# Redwood – Towards clusterless supercomputing in the cloud

Presented by Philipp A. Witte

Microsoft Research for Industry (RFI)

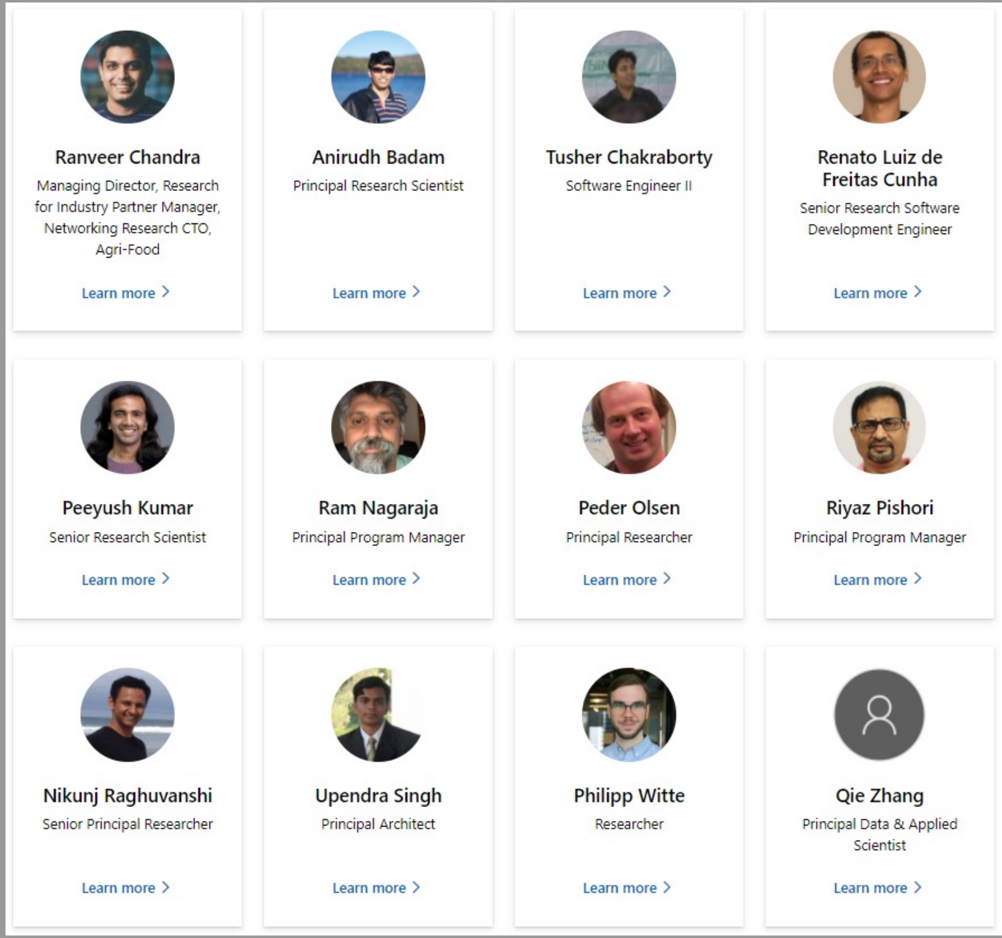
# Microsoft Research for Industry (RFI)

- Bridge the gap between **Industry R&D** and **Microsoft R&D**
- Bring together **MSR** + Academia
- Unsolved problems, Tackle on Azure
- Bring innovations to Microsoft Products
- Contribute to Industry Innovations

Microsoft

# Microsoft Research for Industry (RFI)

## Microsoft RFI Team




## Academic Collaborations (CCS)

**Professor Sally Benson**
Stanford University
$CO_2$ modeling

**Professor Felix J. Herrmann**
Georgia Tech
Seismic inversion,
Devito & AI

**Professor Gerard J. Gorman**
Imperial College London
Devito, HPC

## RFI CCS Interns 2021

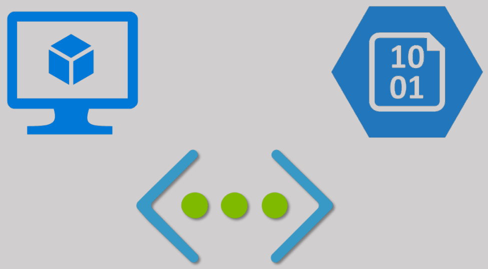
**Harpreet Kaur**
UT Austin
Seismic inversion & AI

**Tugrul Konuk**
Colorado School of Mines
Seismic inversion & AI

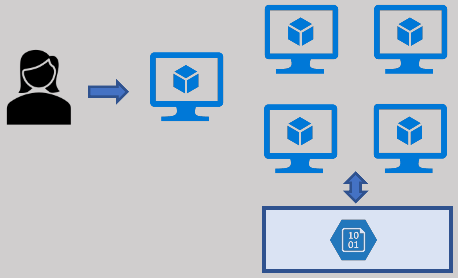
# Challenges of HPC in the cloud

- Users need to manage HPC infrastructure


- Scalable & resilient HPC: only as strong as weakest link




**HPC hardware**



**HPC architecture**


**HPC application**

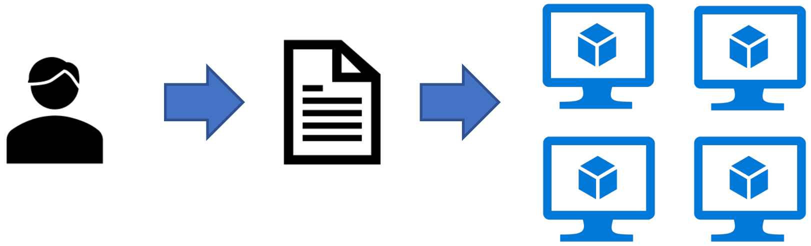
# Vision for HPC in the cloud

- Serverless computing

+ No infra management
+ Focus on application
+ Fast development
+ Usage-based billing

