FRUGAL CLOUD

Enhancing IaaS Consolidation with Resource Oversubscription

Pierre JACQUET

Supervisors
Thomas LEDOUX, *IMT Atlantique*
Romain ROUVOY, *Univ. Lille*

Reviewers
Pascal FELBER, *Univ. Neuchâtel*
Gaël THOMAS, *Inria Saclay*

Examiners
Laurent LEFEVRE, *Inria Lyon*
Anne-Cécile ORGERIE, *CNRS*

Inria Lille, 19th July 2024

Plan

Motivation

Contributions

1. *Oversubscribe under stability consideration*
2. *Introduce per-vCPU oversubscription*
3. *Balance complementary oversubscription levels*

Conclusions & perspectives

Motivation

Why sharing Cloud resources?

Motivation

ICT is responsible for 2.1–3.9% of global GHG emissions…

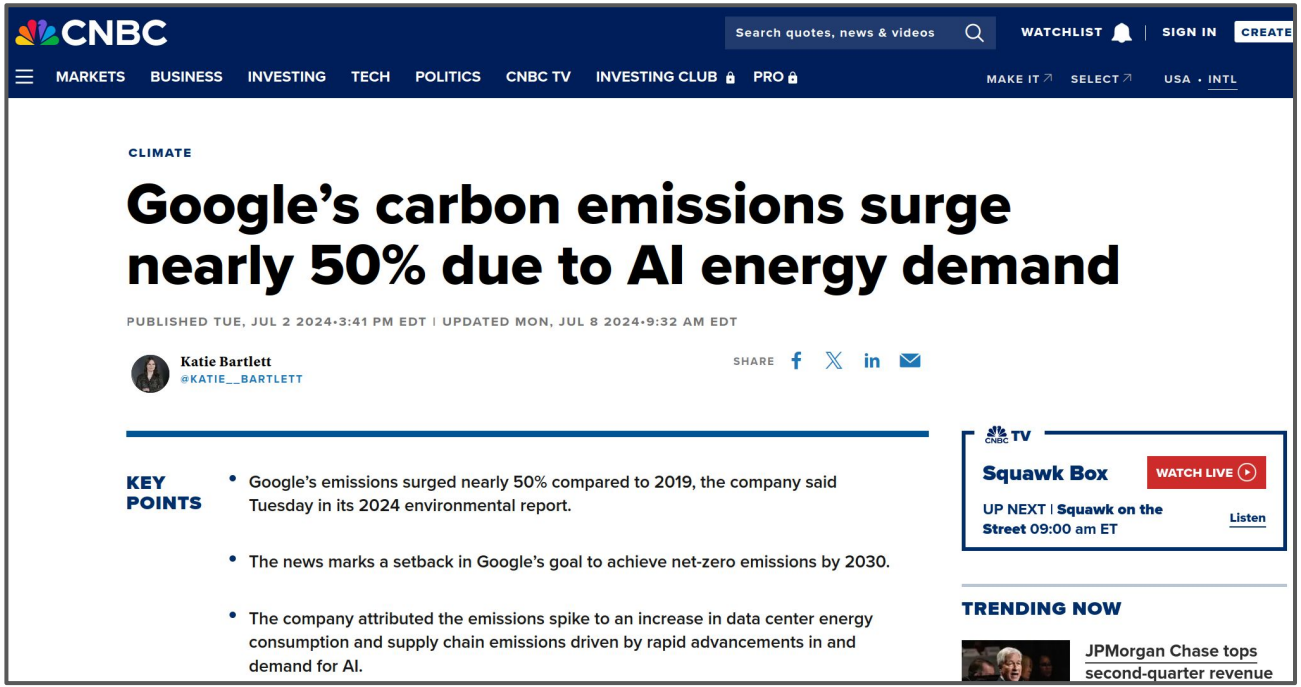
	2015	2022	Change
Internet users	3 billion	5.3 billion	+78%
Internet traffic	0.6 ZB	4.4 ZB	+600%
Data centre workloads	180 million	800 million	+340%
Data centre energy use (excluding crypto)	200 TWh	240-340 TWh	+20-70%
Crypto mining energy use	4 TWh	100-150 TWh	+2300-3500%
Data transmission network energy use	220 TWh	260-360 TWh	+18-64%

…and its emissions keep **increasing**

Freitag et al., IEA

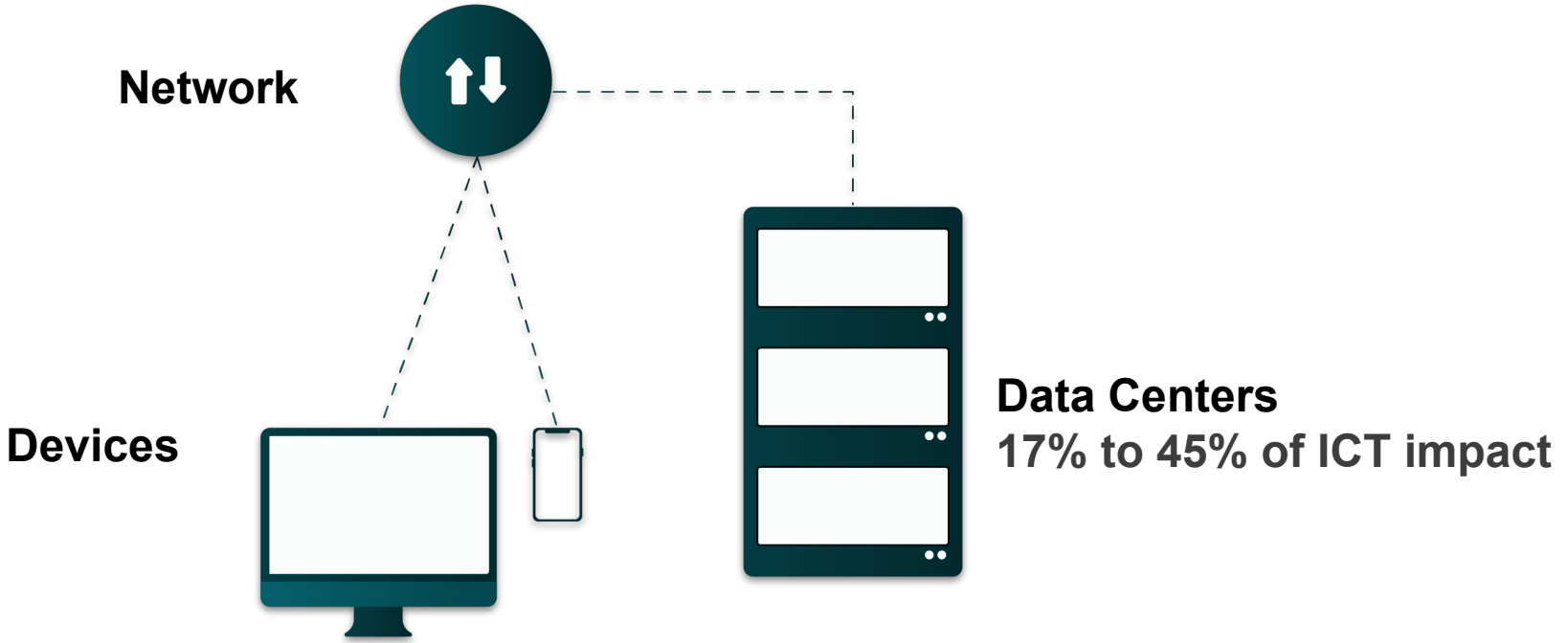
Motivation

ICT is responsible for 2.1–3.9% of global GHG emissions…



…and its emissions keep **increasing**

Motivation



ICT different scopes

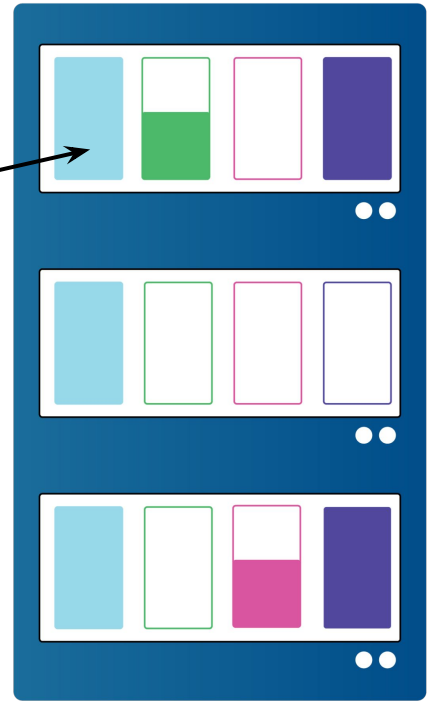
Malmodin et al., Belkhir et al.

Motivation

Cloud servers are composed of**:**

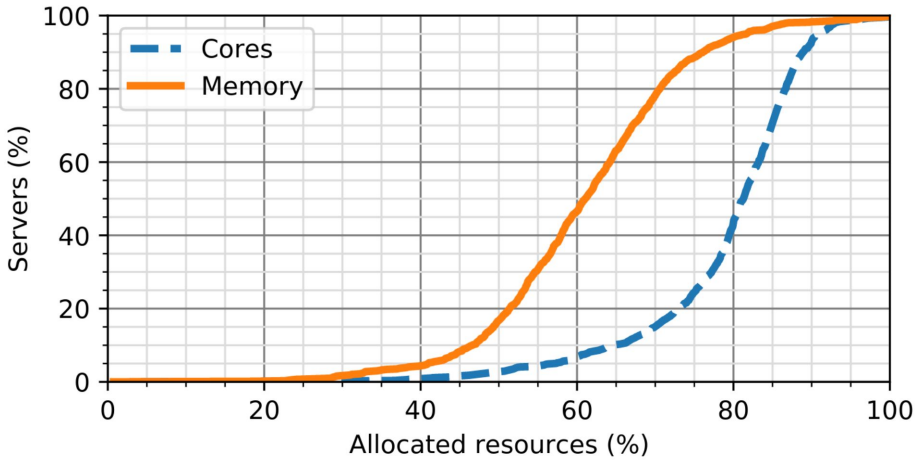
- Computing resources (CPU)
- Memory resources (RAM)
- Network resources
- Storage resources

In an Infrastructure-as-a-Service (**IaaS)** context**,** virtual units compose a virtual machine (**VM**)

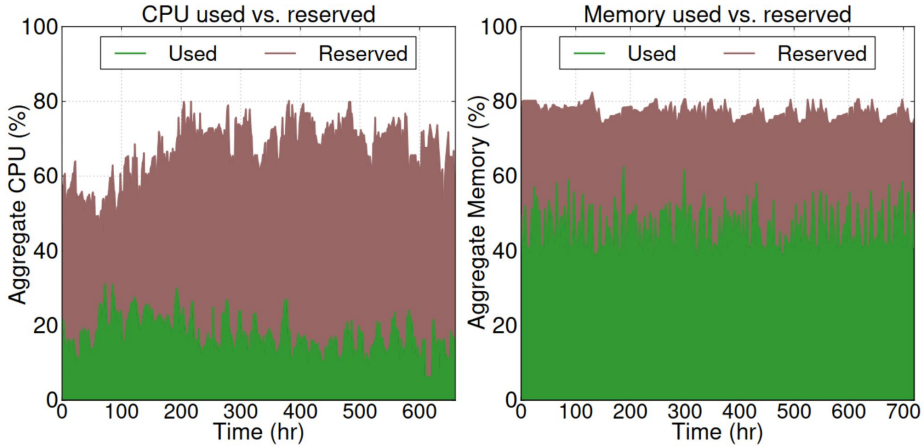


Motivation

Cloud infrastructure are under-used



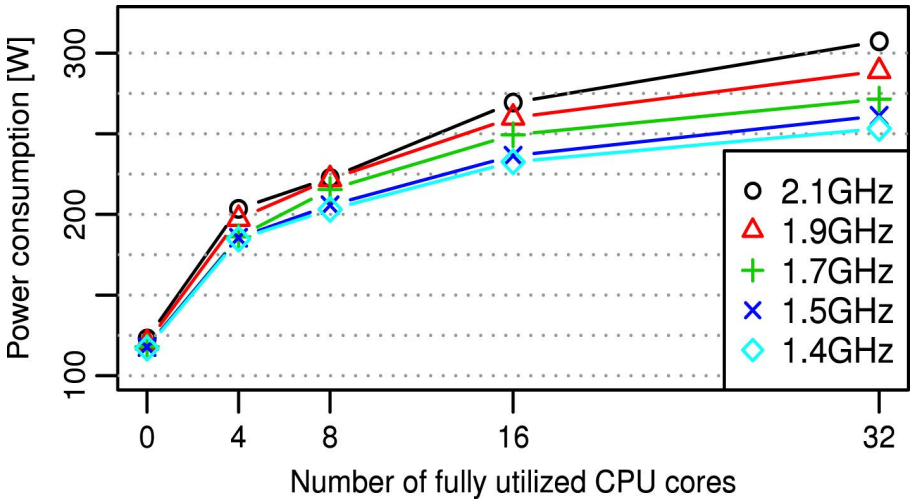
Not all resources are **allocated** [1]



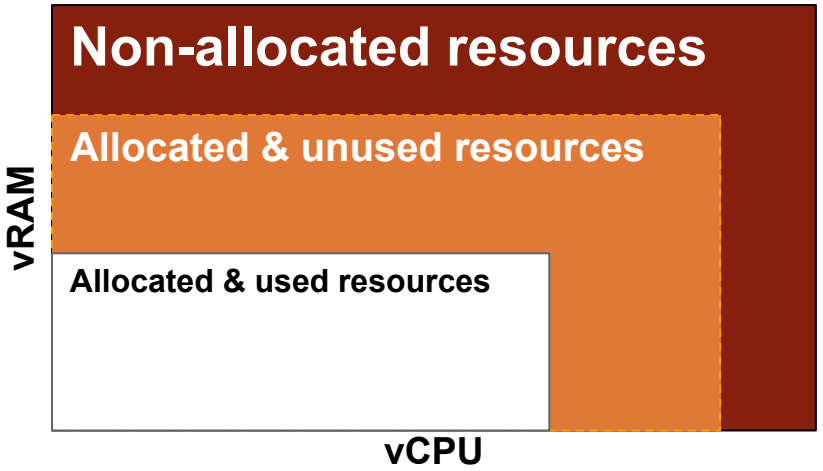
Not all allocated resources are **used** [2]

Motivation

Cloud infrastructure are under-used

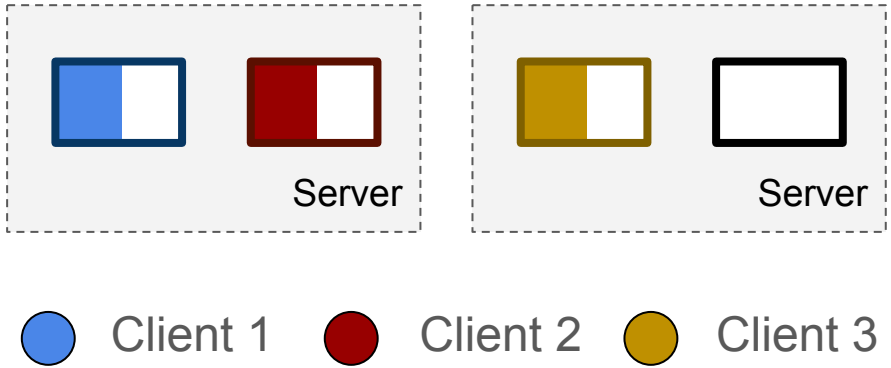


Impacts **server consumption**



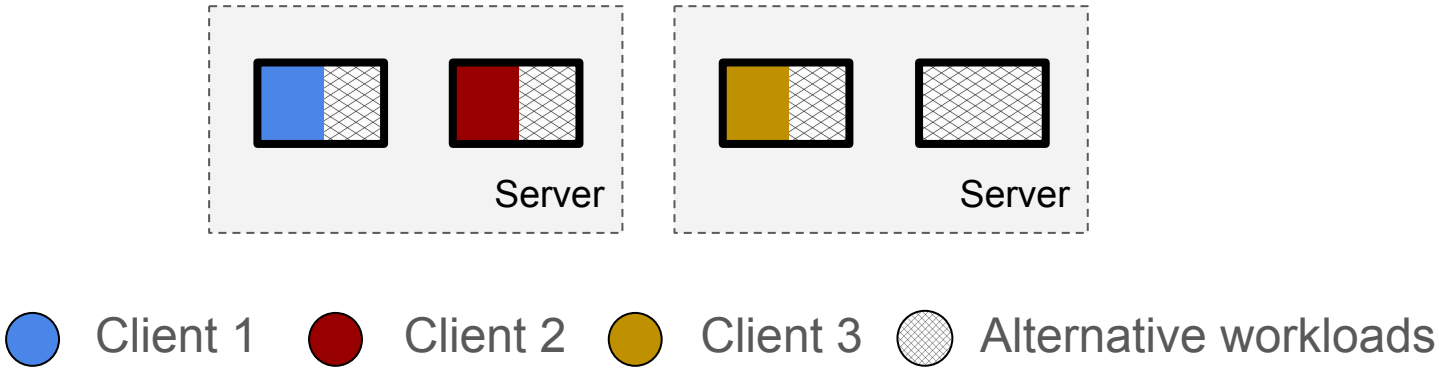
Impacts the number of **provisioned servers**

Motivation



- A problem addressed using different (complementary) approaches

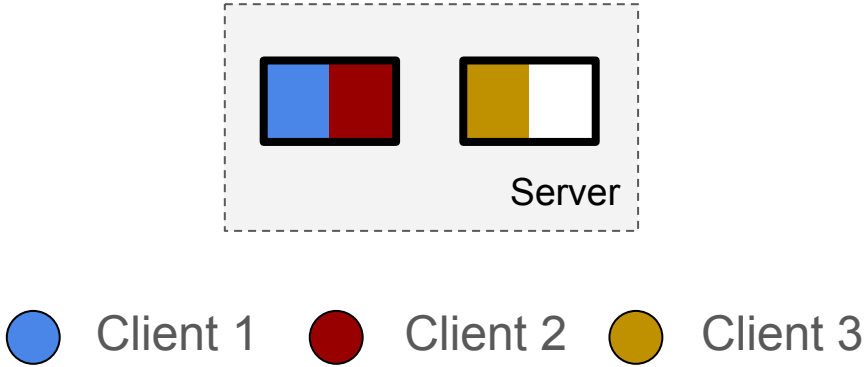
Motivation



- A problem addressed using different (complementary) approaches

  1) Fill with **heterogeneous workloads,** *e.g., SpotVM, HarvestVM*

Motivation



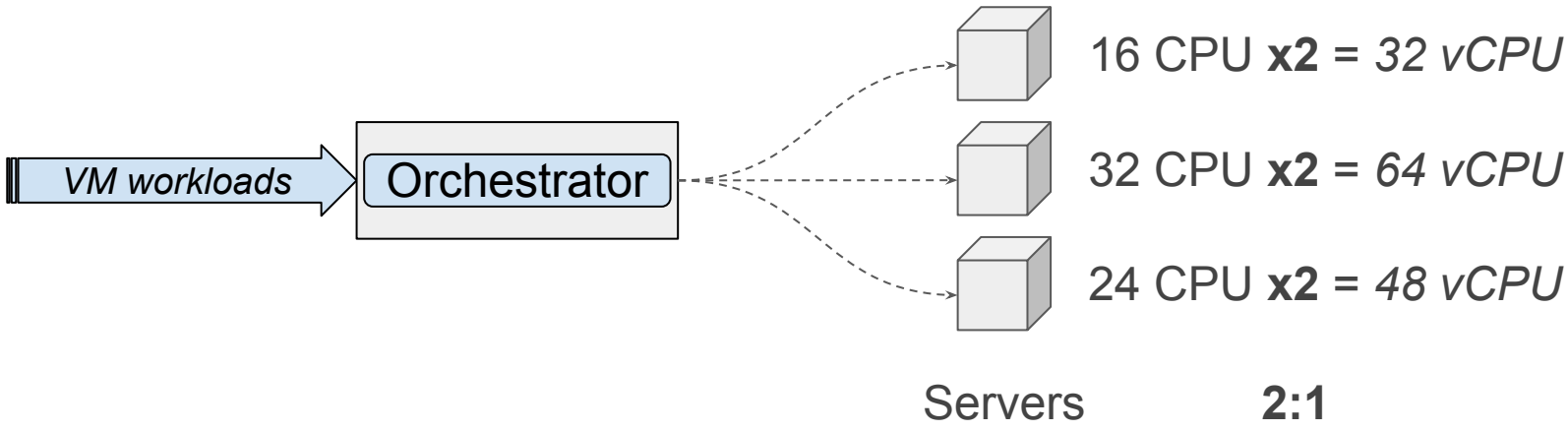
- A problem addressed using different (complementary) approaches

  2) Pack with **homogeneous workloads** by sharing cores
     - *i.e.*, more than 1 client per core, oversubscription (here, **2:1**)

Motivation

How to define the right amount of clients when sharing resources?

