Tungsten

Objective

Understand Project Tungsten principles

Simple performance demos

Tungsten - Motivation

Many big data infrastructures are CPU-bound

- large 10Gbps bandwidth
- fast SSDs: GB/s transfers
- CPUs still ~4GHz

Spark is already quite fast

- column/partition/bucket pruning & making data transfers smaller
- smaller & faster storage formats e.g. Parquet
- new shuffle implementations are much faster than before

Tungsten Efficiency

Memory management

- leverage memory directly: CPU registers, cache, RAM
- bypass the JVM and garbage collection

Cache-aware computation

- leverage modern CPUs and memory hierarchy
- cache-friendly data structures – 8byte-aligned
- equality and hashing performed at the byte level

Code generation

- leverage modern CPUs
    - e.g. generated sort is 3x as fast
- operate directly on binary

Tungsten Details

Off-heap storage

- bypasses the JVM with "unsafe" objects (not eligible for GC)
- cache-aligned data structures: strings, UnsafeRows, ByteToByteMap
- internal "memory paging"
- plain memory pointers

Code generation (WholeStageCodegen)

- avoid virtual function calls
- avoid primitive type boxing
- avoid branches based on object type
- generate custom bytecode with Janino

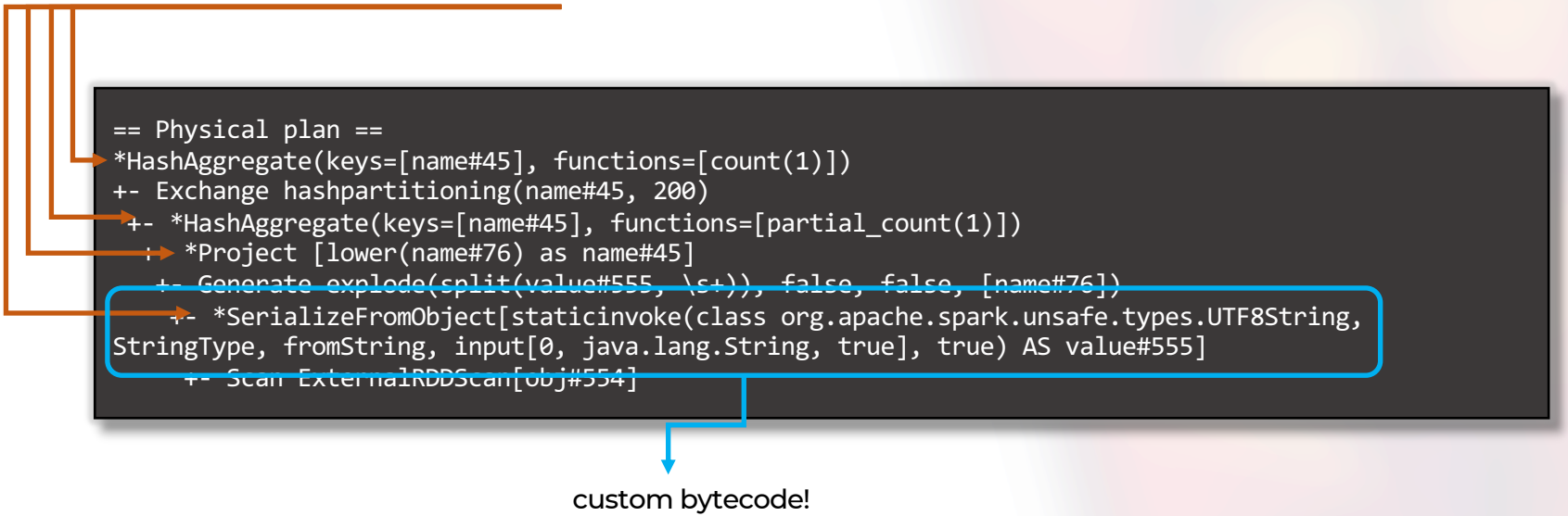
Who benefits from Tungsten

- Spark SQL
- some RDD transformations

Tungsten

How we see it

- in the query plans
- tasks called SerializeFromObject – serialize the data from the beginning
- tasks with asterisk (*) use Tungsten



Tungsten

Benefits

- faster serialization than Kryo and >>>> Java
- much smaller object size than Java
- supports off-heap allocation
- supports Spark operations without serialization
  e.g. you can sort the data while in binary
- avoids the JVM's GC
- much faster, less memory, less CPU
- can process much larger datasets

Tungsten

How do we enable it?

- it's enabled and free!

```
spark.sql.tungsten.enabled = true
```

Spark rocks