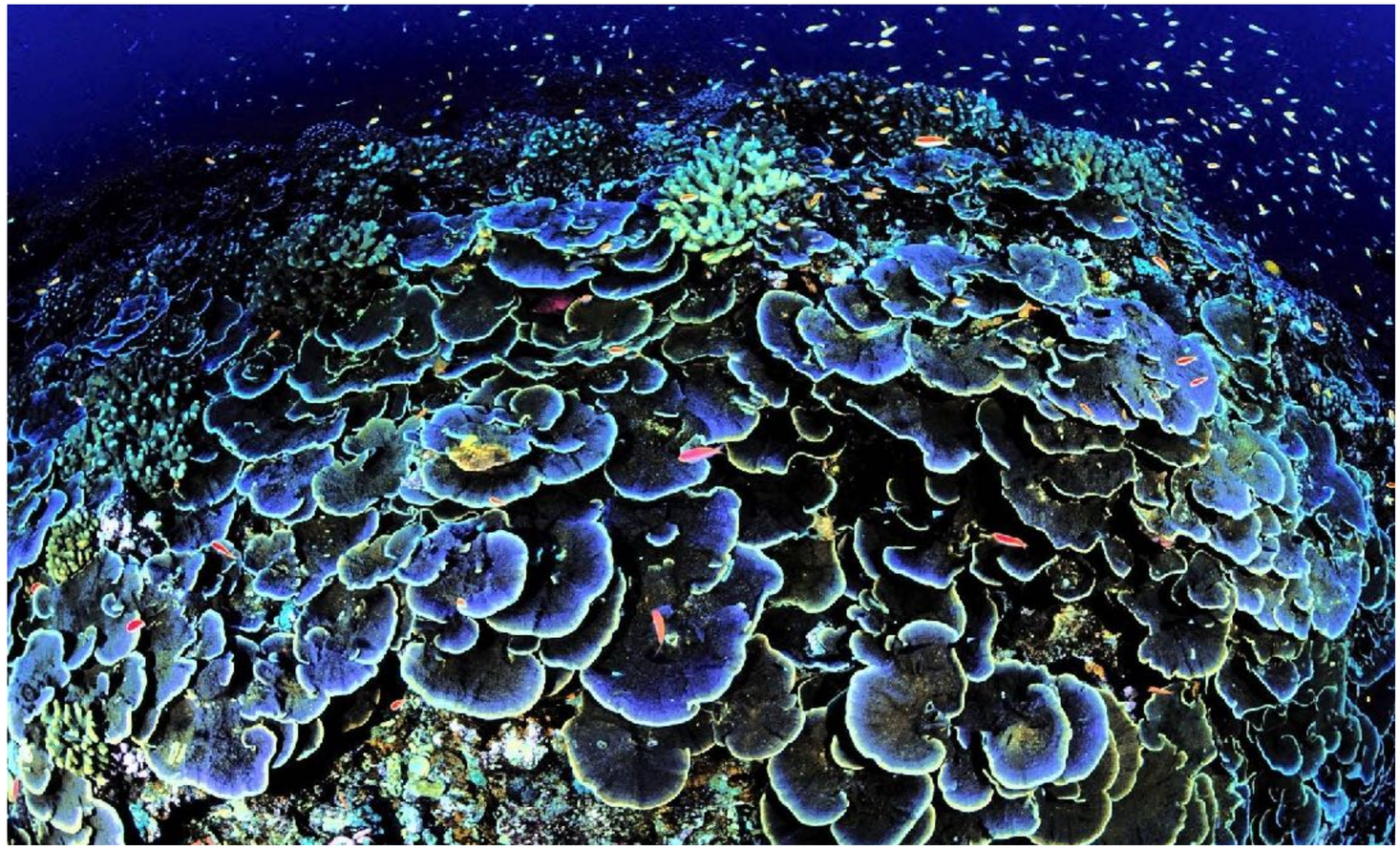


An introduction to the Mesos Framework Zoo

Benjamin Bannier

MESOSPHERE

Benjamin Bannier

benjamin.bannier@mesosphere.io

Software engineer at Mesosphere working on Mesos

Distributed columnar databases at ParStream (now Cisco ParStream)

