Information systems for data taking and data processing

for the SPD experiment

SPD Collaboration Meeting

SPD Experiment Data

- The SPD Experiment have to produce large amounts of data
  - **20** GB/s => ~or **10** PB/year (RAW data), ~**30** billion events per year
- A similar amount of simulated particle collisions for use in detector data analysis will be produced.
- The processing of the experimental data requires a wide variety of auxiliary information from many systems
- Huge numbers of the detector condition and management data should be stored in the databases
  - should be used in every stage of data taking, processing and analysis
  - are essential at nearly every stage of data handling
  - for use in number of versatile applications each with its own requirements

SPD Experiment Data

- Databases can not be considered as the thing in itself
- Databases should be taken as a part of complex **information system** that include
  - Data collection tools
  - Data transportation tools (messaging services, etc)
  - Application layer between the client and the database server, including caching proxies
  - Client software
  - APIs for access from the producton and analysis software
  - Supervisors and monitoring

SPD On-line databases

- Strongly related to the data-taking operations.
- Should be separated from the offline activities that would eventually compromise its availability and performance.
  - Detector and data acquisition configurations
  - On-line filter configurations and additional data
  - Detector calibrations and alignments for on-line filter algorithms
  - Detector conditions data
    - Detector parameters, such as temperatures, voltages, currents, gas flows etc.

Off-line Databases

- Some of the content from the Online databases should be replicated for the use in the offline
- Other information should be gathered from other sources
- Offline condition databases contain
  - Detector geometry and trigger DB
  - Subset of detector conditions data (if necessary)
  - Calibration and alignment constants for offline event reconstruction
  - Conditions metadata
- Used for prompt offline processing, reprocessing, logging of critical operations, etc..

Off-line Databases

- **Physics metadata**. Contain information about
  - datasets and data samples,
  - provenance chains of processed data with links to production task configurations,
  - cross-sections and configurations used for simulations,
  - Offline filter and luminosity information for real and simulated data.
- **Event Index**.
  - Complete catalogue of SPD events, real and simulated data
  - Event IDs, links to data location and event-level metadata
  - Use cases:
    - Event picking (obtain events in the specific format and specific processing version)
    - Production completeness and consistency checks, etc.

Distributed processing and data managemet

- Distributed computing data management and processing book-keeping. Metadata about the data that are processed and stored
- **Distributed Data Management (Rucio):**
  - Dataset contents catalogue: list of files, total size, ownership, provenance, lifetime, status etc.
  - File catalogue: size, checksum, number of events
  - Dataset location catalogue: list of replicas for each dataset
  - Data transfer tools: queue of transfering datasets, status etc.
  - Deletion tools: list of datasets (or replicas) to be deleted, status etc.
  - Storage resource lists, status etc.

Distributed processing and data managemet

- **Production and Distributed Analysis (PanDA)**
  - Lists of requested tasks and their input and output datasets, software versions etc.
  - Lists of jobs with status, running locations, etc.
  - Lists of processing resources with their status etc.
- Both systems use a combination of quasi-static and rapidly changing information
- We expect running few hundreds Kjobs/day, moving hundreds of TB/day every day between storage locations and worknodes

Other

- Monitoring information system
  - Based on time series databases
  - Uses information from other databases, logs, etc.
- Logging and bookkeeping
- Documentation databases (provided by laboratory)
- Hardware database

Database operational and application

- Database clusters built on common base of servers and storages for reducing cost and maintenance efforts serve different type of application workload
- The nature of the distributed computing generates dynamic workload. Coping with high loads on the database side is essential. Three Tier model should be applied when necessary.
- Data sizes and access patterns should be thoroughly evaluated, database layout, objects and quires should be designed according to it, to avoid pitfalls in performance and stability.
- Extensive testing should be performed with appropriate amounts of pseudo data on a separated development DB service before putting in production

Other

- Databases and applications should be designed aiming for scalability, having in mind long term operation in the varying conditions, with different data flows and request rates.
  - Adaptation to the varying data formats and content should be provided.
  - The same with the different versions of client and DAQ software.
- Open source solutions should be preferred
- Development and deployment would take quite a long period, some technologies may become obsolete or not available,others may emerge.
  - Possibility of transfer to the new platform should be kept in mind
  - Use of module design, container-like solution should be encouraged

Conditions Data

- In HEP experiments we use the term Conditions data to refer to non-event data representing the detector status (e.g. calibrations and alignment of the detector sub-components, data taking conditions and similar).
- These data are essential for the processing of physics data, in order to reconstruct events optimally and to exploit the full detector’s potential.
- During data processing these Conditions data will be accessed from the different locations by thousands of jobs in parallel, which requires a solid architecture for their distribution and access.
- Conditions data infrastructure have to deal with the management of several terabytes of data.