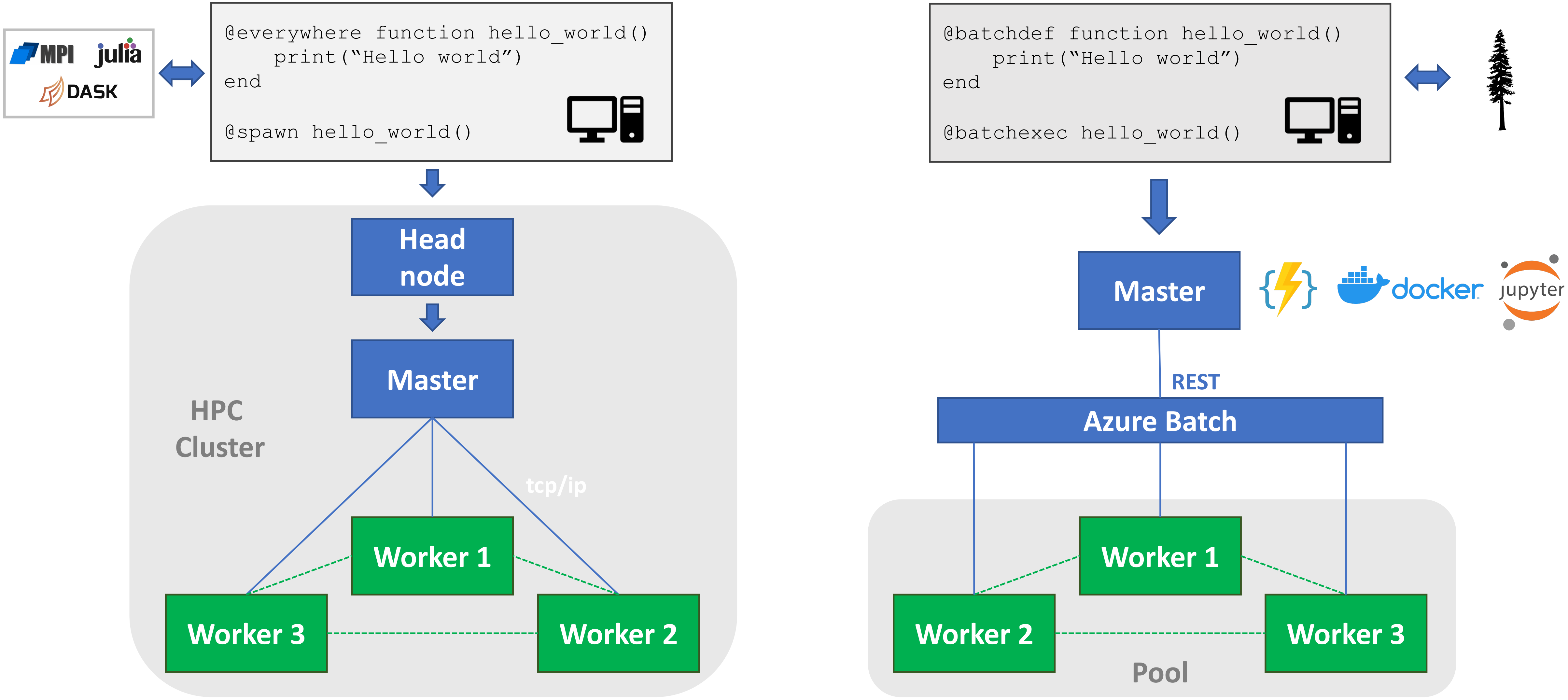
- Very limited hardware
- No orchestration
- Too limited in scope

- Clusterless HPC



+ All features of serverless
+ Run on any hardware
+ Orchestration + resilience
+ Not *just* an extension of serverless
+ Enable wide adoption of HPC

