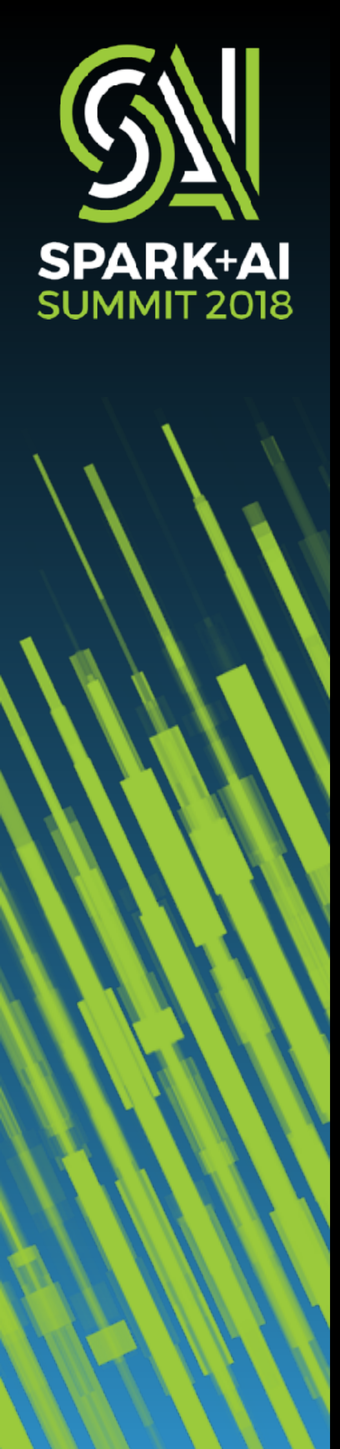
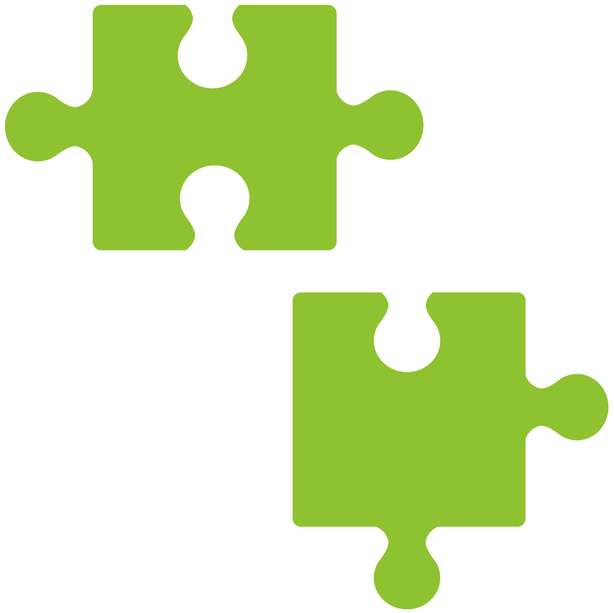


Building parallel machine learning algorithms: scaling out and up

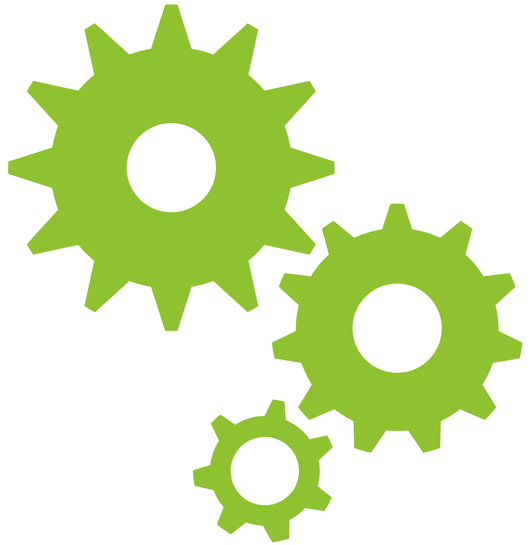
William Benton
willb@redhat.com • @willb

Motivation

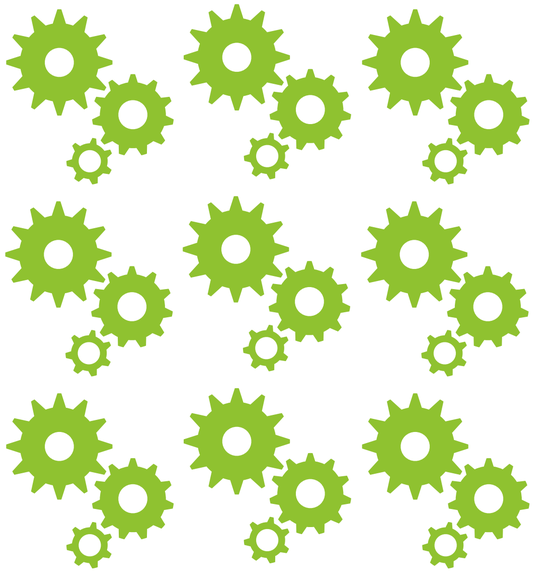


Motivation

Motivation



Motivation



Forecast

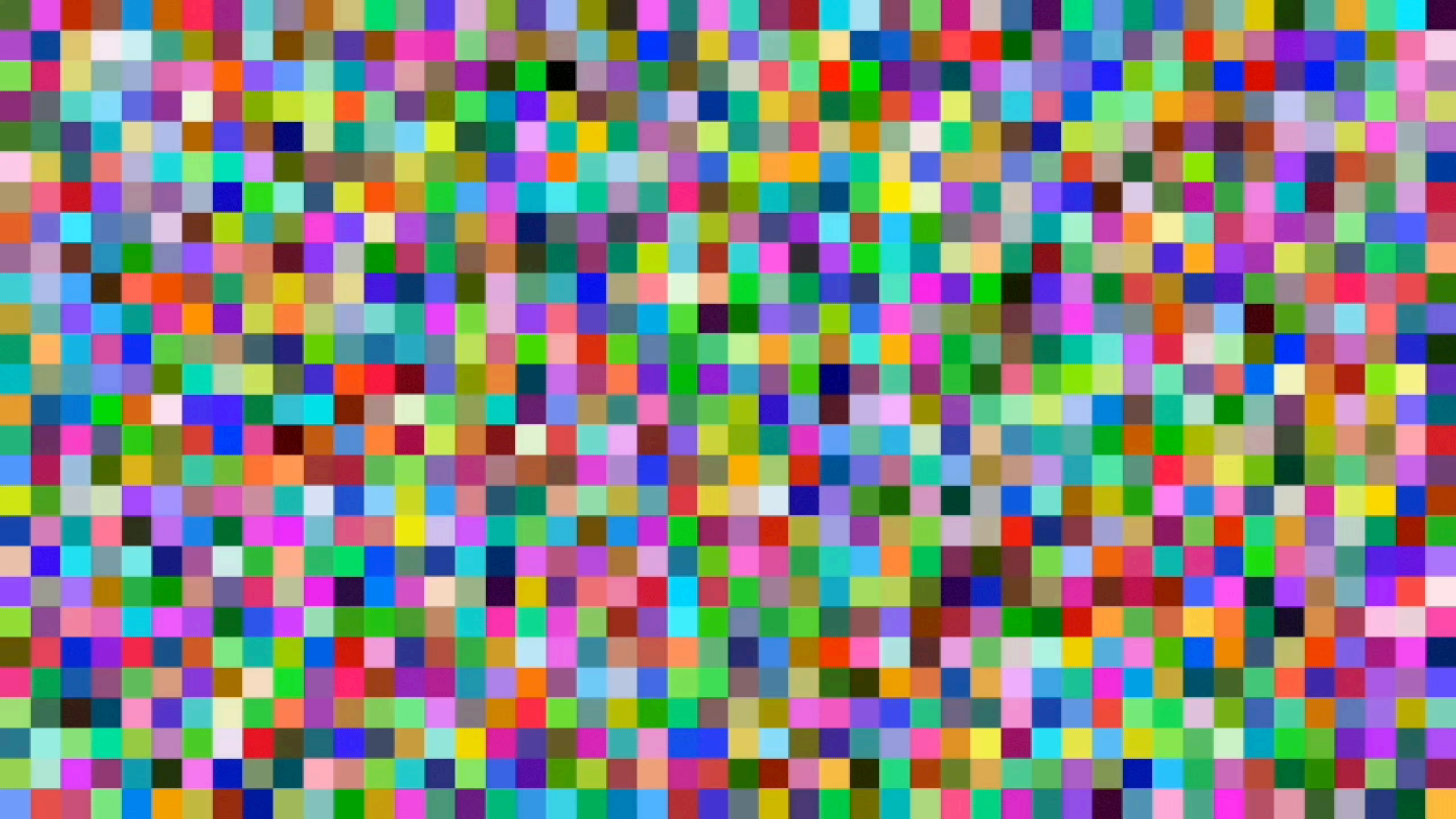
Introducing our case study: **self-organizing maps**

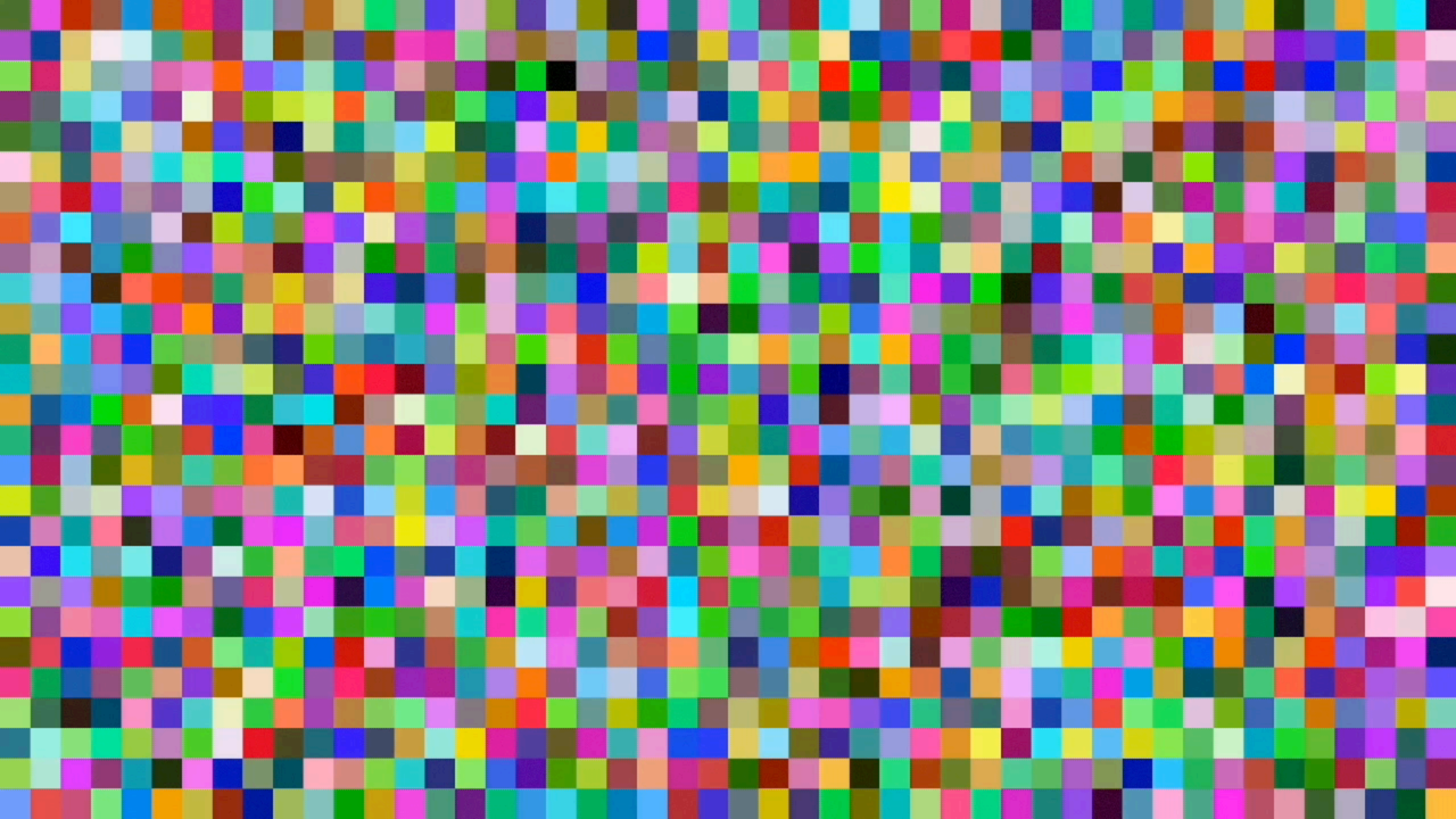
Parallel implementations for **partitioned collections** (in particular, **RDDs**)

