AWS re:Invent

DAT406

Netflix: Iterating on Stateful Services in the Cloud

Joey Lynch
Senior Software Engineer
Netflix

Speaker

Joey Lynch

Senior Software Engineer on Cloud Database Engineering at Netflix

Distributed system addict and data wrangler

Agenda

Netflix cloud philosophy

What makes state difficult

Strategies for managing state

How to take it to 11

Netflix cloud philosophy

AMI is source of truth

- "Base" AMI
  - Operating system (such as Ubuntu)
  - Virtualization type
- Installed software
  - Linux kernel version
  - User software

```
context: !bunch.Bunch
  ami: !bunch.Bunch
    base_ami_name: xenialbase-x86_64-201805030044-ebs
    creator: owner-team-email@netflix.com
    ena_networking: true
    enhanced_networking: true
    suffix: 20181117192736-xenial-hvm-sriov
    tags: !bunch.Bunch {}
    vm_type: hvm
```

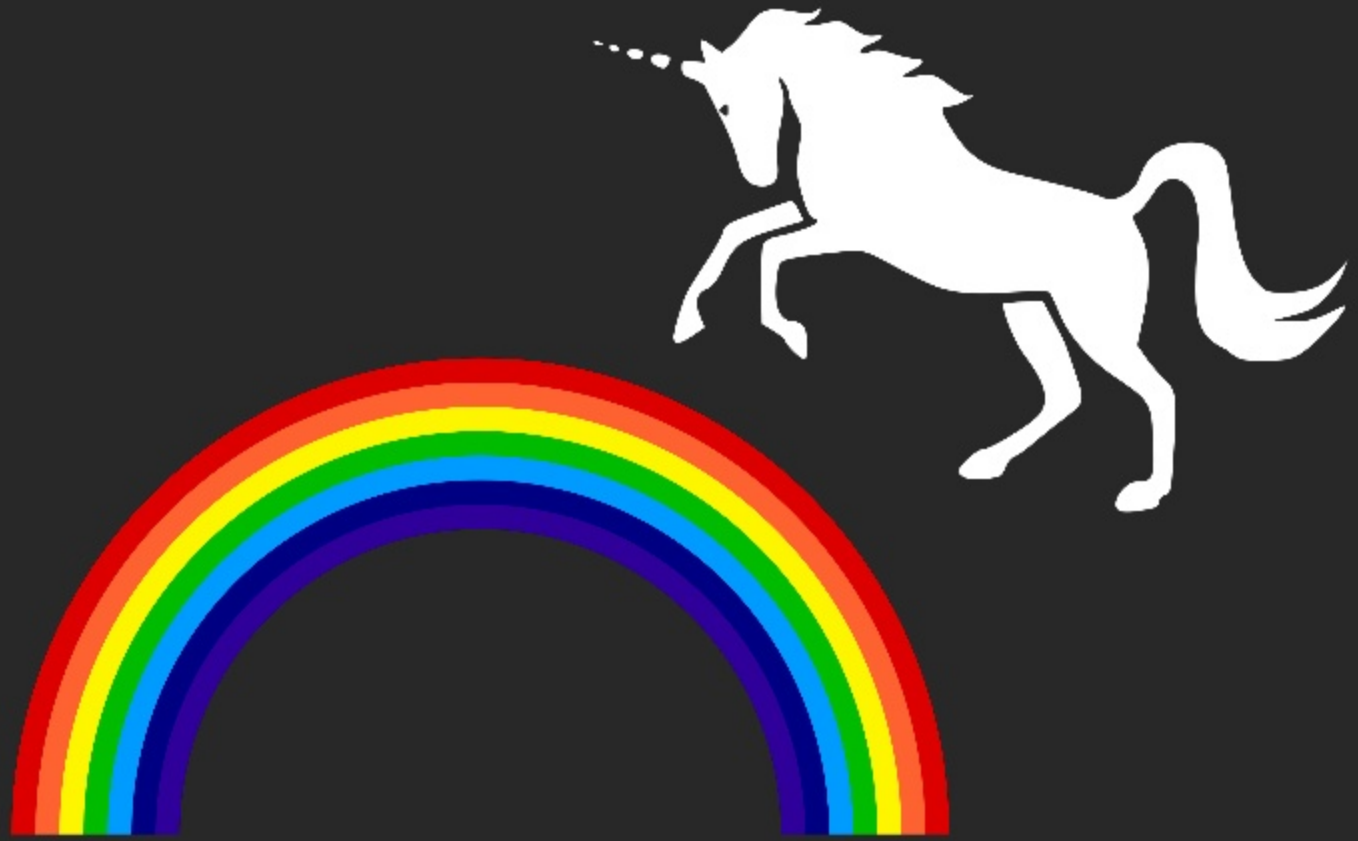
```
2018-11-17 19:28:56 [DEBUG] The following NEW packages will be installed:
  cde-bootup{a} cde-cass-filebeat{a} cde-coredump-uploader{a}
  cdecassostweaks{a} cdecassutils{a} cdedatascrambler-spring{a} cphalo{a}
  dstat{a} gandalf-agent{a} happycache{a} jvmquake{a} libgd3{a} libjbig0{a}
  libjpeg-turbo8{a} libjpeg8{a} liblz4-tool{a} libtiff5{a} libvpx3{a}
  libxpm4{a} libxslt1.1{a} linux-headers-4.13.0-19{a}
  linux-headers-4.13.0-19-generic{a} linux-hwe-edge-tools-4.13.0-19{a}
  linux-image-4.13.0-19-generic{a} linux-image-extra-4.13.0-19-generic{a}
  linux-tools-4.13.0-19-generic{a} metatron-persistence{a} mlocate{a}
  nf-spectator-agent{a} nflx-bolt{a} nflx-cde-kyber{a} nflx-cloudpassage{a}
  nflx-ezconfig{a} nflx-python-3.5.5{a} nflx-python-3.6.4{a} nfpriam2{a}
  nfpy27-awscli{a} nfpy27-base{a} nfpy27-botocore{a} nfpy27-certifi{a}
  nfpy27-chardet{a} nfpy27-colorama{a}
  nfpy27-gevent{a} nfpy27-greenlet{a}
  nfpy27-pip{a} nfpy27-pyasn1{a} nfpy2
  nfpy27-requests{a} nfpy27-rsa{a} nfpy27-s3transfer{a}
  nfpy27-setuptools{a} nfpy27-six{a} nfpy27-urllib3{a} nginx{a}
  nginx-common{a} nginx-core{a} oss-cass3x{a} parallel{a} s3gof3r{a}
  schedstat-collector{a}
```

A "Cassandra" AMI

Huzzah immutability!

- Build: bake an AMI
- Deployment: red-black ASGs with new AMIs
- Scaling + Failure: ASGs just take care of it

Huzzah immutability!



But what's in a typical stateful instance?

Service

- Cassandra
- Elasticsearch
- Memcache
- Zookeeper
- More

Configuration

- Sysctls
- IO Scheduler
- TC qdisc
- Readahead tunings
- More

Tooling

- Perf tools
- State transfer
- Metrics
- Monitoring
- More

And?

Mutable state

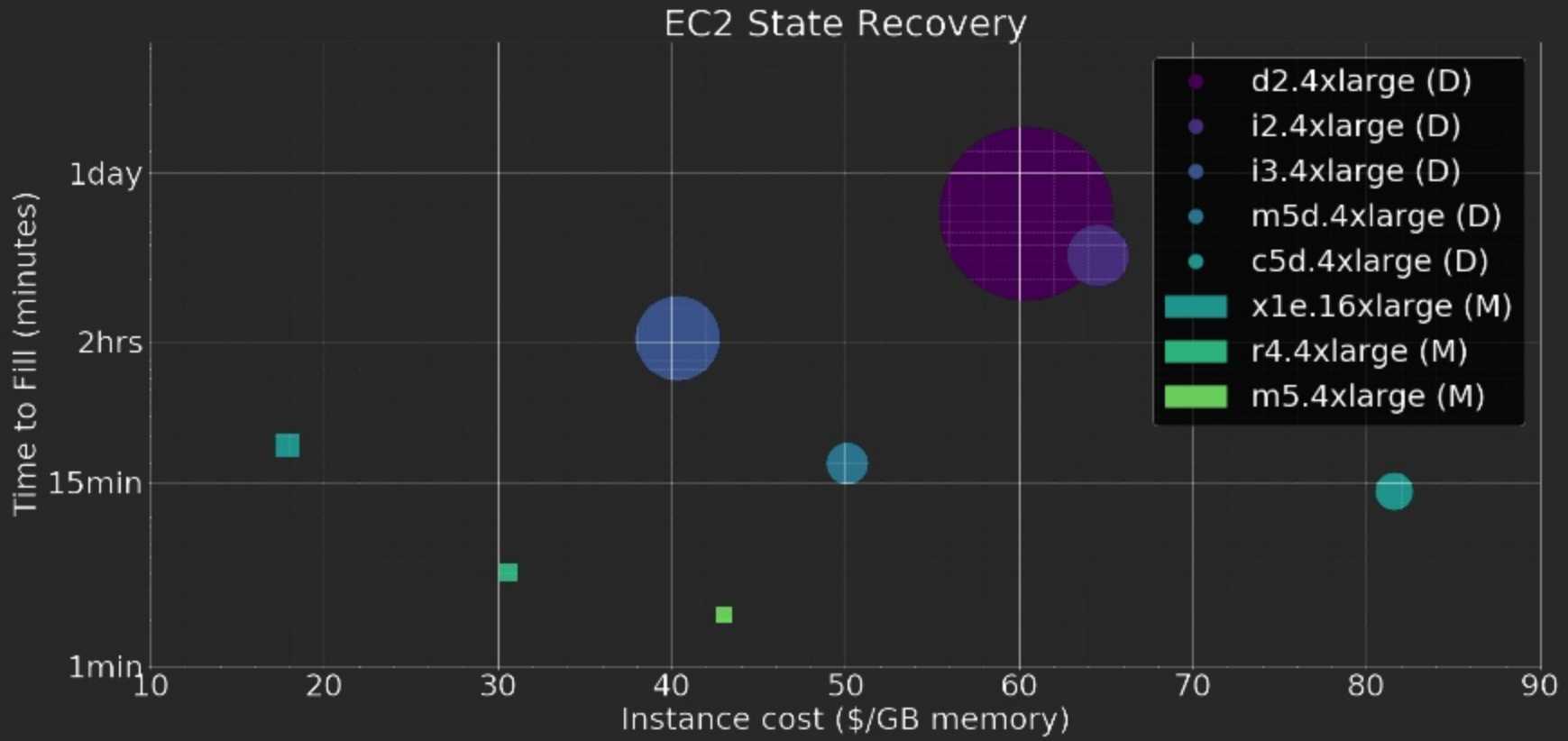
We made a state “closet”

SOAs pushed the challenges of state and state management down a layer into the cache/data tier

