Spark After Dark 1.5+

End-to-End, Real-time, Advanced Analytics and ML Big Data Reference Pipeline

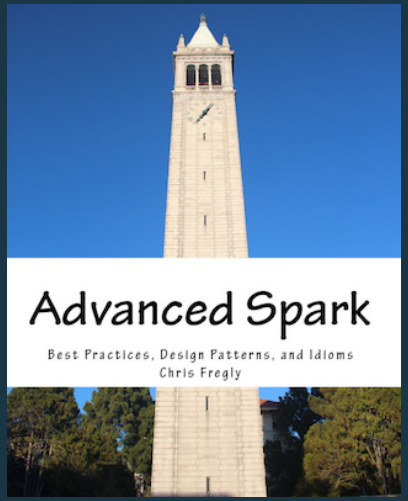
Chris Fregly

Principal Data Solutions Engineer

IBM Spark Technology Center

{spark.tc} advancedspark.com

We're Hiring - Only Nice People!



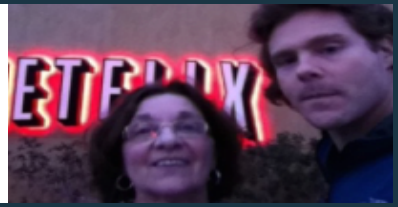
Toronto Apache Spark Meetup

Dec 14th, 2015

Who Am I?

Streaming Data Engineer

Netflix Open Source Committer



Data Solutions Engineer

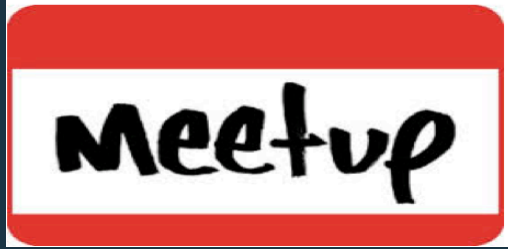
Apache Spark Contributor

Principal Data Solutions Engineer

IBM Spark Technology Center

Meetup Founder

Advanced Apache Spark Meetup



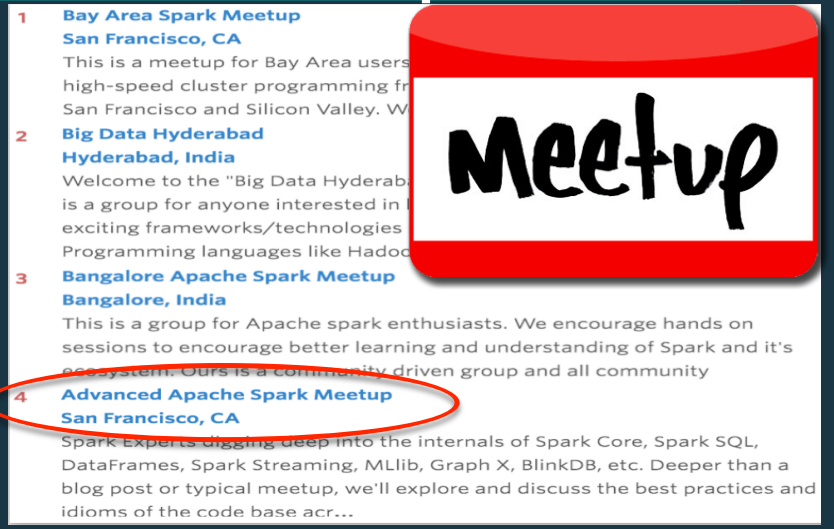
Book Author

AdvancedSpark

Advanced Apache Spark Meetup

Meetup **Metrics**

Top 5 Most Active Spark Meetup!
1800+ Members in just **4 mos**!!
2100+ Downloads of Docker Image with many Meetup Demos!!!
@ **advancedspark.com**



Meetup **Goals**

Code dive deep into Spark and related open source code bases
Study integrations with Cassandra, ElasticSearch,Tachyon, S3, BlinkDB, Mesos, YARN, Kafka, R, etc
Surface and share patterns and idioms of well-designed, distributed, big data processing systems

World Tour: Freg-a-Palooza!

London Spark Meetup (Oct 12th)

Scotland Data Science Meetup (Oct 13th)

Dublin Spark Meetup (Oct 15th)

Barcelon Spark Meetup (Oct 20th)

Madrid Big Data Meetup (Oct 22nd)

Paris Spark Meetup (Oct 26th)

Amsterdam Spark Summit (Oct 27th)

Brussels Spark Meetup (Oct 30th)

Zurich Big Data Meetup (Nov 2nd)

Geneva Spark Meetup (Nov 5th)

Oslo Big Data Hadoop Meetup (Nov 19th)

Helsinki Spark Meetup (Nov 20th)

Stockholm Spark Meetup (Nov 23rd)

Copenhagen Spark Meetup (Nov 25th)

Budapest Spark Meetup (Nov 26th)

Istanbul Spark Meetup (Nov 28th)

Singapore Spark Meetup (Dec 1st)

Sydney Spark Meetup (Dec 8th)

Melbourne Spark Meetup (Dec 9th)

Toronto Spark Meetup (Dec 14th)

LAST MEETUP OF 2015!!

Topics of This Talk (20-30 mins each)

① Spark Streaming and Spark ML

Generating Real-time Recommendations

② Spark Core

Tuning and Profiling

③ Spark SQL

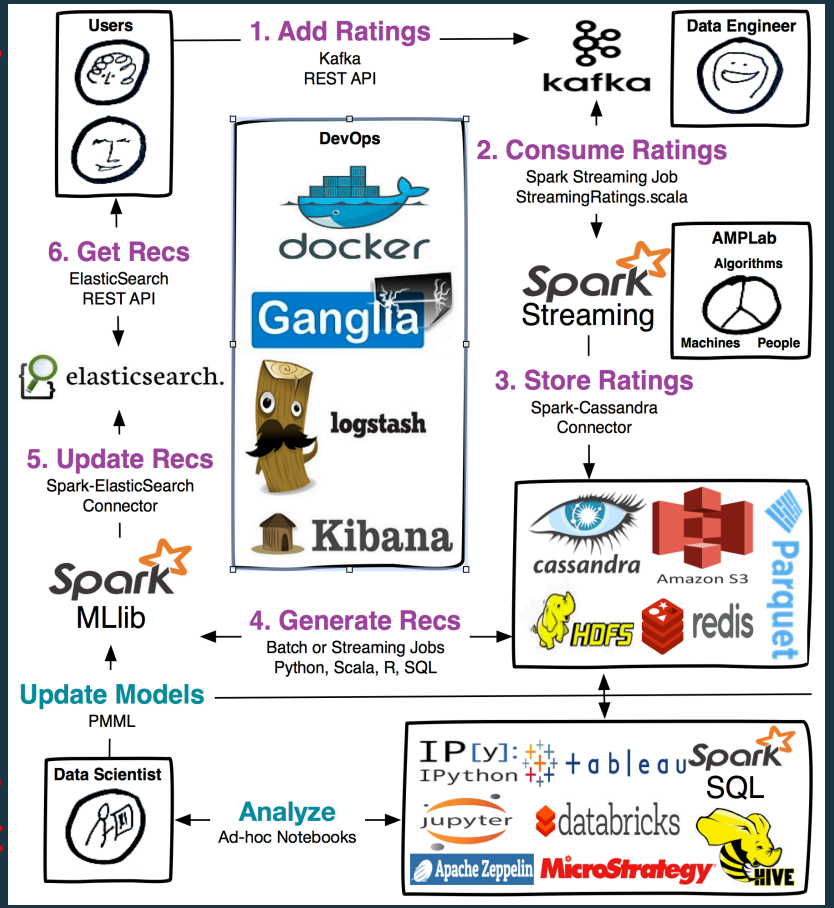
Tuning and Customizing

Live, Interactive Demo!

Kafka, Cassandra, ElasticSearch, Redis, Spark ML

Audience Participation Required!

You ->



Data -> Scientist

Audience Instructions

①Navigate to

sparkafterdark.com

②Swipe software used in

Production Only!

This is Totally Anonymous!!

Topics of This Talk (15-20 mins each)

① Spark Streaming and Spark ML

Kafka, Cassandra, ElasticSearch, Redis, Docker

② Spark Core

Tuning and Profiling

③ Spark SQL

Tuning and Customizing

Mechanical Sympathy

*"**Hardware** and **software** working together."*

- Martin Thompson

http://mechanical-sympathy.blogspot.com

*"Whatever your **data structure**, my **array** wins."*

- Scott Meyers

Every C++ Book, basically

Spark and Mechanical Sympathy

Saturate Network I/O
Saturate Disk I/O

100TB GraySort Challenge
(Spark 1.1-1.2)

Minimize Memory & GC
Maximize CPU Cache

Project Tungsten
(Spark 1.4-1.6+)

CPU Cache Refresher

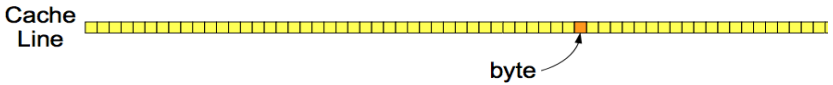
Cache Lines

Caches consist of *lines*, each holding multiple adjacent words.

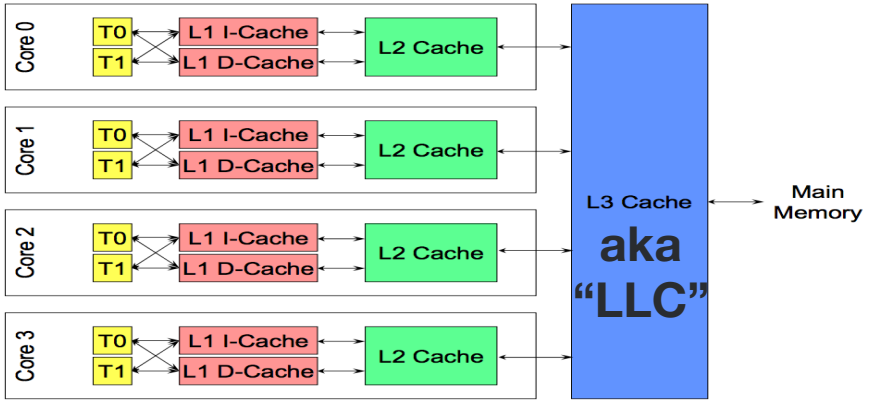
- On Core i7, cache lines hold 64 bytes.
  - 64-byte lines common for Intel/AMD processors.
  - 64 bytes = 16 32-bit values, 8 64-bit values, etc.
    - E.g., 16 32-bit array elements.

Main memory read/written in terms of cache lines.

- Read byte not in cache ⇒ read full cache line from main memory.
- Write byte ⇒ write full cache line to main memory (eventually).



