

# The ATLAS EventIndex and its evolution towards Run 3

Evgeny Alexandrov<sup>1</sup>, Igor Alexandrov<sup>1</sup>, Zbigniew Baranowski<sup>2</sup>, Dario Barberis<sup>3</sup>, Luca Canali<sup>2</sup>, Gancho Dimitrov<sup>4</sup>, Álvaro Fernández Casani<sup>5</sup>, Elizabeth Gallas<sup>6</sup>, Carlos García Montoro<sup>5</sup>, Santiago González de la Hoz<sup>5</sup>, Julius Hrivnac<sup>7</sup>, Andrei Kazymov<sup>1</sup>, Mikhail Mineev<sup>1</sup>, Fedor Prokoshin<sup>8</sup>, Grigori Rybkin<sup>7</sup>, Javier Sánchez<sup>5</sup>, José Salt Cairols<sup>5</sup>, Petya Vasileva<sup>4</sup> and Miguel Villaplana<sup>9</sup>

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

On behalf of the ATLAS Collaboration

## ACAT 2019, Saas Fee, Switzerland

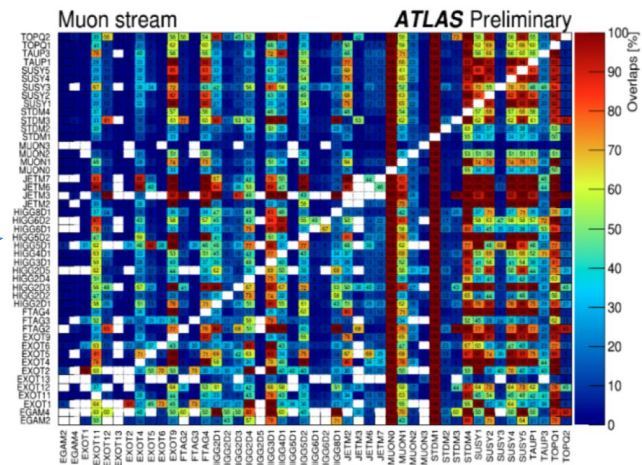
March 12, 2019

- The ATLAS experiment produces several billion events per year
- A database with the references to the files including each event in every stage of processing is necessary in order to retrieve the selected events from data storage systems
- The ATLAS EventIndex project provides a way to store the necessary information using modern data storage technologies
  - allows saving in memory key-value pairs
  - select the best tools to support this application
    - performance, robustness and ease of use
- An infrastructure was created which
  - allows fast and efficient selection of events of interest, based on various criteria, from the billions of events recorded
  - allows an indexing system that points to those events in millions of files scattered in a world-wide distributed computing system
  - contains records of all events processed by ATLAS, in all processing stages

- A system designed to be a [complete catalogue of ATLAS events](#), real and simulated data
- 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