- Ratio commonly set **statically** at the cluster level

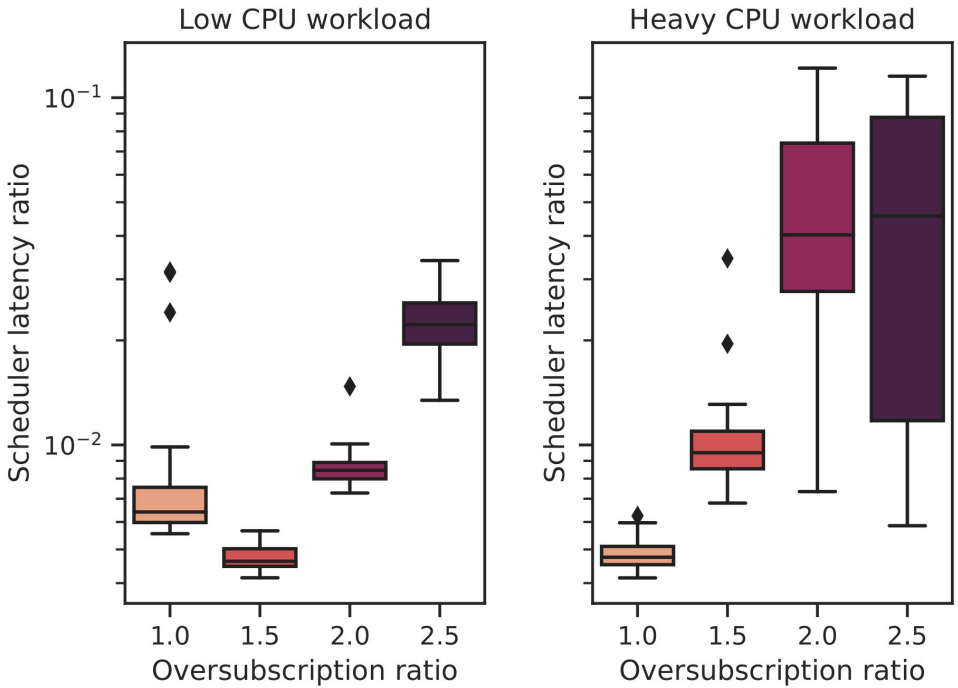


Oversubscribe under stability consideration,

The example of ScroogeVM

ScroogeVM: **Motivation**

- **Static** oversubscription remains the most used approach
  - Or, oversubscription does depend on workload intensity

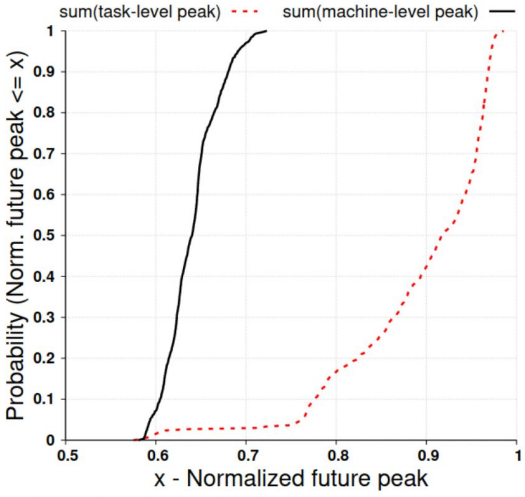


ScroogeVM: **Motivation**

- **Dynamic** oversubscription
  - Relies on pessimistic prediction of **next peak usage**

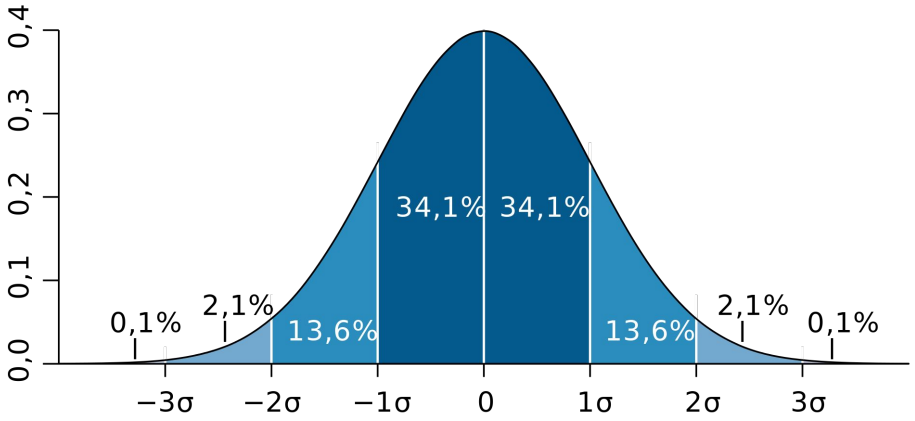
ScroogeVM: **Motivation**

- **Dynamic** oversubscription
  - Relies on pessimistic prediction of **next peak usage**



Microsoft research:

Next peak: sum of VMs <u>percentile</u>

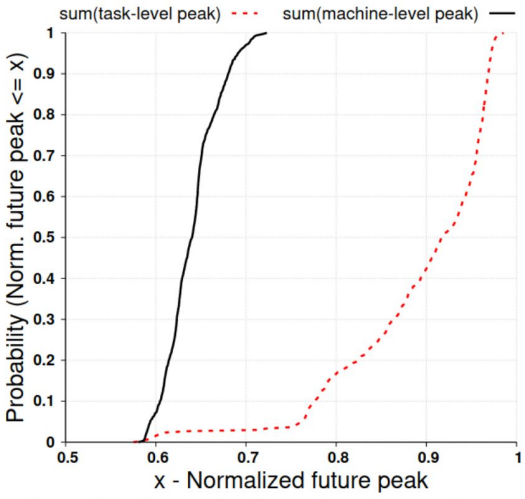


Google research:

Next peak: use server <u>std deviation</u>

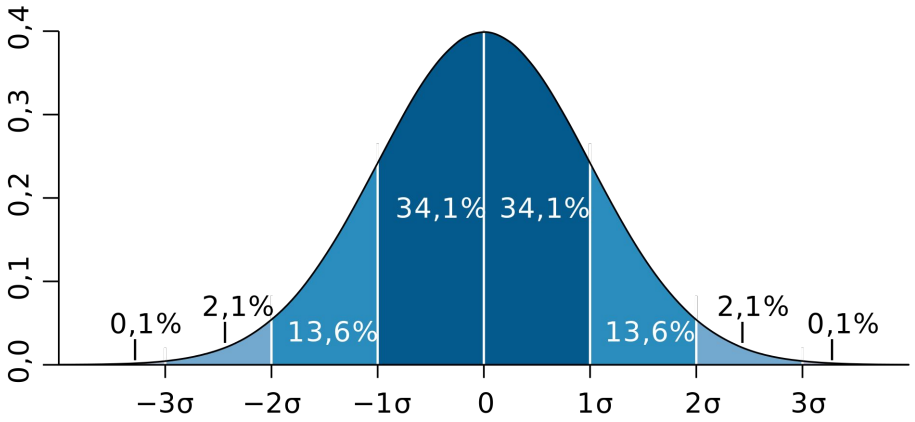
ScroogeVM: **Motivation**

- **Dynamic** oversubscription
  - Relies on pessimistic prediction of **next peak usage**



Microsoft research:

Which <u>percentile</u> to use?



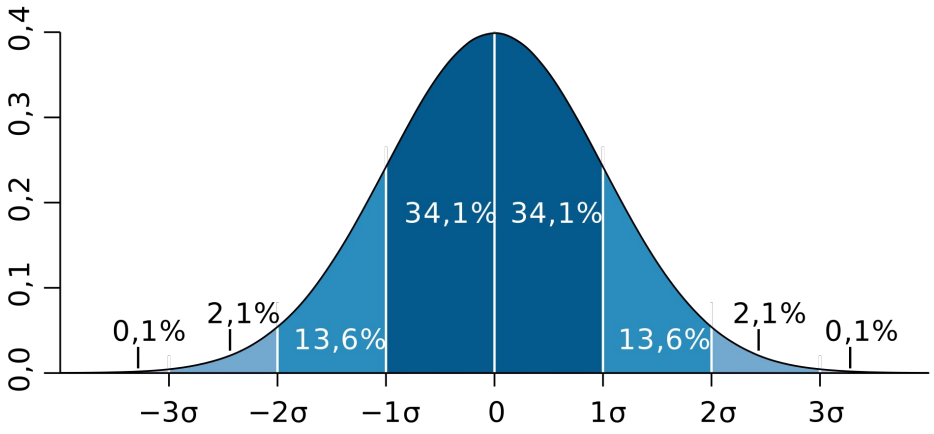
Google research:

Which <u>sigma</u> to use?

ScroogeVM: **Motivation**

Paper assumption:

The **pessimistic degree** of the peak computation depends on quiescent state



$$\overline{cpu} + N \times \sigma$$

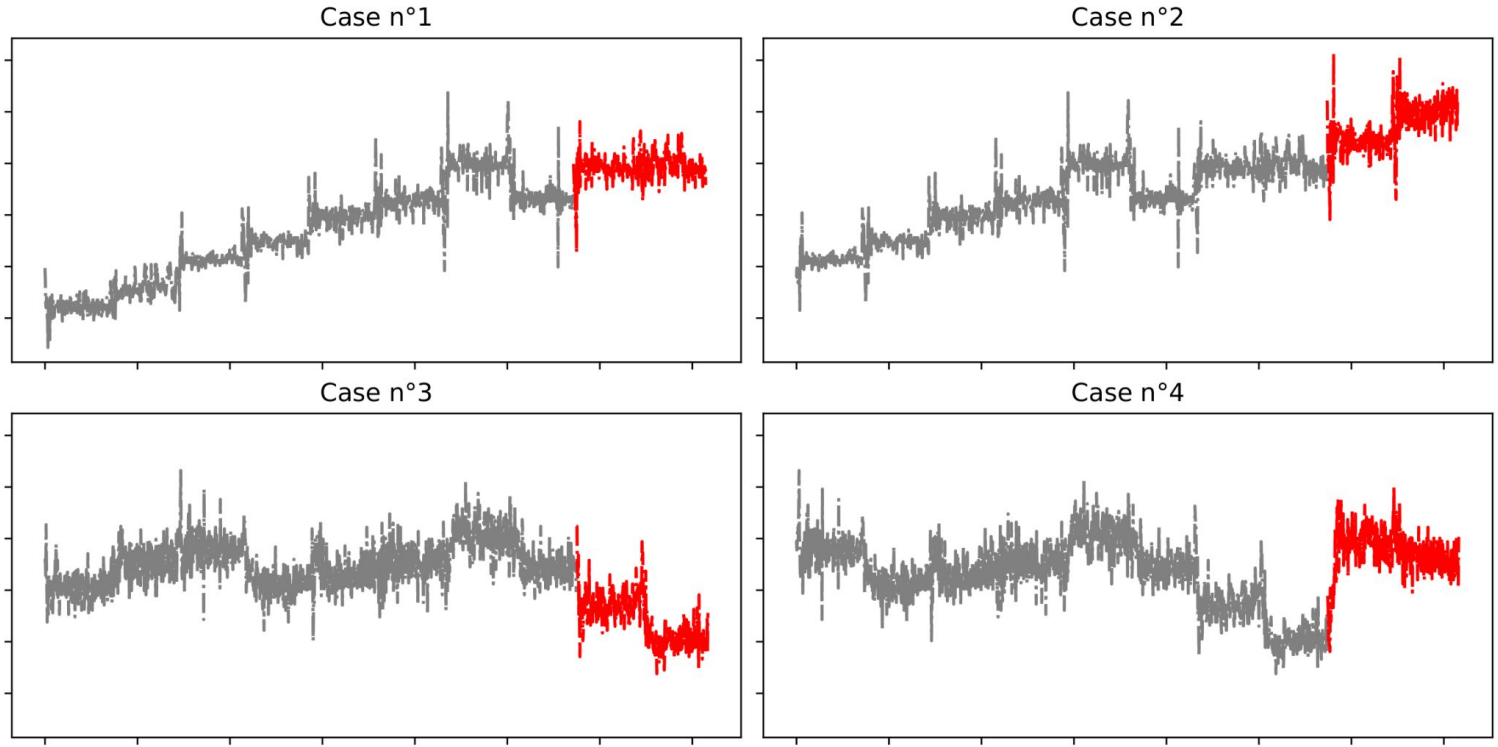
ScroogeVM: **Implementation**

How to evaluate a PM quiescent state?

1. Check on average bounds
2. Check on percentile bounds
3. Null-hypothesis significance testing
4. LSTM prediction

ScroogeVM: **Implementation**

How to evaluate a PM quiescent state?



ScroogeVM: **Implementation**

How to evaluate a PM quiescent state?

Algorithm 1 Quiescent state detection algorithm

Input Historical dataset, last window
Output Quiescent state

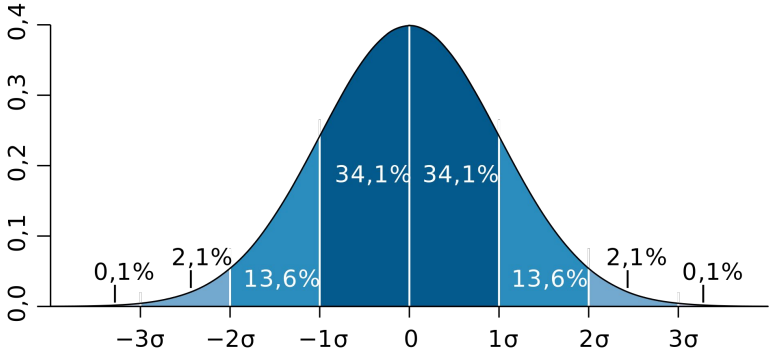
1: $\mathcal{M}_{LSTM} \leftarrow$ Generate model from historical dataset
2: $forecasted \leftarrow predict(\mathcal{M}_{LSTM}, historical\_set)$
3: $predicted \leftarrow predict(\mathcal{M}_{LSTM}, last\_window)$
4: $\delta \leftarrow |RMSE(forecasted) - RMSE(predicted)|$
5: **if** $\delta > threshold$ **then**
6: **return** `UNSTABLE`
7: **end if**
8: **return** `QUIESCENT`

QUANTITATIVE EVALUATION OF QUIESCENT STATE CLASSIFIERS

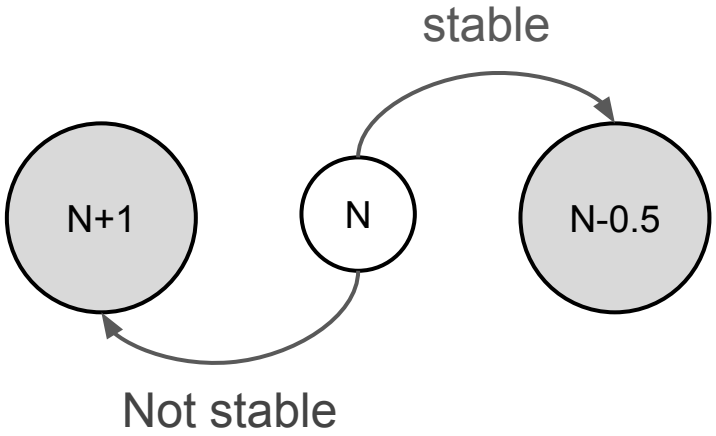
Classifier	Accuracy	Precision	Recall	F-score
Average	0.68	0.8	0.52	0.63
Percentile	0.57	0.55	**0.87**	0.67
P-value	0.52	**1.0**	0.06	0.12
LSTM	**0.88**	0.93	0.84	**0.88**

ScroogeVM: **Implementation**

Oversubscription based on quiescent state

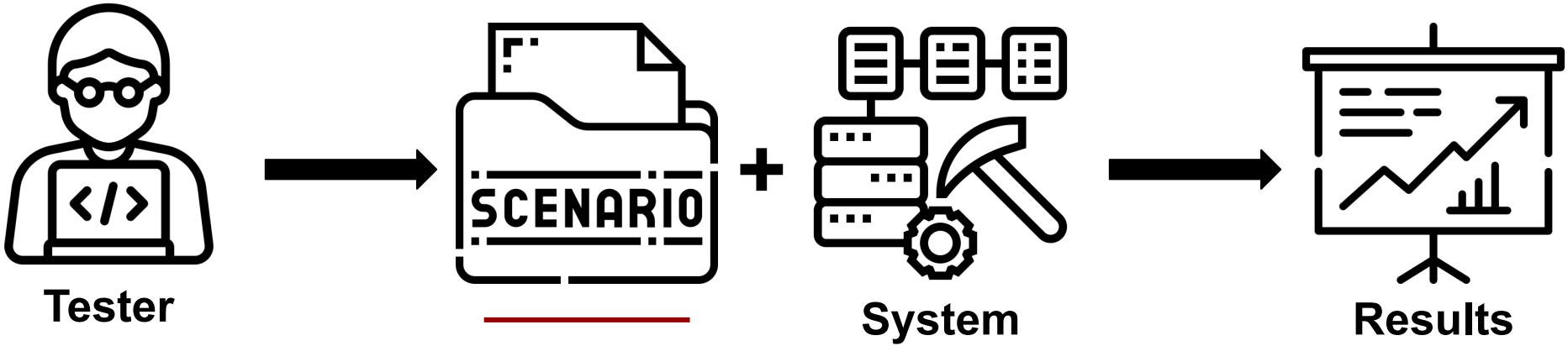


$$\overline{cpu} + N \times \sigma$$



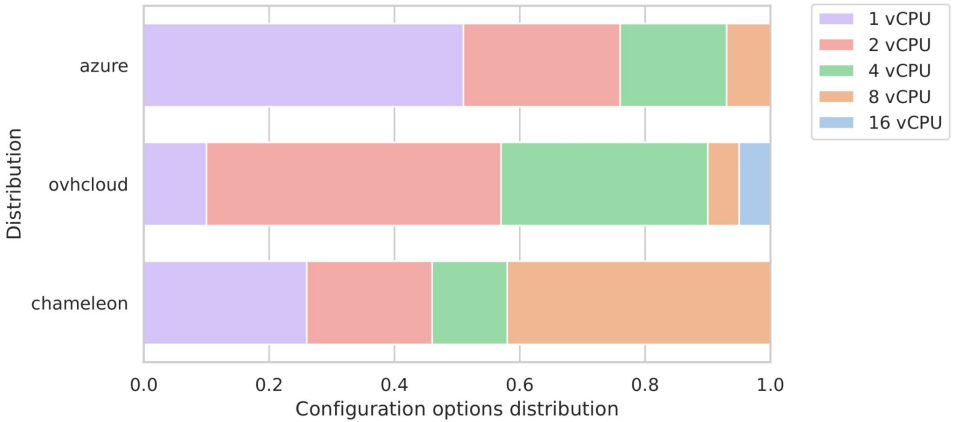
Customize **N** based on quiescent strike

