Data science for the datacenter: processing logs with Apache Spark

William Benton
Red Hat Emerging Technology

BACKGROUND

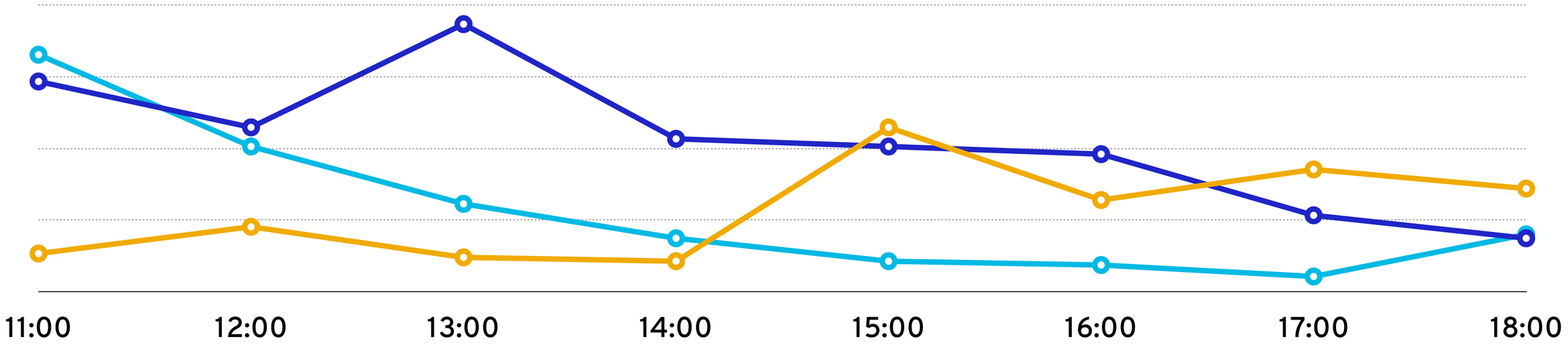
Challenges of log data

Challenges of log data

```
SELECT hostname, DATEPART(HH, timestamp) AS hour, COUNT(msg)
   FROM LOGS WHERE level='CRIT' AND msg LIKE '%failure%'
   GROUP BY hostname, hour
```

Challenges of log data

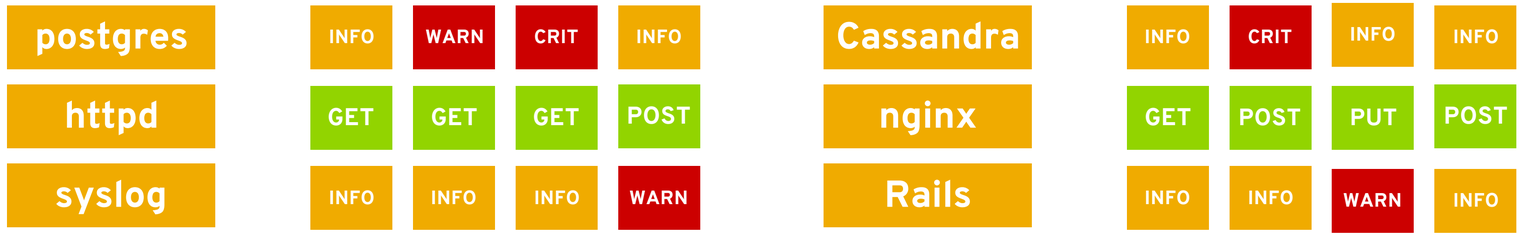
```
SELECT hostname, DATEPART(HH, timestamp) AS hour, COUNT(msg)
    FROM LOGS WHERE level='CRIT' AND msg LIKE '%failure%'
    GROUP BY hostname, hour
```



Challenges of log data

(ca. 2000)

Challenges of log data

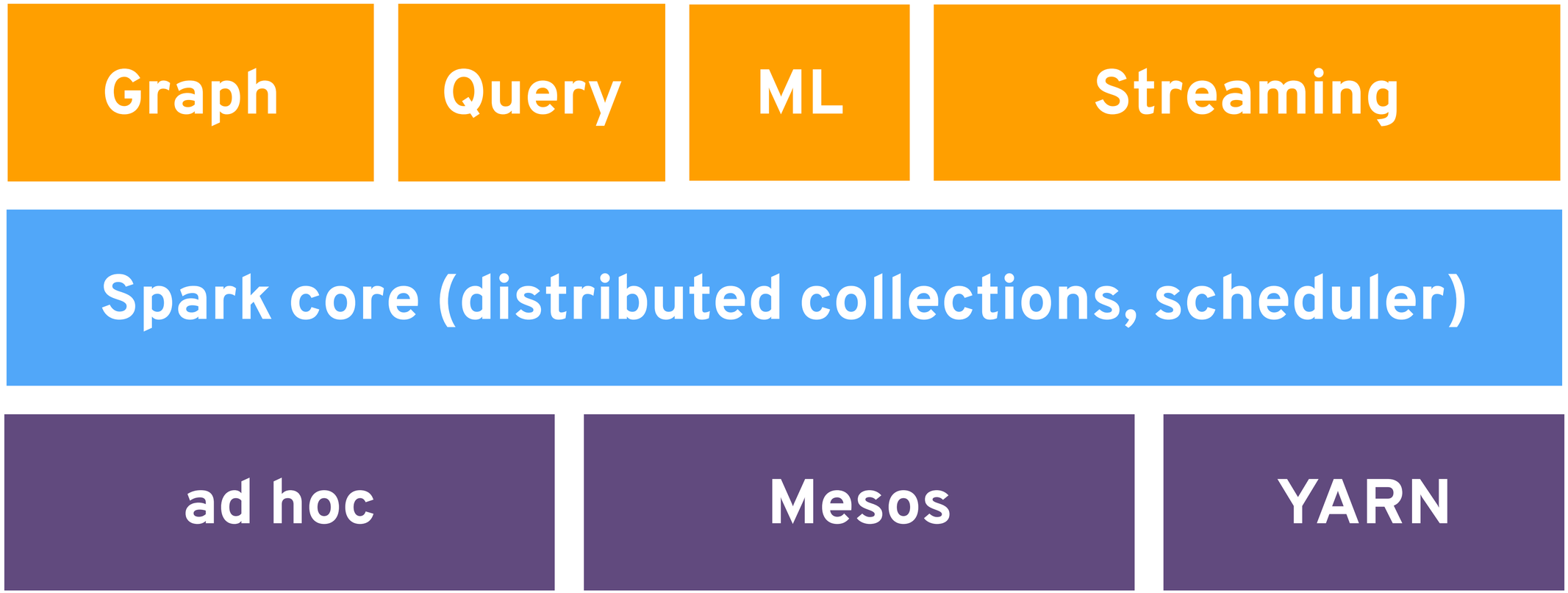


(ca. 2016)

Challenges of log data

How many **services** are generating logs in **your** datacenter **today**?

Apache Spark



DATA INGEST

Collecting log data

collecting

Ingesting live log data via rsyslog, logstash, fluentd

Collecting log data

collecting

Ingesting live log data via rsyslog, logstash, fluentd

normalizing

Reconciling log record metadata across sources

Collecting log data

collecting

Ingesting live log data via rsyslog, logstash, fluentd

normalizing

Reconciling log record metadata across sources

warehousing

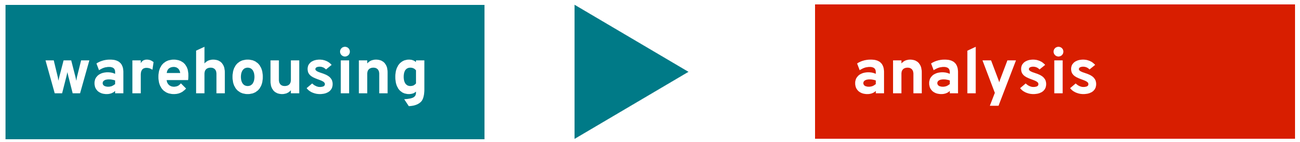
Storing normalized records in ES indices

Collecting log data

warehousing

Storing normalized
records in ES indices

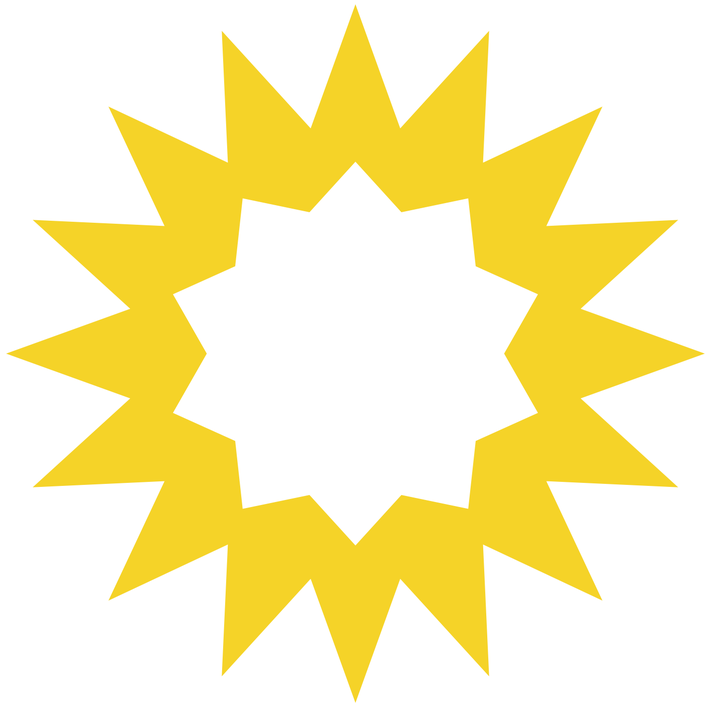
Collecting log data

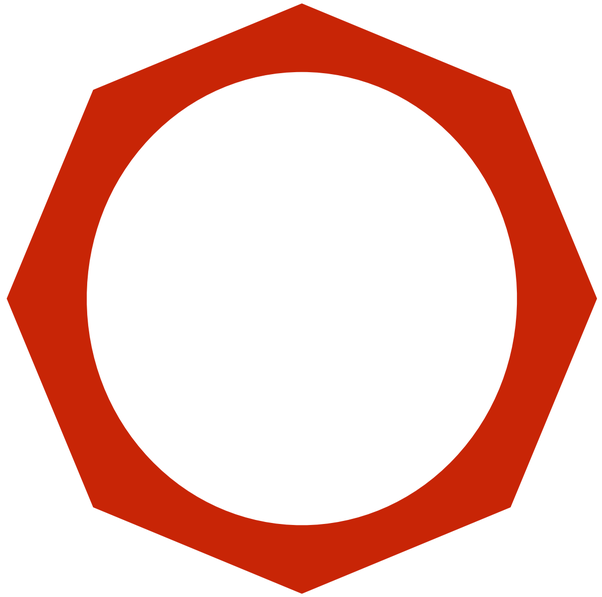


Storing normalized records in ES indices

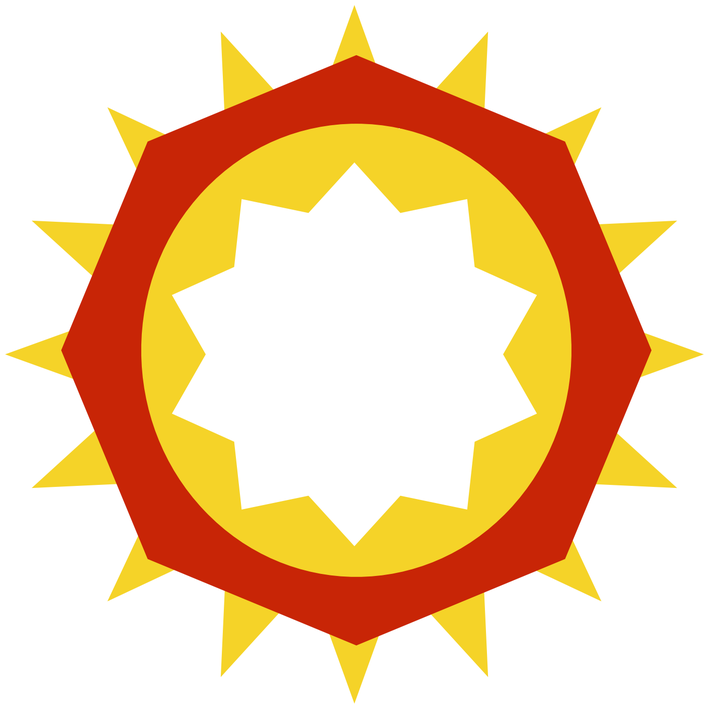
cache warehoused data as Parquet files on Gluster volume local to Spark cluster

Schema mediation

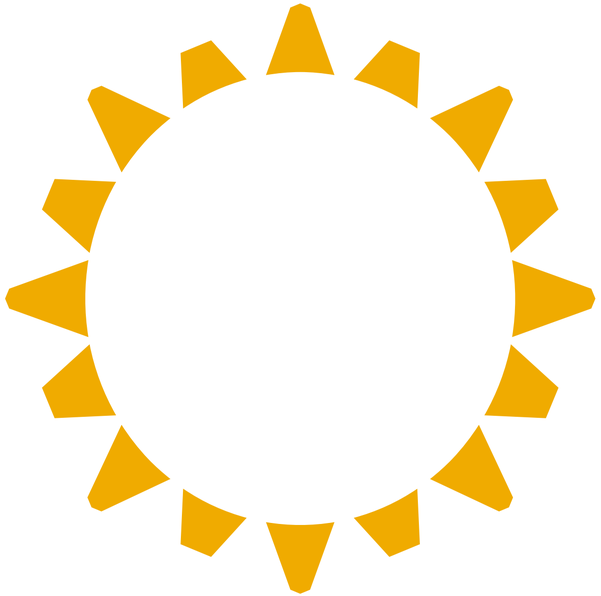




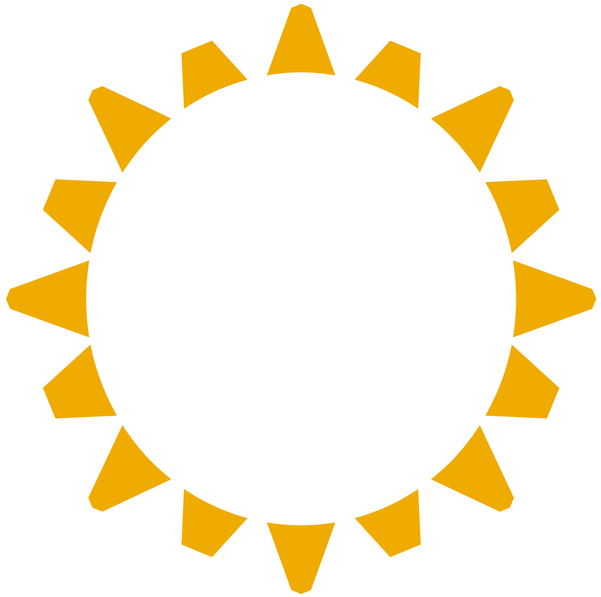
Schema mediation



Schema mediation



Schema mediation



timestamp, level, host, IP addresses, message, &c.

rsyslog-style metadata, like app name, facility, &c.

Structured queries at scale

```
logs
  .select("level").distinct
  .as[String].collect
```

Structured queries at scale

```
logs
  .select("level").distinct
  .as[String].collect
```

```
debug, notice, emerg,
err, warning, crit, info,
severe, alert
```

Structured queries at scale

```
logs
  .select("level").distinct
  .as[String].collect
```

```
debug, notice, emerg,
err, warning, crit, info,
severe, alert
```

```
logs
  .groupBy($"level", $"rsyslog.app_name")
  .agg(count("level").as("total"))
  .orderBy($"total".desc)
```

Structured queries at scale

```
logs
  .select("level").distinct
  .as[String].collect
```

```
debug, notice, emerg,
err, warning, crit, info,
severe, alert
```

```
logs
  .groupBy($"level", $"rsyslog.app_name")
  .agg(count("level").as("total"))
  .orderBy($"total".desc)
```

INFO	kubelet	17933574
INFO	kube-proxy	10961117
ERR	journal	6867921
INFO	systemd	5184475

...

FEATURE ENGINEERING

From log records to vectors

What does it mean for two log records to be similar?

From log records to vectors

What does it mean for two log records to be similar?

```
red
green
blue
orange
```

From log records to vectors

What does it mean for two log records to be similar?

```
red                 -> 000
green               -> 010
blue                -> 100
orange              -> 001
```

From log records to vectors

What does it mean for two log records to be similar?

```
red       -> 000
green     -> 010
blue      -> 100
orange    -> 001
```

```
pancakes
waffles
aebliskiver
omelets
bacon
hash browns
```

From log records to vectors

What does it mean for two log records to be similar?

```
red      -> 000
green    -> 010
blue     -> 100
orange   -> 001
```

```
pancakes      -> 10000
waffles       -> 01000
aebliskiver   -> 00100
omelets       -> 00001
bacon         -> 00000
hash browns   -> 00010
```

From log records to vectors

What does it mean for two log records to be similar?

```
red         -> 000
green       -> 010
blue        -> 100
orange      -> 001
```

```
pancakes    -> 10000
waffles     -> 01000
aebliskiver -> 00100
omelets     -> 00001
bacon       -> 00000
hash browns -> 00010
```

```
red pancakes
orange waffles
```

From log records to vectors

What does it mean for two log records to be similar?

```
red          -> 000
green        -> 010
blue         -> 100
orange       -> 001
```

```
pancakes     -> 10000
waffles      -> 01000
aebliskiver  -> 00100
omelets      -> 00001
bacon        -> 00000
hash browns  -> 00010
```

```
red pancakes      -> 00010000
orange waffles    -> 00101000
```

From log records to vectors

What does it mean for two log records to be similar?

From log records to vectors

What does it mean for two log records to be similar?

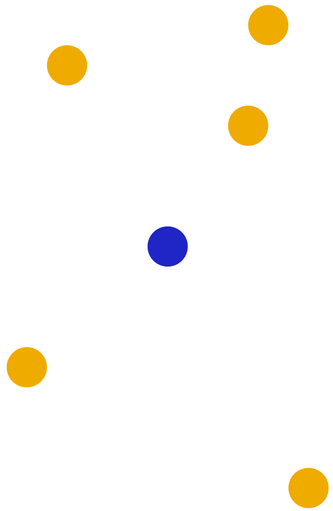
```
{level: INFO, hostname: fred, timestamp: "2016-05-06 00:00:01", ...}
{level: DEBUG, hostname: barney, timestamp: "2016-05-06 00:00:02", ...}
{level: INFO, hostname: fred, timestamp: "2016-05-06 00:00:03", ...}
{level: ERR, hostname: barney, timestamp: "2016-05-06 00:00:03", ...}
{level: ALERT, hostname: wilma, timestamp: "2016-05-06 00:00:03", ...}
```

From log records to vectors

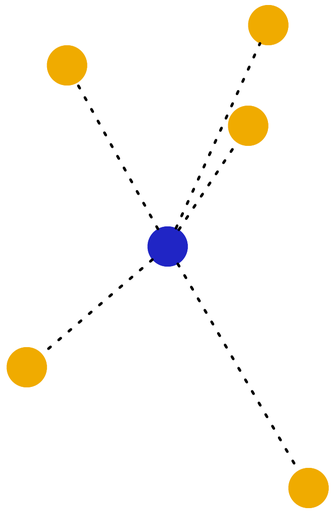
What does it mean for two log records to be similar?

```
{level: INFO, hostname: fred, timestamp: "2016-05-06 00:00:01", ...}        -> 00000100
{level: DEBUG, hostname: barney, timestamp: "2016-05-06 00:00:02", ...}     -> 00100000
{level: INFO, hostname: fred, timestamp: "2016-05-06 00:00:03", ...}        -> 00000100
{level: ERR, hostname: barney, timestamp: "2016-05-06 00:00:03", ...}       -> 00001000
{level: ALERT, hostname: wilma, timestamp: "2016-05-06 00:00:03", ...}      -> 10000000
```

