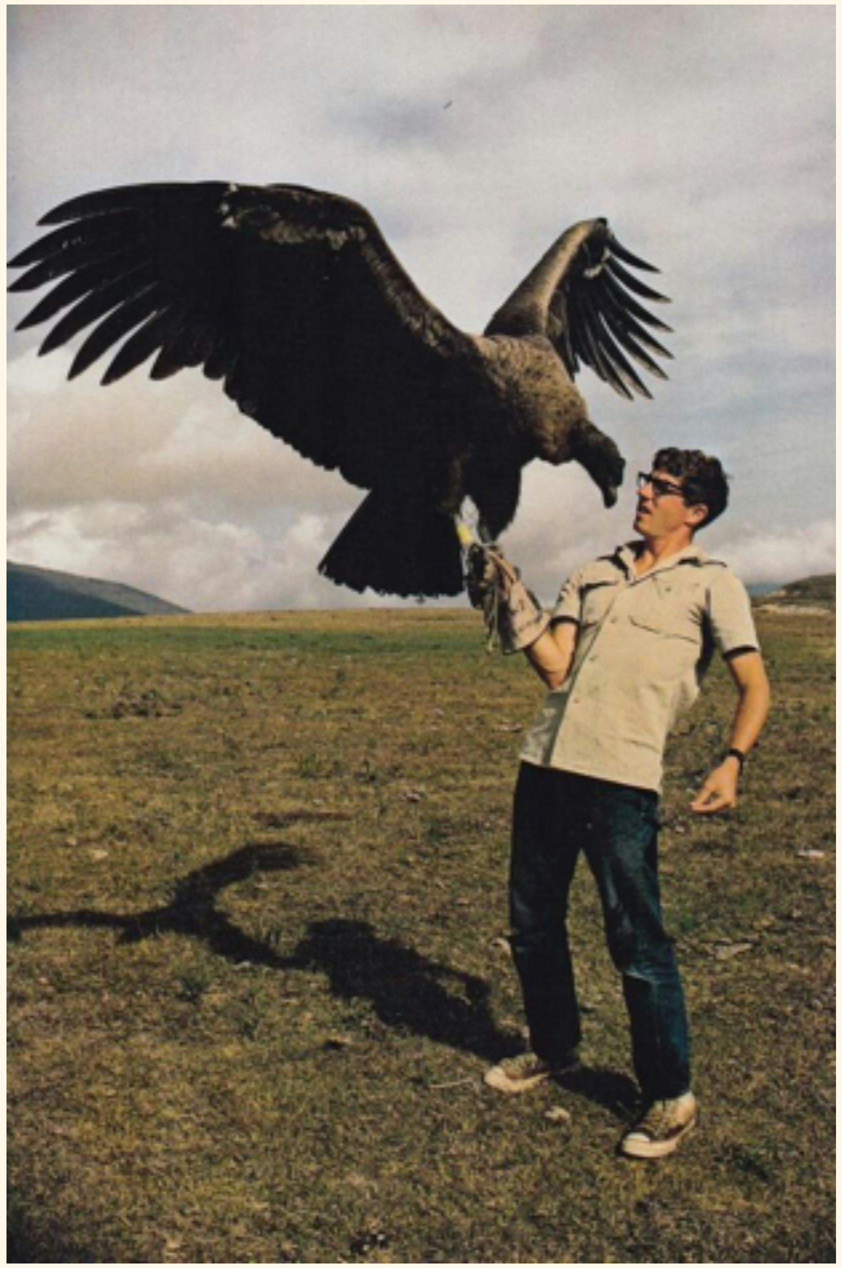
HTCondor & Cloud stuff…



Game plan for today:

Introduction to Condor

Condor use patterns

StarCluster: On-demand clusters, in “The Cloud”!

Course logistics update

Thus far, we've largely talked about Hadoop…

M/R and Hadoop ecosystem is great, but:

… what if you've already got programs written?

… what if your data can't go on HDFS?

… what if your cluster doesn't include Hadoop?

