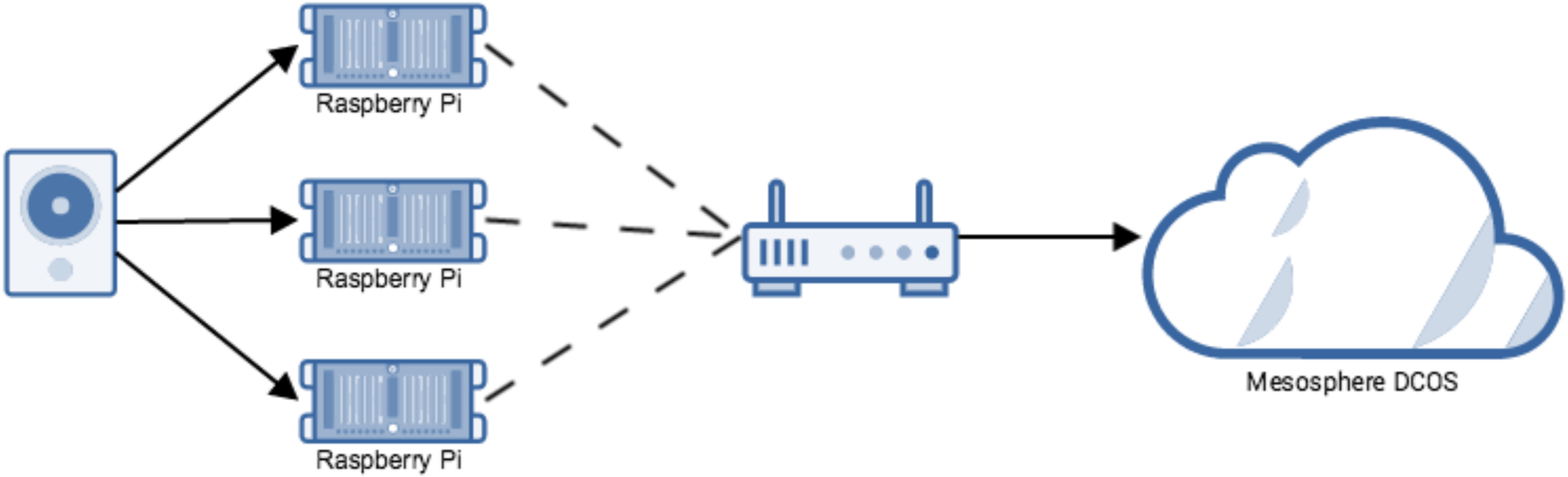
Using Distributed Systems to Ingest Data from the