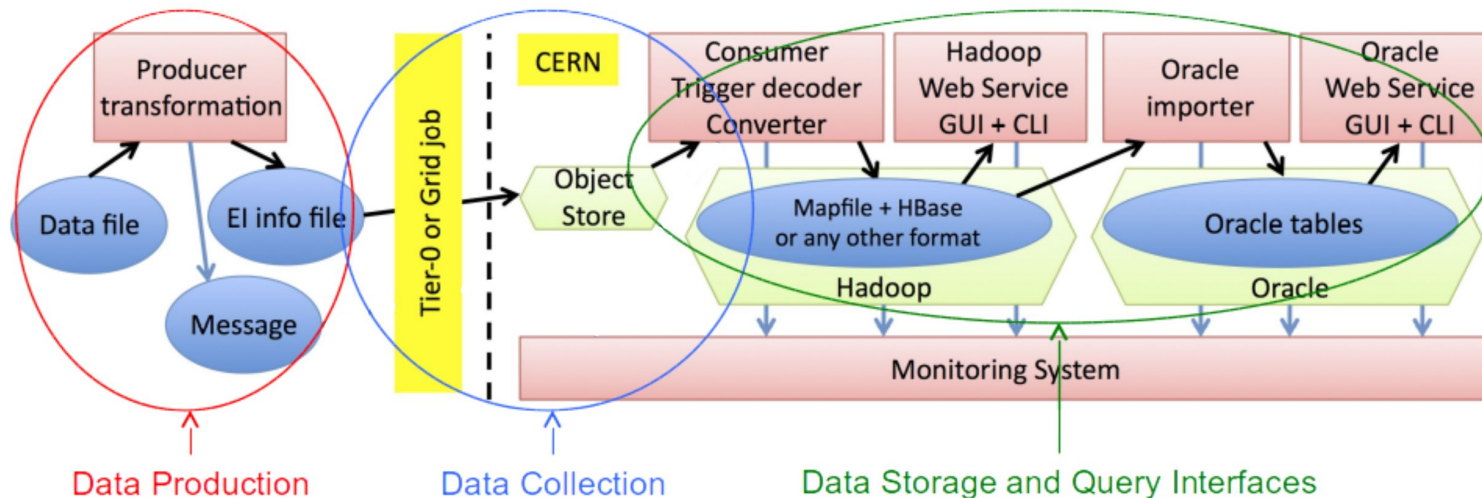
# Use cases

- **Event Picking**
  - Users able to select single events depending on constraints.
  - Order of hundreds of concurrent users
  - Requests ranging from 1 event (common case) to 30k events (occasional)
- **Production consistency checks**
  - **Duplicate event checking**
    - Events with same ID appearing in same or different files/datasets
  - **Overlap detection**
    - Construct the overlap matrix identifying common events across the different files
- **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 satisfy trigger Y



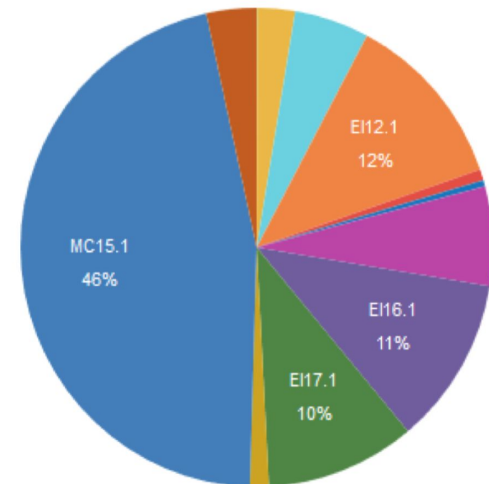
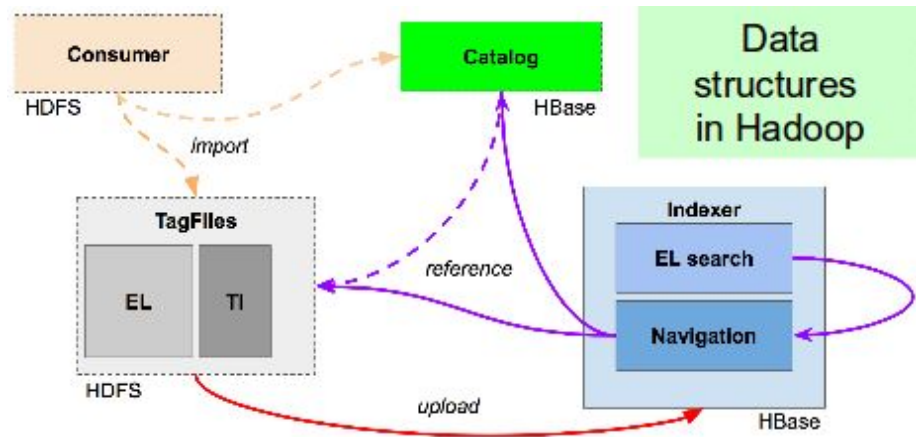
# Architecture

- Partitioned architecture, following the data flow:
  - **Data Production:** extract event metadata from files produced at Tier-0 or on the Grid
  - **Data Collection:** transfer EventIndex information from jobs to the central servers at CERN
  - **Data Storage:** provide permanent storage for EventIndex data and fast access for the most common queries + finite-time response for complex queries
    - Full info in **Hadoop**; reduced info (only real data, no trigger) in **Oracle** for faster queries
- **Monitoring:** keep track of the health of servers and the data flow
- System in continuous operation since Spring 2015