Conditions Data Kinds

- Detector hardware conditions:
  - Temperatures, currents, voltages, gas pressures, etc.
  - Sampled frequently and recorded for offline use
- Detector read-out conditions: valid for the duration of 1 Run
  - Trigger and detector read-out configurations
  - uploaded to online filter
- Detector calibrations: valid for periods within Run
  - Energy calibration for calorimeters, time-over-threshold for pixels, etc.
- Detector alignments: valid for the duration of 1 Run
  - Relative and global alignment of sub-detectors

Conditions Data Kinds

- Physics calibrations:
  - Energy scales and resolutions, reconstruction efficiencies
- Luminosity and polarization measurements:
  - Roughly measured during run
  - Precice values based on event reconstruction
- Around 1 TB of condition data expected, based on experience from the similar experiments

Granularity and versioning

- Various pieces of information are heterogeneous both in data type and in time granularity
  - spanning intervals from minutes to hours in duration
- The data should be organized by "Intervals of Validity" (**IOV**), which is the span in time over which that data is valid
- Except for the detector and trigger configurations, which cannot change once they are applied, all other conditions data can be recomputed at any later stage if the understanding of the detector behavior improves or the quality of the input data increases.
- Careful versioning of groups of conditions data for production use cases is a critical item to guarantee reproducibility.

Conditions Data: Rates

- Typically write-rates for conditions data must support of the order of 1 Hz (to ensure good support for several independent systems writing conditions data every minute), with the majority of conditions data being updated much less frequently than this.
- On the other hand, read-rates up to several kHz must be supported for distributed computing workflows, where thousands of jobs needing the same conditions data may start up at the same time.
- Conditions data are typically written once and read frequently.

Condition Data usage and workflows

- Subsystem calibration:
  - conditions determination and testing
  - uploading conditions to the production database
- Online data processing:
  - This is the first level of processing executed by the online filter software to determine which events are selected for offline processing.
  - In this step the event data are processed using the conditions data that were declared as valid in the past.
  - During this step, we do not have the time to recompute new conditions.
  - In this environment, changes are deployed less frequently.
  - Major conditions updates can enter in the system only after the experts have fully checked the data.

Condition Data usage and workflows

- Primary data processing (Express and Bulk):
  - The first offline data processing is executed by computing farms against all data accepted by the online filter system.

The following procedure may be proposed, based on the ATLAS experiment experience:

  - First stage processing where a representative sample (about 10%) of all the data is analyzed.
  - Subsystem experts are given a time window (~ 36 to 48 hours) to study the results and deduce refined calibrations for the bulk processing stage.
  - The bulk processing then commences, processing all of the data using the best known conditions in calibration and alignment at this point.
  - Its output is then ready and usable for offline physics analysis (or refinements to online data taking conditions)

Condition Data usage and workflows

- Reprocessing Campaign:
  - More detailed offline analysis of promptly reconstructed data inevitably reveals further optimizations in conditions as well as improvements in reconstruction algorithms.
  - In reprocessing campaigns, these optimized conditions data are used to re-reconstruct all the data providing improved parameters for physics analysis.
  - These campaigns should reprocess large volumes of data on the distributed centers, sparing central farms for the prompt reconstruction

Condition Data usage and workflows

- User analysis:
  - In some cases, certain types of condition data are needed for specific physics or detector studies which require more detailed conditions than needed in a typical data analysis.
  - Also, physics calibrations (e.g. energy scales) should be available at this level.
  - These data are used typically by all analyses that involve given physics objects.
  - While this processing may not be required on all data streams, even the subset of data required may be quite large.
  - These jobs, like reprocessing, generally should be deployed using distributed resources.

Condition Data usage and workflows

- Isolated processing:
  - A need may arise to execute jobs on external resources, that have their own environment, that requires adaptation of the processes to it
  - For example, worker nodes may not have external connections to conditions DB interfaces, so delivery of conditions via usual means is not available
  - The needed conditions for these jobs must be provided (file based) along with the input event-wise data files.
  - The conditions architecture should include the capability to generate conditions files containing only the conditions expected to be needed for processing.
  - Having this capability would also satisfy use cases such as users running analysis on machines that are not connected to the network

Conditions data with REST

- We consider adopting CREST (Condition data with REST) project developed for ATLAS experiment condition data with strong contribution of JINR developers
- The project should be relatively easy adopted for use with the SPD, its use with PostgreSQL already has been tested
- After that, it should be developed independently from Atlas framework for use with SPD computing environment
- Applying similar system to the On-line databases may be considered

Conditions Data Architecture

- The conditions data payloads are stored in a master database and are accessed using a client-server design through a REST interface, achieving a high degree of separation between client and server.
- This allows for simple clients that should be as agnostic of the rest of the architecture as it is possible to be in order to improve maintainability.
- Due to the read-rate requirements, caching is also extremely important when thousands of jobs start at the same time and require the same conditions data,
- Caching can be provided using web-proxy technologies.

