Decomposing the Monolith

QCon - New York- 2015

Agenda

- VMTurbo
  - What we do
  - How we do it
  - Who we are
- Monolithic Architecture
- Pain Points
- Architectural Principles
- Organizational Principles
- Evolution Phase 1
  - Team
  - Architecture
- Next Steps

What is VMTurbo Operations Manager?

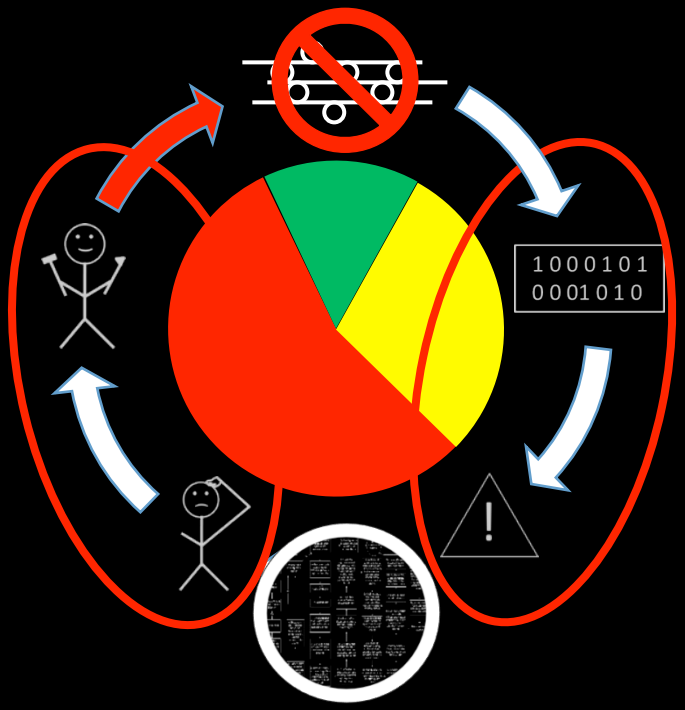
- A Demand-Driven Control System for the Data Center
- Looks at the Resources in the datacenter and the associated workload demands.
- Takes action to keep the workloads in their desired state.

