



# HOMOLOGOUS SPARK CLUSTERS USING NOMAD

Alex Dadgar



## Alex Dadgar

Team Lead of Nomad  
HashiCorp

# Deploying Spark



# Deploying Spark



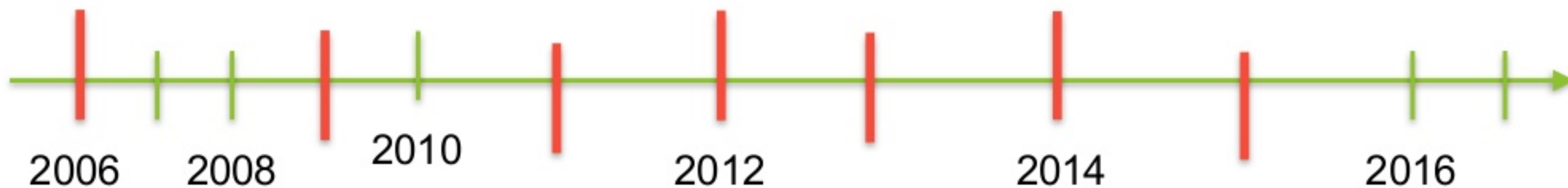
HashiCorp

# Nomad

# Why?



kubernetes



# We're in a transitional period

- Migrations to cloud
- Monolithic applications toward micro-services
- DevOps and an explosion of tooling
- Rise of cluster schedulers



# Big Data is here to stay

- 2014 IDC Research Study
- Compound growth of 42% - near doubling!
- 2010 - 1 ZB
- 2020 - 50 ZB



**Spark is here to stay!**





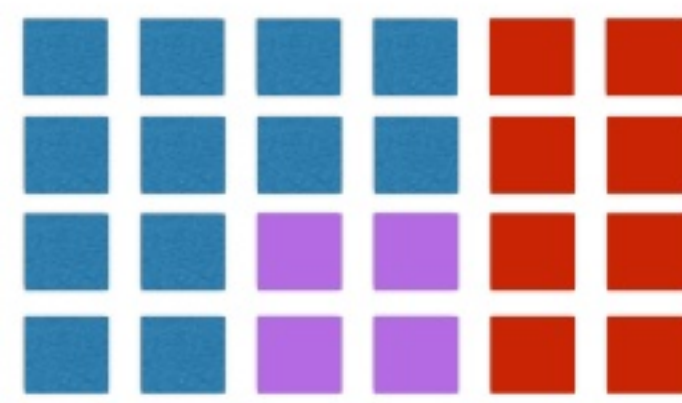
# What is Nomad?



Jobs



Nodes





# Nomad

## Cluster Scheduler

- Easy for Developers
- Operationally Simple
- Built for Scale

# Easy for Developers





```
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
        port "redis" {}
      }
    }
  }
}
```

# Job Specification


Declares **what** to run

# Job Specification

Nomad determines **where** and  
manages **how** to run

# Job Specification

Powerful yet simple



```
# 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
        port "redis" {}
      }
    }
  }
}
```

