HPDC '23, June 16–23, 2023, Orlando, FL, USA

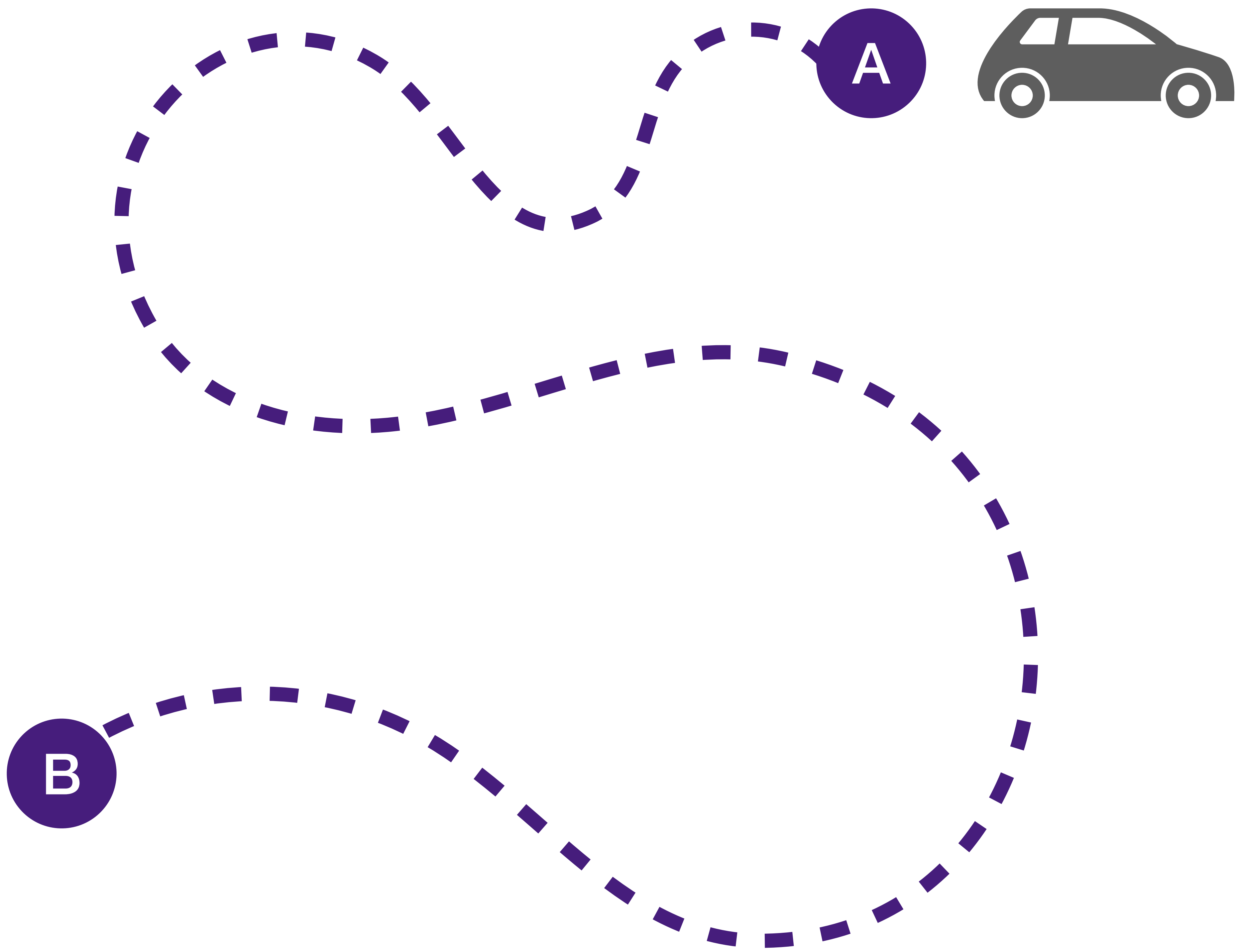
Libra: Harvesting Idle Resources Safely and Timely in Serverless Clusters

Hanfei Yu, Christian Fontenot, Hao Wang, Jian Li*, Xu Yuan†, Seung-Jong Park

Louisiana State University, SUNY-Binghamton University*, University of Louisiana at Lafayette†

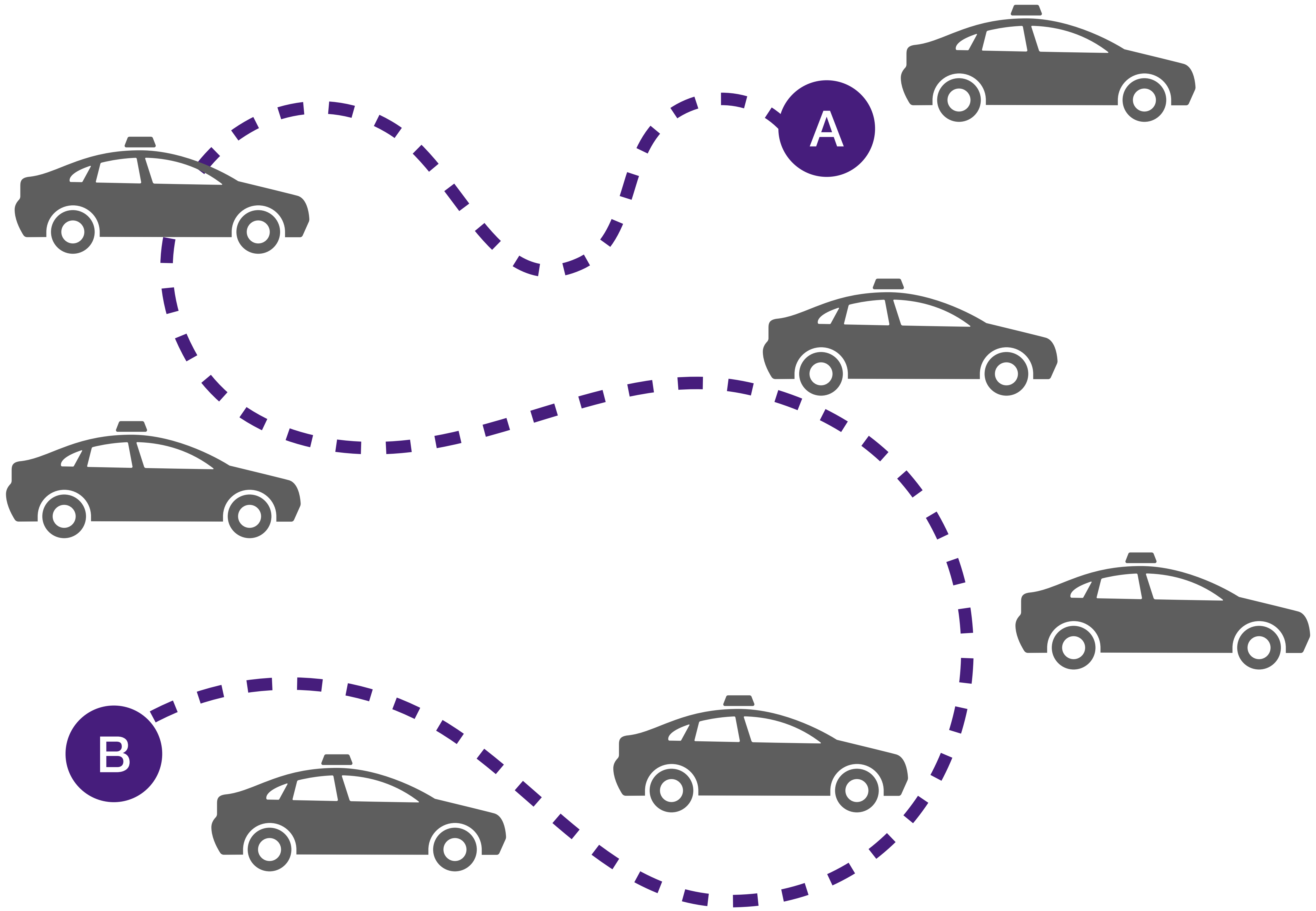
IntelliSys Lab

Cloud / HPC



Car rental

Serverless

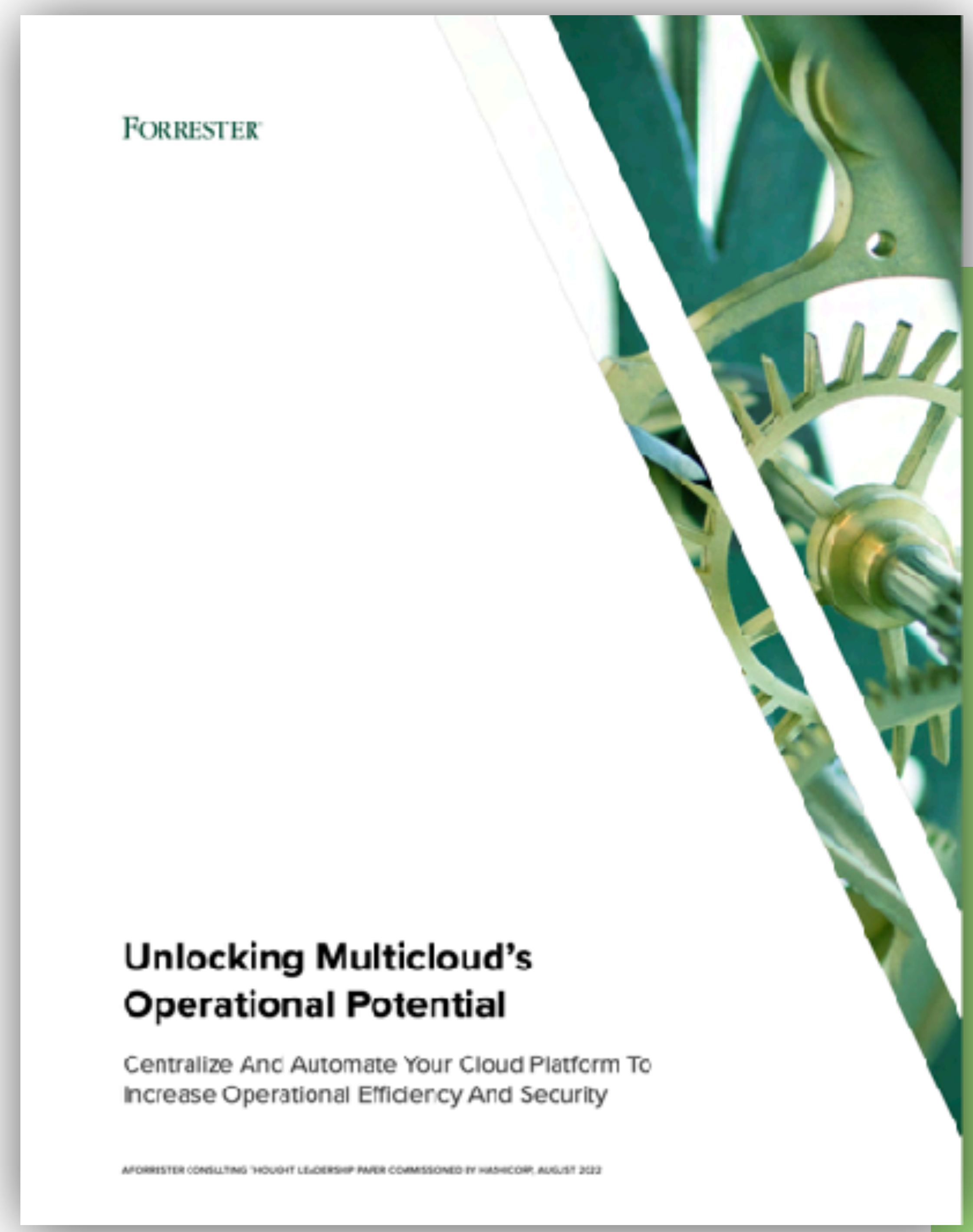


Uber/Lyft

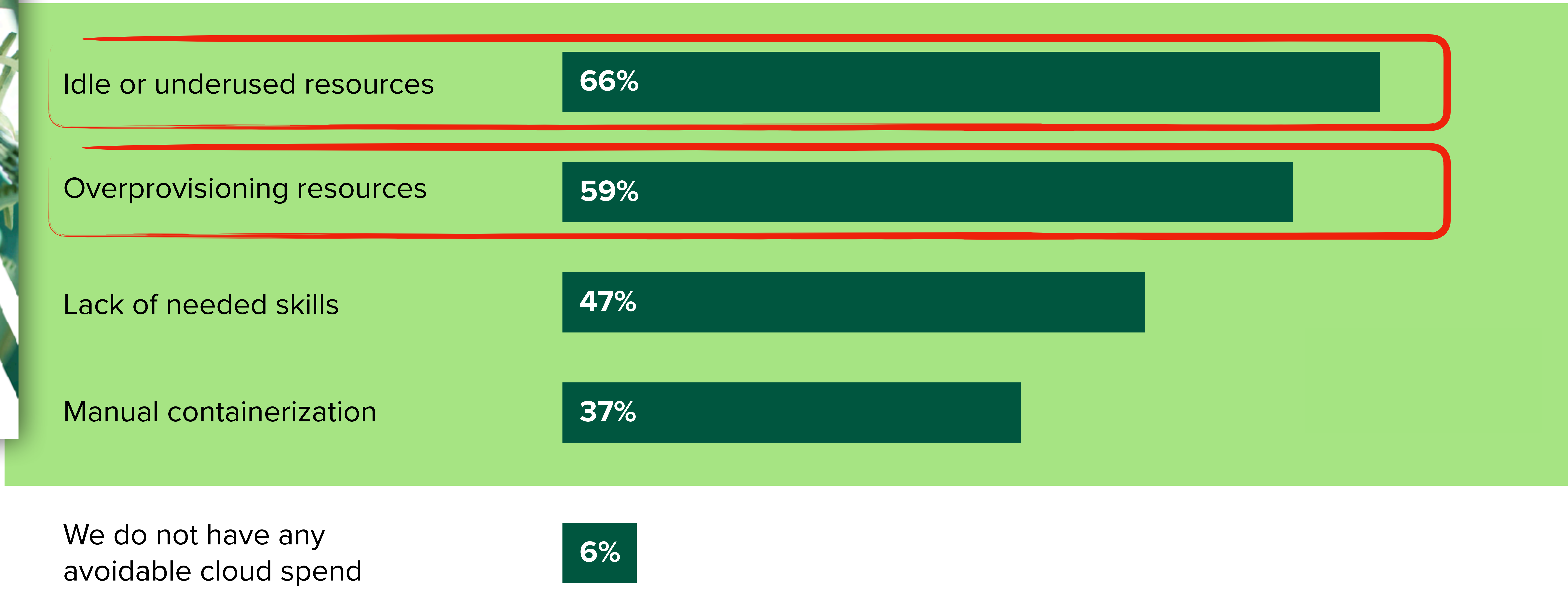
From *A Serverless Vision for Cloud Computing*
by Prof. Ana Klimovic

	Utilized Resources	Unreserved Resources	Idle Resources
	Resources in use	Resources ready to be assigned	Resources reserved but unused
Paid by users	✓	✗	✓
Re-assignable by operators	✗	✓	✗

Idle Resources



"Which of the following factors contribute to avoidable cloud spend, also known as cloud waste, at your organization?"