This didn’t make the problem go away

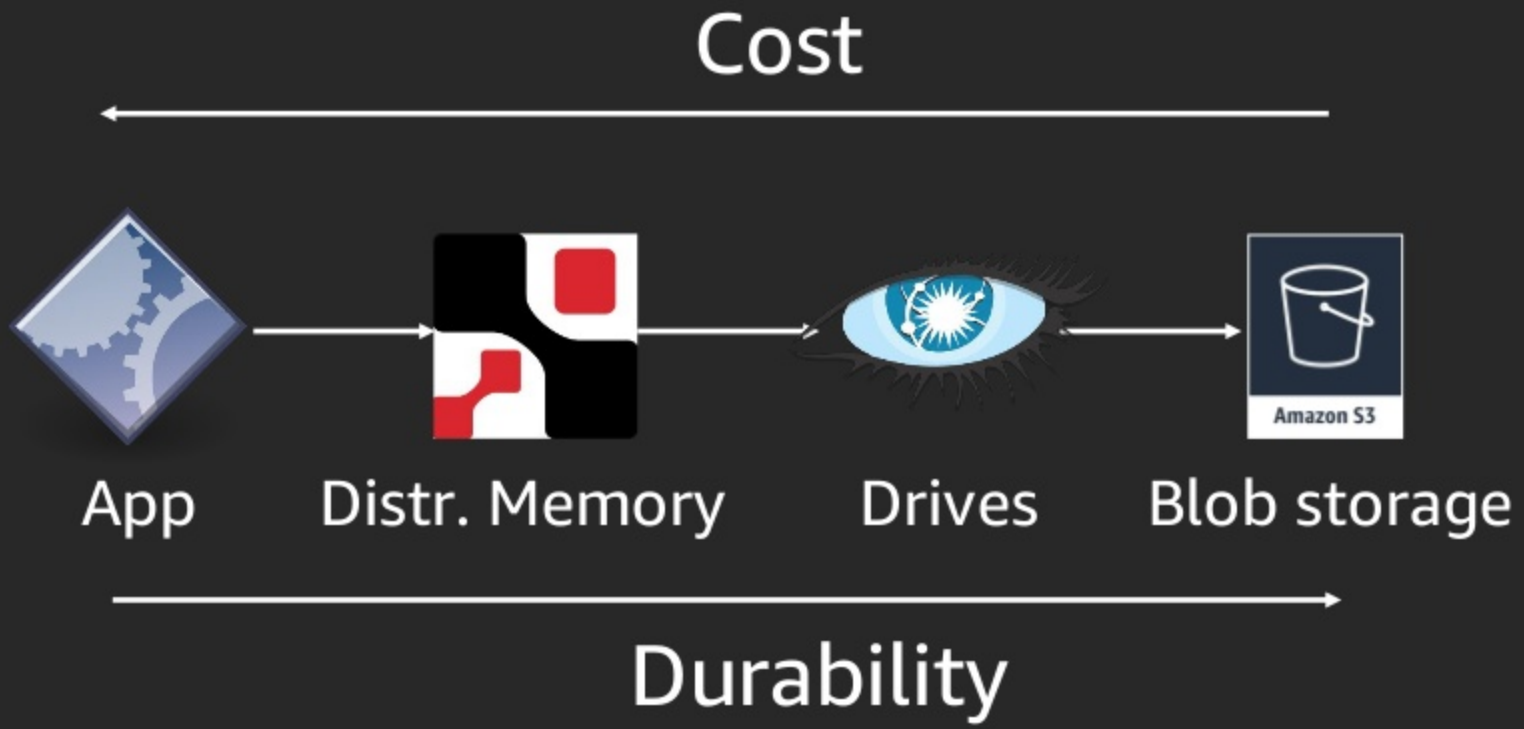
State is heavy

Instance fill time

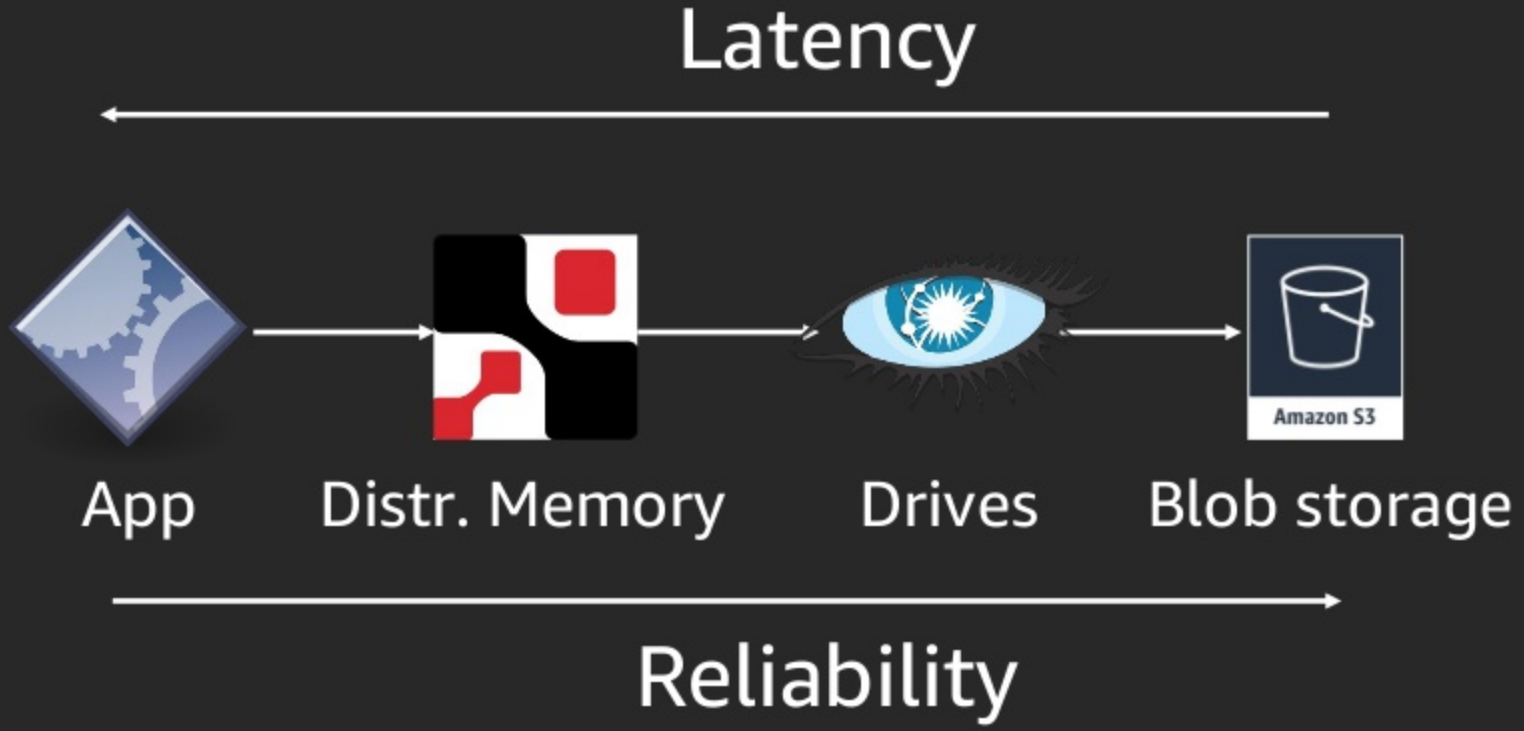


https://github.com/jolynch/python_performance_toolkit/tree/master/notebooks/aws_stateful_instance_analysis

It's caches all the way down



It's caches all the way down



What makes stateful services different?

1) Stateful services have, well … *state*

    Means you can't just throw the instance away

2) Stateful services have many components

    Datastore, sidecars, cron jobs, monitoring, metrics

3) Require different iteration rates

Strategies for managing state

Strategy 0:
Don’t manage state

Use another closet (Amazon Elastic Block Store [Amazon EBS] gp2 and io1)

We can push the state even further into the network layer

The database is now “stateless.”

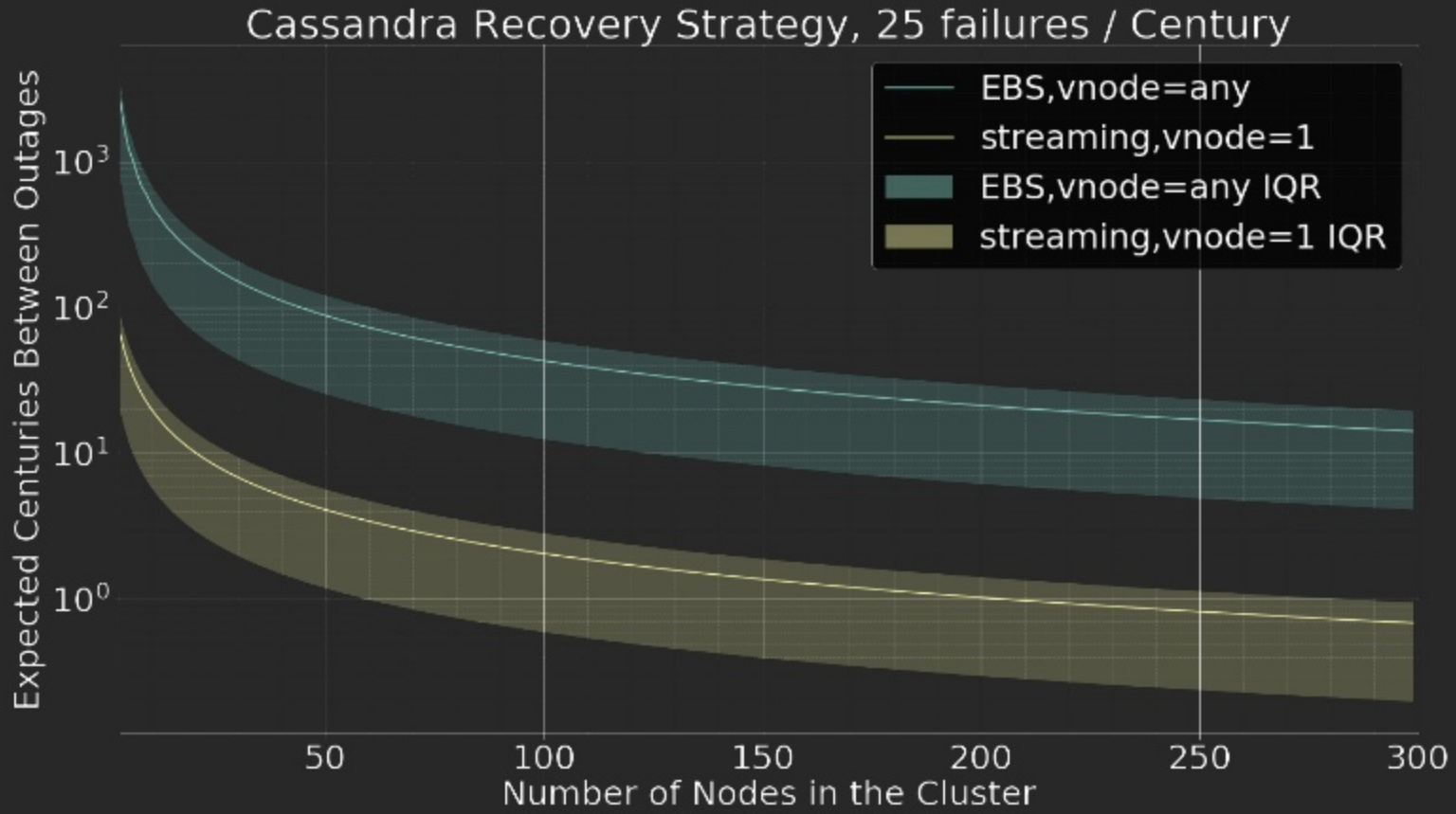
AWS instance state is very fast

```
$ sudo /usr/share/bcc/tools/biosnoop > ios.txt
$ cat ios.txt | tr -s ' ' | grep ' R ' | grep 'nvme.*n1' | cut -f 8 -d ' ' | histogram.py

# NumSamples = 186869; Min = 0.02; Max = 9.34
# Mean = 0.083496; Variance = 0.005143; SD = 0.071713; Median 0.064000
# each ▮ represents a count of 1273
    0.0190 -     0.0190 [     2]:
    ...
    0.0196 -     0.0201 [     2]:
    0.0201 -     0.0213 [     7]:
    0.0213 -     0.0235 [   131]:
    0.0235 -     0.0281 [  3184]: ▮▮
    0.0281 -     0.0372 [  5994]: ▮▮▮▮
    0.0372 -     0.0554 [ 22965]: ▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮
    0.0554 -     0.0918 [ 95519]: ▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮
    0.0918 -     0.1646 [ 54687]: ▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮
    0.1646 -     0.3101 [  3717]: ▮▮
    0.3101 -     0.6012 [   471]:
    0.6012 -     1.1835 [   124]:
    1.1835 -     2.3480 [    31]:
    2.3480 -     4.6770 [    29]:
    4.6770 -     9.3350 [     6]:
```