Beyond the RDD: **data frames** and **ML pipelines**

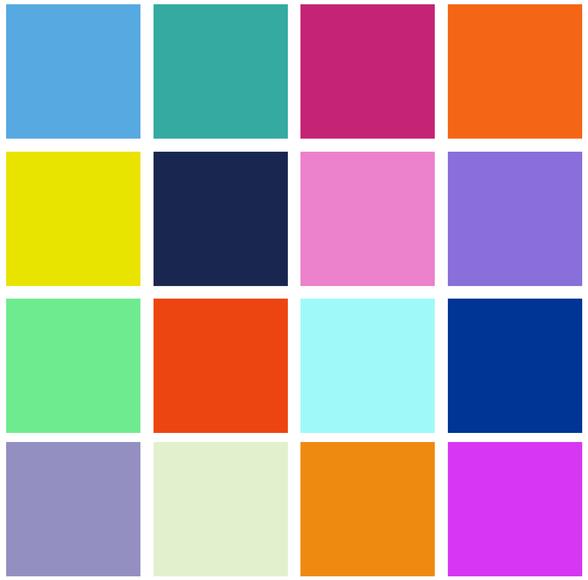
Practical considerations and **key takeaways**

Introducing our case study

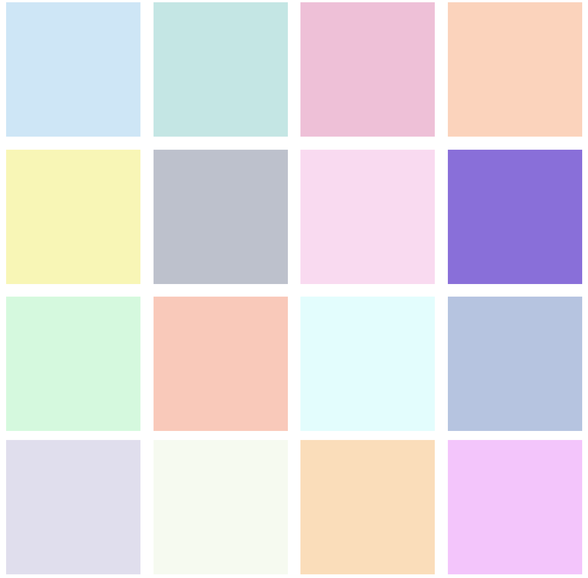




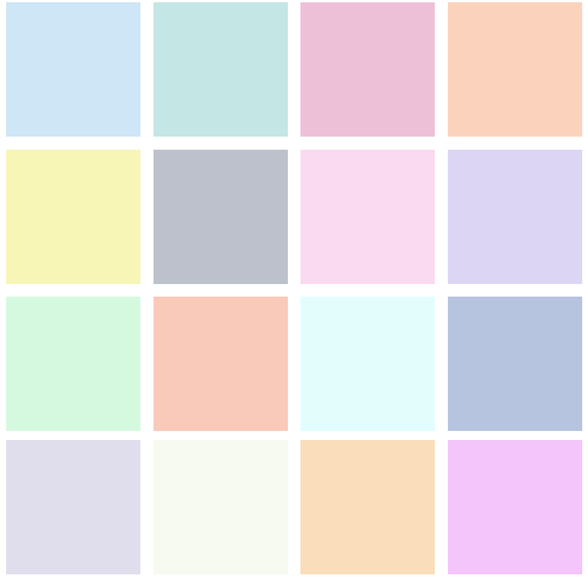
Training self-organizing maps



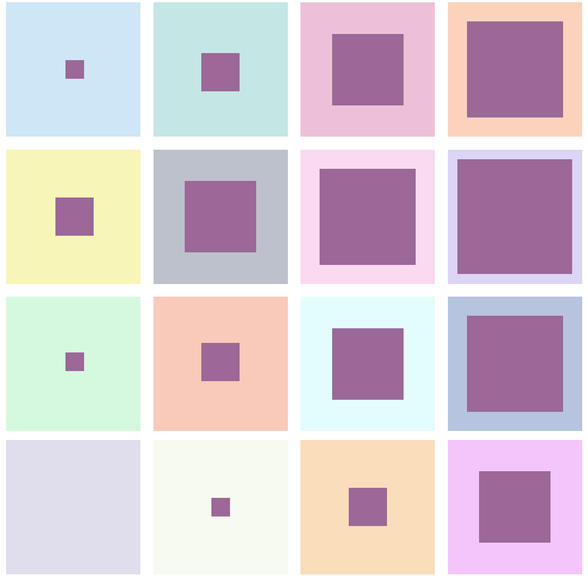
Training self-organizing maps



Training self-organizing maps



Training self-organizing maps



Training self-organizing maps

```
while t < maxupdates:
 random.shuffle(examples)
 for ex in examples:
    t = t + 1
    if t == maxupdates:
      break
    bestMatch = closest(som_t, ex)
    for (unit, wt) in neighborhood(bestMatch, sigma(t)):
      som_t+1[unit] = som_t[unit] + (ex - som_t[unit]) * alpha(t) * wt
```

Training self-organizing maps

```
while t < maxupdates:
 random.shuffle(examples)
 for ex in examples:
    t = t + 1
    if t == maxupdates:
      break
    bestMatch = closest(som_t, ex)
    for (unit, wt) in neighborhood(bestMatch, sigma(t)):
      som_t+1[unit] = som_t[unit] + (ex - som_t[unit]) * alpha(t) * wt
```

process the training set in random order

Training self-organizing maps

```
while t < maxupdates:
 random.shuffle(examples)
 for ex in examples:
    t = t + 1
    if t == maxupdates:
      break
    bestMatch = closest(som_t, ex)
    for (unit, wt) in neighborhood(bestMatch, sigma(t)):
      som_t+1[unit] = som_t[unit] + (ex - som_t[unit]) * alpha(t) * wt
```

process the training set in random order

the neighborhood size controls how much of the map around the BMU is affected

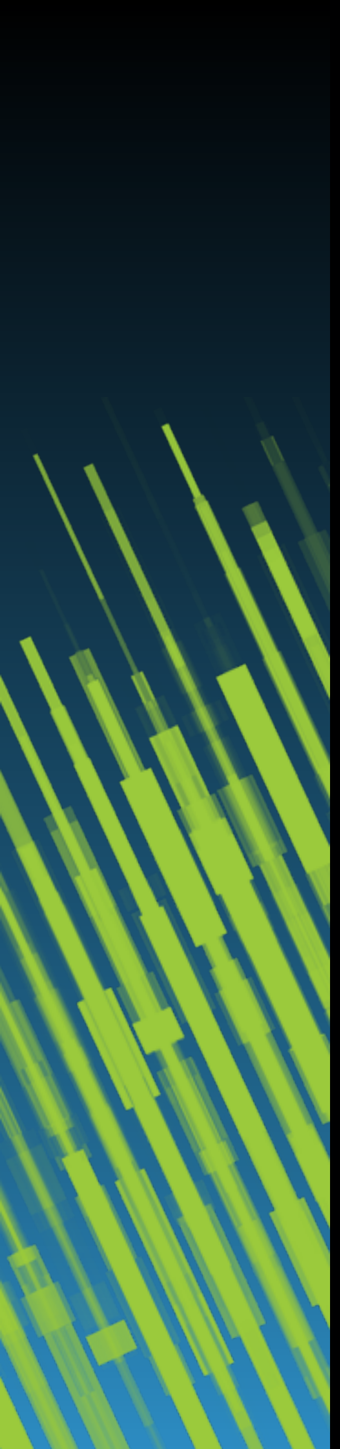
Training self-organizing maps

```
while t < maxupdates:
 random.shuffle(examples)
 for ex in examples:
    t = t + 1
    if t == maxupdates:
      break
    bestMatch = closest(som_t, ex)
    for (unit, wt) in neighborhood(bestMatch, sigma(t)):
      som_t+1[unit] = som_t[unit] + (ex - som_t[unit]) * alpha(t) * wt
```

process the training set in random order

the neighborhood size controls how much of the map around the BMU is affected

the learning rate controls how much closer to the example each unit gets

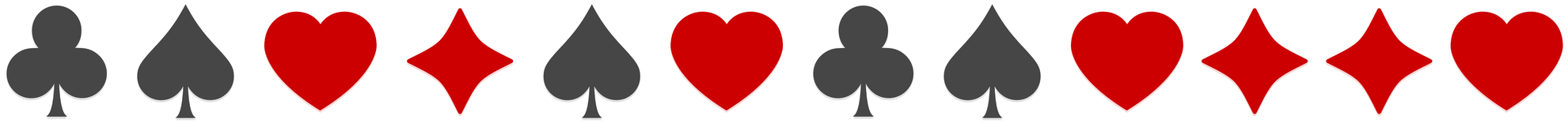


Parallel implementations for partitioned collections

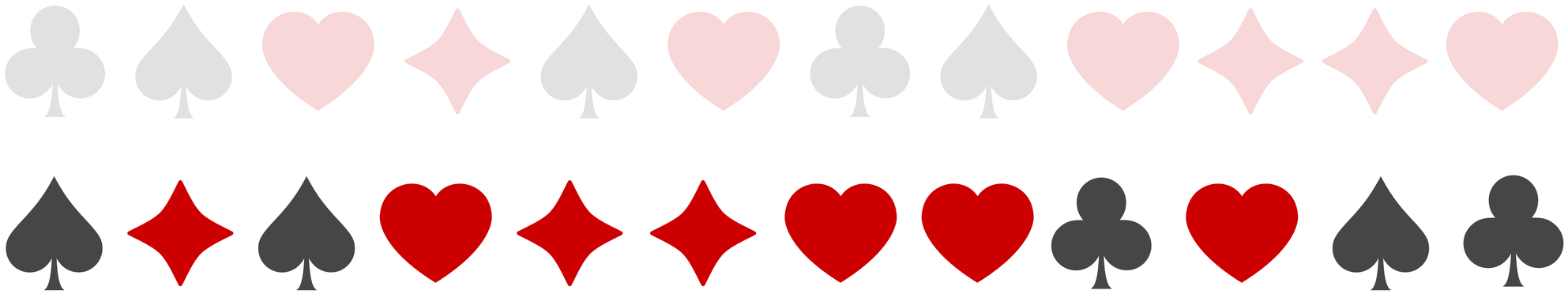
Historical aside: Amdahl's Law

$$\lim_{s_p \to \infty} S_o = \frac{1}{1 - p}$$

What forces serial execution?



What forces serial execution?



What forces serial execution?

```
state[t+1] =
  combine(state[t], x)
```

What forces serial execution?

```
state[t+1] =
  combine(state[t], x)
```

What forces serial execution?

```
f1: (T, T) => T
f2: (T, U) => T
```

What forces serial execution?

```
f1: (T, T) => T
f2: (T, U) => T
```

What forces serial execution?

✓ `f1: (T, T) => T`

✗ `f2: (T, U) => T`

How can we fix these?

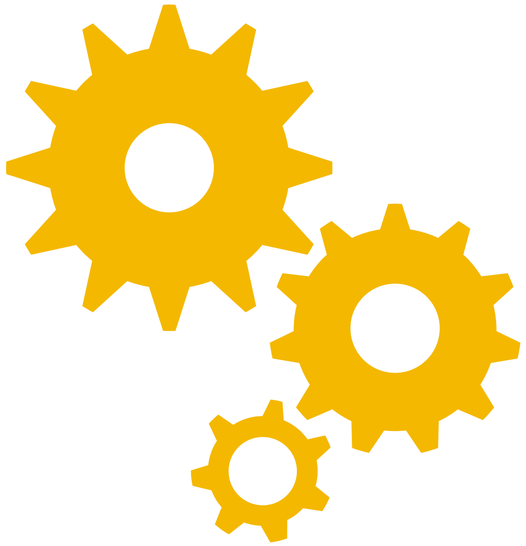
$$a \oplus b = b \oplus a$$

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$

How can we fix these?

$$a \oplus b = b \oplus a$$

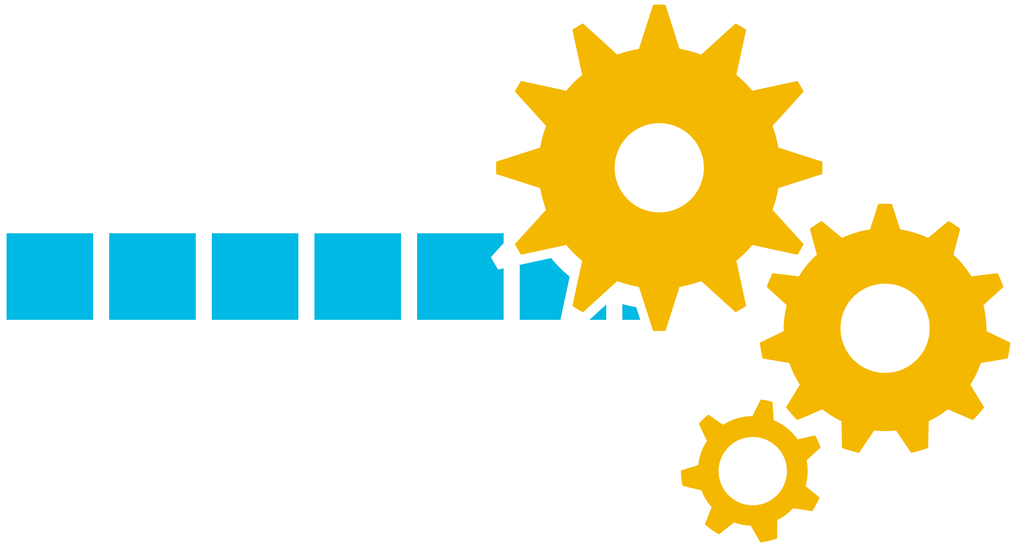
$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$



