Compact and safely: static DSL on Kotlin

Dmitry Pranchuk

About me

- Dmitry Pranchuk
- Developer at 

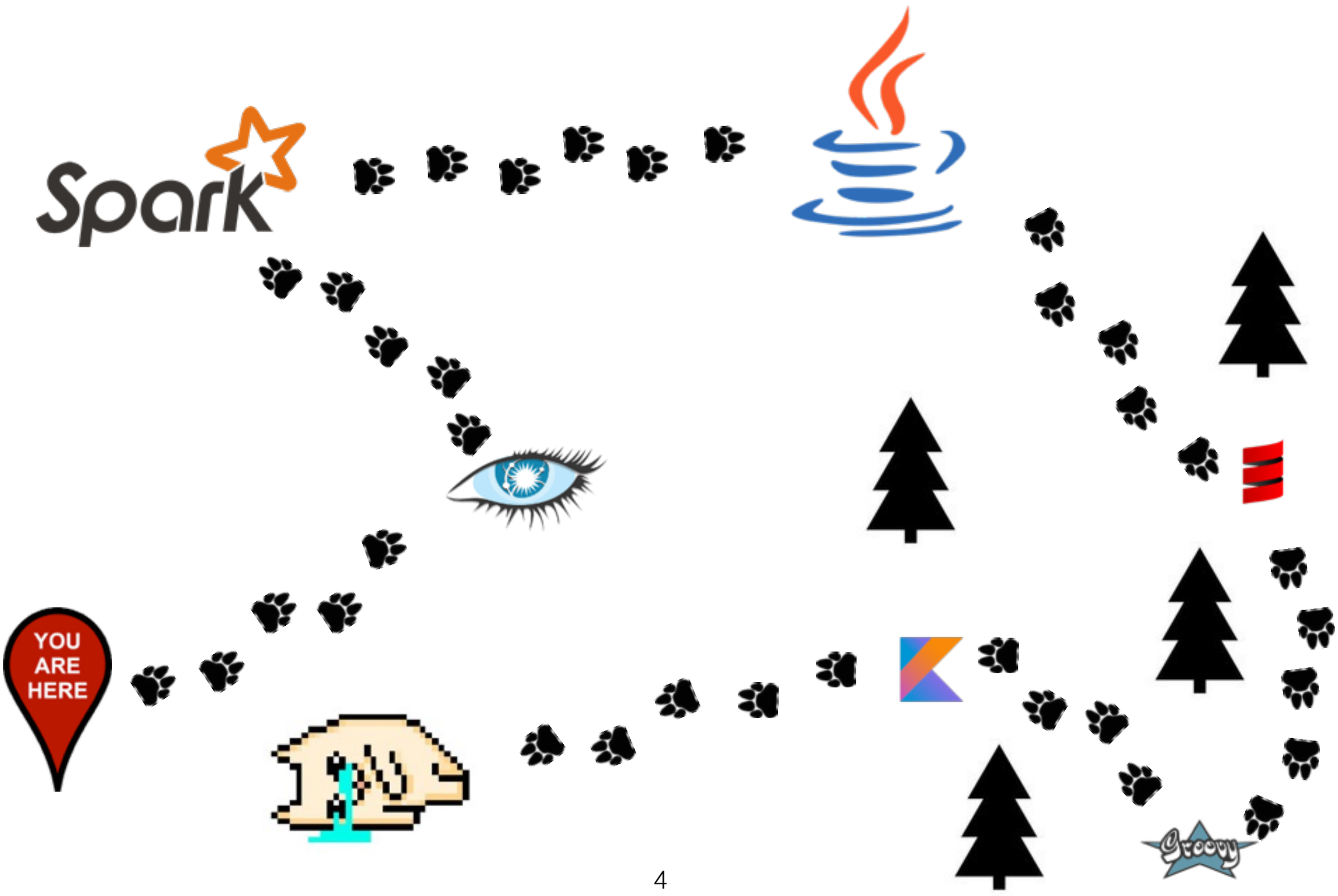
- Back-end + a little bit of devops

- @cortwave at gitter
- d.pranchuk@gmail.com

Problem

1. Some libraries have redundant API
2. It's hard to add some new functionality to library in Java
3. Some JVM languages give you ability to build your own DSL, but sometimes it:
    - implicit
    - unsafe