Experimental High energy nuclear physics

MESOSPHERE

ParStream

STONY BROOK RHIC
relativistic heavy ion collider

The *p*layers

Tasks
perform interesting work





Frameworks
userlands interfaces



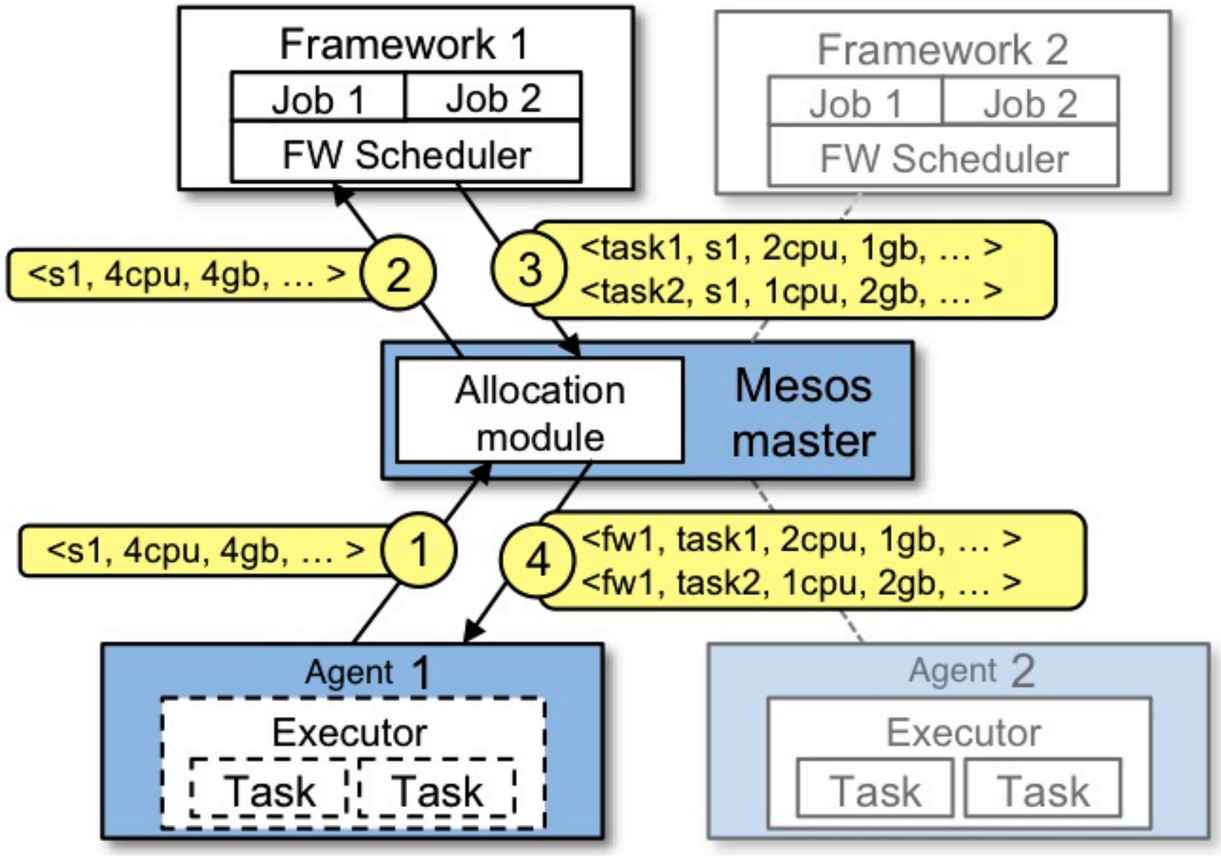

Mesos
abstracts resources

Responsibilities

Mesos is concerned with *tracking and scheduling offering of resources*.

Frameworks are concerned with *scheduling computations* on Mesos resources.

Resource allocation



Two-level scheduling approach separates responsibilities between Mesos and frameworks.

Gist

Tasks concerned with *domain-specific problems*.