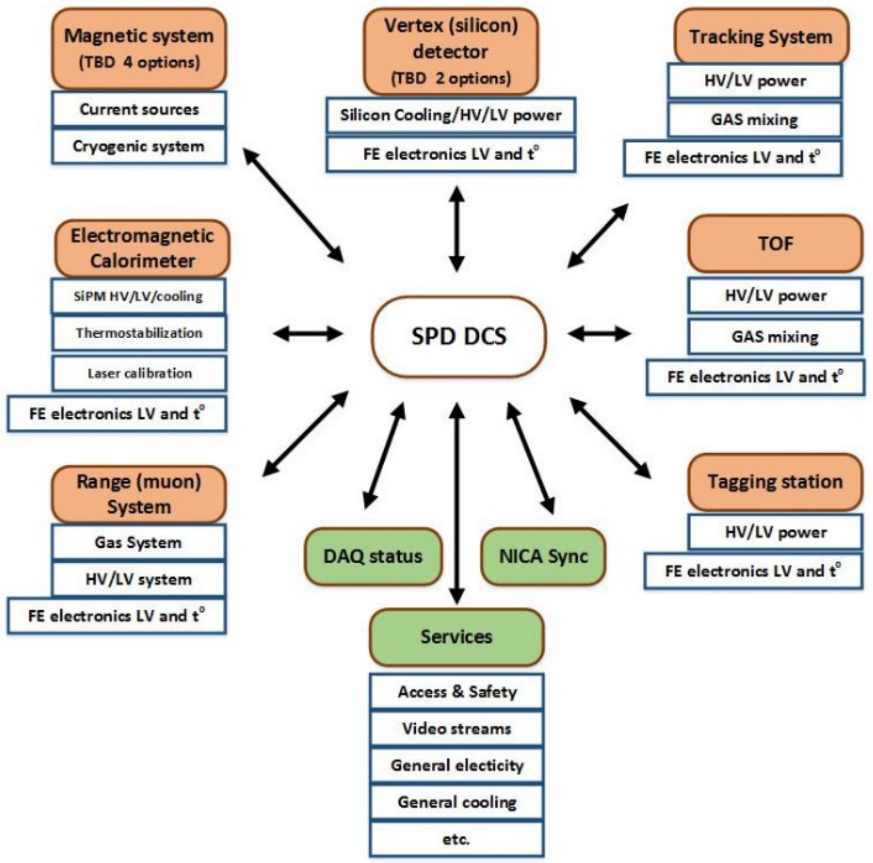
Detector Control System

- The SPD detector control system (DCS) is being designed
  - to control the basic operating modes of the detector parts and the detector as a whole
  - to continuously monitor slowly changing parameters of the detector, engineering means which provide the detector operation, and the environment.
- Provides parameterization of the managed object (sybsystem)
- Implements algorithms for normalization, parameters measurement and control based on these parameters
- Generates the necessary sets of abstractions and options for presenting these abstractions to the operator in an intuitive manner.

Detector Control System

- Parameter values are archived in a database for long-term monitoring of the detector operation and identify possible failures in the operation of the equipment and emergency situations.
- The configurations of the detector parameters saved in the database make it possible to start the detector promptly and use it with various preset parameters and in various operating modes in accordance with the requirements of a particular physics experiment.
- The number of parameters in the system is expected to be significant, therefore, it is assumed that the system should be extendable and flexibly configurable.

Detector Control System



- Implementation of the DCS have to be developed.

  An input from the subsystems required for undemanding of size and design of the DCS
- Just a few numbers:
- DSSD: 368640 x 5 + 400 channels
- ECAL: 26112 x 2 + 3067 ch.
- RS: 106000 x 2 + 629 ch.
- ZDC: ~3000 ch.

Detector Hardware database

- A catalog of hardware components that SPD detector consist of.
- It should contain the information about the detectors and the electronic parts, cables, racks, and crates, as well as the location history of all items
  - It include equipment models, provider, paramenters and other (semi)permanent characteristics
- This should help in maintenance of the detector systems and especially helpful in knowledge transfer between team members.

ATLAS EXPERIMENT

Cables: **83529**
Crates: **7714**
Equipment: **167491**
Racks: **929**
Boards: **11534**
Channels: **26585**
Cable Trays: **1360**
Zones: **21**
Detector Parts: **1618**
Equipment Models: **1984**
Measurements: **10413**

The Event Index

- EventIndex is a system designed to be a complete catalog of SPD events, real and simulated data
- Event 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 (RAW, AOD)
  - Various versions originating from the same detector or simulated events
    - Different reconstruction settings and software version
- EventIndex
  - provides a way to collect and store event information
  - provides various tools to access this information through command line, GUI and RESTful API interfaces

The Event Index

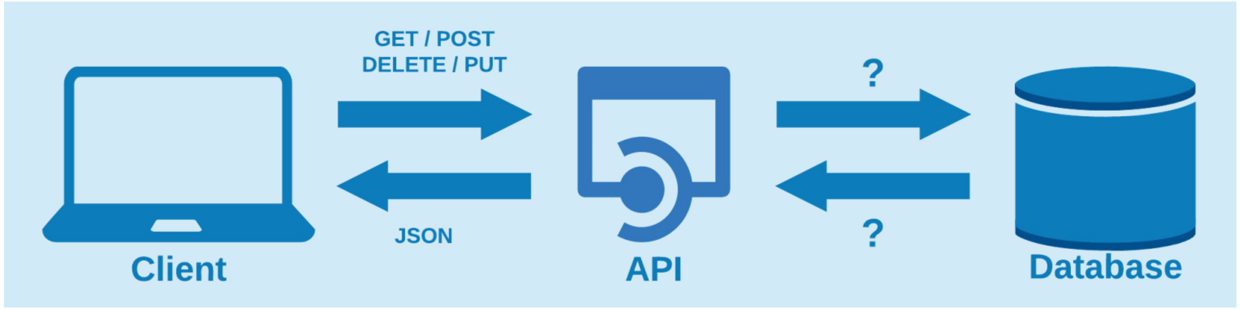
- Event Index records should contain
  - Event identifiers: Run and event number
  - On-line filter decisions: encoded as bit masks
  - References to the events at each processing stage in all permanent files generated by central productions
  - Important parameters of the event useful for pre-selection for analyses
- Event Index use cases:
  - Event picking: selecting of events in some format and processing version by run and event numbers
  - Counting and selection if events based on on-line fliter decisions and parameters
  - Production completeness and consistency checks

