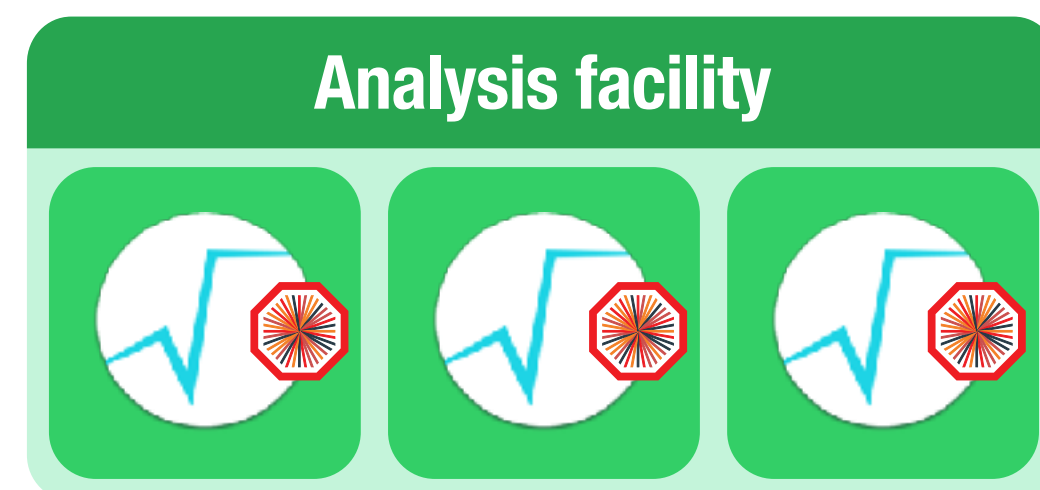
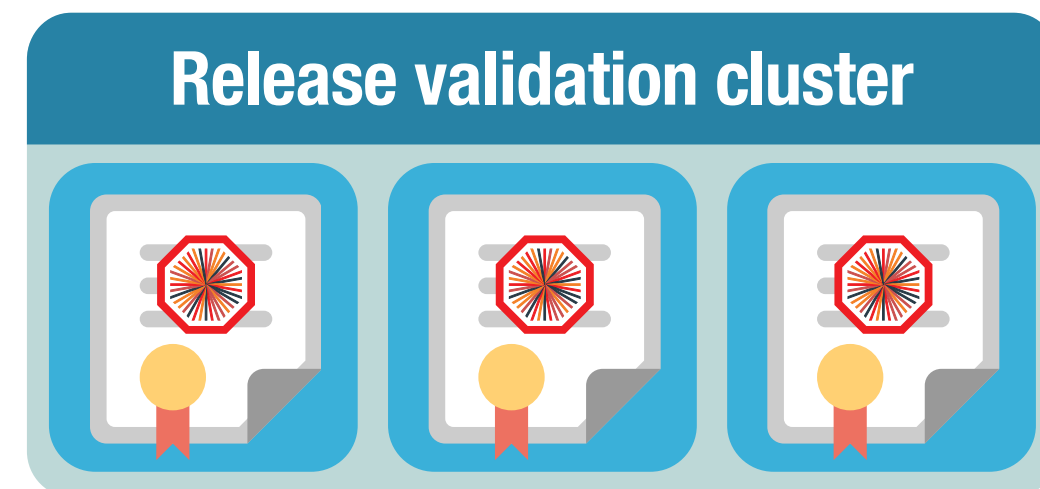


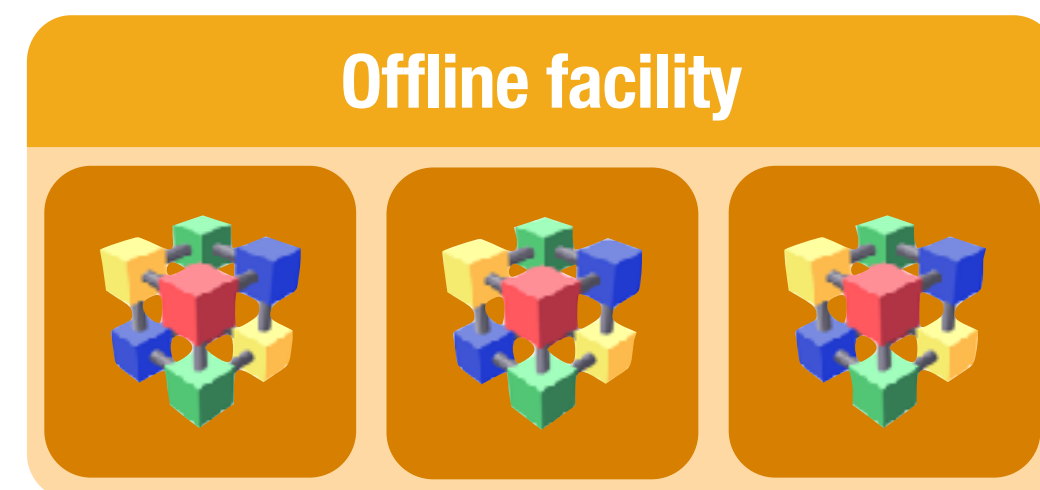
Experiences with the ALICE Mesos infrastructure

Dario Berzano · Giulio Eulisse
for the ALICE collaboration

ALICE computing infrastructures at CERN



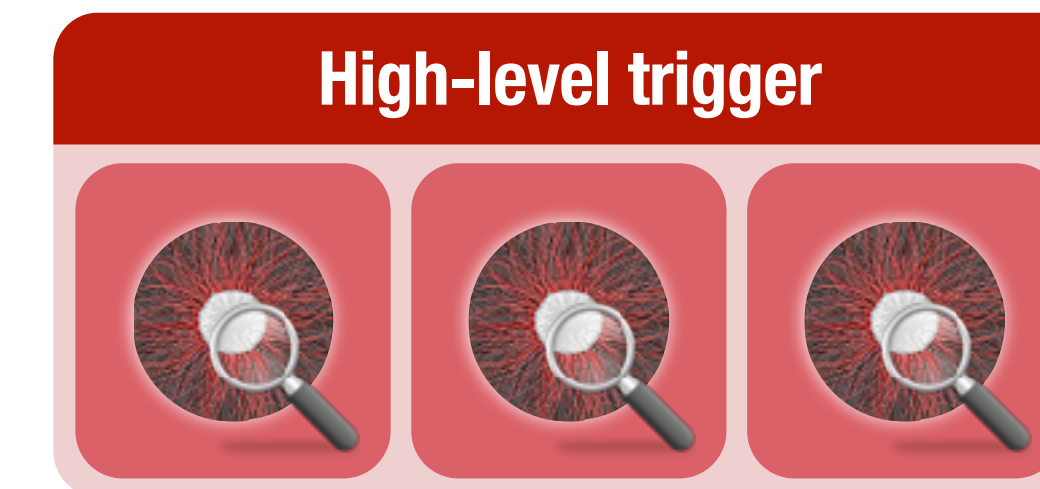
Dynamic on CERN OpenStack
(Slow-ish) IaaS approach



Powerful and not fully used
Could use it opportunistically
quite continuously



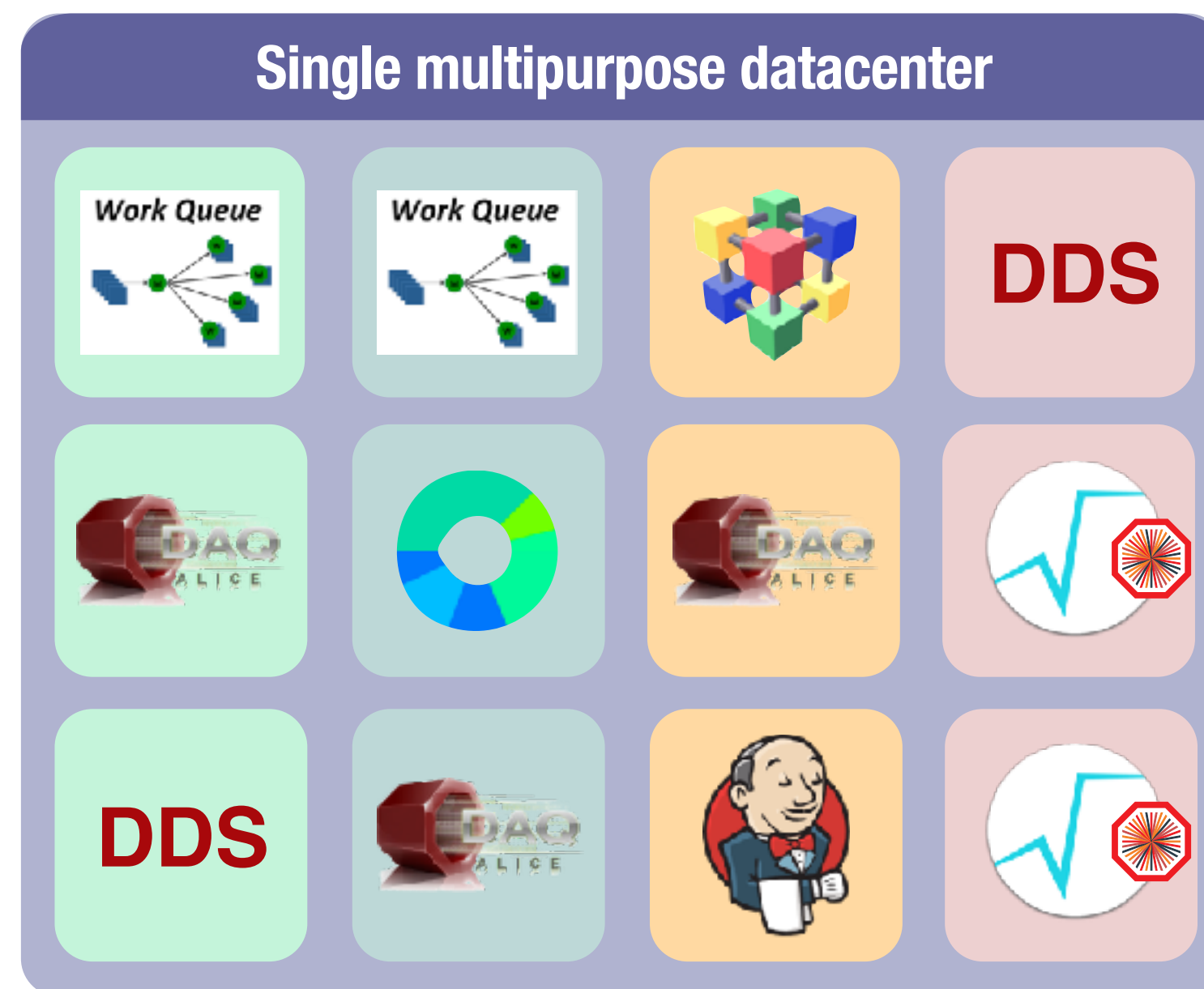
Tests for ALICE upgrade
combining Offline/Online
Design phase: needs a way to
be fully exploited



