SPARK SUMMIT

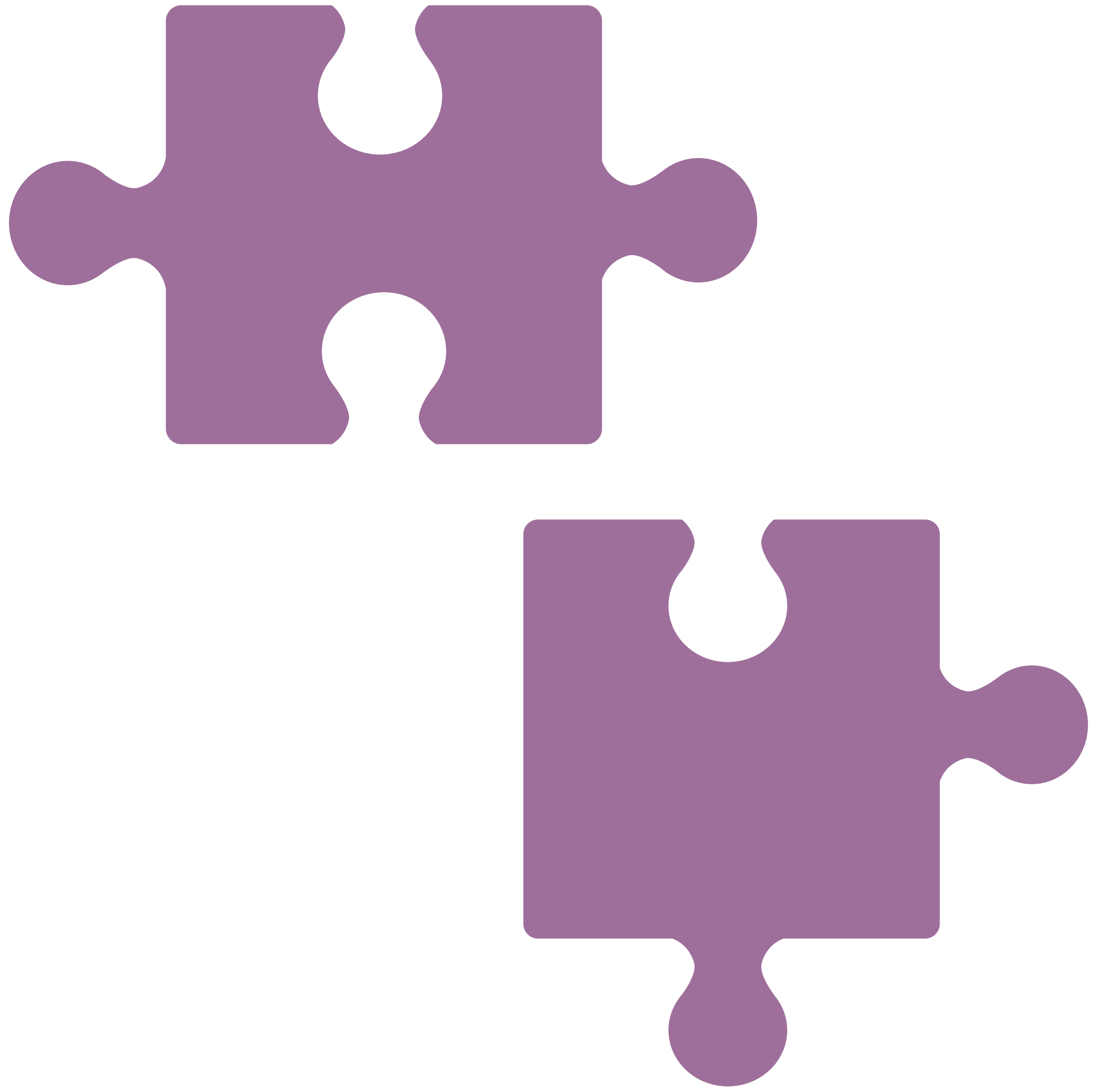
Building machine learning algorithms on Apache Spark

William Benton (@willb)

Red Hat, Inc.

Session hashtag: #EUds5

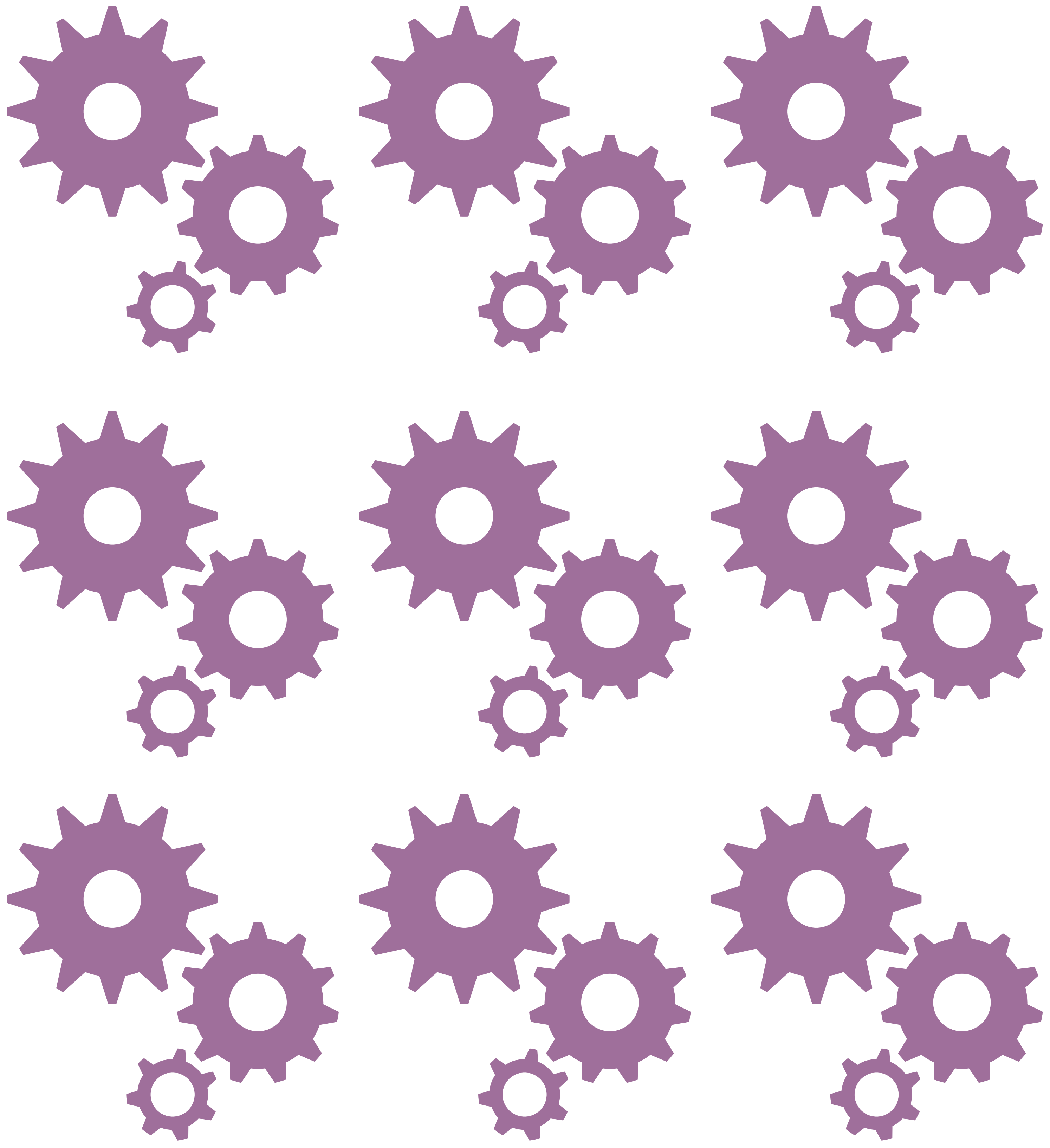
Motivation



Motivation

Motivation

Motivation



Forecast

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

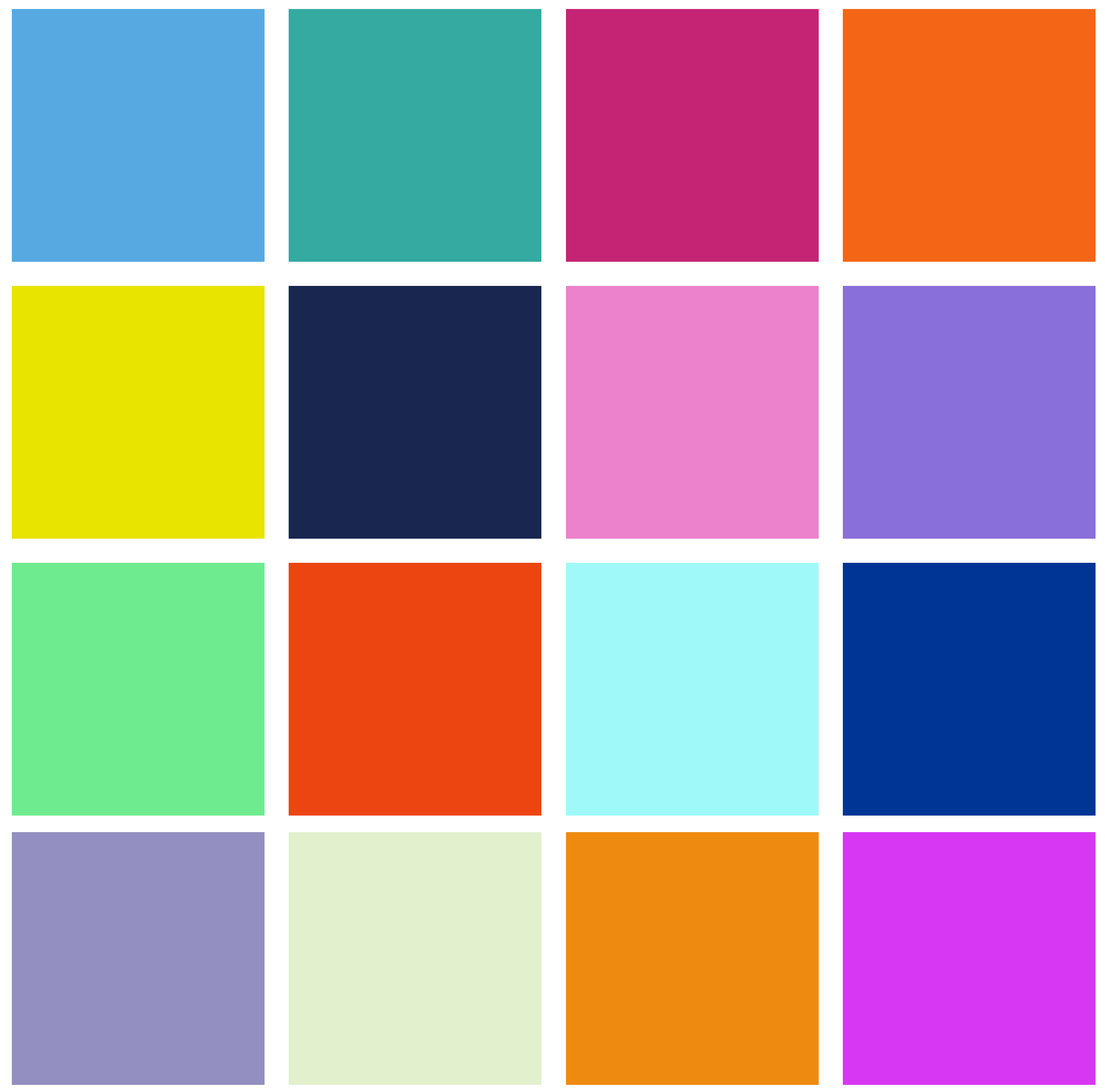
Parallel implementations for **partitioned collections** (in particular, **RDDs**)

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

Practical considerations and **key takeaways**

Introducing self-organizing maps

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(somt, ex)
    for (unit, wt) in neighborhood(bestMatch, sigma(t)):
      somt+1[unit] = somt[unit] + (ex - somt[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 learning rate controls how much closer to the example each unit gets

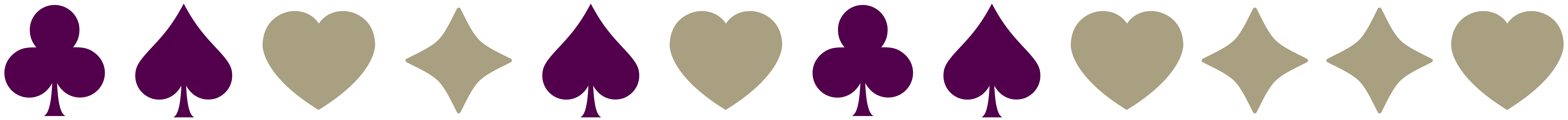
the neighborhood size controls how much of the map around the BMU is affected

Parallel implementations for partitioned collections

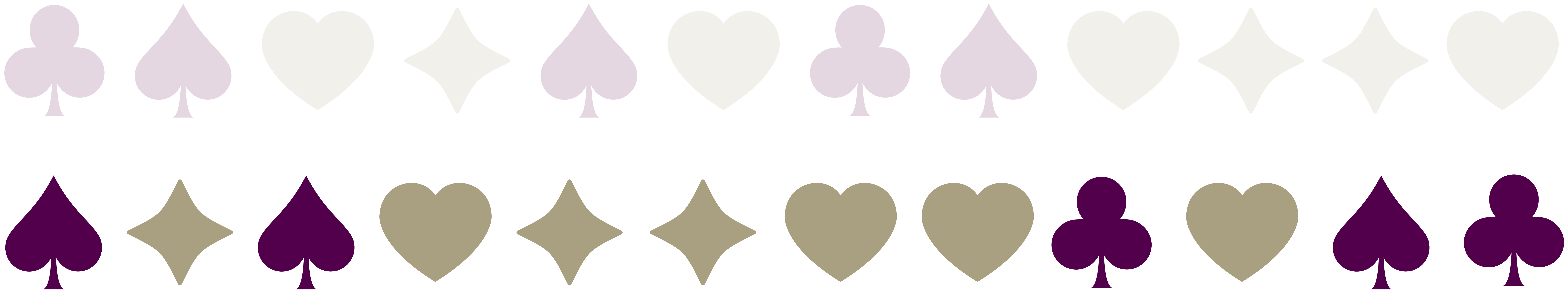
Historical aside: Amdahl's Law

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

What forces serial execution?