VMTURBO

VMTurbo: How it Works

Assure Application Performance WHILE Maximize Efficiency

Desired State



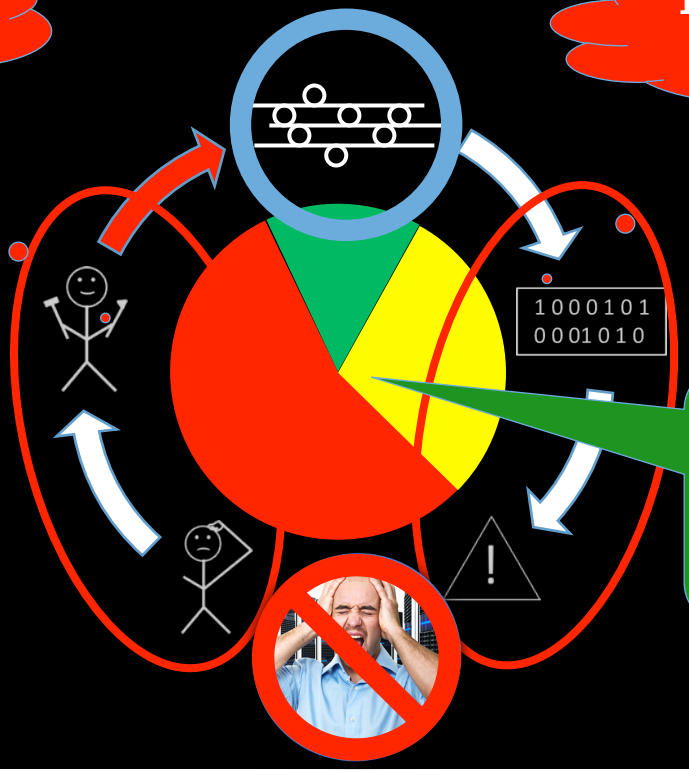
VMTURBO

Assure Application Performance WHILE Maximize Efficiency

Desired State

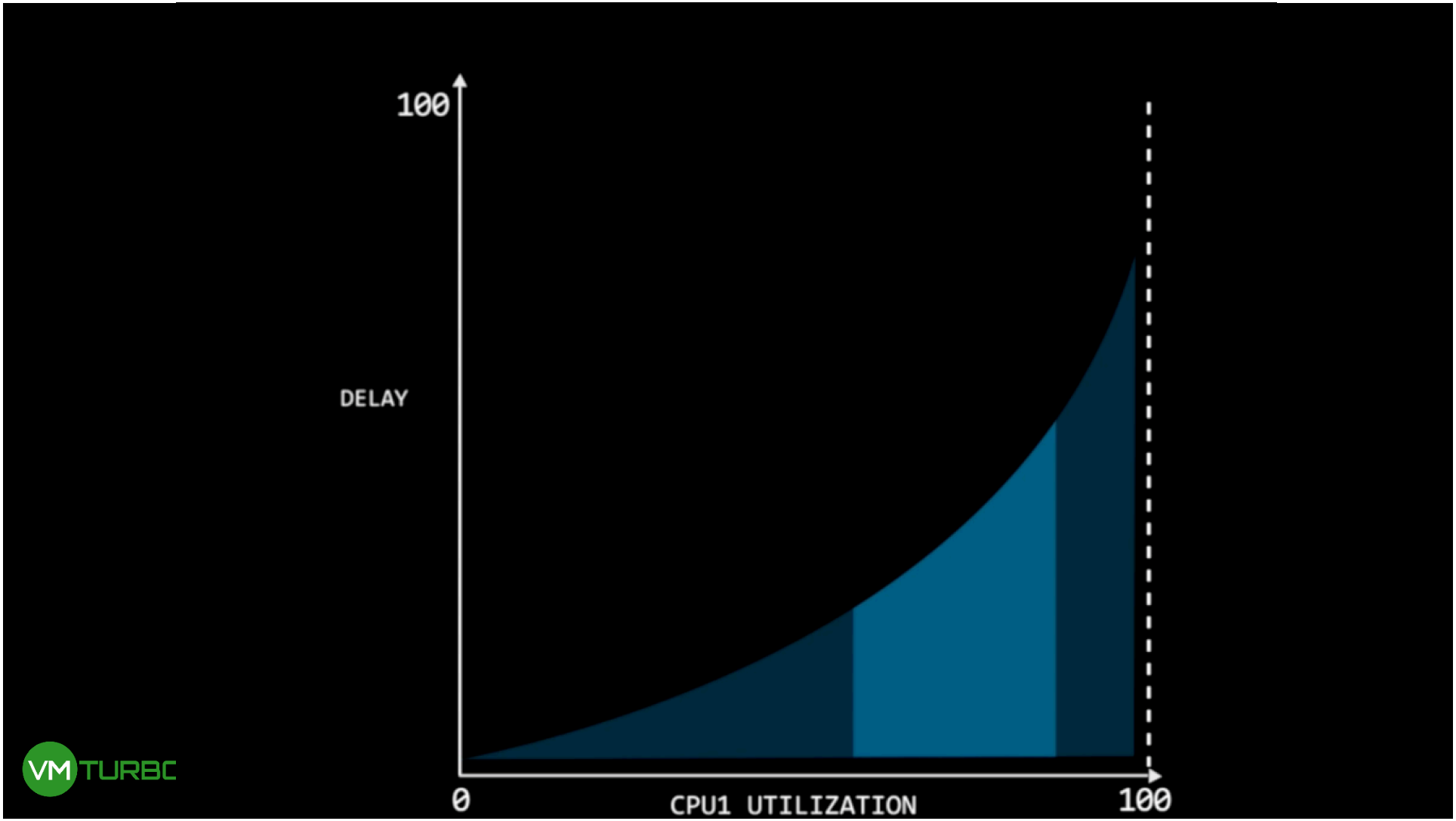
Doesn't assure Performance

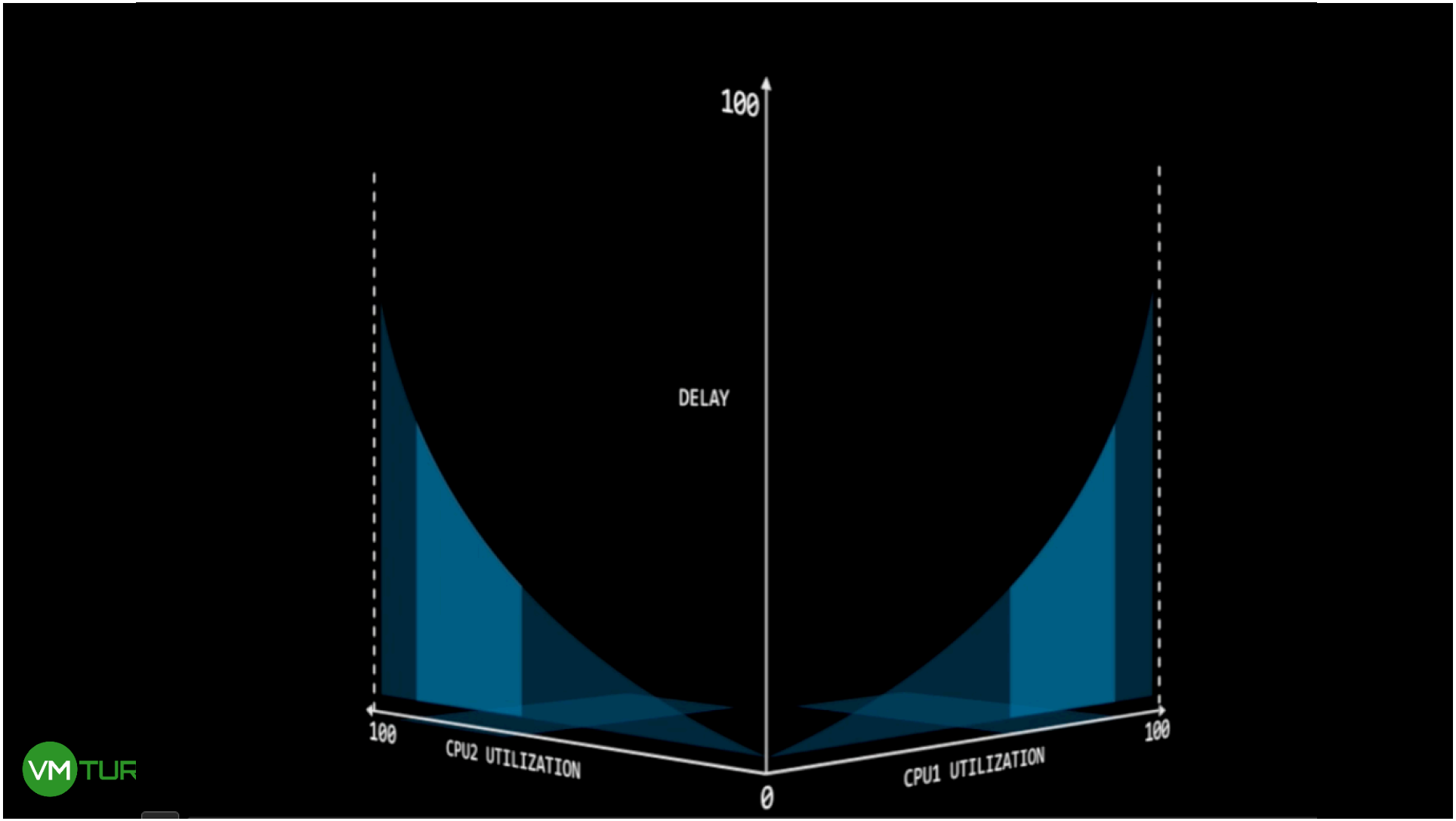
The bubbles of the doing & viewing

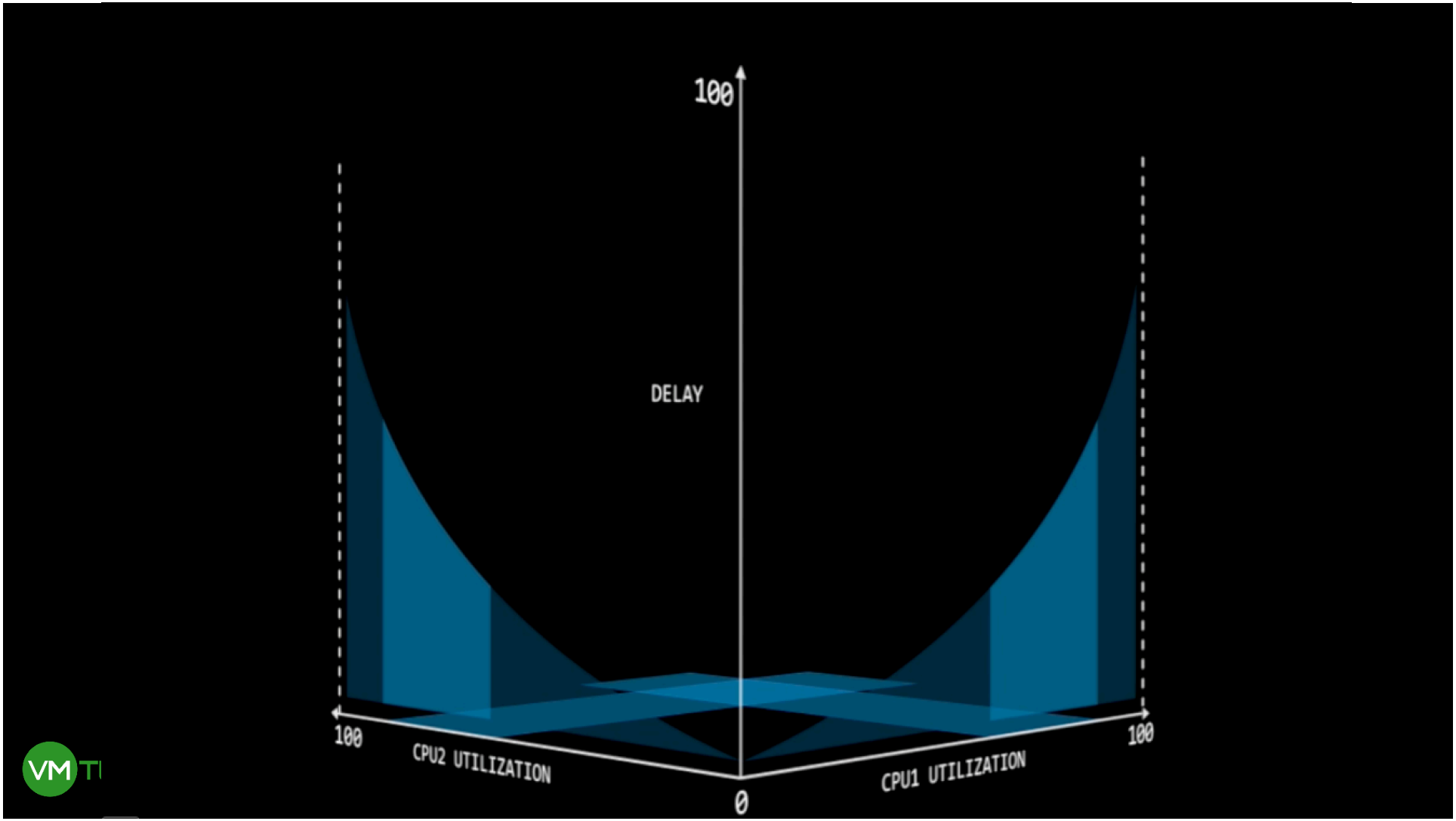


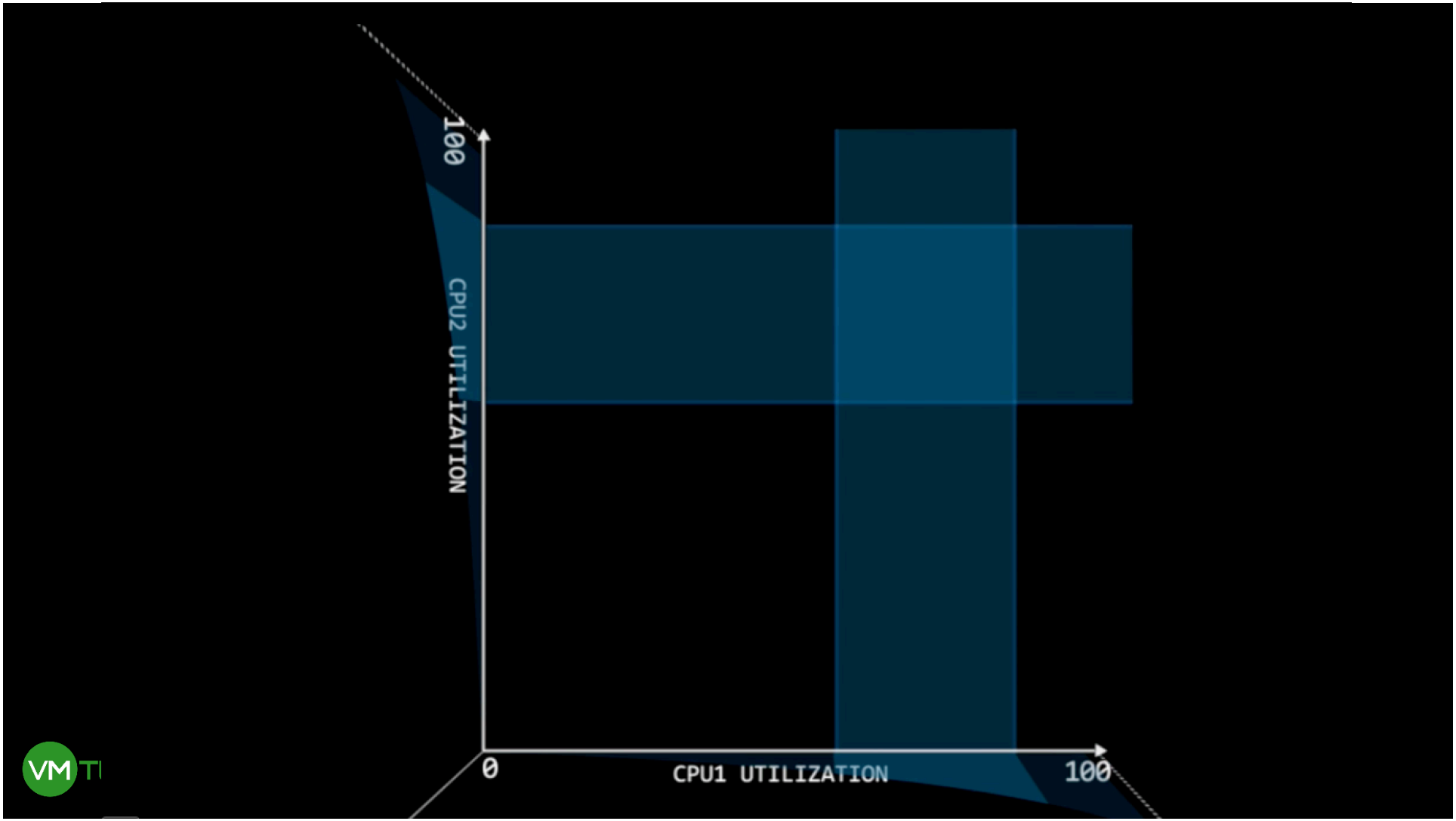
The ONLY way to assure performance is to get out of this loop