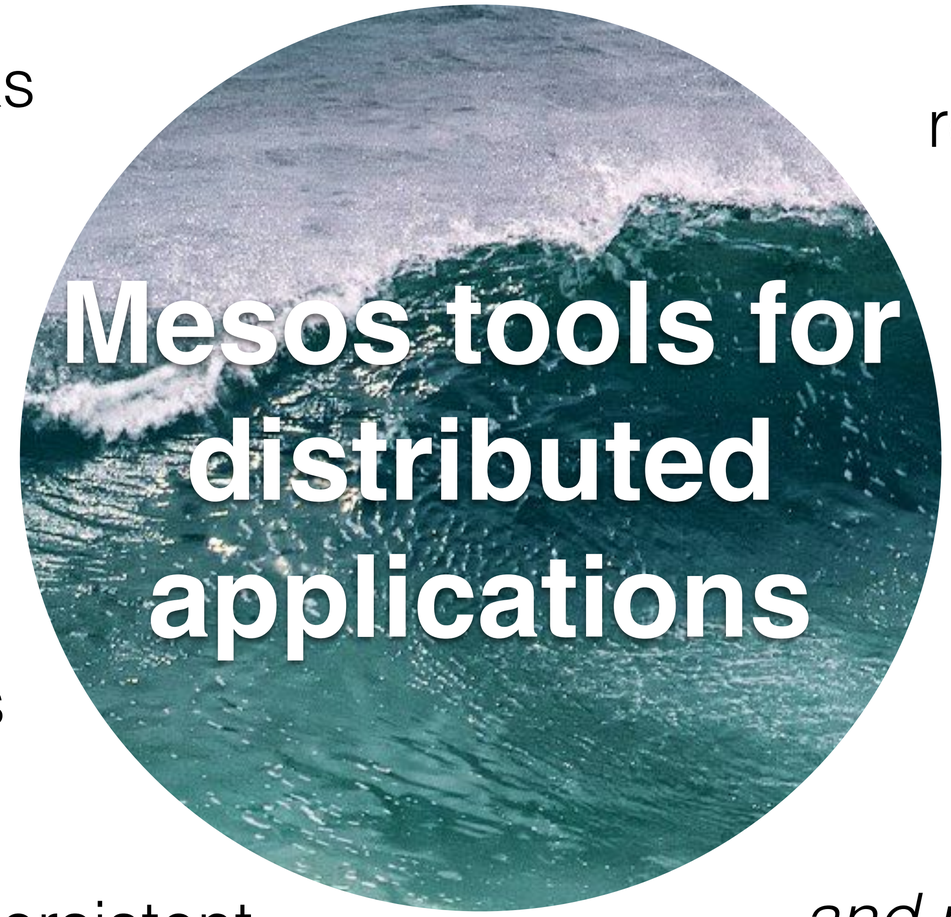
Framework abstract away *operational concerns of distributed environment*.

Mesos provides *low-level abstractions* of physical realities.

Workload isolation

Health checks

Resource reservations



Maintenance mode

Roles

Attributes/labels

Quotas

Persistent volumes

and more

Mesos is a *distributed systems kernel*.

Frameworks act as *userland interfaces* for distributed systems.