Made for data taking, used
opportunistically as Grid node
Data taking on bare metal, Grid
in dynamically deployed VMs

- So far: dynamic control using VMs and IaaS
Dynamically start VMs by looking at batch system
queues: has eased management but not always snappy
- We still have some resources underutilized
VM deployment not always viable (see DAQ), not all use
cases are batch queues
- **Unify infrastructures and cover more use cases**

Design of a modern datacenter



- Manage different infrastructures from a **single point**
- Never have **unused** resources
- Quickly **scale horizontally** - within minutes, not hours
- **High availability** - automatic failover and availability zones
- Handle **production** vs. **development** safely

Apache Mesos in production



- **Apache Mesos:** “program against your datacenter like it’s a single pool of resources”
- Born in research; used by many industry major players
Originally developed at UC Berkeley, now it’s an Apache Foundation open source project.
Used in production at [Twitter](#), [Apple](#), [Netflix](#), [NASA JPL](#), [Airbnb](#)...
- Open source backed by many companies
Plug-n-play solutions available from [Mesosphere](#), [Cisco](#), [Rancher Labs](#)...
- An [operating system](#) for large clouds
Schedules tasks like an OS does with processes, known to work for 10k nodes

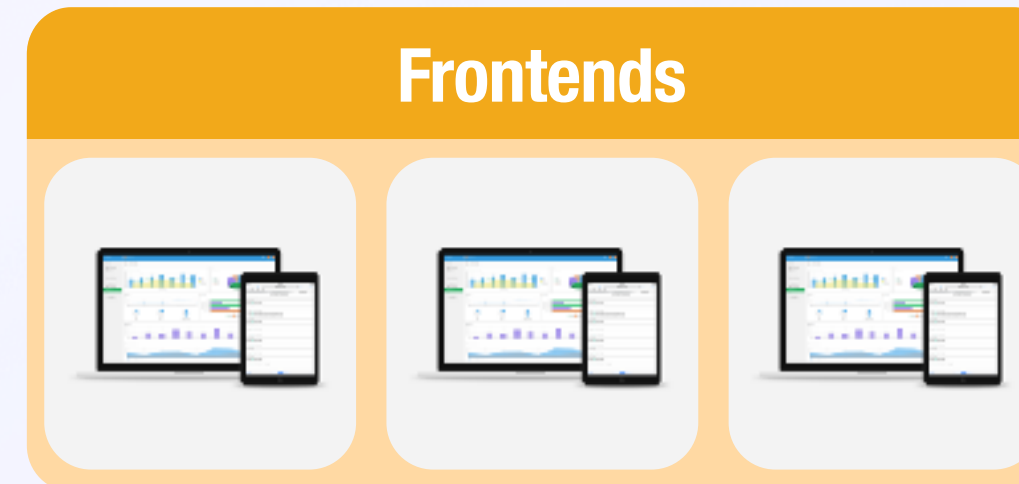
Our Mesos infrastructure setup at CERN for high availability

Pets

the important nodes



- ▶ Different availability zones
- ▶ Automatic leader selection
- ▶ Control/debug web interface
- ▶ Service discovery via DNS



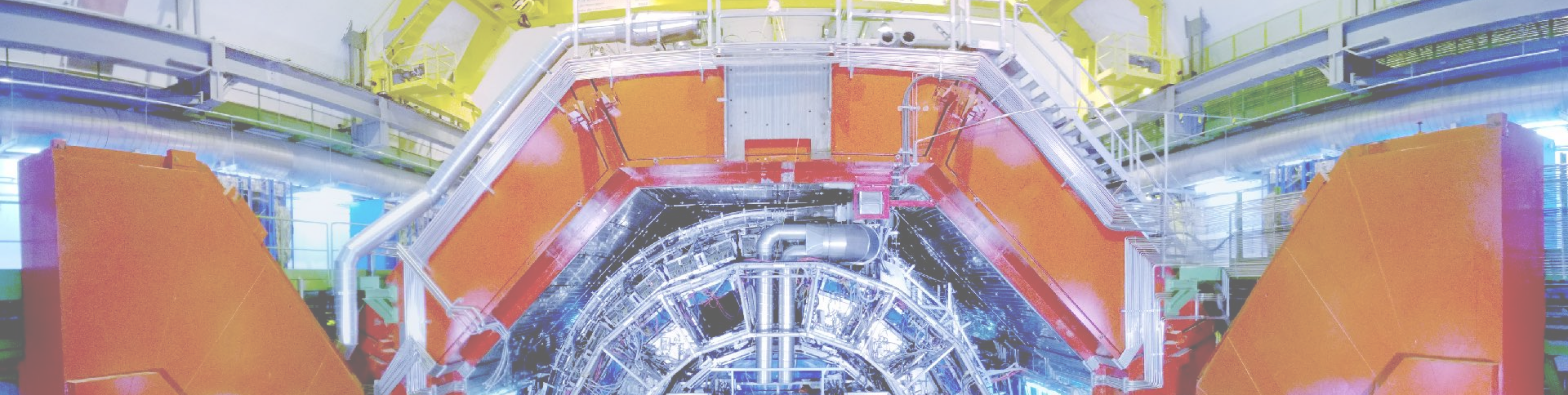
- ▶ Single sign-on
- ▶ Route to internal services
- ▶ Fixed nodes needed as internal services roam freely

Cattle

replaceable if they fail



- ▶ Run (mainly) containerized tasks on hosts
- ▶ Manage actual resources
- ▶ Update possible with no service disruption



ALICE applications on Mesos



Three ways to run applications on Mesos

- Mesos is an **approach** rather than a tool: brings **resource knowledge** to your application and guarantees **fair share**
- Base approach: use an **existing tool** to deploy your existing applications
Zero coding: run/scale tasks with e.g. Marathon, Aurora, Jenkins and get started quickly. We use Marathon extensively for many microservices (web servers, bots...)
- Write a **custom tool (framework)** to integrate an existing use case
A “framework” receives resource offers from Mesos either via the HTTP interface or via C++/Python/Java bindings and decides what to do with them
- Write a fully fledged **application scheduler** yourself
Schedule different tasks within your application using resource offers from Mesos. Apple does it with J.A.R.V.I.S., used to drive Siri
- **All those use cases can coexist: Mesos is made to support multiple frameworks**

Long-running services with Marathon

- Migrated some existing services to Mesos for HA
- Existing and new services are now shipped as Docker containers
- Fixed frontend nodes dynamically mapped to internal Marathon services

