Nomad

HASHICORP

Armon Dadgar

@armon

HASHICORP

Nomad

Distributed

Optimistically Concurrent

Scheduler

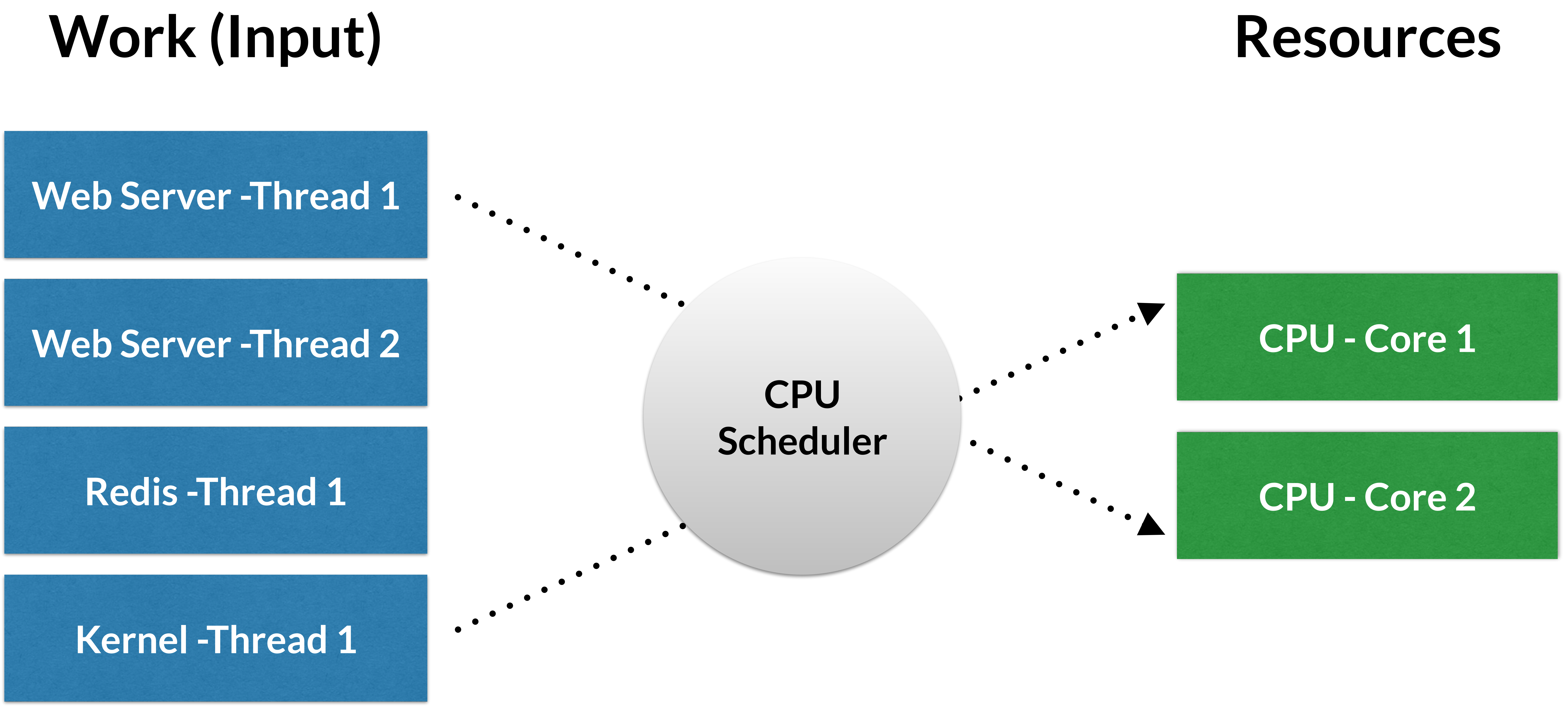
Nomad

Distributed

Optimistically Concurrent

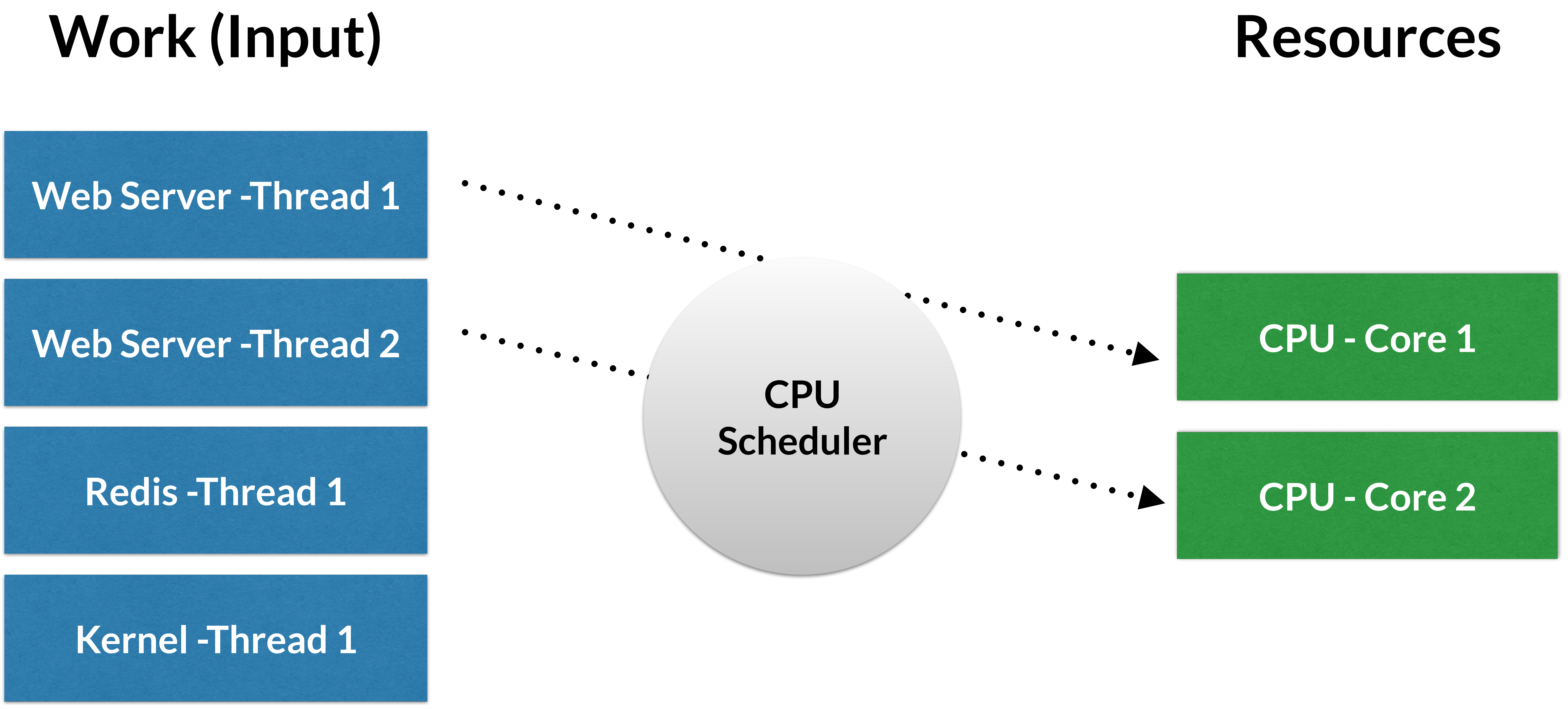
Scheduler

Schedulers map a set of **work** to a set of **resources**



HASHICORP

CPU Scheduler



HASHICORP

CPU Scheduler

Type	Work	Resources
CPU Scheduler	Threads	Physical Cores
AWS EC2 / OpenStack Nova	Virtual Machines	Hypervisors
Hadoop YARN	MapReduce Jobs	Client Nodes
Cluster Scheduler	Applications	Servers