What forces serial execution?

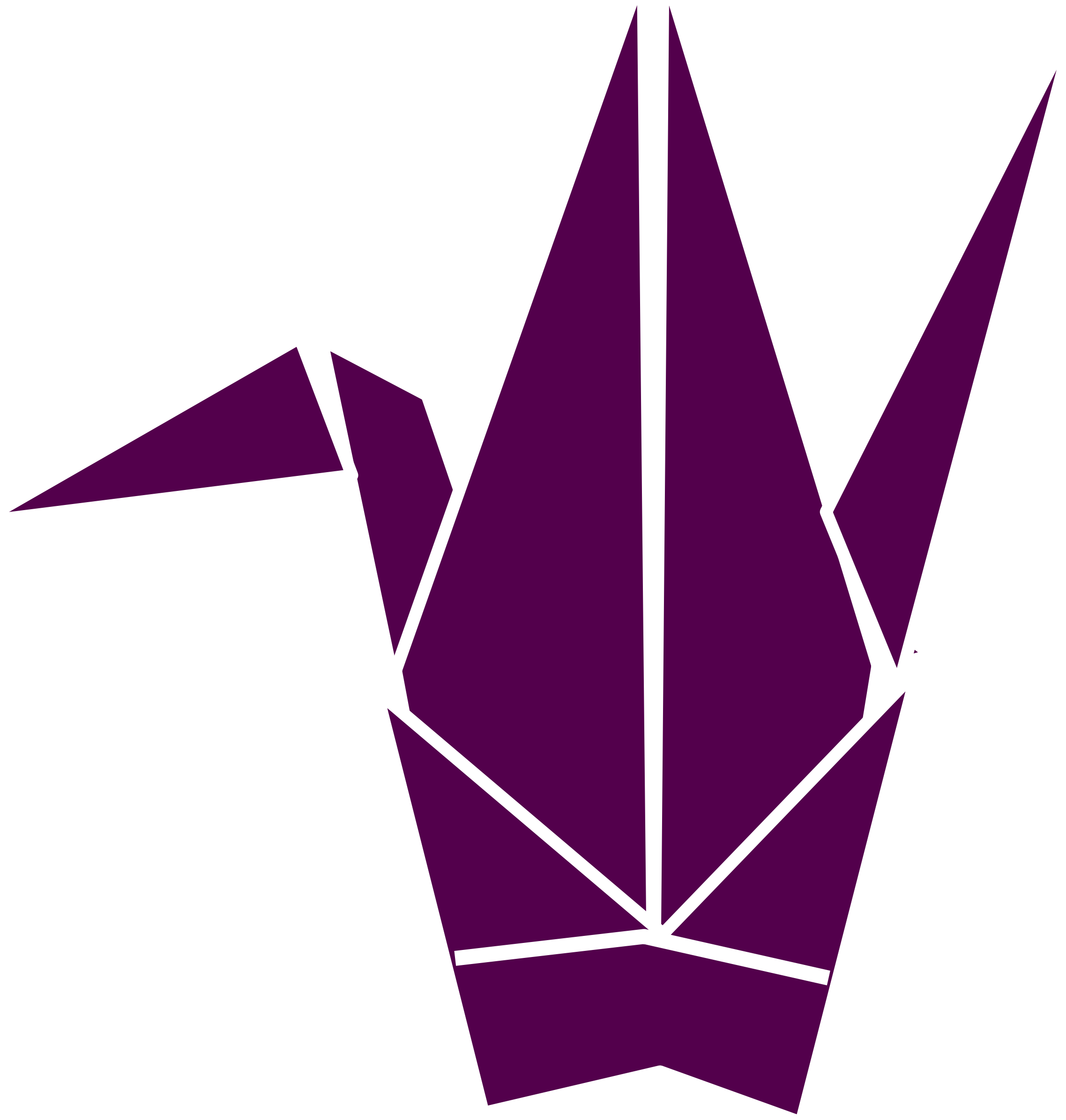


What forces serial execution?

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

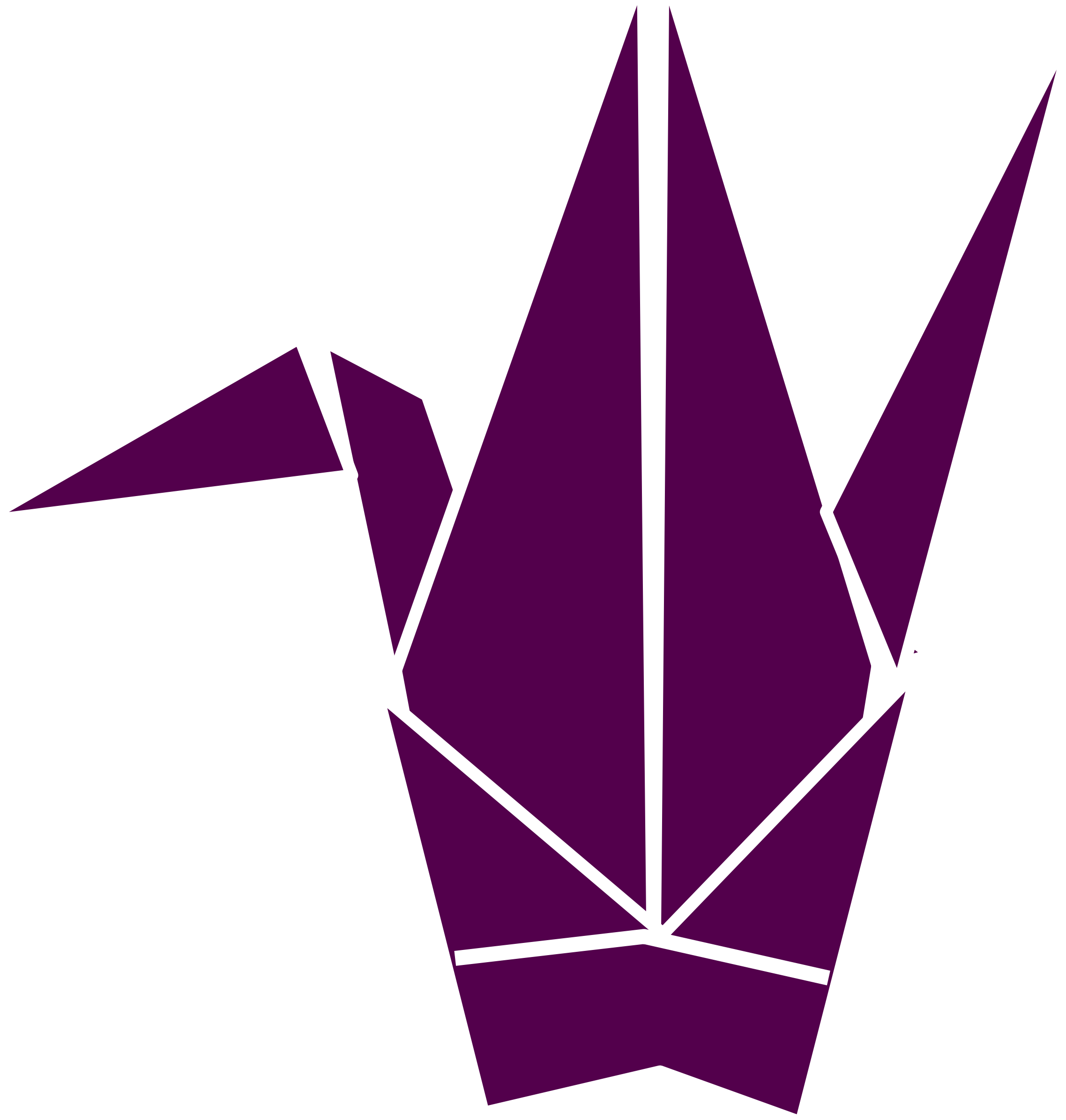
What forces serial execution?

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



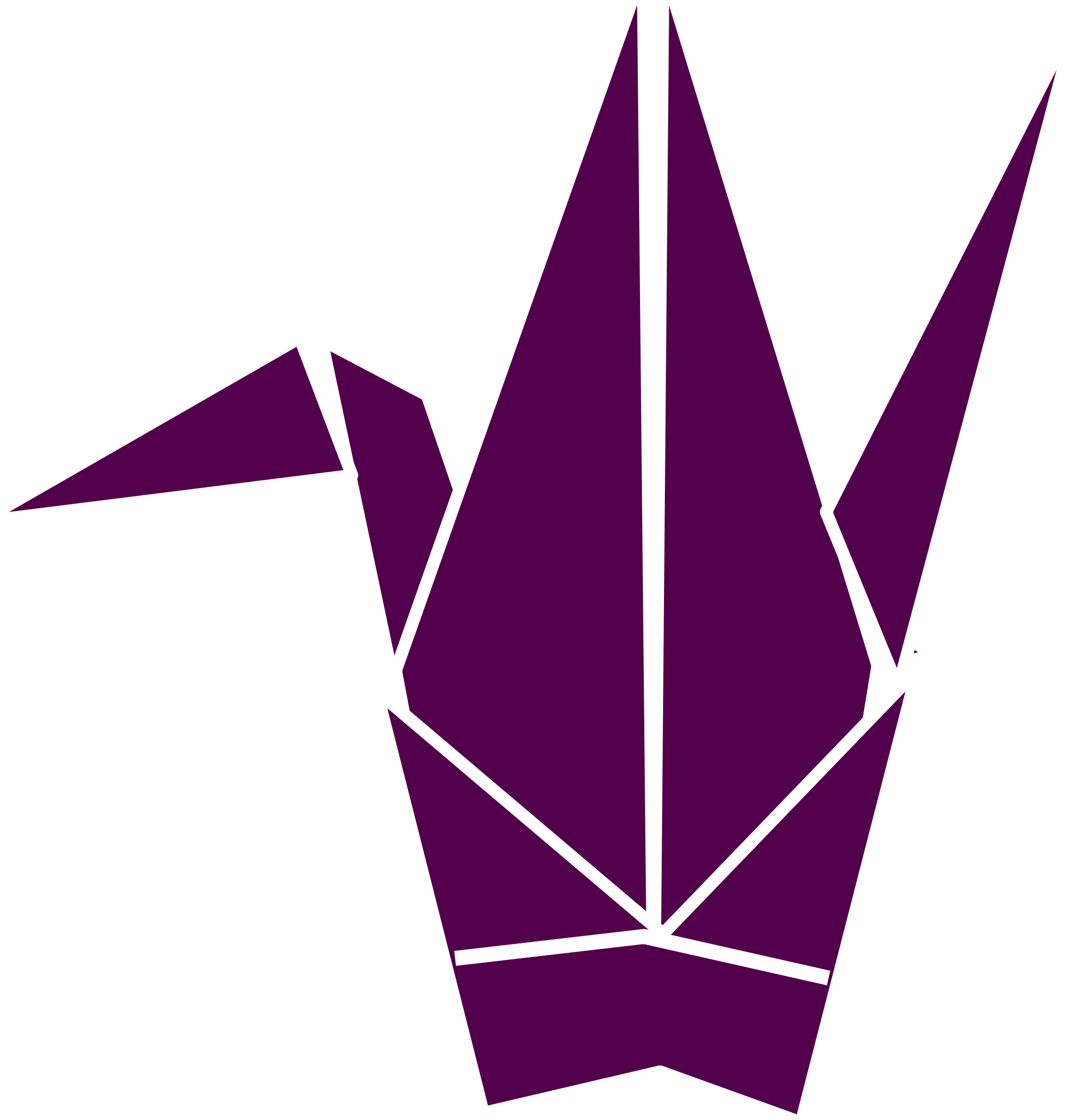
What forces serial execution?

