The EventIndex

ATLAS Events

EventIndex is a system designed to be a complete catalog of ATLAS events, real and simulated data

- Event is the basic unit of ATLAS data
- Each event contains result of a single triggered interactions, plus eventually piled-up interaction
  - Signals from the detector
  - Reconstructed particles with their parameters
  - Trigger decisions
- Uniquely identified by the run number and the event number
- Event information is stored in many instances
- Have different data types to fit analyses needs
- Spread among the hundreds of GRID sites

ATLAS Datasets

ATLAS event data is written in files that are organized in datasets

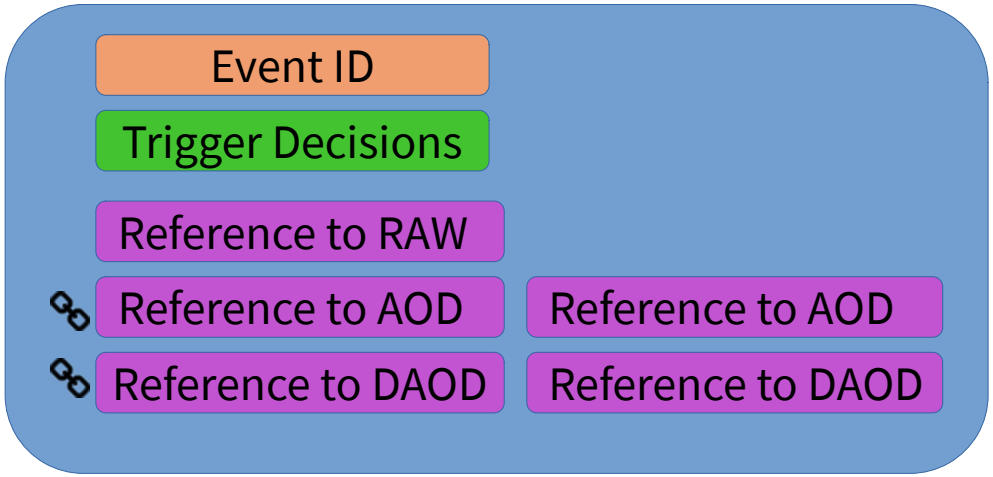
- Datasets can have different data types depending of the processing stage
  - Detector data is first written in the RAW data type
  - AOD datasets are produced after reconstruction.
  - Derived datasets (DAOD) for use in the specific analyses
- + Simulated (MC) datasets are produced on the GRID
  - EVNT datasets contain particles information
  - AOD datasets are produced after reconstruction.
  - Derived datasets (DAOD) for use in the specific analyses
- Various versions of the datasets originating from the same detector or simulated events
  - Different reconstruction settings and software version
  - Reprocessing roughly yearly

ATLAS EventIndex

- The ATLAS Experiment produces large amounts of data
  - several billion events per year
  - a database containing references to the events is necessary in order to efficiently access them from the distributed data storage
- The ATLAS EventIndex
  - provides a way to collect and store event information using modern technologies
    - automatically detects and indexes new data that was produced in this system from events collected from the detector or simulated
  - provides various tools to access this information through command line, GUI and RESTful API interfaces
    - allows fast and efficient selection of events of interest from the billions of events recorded, based on various criteria
    - provides an indexing system that points to these events in millions of files scattered through a worldwide distributed computing system
  - Contains ~300 billion of detector event records and more than 60 billion of the simulated event records

EventIndex records contain the following fields:

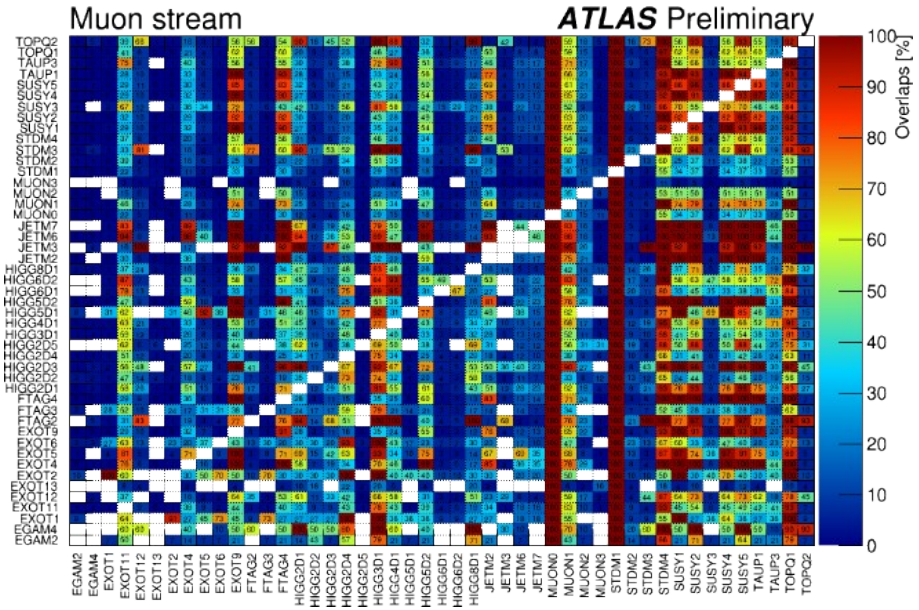
- **Event identifiers**
  - Run and event number
  - Trigger stream
  - Luminosity block
  - Bunch Crossing ID (BCID)