HASHICORP

Schedulers In the Wild

Higher Resource Utilization

Decouple Work from Resources

Better Quality of Service

HASHICORP

Advantages

Higher Resource Utilization

Decouple Work from Resources

Better Quality of Service

Bin Packing

Over-Subscription

Job Queueing

HASHICORP

Advantages

Higher Resource Utilization

Decouple Work from Resources

Better Quality of Service

Abstraction

API Contracts

Standardization

HASHICORP

Advantages

Higher Resource Utilization

Decouple Work from Resources

Better Quality of Service

Priorities

Resource Isolation

Pre-emption

HASHICORP

Advantages

Google

amazon web services™

NETFLIX

Nomad

HASHICORP

Nomad

Cluster Scheduler

Easily Deploy Applications

Job Specification

example.nomad

```
# Define our simple redis job
job "redis" {

  # Run only in us-east-1
  datacenters = ["us-east-1"]

  # Define the single redis task using Docker
  task "redis" {
    driver = "docker"

    config {
      image = "redis:latest"
    }

    resources {
      cpu = 500 # Mhz
      memory = 256 # MB
      network {
        mbits = 10
        dynamic_ports = ["redis"]
      }
    }
  }
}
```

Declares what to run

HASHICORP

Job Specification

Nomad determines where and manages how to run

Abstract work from resources

HASHICORP

Job Specification

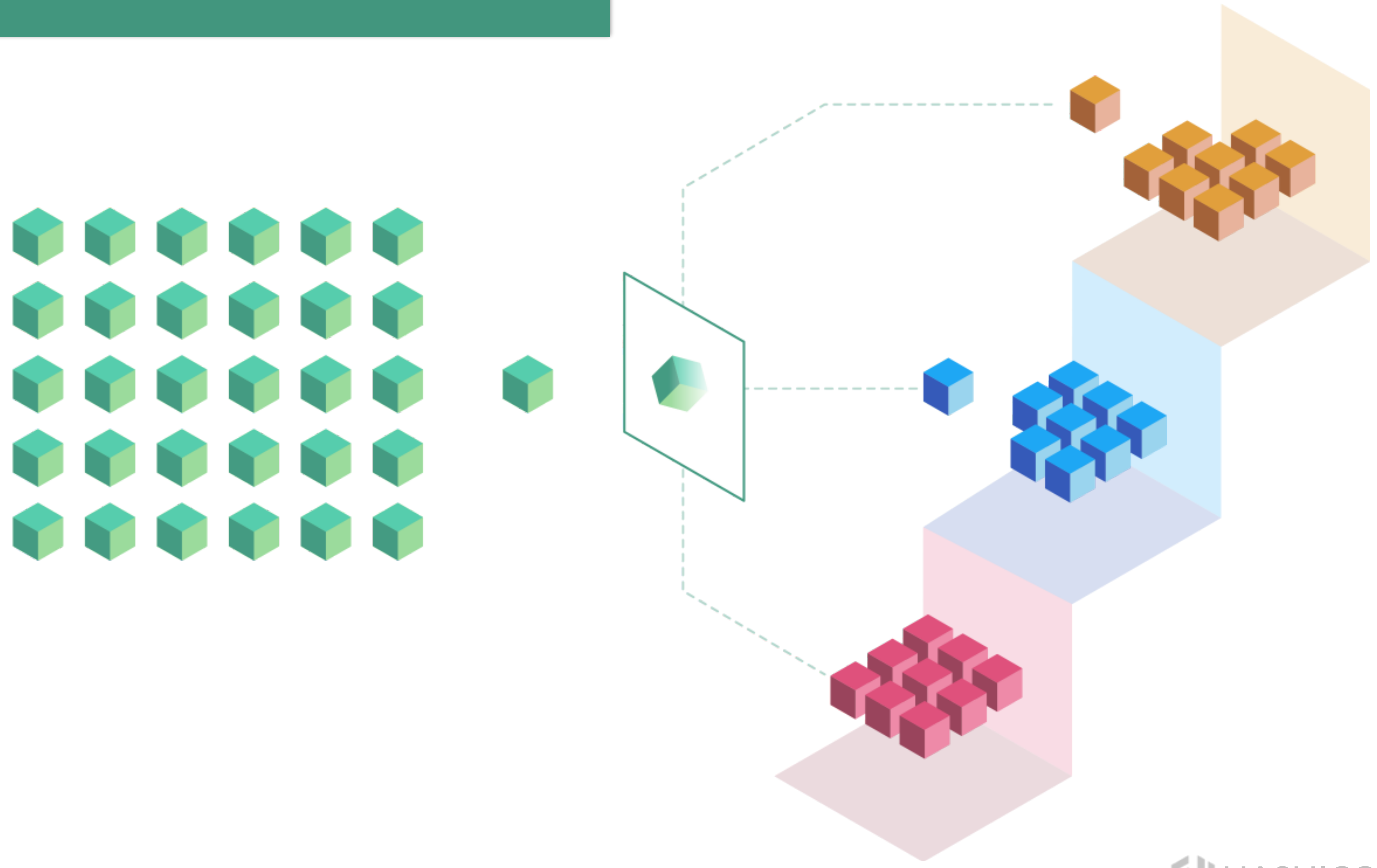
Higher Resource Utilization

Decouple Work from Resources

Better Quality of Service

Nomad

Designing Nomad



Nomad

Multi-Datacenter

Multi-Region

Flexible Workloads

Job Priorities

Bin Packing

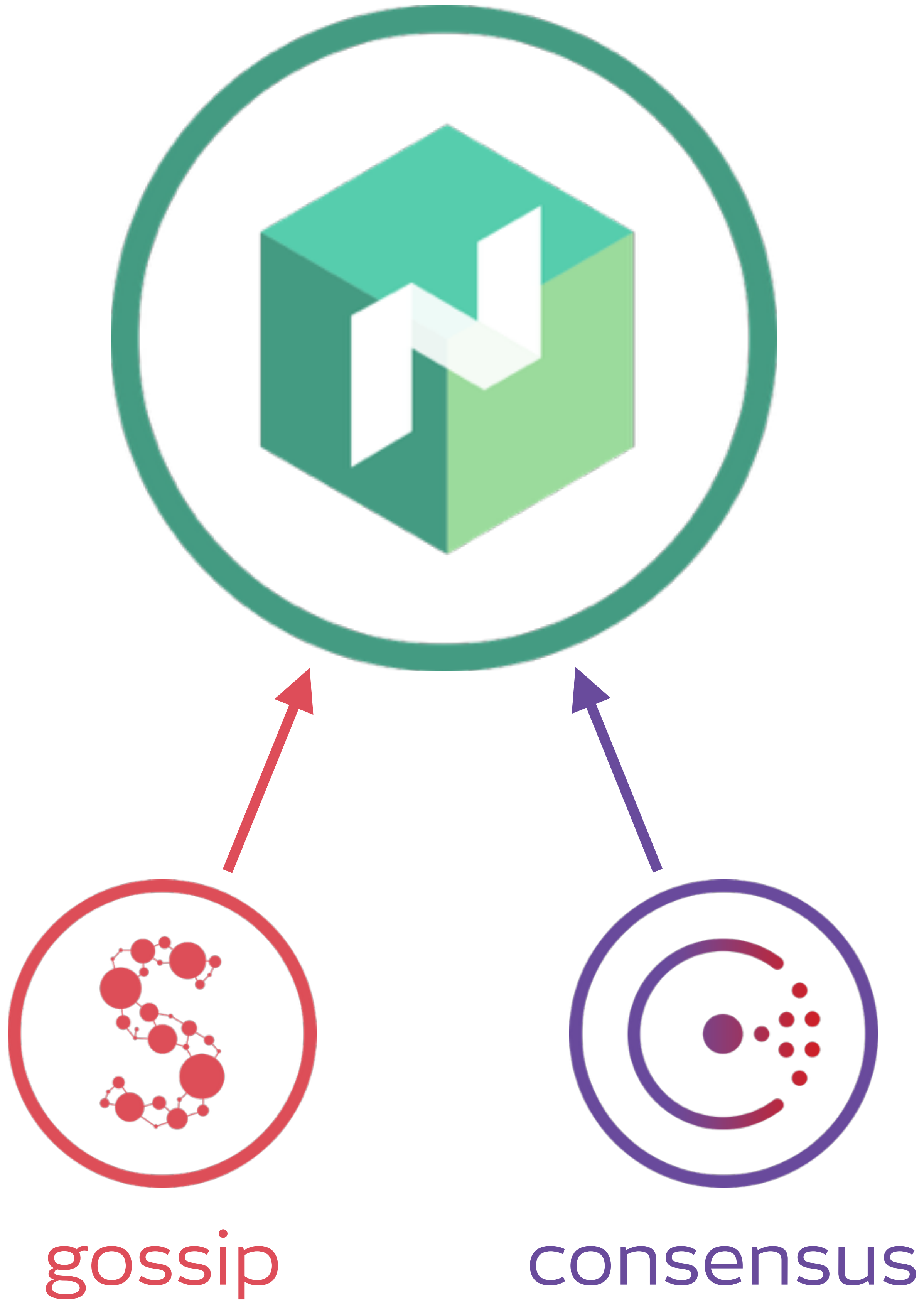
Large Scale

Operationally Simple

Thousands of regions

Tens of thousands of clients per region

Thousands of jobs per region



HASHICORP

Built on Experience

Cluster Management

Gossip Based (P2P)

Membership

Failure Detection

Event System

Gossip Protocol

Large Scale

Production Hardened

Operationally Simple

Service Discovery

Configuration

Coordination (Locking)

