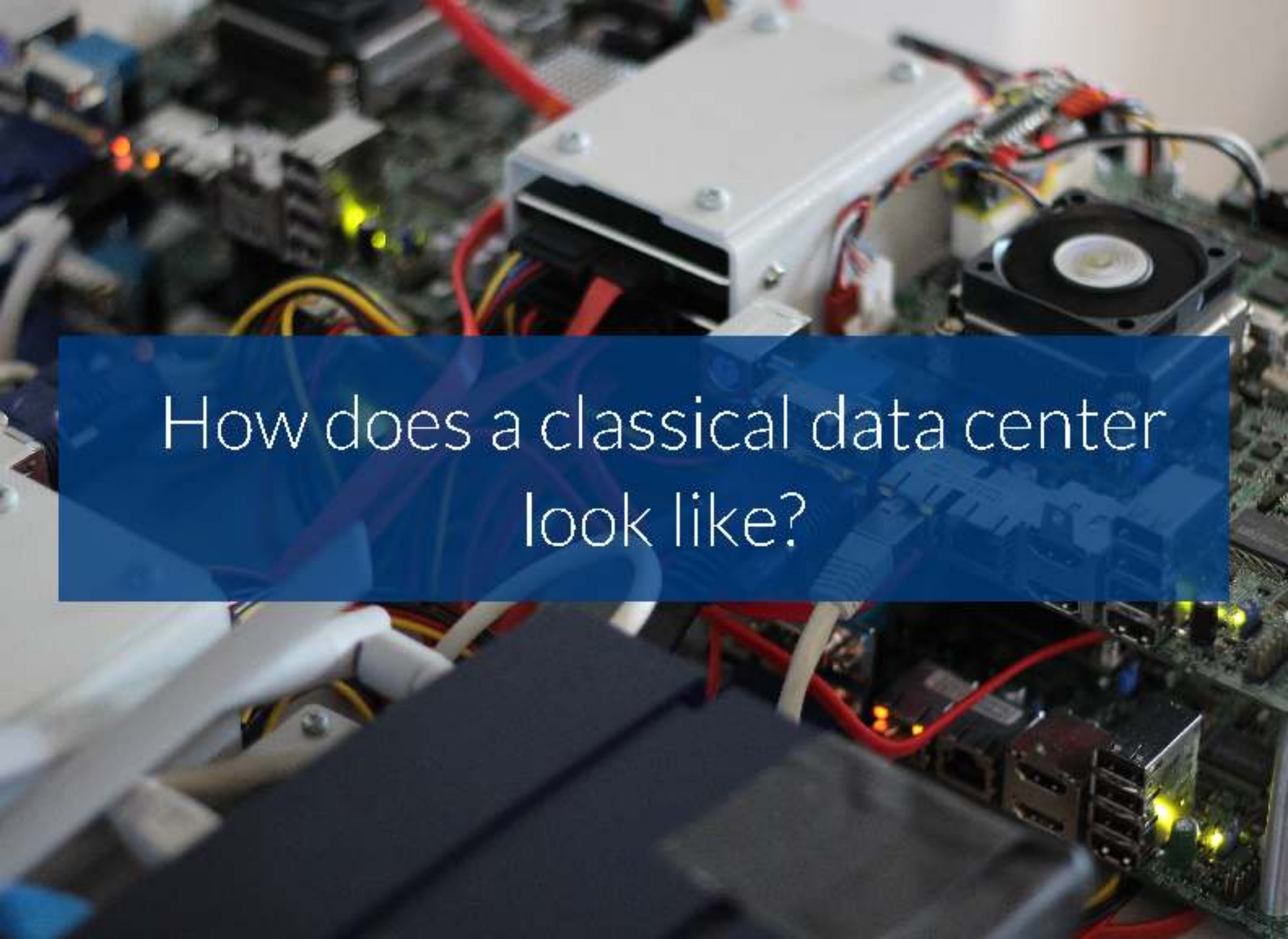
inovex

Software-defined data center

Johannes M. Scheuermann

Johannes M. Scheuermann

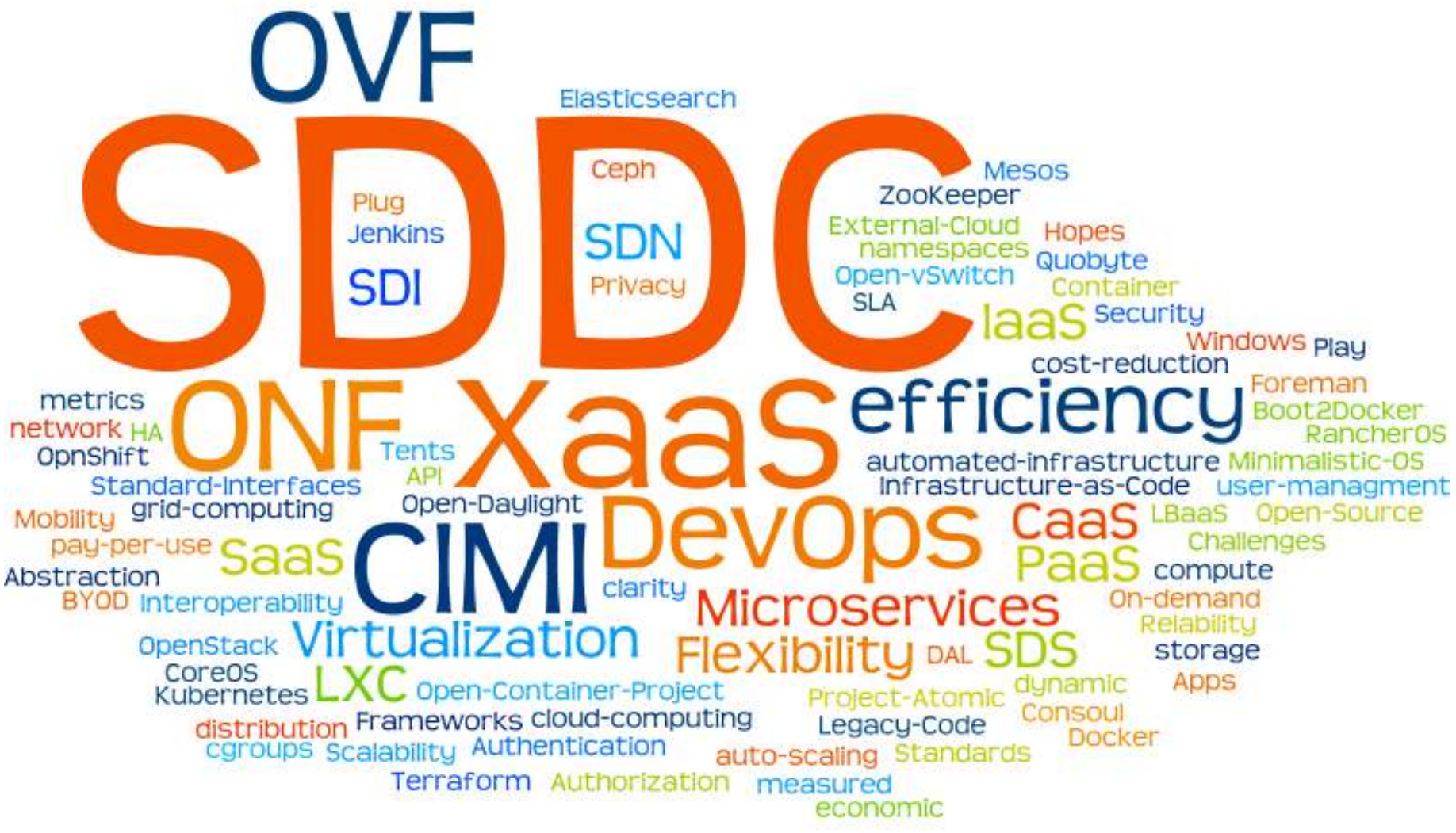
- KIT & Working student
- IT Engineering & Operations @inovex
- Working with new data center technologies
- @johscheuer


How does a classical data center look like?

Challenges from an Ops view


Challenges from a Dev view

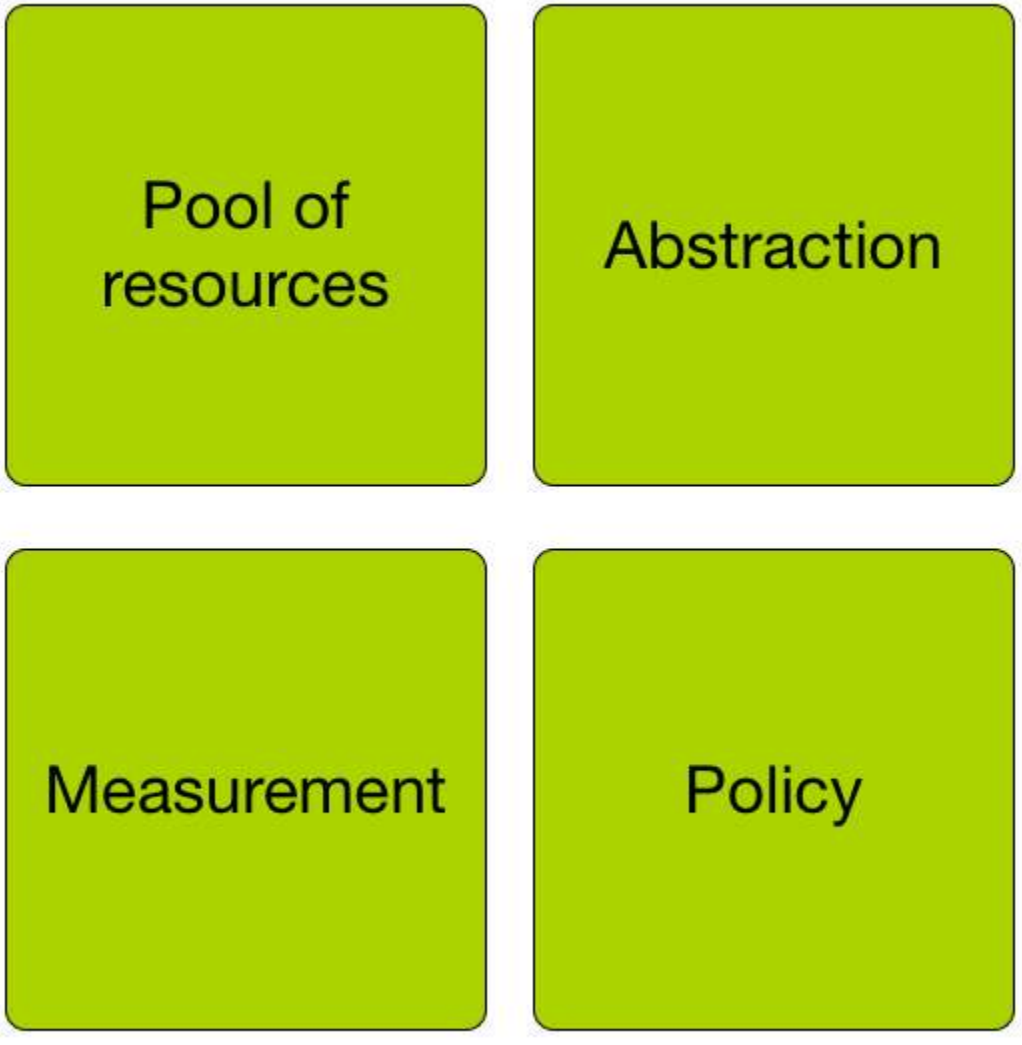

A new approach the SDDC


OVF
SDDC
Elasticsearch
Ceph
Mesos
ZooKeeper
Plug
Jenkins
SDI
SDN
Privacy
External-Cloud
namespaces
Open-vSwitch
SLA
Hopes
Quobyte
Container
IaaS
Security
Windows
Play
cost-reduction
metrics
network
HA
OpnShift
ONF
Tents
API
XaaS
efficiency
Foreman
Boot2Docker
RancherOS
automated-infrastructure
Infrastructure-as-Code
Minimalistic-OS
user-managment
Standard-Interfaces
grid-computing
Open-Daylight
Mobility
CIMI
DevOps
CaaS
LBaaS
Open-Source
pay-per-use
SaaS
PaaS
Challenges
Abstraction
BYOD
Interoperability
clarity
compute
Microservices
On-demand
Relability
OpenStack
Virtualization
Flexibility
DAL
SDS
storage
CoreOS
Kubernetes
LXC
Open-Container-Project
dynamic
Apps
Project-Atomic
Consoul
distribution
Frameworks
cloud-computing
Legacy-Code
Docker
cgroups
Scalability
Authentication
auto-scaling
Standards
Terraform
Authorization
measured
economic

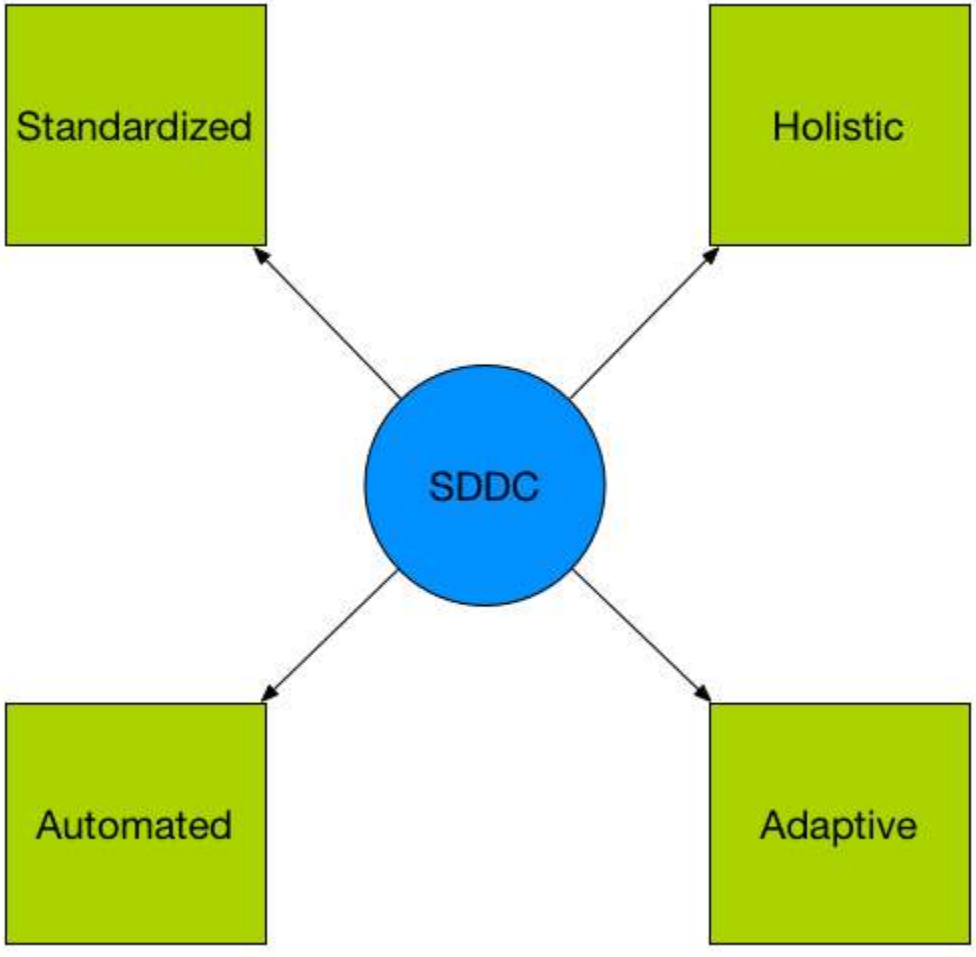
„A Software Defined Data Center (SDDC) is a data storage facility in which all elements of the infrastructure – networking, storage, CPU and security – are virtualized and delivered as a service."

– *DMTF*

Goals



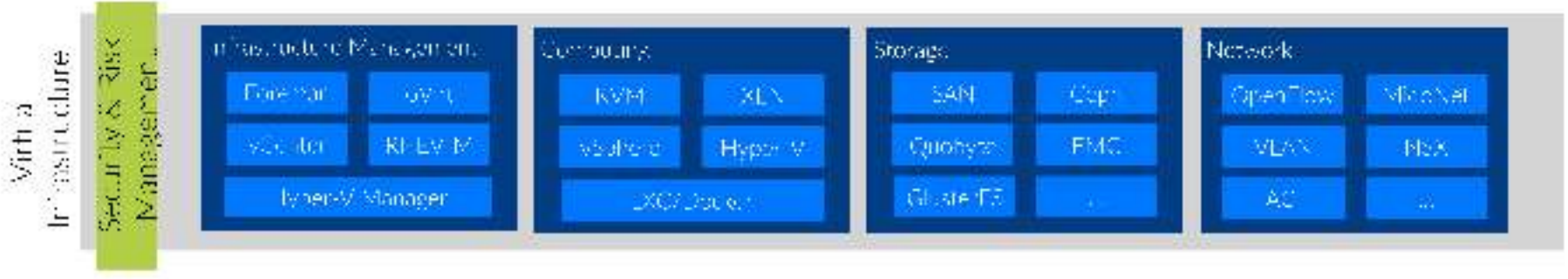
A SDDC should be



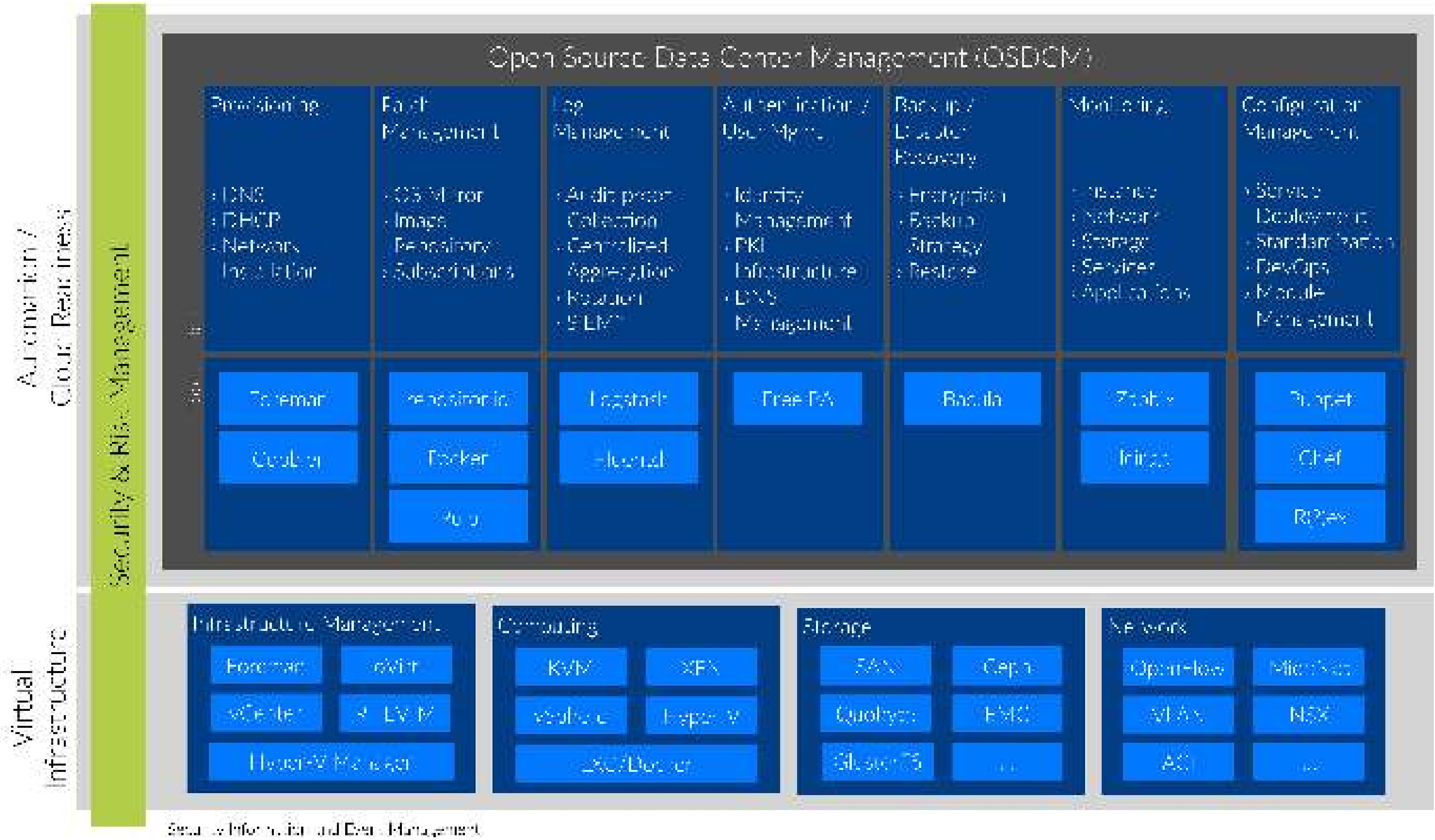
Physical data center


Steps to the SDDC

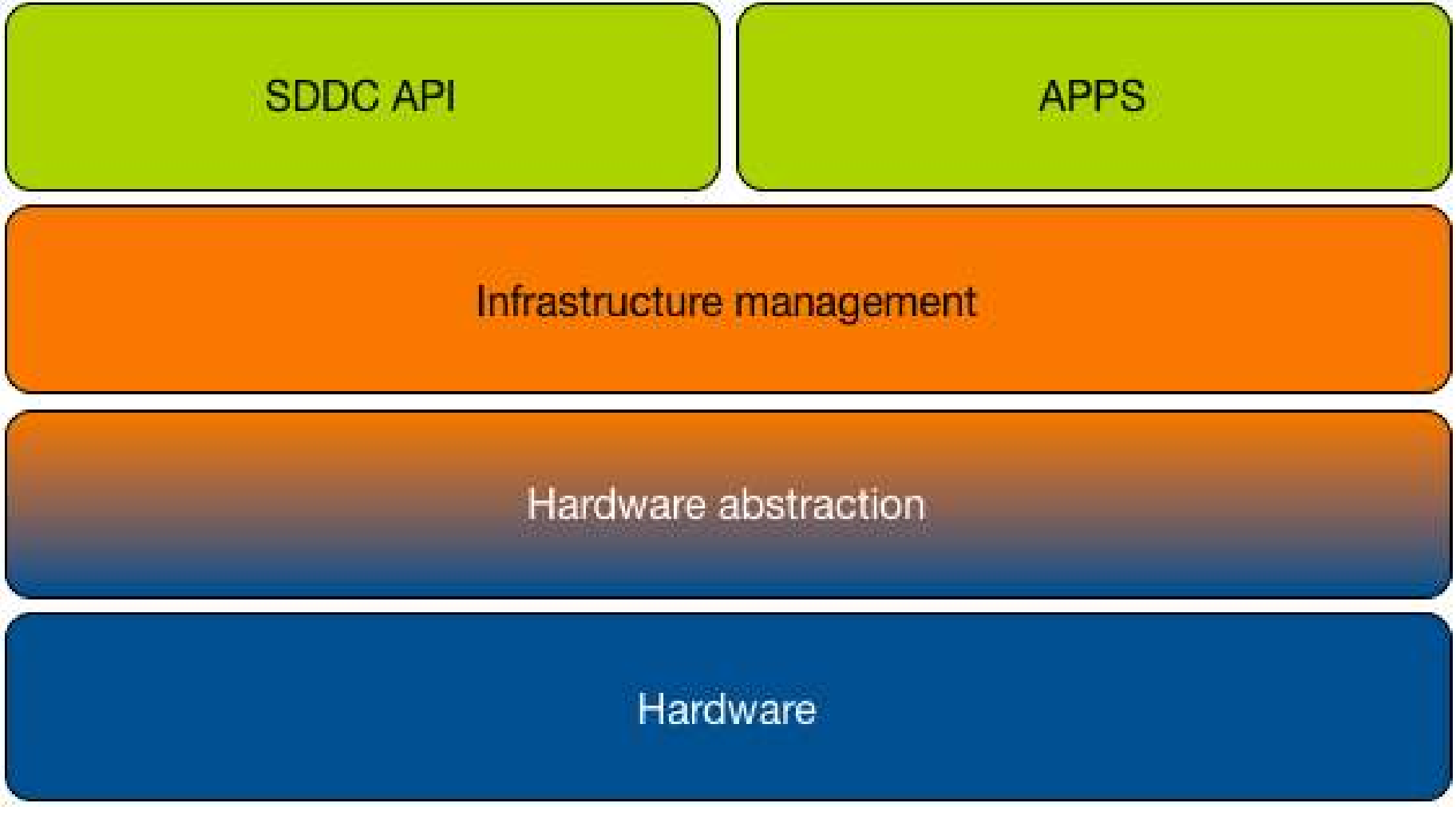
Virtual Infrastructure



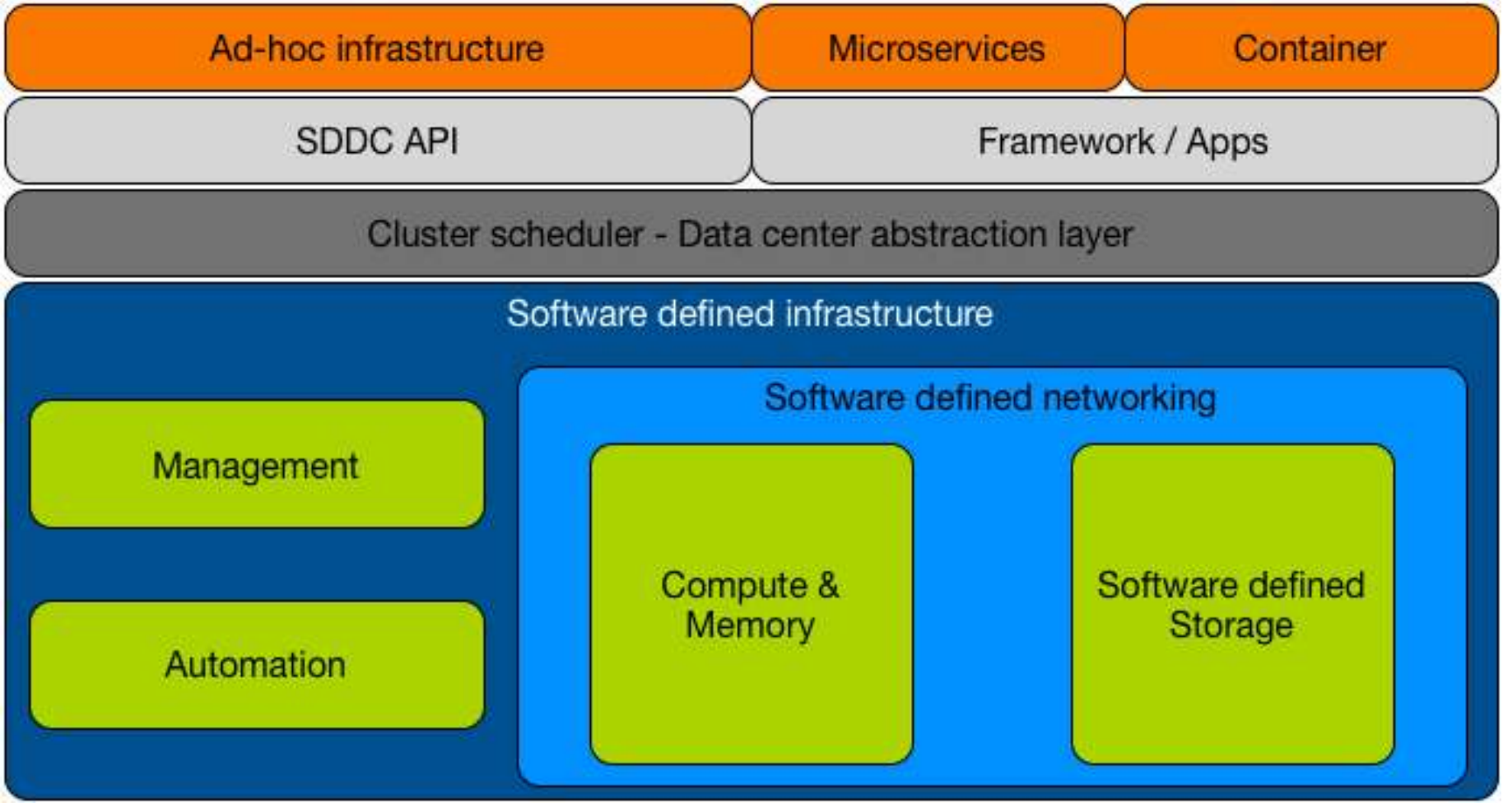
OSDCM


Open Source Data Center Management (OSDCM)
Automation / Cloud Readiness
Security & Risk Management
Provisioning
Patch Management
Log Management
Authentication / User Mgmt
Backup / Disaster Recovery
Monitoring
Configuration Management
Foreman
Cobbler
Docker
Logstash
FreeIPA
Bacula
Zabbix
Puppet
Chef
Virtual Infrastructure
Infrastructure Management
Computing
Storage
Network
KVM
XEN
Ceph
NSX