ScroogeVM: **Evaluation**

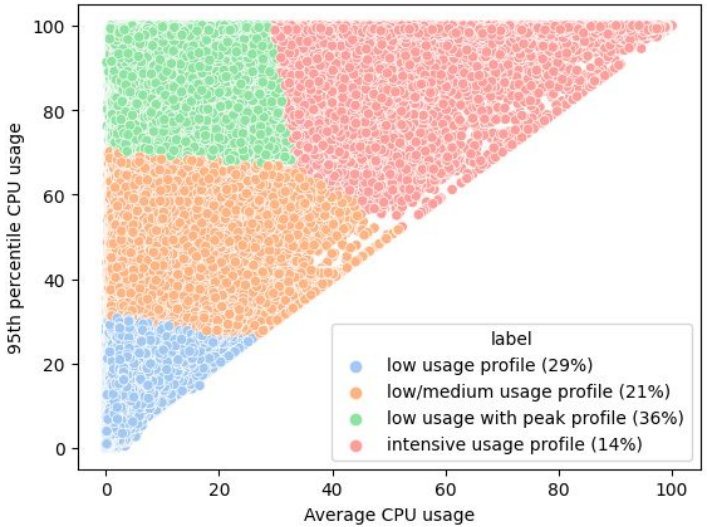


ScroogeVM: **Evaluation**

Realistic scenario using **CloudFactory**



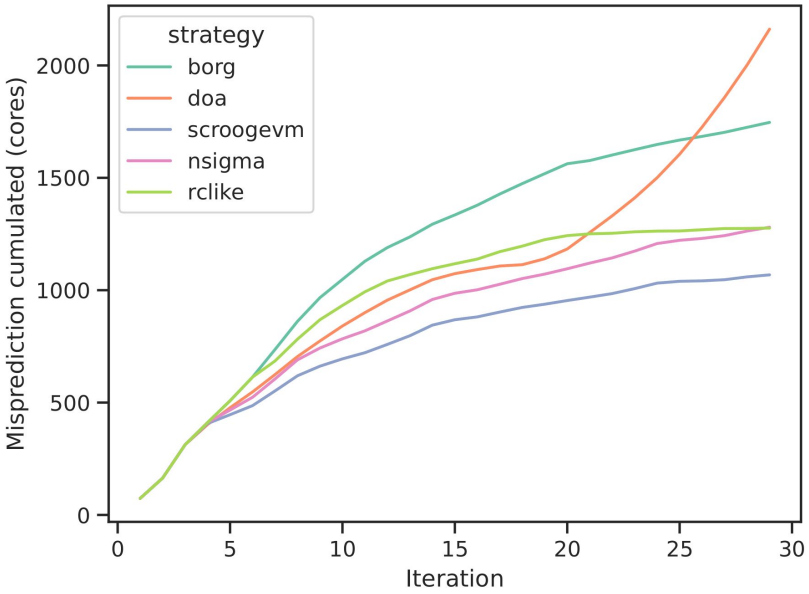
Configuration distribution on various CP



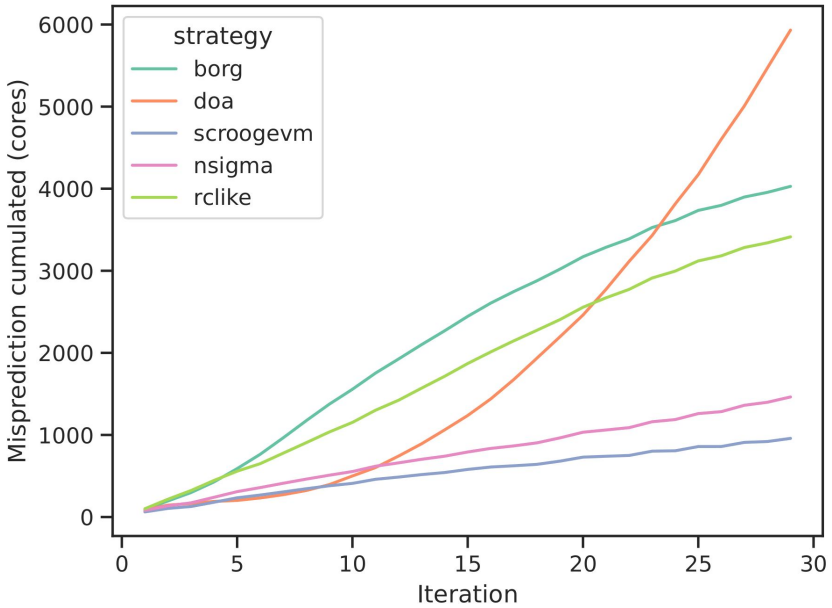
Usage profile on Azure dataset

ScroogeVM: **Evaluation**

Misprediction: Compared what was evaluated as available at J-1 to what is currently available at J



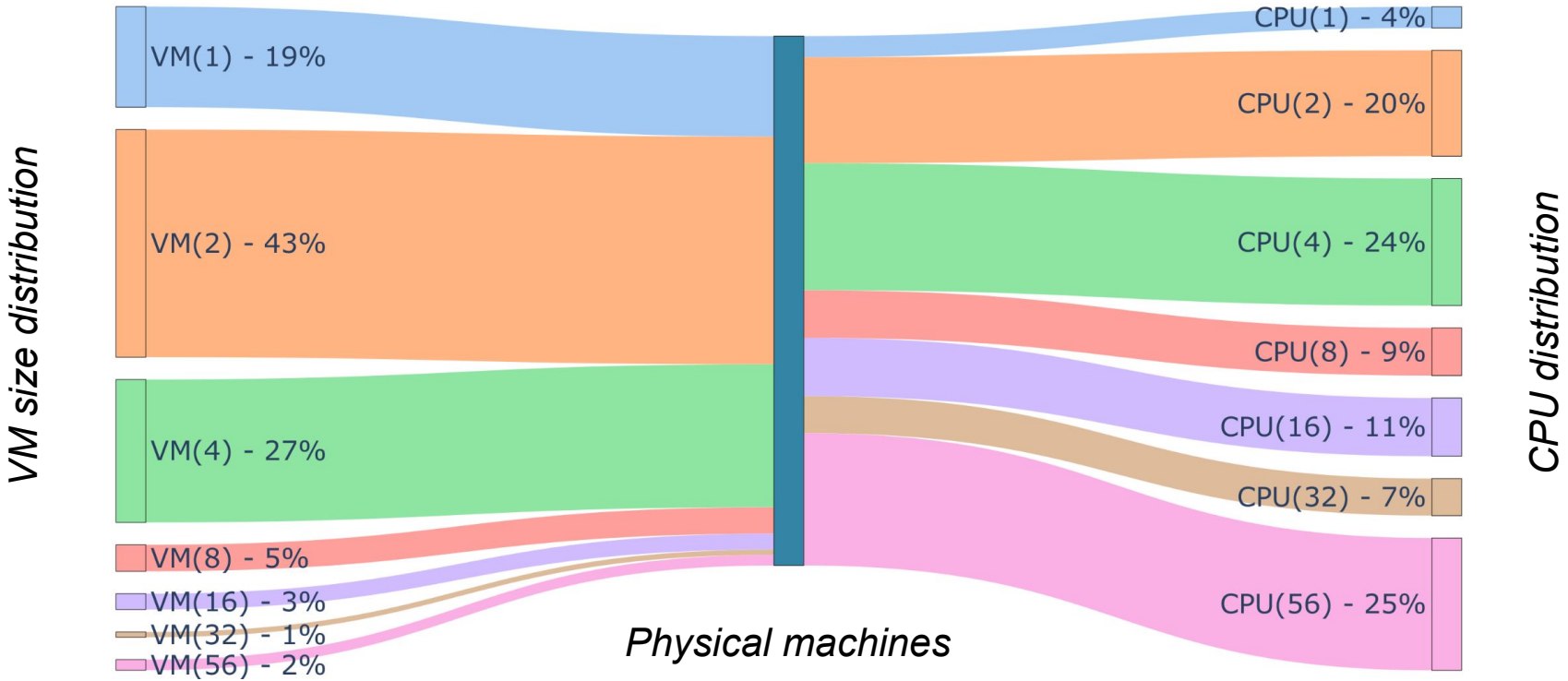
Misprediction on an decreasing workload



Misprediction on an increasing workload

Introducing per-vCPU oversubscription,
The example of SweetSpotVM

SweetSpotVM: **Motivation**



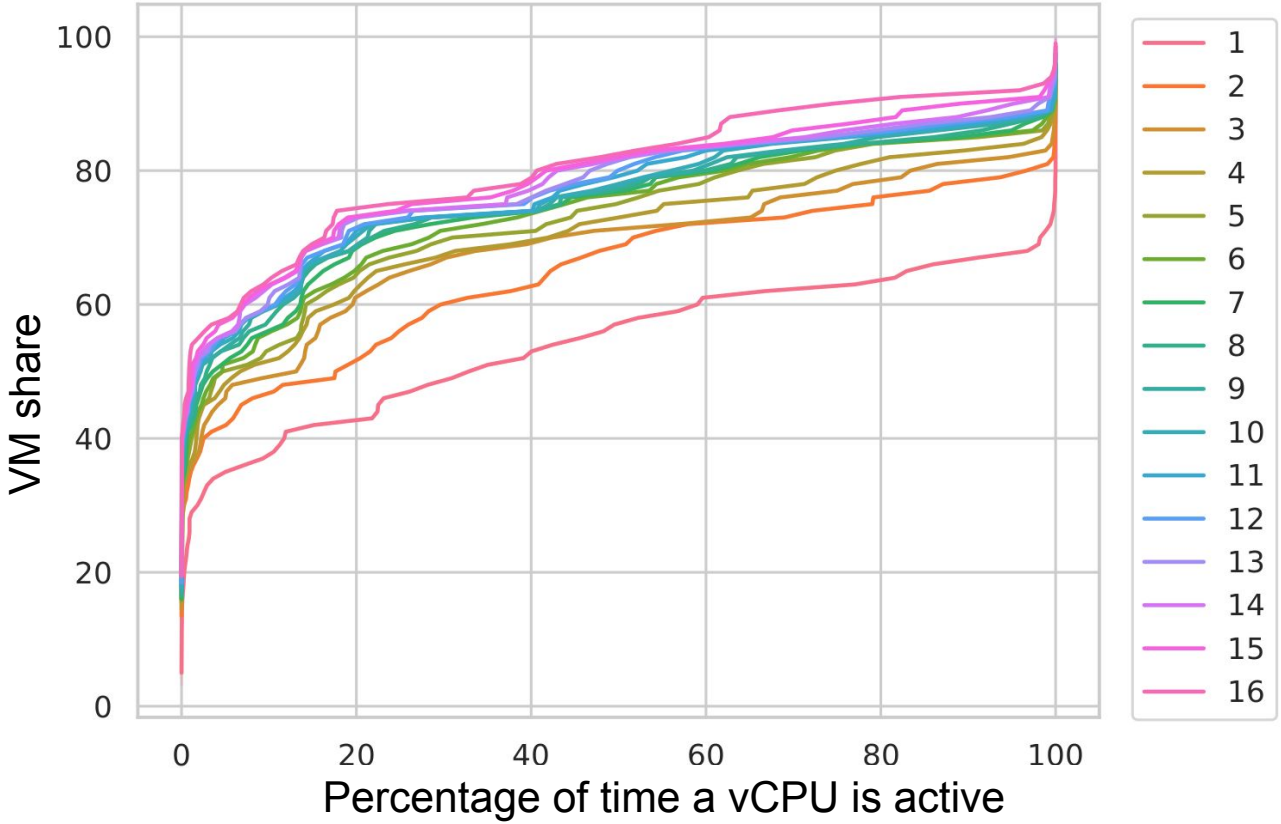
Most of CPUs are provisioned by a **small subset of VMs**

SweetSpotVM: **Motivation**

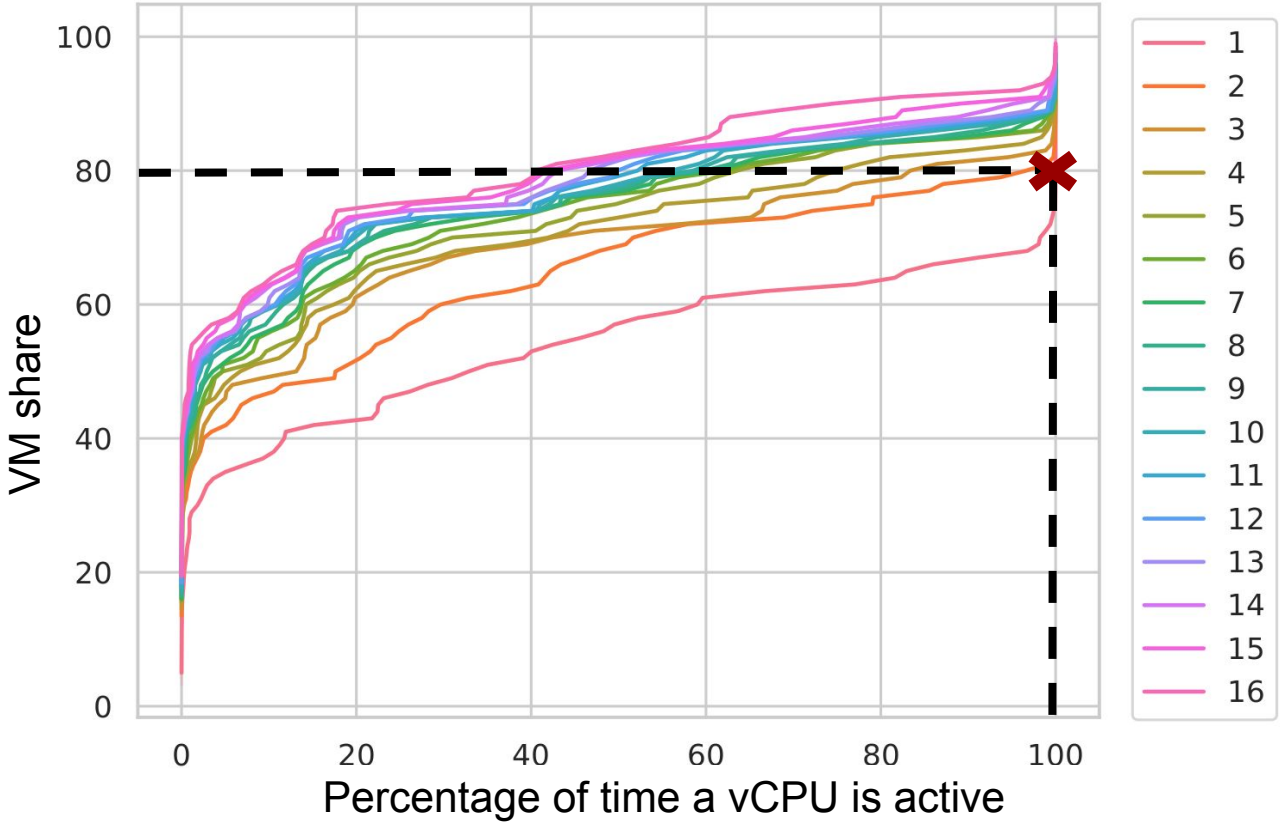
- Is the typical VM workload distributed?