Frameworks

ETL
applications

Interactive
big data
analysis

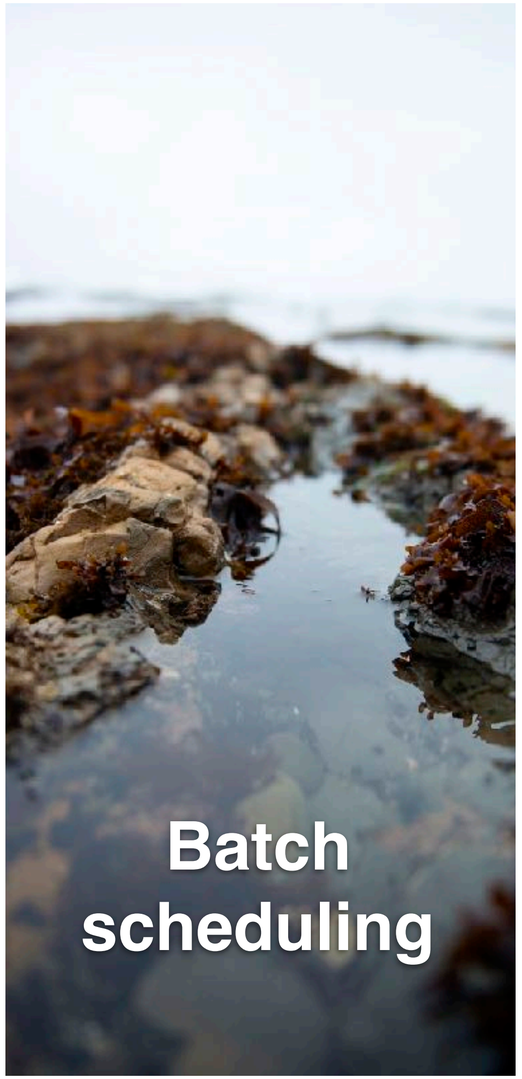
Scalable
applications

Interdependent
distributed
applications

Highly available
applications

more domain-specific


Big data
processing


Batch
scheduling


Long-running
services

Cray Chapel

Dpark

Exelixi

Hadoop

Hama

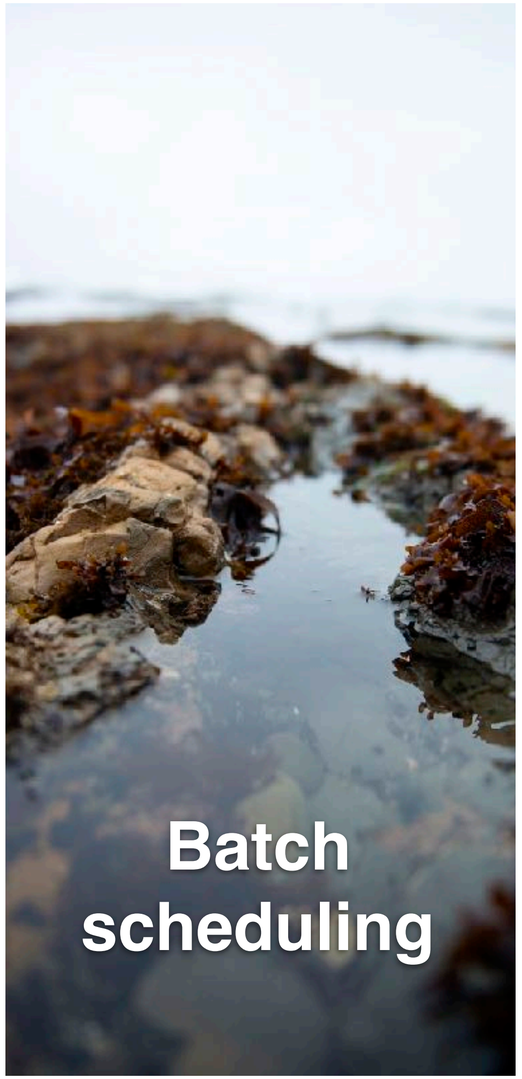
MPI

Spark

Storm

problem domain-specific adaptors to Mesos

distributed `cron`*s,*
pipelines



Chronos

Jenkins

JobServer

GoDocker

Cook

application scaling,
high availability

Aurora

Marathon

Singularity

SSSP

High-level frameworks can be use to *manage arbitrary workloads, including other frameworks*.

Meta-frameworks can implement *shells* for distributed systems.

Apache AURORA™

MARATHON


```
pkg_path = '/vagrant/hello_world.py'
import hashlib
with open(pkg_path, 'rb') as f:
  pkg_checksum = hashlib.md5(f.read()).hexdigest()

# copy hello_world.py into the local sandbox
install = Process(
  name = 'fetch_package',
  cmdline =
    'cp %s . && echo %s && chmod +x hello_world.py'
      % (pkg_path, pkg_checksum))

# run the script
hello_world = Process(
  name = 'hello_world',
  cmdline = 'python -u hello_world.py')

# describe the task
hello_world_task = SequentialTask(
  processes = [install, hello_world],
  resources = Resources(cpu = 1, ram = 1*MB, disk=8*MB))

jobs = [
  Service(cluster = 'devcluster',
          environment = 'devel',
          role = 'www-data',
          name = 'hello_world',
          task = hello_world_task)
]
```


```
{
  "id": "/product",
  "groups": [
    {
      "id": "/product/database",
      "apps": [
         { "id": "/product/mongo", ... },
         { "id": "/product/mysql", ... }
       ]
    },{
      "id": "/product/service",
      "dependencies": ["/product/database"],
      "apps": [
         { "id": "/product/rails-app", ... },
         { "id": "/product/play-app", ... }
      ]
    }
  ]
}
```

Maintenance mode in Aurora

Multitenancy

TODO: Add example of how groups of tasks are managed.

Also, aurora final jobs.

Scheduling: constraints & preemption

Deployment/ updates

Stateful applications

Integration into Service discovery

and more

***Health checks* and *readiness checks* for high-availability**

- Application deployments.
- Restart unresponsive applications.
- Rolling configuration updates.
- Scaling applications.