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



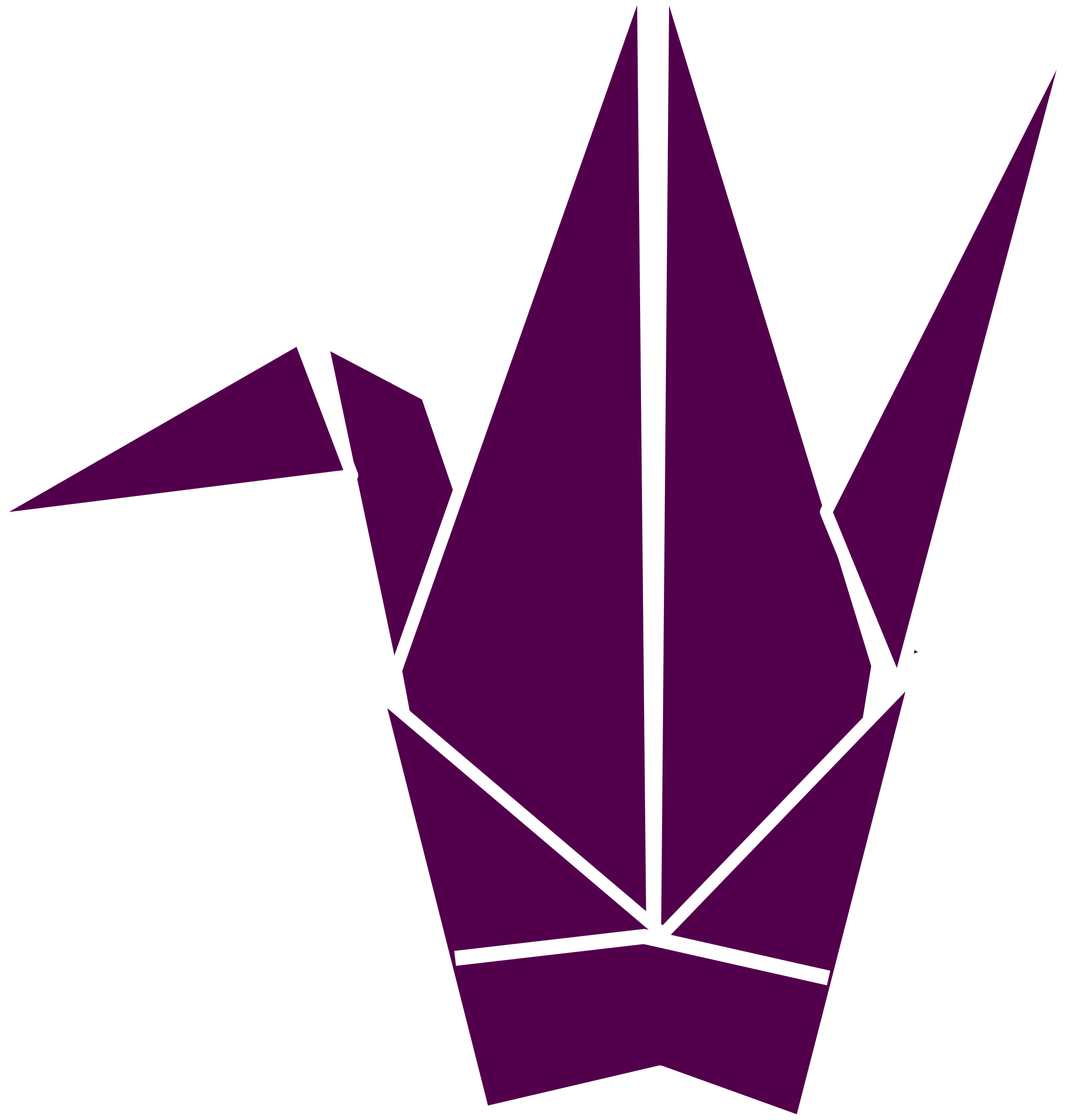
What forces serial execution?

```
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?

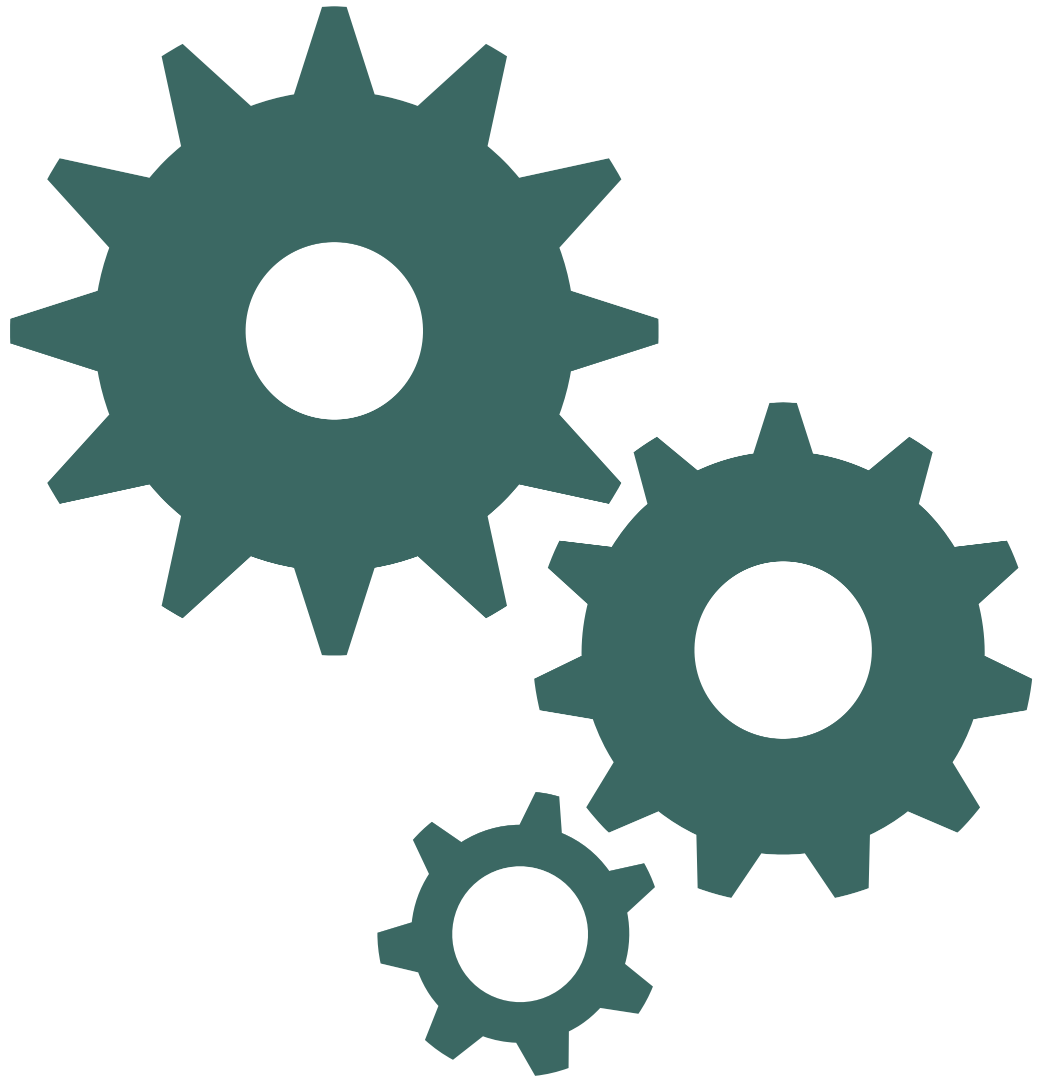
$$a \oplus b = b \oplus a$$

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

How can we fix these?

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

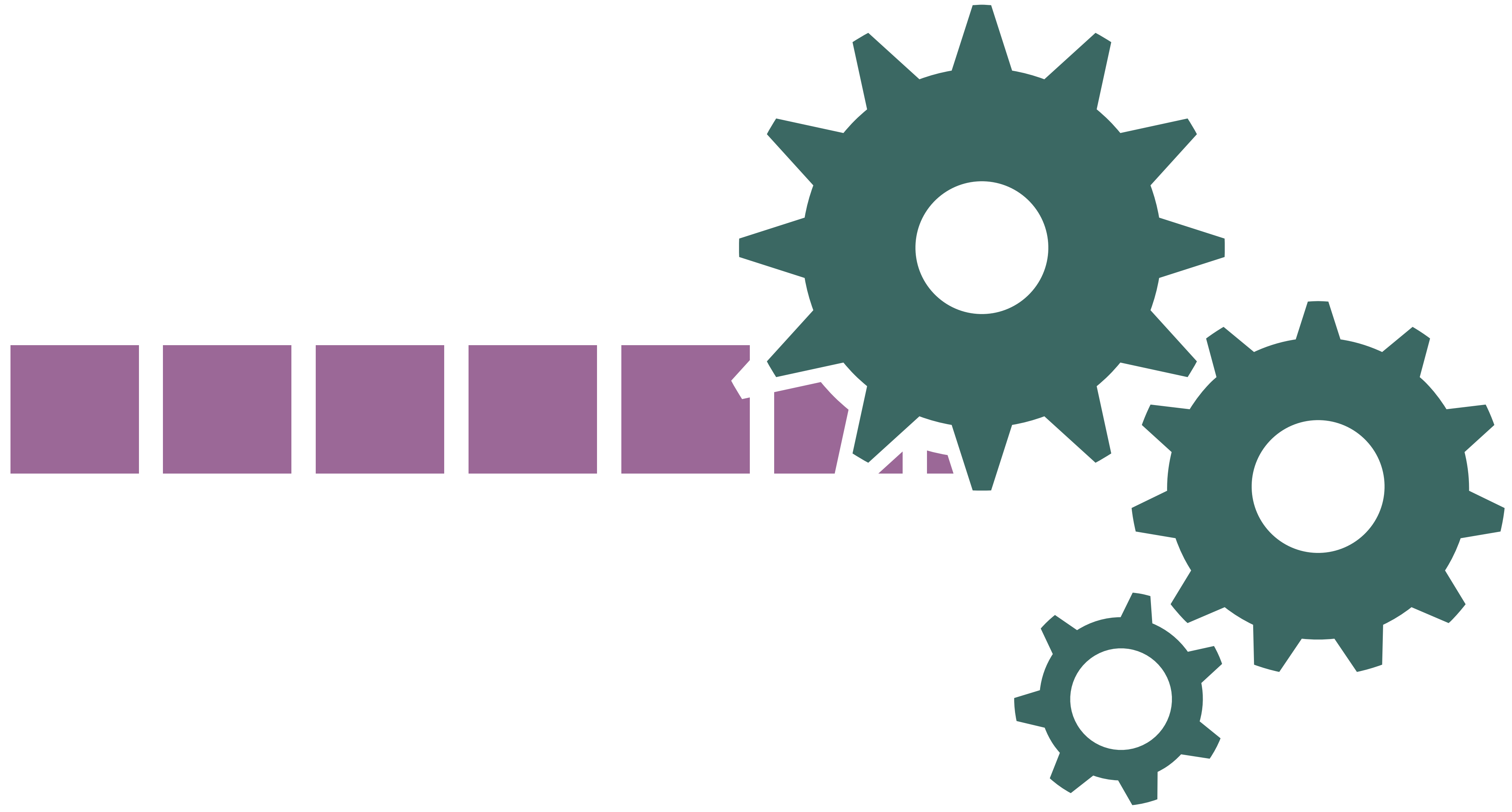
$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$



How can we fix these?

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

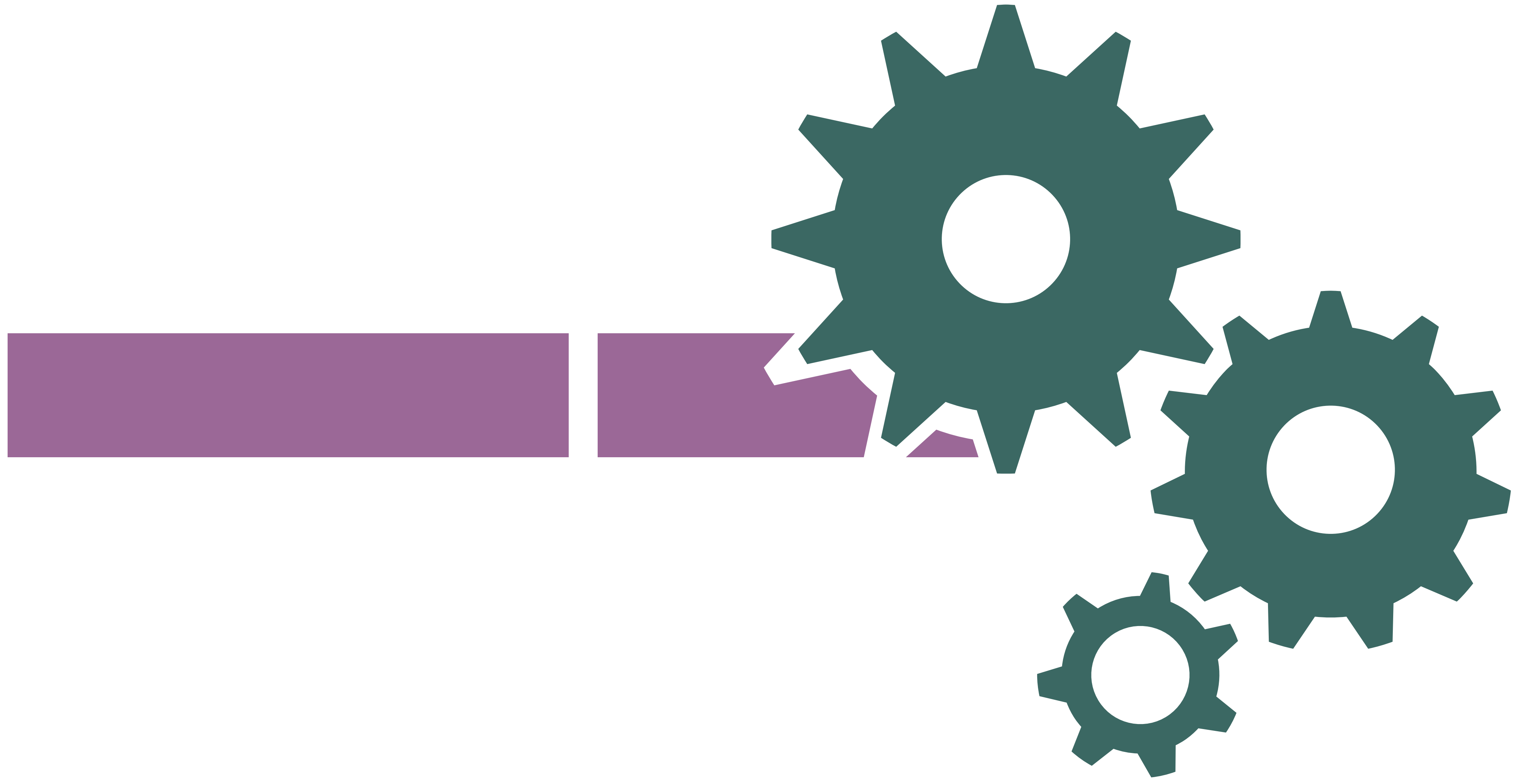
$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$



How can we fix these?

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

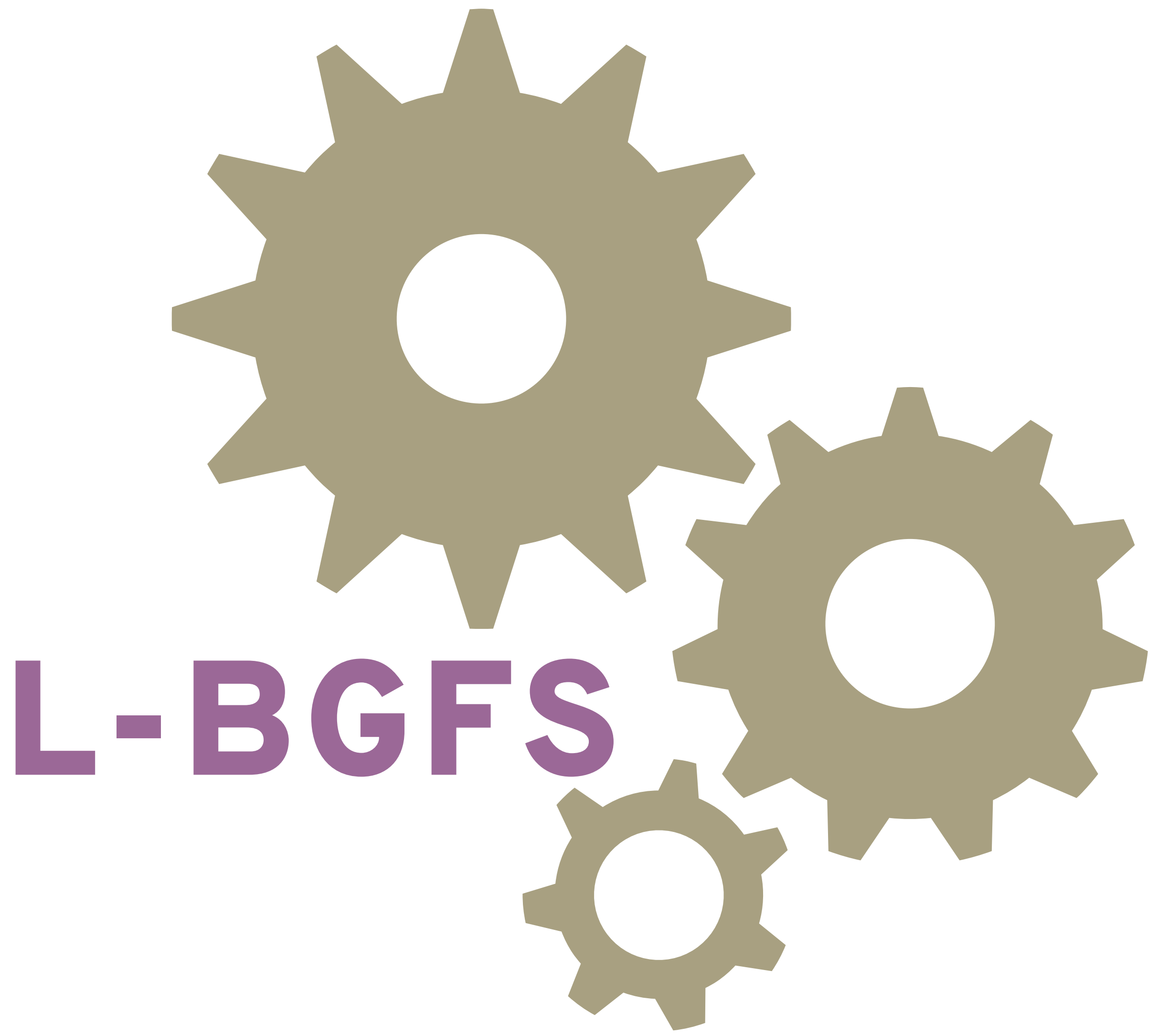
$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$



How can we fix these?

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

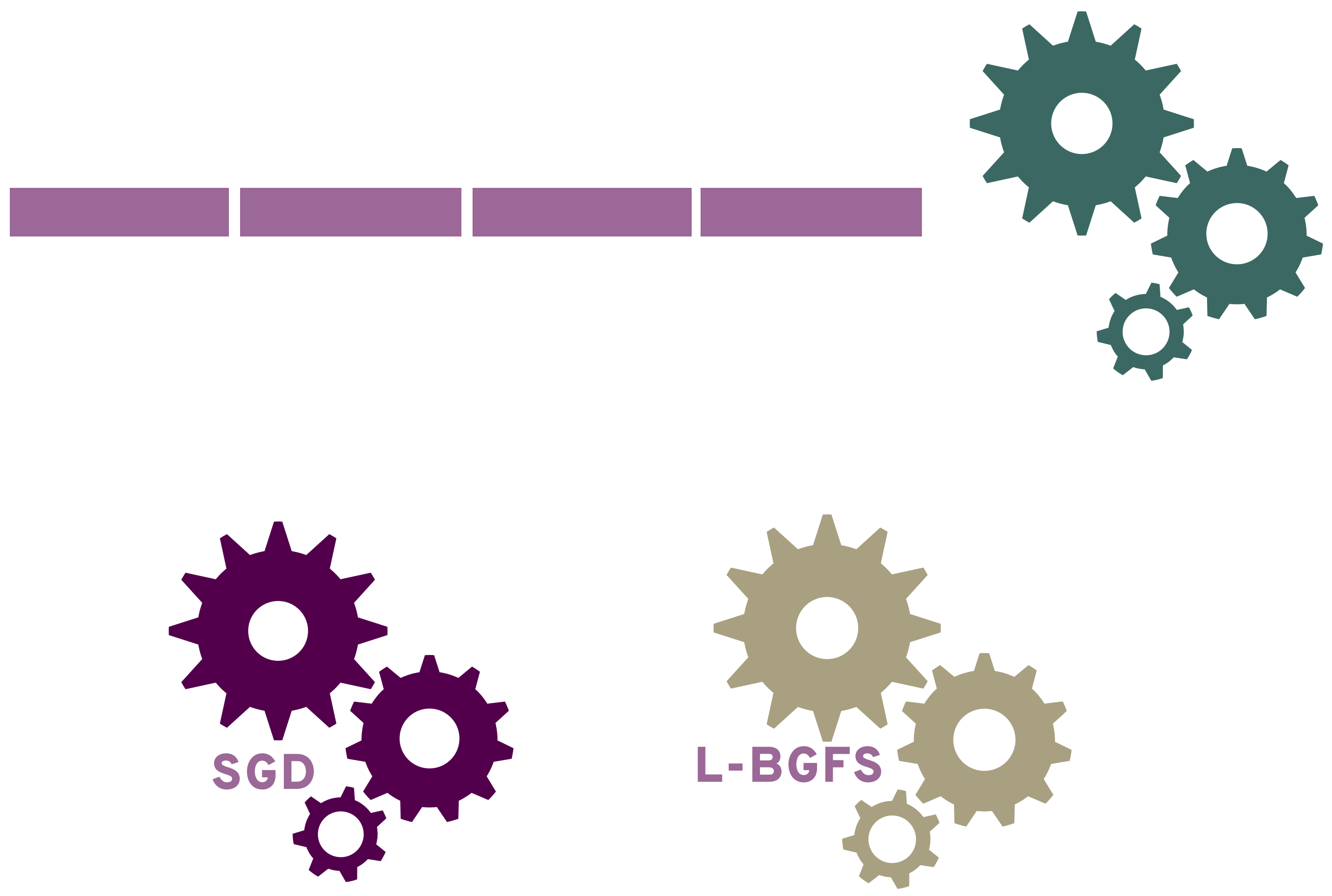
$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$



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(somₜ₋₁, ex)
    hood = neighborhood(bestMatch, sigma(t))
    state.matches += ex * hood
    state.hoods += hood
  somₜ = newSOM(state.matches / state.hoods)
```

