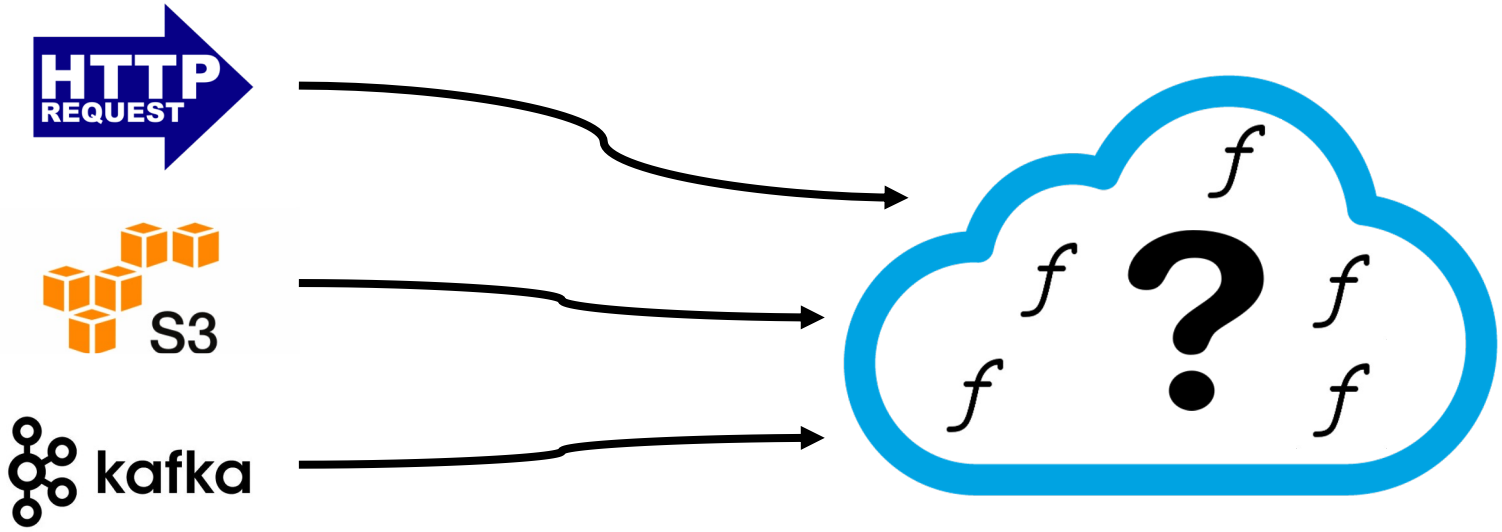
Hermod: Principled and Practical Scheduling for Serverless Functions

Kostis Kaffes, Neeraja J. Yadwadkar, Christos Kozyrakis

Serverless Computing is **Convenient** for **Users**

Users:

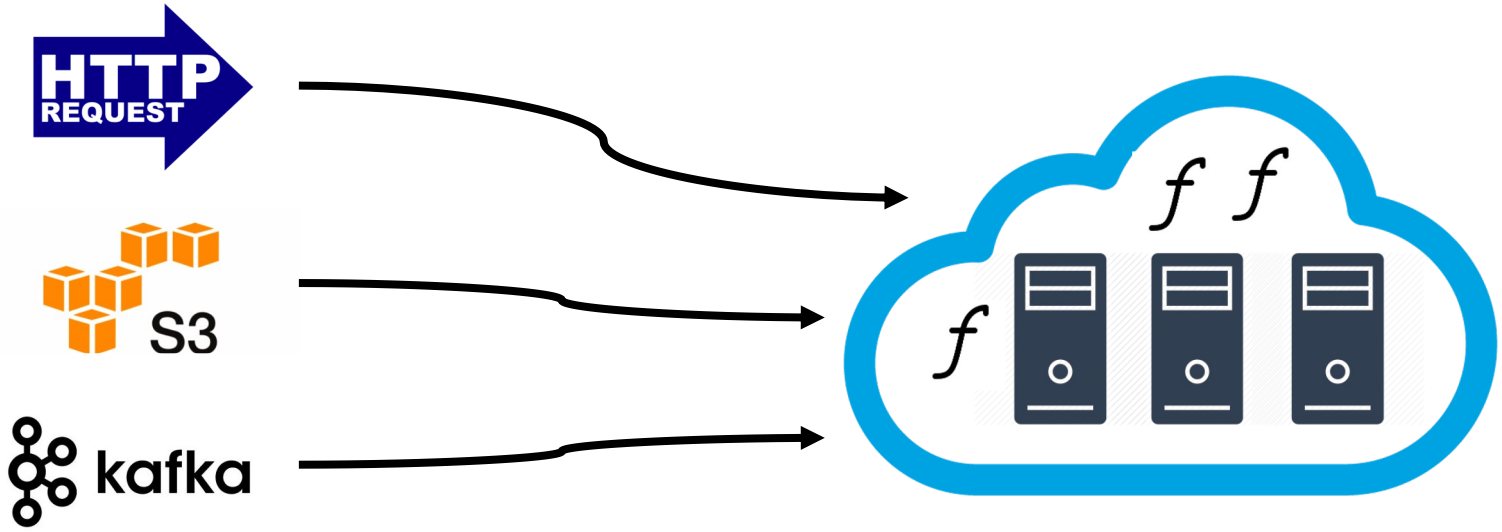
- Define a function
- Specify events as execution triggers
- Pay only for the actual runtime of the function activation