Central Servers +
Distributed Clients

Multi-Datacenter

Raft Consensus

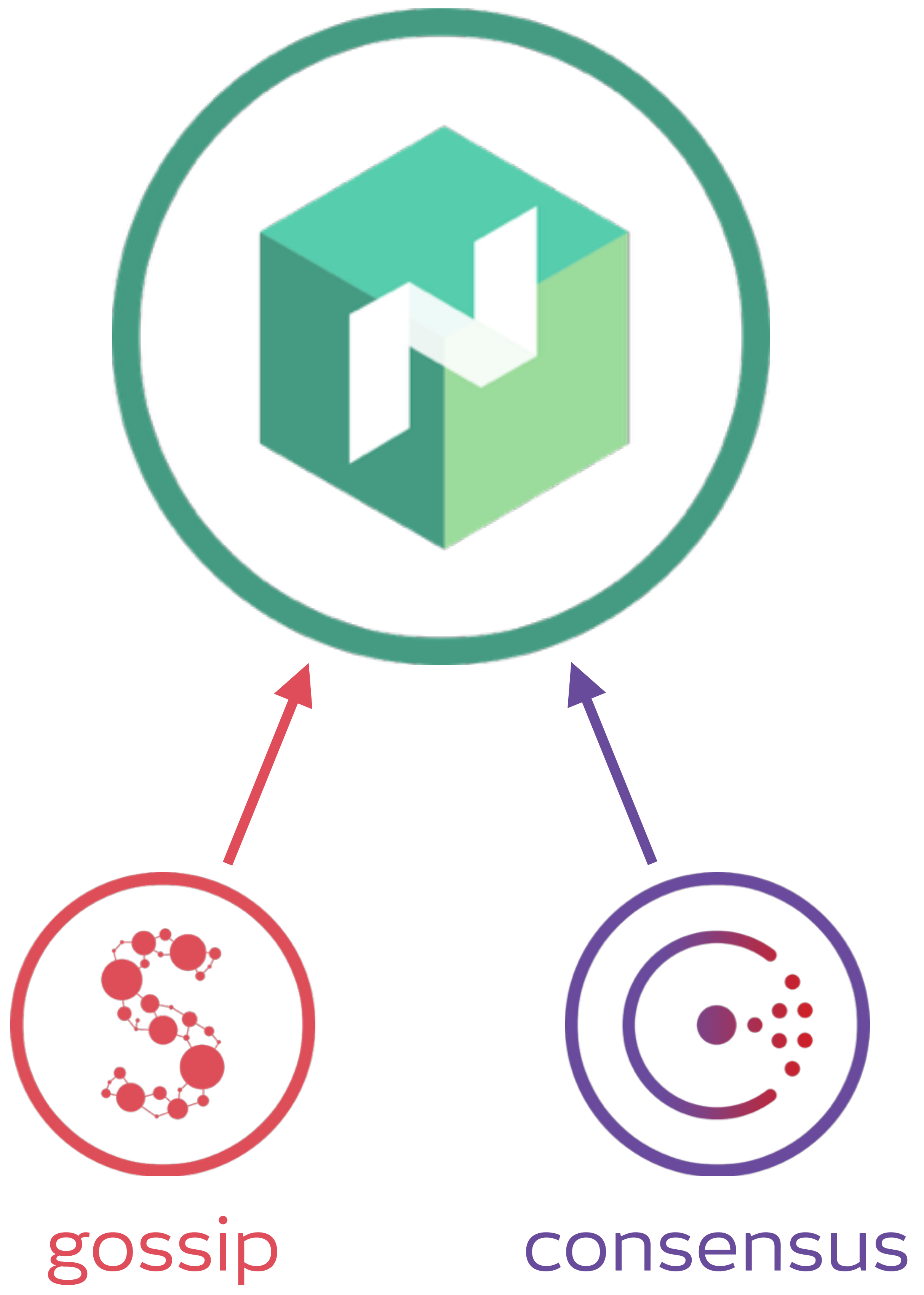
Large Scale

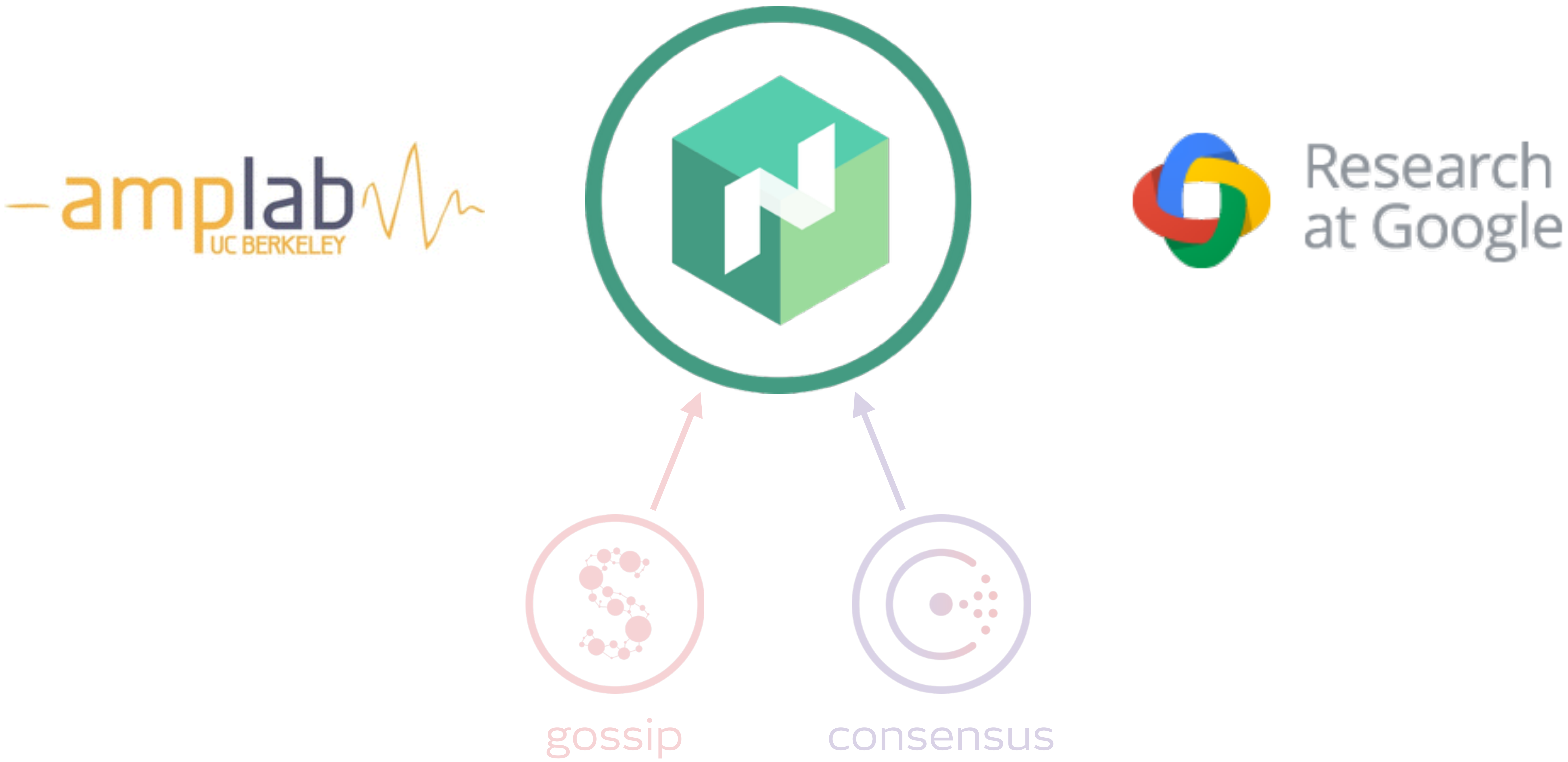
Production Hardened

Mature Libraries

Design Patterns

No Scheduling Logic





HASHICORP

Built on Research

Research at Google

Large-scale cluster management at Google with Borg

Abstract: Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines. It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior.

We present a summary of the Borg system architecture and features, important design decisions, a quantitative analysis of some of its policy decisions, and a qualitative examination of lessons learned from a decade of operational experience with it.

Omega: flexible, scalable schedulers for large compute clusters

Abstract: Increasing scale and the need for rapid response to changing requirements are hard to meet with current monolithic cluster scheduler architectures. This restricts the rate at which new features can be deployed, decreases efficiency and utilization, and will eventually limit cluster growth. We present a novel approach to address these needs using parallelism, shared state, and lock-free optimistic concurrency control. We compare this approach to existing cluster scheduler designs, evaluate how much interference between schedulers occurs and how much it matters in practice, present some techniques to alleviate it, and finally discuss a use case highlighting the advantages of our approach -- all driven by real-life Google production workloads.