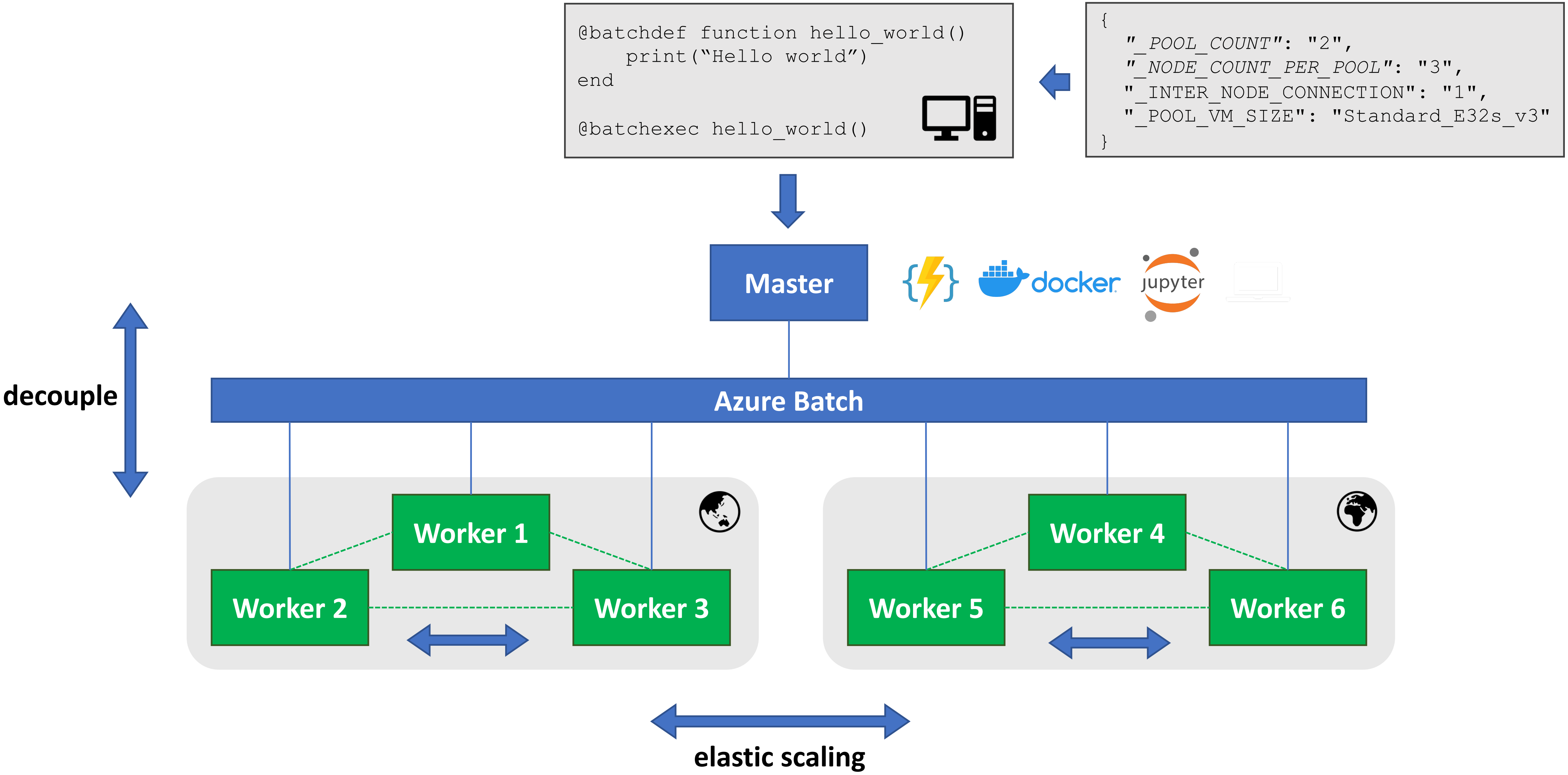
# Redwood: Towards clusterless HPC

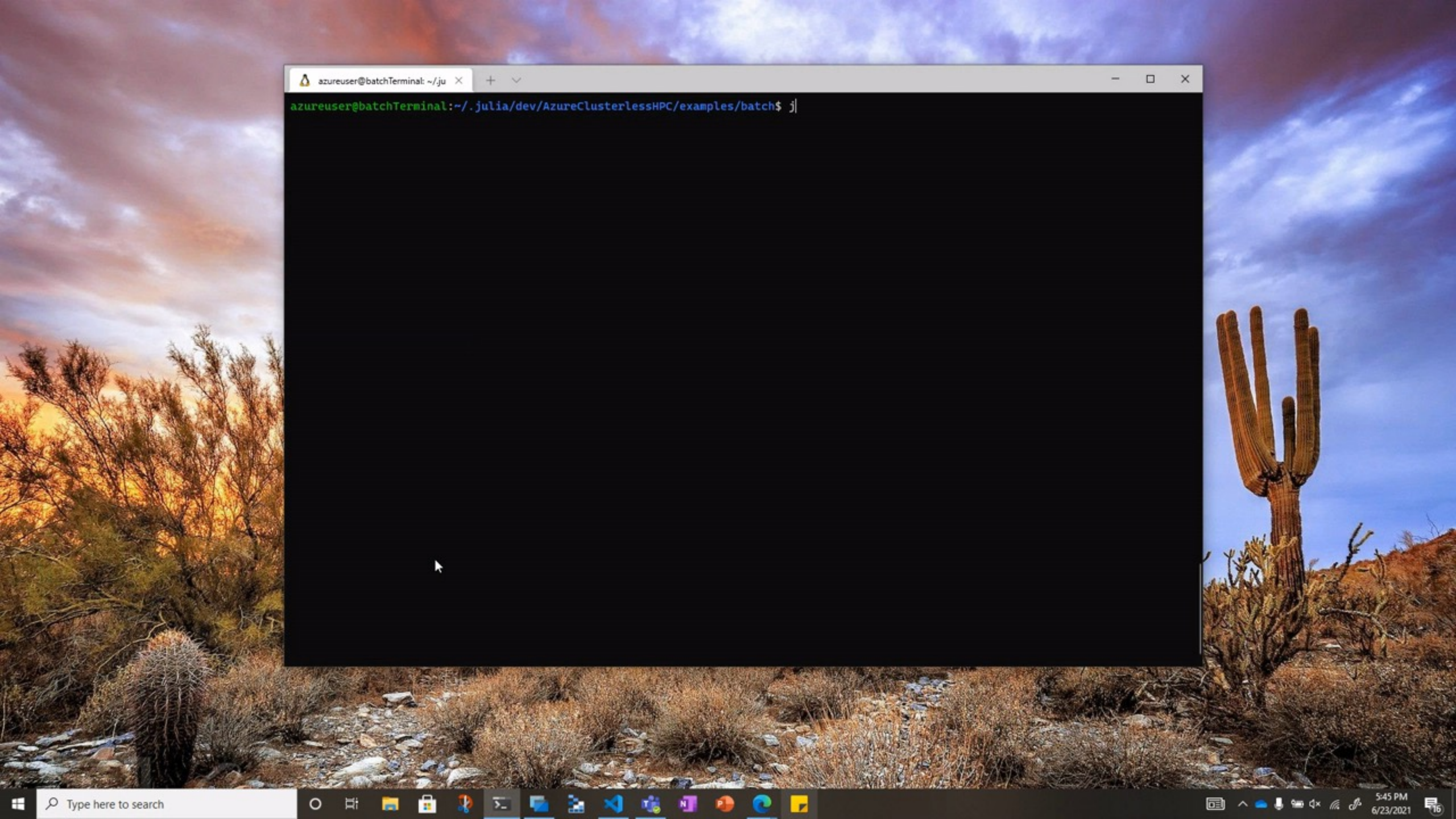



**Conventional HPC**

**Clusterless HPC with Redwood**

# Redwood: Towards clusterless HPC




azureuser@batchTerminal: ~/.ju
azureuser@batchTerminal:~/.julia/dev/AzureClusterlessHPC/examples/batch$ j
Type here to search
5:45 PM
6/23/2021

# Redwood – How does it work?

**Application**

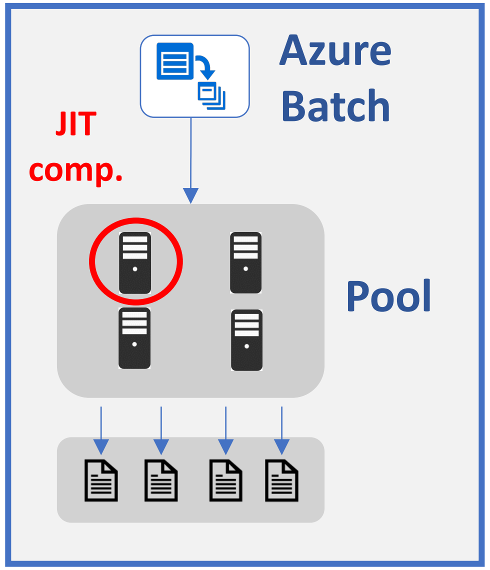
```julia
using Redwood

# Create pool
startup_script = "pool_startup.sh"
create_pool_and_resource_file(startup_script)

# Define function
@batchdef function hello_batch(my_id)
    print("Hello World from node $m_yid")
end

# Execute via Azure Batch
@batchexec pmap(my_id -> hello_world(my_id), 1:4)
```

```json
{
    "_POOL_VM_SIZE": "Standard_E8s_v3",
    "_POOL_COUNT": "3",
    "_NODE_COUNT_PER_POOL": "8",
    "_MPI_RUN": "0"
}
```

