Serverless, Quo Vadis? Towards Granular Management in Serverless Clouds

Prof. Peter Pietzuch

(joint work with **Carlos Segarra**, Simon Shillaker, Guo Li, Eleftheria Mappoura, Rodrigo Bruno, Lluis Vilanova)

Department of Computing and I-X
Imperial College London

http://lsds.doc.ic.ac.uk
prp@imperial.ac.uk

SESAME Workshop – Rotterdam, Netherlands – March 2025

What is Serverless?

SESAME @ ASPLOS&EuroSys'25 Schedule Dates Call for papers **Submit NOW**

The 3rd Workshop on SErverless Systems

Call for Papers

Serverless has emerged as the next dominant cloud architecture and paradigm due to its elastic scalability and flexible billing model. In serverless, developers focus on writing their application's business logic, e.g., as a set of functions and GenAI models connected in a workflow, whereas providers take responsibility for dynamically managing cloud resources, e.g., by scaling the number of instances for each deployed function. This division of responsibilities opens new opportunities for systems researchers

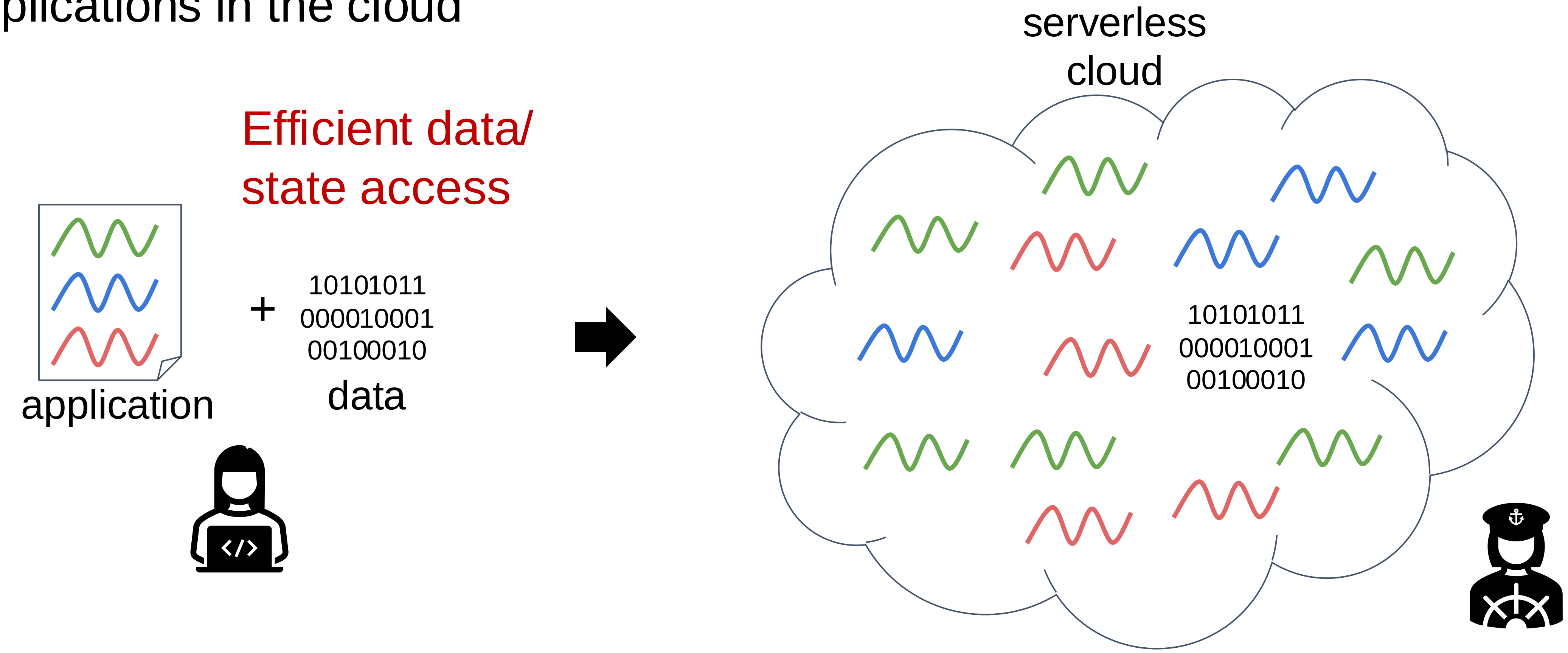
This year's addition: Serverless and efficient GenAI inference serving

The workshop is co-located with ASPLOS'25 and EuroSys'25 and will be conducted in person (no remote participants) on March 31st, bringing the experts from academia and industry to facilitate research in serverless systems in Postillion Hotel & Convention Centre WTC Rotterdam in Mees I room.

The Vision of Serverless Clouds

Promise of efficient and simple online/compute-intensive/machine learning applications in the cloud

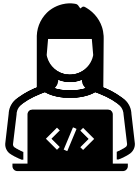
Efficient data/
state access



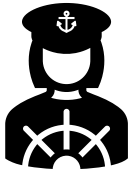
Computation expressed as **functions**

Provider achieves **performance**, **scalability** and **efficiency**

Why is Serverless Great?



Cloud user



Serverless cloud provider

Only need to focus on business logic

Flexible resource management

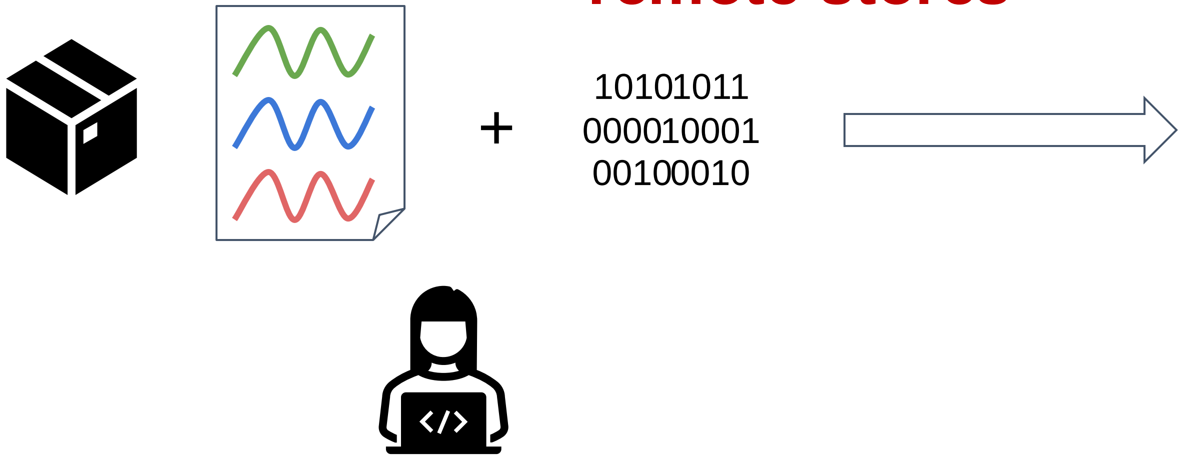
High performance + elastic scaling

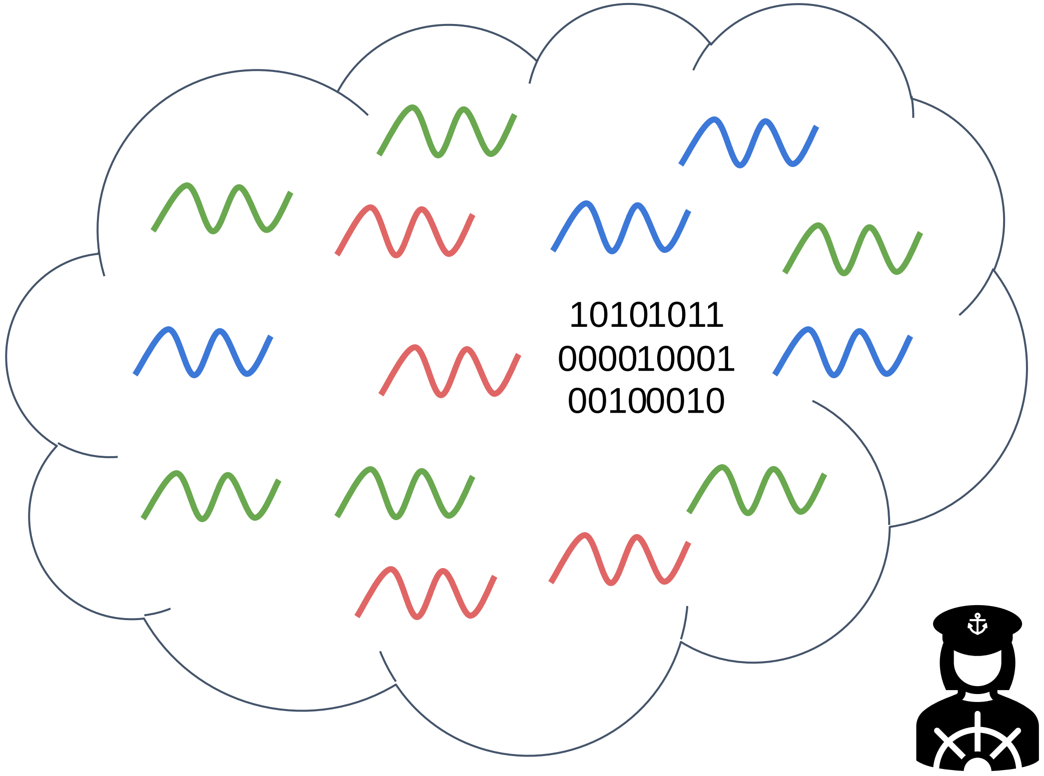
High efficiency + utilisation

The Reality of Serverless Clouds

Promise of efficient and simple online/compute-intensive/machine learning applications in the cloud