amplab UC BERKELEY

Sparrow: Low Latency Scheduling for Interactive Cluster Services

Posted on March 28, 2012 by Patrick Wendell

The Sparrow project introduces a distributed cluster scheduling architecture which supports ultra-high throughput, low latency task scheduling. By supporting very low-latency tasks (and their associated high rate of task turnover), Sparrow enables a new class of cluster applications which analyze data at unprecedented volume and speed. The Sparrow project is under active development and maintained in our public github repository.

Mesos – Dynamic Resource Sharing for Clusters

Posted on November 21, 2011 by kilov

Mesos is a cluster manager that provides efficient resource isolation and sharing across distributed applications, or *frameworks*. It can run Hadoop, MPI,Hypertable, Spark (a new framework for low-latency interactive and iterative jobs), and other applications. Mesos is open source in the Apache Incubator.

Optimistic vs Pessimistic

Internal vs External State

Single vs Multi Level

Fixed vs Pluggable

Service vs Batch Oriented

Research at Google

amplab UC BERKELEY

Nomad

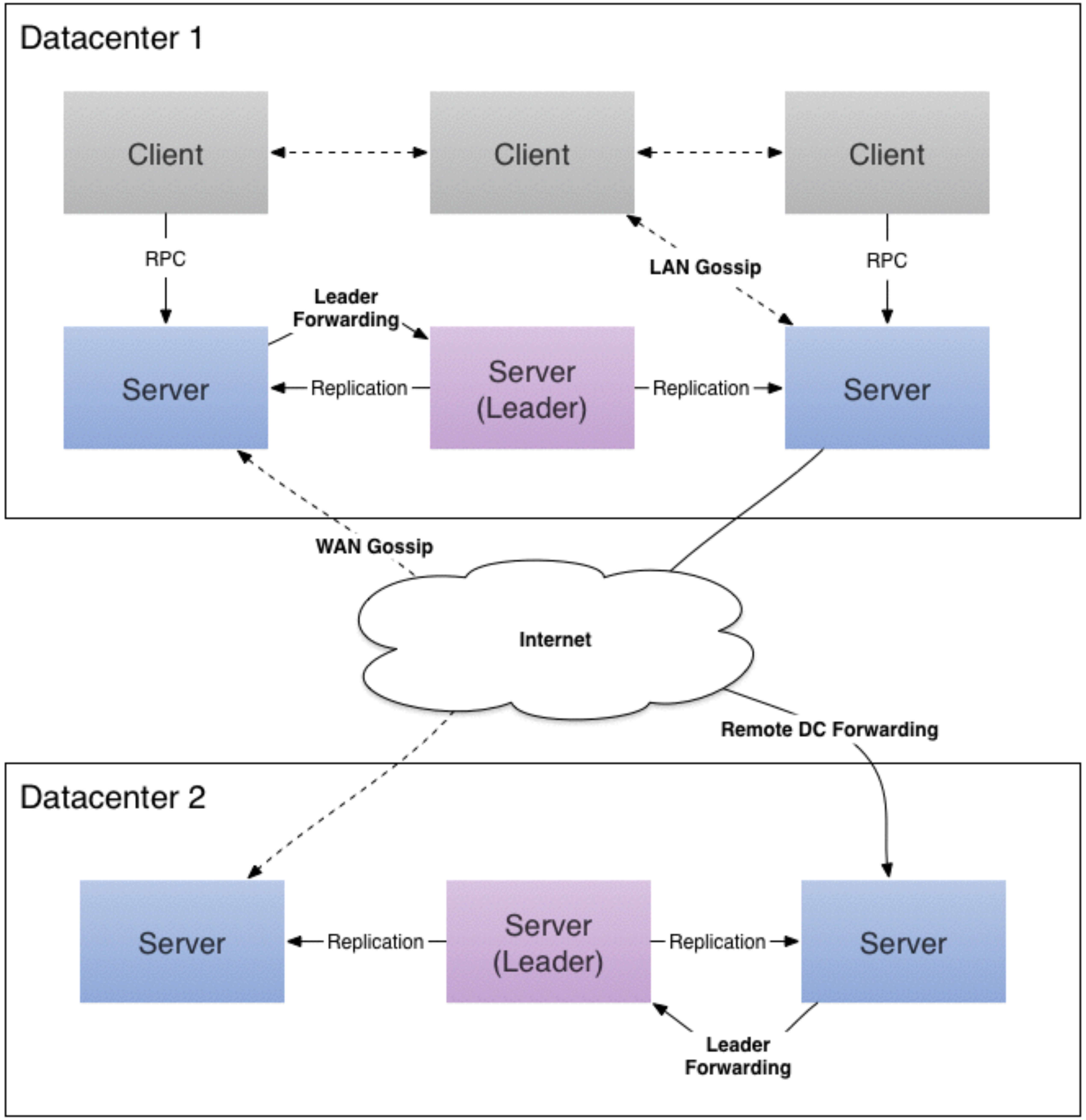
Inspired by Google Omega

Optimistic Concurrency

Internal State and Coordination

Service and Batch workloads

Pluggable Architecture



Datacenter 1
Client
Client
Client
RPC
LAN Gossip
RPC
Leader
Forwarding
Server
Replication
Server
(Leader)
Replication
Server
WAN Gossip
Internet
Remote DC Forwarding
Datacenter 2
Server
Replication
Server
(Leader)
Replication
Server
Leader
Forwarding

