The ATLAS EventIndex and its evolution based on Apache Kudu storage

Evgeny Alexandrov[1], Igor Alexandrov[1], Zbigniew Baranowski[2], Dario Barberis[3], Luca Canali[2], Gancho Dimitrov[4], Alvaro Fernandez Casani[5], Elizabeth Gallas[6], Carlos Garcia Montoro[5], Santiago Gonzalez de la Hoz[5], Julius Hrivnac[7], Aleksandr Iakovlev[1], Andrei Kazymov[1], Mikhail Mineev[1], Fedor Prokoshin[8], Grigori Rybkin[7], Javier Sanchez[5], José Salt Cairols[5], Petya Vasileva[4], Miguel Villaplana Perez[9]

[1] JINR Dubna, [2] CERN-IT, [3] Univ./INFN Genova, [4] CERN-ATLAS, [5] IFIC Valencia, [6] Univ. Oxford, [7] LAL Orsay, [8] CCTVal/UTFSM Valparaíso, [9] Univ./INFN Milano

On behalf of the ATLAS Collaboration

Grid2018 @ JINR (Dubna)
10-14 September 2018

Indexing Events

- The EventIndex is a system designed to be a complete catalog of ATLAS events, real and simulated data
  - The Event is the basic unit of ATLAS data
- Each event contains the measurement of a single bunch collision
  - Signals from the detector
  - Reconstructed particles with their parameters
  - Trigger decisions
- Uniquely identified by the run and event number
- Event information is stored in many instances
  - Have different formats to fit analysis needs
  - Spread among the hundreds of GRID sites

ATLAS Data and Datasets

- ATLAS event data are written in files that are organized in datasets
- Real data datasets formats depend on the processing stage:
  - Detector data are first written in the RAW format
  - Physics (AOD) datasets are produced after reconstruction
  - Derived (DAOD) datasets for use in the specific analyses
- Simulated datasets are produced on the Grid:
  - EVNT datasets contain generated particles
  - HITS datasets contain simulated energy deposits in the detectors
  - RDO datasets contain simulated detector readout signals
- There are various versions of the datasets originating from the same detector events:
  - The same events may be processed multiple times with different reconstruction settings or software version (real events are reprocessed with newer versions roughly yearly)

EventIndex record contents

Each event record contains 3 blocks of information:

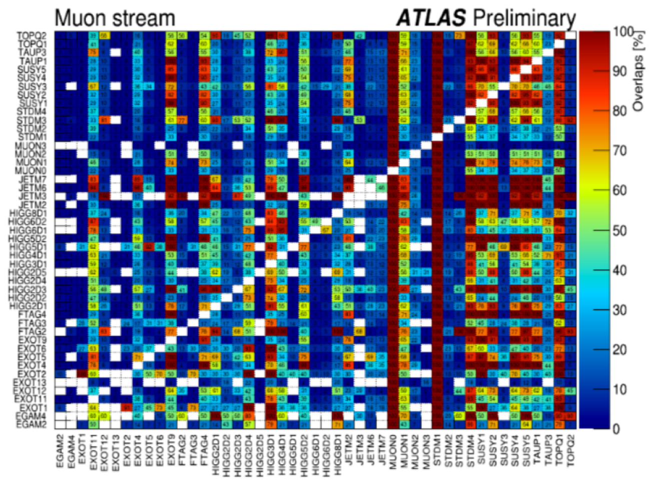
- **Event identifiers**
  - **Run and event number**
  - **Trigger stream**
  - **Luminosity block**
  - **Bunch Crossing ID (BCID)**
- **Trigger decisions**
  - **Trigger masks for each trigger level**
  - **Decoded trigger chains (trigger condition passed)**
- **References' to every event at each processing stage:**
  - **These are unique pointers to that event on the grid, enabling user 'event picking' jobs to retrieve specific events of interest**

Use cases

1) Event picking: users able to select single events depending on constraints. Order of hundreds of concurrent users, with requests ranging from 1 event (common case) to 30k events (occasional).

2) Production consistency checks

- **Duplicate event checkings**: events with same Id appearing in same or different files/datasets.
- **Overlap detection** in derivation framework: construct the overlap matrix identifying common events across the different files.

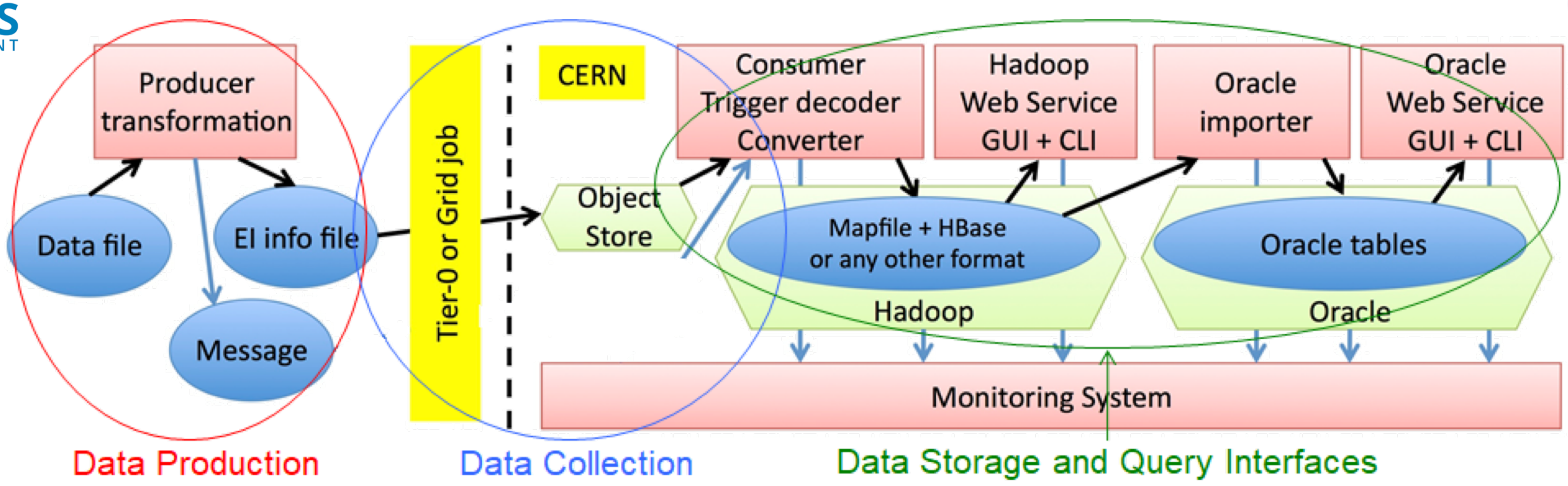


3) Trigger checks and event skimming: Count or give an event list based on trigger selection.

- **Trigger Overlap detection:** number of events in a real data Run/Stream satisfying trigger X which also satisfies trigger Y.

Requirement: Storing and accessing thousands of files and millions of events in reasonable time.

EventIndex Architecture



Partitioned architecture, following the data flow

Data production

- Extract event metadata from files produced at Tier-0 or on the Grid

Data collection

- Transfer EI information from jobs to the central servers at CERN

Data storage

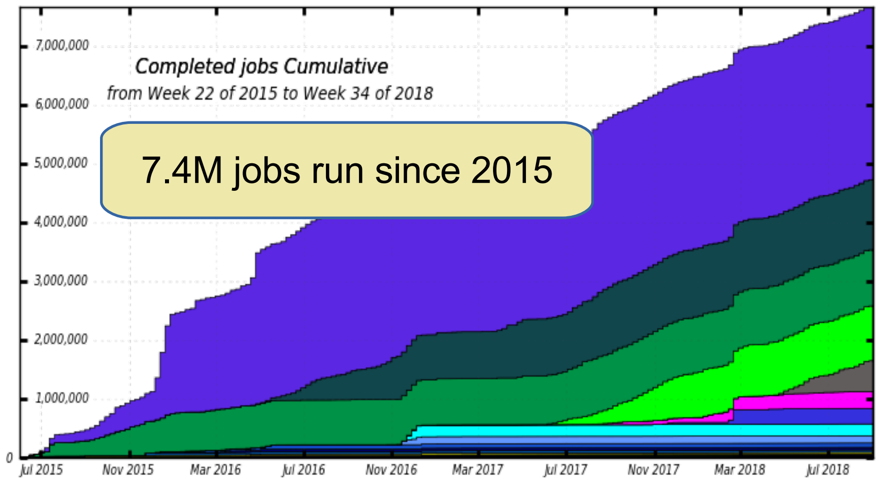
- Provide permanent storage for EventIndex data.
- Full info in Hadoop; reduced info (only real data, no trigger) in Oracle
- Fast access for the most common queries, reasonable time response for complex queries

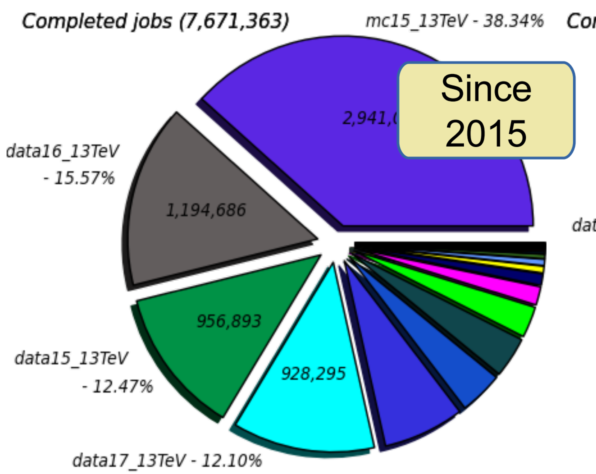
Monitoring

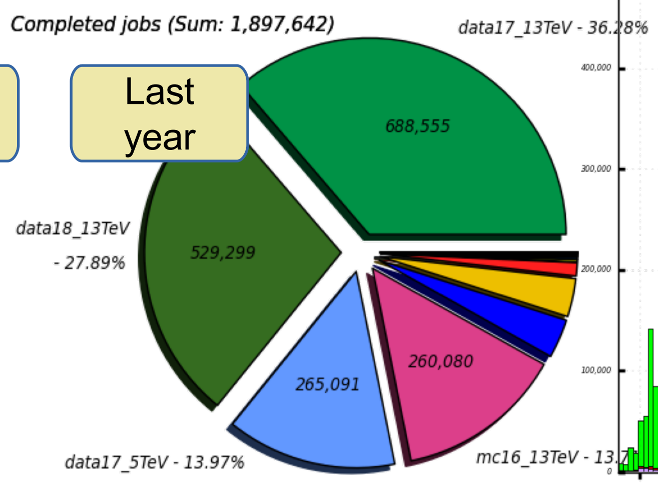
- Keep track of the health of servers and the data flow