Roadmap



Legend

 - Scala code

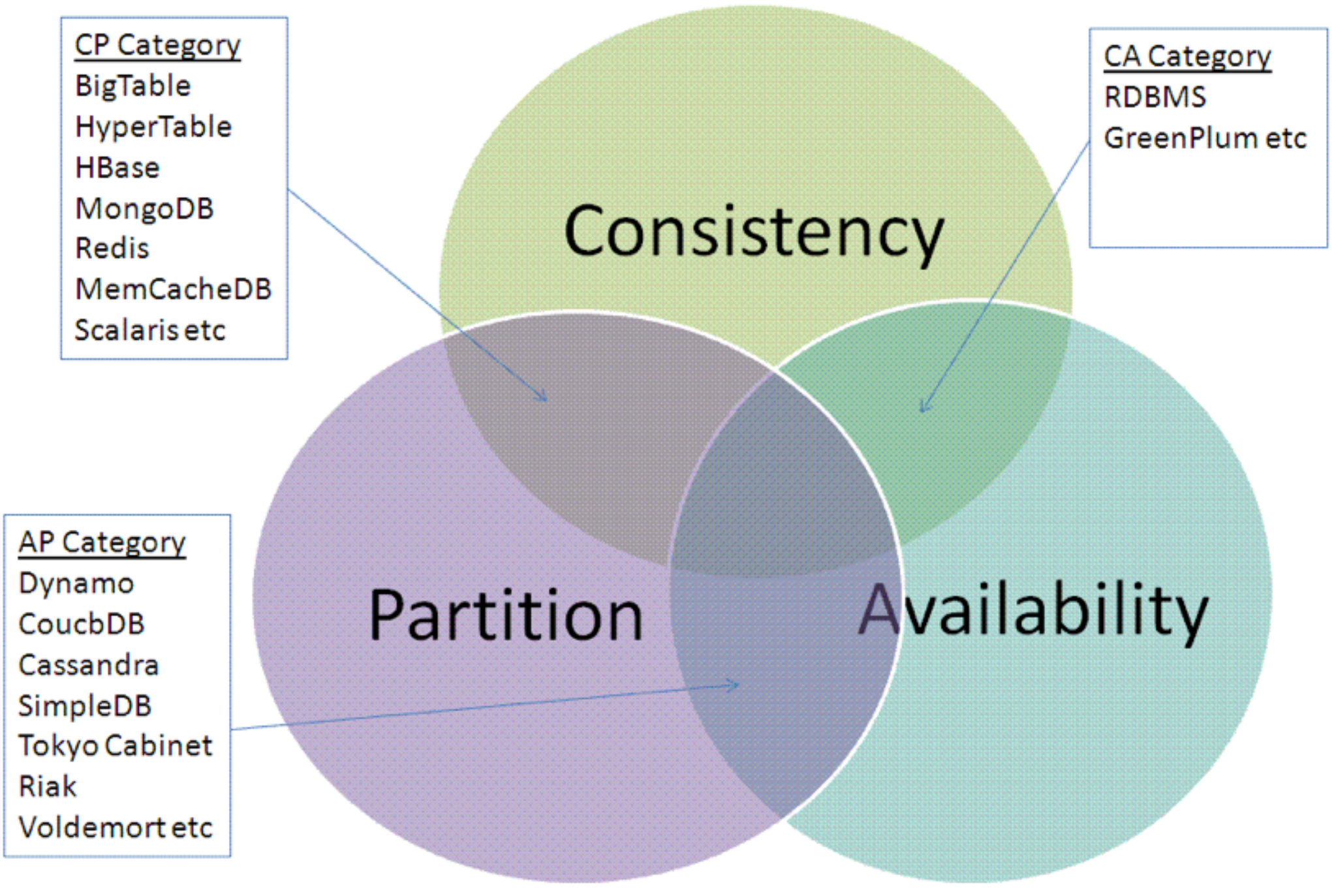
 - Kotlin code

 - Groovy code

 - Java code

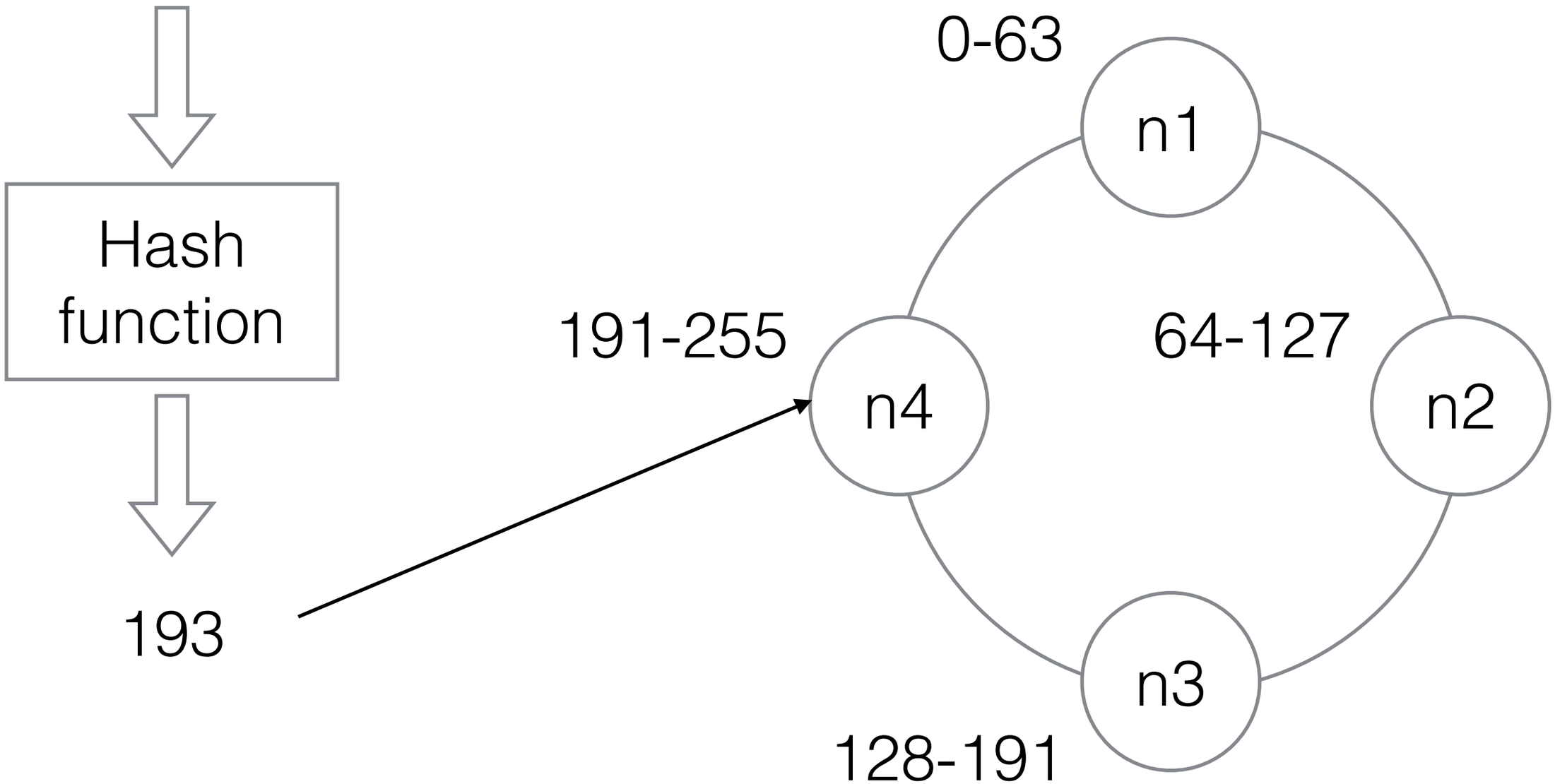

CASSANDRA

CAP theorem


CP Category
BigTable
HyperTable
HBase
MongoDB
Redis
MemCacheDB
Scalaris etc
CA Category
RDBMS
GreenPlum etc
Consistency
Partition
Availability
AP Category
Dynamo
CoucbDB
Cassandra
SimpleDB
Tokyo Cabinet
Riak
Voldemort etc

Primary key

Partition key	Clustering key	C1	C2	C3	C4
		V1	V2	V3	V4



+ fault-tolerance

+ linear scalability

- read restrictions

(pk1, pk2) (ck1, ck2, ck3)

