Unleashing True Utility Computing with Quicksand

Zain Ruan[*] Shihang Li[‡] Kaiyan Fan* Marcos K. Aguilera[†] Adam Belay[*]

Seo Jin Park Malte Schwarzkopf[‡]

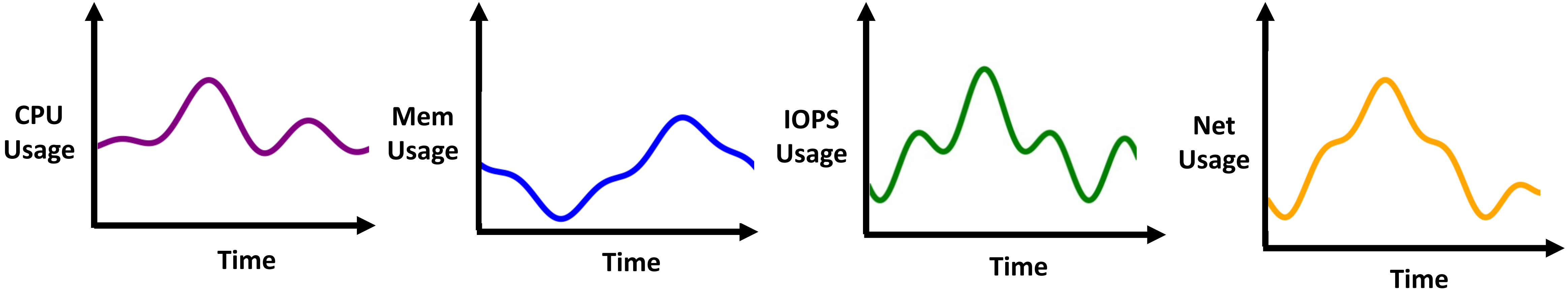
*MIT CSAIL †VMware Research ‡Brown University

Inefficiency 1: resource overprovisioning

- Today’s datacenters are inefficient (Borg [EuroSys’ 20], AlibabaTraca [BigData’ 17])

Inefficiency 1: resource overprovisioning

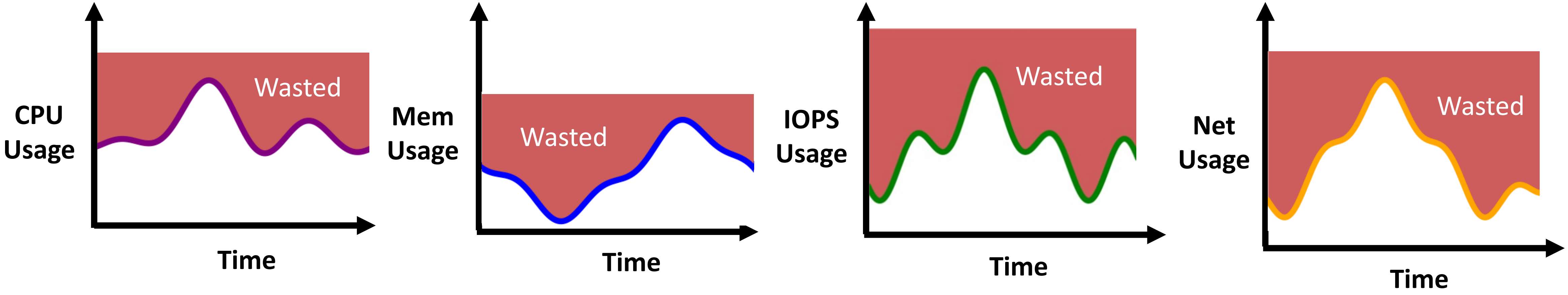
- Today's datacenters are inefficient (Borg [EuroSys' 20], AlibabaTraca [BigData' 17])
- Cloud apps have varying resource consumption.



Inefficiency 1: resource overprovisioning

- Today’s datacenters are inefficient (Borg [EuroSys’ 20], AlibabaTraca [BigData’ 17])
- Cloud apps have varying resource consumption.

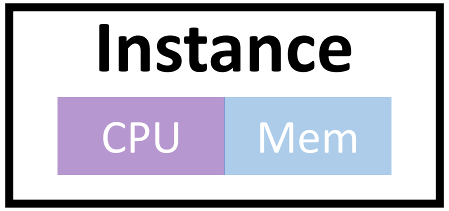
➢Avoid running out of resources → overprovisioning



Inefficiency 1: resource overprovisioning

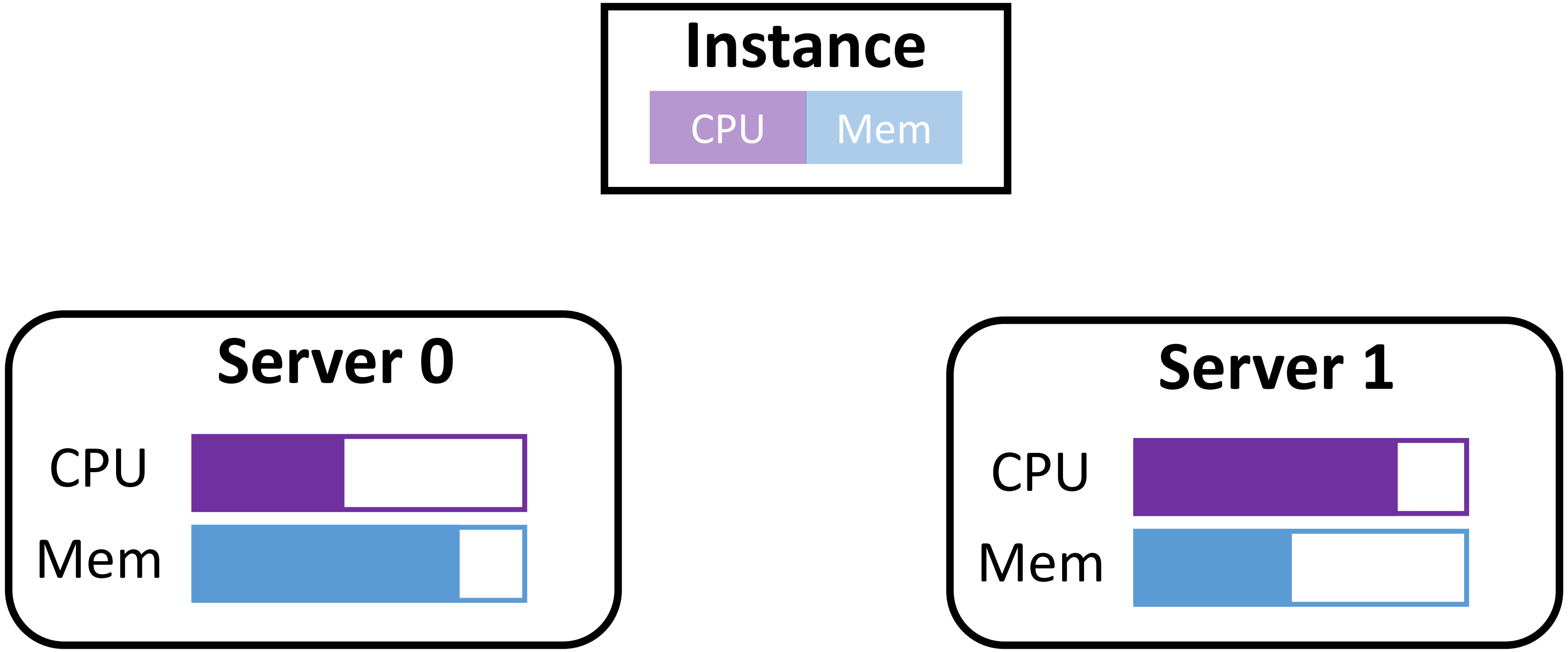
- Today’s datacenters are inefficient (Borg [EuroSys’ 20], AlibabaTraca [BigData’ 17])
- Cloud apps have varying resource consumption.

➢Avoid running out of resources → overprovisioning



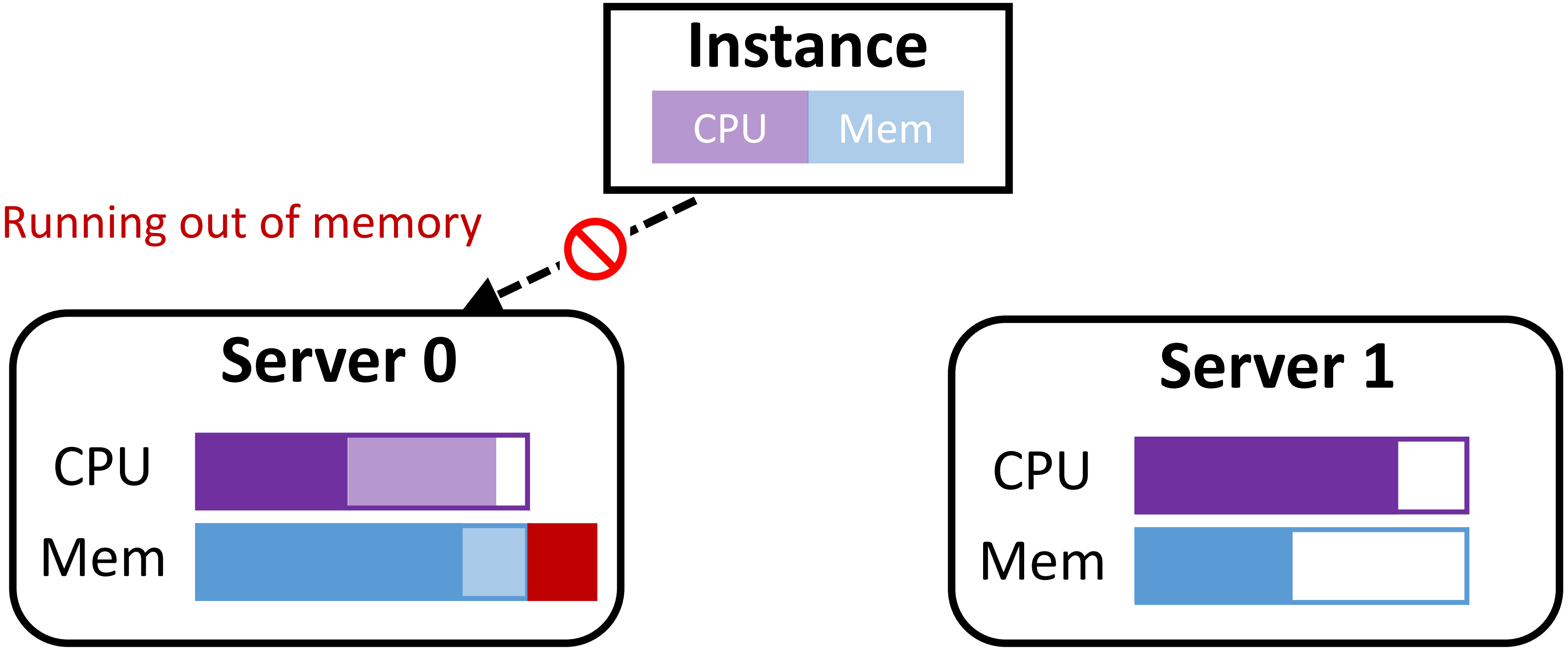
Inefficiency 2: resource stranding

- Try to binpack the instance into available physical machines.