How can we fix these?

$$a \oplus b = b \oplus a$$

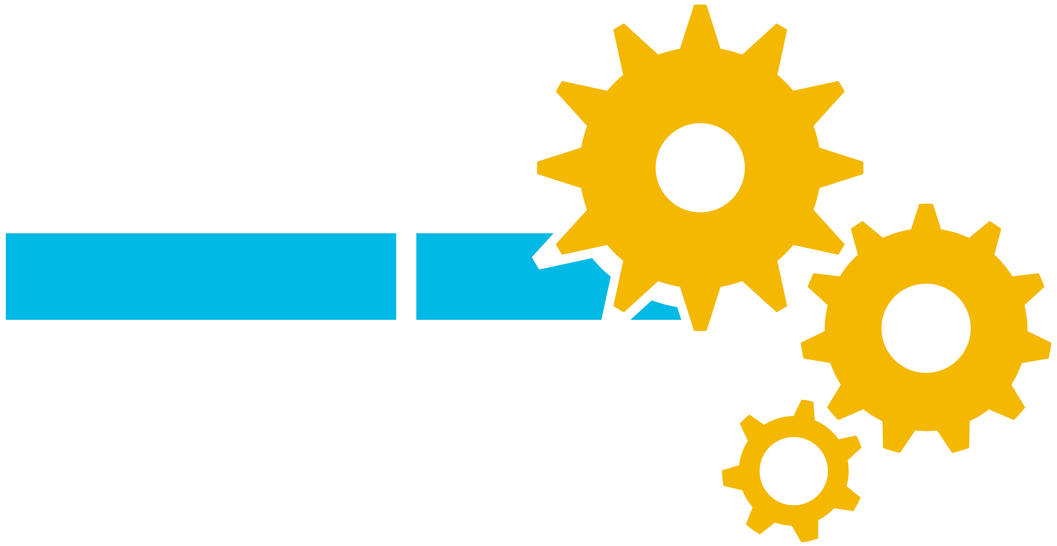
$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$



How can we fix these?

$$a \oplus b = b \oplus a$$

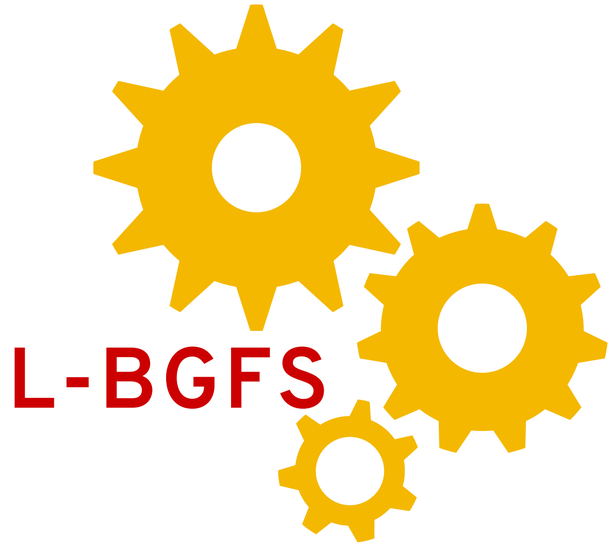
$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$



How can we fix these?

$$a \oplus b = b \oplus a$$

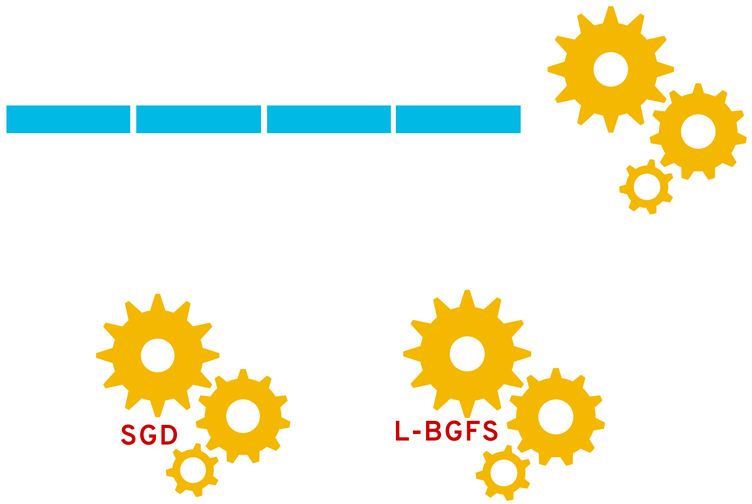
$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$



How can we fix these?

$$a \oplus b = b \oplus a$$

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$



There will be examples of **each** of **these** approaches for **many problems** in the **literature** and in **open-source code!**

Implementing atop RDDs

We'll start with a **batch implementation** of our technique:

```
for t in (1 to iterations):
  state = newState()
  for ex in examples:
    bestMatch = closest(som_{t-1}, ex)
    hood = neighborhood(bestMatch, sigma(t))
    state.matches += ex * hood
    state.hoods += hood
  som_t = newSOM(state.matches / state.hoods)
```

Implementing atop RDDs

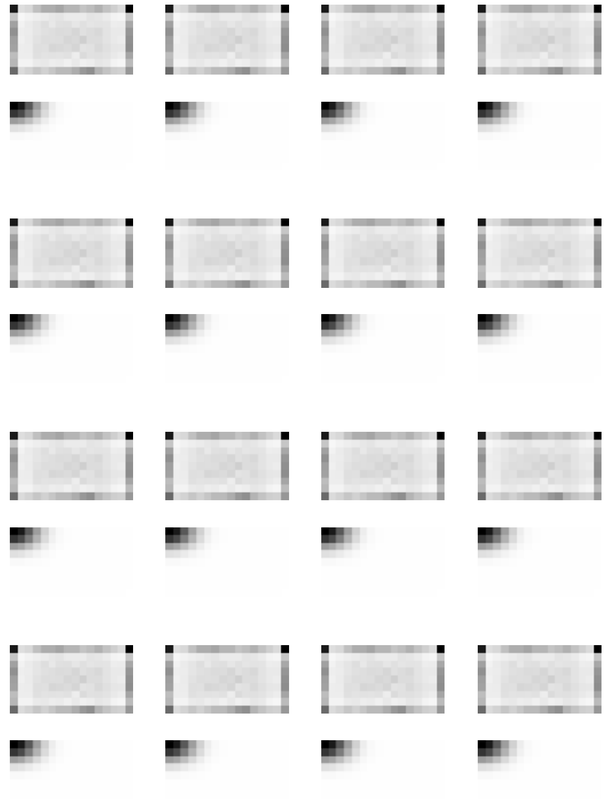
```
for t in (1 to iterations):
  state = newState()
  for ex in examples:
    bestMatch = closest(som_{t-1}, ex)
    hood = neighborhood(bestMatch, sigma(t))
    state.matches += ex * hood
    state.hoods += hood
  som_t = newSOM(state.matches / state.hoods)
```

Each batch produces a model that can be averaged with other models

Implementing atop RDDs

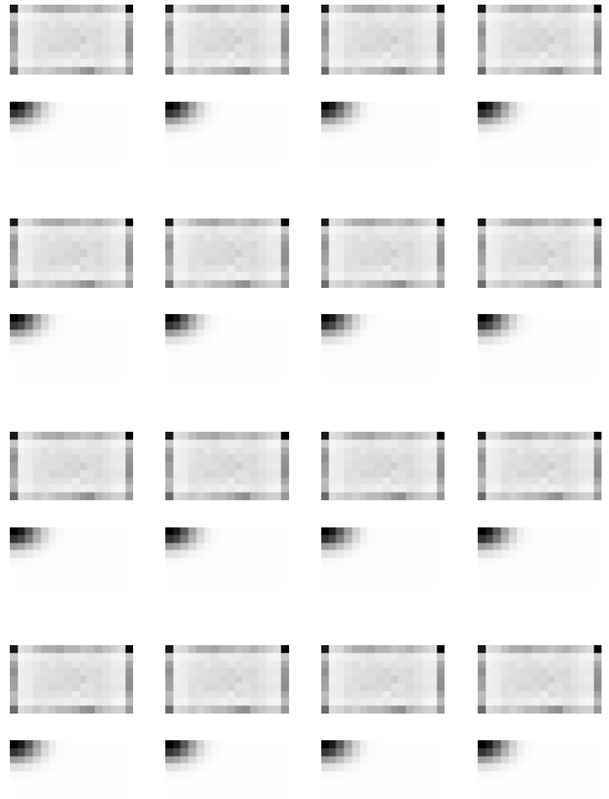
```
for t in (1 to iterations):
  state = newState()
  for ex in examples:
    bestMatch = closest(som_{t-1}, ex)
    hood = neighborhood(bestMatch, sigma(t))
    state.matches += ex * hood
    state.hoods += hood
  som_t = newSOM(state.matches / state.hoods)
```

Each ~~batch~~ partition produces a model that can be averaged with other models



Implementing atop RDDs

```
for t in (1 to iterations):
  state = newState()
  for ex in examples:
    bestMatch = closest(som_{t-1}, ex)
    hood = neighborhood(bestMatch, sigma(t))
    state.matches += ex * hood
    state.hoods += hood
  som_t = newSOM(state.matches / state.hoods)
```



This won't always work!

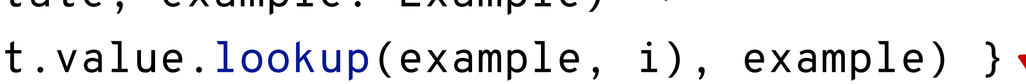
An implementation template

```
var nextModel = initialModel
for (int i = 0; i < iterations; i++) {
  val current = sc.broadcast(nextModel)
  val newState = examples.aggregate(ModelState.empty()) {
    { case (state: ModelState, example: Example) =>
      state.update(current.value.lookup(example, i), example) }
    { case (s1: ModelState, s2: ModelState) => s1.combine(s2) }
  }
  nextModel = modelFromState(newState)
  current.unpersist
}
```

An implementation template

```
var nextModel = initialModel
for (int i = 0; i < iterations; i++) {
  val current = sc.broadcast(nextModel)
  val newState = examples.aggregate(ModelState.empty()) {
    { case (state: ModelState, example: Example) =>
      state.update(current.value.lookup(example, i), example) }
    { case (s1: ModelState, s2: ModelState) => s1.combine(s2) }
  }
  nextModel = modelFromState(newState)
  current.unpersist
}
```

"fold": update the state for this partition with a single new example



An implementation template

```
var nextModel = initialModel
for (int i = 0; i < iterations; i++) {
  val current = sc.broadcast(nextModel)
  val newState = examples.aggregate(ModelState.empty()) {
    { case (state: ModelState, example: Example) =>
      state.update(current.value.lookup(example, i), example) }
    { case (s1: ModelState, s2: ModelState) => s1.combine(s2) }
  }
  nextModel = modelFromState(newState)
  current.unpersist
}
```

"reduce": combine the states from two partitions

An implementation template

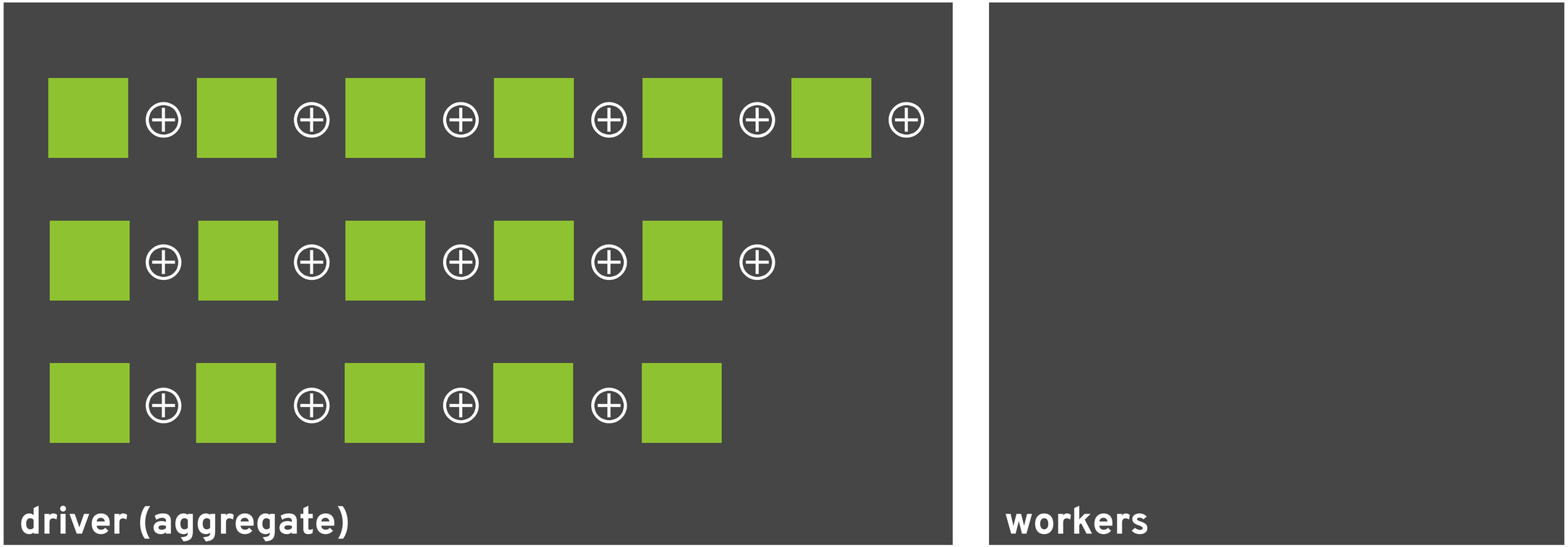
```
var nextModel = initialModel
for (int i = 0; i < iterations; i++) {
  val current = sc.broadcast(nextModel)
  val newState = examples.aggregate(ModelState.empty()) {
    { case (state: ModelState, example: Example) =>
      state.update(current.value.lookup(example, i), example) }
    { case (s1: ModelState, s2: ModelState) => s1.combine(s2) }
  }
  nextModel = modelFromState(newState)
  current.unpersist
}
```