- **Trigger decisions**
  - Trigger masks for each trigger level
  - Decoded trigger chains (trigger condition passed)
- **References to the events at each processing stage in all permanent files generated by central productions (for event picking)**

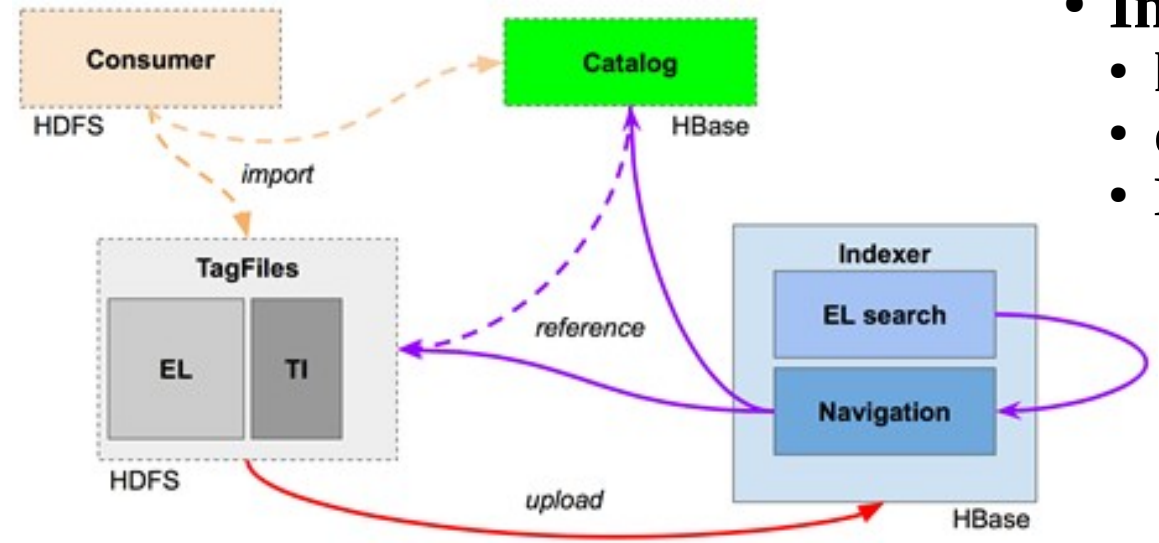
Use cases

- Event picking
  - give me event of specific type and processing version
- Counting and selection events based on trigger decisions
- Production completeness and consistency checks
  - Data corruption, missing and/or duplicated events
- Trigger chain overlap counting
- Derivation overlap counting
- Dataset Browsing
  - Finding datasets of interest
  - Dataset report
  - Dataset Inspection



EventIndex data Storage

- Hadoop was selected the baseline storage technology
  - store large numbers (10-s of billions) of simply-structured records
  - search/retrieve them in reasonable times
- Hadoop "MapFiles" (indexed sequential files) are used as data format, one MapFile per dataset

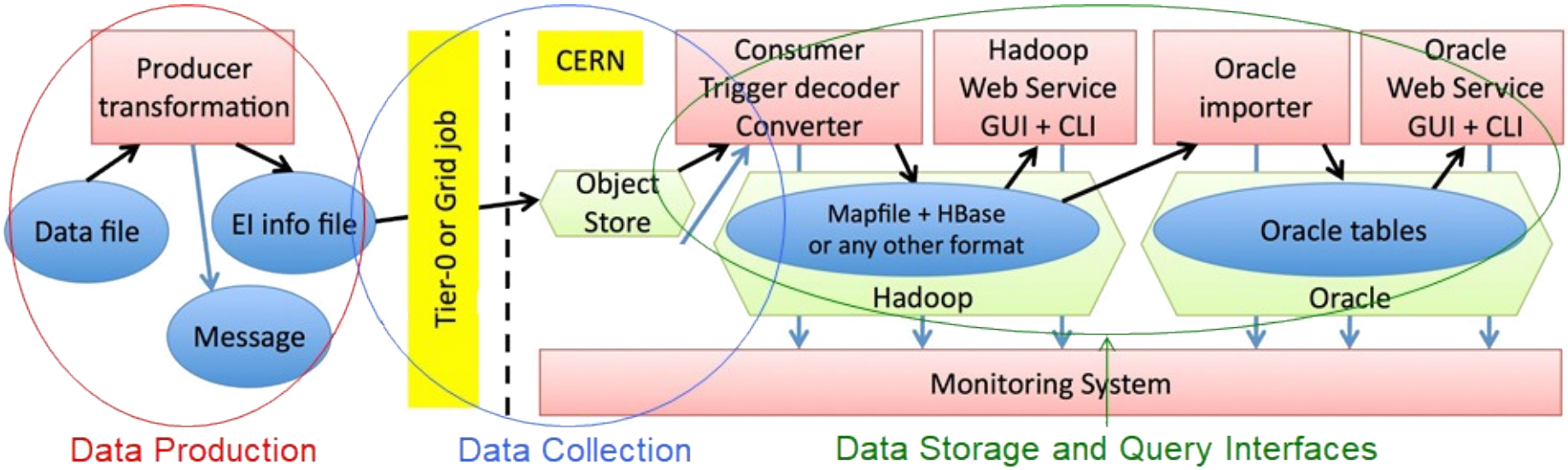


- Internal catalogue in HBase
  - keeps track of what is where
  - dataset-level metadata (status flags)
  - Event Lookup index