[1] Hashicorp-Forrester, "Unlocking Multicloud's Operational Potential," 2022. [Online]. Available: https://www.datocms-assets.com/2885/1659554932-unlocking-multiclouds-operational-potential-forrester-hashicorp.pdf

User over-provisioning

Computation patterns

Varying input data

…

→ **Idle Resources** →

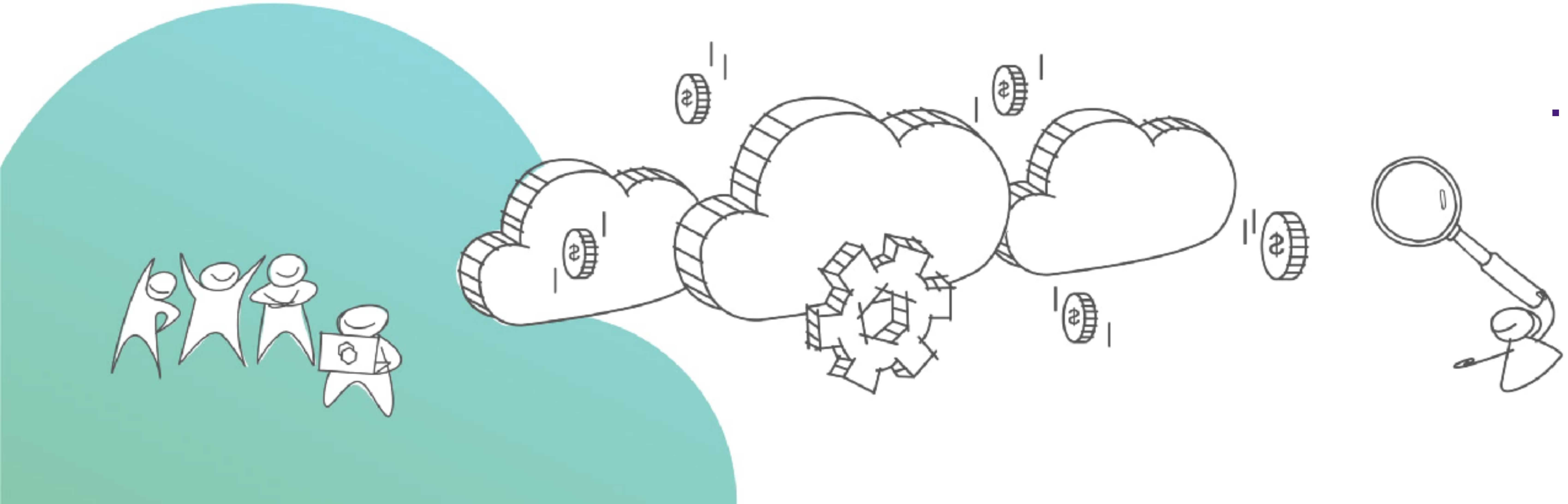
Low-priority VMs
SmartHarvest [EuroSys 21]

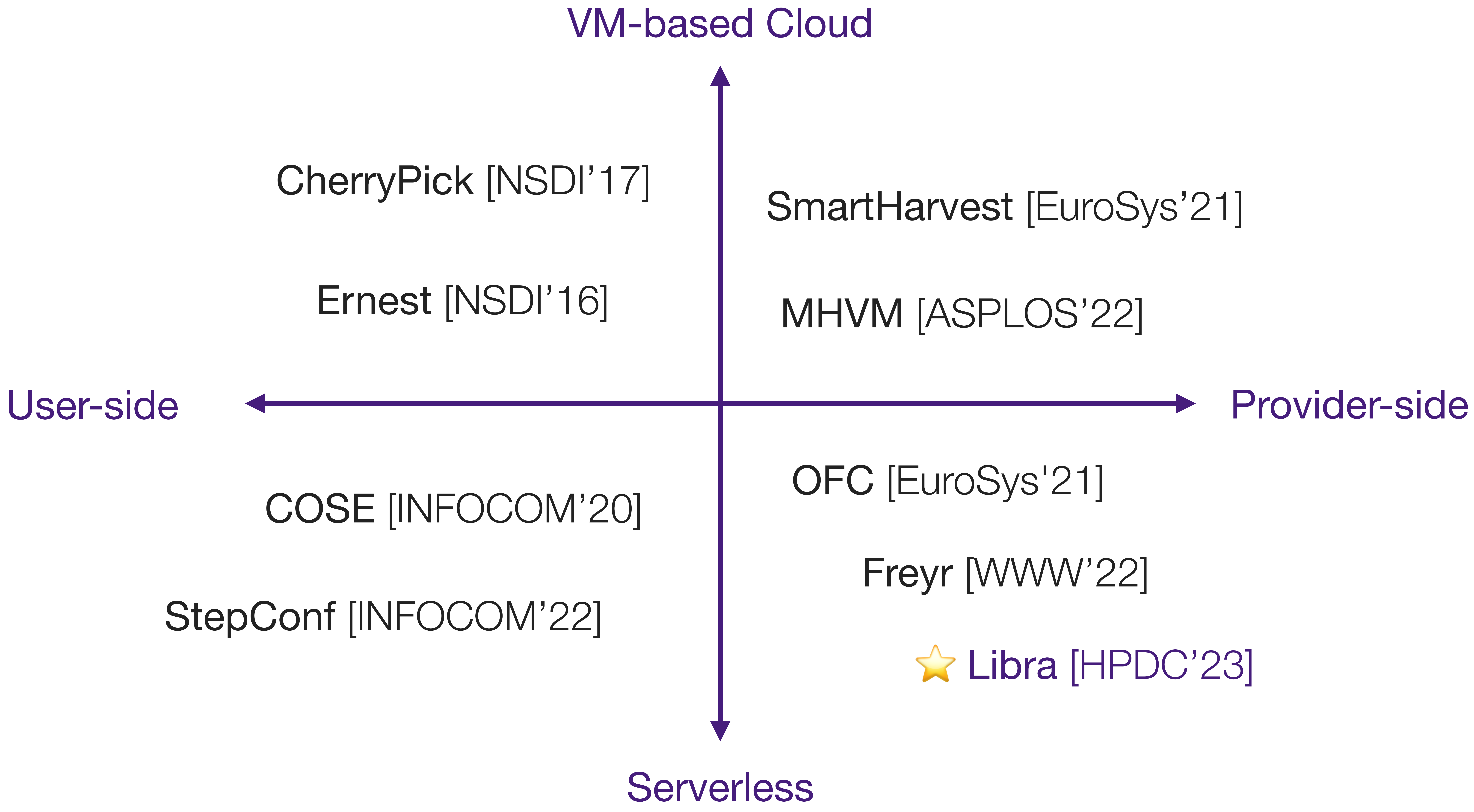
Memory harvesting VMs
MHVM [ASPLOS 22]

Harvest gSSDs
BlockFlex [OSDI 22]

Function acceleration
Freyr [WWW 22]

…




VM-based Cloud
CherryPick [NSDI’17]
SmartHarvest [EuroSys’21]
Ernest [NSDI’16]
MHVM [ASPLOS’22]
User-side
Provider-side
OFC [EuroSys'21]
COSE [INFOCOM’20]
Freyr [WWW’22]
StepConf [INFOCOM’22]
⭐ Libra [HPDC’23]
Serverless

Harvesting idle resources timely and safely?

Harvesting for VM-based cloud / HPC

Long-running

Always-on

Workload-agnostic

Serverless computing

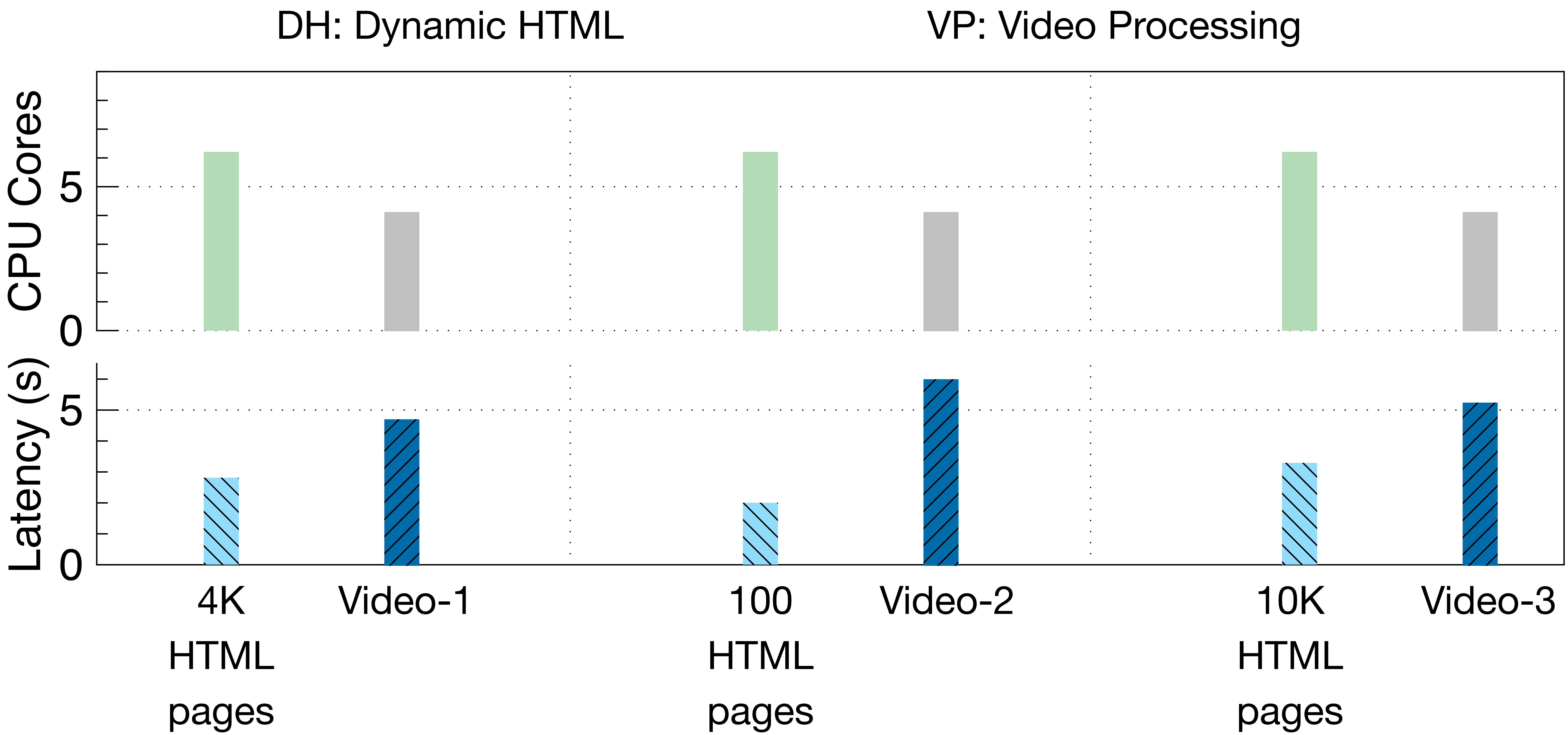
Short-living

Timeliness

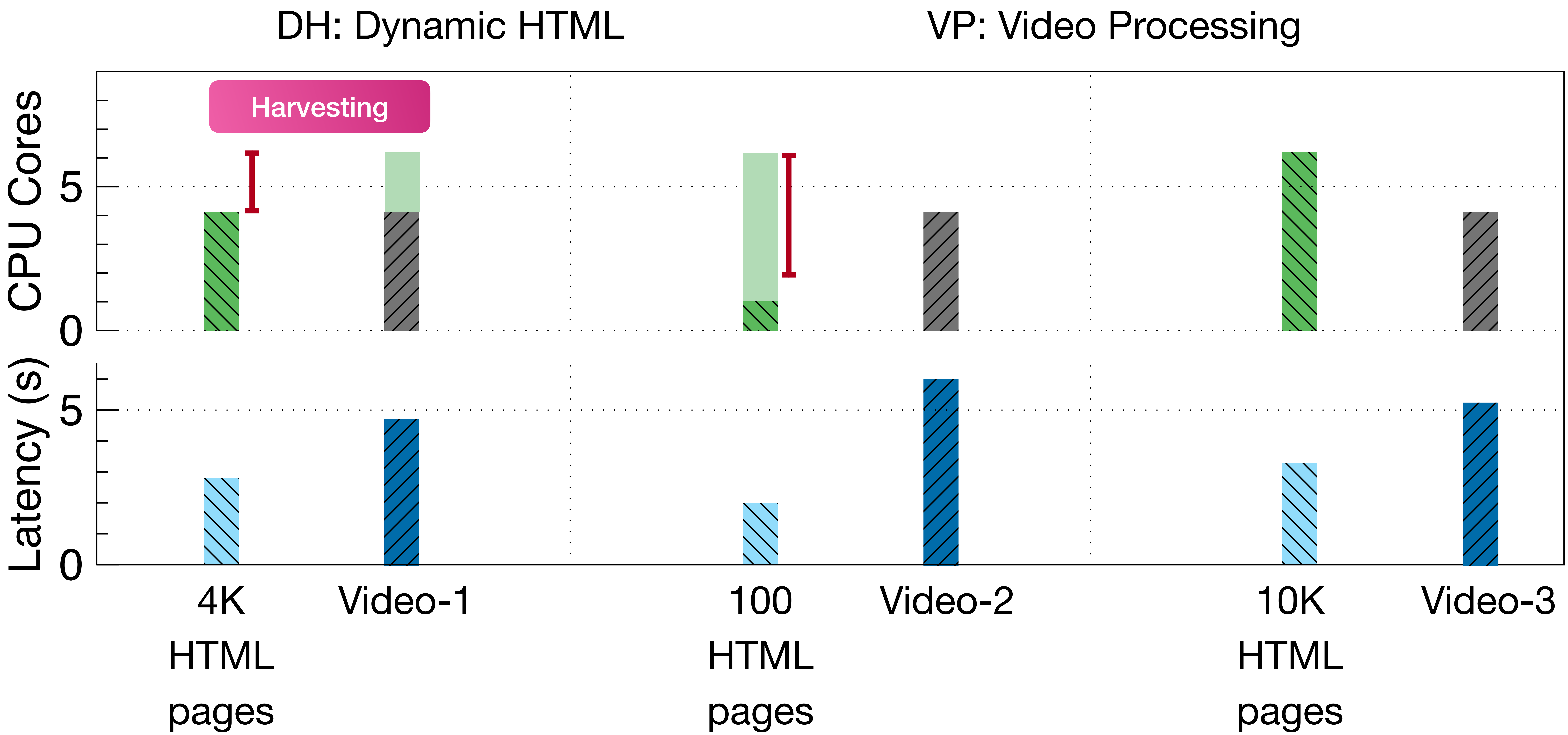
Varying input sizes

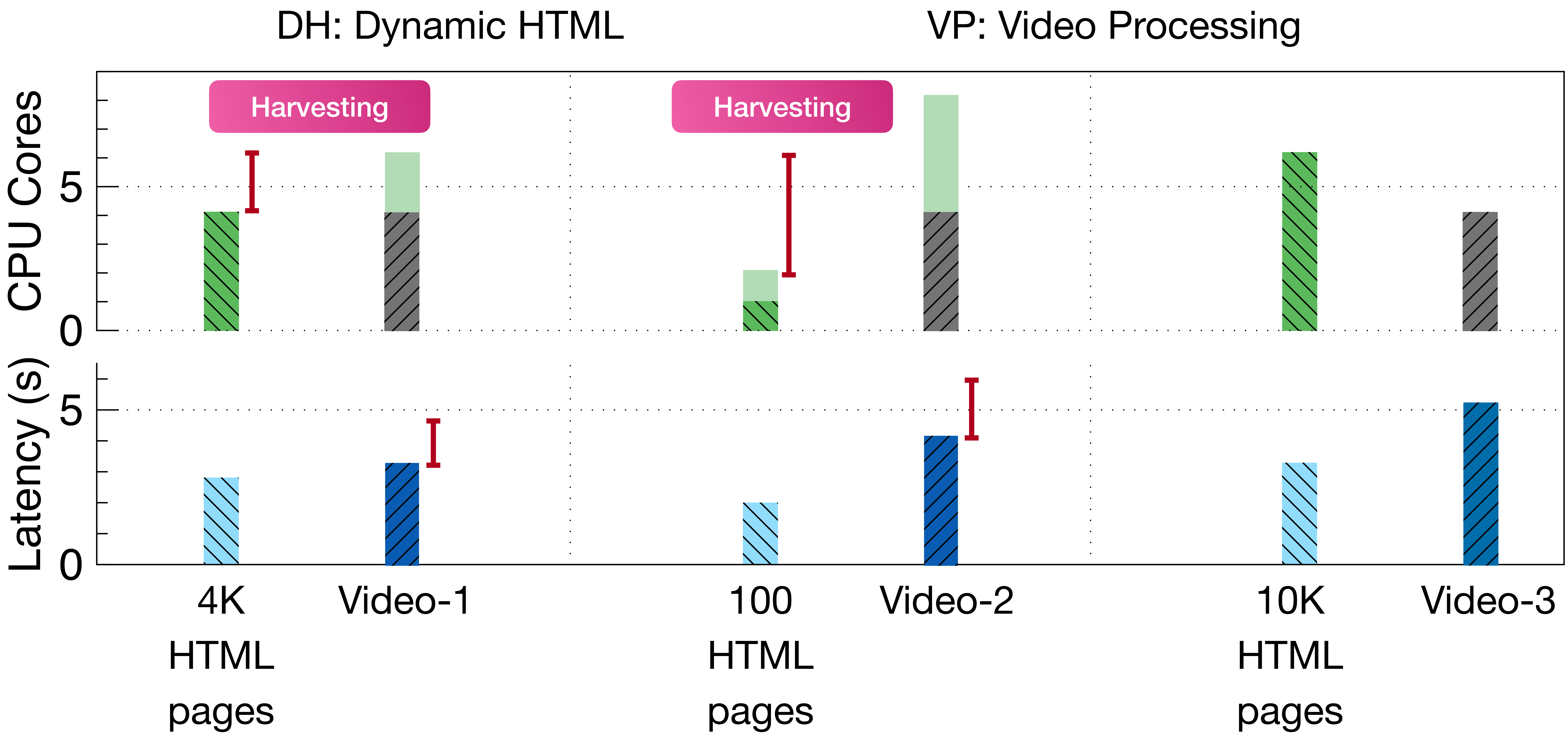
Limitations of Existing Work

- Provider-side works (OFC, Freyr)
    - Hard to generalize to *varying input data*
    - Ignorance of *resource timeliness*
- User-side works (COSE, StepConf)
    - Static configuration cannot satisfy dynamic resource demands