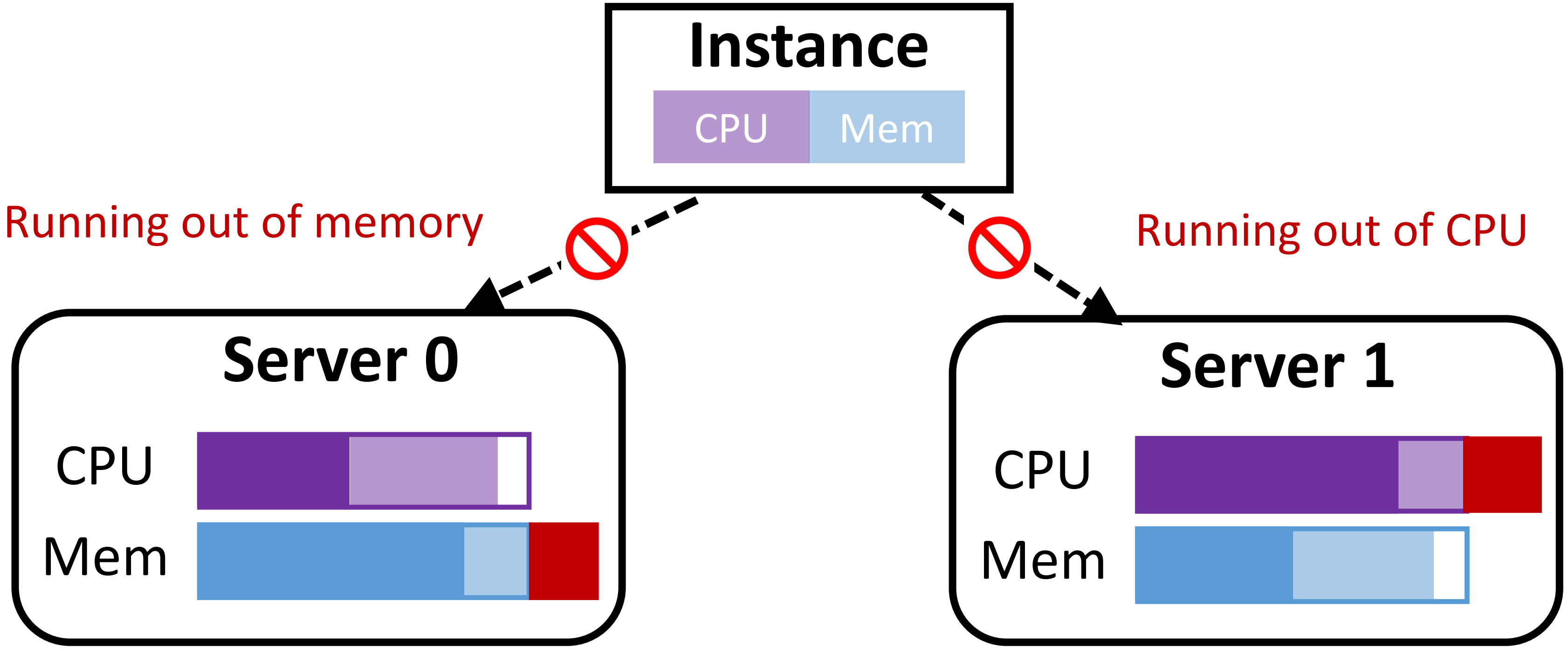
Inefficiency 2: resource stranding

- Try to binpack the instance into available physical machines.
  - ➢Cannot fit into either machine



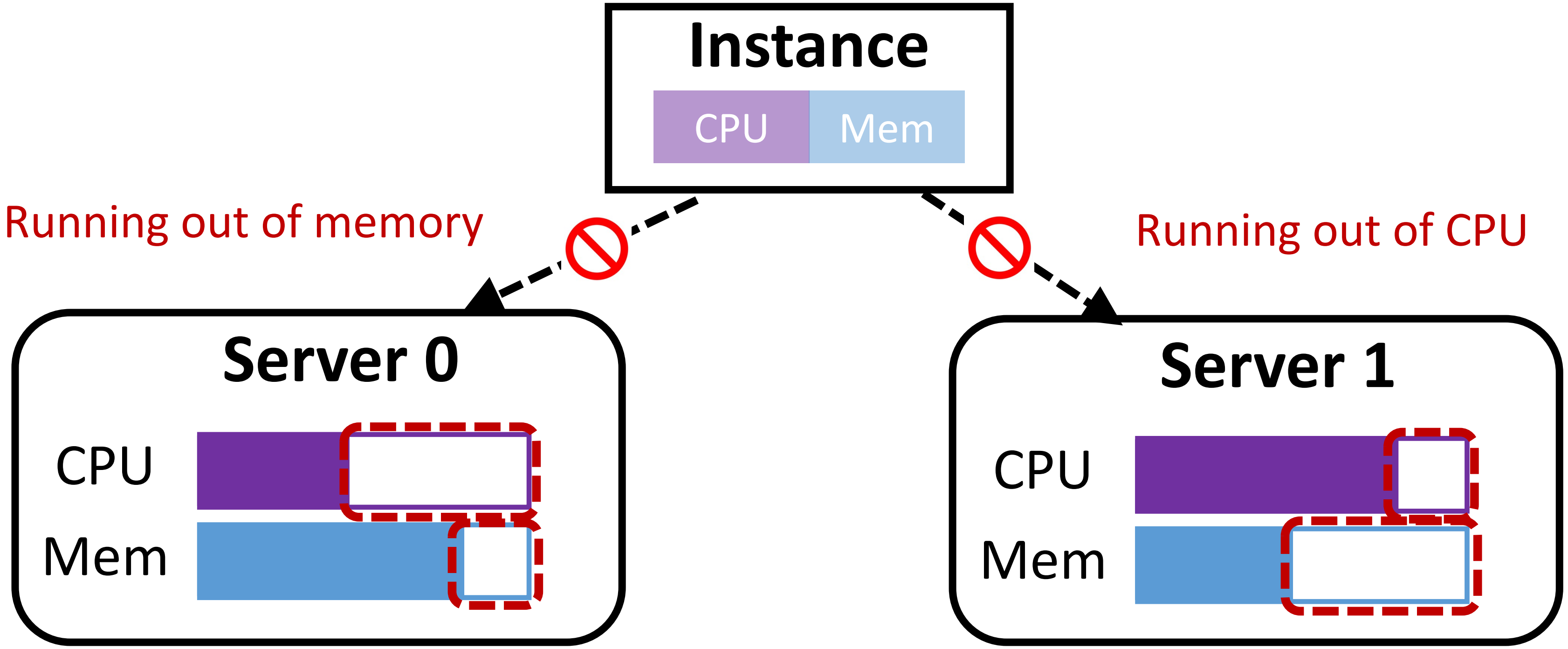
Inefficiency 2: resource stranding

- Try to binpack the instance into available physical machines.
  - ➢Cannot fit into either machine



Inefficiency 2: resource stranding

- Try to binpack the instance into available physical machines.
  - ➢Cannot fit into either machine → Resource stranding

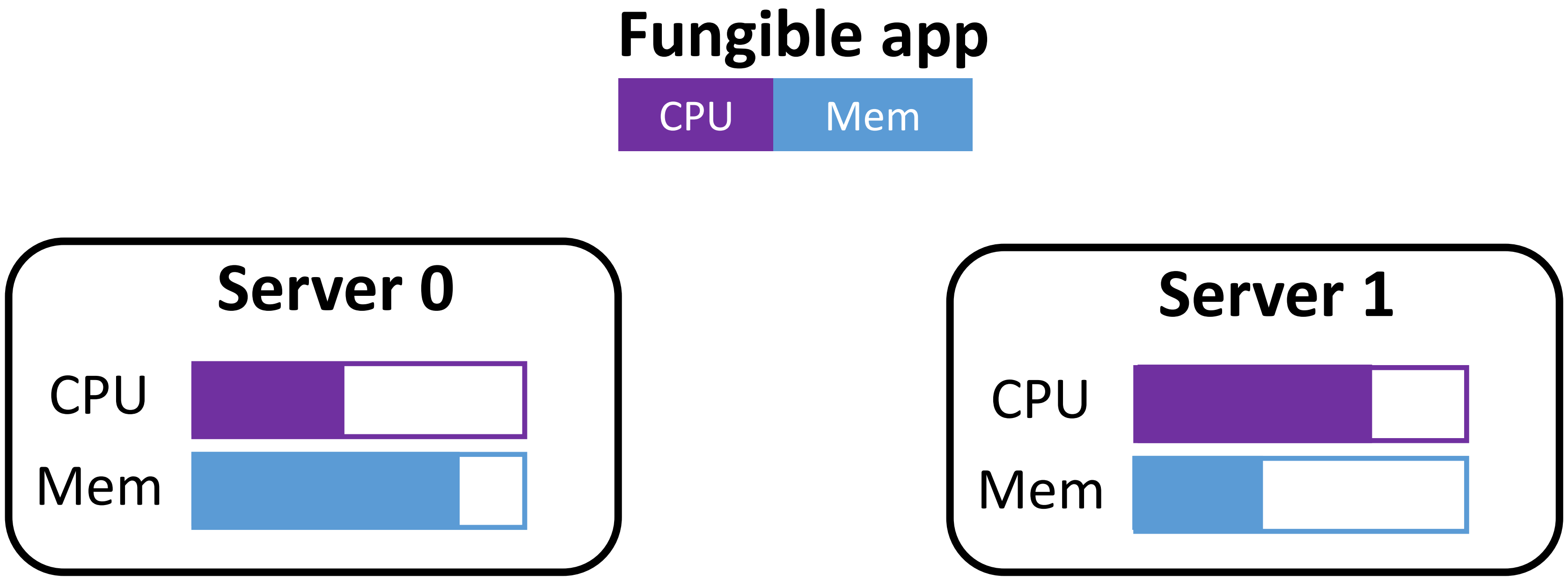


Our approach: resource fungibility

- We advocate for ***fungible*** applications
  - that can use resources wherever they are in the cluster.