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

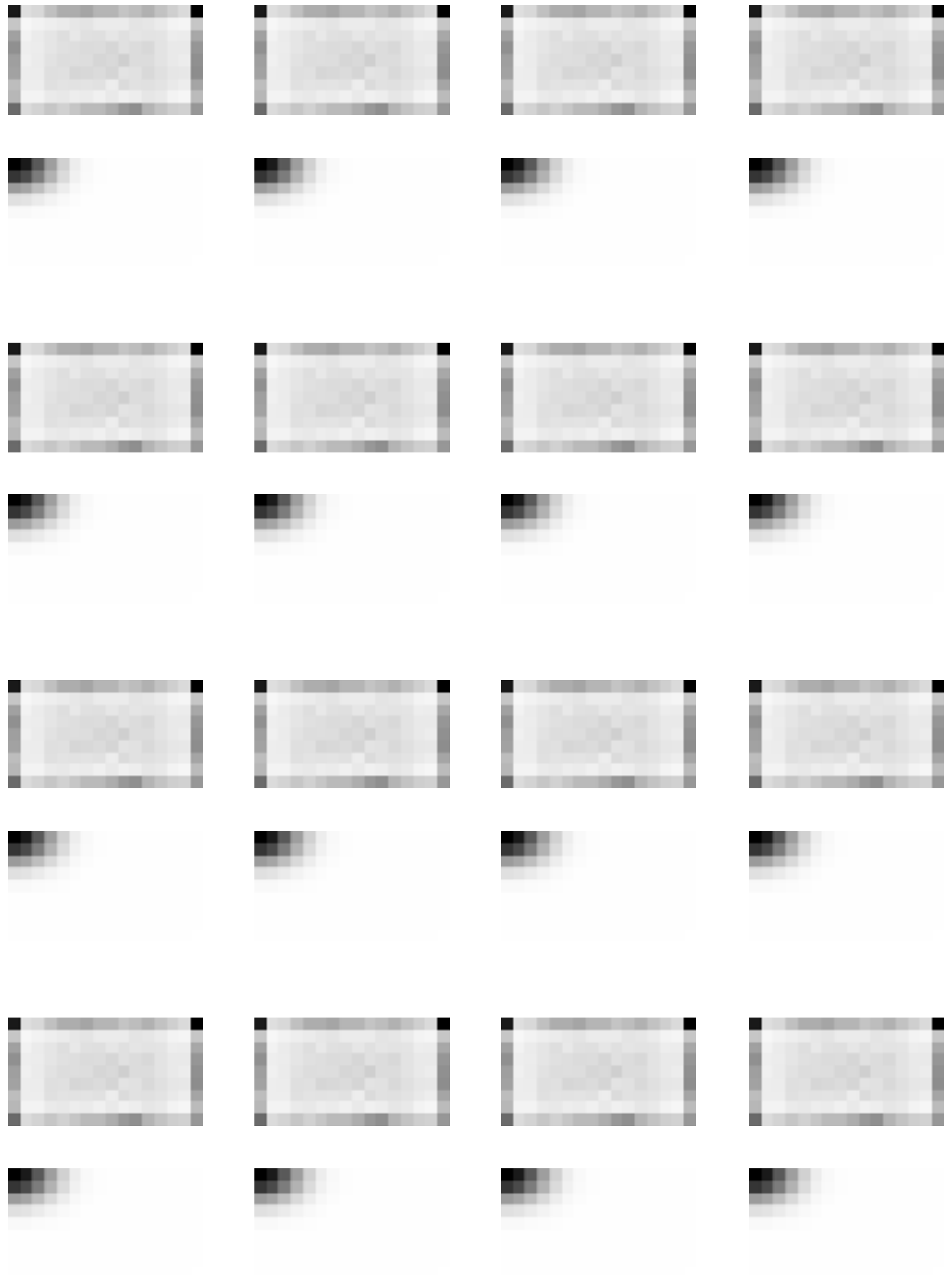
Implementing atop RDDs

```
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 newState = examples.aggregate(ModelState.empty()) {
    { case (state: ModelState, example: Example) =>
      state.update(nextModel.lookup(example, i), example) }
    { case (s1: ModelState, s2: ModelState) => s1.combine(s2) }
  }
  nextModel = modelFromState(newState)

}
```

An implementation template

```
var nextModel = initialModel
for (int i = 0; i < iterations; i++) {

  val newState = examples.aggregate(ModelState.empty()) {
    { case (state: ModelState, example: Example) =>
      state.update(nextModel.lookup(example, i), example) }
    { case (s1: ModelState, s2: ModelState) => s1.combine(s2) }
  }
  nextModel = modelFromState(newState)

}
```

"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 newState = examples.aggregate(ModelState.empty()) {
    { case (state: ModelState, example: Example) =>
      state.update(nextModel.lookup(example, i), example) }
    { case (s1: ModelState, s2: ModelState) => s1.combine(s2) }
  }
  nextModel = modelFromState(newState)

}
```

"reduce": combine the states from two partitions

An implementation template

```
var nextModel = initialModel
for (int i = 0; i < iterations; i++) {

  val newState = examples.aggregate(ModelState.empty()) {
    { case (state: ModelState, example: Example) =>
      state.update(nextModel.lookup(example, i), example) }
    { case (s1: ModelState, s2: ModelState) => s1.combine(s2) }
  }
  nextModel = modelFromState(newState)

}
```

this will cause the model object
to be serialized with the closure

An implementation template

broadcast the current working model for this iteration

