ArchiveDB

Scientific and Technical Data Archive for Wendelstein 7-X

Christine Hennig

Wendelstein 7-X CoDaC* team

***Control, Data Acquisition and Communication**

Outline

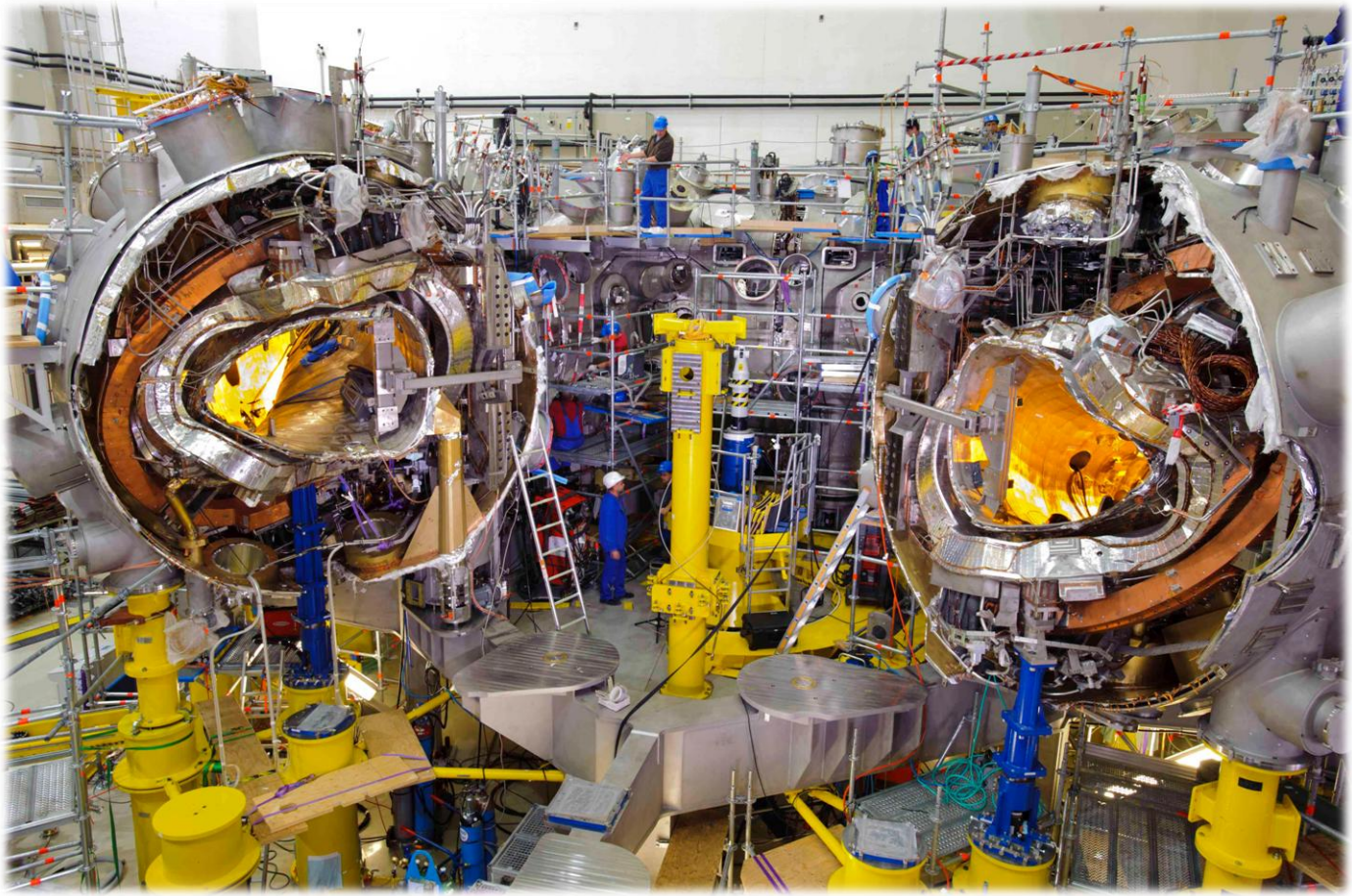
Status of W7-X Machine commissioning and CoDaC (6 slides)

ArchiveDB (18 slides)

- Is this Big Data?
- Internal structure
- Underlying technique
- User API
- Lessons learned

Integration of unsupported diagnostics (4 slides)

Stellarator Wendelstein 7-X (W7-X)