Core i7-9xx Cache Hierarchy



My Laptop

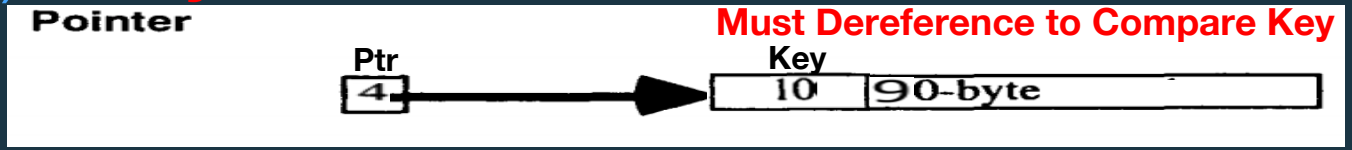
Model Name:	MacBook Pro
Model Identifier:	MacBookPro11,5
Processor Name:	Intel Core i7
Processor Speed:	2.8 GHz
Total Number of Cores:	4
L1 Cache (per Core):	32KB
L2 Cache (per Core):	256 KB
L3 Cache:	6 MB
Memory:	16 GB

My SoftLayer BareMetal

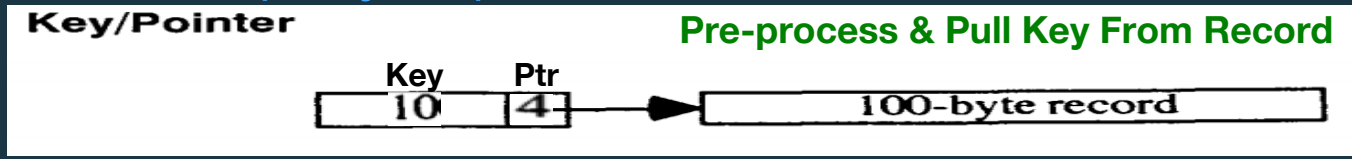
Vendor ID:	GenuineIntel
CPU family:	6
Model:	62
Stepping:	4
CPU MHz:	3000.055
BogoMIPS:	6000.11
Virtualization:	VT-x
L1d cache:	32K
L2 cache:	256K
L3 cache:	25MB

CPU Cache Sympathy (AlphaSort paper)

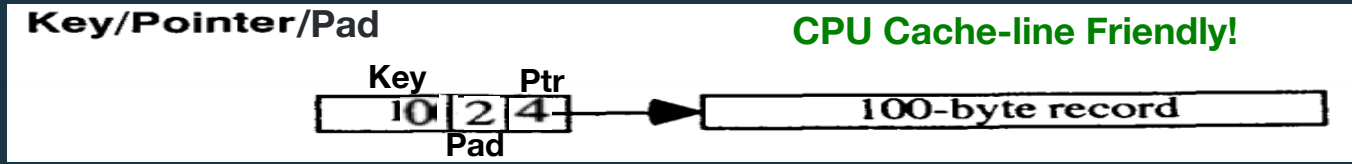
Pointer (4 bytes) = **4 bytes**



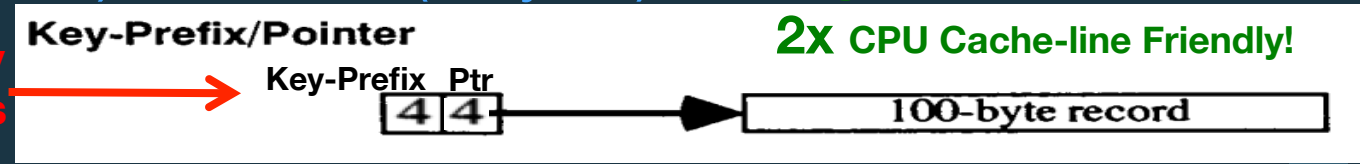
Key (10 bytes) + Pointer (4 bytes) = **14 bytes**



Key (10 bytes) + Pad (2 bytes) + Pointer (4 bytes) = **16 bytes**



Key-Prefix (4 bytes) + Pointer (4 bytes) = **8 bytes**



Sort Performance Comparison

Table 2. CPU times (seconds) for four types of QuickSorts sorting one million records

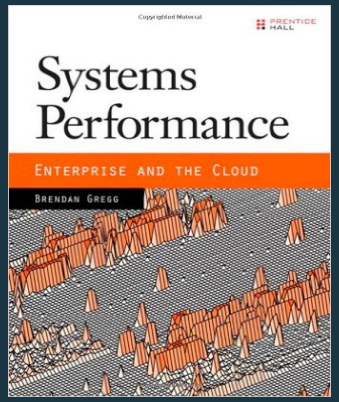
	Generate Pointer Array	QuickSort	Output	Total
Pointer	0.08	12.74	3.52	16.34
Key/Pointer	1.07	4.02	3.41	8.50
Key-Prefix/Pointer	0.84	3.32	3.41	7.57

Sequential vs Random Cache Misses

Set Size	Sequential					Random				
	L2 Hit	L2 Miss	#Iter	Ratio Miss/Hit	L2 Accesses Per Iter	L2 Hit	L2 Miss	#Iter	Ratio Miss/Hit	L2 Accesses Per Iter
2^{20}	88,636	843	16,384	0.94%	5.5	30,462	4721	1,024	13.42%	34.4
2^{21}	88,105	1,584	8,192	1.77%	10.9	21,817	15,151	512	40.98%	72.2
2^{22}	88,106	1,600	4,096	1.78%	21.9	22,258	22,285	256	50.03%	174.0
2^{23}	88,104	1,614	2,048	1.80%	43.8	27,521	26,274	128	48.84%	420.3
2^{24}	88,114	1,655	1,024	1.84%	87.7	33,166	29,115	64	46.75%	973.1
2^{25}	88,112	1,730	512	1.93%	175.5	39,858	32,360	32	44.81%	2,256.8
2^{26}	88,112	1,906	256	2.12%	351.6	48,539	38,151	16	44.01%	5,418.1
2^{27}	88,114	2,244	128	2.48%	705.9	62,423	52,049	8	45.47%	14,309.0
2^{28}	88,120	2,939	64	3.23%	1,422.8	81,906	87,167	4	51.56%	42,268.3
2^{29}	88,137	4,318	32	4.67%	2,889.2	119,079	163,398	2	57.84%	141,238.5

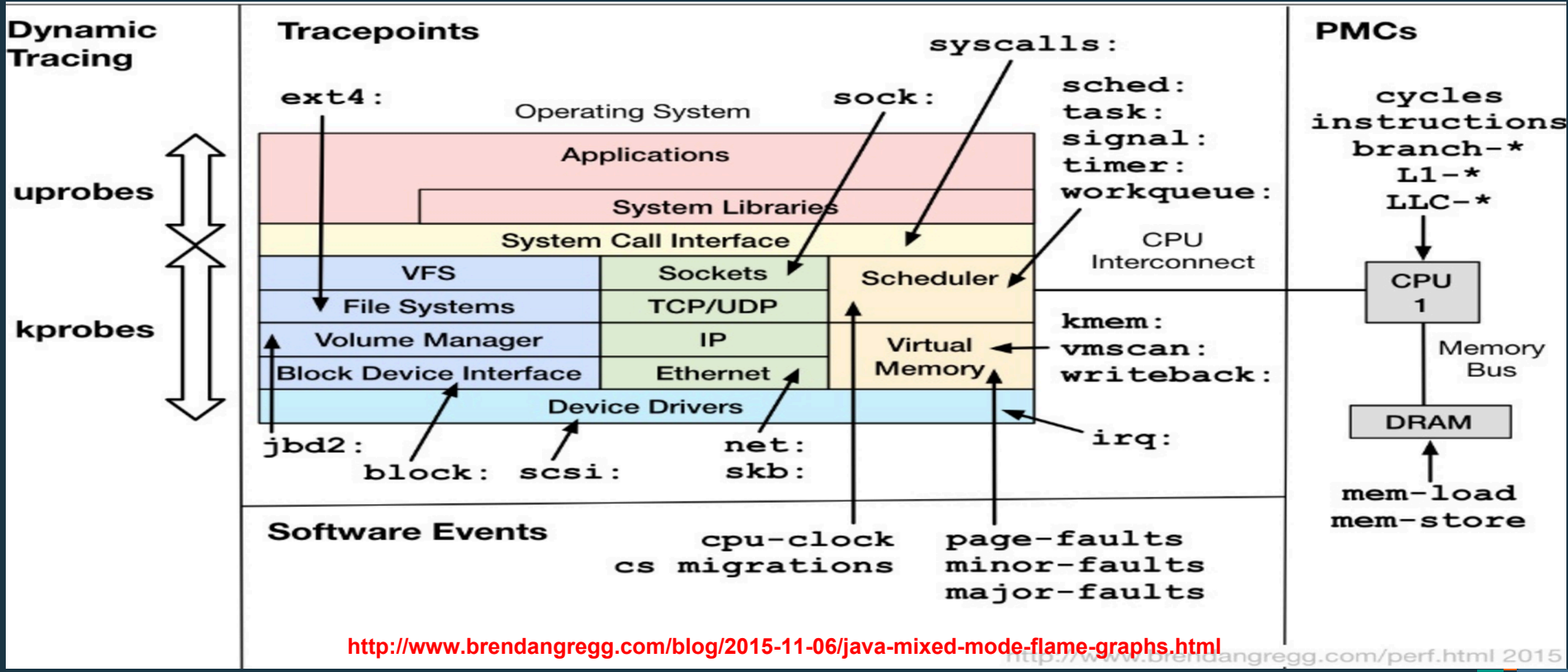
Demo!

Sorting



Instrumenting and Monitoring CPU

Use Linux **perf** command!



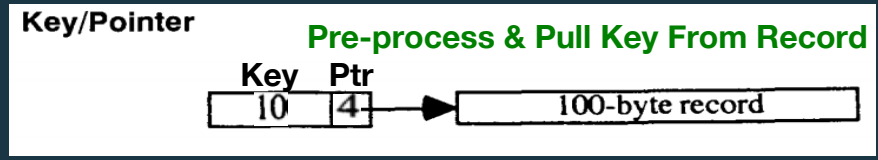
http://www.brendangregg.com/blog/2015-11-06/java-mixed-mode-flame-graphs.html

http://www.brendangregg.com/perf.html 2015

Results of Random vs. Sequential Sort

perf stat –event \
L1-dcache-load-misses,L1-dcache-prefetch-misses,LLC-load-misses,LLC-prefetch-misses

Naïve Random Access Pointer Sort	% Change	Cache Friendly Sequential Key/Pointer Sort
3638556550 L1-dcache-load-misses	-35%	2371344253 L1-dcache-load-misses
2265987264 L1-dcache-prefetch-misses	-55%	1009741567 L1-dcache-prefetch-misses
1710655484 LLC-load-misses	-90%	161927675 LLC-load-misses
2391446019 LLC-prefetch-misses	-68%	760163894 LLC-prefetch-misses
51.700065001 seconds time elapsed	-26%	38.019590096 seconds time elapsed



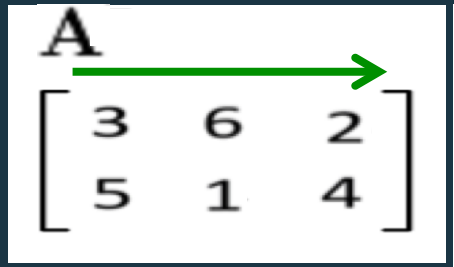
Demo!