**Parameter JSON**

**Redwood**

- Closure
- Code gen.
- API calls

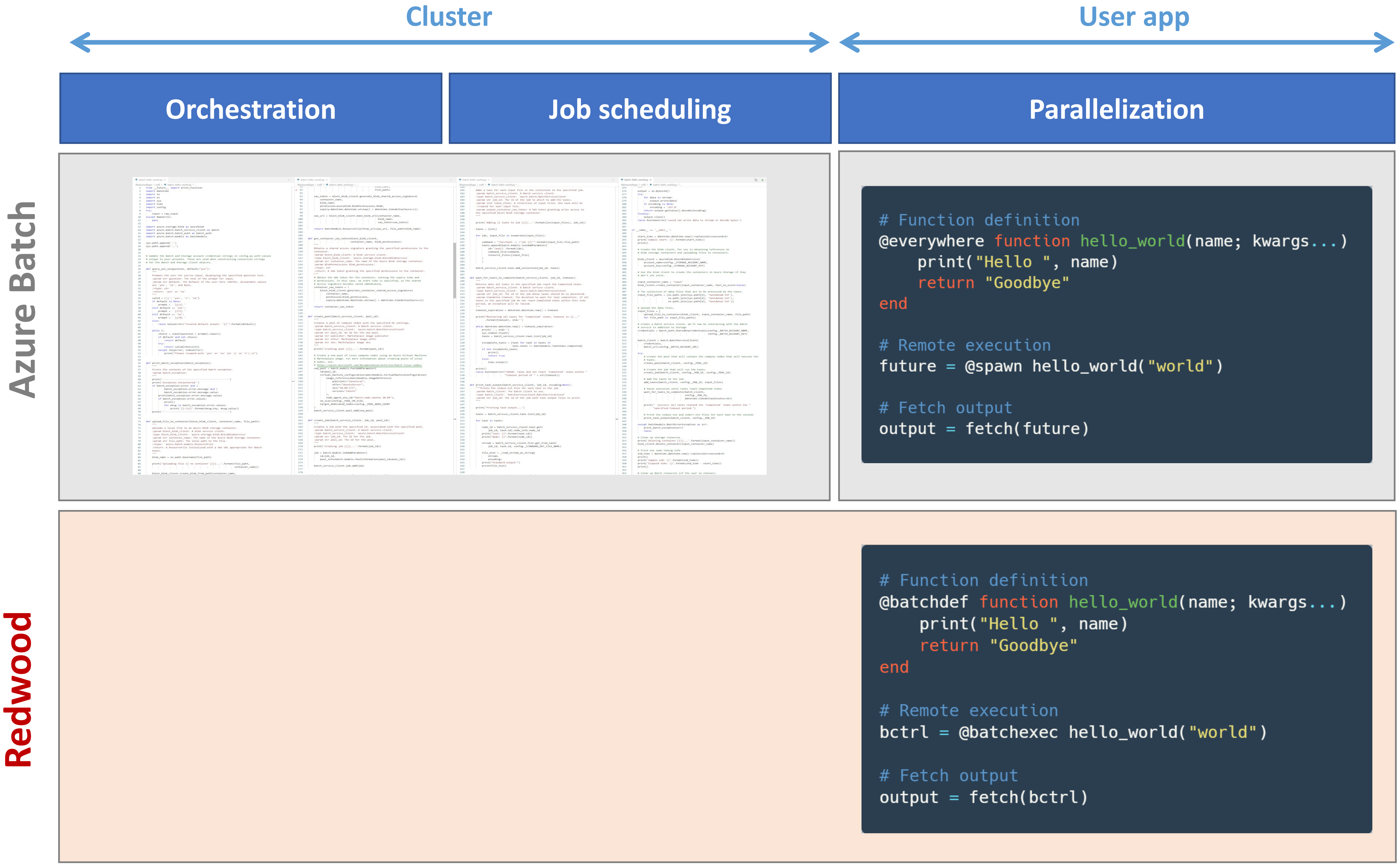



## Redwood leverages Azure Batch:

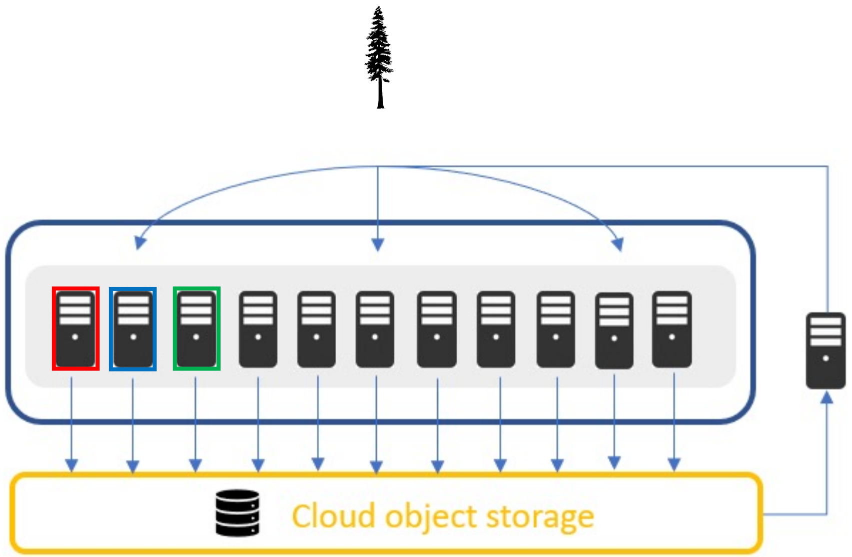
- Pool management
- Execute tagged functions
- I/O, fault tolerance

# Relationship to other services

| | Cluster | | User app |
|---|---|---|---|
| | **Orchestration** | **Job scheduling** | **Parallelization** |
| **Lift & shift** | CycleCloud | SLURM, PBS, HTCondor | MPI, Dask, Julia Distributed,… |
| **Kubernetes** | Kubernetes | | MPI, Dask, Julia Distributed,… |
| **Batch** | Azure Batch | | MPI, Dask (multi-node tasks) |
| **Spark** | Databricks, Kubernetes, Yarn, Mesos | Spark | |
| **Clusterless** | Redwood | | MPI, Dask, … |

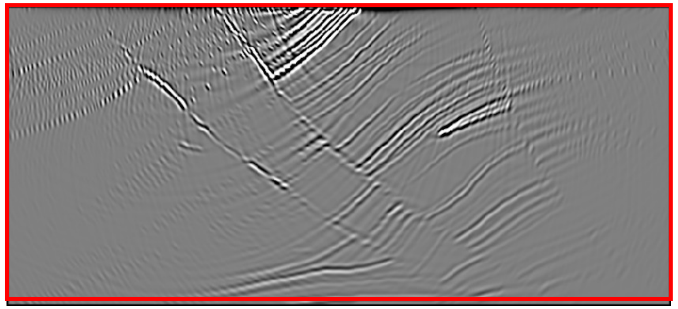

Cluster
User app
Orchestration
Job scheduling
Parallelization
Azure Batch
Redwood
# Function definition
@everywhere function hello_world(name; kwargs...)
    print("Hello ", name)
    return "Goodbye"
end
# Remote execution
future = @spawn hello_world("world")
# Fetch output
output = fetch(future)
# Function definition
@batchdef function hello_world(name; kwargs...)
    print("Hello ", name)
    return "Goodbye"
end
# Remote execution
bctrl = @batchexec hello_world("world")
# Fetch output
output = fetch(bctrl)

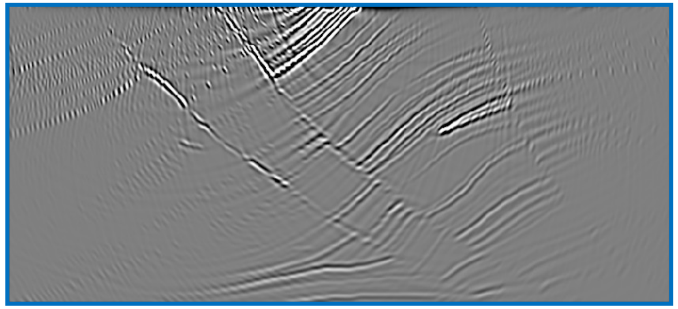
# Acceleration through abstractions

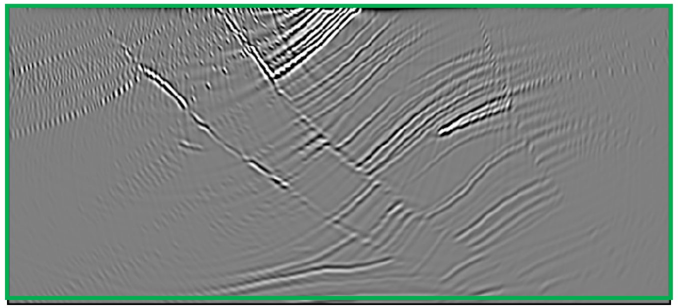



**Batch/MapReduce jobs**

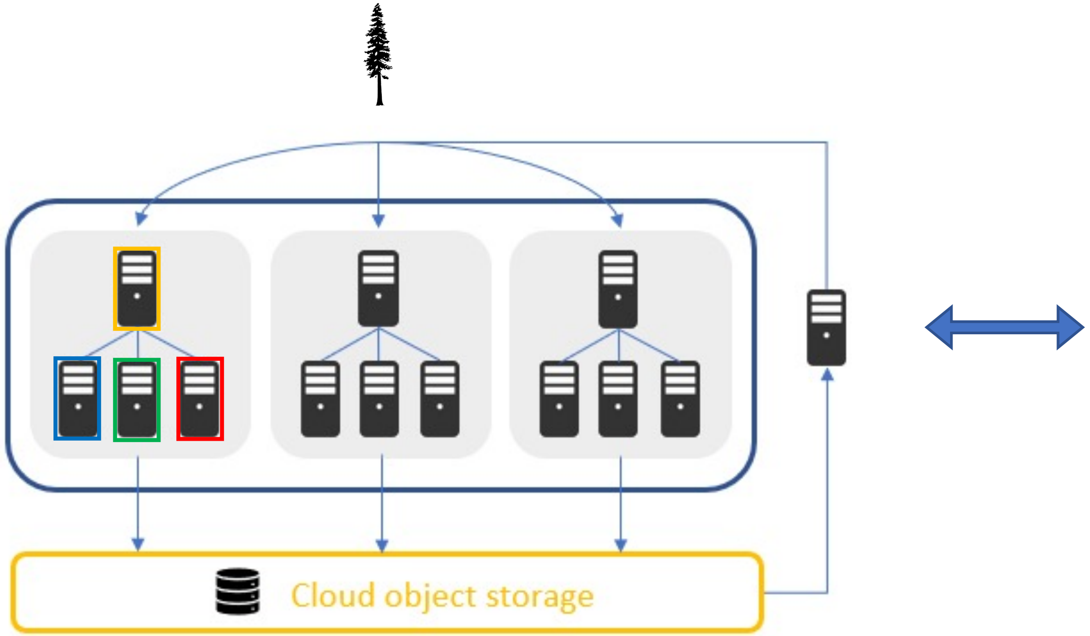
```
using Redwood

# Hello world
@batchdef function hello_world()
    print("Hello world")
end

# Execute tasks in parallel
bctrl = @batchexec hello_world()

# Collect and reduce
output = fetchreduce(bctrl; op=+)
```