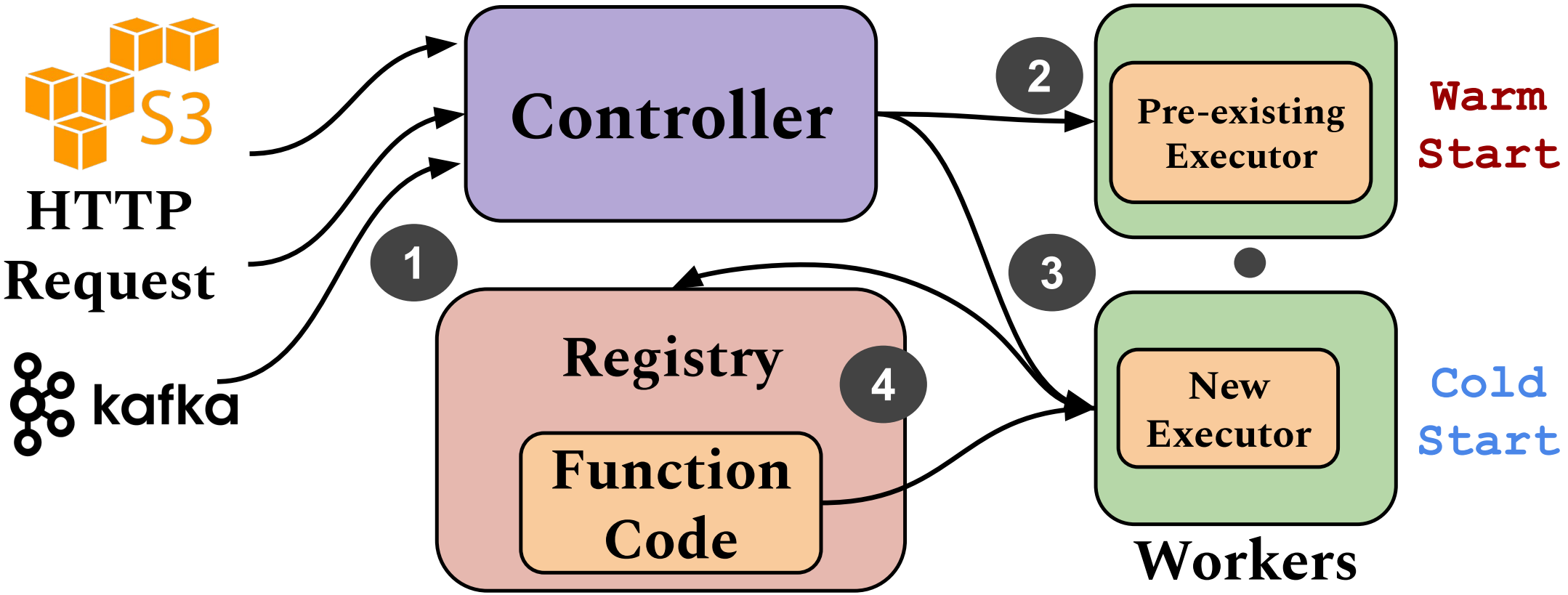
Serverless Computing is **Challenging** for **Providers**

Providers need to manage:

- Function Placement
- Scaling
- Runtime Environment



Serverless Function Lifecycle



Serverless Scheduling Goals

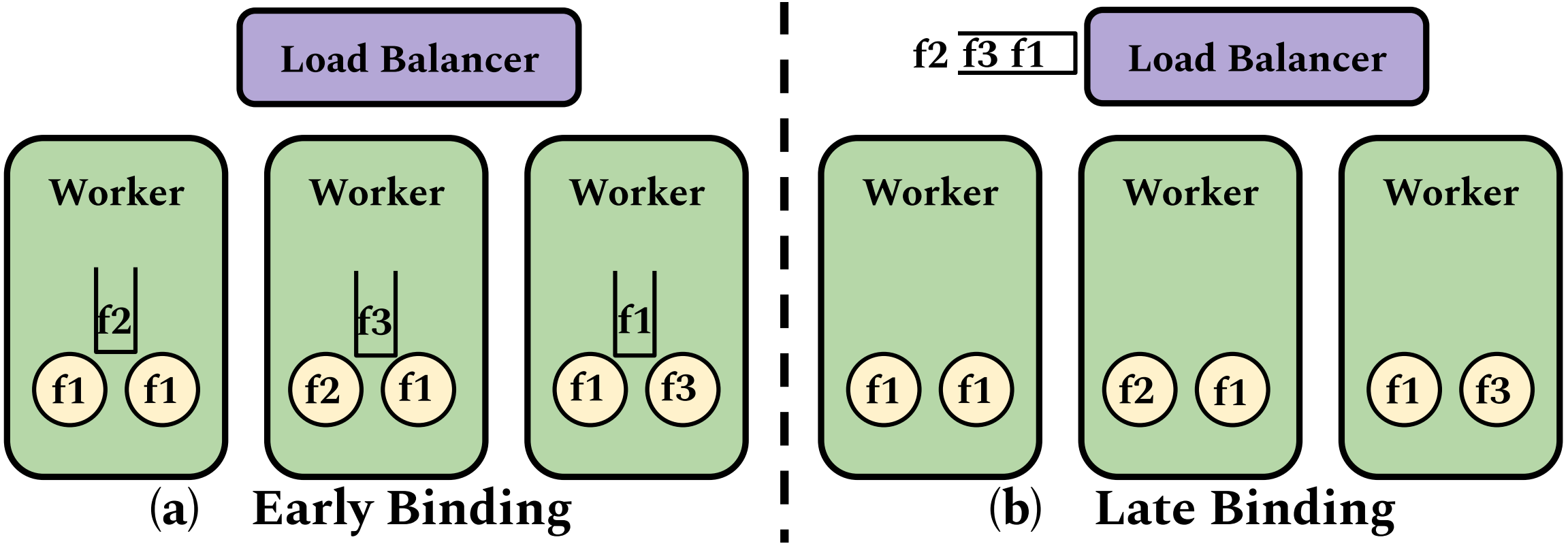
Serverless schedulers need to be:

- **Load-aware** – Avoid excessive queueing
- **Cost-aware** – Use as few servers as possible
- **Locality-aware** – Avoid cold starts

Serverless Scheduling Decisions

- When an invocation should be scheduled to a Worker?
- Which Worker should handle each invocation?
- Which intra-Worker scheduling policy should be used?

