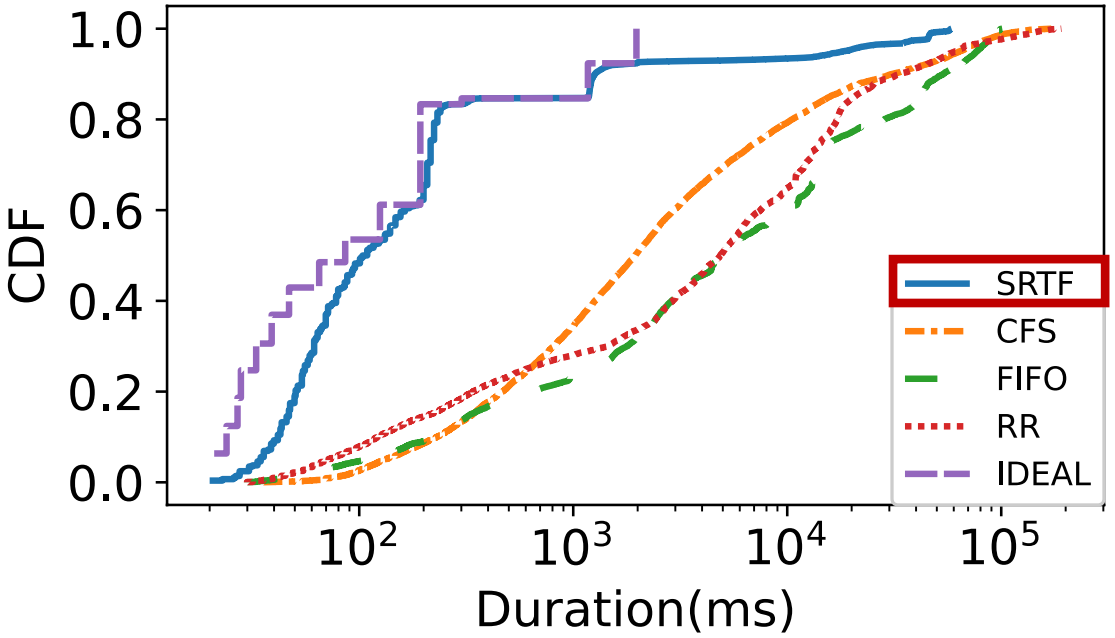


CFS

*** Proportional-Share time slice**

*** Default Linux Scheduler**

Poor performance under existing OS scheduling

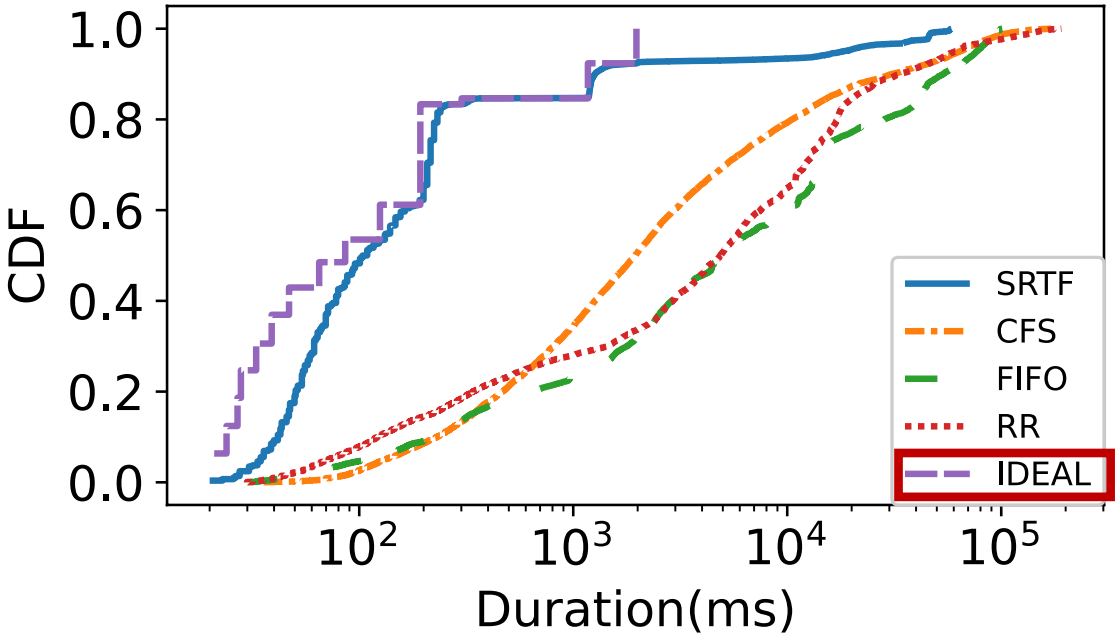


SRTF

*** Optimal**

*** Not practical**

Poor performance under existing OS scheduling

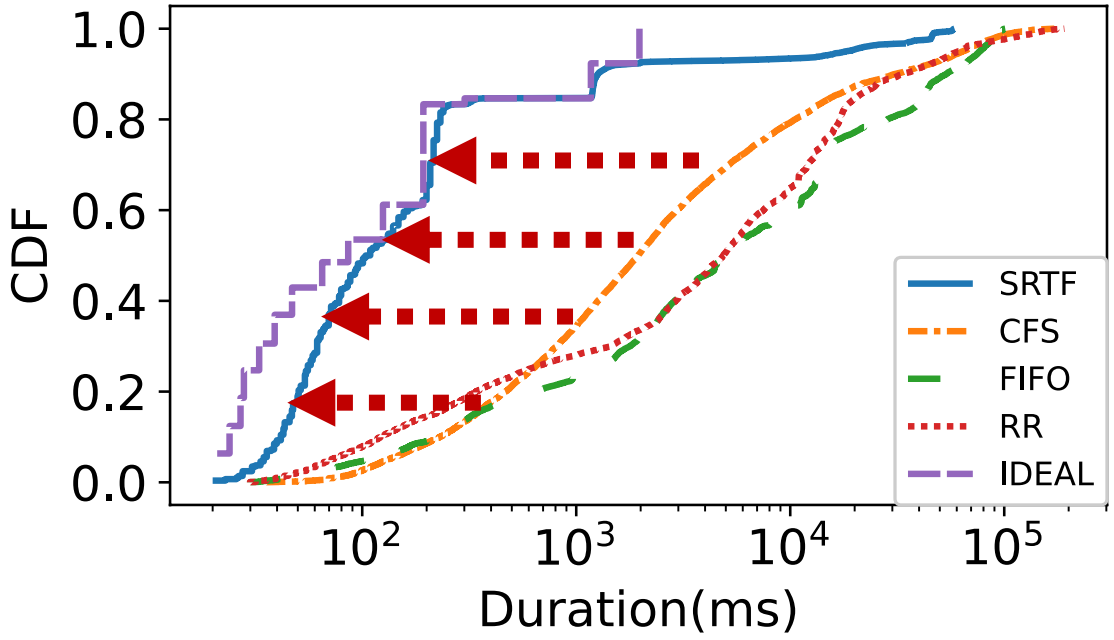


IDEAL

*** Infinite resource**

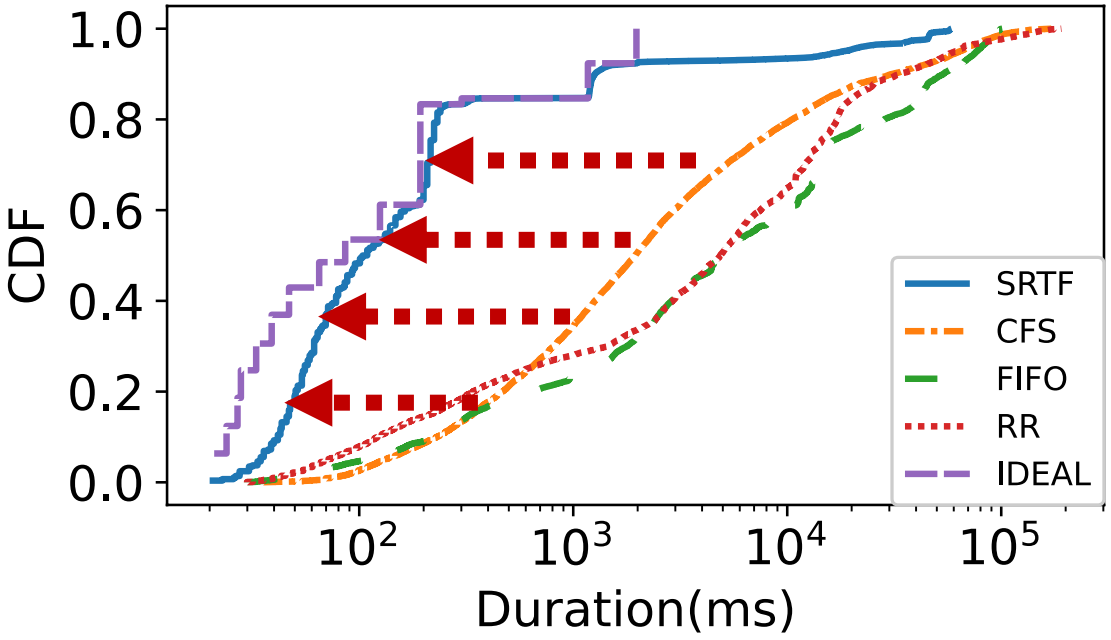
*** No preemption**

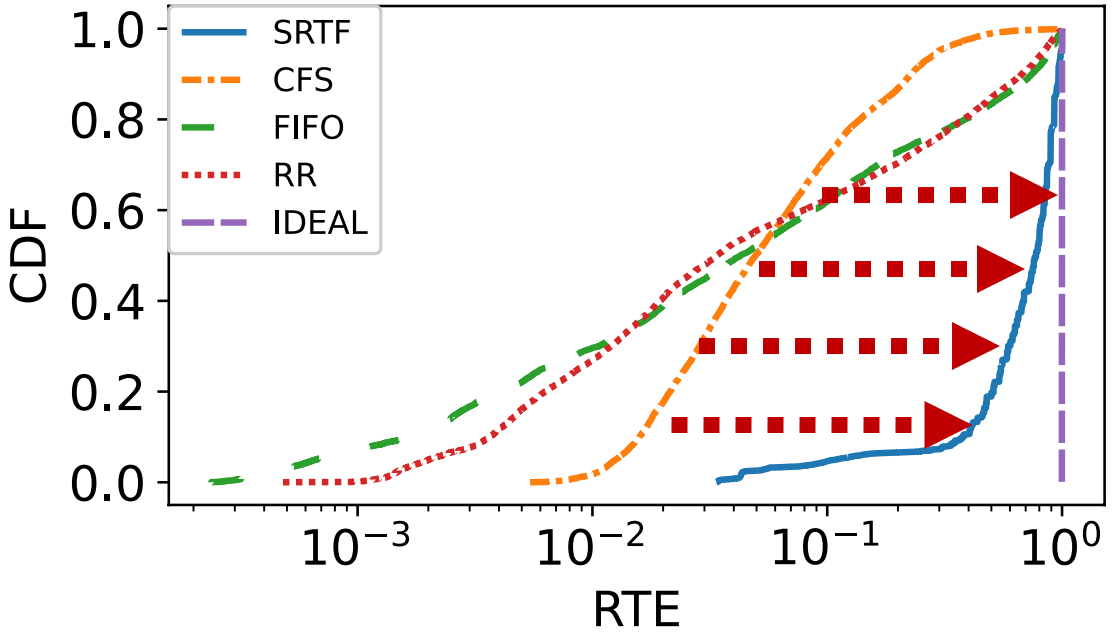
Poor performance under existing OS scheduling



Proportional-Share Scheduling offer poor performance

Poor performance under existing OS scheduling





Proportional-Share Scheduling offer poor performance

Function **Run-Time Effectiveness** = Service time / Turnaround time

CFS frequently preempts functions, causing longer waiting time (w/ smaller RTEs)

Poor performance under existing OS scheduling

Existing Linux scheduling policies are a poor match for emerging FaaS workloads

- **Implication #1:** OS-level function scheduling must be workload-aware
- **Implication #2:** Approximating SRTF (shortest remaining time first) will provide a significant performance boost for short functions

