

# Making Serverless Computing More Serverless

Zaid Al-Ali, Sepideh Goodarzy, Ethan Hunter, Sangtae Ha, Richard Han, Eric Keller  
*University of Colorado Boulder*

Eric Rozner  
*IBM Research*



University of Colorado  
Boulder

**IBM**  
**Research**

# Serverless Background

- Serverless offerings are widespread today



Amazon Lambda



Google Cloud Functions



IBM Cloud Functions



Azure Functions



Apache OpenWhisk



OpenFaaS



fission



Dispatch

# Today's serverless abstraction

- Serverless typically means Function-as-a-Service

# Today's serverless abstraction

- Serverless typically means Function-as-a-Service

## Benefits

No need to manage infrastructure

Simple scale-out support

Low cost

# Today's serverless abstraction

- Serverless typically means Function-as-a-Service

Benefits	Shortcomings
No need to manage infrastructure	Event-driven frameworks only
Simple scale-out support	Stateless
Low cost	Function instance bound to server

# Today's serverless abstraction

- Serverless typically means Function-as-a-Service

Benefits	Shortcomings
Limit types of applications supported by serverless	Event-driven frameworks only
	Stateless
Low cost	Function instance bound to server

# Today's serverless abstraction

- Serverless typically means Function-as-a-Service

Benefits	Shortcomings
Limit types of applications supported by serverless	Event-driven frameworks only
	Stateless
Low cost	Function instance bound to server

This talk: propose new serverless abstraction and overview its design

# Today's serverless abstraction

## Making Serverless Computing More Serverless

FaaS, Lambda, OpenWhisk, ...

Break limitations of single server

Zaid Al-Ali<sup>†</sup>, Sepideh Goodarzy<sup>†</sup>, Ethan Hunter<sup>†</sup>, Sangtae Ha<sup>†</sup>, Richard Han<sup>†</sup>, Eric Keller<sup>†</sup>, Eric Rozner<sup>\*</sup>

<sup>†</sup>University of Colorado Boulder; <sup>\*</sup>IBM Research

**Abstract**—In serverless computing, developers define a function to handle an event, and the serverless framework horizontally scales the application as needed. The downside of this function-based abstraction is it limits the type of application supported and places a bound on the function to be within the physical resource limitations of the server the function executes on. In this paper we propose a new abstraction

through memory or storage. The challenge, of course, is realizing a process-based serverless framework which can map our serverless process abstraction to an underlying, physically distributed infrastructure. To that end, we propose a new architecture called ServerlessOS to enable our vision and argue three key components are necessary to make our

Low cost

Function instance bound to server

This talk: propose new serverless abstraction and overview its design



# A new serverless abstraction

- Expand serverless beyond the bounds of FaaS
- Goals of our new abstraction:



# A new serverless abstraction

- Expand serverless beyond the bounds of FaaS
- Goals of our new abstraction:






Flexible enough to support general set of applications





# A new serverless abstraction

- Expand serverless beyond the bounds of FaaS
- Goals of our new abstraction:
  -  Flexible enough to support general set of applications
  -  Familiar to developers, operating systems, and admins

# A new serverless abstraction

- Expand serverless beyond the bounds of FaaS
- Goals of our new abstraction:
  -  Flexible enough to support general set of applications
  -  Familiar to developers, operating systems, and admins
  -  Easy to transition existing codebases to serverless

# A new serverless abstraction

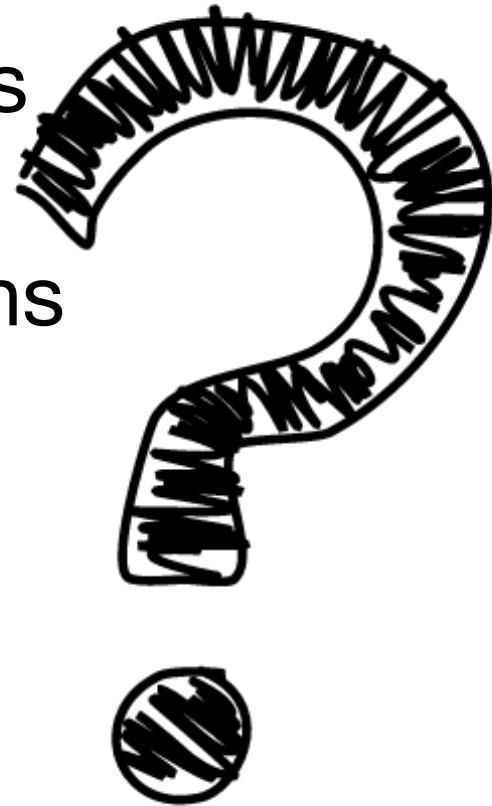
- Expand serverless beyond the bounds of FaaS
- Goals of our new abstraction:
  -  Flexible enough to support general set of applications
  -  Familiar to developers, operating systems, and admins
  -  Easy to transition existing codebases to serverless
  -  Same simplicity and scale-out as FaaS

# A new serverless abstraction

- ✓ Flexible enough to support general set of applications
- ✓ Familiar to developers, operating systems, and admins
- ✓ Easy to transition existing codebases to serverless
- ✓ Same simplicity and scale-out as FaaS

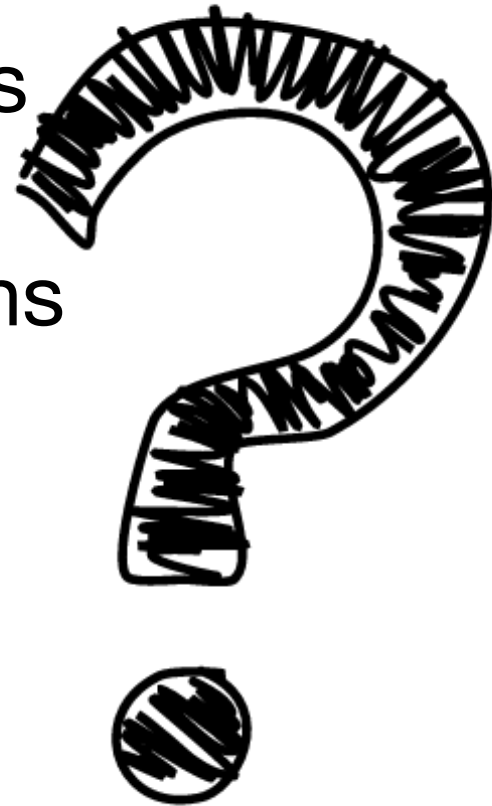
# A new serverless abstraction

- ✓ Flexible enough to support general set of applications
- ✓ Familiar to developers, operating systems, and admins
- ✓ Easy to transition existing codebases to serverless
- ✓ Same simplicity and scale-out as FaaS



# A new serverless abstraction

- ✓ Flexible enough to support general set of applications
- ✓ Familiar to developers, operating systems, and admins
- ✓ Easy to transition existing codebases to serverless
- ✓ Same simplicity and scale-out as FaaS



A Process!



A Process!

# A Process!

- ✓ Flexible enough to support general set of applications
  - ★ Multiple threads, I/O via sockets, persist state, ...

# A Process!

- ✓ Flexible enough to support general set of applications
  - ★ Multiple threads, I/O via sockets, persist state, ...
- ✓ Familiar to developers, operating systems, and admins
  - ★ Abstraction already used today in non-serverless

# A Process!

- ✓ Flexible enough to support general set of applications
  - ★ Multiple threads, I/O via sockets, persist state, ...
- ✓ Familiar to developers, operating systems, and admins
  - ★ Abstraction already used today in non-serverless
- ✓ Easy to transition existing codebases to serverless
  - ★ Pool of CPU, I/O, memory, storage: *server* → *datacenter*

# A Process!

- ✓ Flexible enough to support general set of applications
  - ★ Multiple threads, I/O via sockets, persist state, ...
- ✓ Familiar to developers, operating systems, and admins
  - ★ Abstraction already used today in non-serverless
- ✓ Easy to transition existing codebases to serverless
  - ★ Pool of CPU, I/O, memory, storage: *server* → *datacenter*
- ✓ Same simplicity and scale-out as FaaS



# A Process!

✓ Flexible enough to support general set of applications

★ Multiple threads, I/O via sockets, persist state, ...

✓ Familiar to developers, operating systems, and admins

★ Abstraction already used today in non-serverless

✓ Easy to transition existing codebases to serverless

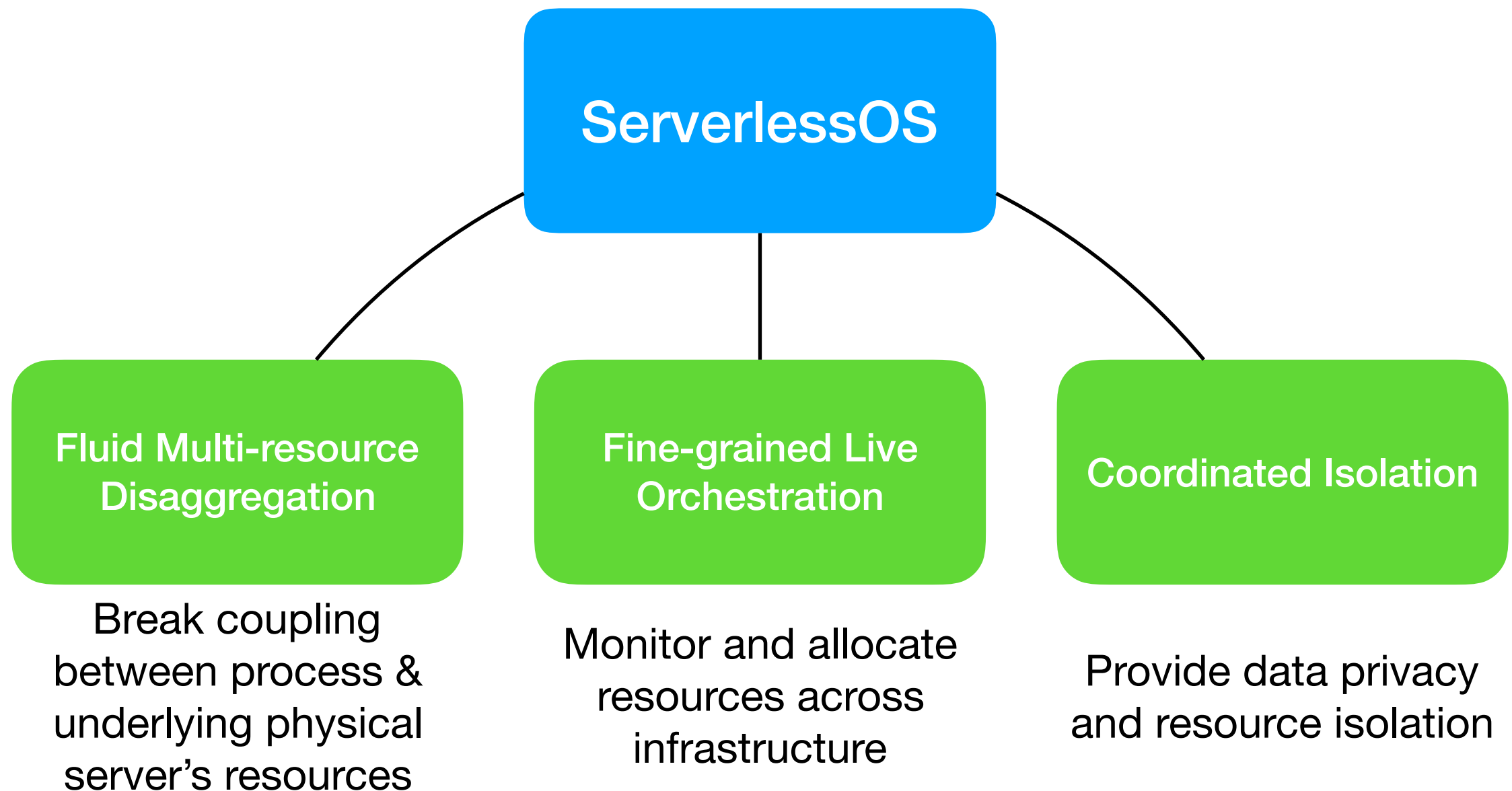
★ Pool of CPU, I/O, memory, storage: *server* → *datacenter*