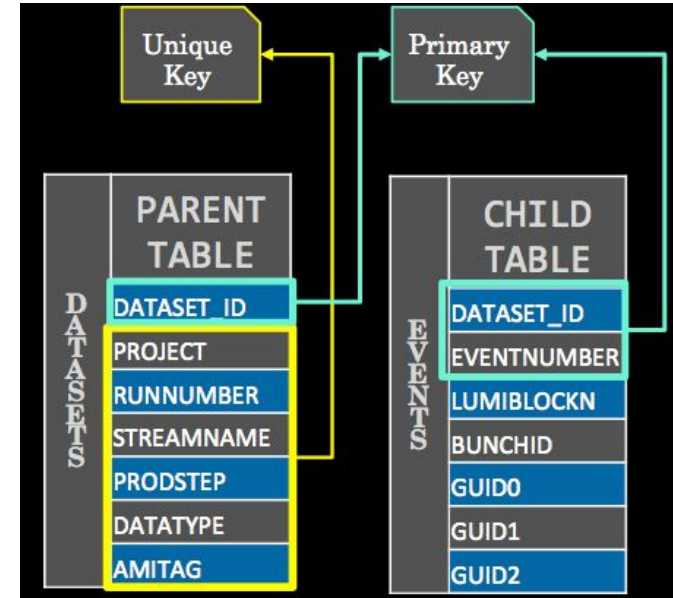
# Hadoop storage

- Hadoop is the **baseline storage technology**
  - it can store large numbers (10s of billions) of simply-structured records
  - and search/retrieve them in reasonable times
- Hadoop "MapFiles" (indexed sequential files) are used as data format
  - One MapFile per dataset
  - [Internal catalogue](#) keeps track of what is where and dataset-level metadata (status flags)
  - [Event Lookup index](#)
- CLI, RESTful API and GUI interfaces available for data inspection, search and retrieval
- Data volumes:
  - Real 2009-2018: 21 TB
  - Simul 2015-2018: 5 TB
  - Other (incl. backup): 150 TB



# 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
  - [optimised for event picking](#)
  - very good performance also for event counting by attributes (lumiblock and bunch ID)
- Connection to the RunQuery and AMI databases to check dataset processing completeness and detect duplicates
- Easy calculation of dataset overlaps
- GUI derived from COMA database browser to search and retrieve info
- **Currently:** 72K Datasets (170 Billion event records) efficiently stored within 3.2TB table segments plus 2.8TB auxiliary index





# Guidelines for Next Generation EventIndex

- An evolution of the EventIndex concepts
  - **Currently:** the same event across each processing step (RAW, ESD, AOD, DAOD, NTUP) is physically stored at different HADOOP HDFS files.
  - **Future:** One and only one logical record per event (Event Identification, Immutable information (trigger, lumiblock, ...), and for each processing step:
    - Link to algorithm (processing task configuration)
    - Pointer(s) to output(s)
    - Flags for offline selections (derivations)
- Support Virtual Datasets:
  - **A logical collection of events**
    - Created either explicitly (giving a collection of Event Ids) or implicitly (selection based on some other collection or event attributes)
    - Labelling individual events by a process or a user with attributes (key:value)
- Evolve EventIndex technologies to future demanding rates:
  - **Currently:** ALL ATLAS processes: ~30billion events/day (up to 350Hz on average) → update rate throughout the whole system (all years, real and simulated data). Read 8M files/day and produce 3M files
  - **Future:** due to expected trigger rates, **need to scale for next ATLAS runs:** at least half an order of magnitude for Run3 (2021-2023): 35 B new real events/year and 100 B new MC event/year.
    - For Run 4: 100 B new real events and 300 B new MC events per year. Then sum up replicas and reprocessing



- **Apache HBase** is the Hadoop database, a distributed, **scalable**, big data store.
  - Open-source, distributed, versioned, non-relational database modeled after the Google BigTable paper
  - It is built on top of HDFS AND provides **fast** record lookups (and updates) for large tables
  
- **HBase organizes data into tables**
  - Tables have **rows** and **columns**, which store values. (Like a spreadsheet)
    - Rows are identified uniquely by their row key
    - Each row can have a different schema
    - Data within a row is grouped by column family. Must be defined up front and are not easily modified
    - **Values can be accessed given row key and column name**
  
- **RowKey design desirable properties:**
  - Have a small size
  - Should identify an event uniquely inside a ‘container’ (event peeking)
  - Allow searches reading the smallest quantity of consecutive data using ‘range scans’
  - Avoid sparse insertions into existing row key space
  - Try to use all regions evenly both, reading and writing
  - When growing, use the RowKey space homogeneously

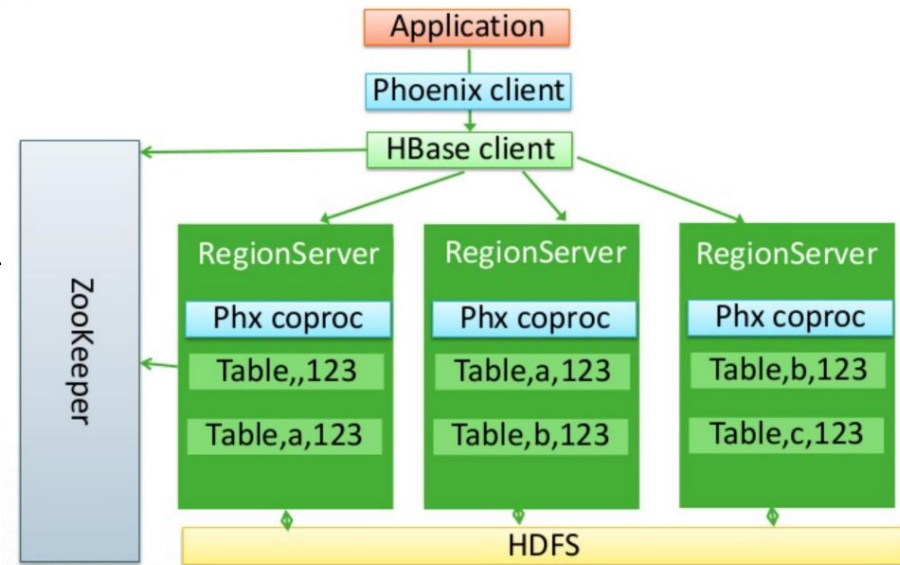
# SQL on HBase

- Various possibilities for SQL on HBase
- **Apache Impala** – handling of a row key mapping has to be on the application side
- **Apache Hive** – has the same issues as Impala
- **Apache Spark** – mainly for batch jobs

