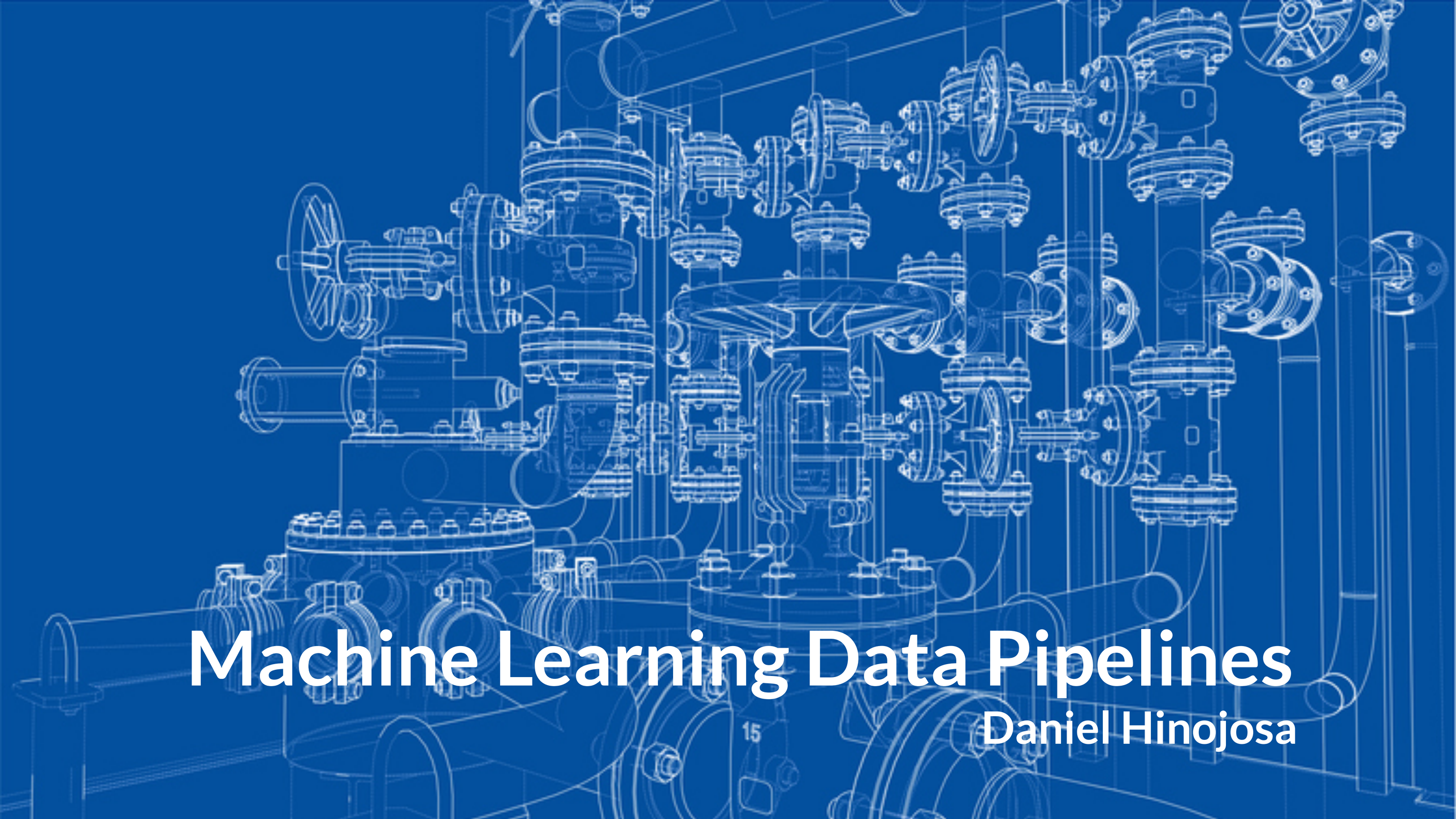


Machine Learning Data Pipelines

Daniel Hinojosa

About Me…

Daniel Hinojosa
Programmer, Consultant, Trainer

Testing in Scala (Book)
Beginning Scala Programming (Video)
Scala Beyond the Basics (Video)

Contact:
dhinojosa@evolutionnext.com
@dhinojosa

Applications

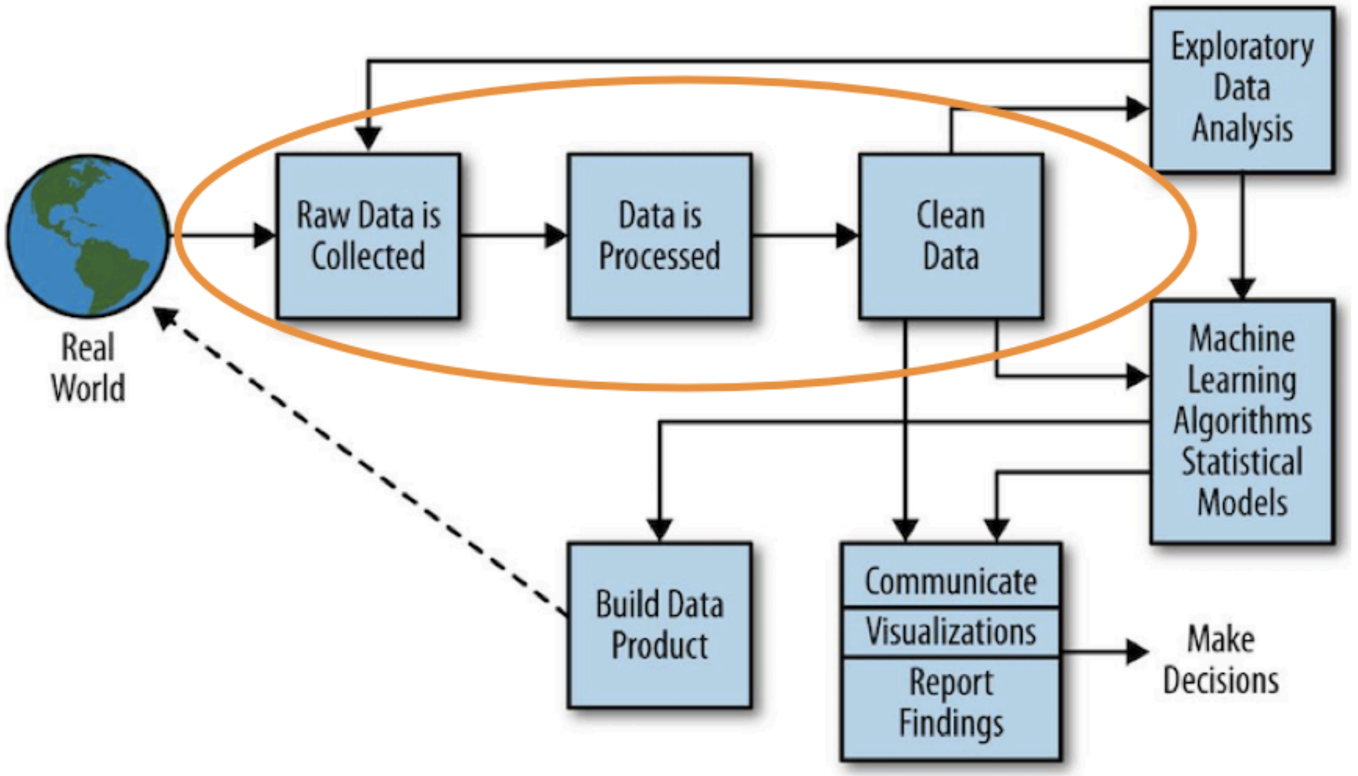
Fraud Detection

Financial Approvals

Picture Recognition

Your Idea Here

Process of Machine Learning


Exploratory
Data
Analysis
Raw Data is
Collected
Data is
Processed
Clean
Data
Real
World
Machine
Learning
Algorithms
Statistical
Models
Build Data
Product
Communicate
Visualizations
Report
Findings
Make
Decisions

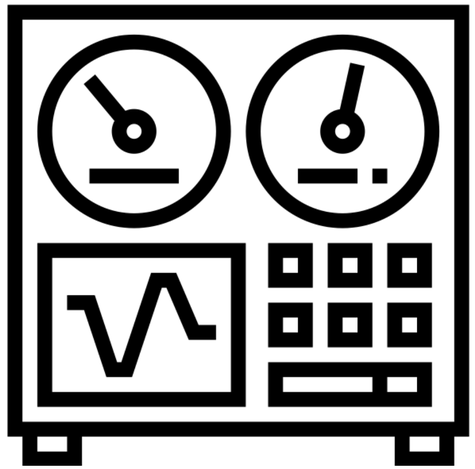
training

0	63	1	233	1
1	37	1	250	0
2	41	0	204	1
3	56	1	236	1
4	57	0	354	0
5	57	1	192	0
6	56	0	294	0
7	44	1	263	1
8	52	1	199	1
9	57	1	168	1

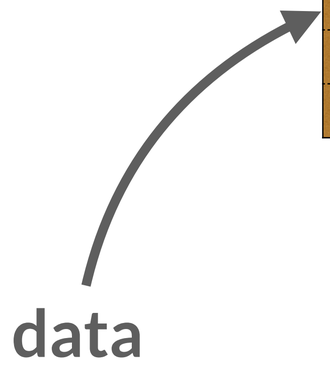
testing

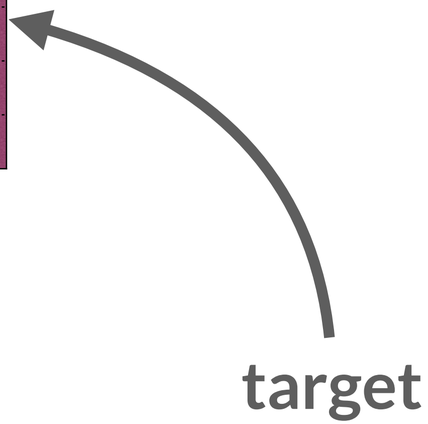
10	54	1	239	1
11	48	0	275	0
12	49	1	266	1
13	64	1	211	0
14	58	0	283	1

training phase



id	age	sex	trestbps	risk
0	20	1	145	1
1	45	1	130	1
2	43	0	130	0
3	33	1	150	0





training

testing

id	age	sex	trestbps	target
0	63	1	233	1
1	37	1	250	0
2	41	0	204	1
3	56	1	236	1
4	57	0	354	0
5	57	1	192	0
6	56	0	294	0
7	44	1	263	1
8	52	1	199	1
9	57	1	168	1
10	54	1	239	1
11	48	0	275	0
12	49	1	266	1
13	64	1	211	0
14	58	0	283	1

id	age	sex	trestbps	target
0	63	1	233	1
1	37	1	250	0
2	41	0	204	1
3	56	1	236	1
4	57	0	354	0
5	57	1	192	0
6	56	0	294	0
7	44	1	263	1
8	52	1	199	1
9	57	1	168	1
10	54	1	239	1
11	48	0	275	0
12	49	1	266	1
13	64	1	211	0
14	58	0	283	1

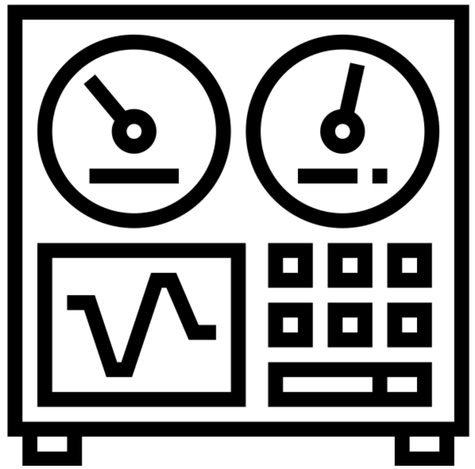
random distribution between training and testing

training

0	63	1	233	1
1	37	1	250	0
2	41	0	204	1
3	56	1	236	1
4	57	0	354	0
5	57	1	192	0
6	56	0	294	0
7	44	1	263	1
8	52	1	199	1
9	57	1	168	1

testing

10	54	1	239	1
11	48	0	275	0
12	49	1	266	1
13	64	1	211	0
14	58	0	283	1



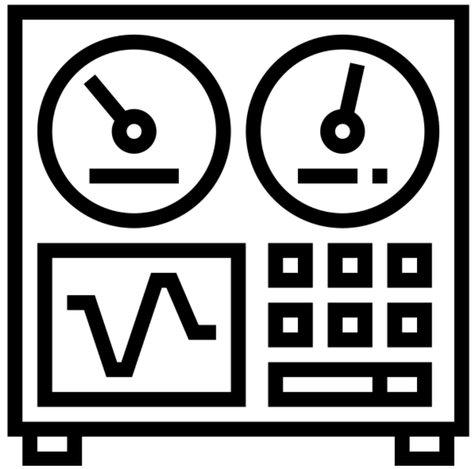
training

0	63	1	233	1
1	37	1	250	0
2	41	0	204	1
3	56	1	236	1
4	57	0	354	0
5	57	1	192	0
6	56	0	294	0
7	44	1	263	1
8	52	1	199	1
9	57	1	168	1

testing

10	54	1	239	1
11	48	0	275	0
12	49	1	266	1
13	64	1	211	0
14	58	0	283	1

model



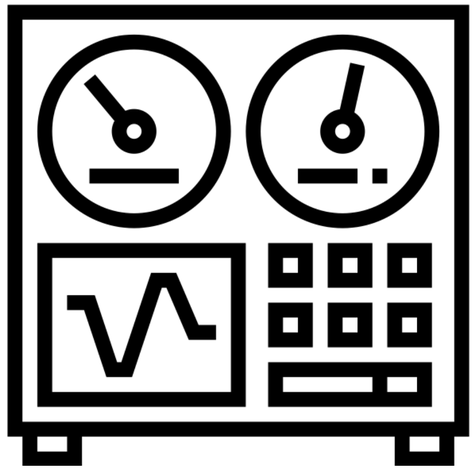
training

0	63	1	233	1
1	37	1	250	0
2	41	0	204	1
3	56	1	236	1
4	57	0	354	0
5	57	1	192	0
6	56	0	294	0
7	44	1	263	1
8	52	1	199	1
9	57	1	168	1

testing

10	54	1	239	1
11	48	0	275	0
12	49	1	266	1
13	64	1	211	0
14	58	0	283	1

training phase

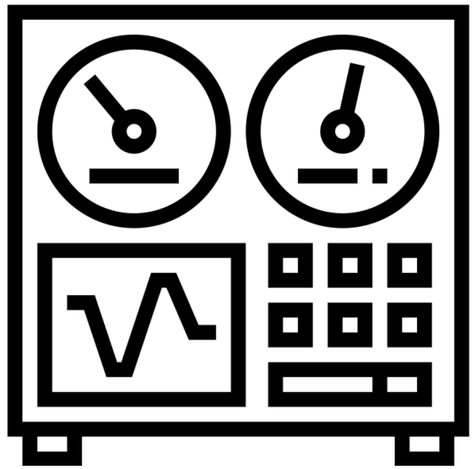


training

0	63	1	233	1
1	37	1	250	0
2	41	0	204	1
3	56	1	236	1
4	57	0	354	0
5	57	1	192	0
6	56	0	294	0
7	44	1	263	1
8	52	1	199	1
9	57	1	168	1

testing

10	54	1	239	1
11	48	0	275	0
12	49	1	266	1
13	64	1	211	0
14	58	0	283	1



training

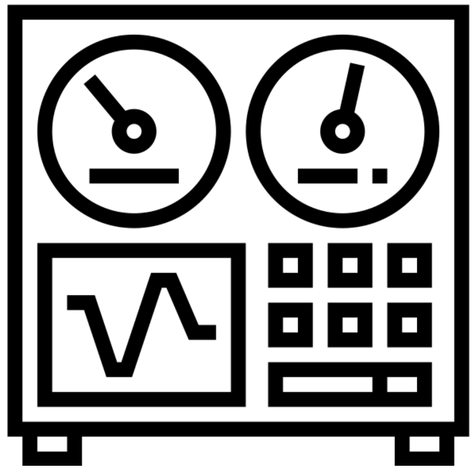
0	63	1	233	1
1	37	1	250	0
2	41	0	204	1
3	56	1	236	1
4	57	0	354	0
5	57	1	192	0
6	56	0	294	0
7	44	1	263	1
8	52	1	199	1
9	57	1	168	1

testing

10	54	1	239
11	48	0	275
12	49	1	266
13	64	1	211
14	58	0	283

actual

1
0
1
0
1



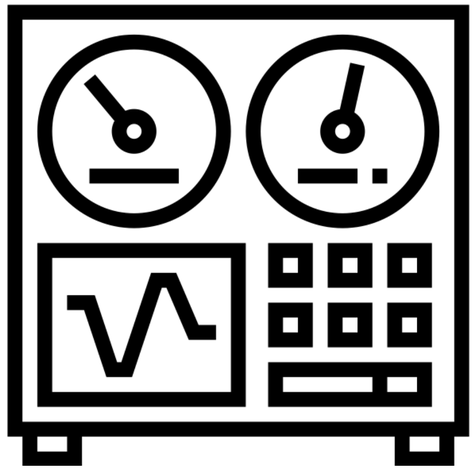
reserve the actual later for verification

training

0	63	1	233	1
1	37	1	250	0
2	41	0	204	1
3	56	1	236	1
4	57	0	354	0
5	57	1	192	0
6	56	0	294	0
7	44	1	263	1
8	52	1	199	1
9	57	1	168	1

testing

10	54	1	239
11	48	0	275
12	49	1	266
13	64	1	211
14	58	0	283



testing phase

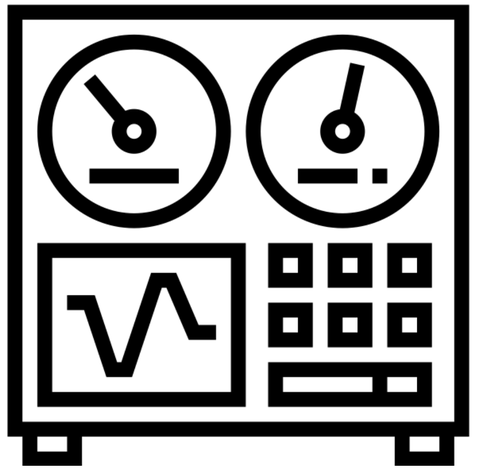
training

0	63	1	233	1
1	37	1	250	0
2	41	0	204	1
3	56	1	236	1
4	57	0	354	0
5	57	1	192	0
6	56	0	294	0
7	44	1	263	1
8	52	1	199	1
9	57	1	168	1

testing

10	54	1	239
11	48	0	275
12	49	1	266
13	64	1	211
14	58	0	283

!!!!!!!



generated result

1
0
1
1
1

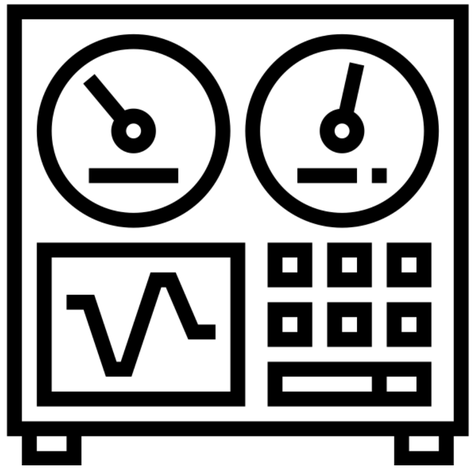
training

0	63	1	233	1
1	37	1	250	0
2	41	0	204	1
3	56	1	236	1
4	57	0	354	0
5	57	1	192	0
6	56	0	294	0
7	44	1	263	1
8	52	1	199	1
9	57	1	168	1

testing

10	54	1	239
11	48	0	275
12	49	1	266
13	64	1	211
14	58	0	283

!!!!!!



generated result

1
0
1
1
1

actual

1
0
1
0
1

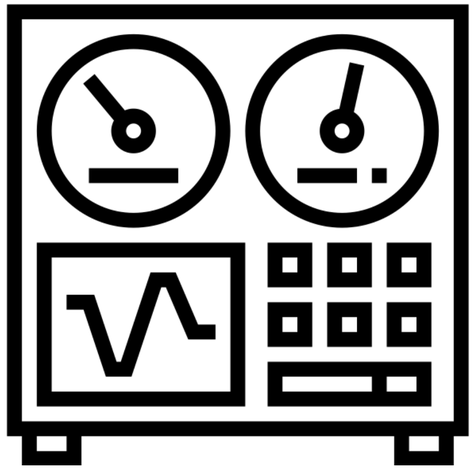
training

0	63	1	233	1
1	37	1	250	0
2	41	0	204	1
3	56	1	236	1
4	57	0	354	0
5	57	1	192	0
6	56	0	294	0
7	44	1	263	1
8	52	1	199	1
9	57	1	168	1

testing

10	54	1	239
11	48	0	275
12	49	1	266
13	64	1	211
14	58	0	283

!!!!!!!



generated result

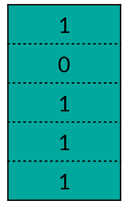
1
0
1
1
1

actual

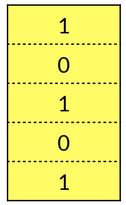
1
0
1
0
1

How did we do?

generated result



actual



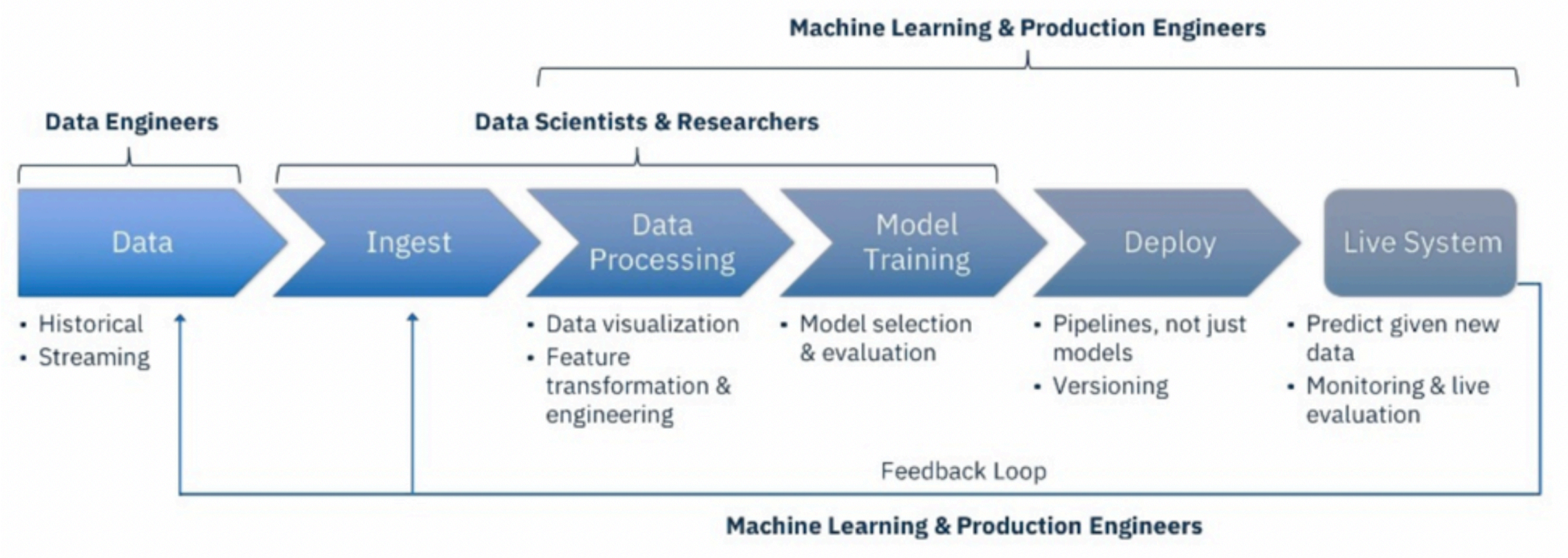
How did we do?

Machine Learning Models

Machine Learning	Deep Learning
KNN	Convolutional Neural Networks
Naive Bayes	Recurrent Neural Networks
Linear Regression	Autoencoders

Roles and Responsibilities

Roles and Responsibilities



What is MLOps?