✗ Same simplicity and scale-out as FaaS

Challenge: map serverless process abstraction to underlying physically distributed architecture

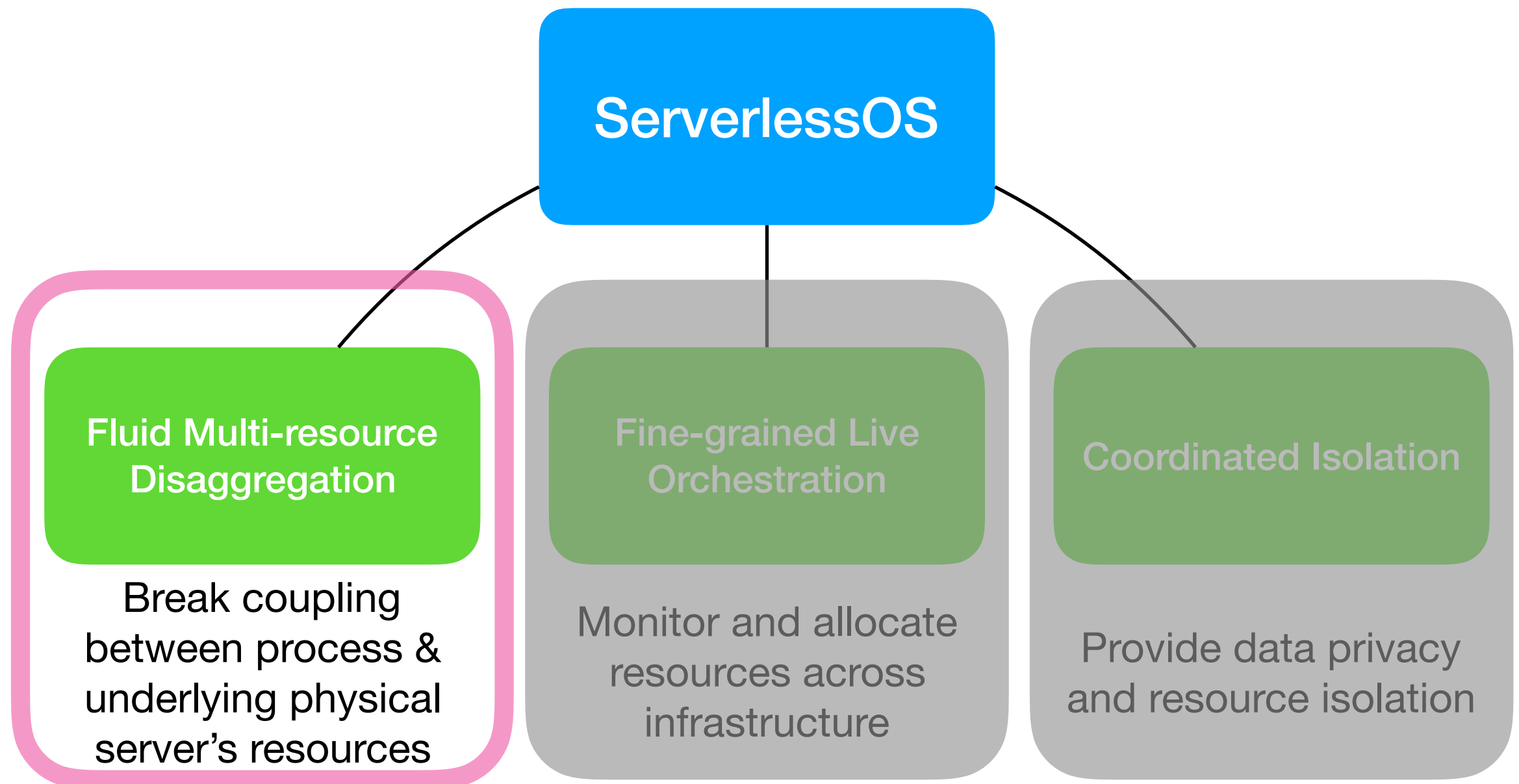
# Outline of talk

- Goal: provide seamless, scale-out process abstraction
- This talk: high-level outline of our *ServerlessOS* vision



# Outline of talk

- Goal: provide seamless, scale-out process abstraction
- This talk: high-level outline of our *ServerlessOS* vision

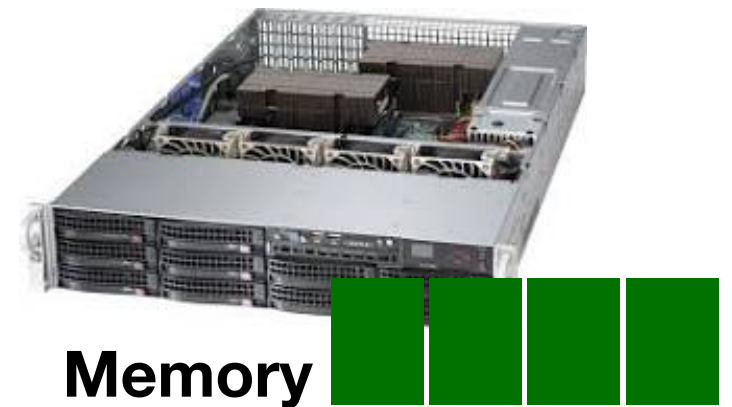
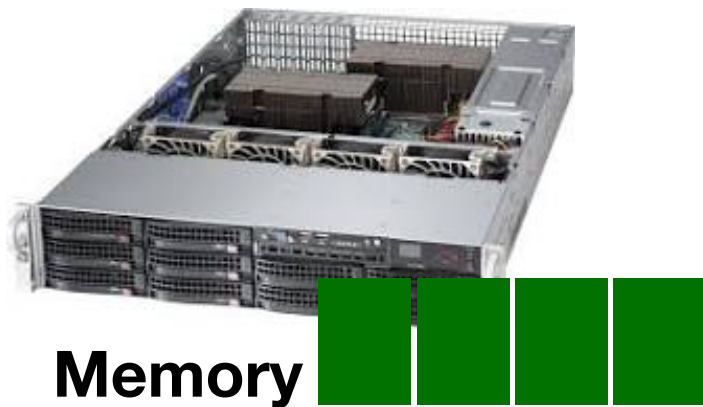




# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

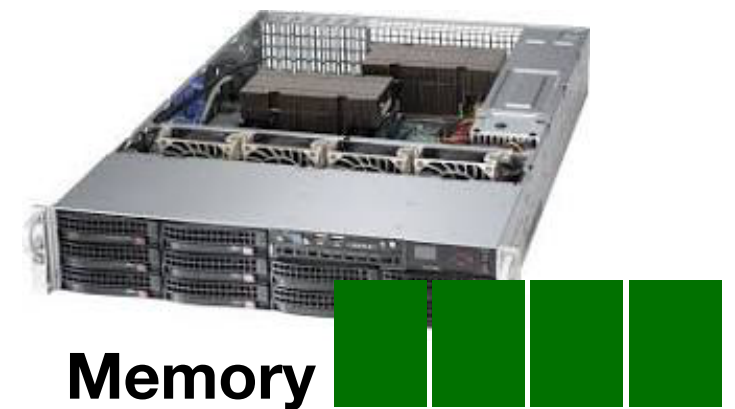
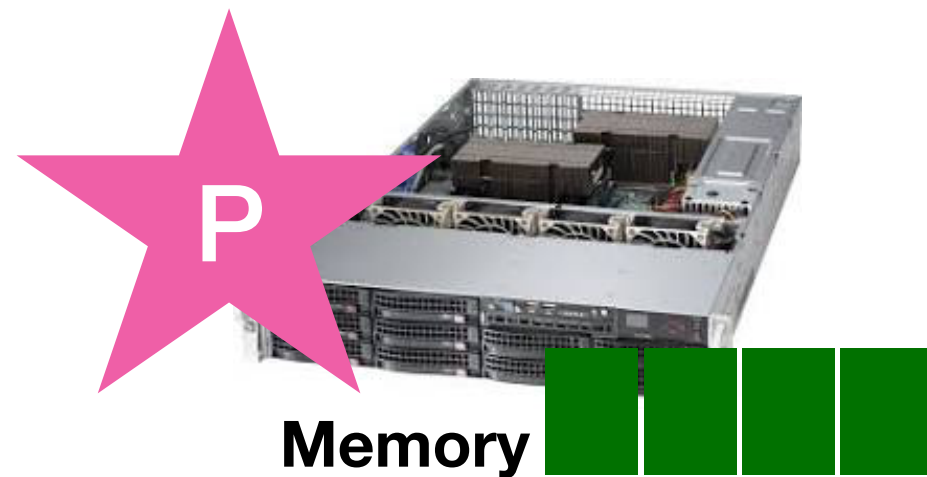
Fluid multi-resource disaggregation



# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

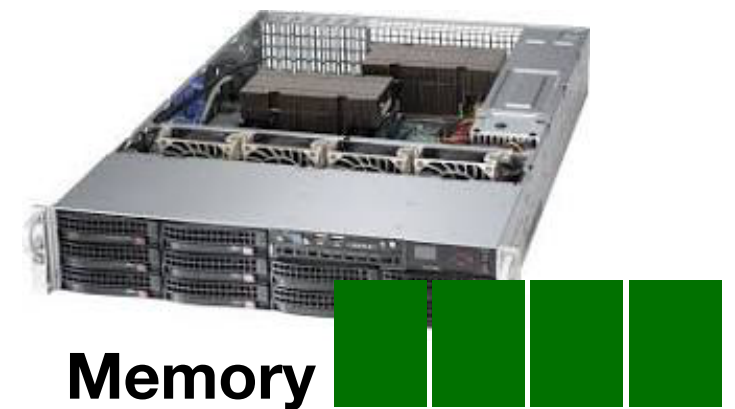
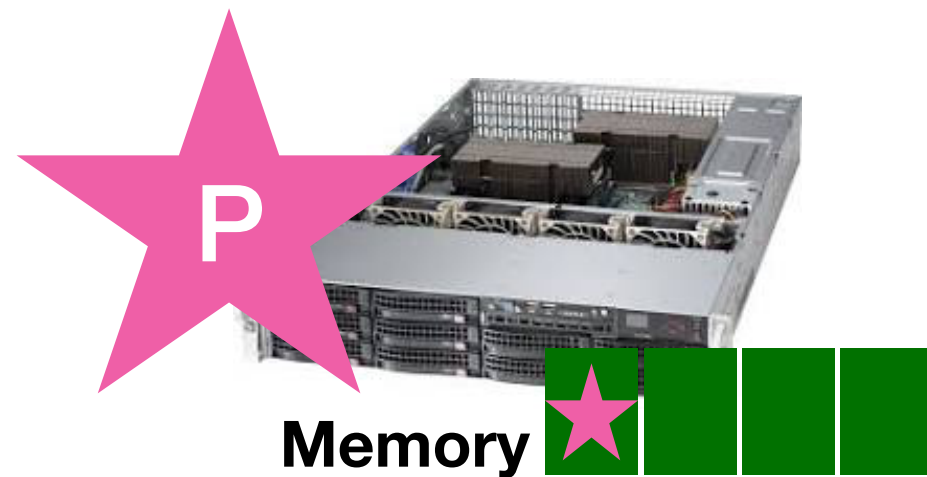
Fluid multi-resource disaggregation



# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

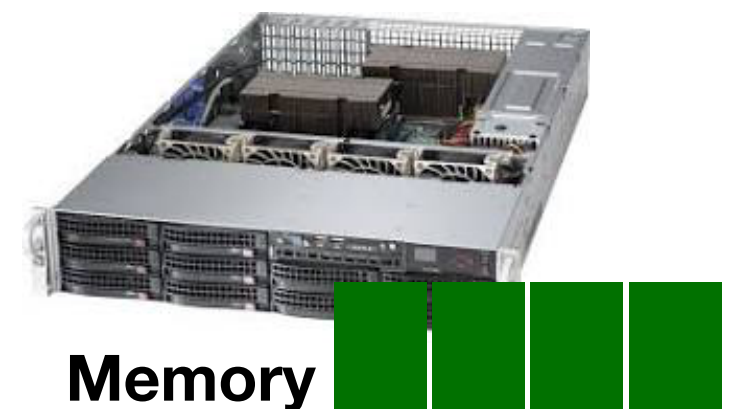
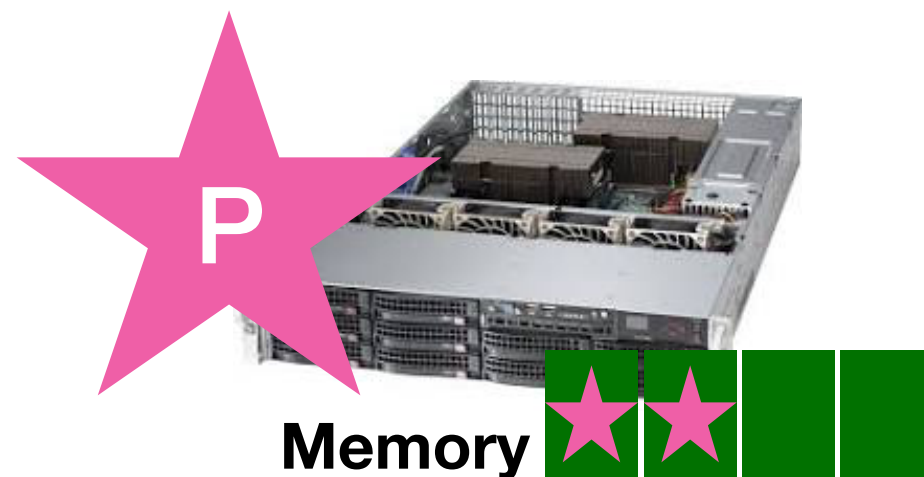
Fluid multi-resource disaggregation



# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

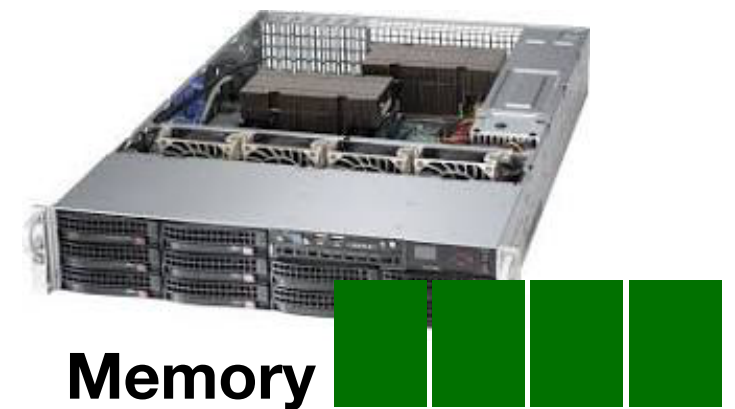
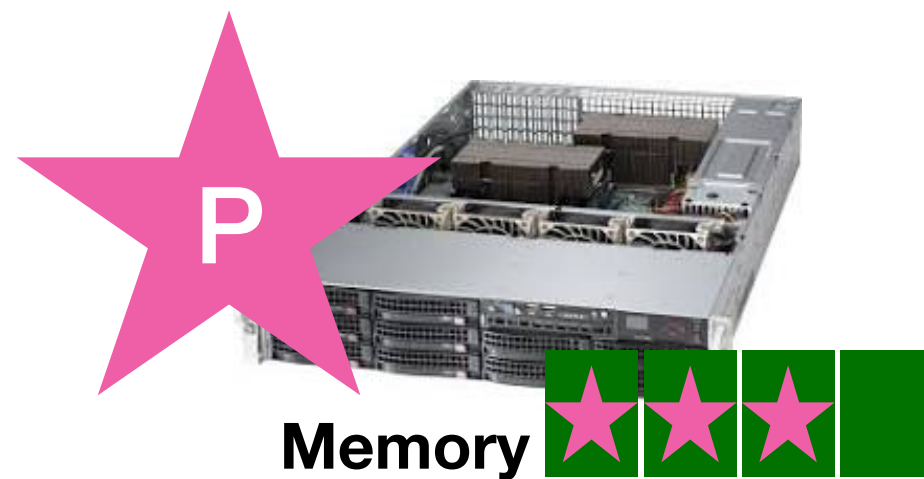
Fluid multi-resource disaggregation



# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

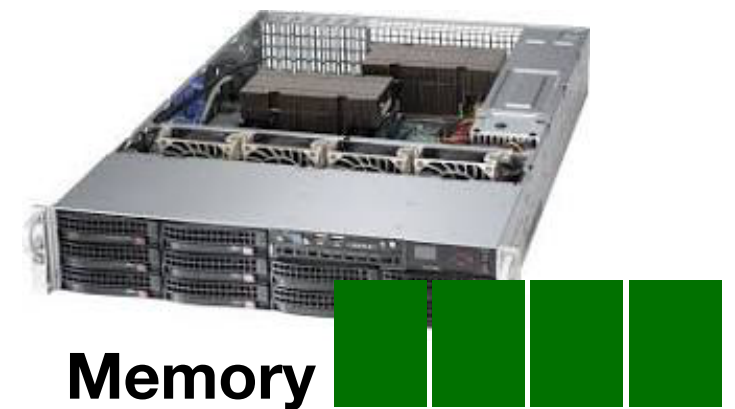
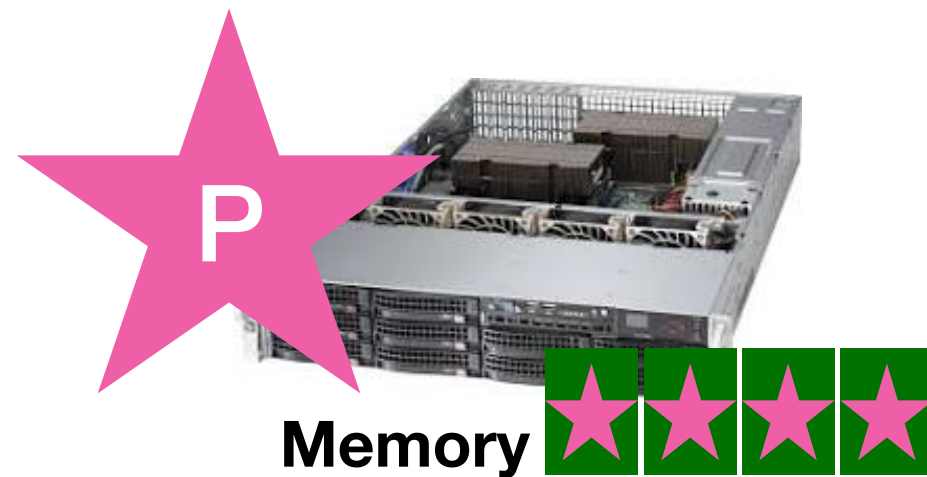
Fluid multi-resource disaggregation



# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

Fluid multi-resource disaggregation



# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

## Fluid multi-resource disaggregation



# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

## Fluid multi-resource disaggregation





# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

## Fluid multi-resource disaggregation



# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

## Fluid multi-resource disaggregation

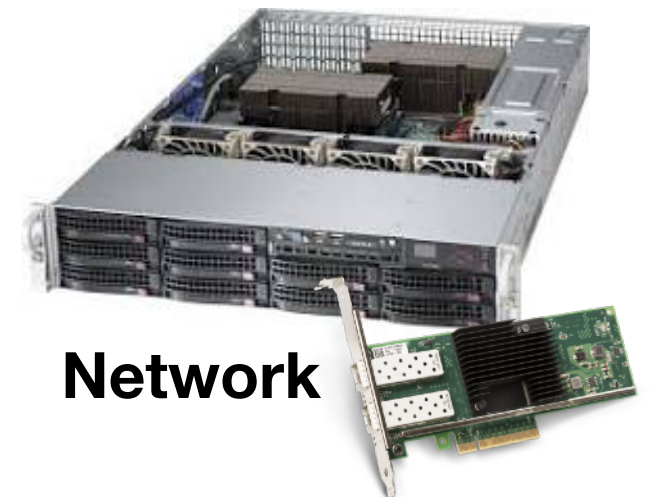
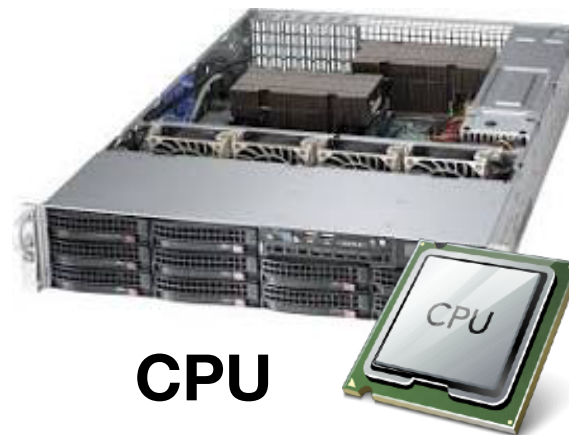
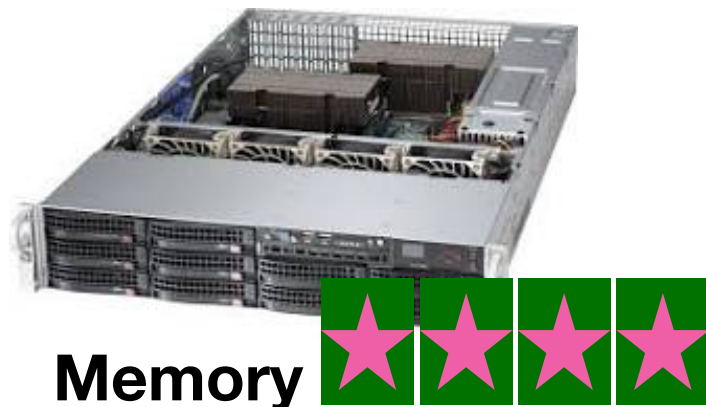