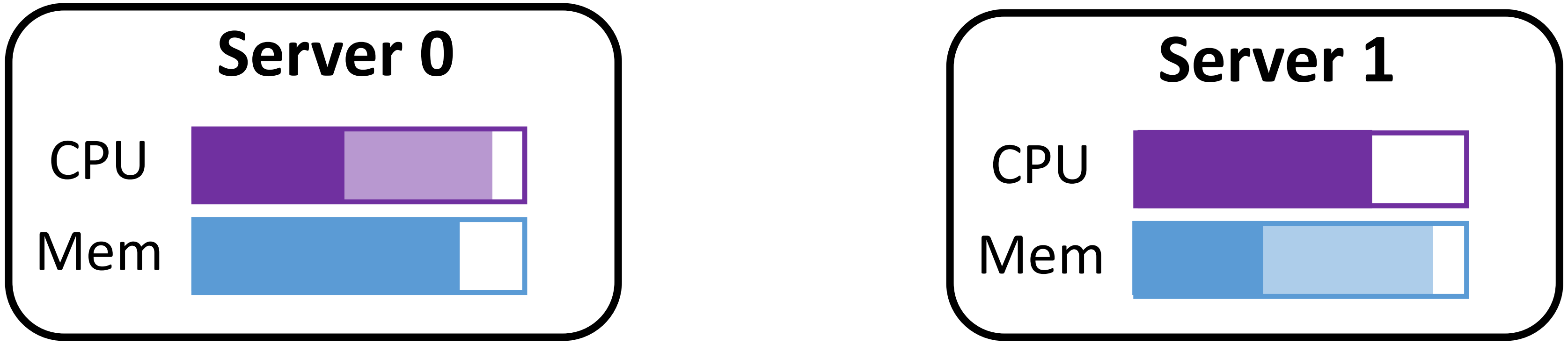
Our approach: resource fungibility

➢ We advocate for ***fungible*** applications
  - that can use resources wherever they are in the cluster.



Our approach: resource fungibility

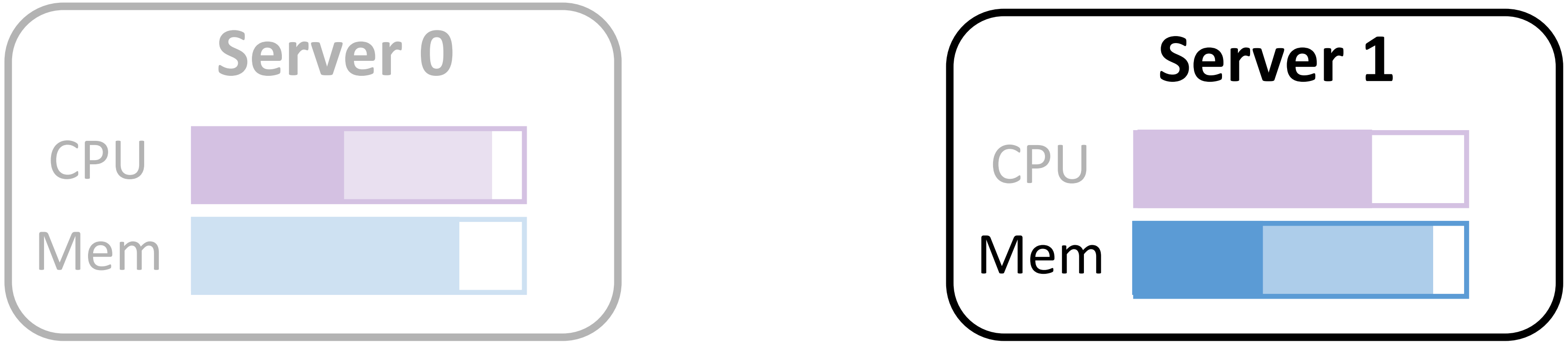
- We advocate for ***fungible*** applications
  - that can use resources wherever they are in the cluster.



Our approach: resource fungibility

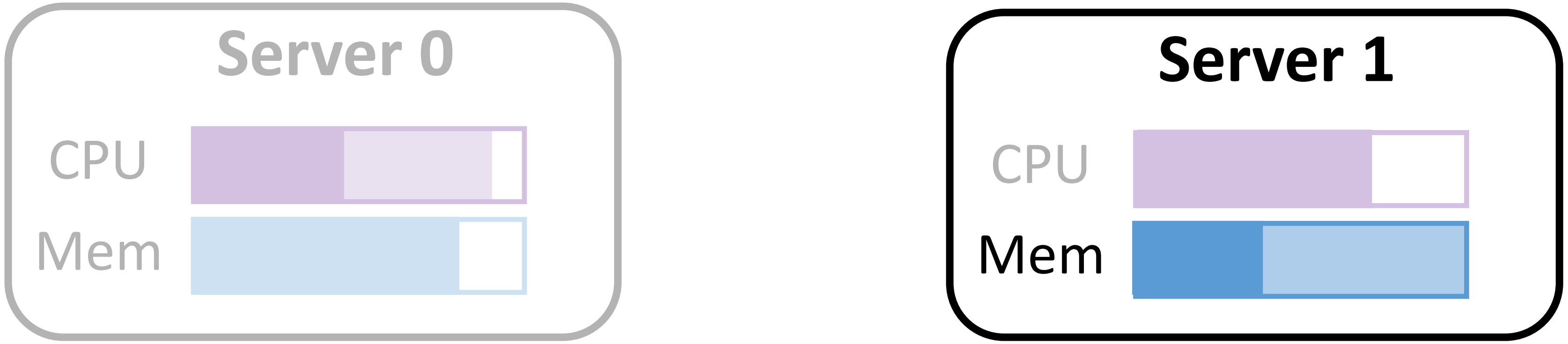
➢ We advocate for ***fungible*** applications

- that can use resources wherever they are in the cluster.



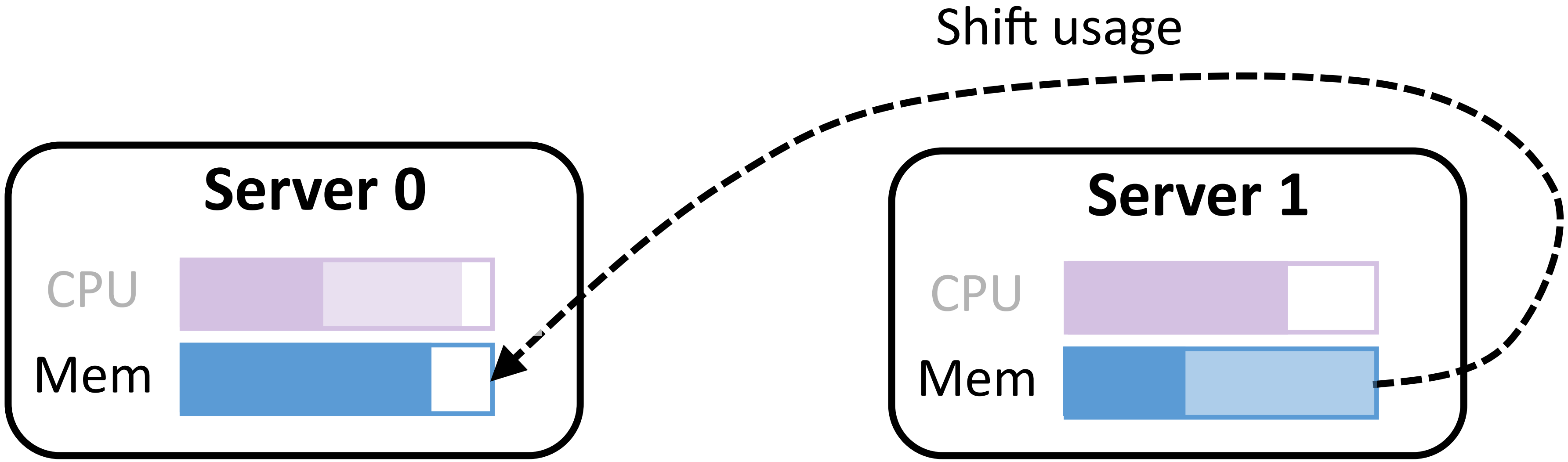
Our approach: resource fungibility

- ➢ We advocate for ***fungible*** applications
  - that can use resources wherever they are in the cluster.



Our approach: resource fungibility

- We advocate for ***fungible*** applications
  - that can use resources wherever they are in the cluster.

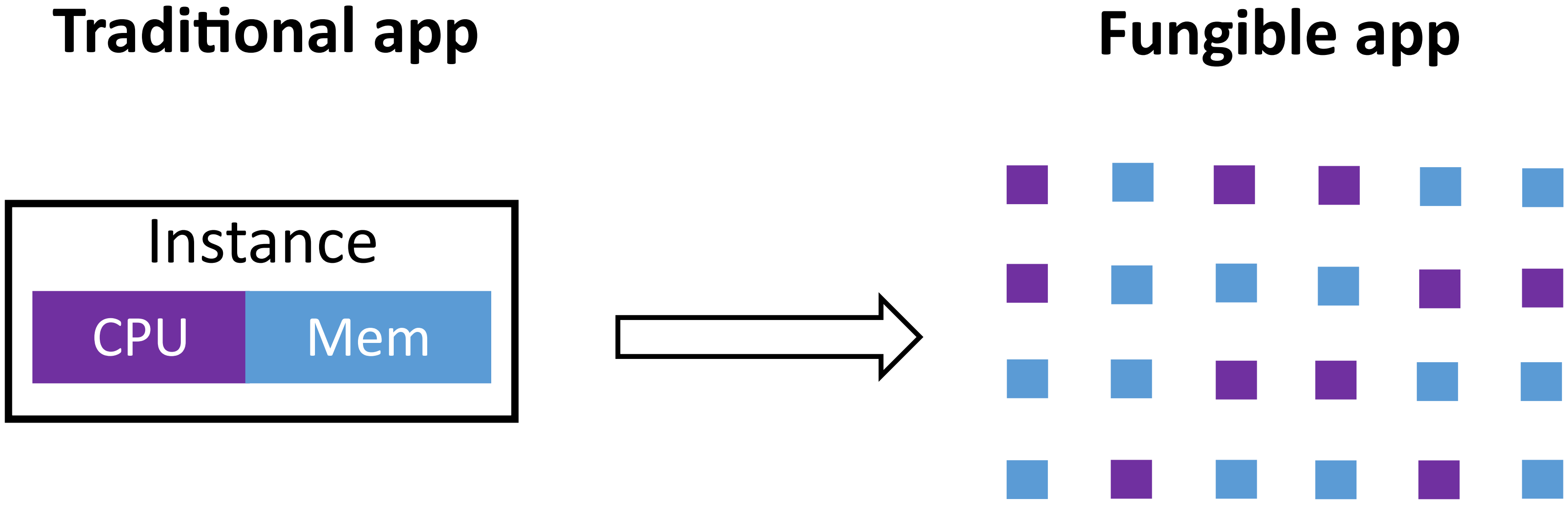


Resource proclet --- a new abstraction

- A resource proclet is an ***independent*** scheduling unit that consumes a ***small amount*** of ***single*** resource.