2 situations reporting on an 50% CPU usage

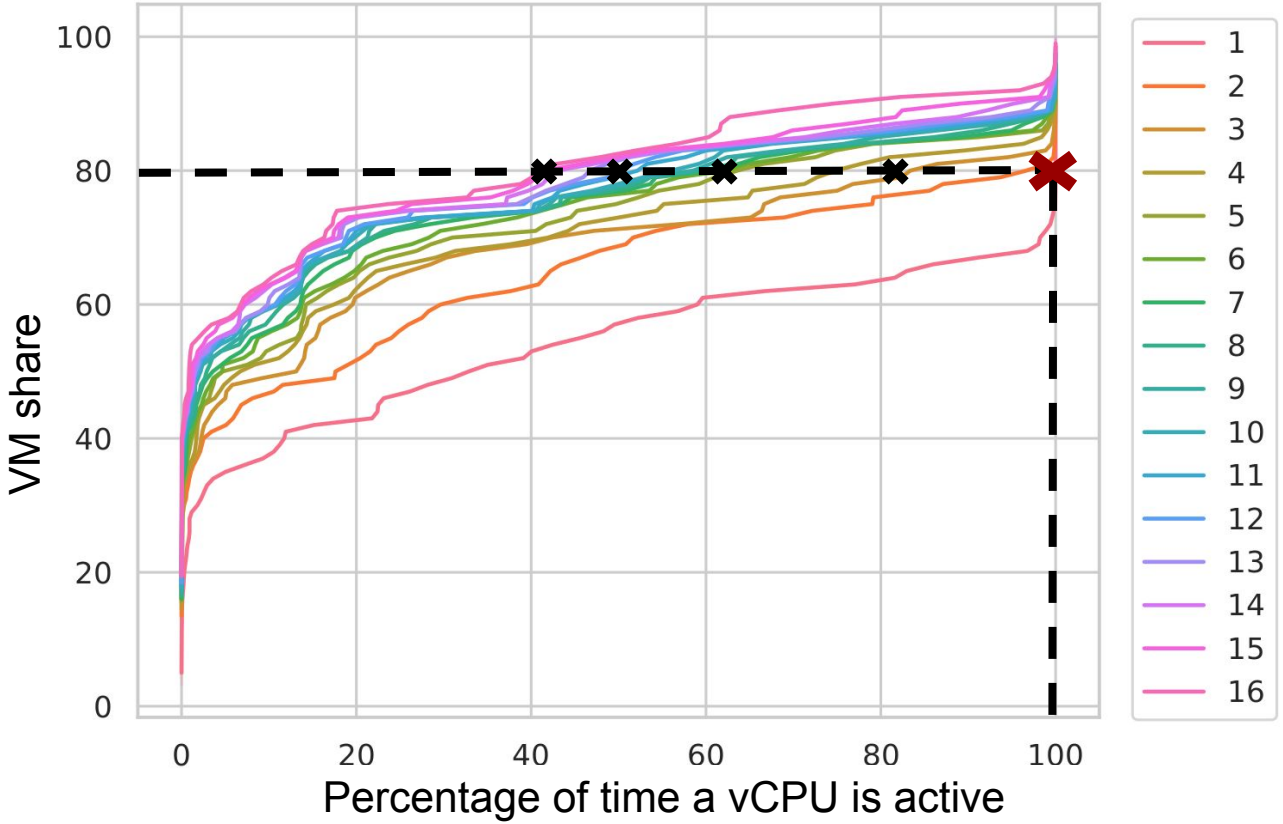
SweetSpotVM: **Motivation**



SweetSpotVM: **Motivation**

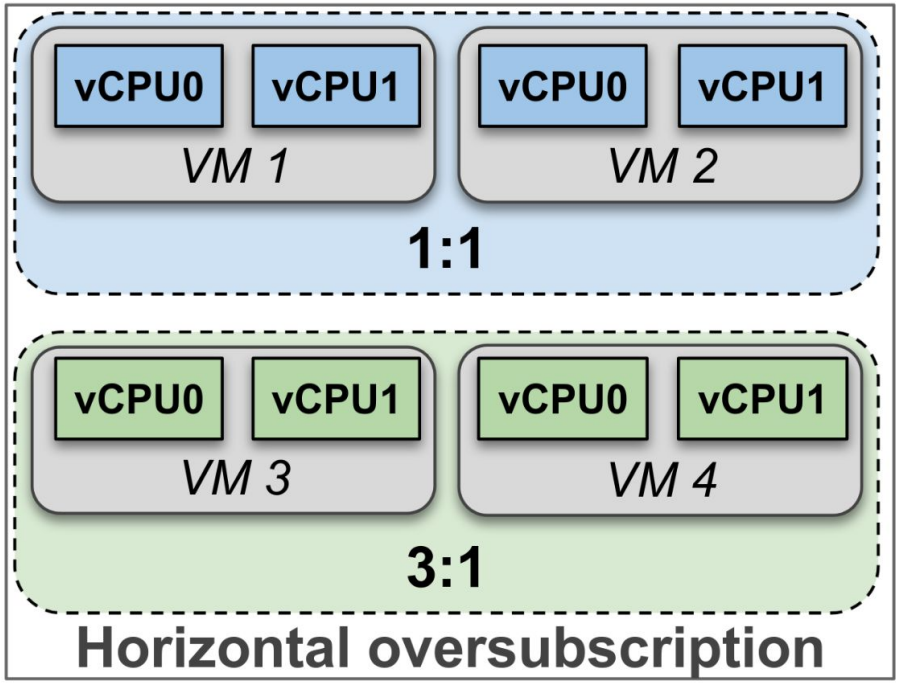


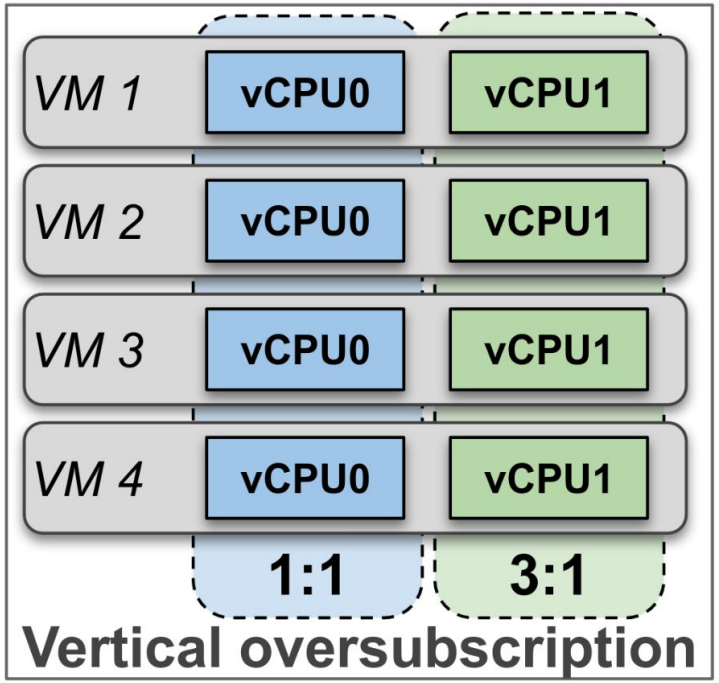
SweetSpotVM: **Motivation**



Large VMs hosted by OVHCloud **do not use all vCPUs equally**

SweetSpotVM: **Motivation**

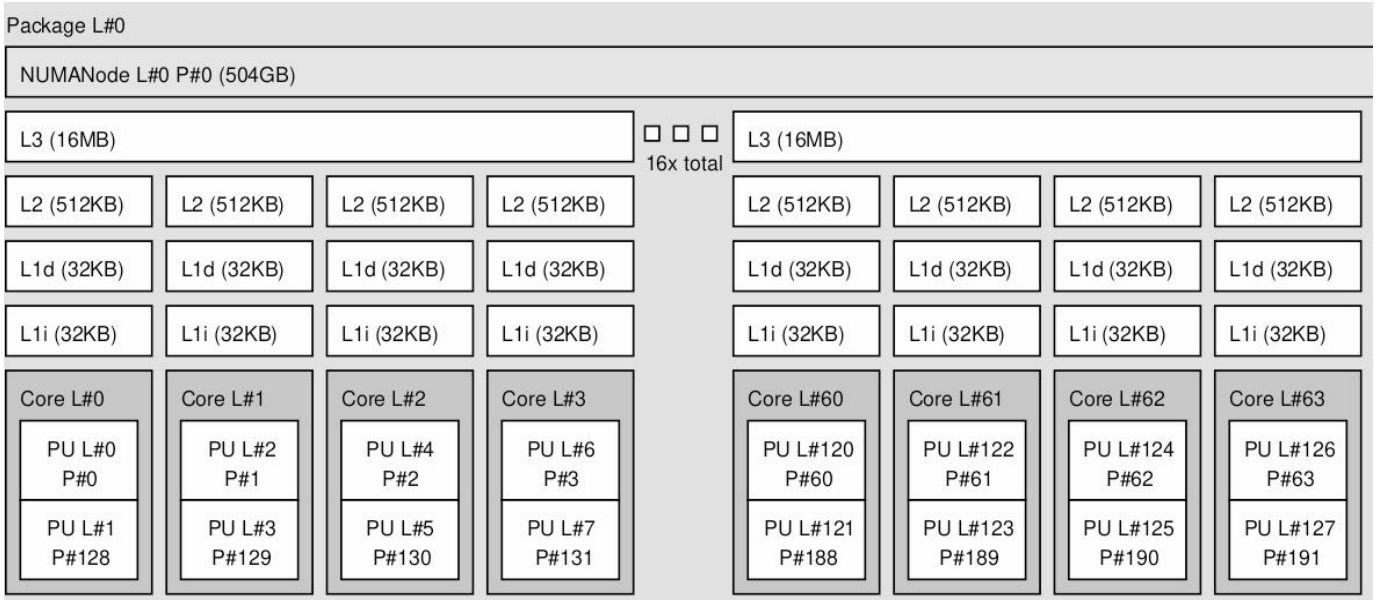




Changing the perspective on vCPU oversubscription

SweetSpotVM: **Implementation**

- How to oversubscribe to multiple levels a given host?

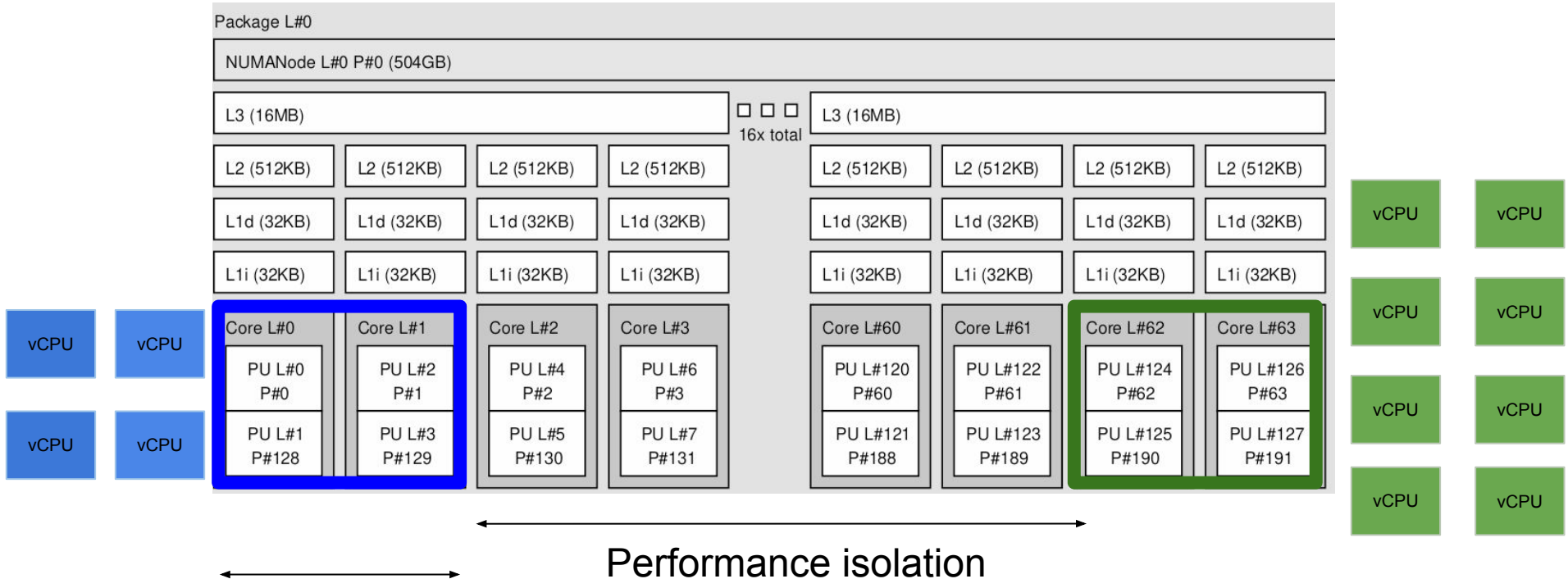


Goals:

- Performance isolation
- CFS / EEVDF awareness

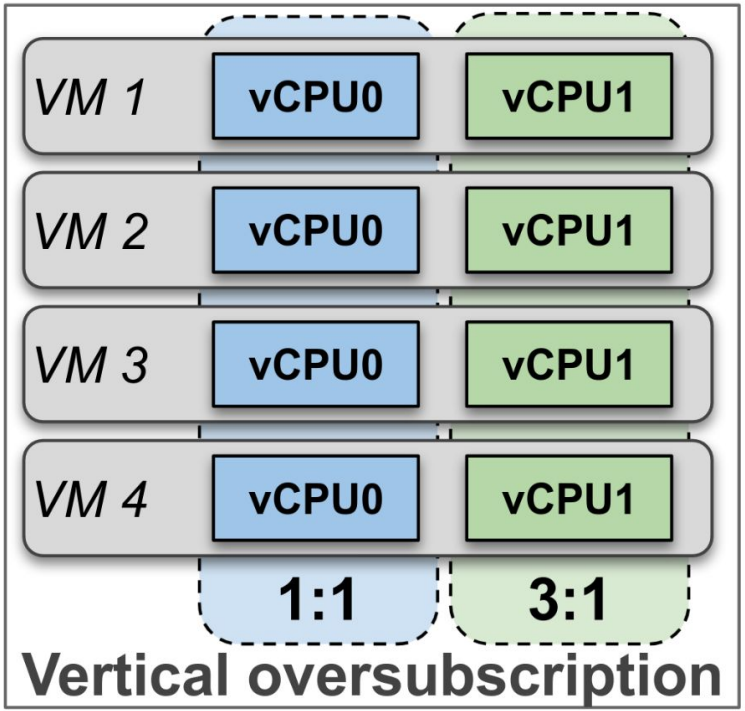
SweetSpotVM: **Implementation**

- How to oversubscribe to multiple levels a given host?



SweetSpotVM: **Implementation**

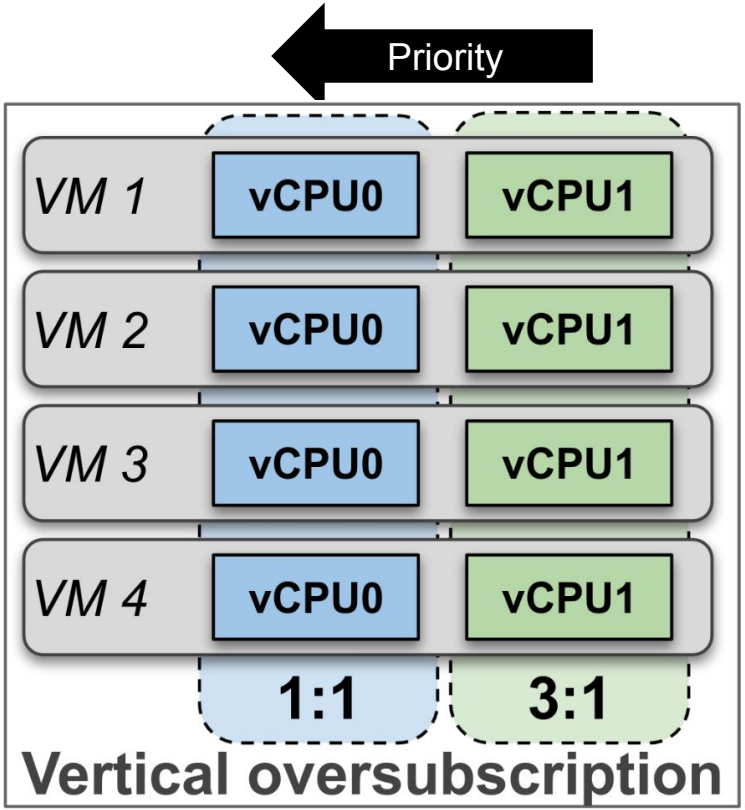
- Local agent written in Python
  - Exposes a REST API
  - Interacts with hypervisor using Libvirt API
  - “Regular” QEMU/KVM VMs



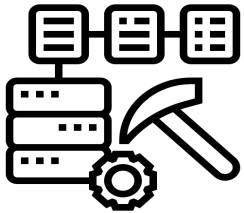
SweetSpotVM: **Implementation**

On core priority

- Passive way: inform VM scheduling
  - ARM Big.Little
  - Intel P-Core/E-core
  - …

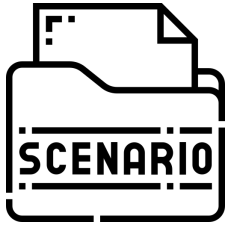


SweetSpotVM: **Evaluation**



256 cores worker:

1. Without oversubscription
2. With horizontal oversubscription
3. With vertical oversubscription



Set of VMs hosting:

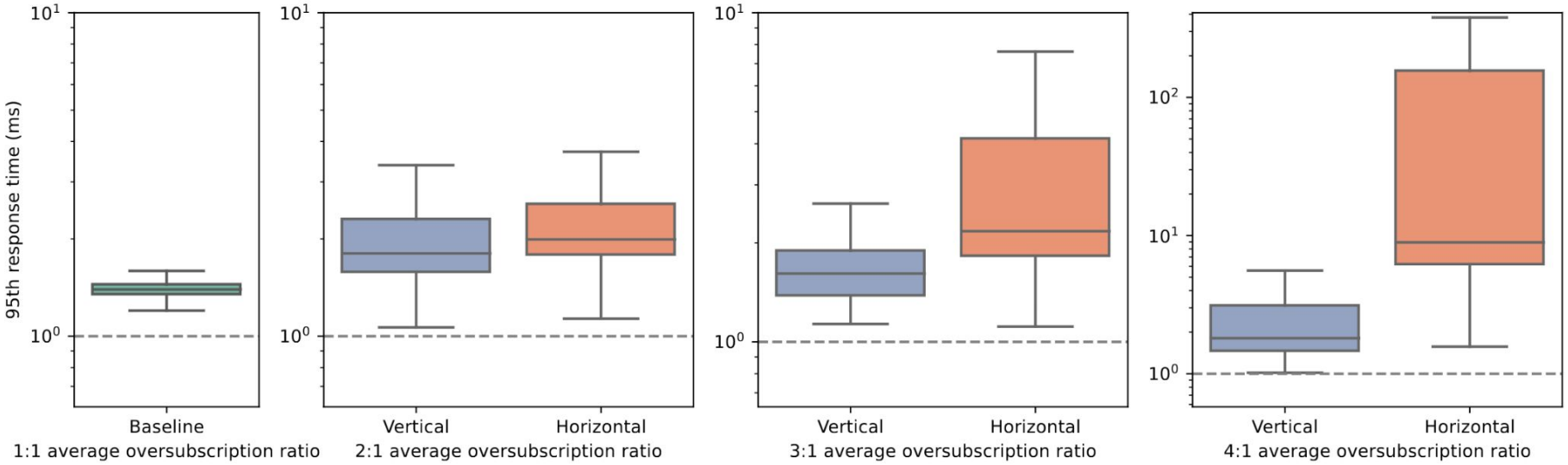
- ∅ (idle)
- `stress-ng`
- DeathStarbench social network*

SweetSpotVM: **Evaluation**

- Vertical oversubscription setting

Oversubscription target	vCPU0 ratio	vCPU1 ratio	vCPU2–7 ratio
2:1	1:1	1.5:1	2.6:1
3:1	1:1	1.5:1	6.0:1
4:1	1:1	1.5:1	16.0:1

SweetSpotVM: **Evaluation**

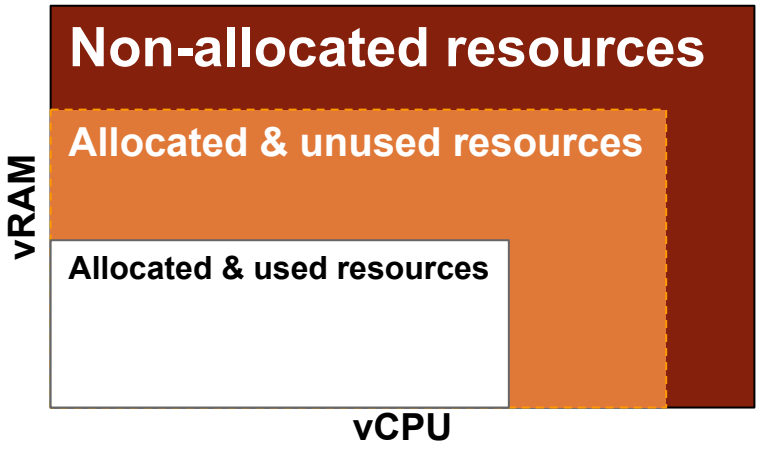


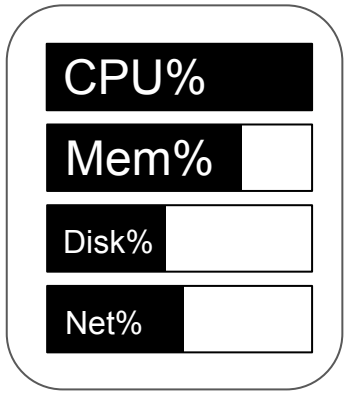
Oversubscribing differently vCPUs can **reduce the performance overhead**