The Event Index

- A PostgreSQL - based back-end is being tested now
  - A lists of generated event pseudo-data used for test

```
 run_number | event_number | olt_result | dsid_raw |               fuid_raw
------------+--------------+------------+----------+--------------------------------------
  280308061 |            8 |      29721 |        1 | 3ad87450-23e6-4b72-afcb-077b5184577e
  280308061 |           14 |      24218 |        1 | 3ad87450-23e6-4b72-afcb-077b5184577e
```

- A simple RestAPI using flask framework have been developed
  - Currently it makes preprocessing of the input data for the event picking provided by the web-GUI, performs SQL requests to the events table and returns list of file UIDS



The Event Index

- A Simple GUI using Angular framework has been created to test RestAPI with event picking requests
  - It allows request one or few events manually or many events from the list
- A testing of various ways to increase data ingestion rates is in progress
  - Using bulk loading via various interfaces, like asyncpg
  - Checkpoint optimization, unregistered tables, switching of triggers, column order optimization, etc
- A modular test environment is being created for testing different approaches with the similar data and measuring tools

SPD Needs Your Input

- As it was mentioned before, Information systems should be tailored to the needs of the project and to the nature and amount off data
- We need input both from hardware and analysis groups to create information systems fitting their need
- Information is needed BEFORE the databases design will be ready

SPD

TO PROVIDE INPUT
FOR THE DATABASES

BACKUP

Data

- Событие - Наименьшая единица набора данных, результат полученный со всех субдетекторов во время одного столкновения сгустков частиц в LHC.
- Run - это интервал времени сбора данных, обычно много часов, в течение которого записываются данные, при этом конфигурация детектора и триггерных систем постоянна.
- The event records, are grouped into files that contain a few thousand events.
- Files containing statistically equivalent events taken under the same conditions are grouped into datasets.
-

Conditions Data

- Various versions of the datasets originate from the same events
- Different reconstruction settings and software version
- Datasets have different data type for every processing stage

SPD Data formats

- Real data from the detector:
  - 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
- 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 Data formats

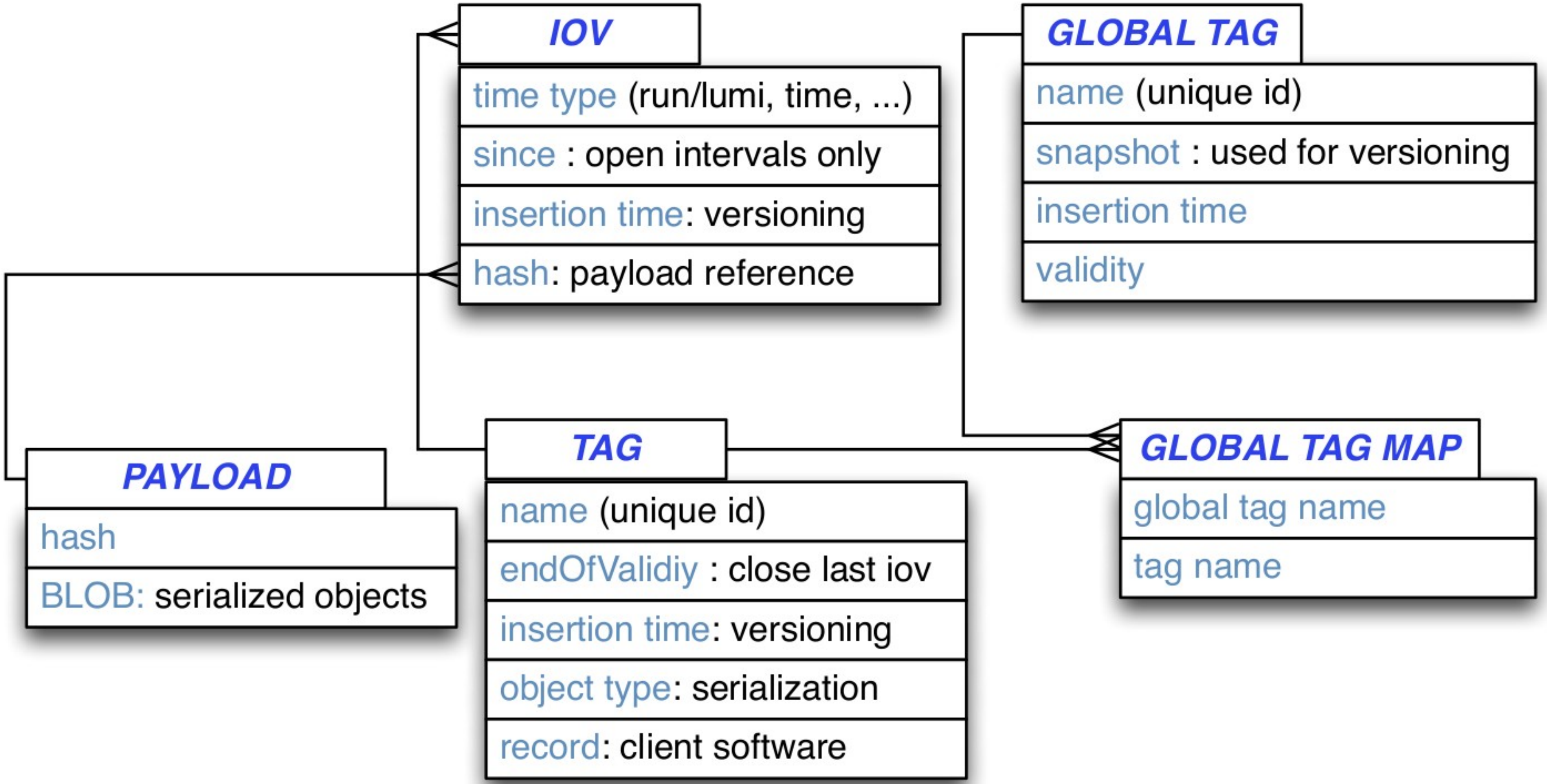
- Each event contains result of interactions within one time slice
- Event content
  - Event header: Run ID, FrameID, BCID, Event ID
  - RAW Hits
  - List of Vertices: Vtx,y,z + list of particles
  - Particles
    - ParticleID, list of tracks, list of ECAL, RS, ZDC clusters, TOF, AEG, BBC hits + MC truth 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

