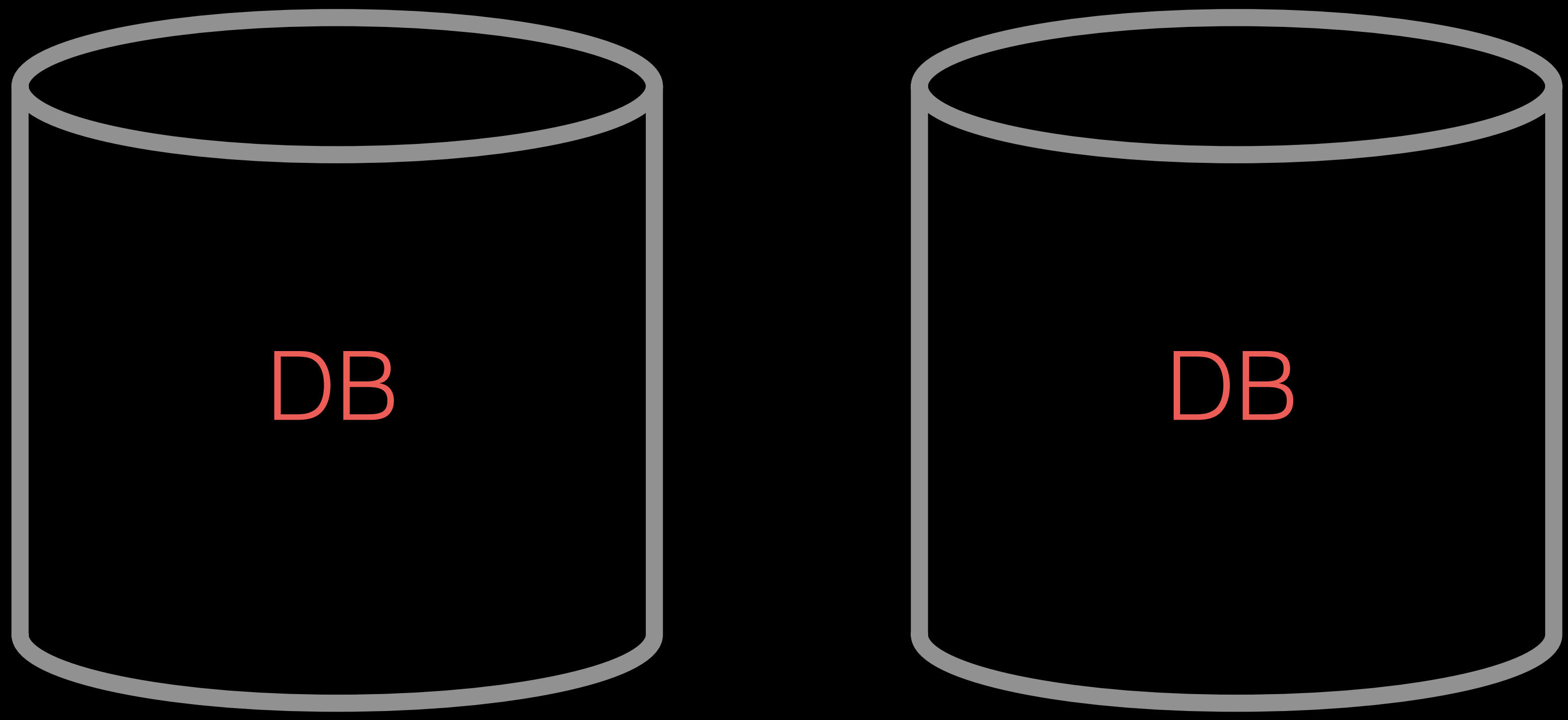
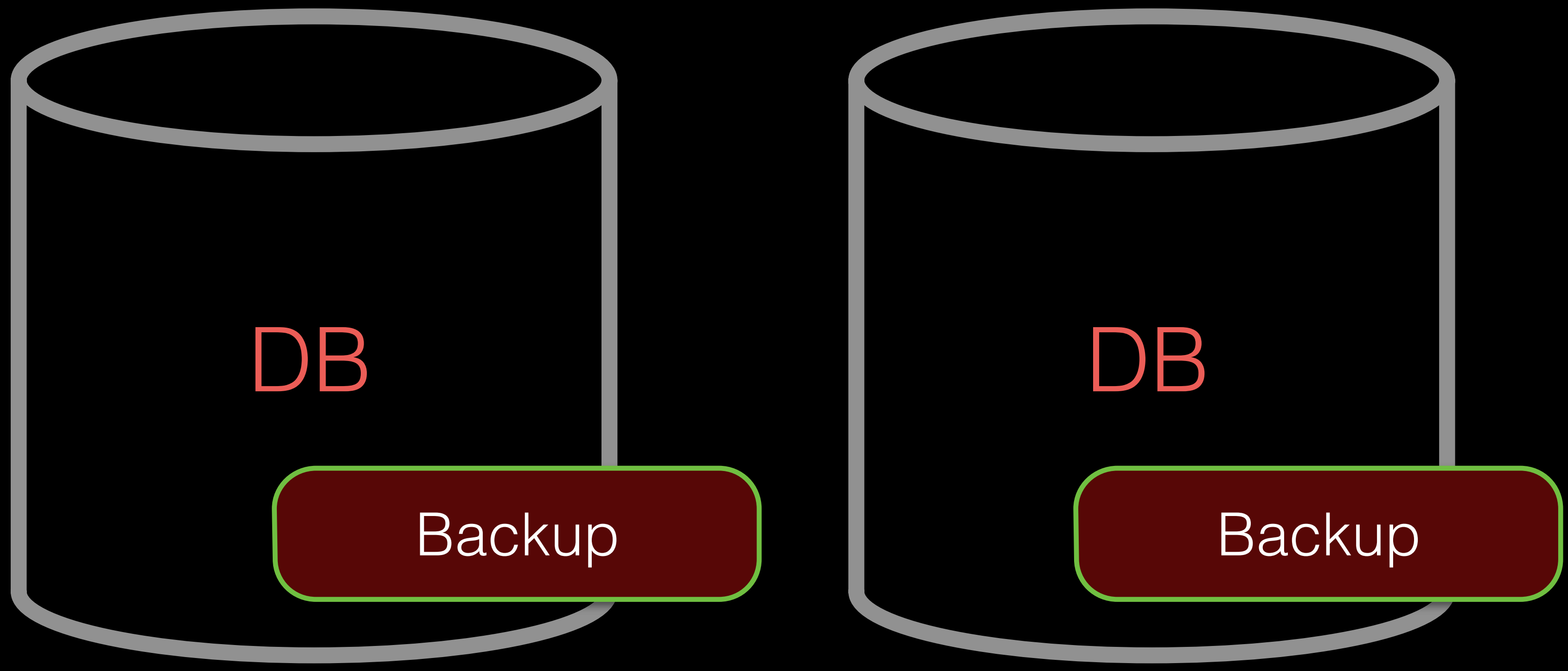
Managing Containers with Helix

Kanak Biscuitwala
Jason Zhang
Apache Helix Committers @ LinkedIn
helix.apache.org
@apachehelix

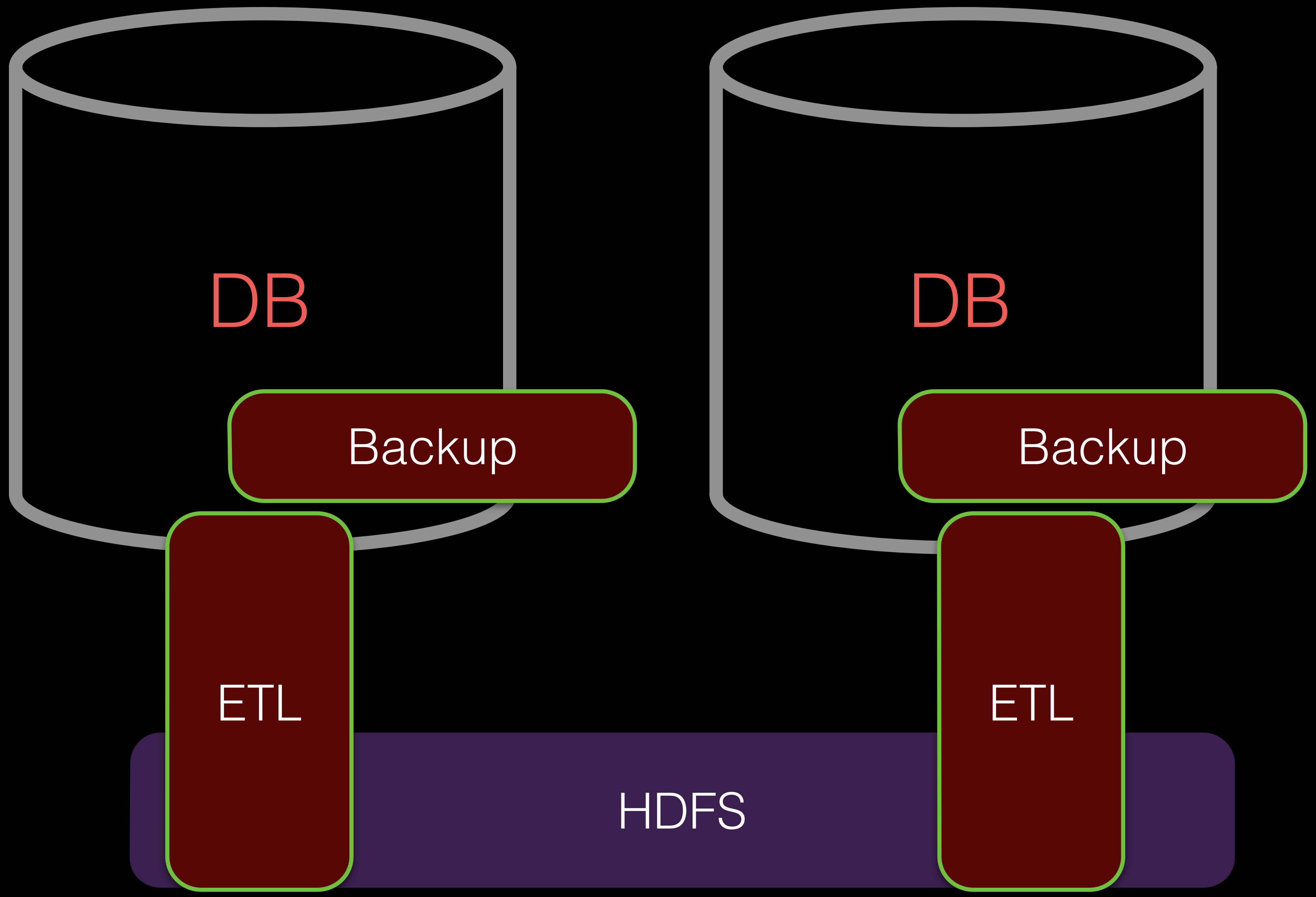
Intersection of Job Types



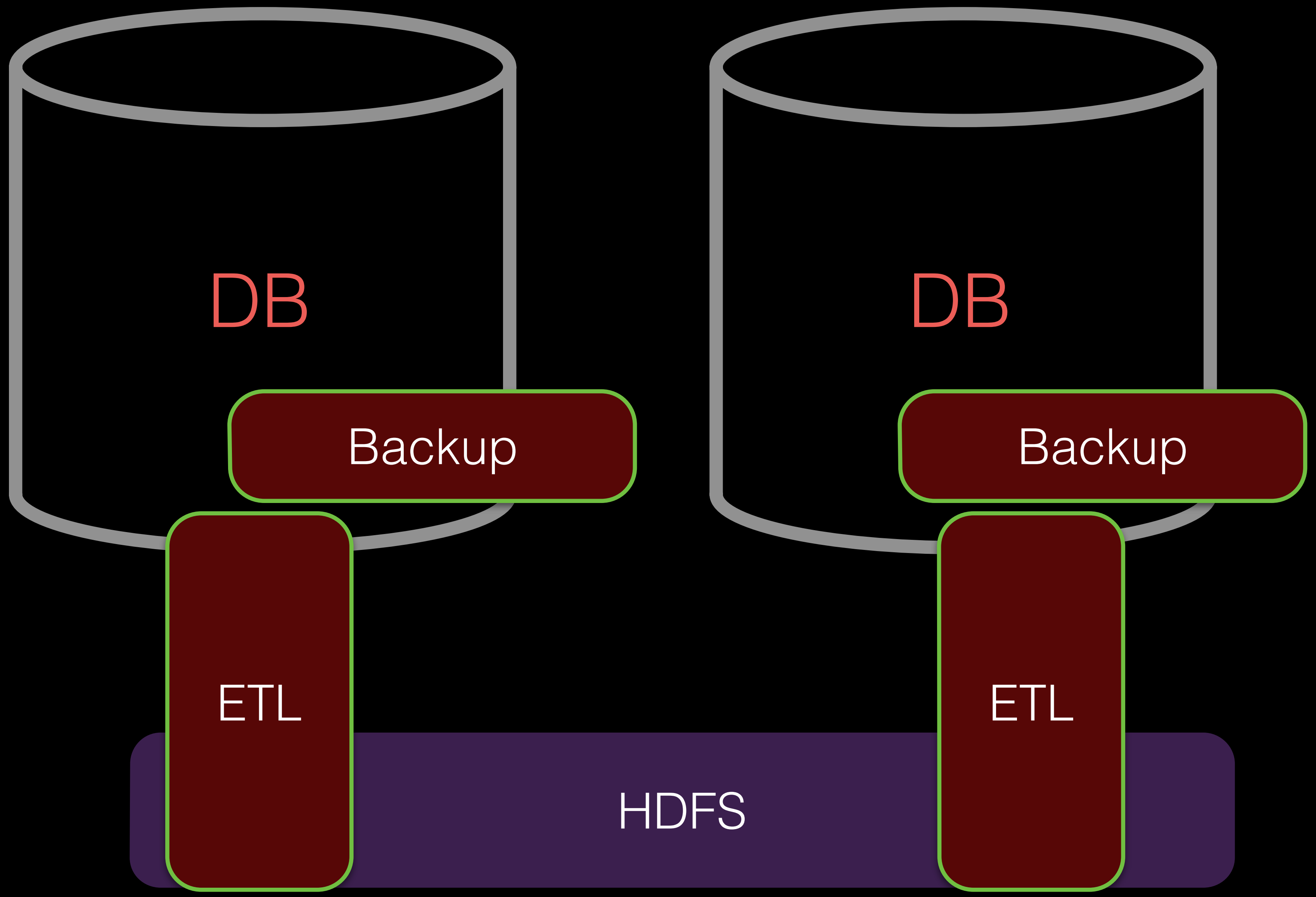
Intersection of Job Types



Intersection of Job Types

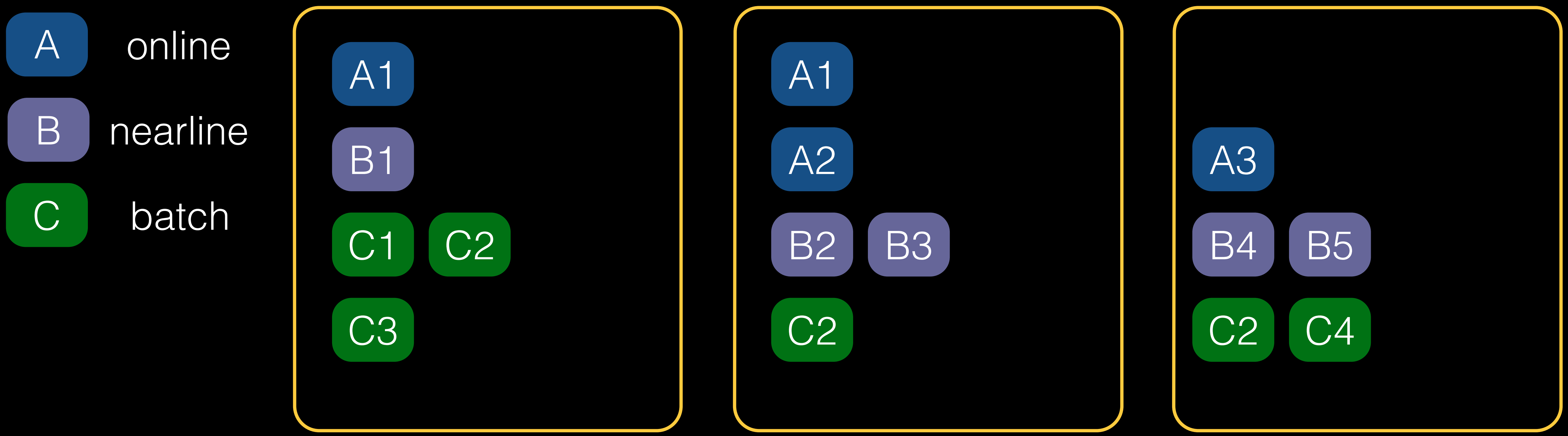


Intersection of Job Types



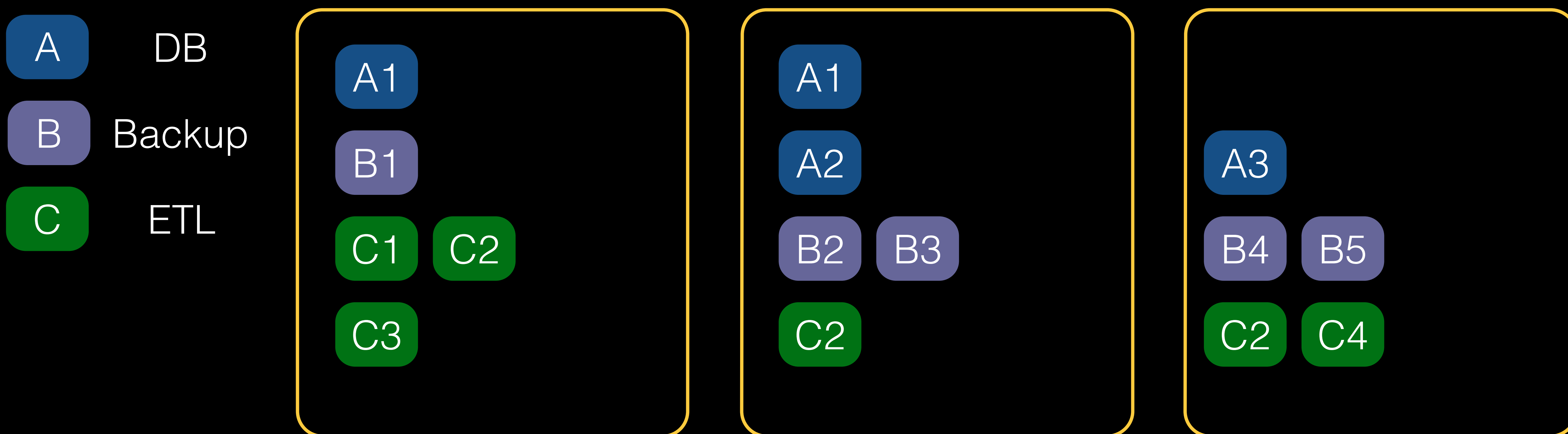
Long-running and batch jobs running together!

Cloud Deployment



Applications with diverse requirements running together in a datacenter

Cloud Deployment



Applications with diverse requirements running together in a datacenter

Processes on Machines

Machine

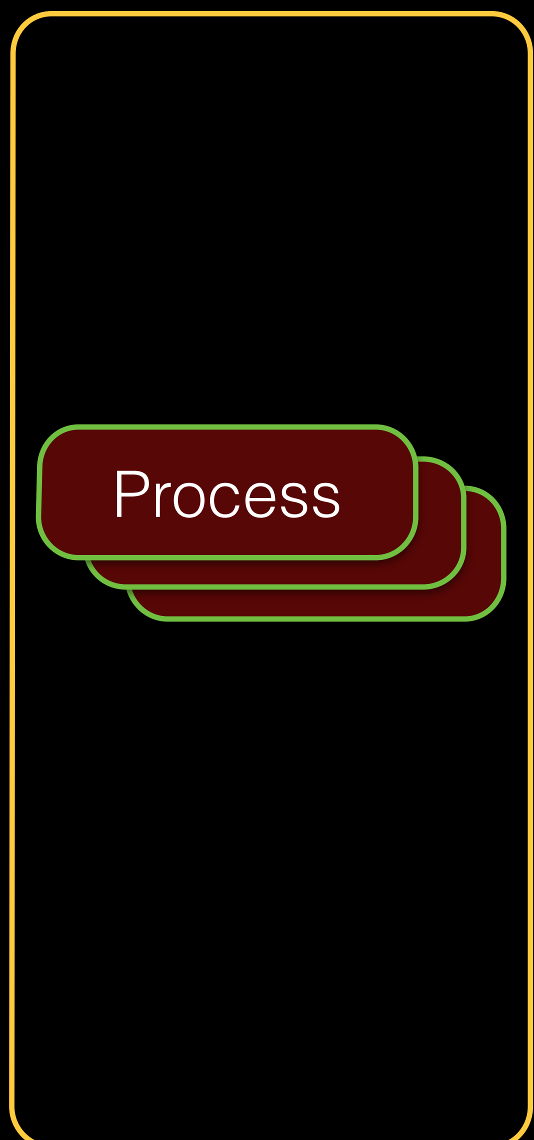
Process

VM

Container

HELIX

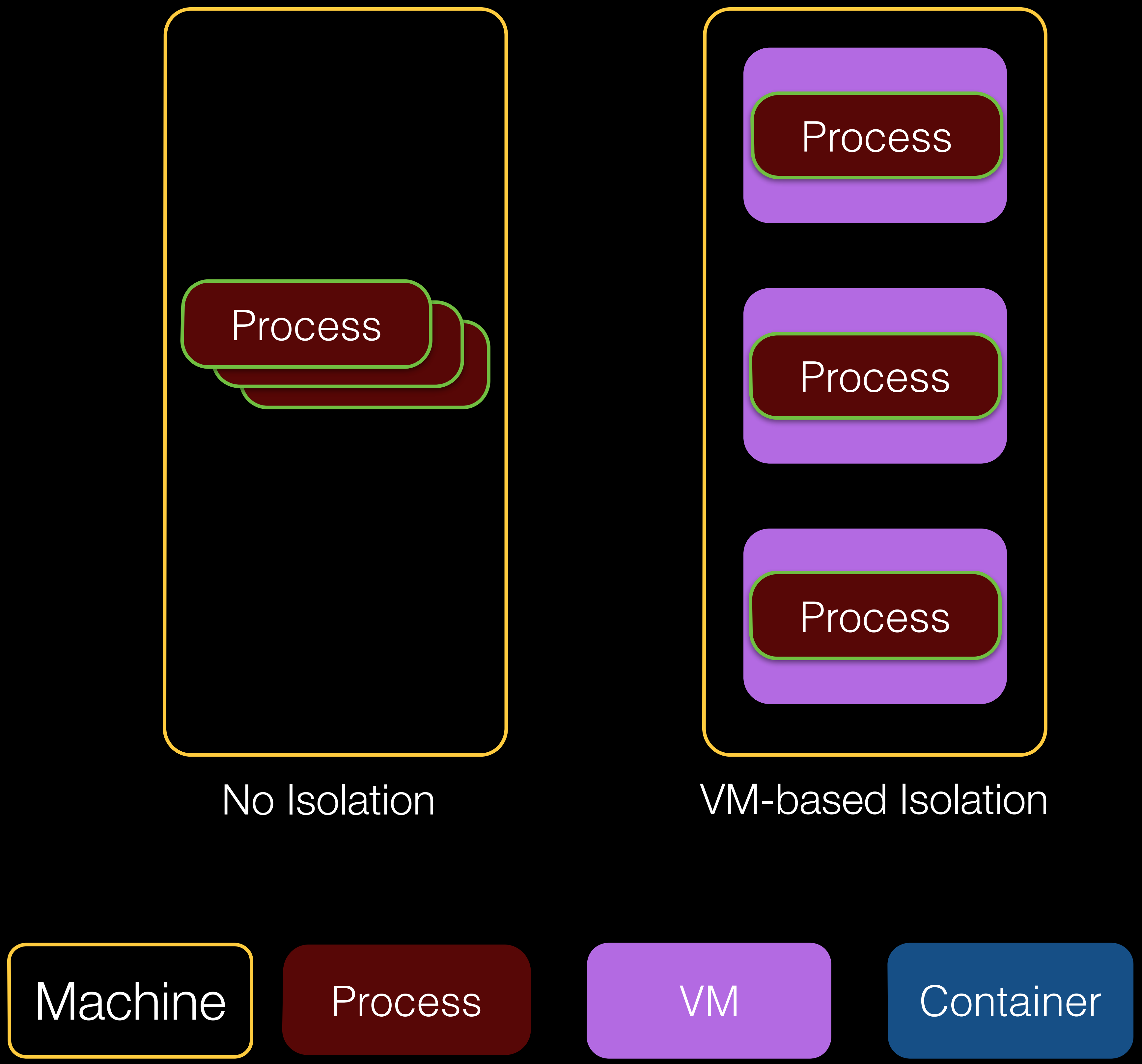
Processes on Machines



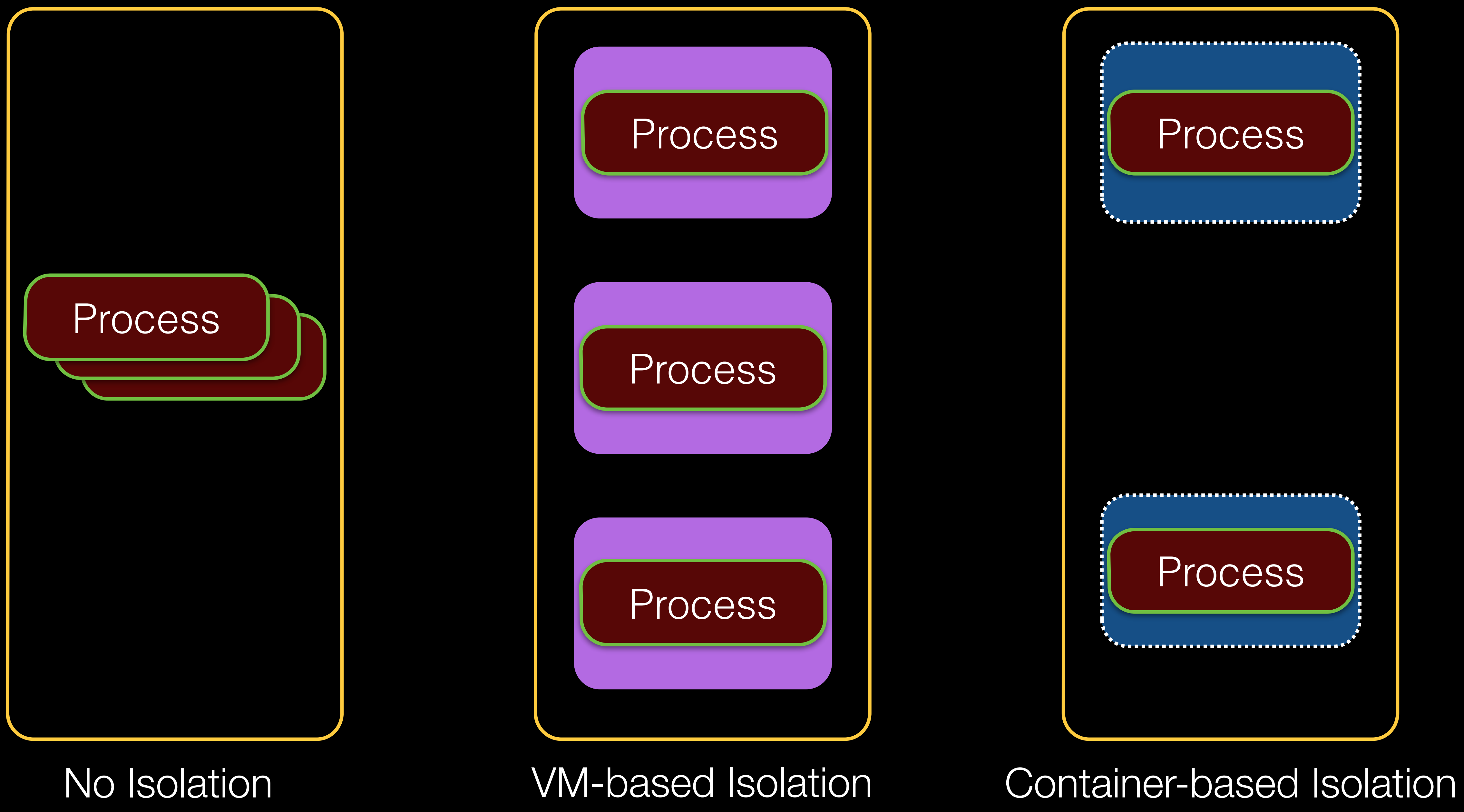
No Isolation

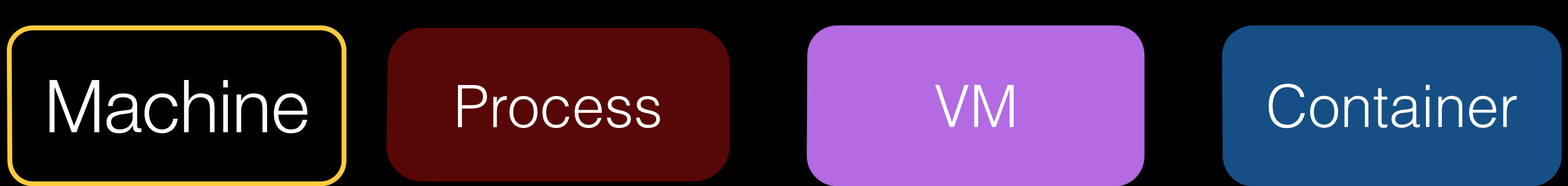
Machine | Process | VM | Container

Processes on Machines



Processes on Machines





Processes on Machines

- Run as individual processes
  - Poor isolation or poor utilization
- Virtual machines
  - Better isolation
  - Xen, Hyper-V, ESX, KVM
- Containers
  - cgroup
  - YARN, Mesos
  - Super lightweight, dynamic based on application requirements

