Deep Dive: Memory Management in Apache Spark

Andrew Or
@andrewor14
May 18th, 2016

DATA BY THE BAY

How familiar are you with Apache Spark?

a) I contribute to it

b) I use it in production

c) I am evaluating it

d) I have nothing to do with it

What is Apache Spark™?

Fast and general engine for big data processing

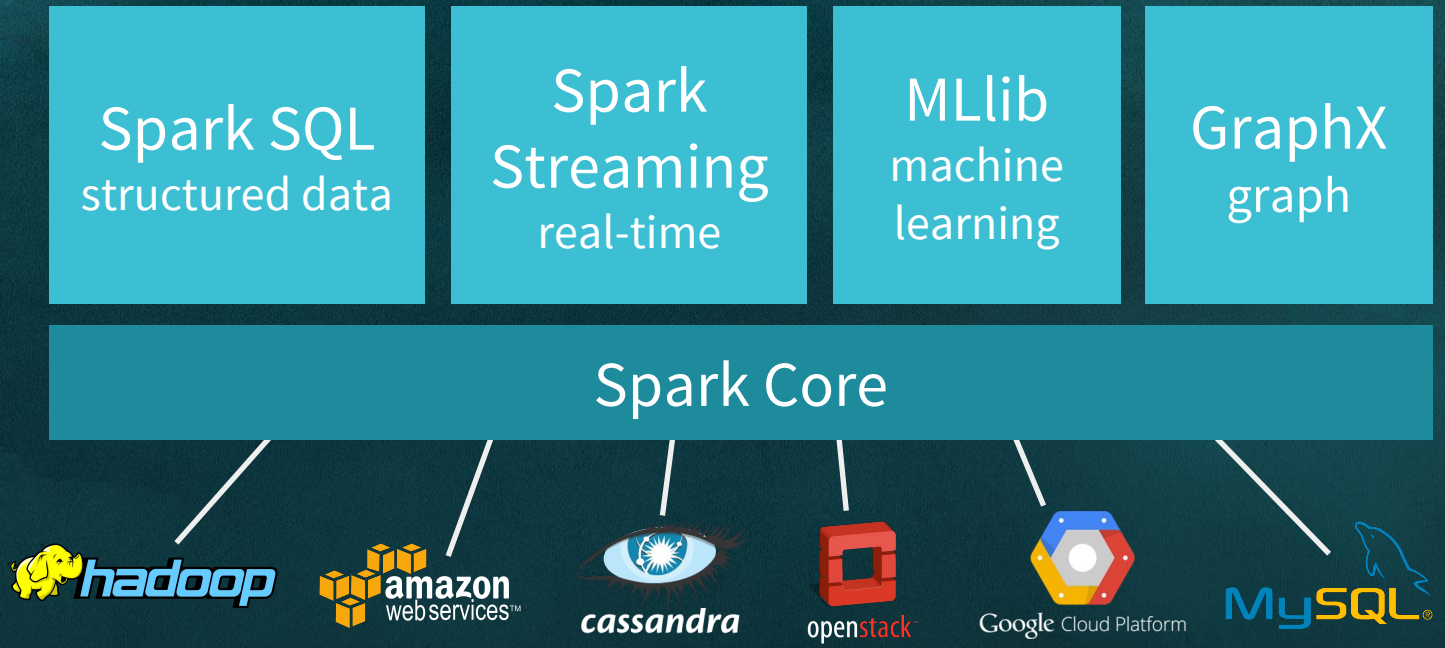
Fast to *run* code

- In-memory data sharing
- General computation graphs

Fast to *write* code

- Rich APIs in Java, Scala, Python
- Interactive shell

What is Apache Spark™?