- **Apache Phoenix**



- OLTP and operational analytics for HBase through SQL
  - Takes SQL query
  - Compiles it into a series of HBase scans
  - Direct use of the HBase API, along with coprocessors and custom filters
  - Produces regular JDBC result sets



- **RowKey** design can be adapted to Phoenix's types and sizes (losing some performance)
- Phoenix allows the use of RowKey fields in queries but they are stored as one entity in HBase

# Tests @ CERN

- Base on experience from previous years we had a mixed feeling about the performance of HBase
  - Mainly, because HBase installations were not tuned – running on default settings
- In the past months CERN IT Hadoop Service put big effort to understand and [optimize HBase configuration](#)
- HBase 2.x seems to be even faster – reduced garbage collecting by JVM
  
- **Test setup**
  - Table schemas
    - Datasets
      - PRIMARY KEY (runnumber, project, streamname, prodstep, datatype, version, dspid)
    - Events (salting enabled)
      - PRIMARY KEY (dspid,eventnumber)
      - DATA\_BLOCK\_ENCODING='FAST\_DIFF', COMPRESSION='SNAPPY', SALT\_BUCKETS = 10;
  - Apache Spark used for reading from sequence files and writing to HBase (via Phoenix API)
  - Test cluster
    - 12 machines
    - 32 vcors, 64 GB of RAM
    - CDH 5.15 installed

- **Loading Atlas EventIndex data to HBase via Phoenix**

- 103 B real events loaded, 50k datasets
- Avg loading speed 80kHz
- Size in HBase: 36.8 TB
- Avg record size: 392B
- Number of regions: 10667
- Avg data extraction time: 0.3 s



- **Phoenix queries**

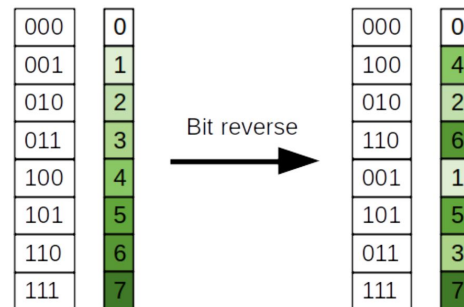
- First Phoenix query on events table takes some time (13s). Then any query returns within 1s
- Joint queries must be done carefully
  - A query may take too long and fail
  - It is safer and typically faster to make queries on datasets and events separately instead of joining them together

# Tests @ IFIC. HBase RowKey design

- We can have a binary composite key with the smallest number of components:

**< (salt).dspid.dstypeid.dssubtypeid.eventno.seq > : < (1).4.1.1.8.2 > B**  
**(Byte).Integer.TinyInt.TinyInt.BigInt.SmallInt**

- **salt**: the slicing function is made internally by Phoenix.
  - **dspid** is an identifier based on: < project, runNumber, streamName, prodStep, version >
  - **dstypeid** and **dssubtypeid** are identifiers of the data type
  - **seq**: allows insertion of duplicate events
- This key allows direct event peeking, range scan over all the events for a given dataset and range scan over all the events for all the derivations streams for the same dspid
  - As a useful side effect, defines a ‘canonical container’ which are all the events that share the same dspid.dstypeid
  - To improve write speed during insertion of events for a dataset, we divide the rowkey space (**salt**), so all the writes don’t go to the same region
  - Use **reverse bit order** function to distribute keys evenly over the whole key space
  - Phoenix allows the use of RowKey fields in queries but they are stored as one entity in HBase
  - All Int fields in RowKey are unsigned



- Families represent related data that is stored together on the file system
- Therefore, all column family members should have the same general access pattern.
- In this tests:
  - A: Event location
  - B: Event provenance
  - C: Event description
  - D: Level 1 trigger (L1)
  - E: High Level Trigger (HLT)

family	name	type	
a	guid	binary(16)	GUID of file
a	oid1	unsigned_int	
a	oid2	unsigned_int	
b	prov	varchar(10000)	Provenance (JSON)
c	smk	unsigned_int	Trigger SuperMasterKey
c	hltpsk	unsigned_int	HLT prescaler key
c	l1psk	unsigned_int	L1 prescaler key
c	lumiblocknr	unsigned_int	
c	bunchid	unsigned_int	
c	eventtime	unsigned_timestamp	
c	lv1id	unsigned_long	
d	lumiblocknr	unsigned_int	
d	bunchid	unsigned_int	
d	tbp	unsigned_smallint[]	Trigger before prescaler
d	tap	unsigned_smallint[]	Trigger after prescaler
d	tav	unsigned_smallint[]	Trigger after veto
e	lumiblocknr	unsigned_int	
e	bunchid	unsigned_int	
e	ph	unsigned_smallint[]	HLT Physics
e	pt	unsigned_smallint[]	HLT Passtrought
e	rs	unsigned_smallint[]	HLT Resurrected