MARATHON

Applications

Deployments

Search all applications

STATUS

☐

Running

24

☐

Deploying

☐

Suspended

4

☐

Delayed

☐

Waiting

3

HEALTH

☐

Healthy

15

☐

Unhealthy

☐

Unknown

12

LABEL

Select

RESOURCES

☐

Volumes

Applications

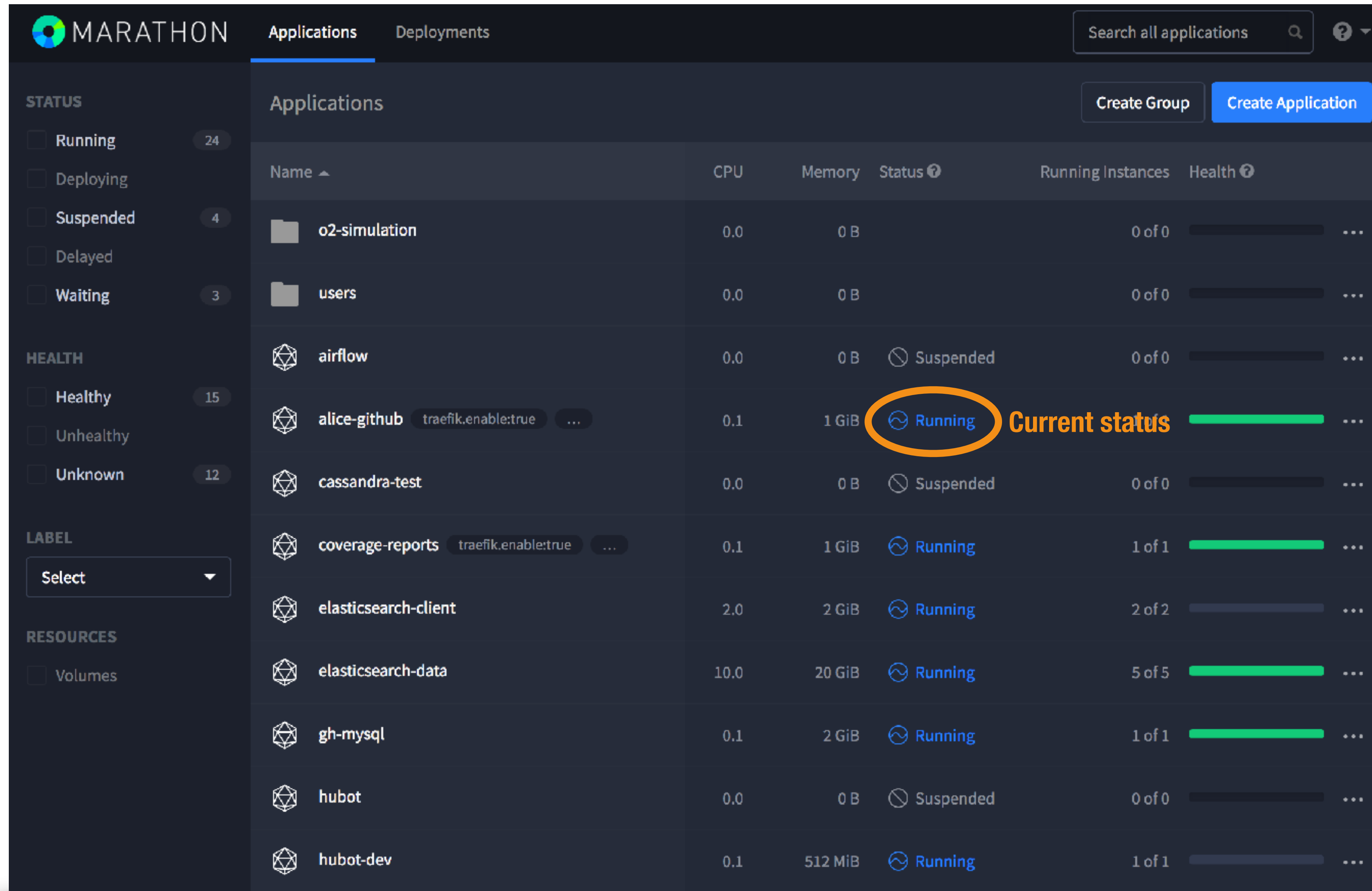
Create Group

Create Application

Name	CPU	Memory	Status	Running Instances	Health
<div><div></div><div>o2-simulation</div></div>	0.0	0 B		0 of 0	
<div><div></div><div>users</div></div>	0.0	0 B		0 of 0	
<div><div></div><div>airflow</div></div>	0.0	0 B	<div><div></div><div>Suspended</div></div>	0 of 0	
<div><div></div><div>alice-github</div><div><div>traefik.enable:true</div><div></div></div></div>	0.1	1 GiB	<div><div></div><div>Running</div></div>	1 of 1	
<div><div></div><div>cassandra-test</div></div>	0.0	0 B	<div><div></div><div>Suspended</div></div>	0 of 0	
<div><div></div><div>coverage-reports</div><div><div>traefik.enable:true</div><div></div></div></div>	0.1	1 GiB	<div><div></div><div>Running</div></div>	1 of 1	
<div><div></div><div>elasticsearch-client</div></div>	2.0	2 GiB	<div><div></div><div>Running</div></div>	2 of 2	
<div><div></div><div>elasticsearch-data</div></div>	10.0	20 GiB	<div><div></div><div>Running</div></div>	5 of 5	
<div><div></div><div>gh-mysql</div></div>	0.1	2 GiB	<div><div></div><div>Running</div></div>	1 of 1	
<div><div></div><div>hubot</div></div>	0.0	0 B	<div><div></div><div>Suspended</div></div>	0 of 0	
<div><div></div><div>hubot-dev</div></div>	0.1	512 MiB	<div><div></div><div>Running</div></div>	1 of 1	

Long-running services with Marathon

- Migrated some existing services to Mesos for HA
- Existing and new services are now shipped as Docker containers
- Fixed frontend nodes dynamically mapped to internal Marathon services



The screenshot displays the Marathon web interface. On the left, there are filters for STATUS (Running: 24, Deploying, Suspended: 4, Delayed, Waiting: 3), HEALTH (Healthy: 15, Unhealthy, Unknown: 12), LABEL (Select), and RESOURCES (Volumes). The main panel shows a table of applications under the 'Applications' tab. The table columns are Name, CPU, Memory, Status, Running Instances, and Health. The 'alice-github' application is highlighted with an orange circle around its 'Running' status, with the text 'Current status' written next to it. Other applications include 'o2-simulation', 'users', 'airflow', 'cassandra-test', 'coverage-reports', 'elasticsearch-client', 'elasticsearch-data', 'gh-mysql', 'hubot', and 'hubot-dev'.

Name	CPU	Memory	Status	Running Instances	Health
o2-simulation	0.0	0 B		0 of 0	
users	0.0	0 B		0 of 0	
airflow	0.0	0 B	Suspended	0 of 0	
alice-github	0.1	1 GiB	Running		
cassandra-test	0.0	0 B	Suspended	0 of 0	
coverage-reports	0.1	1 GiB	Running	1 of 1	
elasticsearch-client	2.0	2 GiB	Running	2 of 2	
elasticsearch-data	10.0	20 GiB	Running	5 of 5	
gh-mysql	0.1	2 GiB	Running	1 of 1	
hubot	0.0	0 B	Suspended	0 of 0	
hubot-dev	0.1	512 MiB	Running	1 of 1	

Long-running services with Marathon

- Migrated some existing services to Mesos for HA
- Existing and new services are now shipped as Docker containers
- Fixed frontend nodes dynamically mapped to internal Marathon services

The screenshot shows the Marathon web interface with the 'Applications' tab selected. The interface includes a sidebar with filters for STATUS (Running: 24, Deploying, Suspended: 4, Delayed, Waiting: 3), HEALTH (Healthy: 15, Unhealthy, Unknown: 12), LABEL (Select), and RESOURCES (Volumes). The main table lists applications with columns for Name, CPU, Memory, Status, Running Instances, and Health. Two applications, 'alice-github' and 'coverage-reports', are highlighted with orange circles. 'alice-github' has a status of 'Running' and 1 GiB of memory. 'coverage-reports' has a status of 'Running' and 1 GiB of memory. The text 'Current status' is written in orange next to the 'Running' status of 'alice-github', and 'Fine grained resources specification' is written in orange below the 'coverage-reports' row.

Name	CPU	Memory	Status	Running Instances	Health
o2-simulation	0.0	0 B		0 of 0	
users	0.0	0 B		0 of 0	
airflow	0.0	0 B	Suspended	0 of 0	
alice-github	0.1	1 GiB	Running		
cassandra-test	0.0	0 B	Suspended	0 of 0	
coverage-reports	0.1	1 GiB	Running	1 of 1	
elasticsearch-client	0.1	1 GiB	Running	2 of 2	
elasticsearch-data	10.0	20 GiB	Running	5 of 5	
gh-mysql	0.1	2 GiB	Running	1 of 1	
hubot	0.0	0 B	Suspended	0 of 0	
hubot-dev	0.1	512 MiB	Running	1 of 1	

Long-running services with Marathon

- Migrated some existing services to Mesos for HA
- Existing and new services are now shipped as Docker containers
- Fixed frontend nodes dynamically mapped to internal Marathon services

The screenshot shows the Marathon web interface with the 'Applications' tab selected. The interface includes a sidebar with filters for STATUS (Running: 24, Deploying, Suspended: 4, Delayed, Waiting: 3), HEALTH (Healthy: 15, Unhealthy, Unknown: 12), LABEL (Select), and RESOURCES (Volumes). The main table lists applications with columns for Name, CPU, Memory, Status, Running Instances, and Health. Annotations highlight specific features:

- Current status:** An orange circle around the 'Running' status icon for the 'alice-github' application.
- Fine grained resources specification:** An orange circle around the '0.1' CPU and '1 GiB' Memory values for the 'coverage-reports' application.
- Deployment status with custom health checks:** An orange circle around the '5 of 5' running instances and the green health bar for the 'elasticsearch-data' application.
- Unhealthy apps (e.g. HTTP not responding) are restarted:** A red text annotation pointing to the 'gh-mysql' application, which is marked as 'Running' with a green health bar.

Name	CPU	Memory	Status	Running Instances	Health
o2-simulation	0.0	0 B		0 of 0	
users	0.0	0 B		0 of 0	
airflow	0.0	0 B	Suspended	0 of 0	
alice-github	0.1	1 GiB	Running		
cassandra-test	0.0	0 B	Suspended	0 of 0	
coverage-reports	0.1	1 GiB	Running	1 of 1	
elasticsearch-client	0.1	1 GiB	Running	2 of 2	
elasticsearch-data	10.0	20 GiB	Running	5 of 5	
gh-mysql	0.1	2 GiB	Running	1 of 1	
hubot	0.0	0 B	Suspended	0 of 0	
hubot-dev	0.1	512 MiB	Running	1 of 1	

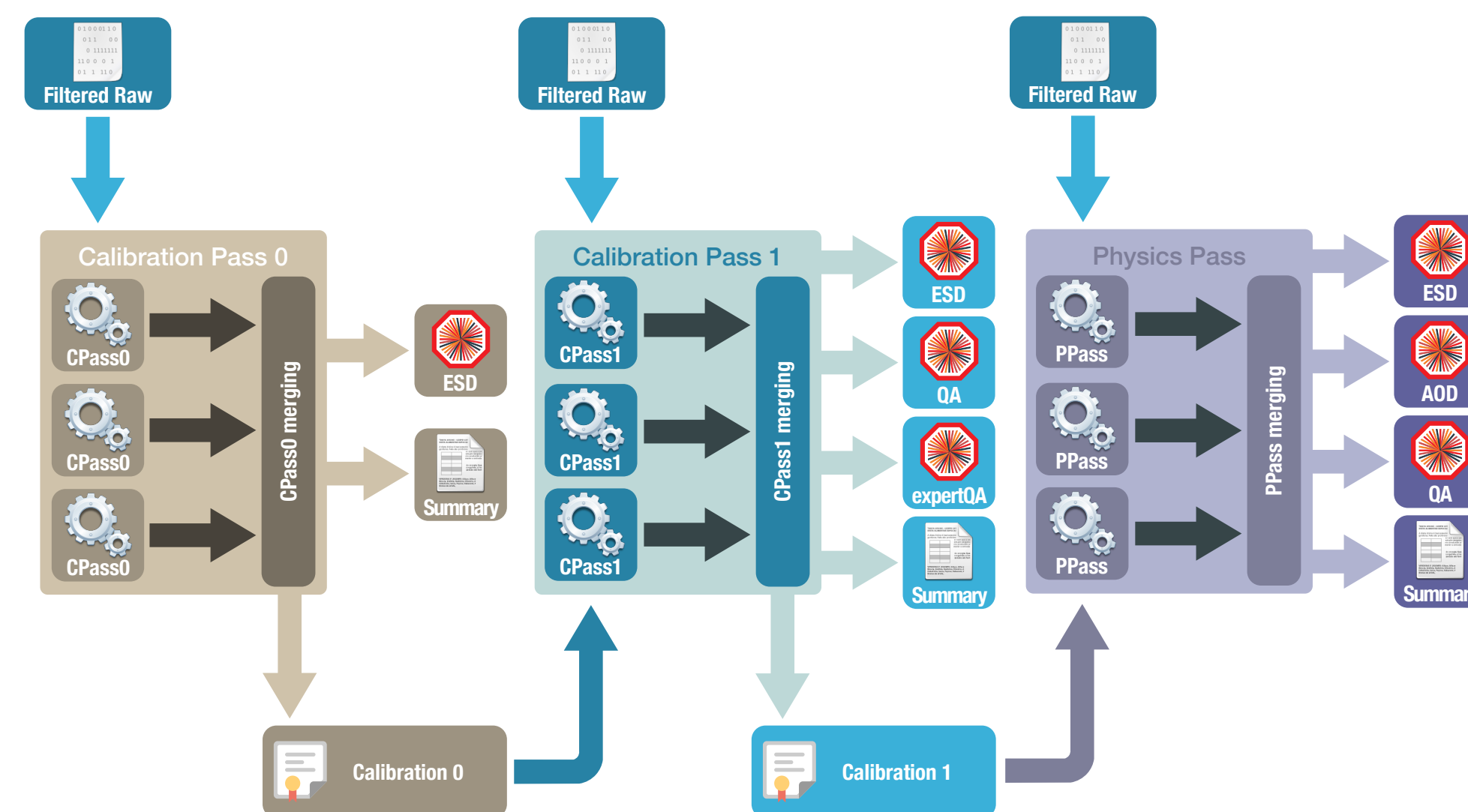
ALICE Continuous Integration

- **Builds and continuous integration with the existing [Jenkins Mesos](#) plugin**
- Good testbed for Mesos: we need [high availability](#) under [unpredictable load](#)
Automatic pull requests test: they might come at any time in various amounts and we must be able to process all of them
- [Decouple](#) build environment from cluster nodes architecture
Running obsolete [SLC5](#) build containers on modern CERN CentOS 7 nodes
- [Friendly](#) to end users
User interacts only via Git commands and GitHub web interface: the underlying infrastructure's complexity is invisible