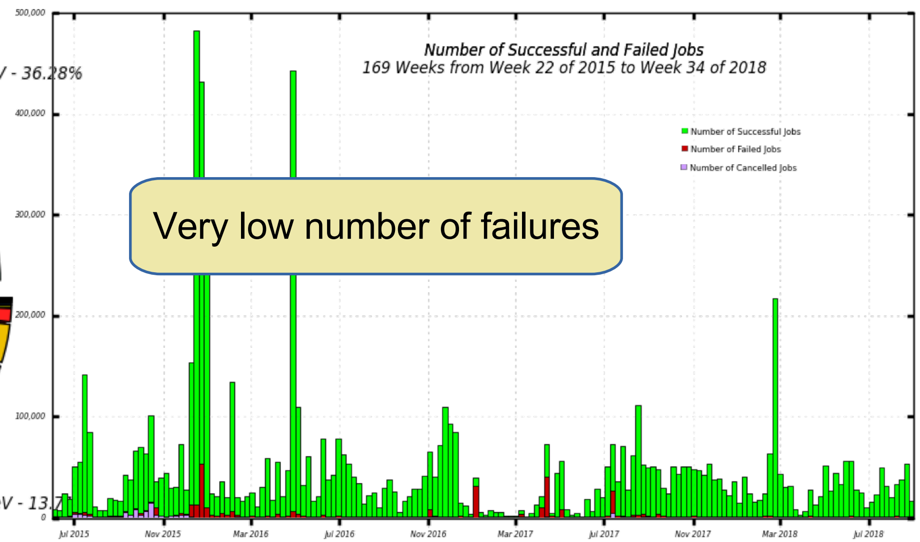
Data Production

- **Tier-0 jobs index merged physics AODs, collecting also references to RAW and (if existing) ESD files**
- **Similarly, Grid jobs collect info from EVNT and AOD datasets as soon as they are produced and complete**
  - **Other data formats (HITS, DAOD etc.) are indexed on demand**
  - **Continuous operation since spring 2015**
- **System now in routine operation**
  - **Very low number of failures:**
    - **Site problems (fixed by retries)**
    - **Corrupted files found occasionally**



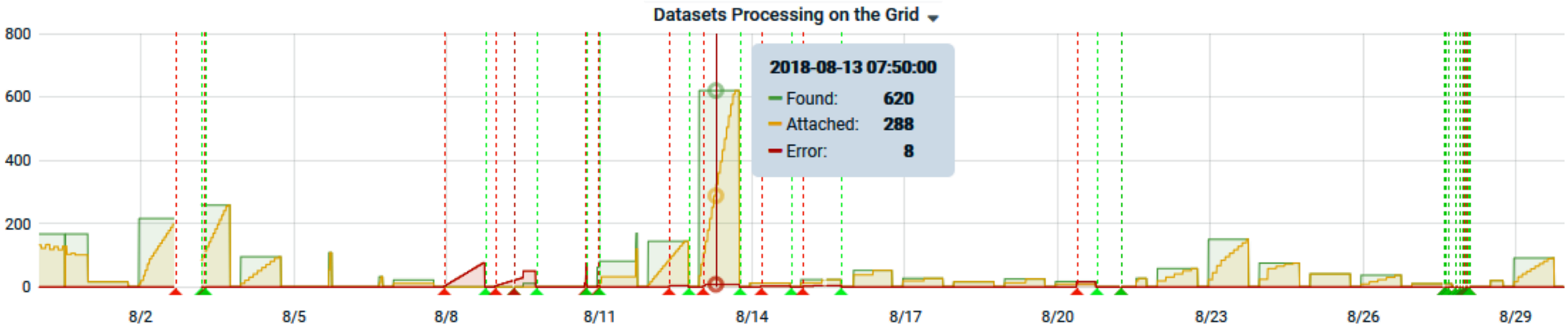






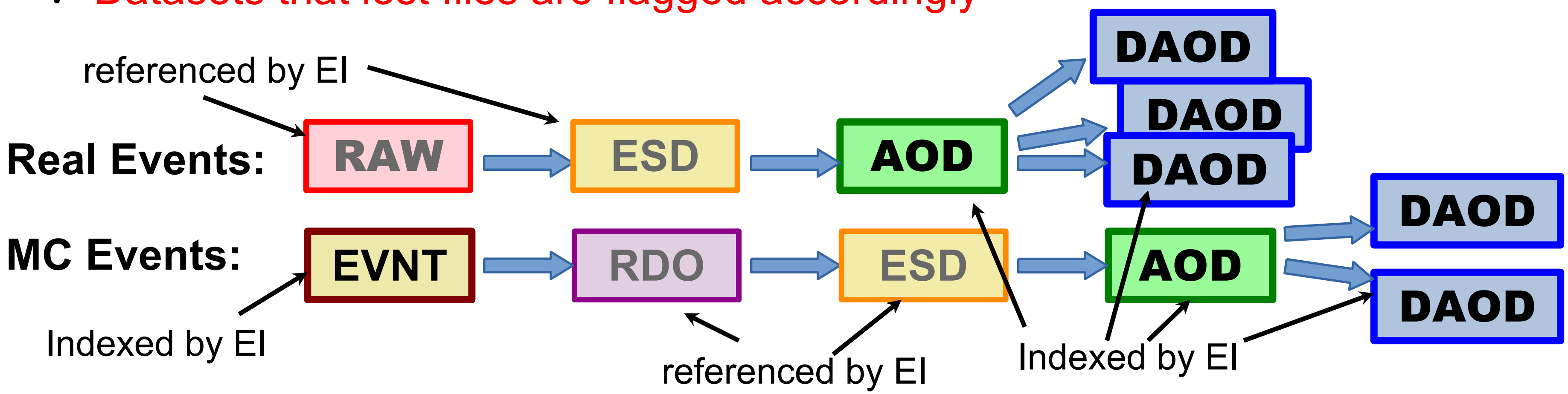
Data Production on the Grid

- We use information from AMI (the ATLAS Metadata Interface) to find datasets that have just been produced
  - If a dataset was completed recently, AMI sets a special field in its database
- A script runs daily and generates list of datasets checking this filed
- Datasets found are then checked for being “good” with AMI and ATLAS data management system (Rucio)
  - “Good” dataset need to be valid and complete, have some events in it
- After the cleanup, datasets are attached to the special “technical containers” in Rucio.
  - The ATLAS Production System picks up these datasets and runs EventIndex production jobs on them.



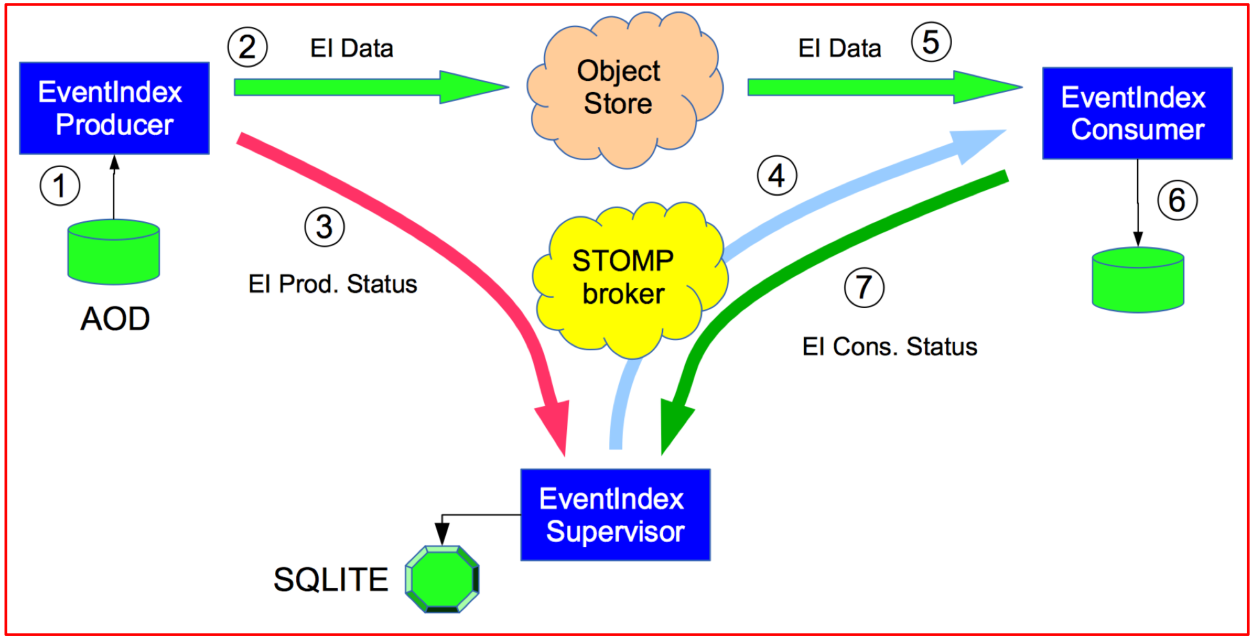
Indexed Data

- **Available data:**
  - All Tier-0 production, data09 to data18, physics streams (and few others for testing)
    - From AOD datasets; references to RAW and ESD also available.
  - Valid Grid reprocessings of Run2 physics streams
    - From AOD datasets; references to RAW and ESD also available.
  - All valid Run2 MC datasets in EVNT and AOD format
    - References to RDO and ESD also available if intermediate datasets were created.
  - Additional data (DAOD) on request of physics groups
- **All data are continuously cross-checked with AMI (metadata catalog)**
  - Deleted datasets are also removed from the EventIndex
  - Datasets that lost files are flagged accordingly