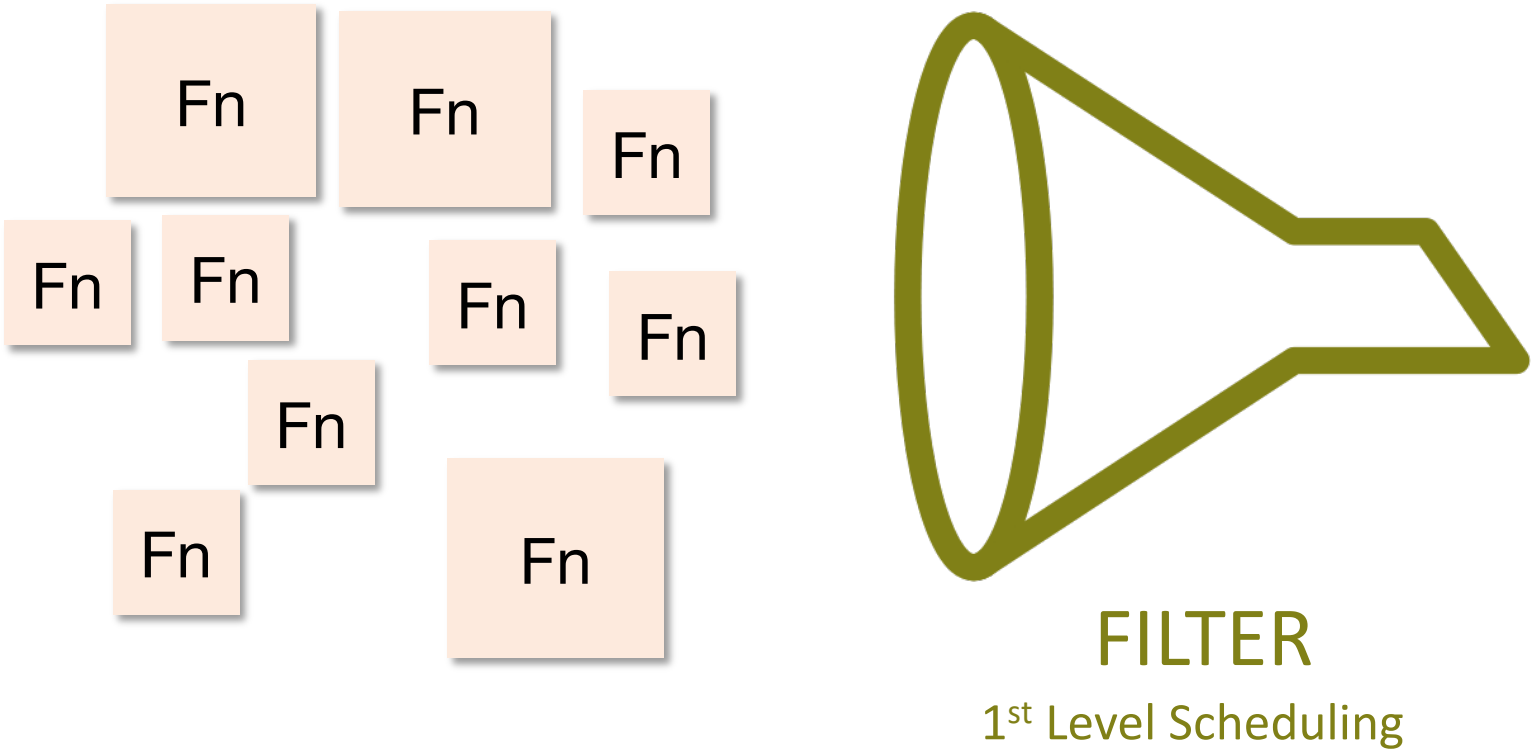
Outline

- Design
- Evaluation
- Conclusion

Smart function scheduler (SFS)

SFS is a **FaaS-aware**, user-space OS scheduler

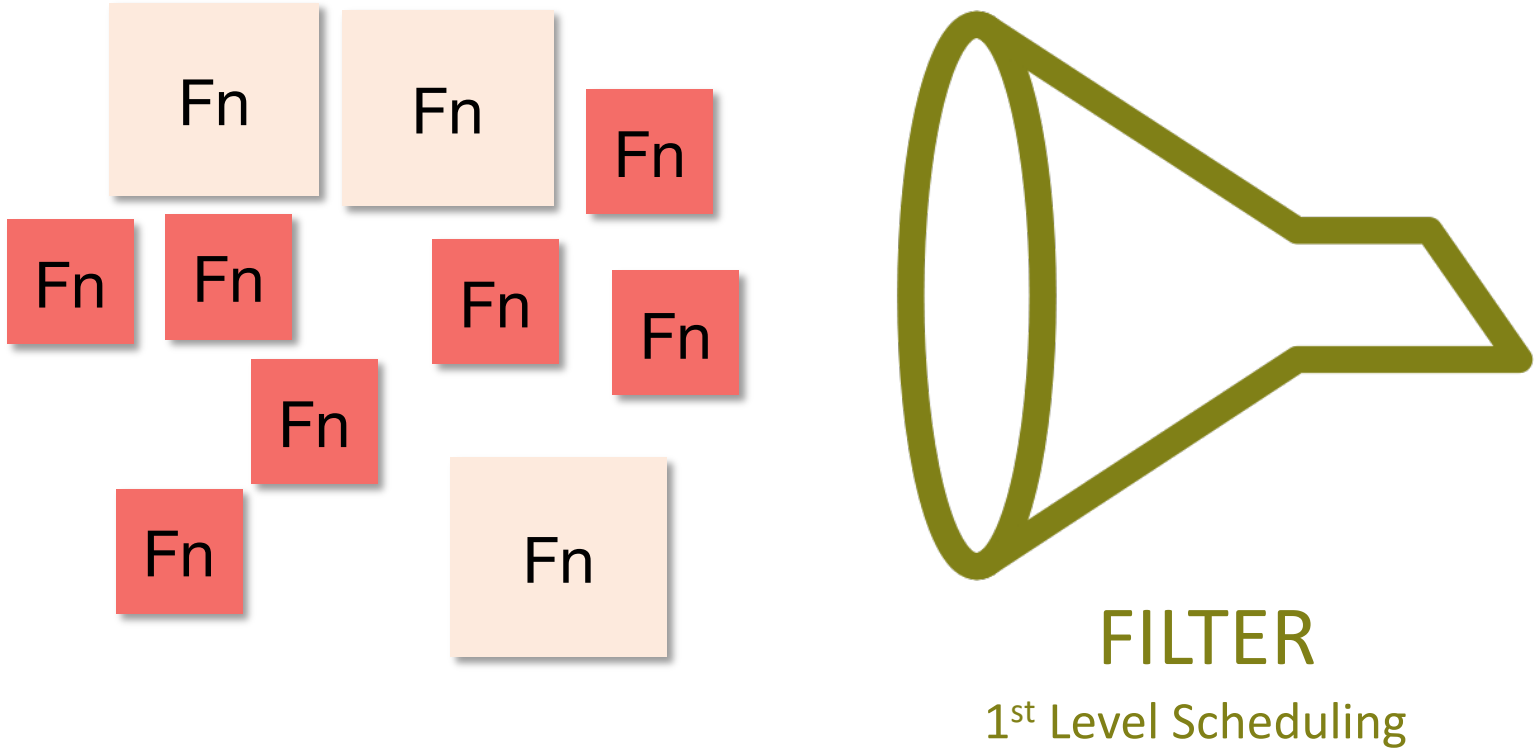
Key idea: Two-level scheduling



Smart function scheduler (SFS)