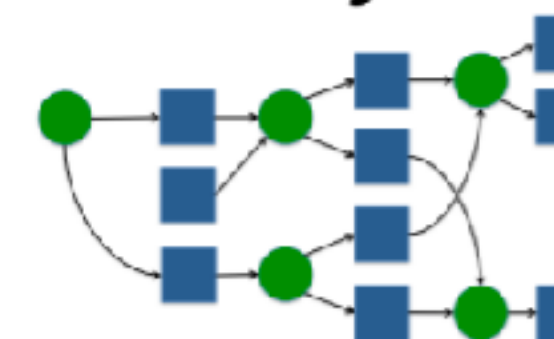
The screenshot shows a GitHub pull request interface. At the top, the title is "Update Mesos to 0.28.2 and improve dependencies #529". Below the title, it says "dberzano wants to merge 1 commit into alisw:IB/v5-08/next from dberzano:mesos". There are tabs for "Conversation" (1), "Commits" (1), and "Files changed" (5). Below the tabs, there is a message: "Add more commits by pushing to the mesos branch on dberzano/alidist." A yellow box highlights the status checks section, which includes: "All checks have failed" (1 failing check), "Jenkins job PR-529 — This commit cannot be built" (with a "Details" link), and "This branch has no conflicts with the base branch" (Merging can be performed automatically.). At the bottom, there is a "Squash and merge" button and a link to "open this in GitHub Desktop" or view "command line instructions".

ALICE Release Validation: workflow

- Full validation jobs on a Release Candidate
Define job dependencies with [Makeflow](#), submit to a lightweight agents system powered by [Work Queue](#) (usually ran on a batch system)
- Mesos gives resources quickly when needed
We don't always run it, but when we do we need our resources promptly
- **Pre-existing use case integrated with Mesos**
Makeflow+Work Queue used already: running on Mesos thanks to our custom C++ framework



Makeflow



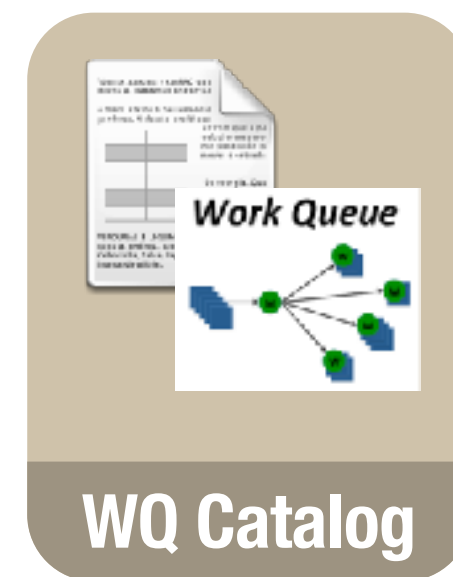
Work Queue



Part of cctools
ccl.cse.nd.edu

The Mesos Work Queue framework

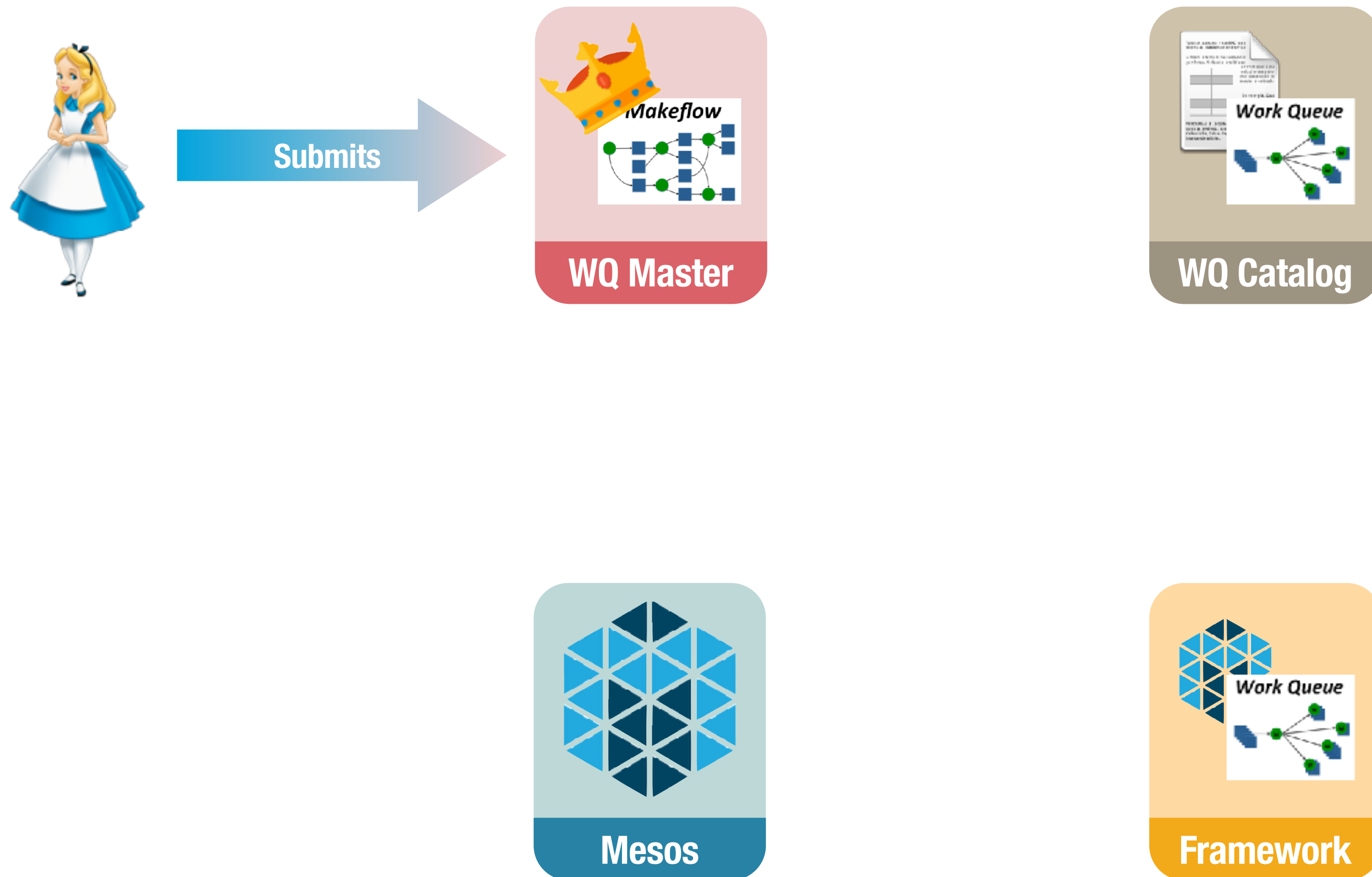
- Makeflow job submitted as before
- Work Queue catalog knows how many resources are needed
- **Work Queue code not modified**



github.com/alisw/mesos-workqueue

The Mesos Work Queue framework

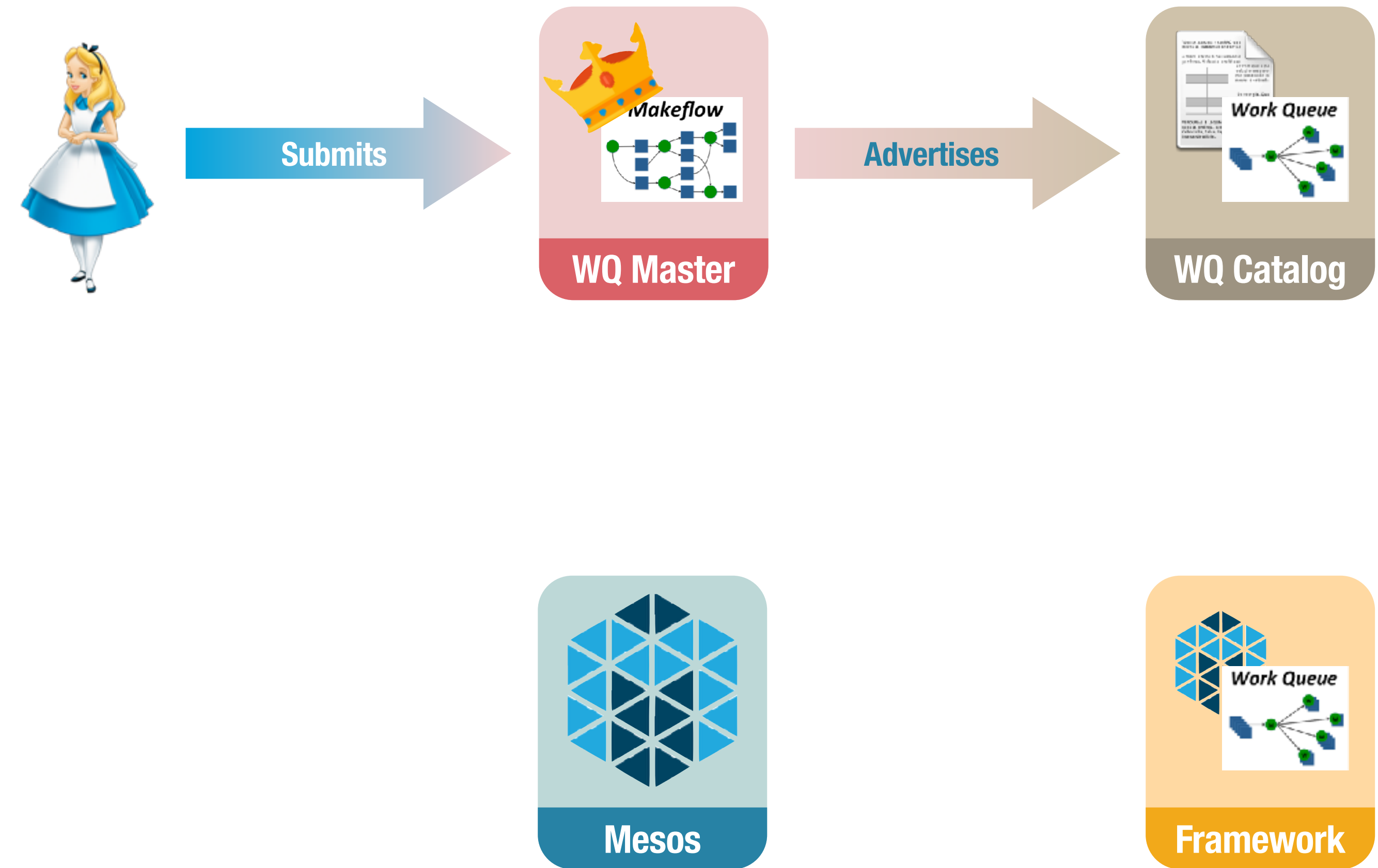
- Makeflow job submitted as before
- Work Queue catalog knows how many resources are needed
- **Work Queue code not modified**