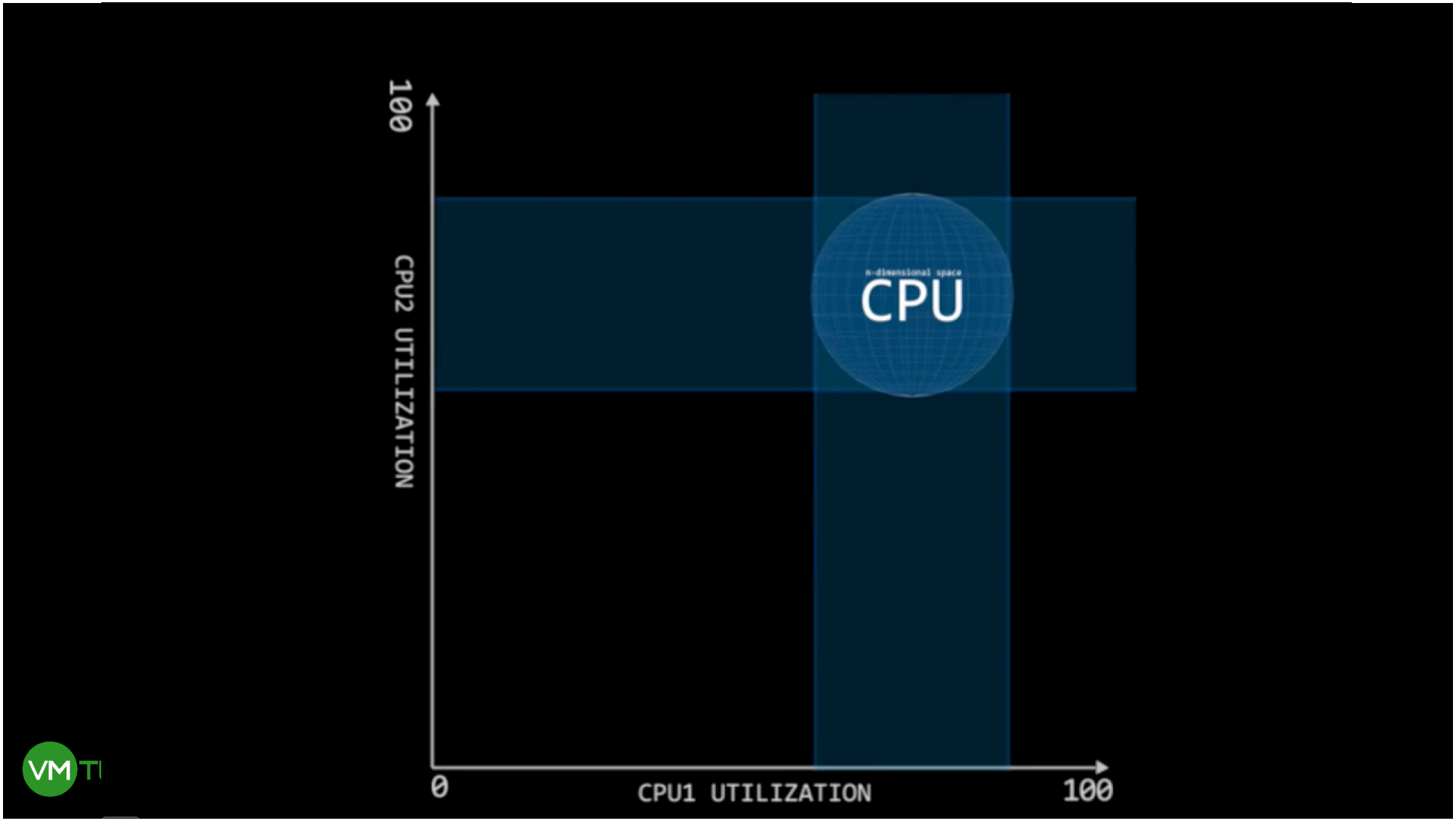
VM TURBO

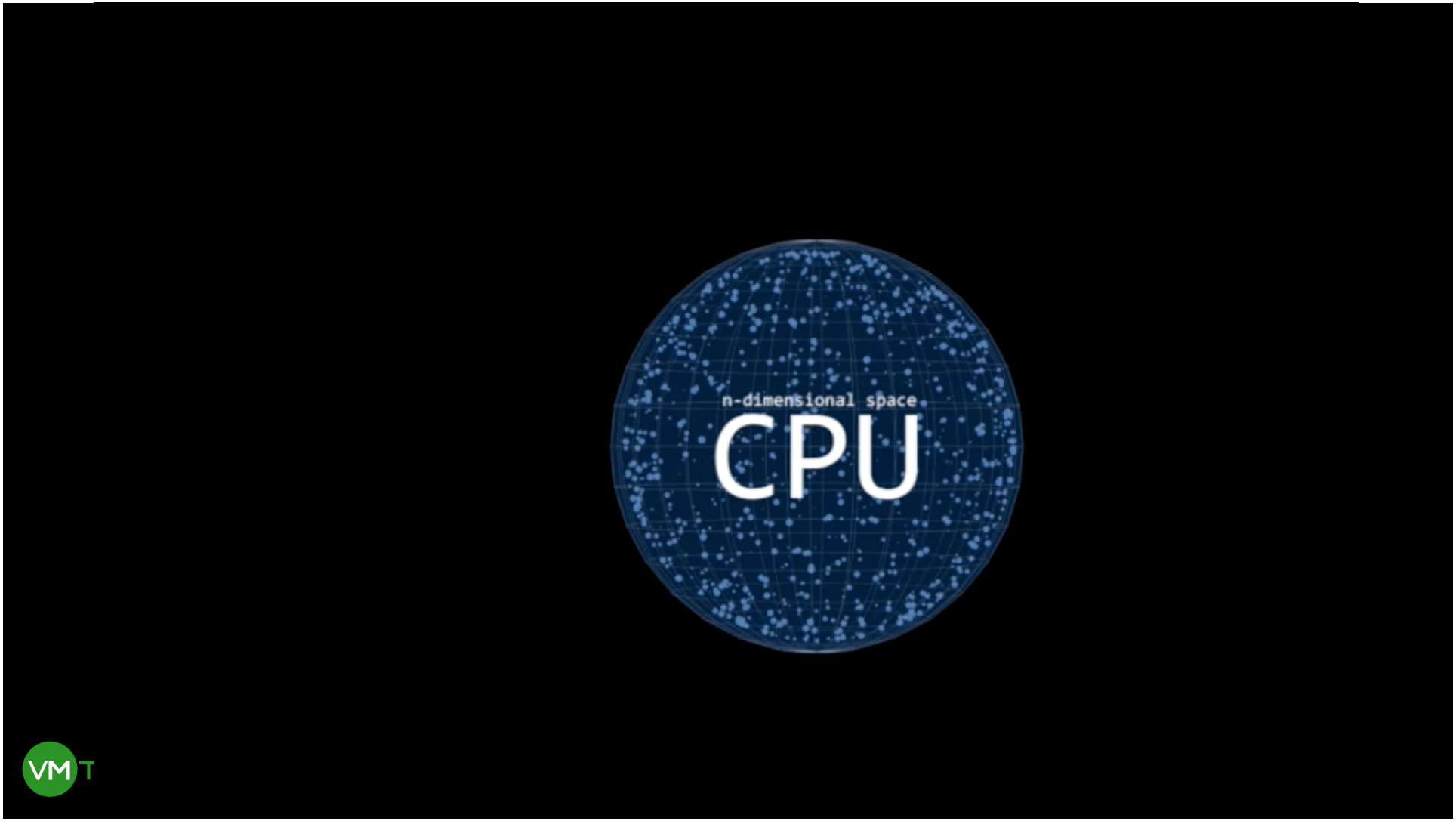

100
DELAY
0
CPU1 UTILIZATION
100
VMTURBO

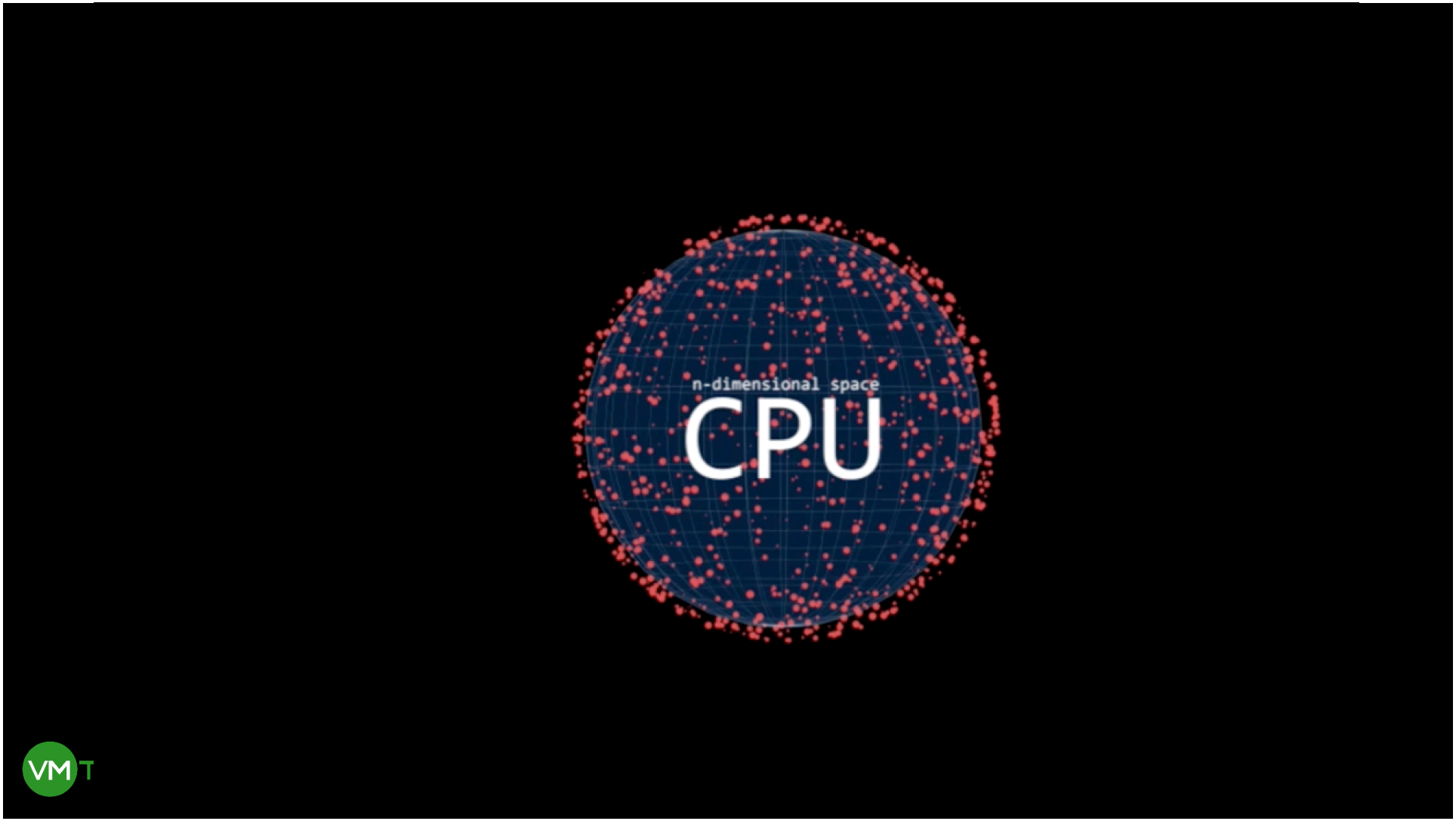

100
DELAY
100
CPU2 UTILIZATION
0
CPU1 UTILIZATION
100
VMTUR


100
DELAY
100
CPU2 UTILIZATION
0
CPU1 UTILIZATION
100

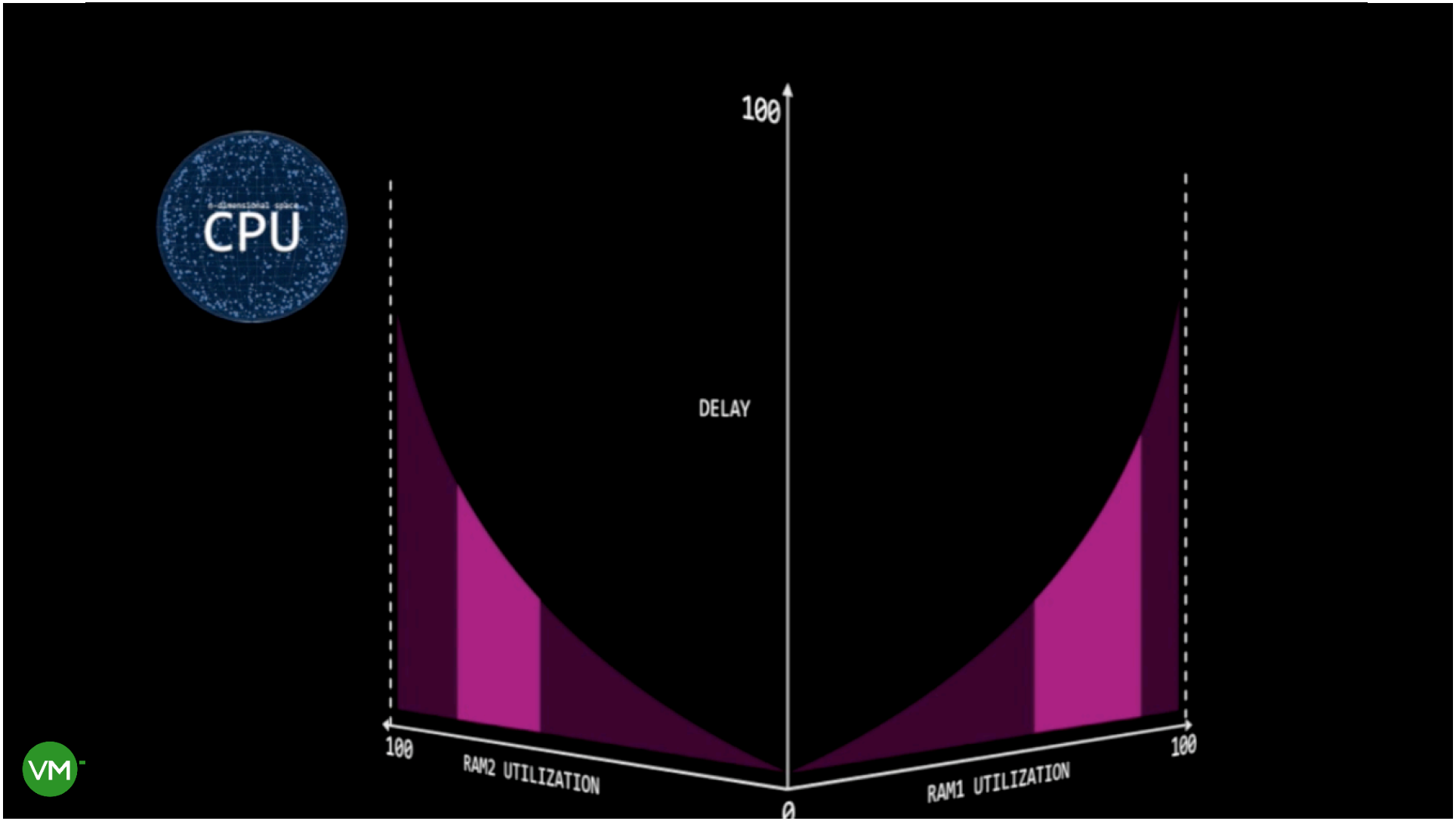

100
CPU2 UTILIZATION
0
CPU1 UTILIZATION
100

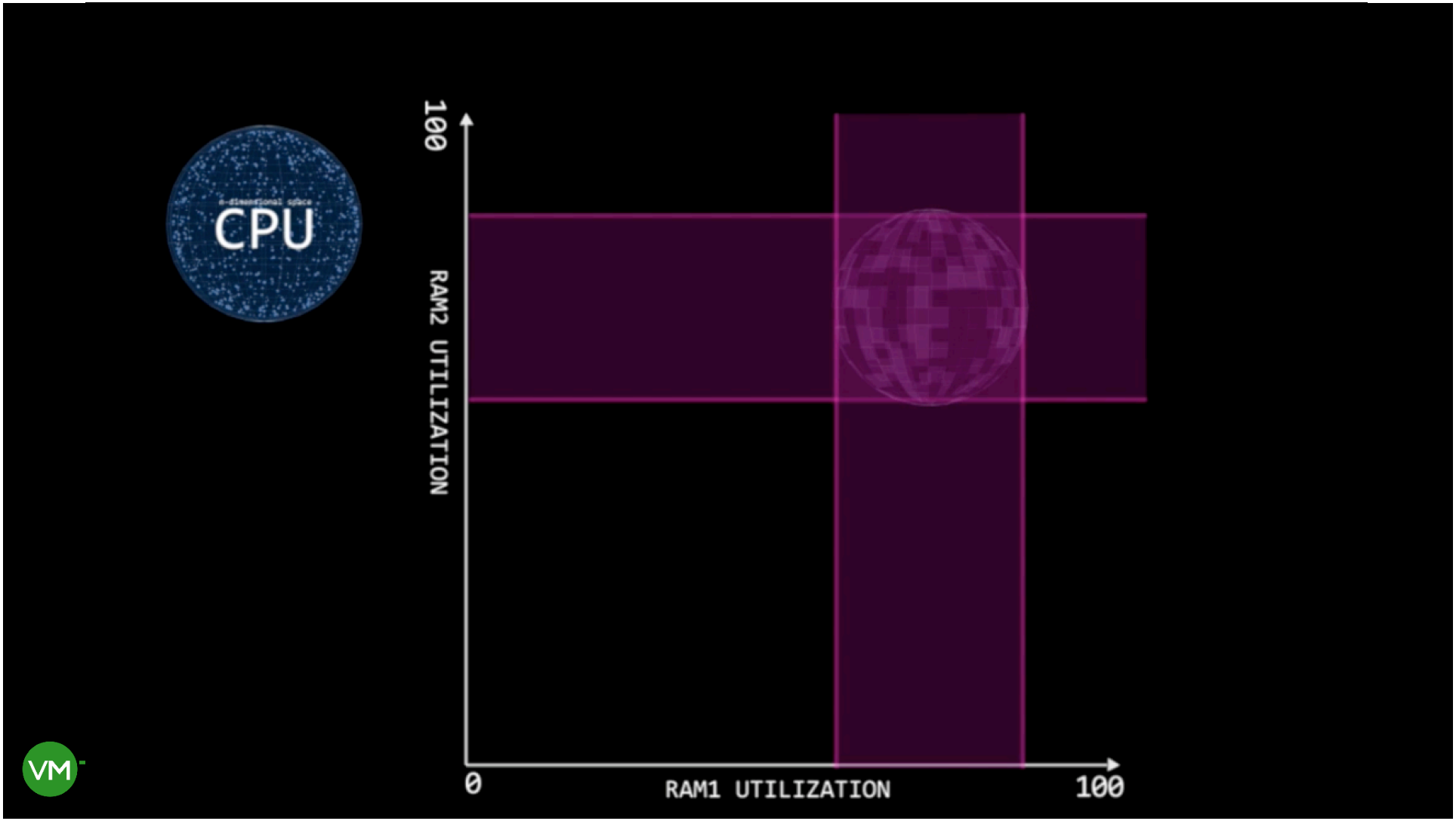

100
CPU2 UTILIZATION
n-dimensional space
