DATA+AI SUMMIT EUROPE
FORMERLY SPARK+AI SUMMIT

What's New with Spark Performance Monitoring in Apache Spark 3.0

Luca Canali
Data Engineer, CERN

Agenda

Intro and Motivations

Spark Monitoring

- Recap of the available instrumentation

Spark 3.0 Improvements

- Memory monitoring
- Plugins for custom monitoring
- Web UI: SQL tab, Streaming monitoring

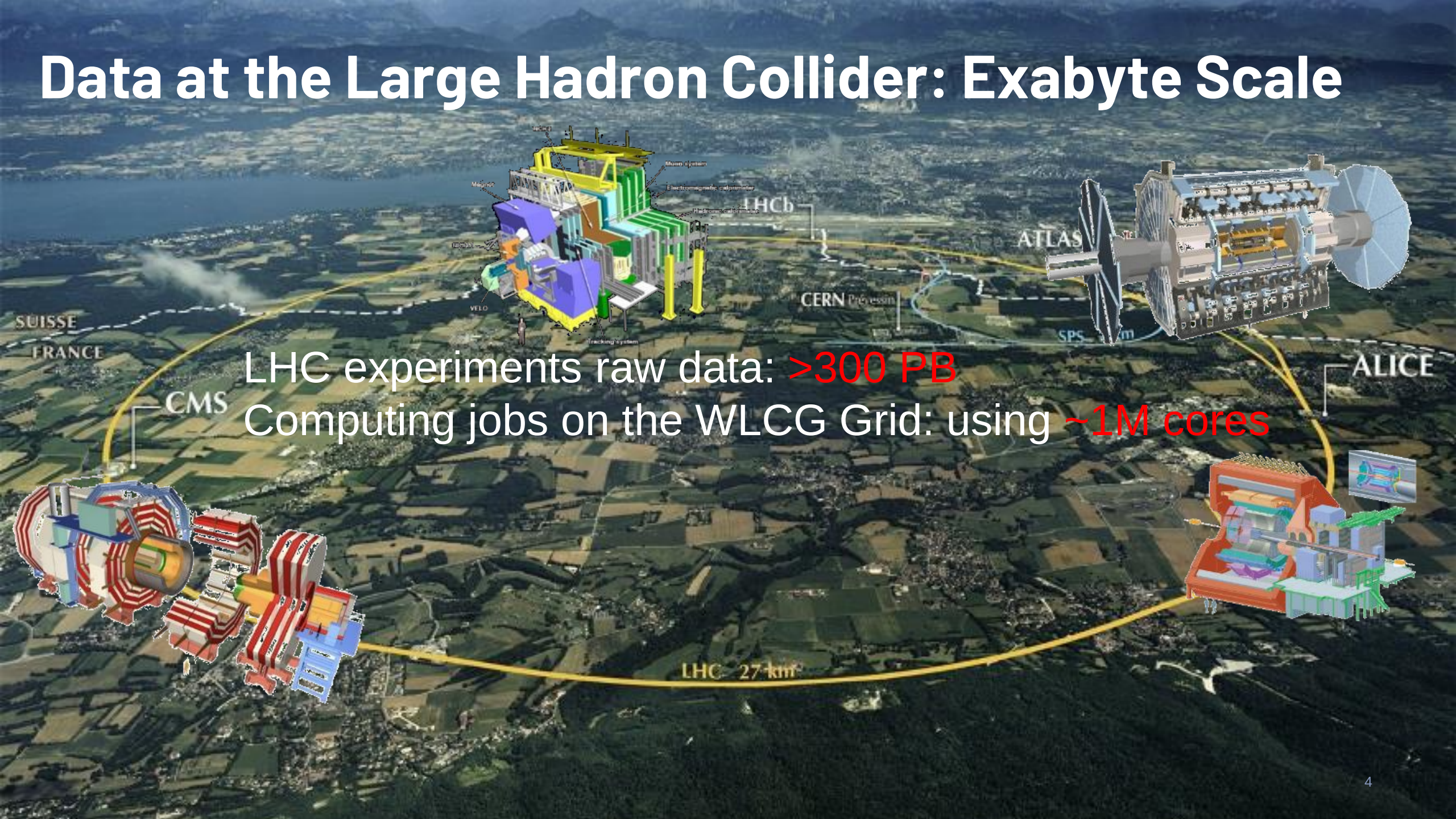
Outlook and Community

About Luca

- Data Engineer at CERN
  - Hadoop and Spark service, database services
  - 20+ years with databases and data engineering
  - Performance engineering
- Blog, tools, contributions to Apache Spark

 @LucaCanaliDB – http://cern.ch/canali

Data at the Large Hadron Collider: Exabyte Scale

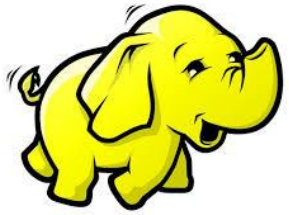


LHC experiments raw data: >300 PB

Computing jobs on the WLCG Grid: using ~1M cores

Apache Spark Clusters at CERN

- Spark running on clusters:
  - YARN/Hadoop -> established
  - Spark on Kubernetes -> growing adoption

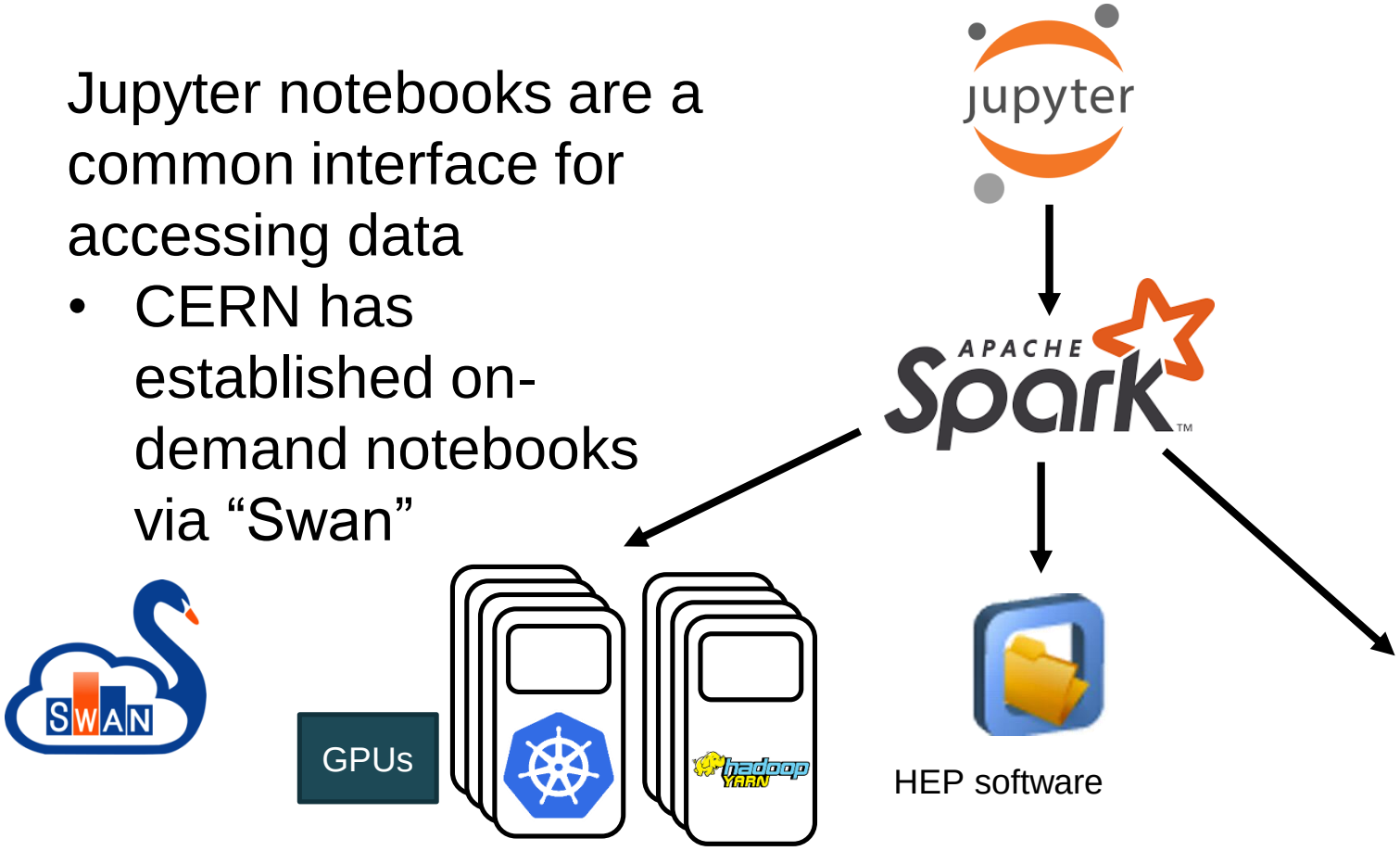


Accelerator logging (part of LHC infrastructure)	Hadoop - YARN - 30 nodes (Cores - 1200, Mem - 13 TB, Storage – 7.5 PB)
General Purpose	Hadoop - YARN, 39 nodes (Cores – 1.7k, Mem – 18 TB, Storage – 12 PB)
Cloud containers	Kubernetes on Openstack VMs, Cores - 270, Mem – 2 TB Storage: remote HDFS or custom storage (CERN EOS, for physics data, S3 on Ceph also available). Note: GPU resources available.