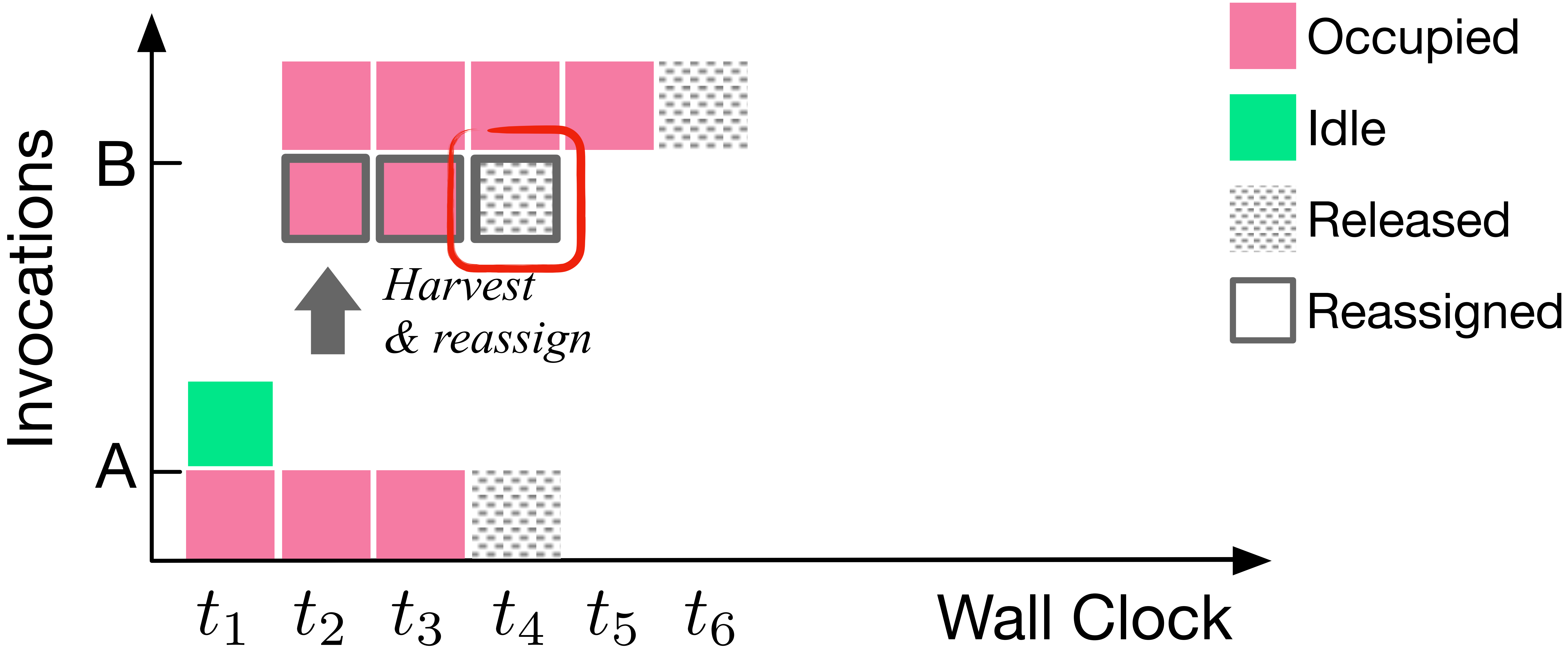
Input data sizes | Input data content → Varying latency → Timeliness


DH: Dynamic HTML
VP: Video Processing
Harvesting
CPU Cores
Latency (s)
0
5
0
5
4K HTML pages
Video-1
100 HTML pages
Video-2
10K HTML pages
Video-3