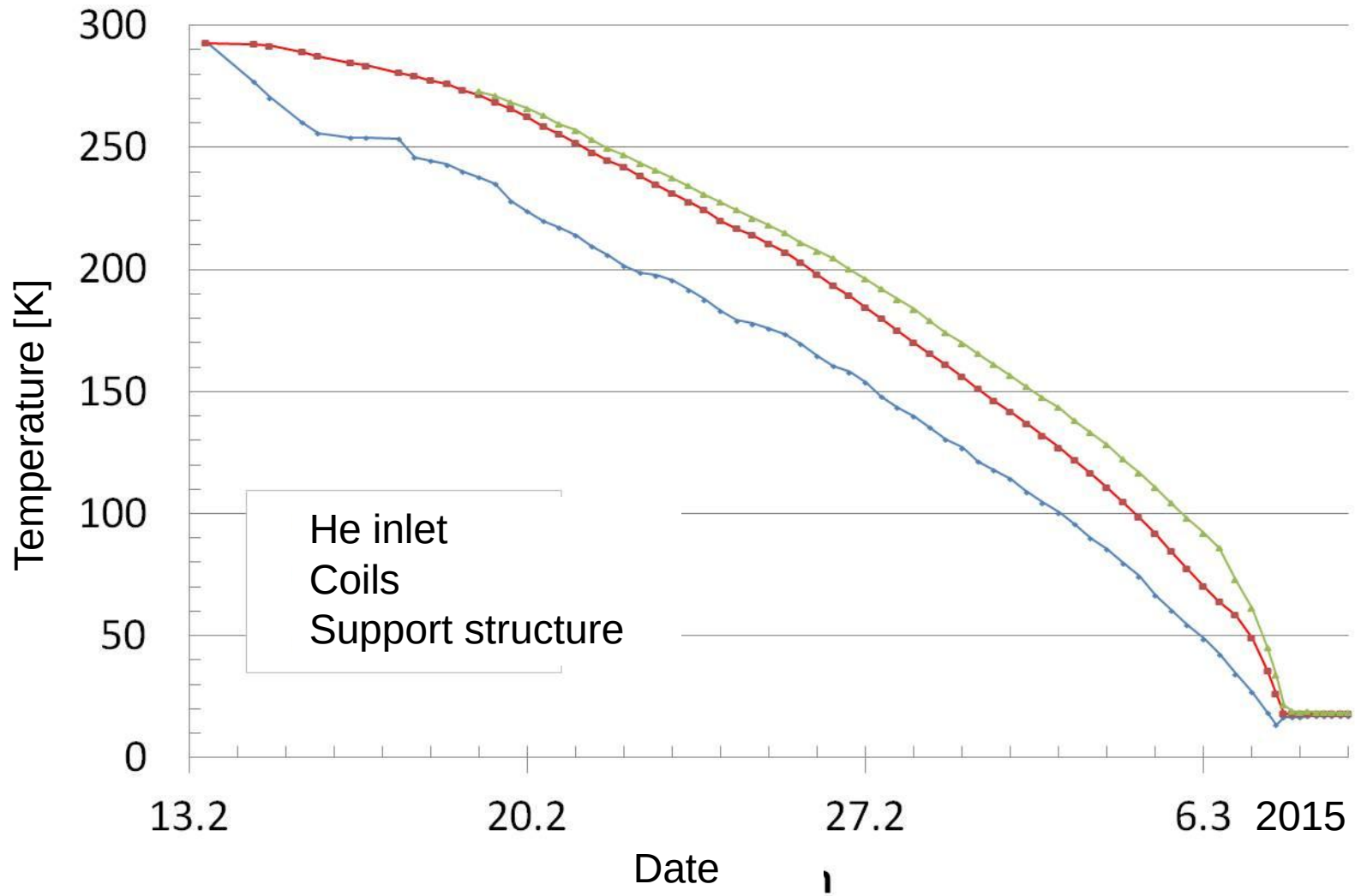


Two years ago…

April 2015: Evacuation of plasma vessel started

Commissioning: Cooldown reached

Cooldown of superconducting coils (< 4K) reached: 12 Mar 2015 (data from ArchiveDB)



Status W7X commissioning (1)

Plans for first plasma, revised

1.

- first plasma operation phase (OP1.1)
- in 2015
- about 3 months
- first plasmas with limiter (five graphite limiter stripes)

2.

- de-installation of the limiter
- installation of the test divertor units (TDU)
- completion of the installation of other plasma facing components
- about one year

3.