SFS is a **FaaS-aware**, user-space OS scheduler

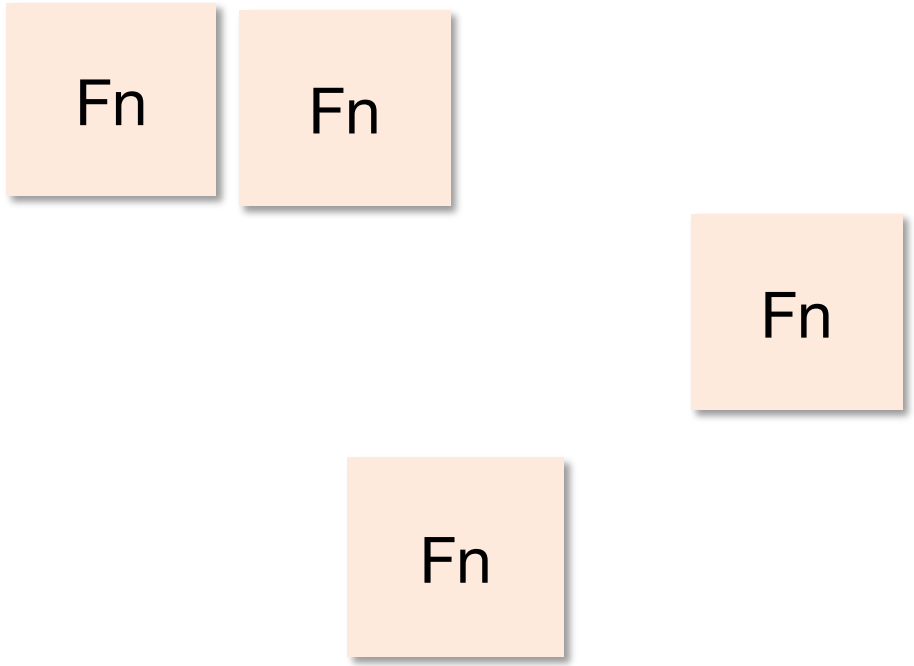
Key idea: Two-level scheduling

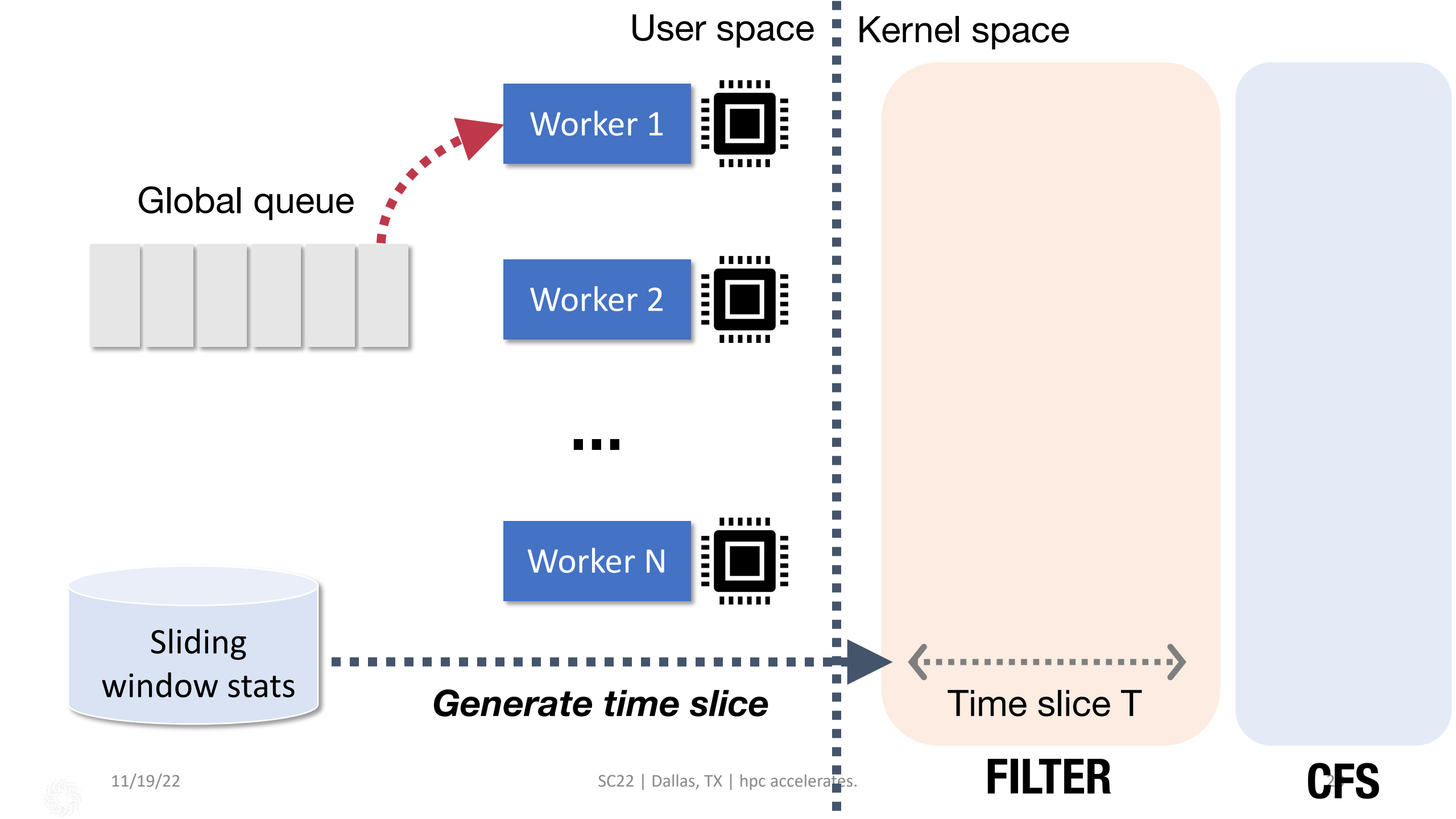


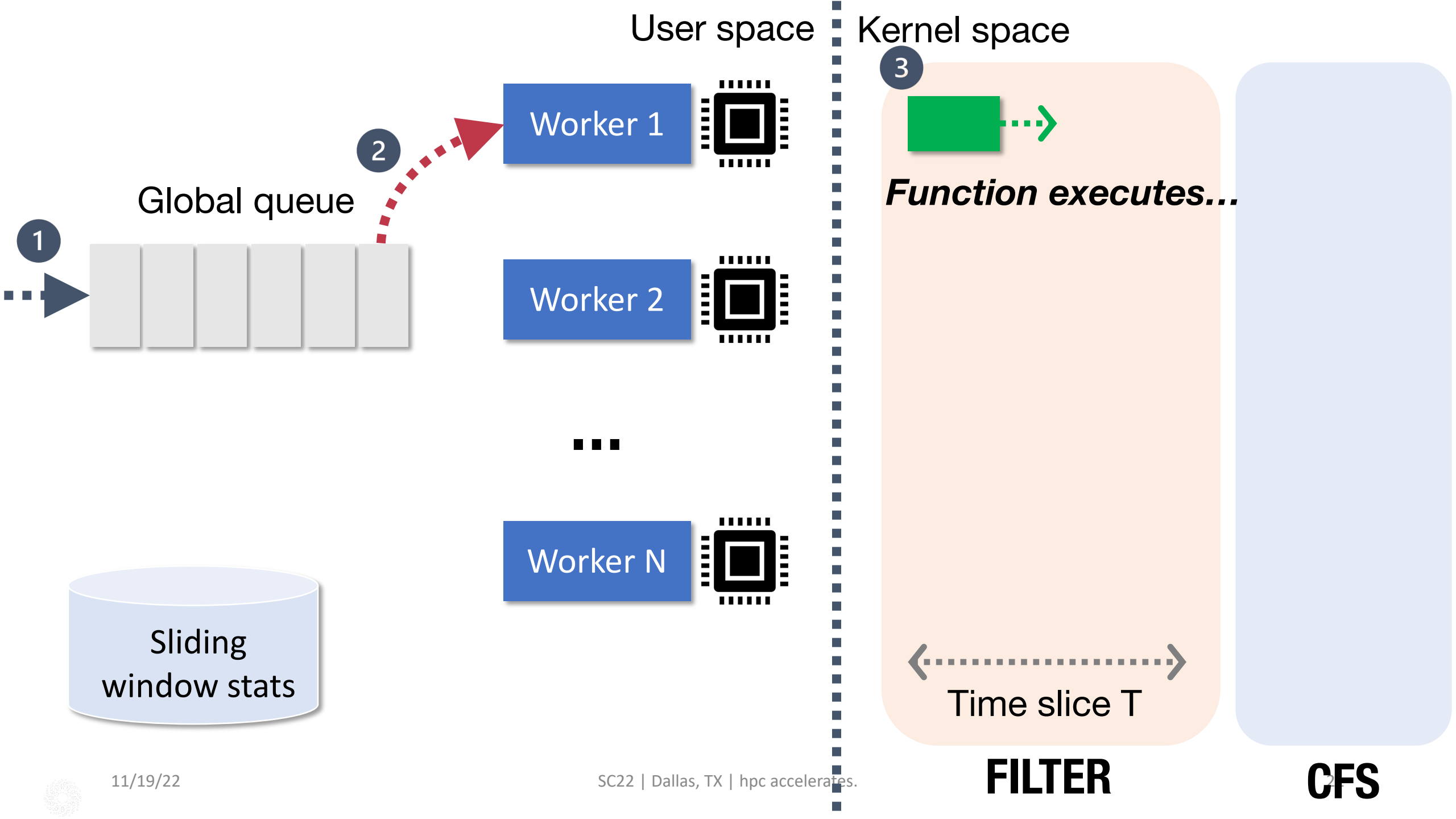
Smart function scheduler (SFS)

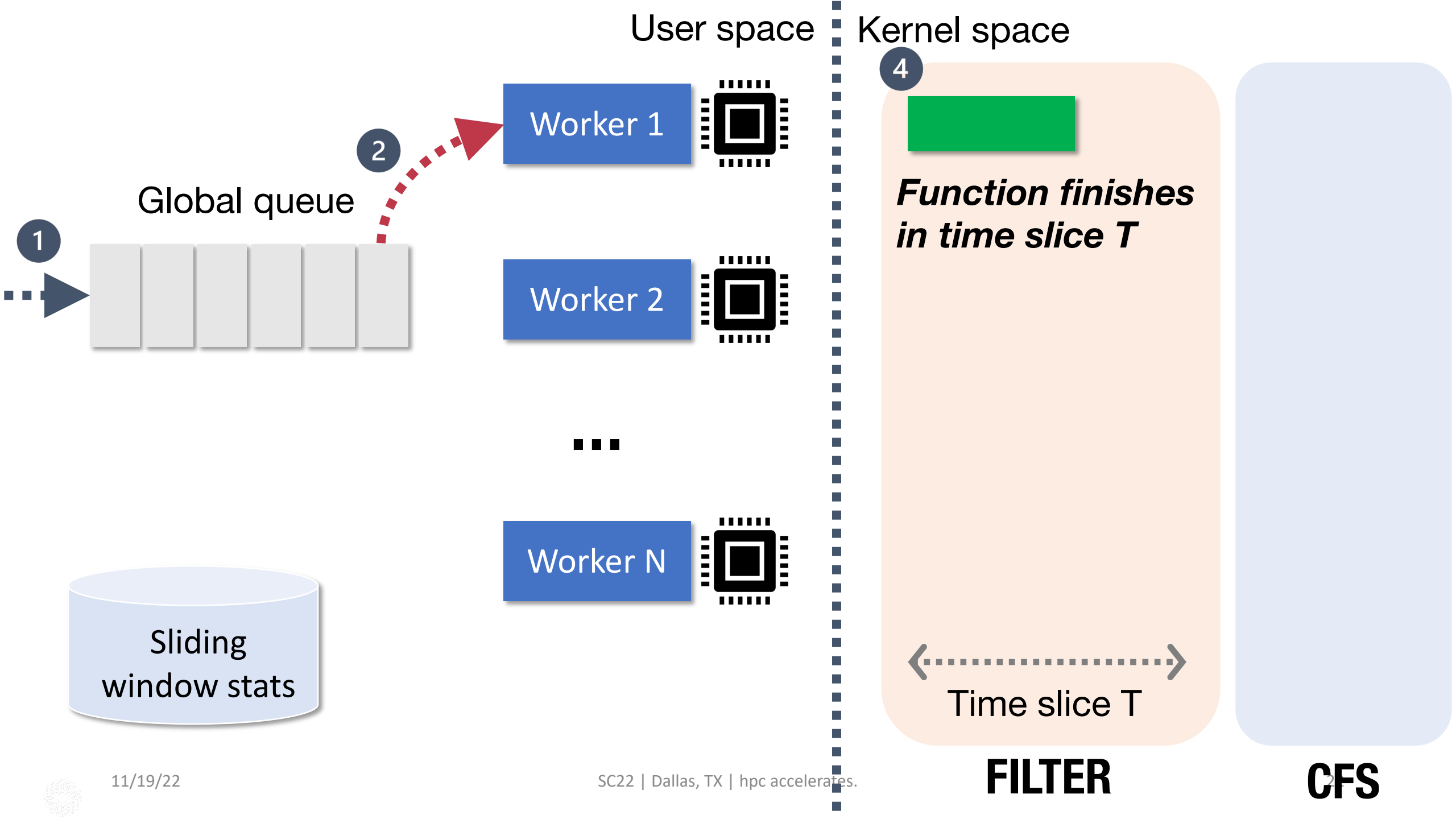
SFS is a **FaaS-aware**, user-space OS scheduler