Abstract View from 5000 feet

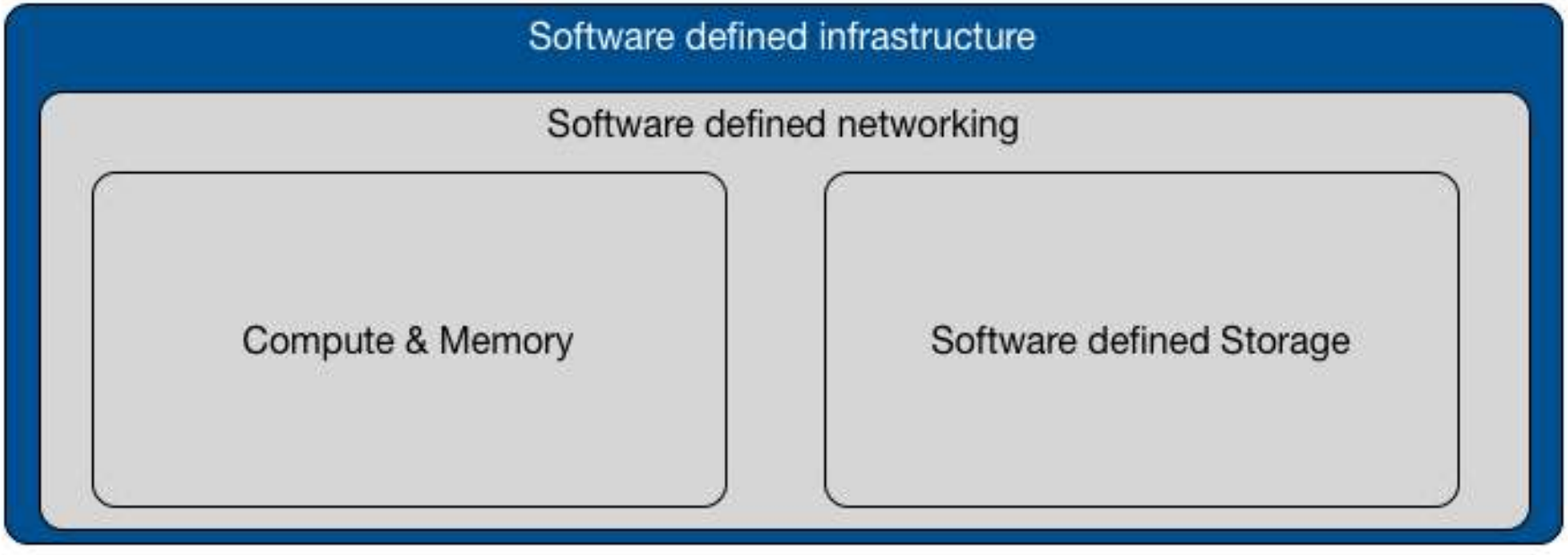


Abstract View from 1000 feet

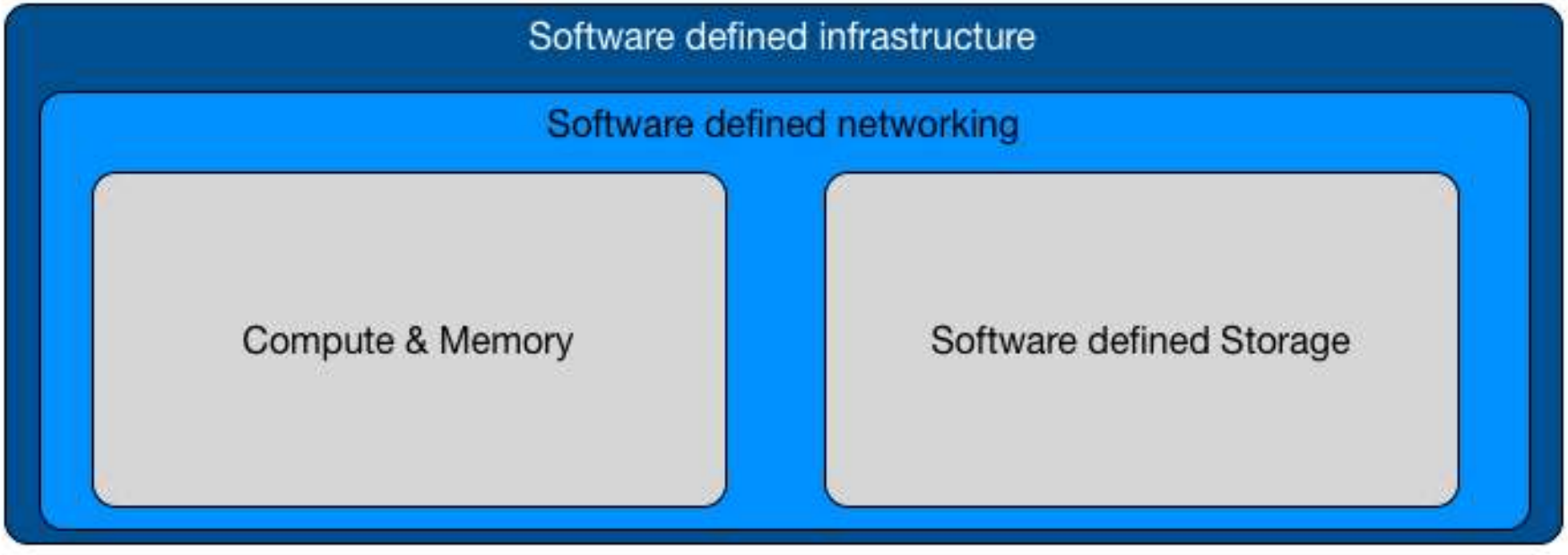


Main parts of a SDDC

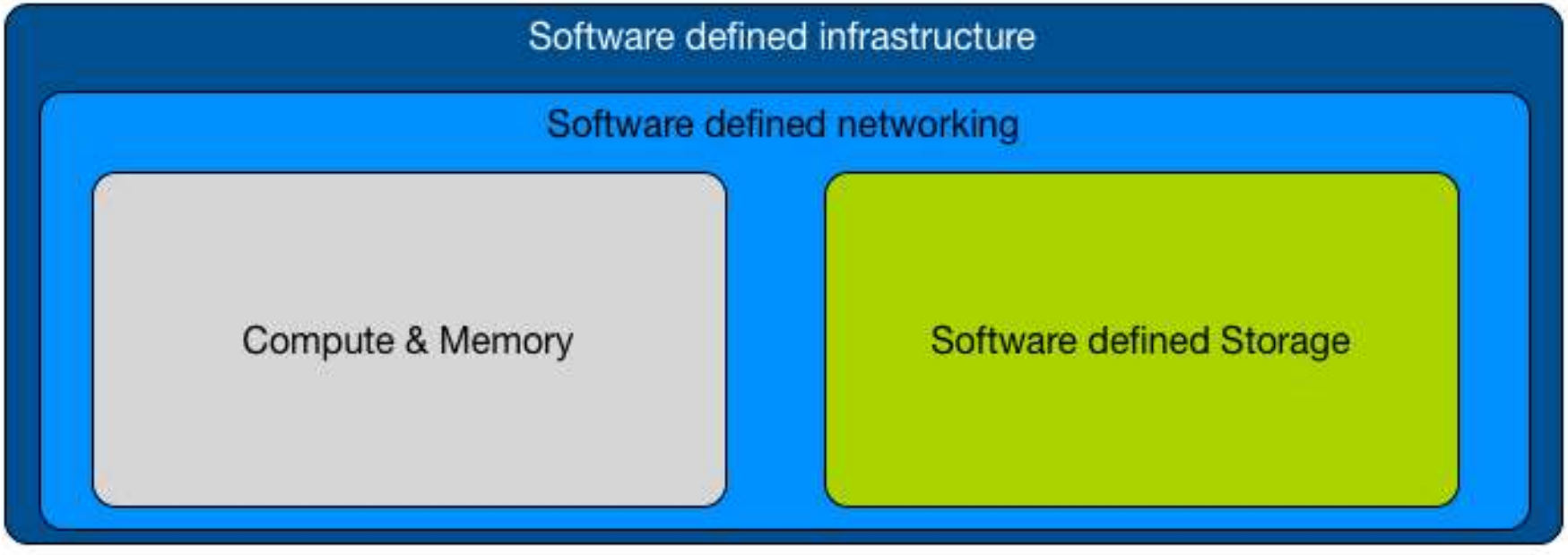
Software defined infrastructure



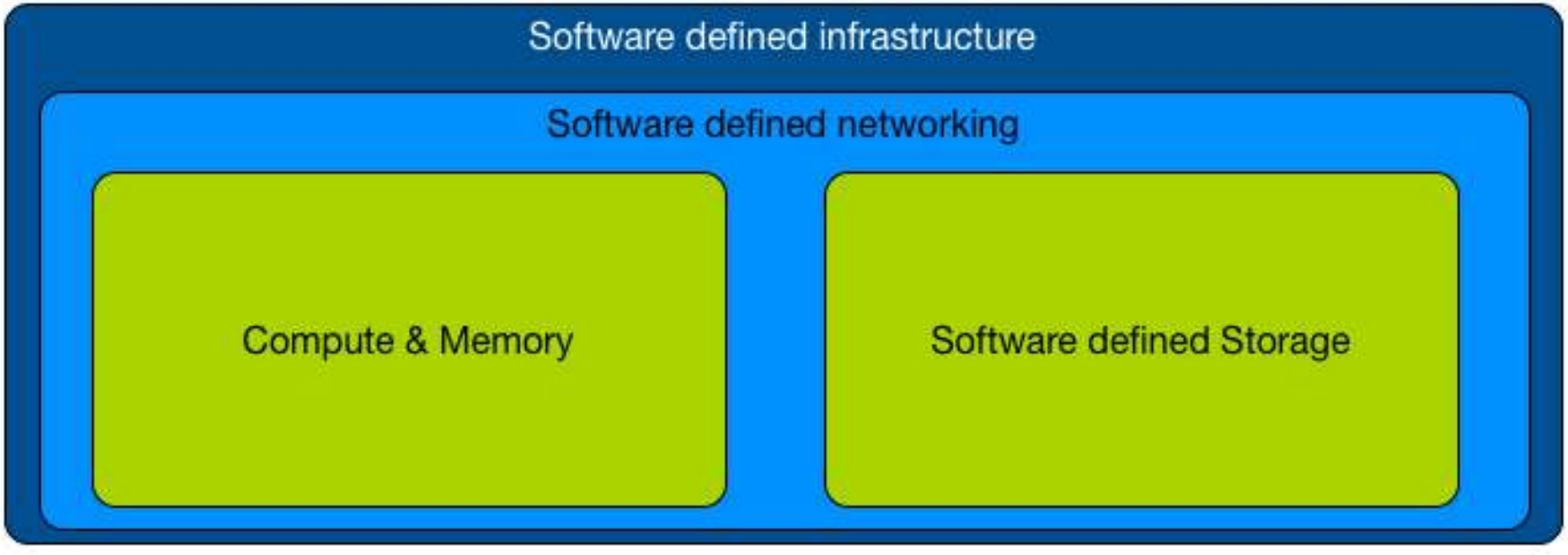
Software defined networking



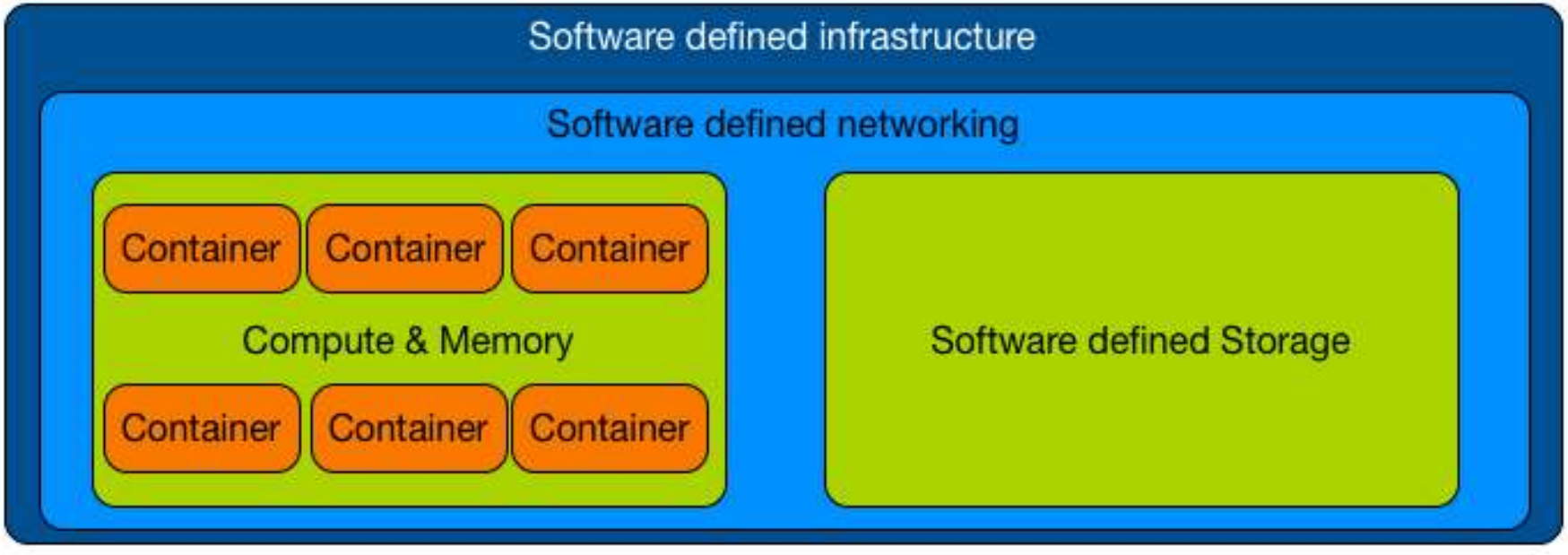
Software defined storage



Server virtualization

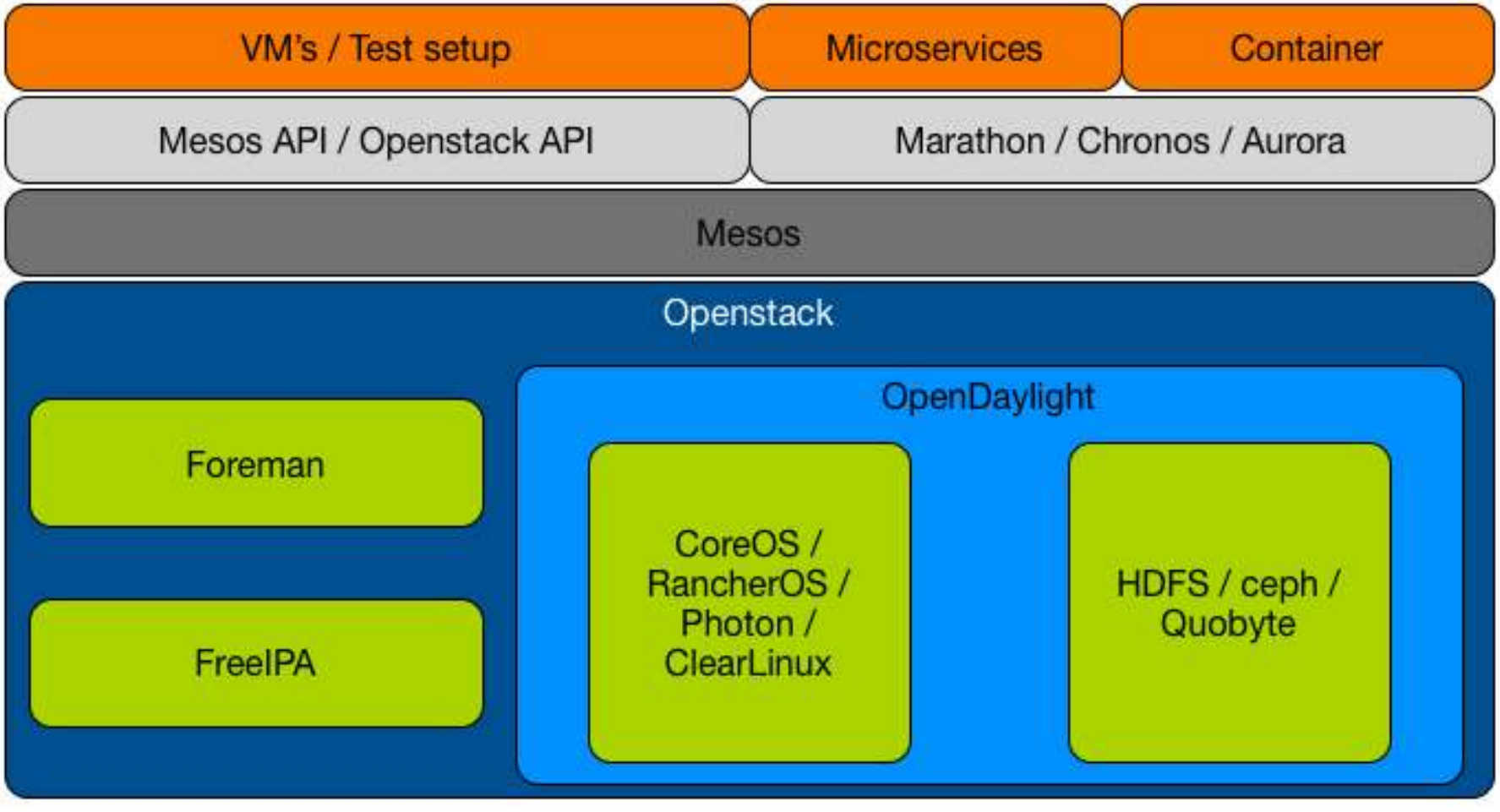


Container


Software defined infrastructure
Software defined networking
Container
Container
Container
Compute & Memory
Container
Container
Container
Software defined Storage