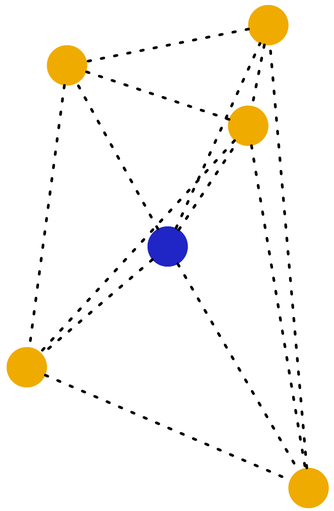
Similarity and distance



Similarity and distance

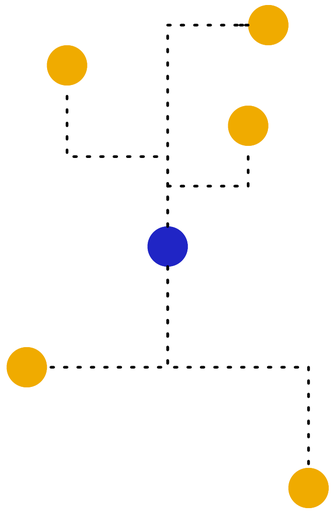


Similarity and distance



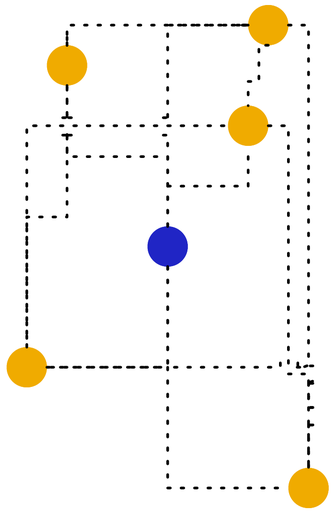
$$\sqrt{(q - p) \cdot (q - p)}$$

Similarity and distance



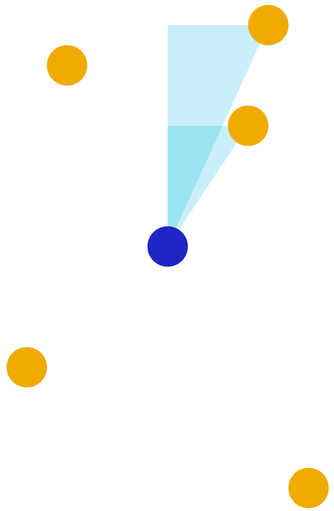
$$\sum_{i=1}^{n} |p_i - q_i|$$

Similarity and distance



$$\sum_{i=1}^{n} |p_i - q_i|$$

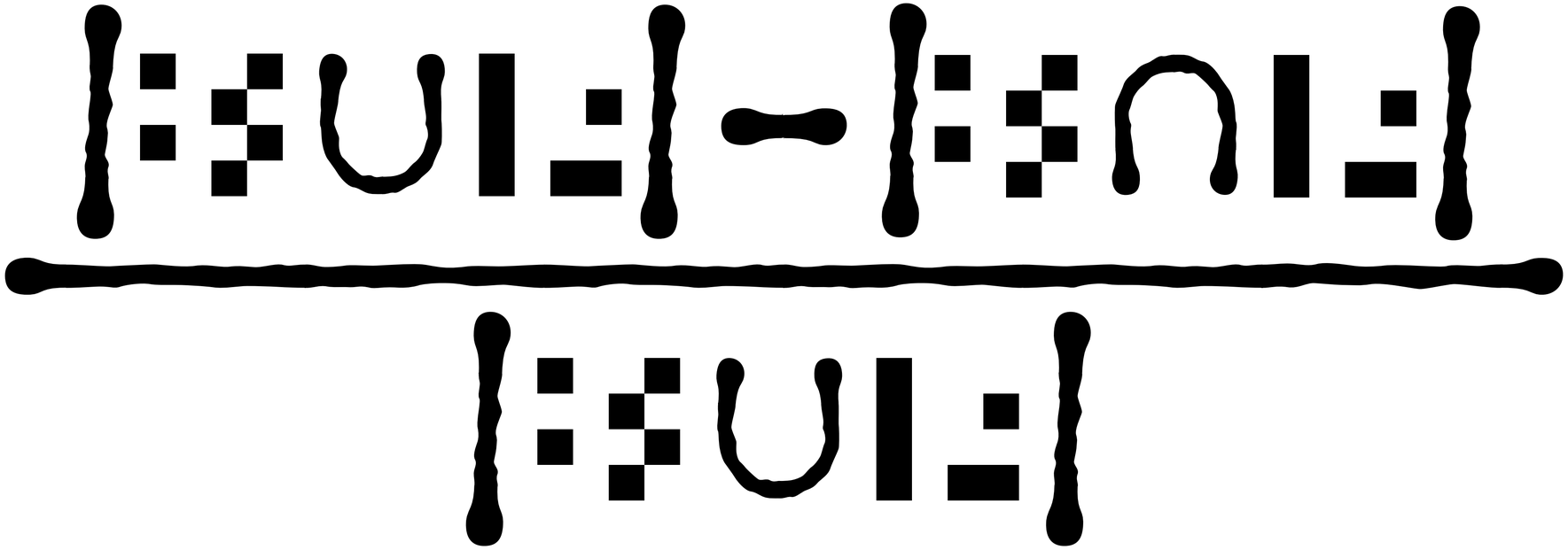
Similarity and distance



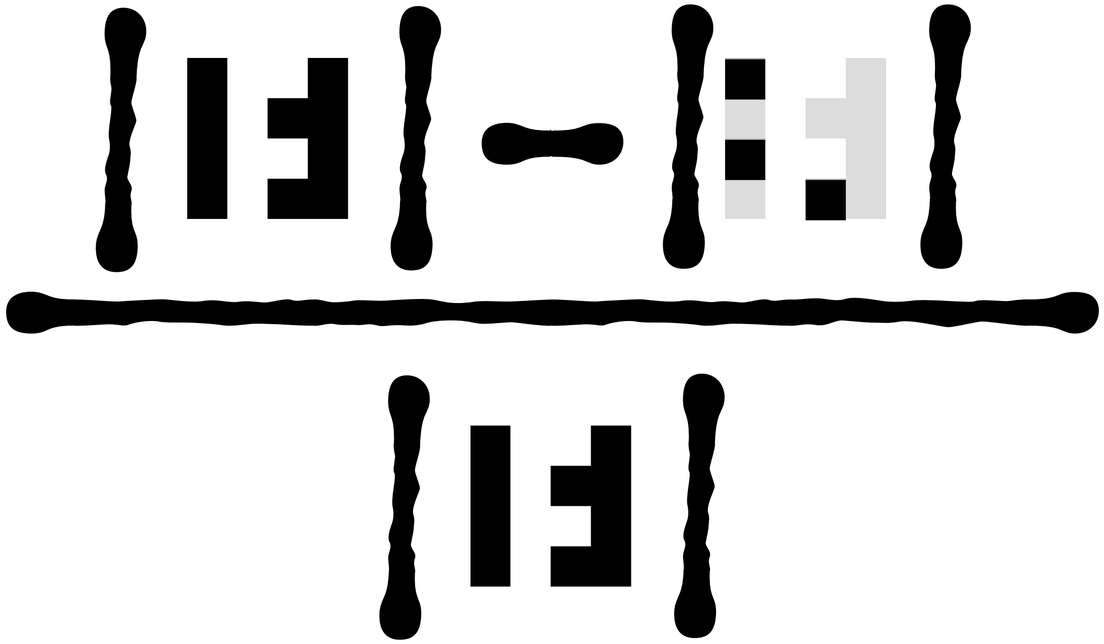
$$\frac{p \cdot q}{\|p\| \|q\|}$$

Similarity and distance

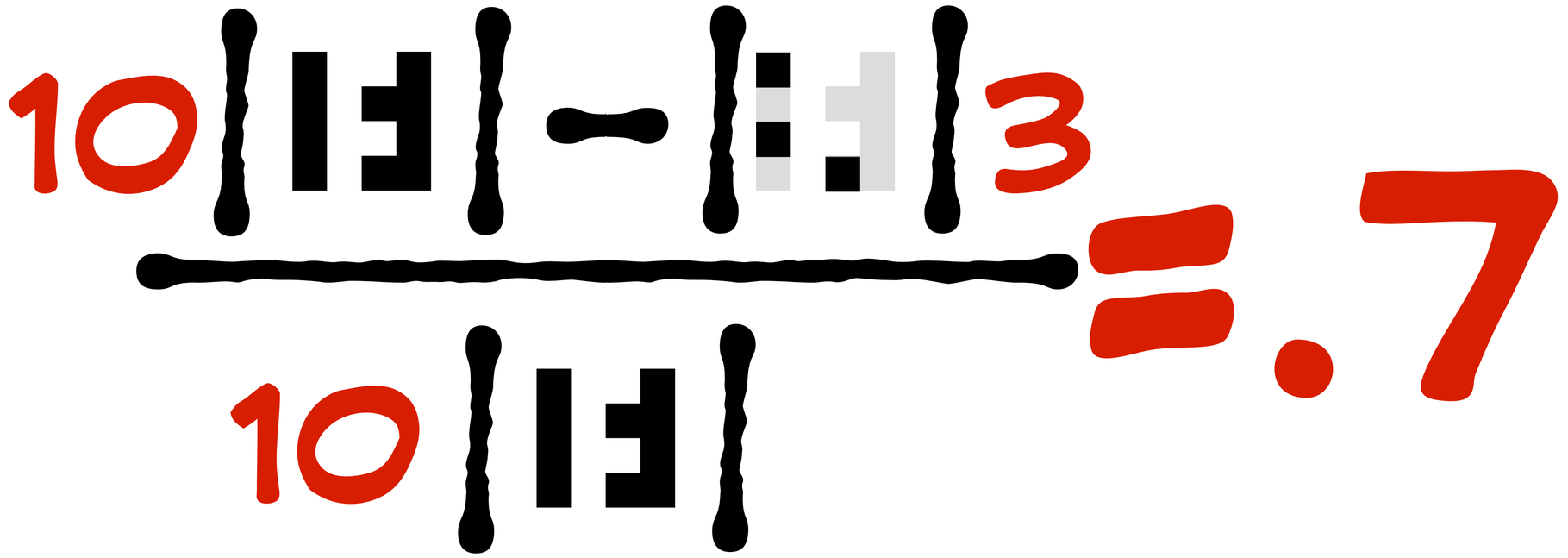
Similarity and distance



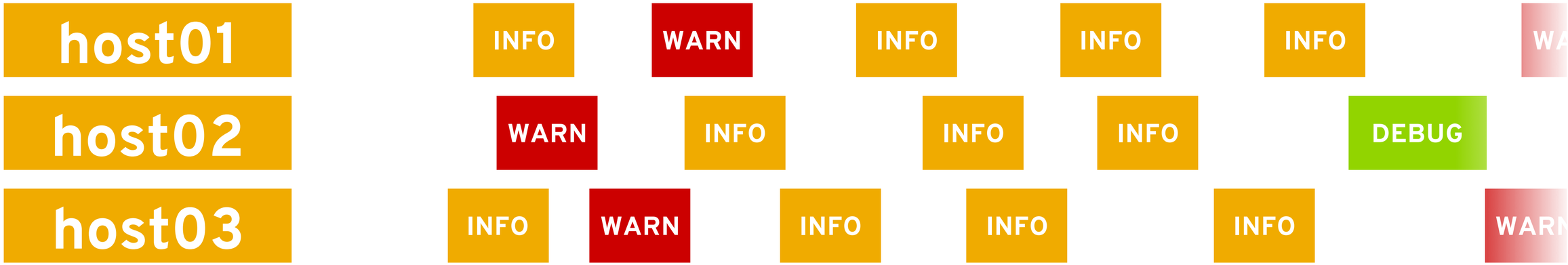
Similarity and distance



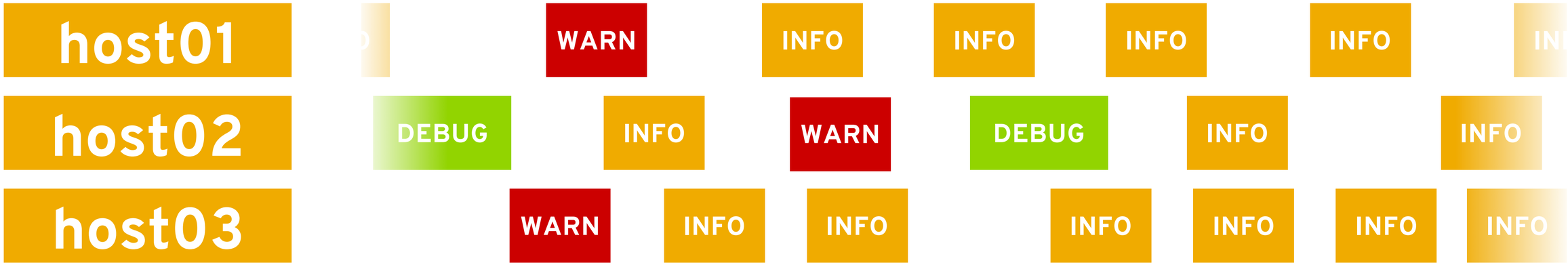
Similarity and distance



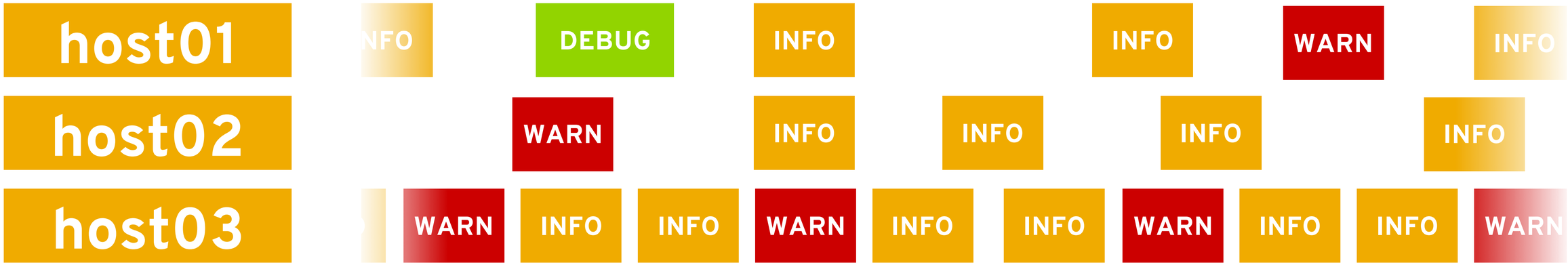
Other interesting features



Other interesting features



Other interesting features



Other interesting features

```
INFO:  Everything is great!  Just checking in to let you know I’m OK.
```

Other interesting features

```
INFO:  Everything is great!  Just checking in to let you know I’m OK.
CRIT:  No requests in last hour; suspending running app containers.
```

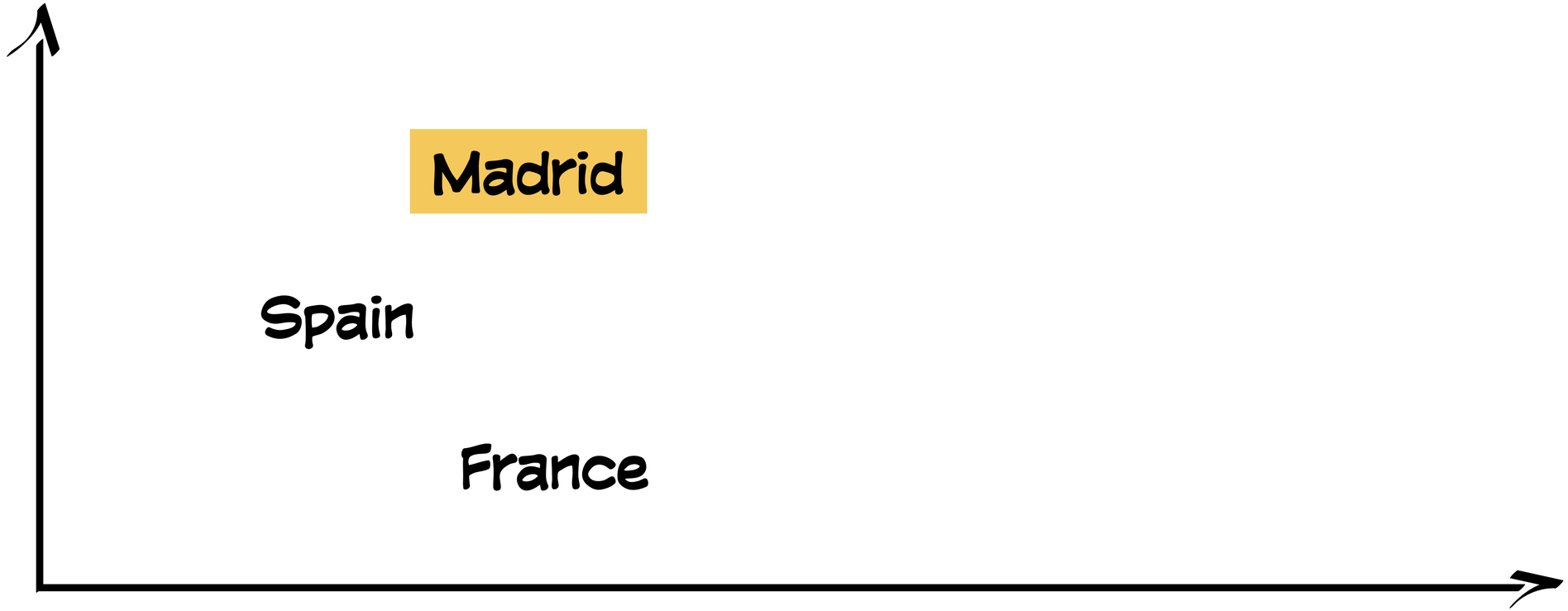
Other interesting features

```
INFO:  Everything is great!  Just checking in to let you know I’m OK.
CRIT:  No requests in last hour; suspending running app containers.
INFO:  Phoenix datacenter is on fire; may not rise from ashes.
```

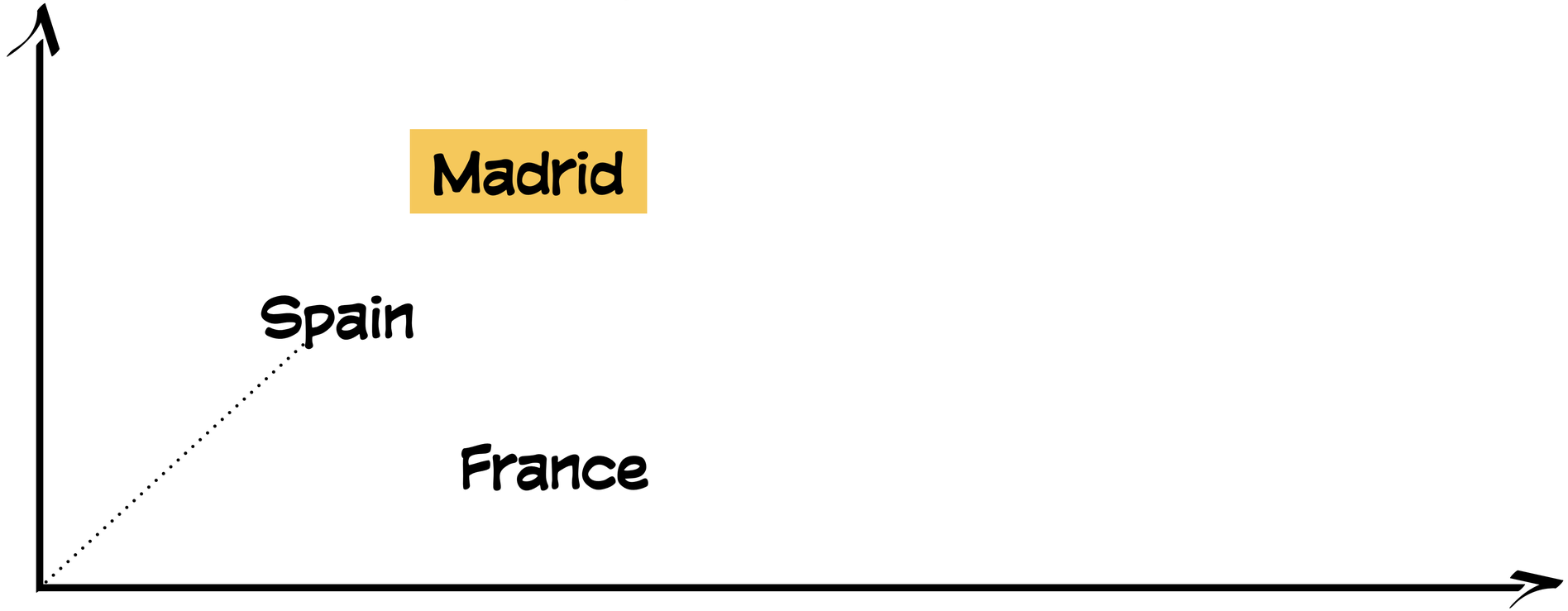
NATURAL LANGUAGE and LOG DATA

Introducing word2vec

Introducing word2vec



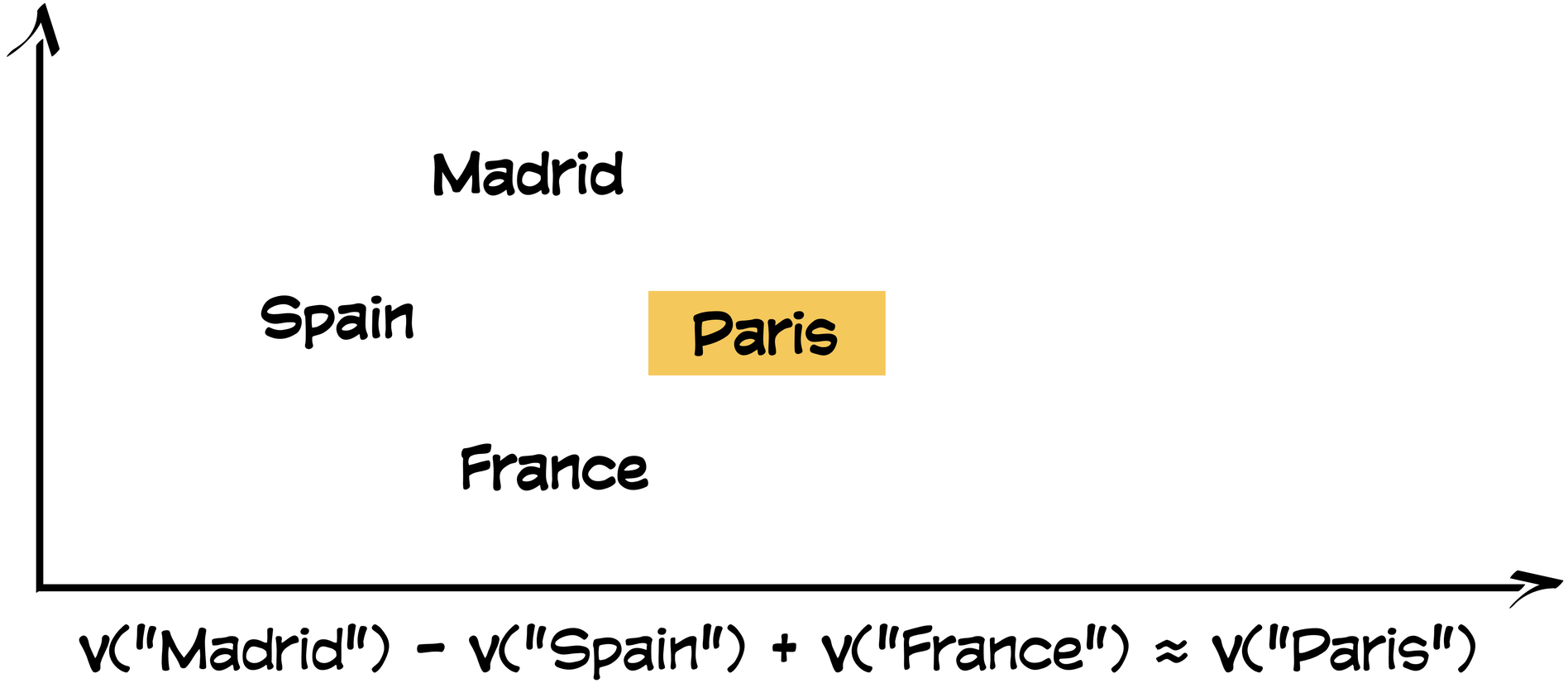
Introducing word2vec



Introducing word2vec

Introducing word2vec

Introducing word2vec



Preprocessing text

What is a word? How are “log words” different from words found in conventional prose?