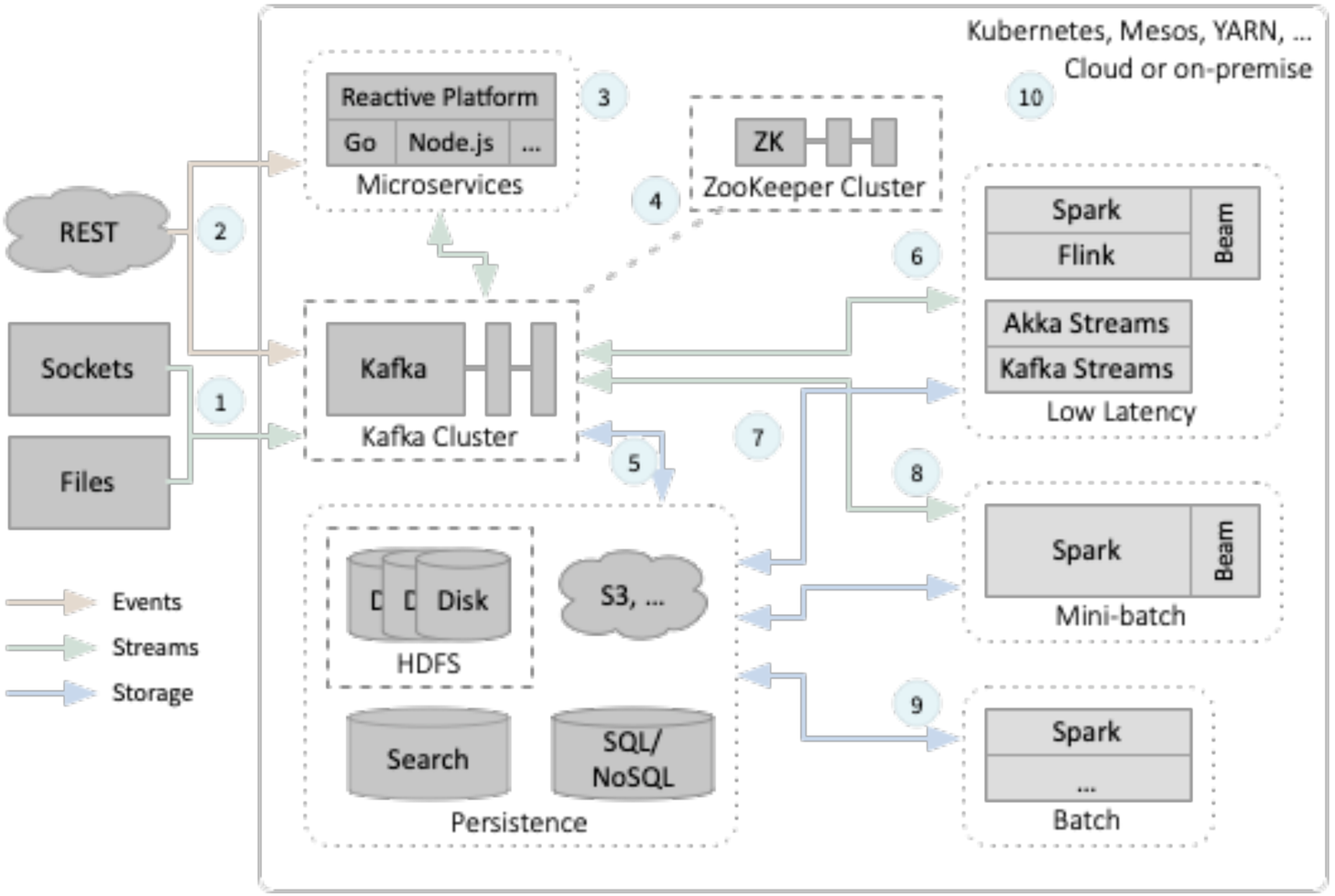
MLOps (a compound of “machine learning” and “operations”) is a practice for collaboration and communication between data scientists and operations professionals to help manage production ML lifecycles.

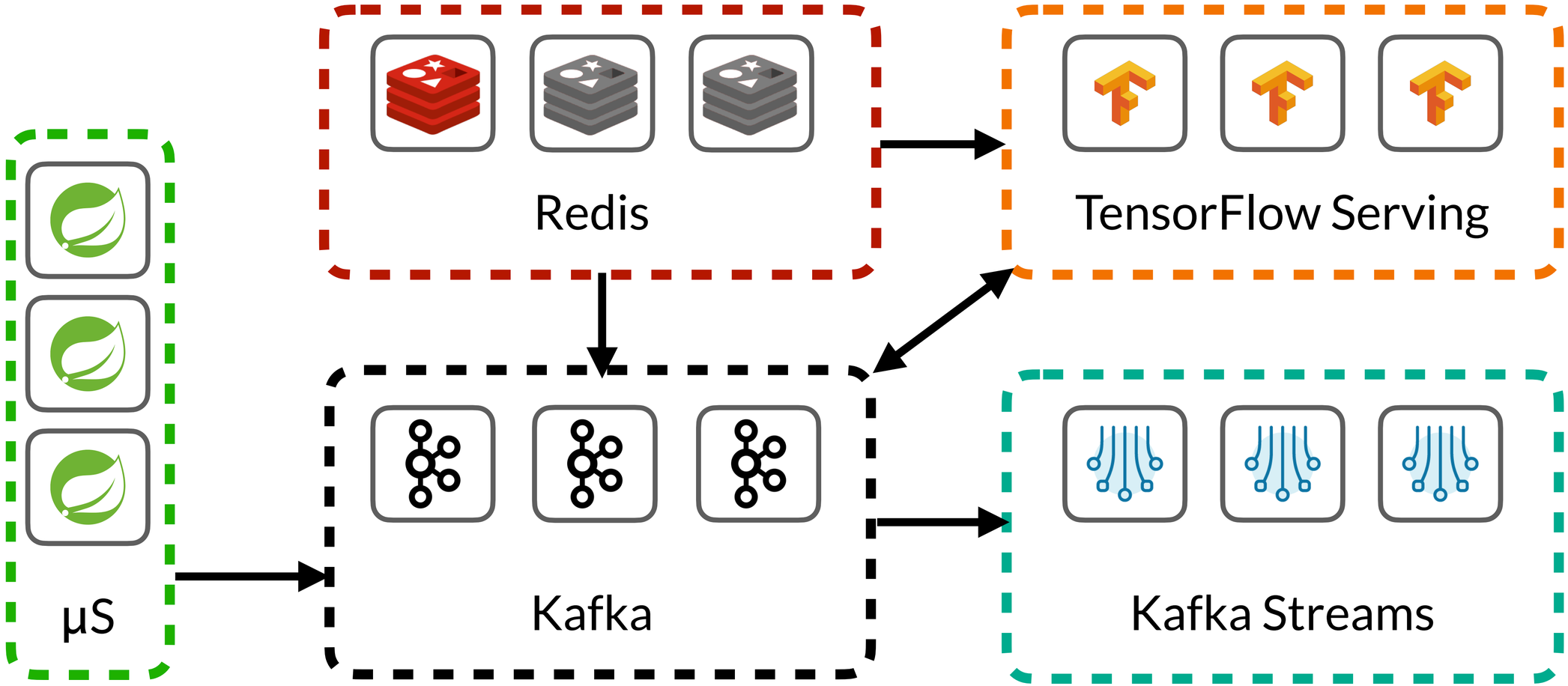
Why the need for communication?


Java™

Architecture

Dean Wampler Architecture




Redis
TensorFlow Serving
µS
Kafka
Kafka Streams

Tensor Flow & Deep Learning

Tensor

Scalar

1

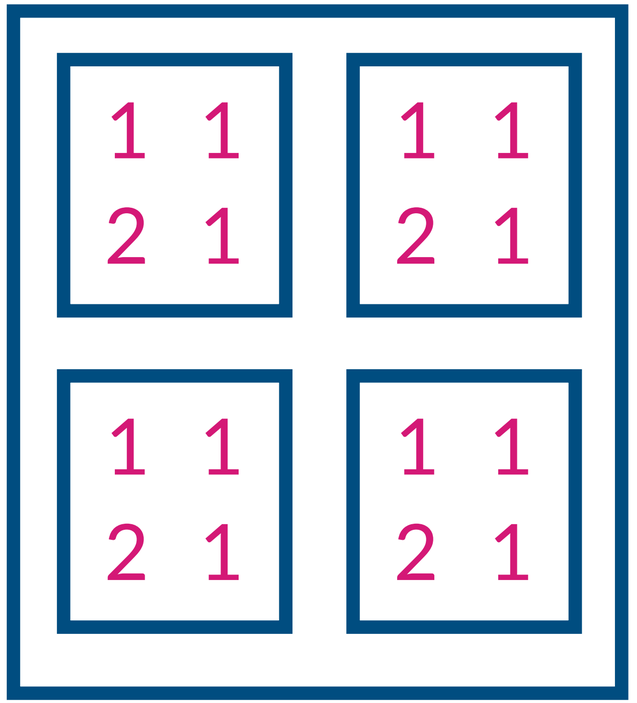
Vector

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

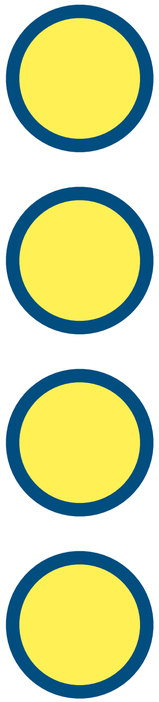
Matrix

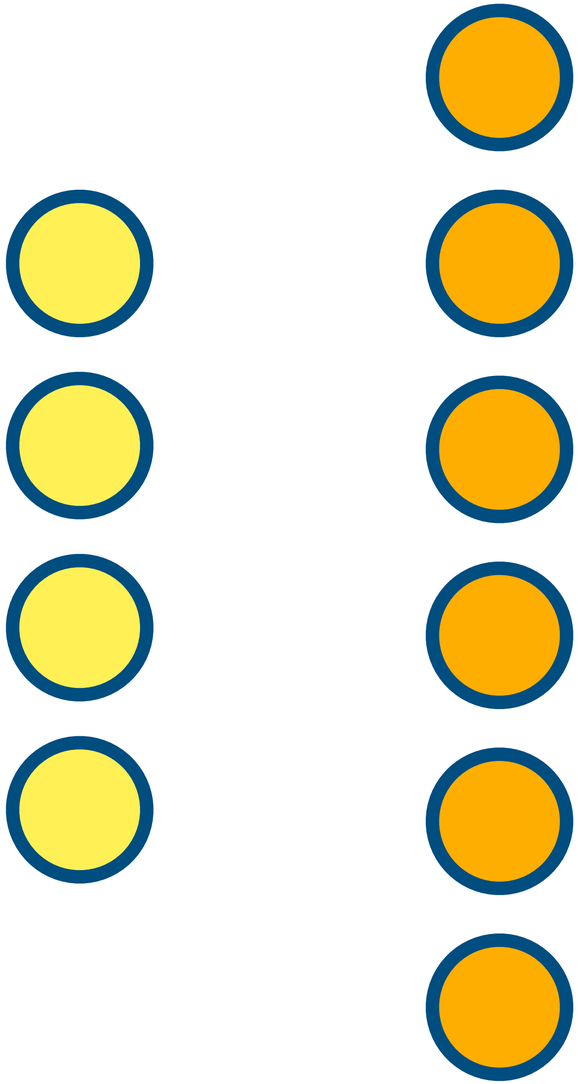
$$\begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}$$