For AOD and ESD

Condition data mdel Data Model

- A global tag is the top-level configuration (version) of all conditions data needed for a particular data processing use case.
- For a given system and a given interval of validity, a global tag will resolve to one, and only one, conditions data payload.
- The Global Tag resolves to a particular subsystem Tag via the Global Tag Map table.
- A subystem Tag consists of many intervals of validity (IOV) or entries in the IOV table that corresponds to a time interval.
- Finally, each entry in the IOV table maps to a payload via its unique hash key in the Payload table.

Other



IOV
time type (run/lumi, time, ...)
since : open intervals only
insertion time: versioning
hash: payload reference
GLOBAL TAG
name (unique id)
snapshot : used for versioning
insertion time
validity
PAYLOAD
hash
BLOB: serialized objects
TAG
name (unique id)
endOfValidiy : close last iov
insertion time: versioning
object type: serialization
record: client software
GLOBAL TAG MAP
global tag name
tag name

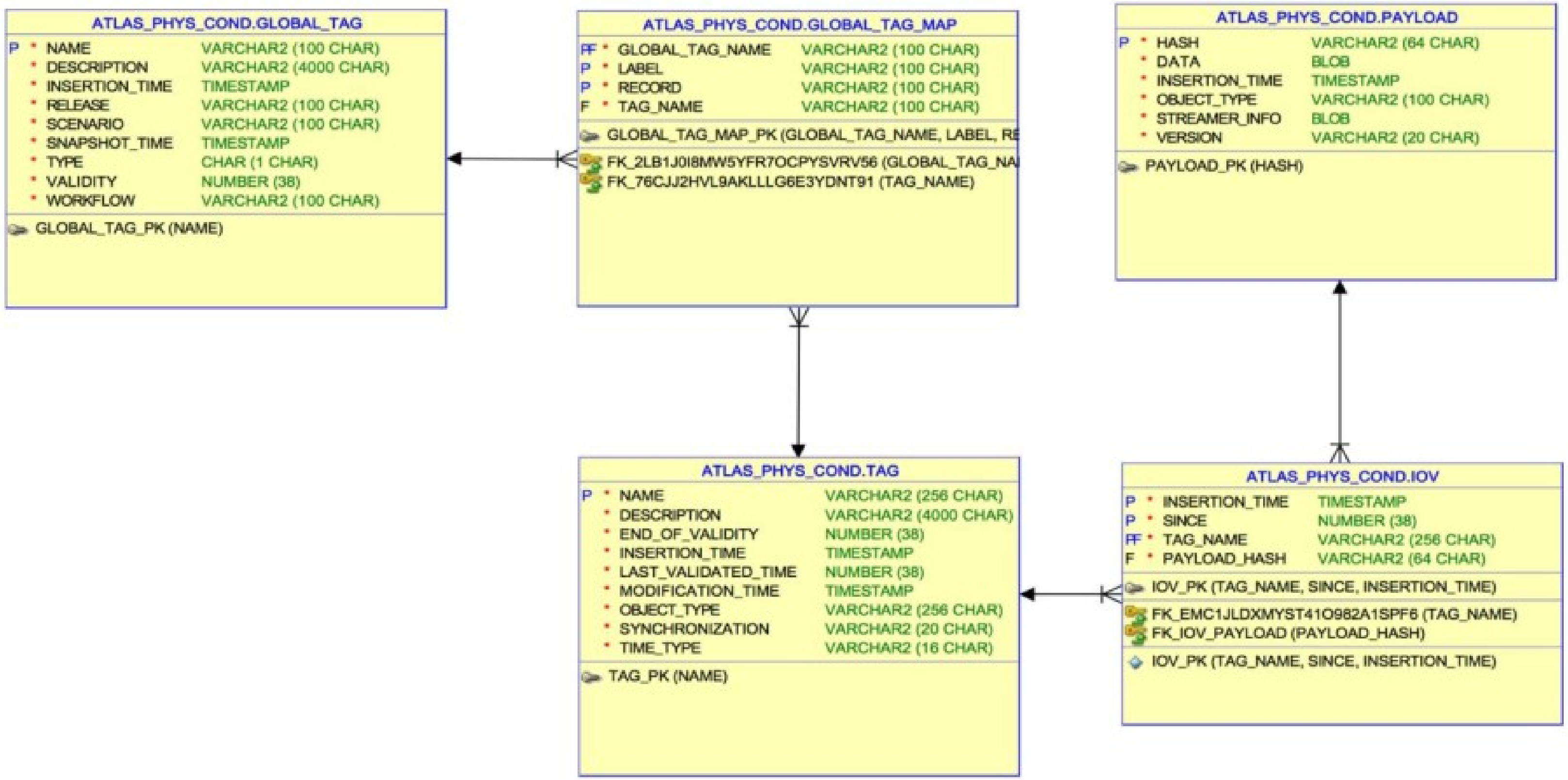
Data Model Features

- Conditions data payloads are uniquely identified by a hash which is the sole reference to any given conditions data payload.
- The payload data has been separated from the data management metadata and could in principle be placed in a separate storage system.
- IOVs are resolved independently of payloads that are cacheable
- Efficient caching is a key design requirement for any conditions database that must support high rate data access.
- A web server (with an internal cache) is used for interactions with the relational database and data input, search and retrieval

Conditions data with REST