Preprocessing text

What is a word? How are “log words” different from words found in conventional prose?

Apache® Oozie™

Java HotSpot™

Preprocessing text

What is a word? How are “log words” different from words found in conventional prose?

Apache® Oozie™

Java HotSpot™

systemd

ZooKeeper

etcd

Preprocessing text

What is a word? How are “log words” different from words found in conventional prose?

Apache® Oozie™

Java HotSpot™

systemd

/dev/null

ZooKeeper

etcd

Preprocessing text

What is a word? How are “log words” different from words found in conventional prose?

Apache® Oozie™

Java HotSpot™

systemd

OutOfMemoryError

/dev/null

ZooKeeper

Kernel::exec

etcd

Preprocessing text

What is a word? How are “log words” different from words found in conventional prose?

Apache® Oozie™

Java HotSpot™

ENOENT

systemd

OPEN_MAX

OutOfMemoryError

/dev/null

ZooKeeper

Kernel::exec

etcd

Preprocessing text

What is a word? How are “log words” different from words found in conventional prose?

Apache® Oozie™

Java HotSpot™

ENOENT

systemd

OPEN_MAX

192.168.0.1

OutOfMemoryError

/dev/null

ZooKeeper

Kernel::exec

etcd

Preprocessing text

What is a word? How are “log words” different from words found in conventional prose?

Apache® Oozie™

Java HotSpot™

ENOENT

systemd

OPEN_MAX

192.168.0.1

QEMU

OutOfMemoryError

/dev/null

ZooKeeper

Kernel::exec

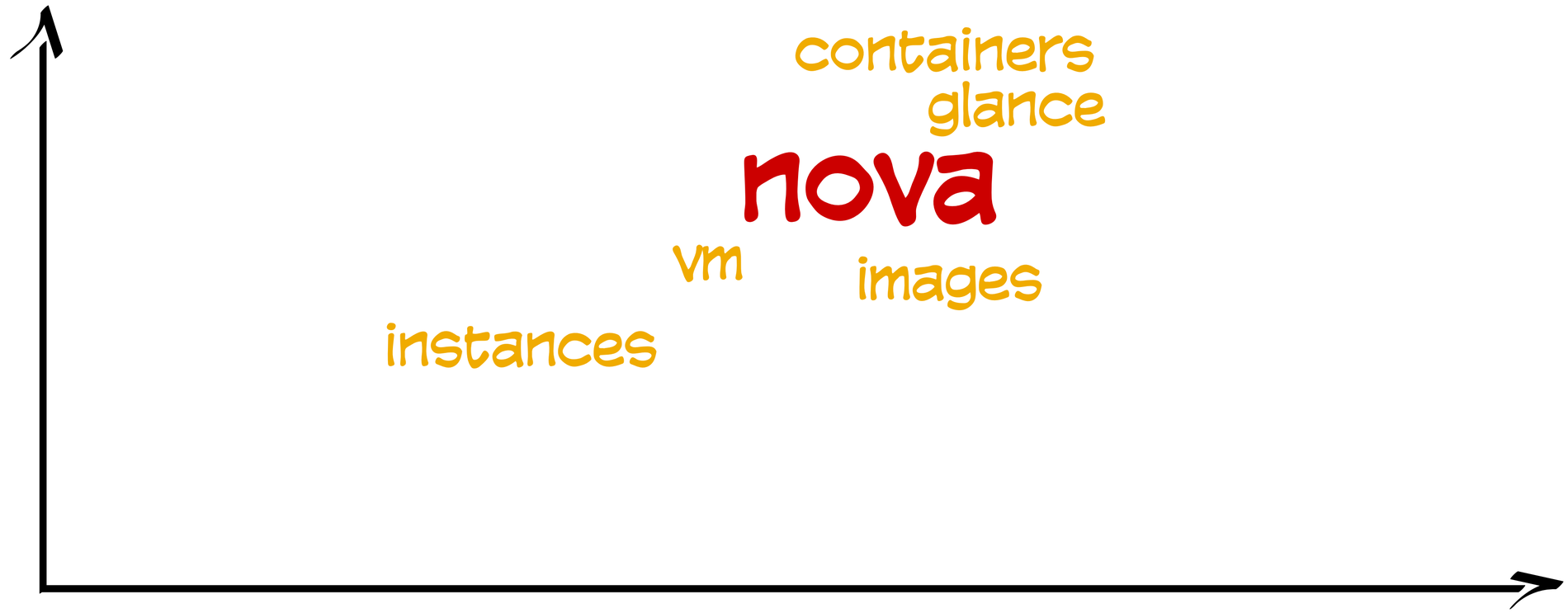
etcd

```
// assume messages is an RDD of log message texts

val oneletter = new scala.util.matching.Regex(".*([A-Za-z]).*")

def tokens(s: String, post: String=>String): Seq[String] =
  collapseWhitespace(s)
    .split(" ")
    .map(s => post(stripPunctuation(s)))
    .collect { case token @ oneletter(_) => token }

val tokenSeqs = messages.map(line => tokens(line), identity[String])

val w2v = new org.apache.spark.mllib.feature.Word2Vec

val model = w2v.fit(tokenSeqs)
```

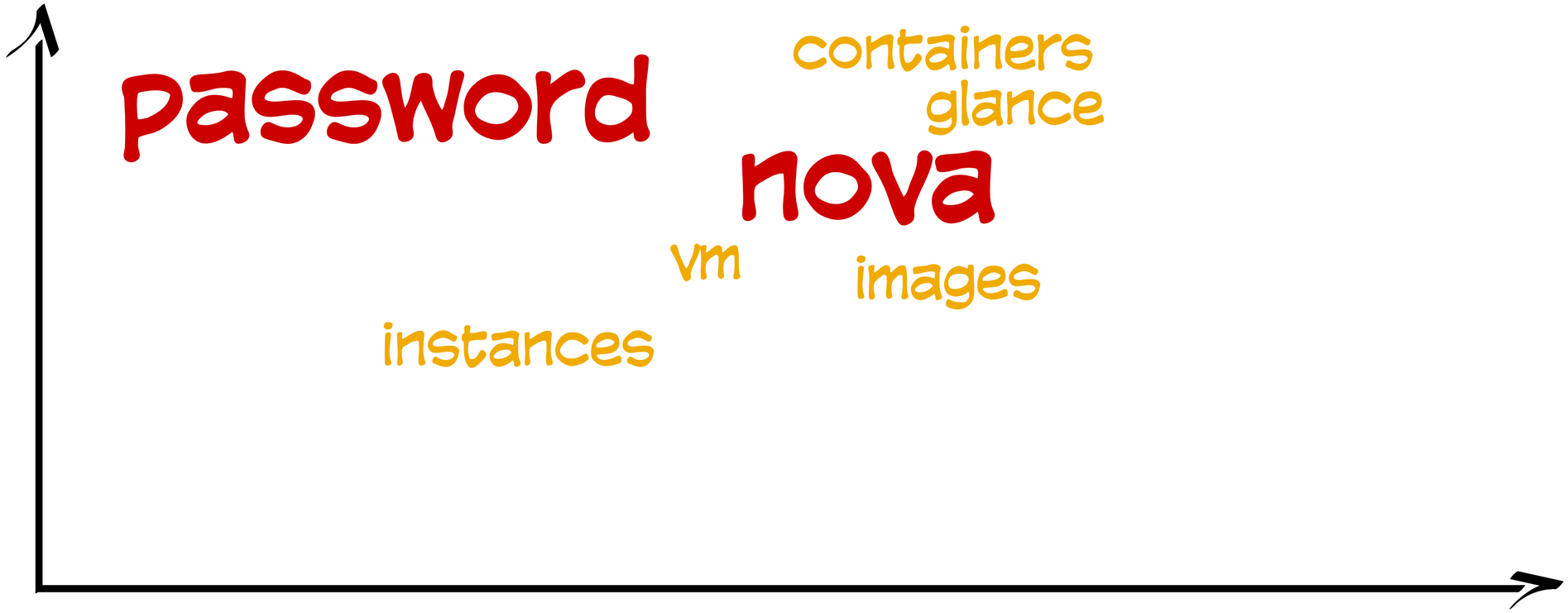
Insights from log messages

Insights from log messages

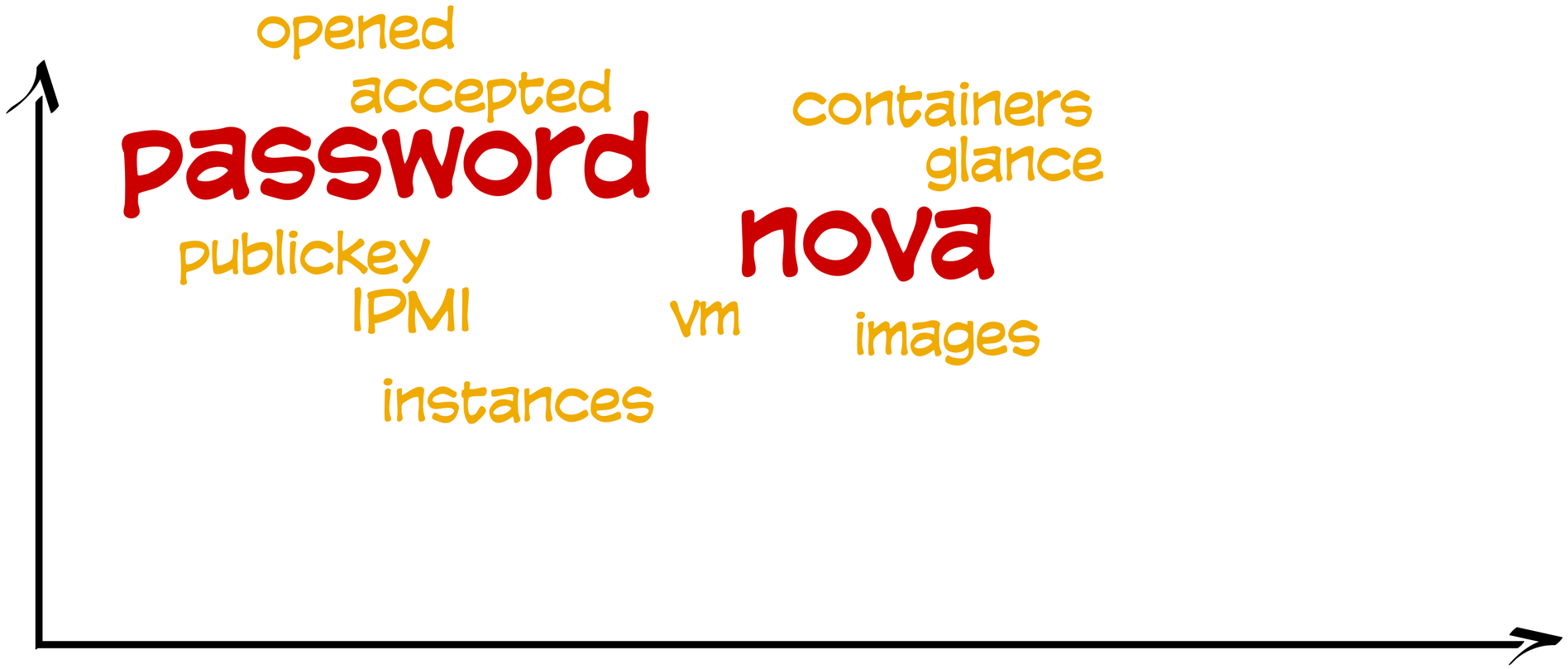
Insights from log messages



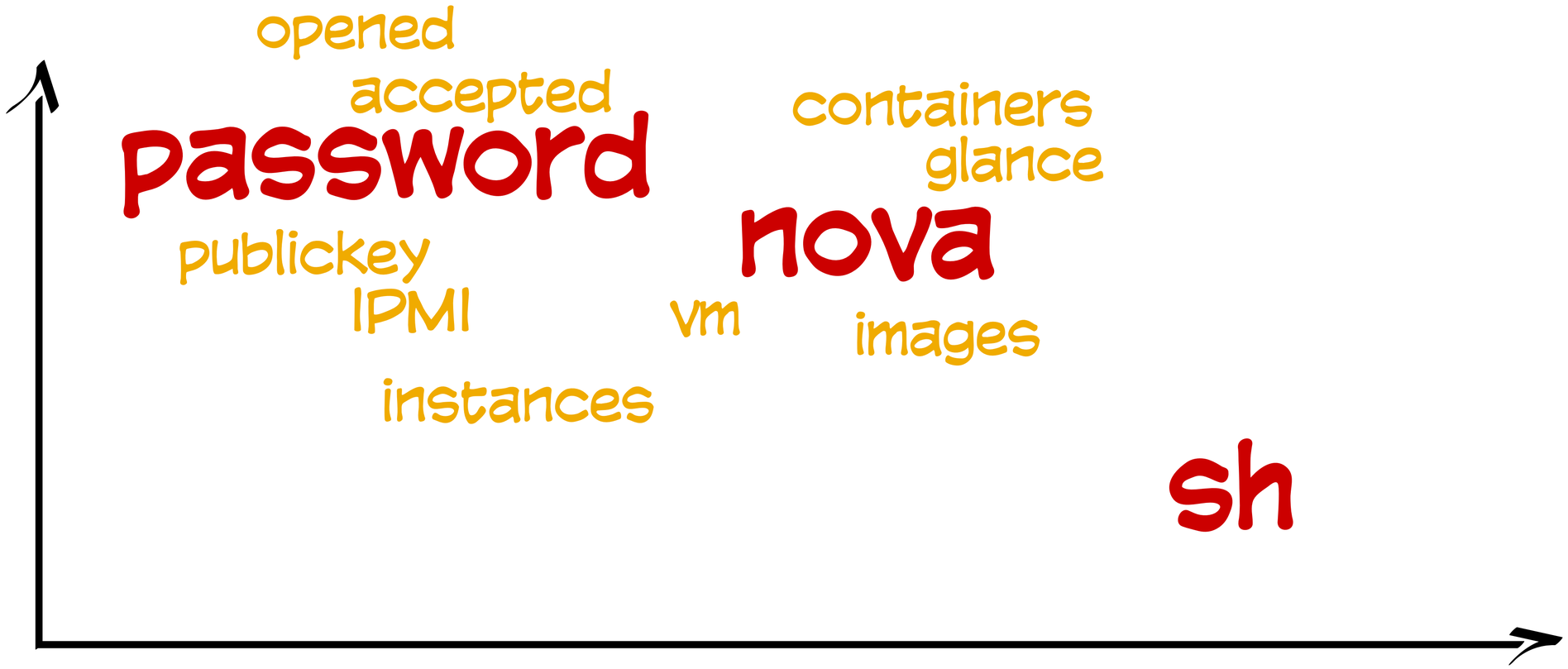
Insights from log messages



Insights from log messages



Insights from log messages

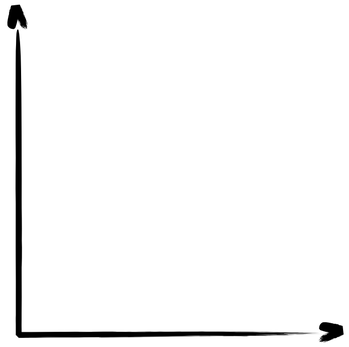


Insights from log messages

VISUALIZING STRUCTURE and FINDING OUTLIERS

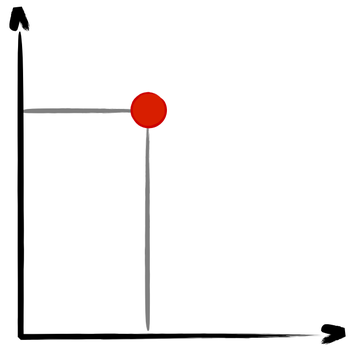
Multidimensional data

Multidimensional data



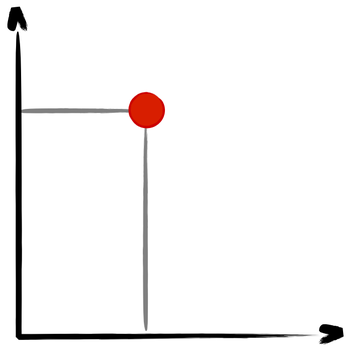
[4,7]

Multidimensional data

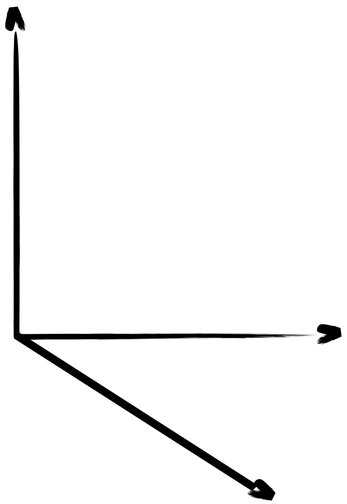


[4,7]

Multidimensional data

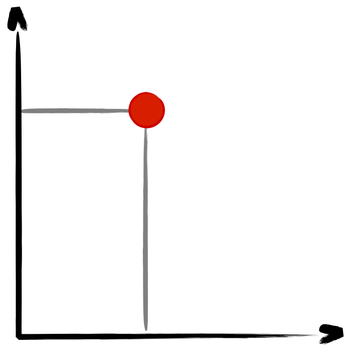


[4,7]

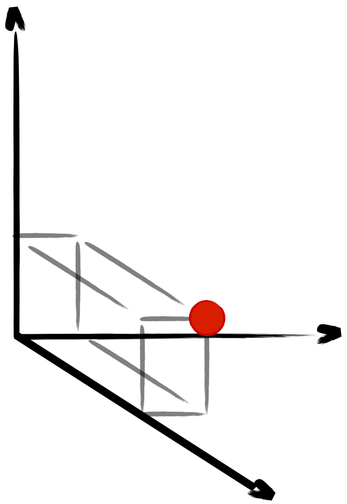


[2,3,5]

Multidimensional data

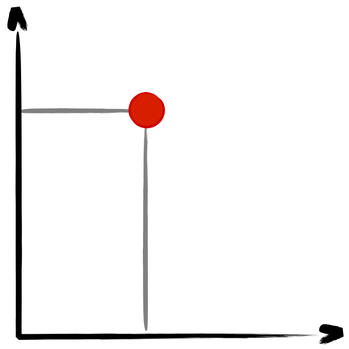


[4,7]

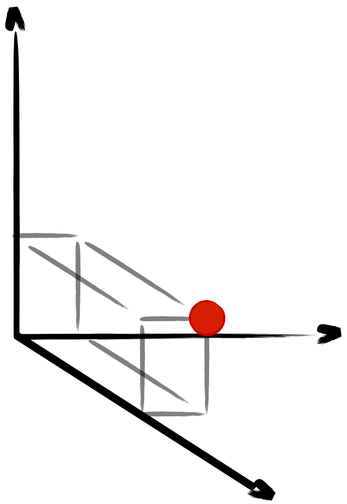


[2,3,5]