Disaggregation: decouple process's resources from single server

# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

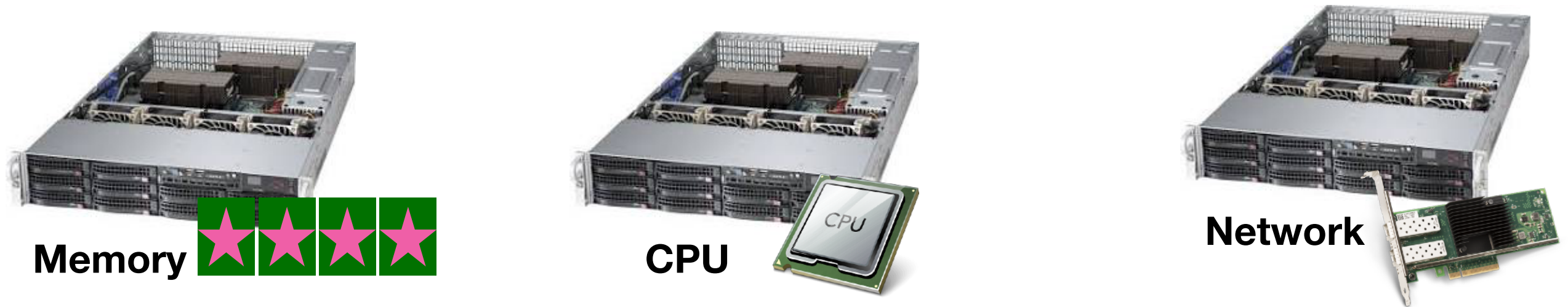
## Fluid multi-resource disaggregation



# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

## Fluid multi-resource disaggregation

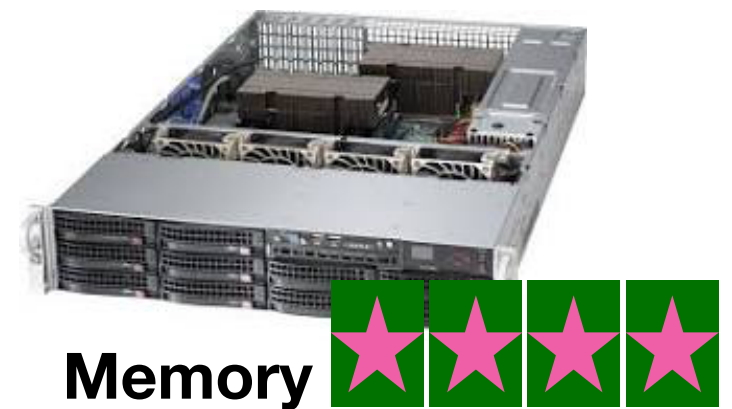
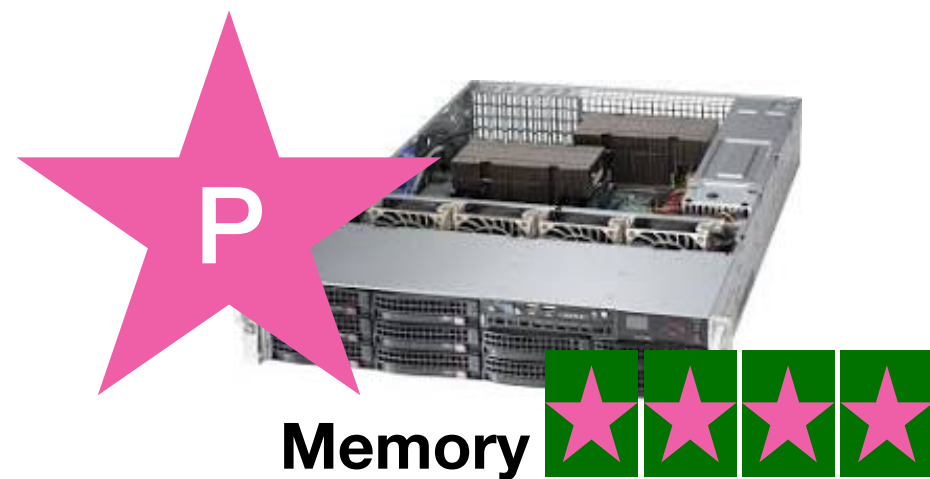


Decouple memory, compute, I/O to increase flexibility

# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

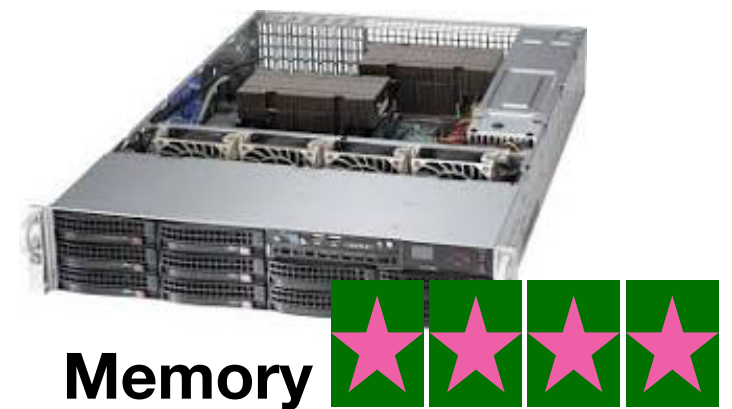
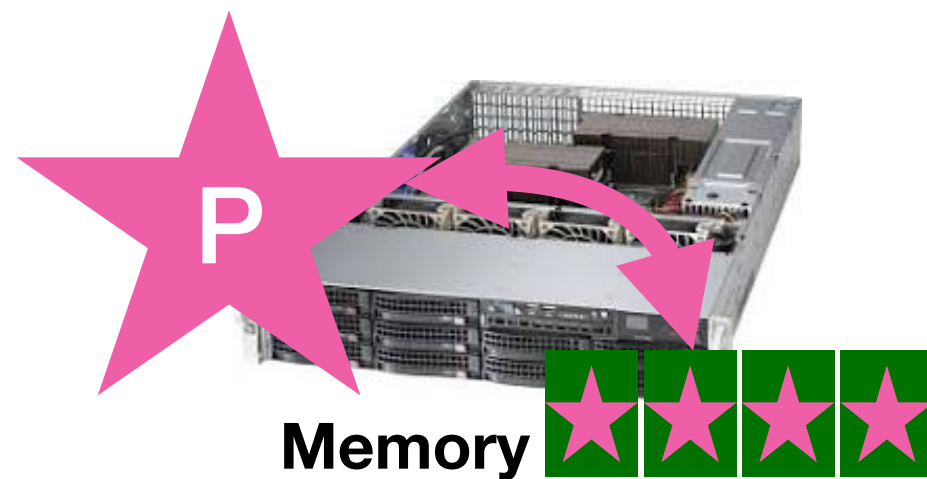
## Fluid multi-resource disaggregation



# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

## Fluid multi-resource disaggregation

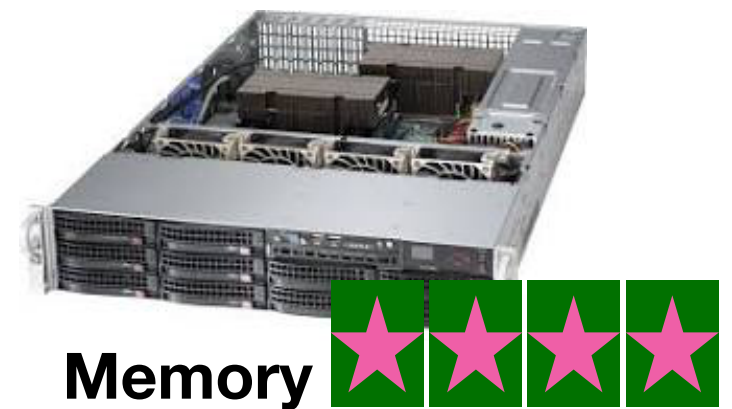
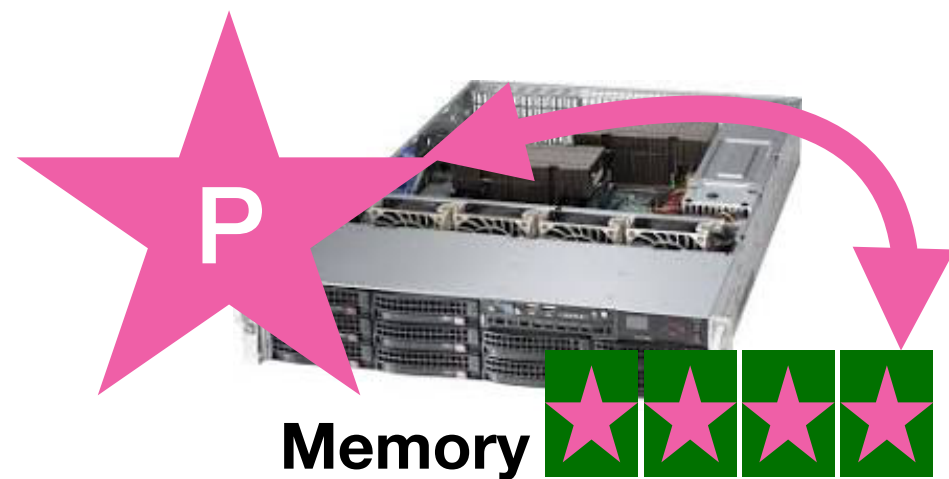




# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

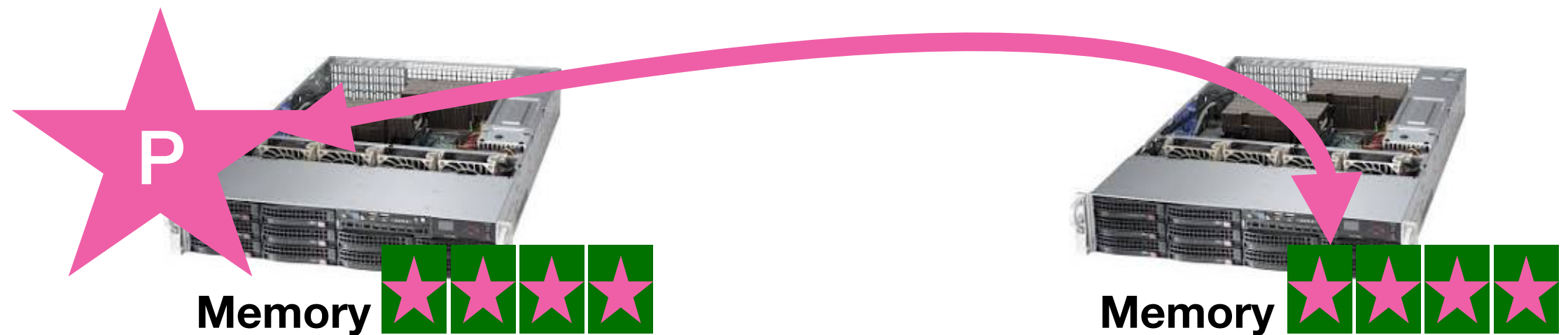
## Fluid multi-resource disaggregation



# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

## Fluid multi-resource disaggregation



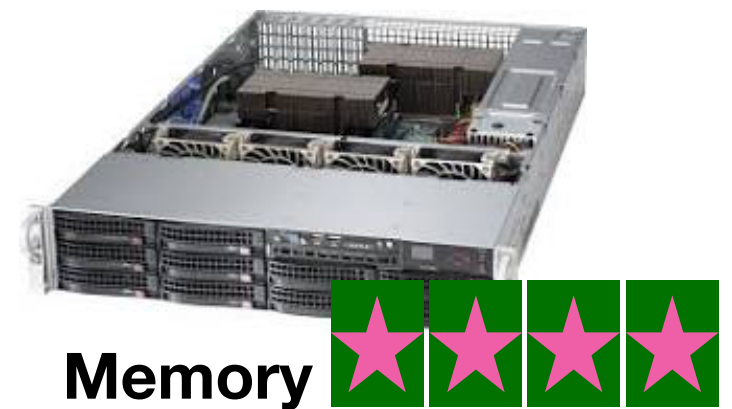
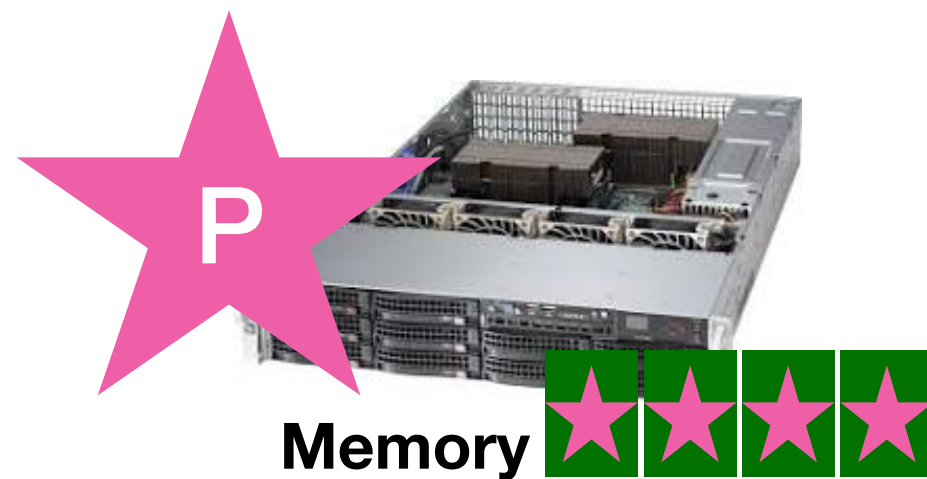
Accessing remote memory incurs much higher overhead than local memory



# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

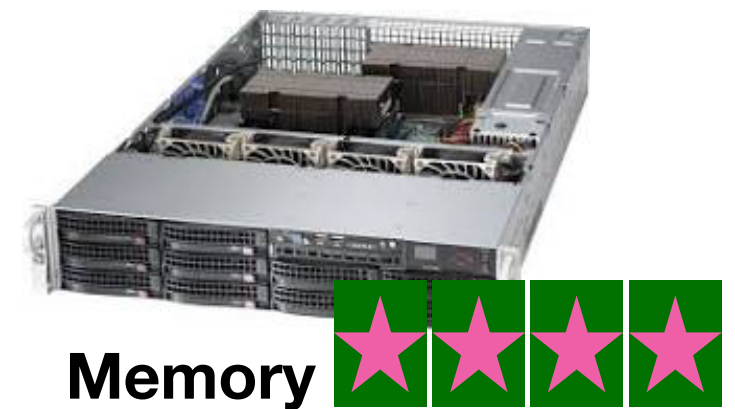
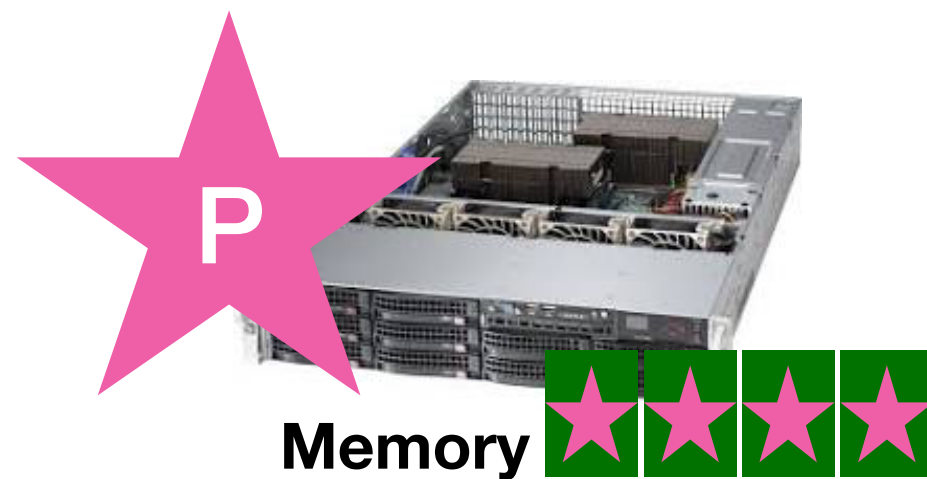
## Fluid multi-resource disaggregation



# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

## Fluid multi-resource disaggregation

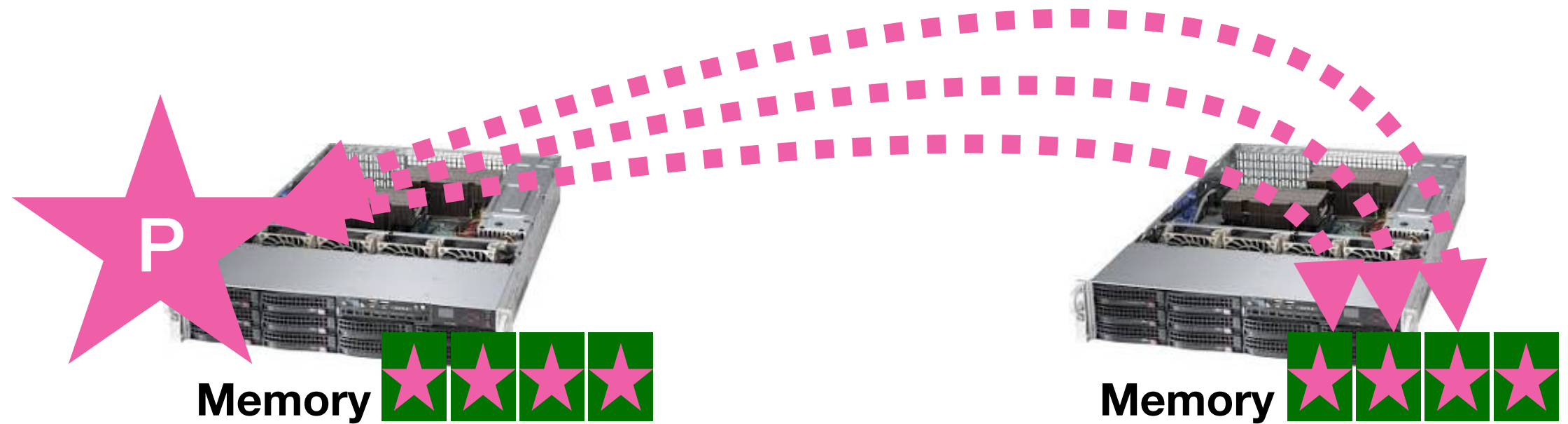


Fluidity: allow process to *move to* data when more efficient

# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

## Fluid multi-resource disaggregation

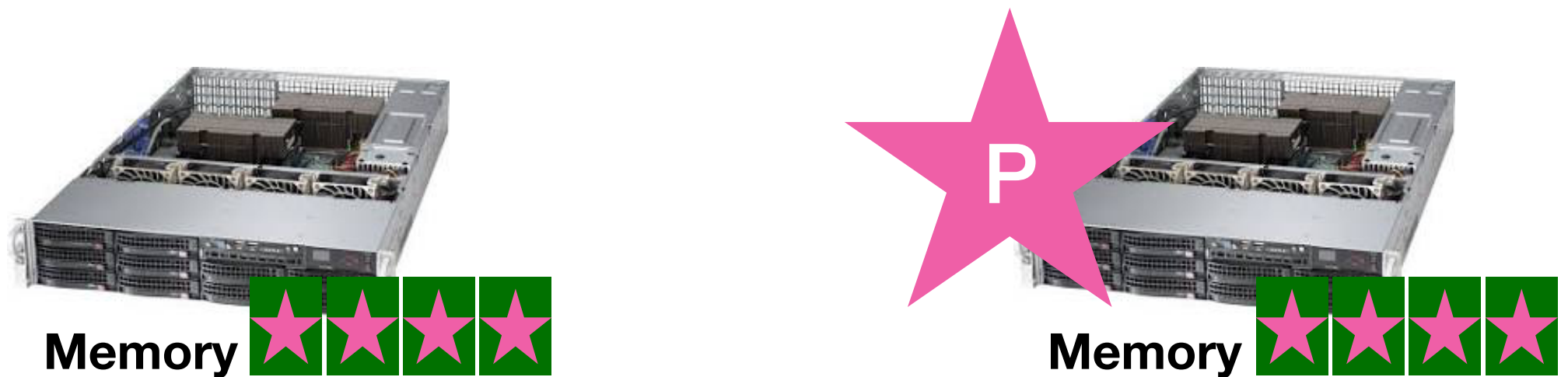


Fluidity: allow process to *move to* data when more efficient

# Fluid multi-resource disaggregation

- Break coupling between process & underlying physical server resources

## Fluid multi-resource disaggregation

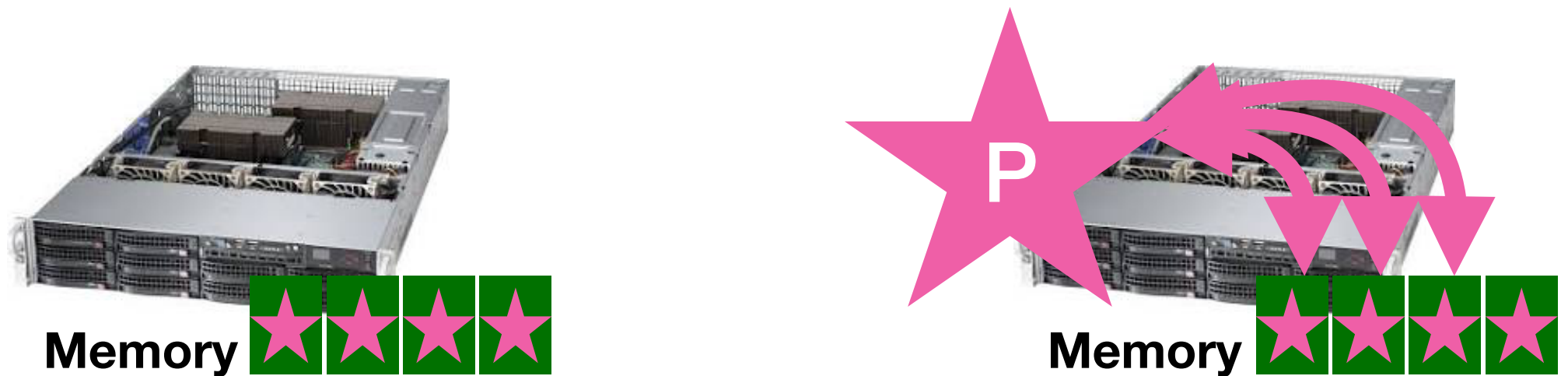


Fluidity: allow process to *move to* data when more efficient

# Fluid multi-resource disaggregation

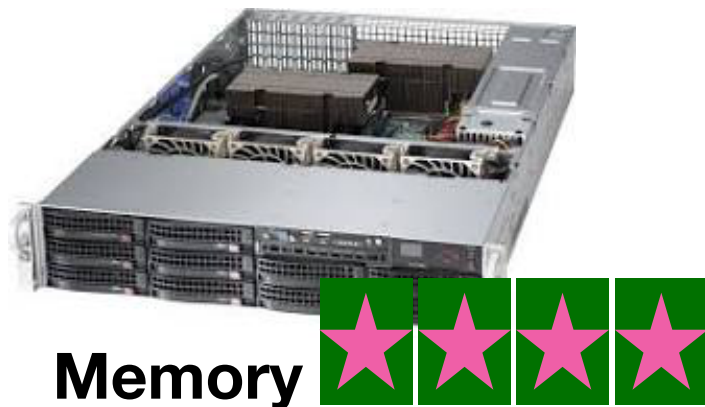
- Break coupling between process & underlying physical server resources