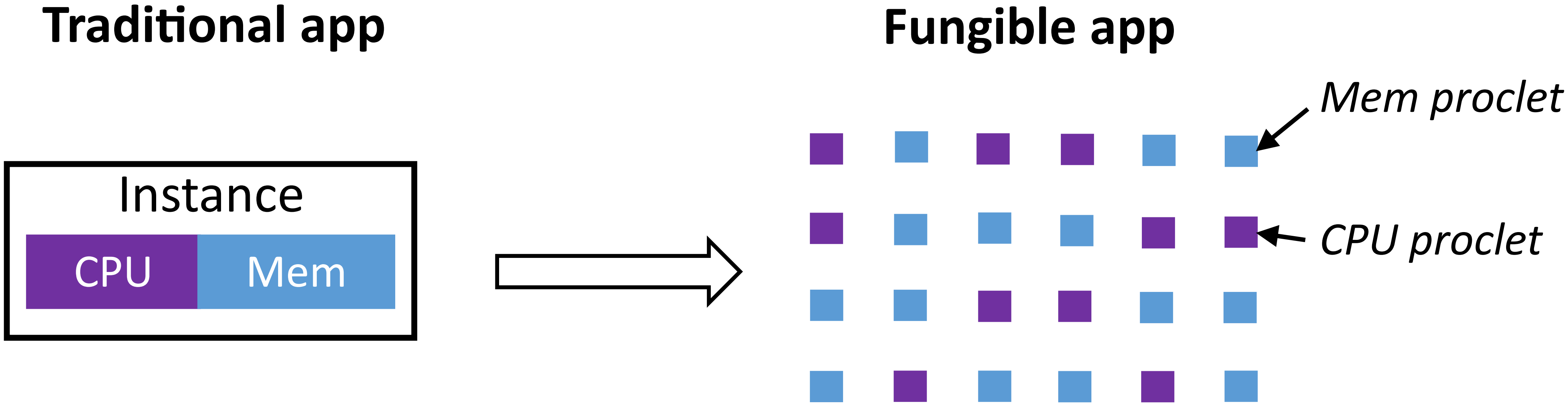
Resource proclet --- a new abstraction

- A resource proclet is an ***independent*** scheduling unit that consumes a ***small amount*** of ***single*** resource.



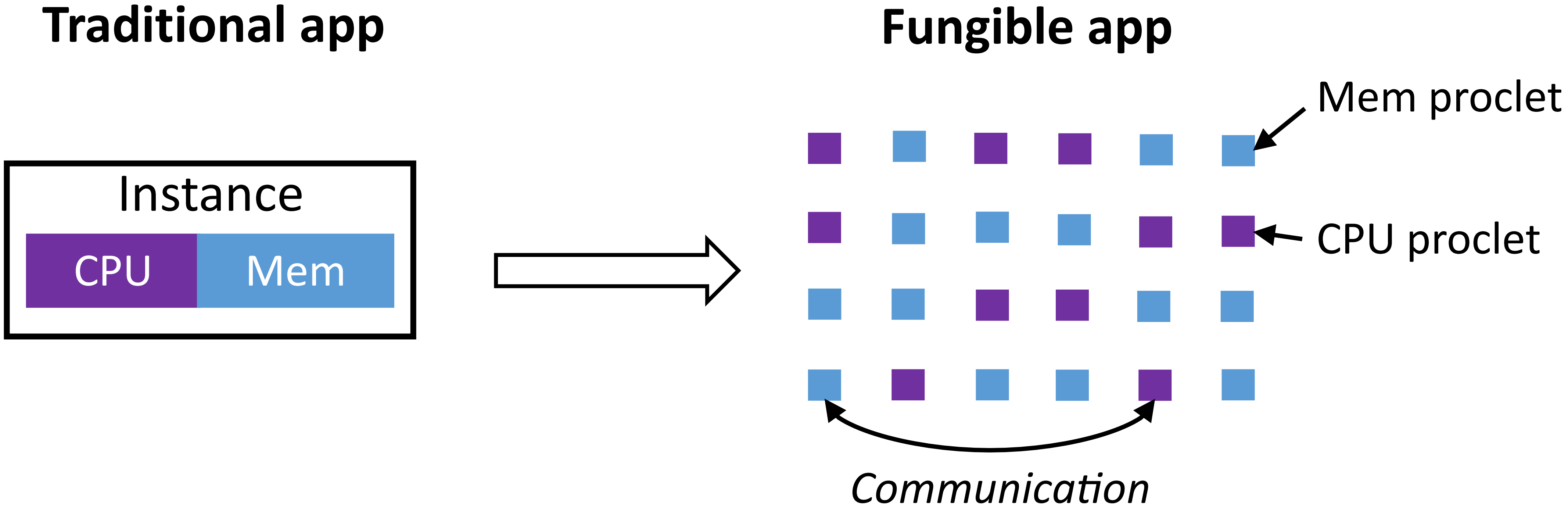
Resource proclet --- a new abstraction

- A resource proclet is an ***independent*** scheduling unit that consumes a ***small amount*** of ***single*** resource.



Resource proclet --- a new abstraction

- A resource proclet is an ***independent*** scheduling unit that consumes a ***small amount*** of ***single*** resource.



Traditional app

Fungible app

Instance

CPU

Mem

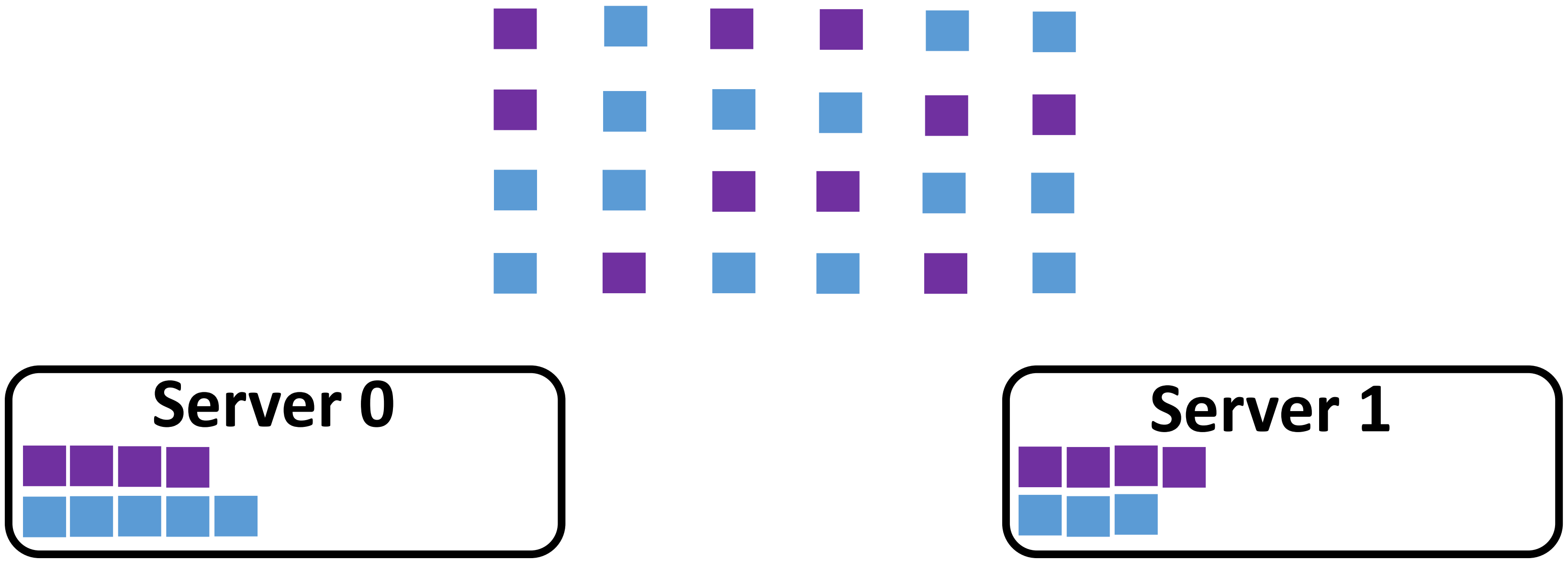
Mem proclet

CPU proclet

Communication

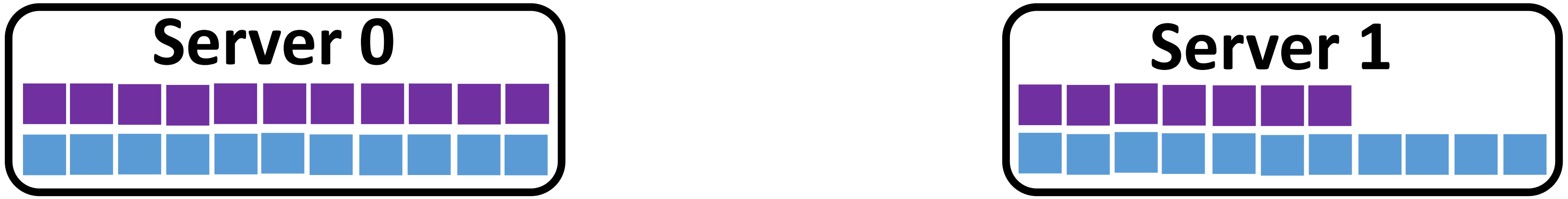
Resource proclet --- a new abstraction

- A resource proclet is an ***independent*** scheduling unit that consumes a ***small amount*** of ***single*** resource.



Resource proclet --- a new abstraction

- A resource proclet is an ***independent*** scheduling unit that consumes a ***small amount*** of ***single*** resource.