Multidimensional data



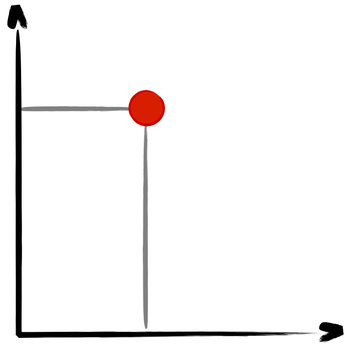
[4,7]



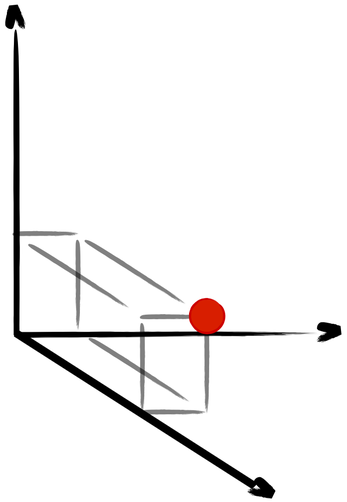
[2,3,5]

[7,1,6,5,12,
8,9,2,2,4,
7,11,6,1,5]

Multidimensional data



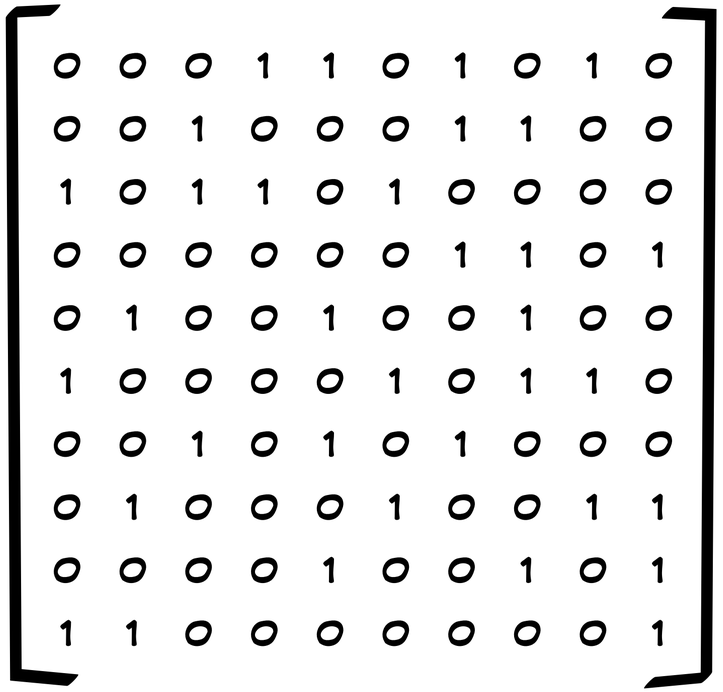
[4,7]



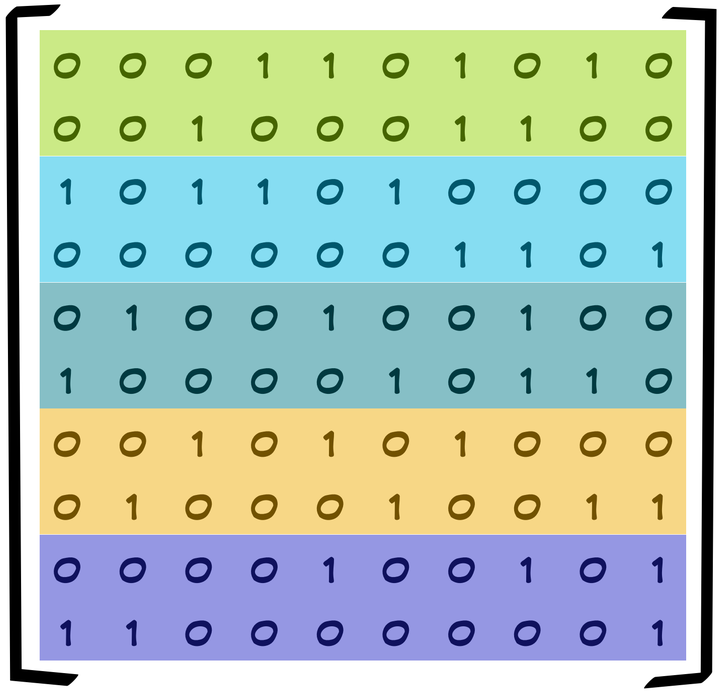
[2,3,5]

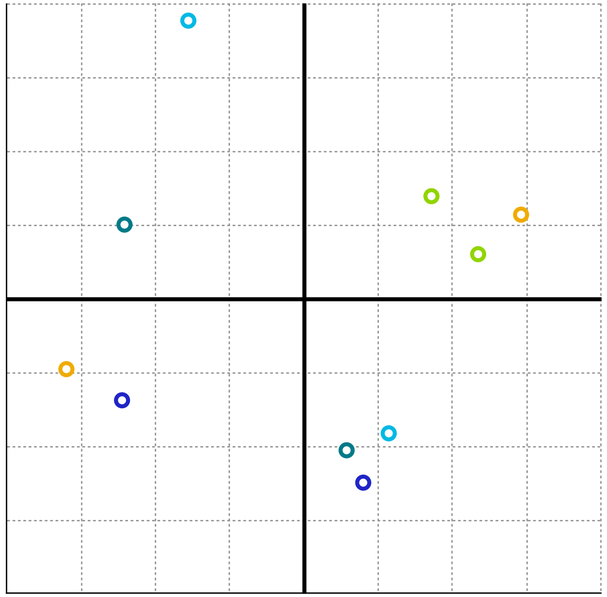
[7,1,6,5,12,
8,9,2,2,4,
7,11,6,1,5]