Oracle data storage

- Part of the data are replicated also to Oracle for faster access but mainly to have a uniform environment between event and dataset metadata
- Simple schema with dataset and event tables
- Filled with all real data, only event identification and pointers to event locations. Optimized for event picking

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**

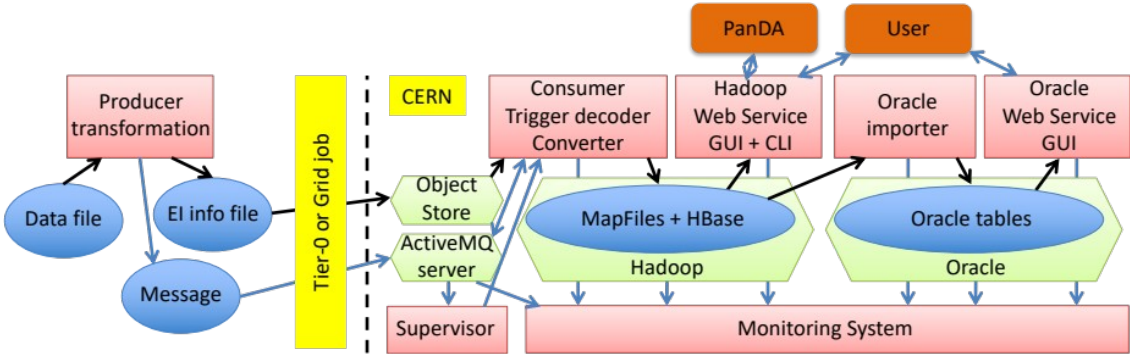
Evolution

- 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
  - Several components were updated or replaced in the course of time
- Baseline storage technology started showing scalability issues as the amount of stored data increases
  - Lots of duplicated Mapfiles on HDFS (Hadoop distributed file system): the same event across each processing step (RAW, ESD,AOD, DAOD, NTUP) is physically stored at different HADOOP HDFS files
- Significant increase in the data rates expected in future LHC runs demands transition to a new technology
- A number technologies were explored to improve
  - Scalability, Performance, Robustness
  - Adherence to modern software standards

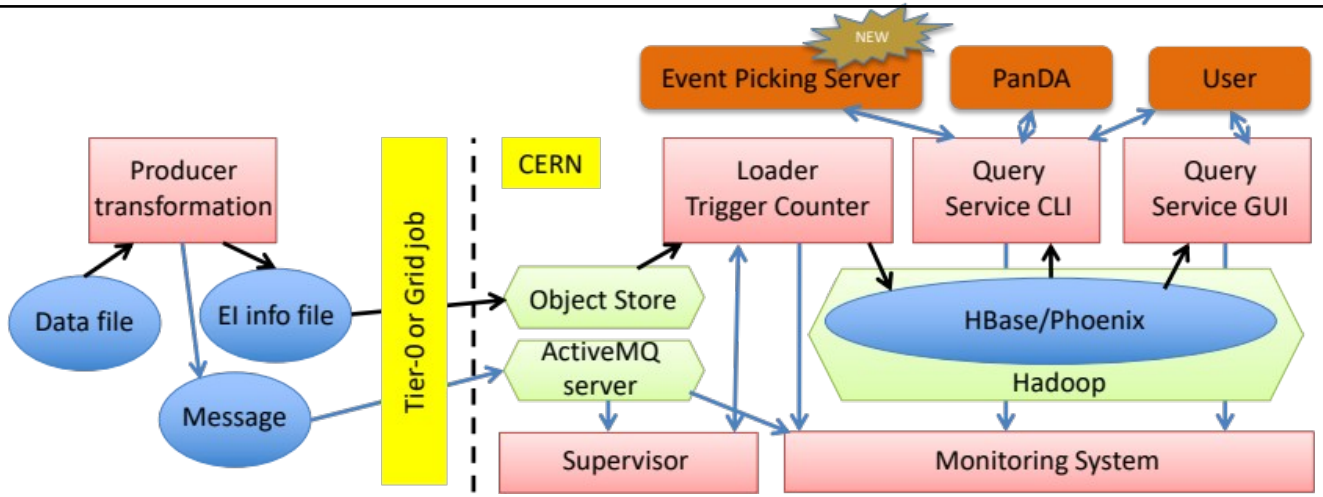
Architecture evolution

- **Replaced the Hadoop MapFiles and the Oracle tables with an HBase/Phoenix implementation**
  - **It joins the features of a BigData store in HBase with the possibility to run SQL queries through the Phoenix interface**
- **Adapted, updated or replaced other components at the same time**
- **Developed a new tool: the Event Picking Service**