Harvesting idle resources to accelerate functions

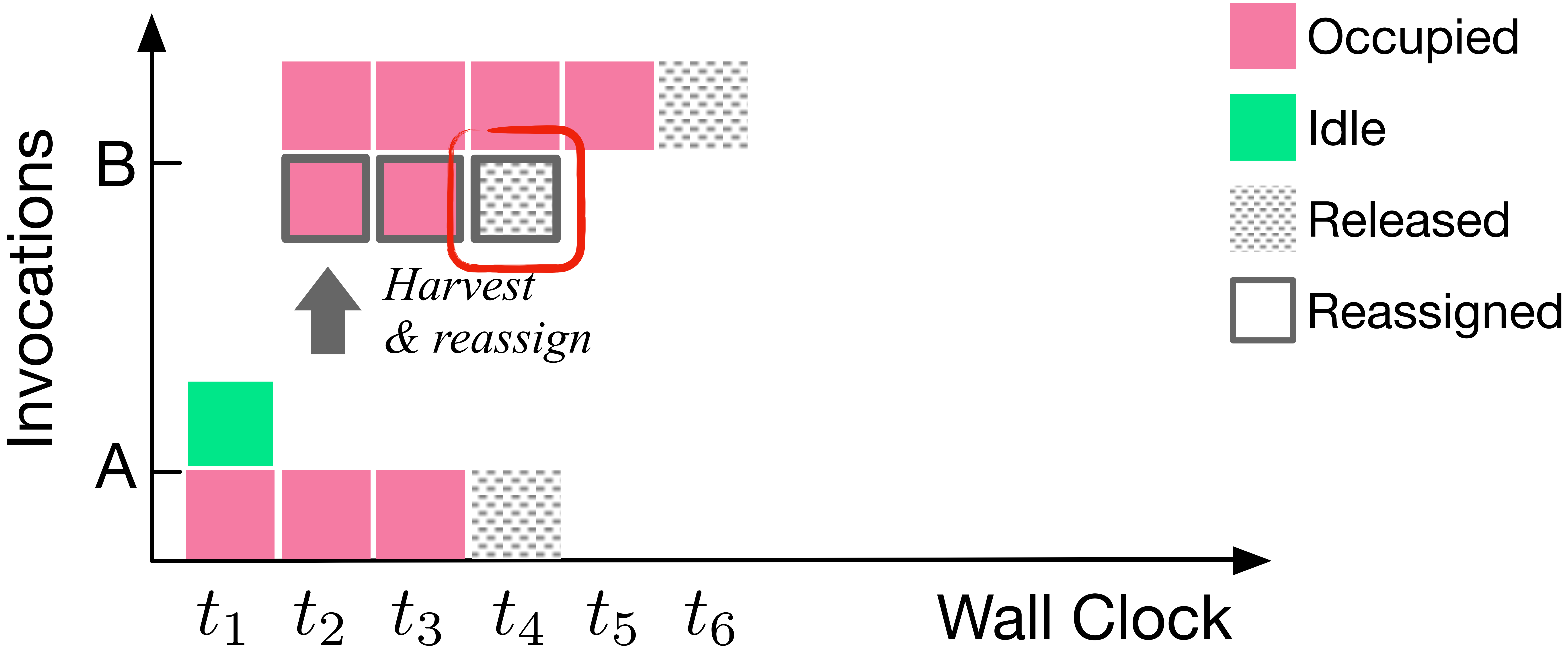
Timeliness of Resources



Invocation A: Over-provisioned, t1—t3

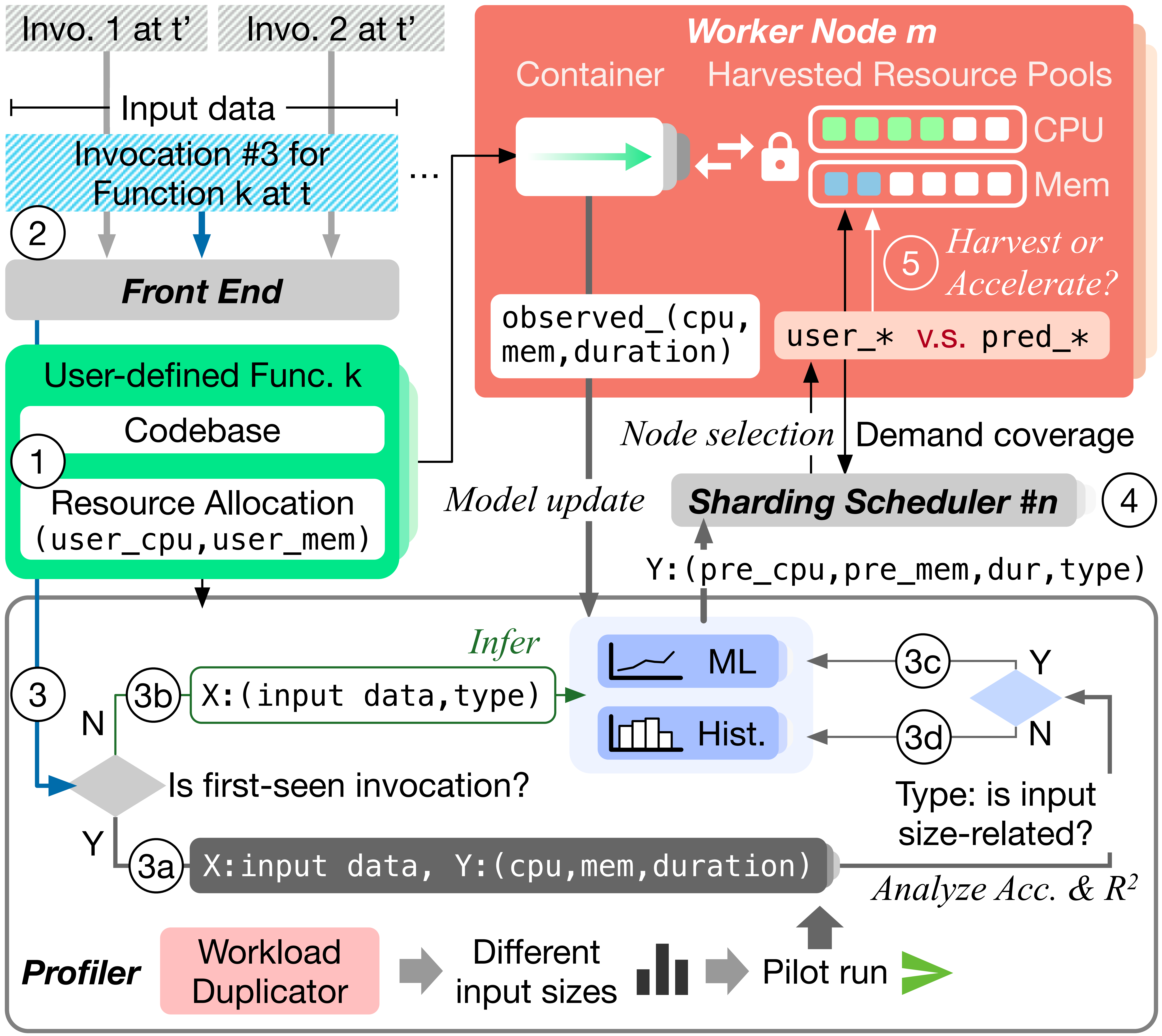
Invocation B: Under-provisioned, t2—t5

Timeliness of Resources



Invocation A: Over-provisioned, t1—t3

Invocation B: Under-provisioned, t2—t5



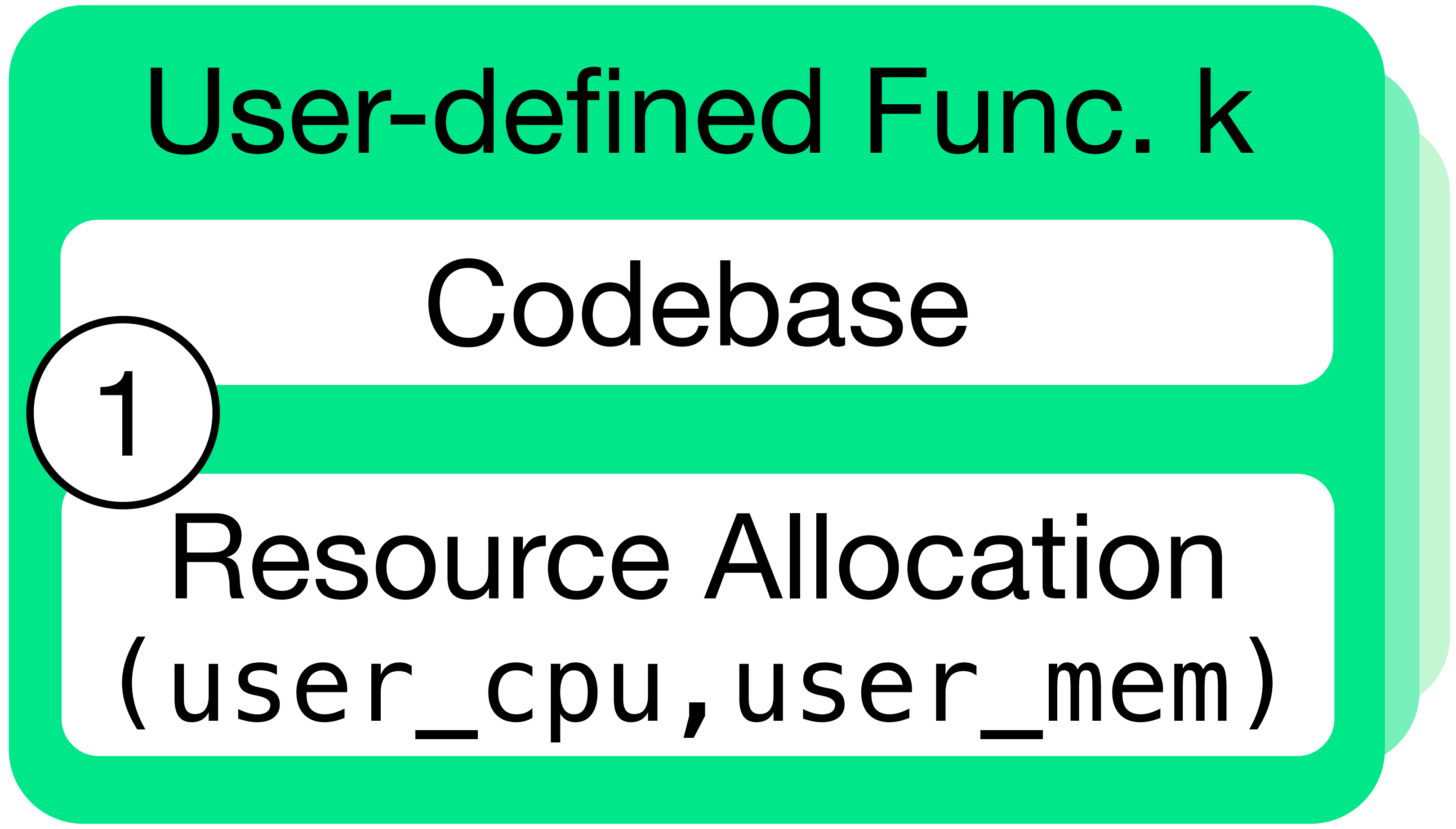
Libra Overview

Profiling

Scheduling

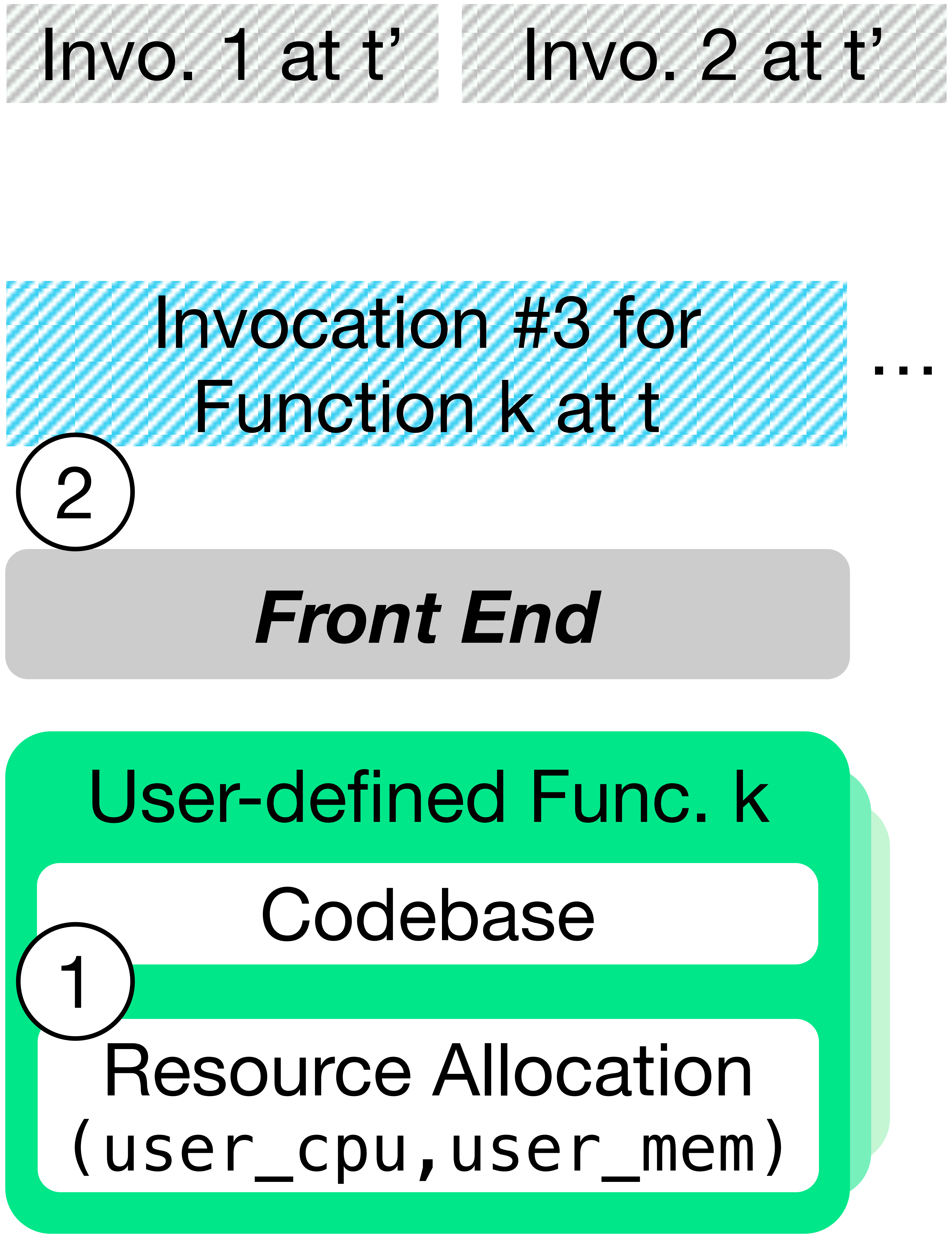
Harvesting & Acceleration

Safeguarding

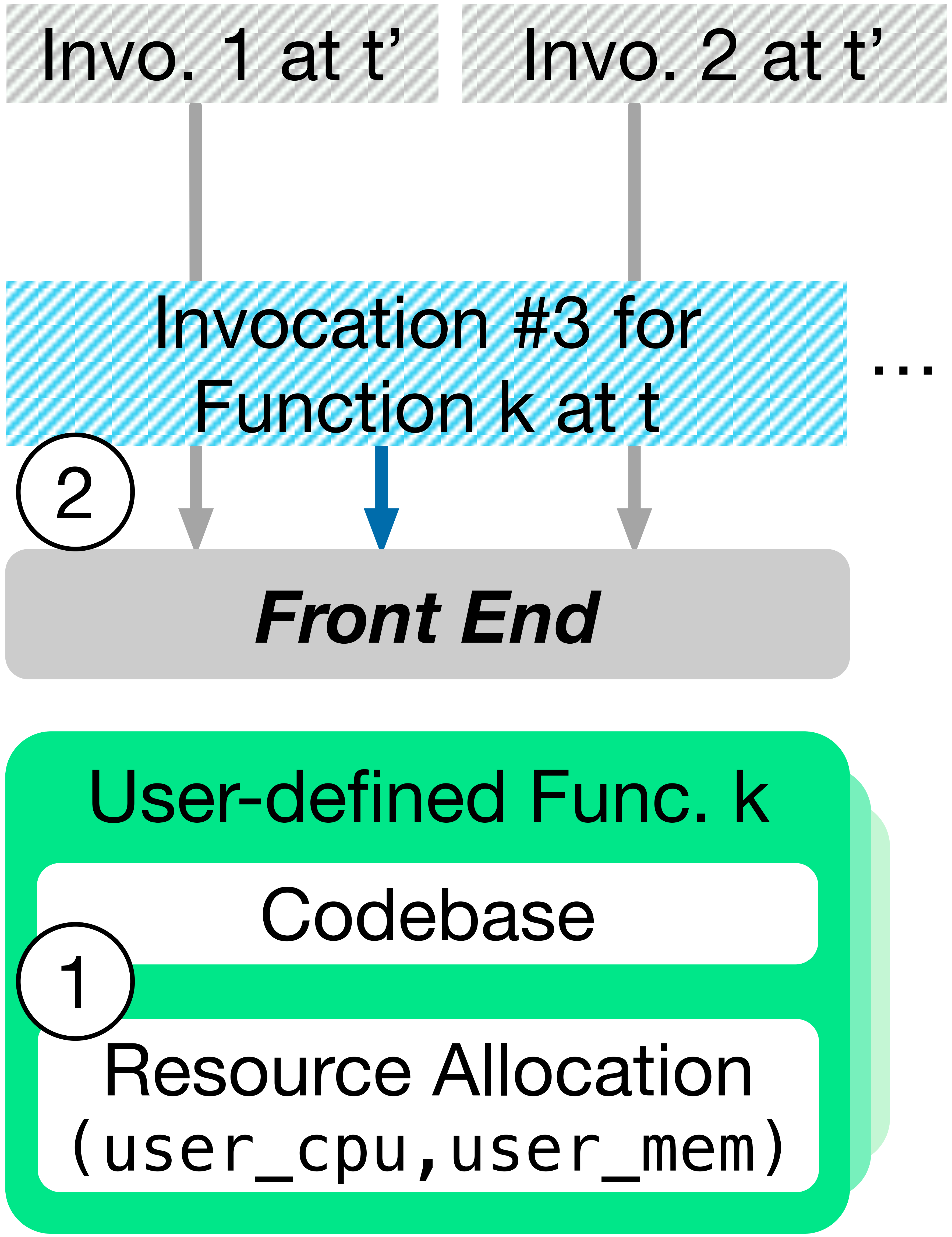


Step 1: Function Deployment

- Deploy code
- User-defined resource allocations

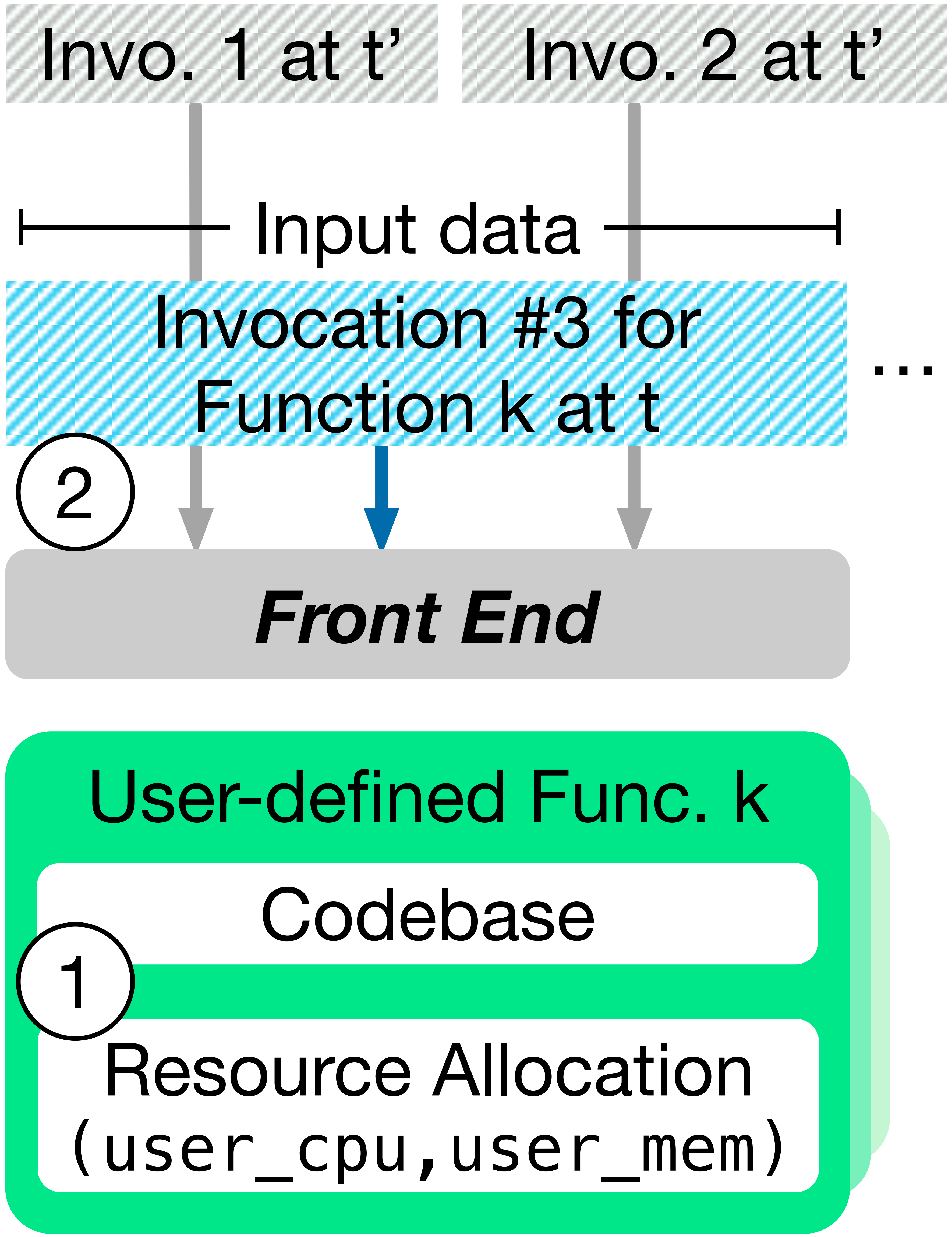


Step 2: Function Invocation



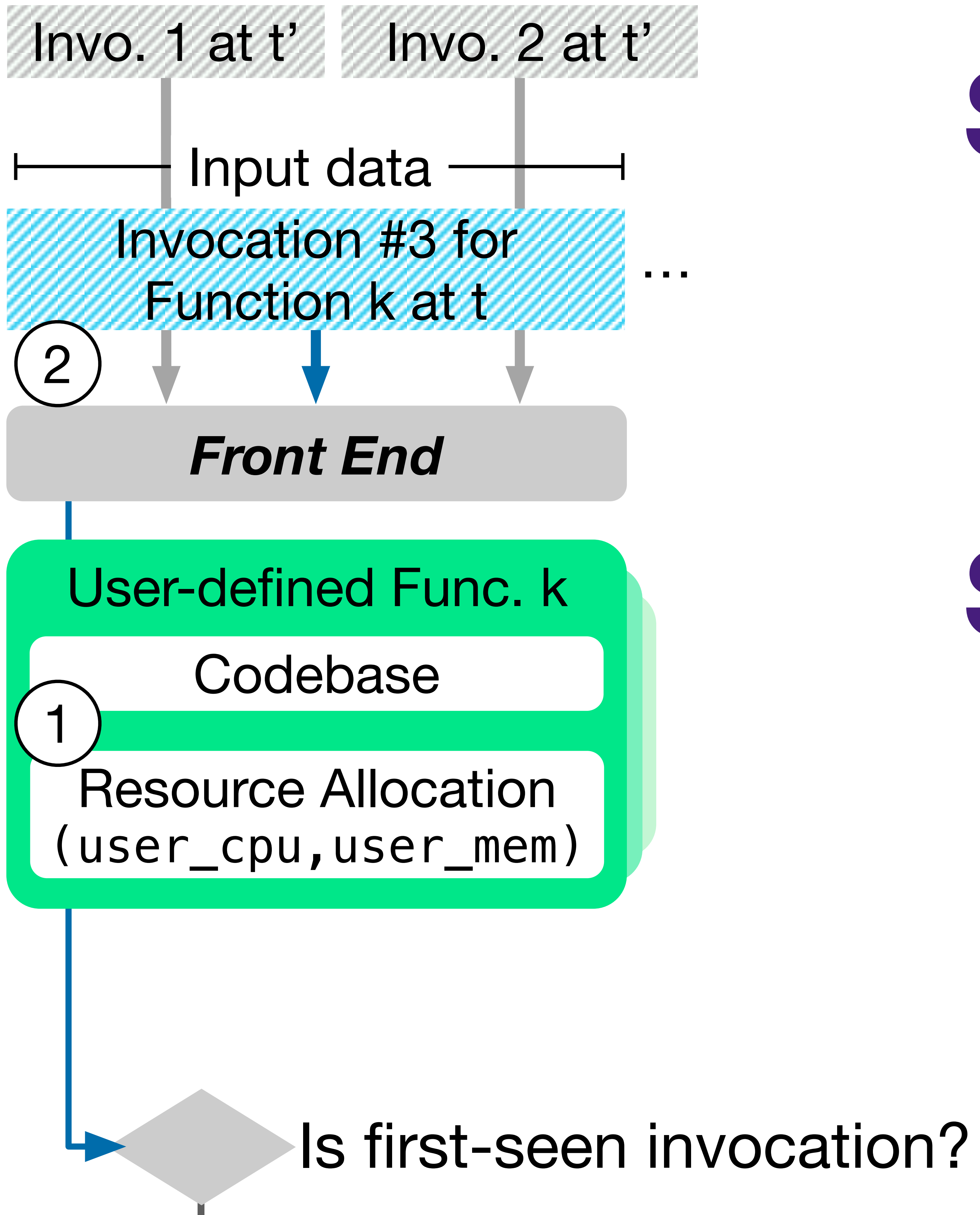
Step 2: Function Invocation

- Function requests arrive



Step 2: Function Invocation

- Function requests arrive
- Input data of varying sizes

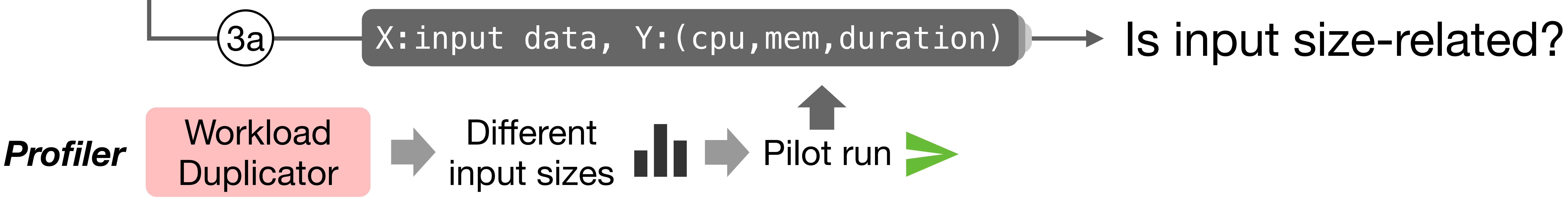


Step 3: Profiling

- Check if current invocation is first-seen
- Profile the function upon its first invocation