- plasma operation phase with the TDU (OP1.2)
- in 2016

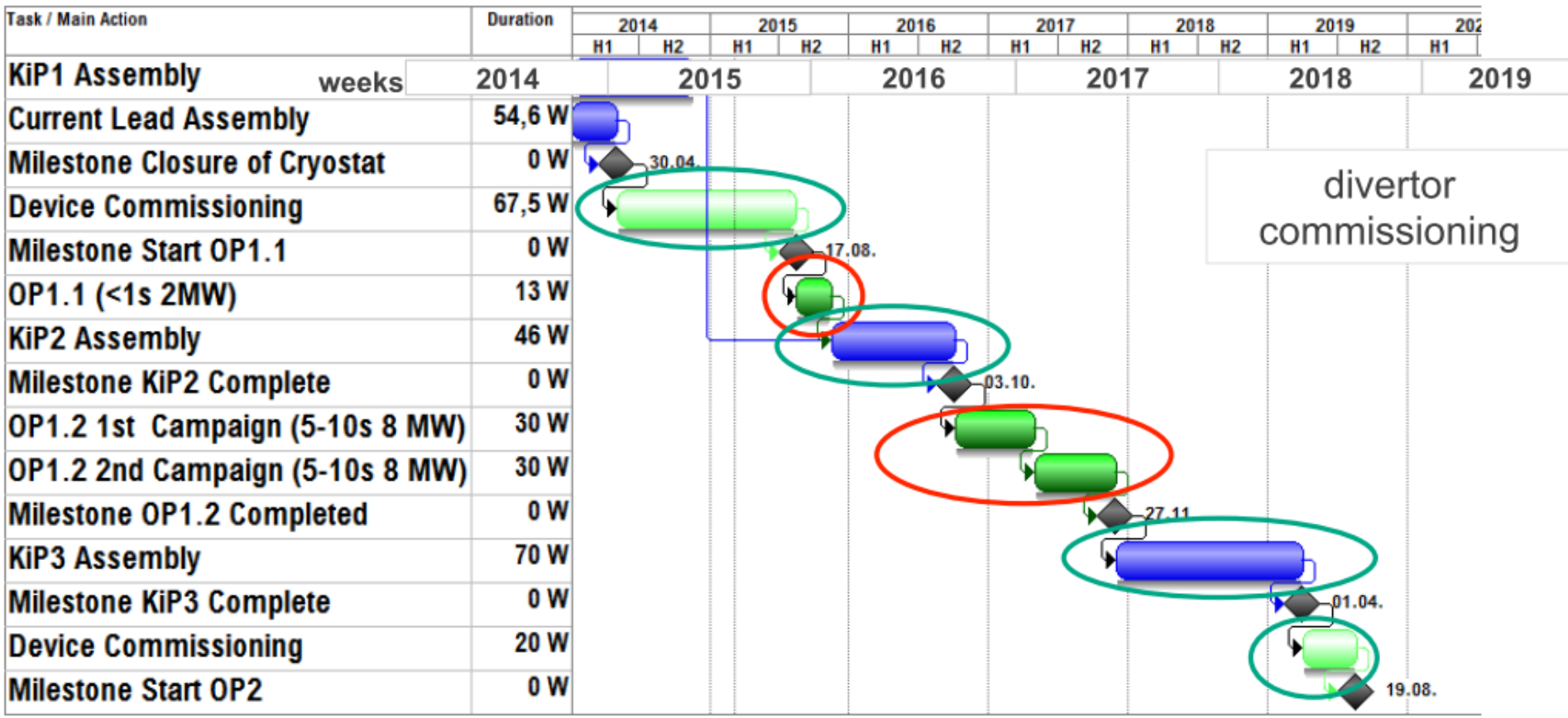
➔ Wendelstein 7-X is rapidly approaching first plasma operation.
➔The first plasmas will be created with a reduced set of in-vessel components.
➔Start with limiter instead of divertor

Status W7X commissioning (2)

Device commissioning (cryostat, fields, vacuum)

Test divertor installation

HHF divertor installation



1st plasma w/o divertor
OP 1.1: This talk

1st divertor plasmas
OP 1.2

steady-state plasmas
OP 2

Key requirement continous operation

New challenges due to continuous operation:

- Automated (complex) data analyses
- Intelligent plasma control: stabilization of favored plasma states
- Continuous high performance data rates
- High quality software products

Continuous operation is a paradigm change!

from “shot and collect” (traditional shot based fusion experiments)

towards “view and react”

➔Implications for CoDaC system and data archive

CoDaC status

- Many changes in software (and hardware) since using Wega as testbed
- W7-X commissioning will be also CoDaC testbed
- Continuous experiment operation is not foreseen in OP1
- Machine diagnostics run round the clock 24*7*365
- Physics Diagnostics come late and all at the same time ➔ we can't integrate all diagnostics before OP1.1
  - ➔ Priority list "A" and "B"
  - ➔ All "A" diagnostics will be integrated
  - ➔ "B" Diagnostics will be integrated later (some will use other co/dac systems in the meantime)
- Pragmatic solutions

Big Data – Big Five

Volume (Data at Rest)
Terabyte … Exabyte

Velocity (Data in Motion)
Streaming data: msec … sec to respond

Variety (Data in many forms)
Structured,
Unstructured,
Multimedia,
Text…

Veracity (Data in doubt)
Uncertainty,
Inconsistency,
Ambiguity,
Incompleteness,
Latency