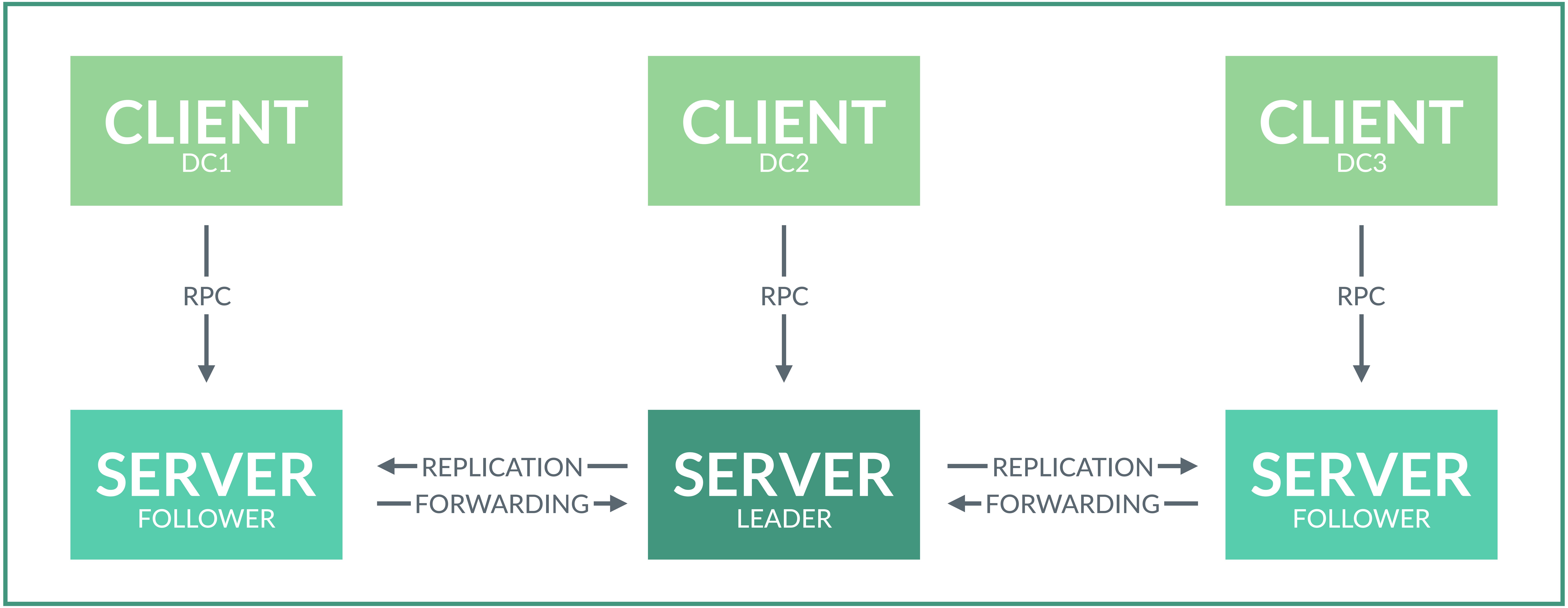
HASHICORP

Multi-Datacenter

Servers Per DC

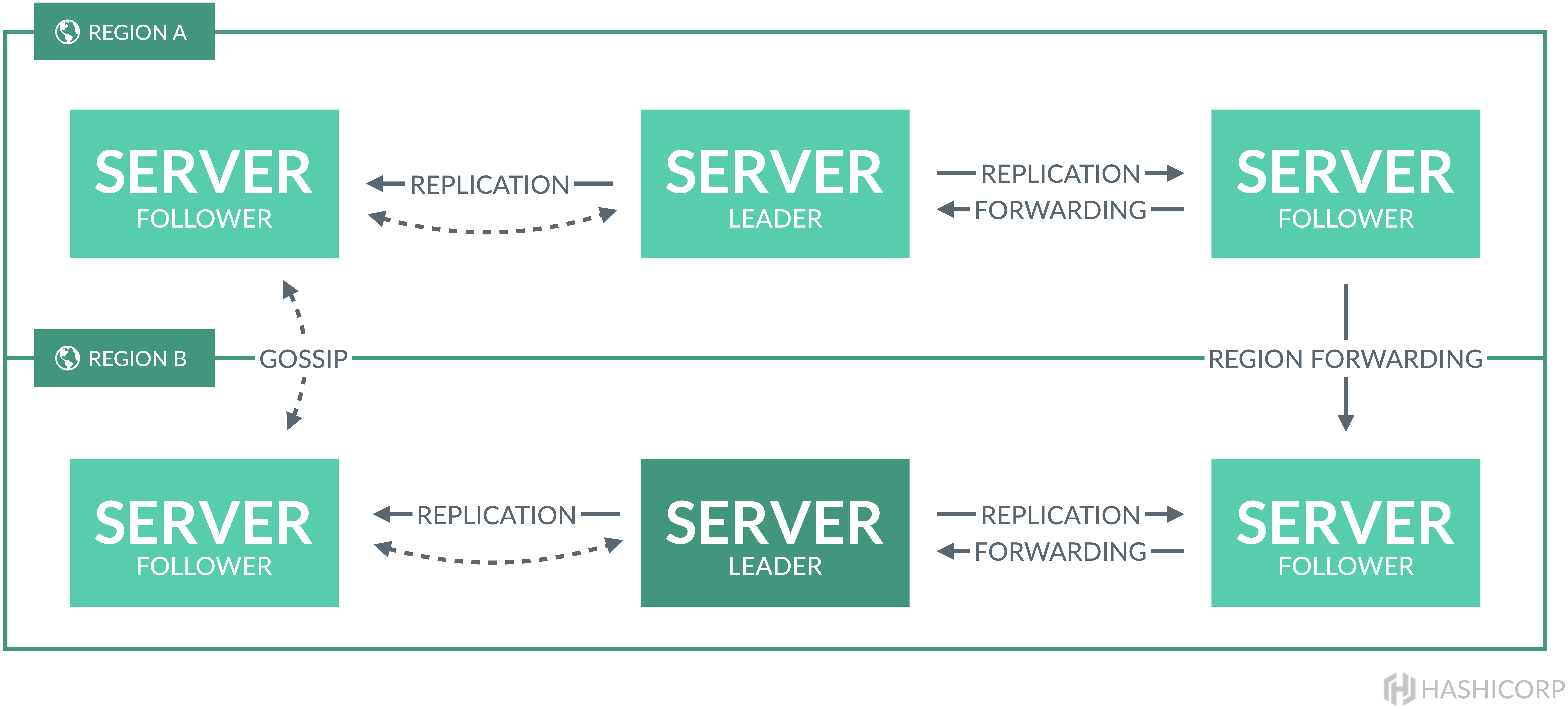
Failure Isolation Domain

is the Datacenter



HASHICORP

Single Region Architecture



Multi Region Architecture

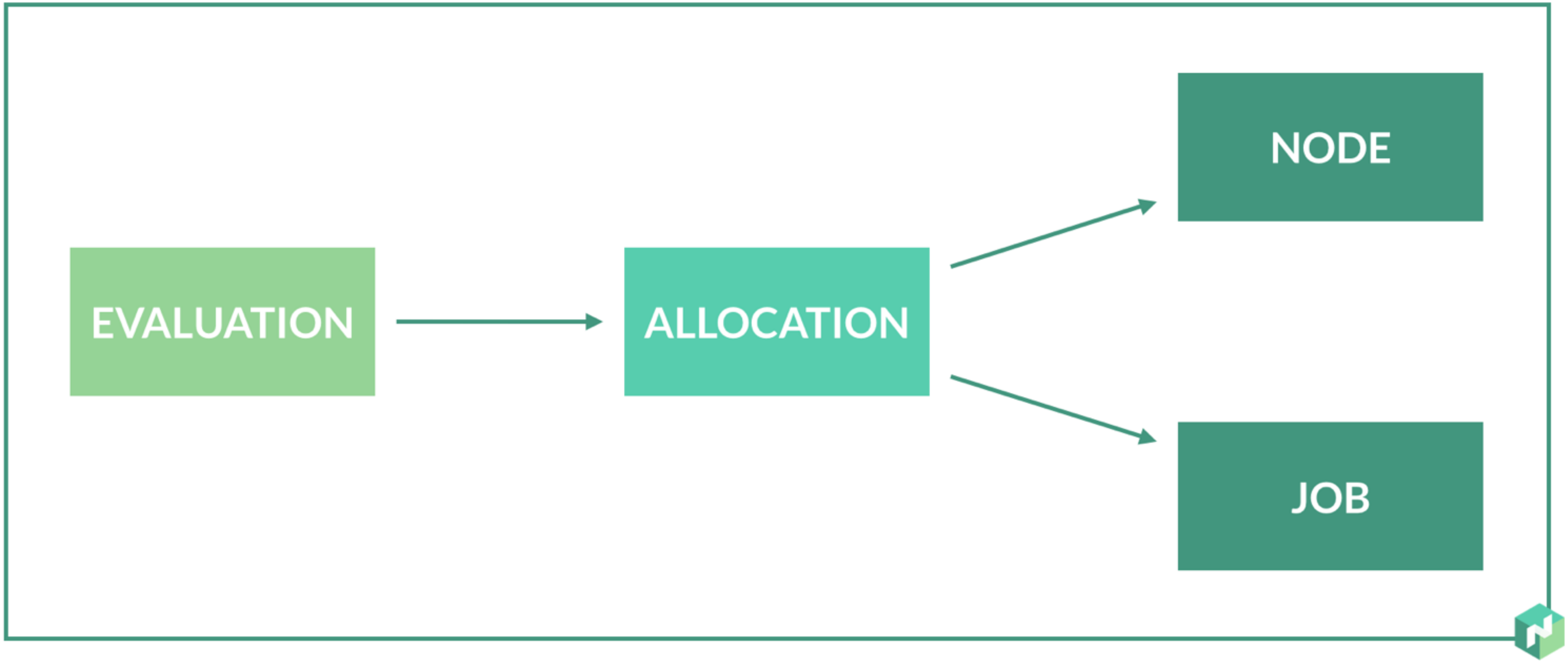
Region is Isolation Domain

1-N Datacenters Per Region

Flexibility to do 1:1 (Consul)