Balancing complementary oversubscription levels,

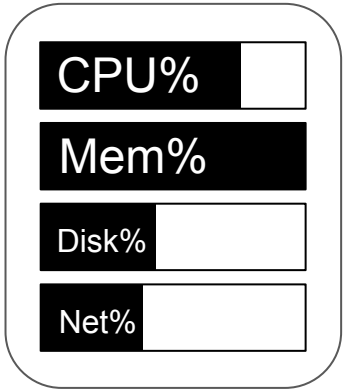
The example of SlackVM

SlackVM: **Motivation**



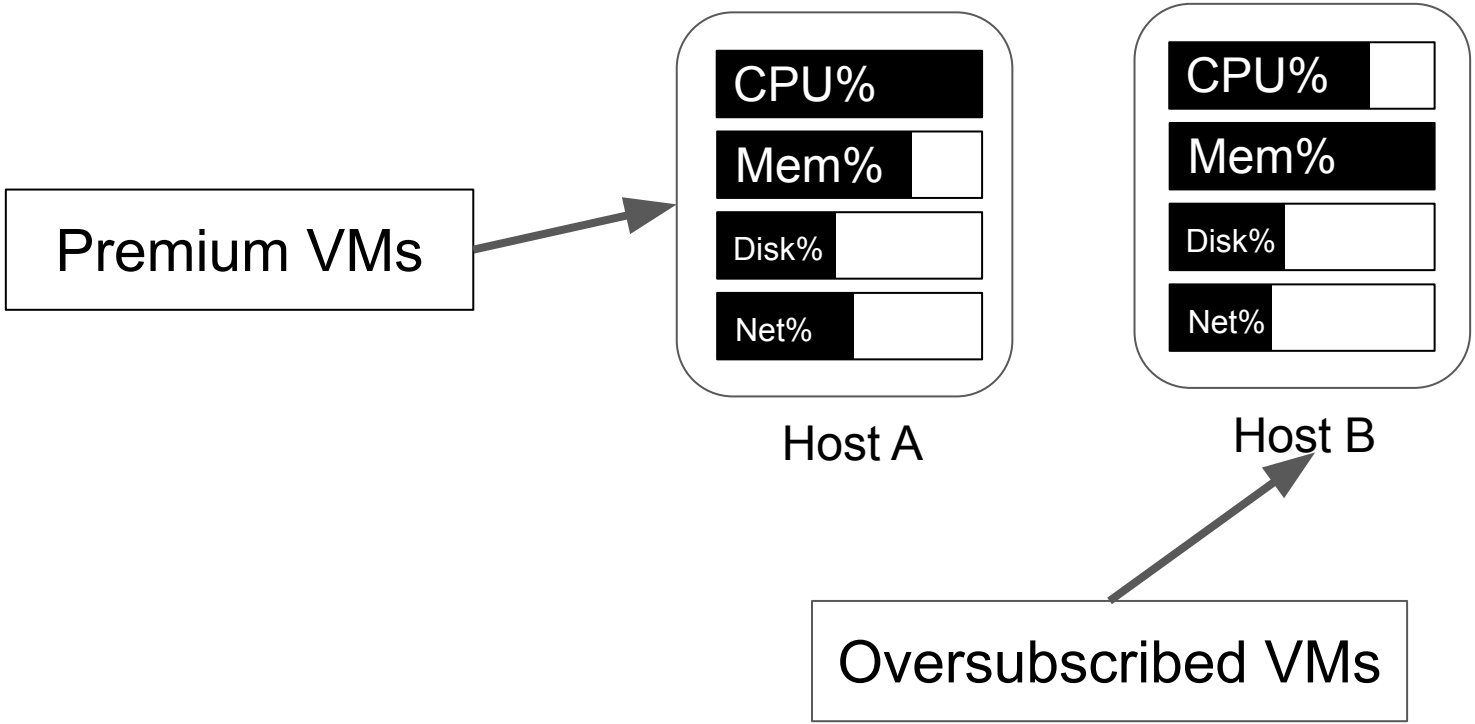


Host A



Host B

SlackVM: **Motivation**



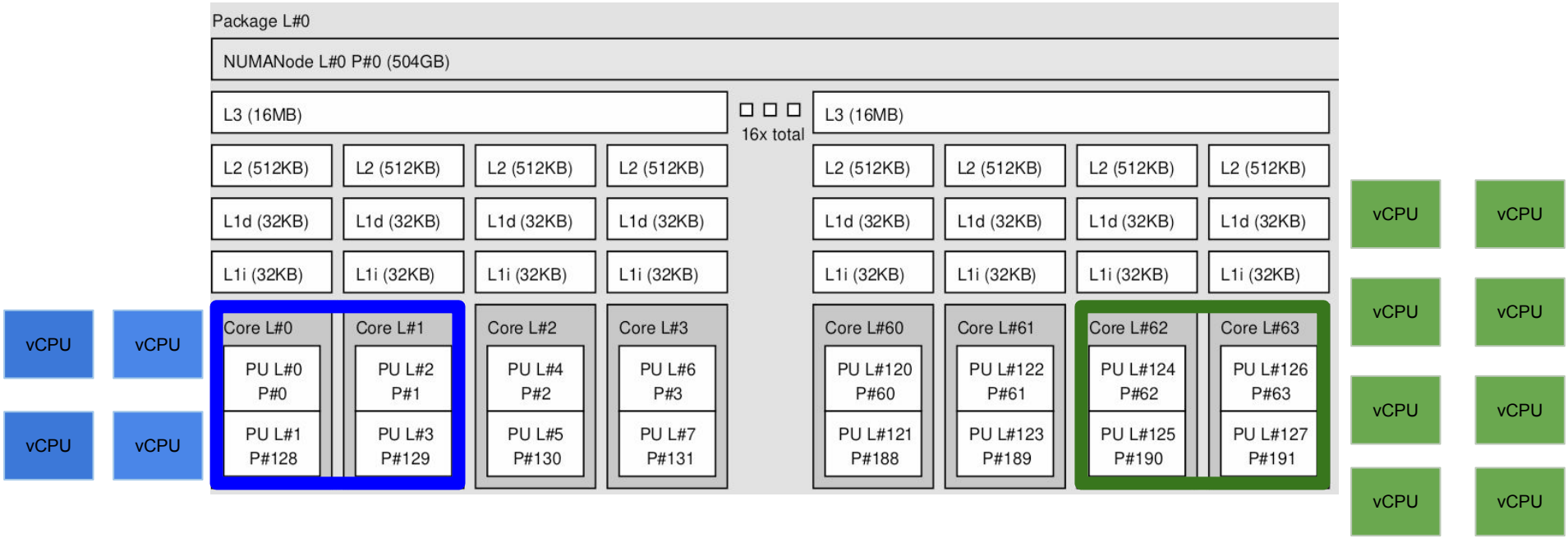
SlackVM: **Implementation**

A single cluster for all oversubscription levels

- How to oversubscribe to multiple levels a given host?
- How to orchestrate resources efficiently?

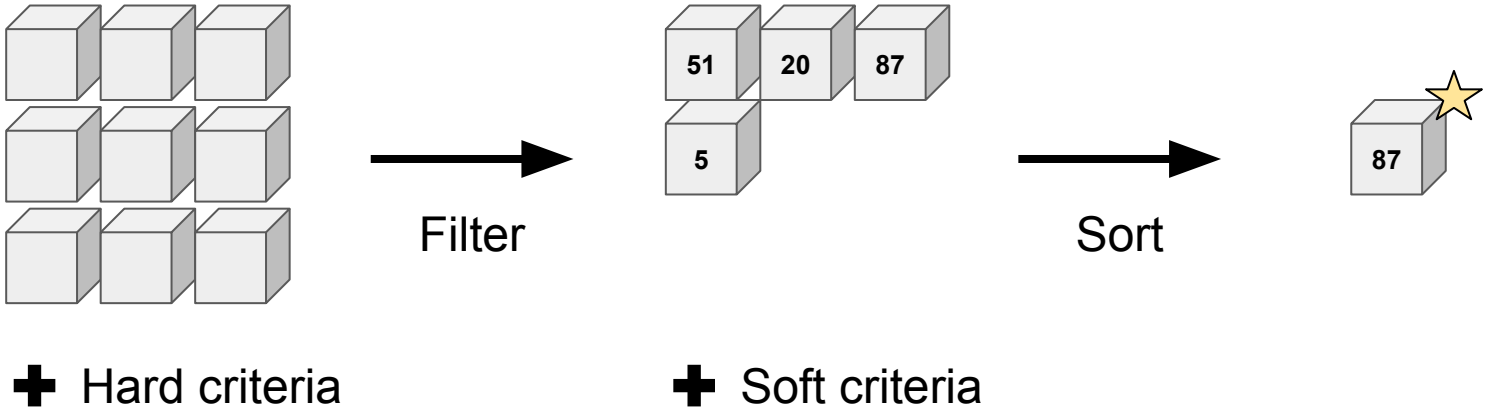
SlackVM: **Implementation**

- How to oversubscribe to multiple levels a given host?



SlackVM: **Implementation**

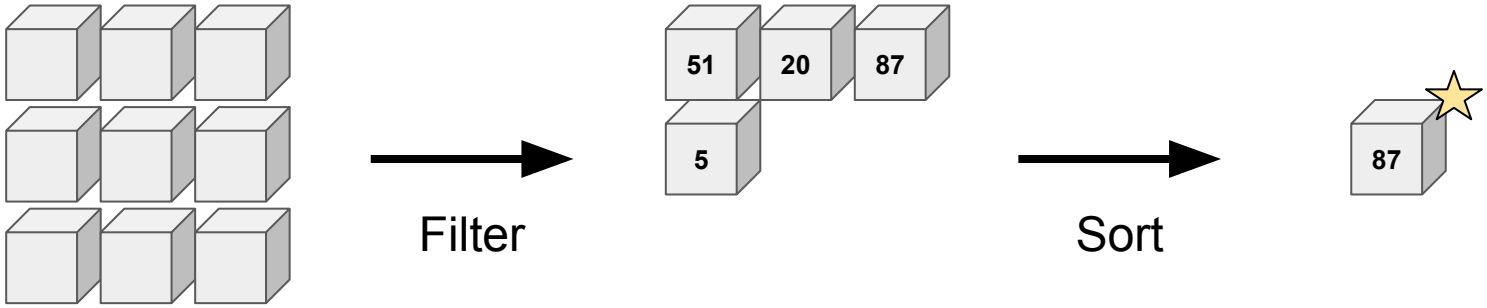
- How to orchestrate resources efficiently?



Cloud orchestrators are score-based

SlackVM: **Implementation**

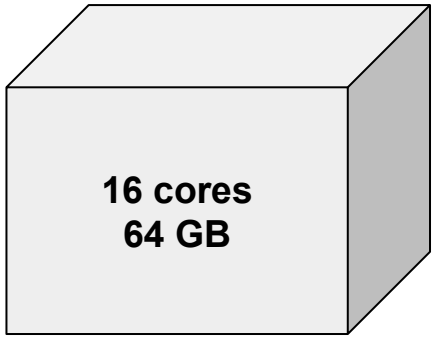
- How to orchestrate resources efficiently?



Objective: Include a soft criteria for the complementarity

SlackVM: **Implementation**

- How to orchestrate resources efficiently?



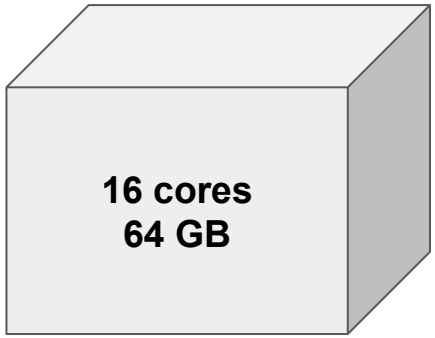
VM1: **1 CPU – 2GB**

VM2: **2 CPU – 8GB**

Servers have a fixed configuration and a dynamic workload

SlackVM: **Implementation**

- How to orchestrate resources efficiently?



VM1: **1 CPU – 2GB**

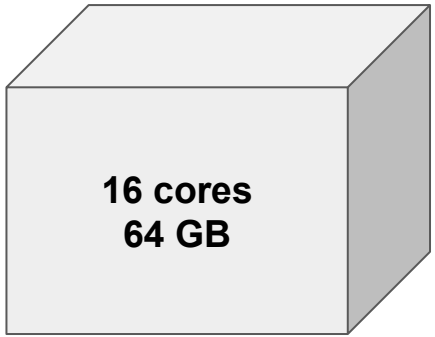
VM2: **2 CPU – 8GB**

3 vCPUs allocated (~20%)
10 GB allocated (~6%)

Servers have a fixed configuration and a dynamic workload

SlackVM: **Implementation**

- How to orchestrate resources efficiently?



VM1: **1 CPU – 2GB**

VM2: **2 CPU – 8GB**

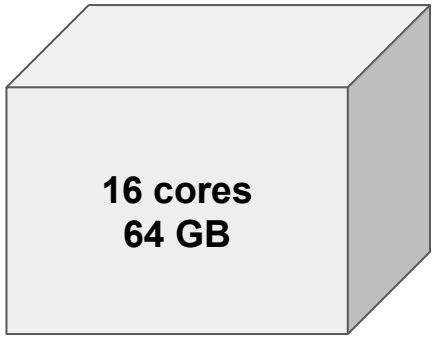
Configuration ratio:
4 GB per core

≠

Workload ratio:
3 GB per core

SlackVM: **Implementation**

- How to orchestrate resources efficiently?



VM1: **1 CPU – 2GB**

VM2: **2 CPU – 8GB**

VM3: **1 CPU – 6GB**

Configuration ratio:
4 GB per core

=

Workload ratio:
4 GB per core

Our goal is to align both ratio

SlackVM: **Implementation**

- How to orchestrate resources efficiently?

Progress toward optimal:

Algorithm 2 Progress towards *target ratio* computation

Input: $configPM$, $allocPM$, vm
Output: $progress$

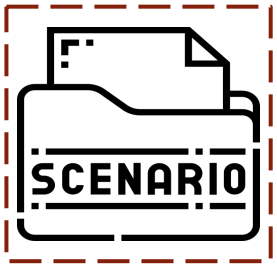
$targetRatio \leftarrow \frac{\text{CONFIGPM}(mem)}{\text{CONFIGPM}(cpu)}$
if $allocPM(cpu) > 0$ **then**
    $currentRatio \leftarrow \frac{\text{ALLOCPM}(mem)}{\text{ALLOCPM}(cpu)}$
    $nextRatio \leftarrow \frac{\text{ALLOCPM}(mem)+\text{VM}(mem)}{\text{ALLOCPM}(cpu)+\text{VM}(cpu)}$
else
    $currentRatio \leftarrow targetRatio$
    $nextRatio \leftarrow \frac{\text{VM}(mem)}{\text{VM}(cpu)}$
end if
$current\Delta \leftarrow |currentRatio - targetRatio|$
$next\Delta \leftarrow |nextRatio - targetRatio|$
$progress \leftarrow current\Delta - next\Delta$
if $progress < 0$ **then**
    $factor \leftarrow 1 + \frac{\text{ALLOCPM}(cpu)}{\text{CONFIGPM}(cpu)}$
    $progress \leftarrow progress \times factor$
end if
return $progress$

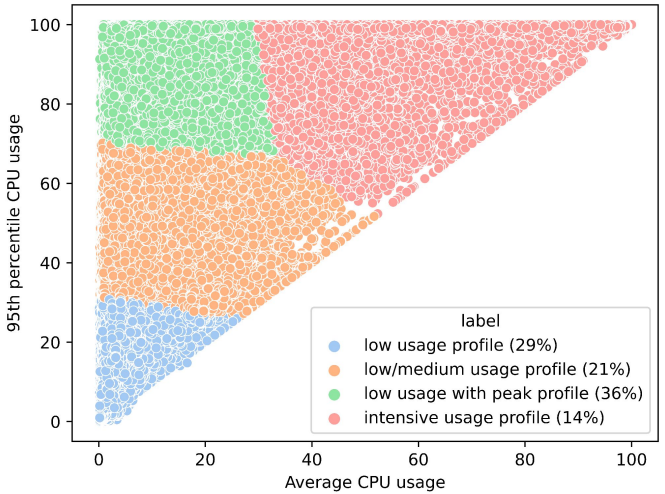
SlackVM: **Evaluation**

- Evaluate realistic IaaS workload on:
  - Real testbed to evaluate performance impact
  - Simulator to evaluate gains at scale