Step 3a: Workload Duplicator

- Duplicate the input data to different sizes
- Capture the relationship between input size and resource utilization/execution time

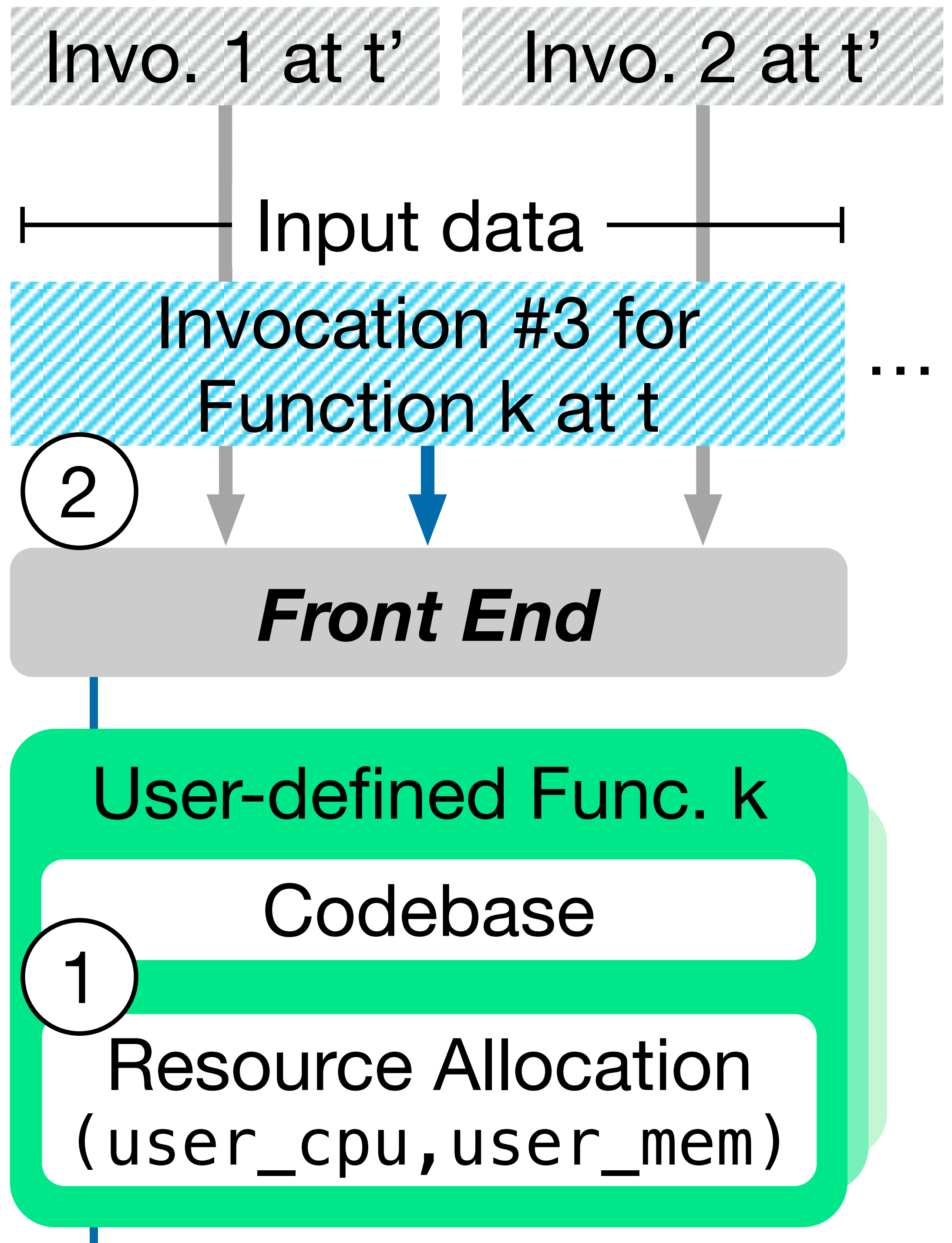


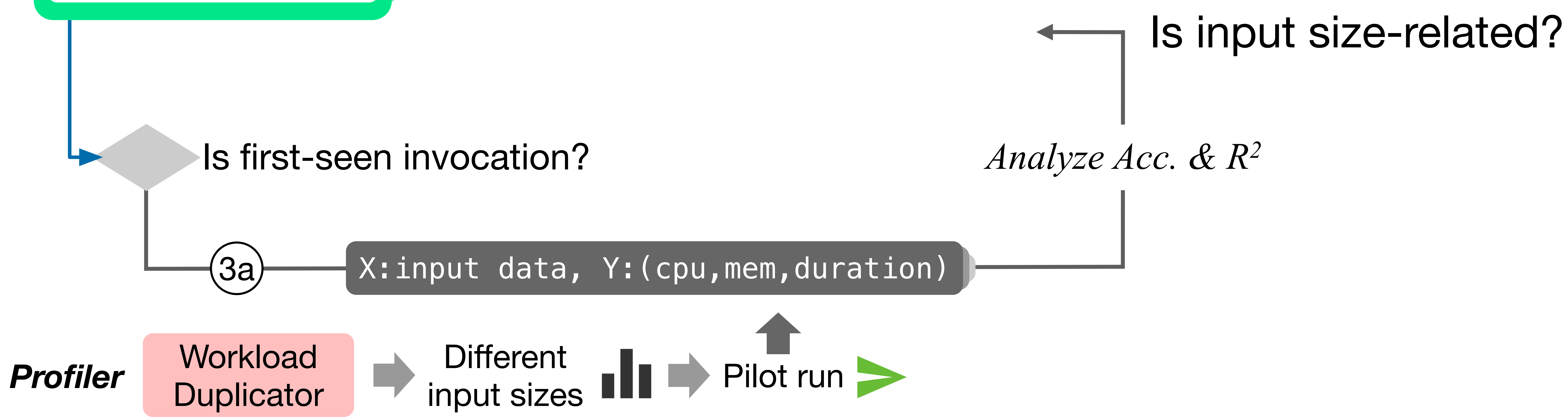
Step 3c:

- Yes, train ML models (RF) predicting future invocations.

Step 3d:

- Or apply Histogram models for conservative estimations.



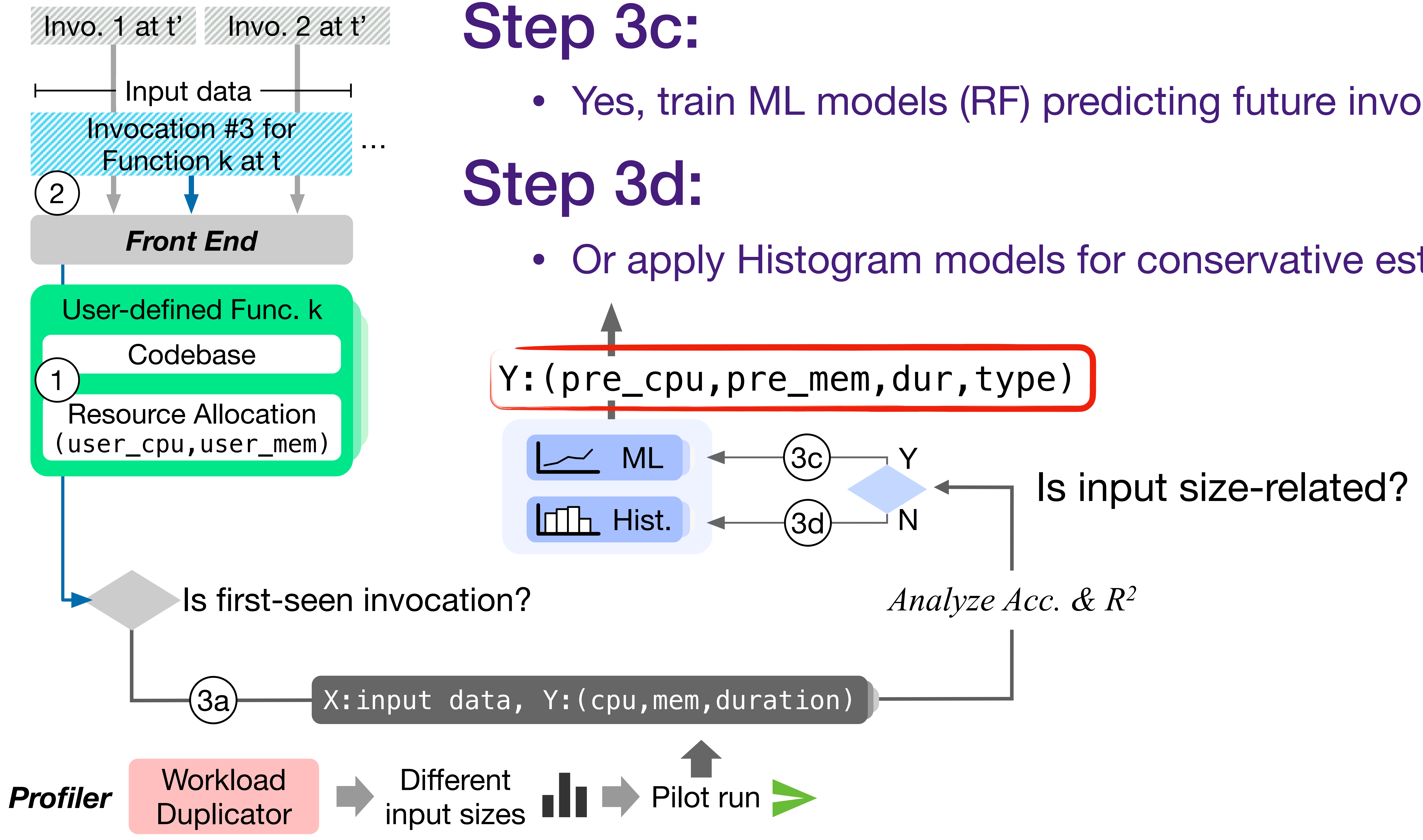


Step 3c:

- Yes, train ML models (RF) predicting future invocations.

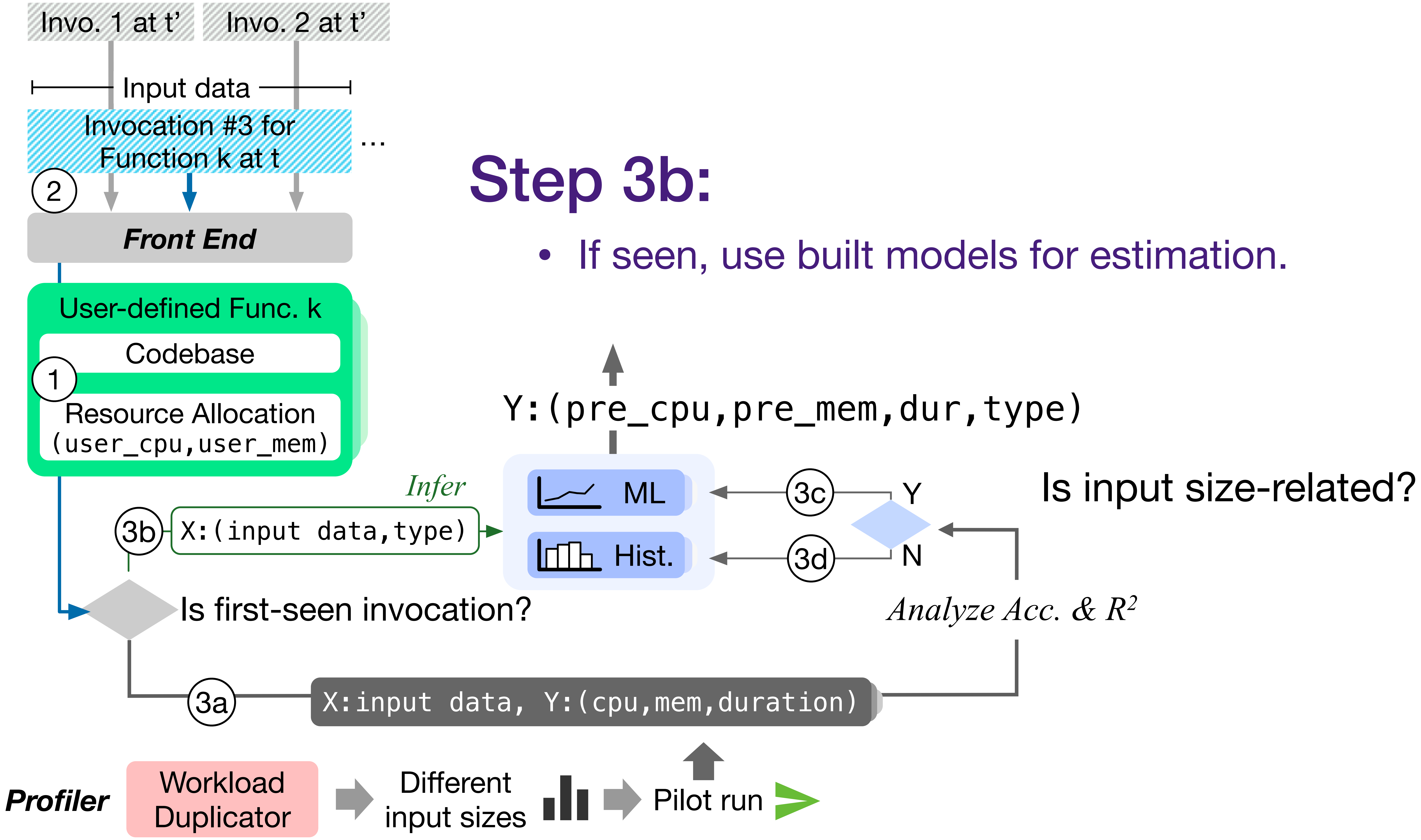
Step 3d:

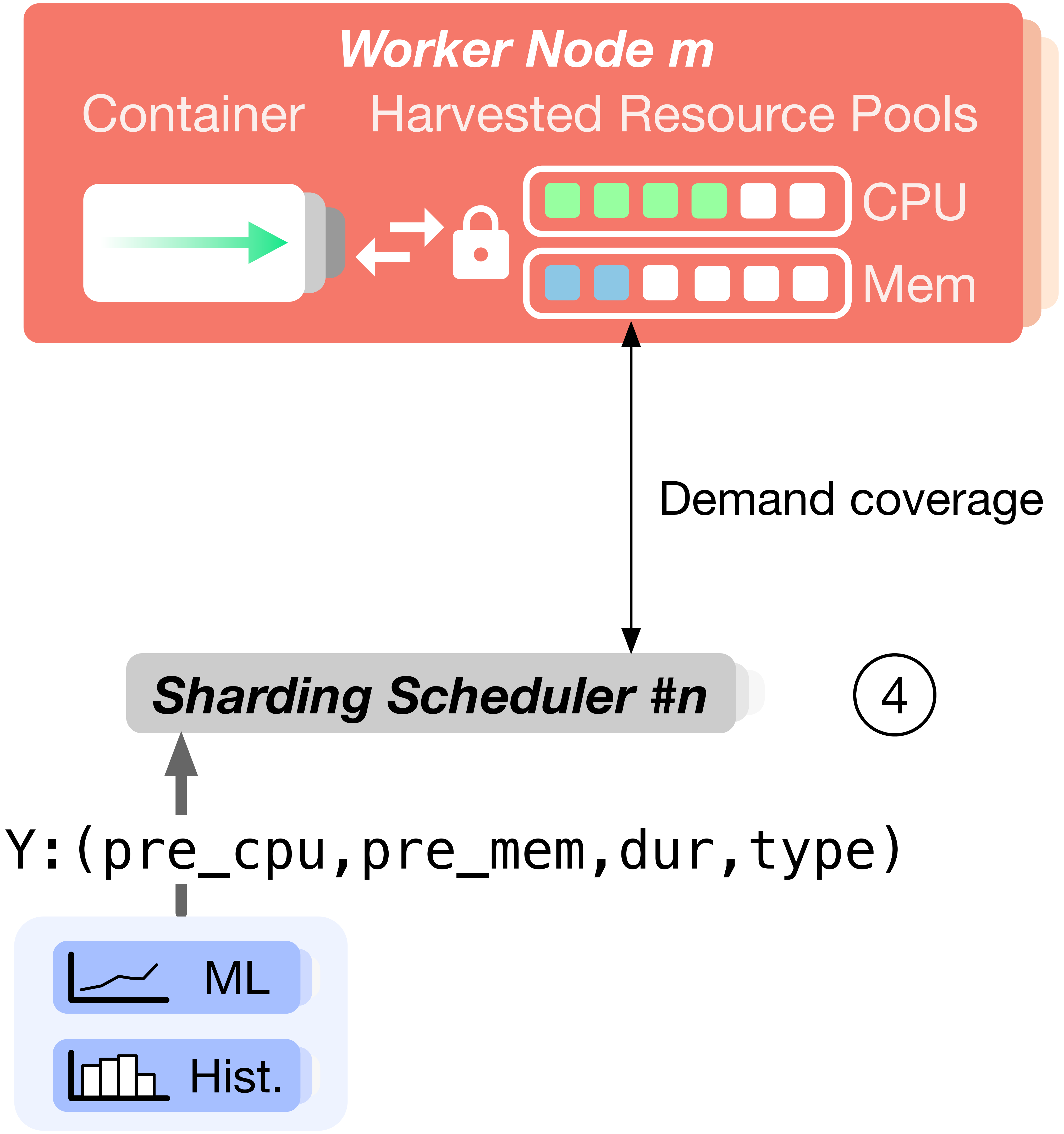
- Or apply Histogram models for conservative estimations.



