
The Docker Story in Apache Mesos

Timothy Chen

About me:

- Previously Mesosphere Lead Engineer, (ex-Microsoft/VMWare)
- Apache Mesos, Drill PMC
- Help maintain Apache Spark on Mesos

What I hope you learn from this talk....

- Learn more about Mesos
- Learn more about Docker
- Make you think twice when you like to build something that integrates with Docker.....

APACHE MESOS



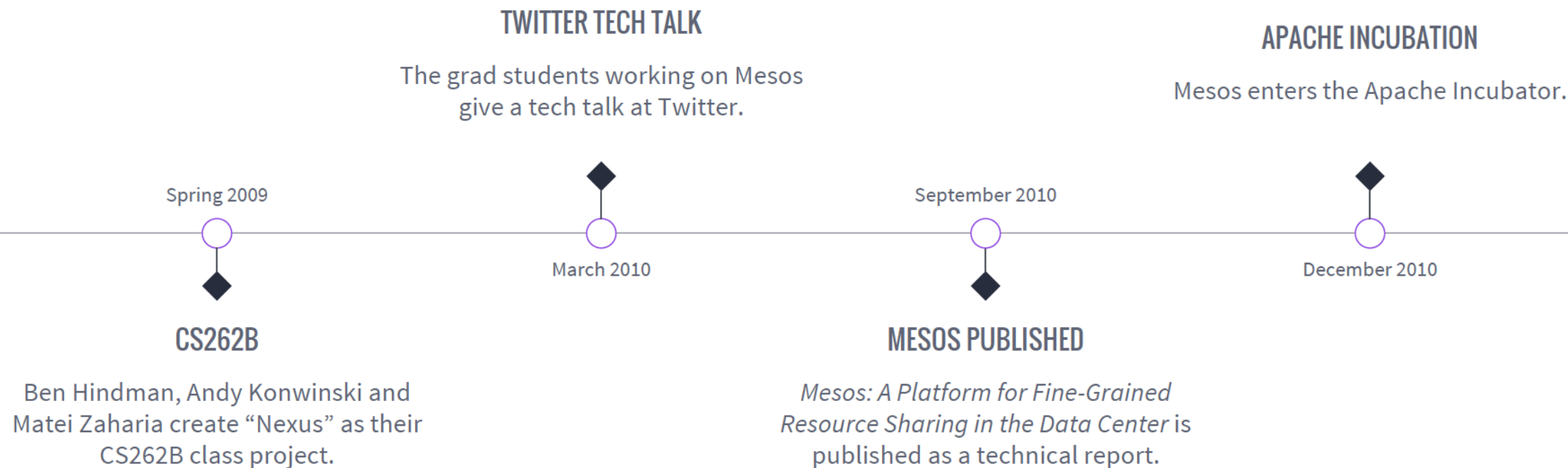
- A top-level ASF project
- A cluster resource negotiator
- Scalable to 10,000s of nodes but also useful for a handful of nodes
- Fault-tolerant, battle-tested
- An SDK for distributed apps
- Native Docker support

mesos.apache.org



Apache Mesos: The Story Of

THE BIRTH OF MESOS





Back in 2014.....





