




Deep Dive: Memory Management in Apache *Spark*

Andrew Or

@andrewor14 

June 8th, 2016



```
students.select("name").orderBy("age").cache().show()
```

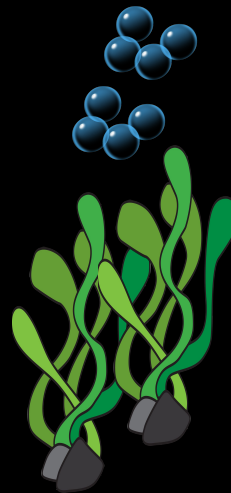


Memory
Contention

Off-heap

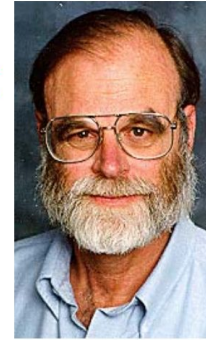
Caching

Tungsten

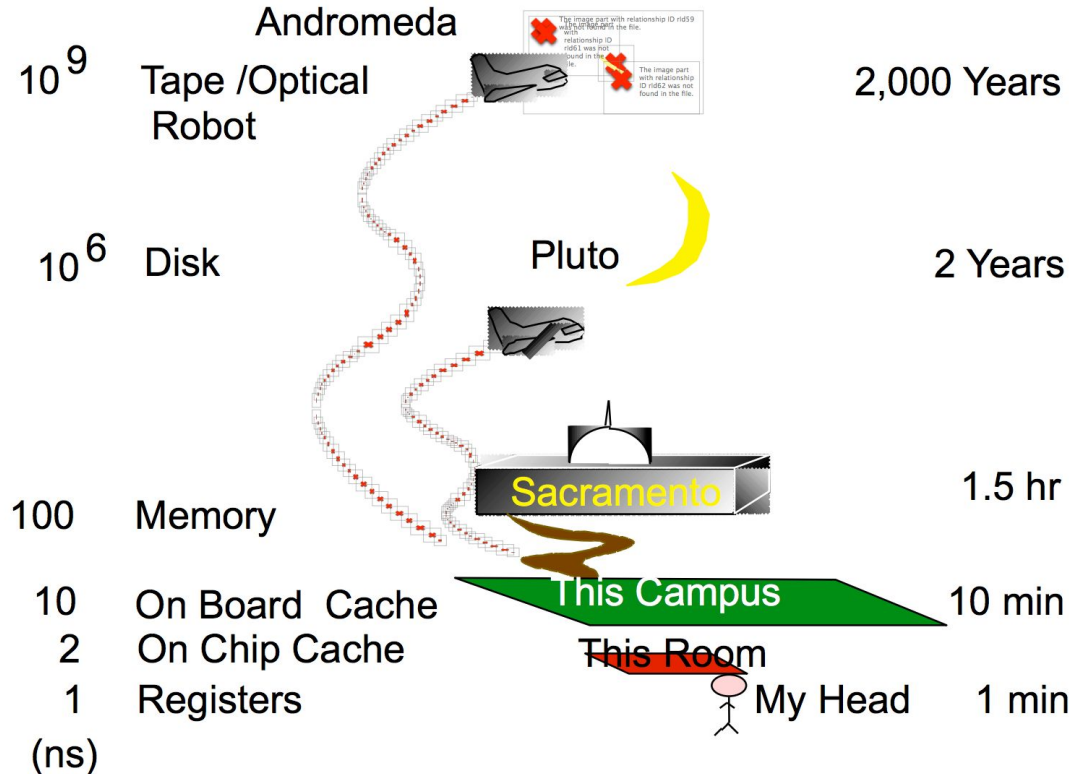


Efficient memory use is
critical to *good performance*

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?