EventIndex for Run 2



EventIndex for Run 3



EventIndex components

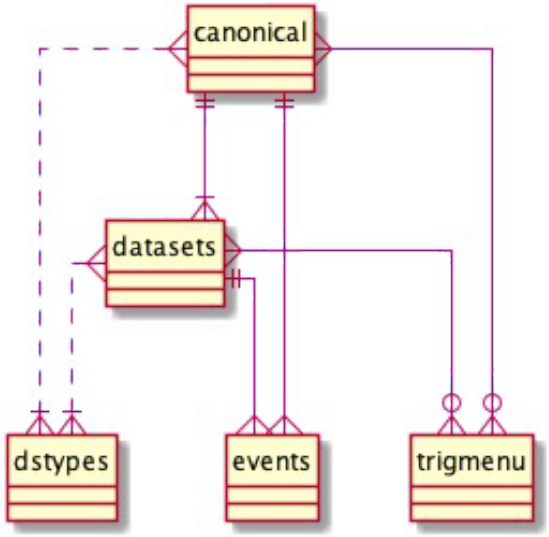
- **Producer: Athena Python transformation, running at Tier-0 and Grid-sites.**
  - **Indexes dataset data and produces a file with event information**
    - **Tier-0 jobs index merged physics AODs, collecting also references to RAW data**
    - **Grid jobs collect info from datasets as soon as they are produced and marked "complete" in ATLAS Metadata Interface (AMI)**
  - **EI Information is sent by each job as a file to the ObjectStore at CERN (CEPH/S3 interface) or to the EOS as intermediary storage.**
  - **A summary message is sent to the Supervisor**
- **Supervisor: Controls all the process, receives processing information, validates data by dataset and orchestrates its insertion in the final back-ends**
  - **Launches Loader to feed event tables**
  - **Feeds other auxiliary tables**
  - **Operated with a web interface**

EventIndex components

- **Phoenix Loader: tool that actually write the EventIndex data into HBASE/Phoenix events table**
  - **Reads data from the object store and write it to the supervisor**
  - **Write summary information to the HDFS to be read back by the Supervisor**
- **Trigger counter: a web service that provides information about the trigger of a given dataset**
- **Event Picking Service: Main scope is to automatise the operations needed for (massive) event picking**
  - **The GUI takes from the user the event list(s) and additional parameters (data type to be searched, data type to be retrieved, stream, AMI tag, etc.**
  - **The EPS splits the list(s) by run number, queries the EventIndex Query CLI, submits the event picking PanDA jobs, retries them if they time out during tape staging, validates outputs, informs the user when done.**
  - **The user can follow the progress using the GUI.**
- **Monitoring:**
  - **Information is being collected by python scripts driven by the schedulaer and then is written to the Influx DB through the HTTP endpoint**
  - **Grafana dashboards are being used for display of this information**

Data structures in HBase/Phoenix

- HBase tables works best for random access – event picking
- A range of data can be accessed by scanning data - analytic use cases



dstypes
• type : tinyint • name : varchar(20)
id : smallint

trigmenu
• smk : integer • type : tinyint • name : varchar(200)
id : smallint

events
• dspid : integer • dstypeid : smallint • eventno : bigint • seq : smallint
a.tid : integer a.sr : binary(32) a.mcc : integer a.mcw : float
b.pv : binary(34) ARRAY[]
c.lb : integer c.bcid : integer c.lpsk : integer c.etime : timestamp c.id : integer c.tbp : smallint[] c.tap : smallint[] c.tav : smallint[]
d.lb1 : integer d.bcid1 : integer d.hpsk : integer d.lph : smallint[] d.ph : smallint[]

datasets
• runno : integer • project : varchar(200) • datatype : varchar(200) • streamname : varchar(200) • prodstep : varchar(200) • version : varchar(200) • tid : integer • dspid : integer • dstypeid : smallint
smk : integer events_rucio : bigint files : integer events : bigint events_uniq : bigint events_dup : bigint files_dup : integer rank : smallint status : varchar(200) rucio_at : timestamp updated_at : timestamp dups_at : timestamp trigger_at : timestamp is_open : boolean is_deleted : boolean has_raw : boolean has_trigger : boolean prov_seen : smallint ARRAY[] sr_cnt : varchar(200) sr_clid : varchar(200) sr_tech : varchar(200)

canonical
• runno : integer • project : varchar(200) • datatype : varchar(200) • streamname : varchar(200) • prodstep : varchar(200) • version : varchar(200) • dspid : integer • dstypeid : smallint
smk : integer events_rucio : bigint files : integer events : bigint events_uniq : bigint events_dup : bigint files_dup : bigint rank : smallint status : varchar(200) rucio_at : timestamp updated_at : timestamp dups_at : timestamp trigger_at : timestamp is_open : boolean is_deleted : boolean has_raw : boolean has_trigger : boolean prov_seen : smallint ARRAY[] is_rucio_eq : boolean sr_cnt : varchar(200) sr_clid : varchar(200) sr_tech : varchar(200)