pk1, pk2 - mandatory fields

ck(i) - can be specified only if ck(1..i-1) specified

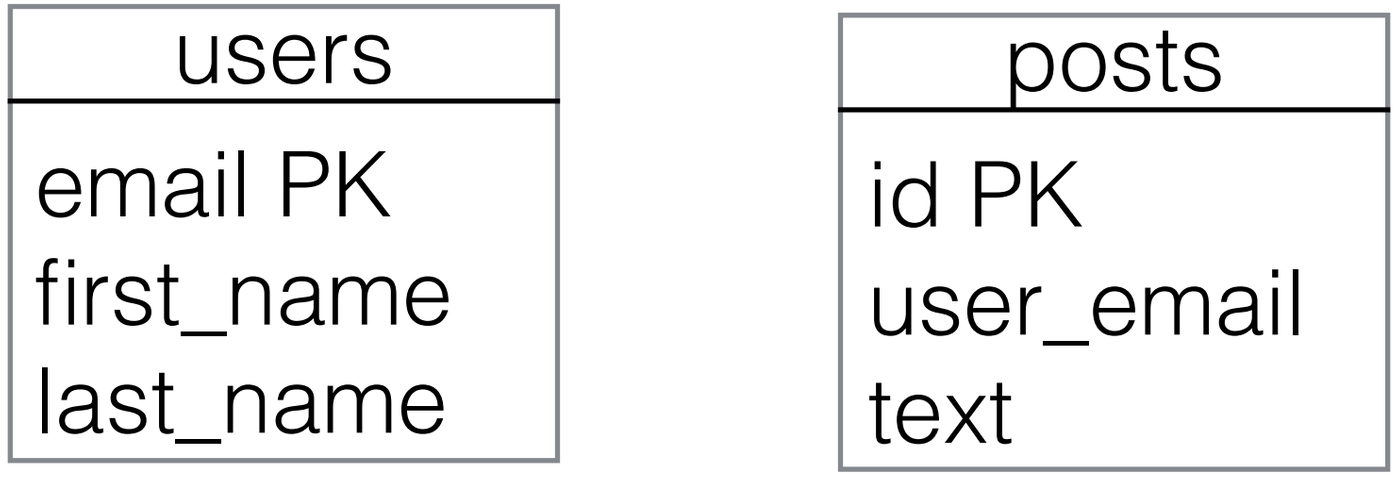
… WHERE pk1=… AND pk2=…

… WHERE pk1=… AND pk2=… AND ck1>…

… WHERE pk1=… AND pk2=… AND ck1>… AND ck2<…

… WHERE pk1=… AND pk2=… AND ck1>… AND ck2<… AND ck3>…

Normalized schema



1) SELECT * FROM users WHERE email=… ✓

2) SELECT * FROM posts WHERE id=… ✓

3) SELECT * FROM posts WHERE user_email=… ✗

Denormalized schema

users
email PK
first_name
last_name

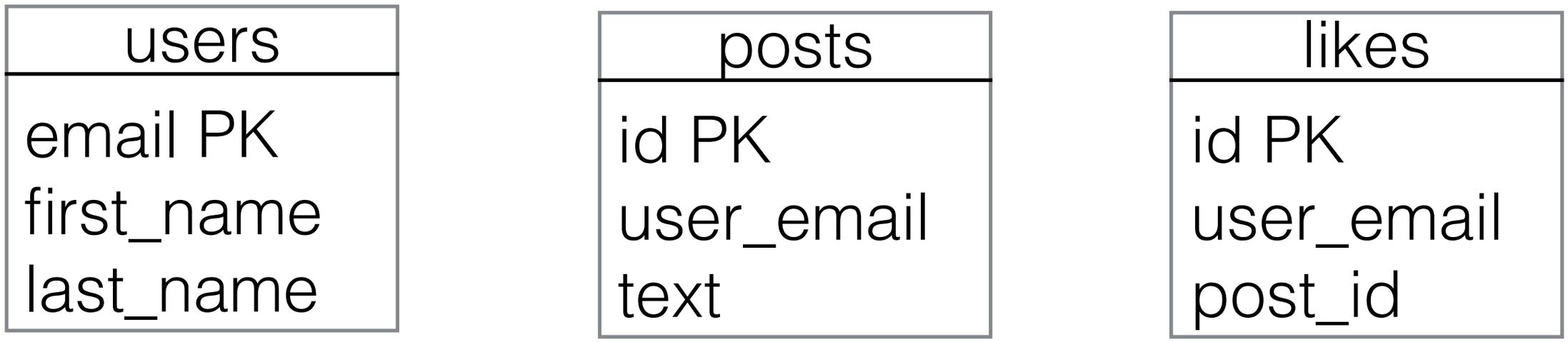
posts
id PK
user_email
text

posts_by_user_email
user_email PK
id CK
text

1) SELECT * FROM users WHERE email=… ✓

2) SELECT * FROM posts WHERE id=… ✓

3) SELECT * FROM posts_by_user_email WHERE user_email=… ✓




```
SELECT u.email, COUNT(l.id) FROM users u
LEFT JOIN posts p ON u.email=p.user_email
LEFT JOIN likes l ON p.id=l.post_id
GROUP BY u.email;
```

users
email PK
first_name
last_name

posts
id PK
user_email
text

likes
id PK
user_email
post_id

likes_by_post_id
post_id PK
id CK
user_email

posts_by_user_email
user_email PK
id CK
text

SELECT id FROM posts_by_user_email WHERE user_email = …

SELECT COUNT(*) FROM likes_by_post_id WHERE post_id IN (…)