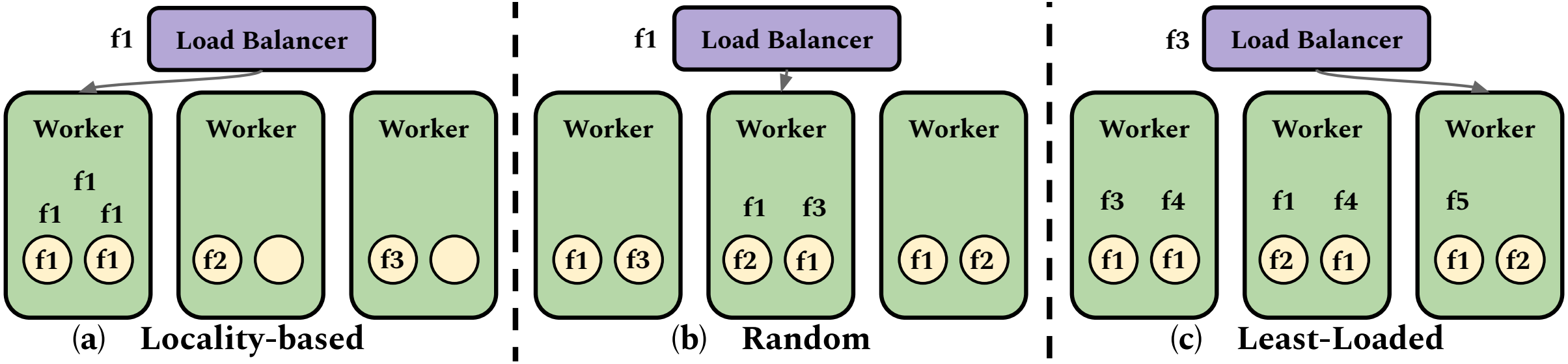
When an invocation should be scheduled to a Worker?



+ Tasks always ready to execute
-- Imperfect load balancing

+ Perfect load balancing
-- Head-of-line blocking

Where should a function invocation execute?



Which intra-Worker scheduling policy should be used?

- First-Come-First-Serve
- Processor-Sharing

Serverless Scheduling Taxonomy

T / LB / S

T: Type of binding used (early E vs. late L)
LB: LOC – locality-based
LL – least-loaded
R – random
S: intra-Worker policy
FCFS – First-Come-First-Serve
PS – Processor Sharing

Existing Approaches

System	Policy	Load-aware	Cost-aware	Locality-aware
OpenWhisk	E/LOC/PS	✗	✗	✓
kNative	E/R/PS	✗	✗	✓
Sparrow	Late Binding	✓	✗	✗
Hermod	E/Hybrid/PS	✓	✓	✓

Exploring the Policy Space using Queueing Simulation

Setup: 4 Workers – 12 cores each

Metric: 99% Slowdown $= \frac{execution\ time + queueing + scheduling}{execution\ time}$

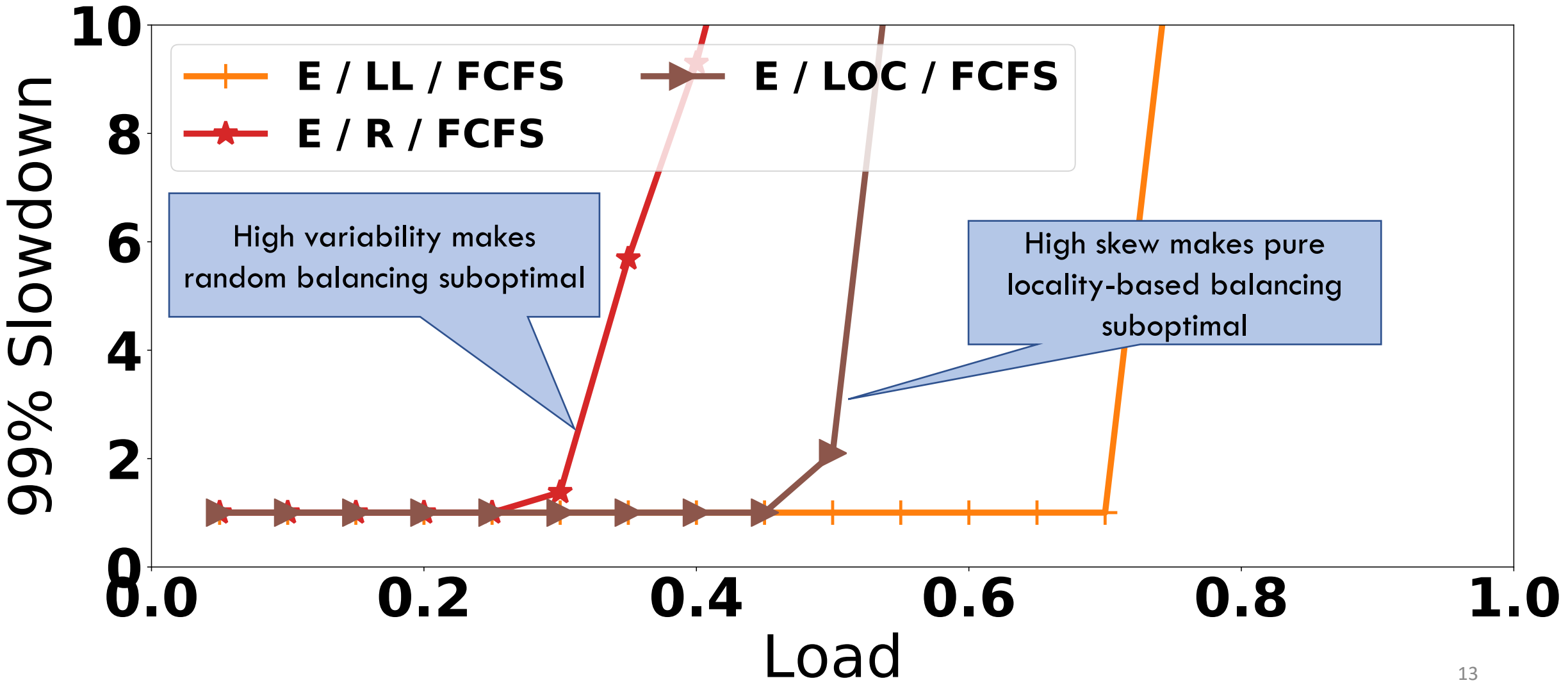
Workload = Azure Trace [ATC 2020]