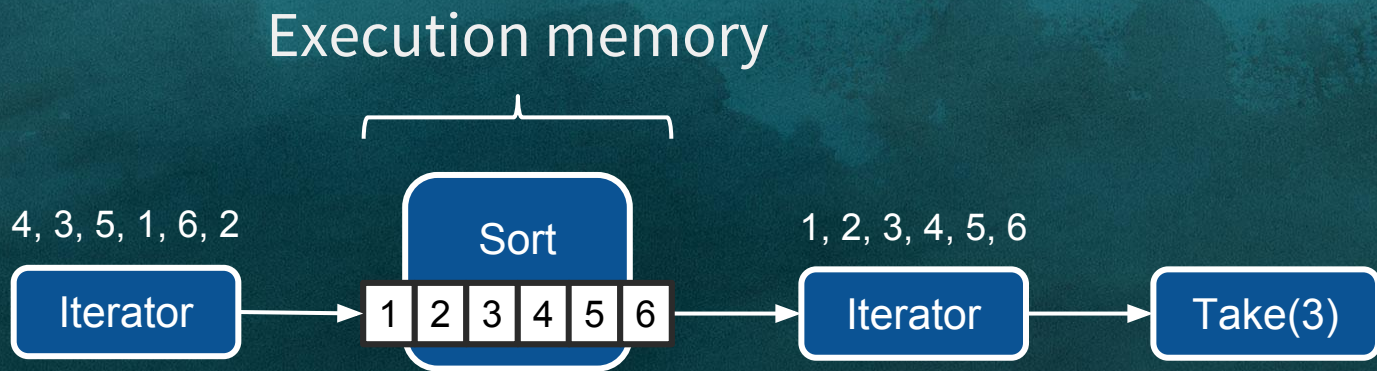
Two usages of memory in Apache Spark

Execution

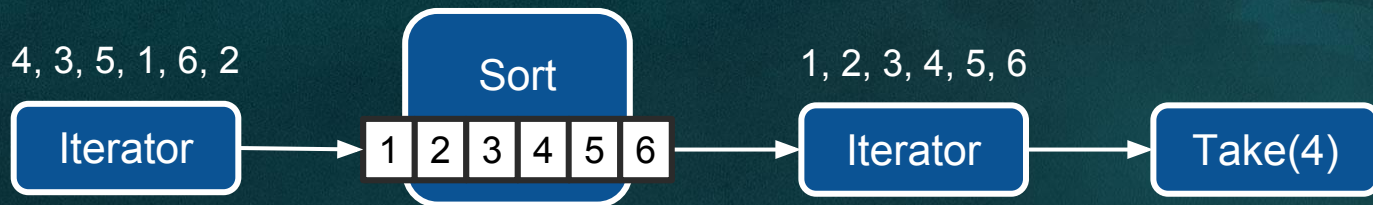
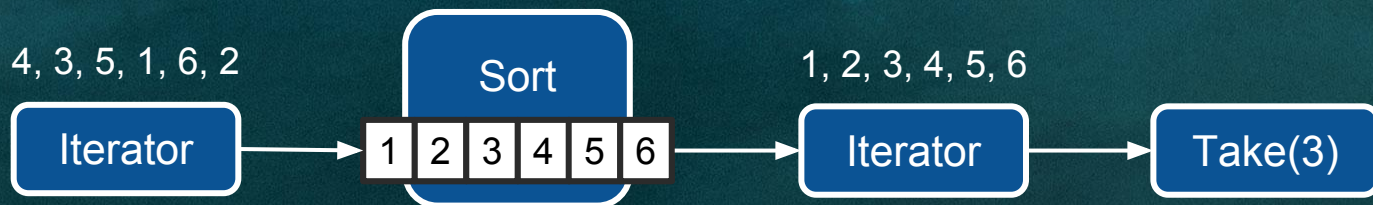
Memory used for shuffles, joins, sorts and aggregations