MESOS



docker



docker



docker



docker



docker



docker



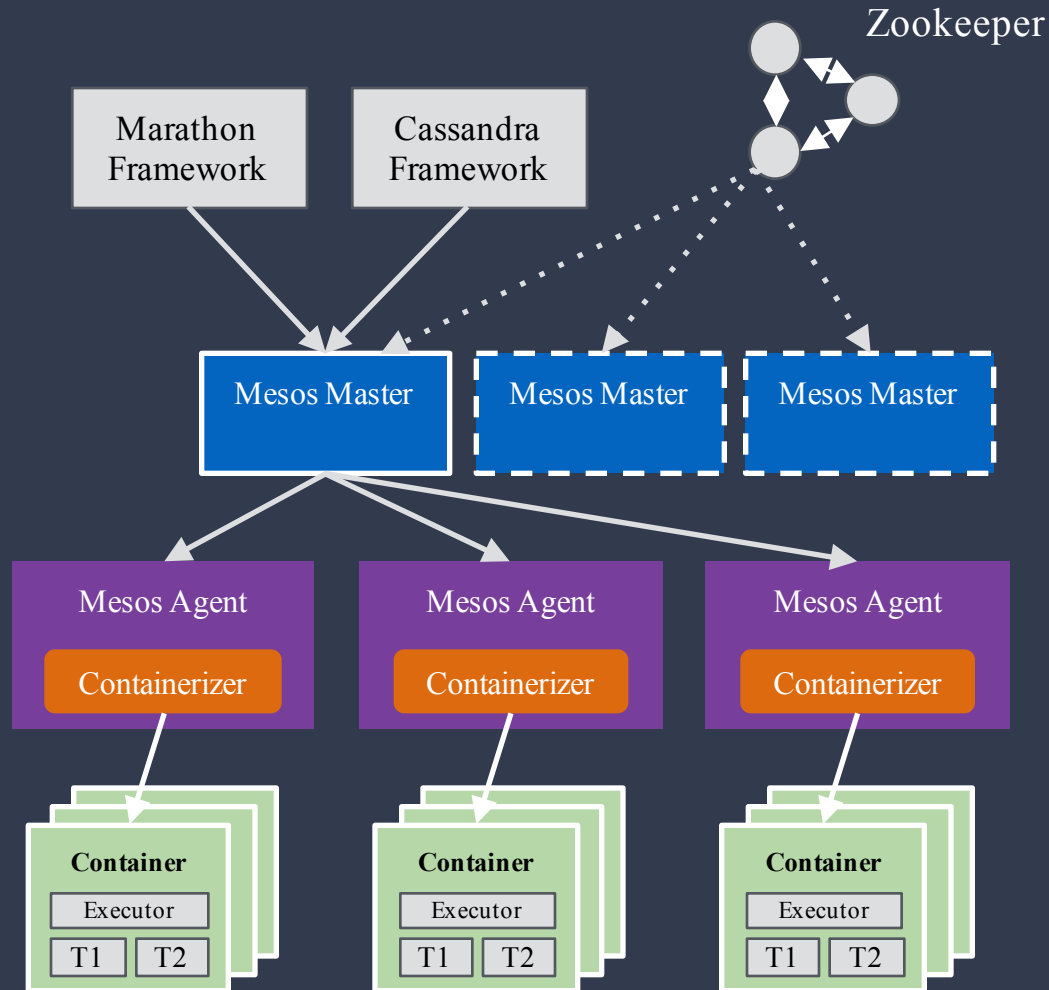
MESOS

+



docker

What is containerizer?