Resource proclet--- a new abstraction

- A resource proclet is an ***independent*** scheduling unit that consumes a ***small amount*** of ***single*** resource.

Server 0
Server 1

How to program with resource proclets?

How to program with resource proclets?

👩‍💻 **High-level Abstraction**

FungibleVector<T>

```
push_back(T)
T operator[](size_t);
```

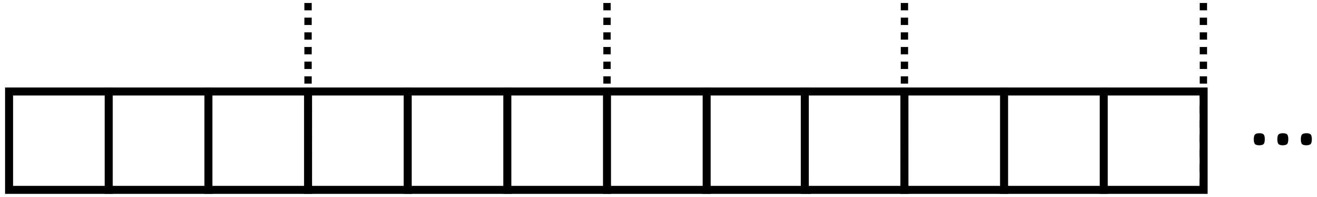
How to program with resource proclets?

👩‍💻 **High-level Abstraction**

FungibleVector<T>

push_back(T)
T operator[](size_t);

Sharding Library



How to program with resource proclets?

👩‍💻 **High-level Abstraction**

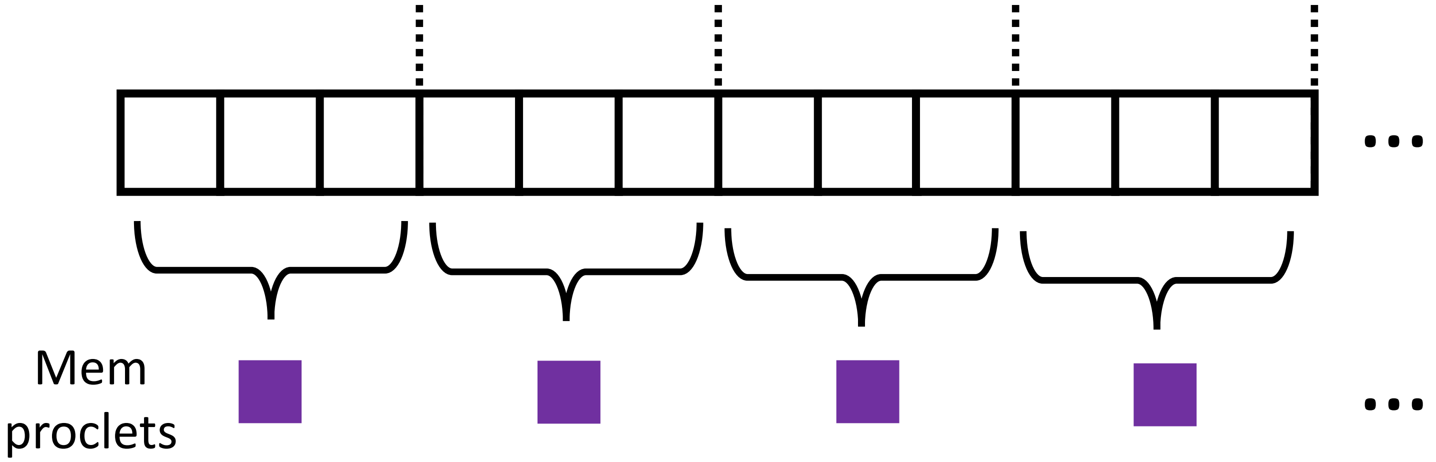
FungibleVector<T>

push_back(T)

T operator[](size_t);

Sharding Library

Resource Proclets



Mem proclets

How to program with resource proclets?

High-level Abstraction

FungibleVector<T>

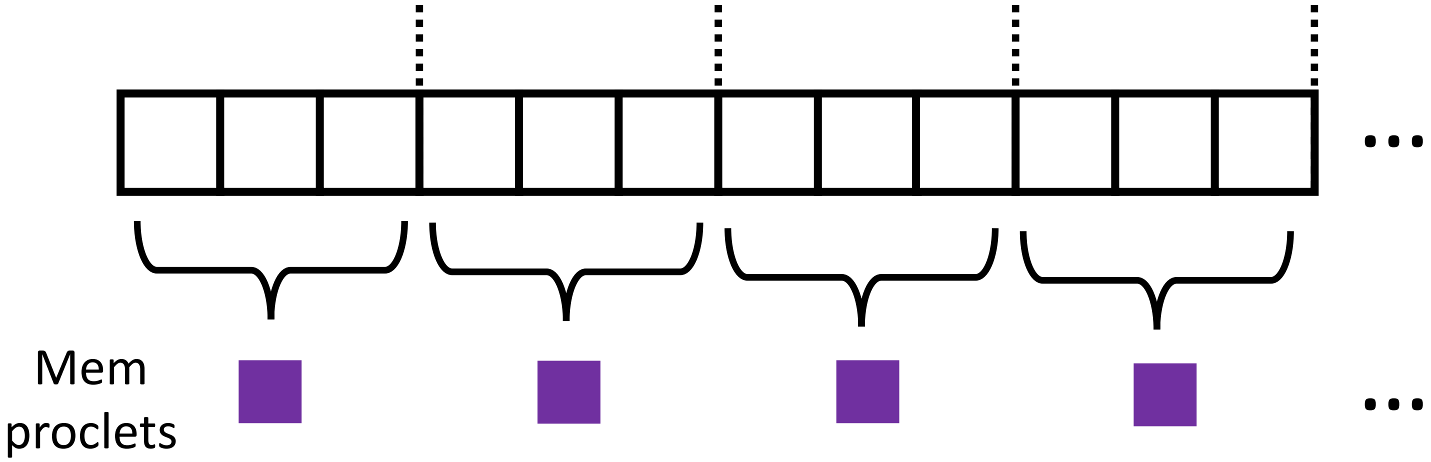
push_back(T)

T operator[](size_t);

➢ forall(λ)

Sharding Library

Resource Proclets



How to program with resource proclets?

High-level Abstraction

FungibleVector<T>

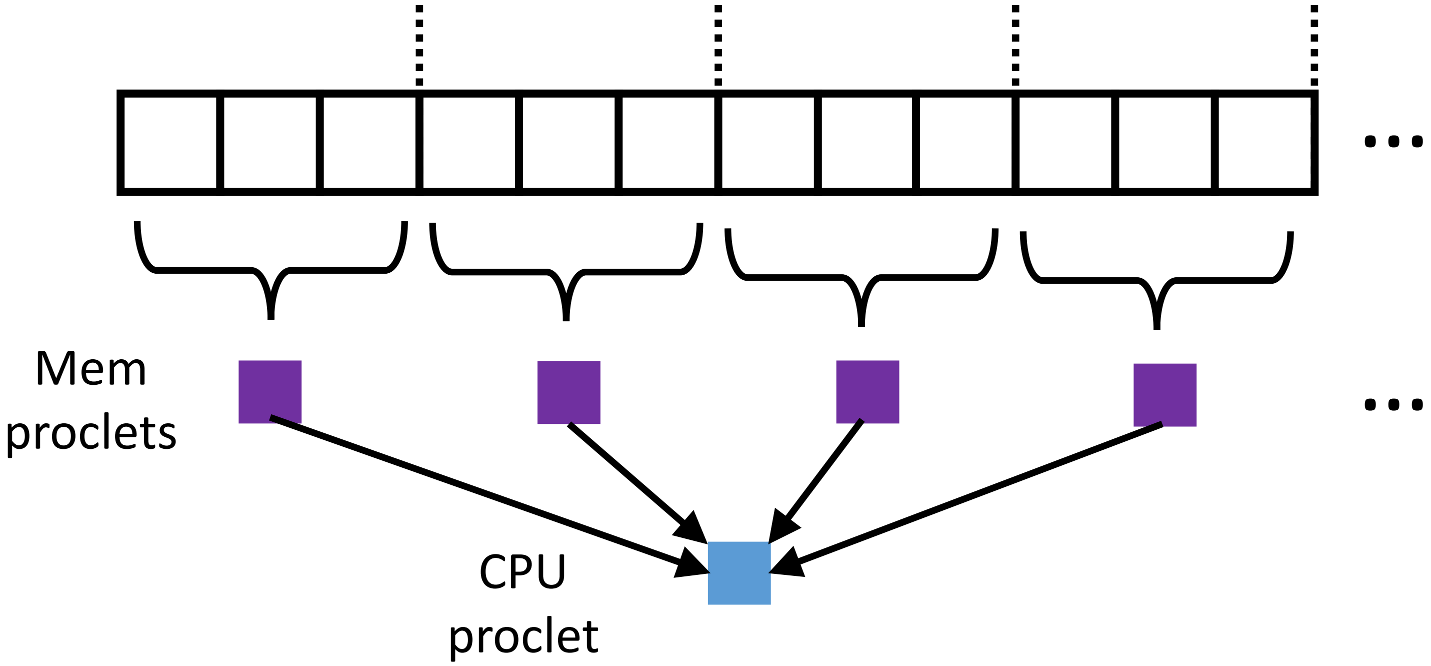
push_back(T)

T operator[](size_t);

➢ forall(λ)

Sharding Library

Resource Proclets



...

Mem proclets

...

CPU proclet

How to program with resource proclets?