Monitors launched process, but also allows customization,

```
health = Process(
  name = 'health',
  cmdline = './health.py {{thermos.ports[health]}}'
)
```

`health.py` needs to respond to

- `GET /health`
- `POST /quitquitquit`
- `POST /abortabortabort`

Directly expose Mesos health checks in Marathon app definition,

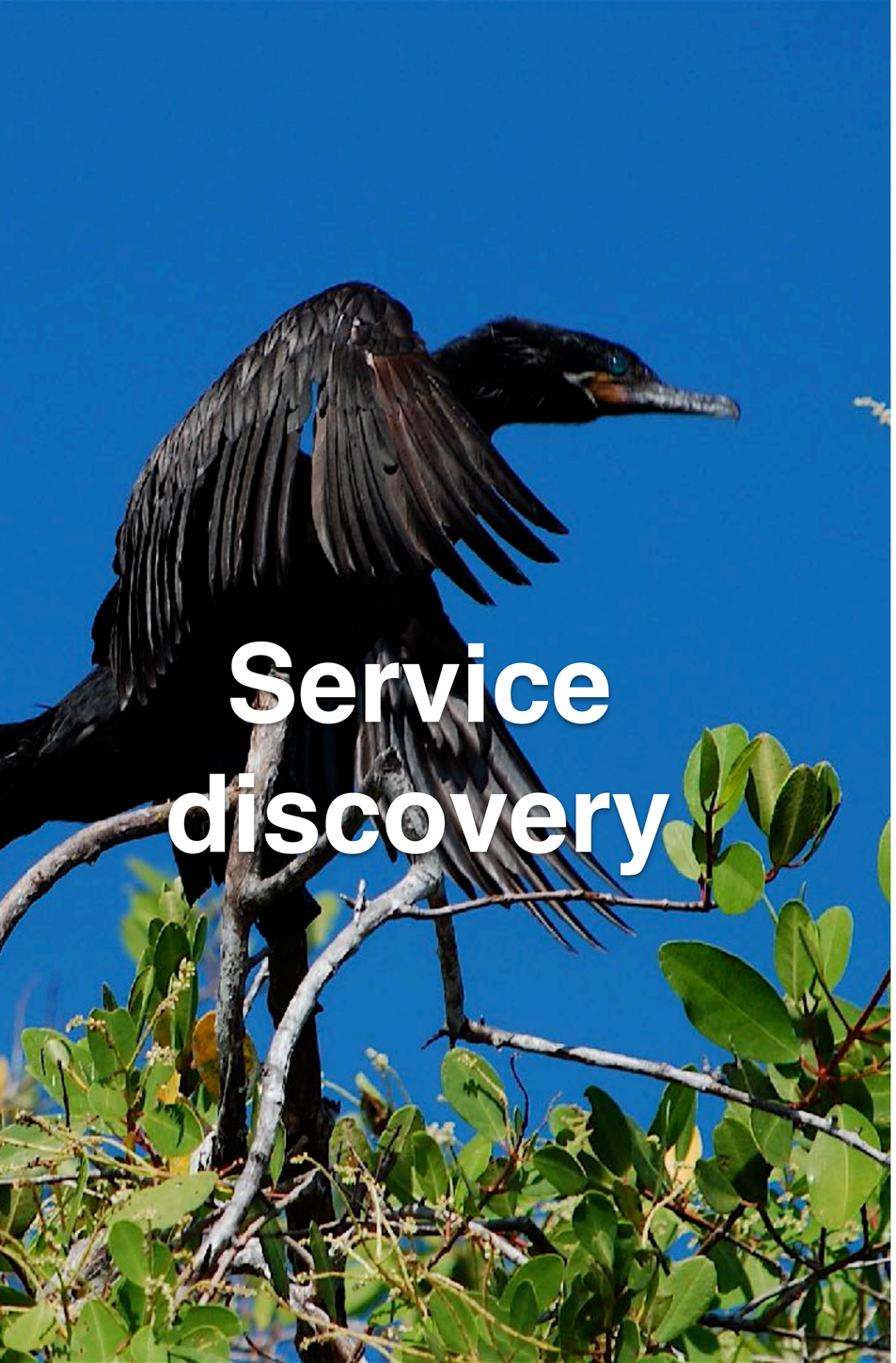
```
{
    "id": "rails_app",
    "cmd": "bundle exec rails server",
    "cpus": 0.1,
    "mem": 512,

    "container" : {
        // ...
    },

    "health": [
        {
            "protocol": "TCP",
            "portIndex": 0,
        }
    ]
}
```

Also possible are command and TCP checks.

Similarly defined are readiness checks.



Service discovery

Distributed applications need to be able to *find other applications*.

Frameworks can announce applications to external service discovery.

This can be integrated with framework-specific ACLs to control visibility.

