Some things you learn running Apache Spark in production for three years

William Benton (@willb)

Red Hat, Inc.

Some things you learn running Apache Spark in production for three years

William Benton (@willb)

Red Hat, Inc.

About me

Since 2013: data engineering and data science focus

Since 2008: distributed systems focus

Ancient history: compiler/VM design, static analysis, logic programming

Forecast

Introducing Apache Spark

How my team has used Spark

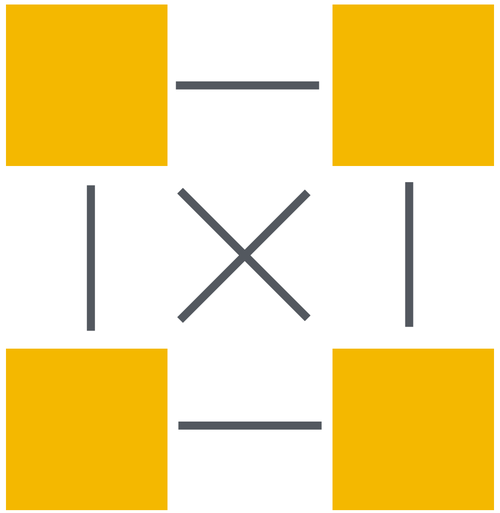
Lessons we've learned

From analytics as a workload to insightful applications

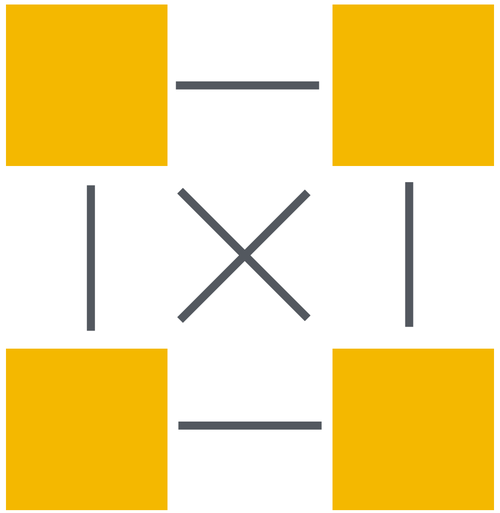
Meet Apache Spark

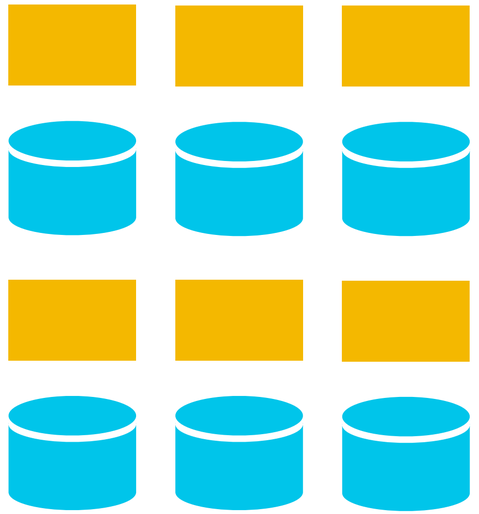
Parallel execution models

Parallel execution models



Parallel execution models





A user-friendly abstraction

Partitioned collections are spread across multiple processors or machines.

Immutable collections are copied, never modified in place.

Lazy operations only execute when necessary.

A user-friendly abstraction

This means we'll have failures

Partitioned collections are spread across multiple processors or machines.

Immutable collections are copied, never modified in place.

Lazy operations only execute when necessary.

A user-friendly abstraction

This means we'll have failures

Partitioned collections are spread across multiple processors or machines.

Immutable collections are copied, never modified in place.

Lazy operations only execute when necessary.

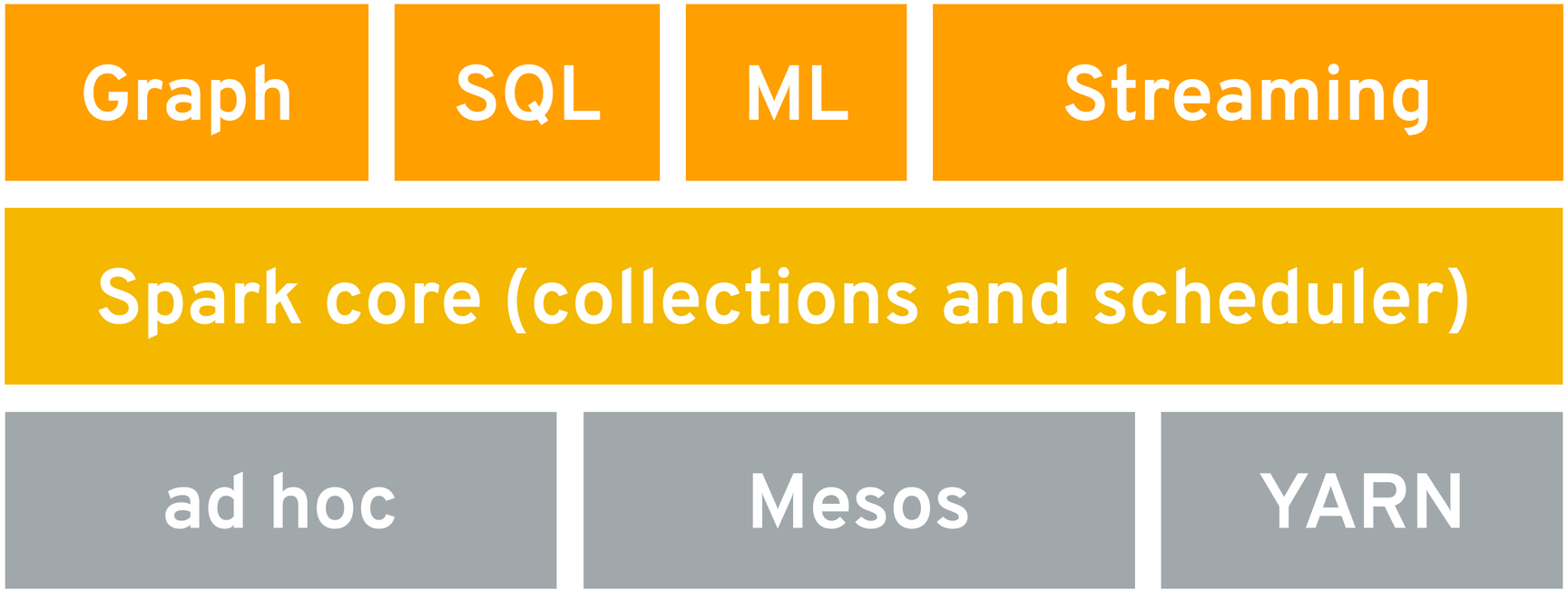
These mean we'll always have a recipe for how to recover

A simple example program

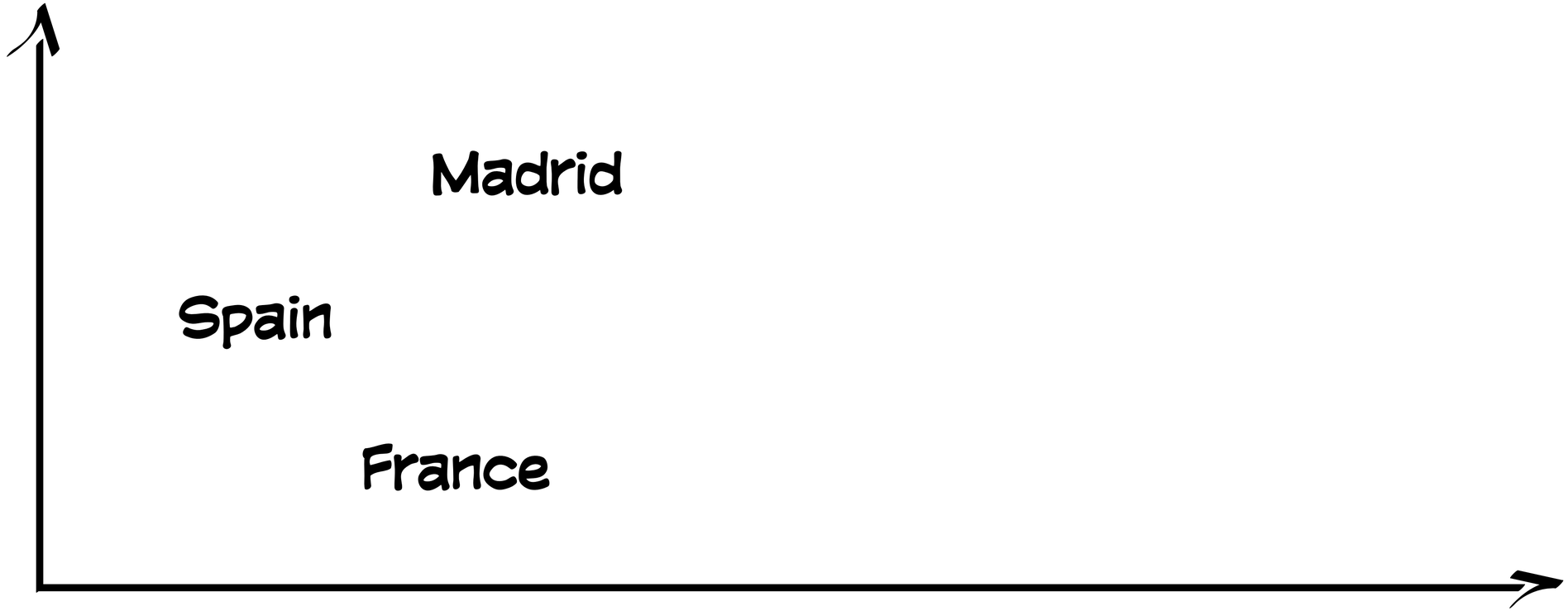
```
file = sc.textFile("file://...")

counts = file.flatMap(lambda l: l.split(" "))
              .map(lambda w: (w, 1))
              .reduceByKey(lambda x, y: x + y)

# computation actually occurs here
counts.saveAsTextFile("file://...")
```

Spark core (collections and scheduler)

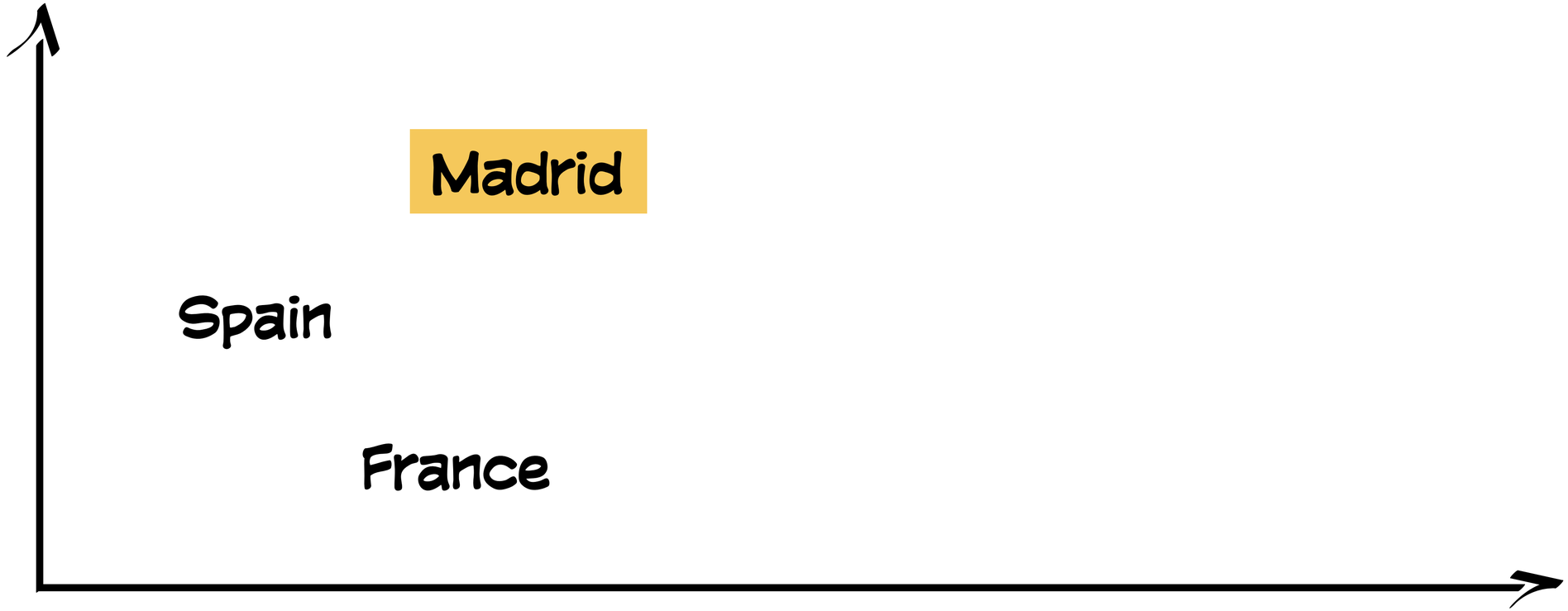

Graph
SQL
ML
Streaming
Spark core (collections and scheduler)