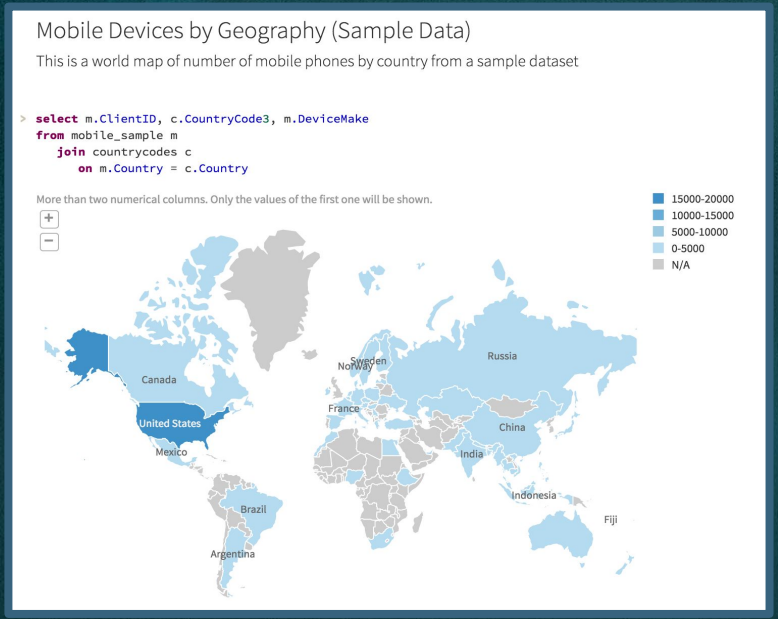


About Databricks

Team that created Spark at UC Berkeley

Offer a hosted service

- Spark in the cloud
- Notebooks
- Plot visualizations
- Cluster management



About Me

Apache Spark committer

Software eng @ Databricks

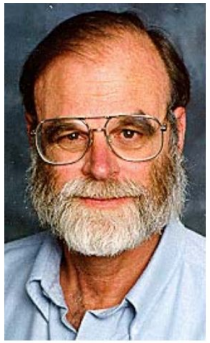
Hadoop Summit ‘15

Spark Summit Europe ‘15

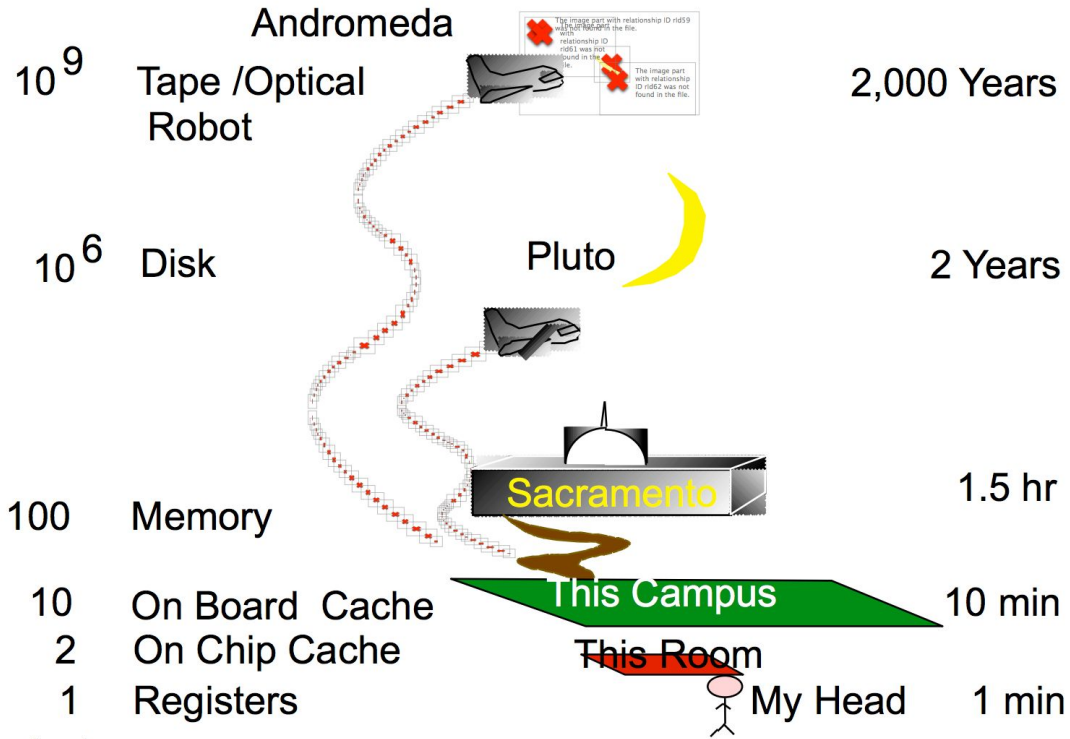
Some other meetup talks

Efficient memory use is critical to *good performance*

Jim Gray's Storage Latency Analogy: How Far Away is the Data?



Jim Gray
Turing Award
B.S. Cal 1966
Ph.D. Cal 1969!



Memory contention poses three challenges for Apache Spark

How to arbitrate memory between execution and storage?

How to arbitrate memory across tasks running in parallel?

How to arbitrate memory across operators running within the same task?

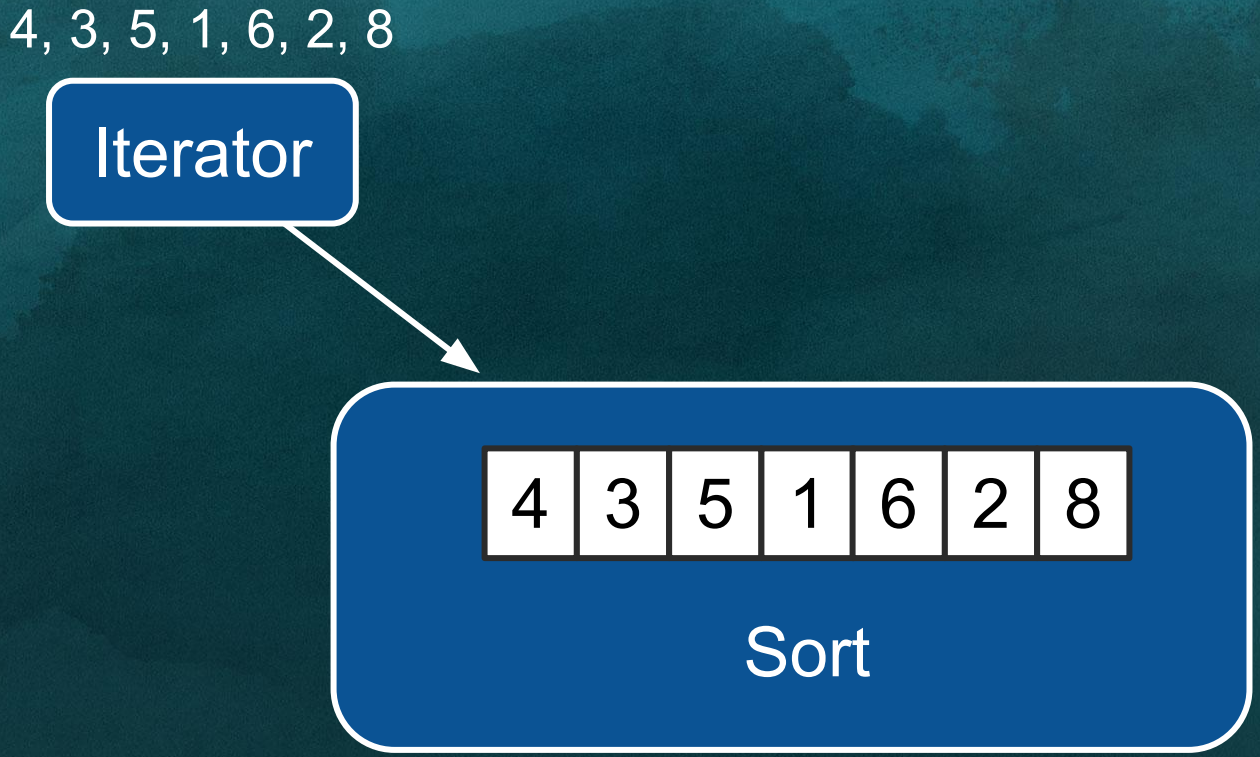
Two usages of memory in Apache Spark

Execution

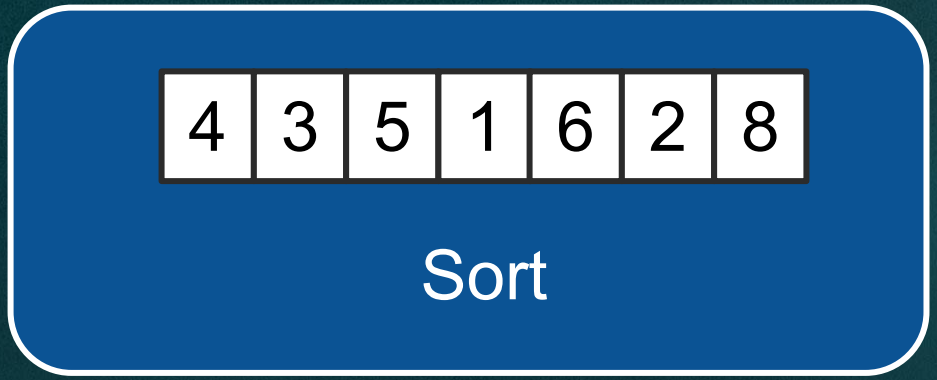
Memory used for shuffles, joins, sorts and aggregations