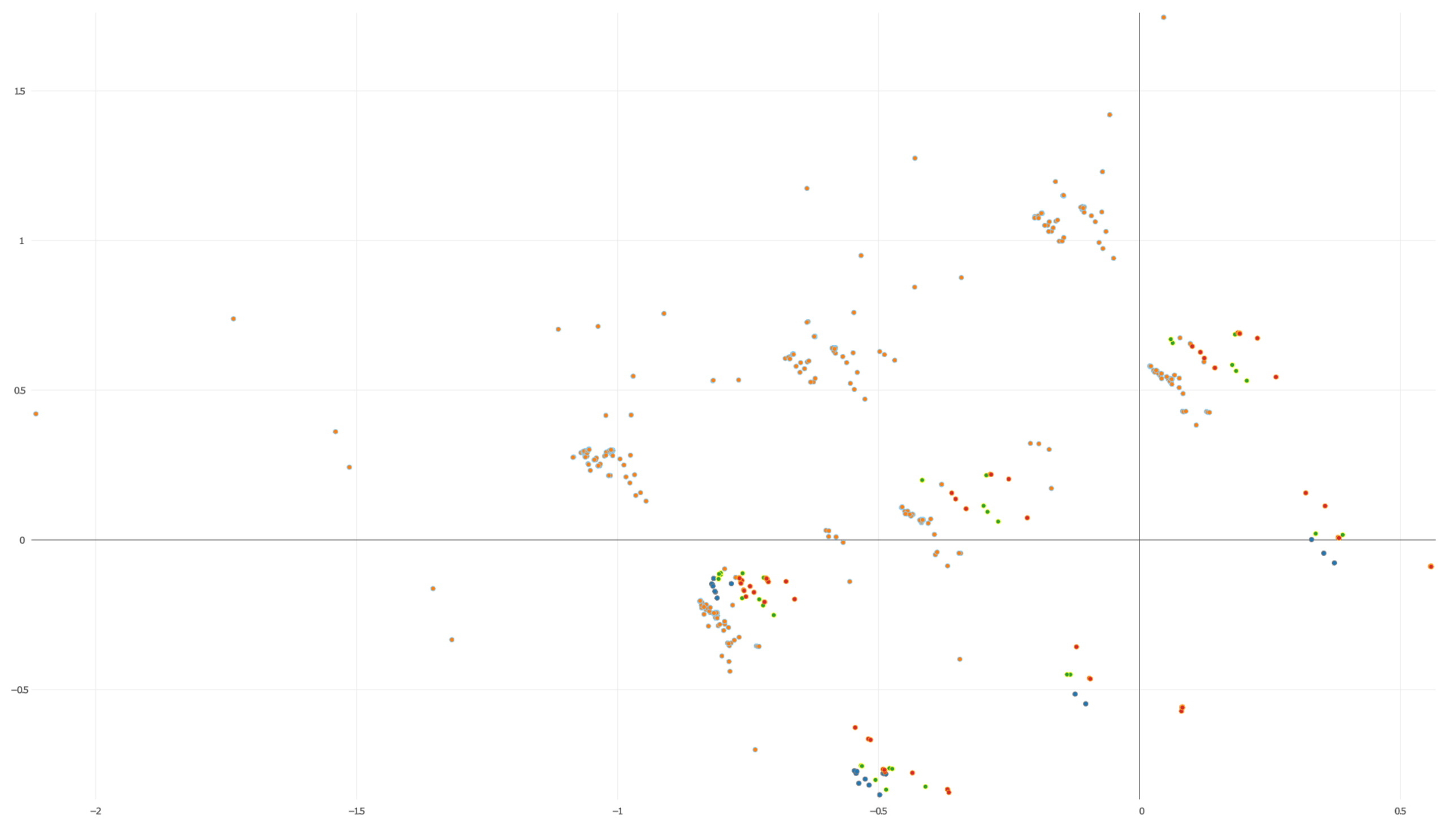
A linear approach: PCA



A linear approach: PCA

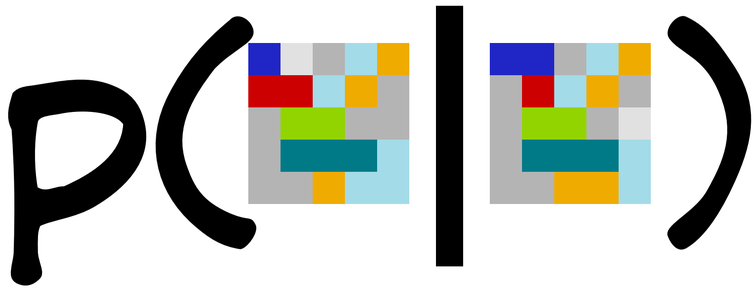




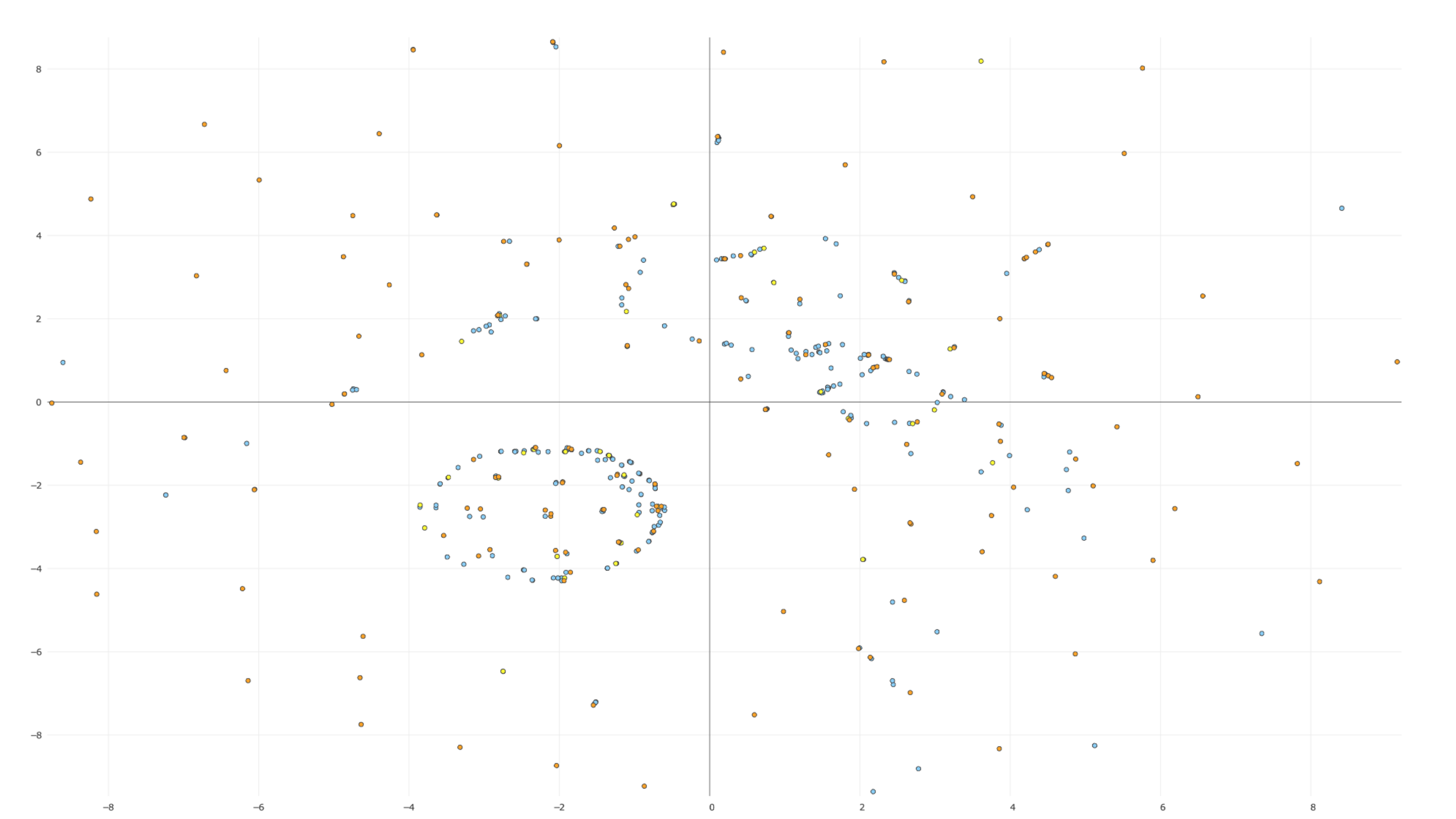

1.5
1
0.5
0
−0.5
−2
−1.5
−1
−0.5
0
0.5

A nonlinear approach: t-SNE

A nonlinear approach: t-SNE

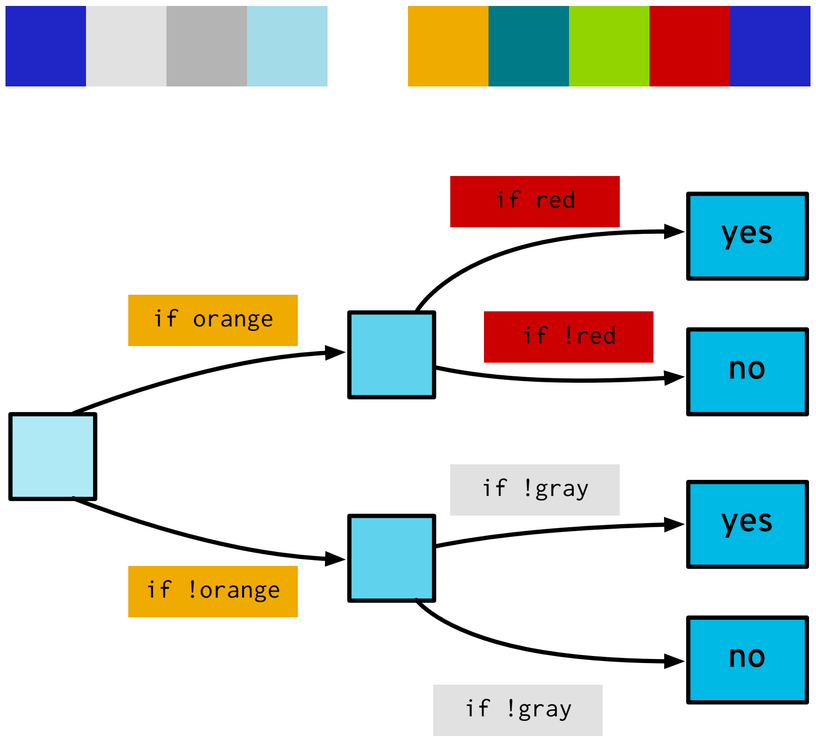


A nonlinear approach: t-SNE

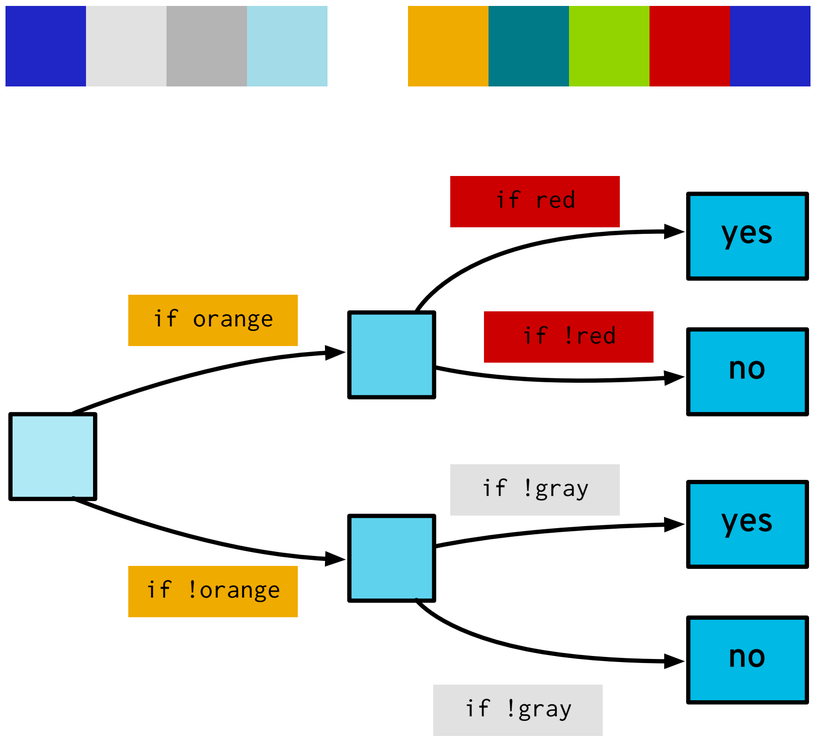

8
6
4
2
0
−2
−4
−6
−8
−8
−6
−4
−2
0
2
4
6
8

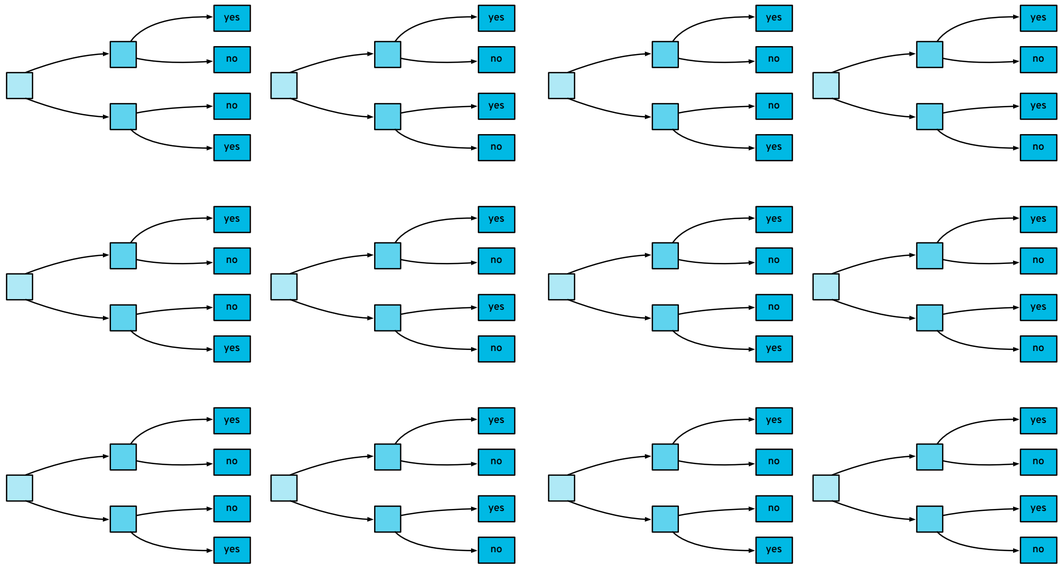
Tree-based approaches

Tree-based approaches

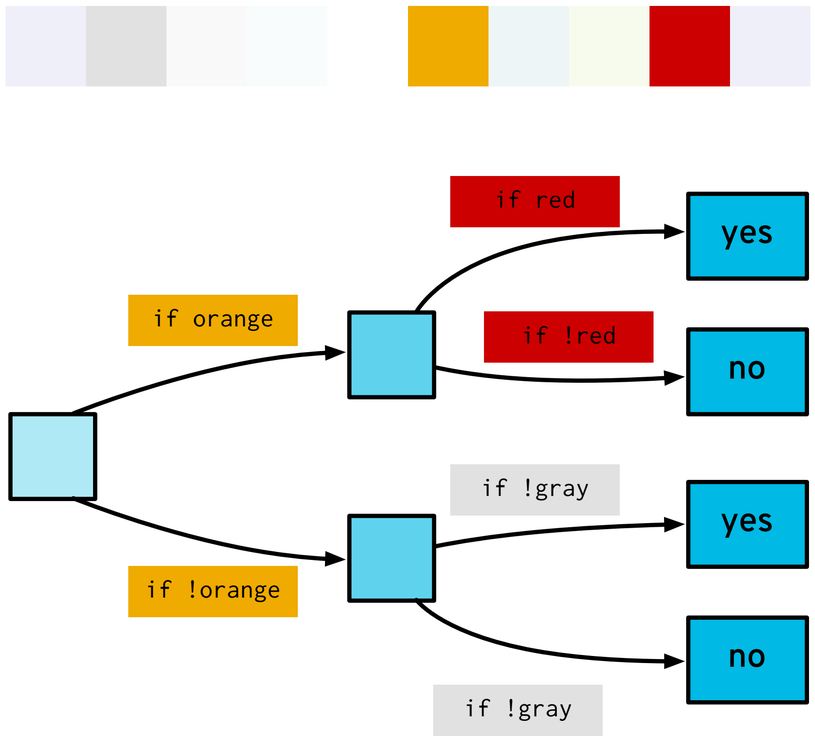


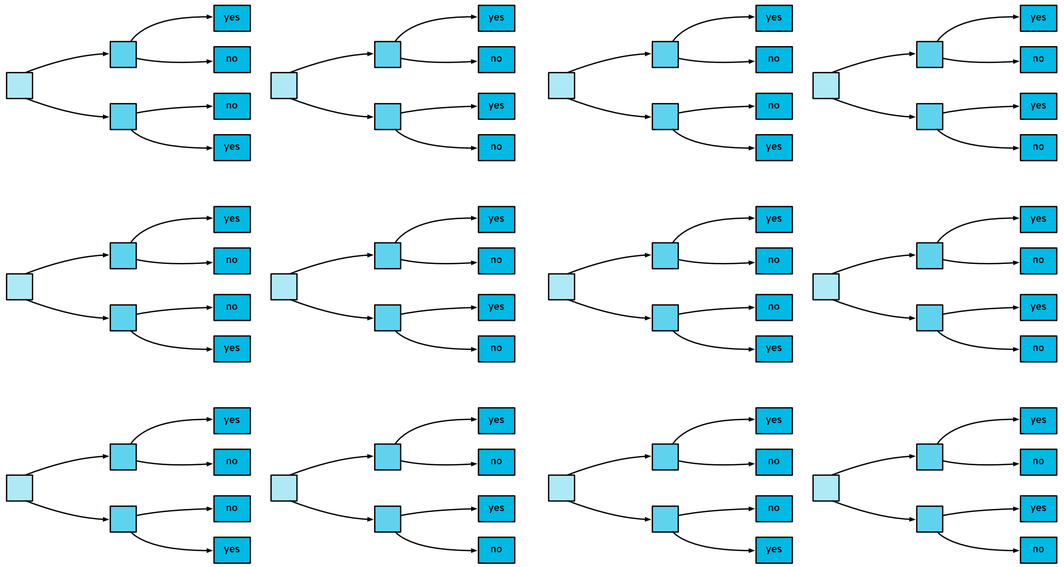
Tree-based approaches



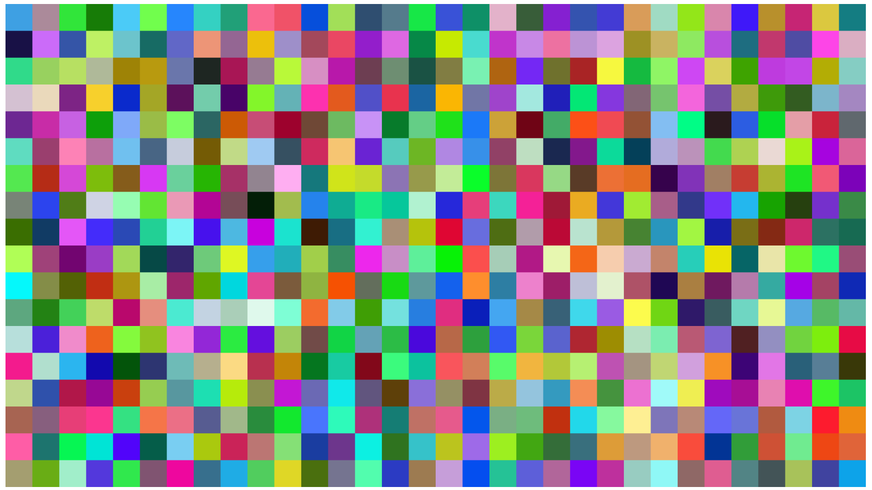


Tree-based approaches

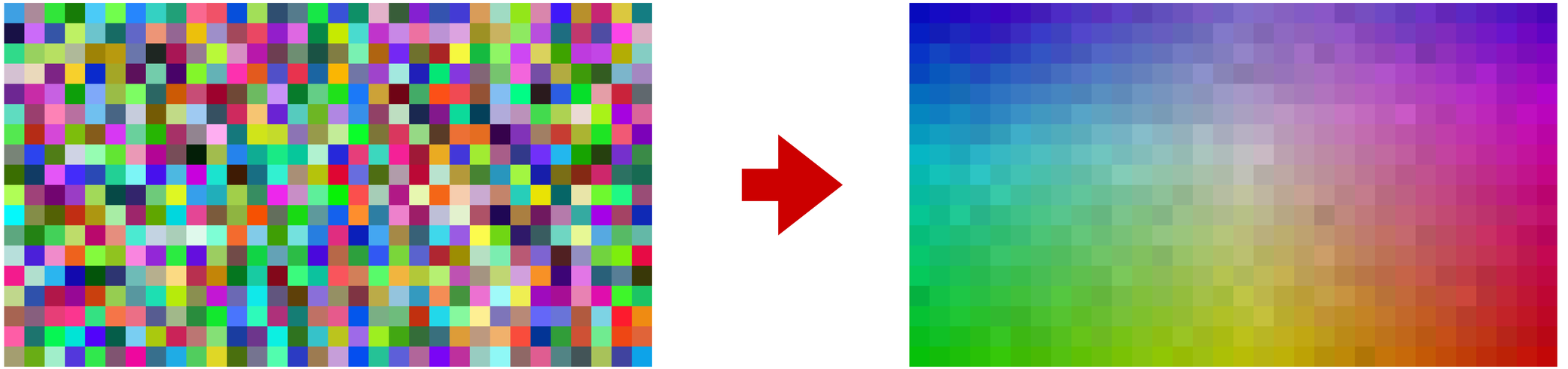




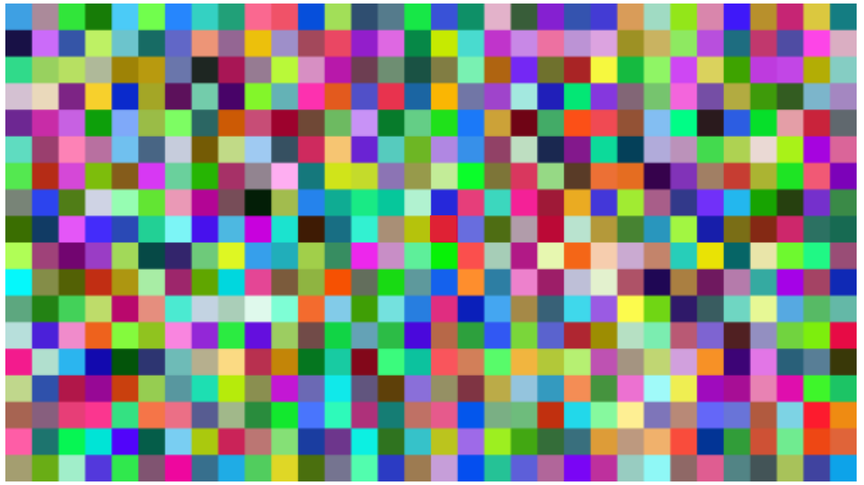
Self-organizing maps



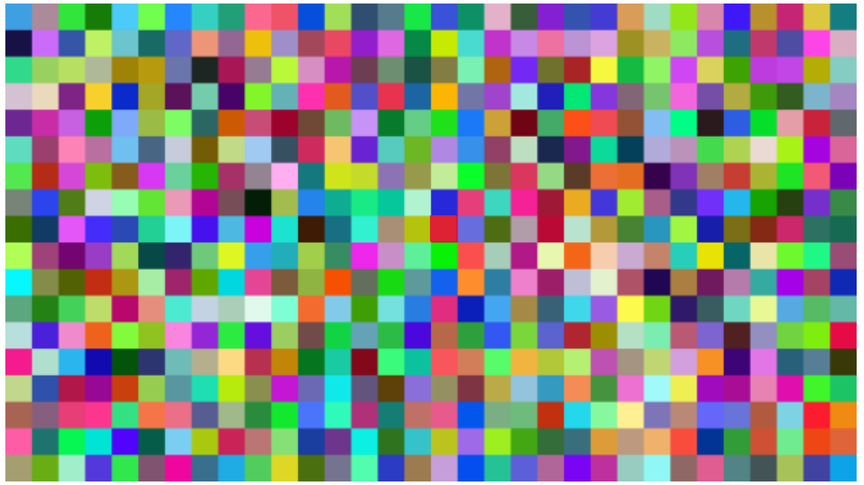
Self-organizing maps



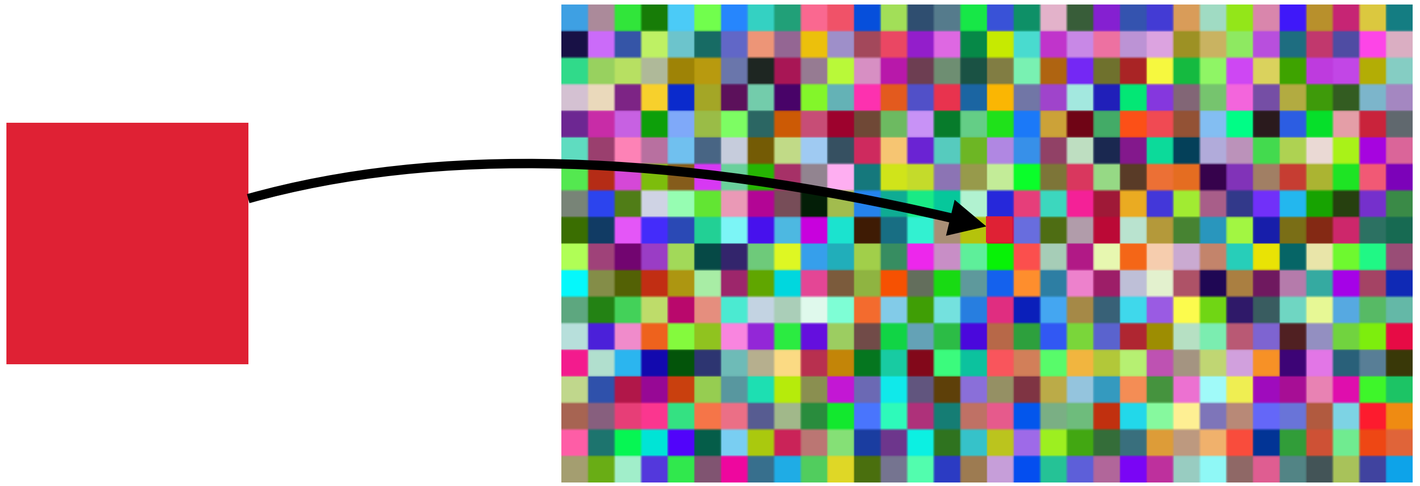
Training SOMs



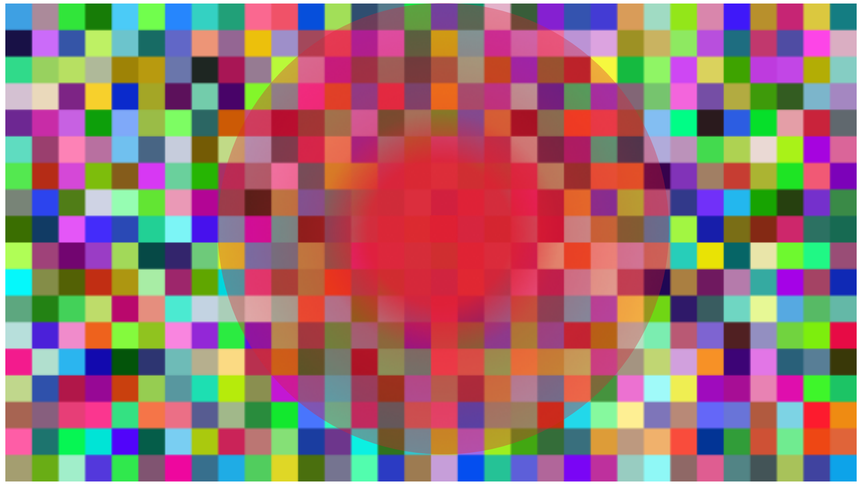
Training SOMs



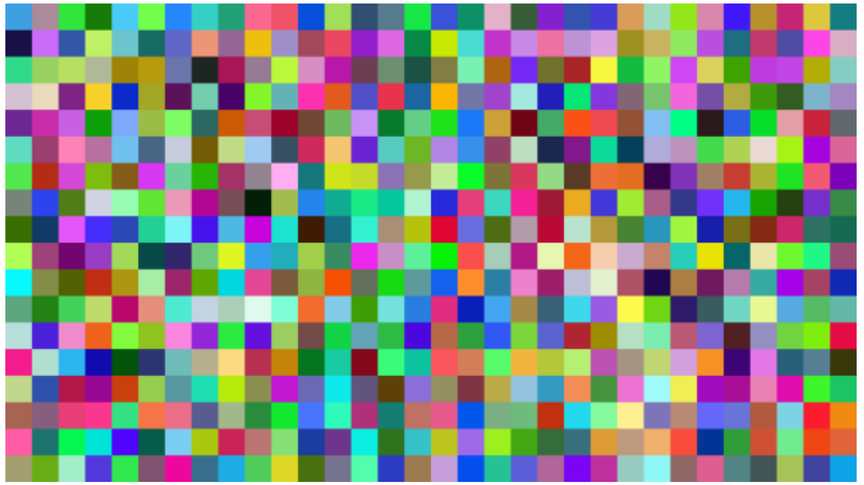
Training SOMs

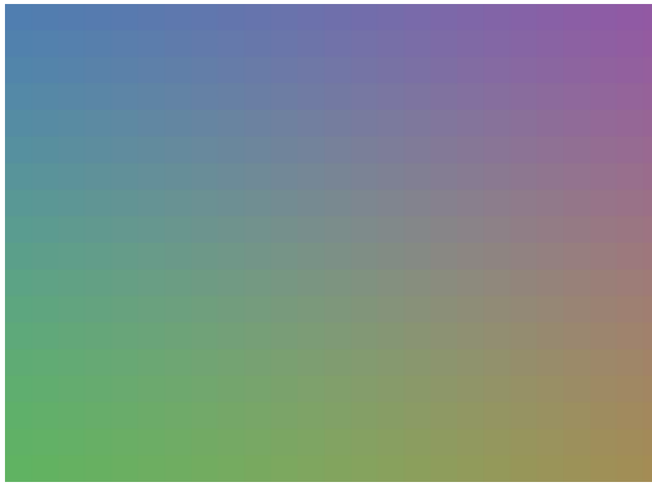