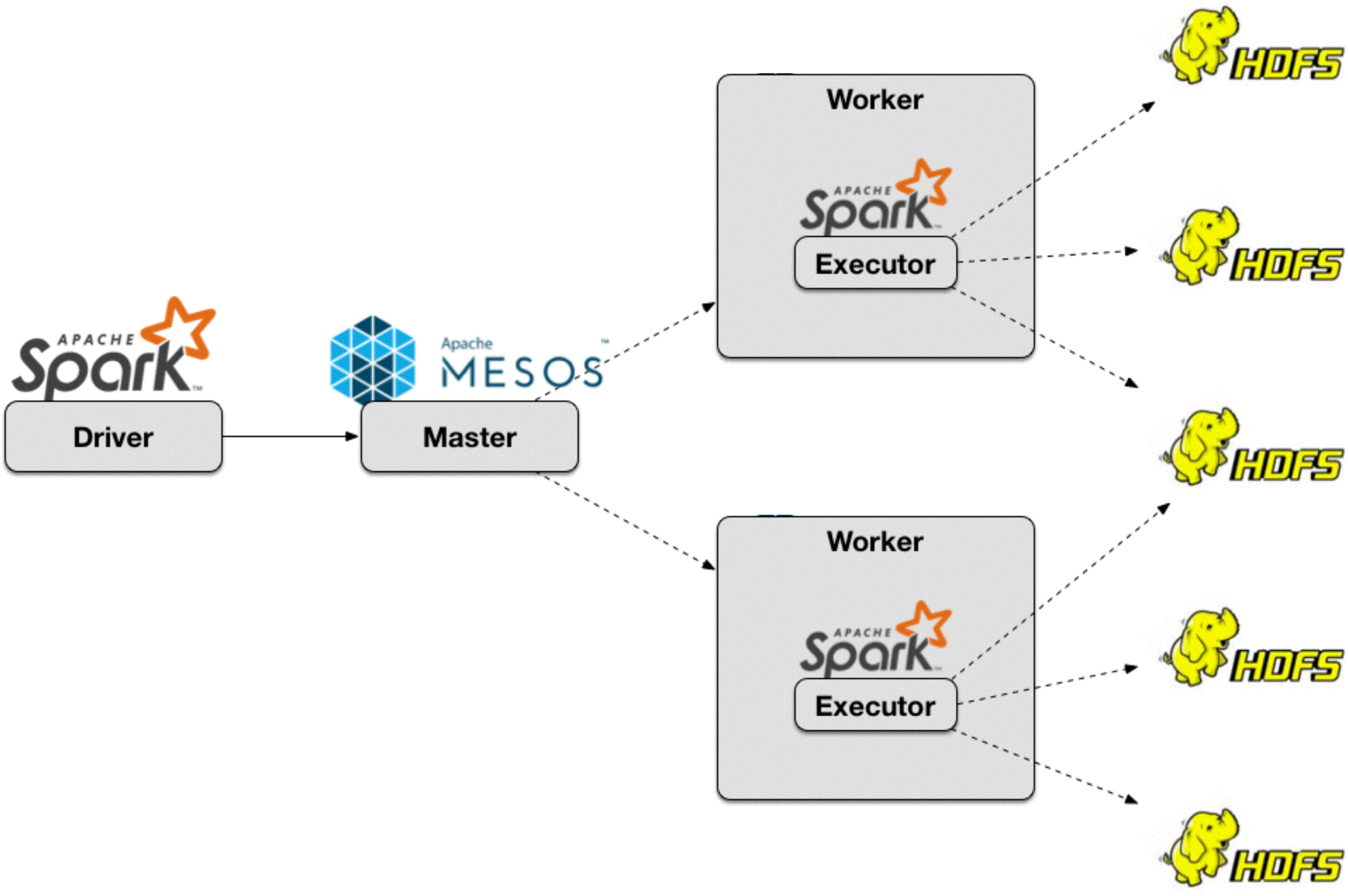

APACHE
Spark

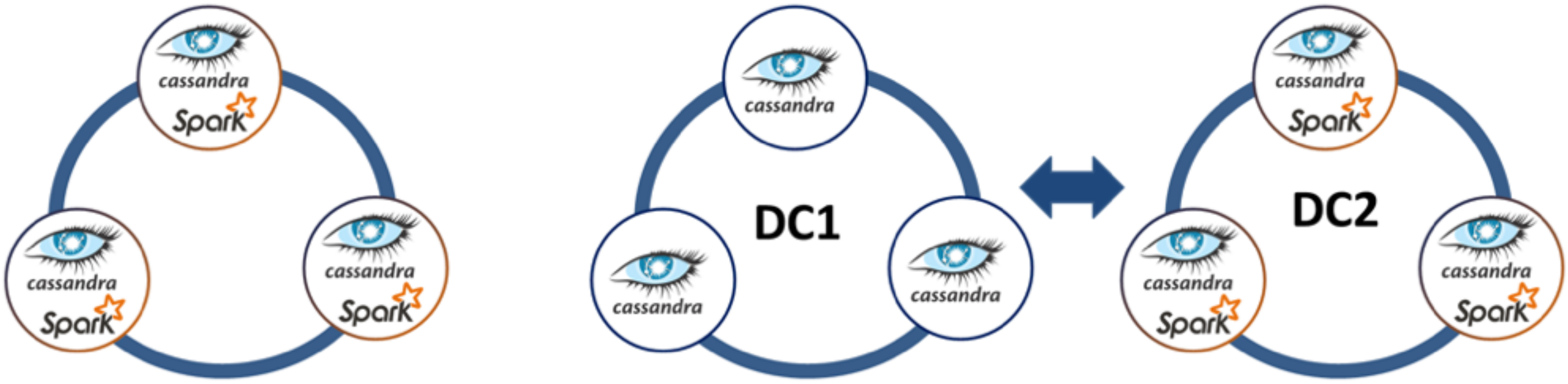

hadoop


cassandra

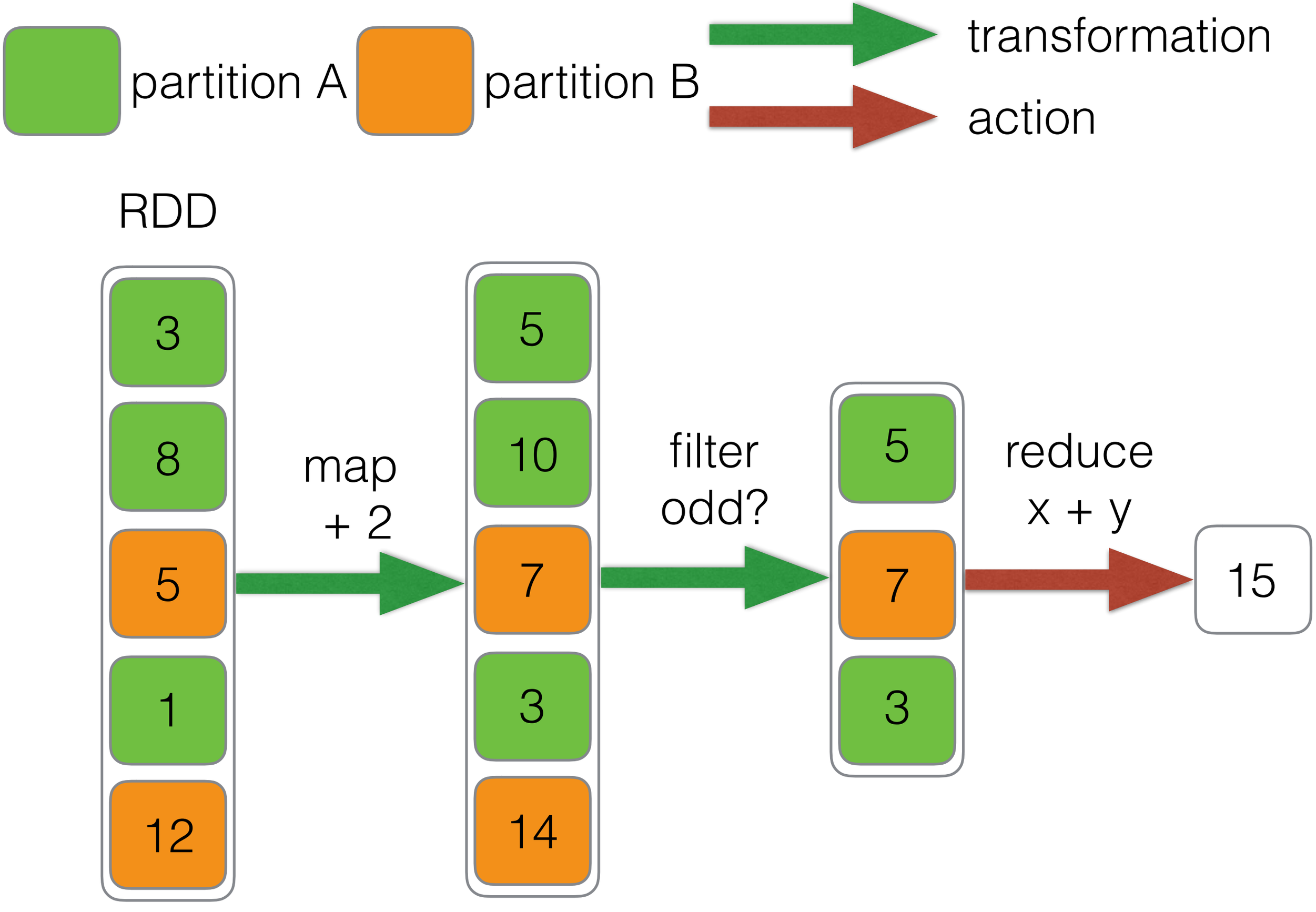

MESOS


APACHE
HBASE


Worker
Executor
Worker
Executor
Driver
Master
HDFS
HDFS
HDFS
HDFS
HDFS
HDFS



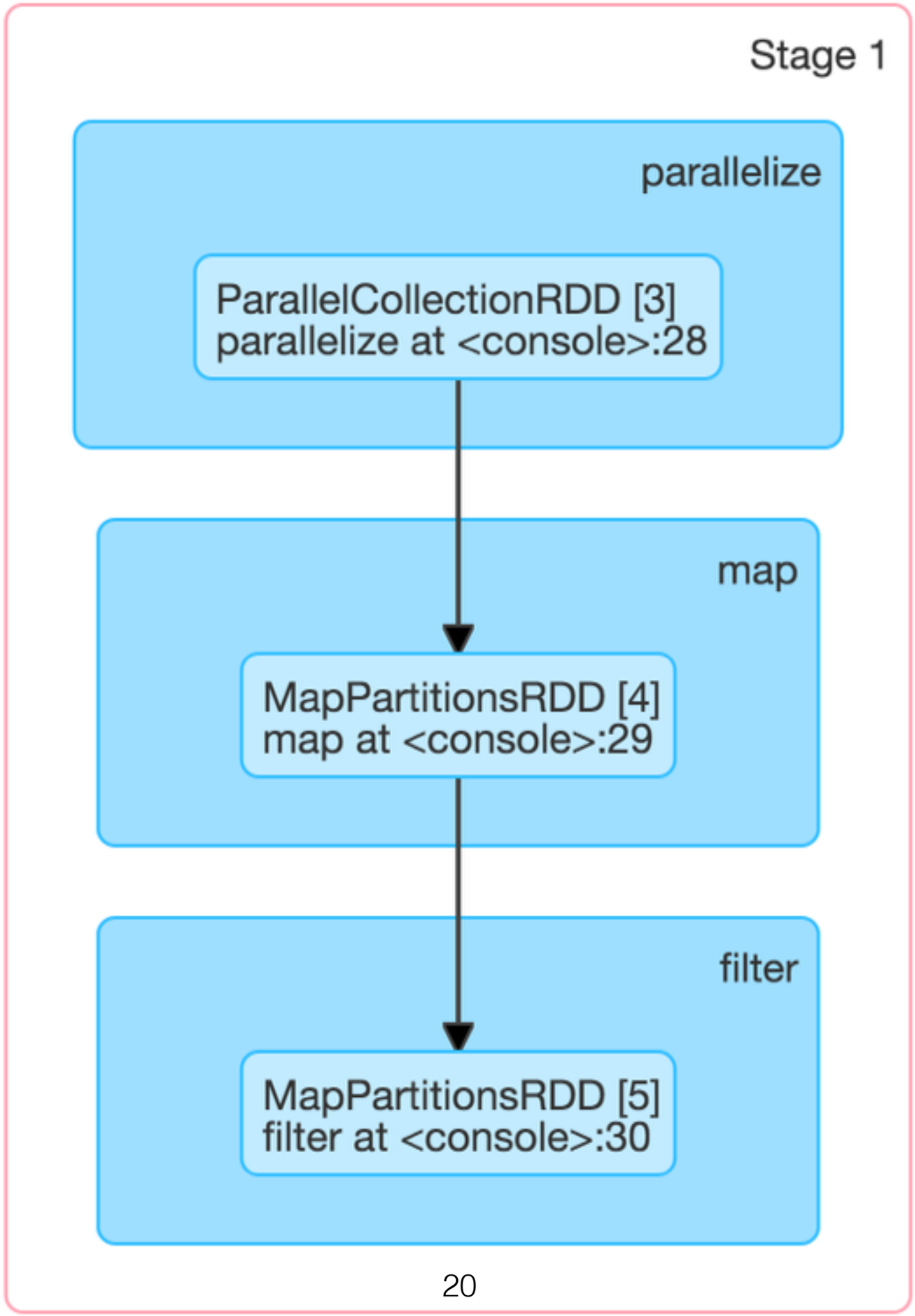
Single DC **V.S.** **Multi DC**


partition A
partition B
transformation
action
RDD
3
8
5
1
12
map
+ 2
5
10
7
3
14
filter
odd?
5
7
3
reduce
x + y
15

Code samples

```java
SparkConf sparkConf = new SparkConf().setMaster("local[*]").setAppName("simple-app");
JavaSparkContext sparkContext = new JavaSparkContext(sparkConf);
List<Integer> elements = Arrays.asList(3, 8, 5, 1, 12);
int result = sparkContext.parallelize(elements)
        .map(x -> x + 2)
        .filter(x -> x % 2 == 1)
        .reduce((x, y) -> x + y);
```

```kotlin
val sparkContext = JavaSparkContext(SparkConf().apply {
    setMaster("local[*]")
    setAppName("simple-app")
})
val elements = listOf(3, 8, 5, 1, 12)
val result = sparkContext.parallelize(elements)
        .map { x -> x + 2 }
        .filter { x -> x % 2 == 1 }
        .reduce { x, y -> x + y }
```


Stage 1
parallelize
ParallelCollectionRDD [3]
parallelize at <console>:28
map
MapPartitionsRDD [4]
map at <console>:29
filter
MapPartitionsRDD [5]
filter at <console>:30

Spark API

Transformations

- map
- filter
- distinct
- sortByKey
- groupByKey
- union
- **join**

Actions

- reduce
- collect
- count
- first
- foreach

Spark Cassandra connector

Load table

```
val sc = new SparkContext(sparkConf)
val rdd = sc.cassandraTable("test", "kv")
println(rdd.count)
println(rdd.first)
println(rdd.map(_.getInt("value")).sum)
```