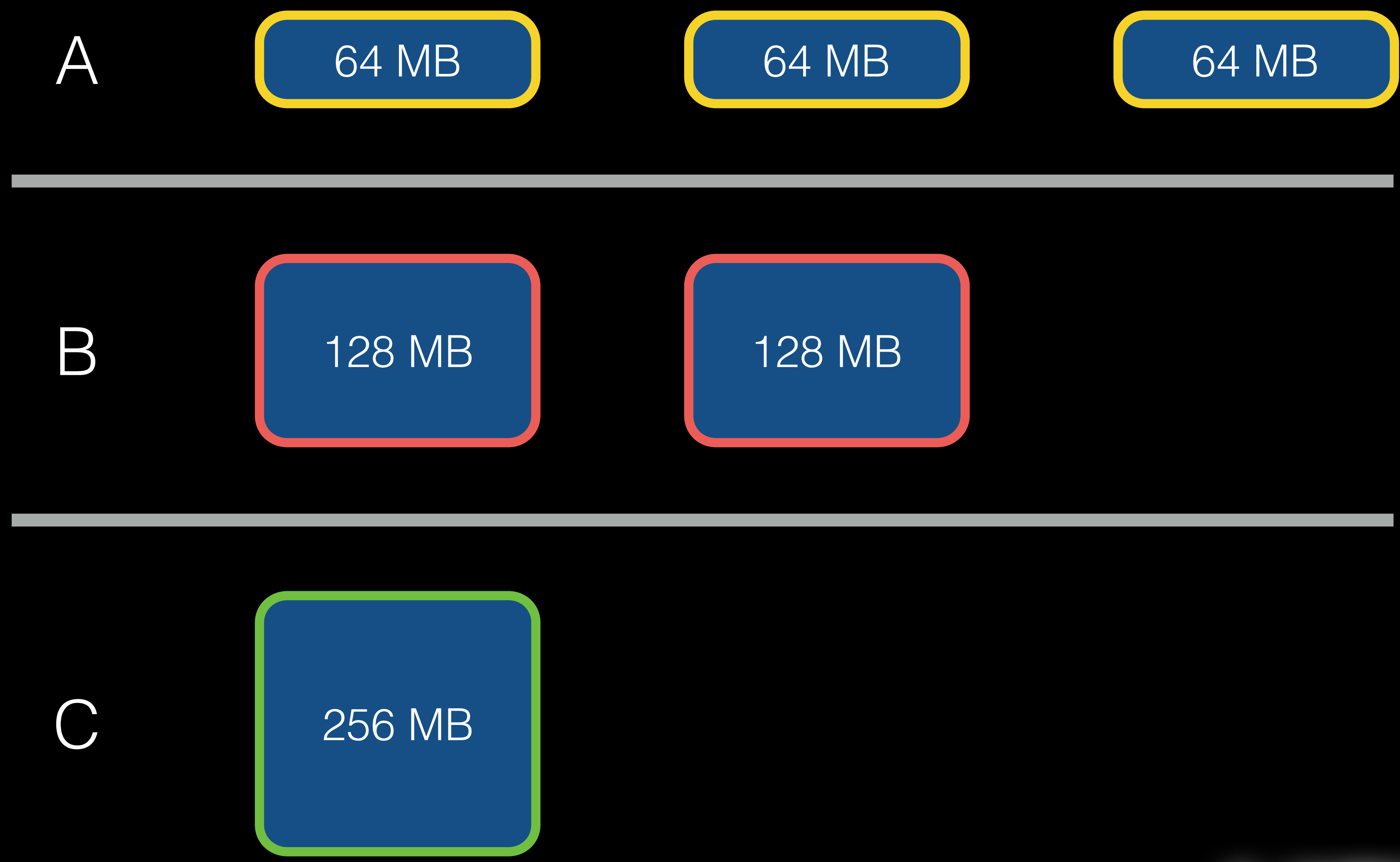
Processes on Machines

Virtualization and containerization significantly improve process isolation and open up possibilities for efficient utilization of physical resources

Container-Based Solution

Container-Based Solution

System Requirements

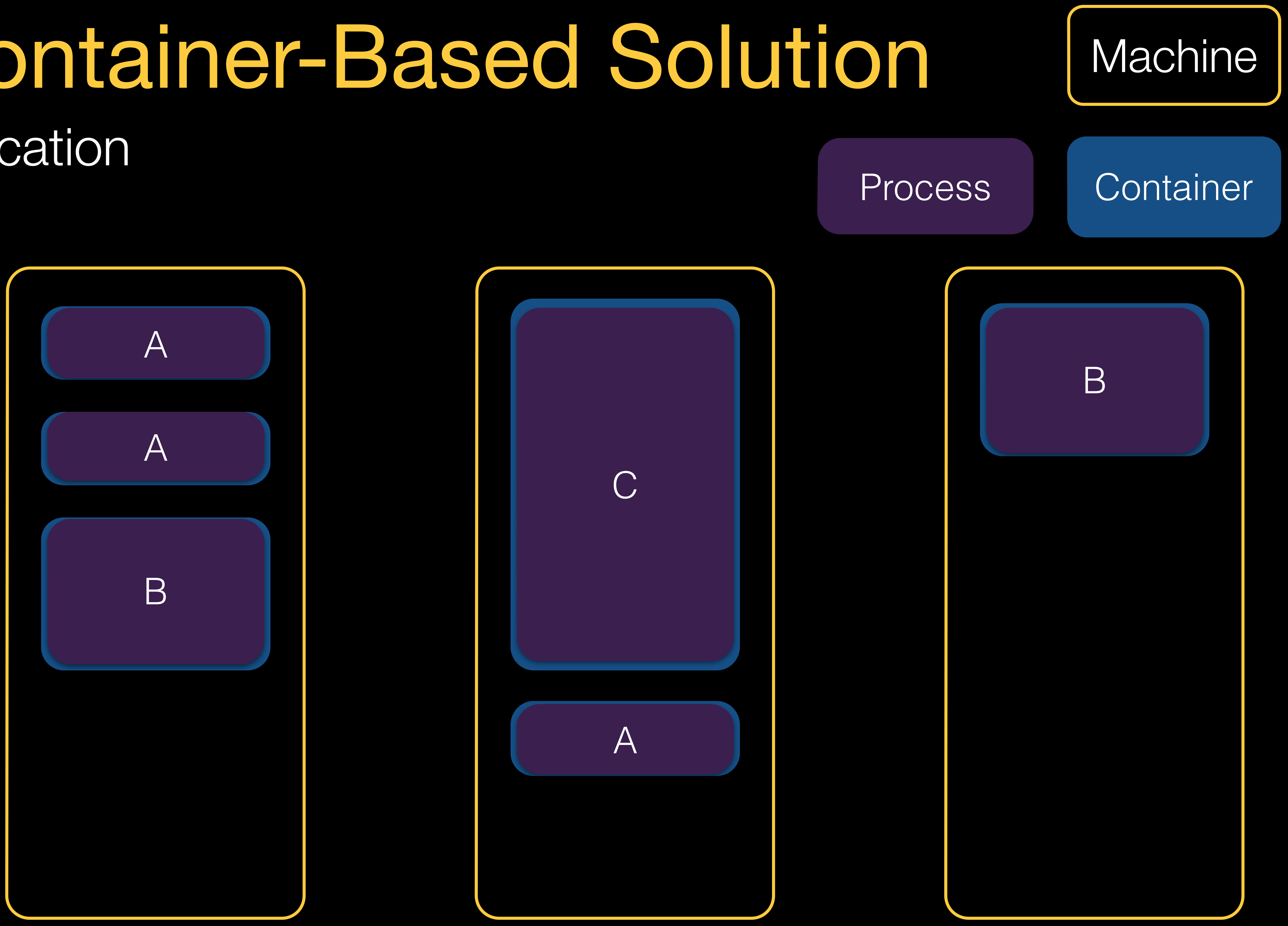


Container-Based Solution

Allocation

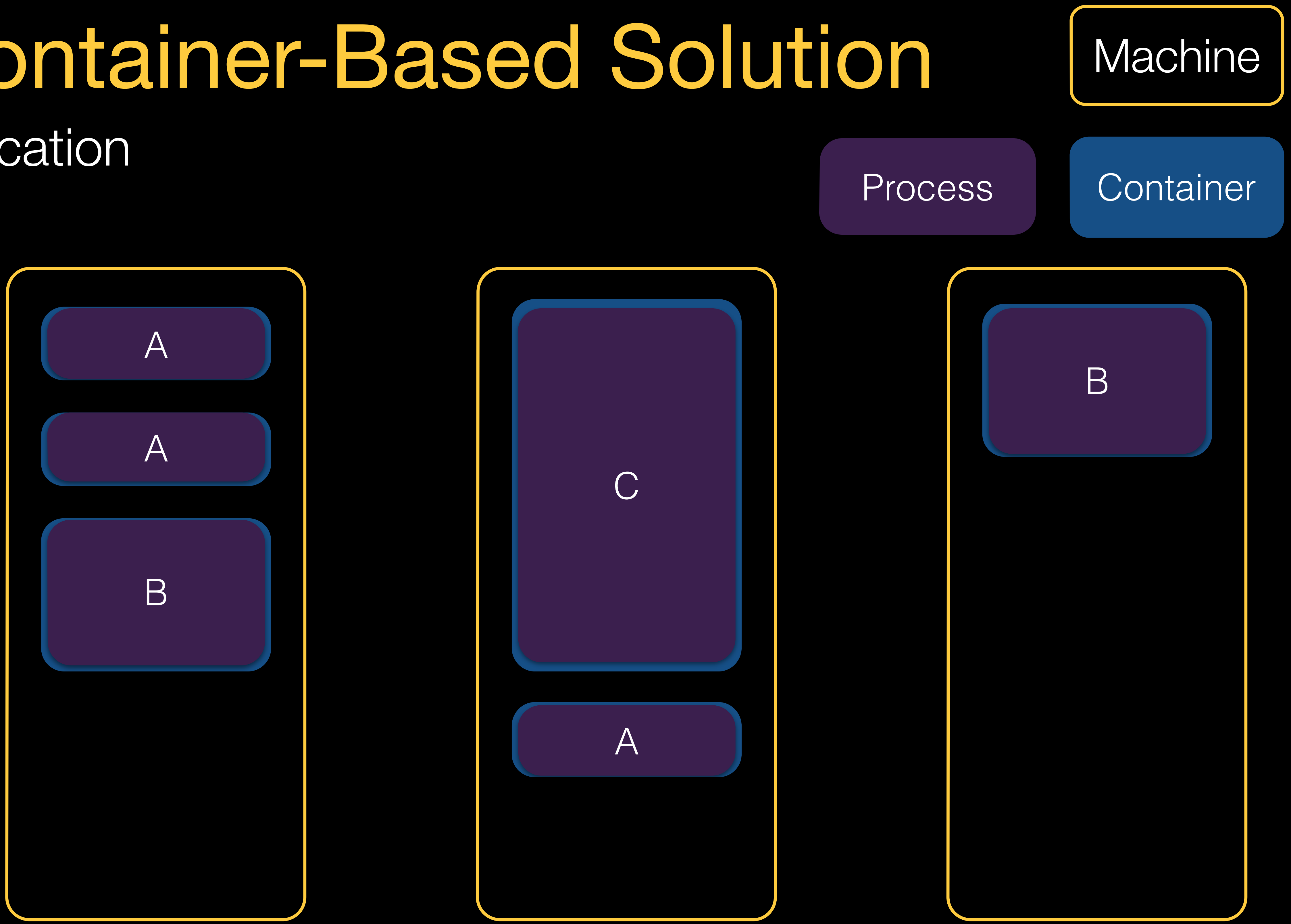
Container-Based Solution

Allocation



Container-Based Solution

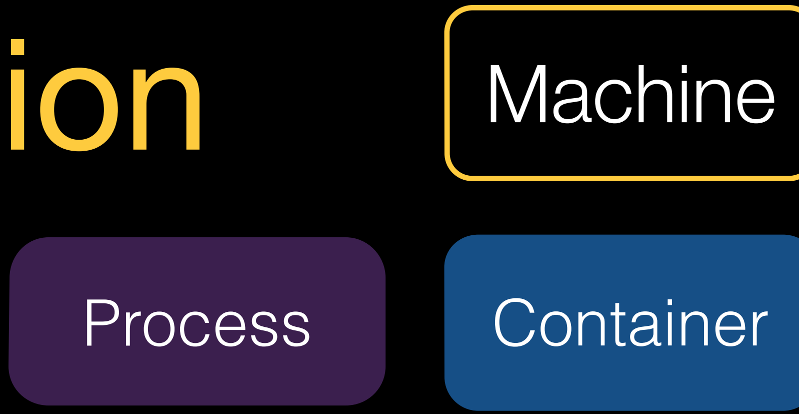
Allocation

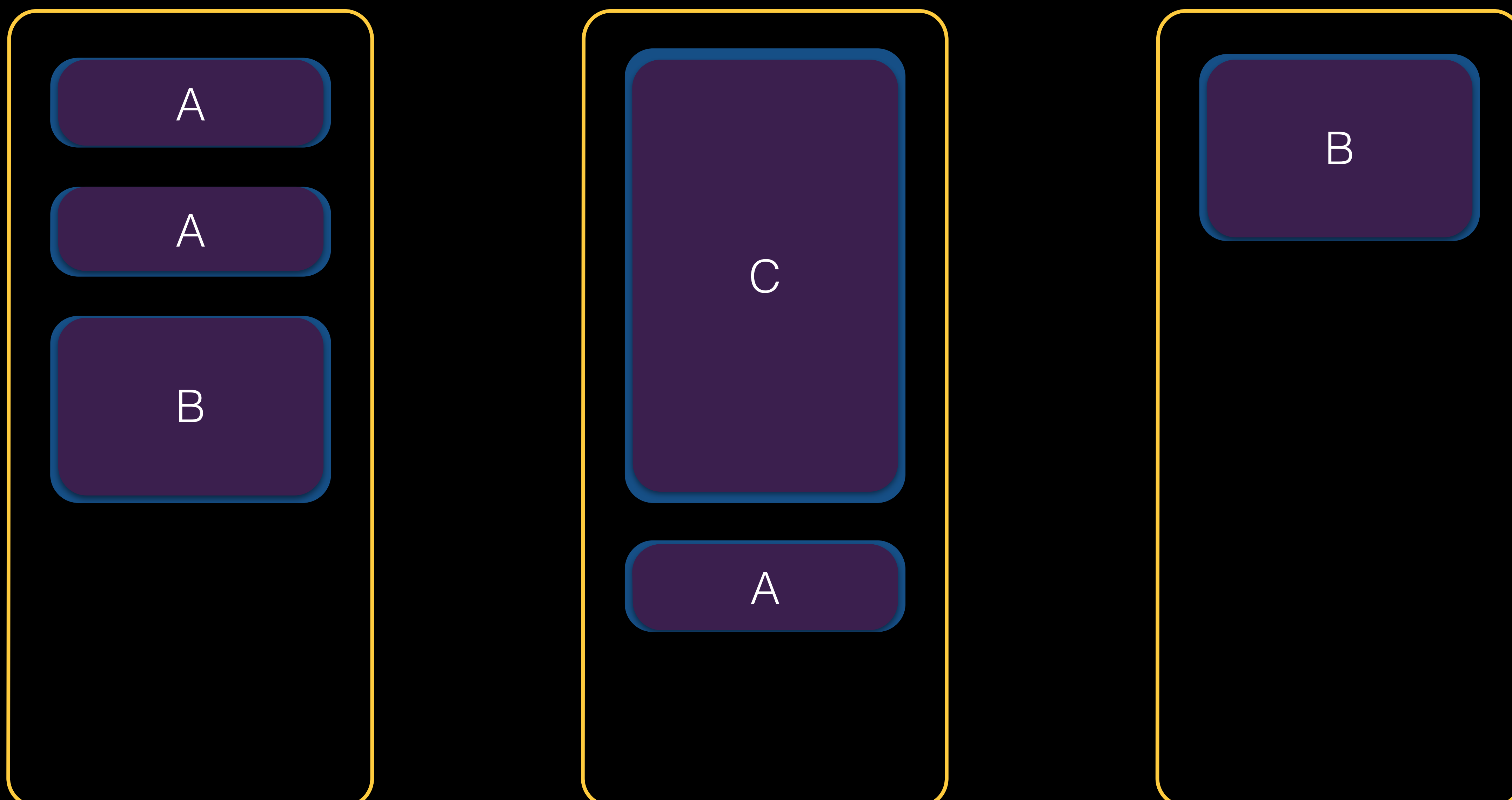


Containerization is powerful!

Container-Based Solution

Allocation





Containerization is powerful!

But do processes always fit so nicely?