Containerized

Docker

Rkt

LXC

---

Virtualized

Qemu / KVM

---

Standalone

Java Jar

Static Binaries

## Containerized

Docker

Windows Server Containers

Rkt

LXC

---

## Virtualized

Qemu / KVM

Xen

Hyper-V

---

## Standalone

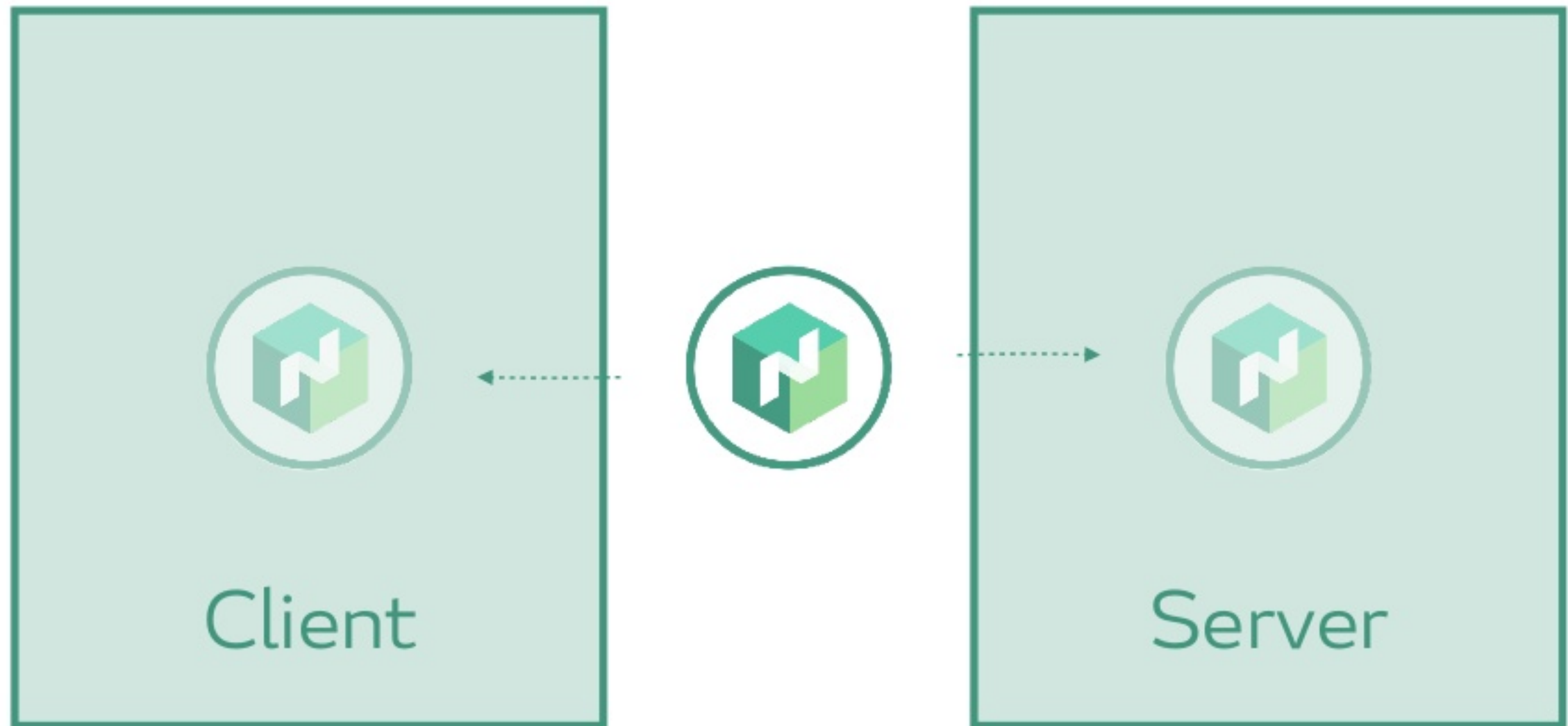
Java Jar

Static Binaries

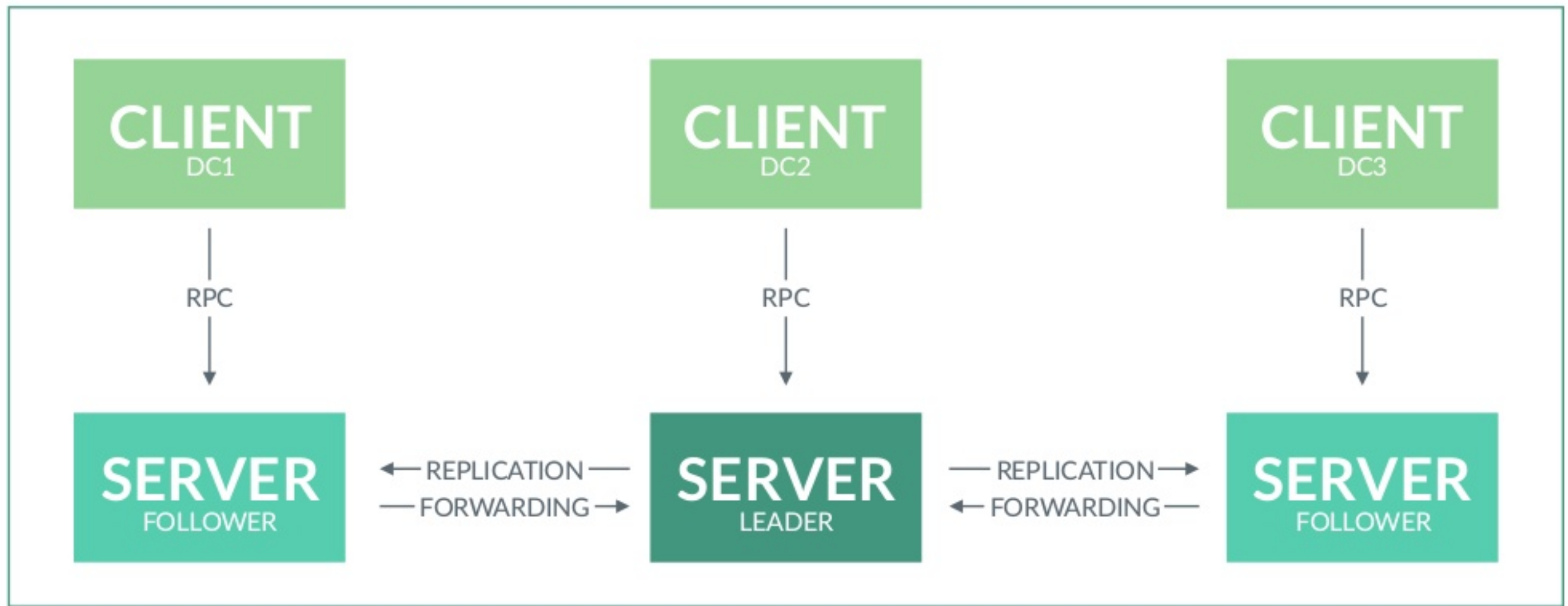
C#



# Operationally Simple



# Single Region Architecture

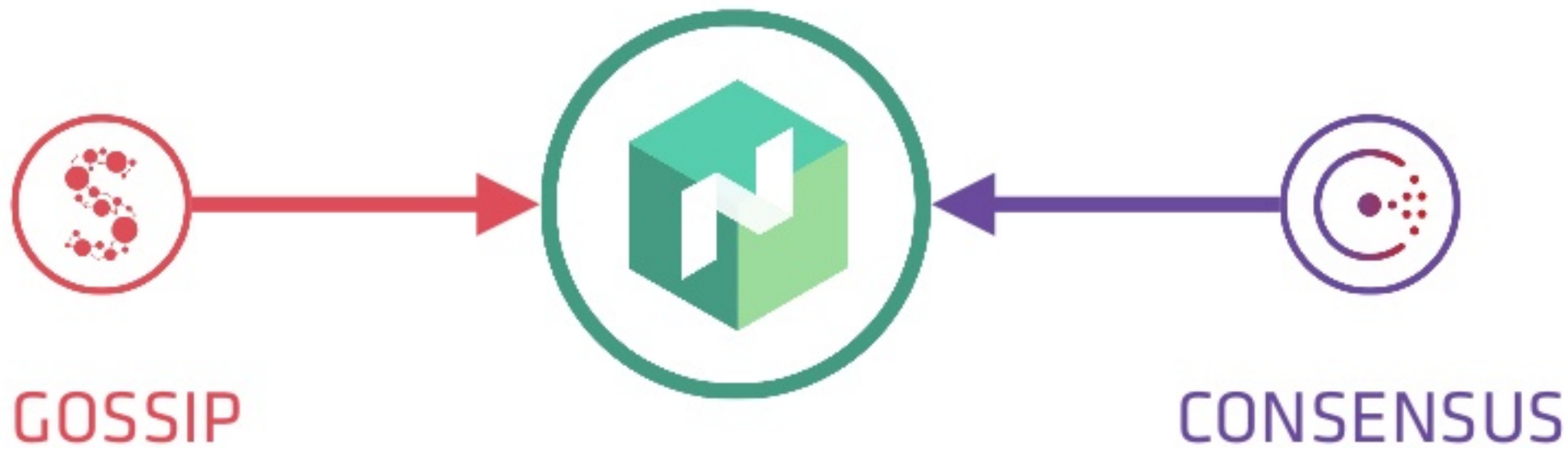