CPU
0
CPU1 UTILIZATION
100

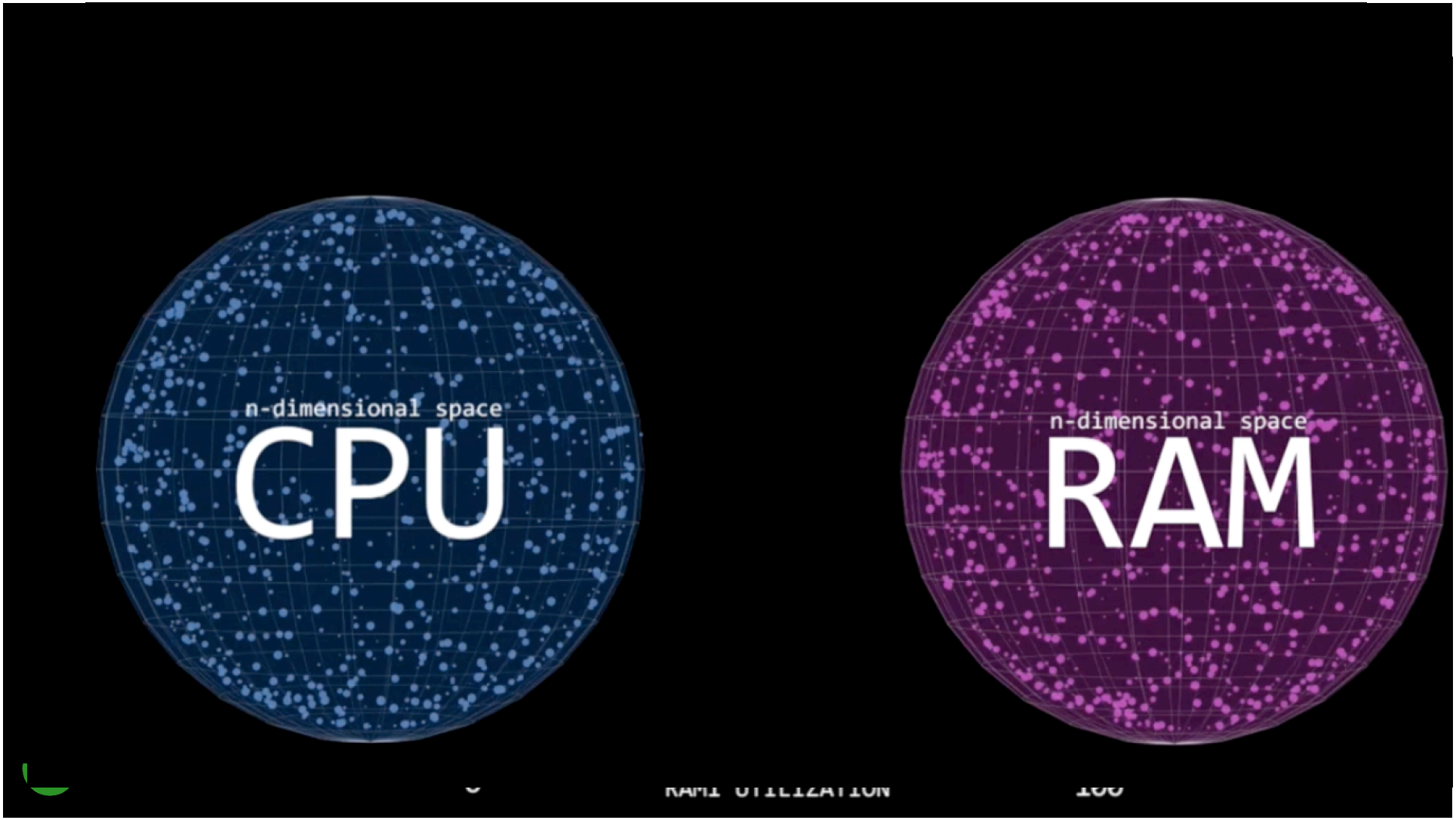

n-dimensional space
CPU

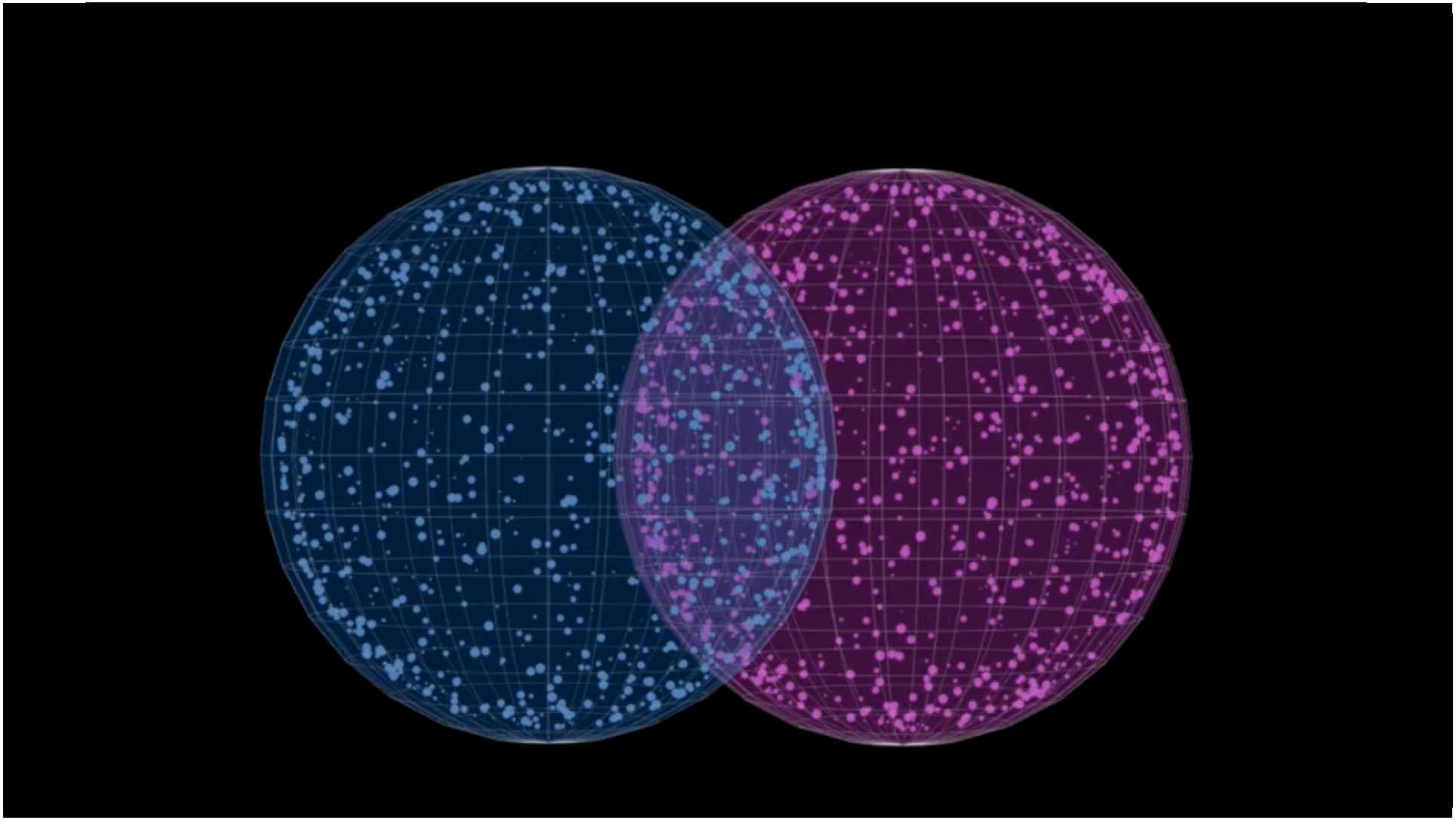

n-dimensional space
CPU

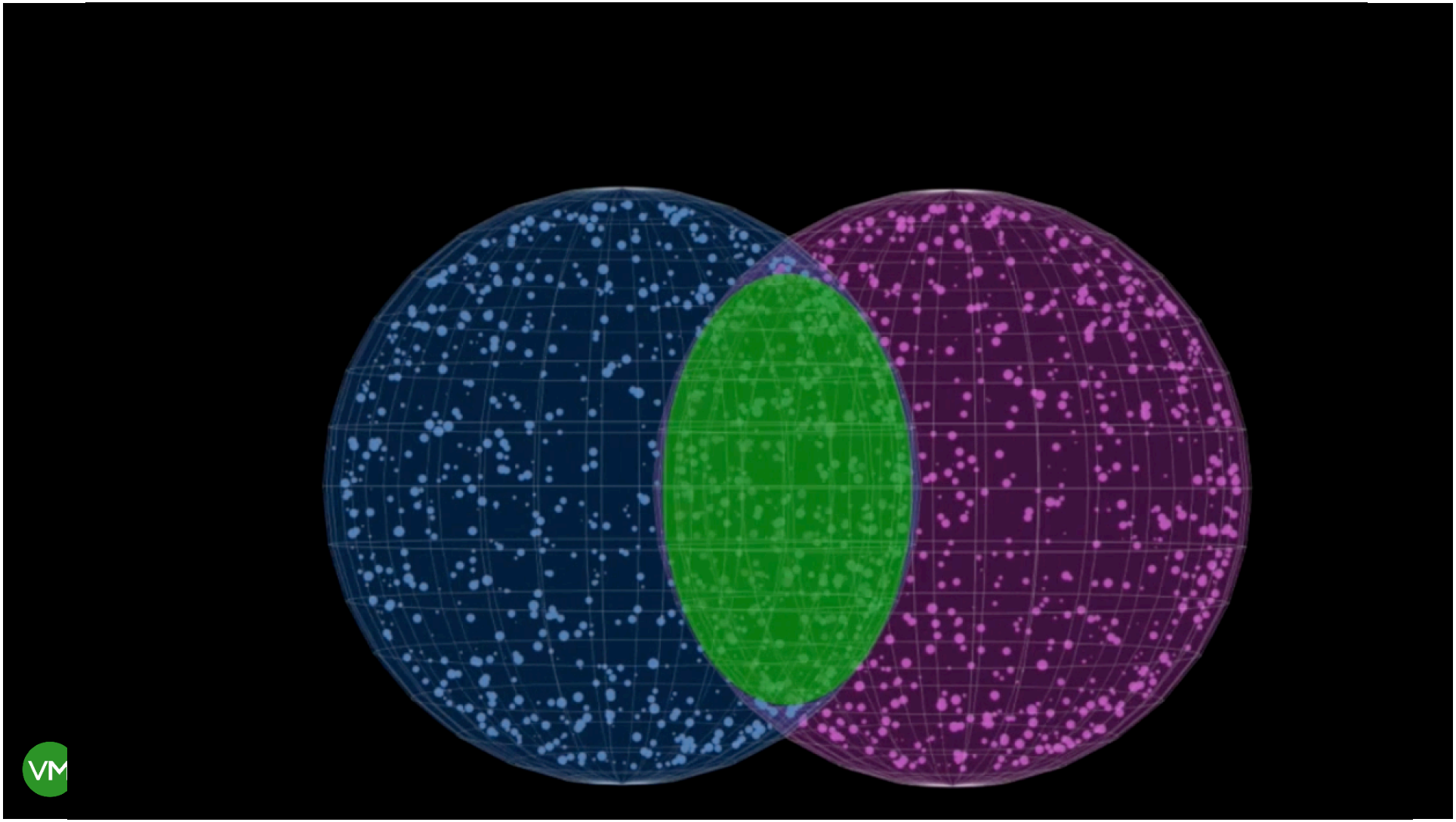

n-dimensional space
CPU

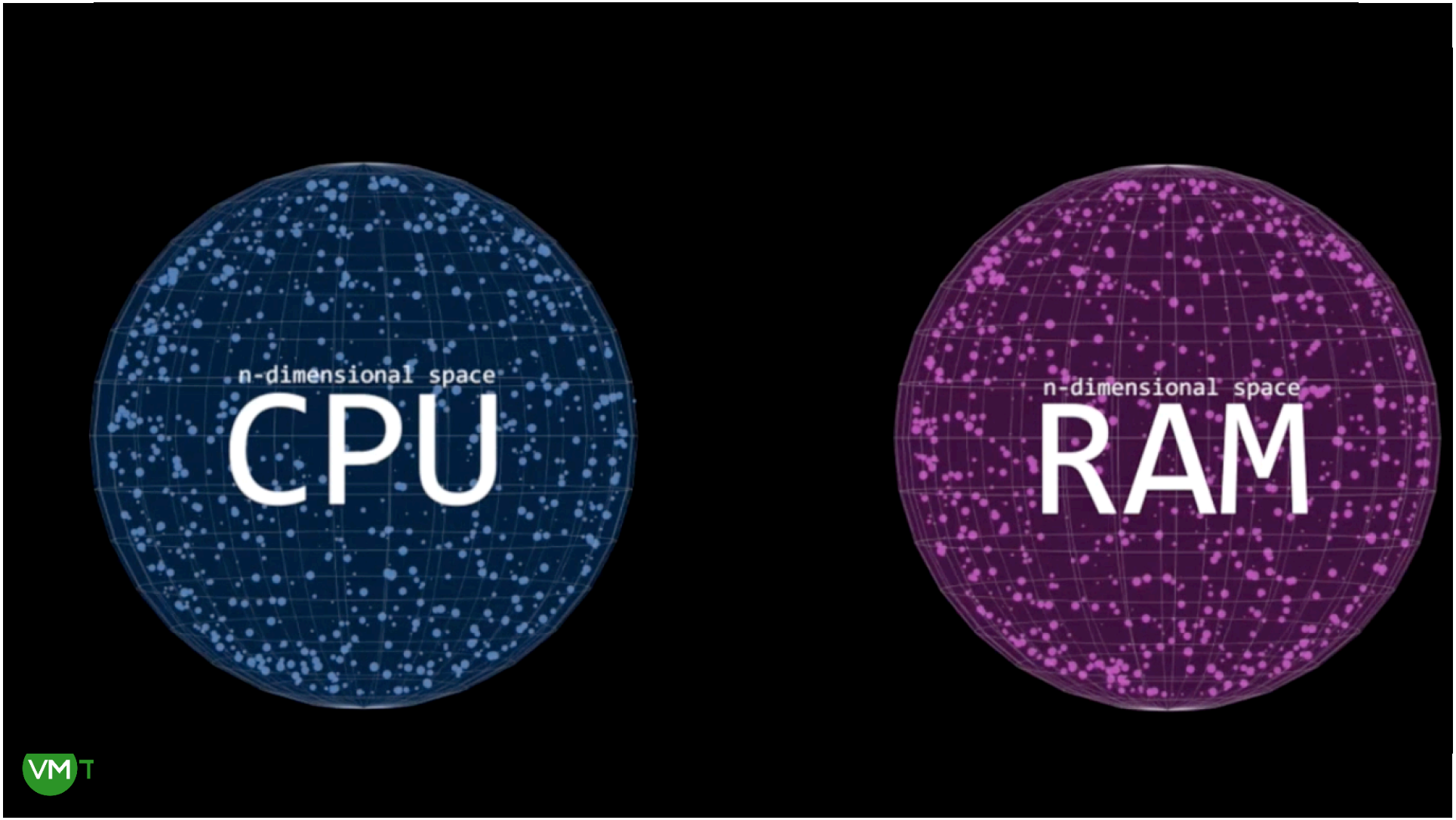

CPU
100
DELAY
100
RAM2 UTILIZATION
0
RAM1 UTILIZATION
100

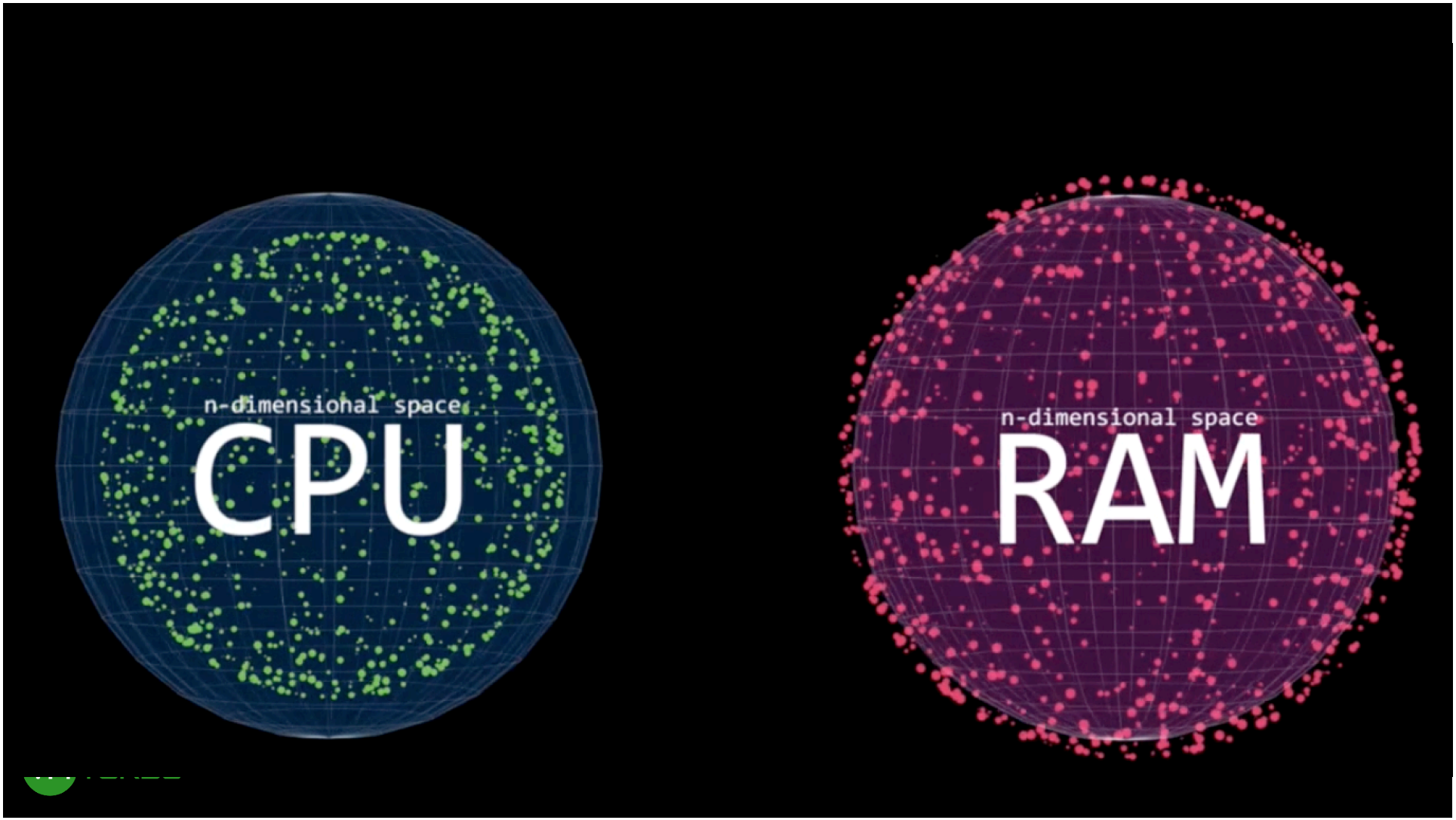

CPU
100
RAM2 UTILIZATION
0
RAM1 UTILIZATION
100

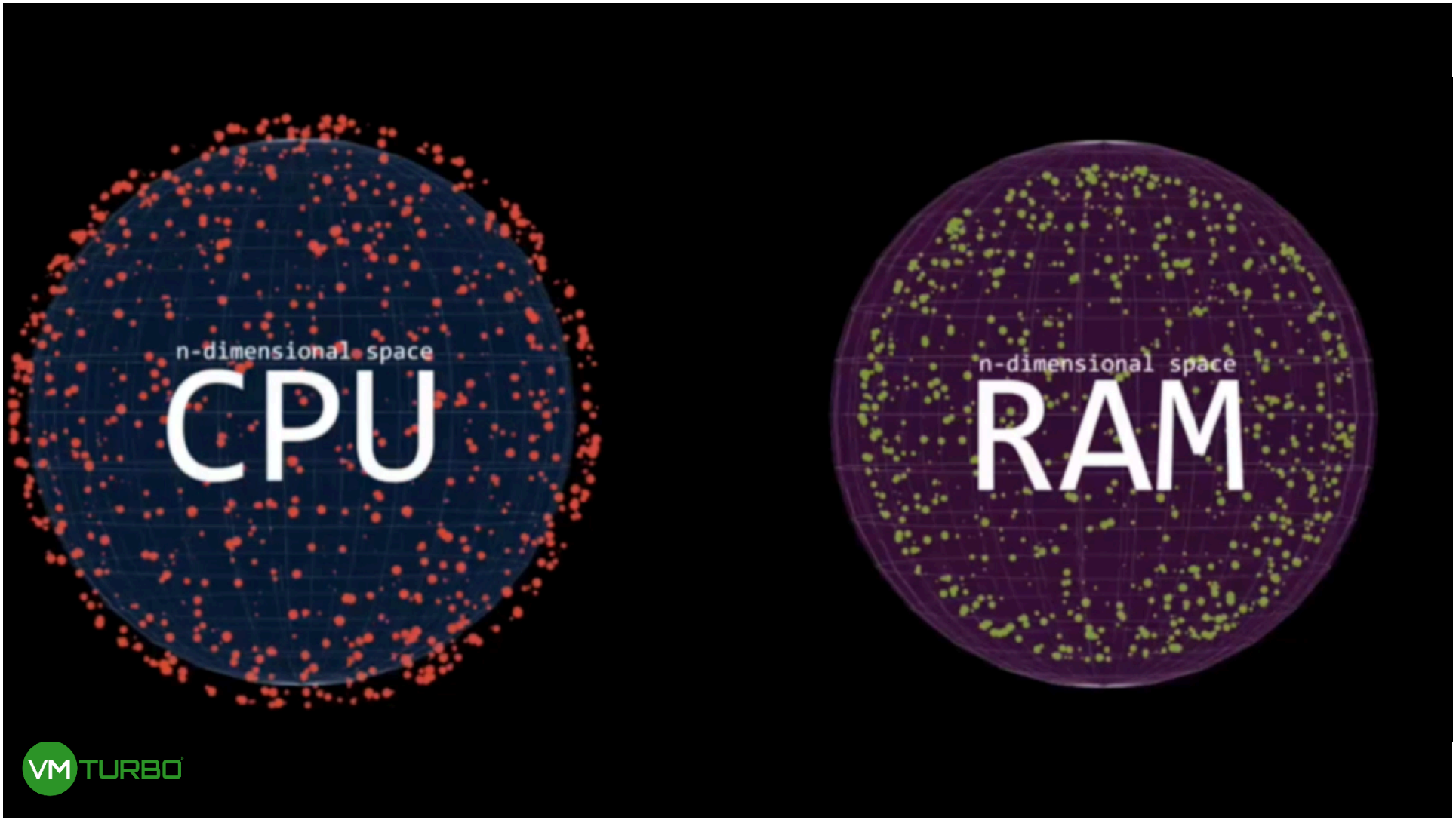