# Tests @ IFIC. HBase compression

- Storage size comparison for datasets totalling 224,389,536 events

compression	A (GB)	B (GB)	C (GB)	TOT (GB)
snappy	5.1	9.2	11.4	25.7
none	17.2	49.1	33.7	100.0
Snappy 1-letter	5.0	9.3	9.3	23.7

compression	A	B	C	TOT
snappy	5.1 GB	9.2 GB	11.4 GB	25.7 GB
	19.8 B/e	35.3 B/e	44.3 B/e	99.4 B/e

- [Snappy compression factor 4:1](#)
- Reducing the columns name to 1 letter improves used space, but not enough to justify the loss of human readability.
  - Try to reduce names to a meaningful minimum

# Alternatives to store information

- **Several alternatives have been tried to store trigger information:**
  - Binary packing. Two codings:
    - Use 10 (12) bits for L1 (HLT). (TC)
    - Use 10 (12) bits for L1 (HLT), and pack the components together
  - Native Phoenix arrays (smallInt [])
    - using one array per trigger: TBP, TAP, TAV, PH, PT, RS
    - using just one array per trigger level (L1 and HLT)
      - storing all components together with their sizes at the end
  - Use also TPB\*, TAP\*, TAV\* that eliminates triggers present in next stages
- **Possible block encodings:**
  - Prefix, Diff, Fast\_diff, Prefix\_tree
- Comparison of size in HDFS have been made using 10 M events
  - HDFS files sizes have been read after compaction. File redundancy is not taken into account





# Test results

- FAST\_DIFF better for Location (A) and Provenance (B)
- DIFF better for Event description (C), L1 and HLT
- DIFF seems to be better but FAST\_DIFF is good enough
- L1 trigger has to be stored without repeating triggers
- Pack trigger into binary data is the best method
- Use one array to store all triggers for a level is not much worse and uses standard Phoenix coding

bytes/event	PREFIX	DIFF	FAST_DIFF	PREFIX_TREE
A	28.8	20.3	19.5	28.8
B	23.0	19.0	17.4	23.0
C	48.8	43.6	47.0	48.8

	PREFIX	DIFF	FAST_DIFF	PREFIX_TREE	Trigger L1
D	82.6	81.7	82.8	82.6	TC
D0	137.0	134.5	142.1	137.0	3 arrays TBP, TAP, TAV
D1	89.3	90.2	98.4	89.3	3 arrays TBP*, TAP*, TAV*
D2	85.5	82.1	84.3	85.5	Cod10. TBP*, TAP*, TAV*
D3	85.0	82.8	84.3	85.0	Cod10. TAV*, TAP*, TBP*
D4	85.5	82.6	84.5	85.5	Cod10. TAP*, TAV*, TBP*
D5	87.3	82.1	84.0	84.9	Cod10. TAP*, TBP*, TAV*
D6	88.3	84.1	87.0	88.3	Array. TBP*, TAP*, TAV*
D7	87.3	87.4	86.5	87.3	Array. TAV*, TAP*, TBP*
D8	87.9	84.4	86.8	87.9	Array. TAP*, TAV*, TBP*
D9	87.8	84.4	86.8	87.8	Array. TAP*, TBP*, TAV*

	PREFIX	DIFF	FAST_DIFF	PREFIX_TREE	Trigger HLT
E	86.5	85.8	87.2	86.5	TC
E1	80.0	73.9	80.9	80.0	3 arrays PH, PT, RS
E2	72.8	71.4	73.4	72.8	Cod12. PH, PT, RS
E3	71.1	67.1	70.6	71.1	Cod12. RS, PT, PH
E6	75.6	74.2	76.7	75.6	Array. PH, PT, RS
E7	75.7	71.5	75.8	75.7	Array. RS, PT, PH

# Tests @ IFIC. Phoenix queries

- Some Phoenix functions have been written to help accessing the data:

- Event Picking**

```
> select EI_REF0(A.SR) from t0trig3.events where DSPID=-1073741824
and DSTYPEID=4 and DSSUBTYPEID=0 and EVENTNO=1901350885;
```

```
+-----+
| EI_REF0(A.SR, ''GUID'') |
+-----+
| 83EFC342-DD11-CB46-B3D6-FD98B0CA354F |
+-----+
1 row selected (0.043 seconds)
```