Minimalistic OS

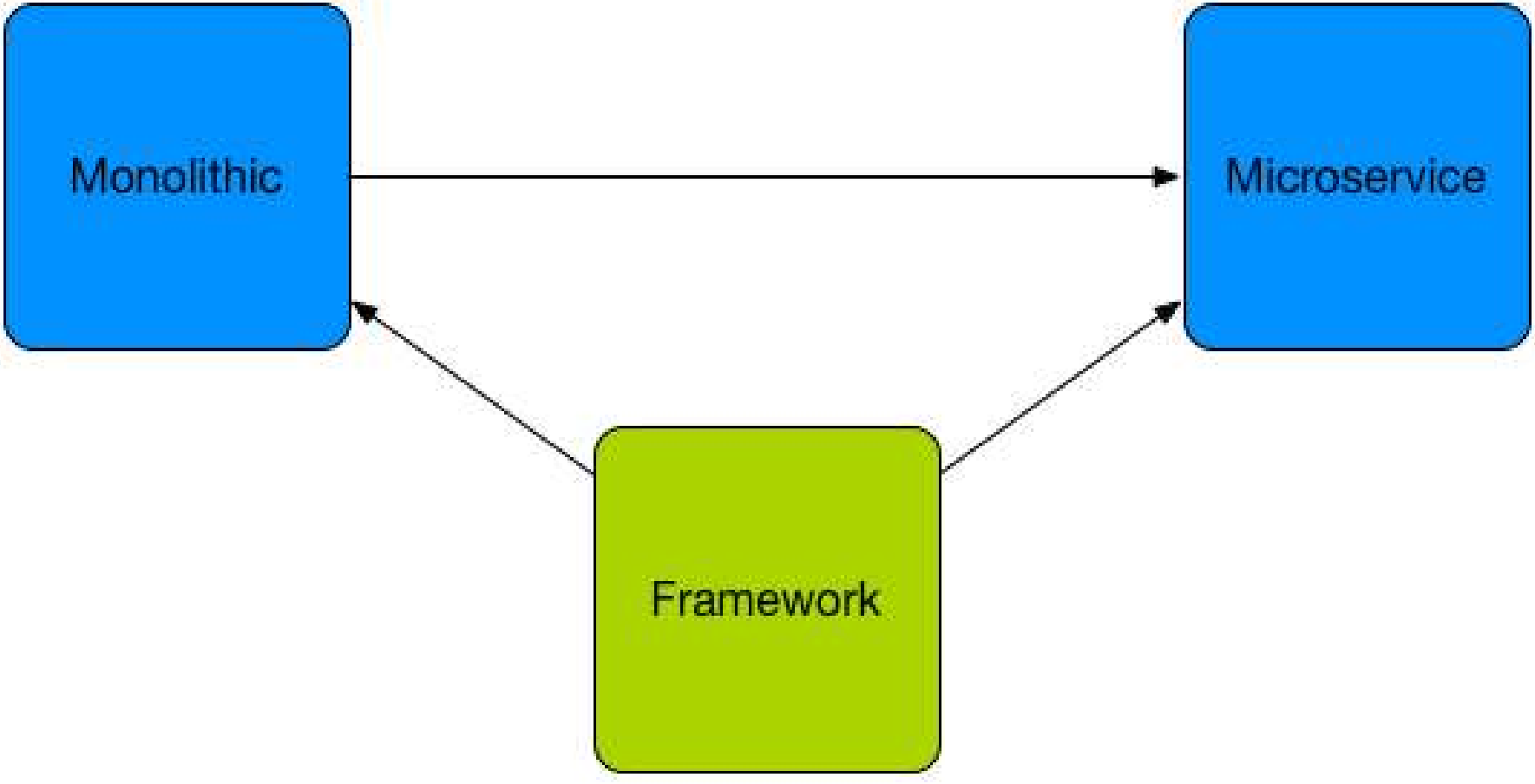
Example Stack



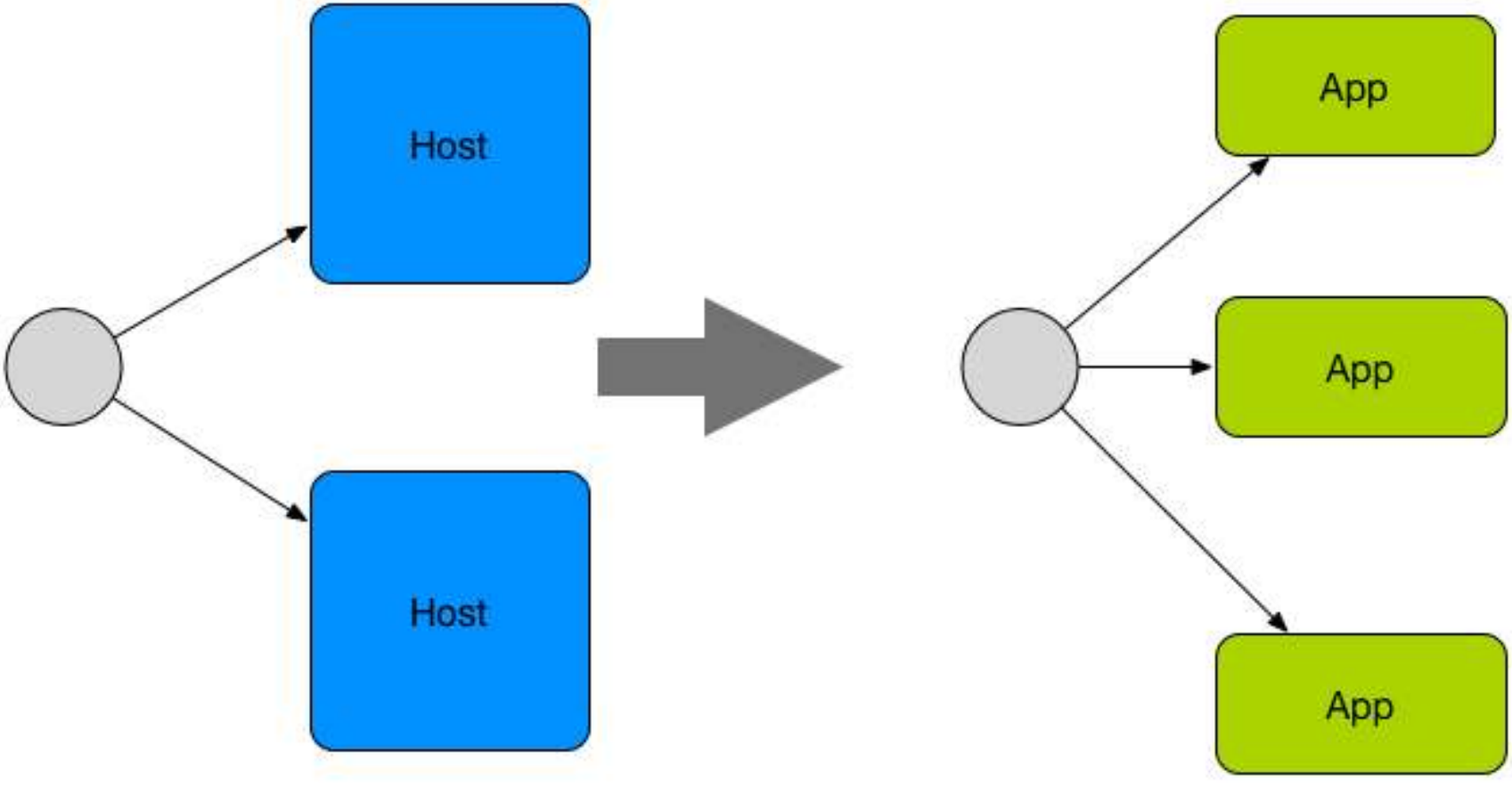
Reference stack - inovex lab

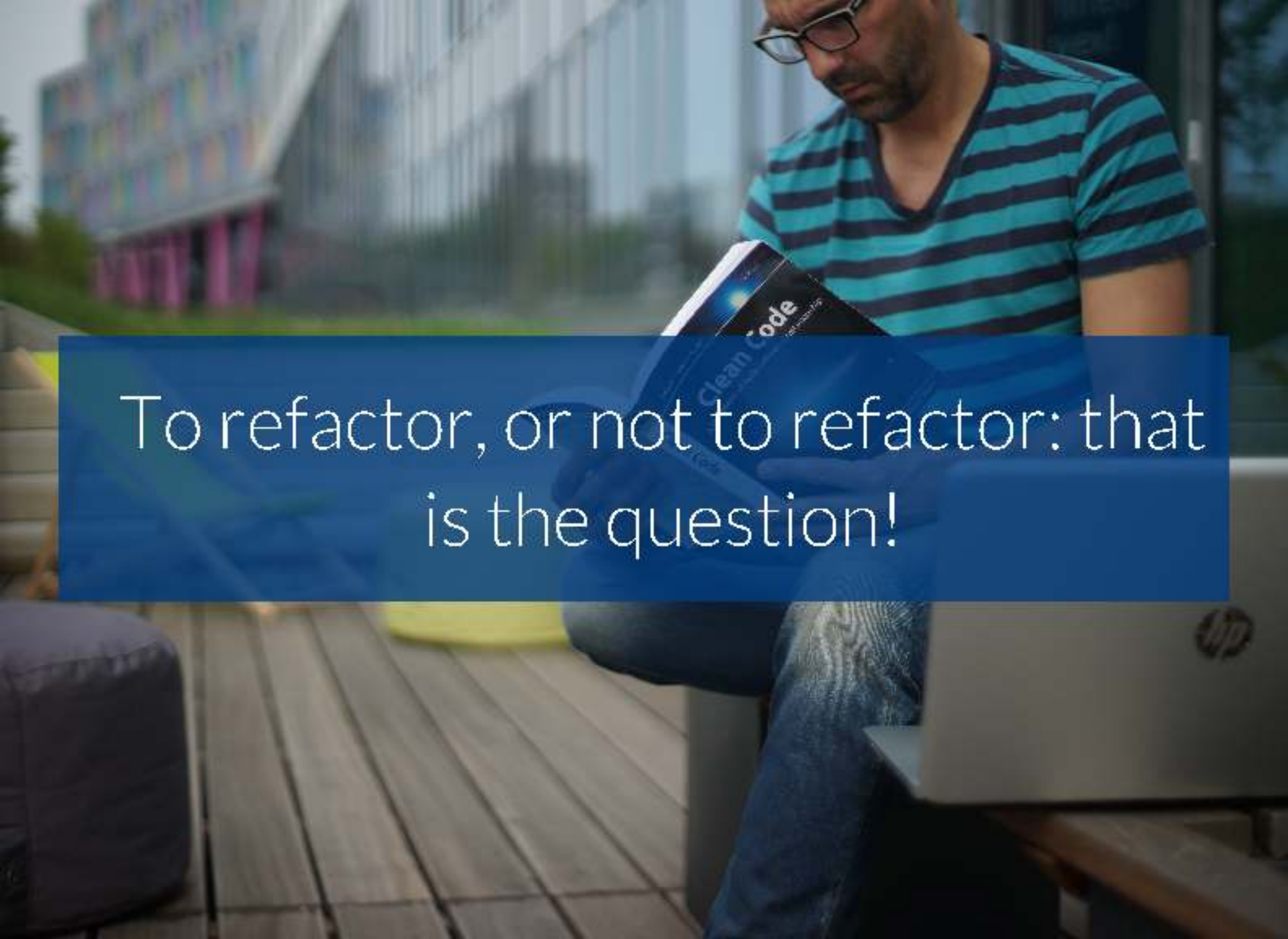
What about the apps?

Apps



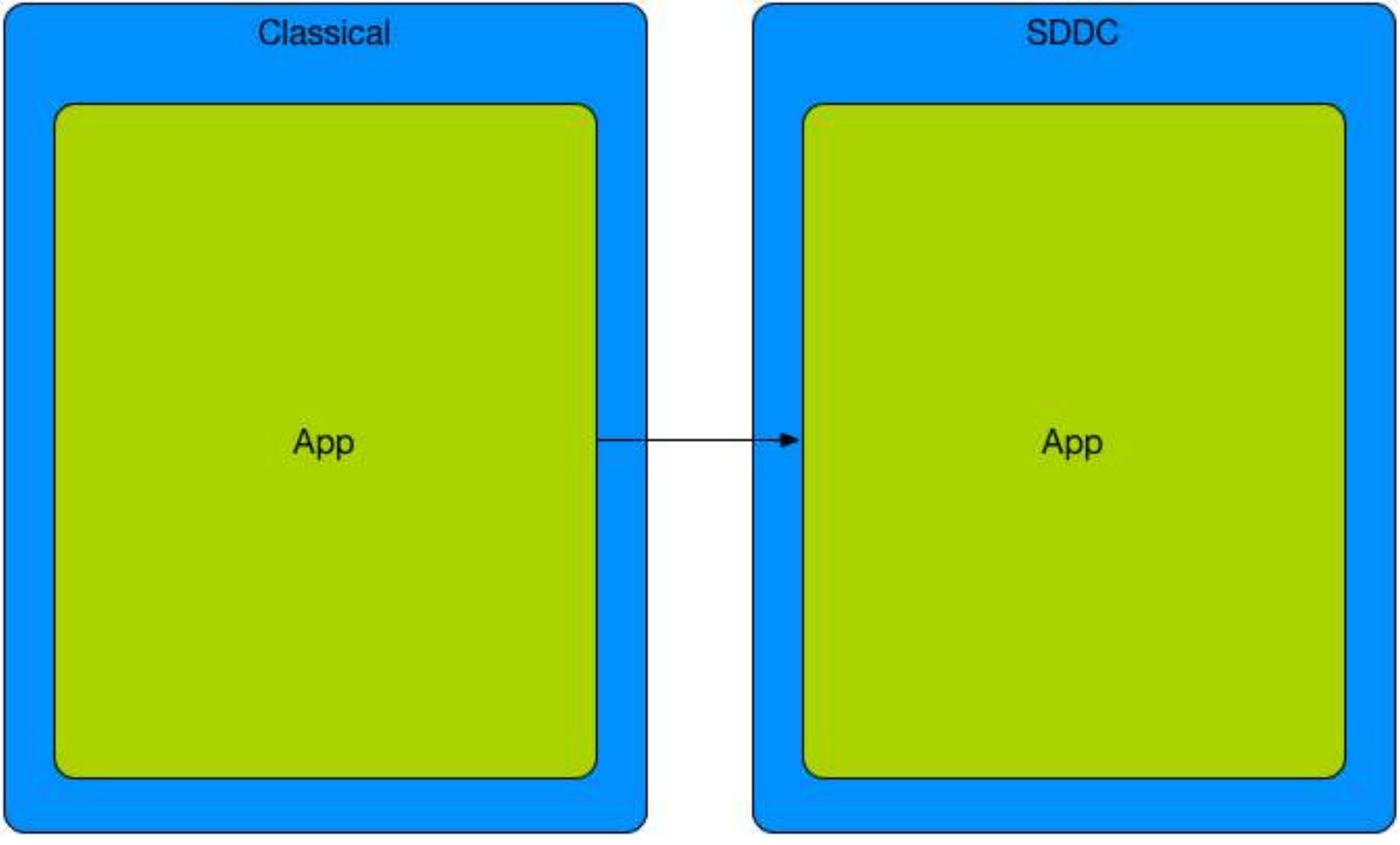
New thinking



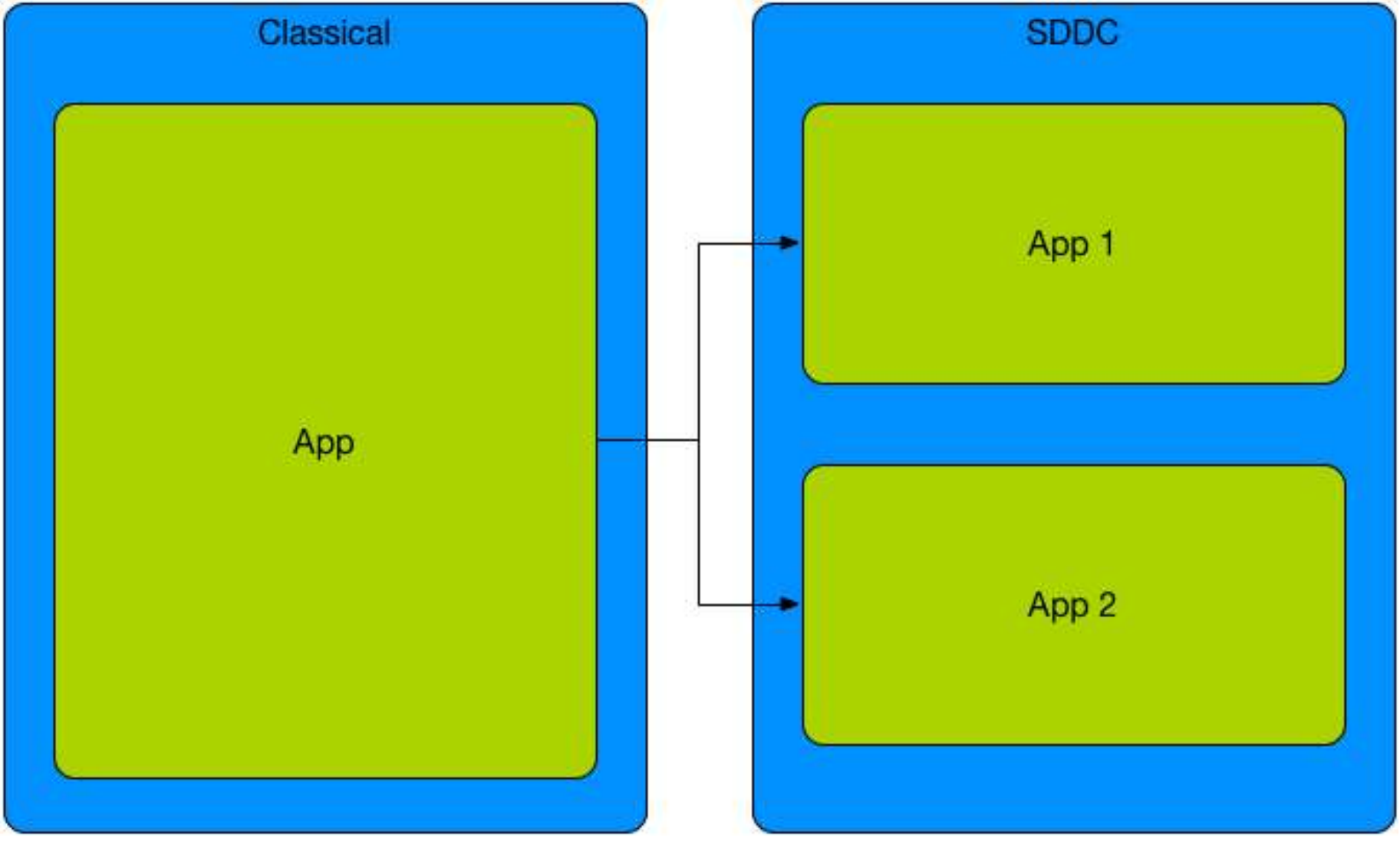


To refactor, or not to refactor: that is the question!

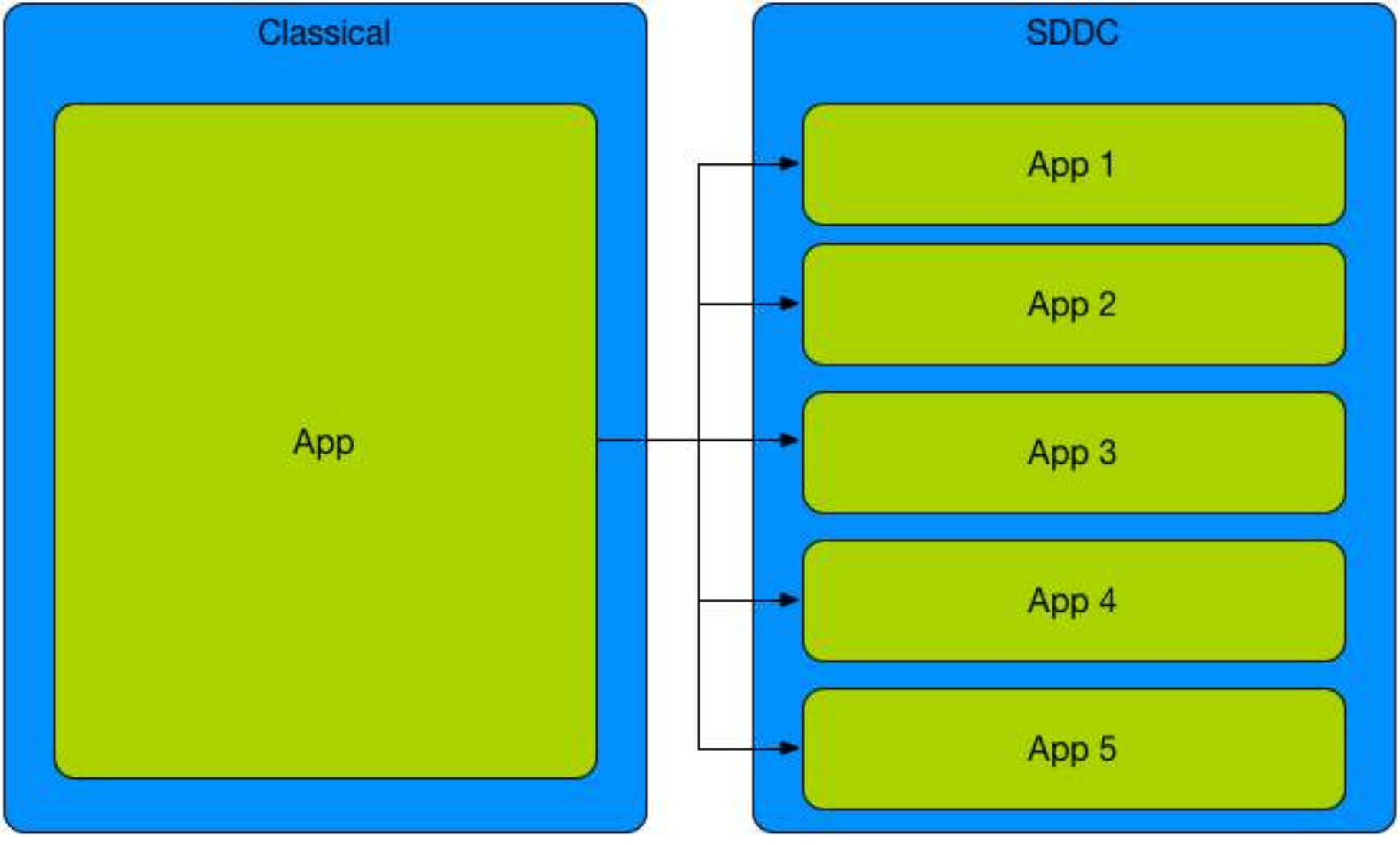
Lift-and-shift



Partial refactor



Complete refactor




Difference SDDC & Cloud

Cloud

- Operational model
- Virtualization technologies
- Delivery and consumption

The difference

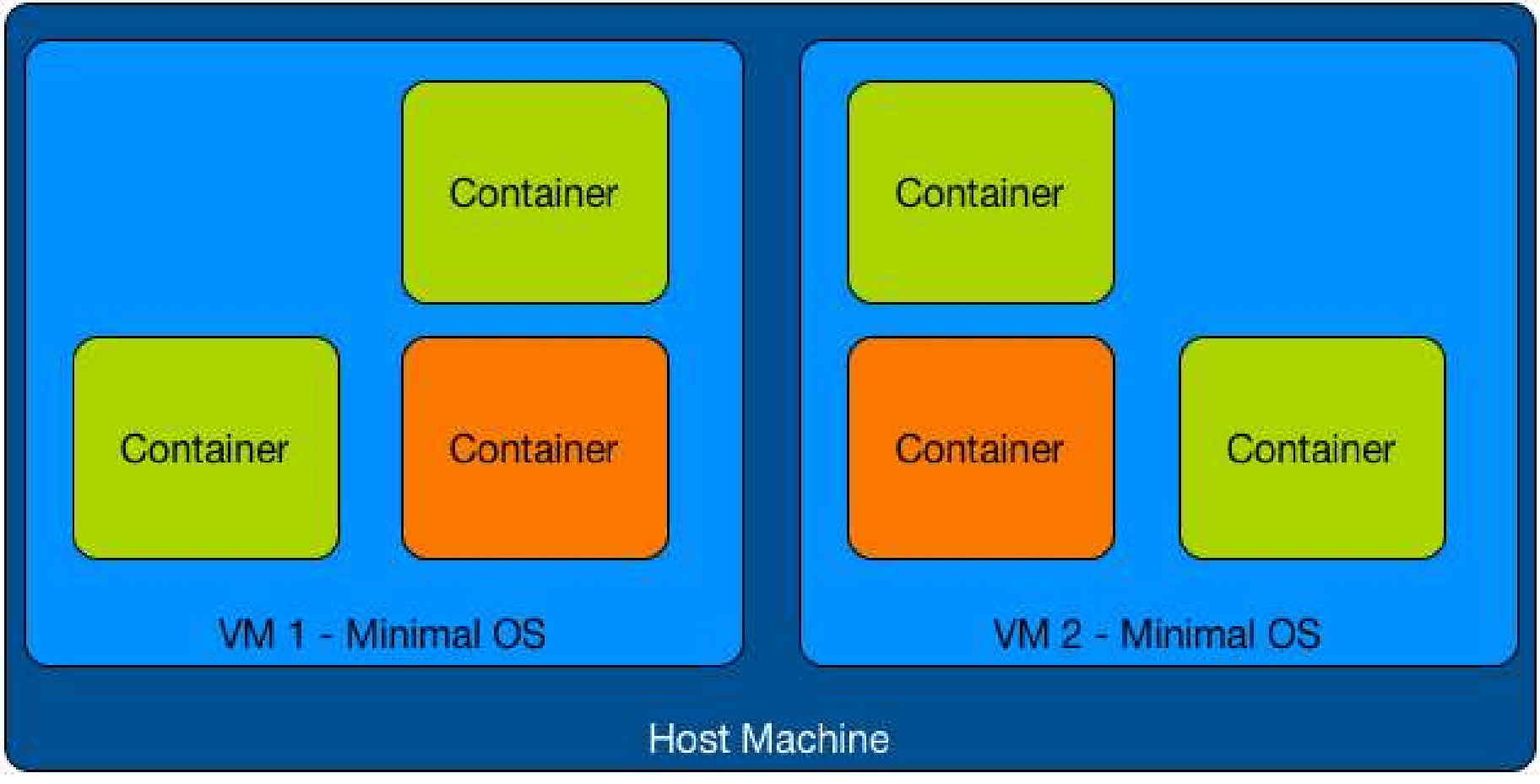
- Enables clouds
- Not the other way around
- Cloud doesn't need as much automation
- Fun fact: Google shutdowns it intranet


Running microservices

Container ≠ Virtual machine

- Package managers
- Security/Separation
- Dependency manager
- Multiple versions

Virtual machines and container



Containers and microservices

- Optimal
- Lightweight
- Scalable


Challenges and hopes
inovex

Challenges

- Infant technologies
- Complexity
- Vendor lock
- Missing standards

Hopes

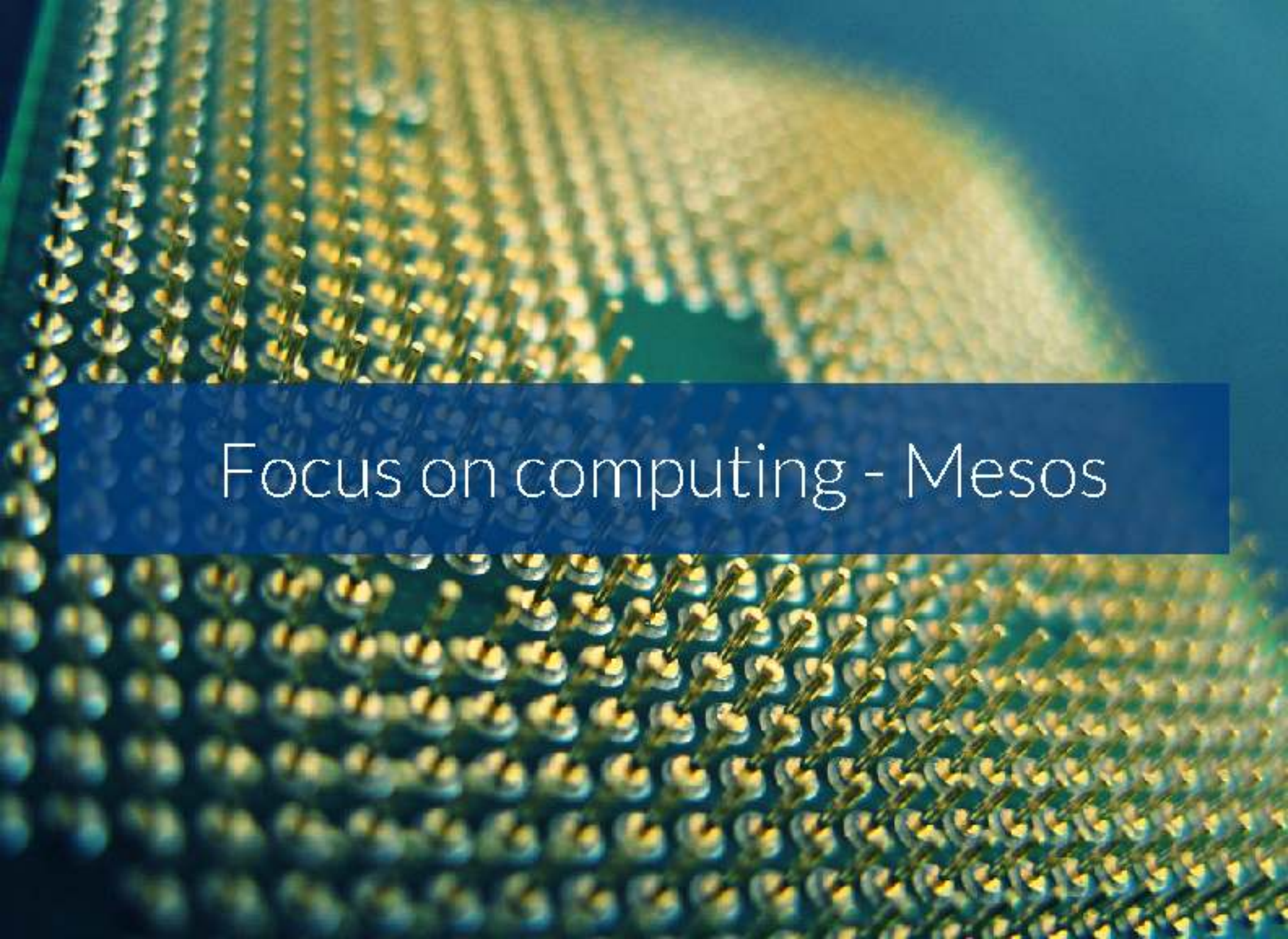
- Rapid resource provisioning
- Elastic scaling
- Policy-driven resource management
- Shared infrastructure
- Instrumentation