Storage

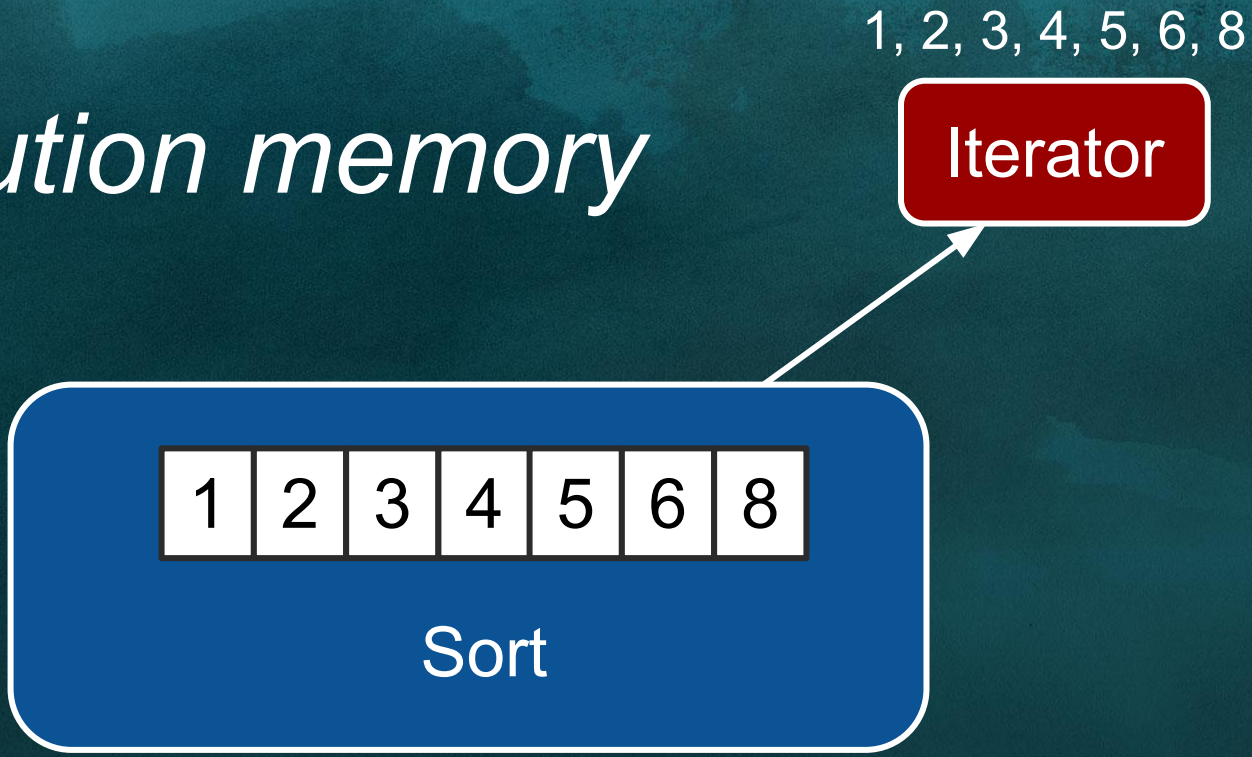
Memory used to cache data that will be reused later