Storage

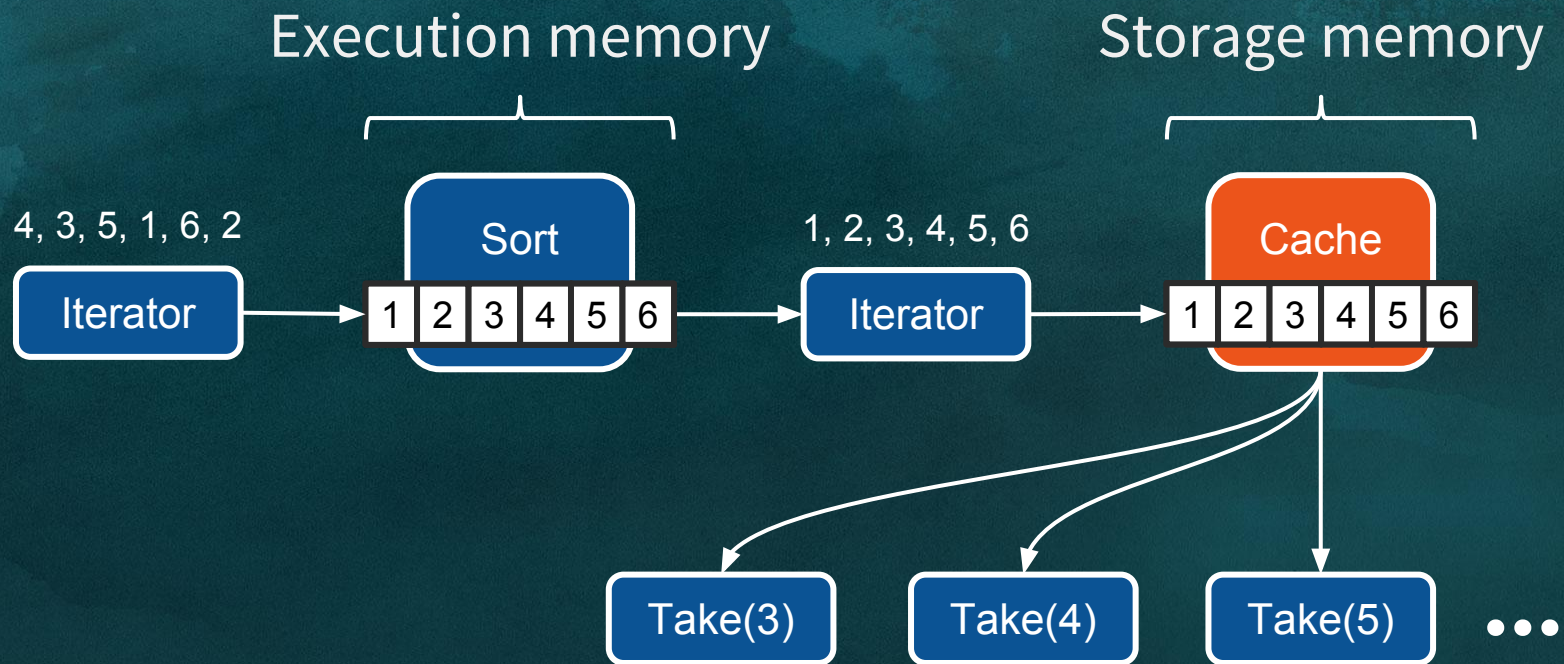
Memory used to cache data that will be reused later



What if I want the sorted values again?



⋮



Challenge #1

How to arbitrate memory between
execution and storage?

Easy, static assignment!



Spark 1.0
May 2014

Easy, static assignment!



Spark 1.0
May 2014

Easy, static assignment!



Spark 1.0
May 2014

Easy, static assignment!

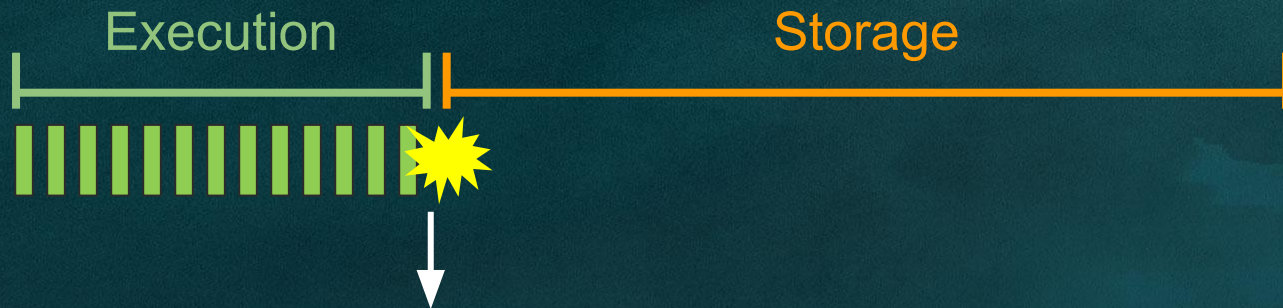


Spark 1.0
May 2014

Inefficient memory use leads to
bad performance

Easy, static assignment!

Spark 1.0
May 2014



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

Easy, static assignment!

Spark 1.0
May 2014



Efficient use of memory required user tuning

Fast forward to 2016...
How could we have done better?



Unified memory management



What happens if there is already storage?