Graph
SQL
ML
Streaming
Spark core (collections and scheduler)
ad hoc
Mesos
YARN

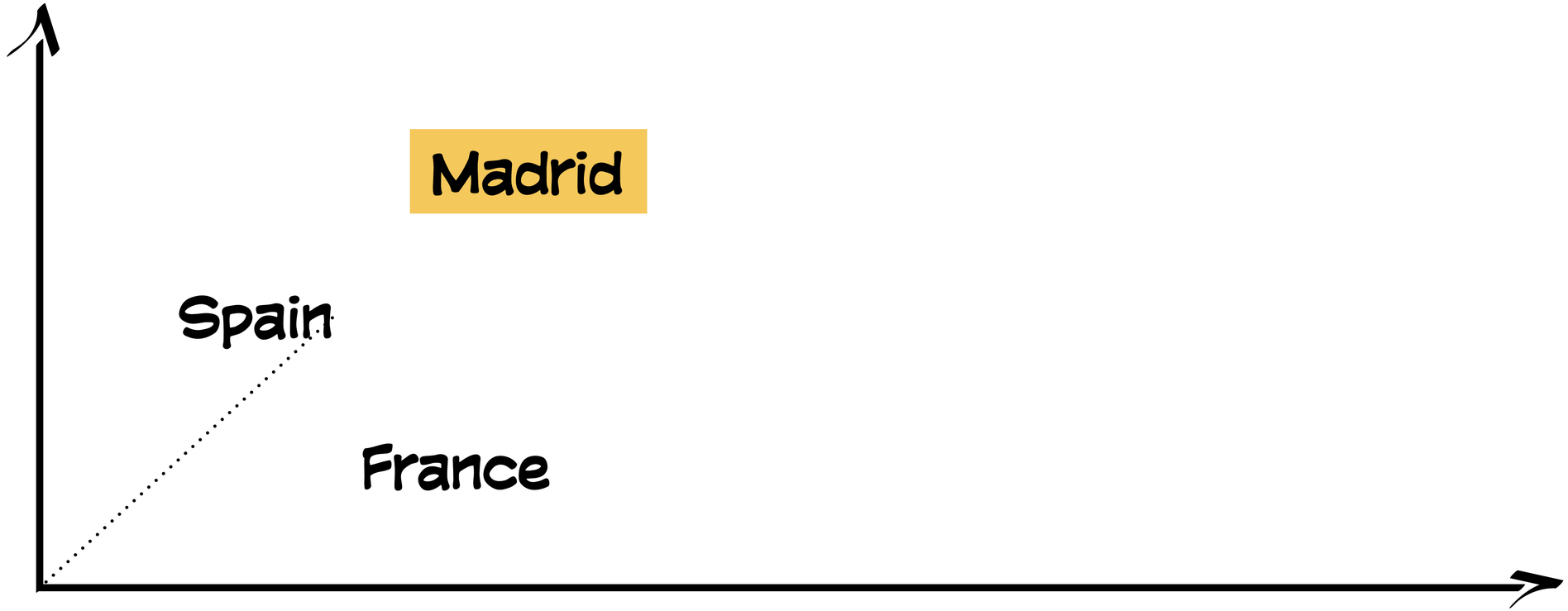
A more interesting example



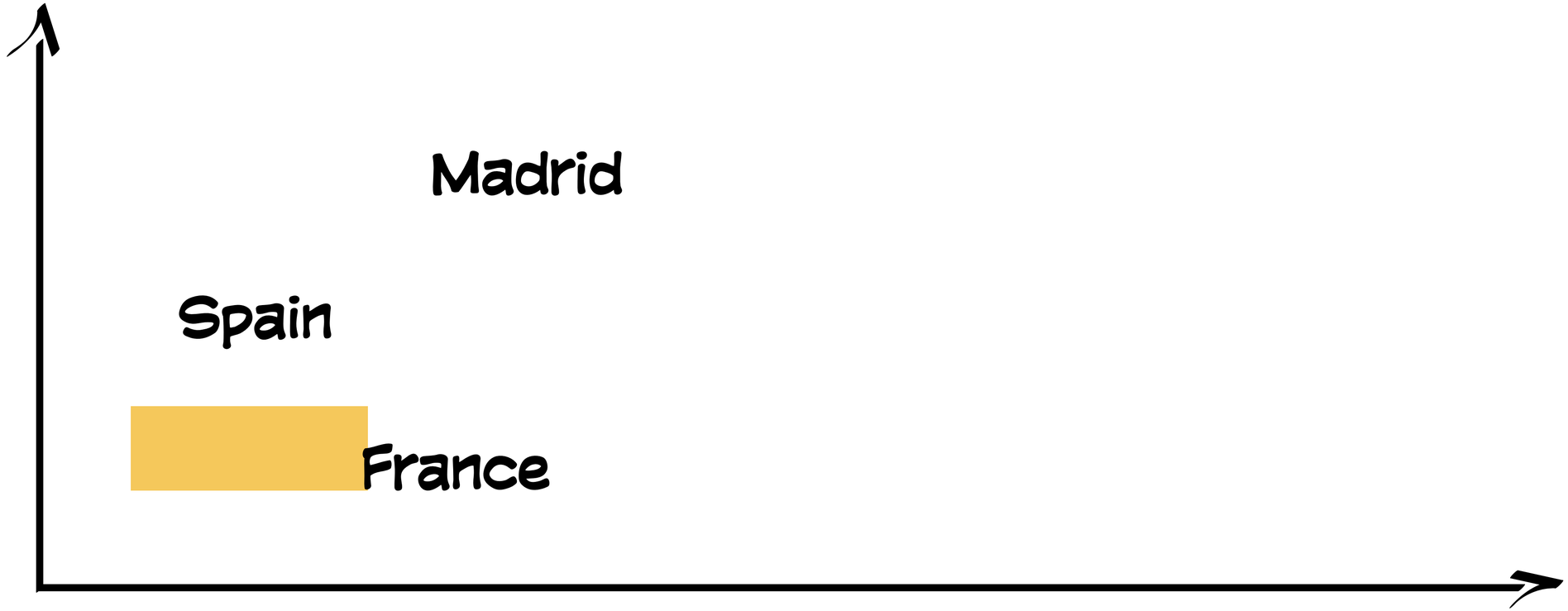
A more interesting example



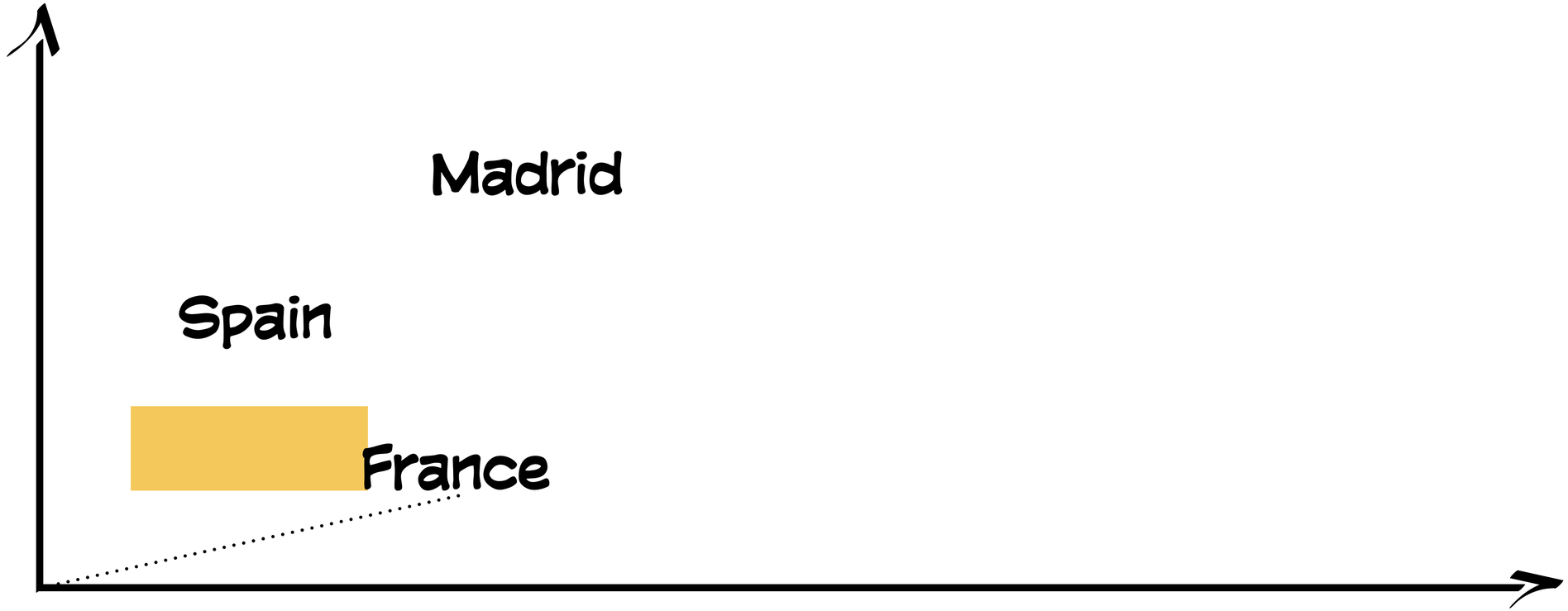
A more interesting example



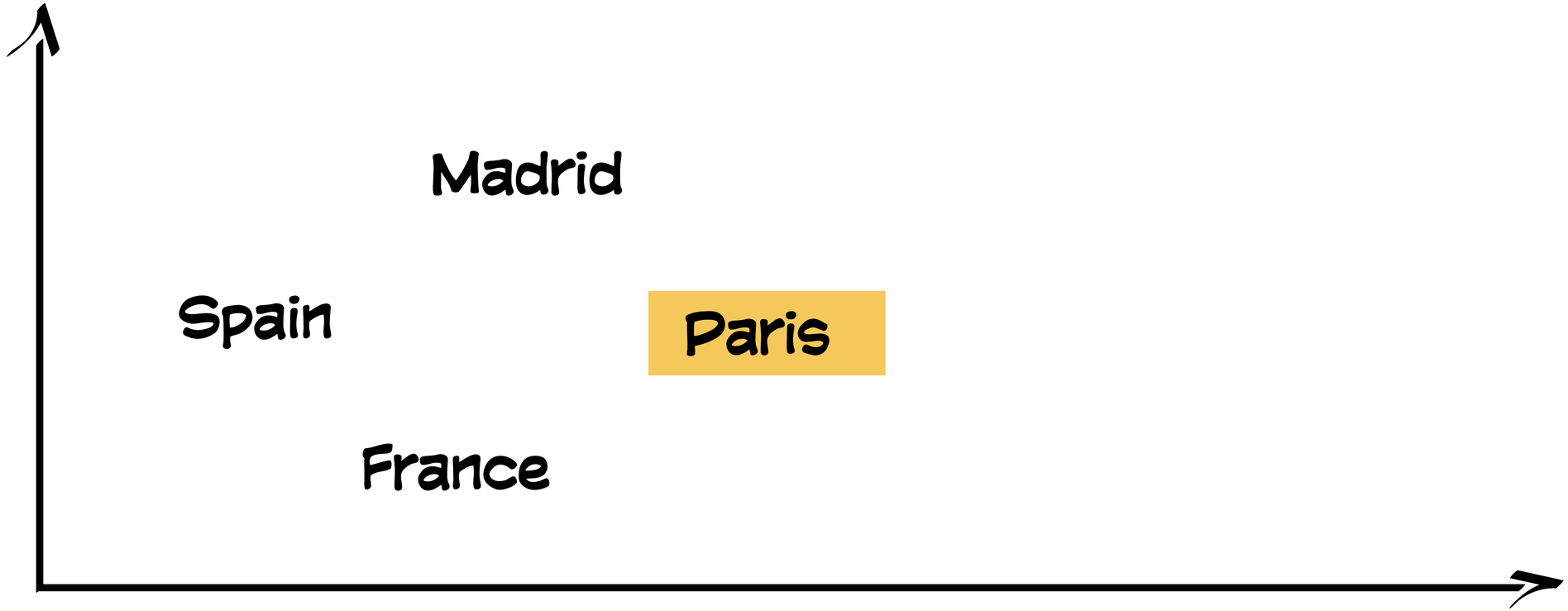
A more interesting example



A more interesting example



A more interesting example

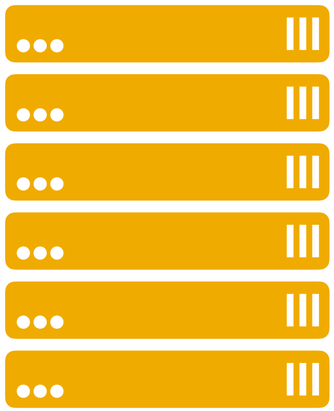


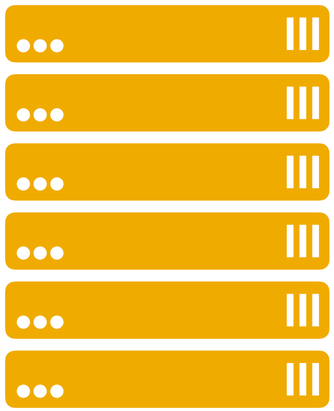
v("Madrid") - v("Spain") + v("France") ≈ v("Paris")

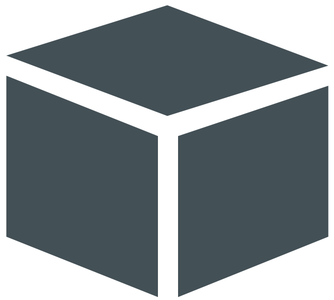
A more interesting example

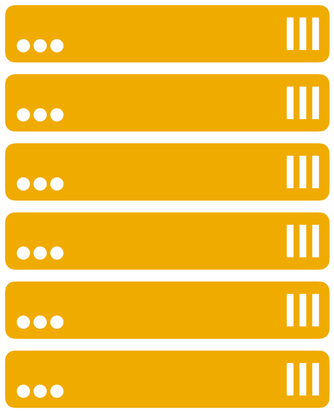
```
# read text file as a data frame
df = spark.read.text("file://...")
        .select(split("value", "\s+").alias("text"))

# fit and use a Word2Vec model
w2v = Word2Vec(inputCol="text", outputCol="result")
model = w2v(df)
model.findSynonyms("data", 5).show()
```

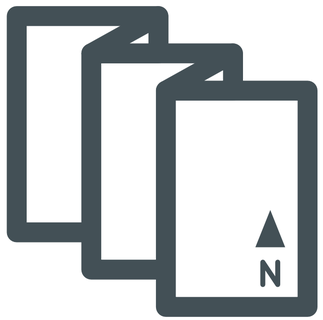
How we've used Spark

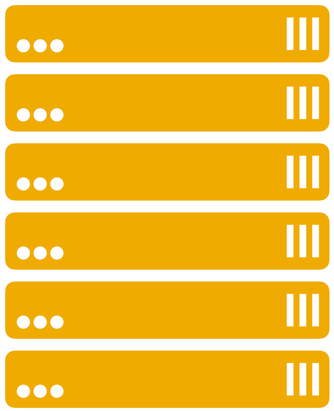


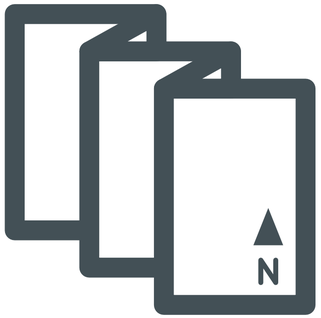


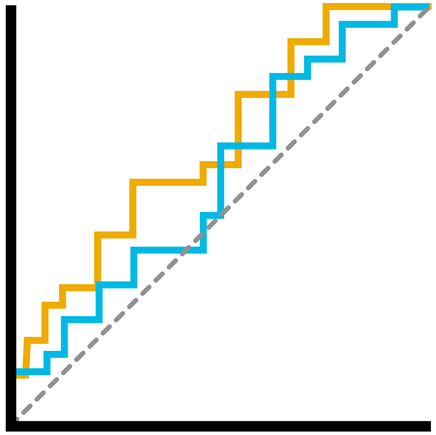




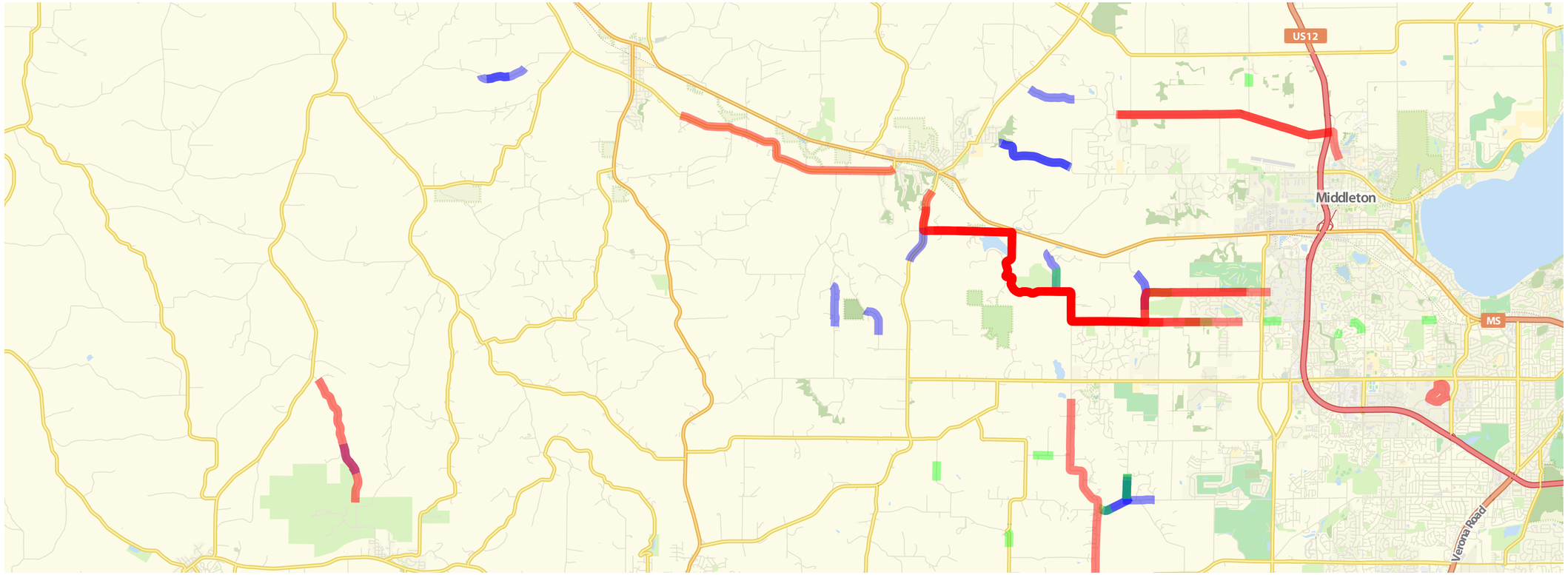

N



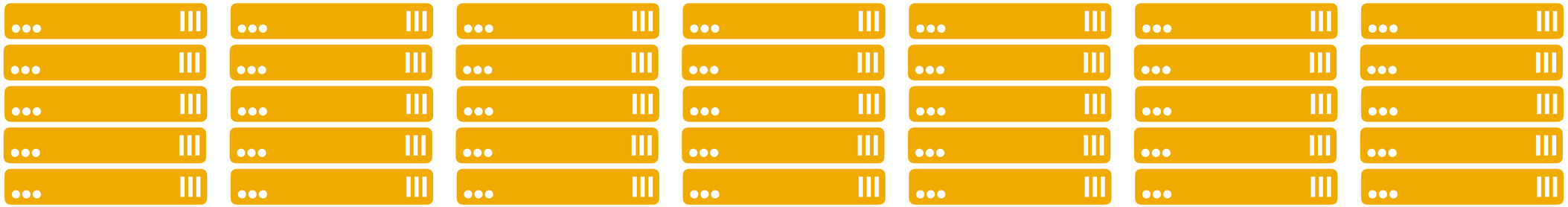

N



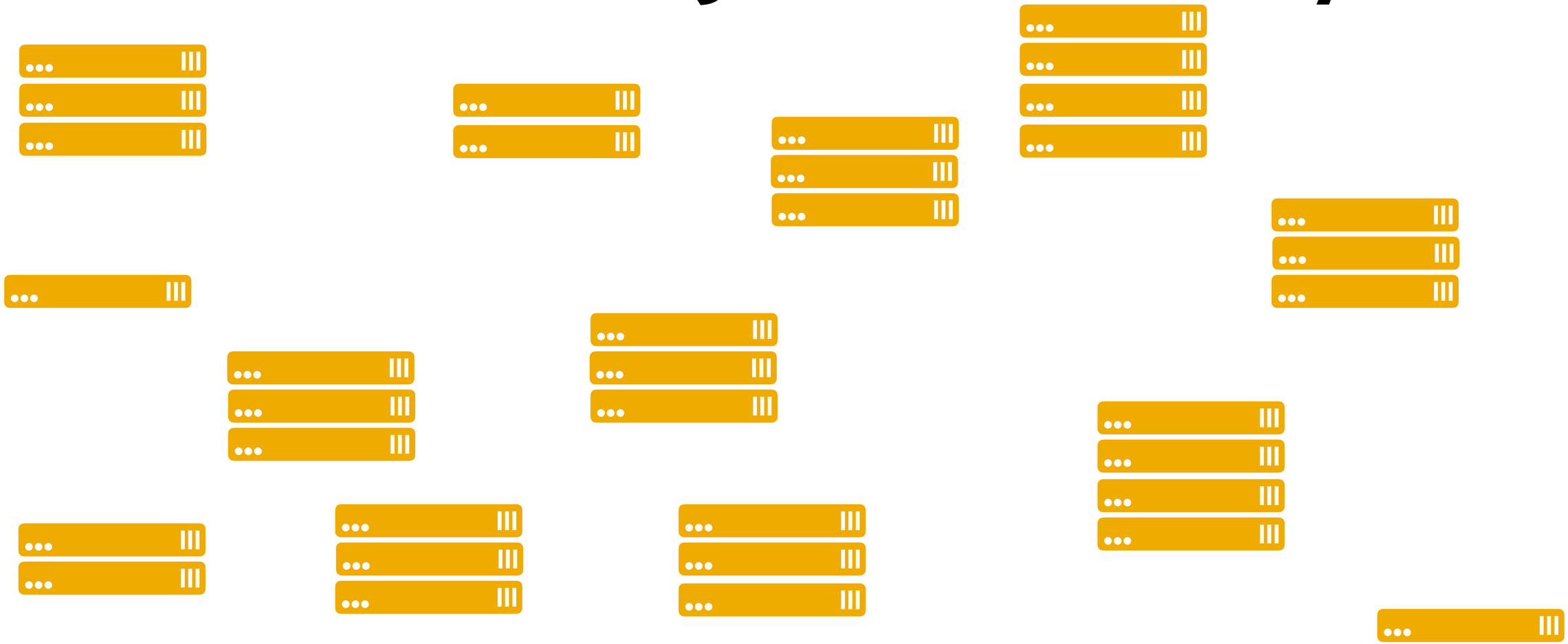
Prototyping new techniques



Machine configuration analysis



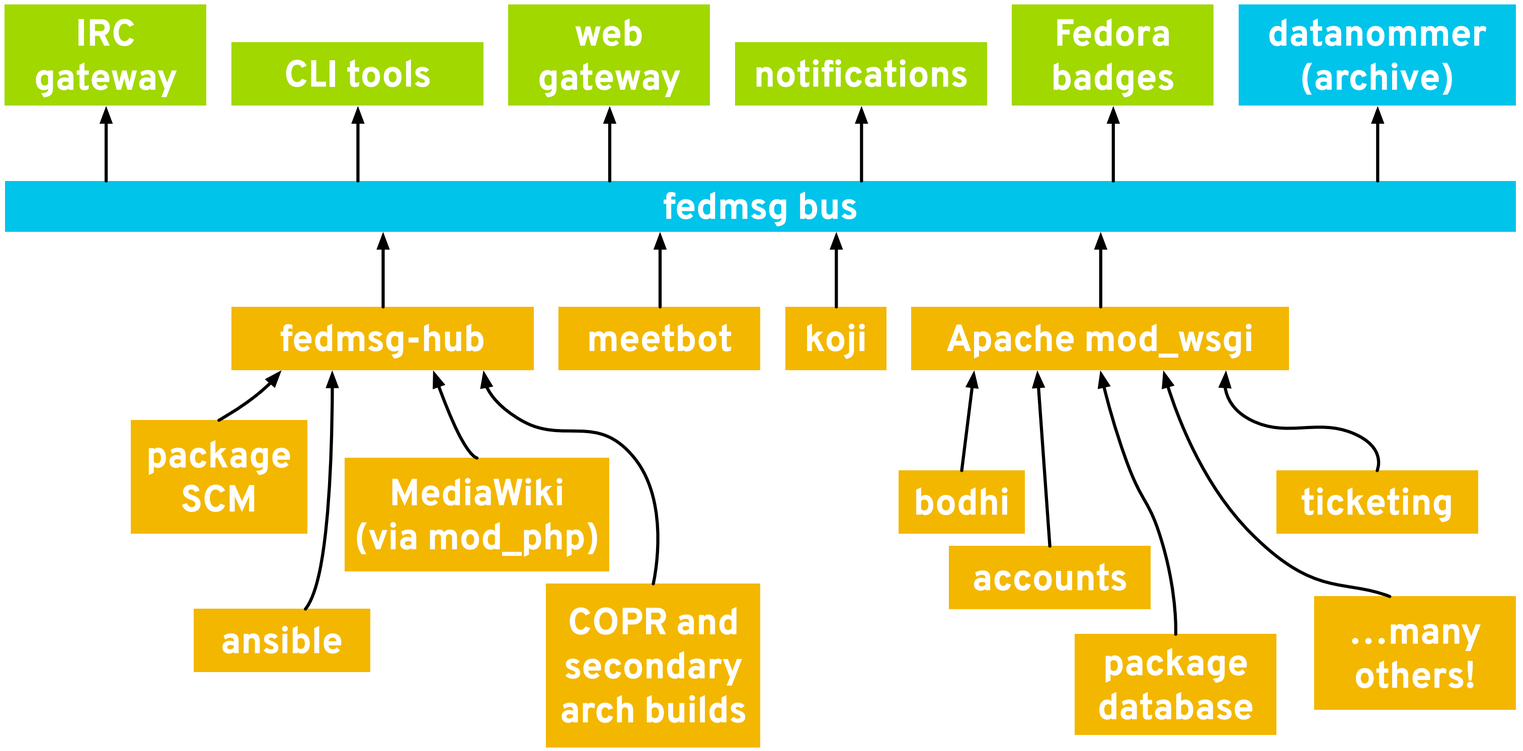
Machine configuration analysis



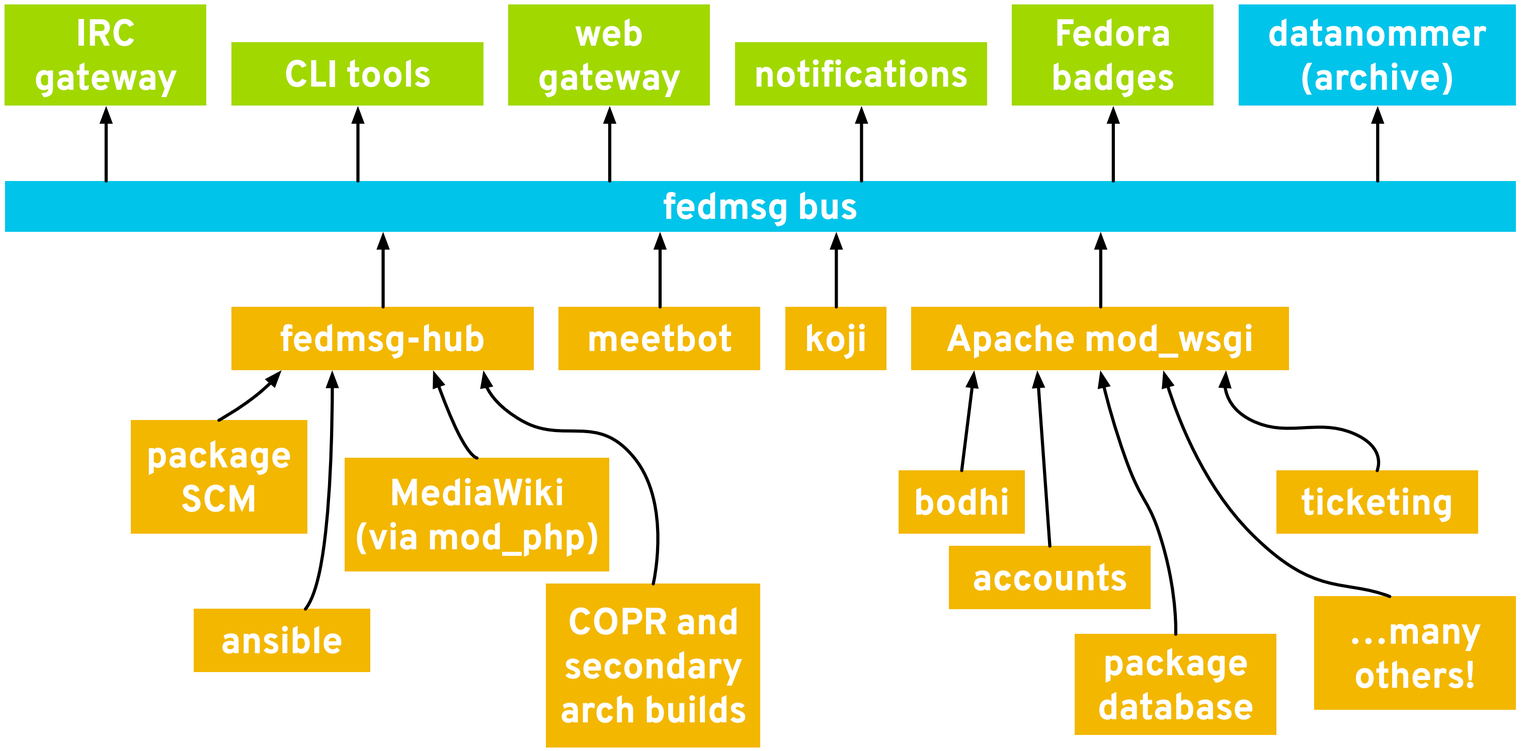
Machine configuration analysis

```
not libhugetlbfs && not fontpackages-filesystem &&
  httpclient && transitional-eap6-jars
p11-kit && not libtheora && not perl-Text-ParseWords &&
  jboss-metadata-appclient
not python-suds && rt61pci-firmware &&
  avahi && apache-commons-io-ea
not sanlock-python && shrinkwrap-parent &&
  not cli-tools-zend-server
not perl-parent && not python-stevedore && jdom &&
  not openshift-origin-cartridge-diy &&
  not redhat-sso-login-module-eap6 && iwl6050-firmware-41
not pytalloc && not libldb && xorg-x11-fonts-Typ &&
  not nodejs010-gyp && hibernate4-entitymanager
...
```

Diagnosing community health

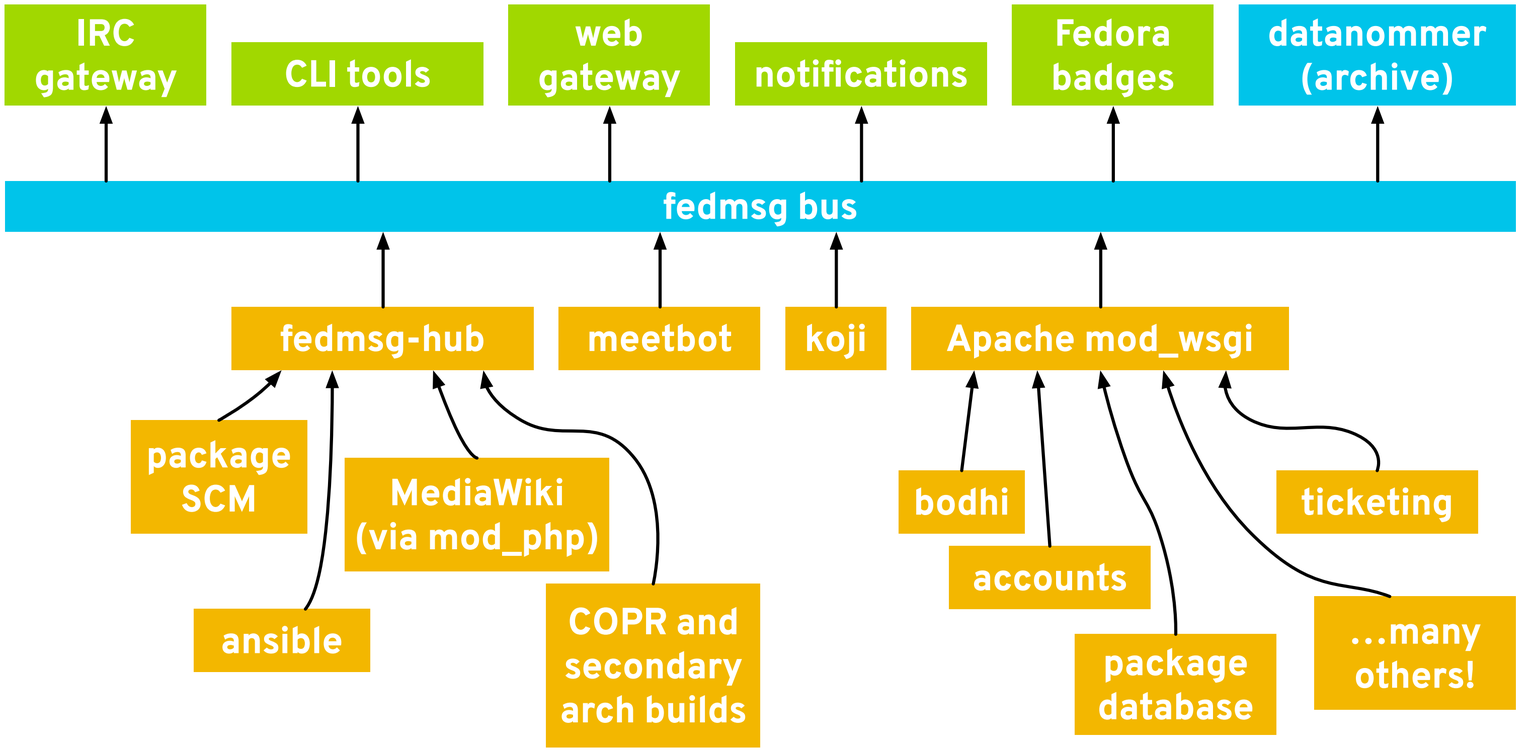


Diagnosing community health



Do people work on Fedora for love or money?