Data/state must live in **remote stores**

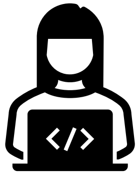




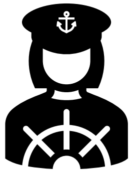
Applications packaged as **containers**

Provider handles **container orchestration**

We Don’t Deliver The Promised Benefits



Cloud user



Serverless cloud provider

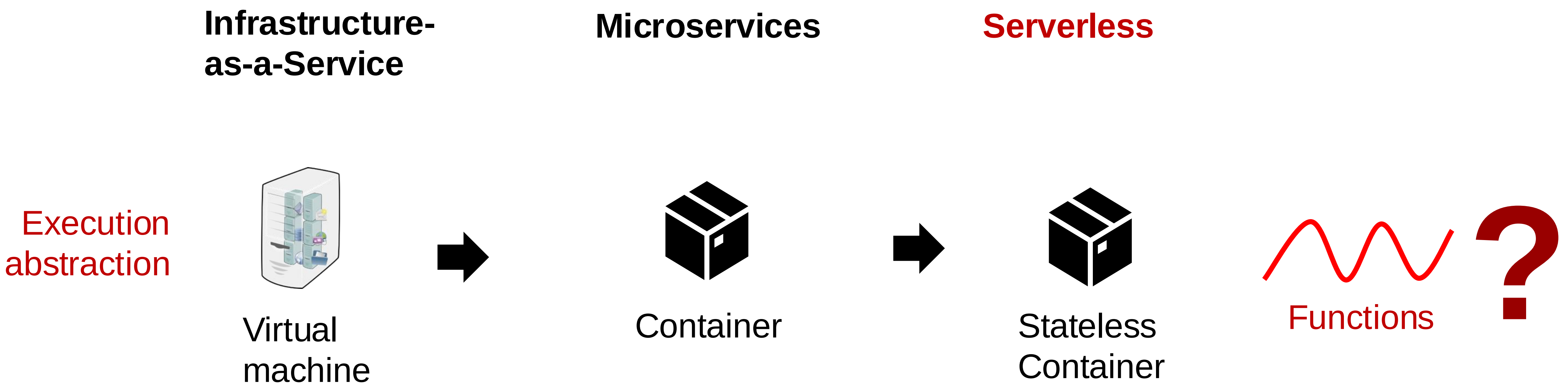
Only need to focus on business logic ✗ Complexity of container deployment

Flexible resource management ✗ Many scheduling constraints

High performance + elastic scaling ✗ Overhead + cold-start effects

High efficiency + utilisation ✗ Limited density + container execution overheads

Serverless – What Went Wrong?



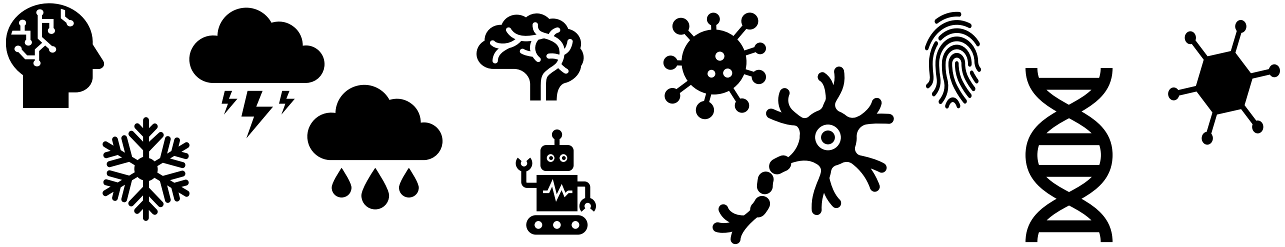
We need a **new execution abstraction**!

Picking an Execution Abstraction for Serverless

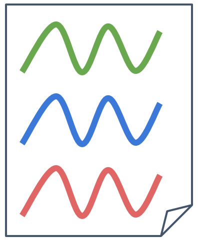
Don’t reinvent the wheel

Consider how existing compute-intensive applications are written

- Machine learning
- Weather forecasting
- Genetic modelling
- Molecule dynamics simulation



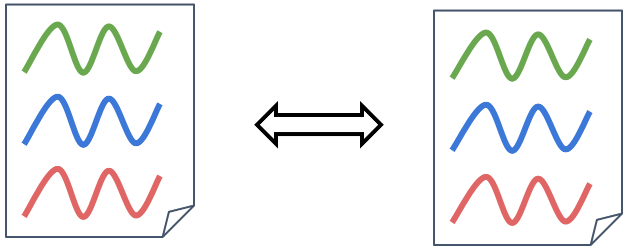
Multi-Threaded + Multi-Process Applications



OpenMP

NumPy

multi-threaded applications with **shared memory**



Open MPI

PyTorch

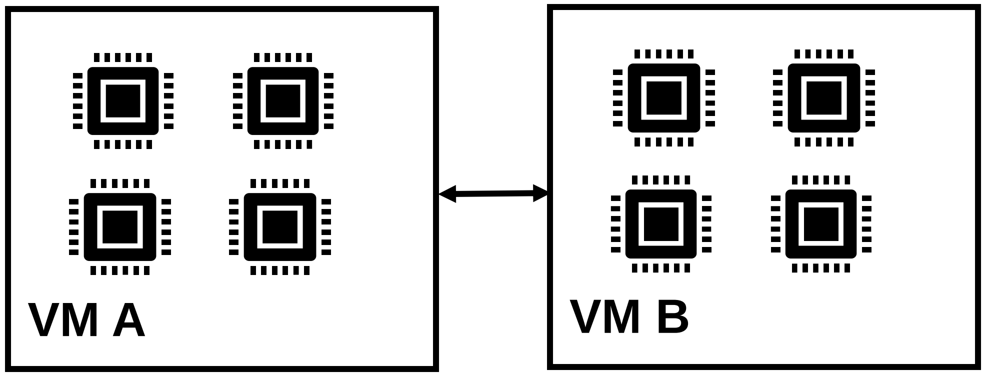
multi-process applications with **message passing** between nodes

Why are such Applications a Good Fit for Serverless?

Multi-threaded/process applications support scaling in the cloud

scale up multi-threaded apps:

allocate more threads to vCPUs

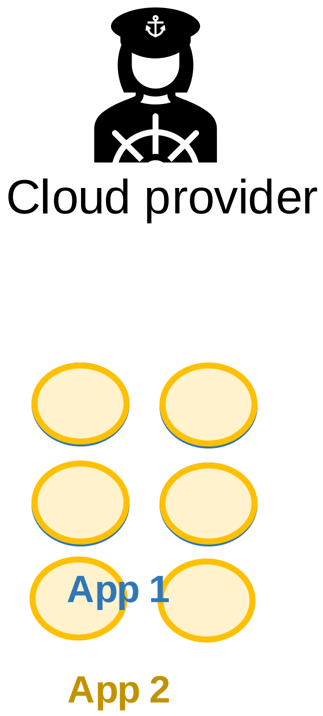


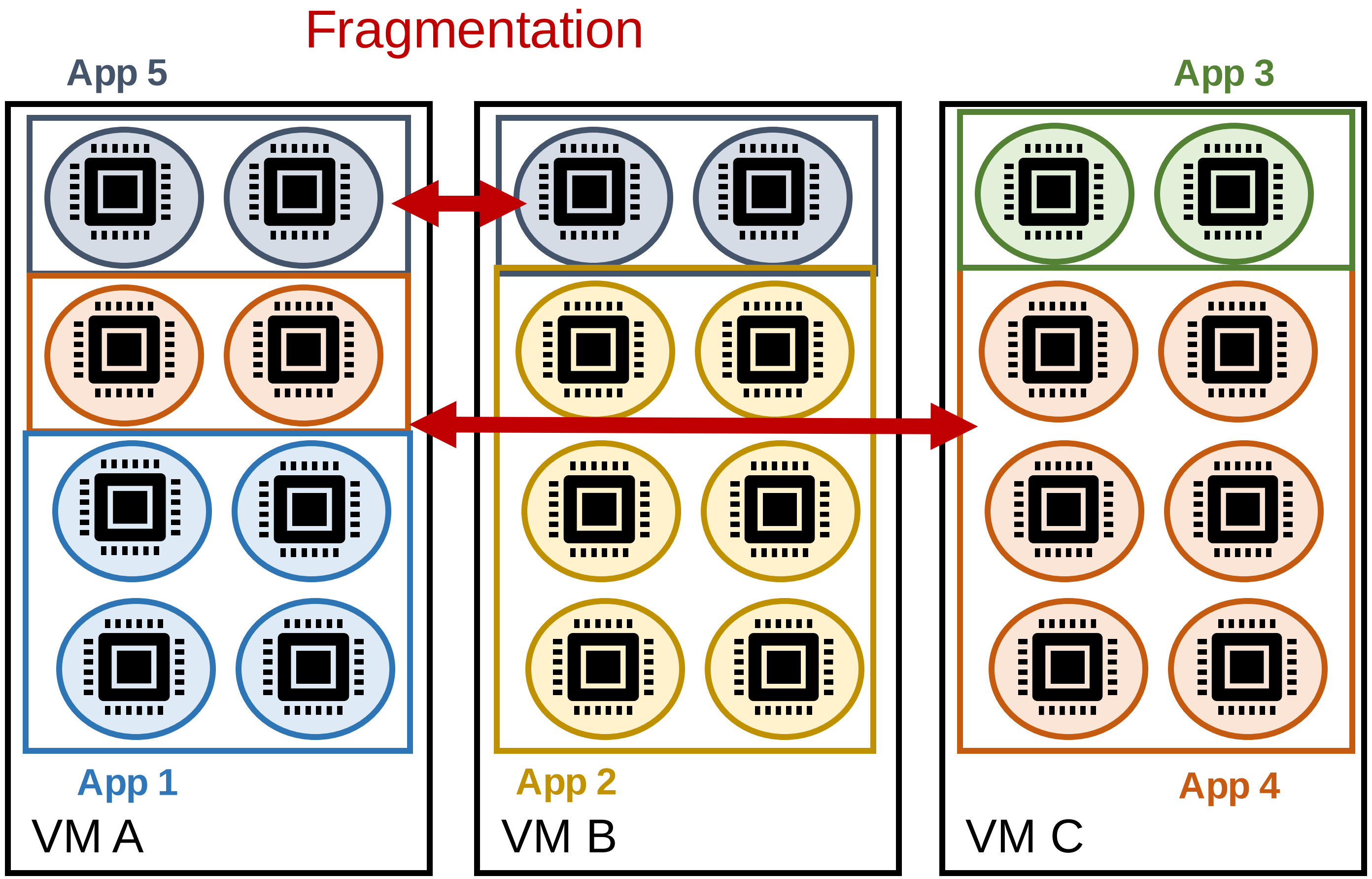
scale out multi-process apps:

allocate more VMs

How to Allocate Such Applications?