Spark 1.6+
Jan 2016

Unified memory management



Evict LRU block to disk

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 assignment!

Worker machine has 4 cores

Each task gets 1/4 of the total memory



Alternative: Dynamic assignment

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



Alternative: Dynamic assignment

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



Alternative: Dynamic assignment

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



Alternative: Dynamic assignment

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



Alternative: Dynamic assignment

Last remaining task gets all the
memory because $N = 1$



Task 3

Spark 1.0+
May 2014

Static vs dynamic assignment

Both are fair and starvation free

Static assignment is simpler

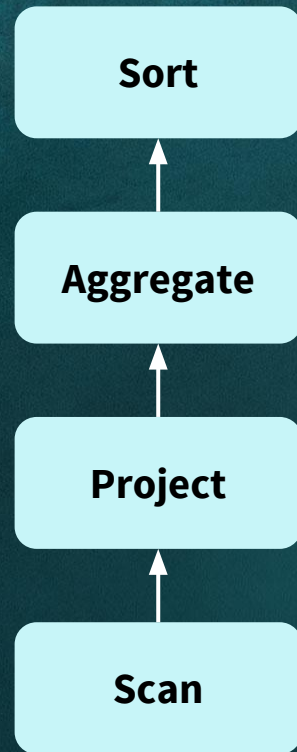
Dynamic assignment handles stragglers better

Challenge #3

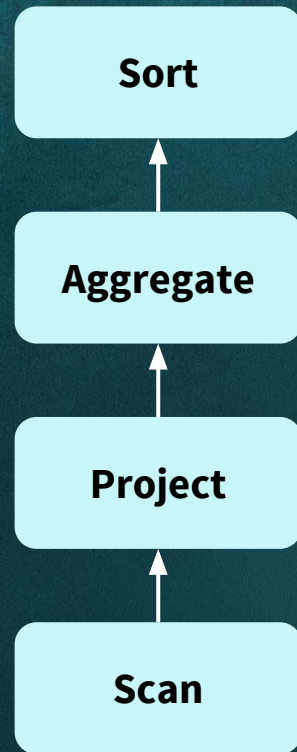
How to arbitrate memory across
operators running within the same task?


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

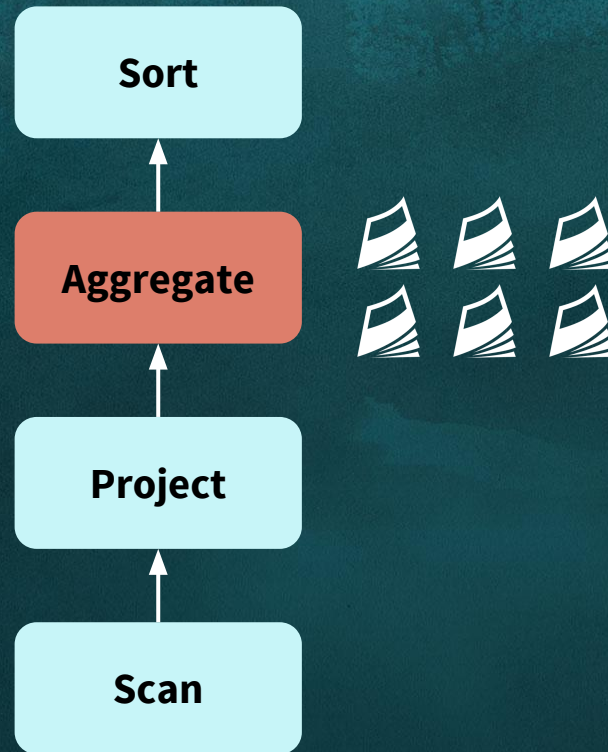
```
students.groupBy("age")
  .avg("height")
  .orderBy("avg(height)")
  .collect()
```