Step 3b:

- If seen, use built models for estimation.





Step 4: Scheduling

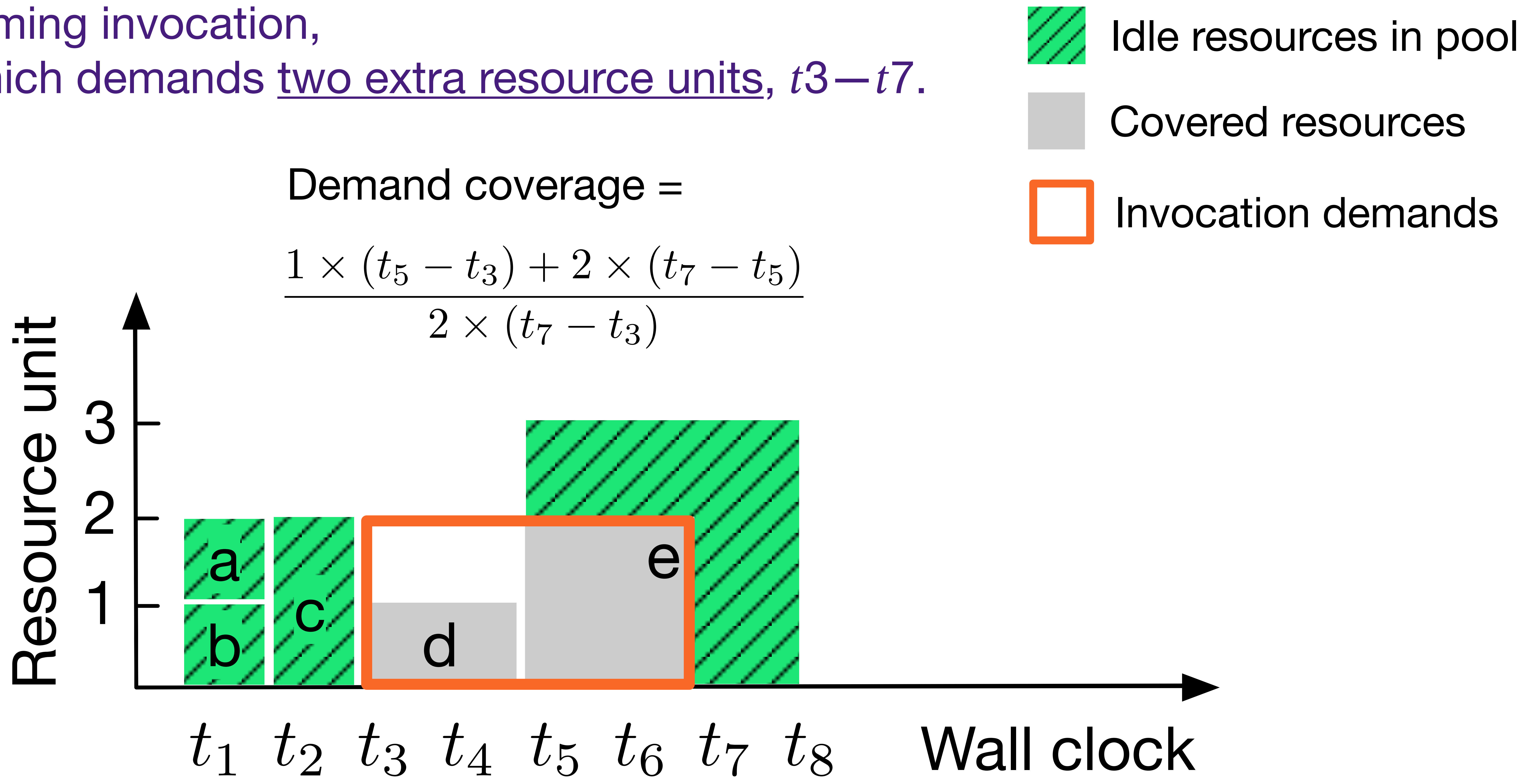
- To harvest
- Or to accelerate?
  - Calculate demand coverage

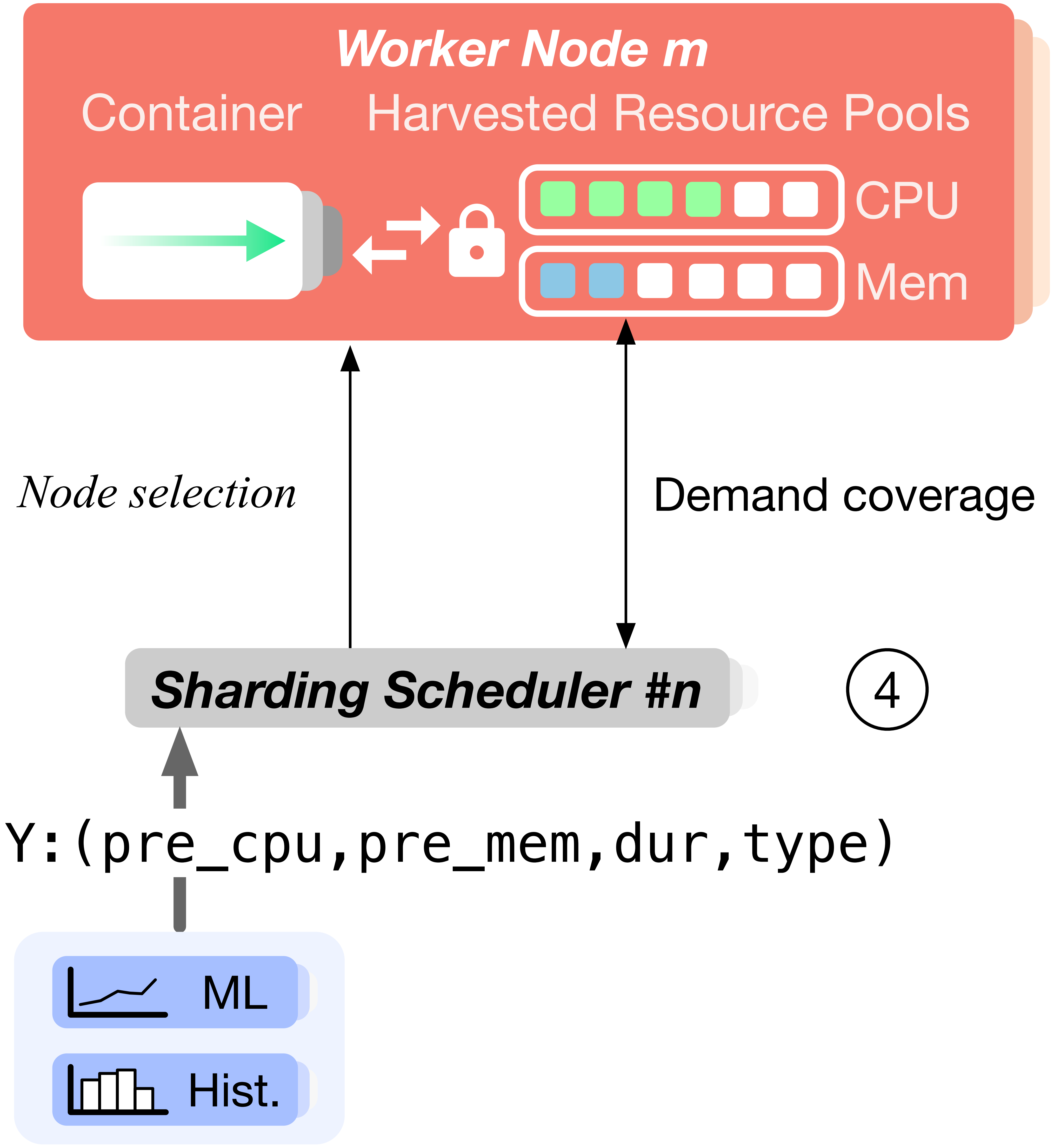
Demand Coverage

An incoming invocation,
which demands two extra resource units, $t3—t7$.

Demand coverage =

$$\frac{1 \times (t_5 - t_3) + 2 \times (t_7 - t_5)}{2 \times (t_7 - t_3)}$$





Step 4: Scheduling

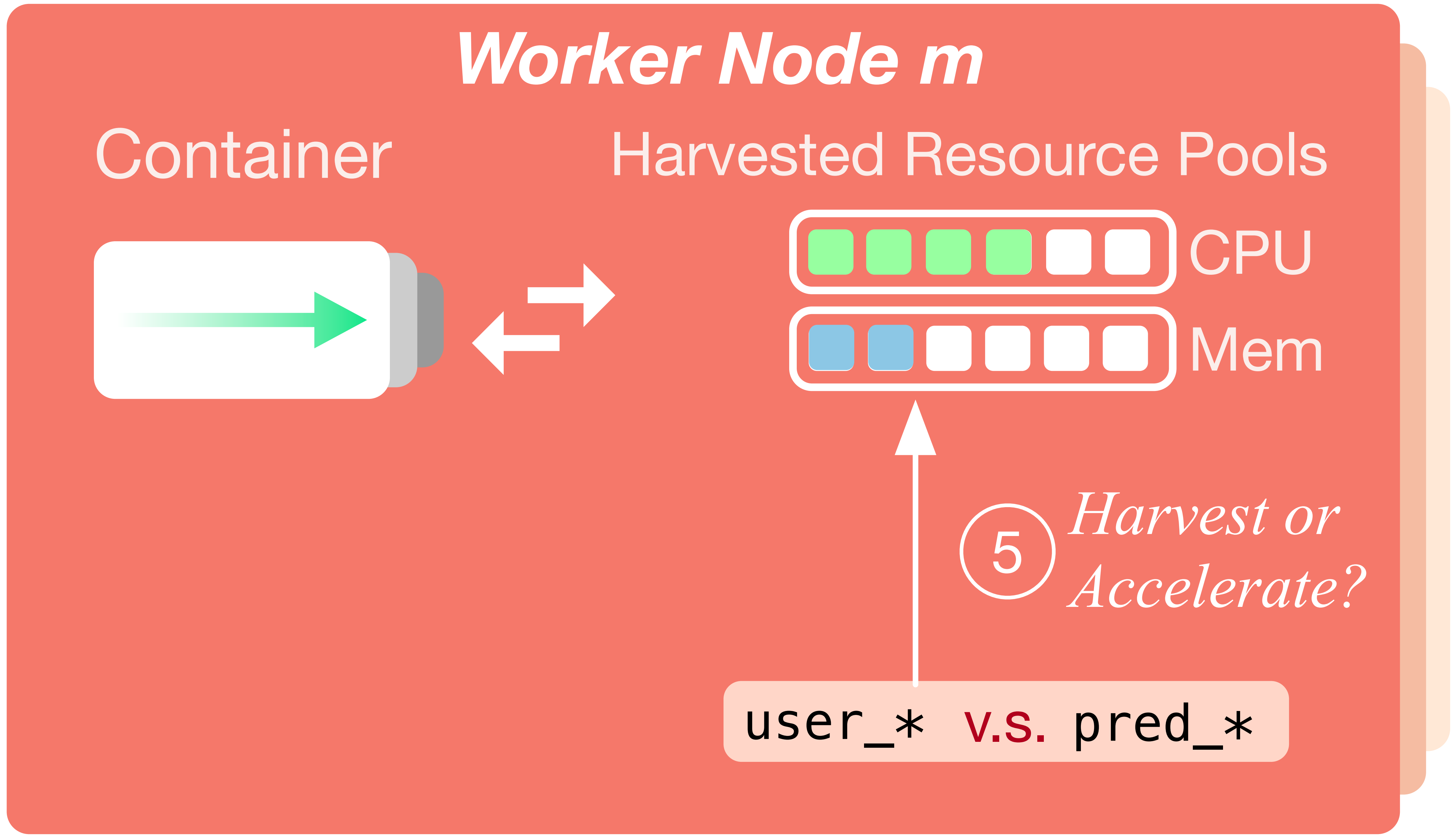
- Calculate demand coverage in realtime
- Select the node with the highest score
- Same score, then consider locality

Coverage score =

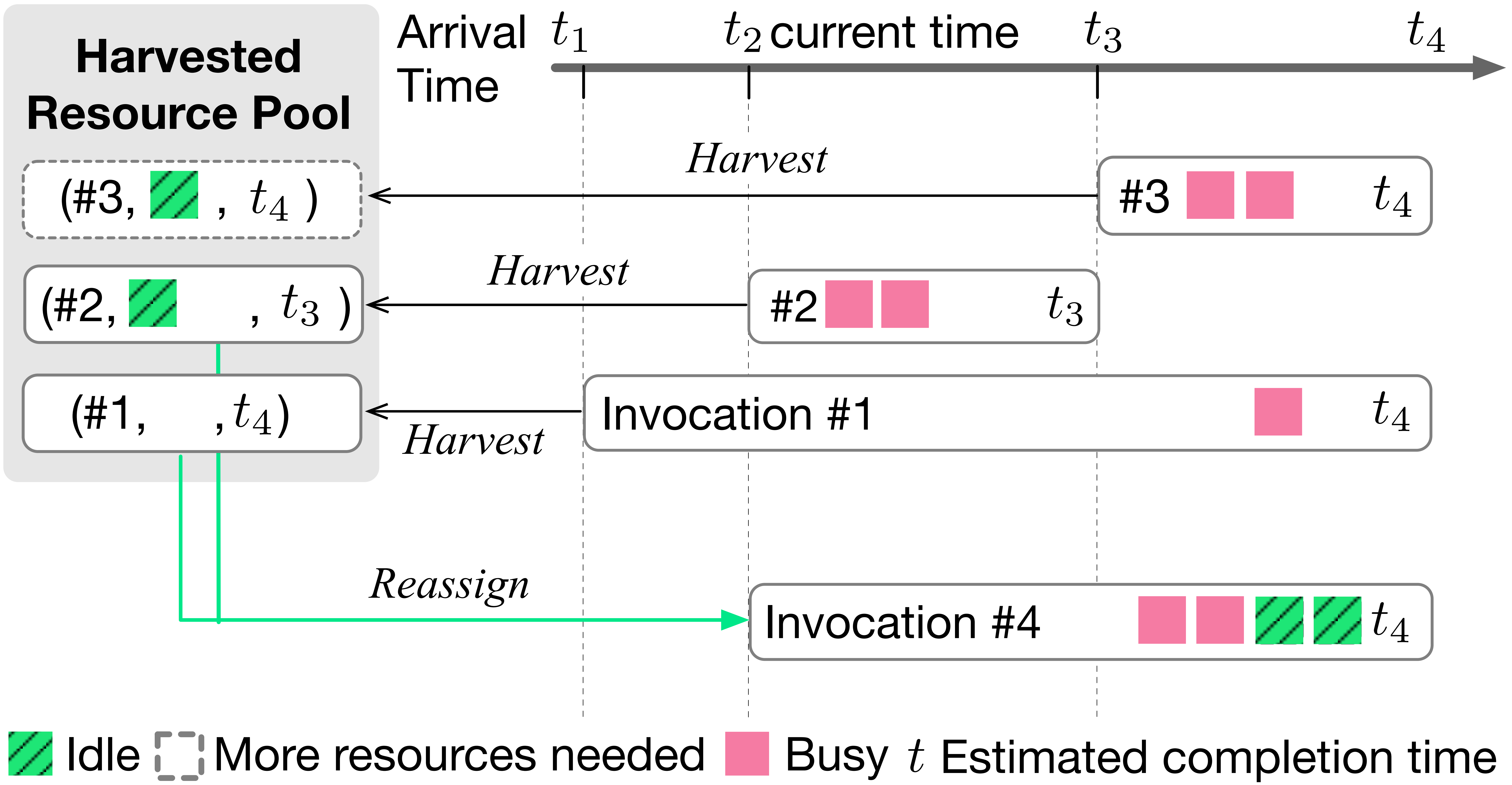
$$\alpha \times D_{cpu} + (1 - \alpha) \times D_{memory}$$

