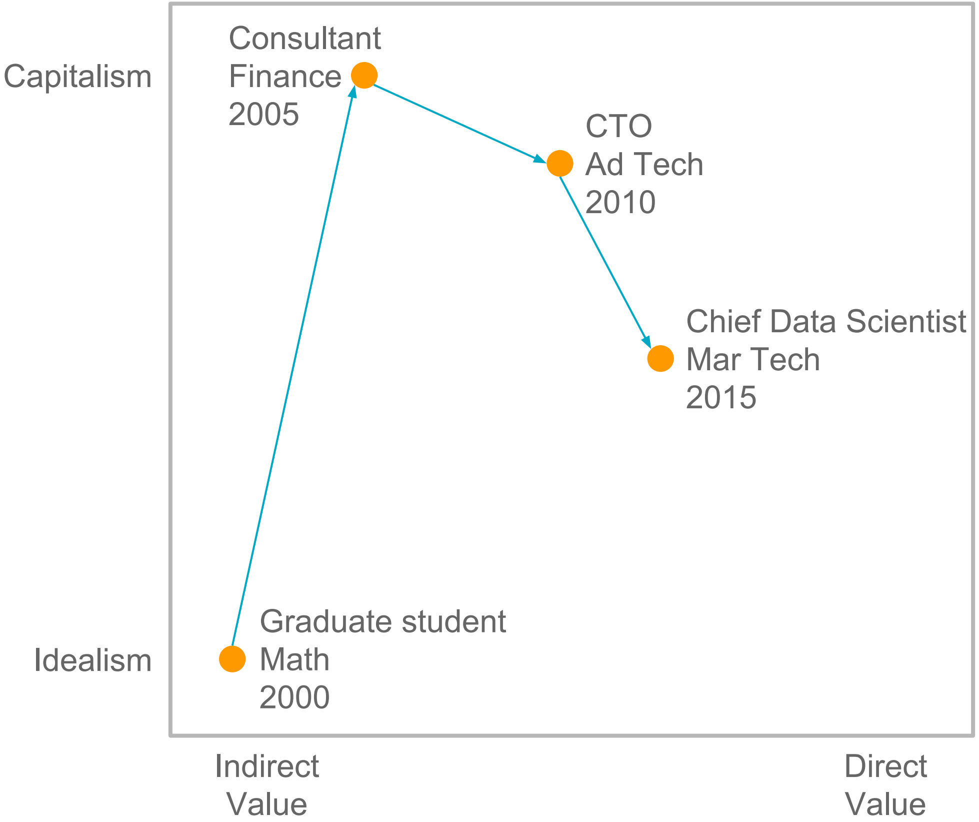
SAILTHRU

Building a ML System to Predict User Behavior on Mesos

Agenda:

- **Background** on me, Sailthru & Sightlines (mercifully short)
- **Cost effective** resources in the AWS cloud
- **Efficient(ish)** application design
- **Easy** maintenance and evolution
- **Lessons** learned
- **New Innovation**

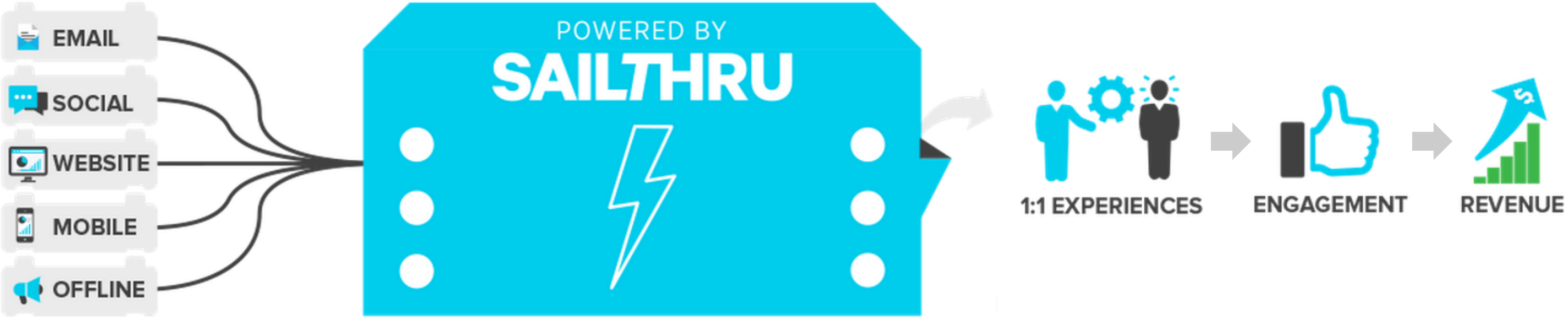
@jeremystan


Capitalism
Idealism
Consultant
Finance
2005
CTO
Ad Tech
2010
Chief Data Scientist
Mar Tech
2015
Graduate student
Math
2000
Indirect
Value
Direct
Value

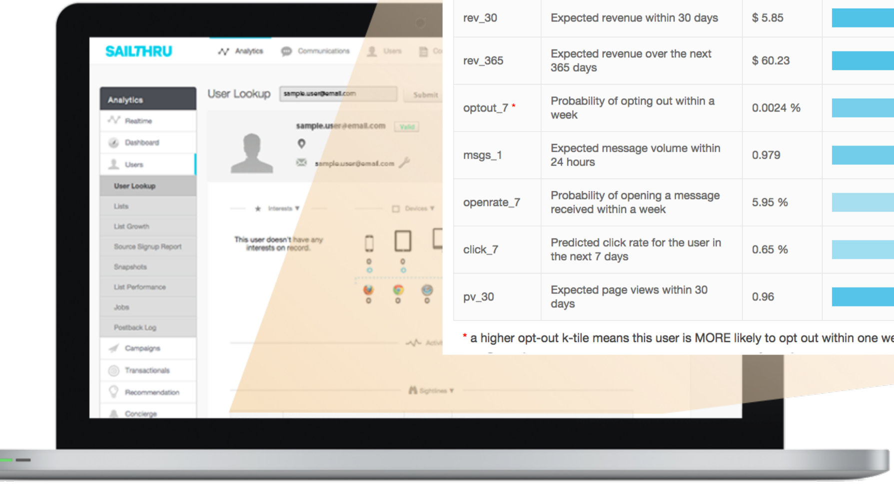
Sailthru

Powering More Than 400 Ecommerce & Media Brands

Mashable · The Economist · BIRCHBOX · ALEX AND ANI · FRANK & OAK



Sightlines



Sightlines ▼

Sightline	Prediction	Number	k-tile
purchase_1	Probability of making any purchase within 24 hours	0.066 %	974
purchase_7	Probability of making any purchase within a week	0.596 %	972
purchase_30	Probability of making any purchase within 30 days	3.314 %	972
aov_7	Expected order value if a purchase occurs within a week	$ 131.91	991
rev_30	Expected revenue within 30 days	$ 5.85	978
rev_365	Expected revenue over the next 365 days	$ 60.23	973
optout_7 *	Probability of opting out within a week	0.0024 %	766
msgs_1	Expected message volume within 24 hours	0.979	791
openrate_7	Probability of opening a message received within a week	5.95 %	494
click_7	Predicted click rate for the user in the next 7 days	0.65 %	529
pv_30	Expected page views within 30 days	0.96	955

* a higher opt-out k-tile means this user is MORE likely to opt out within one week.

Analytics
- Segmentation
- Forecasting

Personalization
- Recommendations
- Discounting

Optimization
- Frequency
- Channel

Requirements

1. ~5 million users per client
2. JSON formatted user data, siloed across clients
3. Predict varying outcomes
   normal, poisson, binomial, quantile, ...
4. Update models & predictions daily
5. Only really care about predictive performance
6. **Scale to 1,000+ clients**

Our Cost Effective Scaling Strategy

1. Get really cheap computing power **10x**
2. Make it work really, really hard **3x**
3. Optimize apps for ease of evolution **0.6x** =

JSON to Features	GBM in Memory
0.2x	1 x
Half our processing	Half our processing

4. Setup identical A/B environments **0.5x**

= 9x

Iterate aggressively based on data:

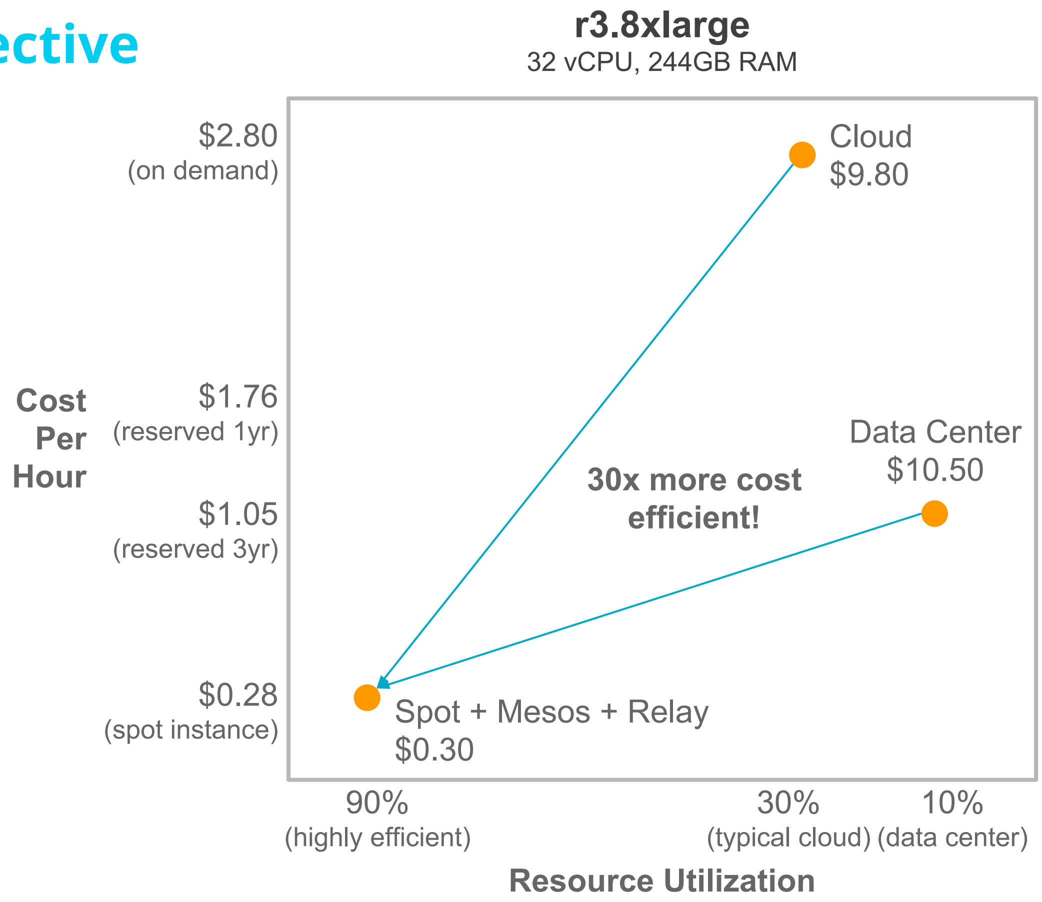
✓ Features

✓ Efficiency

✓ Scale

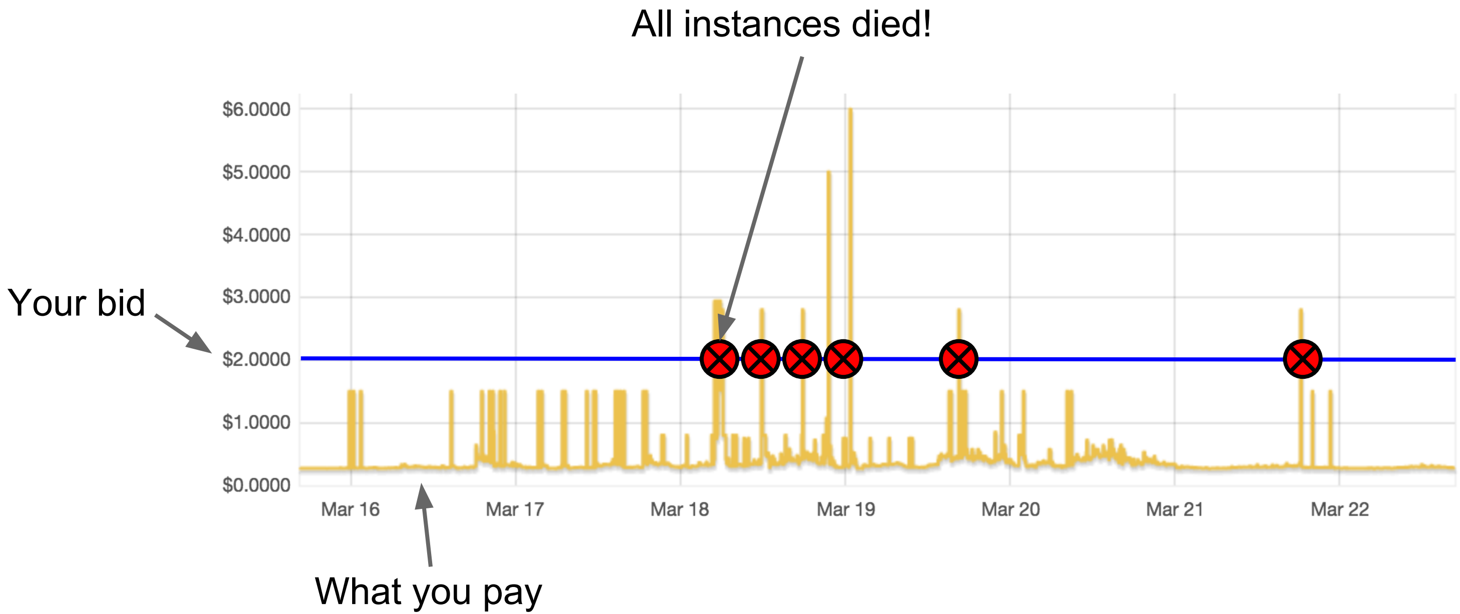
Cost Effective Resources in the AWS Cloud

Cost Effective



($10.50 = $1.05 / 10%)

AWS Spot Instances



Mesos

Cluster: mesos-dev
Server: 172.24.1.137:5050
Version: 0.22.1
Built: 3 months ago by *root*
Started: 11 hours ago
Elected: 11 hours ago

LOG

Slaves

Activated	146
Deactivated	0

Tasks

Staged	261,030
Started	0
Finished	178,048
Killed	32,945
Failed	35,029
Lost	2,554

Resources

	CPUs	Mem
Total	3,410	25450.2 GB
Used	2,567.11	23425.9 GB
Offered	0	0 B
Idle	842.890	2024.3 GB