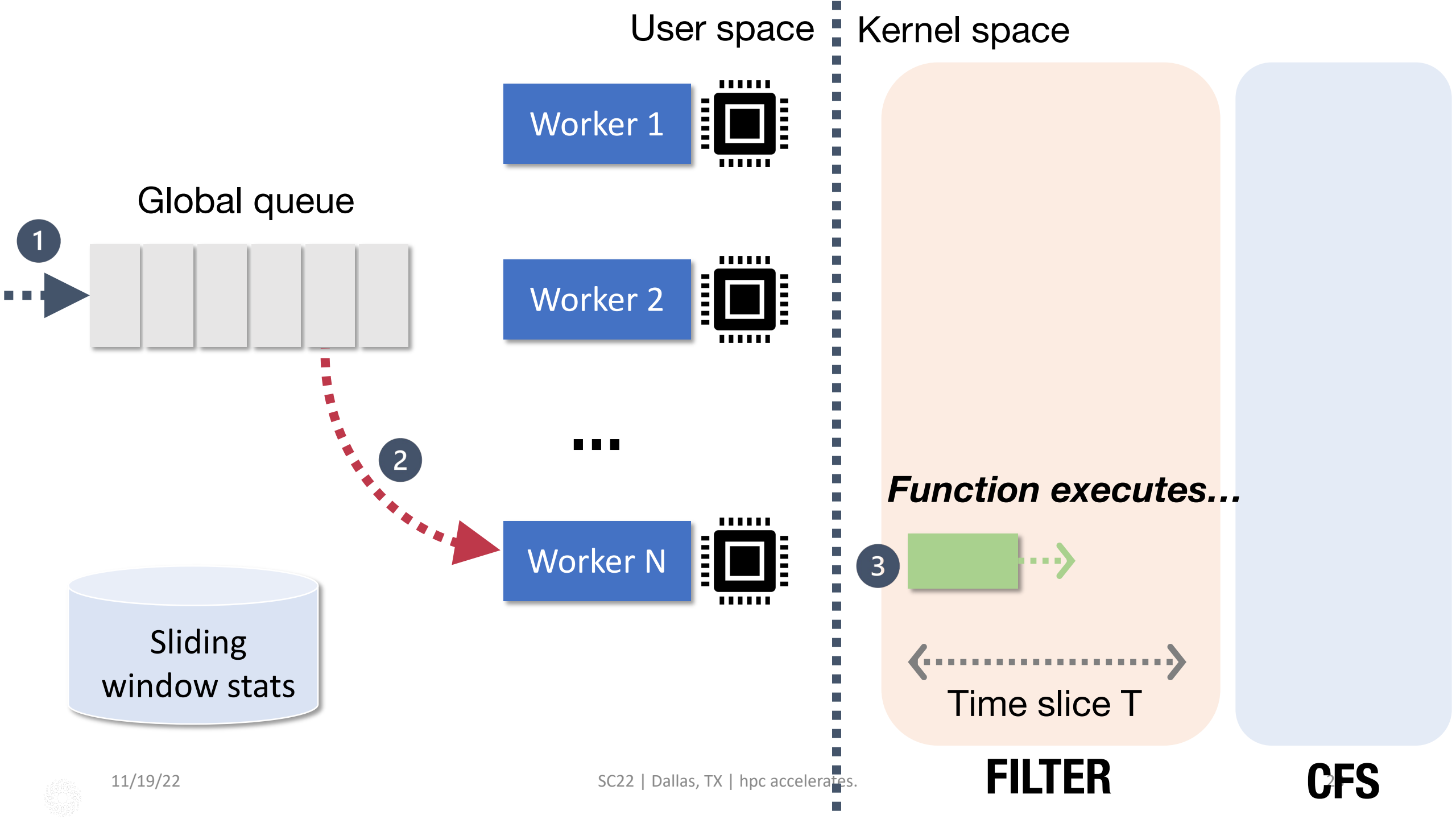
Key idea: Two-level scheduling

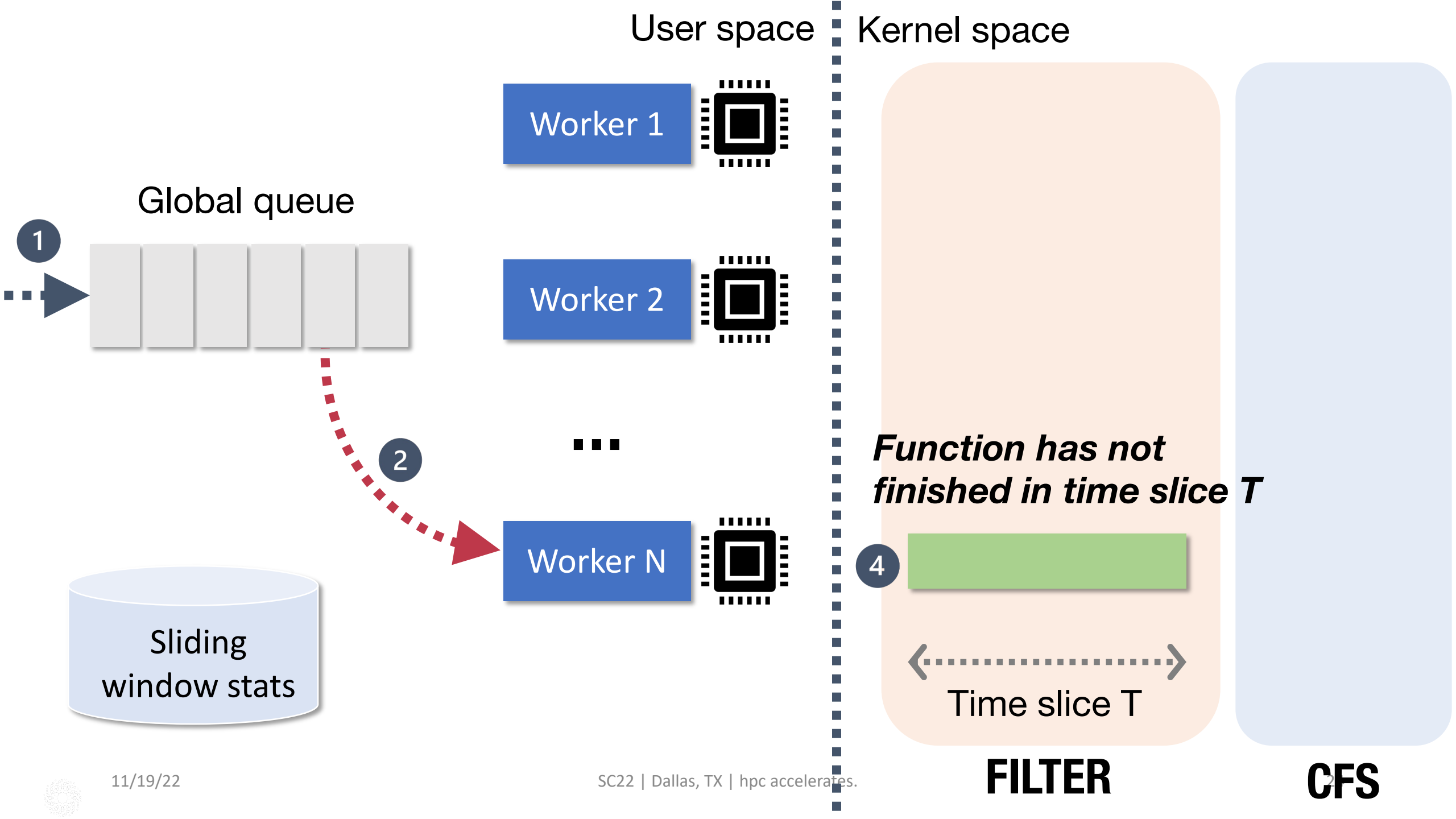


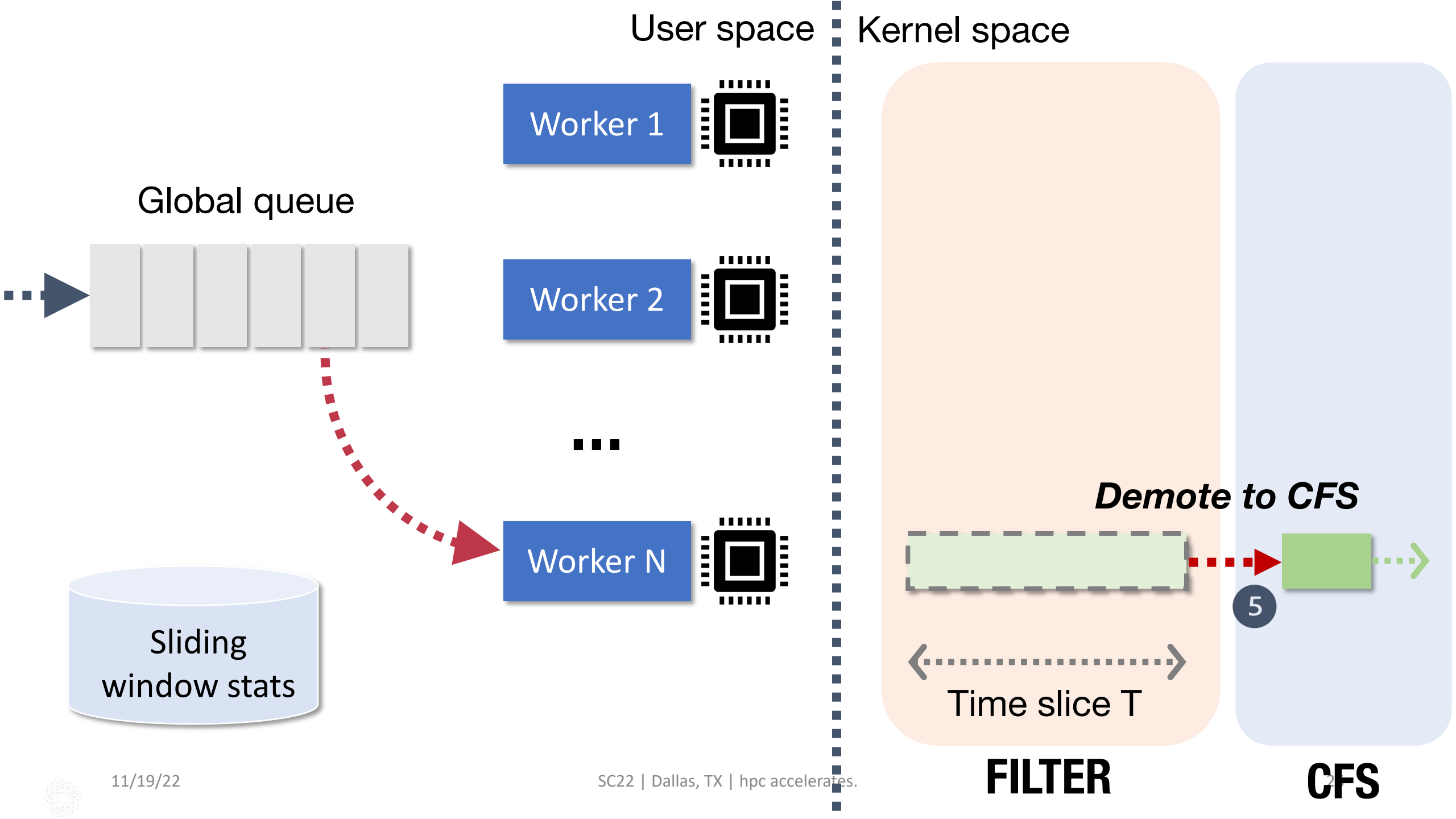

User space
Kernel space
Worker 1
Global queue
Worker 2
...
Worker N
Sliding window stats
Generate time slice
Time slice T
FILTER
CFS

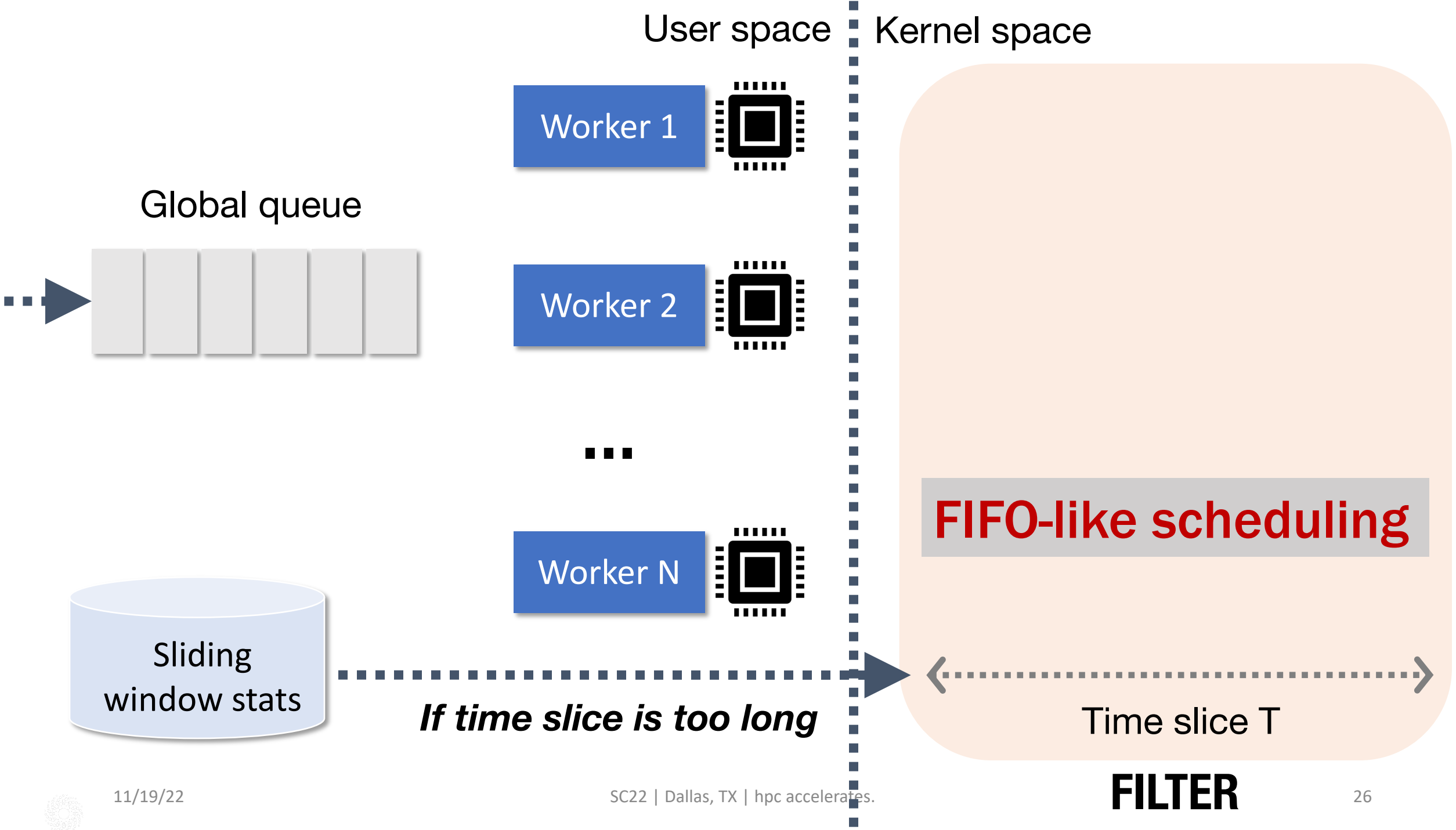

User space
Kernel space
3
Worker 1
2
Global queue
1
Worker 2
...
Worker N
Sliding window stats
Function executes...
Time slice T
FILTER
CFS

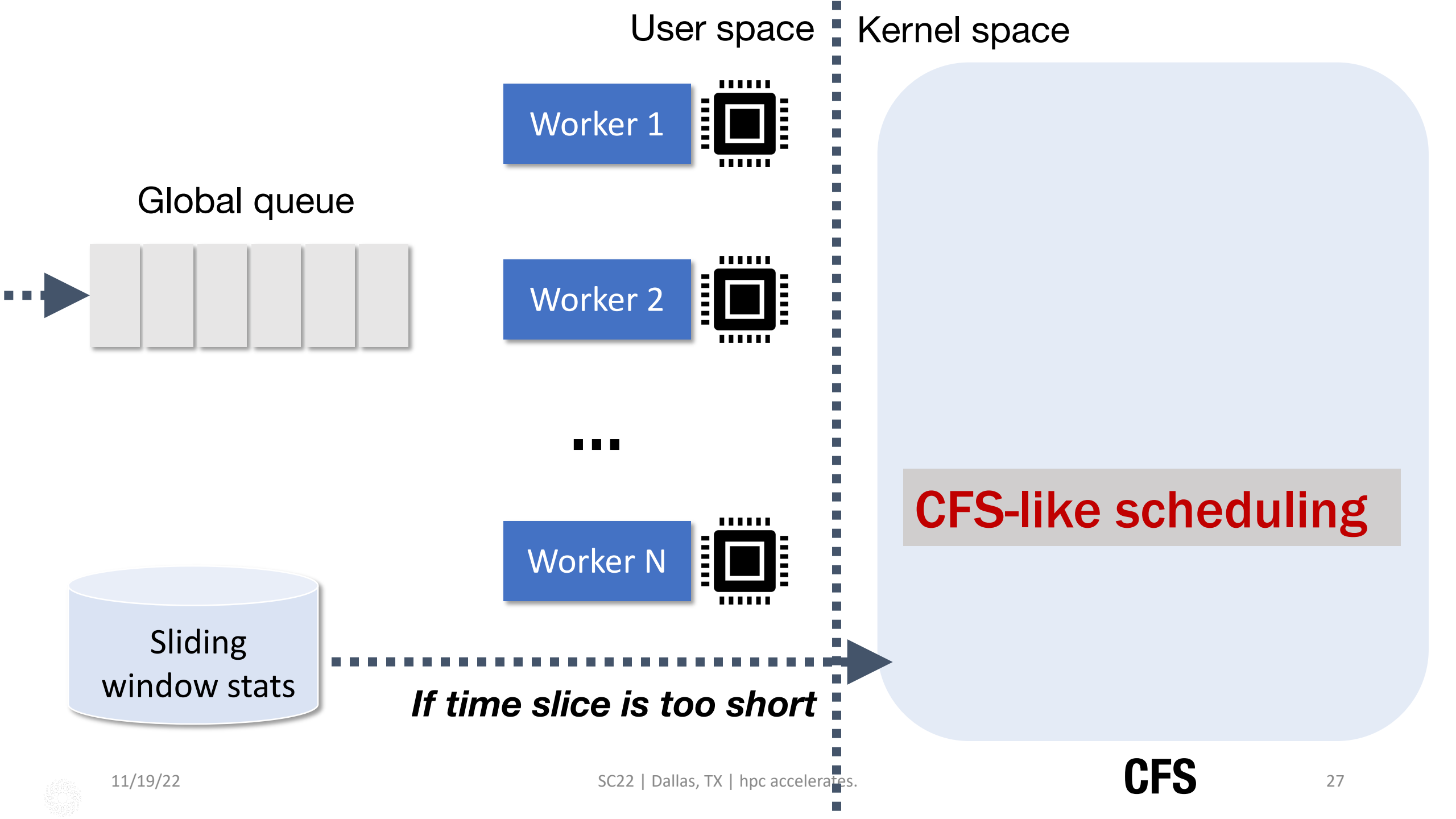

User space
Kernel space
4
Worker 1
2
Global queue
1
Worker 2
...
Worker N
Sliding window stats
Function finishes in time slice T
Time slice T
FILTER
CFS