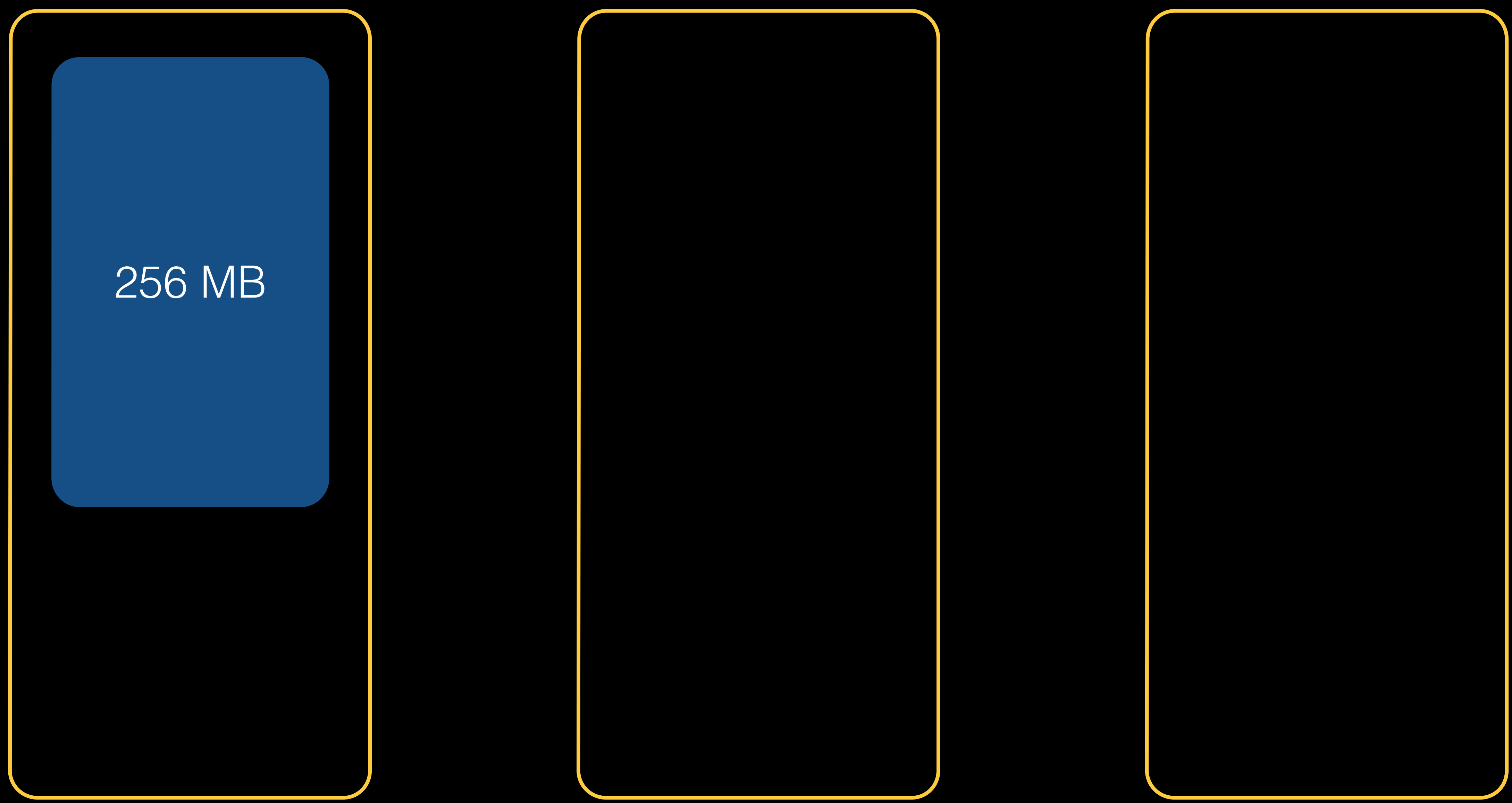
Container-Based Solution

Over-Utilization

Machine

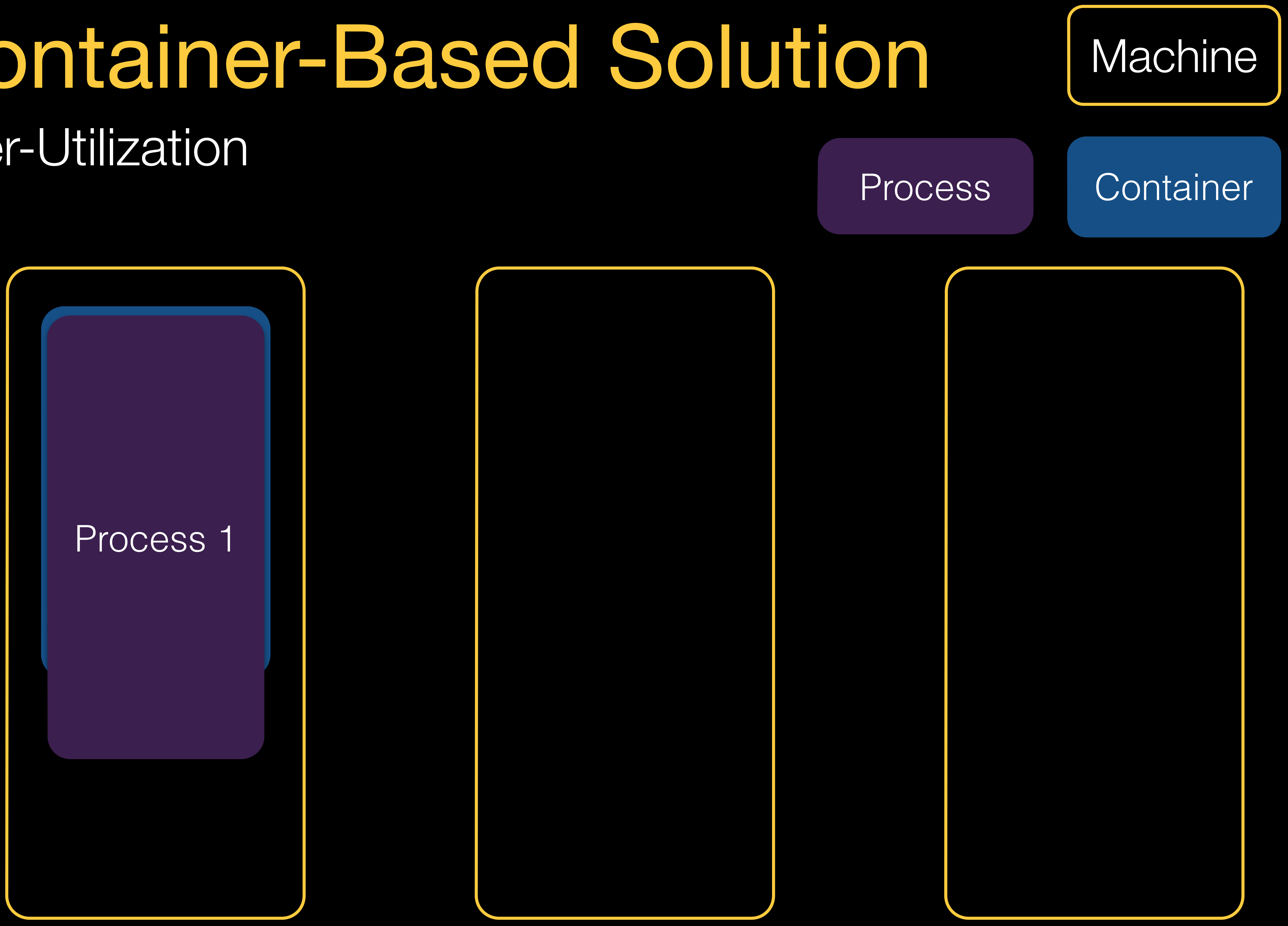
Process

Container



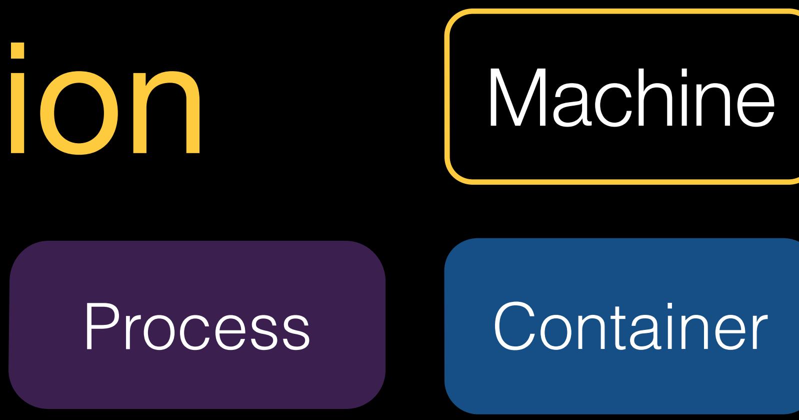
Container-Based Solution

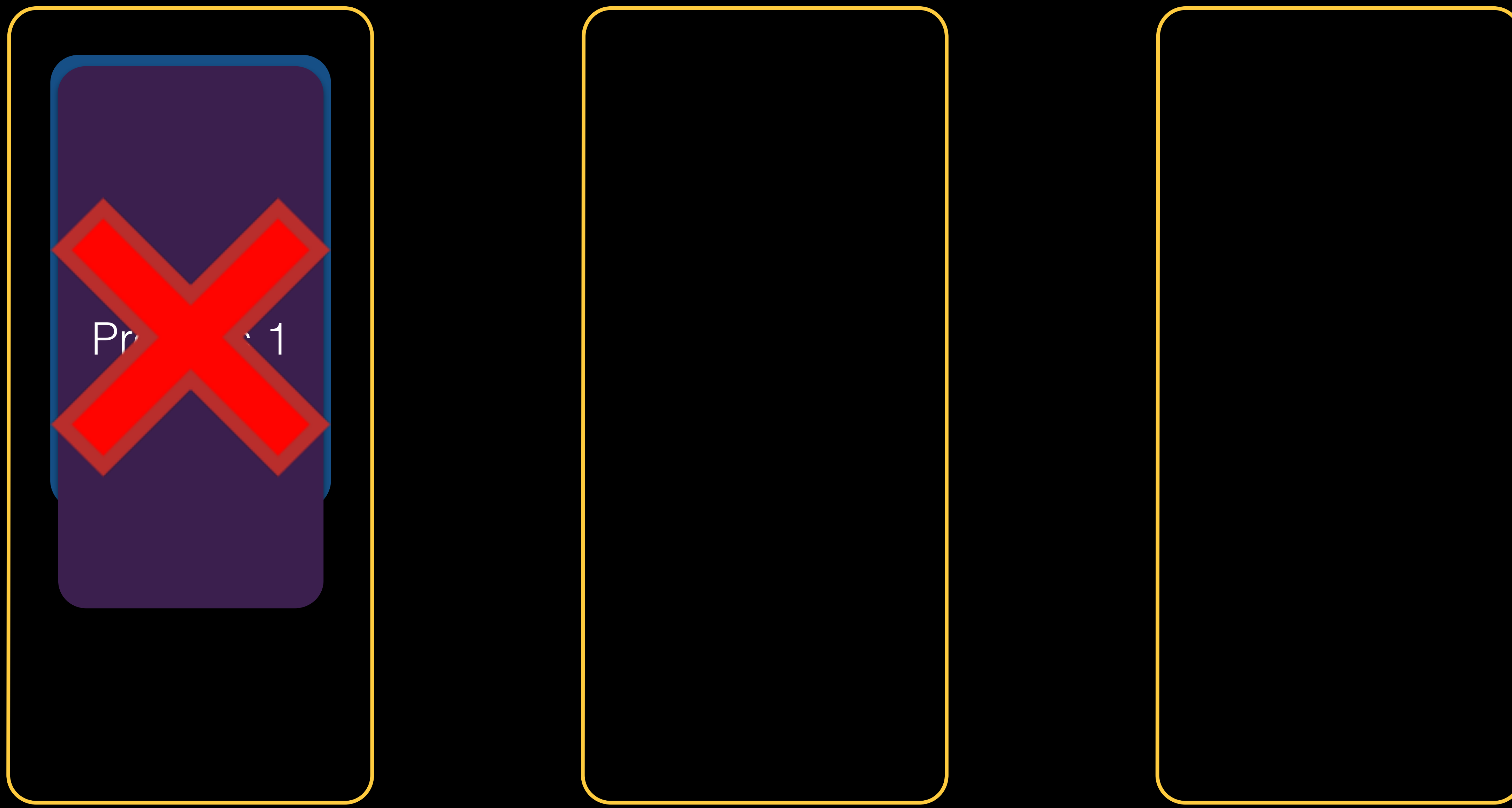
Over-Utilization



Container-Based Solution

Over-Utilization





Outcome: Preemption and relaunch

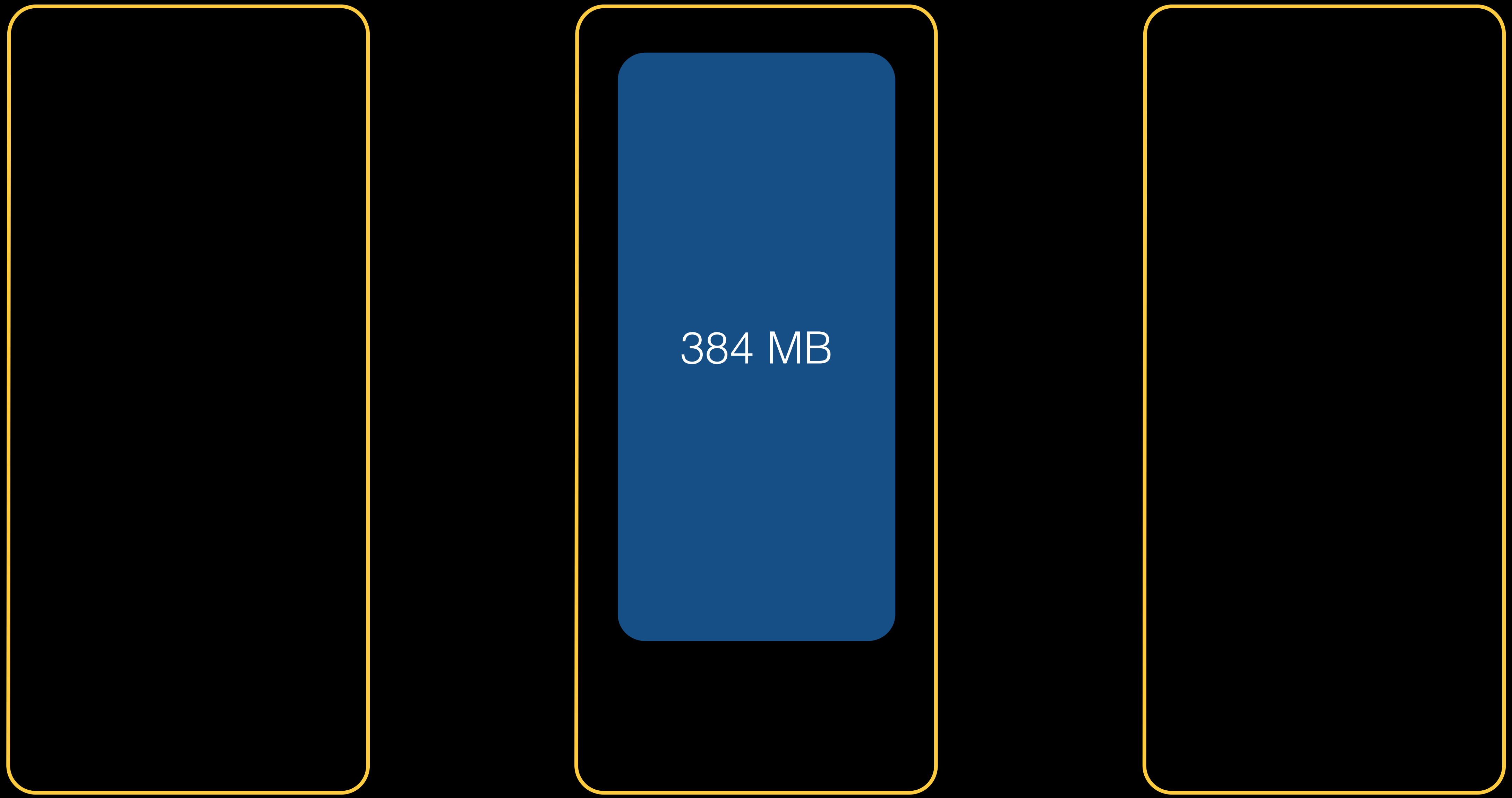
Container-Based Solution

Over-Utilization

Machine

Process

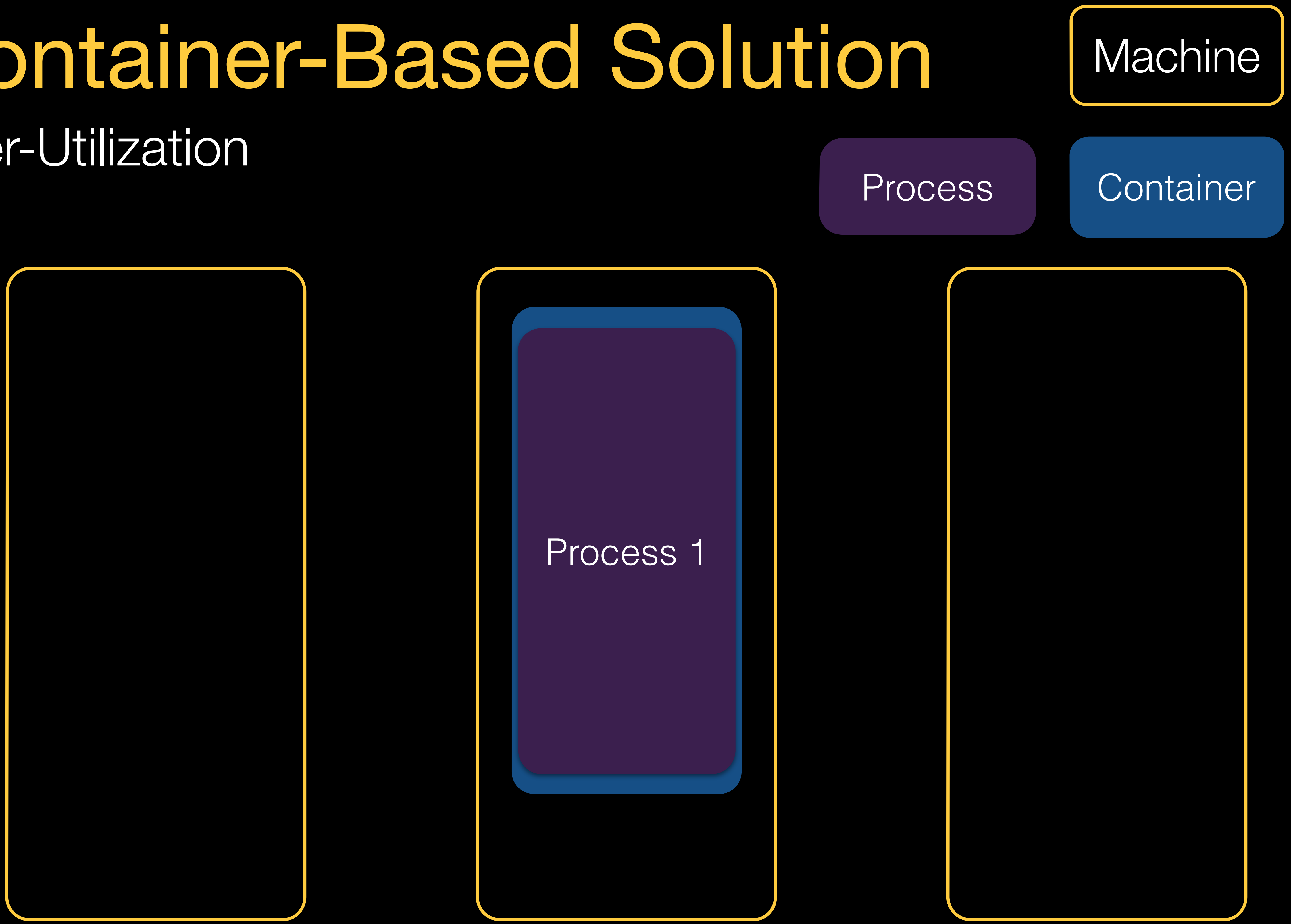
Container



Outcome: Preemption and relaunch

Container-Based Solution

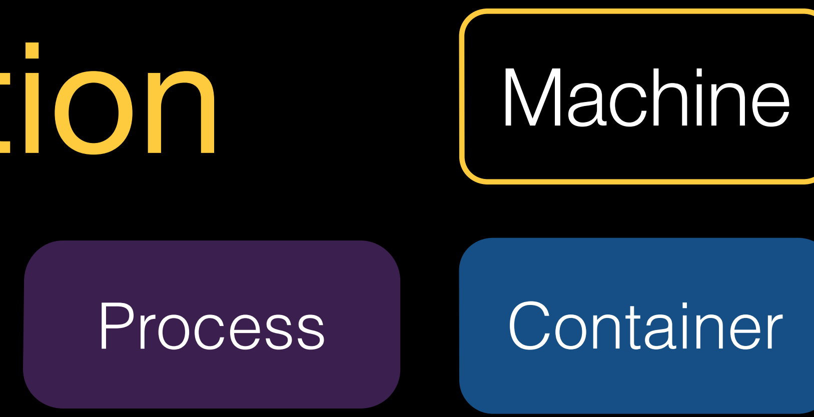
Over-Utilization



Outcome: Preemption and relaunch

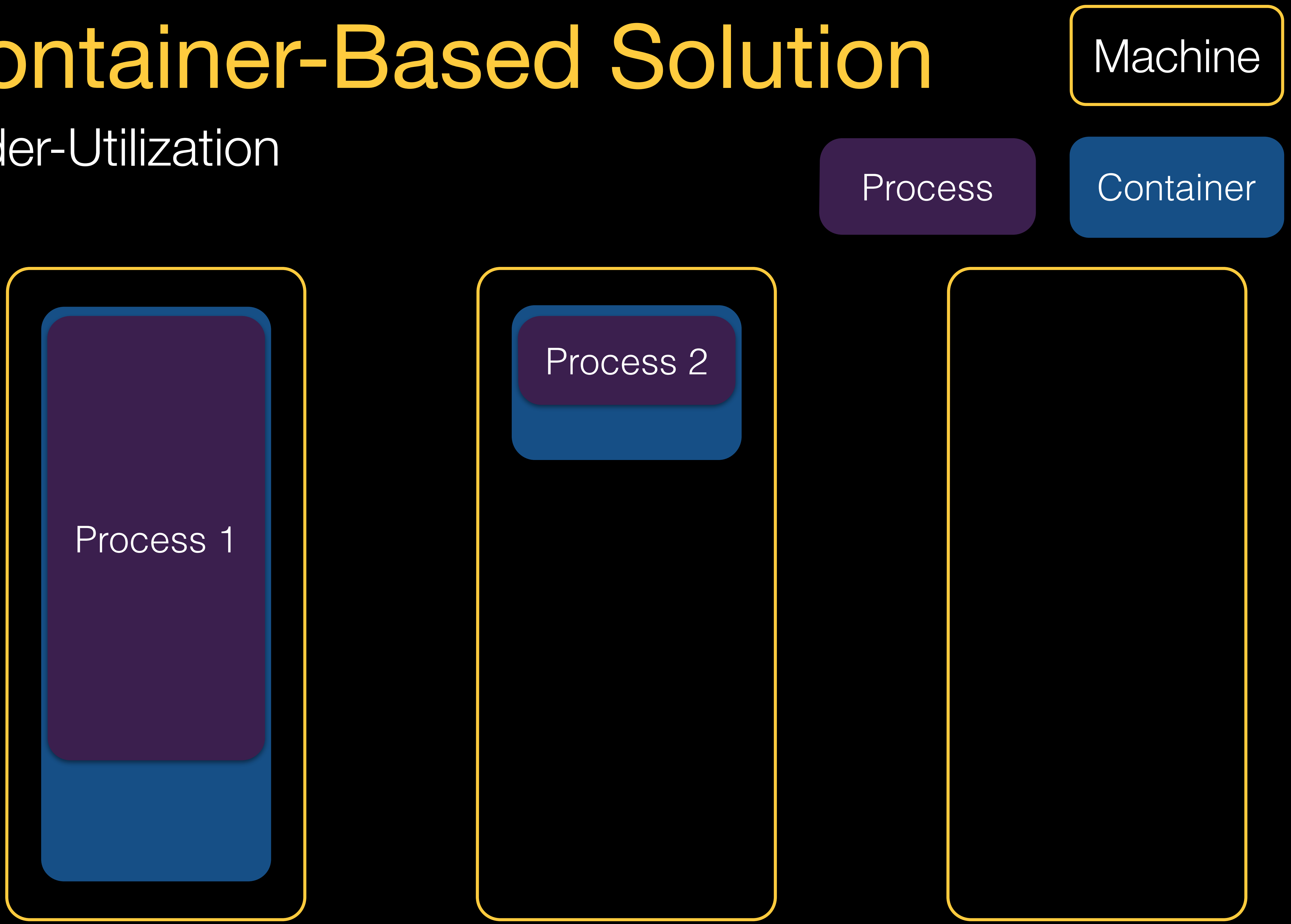
Container-Based Solution

Under-Utilization



Container-Based Solution

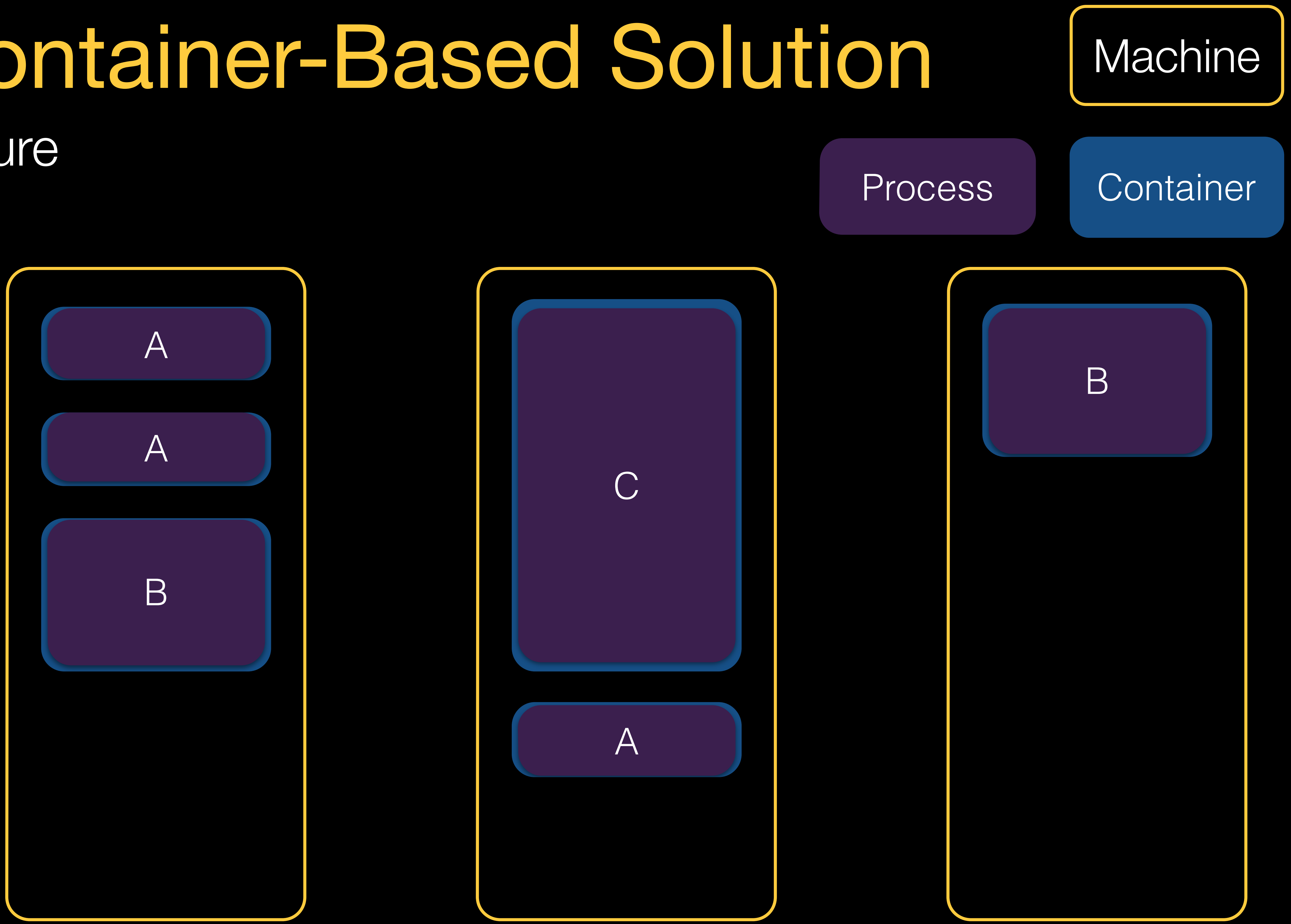
Under-Utilization



Outcome: Over-provisioned until restart

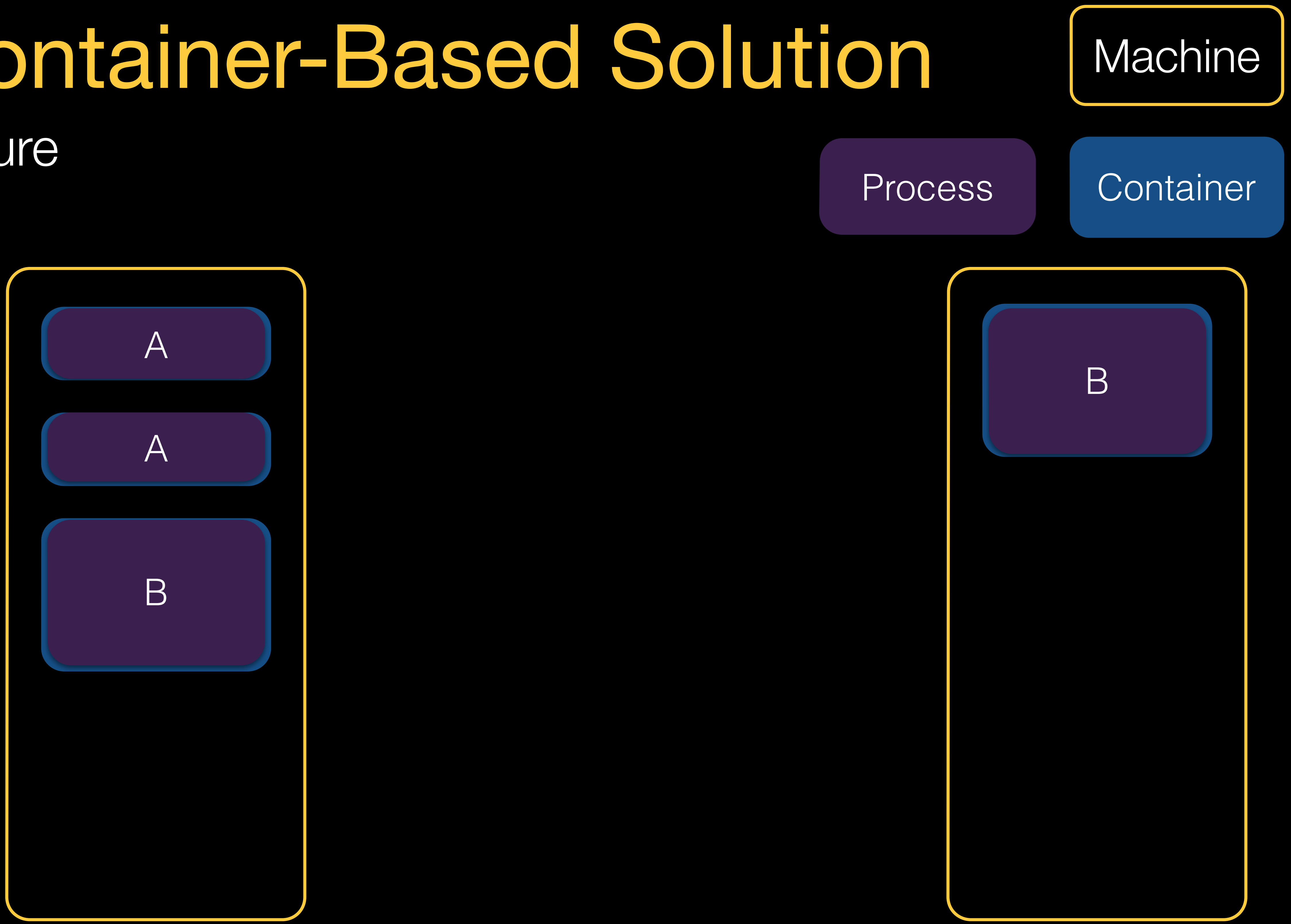
Container-Based Solution

Failure

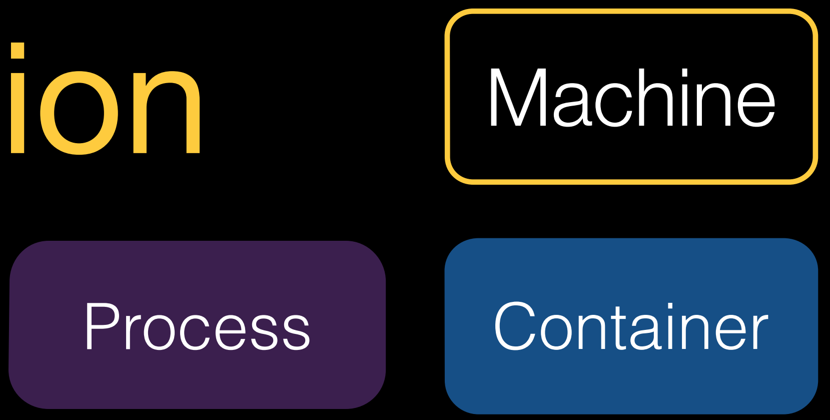


Container-Based Solution

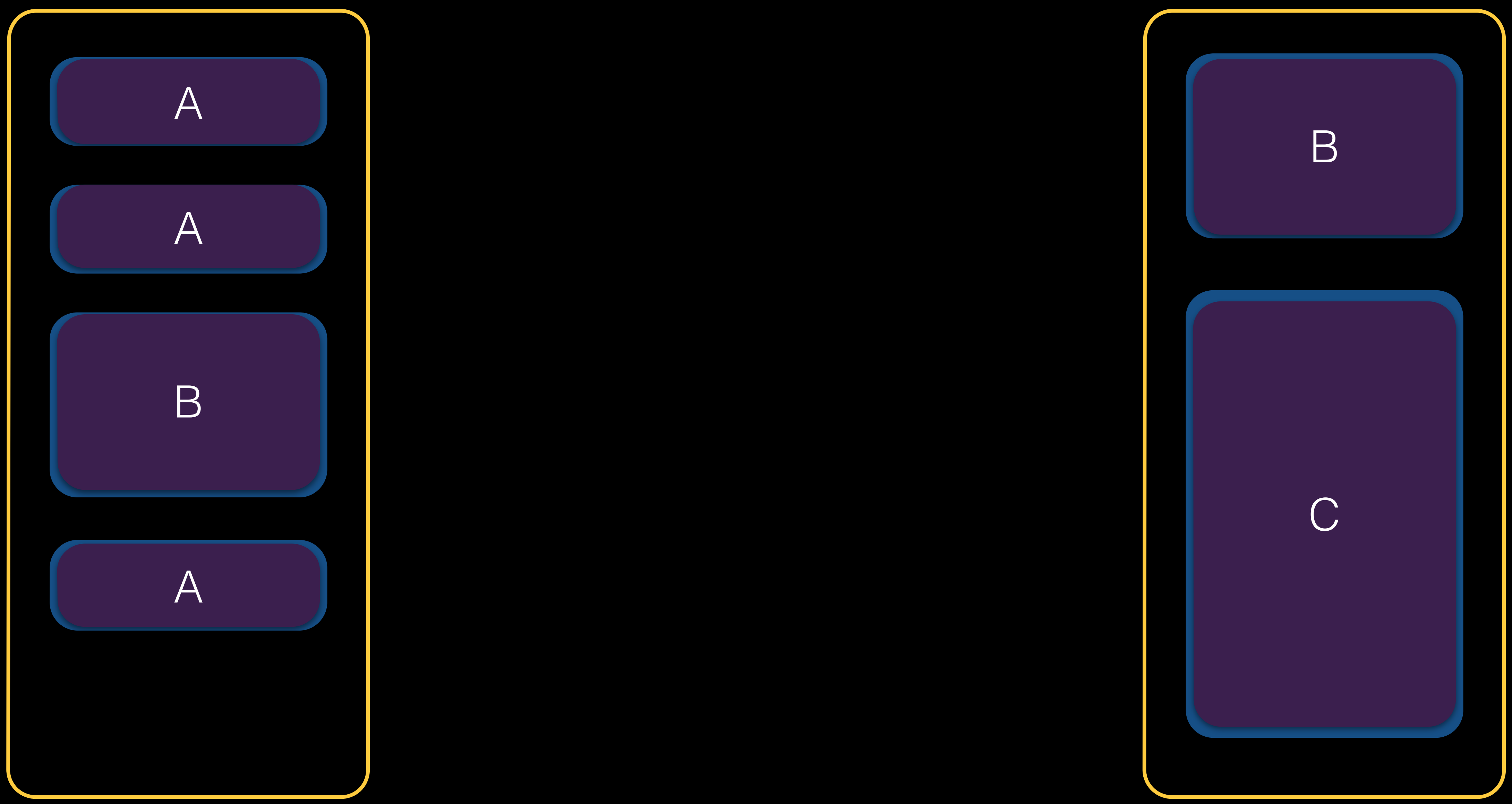
Failure



Container-Based Solution



Failure

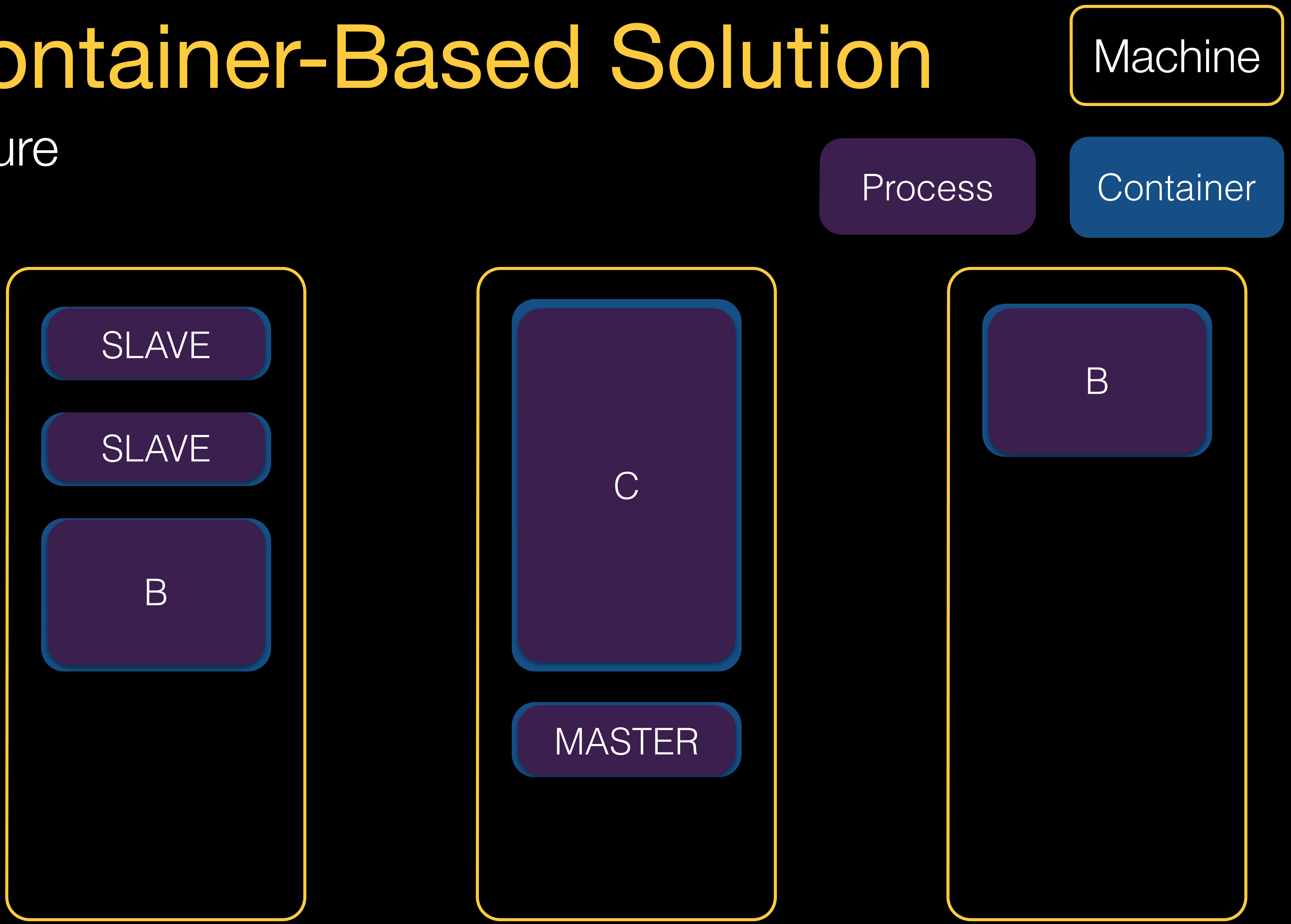


Outcome: Launch containers elsewhere

What about stateful systems?