- We plan to adopt CREST (Condition data with REST) project developed for ATLAS experiment condition data
- It implements model described above, with five tables containing metadata and payload data
- Conditions data are stored in the PAYLOAD table.
- Values are consumed as an aggregated set
  - typically a header and some parameters container(s).
- Conditions meta-data are organized in three tables
  - plus one used essentially for mapping between tags and global tags
- Frontier servers and Squids provide access from Grid jobs and local caches

CREST data model



ATLAS_PHYS_COND.GLOBAL_TAG

P	*	NAME	VARCHAR2 (100 CHAR)
	*	DESCRIPTION	VARCHAR2 (4000 CHAR)
	*	INSERTION_TIME	TIMESTAMP
	*	RELEASE	VARCHAR2 (100 CHAR)
	*	SCENARIO	VARCHAR2 (100 CHAR)
	*	SNAPSHOT_TIME	TIMESTAMP
	*	TYPE	CHAR (1 CHAR)
	*	VALIDITY	NUMBER (38)
	*	WORKFLOW	VARCHAR2 (100 CHAR)

GLOBAL_TAG_PK (NAME)

ATLAS_PHYS_COND.GLOBAL_TAG_MAP

PF	*	GLOBAL_TAG_NAME	VARCHAR2 (100 CHAR)
P	*	LABEL	VARCHAR2 (100 CHAR)
P	*	RECORD	VARCHAR2 (100 CHAR)
F	*	TAG_NAME	VARCHAR2 (100 CHAR)

GLOBAL_TAG_MAP_PK (GLOBAL_TAG_NAME, LABEL, RE

FK_2LB1J0I8MW5YFR7OCPYSVRV56 (GLOBAL_TAG_NA

FK_76CJJ2HVL9AKLLLG6E3YDNT91 (TAG_NAME)

ATLAS_PHYS_COND.PAYLOAD

P	*	HASH	VARCHAR2 (64 CHAR)
	*	DATA	BLOB
	*	INSERTION_TIME	TIMESTAMP
	*	OBJECT_TYPE	VARCHAR2 (100 CHAR)
	*	STREAMER_INFO	BLOB
	*	VERSION	VARCHAR2 (20 CHAR)

PAYLOAD_PK (HASH)

ATLAS_PHYS_COND.TAG

P	*	NAME	VARCHAR2 (256 CHAR)
	*	DESCRIPTION	VARCHAR2 (4000 CHAR)
	*	END_OF_VALIDITY	NUMBER (38)
	*	INSERTION_TIME	TIMESTAMP
	*	LAST_VALIDATED_TIME	NUMBER (38)
	*	MODIFICATION_TIME	TIMESTAMP
	*	OBJECT_TYPE	VARCHAR2 (256 CHAR)
	*	SYNCHRONIZATION	VARCHAR2 (20 CHAR)
	*	TIME_TYPE	VARCHAR2 (16 CHAR)