Highly-variable execution times

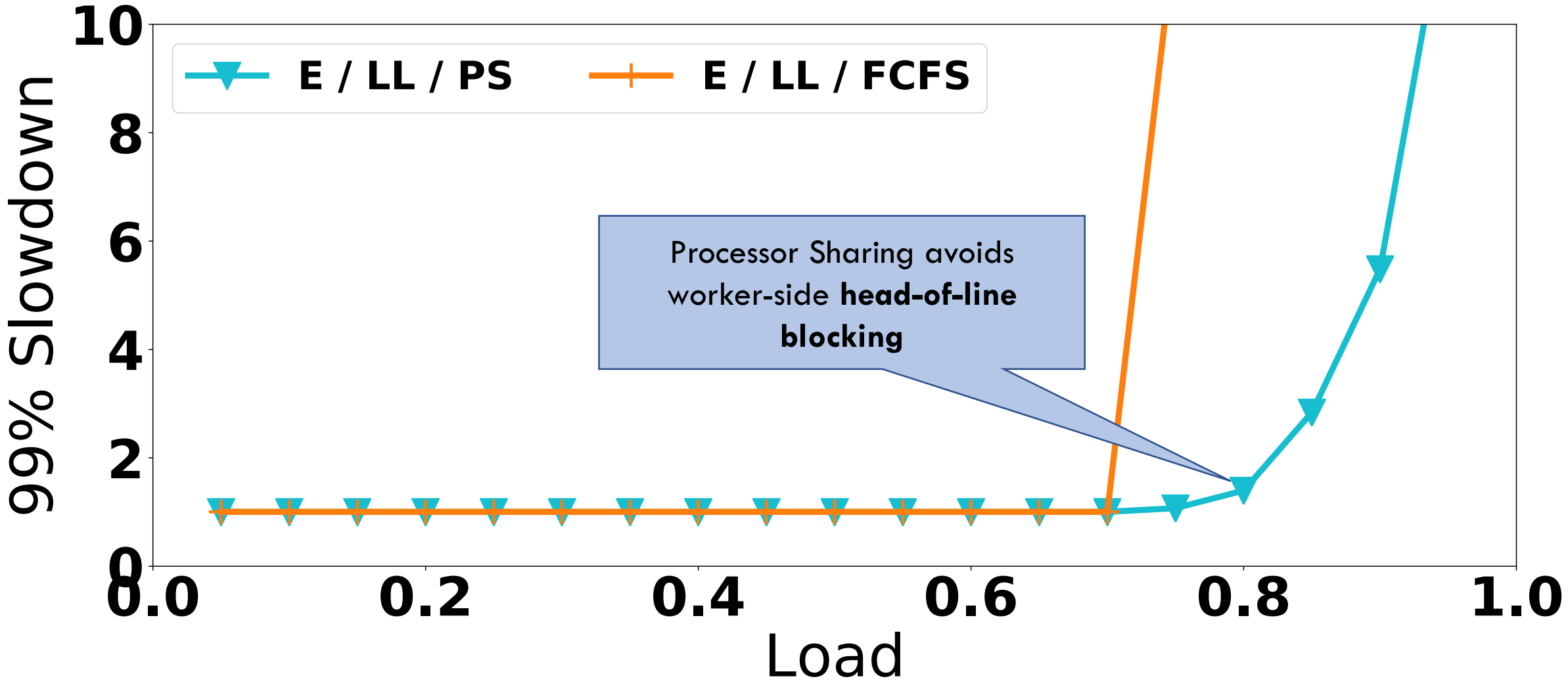
Highly-skewed invocations

50 functions

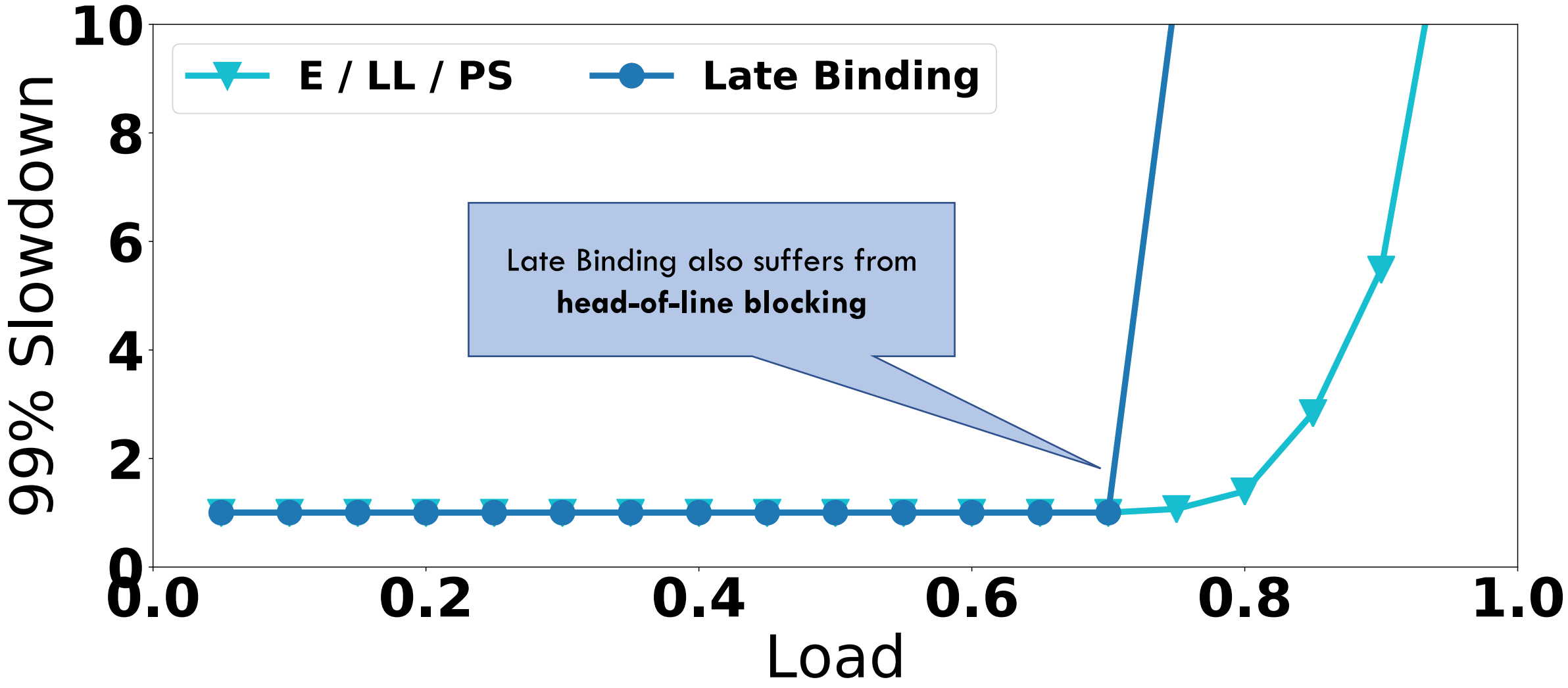
Least-Loaded Balancing Dominates



Processor-Sharing in the Workers is Necessary