Scheduling Boundary

Nomad



Data Model

Evaluations ~= State Change Event

Create / Update / Delete Job

Node Up / Node Down

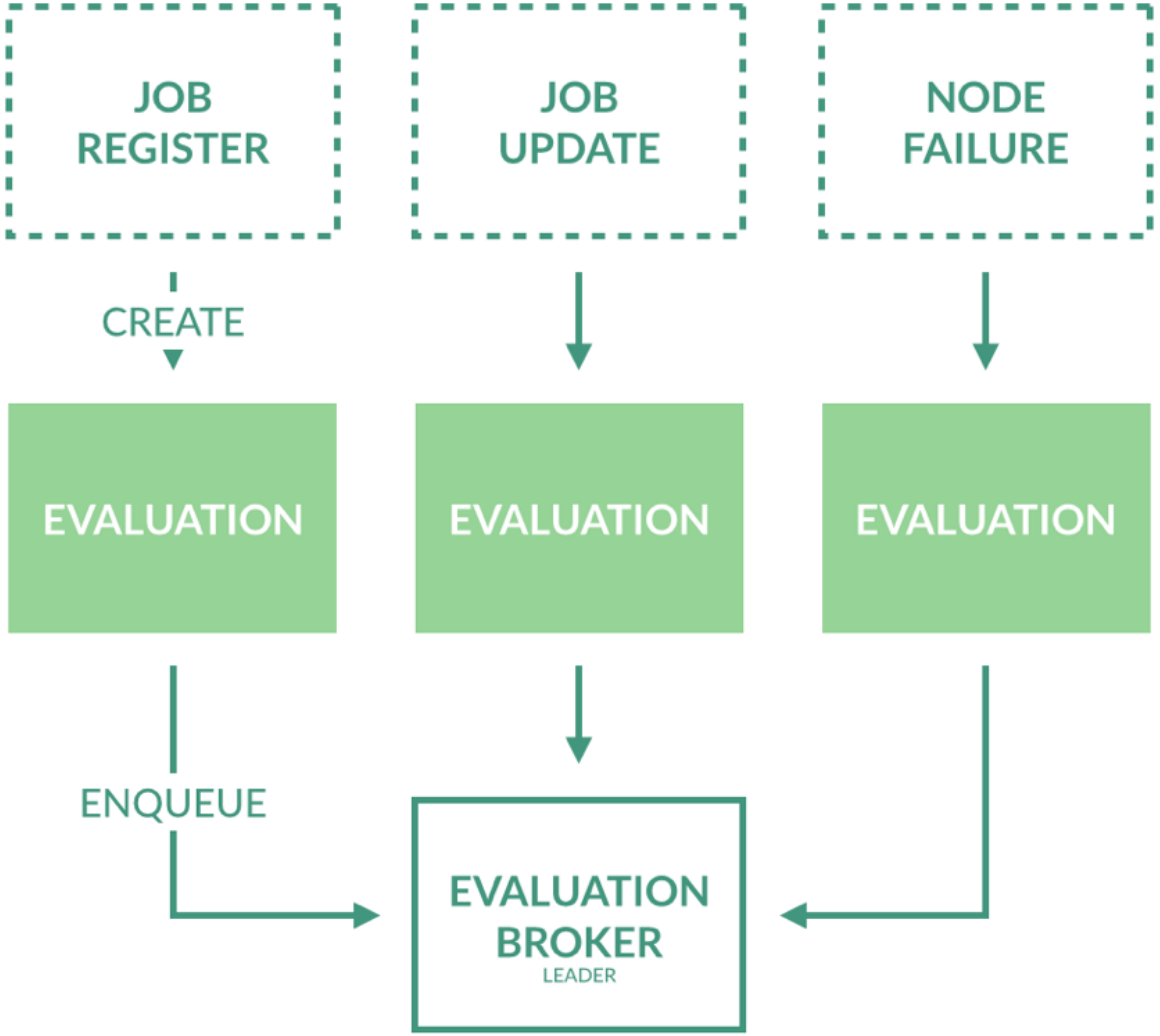
Allocation Failed

“Scheduler” =

```
func(Eval) => []AllocUpdates
```

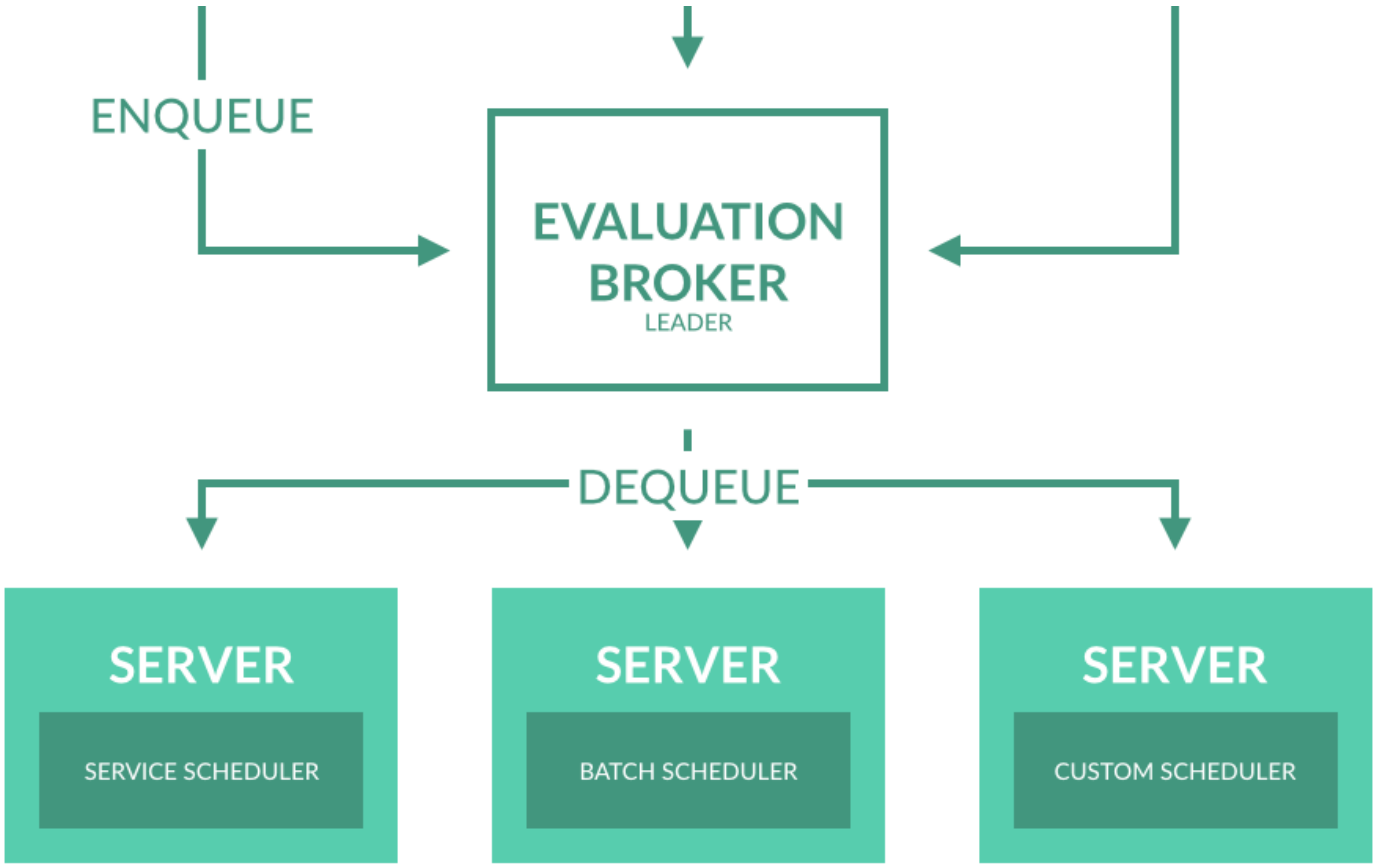
Scheduler func's can specialize

(Service, Batch, System, etc)