Save table

```
val collection = sc.parallelize(Seq(("key3", 3), ("key4", 4)))
collection.saveToCassandra("test", "kv", SomeColumns("key", "value"))
```

github.com/datastax/spark-cassandra-connector

Java API analogs

Load table

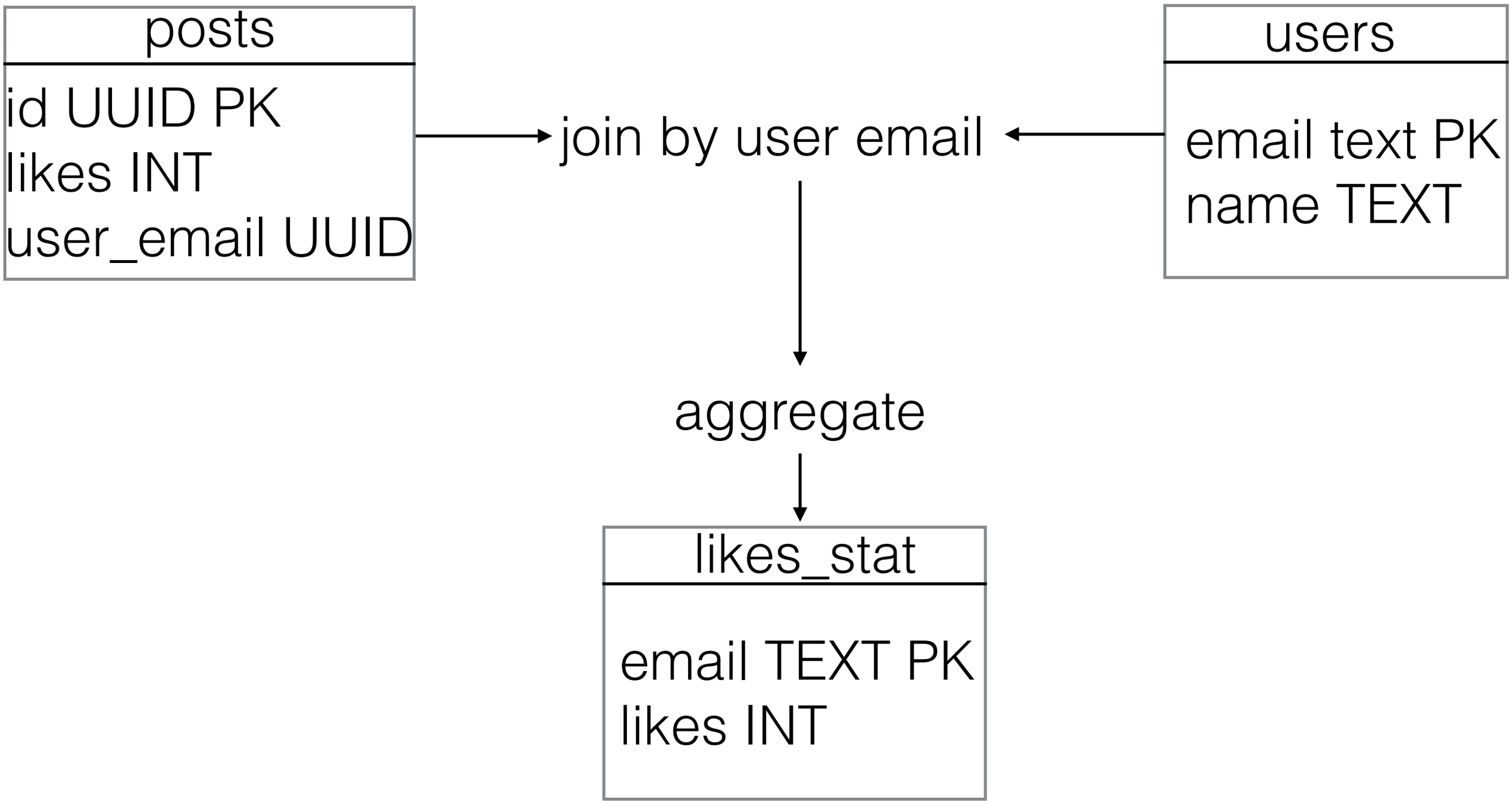
```
JavaRDD<CassandraRow> rdd = CassandraJavaUtil.javaFunctions(sparkContext).cassandraTable("test", "kv");
System.out.println(rdd.count());
System.out.println(rdd.first());
System.out.println(rdd.map(x -> x.getInt("value")).reduce((x, y) -> x + y));
```

Save table

```
class KeyValue implements Serializable {
    // constructor

    String key;
    Integer value;
}
JavaRDD<KeyValue> collection = sparkContext.parallelize(Arrays.asList(new KeyValue("key3", 3),
new KeyValue("key4", 4)));
CassandraJavaUtil.javaFunctions(collection)
                 .writerBuilder(“test”,"kv", CassandraJavaUtil.mapToRow(KeyValue.class))
                 .saveToCassandra();
```

Simple aggregation



Java API

```
JavaRDD<PostUserJoin> postUserJoin = CassandraJavaUtil.javaFunctions(sparkContext)
        .cassandraTable("test", "posts", CassandraJavaUtil.mapRowTo(Post.class))
        .map(post -> new PostUserJoin(post.getUserEmail(), post.getLikes()));
JavaPairRDD<PostUserJoin, User> joinedTables = CassandraJavaUtil.javaFunctions(postUserJoin)
        .joinWithCassandraTable("test", "users",
                CassandraJavaUtil.allColumns,
                CassandraJavaUtil.someColumns("email"),
                CassandraJavaUtil.mapRowTo(User.class),
                CassandraJavaUtil.mapToRow(PostUserJoin.class));
JavaRDD<LikesStatistic> likesStatisticRDD = joinedTables
        .mapToPair(x -> new Tuple2<>(x._2().getEmail(), x._1().getLikes()))
        .reduceByKey((x, y) -> x + y)
        .map(x -> new LikesStatistic(x._1(), x._2()));
CassandraJavaUtil.javaFunctions(likesStatisticRDD)
        .writerBuilder("test", "likes_stat", CassandraJavaUtil.mapToRow(LikesStatistic.class))
        .saveToCassandra();
```

Same on Kotlin

```
val postUserJoin = CassandraJavaUtil.javaFunctions(sparkContext).cassandraTable("test", "posts",
        CassandraJavaUtil.mapRowTo(Post::class.java))
        .map { post -> PostUserJoin(post.userEmail, post.likes) }
val joinedTables = CassandraJavaUtil.javaFunctions(postUserJoin).joinWithCassandraTable("test", "users",
        CassandraJavaUtil.allColumns,
        CassandraJavaUtil.someColumns("email"),
        CassandraJavaUtil.mapRowTo(User::class.java),
        CassandraJavaUtil.mapToRow(PostUserJoin::class.java))
val likesStatisticRDD = joinedTables
        .mapToPair { x -> Tuple2(x._2.email, x._1.likes) }
        .reduceByKey { x, y -> x + y }
        .map { x -> LikesStatistic(x._1, x._2) }
CassandraJavaUtil.javaFunctions(likesStatisticRDD).writerBuilder("test", "likes_stat",
        CassandraJavaUtil.mapToRow(LikesStatistic::class.java)).saveToCassandra()
```

Kotlin DSL

```
sparkContext.cassandraTable<Post>("test", "posts")
.cassandraJoin().with<User>("test", "users", Columns("email" to "userEmail"))
.mapToPair { Tuple(it._2.email, it._1.likes) }
.reduceByKey { x, y -> x + y }
.map { LikesStatistic(email = it._1, likes = it._2) }
.saveToCassandra("test", "likes_stat")
```

Kotlin DSL features

- extension functions
- objects
- operators overloading
- functional parameters
- infix functions
- reified generics