Processor-Sharing in the Workers is Necessary



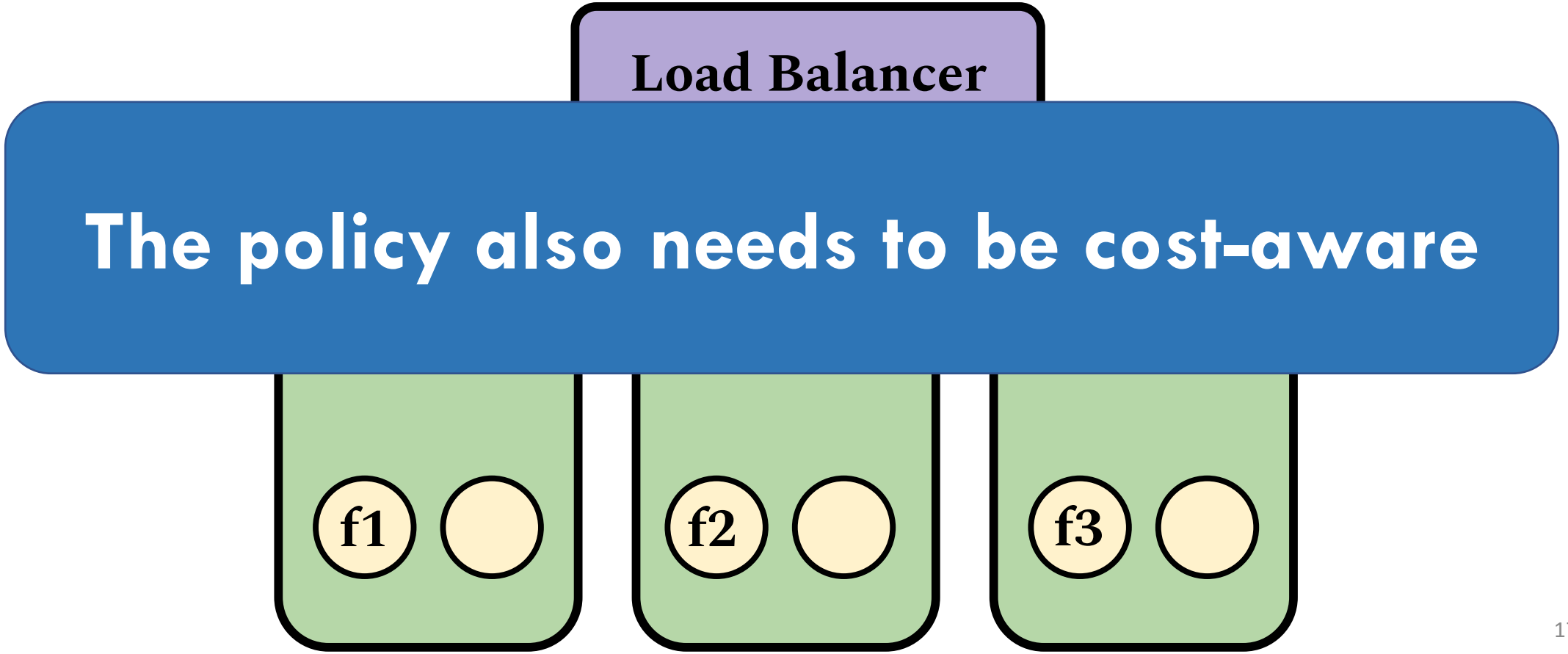
Conclusion

The **load-aware** E/LL/PS policy is optimal

NO

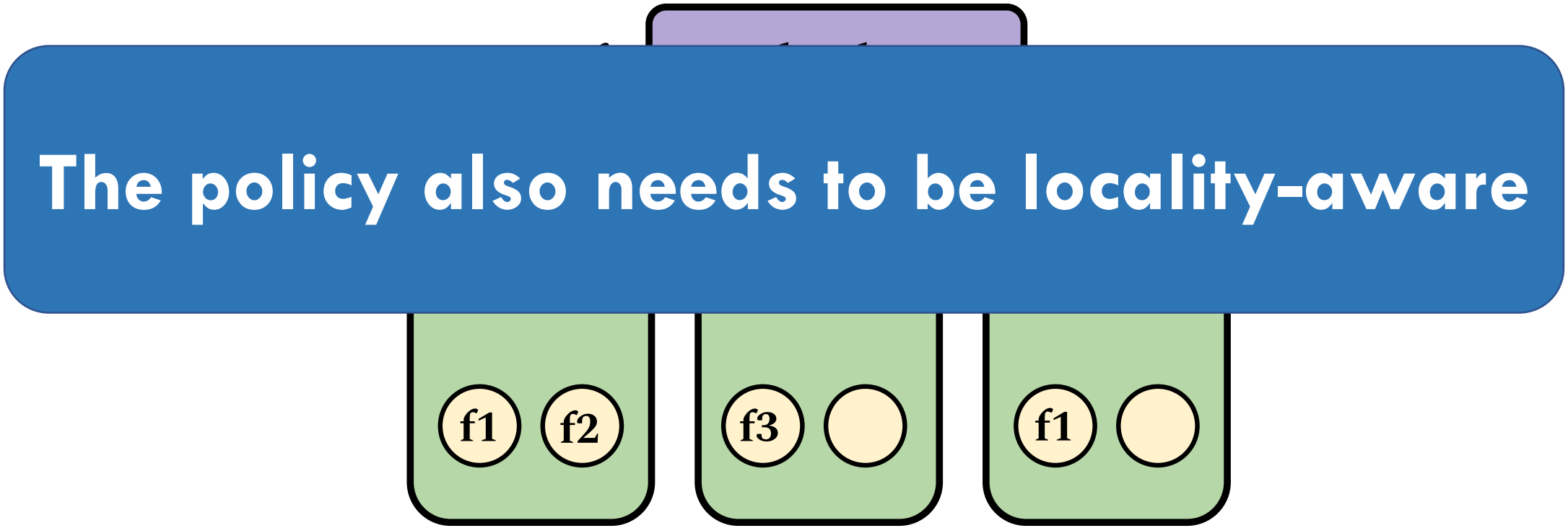
E/LL/PS Suffers from Practical Problems