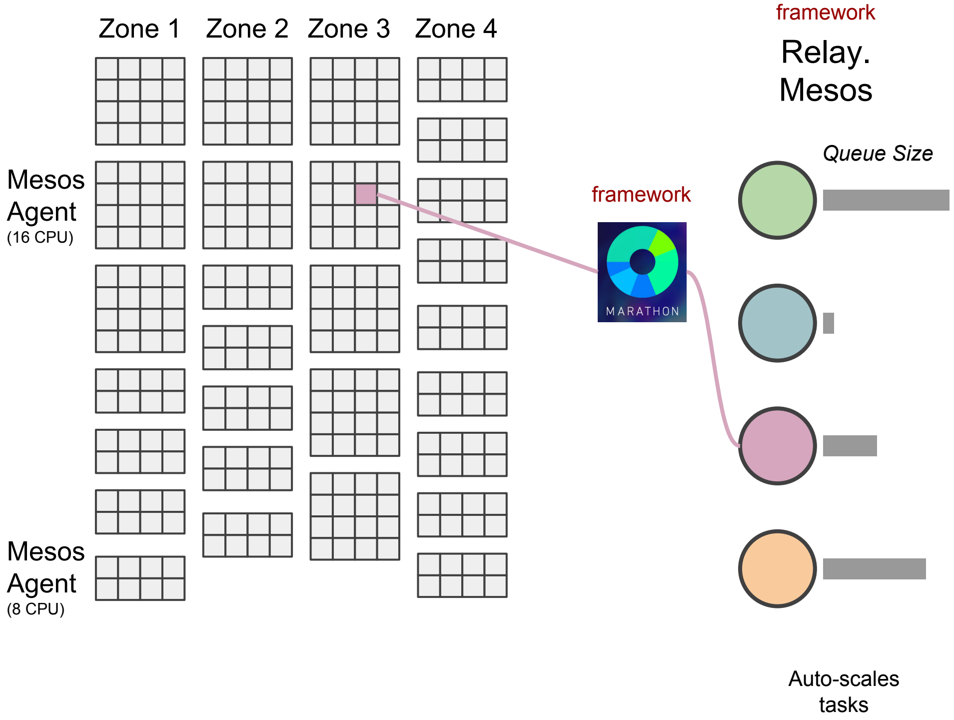
146 “agents”
4 availability zones
2 instance types

3,410 CPUs
25TB of RAM

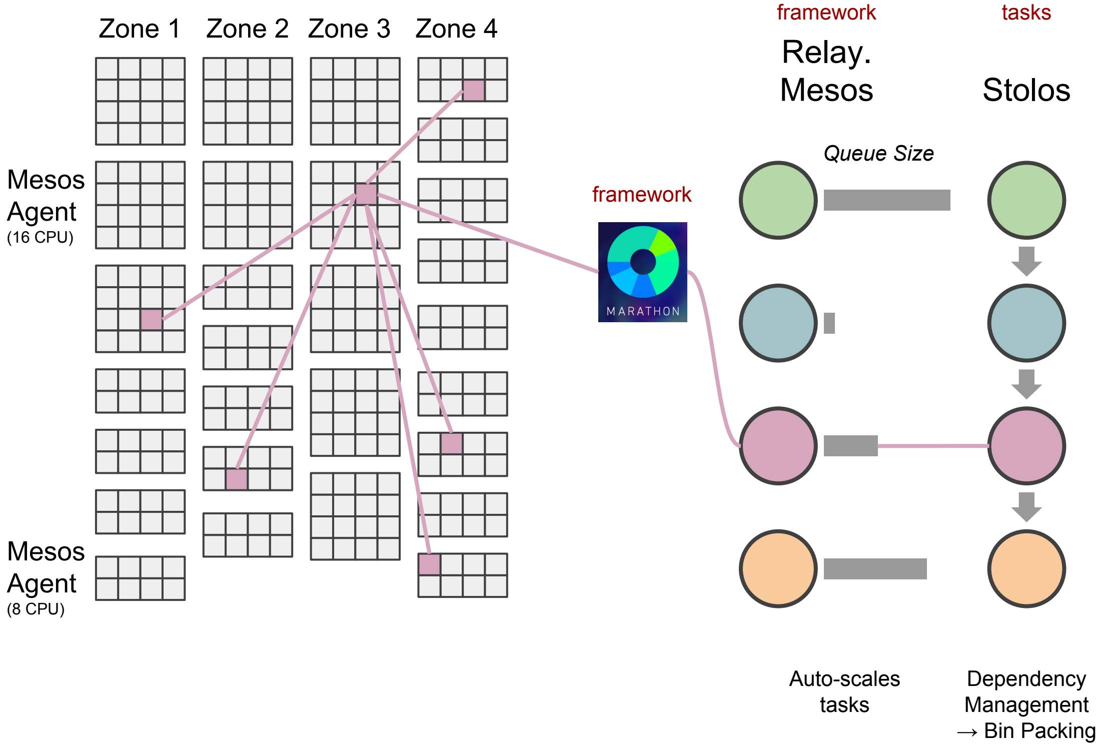
75% CPU utilized
92% RAM utilized

$30 per hour
$260k per year

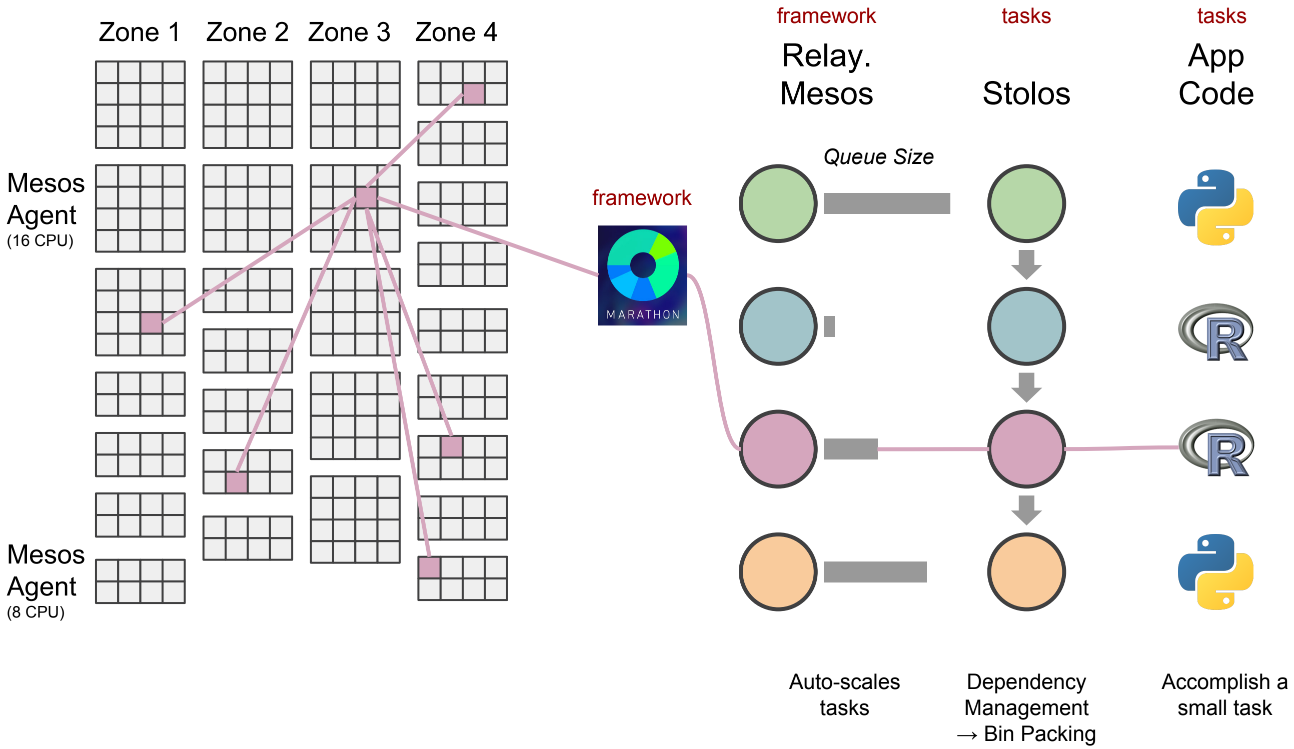
How we use Mesos



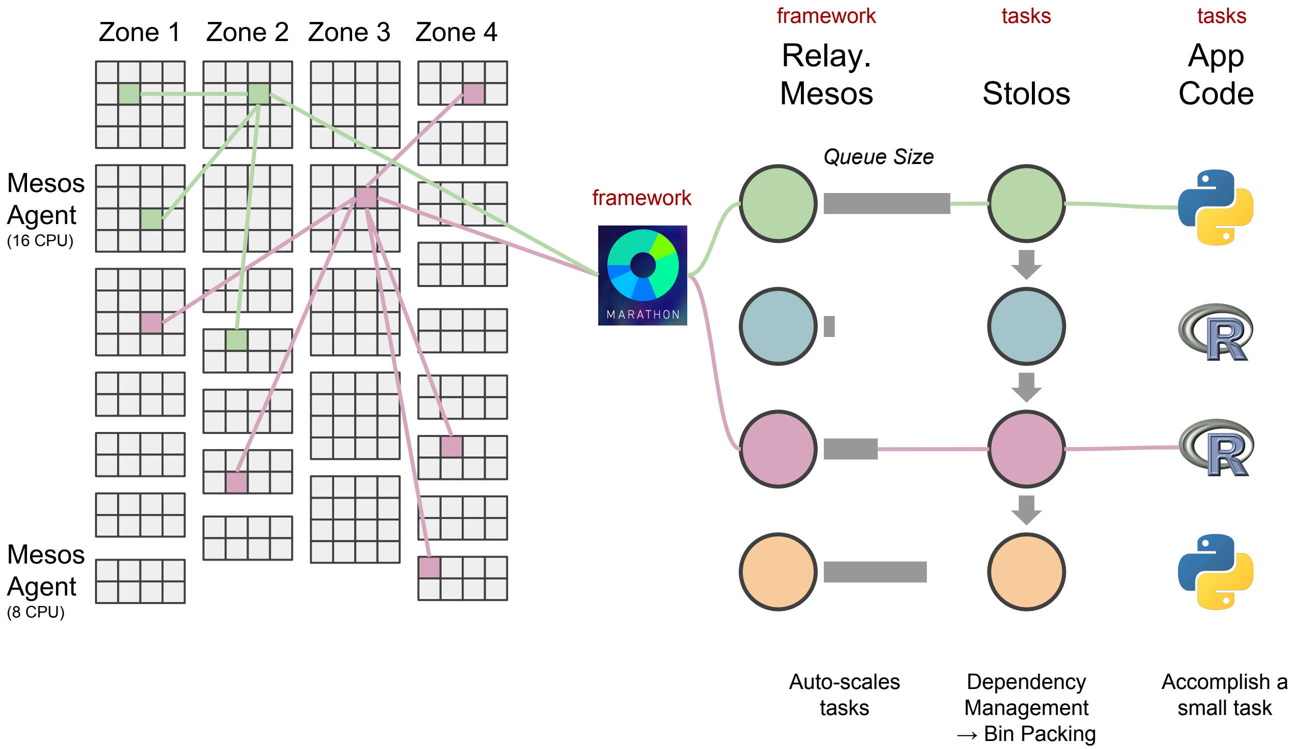
How we use Mesos



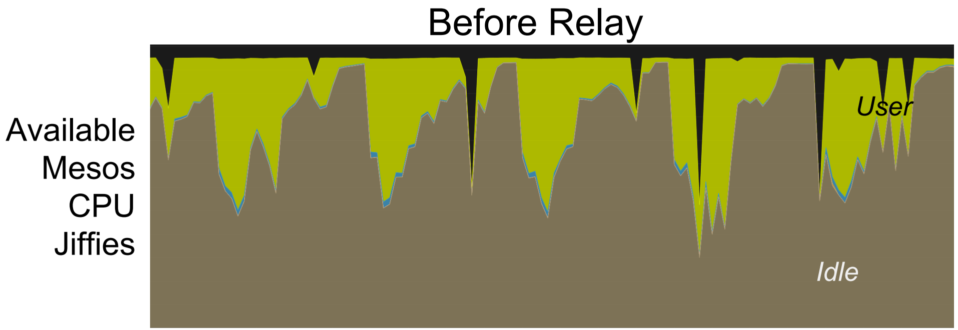
How we use Mesos

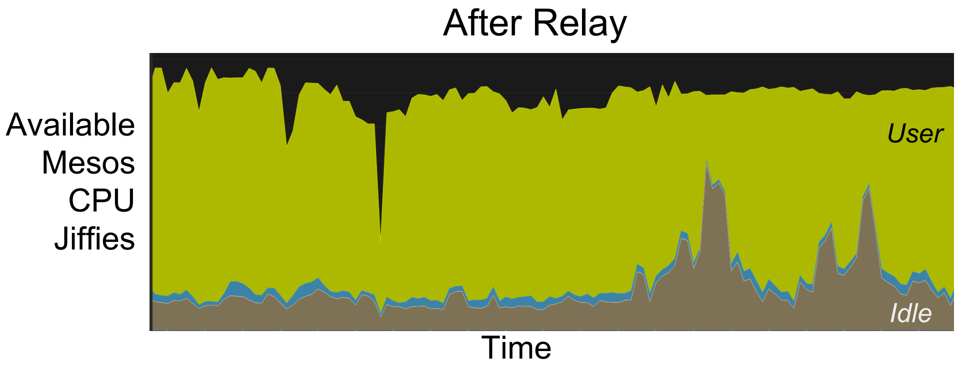


How we use Mesos



Mesos + Relay



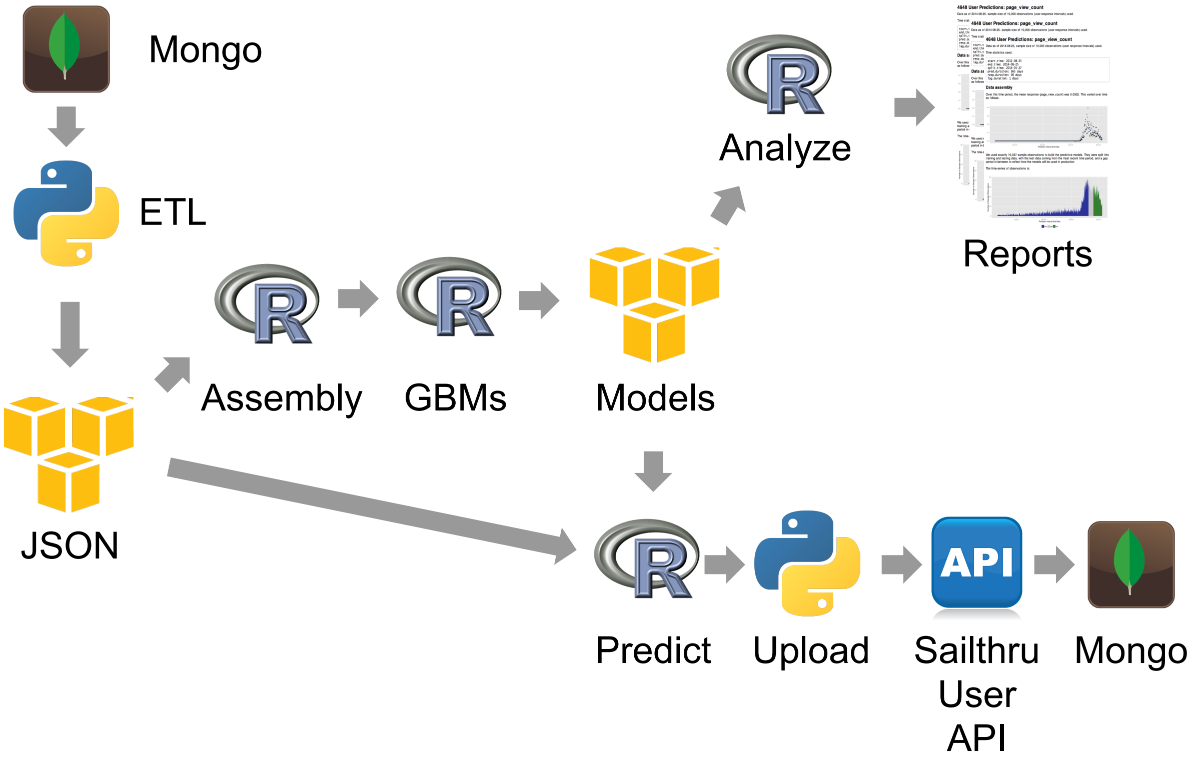


Relay.Mesos
Auto-scaler for distributed applications
github.com/sailthru/relay.mesos