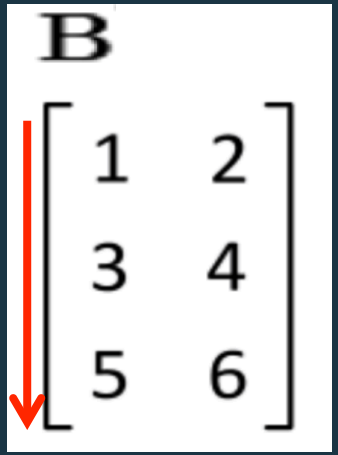
Matrix Multiplication

CPU Cache Naïve Matrix Multiplication

$$(AB)_{ij} = \sum_{k=0}^{N-1} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{i(N-1)} b_{(N-1)j}$$

```
// Dot product of each row & column vector
for (i <- 0 until numRowA)
  for (j <- 0 until numColsB)
    for (k <- 0 until numColsA)
      res[ i ][ j ] += matA[ i ][ k ] * matB[ k ][ j ];
```



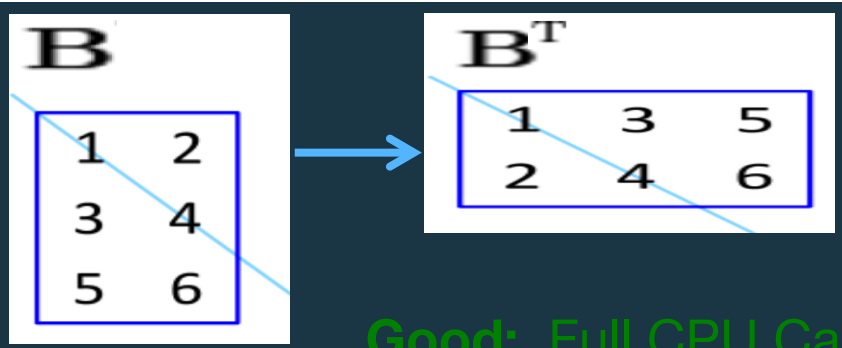


Bad: Row-wise traversal,
not using full CPU cache line,
ineffective pre-fetching

CPU Cache Friendly Matrix Multiplication

$$(AB)_{ij} = \sum_{k=0}^{N-1} a_{ik} b^{\mathrm{T}}_{jk} = a_{i1} b^{\mathrm{T}}_{j1} + a_{i2} b^{\mathrm{T}}_{j2} + \cdots + a_{i(N-1)} b^{\mathrm{T}}_{j(N-1)}$$

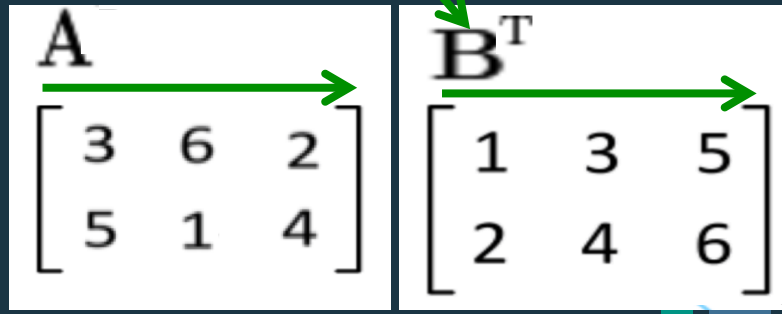
```
// Transpose B
for (i <- 0 until numRowsB)
  for (j <- 0 until numColsB)
    matBT[ i ][ j ] = matB[ j ][ i ];
```




```
// Modify algo for Transpose B
for (i <- 0 until numRowsA)
  for (j <- 0 until numColsB)
    for (k <- 0 until numColsA)
      res[ i ][ j ] += matA[ i ][ k ] * matBT[ j ][ k ];
```

OLD: matB [**k**][**j**];

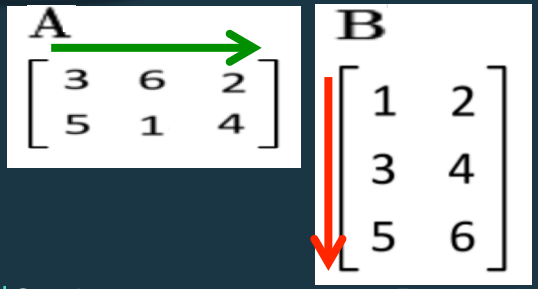
Good: Full CPU Cache Line, Effective Prefetching

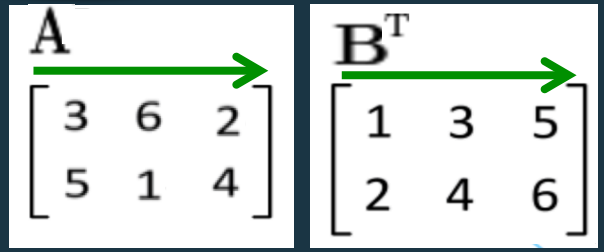


Results Of Matrix Multiplication

perf stat –event \
L1-dcache-load-misses,L1-dcache-prefetch-misses,LLC-load-misses, \
LLC-prefetch-misses,cache-misses,stalled-cycles-frontend

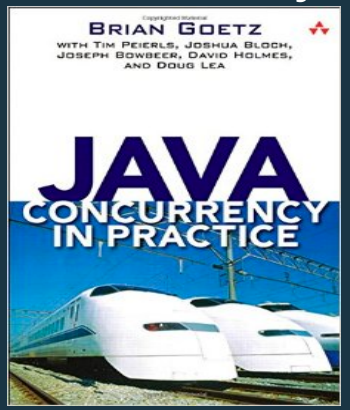
Naïve Matrix Multiply	% Change	Cache-Friendly Matrix Multiply
214482863052 L1-dcache-load-misses	-96%	7916379222 L1-dcache-load-misses
11487838928 L1-dcache-prefetch-misses	-63%	4248568884 L1-dcache-prefetch-misses
4492612296 LLC-load-misses	-93%	336612743 LLC-load-misses
49758059 LLC-prefetch-misses	+8543%?	4300544980 LLC-prefetch-misses
4447068200 cache-misses	-93%	320086472 cache-misses
5246311477291 stalled-cycles-frontend	-70%	1575227969401 stalled-cycles-frontend
2298.398022413 seconds time elapsed	-53%	1073.199705670 seconds time elapsed

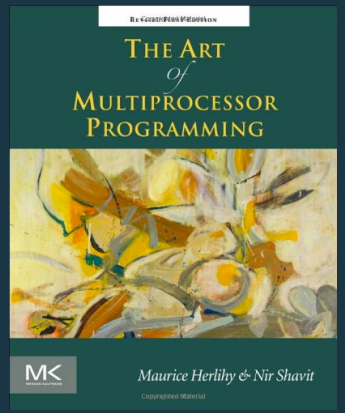




Demo!

Thread Synchronization





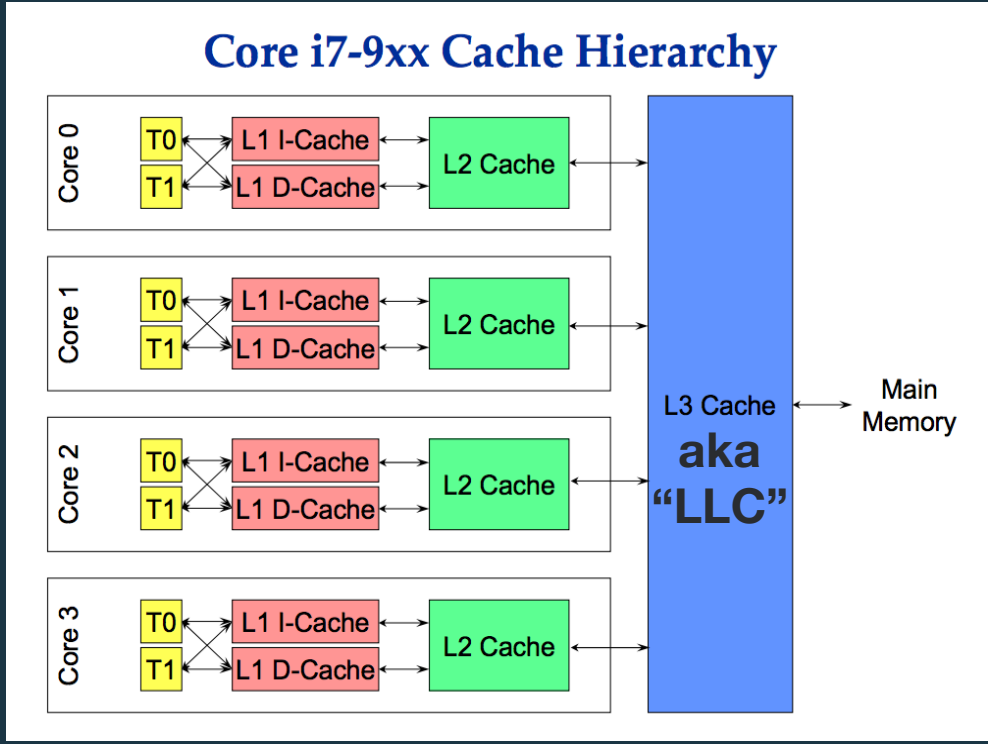
Thread and Context Switch Sympathy

Problem

Atomically Increment 2 Counters (each at different increments) by 1000's of Simultaneous Threads

Possible Solutions

① Synchronized Immutable
② Synchronized Mutable
③ AtomicReference CAS
④ Volatile?



Context Switches are Expen$ive!!

Synchronized Immutable Counters

```
case class Counters(left: Int, right: Int)

object SynchronizedImmutableCounters {
  var counters = new Counters(0,0)
  def getCounters(): Counters = {
    this.synchronized { counters }
  }
  def increment(leftIncrement: Int, rightIncrement: Int): Unit = {
    this.synchronized {
      counters = new Counters(counters.left + leftIncrement,
                              counters.right + rightIncrement)
    }
  }
}
```

Locks whole outer object!!

Synchronized Mutable Counters

```
class MutableCounters(left: Int, right: Int) {
  def increment(leftIncrement: Int, rightIncrement: Int): Unit={
    this.synchronized {…}
  }
  def getCountersTuple(): (Int, Int) = {
    this.synchronized{ (counters.left, counters.right) }
  }
}
object SynchronizedMutableCounters {
  val counters = new MutableCounters(0,0)
  …
  def increment(leftIncrement: Int, rightIncrement: Int): Unit = {
    counters.increment(leftIncrement, rightIncrement)
  }
}
```

Locks just MutableCounters

Lock-Free AtomicReference Counters

```
object LockFreeAtomicReferenceCounters {
  val counters = new AtomicReference[Counters](new Counters(0,0))
    ...
  def increment(leftIncrement: Int, rightIncrement: Int) : Long = {
     var originalCounters: Counters = null
     var updatedCounters: Counters = null
     do {
        originalCounters = getCounters()
        updatedCounters = new Counters(originalCounters.left+ leftIncrement,
                                       originalCounters.right+ rightIncrement)
      }
      // Retry lock-free, optimistic compareAndSet() until AtomicRef updates
      while !(counters.compareAndSet(originalCounters, updatedCounters))
  }
}
```

Lock Free!!