Data Collection

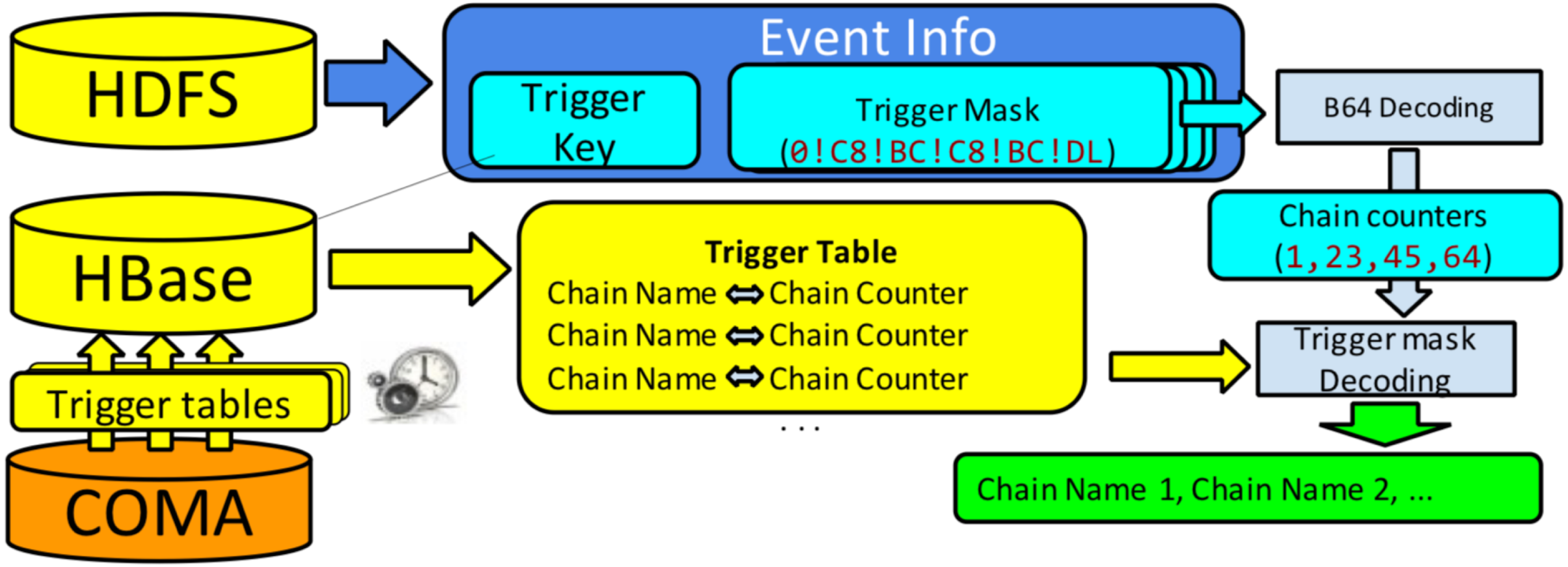
2015 - mid 2017: Pure Messaging Based architecture (ActiveMQ brokers / Stomp protocol). Json data encoding.
Production peaks showed bottlenecks on messaging brokers
mid 2017 onwards: ObjectStore (CEPH / S3 interface) as intermediary storage. Google protobuf data encoding (compressed)



Producer: Athena Python transformation, running at Tier-0 and grid-sites. Indexes AOD data and produces an **EventIndex file**, stored in **ObjectStore**
Supervisor: Controls all the process, receives processing information and validates data by dataset. Signals valid unique data for ingestion to Consumers. Operated with a web interface
Consumers: Retrieves ObjectStore data, groups by dataset and ingest it into **HDFS** J**(Hadoop distributed Filesystem)**

Trigger Decoding

- **Trigger decisions are stored in data files as trigger bit masks.**
  - **One or more bits of the mask are set corresponding to the triggers satisfied by the event**
  - **The mapping of bits to triggers for each set of events is known from the trigger database**
- **Trigger masks from event records are decoded before storing in Hadoop**
  - **Chain counters are converted to chain names using trigger tables replicated from COMA (the conditions metadata database) for real data**
  - **Decoding of trigger information for the MC needs information from the MC trigger database (TRIGGERDBMC)**



Much more information in M. Mineev's talk this afternoon

Hadoop storage and queries

We use Hadoop as the baseline storage technology

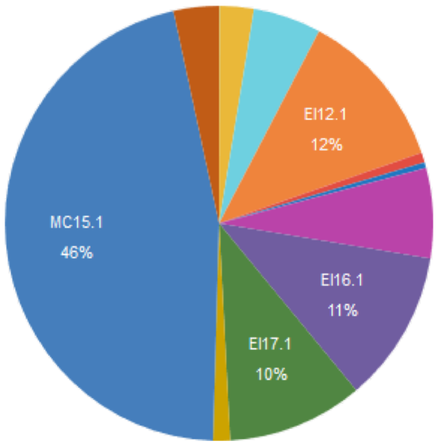
- It can store large numbers (10s of billions) of simply-structured records and search/retrieve them in reasonable times

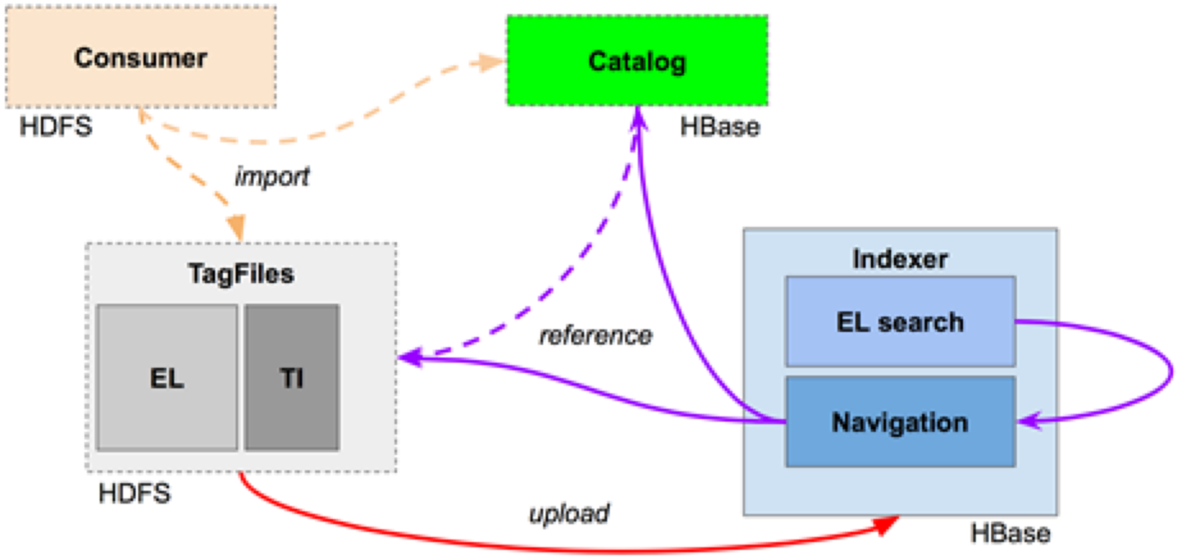
Hadoop compressed "MapFiles" (indexed sequential files) are used as data format

- **One MapFile per dataset**
- **Internal catalogue in HBase** (the Hadoop database) keeps track of what is where and dataset-level metadata (status flags)
- Event Lookup index in HBase

Data volumes:

- **Real 2009-2018: 21 TB**
- **MC 2015-2018: 5 TB**
- **Other: 150 TB**

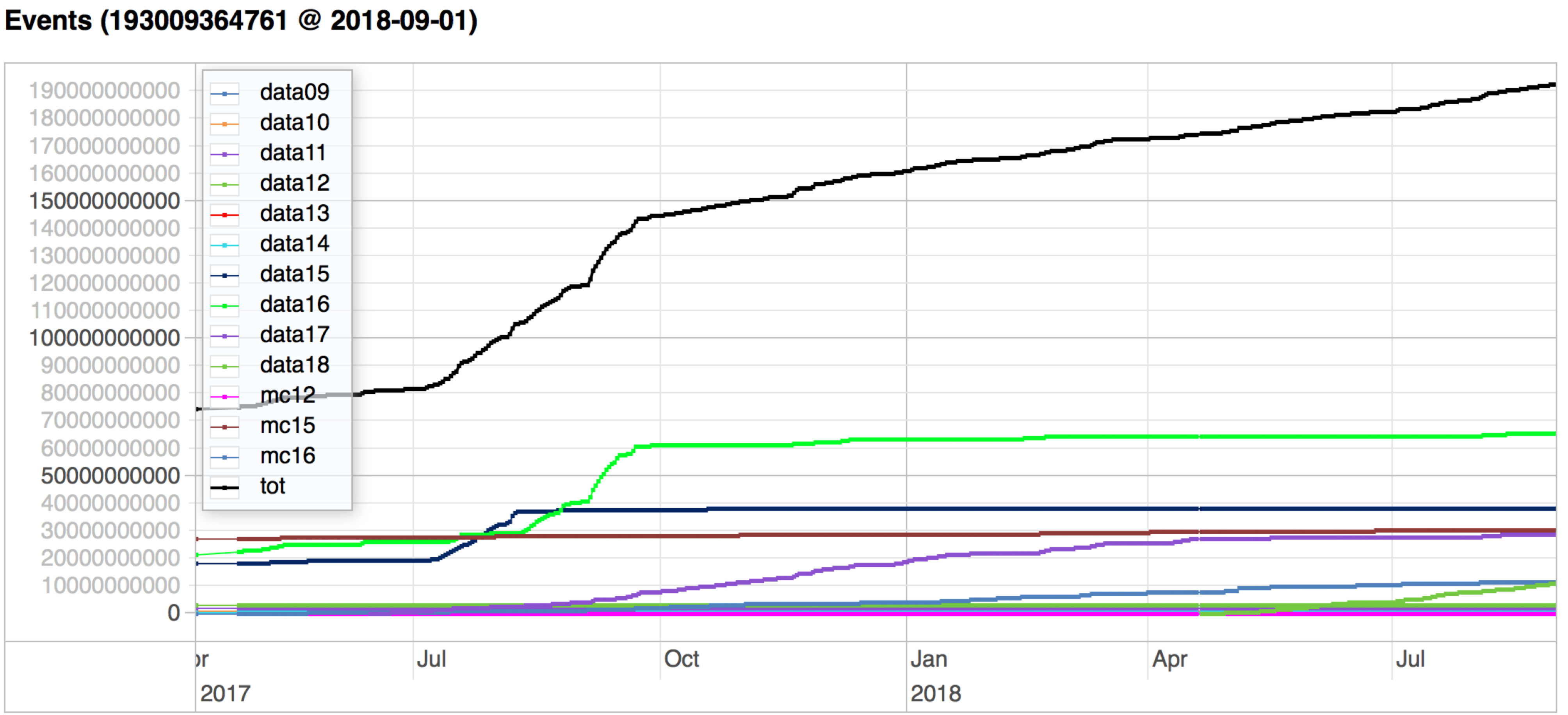




Data import in Hadoop

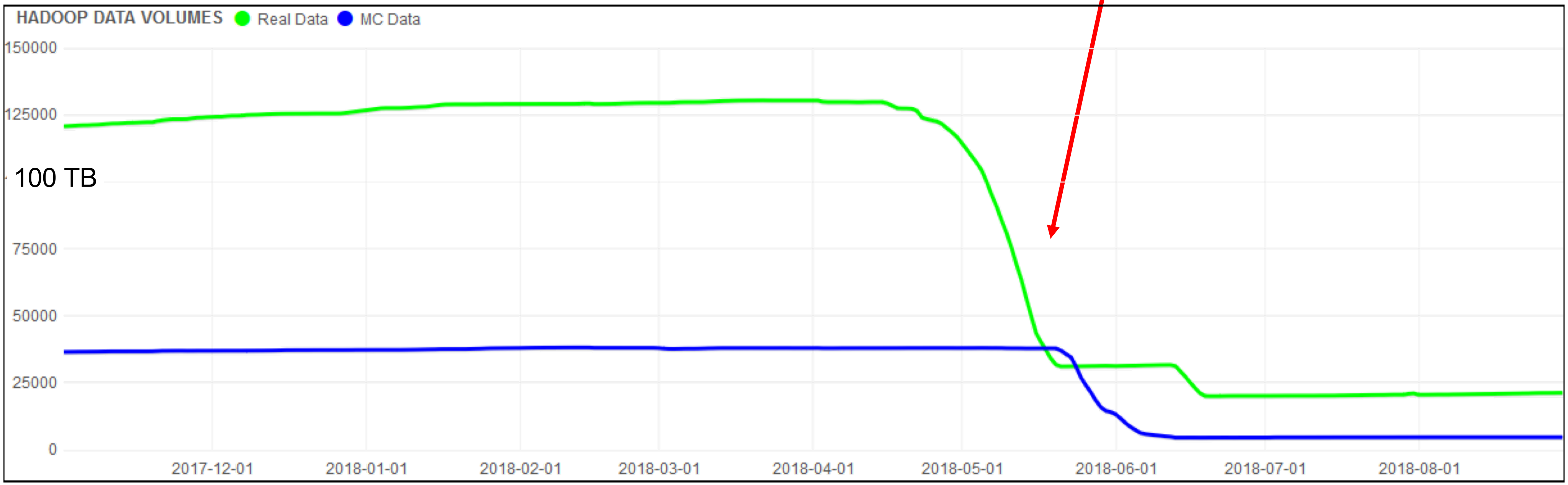
- **Data import in Hadoop is easily keeping up with the current production rates**

- **By now imported 193 billion event records from 160k datasets**

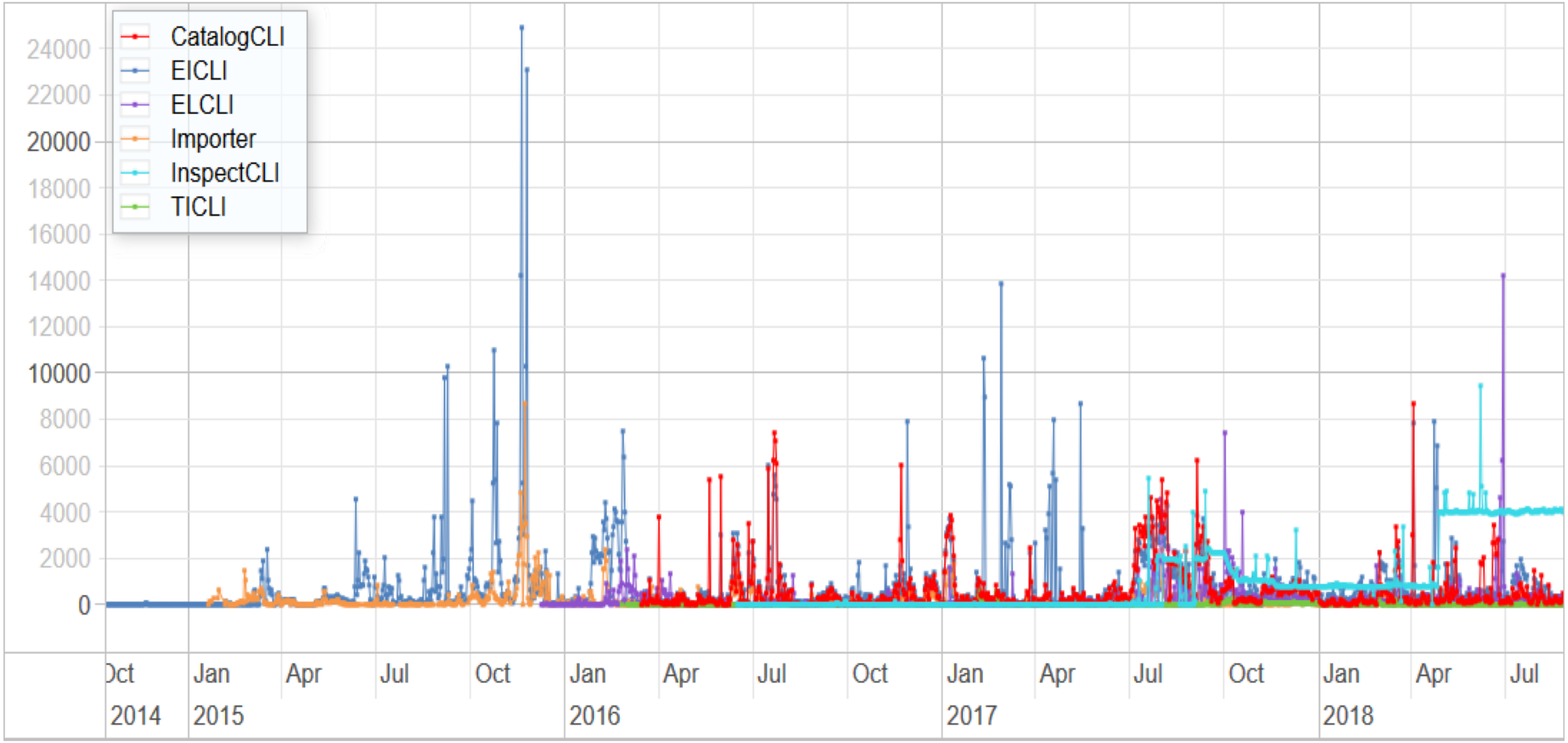


Data volumes in Hadoop

- **The size of Tag/Map Files is growing and we ran out of space earlier this year.**
- **A BLOCK compression was introduced to reduce data files size**
  - **With BLOCK compression groups or blocks of keys and records are compressed together**
- **Using BLOCK compression for Tag/Map Files we had ~10 times space savings**



Hadoop daily access statistics



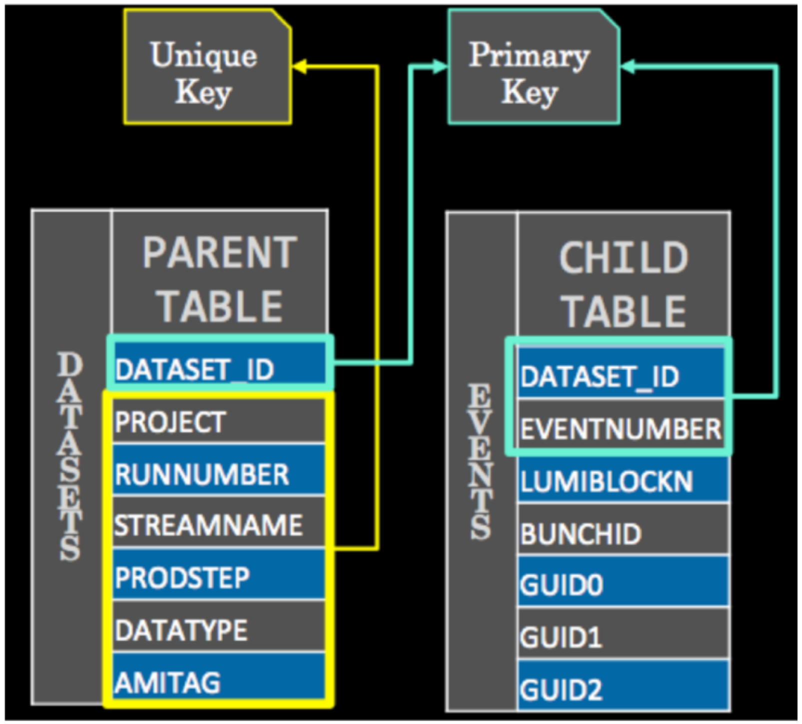
- **CatalogCLI, EICLI, ELCLI and TICLI are user actions**
- **Importer and InspectCLI are system actions**