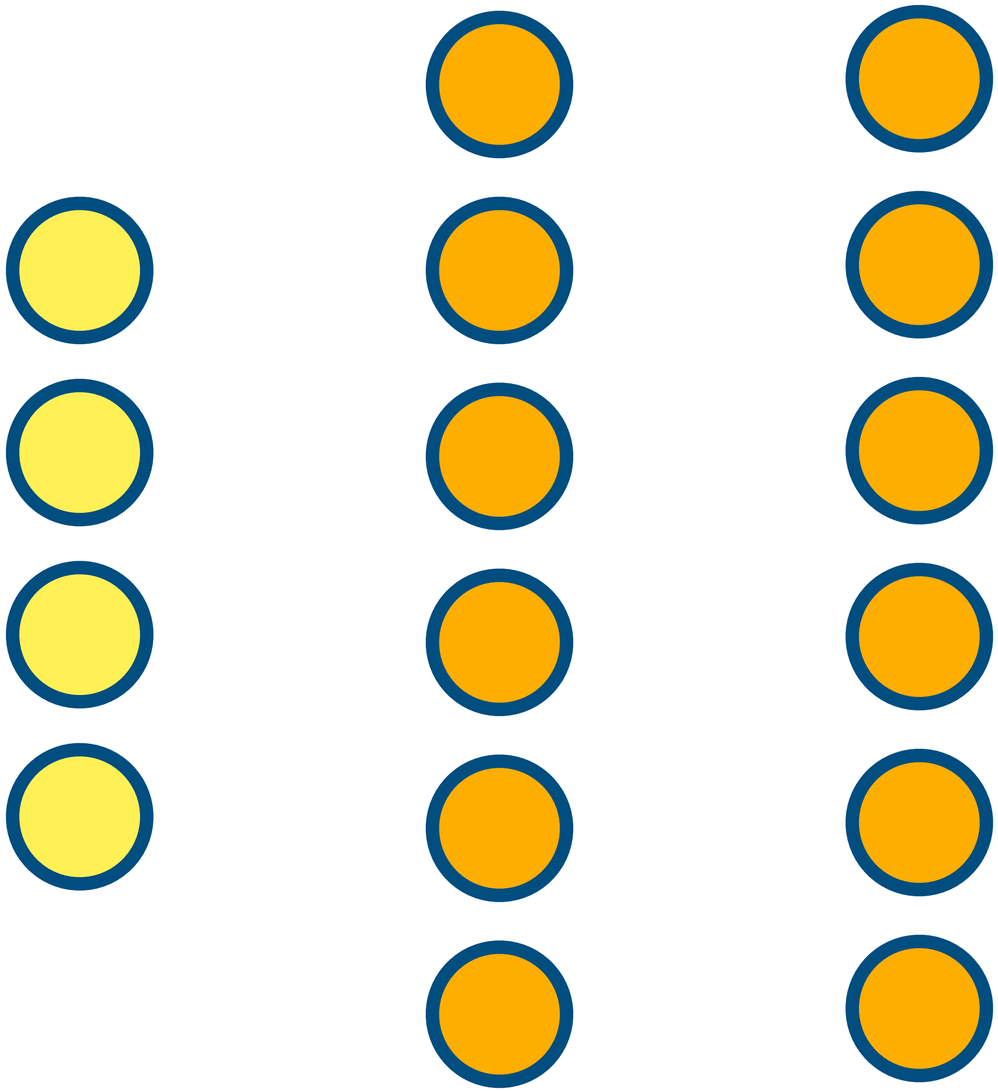
Tensor

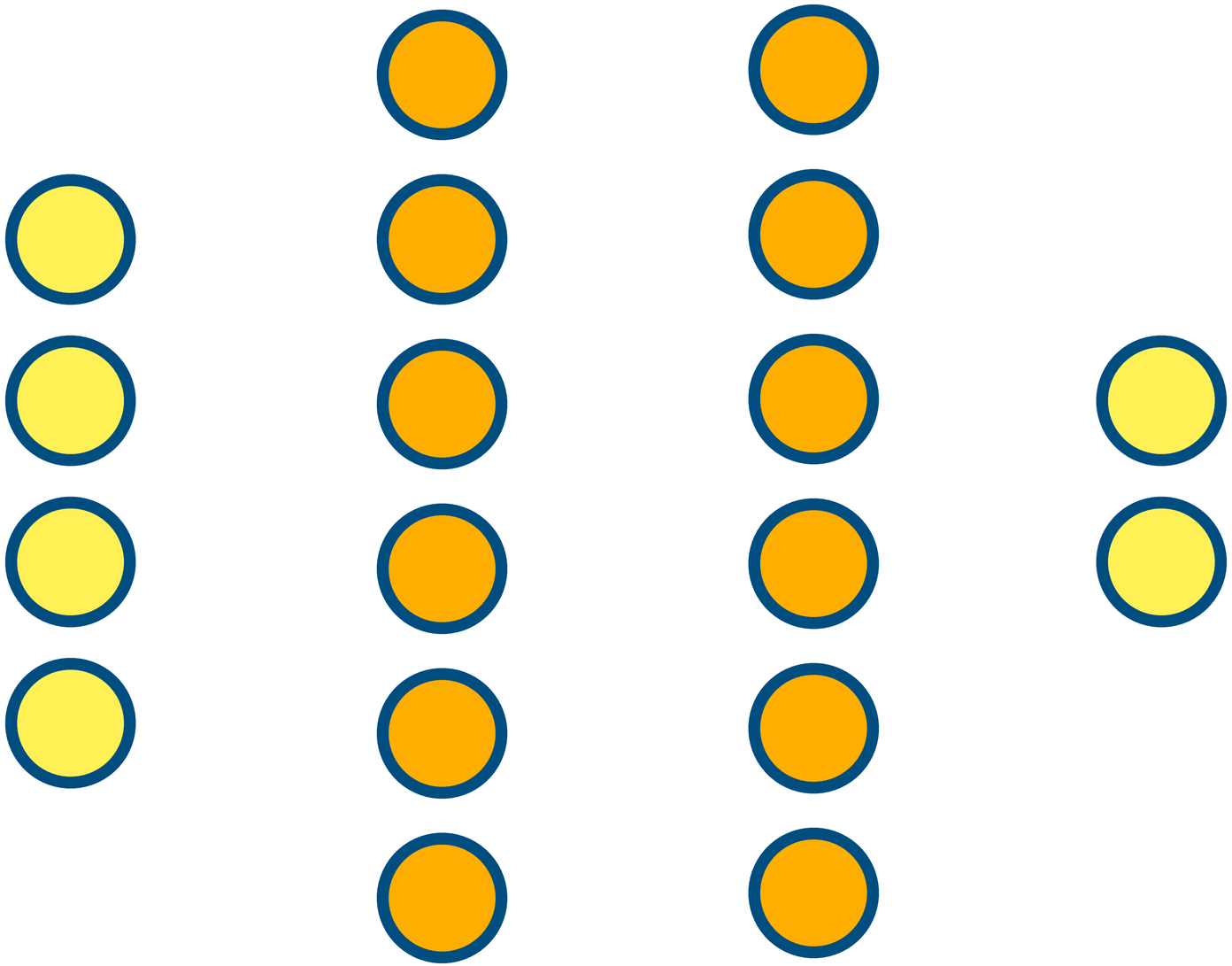


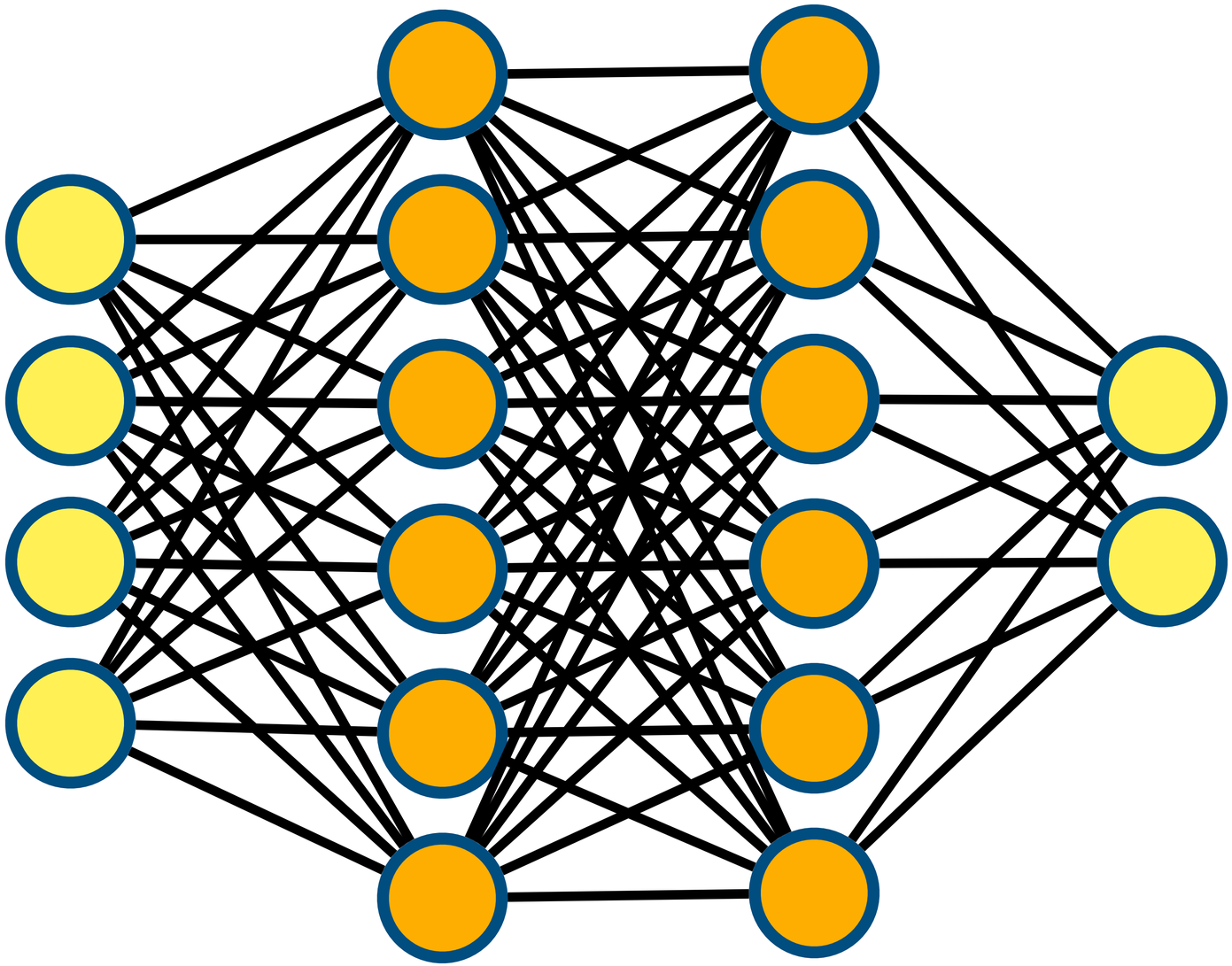
Neural Network



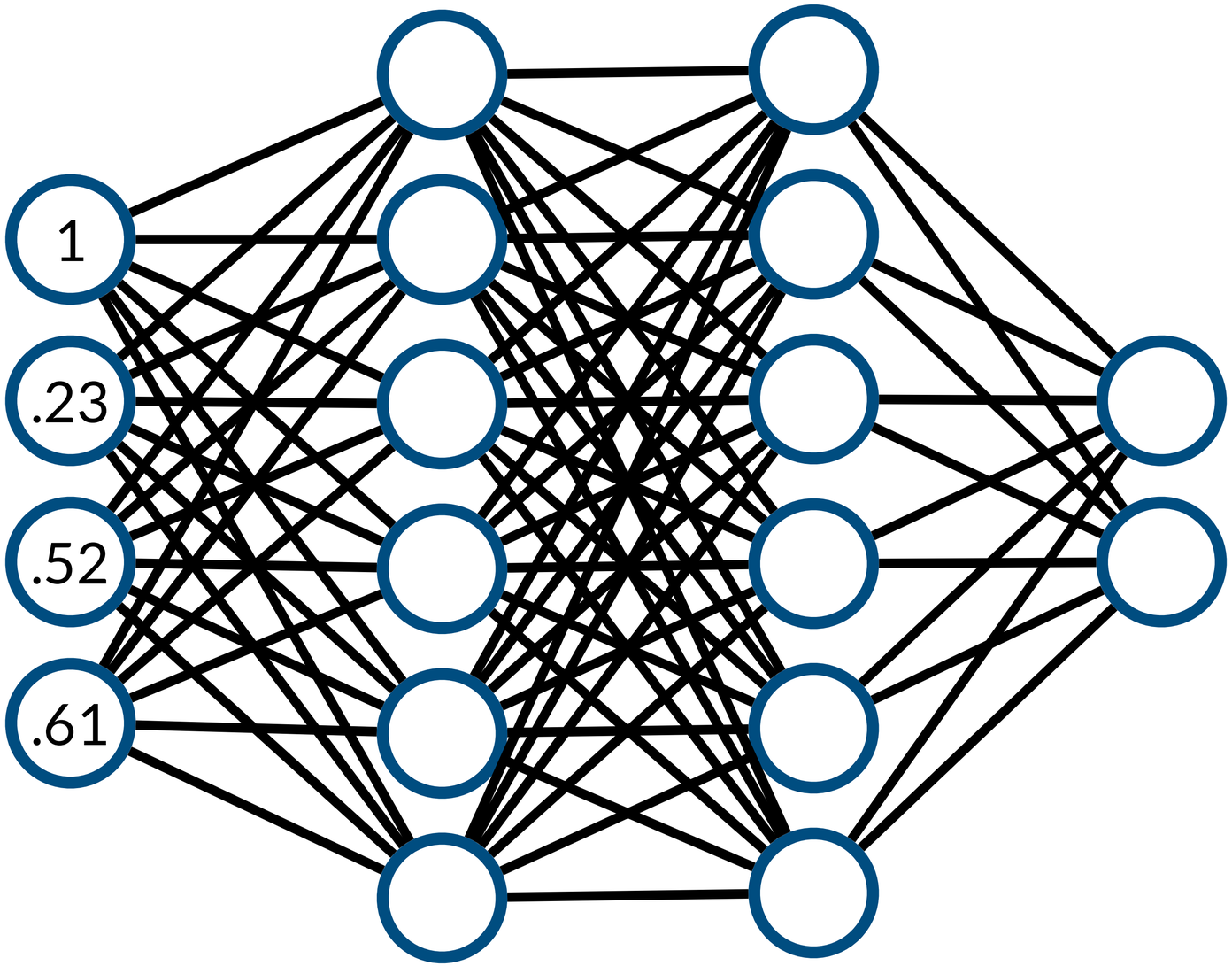


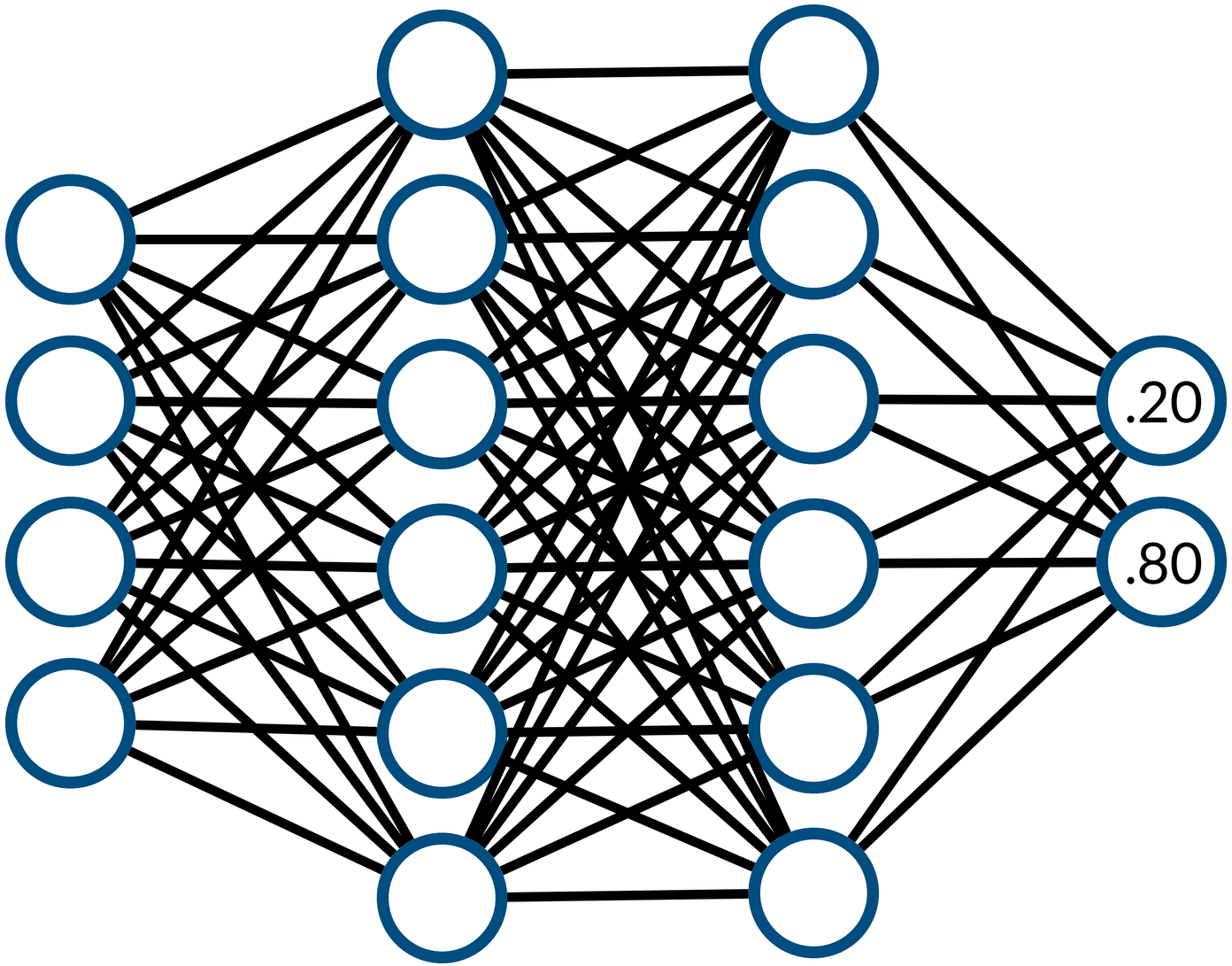


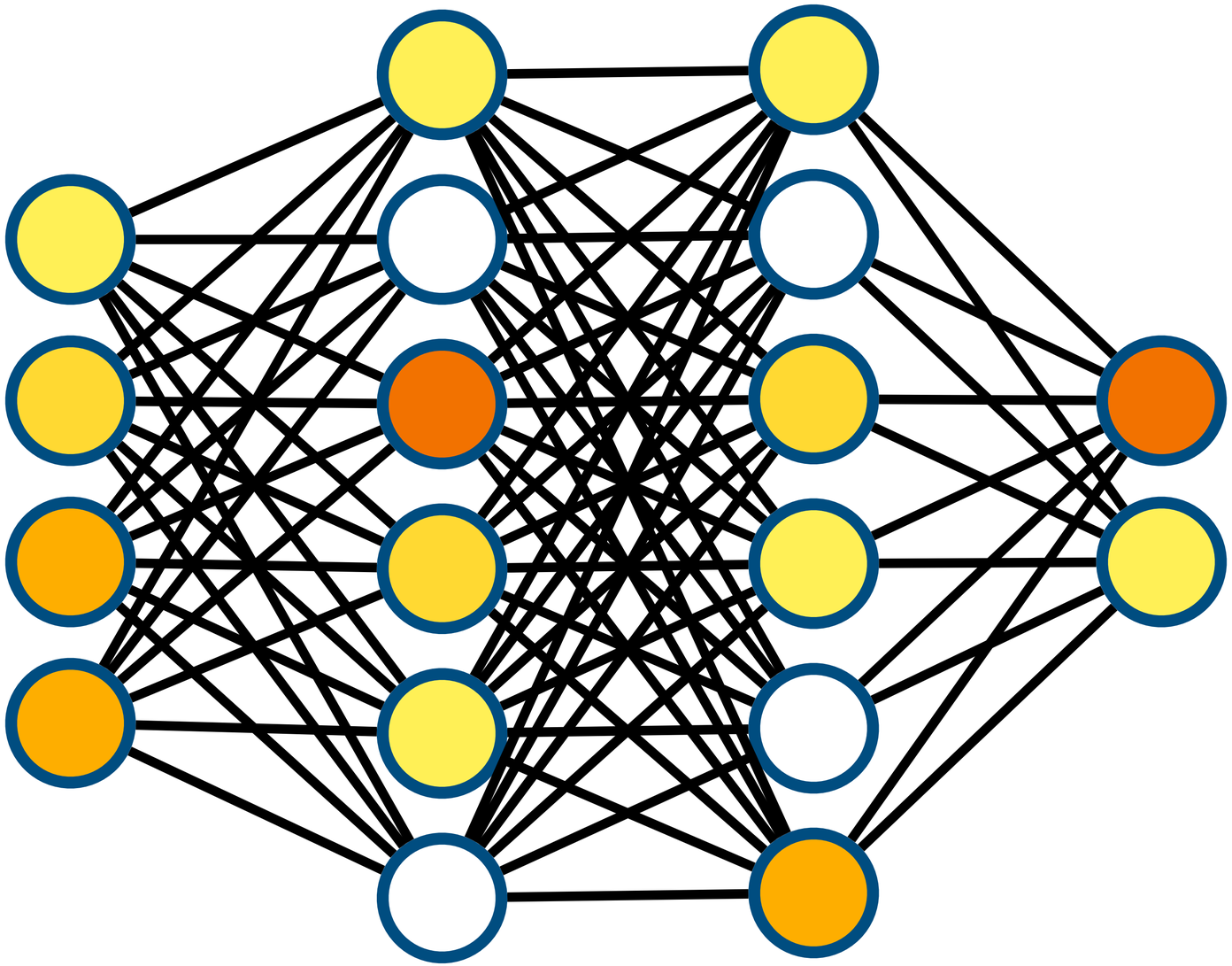




Activation Function


1
.23
.52
.61


.20
.80

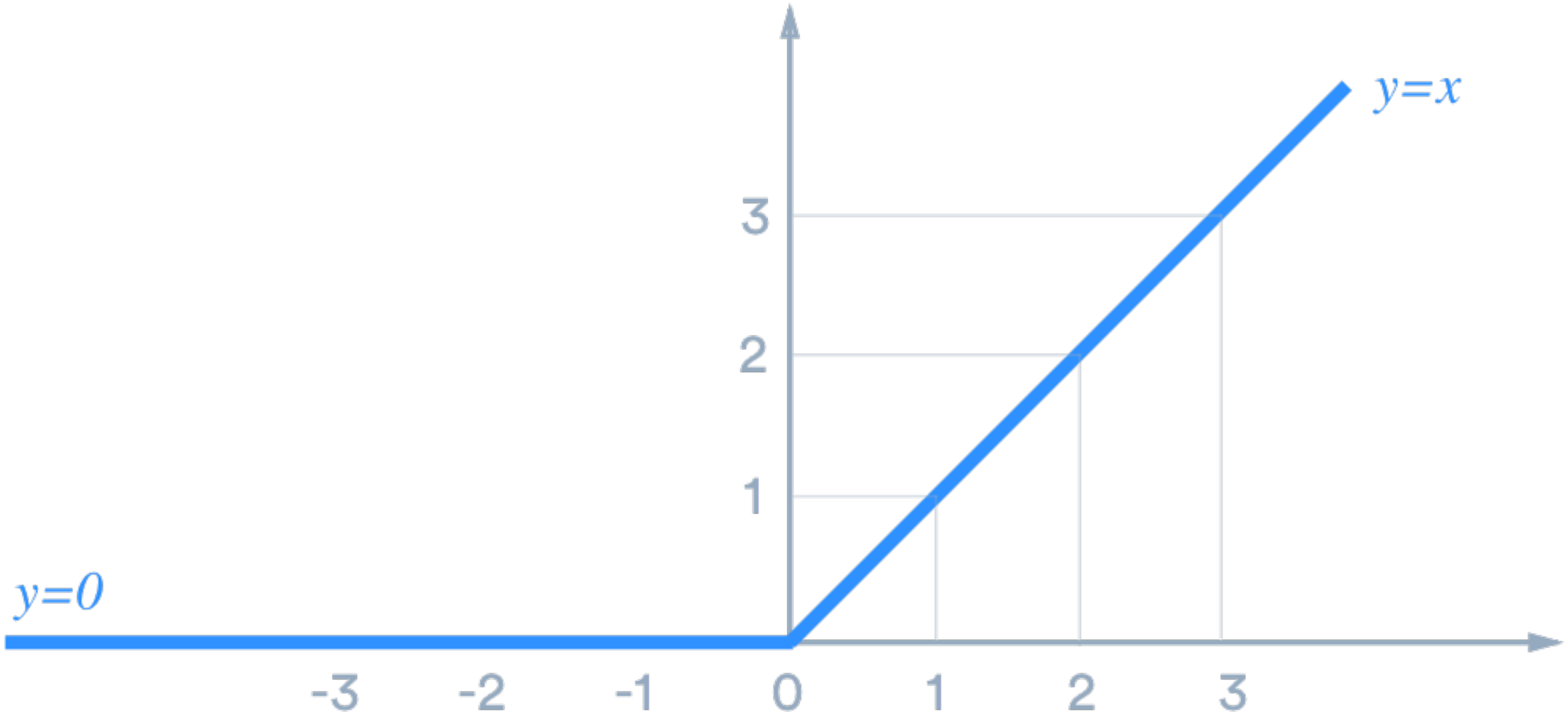


Activation Function

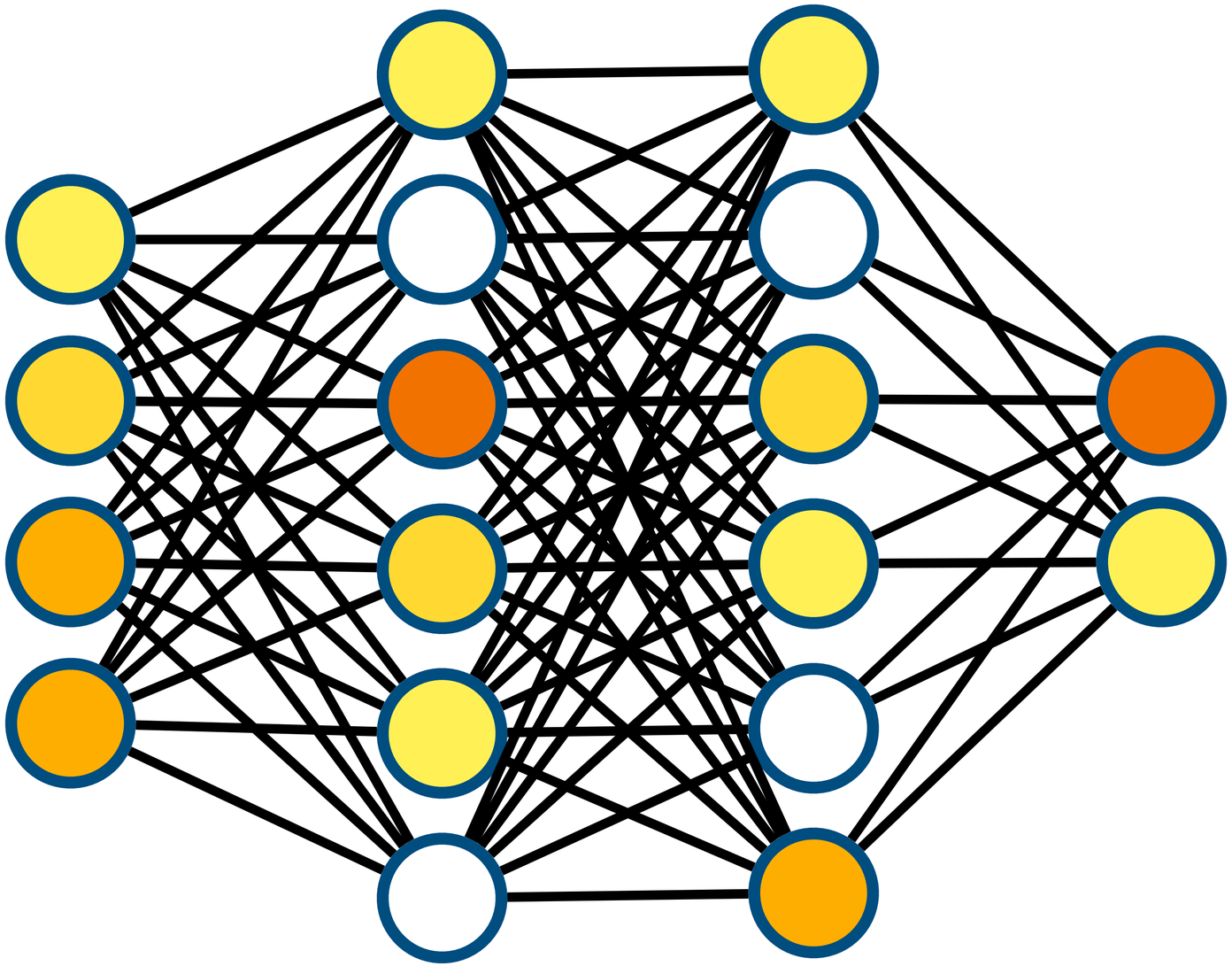
- **Definition** - A non-linear function which decides if the output of a neuron should be propagated or allowed to continue forward
- Without activation functions, neural networks will not be able to solve non-linear problems, without it, it is the same a linear regression

- Functions include:
  - Sigmoid (final - candidate)
  - `tanh`
  - `relu`
  - `selu`
  - leaky `relu`
  - `softmax` (final)

Rectified Linear Unit



https://medium.com/@sonish.sivarajkumar/relu-most-popular-activation-function-for-deep-neural-networks-10160af37dda



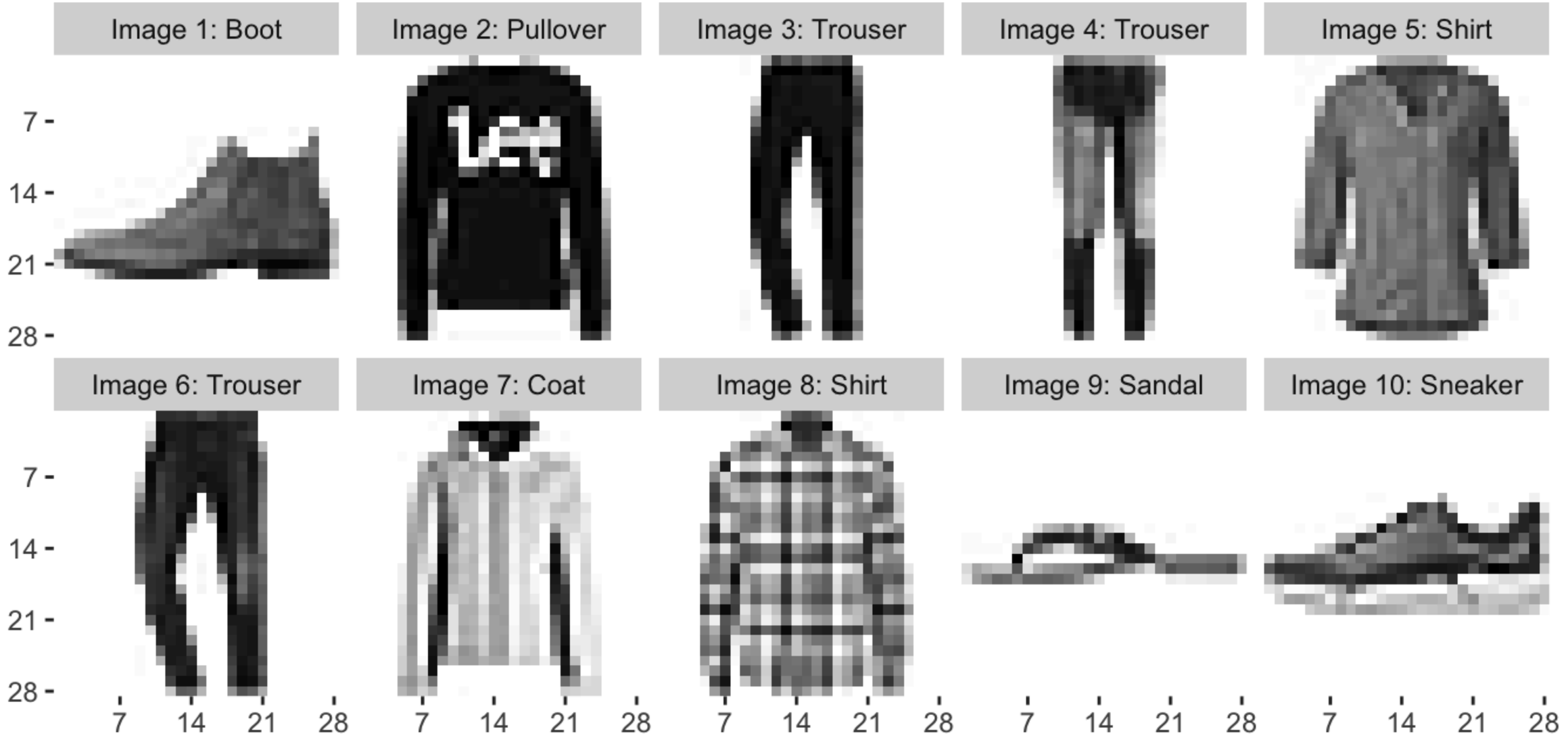


Image 1: Boot
Image 2: Pullover
Image 3: Trouser
Image 4: Trouser
Image 5: Shirt
Image 6: Trouser
Image 7: Coat
Image 8: Shirt
Image 9: Sandal
Image 10: Sneaker
7
14
21
28