Value
„Big Data is the new Oil“

Volume
1,4 PByte/year

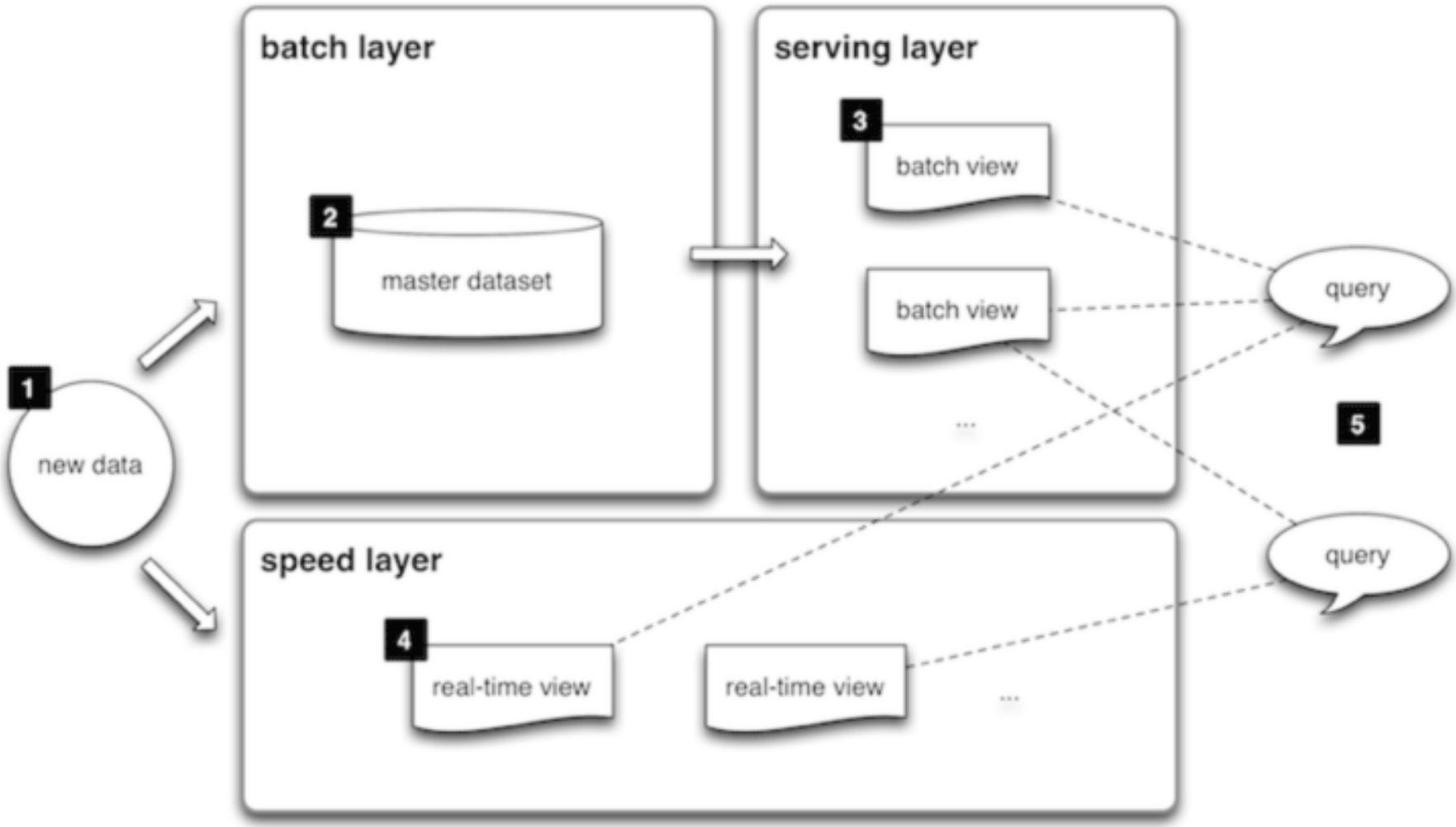
Velocity
30 GByte/sec

Variety
Tree structured (parameter)
Sensor data: 2dimensional,
Images,
Video
Experiment execution data, user comments