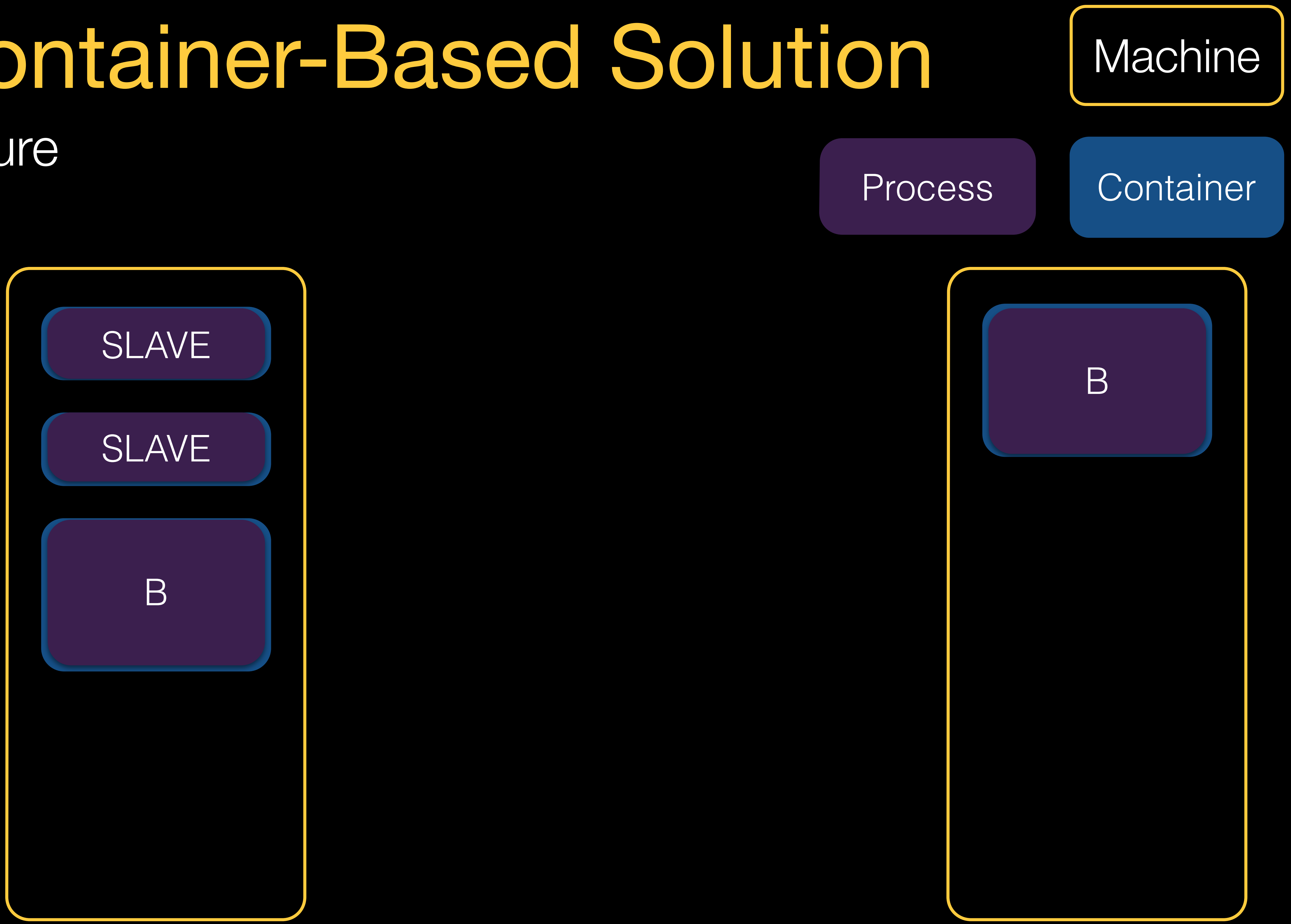
Container-Based Solution

Failure



Container-Based Solution

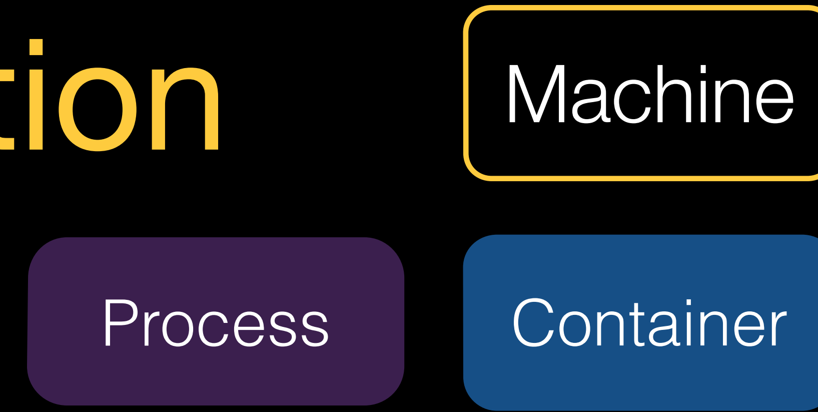
Failure

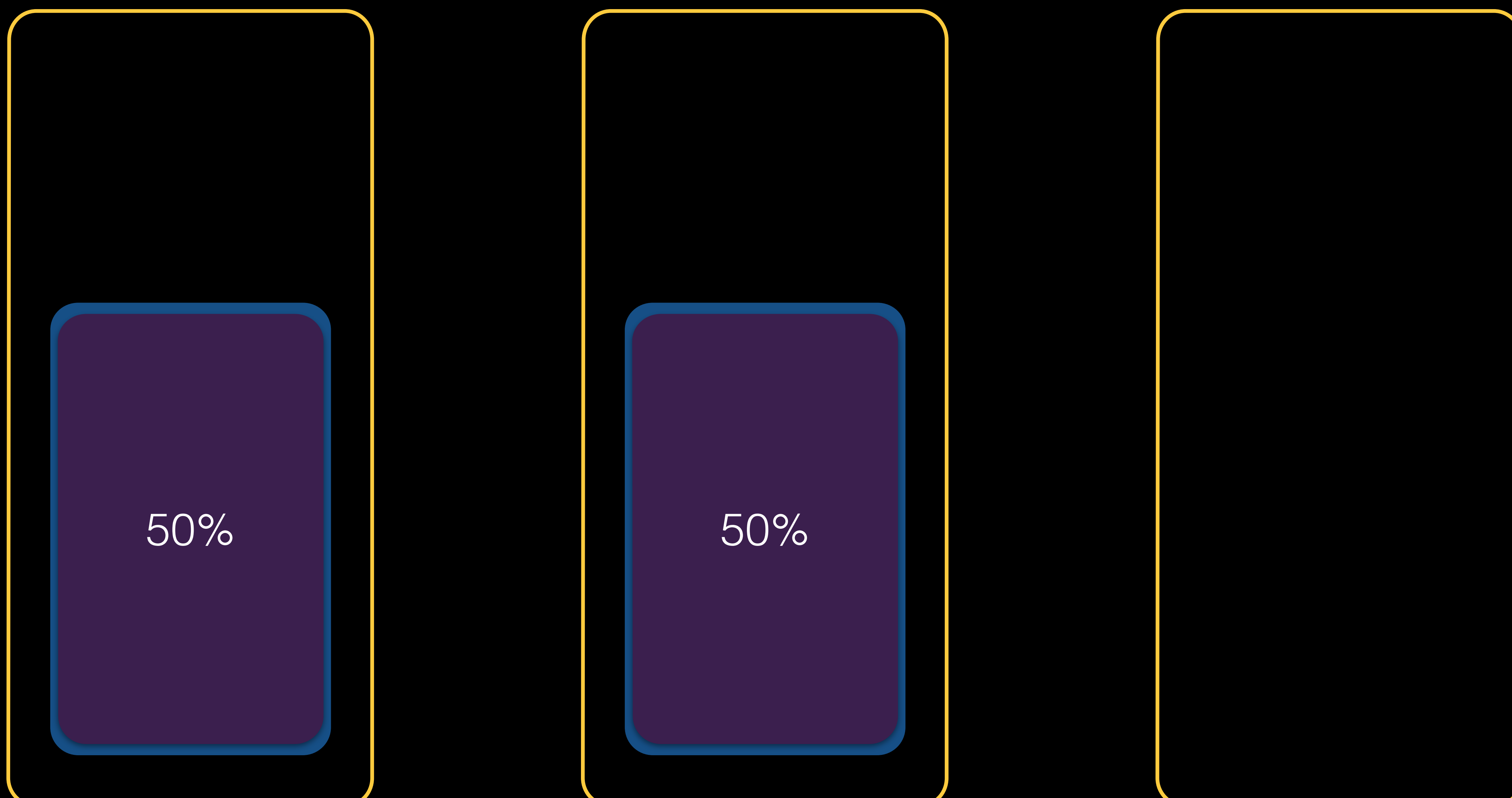


Without additional information, the master is unavailable until restart

Container-Based Solution

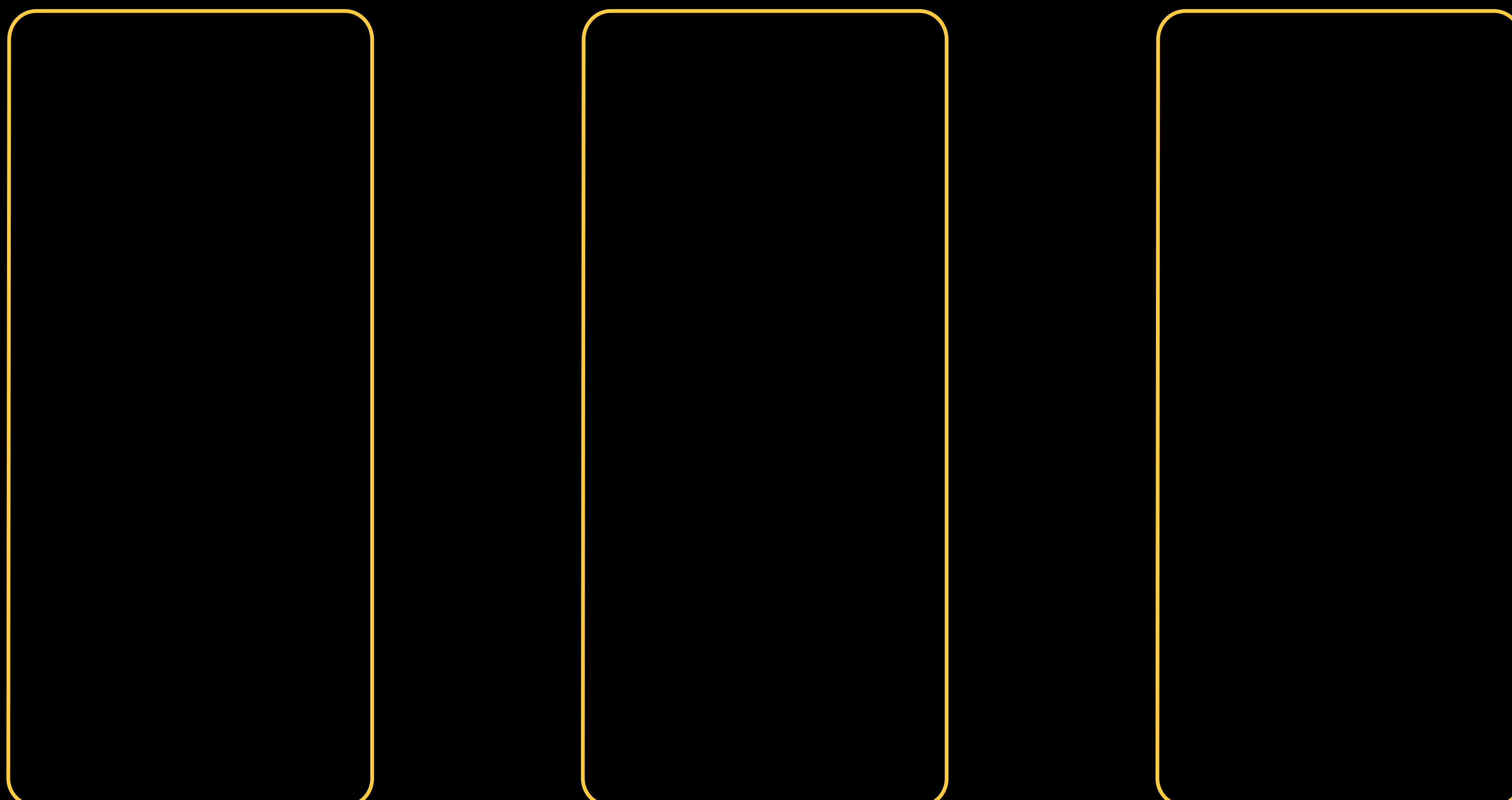
Scaling





Container-Based Solution

Scaling



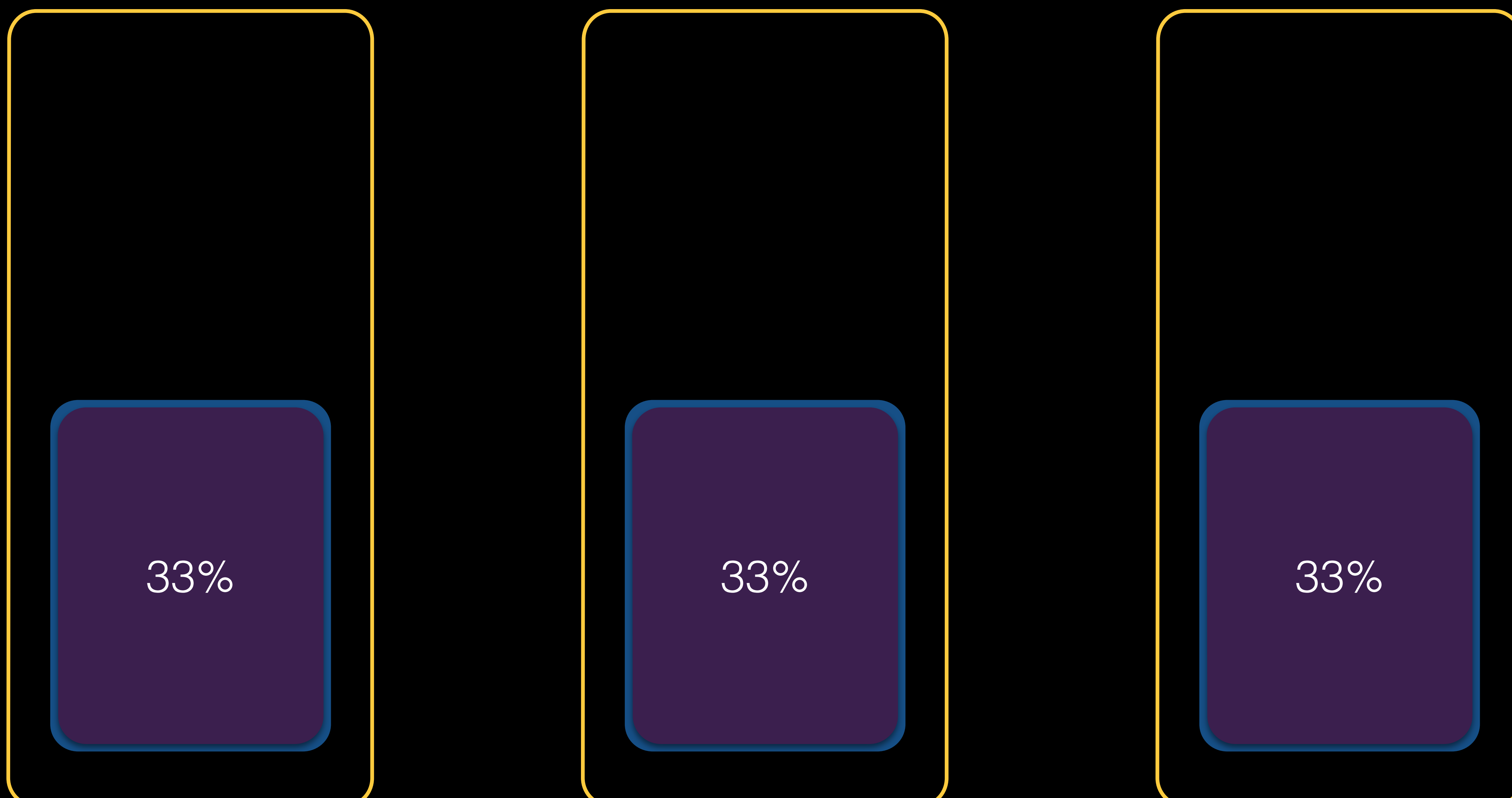
Container-Based Solution

Scaling

Machine

Process

Container



Outcome: Relaunch with new sharding

Container-Based Solution

	Container-Based Solution
Utilization	Application requirements define container size
Fault Tolerance	New container is started
Scaling	Workload is repartitioned and new containers are brought up
Discovery	Existence

HELIX

Container-Based Solution

The container model provides flexibility within machines, but assumes homogeneity of tasks within containers

We need something finer-grained

Task-Based Solution

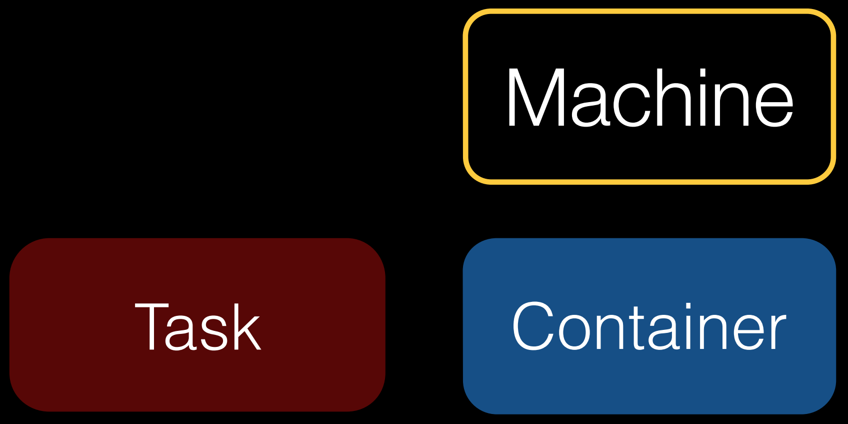
Task-Based Solution

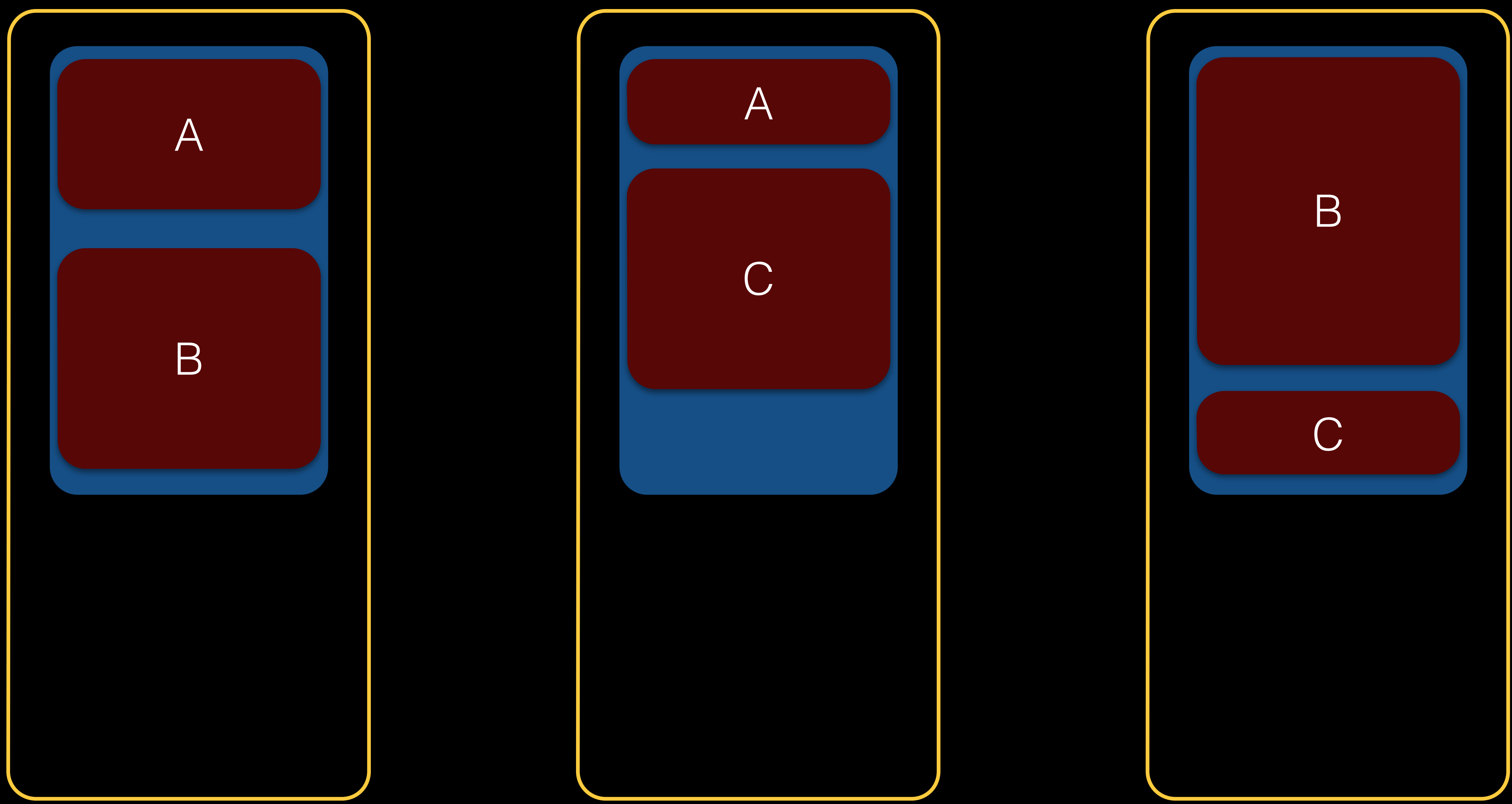
System Requirements

A	complete in less than 5 hours
B	always have 2 containers running
C	response time should be less than 50 ms