n-dimensional space
CPU
n-dimensional space
RAM



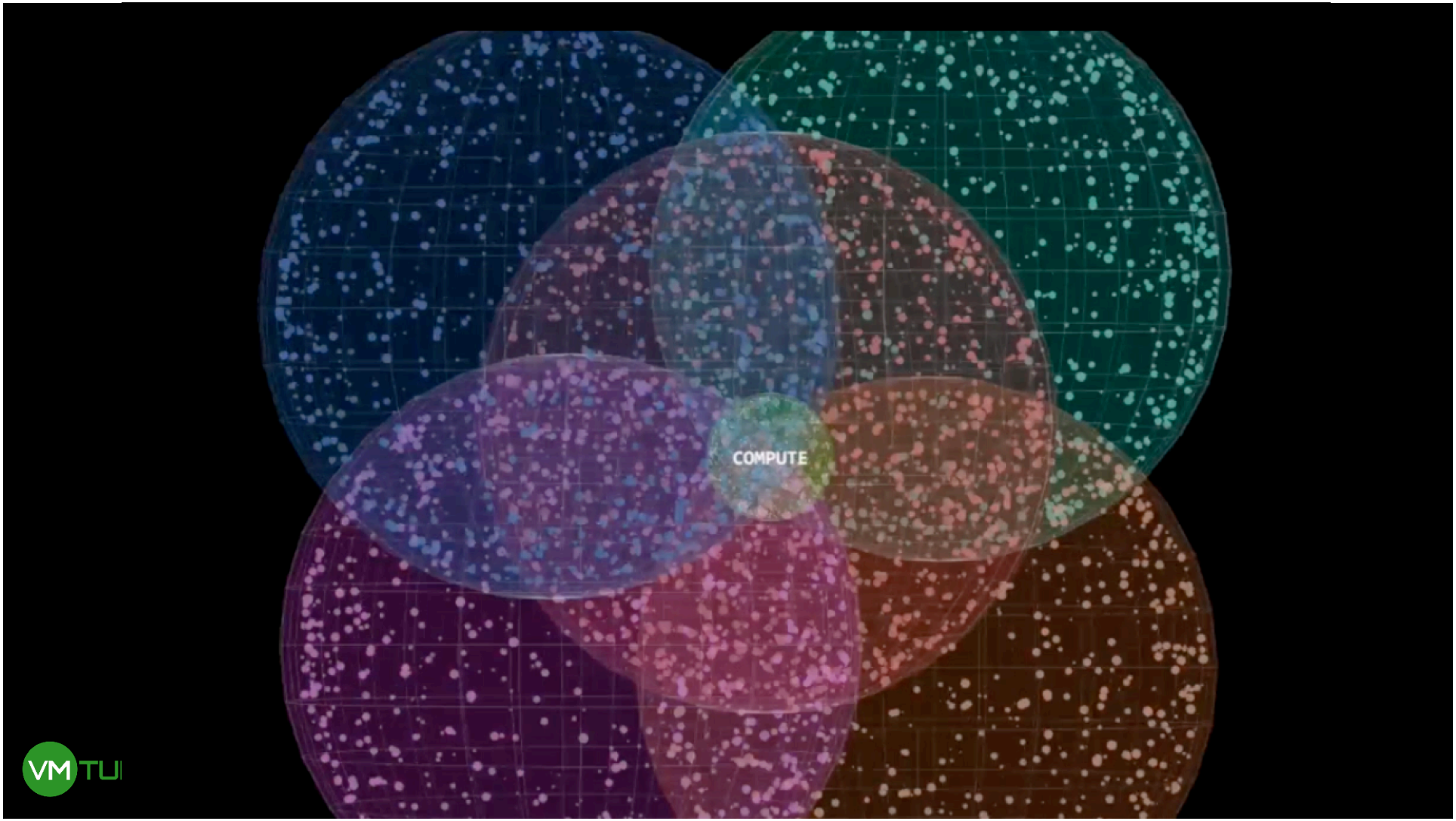


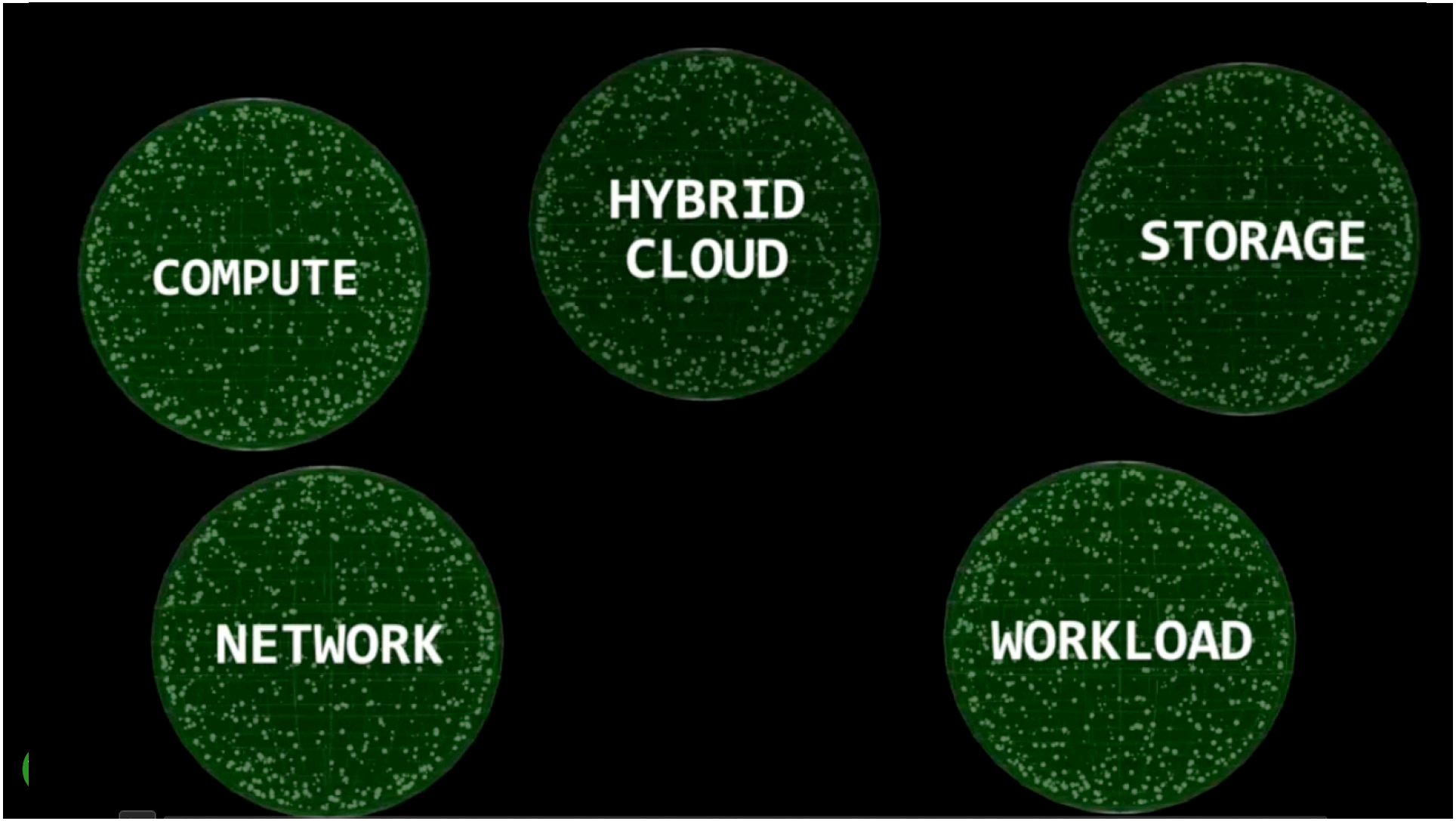

n-dimensional space
CPU
n-dimensional space
RAM
VMT


n-dimensional space
CPU
n-dimensional space
RAM

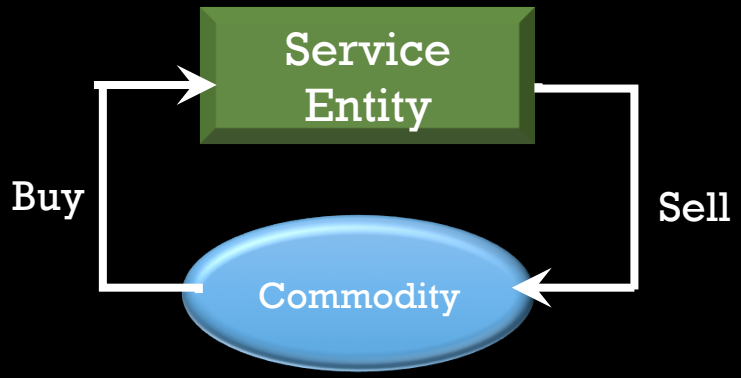

n-dimensional space
CPU
n-dimensional space
RAM
VM TURBO


n-dimensional space
CPU
n-dimensional space
IOPs
n-dimensional space
Swap
n-dimensional space
RAM
n-dimensional space
Net