- Dataset composition:** which RAW data files is this dataset coming from?

```
> select EI_PRV0(pv,'guid',1),count(*) from t0trig3.events where DSPID=-
1073741824 and DSTYPEID=4 GROUP BY EI_PRV0(pv,'guid',1) limit 10;
```

```
+-----+
| EI_PRV0(B.PV, 'guid', 1, -1) | COUNT(1) |
+-----+
| 00039BAE-3C8C-E811-B50C-44A8420A88BC | 1792 |
| 000488BB-5A8C-E811-881D-44A8420A88BC | 2360 |
| 000A4651-A18C-E811-9F0D-1866DA6D106D | 2970 |
| 000BC1D9-608C-E811-9F42-44A8420A88BC | 426 |
| 0011F872-448C-E811-A006-44A8420A8576 | 1956 |
| 00122CA2-808C-E811-AC7C-1866DA6D0964 | 2649 |
| 0015AAC4-918C-E811-85AC-44A8420A5EB7 | 297 |
| 0016C844-3E8C-E811-ADDC-44A8420A7621 | 2057 |
| 002167F6-418C-E811-85AC-44A8420A5EB7 | 2103 |
| 0023A65D-588C-E811-BCDD-44A8420A8576 | 2324 |
+-----+
10 rows selected (6.169 seconds)
```

- Select events using trigger**

```
> select eventno,EI_REF0(sr,'full') from t0trig3.events where
EI_TRIG0(d.l1,'tav','5 and 45') limit 10;
```

```
+-----+
| EVENTNO | EI_REF0(A.SR, 'full') |
+-----+
| 1510912226 | 98336FF4-AAD0-7C42-9FD6-D01A98DD6655:00000261-000024C3 |
| 2518915605 | 78DA7134-DDDE-1046-ADC0-711847F58B8B:0000025F-000027DF |
| 3220563715 | 7E2377FD-AF64-5F45-83E0-49A493E9A091:0000025F-00004410 |
| 3310720059 | BFA832E5-4759-D442-99D8-C73AC318C274:0000025E-000011D8 |
| 1901350885 | 83EFC342-DD11-CB46-B3D6-FD98B0CA354F:0000025F-00002FEF |
| 3741069365 | F11E806B-50CC-A24F-A571-2B28895F4BF4:00000261-00002B31 |
| 3327013818 | 4378FBF6-77FC-C04E-B081-CA3195CF9477:00000261-000013E7 |
+-----+
7 rows selected (10.811 seconds)
```

- Event duplicates**

```
> select count(*),SUM(a) from (select eventno,count(*) as a from
deriv4.events group by eventno having (count(*)>1));
```

```
+-----+
| COUNT(1) | SUM(A) |
+-----+
| 97140 | 214550 |
+-----+
1 row selected (2.967 seconds)
```

- Derivation overlaps** computation have been tested using MR and PHOENIX api

- 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)
- **Phoenix queries** and HBase new event **table prototypes** have been tested, and show encouraging results
  - We have a table schema candidate
  - Basic functionality is ready. Working towards improved performance and better interfaces
  - Need to keep testing with more data and get performance metrics
- If all goes well, we plan to have the new system operational during 2019 in parallel with the old one, and phase out the old system during 2020 (well in advance of the start of LHC Run 3)

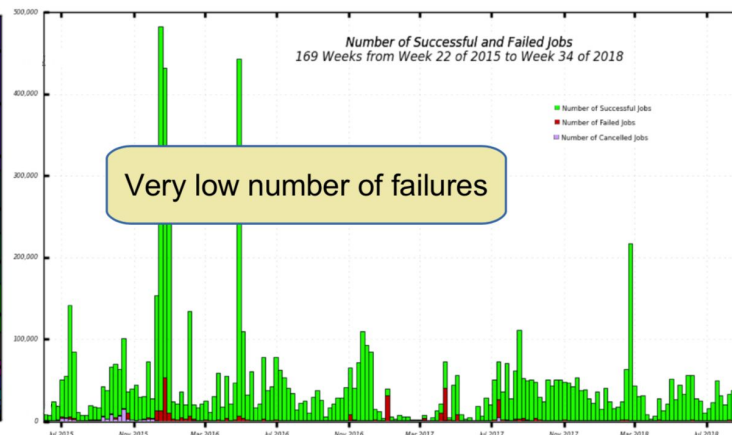
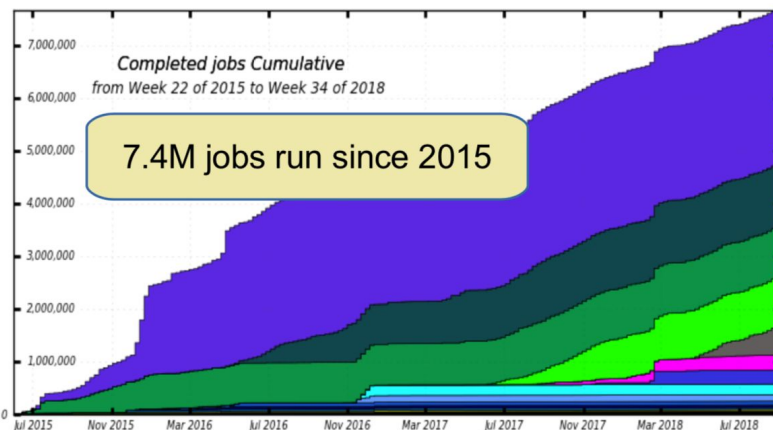
# Thanks