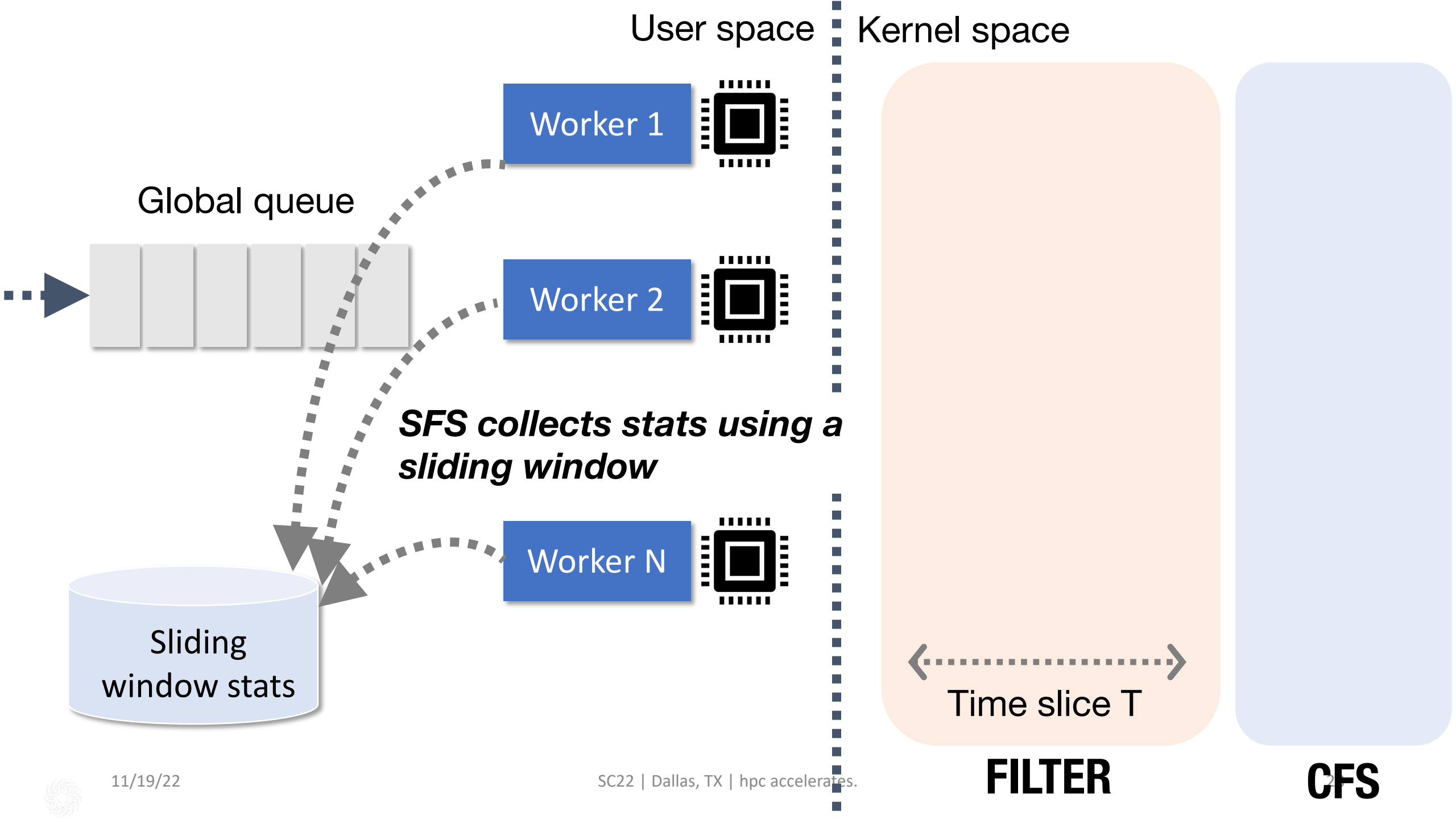

User space
Kernel space
Worker 1
Worker 2
Worker N
Global queue
1
2
3
Sliding window stats
Function executes…
Time slice T
FILTER
CFS

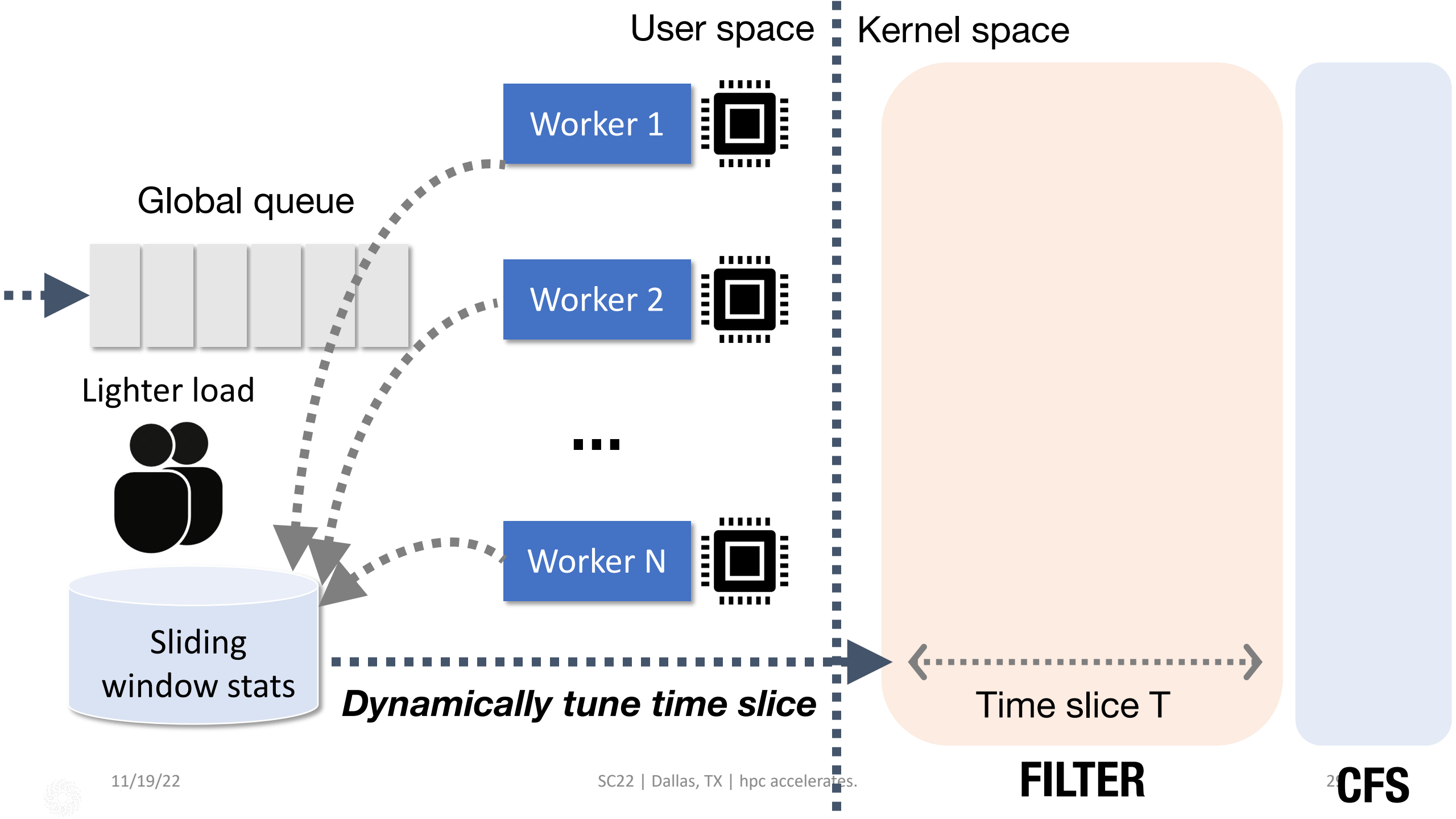

User space
Kernel space
Worker 1
Worker 2
Worker N
Global queue
1
2
Sliding window stats
Function has not finished in time slice T
4
Time slice T
FILTER
CFS

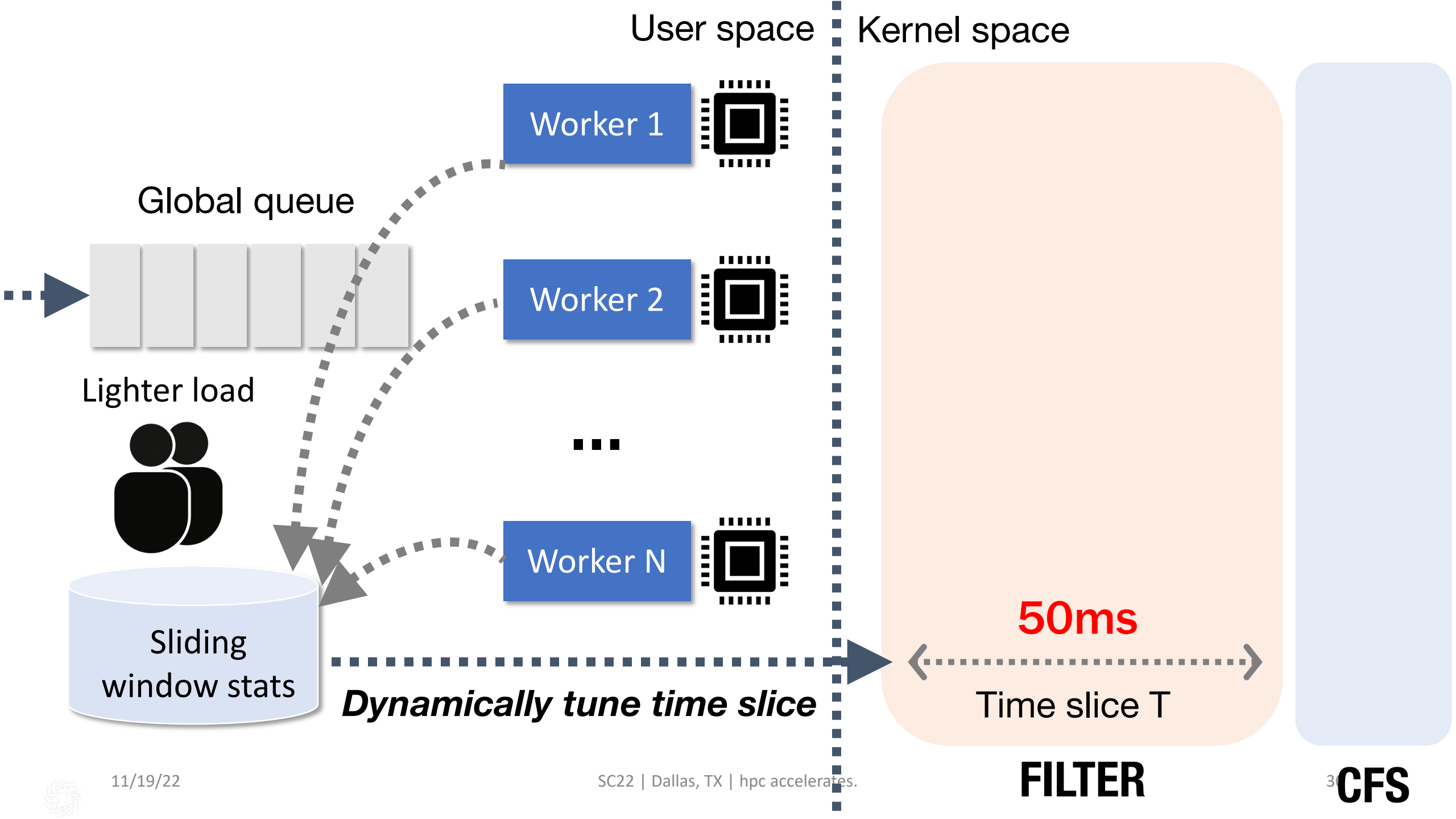

User space
Kernel space
Worker 1
Worker 2
Worker N
Global queue
Sliding window stats
Demote to CFS
5
Time slice T
FILTER
CFS