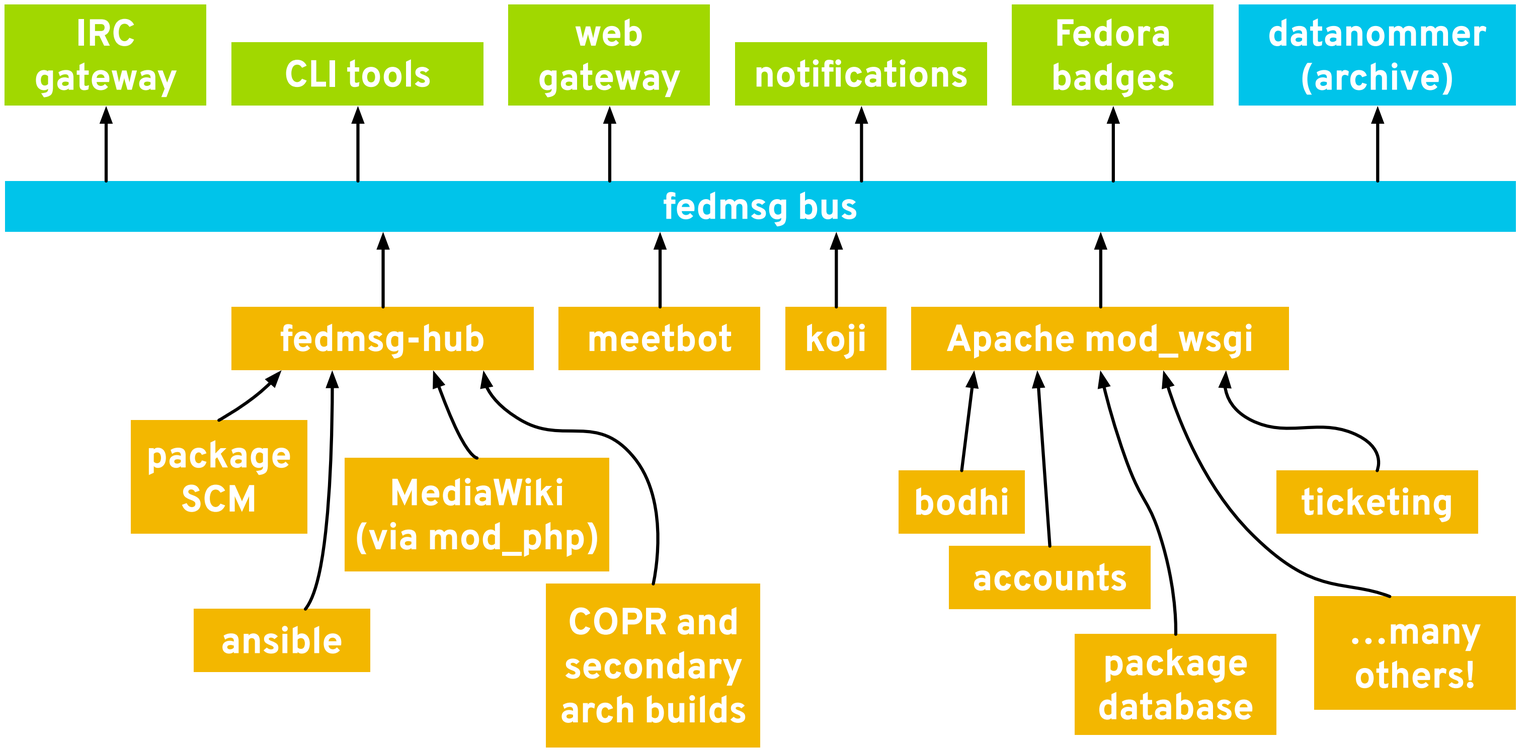
Diagnosing community health



Do people work on Fedora for **love** or **money**?

Is there anyone the community **couldn't live without**?

Diagnosing community health

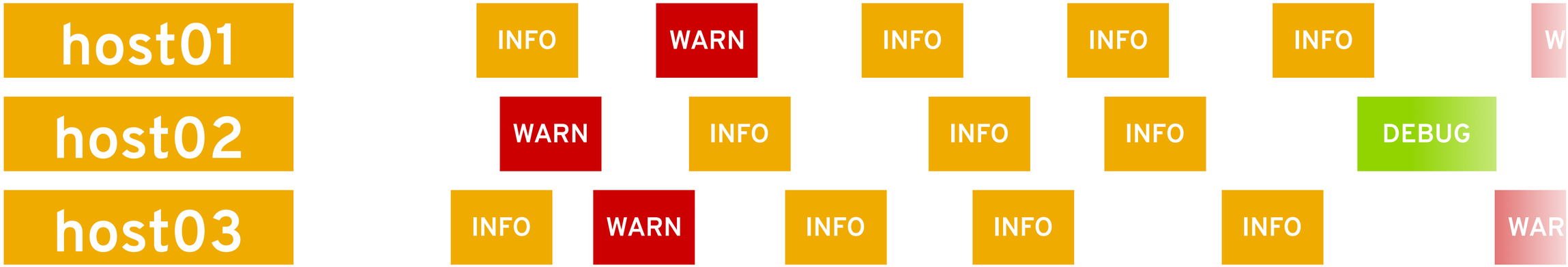


Do people work on Fedora for **love** or **money**?

Is there anyone the community **couldn't live without**?

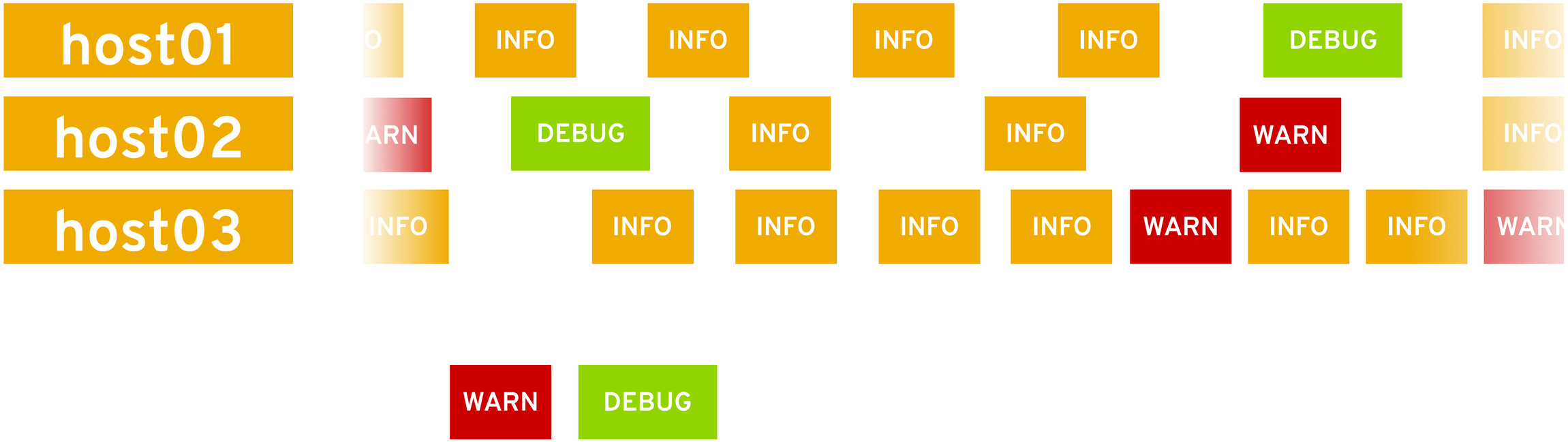
How do we characterize **breadth** and **depth** of community engagement?