SlackVM: **Evaluation**

- Experiment generated with an extended version of CloudFactory

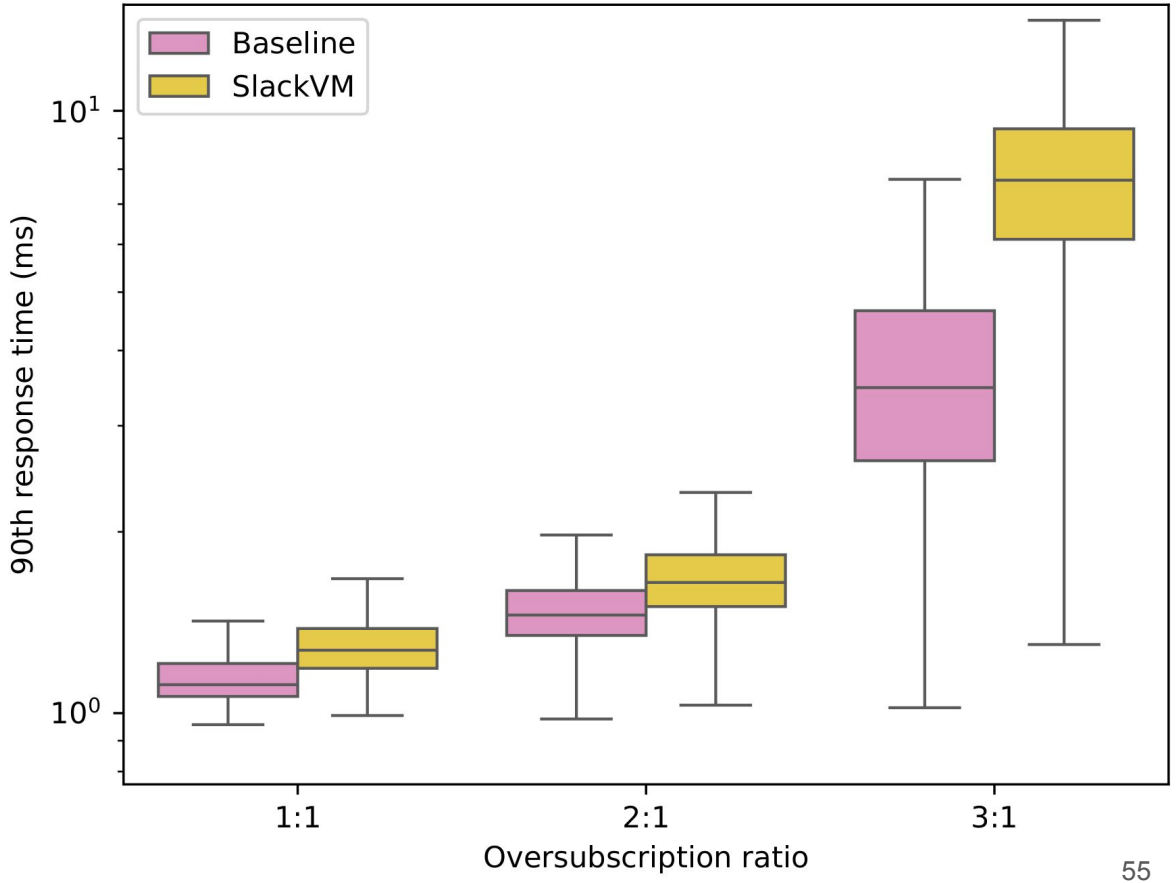




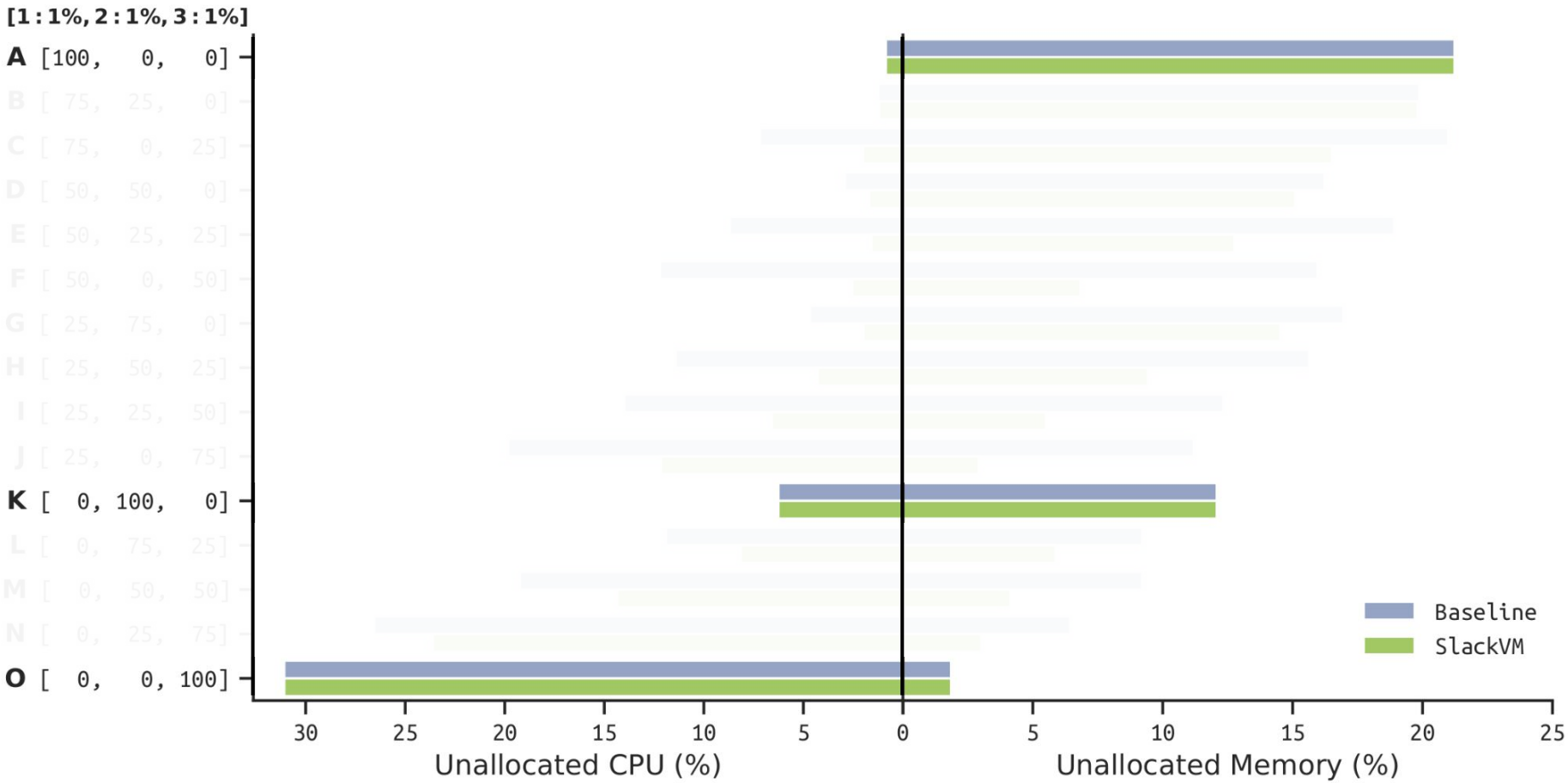
+ impact of oversubscription **levels distribution** (1:1 ; 2:1 ; 3:1)

SlackVM: **Evaluation**

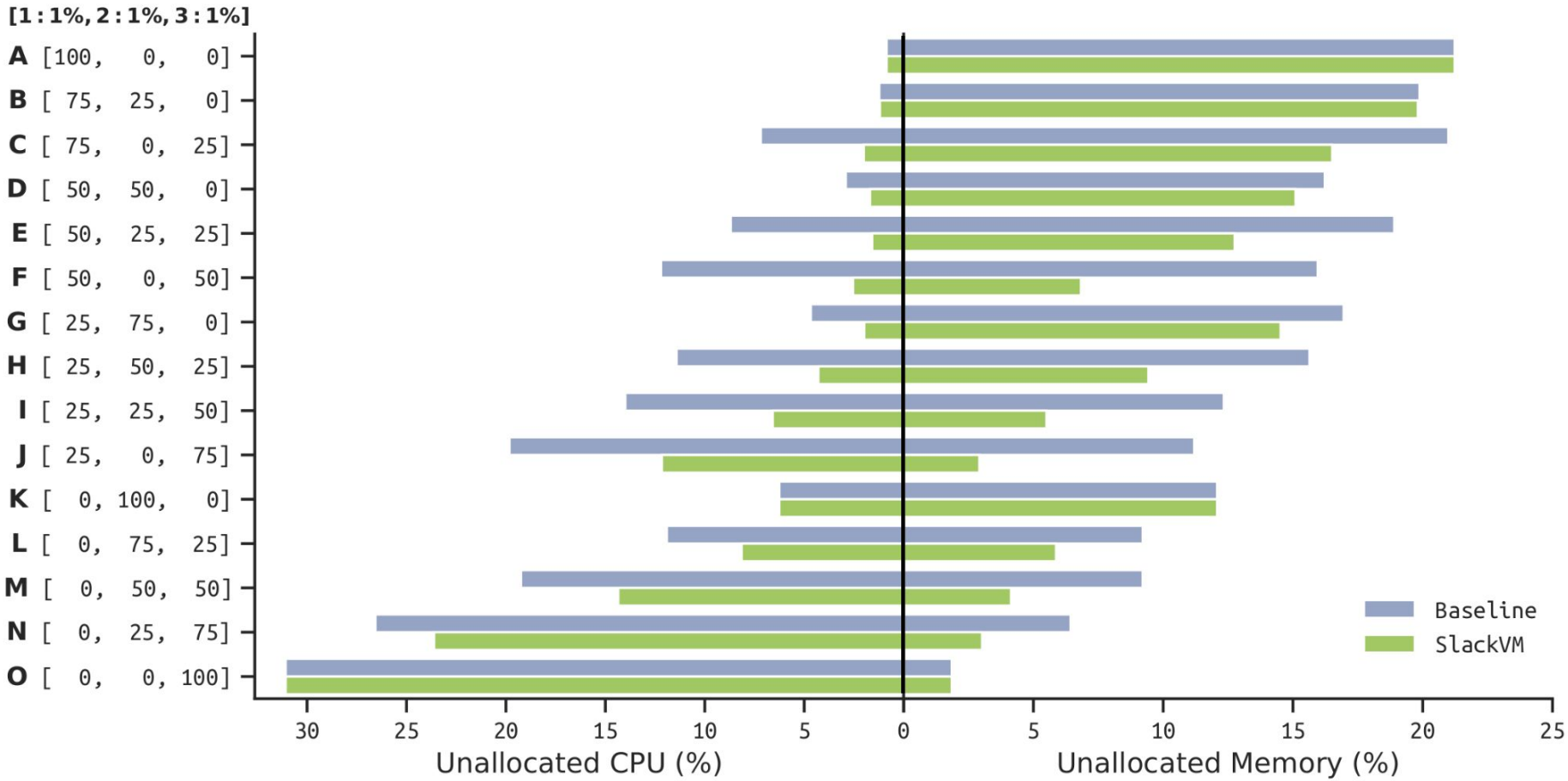
Small impact on low oversubscribed VMs



SlackVM: Evaluation

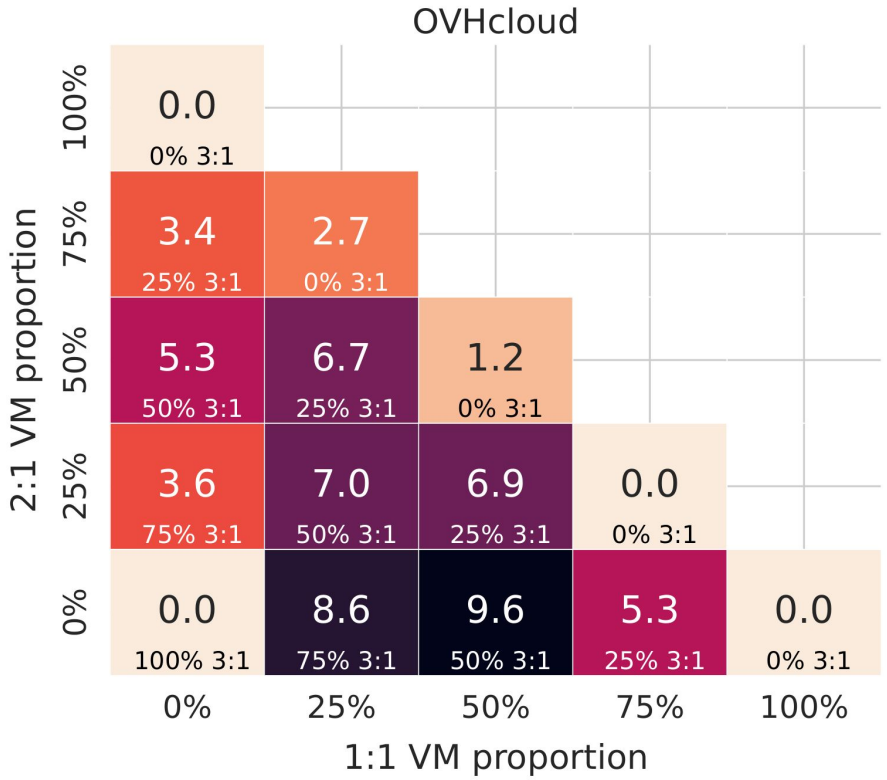


SlackVM: Evaluation



SlackVM: **Evaluation**

Combining oversubscription levels can **save up to 10% physical machines**



Conclusion

Takeaways & perspectives

State of the art

- Ratios at the cluster level
  - Statically defined
- Ratios at the server level
  - Dynamically defined
    - Generic formulas on VM percentile or host standard deviation

Contributions

- Ratios at the cluster level
  - Statically defined
- Ratios at the server level
  - Dynamically defined
    - Generic formulas on VM percentile or host standard deviation
    - **ScroogeVM**[1]: per server formula customization
- **SweetSpotVM**[2]: ratios at the resource level (vCPU)
- **SlackVM**[3]: ratios at the VM level

[1] *IEEE TSUSC* | [2] *CCGrid'24* | [3] *Cluster'24*

Contributions

- Ratios at the cluster level
  - Statically defined
- Ratios at the server level
  - Dynamically defined
    - Generic formulas on VM percentile or host standard deviation
    - **ScroogeVM**[1]: per server formula customization
- **SweetSpotVM**[2]: ratios at the resource level (vCPU)
- **SlackVM**[3]: ratios at the VM level

} **CloudFactory**[4]

[1] *IEEE TSUSC* | [2] *CCGrid'24* | [3] *Cluster'24* | [4] *IC2E'23*

Takeaways

- Cloud resources are **underused**
  - “All clients do not need all their resources all the time”
- Oversubscription is a way to improve server **packing**
- Defining oversubscription ratios **closer to individual usage** is promising

Perspectives

Short-term perspectives

- Multiple **dynamic** ratio
- **GPU** oversubscription
- Linux scheduler **cooperation**

Long-term perspectives

- **Sharing** resources through other mechanisms
- **Reduce** consumption through better feedback
- **Renounce** based on social utility

Backup: Cumulated Gains

Hypothesis 1: all VMs are oversubscribed

Reduced by:

- A dynamic approach (*ScroogeVM*)
- A finer static approach (*SweetSpotVM*)

30% less servers with an average ratio of 1.5:1

Backup: Cumulated Gains

Hypothesis 2: 50% of VMs are oversubscribed

Reduced by:

- A dynamic approach (*ScroogeVM*)
- A finer static approach (*SweetSpotVM*)

15% less servers with an average ratio of 1.5:1

Reduced by: static co-location (*SlackVM*)
5-10% less servers

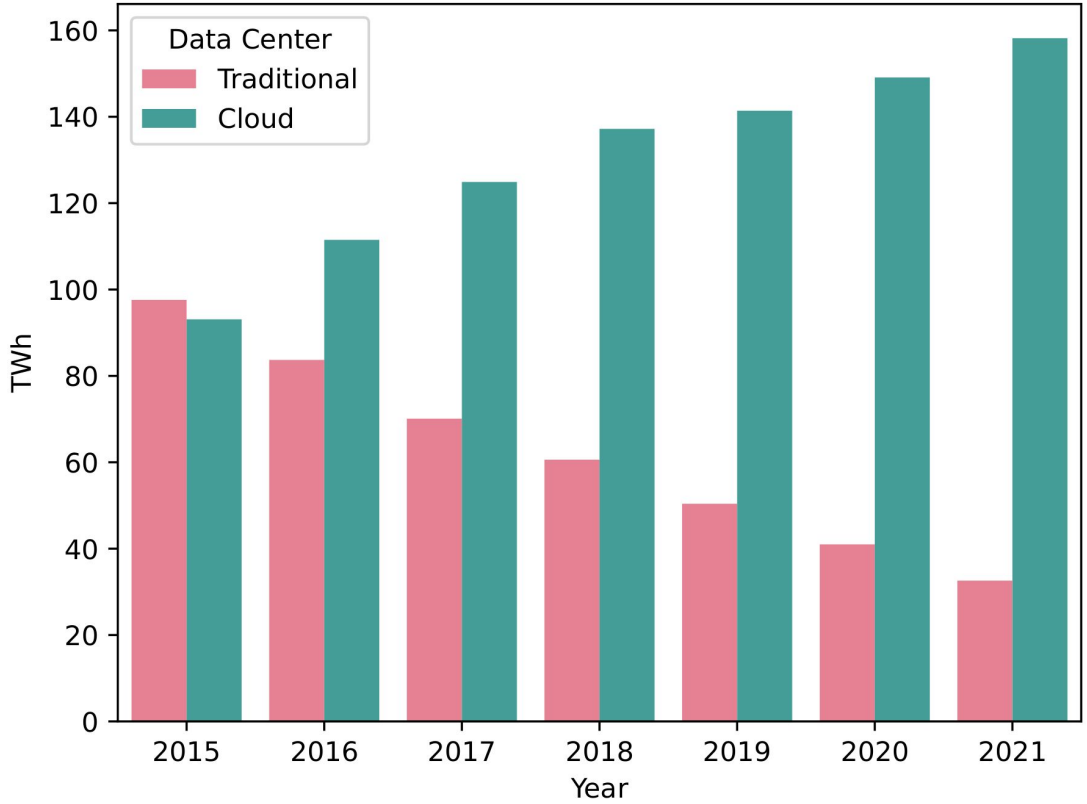
Backup: Complementarity

- **SweetSpotVM** ∧ **SlackVM**
  - Straightforward

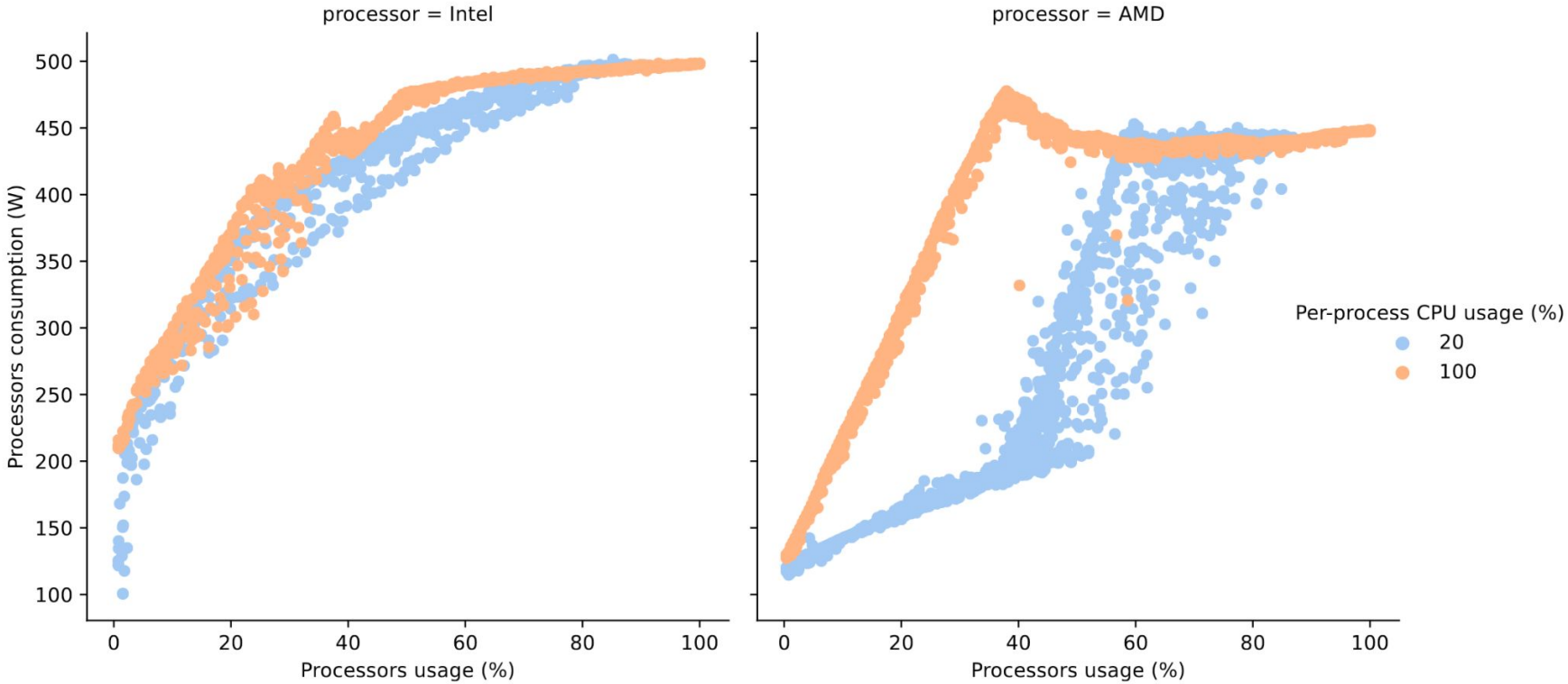
- **ScroogeVM** ∧ (**SweetSpotVM** ∨ **SlackVM**)
  - **Semi-static**:
    - Straightforward
  - Transition to fully **dynamic**
    - RQ1: How to dynamically equilibrate different pools?
    - RQ2: How to adapt orchestration?