Internet of Pis

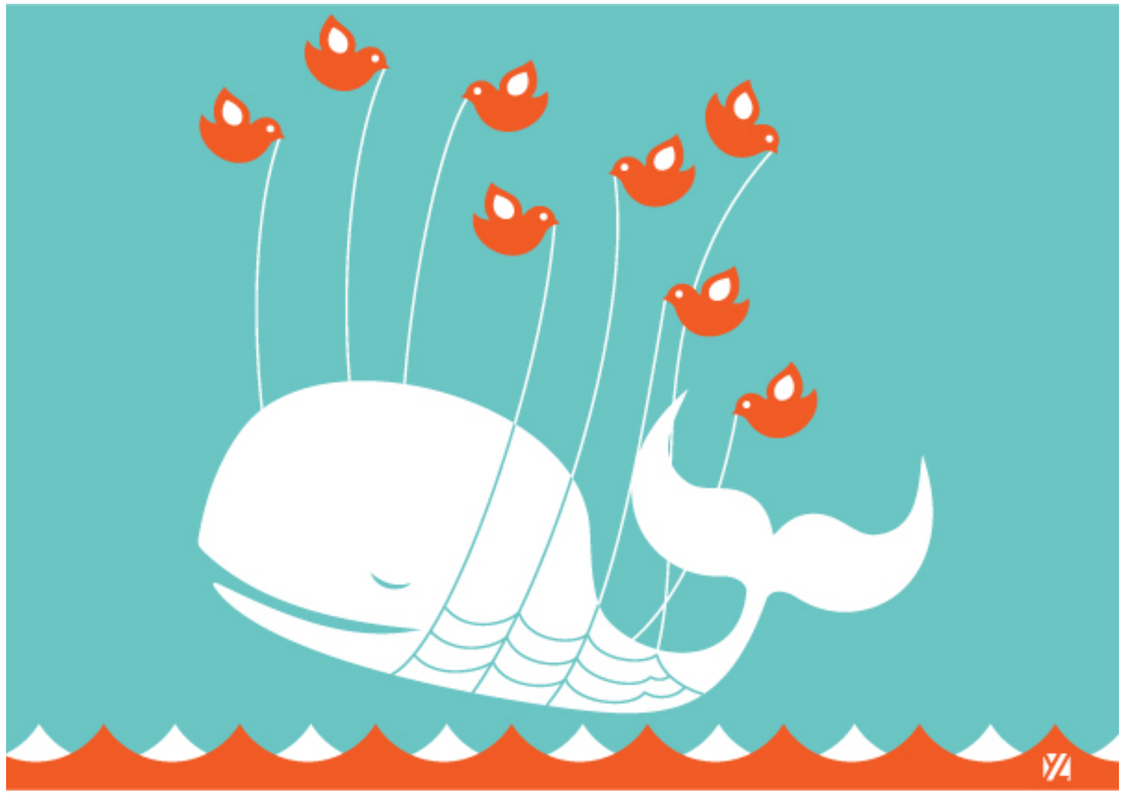
MESOSPHERE

What’s the point?

Internet of Pis

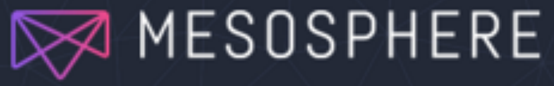


Scaling manually is hard!



- Mesos is credited with “slaying the fail whale”.
- Twitter’s user growth outstripped their infrastructure growth.