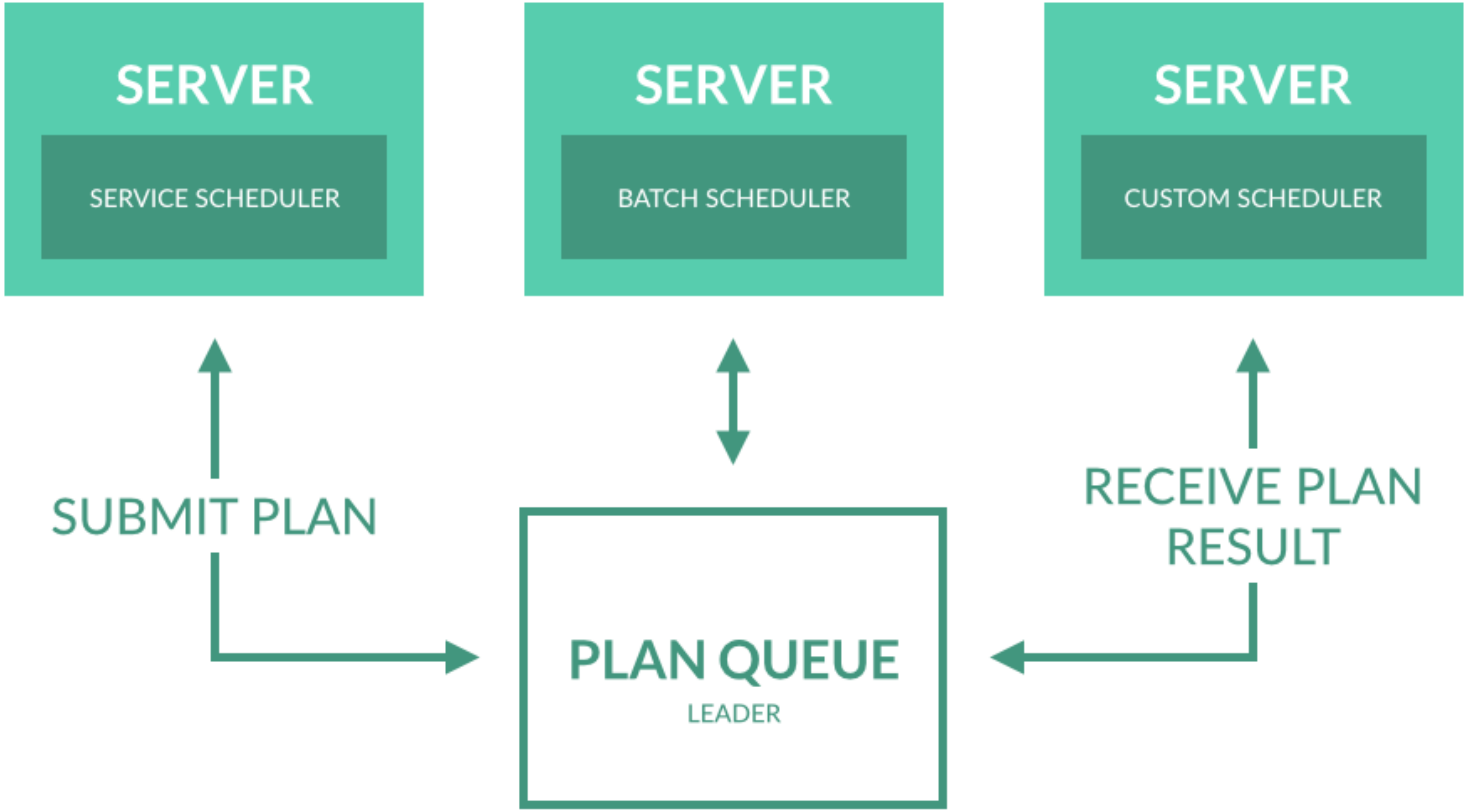
HASHICORP

Evaluation Enqueue



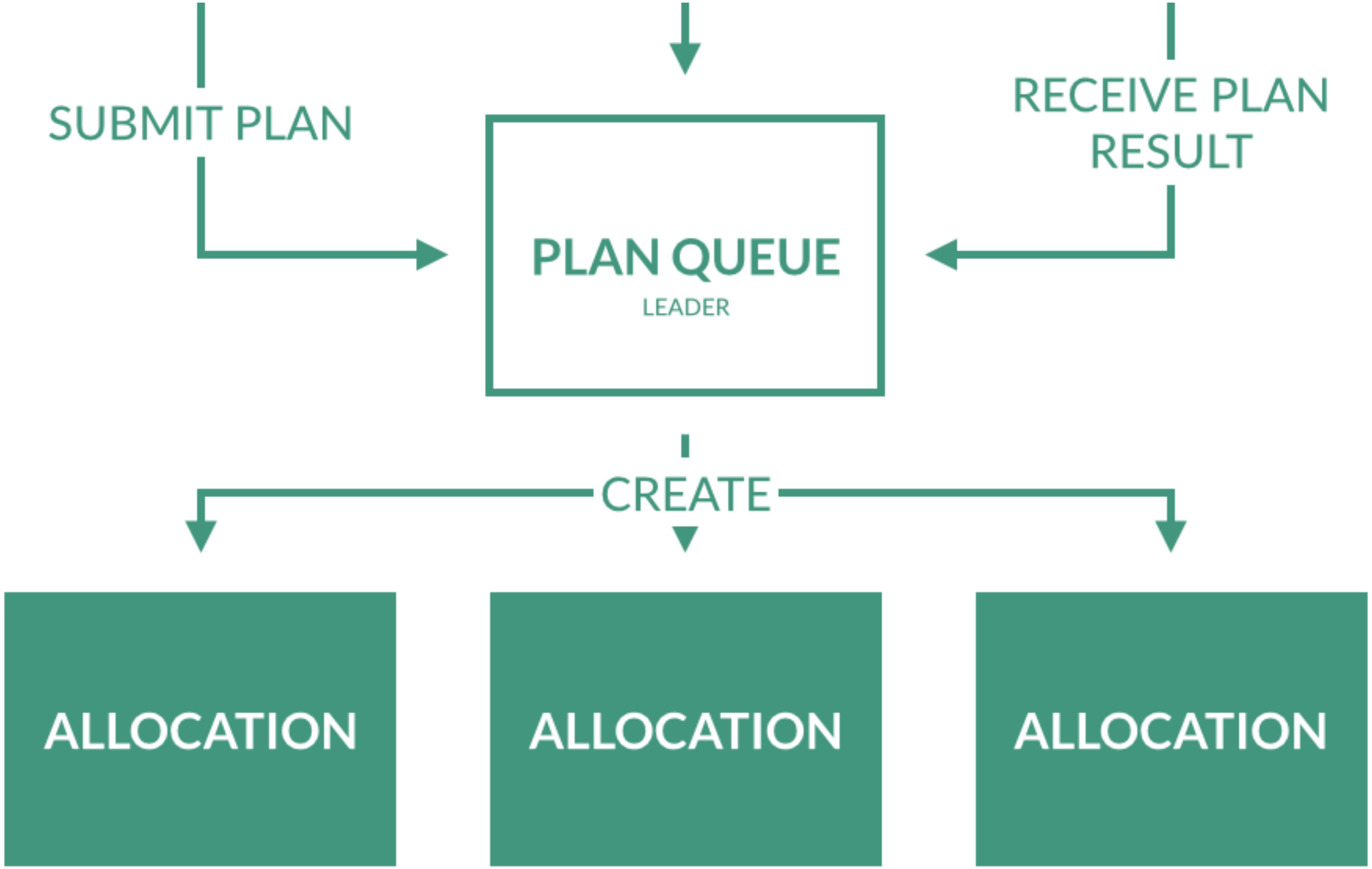
HASHICORP

Evaluation Dequeue



HASHICORP

Plan Generation



HASHICORP

Plan Execution

External Event

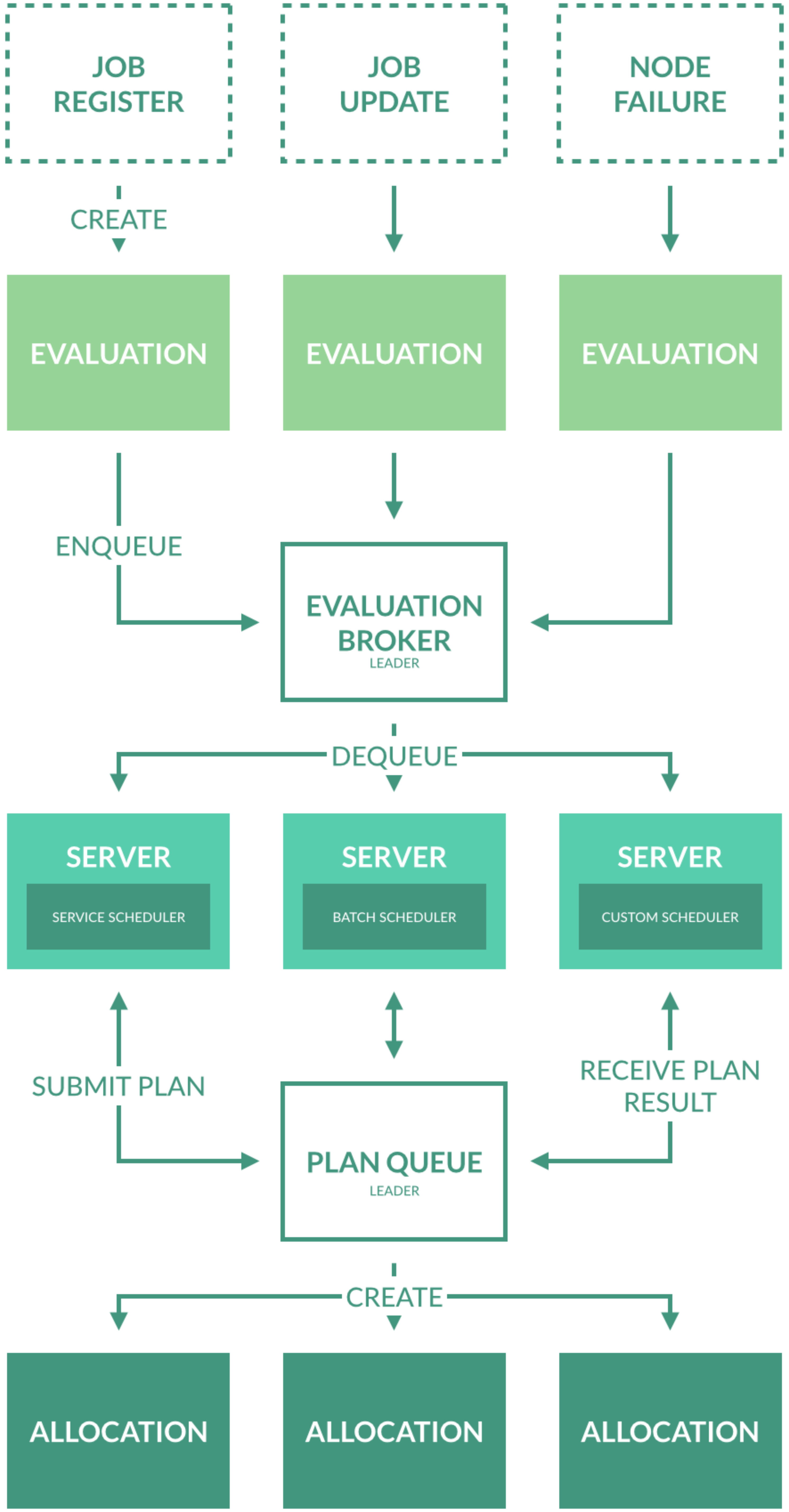
Evaluation Creation

Evaluation Queuing

Evaluation Processing

Optimistic Coordination

State Updates



HASHICORP

Omega Class Scheduler

Pluggable Logic

Internal Coordination and State

Multi-Region / Multi-Datacenter

HASHICORP

Server Architecture

Broad OS Support

Host Fingerprinting

Pluggable Drivers

HASHICORP

Client Architecture

Type	Examples
Operating System	Kernel, OS, Versions
Hardware	CPU, Memory, Disk
Applications	Java, Docker, Consul
Environment	AWS, GCE

Fingerprinting

Constrain Placement and Bin Pack

"Task Requires Linux, Docker, and PCI-Compliant Hardware" expressed as Constraints

HASHICORP

Fingerprinting

"Task needs 512MB RAM and 1 Core"
expressed as Resource Ask

Execute Tasks

Provide Resource Isolation

HASHICORP

Drivers

Containerized

Docker

Rocket

Virtualized

Qemu / KVM

Standalone

Java Jar