Apache AURORA

```
app = Process(
    name = 'app',
    cmdline =
      "python -m SimpleHTTPServer {{thermos.ports[http]}}")

task = SequentialTask(
    process = [app],
    resources = Resources(cpu=0.5, ram=32*MB, disk=1*MB))

jobs = [Service(
    task = task,
    cluster = 'cluster',
    environment = 'production',
    role = 'www',
    name = 'app',
    announce = Announcer())]
```

Creates ZK nodes `/aurora/www/production/app/memberXYZ`:

```
{
   "status": "ALIVE",
   "additionalEndpoints": {
        "aurora": {
            "host": "192.168.33.7",
            "port": 31254
        },
        "http": {
            "host": "192.168.33.7",
            "port": 31254
        }
```

MARATHON

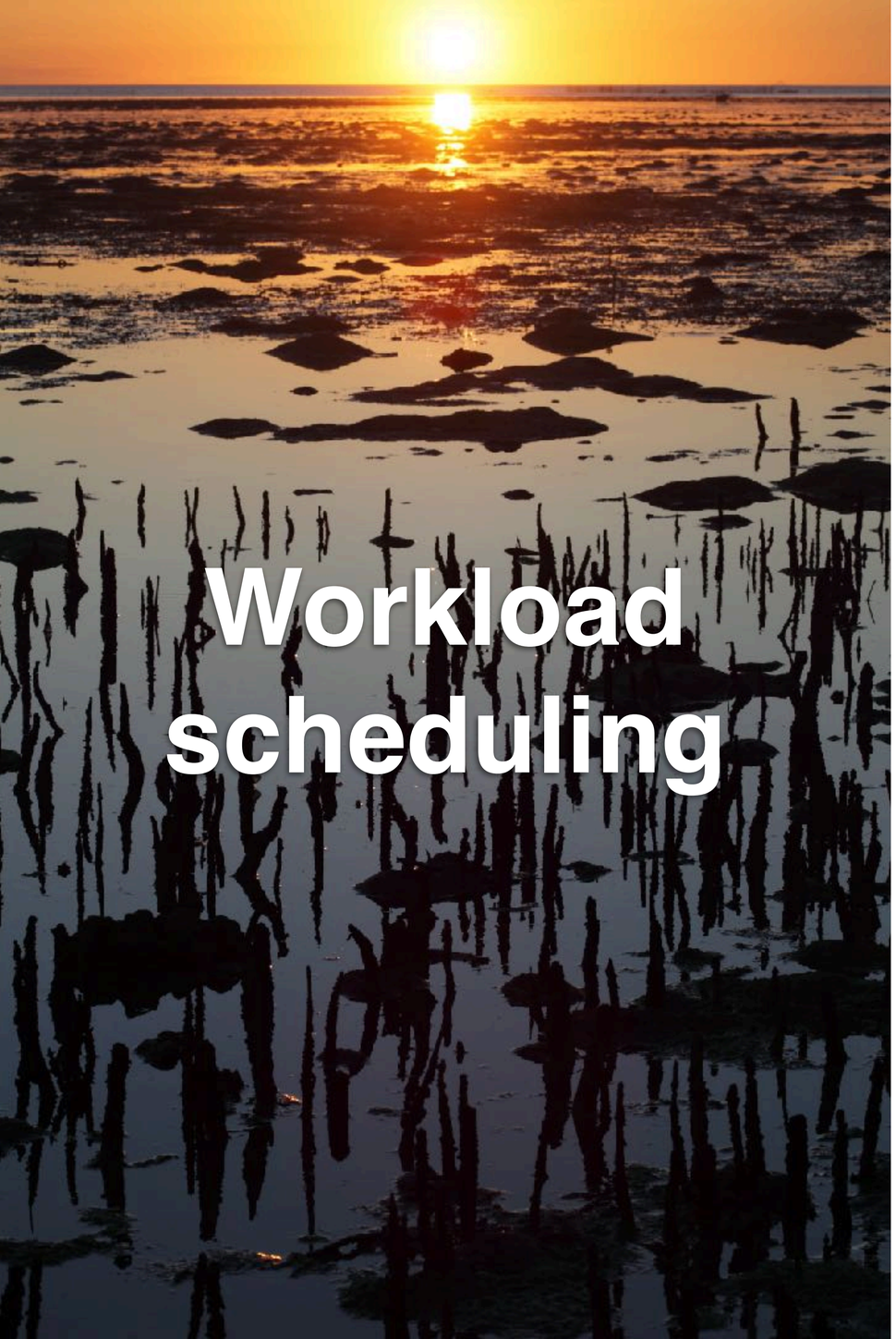
```
{
    "id": "app",
    "cmd": "python -m SimpleHTTPServer $PORT0",
    "cpus": 0.5,
    "mem": 32,

    // ”ports”: [0]
    "portDefinitions": [
        "port": 0,
        "protocol": "tcp",
        "name": "http"
    ]

}
```