Outlier detection in log data

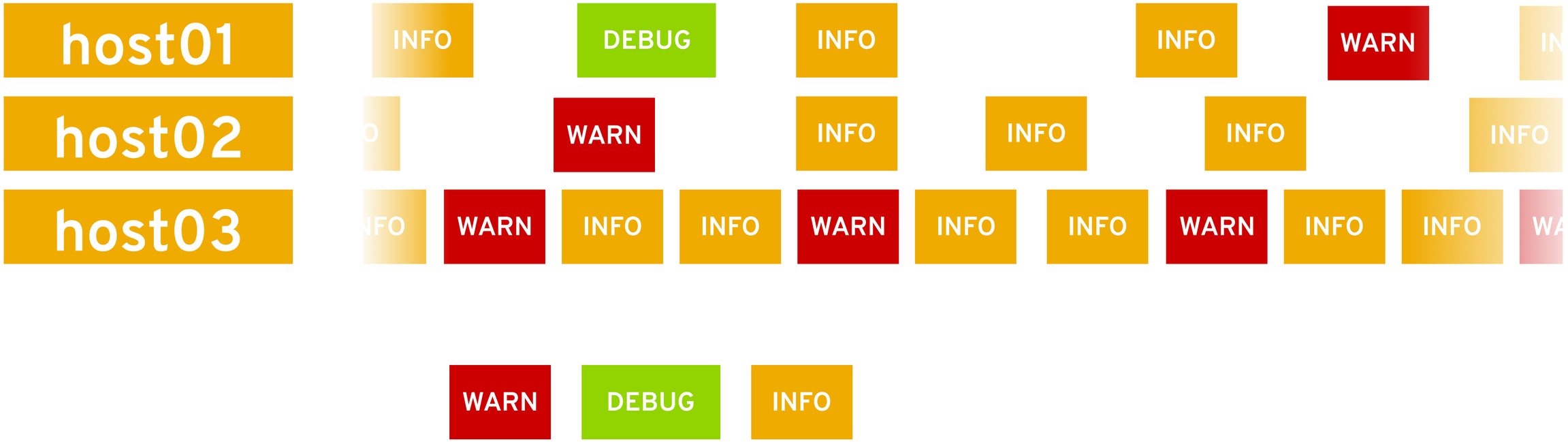


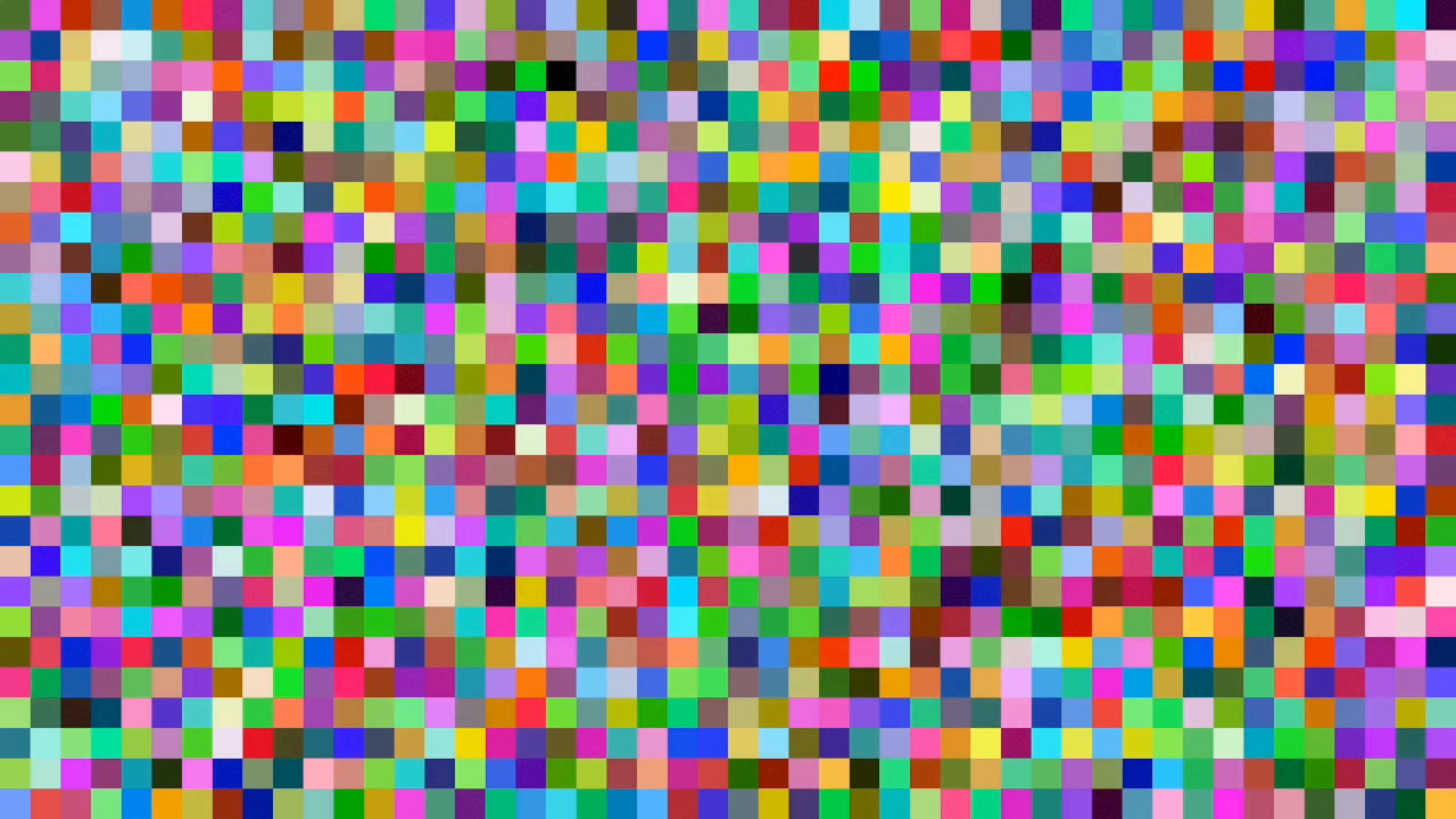
Outlier detection in log data

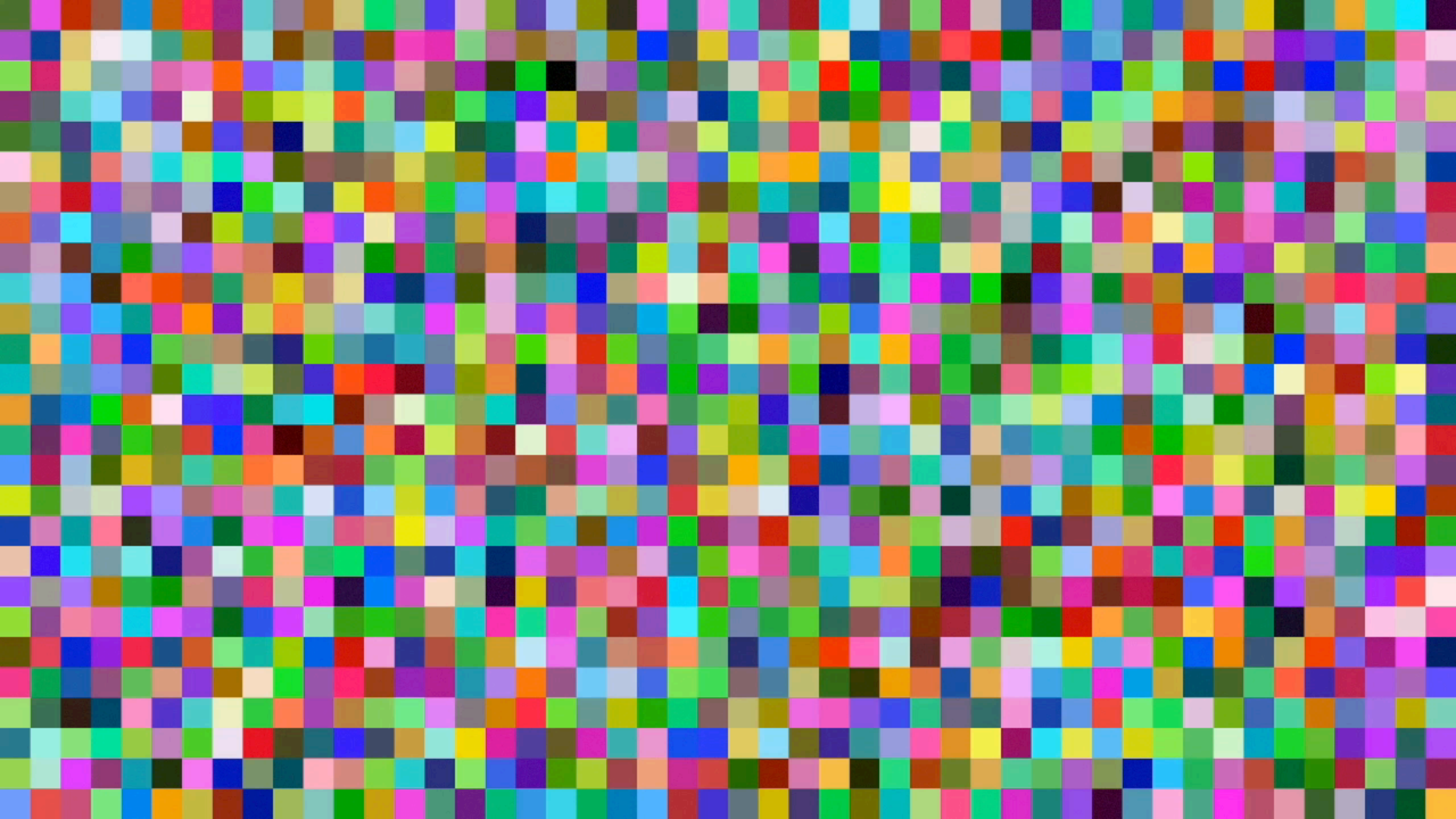
Outlier detection in log data

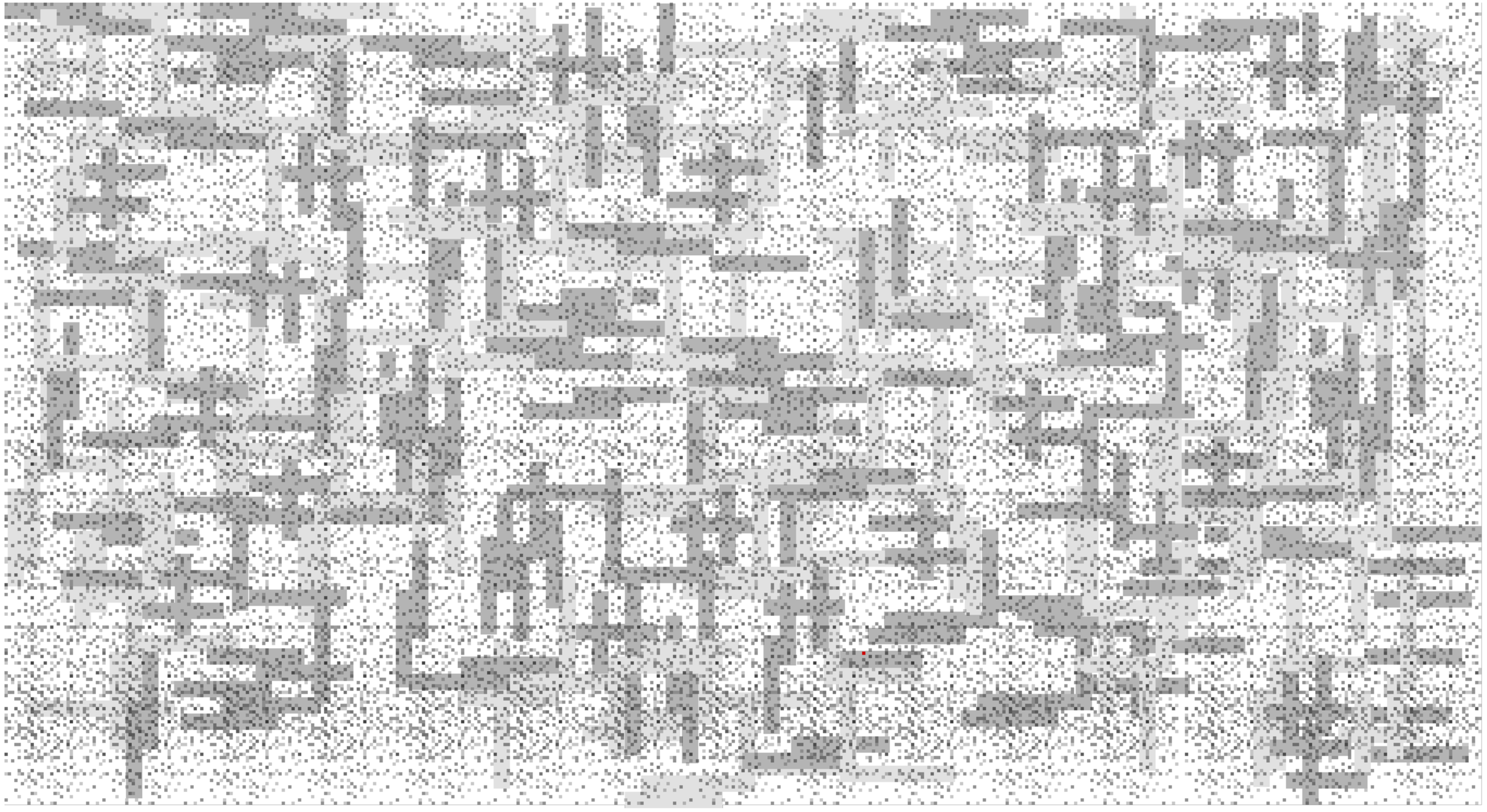


Outlier detection in log data

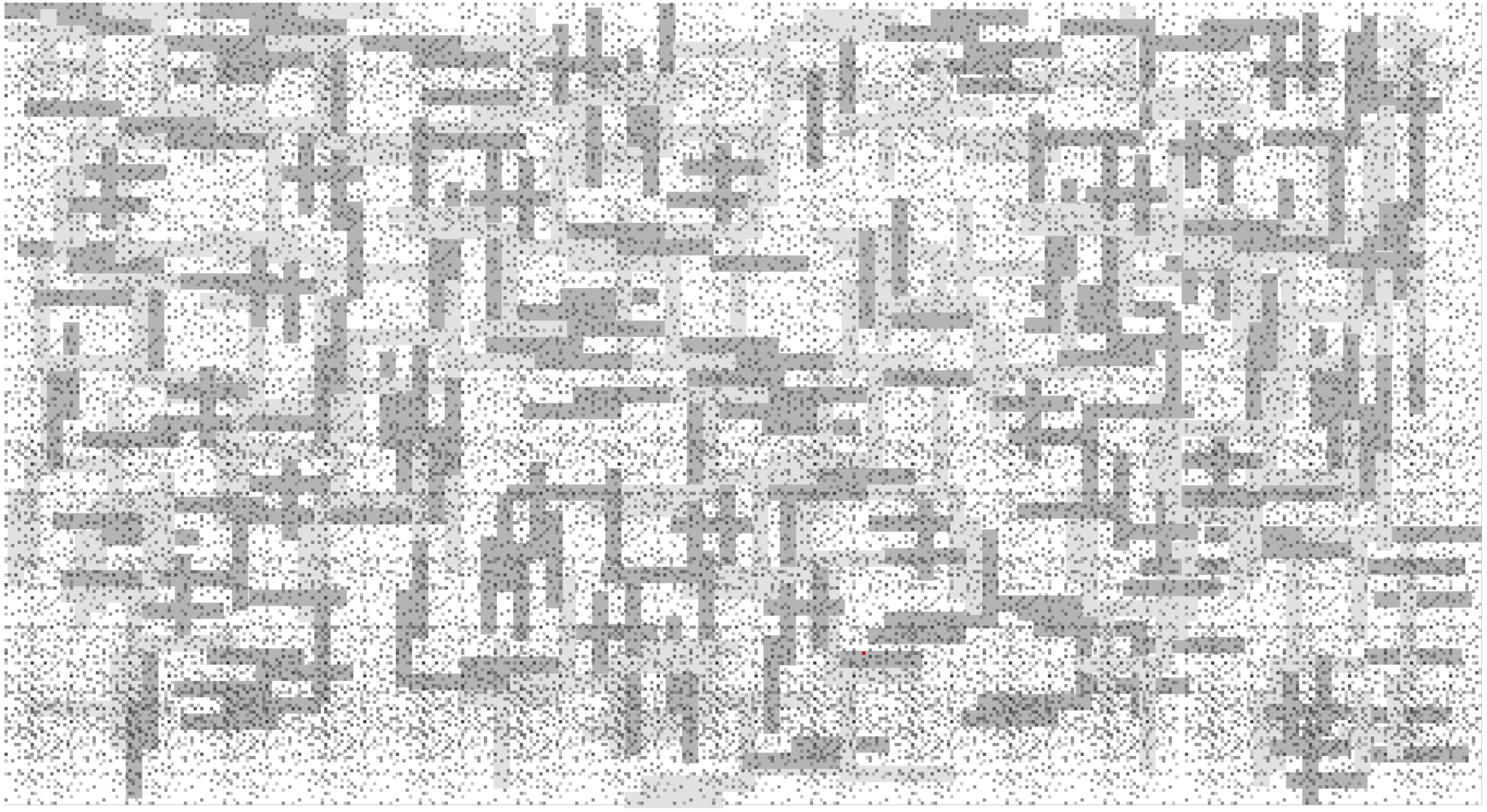


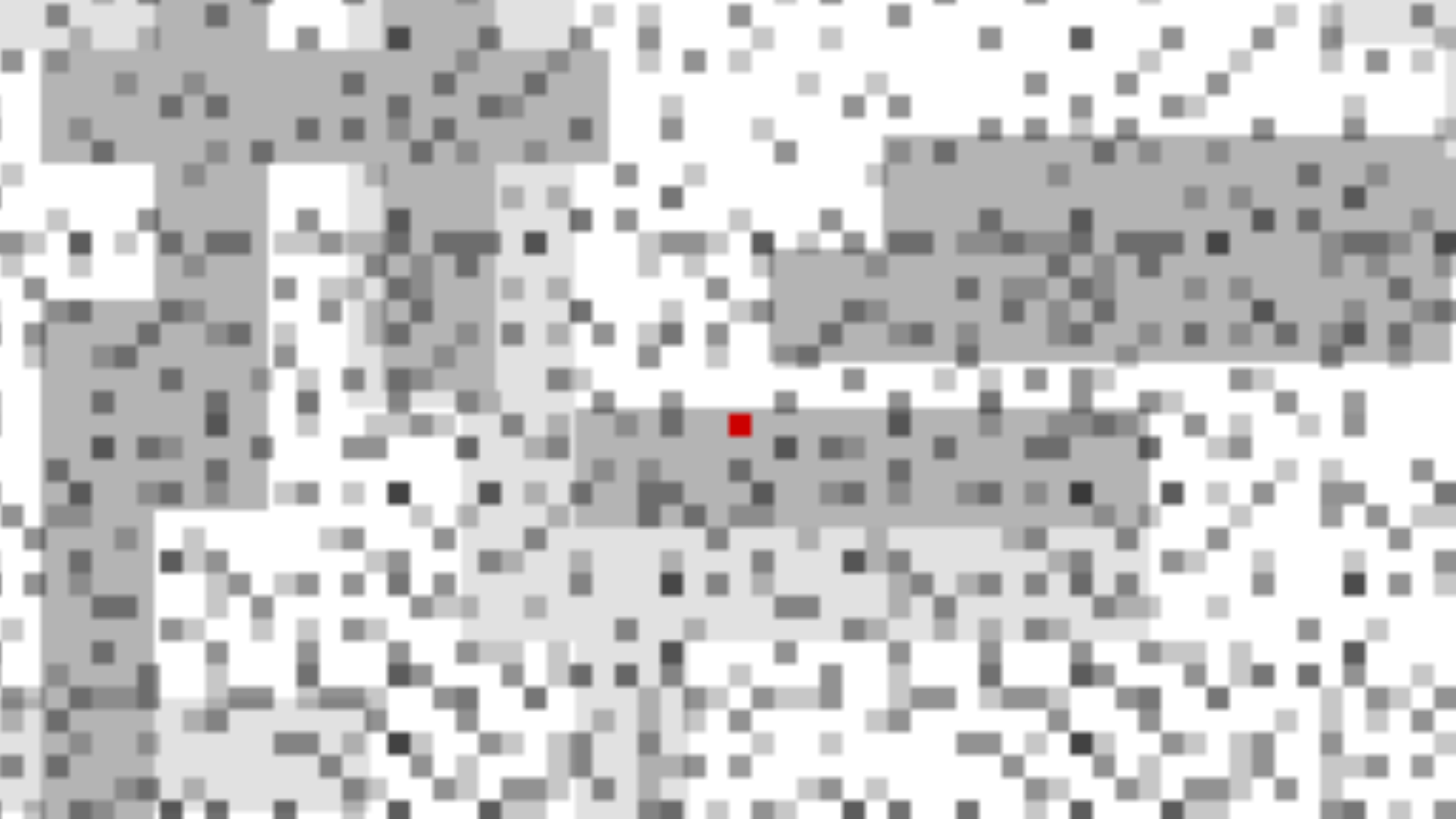







Out of 310 million log
records, we identified
0.0012% as outliers.

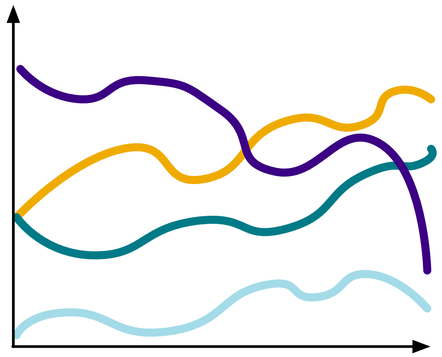




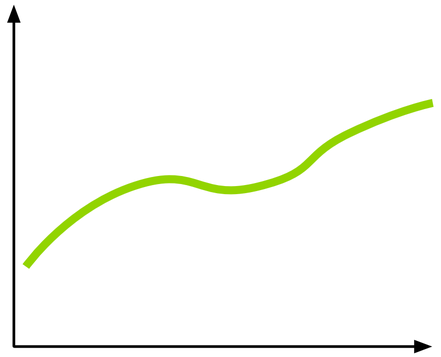
Thirty most extreme outliers

10	Can not communicate with power supply 2.
9	Power supply 2 failed.
8	Power supply redundancy is lost.
1	Drive A is removed.
1	Can not communicate with power supply 1.
1	Power supply 1 failed.

Modeling infrastructure costs



SYSTEM
METRICS



CLOUD
SPENDING

```
SELECT COUNT(value), MIN(value), MAX(value),
       AVG(value), items.key_, items.hostid
  FROM history, items
  WHERE history.itemid = items.itemid
  GROUP BY history.itemid
```

```
SELECT COUNT(value), MIN(value), MAX(value),
       AVG(value), items.key_, items.hostid
  FROM history, items
  WHERE history.itemid = items.itemid
  GROUP BY history.itemid
```

~120gb of data on
one node with 40 threads
and 384gb of RAM

```
SELECT COUNT(value), MIN(value), MAX(value),
       AVG(value), items.key_, items.hostid
  FROM history, items
  WHERE history.itemid = items.itemid
  GROUP BY history.itemid
```

~120gb of data on
one node with 40 threads
and 384gb of RAM

RDBMS: ~15 hours

```
SELECT COUNT(value), MIN(value), MAX(value),
       AVG(value), items.key_, items.hostid
  FROM history, items
  WHERE history.itemid = items.itemid
  GROUP BY history.itemid
```

~120gb of data on
one node with 40 threads
and 384gb of RAM

RDBMS: ~15 hours

Spark: ~15 minutes

Lessons we learned

METALESSON: HOW TO MASTER DECLARATIVE PROGRAMMING

Three steps to mastery

Understand the programming model.

Understand the execution model.

Understand when to let the environment work for you.

Three steps to mastery

"What does this mean?"

Understand the execution model.

Understand when to let the environment work for you.

Three steps to mastery

"What does this mean?"

"What does this do?"

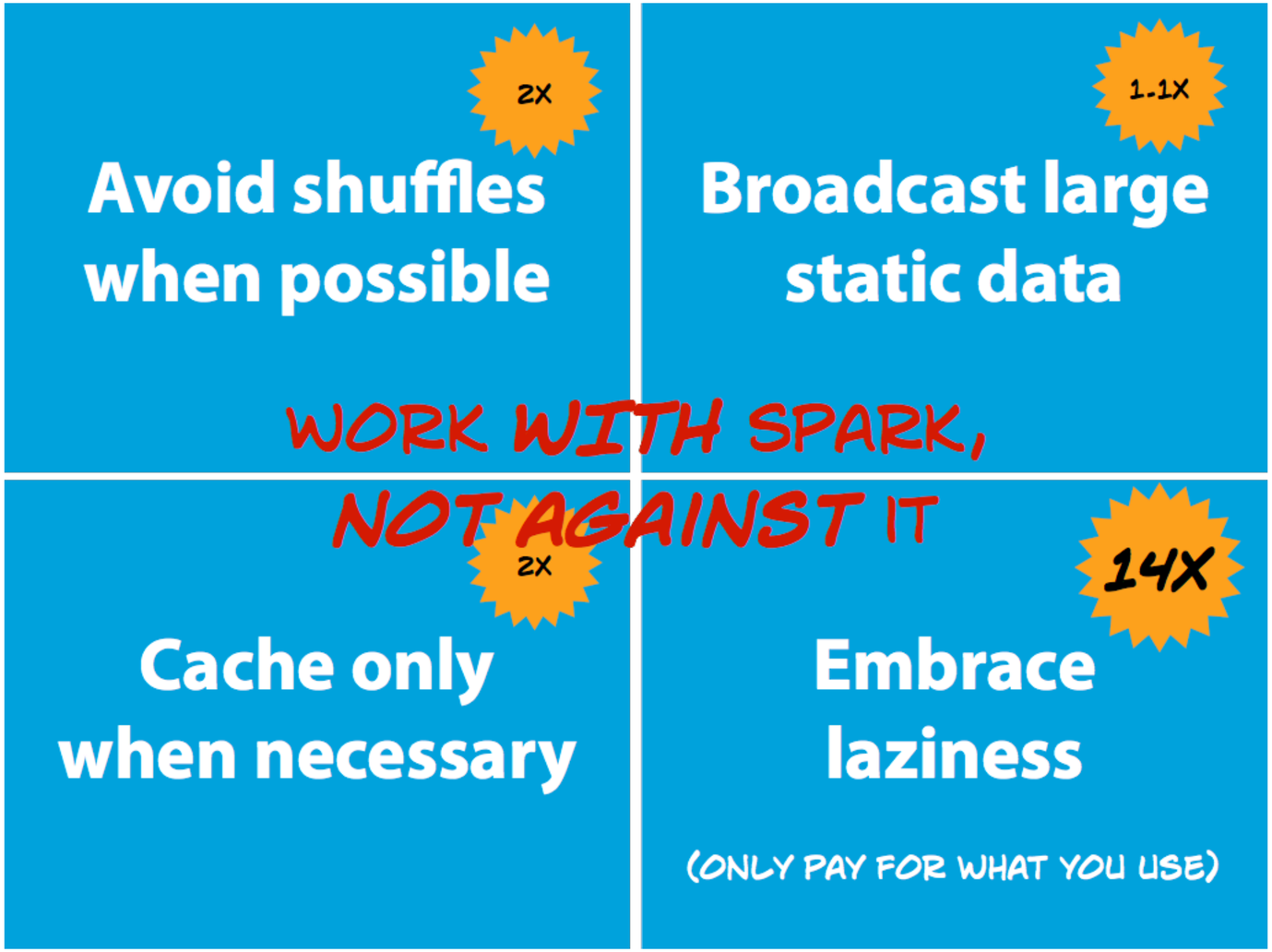
Understand when to let the environment work for you.

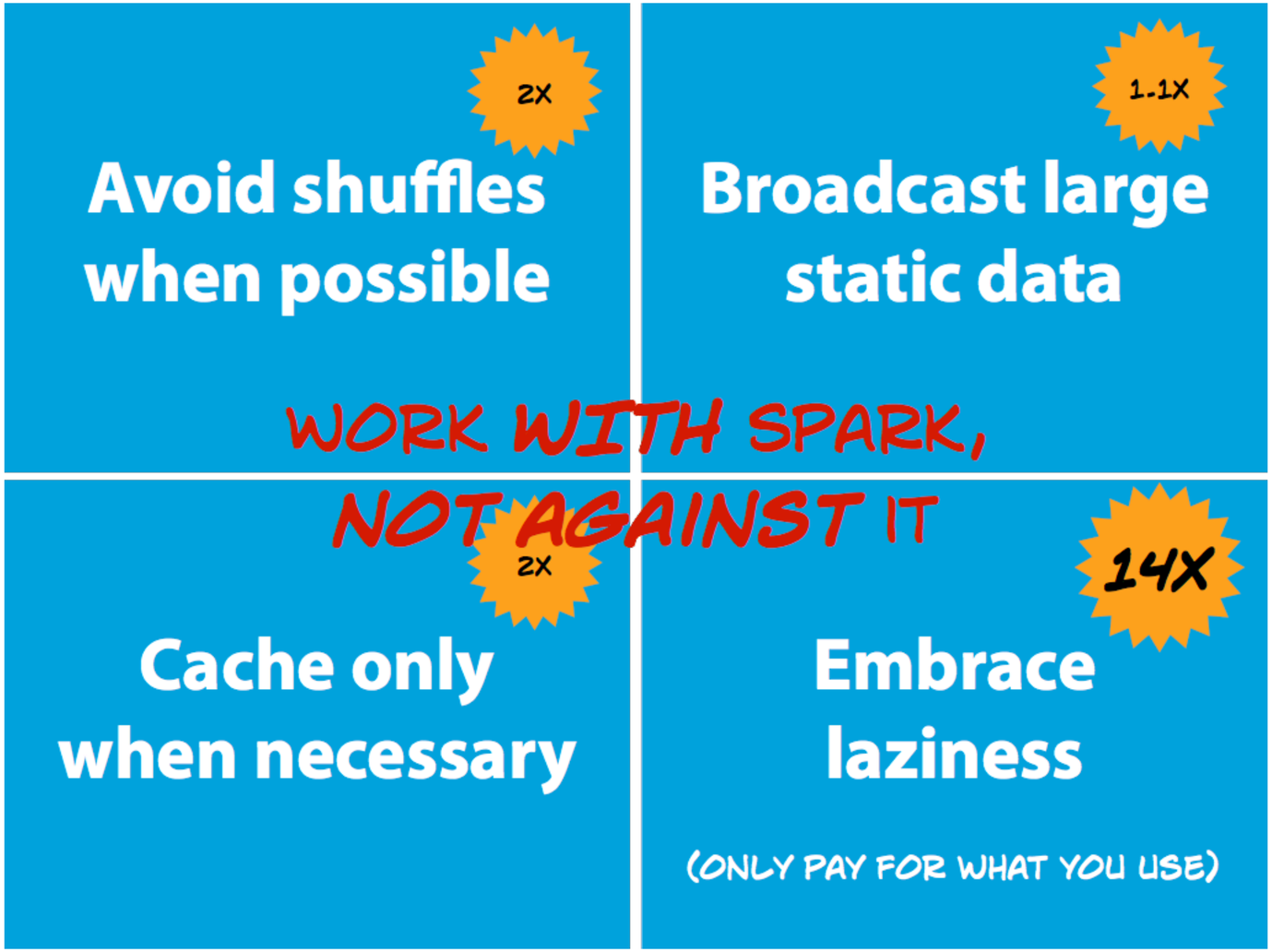
Three steps to mastery

"What does this mean?"

"What does this do?"

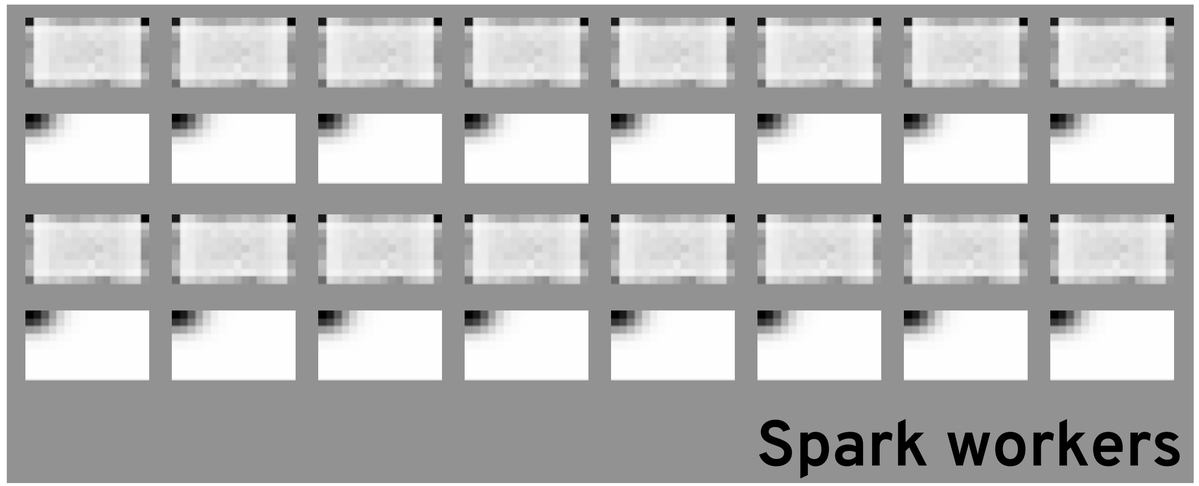
"How can I get out of its way?"