But … Amazon EBS gp2 is comparatively slow

```
$ sudo /usr/share/bcc/tools/biosnoop > ios.txt
$ cat ios.txt | tr -s ' ' | grep ' R ' | grep '<drive>' | cut -f 8 -d ' ' | histogram.py

# NumSamples = 36328; Min = 0.20; Max = 11.56
# Mean = 1.014655; Variance = 0.394151; SD = 0.627815; Median 0.862000
# each ▮ represents a count of 350
    0.2050 -     0.2050 [     1]:
    0.2050 -     0.2050 [     0]:
    ...
    0.2105 -     0.2161 [     1]:
    0.2161 -     0.2272 [     0]:
    0.2272 -     0.2494 [     1]:
    0.2494 -     0.2937 [     0]:
    0.2937 -     0.3825 [   197]:
    0.3825 -     0.5600 [   912]: ▮▮
    0.5600 -     0.9150 [ 26303]: ▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮
    0.9150 -     1.6250 [  5809]: ▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮
    1.6250 -     3.0450 [  2138]: ▮▮▮▮▮▮
    3.0450 -     5.8850 [   937]: ▮▮
    5.8850 -    11.5650 [    29]:
```

Amazon EBS GP2 means more availability

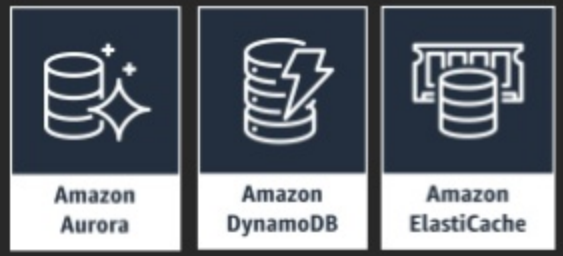


https://github.com/jolynch/python_performance_toolkit/tree/master/notebooks/aws_stateful_instance_analysis

Use a bigger closet (Amazon Aurora, Amazon DynamoDB ...)

We can push the state into datastore as a service options

The state is now someone else's problem!



Should I run my own datastores?

1) Do you need specific datastore APIs or services?

2) Do you need to be under full control of the SLAs of that system?

3) Do you need to meet specific regulatory requirements?

4) Do you have a limited budget and need maximum efficiency?

5) Do you enjoy debugging difficult distributed systems problems?

Before starting: Isolate the state

Instance layout matters

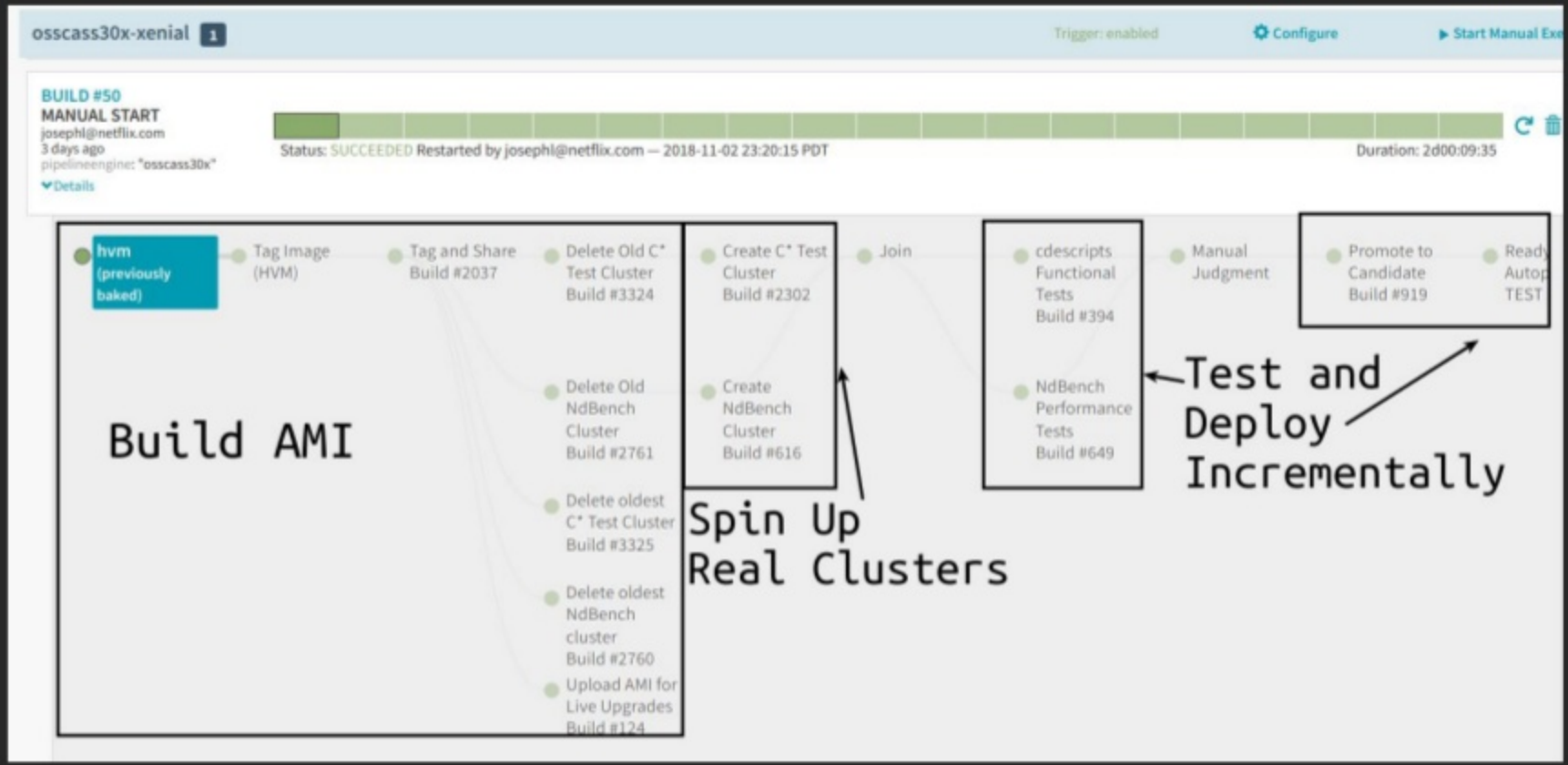
Stateful components

- Data: /mnt/data
- Logs: /var/log

Stateless components

- Primary service
- OS image
- System packages
- Background services
- Metrics, discovery
- More

Test your stateful systems E2E



Strategy 1: Mutate instances in place

Make mutation safe

- Configuration management works
  - Puppet and chef are mature products
- Package upgrades works
  - Linux OS packaging is mature and works
- Container upgrades work
  - Less mature but safer than packages

Why mutate: Speed!

	Interval	Mutation Time	Mutation Time (cluster)
Datastore Upgrade	~months	~1 hour	~days
Configuration Change	~days	~1 second	~minutes
OS Upgrade	~months	~15 minutes	~days
Background Process	~days	~30 seconds	~days
Hardware Change	~weeks	N/A*	N/A*

Mutate instances

Pros

- Super easy
- Fast, as you don't have to move data
- Tried-and-true technique
- Containers + Agents = Less foot shooting

Cons

- Totally violates AMI as source of truth
- Completely untested
  - Transitions
  - States
- Can easily have fleet drift

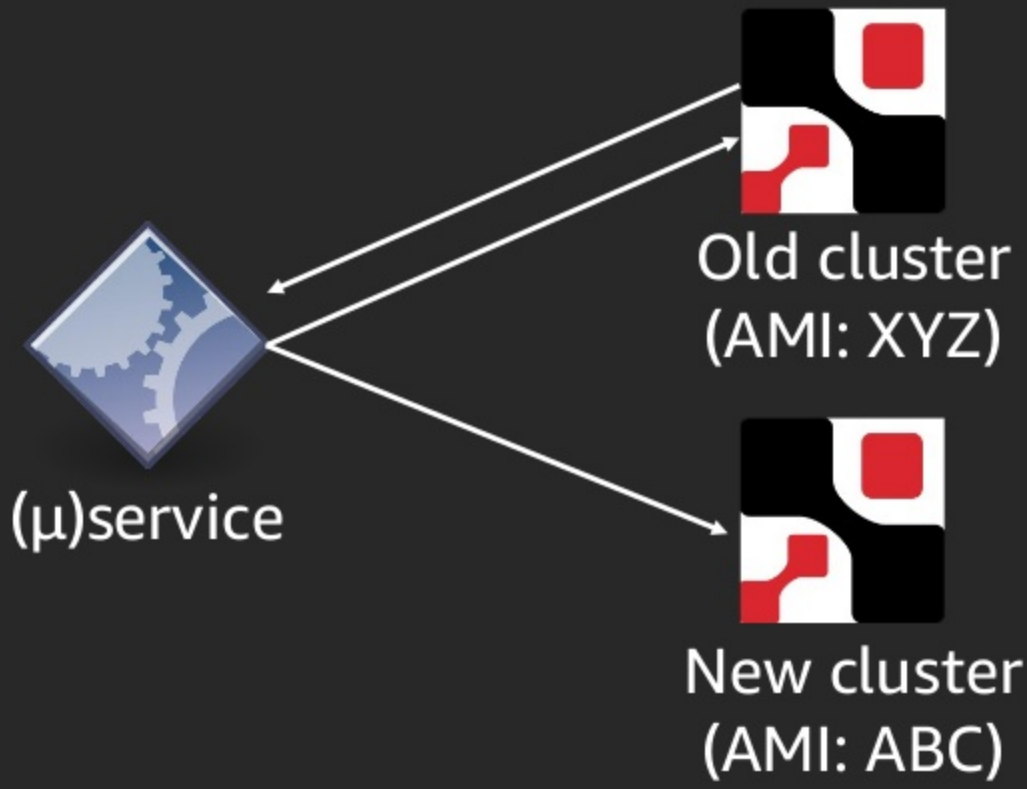
Which strategies does Netflix use?