broadcast the current working model for this iteration

remove the stale broadcasted model

Implementing on RDDs

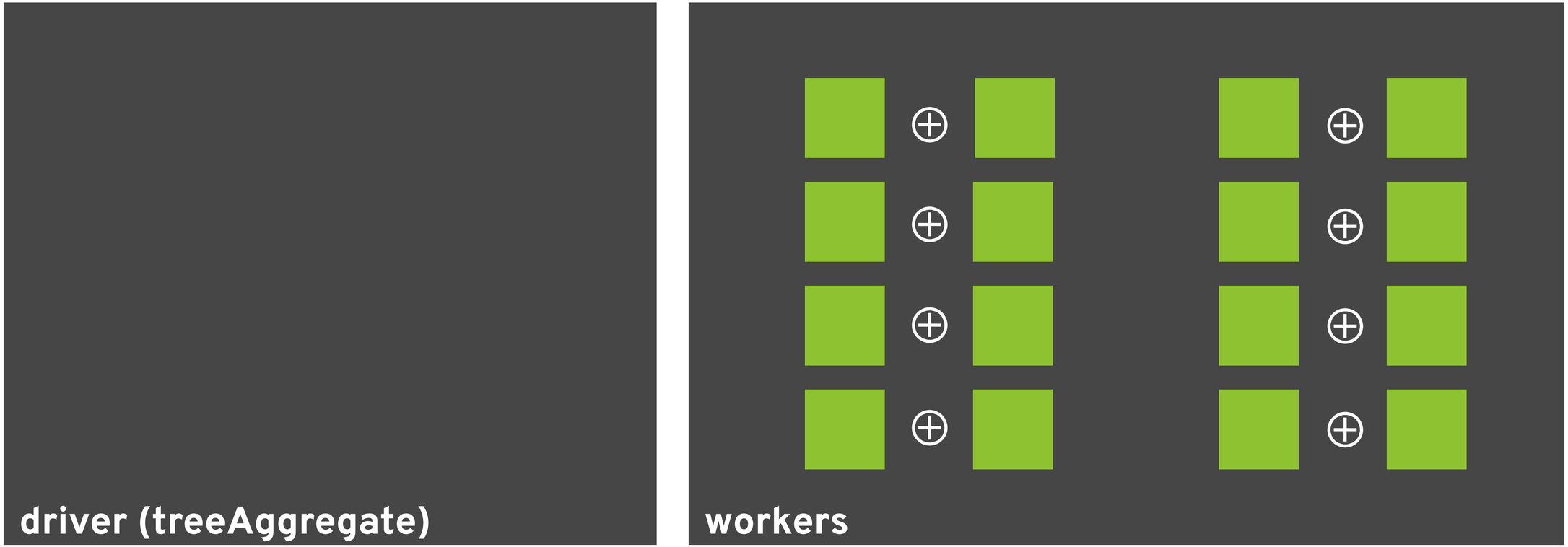
Implementing on RDDs



Implementing on RDDs

Implementing on RDDs

Implementing on RDDs



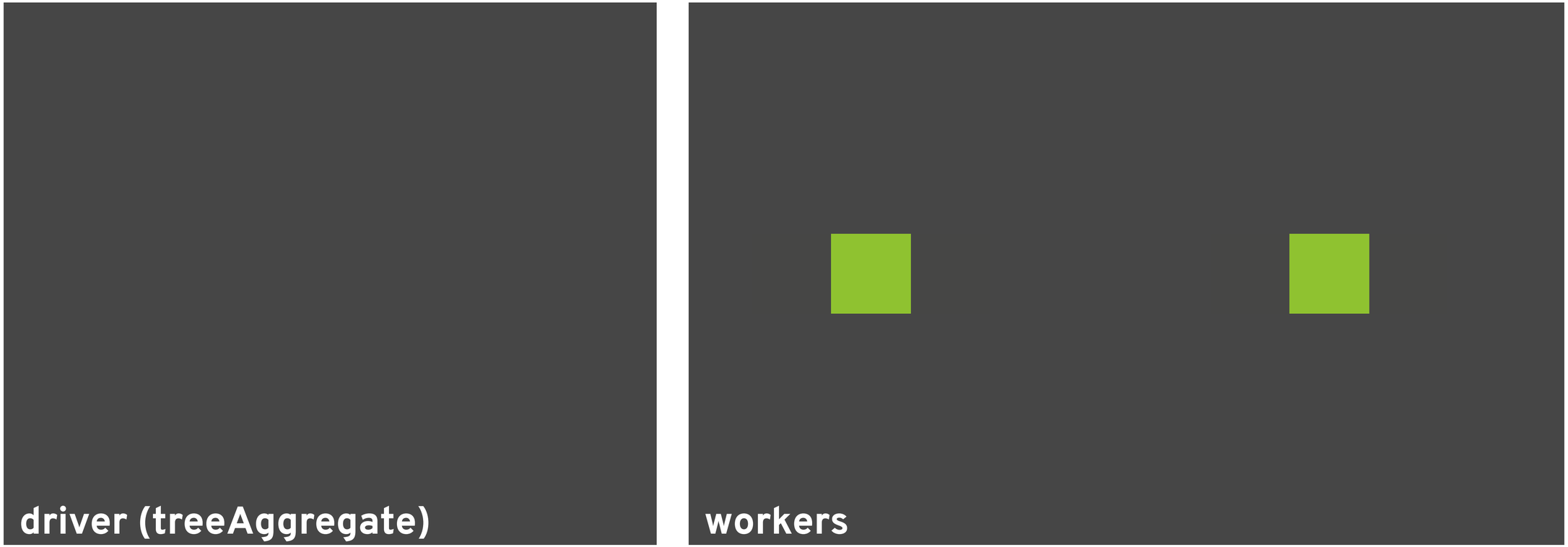
Implementing on RDDs

Implementing on RDDs

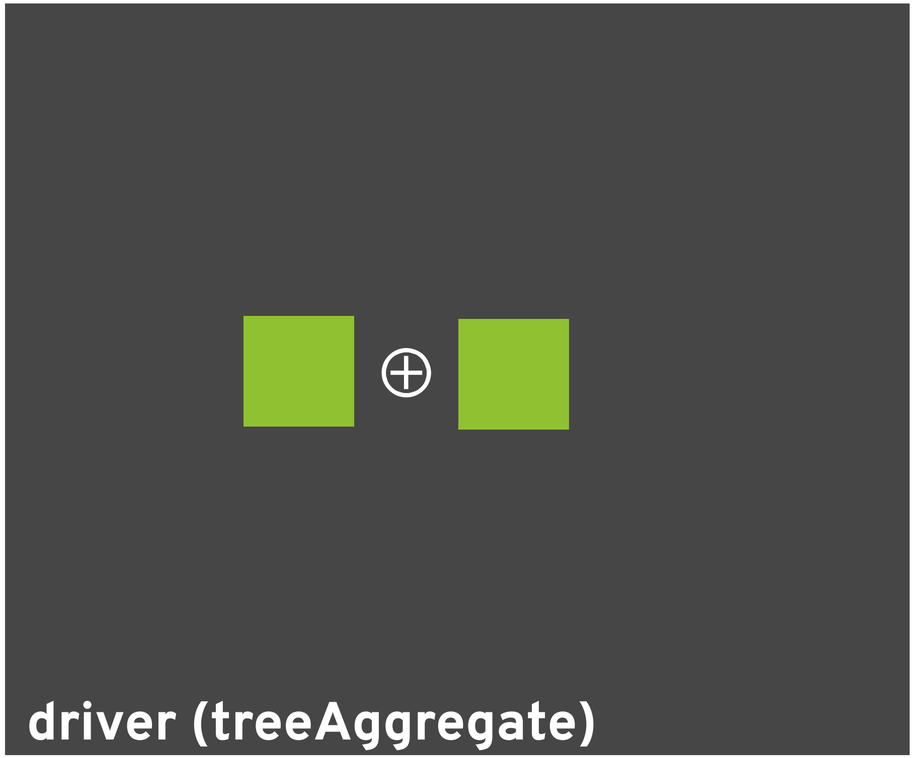
Implementing on RDDs

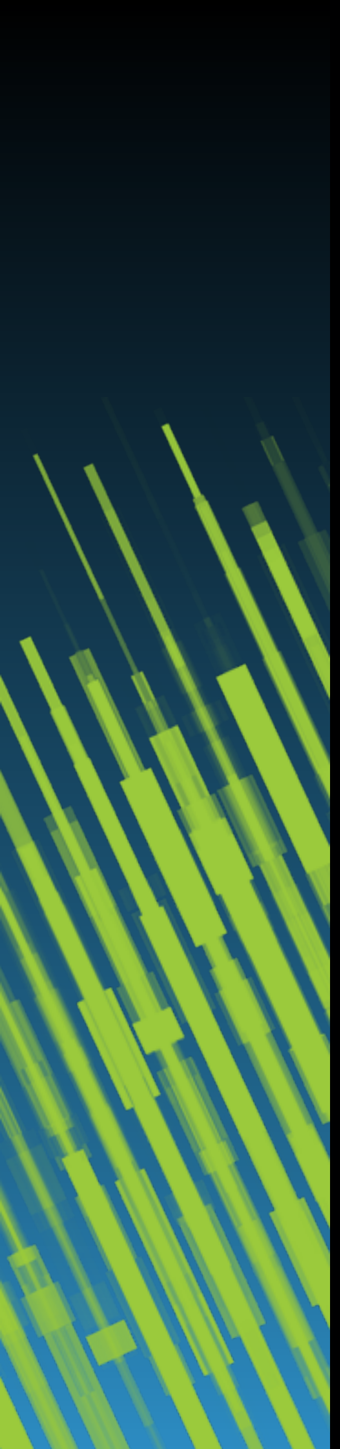
Implementing on RDDs

Implementing on RDDs



Implementing on RDDs





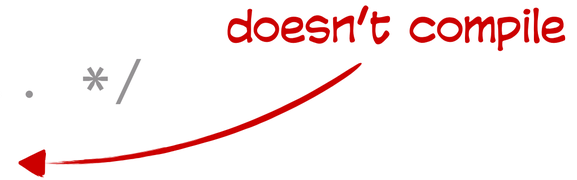
Beyond the RDD: Data frames and ML Pipelines

RDDs: some good parts

```
val rdd: RDD[String] = /* ... */
rdd.map(_ * 3.0).collect()
```

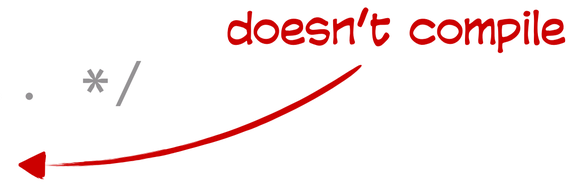
RDDs: some good parts

```
val rdd: RDD[String] = /* ... */
rdd.map(_ * 3.0).collect()
```



RDDs: some good parts

```
val rdd: RDD[String] = /* ... */
rdd.map(_ * 3.0).collect()
```




```
val df: DataFrame = /* data frame with one String-valued column */
df.select($"_1" * 3.0).show()
```

RDDs: some good parts

```
val rdd: RDD[String] = /* ... */
rdd.map(_ * 3.0).collect()
```

doesn't compile

```
val df: DataFrame = /* data frame with one String-valued column */
df.select($"_1" * 3.0).show()
```

crashes at runtime

RDDs: some good parts

```
rdd.map {
  vec => (vec, model.value.closestWithSimilarity(vec))
}
```

RDDs: some good parts

```
rdd.map {
  vec => (vec, model.value.closestWithSimilarity(vec))
}
```

```
val predict = udf ((vec: SV) =>
  model.value.closestWithSimilarity(vec))

df.withColumn($"predictions", predict($"features"))
```

RDDs versus query planning

```
val numbers1 = sc.parallelize(1 to 100000000)
val numbers2 = sc.parallelize(1 to 1000000000)
numbers1.cartesian(numbers2)
     .map((x, y) => (x, y, expensive(x, y)))
     .filter((x, y, _) => isPrime(x), isPrime(y))
```