4, 3, 5, 1, 6, 2, 8
Iterator
4
3
5
1
6
2
8
Sort

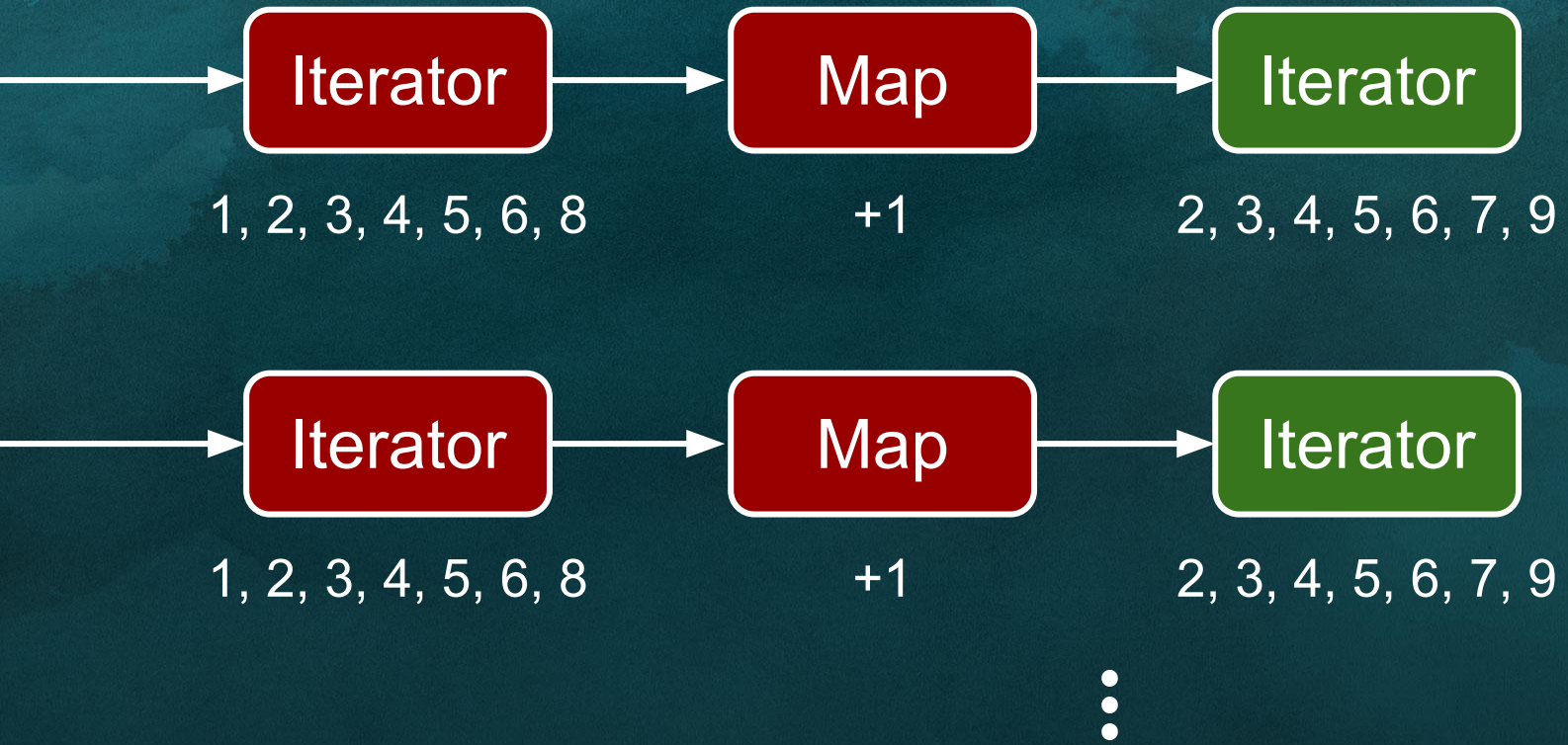
Execution memory

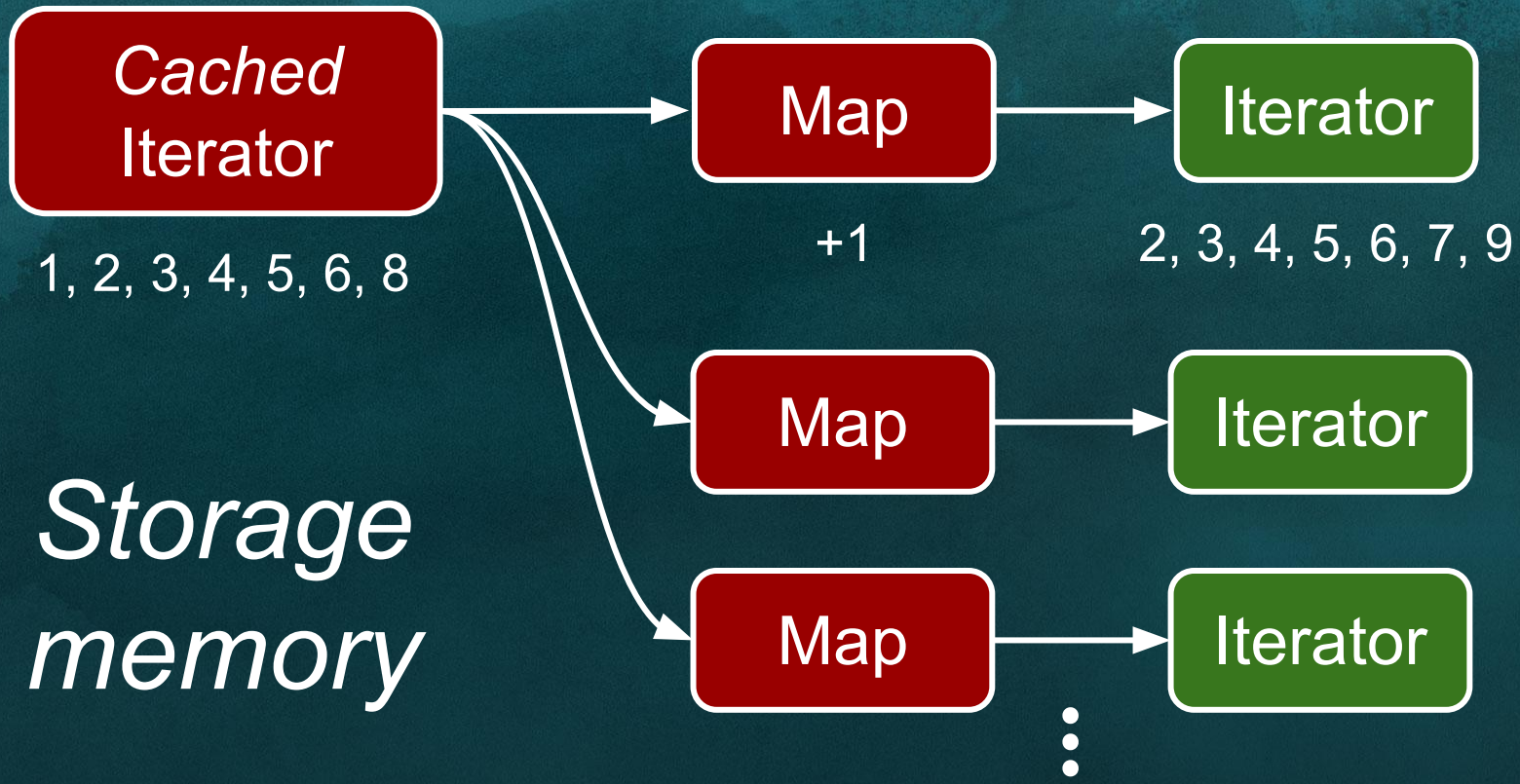


Execution memory




Iterator
Map
Iterator
1, 2, 3, 4, 5, 6, 8
+1
2, 3, 4, 5, 6, 7, 9


Iterator
Map
Iterator
1, 2, 3, 4, 5, 6, 8
+1
2, 3, 4, 5, 6, 7, 9
Iterator
Map
Iterator
1, 2, 3, 4, 5, 6, 8
+1
2, 3, 4, 5, 6, 7, 9


Cached Iterator
1, 2, 3, 4, 5, 6, 8
Map
+1
Iterator
2, 3, 4, 5, 6, 7, 9
Map
Iterator
Map
Iterator
Storage memory

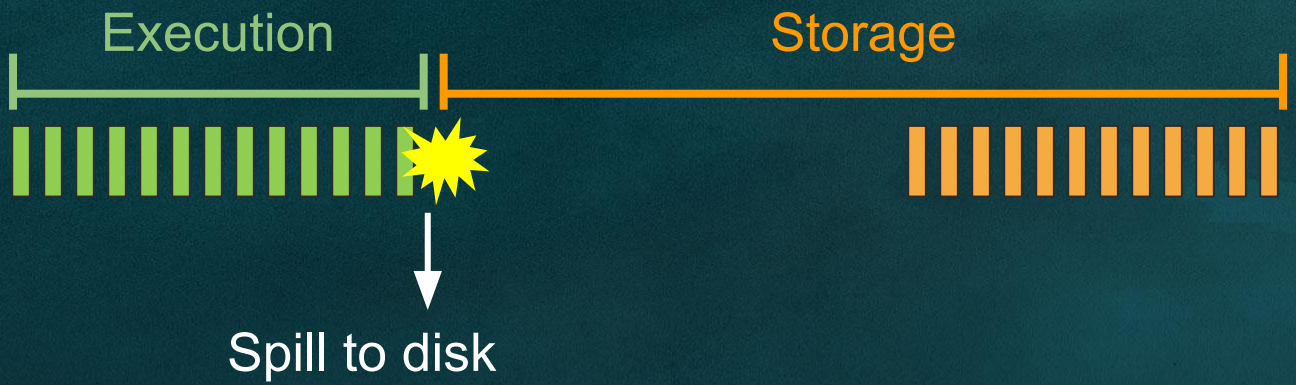
Challenge #1

How to arbitrate memory between execution and storage?

Easy, static allocation!

Spark 1.0
May 2014

Easy, static allocation!



Spark 1.0
May 2014

Easy, static allocation!

Spark 1.0
May 2014

Easy, static allocation!

Spark 1.0
May 2014

Inefficient memory use means

bad performance

Easy, static allocation!

Spark 1.0
May 2014

Execution can only use a fraction of the memory, even when there is no storage!

Easy, static allocation!

Spark 1.0
May 2014

Efficient use of memory required user tuning

Fast forward to 2016…

How could we have done better?

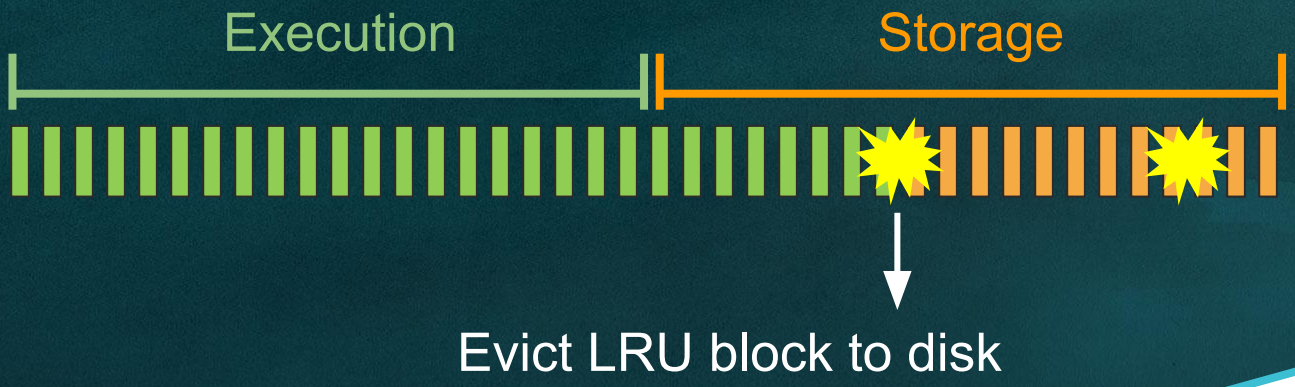

Execution
Storage

Unified memory management

What happens if there is already storage?