Don’t manage	Mutate
Amazon EBS st1, gp2 Aurora Amazon Relational Database Service (Amazon RDS) DynamoDB Amazon Simple Storage Service (Amazon S3)	All of them

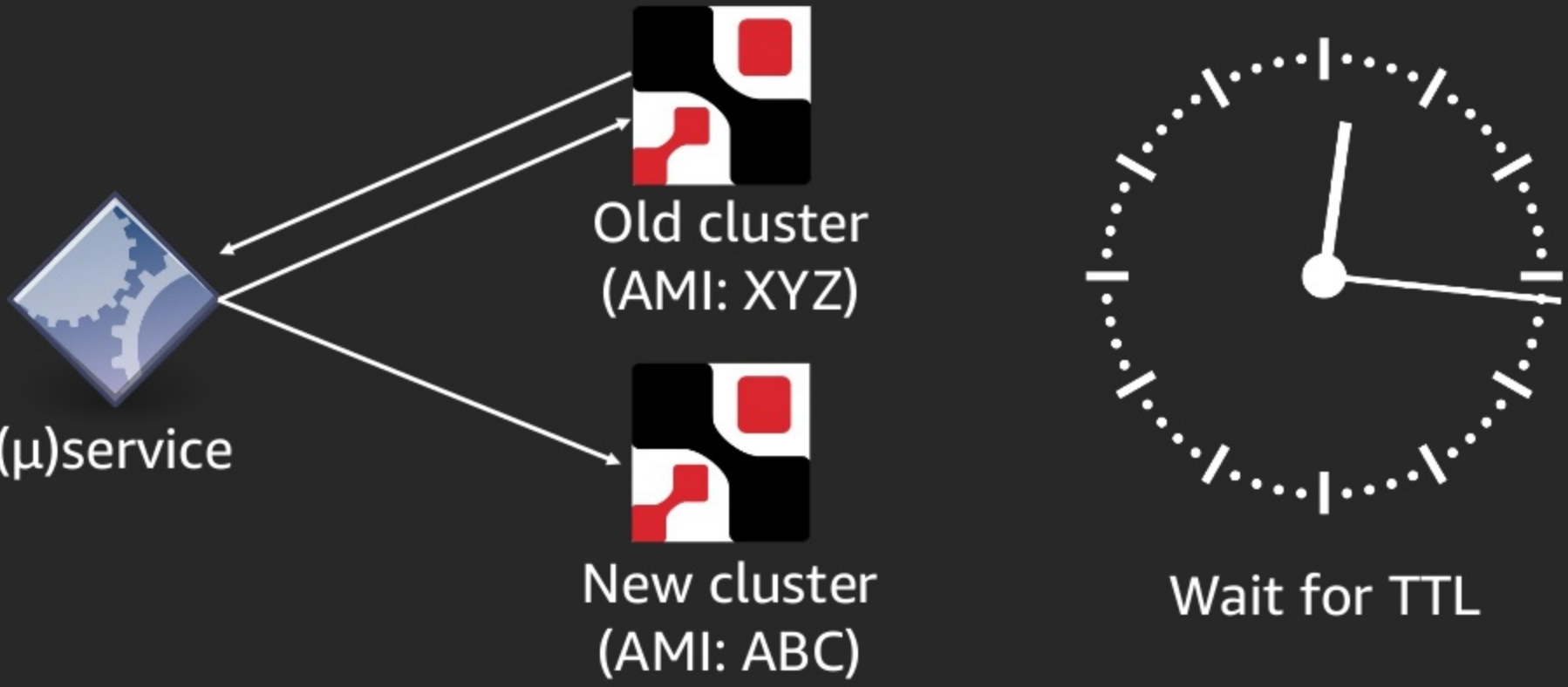
Strategy 2:
Migrate state at the client

Mirroring state

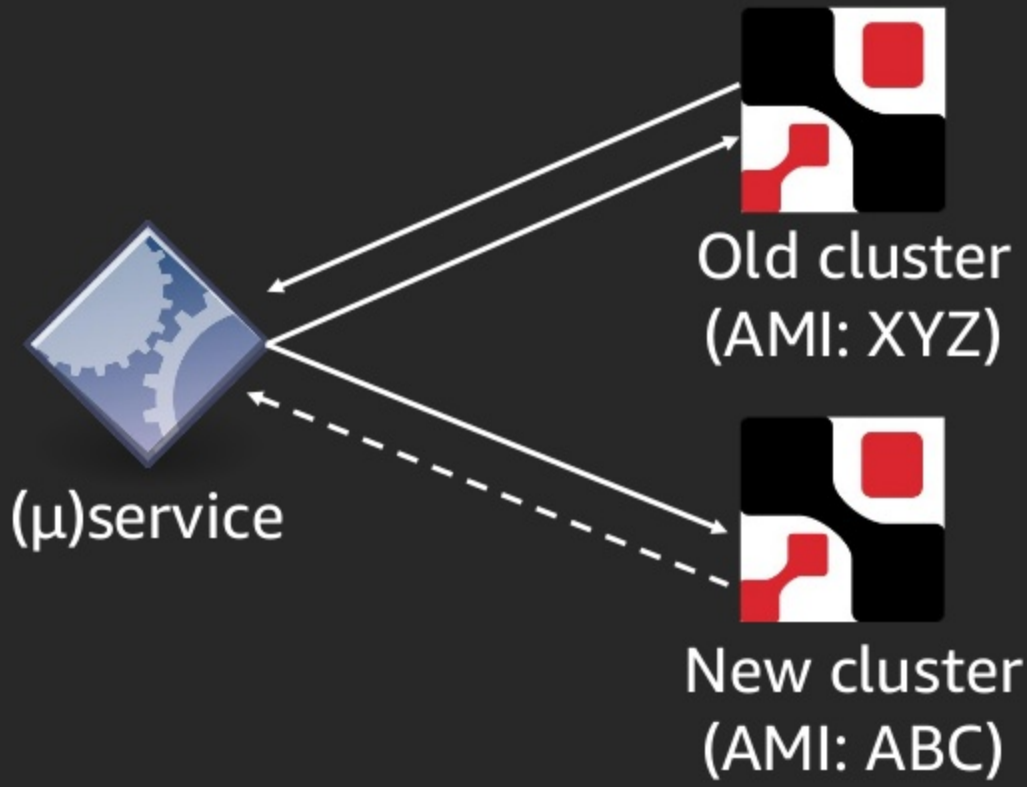
Step 1: Mirror writes



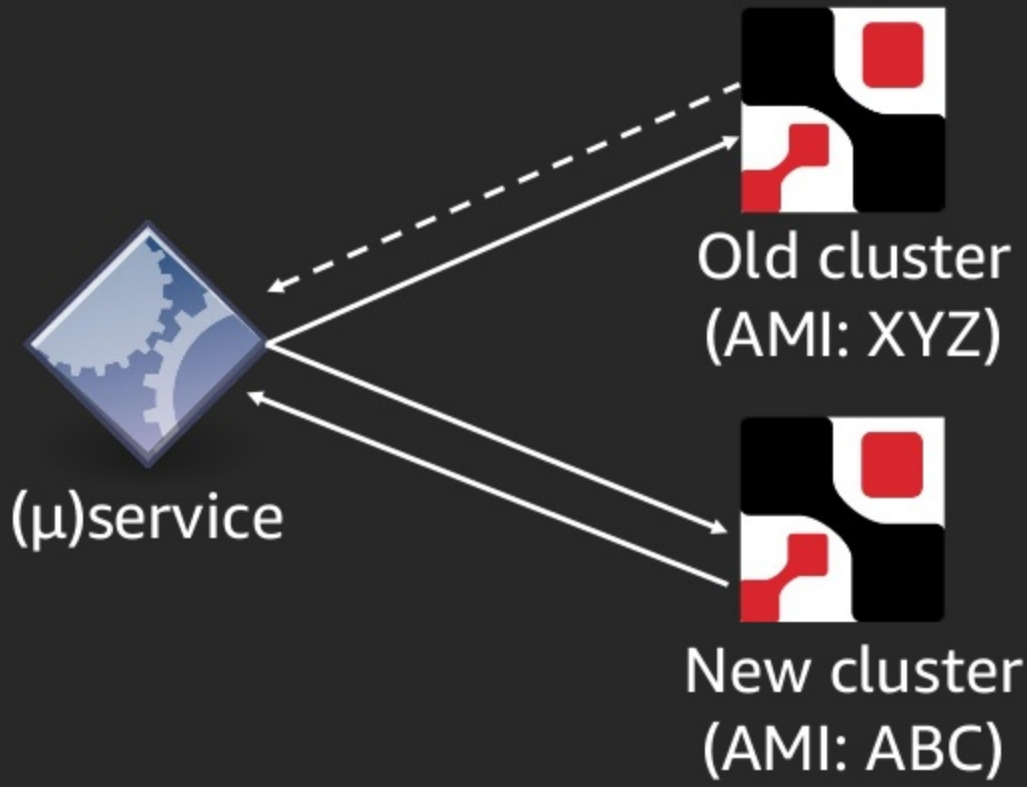
Step 2: Wait



Step 3: Concurrent reads*



Step 4: Live launch reads



Step 5: Retire the old cluster

Migrate state at client

Pros

- Relatively easy to do
- Works well for ephemeral data
  - Caches
  - Search engines
  - Logs

Cons

- Not 100% consistent
- Hard to use with non ephemeral data
  - Persistent caches (yes)
  - Sources of truth
- Harder than mutating instances