RDDs versus query planning

```
val numbers1 = sc.parallelize(1 to 100000000)
val numbers2 = sc.parallelize(1 to 1000000000)
numbers1.filter(isPrime(_))
    .cartesian(numbers2.filter(isPrime(_)))
    .map((x, y) => (x, y, expensive(x, y)))
```

RDDs and the Java heap

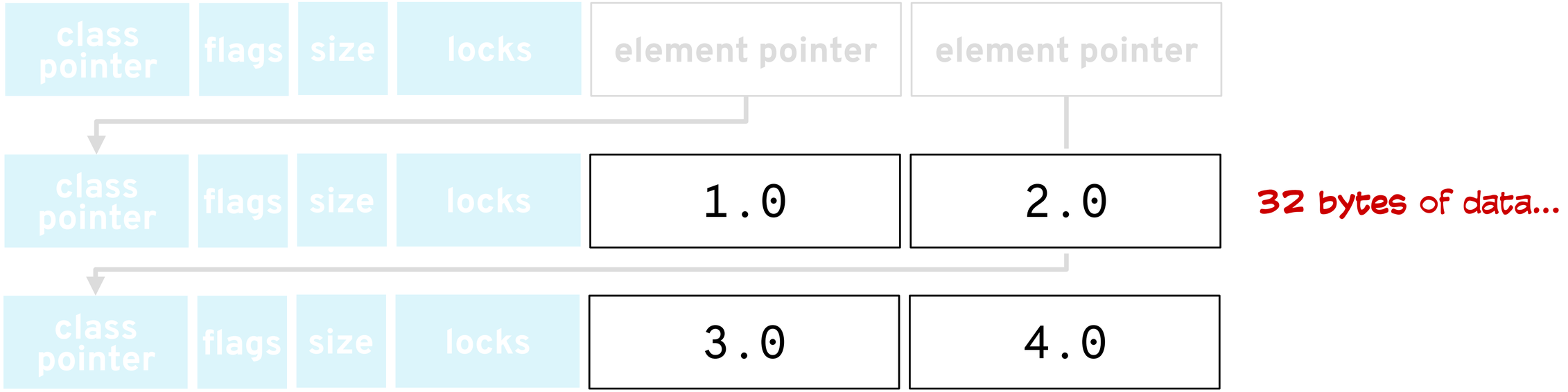
```
val mat = Array(Array(1.0, 2.0), Array(3.0, 4.0))
```

RDDs and the Java heap

```
val mat = Array(Array(1.0, 2.0), Array(3.0, 4.0))
```

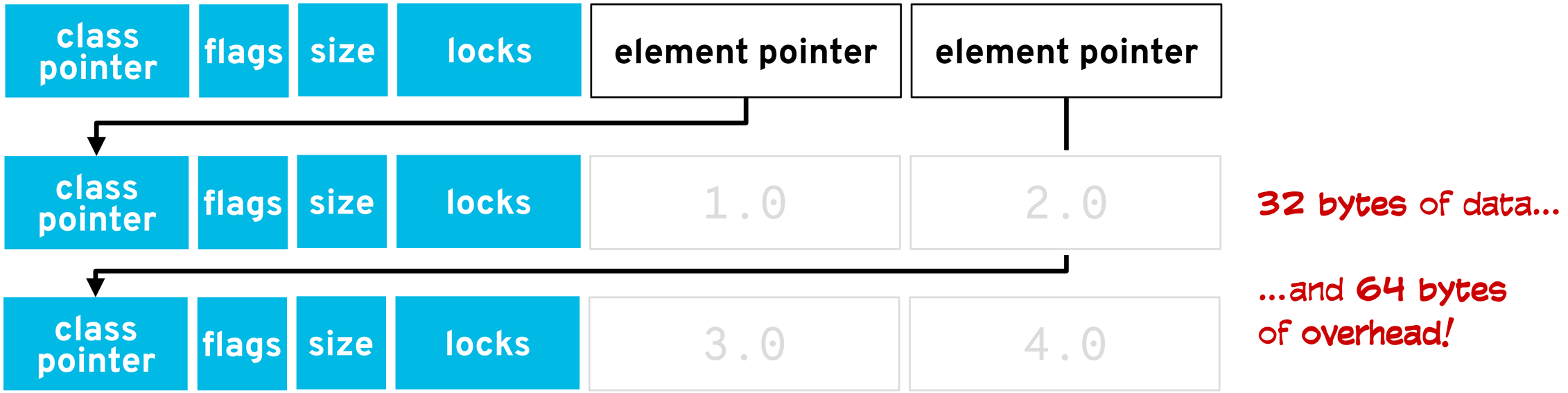
RDDs and the Java heap

```
val mat = Array(Array(1.0, 2.0), Array(3.0, 4.0))
```



RDDs and the Java heap

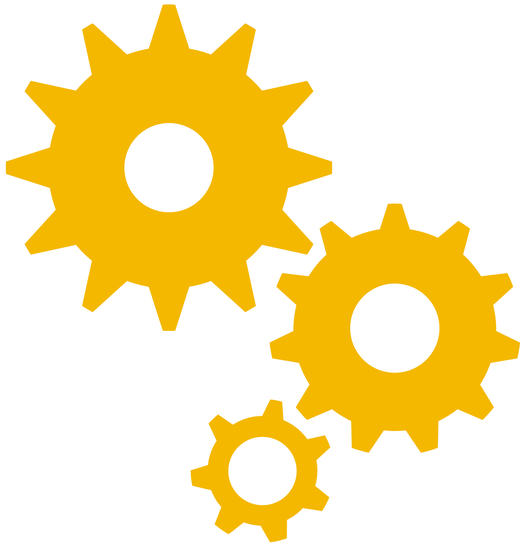
```
val mat = Array(Array(1.0, 2.0), Array(3.0, 4.0))
```



ML pipelines: a quick example

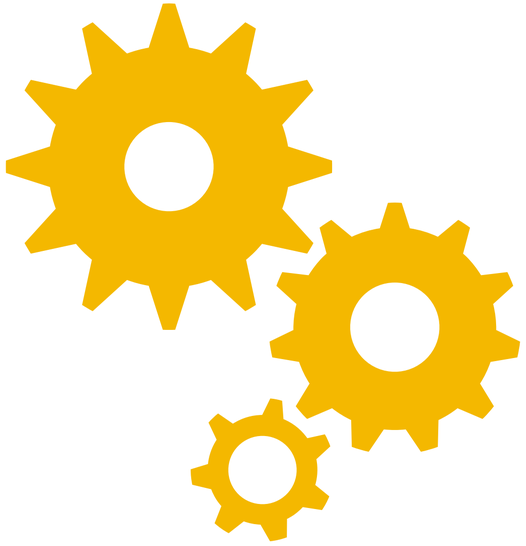
```
from pyspark.ml.clustering import KMeans

K, SEED = 100, 0xdea110c8

randomDF = make_random_df()

kmeans = KMeans().setK(K).setSeed(SEED).setFeaturesCol("features")
model = kmeans.fit(randomDF)
withPredictions = model.transform(randomDF).select("x", "y", "prediction")
```

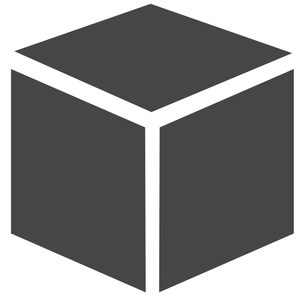
Working with ML pipelines



```
estimator.fit(df)
```

Working with ML pipelines

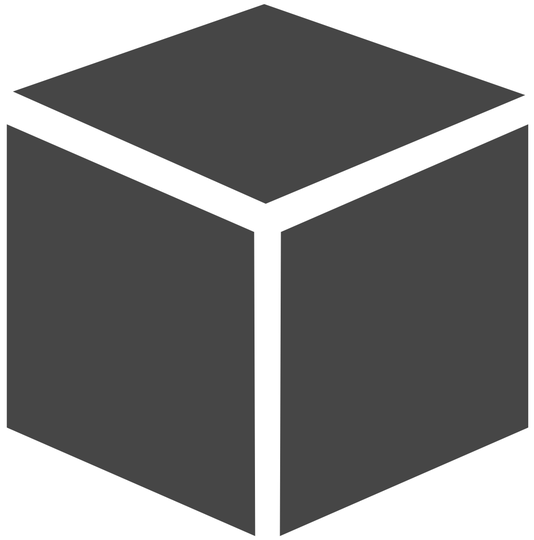




```
estimator.fit(df)
```

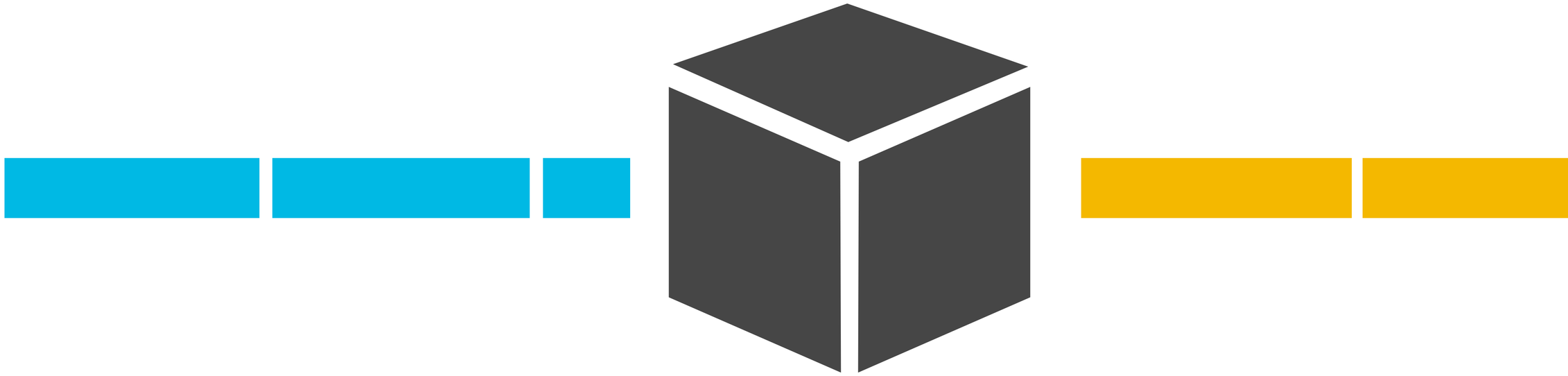
```
model.transform(df)
```

Working with ML pipelines



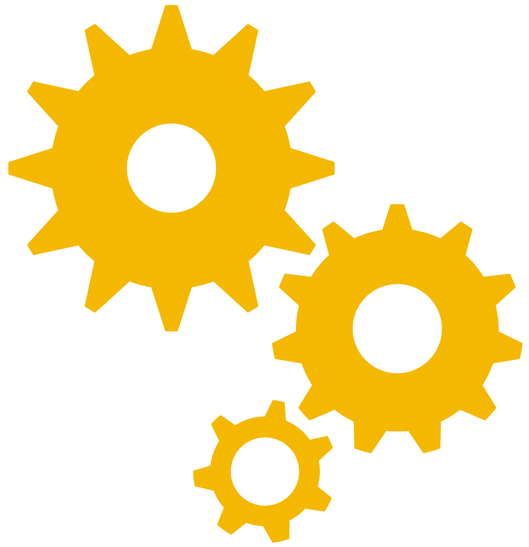
```
model.transform(df)
```

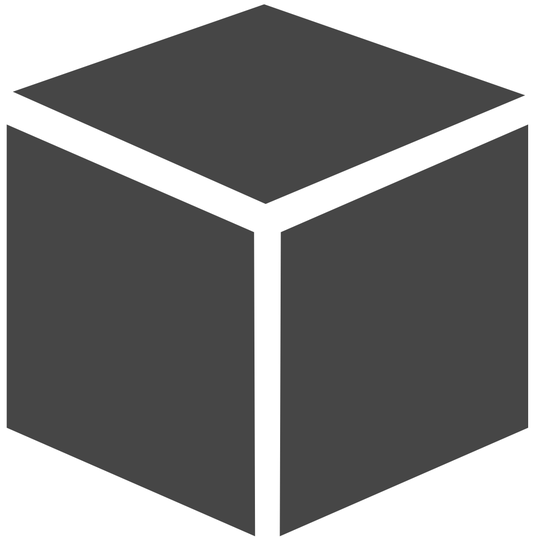
Working with ML pipelines



```
model.transform(df)
```