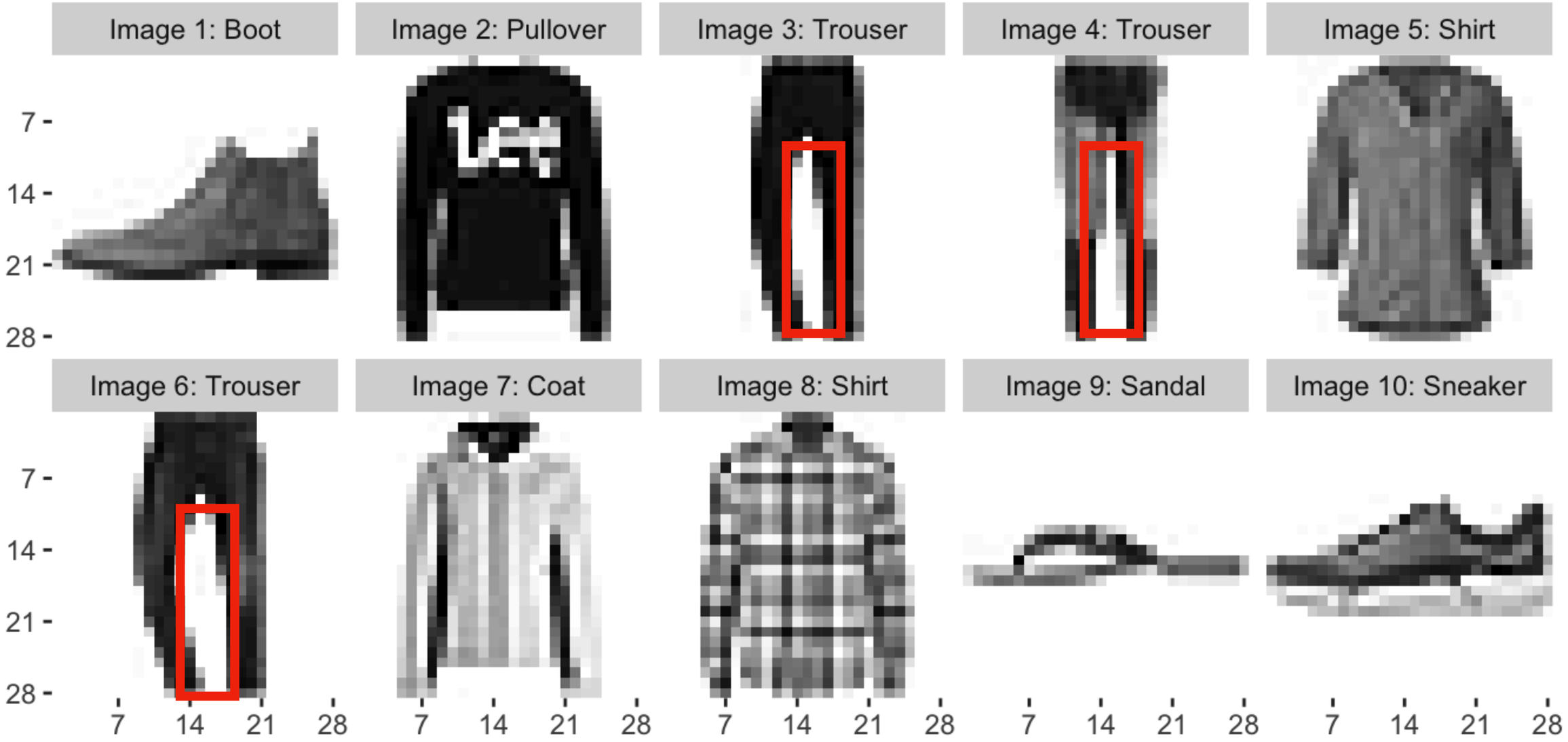
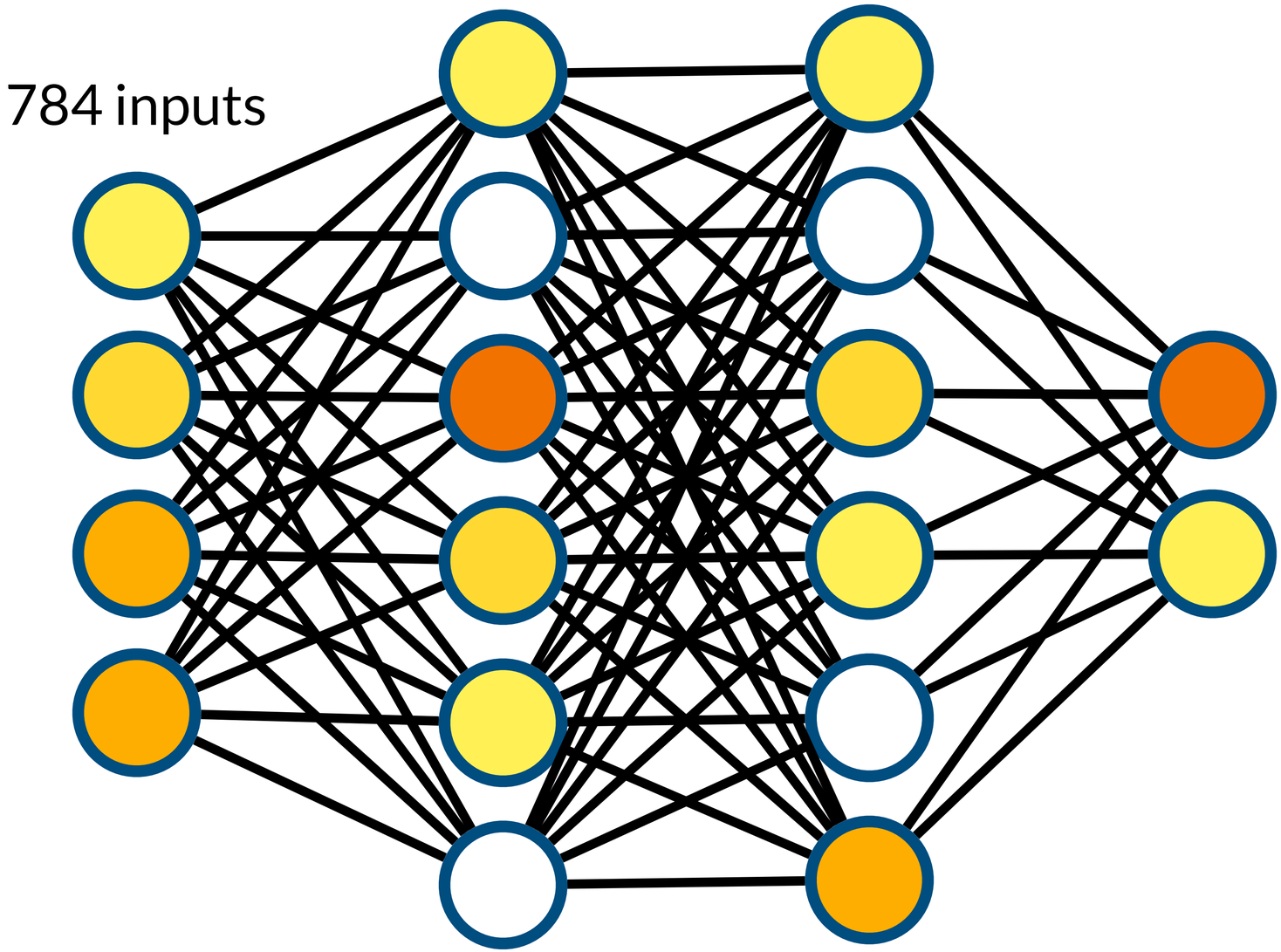
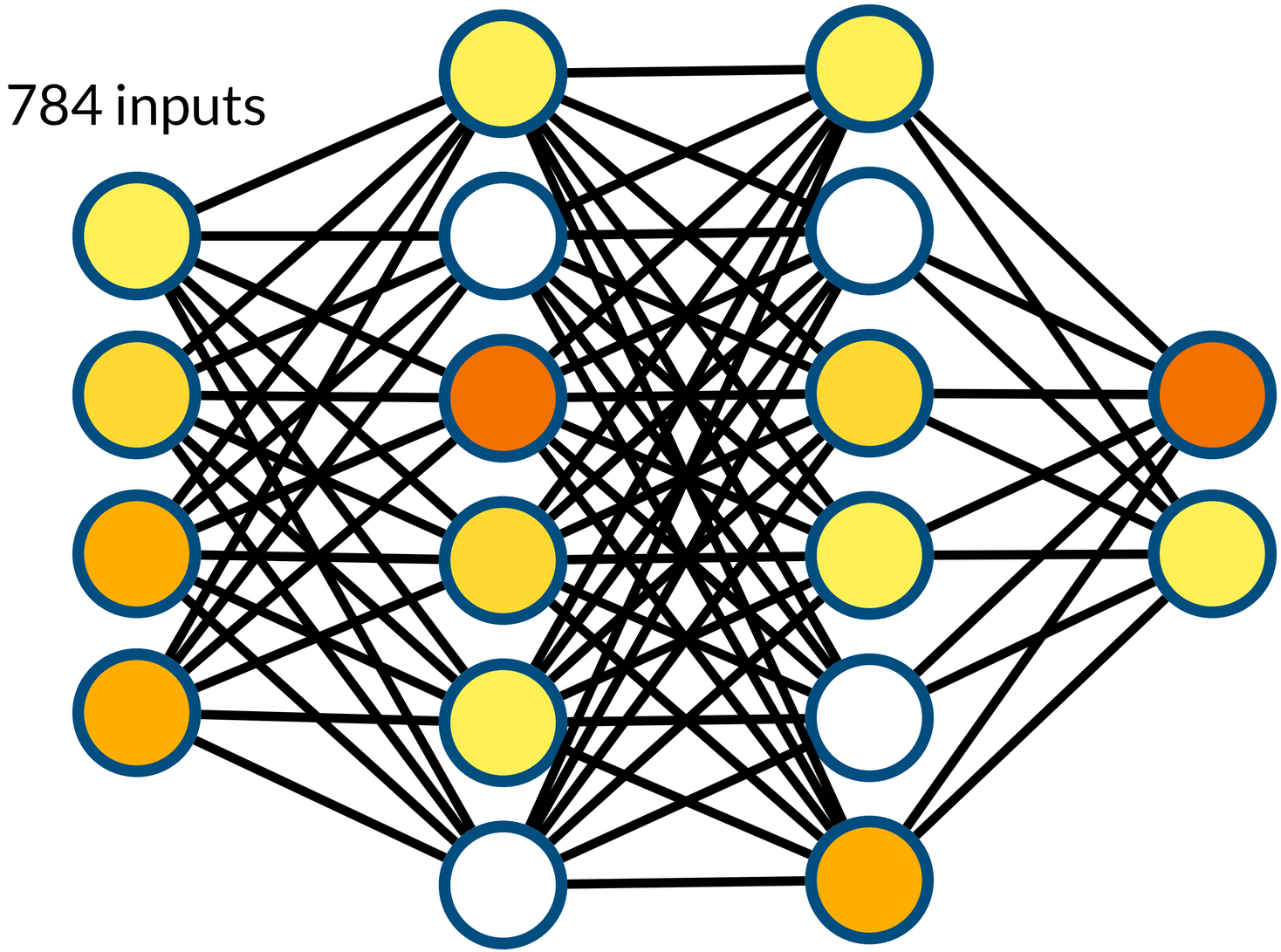
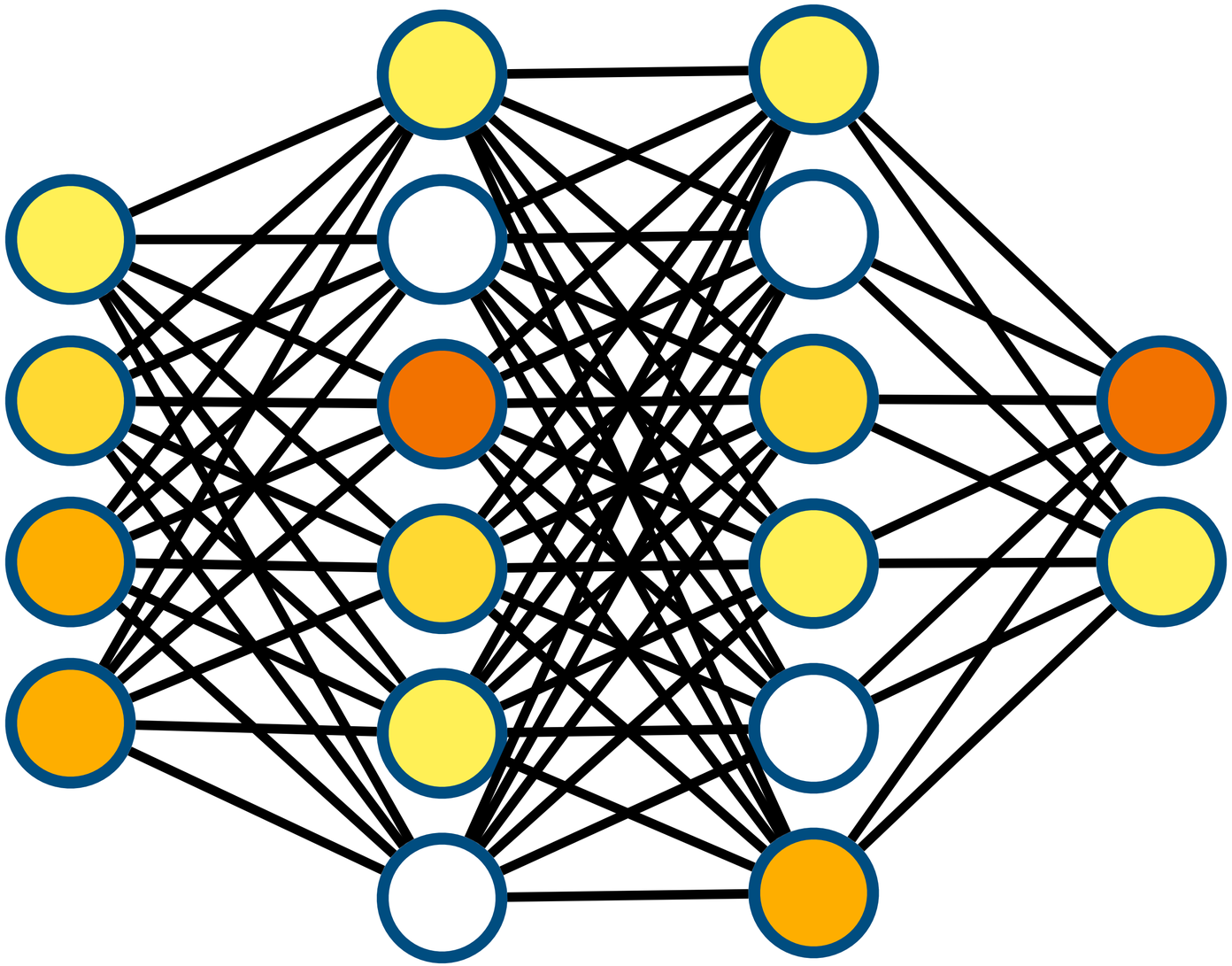
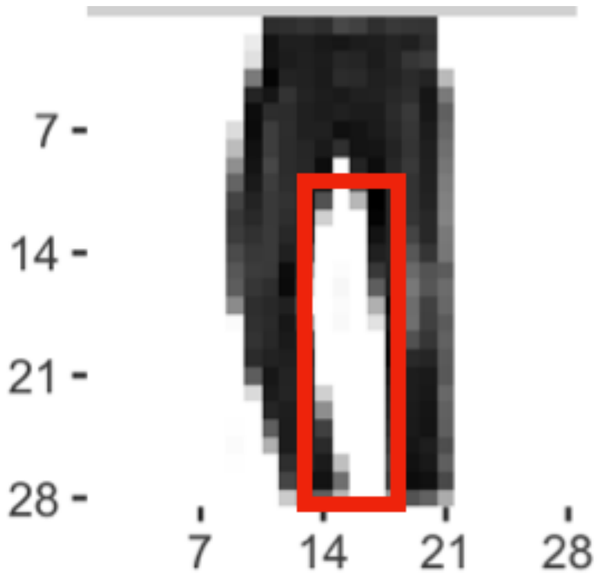


Image 1: Boot
Image 2: Pullover
Image 3: Trouser
Image 4: Trouser
Image 5: Shirt
Image 6: Trouser
Image 7: Coat
Image 8: Shirt
Image 9: Sandal
Image 10: Sneaker
7
14
21
28


784 inputs


784 inputs

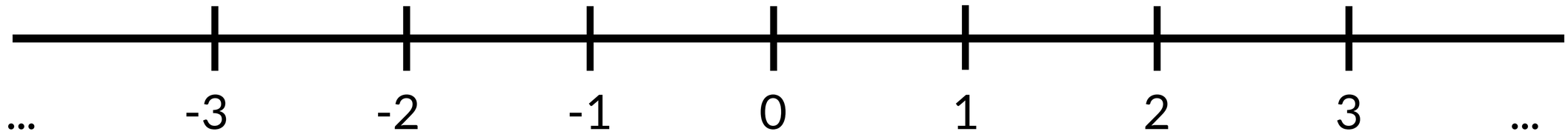




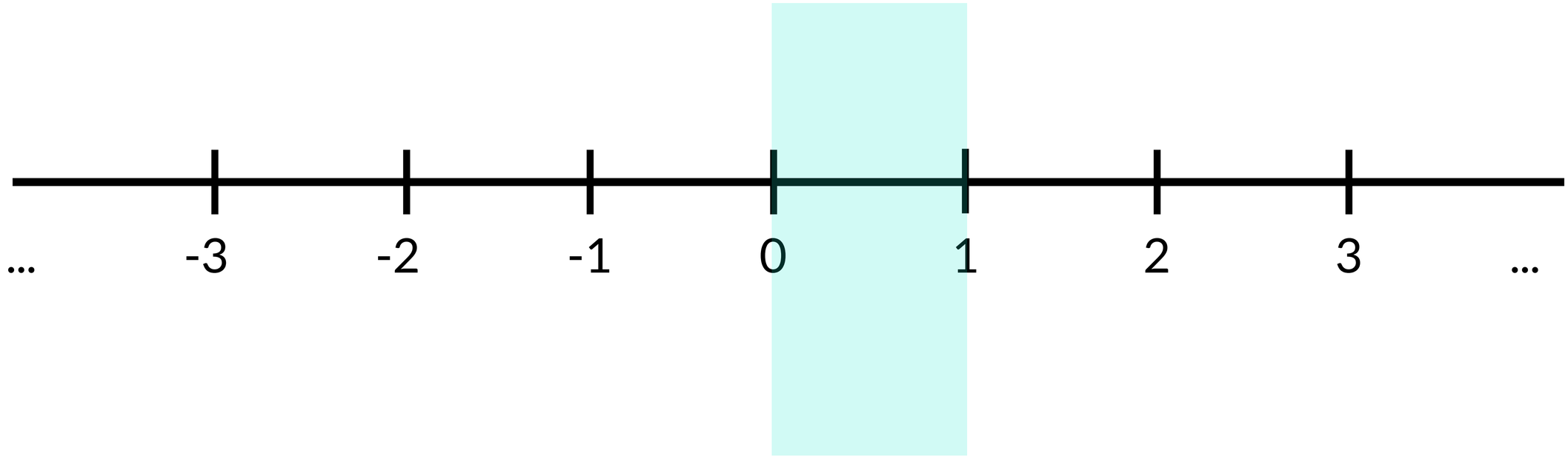
values = $(a1, a2, a3 \ldots a784)$

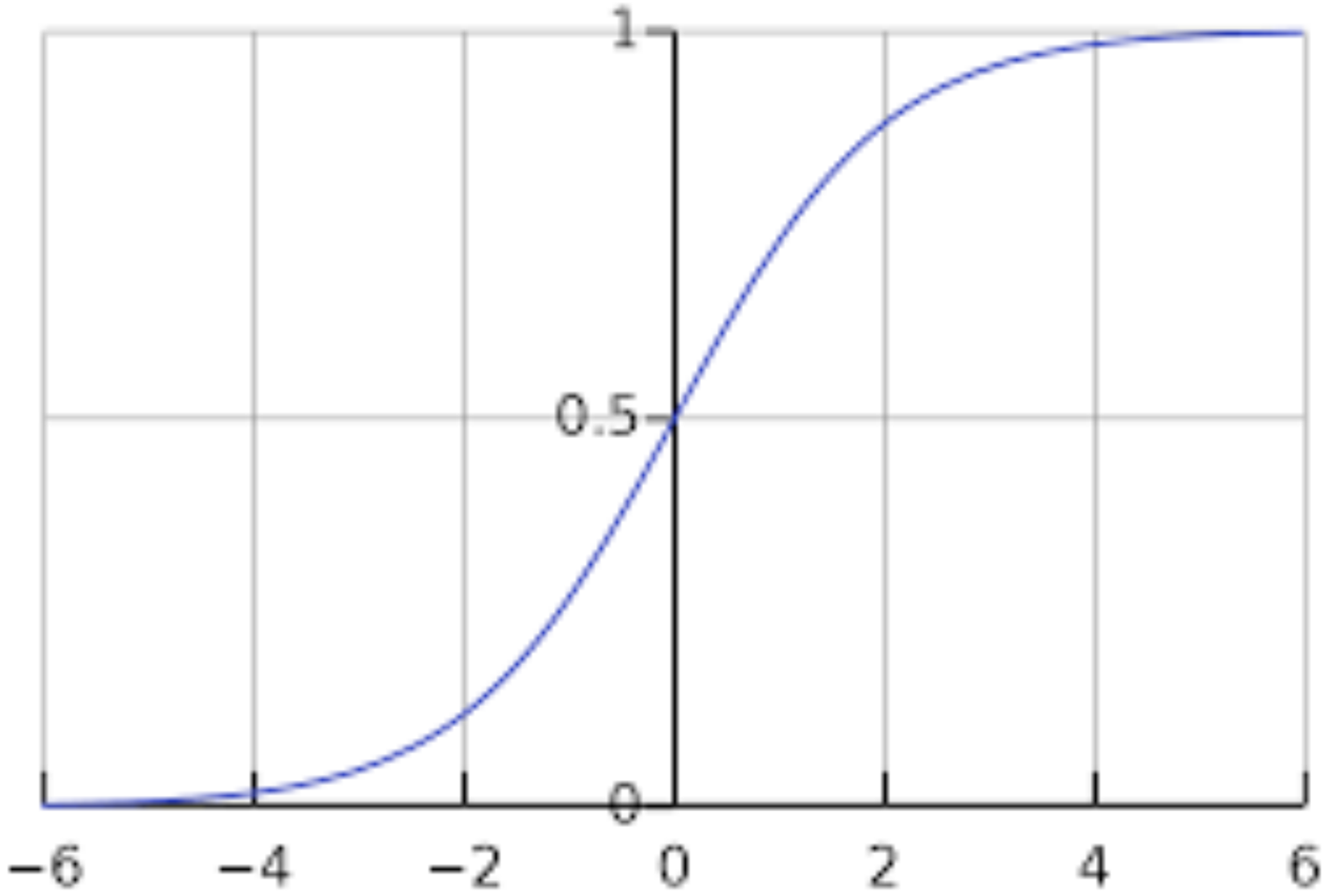
weighted values = $(w1 * a1 + w2 * a2, \ldots, w784 * a784)$

weighted values = $(w1 * a1 + w2 * a2, \ldots, w784 * a784)$



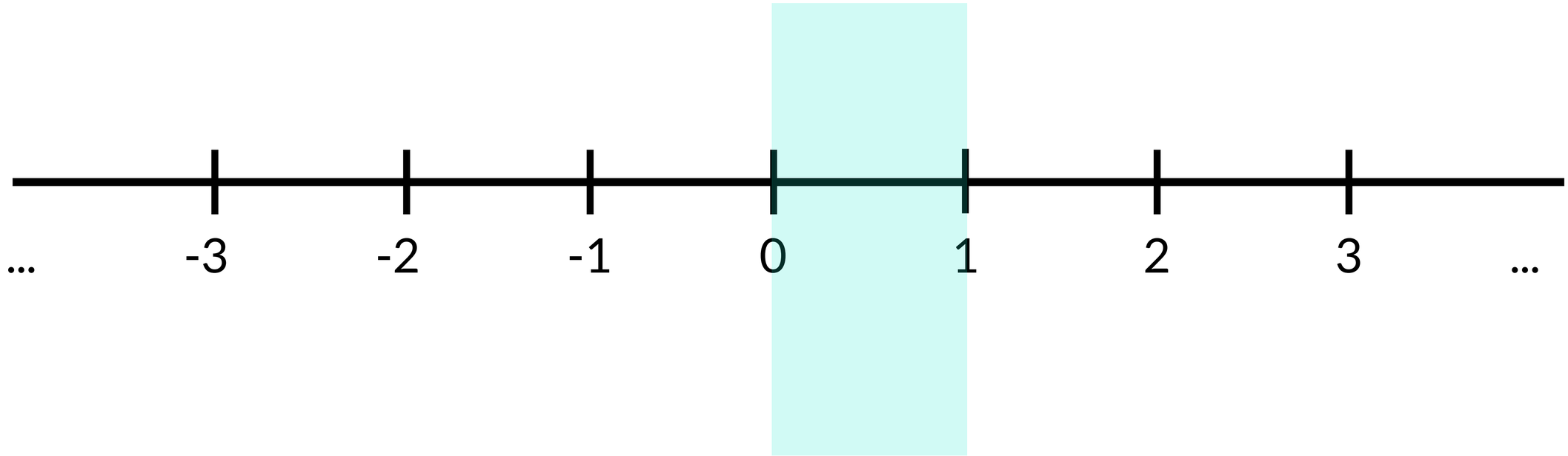
weighted values = $(w1 * a1 + w2 * a2, \ldots, w784 * a784)$



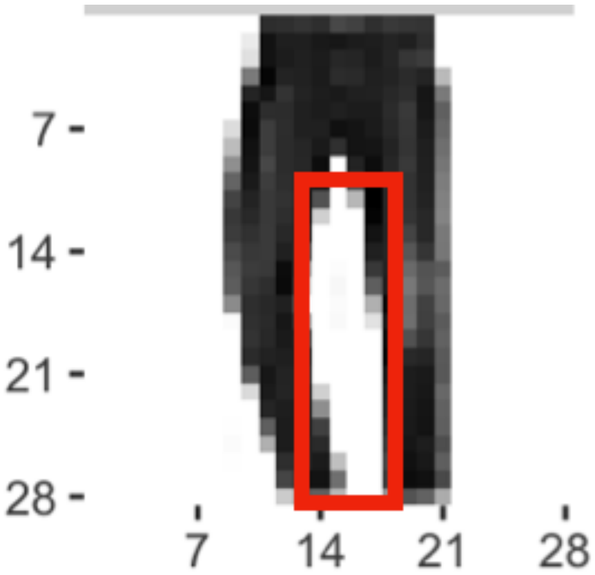


Sigmoid Function

weighted values = $\sigma(w1 * a1 + w2 * a2, \ldots, w784 * a784)$



If I want to make a bit harder to activate



values = $(a1, a2, a3 \ldots a784)$

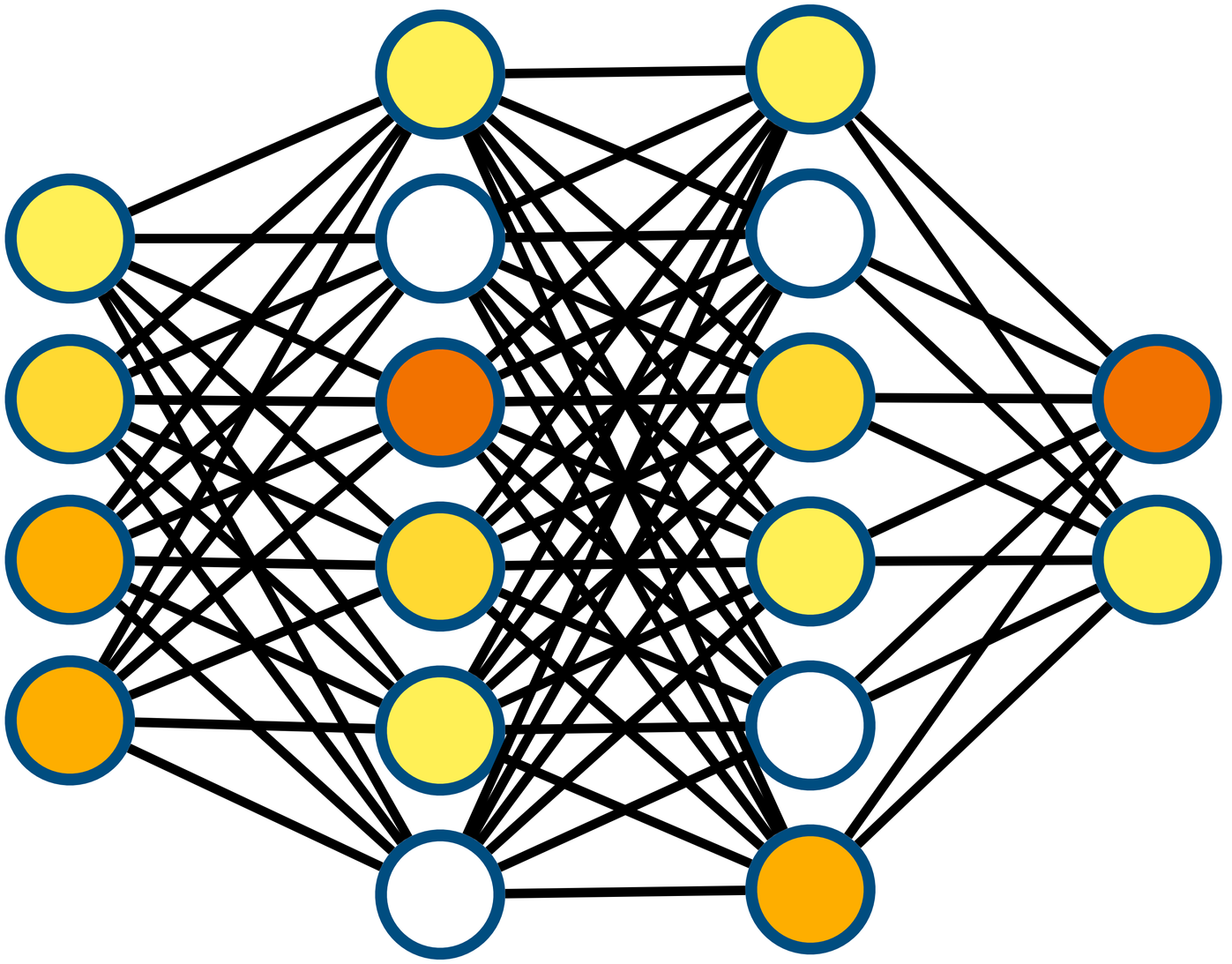
weighted values = $(w1 * a1 + w2 * a2, \ldots, w784 * a784 \mathbf{- bias})$