Veracity
Measuring errors
Defect cables/sensors …
Indirect measurements

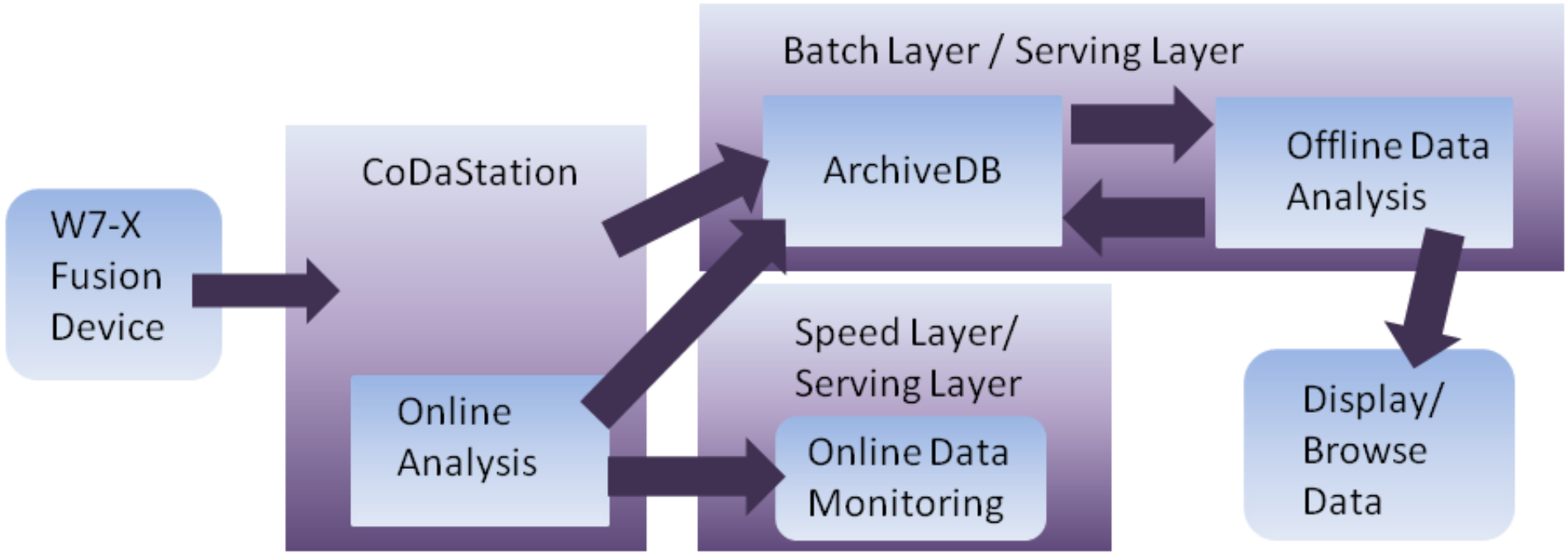
Value
Experiment is non reproducible
Machine is unique

Lambda architecture pattern



graphics: http://www.datasciencecentral.com/profiles/blogs/lambda-architecture-for-big-data-systems
Book: Nathan Marz and James Warren
“Big Data: Principles and best practices of scalable realtime data systems”
Softbound print: April 2015 (est.)

Lambda architecture pattern



Architecture first proposed by P. Heimann et al.

(Fusion Engineering and Design 71 (2004) 219–224: Status report on the development of the data acquisition system of Wendelstein 7-X)

Previous IAEA TMs: talks and papers about

- CoDaStation including Online Analysis (T. Bluhm)
- Monitoring (Ch. Hennig)
- DataAccess/DataBrowser (T. Bluhm)

This conference: paper citing related work (BigData)

Change in underlying technique

Transition from object data base to plain files

- Initial version used the object database ObjectivityDB
- Object store for both (data and parameter) boxes and (time) index data
- Productive from 2005 to 2014
- ObjectivityDB maintains the schema of all data (Java classes schema)
- Objectivity supports schema migration
- Object schema is relatively rigid and needs 1:1 mapping of our business classes

Transition from schema based to schema less

- Revised the storage of the structured parameter data
- Storage is now schema-less
- Each structured parameter item covers both its values and its self-contained schema

Change in underlying filesystem

Use of a distributed file system for horizontal scaling

- File based archive solution ➔ file system plays a major role in fulfilling the scalability requirements
- Distributed scalable file system General Parallel File System (GPFS)
- Distribution of data between multiple archive servers both for reading and writing.
- For every Archive Stream a single writer instance writes on a dedicated server.
- Multiple readers can read from multiple servers
- Data is distributed as files
- Data is available to all potential readers very shortly after storage
- Horizontal scaling (scale out: add more servers).
- Connecting GPFS to the High Performance Storage System (HPSS) using GHI
- ➔ “Unlimited” data store: Data can be on disk, on tape or both