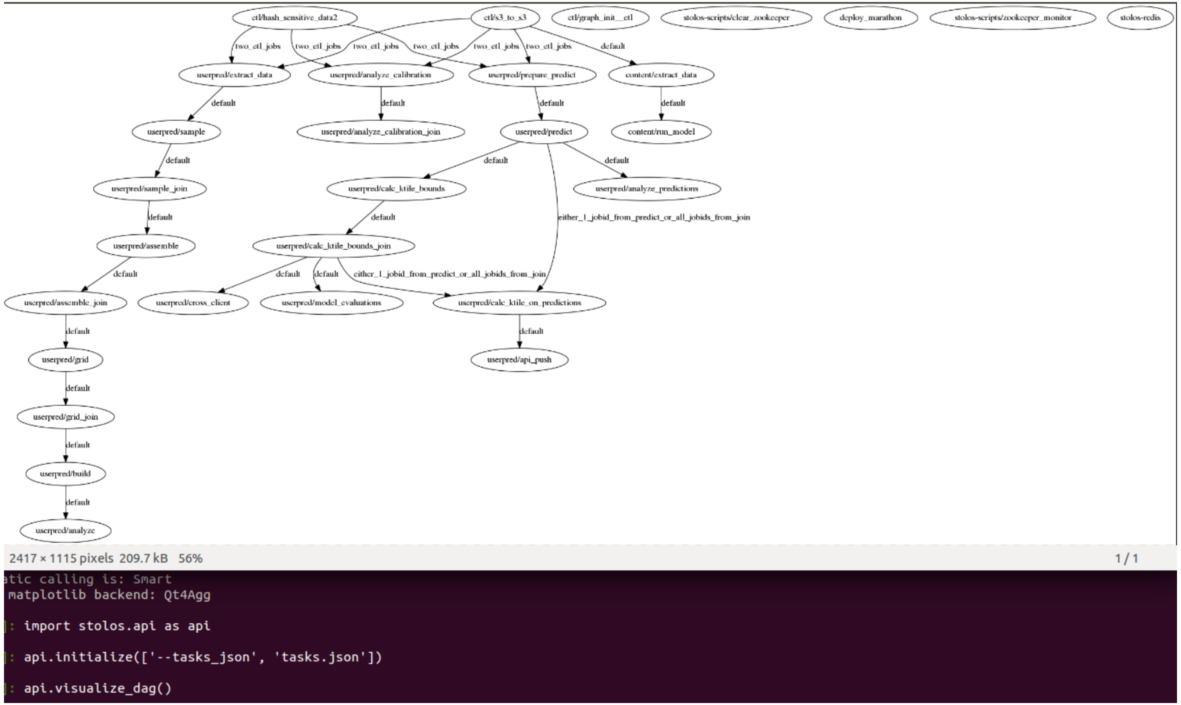
- Allocates resources based on queue size
- Wraps applications inside Mesos agents
- Can significantly improve cluster utilization

Efficient(ish) Application Design

Application Pipeline (simplified)



Application Pipeline (actual)



Actually much more complex

- ~1,000 clients
- ~10 models
- ~30 steps
- ~100 sub-tasks

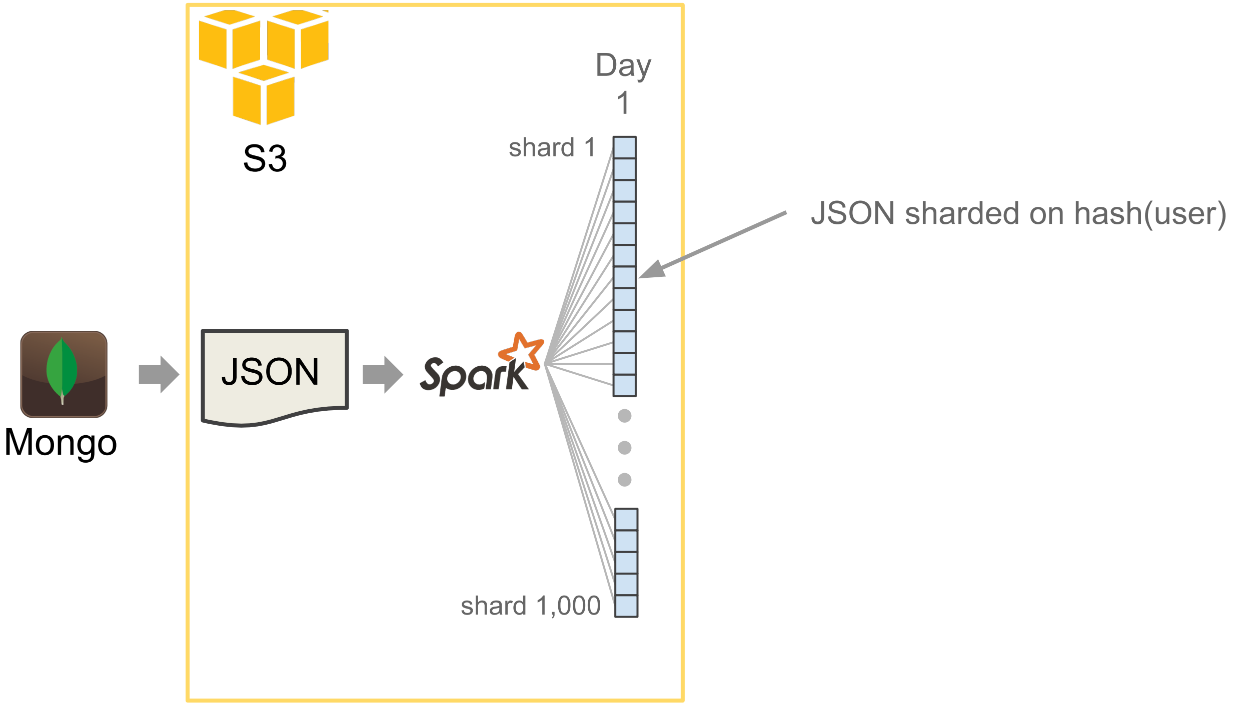
Stolos

Distributed task dependency manager

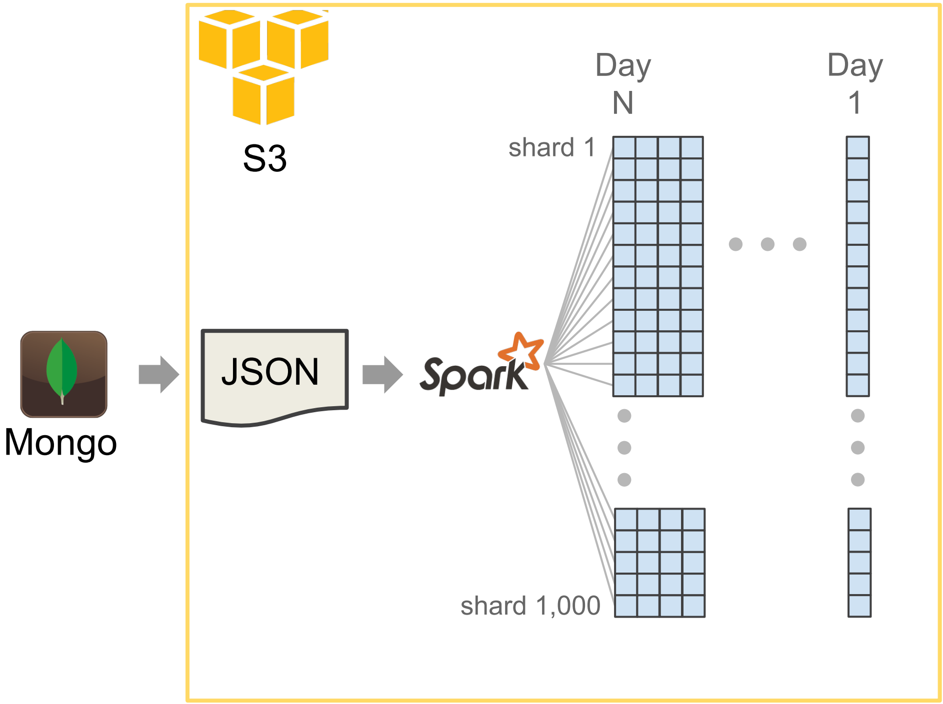
github.com/sailthru/stolos

- Directed acyclic graph
- Parameterizable templates
- Handles queueing
- Ensures idempotent