1. **Low resource efficiency**



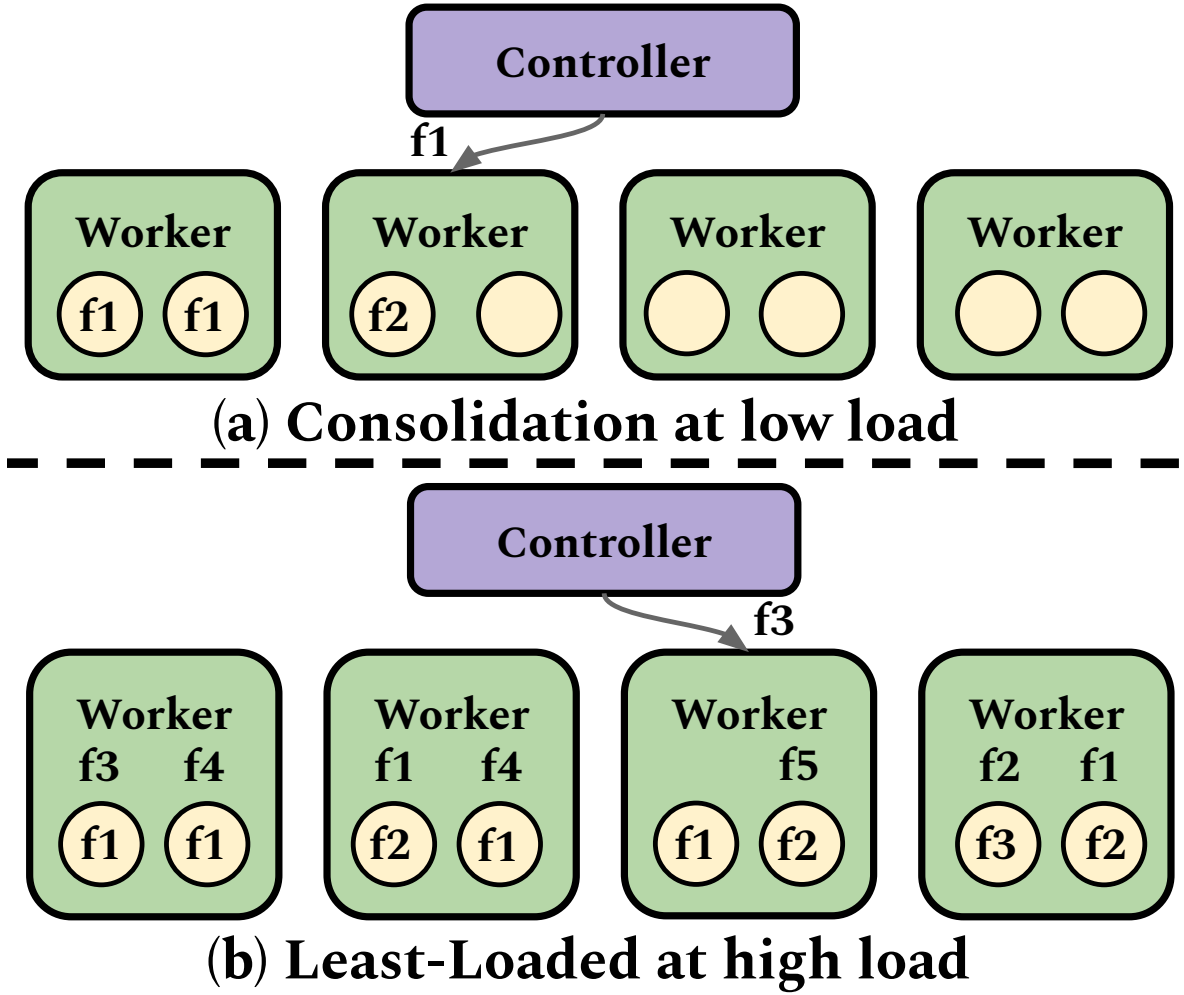
E/LL/PS Suffers from Practical Problems

1. Low resource efficiency
2. **Increased Cold Starts**



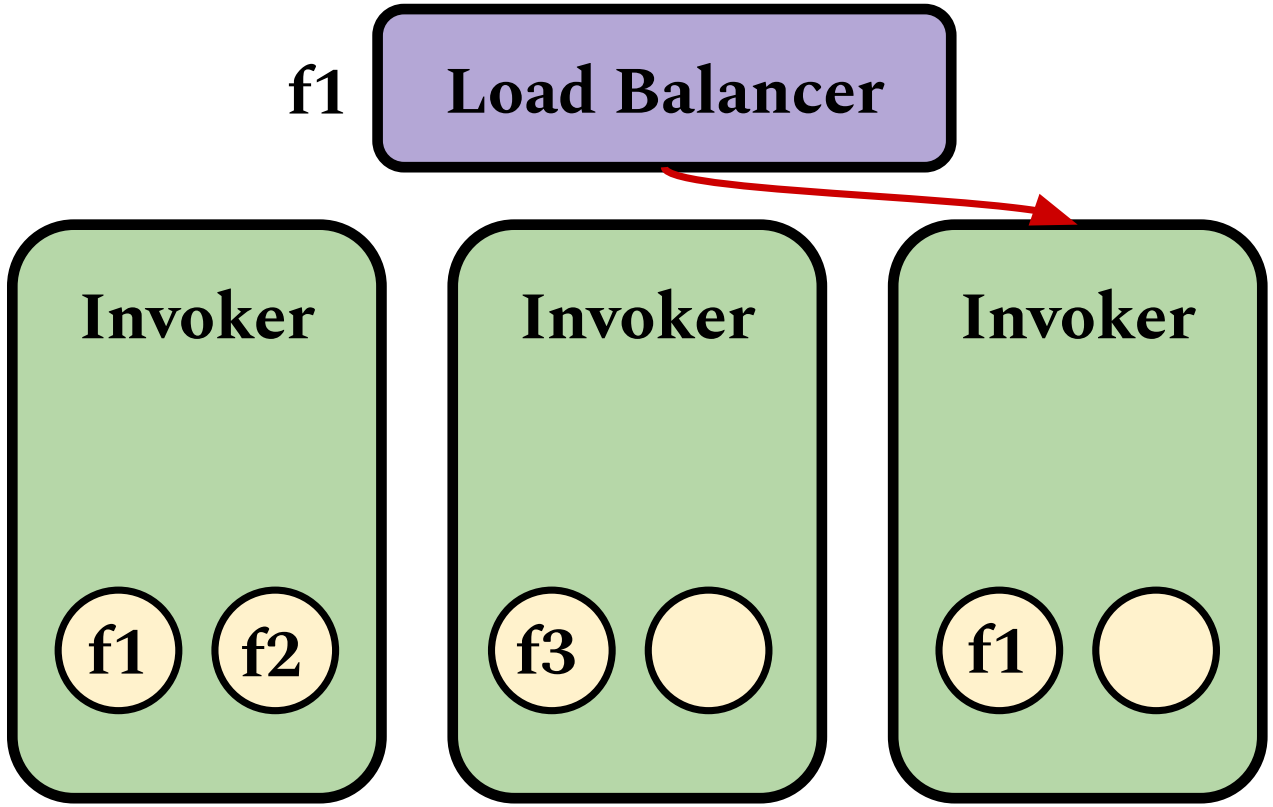
Solution: Hermod (E / Hybrid / PS)