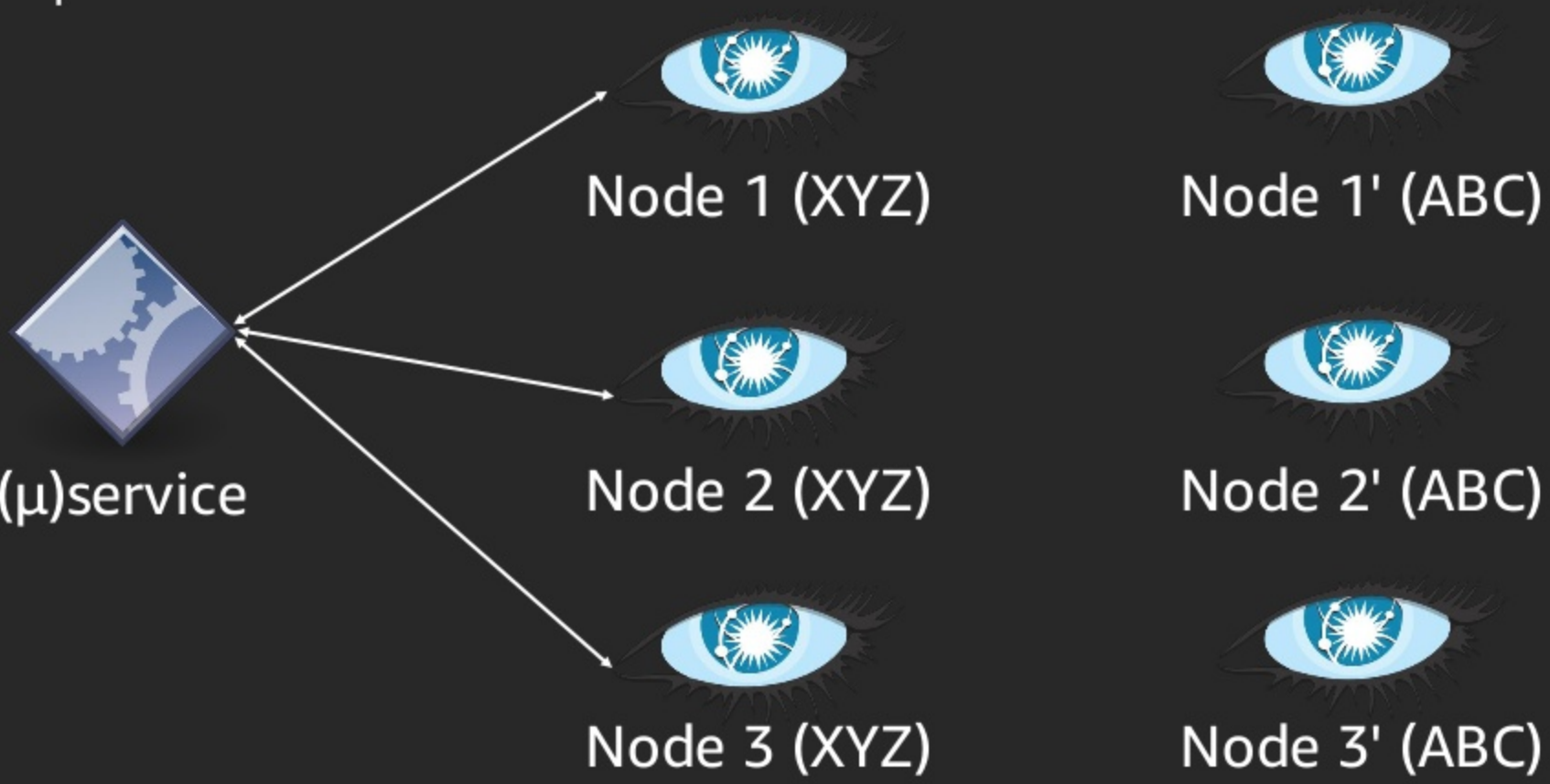
Which strategies does Netflix use?

Don’t manage	Mutate	Client
Amazon EBS st1, gp2 Aurora Amazon RDS DynamoDB Amazon S3	All of them	Caches Search engines Logs