Cloud provider wants high utilisation of vCPUs

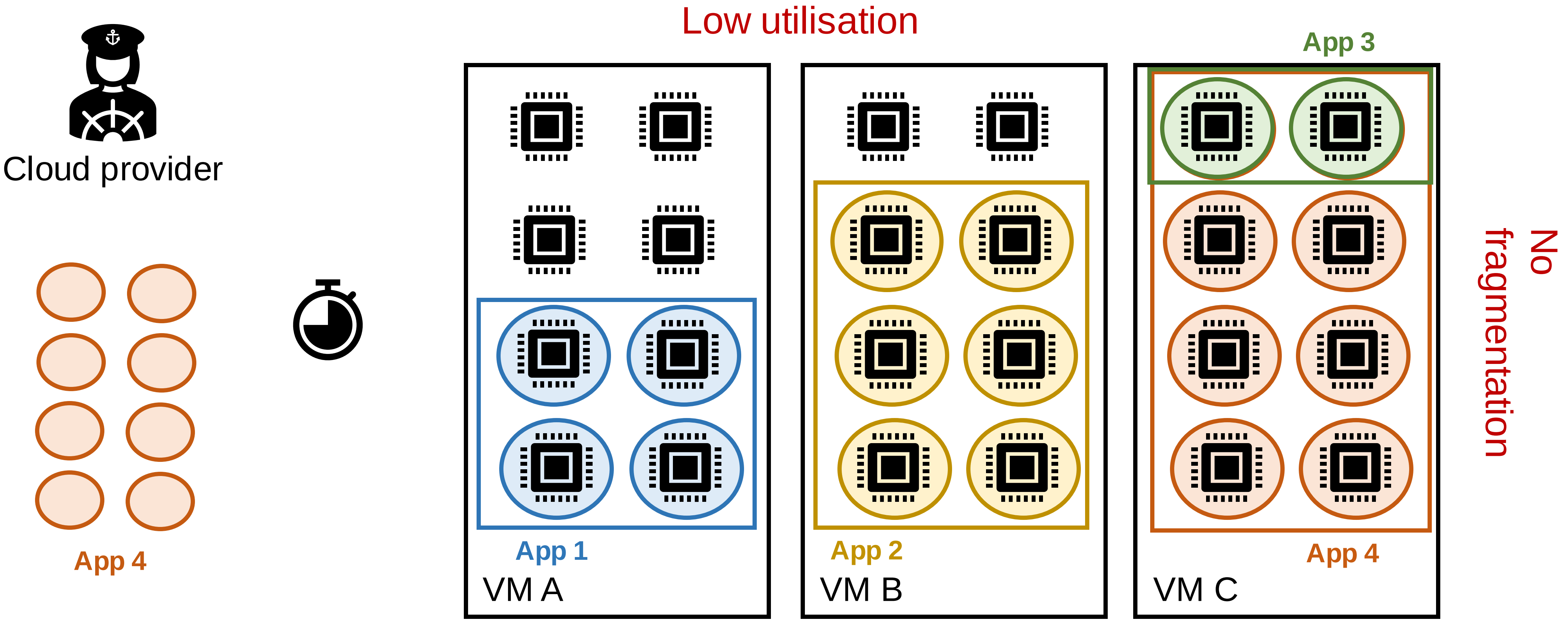




High utilisation leads to fragmentation, thus worse performance

But Cloud Users Want Good Performance

Cloud provider can delay allocation to avoid fragmentation

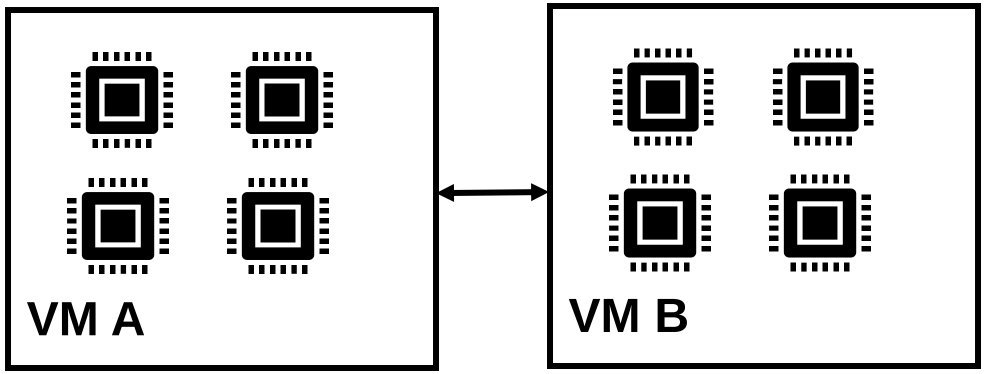


Idea: Can we dynamically change the allocation of multi-threaded/process apps?

How To Support Dynamic Allocation in Cloud?

Requires elastic scaling of multi-threaded applications

Challenge:
Adding threads without violating consistency



Requires dynamic migration of multi-process applications

Challenge:
Migrating processes efficiently and correctly

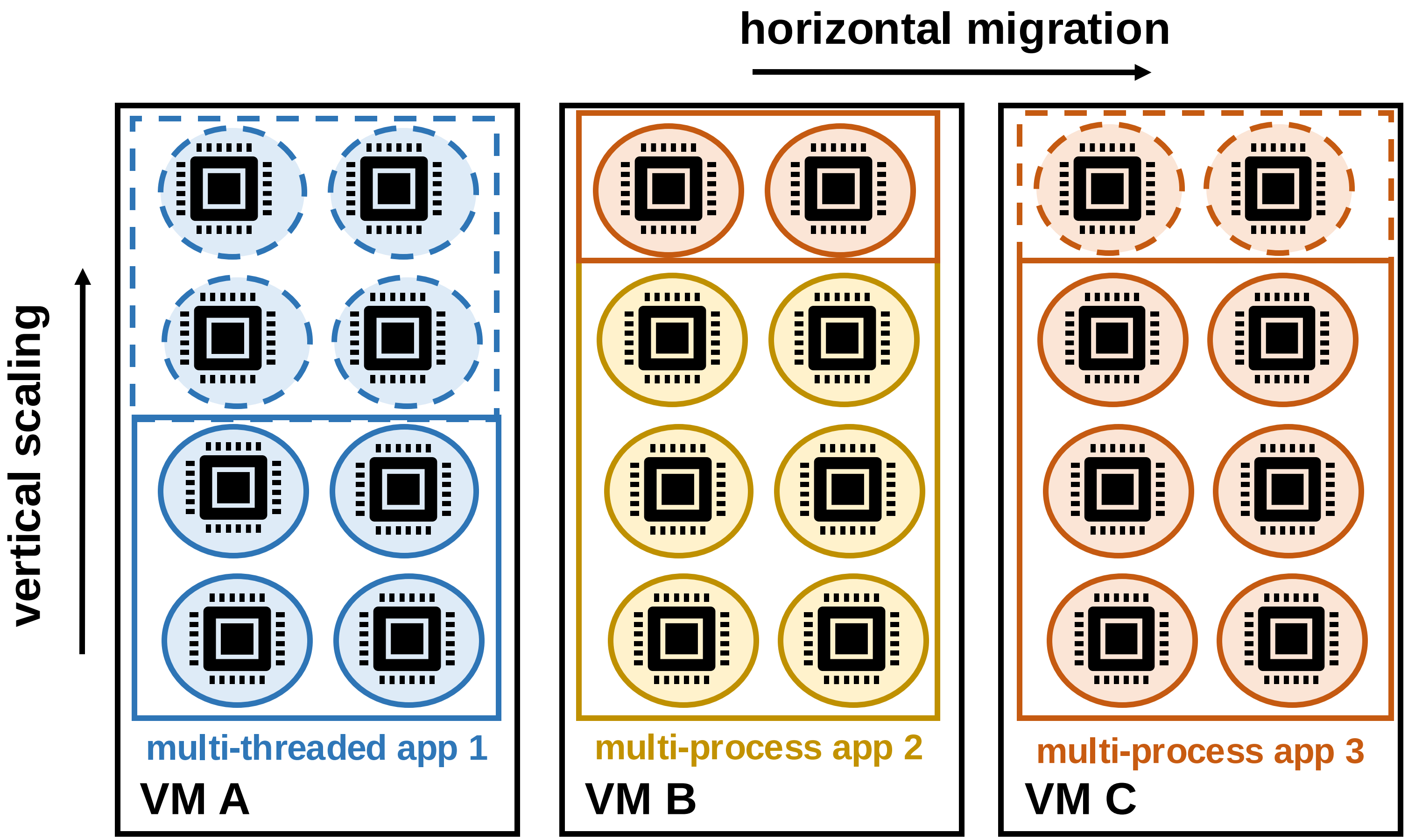
Granny: Runtime for Multi-Threaded/Process Serverless Apps

Granny enables cloud providers to dynamically manage multi threaded/process applications

Granny supports:

- adding vCPUs to multi-threaded apps
- migrating vCPUs of multi-process apps

Granny executes unmodified multi-threaded (OpenMP) and multi-process (MPI) apps

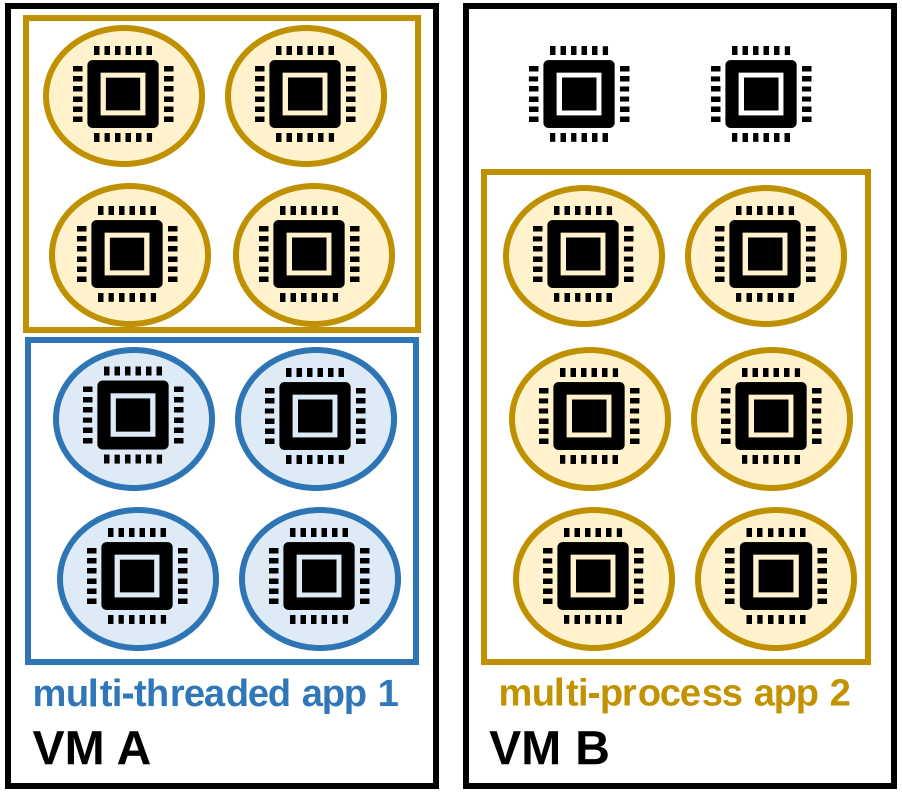


Granule Execution Abstraction

A Granule can execute with thread or process semantics

Each Granule occupies 1 vCPU

Multi-threaded apps use multiple Granules

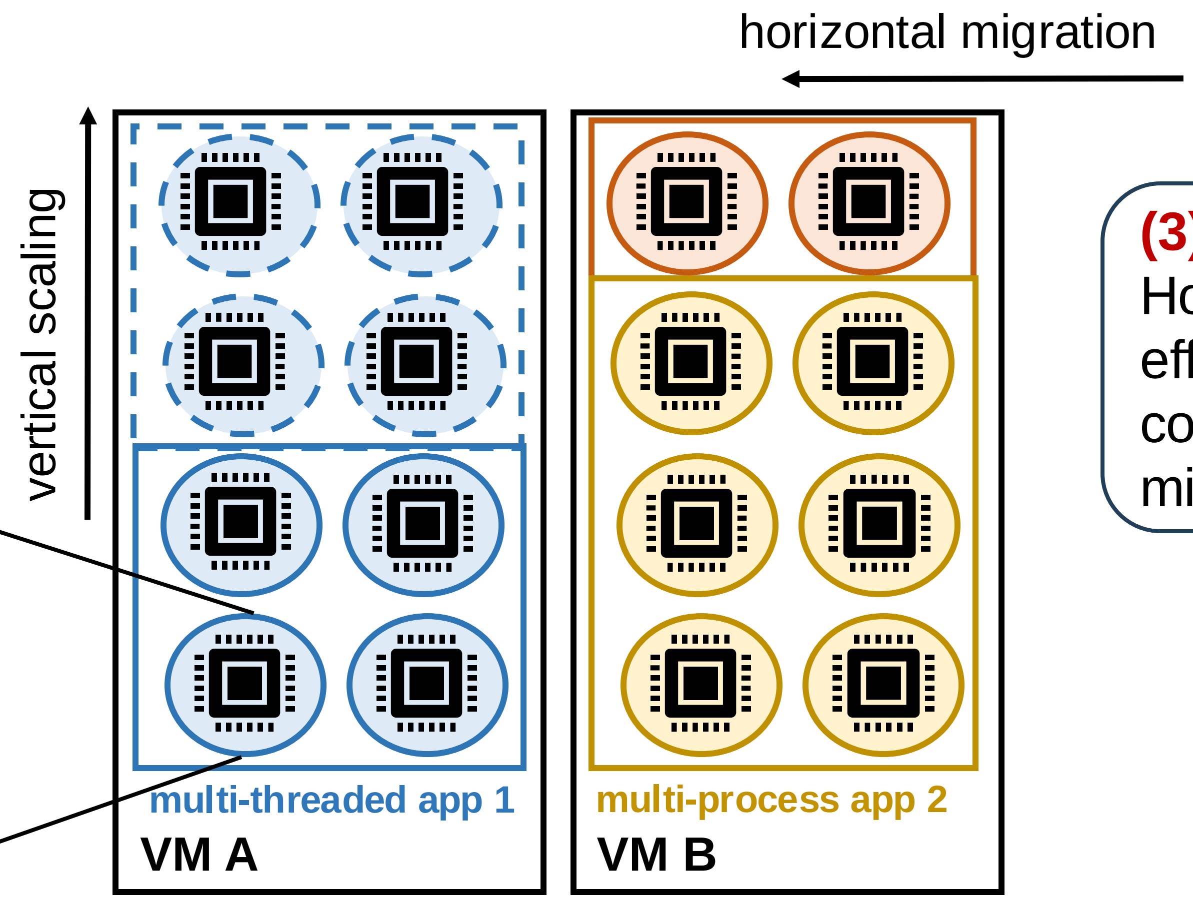


Multi-process apps have Granules spanning multiple VMs

Granule Execution Abstraction

(2) Consistency:
How to ensure memory consistency?

(1) Process/thread semantics:
How to express both thread and process state?



(3) Migration:
How to do efficient and correct Granule migration?

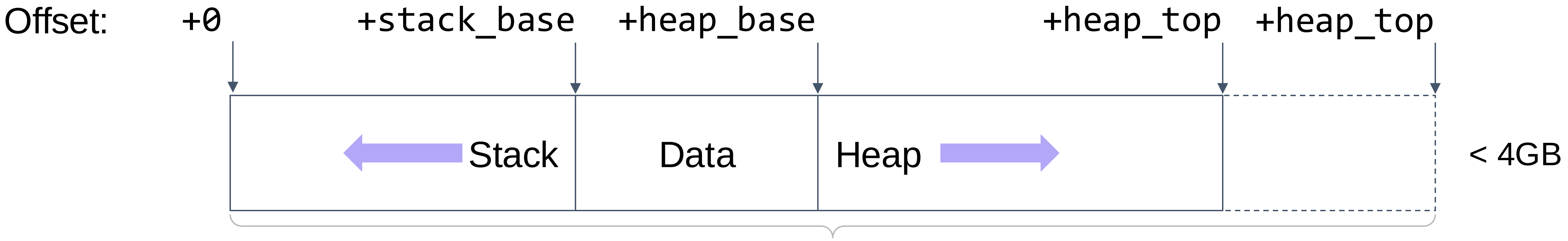
Memory Layout Based on WebAssembly (WASM)

WebAssembly

Proposed for browsers but increasingly used in edge/data centres
- e.g., Fastly, Cloudflare, Krustlet, …

Memory-safe intermediate language
- Prevents instructions from accessing unauthorised memory using software fault isolation (SFI)

Simple linear memory layout for each WASM module:



`std::vector<uint8_t> wasmMemory;`

Memory layout makes Granules efficient to snapshot

Granules with Process and Thread Semantics

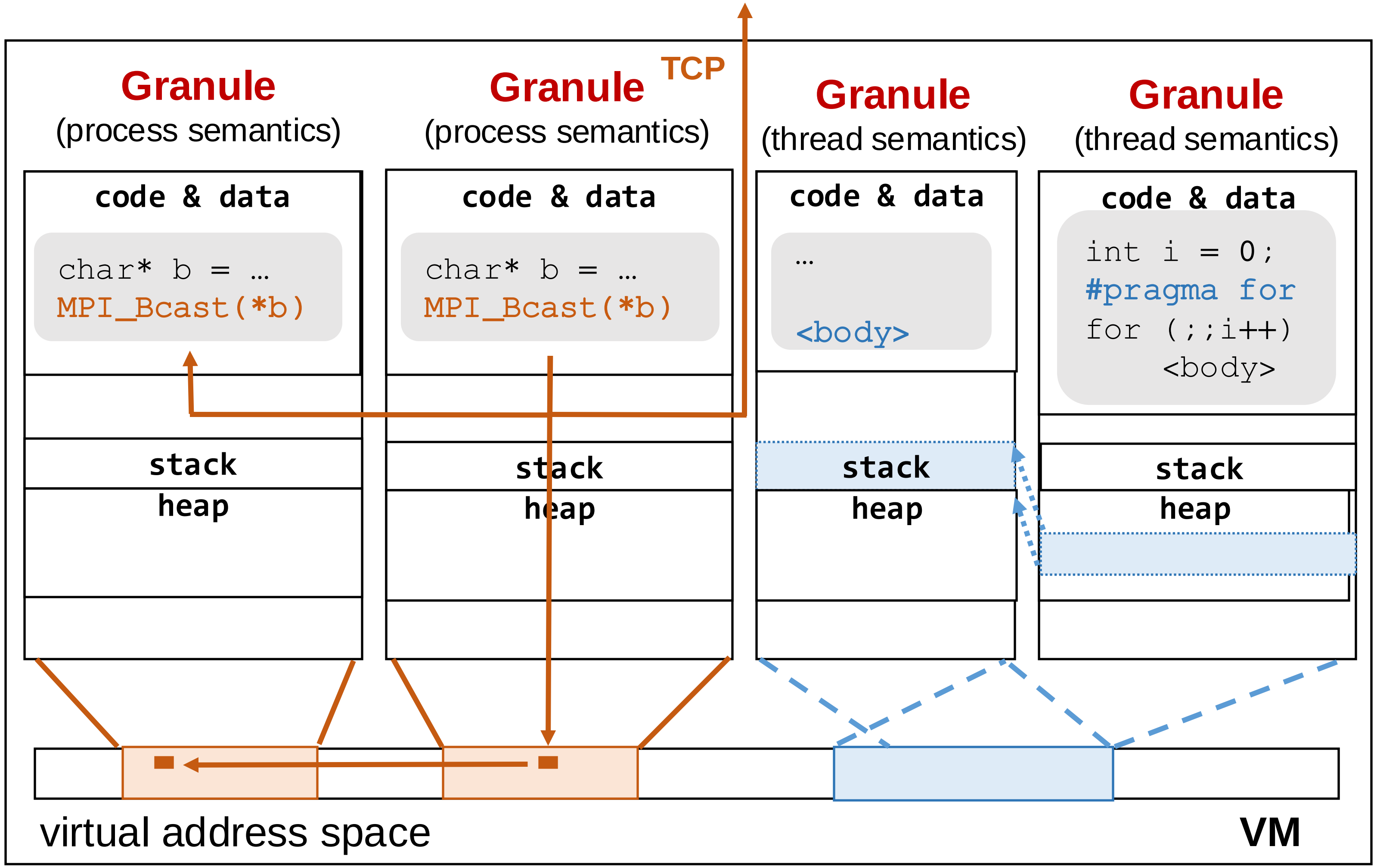
Granules in same VM share single virtual address space

Process semantics:

- **Separate page mapping**
- Send messages
- Runtime manages delivery

Thread semantics:

- **Shared page mapping**
- Operate on shared memory
- Runtime manages synchronisation