# Built for Scale

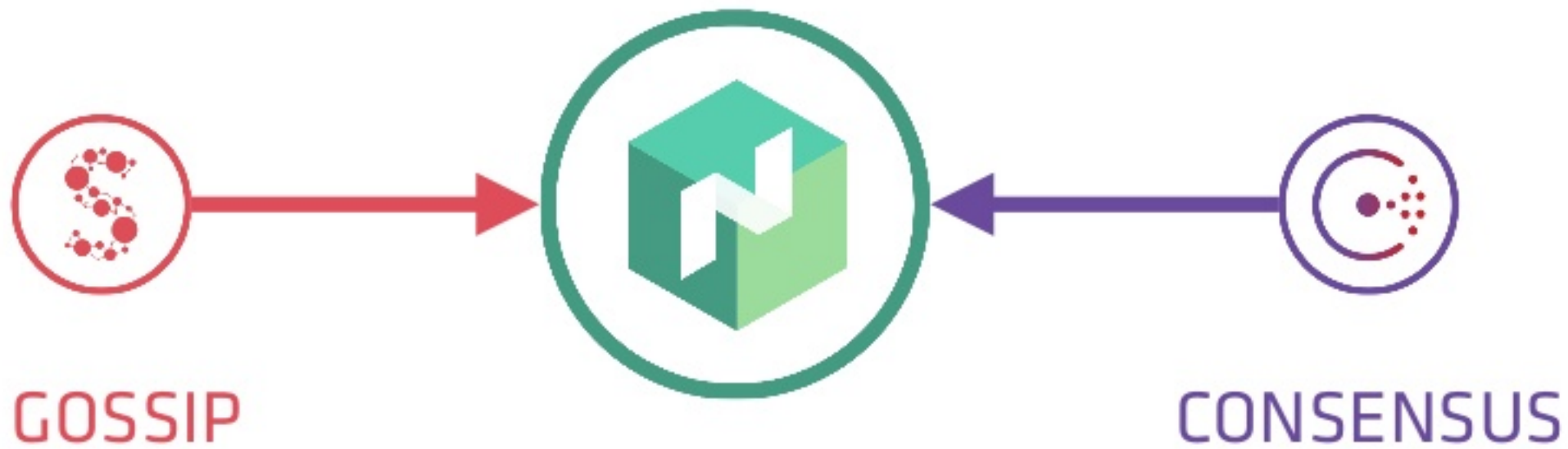
# Built on Experience

Mature Libraries

Proven Design Patterns



# Built on Research







## 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.



## 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](#).

## 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.

## 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](#).





# Nomad

## Million Container Challenge

1,000 Jobs

1,000 Tasks per Job