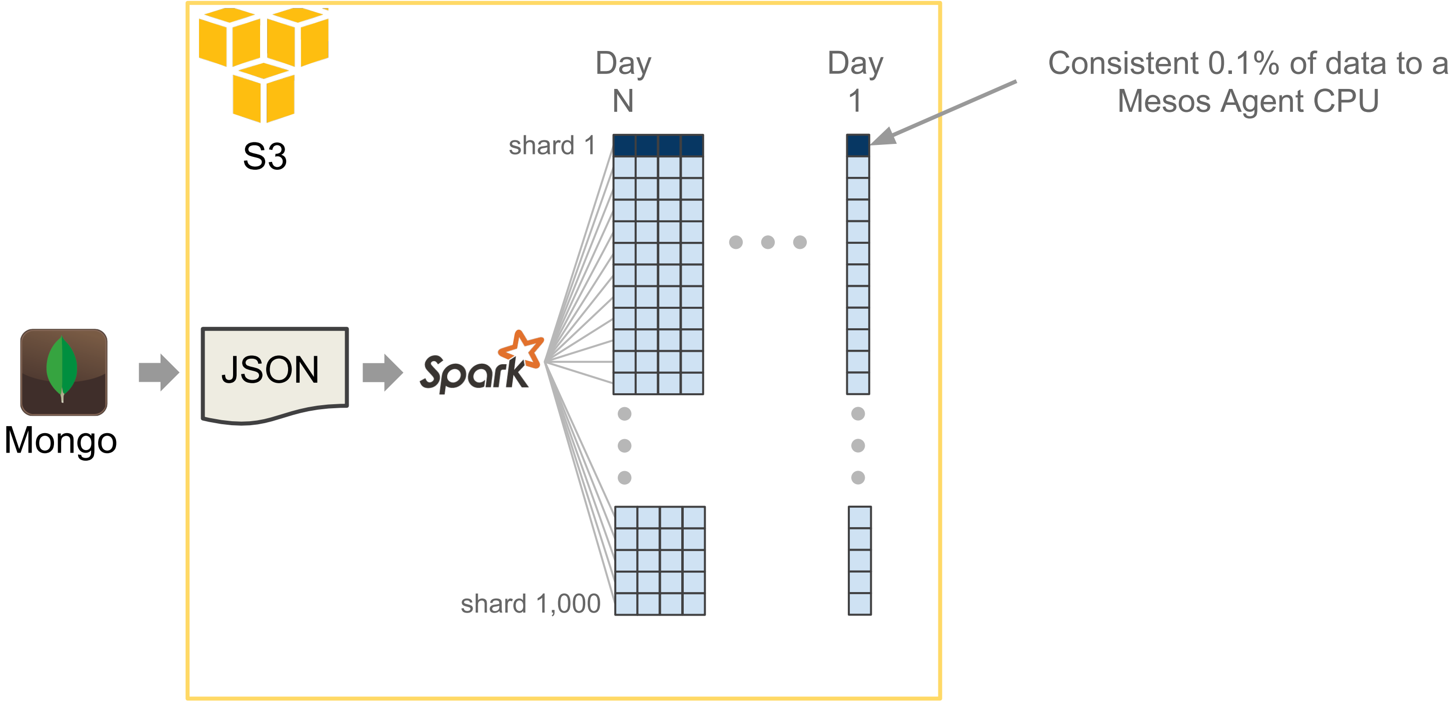
Sampling Strategy



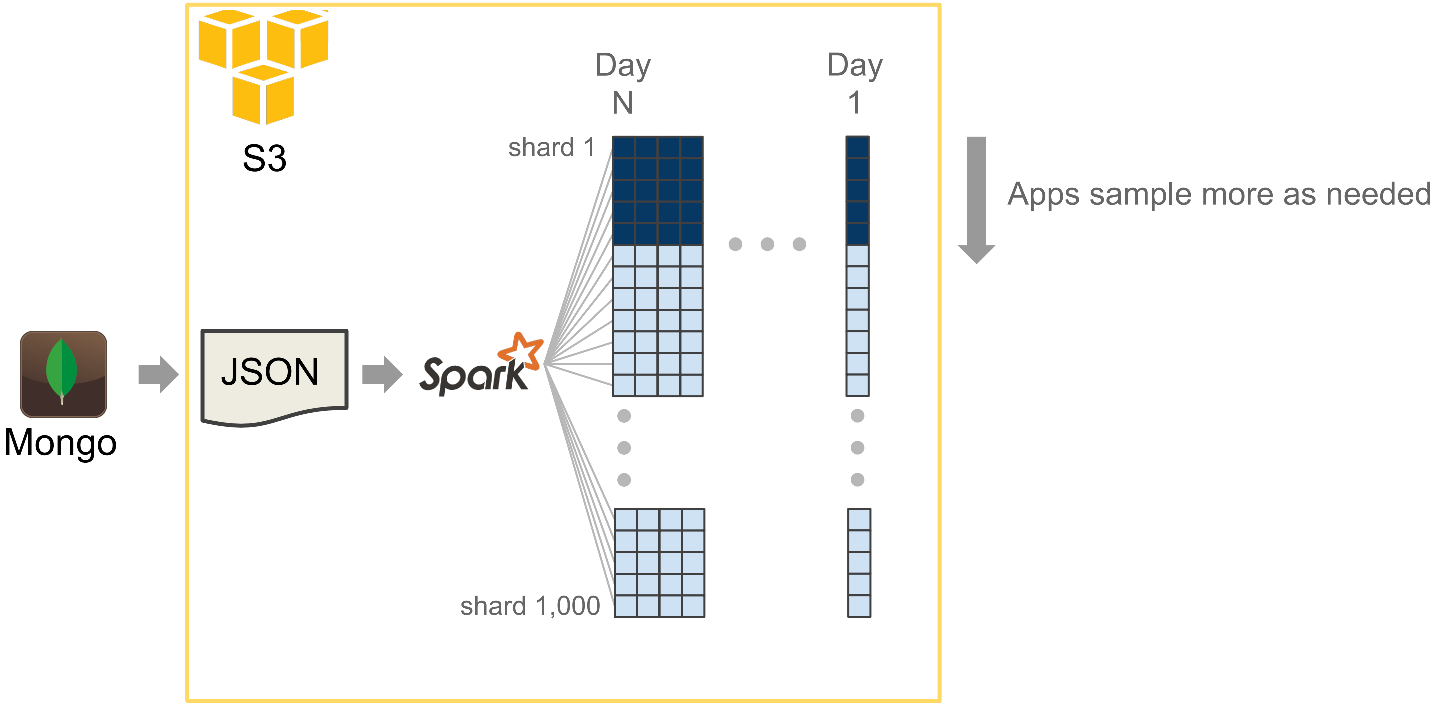
Sampling Strategy



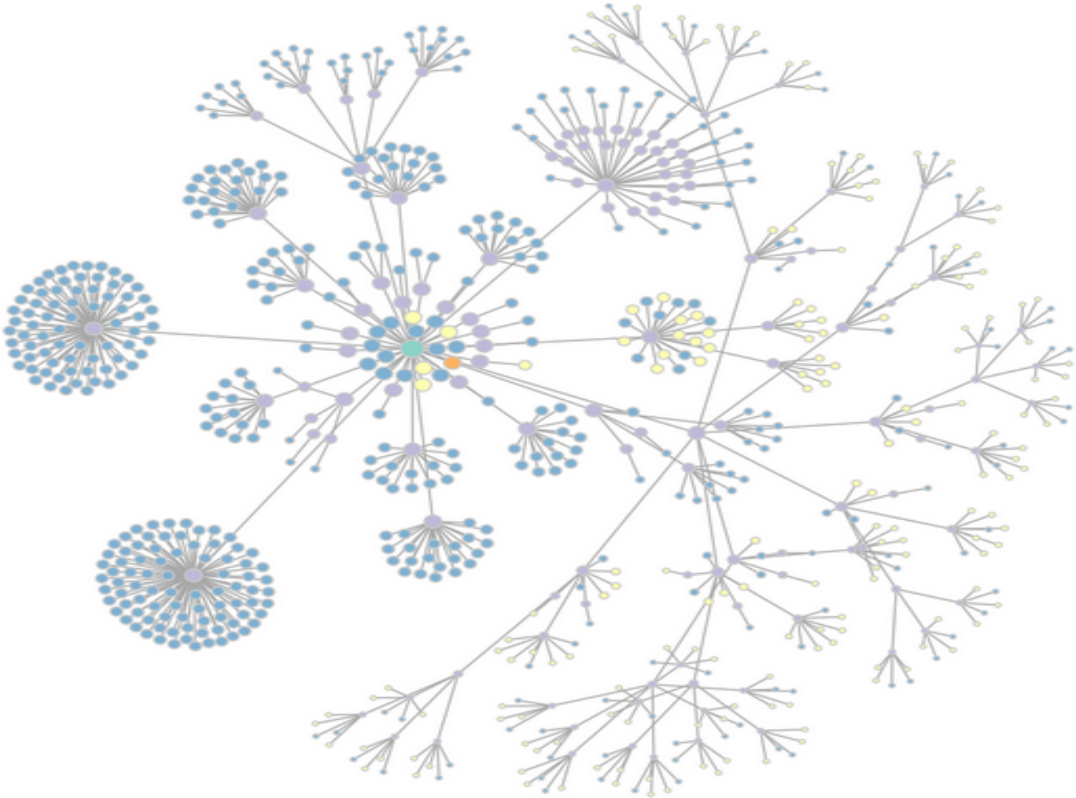
Sampling Strategy



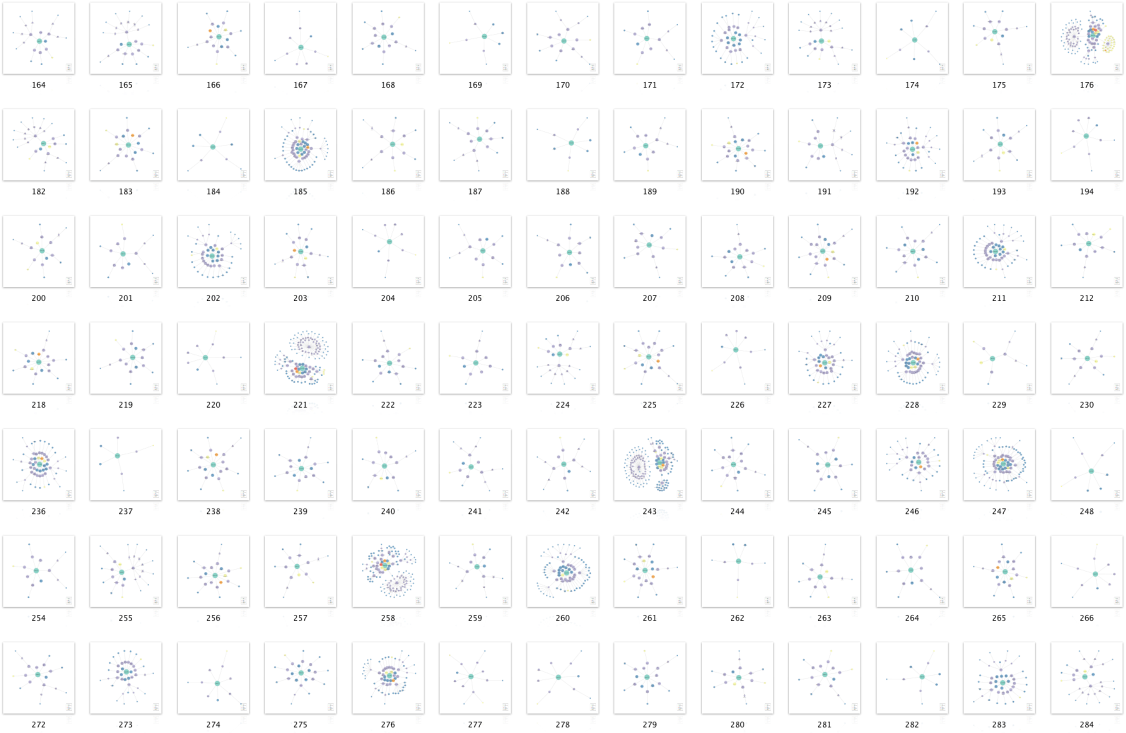
Sampling Strategy

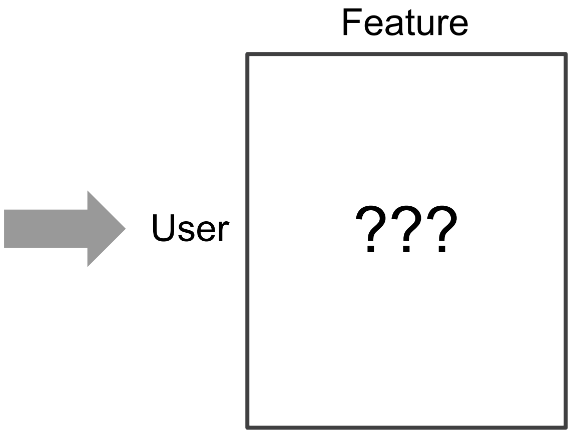


User Profile JSON Data

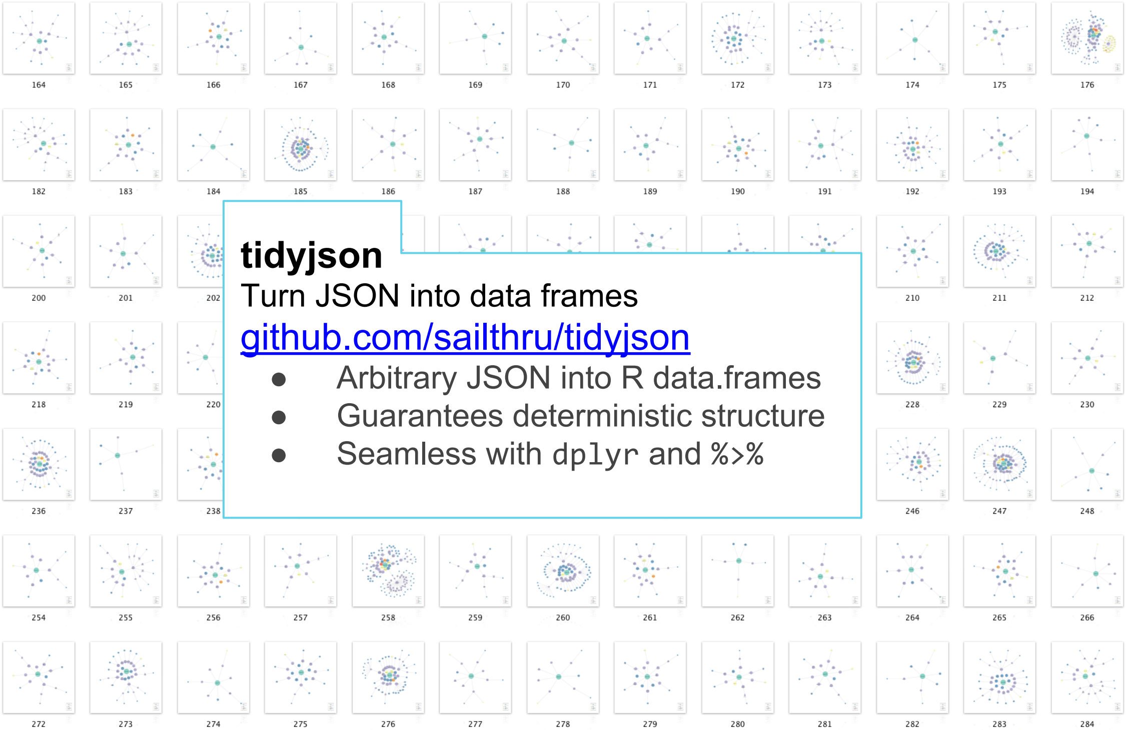


Each User Radically Different





Each User Radically Different

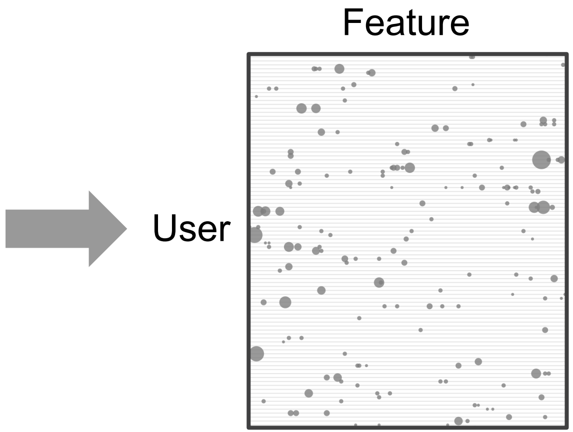


tidyjson

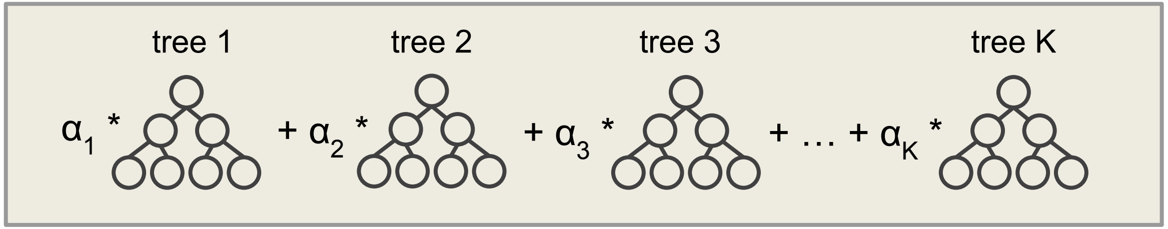
Turn JSON into data frames

github.com/sailthru/tidyjson

- Arbitrary JSON into R data.frames
- Guarantees deterministic structure
- Seamless with `dplyr` and `%>%`



What is a Gradient Boosting Machine? (GBM)

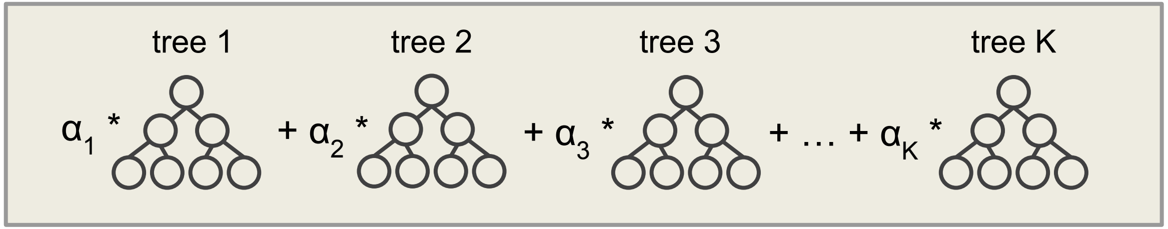


1. Build a simple decision tree to predict your response
2. Evaluate it’s performance, and trust it a small amount
3. Build another decision tree to correct it’s mistakes
4. Iterate to some fixed number of trees

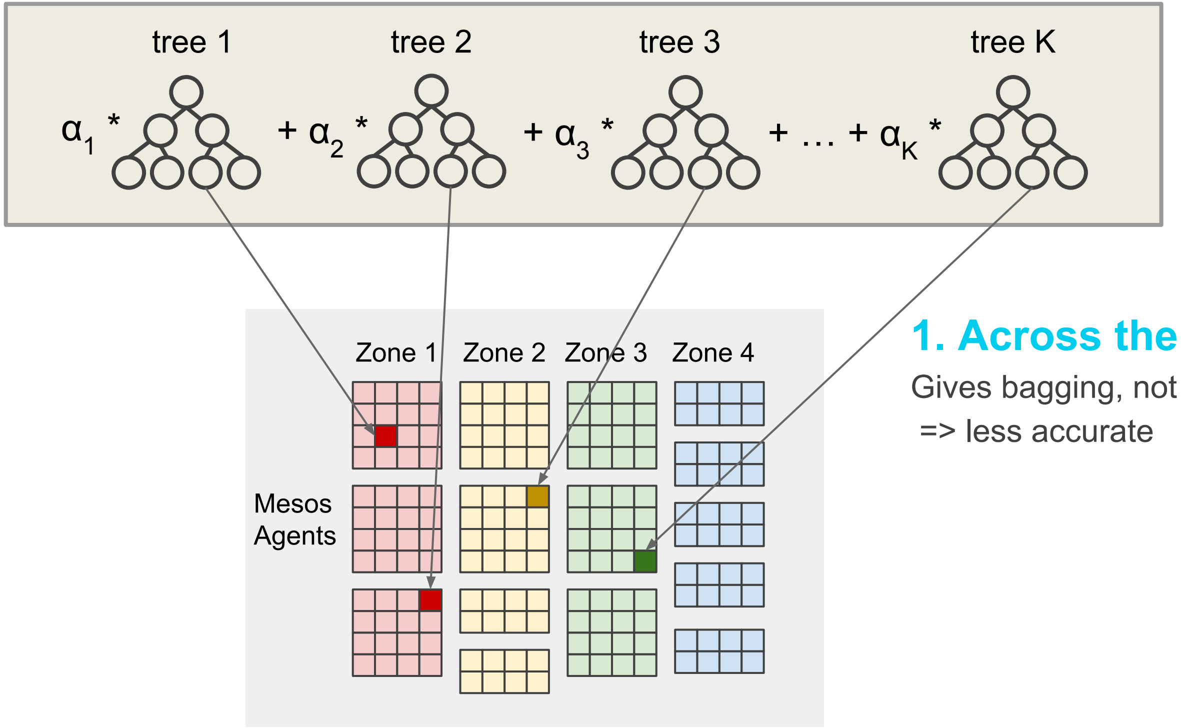
Why GBMs?