TAG_PK (NAME)

ATLAS_PHYS_COND.IOV

P	*	INSERTION_TIME	TIMESTAMP
P	*	SINCE	NUMBER (38)
PF	*	TAG_NAME	VARCHAR2 (256 CHAR)
F	*	PAYLOAD_HASH	VARCHAR2 (64 CHAR)

IOV_PK (TAG_NAME, SINCE, INSERTION_TIME)

FK_EMC1JLDXMYST41O982A1SPF6 (TAG_NAME)

FK_IOV_PAYLOAD (PAYLOAD_HASH)

IOV_PK (TAG_NAME, SINCE, INSERTION_TIME)

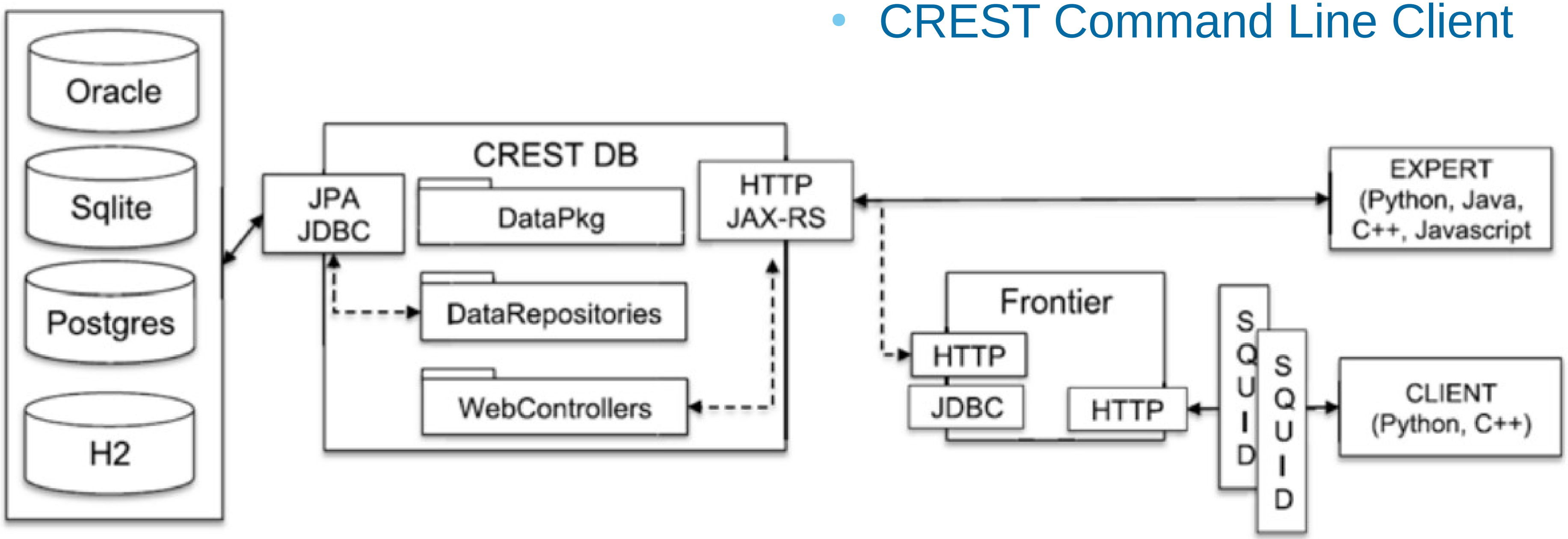
CREST data model scheme

CREST data model

- IOV in CREST contains the time information
  - stored in one time column
  - time can be represented as a timestamp, a run number etc.
  - it is valid by default until the next entry in time
- IOV points to one payload via an sha256 hash key
  - The IOVs are retrieved separately from the PAYLOADS
- TAG in CREST is a label used to identify a specific set of IOVs.
- GLOBAL TAG is a label used to identify a consistent set of TAGs, involved in a given data flow
  - e.g. a reprocessing campaign, a MC production etc.
- The same TAG can be associated to many GLOBAL TAGs.

CREST Database Architecture

- The CREST project consists of several components:
  - CREST Server
  - C++ Client library
  - CREST Command Line Client



CREST Server

- CREST DB server exposes functionalities via REST.
  - The API is described using OpenAPI specifications.
  - SQL is not involved in the dialogue between client and server,
  - Instead the internal resources are accessible via URLs.
- All HTTP Methods can be used:
  - POST/PUT (to create/update resources), GET and DELETE.
- The request and response bodies are formatted in JSON.
- The header of the requests can be used for formatting the output, deal with caching related parameters etc.
- The client library (in Python) and server stubs (in Java JAX-RS) are generated via OpenApi by the Swagger Codegen library.

C++ CREST Client library

- C++ client library (CrestApi) created to simplify the CREST Server access to the C++ developers and the Athena framework
- CrestApi library is a request library to the CREST Server
  - or to the local file storage
- Allows to store, read (and update) the data on the CREST Server.
- The data transferring mechanism (low level request methods) can be changed in the CrestApi library. Now it uses the CURL library
- Data exchanged with the server are in JSON format.