Spark 1.6+
Jan 2016

Unified memory management



Spark 1.6+
Jan 2016

Unified memory management

What about the other way round?

Spark 1.6+
Jan 2016

Unified memory management

Spark 1.6+
Jan 2016

Design considerations

Why evict storage, not execution?

Spilled execution data will always be read back from disk, whereas cached data may not.

What if the application relies on caching?

Allow the user to specify a minimum unevictable amount of cached data (not a reservation!).

Spark 1.6+
Jan 2016

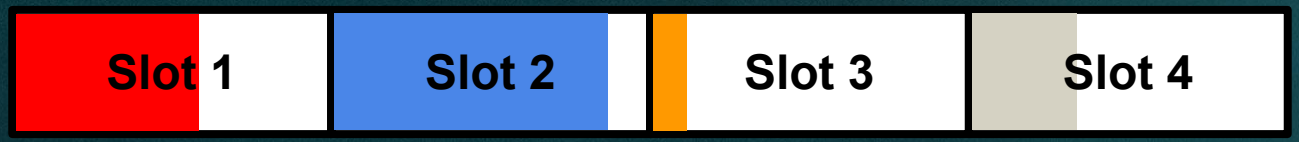
Challenge #2

How to arbitrate memory across tasks running in parallel?

Easy, static allocation!

Worker machine has 4 cores

Each task gets 1/4 of the total memory



Alternative: What Spark does

Worker machine has 4 cores

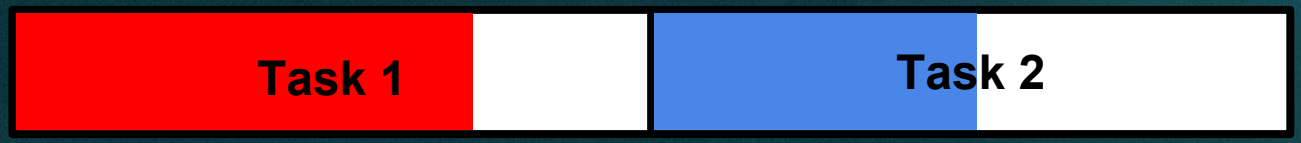
The share of each task depends on number of actively running tasks (N)

Alternative: What Spark does

Now, another task comes along
so the first task will have to spill

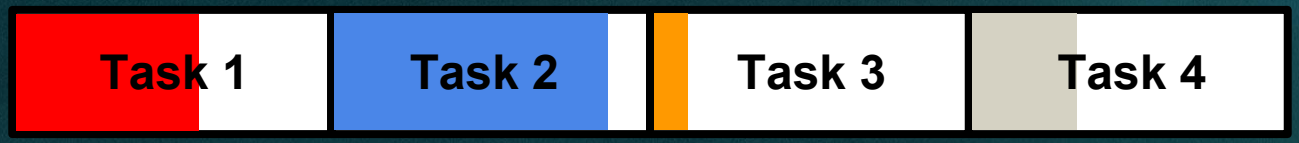
Alternative: What Spark does

Each task is assigned 1/N of the memory, where N = 2



Alternative: What Spark does

Each task is assigned 1/N of the memory, where N = 4



Alternative: What Spark does

Last remaining task gets all the memory because N = 1

Spark 1.0+
May 2014

Static allocation vs What Spark does

Both are fair and starvation free

Static allocation is simpler

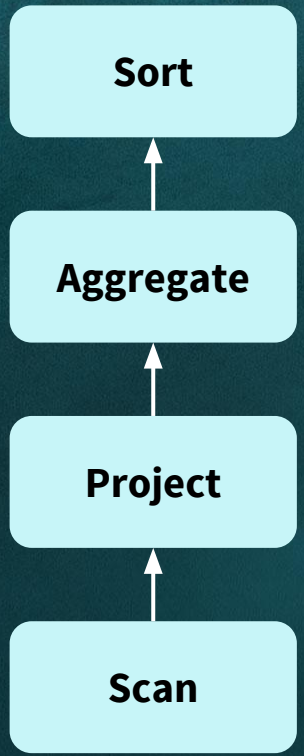
What Spark does handles stragglers better

Challenge #3

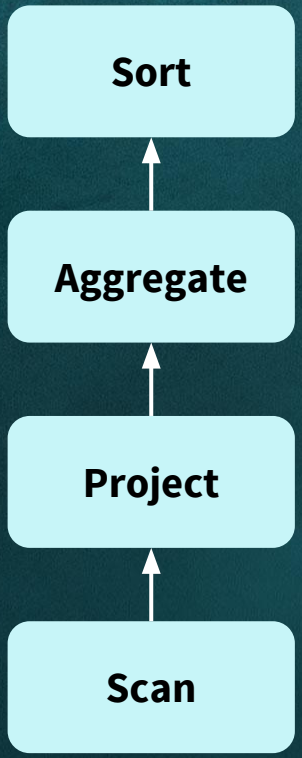
How to arbitrate memory across operators running within the same task?

```
SELECT age, avg(height)
  FROM students
  GROUP BY age
  ORDER BY avg(height)
```

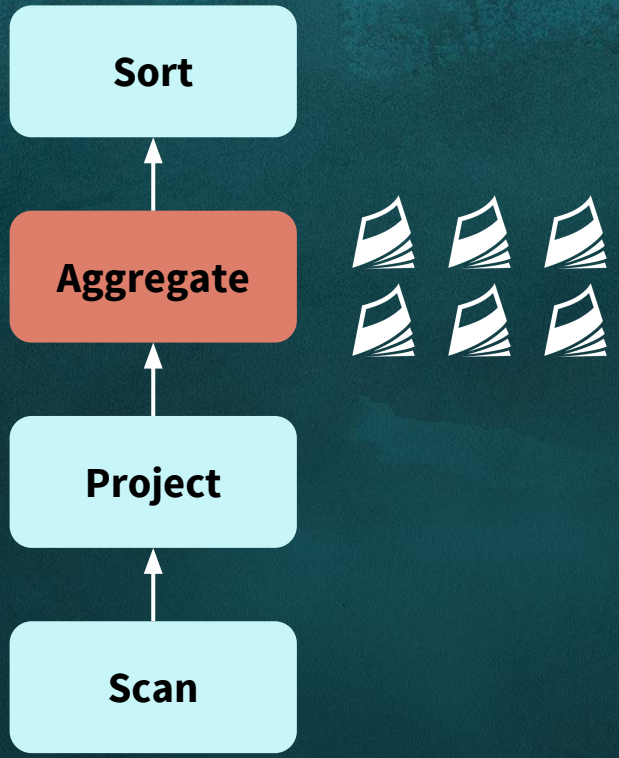
```
students.groupBy("age")
  .avg("height")
  .orderBy("avg(height)")
  .collect()
```