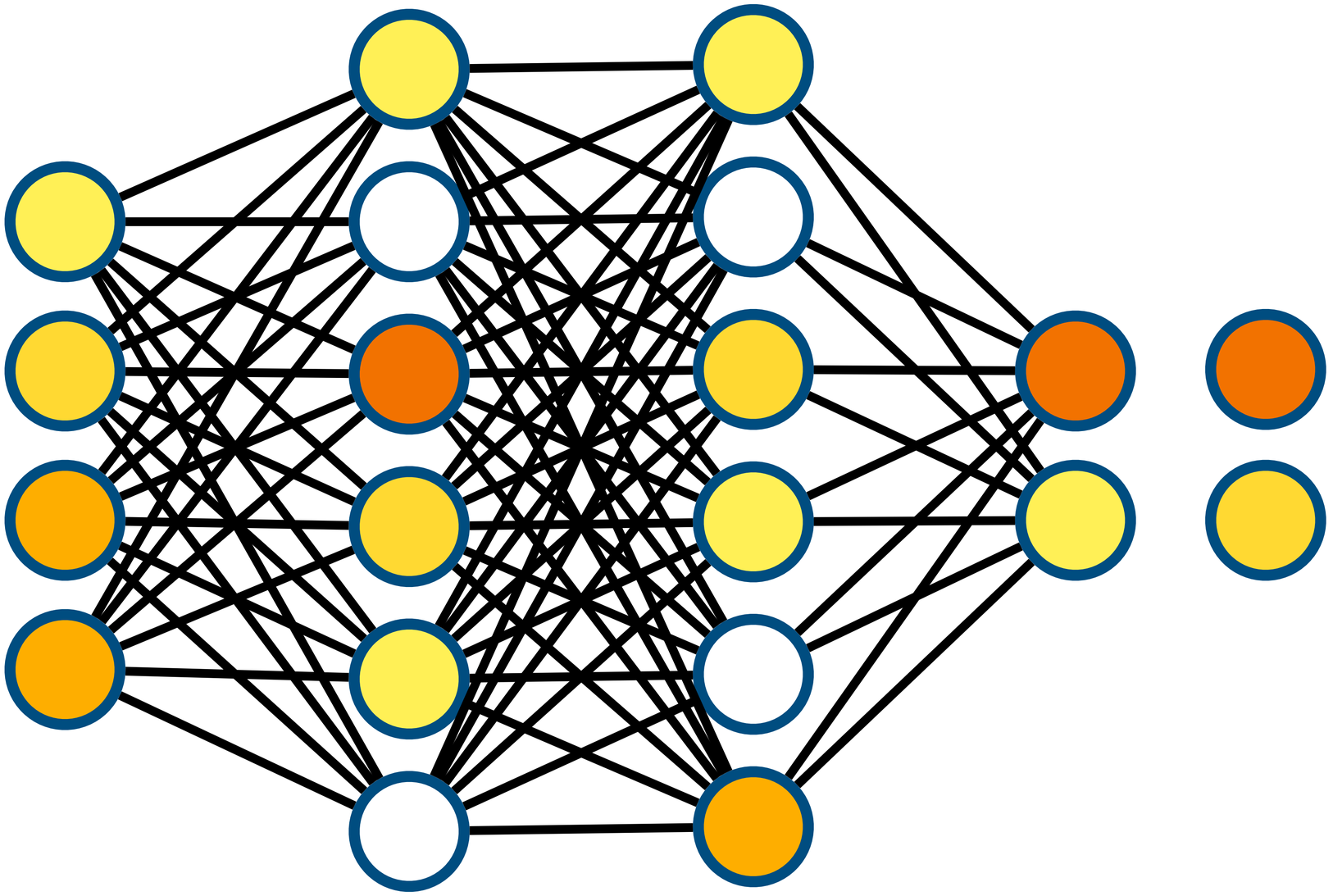
weighted between 0 and 1 = $\sigma(w1 * a1 + w2 * a2, \ldots, w784 * a784 \mathbf{- bias})$

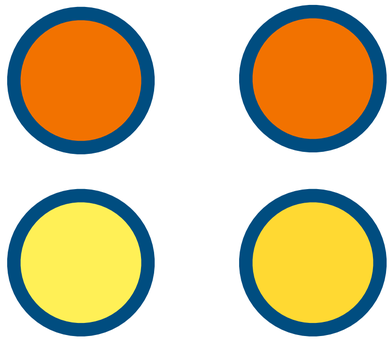
$$\sigma(w1 * a1 + w2 * a2, ..., w784 * a784 - \mathbf{bias})$$

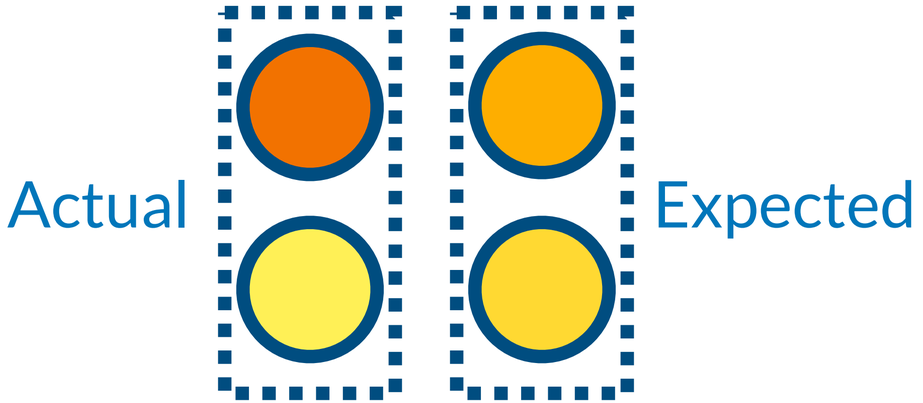
$$\sigma(\mathbf{W} + \mathbf{b})$$

Cost Function



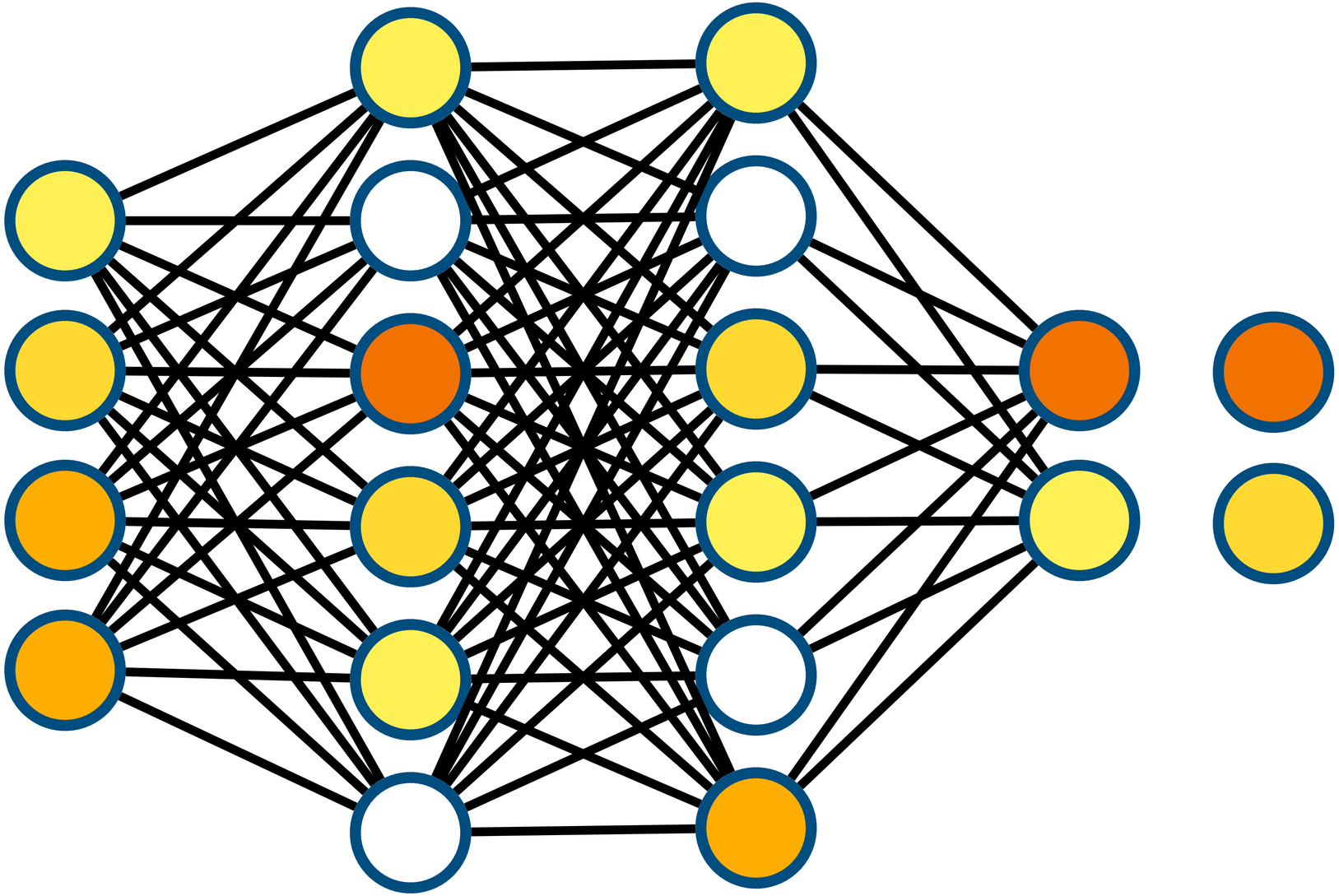




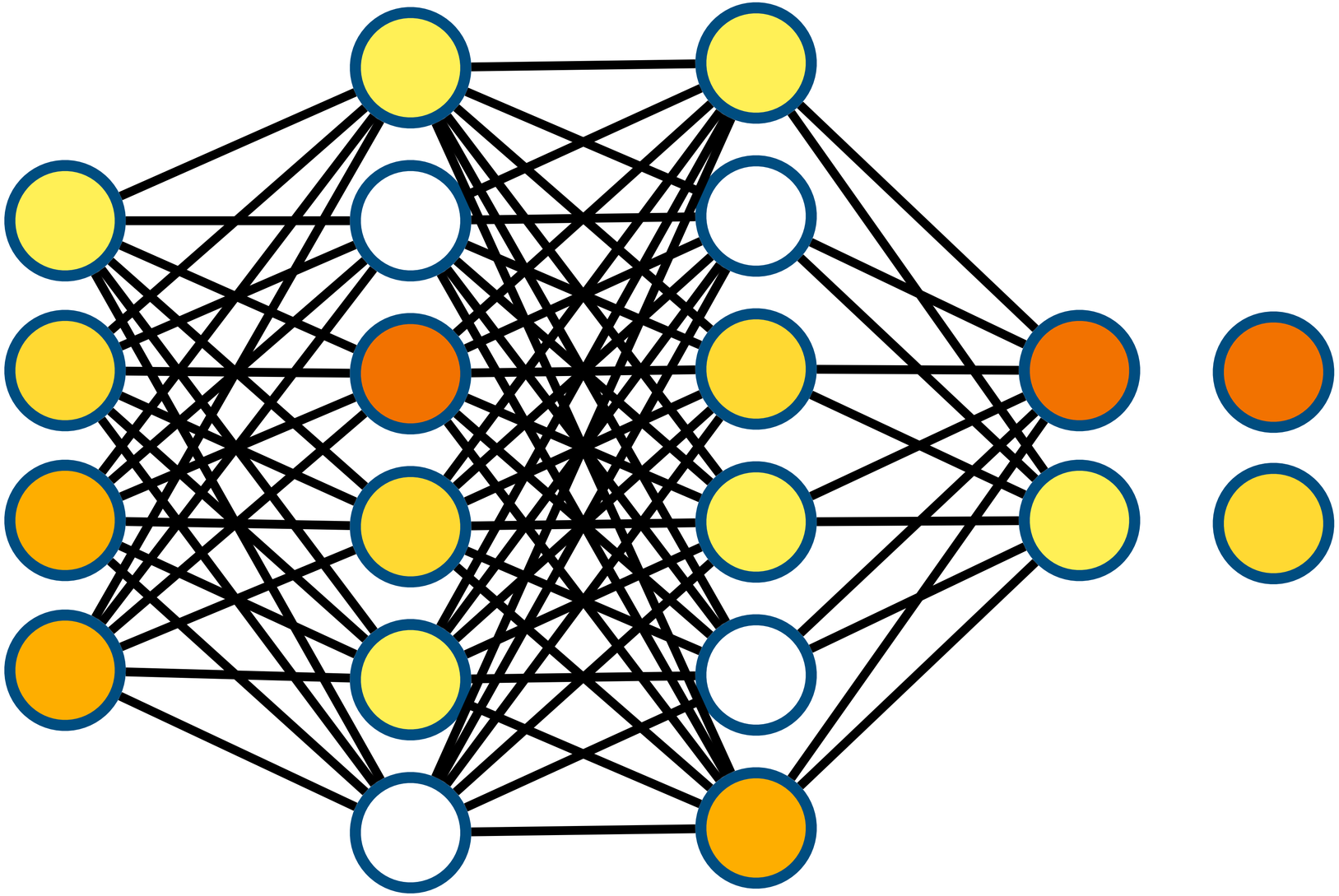


The "cost" is what is the difference: (actual - expected)^2

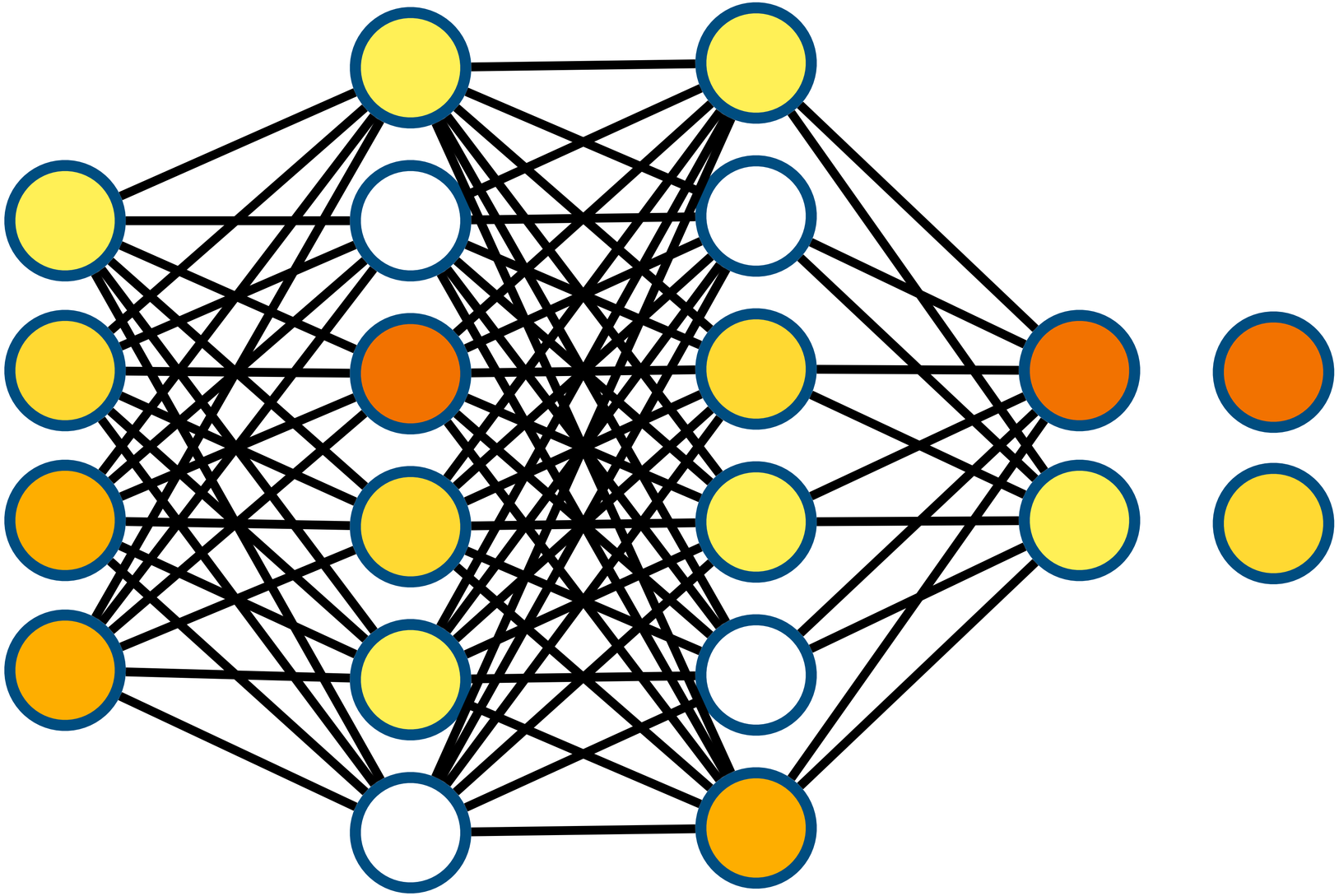
Backpropagation



Goal: How can we minimize the cost function?



Go back and "tweak" each of the neurons and measure the response

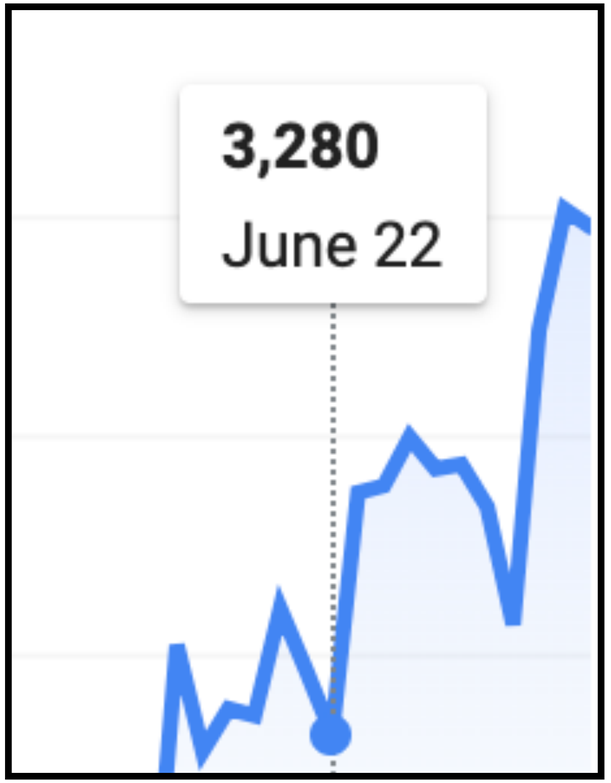


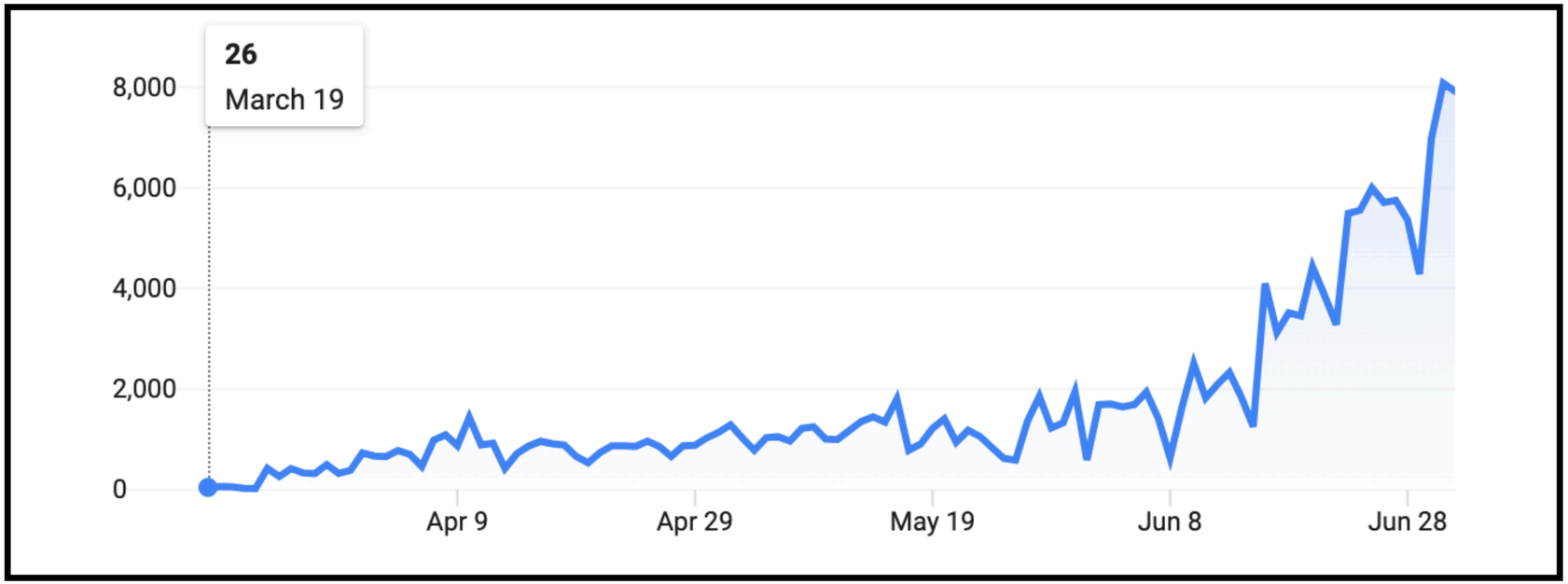
The Neural Network will determine which is more sensitive with small adjustments

Gradient Descent

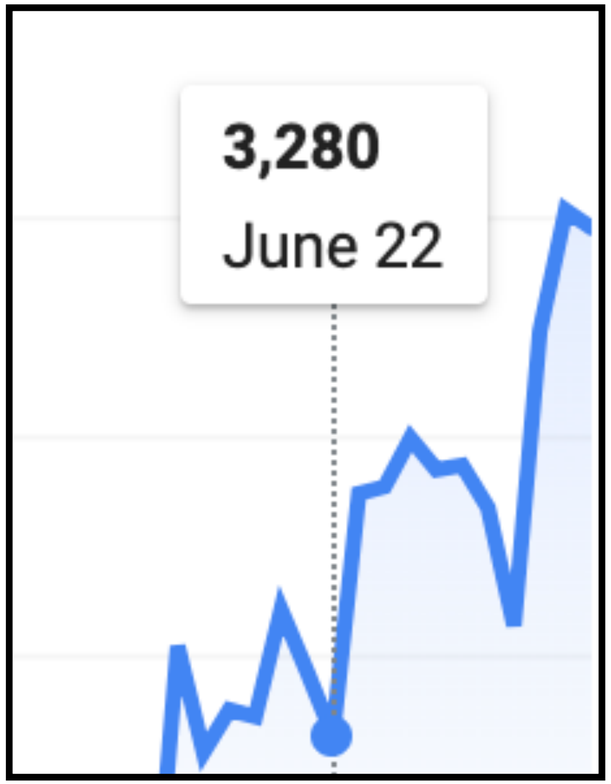
Local Minima/Global Minima

Local Minima vs Global Minima





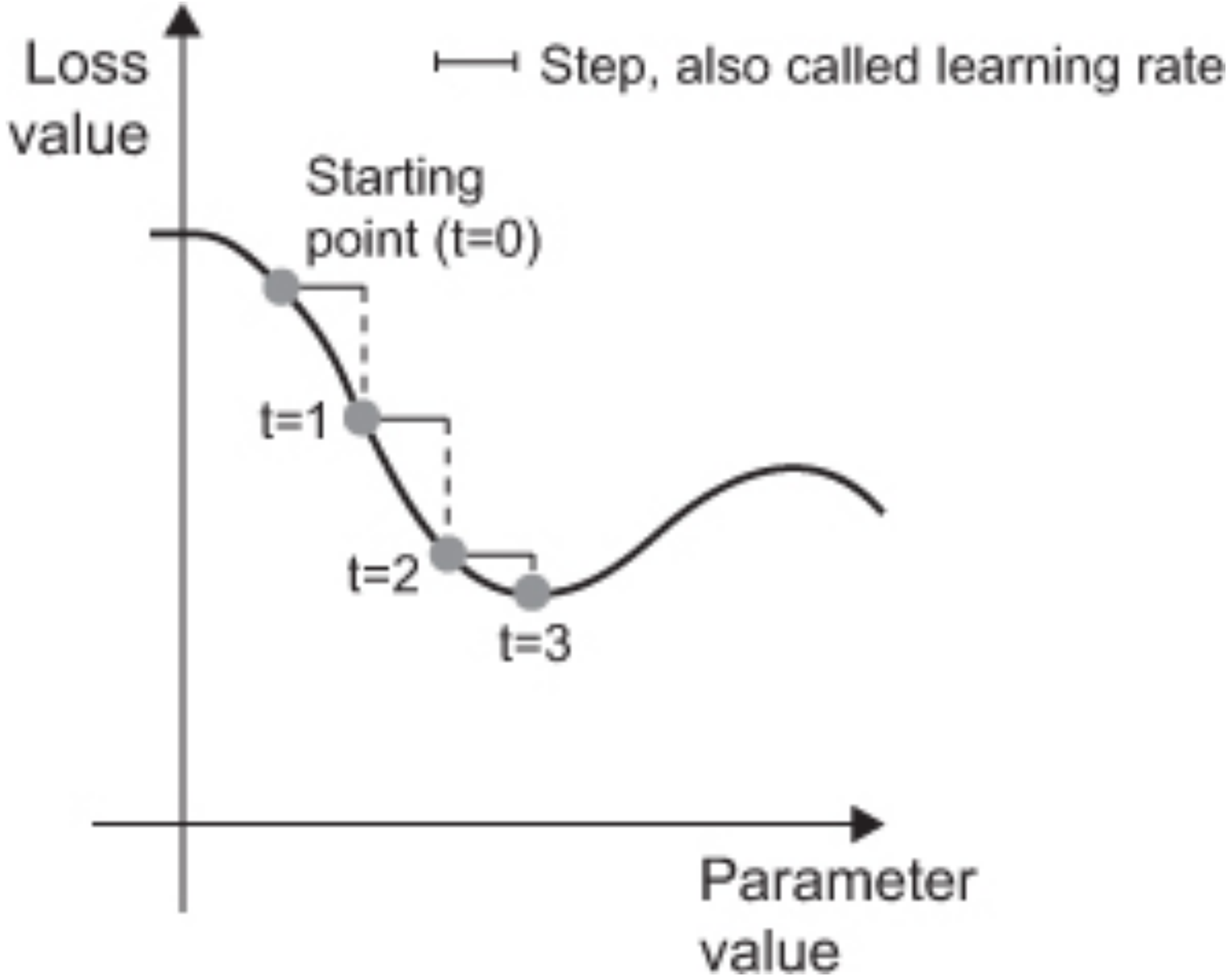
Local Minima vs Global Minima



Definition - We pass the same dataset multiple times to the network in order to find optimal weights; one training session.