```
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
}
```

get the value of the broadcast variable

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
}
```

remove the stale broadcasted model

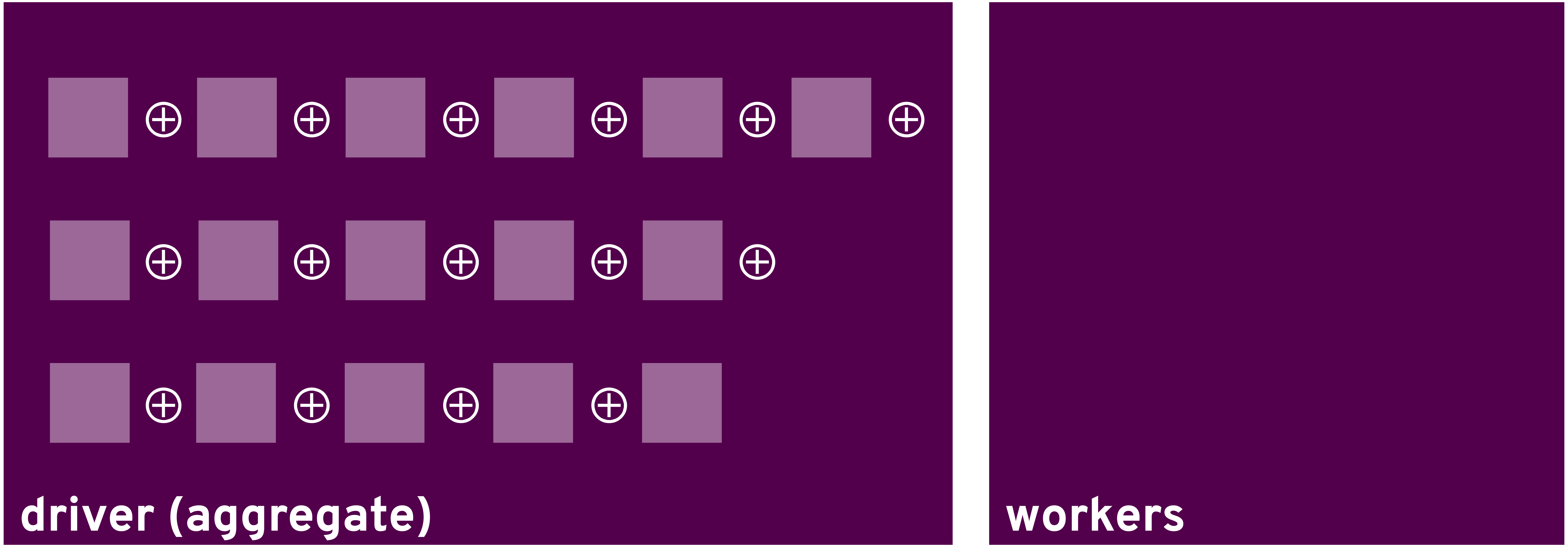
An implementation template

```
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
}
```

the wrong implementation
of the right interface

Implementing on RDDs

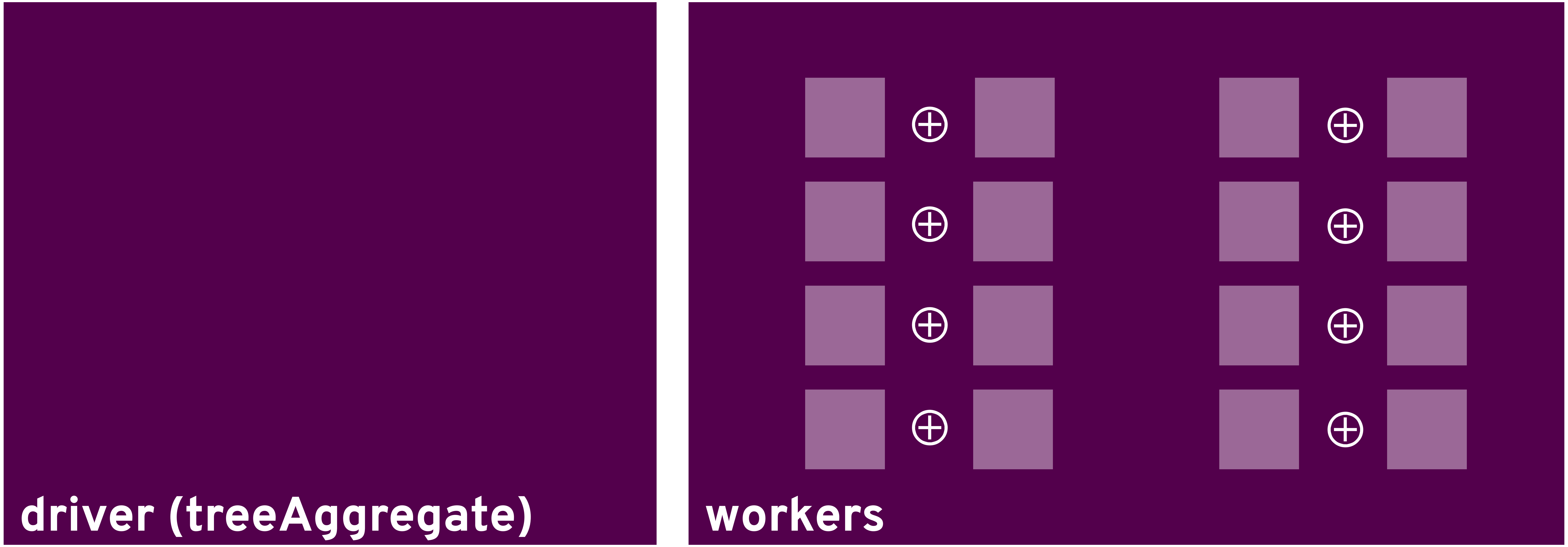
Implementing on RDDs



Implementing on RDDs

Implementing on RDDs

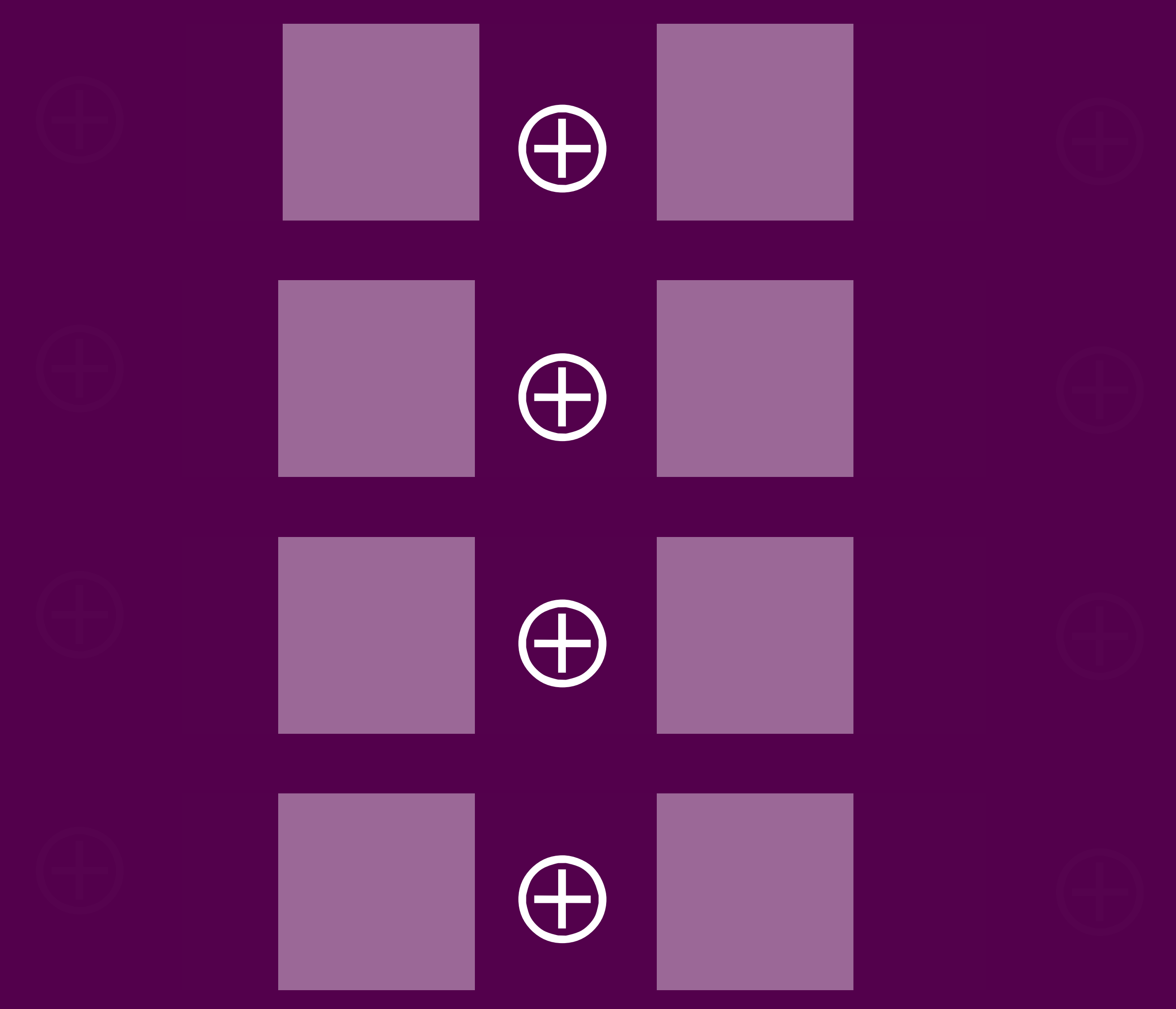
Implementing on RDDs



Implementing on RDDs

Implementing on RDDs

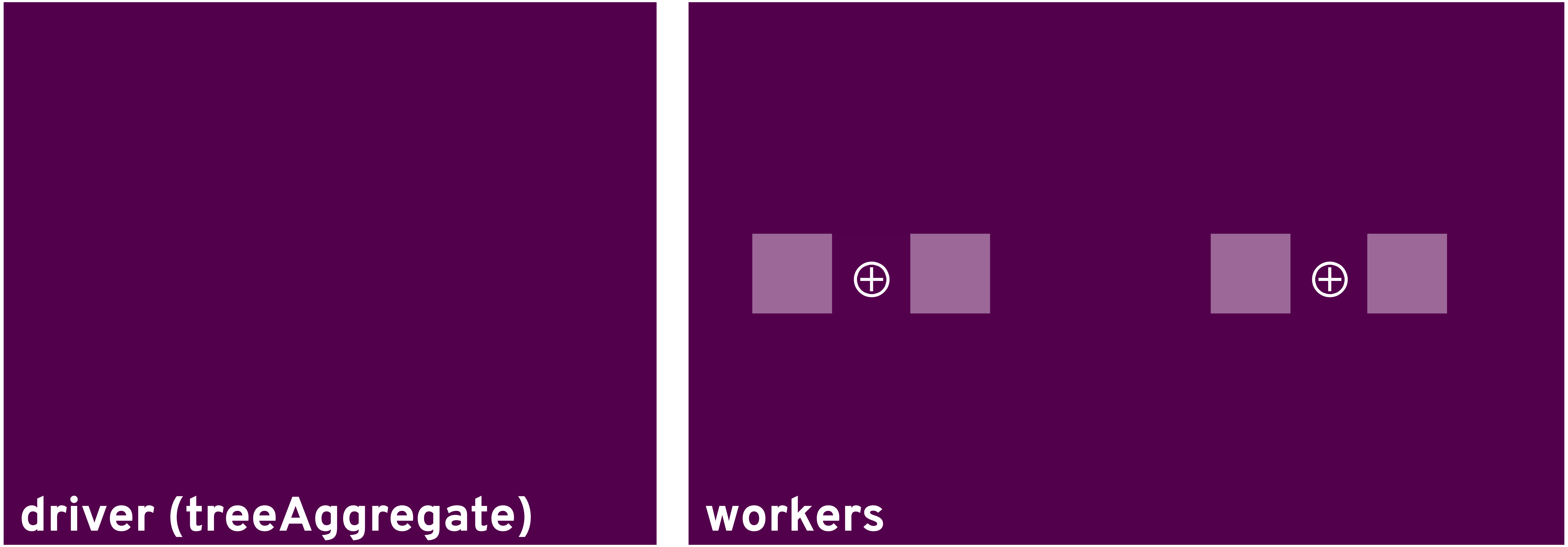
driver (treeAggregate)



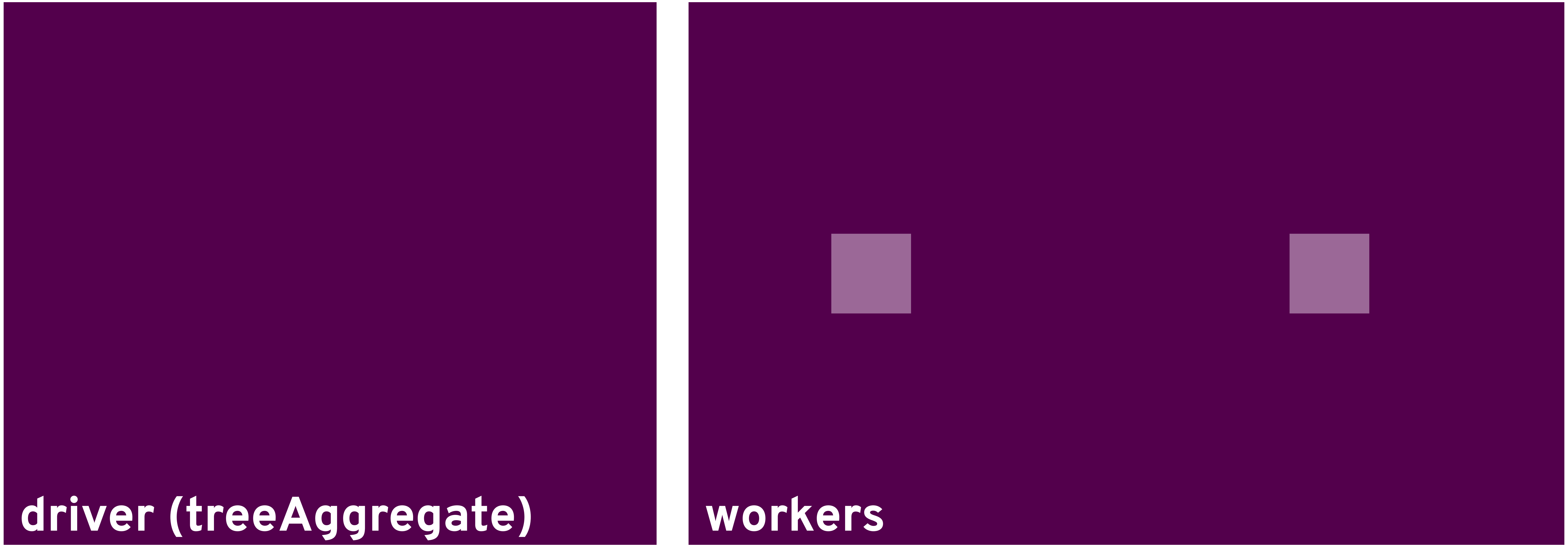
workers

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

doesn't compile

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

RDDs: some good parts

doesn't compile

```
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()

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

doesn't compile

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 JVM 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))
```

32 bytes of data…

RDDs and the Java heap

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

32 bytes of data…

…and 64 bytes of overhead!