Extension functions

Before

```
CassandraJavaUtil.javaFunctions(sparkContext).cassandraTable("test", “posts",
CassandraJavaUtil.mapRowTo(Post.class));
```

After

```
sparkContext.cassandraTable("test", "posts", CassandraJavaUtil.mapRowTo(Post.class))
```

Groovy way

```
String.metaClass.withBrackets = {
    “(“($delegate)”
}
```

```groovy
class ExtendedString {
    static String withBrackets(String text) {
        "($text)"
    }
}

String.mixin(ExtendedString)
```

* deprecated since Groovy 2.3

```groovy
trait ExtendedString {
    abstract def toCharArray()

    String withBrackets() {
        "(${toCharArray()})"
    }
}

def s = ("test" as ExtendedString)
println(s.withBrackets()) // (test)
```

* since Groovy 2.3

Scala way

```scala
implicit def string2extendedString(s: String): ExtendedString = new ExtendedString(s)
```

```scala
class ExtendedString(s: String) {
    def withBrackets() = s"($s)"
}

println("hello".withBrackets()) // (hello)
```

Kotlin way

```
fun String.withBrackets(): String {
    return "($this)"
}
```

Kotlin extension properties

```
val Int.seconds: Duration
    get() = Duration.standardSeconds(this.toLong())
```

example of using

```
sparkContext.cassandraTable<User>("test", "users")
        .map { it.copy(age = it.age + 1) }
        .writeBuilder("test", "users")
        .withConstantTTL(3.seconds)
        .saveToCassandra()
```

Examples of using in standard Kotlin library

```
public fun File.readLines(charset: Charset = Charsets.UTF_8): List<String> {
    val result = ArrayList<String>()
    forEachLine(charset) { result.add(it); }
    return result
}
```

```
public inline fun <T, R> Iterable<T>.map(transform: (T) -> R): List<R> {
    return mapTo(ArrayList<R>(collectionSizeOrDefault(10)), transform)
}
```

Objects and operators overloading

Before

```
rdd.mapToPair(post -> new Tuple2<>(post.id, post.likes));
```

After

```
rdd.mapToPair { post ->  Tuple(post.id, post.likes) }
```

```
object Tuple {
    operator fun<T> invoke(t: T): Tuple1<T> = Tuple1(t)
    operator fun<T1, T2> invoke(t1: T1, t2: T2): Tuple2<T1, T2> = Tuple2(t1, t2)
    operator fun<T1, T2, T3> invoke(t1: T1, t2: T2, t3: T3): Tuple3<T1, T2, T3> = Tuple3(t1, t2, t3)
    …
}
```

Groovy way

Operator	Method	Operator	Method
`+`	a.plus(b)	`a[b]`	a.getAt(b)
`-`	a.minus(b)	`a[b] = c`	a.putAt(b, c)
`*`	a.multiply(b)	`a in b`	b.isCase(a)
`/`	a.div(b)	`<<`	a.leftShift(b)
`%`	a.mod(b)	`>>`	a.rightShift(b)
`**`	a.power(b)	`>>>`	a.rightShiftUnsigned(b)
`|`	a.or(b)	`++`	a.next()
`&`	a.and(b)	`--`	a.previous()
`^`	a.xor(b)	`+a`	a.positive()
`as`	a.asType(b)	`-a`	a.negative()
`a()`	a.call()	`~a`	a.bitwiseNegate()

```groovy
class Complex {
    Double real
    Double img

    def plus(Complex c) {
        new Complex(this.real + c.real, this.img + c.img)
    }
    …
}

def c1 = new Complex(1.5, 2.3)
def c2 = new Complex(1.0, 1.2)

println(c1 + c2) // 2.5 + 3.5i
```

Scala way

```
//Valid
someString //#1
SomeString //#1
a123 //#1
someString_? //#1
a_b_? //#1
? //#2
?+-<>:|!&%#\^@~*_ //#2
`lorem ipsum 123 []` //#3

//Invalid
123 //Doesn't start with a letter
someString? //Ends with an operator char without preceding    underscore
someString_a? //Same as above
?a //Doesn't match #2 as it contains a letter
a_?_b //To match #1 any operator char must come last
```

```scala
class Complex(val real: Double, val img: Double) {
    def +(c: Complex) = new Complex(this.real + c.real, this.img + c.img)

    def ->() = ...
}
```

```java
Complex c1 = new Complex(1.2, -1.0);
Complex c2 = new Complex(2.9, 7.9);
Complex c3 = c1.$plus(c2);
c3.$minus$greater();
```

Edge Category	Edge Class	Shortcut	Description
Hyperedge	`HyperEdge`	~	hyperedge
	`WHyperEdge`	~%	weighted hyperedge
	`WkHyperEdge`	~%#	key-weighted hyperedge
	`LHyperEdge`	~+	labeled hyperedge
	`LkHyperEdge`	~+#	key-labeled hyperedge
	`WLHyperEdge`	~%+	weighted labeled hyperedge
	`WkLHyperEdge`	~%#+	key-weighted labeled hyperedge
	`WLkHyperEdge`	~%+#	weighted key-labeled hyperedge
	`WkLkHyperEdge`	~%#+#	key-weighted key-labeled hyperedge

```
val flights = Graph(
        (jfc ~+#> fra)(Flight("LH 400" ,10 o 25, 8 h 20)),
        (fra ~+#> dme)(Flight("LH 1444", 7 o 50, 3 h 10))
```

*from scala-graph library docs

Kotlin way

Binary operations

Expression	Translated to
`a + b`	`a.plus(b)`
`a - b`	`a.minus(b)`
`a * b`	`a.times(b)`
`a / b`	`a.div(b)`
`a % b`	`a.mod(b)`
`a..b`	`a.rangeTo(b)`

Symbol	Translated to
`a[i]`	`a.get(i)`
`a[i, j]`	`a.get(i, j)`
`a[i_1, ..., i_n]`	`a.get(i_1, ..., i_n)`
`a[i] = b`	`a.set(i, b)`
`a[i, j] = b`	`a.set(i, j, b)`
`a[i_1, ..., i_n] = b`	`a.set(i_1, ..., i_n, b)`

Unary operations

Expression	Translated to
`+a`	`a.unaryPlus()`
`-a`	`a.unaryMinus()`
`!a`	`a.not()`

```
class Complex(val real: Double, val img: Double) {
    operator fun plus(c: Complex) = Complex(this.real + c.real, this.img + c.img)
}
```

Examples of using in standard Kotlin library

```
public open class Any public constructor() {
    public open operator fun equals(other: kotlin.Any?): kotlin.Boolean { /* compiled code */ }
    …
}
```

```
public final class Array<T> : kotlin.Cloneable {
    public final operator fun get(index: kotlin.Int): T { /* compiled code */ }
    public final operator fun set(index: kotlin.Int, value: T): kotlin.Unit { /* compiled code */ }
    …
}
```