Training SOMs

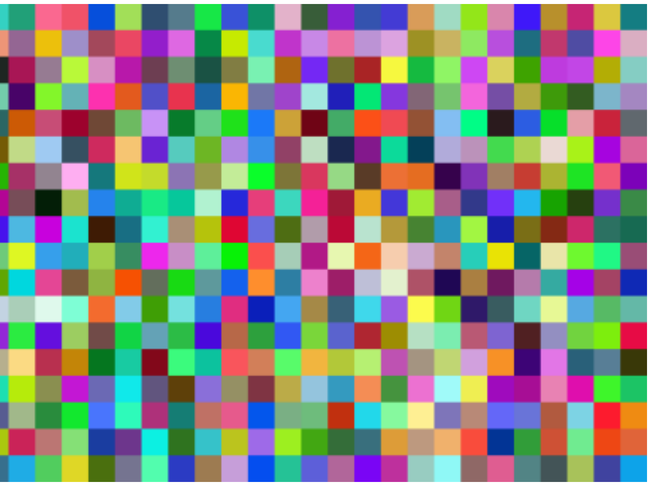


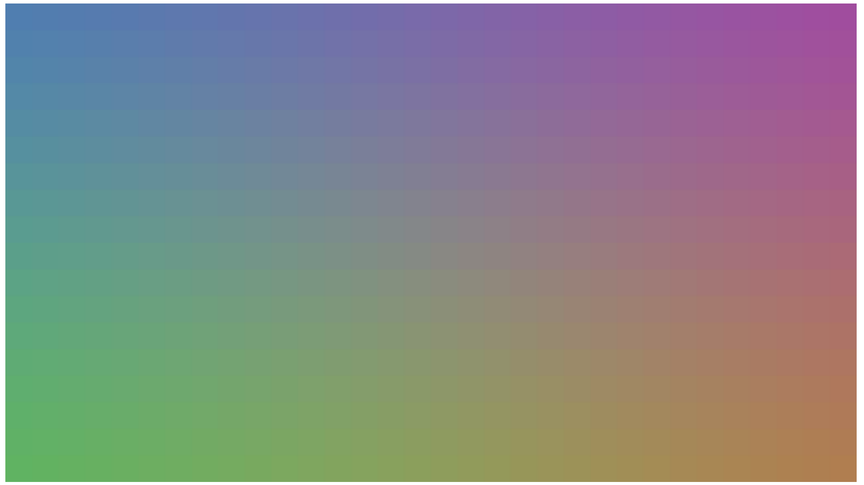
Self-organizing maps

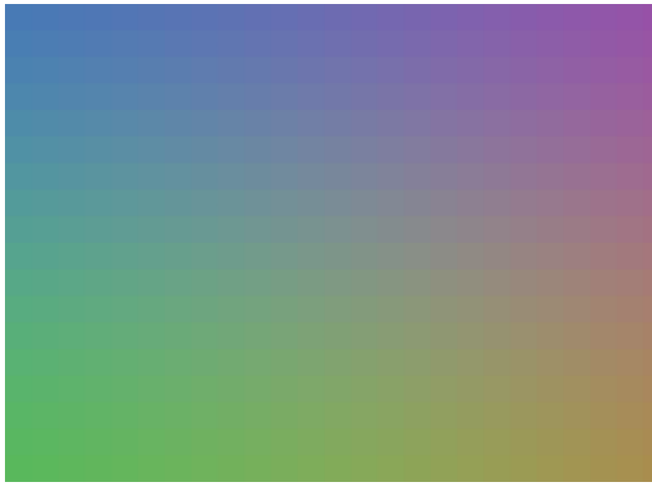




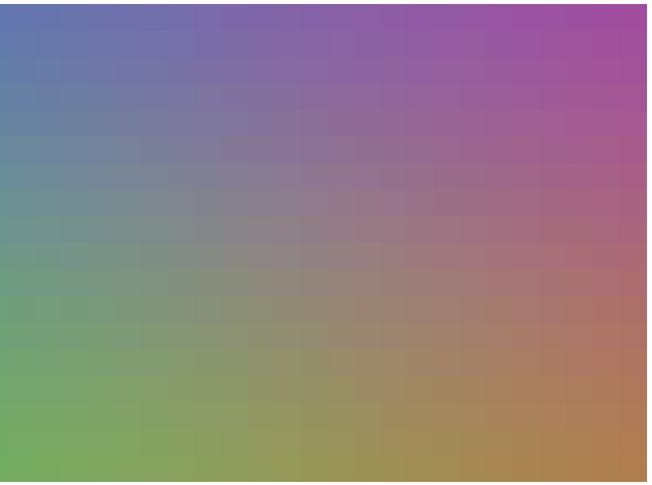
Self-organizing maps

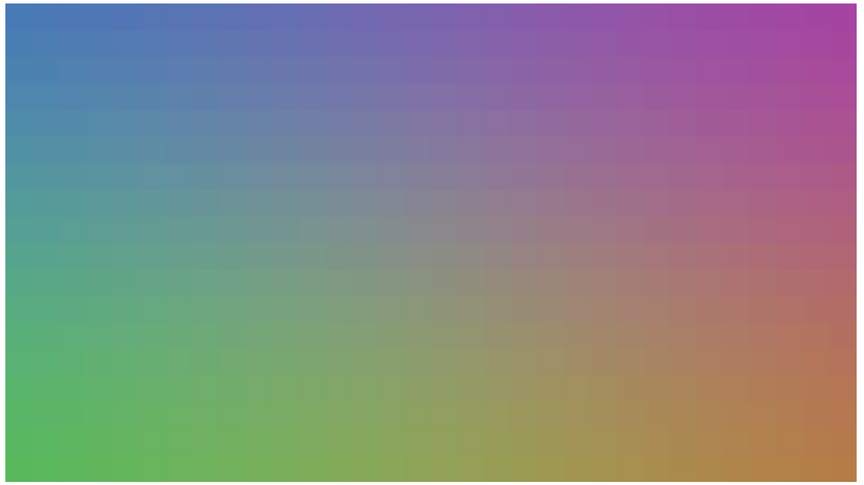


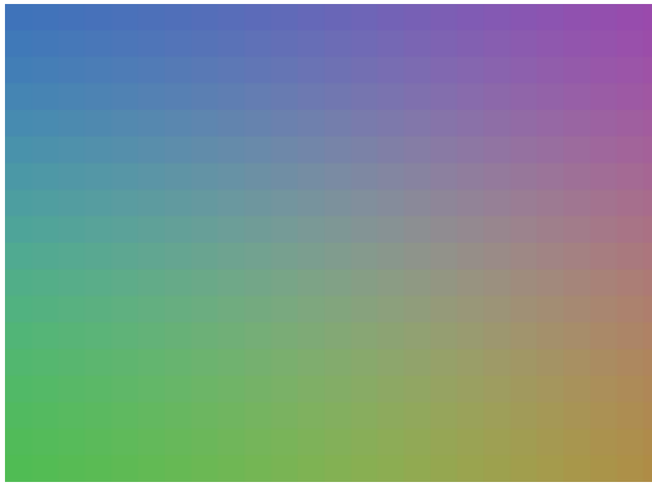




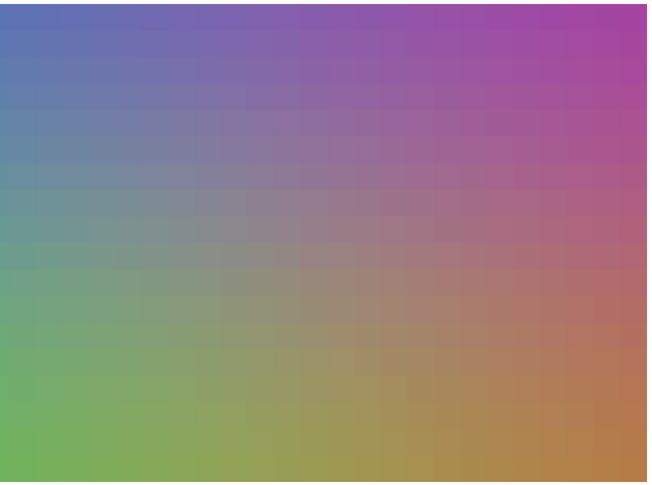
Self-organizing maps

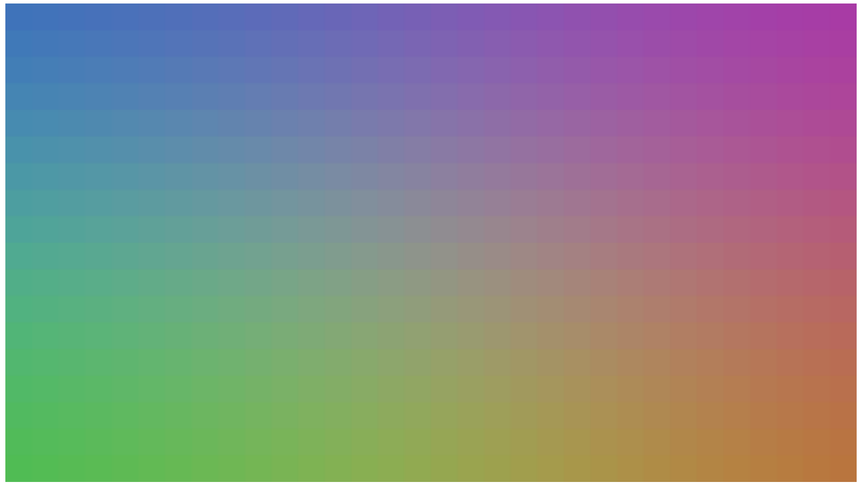


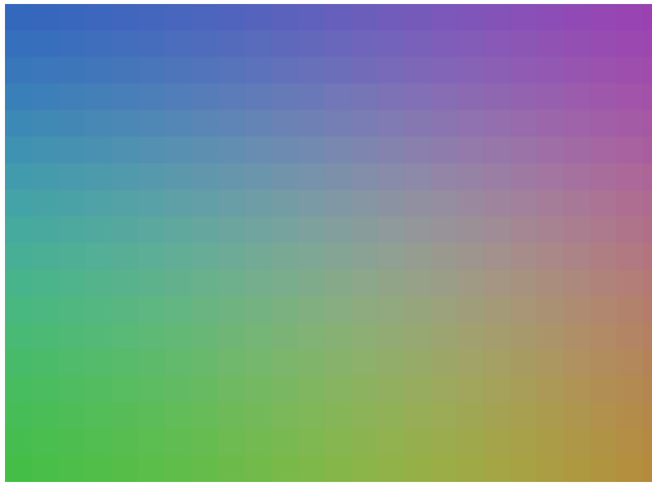




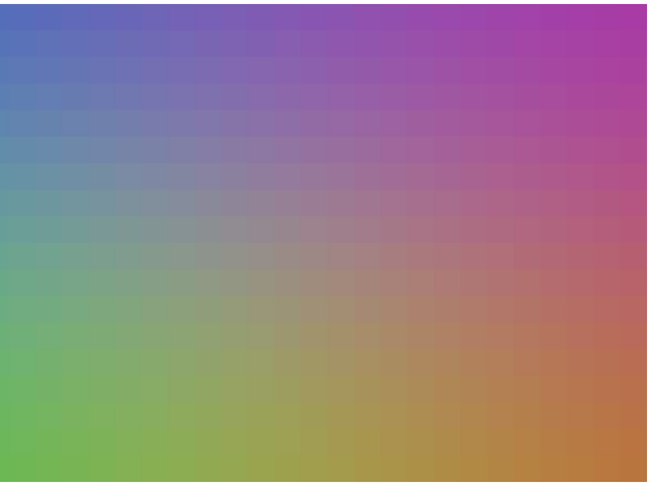
Self-organizing maps

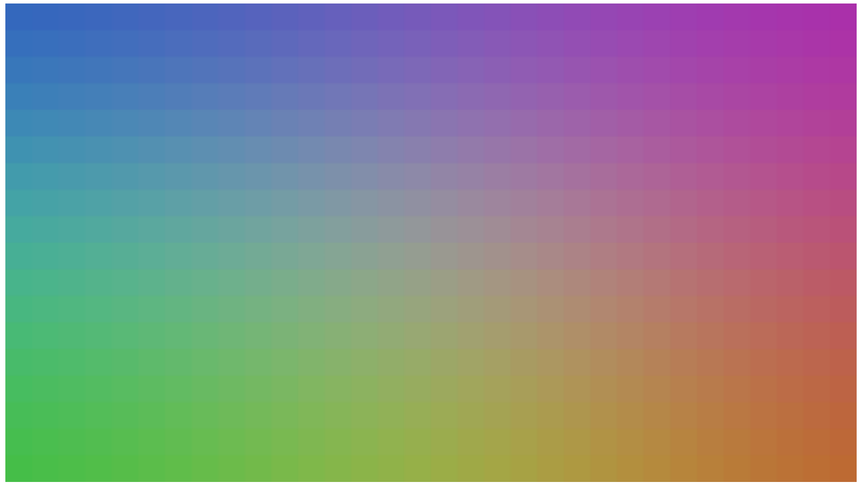


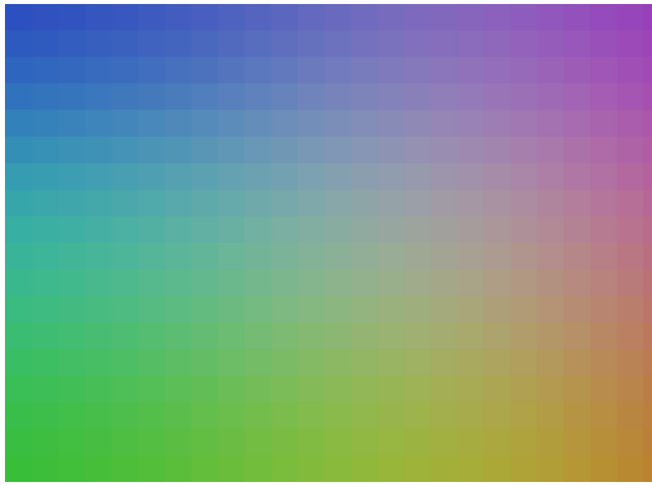




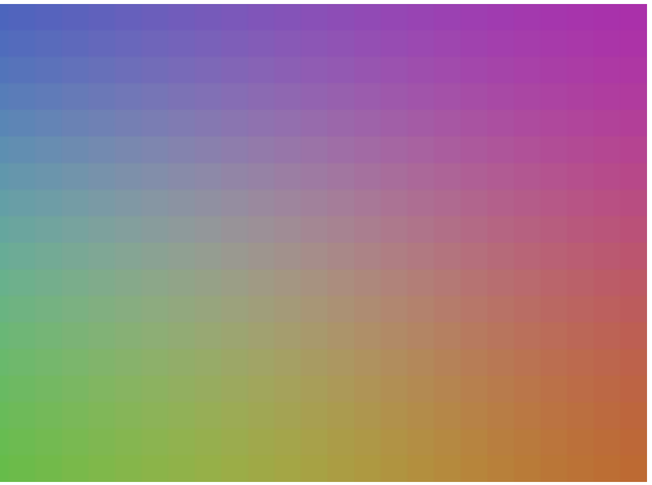
Self-organizing maps

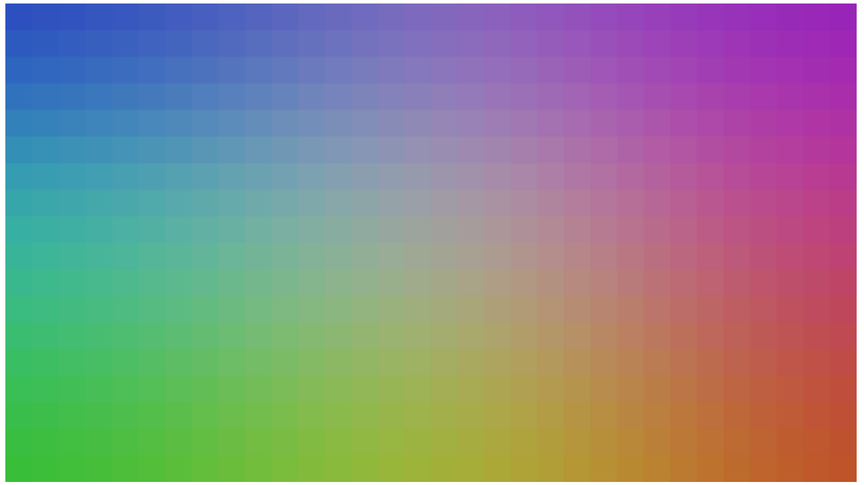


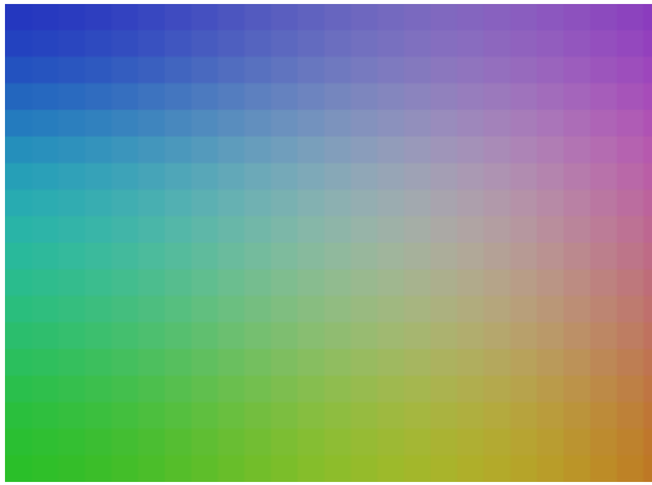




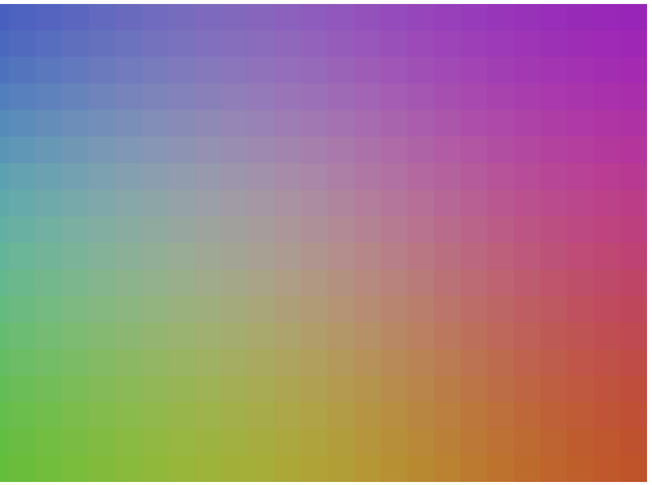
Self-organizing maps

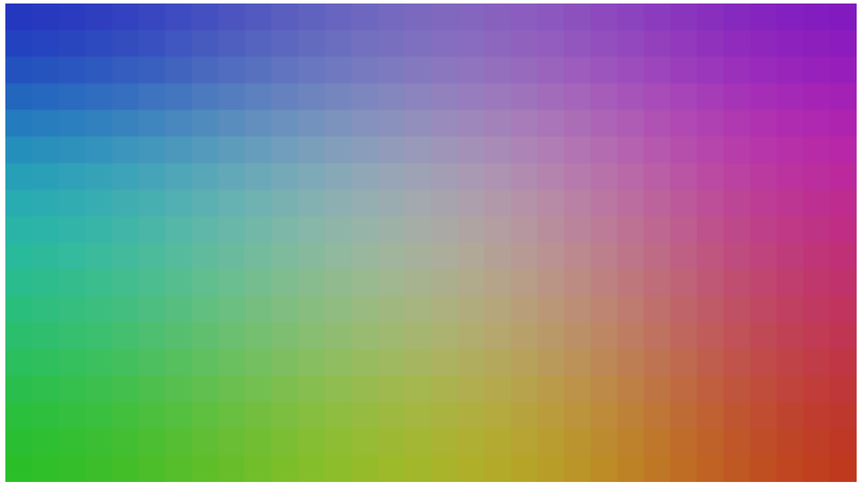


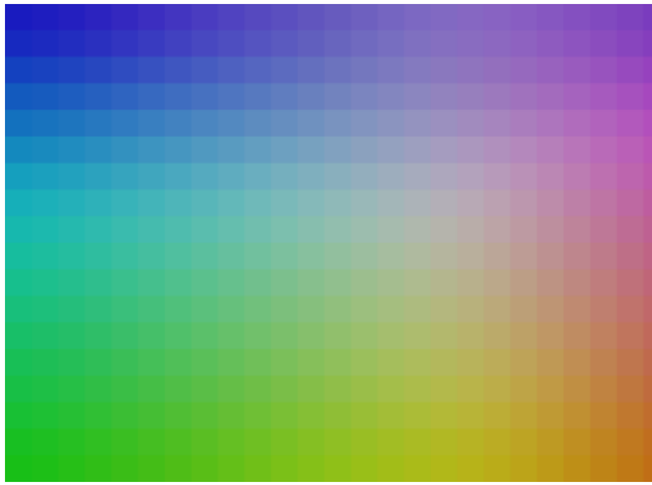




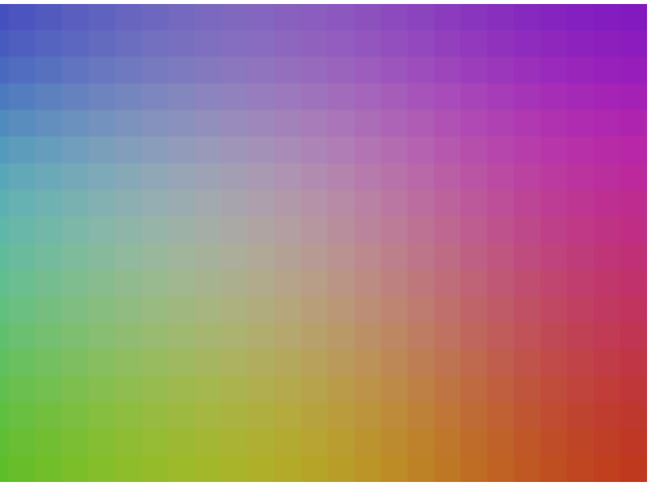
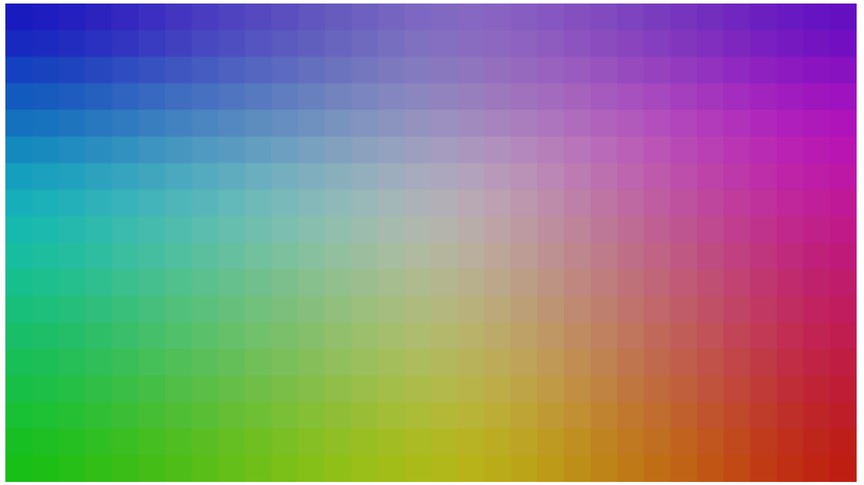
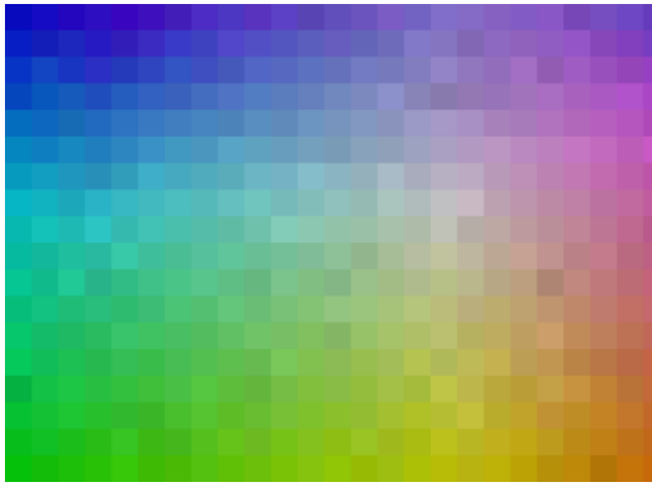
Self-organizing maps



