Storing EPICS process variables in HDF5 files for ITER

Rodrigo Castro [1], Yury Makushok [2], Lana Abadie [3], Bertrand Bauvir [3], Ralph Lange [3], Andre Neto [4]

[1]Laboratorio Nacional de Fusión, CIEMAT. Madrid. Spain
[2]MINSAIT - INDRA, Madrid, Spain
[3]ITER Organization, St Paul lez Durance, France
[4] Fusion for Energy, Barcelona, Spain

2022 European HDF5 User Group

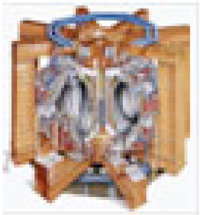
01/06/2022

iter china eu india japan korea russia usa

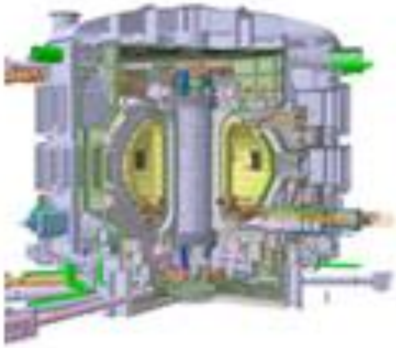
Ministerio de Educación y Ciencia — Ciemat Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas

indra

Current context



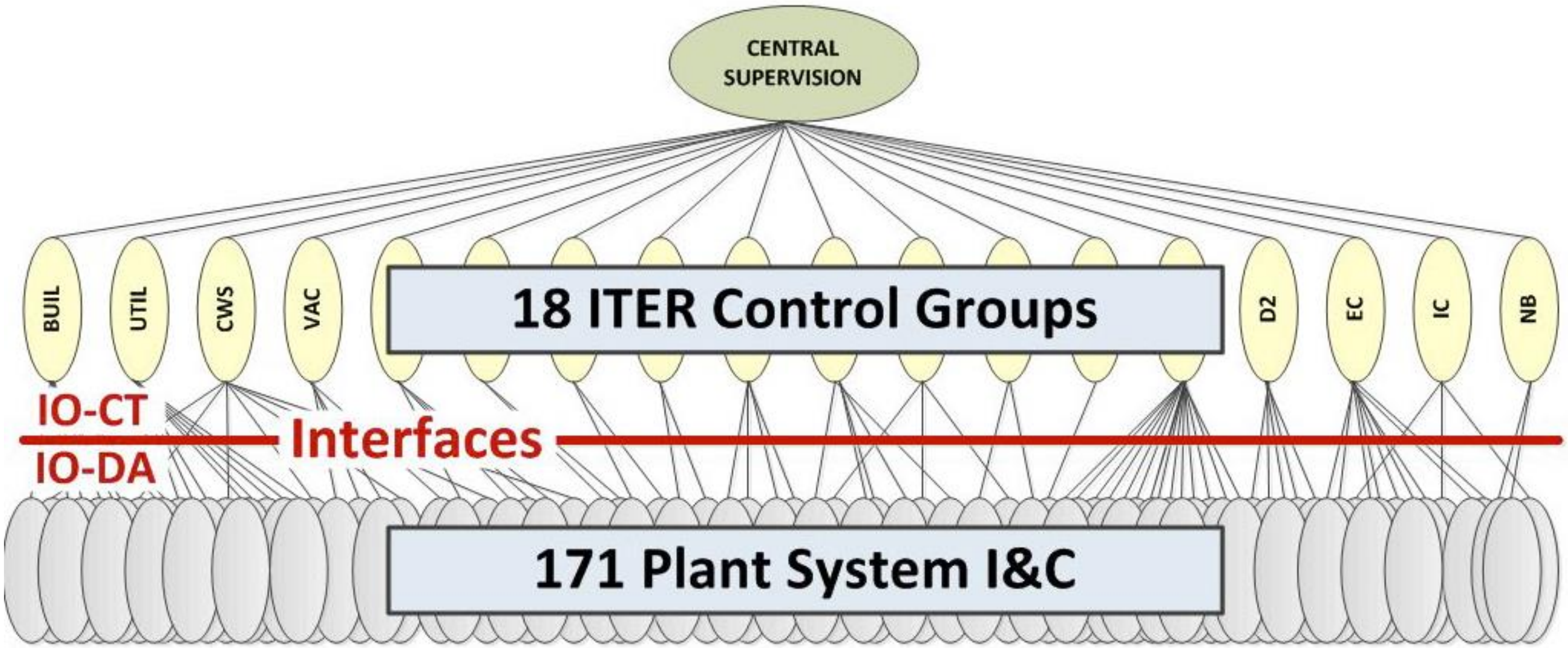
JET
80 m^3
~16 MWh



ITER
800 m^3
~500 MWh

DEMO
1000-3500 m^3
~2000-4000 MWh