Task-Based Solution

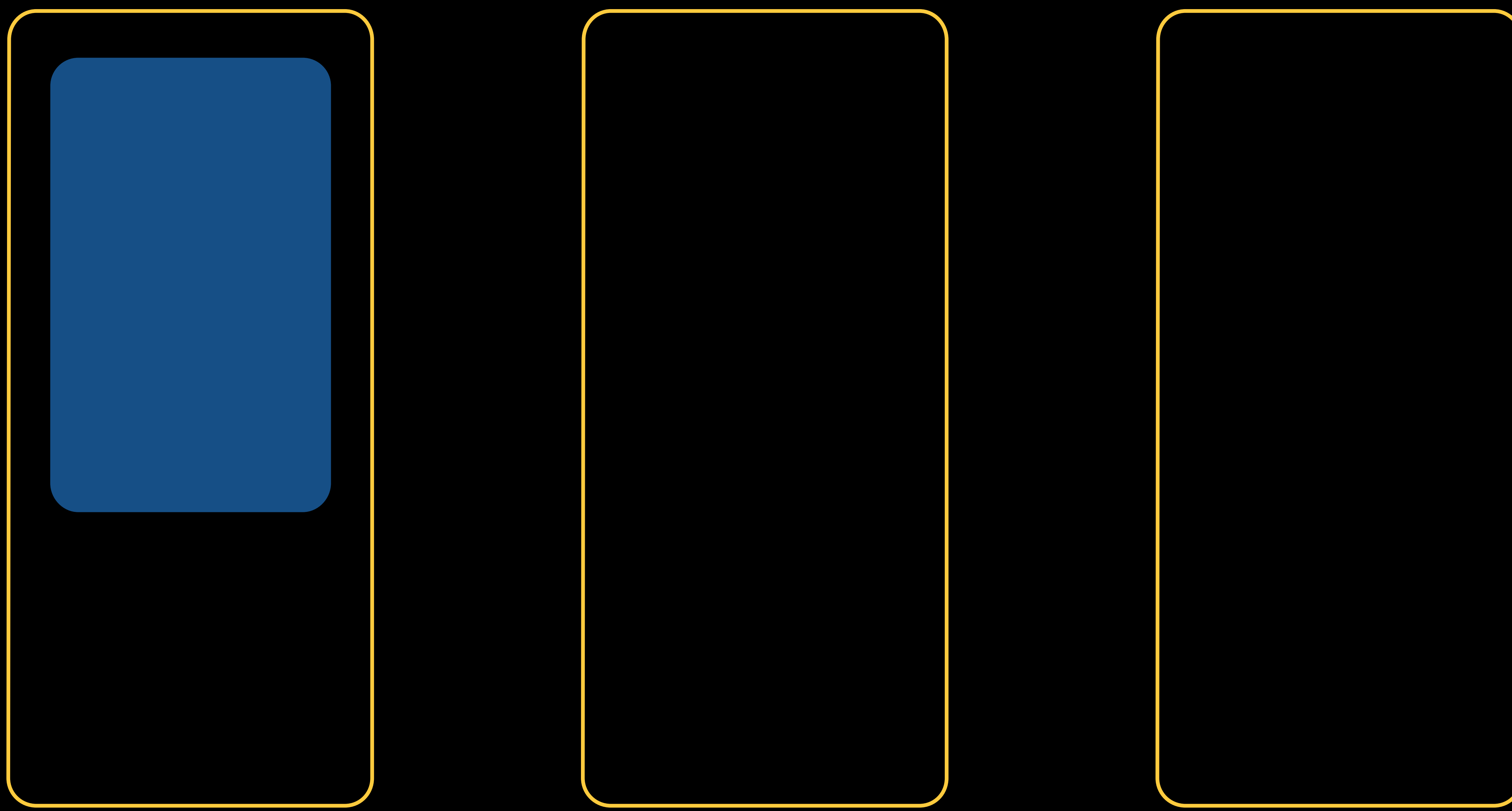
Allocation





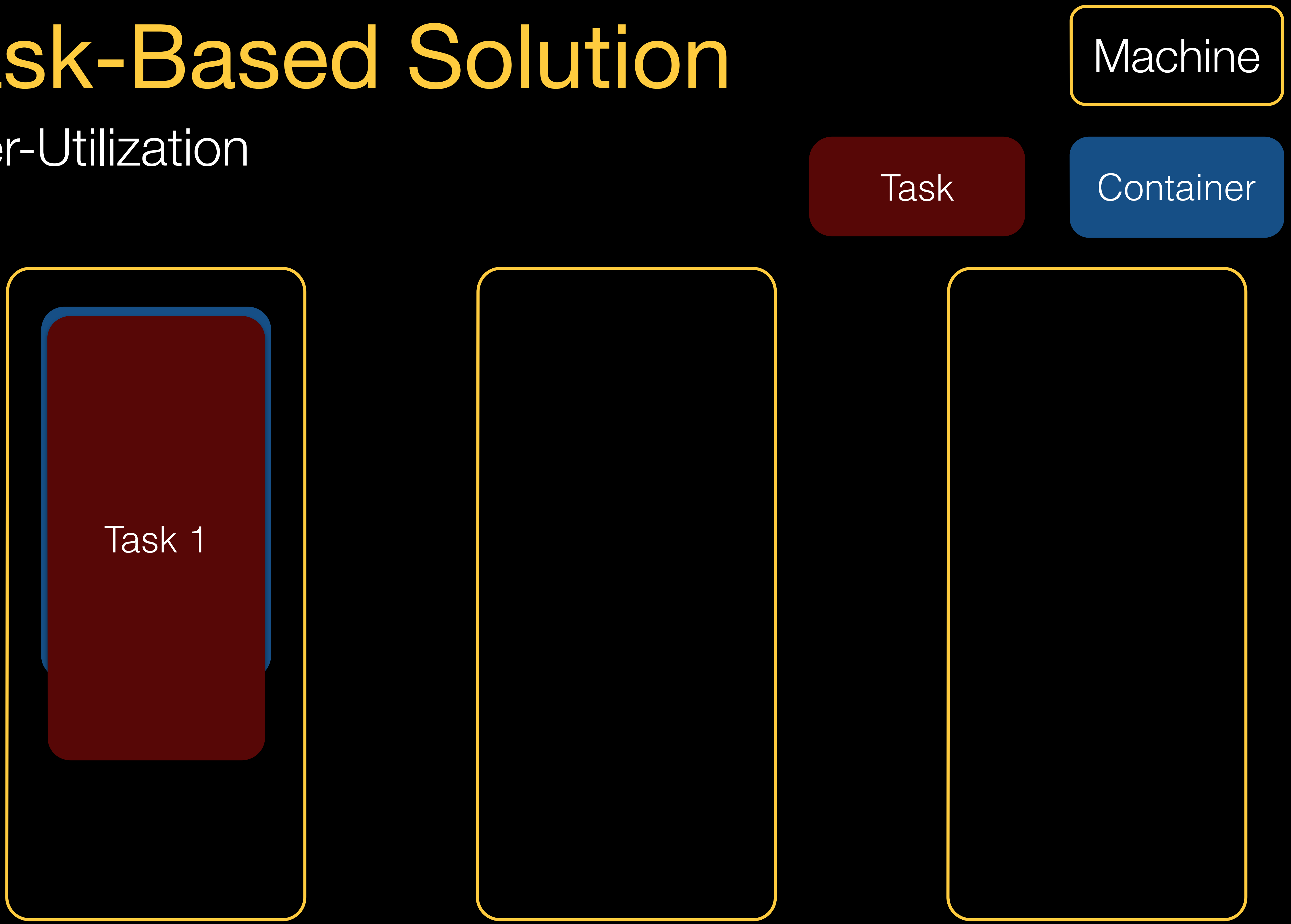
Task-Based Solution

Over-Utilization



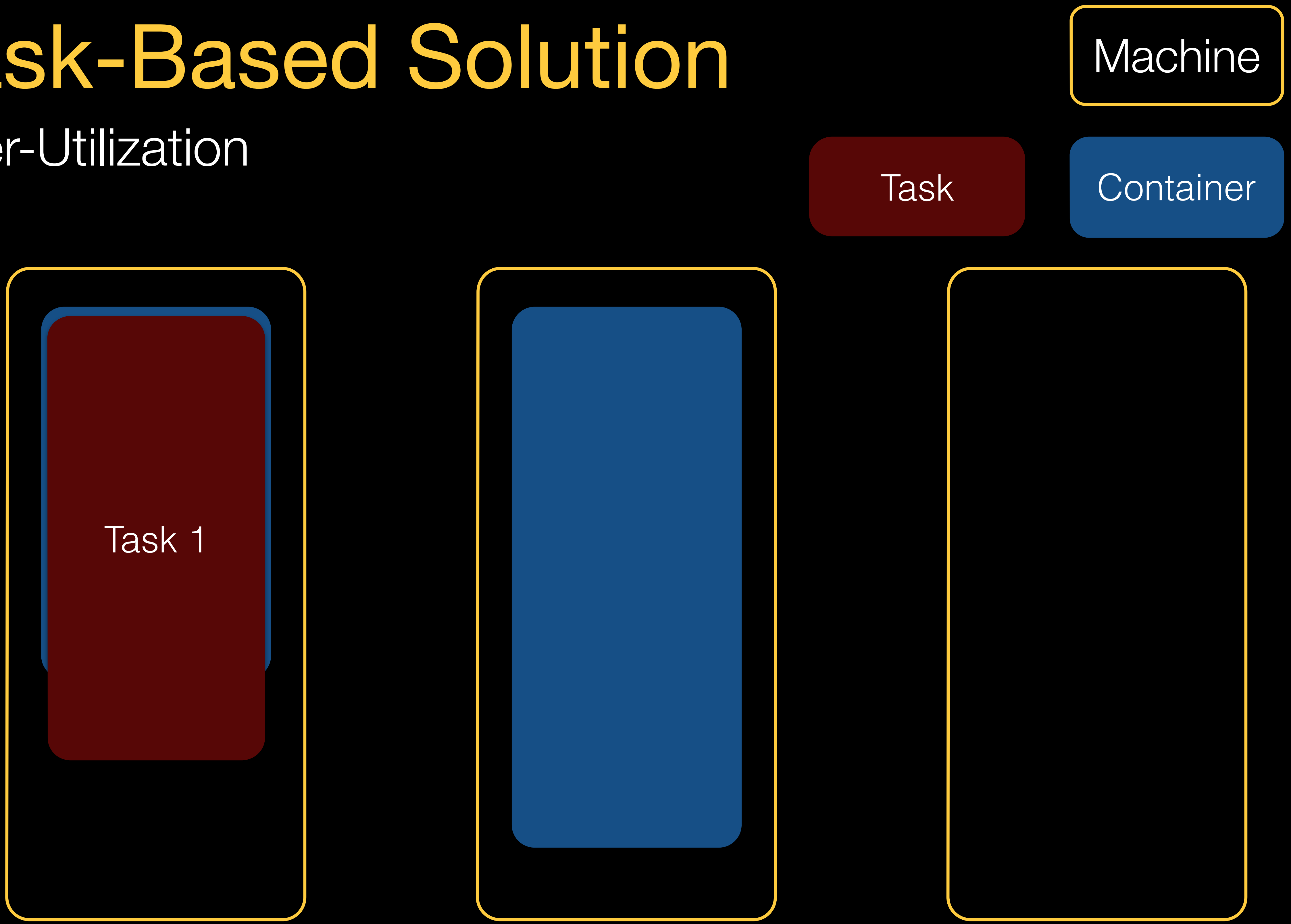
Task-Based Solution

Over-Utilization



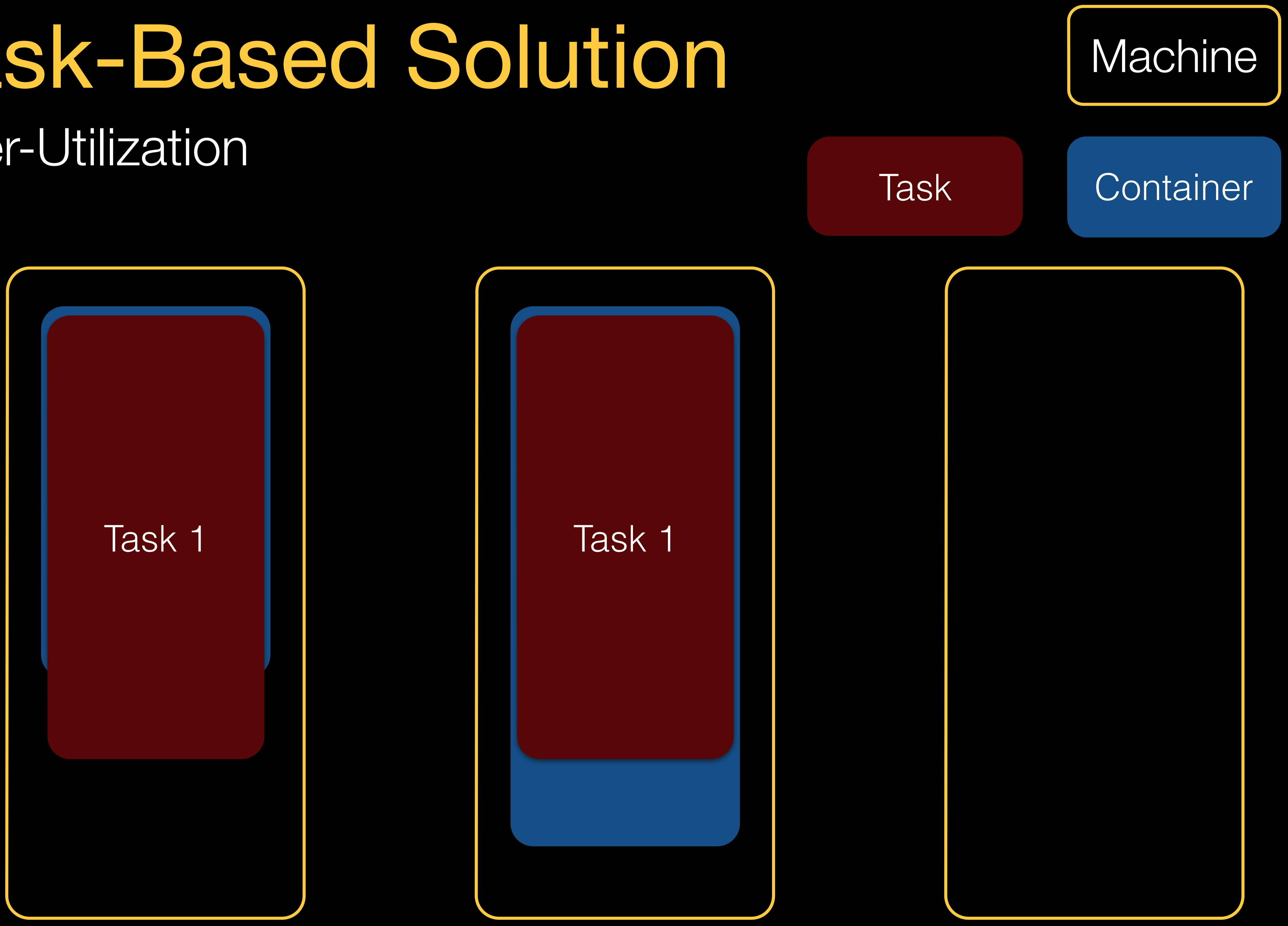
Task-Based Solution

Over-Utilization



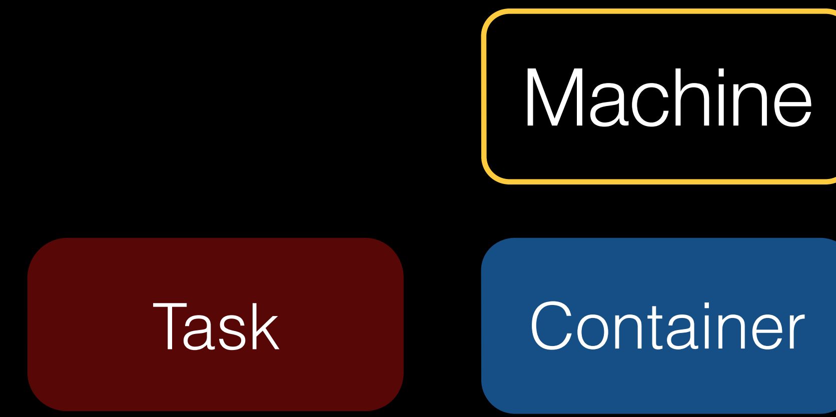
Task-Based Solution

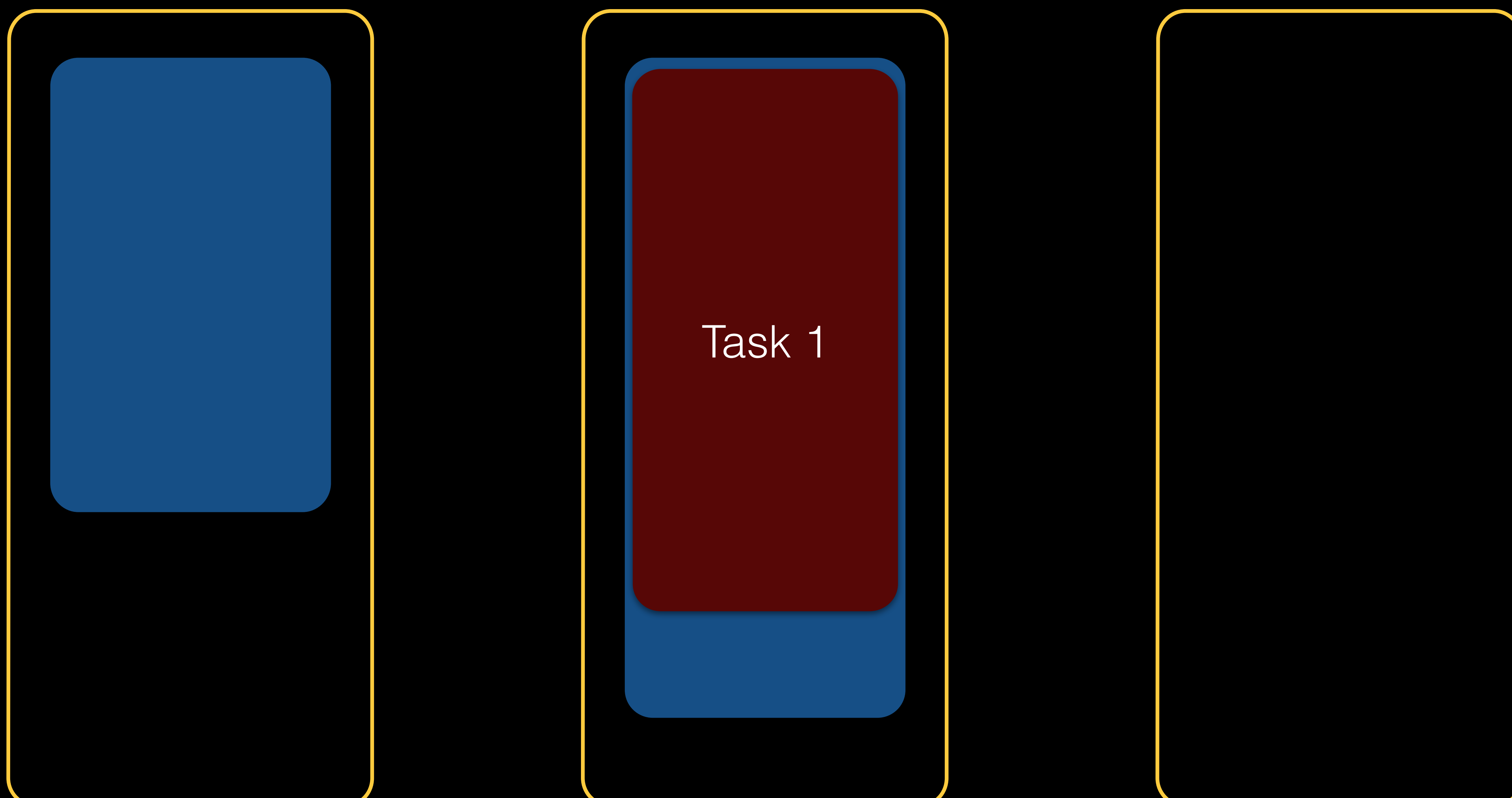
Over-Utilization



Task-Based Solution

Over-Utilization

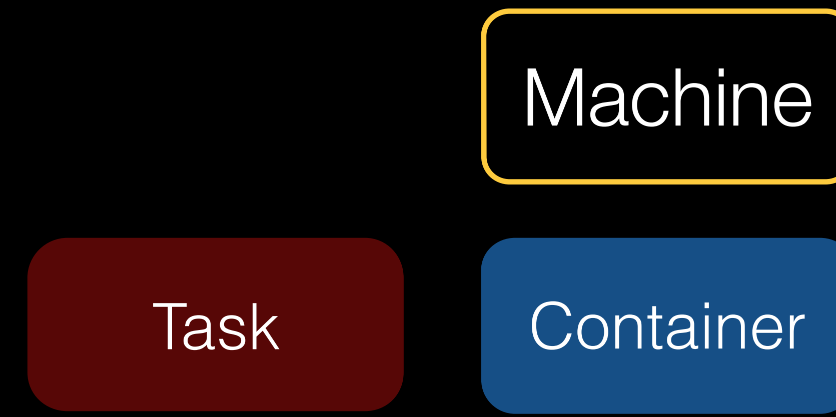




Hide the overhead of a container restart

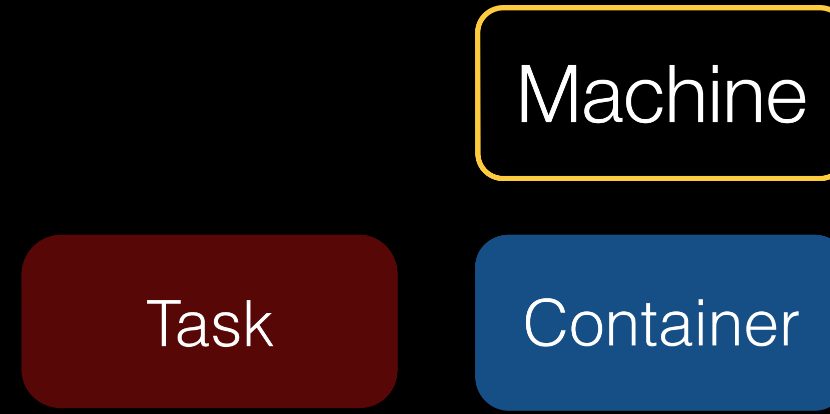
Task-Based Solution

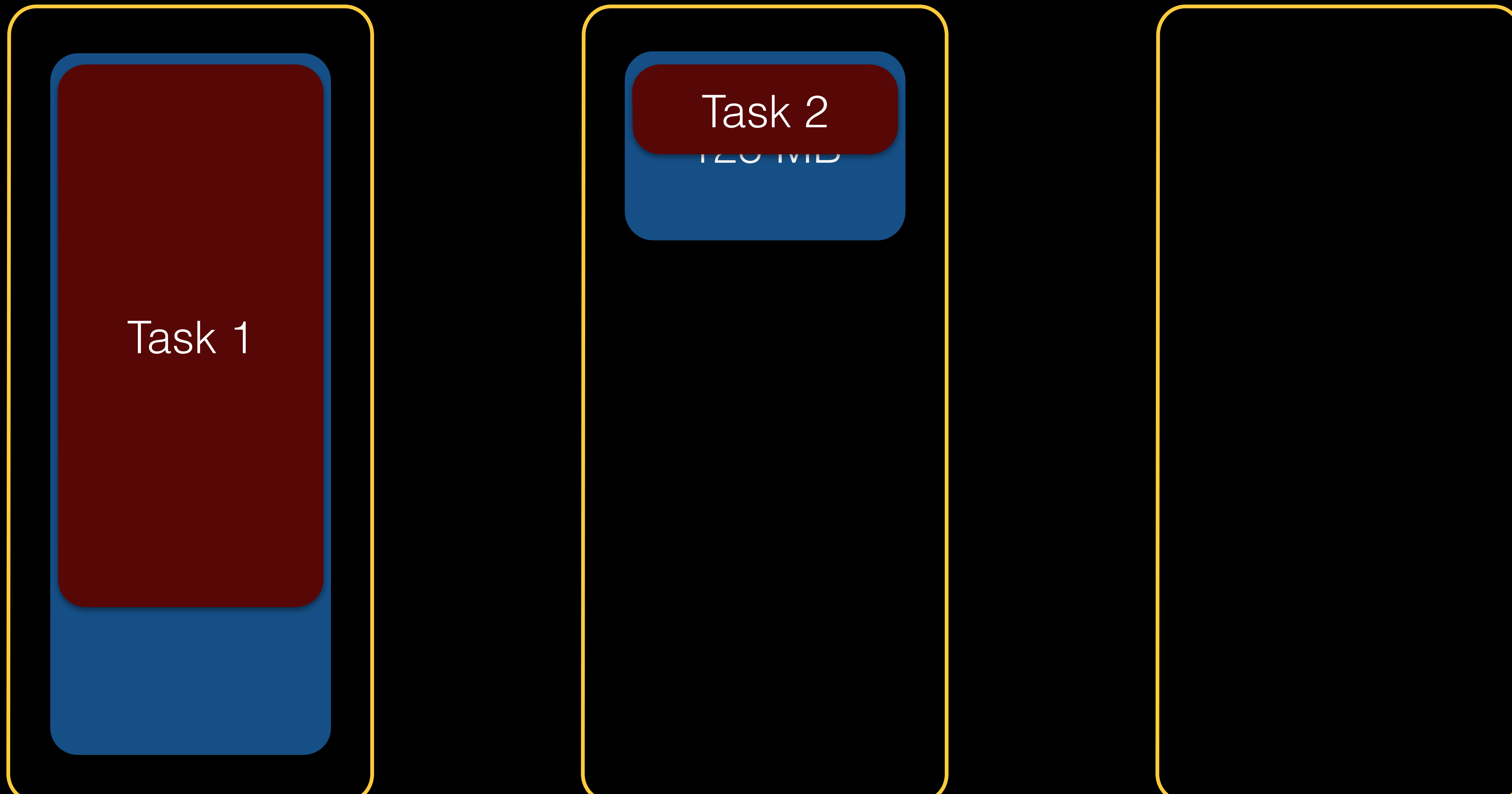
Under-Utilization



Task-Based Solution

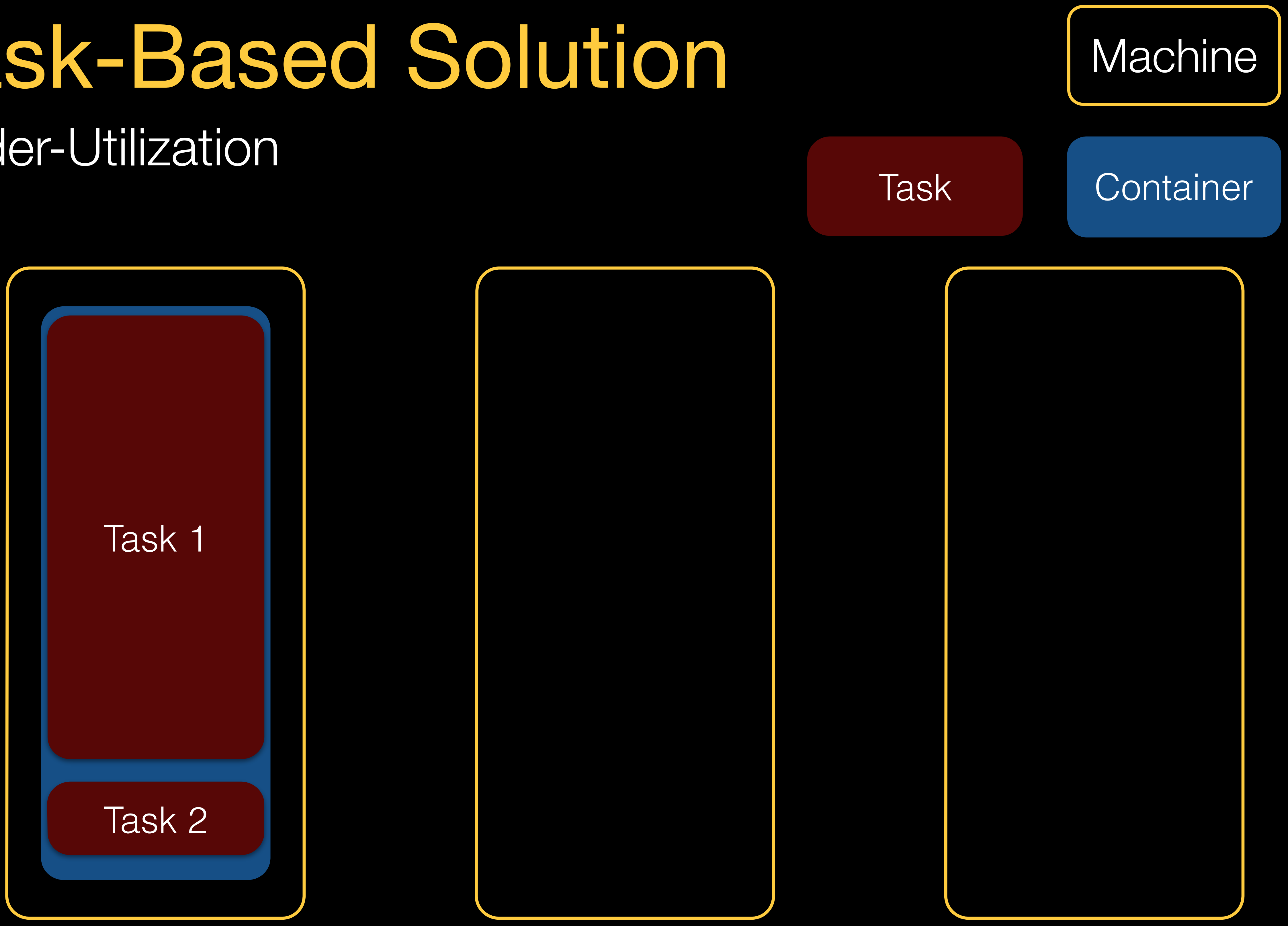
Under-Utilization





Task-Based Solution

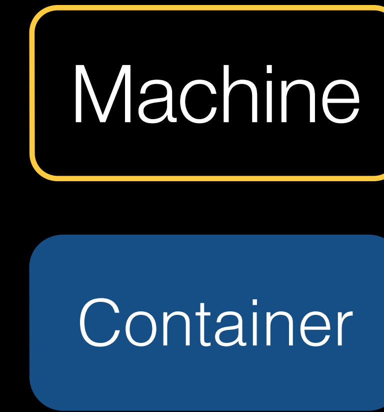
Under-Utilization

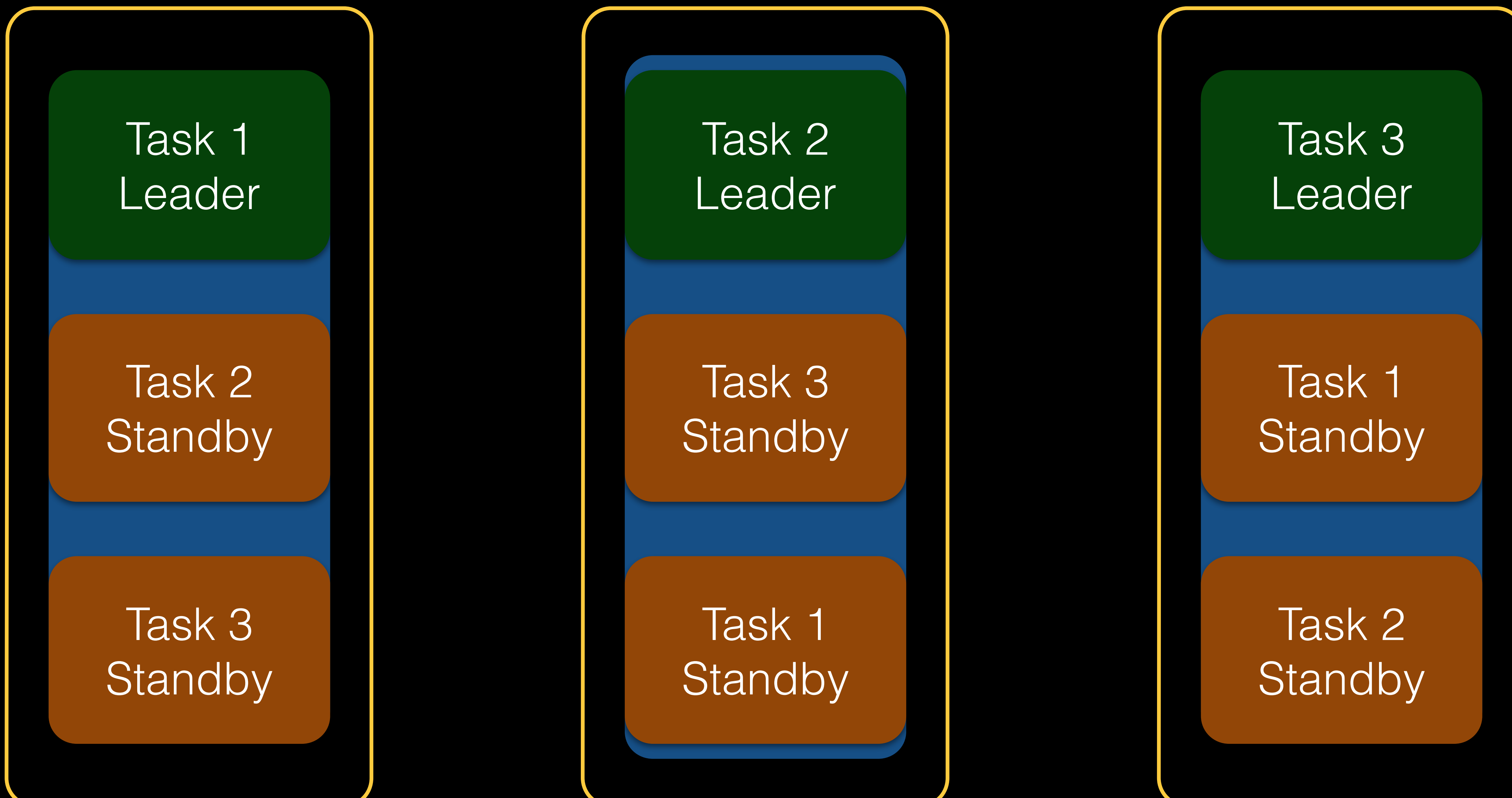


Optimize container allocations based on usage

Task-Based Solution

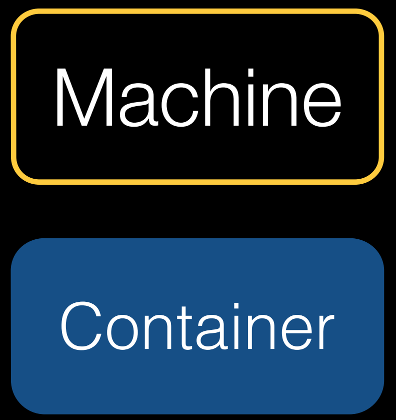
Failure

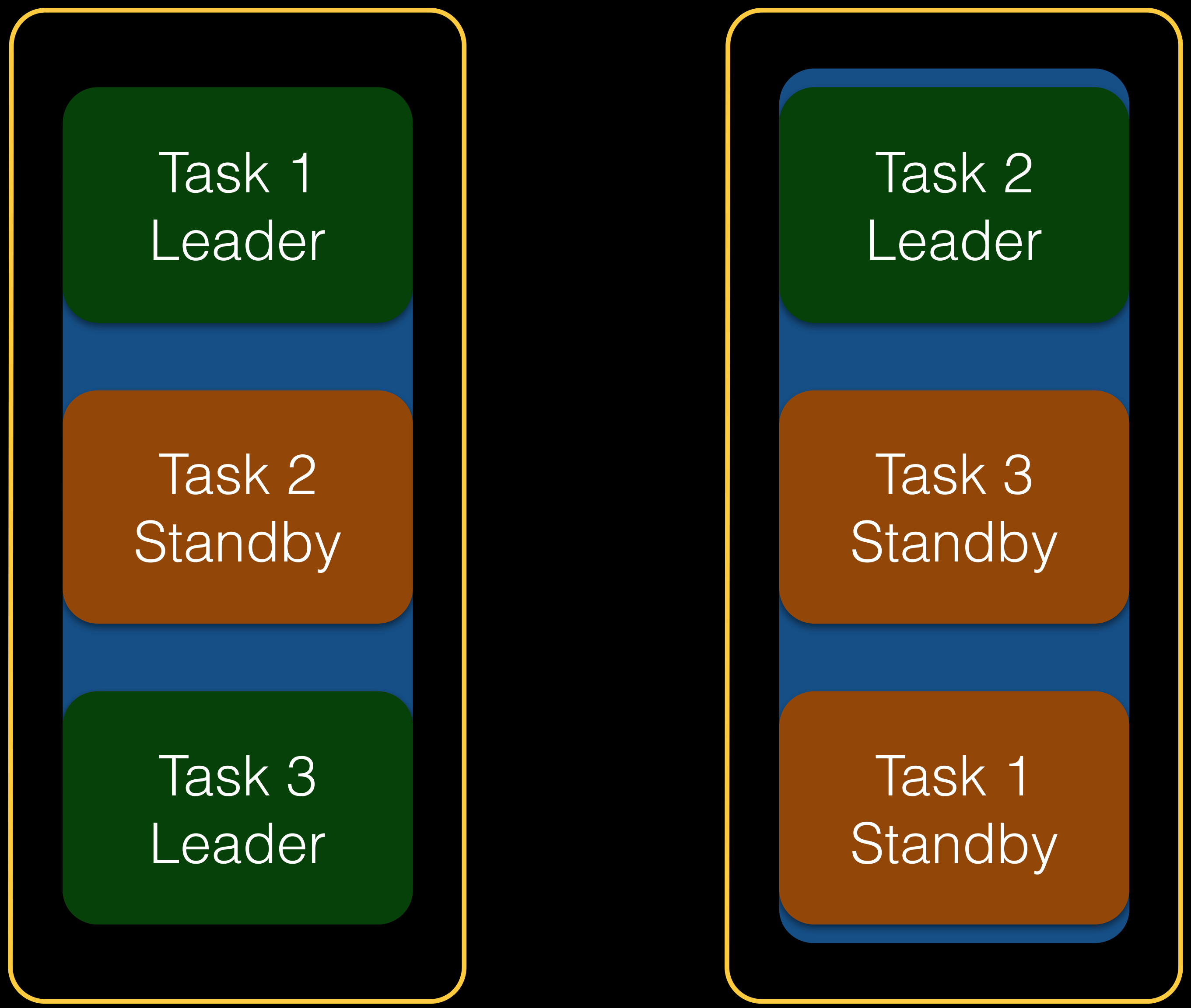




Task-Based Solution

Failure



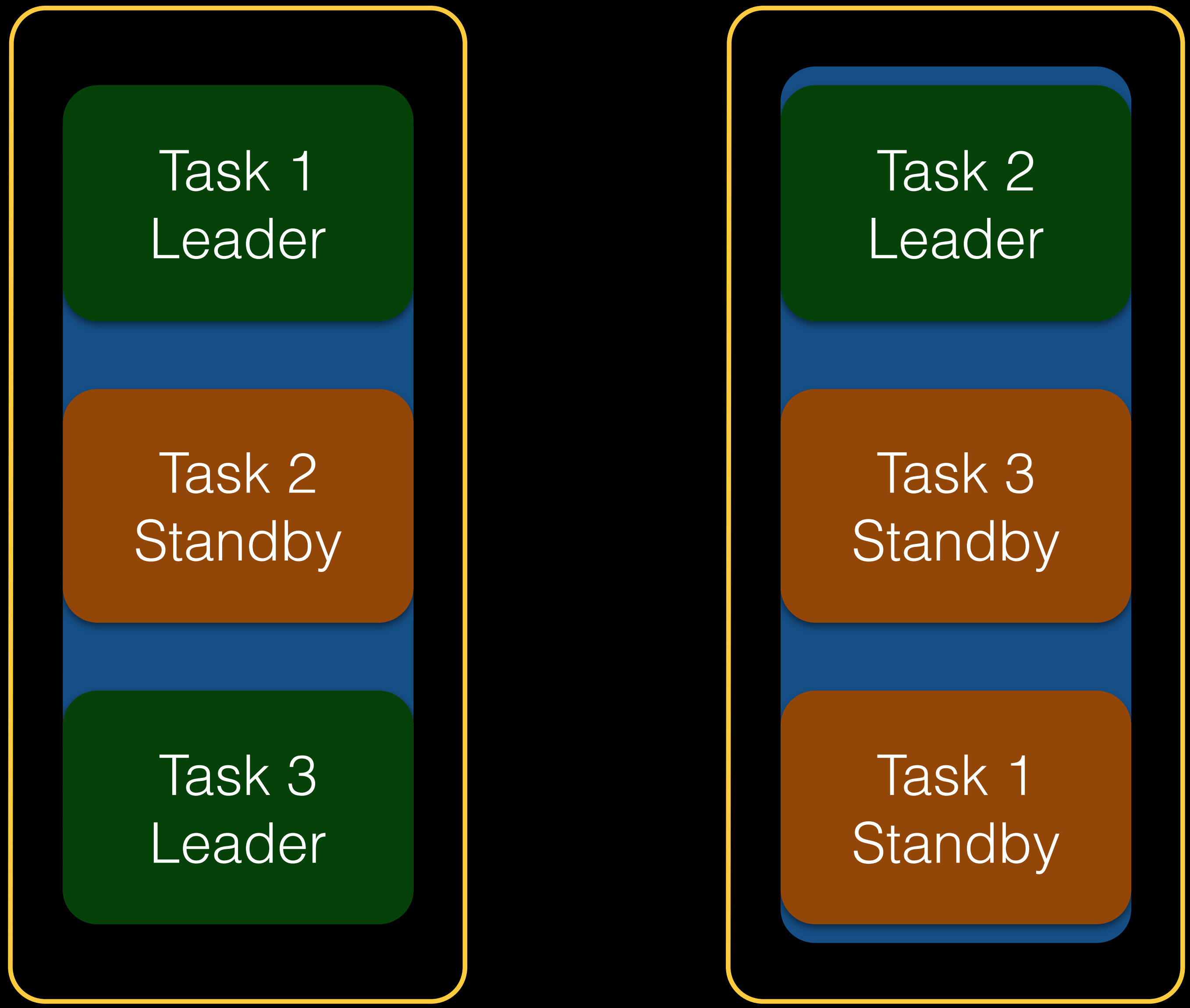


Task-Based Solution

Failure

Machine

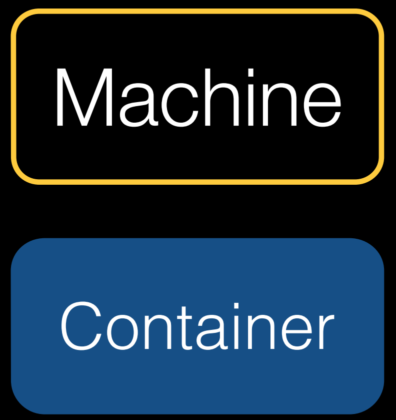
Container

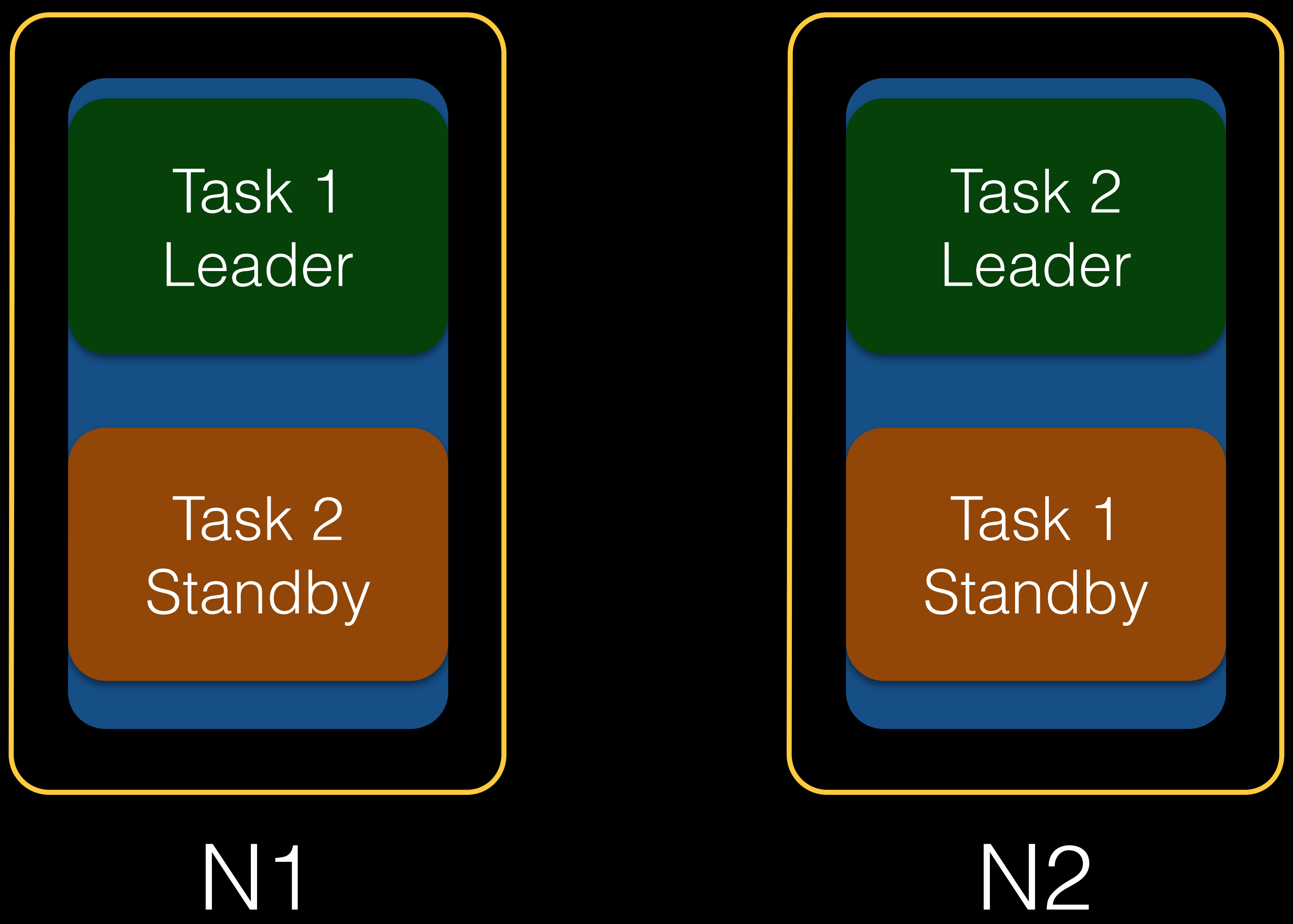


Some systems cannot wait for new containers to start

Task-Based Solution

Discovery



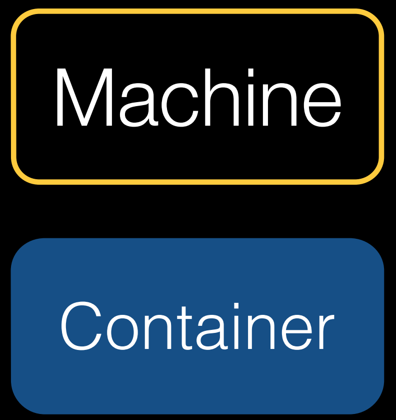


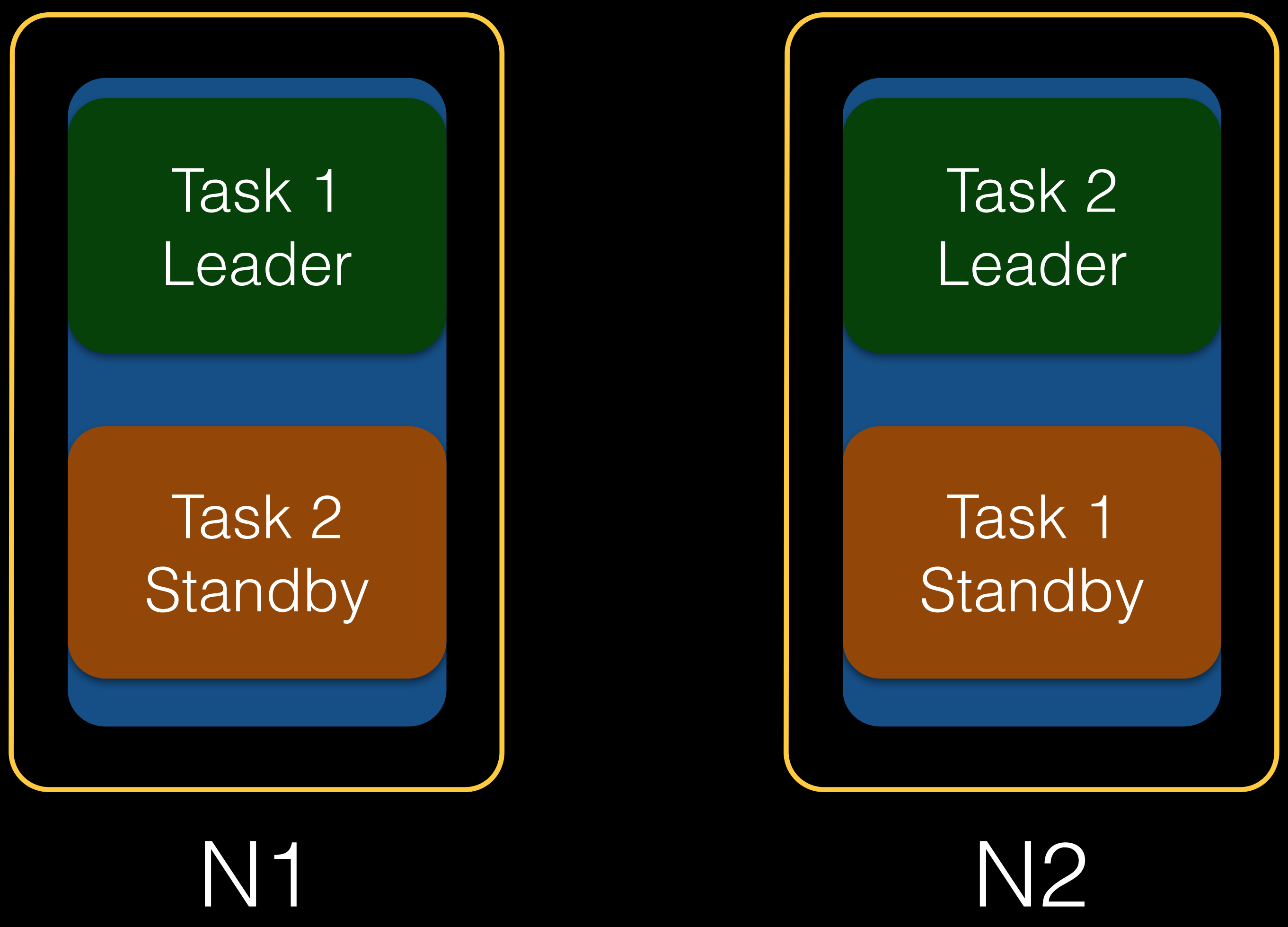
Task 1:
Leader at N1
Standby at N2

Task 2:
Leader at N2
Standby at N1

Task-Based Solution

Discovery





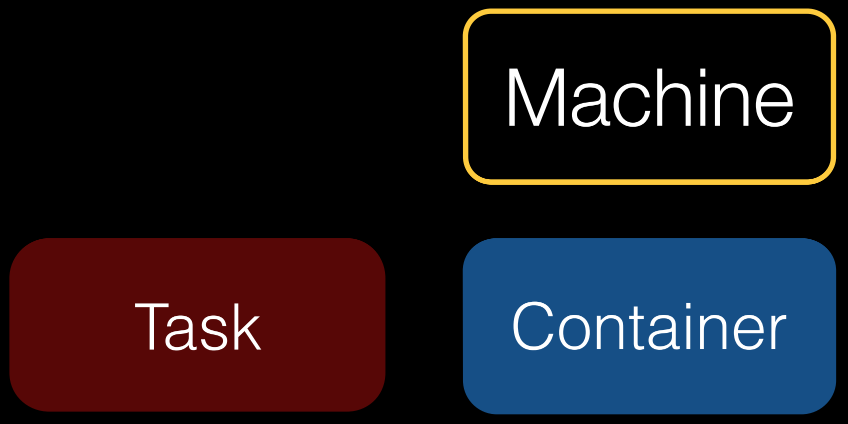
Task 1:
Leader at N1
Standby at N2

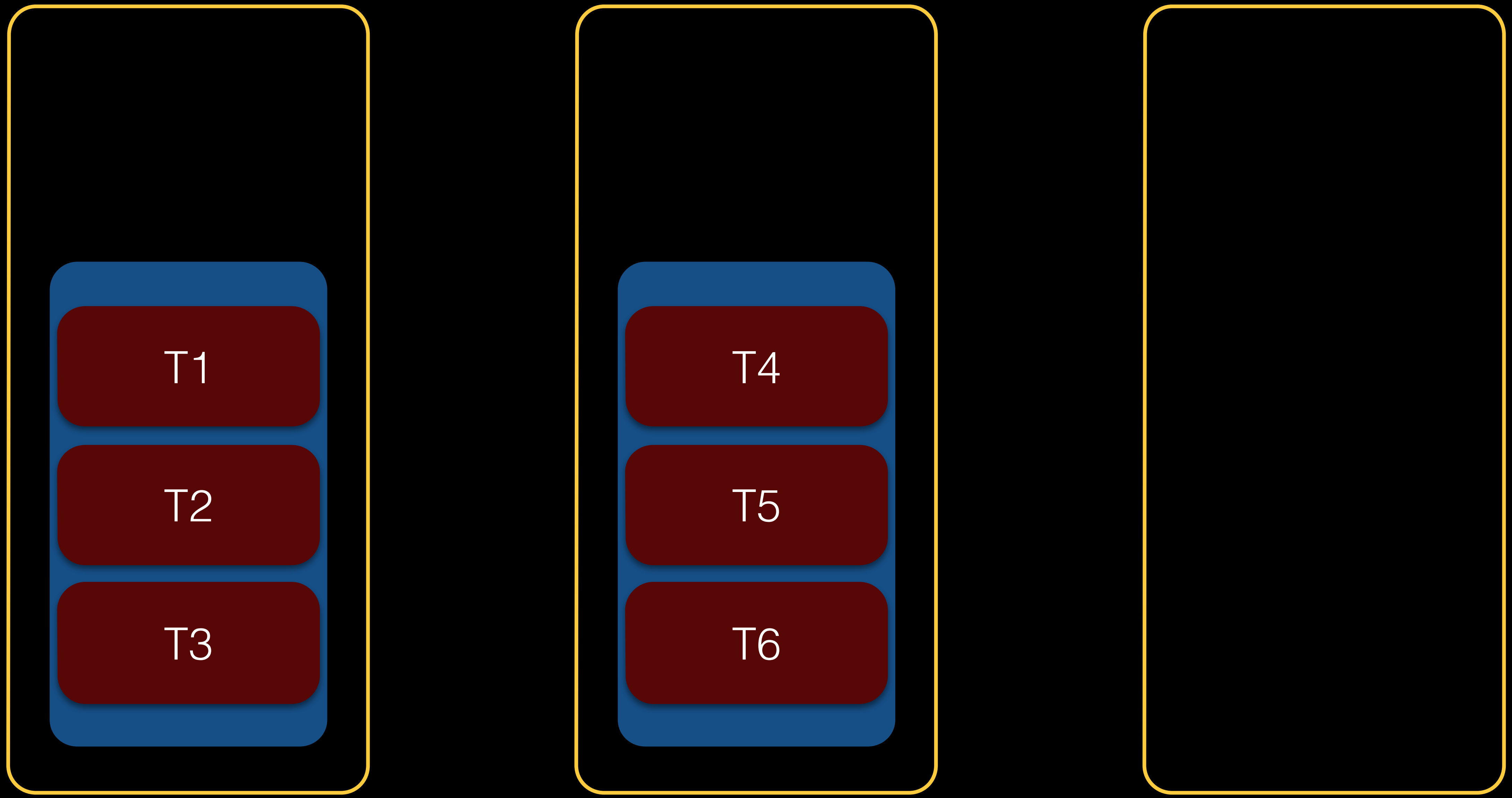
Task 2:
Leader at N2
Standby at N1

Learn where everything runs, and what state each task is in

Task-Based Solution

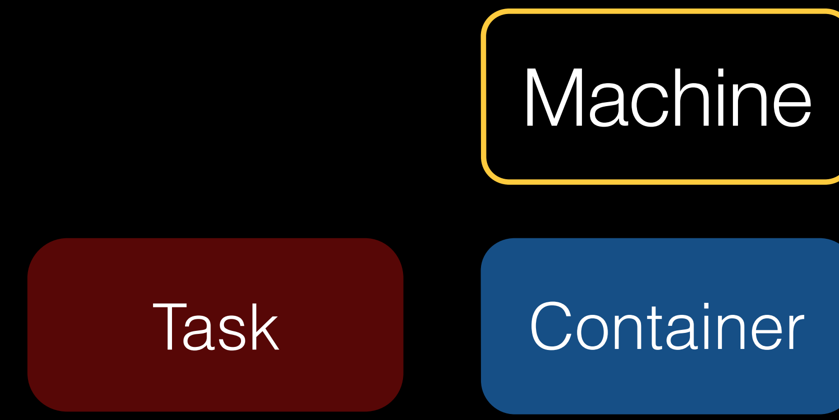
Scaling

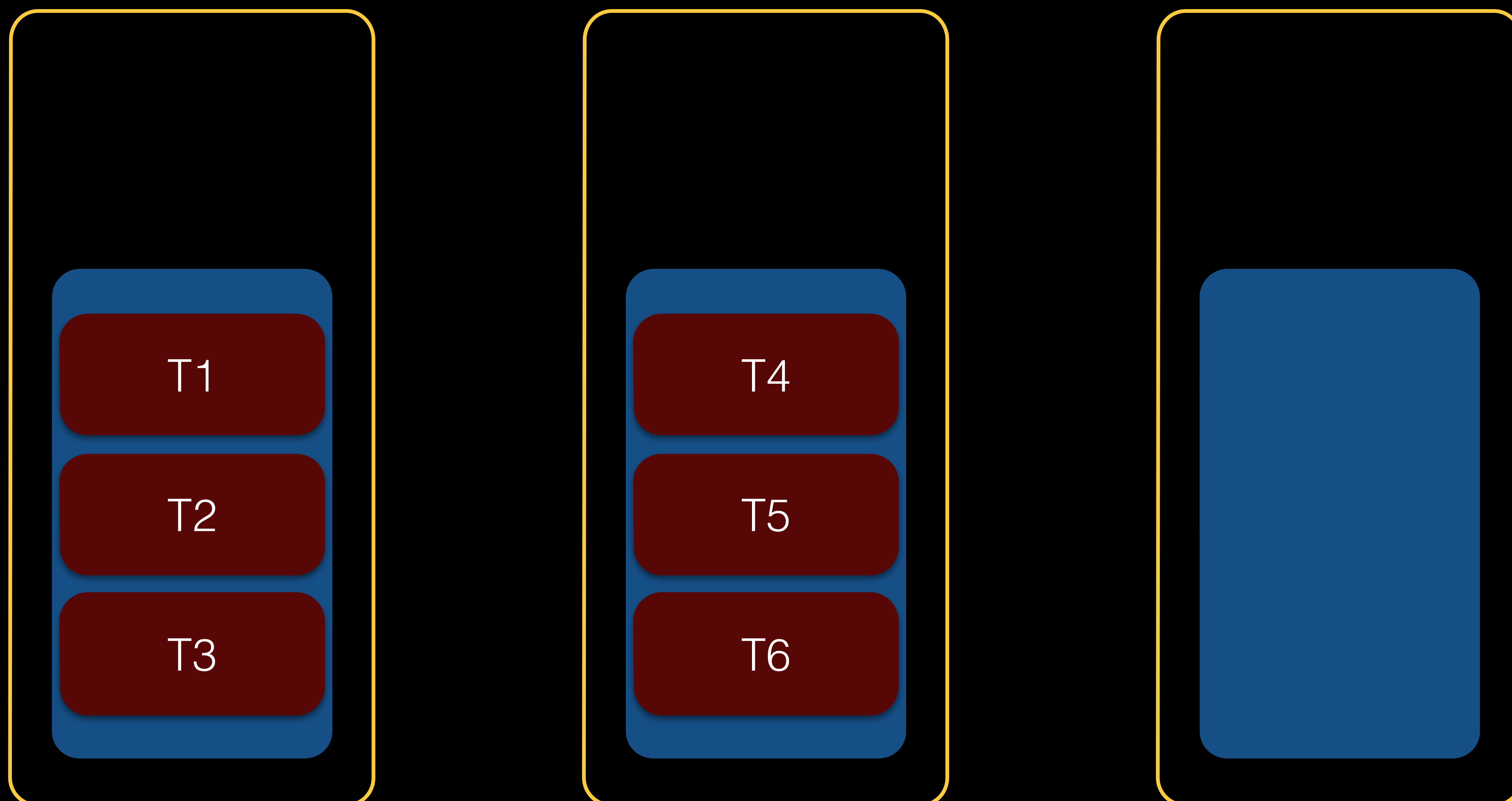




Task-Based Solution

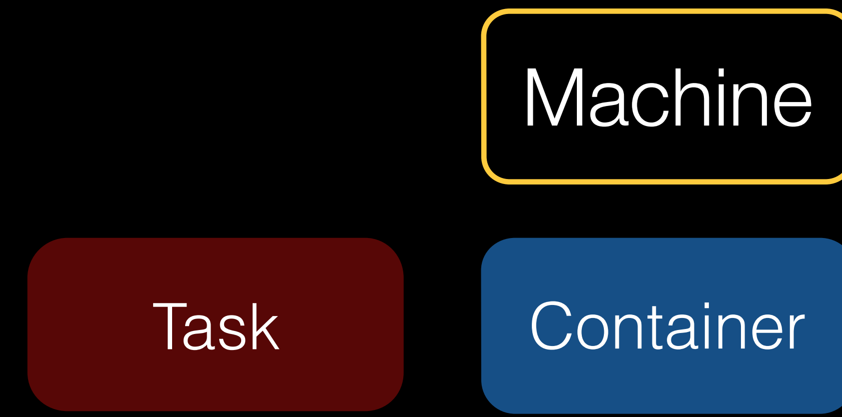
Scaling

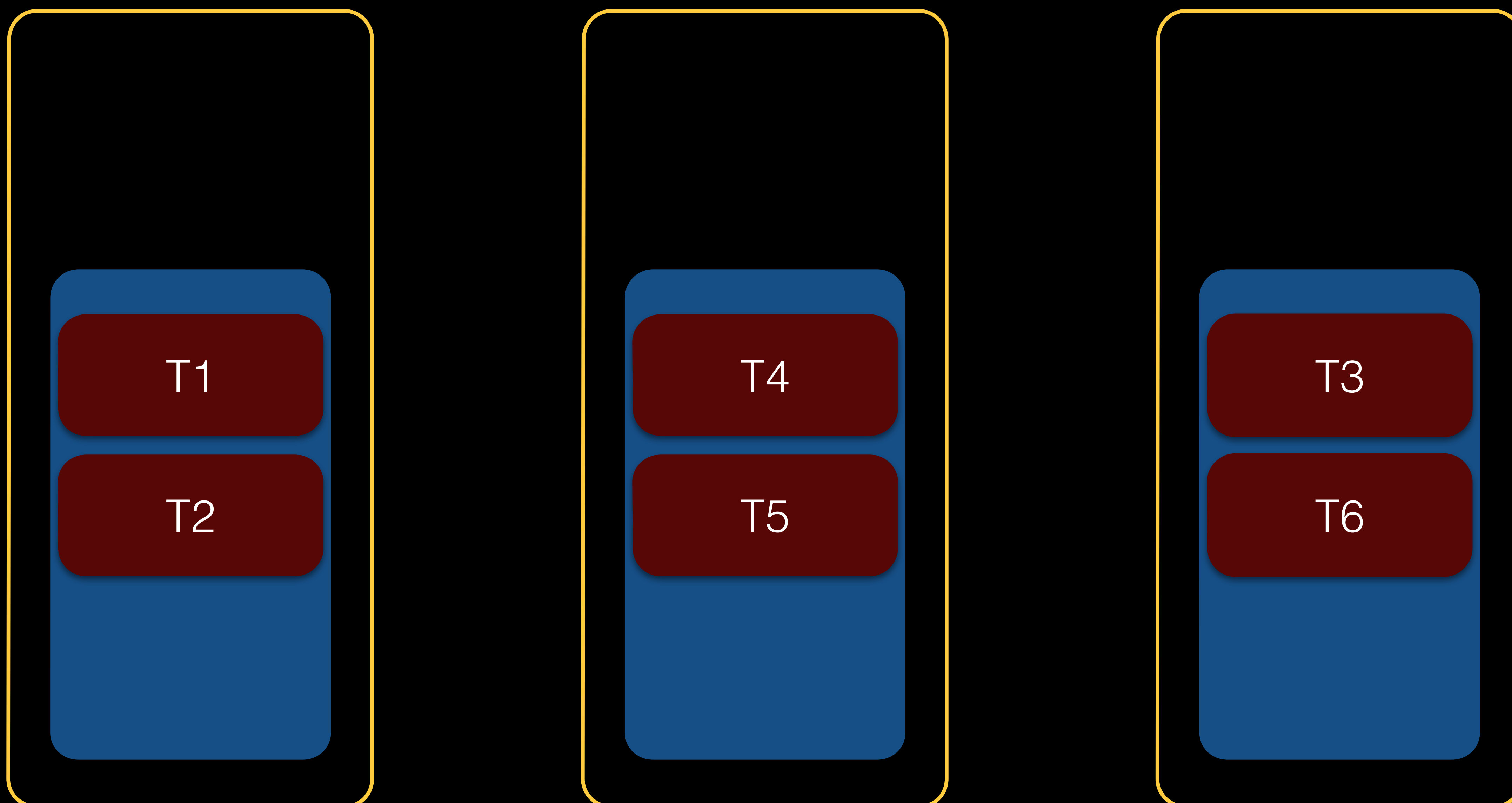




Task-Based Solution

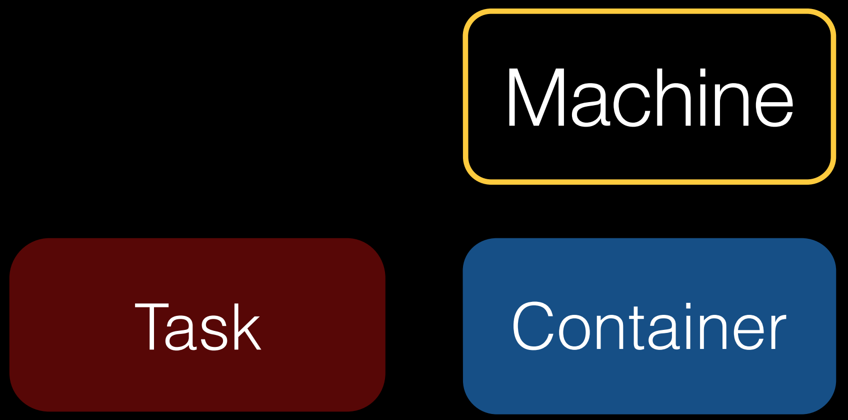
Scaling

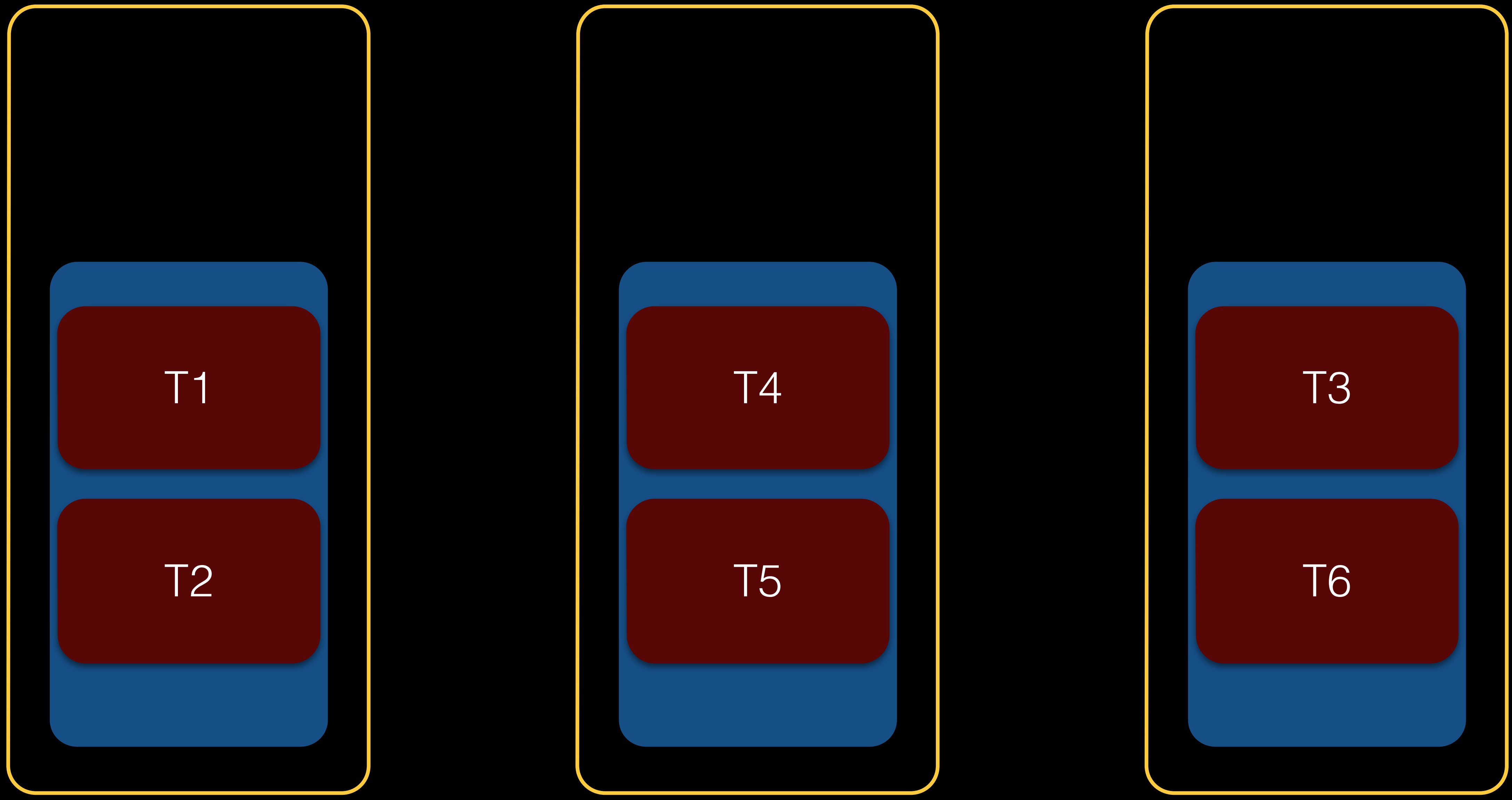




Task-Based Solution

Scaling





Comparing Solutions

	Container Solution	Task + Container Solution