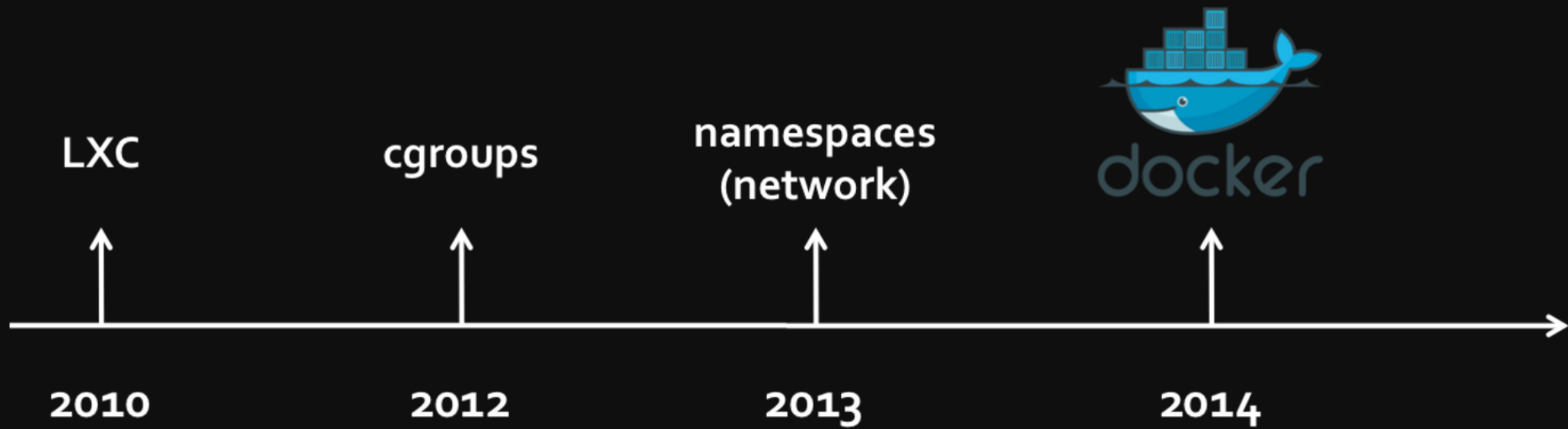
Containerizer

Between agents and containers

Launch/update/destroy containers

Provide isolations between containers

Report container stats and status

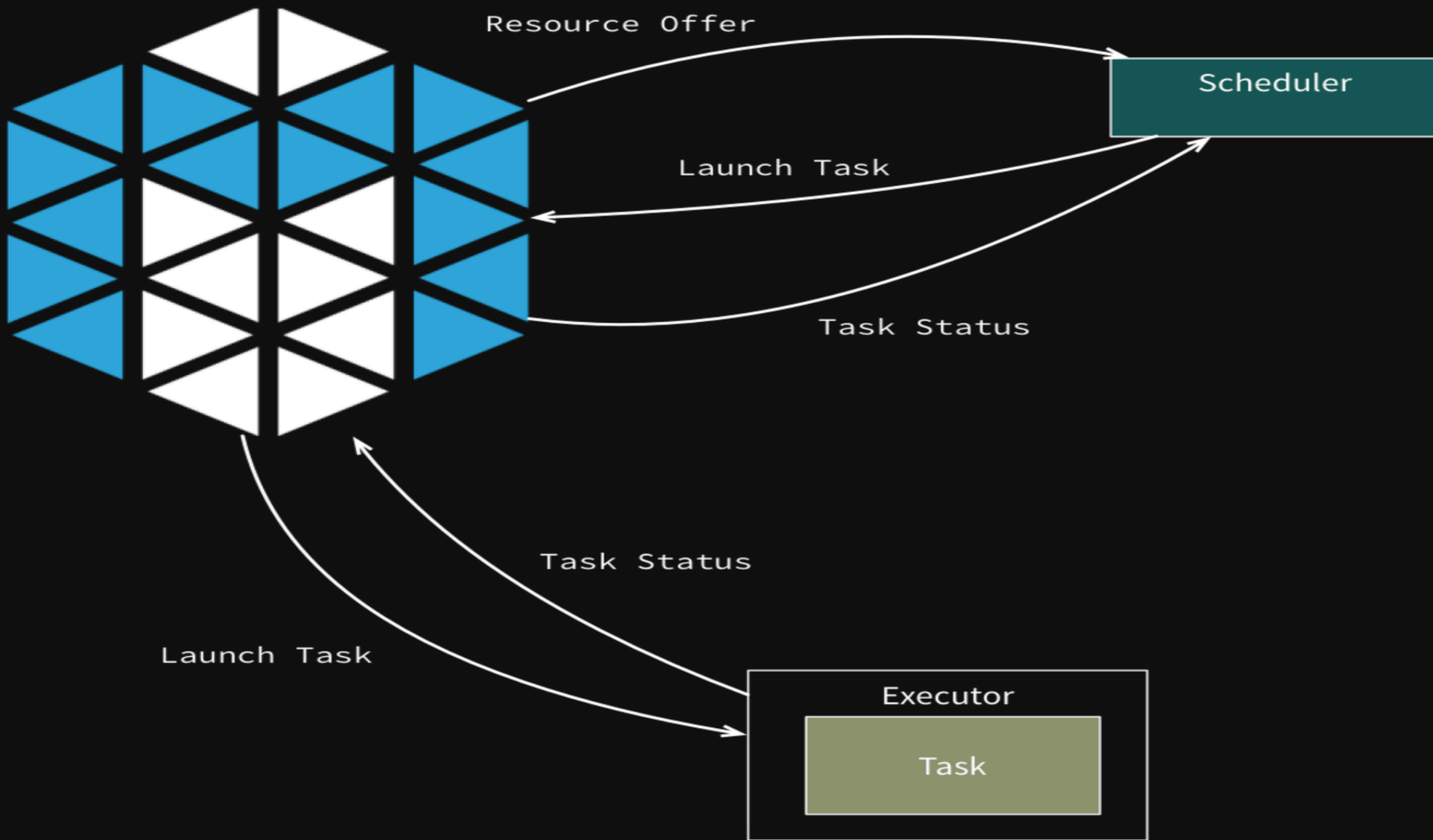


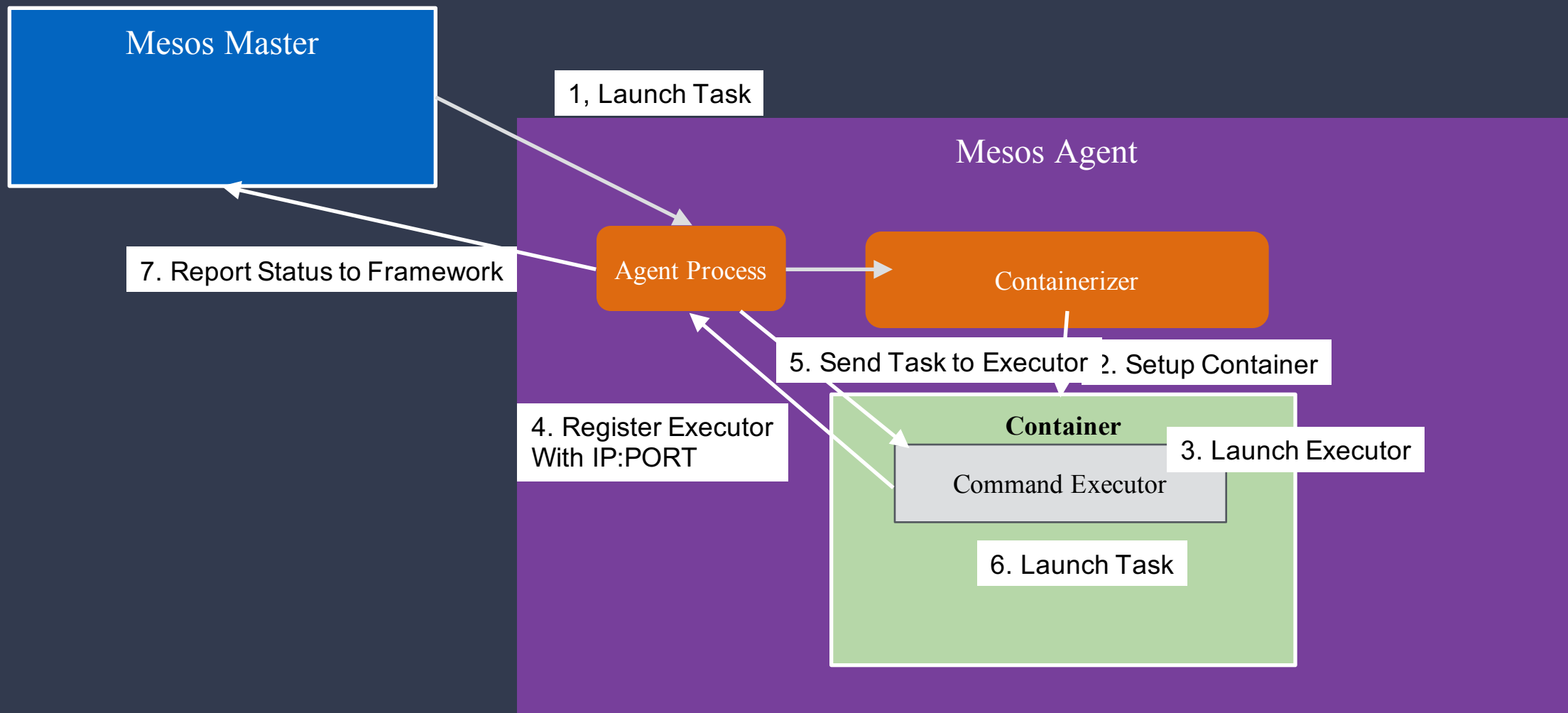
Containerization in Mesos, a brief history