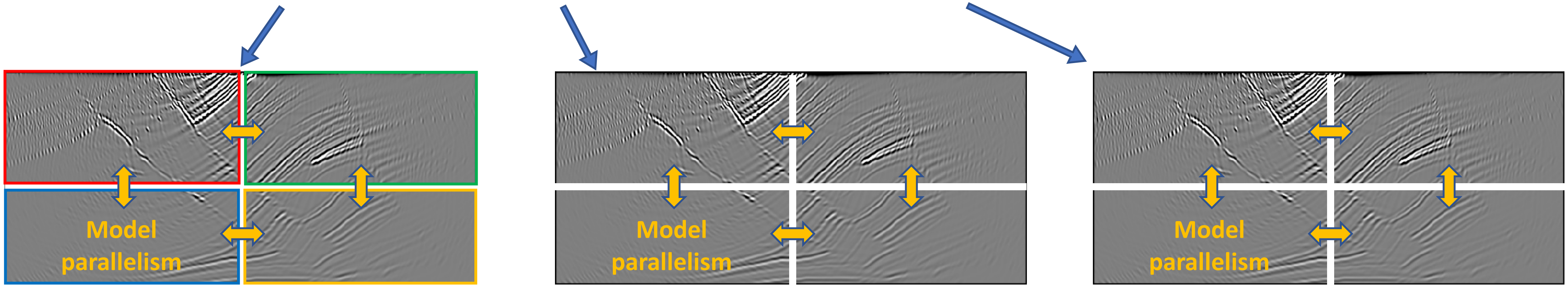


# Acceleration through abstractions

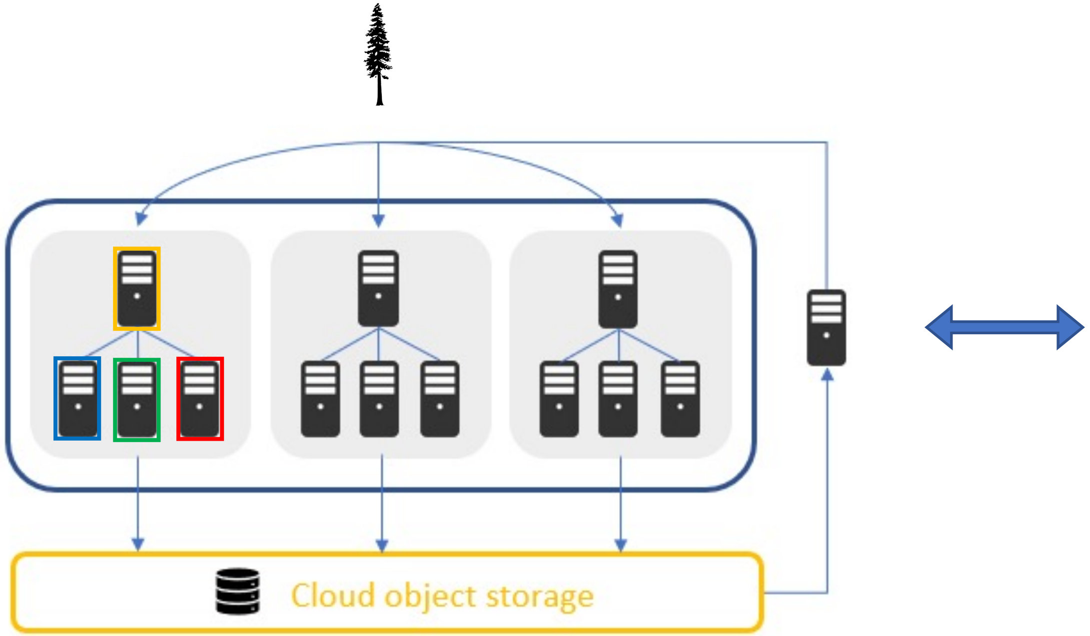



**Multi-MPI jobs**

```
using Redwood

@batchdef using MPI

# Define MPI function
@batchdef function hello_world()
    MPI.Init()
    comm = MPI.COMM_WORLD
    print("Hello world, I am rank $(MPI.Comm_rank(comm))\n")
    MPI.Barrier(comm)
end

# Execute three MPI jobs in parallel
bctrl = @batchexec pmap(i -> hello_world(), 1:3)
```

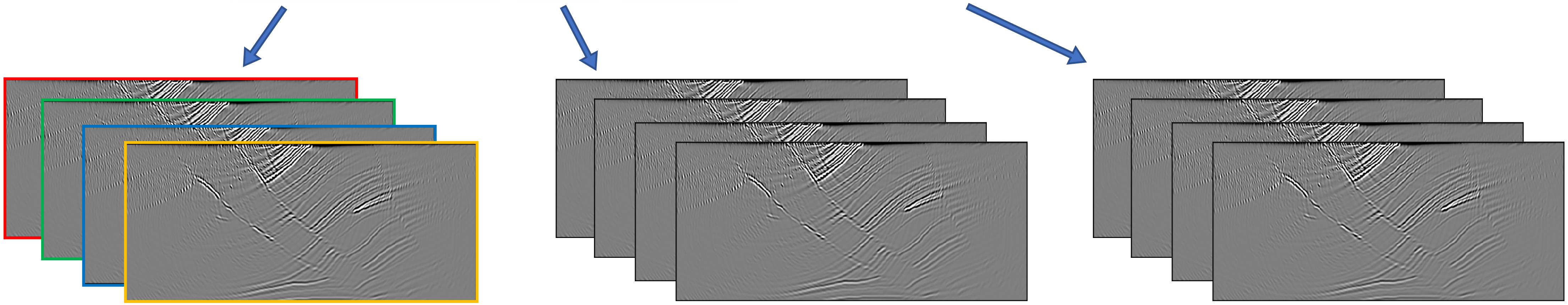


# Acceleration through abstractions




**Multi-pmap jobs**

```
using Redwood

@batchdef using Distributed

# Hello world
@batchdef hello_world() = print("Hello world")

# Define pmap function
@batchdef function say_hello()
    output = pmap(i -> hello_world(), 1:4)
    return output
end

# Execute three multi-task jobs in parallel
bctrl = @batchexec pmap(i -> say_hello(), 1:3)
```



# Redwood – Examples & applications

# (1) Seismic imaging (RTM)

```
# Packages
@everywhere using Distributed, Jets, JetPackWaveFD

# Functions
@everywhere function modelshot(isrc, sx, _v; kwargs...)
    # ...
end

# Remote execution
shot = epmap(isrc -> modelshot(isrc, sx, _v; kwargs...), 1:10)
```

```
# Packages
@batchdef using Distributed, Jets, JetPackWaveFD

# Functions
@batchdef function modelshot(isrc, sx, _v; kwargs...)
    # ...
end

# Remote execution
bctrl = @batchexec pmap(isrc -> modelshot(isrc, sx, _v; kwargs...), 1:10)
shot = fetch(bctrl)
```

## Redwood version of COFII examples

### Seismic modeling with COFII.jl and Redwood.jl

This examples demonstrates how to model seismic data using Chevron's `COFII` framework. Unlike the original example, which is based on Azure Scale Sets, we use `Redwood` to execute the computations as an Azure Batch job.

The first step is setting environment variables that point to our Azure Batch and Storage credentials, as well as to our parameter file. This file contains basic parameters of our batch pool, including the pool and job id, the VM type and the number of nodes in the pool.