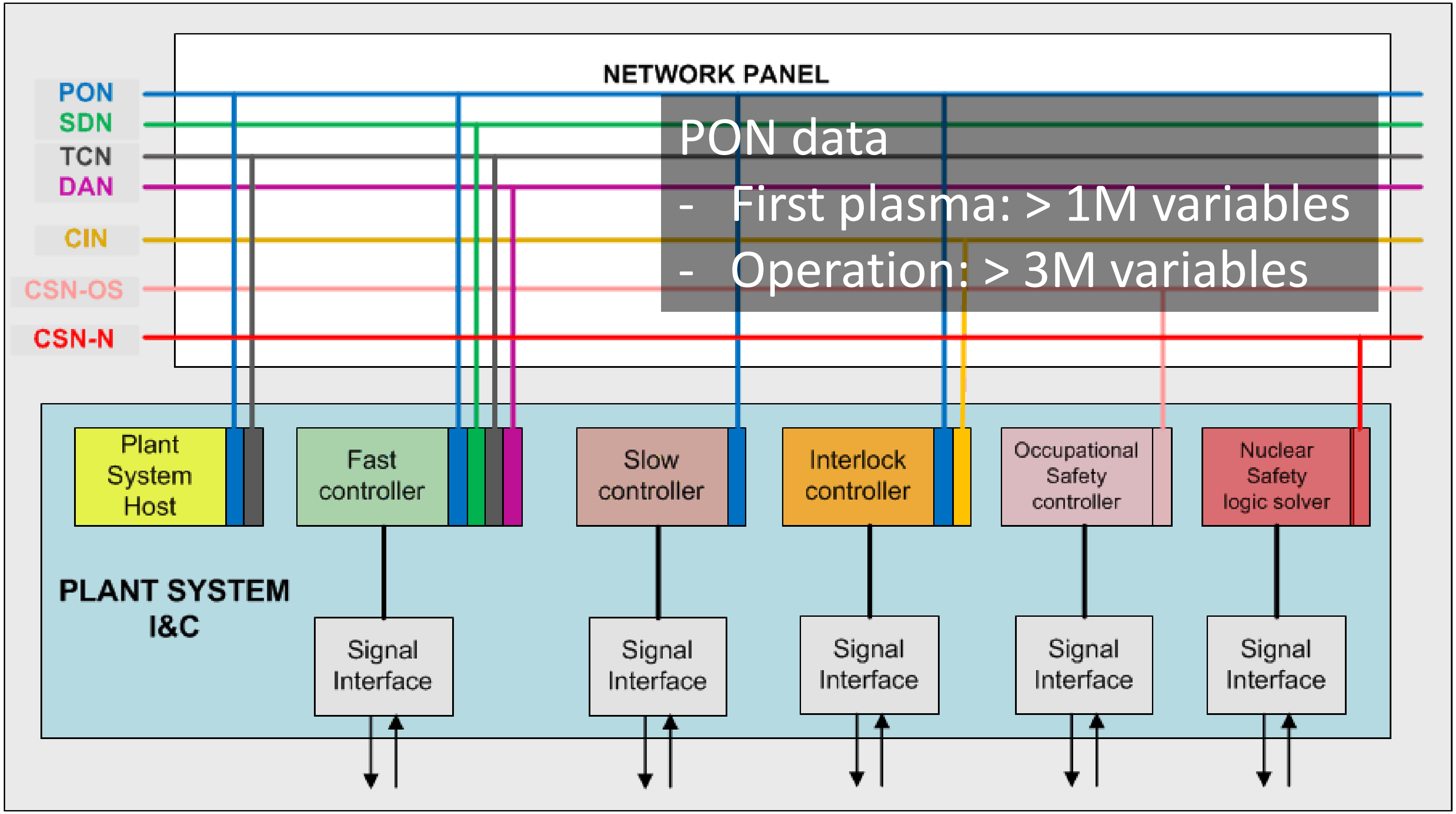
ITER control system organization



- ITER CODAC: Common language for all PS I&C
- Distributed control system based on EPICS