- Mesos, a cluster resource manager, let them scale.


Siri: Today
Third Generation
powered by



PUT YOUR DATACENTER ON AUTOPILOT.

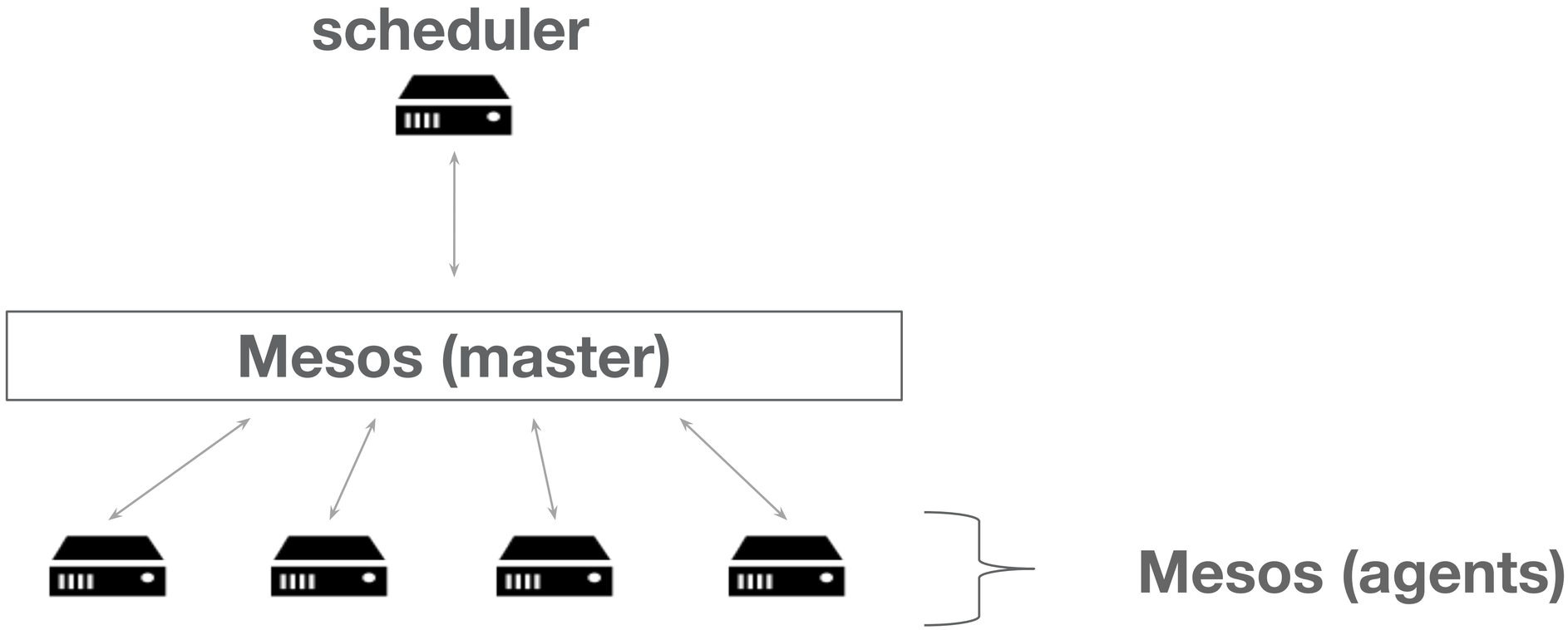
Deploy apps faster. Scale massively. Automate effortlessly.