Hopes

- Self-service
- Accounting and auditing
- Programmable infrastructure
- Extremely high automation

„Developers will ask for an API not for a VM to run their software“

– *Anonymous*


Focus on computing - Mesos

Challenges in a data center

- Multiple clusters
- Low resource utilization
- Life cycle
- Forget running VMs

Apache Mesos

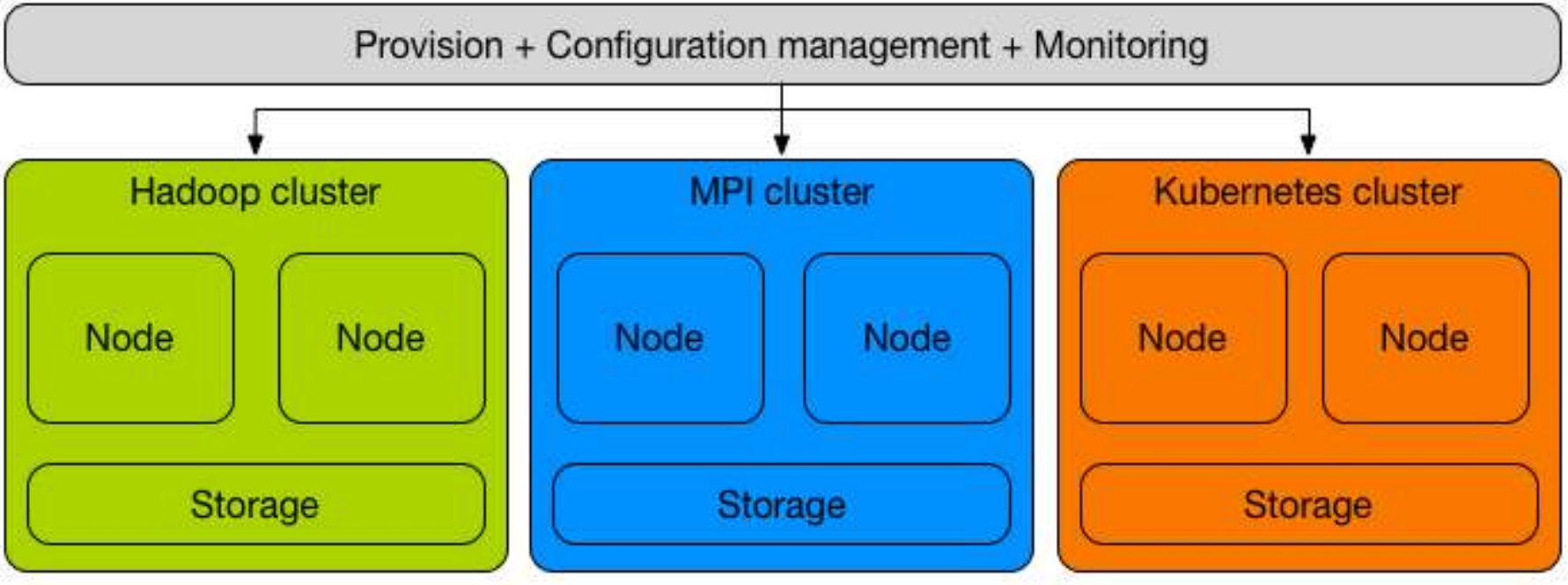
- Data center operating system
- Abstracts CPU, memory, ...
- Provides API for resource management
- Production proven

Other cluster schedulers

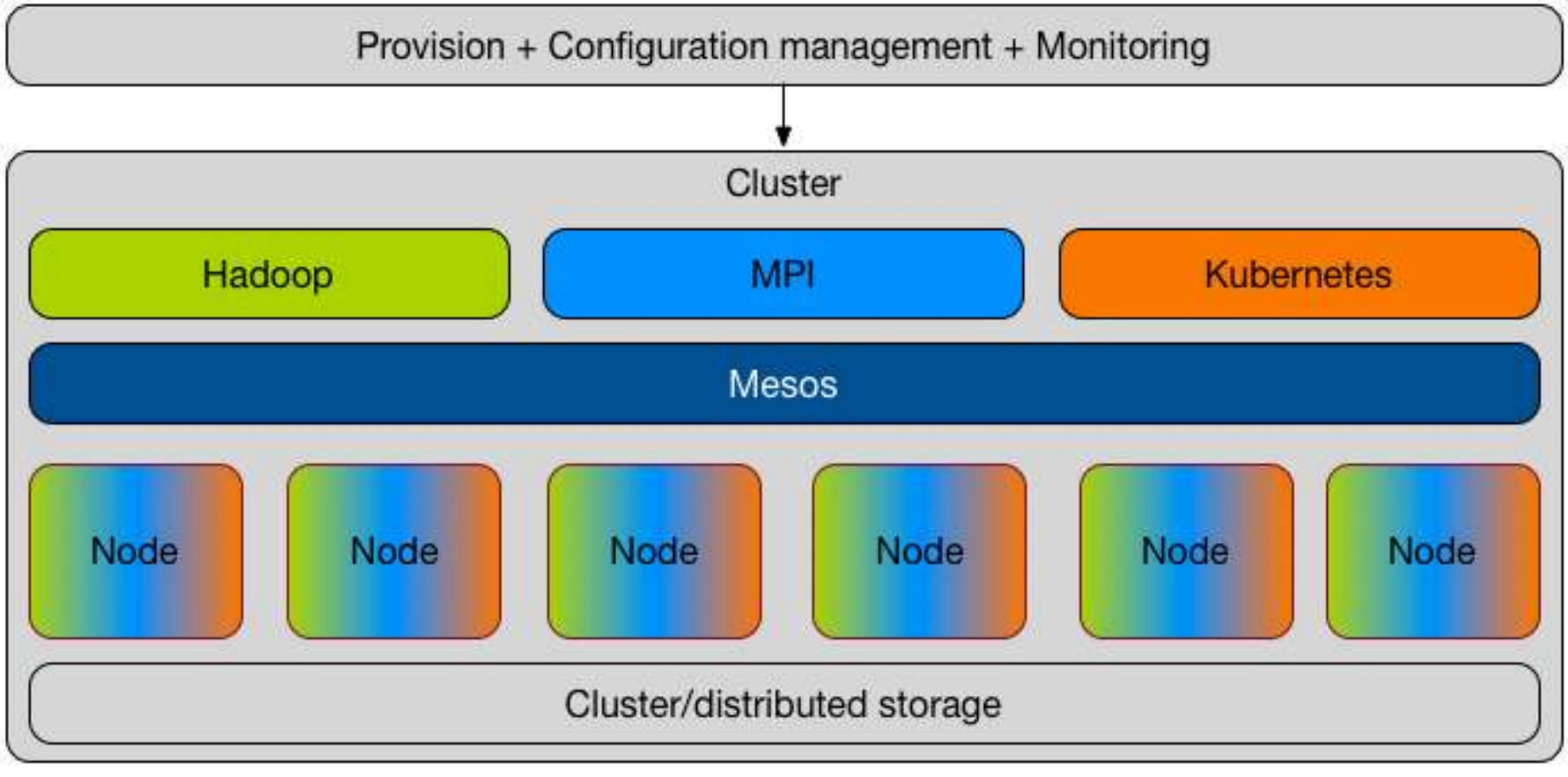
- YARN (Hadoop)
- BORG & OMEGA (Google)
- Quasar (Stanford)
- Dryad (Microsoft)

Why Mesos (only a sample)

Classical setup

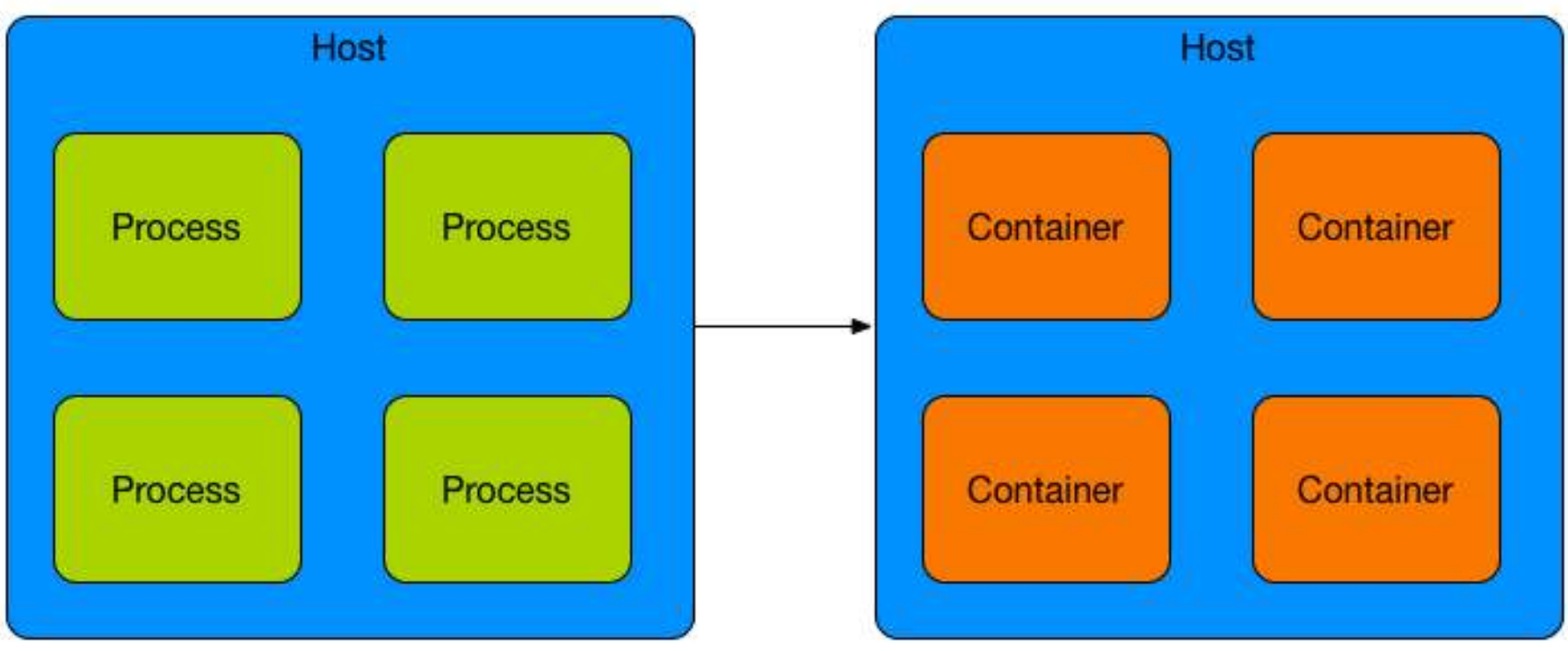


Mesos setup

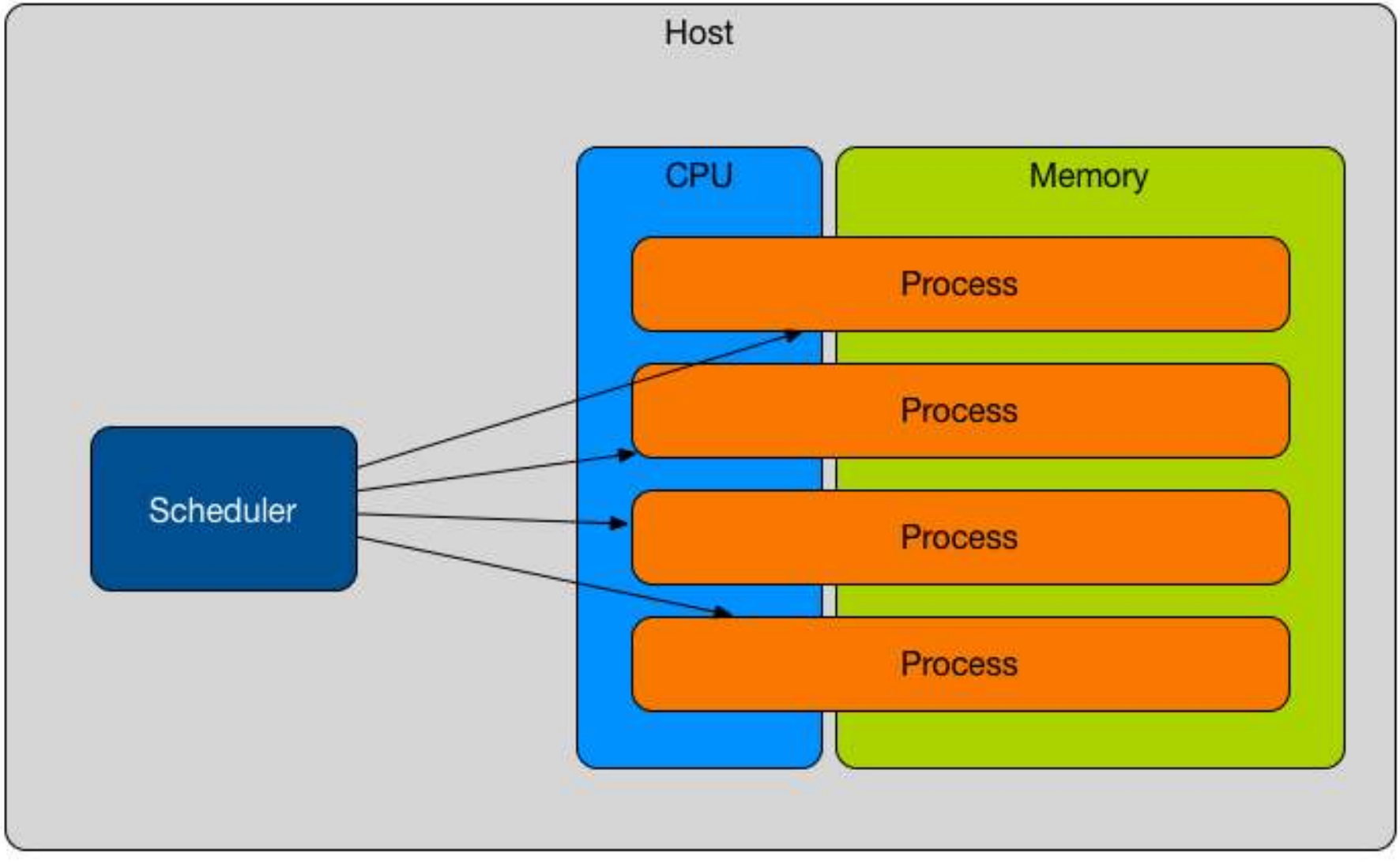

Provision + Configuration management + Monitoring
Cluster
Hadoop
MPI
Kubernetes
Mesos
Node
Node
Node
Node
Node
Node
Cluster/distributed storage


Mesos compared to a “normal” OS

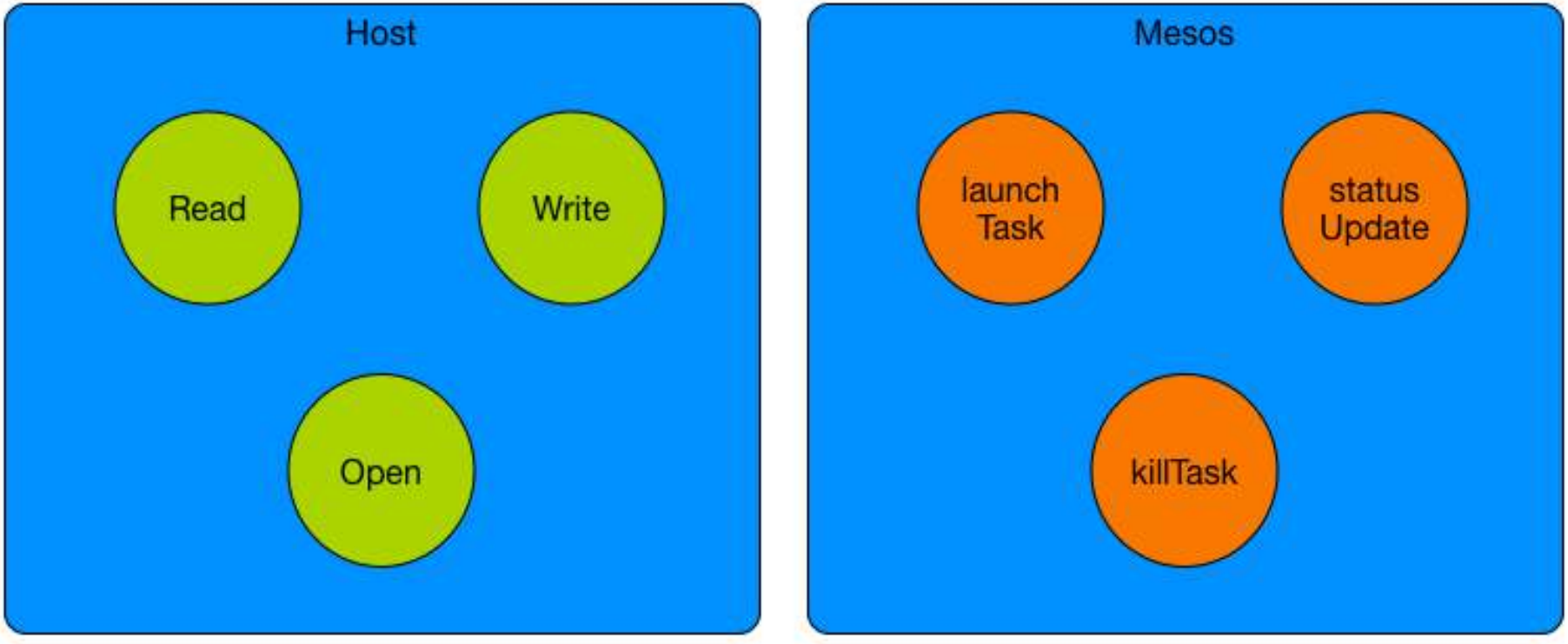
Isolation

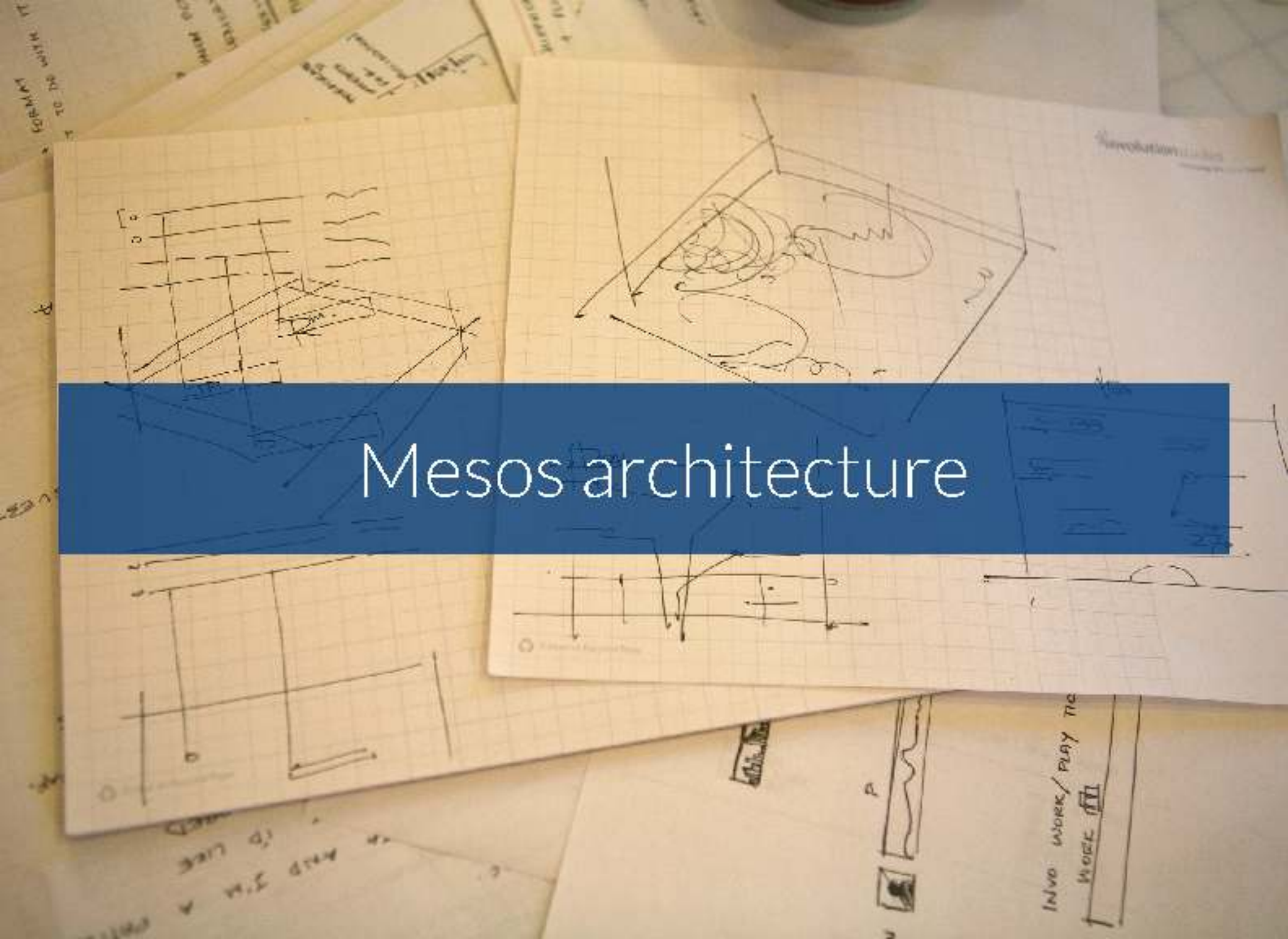

Host
Process
Process
Process
Process
Host
Container
Container
Container
Container