Marathon publishes `DiscoveryInfo` for tasks to Mesos which can be queried there, e.g.,

```
{
  "visibility": "FRAMEWORK",
  "name": "http",
  "ports": {
    "ports": [
      {
        "number": 31422,
        "name": "http",
        "protocol": "tcp"
      }
    ]
  }
}
```



Workload scheduling

Task placement requirements

- run job *n*-times per machine, rack, cluster
- (not) run job on specific machine

Meta-frameworks provide DSLs to express complex scheduling constraints.

Apache AURORA™

```
rails = Process(
    name = 'rails_app',
    cmdline = 'bundle exec rails server'
)

rails_task = SimpleTask(
    processes = [rails],
    resources = Resources(cpu=1, ram=32*MB, disk=0*MB)
)

jobs = [
    Service(
        cluster = 'cluster',
        environment = 'production',
        role = 'www',
        name = 'rails',
        task = rails_task,
        instances = 10,
        constraints = {
            'type': 'public',
            'rack_id', 'limit:3',
            'host': 'limit:1'
        },
        container = ...
    )
]
```

MARATHON

```
{
  "id": "rails_app",
  "cmd": "bundle exec rails server",
  "cpus": 1,
  "mem": 32,
  "disk": 0,
  "instances": 10,
  "constraints": [
    [ "type", "CLUSTER", "public_node" ],
    [ "rack_id", "GROUP_BY" ],
    [ "rack_id", "MAX_PER", "3" ],
    [ "hostname", "UNIQUE"]
    // Also LIKE and UNLIKE.
  ],
  "container": {...},
}
```

Stateful applications

Big Data applications work with state

Application persists state in disk volume

- needs to be restarted on *same machine*
- other consumers should be executed *close to the data* as well.

Leverage *Mesos persistent volumes* and other *scheduling constraints*.

Apache AURORA™

```
$ mesos-agent --attributes='dedicated:db/data' ...
```

```
db = Process(
    name = 'db',
    cmdline = './db.sh /DATA'
)

db_task = SimpleTask(
    processes = [db],
    resources = Resources(cpu=1, ram=32*MB, disk=0*MB)
)

jobs = [
    Service(
        cluster = 'cluster',
        environment = 'testing',
        role = 'www',
        name = 'rails',
        task = db_task,
        instances = 10,
        constraints = { 'dedicated': 'db/DATA' }
    )
]
```