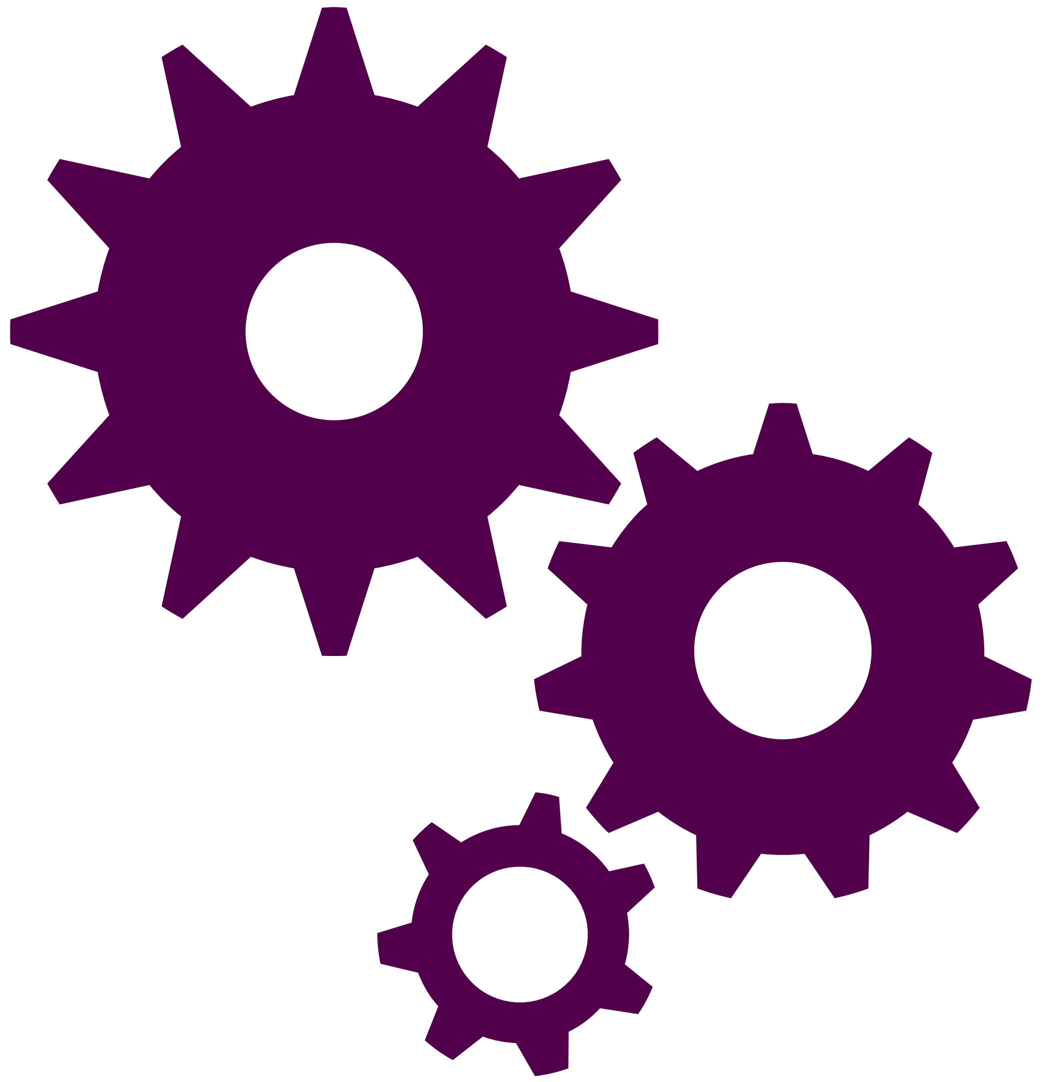
ML pipelines: a quick example

```python
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

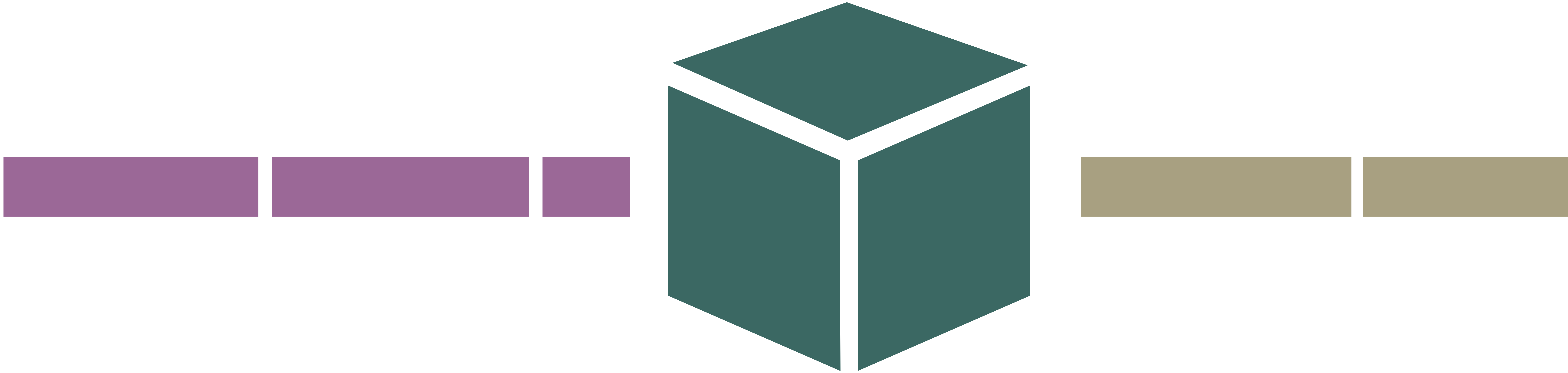


`estimator.fit(df)`   `model.transform(df)`

Working with ML pipelines

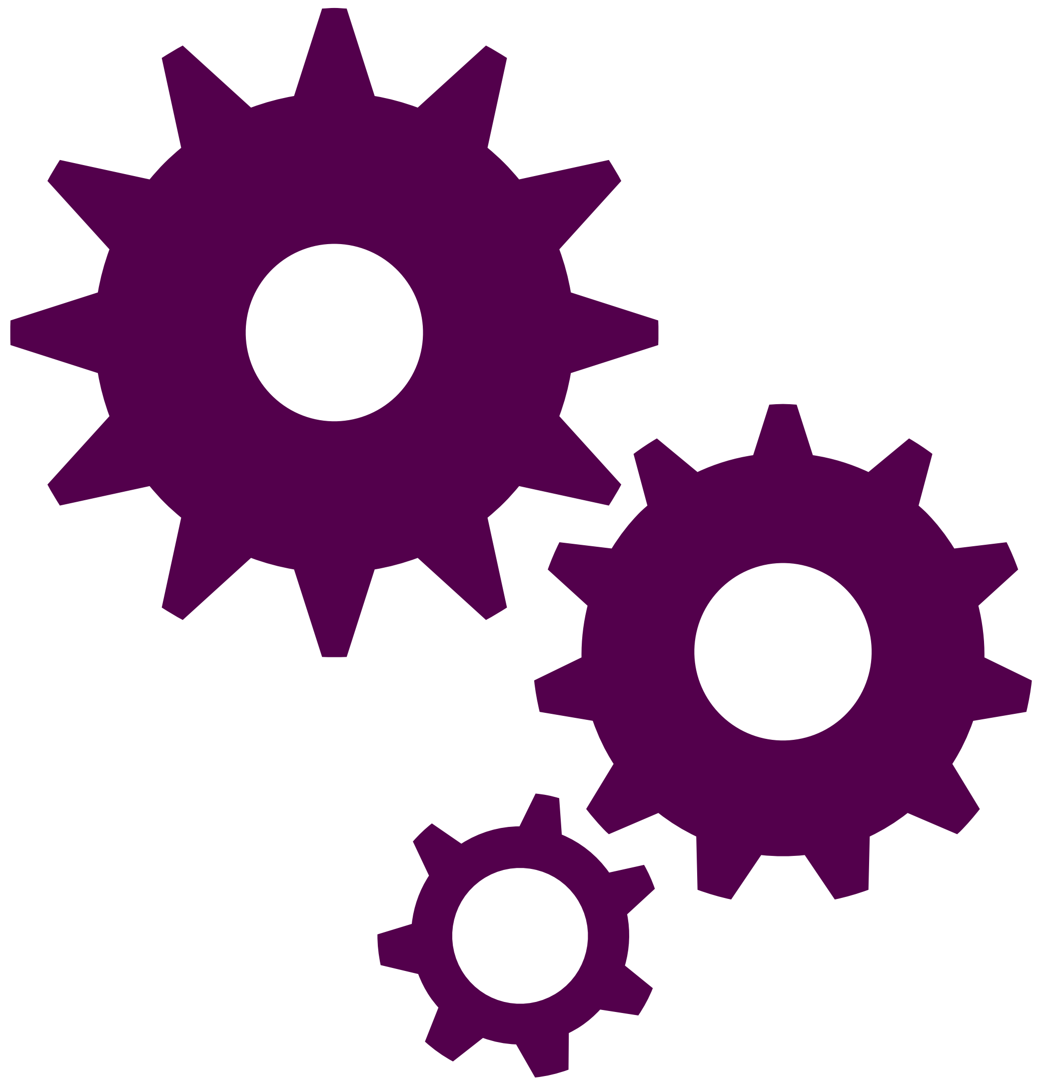
```
model.transform(df)
```

Working with ML pipelines



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

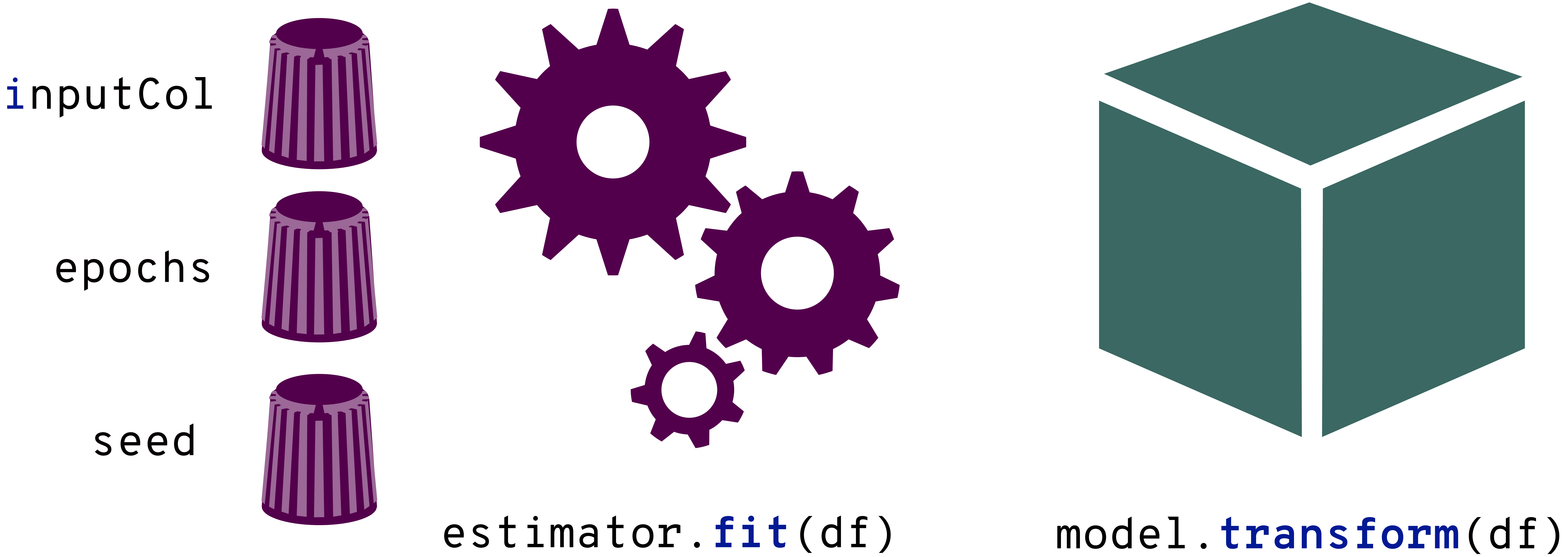
Working with ML pipelines



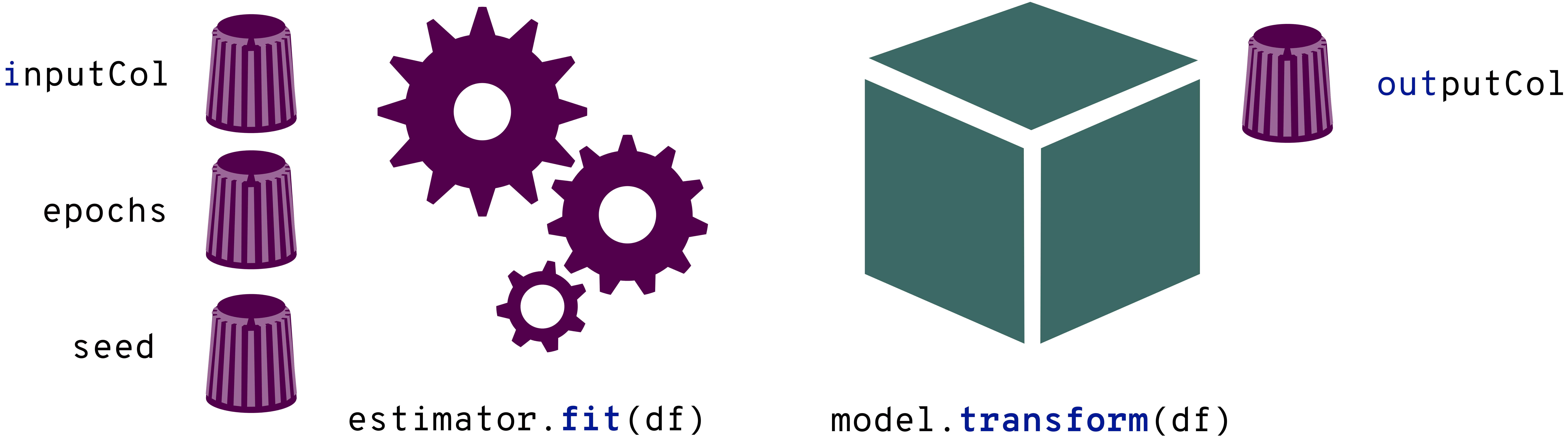
`estimator.fit(df)`

`model.transform(df)`

Working with ML pipelines



Working with ML pipelines



Defining parameters