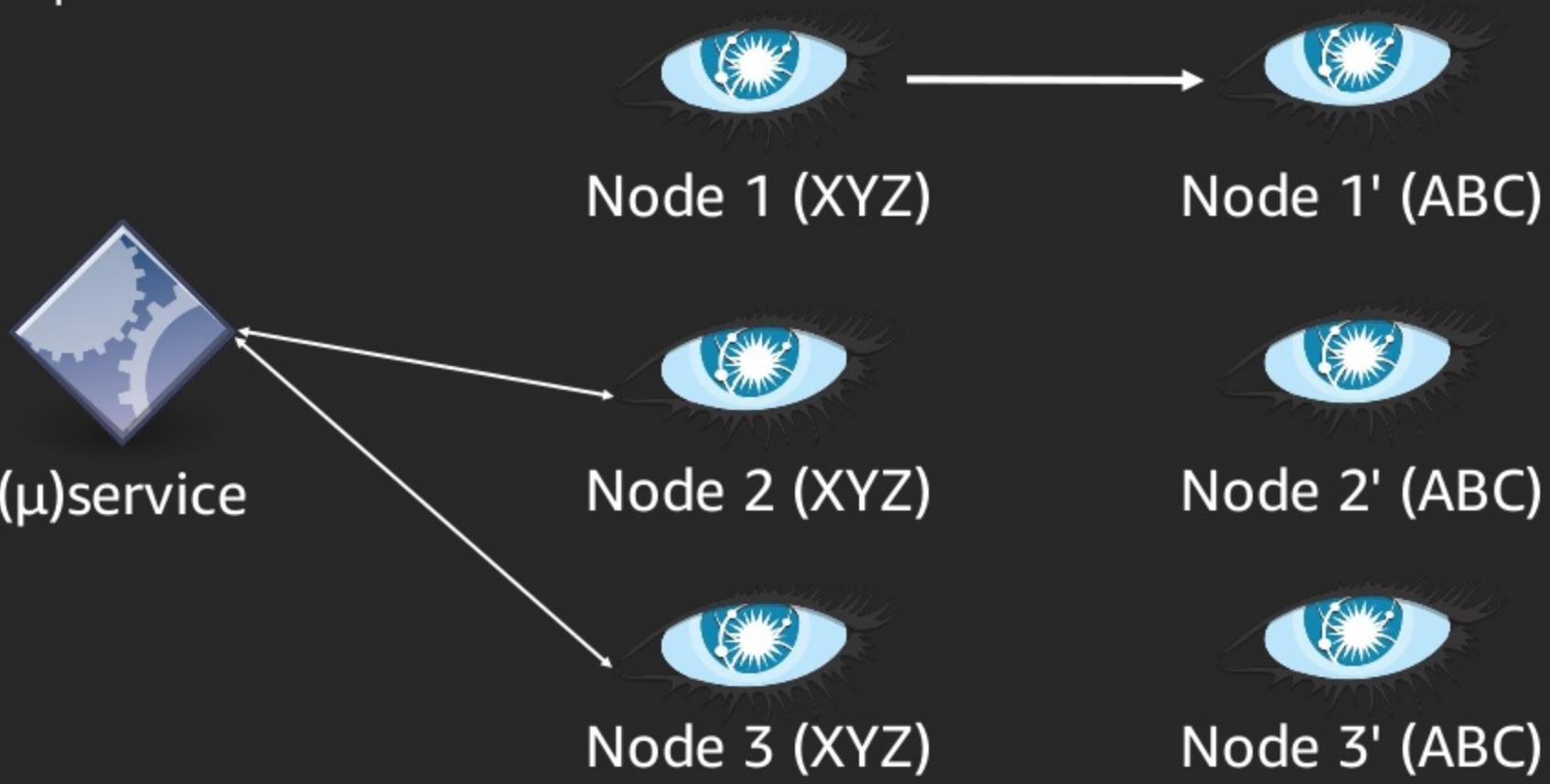
Strategy 3:
Migrate state at the server

Migrate state

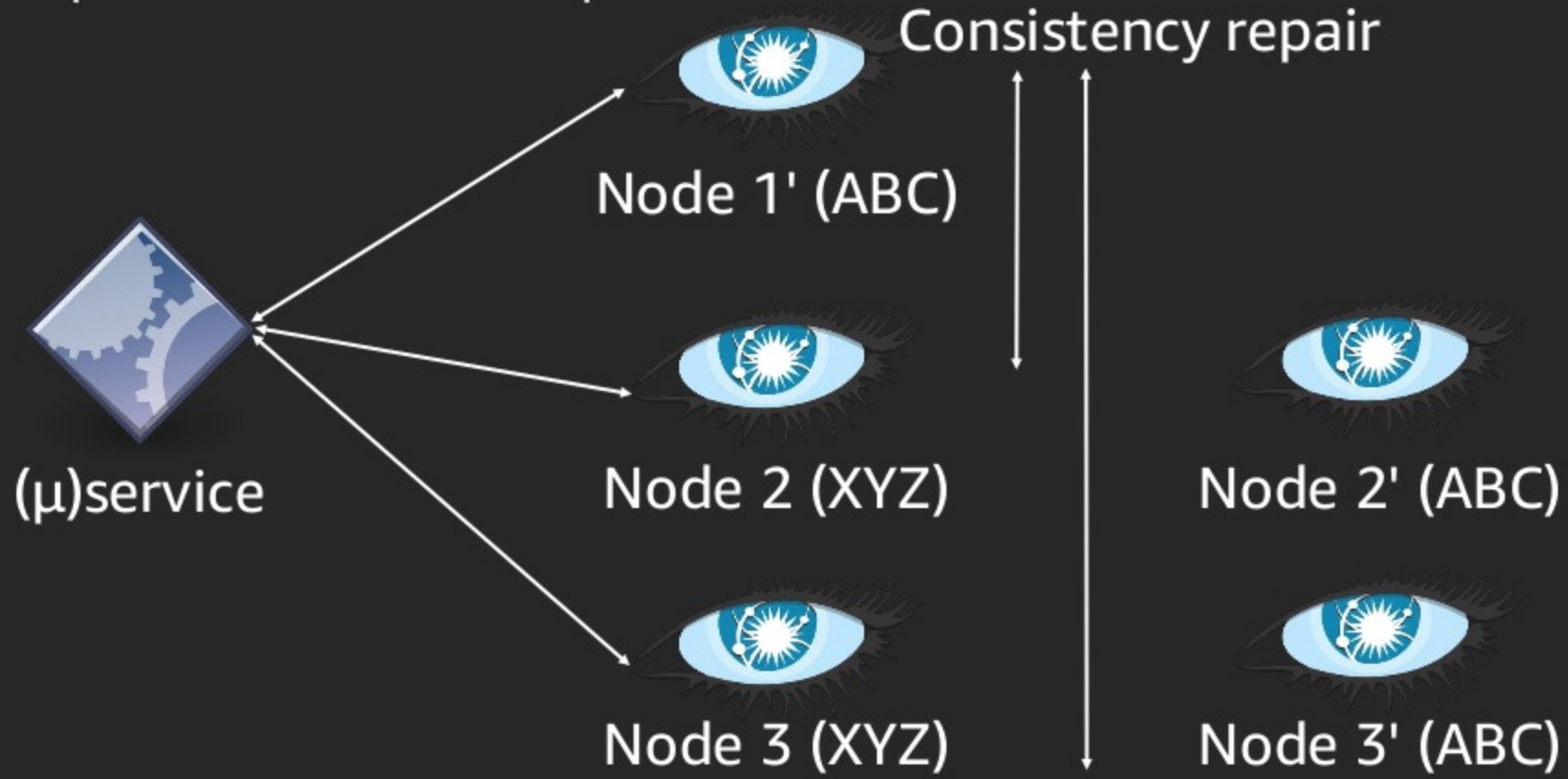
Step 1: Launch new nodes



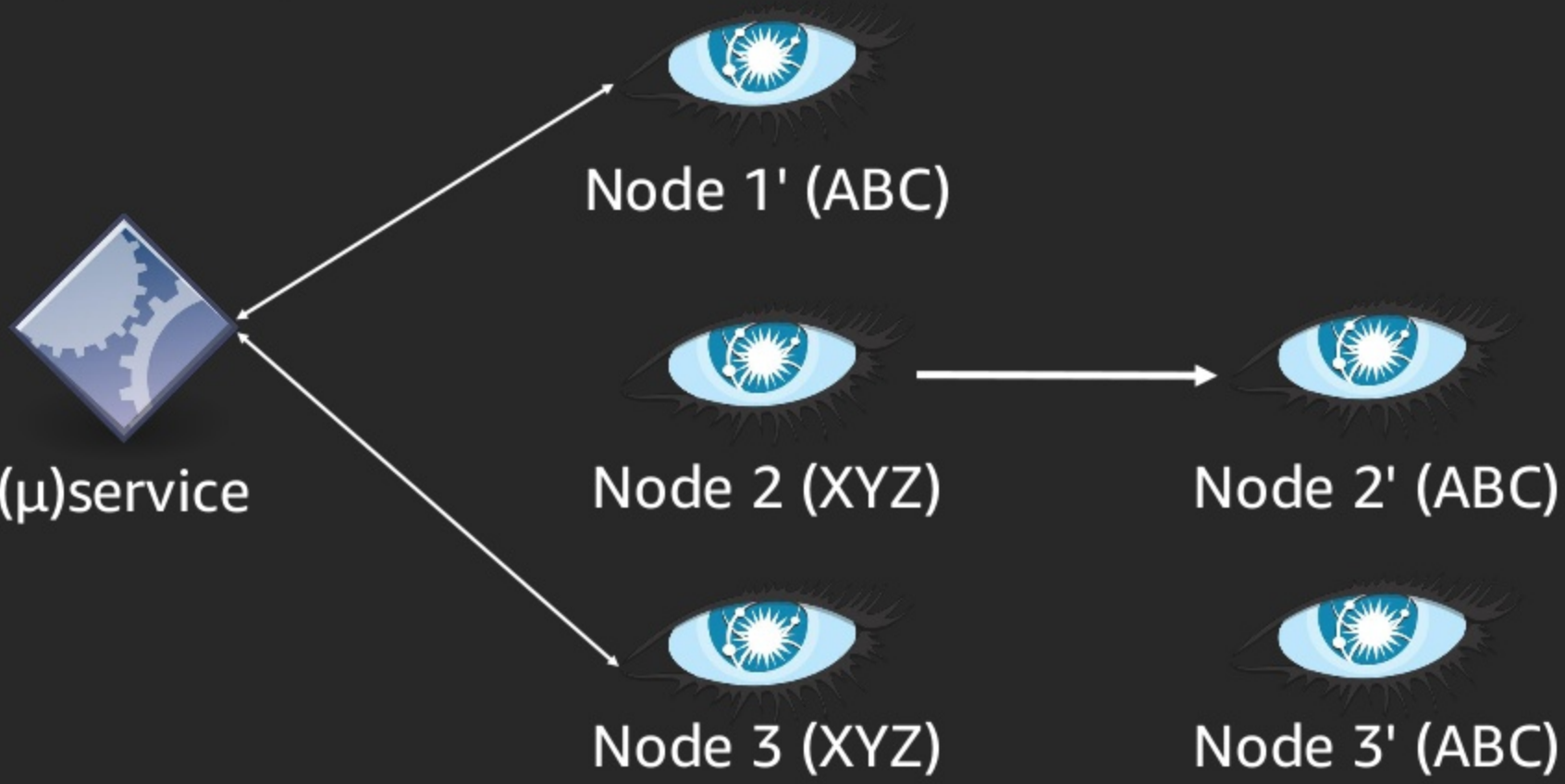
Step 2: Partition and transfer state



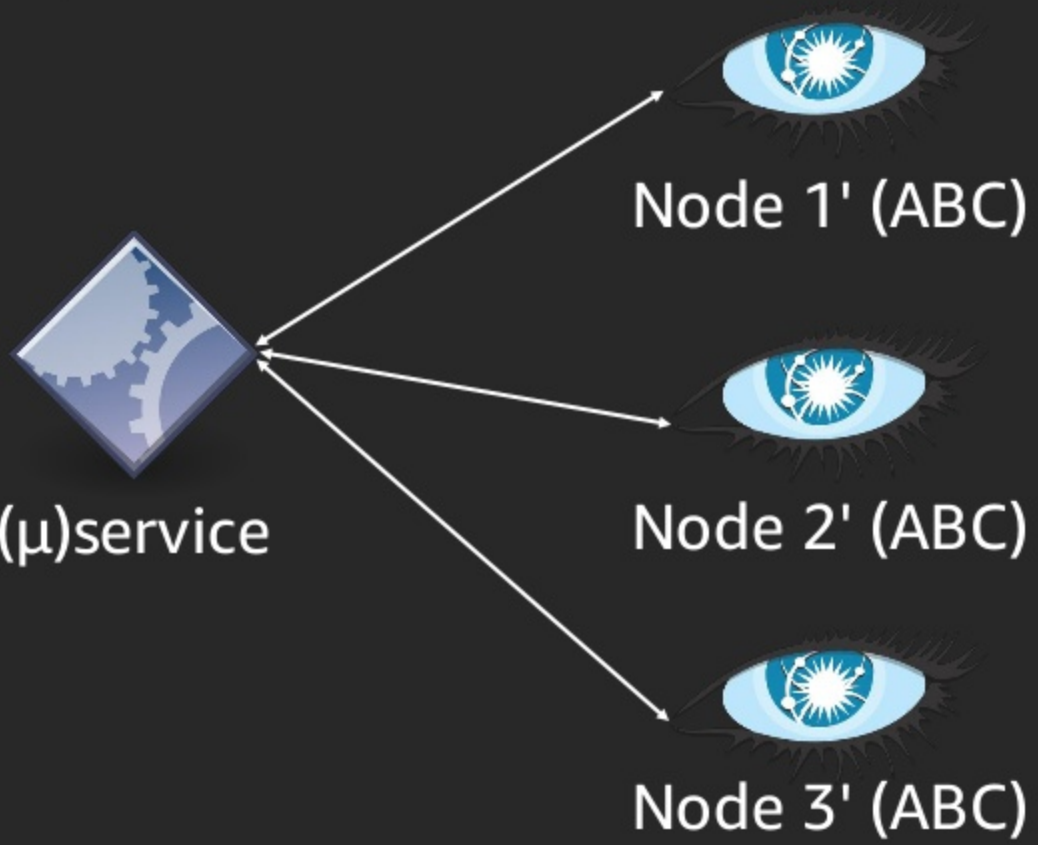
Step 3: Heal and repair



Step 4: Repeat



Step 5: Heal inconsistencies



Data movement

- Consistency repair
- Exploit "single node" failures
- Must ensure data fidelity

Migrate state at server

Pros

- Works for almost any distributed database
  - "A single node can fail"
- Exercise chaos on your clients
  - "But we can't restart our database!!??"

Cons

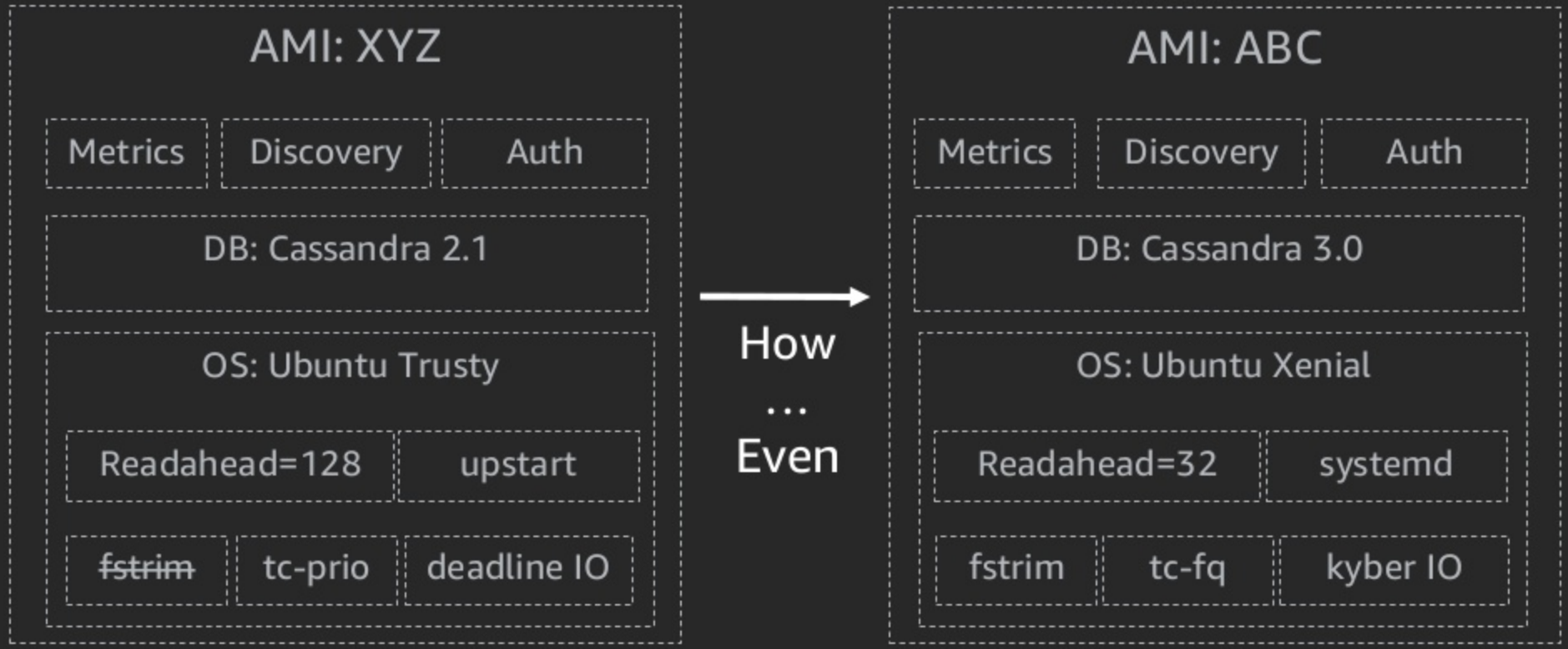
- Pretty complicated
- Moving state is slow
  - Faster with Amazon EBS
- Presents performance challenges
- Requires API backwards compatibility

Which strategies does Netflix use?

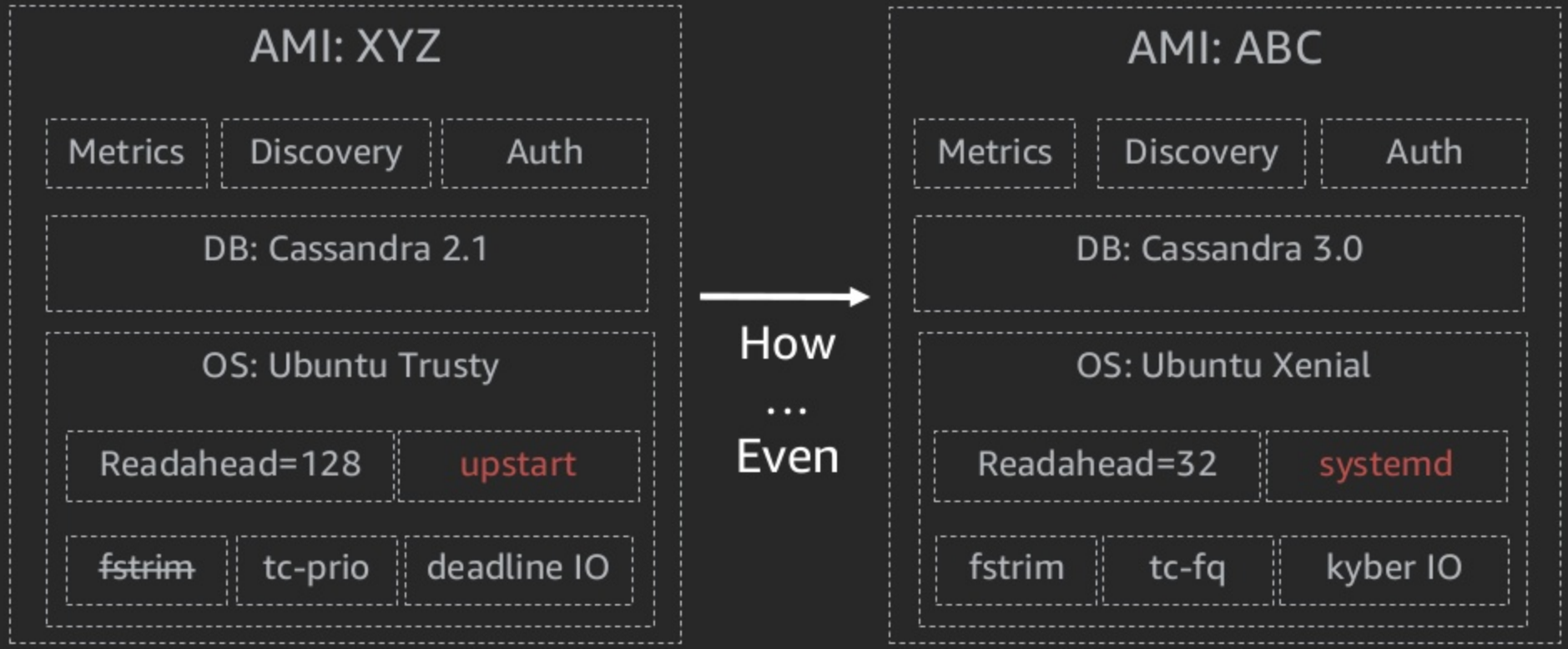
Don’t manage	Mutate	Client	Server
Amazon EBS st1, gp2 Aurora Amazon RDS DynamoDB Amazon S3	All of them	Caches Search engines Logs	Sources of truth