Lock-Free AtomicLong Counters

```
object LockFreeAtomicLongCounters {
  // a single Long (64-bit) will maintain 2 separate Ints (32-bits each)
  val counters = new AtomicLong()
  …
  def increment(leftIncrement: Int, rightIncrement: Int): Unit = {
    var originalCounters = 0L
    var updatedCounters = 0L
    do {
      originalCounters = counters.get()
      …
      // Store two 32-bit Int into one 64-bit Long
      // Use >>> 32 and << 32 to set and retrieve each Int from the Long
    }
    // Retry lock-free, optimistic compareAndSet() until AtomicLong updates
    while !(counters.compareAndSet(originalCounters, updatedCounters))
  }
```

Q: Why not @volatile long?

A: The JVM does not guarantee atomic updates of 64-bit longs and doubles

Lock Free!!

Results of Thread Synchronization

perf stat –event \
context-switches,L1-dcache-load-misses,L1-dcache-prefetch-misses, \
LLC-load-misses, LLC-prefetch-misses,cache-misses,stalled-cycles-frontend

Immutable Case Class

```
     1312345 context-switches
  4976648059 L1-dcache-load-misses
  1027608159 L1-dcache-prefetch-misses
   243189136 LLC-load-misses
   147531078 LLC-prefetch-misses
   194167875 cache-misses
532447178108 stalled-cycles-frontend

       12675 millis
```

% Change

-17%
-31%
-32%
-64%
-33%
-46%
-27%

-33%

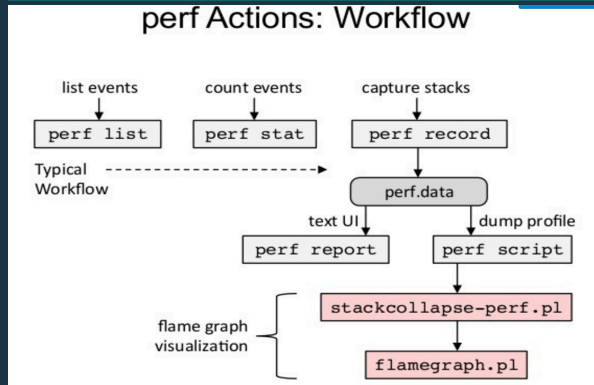
Lock-Free AtomicLong

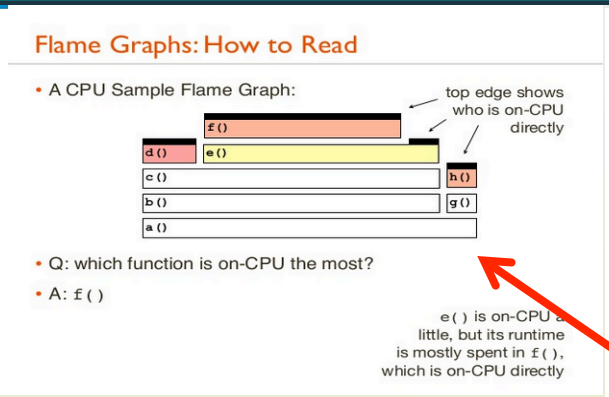
```
     1089963 context-switches
  3423011573 L1-dcache-load-misses
   698607704 L1-dcache-prefetch-misses
    86616269 LLC-load-misses
    99165991 LLC-prefetch-misses
   105355446 cache-misses
390206794124 stalled-cycles-frontend

        8489 millis
```

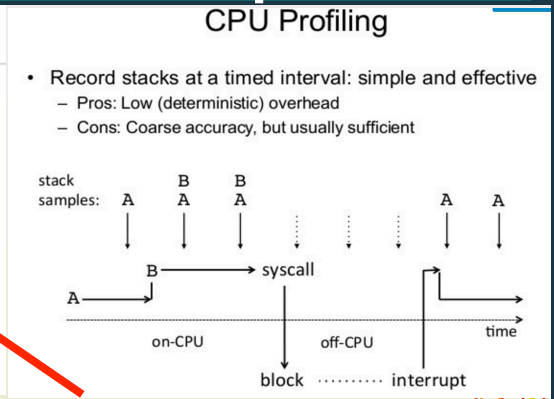
```
case class Counters(left: Int, right: Int)
...
this.synchronized {
 counters = new Counters(counters.left + leftIncrement,
                         counters.right + rightIncrement)
}
```

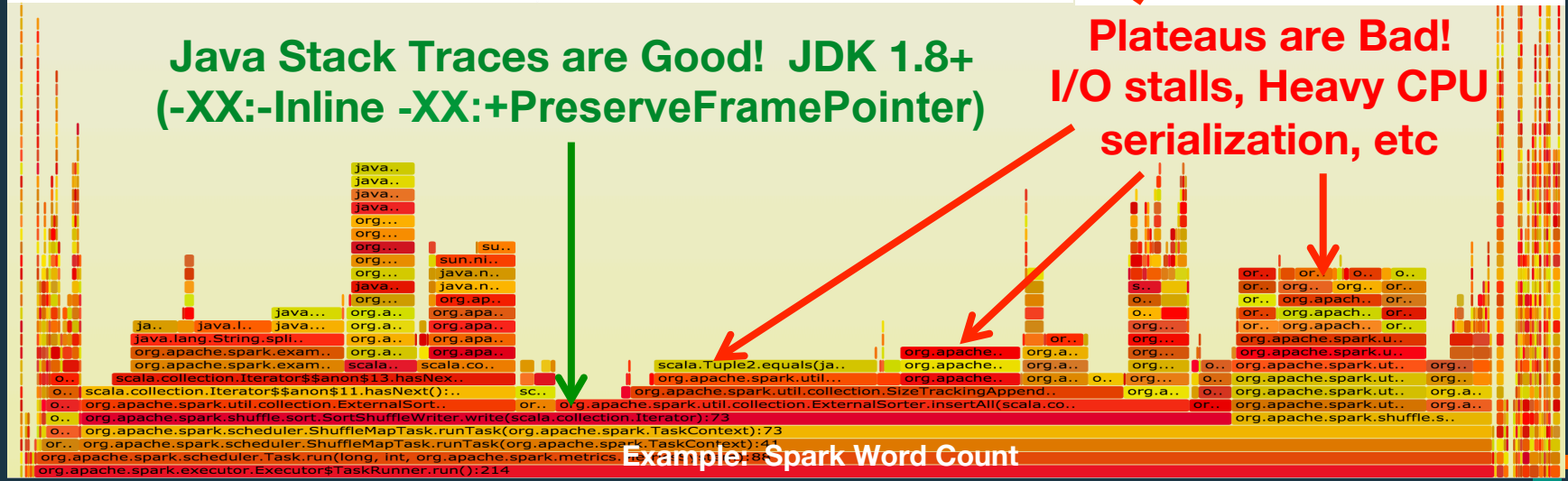
```
val counters = new AtomicLong()
…
do {
…
} while !(counters.compareAndSet(originalCounters,
          updatedCounters))
```

Profile Visualizations: Flame Graphs









Project Tungsten: CPU and Memory

Create Custom Data Structures & Algorithms

- Operate on serialized and compressed ByteArrays!

Minimize Garbage Collection

- Reuse ByteArrays
- In-place updates for aggregations

Maximize CPU Cache Effectiveness

- 8-byte alignment
- AlphaSort-based Key-Prefix

Utilize Catalyst Dynamic Code Generation

- Dynamic optimizations using entire query plan
- Developer implements genCode() to create Scala source code (String)
- Project Janino compiles source code into JVM bytecode

Why is CPU the Bottleneck?

CPU for serialization, hashing, & compression

Spark 1.2 updates saturated Network, Disk I/O

10x increase in I/O throughput relative to CPU

More partition, pruning, and pushdown support

Newer columnar file formats help reduce I/O

Custom Data Structs & Algos: Aggs

UnsafeFixedWidthAggregationMap

- Uses BytesToBytesMap internally
- In-place updates of serialized aggregation
- No object creation on hot-path

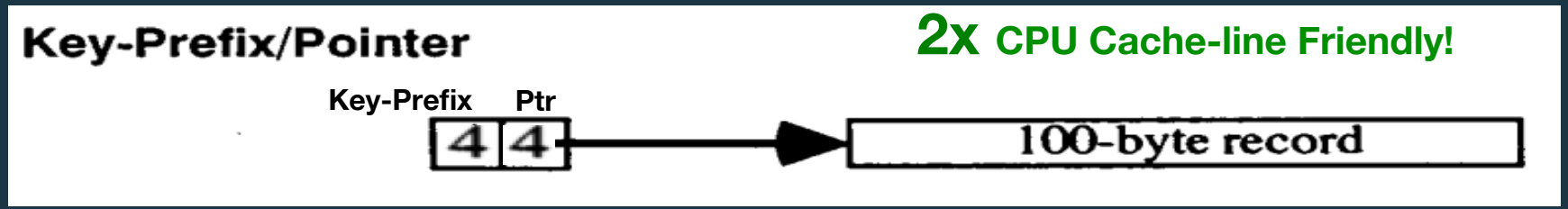
TungstenAggregate & TungstenAggregationIterator

- Operates directly on serialized, binary UnsafeRow
- 2 steps to avoid single-key OOMs
  1. Hash-based (grouping) agg spills to disk if needed
  2. Sort-based agg performs external merge sort on spills

Custom Data Structures & Algorithms

o.a.s.util.collection.unsafe.sort.

- UnsafeSortDataFormat ← SortDataFormat<RecordPointerAndKeyPrefix, Long[ ]>
  Note: Mixing multiple subclasses of SortDataFormat simultaneously will prevent JIT inlining.
- UnsafeExternalSorter ← In-place external sorting of spilled BytesToBytes data
- UnsafeShuffleWriter ← Supports merging compressed records (if compression CODEC supports it, ie. LZF)
- UnsafeInMemorySorter ← In-place sorting of BytesToBytesMap data
- RecordPointerAndKeyPrefix ← AlphaSort-based, 8-byte aligned sort key



Code Generation

Problem

Boxing creates excessive objects

Expression tree evaluations are costly

JVM can't inline polymorphic impls

Lack of polymorphism == poor code design

Solution

Code generation enables inlining

Rewrite and optimize code using overall plan, 8-byte align

Defer source code generation to each operator, UDF, UDAF

Use Janino to compile generated source code into bytecode

(More IDE friendly than Scala quasiquotes)

Autoscaling Spark Workers (Spark 1.5+)

Scaling up is easy ☺

- SparkContext.addExecutors() until max is reached

Scaling down is hard ☹

- SparkContext.removeExecutors()
- Lose RDD cache inside Executor JVM
- Must rebuild active RDD partitions in another Executor JVM

Uses External Shuffle Service from Spark 1.1-1.2

- If Executor JVM dies/restarts, shuffle keeps shufflin'!