Reified generics

Before

```
val usersTable = sparkContext.cassandraTable("test", "users", CassandraJavaUtil.mapRowTo(User::class.java))
userTable.saveToCassandra("test", "users", CassandraJavaUtil.mapToRow(User::class.java))
```

After

```
val usersTable = sparkContext.cassandraTable<User>("test", "users")
usersTable.saveToCassandra("test", "users")
```

Scala way

Manifest

ClassTag

TypeTag

WeakTypeTag

```scala
def printClassInfo[T](t: T)(implicit tag: TypeTag[T]) = {
    println(tag.tpe)
}
```

```scala
val a: Array[Seq[List[String]]] = null
printClassInfo(a) // Array[Seq[List[String]]]
```

```
def printClassInfo[T](t: T)(implicit tag: TypeTag[T]) = {
    println(tag.tpe)
}

def foo[T](t: T) = printClassInfo(t) // compilation error!
```

```
val a: Array[Seq[List[String]]] = null
foo(a)
```

```scala
def printClassInfo[T](t: T)(implicit tag: WeakTypeTag[T]) = {
    println(tag.tpe)
}

def foo[T](t: T) = printClassInfo(t)

val a: Array[Seq[List[String]]] = null
foo(a) // T
```

Kotlin way

```
inline fun <reified T : Any> JavaRDD<T>.saveToCassandra(keyspace: String, table: String) {
    CassandraJavaUtil.javaFunctions(this)
            .writerBuilder(keyspace, table, CassandraJavaUtil.mapToRow(T::class.java)).saveToCassandra()
}
```

reified generics limitations

- Only a type parameter of an *inline* function can be marked *reified*

- The built-in class *Array* is the only class whose type parameter is marked *reified*. Other classes are not allowed to declare *reified* type parameters.

- Only a runtime-available type can be passed as an argument to a *reified* type parameter

Runtime-available types examples:

- String, Int, File…
- Array<String>, Array<Int>…
- List<*>
- T (if T is reified)

Non-runtime-available types examples:

- Nothing
- List<String>
- T (if T isn’t reified)

Examples of using in standard Kotlin library

```
public inline fun <reified T> arrayOf(vararg elements: T): kotlin.Array<T> { /* compiled code */ }

public fun <reified T> arrayOfNulls(size: kotlin.Int): kotlin.Array<T?> { /* compiled code */ }
```

Valid

```
fun transform(array: Array<Int?>) {…}

transform(arrayOfNulls(5))
```

Invalid

```
fun transform(array: Array<*>) {…}

transform(arrayOfNulls(5))
```

Examples of using in libraries

jackson-module-kotlin

```
inline fun <reified T: Any> ObjectMapper.readValue(jp: JsonParser): T = readValue(jp, object: TypeReference<T>() {})

val state = mapper.readValue<MyStateObject>(json)

// or

val state: MyStateObject = mapper.readValue(json)
```

RxKotlin

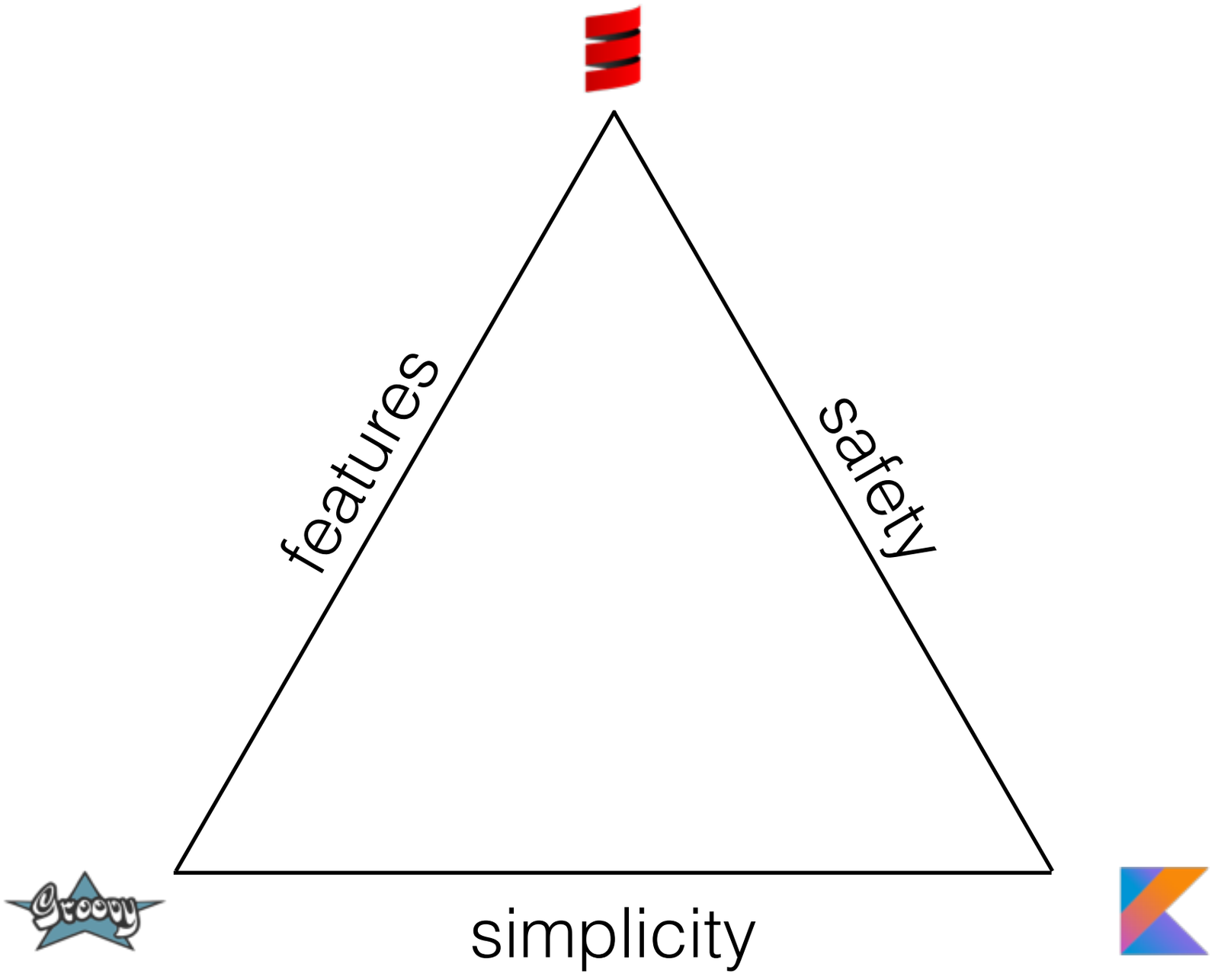
```
inline fun <reified R : Any> Observable<*>.cast(): Observable<R> = cast(R::class.java)

val a: Observable<String> = createStringObservable()
val b: Observable<Int> = a.cast()

// or

val b = a.cast<Int>()
```

Conclusions



MAKE KOTLIN

LOVE KATS