Static Binaries

Containerized	Docker Rocket Windows Server Containers
Virtualized	Qemu / KVM Xen Hyper-V
Standalone	Java Jar Static Binaries C#

Nomad

Workload Flexibility:

Schedulers

Fingerprints

Drivers

Job Specification

Nomad

Operational Simplicity:

Single Binary

No Dependencies

Highly Available

HASHICORP

Released in October

Service and Batch Scheduler

Docker, Qemu, Exec, Java Drivers

HASHICORP

Nomad 0.1

DigitalOcean

HASHICORP

Case Study

3 servers in NYC3

100 clients in NYC3, SFO1, AMS2/3

1000 Containers

DigitalOcean

HASHICORP

Case Study

<1s to schedule

1s to first start

6s to 95%

8s to 99%

DigitalOcean

HASHICORP

Case Study

Service Discovery

System Scheduler

Restart Policies

Enhanced Constraints

HASHICORP

Nomad 0.2 - Service Workloads

Cron

Job Queuing

Latency-Aware Scheduling

Nomad 0.2 in Prod

Stress Testing

Atlas Integration

HASHICORP

Production Hardening

Nomad

Cluster Scheduler

Easily Deploy Applications

Job Specification

Nomad

Higher Resource Utilization

Decouple Work from Resources

Better Quality of Service

Thanks!

Q/A