Working with ML pipelines

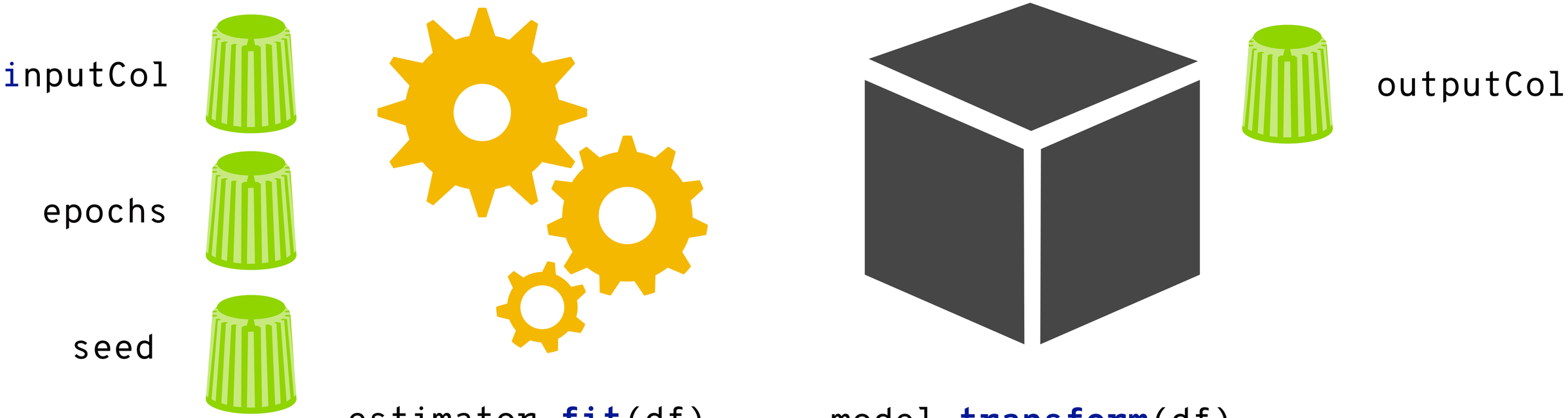




`estimator.fit(df)`

`model.transform(df)`

Working with ML pipelines



Defining parameters

```
private[som] trait SOMParams extends Params
    with DefaultParamsWritable {
  final val x: IntParam =
    new IntParam(this, "x", "width of self-organizing map (>= 1)",
                         ParamValidators.gtEq(1))

  final def getX: Int = $(x)

  final def setX(value: Int): this.type = set(x, value)
  // ...
```

Defining parameters

```
private[som] trait SOMParams extends Params
    with DefaultParamsWritable {
  final val x: IntParam =
    new IntParam(this, "x", "width of self-organizing map (>= 1)",
                         ParamValidators.gtEq(1))

  final def getX: Int = $(x)

  final def setX(value: Int): this.type = set(x, value)
  // ...
```

Defining parameters

```
private[som] trait SOMParams extends Params
    with DefaultParamsWritable {
  final val x: IntParam =
    new IntParam(this, "x", "width of self-organizing map (>= 1)",
                         ParamValidators.gtEq(1))

  final def getX: Int = $(x)

  final def setX(value: Int): this.type = set(x, value)
  // ...
```

Defining parameters

```
private[som] trait SOMParams extends Params
    with DefaultParamsWritable {
  final val x: IntParam =
    new IntParam(this, "x", "width of self-organizing map (>= 1)",
                         ParamValidators.gtEq(1))

  final def getX: Int = $(x)

  final def setX(value: Int): this.type = set(x, value)
  // ...
```

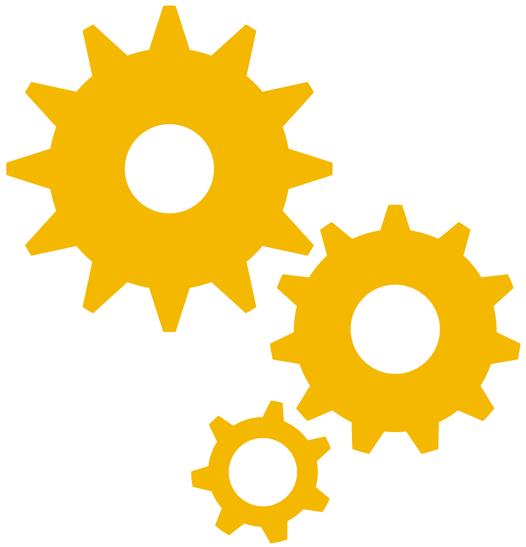
Defining parameters

```
private[som] trait SOMParams extends Params
    with DefaultParamsWritable {
  final val x: IntParam =
    new IntParam(this, "x", "width of self-organizing map (>= 1)",
                         ParamValidators.gtEq(1))

  final def getX: Int = $(x)

  final def setX(value: Int): this.type = set(x, value)
  // ...
```

Don’t repeat yourself

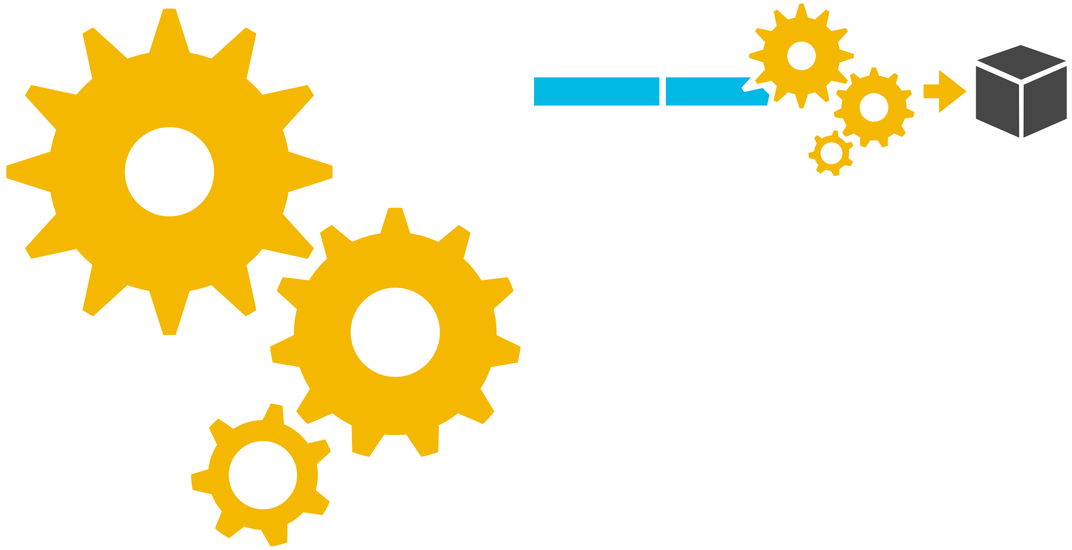
```
/**
 * Common params for KMeans and KMeansModel
 */
private[clustering] trait KMeansParams extends Params
    with HasMaxIter with HasFeaturesCol
    with HasSeed with HasPredictionCol with HasTol { /* ... */ }
```

Estimators and transformers



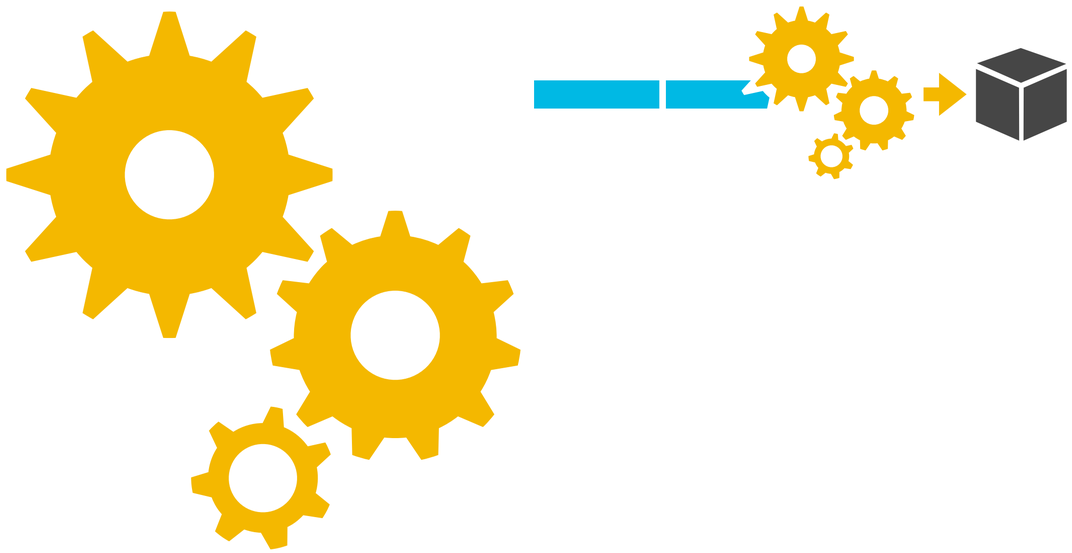
```
estimator.fit(df)
```

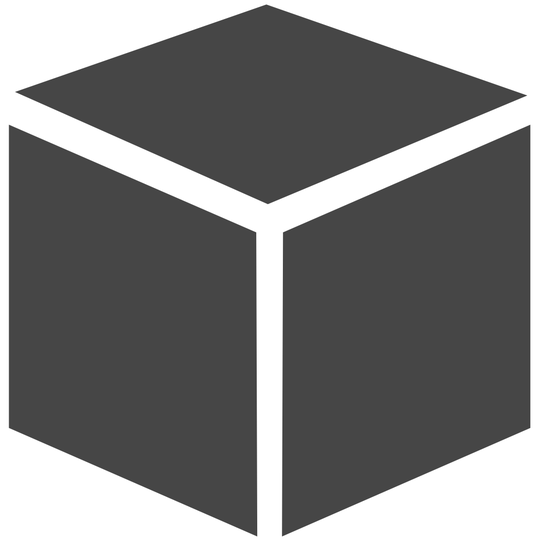
Estimators and transformers



```
estimator.fit(df)
```