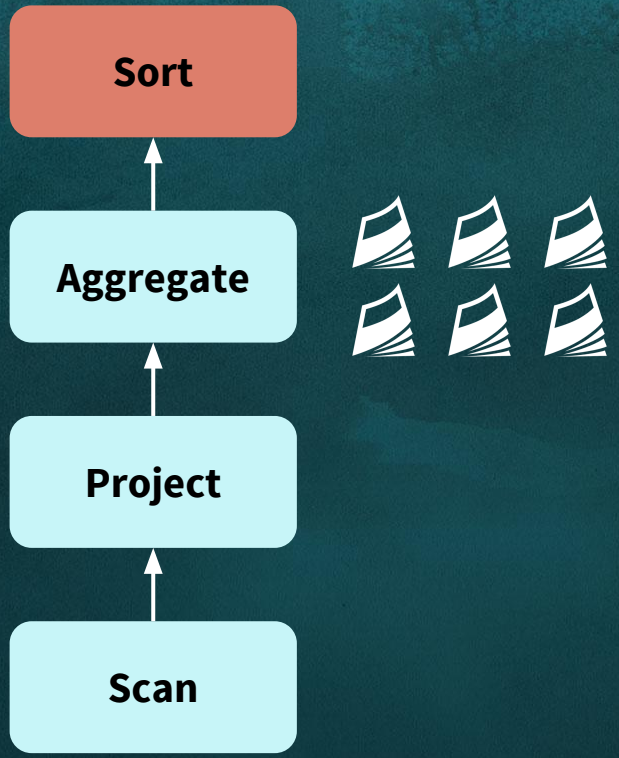
Worker has 6 pages of memory



```
Map { // age → heights
  20 → [154, 174, 175]
  21 → [167, 168, 181]
  22 → [155, 166, 188]
  23 → [160, 168, 178, 183]
}
```



All 6 pages were used by *Aggregate*, leaving no memory for *Sort*!

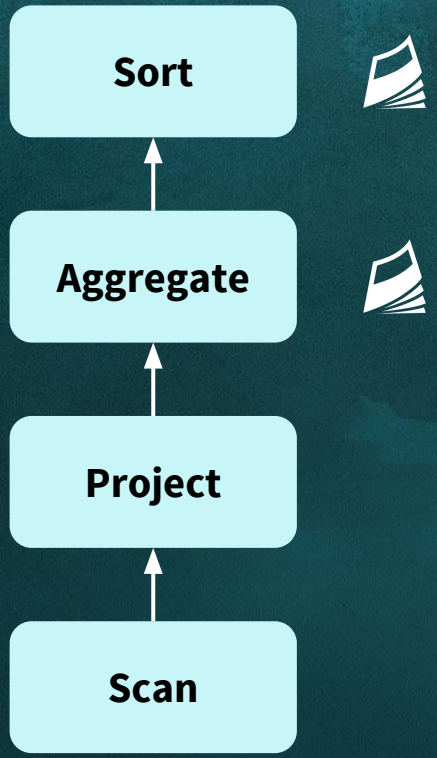


Solution #1:
Reserve a page for each operator

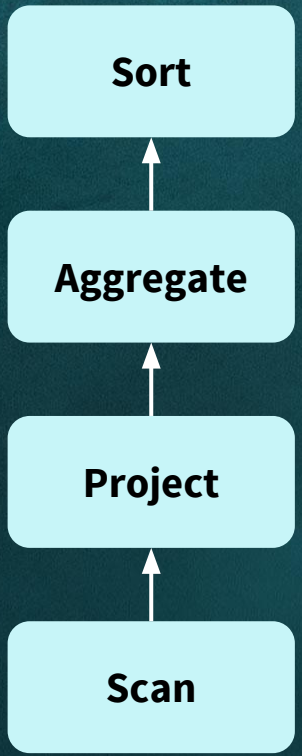
Solution #1:
Reserve a page for each operator

Starvation free, but still not fair…

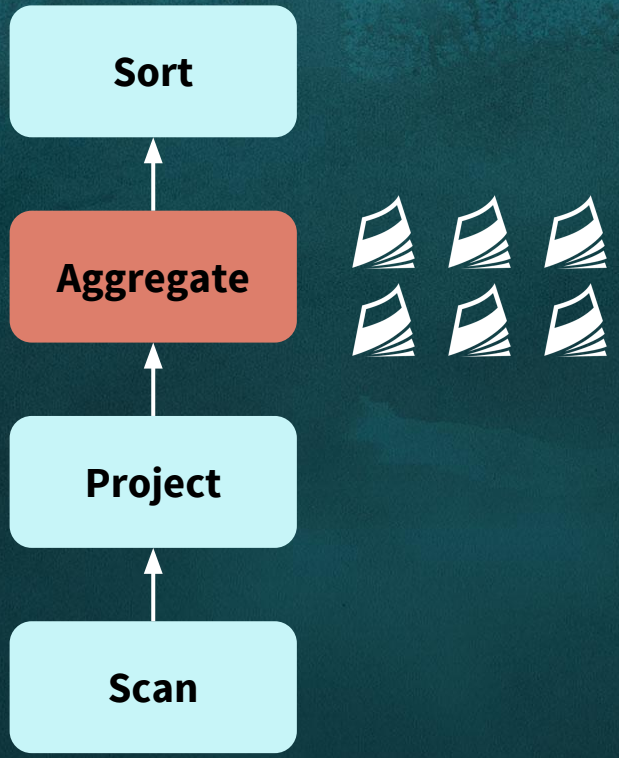
What if there were more operators?