COMPUTE
VMTUI

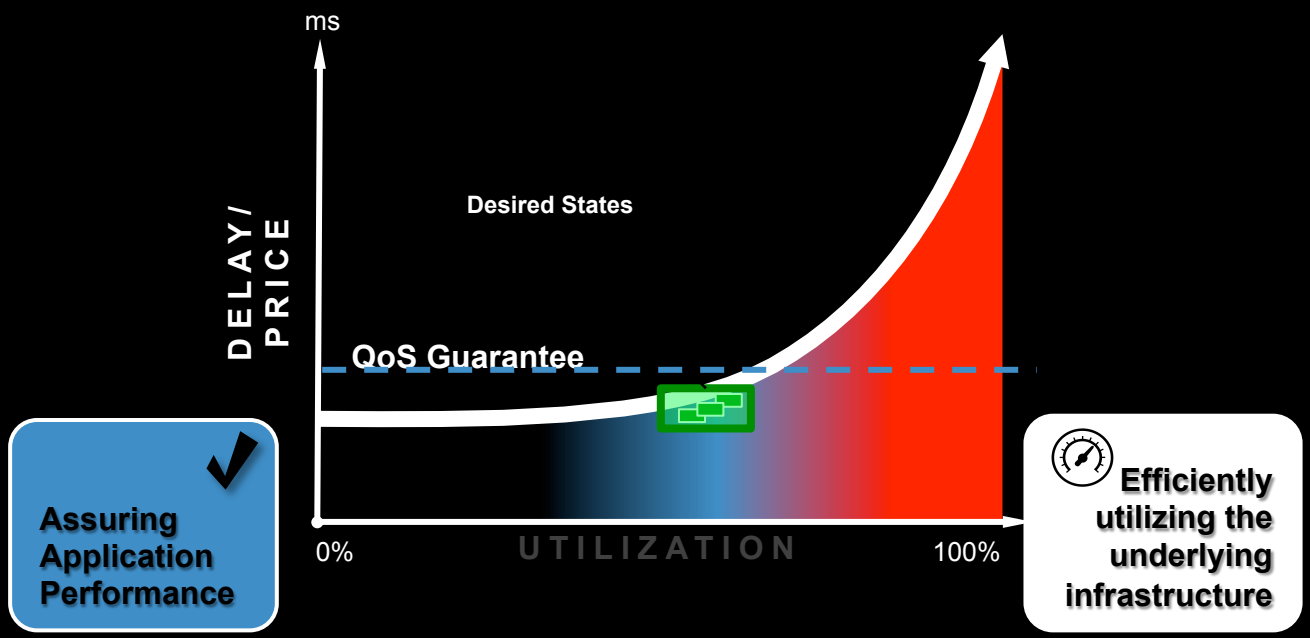

COMPUTE
HYBRID CLOUD
STORAGE
NETWORK
WORKLOAD

A Marketplace of Buyers and Sellers



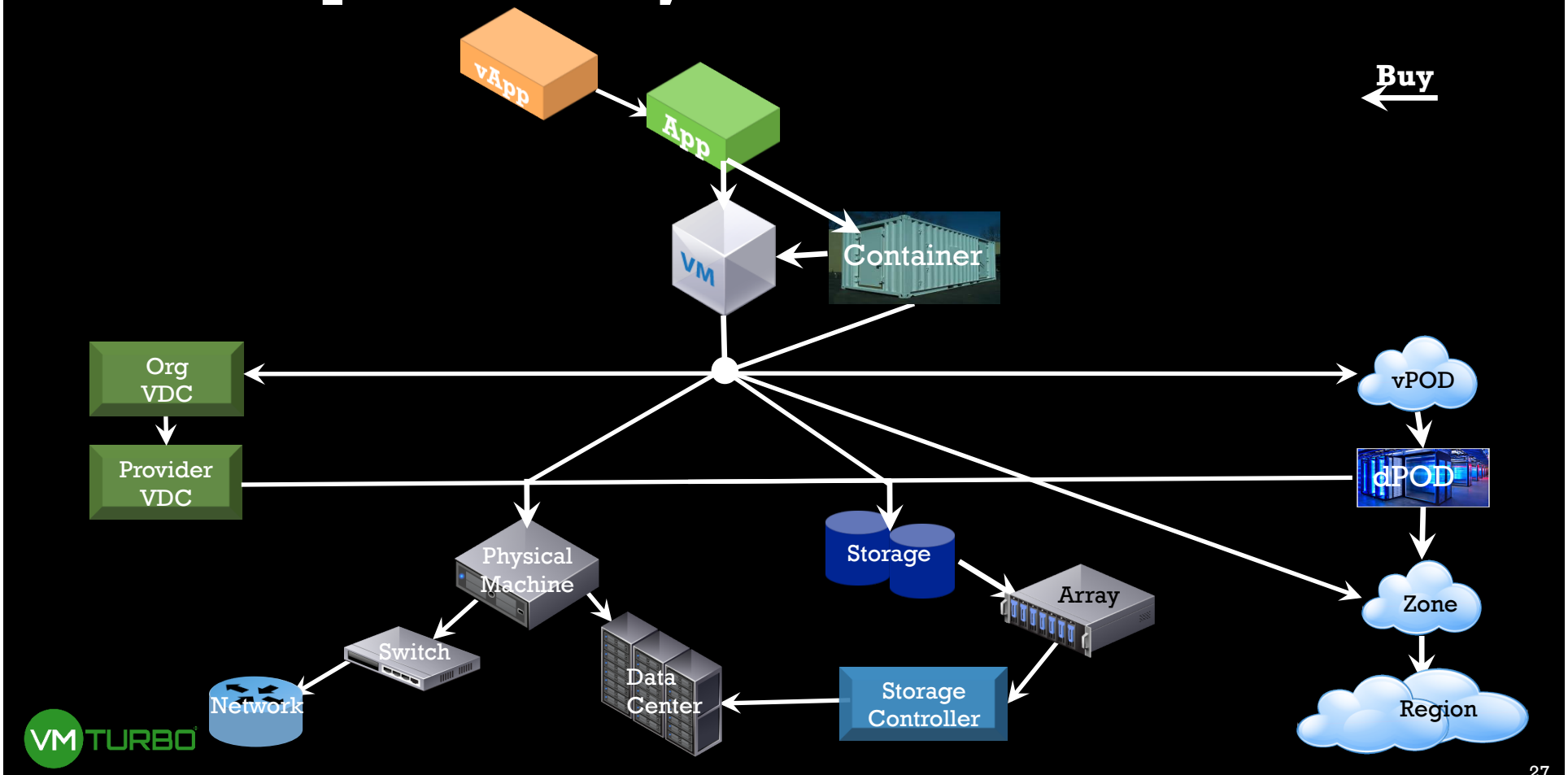
VMTURBO

A Marketplace of Buyers and Sellers

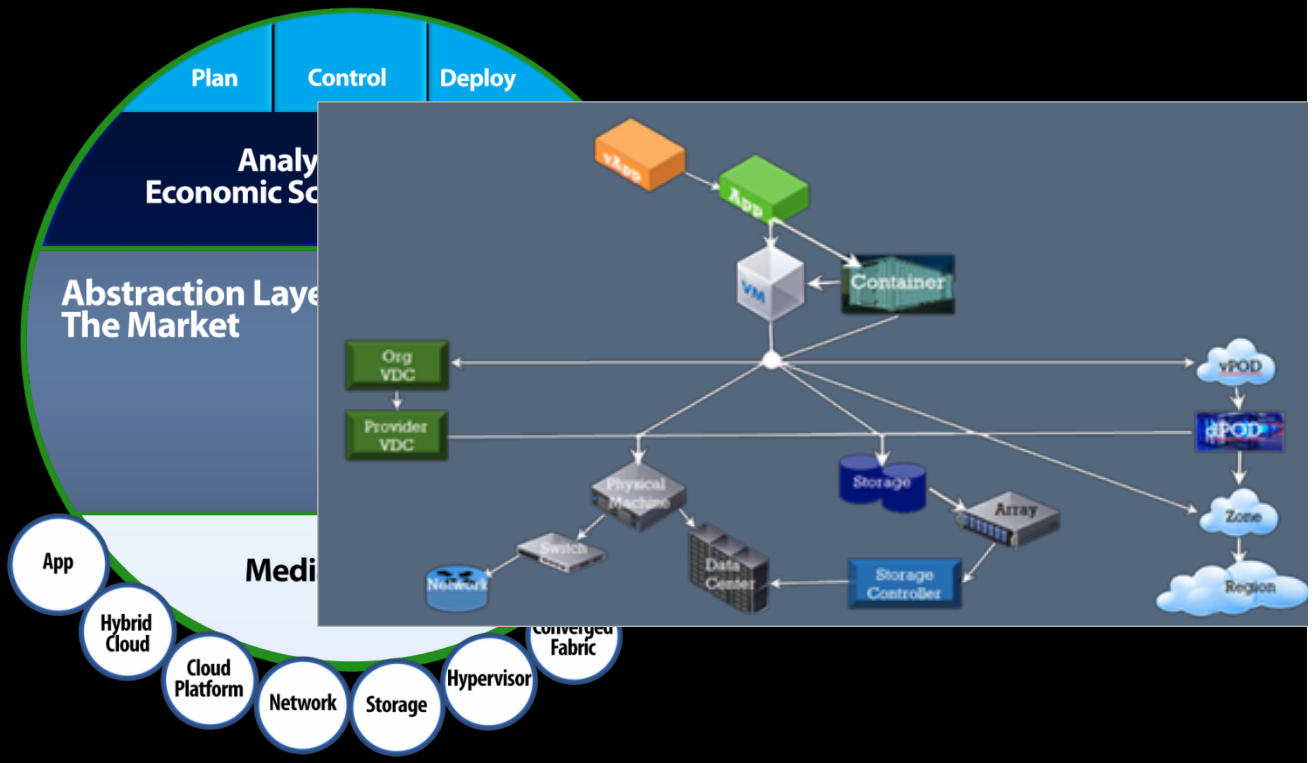


VMTURBO

A Marketplace of Buyers and Sellers

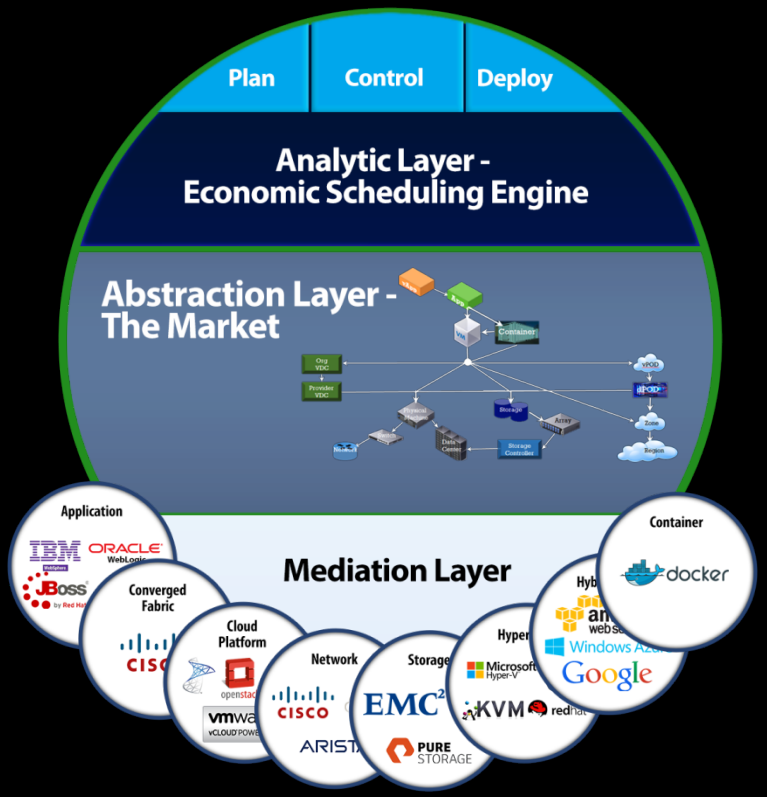


Unified Demand-Driven Control Platform



VMTURBO

VMTurbo Components…



VMTURBO

Who We Are

50+ Engineers

Expected to double in 6 months

Founded in 2009

VM TURBO

Who We Are

Geographically Distributed

VMTURBO

The Monolith

VM TURBO

MonolithFirst

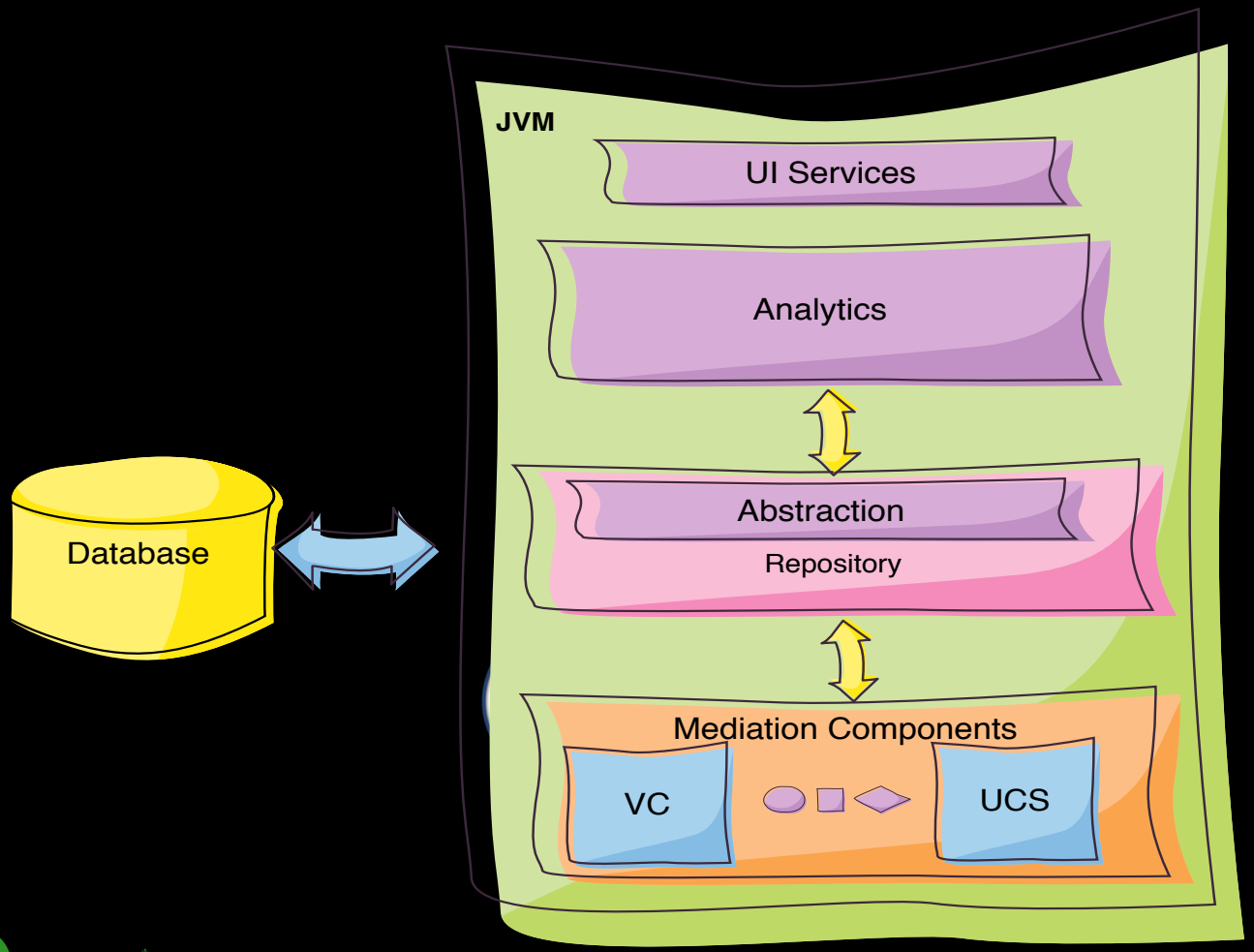
Martin Fowler
3 June 2015

Going directly to a microservices architecture is risky

A monolith allows you to explore both the complexity of a system and its component boundaries

As complexity rises start breaking out some microservices

Continue breaking out services as your knowledge of boundaries and service management increases



JVM (Tomcat)

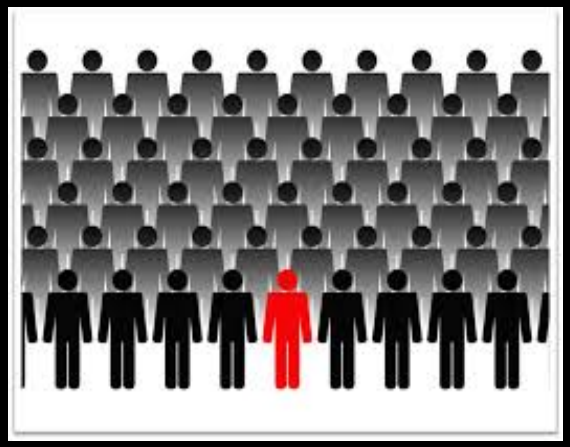
~~Eclipse Modeling Framework~~

~~Home Grown Repository~~

~~MySQL~~

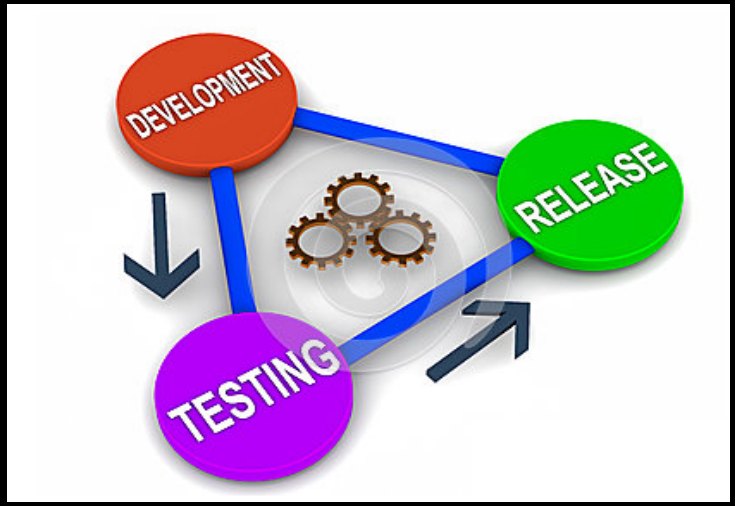
VMTURBO

Team Structure



Monolithic

Release Cycle



Major Release Every 6 Months

Interim Patches

VMTURBO

Metrics?

VMTURBO

Summary of Initial State

- Monolithic Architecture
- Monolithic Team Structure
- Release Every Six Months
- No Metrics Captured

VM TURBO

Pain Points

- Monolithic Architecture
- Monolithic Team Structure
- Release Every Six Months
- No Metrics Captured

VM TURBO

Pain Points

- Monolithic Architecture
  - Scalability Issues
  - Concurrency Issues
  - Tangled Interfaces between Components

VM TURBO

Pain Points

- Monolithic Team Structure
  - Divided Focus limits Team Velocity
  - Customer Issues vs. New features

VM TURBO

Catalysts for Change

Growth in Customer Base – to over 1000

More Large Environments – up to 75K VMs under management

Geographical Spread of Team – US / Canada / Italy / Greece / Russia

More Frequent Deliveries – Semi-Monthly vs. Every 6 mos

Expanding Feature Base – Across the Datacenter

VM TURBO

Architectural Principles

Design by Interface When designing a component, focus on the behavior of the component. Users of components are aware of interfaces only. No concrete classes are known to the user of the components.

VM TURBO

Architectural Principles

Separation of Concerns A component should only implement the business function it is mandated to cover and not be concerned with other issues. For example, an Analysis component contains code to compute action items. There should be no code related to mediation or databases. These concerns are handled outside of the Analysis component.

VM TURBO

Architectural Principles

Inversion of Control (IoC)
If component A depends on component B (e.g., ESE depends on a Pricing Function), B is provided to A rather than A being coded with the way to find B. This results in concise and configurable components. IoC is a function of the Micro Container.

VM TURBO

Architectural Principles

Code Instrumentation
Components will need to be instrumented in order to be managed (Stopped, Started, Paused, Resumed, Inspected) at runtime.

VM TURBO

Architectural Principles

Façades
VMTurbo cannot dictate its future technology choices but rather endeavors to easily adapt to them. Everything in VMTurbo should be replaceable by another implementation that conforms to the prescribed set of interfaces.

VMTURBO

Architectural Principles

Well-formed Components
Well-formed components are those that adhere to the principles listed above. In addition, well-formed components are delivered with built-in, non-regression functional and performance tests. In addition, standard documentation is packaged with components and may be browsed once deployed.

.

VMTURBO

Organizational Principles

Small Teams

Decomposing the monolith affects teams as well. Small teams operating in an agile fashion is essential.

Separation of Concerns

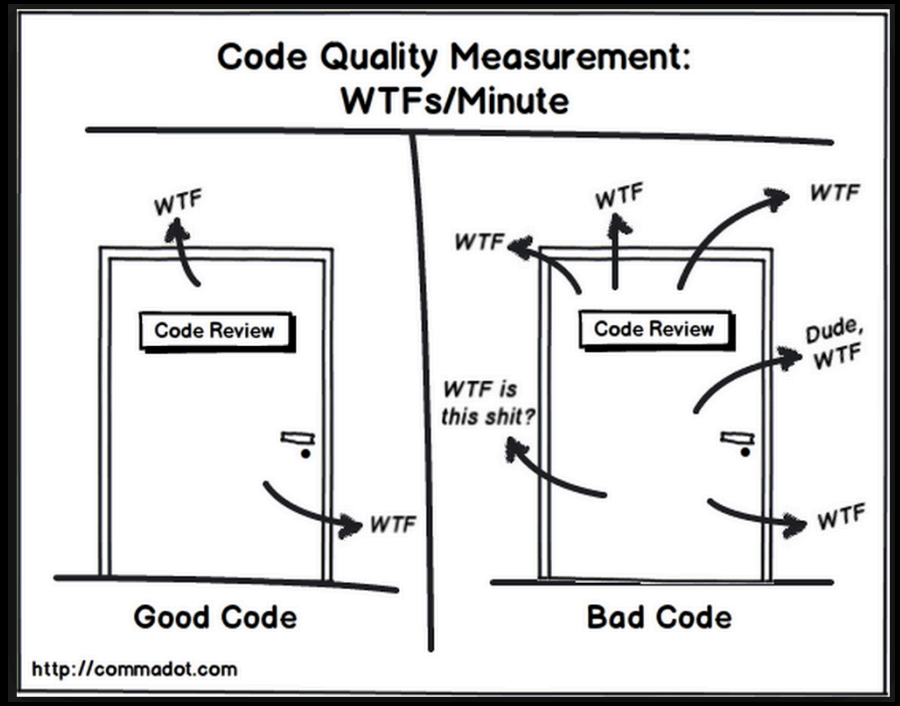
.

VM TURBO

Organizational Principles for Microservices

Peer Review

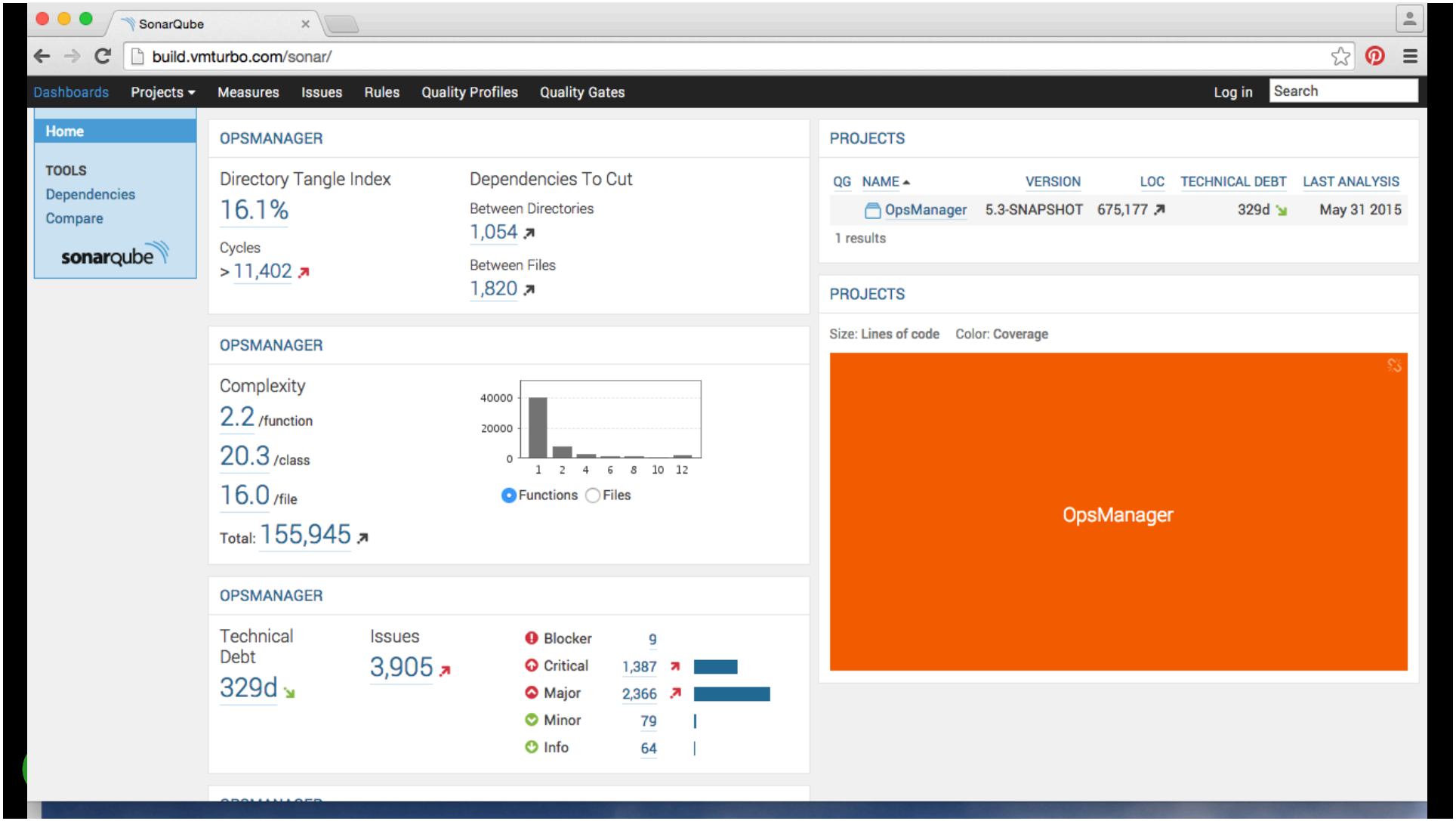
.