- **Predict varying outcomes**
  normal, poisson, binomial, quantile, …
- **Flexible enough to capture non-linearity & complex interactions**
  no need to feature engineer for each client
- **Minimal number of hyper-parameters**
  depth, shrinkage, number of trees
- **Robust to missing values**
  no need to impute

Distributing a GBM



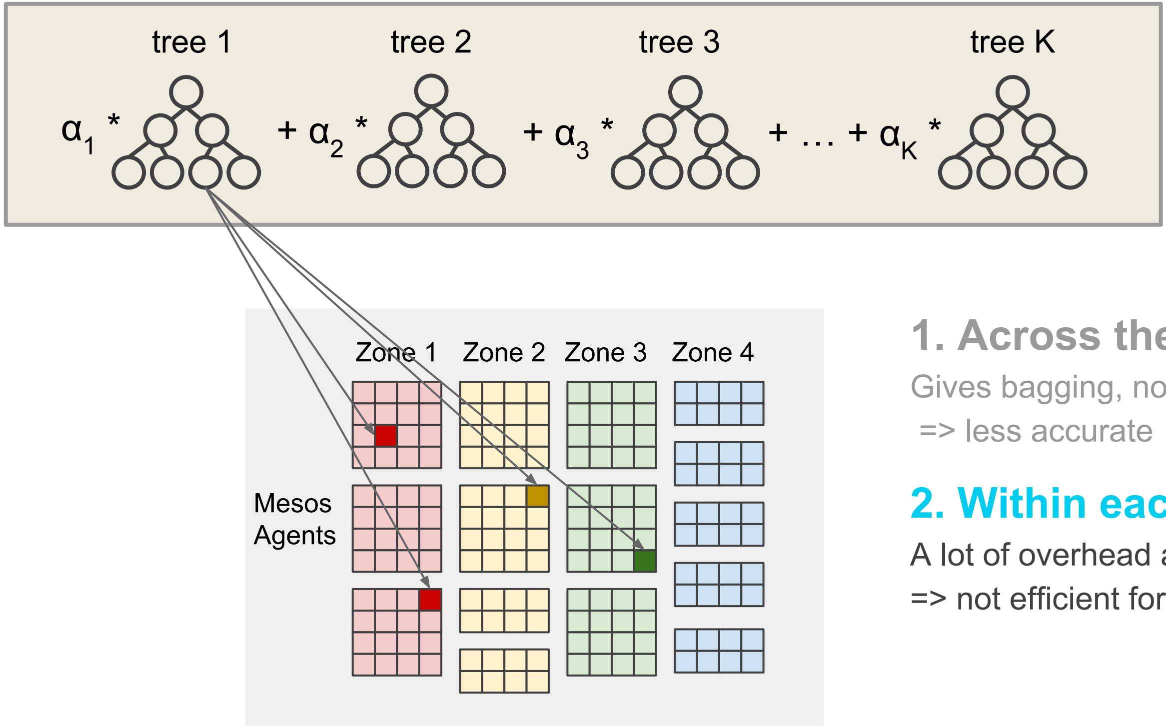
Distributing a GBM



1. Across the sum

Gives bagging, not boosting (iterative)
=> less accurate

Distributing a GBM



1. Across the sum

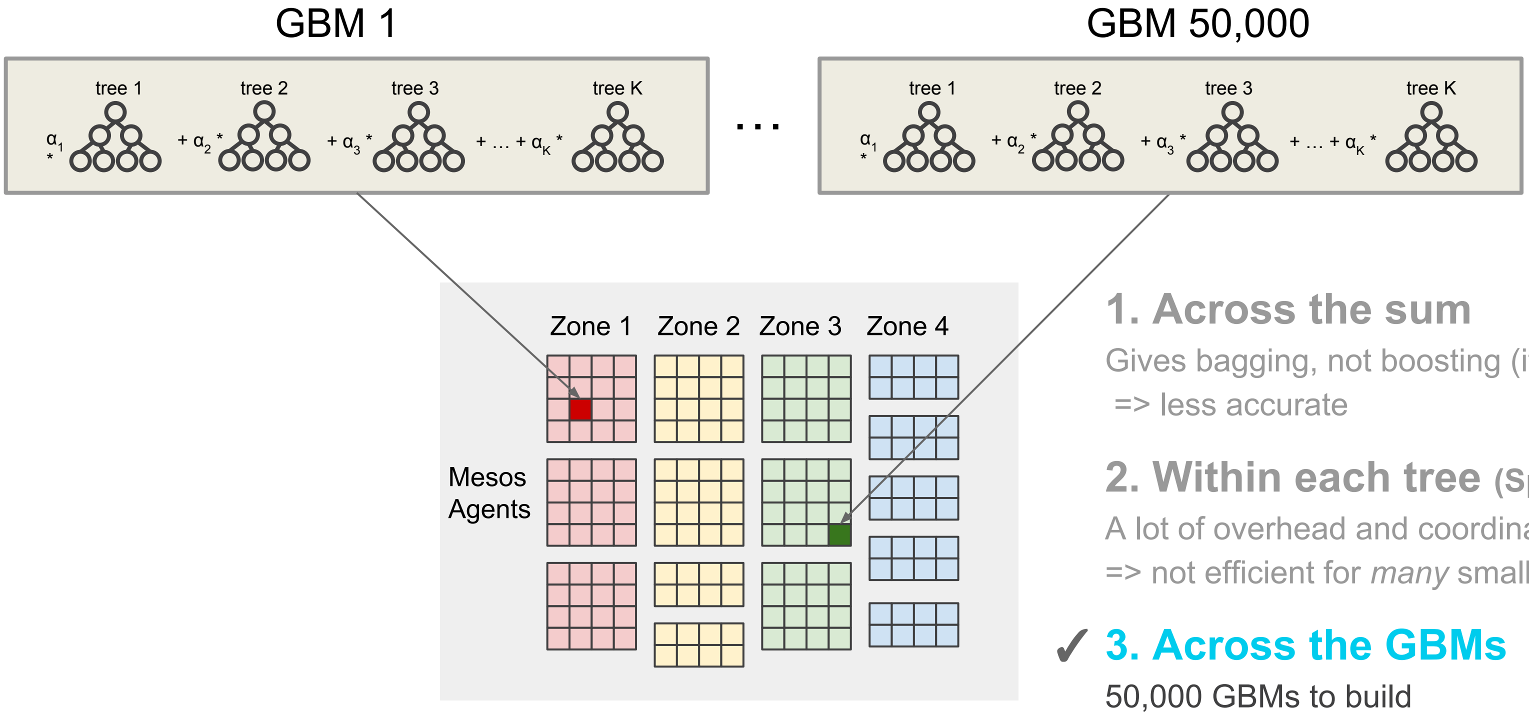
Gives bagging, not boosting (iterative)
=> less accurate

2. Within each tree (Spark MLLib, H20)

A lot of overhead and coordination
=> not efficient for *many* small GBMs

Distributing a GBM

*50,000 = 1,000 clients * 10 models * 5-fold CV*



1. Across the sum

Gives bagging, not boosting (iterative)
=> less accurate

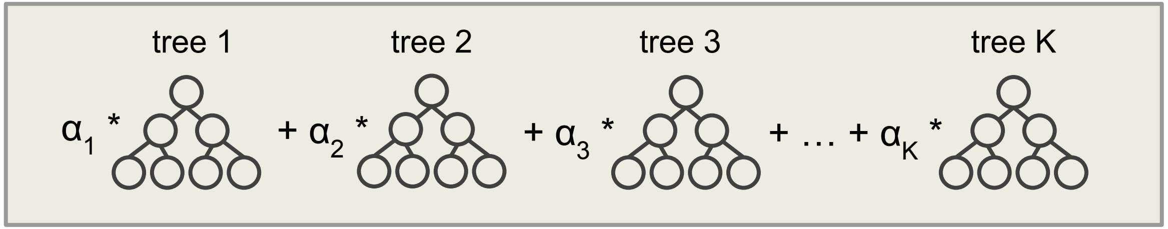
2. Within each tree (Spark MLLib, H20)

A lot of overhead and coordination
=> not efficient for *many* small GBMs

✓ 3. Across the GBMs

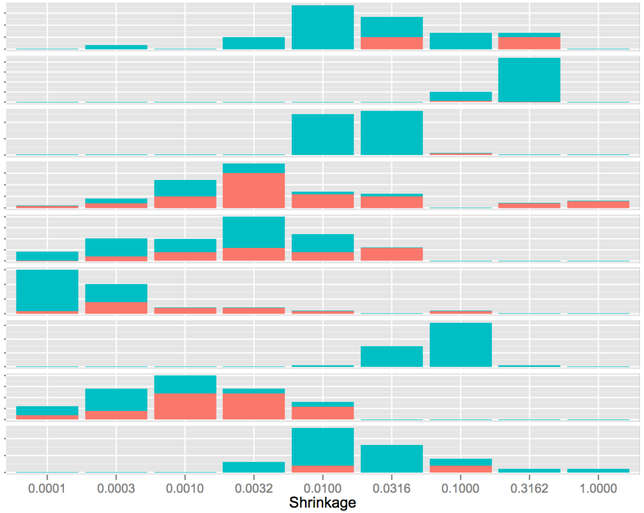
50,000 GBMs to build
=> each can be built independently

Grid Search



For each client & model:

1. Grid search over:
   a. Depth: size of trees
   b. Shrinkage: λ "learning rate" for $\{\alpha_i\}$
2. Cross-validate for optimal # of trees