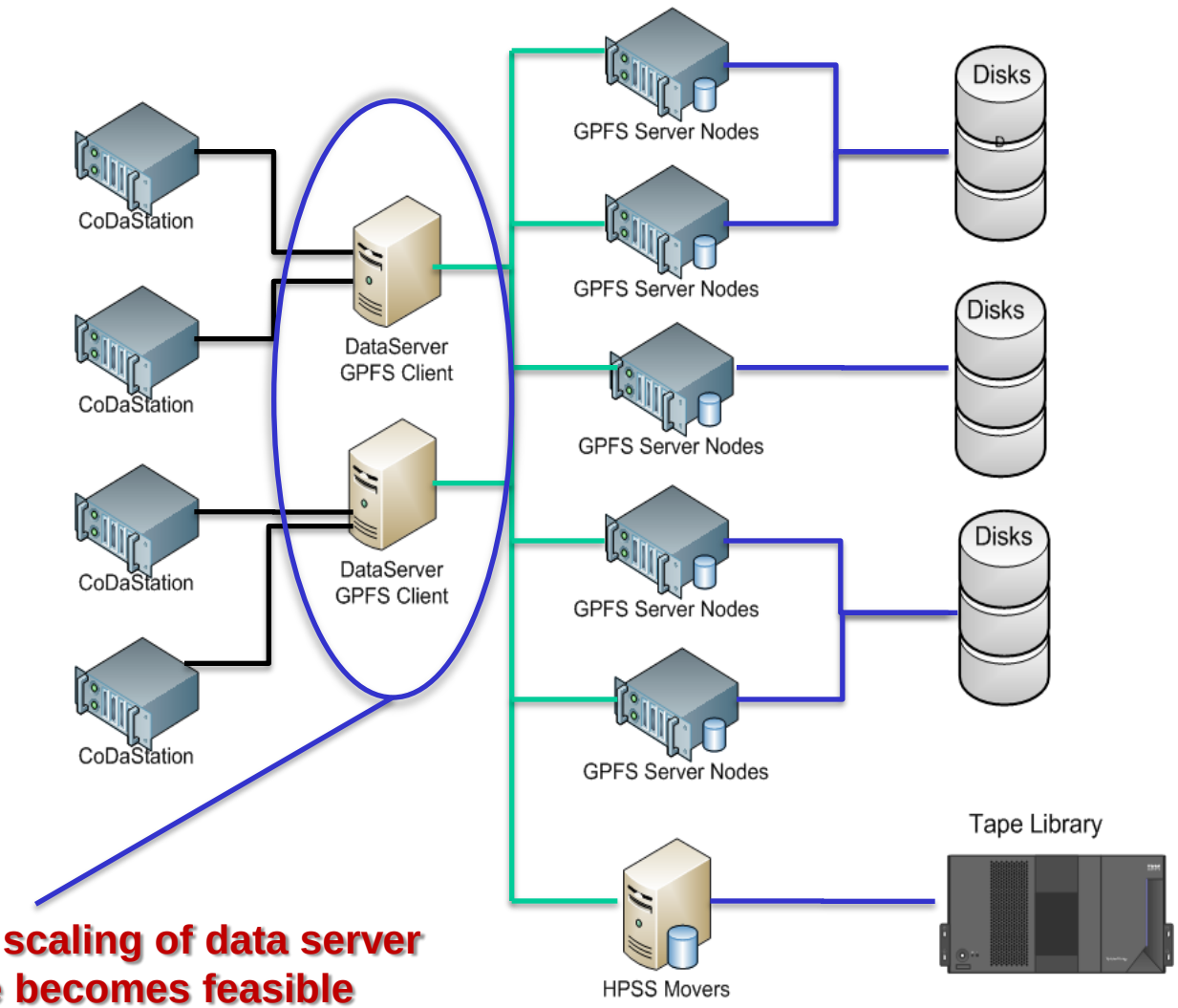
horizonal scaling

DAQ station (CoDaStation) data streaming to data server (decoupling)
25 servers

Data server with parallel filesystem (IBM GPFS)
11 Servers

Data migration to tape with HPSS (IBM)
Based on policies

Daily growth: ~1.7 TByte/day of W7-X commissioning data reaches ArchiveDB



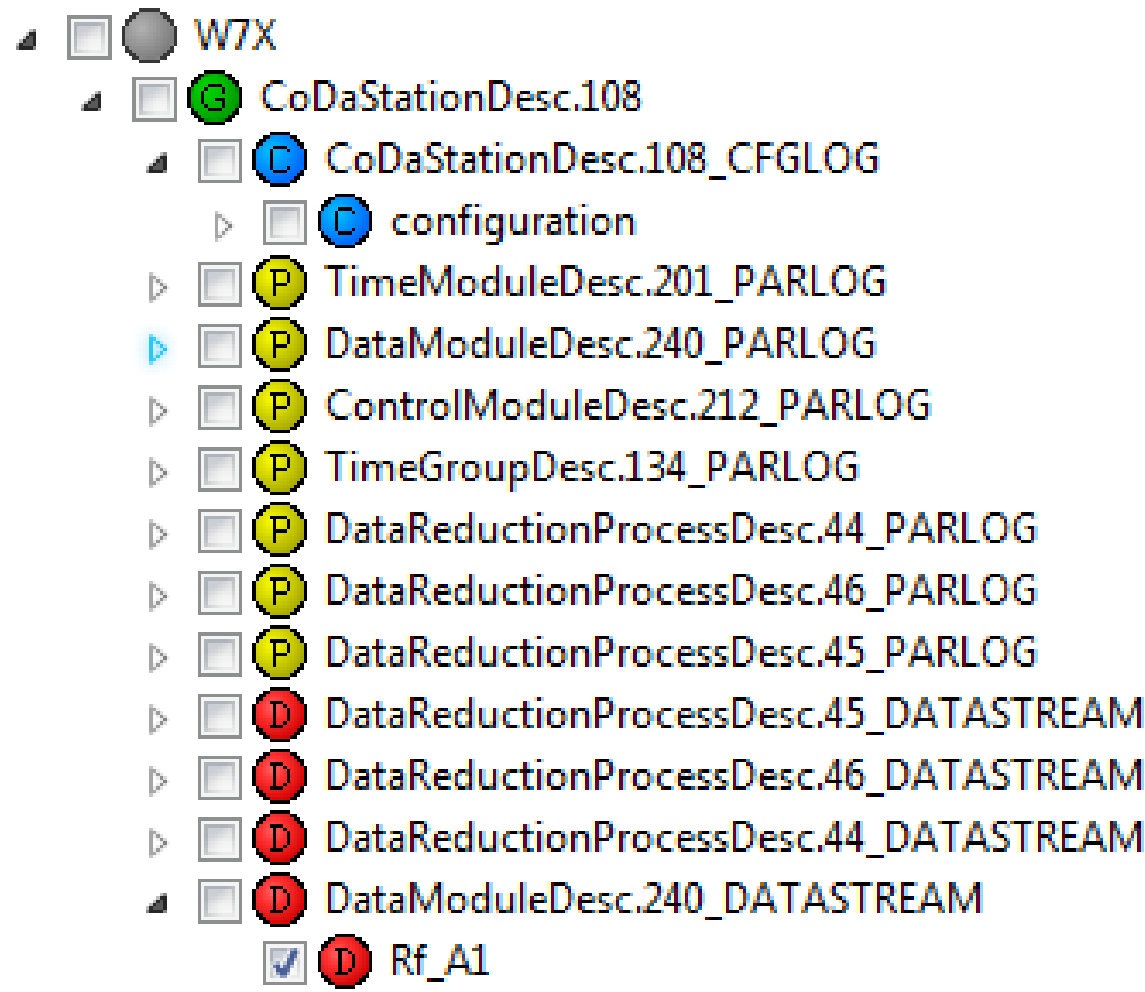
configuration less scaling of data server and fault tolerance becomes feasible