## Fluid multi-resource disaggregation

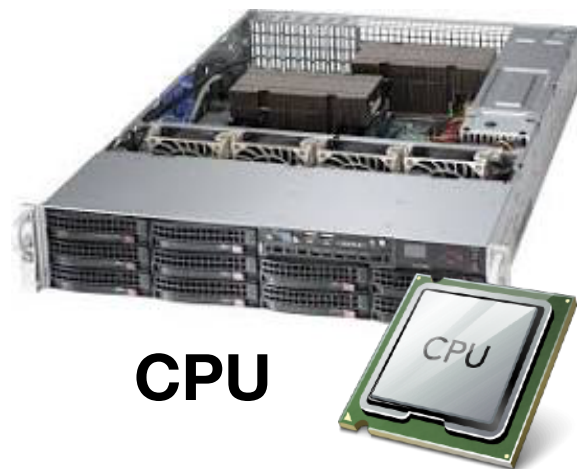


Fluidity: enable process to exploit locality to improve performance

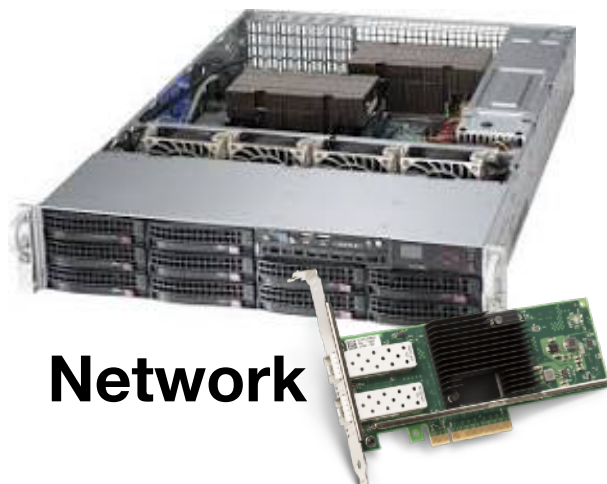
# Fluidity over multiple resources



Already provided by prior works  
(*RamCloud, DSM, InfiniSwap, ...*)



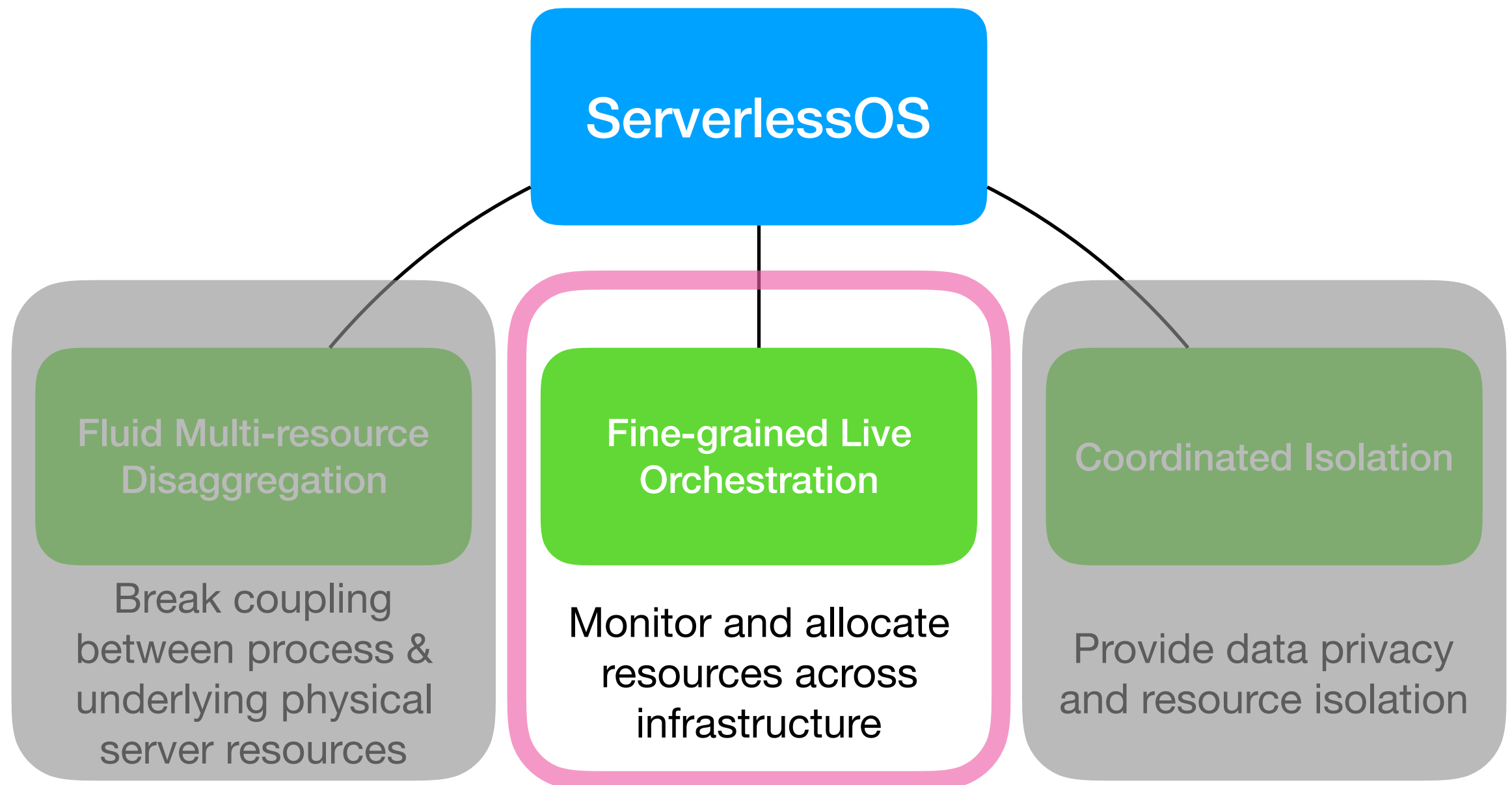
Move processing to data or other  
server with more compute resources  
(*Initial results show 2-3x speedup over a DSM scheme*)



Decouple device that captured I/O from device that will  
process I/O. Additionally, move I/O to more bandwidth.  
(*CPU fluidity can move processing with socket*)

# Outline of talk

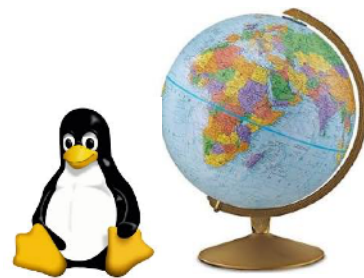
- Goal: provide seamless, scale-out process abstraction
- This talk: high-level outline of our *ServerlessOS* vision





# Fine-grained live orchestration layer

- Monitor, allocate, and optimize run-time performance by automatically assigning, migrating, or scaling workloads



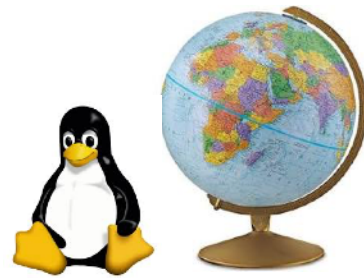
Global decision making



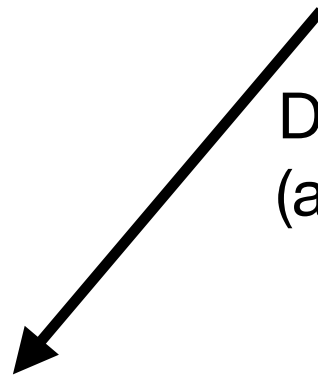


# Fine-grained live orchestration layer

- Monitor, allocate, and optimize run-time performance by automatically assigning, migrating, or scaling workloads



Global decision making

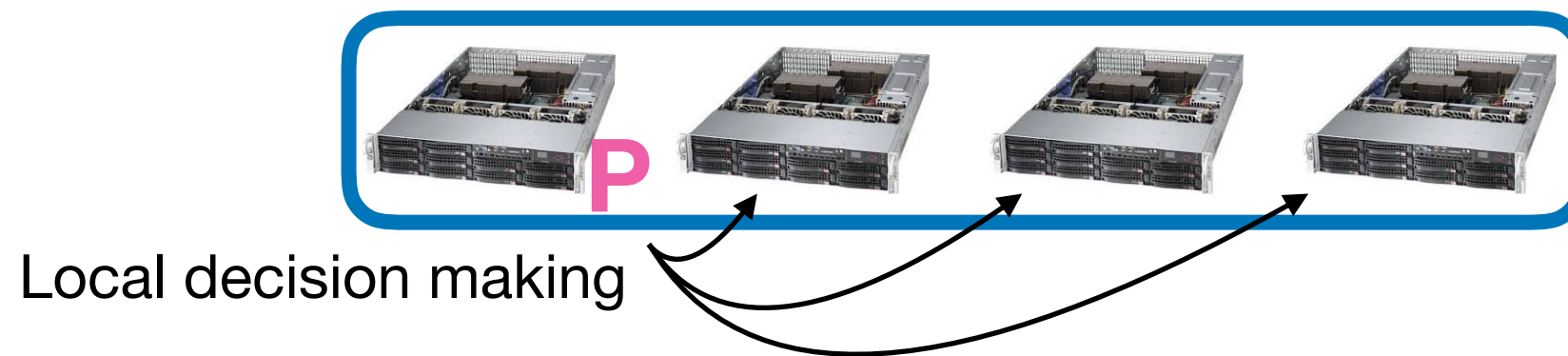


Determine subset of nodes  
(and resources) available to  
process **P**



# Fine-grained live orchestration layer

- Monitor, allocate, and optimize run-time performance by automatically assigning, migrating, or scaling workloads



- When to expand (or contract)
- Where to expand (or contract)

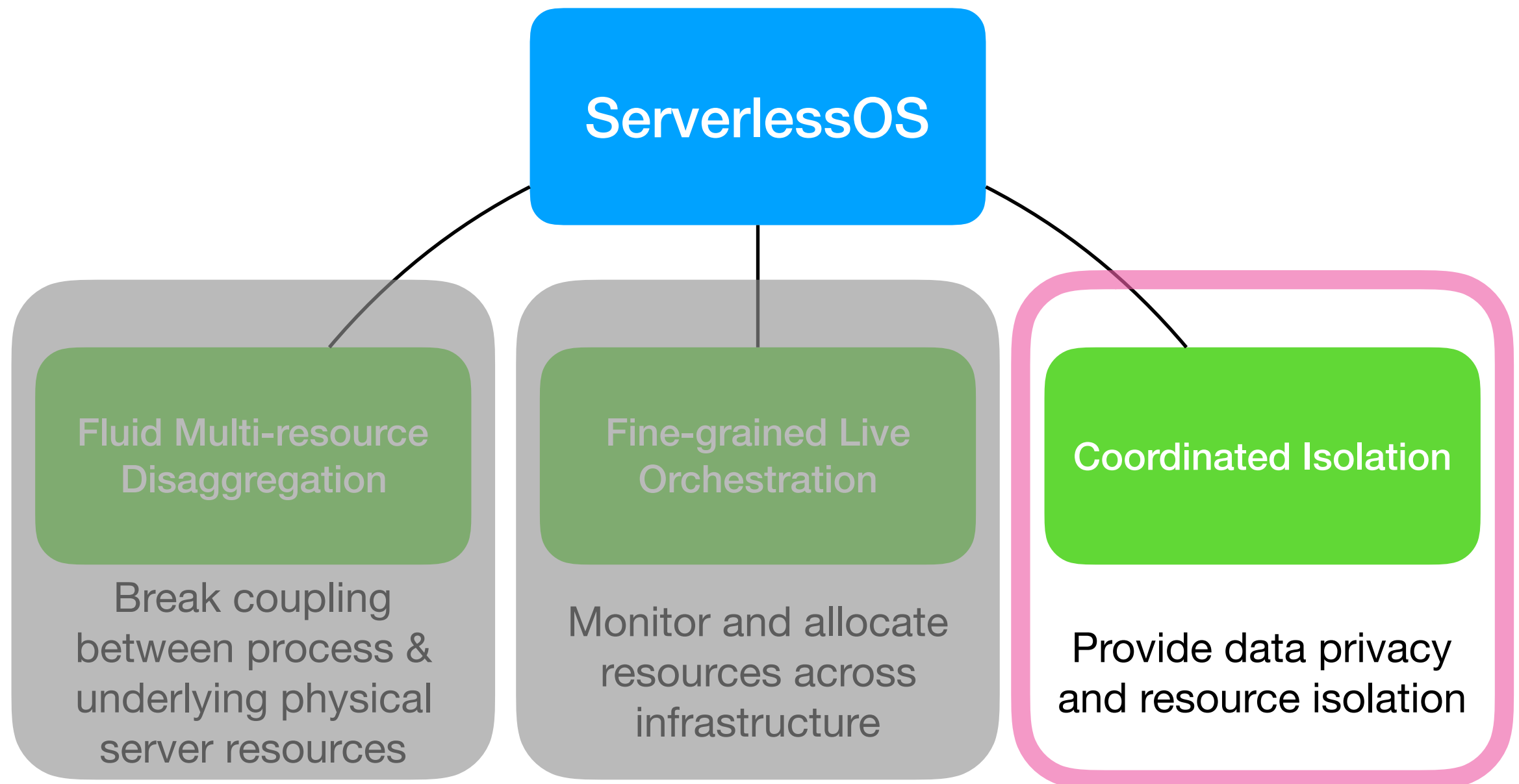
Both decisions influenced by state of other local nodes:

- CPU
- Memory
- Network

Backpressure algorithm avoids poor decisions

# Outline of talk

- Goal: provide seamless, scale-out process abstraction
- This talk: high-level outline of our *ServerlessOS* vision



# Coordinated Isolation

## Data Privacy

Ensure application cannot read or write state from another application

## Resource Isolation

Bound CPU, memory, storage, and network usage of workloads

# Coordinated Isolation

Data Privacy

Ensure application cannot read or write state from another application

Resource Isolation

Bound CPU, memory, storage, and network usage of workloads

Linux Kernel

# Coordinated Isolation

Data Privacy

Ensure application cannot read or write state from another application

namespaces

Resource Isolation

Bound CPU, memory, storage, and network usage of workloads

control groups

Linux Kernel

# Coordinated Isolation

Data Privacy

Ensure application cannot read or write state from another application

namespaces

Resource Isolation

Bound CPU, memory, storage, and network usage of workloads

control groups

Linux Kernel

ServerlessOS: extend isolation across multiple servers in coordinated fashion

# Coordinated Isolation

namespaces

Extend process namespace across multiple servers

control groups

Centralize state in orchestration layer, but minimize overheads



Global store of CPU shares



Core 0      Core 1      Core 2

From single server...



# Coordinated Isolation

namespaces

Extend process namespace across multiple servers

control groups

Centralize state in orchestration layer, but minimize overheads



Global store of CPU shares



Core 0

Core 1

Core 2

From single server...

# Coordinated Isolation

namespaces

Extend process namespace across multiple servers

control groups

Centralize state in orchestration layer, but minimize overheads



Global store of CPU shares



Core 0

Core 1

Core 2

From single server...

# Coordinated Isolation

namespaces

Extend process namespace across multiple servers

control groups

Centralize state in orchestration layer, but minimize overheads



Global store of CPU shares



Core 0

Core 1

Core 2

From single server...

# Coordinated Isolation

namespaces

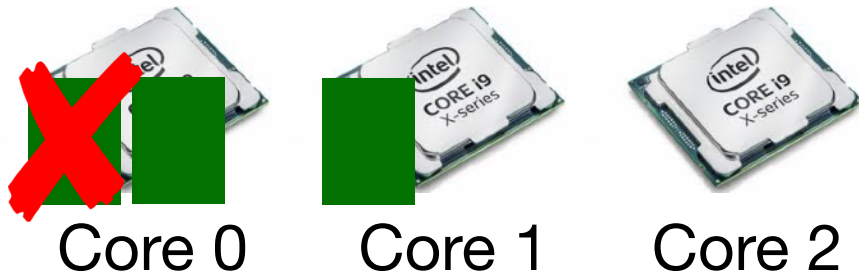
Extend process namespace across multiple servers

control groups

Centralize state in orchestration layer, but minimize overheads



Global store of CPU shares



Core 0

Core 1

Core 2

From single server...