2X
Avoid shuffles when possible
1-1X
Broadcast large static data
WORK WITH SPARK,
NOT AGAINST IT
2X
Cache only when necessary
14X
Embrace laziness
(ONLY PAY FOR WHAT YOU USE)

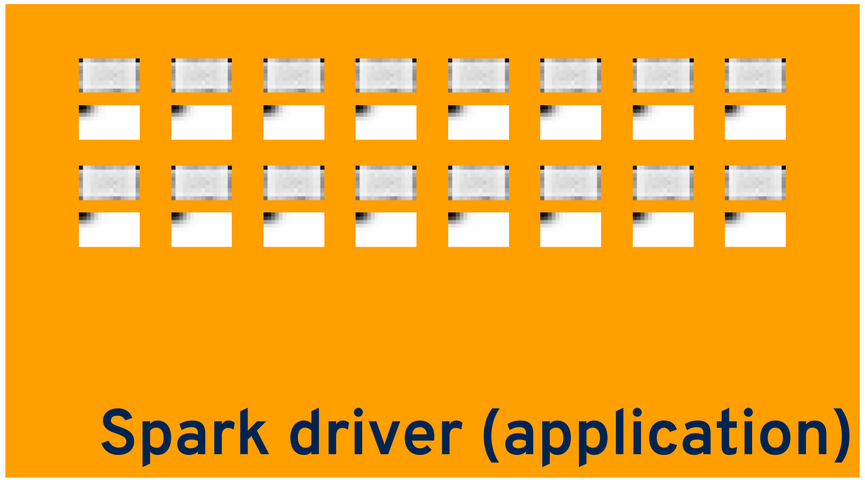

2X
Avoid shuffles when possible
1-1X
Broadcast large static data
WORK WITH SPARK, NOT AGAINST IT
2X
Cache only when necessary
14X
Embrace laziness
(ONLY PAY FOR WHAT YOU USE)

LESSON: LEARN THE API; USE THE RIGHT METHODS

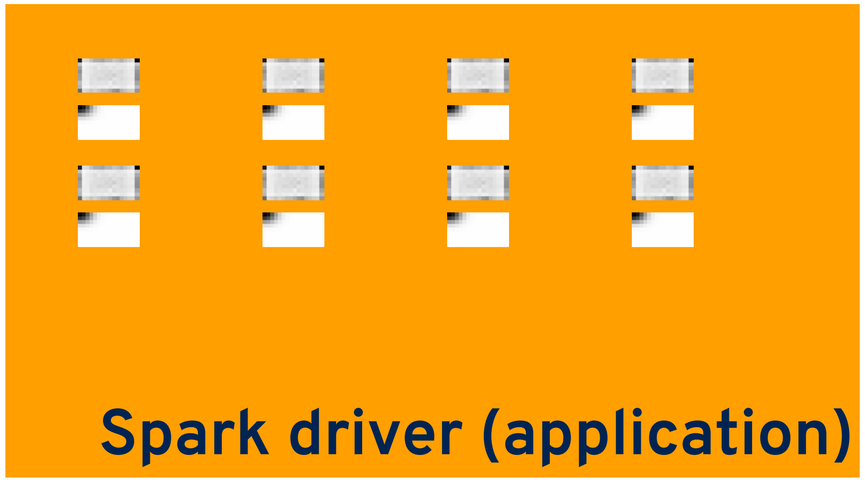
Take advantage of the model



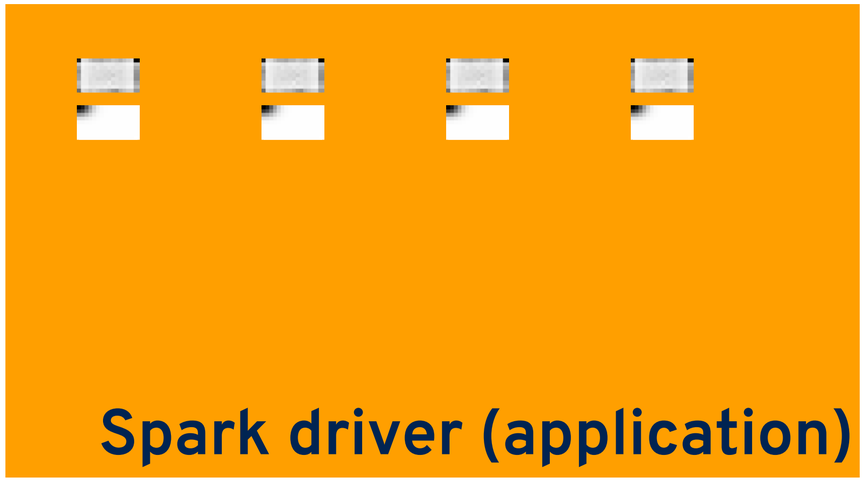
Take advantage of the model



Take advantage of the model



Take advantage of the model



Take advantage of the model

Take advantage of the model

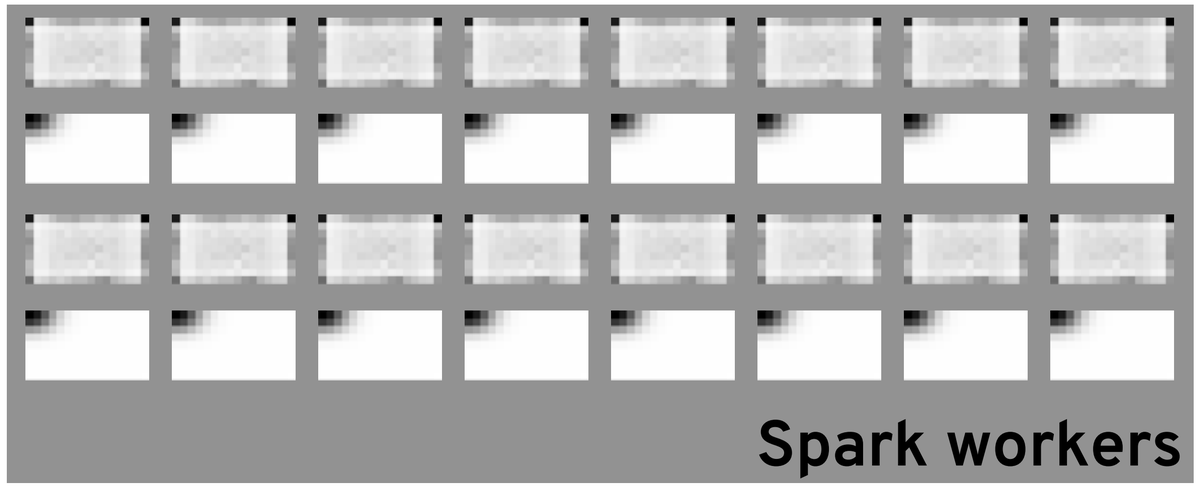
Take advantage of the model

Aggregating at the driver: increased memory pressure, decreased parallelism.

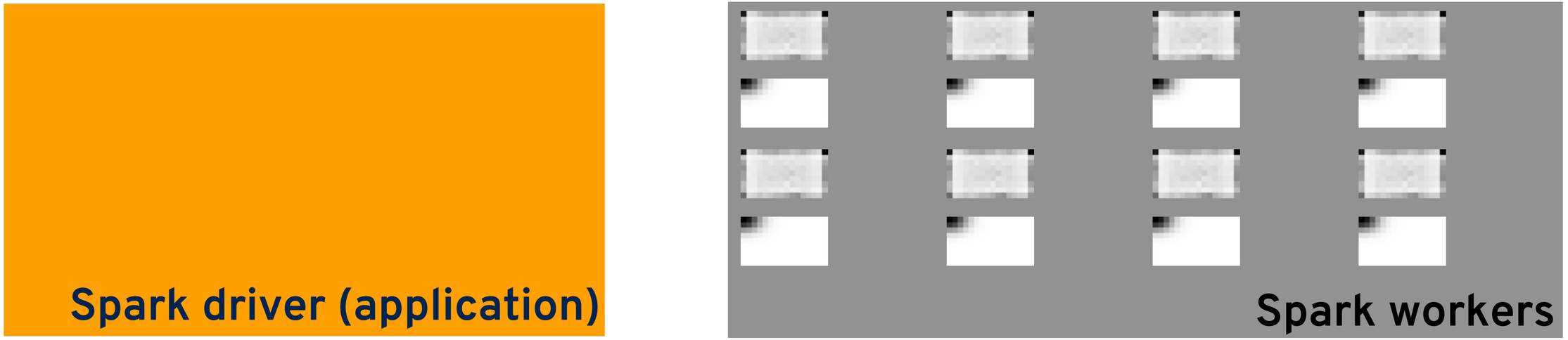
Spark driver (application)

Spark workers

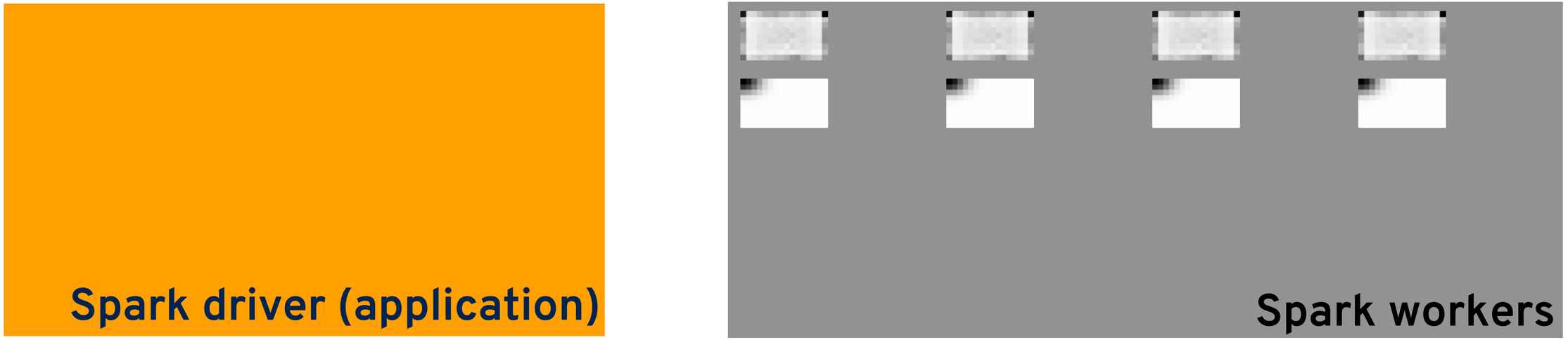
Distributed aggregation



Distributed aggregation



Distributed aggregation



Distributed aggregation

Distributed aggregation

Distributed aggregation

Distributed aggregation

Aggregate at the workers instead of in the driver!

LESSON: LET SPARK WORK FOR YOU WHENEVER POSSIBLE

Two favorite features

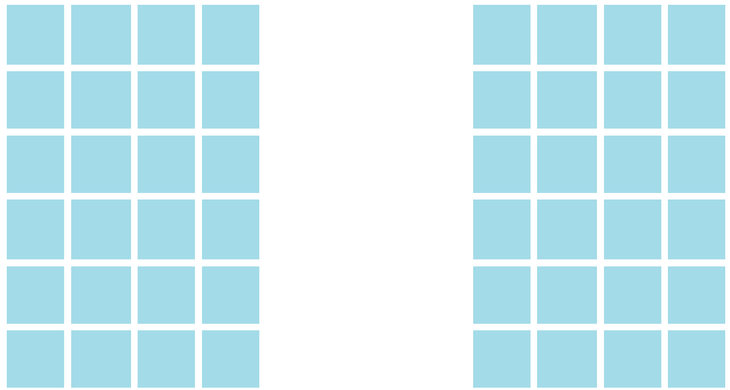
Query planning makes dumb code run *faster*.

Typed APIs prevent *really* dumb code from running *at all*.

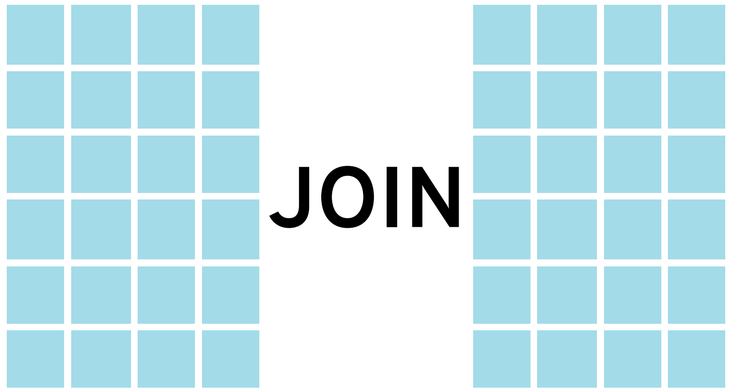
Query planning

```
SELECT * FROM A, B WHERE
  A.ID = B.ID AND
  uncommon(A.X) AND
  extremelyRare(B.Y)
```