CREST Command Line Client

- The CREST command line client (CrestCmd) is an access tool to the data stored on the CREST server
  - Written to simplify the development of the other CREST project components.
- CrestCmd can be used for quick interactions with the CREST server, mainly with the goal to provide management functionalities and browsing capabilities to users.
- CrestCmd works with the main CREST data types
  - tags, tag meta infos, global tags, global tag maps and an IOVs together with payloads.

CREST

- The CREST project appeared to replace previous Conditions DB based on COOL/CORAL, that had a number of serious drawbacks.
- The CREST project server prototype and its main components are implemented.
- The CREST C++ client library (CrestApi) was written and included in the official ATLAS software offline release (Athena).
- The project should be relatively easy adopted for use with the SPD, its use with PostgreSQL already has been tested
- After that, it should be developed independently from Atlas framework for use with SPD computing environment

Выбор платформы

- Разработку EventIndex естественно начать с выбора платформы для хранения и управления данными.
- Нужна Open Source СУБД позволяющая быстро и надежно оперировать терабайтными объемами данных
- Рассматриваются несколько вариантов
  - ClickHouse от Яндекс
    - столбцовая СУБД для онлайн обработки аналитических запросов (OLAP)
  - Apache HBase
    - распределенная, колоночно-ориентированная, мультиверсионная база типа «ключ-значение».
  - PostgreSQL
    - объектно-реляционная система управления базами данных (СУБД)

Особенности ClickHouse

- Столбцовое хранение данных — данные считываются только из нужных колонок, и однотипная информация эффективно сжимается
- Поддержка приближённых вычислений на части выборки — снижается число обращений к жёсткому диску, что ещё больше повышает скорость обработки данных;
- распараллеливание операций как в пределах одного сервера на несколько процессорных ядер, так и в рамках распределённых вычислений на кластере за счёт механизма шардирования;
- линейная масштабируемость — есть возможность построить кластер очень большого размера

Недостатки ClickHouse

- Разреженный индекс делает ClickHouse плохо пригодным для точечных чтений одиночных строк по своим ключам.
- Возможность изменять или удалять ранее записанные данные с низкими задержками и высокой частотой запросов не предоставляется. Есть массовое удаление и изменение данных для очистки более не нужного
- Отсутствие уникальности первичных ключей, что вместе с предыдущим пунктом в нашем случае осложняет добавление новых полей в EventRecord

Apache HBase

- Данные организованы в таблицы, проиндексированные первичным ключом, который в Hbase называется RowKey.
- Для каждого RowKey ключа может храниться неограниченны набор атрибутов (или колонок).
- Колонки организованны в группы колонок, называемые Column Family.
  - Как правило в одну Column Family объединяют колонки, для которых одинаковы паттерн использования и хранения.
- Для каждого атрибута может храниться несколько различных версий. Разные версии имеют разный timestamp.

Apache HBase

- Записи физически хранятся в отсортированном по RowKey порядке. При этом данные соответствующие разным Column Family хранятся отдельно, что позволяет при необходимости читать данные только из нужного семейства колонок.
- Атрибуты, принадлежащие одной группе колонок и соответствующие одному ключу физически хранятся как отсортированный список. Любой атрибут может отсутствовать или присутствовать для каждого ключа, при этом если атрибут отсутствует — это не вызывает накладных расходов на хранение пустых значений.
- Список и названия групп колонок фиксирован и имеет четкую схему.

PostgreSQL

- PostgreSQL это а объектно-реляционная СУБД с открытым исходным кодом.
  - Поддержка многочисленных типов данных
    - Численные, булевые, символьные, составные, сетевые типы данных, перечисление, типы «дата/время», массивы, etc.
  - Поддержка JSON, что позволяет использовать schema-less данные
    - Встроенные специализированные JSON операторы и функции
  - Поддержка пользовательских объектов и их поведения, включая типы данных, функции, операции и индексы.
  - Индексирование: частичное, функциональное, GiST, GIN, BRIN
  - Функции виртуальных таблиц, Материализованные представления
  - Возможность добавления/изменения столбцов

PostgreSQL

- Обеспечивает высокую надежность и производительность:
  - соответствие принципам ACID (атомарность, изолированность, непротиворечивость, сохранность данных)
  - Многоверсионный контроль конкурентных транзакций и изоляция транзакций
    - возможность изменения баз данных одновременно разными пользователями.
    - минимизирует блокировки данных и позволяет увеличить производительность
  - Журналы опережающей записи (**W**rite **A**head **L**ogging)
    - фиксирующая все изменения до их фактического применения.
  - Резервное копирование и восстановление
  - Возможность восстановления базы данных Point in Time Recovery
    - Откат состояние базы данных к предыдущему стоянию используя WAL.

- Основные ограничения

Максимальный размер БД	Неограничен
Максимальный размер таблицы	32 TB
Максимальный размер строки	1.6 TB
Максимальный размер поля	1 GB
Максимальное количество строк в таблице	Неограниченно
Максимальное количество столбцов в таблице	250-1600
Максимальное количество индексов в таблице	Неограничено