Utilization	Application requirements define container size	Tasks are distributed as needed to a minimal container set as per SLA
Fault Tolerance	New container is started	Existing task can assume a new state while waiting for new container
Scaling	Workload is repartitioned and new containers are brought up	Tasks are moved across containers
Discovery	Existence	Existence and state

Comparing Solutions

Benefits of a Task-Based Solution

Container reuse

Minimize overhead of container relaunch

Fine-grained scheduling

Comparing Solutions

Benefits of a Task-Based Solution

Container reuse

Minimize overhead of container relaunch

Fine-grained scheduling

Task : Container :: Thread : Process

Task is the right level of abstraction

Comparing Solutions

We need a reactive approach to resource assignment

Working at task granularity is powerful

Comparing Solutions

We need a reactive approach to resource assignment

Working at task granularity is powerful

How can Helix help?

Comparing Solutions

We need a reactive approach to resource assignment

Working at task granularity is powerful

How can Helix help?

YARN/Mesos: containers bring flexibility in a machine

Helix: tasks bring flexibility in a container

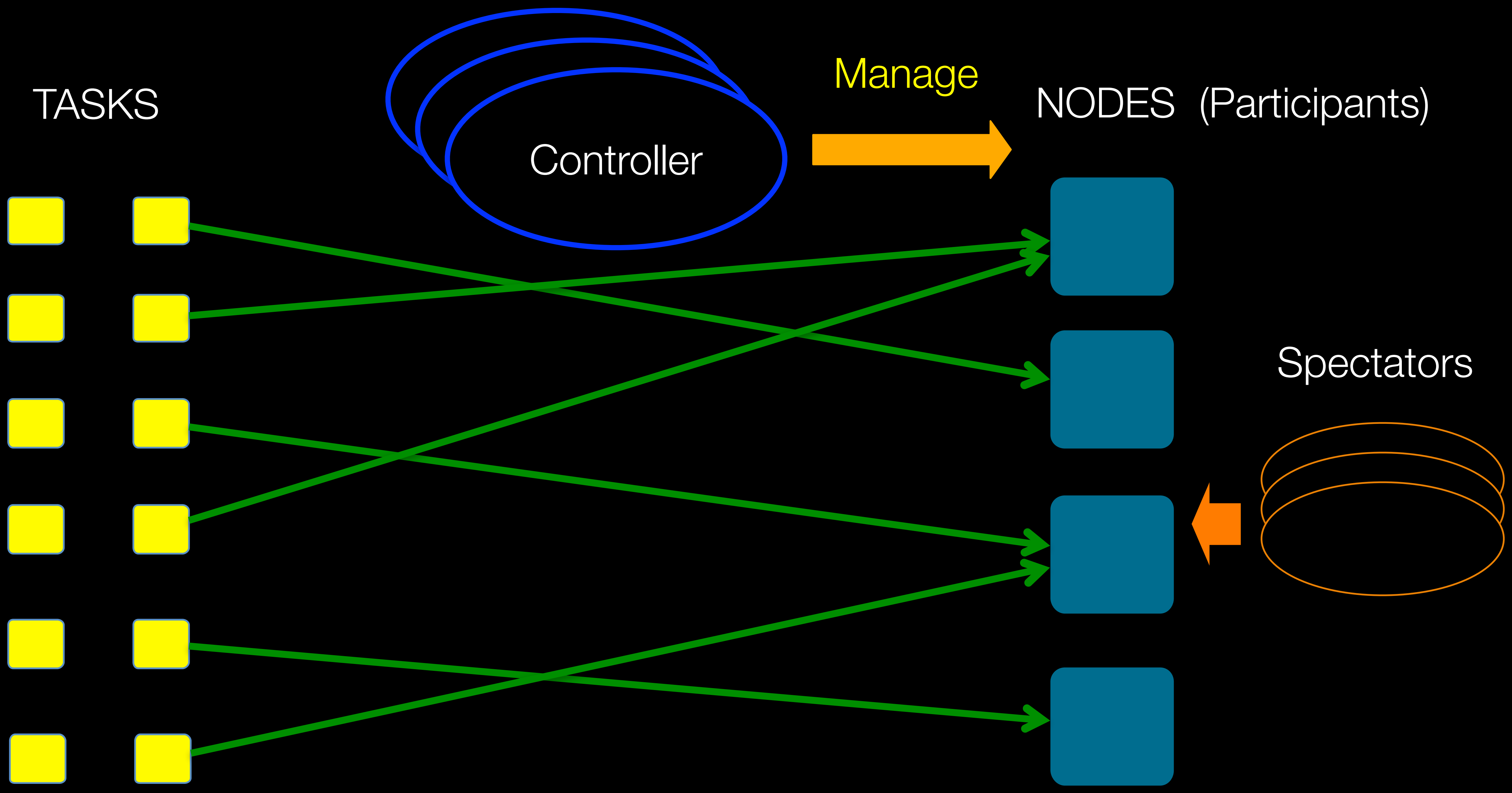
Task Management with Helix

Application Lifecycle

Capacity Planning	Allocating physical resources for your load
Provisioning	Deploying and launching tasks
Fault Tolerance	Staying available, ensuring success
State Management	Determining what code should be running and where

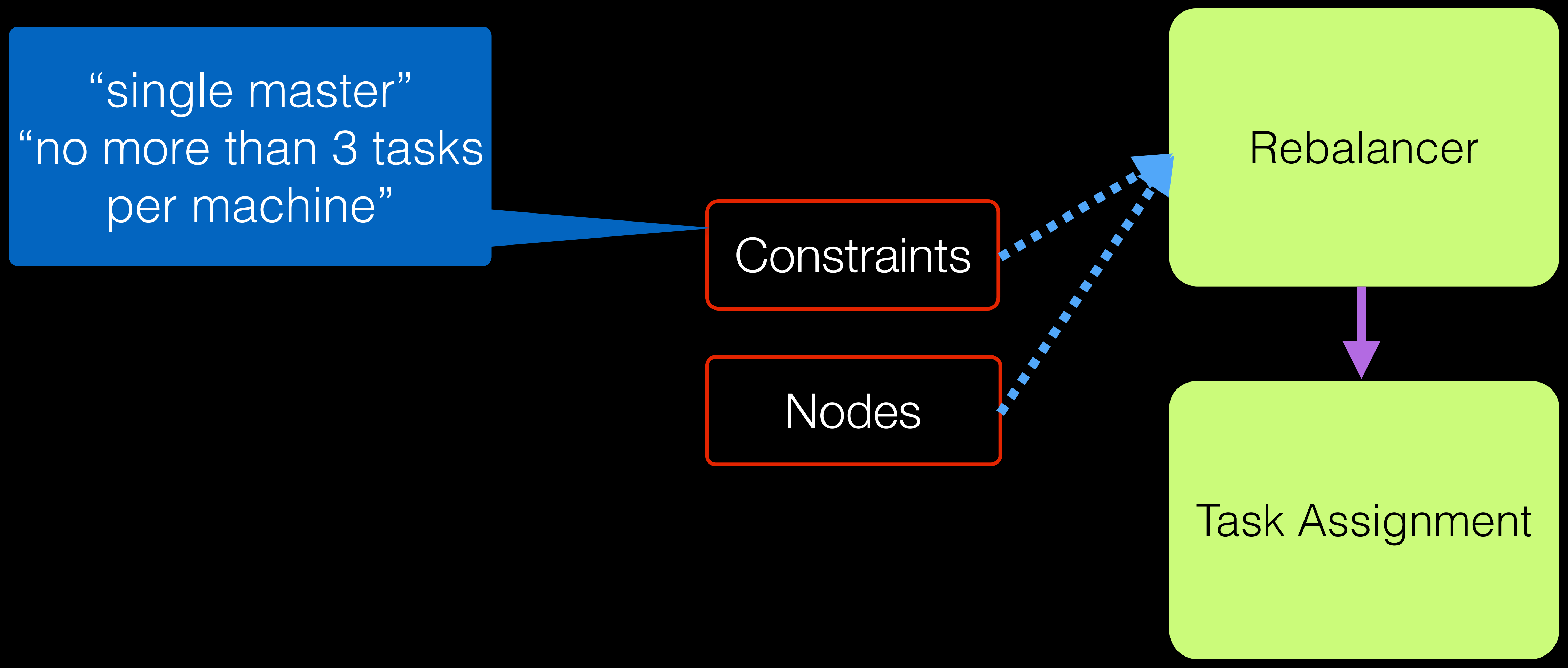
Helix Overview

Cluster Roles



Helix Controller

High-Level Overview



Helix Controller

Rebalancer

```
ResourceAssignment computeResourceMapping(
    RebalancerConfig rebalancerConfig,
    ResourceAssignment prevAssignment,
    Cluster cluster,
    ResourceCurrentState currentState);
```

Based on the current nodes in the cluster and constraints, find an assignment of task to node

Helix Controller

Rebalancer

```
ResourceAssignment computeResourceMapping(
    RebalancerConfig rebalancerConfig,
    ResourceAssignment prevAssignment,
    Cluster cluster,
    ResourceCurrentState currentState);
```

Based on the current nodes in the cluster and constraints, find an assignment of task to node

What else do we need?

Helix Controller

What is Missing?