Analytics Platform Outlook

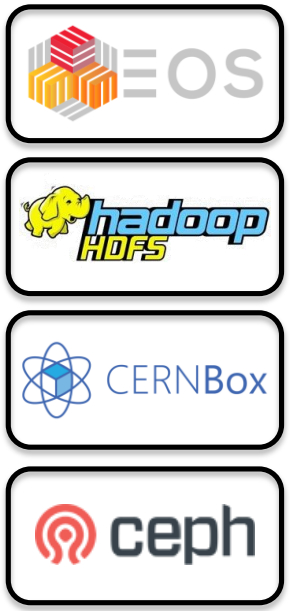
Jupyter notebooks are a common interface for accessing data

- CERN has established on-demand notebooks via “Swan”



Integrating with existing infrastructure:

- Software
- Data



Experiments storage

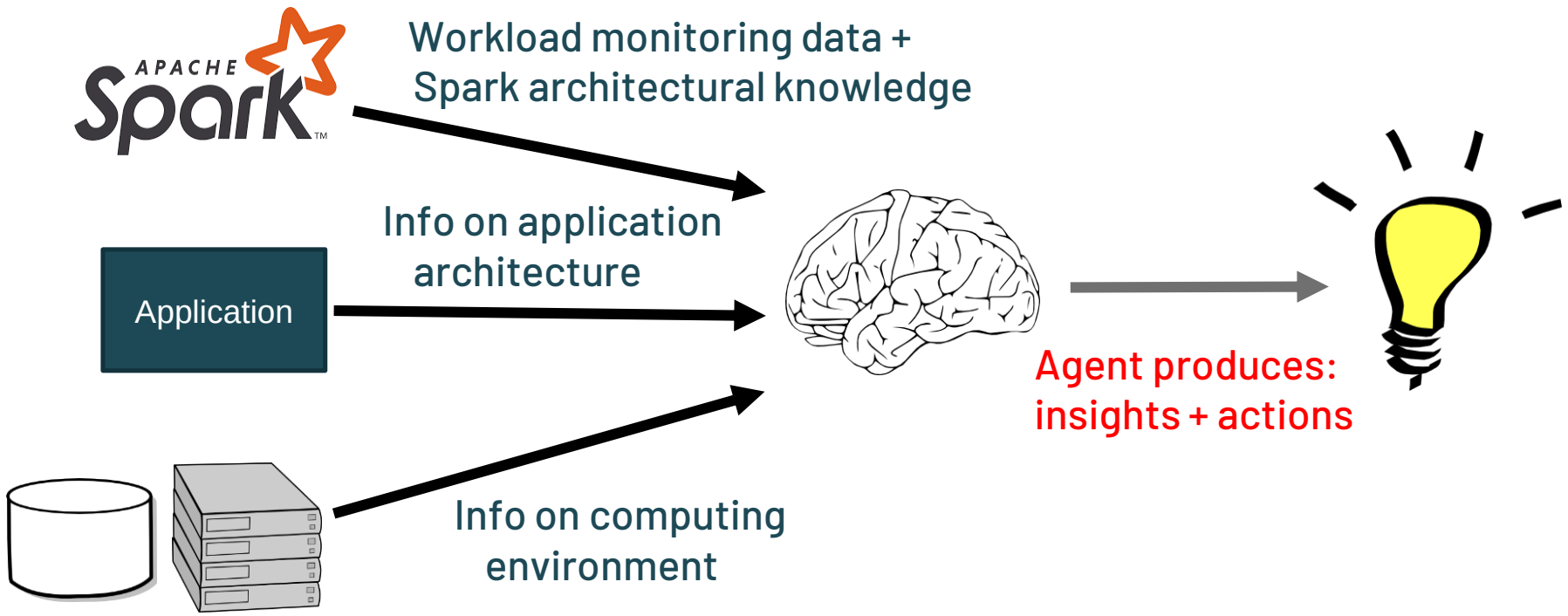
HDFS

Personal storage

Other Cloud storage, S3 on Ceph

Monitoring, Measuring and Troubleshooting

- Collect and expose metrics (Spark, Cluster, OS)
- For troubleshooting and insights on performance



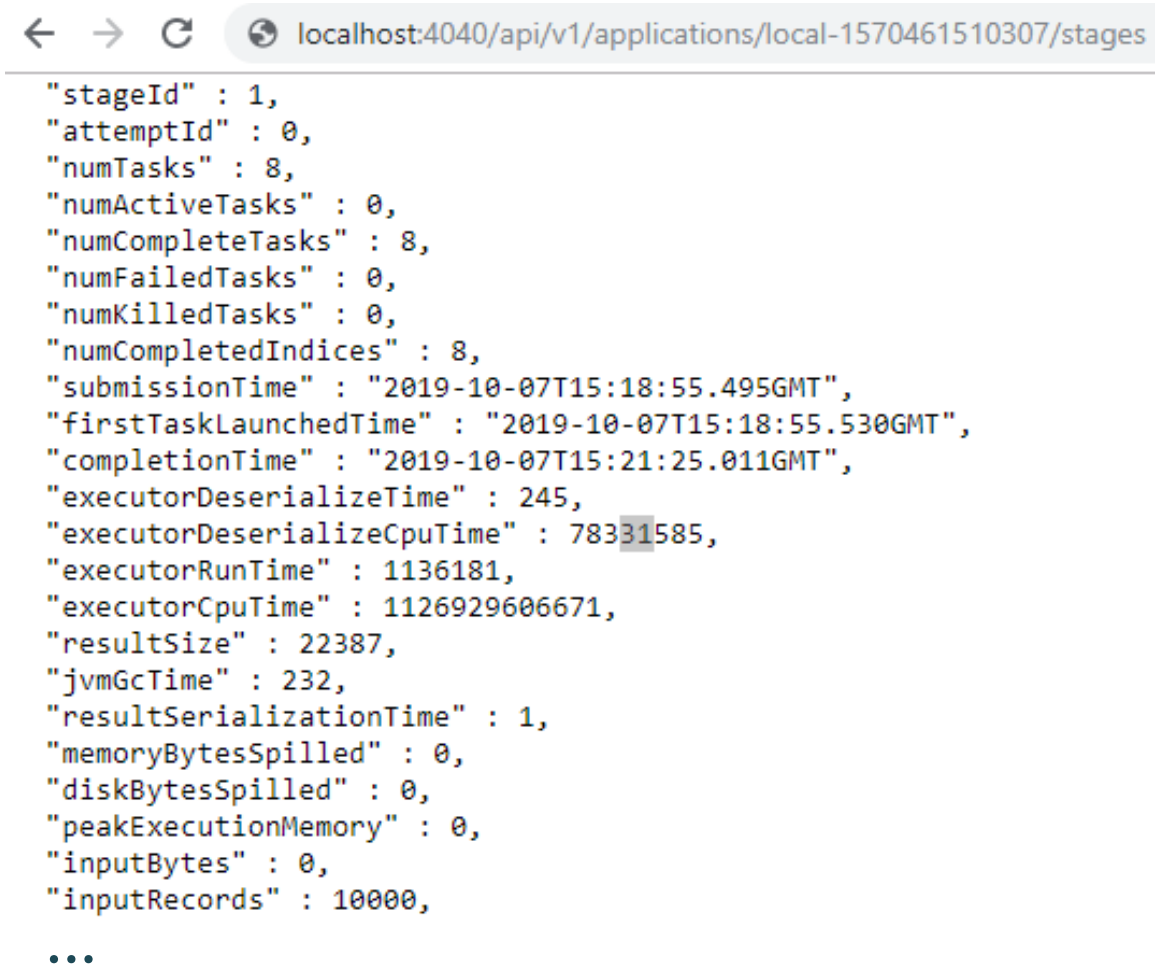
Monitoring Features in Apache Spark, a Recap

Web UI

- First and main point of access to monitor Spark applications
- Details on jobs, stages, tasks, SQL, streaming, etc
- Default URL: http://driver_host:4040

- New doc in Spark 3.0: https://spark.apache.org/docs/latest/web-ui.html

Spark Metrics with the REST API



Spark Listeners: @DeveloperApi

- Custom class, extends SparkListener

```
class StageInfoRecorderListener extends SparkListener {
```