“Hidden” Spark Submit REST API

http://arturmkrtchyan.com/apache-spark-hidden-rest-api

(the snitch)
Artur Mkrtchyan

Submit Spark Job

```
curl -X POST http://127.0.0.1:6066/v1/submissions/create \
    --header "Content-Type:application/json;charset=UTF-8" \
    --data ’{"action" : "CreateSubmissionRequest”,
      "mainClass" : "org.apache.spark.examples.SparkPi”,
       "sparkProperties" : {
          "spark.jars" : "file:/spark/lib/spark-examples-1.5.1.jar",
         "spark.app.name" : "SparkPi",…
    }}’
```

Get Spark Job Status

```
curl http://127.0.0.1:6066/v1/submissions/status/<job-id-from-submit-request>
```

Kill Spark Job

```
curl -X POST http://127.0.0.1:6066/v1/submissions/kill/<job-id-from-submit-request>
```

Outline

① Spark Streaming and Spark ML

Kafka, Cassandra, ElasticSearch, Redis, Docker

② Spark Core

Tuning and Profiling

③ Spark SQL

Tuning and Customizing

Parquet Columnar File Format

Based on Google Dremel paper ~2010

Collaboration with Twitter and Cloudera

Columnar storage format for fast columnar aggs

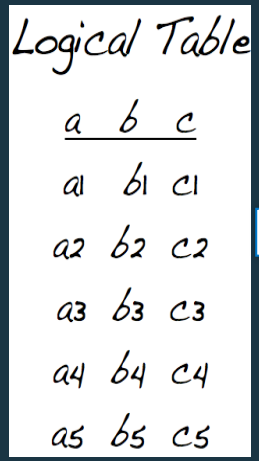
Supports evolving schema

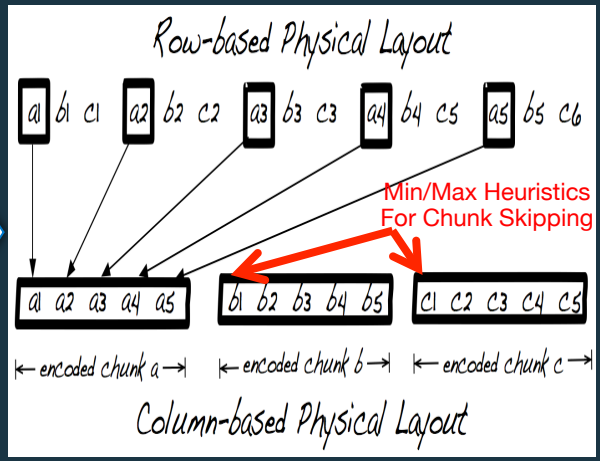
Supports pushdowns

Support nested partitions

Tight compression

Min/max heuristics enable file and chunk skipping





Partitions

Partition Based on Data Access Patterns

/genders.parquet/gender=M/…
/gender=F/… <-- Use Case: Access Users by Gender
/gender=U/…

Dynamic Partition Creation (Write)

Dynamically create partitions on write based on column (ie. Gender)

SQL: INSERT TABLE genders PARTITION (gender) SELECT …

DF: gendersDF.write.format("parquet").partitionBy("gender")
.save("/genders.parquet")

Partition Discovery (Read)

Dynamically infer partitions on read based on paths (ie. /gender=F/…)

SQL: SELECT id FROM genders WHERE gender=F

DF: gendersDF.read.format("parquet").load("/genders.parquet/").select($"id").
.where("gender=F")

Pruning

Partition Pruning

Filter out rows by partition

SELECT id, gender FROM genders WHERE gender = ‘F’

Column Pruning

Filter out columns by column filter

Extremely useful for columnar storage formats (ie. Parquet)

Skip entire blocks of columns

SELECT ~~id~~, gender FROM genders

Pushdowns

aka. Predicate or Filter Pushdowns

Predicate returns true or false for given function

Filters rows deep into the data source

Reduces number of rows returned

Data Source must implement PrunedFilteredScan

```
def buildScan(requiredColumns: Array[String],
              filters: Array[Filter]): RDD[Row]
```

Demo!

File Formats, Partitions, Pushdowns, and Joins

Predicate Pushdowns & Filter Collapsing

Ratings Partitioned Parquet, Genders Unpartitioned JSON Join

FINISHED

```
val df = ratingsPartitionedParquetDF.select($"toUserId", $"rating").where($"rating" <= 6).filter($"rating" >= 4)
  .join(gendersUnpartitionedJsonDF.select($"id", $"gender").filter("gender != 'F'").where("gender != 'M'"),
  $"toUserId" === $"id")

df.explain(true)
df.count()
```

```
df: org.apache.spark.sql.DataFrame = [toUserId: bigint, rating: int, id: bigint, gender: string]
== Parsed Logical Plan ==
Join Inner, Some((toUserId#94L = id#92L))
 Filter (rating#95 >= 4)
  Filter (rating#95 <= 6)
   Project [toUserId#94L,rating#95]
    Relation[fromUserId#93L,toUserId#94L,rating#95] ParquetRelation[file:/root/pipeline/datasets/dating/ratings-partitioned.
parquet]
 Filter NOT (gender#91 = M)
  Filter NOT (gender#91 = F)
   Project [id#92L,gender#91]
    Relation[gender#91,id#92L] JSONRelation[file:/root/pipeline/datasets/dating/genders.json.bz2]

== Physical Plan ==
BroadcastHashJoin [toUserId#94L], [id#92L], BuildRight
 ConvertToUnsafe
  Scan ParquetRelation[file:/root/pipeline/datasets/dating/ratings-partitioned.parquet][toUserId#94L,rating#95]
 ConvertToUnsafe
  Filter (NOT (gender#91 = F) && NOT (gender#91 = M))
   Scan JSONRelation[file:/root/pipeline/datasets/dating/genders.json.bz2][id#92L,gender#91]
```

2 Extra Filter Passes

Filter pushdown!
No Extra Filter Pass

Filter collapse
1 Extra Filter Pass

Join Between Partitioned & Unpartitioned

Ratings Partitioned Parquet, Genders Unpartitioned JSON Join

FINISHED

```
val df = ratingsPartitionedParquetDF.select($"toUserId", $"rating").where($"rating" <= 6).filter($"rating" >= 4)
  .join(gendersUnpartitionedJsonDF.select($"id", $"gender").filter("gender != 'F'").where("gender != 'M'"),
  $"toUserId" === $"id")

df.explain()
df.count()
```

```
df: org.apache.spark.sql.DataFrame = [toUserId: bigint, rating: int, id: bigint, gender: string]
== Physical Plan ==
BroadcastHashJoin [toUserId#64L], [id#62L], BuildRight
 ConvertToUnsafe
  Scan ParquetRelation[file:/root/pipeline/datasets/dating/ratings-partitioned.parquet][toUserId#64L,rating#65]
 ConvertToUnsafe
  Filter (NOT (gender#61 = F) && NOT (gender#61 = M))
   Scan JSONRelation[file:/root/pipeline/datasets/dating/genders.json.bz2][id#62L,gender#61]
res26: Long = 1123909
```

Took 2 seconds.

Ratings Unpartitioned JSON, Genders Partitioned Parquet Join

FINISHED

```
val df = ratingsUnpartitionedJsonDF.select($"toUserId", $"rating").where($"rating" <= 6).filter($"rating" >= 4)
  .join(gendersPartitionedParquetDF.select($"id", $"gender").filter("gender != 'F'").where("gender != 'M'"),
  $"toUserId" === $"id")

df.explain()
df.count()
```

```
df: org.apache.spark.sql.DataFrame = [toUserId: bigint, rating: bigint, id: bigint, gender: string]
== Physical Plan ==
BroadcastHashJoin [toUserId#2L], [id#8L], BuildRight
 ConvertToUnsafe
  Filter ((rating#1L <= 6) && (rating#1L >= 4))
   Scan JSONRelation[file:/root/pipeline/datasets/dating/ratings.json.bz2][toUserId#2L,rating#1L]
 ConvertToUnsafe
  Scan ParquetRelation[file:/root/pipeline/datasets/dating/genders-partitioned.parquet][id#8L,gender#9]
res72: Long = 1123909
```

Took 30 seconds.

Join Between Partitioned & Partitioned

Ratings Partitioned Parquet, Genders Partitioned Parquet Join

FINISHED

```
val df = ratingsPartitionedParquetDF.select($"toUserId", $"rating").where($"rating" <= 6).filter($"rating" >= 4)
  .join(gendersPartitionedParquetDF.select($"id", $"gender").filter("gender != 'F'").where("gender != 'M'"),
  $"toUserId" === $"id")

df.explain()
df.count()
```

```
df: org.apache.spark.sql.DataFrame = [toUserId: bigint, rating: int, id: bigint, gender: string]
== Physical Plan ==
BroadcastHashJoin [toUserId#6L], [id#8L], BuildRight
 ConvertToUnsafe
  Scan ParquetRelation[file:/root/pipeline/datasets/dating/ratings-partitioned.parquet][toUserId#6L,rating#7]
 ConvertToUnsafe
  Scan ParquetRelation[file:/root/pipeline/datasets/dating/genders-partitioned.parquet][id#8L,gender#9]
res84: Long = 1123909
```

Took 1 seconds.

Cartesian Join vs. Inner Join

Cartesian Join

ERROR

```
val df = ratingsPartitionedParquetDF.select($"toUserId", $"rating").where($"rating" <= 6).filter($"rating" >= 4)
  .join(gendersPartitionedParquetDF.select($"id", $"gender").filter("gender != 'F'").where("gender != 'M'"))

df.explain()
df.count()
```

```
df: org.apache.spark.sql.DataFrame = [toUserId: bigint, rating: int, id: bigint, gender: string]
== Physical Plan ==
CartesianProduct
 Scan ParquetRelation[file:/root/pipeline/datasets/dating/ratings-partitioned.parquet][toUserId#6L,rating#7]
 Scan ParquetRelation[file:/root/pipeline/datasets/dating/genders-partitioned.parquet][id#8L,gender#9]
org.apache.spark.SparkException: Job aborted due to stage failure: Task 1 in stage 4.0 failed 4 times, most recent failure:
Lost task 1.3 in stage 4.0 (TID 25, 127.0.0.1): ExecutorLostFailure (executor 5 lost)
Driver stacktrace:
        at org.apache.spark.scheduler.DAGScheduler.org$apache$spark$scheduler$DAGScheduler$$failJobAndIndependentStages(DAGS
```

Inner Join

FINISHED

```
val df = ratingsPartitionedParquetDF.select($"toUserId", $"rating").where($"rating" <= 6).filter($"rating" >= 4)
  .join(gendersPartitionedParquetDF.select($"id", $"gender").filter("gender != 'F'").where("gender != 'M'"),
  $"toUserId" === $"id")

df.explain()
df.count()
```

```
df: org.apache.spark.sql.DataFrame = [toUserId: bigint, rating: int, id: bigint, gender: string]
== Physical Plan ==
BroadcastHashJoin [toUserId#64L], [id#66L], BuildRight
 ConvertToUnsafe
  Scan ParquetRelation[file:/root/pipeline/datasets/dating/ratings-partitioned.parquet][toUserId#64L,rating#65]
 ConvertToUnsafe
  Scan ParquetRelation[file:/root/pipeline/datasets/dating/genders-partitioned.parquet][id#66L,gender#67]
res34: Long = 1123909
```

Took 1 seconds.

Broadcast Join vs. Normal Shuffle Join

Broadcast Join

FINISHED

```
sqlContext.sql("set spark.sql.autoBroadcastJoinThreshold=10485760") // default = 10 MB

val df = ratingsPartitionedParquetDF.select($"toUserId", $"rating").where($"rating" <= 6).filter($"rating" >= 4)
  .join(gendersPartitionedParquetDF.select($"id", $"gender").filter("gender != 'F'").where("gender != 'M'"),
    $"toUserId" === $"id")

df.explain()
df.count()
```

```
res36: org.apache.spark.sql.DataFrame = [key: string, value: string]
df: org.apache.spark.sql.DataFrame = [toUserId: bigint, rating: int, id: bigint, gender: string]
== Physical Plan ==
BroadcastHashJoin [toUserId#64L], [id#66L], BuildRight
 ConvertToUnsafe
  Scan ParquetRelation[file:/root/pipeline/datasets/dating/ratings-partitioned.parquet][toUserId#64L,rating#65]
 ConvertToUnsafe
  Scan ParquetRelation[file:/root/pipeline/datasets/dating/genders-partitioned.parquet][id#66L,gender#67]
res40: Long = 1123909
```

Took 1 seconds.