Organizational Principles for Microservices

Metrics, Metrics and More Metrics

Testing, testing and more Testing


SonarQube
build.vmturbo.com/sonar/
Dashboards
Projects
Measures
Issues
Rules
Quality Profiles
Quality Gates
Log in
Search
Home
TOOLS
Dependencies
Compare
sonarqube
OPSMANAGER
Directory Tangle Index
16.1%
Cycles
> 11,402
Dependencies To Cut
Between Directories
1,054
Between Files
1,820
OPSMANAGER
Complexity
2.2 /function
20.3 /class
16.0 /file
Total: 155,945
40000
20000
0
1 2 4 6 8 10 12
Functions
Files
OPSMANAGER
Technical Debt
329d
Issues
3,905
Blocker 9
Critical 1,387
Major 2,366
Minor 79
Info 64
PROJECTS
QG NAME VERSION LOC TECHNICAL DEBT LAST ANALYSIS
OpsManager 5.3-SNAPSHOT 675,177 329d May 31 2015
1 results
PROJECTS
Size: Lines of code Color: Coverage
OpsManager

Evolution Phase 1

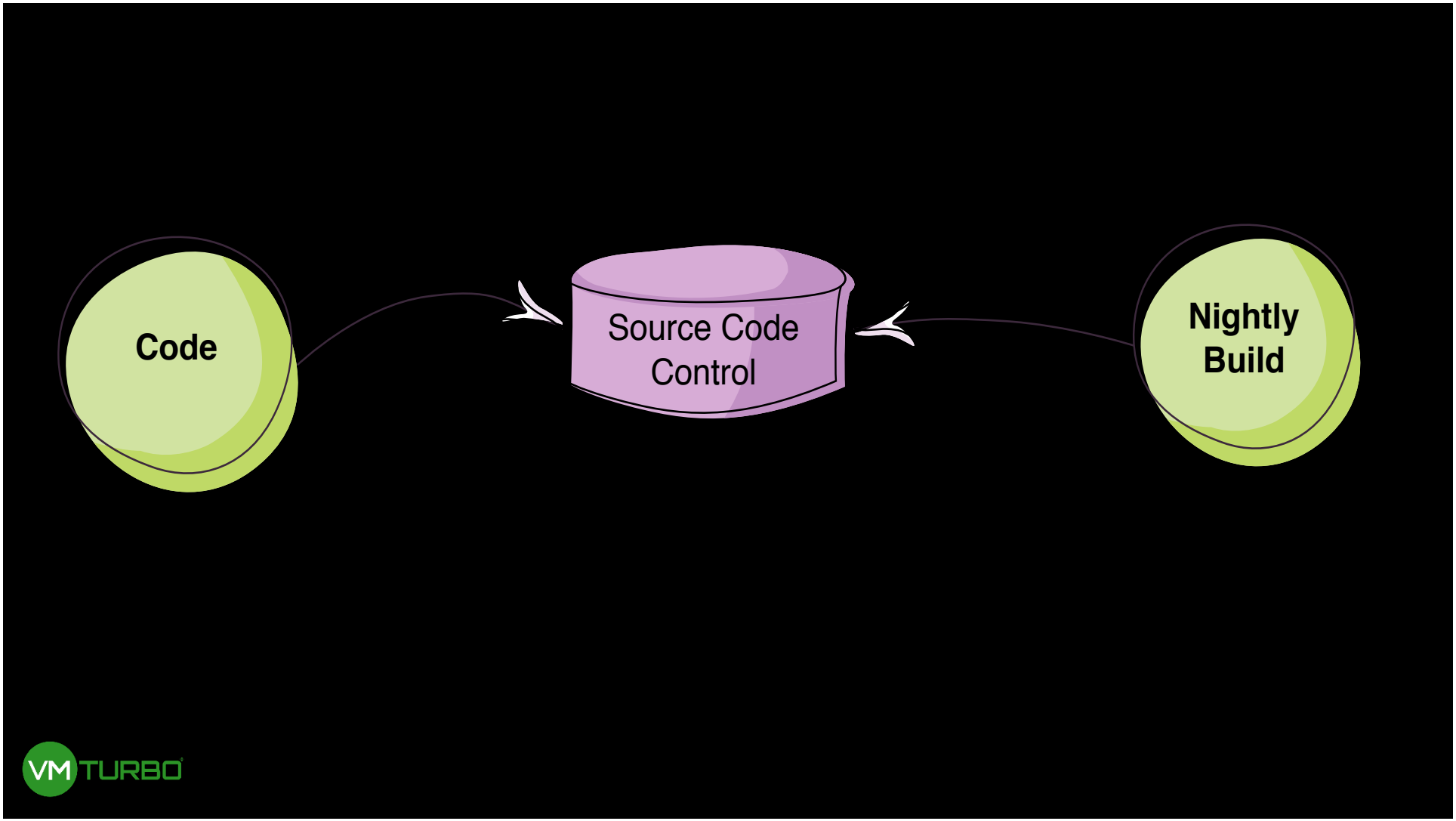
Decomposing the Monolith

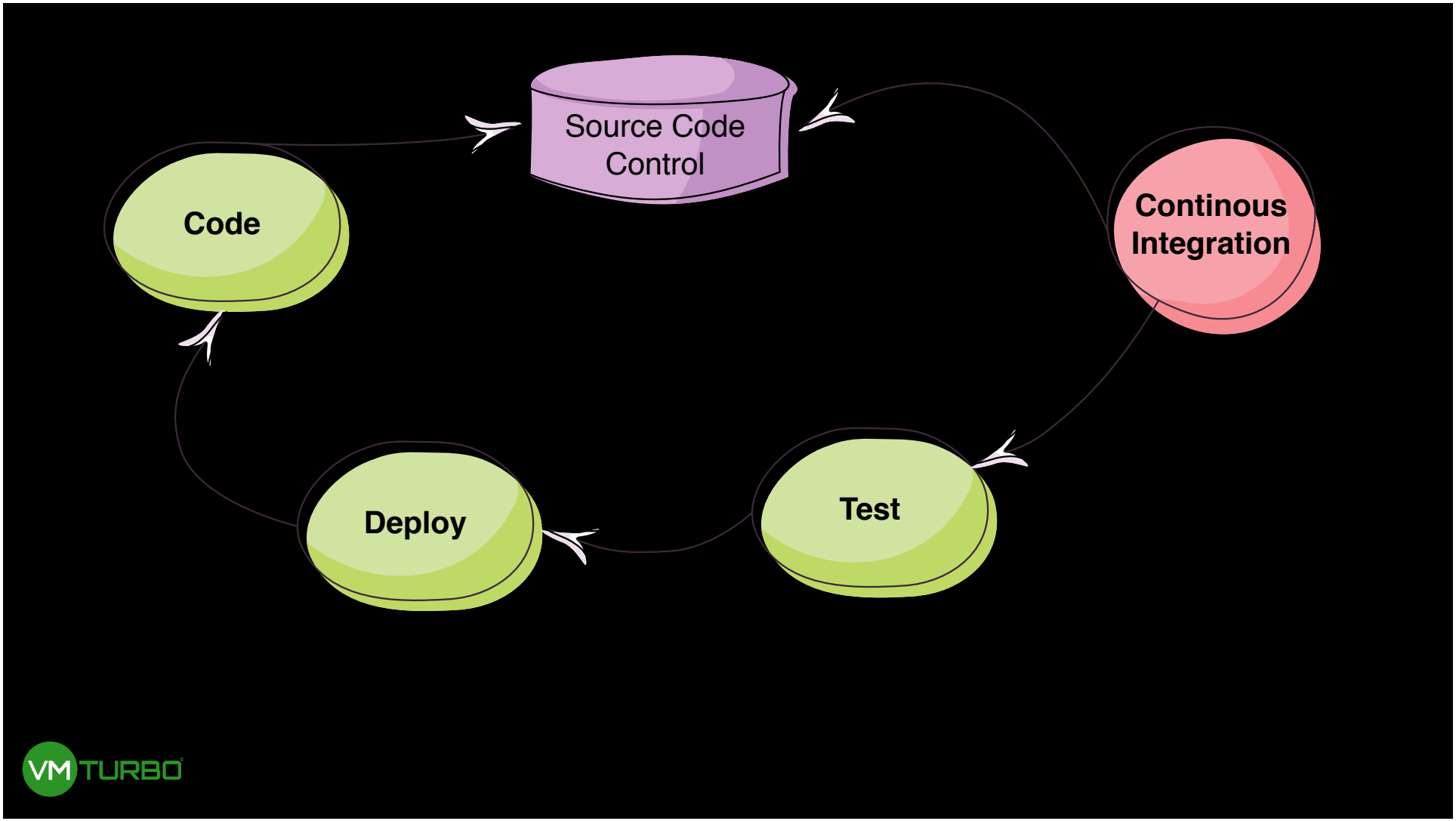
The Journey Starts

VM TURBO

Continuous Integration

VM TURBO


Code
Source Code Control
Nightly Build
VMTURBO


Source Code Control
Code
Continous Integration
Deploy
Test
VMTURBO

Metrics

VM TURBO

- %Code Coverage
- %Documentation of Public APIs
- Performance Metrics
- Find Bugs Issues

VM TURBO

Mediation Microservices

The First VMTurbo Microservice

VMTURBO

Architectural Principles

Separation of Concerns A component should only implement the business function it is mandated to cover and not be concerned with other issues. For example, an Analytics component contains code to compute action items. There should be no code related to mediation or databases. These concerns are handled outside of the ESE component.

VM TURBO

Separate Analysis and Mediation Models

After
Mediation Model == Projection of The Analysis Model

VM TURBO

Architectural Principles

Design by Interface When designing a component, focus on the behavior of the component. Users of components are aware of interfaces only. No concrete classes are known to the user of the components.

VM TURBO

Mediation Behaviors and Interface to Analysis

- Initial Population VMTurbo Abstraction
- Updating of Supply and Demand data.
- Updating of Entity Data
- Execute Control Actions Identified by Analysis

VMTURBO

Before Decomposition

Functions	Number of Ways
Model Instantiation	dozens
Update Supply and Demand Data	Even more
Update Entity Data	Even more than that

VM TURBO

```
1360        StorageAmount storageCommodity = (StorageAmount) commoditySold(da,AbstractionPackage.eINSTANCE.getStorageAmount(),STORAGE_ACCESS_PREFIX+da.getName());
1361        NetAppDADiscExt storageExt = (NetAppDADiscExt) storageCommodity.createExtension(DiscoveryExtensionsPackage.eINSTANCE.getNetAppDADiscExt());
1362        setExtProps(storageExt,target,daExt.getDisplayName(),daExt.getLocalName(),STORAGE_PREFIX+da.getUuid(), logPrefix);
1363
1364        // If Hybrid, Storage Amount of the HDDs alone considered
1365        if((aggr.isHybridEnabled() != null && aggr.isHybridEnabled()) && (aggr.isHybrid() !=null && aggr.isHybrid()) && aggregateSSDSizes.get(aggrName) != null)
1366            storageCommodity.setCapacity(aggr.getSizeTotal().floatValue()/Mega - aggregateSSDSizes.get(aggrName));
1367        else
1368            storageCommodity.setCapacity(aggr.getSizeTotal().floatValue()/Mega);
1369
1370        // Create StorageProvisioned Commodity
1371        StorageProvisioned stprovCommodity = (StorageProvisioned) commoditySold(da,
1372                AbstractionPackage.eINSTANCE.getStorageProvisioned(),STORAGE_PROVISIONED_PREFIX+da.getName());
1373        NetAppDADiscExt stproveExt = (NetAppDADiscExt) stprovCommodity.createExtension(DiscoveryExtensionsPackage.eINSTANCE.getNetAppDADiscExt());
1374        setExtProps(stproveExt,target,daExt.getDisplayName(),daExt.getLocalName(),STORAGE_PROVISIONED_PREFIX+da.getUuid(), logPrefix);
1375        stprovCommodity.setCapacity(aggr.getSizeTotal().floatValue()/Mega * ((Double)StorageSettingsManagerImpl.vmtMANAGER.getSetting(da,
1376                AnalysisPackage.eINSTANCE.getStorageSettingsManager_Capacity_DAProvisioned())).floatValue()/100);
1377
1378        // Create StorageAccess (IOPS) commodity
1379        StorageAccess stAccess = (StorageAccess) commoditySold(da,AbstractionPackage.eINSTANCE.getStorageAccess(),STORAGE_PREFIX+da.getName());
1380        NetAppDADiscExt stExt = (NetAppDADiscExt) stAccess.createExtension(DiscoveryExtensionsPackage.eINSTANCE.getNetAppDADiscExt());
1381        setExtProps(stExt,target,daExt.getDisplayName(),daExt.getLocalName(),STORAGE_PREFIX+da.getUuid(), logPrefix);
1382        stAccess.setCapacity(iopsCapacity);
1383
1384        // Create StorageLatency (LAT) commodity
1385        StorageLatency stLatency = (StorageLatency) commoditySold(da,AbstractionPackage.eINSTANCE.getStorageLatency(),STORAGE_LATENCY_PREFIX+da.getName());
1386        NetAppDADiscExt latExt = (NetAppDADiscExt) stLatency.createExtension(DiscoveryExtensionsPackage.eINSTANCE.getNetAppDADiscExt());
1387        setExtProps(latExt,target,daExt.getDisplayName(),daExt.getLocalName(),STORAGE_LATENCY_PREFIX+da.getUuid(), logPrefix);
1388
1389        // Create the bought commodities and associate with the underlying controller
1390        if (sc != null) {
1391            CPU cpuBought = (CPU) commodityBought(da,AbstractionPackage.eINSTANCE.getCPU(),CPU_PREFIX+da.getName()+"_"+sc.getName());
1392            NetAppDADiscExt cpuExt = (NetAppDADiscExt) cpuBought.createExtension(DiscoveryExtensionsPackage.eINSTANCE.getNetAppDADiscExt());
1393            setExtProps(cpuExt,target,daExt.getDisplayName(),daExt.getLocalName(),CPU_PREFIX+sc.getUuid(), logPrefix);
1394            Commodity cpuSold = sc.getCommoditiesMap().get(CommodityType.fetch(AbstractionPackage.eINSTANCE.getCPU(), null));
1395            cpuSold.getConsumedBy().add(cpuBought);
1396
1397            String saKey = CommodityImpl.dynCommKey(sc);
1398            StorageAmount stAmountBought = (StorageAmount) da.buyDynamicCommodity(AbstractionPackage.eINSTANCE.getStorageAmount(), saKey, 1);
1399            NetAppDADiscExt stAmountExt = (NetAppDADiscExt) stAmountBought.createExtension(DiscoveryExtensionsPackage.eINSTANCE.getNetAppDADiscExt());
1400            setExtProps(stAmountExt,target,daExt.getDisplayName(),daExt.getLocalName(),STORAGE_PREFIX+da.getUuid(), logPrefix);
1401            stAmountBought.setThin(false);
1402            // Set Key for Storage Amount as Storage Controller if 7 mode
1403            Commodity stAmountSold = sc.getCommoditiesMap().get(CommodityType.fetch(AbstractionPackage.eINSTANCE.getStorageAmount(), saKey));
1404            stAmountSold.getConsumedBy().add(stAmountBought);
1405
1406            storageCommodity.setResizeable(true); storageCommodity.getChargedBy().add(stAmountBought);
1407            stprovCommodity.setResizeable(true); stprovCommodity.getChargedBy().add(stAmountBought);
1408
1409            da.setHostedBy(sc);
1410        }
1411
1412        handleNewSE(da, removedObjects);
1413        dm.addNewObject(da);
1414        da.powerStateChanged(EntityPowerState.POWERED_ON);
1415        return da;
1416    }
```

Repeated patterns. Difficult to comprehend

VMTURBO

After Decomposition

Functions	Number of Ways
Model Instantiation	1
Update Supply and Demand Data	1
Update Entity Data	1