[home.infn.it](http://home.infn.it)

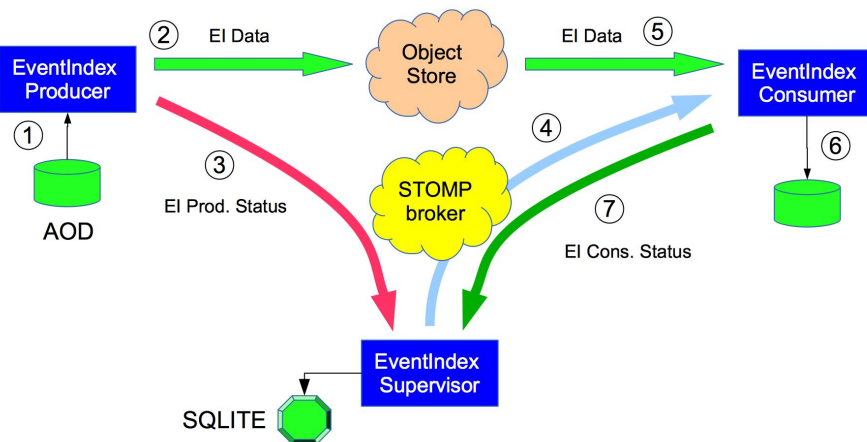
# Data production

- **Tier-0 jobs** index merged physics data (AODs), collecting also references to RAW data
- Similarly, **Grid jobs** collect info from datasets as soon as they are produced and marked as “complete” in the ATLAS Metadata Interface (AMI)
  - Other data formats (HITS, DAOD etc.) can be (and 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 collection

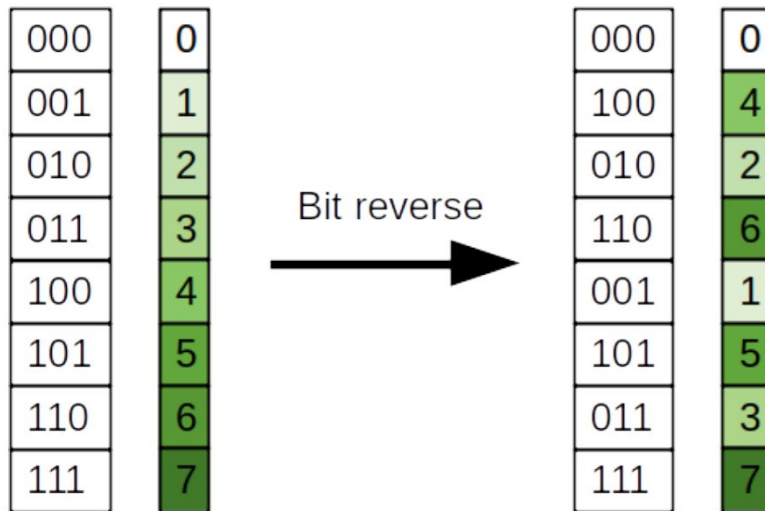
- **Producer:** Athena Python transformation, running at Tier-0 and grid sites. Indexes AOD data and produces an EventIndex file
  - EI information is sent by each job as a file to the ObjectStore at CERN ([CEPH/S3 Interface](#)) as intermediary storage
  - [Google protobuf](#) data encoding (compressed)



- **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 [Hadoop](#) storage

# Tests @ IFIC. RowKey space distribution

- If dspid increases monotonically, key distribution will be a problem
- So, we keep dspid increasing monotonically, but we don't use it directly
- We use instead  $\text{rev}(\text{dspid})$  where  $\text{rev}$  means **reverse bit order**



- This function has the nice property to **distribute keys evenly over the whole key space** from the beginning to the end
- Regions can be created in advance using the HBase pre-splitting feature