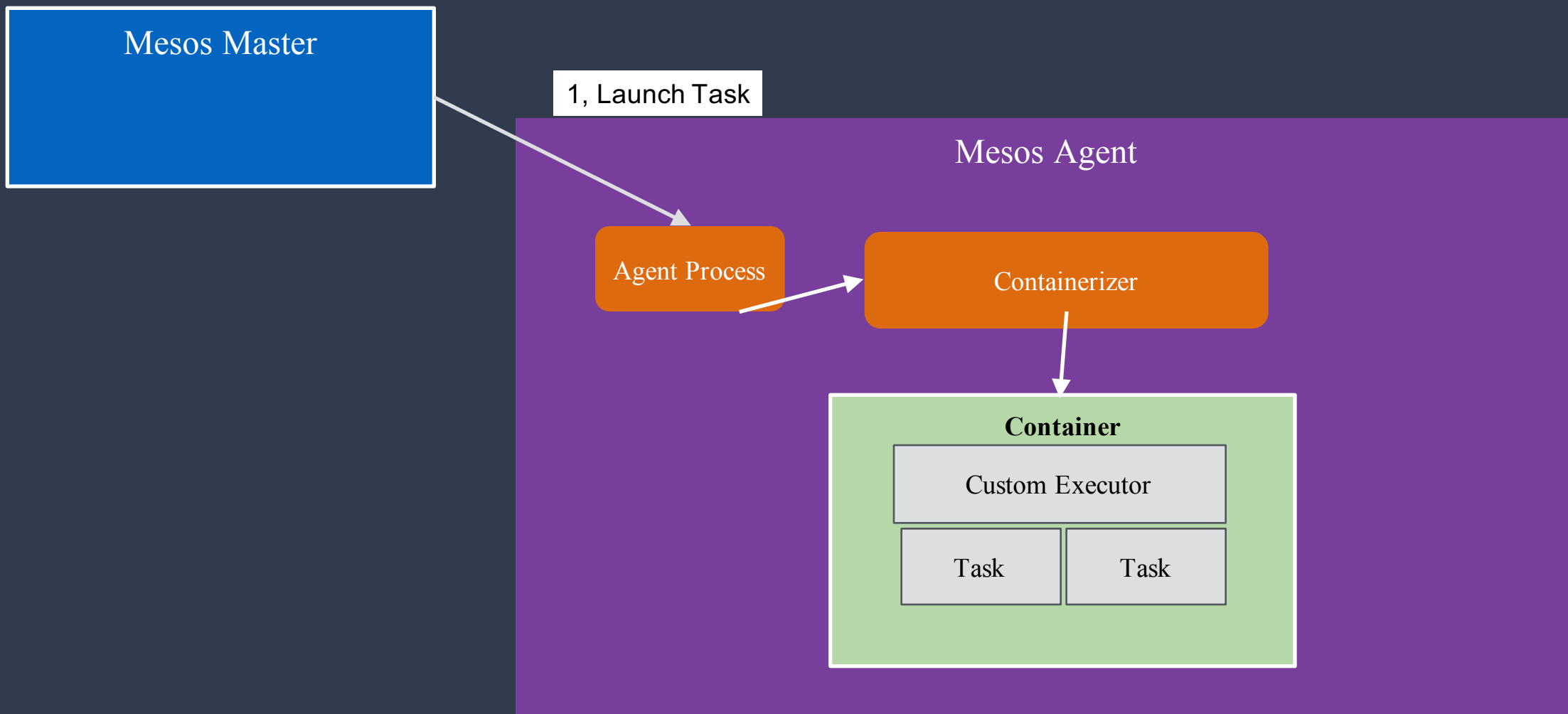
Mesos Containerizer

- Isolation using standard OS features (e.g., cgroups, namespaces)
- Pluggable architecture allowing customization and extension with isolators

How Mesos Task Launch Work?







Docker integration early 2014....

Some considerations

- Is Docker going to gain a lot of adoption (obvious now)?
- Are there more Container integrations we need to do?
- Docker is moving fast, how do we keep up?
- The production users of Mesos isn't going to be using Docker, how do we change as little as possible?

Here comes the External Containerizer + Deimos...

External Containerizer

- All calls to the containerizer invokes defined script names, i.e:
Launch -> /bin/sh -c launch...

(<http://mesos.apache.org/documentation/latest/external-containerizer>)

We built a Docker plugin for the External Containerizer in Python:

<https://github.com/mesosphere/deimos>

And?

- Docker became very popular
- There wasn't any more container integration we can see
- External Containerizer + Deimos was buggy and hard to use:
- All Docker/Deimos activity is external and we have no visibility in Mesos
- Too hard to setup, as it needs to also install Deimos on every box with python

External Containerizer is eventually deprecated..