Easy Maintenance & Evolution

Tools Used

Cluster

AWS Spot
Compute

Asgard
Auto Scaling

Mesos
Sharing

State

AWS S3
Batch

Zookeeper
Coordination

Maintenance

ELK
Log Mgmt

Librato
Monitoring

Sensu
Alerting

Configuration

Consul
Discovery

Chef
Automation

Frameworks

Spark
Map Reduce

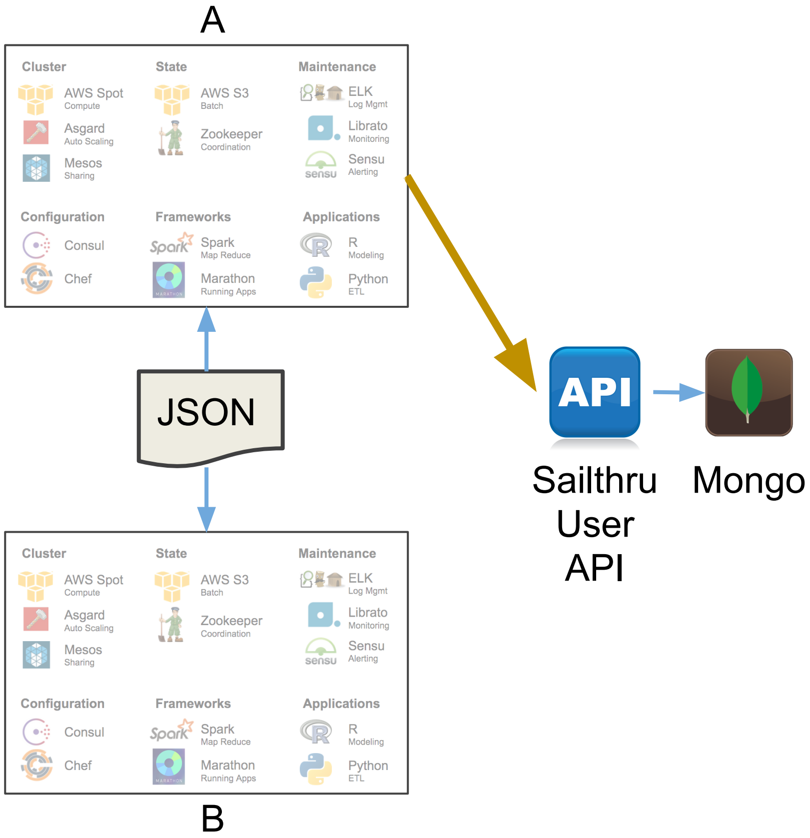
Marathon
Running Apps

Applications

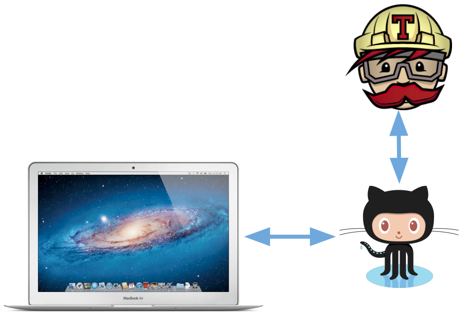
R
Modeling

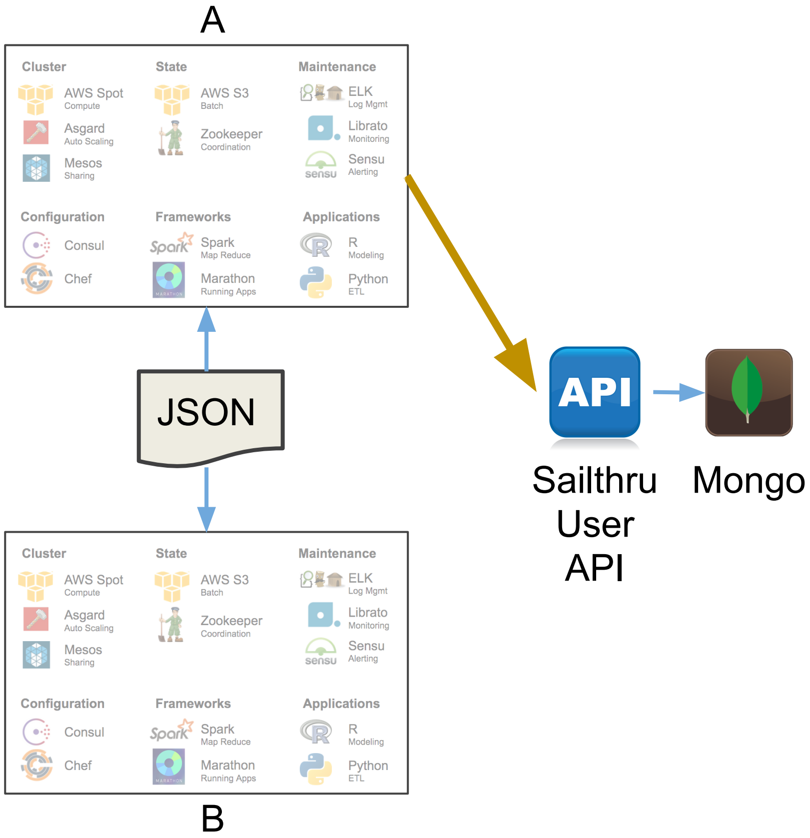
Python
ETL

How we Iterate

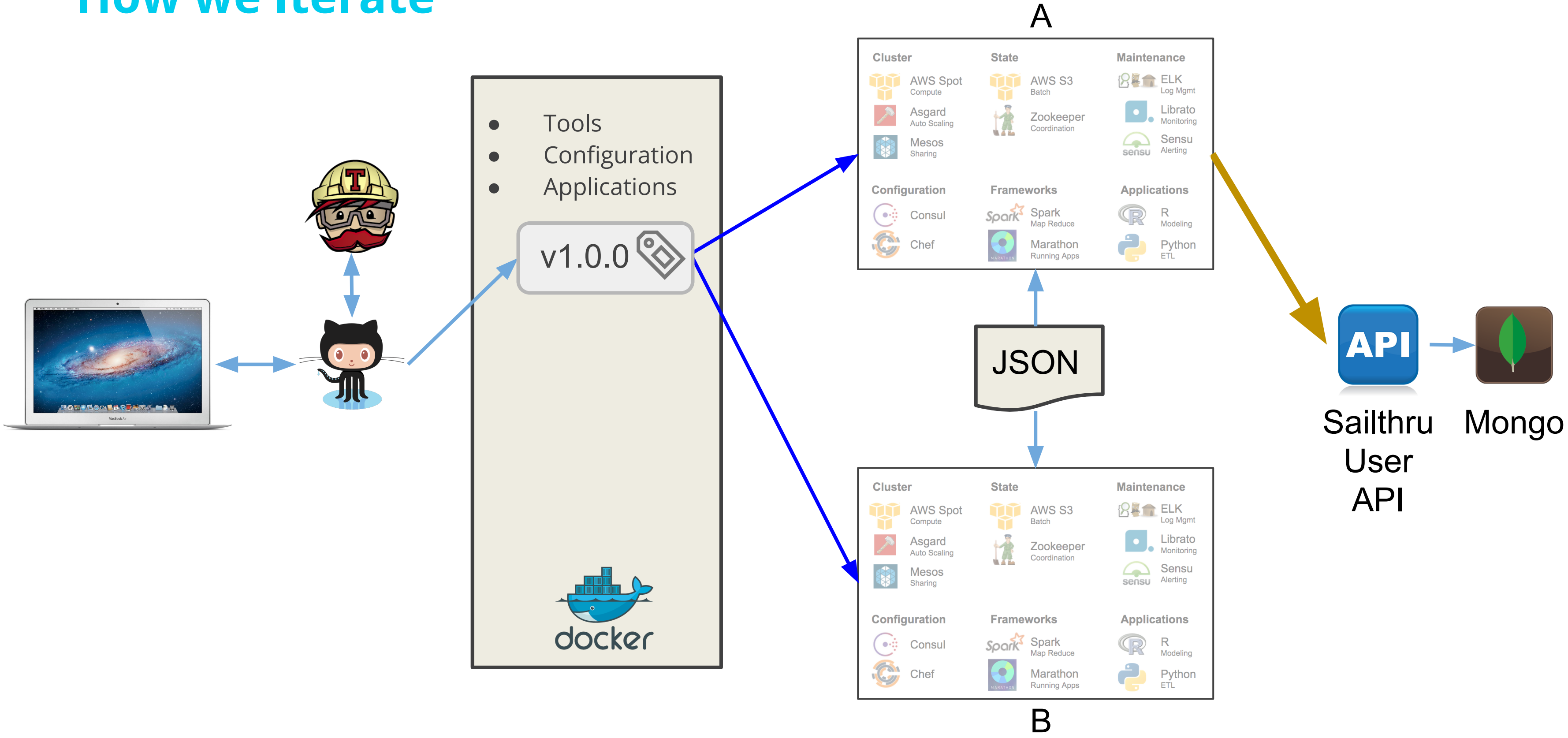


How we Iterate