Take it to 11

Problem with mutation: Risk

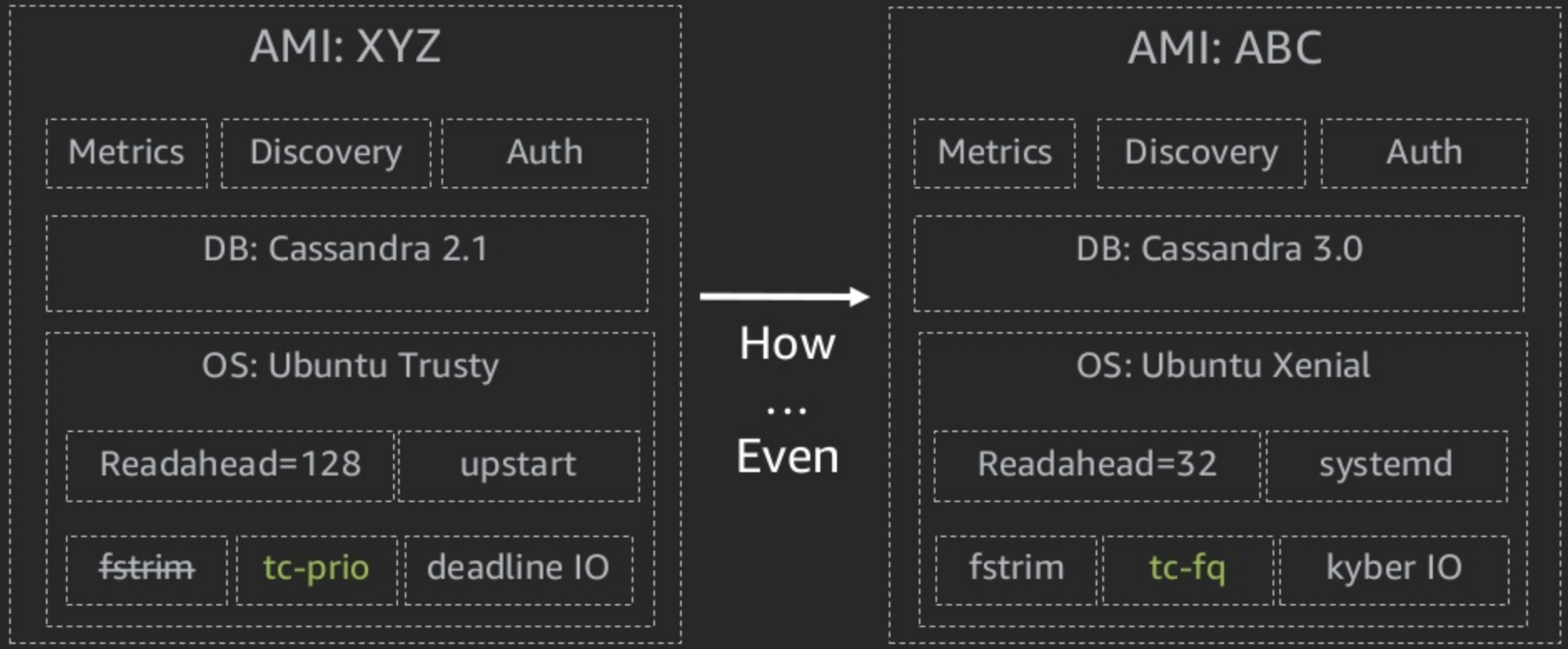


Problem with mutation: Risk

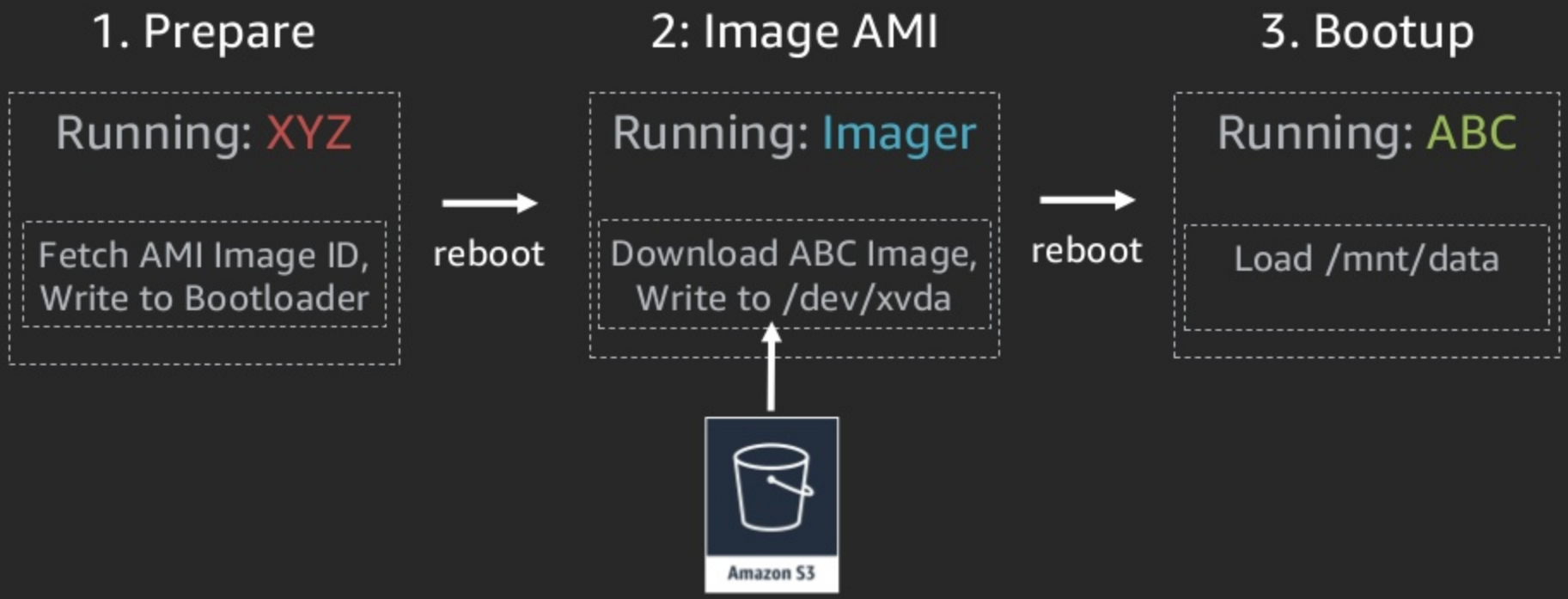


Problem with mutation: Risk

Problem with mutation: Risk



Solution: Image new AMIs directly



AMI live upgrade

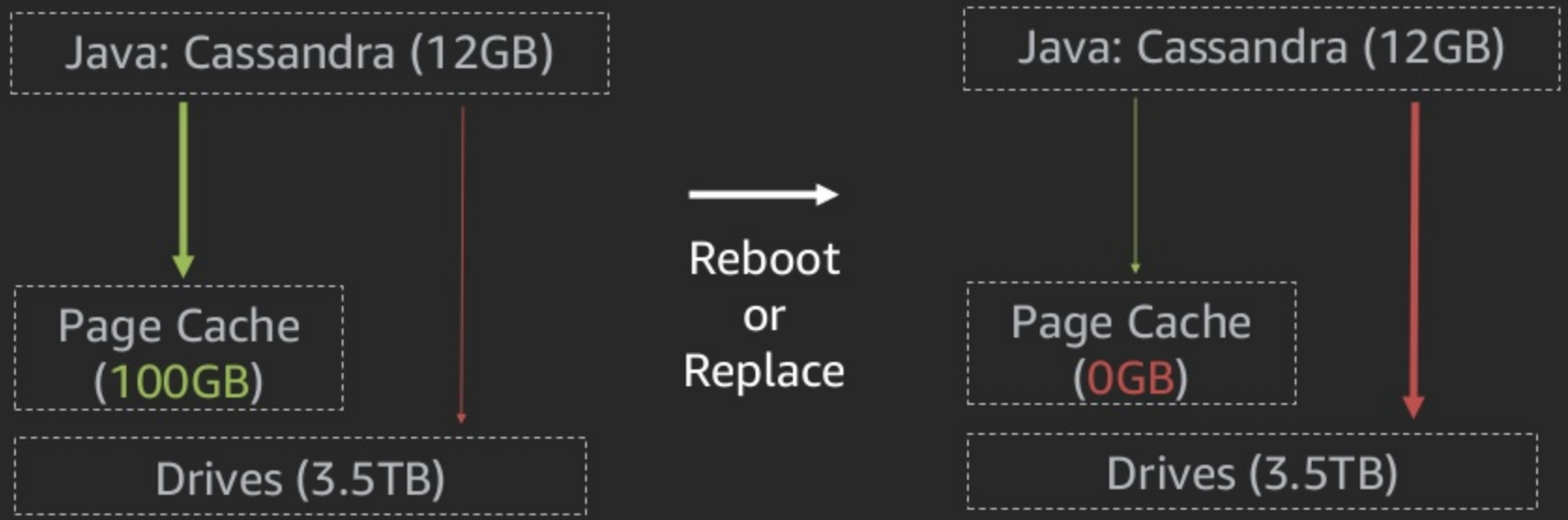
Pros

- Only ever run tested AMIs
- No data transfer or IP change
  - P(data corruption) = low
- Pretty fast (~10 minutes per node)