5,000 Hosts on GCE

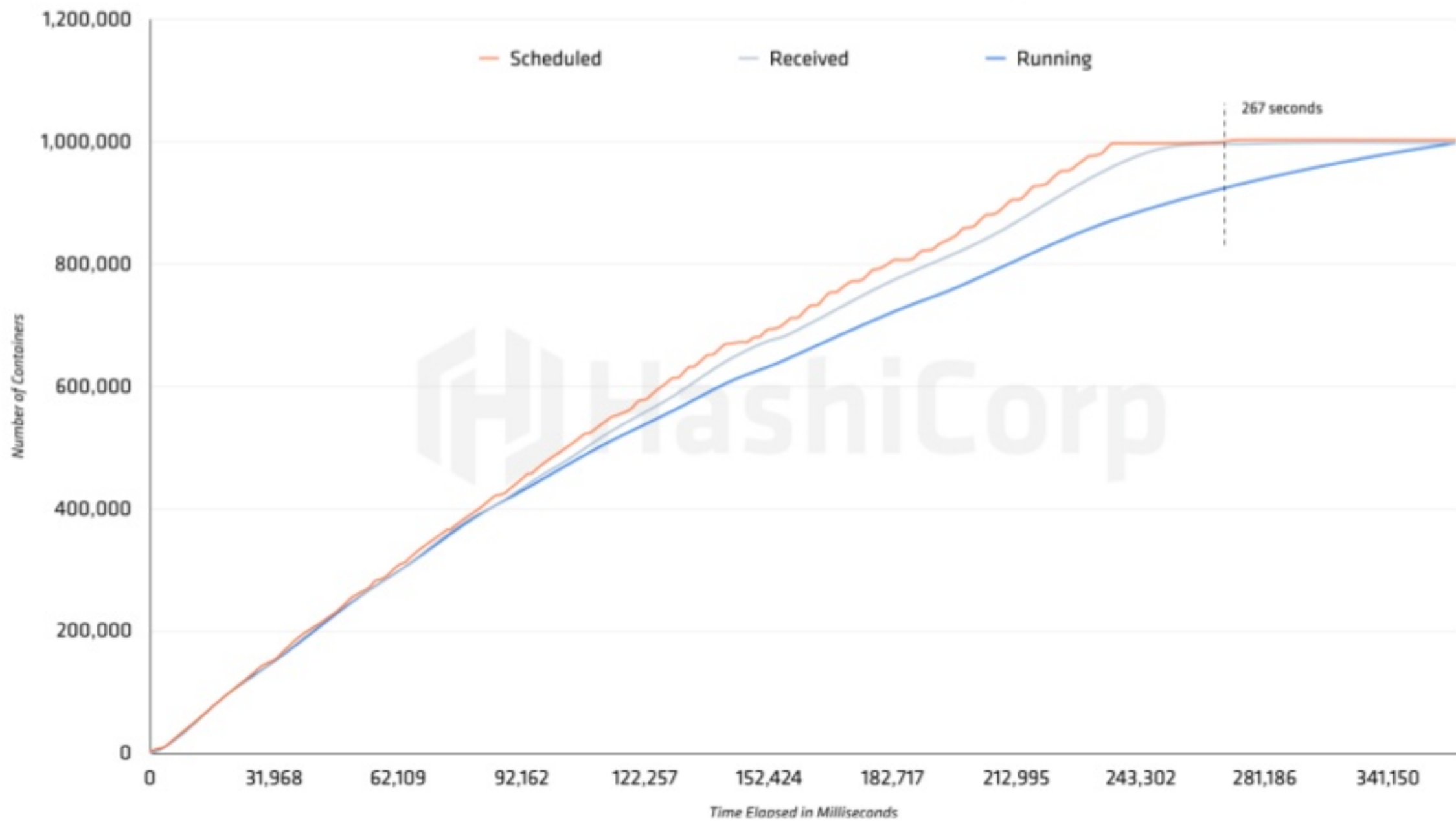
1,000,000 Containers



==> Nomad agent started! Log data will stream in below:

```
2016/03/14 17:34:26 [DEBUG] client: available drivers [raw_exec java qemu docker exec]
2016/03/14 17:34:26 [INFO] client: server address list: [127.0.0.1:4647]
2016/03/14 17:34:26 [DEBUG] client: checking for node changes at duration 5s
2016/03/14 17:34:27 [DEBUG] client: complete
2016/03/14 17:34:28 [DEBUG] client:
2016/03/14 17:35:05 [DEBUG] client: 8 (pulled 1) (filtered 0)
2016/03/14 17:35:05 [DEBUG] client: updated 0) (ignore 0)
2016/03/14 17:35:05 [DEBUG] client: 76cb4259-a20f-1581-877f-1a
2016/03/14 17:35:05 [DEBUG] client: 76cb4259-a20f-1581-877f-1a
2016/03/14 17:35:05 [DEBUG] client: (filtered 0)
2016/03/14 17:35:05 [DEBUG] client: 99-9a
2016/03/14 17:35:05 [DEBUG] client: 8-97f
```

Nomad C1M (1 Million Docker Containers)



“640 KB ought to be enough for anybody.