- Dynamic Container Allocation
- Automated Service Deployment
- Container Isolation
- Resource Utilization Monitoring

Helix Controller

Target Provider

- Fixed
- CPU
- Memory
- Bin Packing

Based on some constraints, determine how many containers are required in this system

We're working on integrating with monitoring systems in order to query for usage information

Helix Controller

Target Provider

```
TargetProviderResponse evaluateExistingContainers(
    Cluster cluster,
    ResourceId resourceId,
    Collection<Participant> participants);
```

```
class TargetProviderResponse {
  List<ContainerSpec> containersToAcquire;
  List<Participant> containersToRelease;
  List<Participant> containersToStop;
  List<Participant> containersToStart;
}
```

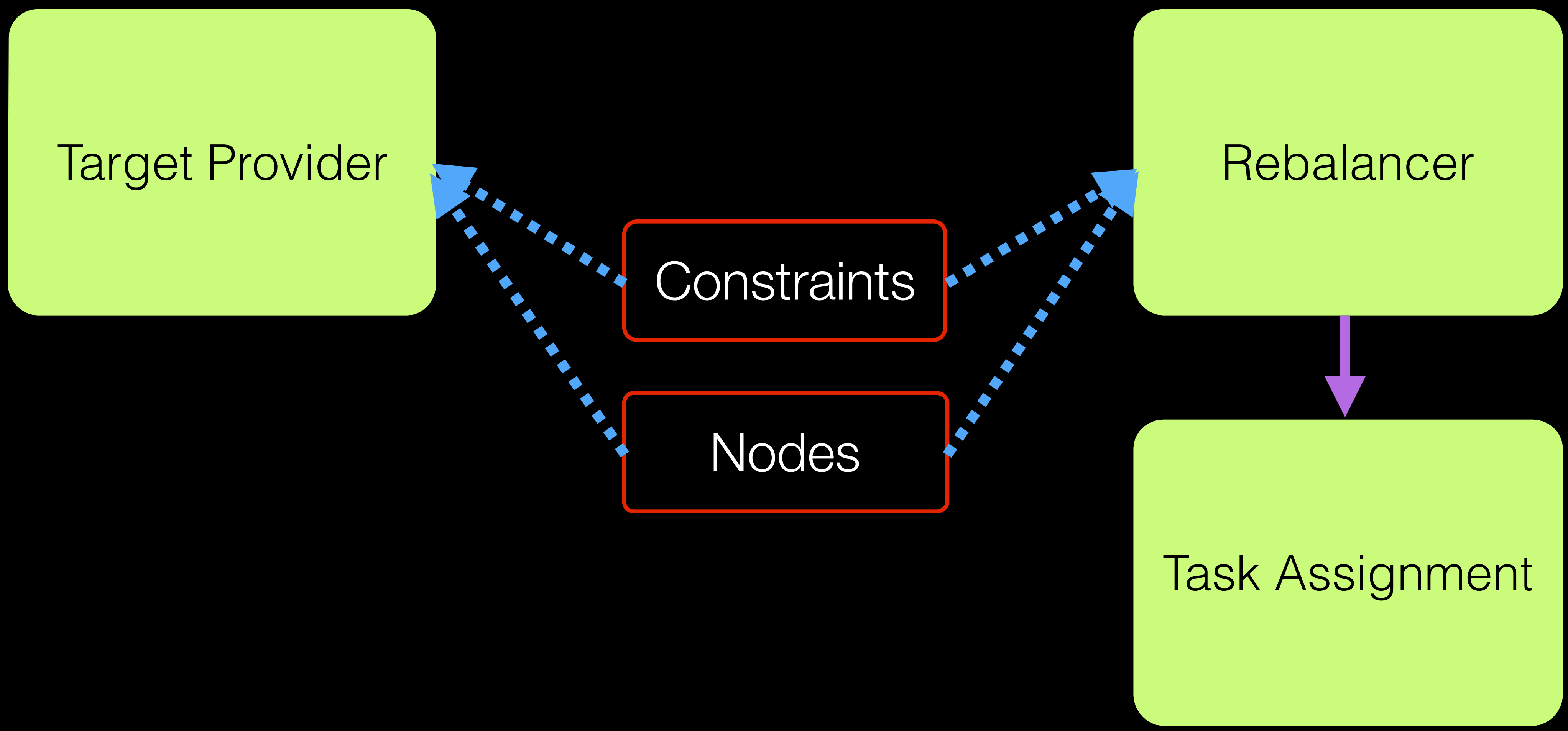
- Fixed
- CPU
- Memory
- Bin Packing

Based on some constraints, determine how many containers are required in this system

We're working on integrating with monitoring systems in order to query for usage information

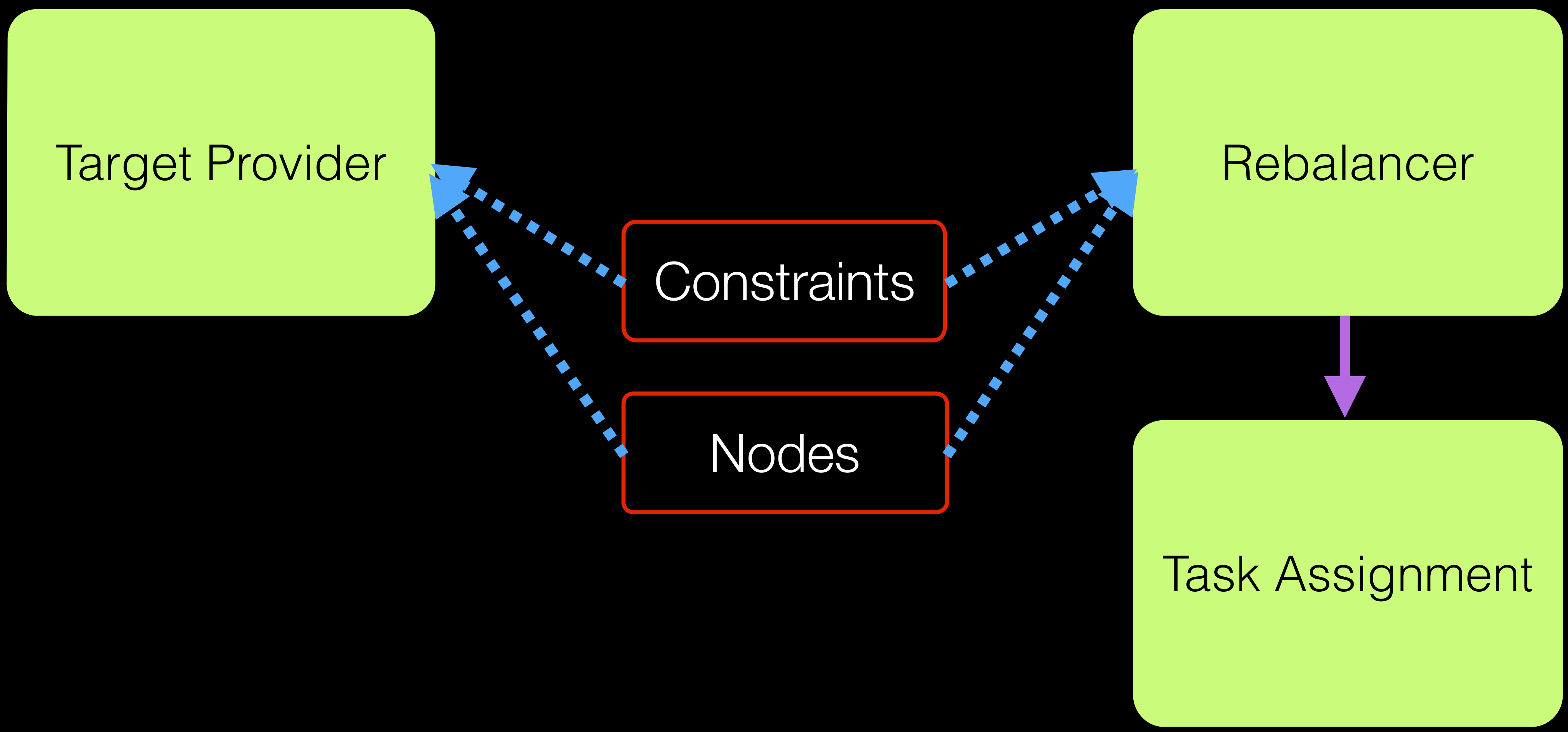
Helix Controller

Adding a Target Provider



Helix Controller

Adding a Target Provider



How do we use the target provider response?

Helix Controller

Container Provider

YARN

Mesos

Local

Given the container requirements, ensure that number of containers are running

Helix Controller

Container Provider

```
ListenableFuture<ContainerId>
allocateContainer(ContainerSpec spec);

ListenableFuture<Boolean>
deallocateContainer(ContainerId containerId);

ListenableFuture<Boolean>
startContainer(ContainerId containerId,
    Participant participant);

ListenableFuture<Boolean>
stopContainer(ContainerId containerId);
```

YARN

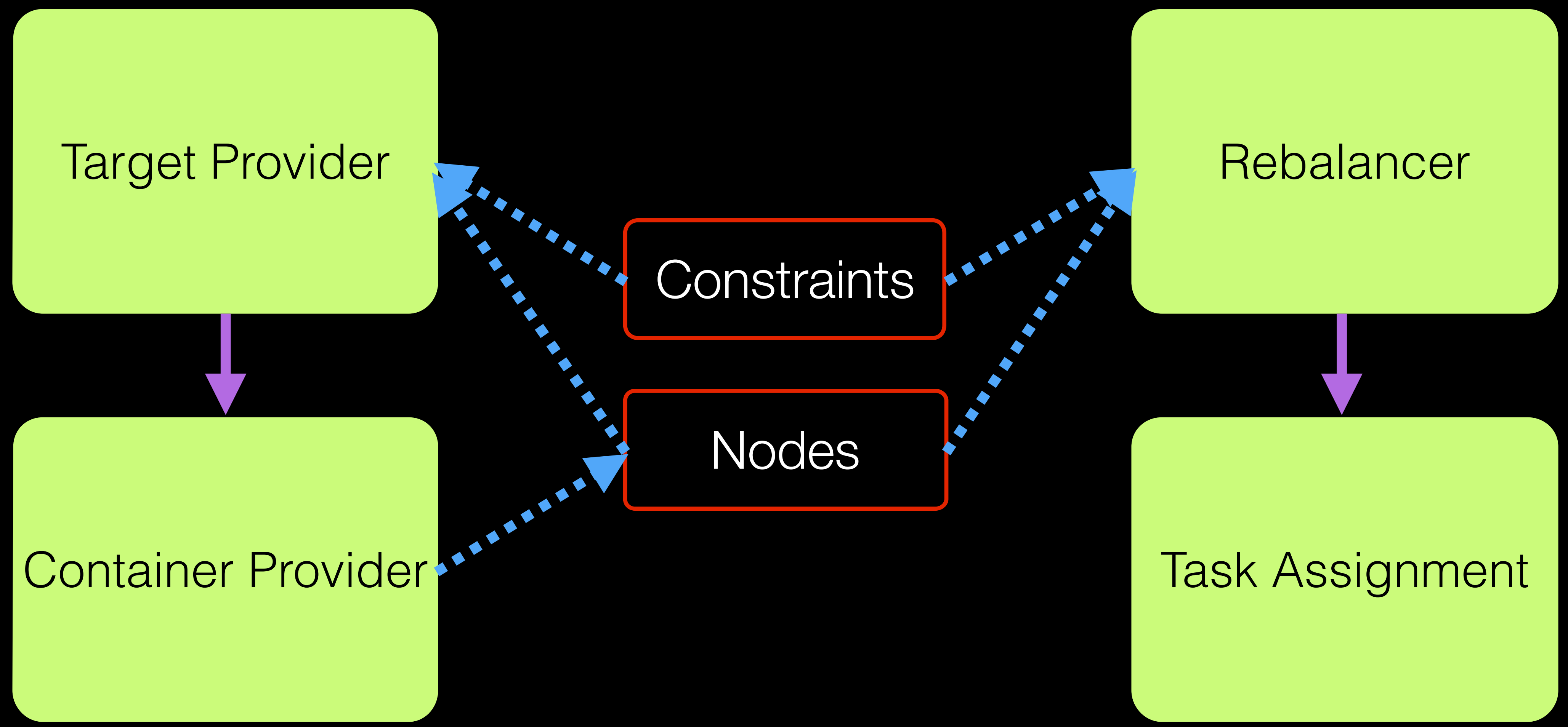
Mesos

Local

Given the container requirements, ensure that number of containers are running

Helix Controller

Adding a Container Provider



Target Provider + Container Provider = Provisioner

Application Lifecycle

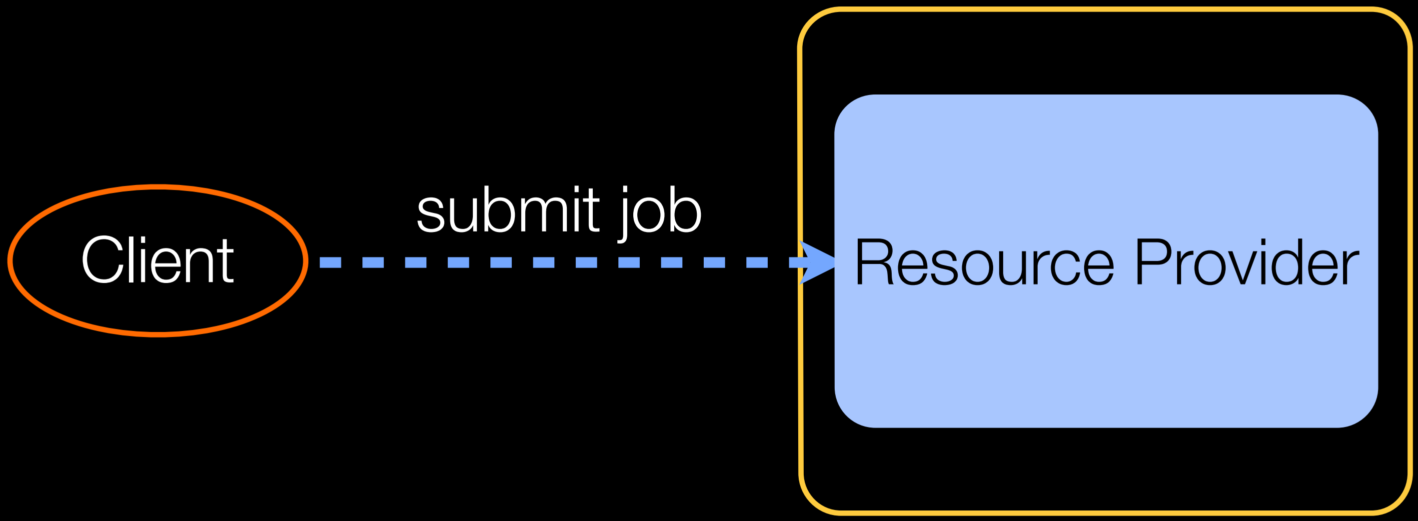
With Helix and the Task Abstraction

Capacity Planning	Target Provider
Provisioning	Container Provider
Fault Tolerance	Existing Helix Controller (enhanced by Provisioner)
State Management	Existing Helix Controller (enhanced by Provisioner)