Normal Shuffle Join

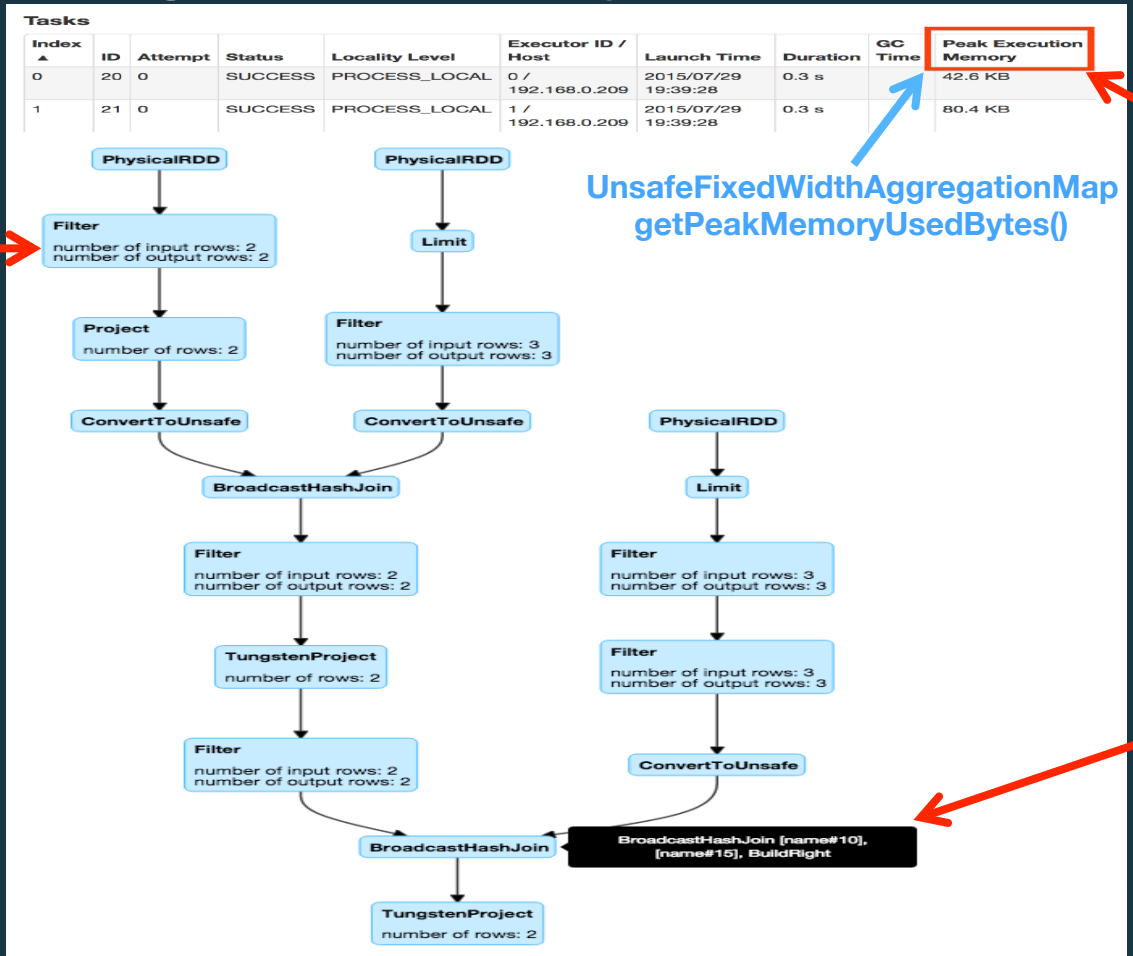
FINISHED

```
sqlContext.sql("set spark.sql.autoBroadcastJoinThreshold=-1")

val df = ratingsPartitionedParquetDF.select($"toUserId", $"rating").where($"rating" <= 6).filter($"rating" >= 4)
  .join(gendersPartitionedParquetDF.select($"id", $"gender").filter("gender != 'F'").where("gender != 'M'"),
    $"toUserId" === $"id")

df.explain()
df.count()
```

```
res50: org.apache.spark.sql.DataFrame = [key: string, value: string]
df: org.apache.spark.sql.DataFrame = [toUserId: bigint, rating: int, id: bigint, gender: string]
== Physical Plan ==
SortMergeJoin [toUserId#64L], [id#66L]
 TungstenSort [toUserId#64L ASC], false, 0
  TungstenExchange hashpartitioning(toUserId#64L)
   ConvertToUnsafe
    Scan ParquetRelation[file:/root/pipeline/datasets/dating/ratings-partitioned.parquet][toUserId#64L,rating#65]
 TungstenSort [id#66L ASC], false, 0
  TungstenExchange hashpartitioning(id#66L)
   ConvertToUnsafe
    Scan ParquetRelation[file:/root/pipeline/datasets/dating/genders-partitioned.parquet][id#66L,gender#67]
res54: Long = 1123909
```

Took 4 seconds.

Visualizing the Query Plan



Effectiveness of Filter

UnsafeFixedWidthAggregationMap getPeakMemoryUsedBytes()

Peak Memory for Joins and Aggs

Cost-based Join Optimization Similar to MapReduce Map-side Join & DistributedCache

Data Source API

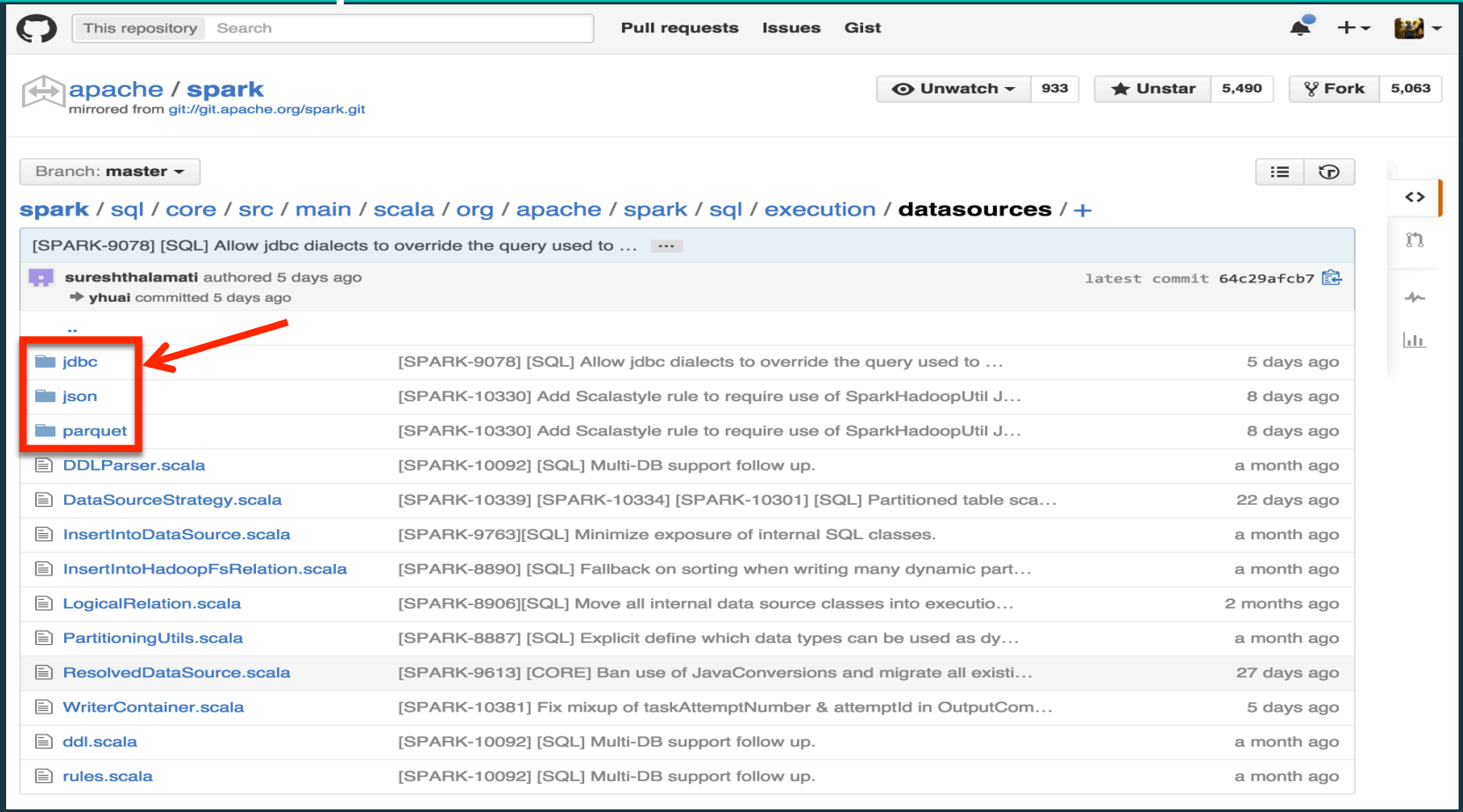
Relations (o.a.s.sql.sources.interfaces.scala)

- BaseRelation (abstract class): Provides schema of data
  - TableScan (impl): Read all data from source
  - PrunedFilteredScan (impl): Column pruning & predicate pushdowns
  - InsertableRelation (impl): Insert/overwrite data based on SaveMode
- RelationProvider (trait/interface): Handle options, BaseRelation factory

Filters (o.a.s.sql.sources.filters.scala)

- Filter (abstract class): Handles all filters supported by this source
  - EqualTo (impl)
  - GreaterThan (impl)
  - StringStartsWith (impl)

Native Spark SQL Data Sources



JSON Data Source

DataFrame

```
val ratingsDF = sqlContext.read.format("json")
 .load("file:/root/pipeline/datasets/dating/ratings.json.bz2")
```

-- or –

```
val ratingsDF = sqlContext.read.json
 ("file:/root/pipeline/datasets/dating/ratings.json.bz2")
```

json() convenience method

SQL Code

```
CREATE TABLE genders USING json
OPTIONS
 (path "file:/root/pipeline/datasets/dating/genders.json.bz2")
```

Parquet Data Source

Configuration

spark.sql.parquet.filterPushdown=true
spark.sql.parquet.mergeSchema=false (unless your schema is evolving)
spark.sql.parquet.cacheMetadata=true (requires sqlContext.refreshTable())
spark.sql.parquet.compression.codec=[uncompressed,snappy,gzip,lzo]