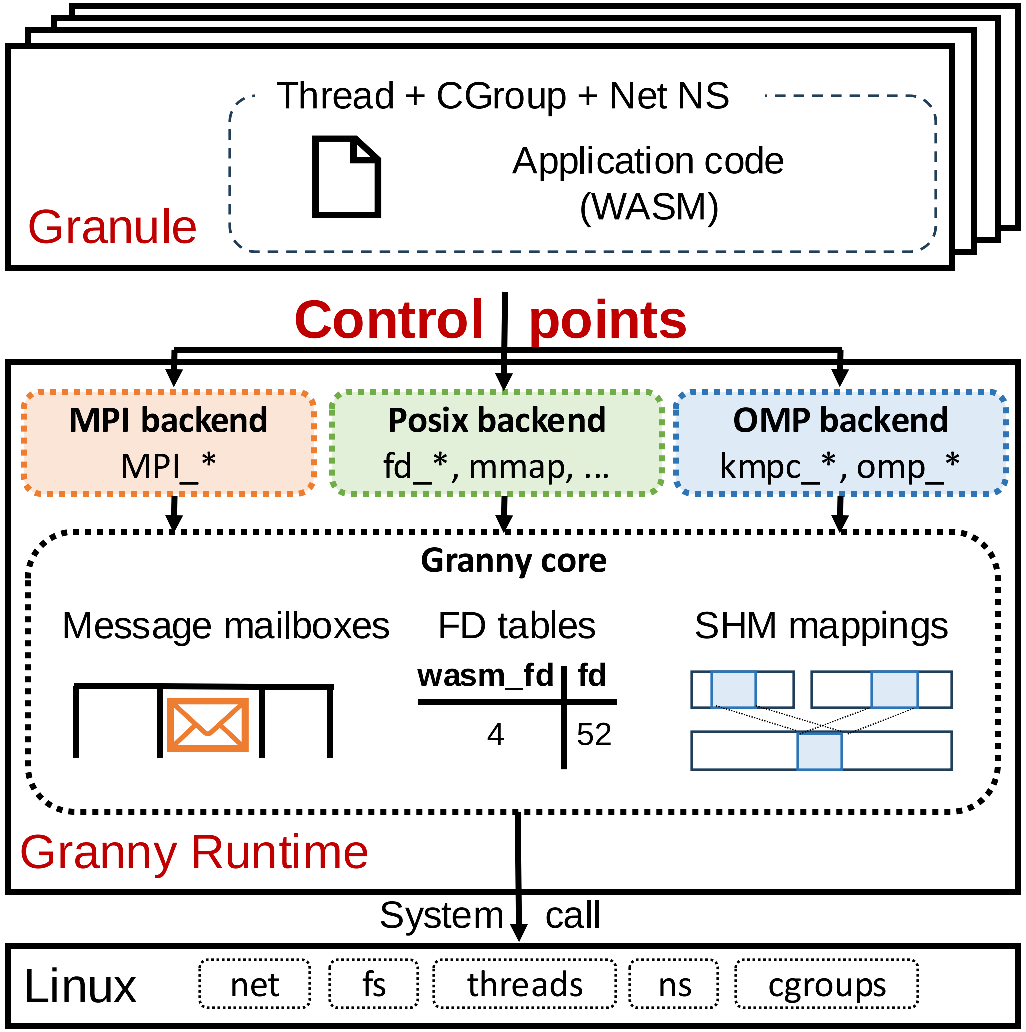
Granny Runtime Implementation

Each Granule executes as OS thread
- Application code isolated as WASM module

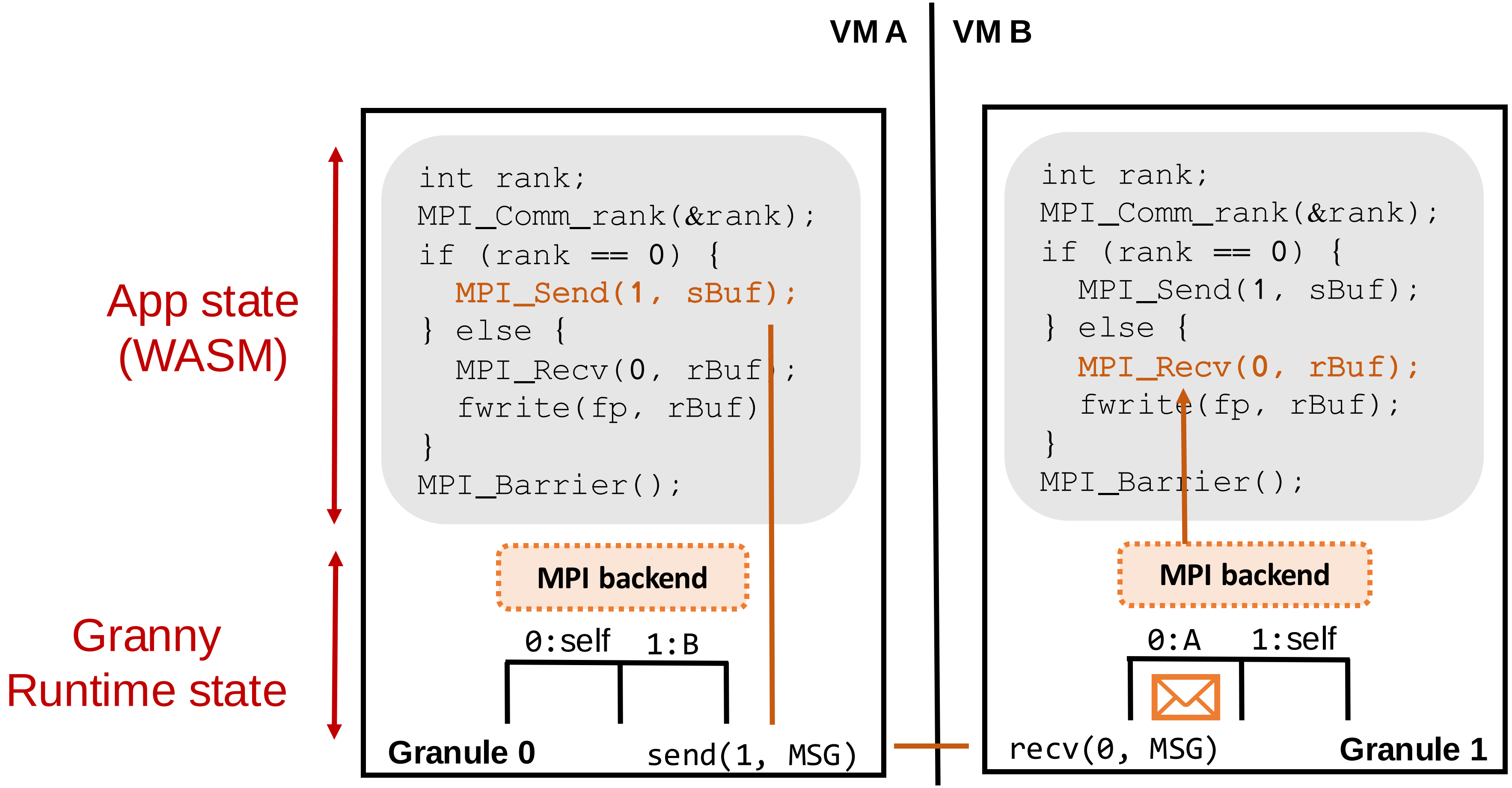
How to ensure consistent Granule state?

Granules interact with Granny Runtime at control points:
- Backends implement specific APIs (OpenMP, MPI, POSIX)
- Granny makes system calls as necessary

Granny Runtime enforces consistent management actions
- No inconsistent thread state
- No in-flight messages that require migration