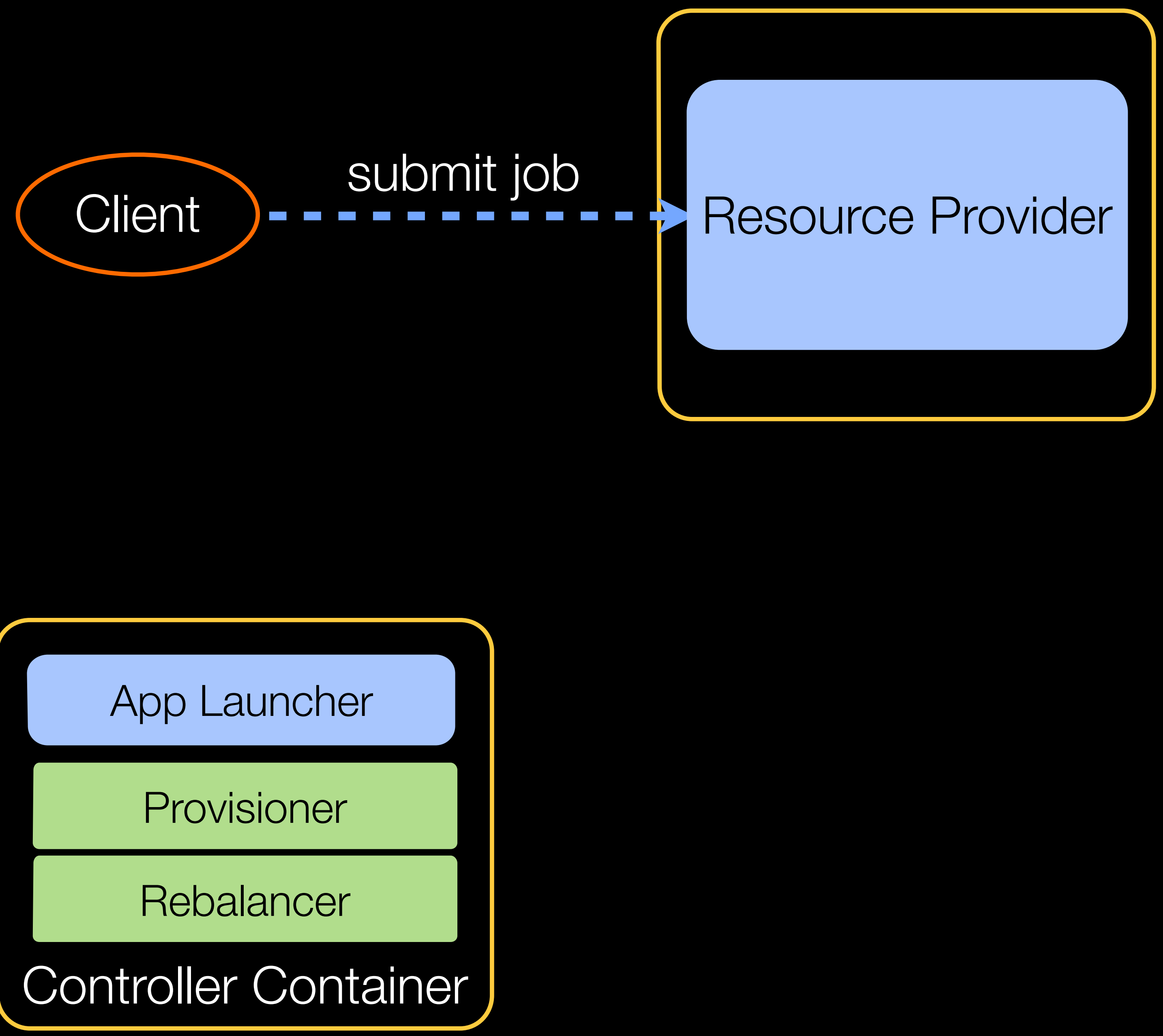
System Architecture

System Architecture

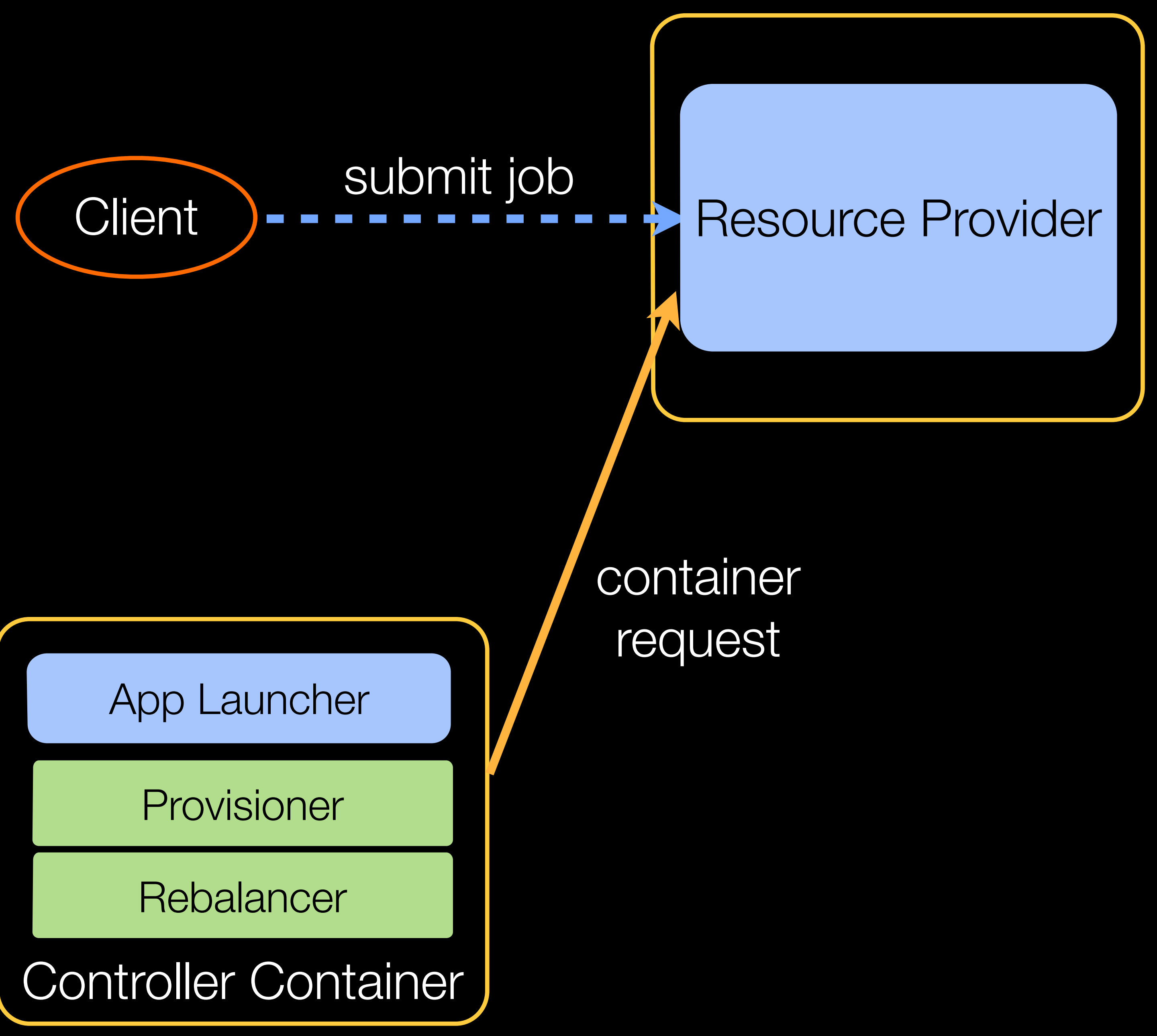
System Architecture



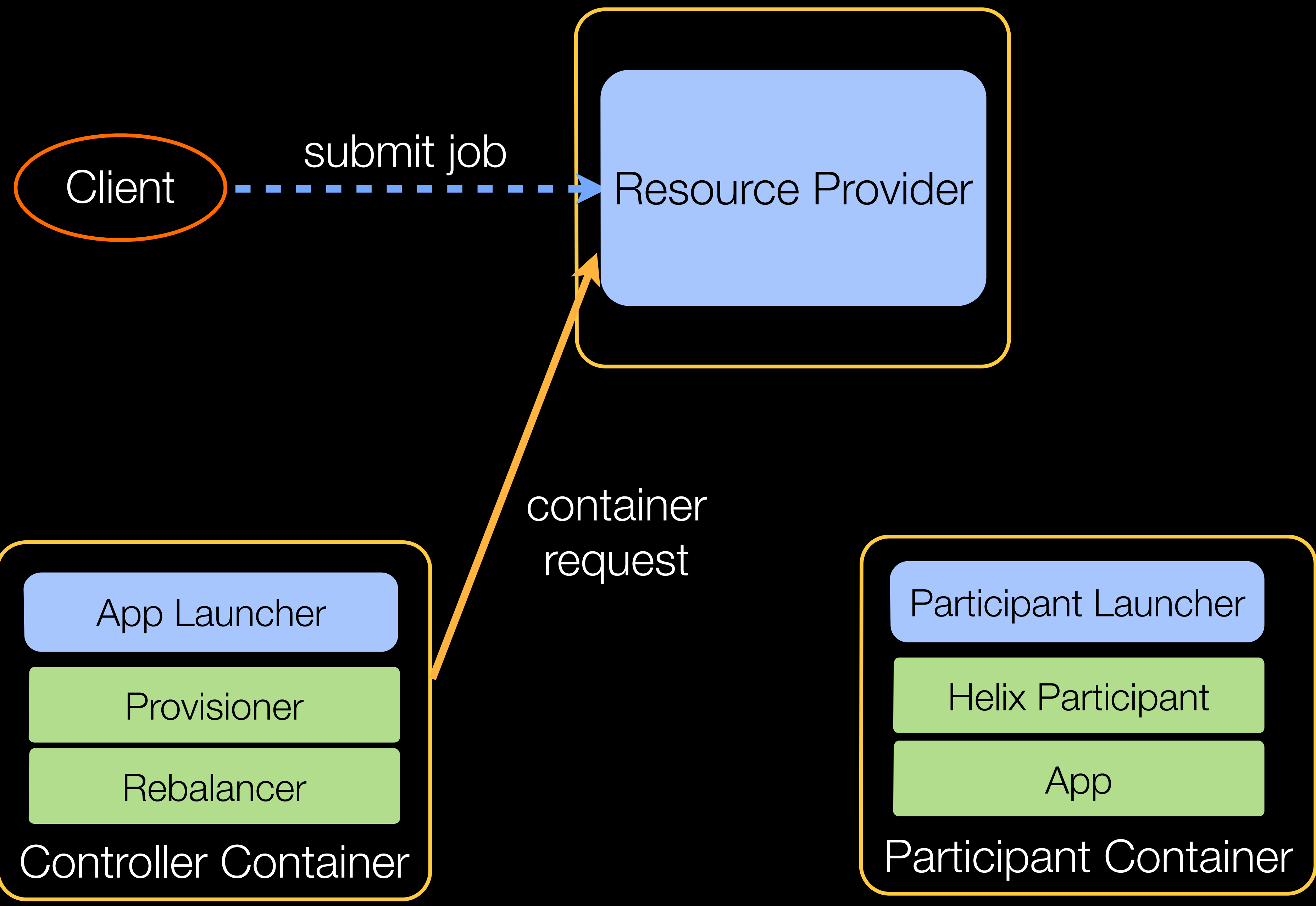
System Architecture



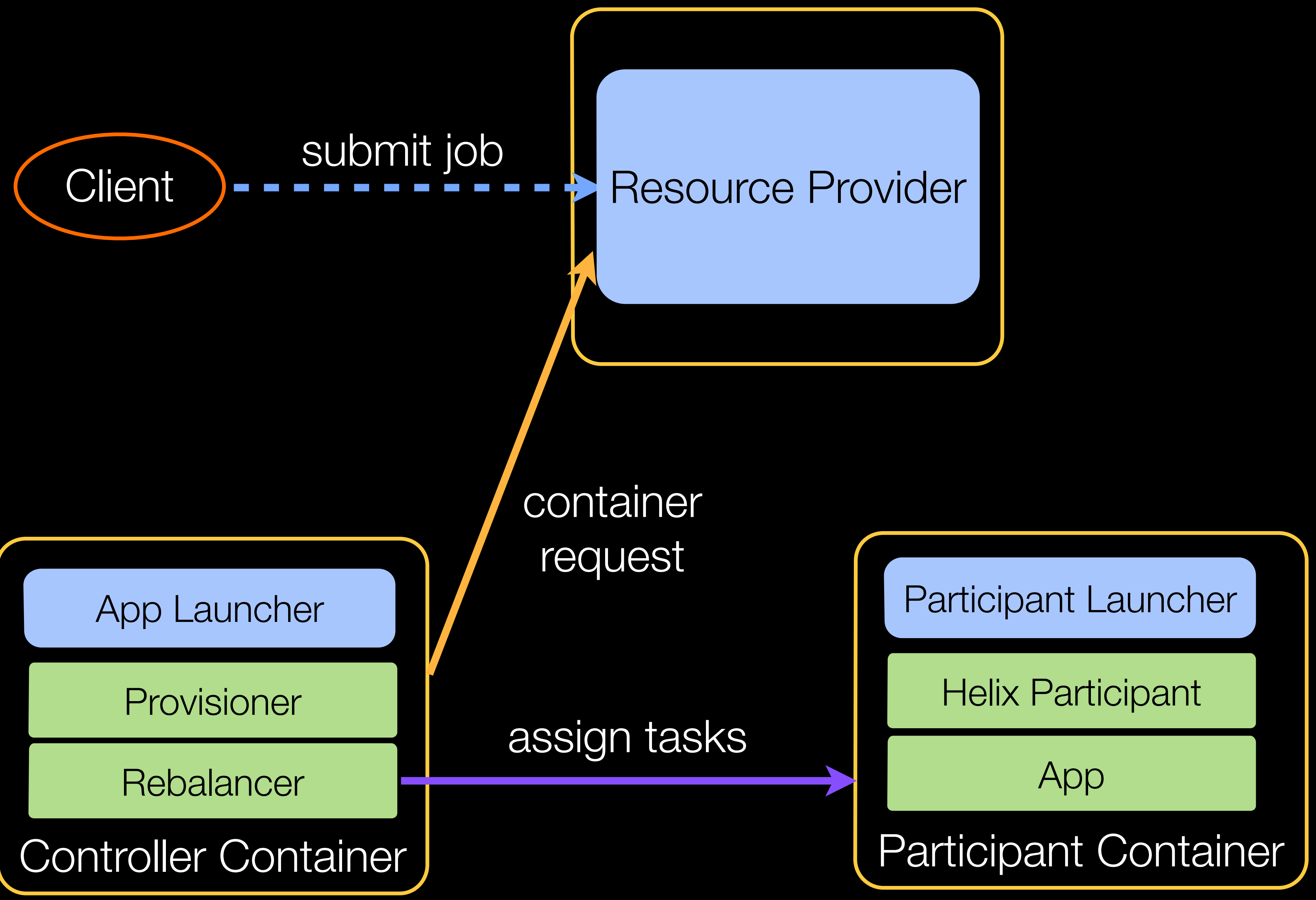
System Architecture



System Architecture

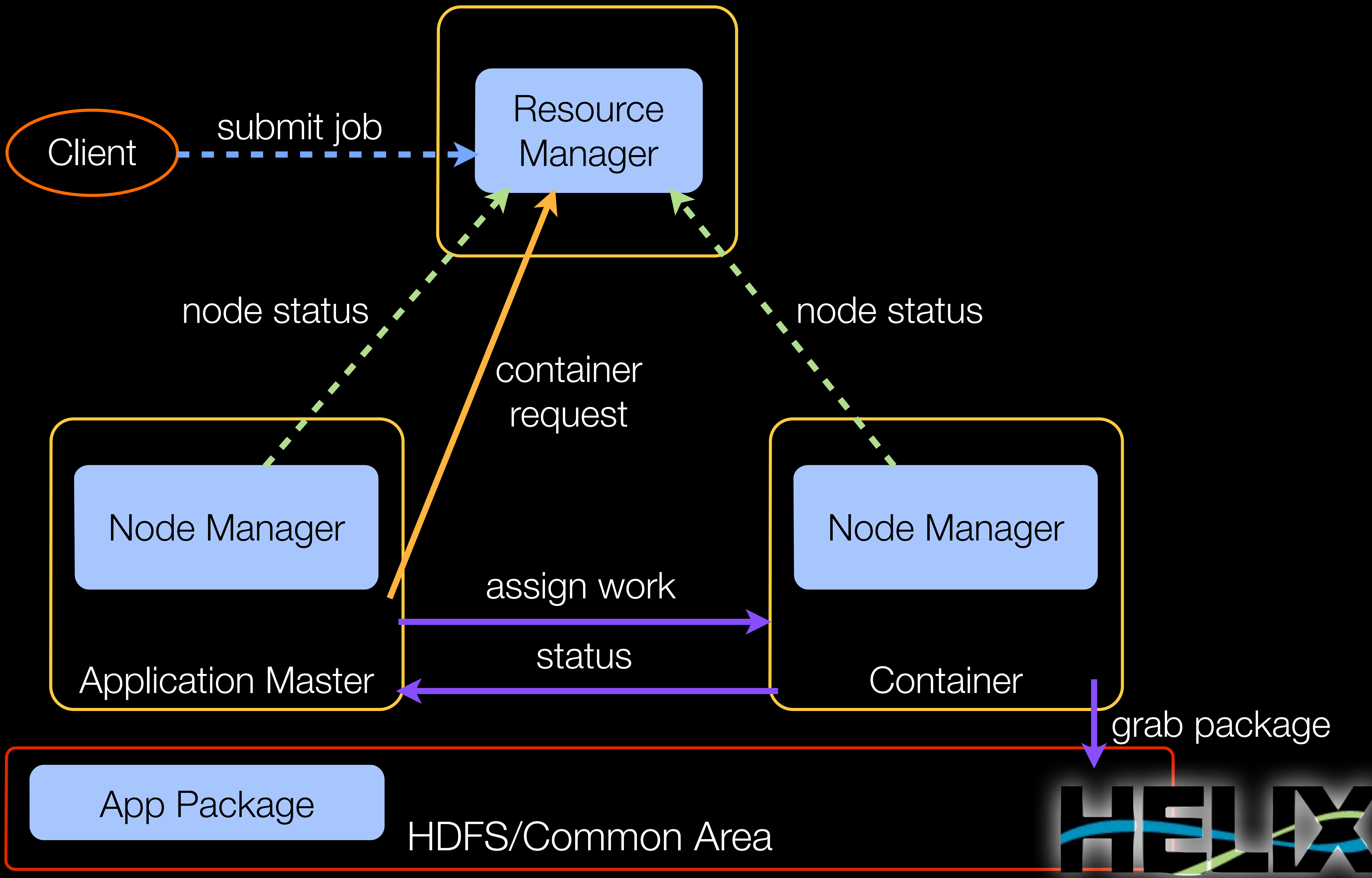


System Architecture



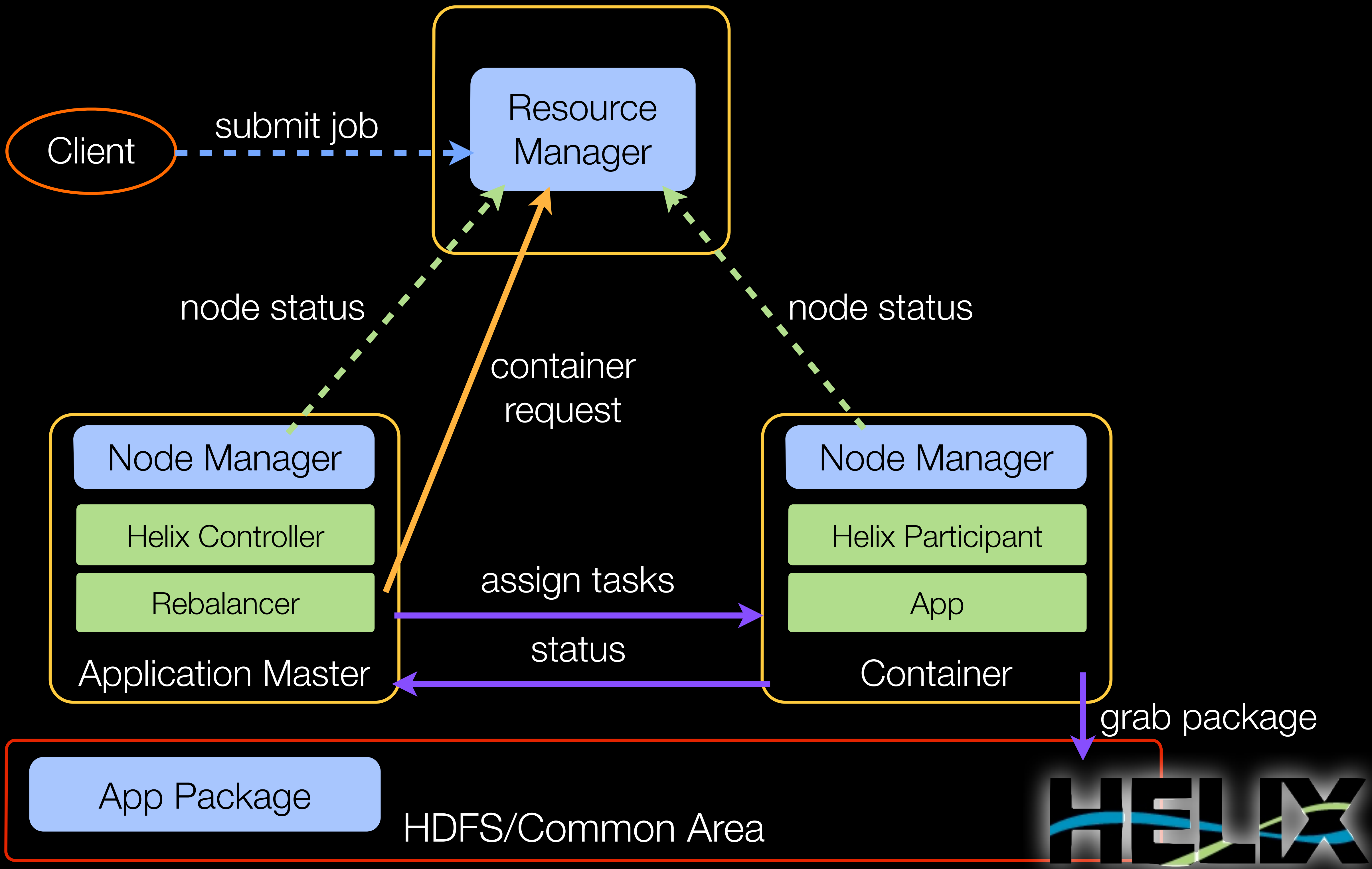
Helix + YARN

YARN Architecture



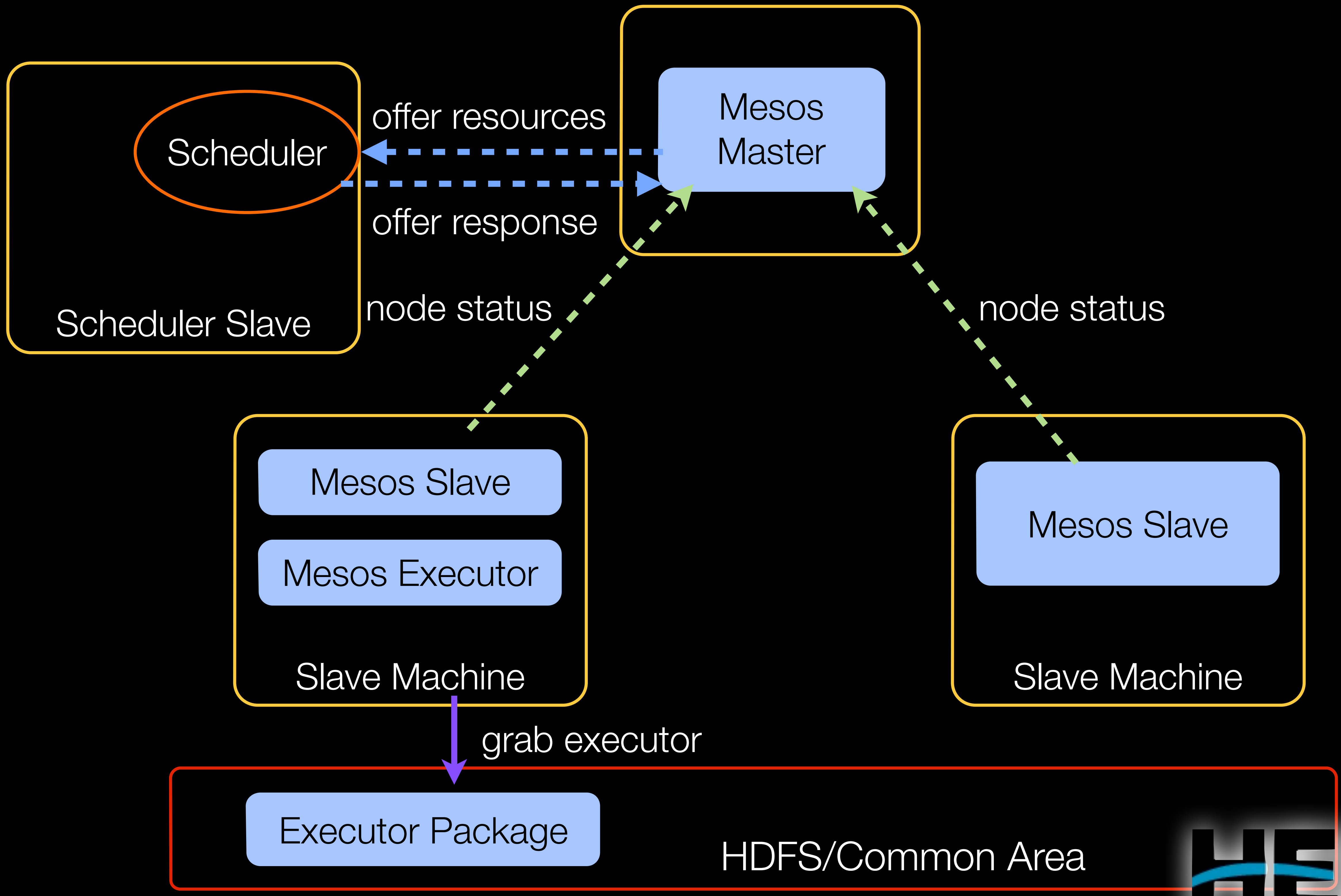
Helix + YARN

Helix + YARN Architecture



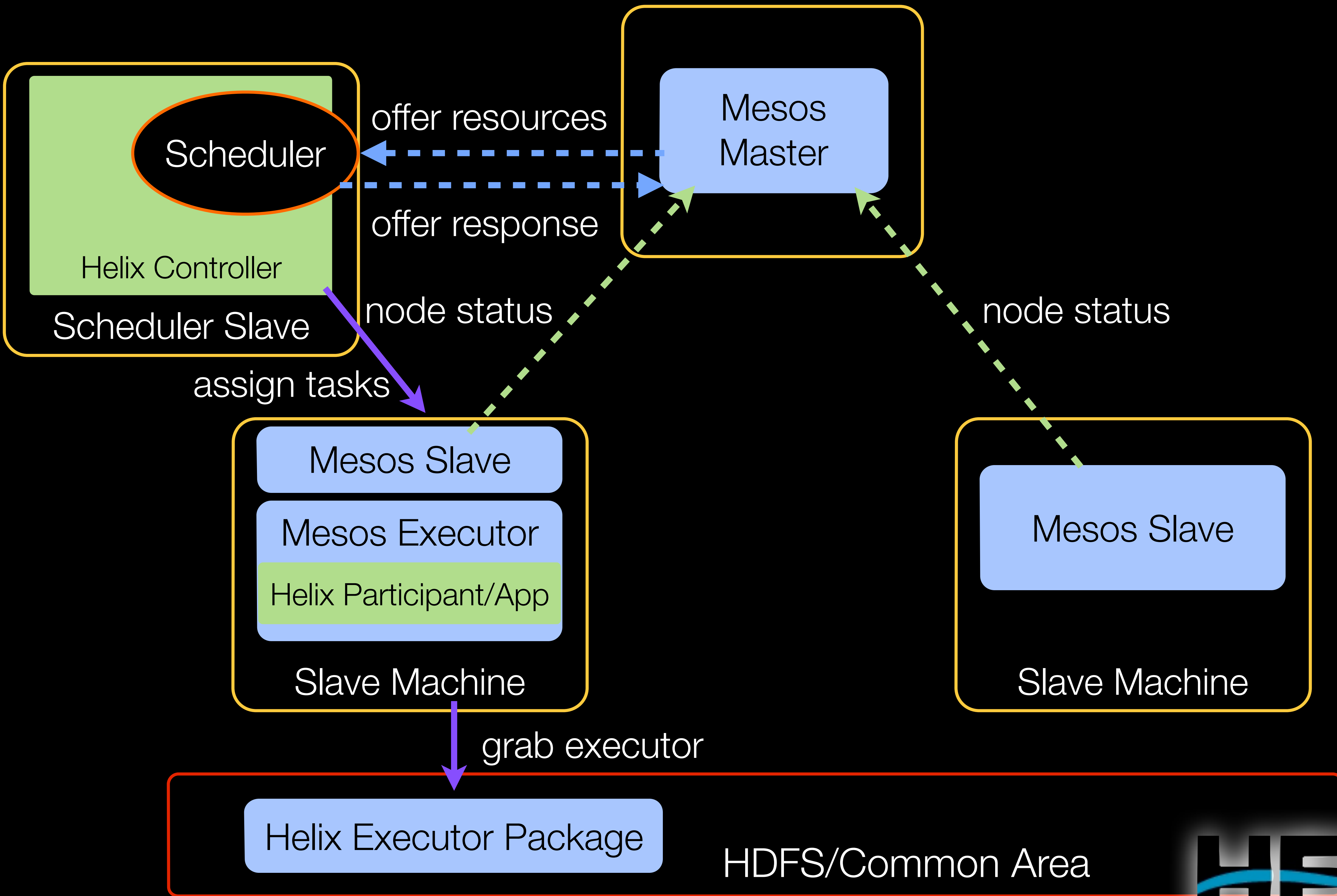
Helix + Mesos

Mesos Architecture



Helix + Mesos

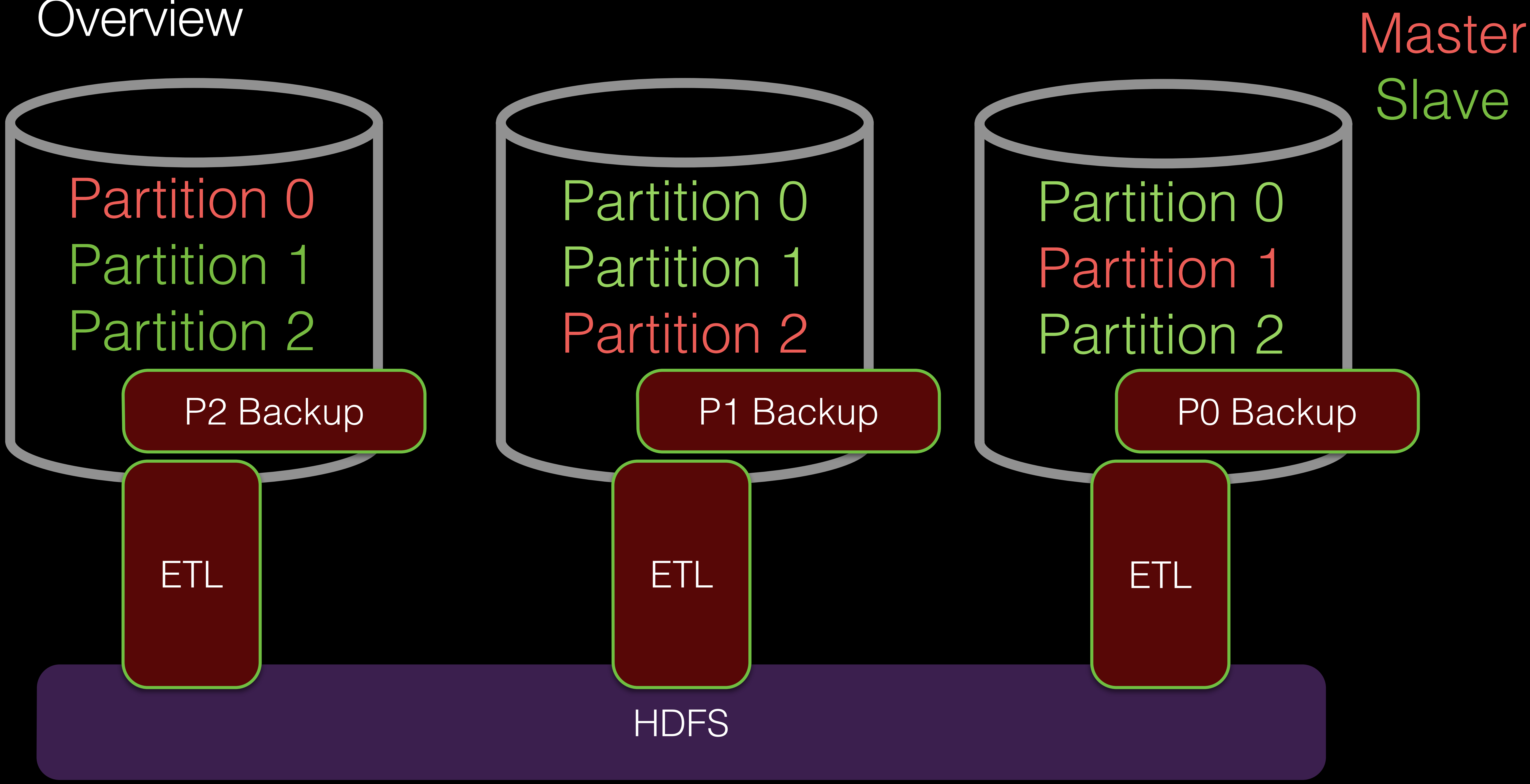
Helix + Mesos Architecture



Example

Distributed Document Store

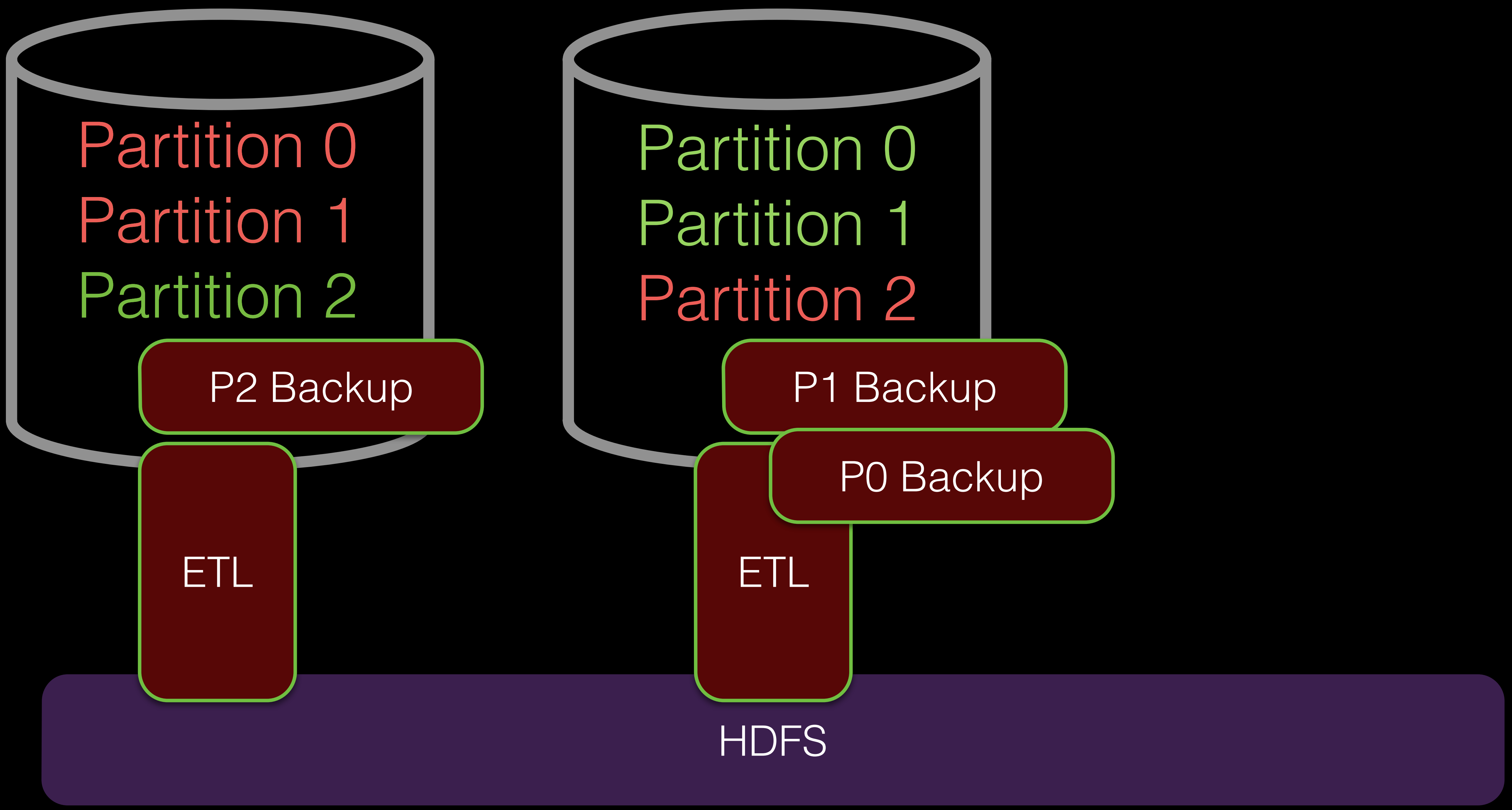
Overview



Distributed Document Store

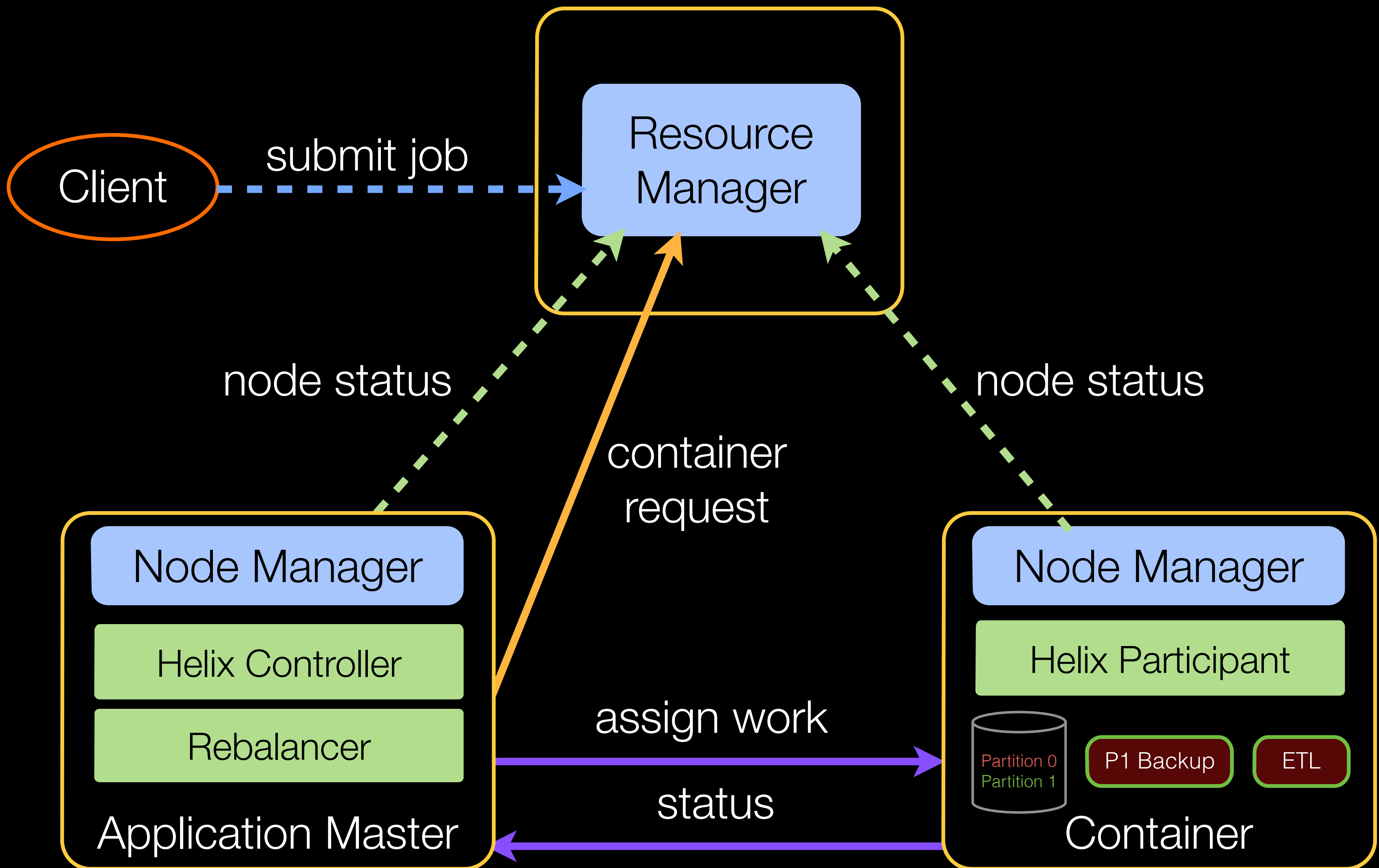
Overview

Master
Slave



Distributed Document Store

YARN Example



Distributed Document Store

YAML Specification

```yaml
appConfig: { config: { k1: v1 } }
appPackageUri: 'file://path/to/myApp-pkg.tar'
appName: myApp
services: [DB, ETL] # the task containers
serviceConfigMap:
  {DB: { num_containers: 3, memory: 1024 }, ...
   ETL: { time_to_complete: 5h, ... }, ...}
servicePackageURIMap: {
  DB: ‘file://path/to/db-service-pkg.tar', ...
}
...
```

HELIX

Distributed Document Store

YAML Specification

```
appConfig: { config: { k1: v1 } }
appPackageUri: 'file://path/to/myApp-pkg.tar'
appName: myApp
services: [DB, ETL] # the task containers
serviceConfigMap:
  {DB: { num_containers: 3, memory: 1024 }, ...
   ETL: { time_to_complete: 5h, ... }, ...}
servicePackageURIMap: {
  DB: ‘file://path/to/db-service-pkg.tar', ...
}
...
```

TargetProvider specification

Distributed Document Store

Service/Container Implementation

```
public class MyQueuerService
     extends StatelessParticipantService {
   @Override
   public void init() { ... }

   @Override
   public void onOnline() { ... }

   @Override
   public void onOffline() { ... }
}
```

Distributed Document Store

Task Implementation

```
public class BackupTask extends Task {
  @Override
  public ListenableFuture<Status> start() { ... }

  @Override
  public ListenableFuture<Status> cancel() { ... }

  @Override
  public ListenableFuture<Status> pause() { ... }

  @Override
  public ListenableFuture<Status> resume() { ... }
}
```

Distributed Document Store

State Model-Style Callbacks

```
public class StoreStateModel extends StateModel {
  public void onBecomeMasterFromSlave() { ... }

  public void onBecomeSlaveFromMaster() { ... }

  public void onBecomeSlaveFromOffline() { ... }

  public void onBecomeOfflineFromSlave() { ... }
}
```

Distributed Document Store

Spectator (for Discovery)

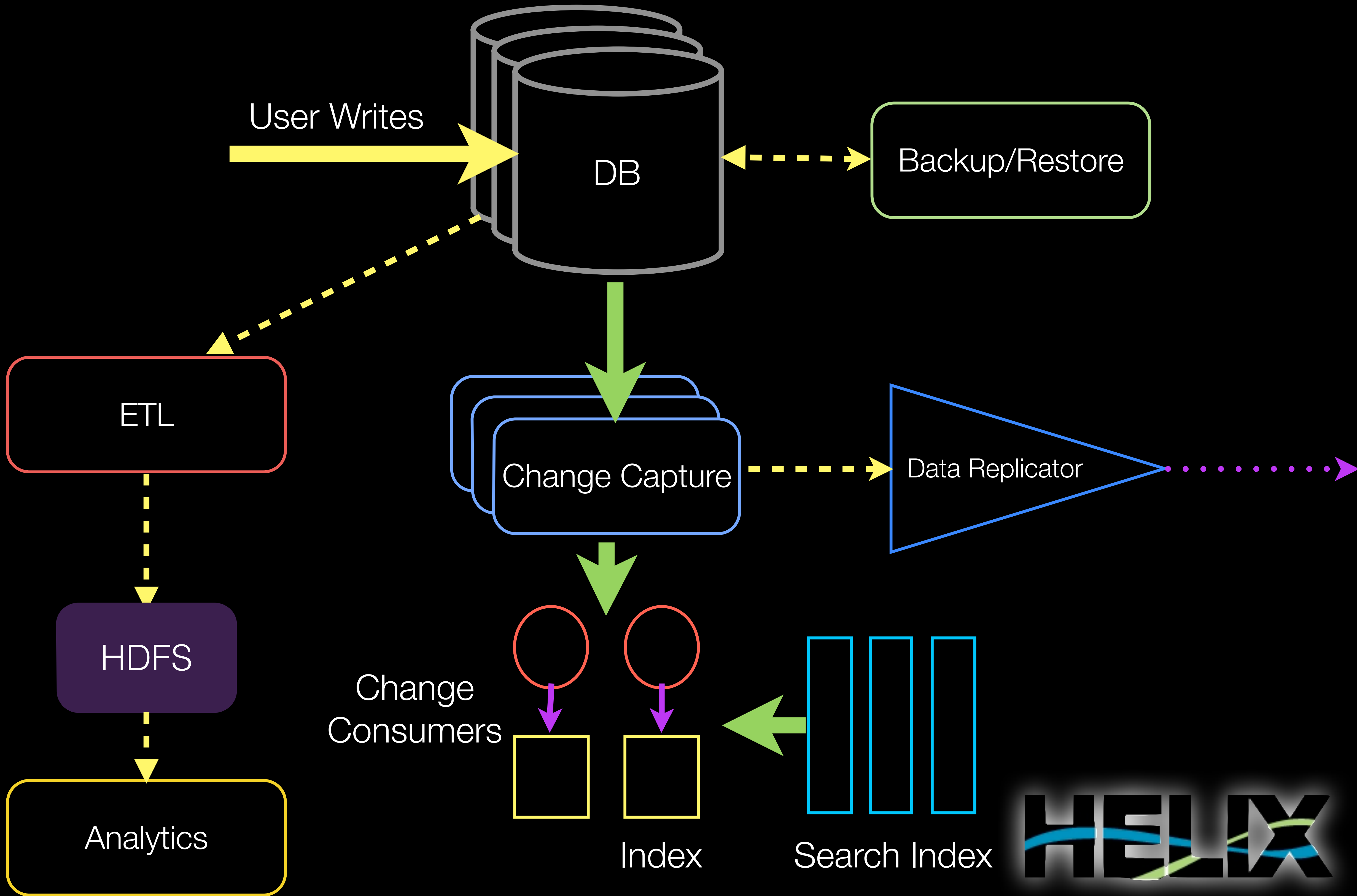
```
class RoutingLogic {
   public void write(Request request) {
     partition = getPartition(request.key);
     List<Participant> nodes =
         routingTableProvider.getInstance(
             partition, “MASTER”);
     nodes.get(0).write(request);
   }

   public void read(Request request) {
     partition = getPartition(request.key);
     List<Participant> nodes =
         routingTableProvider.getInstance(partition);
     random(nodes).read(request);
   }
}
```

Helix at LinkedIn

Helix at LinkedIn

In Production



Helix at LinkedIn

In Production

Over 1000 instances covering over 30000 database partitions

Over 1000 instances for change capture consumers

As many as 500 instances in a single Helix cluster

(all numbers are per-datacenter)

Summary

- Container abstraction has become a huge win
- With Helix, we can go a step further and make tasks the unit of work
- With the TargetProvider and ContainerProvider abstractions, any popular provisioner can be plugged in

Questions?

?

Jason	zzhang@apache.org
Kanak	kanak@apache.org
Website	helix.apache.org
Dev Mailing List	dev@helix.apache.org
User Mailing List	user@helix.apache.org
Twitter	@apachehelix