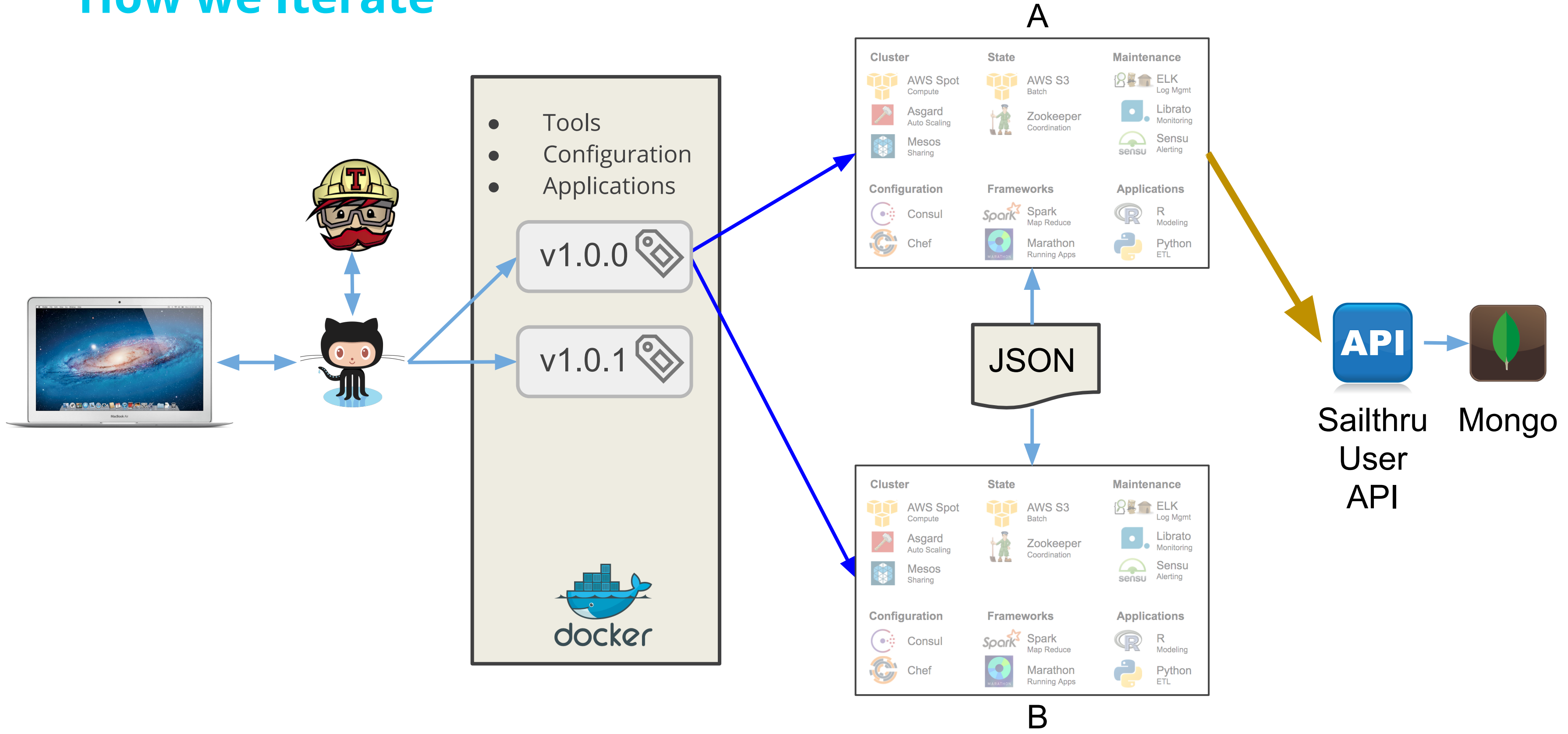




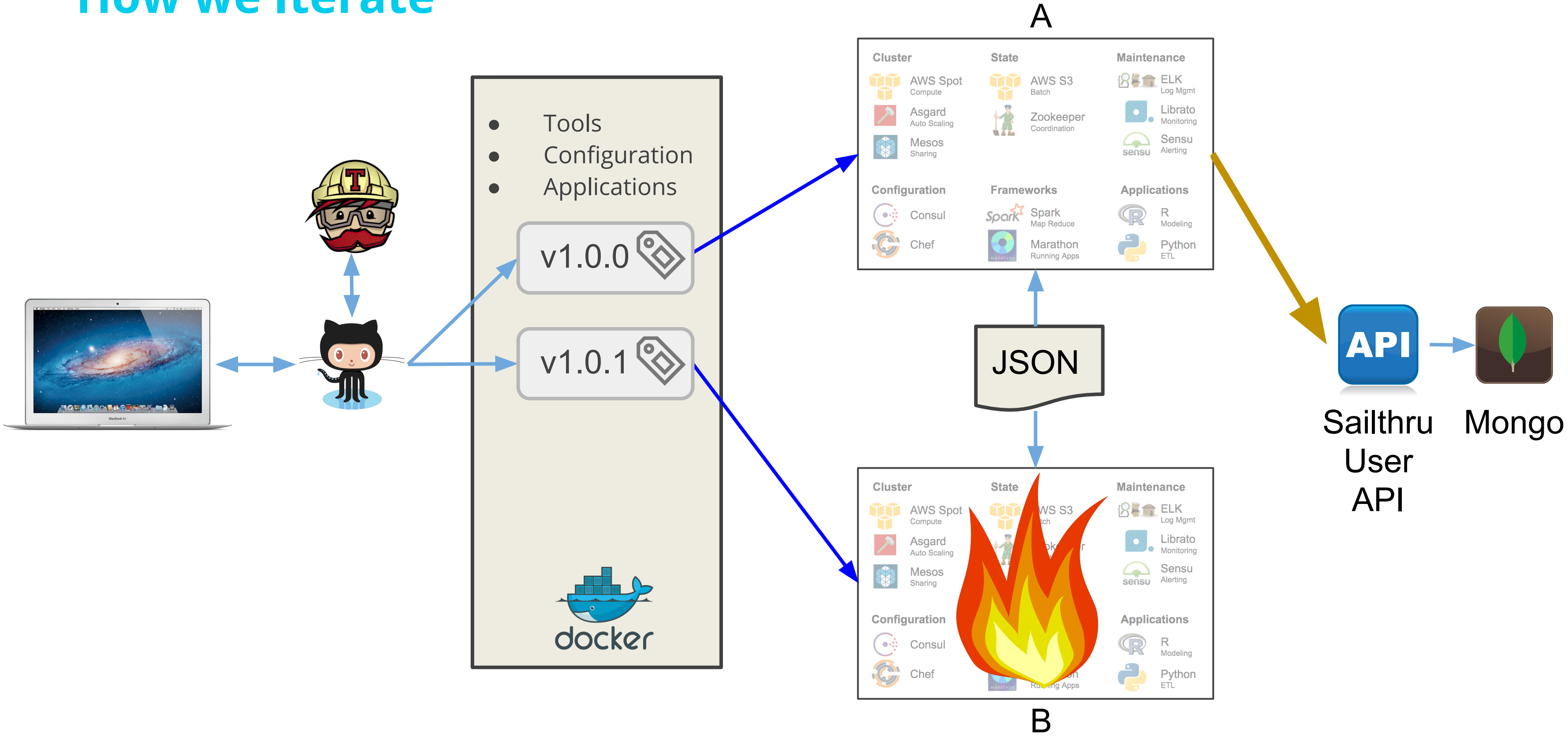
How we Iterate



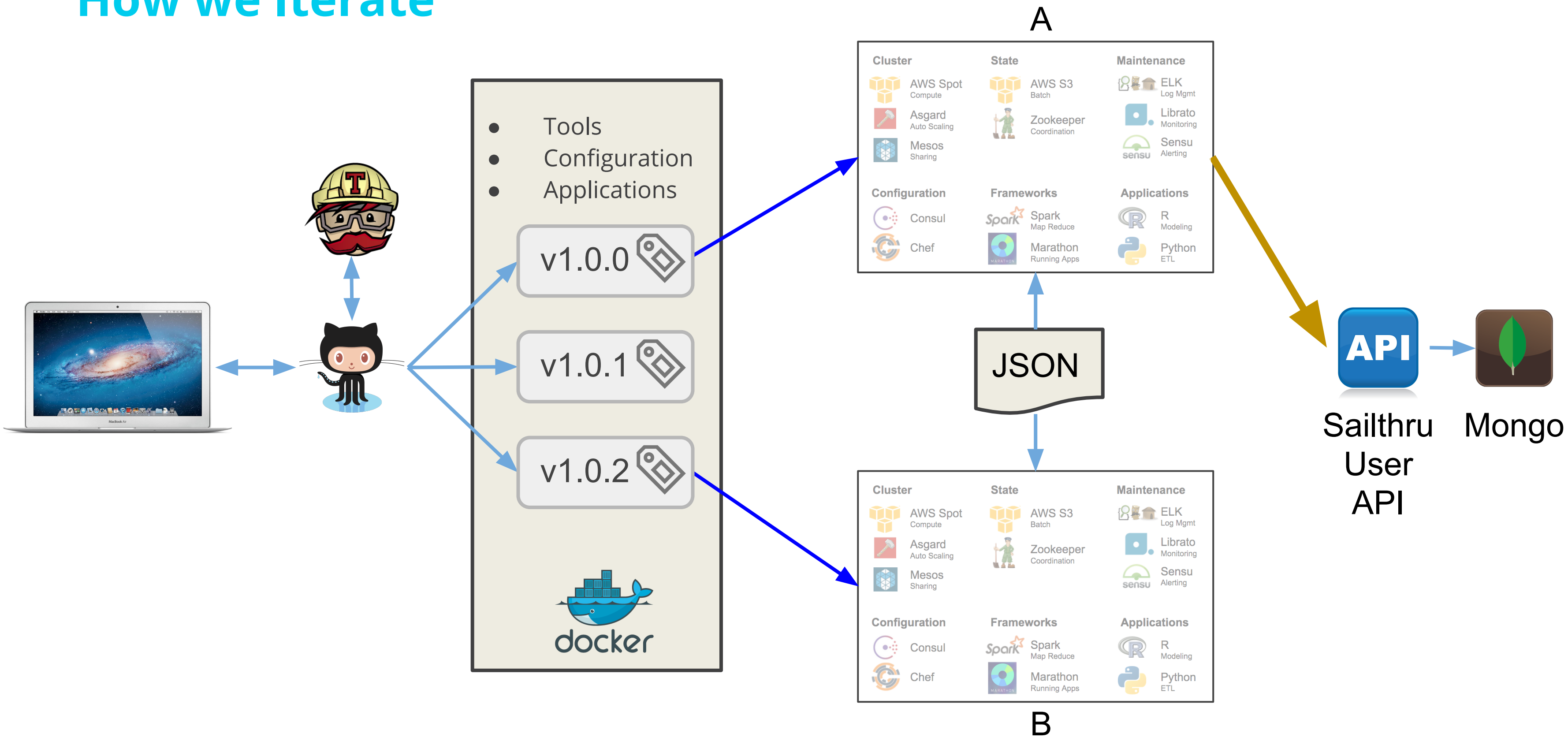
How we Iterate



How we Iterate

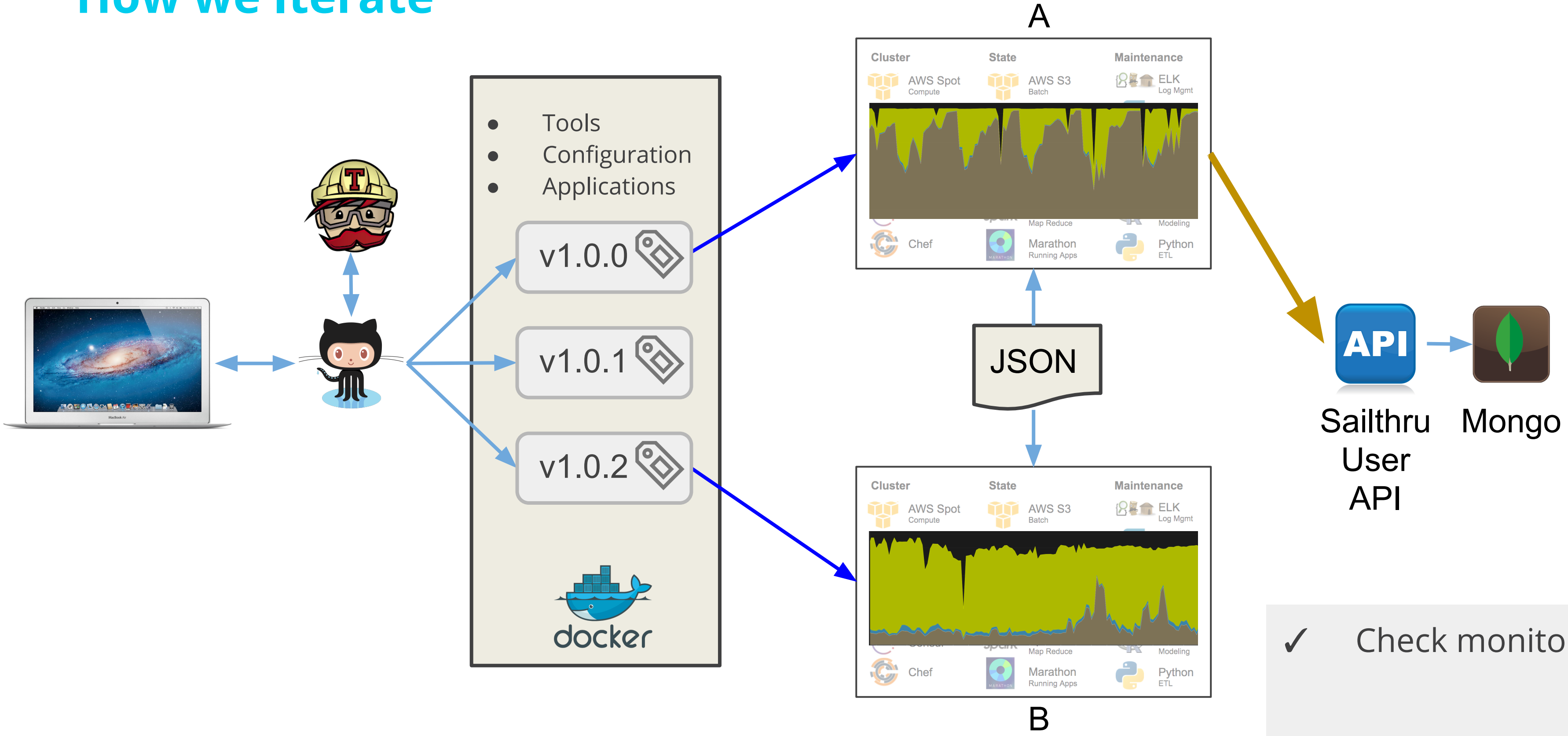


How we Iterate



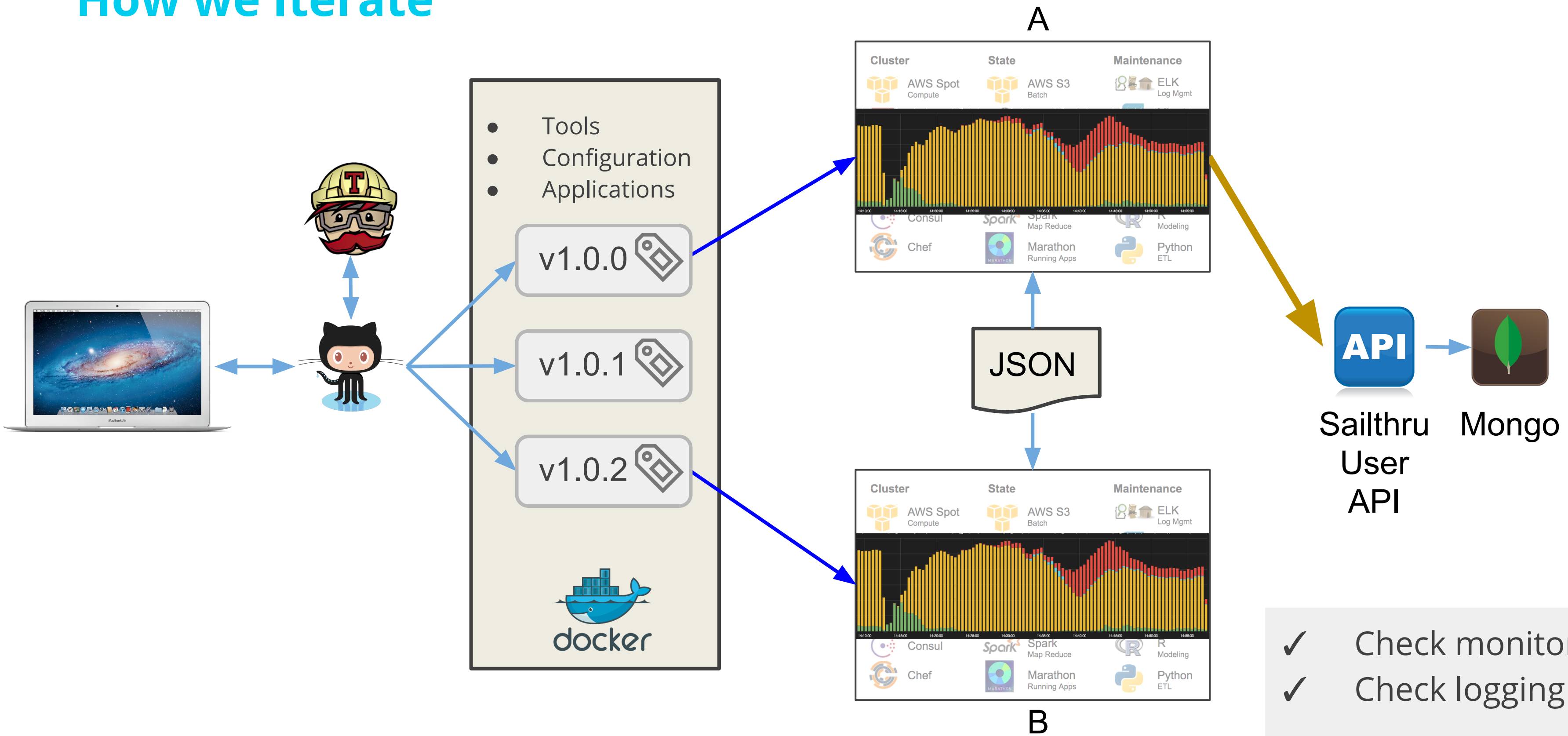
SAILTHRU

How we Iterate



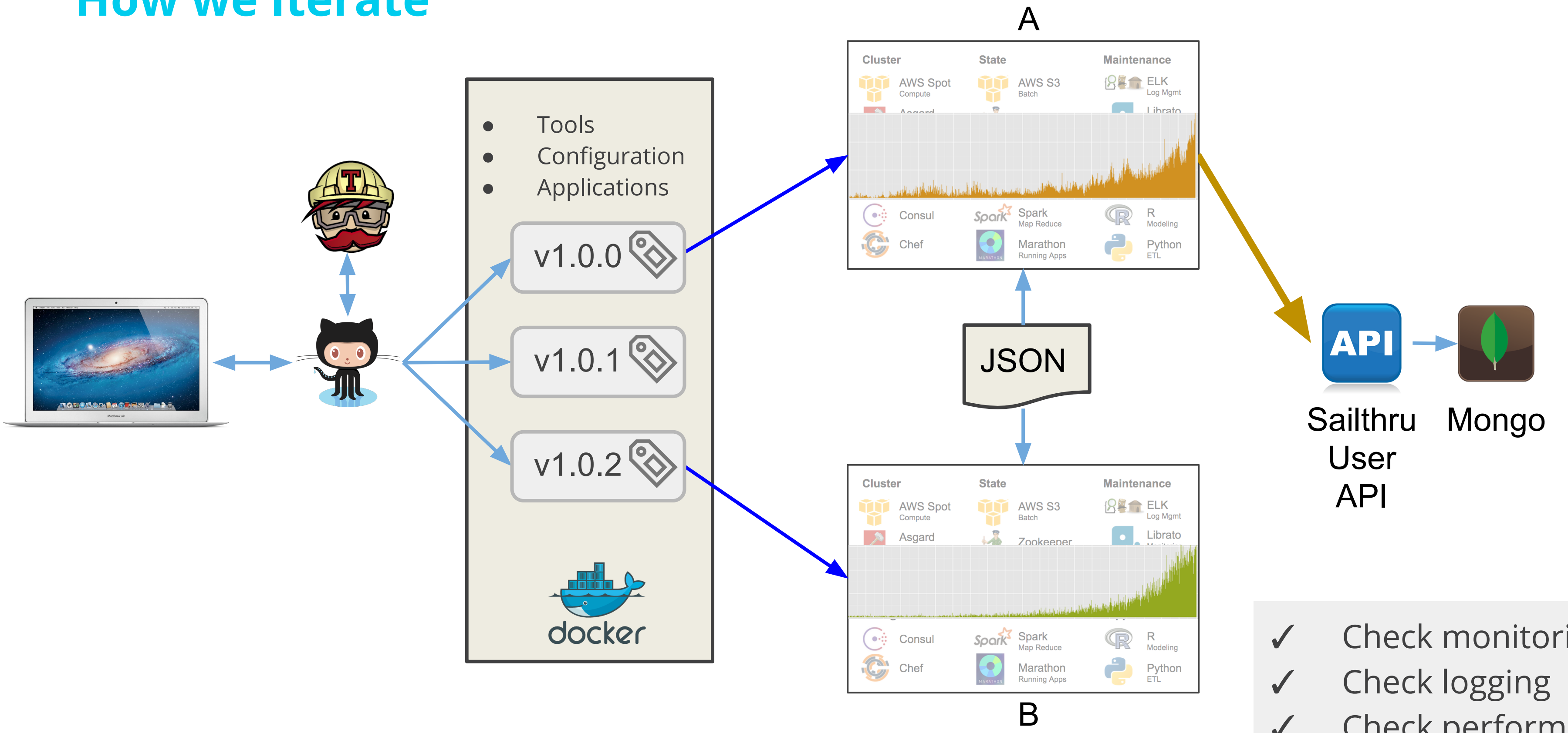
✓ Check monitoring

How we Iterate



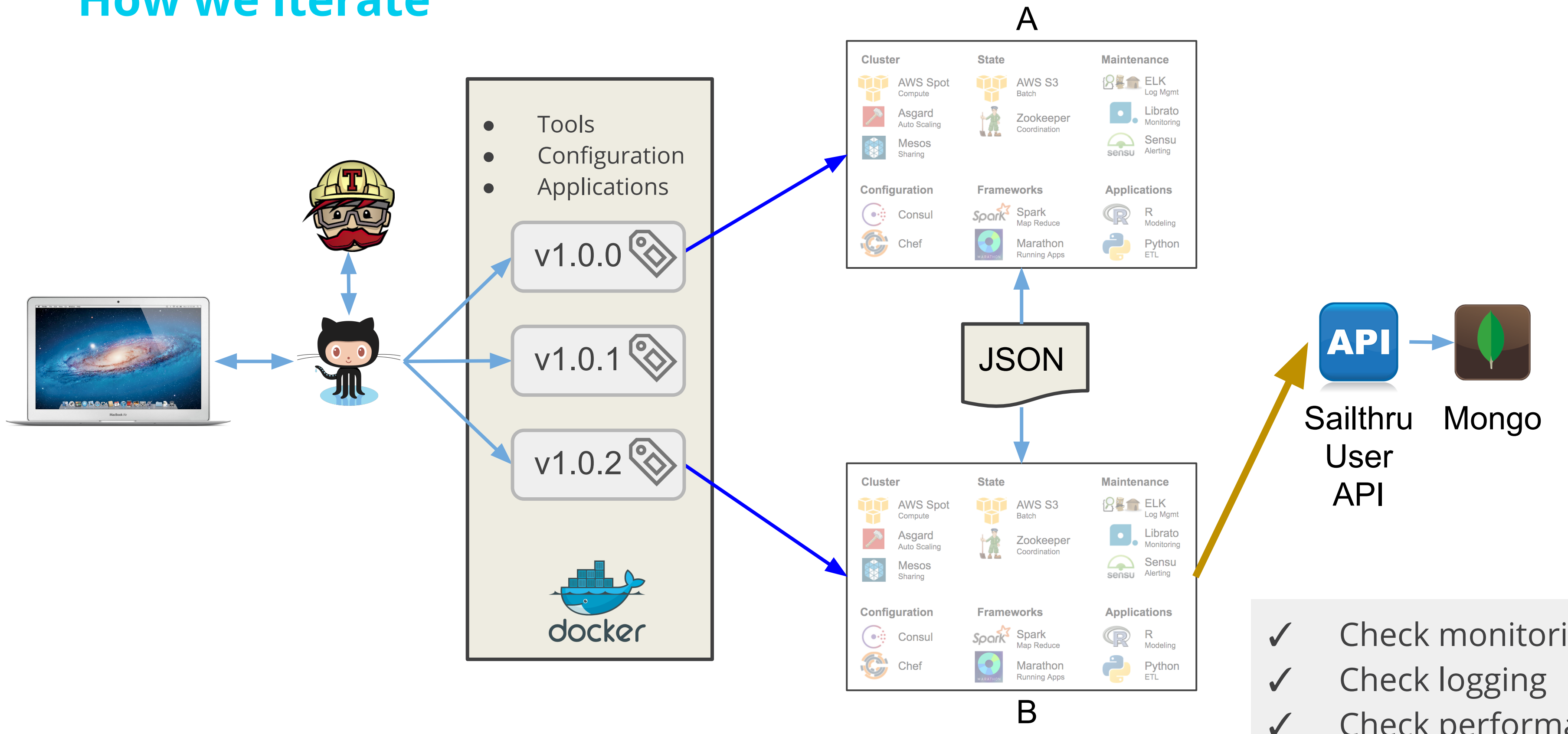
- ✓ Check monitoring
- ✓ Check logging

How we Iterate



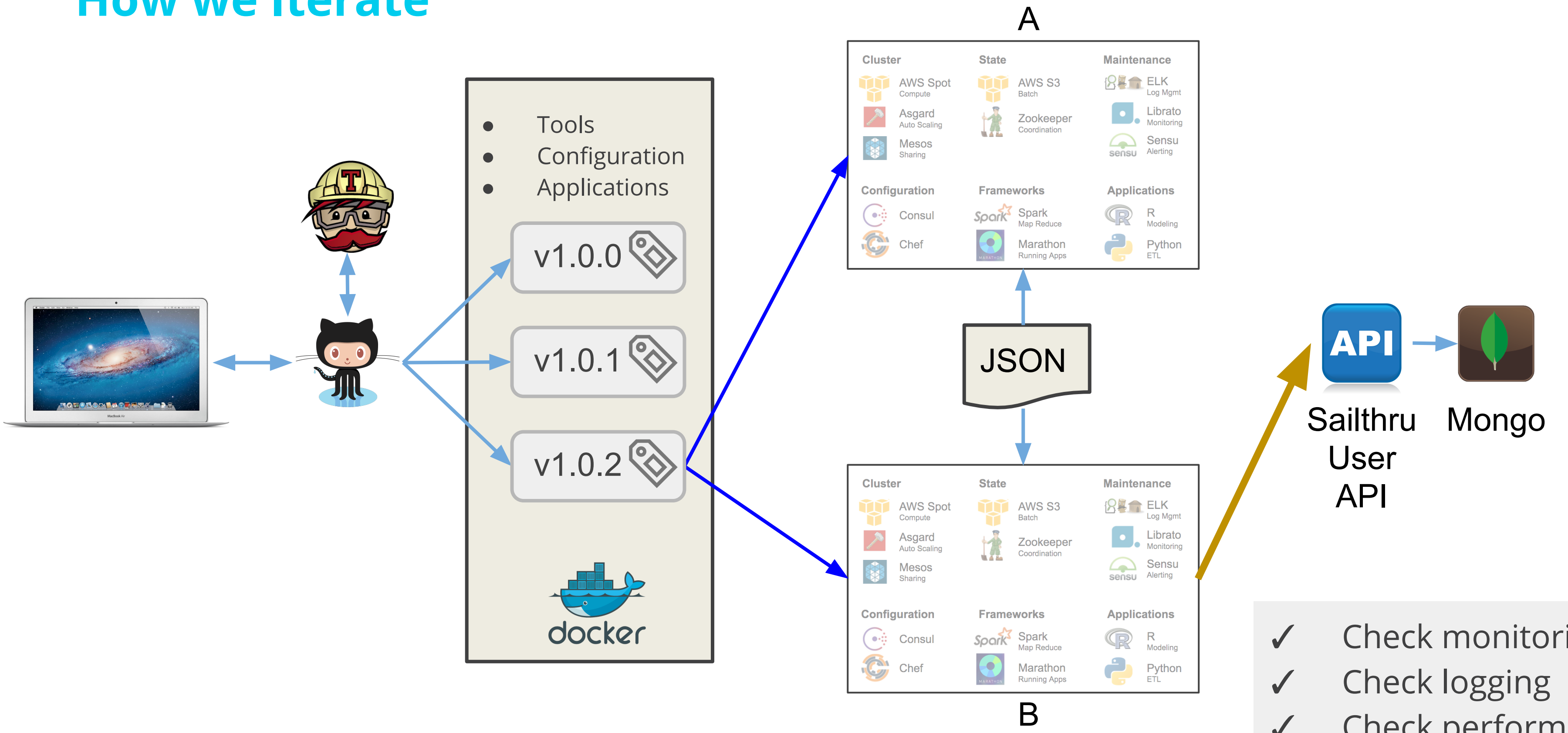
- ✓ Check monitoring
- ✓ Check logging
- ✓ Check performance