Get Started Today

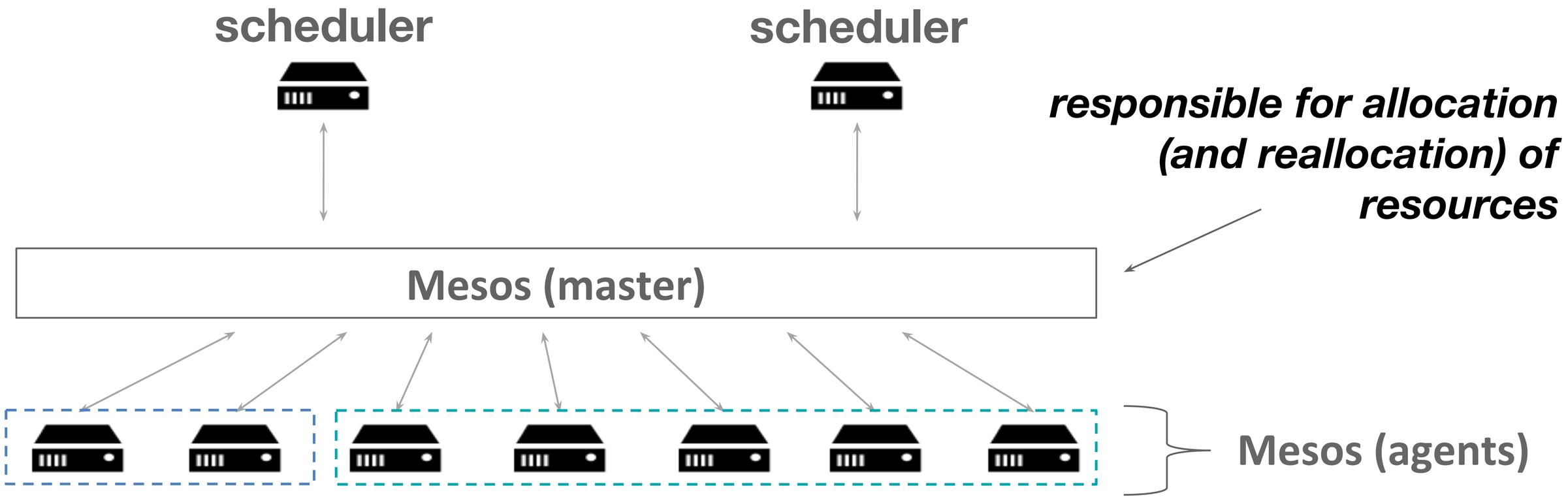
Learn More

Mesoswho

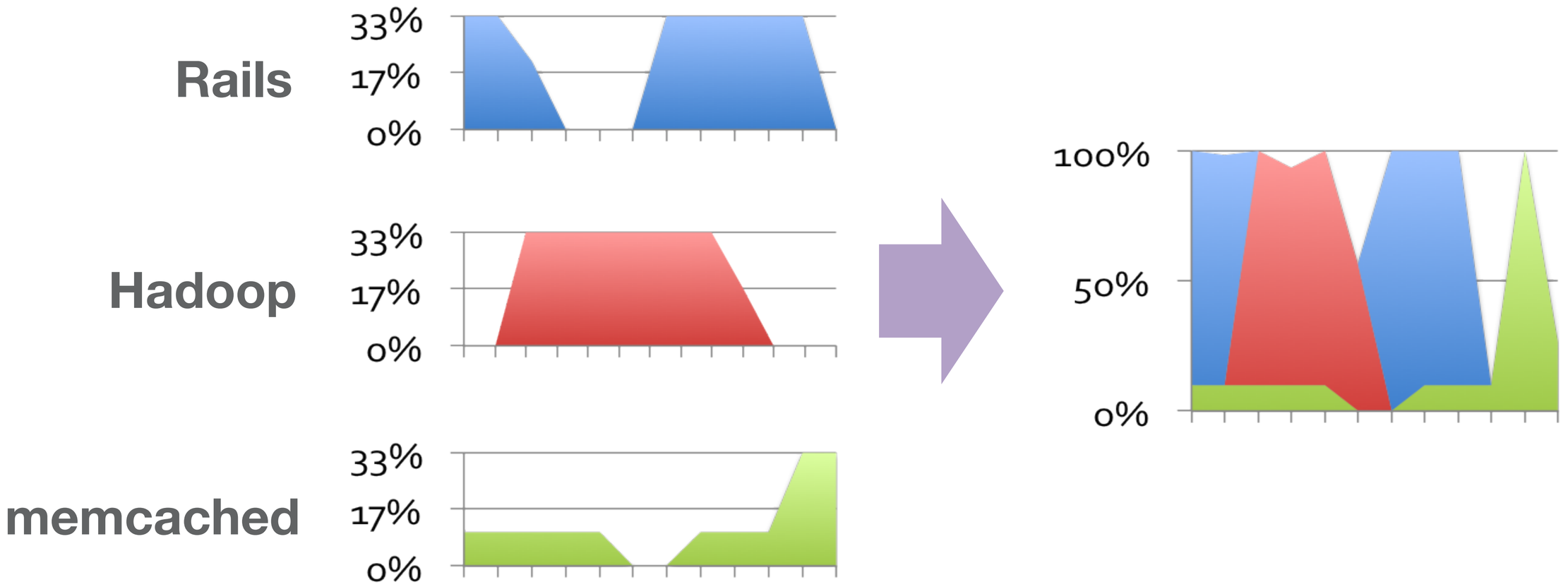
Mesos: Level of Indirection



Mesos: Level of Indirection



Mesos: Improves Utilization



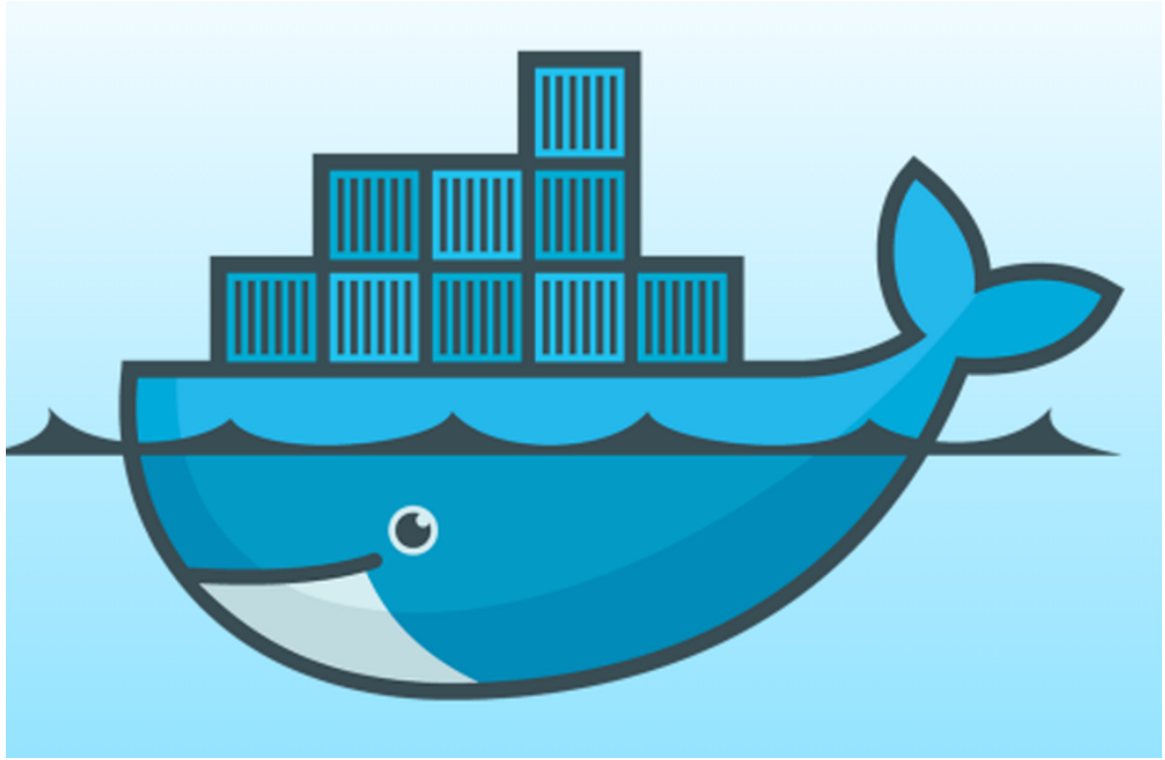
Mesos: Battle Tested

- A top-level Apache project
- A cluster resource negotiator
- Scalable to 10,000s of nodes
- Fault-tolerant, battle-tested
- An SDK for distributed apps

Containerwhat

What are containers?

- Shippable unit of execution.
- Take an operating system, bundle together libraries and application files and ship it around.
- Allows deployment environments (i.e. servers) to be homogenous, “Cattle, not pets”



Workshop!

http://hackers-at-berkeley.mesosphere.io

1. Contain
2. Deploy
3. Discover

Philosophy of the
DISTRIBUTED CHAMPION

With me, Tyler!

First Steps

Set up Docker

Docker is a format and tool that allows you to build and run containers.

We'll use it to package our applications for deployment on the cluster.

1. Install Docker on your local machine: https://docs.docker.com/installation/
2. Sign up for a Docker Hub account: https://hub.docker.com

Set up the DCOS CLI

DCOS is Mesosphere's Mesos distribution.

We will use the DCOS CLI, a command line client, to easily manage and deploy applications on a DCOS cluster.

1. Install the DCOS CLI: https://docs.mesosphere.com/install/cli,

   using the hostname `http://dcos.hackers-at-berkeley.mesosphere.io`

   If you've already installed the CLI, you can update it by running:

   ```
   dcos config set core.dcos_url http://dcos.hackers-at-berkeley.mesosphere.io
   ```

Get the example files

1. Clone the workshop Github repository to get our example files: https://github.com/mesosphere/hackers-at-berkeley

(for git help, try https://help.github.com/articles/cloning-a-repository/)

Exercise 1

Build and run a Python webservice

Exercise 1

Building a Webservice

We're going to run a simple web service and access it locally. Using a Python module called Flask, this is pretty easy!