In [1]:

```
# Set paths to credentials + parameters
ENV["CREDENTIALS"] = joinpath(pwd(), "credentials.json")
ENV["PARAMETERS"] = joinpath(pwd(), "parameters.json")

# Load package
using Redwood;
```

Next, we create the batch pool, which involves passing a startup script that specifies the necessary Julia packages that will installed on each node in the pool. In this example, this includes the COFII packages as specified here.

In [2]:

```
# Create pool
startup_script = "pool_startup_script_cofii.sh"
create_pool_and_resource_file(startup_script);
```

```
Pool 1 of 1 in canadacentral already exists.
```

We load all necessary packages on our local machine, as well as on the remote batch workers using the `@batchdef` macro:

In [3]:

```
@batchdef using Distributed, DistributedArrays, DistributedJets, Jets, WaveFD
@batchdef using JetPackWaveFD, DistributedOperations, Schedulers, Random;
```

Next, we read the Marmousi velocity model for which we will model the seismic data:

In [4]:

```
# Load model
v = read!("../../data/marmousi_vp_20m_176x851.bin", Array{Float32}(undef, 176, 851));
dz, dx = 20.0, 20.0
nz, nx = size(v)
@show dz, dx
@show nz, nx;
```

```
(dz, dx) = (20.0, 20.0)
(nz, nx) = (176, 851)
```

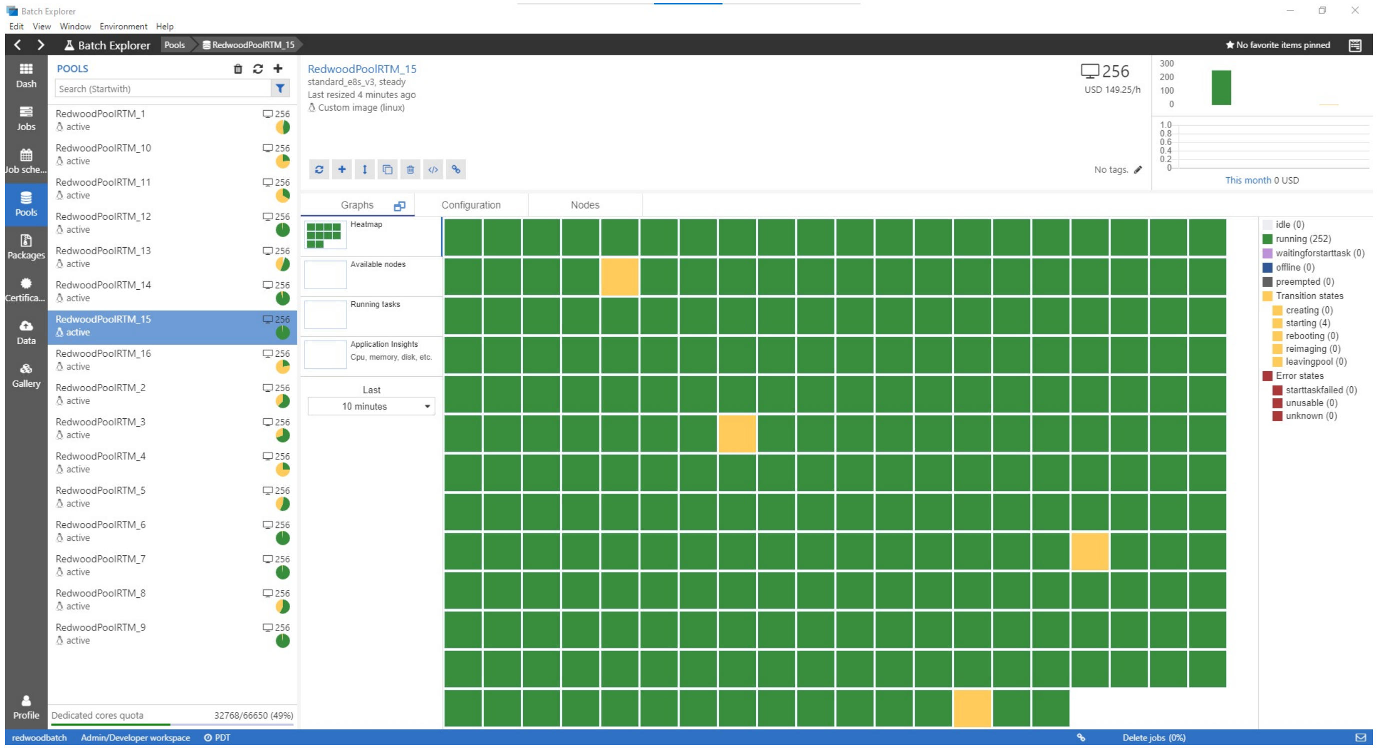
We specify the range of seismic source locations for which the data will be generated. In this example, we create 10 source locations:

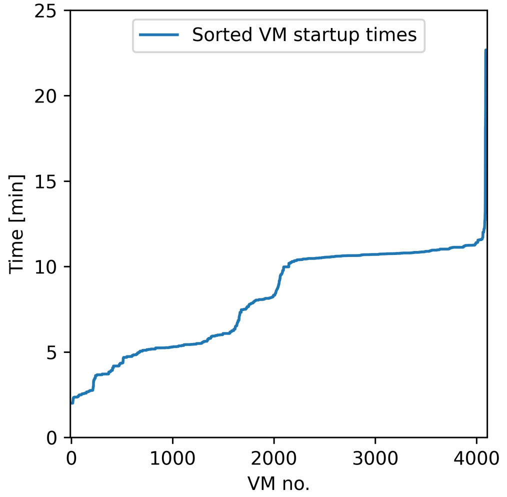
In [5]:

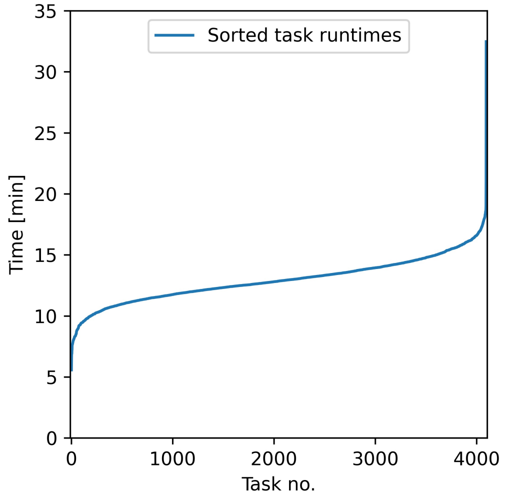
```
# Source range
sx = range(0, length=8, stop=(851-1)*20)
nshots = length(sx)
@show nshots;
```

```
nshots = 8
```

# (1) Seismic imaging (RTM)




## Redwood scalability

- Enable multiple batch pools and/or accounts
- Scheduling of jobs across many pools
- RTM using 16 x 256 = **4,096 VMs**

# (2) Scale across continents

```
In [2]:  # Create pool
         startup_script = "pool_startup_script_cofii.sh"
         create_pool_and_resource_file(startup_script);
```