Data structures in HBase/Phoenix

- **Row keys**:
  - **Should include the most needed info**
  - **Have minimal possible size**
- **Chosen structure**:

dspid.dstypeid.eventno.seq

an identifier for the dataset name

dataType

event number

unique value in case of dataset name and EventNumber duplication

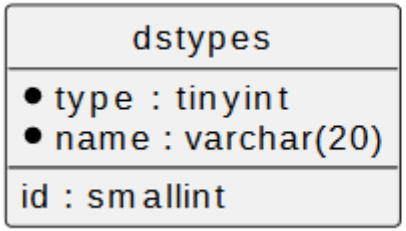
- **Data families**:
  - **A: Event location (and MC info)**
  - **B: Event provenance**
  - **C: Level 1 trigger (L1).**
  - **D: High Level Trigger (EF of HLT) and L2 for Run1 data**
- **Data import can be performed with Map/Reduce or Spark jobs**

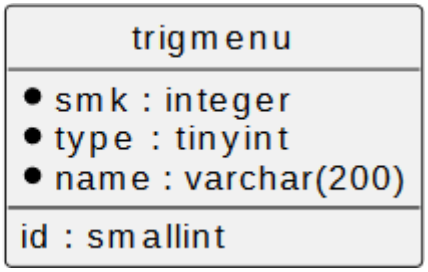
```
dspid           integer             NOT NULL ,
dstypeid        smallint            NOT NULL ,
eventno         bigint              NOT NULL ,
seq             smallint            NOT NULL ,

a.tid           integer                      ,
a.sr            binary(24)                   ,
a.mcc           integer,
a.mcw           float,

b.pv            binary(26) array             ,

c.lb            integer                      ,
c.bcid          integer                      ,
c.lpsk          integer                      ,
c.etime         timestamp                    ,
c.id            bigint                       ,
c.tbp           smallint array               ,
c.tap           smallint array               ,
c.tav           smallint array               ,

d.lb1           integer,
d.bcid1         integer,
d.hpsk          integer,
d.lph           smallint array,
d.lpt           smallint array,
d.lrs           smallint array,
d.ph            smallint array,
d.pt            smallint array,
d.rs            smallint array,

CONSTRAINT events_pk PRIMARY KEY
(dspid, dstypeid, eventno, seq)
```

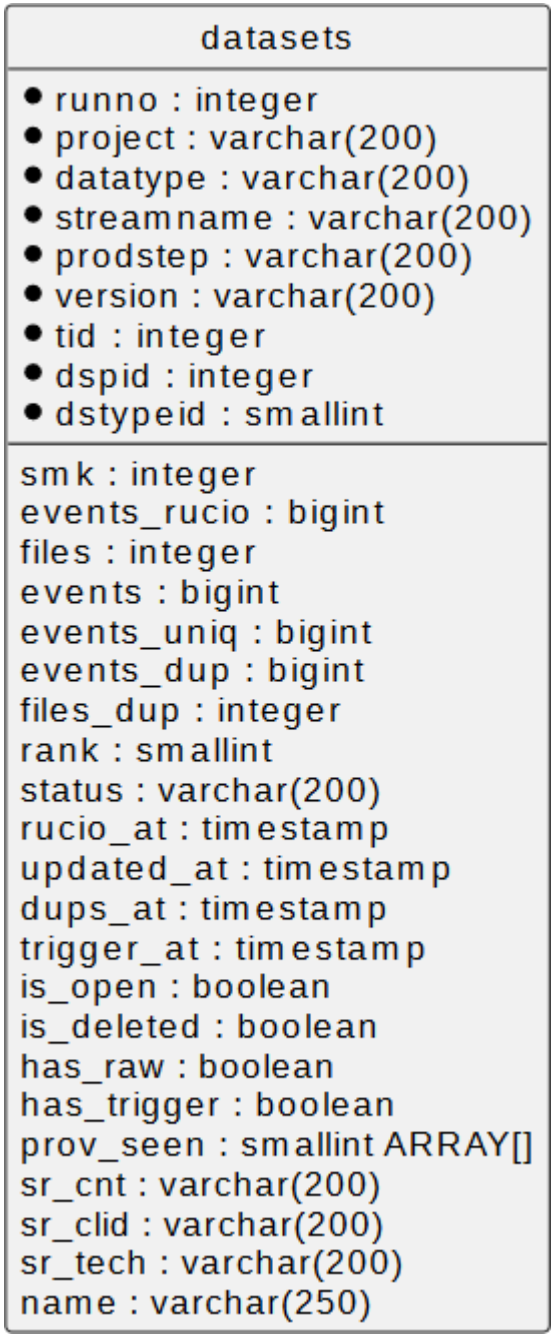
Data structures in HBase/Phoenix

Auxiliary tables to keep the dataset generated identifiers and bookkeeping data

- **Datasets table:**
  - Dataset location
  - Generated identifiers (dspid)
  - Import status
  - Some metadata information: number of all events, unique events, duplications
- **Data types table:**
  - data types (RAW, EVNT, AOD, DAOD, . . .)
  - subtypes for derivations