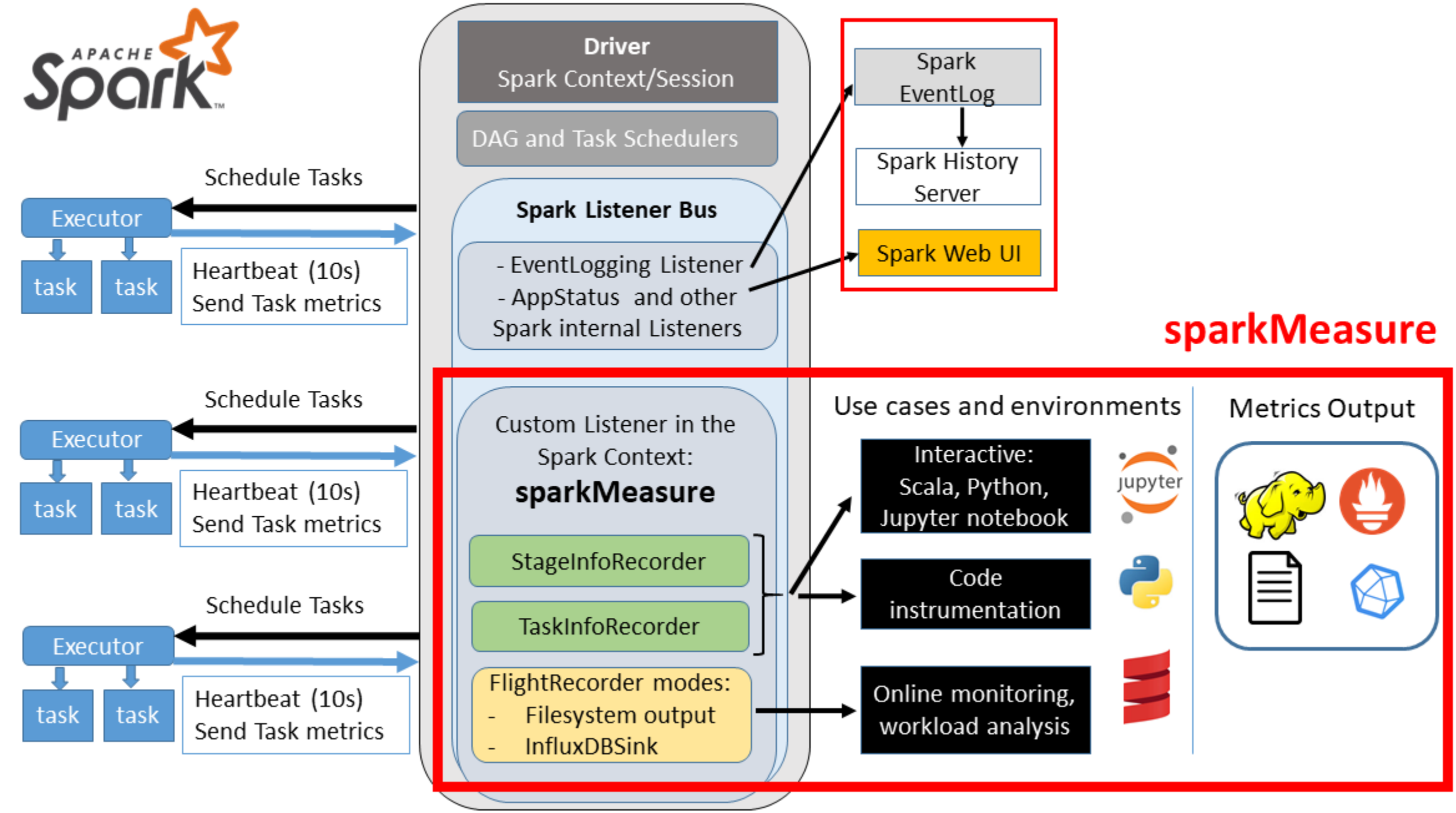
- Methods react on events to collect data, example:

```
override def onStageCompleted(stageCompleted: SparkListenerStageCompleted): Unit = {
  val stageInfo = stageCompleted.stageInfo
  val taskMetrics = stageInfo.taskMetrics
  val jobId = StageIdtoJobId(stageInfo.stageId)
```

- Attach custom Listener class to Spark Session

```
--conf spark.extraListeners=..
```

Spark Task Metrics, the Listener Bus and SparkMeasure Architecture



Tooling: sparkMeasure

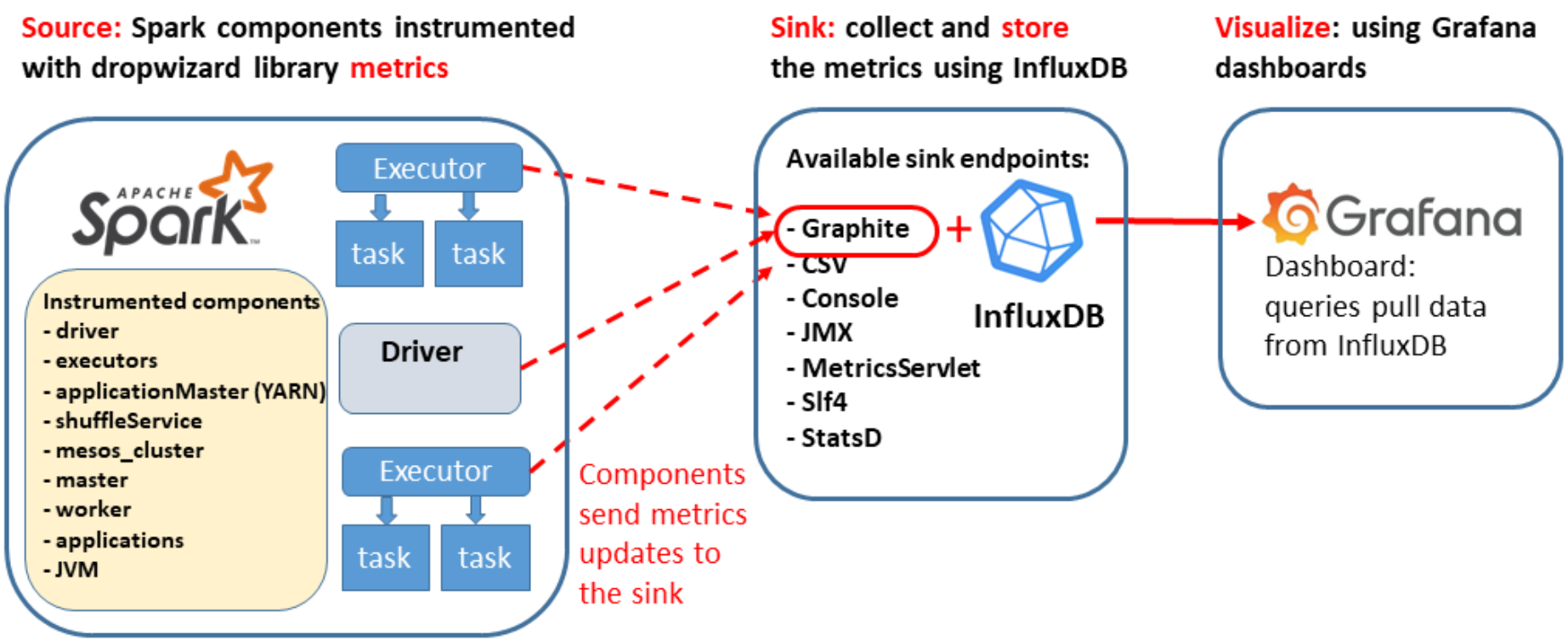
```
bin/spark-shell --packages ch.cern.sparkmeasure:spark-measure_2.12:0.17

val stageMetrics = ch.cern.sparkmeasure.StageMetrics(spark)

val myQuery = "select count(*) from range(1000) cross join range(1000) cross join range(1000)"

stageMetrics.runAndMeasure(spark.sql(myQuery).show())
```

Task Metrics Measured with sparkMeasure

```
Spark Context default degree of parallelism = 8
Aggregated Spark stage metrics:
numStages => 3
numTasks => 17
elapsedTime => 13520 (14 s)
stageDuration => 13411 (13 s)
executorRunTime => 100020 (1.7 min)
executorCpuTime => 98899 (1.6 min)
executorDeserializeTime => 4358 (4 s)
executorDeserializeCpuTime => 1887 (2 s)
resultSerializationTime => 2 (2 ms)
jvmGCTime => 56 (56 ms)
shuffleFetchWaitTime => 0 (0 ms)
shuffleWriteTime => 11 (11 ms)
resultSize => 19955 (19.0 KB)
diskBytesSpilled => 0 (0 Bytes)
memoryBytesSpilled => 0 (0 Bytes)
peakExecutionMemory => 0
recordsRead => 2000
bytesRead => 0 (0 Bytes)
recordsWritten => 0
bytesWritten => 0 (0 Bytes)
shuffleRecordsRead => 8
shuffleTotalBlocksFetched => 8
shuffleLocalBlocksFetched => 8
shuffleRemoteBlocksFetched => 0
shuffleTotalBytesRead => 472 (472 Bytes)
shuffleLocalBytesRead => 472 (472 Bytes)
shuffleRemoteBytesRead => 0 (0 Bytes)
shuffleRemoteBytesReadToDisk => 0 (0 Bytes)
shuffleBytesWritten => 472 (472 Bytes)
shuffleRecordsWritten => 8
```