**-Bill Gates**

2nd Largest Hedge Fund

18K Cores

5 Hours

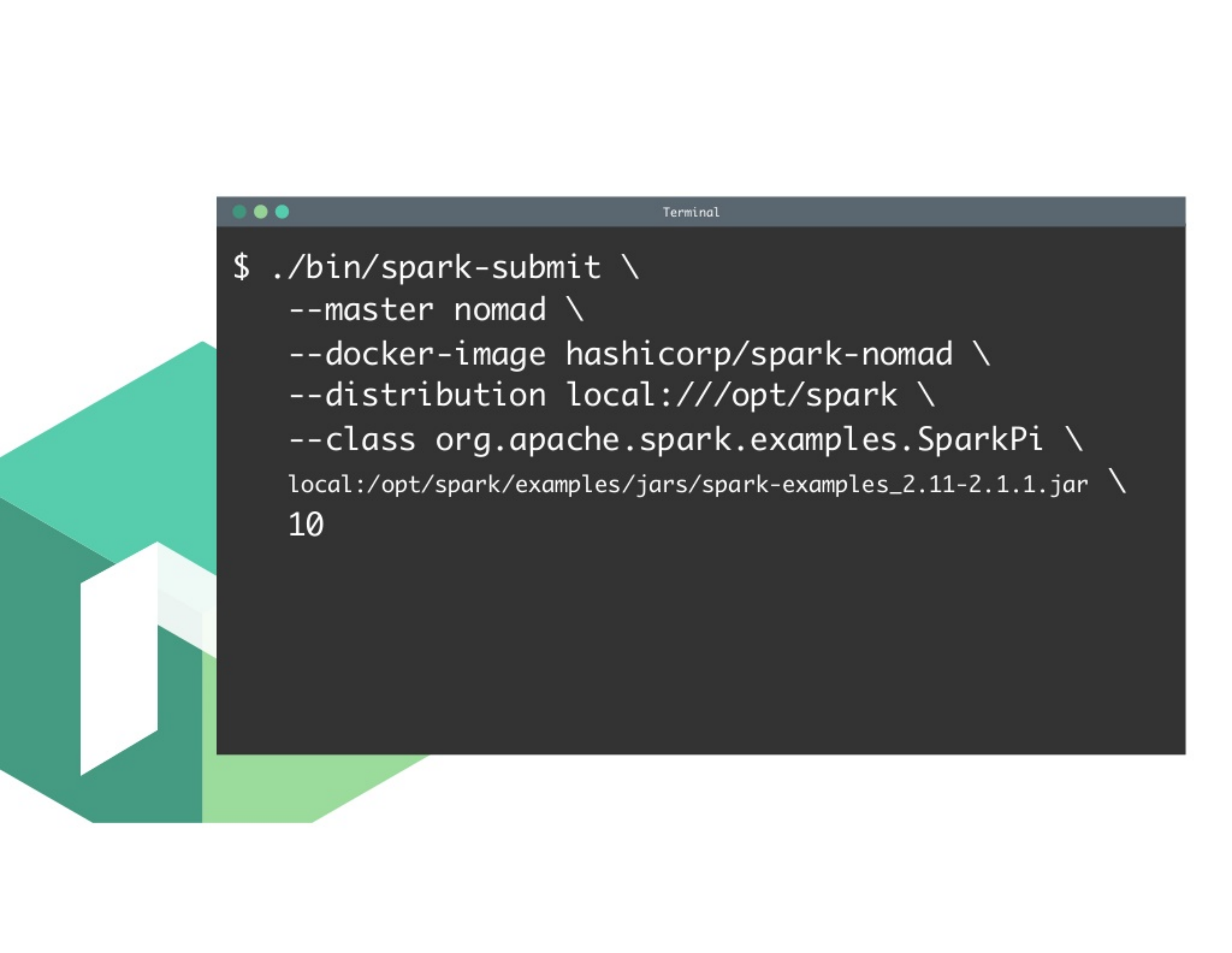
**2,200** Containers/second



# Benefits to Deploying on Nomad Today and in the Future

# Today: Uncompromised Spark

- ./spark-submit workflow remains
- Supports: Scala, Java, R, Python
- Support Spark Shell
- Dynamic Executors
- Run with or without Docker

A terminal window titled "Terminal" is shown with a dark background and white text. To the left of the terminal is a large, stylized green geometric shape resembling a hexagon with a white rectangular cutout. The terminal contains a command to submit a Spark application using Nomad as the orchestrator. The command is split across several lines, each ending with a backslash to indicate continuation. The command specifies the master as 'nomad', the Docker image as 'hashicorp/spark-nomad', the distribution as 'local:///opt/spark', the class as 'org.apache.spark.examples.SparkPi', and the path to the Spark examples JAR file as 'local:/opt/spark/examples/jars/spark-examples\_2.11-2.1.1.jar'. The number '10' is entered at the end of the command line.

```
$ ./bin/spark-submit \  
  --master nomad \  
  --docker-image hashicorp/spark-nomad \  
  --distribution local:///opt/spark \  
  --class org.apache.spark.examples.SparkPi \  
  local:/opt/spark/examples/jars/spark-examples_2.11-2.1.1.jar \  
  10
```