Resource sharing

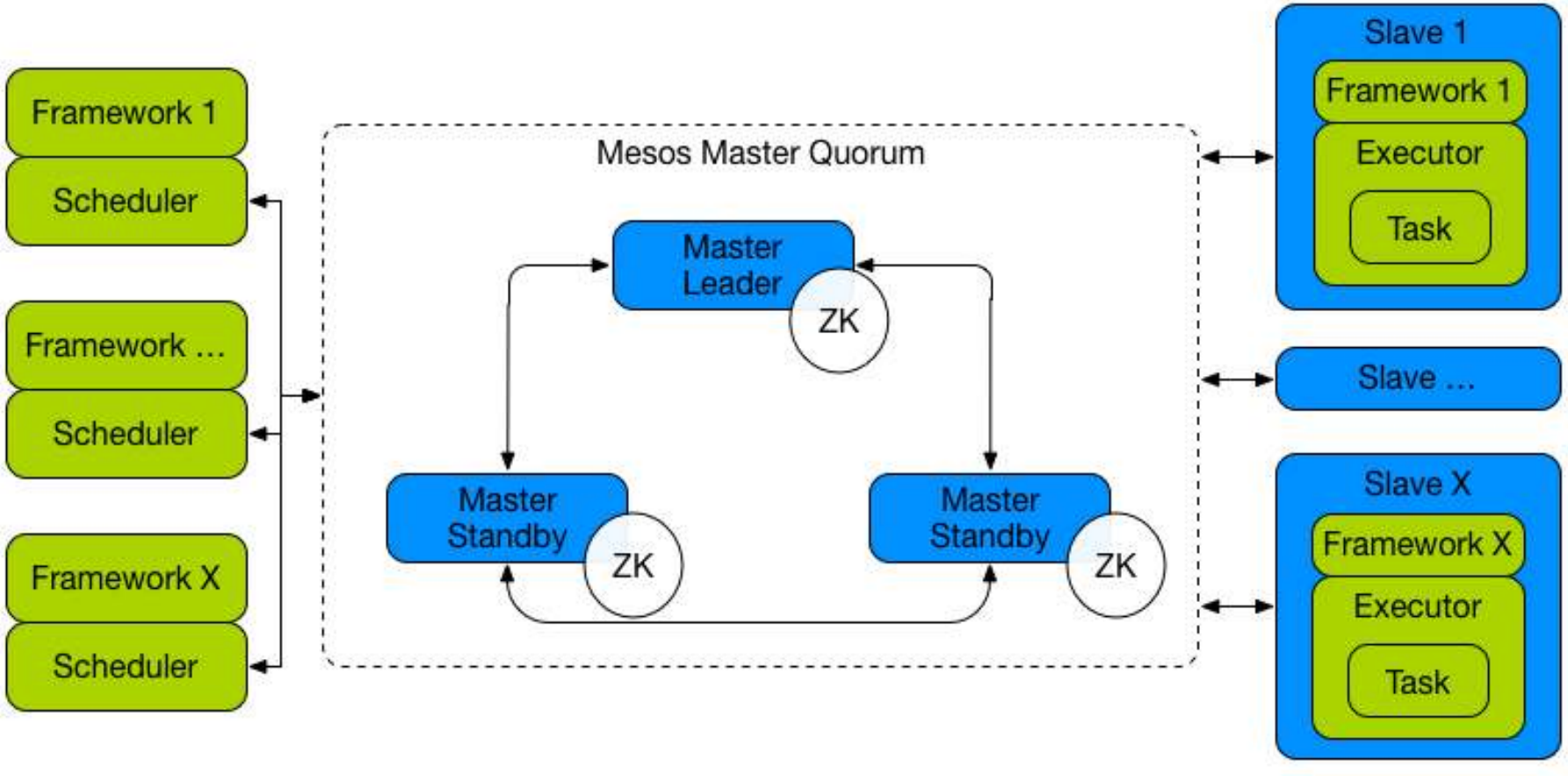


Common infrastructure

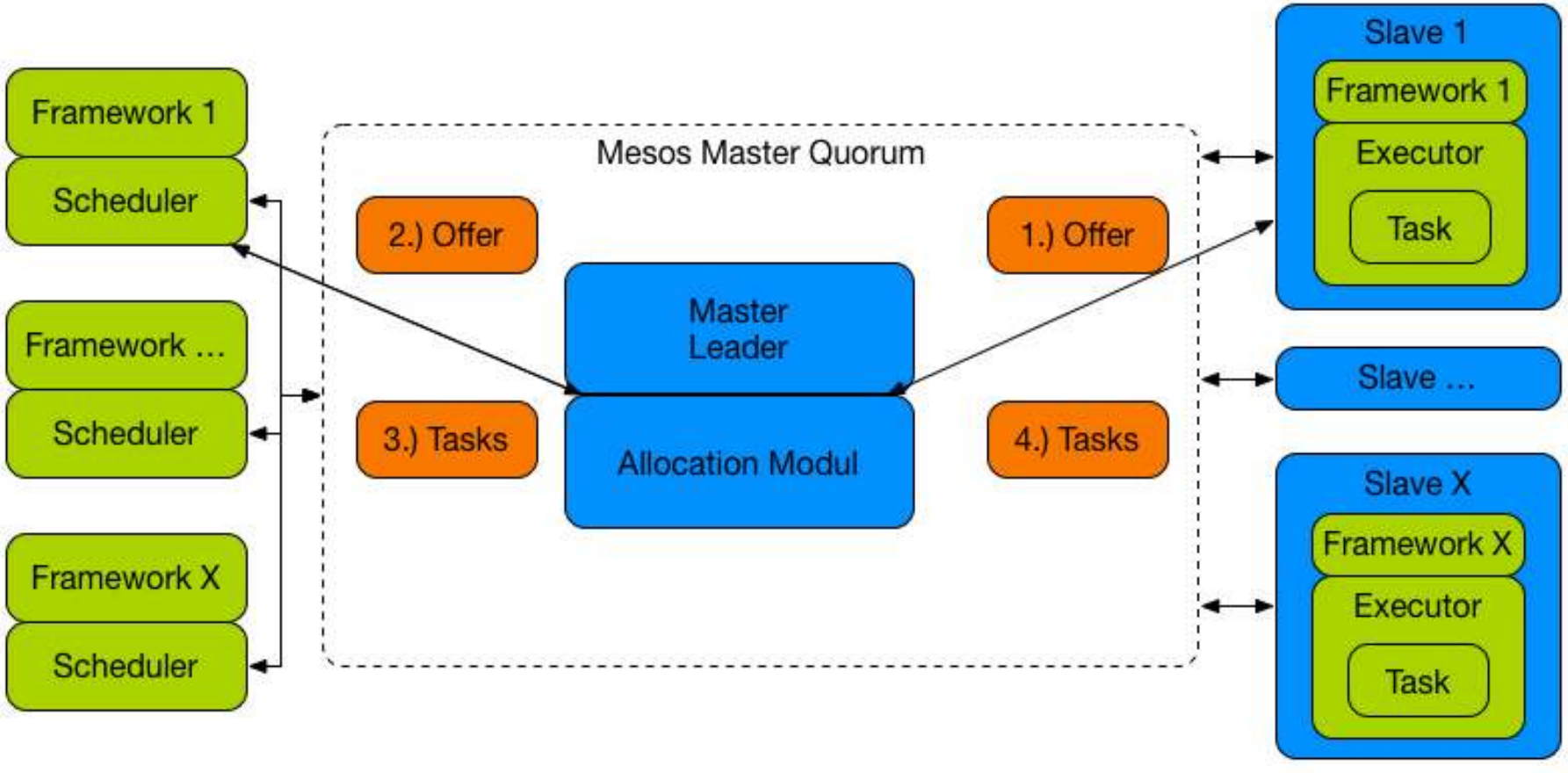



Mesos architecture

Architecture

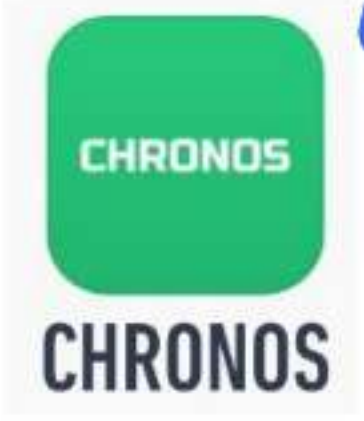


Resource offer




How to run tasks

Use an existing framework



Build your own framework

- If no framework fits
- Special needs
- C++, Python, Java, Scala, Go


Marathon

What is Marathon?

- Obviously not the sport :)
- Cluster-wide init and control system
- Cgroups or Docker
- Supports SSL and basic auth
- Per default HA (with zookeeper)

Work with Marathon

MESOSPHERE

- Listens per default on 8080
- Run Tasks over API
- Or use the fancy UI

To simplify the working

```
#On your (local) machine:
echo "Master-Pub-IP              kit-mesos-master" >> /etc/hosts
echo "Slave1-Pub-IP              kit-mesos-slave1" >> /etc/hosts
echo "Slave2-Pub-IP              kit-mesos-slave2" >> /etc/hosts
echo "Slave3-Pub-IP              kit-mesos-slave3" >> /etc/hosts
```

To simplify the working -2

```
#!/bin/bash
#On VM's (best over ssh -> you can use the hostnames):
echo "Master-Priv-IP            kit-mesos-master" >> /etc/hosts
echo "Slave1-Priv-IP            kit-mesos-slave1" >> /etc/hosts
echo "Slave2-Priv-IP            kit-mesos-slave2" >> /etc/hosts
echo "Slave3-Priv-IP            kit-mesos-slave3" >> /etc/hosts
```

```
# Pack this in a simple shell script and run
scp set_hostnames.sh gks@kit-mesos-XXXX:/tmp/set_hostnames.sh
ssh -t gks@kit-mesos-XXXX "sudo sh /tmp/set_hostnames.sh"
```

Let's start - Hello GridKA

```
{
  "id": "hello-gridka",
  "cmd": "while [ true ]; do echo 'Hello GridKA'; sleep 5; done",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 1
}
```

Let's run the task

```
curl -X POST http://kit-mesos-master:8080/v2/apps \
-d @hello-gridka.json -H "Content-type: application/json"
```

```
http --json POST http://kit-mesos-master:8080/v2/apps \
< hello-gridka.json
```

Web UI

Let's have a look at the Marathon UI (8080) and the Mesos UI (5050)

Your turn

Create a task:

- Counts from 0 to 25
- Sleeps 1s after each increment
- 1 instance

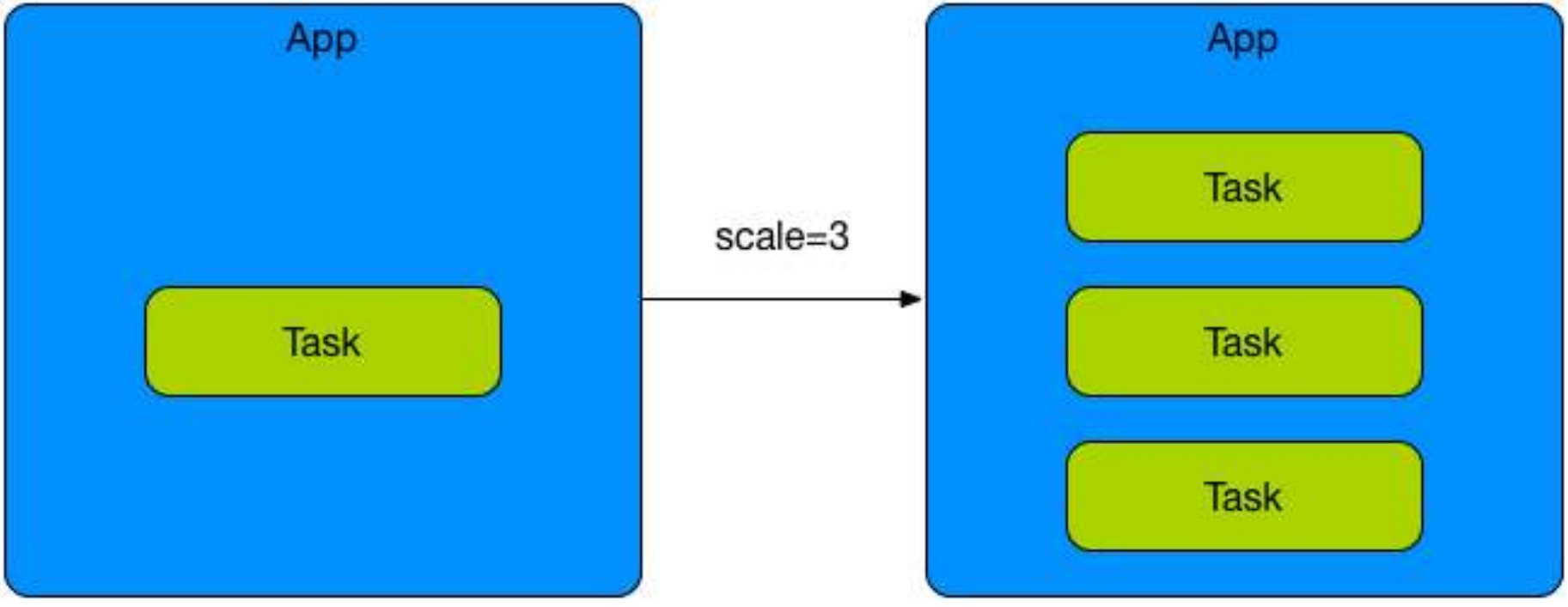
A possible solution

```
{
  "id": "counter",
  "cmd": "for i in {1..25}; do echo $i; sleep 1; done",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 1
}
```

What happens when a task finishes?

- A new task is started
- 1 Task is kept running
- Focused on long running tasks

Scale out



Scale out - API

- Adjust the config file
- HTTP PUT to API
- We can suspend apps

Scale out

```
{
  "id": "counter",
  "cmd": "for i in {1..25}; do echo $i; sleep 1; done",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 3
}
```

Submit change

```
curl -X POST http://kit-mesos-master/v2/apps/counter \
-d @counter-scale-out.json -H "Content-type: application/json"
```

```
http --json POST  http://kit-mesos-master/v2/apps/counter \
 < counter-scale-out.json
```

Your turn

Updates work the same way

- Let the app count to 100

Update app

```
{
  "id": "counter",
  "cmd": "for i in {1..100}; do echo $i; sleep 1; done",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 3
}
```

Your turn

We don't need the counter task anymore:

- Suspend the counter task
- Submit the change

Suspend counter task

```
{
  "id": "counter",
  "cmd": "for i in {1..100}; do echo $i; sleep 1; done",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 0
}
```

Delete an app

```
curl -X DELETE http://kit-mesos-master/v2/apps/counter
```

```
http DELETE http://kit-mesos-master/v2/apps/counter
```

Your turn

Remove the first app with the API:

Fetch the App ID

> Web UI

> REST API

Running Docker

Start an app which runs Docker

```
{
  "id": "docker-python-server",
  "cmd": "python3 -m http.server 8080",
  "cpus": 0.5,
  "mem": 32.0,
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "python:3",
      "network": "BRIDGE",
      "portMappings": [
        { "containerPort": 8080, "hostPort": 0 }
      ]
    }
  }
}
```

Your turn again

Create an App:

- echos your name
- Sleeps 10s
- 3 instances
- Runs in a container
- Uses busybox as image

A possible solution

```
{
  "id": "docker-hello",
  "cmd": "echo Johannes && sleep 10",
  "cpus": 0.5,
  "mem": 32.0,
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "busybox"
    }
  },
  "instances": 3
}
```

Your turn

We don't need the docker-hello task anymore:

- Delete the docker-hello task

We can also use remote resources

```
{
  "id": "hello-gridka-remote",
  "cmd": "./hello-gridka.sh",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 1,
  "uris": [
        "http://kit-mesos-master:9000/hello-gridka.sh"
    ]
}
```

Your turn

- Log into the master
- Create a dir "www"
- Create in "www" the hello-gridka.sh
- Script echos every 5s "hello"
- Create a Marathon definiton
- And run your app

```
python -m SimpleHTTPServer 9000
```

Solution

```
{
  "id": "hello-gridka-remote",
  "cmd": "chmod u+x ./hello-gridka.sh && ./hello-gridka.sh",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 1,
  "uris": [
        "http://kit-mesos-master:9000/hello-gridka.sh"
    ]
}
```

Supported URI types

- file
- http(s)
- ftp(s)
- hdfs
- s3

Remote packed resources

- .tar.gz (.tgz)
- .tar.bz2 (.tbz2)
- .tar.xz (.txz)
- .zip

Your turn

- Pack the script into a tar.gz (tar cfz)
- Modify the app
- Name it "hello-gridka-repacked"
- Run the app

Solution

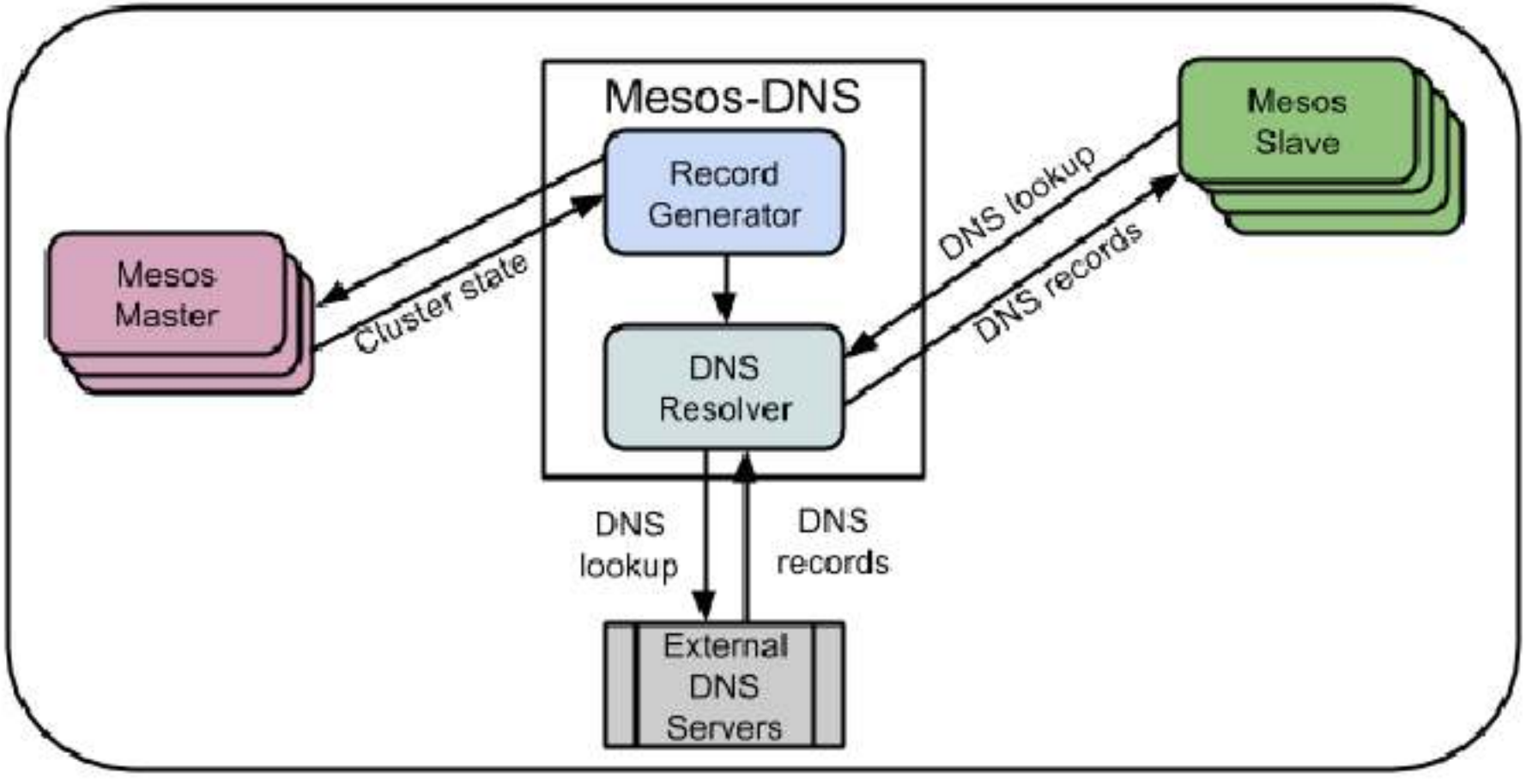
```
{
  "id": "hello-gridka-repacked",
  "cmd": "chmod u+x ./hello-gridka.sh && ./hello-gridka.sh",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 1,
  "uris": [
        "http://kit-mesos-master:9000/app.tar.gz"
    ]
}
```

Service discovery

The good news

- Mesos-DNS
- DNS-based service discovery
- Creates dynamic DNS entries
- Periodically queries the Mesos master(s)

Mesos DNS


Mesos-DNS
Record Generator
DNS Resolver
Mesos Master
Mesos Slave
Cluster state
DNS lookup
DNS records
DNS lookup
DNS records
External DNS Servers

Running Mesos-DNS on slave 1

- Login into slave 1
- Create /usr/local/mesos-dns
- Copy/Download mesos-dns
- Now we create the Configuration

```
https://github.com/inovex/GridKA-SDDC-2015/tree/master/binaries
```

Mesos-DNS cofiguration file

```
{
        "masters": ["kit-mesos-master:5050"],
        "refreshSeconds": 60,
        "ttl": 60,
        "domain": "mesos",
        "port": 53,
        "resolvers": ["8.8.8.8"],
        "timeout": 5,
        "dsnon": true,
        "externalon": true,
        "listener": "kit-mesos-slave1"
}
```

Let's start mesos-dns

```
sudo ./mesos-dns -config=./mesos-dns-config.json &
```

Validate it

```
dig leader.mesos  @kit-mesos-slave1 +short
```

Configure all machines

SSH into each machine

```
sudo sed -i '1s/^/nameserver Slave1-IP\n /' /etc/resolv.conf
```

or SSH

```
ssh -t root@$Slave-IP "command above"
```

Now we don't need to pass the DNS Server

```
dig leader.mesos
```

Obviously this isn't the best way

The simple solution

Mesos-DNS definition

```
{
    "cmd": "/usr/local/mesos-dns/mesos-dns -config=/usr/local/mesos-
    "cpus": 1.0,
    "mem": 1024,
    "id": "mesos-dns",
    "instances": 1,
    "constraints": [["hostname", "CLUSTER", "$SLAVE1-PUB-IP"]]
}
```

How are the DNS names are build

- task
- framework
- domain

Example

- docker-python-server.marathon.mesos

There is also a Mesos-DNS-API

```
curl http://kit-mesos-slave1:8123/v1/version
curl http://kit-mesos-slave1:8123/v1/config
curl http://kit-mesos-slave1:8123/v1/hosts/$host
curl http://kit-mesos-slave1:8123/v1/services/$service
```

Your turn

- Start the docker-python-server again
- Fetch the IP over the API
- Fetch the IP with dig
- Fetch the SRV Record with dig
- Fetch the SRV over the API

```
dig SRV _task._protocol.framework.domain
```

Solution

API

```
curl .../v1/hosts/docker-python-server.marathon.mesos
```

IP with dig

```
dig docker-python-server.marathon.mesos +short
```

SRV record with dig

```
dig _docker-python-server._tcp.marathon.mesos +short SRV
```

SRV record over API

```
curl .../v1/services/_docker-python-server._tcp.marathon.mesos
```

DNS Server and Docker

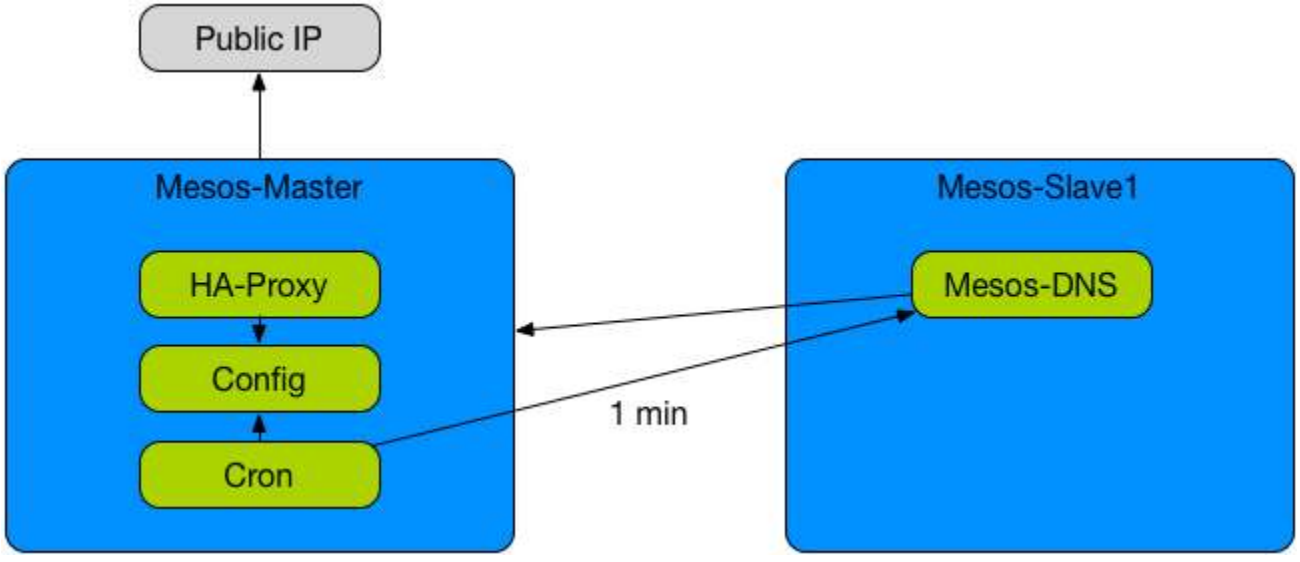
Restart Docker

```
sudo systemctl restart docker
```