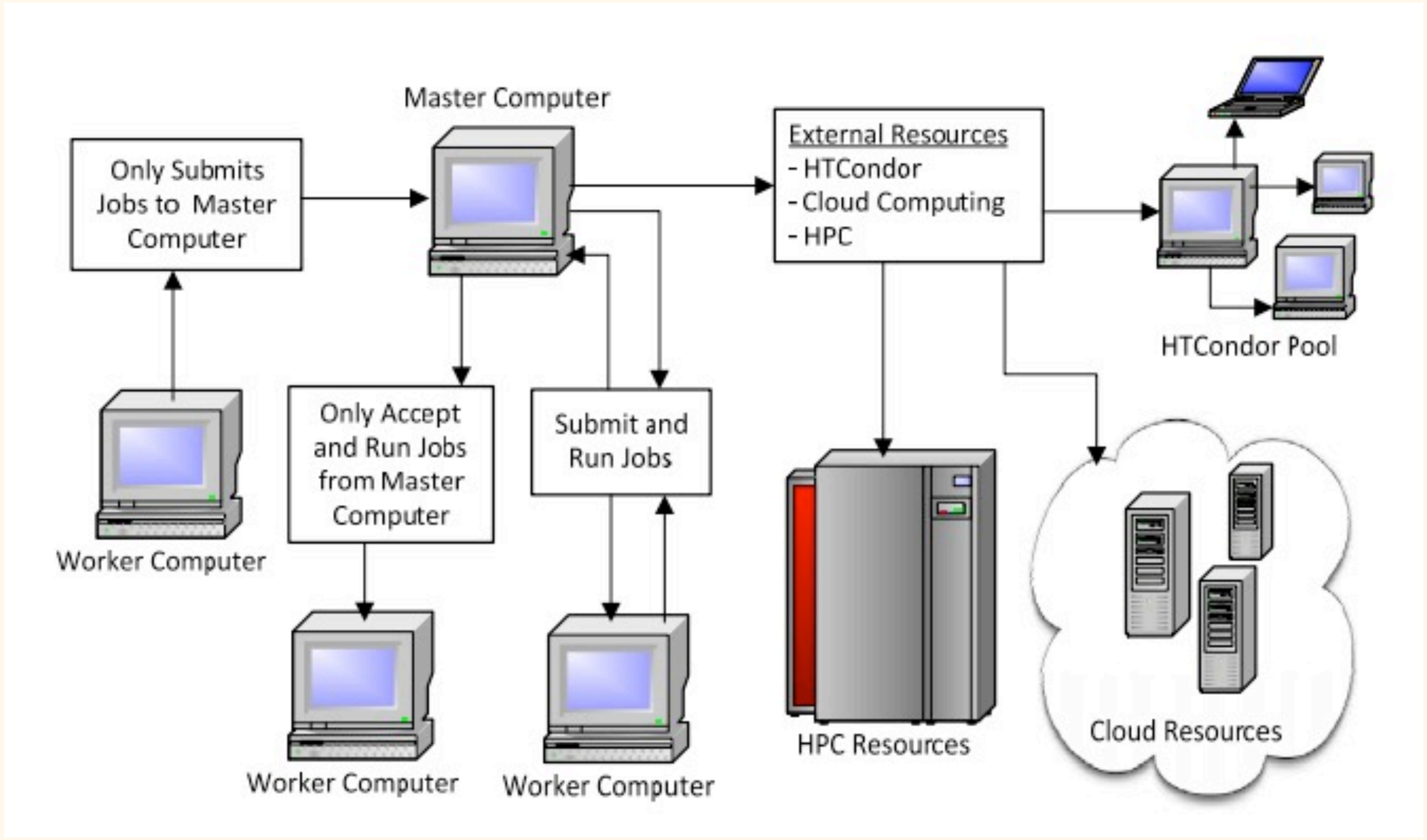
There are other paradigms!

HTCondor is an example of a “queue”-based clustering system.

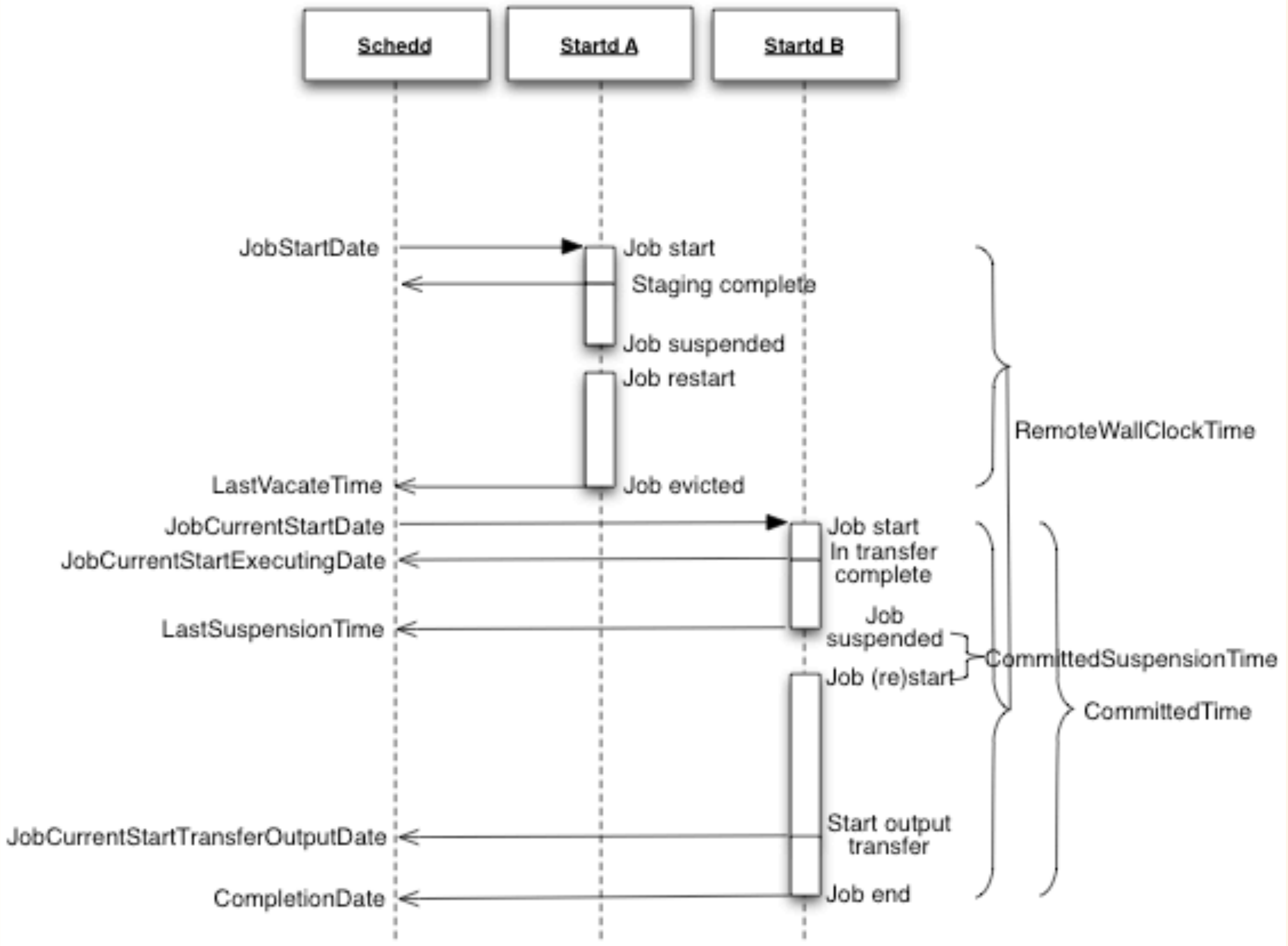
Users submit jobs, which are processed in order by worker machines.