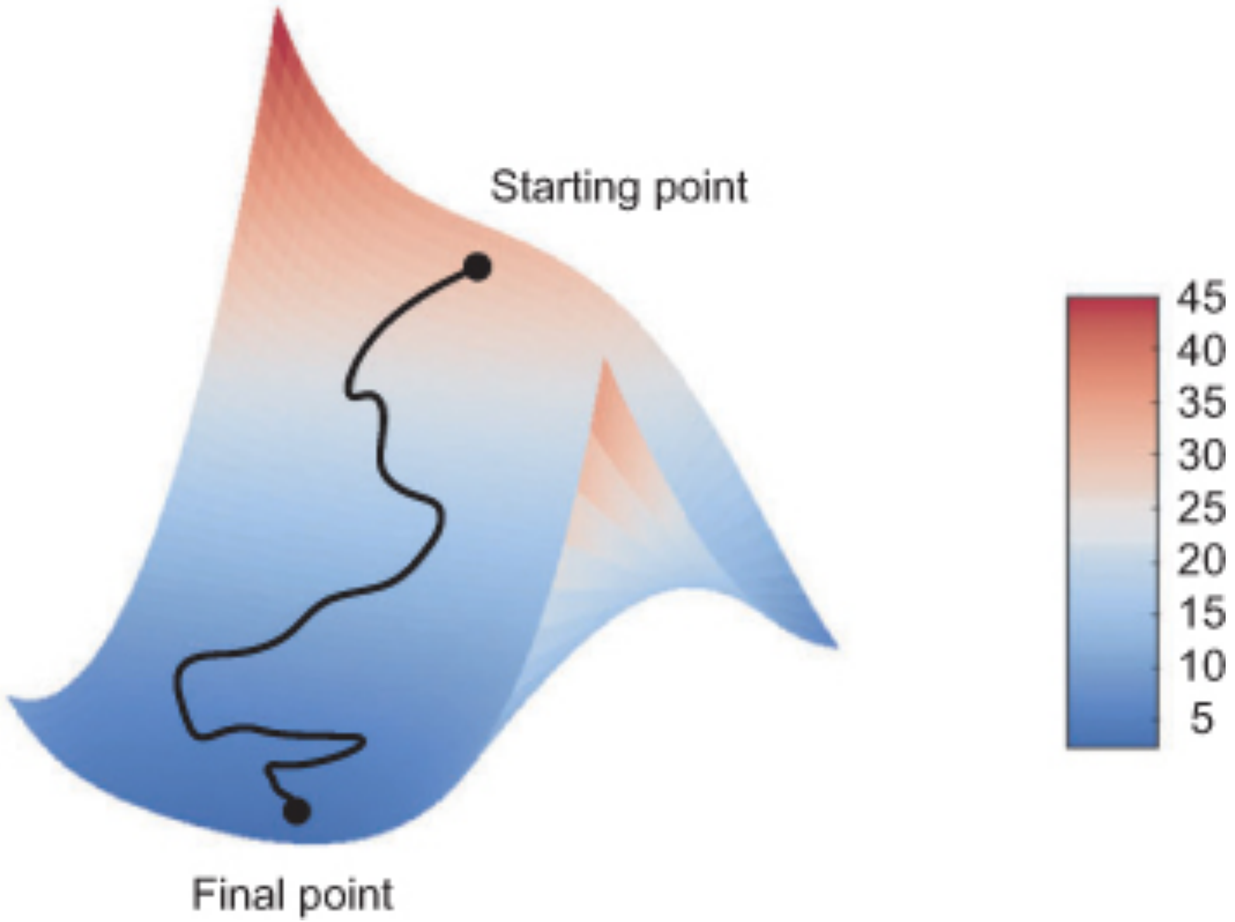
The More the Merrier - with just a few epochs you can reach only a local minimum, but with more epochs, you can reach a global minimum or at least a better local minimum

Back to Gradient Descent



Each Epoch we are learning and trying to lower our cost function

Deep Learning with Python by Francois Chollet



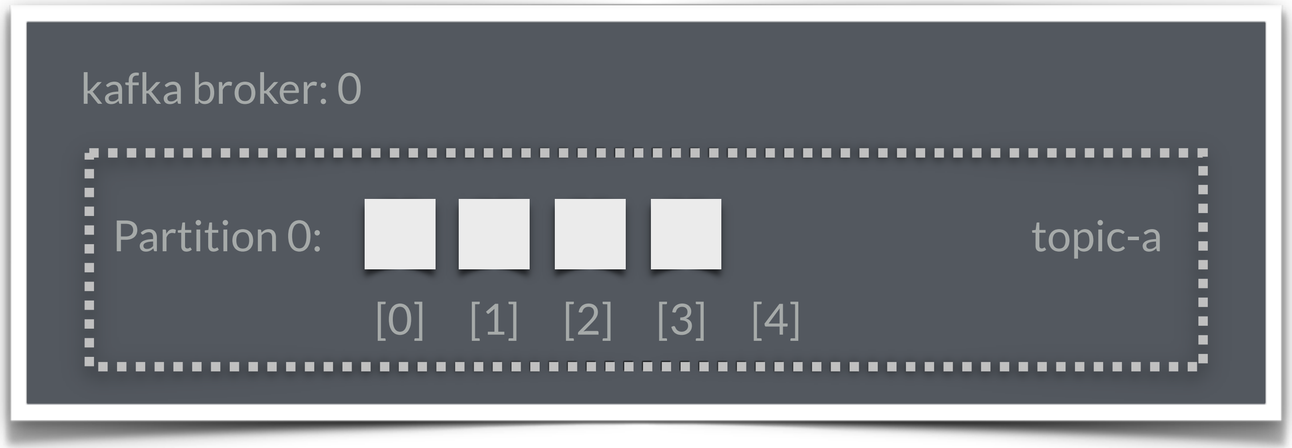
2d gradient descent, finding the lowest cost function

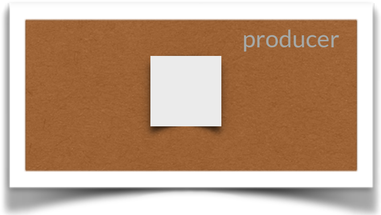
Deep Learning with Python by Francois Chollet

Lab: TensorFlow

Kafka & Kafka Streams

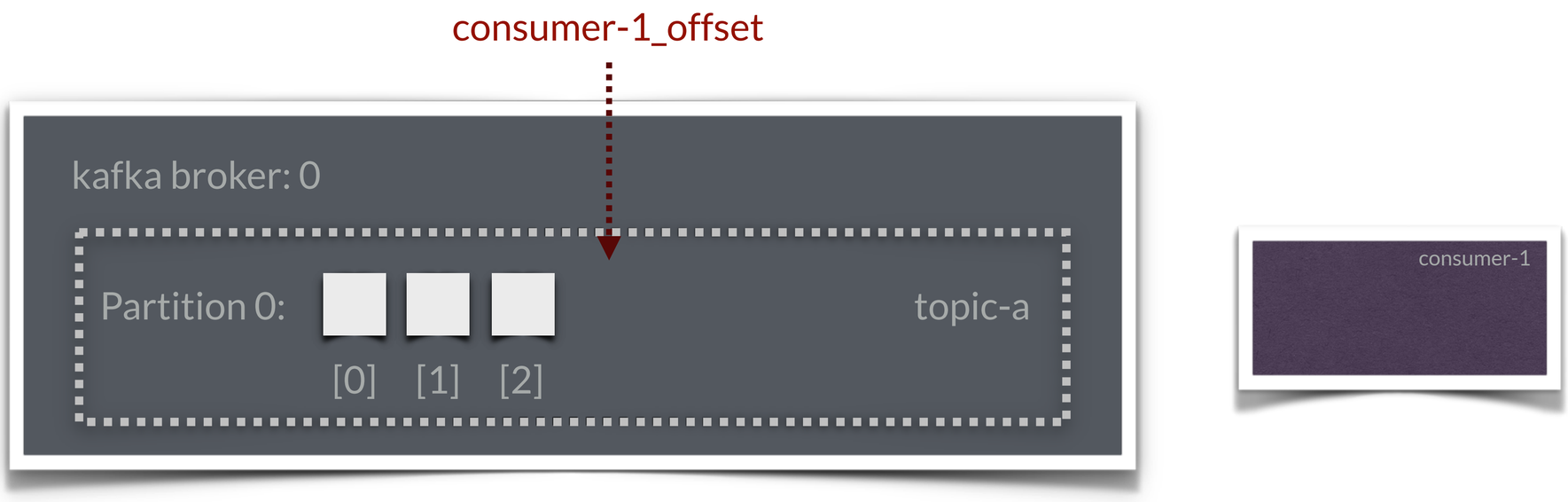
Kafka Producers

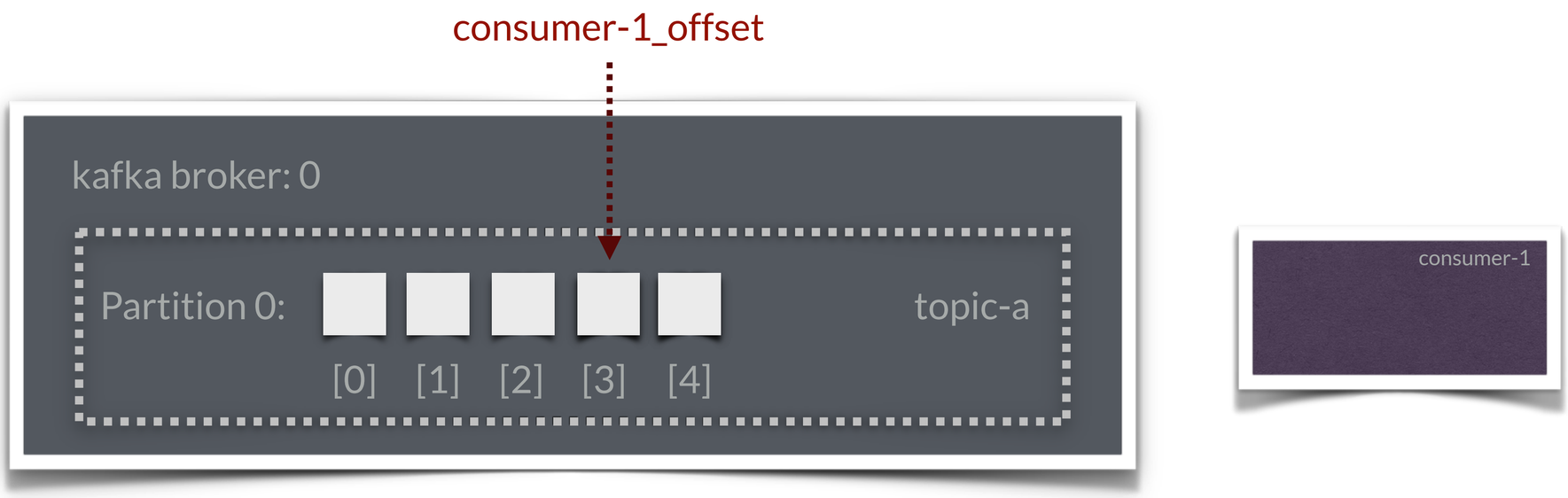

kafka broker: 0
Partition 0:
topic-a
[0]
[1]
[2]
[3]
[4]


producer

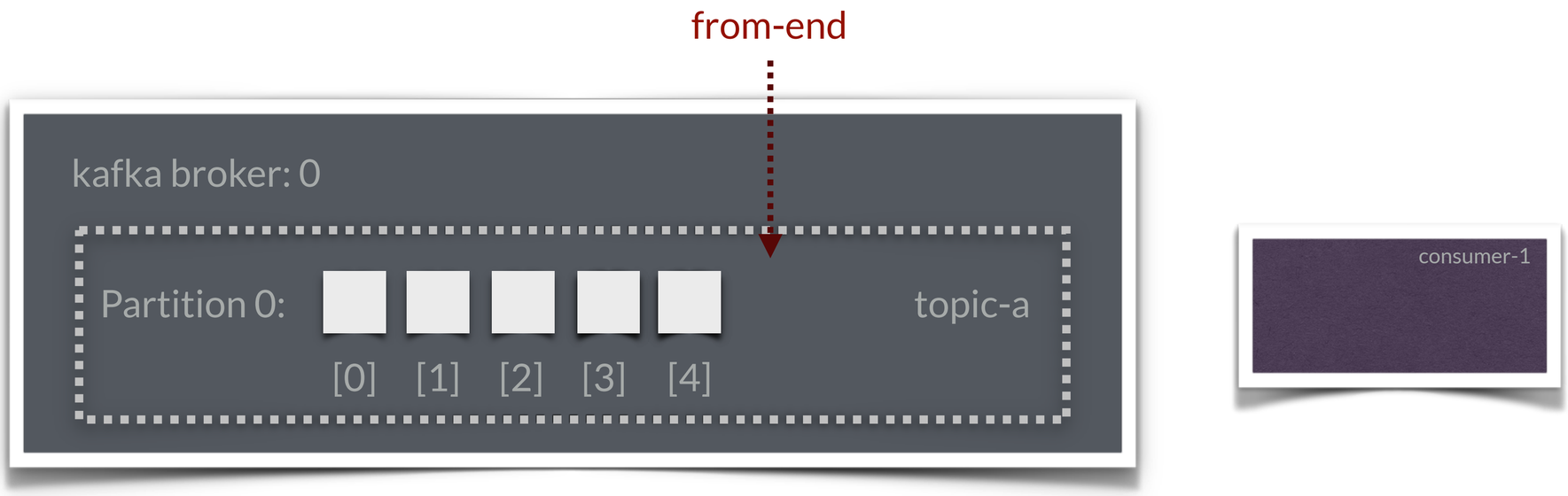
Retention: The **data is temporary**

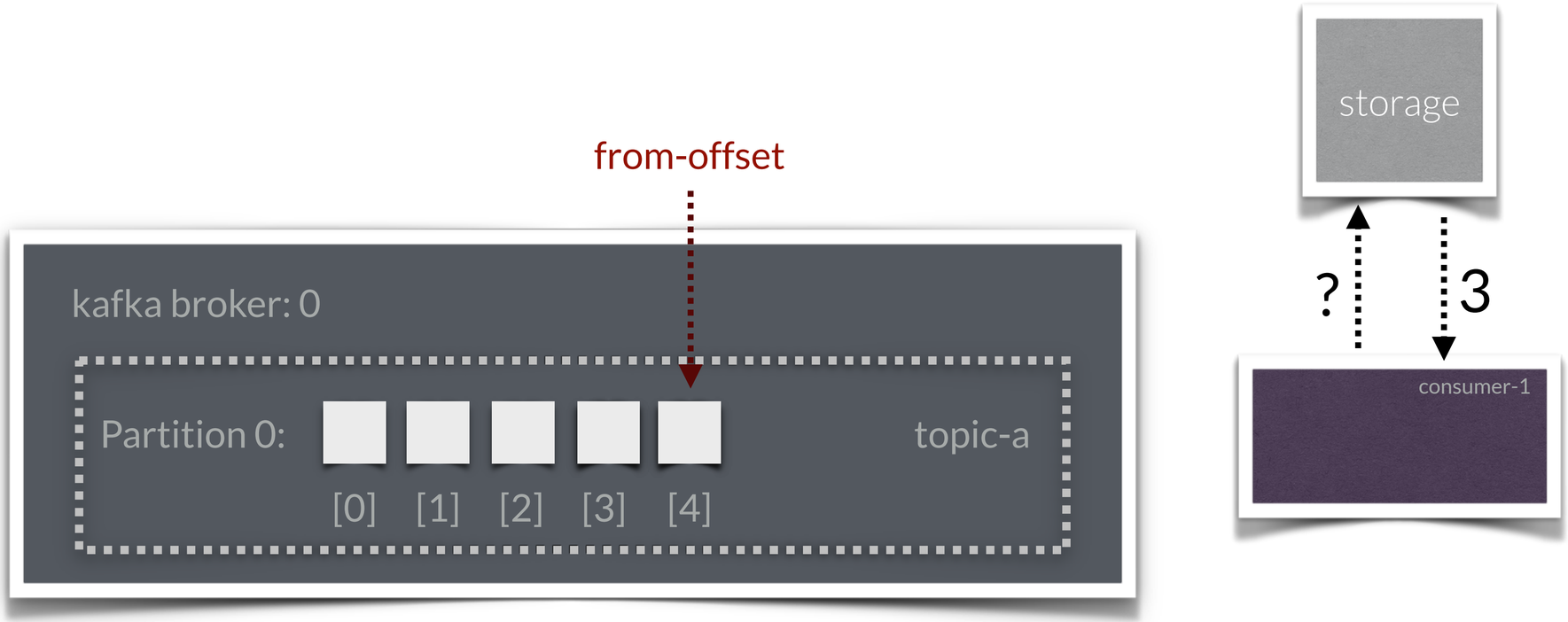
Kafka Consumers


consumer-1_offset
kafka broker: 0
Partition 0:
[0]
[1]
[2]
topic-a
consumer-1

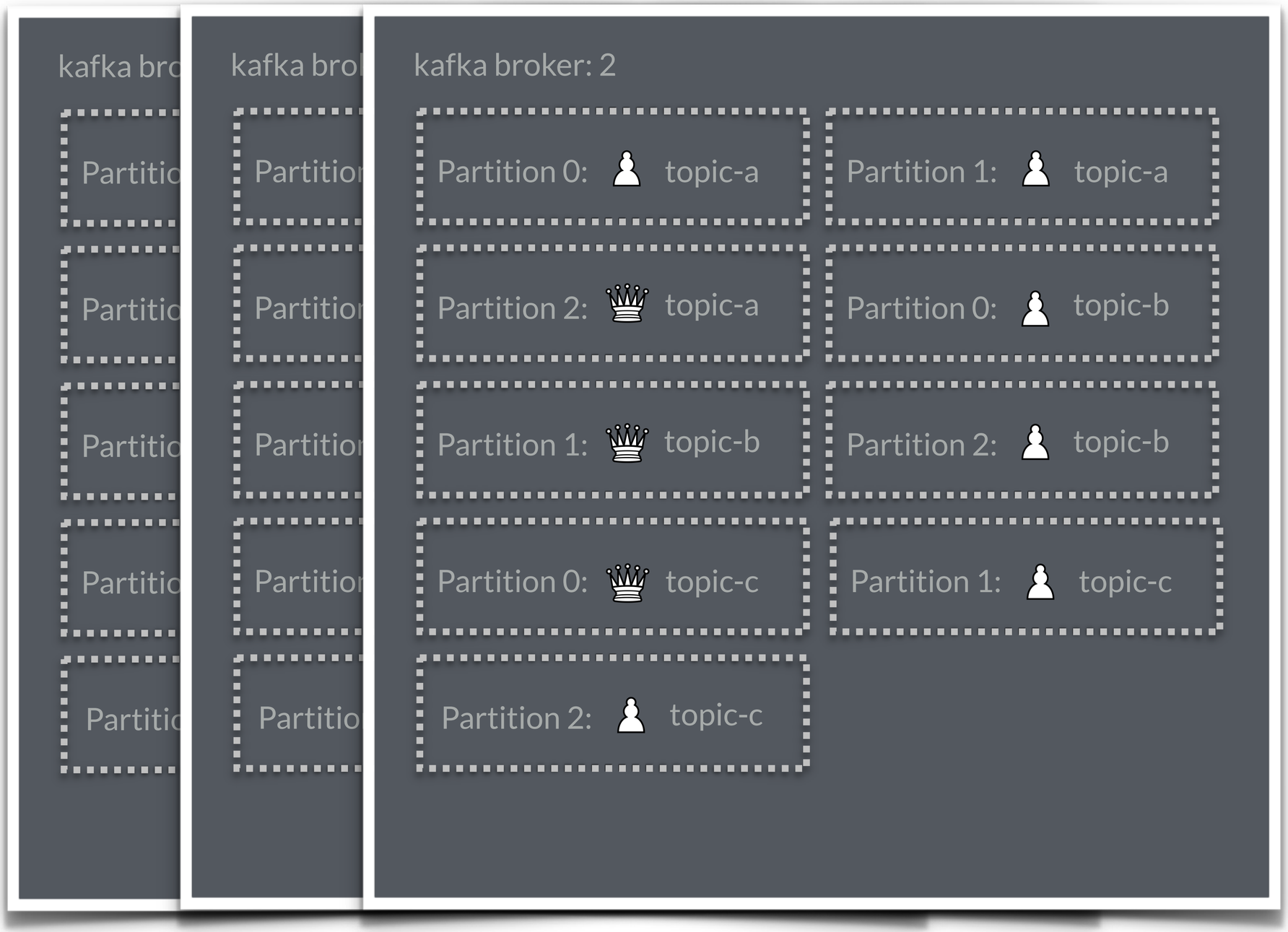

consumer-1_offset
kafka broker: 0
Partition 0:
topic-a
[0]
[1]
[2]
[3]
[4]
consumer-1


from-beginning
kafka broker: 0
Partition 0:
topic-a
[0]
[1]
[2]
[3]
[4]
consumer-1


from-end
kafka broker: 0
Partition 0:
[0] [1] [2] [3] [4]
topic-a
consumer-1


from-offset
kafka broker: 0
Partition 0:
[0]
[1]
[2]
[3]
[4]
topic-a
storage
?
3
consumer-1

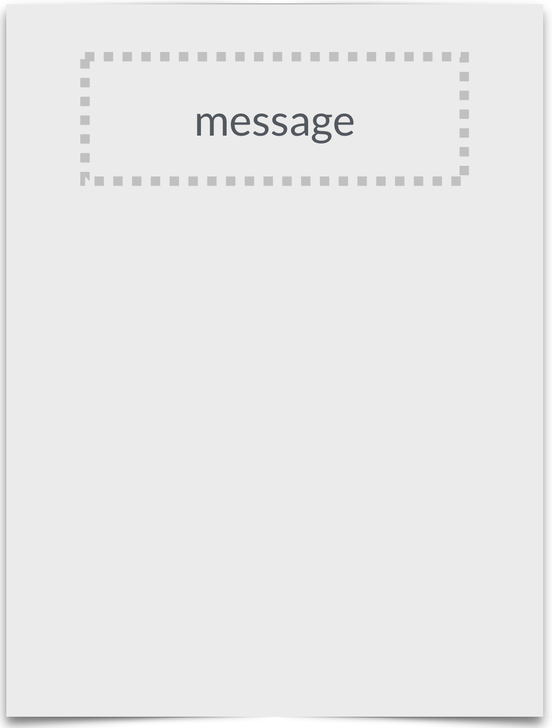
Each partition is on a different broker,
therefore a single topic is scaled


kafka bro
Partitio
Partitio
Partitio
Partitio
Partitio
kafka brok
Partition
Partition
Partition
Partition
Partitio
kafka broker: 2
Partition 0: topic-a
Partition 1: topic-a
Partition 2: topic-a
Partition 0: topic-b
Partition 1: topic-b
Partition 2: topic-b
Partition 0: topic-c
Partition 1: topic-c
Partition 2: topic-c

What is a message?