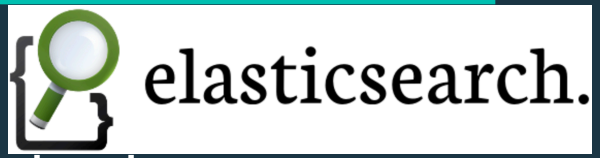
DataFrames

```
val gendersDF = sqlContext.read.format("parquet")
  .load("file:/root/pipeline/datasets/dating/genders.parquet")
gendersDF.write.format("parquet").partitionBy("gender")
  .save("file:/root/pipeline/datasets/dating/genders.parquet")
```

SQL

```
CREATE TABLE genders USING parquet
OPTIONS
    (path "file:/root/pipeline/datasets/dating/genders.parquet")
```

ElasticSearch Data Source



Github

https://github.com/elastic/elasticsearch-hadoop

Maven

org.elasticsearch:elasticsearch-spark_2.10:2.1.0

Code

```
val esConfig = Map("pushdown" -> "true", "es.nodes" -> "<hostname>",
                   "es.port" -> "<port>")
df.write.format("org.elasticsearch.spark.sql").mode(SaveMode.Overwrite)
    .options(esConfig).save("<index>/<document-type>")
```

Cassandra Data Source

Github

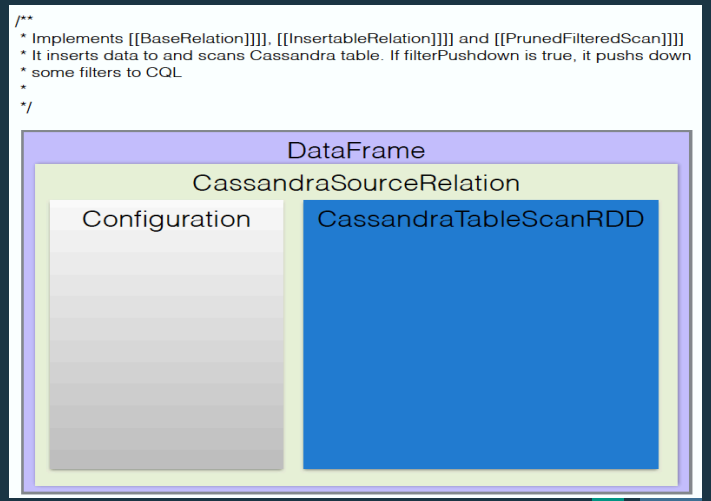
https://github.com/datastax/spark-cassandra-connector

Maven

com.datastax.spark:spark-cassandra-connector_2.10:1.5.0-M1

Code

```
ratingsDF.write
    .format("org.apache.spark.sql.cassandra")
    .mode(SaveMode.Append)
    .options(Map("keyspace"->"<keyspace>",
                 "table"->"<table>")).save(…)
```



Tips for Cassandra Analytics

By-pass Cassandra CQL "front door"

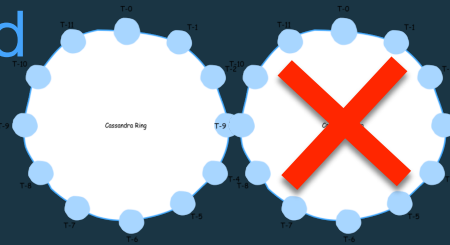
- CQL Optimized for Transactions

Bulk read and write directly against SSTables

- Check out Netflix OSS project "Aegisthus"

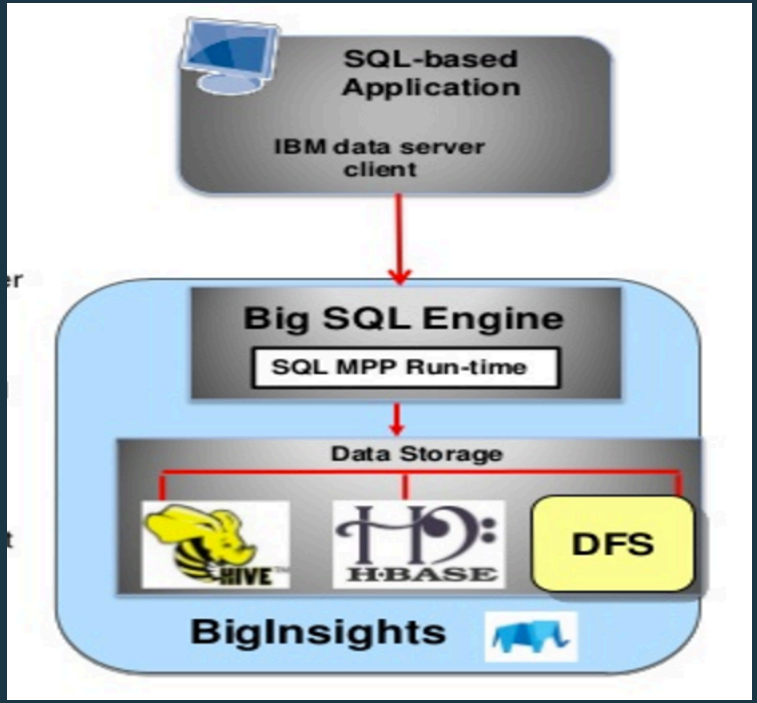
Cassandra becomesa first-class analytics option

- Replicated analytics cluster no longer needed



DB2 and BigSQL Data Sources (IBM)

Coming Soon!



Creating a Custom Data Source

① Study existing implementations
o.a.s.sql.execution.datasources.jdbc.JDBCRelation

② Extend base traits & implement required methods
o.a.s.sql.sources.{BaseRelation,PrunedFilterScan}

Spark JDBC (o.a.s.sql.execution.datasources.jdbc)

```
class JDBCRelation extends BaseRelation
    with PrunedFilteredScan
    with InsertableRelation
```

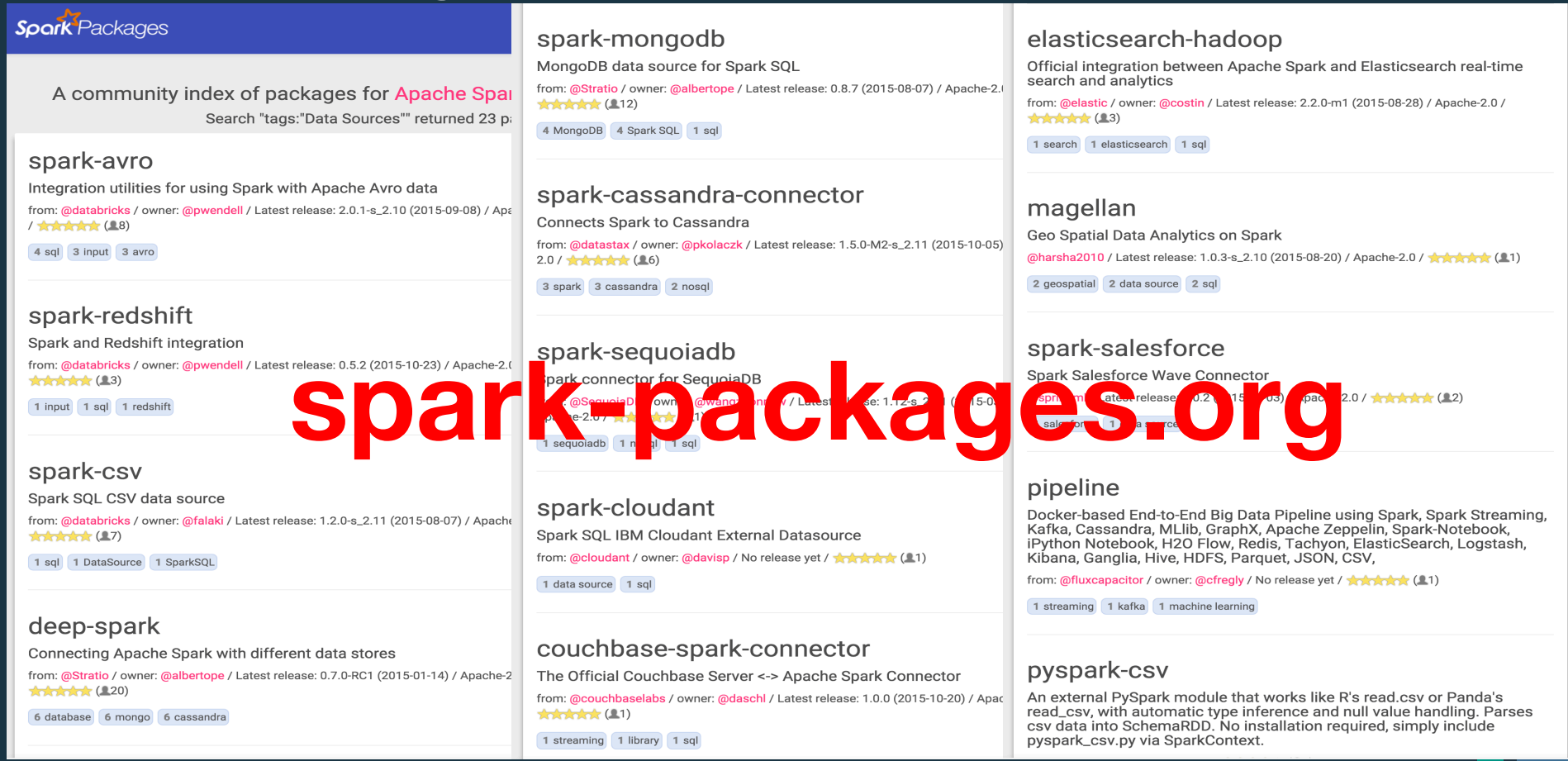
DataStax Cassandra (o.a.s.sql.cassandra)

```
class CassandraSourceRelation extends BaseRelation
    with PrunedFilteredScan
    with InsertableRelation
```

Demo!

Create a Custom Data Source

Publishing Custom Data Sources



spark-packages.org

Spark SQL UDF Code Generation

100+ UDFs now generating code

More to come in Spark 1.6+

Every UDF must use Expressions and implement Expression.genCode() to participate in the fun

Lambdas (RDD or Dataset API) and sqlContext.udf.registerFunction() are not enough!!

Details in SPARK-8159, SPARK-9571

Creating a Custom UDF with Code Gen

① Study existing implementations

o.a.s.sql.catalyst.expressions.Substring

② Extend and implement base trait

o.a.s.sql.catalyst.expressions.Expression.genCode

③ Don’t forget about Python!

python.pyspark.sql.functions.py

Demo!