Jobs specify what to do, where to do it, etc.

Others include SGE, OpenGrid, etc.



http://www.hpcwire.com/2013/05/30/running_stochastic_models_on_htcondor/


Schedd
Startd A
Startd B
JobStartDate
Job start
Staging complete
Job suspended
Job restart
LastVacateTime
Job evicted
JobCurrentStartDate
Job start
JobCurrentStartExecutingDate
In transfer complete
LastSuspensionTime
Job suspended
Job (re)start
JobCurrentStartTransferOutputDate
Start output transfer
CompletionDate
Job end
RemoteWallClockTime
CommittedSuspensionTime
CommittedTime

HTCondor was designed to work in a very heterogeneous environment…

… and so provides lots of flexibility in its configuration!

Condor is built around “slots” and “ads”

Compute nodes provide some number of “slots”...

... and advertise those slots capabilities:

CPU type, memory size, other capabilities, etc.

Jobs then submit requests specifying their runtime requirements: “> 1 gig of RAM”, “2 CPUs”, etc.

Condor figures out which slot should service which jobs!

To run a condor job, first write a jobfile…

```
executable = run_monkeys.py
universe = vanilla
arguments = -brachiate=True -num_bananas=$(Process)

output = /g/steven/monkeys/out/$(Process).out
error = /g/steven/monkeys/err/$(Process).err
initialdir = /g/steven/monkey_data/

queue 10
```

To run a condor job, first write a jobfile...

```
executable = run_monkeys.py
universe = vanilla
arguments = -brachiate=True -num_bananas=$(Process)

output = /g/steven/monkeys/out/$(Process).out
error = /g/steven/monkeys/err/$(Process).err
initialdir = /g/steven/monkey_data/

queue 10
```

... and submit it:

```
$ condor_submit monkey_job.condor
Submitting job(s)..........
10 job(s) submitted to cluster 4.
```

Jobfiles can contain all sorts of other information:

```
request_memory = 1 GB

Requirements = (Arch == "INTEL" && OpSys == "LINUX") || \
               (Arch == "X86_64" && OpSys =="LINUX")

Requirements = (OpSysName == "Ubuntu")

request_GPUs = 1
```

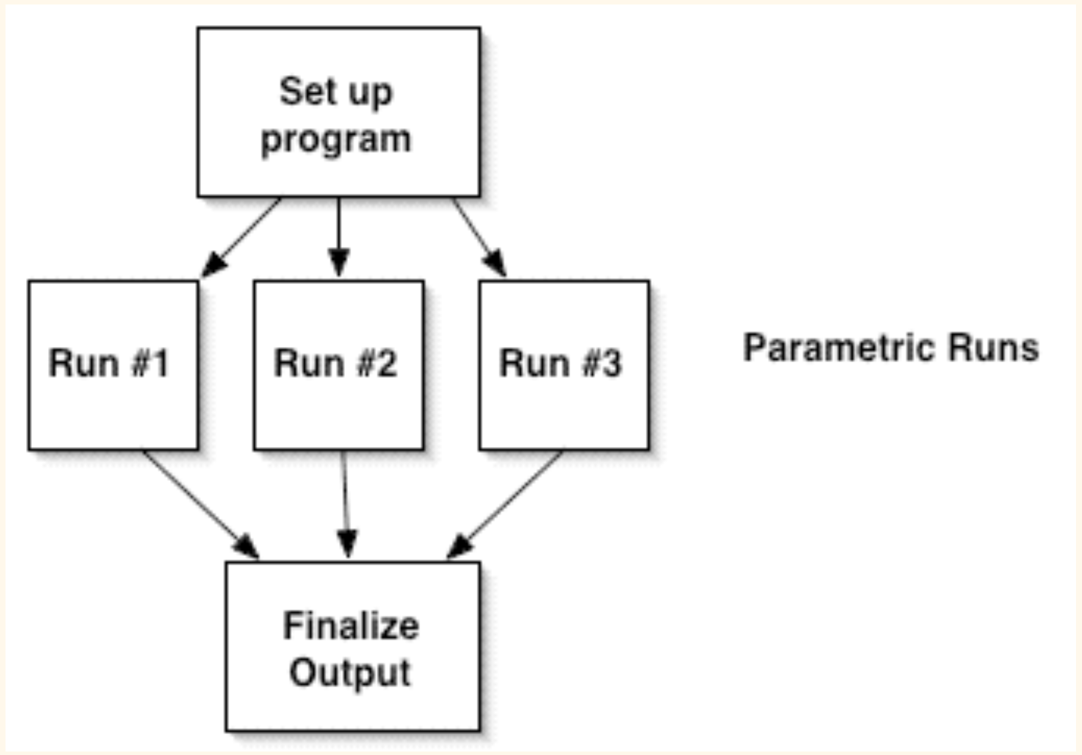
They can also provide hints to the scheduler:

```
TimeTaken = 60
+MaxExecutionTime = $(TimeTaken) # Short jobs get priority
+JobPrio =  -$(TimeTaken)

Rank = memory  # Use amount of RAM to break ties
```