<https://issues.apache.org/jira/browse/MESOS-3370>

▼  Jie Yu added a comment - 09/Aug/16 20:14

commit f6a0e72c5d142d247dfe5d5c5b6cfb1c3aa1b8cd

Author: Gilbert Song <songzihao1990@gmail.com>

Date: Tue Aug 9 13:13:32 2016 -0700

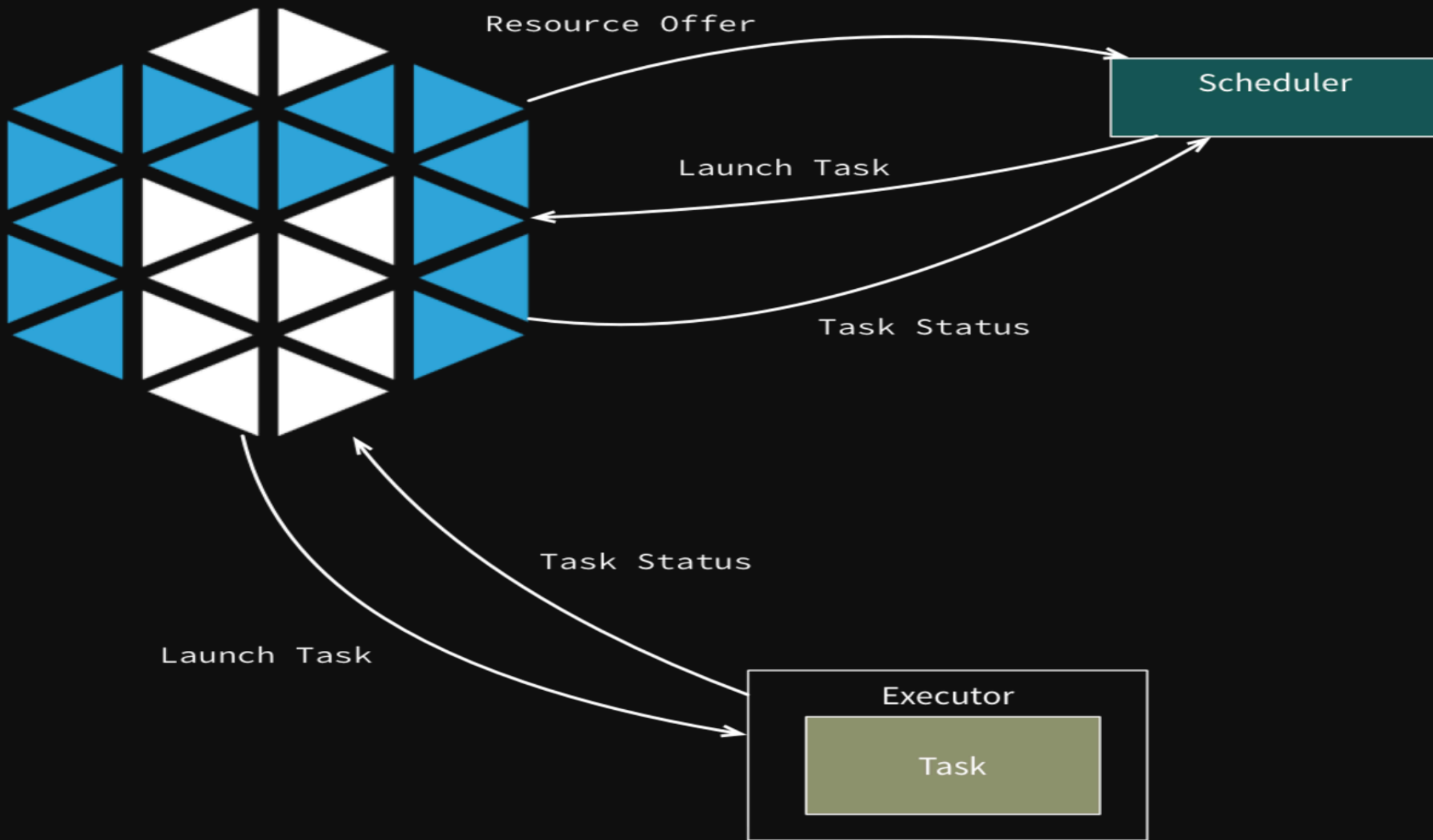
Removed the external containerizer.

Review: <https://reviews.apache.org/r/50914/>



What we want is...

- Native integration with Docker
- No extra setup required, just the Docker daemon and client
- Clean and simple way to use from the Framework
- Able to recover from slave crash and containers still run (we do better than the Docker daemon 😊)



Docker Containerizer

- Communicate with Docker daemon with local Docker client (docker run, pull, inspect, etc)
- Add specific protobuf Docker options to Schedulers
- Stream all Docker logs to stdout/stderr files