Further Instrumentation in Spark: the Spark Metrics System

Apache Spark Metrics System + InfluxDB + Grafana => Dashboard



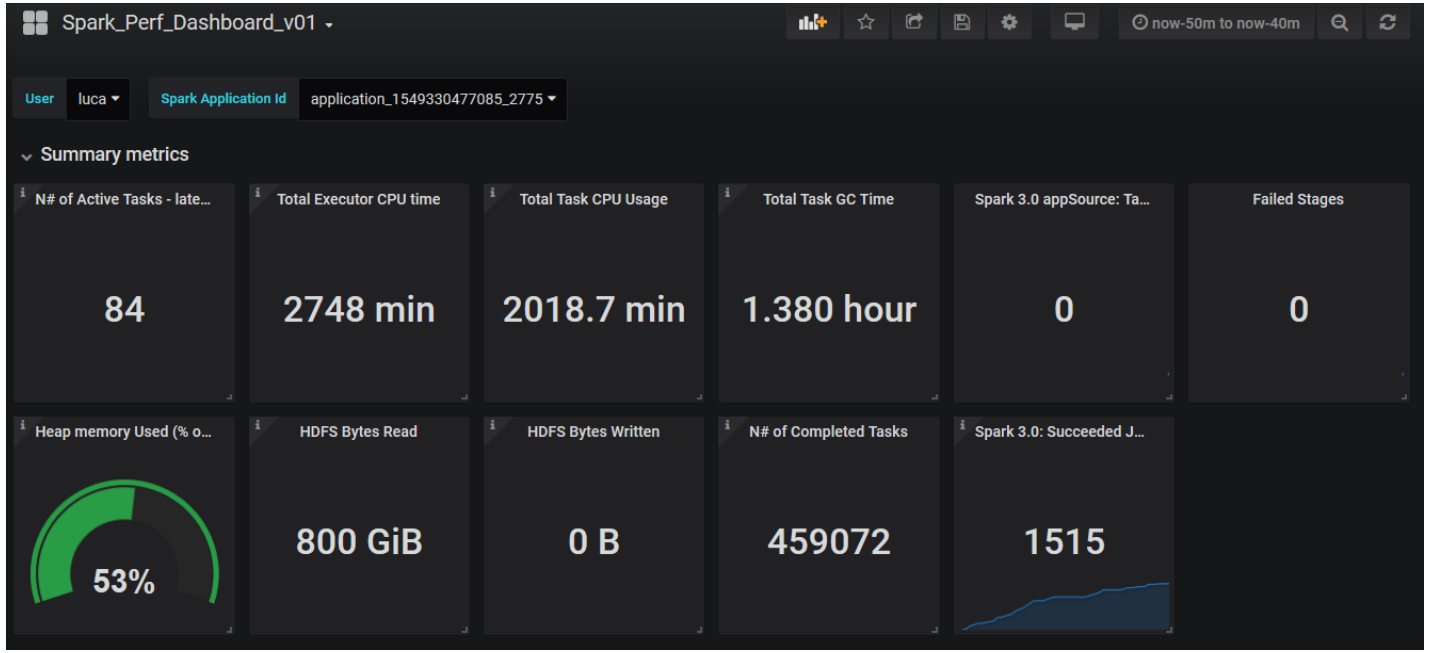
Send Spark Metrics to a Sink

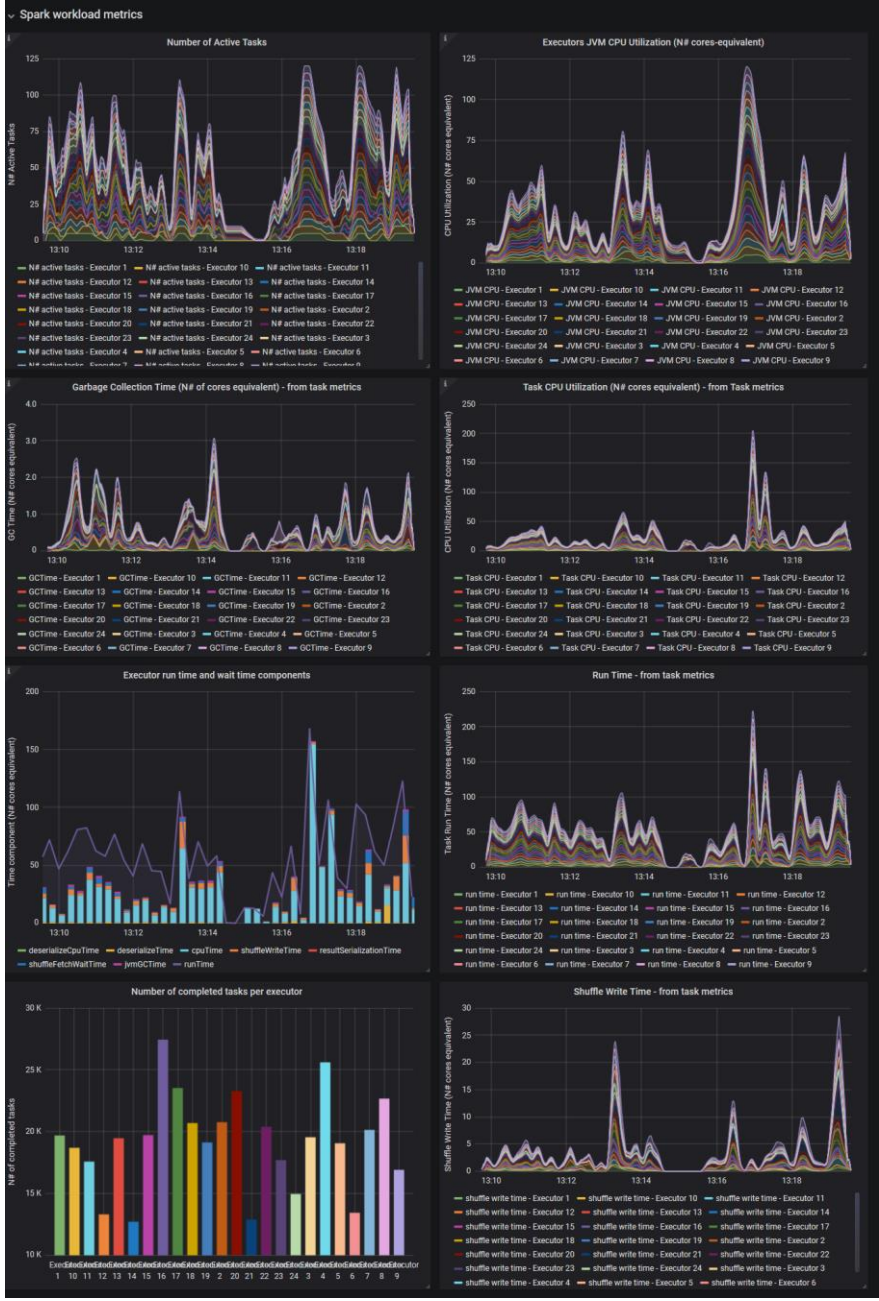
- Edit $SPARK_HOME/conf/metrics.properties
- Alternative: use the config parameters spark.metrics.conf.*

```
$ SPARK_HOME/bin/spark-shell \
--conf "spark.metrics.conf.driver.sink.graphite.class"="org.apache.spark.metrics.sink.GraphiteSink"\
--conf "spark.metrics.conf.executor.sink.graphite.class"="org.apache.spark.metrics.sink.GraphiteSink"
--conf "spark.metrics.conf.*.sink.graphite.host"="graphiteEndPoint_influxDB_hostName>" \
--conf "spark.metrics.conf.*.sink.graphite.port"=<graphite_listening_port> \
--conf "spark.metrics.conf.*.sink.graphite.period"=10 \
--conf "spark.metrics.conf.*.sink.graphite.unit"=seconds \
--conf "spark.metrics.conf.*.sink.graphite.prefix"="lucatest" \
--conf "spark.metrics.conf.*.source.jvm.class"="org.apache.spark.metrics.source.JvmSource"
```

Spark Grafana Dashboard

- Visualize Spark metrics
  - In real time and historical data
  - Summaries and time series of key metrics
- Data for drill-down and root-cause analysis





What's New in Apache Spark 3.0

Memory Usage Monitoring in Spark 3.0

- The problem we want to solve
  - Memory is key for the performance and stability of Spark jobs
  - Java OOM (out of memory) errors can be hard to troubleshoot
  - We want to allocate the needed memory, not overshooting/wasting

- New "Executor Metrics" in Spark 3.0
  - Measure details of memory usage by memory component
  - Peak values measurements (you get OOM on peak allocation not average)
  - Spark Metrics System: report memory metrics as time series

Executor Metrics, REST API

REST API:
peak memory usage values per executor

WebUI URL +
/api/v1/applications/<application_id>/executors

Extra tunables:

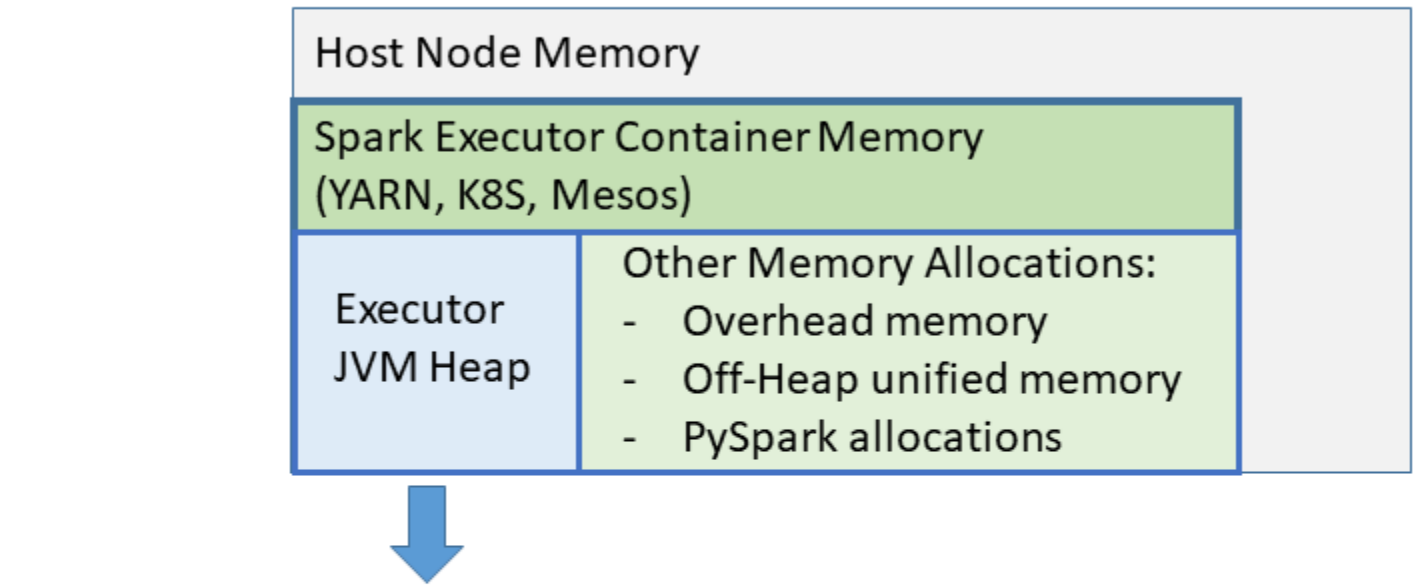
spark.executor.processTreeMetrics.enabled=true

spark.executor.metrics.pollingInterval=<time in ms>


```
"peakMemoryMetrics" : {
  "JVMHeapMemory" : 29487812552,
  "JVMOffHeapMemory" : 149957200,
  "OnHeapExecutionMemory" : 12458956272,
  "OffHeapExecutionMemory" : 0,
  "OnHeapStorageMemory" : 83578970,
  "OffHeapStorageMemory" : 0,
  "OnHeapUnifiedMemory" : 12540212490,
  "OffHeapUnifiedMemory" : 0,
  "DirectPoolMemory" : 66809076,
  "MappedPoolMemory" : 0,
  "ProcessTreeJVMVMemory" : 38084534272,
  "ProcessTreeJVMRSSMemory" : 36998328320,
  "ProcessTreePythonVMemory" : 0,
  "ProcessTreePythonRSSMemory" : 0,
  "ProcessTreeOtherVMemory" : 0,
  "ProcessTreeOtherRSSMemory" : 0,
  "MinorGCCount" : 561,
  "MinorGCTime" : 49918,
  "MajorGCCount" : 0,
  "MajorGCTime" : 0
},
```

Executor Memory Configuration

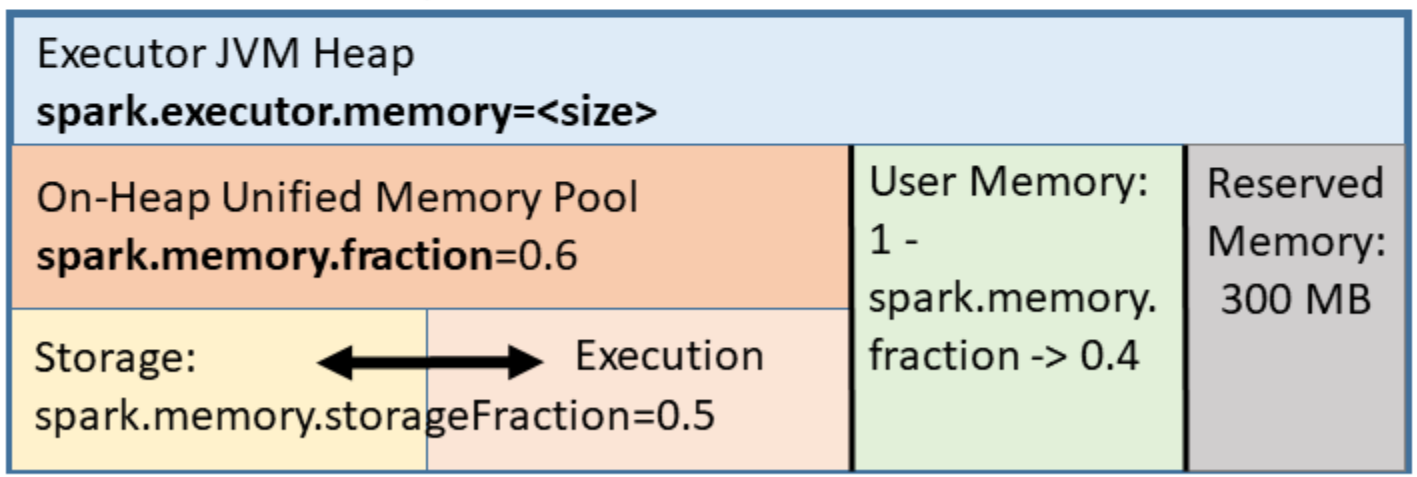
Versions 3.0 and 2.4

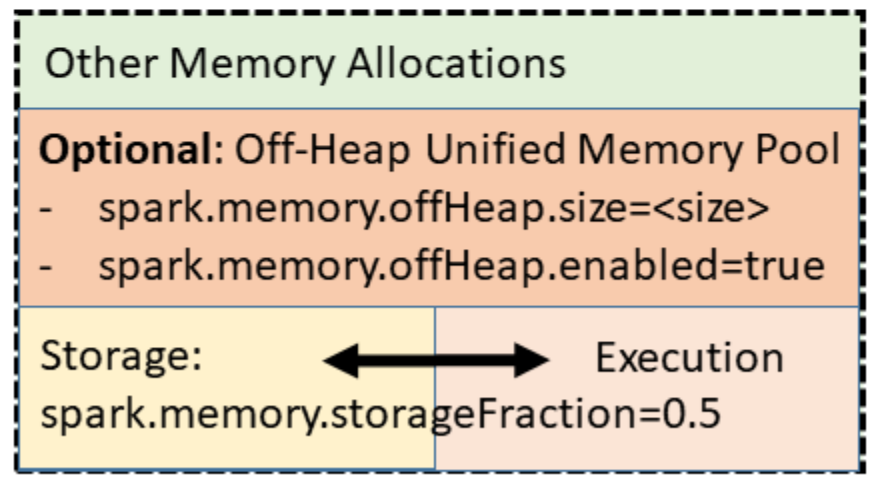


Overhead memory: for OS, filesystem cache, redundancy, and other native memory allocations:
- Default: max(0.1 * spark.executor.memory), 384MB)
- Configuration: spark.executor.memoryOverhead=<size>
- K8S tunable: spark.kubernetes.memoryOverheadFactor

Optional extra memory for off-heap unified memory pool:
- spark.memory.offHeap.size=<size>
- spark.memory.offHeap.enabled=true
- K8S: see SPARK-32661

Optional extra memory for PySpark allocations:
- YARN: spark.executor.pyspark.memory
- K8S: set also spark.kubernetes.resource.type="python"





Memory Monitoring Graphs in the Dashboard

Memory metrics as time series

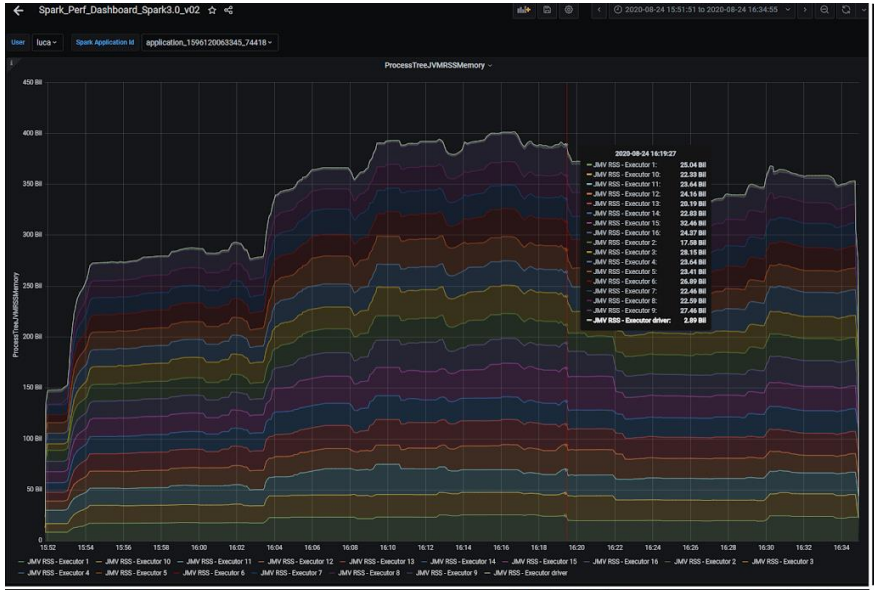
Dashboard:

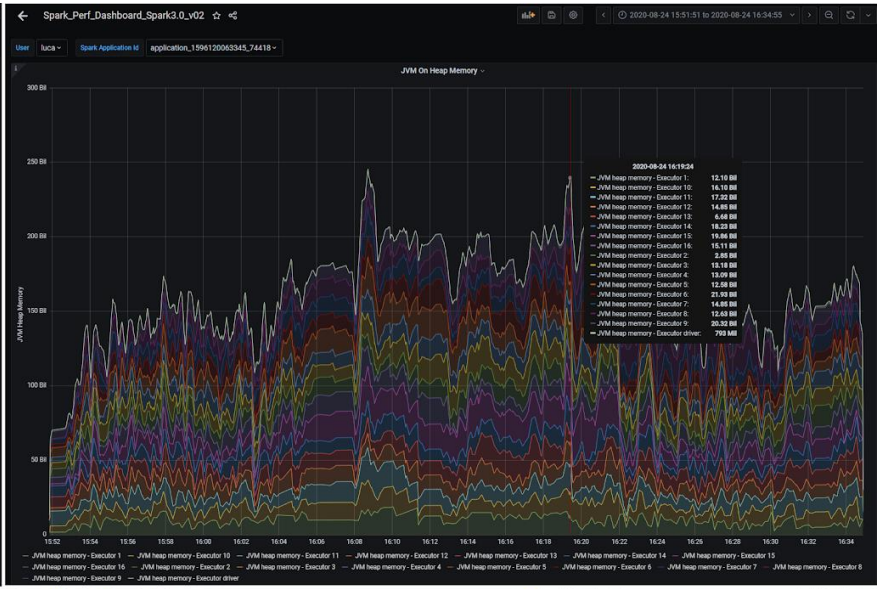
Visualize memory usage over time

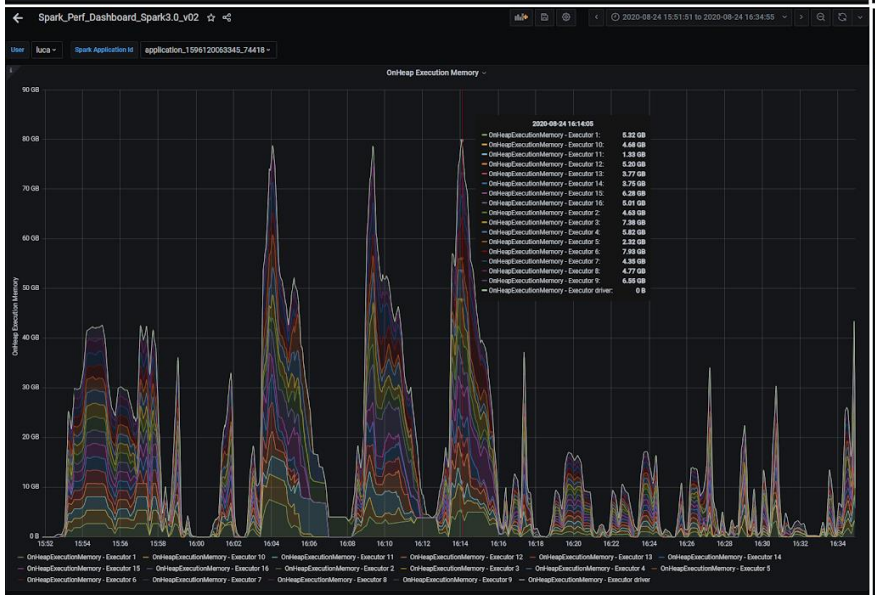
Details of on-heap, unified memory, etc

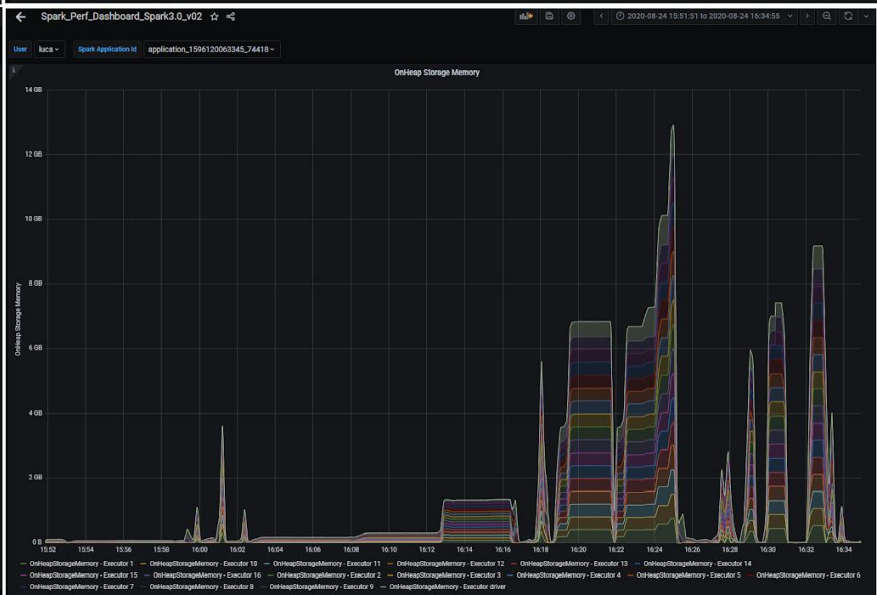
Compare with other metrics

Correlate with jobs/sql start time









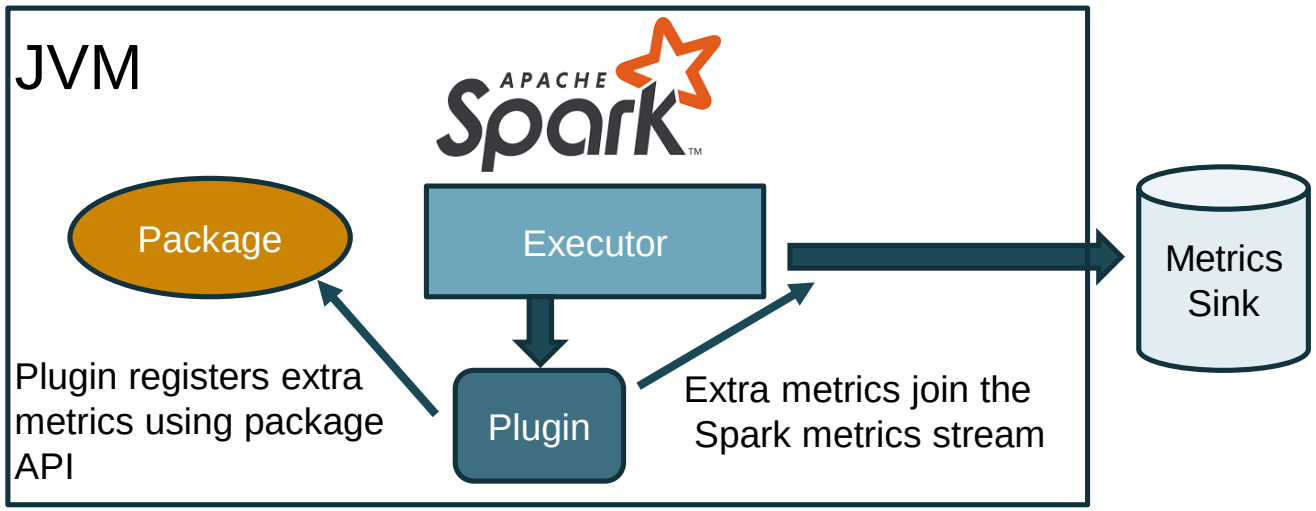
Custom Monitoring with Spark Plugins

- Spark 3.0 Plugins
  - Plugins are executed at the startup of executors and the driver
  - Plugins allow to extend Spark metrics with custom code and instrumentation

- Examples of monitoring enhancements with plugins
  - Cloud Storage monitoring (S3A, GS, WASBS, OCI, CERN's ROOT, etc)
  - Improved HDFS monitoring
  - OS metrics: cgroup metrics for Spark on Kubernetes
  - Custom application metrics

Plugins Extend Spark Metrics

- Custom metrics for instrumentation or debugging
- Metrics from libraries/packages on top of Apache Spark
- Spark Plugins provide:
  - API for integrating custom instrumentation with the rest of the Spark monitoring