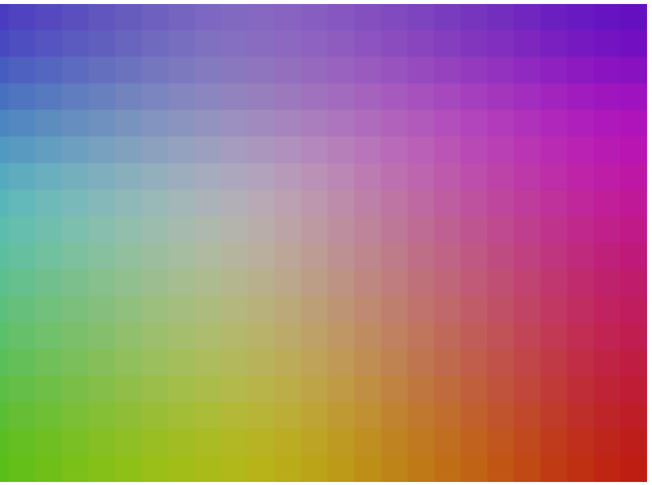


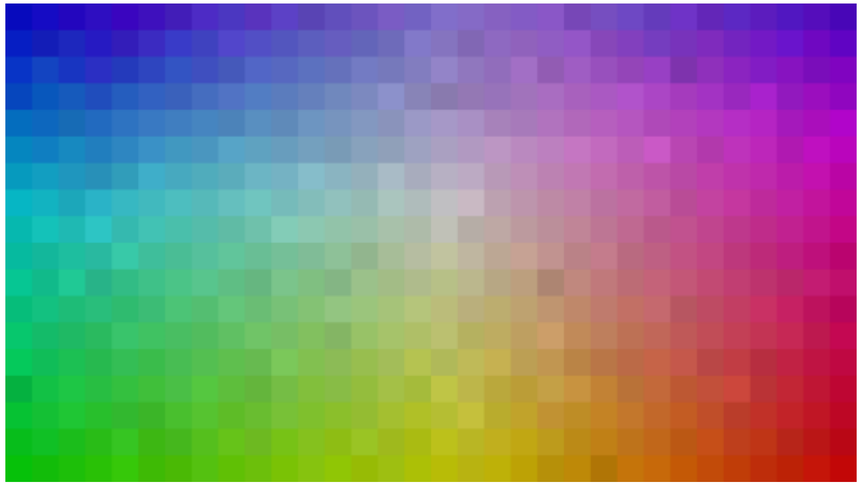


Self-organizing maps

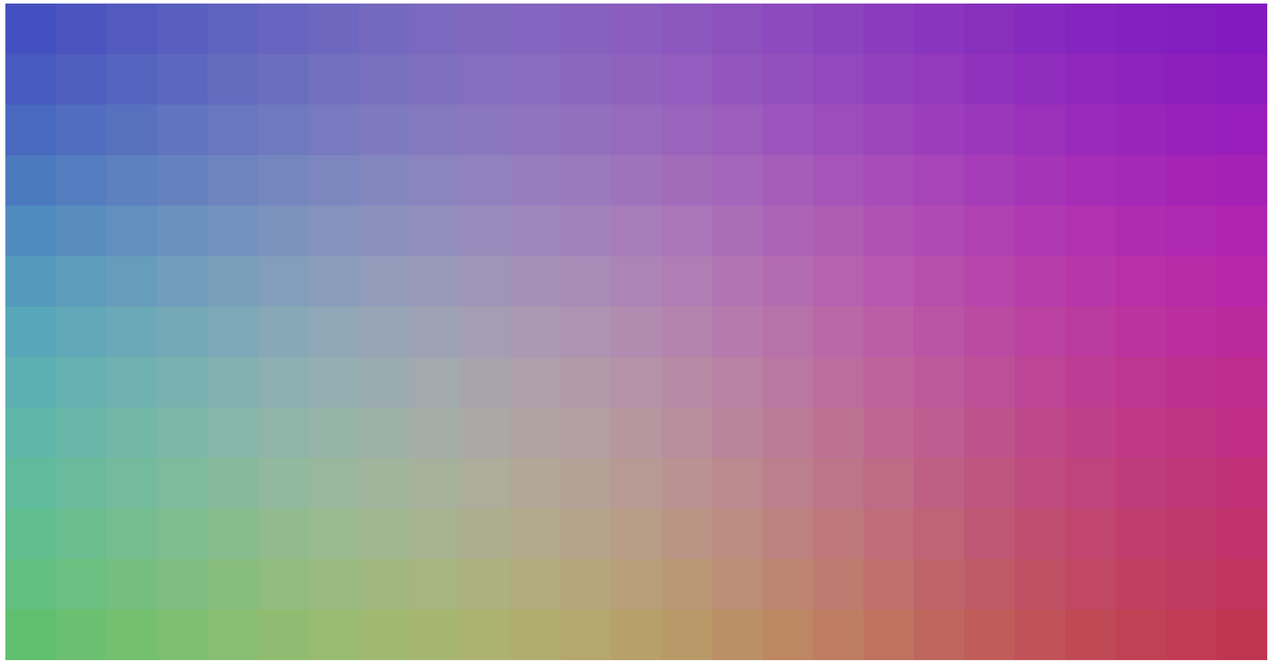




Self-organizing maps

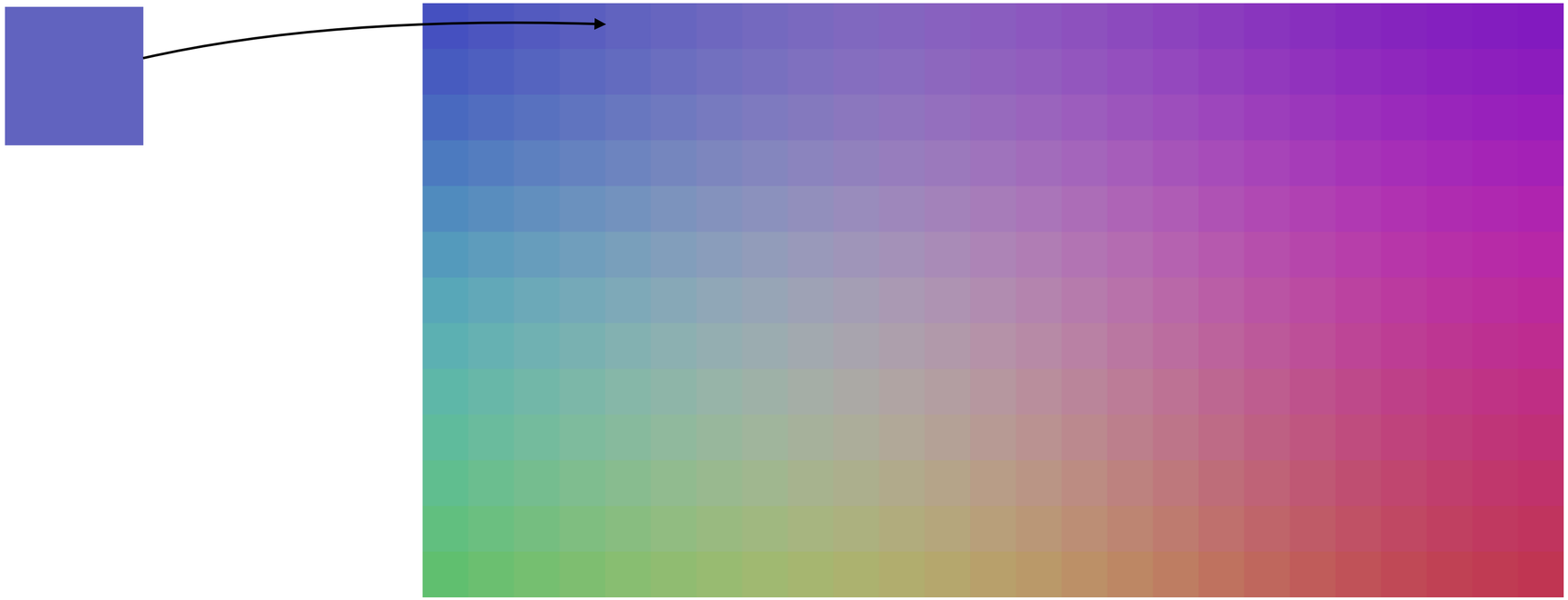




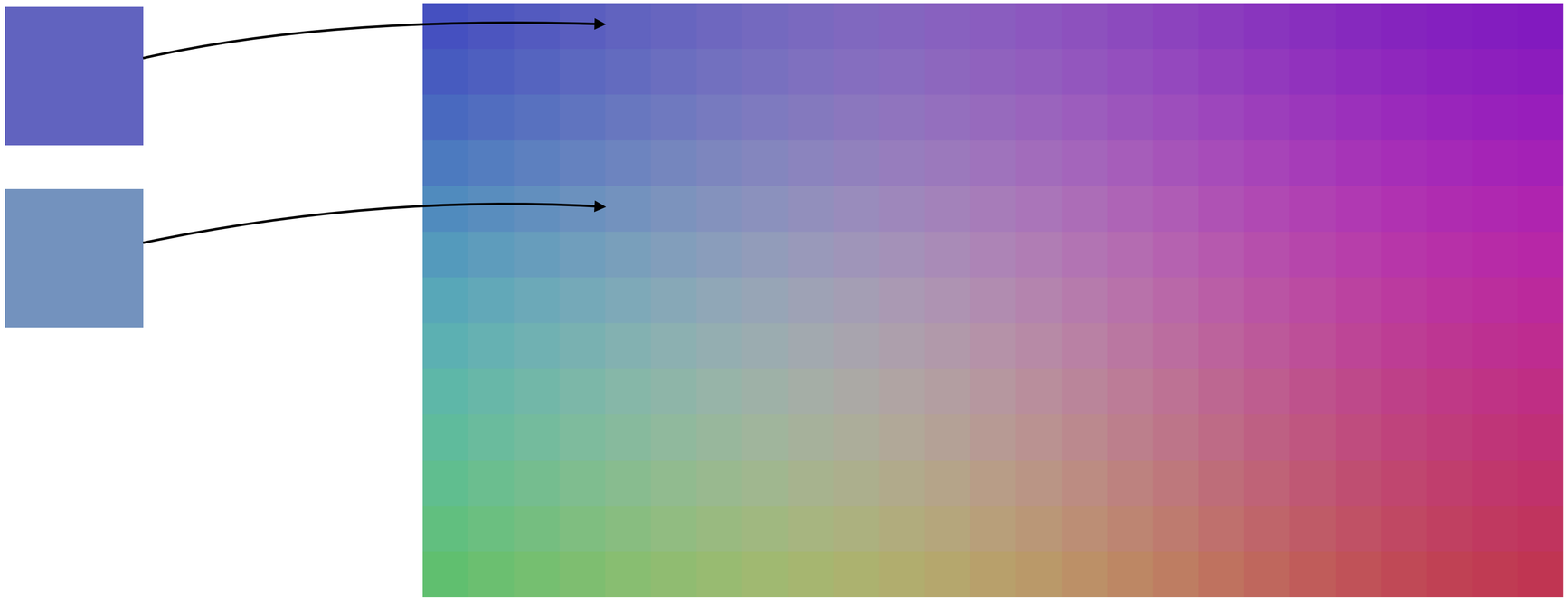
Finding outliers with SOMs



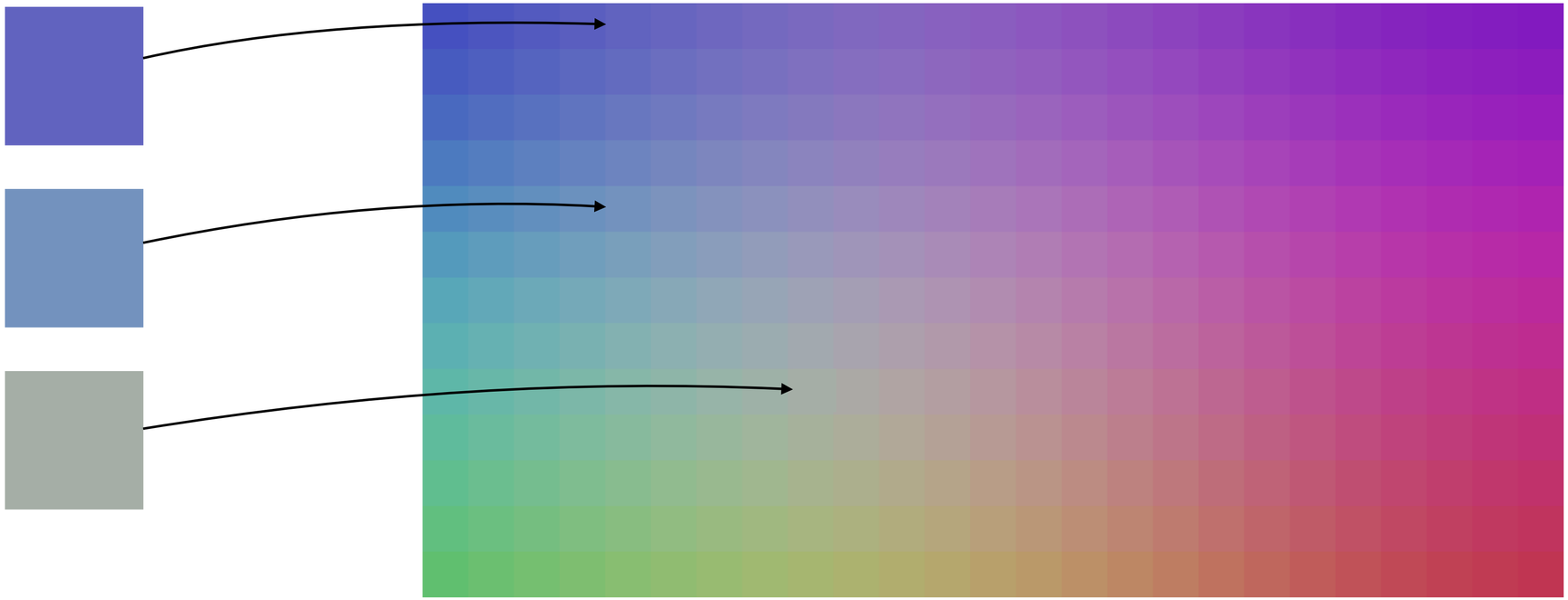
Finding outliers with SOMs



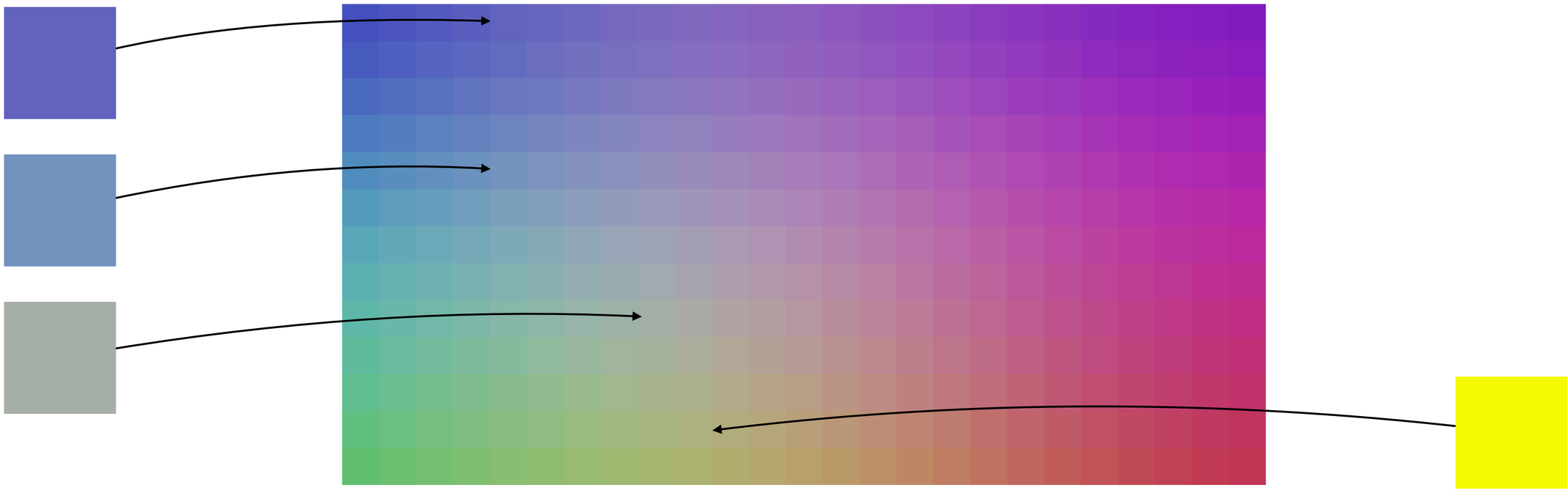
Finding outliers with SOMs



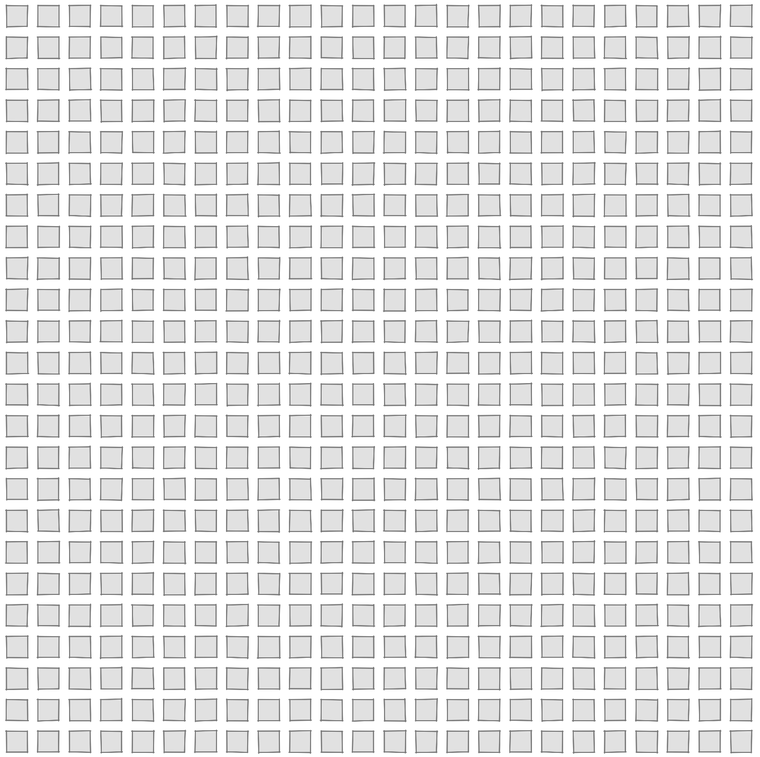
Finding outliers with SOMs



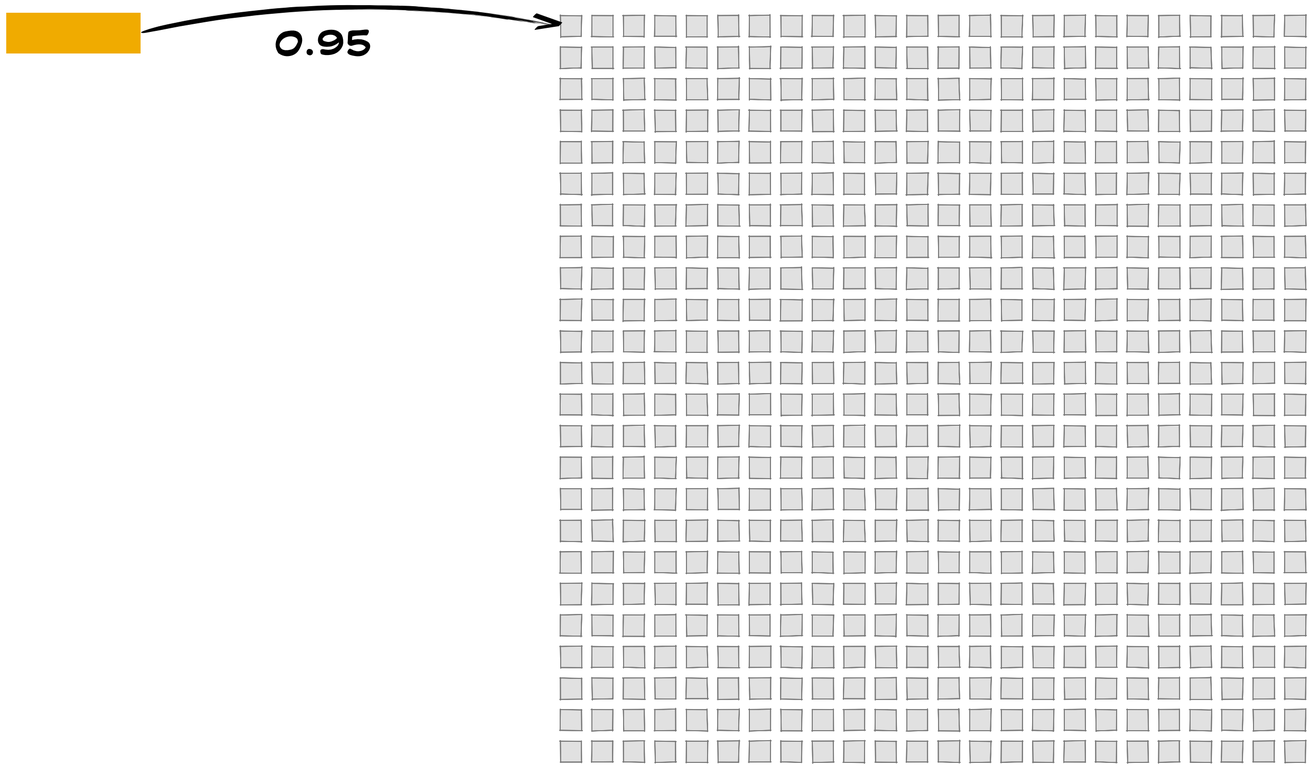
Finding outliers with SOMs



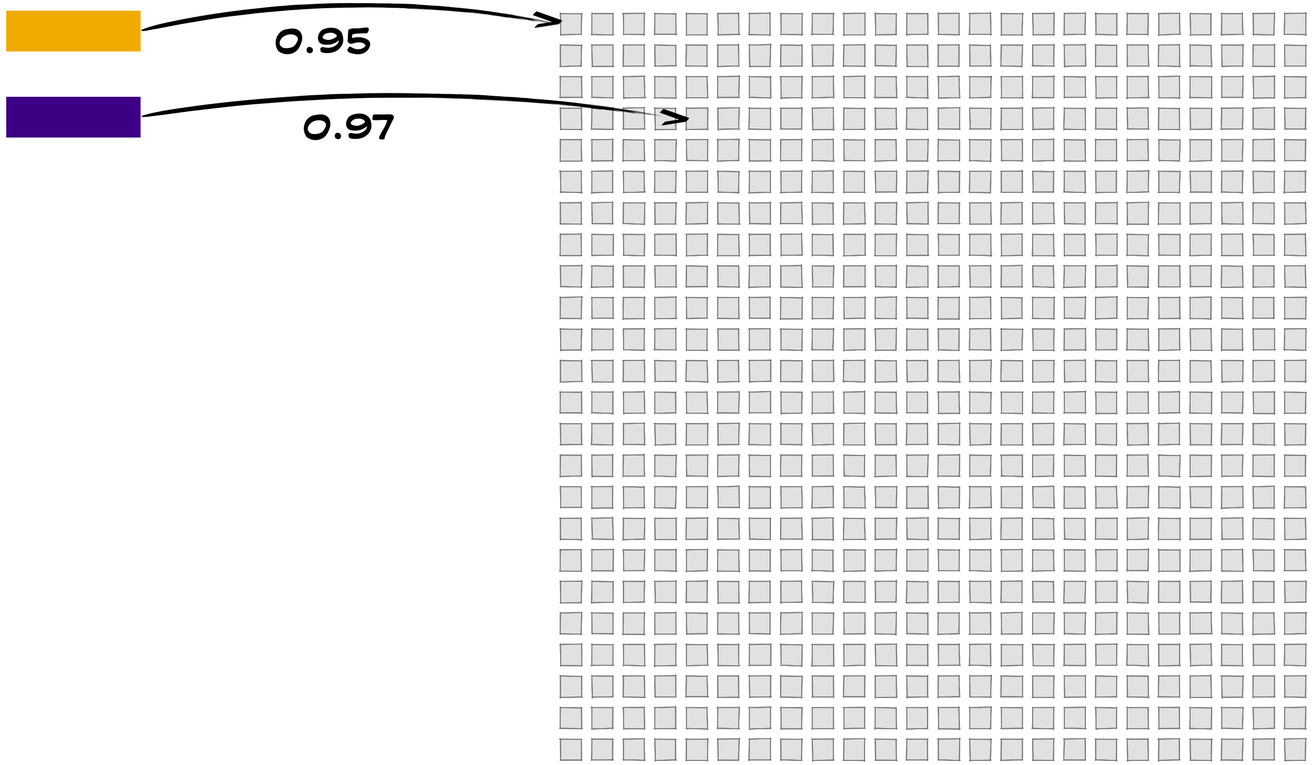
Outliers in log data



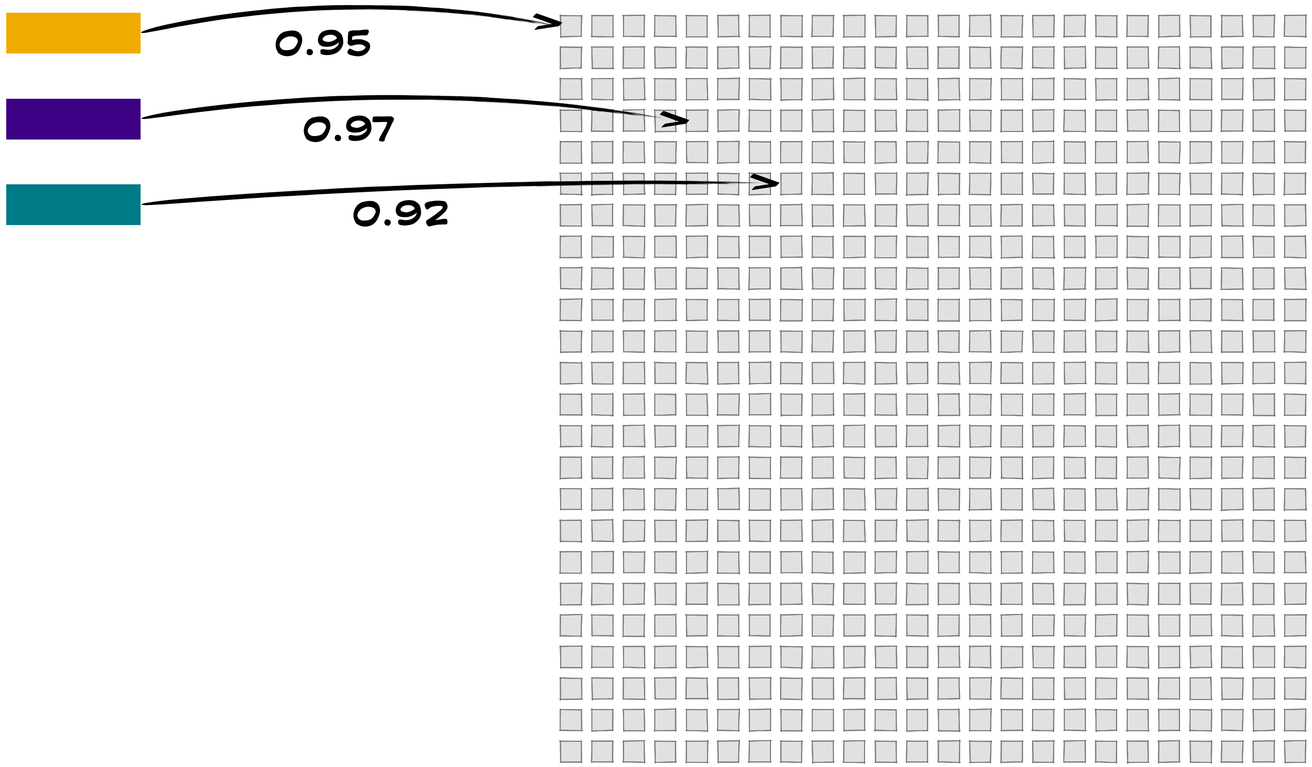
Outliers in log data



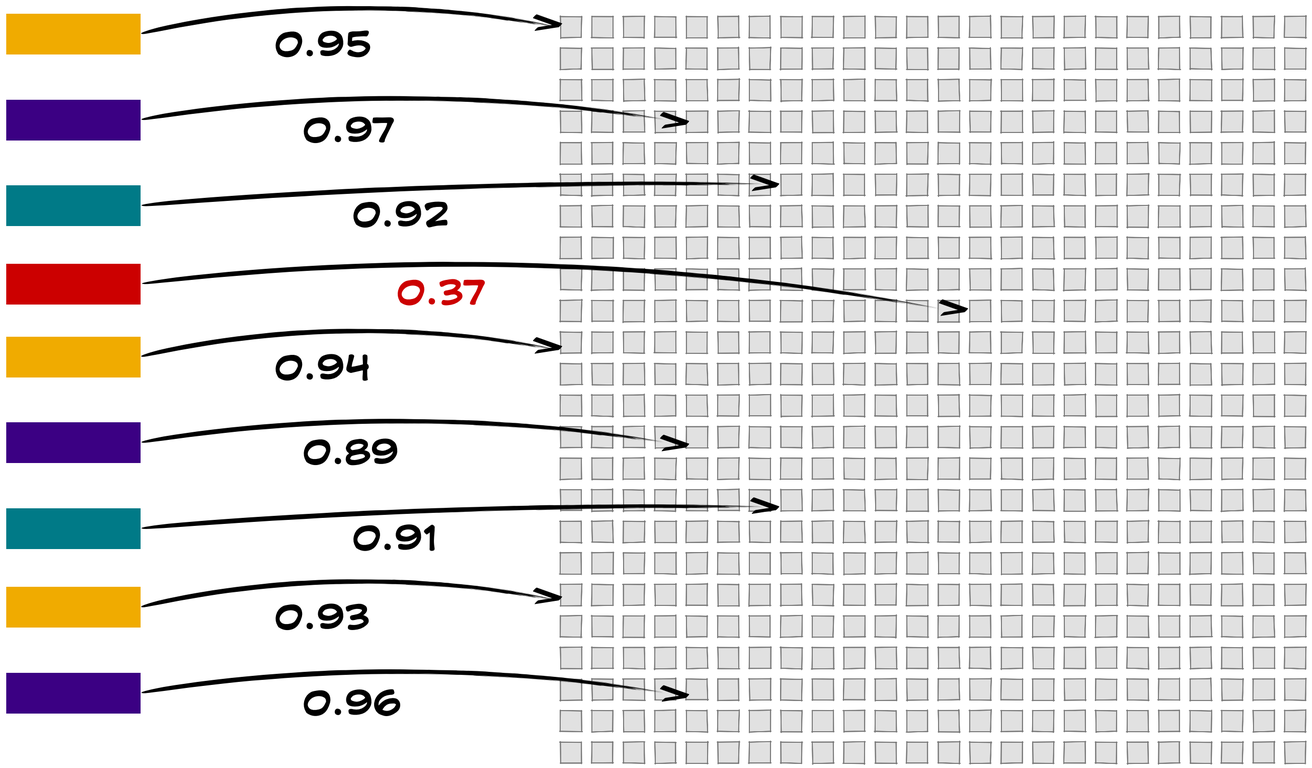
Outliers in log data



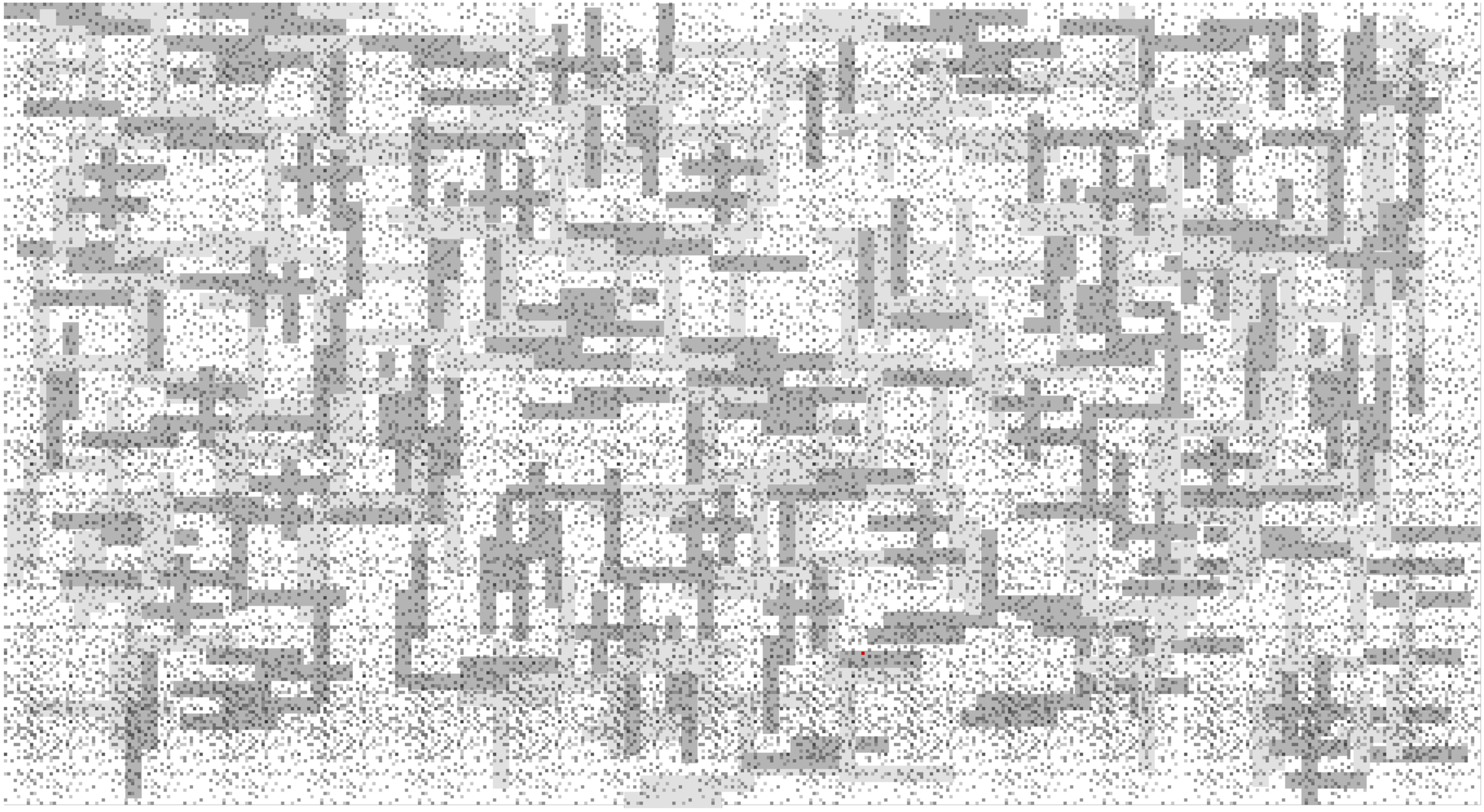
Outliers in log data



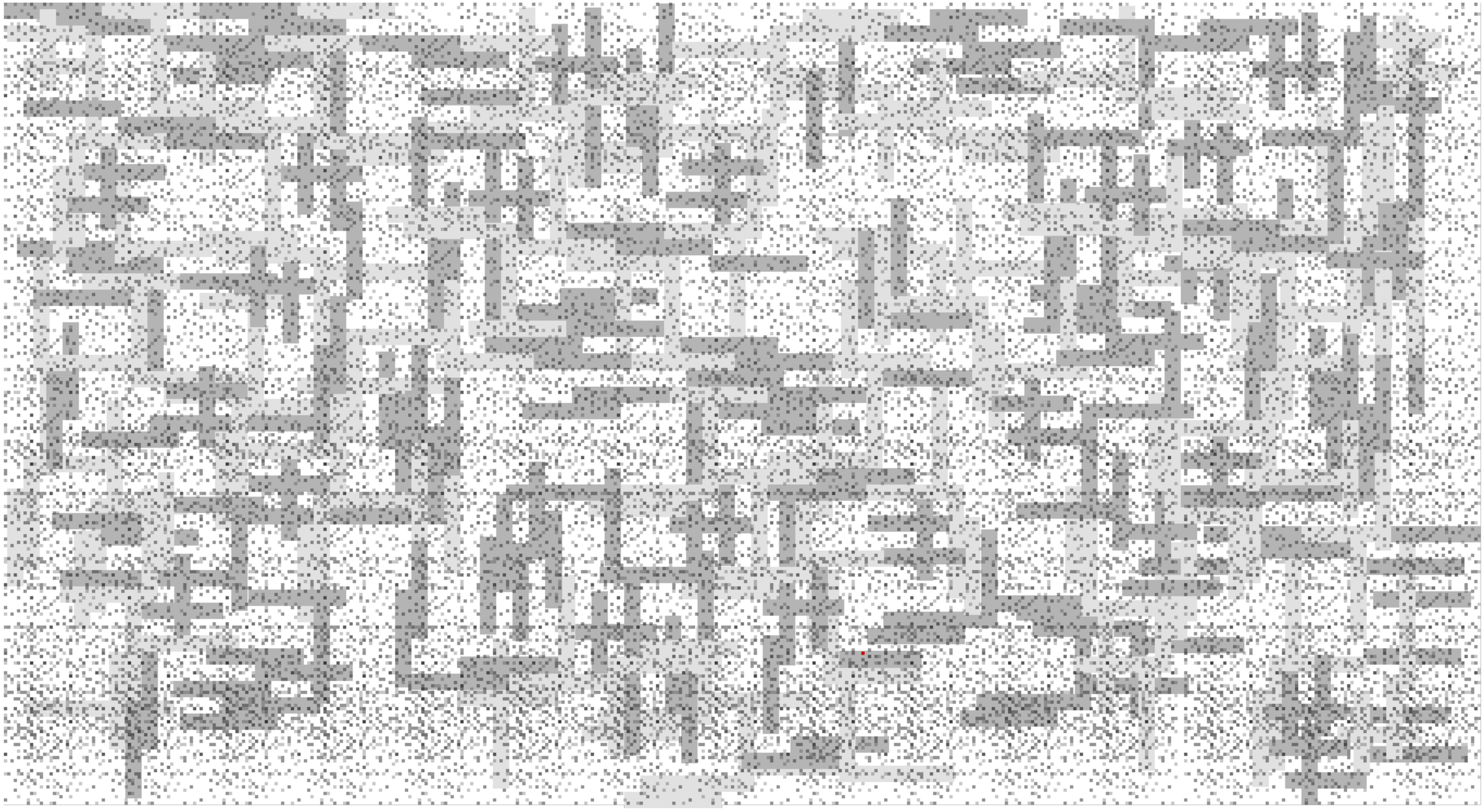
Outliers in log data

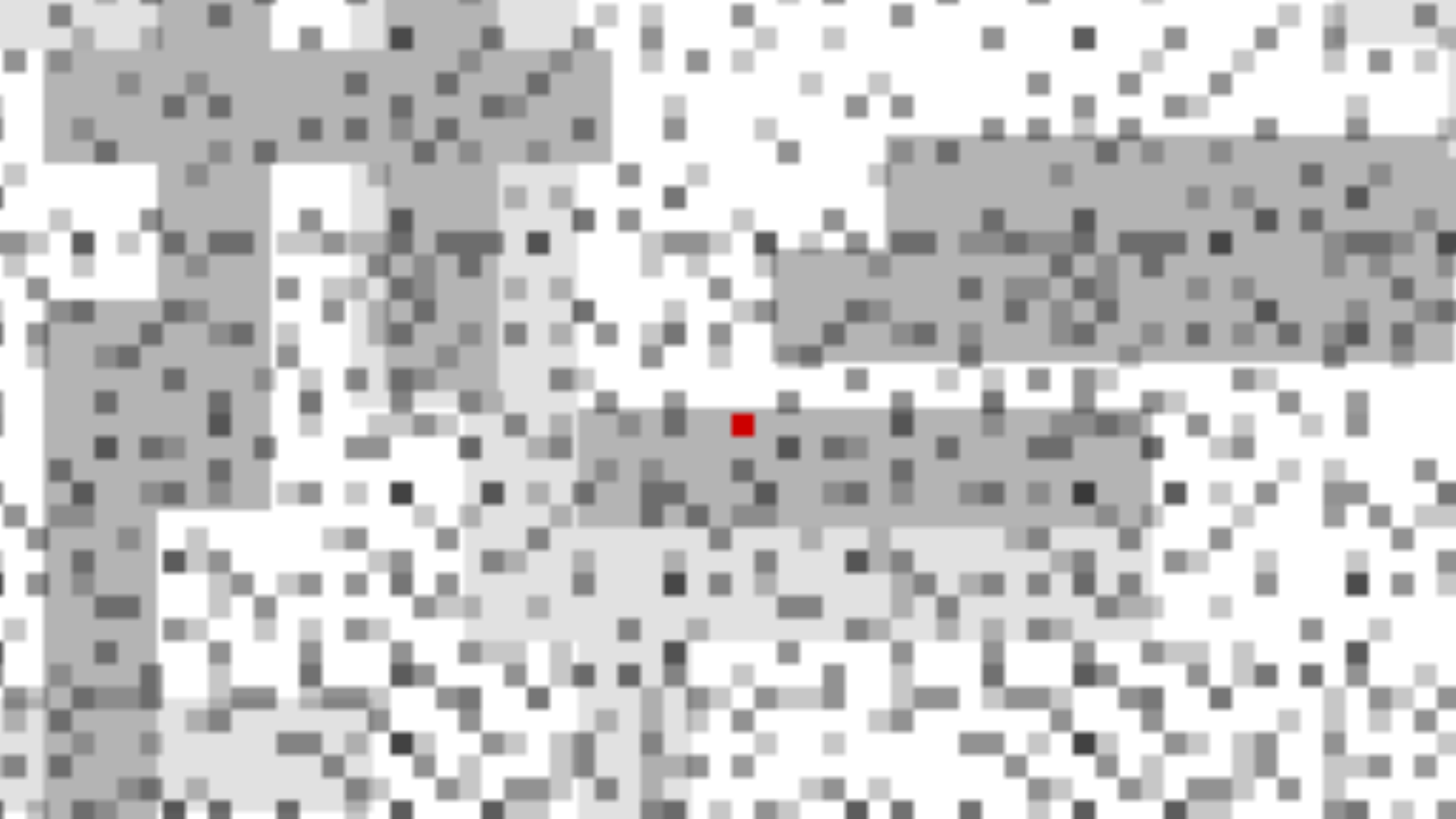


An **outlier** is any record whose best match was **at least** 4σ below the mean.




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.

LESSONS LEARNED and KEY TAKEAWAYS

Spark and ElasticSearch

Data locality is an issue and **caching** is even more important than when running from local storage.

The **best practice** is to warehouse ES indices to Parquet files and write Spark jobs against them.

Structured queries in Spark

Always program defensively: mediate schemas, explicitly convert `null` values, etc.

Use the `Dataset` **API whenever possible** to **minimize boilerplate** and benefit from **query planning** without (entirely) forsaking **type safety**.

Natural language tips

Conventional preprocessing pipelines **probably won't get you the best results** on log message texts.

Consider the sorts of "words" and word variants you'll want to recognize with your preprocessing pipeline.

Use an **approximate whitelist** for unusual words.

Memory and partitioning

Large JVM heaps can lead to **appalling GC pauses.**

You probably won't be able to use all of your memory in a single JVM without **executor timeouts.**

Consider memory budget at the driver when choosing a partitioning strategy.

Feature engineering

Design your feature engineering pipeline so you can **translate feature vectors back to factor values.**

Favor **feature engineering effort** over **complex or novel learning algorithms.**

Prefer approaches that train **interpretable models.**

THANKS!

@willb • willb@redhat.com
https://chapeau.freevariable.com