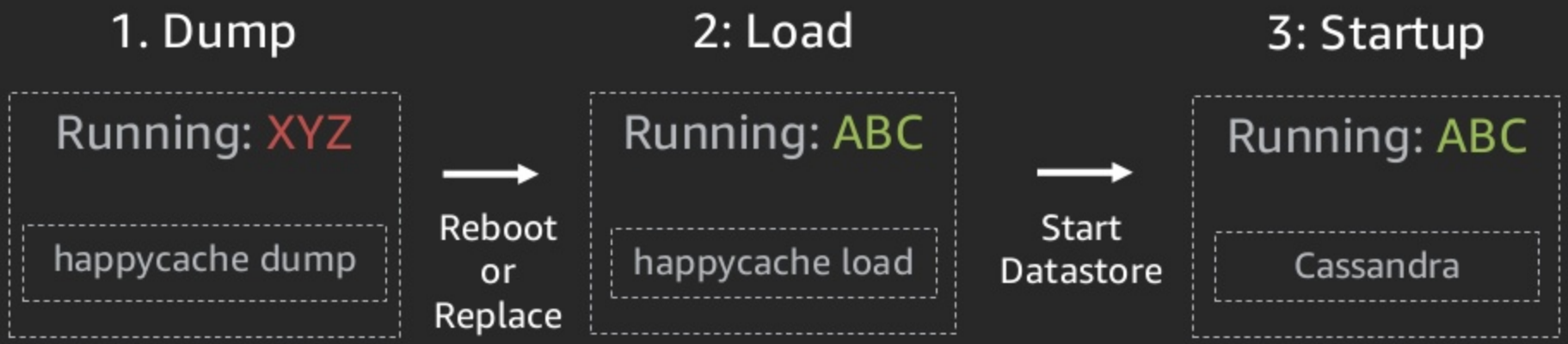
Cons

- Probably better ways
  - Containers
  - Active/Passive flashing
- Doesn't work for instance type upgrades
- Confuses your security team

Problem with state transfer: Page cache

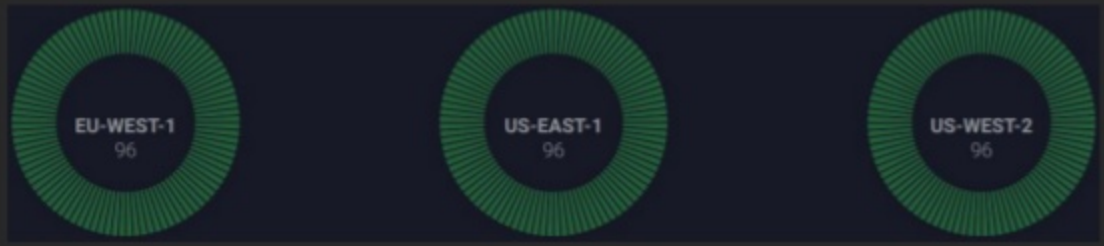


Solution: Warm the caches



https://github.com/hashbrowncipher/happycache

Problem with state transfer: Scale



Instance type: i3.4xlarge
Dataset: 400GB
Time to replace:
~20 minutes per node

= 4 hours for cluster

Instance type: i3.4xlarge
Dataset: 400GB
Time to replace:
~20 minutes per node

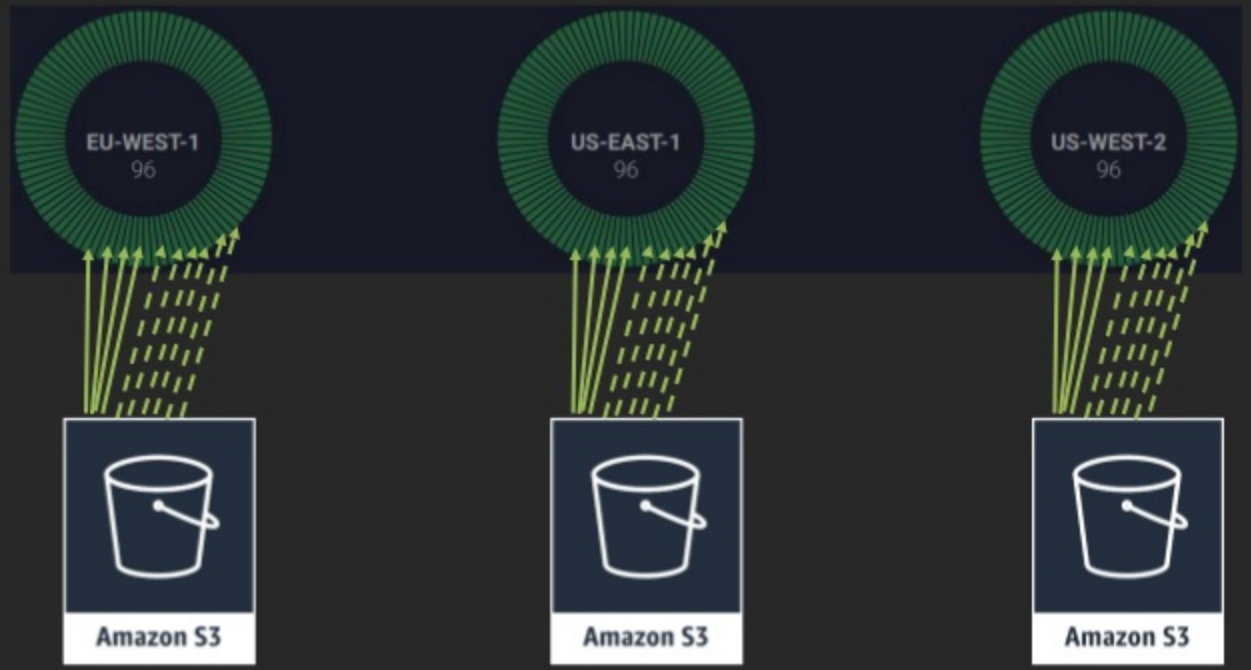
= 4 days for cluster

Solution: Massive parallelism

Instance type: i3.4xlarge
Dataset: 400GB
4 nodes per AZ at once

Time to replace:
~7 minutes per node + 20

= 3.2 hours for cluster!!

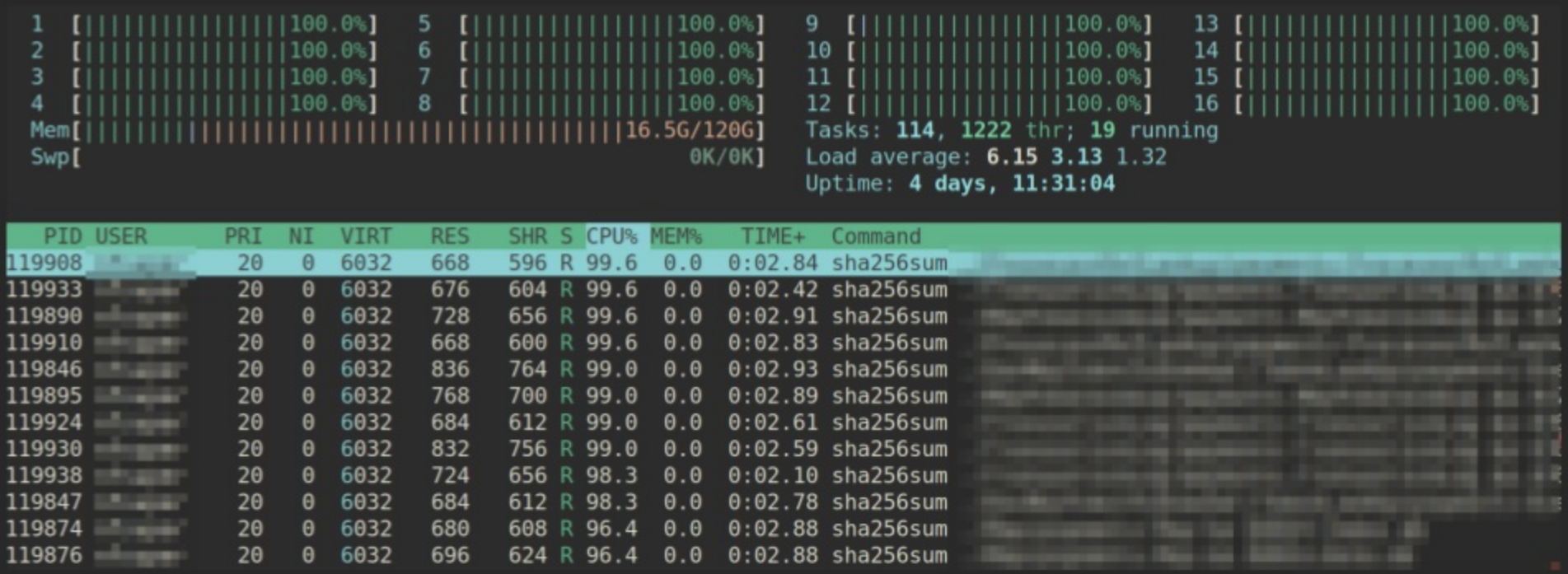


With entire AZ, 50 minutes

Problem with state transfer: Validation is slow

```
$ du –shc .
885G .
885  total

$ find . -type f  | xargs -IX du --block-size=1G X | cut -f 1 | histogram.py -l
   1.0000 -     1.7263 [   691]: ∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎
   1.7263 -     3.1789 [     8]:
   3.1789 -     6.0841 [     2]:
   6.0841 -    11.8944 [     1]:
  11.8944 -    46.7566 [     1]:
  46.7566 -   372.1369 [     1]:
 372.1369 -   744.0000 [     1]:

$ time find . -type f | xargs -IX -P 16 sha256sum X > /dev/null

real    1h22m27.694s
user    1h36m44.131s
sys     4m40.120s
```

Problem with state transfer: Validation is slow



Problem with state transfer: Validation is slow

Device:	tps	MB_read/s	MB_wrtn/s	MB_read	MB_wrtn
xvda	2.00	0.00	0.01	0	0
nvme1n1	4623.00	144.00	0.00	287	0
nvme0n1	4638.00	144.35			
md0	9312.50	409.17	0.00	818	0

410 MBps ... very bad

Device:	tps	MB_read/s	MB_wrtn/s	MB_read	MB_wrtn
xvda	2.00	0.00	0.01	0	0
nvme1n1	2508.50	78.37	0.04	156	0
nvme0n1	2511.00	78.34			0
md0	5019.50	166.48	0.11	332	0

166 MBps single threaded ... Sad

Solution: Use a better hash (xxHash)

```
$ instance-type
i3.4xlarge

$ du –shc .
885G .
885  total

$ time find . -type f | xargs -IX -P 16 ~/xxHash/xxh64sum --quiet X > /dev/null

real    12m10.694s
user    1m44.548s
sys     7m17.611s
```

https://github.com/Cyan4973/xxHash

Solution: xxHash

Device:	tps	MB_read/s	MB_wrtn/s	MB_read	MB_wrtn
xvda	4.50	0.01	0.03	0	0
nvme1n1	30558.00	1908.87	0.05	3817	0
nvme0n1	30567.50	1908.78	0.05	3817	0
md0	61126.00	3817.69	0.07	7635	0

4 GBps is more like it

Device:	tps	MB_read/s	MB_wrtn/s	MB_read	MB_wrtn
xvda	2.00	0.00	0.01	0	0
nvme1n1	8937.50	558.53	0.00	1117	0
nvme0n1	8938.00	558.55	0.00	1117	0
md0	17875.50	1117.09	0.00	2234	0

1.1 GBps single threaded

Manage state Netflix style

Don’t manage	Mutate	Client	Server
This is a very viable option Often the right choice	Make it safe Avoid drift	Careful with consistency Coordination is key	Safest option Can be fast as well!

Thank you!

Joey Lynch
josephl@netflix.com

Please complete the session survey in the mobile app.