Spark Plugins API

- Code snippets

```
import com.codahale.metrics.{Gauge, MetricRegistry}
import org.apache.spark.api.plugin.{DriverPlugin, ExecutorPlugin, PluginContext, SparkPlugin}
import org.apache.spark.SparkContext

class DemoMetricsPlugin extends SparkPlugin {

// Return the plugin's executor-side component.
override def executorPlugin(): ExecutorPlugin = {
  new ExecutorPlugin {
    override def init(myContext:PluginContext, extraConf:JMap[String, String]): Unit  = {
      // Gauge for testing
      val metricRegistry = myContext.metricRegistry
      metricRegistry.register(MetricRegistry.name( name = "ExecutorTest42"), new Gauge[Int] {
        override def getValue: Int = 42
        })
    }
  }
}
```

Spark 3.0 Plugin API

This is how you hook to the Spark Metrics instrumentation

Getting Started with Spark Plugins

- From https://github.com/cerndb/SparkPlugins
- Plugins for demo + plugins for Cloud I/O and OS monitoring
  - RunOSCommandPlugin runs an OS command at executor startup
  - DemoMetricsPlugin show how to integrate with Spark Metrics

```
bin/spark-shell –master yarn
--packages ch.cern.sparkmeasure:spark-plugins_2.12:0.1 \
--conf spark.plugins=ch.cern.RunOSCommandPlugin,\
ch.cern.DemoMetricsPlugin
```

Measuring OS and Container Metrics

- Example of how to measure OS metrics from cgroup instrumentation
- Useful for Spark on K8S
  - Brings together OS metrics with other Spark-workload metrics
  - By default Apache Spark instruments only CPU usage

```
bin/spark-shell --master k8s://https://<K8S URL>:6443 \
--packages ch.cern.sparkmeasure:spark-plugins_2.12:0.1 \
--conf spark.kubernetes.container.image=<registry>/spark:v310-SNAP \
--conf spark.plugins=ch.cern.CgroupMetrics \
...
```

Cgroups Metrics – Use for Spark on K8S

- Metrics implemented (gauges), with prefix ch.cern.CgroupMetrics:
  - CPUTimeNanosec: CPU time used by the processes in the cgroup
    - this includes CPU used in Python
  - MemoryRss: number of bytes of anonymous and swap cache memory.
  - MemorySwap: number of bytes of swap usage.
  - MemoryCache: number of bytes of page cache memory.
  - NetworkBytesIn: network traffic inbound.
  - NetworkBytesOut: network traffic outbound.

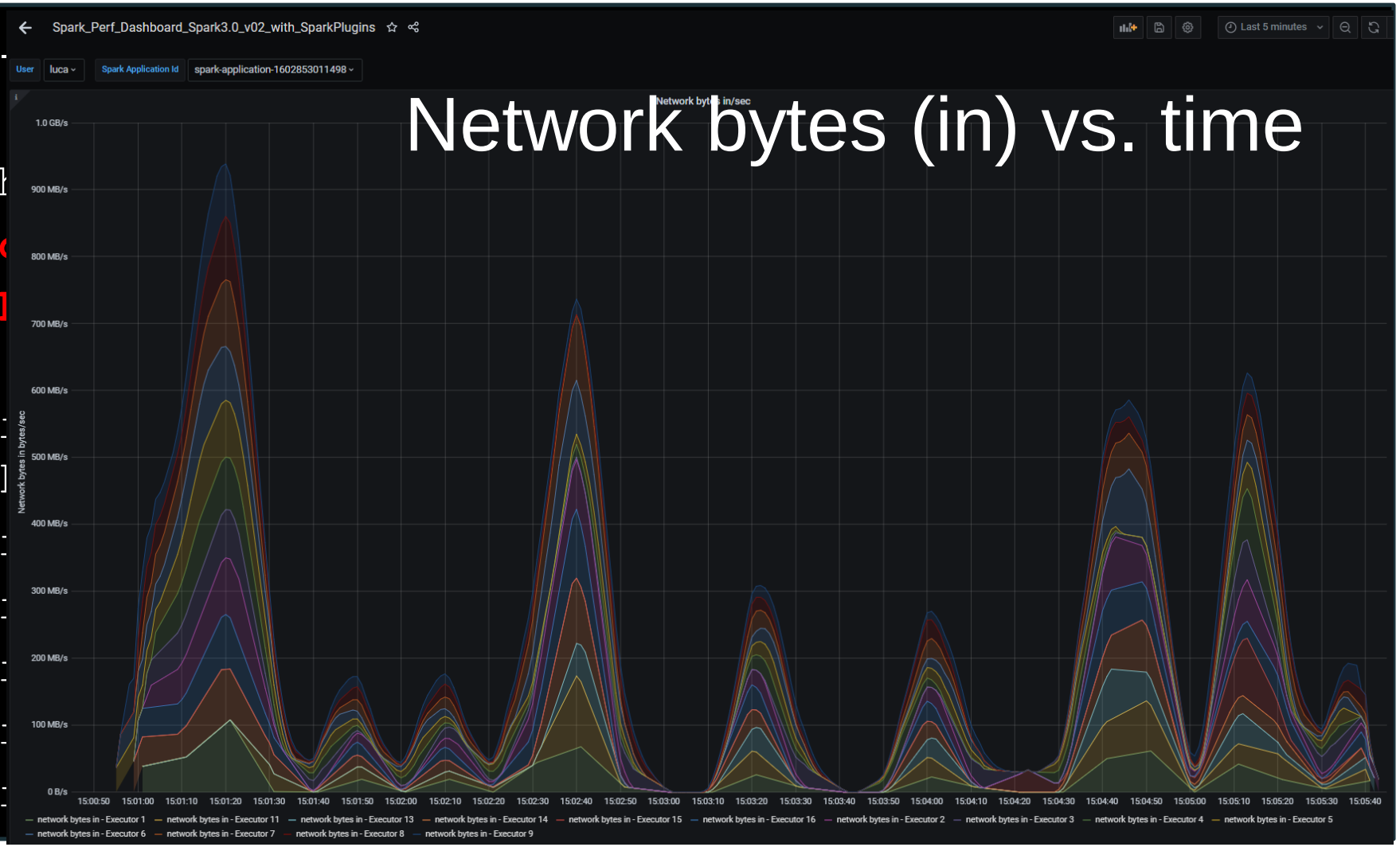
Example: Dashboard for Spark on K8S

```
docker run --network=host -d lucacanali/spark-dashboard:v01
bin/spark-shell --master k8s://https:...\
--packages ch.cern.sparkmeasure:spark-plugins_2.12:0.1 \
--conf spark.plugins=ch.cern.CgroupMetrics \
--conf
"spark.metrics.conf.*.sink.graphite.class"="org.apache.spark.
metrics.sink.GraphiteSink" \
--conf "spark.metrics.conf.*.sink.graphite.host"=`hostname` \
--conf "spark.metrics.conf.*.sink.graphite.port"=2003 \
--conf "spark.metrics.conf.*.sink.graphite.period"=10 \
--conf "spark.metrics.conf.*.sink.graphite.unit"=seconds \
--conf "spark.metrics.conf.*.sink.graphite.prefix"="luca"
```

Network bytes (in) vs. time

Example: Dashboard for Spark on K8S

```
docker run -

bin/spark-sh
--packages
--conf spark
--conf
"spark.metri
metrics.sinl
--conf "spa
--conf "spa
--conf "spa
--conf "spa
--conf "spa
```