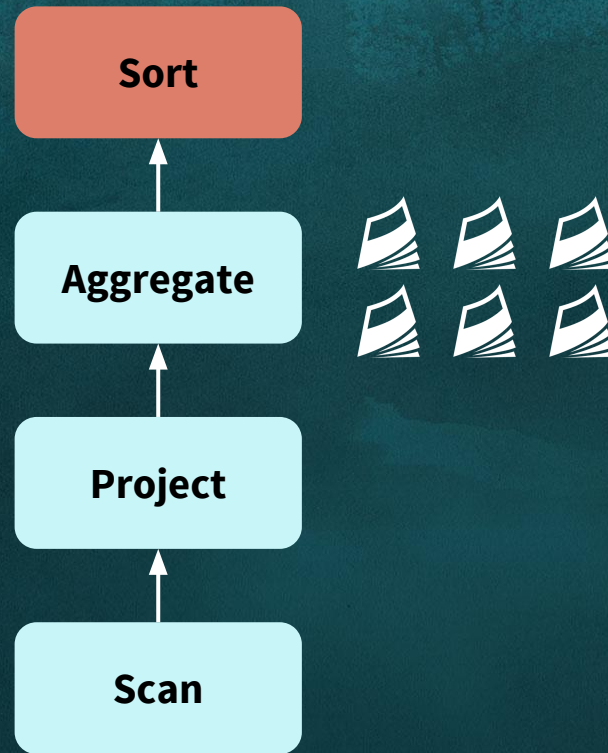
Worker has 6
pages of memory



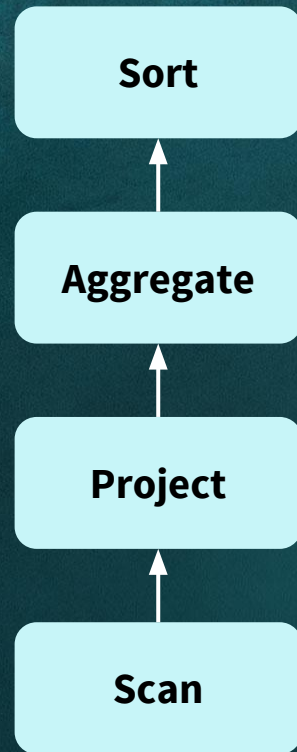

```
Map { // age → heights  
  20 → [154, 174, 175]  
  21 → [167, 168, 181]  
  22 → [155, 166, 188]  
  23 → [160, 168, 178, 183]  
}
```



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



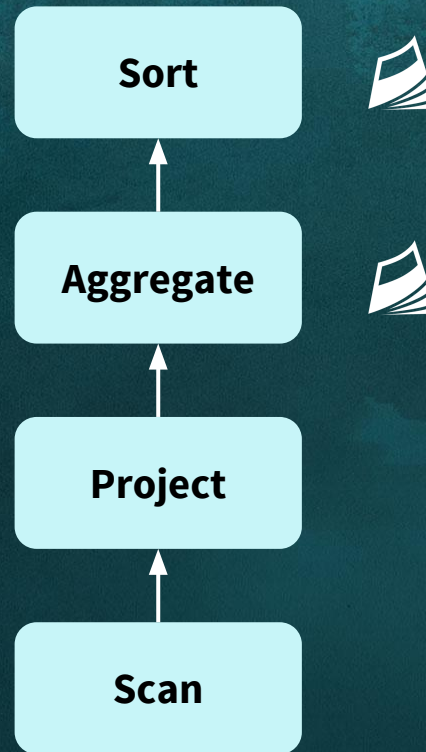
Solution #1:
Reserve a page for
each operator



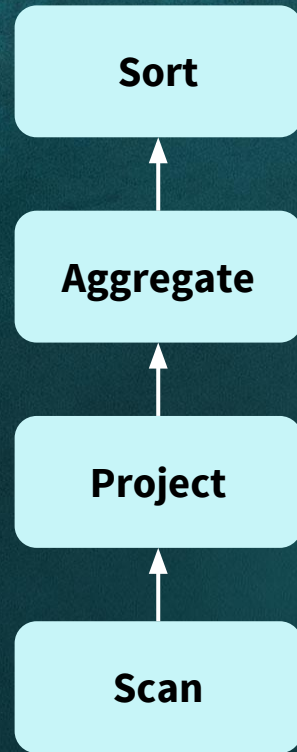
Solution #1:
Reserve a page for
each operator