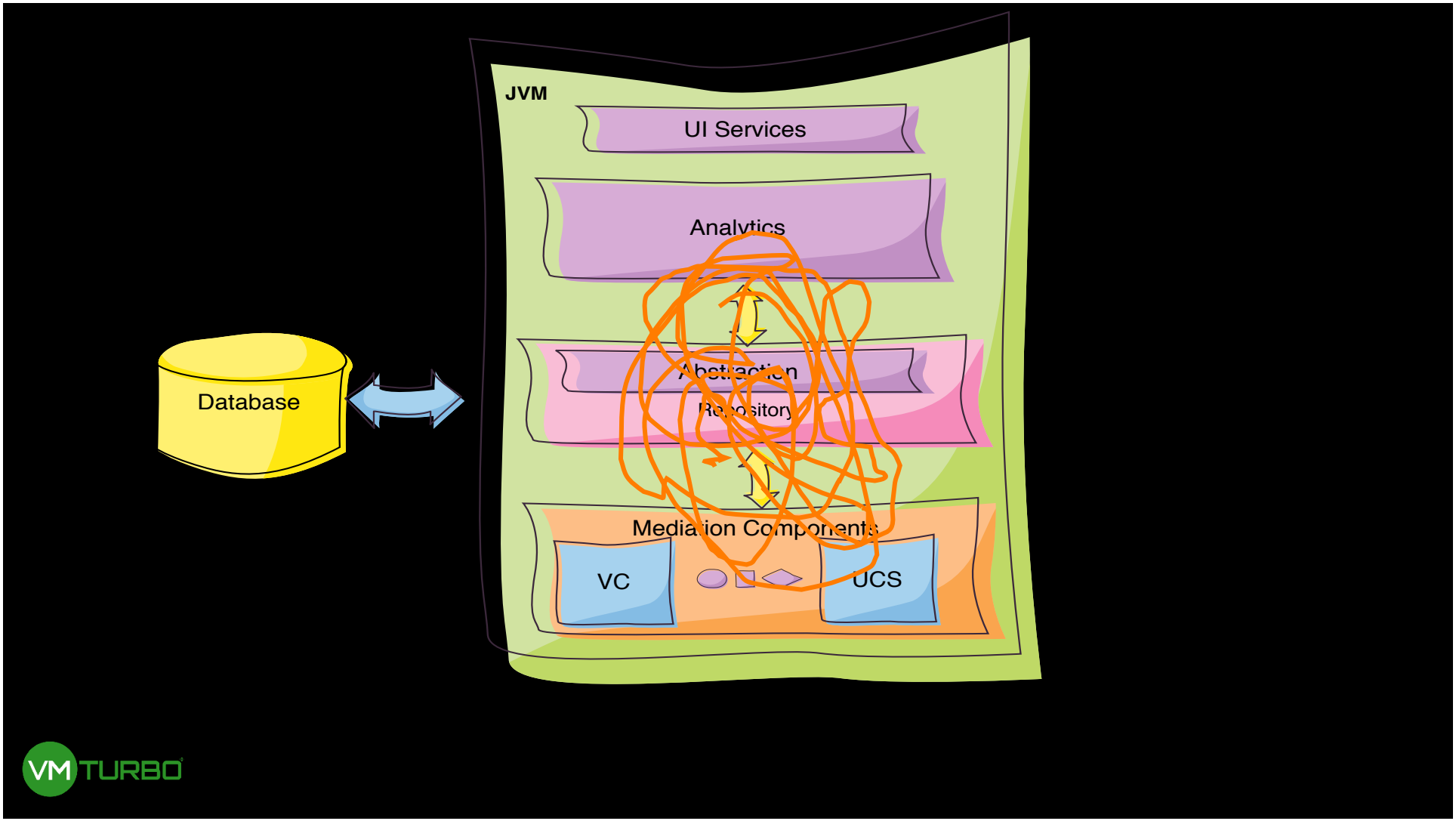
VM TURBO

```
172
173         // DiskArray entity DTO
174         DiskArrayBuilder da = new DiskArrayBuilder(DA1_ID)
175         .displayName(DA1_NAME)
176         .lunId(lunUUIDs)
177         .path(DA1_PATH_VAL)
178         // Commodities sold
179         .storageAcess(100F)
180         .storageAmount(100F)
181         .storageProvisioned(100F)
182         .storageLatency(100f)
183         .storageExtent(100f)
184         // Commodities bought, with corresponding provider
185         .storageController(SC_ID)
186         .storageAmountBought(null, 100f, 1f)
187         .cpuBought(null, 100f, 1f);
188         EntityDTO dae = da.configure();
```

VM TURBO

Untangling the Interfaces

VM TURBO


JVM
UI Services
Analytics
Abstraction
Repository
Mediation Components
VC
UCS
Database
VM TURBO

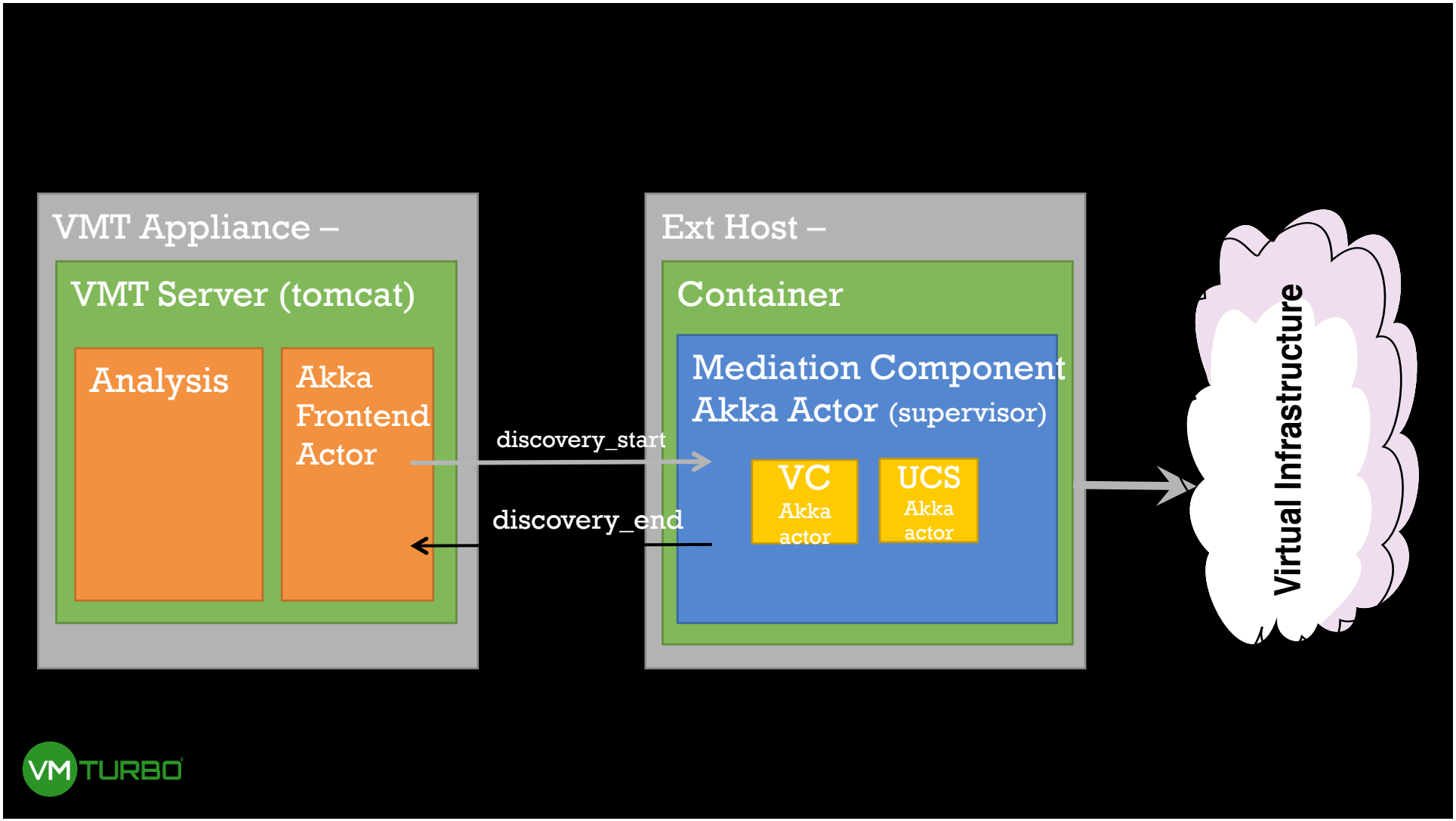
- The Approach
  - Clean up Interface between Mediation and Analysis
  - Separate Mediation functionality from the Monolith
  - Publish APIs

VM TURBO

- Analysis → Mediation Service
  - DISCOVERY_START
  - EXECUTE_ACTION
  - TERMINATE, RESTART, PAUSE

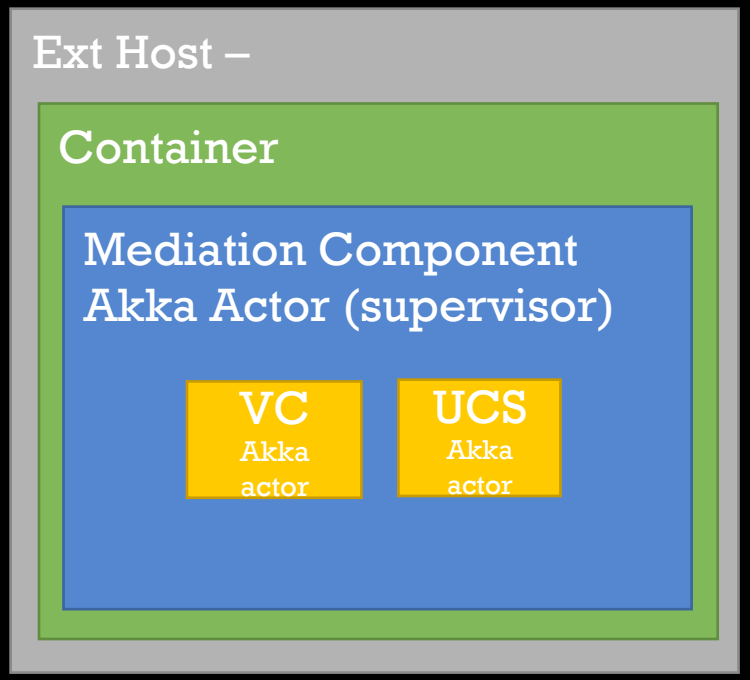
- Mediation Service → Analysis
  - REGISTER COMPONENT
  - ADD_ENTITY
  - UPDATE_ENTITY
  - DELETE_ENTITY
  - END_DISCOVERY
  - UNREGSITER

VM TURBO


VMT Appliance –
VMT Server (tomcat)
Analysis
Akka Frontend Actor
discovery_start
discovery_end
Ext Host –
Container
Mediation Component Akka Actor (supervisor)
VC Akka actor
UCS Akka actor
Virtual Infrastructure
VMTURBO

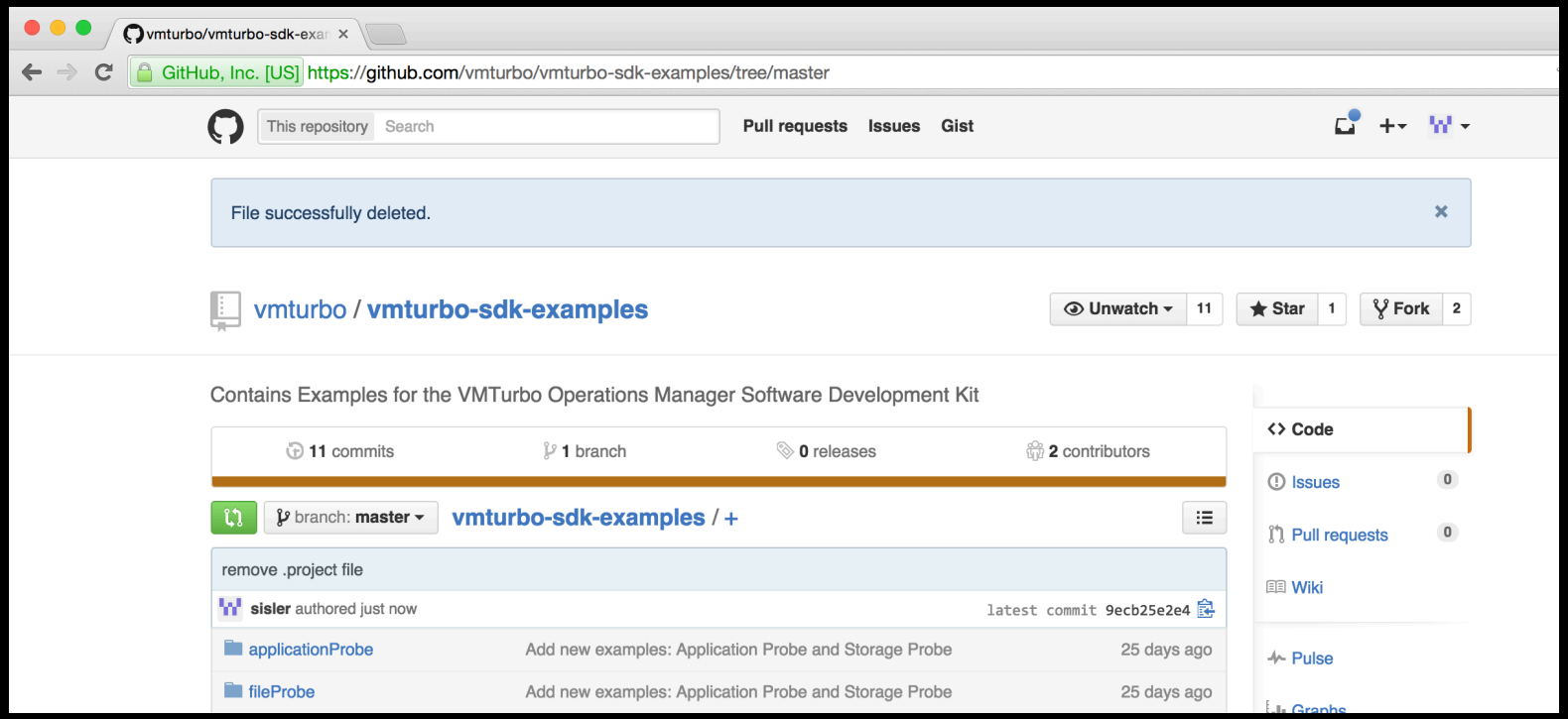
Key Features

- Maintains No State
- Operations on a projection of the Analysis
- Model
- Conforms to a very simple interface
- Available in the VMTurbo SDK



VMTURBO

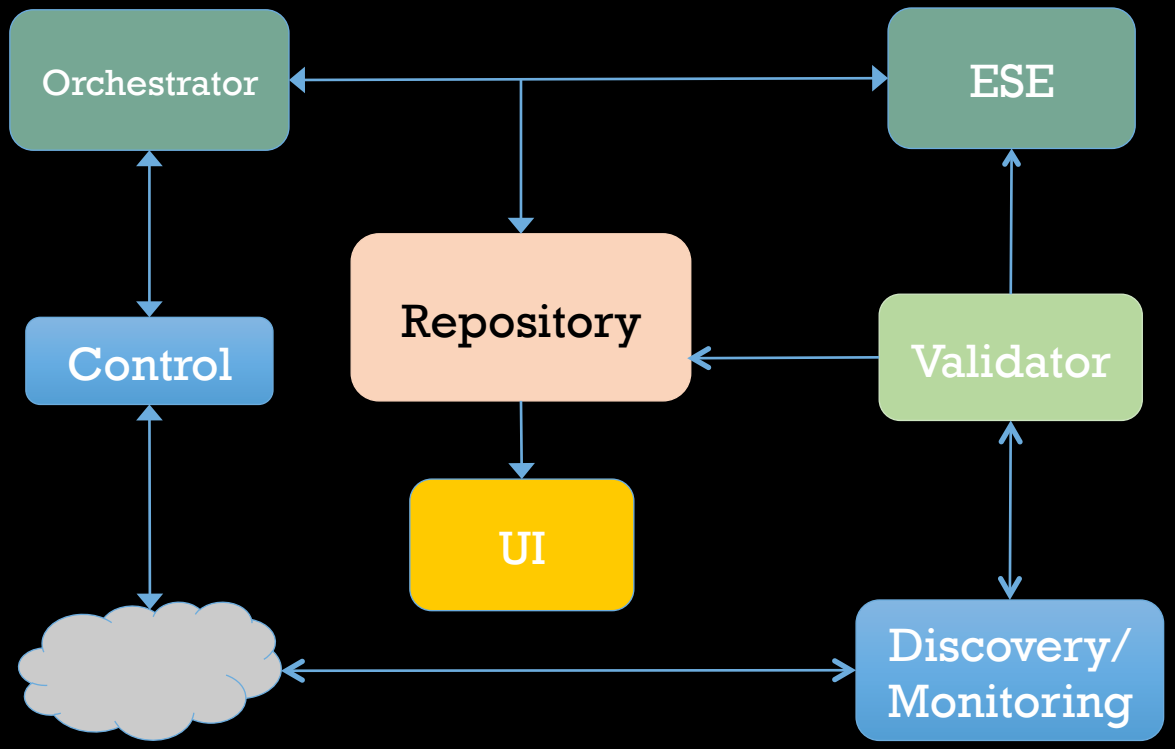
Examples available on github:



The VMTurbo Component Ecosystem – Next Steps

VMTURBO

Components and High Level Flow



Discovery Monitoring
Domain specific mediation Components

Validator:
Consumes mediation data.
Constructs and validates Market Participants and Commodities

ESE
Economic Scheduling Engine. Consumes Market Participants and Commodities
Emits action items

Orchestrator
Consumes action items from the ESE,
Filters them based on constraints and
Provides them to Control.

Control
Executes actions on the virtual infrastructure

Communication is not necessarily point to point

VM TURBO


Fin
VM TURBO