1. In your terminal, navigate to the `/workshop/exercise1` subdirectory of the `hackers-at-berkeley` GitHub repository.
2. `simple_webservice.py` is the code for our web service. Check it out!
3. Run the web service:
   `./simple_webservice.py`
4. In a web browser, navigate to `http://localhost:8080/hey-you`. You should see that your message is passed into the service. Nice!

Exercise 1

Containerizing the Webservice

Now that we have our Python webservice written, we're going to create a Docker image containing it, along with any dependencies it needs.

1. We've provided you with a minimal Dockerfile, a bit like a Makefile but for Docker images. Check it out in `workshop/exercise1/Dockerfile`. This file specifies a base image (FROM), some commands to install dependencies (RUN) and the command to execute (CMD).
2. To build this container image, specify your Docker Hub username and a tag name:
   `docker build -t <your_dockerhub_name>/simple-webservice .`

Exercise 1

Running the Container

Now that we've built the container, we can run it locally trivially.

1. To run the built container image:
   `docker run -it --rm -p 8080:8080 <your_dockerhub_name>/simple-webservice`
2. You can then view it in your browser at http://localhost:8080
   (If you're using a Mac, you'll need to use `docker-machine ip default` to discover the IP that the Docker virtual machine binds to.)

Exercise 2

Deploy a Python webservice

Exercise 2

Marathon Basics

We’re going to deploy it to a DCOS cluster using a Mesos framework called Marathon, which is a bit like Heroku, or Google App Engine but for containers.

Marathon takes an application definition (normally written in JSON) and attempts to achieve the defined state.

For example:

- an application is called X
- it uses container image mesosphere/x
- it needs 10 instances
- each instance should have 0.5 CPUs and 256MB of memory.

Exercise 2

Sharing Container Images

Assuming you've build a Docker image locally with a name like `<your_dockerhub_name>/simple-webservice`, we now need to put it somewhere that our cluster can fetch it. Docker Hub is a public registry of Docker images which we can use for this purpose.

1. First, login to Docker Hub:
   ```
   docker login
   ```
2. Then, push your image:
   ```
   docker push <your_dockerhub_name>/simple-webservice
   ```

Exercise 2

Deploying Using Marathon

Now we just need to construct an application definition for Marathon to use.

We've provided an example `marathon.json` in `workshop/example2` for you to customise.

1. Make sure to change the application name (this must be unique):
   `"id": "/simple-webservice-yourname"`
2. Update the container image name to the one you pushed earlier:
   `"image": "<your_dockerhub_name>/simple-webservice"`
3. Deploy the application to the cluster:
   `dcos marathon app add marathon.json`

Exercise 2

Getting at your Webservice

You can check to see that your application is running successfully by visiting the Marathon UI: http://dcos.hackers-at-berkeley.mesosphere.io/service/marathon/

Our application specifies a hostPort of 0, which is a magic number for Marathon - it will deploy it to any port that is available. In order to find it, we normally use the Marathon UI.

To make it easier, we've created a small webservice to get the public IPs for your application:

1. Visit http://public.hackers-at-berkeley.mesosphere.io:8080/simple-webservice-yourname, replacing the part after the / with your application's name.
2. This will return a JSON array containing just one IP address and port (assuming you only deployed 1 instance).
3. Paste the IP address and port into your browser to reach your webservice!

Exercise 2

Done!

If all went correctly, you should now see your webservice running on the cluster!

So that we have enough capacity on the cluster, please “Destroy” your application from the UI.