Partitioning of data (groups, streams)

Archive Stream Group: contains all data of a data acquisition system (e.g. CoDaStation)

Separated in Archive Streams

Easily sums up to 20 to 1000 Archive Streams per group



Screenshot DataBrowser: archive data structures

- *acquired data (D, red)*
- *related parameter (P, yellow)*
- *configuration (C, blue)*
- *grouped together (G, green)*

Partitioning of data (groups, streams)

Mass data (stored continuously, single primitive type)

DATASTREAM

- Sensor data e.g. sampled by an ADC stored in a single Archive Stream

Describing data (stored whenever a change happens, manifold structured data)

PARLOG (1:1 DATASTREAM)

- parameter data that is describing both the sensors and the ADC characteristics
- stored whenever a parameter changes
- ➔ allows for interpretation of the stored sensor data

CFGLOG

- information about the CoDaStation e.g.
  - when a station is started or stopped

MORE

- Segment execution information (both user level and hardware level)
- User Comments
- Overall system information

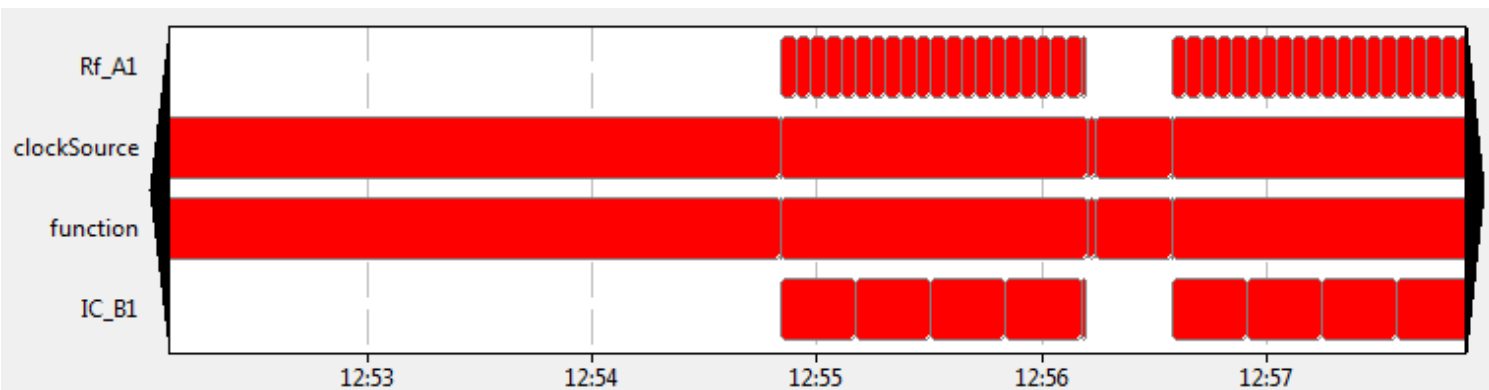
Storage details (chunking in time)

Continuous data

- Partitioned (in time)
- “chunks” of well defined time intervals
- Storage in “boxes” of the respective primitive data type
- Storage together with measured timestamps
- For every (primitive) data type a special data boxes type is available.