Solution #2:
Cooperative spilling

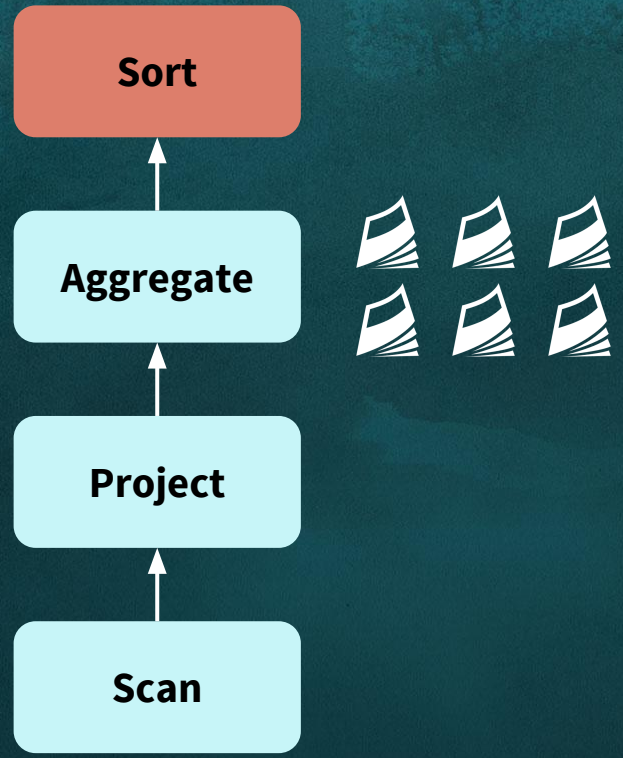


Solution #2:
Cooperative spilling



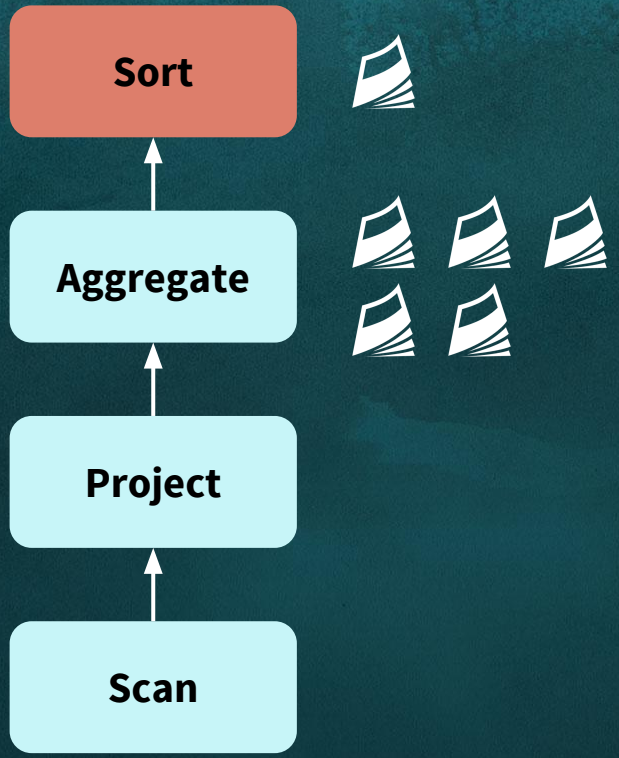
Solution #2:
Cooperative spilling

Sort forces *Aggregate* to spill a page to free memory



Solution #2:
Cooperative spilling

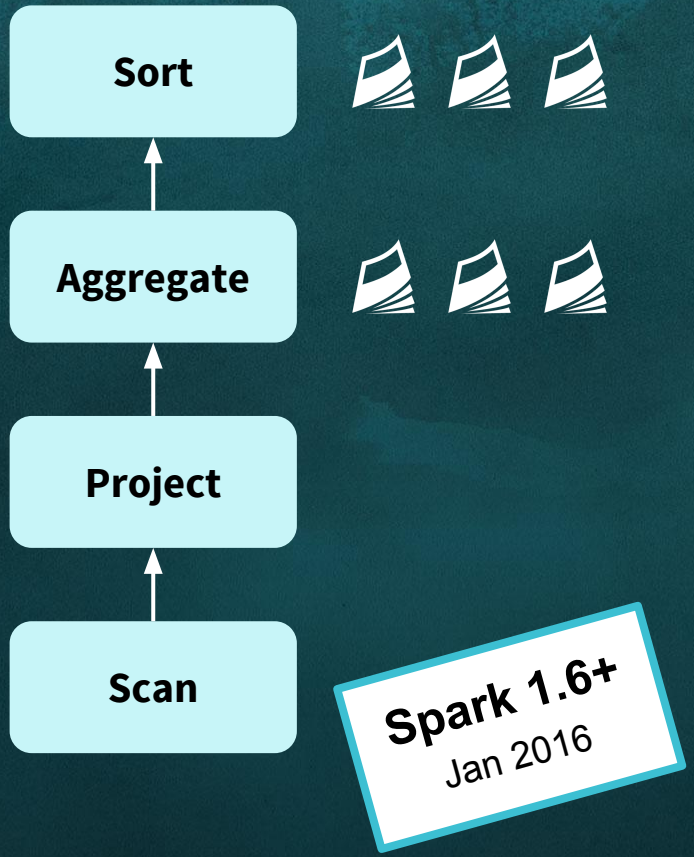
Sort needs more memory so it forces *Aggregate* to spill another page (and so on)



Solution #2:
Cooperative spilling

Sort finishes with 3 pages

Aggregate does not have to spill its remaining pages



Recap: Three sources of contention

How to arbitrate memory …

- between execution and storage?
- across tasks running in parallel?
- across operators running within the same task?

Instead of avoid statically reserving memory in advance, deal with memory contention when it arises by forcing members to spill

Project Tungsten

Binary in-memory data representation

Cache-aware computation

Code generation (next time)

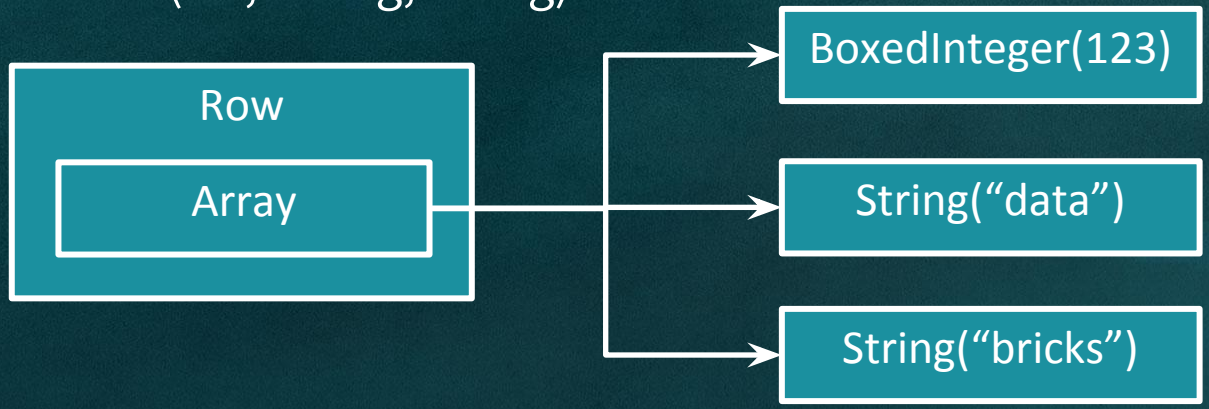
Java objects have large overheads

“abcd”

- Native: 4 bytes with UTF-8 encoding
- Java: 48 bytes
  - 12 byte header
  - 2 bytes per character (UTF-16 internal representation)
  - 20 bytes of additional overhead
  - 8 byte hash code