EventIndex Oracle storage

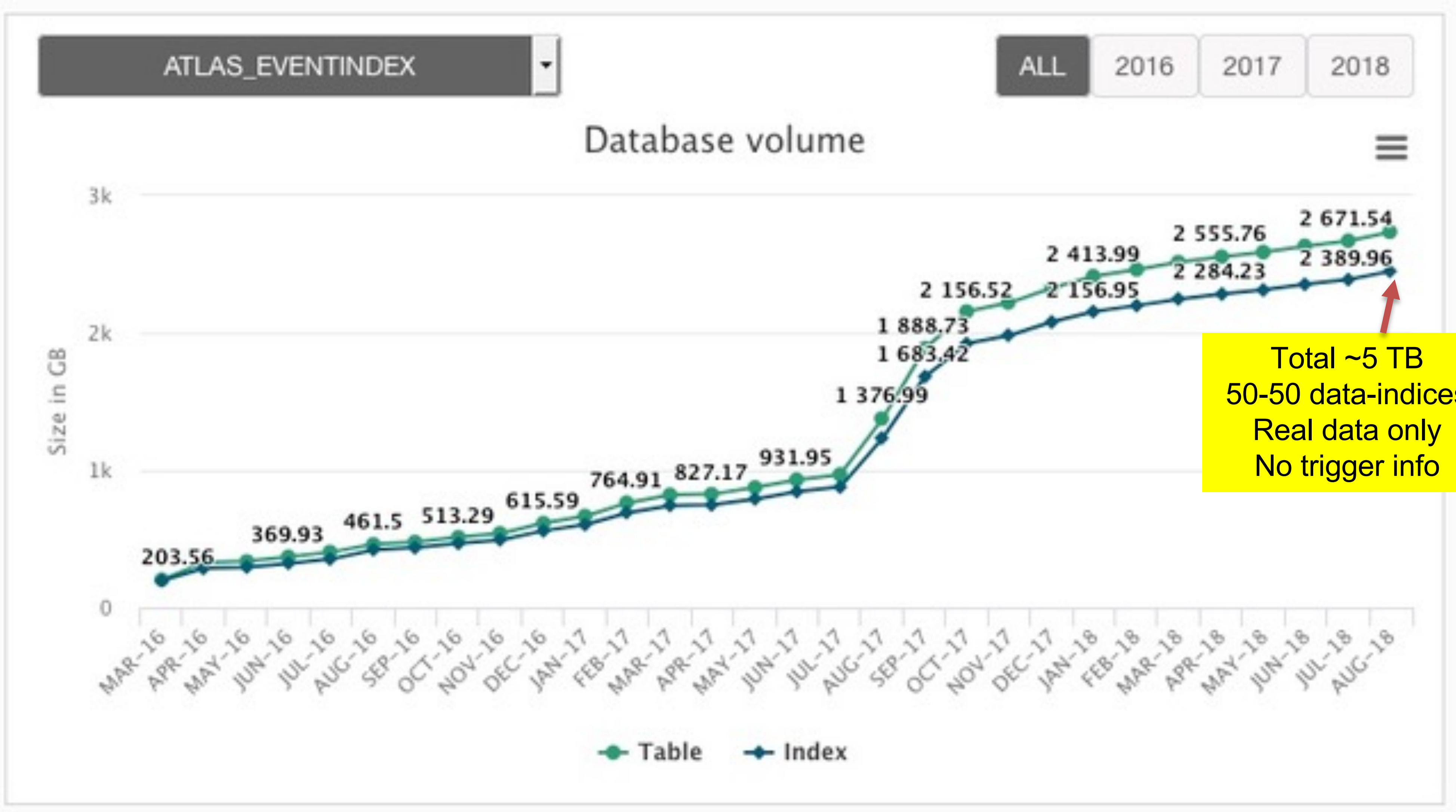
- Simple schema with dataset and event tables
  - **Exploiting the relational features of Oracle**
- Filled with all real data, only event identification and pointers to event locations
  - **Optimized for event picking**
  - **Very good performance also for event counting by attributes (LumiBlock and bunchID)**

- Connection to the RunQuery and AMI databases to check dataset processing completeness and to detect duplicates
- Easy calculation of dataset overlaps

64k Datasets (150 Billion event records)



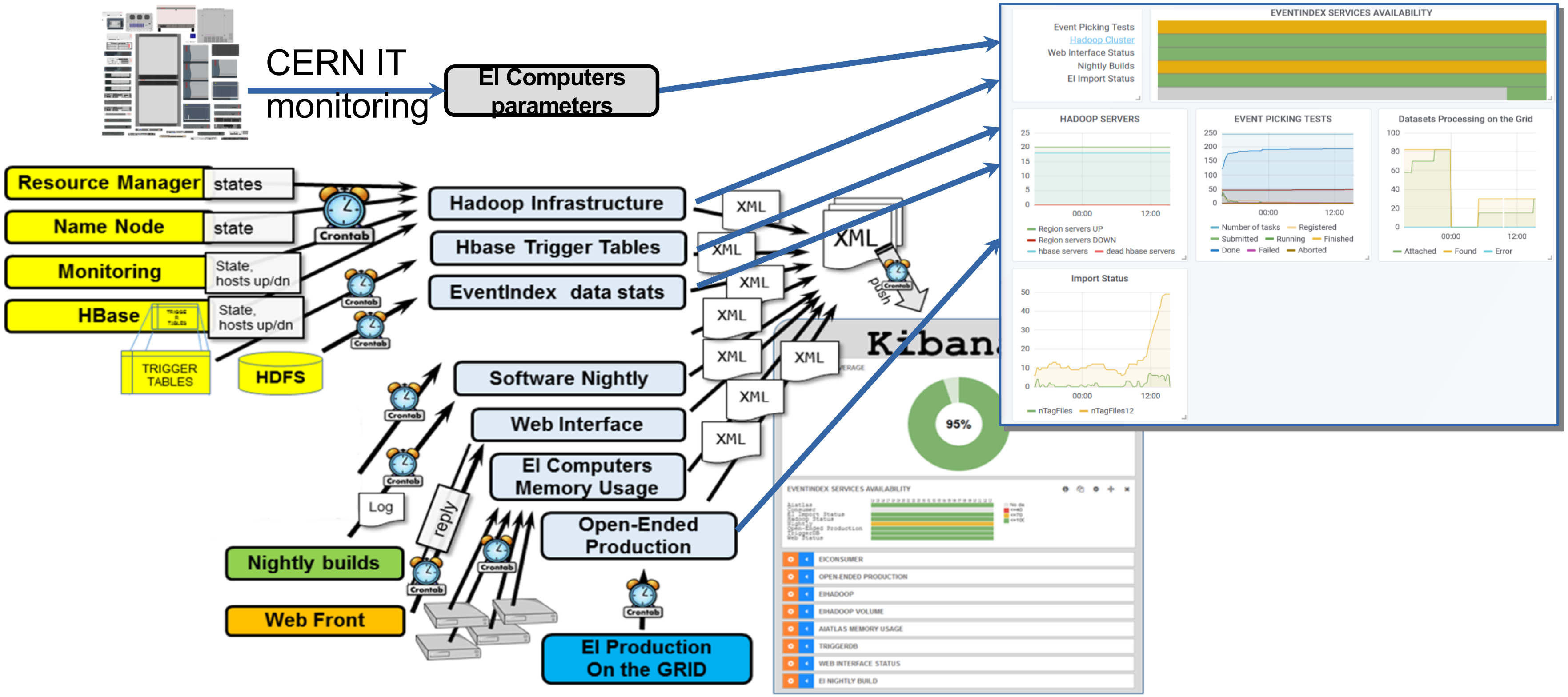
Data in Oracle



Total ~5 TB
50-50 data-indices
Real data only
No trigger info

Monitoring

- **Uses various ways of data collection and processing (log and web pages parsing, statistic messages, HDFS browsing)**
  - **Information collected by acron jobs (~15k values daily)**
  - **Information is organized into xml files and then pushed to Kibana**
  - **Information is being pushed to Grafana via REST interface**



Much more information in the talk by E. Alexandrov and A. Kazymov this afternoon

EventIndex evolution using Kudu

- Current EventIndex was designed in 2012-2013 using best BigData technology available at that time (Hadoop), implemented in 2014 using MapFiles and HBase, in operation since 2015 with satisfactory results
- Use cases extended in the meantime from event picking and production completeness checks to trigger overlap studies, duplicate event detection and derivation streams (offline triggers) overlaps
- Fast data querying based on traditional relational database (Oracle) involving a subset of information for real events only no longer sufficient
- Also event rate increased steadily throughout Run 2
- BigData technologies advanced in the meantime and now we have the choice between many different products and options
- **Studies of new data formats and/or new storage technologies performed over the last 2 years concluded that Kudu is the most promising technology for the next few years**
- Hence this prototype!