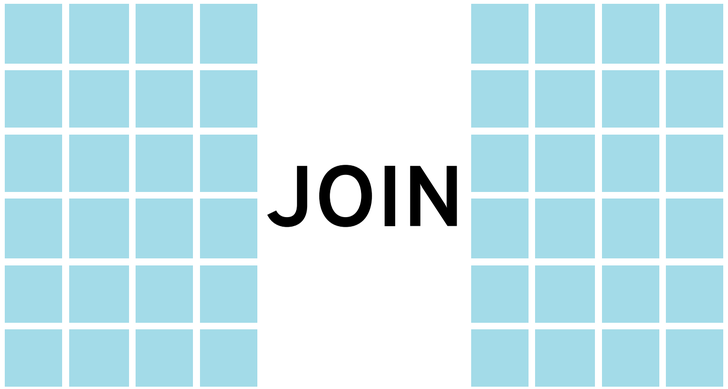
A naïve plan

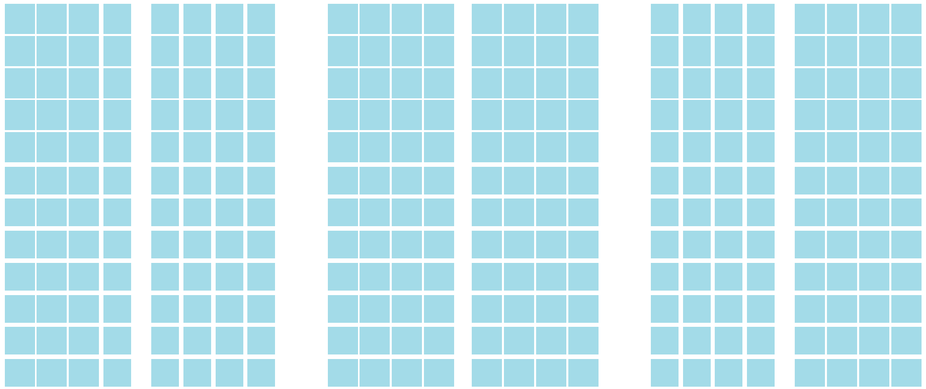


A naïve plan

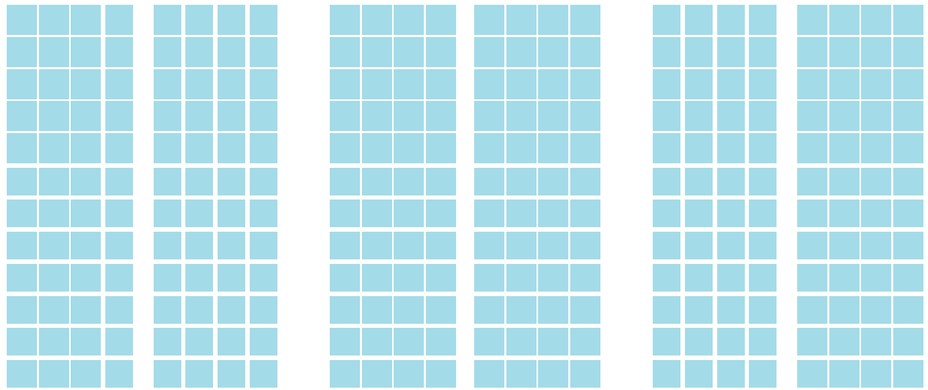


A naïve plan

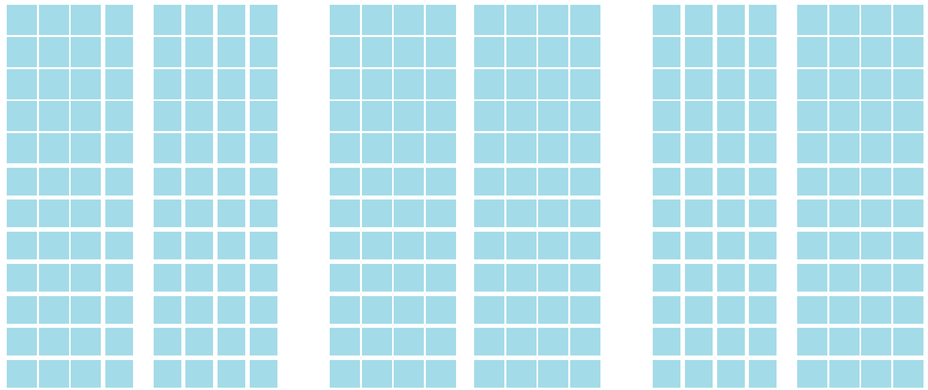




A naïve plan

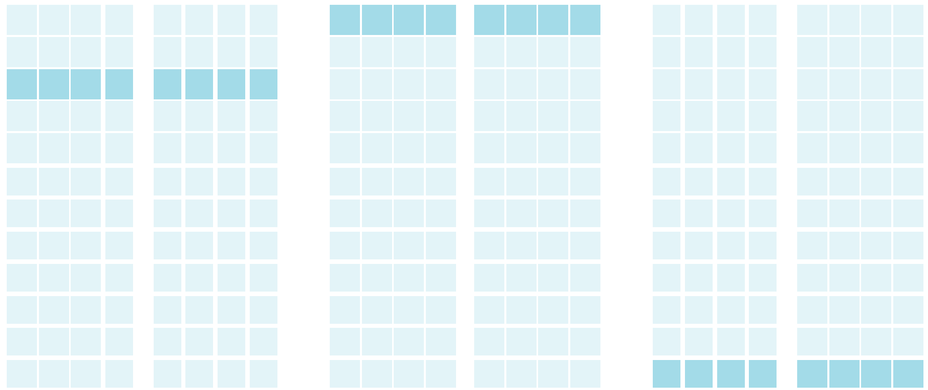


A naïve plan



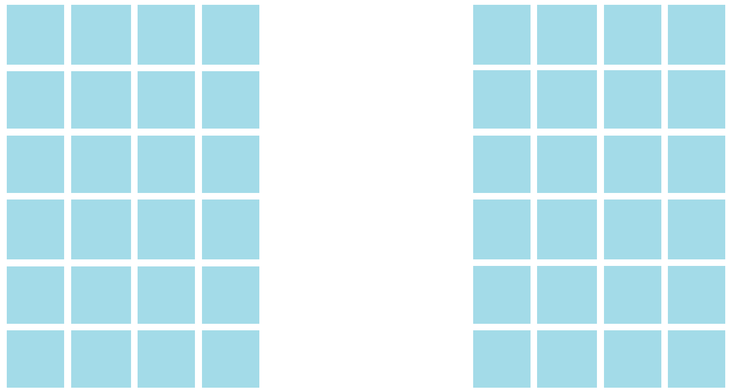
FILTER

A naïve plan

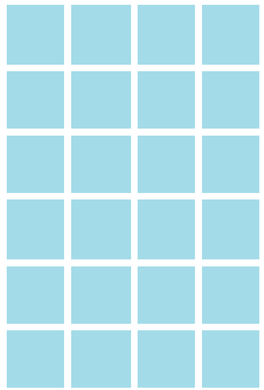


FILTER

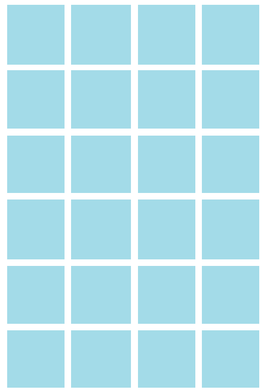
An optimized plan



An optimized plan

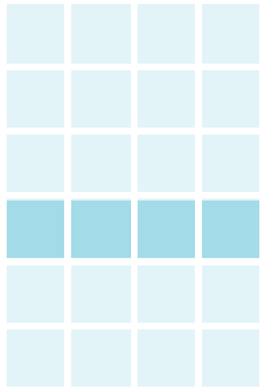


FILTER

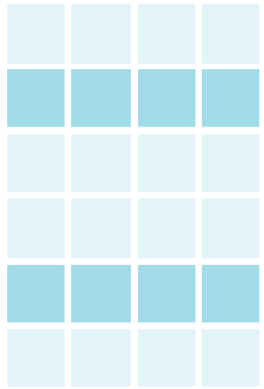


FILTER

An optimized plan

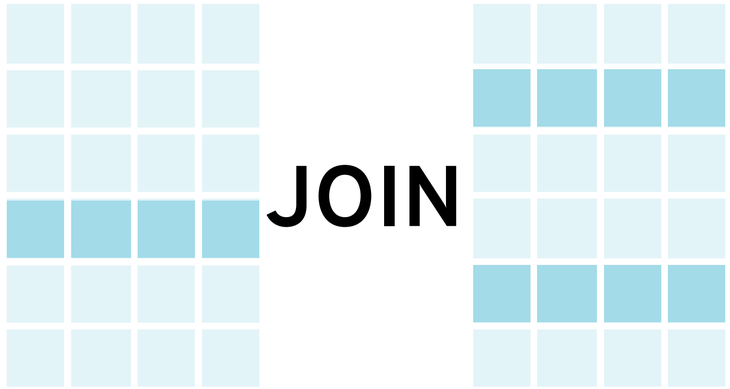


FILTER



FILTER

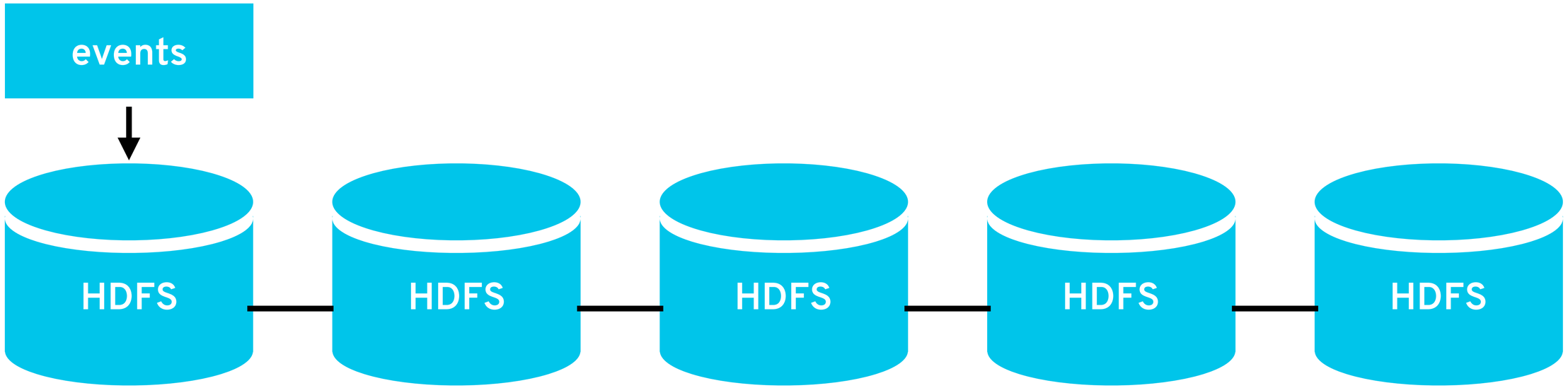
An optimized plan



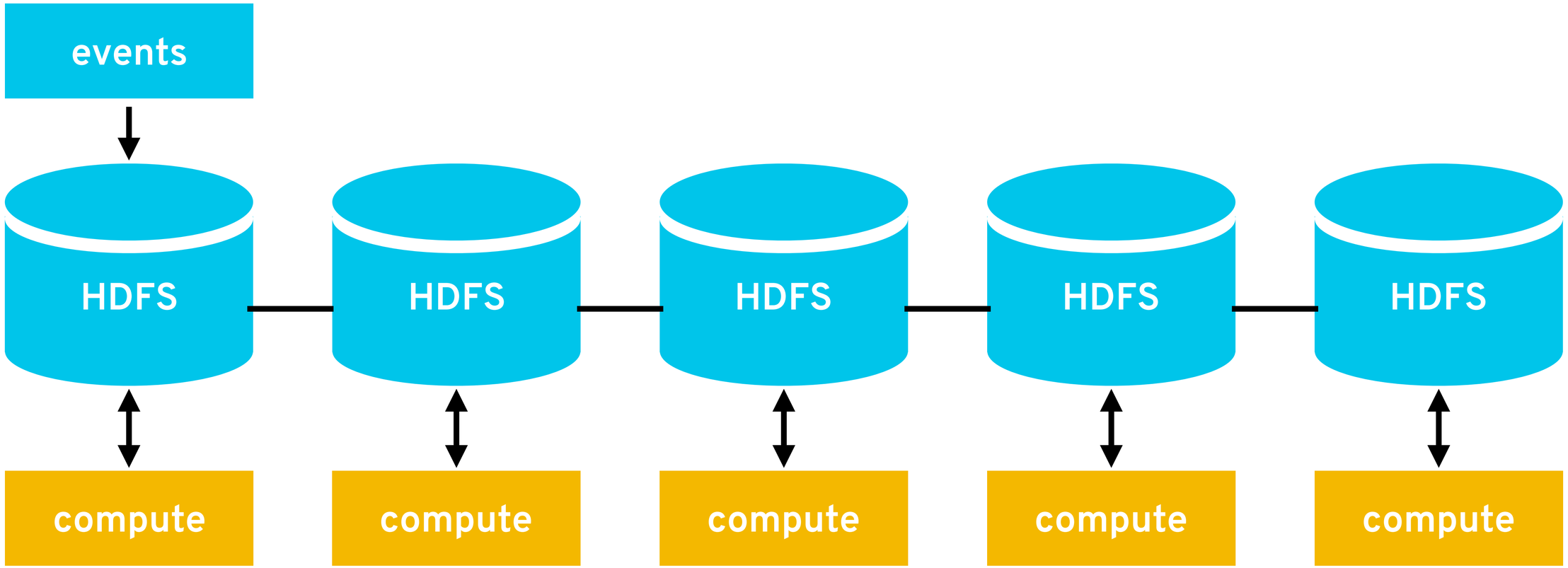
An optimized plan

LESSON: STORAGE FORMATS MATTER MORE THAN LOCALITY

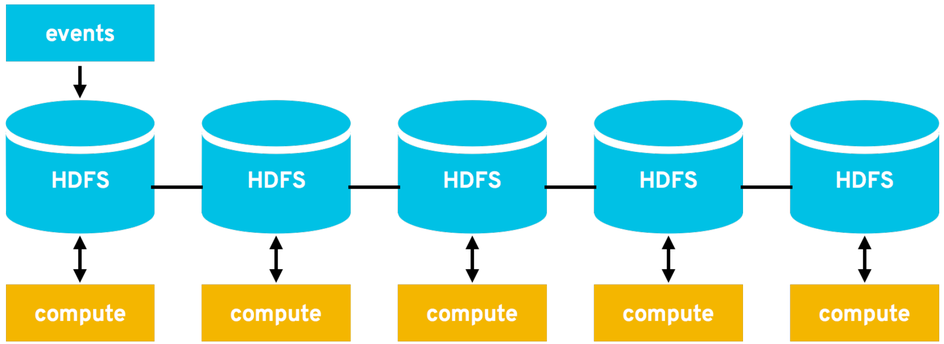
History lesson: Hadoop (2005)



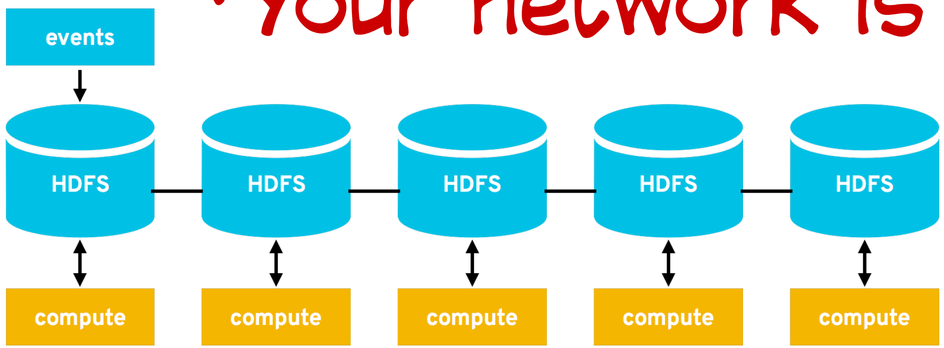
History lesson: Hadoop (2005)



History lesson: Hadoop (2005)



History lesson: Hadoop (2005)



"Your network is a bottleneck"

"Disks are too slow"

"Memories are too small"

"Locality is king"

“For the workloads from **Facebook** and **Bing**, we see that **96%** and **89%** of the active jobs respectively can have their **data entirely fit in memory**, given an allowance of 32GB memory per server for caching”

—“PACMan: Coordinated Memory Caching for Parallel Jobs.”
G. Ananthanarayanan et al., in *Proceedings of NSDI ’12*.

“**Recent studies have shown** that reading data from **local disks** is **only about 8% faster** than reading it from **remote disks** over the network … [and] **this 8% number is decreasing**.”

—Tom Phelan, “The Elephant in the Big Data Room: Data Locality is Irrelevant for Hadoop” (goo.gl/MnCKuM)

“**Three out of ten** hours of job runtime were spent **moving files** from the staging directory to the final directory **in HDFS**…
We were essentially **compressing, serializing, and replicating three copies** for a **single read.**”

—“Apache Spark @Scale: a 60+ TB production use case”
Facebook Engineering Blog Post

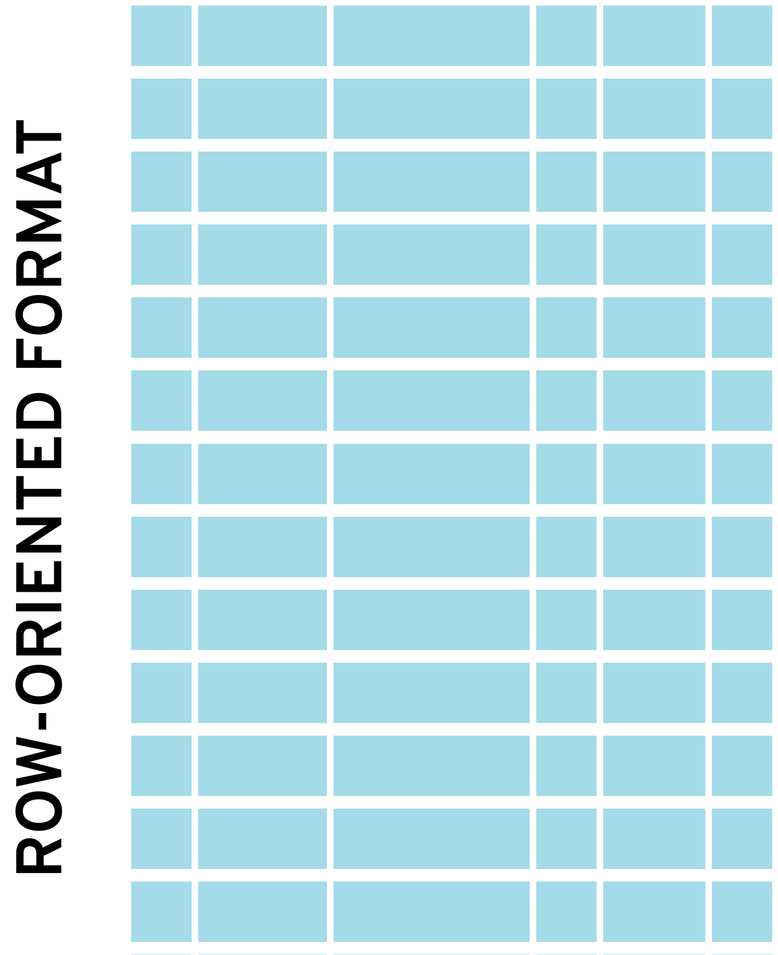
Do you need node locality?

Working set sizes **typically fit in cluster memory** even if raw data don’t.

I/O-heavy frameworks designed for colocated compute and storage **perform worse** than iterative processing in memory.

Colocating compute and storage **prevents independent scale-out of compute and storage**.

An optimization that matters

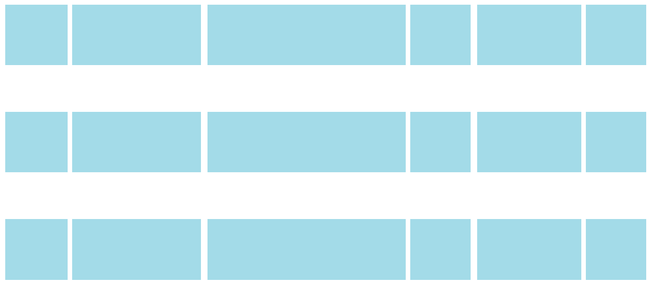


An optimization that matters

ROW-ORIENTED FORMAT

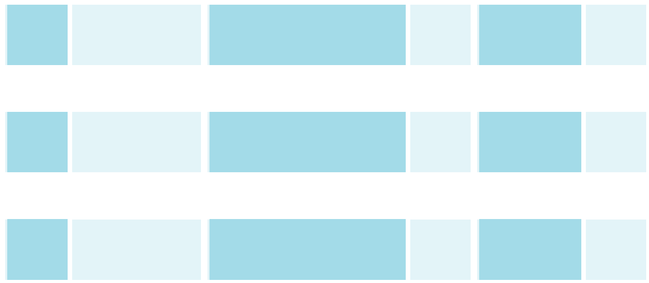
An optimization that matters

ROW-ORIENTED FORMAT



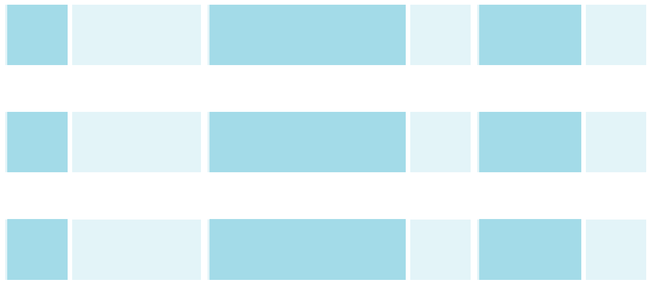
An optimization that matters

ROW-ORIENTED FORMAT



An optimization that matters

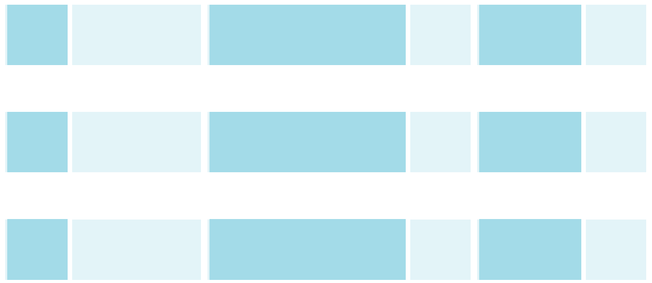
ROW-ORIENTED FORMAT



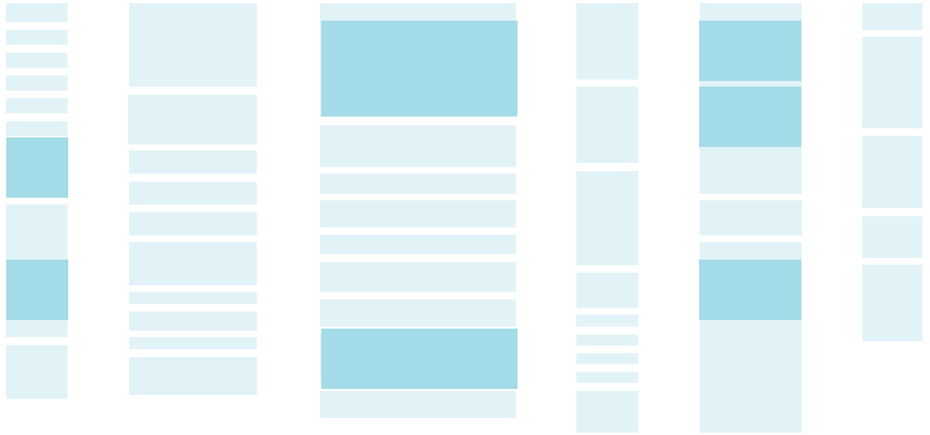
COLUMNAR FORMAT

An optimization that matters

ROW-ORIENTED FORMAT

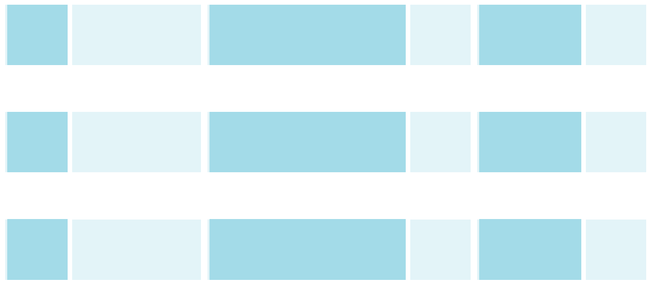


COLUMNAR FORMAT



An optimization that matters

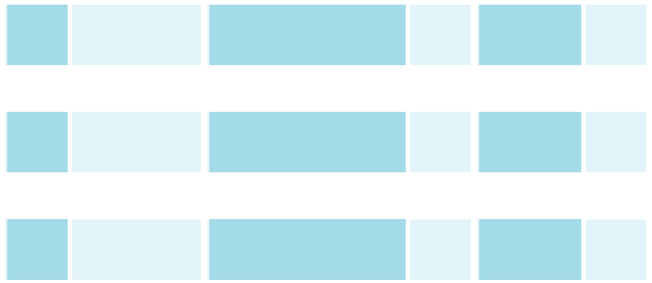
ROW-ORIENTED FORMAT



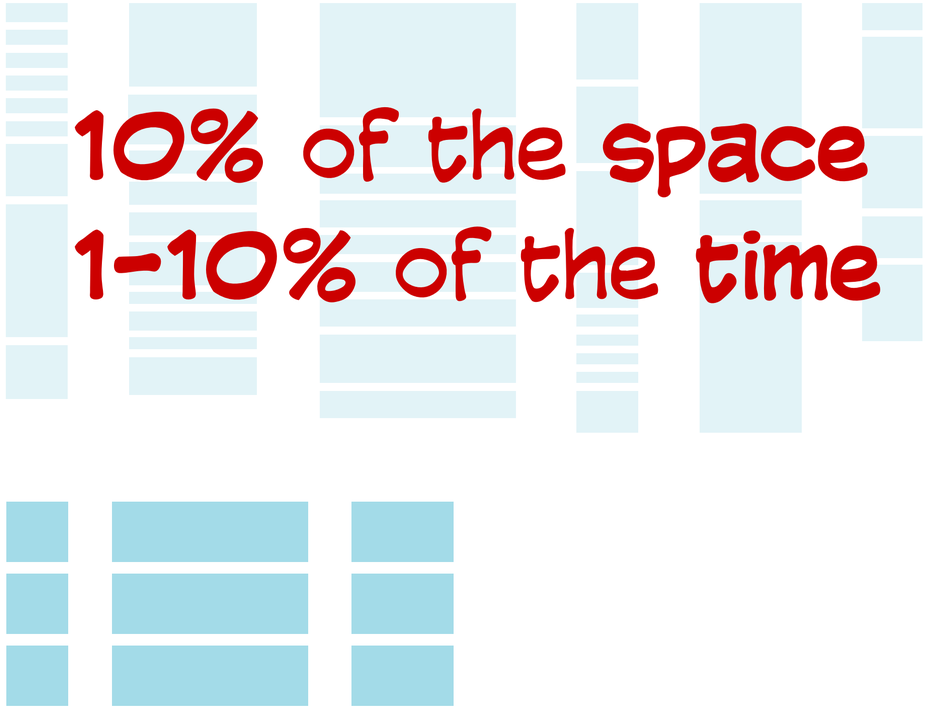
COLUMNAR FORMAT

An optimization that matters

ROW-ORIENTED FORMAT

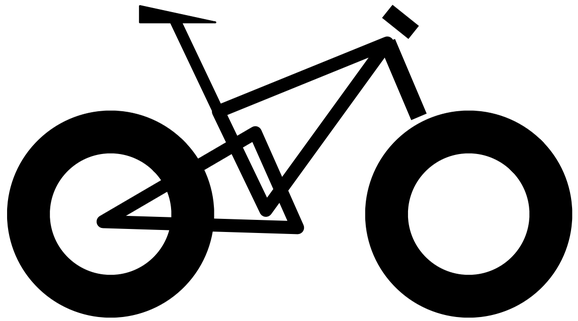


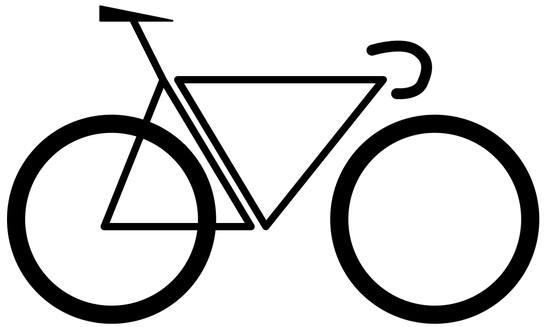
COLUMNAR FORMAT



LESSON: THINGS TO CONSIDER WHEN PREDICTING THE FUTURE

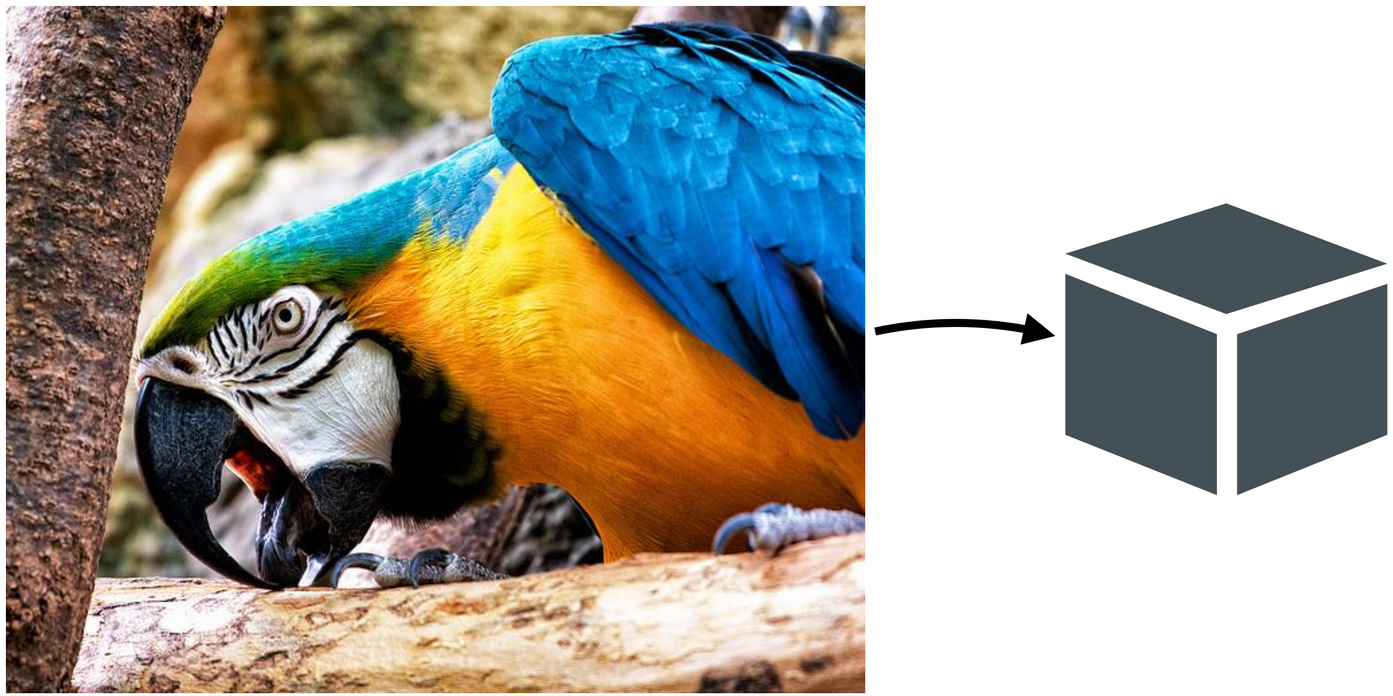
Feature engineering



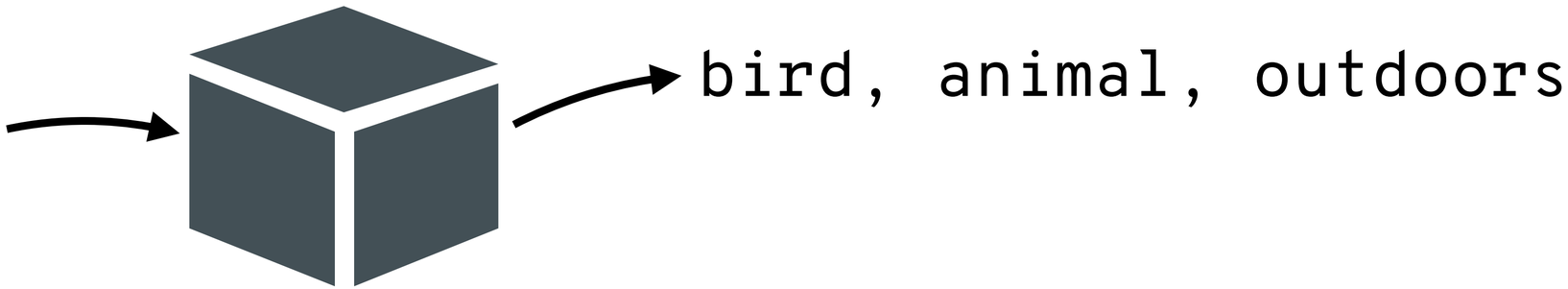


LABEL	DROP	FLAT	TIRE SIZE	TIRE KNOBS	FRONT	REAR
	HANDLEBAR TYPE				SUSPENSION?	
mountain bike	0	1	0.35	1	1	1
cyclocross bike	1	0	0.13	1	0	0