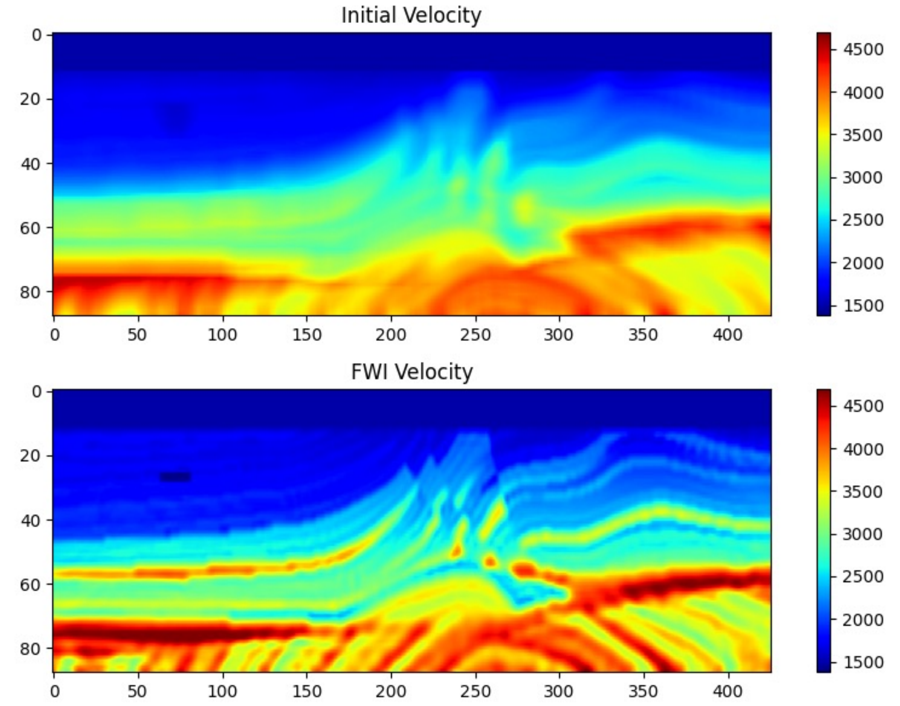
```
Created pool 1 of 10 in australiacentral with 2 nodes.
Created pool 2 of 10 in brazilsouth with 2 nodes.
Created pool 3 of 10 in eastasia with 2 nodes.
Created pool 4 of 10 in eastus2 with 2 nodes.
Created pool 5 of 10 in francecentral with 2 nodes.
Created pool 6 of 10 in koreasouth with 2 nodes.
Created pool 7 of 10 in southafricanorth with 2 nodes.
Created pool 8 of 10 in southeastasia with 2 nodes.
Created pool 9 of 10 in uaenorth with 2 nodes.
Created pool 10 of 10 in westus2 with 2 nodes.
```

```
topt = @elapsed begin
    result = optimize(Optim.only_fg!(_fg!), v2, solver,
        Optim.Options(
            iterations = niter,
            callback = mycallback
        )
    )
end
```

**Quasi Newton optimization algorithm from Optim.jl**

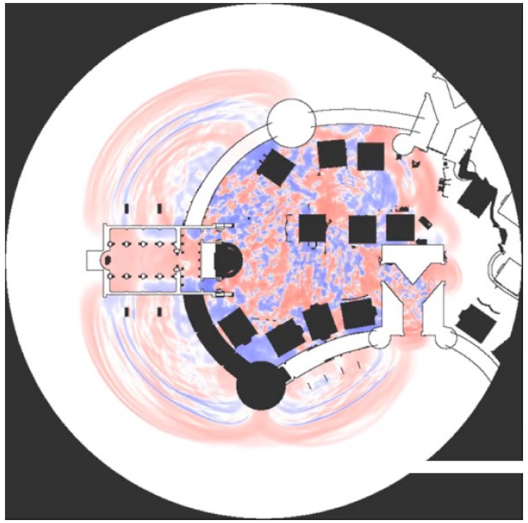
**"Global FWI"**

- Globally distributed pools
- Run FWI on 6 continents
- Harvest resources across globe

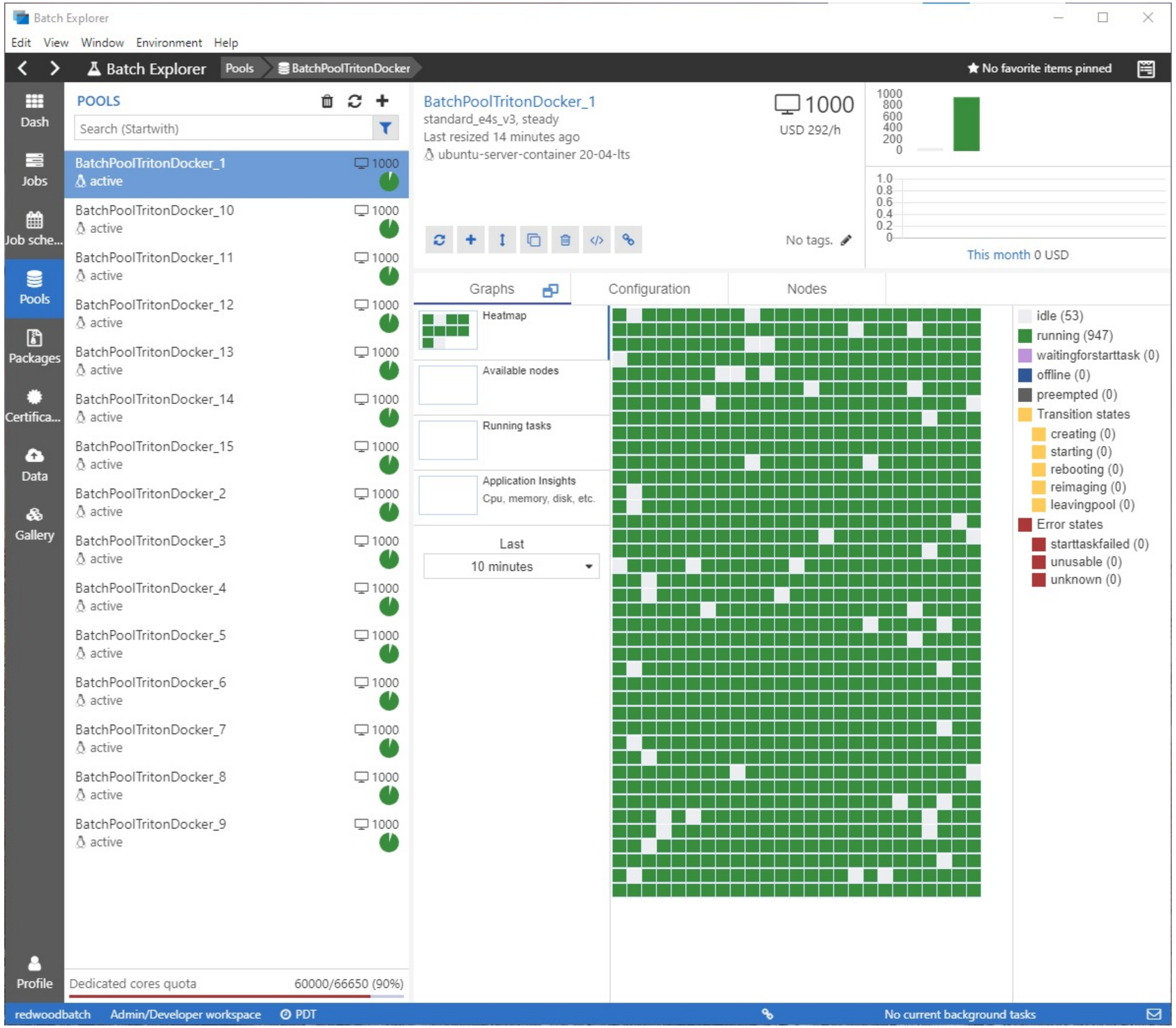


# (3) Sound simulation

- Project Triton from MSR
- Sound simulations for games
- Simulate **14,197** probes in parallel