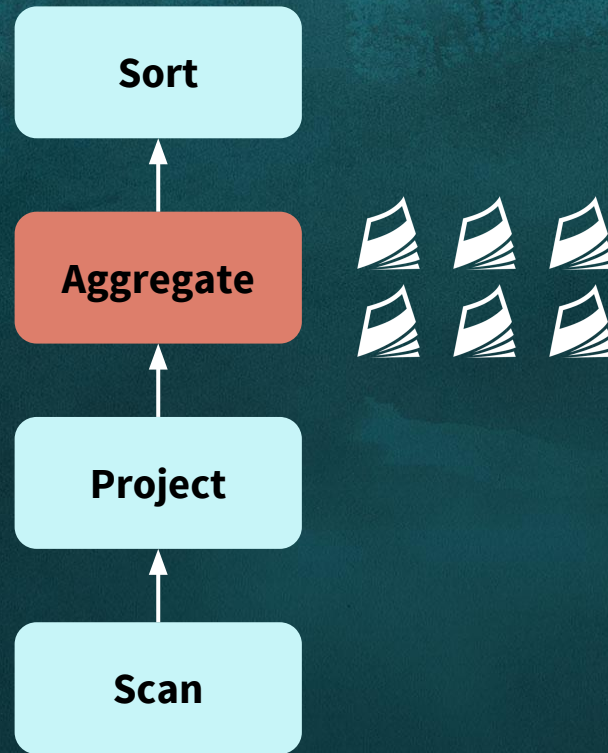
Starvation free, but still not fair...
What if there were more operators?