👩‍💻 **High-level Abstraction**

FungibleVector<T>

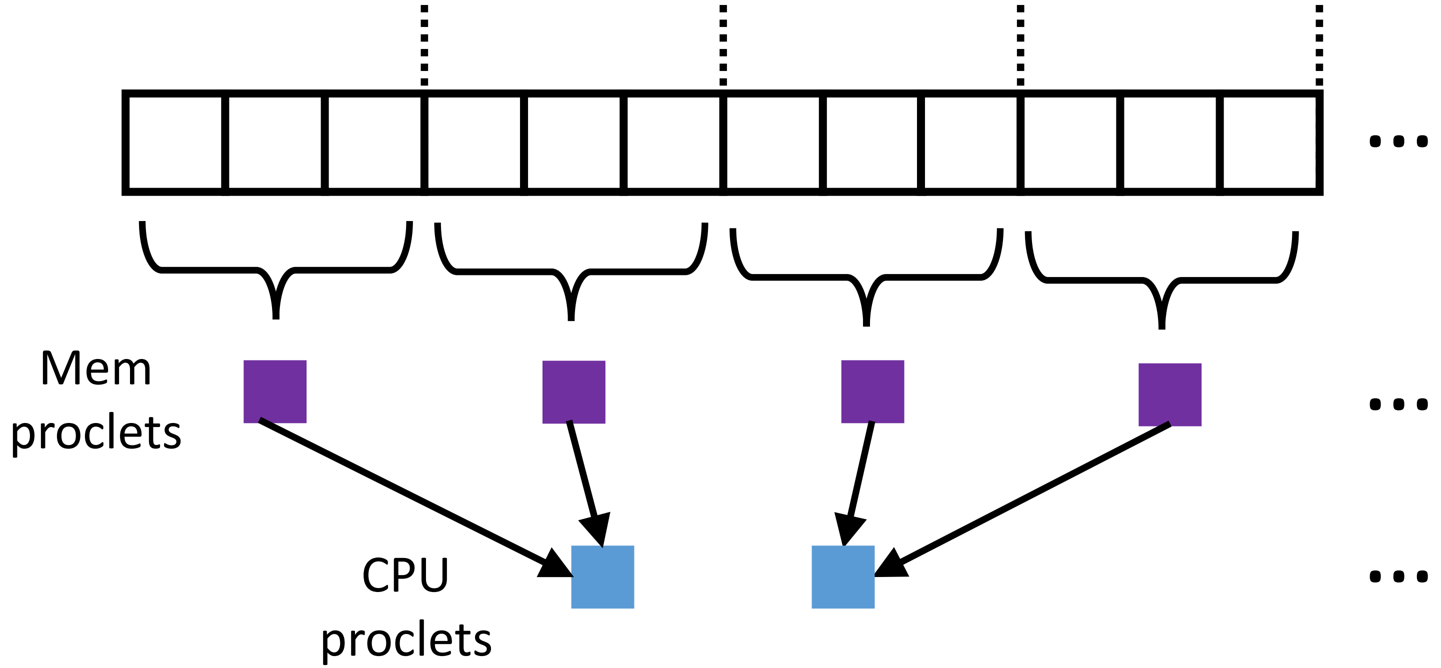
push_back(T)

T operator[](size_t);

➢ forall(λ)

Sharding Library

Resource Proclets

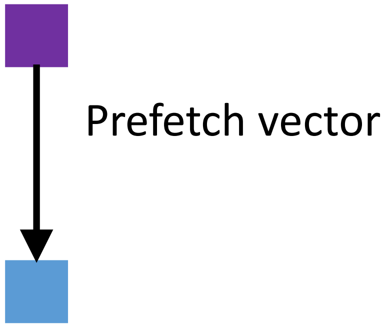


How to overcome communication cost?

- Problem: data locality is crucial for apps with low compute intensity.
  - E.g., vector.forall(x -> x + 1).

How to overcome communication cost?

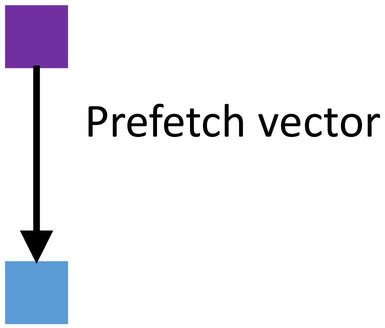
- Problem: data locality is crucial for apps with low compute intensity.
  - E.g., vector.forall(x-> x + 1).
- Solutions:

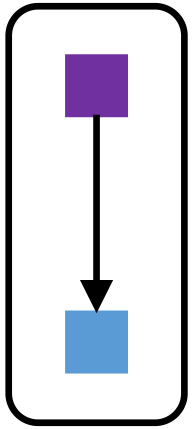


➢ **1. Prefetching**

How to overcome communication cost?

- Problem: data locality is crucial for apps with low compute intensity.
  - E.g., vector.forall(x -> x + 1).
- Solutions:





- **1. Prefetching**

➢ **2. Colocation**

Performance Under Imbalance

- Can we successfully combine resources to achieve fungibility?

Performance Under Imbalance

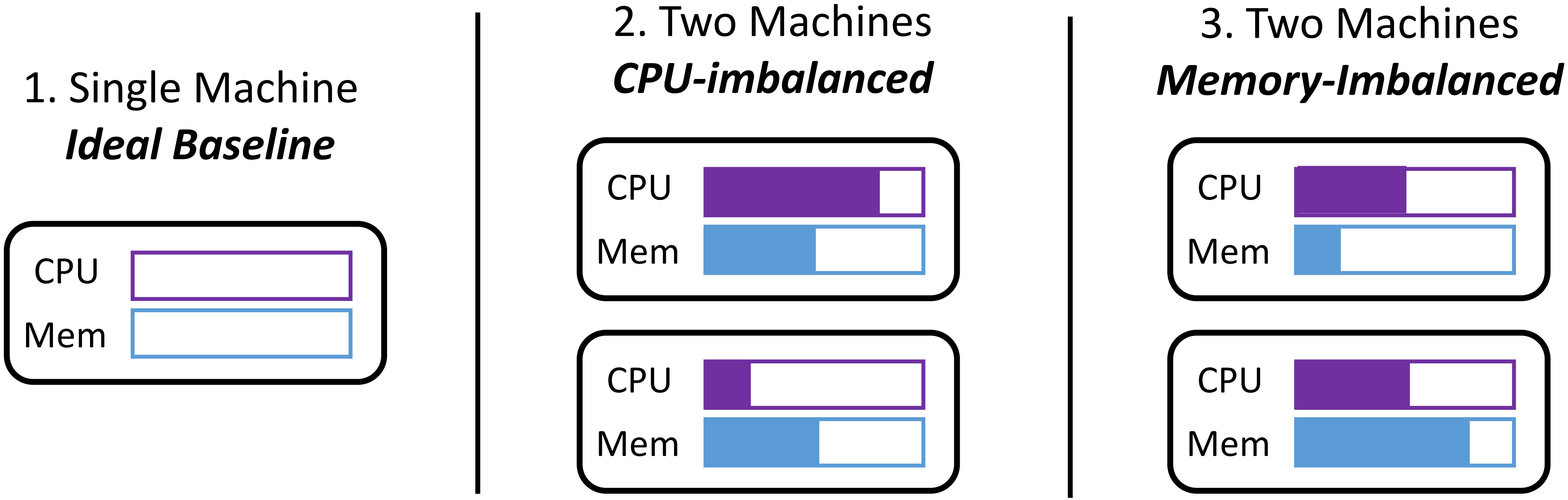
- Can we successfully combine resources to achieve fungibility?

➢Built an initial prototype *Quicksand*; workload: image preprocessing.

Performance Under Imbalance

- Can we successfully combine resources to achieve fungibility?

➢Built an initial prototype *Quicksand*; workload: image preprocessing.