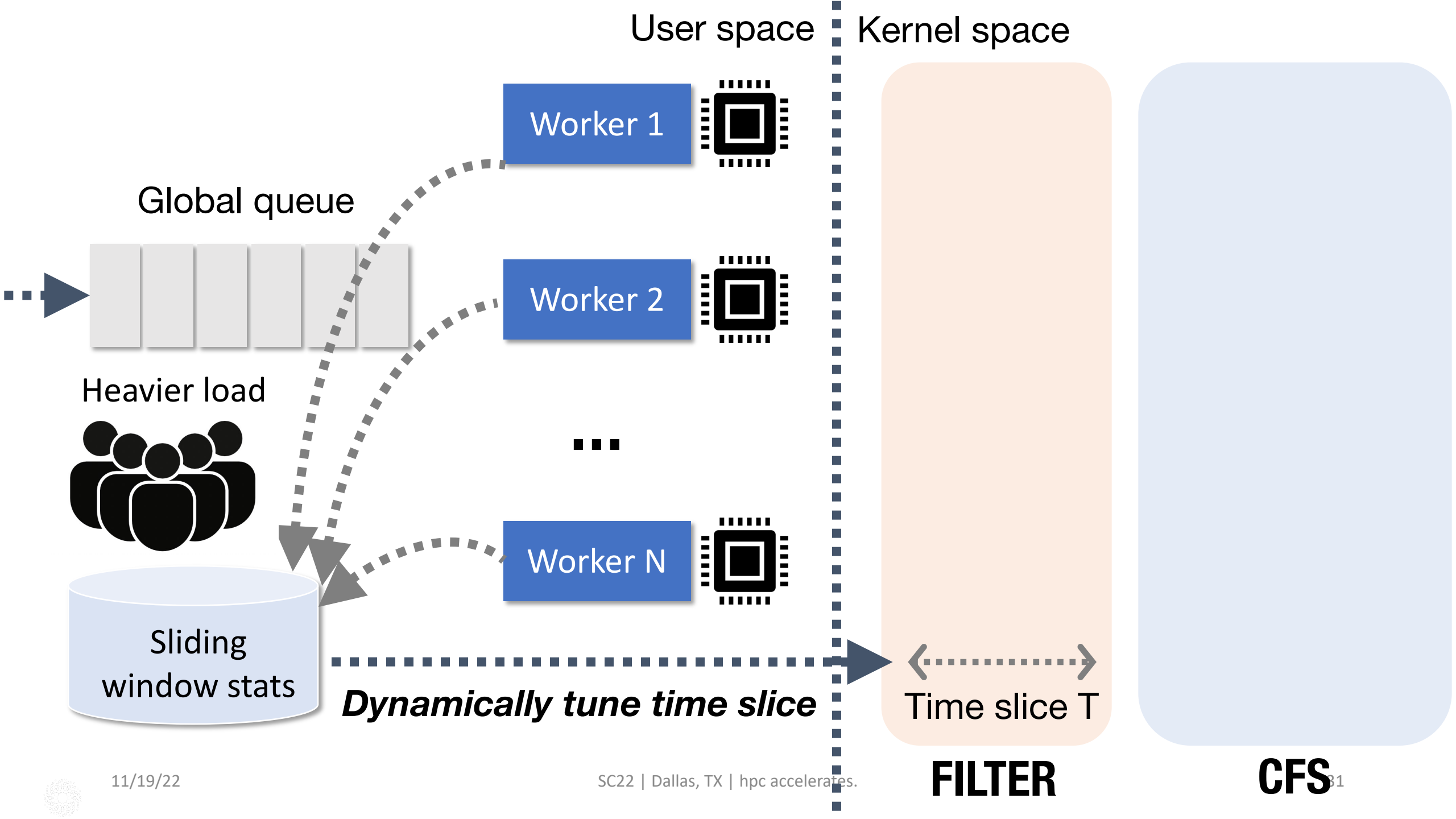

User space
Kernel space
Worker 1
Worker 2
...
Worker N
Global queue
Sliding window stats
If time slice is too long
FIFO-like scheduling
Time slice T
FILTER

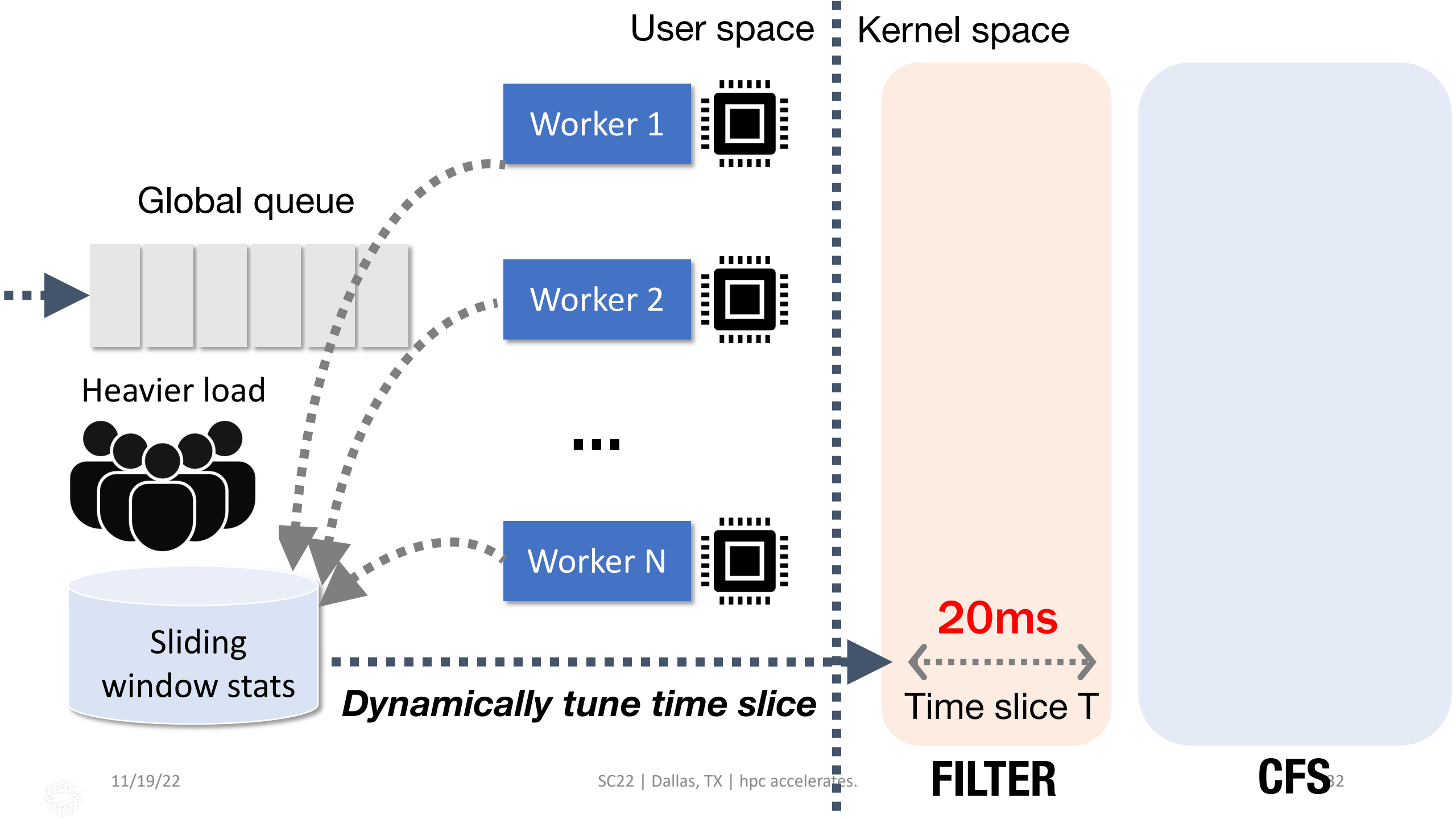

User space
Kernel space
Worker 1
Worker 2
Worker N
Global queue
Sliding window stats
If time slice is too short
CFS-like scheduling
CFS

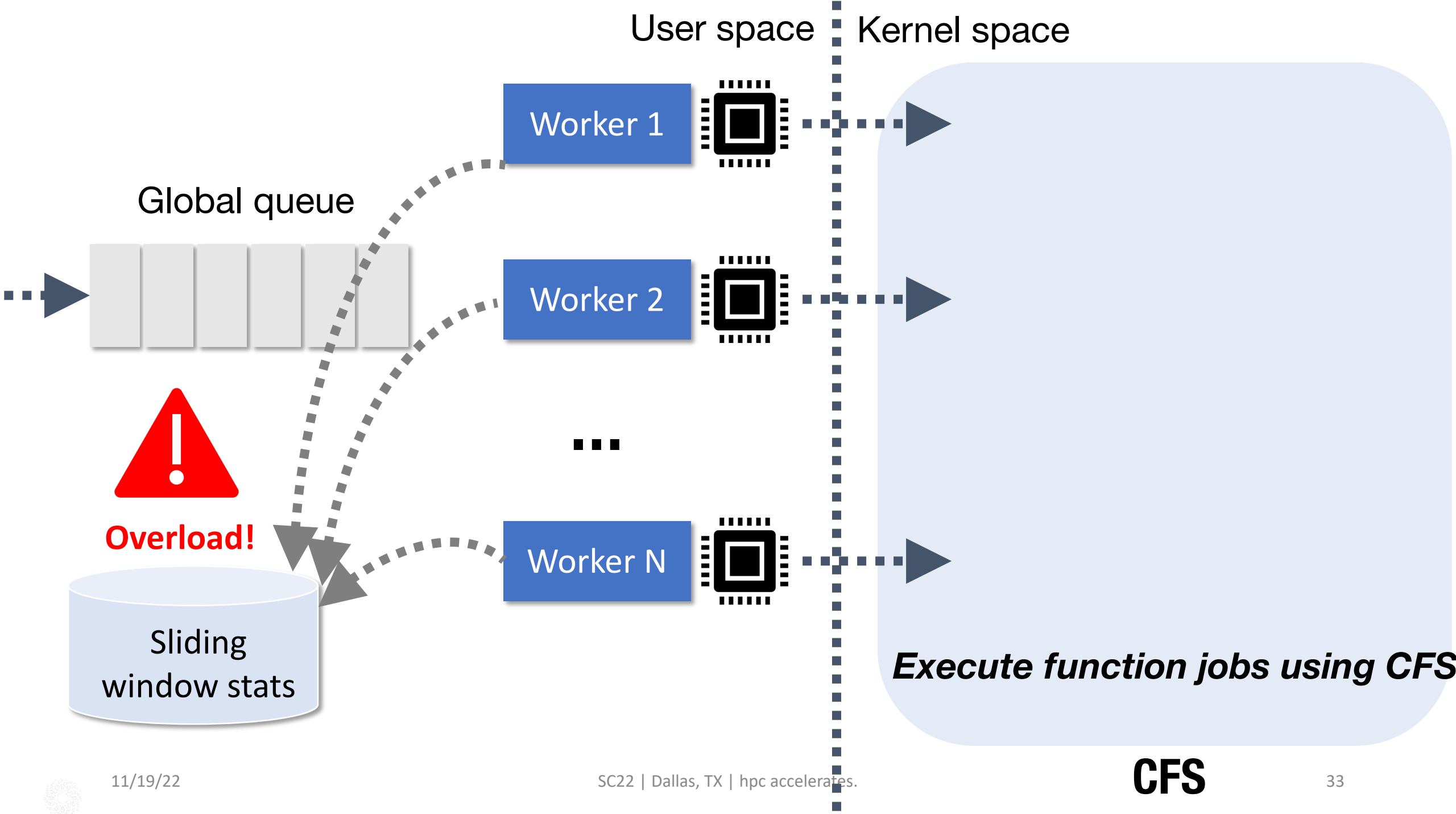

User space
Kernel space
Worker 1
Global queue
Worker 2
SFS collects stats using a sliding window
Worker N
Sliding window stats
Time slice T
FILTER
CFS


User space
Kernel space
Worker 1
Worker 2
Worker N
Global queue
Lighter load
Sliding window stats
Dynamically tune time slice
Time slice T
FILTER
CFS