```scala
private[som] trait SOMParams extends Params
    with DefaultParamsWritable {
```

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

```scala
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

```scala
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)
  // ...
```

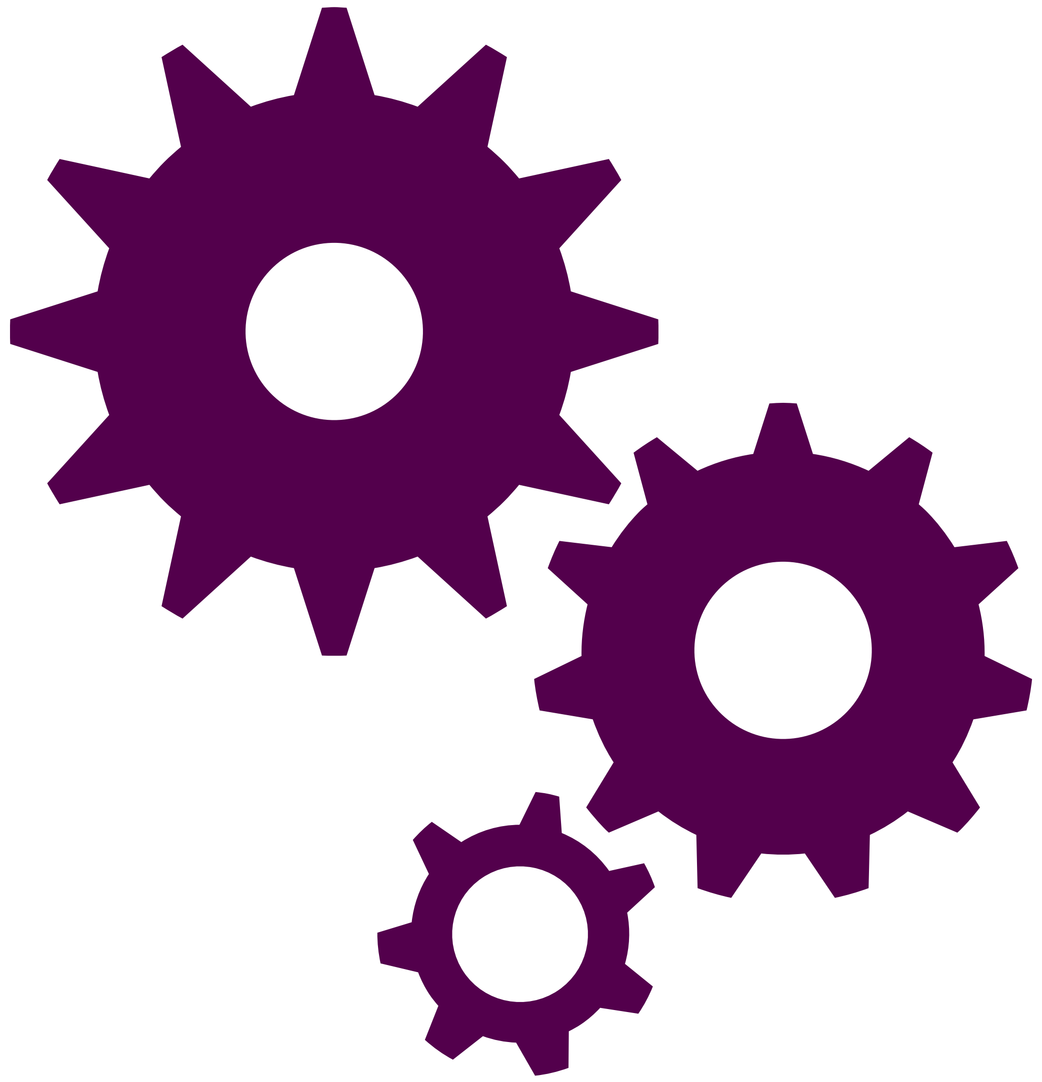
Defining parameters

```scala
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

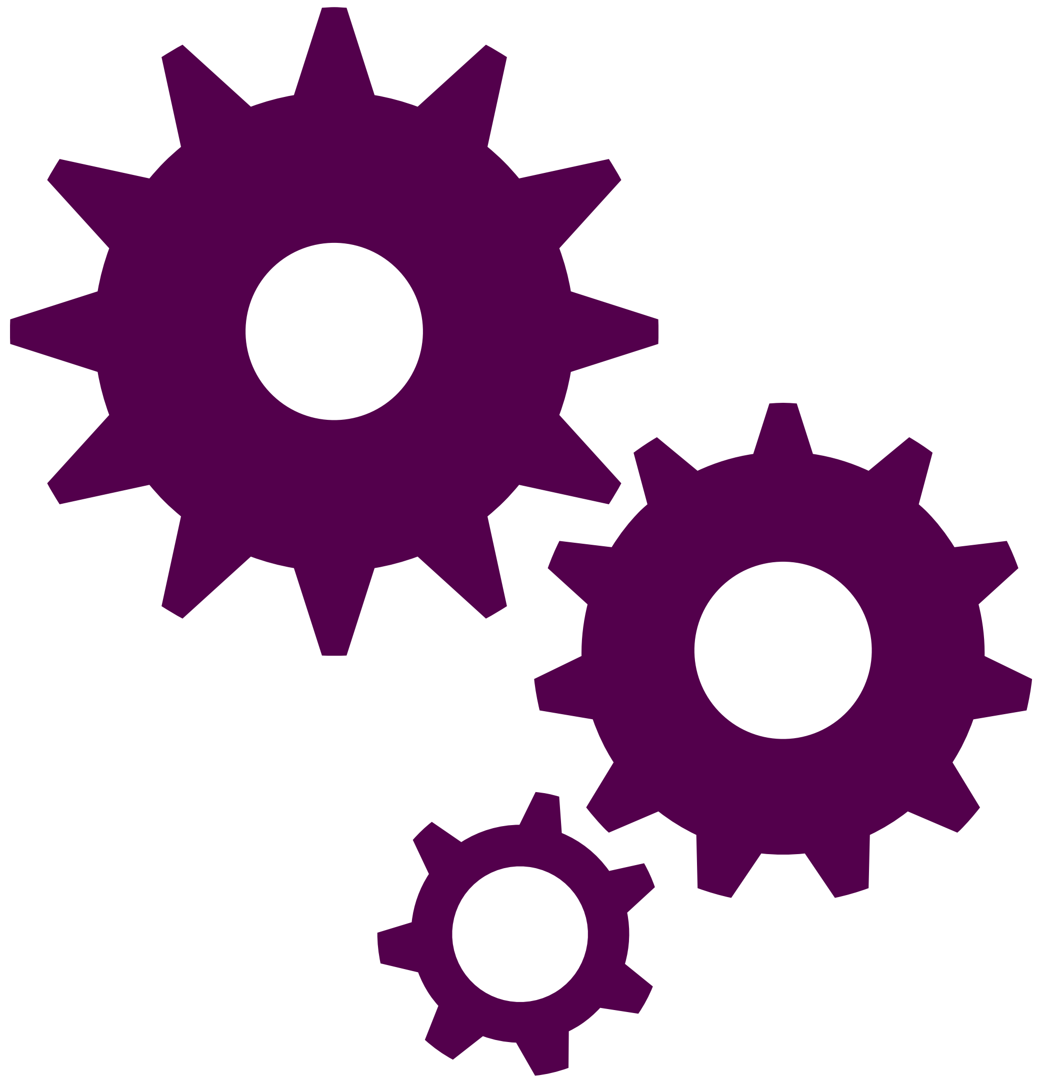
```scala
/**
 * Common params for KMeans and KMeansModel
 */
private[clustering] trait KMeansParams extends Params
    with HasMaxIter with HasFeaturesCol
    with HasSeed with HasPredictionCol with HasTol { /* ... */ }
```

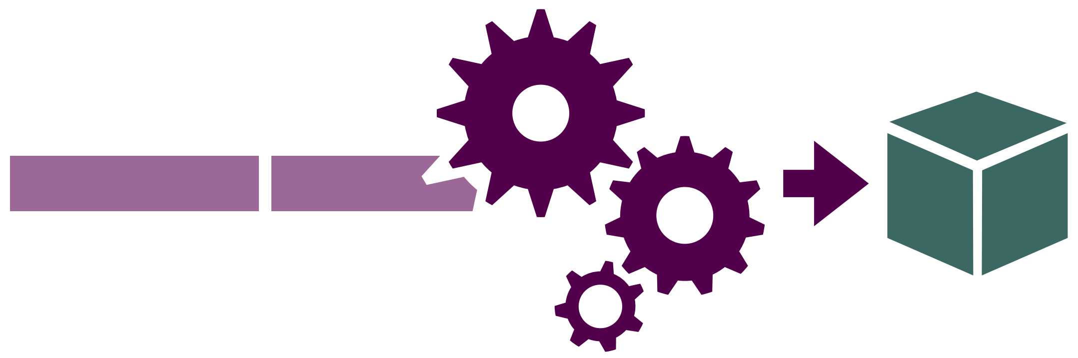
Estimators and transformers



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

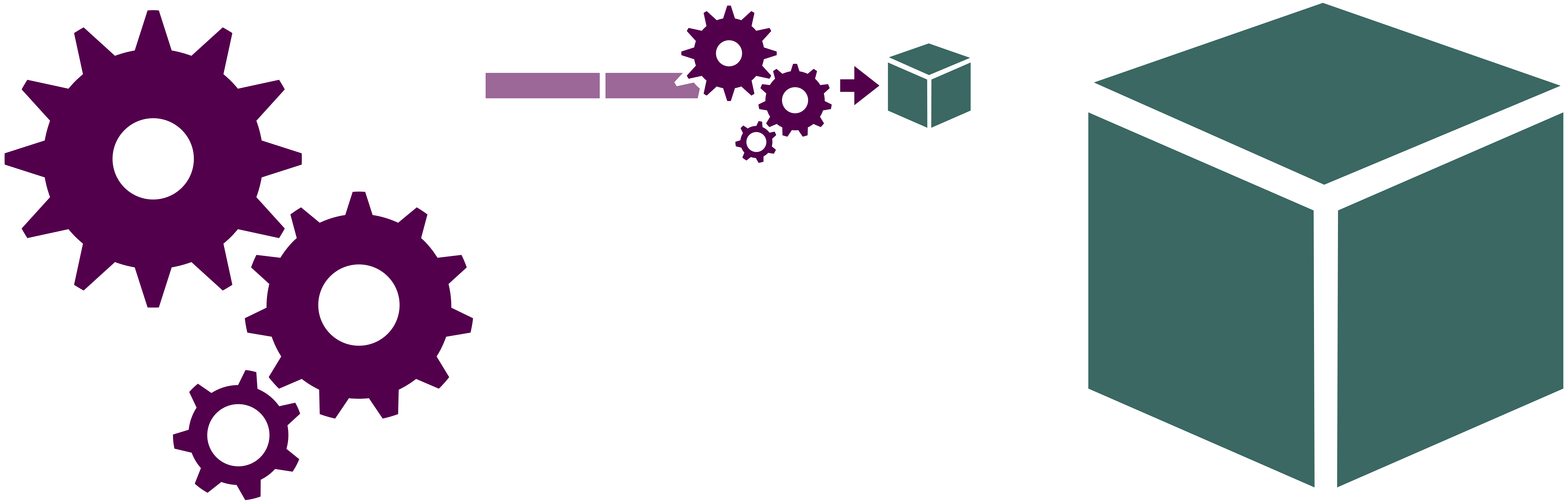
Estimators and transformers





`estimator.fit(df)`

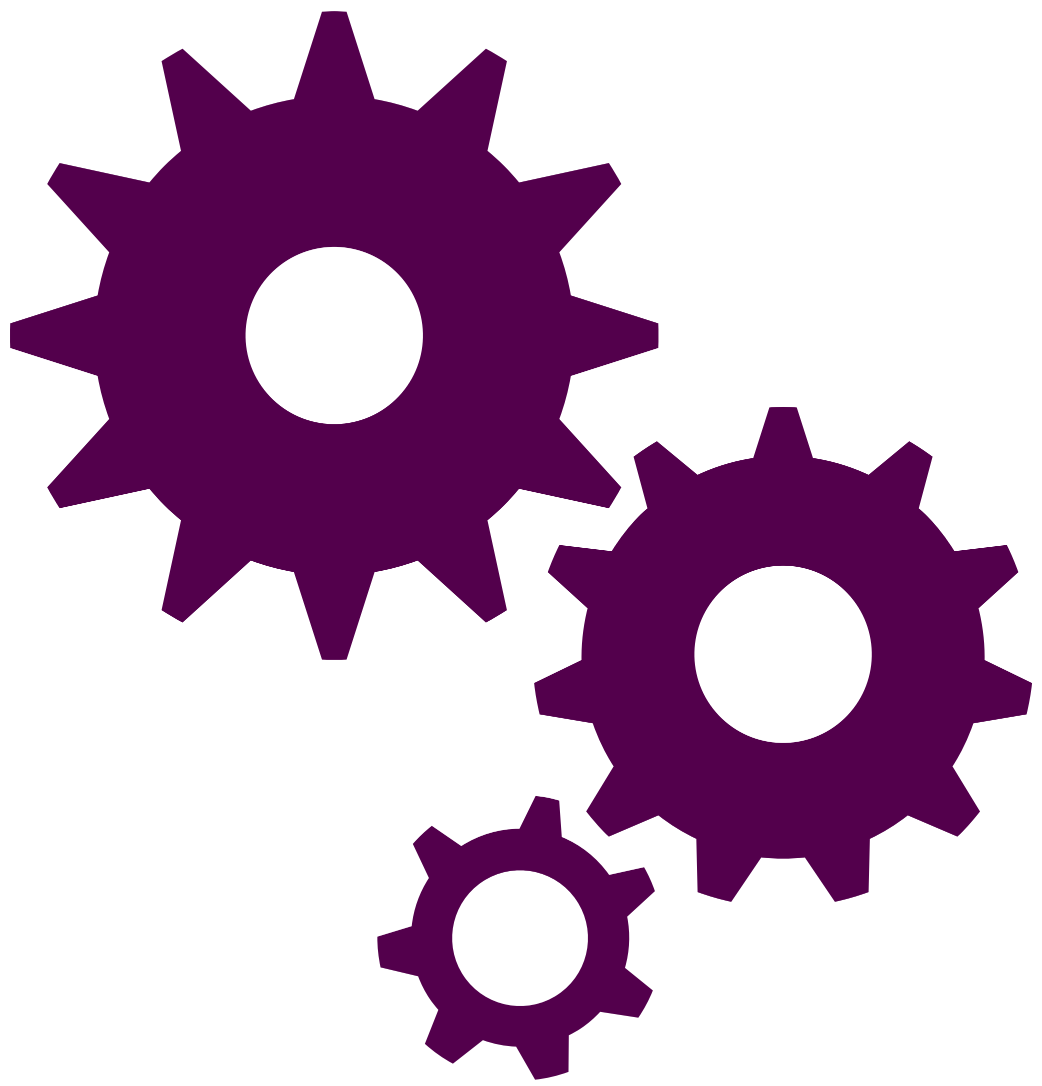
Estimators and transformers

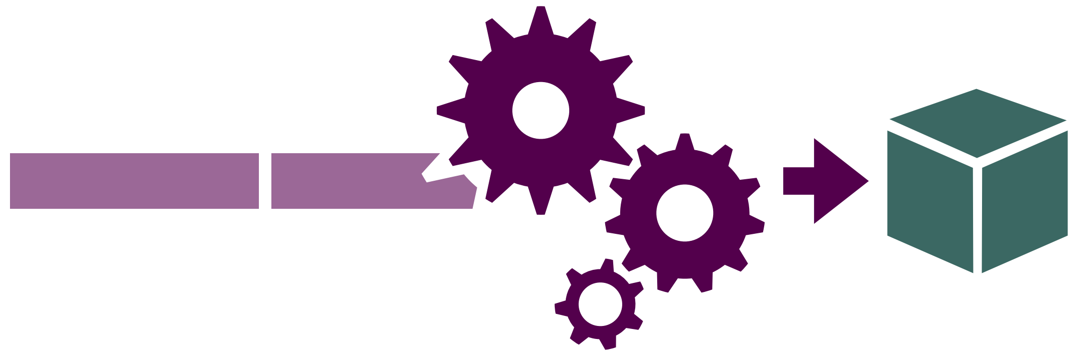


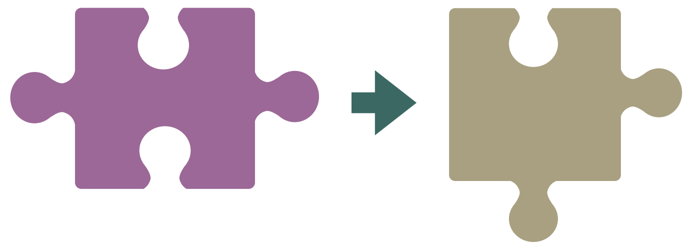
`estimator.fit(df)`

`model.transform(df)`

Estimators and transformers



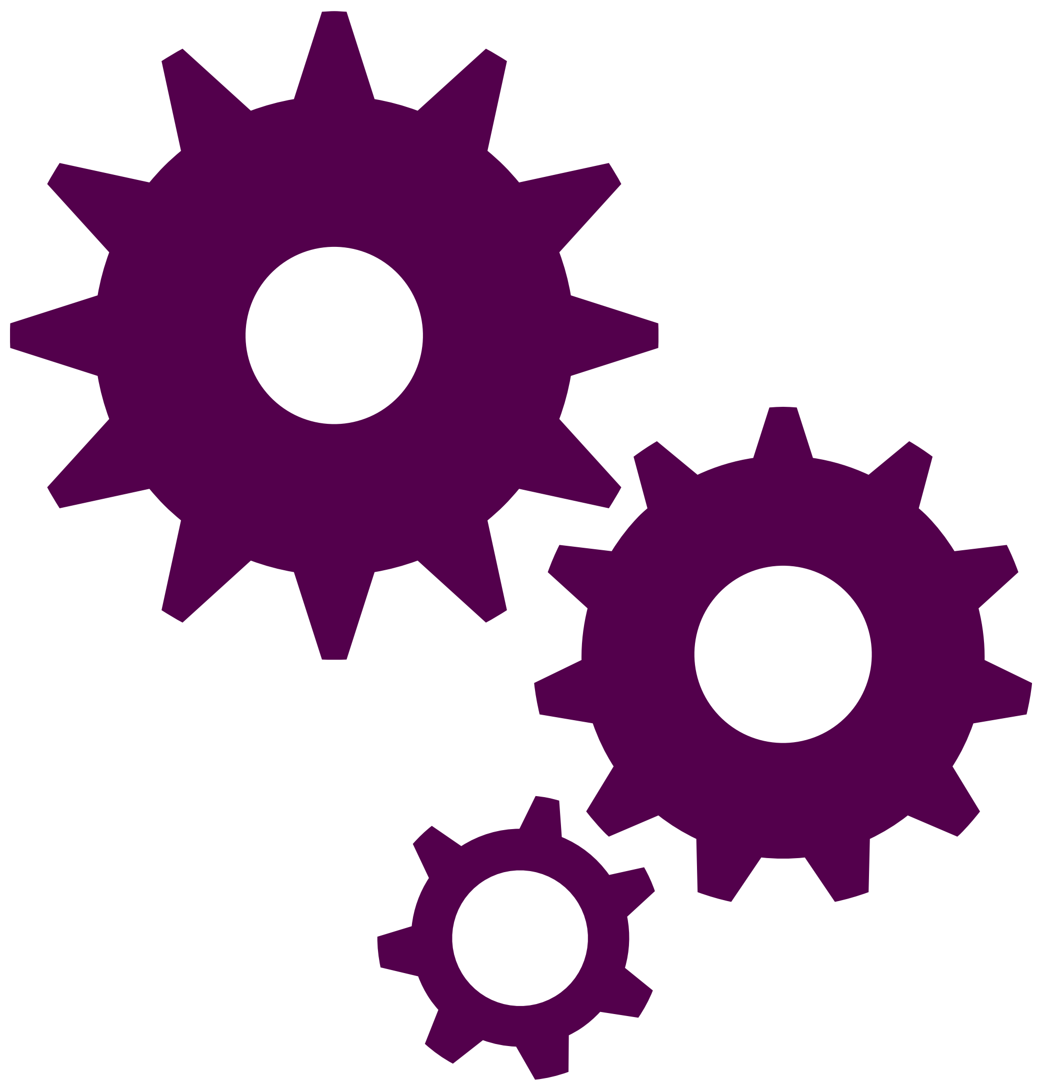


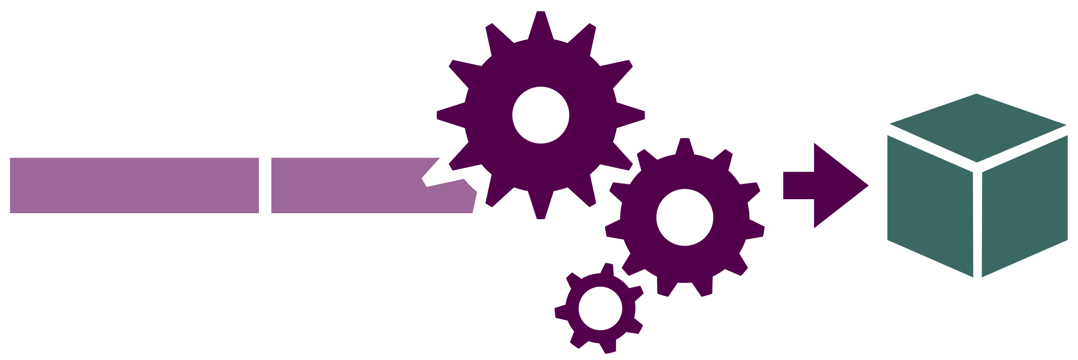


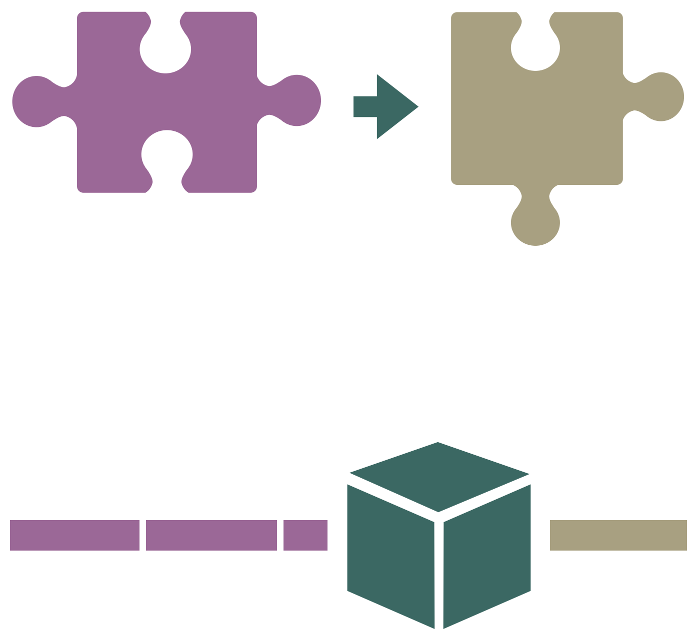
`estimator.fit(df)`

`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

```scala
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 and key takeaways

Retire your visibility hacks

```scala
package org.apache.spark.ml.hacks

object Hacks {