Check the status of Docker

```
sudo systemctl status docker
```

HAProxy and Mesos-DNS



HAProxy on the Master

Installation

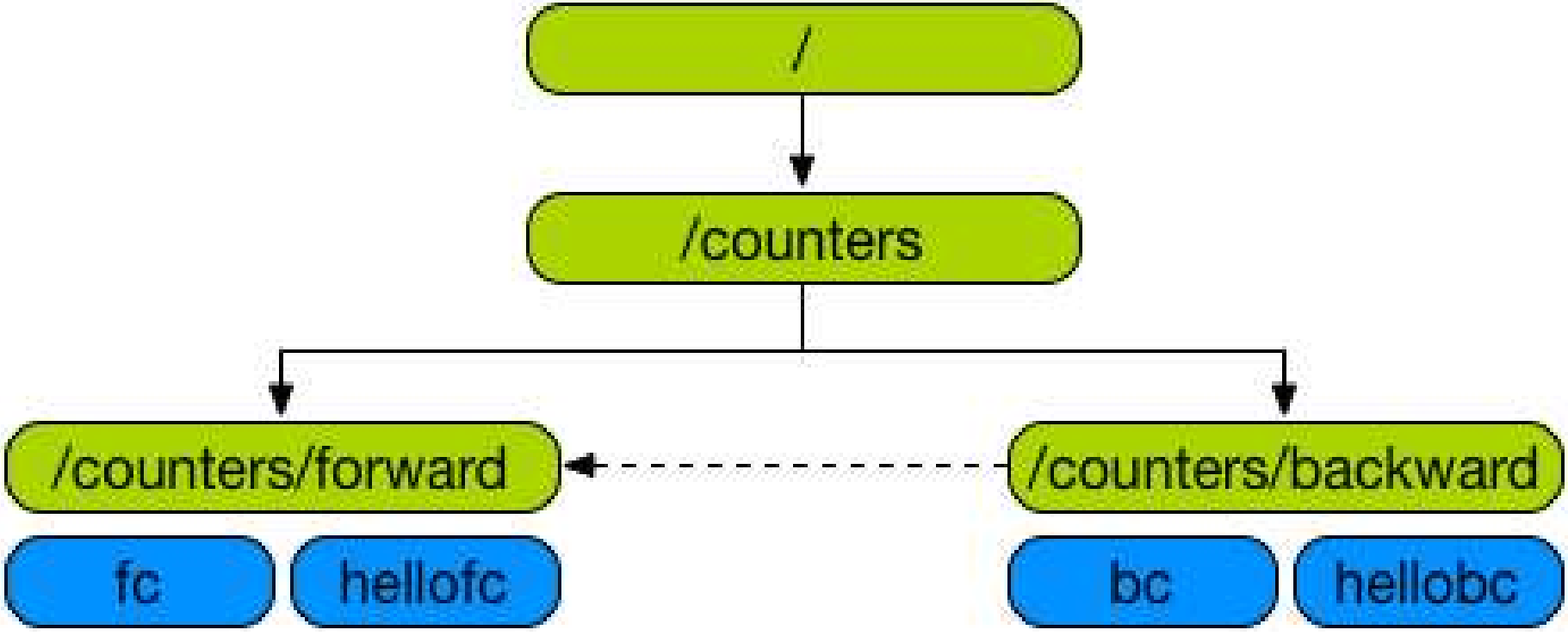
```
curl -O  https://raw.githubusercontent.com/mesosphere/marathon/maste
chmod +x ./haproxy-marathon-bridge
./haproxy-marathon-bridge install_haproxy_system kit-mesos-master-IP
```

Verify

```
cat /etc/haproxy-marathon-bridge/marathons
ll /usr/local/bin/haproxy-marathon-bridge
cat /etc/cron.d/haproxy-marathon-bridge
systemctl status haproxy
```


Grouping apps

Grouping apps



Group demo app

```
{
  "id": "/counters",
  "groups": [
    {
      "id": "/counters/forward",
      "apps": [
         {  "id": "/counters/forward/fc", ... },
         {  "id": "/counters/forward/hellofc", ... }
       ]
    },{
      "id": "/counters/backward",
      "dependencies": ["/counters/forward"],
      "apps": [
         {  "id": "/counters/backward/bc", ... },
         {  "id": "/counters/backward/hellobc", ... }
      ]
    }
```

Your turn

- fc counts from 1 - 100
- hellofc says "Hello GrikdKA" 100x
- bc counts from 100 - 1
- hellobc says "Hello GrikdKA" reverse 100x
- 1 instance per app
- "cpus": 0.1, "mem": 10.0

Possible solution ../forward

```
"id": "/counters/forward",
"apps": [
 {
    "id": "/counters/forward/fc",
    "cmd": "for i in {1..100}; do echo $i; sleep 1; done",
    "cpus": 0.1,
    "mem": 10.0,
    "instances": 1
  },{
    "id": "/counters/forward/hellofc",
    "cmd": "for i in {1..100}; do echo 'Hello GrikdKA'; sleep 1; don
    "cpus": 0.1,
    "mem": 10.0,
    "instances": 1
  }
```

Possible solution ../backward

```
"id": "/counters/backward",
"dependencies": ["/counters/forward"],
"apps": [
 {
    "id": "/counters/backward/bc",
    "cmd": "for i in {100..1}; do echo $i; sleep 1; done",
    "cpus": 0.1,
    "mem": 10.0,
    "instances": 1
 },{
    "id": "/counters/backward/hellobc",
    "cmd": "for i in {1..100}; do echo 'Hello GrikdKA' | rev; sleep
    "cpus": 0.1,
    "mem": 10.0,
    "instances": 1
 }
```

Scale a group

```
http --json .../v2/groups/counters < scale.json
```

Content of scale.json

```
cat scale.json
{
  "scaleBy": 2
}
```

Delete the hole group

```
http --json DELETE kit-mesos-master:8080/v2/groups/counters
```

Groups and Mesos DNS

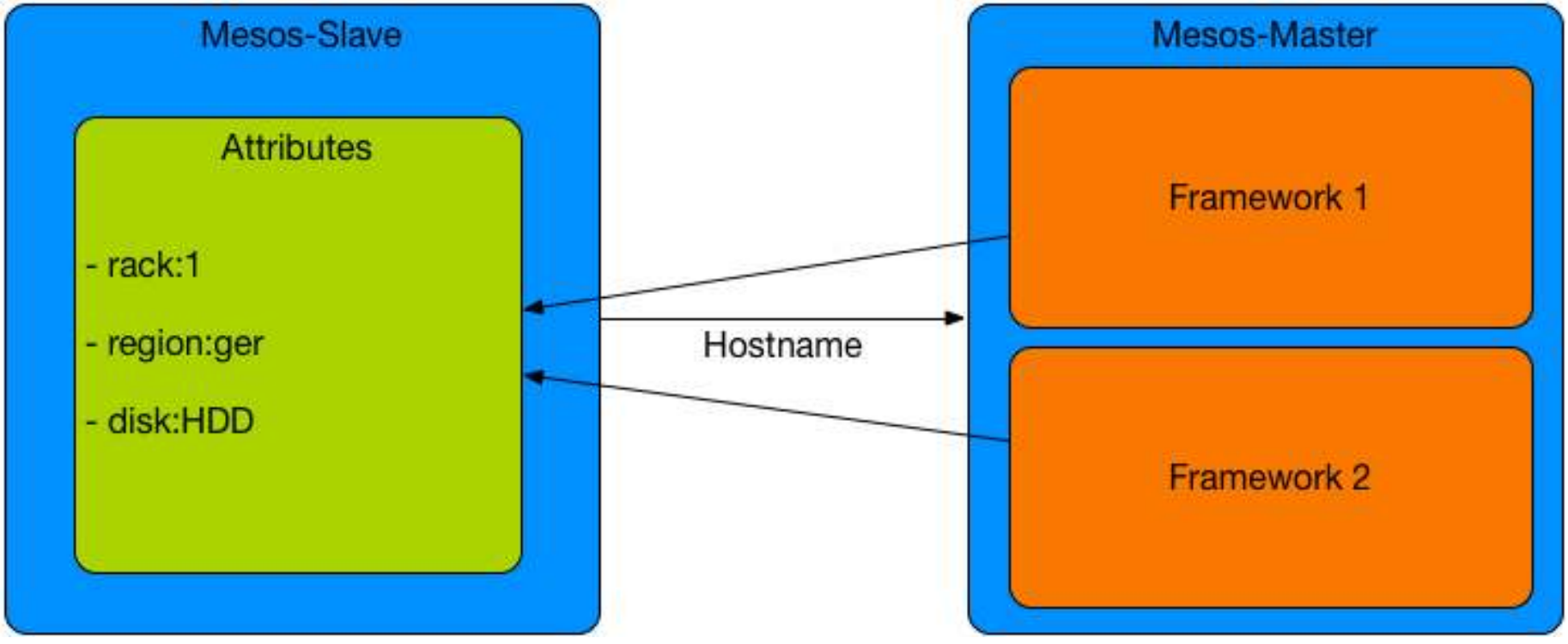
- App-name-Group-ID(s).framework.domain

Example

- hellofc-forward-counters.marathon.mesos


Constraints

Fields


Mesos-Slave
Attributes
- rack:1
- region:ger
- disk:HDD
Hostname
Mesos-Master
Framework 1
Framework 2

Setting an attribute

As Parameter

```
./mesos-slave --attributes="rack:1;region:ger;disk:hdd"
```

Inside /etc/mesos-slave/attributes

```
echo "rack:1;region:ger;disk:hdd" > /etc/mesos-slave/attributes
```

Operators

- UNIQUE
- CLUSTER
- GROUP_BY
- LIKE
- UNLIKE

UNIQUE hostname

```
{
  "id": "hello-gridka-unique",
  "cmd": "while [ true ]; do echo 'Hello GridKA'; sleep 5 ; done",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 4,
  "constraints": [["hostname", "UNIQUE"]]
}
```

CLUSTER operator

```
{
  "id": "hello-gridka-cluster",
  "cmd": "while [ true ]; do echo 'Hello GridKA'; sleep 5 ; done",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 3,
  "constraints": [["disk", "CLUSTER", "hdd"]]
}
```

GROUP_BY operator

```
{
  "id": "hello-gridka-groupby",
  "cmd": "while [ true ]; do echo 'Hello GridKA'; sleep 5 ; done",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 4,
  "constraints": [["rack", "GROUP_BY", "3"]]
}
```

LIKE operator

```
{
  "id": "hello-gridka-like",
  "cmd": "while [ true ]; do echo 'Hello GridKA'; sleep 5 ; done",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 4,
  "constraints": [["region", "LIKE", "ger"]]
}
```

UNLIKE operator

```
{
  "id": "hello-gridka-unlike",
  "cmd": "while [ true ]; do echo 'Hello GridKA'  sleep 5 ; done",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 4,
  "constraints": [["region", "UNLIKE", "us"]]
}
```

Your turn

Task 1

- Says Hello
- Runs on Slave 2

Task 2

- Says Bye
- Runs on Slave 3

Task 1

```
{
  "id": "hello-slave-2",
  "cmd": "while [ true ] ; do echo 'Hello' ; sleep 5 ; done",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 1,
  "constraints": [["hostname", "CLUSTER", "$Slave-2-IP"]]
}
```

Task 2

```
{
  "id": "bye-slave-3",
  "cmd": "while [ true ] ; do echo 'Bye' ; sleep 5 ; done",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 1,
  "constraints": [["hostname", "LIKE", "$Slave-3-IP"]]
}
```

Or even

```
{
  "id": "bye-slave-3",
  "cmd": "while [ true ] ; do echo 'Bye' ; sleep 5 ; done",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 1,
  "constraints": [["hostname", "Unlike", "XXX.XXX.XXX.XX[Y]"]]
}
```


Containers and Mesos in depth

Volumes

```
{
  "container": {
        ...,
    "volumes": [
      {
        "containerPath": "/etc/config",
        "hostPath": "/var/data/config",
        "mode": "RO"
      },{
        "containerPath": "/etc/files",
        "hostPath": "/var/data/files",
        "mode": "RW"
      }
    ]
  }
}
```

Task 1

- Writes the current time every 5s into a file (/tmp/..)

Task 2

- Reads the file every 5s

Task 1

```
{
  "id": "write-to-disk",
  "cmd": "while [ true ]; do echo $(date) > /tmp/date.txt; sleep 5;
  "cpus": 0.5,
  "mem": 64.0,
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "busybox"
    },
    "volumes": [
      {
        "containerPath": "/tmp",
        "hostPath": "/tmp",
        "mode": "RW"
      }
    ]
```

Task 2

```
{
  "id": "read-from-disk",
  "cmd": "while [ true ]; do  cat /tmp/date.txt; sleep 5; done",
  "cpus": 0.5,
  "mem": 64.0,
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "busybox"
    },
    "volumes": [
      {
        "containerPath": "/tmp",
        "hostPath": "/tmp",
        "mode": "RO"
      }
    ]
```

Networking

```
{
  ...,
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "python:3",
      "network": "BRIDGE",
      "portMappings": [
        { "containerPort": 8080, "hostPort": 0, "servicePort": 9000,
        { "containerPort": 161, "hostPort": 0, "protocol": "udp"}
      ]
    }
  }
}
```

Your turn

Create a docker hello web server

- Use "johscheuer/go-webserver" as docker image
- Server runs on port 8000 in the container
- Host port should be random
- Service port should be 80
- 3 instances

Possible solution

```
{
  "id": "hello-server",
  "cpus": 0.5,
  "mem": 32.0,
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "johscheuer/go-webserver",
      "network": "BRIDGE",
      "portMappings": [
        { "containerPort": 8000, "servicePort": 80 }
      ]
    }
  },
  "instances": 3
}
```

```
curl kit-mesos-master
```

Private Docker registry

- Supported
- Authentication
- Private images
- Can run as an app :)

Arguments

```
...,
"args": [
    "-arg1", "value1",
    "-arg2", "value2"
  ],
"cpus": 0.2,
"mem": 32.0,
...
```

Your turn

Use arguments

- Use "johscheuer/arg-example:v1.0" as docker image
- Arguments are "-name" and "-street"
- 1 instance

Possible solution

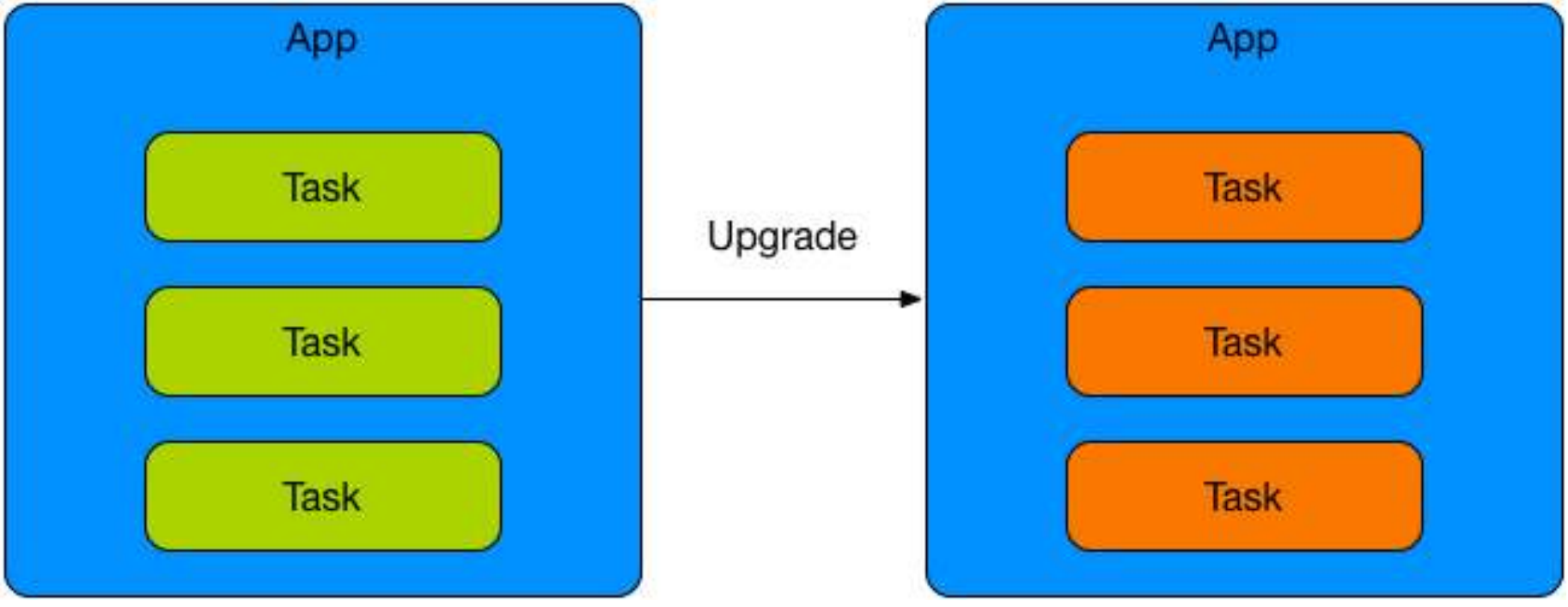
```
{
  "id": "arg-example",
  "cpus": 0.2,
  "mem": 32.0,
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "johscheuer/arg-example:v1.0"
    }
  },
  "instances": 1,
   "args": [
      "-name", "Bert",
      "-street", "Sesame Street"
    ]
}
```

Privileged and parameters

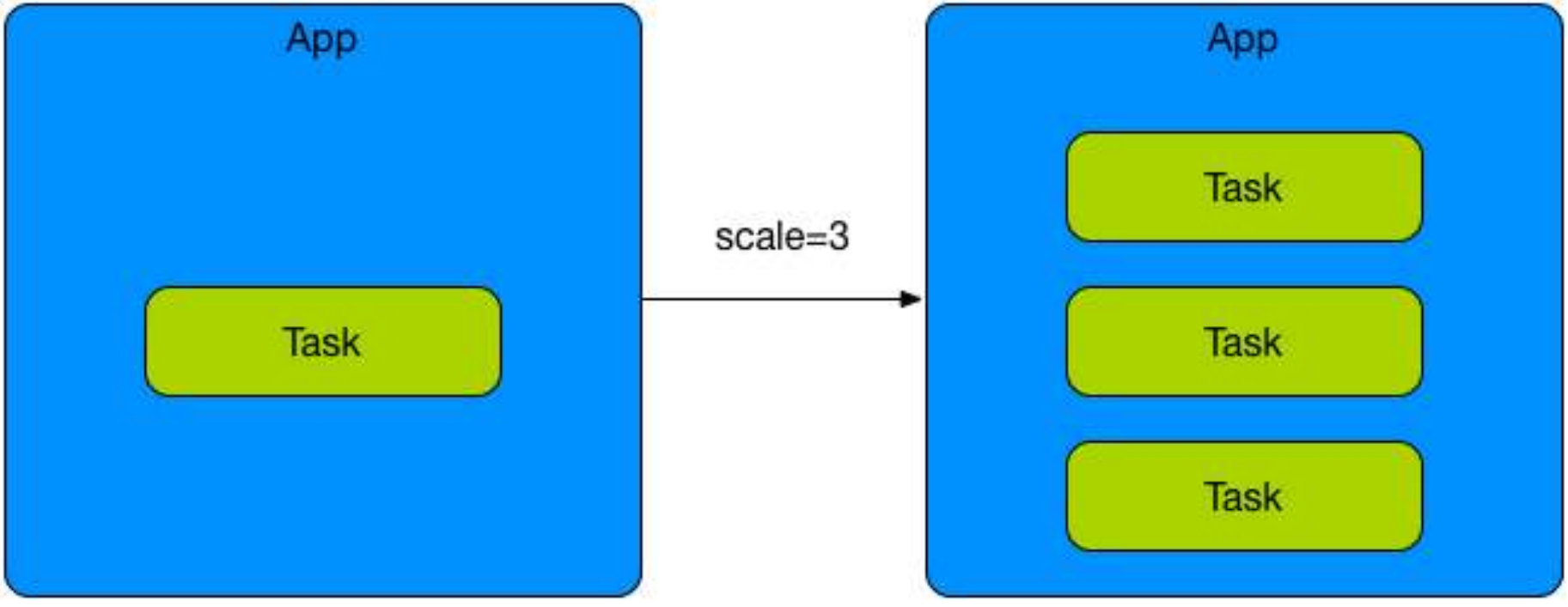
```
...,
"container": {
    "docker": {
        "image": "busybox",
        "privileged": true,
        "parameters": [
            { "key": "name", "value": "busy-container" },
            { "key": "restart", "value": "no" }
        ]
    },
    "type": "DOCKER"
},
...,
```


Deployments extended

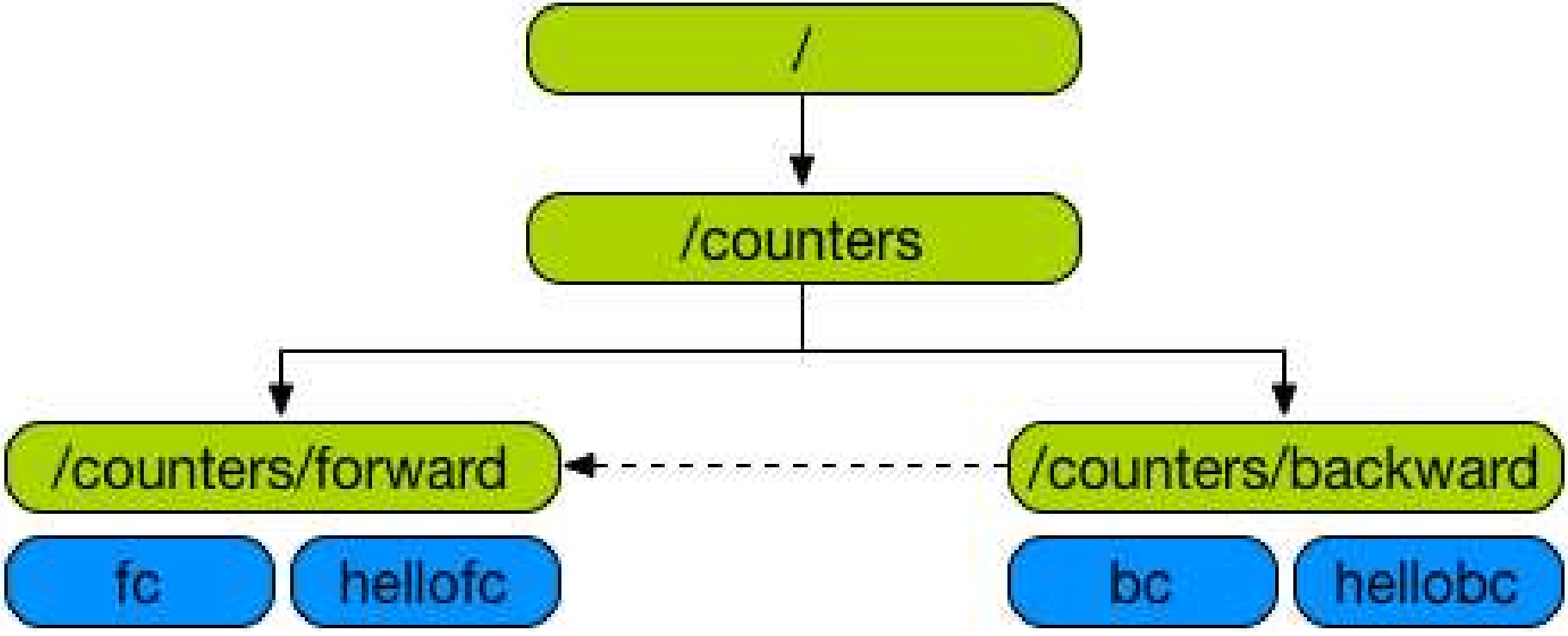
Upgrade



Scale



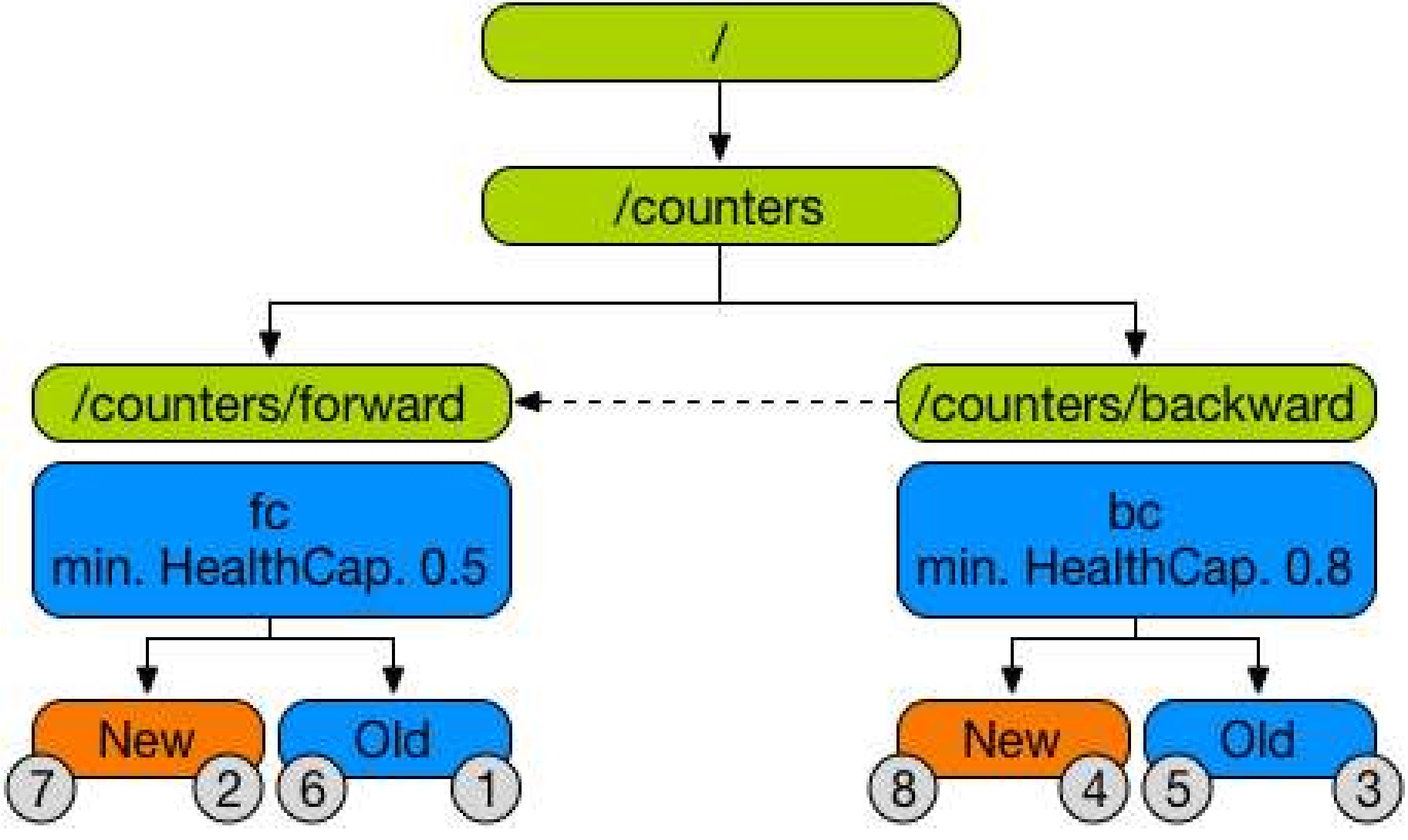
Dependencies



Rolling updates

- minimumHealthCapacity
- Start/Stop phases
- Dependencies respected

Rolling updates and dependencies



Force a Deployment

- Only one deployment at a time
- Additional changes have to wait
- Or you force the deployment
- Can lead to inconsistent state


Why could it fail?