github.com/alisw/mesos-workqueue

The Mesos Work Queue framework

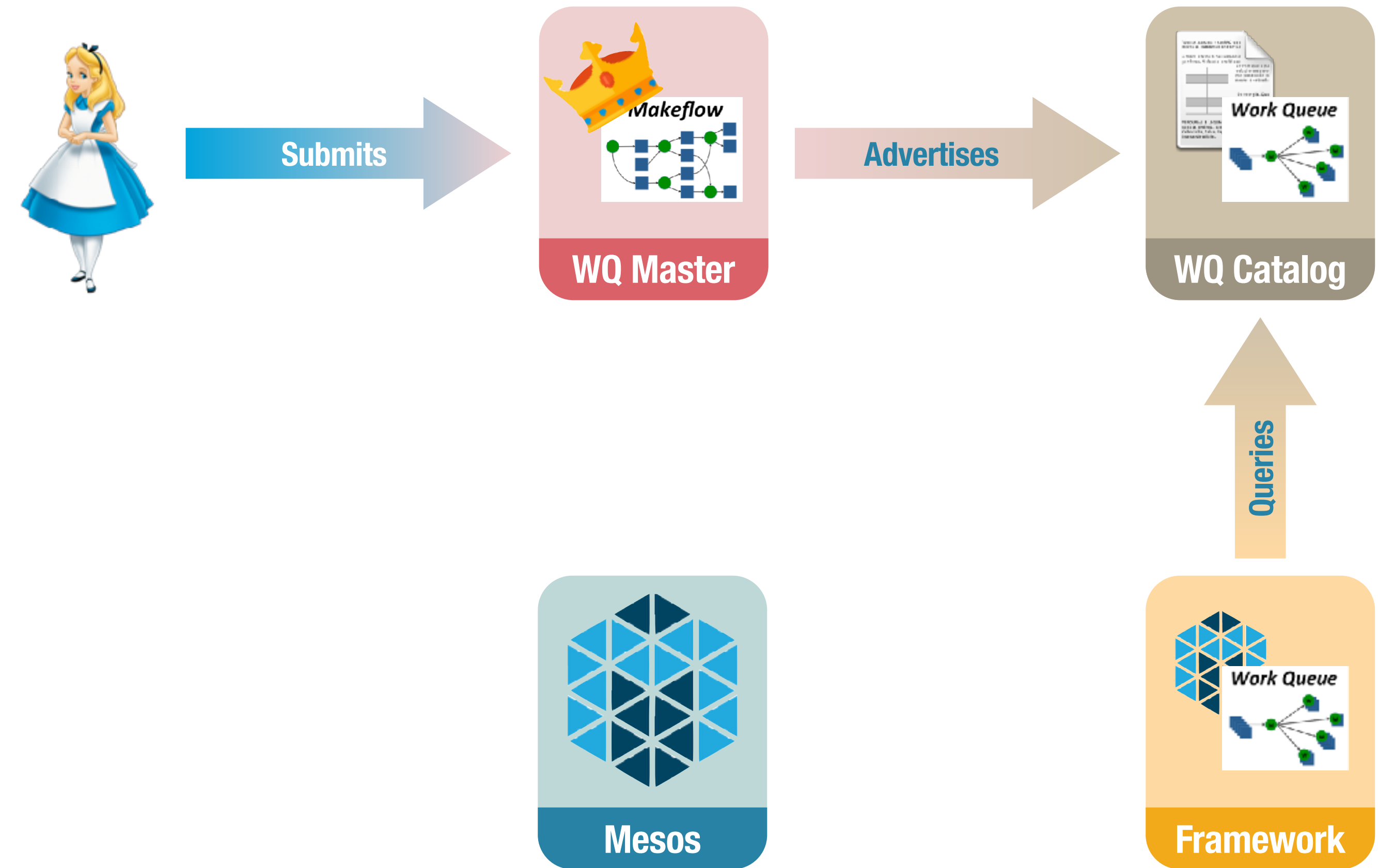
- Makeflow job submitted as before
- Work Queue catalog knows how many resources are needed
- **Work Queue code not modified**



github.com/alisw/mesos-workqueue

The Mesos Work Queue framework

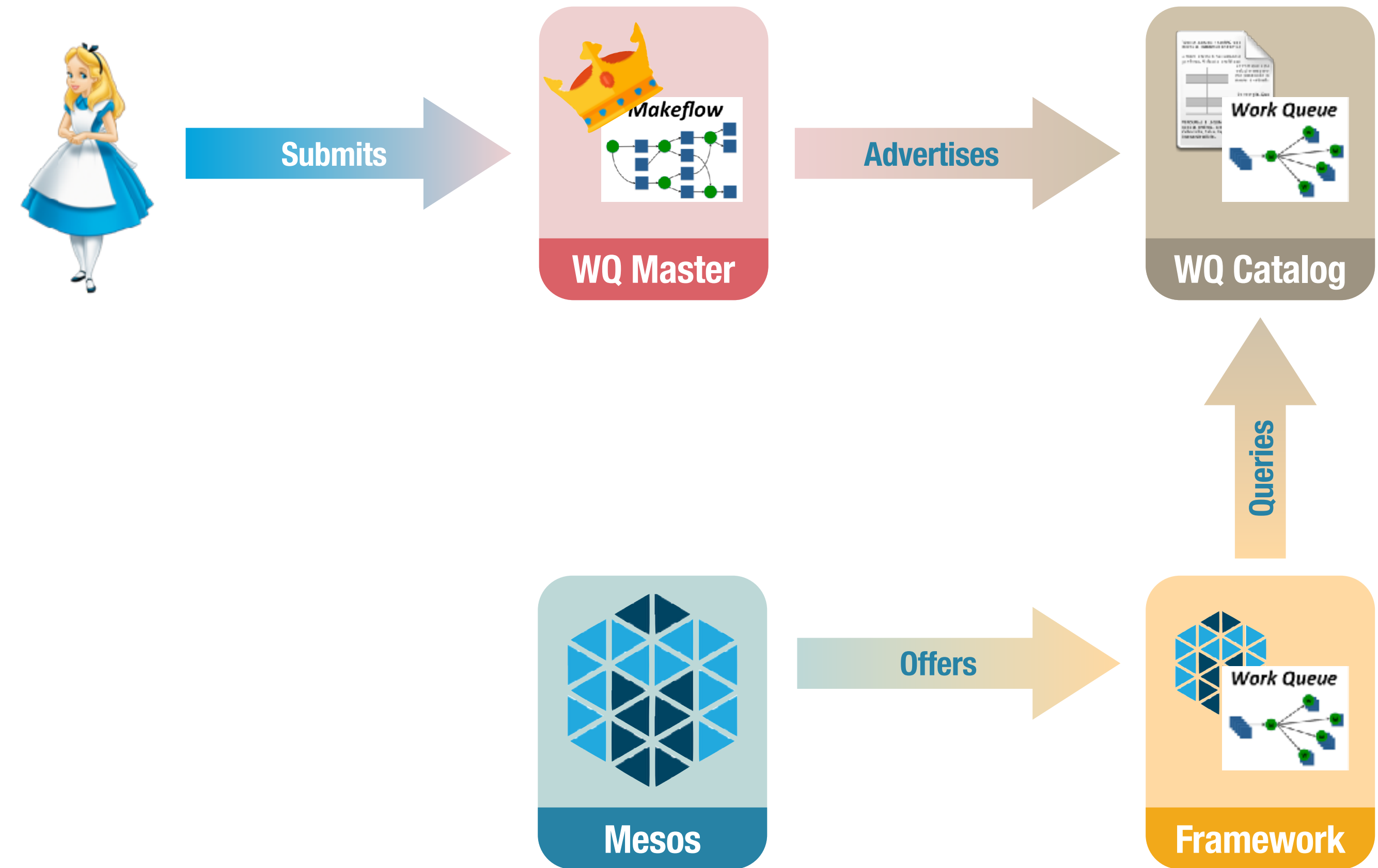
- Makeflow job submitted as before
- Work Queue catalog knows how many resources are needed
- **Work Queue code not modified**



github.com/alisw/mesos-workqueue

The Mesos Work Queue framework

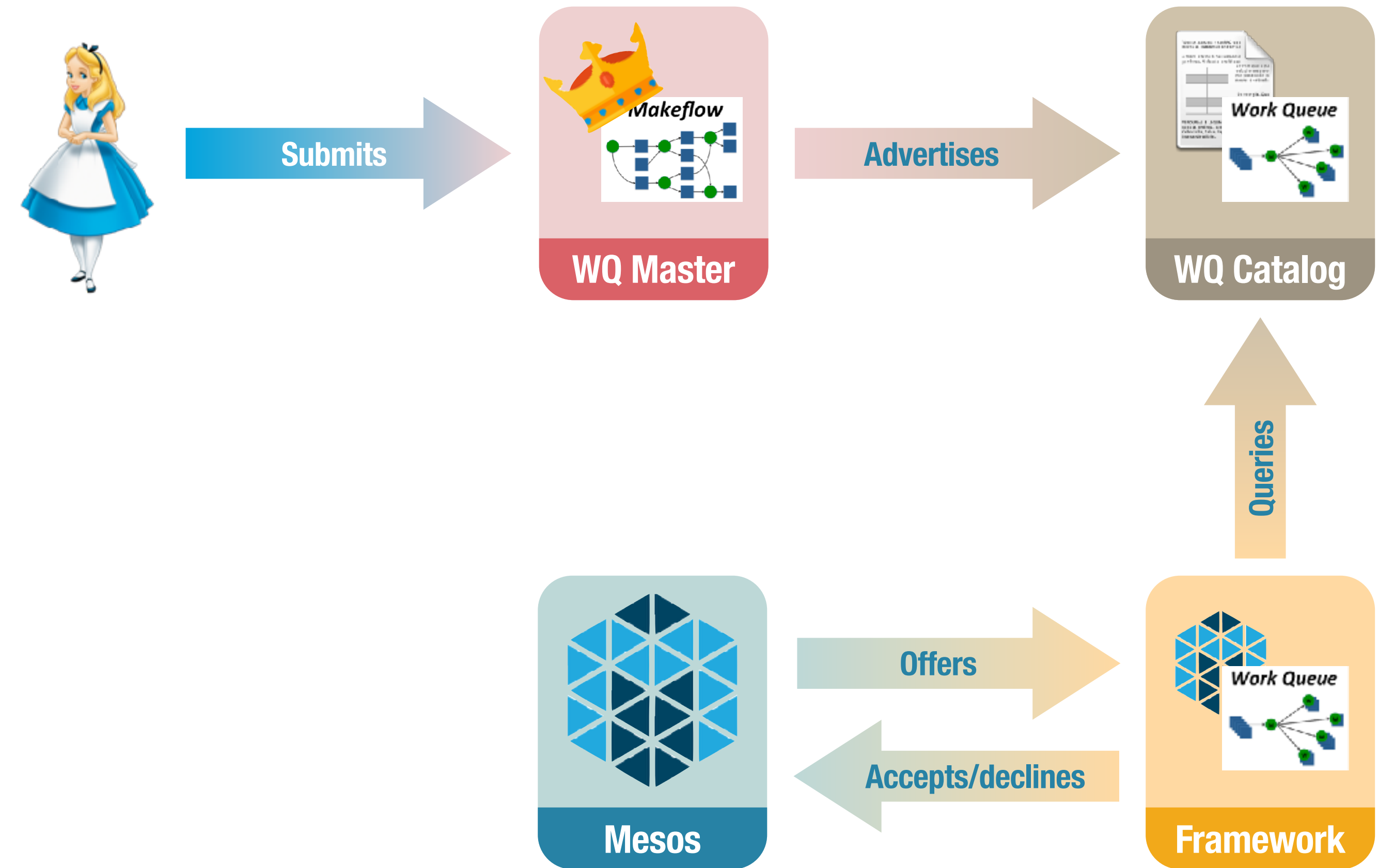
- Makeflow job submitted as before
- Work Queue catalog knows how many resources are needed
- **Work Queue code not modified**