cost-aware hybrid load balancing



Solution: Hermod (E / Hybrid / PS)

locality-aware load balancing when cost and load allow it



Evaluation

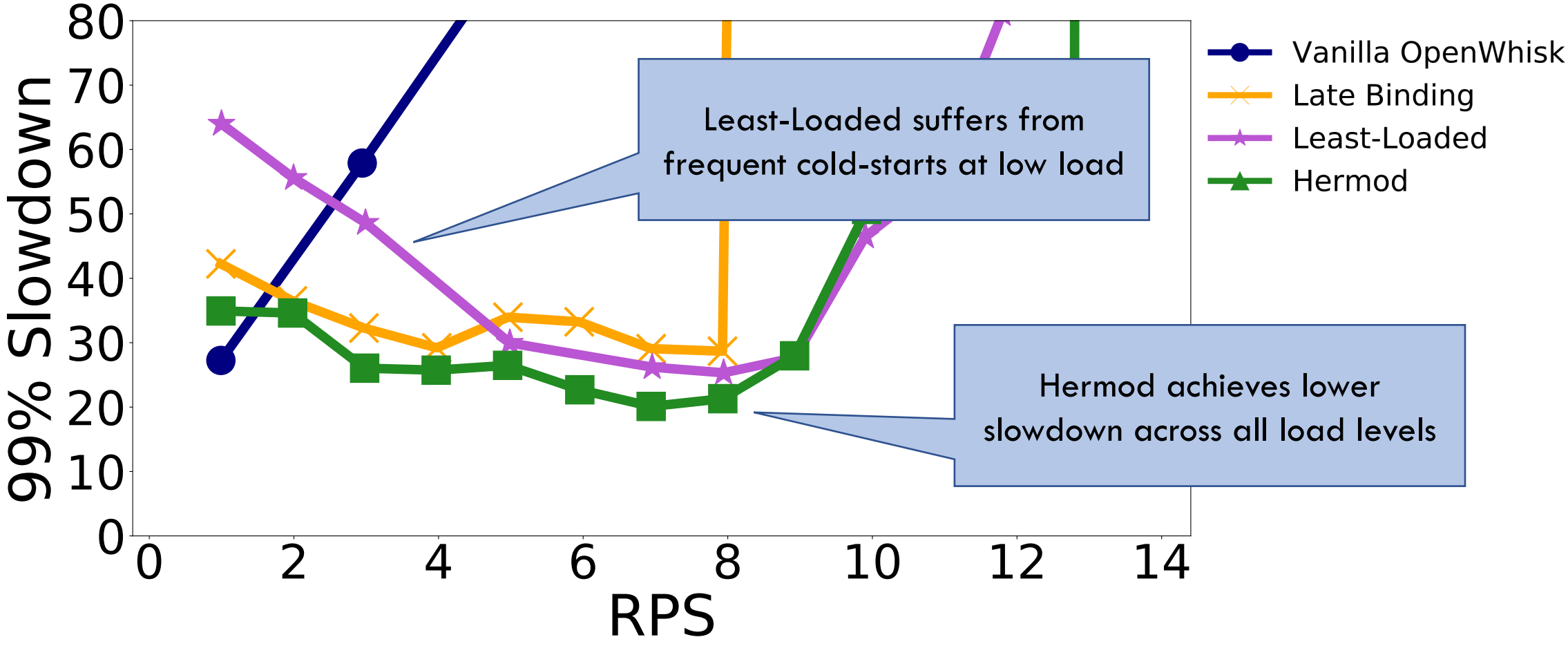
Baselines

- OpenWhisk (E/LOC/PS)
- Late Binding (Sparrow)
- Least-Loaded (E/LL/PS)