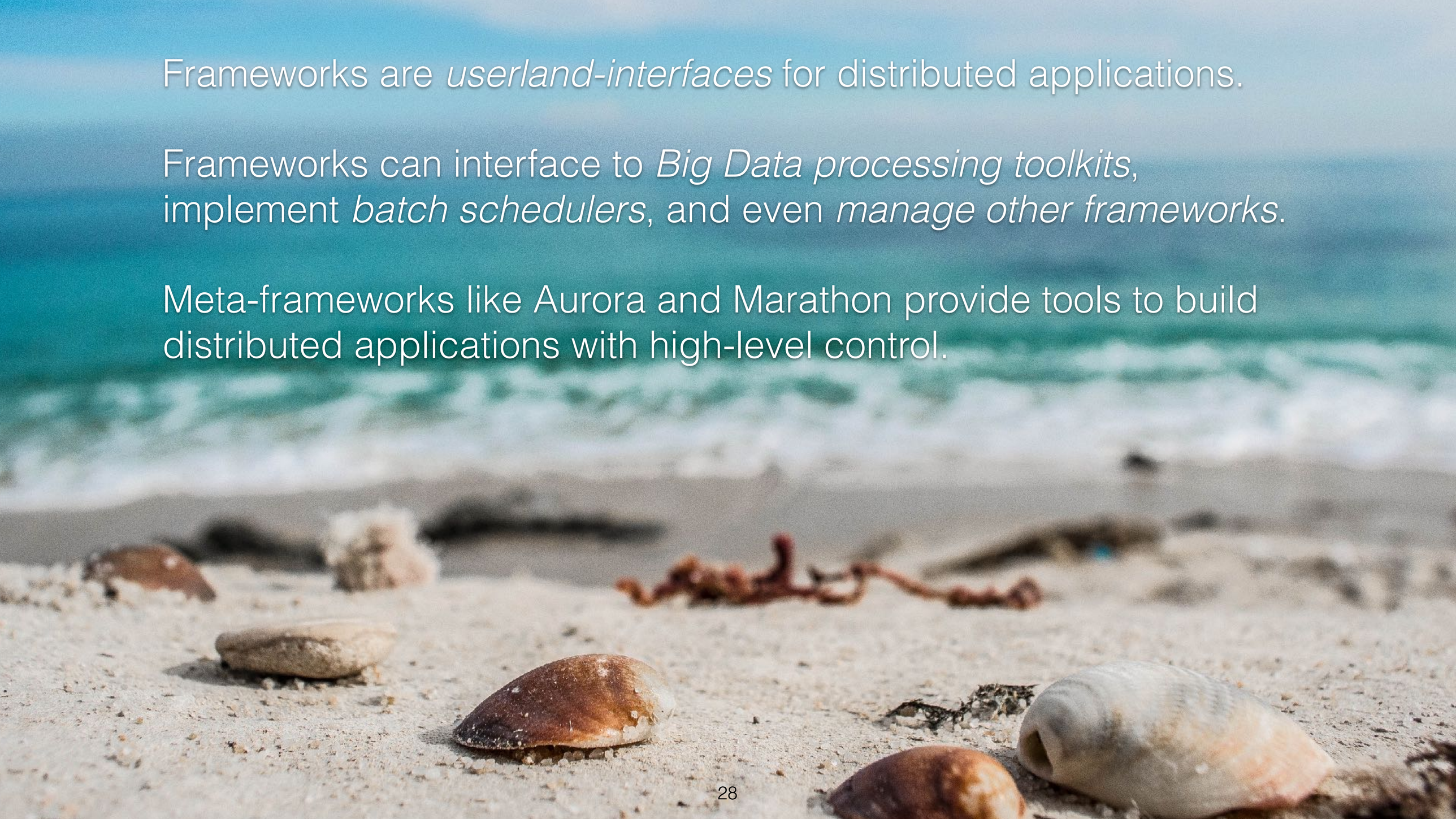
MARATHON

```
{
  "id": "foo",
  "cmd": "./db.sh ./data",
  "cpus": 1,
  "mem": 32,
  "instances": 1,
  "container": {
    "volumes": [
      {
        "containerPath": "data",
        "persistent": {
          "size": 128
        },
        "mode": "RW"
      }
    ],
    "type": "MESOS"
  }
}
```

Frameworks are *userland-interfaces* for distributed applications.

Frameworks can interface to *Big Data processing toolkits*, implement *batch schedulers*, and even *manage other frameworks*.

Meta-frameworks like Aurora and Marathon provide tools to build distributed applications with high-level control.




Thank you!

Backup

Mesos health check definition

```
message HealthCheck {
  enum Type {
    UNKNOWN = 0;
    COMMAND = 1;
    HTTP = 2;
    TCP = 3;
  }

  message HTTPCheckInfo {
    optional string scheme = 3;
    required uint32 port = 1;
    optional string path = 2;
    repeated uint32 statuses = 4;
  }

  message TCPCheckInfo {
    required uint32 port = 1;
  }

  optional double delay_seconds = 2 [default = 15.0];
  optional double interval_seconds = 3 [default = 10.0];
  optional double timeout_seconds = 4 [default = 20.0];
  optional uint32 consecutive_failures = 5 [default = 3];
  optional double grace_period_seconds = 6 [default = 10.0];

  optional Type type = 8;
  optional CommandInfo command = 7;
  optional HTTPCheckInfo http = 1;
  optional TCPCheckInfo tcp = 9;
}
```

Picture references

- Coral at Jarvis Island National Wildlife Refuge: USFWS/Jim Maragos, link
- Waves, Steve Austin, link
- Kelf forest, Andy Blackledge, link
- Great Barrier Reef, Kyle Taylor, link
- Tide pool, Hitchster, link
- White smoker, NOAA, link
- Pair of Common Lionfish (Pterois volitans), Rob, link
- Tropical Fish, Rich Brooks, link
- Mangroves growing in the coral rocks of the Guam coastline, David Burdick, link
- Fiddler crab, link
- Scout Key: Mangrove Ecosystem, Florida Keys, Phil's 1stPix, link
- Mangrove Sunset, Klaus Stiefel, link
- Mangrove island, Corn Farmer, link
- Sea shells, Mohamed Aymen Bettaieb, link