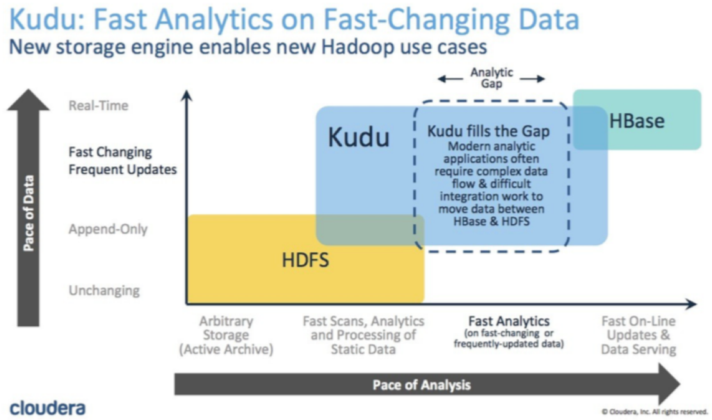
Exploring Kudu

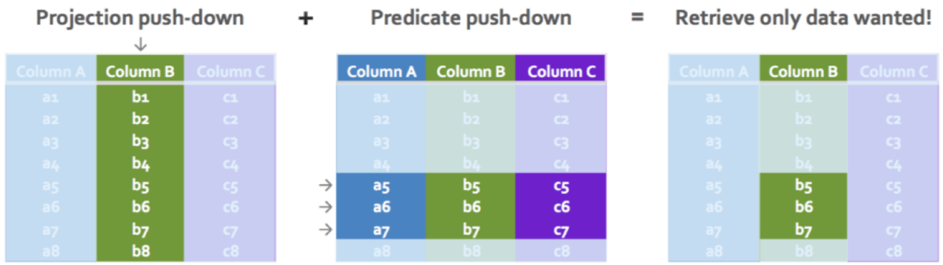
Kudu is a new technology in the Hadoop ecosystem

Optimization and unification of data storage for the EventIndex

Apache Kudu: new columnar-based storage that allows fast insertions and retrieval.

- Tables with defined schema, primary keys and partitions. No foreign keys
- Ingestion and query scanning are distributed among the servers holding the partitions (tablets)
- Partition pruning and projection/predicate pushdown





Benefits for EventIndex:

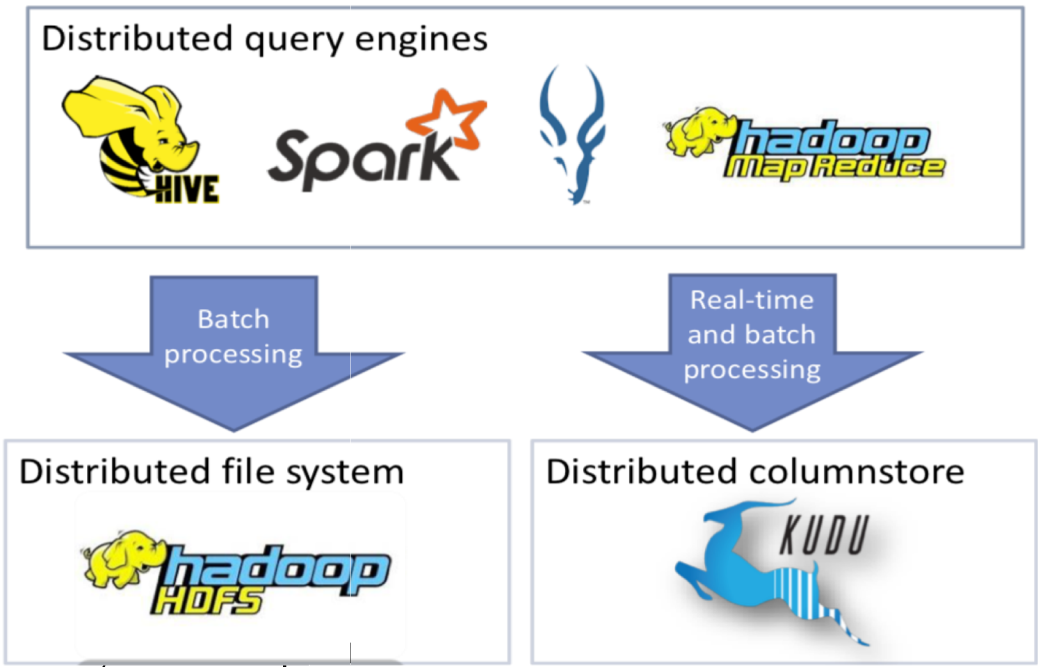
- Unify data for all use cases (random access + analytics)
- Related data (reprocessings) sit close to each other on disc. Reduce redundancies and improve navigation.

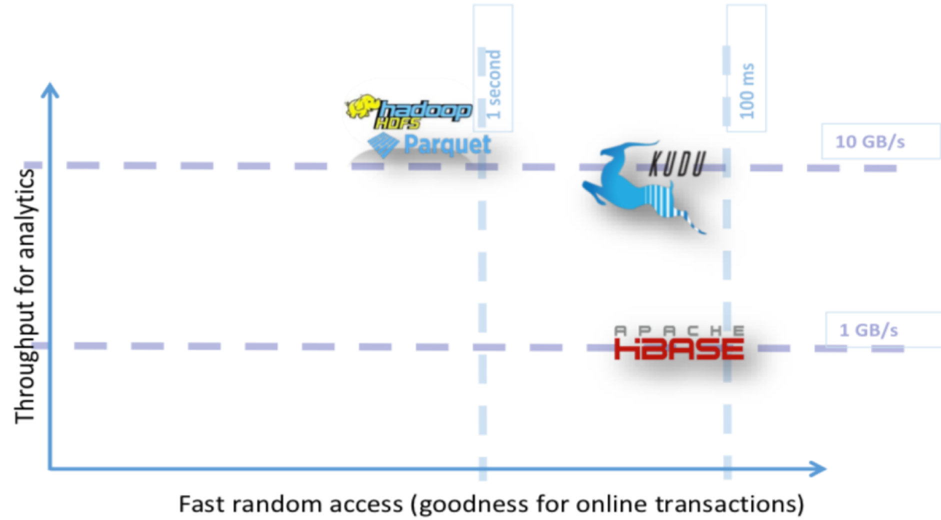
What can we gain with Kudu

- Reduce ingestion latency by removal of multi-staged data loading into HDFS
- Enable in-place data mutation
- Enable common analytic interfaces Spark and Impala (SQL, scala, python)
- ...and improve random lookup and analytics performance

More on Kudu

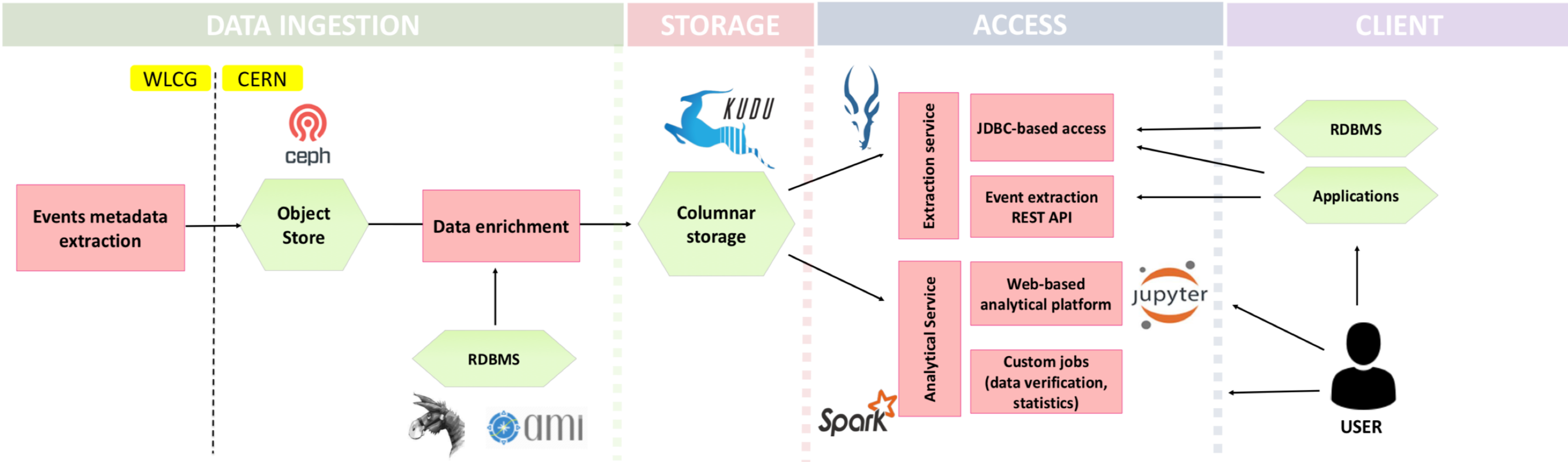
- Next generation scalable and distributed table-based storage designed for HTAP systems – **Hybrid Transactional and Analytical Processing** [5]
- Unlike Hadoop Distributed File System (HDFS), Kudu provides indexing and columnar data organization natively – this is to establish a good compromise between random **data lookups** and **analytics** performance
- Organization of the data in sharded tables with named columns, types and a primary index makes Kudu very attractive for systems with relational data models that needs to scale-out
- Apache Kudu is supported by top open-source frameworks for parallel data processing and computation including
  - Apache Spark, Apache Impala, Apache Hive, MapReduce,...