SPD Events amount

- The SPD Experiment have to produce large amounts of data
  - 20 GB/s => ~or 200 PB/year (RAW data)
    - *the same order or larger than ATLAS*
  - ~30 billion events per year
    - *the same order or larger than ATLAS*
- Event information will be reconstructed and processed to be used for physics analyses
- The result will be stored in few instances, having different data types to fit analyses needs
- Spread among the number of computing sites
- A database containing references to the event instances is necessary in order to efficiently access them from the distributed data storage
- Event data is written in files that are organized in datasets
- Various versions of the datasets originate from the same events
  - Different reconstruction settings and software version

SPD Datasets

Datasets have different data type for every processing stage

- **Real data from the detector:**
  - Time slices data written in the RAW format (transient ?)
  - After Online Filter → filtered RAW format
    - Fast reco event data, Unpacked raw hits, Raw data blocks
  - AOD and ESD datasets are produced after offline reconstruction.
    - AOD: Physics objects: particles, tracks, clusters …
    - ESD: + raw hits
  - Derived datasets for use in the specific analyses (?)
- **Simulated (MC) datasets.**
  - Event Generator produces EVGEN datasets
    - contain MC truth
  - After Simulation → RAW MC EVENTS
    - MC truth + unpacked raw hits
  - AOD and ESD datasets are produced after offline reconstruction.
    - The same as for real data + Mc truth

SPD Events

- Each event contains result of interactions within one time slice
- Event content
  - Event header: Run ID, FrameID, BCID, Event ID
  - RAW Hits

For RAW and ESD

For AOD and ESD

  - List of Vertices: Vtx,y,z + list of particles
  - Particles
    - ParticleID, list of tracks, list of ECAL, RS, ZDC clusters, TOF, AEG, BBC hits + MC truth for MC
  - ECAL, ZDC, RS clusters: X,Y,Z, energy, error, cluster shape (?)
  - BBC, TOF, AEG hits: X,Y,Z, Amplitude or TOF
  - Polarization: Polarization degree ?
  - **No online trigger information**
    - "Offline trigger" can be introduced instead
  -

EventIndex records may contain the following fields:

- Event identifiers
  - Run number
    - Can be run start time in a format YYMMDDhhmm: 2803080622
  - Event number
  - Frame ID
  - Bunch Crossing ID (BCID)
  - Stream (?)
- References to the events at each processing stage in all permanent files generated by central productions (for event picking)
- Selection parameters
  - Offline trigger chains
  - Event parameters (Polarization, etc...)

EventIndex record

- **EventIndex record example:**

very schematic

- ‘RunNumber’ UInt32 YYMMDDhhmm 2803080622
- ‘EventNumber’ UInt32 [0 : 4 294 967 295] 474 836
- ‘BCID’ UInt32 [0 : 4 294 967 295] 33
- ‘FrameID’ UInt32 [0 : 4 294 967 295] 500
- ‘DatasetNameRaw’ String ’data28.2803080622.RAW.root’

Stored at once

- ‘DatasetNameESD’ String ’data28.2803080622.ESD.version_3.root’
- ‘DatasetNameAOD1’ String ’data28.2803080622.AOD.software_version_55.root’
- ‘DatasetNameAOD2’ String ’data28.2803080622.AOD.software_version_155.root’
- ...
- ‘DatasetNameAOD9’ String ’data28.2803080622.AOD.software_version_6555.root’

- ‘EventParameter1’ UInt32 25565
- ‘EventParameter2’ Bool (Uint8) true
- ‘EventParameter3’ Float64 -3.14
- ...

Event parameters

- ‘OfflineTrigger1’ String ’2mu_20’
- ‘OfflineTrigger2’ String ’5tracks_p30’
- ...

Trigger chains

Added later

Use cases

- **Event picking**
  - give me event of specific data type and processing version
- **Count and select events**
  - based on event parameters
  - based on offline trigger decisions
- **Production completeness and consistency checks**
  - Data corruption, missing and/or duplicated events
- **Virtual datasets**
- **Selection overlap counting**
- **Derivation overlap counting**
- **Dataset Browsing**
  - Finding datasets of interest
  - Dataset report
  - Dataset Inspection

Command line tools

- **Atlas EventIndex event-lookup client:**
  - User queries event-lookup service, it sends request to the EventIndex and returns output to user
  - User provide list of events to look for
  - The result can be presented in different formats:
    - simple list
    - verbose list with additional parameters
    - JSON (usefull)
  - Example:

```
$ event-lookup -c short 348895:3613825 --guid-data-type RAW
348895 3613825 78c6837f-f448-e811-ae3c-44a8420a7621
$ event-lookup -c name 348895:3613825 --guid-data-type RAW
348895 3613825 78c6837f-f448-e811-ae3c-44a8420a7621 RAW\
 data18_13TeV.00348895.physics_Main.deriv.DAOD_HDBS3.f937_m1972_p4928
```

  - Can be used by PanDA (Production and Distributed Analysis System):
    - It calls command line tool to perform event lookup and use result as an input

Command line tools

- Trigger tool from the previous EventIndex:
  - User can look for the events that satisfy some trigger conditions (passed some trigger chains)
    - User can specify trigger chain list to search for
    - List of vetoes can be also provided
  - Usually it took some time as the EventIndex had to run Map/reduce job on the huge amount of data
  - The result is the list of events
  - The same tool also provide trigger overlaps matrix
    - Useful for optimizing trigger chains
    - Was running automatically on the new datasets
  - Replaced by the trigger counter service in the new system
    - Runs on the new data during import