How we Iterate



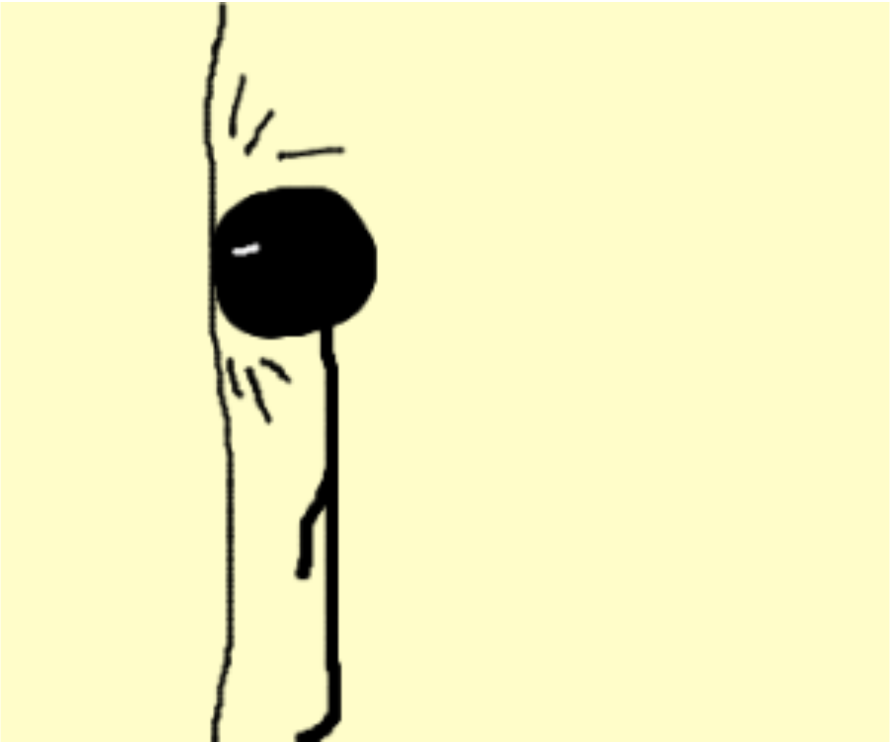
- ✓ Check monitoring
- ✓ Check logging
- ✓ Check performance

How we Iterate



✓ Check monitoring
✓ Check logging
✓ Check performance

Lessons Learned



Lessons Learned

1) **Build multiple layers of fault tolerance**
- Infrastructure - distributed, redundant
- Scheduling - 1+ execution and idempotent apps
- Application - fall back to stale data if need be

Lessons Learned

2) Keep apps and infrastructure isolated and simple

- If you can't explain it in a sentence or need a lot of tests, it's too complex
    - Mesos - resource management
    - Zookeeper - consistent cluster state
    - Marathon - init process for long-running services
    - Relay - task auto-scaling
    - Stolos - DAG scheduling
    - Consul - infrastructure service discovery
    - etc.

Lessons Learned

3) Bound investments in tools, evolve use or give them up quickly

- Marathon - doesn’t handle a huge number of short lived tasks well
- Chronos - cannot handle thousands of independent DAGs
- Spark - use only if you really can’t fit your data into RAM
- HDFS - use S3 if you’re in AWS and design for eventual consistency

Lessons Learned

4) Avoid static partitioning of infrastructure / services / batch

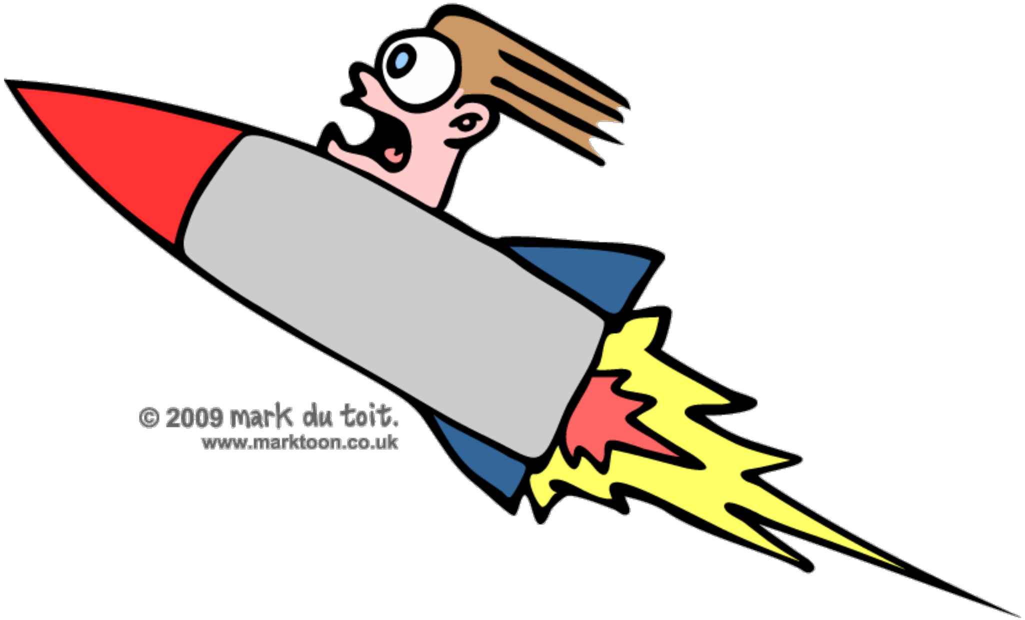
- Much more cost effective to pool resources across them all
- Design all to be equally tolerant to failures
- But must have a means of guaranteeing minimum requirements for some

Lessons Learned

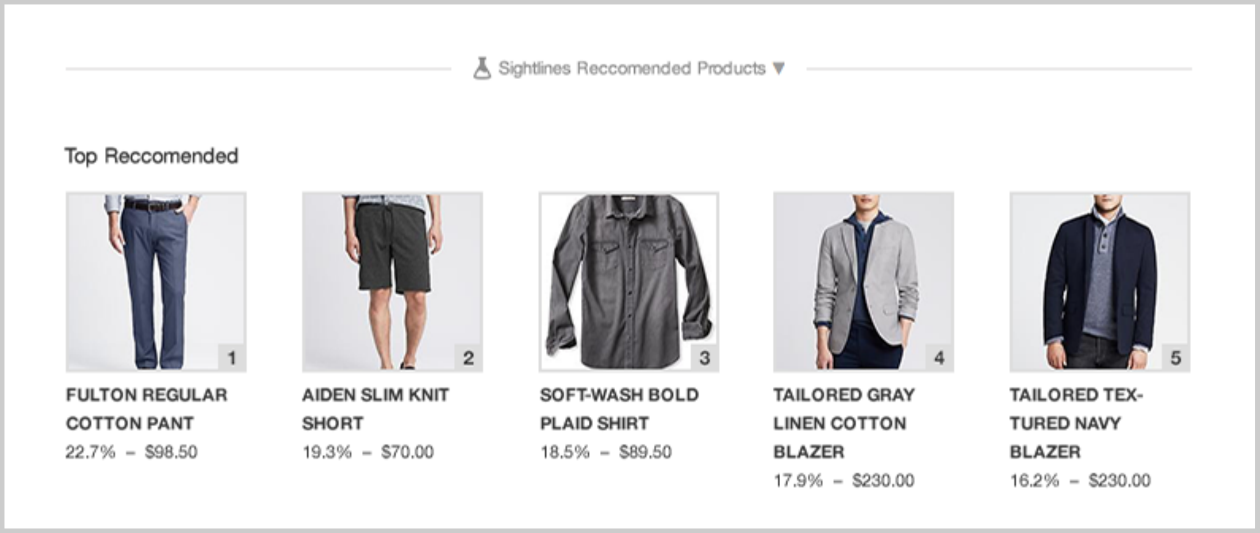
5) Optimize for innovation

- Build a MVP that meets product requirements
- Focus on redundancy, deployment and monitoring early (get this right)
- Stay 10x ahead of scale requirements to minimize disruption from “events”
- Then make iterative infrastructure and app investments to drive ROI

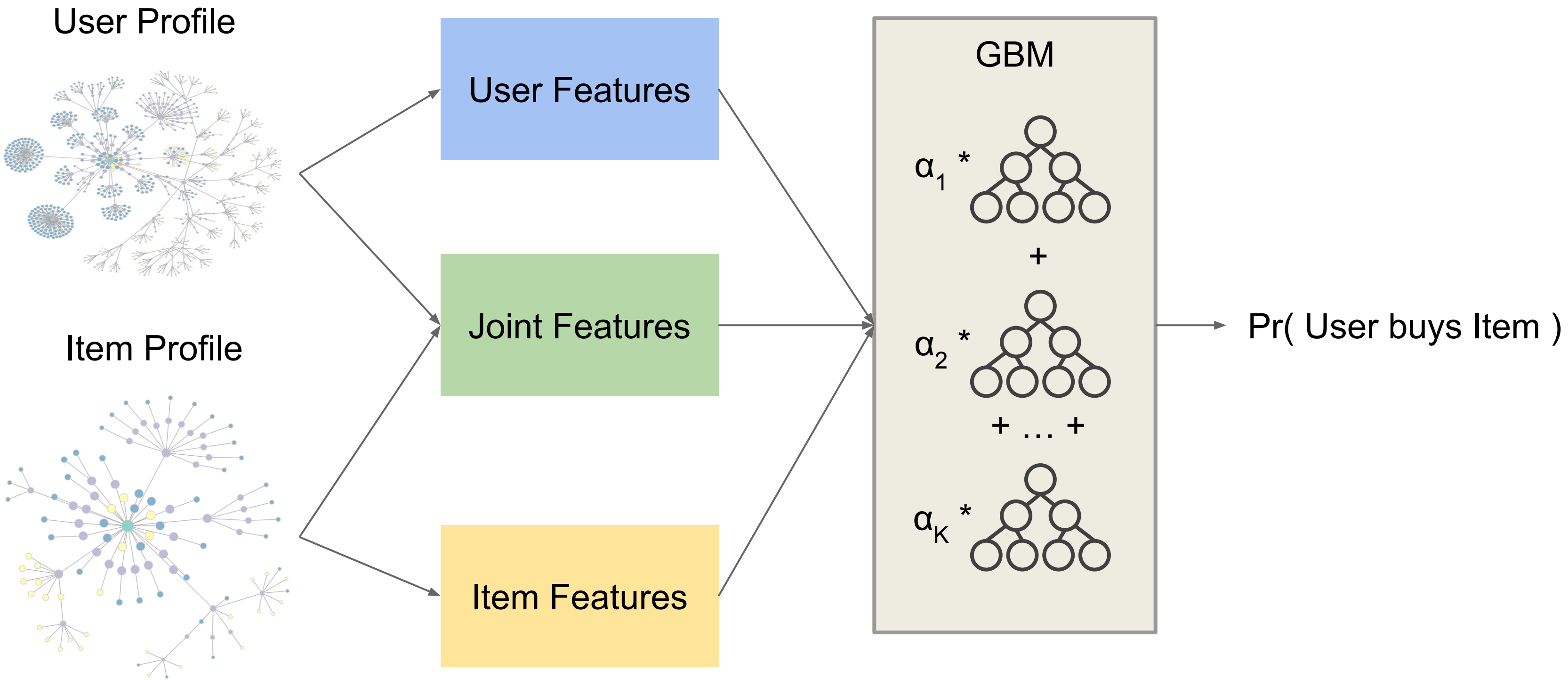
New Innovation



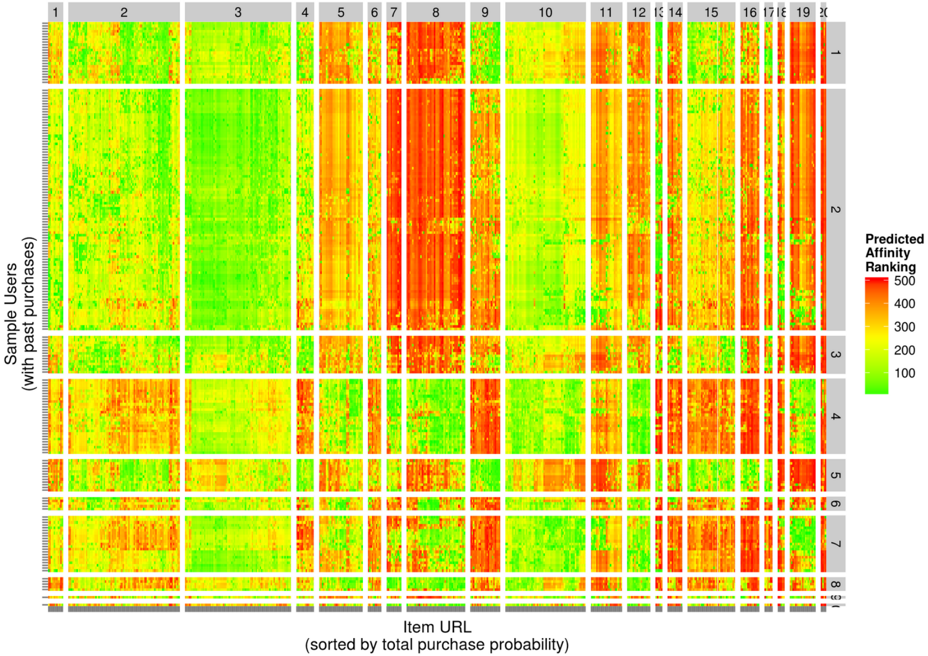
Item Predictions - Reverse Search



Item Predictions - Methodology



Item Predictions - Results



Thank You! Our team:

Divyanshu Vats

Alex Gaudio

Andras Kerekes

Jeremy Stanley

Max Sperlich

SAILTHRU

Connect with us.

www.sailthru.com
sales@sailthru.com
817.812.8689

NYC HQ
160 Varick St., 12th Floor
New York, NY 10013

San Francisco
25 Taylor St., Room 724
San Francisco, CA 94102

Los Angeles
7083 Hollywood Blvd
Los Angeles, CA 90028

London
18 Soho Square
London, UK, W1D 3QL