Creating a Custom UDF participating in Code Generation

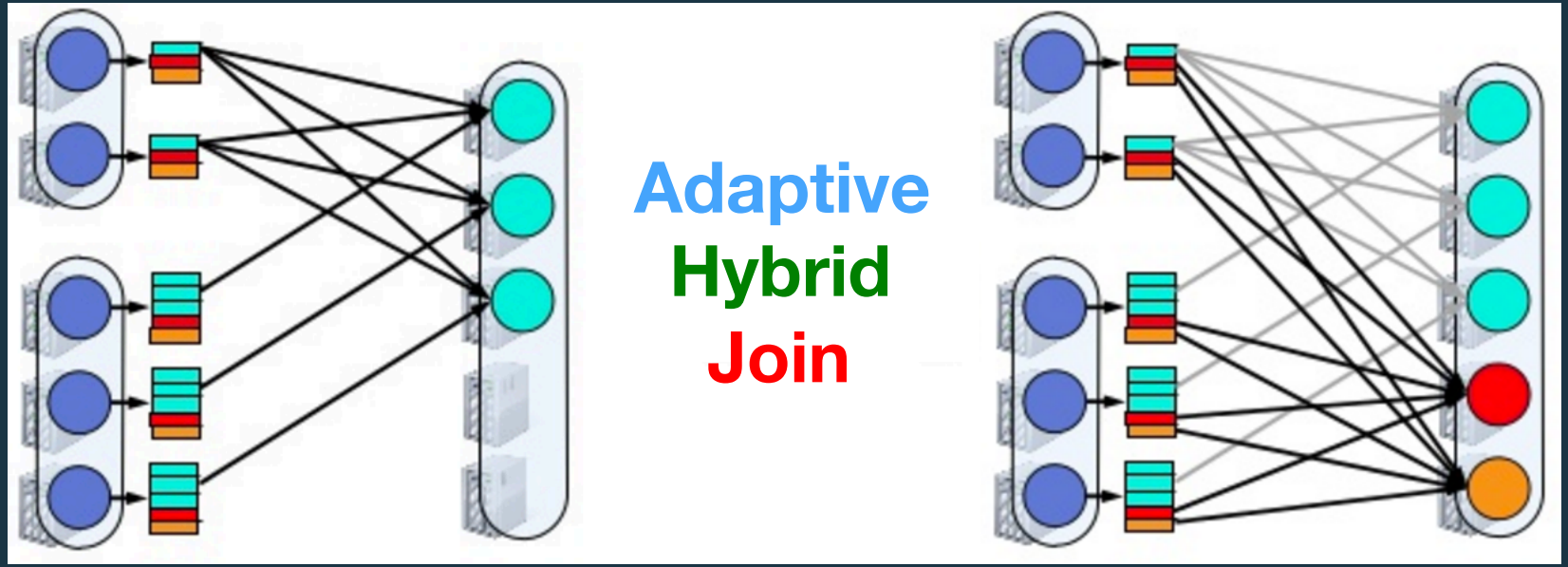
Spark 1.6 Improvements

Adaptiveness, Metrics, Datasets, and Streaming State

Adaptive Query Execution

Adapt query execution using data from previous stages

Dynamically choose spark.sql.shuffle.partitions (default 200)



Broadcast Join
(popular keys)

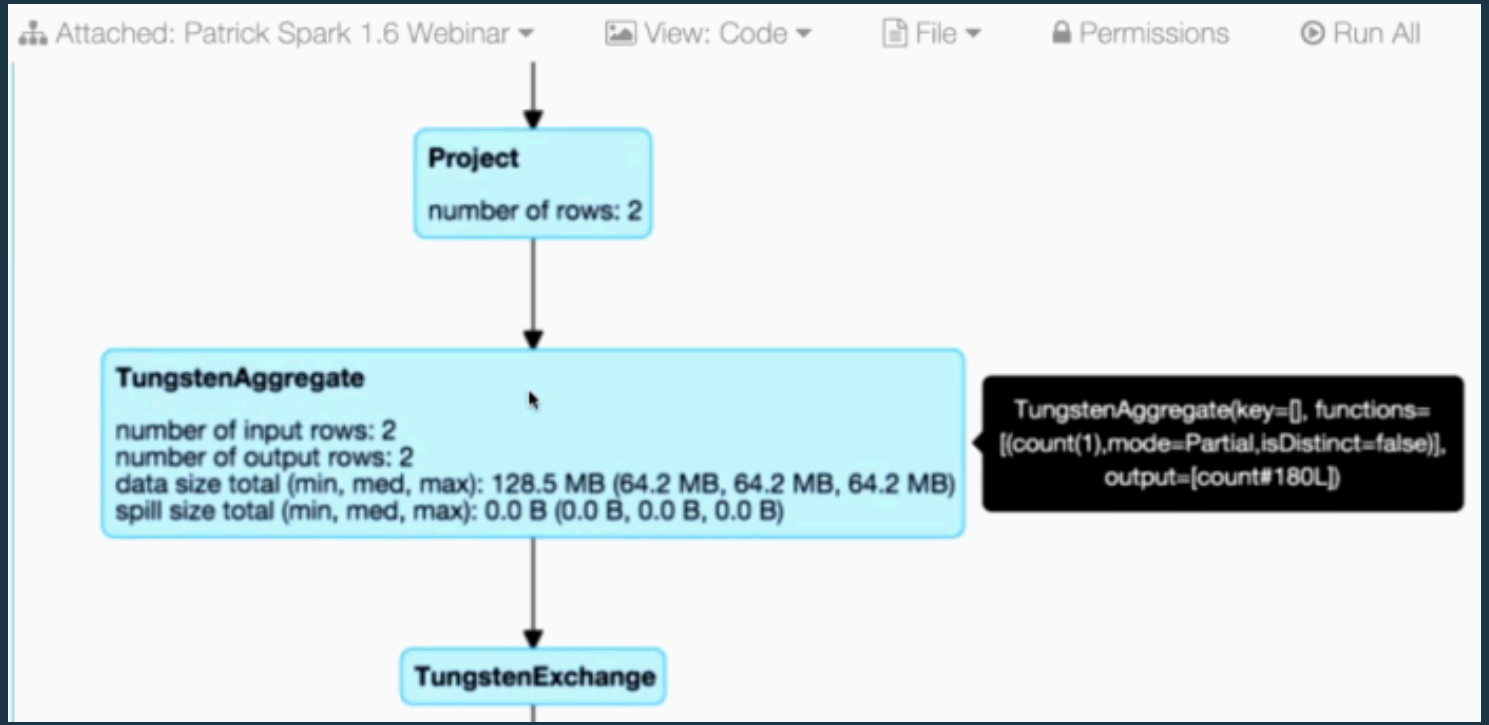
Shuffle Join
(not-so-popular keys)

Adaptive Memory Management

- Spark <1.6
  - Manual configure between 2 memory regions
    - Spark execution engine (shuffles, joins, sorts, aggs)
      - spark.shuffle.memoryFraction
    - RDD Data Cache
      - spark.storage.memoryFraction
- Spark 1.6+
  - Unified memory regions
  - Dynamically expand/contract memory regions
  - Supports minimum for RDD storage (LRU Cache)

Metrics

Shows exact memory usage per operator & per node
Helps debugging and identifying skew



Spark SQL API

Datasets type safe API (similar to RDDs) utilizing Tungsten

```
val ds = sqlContext.read.text("ratings.csv").as[String]
val df = ds.flatMap(_.split(",")).filter(_ != "").toDF()  // RDD API, convert to DF
val agg = df.groupBy($"rating").agg(count("*") as "ct").orderBy($"ct" desc)
```

Typed Aggregators used alongside UDFs and UDAFs

```
val simpleSum = new Aggregator[Int, Int, Int] with Serializable {
  def zero: Int = 0
  def reduce(b: Int, a: Int) = b + a
  def merge(b1: Int, b2: Int) = b1 + b2
  def finish(b: Int) = b
}.toColumn
val sum = Seq(1,2,3,4).toDS().select(simpleSum)
```

Query files directly without registerTempTable()

```
%sql SELECT * FROM json.`/datasets/movielens/ml-latest/movies.json`
```

Spark Streaming State Management

New trackStateByKey()

- Store deltas, compact later
- More efficient per-key state update
- Session TTL

Integrated A/B Testing (?!)

Show Failed Output in Admin UI

- Better debugging

Thank You!!!

Chris Fregly

IBM Spark Tech Center

(http://spark.tc)

San Francisco, California, USA

advancedspark.com

Sign up for the Meetup and Book

Contribute on Github

Run All Demos in Docker

2100+ Docker Downloads!!

Find me on LinkedIn, Twitter, Github, Email, Fax

What's Next?

Advanced Spark

Upcoming Features of Advanced Spark

Autoscaling Spark Workers

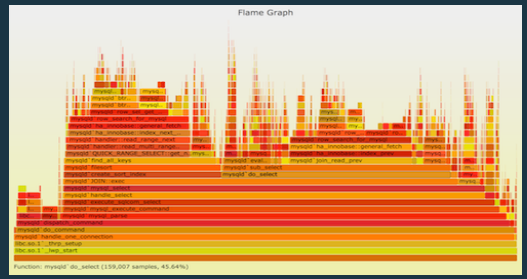
- Completely Docker-based
- Docker Compose, Google Kubernetes

Lots of Demos and Examples

- More Zeppelin, iPython/Jupyter notebooks
- More advanced analytics use cases

Performance Profiling

- Work closely with Netflix and Databricks
- Identify and fix performance bottlenecks



World Tour: Freg-a-Palooza!

London Spark Meetup (Oct 12th)

Scotland Data Science Meetup (Oct 13th)

Dublin Spark Meetup (Oct 15th)

Barcelona Spark Meetup (Oct 2[illegible])

Madrid Big Data Meetup (Oct 2[illegible])

Paris Spark Meetup (Oct 26th)

Amsterdam Spark Summit (Oct 27th)

Brussels Spark Meetup (Oct 30th)

Zurich Big Data Meetup (Nov 2nd)

Geneva Spark Meetup (Nov 5th)

Oslo Big Data Hadoop Meetup (Nov 19th)

Helsinki Spark Meetup (Nov 20th)

Stockholm Spark Meetup (Nov 23rd)

Copenhagen Spark Meetup (Nov 25th)

Budapest Spark Meetup (Nov 26th)

Istanbul Spark Meetup (Nov 28th)

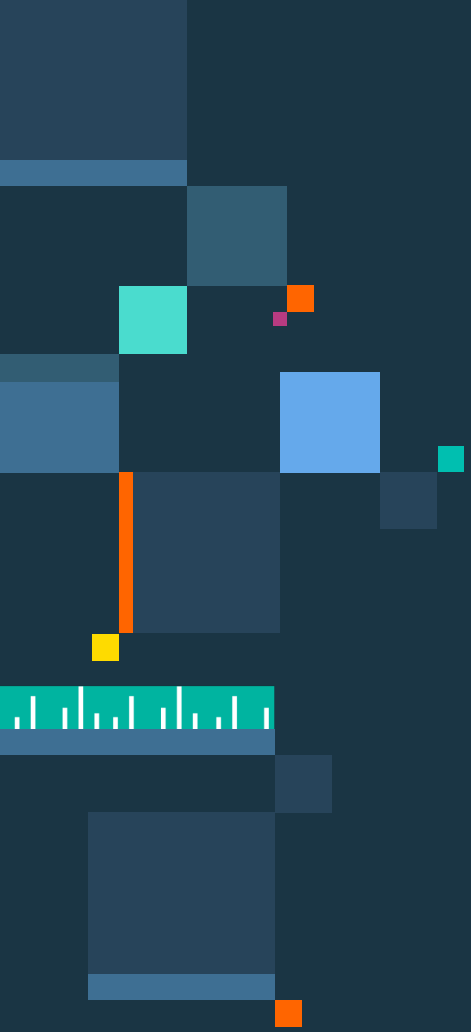
Singapore Spark Meetup (Dec 1st)

Sydney Spark Meetup (Dec 8th)

Melbourne Spark Meetup (Dec 9th)

Toronto Spark Meetup (Dec 14th)

I'M DONE FOR 2015!!



{spark.tc}

IBM | Spark

Power of data. **Simplicity** of design. **Speed** of innovation.

advancedspark.com

@cfregly