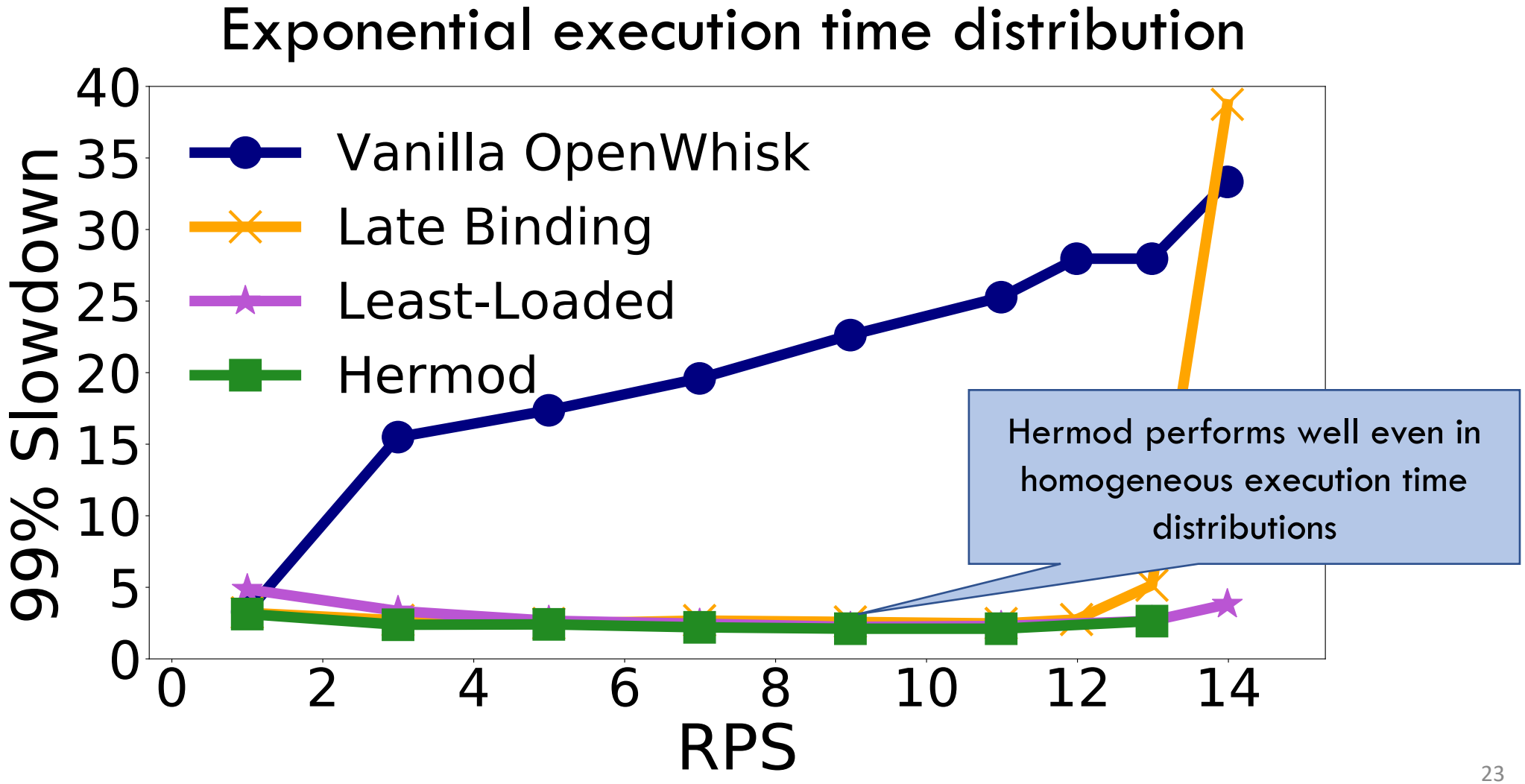
Testbed: 9 x 12-core servers

Workload: Azure Trace scaled down to 50 functions

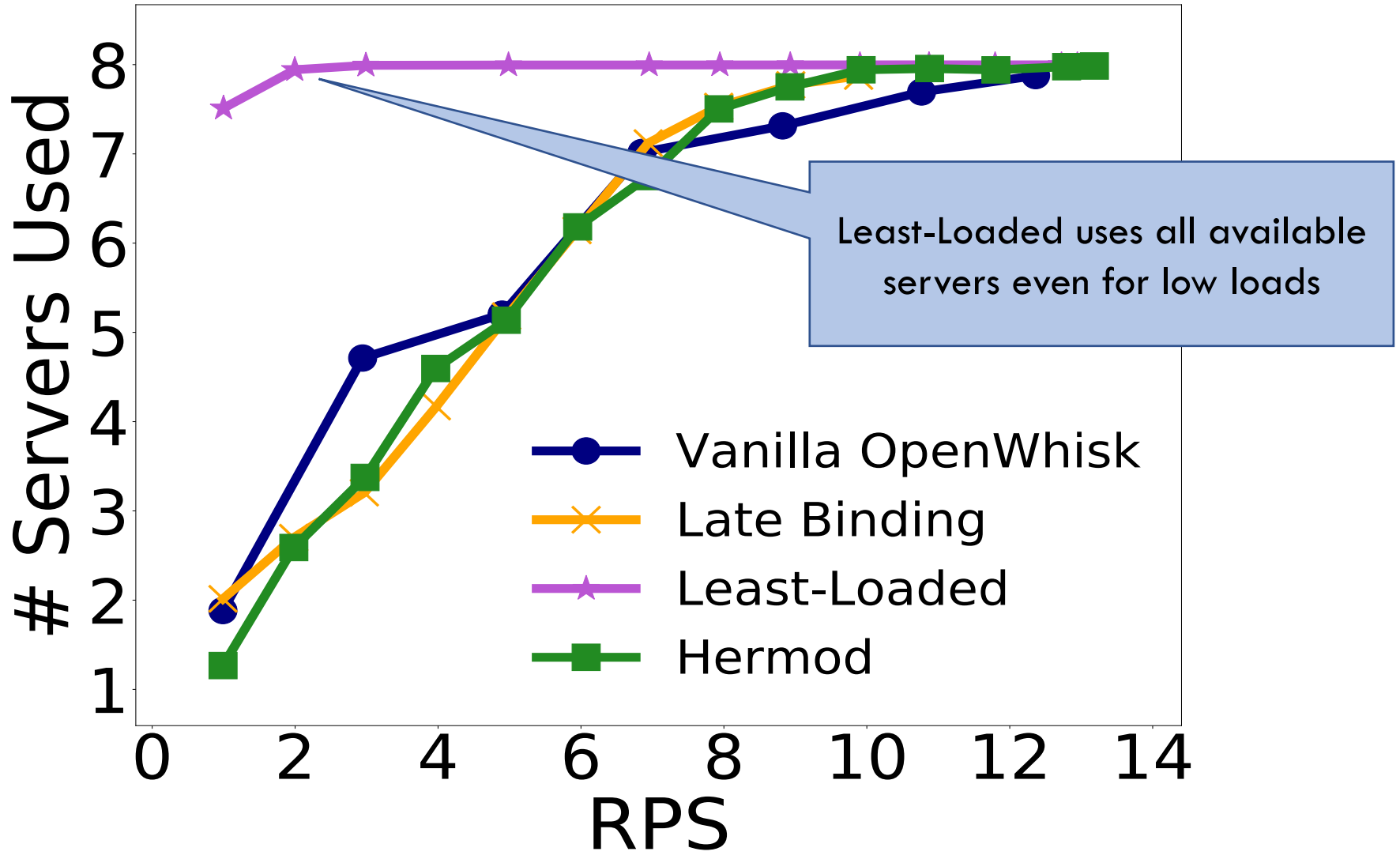
How does Hermod improve performance?



How does Hermod behave with different distributions?



How does Hermod affect resource consumption?



More details in the paper

- Simulation of larger setups and more complex policies (SRPT)
- Median and tail latency results
- More workloads
- Cold start analysis
- Overhead analysis

Find me at:
kkaffes@stanford.edu

Conclusion

Serverless schedulers need to be:

- Load-aware
- Cost-aware
- Locality-aware

Hermod achieves these goals using three key techniques:

✓ Early Binding
✓ Hybrid Load Balancing
✓ Processor Sharing