Useful Condor commands:

```
$ condor_q   # list local Condor jobs

$ condor_q - global   # list all Condor jobs

$ condor_q - global - submitter YOURNAME   # list your jobs

$ condor_status   # query the Condor system for other data

$ condor_status HOSTNAME  # view slot info on HOSTNAME

# view stats on all hosts with >= 20 CPU cores

$ condor_status -format “%s “ Name -format “%d “ Cpus \
  -format “%d\n” Memory -constraint “Cpus >= 20”

$ condor_submit -interactive  # get shell on a compute node
```

Users can specify a DAG of related condor jobs:



```
Job Setup setup.submit
Job WorkerNode_1 run1.submit
Job WorkerNode_2 run2.submit
Job WorkerNode_3 run3.submit
Job Finalize finalize.submit
PARENT Setup CHILD WorkerNode_1 WorkerNode_2 WorkerNode_3
PARENT WorkerNode_1 WorkerNode_2 WorkerNode_3 CHILD Finalize
```

```
$ condor_submit_dag myjob.dag
```

http://www.dartmouth.edu/~rc/classes/intro_grid/CondorG_Example7.html

Other (potentially) useful Condor features:

- The “java” universe
- The “parallel” universe
- File transfer
  - Not often necessary, when there is shared storage...
- Checkpointing
  - Only sometimes possible...

Condor use patterns:

Scenario: Process lots of input data in parallel

```
$ run_monkeys.py -brachiate True -num_bananas 4 -input my_file.txt
```

Solution: shard data, key shards by process ID:

```
executable = run_monkeys.py
universe = vanilla
initialdir = /g/steven/monkey_data/

arguments = -brachiate True -num_bananas 1 -input $(Process).txt
output = /g/steven/monkeys/out/$(Process).out
error = /g/steven/monkeys/err/$(Process).err
queue 100 # or however many shards there are
```

Use DAG to consolidate output?

Condor use patterns:

Scenario: Grid-searching over parameter space

```
$ run_monkeys.py -brachiate True -num_bananas 1 -input my_input.txt
```

1. Programmatically generate submit file, queue once

2. Pass process ID in as argument, queue many

1. Programmatically generate submit file, queue once

```
executable = run_monkeys.py
universe = vanilla
initialdir = /g/steven/monkey_data/

arguments = -brachiate True -num_bananas 1 -input my_input.txt
output = /g/steven/monkeys/out/brachiate.True.num.1.out
error = /g/steven/monkeys/err/brachiate.True.num.1.err
queue

arguments = -brachiate True -num_bananas 2 -input my_input.txt
output = /g/steven/monkeys/out/brachiate.True.num.2.out
error = /g/steven/monkeys/err/brachiate.True.num.2.err
queue

...
arguments = -brachiate False -num_bananas 1 -input my_input.txt
output = /g/steven/monkeys/out/brachiate.False.num.1.out
error = /g/steven/monkeys/err/brachiate.False.num.1.err
queue
```

2. Pass process ID in as argument, queue many

```
executable = run_monkeys.py
universe = vanilla
initialdir = /g/steven/monkey_data/

arguments = -setting $(Process) -input my_input.txt
output = /g/steven/monkeys/out/$(Process).out
error = /g/steven/monkeys/err/$(Process).err
queue 200
```

```
true 1
true 2
true 3
true 4
...
true 100
false 1
false 2
...
```

```
should_brachiate = parameters[args.setting][0]
num_bananas = parameters[args.setting][1]
```

Game plan for today:

Introduction to Condor

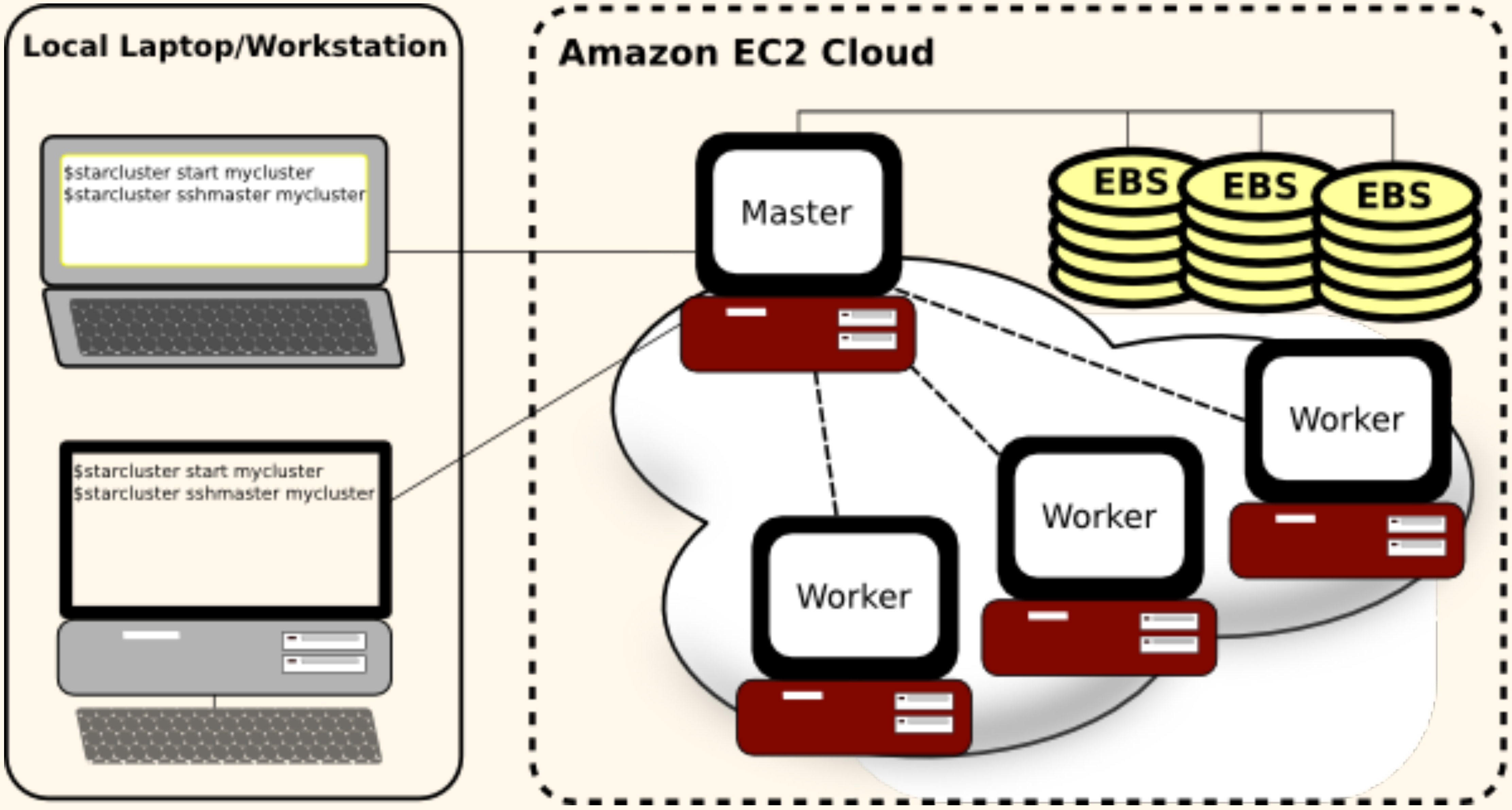
Condor use patterns

StarCluster: On-demand clusters, in “The Cloud”!

Course logistics update

StarCluster:

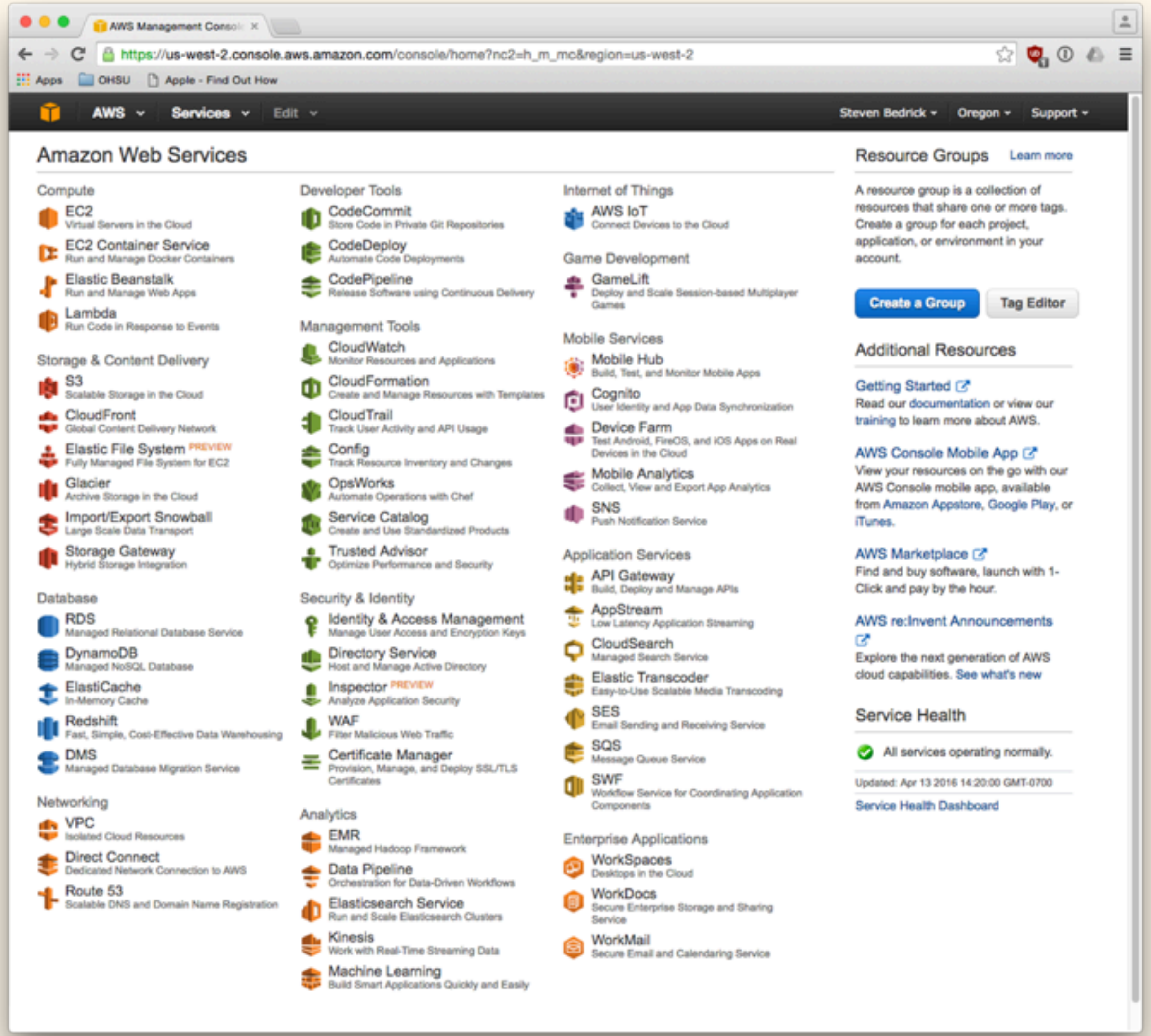
A Python library to automatically setup and configure clusters on AWS!



```
$starcluster start mycluster
$starcluster sshmaster mycluster
```

```
$starcluster start mycluster
$starcluster sshmaster mycluster
```

AWS is Amazon.com's suite of cloud services:



AWS Management Conso

https://us-west-2.console.aws.amazon.com/console/home?nc2=h_m_mc&region=us-west-2

Apps OHSU Apple - Find Out How

AWS Services Edit Steven Bedrick Oregon Support

Amazon Web Services

Compute
- EC2 — Virtual Servers in the Cloud
- EC2 Container Service — Run and Manage Docker Containers
- Elastic Beanstalk — Run and Manage Web Apps
- Lambda — Run Code in Response to Events

Storage & Content Delivery
- S3 — Scalable Storage in the Cloud
- CloudFront — Global Content Delivery Network
- Elastic File System PREVIEW — Fully Managed File System for EC2
- Glacier — Archive Storage in the Cloud
- Import/Export Snowball — Large Scale Data Transport
- Storage Gateway — Hybrid Storage Integration

Database
- RDS — Managed Relational Database Service
- DynamoDB — Managed NoSQL Database
- ElastiCache — In-Memory Cache
- Redshift — Fast, Simple, Cost-Effective Data Warehousing
- DMS — Managed Database Migration Service

Networking
- VPC — Isolated Cloud Resources
- Direct Connect — Dedicated Network Connection to AWS
- Route 53 — Scalable DNS and Domain Name Registration

Developer Tools
- CodeCommit — Store Code in Private Git Repositories
- CodeDeploy — Automate Code Deployments
- CodePipeline — Release Software using Continuous Delivery

Management Tools
- CloudWatch — Monitor Resources and Applications
- CloudFormation — Create and Manage Resources with Templates
- CloudTrail — Track User Activity and API Usage
- Config — Track Resource Inventory and Changes
- OpsWorks — Automate Operations with Chef
- Service Catalog — Create and Use Standardized Products
- Trusted Advisor — Optimize Performance and Security

Security & Identity
- Identity & Access Management — Manage User Access and Encryption Keys
- Directory Service — Host and Manage Active Directory
- Inspector PREVIEW — Analyze Application Security
- WAF — Filter Malicious Web Traffic
- Certificate Manager — Provision, Manage, and Deploy SSL/TLS Certificates

Analytics
- EMR — Managed Hadoop Framework
- Data Pipeline — Orchestration for Data-Driven Workflows
- Elasticsearch Service — Run and Scale Elasticsearch Clusters
- Kinesis — Work with Real-Time Streaming Data
- Machine Learning — Build Smart Applications Quickly and Easily

Internet of Things
- AWS IoT — Connect Devices to the Cloud

Game Development
- GameLift — Deploy and Scale Session-based Multiplayer Games

Mobile Services
- Mobile Hub — Build, Test, and Monitor Mobile Apps
- Cognito — User Identity and App Data Synchronization
- Device Farm — Test Android, FireOS, and iOS Apps on Real Devices in the Cloud
- Mobile Analytics — Collect, View and Export App Analytics
- SNS — Push Notification Service

Application Services
- API Gateway — Build, Deploy and Manage APIs
- AppStream — Low Latency Application Streaming
- CloudSearch — Managed Search Service
- Elastic Transcoder — Easy-to-Use Scalable Media Transcoding
- SES — Email Sending and Receiving Service
- SQS — Message Queue Service
- SWF — Workflow Service for Coordinating Application Components

Enterprise Applications
- WorkSpaces — Desktops in the Cloud
- WorkDocs — Secure Enterprise Storage and Sharing Service
- WorkMail — Secure Email and Calendaring Service

Resource Groups Learn more

A resource group is a collection of resources that share one or more tags. Create a group for each project, application, or environment in your account.