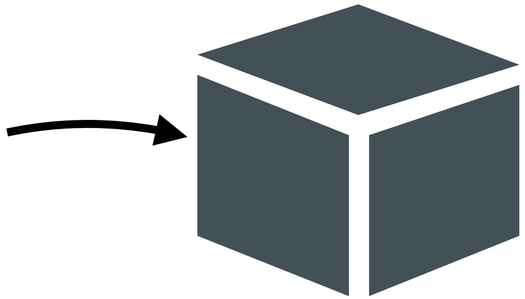
Interpretable models



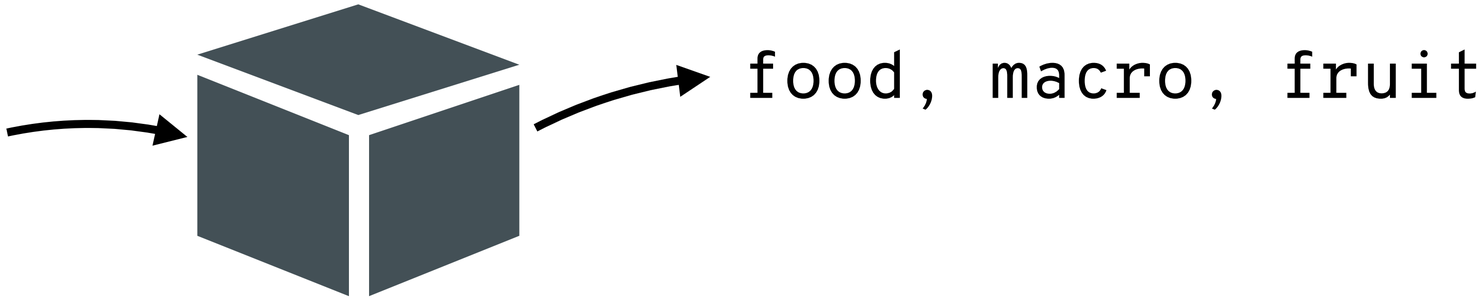
Interpretable models



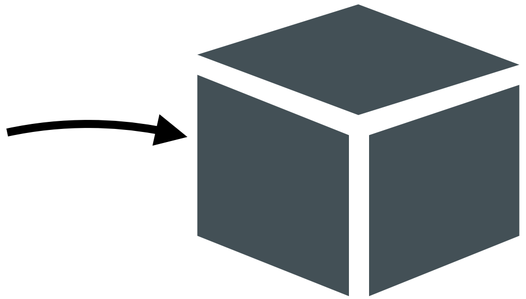
Interpretable models



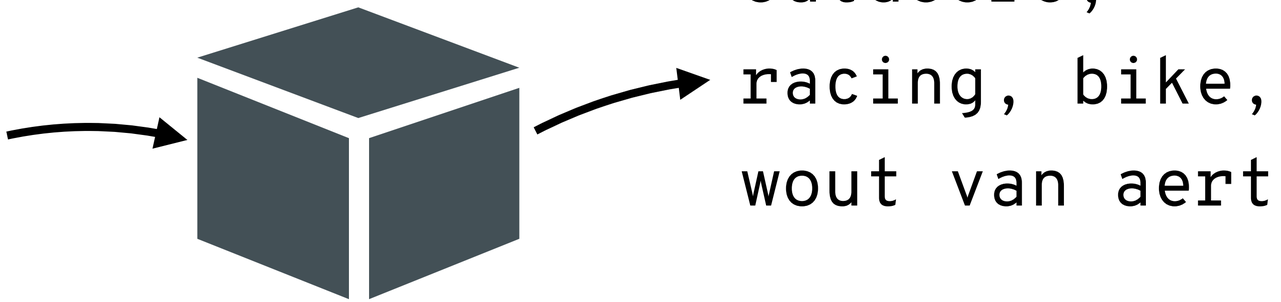
Interpretable models



Interpretable models

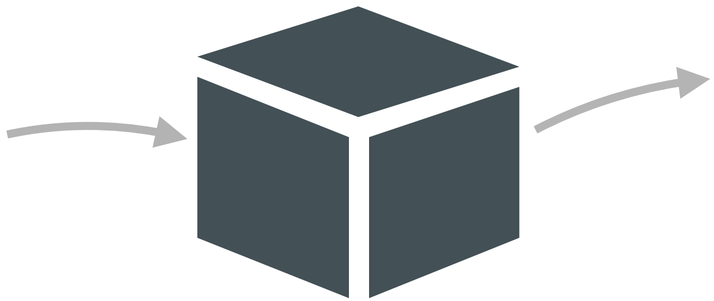


Interpretable models



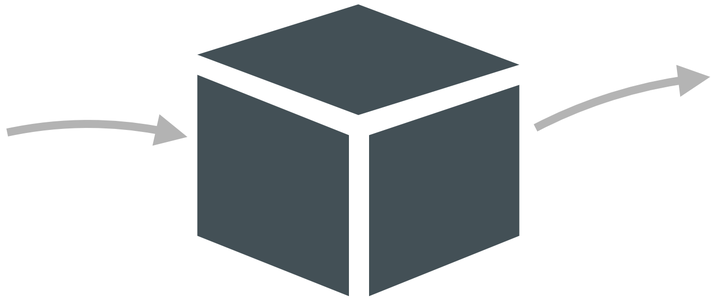
outdoors,
racing, bike,
wout van aert

Interpretable models



outdoors,
racing, bike,
wout van aert

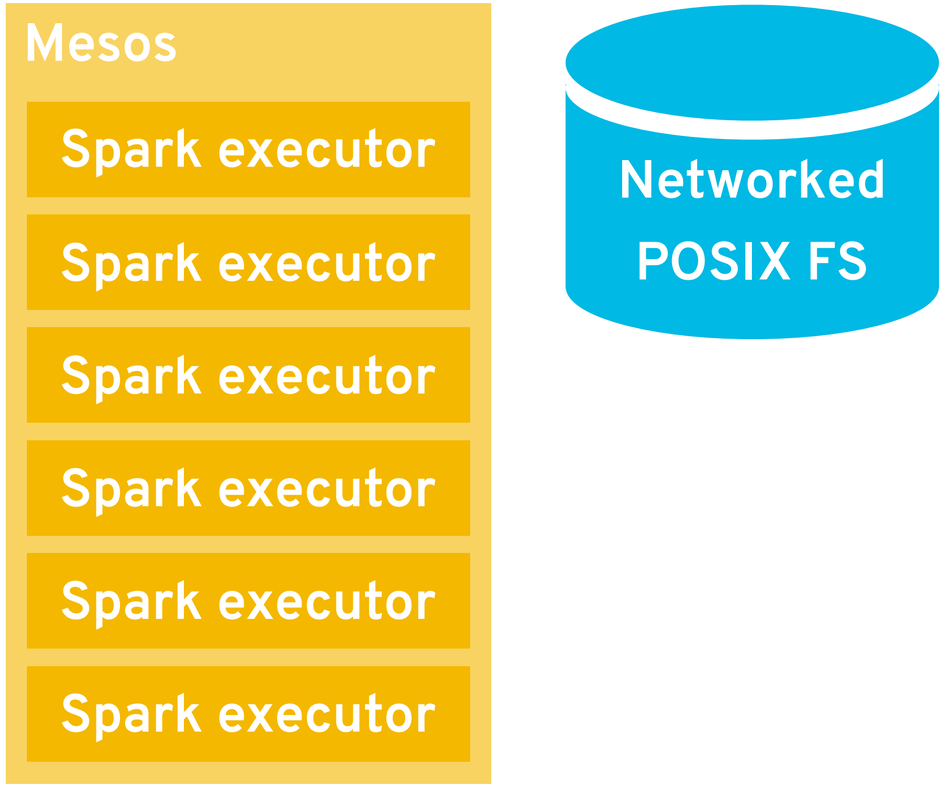
Interpretable models



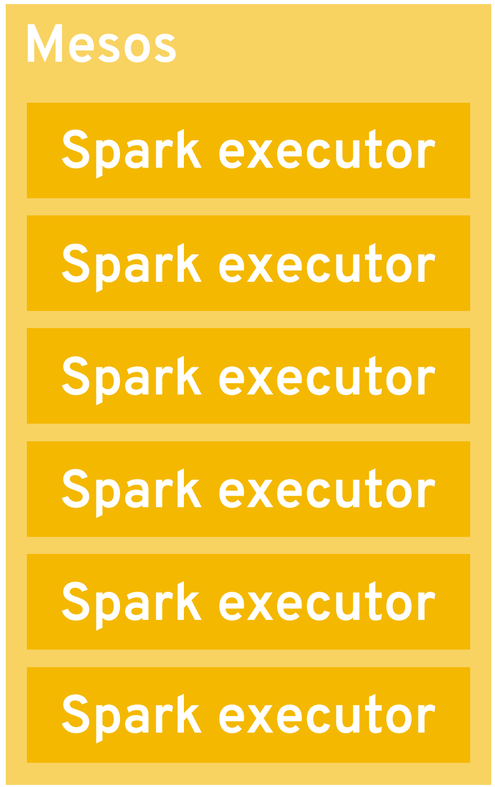
outdoors,
racing, bike,
wout van aert

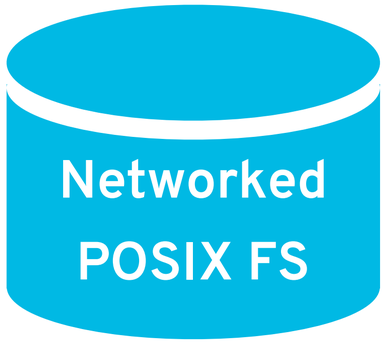
Accurate predictions are only part of a model's value!

From analytics as a workload to insightful applications

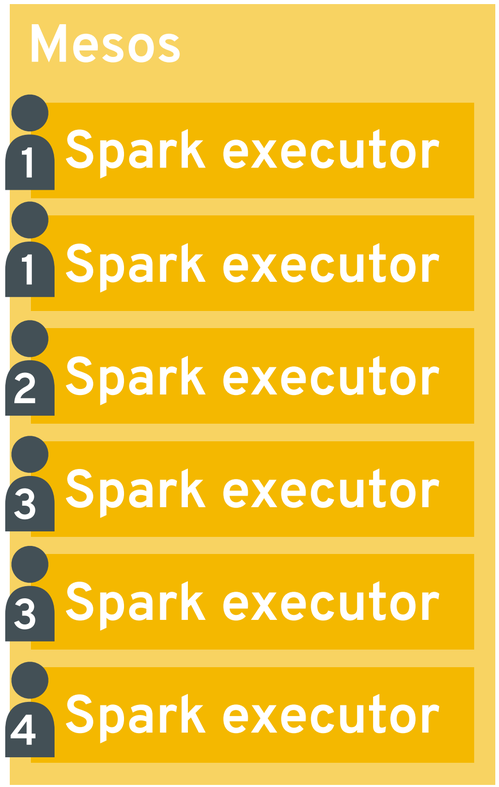

Mesos
Spark executor
Spark executor
Spark executor
Spark executor
Spark executor
Spark executor
Networked
POSIX FS

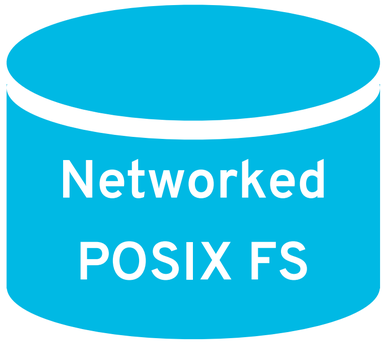

1
2
3
4

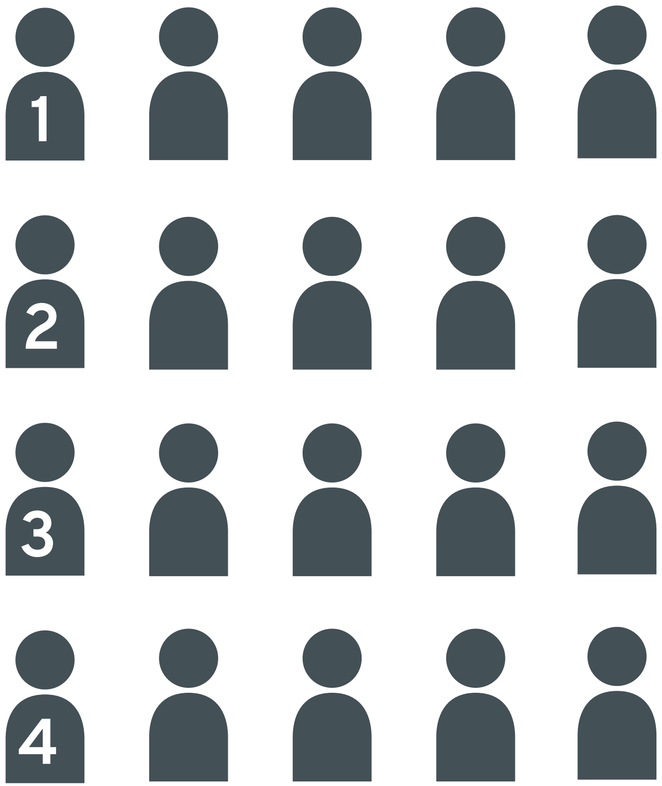

Mesos
Spark executor
Spark executor
Spark executor
Spark executor
Spark executor
Spark executor


Networked
POSIX FS


1
2
3
4


Mesos
1
Spark executor
1
Spark executor
2
Spark executor
3
Spark executor
3
Spark executor
4
Spark executor