Estimators and transformers

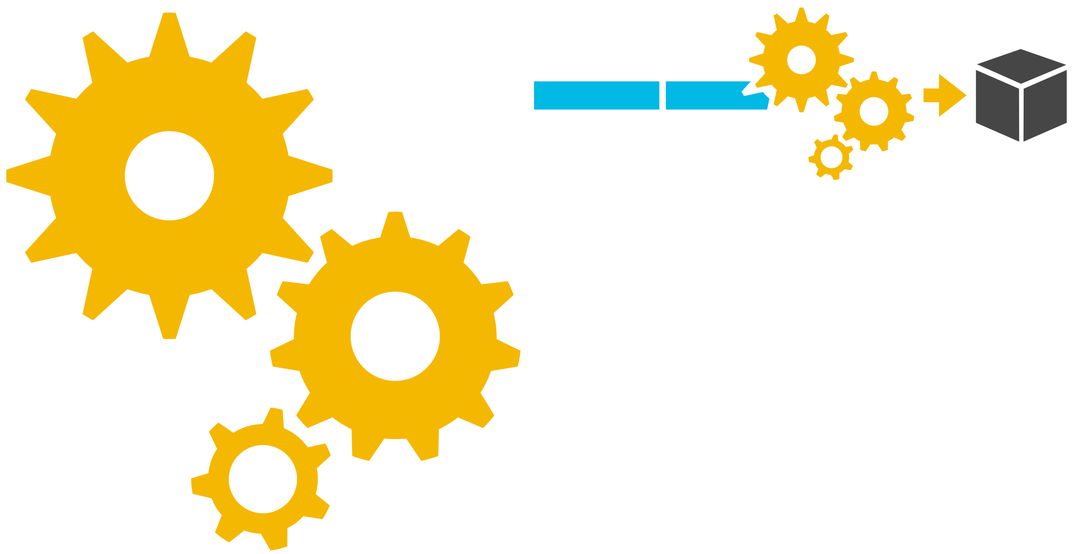


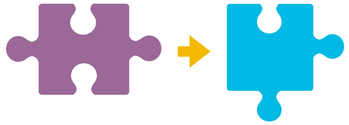


`estimator.fit(df)`

`model.transform(df)`

Estimators and transformers

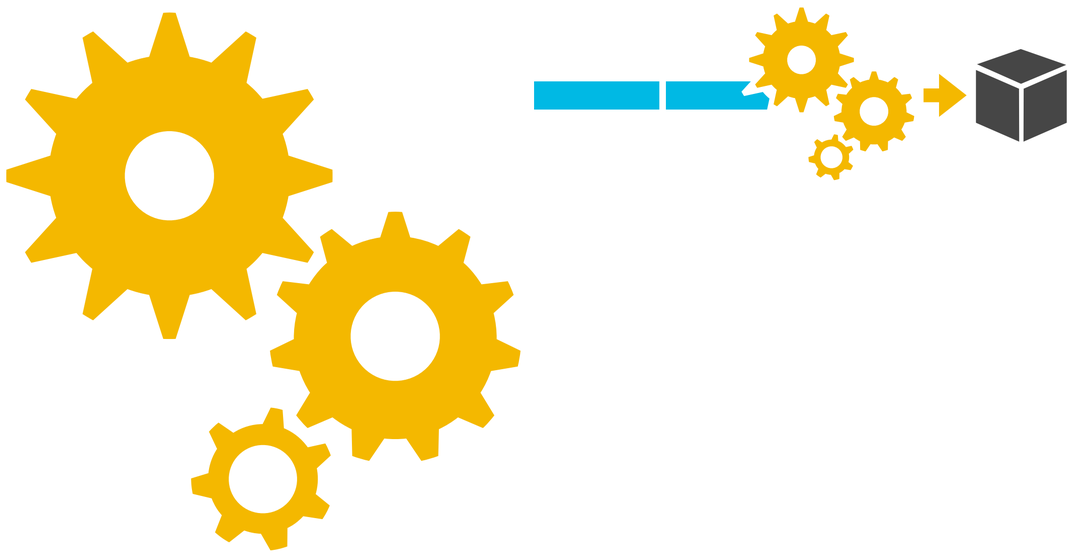


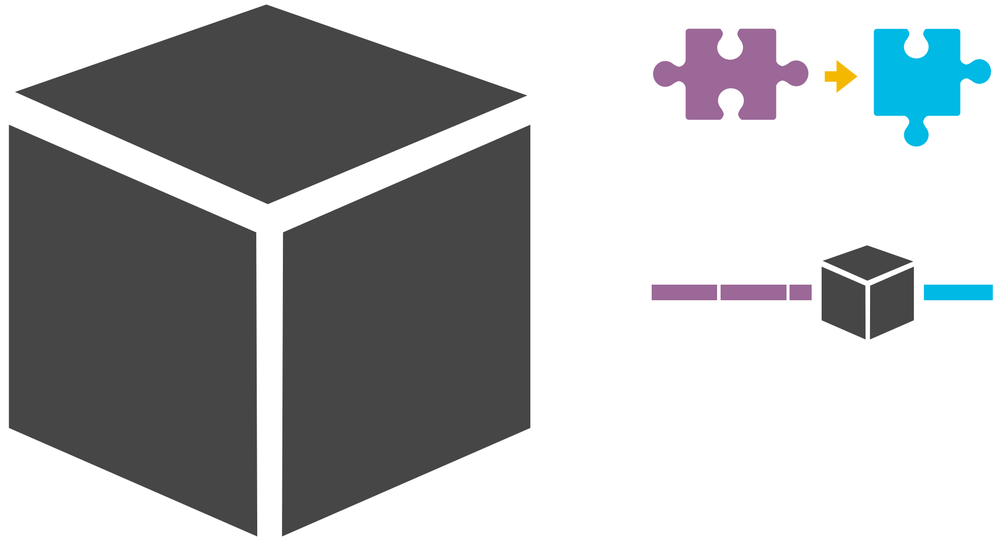


`estimator.fit(df)`

`model.transform(df)`

Estimators and transformers





`estimator.fit(df)`

`model.transform(df)`

Validate and transform at once

```
def transformSchema(schema: StructType):
    StructType = {
  // check that the input columns exist...
  // ...and are the proper type
  // ...and that the output columns don't exist
  // ...and then make a new schema
}
```

Validate and transform at once

```
def transformSchema(schema: StructType):
    StructType = {
  // check that the input columns exist…
  require(schema.fieldNames.contains($(featuresCol)))
  // ...and are the proper type
  // ...and that the output columns don’t exist
  // ...and then make a new schema
}
```

Validate and transform at once

```
def transformSchema(schema: StructType):
    StructType = {
  // check that the input columns exist...
  // ...and are the proper type
  schema($(featuresCol)) match {
    case sf: StructField => require(sf.dataType.equals(VectorType))
  }
  // ...and that the output columns don't exist
  // ...and then make a new schema
}
```

Validate and transform at once

```
def transformSchema(schema: StructType):
    StructType = {
  // check that the input columns exist…
  // ...and are the proper type
  // ...and that the output columns don’t exist
  require(!schema.fieldNames.contains($(predictionCol)))
  require(!schema.fieldNames.contains($(similarityCol)))
  // ...and then make a new schema
}
```

Validate and transform at once

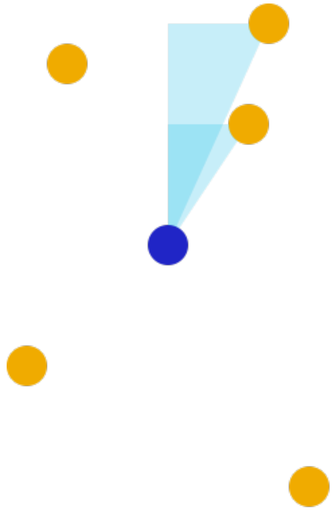
```
def transformSchema(schema: StructType):
    StructType = {
  // check that the input columns exist…
  // ...and are the proper type
  // ...and that the output columns don’t exist
  // ...and then make a new schema
  schema.add($(predictionCol), "int")
        .add($(similarityCol), "double")
}
```

Training on data frames

```
def fit(examples: DataFrame) = {
  import examples.sparkSession.implicits._
  import org.apache.spark.ml.linalg.{Vector=>SV}

  val dfexamples = examples.select($(exampleCol)).rdd.map {
    case Row(sv: SV) => sv
  }
  /* construct a model object with the result of training */
  new SOMModel(train(dfexamples, $(x), $(y)))
}
```

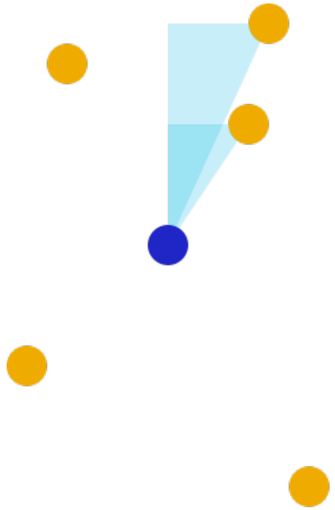
Practical considerations

Improve serial execution times



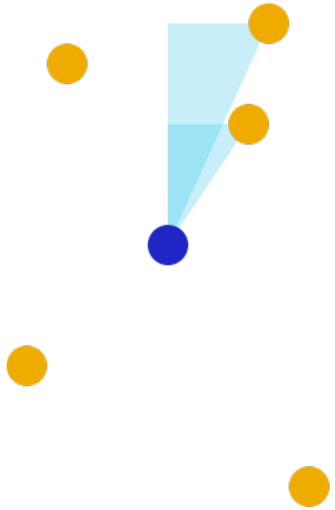
$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix}$$

Improve serial execution times

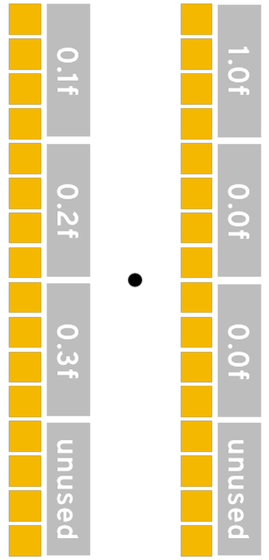


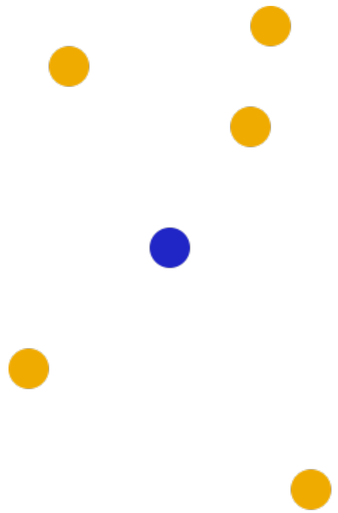
$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix}$$

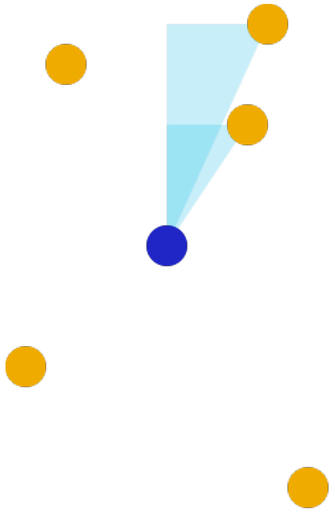
Improve serial execution times

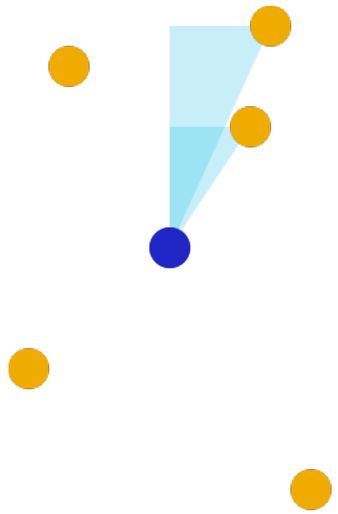


$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix}$$

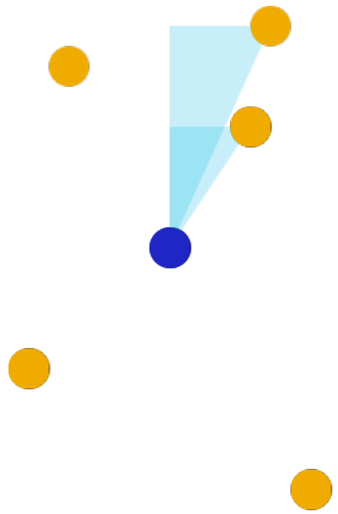




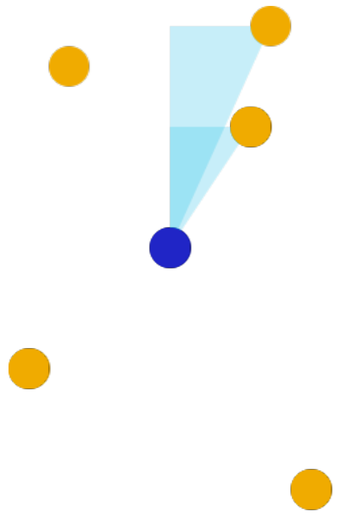




$$\frac{p \bullet q}{||p|| \bullet ||q||}$$



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



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

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \bullet \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \bullet \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \bullet \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix}$$

```
libraryDependencies +=
  "org.scalanlp" %% "breeze-natives" % "0.13.1"
```

```
val vec = Array[Double](/* ... */)
```

```
val vec = Array[Double](/* ... */)
```

```
val vec = Array[Double](/* ... */)
```

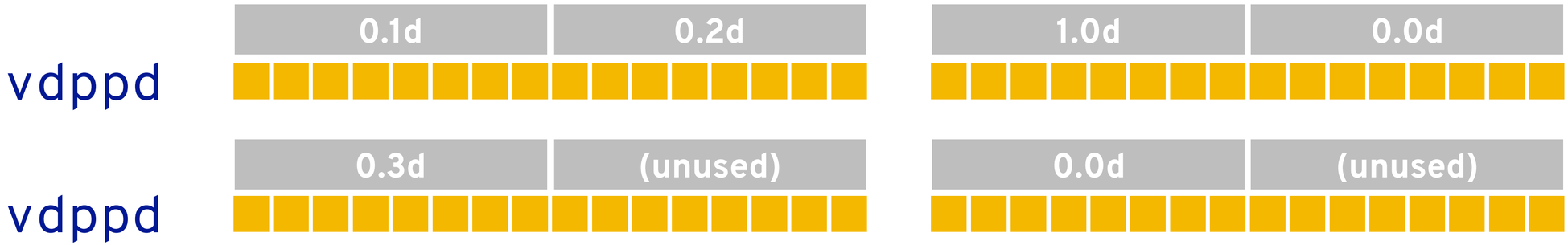
```
def dot[S](a: Array[S], b: Array[S])
  (implicit num: Numeric[S]): S = {
  import num._
  (0 until a.length).foldLeft(num.zero)({
    (acc, i) => acc + a(i) * b(i)
  })
}
```

```
val vec = Array[Double](/* ... */)
```

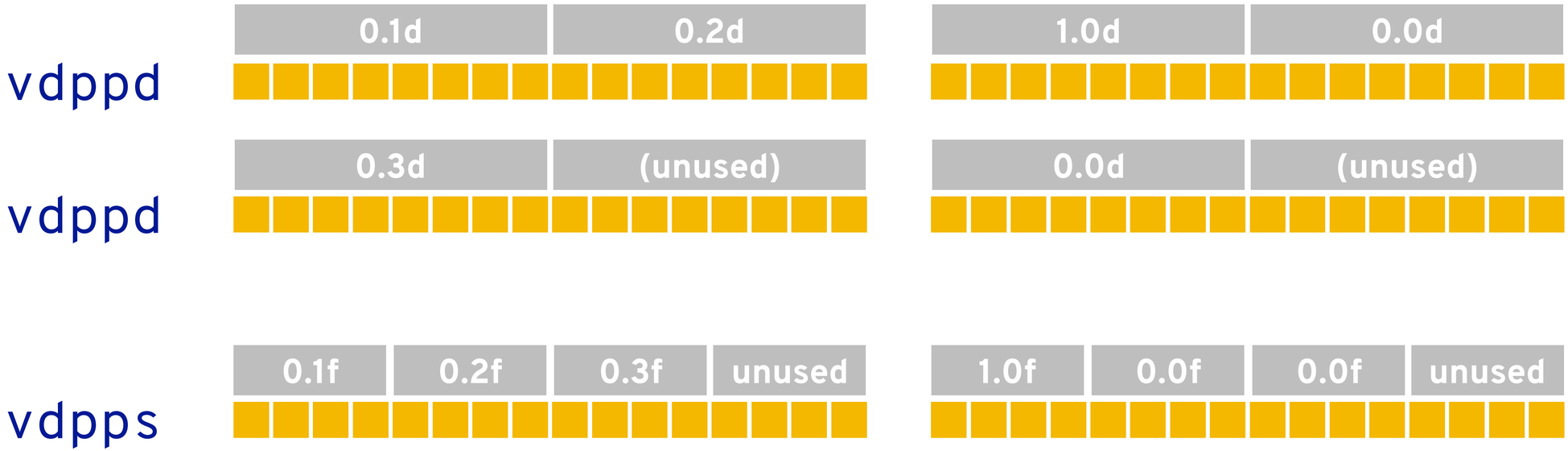
```
def dot[S](a: Array[S], b: Array[S])
  (implicit num: Numeric[S]): S = {
  import num._
  (0 until a.length).foldLeft(num.zero)({
    (acc, i) => acc + a(i) * b(i)
  })
}
```

```
dot(Array(0.1d, 0.2d, 0.3d), Array(1.0d, 0.0d, 0.0d))
dot(Array(0.1f, 0.2f, 0.3f), Array(1.0f, 0.0f, 0.0f))
```

```scala
val vec = Array[Double](/* ... */)

def dot[S](a: Array[S], b: Array[S])
  (implicit num: Numeric[S]): S = {
  import num._
  (0 until a.length).foldLeft(num.zero)({
    (acc, i) => acc + a(i) * b(i)
  })
}
```



```scala
val vec = Array[Double](/* ... */)

def dot[S](a: Array[S], b: Array[S])
  (implicit num: Numeric[S]): S = {
  import num._
  (0 until a.length).foldLeft(num.zero)({
    (acc, i) => acc + a(i) * b(i)
  })
}
```