User space
Kernel space
Worker 1
Worker 2
Worker N
Global queue
Lighter load
Sliding window stats
Dynamically tune time slice
50ms
Time slice T
FILTER
CFS


User space
Kernel space
Worker 1
Worker 2
Worker N
Global queue
Heavier load
Sliding window stats
Dynamically tune time slice
Time slice T
FILTER
CFS


User space
Kernel space
Worker 1
Worker 2
Worker N
Global queue
Heavier load
Sliding window stats
Dynamically tune time slice
20ms
Time slice T
FILTER
CFS


User space
Kernel space
Worker 1
Worker 2
Worker N
Global queue
Overload!
Sliding window stats
Execute function jobs using CFS
CFS

Smart function scheduler (SFS)

SFS is a **FaaS-aware**, user-space OS scheduler

Key idea: Two-level scheduling

- A FILTER level that dynamically tunes a global time slice for newly arrived functions
- Filtered functions from top level continue in CFS

Orchestrates existing Linux scheduling policies

- Short functions run to completion → Improved turnaround time
- Online policy with minimum historical stats → Simply and practical heuristic
- Transparent to both upper-level FaaS platform and underlying OSes → Ready to deploy

Outline

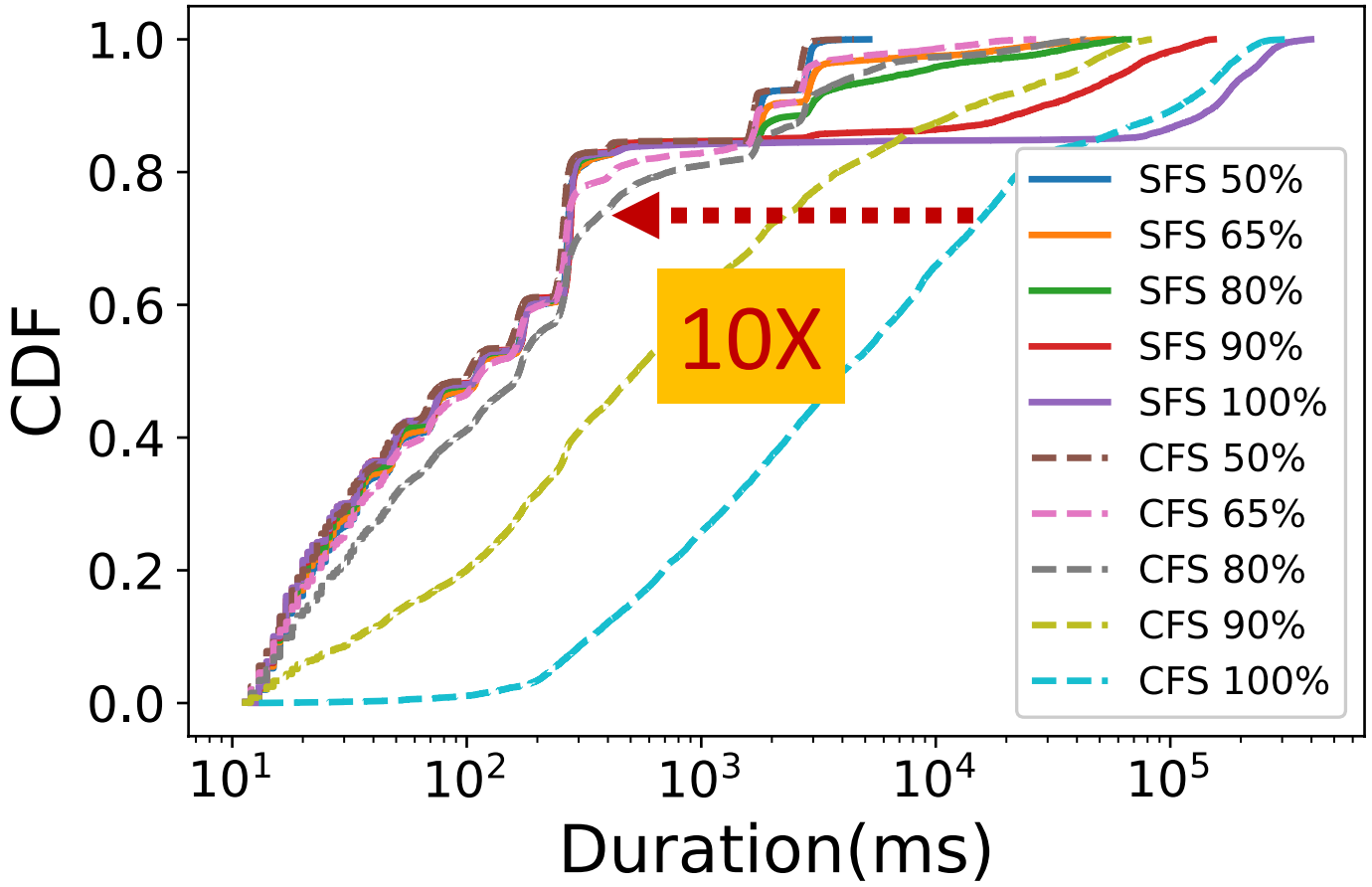
- SFS Design
- Evaluation
- Conclusion

Experimental Setup

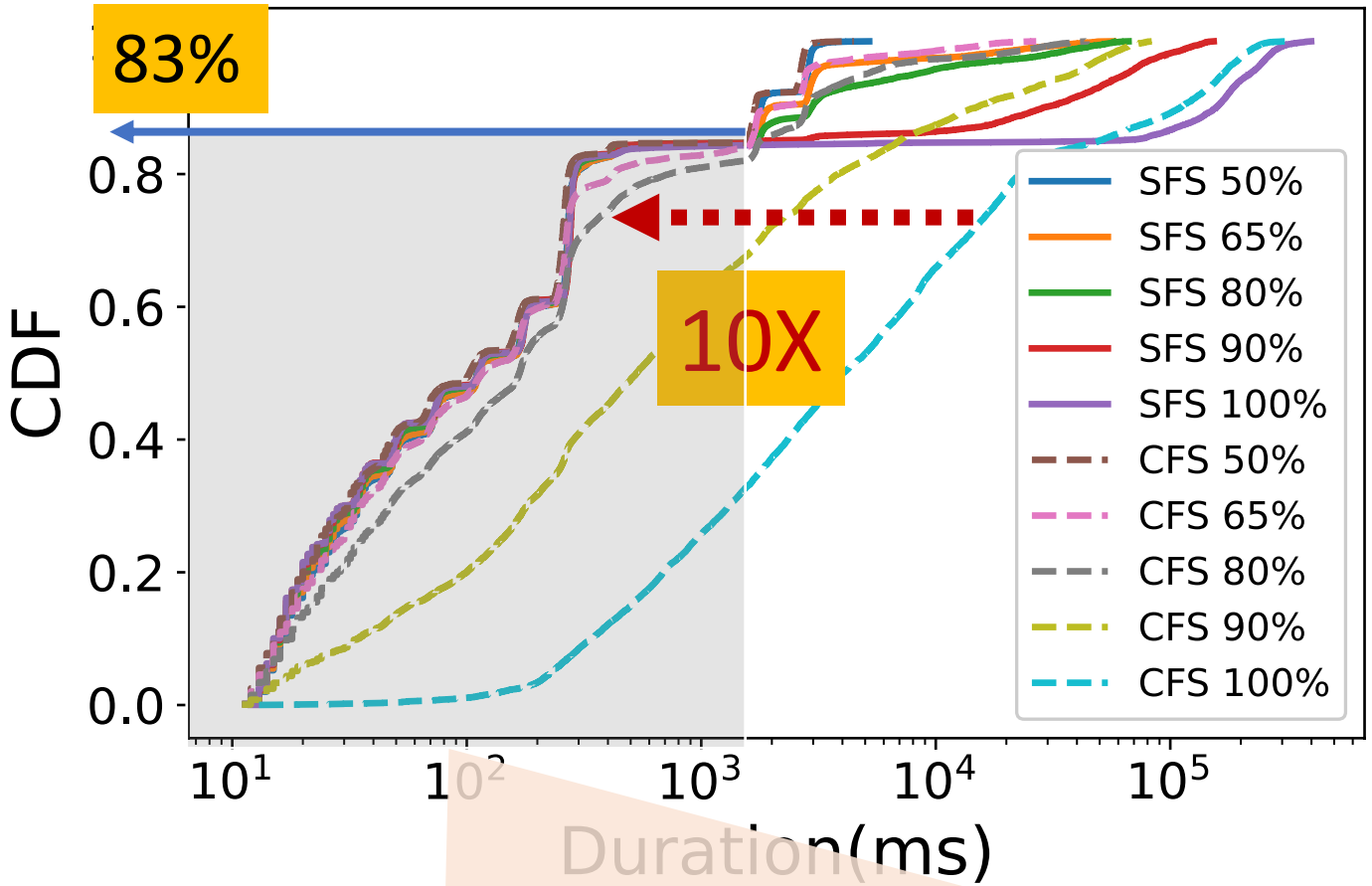
- Standalone
  - **16-core** EC2 VM
- SFS-ported OpenLambda* (*HotCloud' 16)*
  - **72-core** EC2 bare-metal VM
  - By modifying **29 lines** of Go/Python code in OpenLambda
- Day one of the Azure Functions Trace
  - 49, 712 function requests
  - Breakdowns (min, median, max, percentiles)

* "Serverless Computation with {OpenLambda}." Hendrickson et al. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*. 2016.

SFS Standalone – Turnaround time

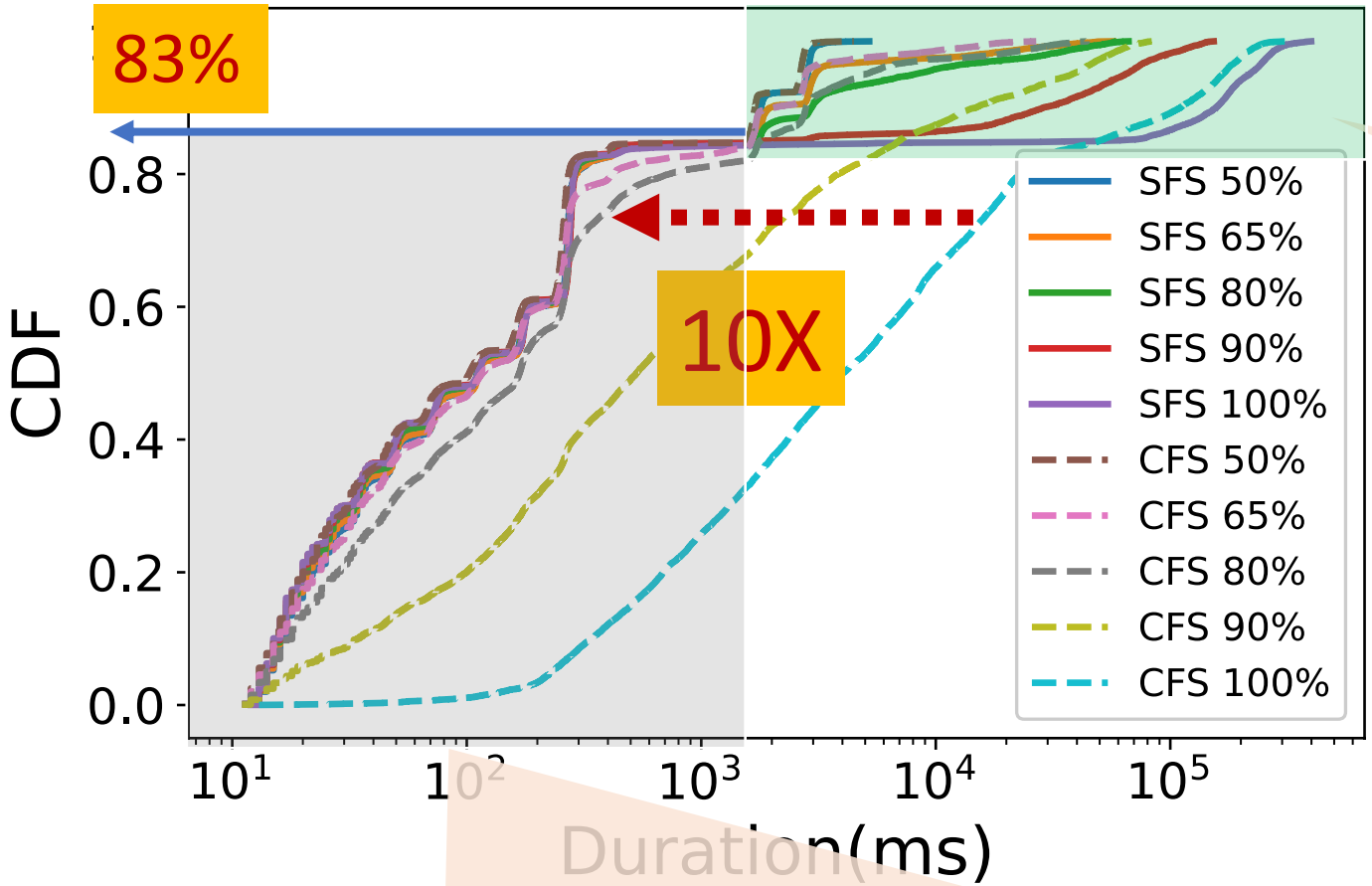


SFS Standalone – Turnaround time



SFS maintains almost identical performance for 83% of the function requests across all load levels