Create a Group | Tag Editor

Additional Resources

Getting Started
Read our documentation or view our training to learn more about AWS.

AWS Console Mobile App
View your resources on the go with our AWS Console mobile app, available from Amazon Appstore, Google Play, or iTunes.

AWS Marketplace
Find and buy software, launch with 1-Click and pay by the hour.

AWS re:Invent Announcements
Explore the next generation of AWS cloud capabilities. See what's new

Service Health

All services operating normally.

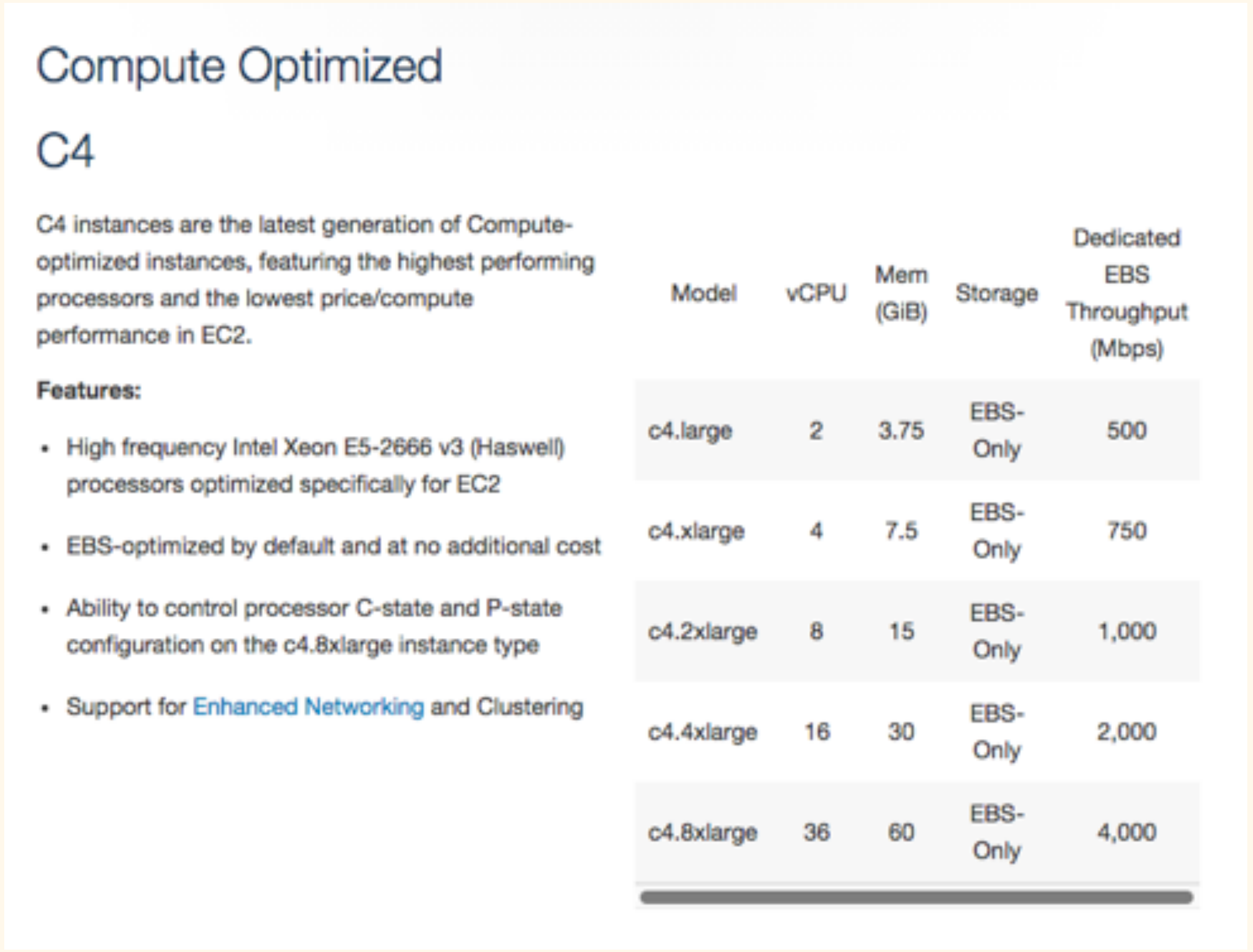
Updated: Apr 13 2016 14:20:00 GMT-0700

Service Health Dashboard

Key ideas:

EC2 "instances" are virtual machines...

Instances can be of many sizes:



Compute Optimized

C4

C4 instances are the latest generation of Compute-optimized instances, featuring the highest performing processors and the lowest price/compute performance in EC2.