# Tests @ IFIC. R&W optimisation

- To improve write speed during insertion of events for a dataset, we can divide the rowkey space once more, so all the writes don't go to the same region
- Add a byte prefix to the rowkey <slice.dspid.dstypeid.eventnumber.seq>, where  $\text{slice} = \text{mod}(\text{eventnumber} \% \text{number\_slices})$
- To allow further division in the future, we use slice bits starting from the MSB to the LSB, so subdivisions include previous ones

	2 slices	4 slices	
A	00000000	00000000	A1
		01000000	A2
B	10000000	10000000	B1
		11000000	B2

- Events in slice A will be found in slice A1, even though this is not the 'correct' slice anymore
- Event peeking will involve looking on several rowkeys or just one if we keep the number\_slices along with the dspid



# Tests @ IFIC. HDFS L1 Trigger File Format

SEQ file with:

- Key: <LongWritable>      Value: <BytesWritable>

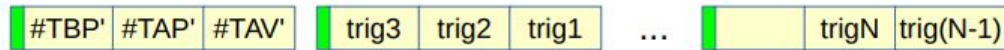
EventNumber

LB

BCID

L1Trigger

- LB and BCID are uint32
- L1Trigger is a collection of packed 10 bits triggerNumbers into 32 bit words:

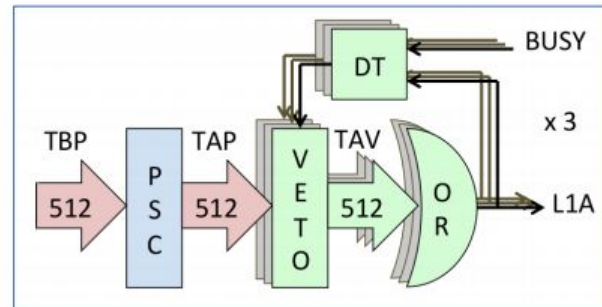


- Since  $TAV \subseteq TAP \subseteq TBP$  :

$$TAV' = TAV$$

$$TAP' = TAP - TAV$$

$$TBP' = TBP - TAP$$






# HBase block encoding types

- In HBase compressors and data block encoding can be used together on the same Column Family.
  - Data **block encoding** attempts to limit duplication of information in keys
  - **Compressors** reduce the size of large, opaque byte arrays in cells, and can significantly reduce the storage space needed to store uncompressed data
- **Data Block Encoding Types**
  - In **Prefix** encoding, an extra column is added which holds the length of the prefix shared between the current key and the previous key
  - In **Diff** encoding, instead of considering the key sequentially as a monolithic series of bytes, each key field is split so that each part of the key can be compressed more efficiently. Two new fields are added: timestamp and type
  - **Fast Diff** works similar to Diff, but uses a faster implementation. It also adds another field which stores a single bit to track whether the data itself is the same as the previous row. If it is, the data is not stored again
  - **Prefix tree** encoding provides similar memory savings to the Prefix, Diff, and Fast Diff encoder, but provides faster random access at a cost of slower encoding speed

Key Len	Val Len	Key	Value
24	...	RowKey:Family:Qualifier0	...
24	...	RowKey:Family:Qualifier1	...
25	...	RowKey:Family:QualifierN	...
25	...	RowKey2:Family:Qualifier1	...
25	...	RowKey2:Family:Qualifier2	...
...	...	...	...

**Prefix** 

Key Len	Val Len	Prefix Len	Key	Value
24	...	0	RowKey:Family:Qualifier0	...
1	...	23	1	...
1	...	23	N	...
19	...	6	2:Family:Qualifier1	...
1	...	24	2	...
...	...	...	...	...

- Currently used Cloudera 5.15 cannot be mapped directly to any Apache version
  - 2.6.x  $\leftrightarrow$  2.9.x
  - Same for Hbase: 0.98.x  $\leftrightarrow$  1.2.x
- Currently in other production clusters we use
  - Hadoop 2.7.6 and HBase 1.4.9
  - and it is stable, but not without bugs
  - planning to move to Hadoop 2.8.x soon
- On the other hand Hadoop 3.x seems to be production ready, we are currently testing it

# HBase RowKey compatibility with Phoenix

- RowKey is adapted to Phoenix types and sizes losing some optimizations
- Dstypeid is subdivided into dstypeid and dssubtypeid  
`< (salt).dspid.dstypeid.dssubtypeid.eventno.seq > : < (1).4.1.1.8.2 > B`  
`(Byte).Integer.TinyInt.TinyInt.BigInt.SmallInt`
- Seq allows event duplicates  
`seq = crc16 ( guid:oid1-oid2 )`
- The slicing function is made internally by Phoenix using:  
`salt = hash(rowkey) % bucketNum`  
and it is transparent to users creating the necessary subscans if needed
- Phoenix allows the use of RowKey fields in queries but they are stored as one entity in HBASE
- All Int fields in RowKey are unsigned