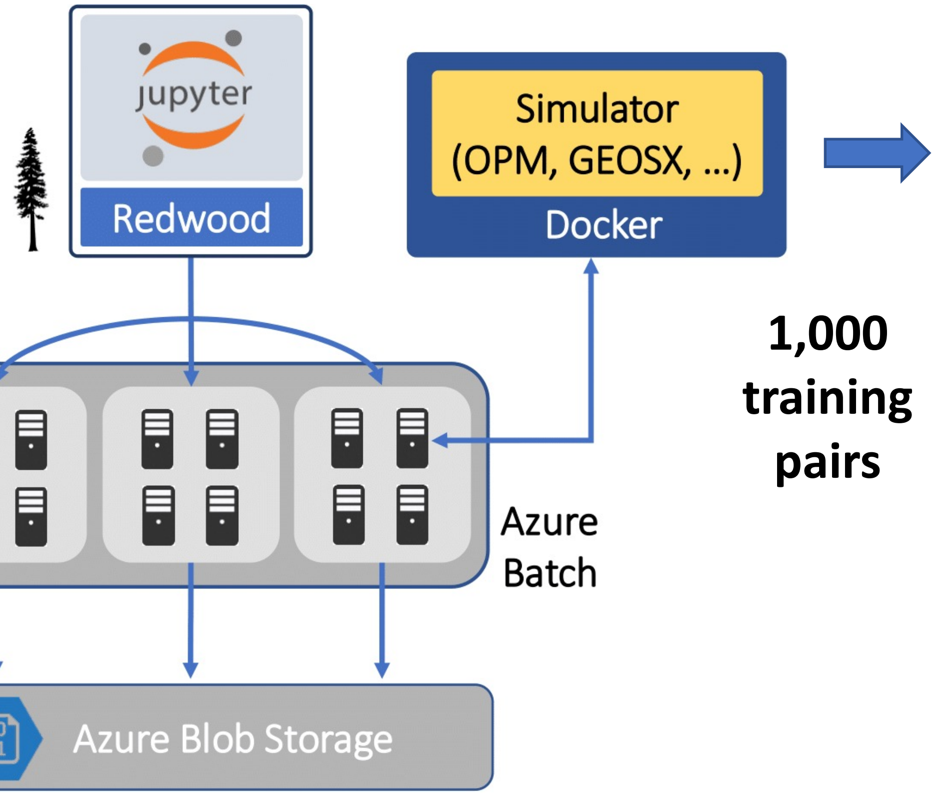


| Tasks | Desktop | Single batch pool | Redwood |
|---|---|---|---|
| 14,197 | 147 days | 3.5 hours | 0.25 hours |

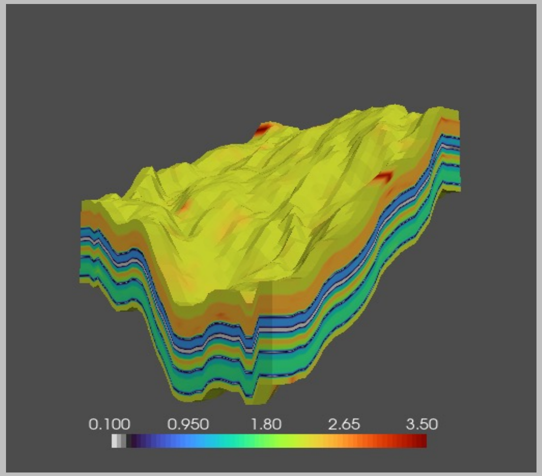



**15,000 node cluster**

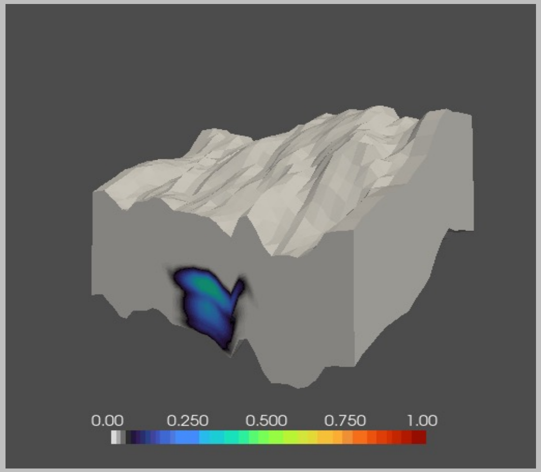
# (4) Reservoir simulation



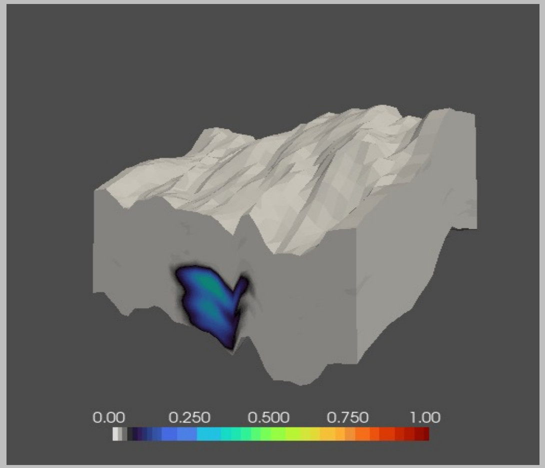

1,000 training pairs




Input X
(Permeability)




Output Y
(Saturation history)




Network prediction
(SNR 12.86)

# Open-source repository

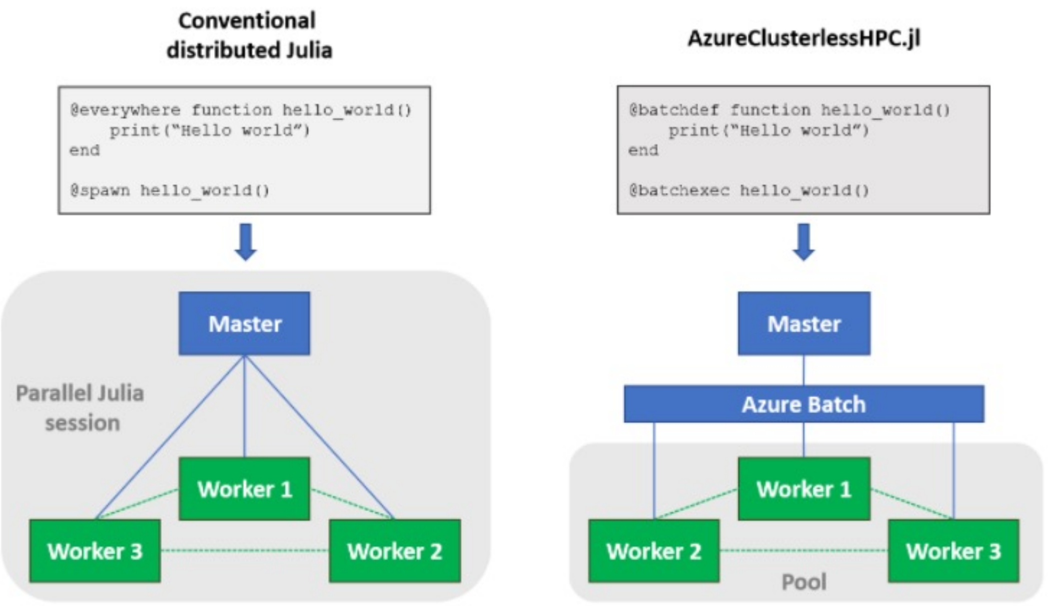
README.md

docs stable | CI passing

## AzureClusterlessHPC.jl - Simplified distributed computing

### Overview

**AzureClusterlessHPC.jl** is a package for simplified parallal computing on Azure. AzureClusterlessHPC.jl borrows the syntax of Julia's Distributed Programming package to easily execute parallel Julia workloads in the cloud using Azure Batch. Instead of a parallel Julia session, users create one or multiple worker pools and remotely execute code on them.




https://github.com/microsoft/AzureClusterlessHPC.jl