Web interface: event lookup

- **Web interface for event lookup query:**

Event Lookup

List of 'runnumber evtnumber' (-e)	00278880 558085589, 00278880 210318172
AMI tag (-p) (substring match)	
Stream name (-s)	Choose a stream name
Data type (-d)	Choose a data type
GUID type (-t)	○ AOD ○ ESD ○ RAW ◉ all
-api	○ simple ○ rich ◉ indexer ○ mc (indexer)
-details	☐ event ☑ type ☐ id ☐ dataset ☐ rich ouput
-email	(implies asynchronous execution)

Search Reset

-e 00278880 558085589, 00278880 210318172
-details typenullnullnullnull
-api indexer

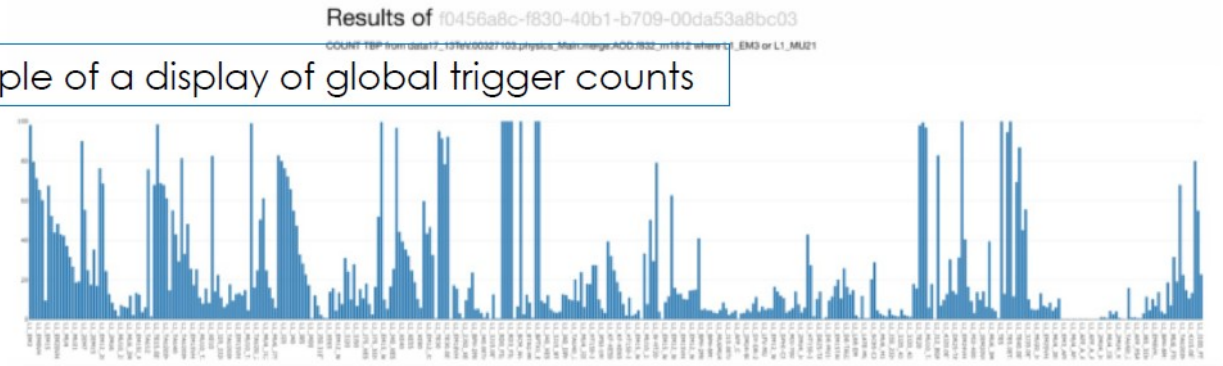
```
6 guids found for 1 runs with 2 events, 0 guids missing, 0s spend
148C774B-B5BB-BF41-9346-F82F213EC4A6 StreamDAOD_HIGG1D1
6C2A1CB8-8F8F-4B42-9FAF-AC55B0957CFD StreamAOD
886C7D3A-3E56-E511-9D00-44A8420A5EB7 StreamRAW
99ACB5FB-11FB-464F-B60F-E496213CABF1 StreamDAOD_HIGG1D1
EBC919C0-BAE8-9F42-BA63-21D782C0FAB7 StreamAOD
FC28C1E6-1D56-E511-AEDE-44A8420A8576 StreamRAW
```

Web interface: Trigger counter

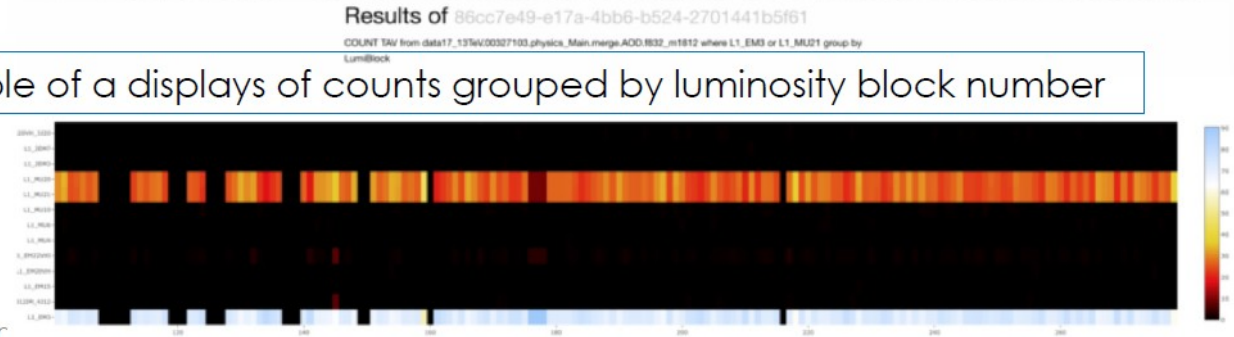
- **Web service able to provide information about the trigger composition of a given dataset.**

The web form to use the Trigger Counter

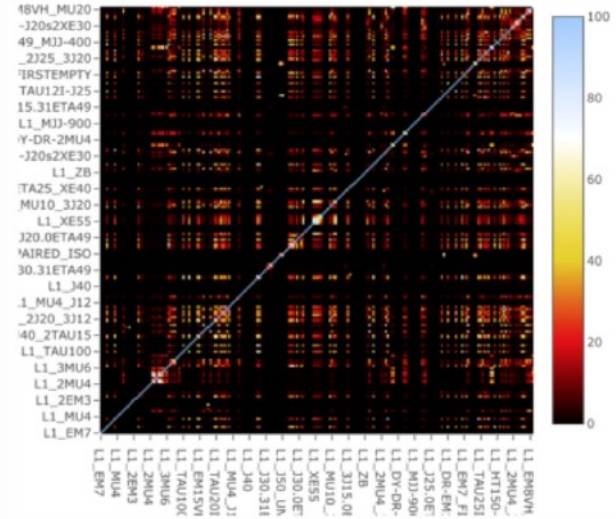
Example of a display of global trigger counts