A Kafka Message

- Similar to a *row* or a *record*
- Message is an array of bytes
- No special serialization, that is done at the producer or consumer



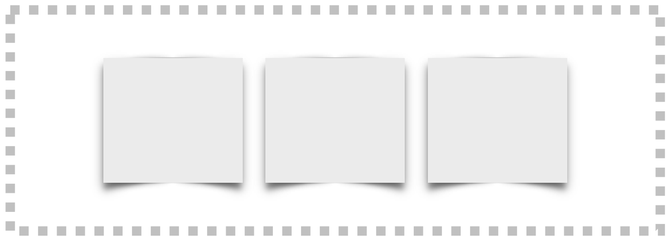
A Kafka Message Key

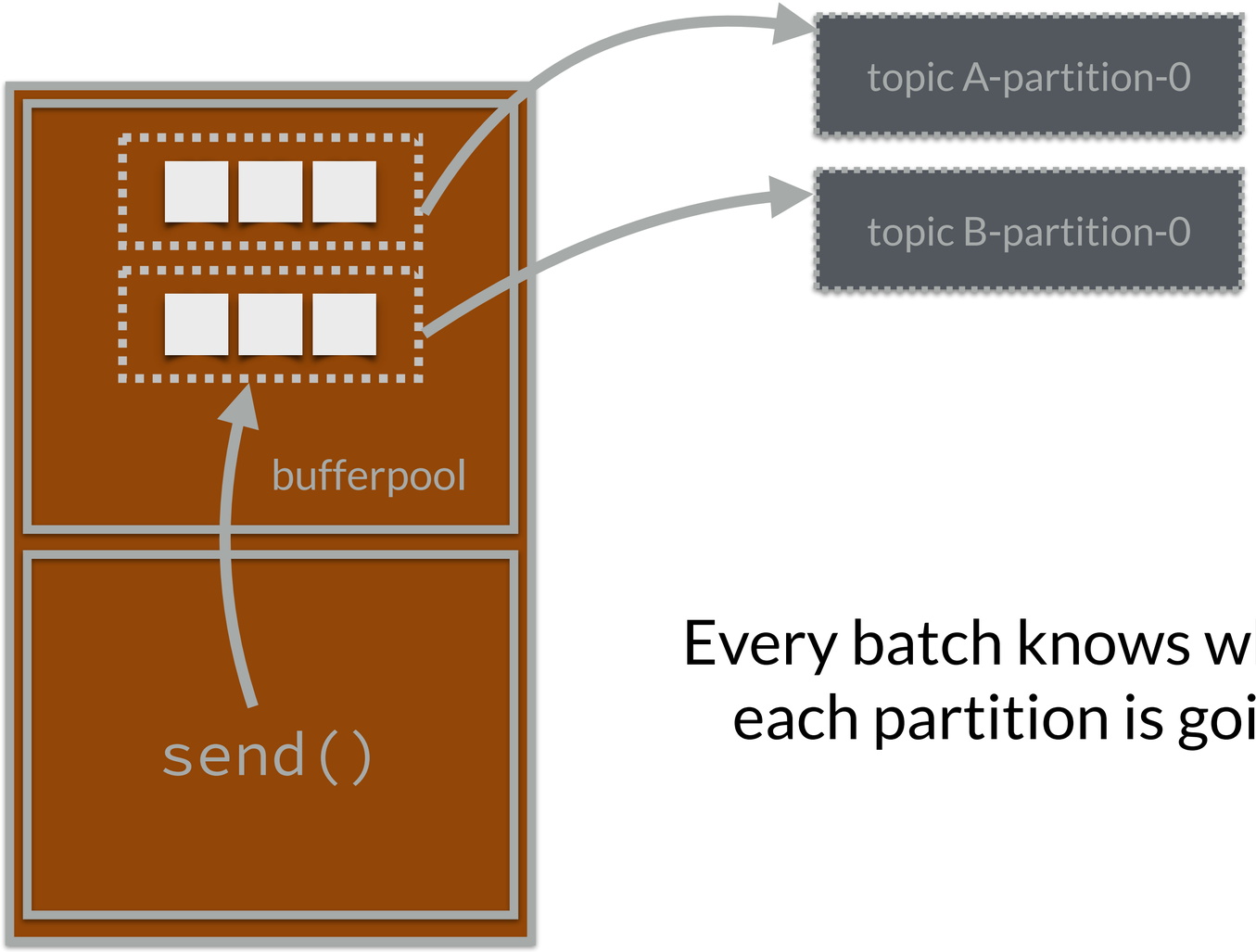
- Message may contain a *key* for better distribution to partitions
- The *key* is also an array of bytes
- If a *key* is provided, a partitioner will hash the key and map it to a single partition
- Therefore it is the only time that something is guaranteed to be in order

Programming a Producer

A Kafka Batch

- A collection of messages, that is sent, configured in *bytes*
- Sent to the same *topic* and the same *partition*
- *Avoids overhead of sending multiple message over the wire*





Every batch knows where each partition is going

```
murmur2(bytes) % number partitions
```

Acks

Acks

acks controls how many partition replicas must receive the record before the write is considered a success.

Acks

acks	description
0	No acknowledgment, assume all is well
1	At least one replica will Producer will receive a success response, error if unsuccessful, up to client to hand
all	All replicas must acknowledge. Higher latency, safest

Establishing Properties

- Construct a `java.util.Properties` object
- Provide two or more locations where the bootstrap servers are located
- Provide a Serializer for the key
- Provide a Serializer for the value

```
Properties properties = new Properties();
properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
    "localhost:9092");
properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
    StringSerializer.class);
properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
    IntegerSerializer.class);
```

Create a Producer Object

- Construct a `org.apache.kafka.clients.producer.KafkaProducer` object
- A Kafka Producer is thread-safe
- Inject the `Properties` into the `Producer` object

```
KafkaProducer<String, Integer> producer = new KafkaProducer<>(properties);
```

Creating a Record/Message

- Create a `org.apache.kafka.clients.producer.ProducerRecord`
- Accepts many parameters but the main ones are:
  - Topic
  - Key (if applicable)
  - Value

```
ProducerRecord<String, Integer> producerRecord =
    new ProducerRecord<>("my_orders", state, amount);
```

Sending a message

- Send the record by calling `send` on the `Producer`
- Returns a `Future` object to process the results on another Thread.
- The `Future` will contain `RecordMetadata`, an object that has information about your send.

```
Future<RecordMetadata> send = producer.send(producerRecord);
```

RecordMetadata

Contains information about your send including the messages

```
if (metadata.hasOffset()) {
    System.out.format("offset: %d\n",
        metadata.offset());
}
System.out.format("partition: %d\n",
    metadata.partition());
System.out.format("timestamp: %d\n",
    metadata.timestamp());
System.out.format("topic: %s\n", metadata.topic());
System.out.format("toString: %s\n",
    metadata.toString());
```

Sending with a Callback

- Alternately, you can send with a `Callback` (lambda)
- `Callback` is an interface, can be use as a lambda
- If `RecordMetadata` is null there was an error, if `Exception` is `null` the send was successful

```
producer.send(producerRecord, new Callback() {
    @Override
    public void onCompletion(RecordMetadata metadata,
                             Exception e) {
        ...
    }
}
```

Using a Closure to capture Key and Value

- RecordMetadata does not have information on key and value
- Using a closure you can obtain that in the block of your lambda

```
producer.send(producerRecord, (metadata, e) -> {
    if (metadata != null) {
        System.out.println(producerRecord.key());
        System.out.println(producerRecord.value());
    }
}
```

Be a good citizen, close your resources

- When you need to terminate, flush messages from the bufferpool
- Close the Producer

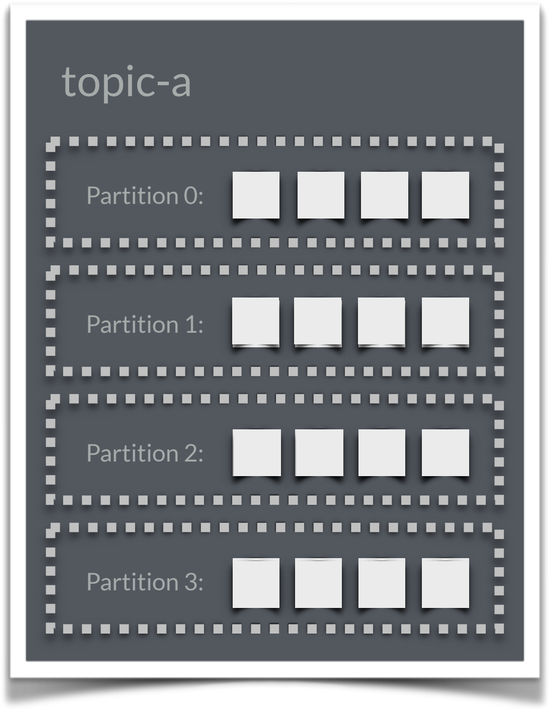
```
producer.flush();
producer.close();
```

Closing Resources in a Shutdown Hook

- Runtime.getRuntime().addShutdownHook(...) will listen for SIGTERM (CTRL+C)
- This would make an excellent place to flush and close, and close any loops that you may have created

```
Runtime.getRuntime().addShutdownHook(new Thread(() -> {
    done.set(true);
    producer.flush();
    producer.close();
}));
```

Programming a Consumer


topic-a
Partition 0:
Partition 1:
Partition 2:
Partition 3:

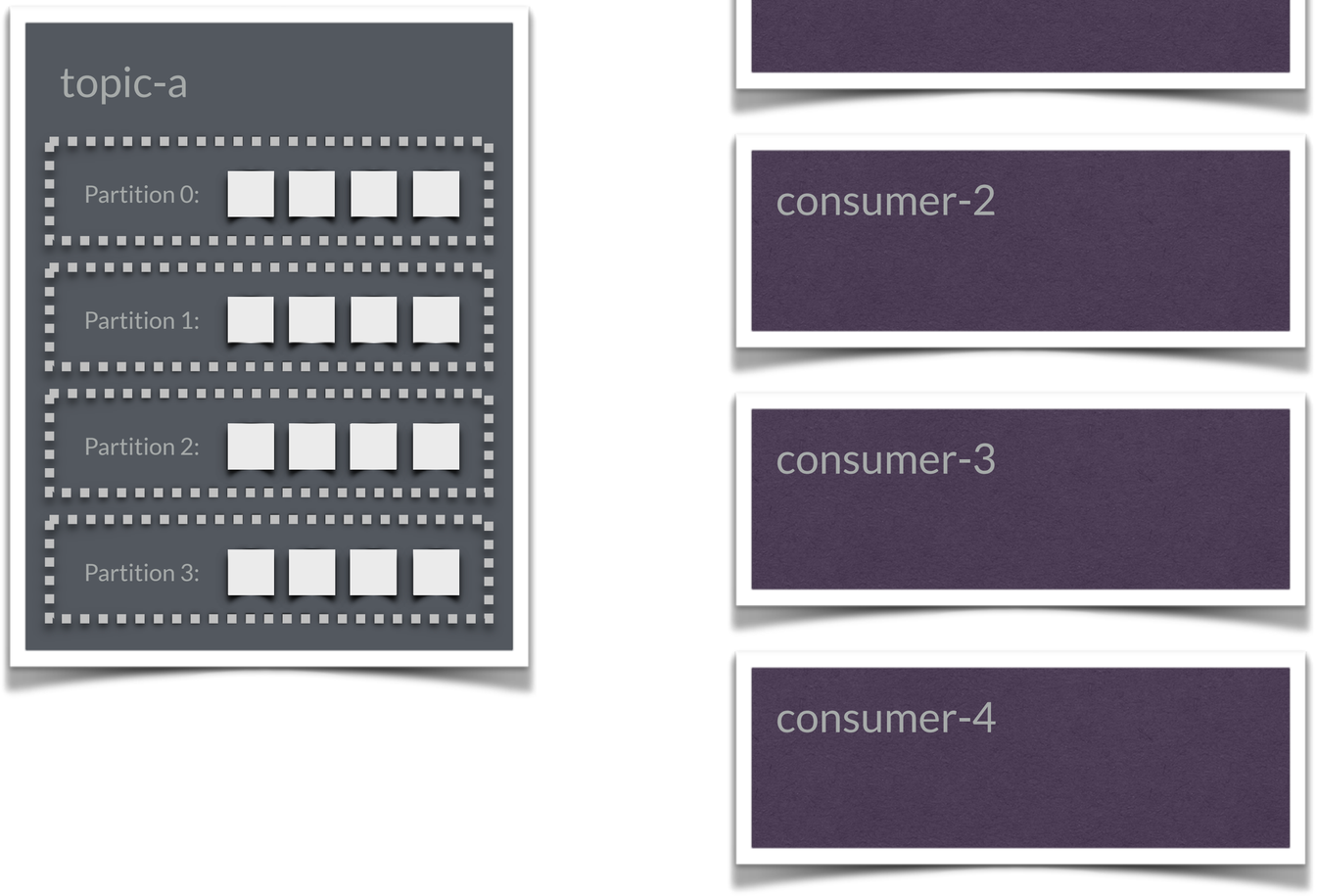

consumer-1

Kafka Consumer Groups

Consumers are typically done as a group

A single consumer will end up inefficient with large amounts of data

A consumer may never catch up


topic-a
Partition 0:
Partition 1:
Partition 2:
Partition 3:
consumer-1
consumer-2
consumer-3
consumer-4

__consumer_offsets

Topic on Kafka brokers that contain information about consumer and their offsets, stored in Kafka’s data directory

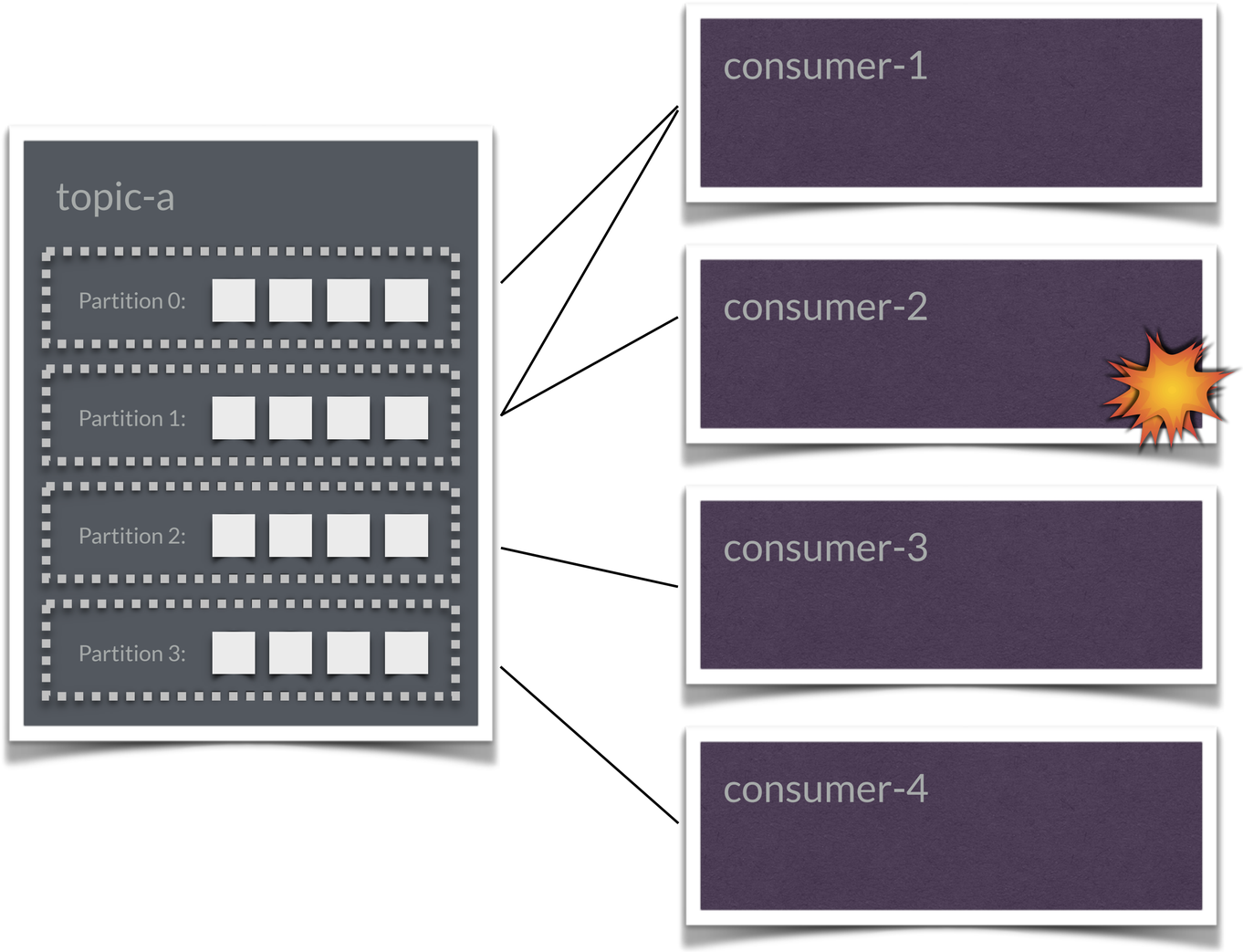
Consumer Rebalancing

Consumer Rebalance

When one partition is moved from one consumer to another, this is known as a *rebalance*.

A way to mitigate when either consumers go down, or when consumers or added.

Although unavoidable, this will cause an unfortunate pause, and it ***will lose state***.


topic-a
Partition 0:
Partition 1:
Partition 2:
Partition 3:
consumer-1
consumer-2
consumer-3
consumer-4

Lab: Deploying Redis & Kafka

Kafka Streams


TK45
115m
ÚNIK.C.

Stream Processing

Endless supply of data

“Replayable"

Real Time Processing for Fraud Detection, High End Sales Detection, Internet of Things, and More.

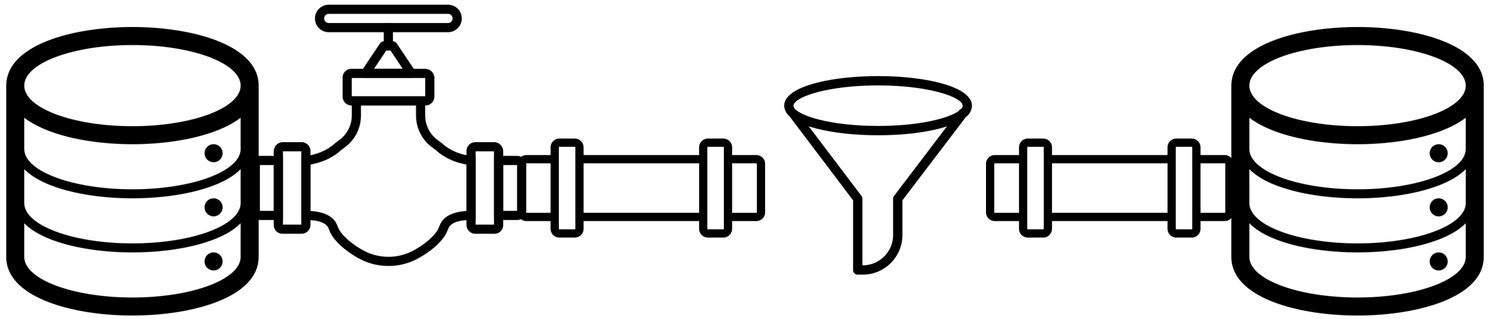
Will require built-in accumulator table to perform real time data processing, like *counting*, and *grouping*