Kudu tests @ CERN

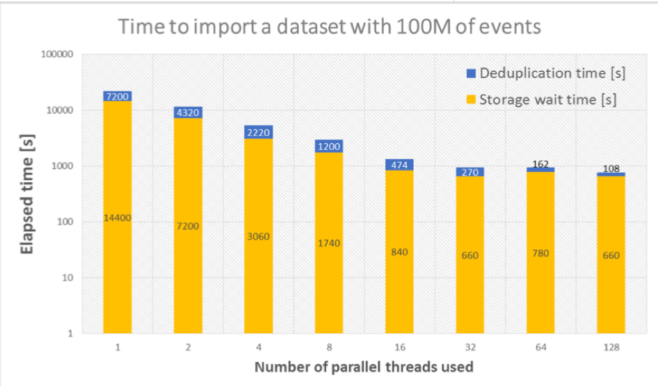
CONCEPT OF THE NEW ATLAS EVENT INDEX PLATFORM



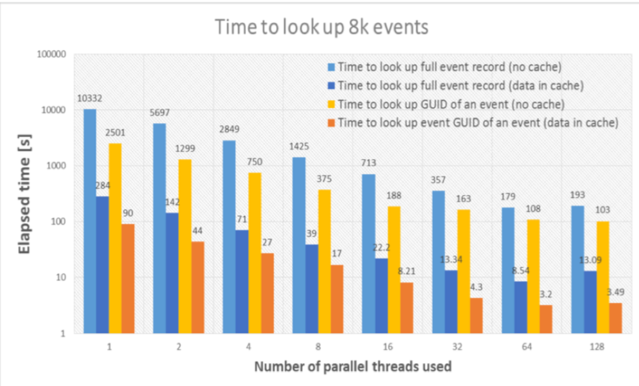
MEASURED PERFORMANCE WITH APACHE KUDU STORAGE

DATA INGESTION | ACCESS | ANALYTICS

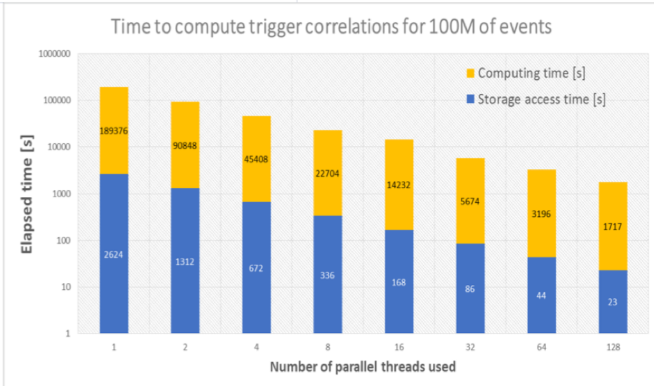
Hardware specification: cluster of 12 machines with 2 x 8 cores @2.60GHz, 64GB RAM, 48 SAS drives



- Data loading tests were performed with **Apache Spark** 2.2.1 using **real** data from the current production system
- Before loading a dataset to Apache Kudu, duplicated events are filtered out and stored in a dedicated table
- Measured average writing speed was **5kHz** per thread, max overall writing speed to a Kudu cluster was **120kHz – this is ~10x more than what is needed today**

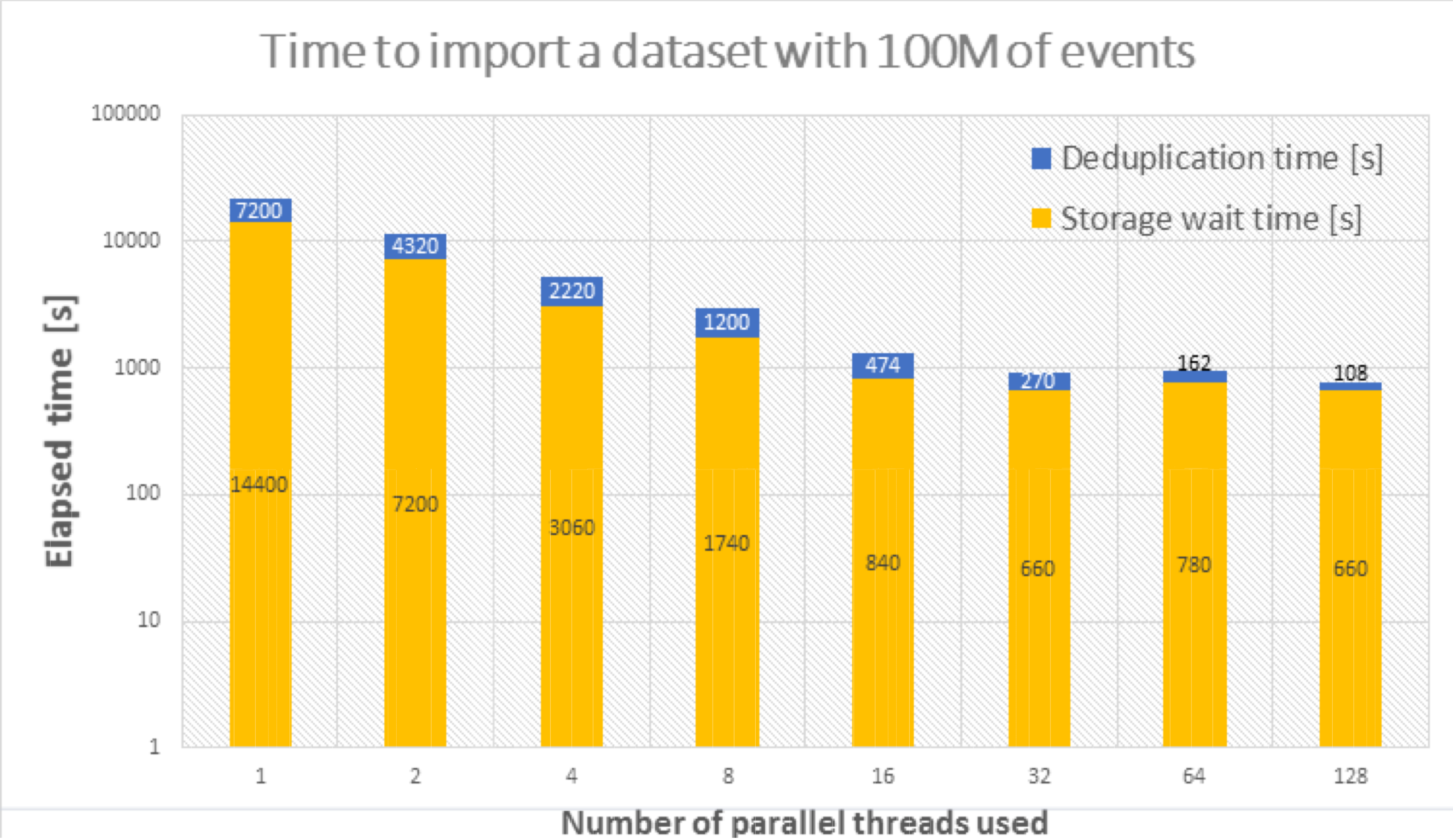


- On the plot above, we present the time to look up eight thousands of random events records (full record or just a GUID attribute) from Apache Kudu with a simple client written in **Python**
- The results for each type of a lookup were grouped into two cases; a pessimistic one (no cache used) and an optimistic one (all data where lookup from Kudu cache)
- **Average event lookup time below 1s and ability to handle more then 400 requests per second fully satisfies the system needs**



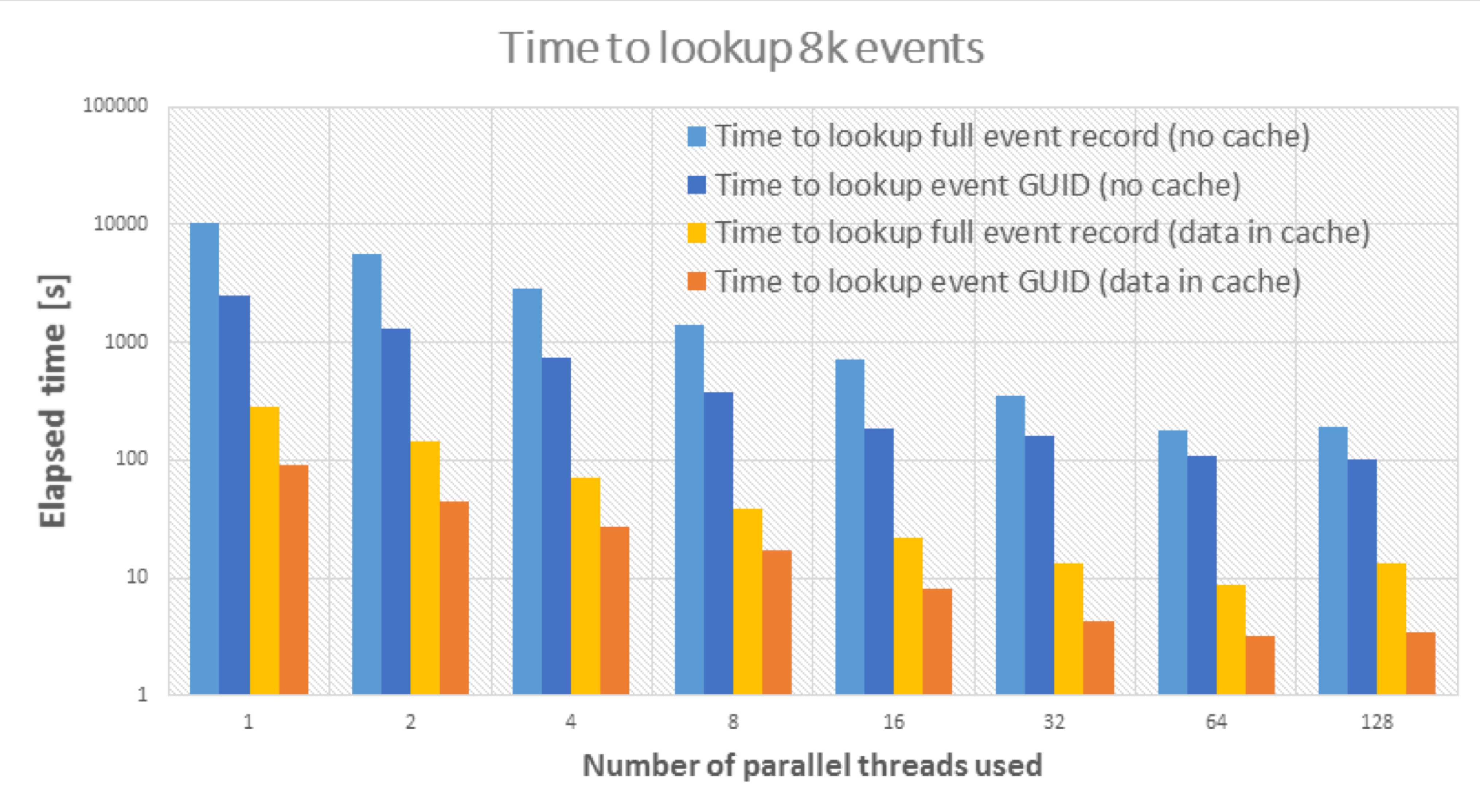
- Data analytics tests were performed with **Apache Spark** 2.2.1 reading Atlas EventIndex from data stored in Apache Kudu
- In the test case, we count occurrences of all combinations of trigger bits pairs within a dataset of 100M events
- The trigger count computation on a Spark cluster takes the majority of the wall time (52 hours), when data retrieval from Kudu is just a small fraction (< 2%) of it (45 minutes). A single scanner thread could deliver 40k of records per second
- Scalable data scan performance in combination with modern data processors (Spark, Impala) opens the system to new use cases on a filed of data exploration and analytics (like counting trigger correlations)