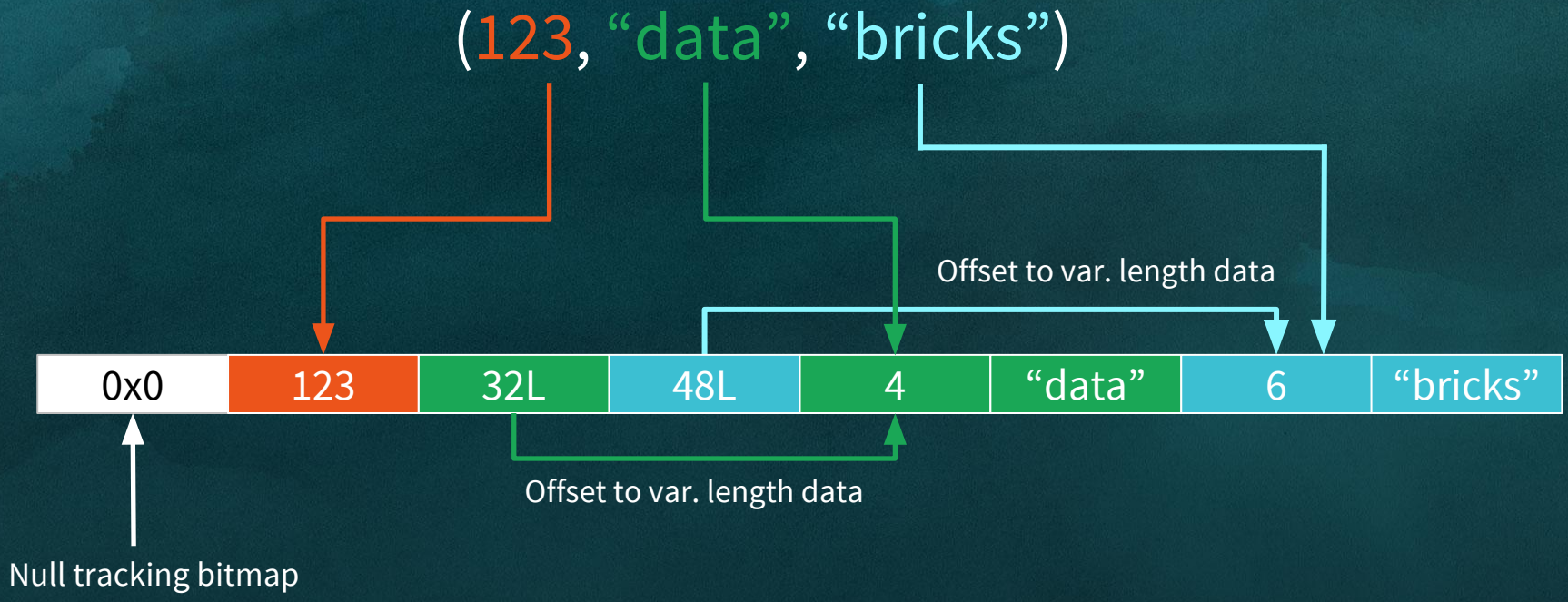
Java objects based row format

Schema: (Int, String, string)



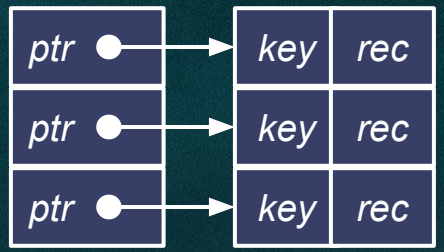
5+ objects, high space overhead, expensive hashCode()

Tungsten row format



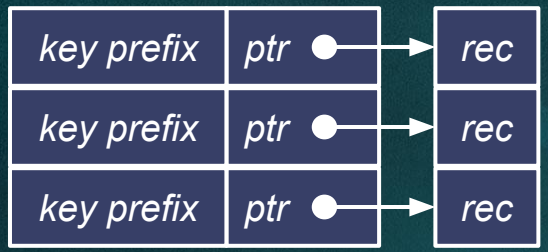
Cache-aware Computation

E.g. sorting a list of records



Naive layout

Poor cache locality



Cache-aware layout

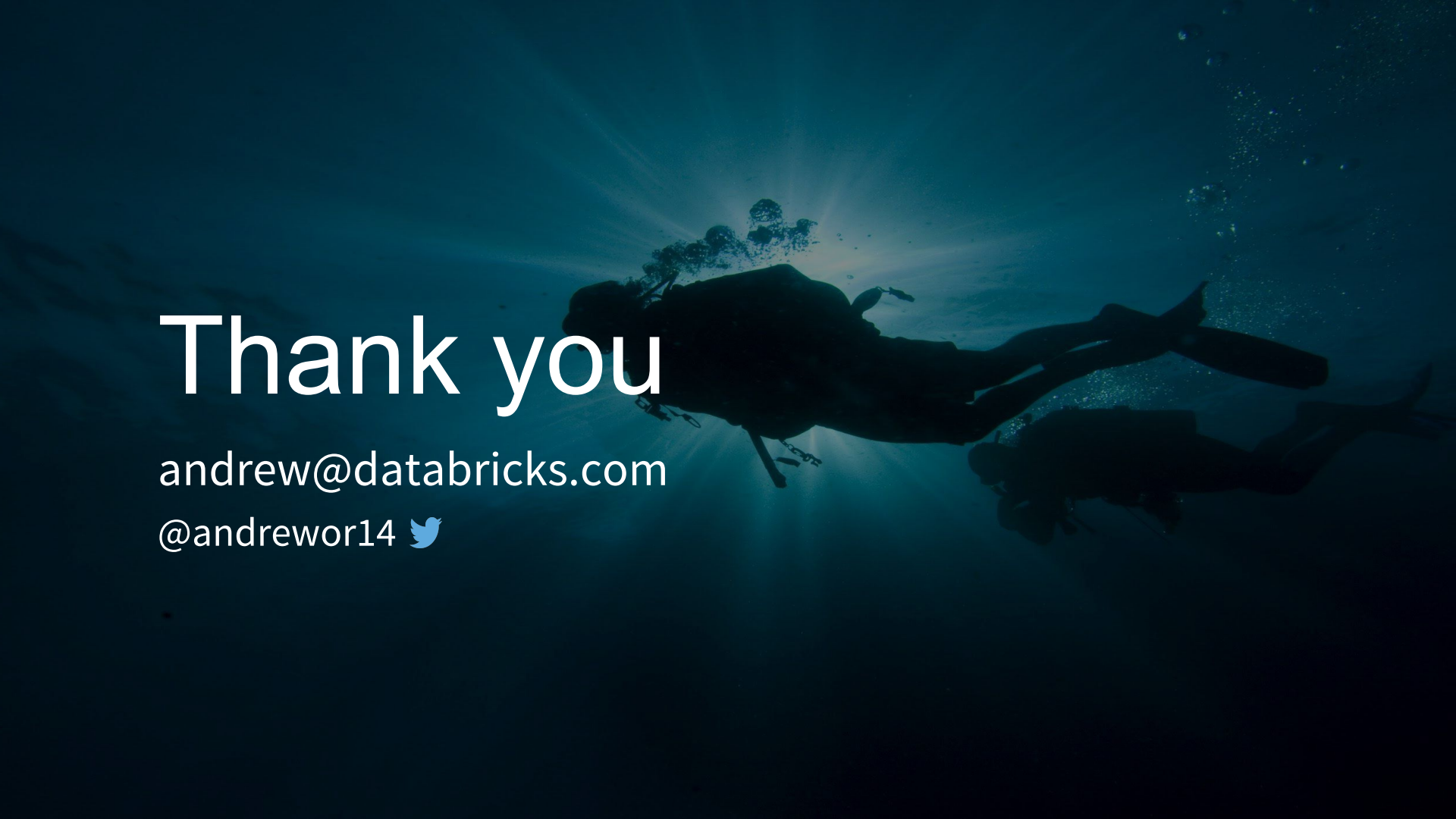
Good cache locality

For more info...

Deep Dive into Project Tungsten: Bringing Spark Closer to Bare Metal
https://www.youtube.com/watch?v=5ajs8EIPWGI

Spark Performance: What's Next
https://www.youtube.com/watch?v=JX0CdOTWYX4

Unified Memory Management
https://issues.apache.org/jira/browse/SPARK-10000


Thank you
andrew@databricks.com
@andrewor14