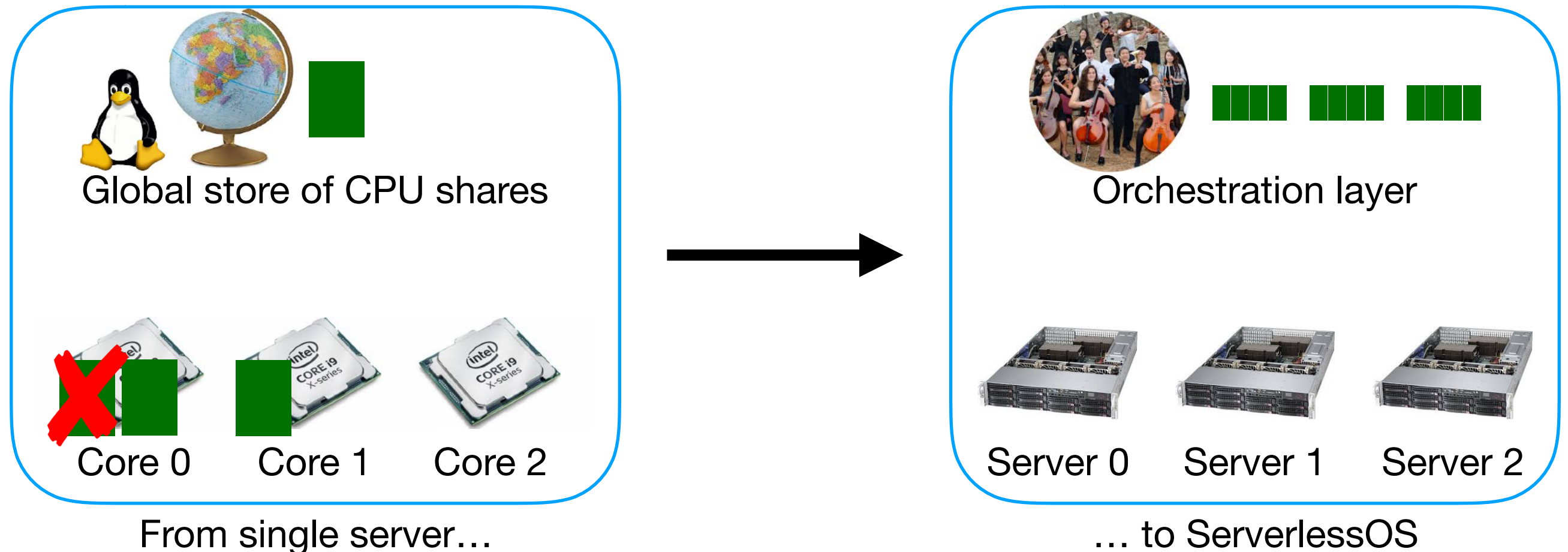
# Coordinated Isolation

namespaces

Extend process namespace across multiple servers

control groups

Centralize state in orchestration layer, but minimize overheads



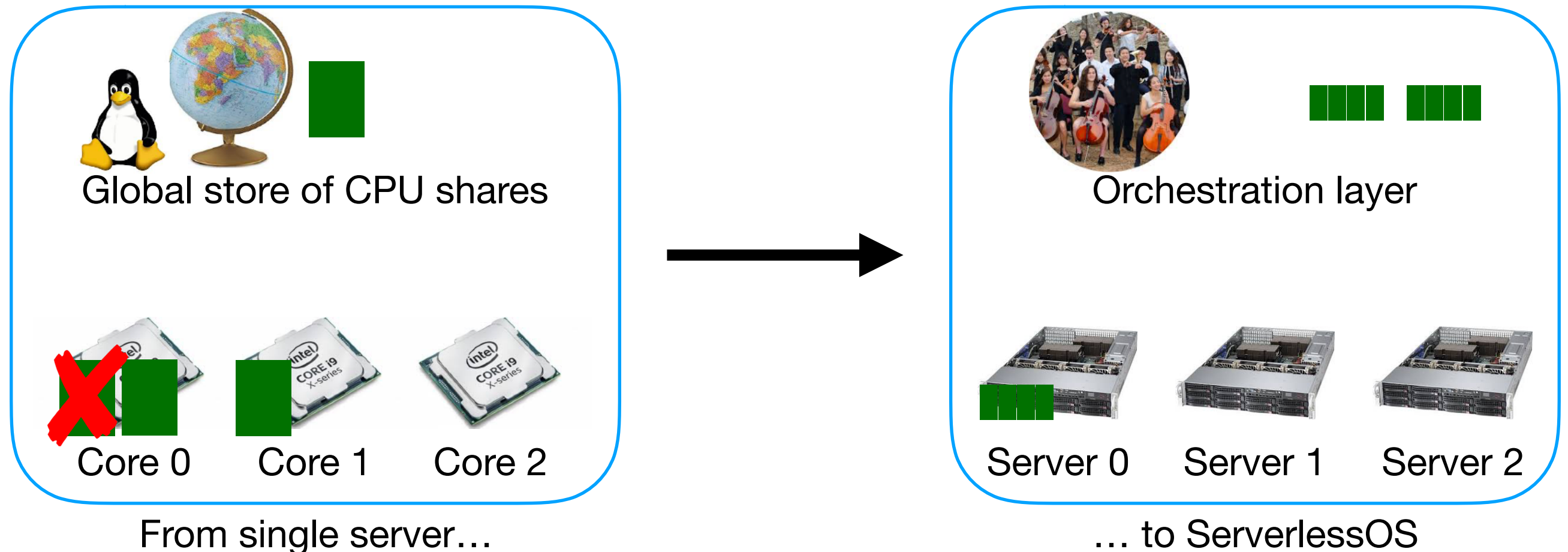
# Coordinated Isolation

namespaces

Extend process namespace across multiple servers

control groups

Centralize state in orchestration layer, but minimize overheads



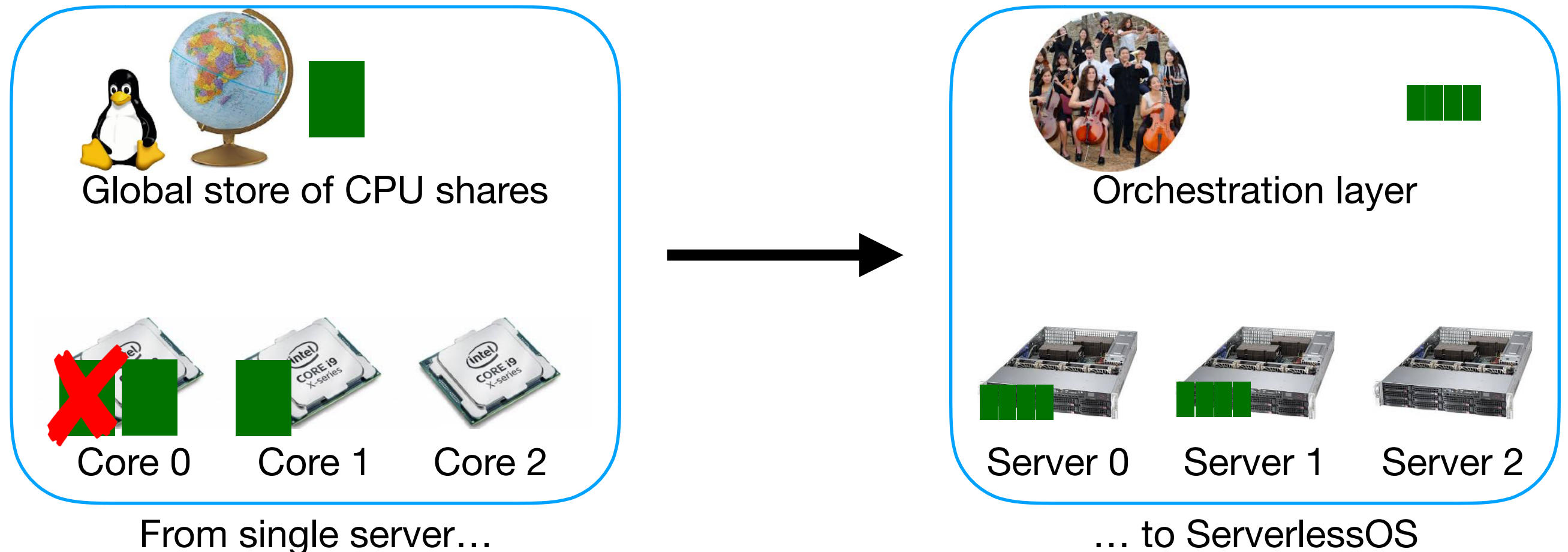
# Coordinated Isolation

namespaces

Extend process namespace across multiple servers

control groups

Centralize state in orchestration layer, but minimize overheads





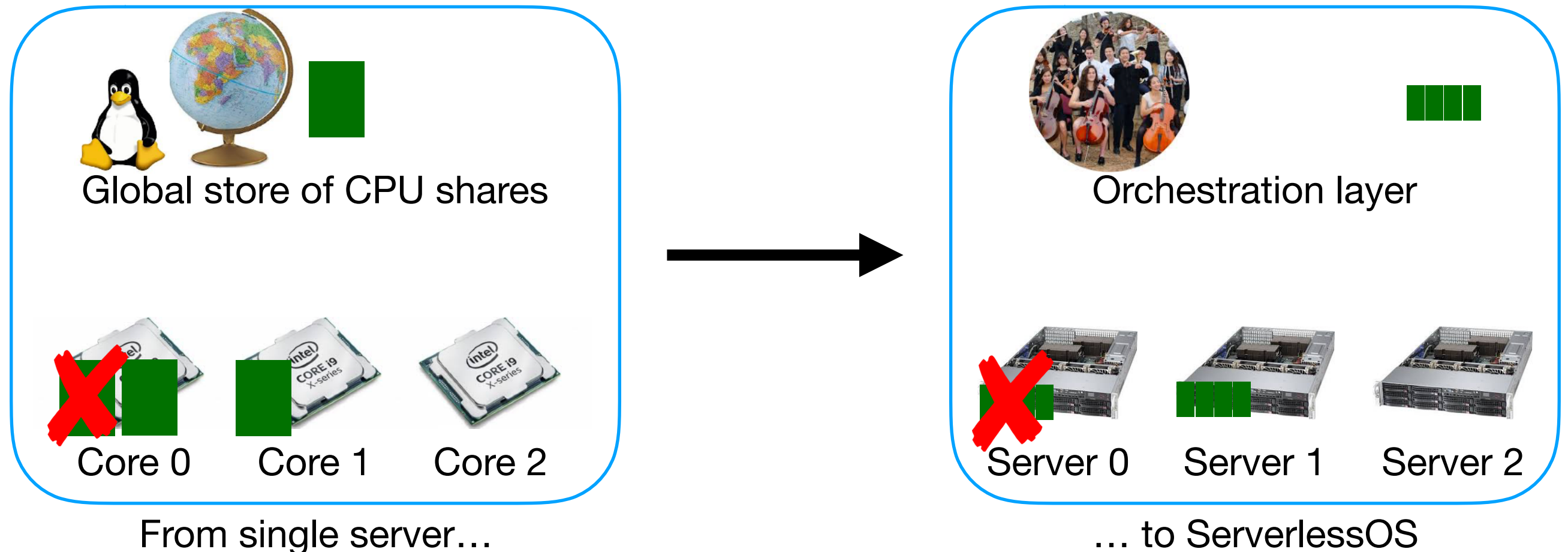
# Coordinated Isolation

namespaces

Extend process namespace across multiple servers

control groups

Centralize state in orchestration layer, but minimize overheads





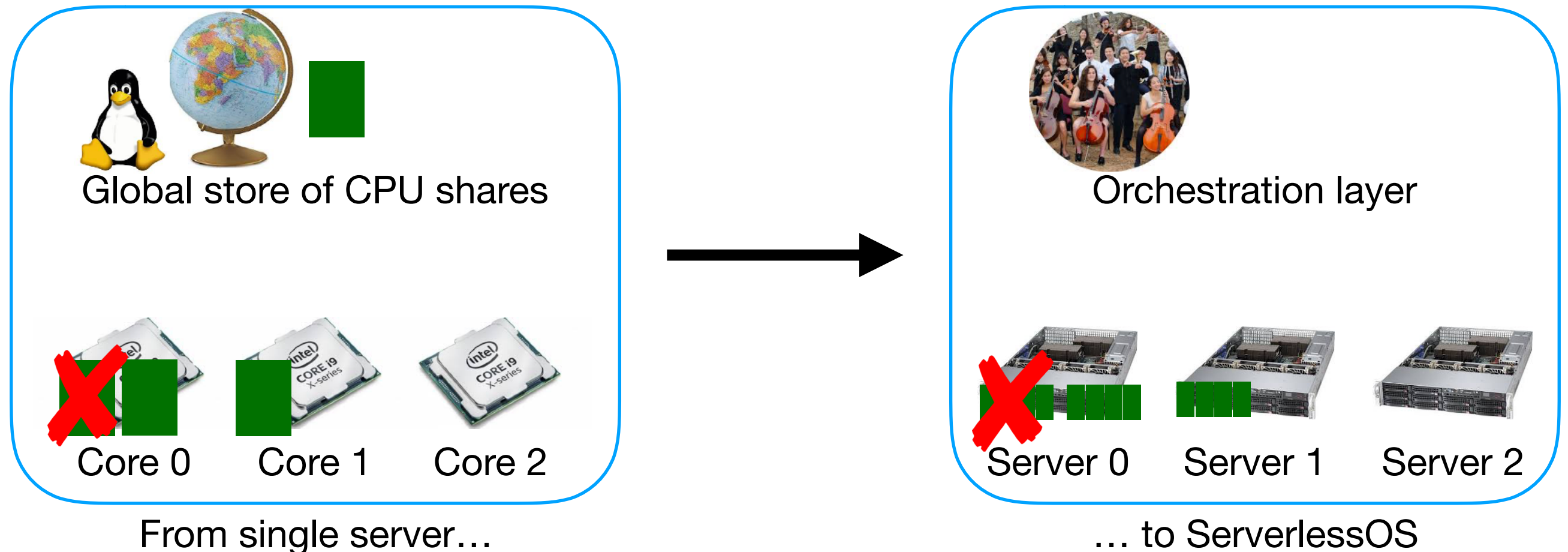
# Coordinated Isolation

namespaces

Extend process namespace across multiple servers

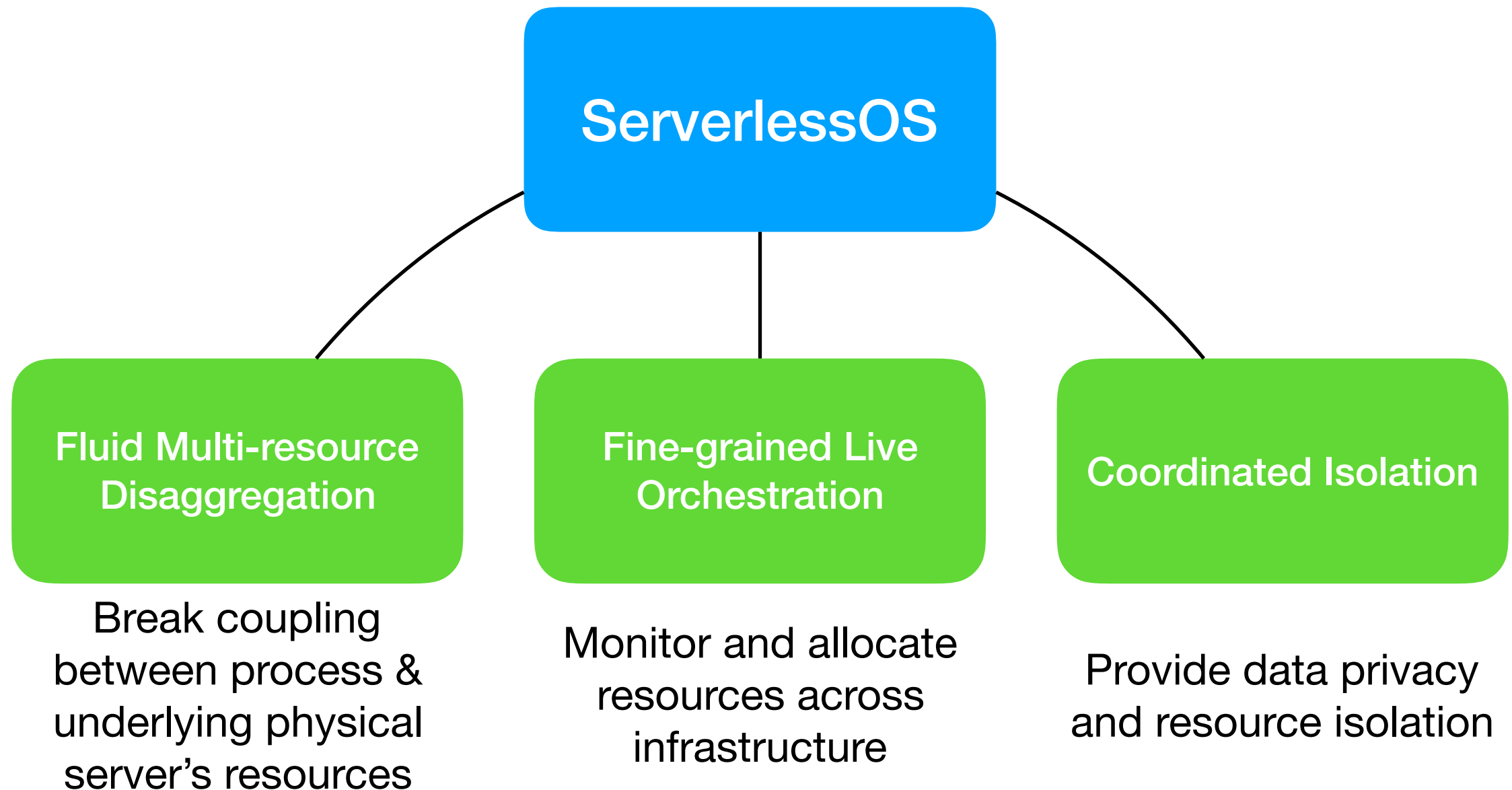
control groups

Centralize state in orchestration layer, but minimize overheads



# Outline of talk

- Goal: provide seamless, scale-out process abstraction
- This talk: high-level outline of our *ServerlessOS* vision



# Conclusions

- New abstraction for serverless: a seamless, scale-out process
- High-level overview of ServerlessOS architecture
  - Fluid multi-resource disaggregation
  - Fine-grained live orchestration layer
  - Coordinated isolation
- Complementary to current serverless techniques
- Next steps: refine design, build prototype, conquer the world!
- Thanks! [mailto: erozner@us.ibm.com](mailto:erozner@us.ibm.com)