github.com/alisw/mesos-workqueue

The Mesos Work Queue framework

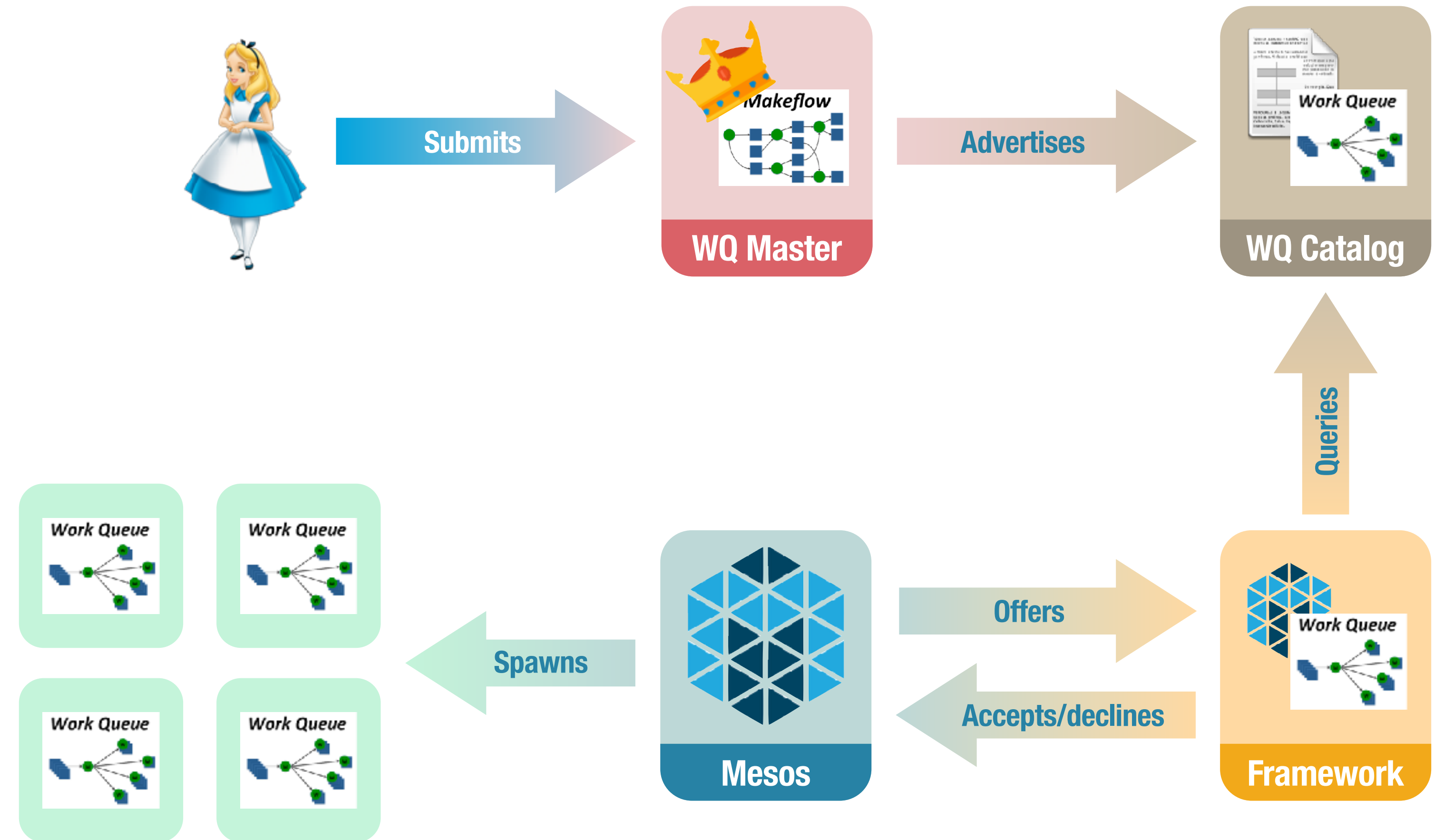
- Makeflow job submitted as before
- Work Queue catalog knows how many resources are needed
- **Work Queue code not modified**



github.com/alisw/mesos-workqueue

The Mesos Work Queue framework

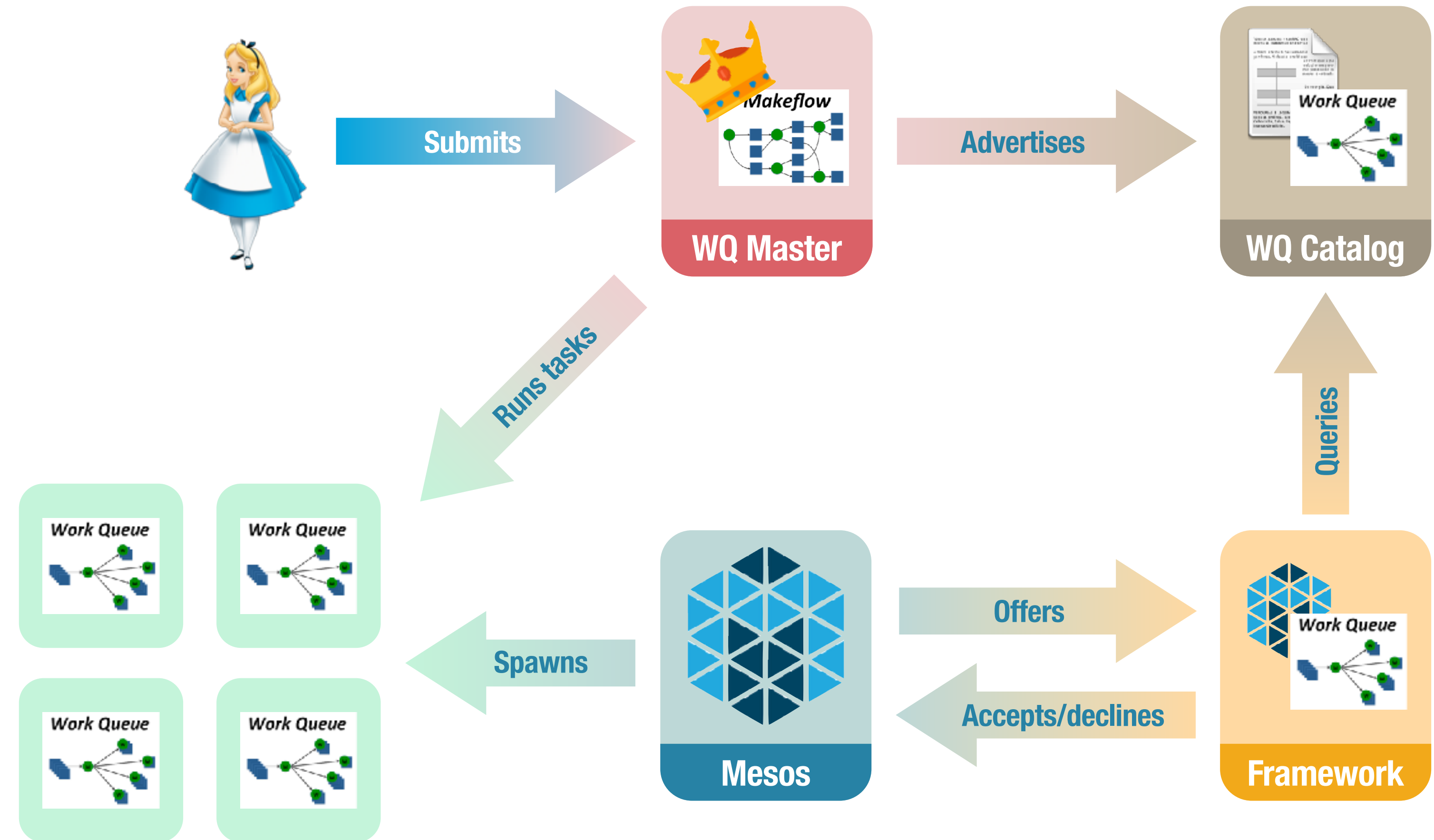
- Makeflow job submitted as before
- Work Queue catalog knows how many resources are needed
- **Work Queue code not modified**