Measuring S3A and other Cloud Filesystems

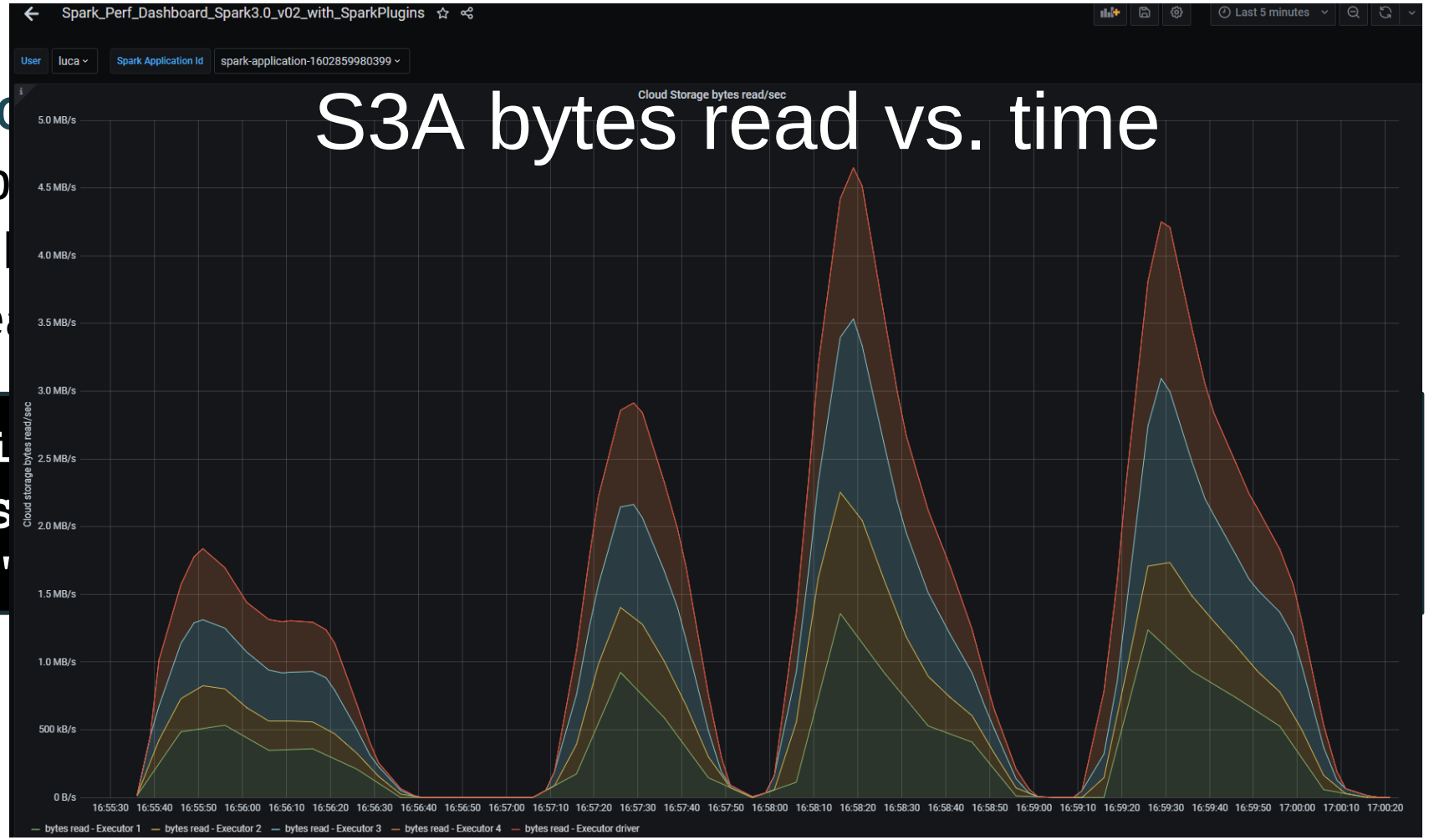
- Example of how to measure S3A throughput metrics
  - Note: Apache Spark instruments only HDFS and local filesystem
  - Plugins uses HDFS client API for Hadoop Compatible filesystems
  - Metrics: bytesRead, bytesWritten, readOps, writeOps

```
--conf spark.plugins=ch.cern.CloudFSMetrics
--conf spark.cernSparkPlugin.cloudFsName=<name of the filesystem>
(example: "s3a", "gs", "wasbs", "oci", "root", etc.)
```

Measuring S3A and other Cloud Filesystems

- Example of how to
  - Note: Apache Sp
  - Plugins uses HDI
  - Metrics: bytesRe

```
--conf spark.plugi
--conf spark.cernS
(example: "s3a", "
```



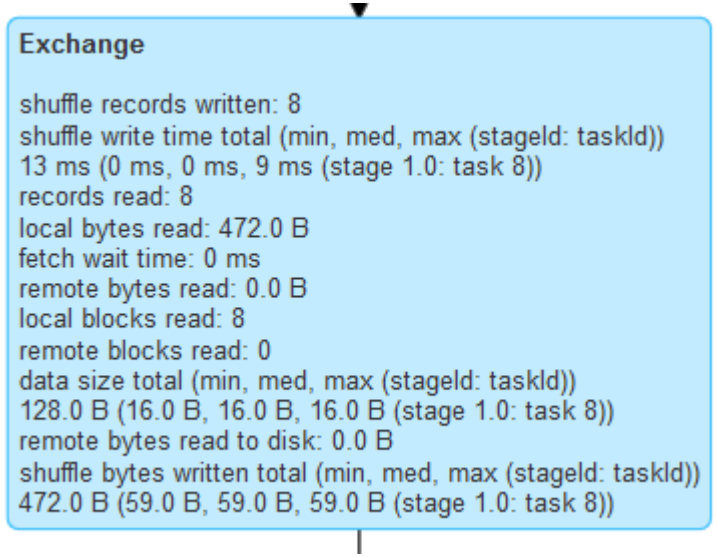
Experimental: Measuring I/O Time

- Plugins used to measure I/O read time with HDFS and S3A
  - Use for advanced troubleshooting
- Also example of plugins used to instrument custom libraries
  - Plugin: **--conf spark.plugins=ch.cern.experimental.S3ATimeInstrumentation**
  - Custom S3A jar with time instrumentation: https://github.com/LucaCanali/hadoop/tree/s3aAndHDFSTimeInstrumentation
  - Metrics: S3AReadTimeMuSec, S3ASeekTimeMuSec, S3AGetObjectMetadataMuSec

SQL Monitoring Improvements in Spark 3.0

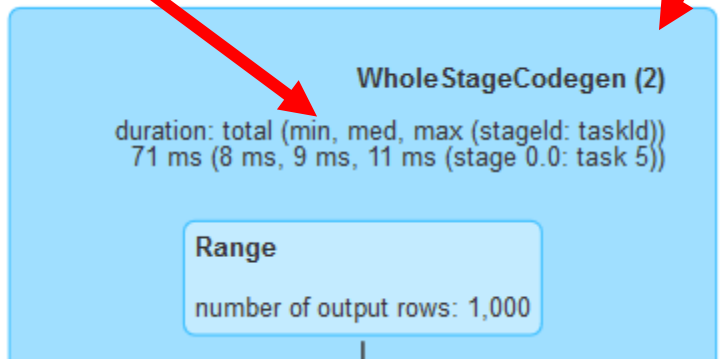
- Improved SQL Metrics and SQL tab execution plan visualization
  - Improved SQL metrics instrumentation
  - SQL metrics documented at https://spark.apache.org/docs/latest/web-ui.html#sql-metrics
  - Improved execution plan visualization and available metrics

Improved metrics list for shuffle operations



Improved stats visualization + info on max value

Added Codegen Id



More Monitoring Improvements in Spark 3.0

- Spark Streaming tab 

  - Metrics and graphs
  - New Structured Streaming UI (SPARK-29543)
- Experimental support for Prometheus
  - REST API: /metrics/executors/Prometheus conditional to spark.ui.prometheus.enabled=true
- Improved documentation
  - New: Web UI doc https://spark.apache.org/docs/latest/web-ui.html
  - Monitoring doc: https://spark.apache.org/docs/latest/monitoring.html

Outlook and Community

Community Effort
What You Can Do To Improve Spark Monitoring

- (developers) Tooling: Improving Spark Metrics and instrumentation
  - Sink for InfluxDB and Prometheus
  - Further instrument Spark core and ecosystem (ex: I/O time, Python UDF)
  - Develop and share plugins: instrument libraries, OS metrics (GPUs), etc
- (product experts) Methods: root-cause performance analysis
  - Use metrics and graphs for troubleshooting and root-cause analysis
  - Adopt tools and methods to your platform/users
- (innovators) Holy Grail of Monitoring:
  - Building automated (AI) systems that can perform root-cause analysis and autotune data/database systems

Improvements Expected in Spark 3.1 and WIP

- [SPARK-27142] New SQL REST API
- [SPARK-32119] Plugins can be distributed with --jars and -packages on YARN, this adds support for K8S and Standalone
- [SPARK-33088] Enhance Executor Plugin API to include callbacks on task start and end events
- [SPARK-23431] Expose stage level peak executor metrics via REST API
- [SPARK-30985] Support propagating SPARK_CONF_DIR files to driver and executor pods
- [SPARK-31711] Executor metrics in local mode
- WIP [SPARK-30306] Python execution time instrumentation
- WIP on Hadoop (targeting Hadoop 3.4)
  - [HADOOP-16830] Add public IOStatistics API

Conclusions

- Monitoring and instrumentation improvements:
  - One more reason to upgrade to Apache Spark 3.0
- Memory monitoring with Executor Metrics in Spark 3.0
  - Help troubleshooting and preventing OOM
- Spark Plugin API
  - Use to measure Cloud FileSystems I/O, OS and container metrics
  - Use to build your custom application metrics
  - Build and share!
- Web UI, streaming and Spark SQL monitoring also improved

Acknowledgements and Links

- Thanks to colleagues at CERN, Hadoop and Spark service
- Thanks to Apache Spark committers and community
  - For their help with JIRAs and PRs: SPARK-26928, SPARK-27189, SPARK-28091, SPARK-29654, SPARK-30041, SPARK-30775

- Links:
  - Executor Metrics and memory monitoring: SPARK-23429 and SPARK-27189
  - Spark Plugins: SPARK-29397, SPARK-28091
  - https://db-blog.web.cern.ch/blog/luca-canali/2020-08-spark3-memory-monitoring
  - https://github.com/LucaCanali/sparkMeasure
  - https://github.com/cerndb/spark-dashboard
  - https://github.com/cerndb/SparkPlugins

Feedback

Your feedback is important to us.

Don’t forget to rate and review the sessions.