Demand coverage of CPU

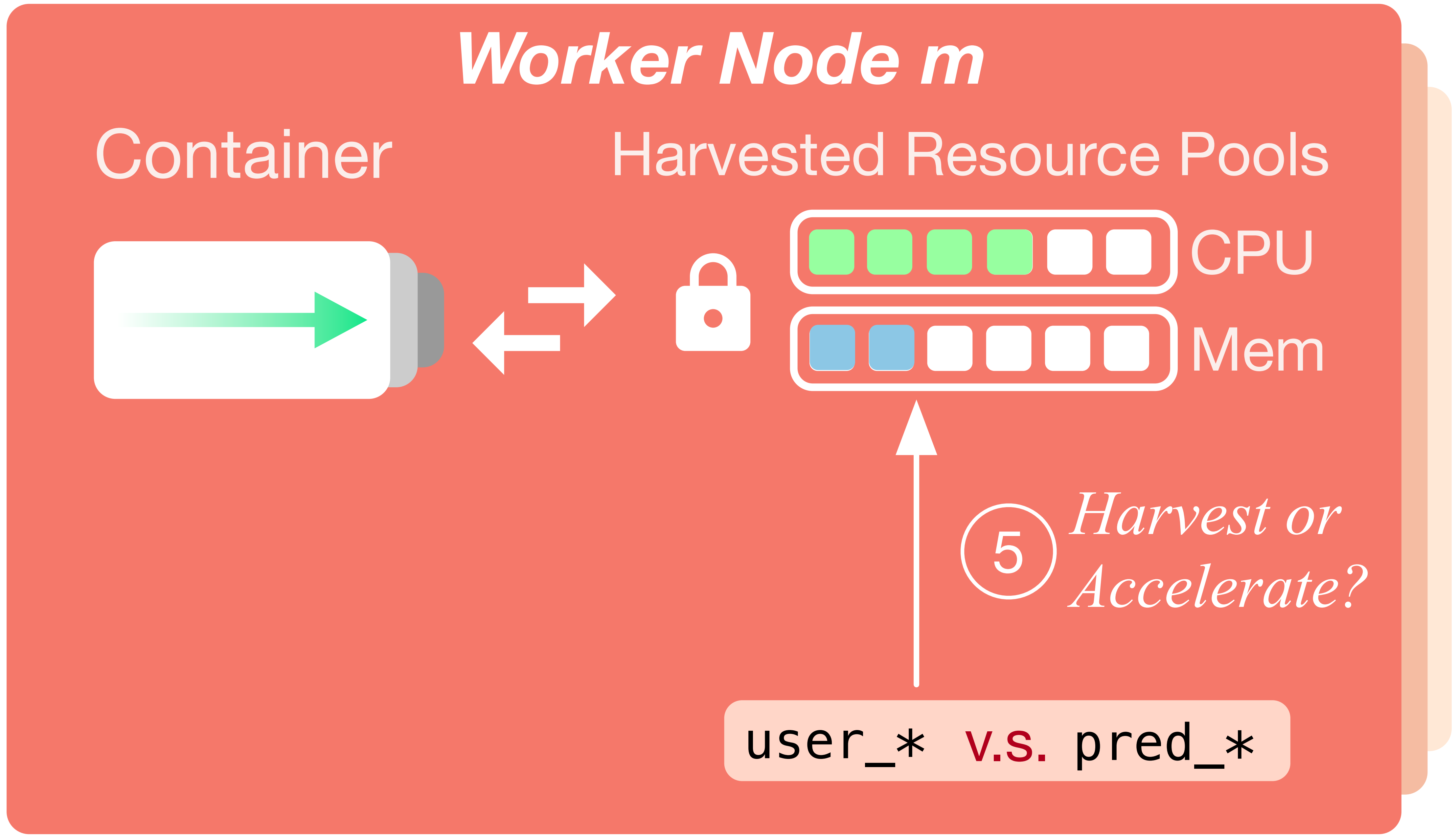
Demand coverage of Memory



Step 5: Harvesting & Acceleration

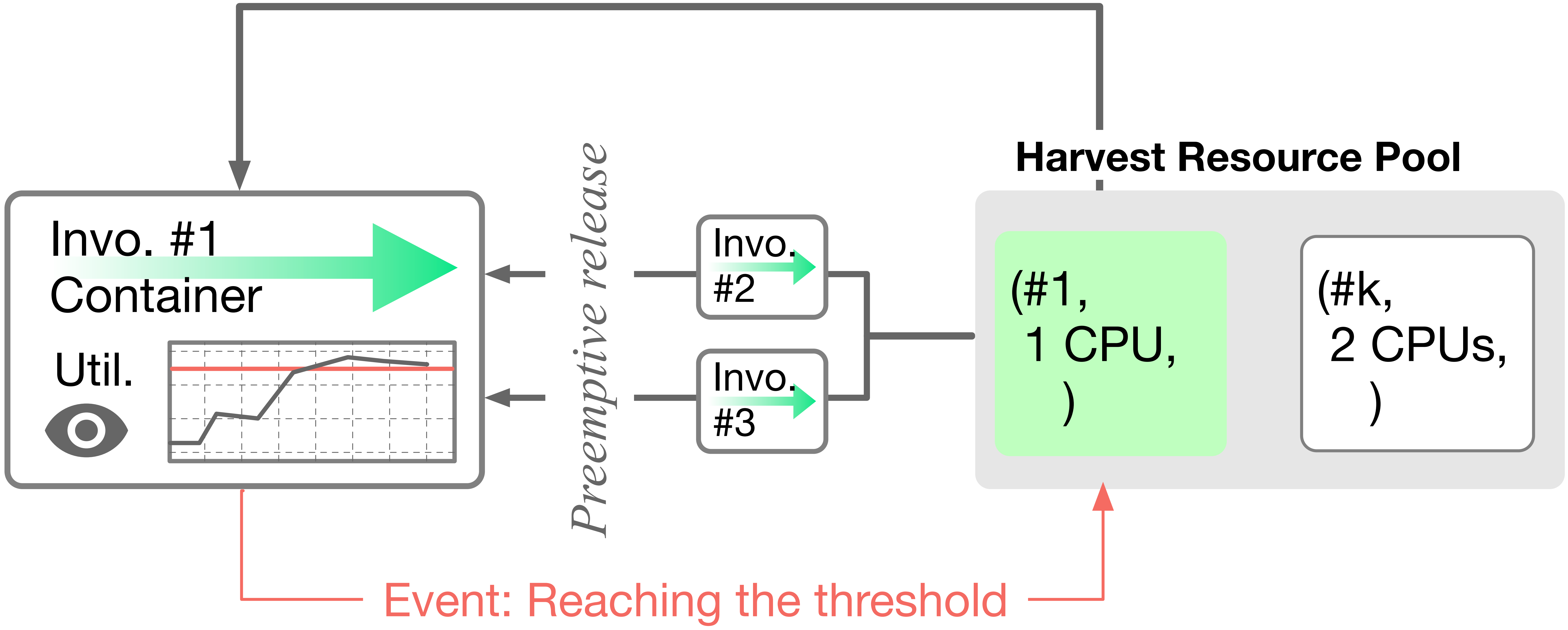


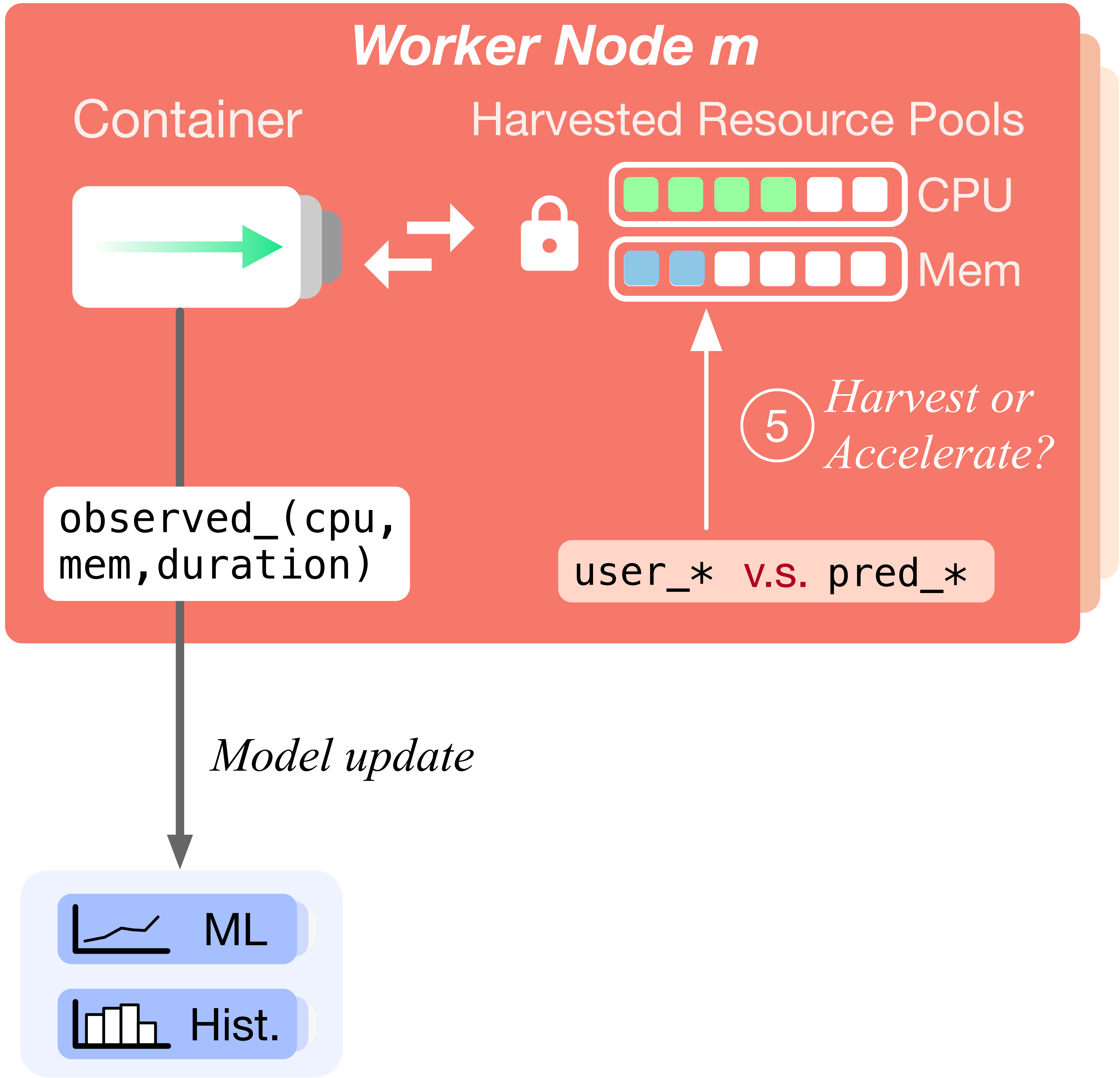
Harvested Resource Pool
Arrival Time
t_1
t_2 current time
t_3
t_4
(#3, , t_4)
(#2, , t_3)
(#1, , t_4)
Harvest
#3 t_4
Harvest
#2 t_3
Invocation #1 t_4
Harvest
Reassign
Invocation #4 t_4
Idle
More resources needed
Busy
t Estimated completion time


Worker Node m
Container
Harvested Resource Pools
CPU
Mem
5
Harvest or Accelerate?
user_* v.s. pred_*

SafeGuard

Safeguard in Realtime





Updating observations

Implementation

Libra is prototyped on top of Apache OpenWhisk using Scala

Profiler	**Scheduler**	**Harvest Pool**	**Safeguard**
Python Scikit-learn	OpenWhisk's Load Balancer	OpenWhisk's Invoker	Docker container Linux `cgroups`

Evaluation

Baselines for scheduling

OpenWhisk default
Min-Worker-Set [5]
Join-the-Shortest-Queue
Round-robin

Baselines for harvesting

OpenWhisk default
Freyr [4]

Metrics

Function response latency
Resource utilization

Benchmarks

SeBS [6]
ServerlessBench [7]
ENSURE [8]

Traces

Azure Functions traces [9]
1K+ invocation traces

[4] Yu, Hanfei, et al. “Accelerating serverless computing by harvesting idle resources." WWW 22
[5] Zhang, Yanqi, et al. "Faster and cheaper serverless computing on harvested resources." SOSP 21
[6] Copik, Marcin, et al. "Sebs: A serverless benchmark suite for function-as-a-service computing." MIDDLEWARE 21
[7] Yu, Tianyi, et al. "Characterizing serverless platforms with serverlessbench." SoCC 20
[8] Suresh, Amoghavarsha, et al. "Ensure: Efficient scheduling…” ACSOS 20
[9] Shahrad, Mohammad, et al. “Serverless in the Wild…” ATC 20