Describing data

- Stored together with timestamps (define validity intervals)
- Mapping via timestamps and naming convention



Screenshot DataBrowser:
boxes time as index
• matching between multiple archive streams (streams as rows)
• using the absolute time (Time at x axis)

Storage details (serialization, timestamps)

Serialization

- All archive data is stored as plain old java objects (POJOs)
- Data is contained in POJOs
- POJOs are serialized for storage
- Boxes files per stream and day contain the archive payload (POJOs)
- Boxes files are stored as binary random access files
- Boxes are accompanied with index files per boxes file
- Index files are binary files that store index POJOs

Timestamps

- all data (mass data and parameters) must be stored together with absolute W7-X timestamps (W7-X time: ns since 1970, 64 bit long values)
- sampled W7-X time provided via TTE system (timing hardware device or NTP)
- matching of related data by time of interest
- make the content of the archive comparable in time

API and Queries

A typical SQL-like time series query example in NoSQL cassandra

```
SELECT temperature
FROM temperate
WHERE weatherstation_id='123ABC'
AND event_Time > '2015-04-03 07:01:00'
AND event_Time < '2015-04-03 07:04:00';
```

The corresponding query in the (internal) ArchiveDB API exposing internal structures

```
StreamAccess streamAccessObject = StreamAccess.of(“W7X”, “123ABC”, “temp_sensor” );
final long t0 = W7Xtime( '2015-04-03 07:01:00' );
final long t1 = W7Xtime( '2015-04-03 07:04:00‘ );
streamAccessObject.getBox( TimeInterval.with (t0, t1) );
```

The corresponding query in the high level API **SignalAccess for users**

```
SignalAddress sigaddr = selectFromList( “W7X”, “123ABC”,  “temp_sensor”, temperature3 );
sigreader  = SignalToolsFactory.makeSignalReader( sigaddr );
final TimeInterval ti = TimeInterval .with ( W7Xtime( '2015-04-03 07:01:00' );
                                             W7Xtime( '2015-04-03 07:04:00‘ ));
 final Signal sig = sigreader.readSignal( ti, ReadOptions.firstNSamples( 510 );
```

ArchiveDB: Lessons learned (1)

Schema-less parameter store

- Separation of Archive data from CoDaC runtime configuration
- Storage of all previous runtime information as “historical data” in ArchiveDB
- ➔ Changes and evolution of the configuration schema must be supported
- Archival of flexible tree structures of any structured data makes the archive schema less, schema is carried “piggyback” with data itself

ArchiveDB: Lessons learned (2)

Timestamps as index criterion and chunking of continuous data

- Technology of storing data as objects together with timestamps and chunking in boxes ➔ proven stable over a decade
- Users have to split large queries in time (arbitrary time intervals possible) into smaller ones by looping over smaller time intervals ➔ users must be trained to become familiar with continuous operation
- Major challenge for users from fusion experiments that are used to use shot numbers to identify their data ➔ provide similar mechanism

ArchiveDB Data import: Web API

REST = Representational State Transfer

The architecture of the WWW

Idea of REST:

- Data is represented as **resources** on a server
- Resources are accessible via an address (URL)
- Hyper Links to related resources (for navigation)
- Fixed set of operations to perform on resources
  - HTTP methods supported:
    - **GET** for reading resources
    - **POST** for creating new resources
- **Different representations** of resources
  - e. g. HTML, XML, JSON, image formats etc.
- Data representation differs from archive internal storage

ArchiveDB Data import: Web API

Reading formats

- HTML sites for previews (some browsing functionality)
- JSON for reading of data with your own programs
- JPEG and PNG for image data

Writing formats

- JSON for ADC-like data and analysis data
- HDF5 for the upload of image data (video diagnostics)

Do not expect to read back what you have imported with the same format!

Timestamps via trigger timing information

Timing and triggering of not (yet) integrated systems

- CoDaC will provide several triggers
- CoDaC does precise measuring timestamps of the triggers
- Trigger timestamps available **afterwards** (~*sec*) in ArchiveDB
- Calculate / correct timestamps using the information provided

➔ **Precise timestamps must be computed using external information that is available only after data acquisition**

Timestamps via NTP

Every data entity needs a timestamp ➔ **need to calculate from available information**

Network Time Protocol (NTP) synchronization

- NTP server provided via Local Area Network (LAN)
- Typical NTP accuracy on the Internet from about 5ms to 100ms
- NTP accuracy in LAN is better
- Suitable for systems with slow sample frequencies
- e.g. all PLC systems synchronize time with NTP

➔ **Synchronization to W7-X time in NTP accuracy is available**

Archive DB Conclusions

- Continuous archival of time stamped sensor data for a scientific fusion Experiment
- Archival of structured parameter data
- Main technique consists of chunking the data in boxes with timestamps
- Changes of underlying storage techniques have already mastered once (invisible to users)
- A decade of operation with multiple and diverse data sources show the success of the data archival
- Novel user API for queries
- The system can well compete as niche technique with emerging general purpose big data technologies
- Scalable system
- User friendly view for data access is necessary

Questions?

- Thank you.