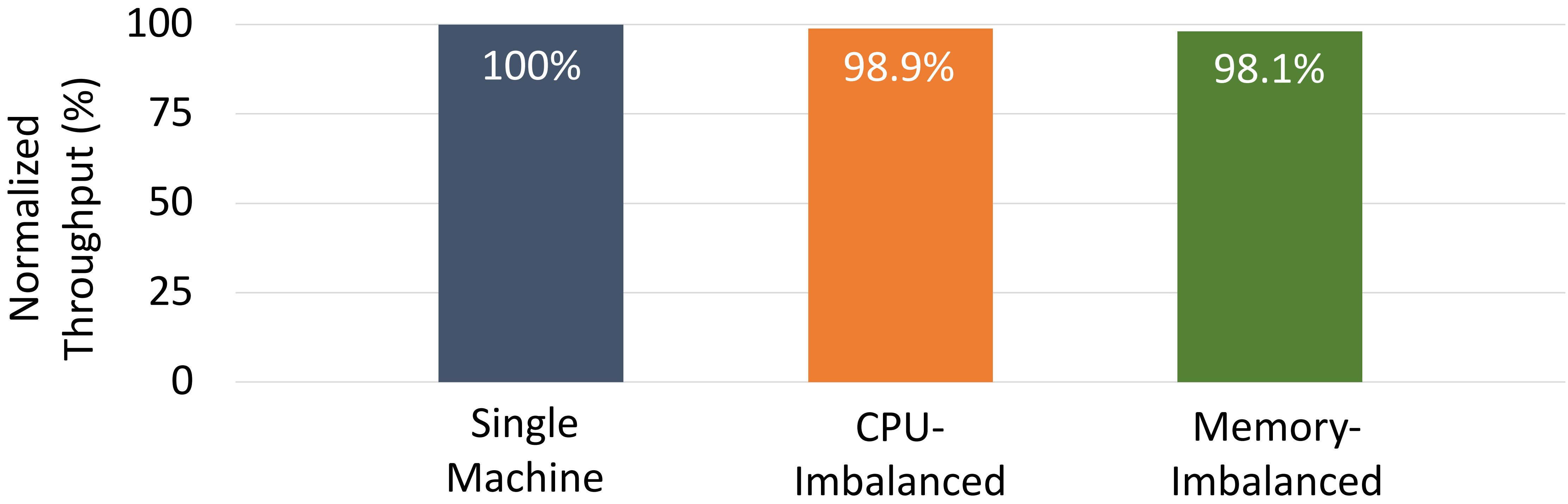


1. Single Machine
Ideal Baseline

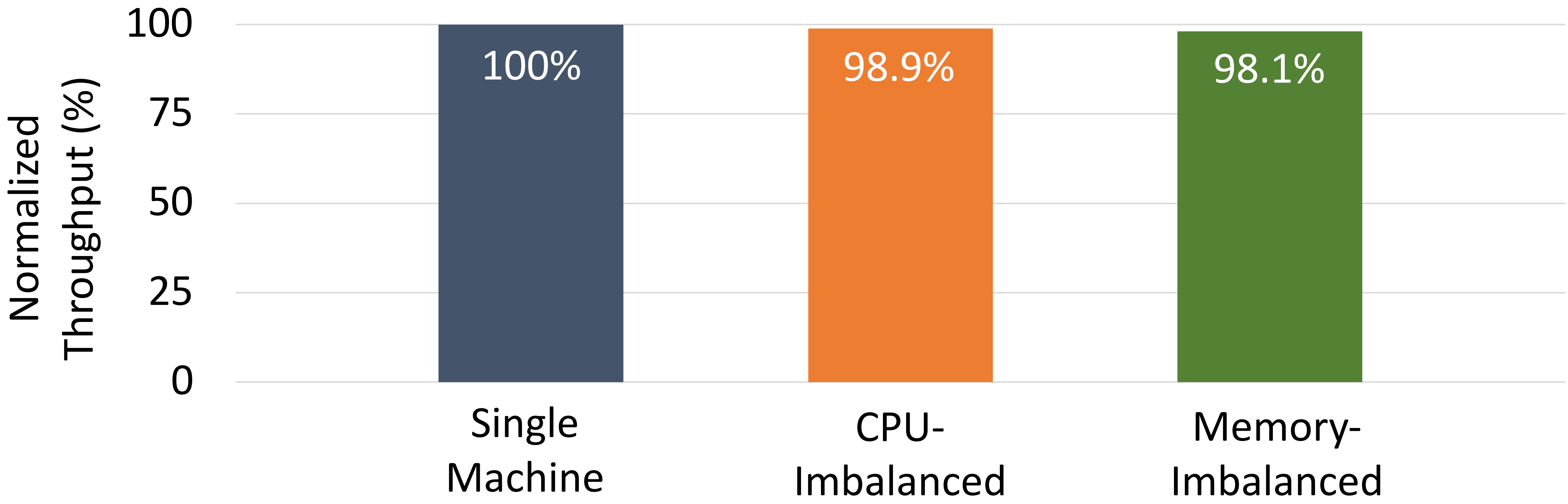
2. Two Machines
CPU-imbalanced

3. Two Machines
Memory-Imbalanced

Performance Under Imbalance

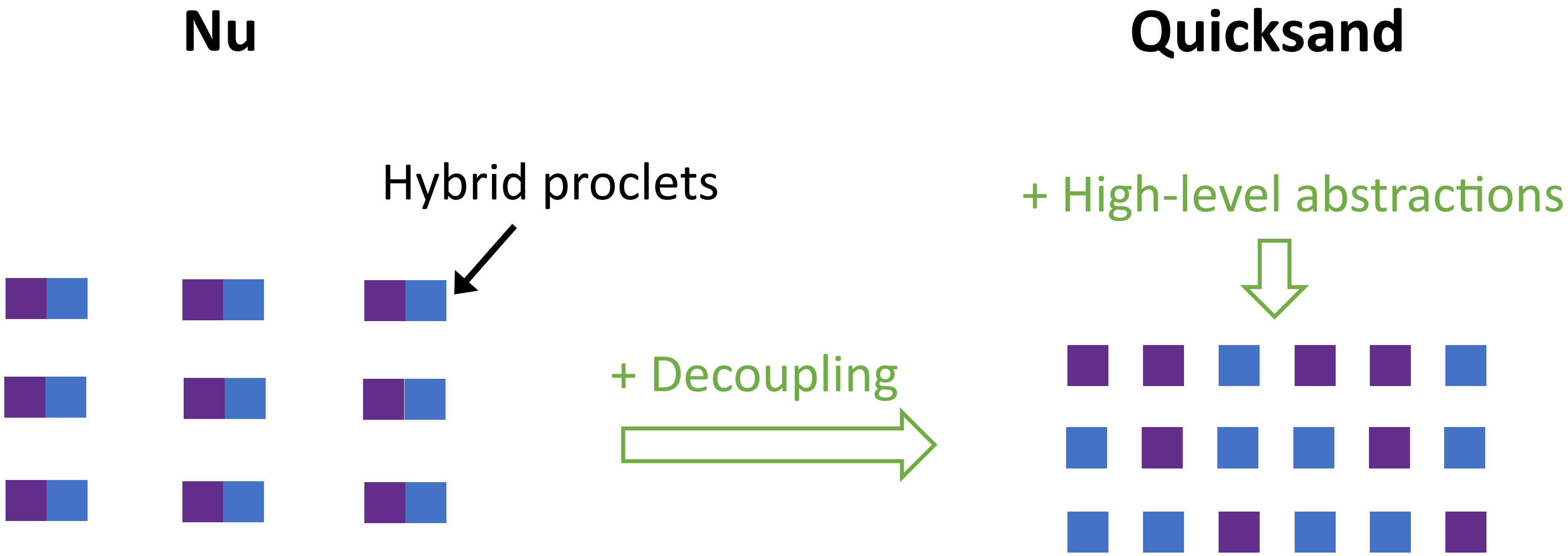


Performance Under Imbalance



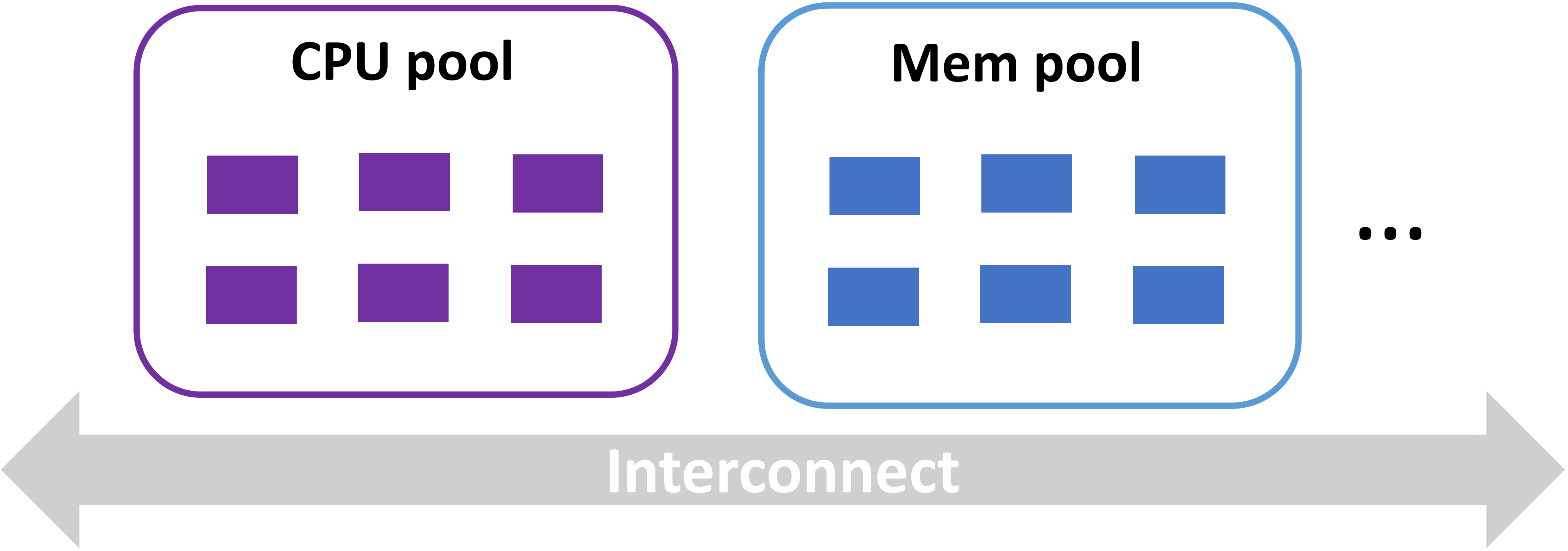
- **Promising to achieve fungibility with today’s datacenter hardware!**

Related work 1: Nu [NSDI' 23]