Exercise 3

Create and deploy a simple frontend/backend Python application

Exercise 3

Front & Back-end Services

Now that you've successfully built and deployed some Dockerized applications, let's try deploying two applications that talk to one another. We'll create a front-end, which is where users will first reach, and a back-end, where some piece of logic is being performed in a separate process. You'll find these apps in `/workshop/exercise3/frontend_service` and `/workshop/exercise3/backend_service`.

Exercise 3

Front & Back-end Webservices

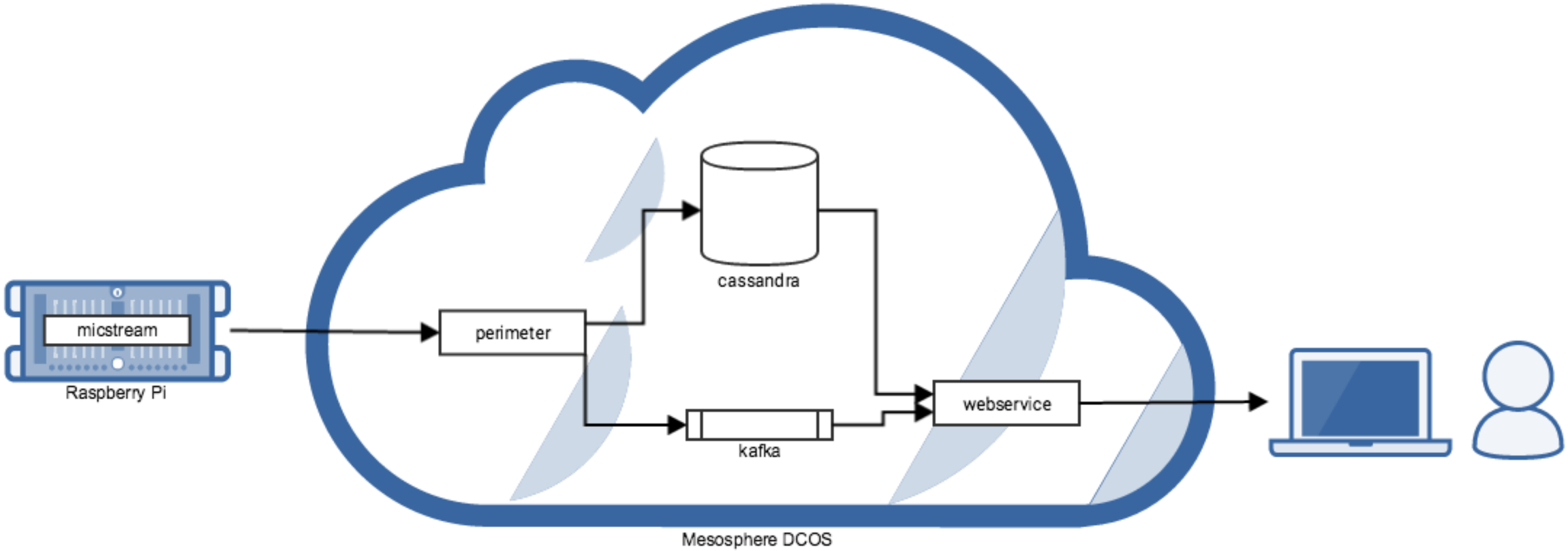
- These examples rely on SRV records, which are a DNS extension that allows one to retrieve both a host and an IP address based on a service name. This is how the find-hack app worked, for example.
-

Exercise 4

Create and deploy a Python application that talks to Cassandra

An “Internet of Things” Service

In the subdirectory for Exercise 4, you’ll find a web app that reads data being posted by the microphones throughout the room. Using your experience from the first 3 exercises, get this app running on Marathon!



Thanks!

(come and work for us)

ACT 3 - Service Discovery

- Our apps will be scheduled to ports that are available on the machine, which means they cannot be predicted ahead of time.
- We can use SRV DNS records to find both an (internal) IP and a port for a name
  - examples/snippets/marathon_SRV_discovery.py
  - `srvlookup.lookup('myapp', 'tcp', 'marathon.mesos')`