KEY TAKEAWAYS

$$\lim_{s_p \to \infty} S_o = \frac{1}{1 - p}$$

$$a \oplus b = b \oplus a$$
$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$

SGD L-BGFS

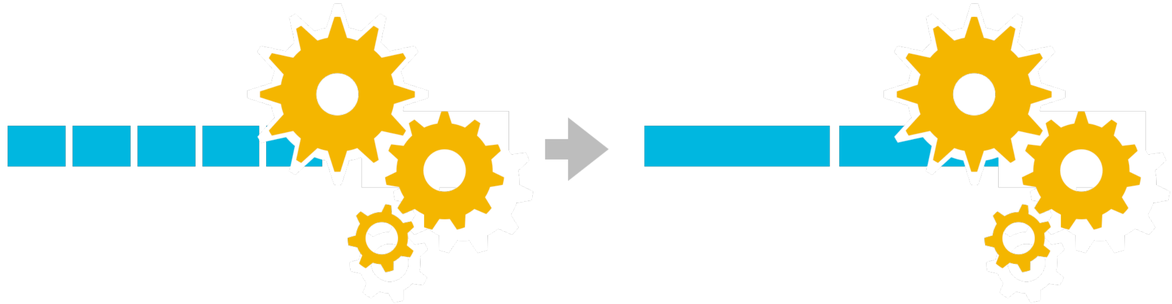
$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix}$$


```
dot(Array(0.1d, 0.2d, 0.3d), Array(1.0d, 0.0d, 0.0d))
dot(Array(0.1f, 0.2f, 0.3f), Array(1.0f, 0.0f, 0.0f))
```


estimator.fit(df)

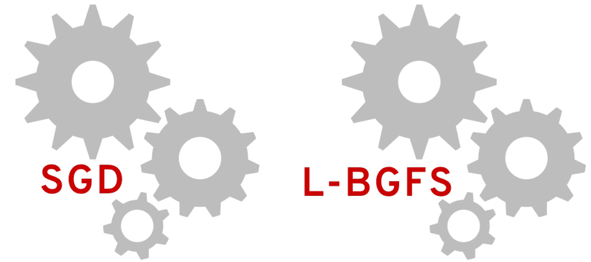
model.transform(df)

$$\lim_{s_p \to \infty} S_o = \frac{1}{1 - p}$$



$$a \oplus b = b \oplus a$$

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$



estimator.fit(df)

model.transform(df)

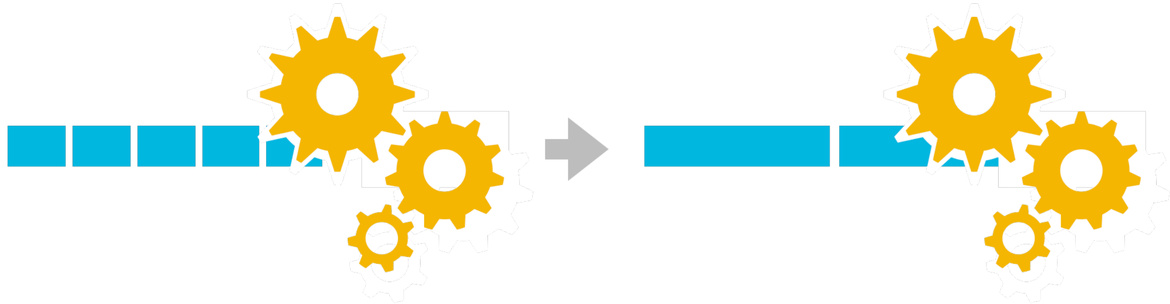
$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$

```
dot(Array(0.1d, 0.2d, 0.3d), Array(1.0d, 0.0d, 0.0d))
dot(Array(0.1f, 0.2f, 0.3f), Array(1.0f, 0.0f, 0.0f))
```

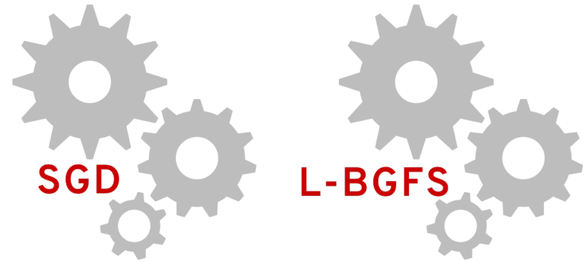
vdpps

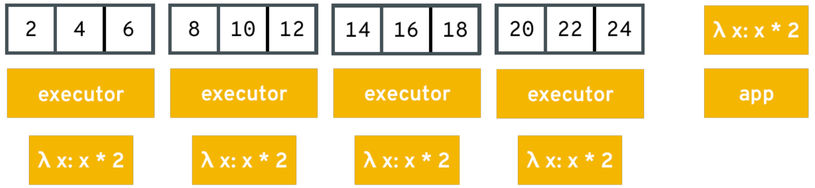
$$\lim_{s_p \to \infty} S_o = \frac{1}{1 - p}$$

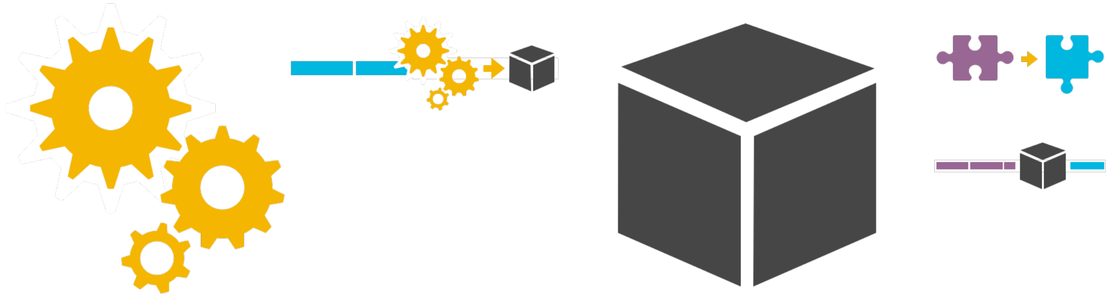


$$a \oplus b = b \oplus a$$

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$





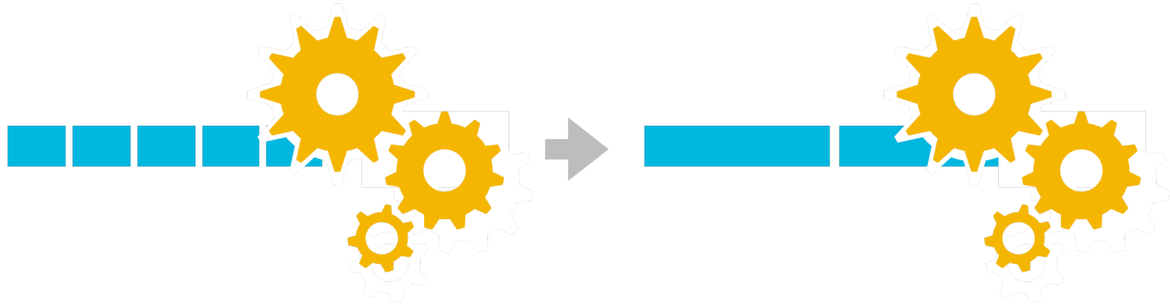


estimator.**fit**(df)

model.**transform**(df)

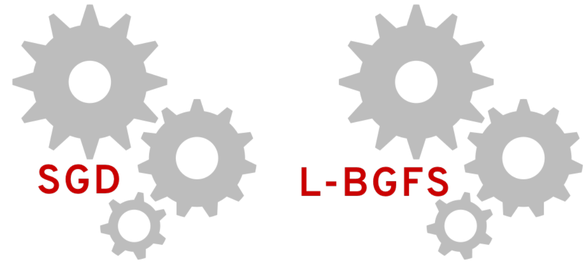

```
dot(Array(0.1d, 0.2d, 0.3d), Array(1.0d, 0.0d, 0.0d))
dot(Array(0.1f, 0.2f, 0.3f), Array(1.0f, 0.0f, 0.0f))
```

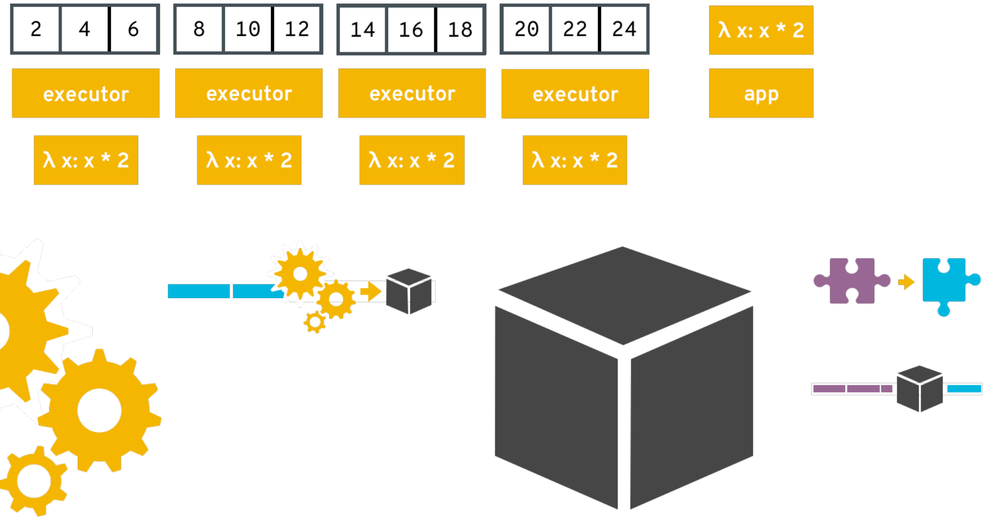
$$\lim_{s_p \to \infty} S_o = \frac{1}{1 - p}$$



$$a \oplus b = b \oplus a$$

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$





estimator.**fit**(df)

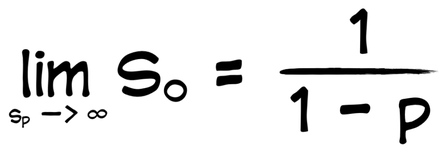
model.**transform**(df)

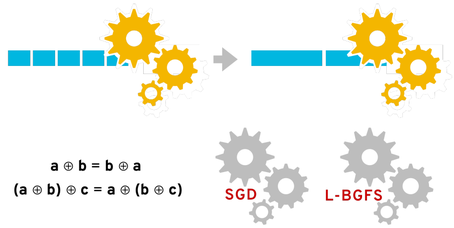
$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix}$$

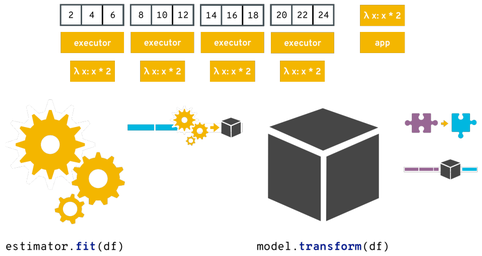
$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0.1 & 0.7 & 0.2 \end{bmatrix}$$

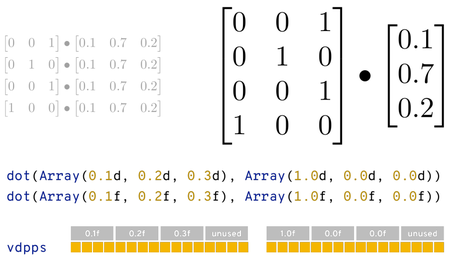
```
dot(Array(0.1d, 0.2d, 0.3d), Array(1.0d, 0.0d, 0.0d))
dot(Array(0.1f, 0.2f, 0.3f), Array(1.0f, 0.0f, 0.0f))
```

vdpps









THANKS!

willb@redhat.com • @willb

https://chapeau.freevariable.com

https://radanalytics.io

also: “Spark for Library Developers”

Room 2014 at 5:40 PM today