Docker Containerizer

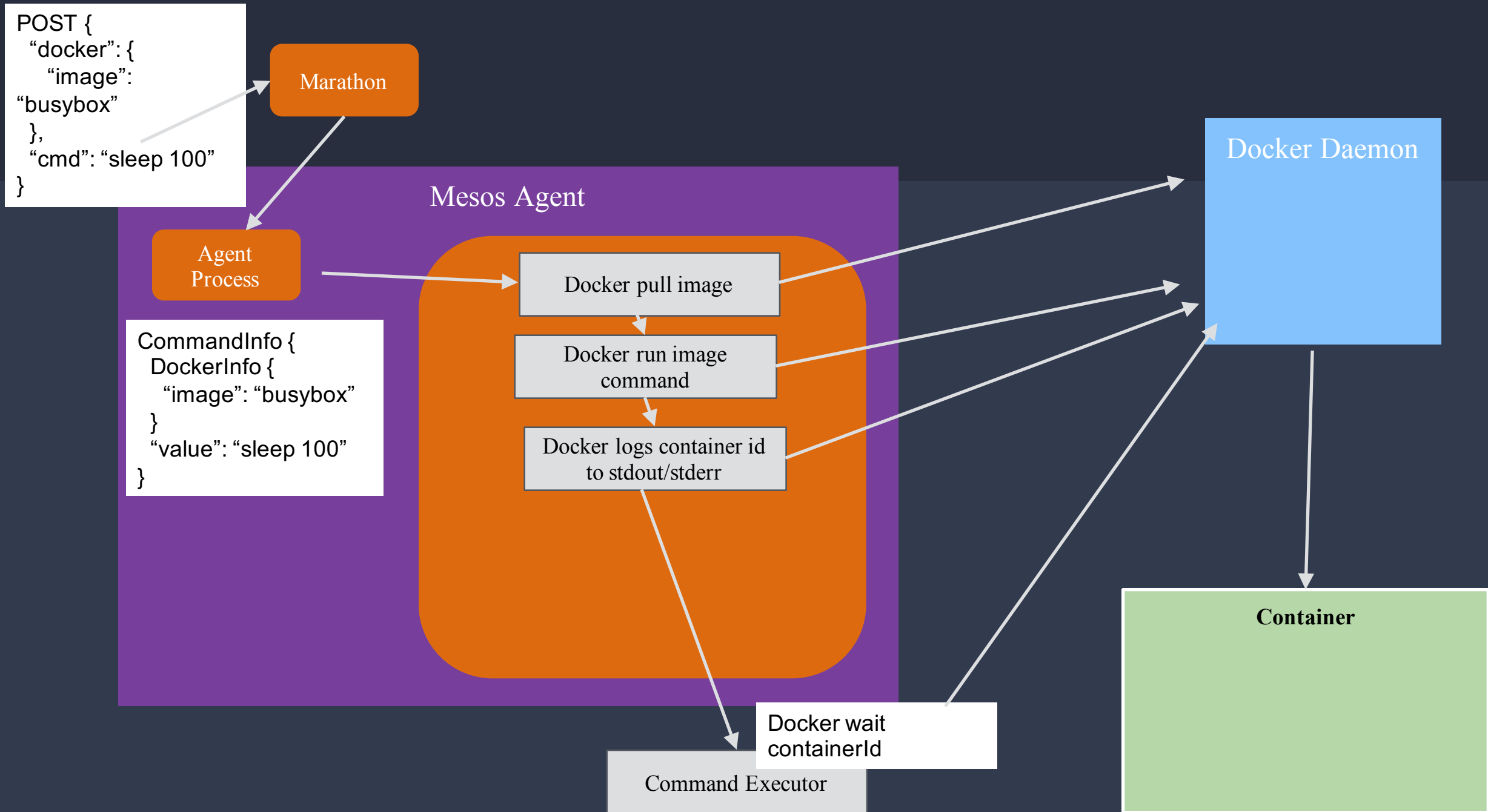
- Communicate with Docker daemon with Docker client (docker run, pull, inspect, etc)
- (HTTP API is not stable!)
- Add specific protobuf Docker options to Schedulers
- Stream all Docker logs to stdout/stderr files

Docker Containerizer



Docker Containerizer





```
{
// Redirect the logs into stdout/stderr.
//
// TODO(benh): This is an intermediate solution for now until we can
// reliably stream the logs either from the CLI or from the REST
// interface directly. The problem is that it's possible that the
// 'docker logs --follow' command will be started AFTER the
// container has already terminated, and thus it will continue
// running forever because the container has stopped. Unfortunately,
// when we later remove the container that still doesn't cause the
// 'logs' process to exit. Thus, we wait some period of time until
// after the container has terminated in order to let any log data
// get flushed, then we kill the 'logs' process ourselves. A better
// solution would be to first "create" the container, then start
// following the logs, then finally "start" the container so that
// when the container terminates Docker will properly close the log
// stream and 'docker logs' will exit. For more information, please
// see: https://github.com/docker/docker/issues/7020
```

```
string logs =
    "logs() {\n"
    "  " + path + " logs --follow $1 &\n"
    "  pid=$!\n"
    "  " + path + " wait $1 >/dev/null 2>&1\n"
    "  sleep 10\n" // Sleep 10 seconds to make sure the logs are flushed.
    "  kill -TERM $pid >/dev/null 2>&1 &\n"
    "}\n"
    "logs " + container;
```

**And of course,
we start hitting
Bugs...**

How about recovery?

Docker itself doesn't support daemon crashing and recovering containers (finally until few months ago?)

Assume just the Mesos agent restarted, how do we recover all docker containers launched by Mesos?

Mesos agent supports checkpointing, which writes the executor pid into filesystem.

Tag Containers

Docker labels isn't available yet, so let's name them!

- Docker run --name mesos-<mesos_container_id>

On Docker Containerizer restart:

- docker ps -a, read output and find all containers starting with mesos-
- Wait again on the checkpointed command executor pids with a matching container

How about custom executors?

- We can launch the custom executor in a Docker container by allowing the user to specify a Docker image in the ExecutorInfo, it will connect back to the Agent and everything should work?

How about custom executors?

- But how do we update the size of the executor container while it's running? (Docker until today doesn't support update in their API)
- We have no other choice but to go behind Docker to update it....
 - Docker inspect container to get the pid
 - Modify `/sys/fs/cgroup/cpu/memory/.....`

Done!



Not so fast...