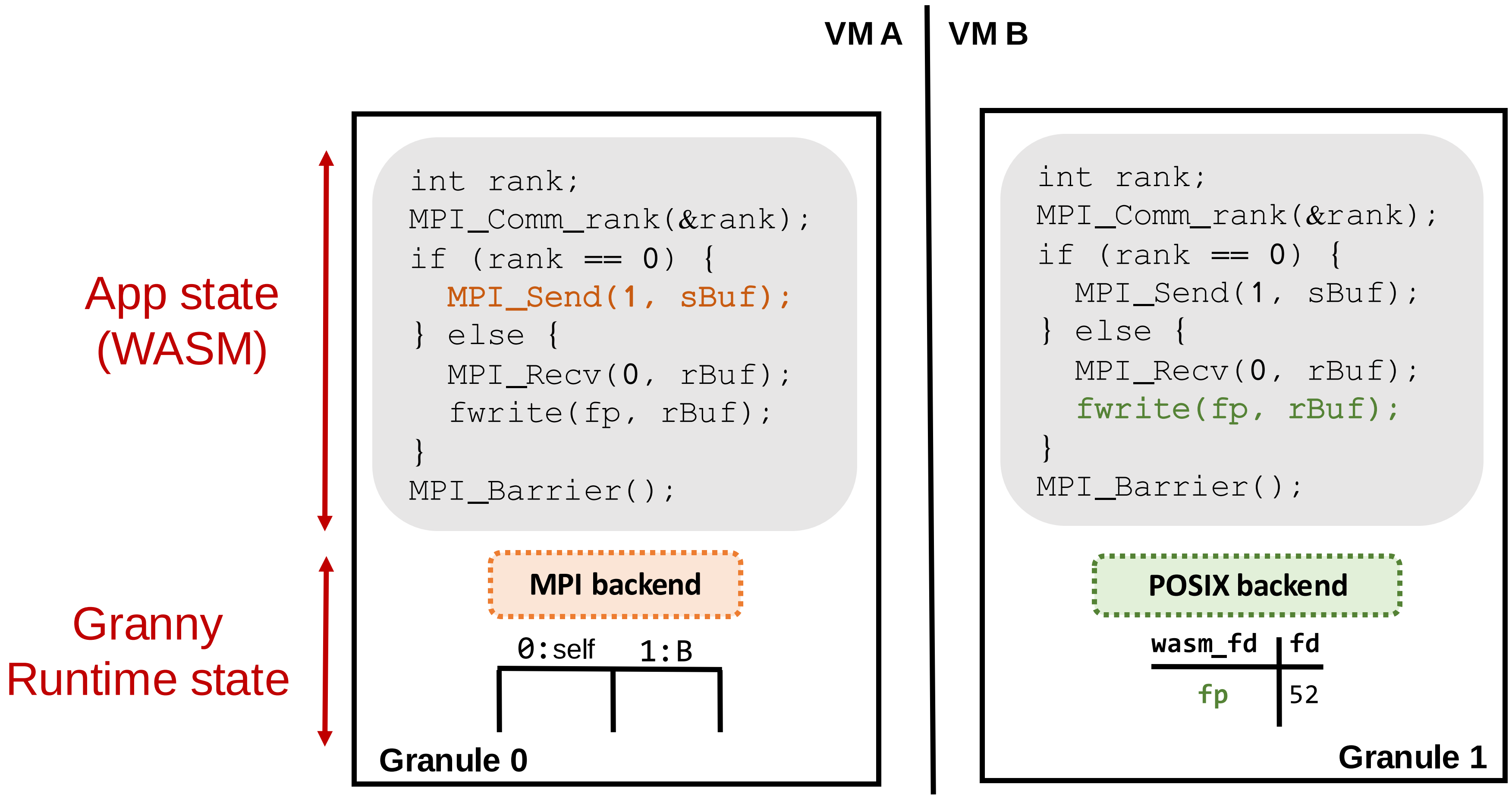


Granules: Capturing State



MPI backend sends/receives messages

Granules: Capturing State



POSIX backend manages files

Migrating Granules

Snapshot Granule at distributed barrier

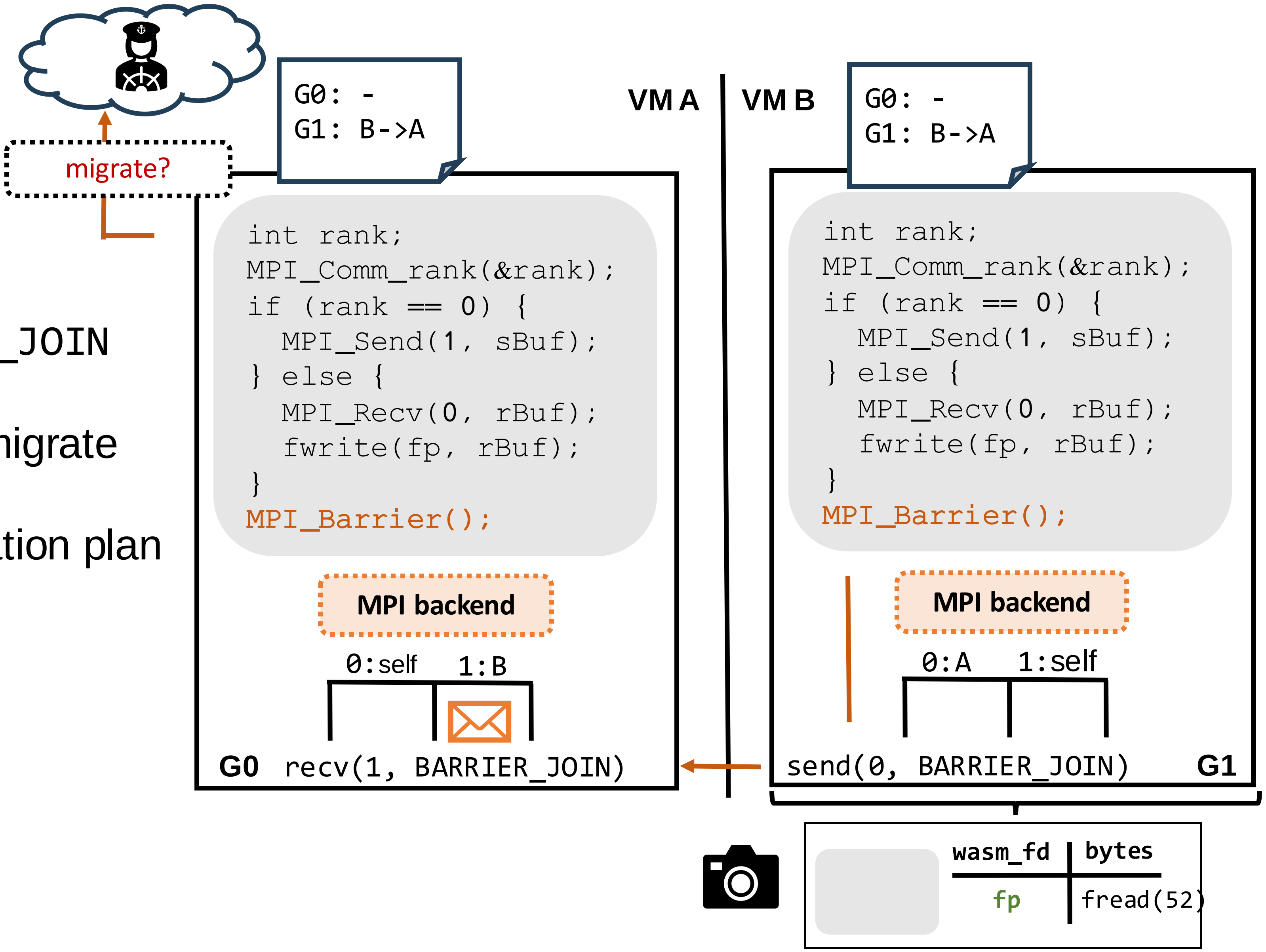
(1) Granules reaches `BARRIER_JOIN`

(2) Cloud provider decides to migrate

(3) Cloud provider sends migration plan

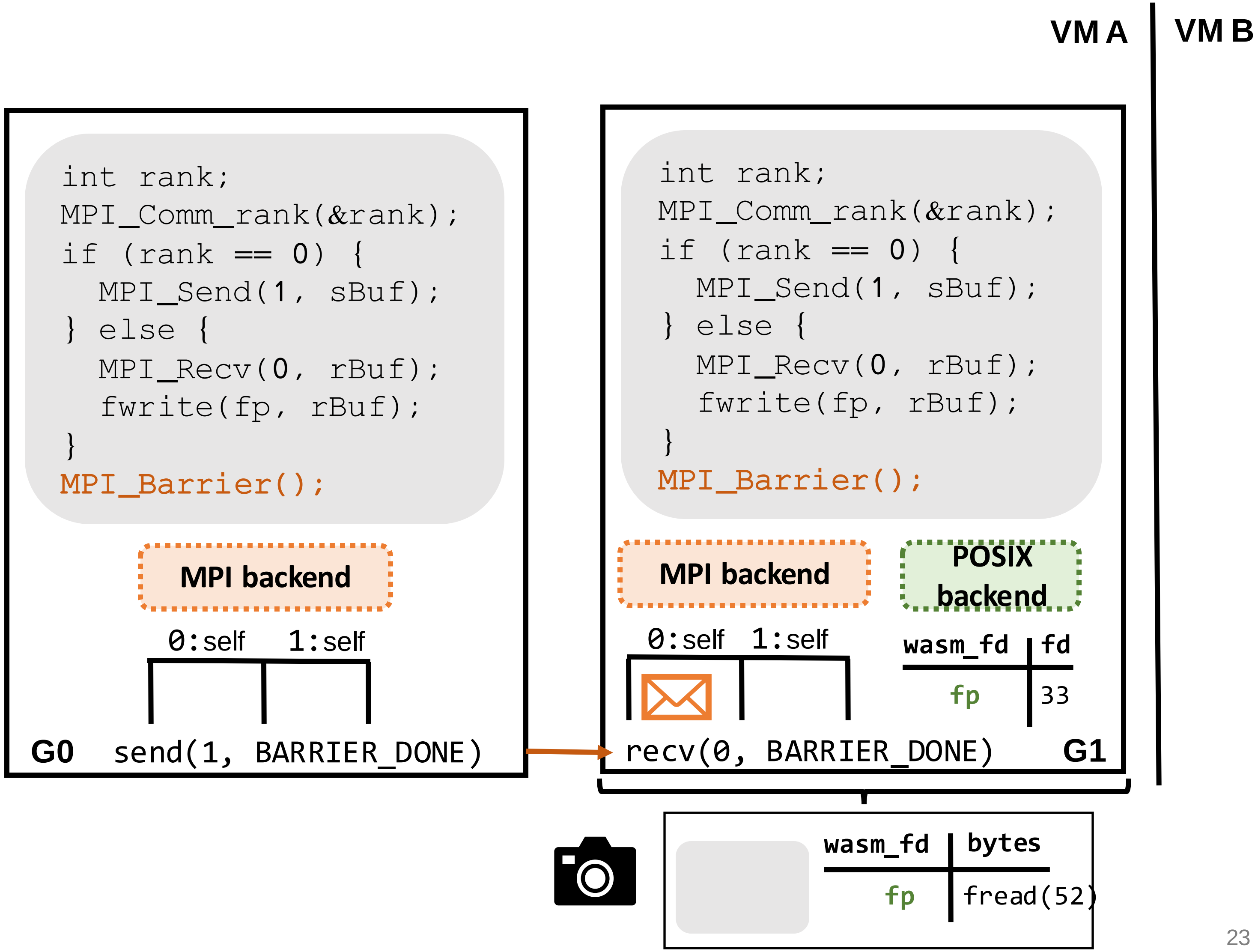
(4) Snapshot Granule

(5) Stop snapshotted Granule



Migrating Granules

Resume Granule on new VM A



(6) Restore Granule from snapshot

(7) Granules resume at barrier

What Management Policies Does Granny Enable?

(1) Compaction Policy
- Trade-off between utilisation and fragmentation

(2) Elastic Policy
- Dynamically scale-up to available vCPUs

(3) Spot VM Policy
- Use spot VMs until eviction

Compare Granny policies to:

Azure Batch scheduler:
- allocates resources at **VM granularity**
- low fragmentation but also low utilisation

Slurm scheduler:
- Allocates resources at **vCPU granularity**
- high utilisation but also high fragmentation

Evaluation Workloads and Metrics

Workloads:

Distributed multi-process (MPI) app: LAMMPS molecule dynamic simulator

Multi-threaded (OpenMP) app: ParRes P2P Kernel on large-scale matrix

Schedule N applications on 32 VMs with 8 vCPUs each

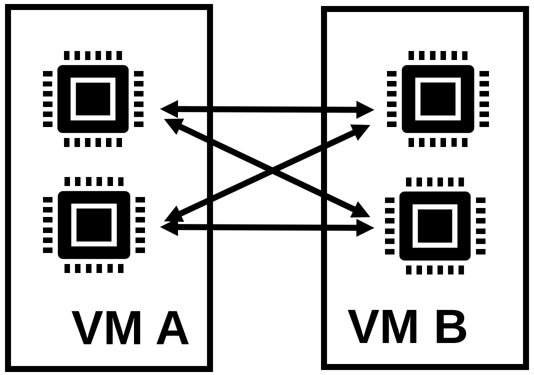
Run applications to completion

Metrics:

Application performance: job completion time (JCT)