Example of a displays of counts grouped by luminosity block number



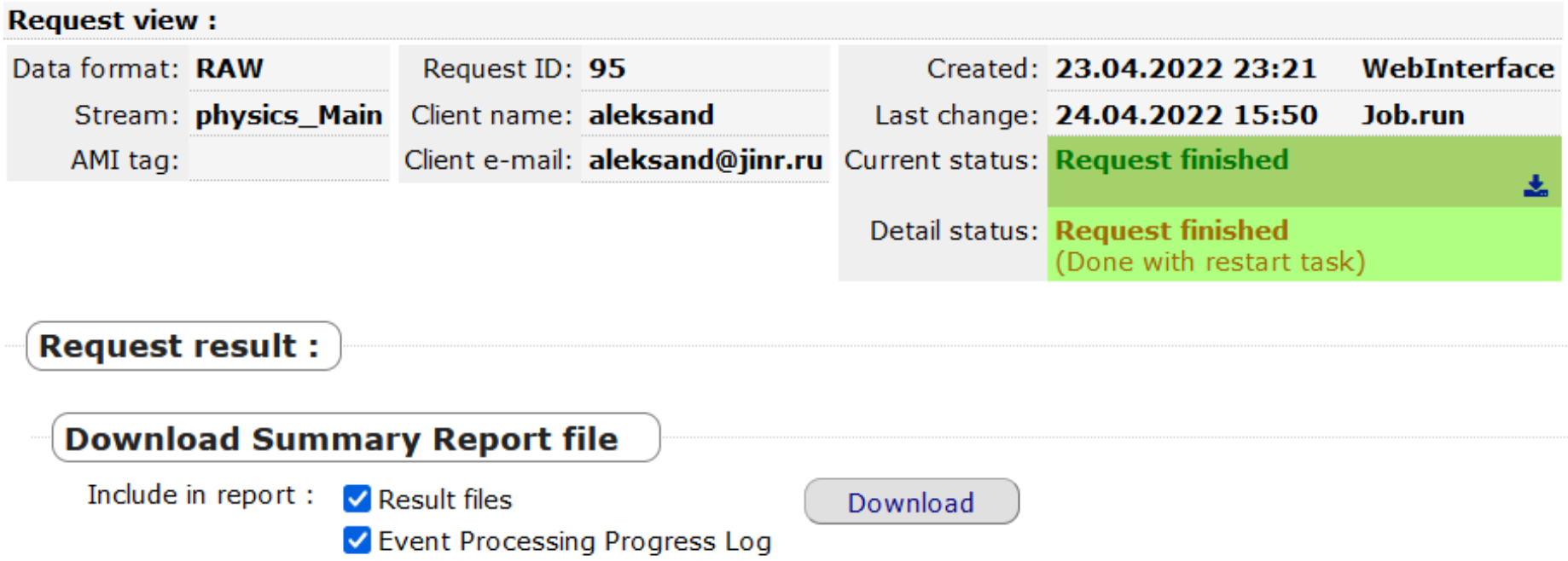
The heat map of trigger overlaps within a given dataset



5 July 20

Web interface: Event picking service

- The GUI takes from the user the event list(s) and additional parameters (data type to be searched, data type to be retrieved, stream, AMI tag, etc.
- The EPS splits the list(s) by run number, queries the EventIndex Query CLI, submits the event picking PanDA jobs, retries them if they time out during tape staging, validates outputs, informs the user when done.
- The user can follow the progress using the GUI.



APIs and interfaces

- Event Index can provide API and/or network interface, depending on the platform that will be used
- Interface or API can be used by client application or by production or information system
- Currently we consider ClickHouse DBMS as a core platform for the EventIndex
- It provides some native interfaces that can be used:
  - HTTP: lets you use ClickHouse on any platform from any programming language in a form of REST API.
  - gRPC: a cross-platform open source high performance Remote Procedure Call (RPC) framework.
    You can write a client in any of the programming languages supported by gRPC using the provided specification.
  - JDBC driver
  - ODBC driver
  - C++ client library

APIs and interfaces

- However it's a common practice to add application layer between database and user client (Three Tier Architecture)
- It is responsible for communicating the user's request to the DBMS system and send the response from the DBMS to the user
- The application layer also processes functional logic, constraint, and rules before passing data to the user or down to the DBMS
- Such approach hav a number of advantages:
  - It allows separate external presentation of data from the internal structure of database, so changes in it will be kept under the hood and will not affect users
  - It improves reliability and security of the system, providing workload management. It makes more difficult for a user to reak something or overload DBMS
- Application level will provide its own API, it can be REST, DBMS or for example, custom python API