Solution #2: Cooperative spilling

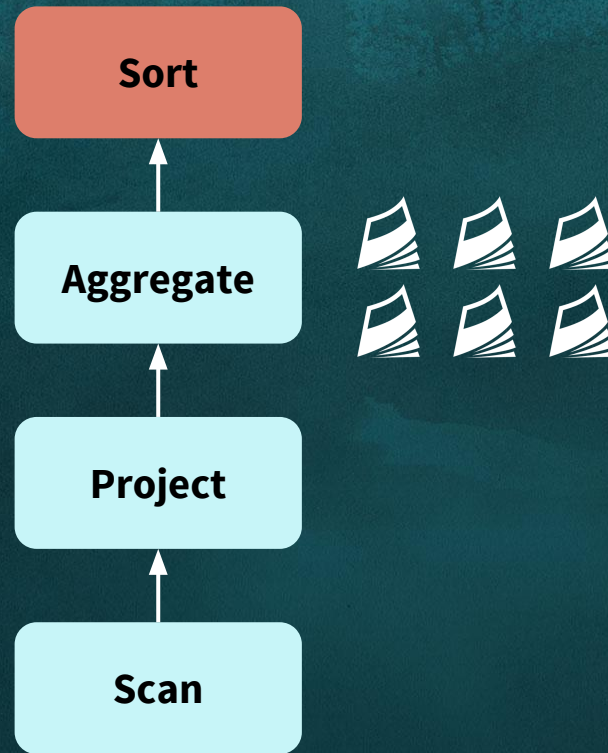


Solution #2:
Cooperative spilling



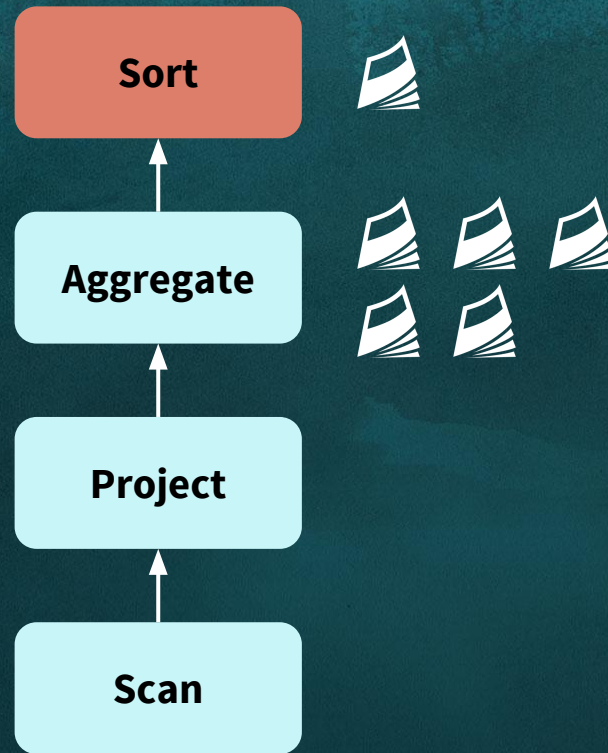
Solution #2:
Cooperative spilling

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



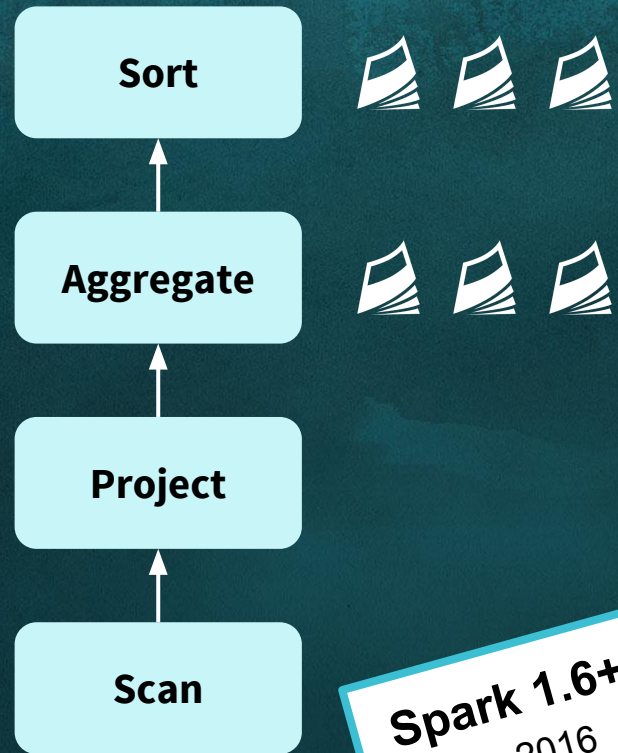
Solution #2: Cooperative spilling

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



Spark 1.6+
Jan 2016

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?

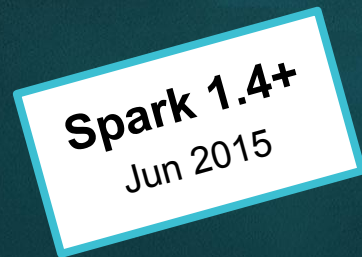
Instead of avoid statically reserving memory in advance, deal with memory contention when it arises by forcing members to spill

Project Tungsten

Binary in-memory data representation

Cache-aware computation

Code generation (next time)