github.com/alisw/mesos-workqueue

The Mesos Work Queue framework

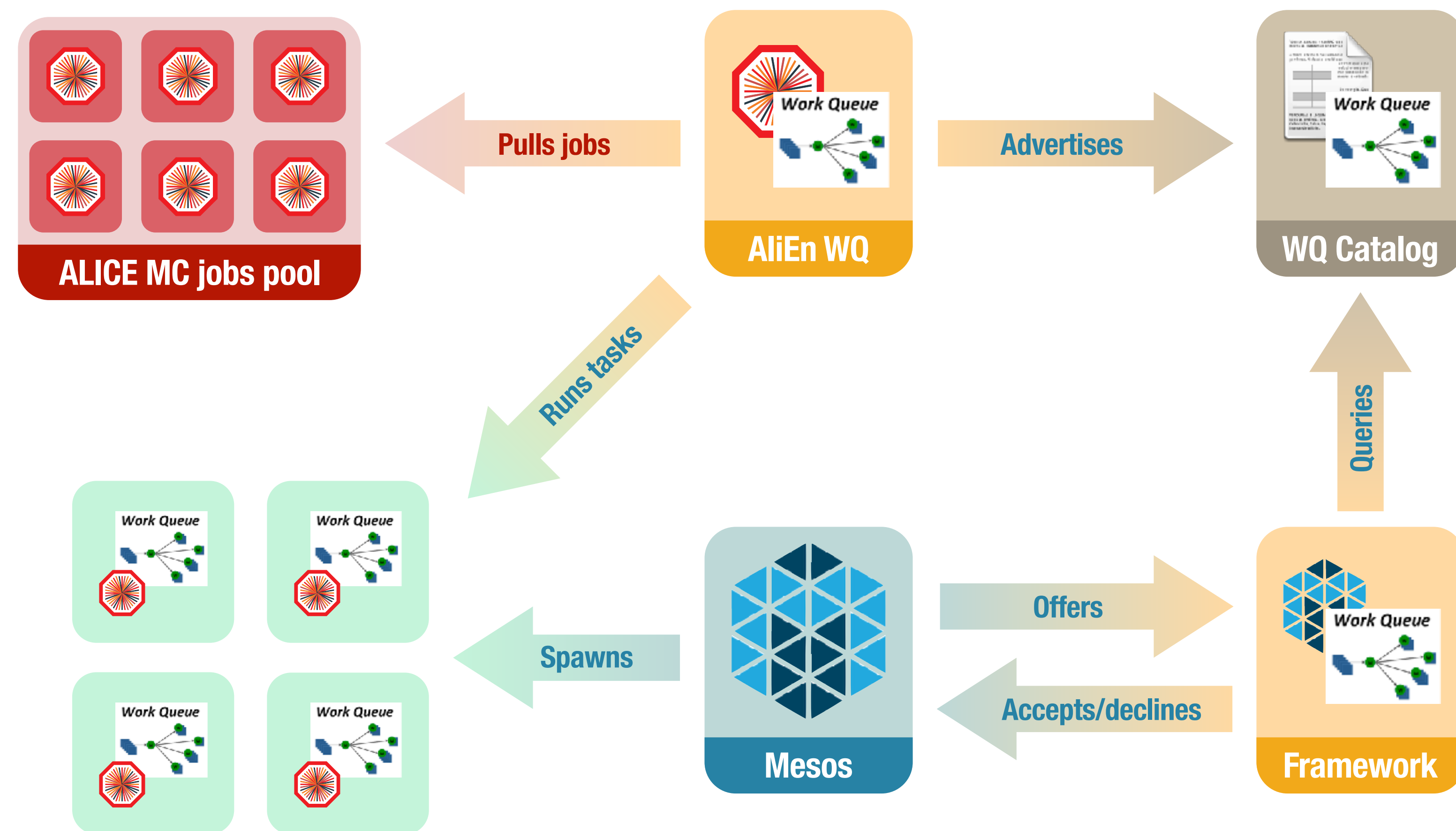
- Makeflow job submitted as before
- Work Queue catalog knows how many resources are needed
- **Work Queue code not modified**



github.com/alisw/mesos-workqueue

Monte Carlo Grid jobs on Mesos

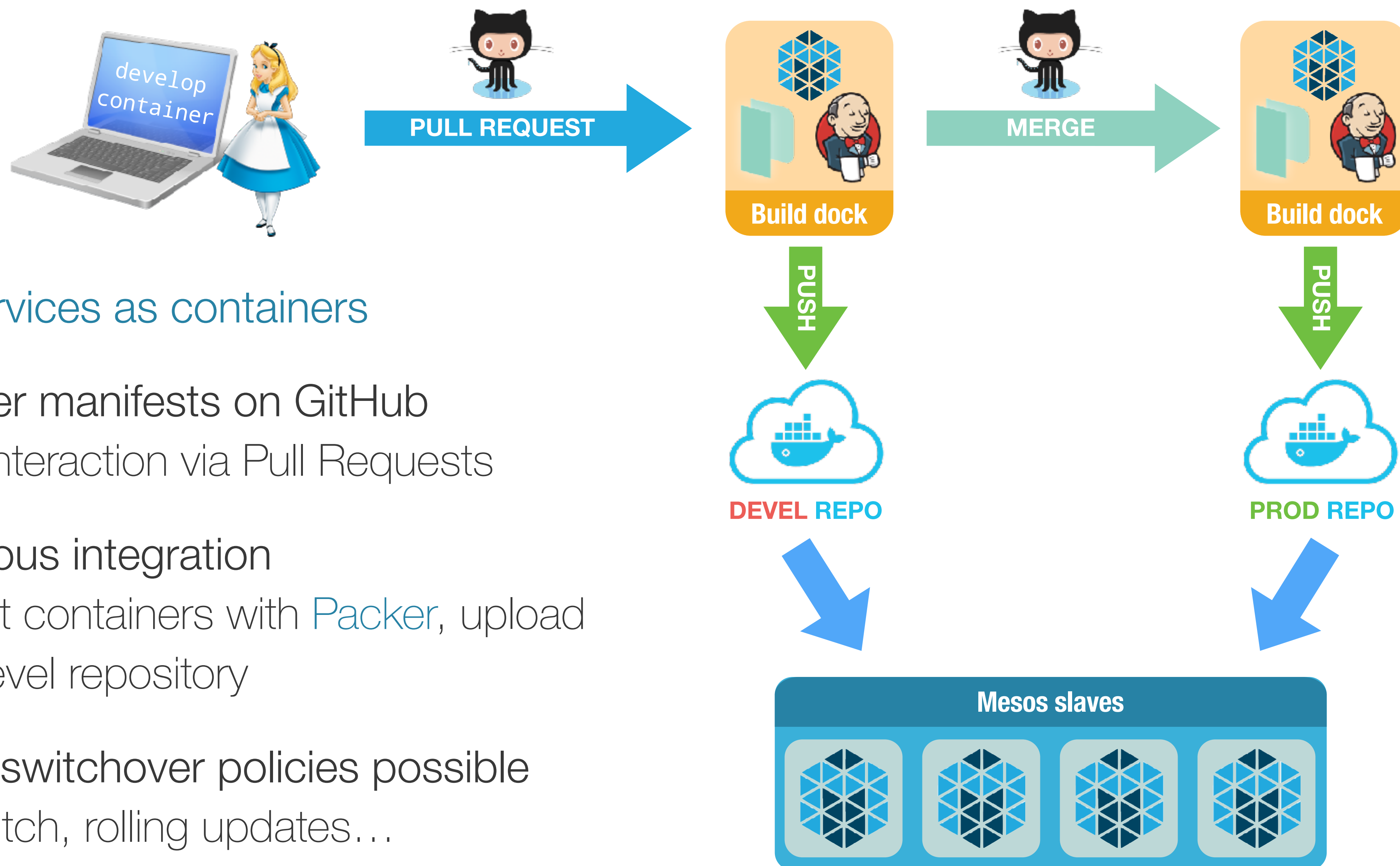
- Custom C++ adapter enables Mesos to run ALICE Grid jobs
- Reusing the Mesos Work Queue framework
- Limit to Monte Carlo: CPU intensive, fill the Mesos cluster when idle
- Original ALICE Grid software (AliEn) was not modified

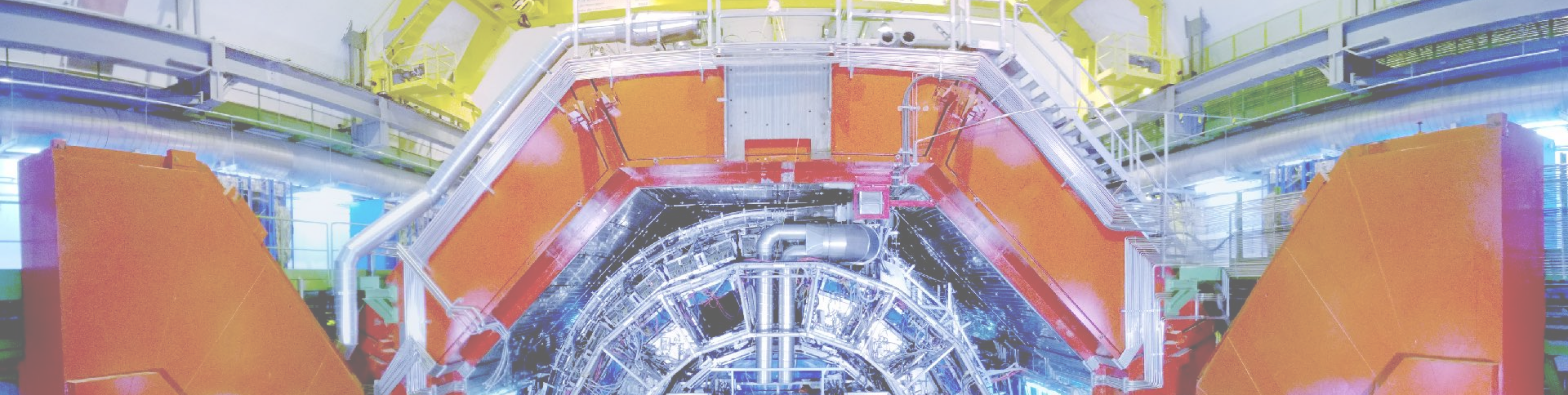


github.com/alisw/alien-workqueue

Testing and deploying container updates

- Microservices as containers
- Container manifests on GitHub
Natural interaction via Pull Requests
- Continuous integration
Build test containers with [Packer](#), upload to the devel repository
- Multiple switchover policies possible
Hard switch, rolling updates...