Networked
POSIX FS



Mesos

1 Spark executor

1 Spark executor

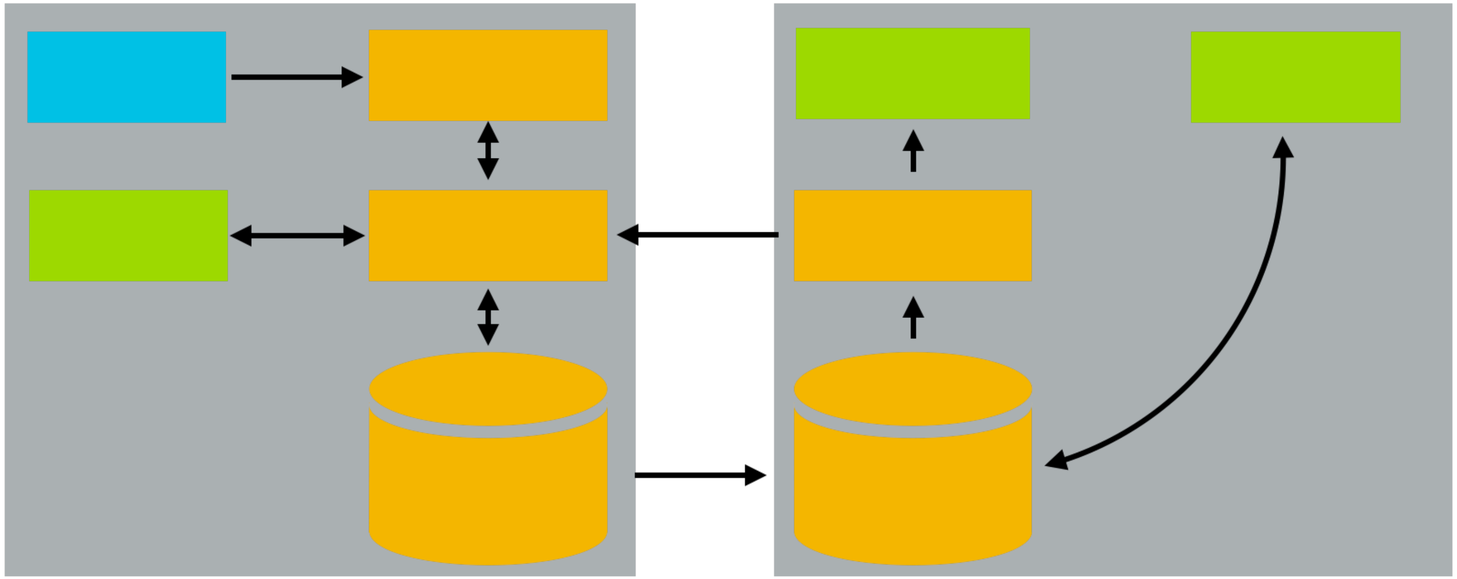
2 Spark executor

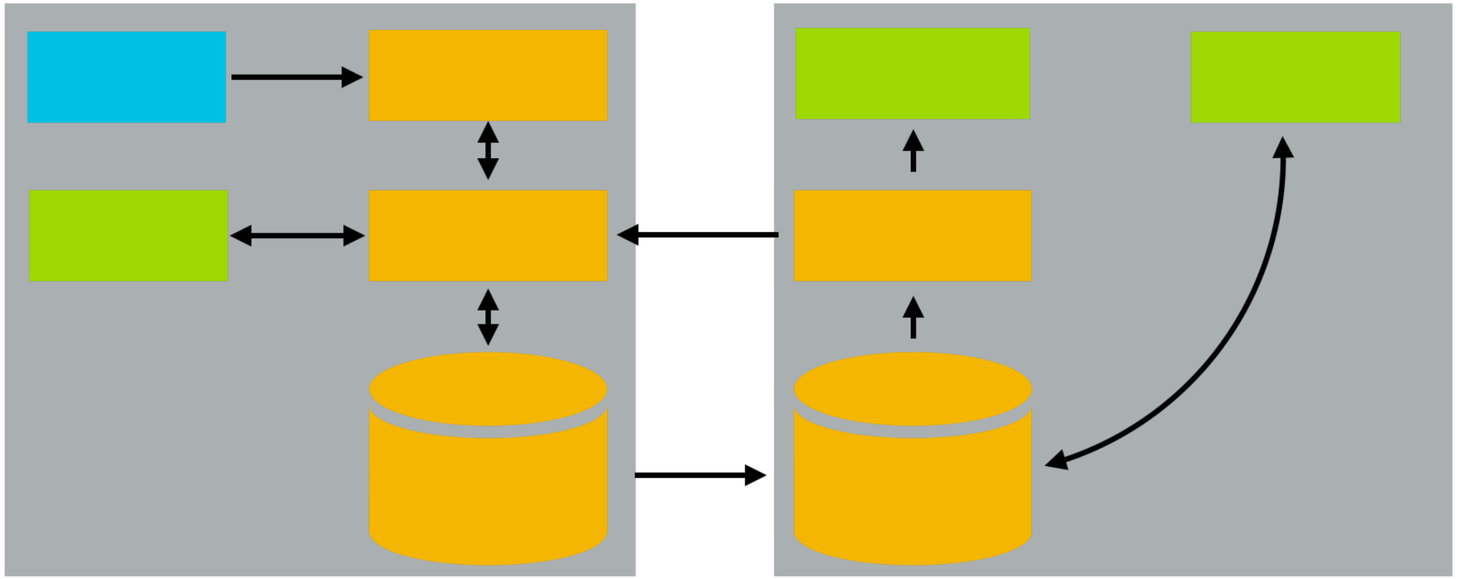
3 Spark executor

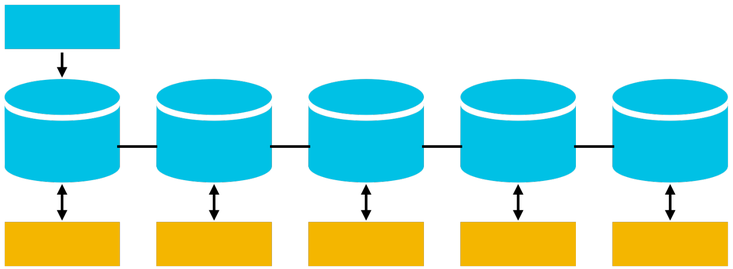
3 Spark executor

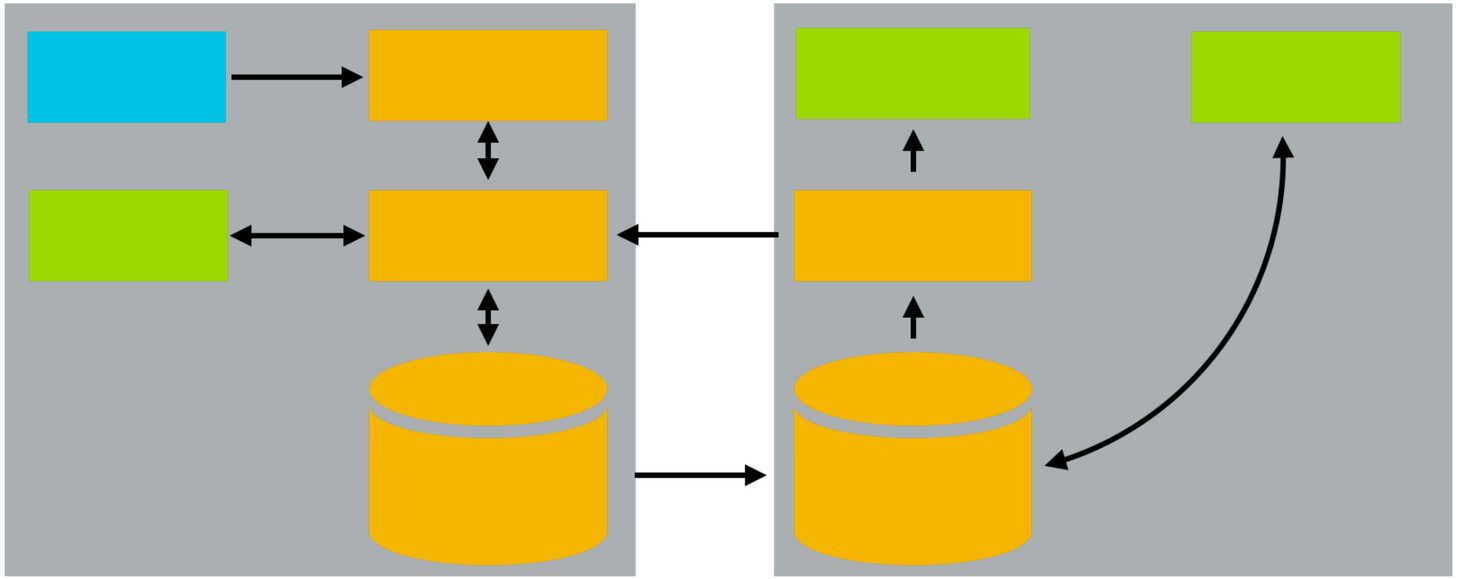
4 Spark executor

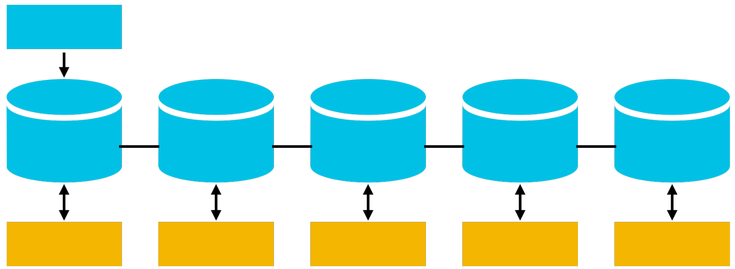
Networked
POSIX FS

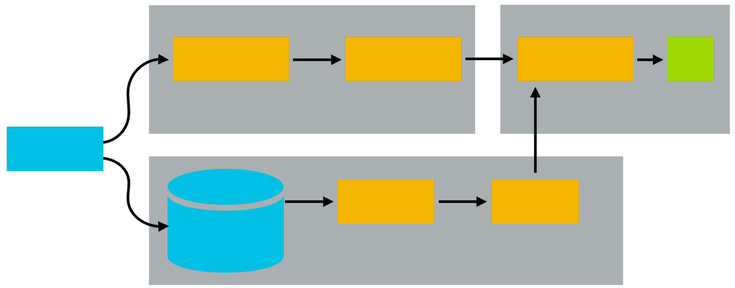


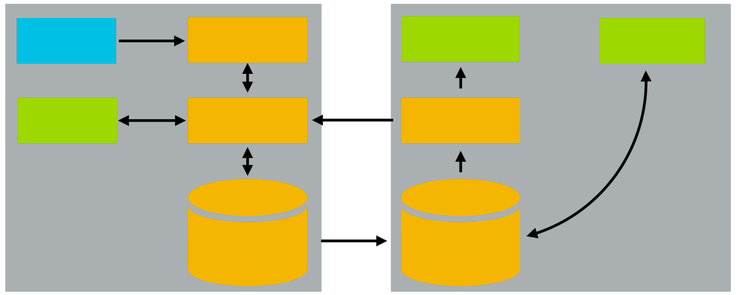


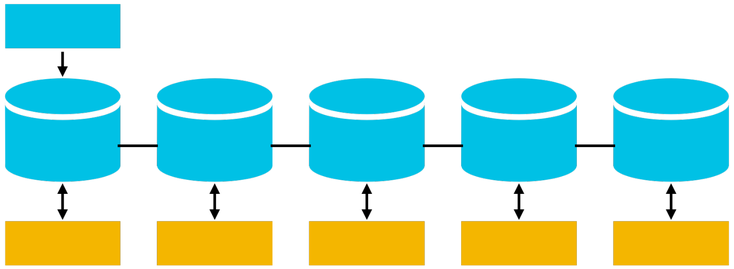


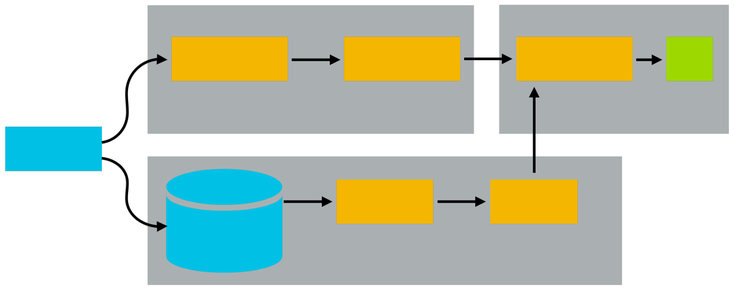


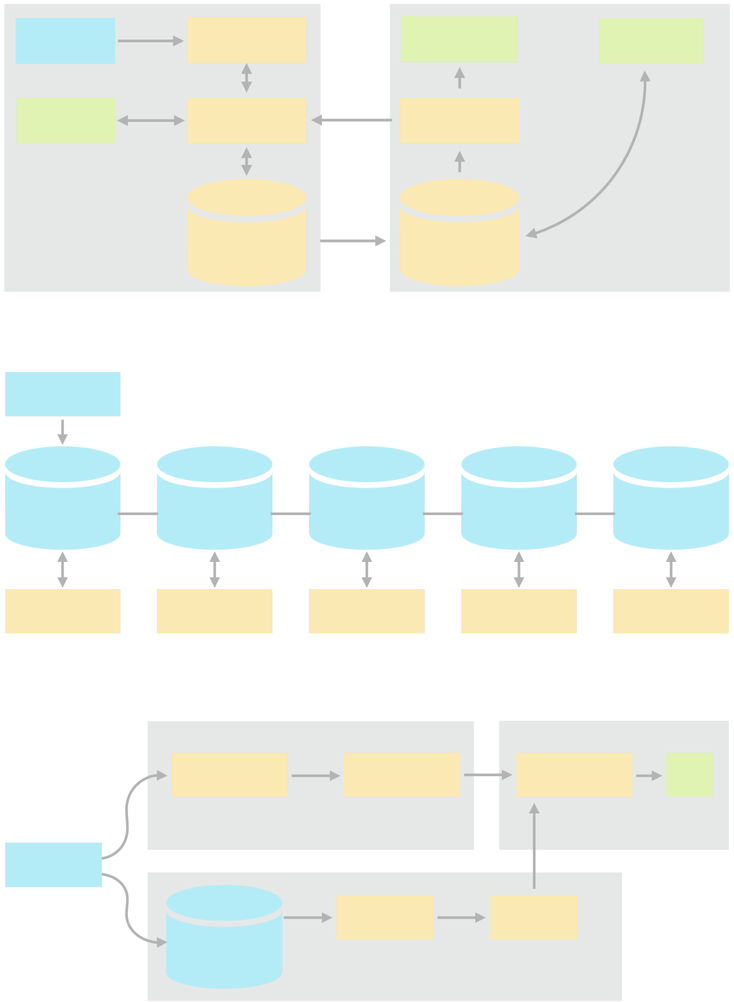


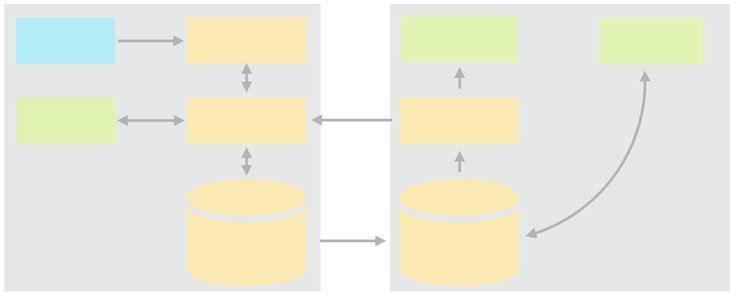


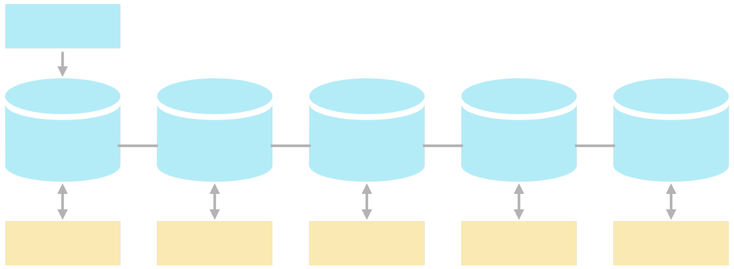


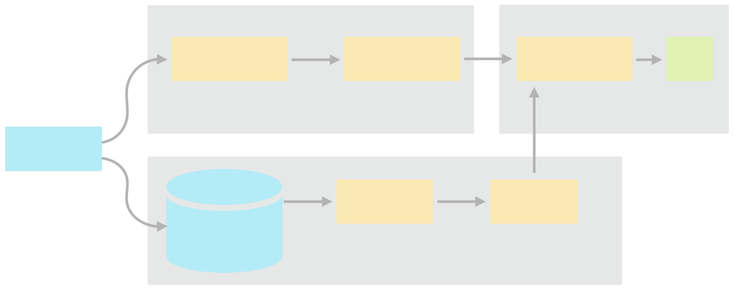


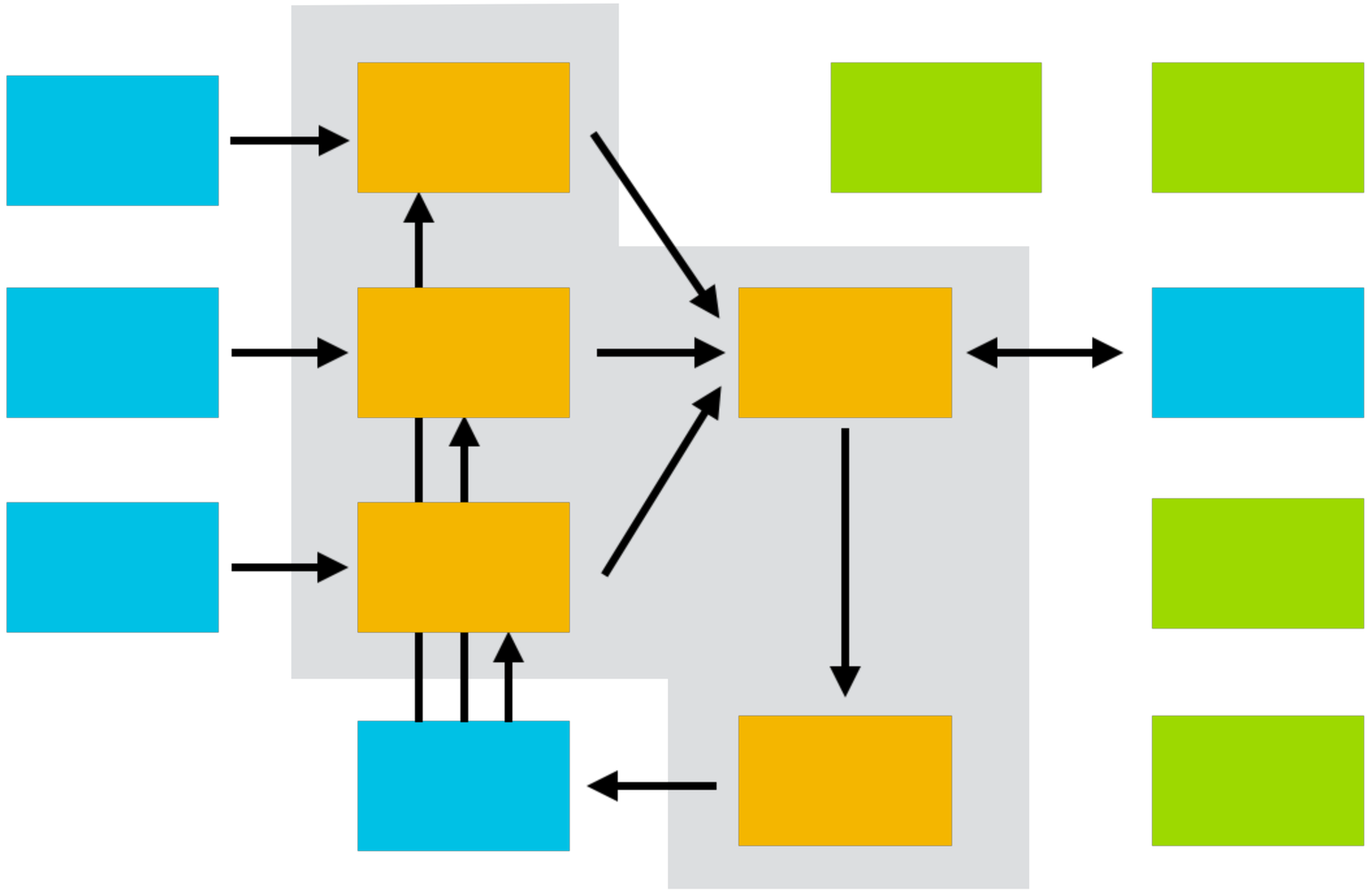


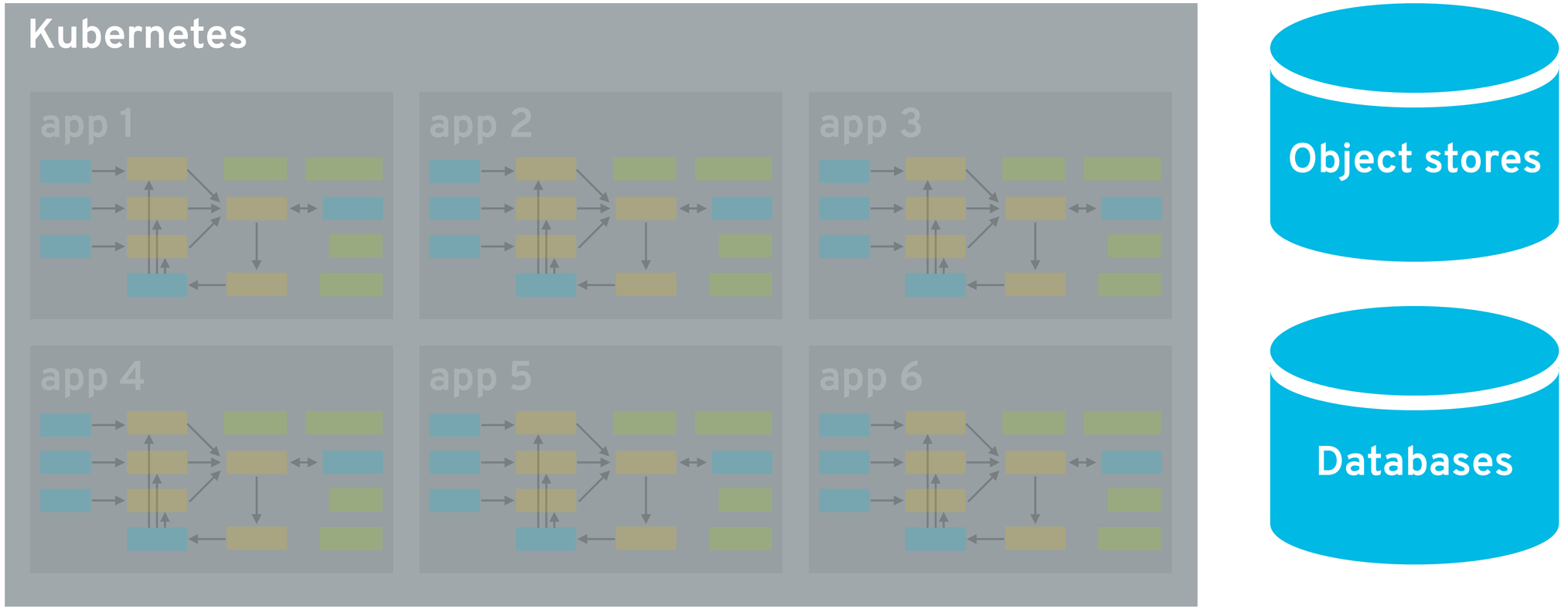


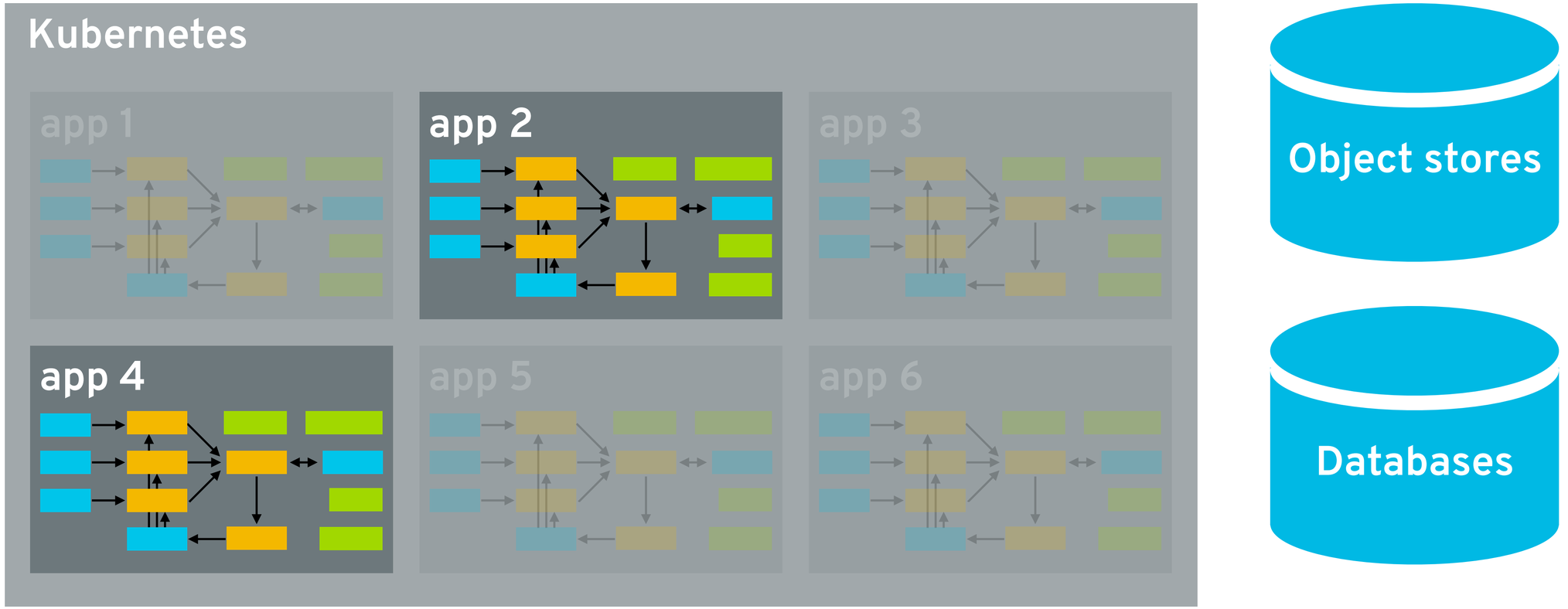






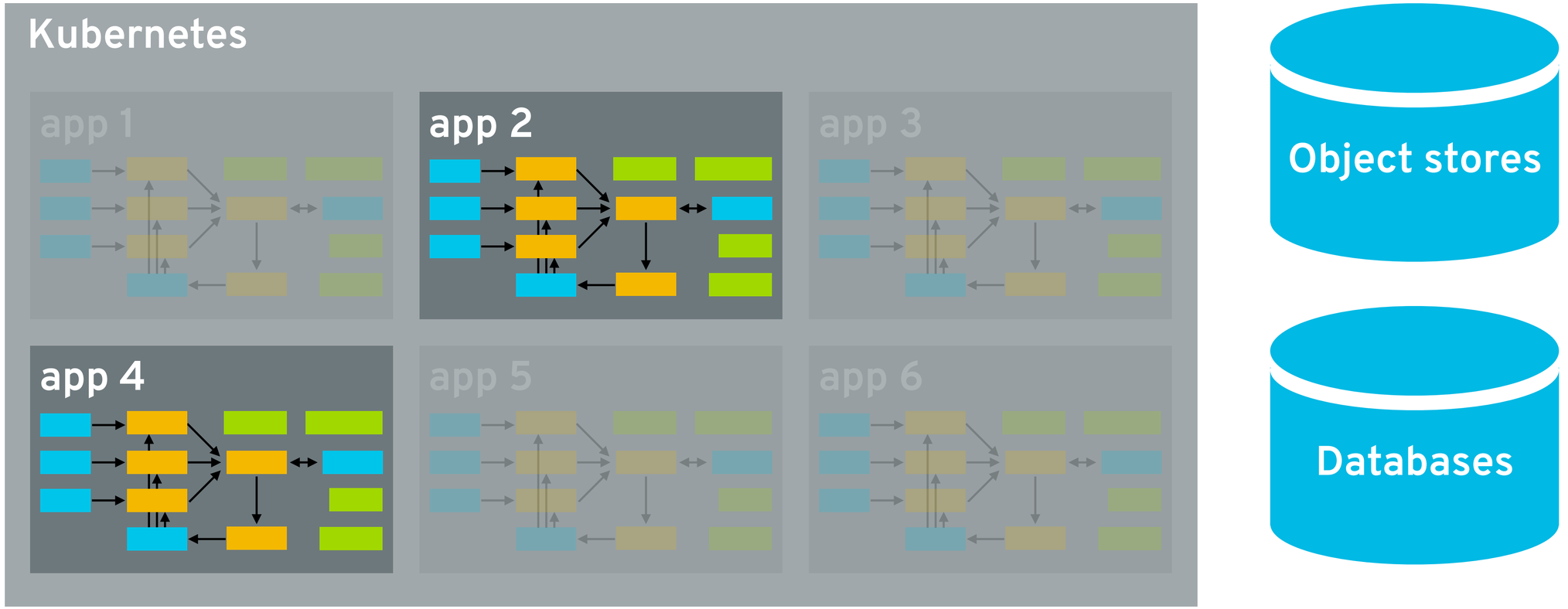



Kubernetes
app 1
app 2
app 3
app 4
app 5
app 6
Object stores
Databases



Object stores

Databases



http://radanalytics.io

Takeaways

Let Spark work for you.

Use efficient storage formats but don’t worry about data locality (yet).

ETL input data to Parquet early in your process.

Feature engineering effort often trumps fancy models.

Prefer interpretable models and easy-to-implement algorithms.

THANKS!

willb@redhat.com • @willb

https://chapeau.freevariable.com

http://radanalytics.io