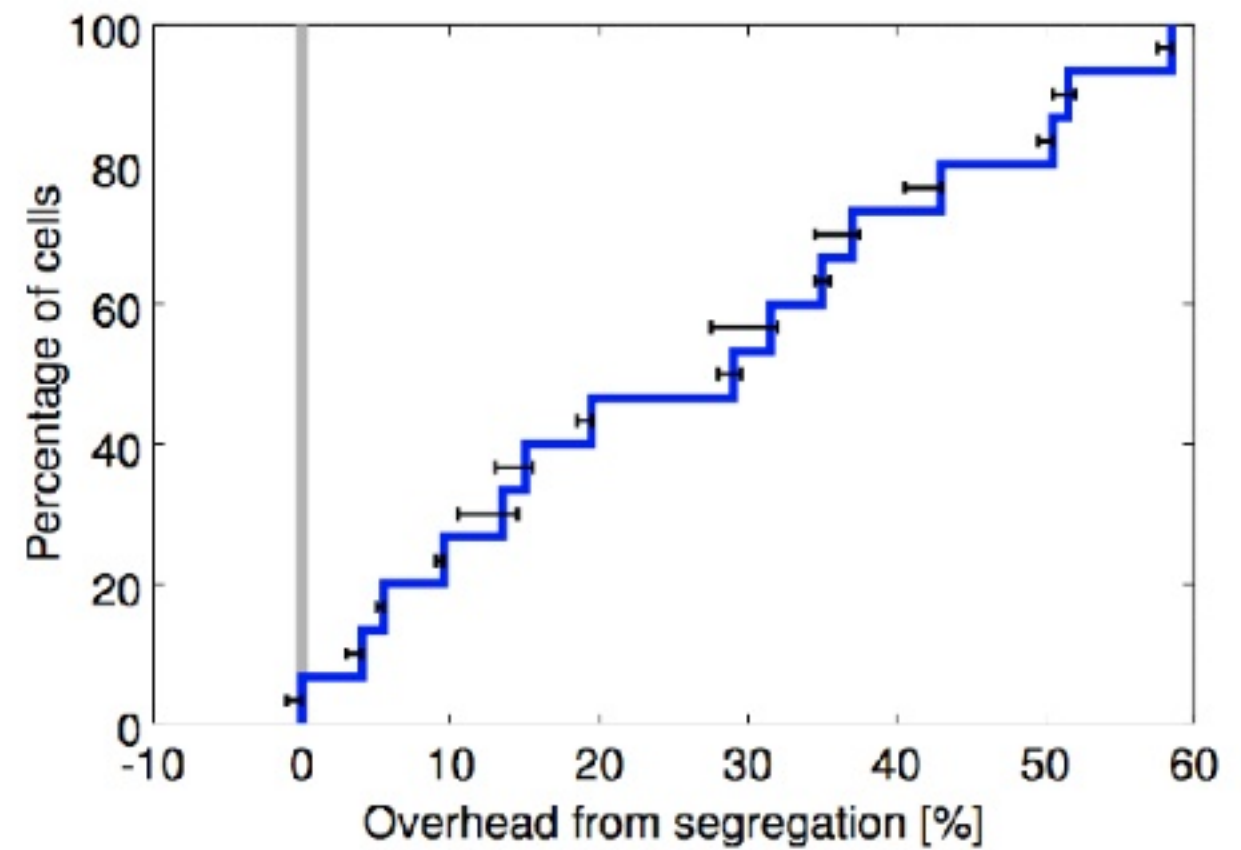


# Today: Shared Batch/Service Cluster

- No separate Spark cluster and Service cluster
- Higher density and reduced cost
- Operators manage one infrastructure
- Developers learn one tool

## Google Borg Paper


Figure 5 shows that segregating prod and non-prod work would need 20–30% more machines in the median cell to run our workload.



(b) CDF of additional machines that would be needed if we segregated the workload of 15 representative cells.

# Today: Security

- Integration with HashiCorp Vault
- Vault stores static secrets and can generate dynamic secrets
  - IAM credentials
- Don't bake secrets into Spark jobs



```
task "payment-api" {
  ...
  vault {
    policies = ["s3_user_data_rw"]
  }

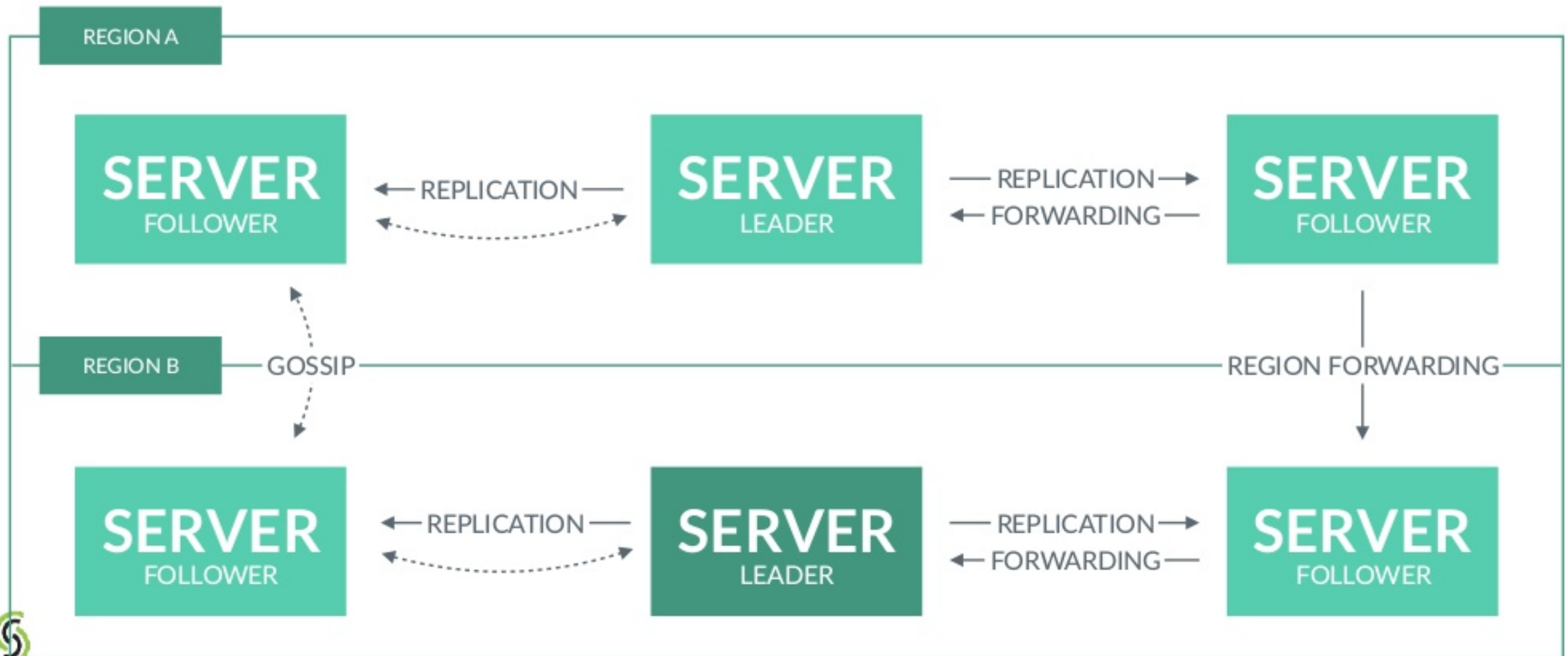
  template {
    data = <<END
    {{with $secret := vault "aws/creds/deploy" }}
    AWS_SECRET_ACCESS_KEY={{$secret.Data.access_key}}
    AWS_ACCESS_KEY_ID={{$secret.Data.secret_key}}
    {{end}}
    <<END
    dest = "secrets/aws_creds"
    env  = true
  }
}
```