Testbed Clusters

Evaluating harvesting

3 nodes
72 Intel Xeon
E5-2670 CPU cores
72 GB memory

Evaluating scheduling

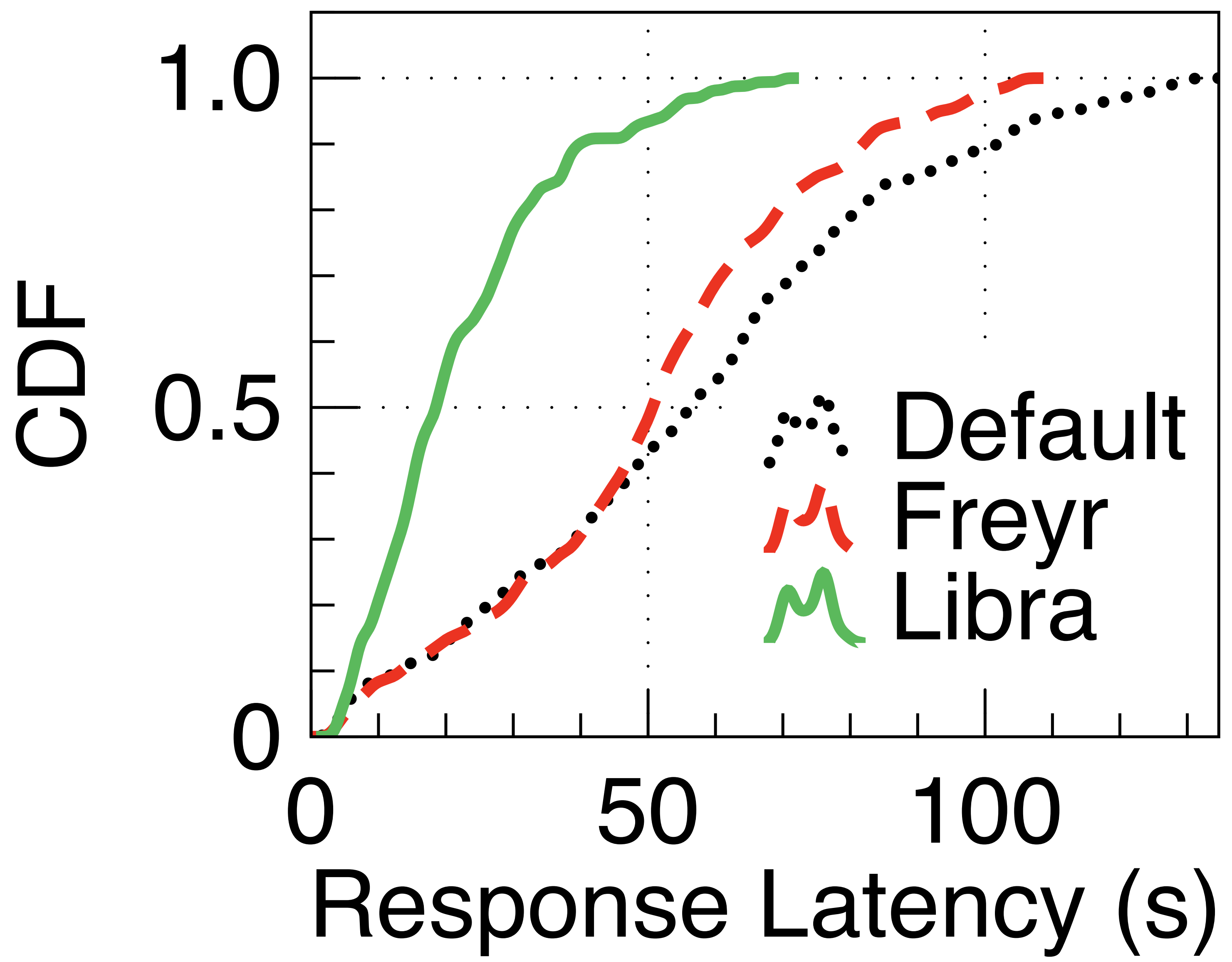
6 nodes
160 Intel Xeon
E5-2420 CPU cores
160 GB memory

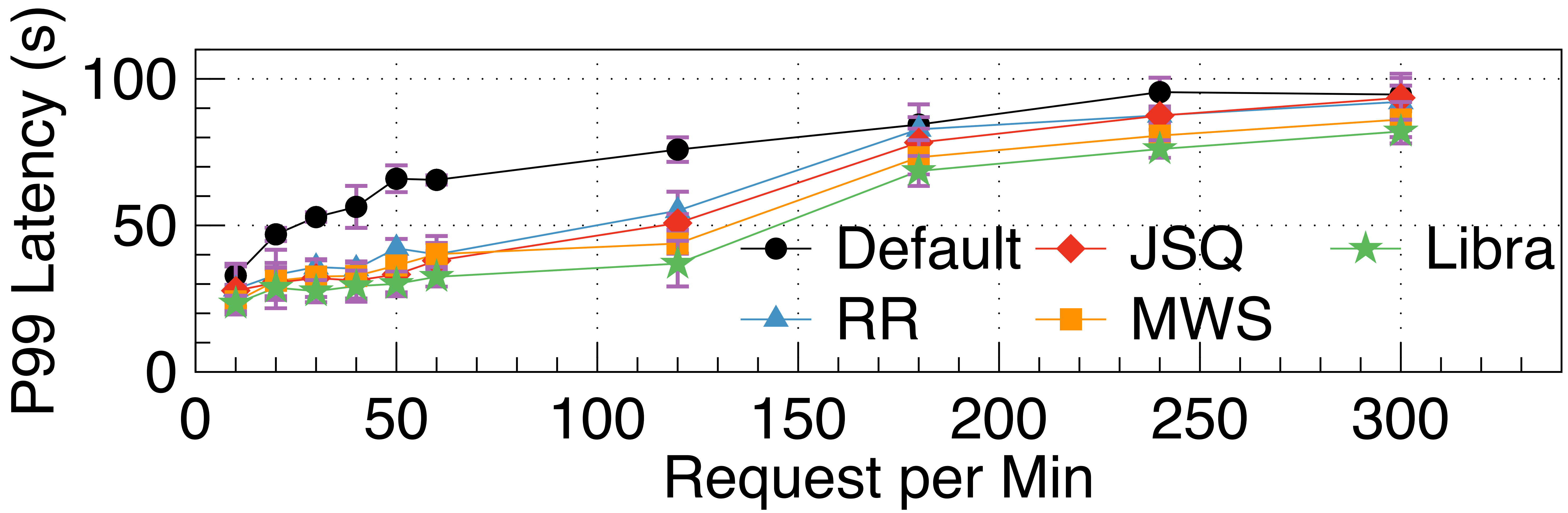
Evaluating scalability

50 Jetstream nodes
1,200 Intel Xeon
E5-2680 CPU cores
1,200 GB memory

Function Response Latency

Libra provides lowest function response latency

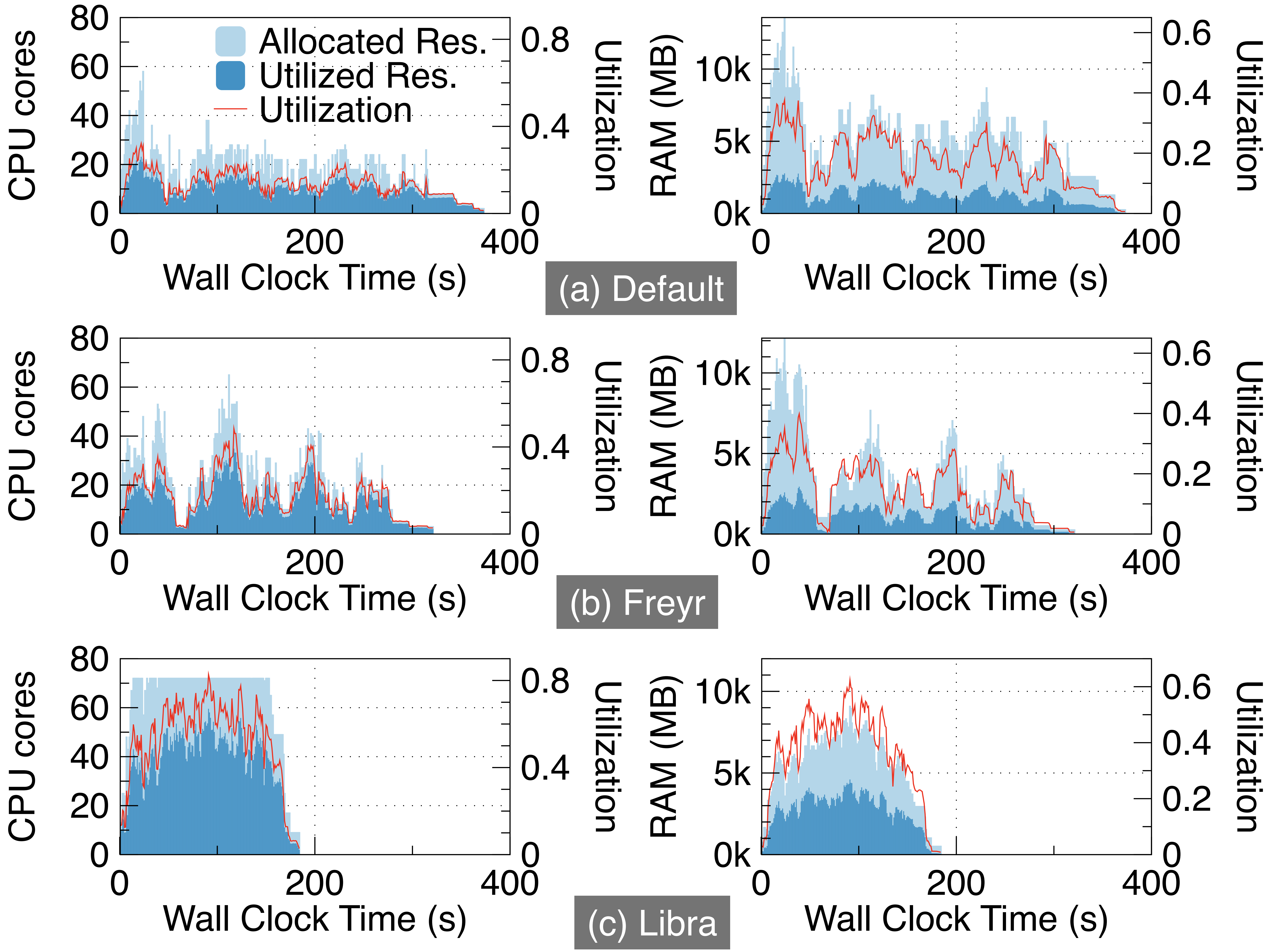




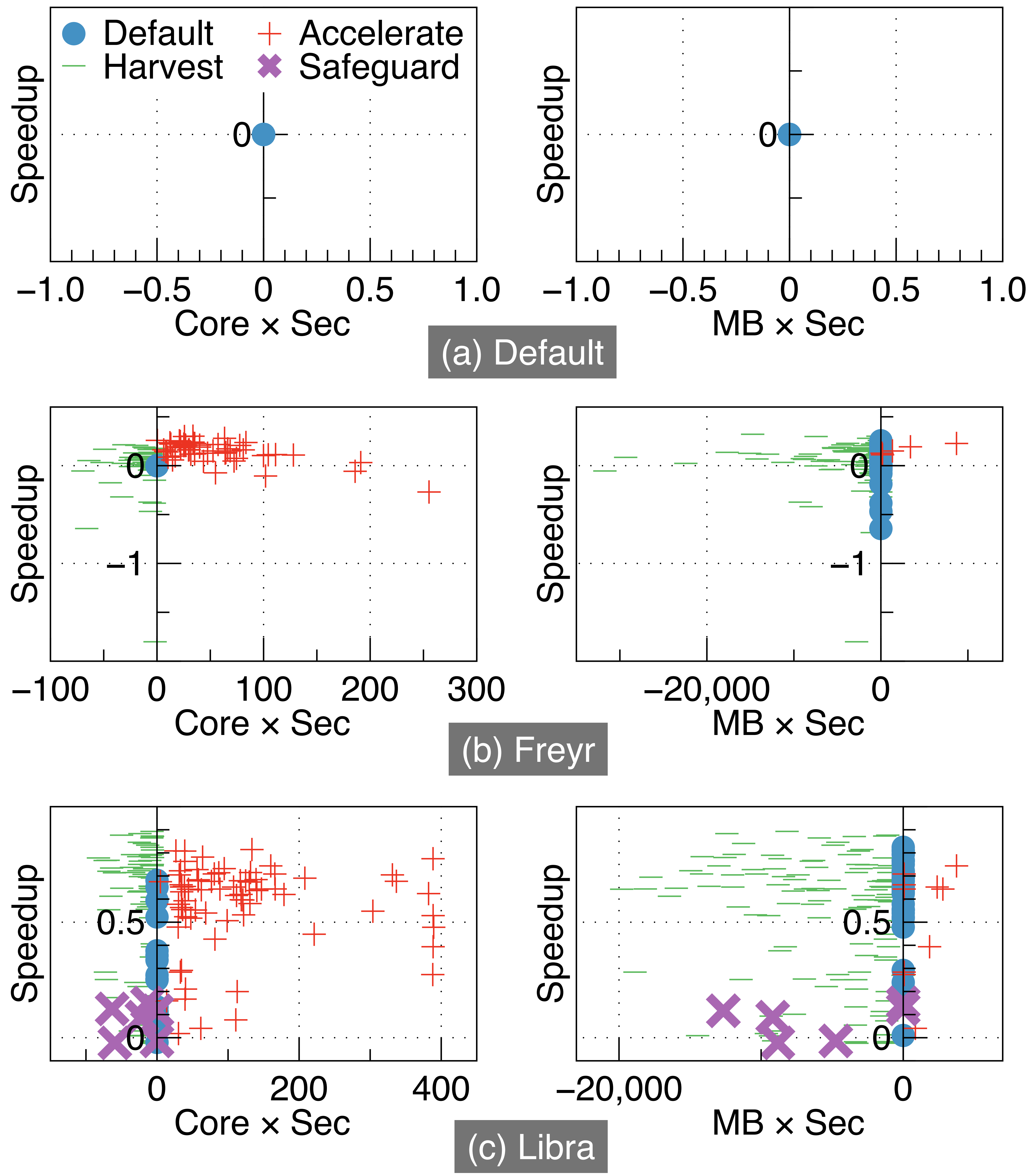
Default: OpenWhisk default **JSQ**: Join-the-Shortest-Queue

MWS: Min-Worker-Set **RR**: Round-robin

Resource Utilization

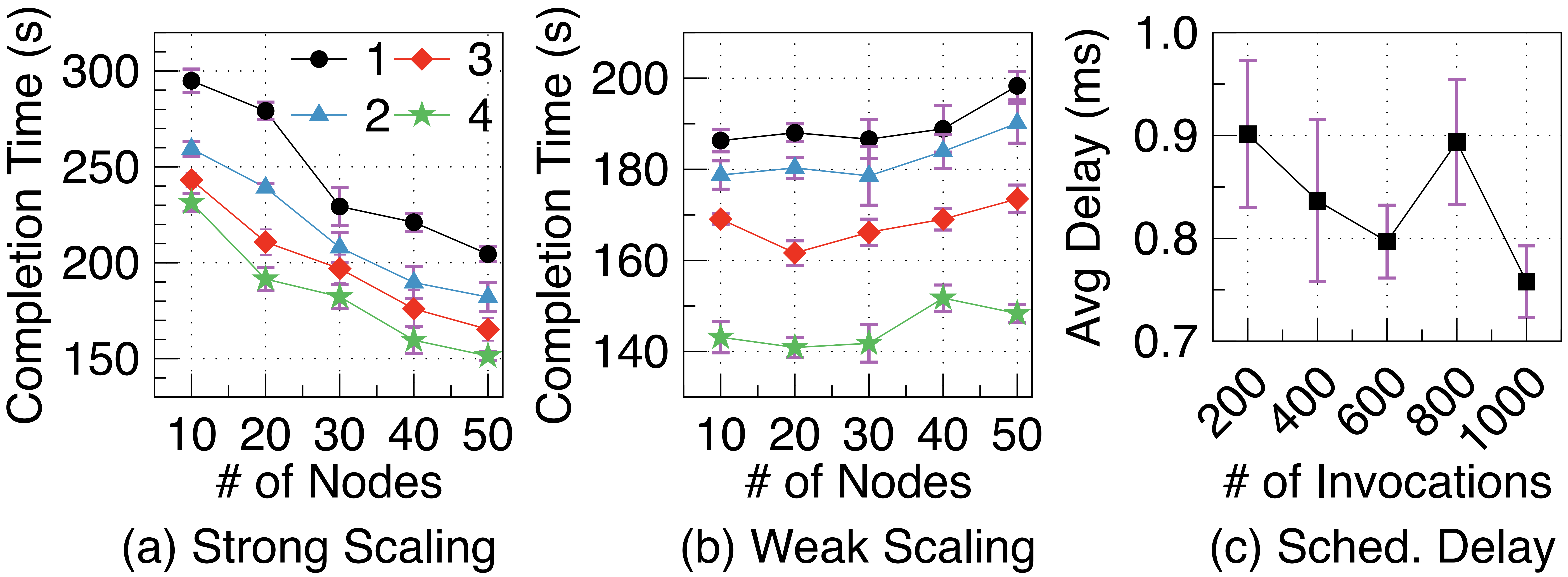


Harvesting and Safeguarding



$$Speedup := \frac{t^{user} - t^{*}}{t^{user}}$$

Scalability & Overhead



Timeliness-aware resource harvesting & scheduling

Input data size-awareness

Timely safeguard

Libra

39%

lower function response latency

3x

higher resource utilization

Libra Code Repo:

https://github.com/IntelliSys-Lab/Libra-HPDC23

Corresponding Author:

Hanfei Yu <hyu25@lsu.edu>

Hao Wang <haowang@lsu.edu>

IntelliSys Lab