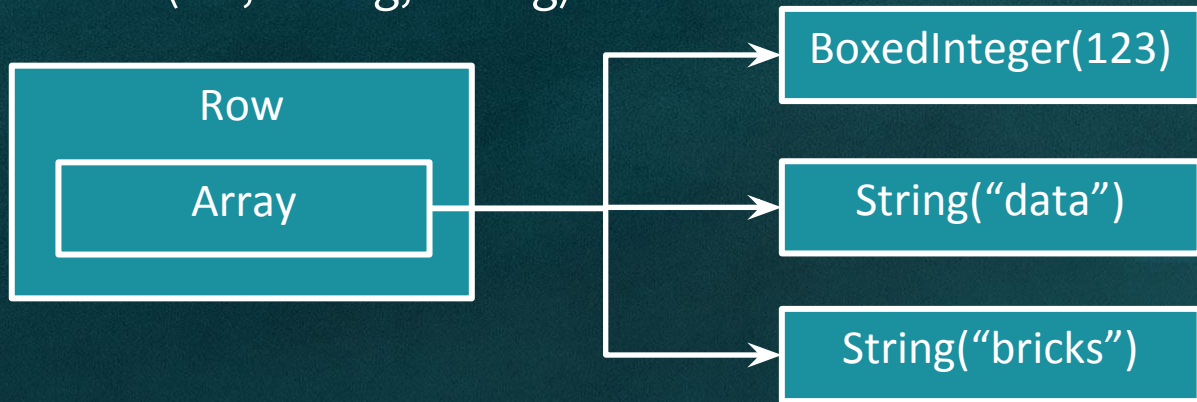
Java objects have large overheads

“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

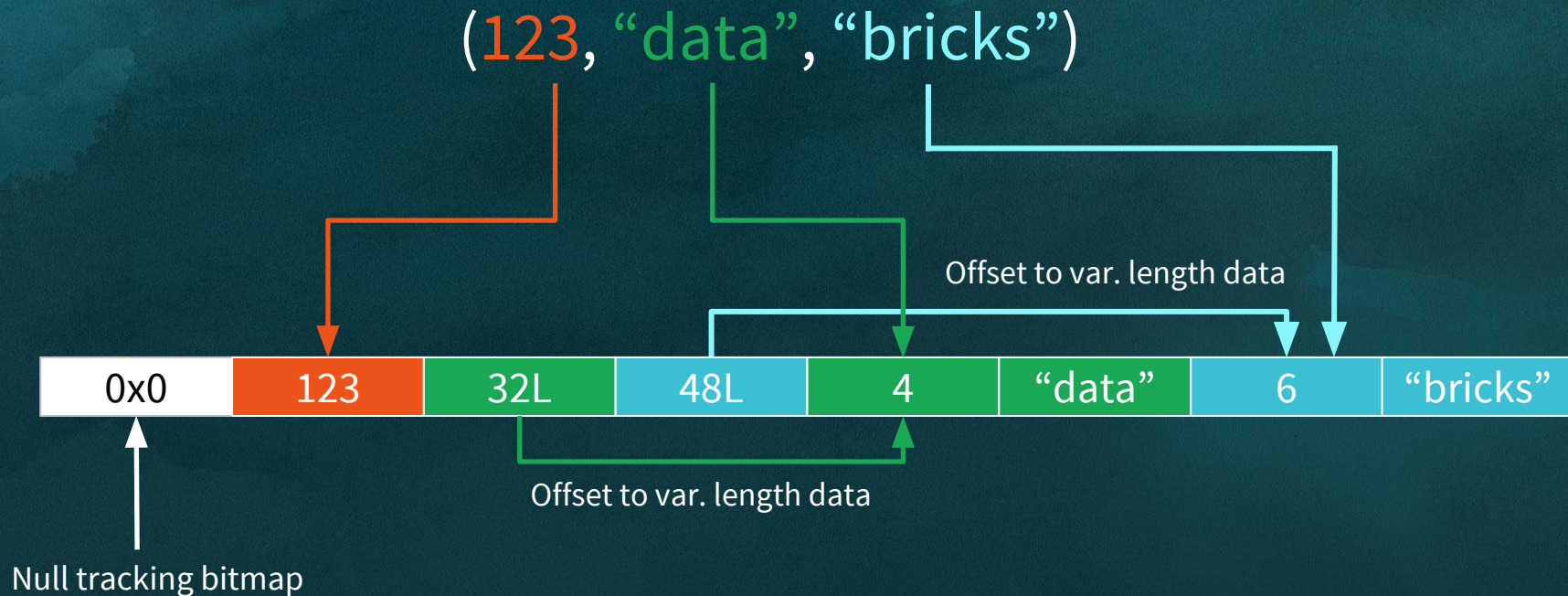
Java objects based row format

Schema: (Int, String, String)



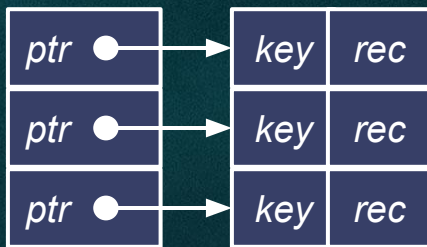
5+ objects, high space overhead, expensive hashCode()

Tungsten row format

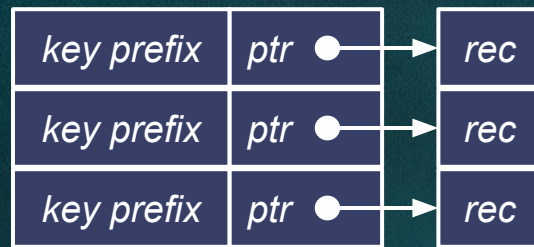


Cache-aware computation

E.g. sorting a list of records



Naive layout
Poor cache locality



Cache-aware layout
Good cache locality

Off-heap memory

Available for execution since Apache Spark 1.6

Available for storage since Apache Spark 2.0

Very important for large heaps

Many potential advantages: memory sharing, zero copy I/O, dynamic allocation

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>



Databricks Community Edition

Free version of cloud based platform in beta

More than 8,000 users registered

Users created over 61,000 notebooks in different languages

<http://www.databricks.com/try>



The background of the slide is a deep blue underwater scene. Two divers are visible in silhouette, swimming towards the right. Bubbles are rising from the divers, and light rays from the surface create a dramatic, sunburst-like effect in the center of the image.

Thank you

andrew@databricks.com

[@andrewor14](https://twitter.com/andrewor14) 