Your turn

```
{
  "id": "hello-gridka-200",
  "cmd": "while [ true ]; do echo 'Hello GridKA'; sleep 5 ; done",
  "cpus": 0.1,
  "mem": 10.0,
  "instances": 200
}
```

```
http --json GET kit-mesos-master:8080/v2/deployments
```

Your turn 2

- Of course this deployment will not succeed
- We have to stop it
- Fetch the deployment ID
- And stop the deployment


Health checks

Example

```
{
  "protocol": "HTTP",
  "path": "/api/health",
  "portIndex": 0,
  "gracePeriodSeconds": 300,
  "intervalSeconds": 60,
  "timeoutSeconds": 20,
  "maxConsecutiveFailures": 3,
  "ignoreHttp1xx": false,
}
```

Your turn

- Take the hello-server app
- Implement a health check
- Start/Upgrade your application

Possible solution

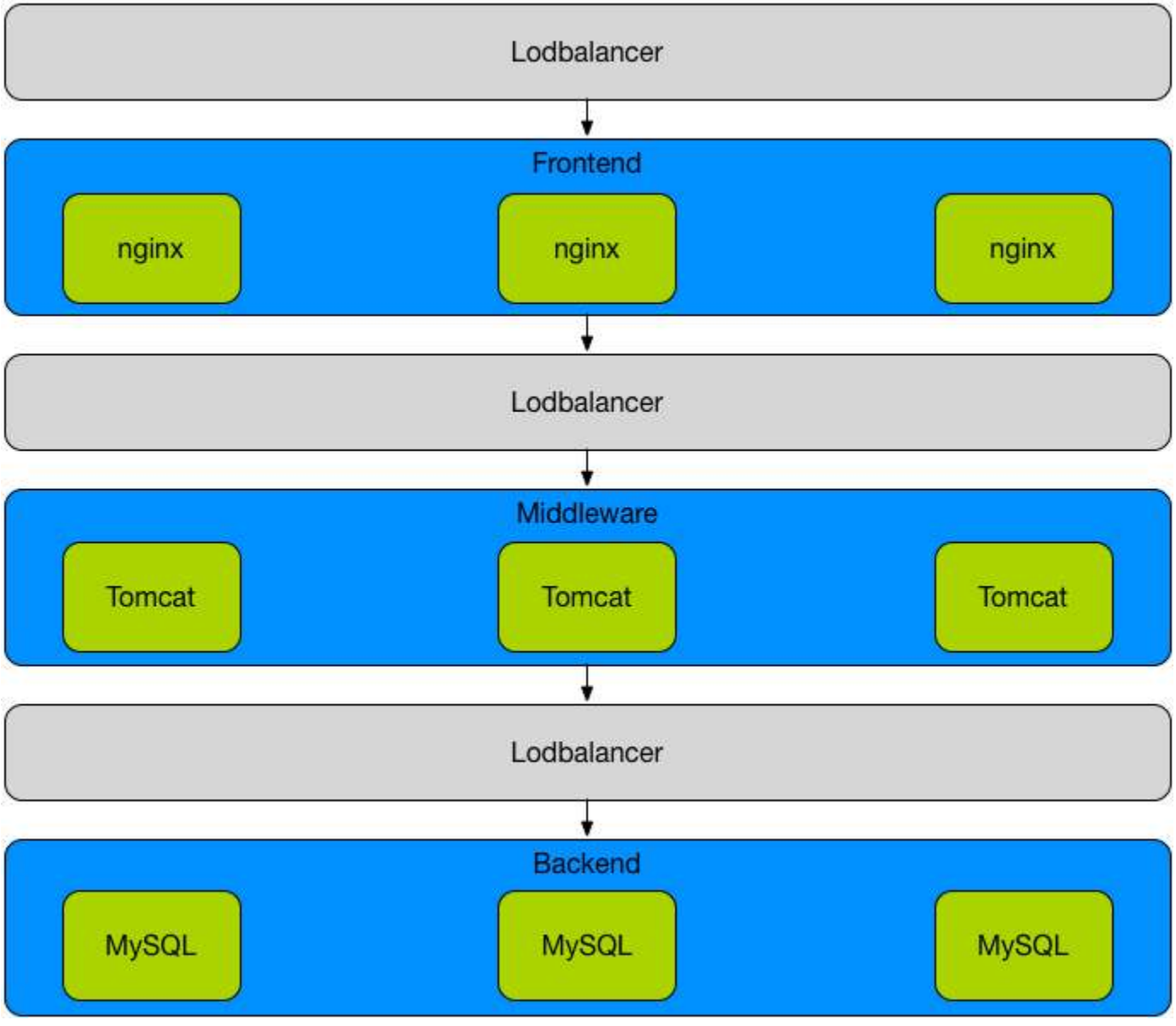
```
...,
"healthChecks": [
    {
        "protocol": "HTTP",
        "path": "/",
        "gracePeriodSeconds": 3,
        "intervalSeconds": 20,
        "portIndex": 0,
        "timeoutSeconds": 10,
        "maxConsecutiveFailures": 3
    }
],
...
```

Possible solution 2

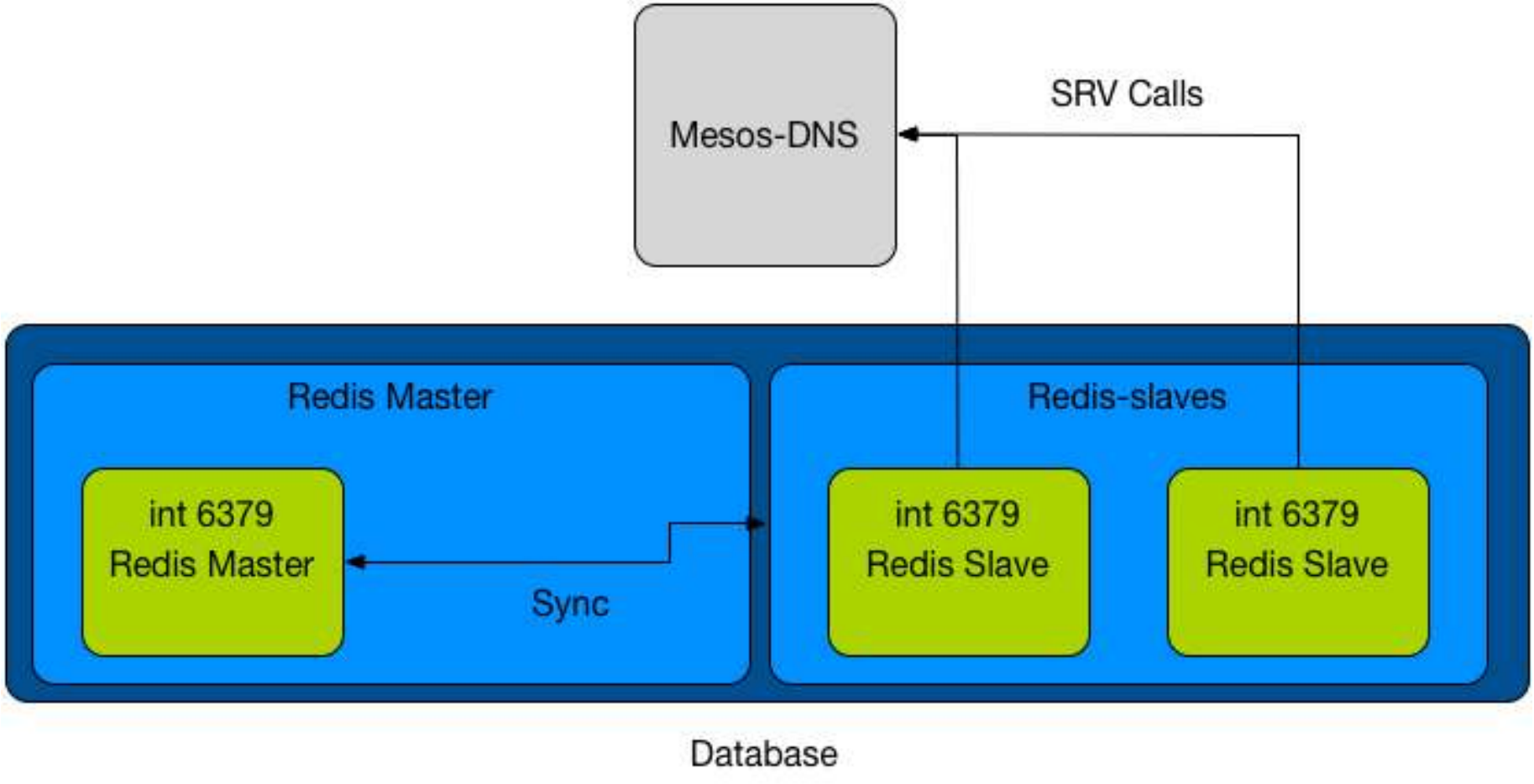
```
...,
"healthChecks": [
    {
        "protocol": "COMMAND",
        "command": {"value": "curl -f -X GET http://$HOST:$PORT0"},
        "gracePeriodSeconds": 300,
        "intervalSeconds": 60,
        "timeoutSeconds": 20,
        "maxConsecutiveFailures": 3
    }
],
...
```

Example web stack

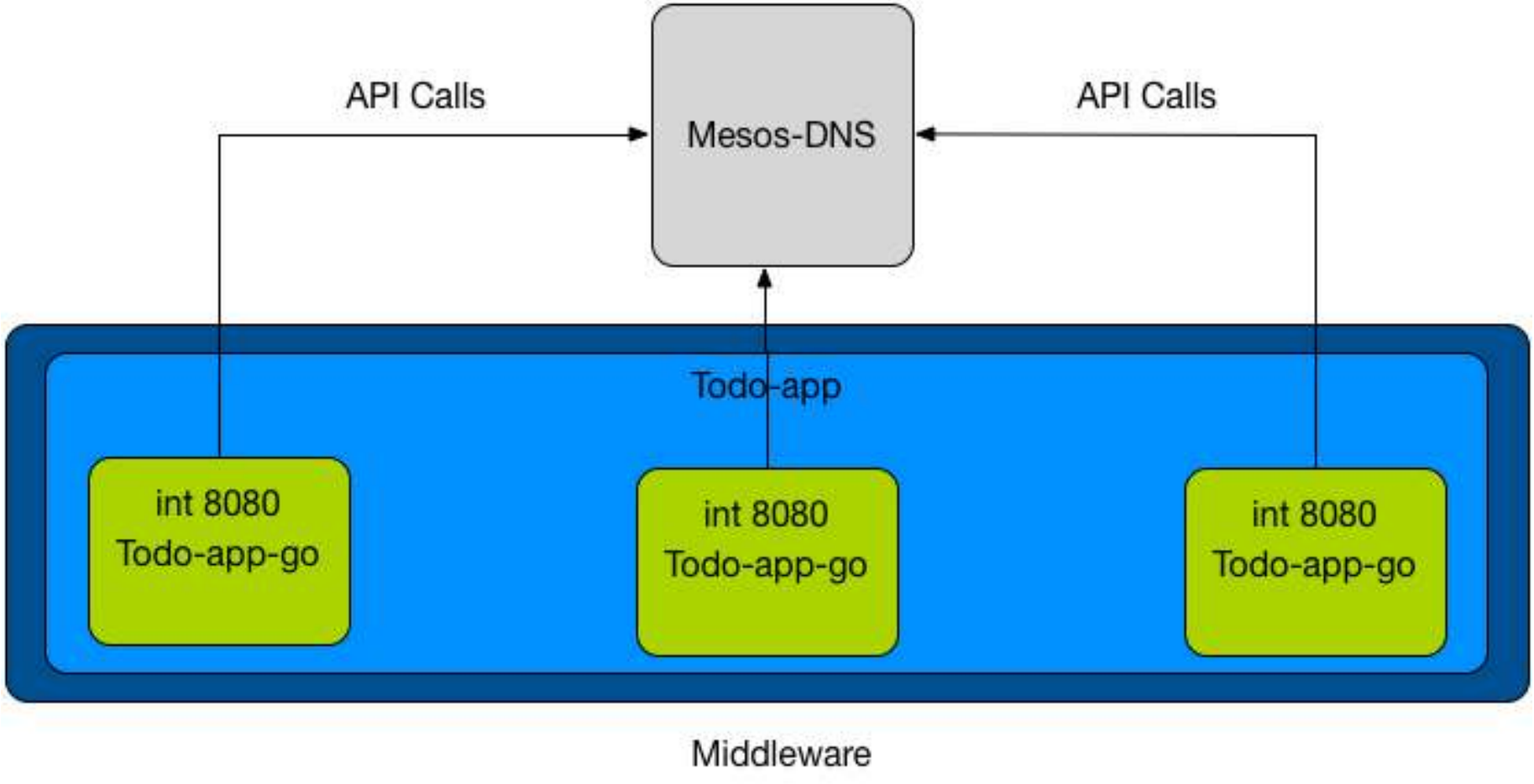
Classical


Lodbalancer
Frontend
nginx
nginx
nginx
Lodbalancer
Middleware
Tomcat
Tomcat
Tomcat
Lodbalancer
Backend
MySQL
MySQL
MySQL