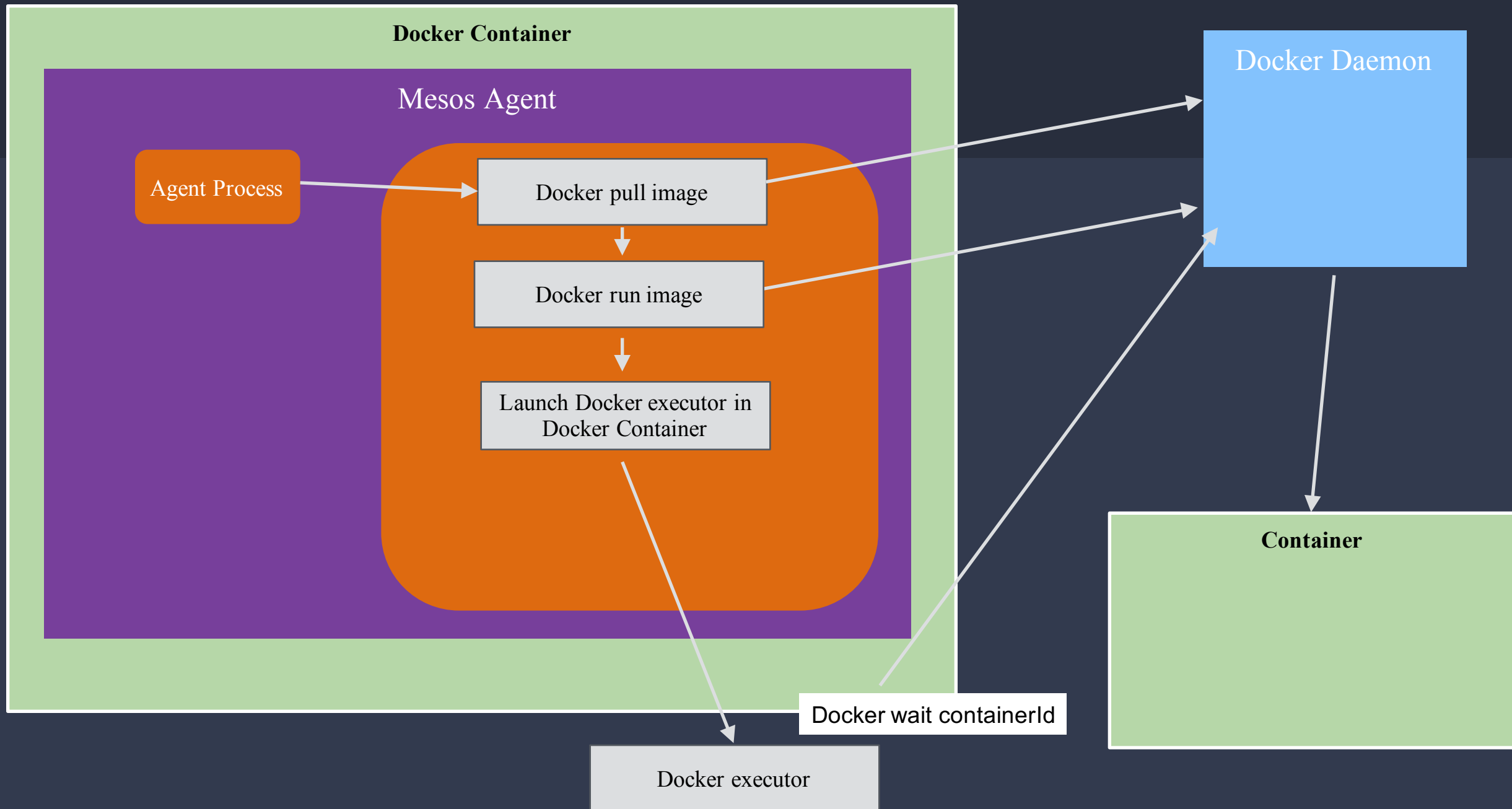
Users started to report problems

- Docker containers cannot be recovered after restart
- Unable to update container sizes

Running Mesos in Docker to run Docker

We realized users was running Mesos agent in Docker containers

- Executors are gone after Agent restarted, because all forked processes are reaped after container exited
- Mesos was unable to modify cgroup because we chrooted



Recovery again

- Find all running executors in Docker containers, rewait for those container's pid
- (What about when there are multiple agents running in the same box?)

Everything good now....?

- Docker moves fast, new command line options and new features available over time

```
if (flags.docker_mesos_image.isSome()) {  
  Try<Nothing> validateResult = docker->validateVersion(Version(1, 5, 0));  
  if (validateResult.isError()) {  
    string message = "Docker with mesos images requires docker 1.5+";  
    message += validateResult.error();  
    return Error(message);  
  }  
}
```

```
Try<Nothing> validateResult = docker->validateVersion(Version(1, 3, 0));  
ASSERT_SOME(validateResult)  
<< "-----\n"  
<< "We cannot run this test because of 'docker exec' command \n"  
<< "require docker version greater than '1.3.0'. You won't be \n"  
<< "able to use the docker exec method, but feel free to disable\n"  
<< "this test.\n"  
<< "-----";
```

Cgroups disappeared?

- Users started to report Mesos agent crashed after not able to recover containers and update them.
- Looking at the logs, all we see is that it wasn't able to find cgroups root at the same location anymore.
- How can a cgroup root disappear while the container is running?!

Systemd + Docker + Mesos love story

- After days of investigation, able to reproduce the problem when we run systemd config update, and noticing Docker containers cgroup just went missing.
(<https://issues.apache.org/jira/browse/MESOS-3009>)
- We started reading Systemd source code, looking for clues
- We realize Systemd migrates pids and recreates cgroups when processes finish according to some conditions

Systemd + Docker + Mesos love story continues

- Mesos will now create a Systemd specific container launch
- Registers a new systemd slice and uses the delegate flag to ask systemd not to migrate. (Delegate is not available in older versions.....)

We have a bigger problem...