Utilisation: percentage of idle vCPUs

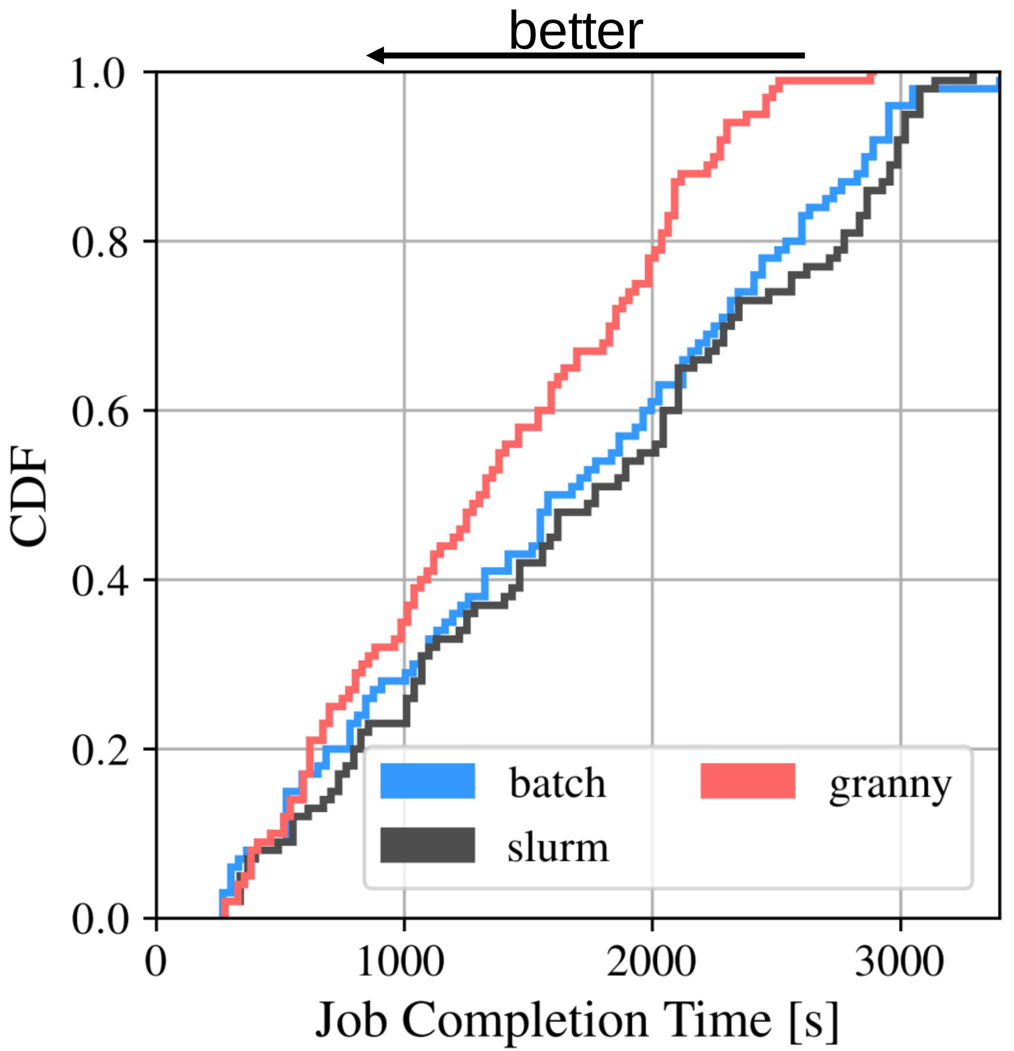
Fragmentation: number of cross-VM communication links



4 cross-VM links

Compaction Policy with Horizontal Migration

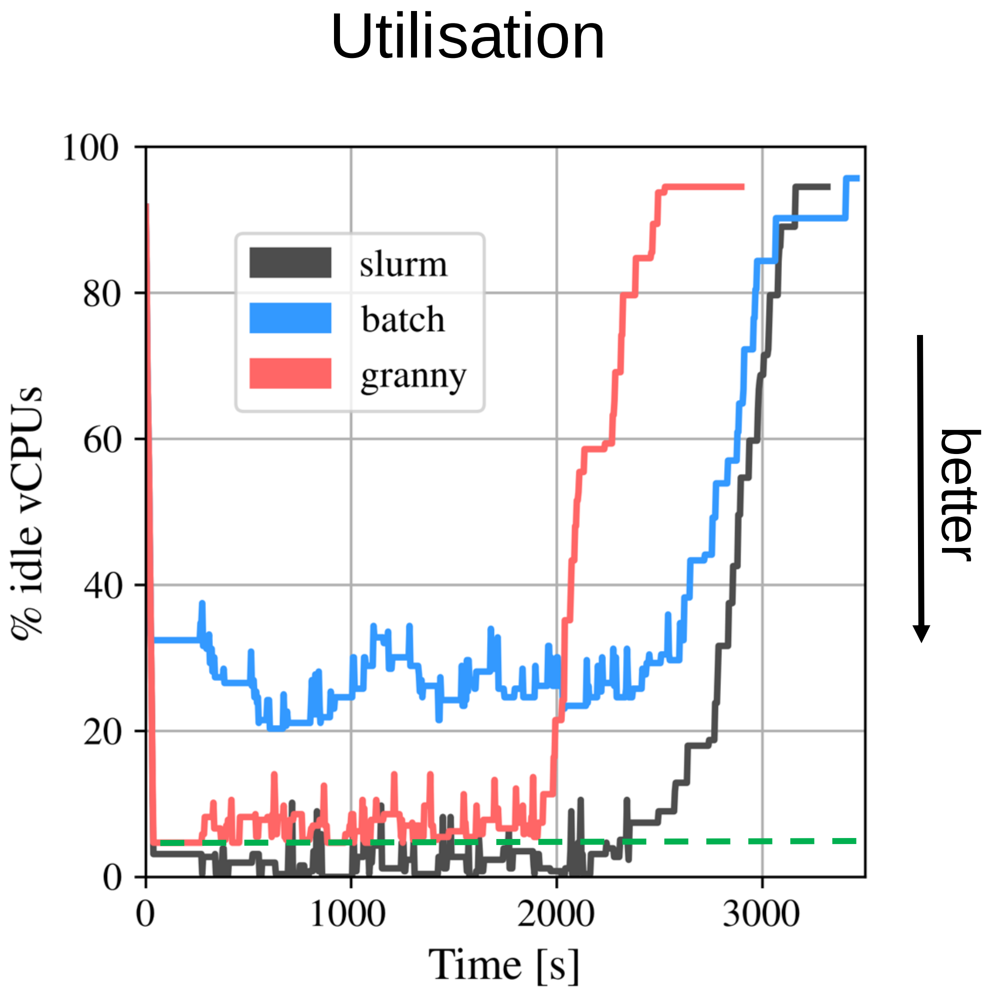
Schedule 100 instances of LAMMPS (distributed multi-process) eagerly