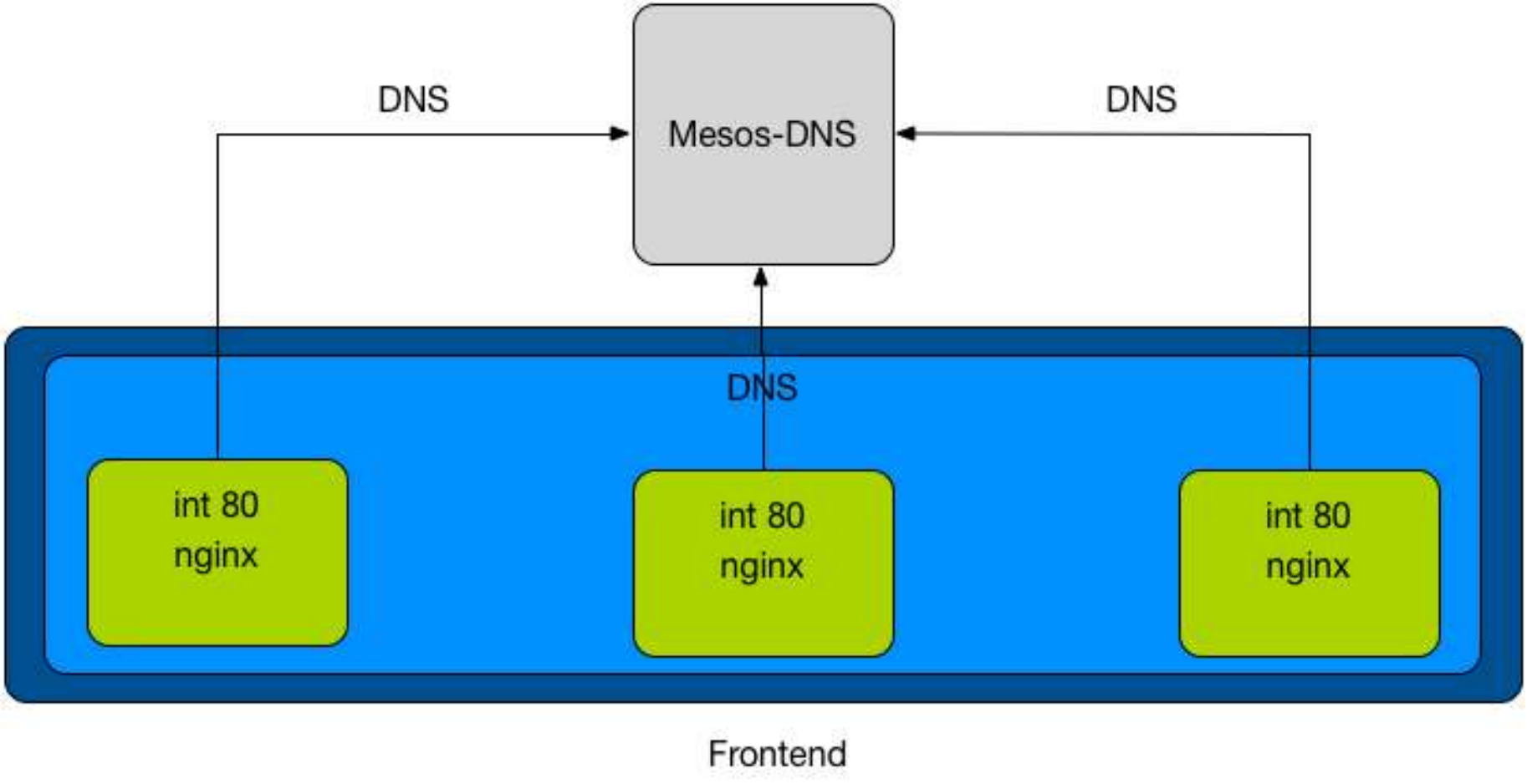
Modern database



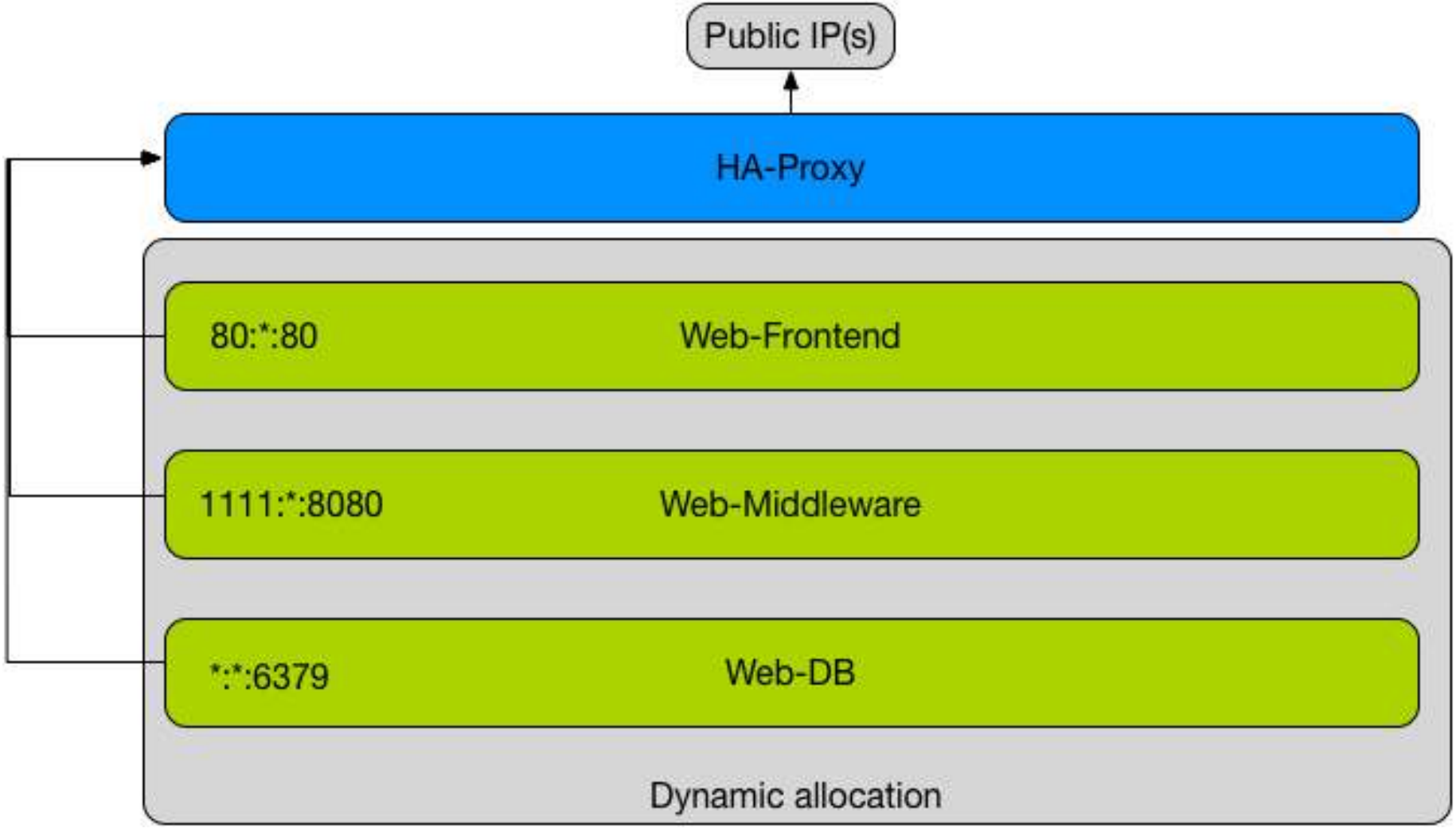
Modern middleware



Modern frontend

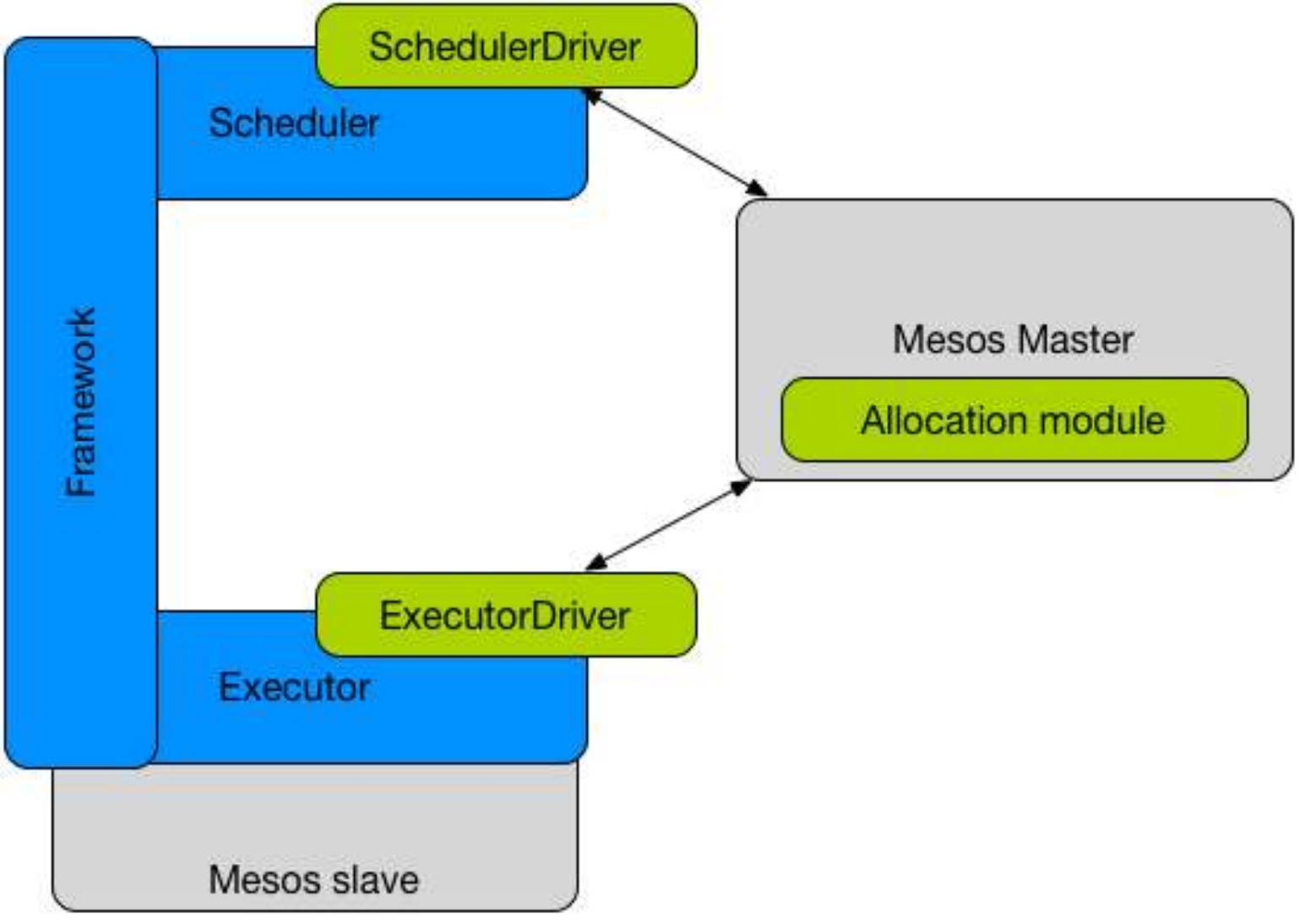


Big Picture

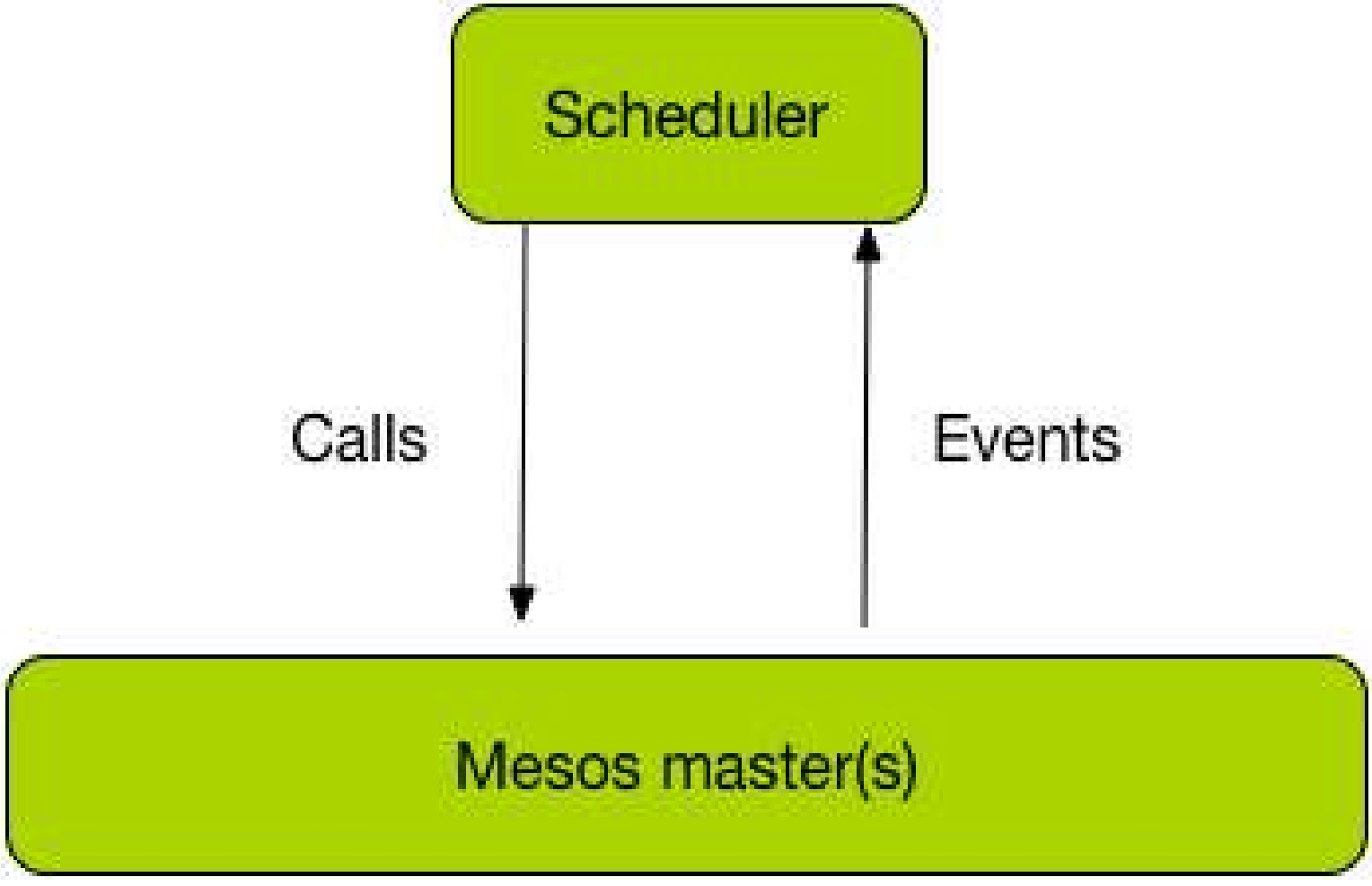

Public IP(s)
HA-Proxy
80:*:80
Web-Frontend
1111:*:8080
Web-Middleware
::6379
Web-DB
Dynamic allocation

Build your own framework

Architecture of a framework



Calls and events

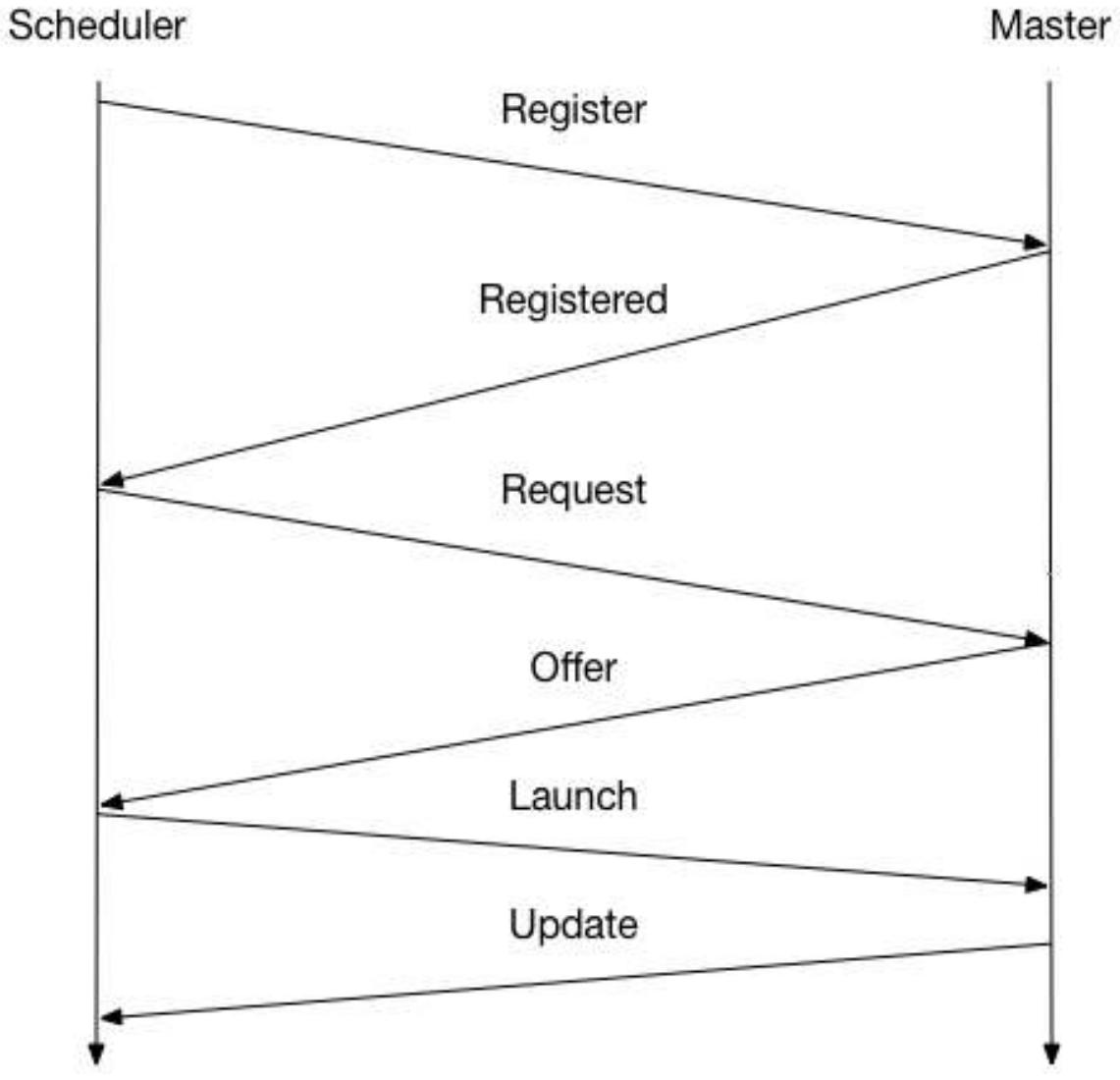


Scheduler API (the important ones)

- registered
- resourceOffers
- statusUpdate

Complete API

Communication Scheduler - Master



Languages

Get the python dev setup

```
wget http://downloads.mesosphere.io/master/centos/7/mesos-0.23.0-py2
easy_install mesos-0.23.0-py2.7-linux-x86_64.egg
```


Let's start with a simple Hello World

The imports

```
#!/usr/bin/env python
import sys
import logging
import uuid
import time
# Mesos imports
from mesos.interface import Scheduler, mesos_pb2
from mesos.native import MesosSchedulerDriver
```

The Scheduler - registered

```
class HelloWorldScheduler(Scheduler):
    '''
    Called when the scheduler successfully
    registers with a Mesos master
    Contains a unique Framework ID generate by the master
    '''
    def registered(self, driver, framework_id, master_info):
      logging.info("Registered with framework id: %s on: %s",
           framework_id, master_info.hostname)
```

The Scheduler - ressourceOffers

```
def resourceOffers(self, driver, offers):
    logging.info("Recieved resource offers: %s",
                 [o.id.value for o in offers])
    # whenever we get an offer
    # we accept it and use it to launch a task that
    # just echos hello world to stdout
    for offer in offers:
        task = new_task(offer)
        task.command.value = "while [ true ]; do echo hello world &&
        time.sleep(5)
        logging.info("Launching task %s "
                     "using offer %s.",
                     task.task_id.value,
                     offer.id.value)
        driver.launchTasks(offer.id, [task])
```

The Scheduler - statusUpdate

```
def statusUpdate(self, driver, update):
    '''
    when a task is started, over,
    killed or lost (slave crash, ....), this method
    will be triggered with a status message.
    '''
    logging.info("Task %s is in state %s" %
                 (update.task_id.value,
                  mesos_pb2.TaskState.Name(update.state)))
```

The Scheduler - new Task

```
def new_task(offer):
    task = mesos_pb2.TaskInfo()
    task.task_id.value = str(uuid.uuid4())
    task.slave_id.value = offer.slave_id.value
    task.name = "HelloWorld {}".format(task.task_id.value)

    cpus = task.resources.add()
    cpus.name = "cpus"
    cpus.type = mesos_pb2.Value.SCALAR
    cpus.scalar.value = 0.1

    mem = task.resources.add()
    mem.name = "mem"
    mem.type = mesos_pb2.Value.SCALAR
    mem.scalar.value = 10
```

Mesos Proto is your friend

The main method

```
if __name__ == '__main__':
  framework = mesos_pb2.FrameworkInfo()
  framework.user = ""  # Have Mesos fill in the current user.
  framework.name = "hello-world"
  driver = MesosSchedulerDriver(
      HelloWorldScheduler(),
      framework,
      "zk://localhost:2181/mesos"  # assumes running on the master
  )
  driver.run()
```

Let's run the framework

Log into the Master

```
python hello_mesos.py
```

What's next?

- We want to exit the framework
- We want to run tasks only if the offer fits
- We want to run only a certain amount of tasks
- We want to run containers (Docker)
- We want to run different commands
- We want to run an executor
- We want to make the framework configurable

Run framework asynchronously

```
# Inside the main method
    driver.start()
    logging.info("Listening for Ctrl-C")
    signal.signal(signal.SIGINT, shutdown)
    while True:
        time.sleep(5)

# Defnied as method
def shutdown(signal, frame):
    logging.info("Shutdown signal")
    driver.stop()
    time.sleep(5)
    sys.exit(0)
```

Let's run the framework

Log into the Master

```
python hello_mesos_exitable.py
```

What's next?

- We want to exit the framework ✓
- We want to run tasks only if the offer fits
- We want to run only a certain amount of tasks
- We want to run containers (Docker)
- We want to run different commands
- We want to run an executor
- We want to make the framework configurable

Calculate task offer

```
def maxTasksToRunWithOffer(self, offer):
        count = cpus = mem = 0
        for rsc in offer.resources:
            if rsc.name == "cpus":
                cpus += rsc.scalar.value
            if rsc.name == "mem":
                mem += rsc.scalar.value
        logging.info("CPUs: %s MEM: %s", cpus, mem)

        while cpus >= TASK_CPUS and mem >= TASK_MEM:
            count += 1
            cpus -= TASK_CPUS
            mem -= TASK_MEM
        return count
```

Calculate task offer (short)

```
def maxTasksToRunWithOffer(self, offer):
    logging.info("CPUs: %s MEM: %s",
                 offer.resources[0].scalar.value,
                 offer.resources[1].scalar.value)

    cpu_tasks = int(offer.resources[0].scalar.value/TASK_CPUS)
    mem_tasks = int(offer.resources[1].scalar.value/TASK_MEM)

    return cpu_tasks if cpu_tasks <= mem_tasks else mem_tasks
```

Framework - resourceOffers 2.0

```
def resourceOffers(self, driver, offers):
    for offer in offers:
        countTasks = self.maxTasksToRunWithOffer(offer)
        if countTasks == 0:
            driver.declineOffer(offer.id)
            return

        tasks = []
        for i in range(countTasks):
            task = new_task(offer)
            task.command.value = "while [ true ]; do echo hello worl
            tasks.append(task)
        driver.launchTasks(offer.id, tasks)
```

Let's run the framework

Log into the Master

```
python hello_mesos_offers.py
```

What's next?

- We want to exit the framework ✓
- We want to run tasks only if the offer fits ✓
- We want to run only a certain amount of tasks
- We want to run containers (Docker)
- We want to run different commands
- We want to run an executor
- We want to make the framework configurable

Your turn

- Only 5 tasks should be running
- Use statusUpdate(self, driver, update)
- Check resourceOffers(..) if you created enough tasks

Task states in Mesos

```
enum TaskState {
  TASK_STAGING = 6; // Initial state. Framework status updates shoul
  TASK_STARTING = 0;
  TASK_RUNNING = 1;
  TASK_FINISHED = 2; // TERMINAL. The task finished successfully.
  TASK_FAILED = 3; // TERMINAL. The task failed to finish successful
  TASK_KILLED = 4; // TERMINAL. The task was killed by the executor.
  TASK_LOST = 5; // TERMINAL. The task failed but can be rescheduled
  TASK_ERROR = 7; // TERMINAL. The task description contains an erro
}
// Usage: e.g. mesos_pb2.TASK_RUNNING
```

The solution - statusUpdate

```
def statusUpdate(self, driver, update):
        logging.info("Task %s is in state %s" %
                     (update.task_id.value,
                      mesos_pb2.TaskState.Name(update.state)))

        if update.state == mesos_pb2.TASK_RUNNING:
            self.runningTasks += 1
            logging.info("Running tasks: %s", self.runningTasks)
            return

        if update.state != mesos_pb2.TASK_RUNNING or\
           update.state != mesos_pb2.TASK_STARTING or\
           update.state != mesos_pb2.TASK_STAGING:
            self.runningTasks -= 1
            logging.info("Running tasks: %s", self.runningTasks)
```

The solution - resourceOffers

```
def resourceOffers(self, driver, offers):
    tasks_to_start = RUNNING_TASKS - self.runningTasks
    for offer in offers:
        if RUNNING_TASKS <= self.runningTasks:
            driver.declineOffer(offer.id)
            return
        count_tasks = max_tasks_to_run_with_offer(offer)
        start_tasks = count_tasks if count_tasks <= tasks_to_start e
        tasks_to_start -= start_tasks

        if start_tasks <= 0:
            driver.declineOffer(offer.id)
            return

        tasks = []
        for i in range(start_tasks): ...
```

What's next?

- We want to exit the framework ✓
- We want to run tasks only if the offer fits ✓
- We want to run only a certain amount of tasks ✓
- We want to run containers (Docker)
- We want to run different commands
- We want to run an executor
- We want to make the framework configurable

Reminder - new Task

```
def new_task(offer, name, cmd):
    task = mesos_pb2.TaskInfo()
    task.task_id.value = str(uuid.uuid4())
    task.slave_id.value = offer.slave_id.value
    task.name = name
    task.command.value = cmd

    cpus = task.resources.add()
    cpus.name = "cpus"
    cpus.type = mesos_pb2.Value.SCALAR
    cpus.scalar.value = TASK_CPUS

    mem = task.resources.add()
    mem.name = "mem"
    mem.type = mesos_pb2.Value.SCALAR
    mem.scalar.value = TASK_MEM
```

Docker Task

```
def new_docker_task(offer, prefix, cmd):
    task = new_task(offer, prefix, cmd)

    container = mesos_pb2.ContainerInfo()
    container.type = 1

    docker = mesos_pb2.ContainerInfo.DockerInfo()
    docker.image = "busybox"
    docker.network = 3
    docker.force_pull_image = True

    container.docker.MergeFrom(docker)
    task.container.MergeFrom(container)

    return task
```

Docker Task - alternative

```
def new_docker_task(offer, prefix, cmd):
    task = new_task(offer, prefix, cmd)
    task.container.type = 1
    task.container.docker.image = "busybox"
    task.container.docker.network = 3
    task.container.docker.force_pull_image = True
    return task
```

RessourceOffer

```
def resourceOffers(self, driver, offers):
    logging.info("Recieved resource offers: %s",
                 [o.id.value for o in offers])
    ...
    for offer in offers:
        ...
        for i in range(countTasks):
            task = new_docker_task(offer,
                                   "Hello Docker",
                                   "while [ true ]; do echo 'Hello D
            tasks.append(task)
    ...
```

Mount a volume

```
container = mesos_pb2.ContainerInfo()
container.type = 1

volume = container.volumes.add()
volume.container_path = "/etc/tmp"
volume.host_path = "/tmp"
volume.mode = 2
```

Equal to

```
docker run -v /tmp:/etc/tmp:ro
```

Your turn

- Create a file on the slaves
- Put "Hello Volumes" inside
- Read the file inside a Docker container
- Read only mode is enough

Solution

At first for each 3 slaves

```
ssh root@kit-mesos-slave1 'echo "Hello Volumes" > /tmp/readme.txt'
```

Inside the new_docker_task

```
volume = container.volumes.add()
volume.container_path = "/tmp"
volume.host_path = "/tmp"
volume.mode = 2
```

Inside the resourceOffers

```
task = new_docker_task(offer,
     "Hello Docker",
     "while [ true ]; do cat /tmp/readme.txt; sleep 5; done")
```

Port assignment

```
# Add a resource
mesos_ports = task.resources.add()
mesos_ports.name = "ports"
mesos_ports.type = mesos_pb2.Value.RANGES
# Fetch the available ports from the offer
available_port = get_available_port(offer)
# Create the port range
port_range = mesos_ports.ranges.range.add()
port_range.begin = available_port
port_range.end = available_port
# Define the port mapping for Docker
docker_port = docker.port_mappings.add()
docker_port.host_port = available_port
docker_port.container_port = Container-Port
```

Get available port

```
def get_available_port(offer):
    begin = offer.resources[3].ranges.range[0].begin
    end = offer.resources[3].ranges.range[0].end
    logging.info("Port range [%s:%s]", begin, end)
    # We could also simply increment but that would be to easy :)
    return random.randint(begin, end)
```

Your turn

- Run a Docker container (python:3)
- Run a web server (python3 -m http.server 8080)
- Assign free ports to the containers
- docker.network = 2 (Bridge mode)

What's next?

- We want to exit the framework ✓
- We want to run tasks only if the offer fits ✓
- We want to run only a certain amount of tasks ✓
- We want to run containers (Docker) ✓
- We want to run different commands
- We want to run an executor
- We want to make the framework configurable

Python commandline parser

```
import argparse
...
...
parser = argparse.ArgumentParser(description='A simple Mesos Framewo
parser.add_argument('--cmd', type=str, default='echo Hello World',
                    help='Command to run with the framework')

args = parser.parse_args()
print args.cmd
```

Usage

```
python mesos_args.py --cmd "echo Hello World && sleep 500"
```

Your turn

- We want to specify when we run Docker
- We want to specify the Docker image
- We want to specify the command
- We want to specify the task cpu/mem
- We want to make this static (only at start up)

Run the example

```
python mesos_args.py \
--cmd "echo Hello World && sleep 500" \
--docker True \
--mem 1024.0 \
--cpu 1.0 \
--image ubuntu
```

What's next?

- We want to exit the framework ✓
- We want to run tasks only if the offer fits ✓
- We want to run only a certain amount of tasks ✓
- We want to run containers (Docker) ✓
- We want to run different commands ✓
- We want to run an executor
- We want to make the framework configurable

Executor


Mesos slave
Hello Executor
Hello task 1
Hello task 2
Hello task 3
GoodBye Executor
GoodBye task 1
GoodBye task 2
GoodBye task 3

A simple executor

In main method

```
helloWorldExecutor = mesos_pb2.ExecutorInfo()
helloWorldExecutor.executor_id.value = "HelloWorld-executor"
helloWorldExecutor.command.value = "while [ true ]; do echo hello wo
helloWorldExecutor.name = "HelloWorld"
helloWorldScheduler = HelloWorldScheduler(helloWorldExecutor)
```

In resourceOffer()

```
task = new_task(offer)
task.executor.MergeFrom(self.hello_executor)
```

Run the example

```
python hello_mesos_executor.py
```

Executor - implementation

- We create a python script for the Scheduler
- And one for the Executor
- The Slave fetches the executor script
- And then executes it

Executor imports

```
import threading
import sys
import time
from mesos.interface import mesos_pb2, Executor
from mesos.natie import MesosExecutorDriver
```

HelloWorldExecutor - 1

```
class HelloWorldExecutor(Executor):
    def registered(self, driver, executorInfo, frameworkInfo, slaveI
        print "HelloWorldExecutor registered"

    def reregistered(self, driver, slaveInfo):
        print "HelloWorldExecutor reregistered"

    def disconnected(self, driver):
        print "HelloWorldExecutor disconnected"
```

HelloWorldExecutor - 2

```
def launchTask(self, driver, task):
    def run_task():
        def task_update(state):
            update = mesos_pb2.TaskStatus()
            update.task_id.value = task.task_id.value
            update.state = state
            driver.sendStatusUpdate(update)

        task_update(mesos_pb2.TASK_RUNNING)

        for i in range(0, 100):
            print "%s says Hello World!" % task.task_id.value
            time.sleep(2)

        task_update(mesos_pb2.TASK_FINISHED)
        return
```

HelloWorldScheduler 2.0

```
if __name__ == '__main__':
  hello_executor = mesos_pb2.ExecutorInfo()
  hello_executor.executor_id.value = "hello-executor"
  hello_executor.name = "Hello"
  hello_executor.command.value = "python hello_executor.py"

  uri_proto = hello_executor.command.uris.add()
  uri_proto.value = "http://kit-mesos-master:9000/hello_executor.py"
  uri_proto.extract = False

  framework = mesos_pb2.FrameworkInfo()
  ...
  HelloWorldScheduler(hello_executor)
```

ressourceOffer() 2.0

```
.....
task = new_task(offer, "Hello ")
task.executor.MergeFrom(self.hello_executor)
....
```

Run the example

```
python mesos-executor.py
```

Your turn

- Create a second Executor
- Says goodbye
- One half should say "hello" the other half "goodbye"

Solution

- Like the HelloWorld Executor
- Created a GoodBye Executor inside the Scheduler
- I used odd or even to decide which executor to run

Your turn

Scheduler 2.0

- We can now add a graceful shutdown
- We want to give the tasks a timeout to stop
- The number of running task can be used
- Set timeout to 15s

graceful_shutdown()

```
def graceful_shutdown(signal, frame):
    print "HelloWorldScheduler is shutting down"
    helloWorldScheduler.shuttingDown = True

    wait_started = datetime.datetime.now()
    while (helloWorldScheduler.runningTasks > 0) and \
            (SHUTDOWN_TIMEOUT > (datetime.datetime.now() - wait_star
        time.sleep(1)

    if helloWorldScheduler.runningTasks > 0:
        print "Shutdown by timeout, %d task(s) have not completed" %

    hard_shutdown()
```

What's next?

- We want to exit the framework ✓
- We want to run tasks only if the offer fits ✓
- We want to run only a certain amount of tasks ✓
- We want to run containers (Docker) ✓
- We want to run different commands ✓
- We want to run an executor ✓
- We want to make the framework configurable

What do we need to make a framework configurable?

What's next?

- We want to exit the framework ✓
- We want to run tasks only if the offer fits ✓
- We want to run only a certain amount of tasks ✓
- We want to run containers (Docker) ✓
- We want to run different commands ✓
- We want to run an executor ✓
- We want to make the framework configurable ✓

Thank you!

Johannes M. Scheuermann
IT Engineering & Operations

inovex GmbH
Office Karlsruhe
Ludwig-Erhard-Allee 6
76131 Karlsruhe

Mail:
johannes.scheuermann@inovex.de

Images

- Page 10 - https://flic.kr/p/dZ2xUn
- Page 32 - https://flic.kr/p/pbJ4VJ
- Page 44 - https://flic.kr/p/9PmF6j
- Page 55 - https://flic.kr/p/fVfDks
- Page 58 - https://flic.kr/p/4q7V3x
- Page 61 - https://flic.kr/p/ovRNdh
- Page 95 - https://flic.kr/p/oyD4EY
- Page 102 - https://flic.kr/p/crTsSh

Images

- Page 112 - https://flic.kr/p/9xNg55
- Page 120 - https://flic.kr/p/w33AFL
- Page 133 - https://flic.kr/p/5MQQVH
- Page 146 - https://flic.kr/p/d8y4tS
- Page 153 - https://flic.kr/p/7Li88d
- Page 156 - https://flic.kr/p/8xB9sr
- Page 174- https://flic.kr/p/7FHvkj