Backup: Context

2 different kinds of data centers

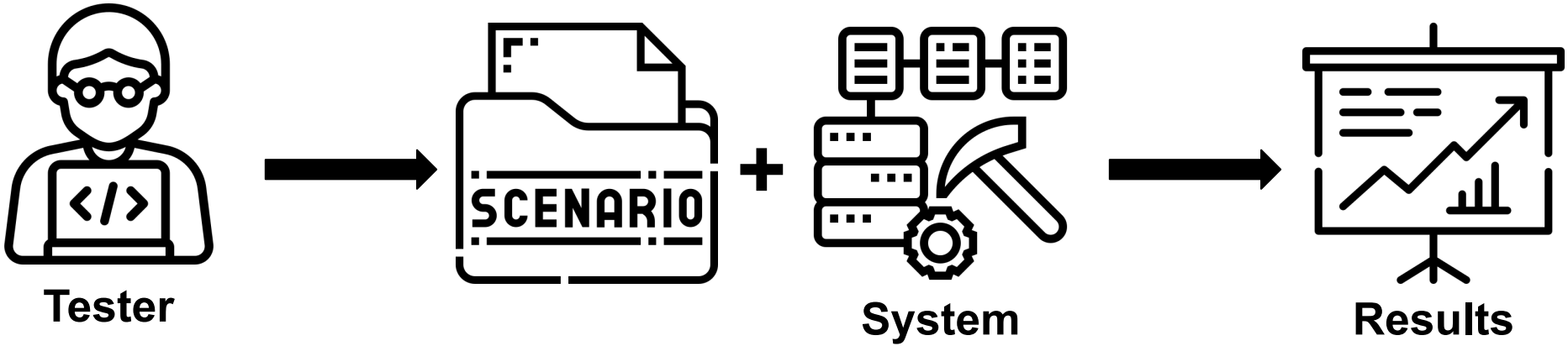


Backup: Paper introduction

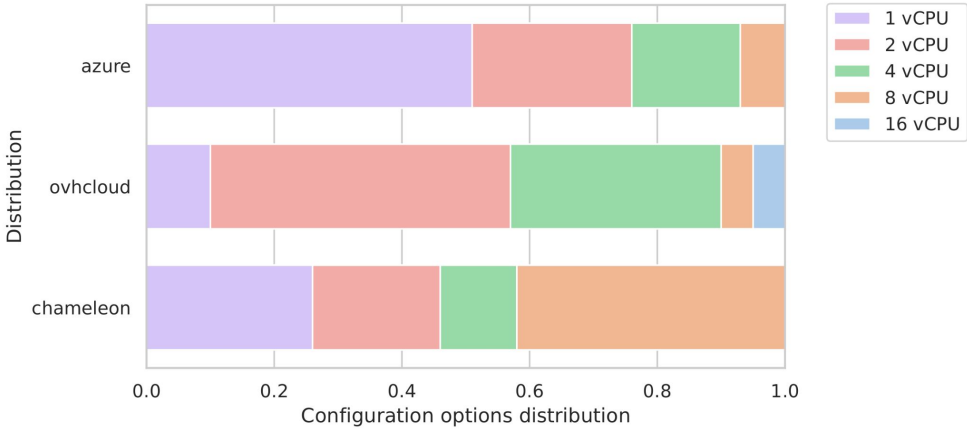


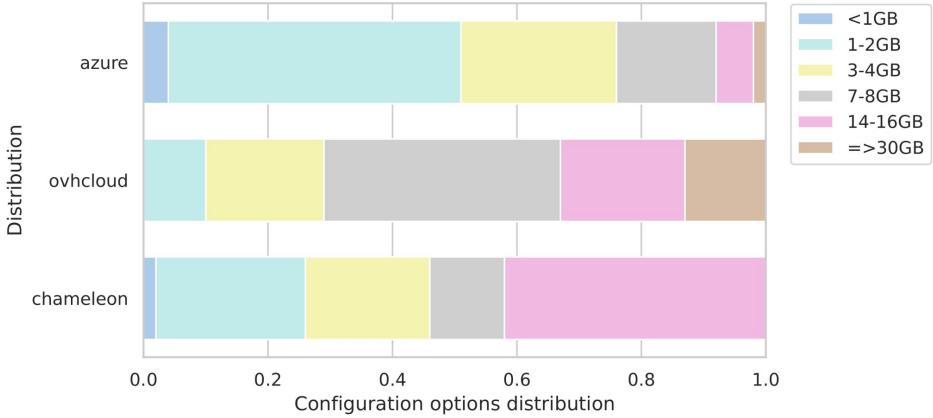
x4 Intel Xeon Gold 6130
x2 AMD EPYC 7662

Backup: CloudFactory



Backup: CloudFactory



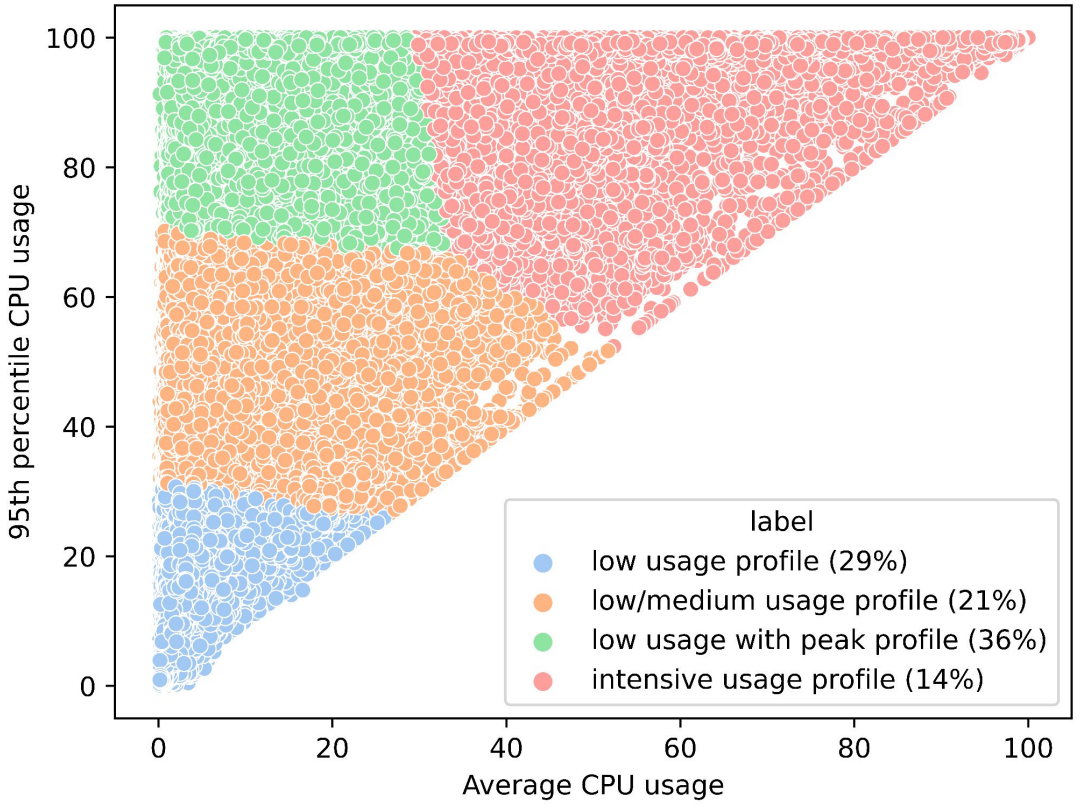


Average VM distribution:

- Azure: 2.25 vCPU, 4GB
- Chameleon: 4.5 vCPU, 9GB

Backup: CloudFactory

- Compute VM usage as profiles
  - CPU bounds
  - Departure/arrival rate
  - Periodicity



Backup: CloudFactory

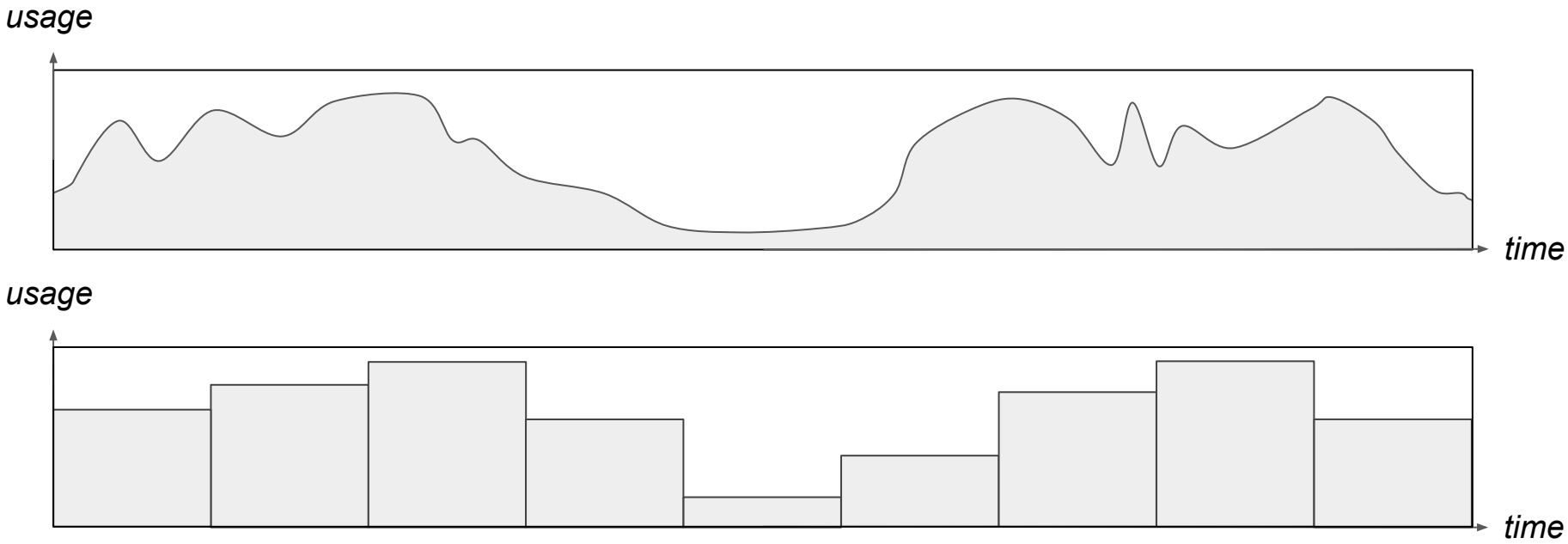
- Generated profiles can be exported
  - Sharing high level statistics may be easier for Cloud Providers than raw datasets

.yaml

```yaml
vm_usage:
  low_usage_profile:
    avg:
      max: 25.9
      min: 0.0
    freq: 0.29
    per:
      max: 31.2
      min: 0.0
    rate:
      arrival: 0.15
      departure: 0.15
      periodicity: 0.004
```

Backup: CloudFactory

- VM usage levels are not static

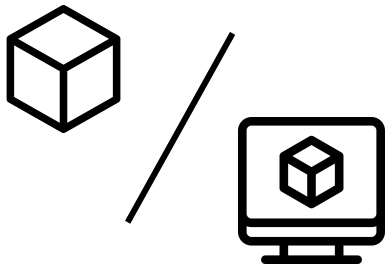


Backup: CloudFactory

- **Bash script**
  - *Deploy a simple libvirt based node*
    - *Hosting heterogeneous VM*

- **CBTool script**
  - *More complex deployments*
    - *Private ones : OpenStack, libvirt*
    - *Public ones : AWS, GCP*

- **CloudSimPlus scenario**
  - *Simulate cloud nodes*



Backup: CloudFactory

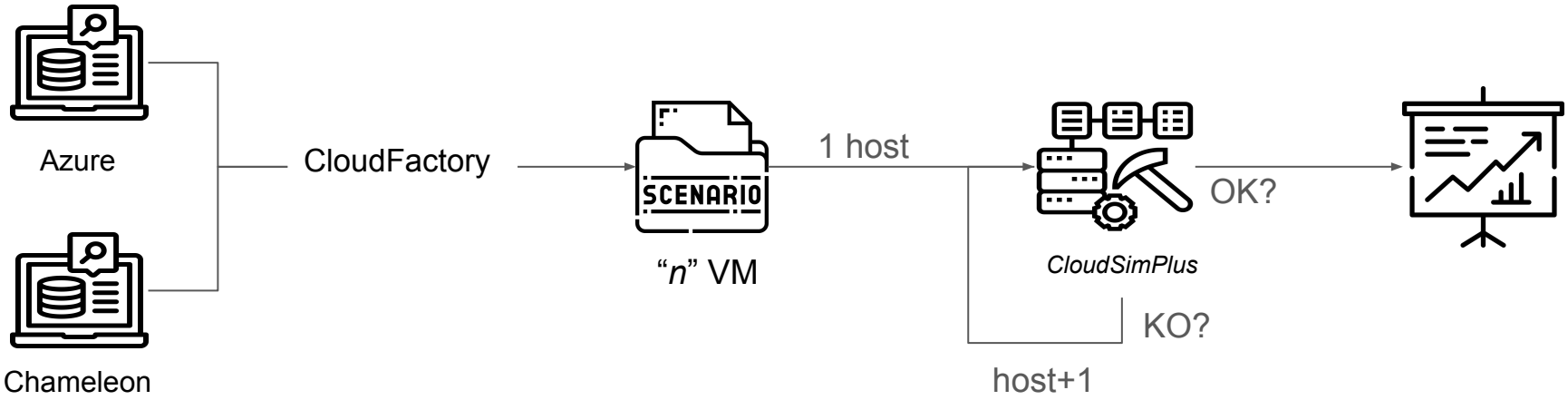
- Convert CPU usage to workload level : fully customizable

```
vm_workloads:
  workloads:
     stressng:
       constraint:
          freq: 0.20
       acronyms:
         §value: "math.ceil(§target) + 1"
     command: "§ssh §name ‘stress-ng --timeout §time --cpu 0 -l §value‘"
```

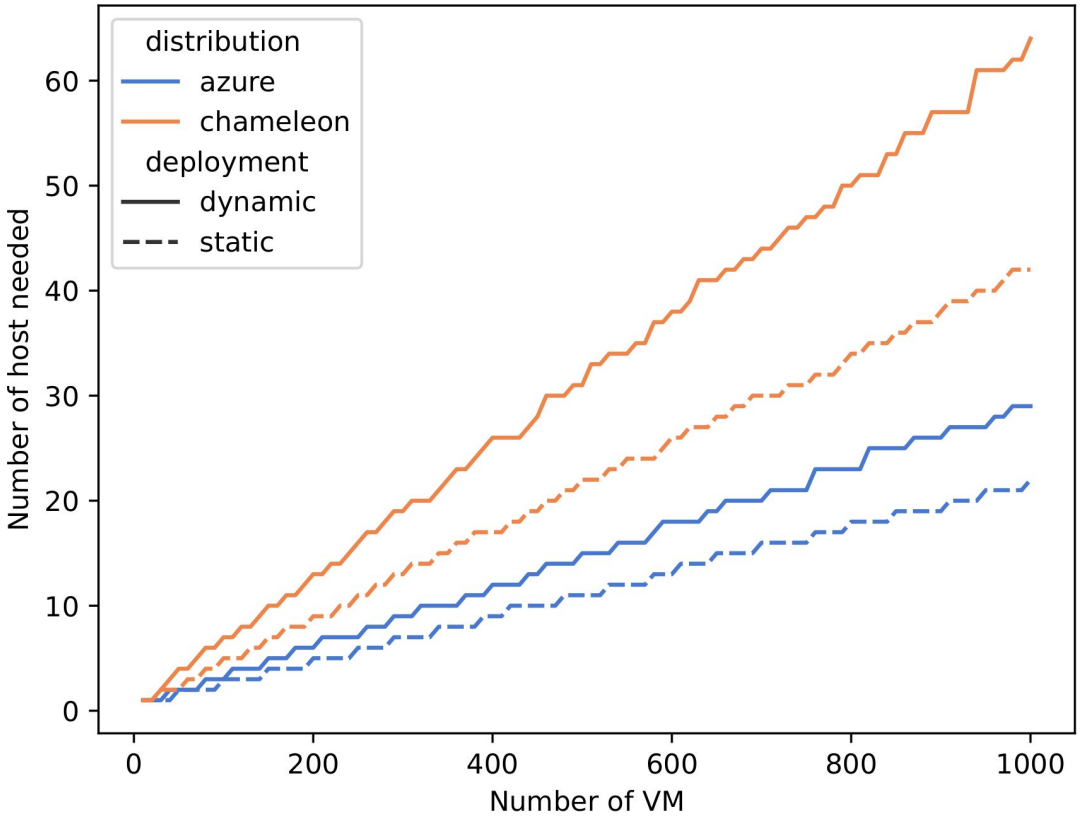
Backup: CloudFactory

With a “*n*” VM hosting objective, how many servers do I need?

- **Baseline**:
  - VMs are hosted statically
- **Heavy-tail**:
  - Departure and arrival rates are taken into account



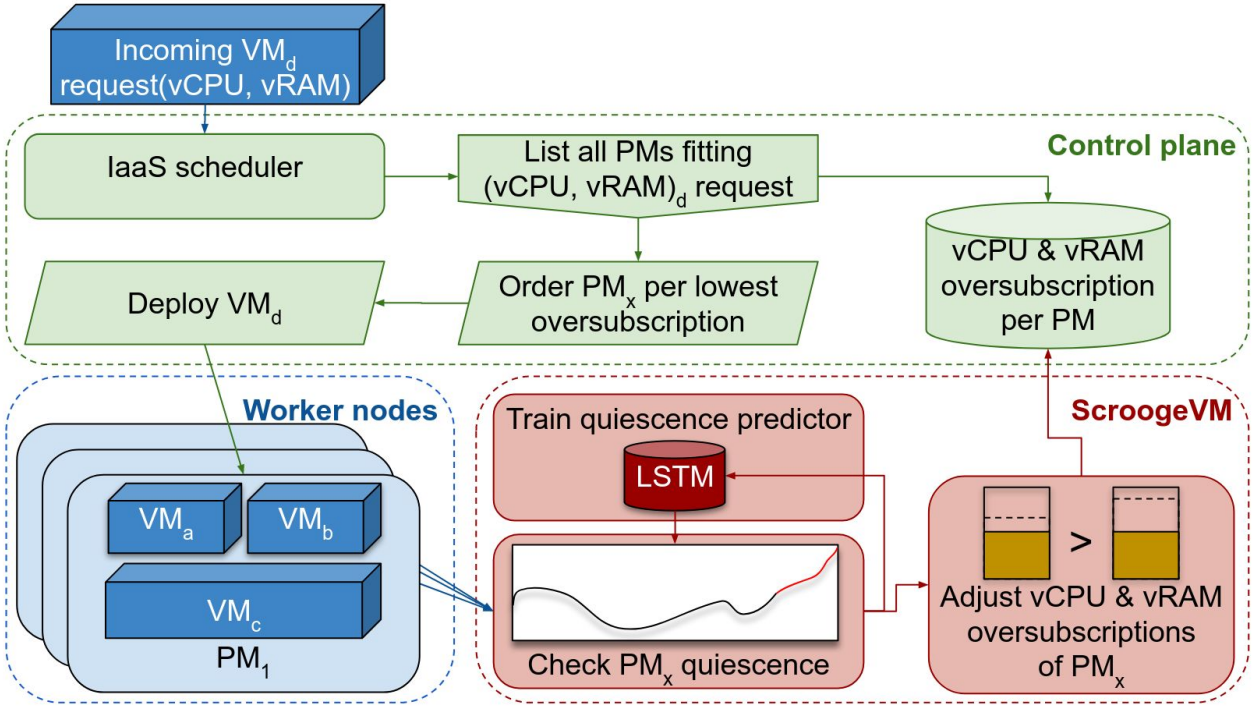
Backup: CloudFactory



Instantaneous availability server cost

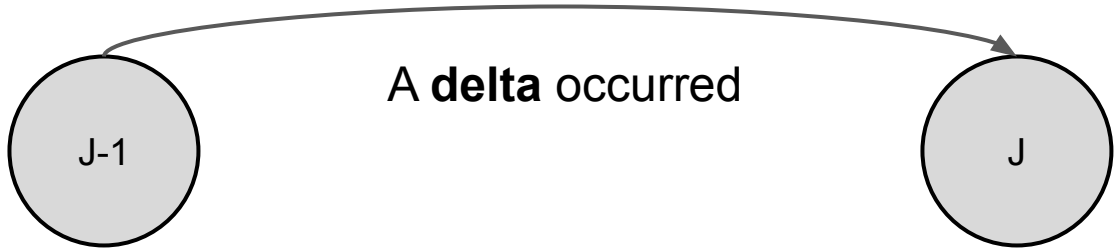
- Up to 30% for Azure
- Up to 52% for Chameleon