  import org.apache.spark.ml.linalg.VectorUDT
  val vectorUDT = new VectorUDT
}
```

Retire your visibility hacks

```scala
package org.apache.spark.ml.linalg

/* imports, etc., are elided ... */
@Since("2.0.0")
@DeveloperApi
object SQLDataTypes {
  val VectorType: DataType = new VectorUDT
  val MatrixType: DataType = new MatrixUDT
}
```

Caching training data

```
val wasUncached = examples.storageLevel == StorageLevel.NONE

if (wasUncached) { examples.cache() }

/* actually train here */

if (wasUncached) { examples.unpersist() }
```

Improve serial execution times

Are you repeatedly comparing training data to a model that only changes once per iteration? **Consider caching norms.**

Are you doing a lot of dot products in a `for` loop? **Consider replacing these loops with a matrix-vector multiplication.**

Seek to limit the number of library invocations you make and thus the time you spend copying data to and from your linear algebra library.

Key takeaways

There are **several techniques** you can use to develop **parallel implementations** of **machine learning algorithms.**

The **RDD API** may not be your favorite way to interact with Spark **as a user**, but it can be **extremely valuable** if you're **developing libraries for Spark.**

As a library developer, you might need to **rely on developer APIs** and **dive in to Spark's source code,** but things are getting easier with each release!

Thanks!

@willb • willb@redhat.com

https://chapeau.freevariable.com

https://radanalytics.io