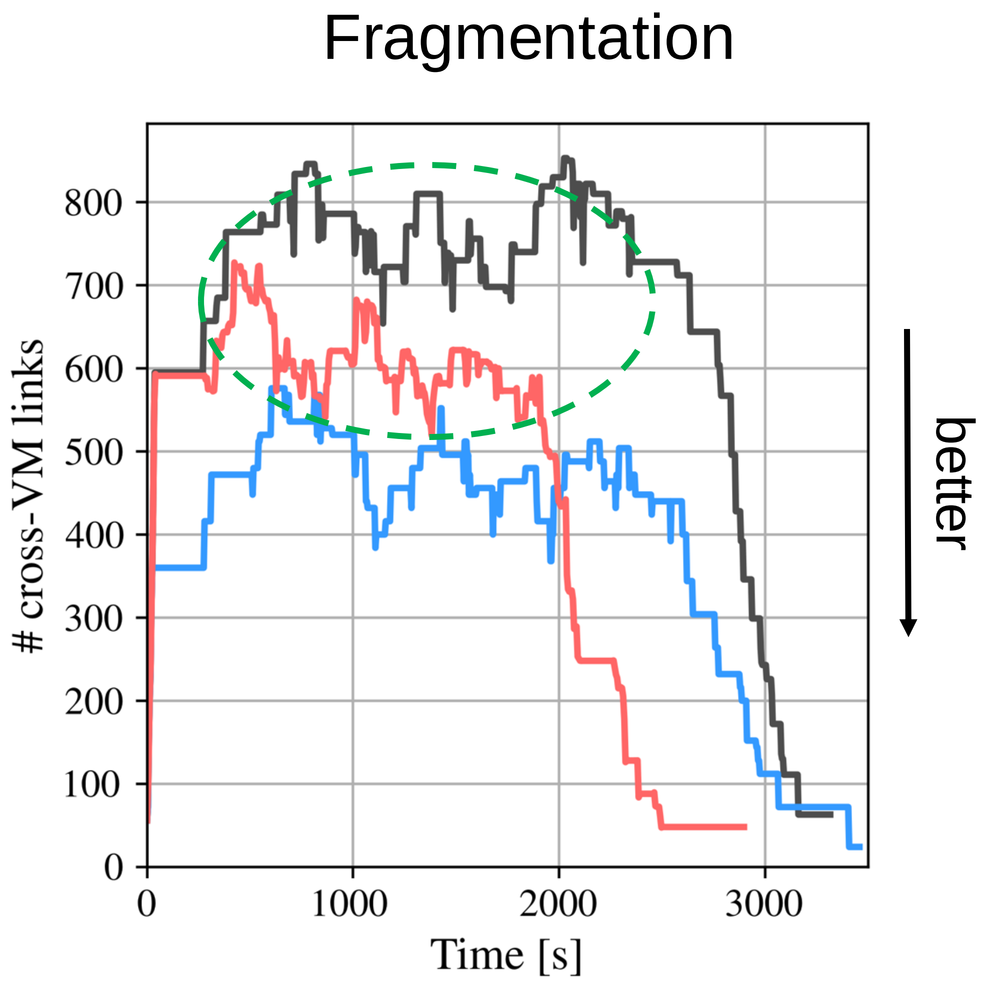


Granny leaves 5% of vCPUs unallocated to enable migration

By improving locality through migration, Granny lowers median and tail JCTs

Compaction Policy: Utilisation and Fragmentation

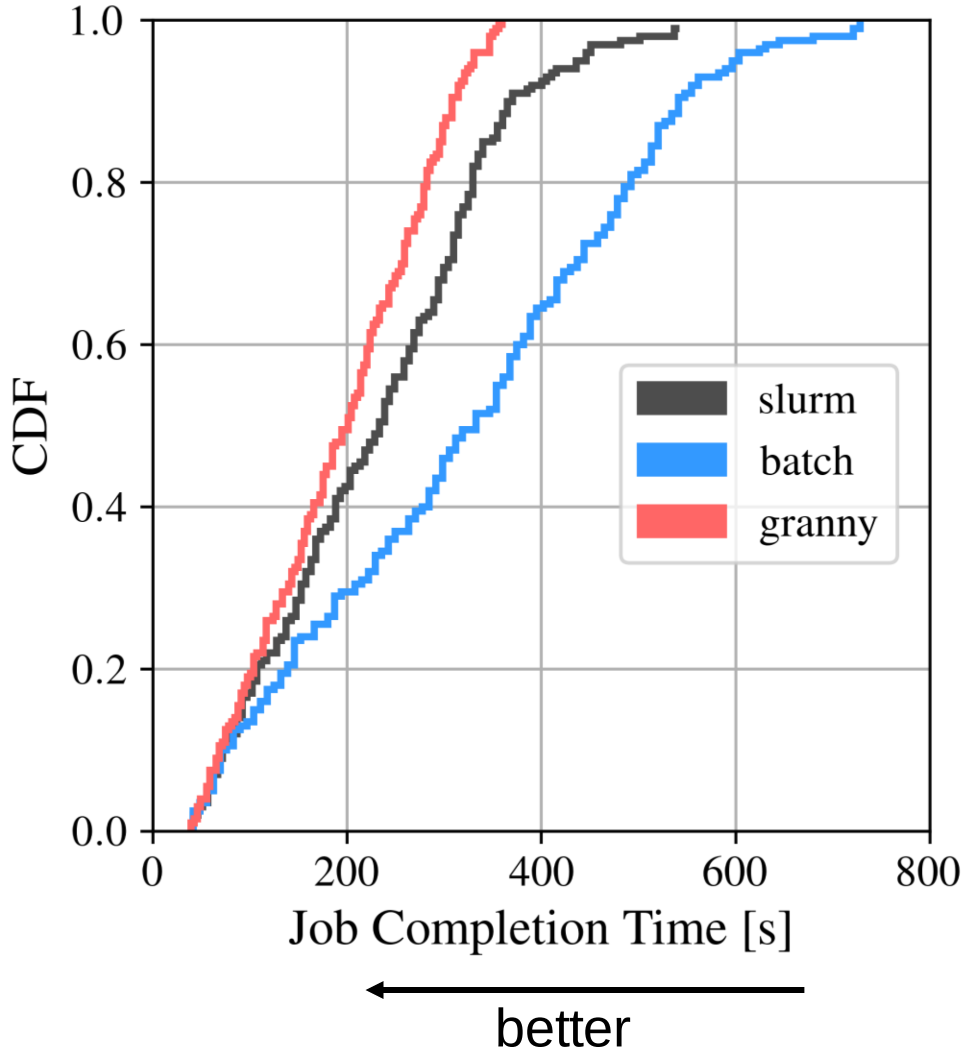




With only 5% of vCPUs idle, Granny reduces fragmentation by 25%

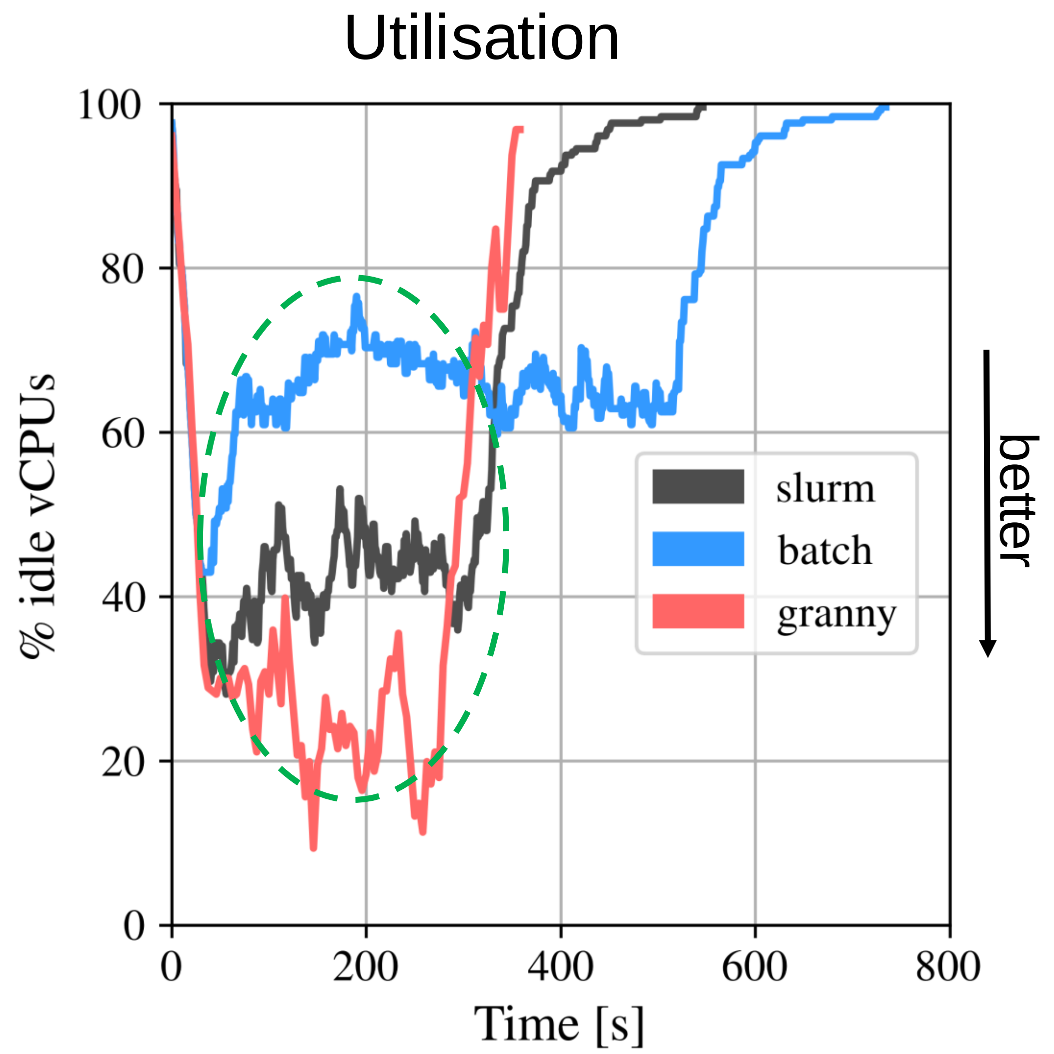
Elastic Policy for Multi-Threaded Apps

Schedule 100 instances of ParRes P2P (multi-threaded) eagerly



Granny uses idle vCPUs elastically to lower median and tail JCTs

Elastic Policy: Utilisation



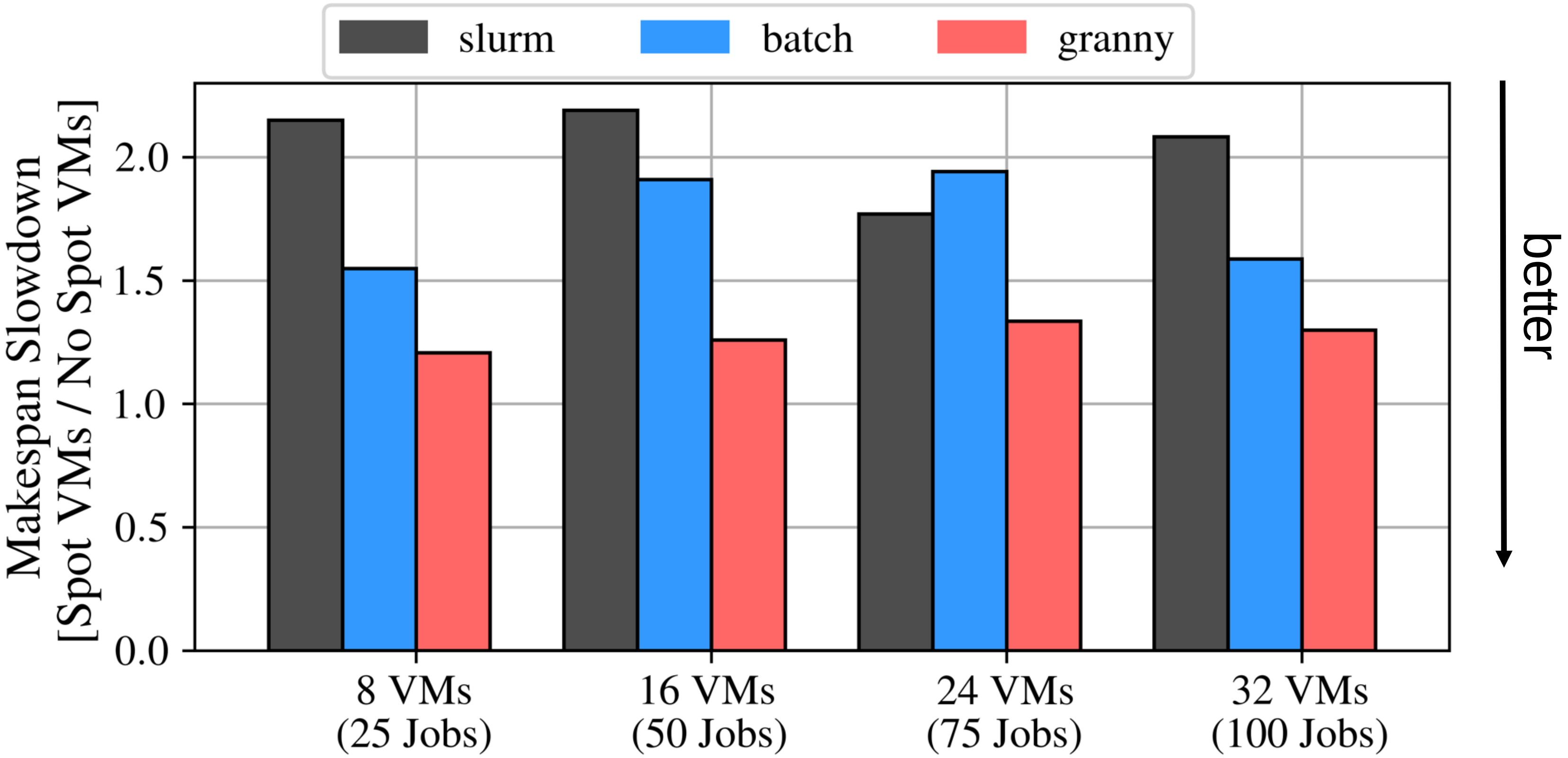
Bin-packing leaves >40% of vCPUs idle, and Granny exploits this with elastic scaling

Spot VM Policy with Eviction Migration

Deploy LAMMPS (distributed multi-process) with spot VMs

- Spot VMs have 1 min eviction grace period

Granules migrate away from evicted VMs



Granny's proactive migration loses less work under spot evictions

Towards Granular Management in Serverless Clouds

Serverless is an exciting cloud computing with lots of potential
- There are opportunities in rethinking execution abstractions for serverless
- POSIX is a blessing and a curse

Example: Granules to enable flexible management policies
- Granules combines multi-threaded and multi-process execution
- Granules enables low overhead elastic scaling + dynamic migration

What are other serverless execution models?

github.com/faasm/faasm
github.com/faasm/granny-experiments

Granny NSDI'25 paper:

Thank You! Peter Pietzuch <prp@imperial.ac.uk>