NY
100.00

Filtering

NY

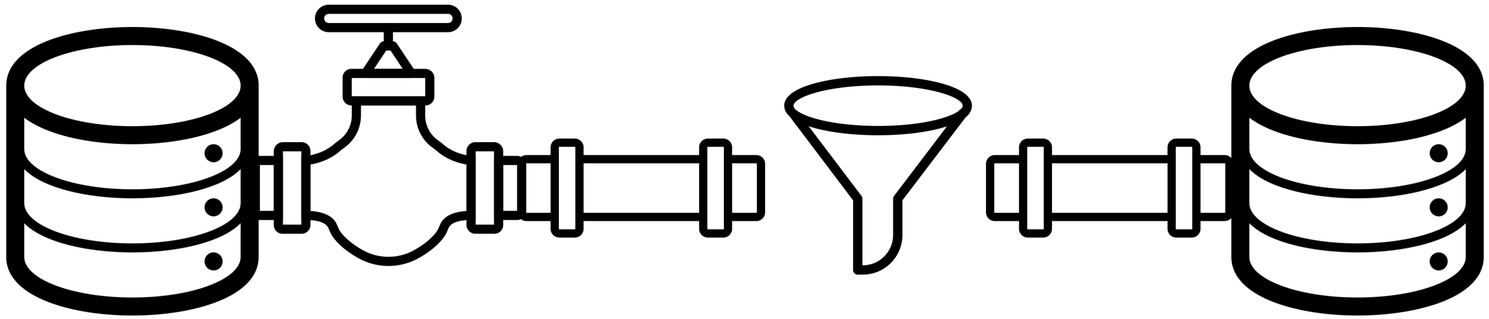
100.00



```
value > 10000
```

OH

20000

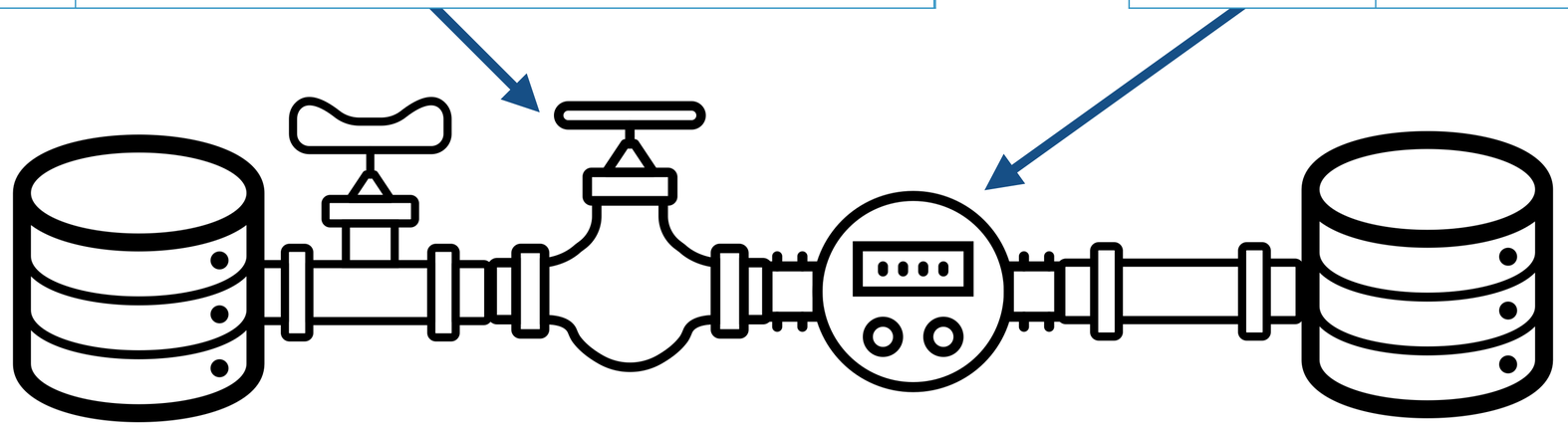


```
value > 10000
```

Aggregating

Key	Value
OH	100.0

Key	Value
OH	100.0



OH

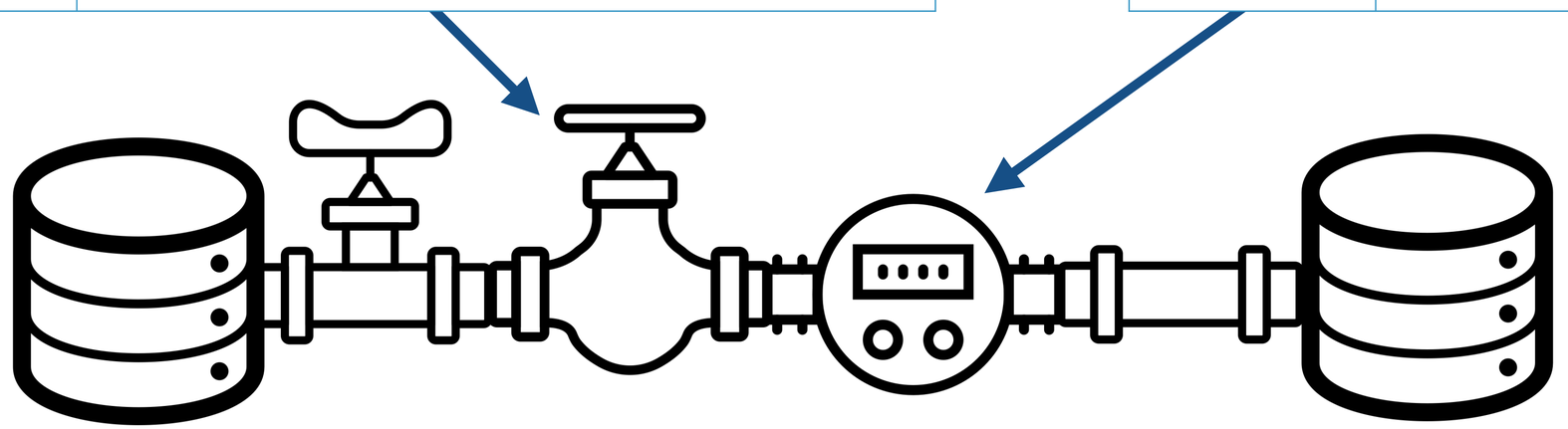
100.00

OH

100.00

Key	Value
OH	100.0
NY	20.0

Key	Value
OH	100.0
NY	20.0



NY

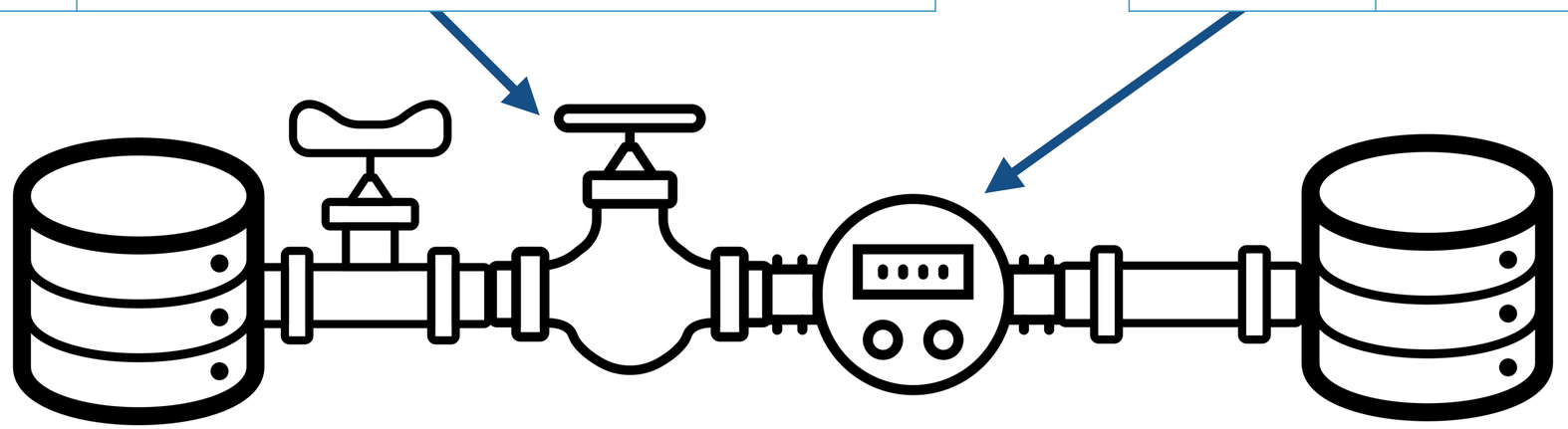
20

NY

20

Key	Value
OH	100.0
NY	20.0, 60.0

Key	Value
OH	100.0
NY	80.0

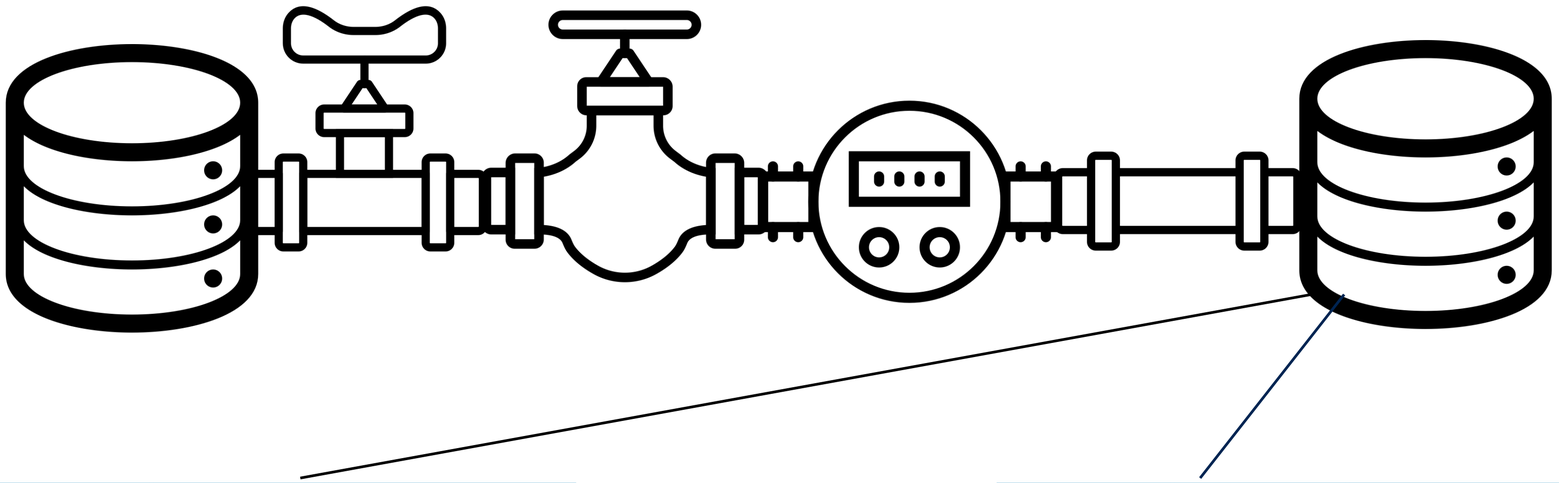


NY

60

NY

80.0

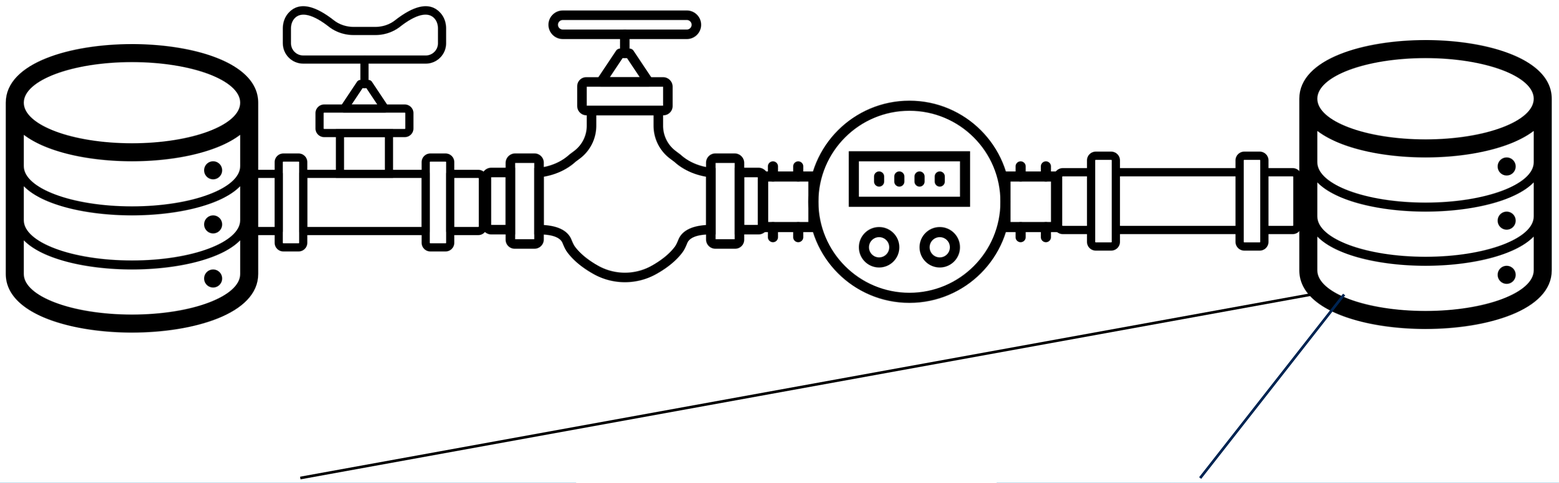


Offset	Key	Value
0	**OH**	100.0

Partition 0

Offset	Key	Value
0	**NY**	20.0
1	**NY**	80.0

Partition 1



Offset	Key	Value
0	**OH**	100.0

Partition 0

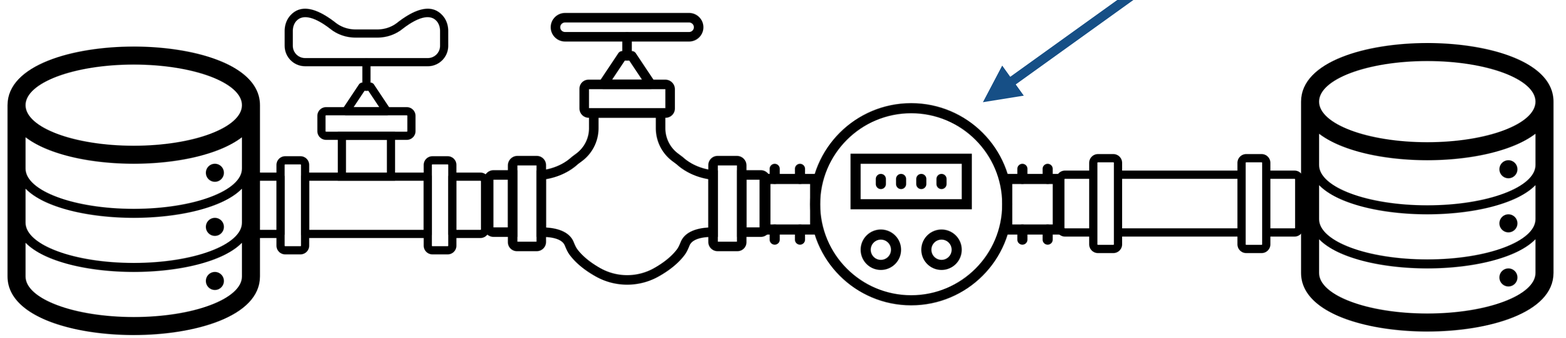
Offset	Key	Value
1	**NY**	80.0

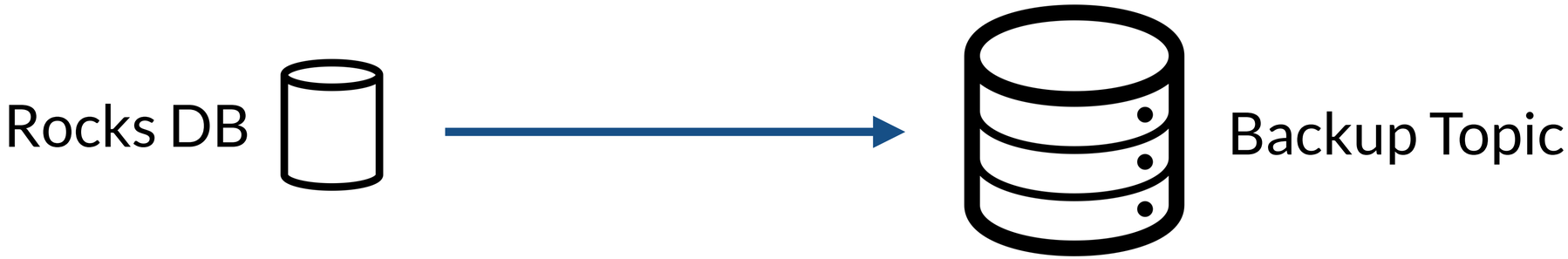
Partition 1

After Compaction

Rocks DB

Key	Value
OH	100.0
NY	80.0




Rocks DB
Backup Topic

Lab: Deploying Our Stream

KSQL

KSQL

Open source streaming SQL engine for Apache Kafka

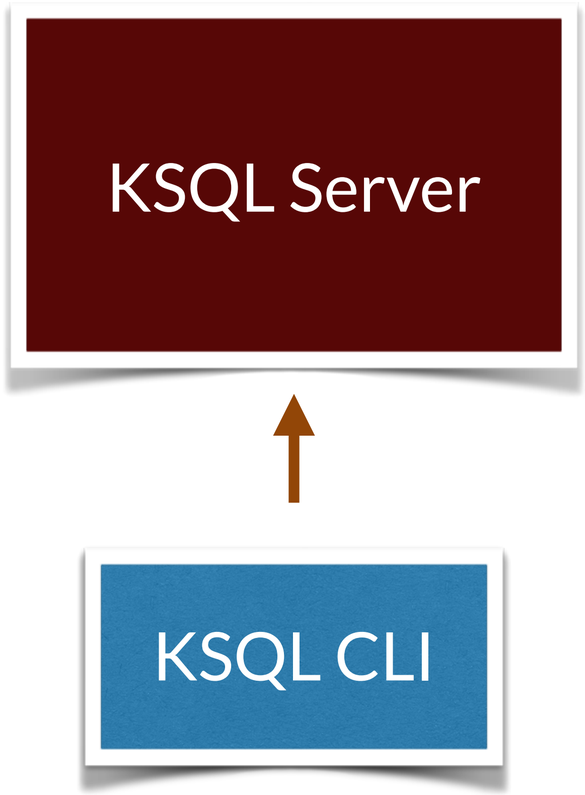
Provides SQL interface for stream processing on Kafka

Scalable, elastic, fault-tolerant, and real-time.

KSQL

Supports a wide range of streaming operations, including:

- Data filtering
- Transformations
- Aggregations
- Joins
- Windowing
- Sessionization


KSQL Server
KSQL CLI

KSQL Architecture

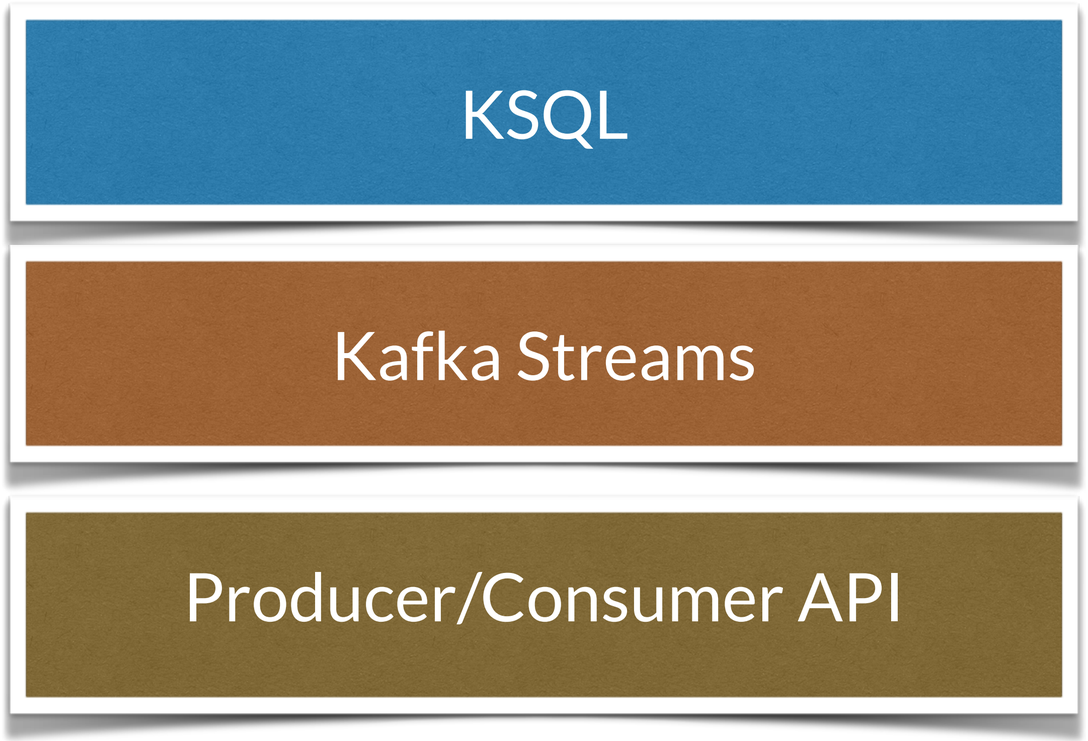
KSQL Server communicates with the broker

KSQL servers are run separately from the KSQL CLI client and Kafka brokers.

You can deploy servers on remote machines, VMs, or containers

KSQL CLI interacts with KSQL Servers via a REST API

KSQL Server runs in two modes: CLI, Headless


KSQL
Kafka Streams
Producer/Consumer API

KSQL CLI Mode

```
/bin/ksql http://localhost:8088
```

KSQL in Headless Mode

```
/bin/ksql-start-server \
server_path/etc/ksql/ksql-server.properties \
--queries-file /path/to/queries.sql
```

Tapping a Stream Based Topic

```
CREATE STREAM pageviews \
  (viewtime BIGINT, \
   userid VARCHAR, \
   pageid VARCHAR) \
  WITH (KAFKA_TOPIC='pageviews', \
        VALUE_FORMAT='DELIMITED');
```

Tapping a Table Based Topic

```
CREATE TABLE users \
  (registertime BIGINT, \
   userid VARCHAR, \
   gender VARCHAR, \
   regionid VARCHAR) \
  WITH (KAFKA_TOPIC = 'users', \
        VALUE_FORMAT='JSON', \
        KEY = 'userid');
```

Showing the Streams

```
SHOW STREAMS;

Stream Name | Kafka Topic | Format
-----------------------------------------
PAGEVIEWS   | pageviews   | DELIMITED
-----------------------------------------
```

Showing the Tables

```
SHOW TABLES;

 Table Name   | Kafka Topic | Format
-----------------------------------------
 USERS        | users       | JSON
-----------------------------------------
```

Supported SQL Types

- `BOOLEAN`
- `INTEGER`
- `BIGINT`
- `DOUBLE`
- `VARCHAR` (or `STRING`)
- `ARRAY<ArrayType>` (JSON and AVRO only. Index starts from 0)
- `MAP<VARCHAR, ValueType>` (JSON and AVRO only)
- `STRUCT<FieldName FieldType, … >` (JSON and AVRO only)

Various Operators

Scalar Operators:

```
ABS, ARRAYCONTAINS, CEIL, CONCAT,
EXTRACTJSONFIELD, FLOOR, GEO_DISTANCE, IFNULL, LC
ASE, LEN, MASK, RANDOM, ROUND,
STRINGTOTIMESTAMP, SUBSTRING, TIMESTAMPTOSTRING,
TRIM, UCASE
```

Aggregator Operators:

```
COUNT, MAX, MIN, SUM, TOPK, TOPKDISTINCT, WINDOWS
TART, WINDOWEND
```

Displays Information

`SHOW TOPICS`	Show all topics
`SHOW <TOPIC>`	Display Topic
`SHOW STREAMS`	Show Streams
`SHOW TABLES`	Show Tables
`SHOW FUNCTIONS`	Show Available Functions
`SHOW QUERIES`	Show Persistent Queries

Describing

`DESCRIBE`	List columns in stream or table
`DESCRIBE EXTENDED`	Describe in detail stream or table; runtime statistics, and queries that populate the stream or table

Persistent Query

```
CREATE TABLE users_female AS \
  SELECT userid, gender, regionid FROM users \
  WHERE gender='FEMALE';
```

What makes this a persistent query is the form
`CREATE (TABLE|STREAM) AS SELECT`
Once executed, it will continuously run, until terminated with `TERMINATE` command

Non-Persistent Query

```
SELECT * FROM pageviews
  WHERE ROWTIME >= 1510923225000
    AND ROWTIME <= 1510923228000;
```

A `LIMIT` can be used to limit the number of rows returned.
Once the limit is reached the query will terminate.

Persistent Vs. Non-Persistent

	Persistent	Non Persistent
Where does data go?	Will store into a topic	Will display on screen
How do I stop it?	Find query id using `SHOW QUERIES` and `TERMINATE <id>`	`CTRL + C`
How to create?	`CREATE STREAM AS SELECT ...`	`SELECT ....`

Selecting Customized Fields

```
CREATE STREAM pageviews_transformed \
  WITH (TIMESTAMP='viewtime', \
        PARTITIONS=5, \
        VALUE_FORMAT='JSON') AS \
  SELECT viewtime, \
         userid, \
         pageid, \
         TIMESTAMPTOSTRING \
           (viewtime, 'yyyy-MM-dd HH:mm:ss.SSS') \
              AS timestring \
  FROM pageviews \
  PARTITION BY userid;
```

Start from the Beginning

```
SET 'auto.offset.reset' = 'earliest';
```

Tumbling Window

```
SELECT item_id, SUM(quantity)
  FROM orders
  WINDOW TUMBLING (SIZE 20 SECONDS)
  GROUP BY item_id;
```

Hopping Window

```
SELECT item_id, SUM(quantity)
  FROM orders
  WINDOW HOPPING (SIZE 20 SECONDS,
     ADVANCE BY 5 SECONDS)
  GROUP BY item_id;
```

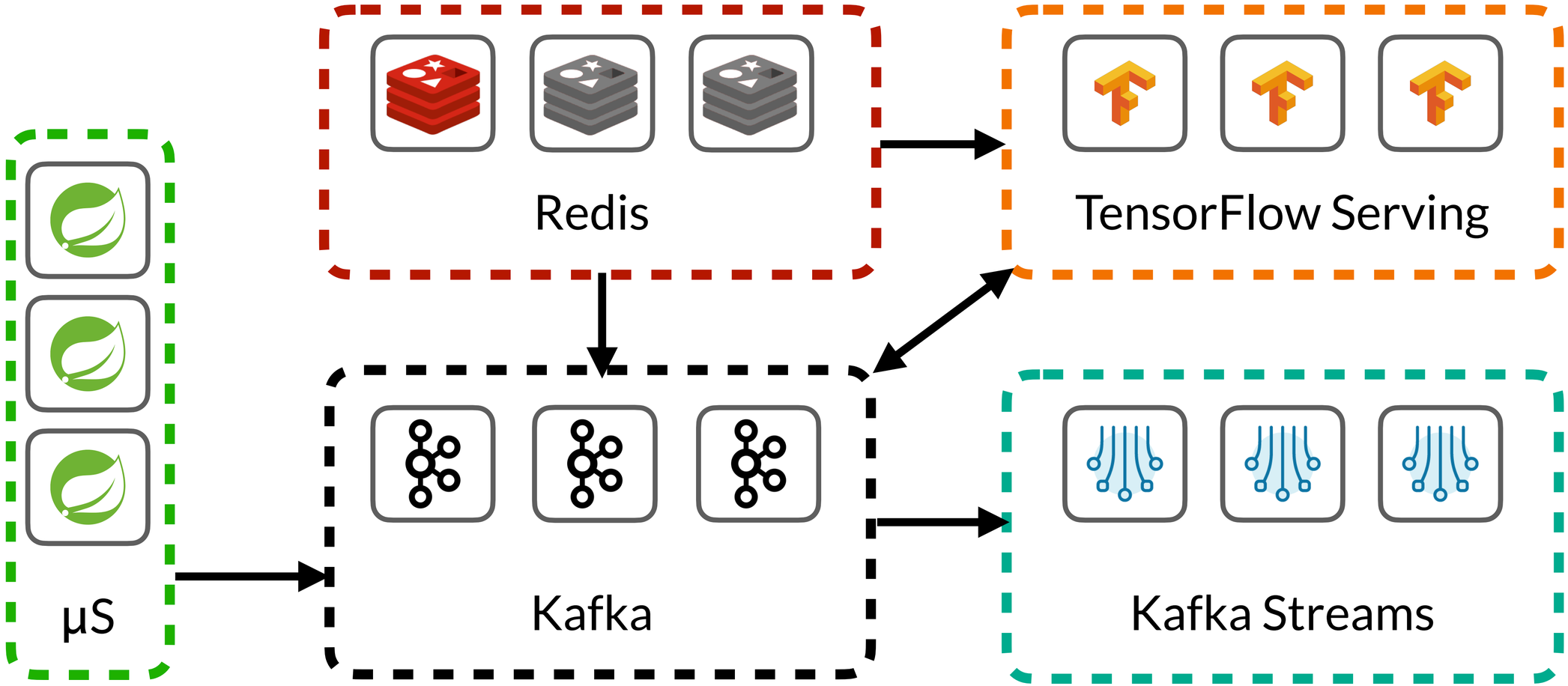
Session Window

```
SELECT item_id, SUM(quantity)
  FROM orders
  WINDOW SESSION (20 SECONDS)
  GROUP BY item_id;
```

Lab: Deploying KSQL

Conclusion


Java™


Redis
TensorFlow Serving
µS
Kafka
Kafka Streams

Conclusion

What your DataScience team requires will determine where you will operationalize your ML

Thank You

- Email: dhinojosa@evolutionnext.com
- Github: https://www.github.com/dhinojosa
- Twitter: http://twitter.com/dhinojosa
- Linked In: http://www.linkedin.com/in/dhevolutionnext