Features:

- High frequency Intel Xeon E5-2666 v3 (Haswell) processors optimized specifically for EC2
- EBS-optimized by default and at no additional cost
- Ability to control processor C-state and P-state configuration on the c4.8xlarge instance type
- Support for Enhanced Networking and Clustering

Model	vCPU	Mem (GiB)	Storage	Dedicated EBS Throughput (Mbps)
c4.large	2	3.75	EBS-Only	500
c4.xlarge	4	7.5	EBS-Only	750
c4.2xlarge	8	15	EBS-Only	1,000
c4.4xlarge	16	30	EBS-Only	2,000
c4.8xlarge	36	60	EBS-Only	4,000

https://aws.amazon.com/ec2/instance-types/

Payment by cpu time, varies by instance size

Key ideas:

EBS = “Elastic Block Volumes”

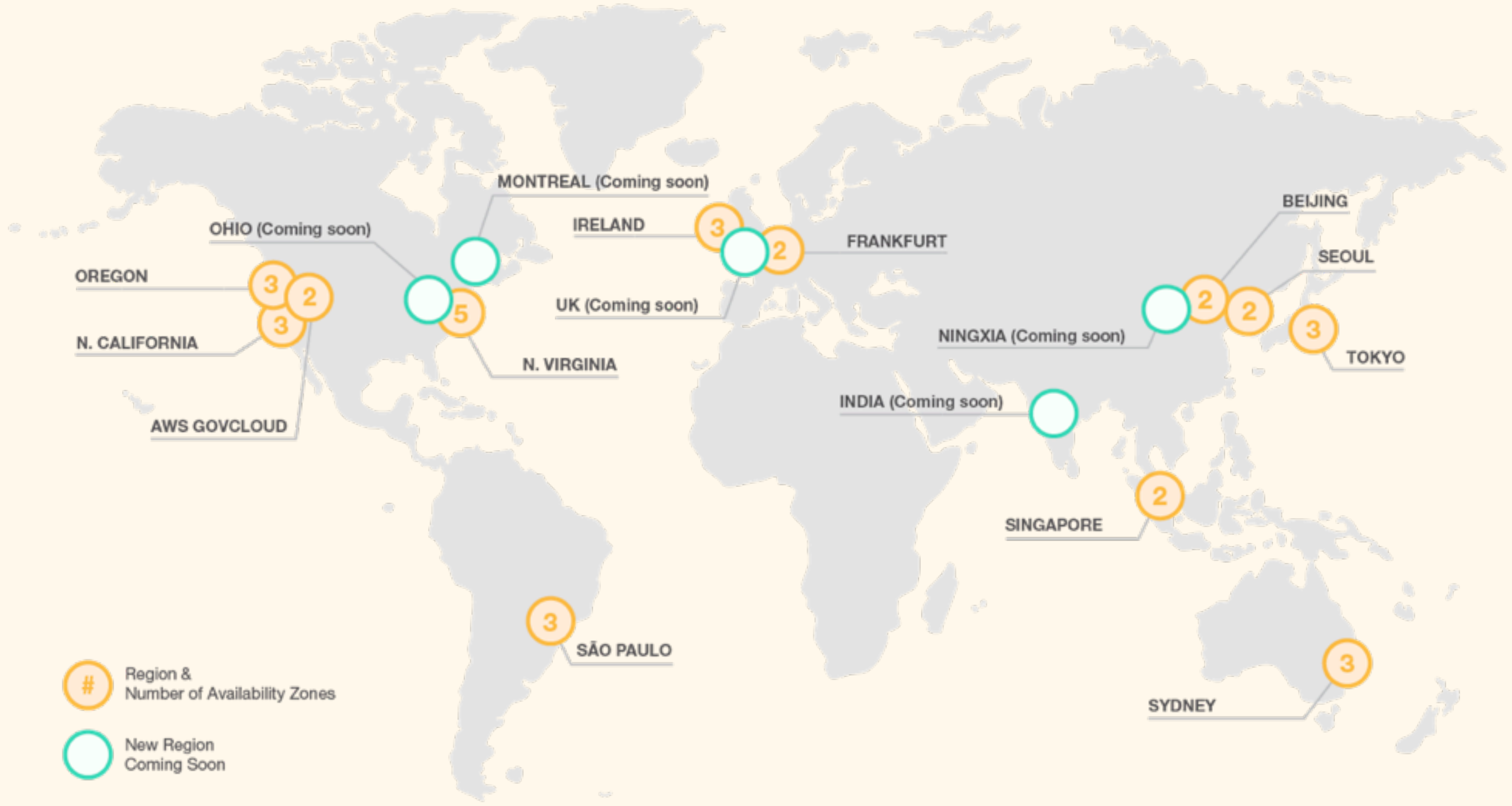
Virtual storage, can be of (almost) any size or type.

Payment based on space used and amount of I/O.

S3 = “Simple Storage System”

“Bucket” of data in the sky

Payment based on space used and number of requests


MONTREAL (Coming soon)
IRELAND
3
2
FRANKFURT
OHIO (Coming soon)
OREGON
3
2
3
5
N. CALIFORNIA
UK (Coming soon)
N. VIRGINIA
AWS GOVCLOUD
BEIJING
SEOUL
2
2
3
NINGXIA (Coming soon)
TOKYO
INDIA (Coming soon)
2
SINGAPORE
3
SÃO PAULO
3
SYDNEY
#
Region &
Number of Availability Zones
New Region
Coming Soon

Setting up an EC2 instance is easy...

... configuring many instances gets old fast.

Enter StarCluster!