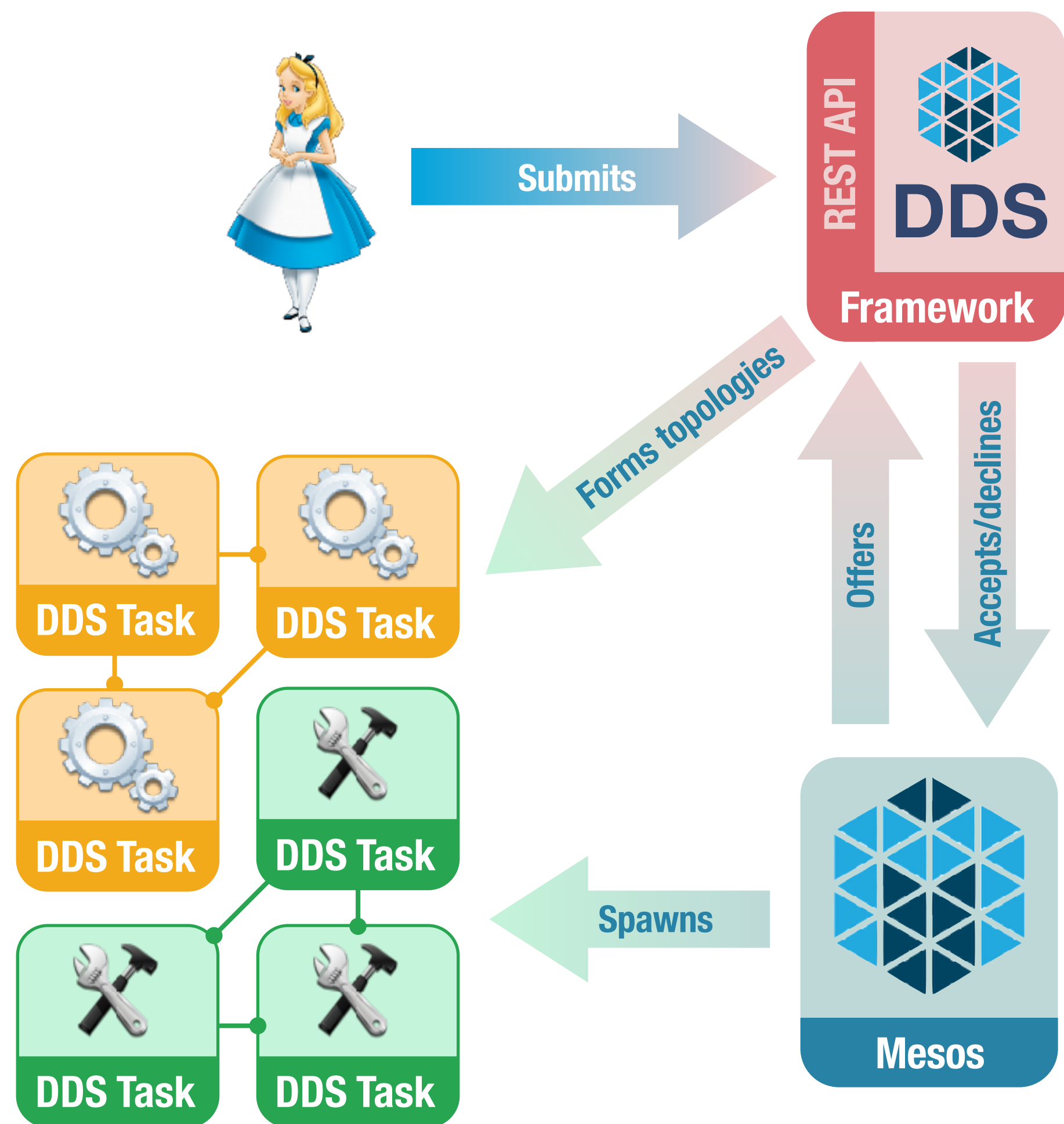




ALICE Run 3 Online/Offline on Mesos



Dynamic Deployment System on Mesos

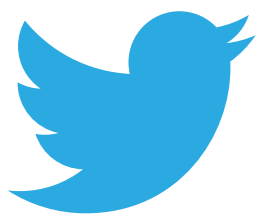
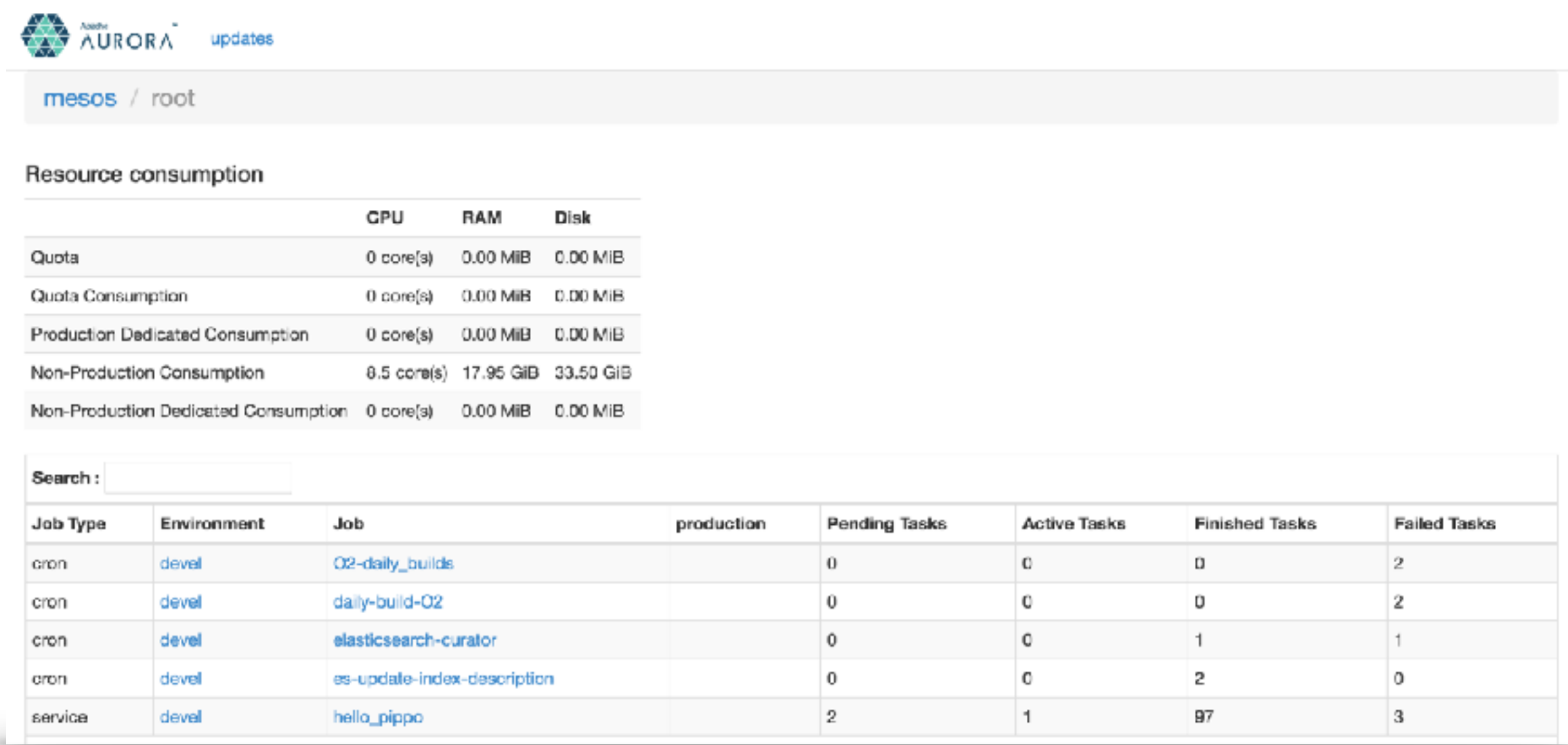


github.com/alisw/mesos-dds

- **ALICE Run 3: Online and Offline processing on the same cluster with defined task topologies**
- Dynamic Deployment System: dds.gsi.de
Baseline for ALICE Run 3: we use it to define and launch tasks interconnected by a certain topology
- Run DDS workers as Mesos jobs
Leverage Mesos to bring resource knowledge to DDS and retain the same user tasks submission interface
- **The DDS Mesos plugin**
We have written a C++ plugin using Mesos as a “scheduler for our scheduler” without modifying DDS: we can share DDS resources with all other use cases

Future of ALICE software: datacenter with Aurora

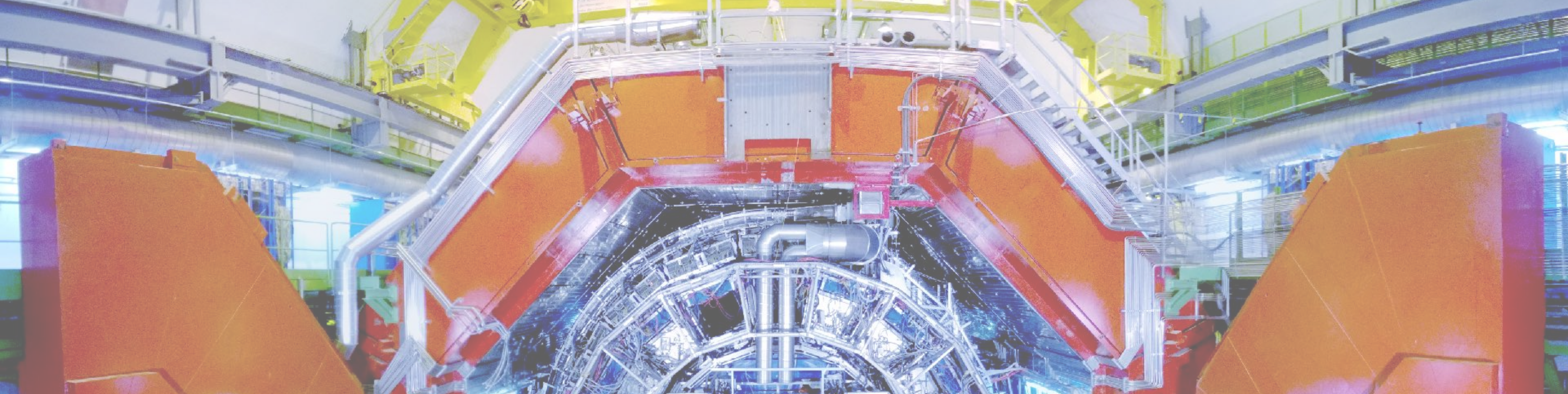
- **Aurora is a Mesos framework for long-running services and cron jobs**
- Aurora: multiple **users** and **environments** on top of Mesos
Allows to define priorities and preemption policies based on users and prod/devel environment
- Our challenge: **single entry point for future ALICE Run 3 operations** on a single datacenter
Users can launch tasks on a single datacenter: Aurora makes sure production use cases always have top priority and makes possible to seamlessly switch development to production



Developed and used by Twitter
with up to 10^5 containers:
production and development
seamlessly integrated on a
single datacenter

Conclusions and outlook

- **We are happy with Mesos and we are constantly working to do more with it**
- Many ALICE Run 2 central Offline services migrated to it
 - Reasonable effort: zero to little code to write, no modifications in current use cases
 - Tangible results: resources used better, high availability, stability
- Currently testing it on a larger scale for Run 3 operations
 - Aurora seems ideal to allow users to use the same production cluster
 - Aurora and DDS on Mesos will possibly deliver infrastructure reliability for critical Run 3 operations where Online and Offline will be together



Thank you!