Backup: ScroogeVM



Backup: ScroogeVM

Details on available misprediction computation:



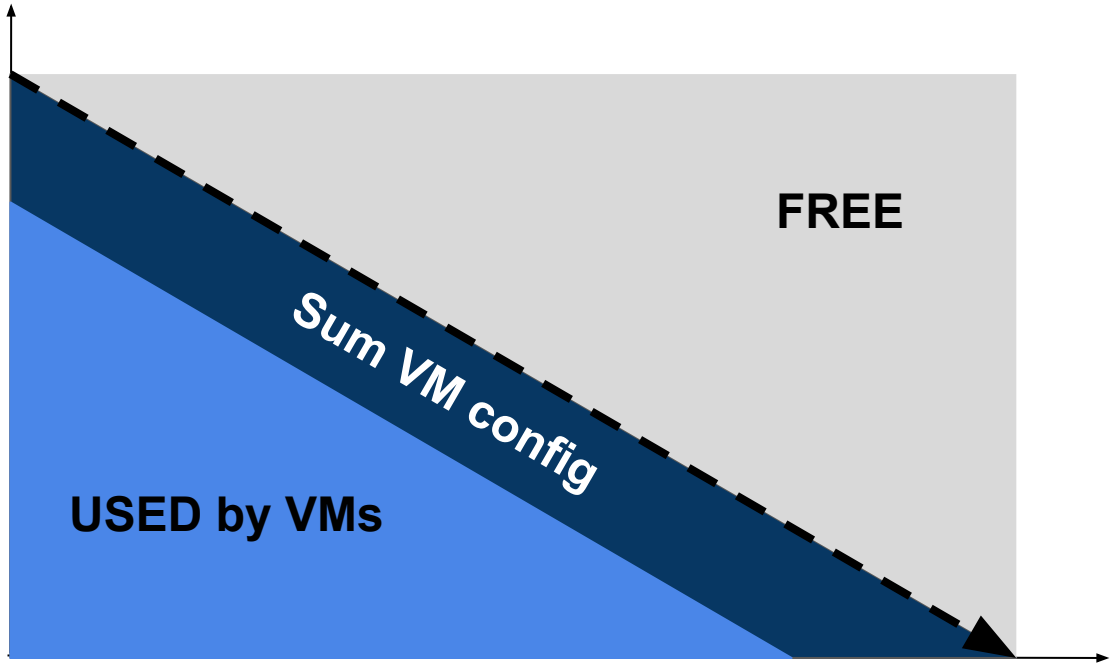
A **prediction** is emitted

A **groundtruth** was mesured

misprediction = (**groundtruth - prediction) - delta**

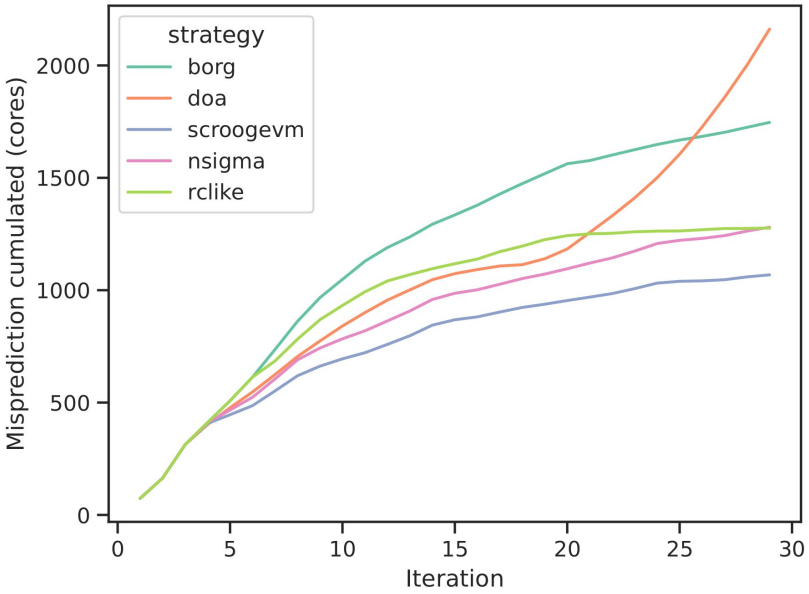
Backup: ScroogeVM

Available misprediction weight **decreases** on a decreasing workload

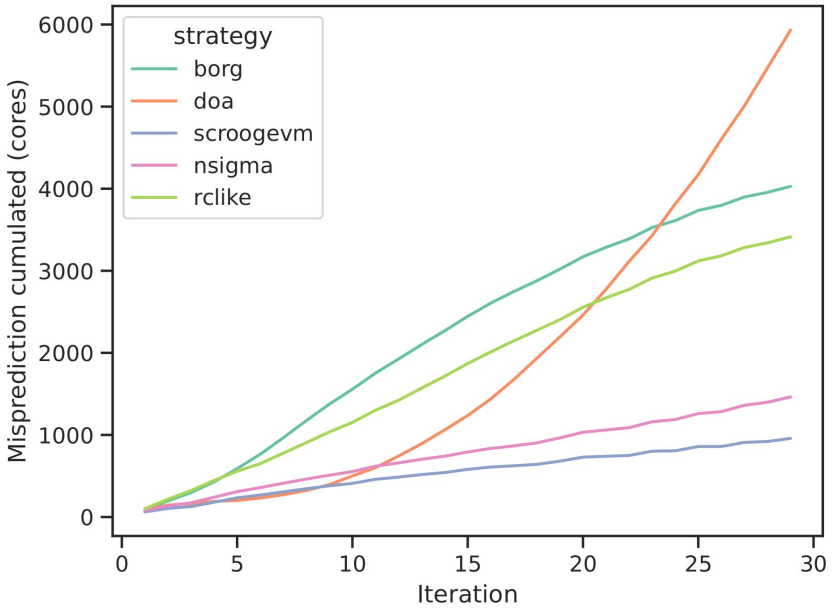


Backup: ScroogeVM

Misprediction: Compared what was evaluated as available at J-1 to what is currently available at J



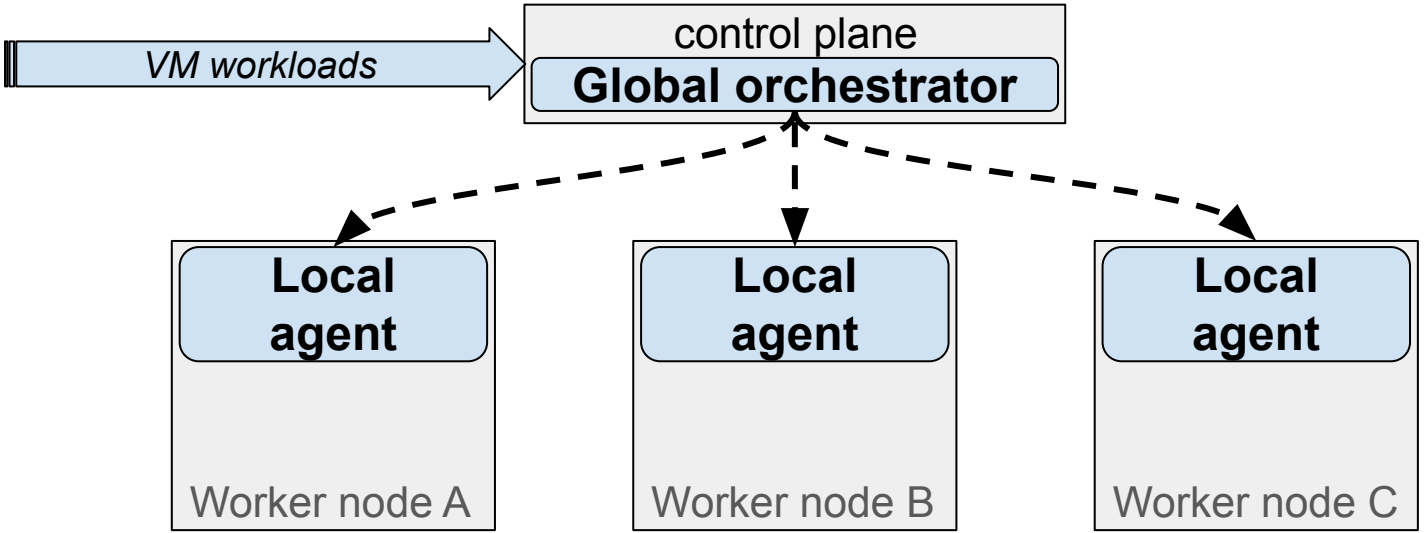
Misprediction on an decreasing workload



Misprediction on an increasing workload

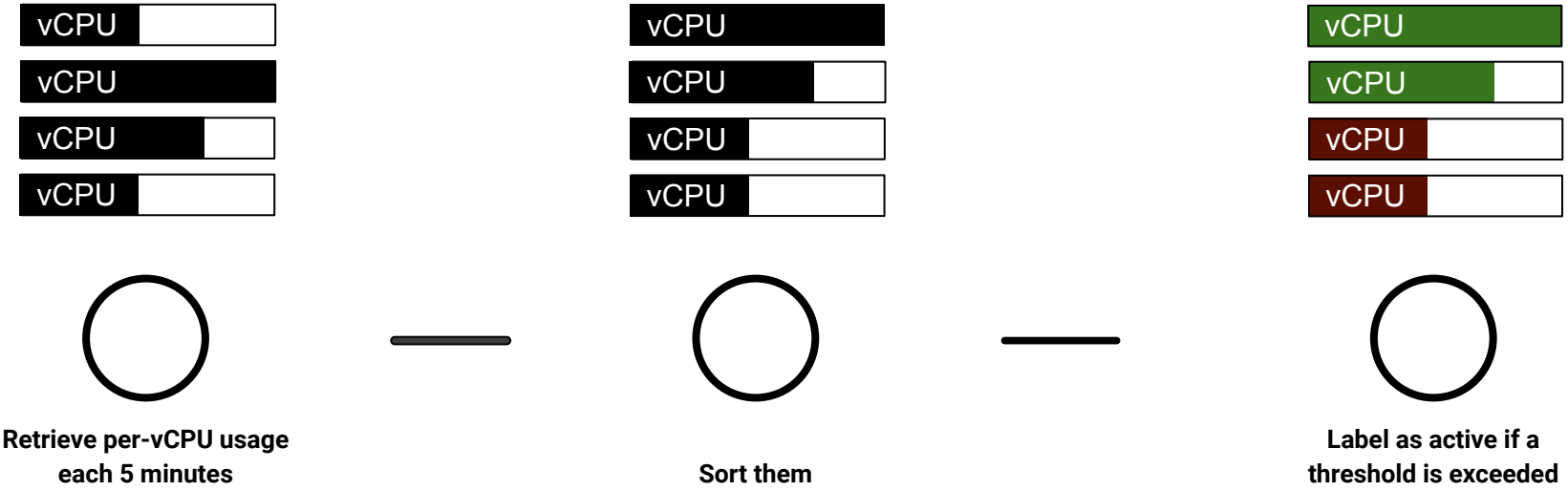
Backup: SweetSpotVM

- On IaaS orchestration components

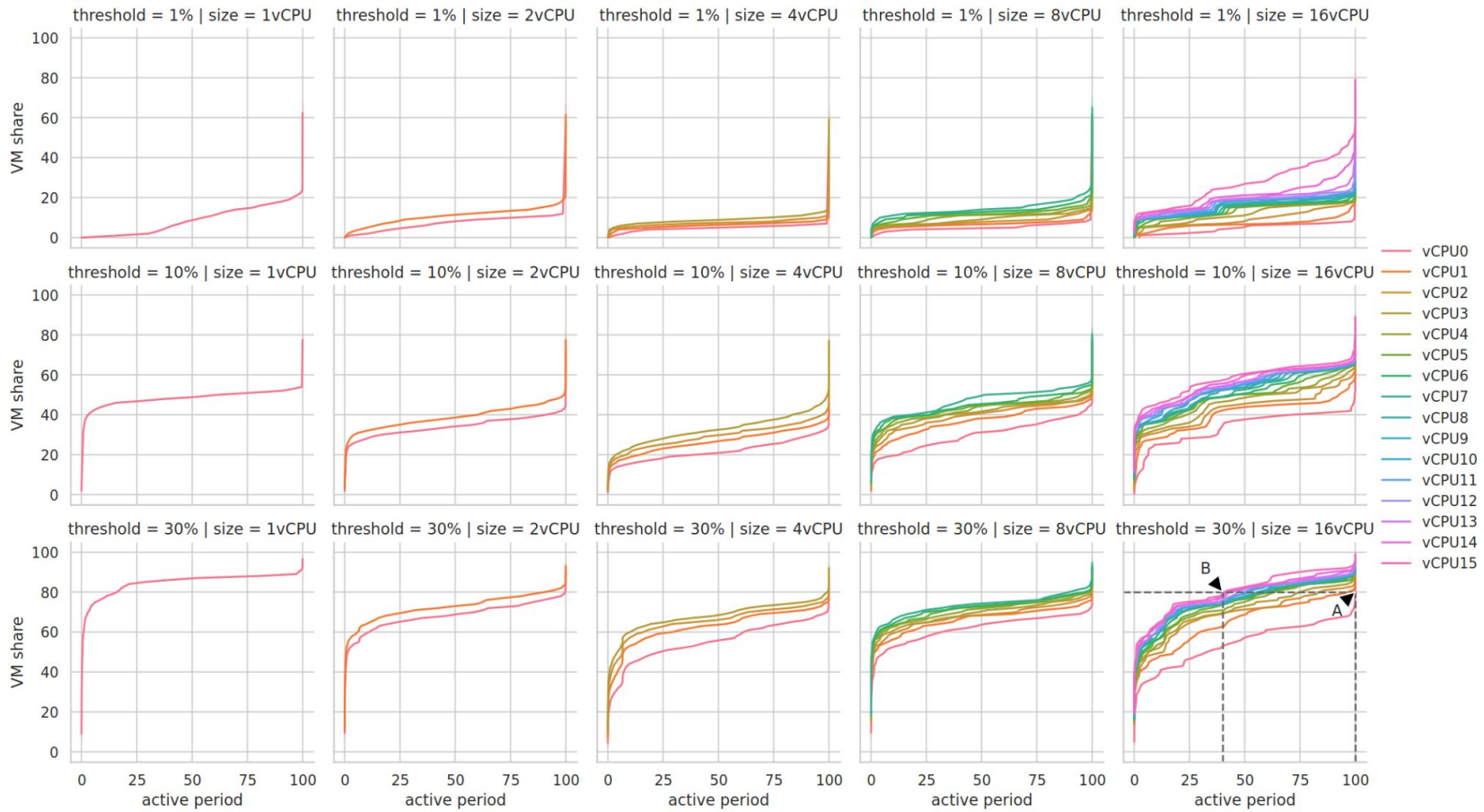


Backup: SweetSpotVM

- Is the typical VM workload distributed?



Backup: SweetSpotVM



Backup: SweetSpotVM

- How to oversubscribe to multiple levels a given host?
  - Extend NUMA distance

Algorithm 1 Distance (Δ) computation between 2 cores

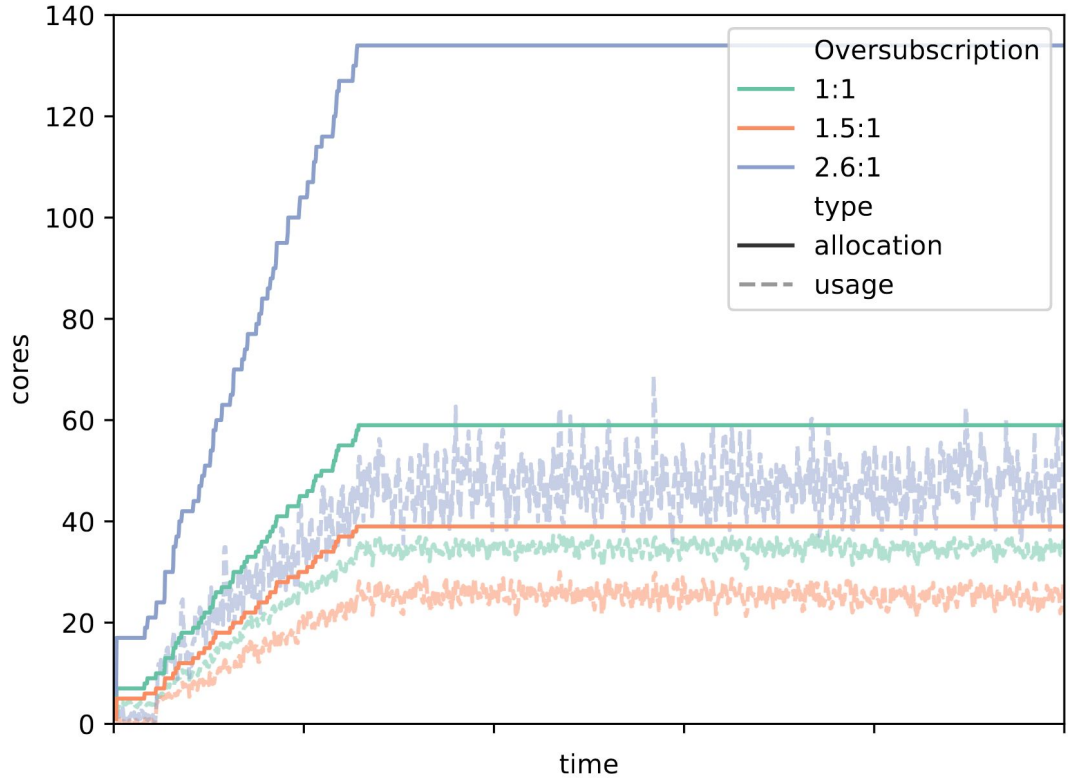
Input: $core_0$, $core_1$
Output: Δ

1: $\Delta \leftarrow 0$
2: **for** <cachelevels> **do**
3: **if** LEVEL$(core_0)$ == LEVEL$(core_1)$ **then**
4: **return** Δ
5: **end if**
6: $\Delta \leftarrow \Delta + 10$
7: **end for**
8: **return** Δ + NUMA-DISTANCE$(core_0, core_1)$

Backup: SweetSpotVM

- Complementary considerations
  - Dynamic sizing
  - Enhance VM heterogeneity
    - **if** “no more than 1 vCPU per CPU” **is True**
      then “no more than 2 vCPU per CPU” **is True**

Backup: SweetSpotVM



Backup: Publications

- Journal:
  - **TSUSC 2023:** *ScroogeVM: Boosting Cloud Resource Utilization with Dynamic Oversubscription*, **P. Jacquet**, T. Ledoux, R. Rouvoy
- Conferences:
  - **IC2E 2023**: *CloudFactory: An open toolkit to generate production-like workloads for cloud infrastructures*, **P. Jacquet**, T. Ledoux, R. Rouvoy
  - **CCGRID 2024**: *SweetSpotVM: Oversubscribing CPU without Sacrificing VM Performance*, **P. Jacquet**, T. Ledoux, R. Rouvoy
  - **CLUSTER 2024:** *SlackVM: Packing Virtual Machines in Oversubscribed Cloud Infrastructures,* **P. Jacquet**, T. Ledoux, R. Rouvoy