Related work 2: HW resource disaggregation

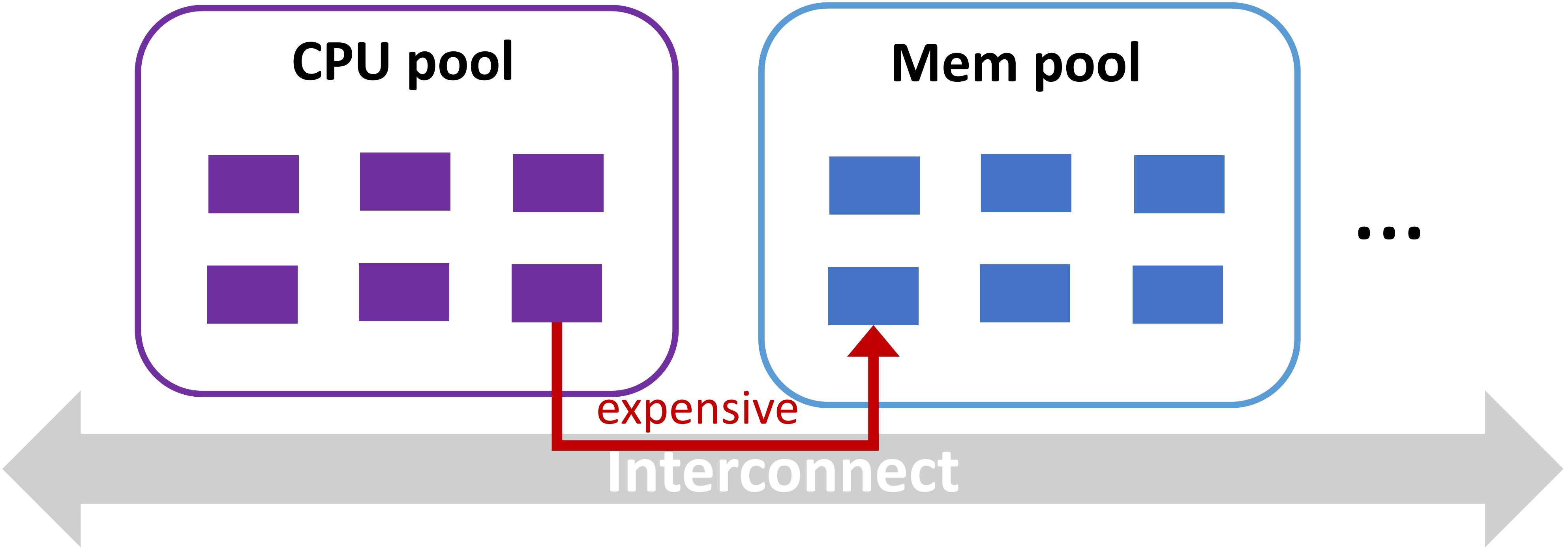
+ Transparent



Related work 2: HW resource disaggregation

+ Transparent

-- Loses the control over resource placement

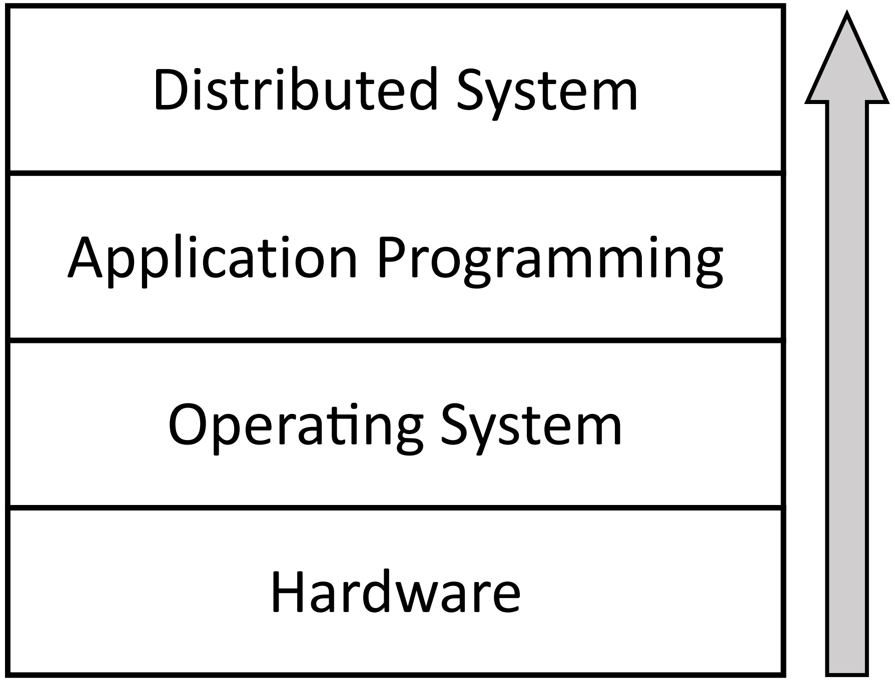


Related work 3: distributed programming model

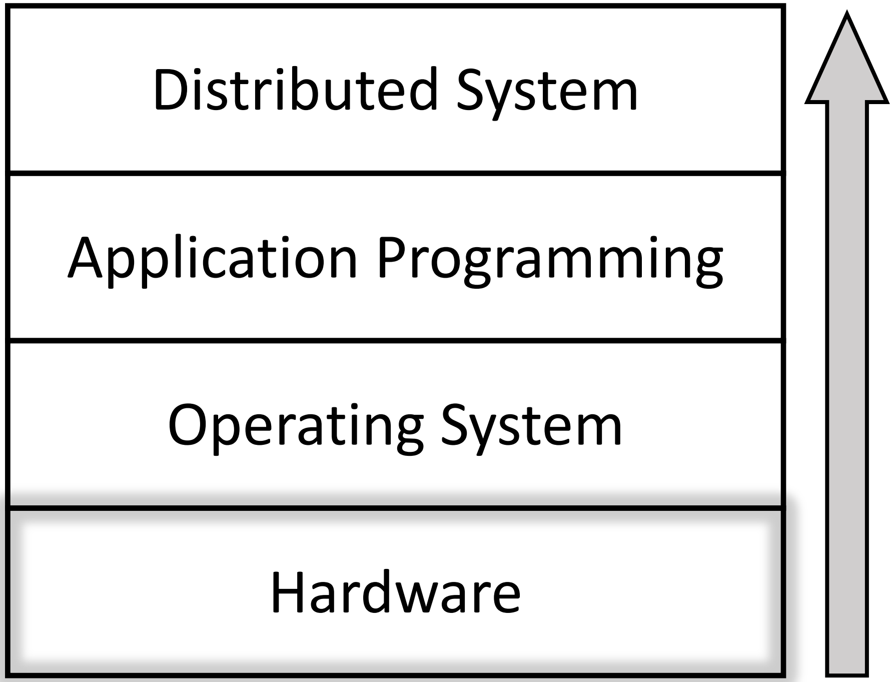
- Actor --- ServiceWeaver [HotOS’ 23], Ray [OSDI’ 18]
- Microservice --- Nightcore [ASPLOS’ 21]
- Serverless --- Boki [SOSP’ 21]

➢**Shared trend: applications are going granular.**

Many untapped opportunities



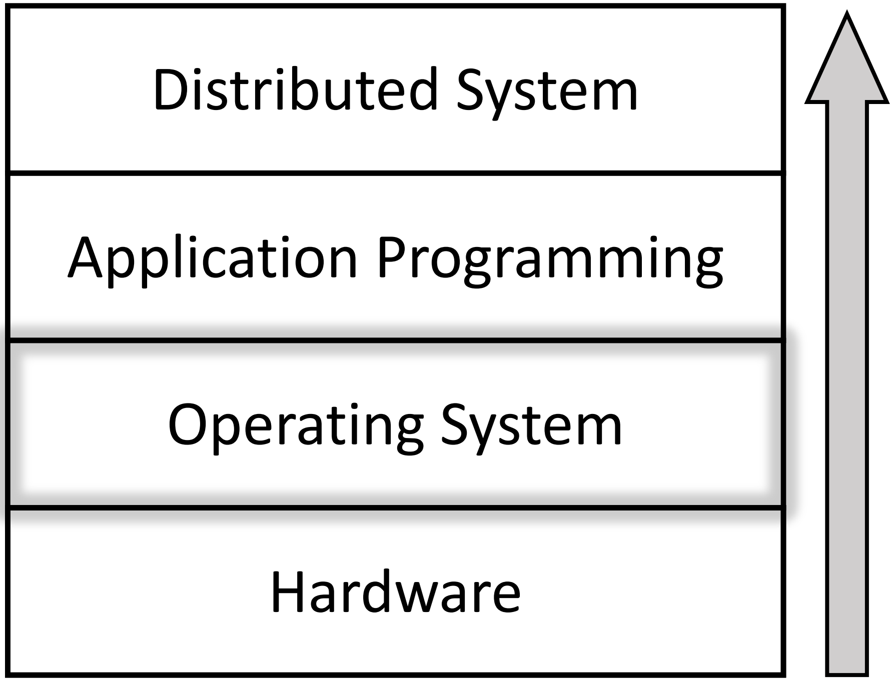
Many untapped opportunities



- More types of resources to decouple

  Future interconnect like CXL can help

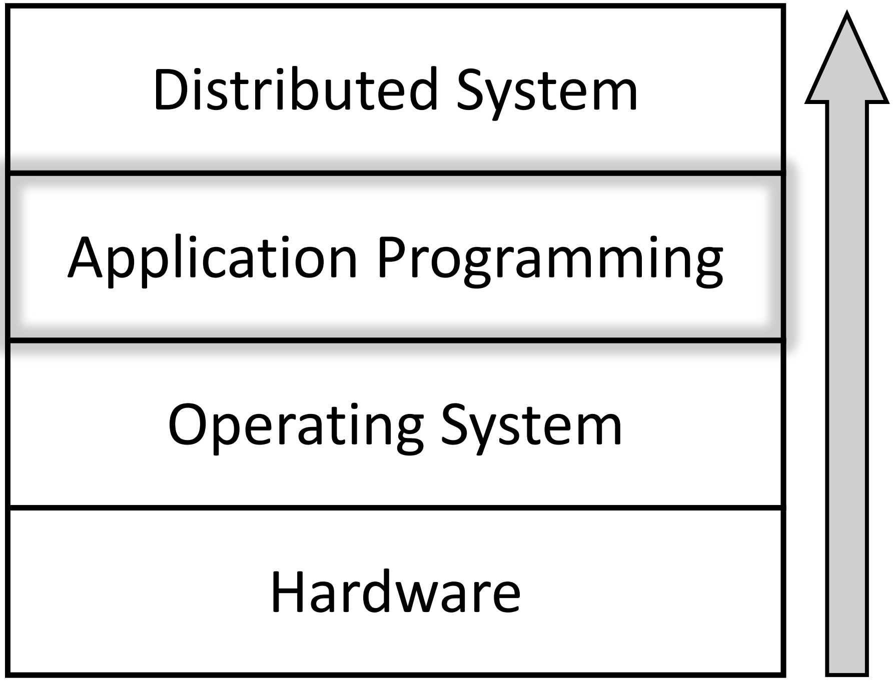
Many untapped opportunities



- ➢ OS can provide a native support to RPs
- More types of resources to decouple

  Future interconnect like CXL can help

Many untapped opportunities



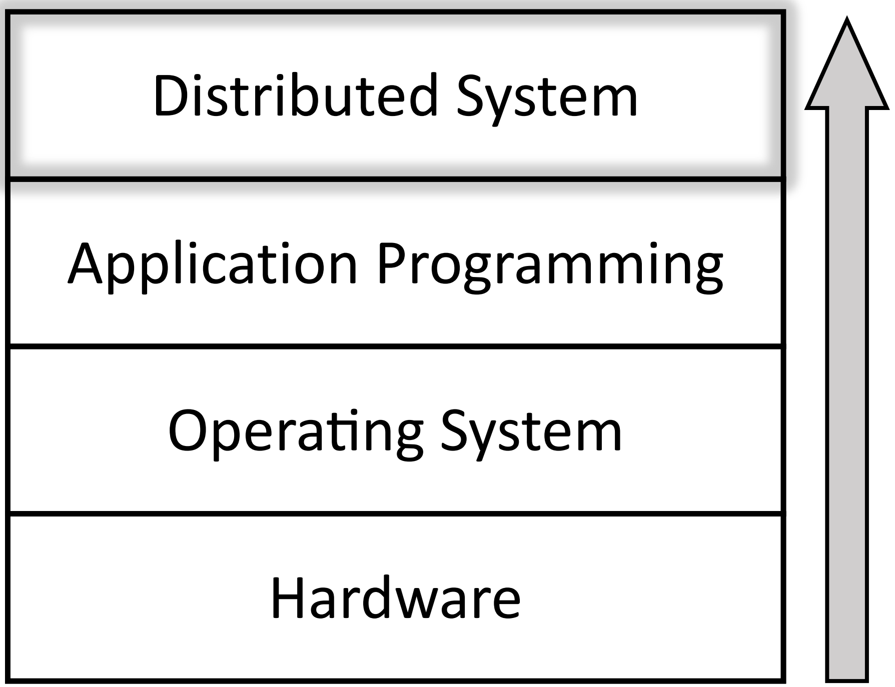
- ➢ Compiler can minimize code change

- OS can provide a native support to RPs

- More types of resources to decouple

  Future interconnect like CXL can help

Many untapped opportunities



- ➢ Scheduler can optimize locality
- Compiler can minimize code change
- OS can provide a native support to RPs
- More types of resources to decouple
  Future interconnect like CXL can help

Now is the time to realize resource fungibility!