Kudu tests @ CERN



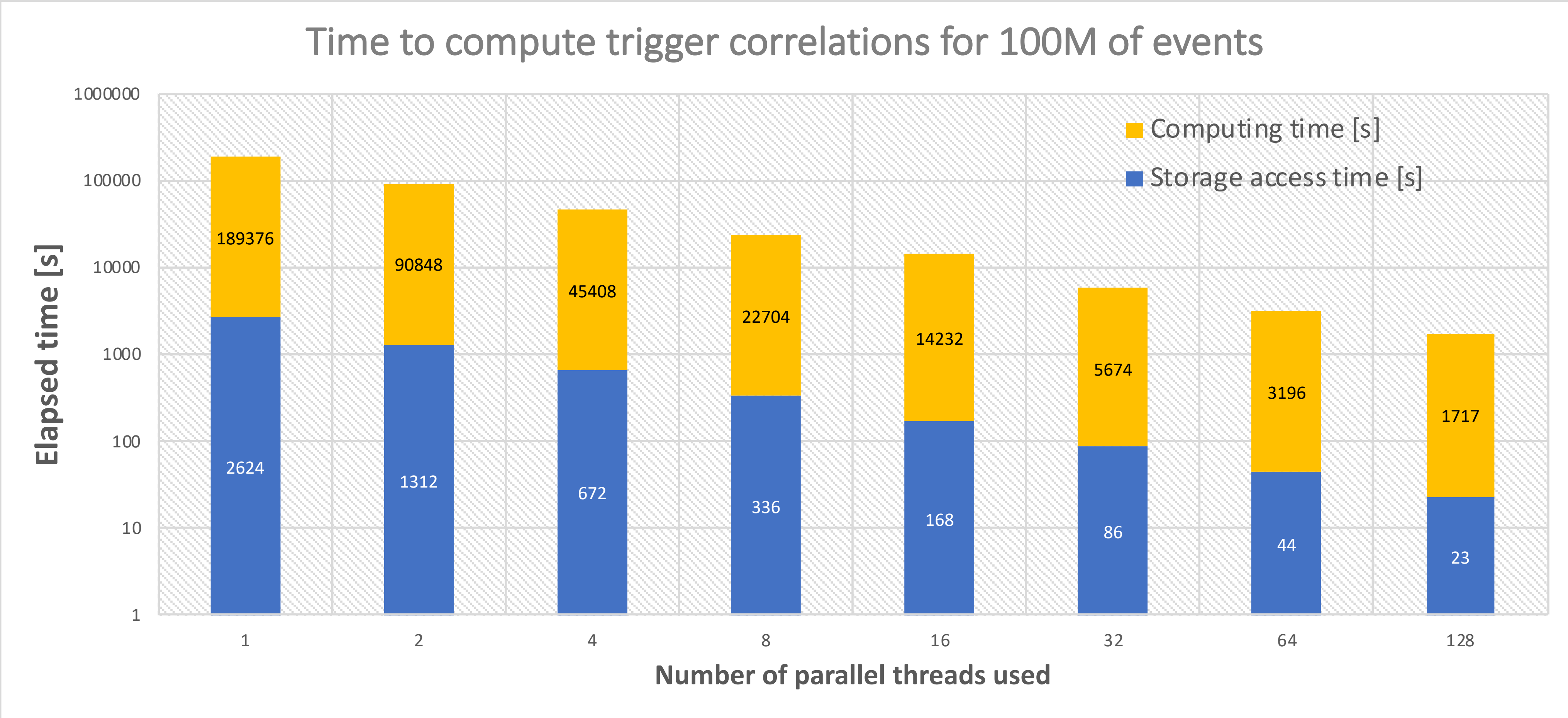
- Data loading tests were performed with Apache Spark 2.2.1 using real data from the current production system
- Before loading a dataset to Apache Kudu, duplicated events are filtered out and stored in a dedicated table
- Measured average writing speed was 5kHz per thread, max overall writing speed to a Kudu cluster was 120kHz – this is ~10x more than what is needed today

Kudu tests @ CERN



- The plot shows the time to look up eight thousand random events records (full record or just a GUID attribute) from Apache Kudu with a simple client written in Python
- The results for each type of a lookup were grouped into two cases:
  - a pessimistic one (no cache used) and an optimistic one (all data where lookup from Kudu cache)
- Average event lookup time below 1s and ability to handle more then 400 requests/second fully satisfies the system needs

Kudu tests @ CERN

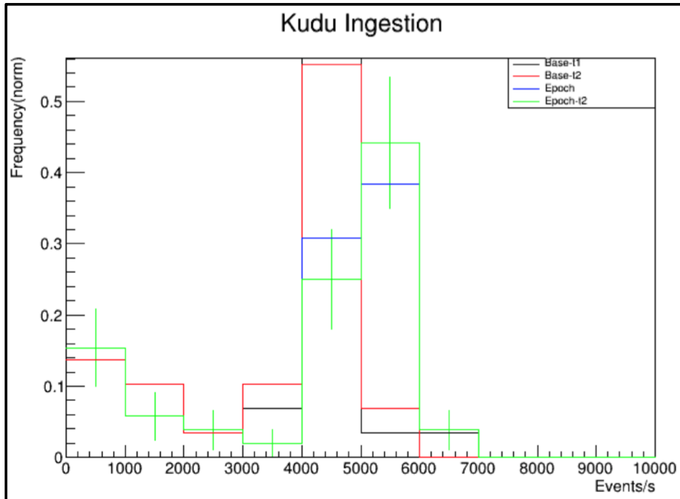


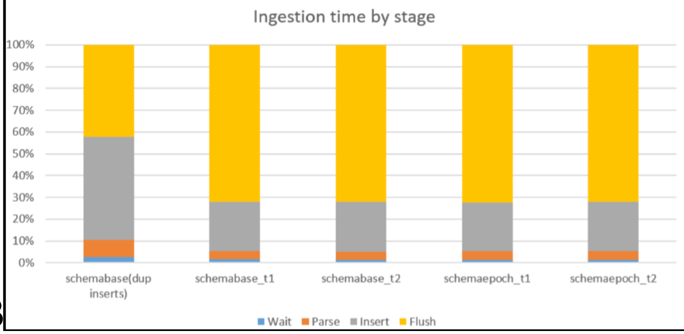
- Data analytics tests performed with Apache Spark 2.2.1 reading Atlas EventIndex from data stored in Apache Kudu
- In the test case, we count occurrences of all combinations of trigger bits pairs within a dataset of 100M events
- The trigger count computation on a Spark cluster takes the majority of the wall time (52 hours), when data retrieval from Kudu is just a small fraction (< 2%) of it (45 minutes). A single scanner thread could deliver 40k of records per second
- Scalable data scan performance in combination with modern data processors (Spark, Impala) opens the system to new use cases on a filed of data exploration and analytics (like counting trigger correlations)

Kudu tests @ IFIC Valencia

- Current setup at IFIC
  - Kudu 1.7 + Impala 2.11 +Spark 1.6 (cdh5.14.2)
- 5 machines with:
  - 2x Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz (14 cores/CPU)
  - 16x 16 GB RAM DDR4 @ 2400 MHz (256 GB)
  - 8x data disks SATA SEAGATE ST6000NM0034 (6TB)
  - 1x os disk SSD SAMSUNG MZ7KM240 (240GB)
  - 1x Intel SSD DC P3700 (1.5 TB) pci nvme
  - 2x 10Gpbs ethernet controller
- Current configuration:
  - 1 master, 4 tablet servers
  - 1 big data disk (RAID10)
  - WAL on Intel SSD

Data ingestion test





- 1 consumer per table performance

- Input data: datasets from May 2018 (mainly tier0)
- Tested different tables/configuration:

Base-t1: HASH(eventnumber)=8 RANGE(runnumber)
-all May'18 ds in same range(runnumber)
Base-t2: same as Base-t1 with key ending
<…,runnumber, eventnumber>
Epoch: HASH(eventnumber)=4 RANGE(epoch)=4
Epoch-t2: HASH(eventnumber)=8 RANGE(epoch)=4

Ingestion mean rate: ~5K events/s

Consumer Ingestion Stages:
Wait: for data valid (1%)
Parse: data conversion (4%)
Insert: into Kudu client buffers (23%)
Flush: buffers to Kudu (72%)

Outlook

- The EventIndex project started in 2012 at the end of LHC Run 1 driven by the need of having a functional event picking system for ATLAS data
  - The data storage and search technology selected in the first phase of the project (Hadoop MapFiles and Hbase, in 2013-2014) was the most advanced available at that time in the fast-growing field of BigData and indeed after a couple of initial hiccups it proved reliable and performed satisfactorily
    - Part of the data are replicated also to Oracle for faster access but mainly to have a uniform environment between event and dataset metadata
- Nevertheless the current implementation of the EventIndex started showing scalability issues as the amount of stored data increases
  - Slower queries, lots of storage (compression helped)
- Kudu looks like a very promising solution that can carry the EventIndex through Run 3 (2021-2024)
  - Faster data injection and queries
  - Possibility of using analytics tools
  - Compatibility with SQL queries (connection to other info in relational databases)
- The plan is to finalise the schema by the end of 2018 and then load all Run 1 and Run 2 data from the Hadoop EventIndex and run them in parallel
  - If all goes well, by the end of 2019 we can run only Kudu and be happy!