- Docker works great, until you don't need to work around it
- Also works great when you don't have a lot of containers to run (stability problems)
- There has been a lot of changes since the beginning about Docker since 2 years ago (rkt, runc, containerd, etc)
- But it's still not enough to add features around Docker containers (GPU, isolators, etc)

What we really want from Docker

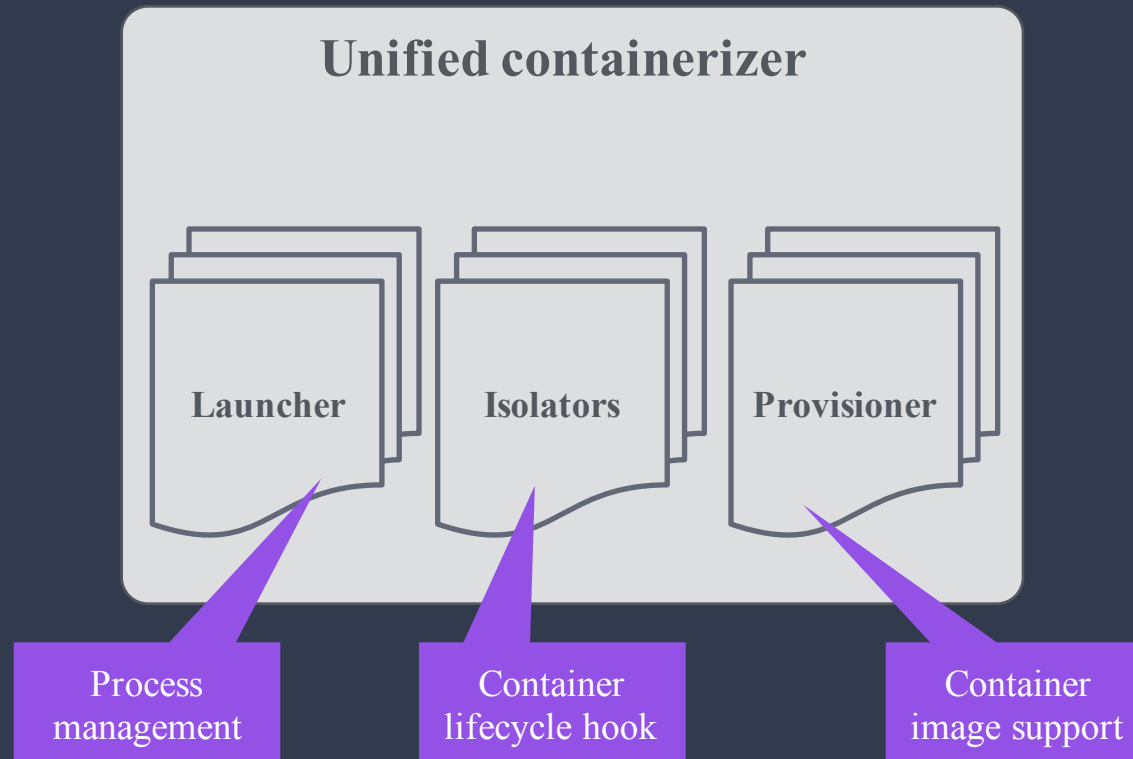
- We realize the biggest feature we want from Docker, is not the daemon, but just the image format

Unified containerizer

- Pluggable architecture
- **Container image**
- Container network
- Container storage
- Container security
- Customization and extensions

Unified containerizer

Pluggable architecture



Container image support

Start from 0.28, you can run your Docker container on Mesos without a Docker daemon installed!

- One less dependency in your stack

- Agent restart handled gracefully, task not affected

- Compose well with all existing isolators

- Easier to add extensions

Provisioner

Manage container images

Store: fetch and cache image layers

Backend: assemble rootfs from image layers

- E.g., copy, overlayfs, bind, aufs

Store can be extended

Currently supported: Docker, Appc

Plan to support: CVMFS (join the MesosCon talk!)

Container image framework API

```
message Image {
  enum Type {
    DOCKER = 1;
    APPC = 2;
  }
  required Type type;
  optional Appc appc;
  optional Docker docker;
  optional bool cached;
  message Docker { ... }
  message Appc { ... }
}
```


```
message TaskInfo {
  message ContainerInfo {
    enum Type {
      DOCKER = 1;
      MESOS = 2;
    }
    message MesosInfo {
      optional Image image;
    }
    required Type type;
    optional MesosInfo mesos;
  }
  optional ContainerInfo container;
}
```


Example: launch a Docker container w/ unified containerizer

```
TaskInfo {
```

```
  ...  
  "container" : {  
    "type" : "MESOS",  
    "mesos" : {  
      "image" : {  
        "type" : "DOCKER",  
        "docker" : { "name" : "busybox" }  
      }  
    }  
  }  
}
```

Instead of “DOCKER”,
which uses Docker
containerizer



More details can be found at:

<https://github.com/apache/mesos/blob/master/docs/container-image.md>

Future of containerization in Mesos

Future: unified containerizer!

Make it awesome

- Nested container
- VM support
- Unified fetching and caching
- Better abstraction for isolators

Nested container

Custom executor wants to create sub-containers

- Isolation between sub-containers

- Sub-containers have container images (e.g., Docker)

- When executor dies, sub-containers will be destroyed

Use cases:

- K8s on Mesos

- Jenkins on Mesos

- Native POD support

Future work

VM support

Goal: launching Mesos tasks/executors in VMs

Motivation and use cases

More secure containers

VM workload

OpenStack integration

Possible implementations

A new containerizer?

A plugin to unified containerizer?

Unified fetching and caching

Problems:

Different ways to fetch URIs and
container images

Cached in different places

Pluggable fetcher

A fetcher for each URI scheme

Allow URIs with custom scheme

Fetcher is modularized and can be
extended (e.g., p2p)

Unified caching

All artifacts are cached the same way

Content addressable storage

Garbage collection

Pre-fetching support

Better abstraction for isolators

Problems with the existing abstraction

- Cannot specify dependencies between isolators

- Sharing information between isolators is not possible

- Upgrading isolators in a backward compatible way is hard

Potential solutions

- Explicit isolator dependency, both data and control

- Isolator versioning, and version checkpointing

- Isolator registry?

Next big question to ask...

- **How to solve utilization, fairness and performance when running all of these services at scale?**

Thanks!

tnachen@gmail.com