SFS Standalone – Turnaround time

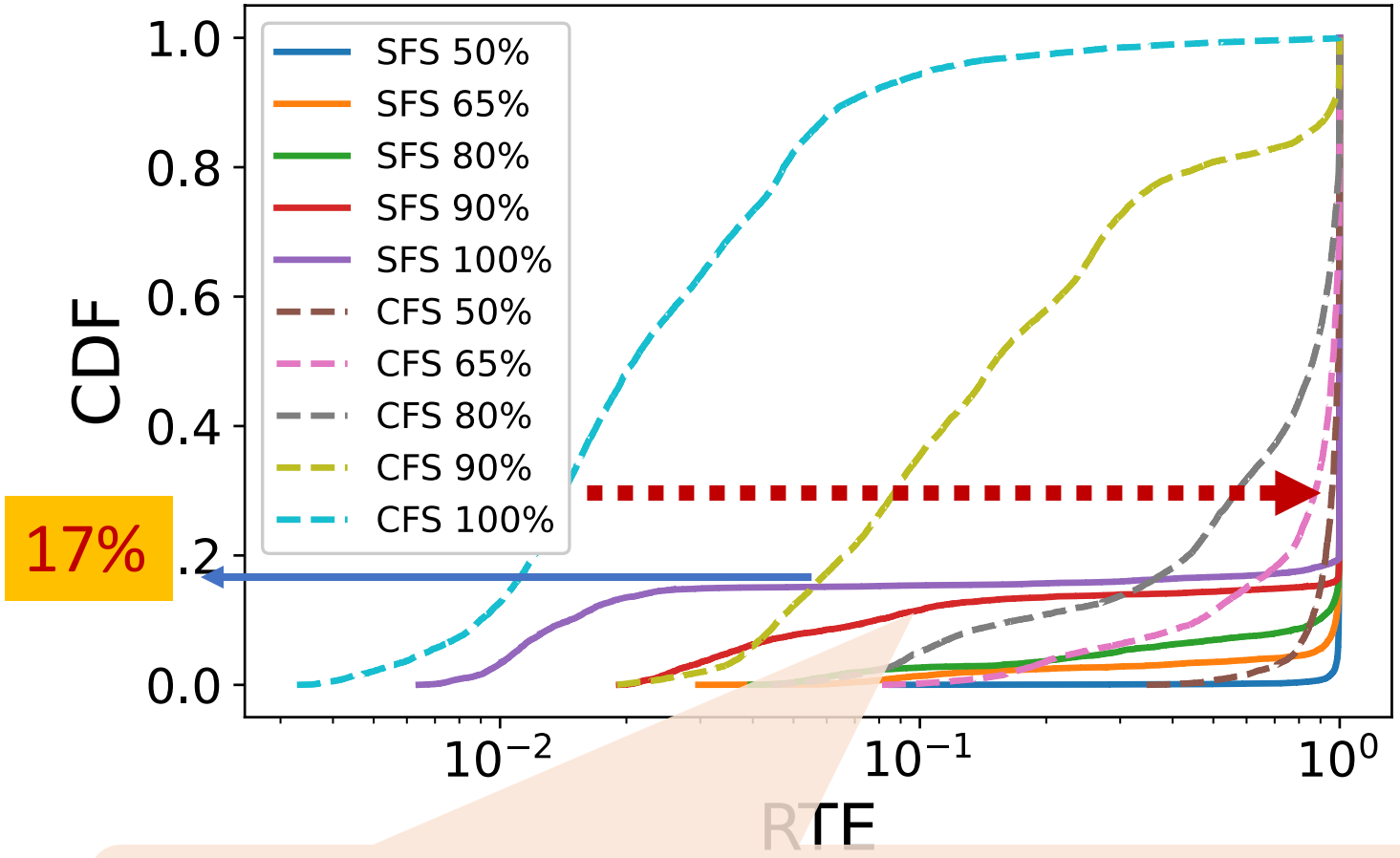


Remaining longer jobs observed slightly higher tail latency

SFS maintains almost identical performance for 83% of the function requests across all load levels

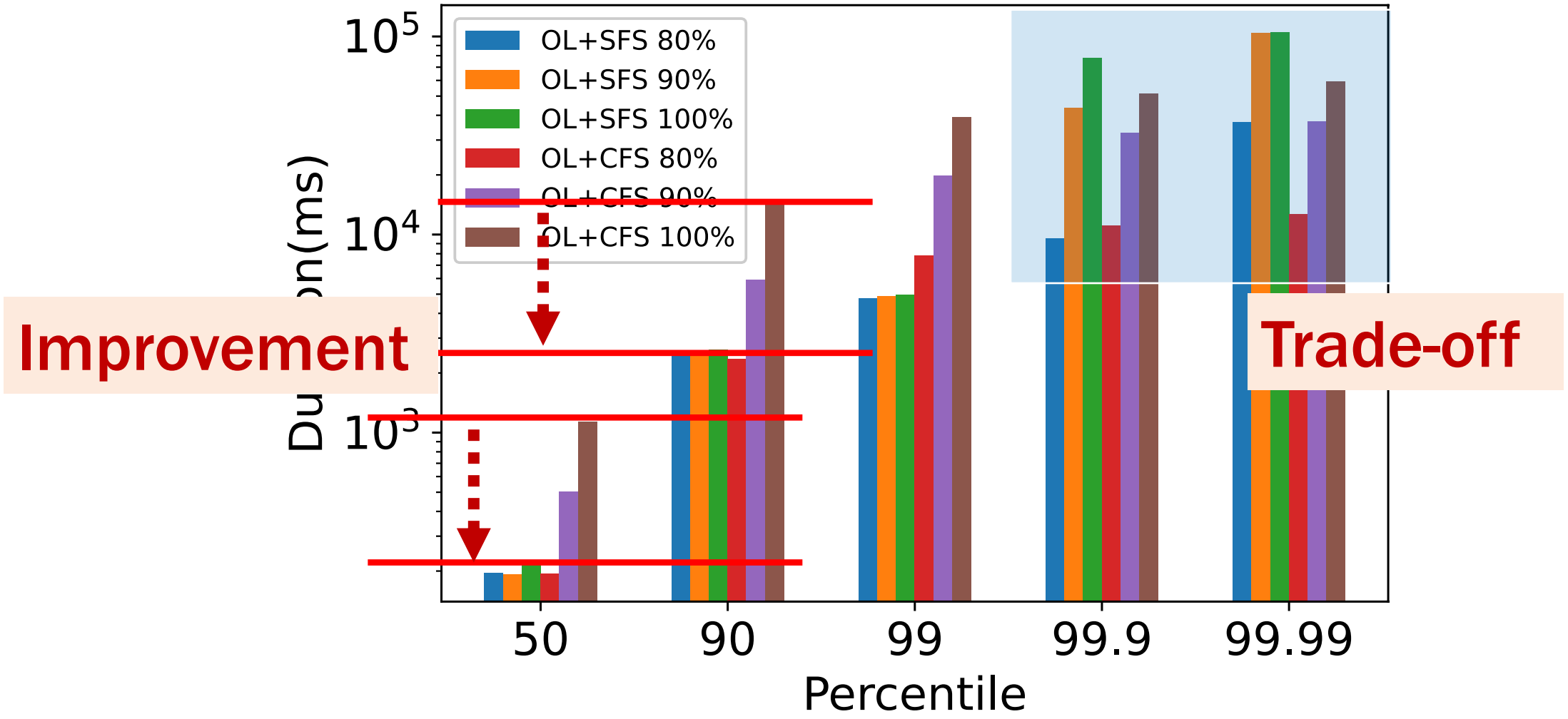
SFS Standalone – RTE

Function **Run-Time Effectiveness**
= Service time / Turnaround time

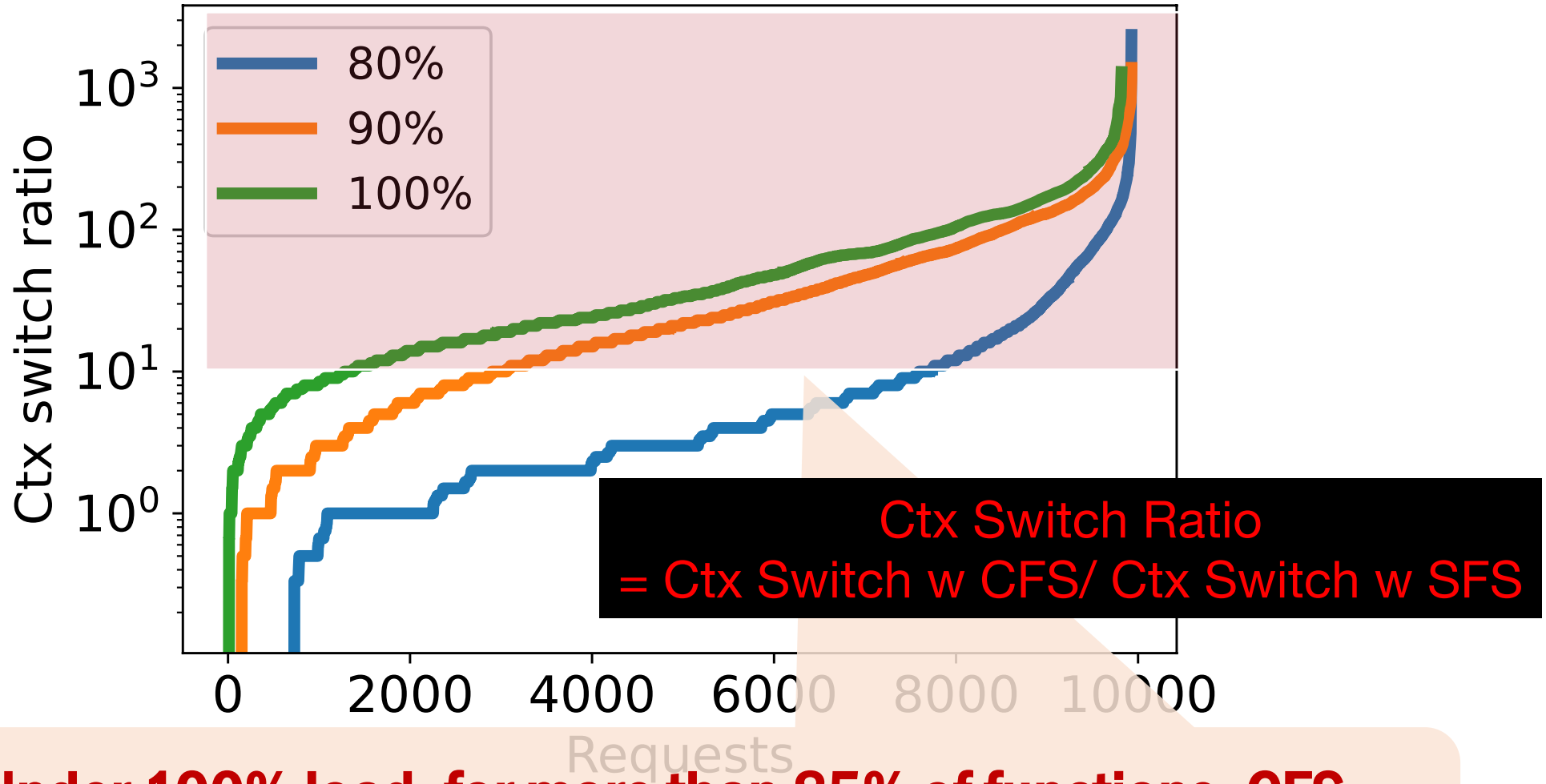


SFS performs optimal RTE for short functions

SFS-ported OpenLambda



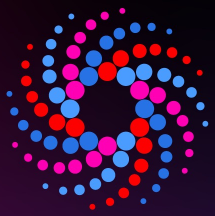
SFS-ported OpenLambda



Under 100% load, for more than 85% of functions, CFS suffers 10X more context switches than SFS

Conclusion

- SFS addresses the poor performance issue of CFS in FaaS workloads through a two-level scheduling approach
- SFS adaptively tunes a high-priority FILTER pool that optimizes the performance of short functions
- Experimental results show SFS outperforms CFS up to 50x for a production FaaS workload



SC22
Dallas, TX | hpc accelerates.

Thank You
Questions?

- Contact: Yuqi Fu (jwx3px@Virginia.edu)
- https://github.com/ds2-lab/SFS

National Science Foundation

CloudLab

Back slides