# Today: Security

- Went to great lengths to minimize the exposure of Vault token
  - Servers never see token
  - One time access (can detect tampering)
  - Write to in-memory file (tmpfs)
- Full talk: <https://youtu.be/4gAYyAA6h9E>



# Today: Multi Region/DC



# Today: Cron Spark Jobs

- Run a Spark Job on a cron schedule
- Responsibility of Nomad to manage and launch the job
- Higher Reliability

# Today: Templated Spark Jobs

- Spark Submit can take a Nomad Job file as a template
- Merges generated Spark job
- Fully customizable



# Today: Templated Spark Jobs

- Run a logging sidecar to ship Spark logs
- Retrieve secrets securely from Vault
- Register Spark jobs in service discovery
- Customize any Nomad tunable




```
job "template" {
  group "driver" {
    task "driver" {
      meta { "spark.nomad.role" = "driver"}
    }


    task "log-forwarding-sidecar" {
      # sidecar task definition here
    }
  }

  group "executor" {
    task "executor" {
      meta { "spark.nomad.role" = "executor" }
    }

    task "log-forwarding-sidecar" {
      # sidecar task definition here
    }
  }
}
```



```
Terminal
$ ./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master nomad \
  --docker-image hashicorp/spark-nomad \
  --distribution local:///opt/spark \
  --conf spark.nomad.job.template=template.json \
  local:/opt/spark/examples/jars/spark-examples_2.11-2.1.1.jar \
  10
```



```
job "template" {
  group "executor" {
    task "executor" {
      meta { "spark.nomad.role" = "executor" }

      template {
        data = <<END
{{with $secret := vault "aws/creds/deploy" }}
  AWS_SECRET_ACCESS_KEY={{$secret.Data.access_key}}
  AWS_ACCESS_KEY_ID={{$secret.Data.secret_key}} {{end}}
<<END
        dest = "secrets/aws_creds"
        env  = true
      }

      vault {
        policies = ["s3-mydata-rw"]
      }
    }
  }
}
```

# Future: Pre-emption

- Job Priorities: 0-100
- Run critical services at higher priority
- Run Spark Driver at higher priority than executors
- Preempt lower priority Spark Executors
- Still make progress

# Future: Quotas and Chargebacks

- Enable multi-tenant clusters
- Gate job-submission based on quota
- Control hogging of cluster
- Fine-grain chargebacks

# Future: GPU

- Speed up Machine-Learning Tasks
- Nomad Clients detect GPUs
- Spark jobs can annotate desire to run on GPU machines
- Other tasks on host won't have access to GPU



# Future: Over-Subscription

- Jobs declare their resource requirement
- Often don't use all of it
  - Ask for 4 GB of Memory and use 1 GB
- Detect unused resource and make available to batch jobs



# Play with it!

- PR is out: <https://github.com/apache/spark/pull/18209>
- Docker Image: <https://hub.docker.com/r/hashicorp/spark-nomad/>



# Thank You.

Twitter: @adadgar

GitHub: @dadgar