EPICS

Plant Operation Network (PON) data



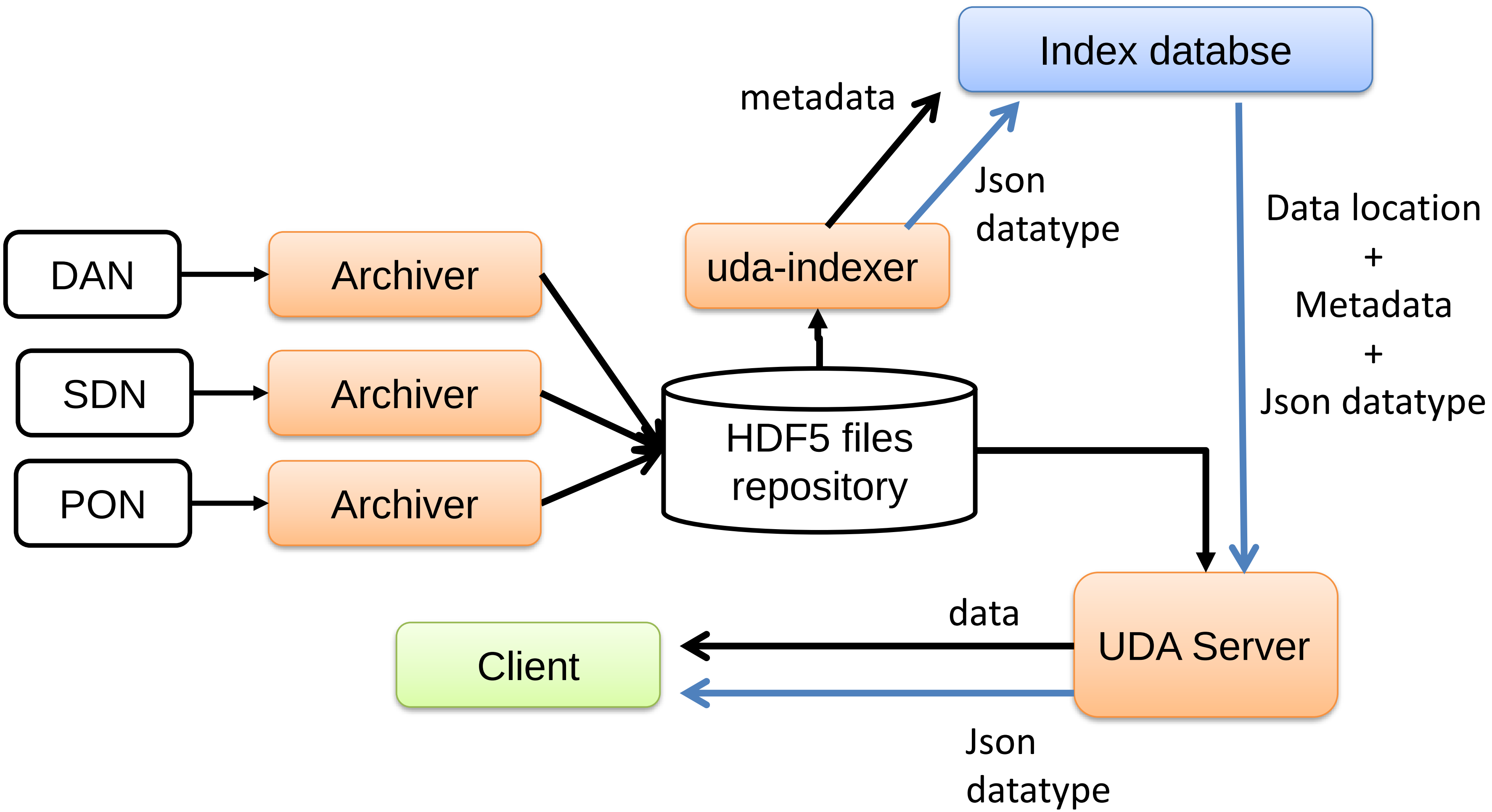
PON data

- First plasma: > 1M variables
- Operation: > 3M variables

Design and implementation consideration

- Time evolution data in steady-state operation
  - Requests based on time interval
  - Different files along the time can be involved
  - Different data nature along the time can be involved
    - Different dimensionality
    - Different sampling rate
    - Different units
    - Different datatype
- Data must be accessible on fly
  - Data flush timeouts
  - Real time indexing mechanism
    - New files / growing files
    - New files, open for writing files and closed for writing files must be notified

Data archiving/retrieving cycle



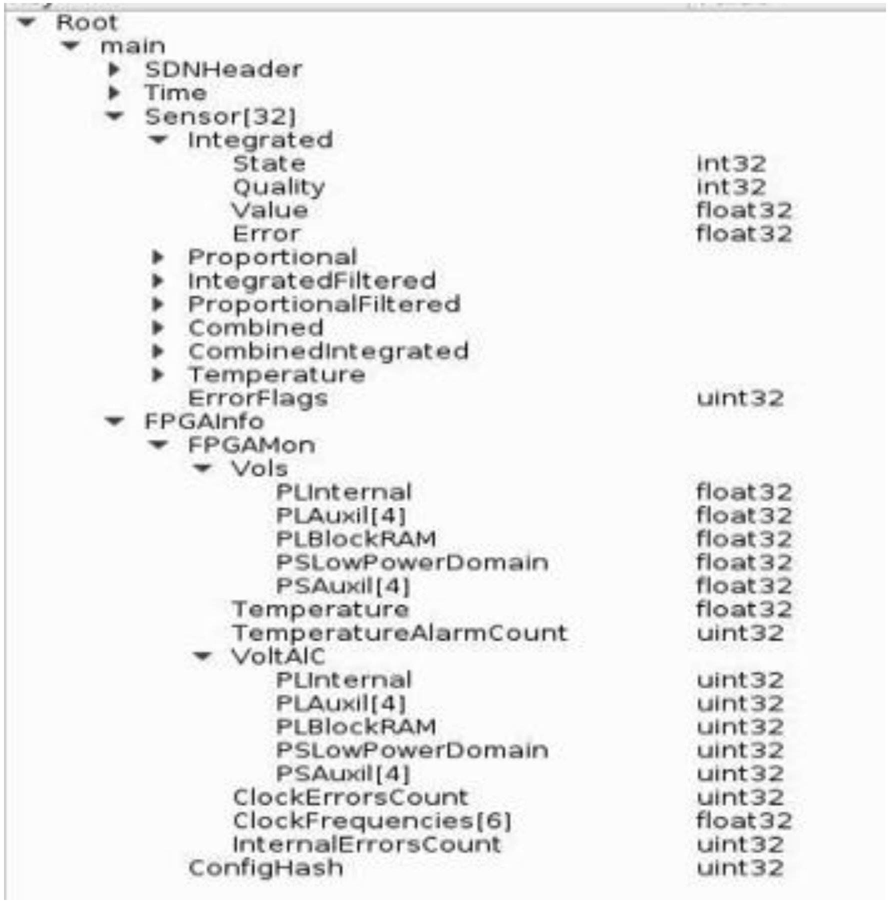
EPICS Process Variables

- Before EPICS v7: Channel Access Protocol
- Values have a simple composite type
  - Timestamp
  - Status
  - Severity
  - Value
- Support a set of primitive data types: integer, double, string, enum
- Some metadata can be retrieved
  - Enum labels
  - Visualization metadata

EPICS Process Variables

❑ After EPICS v7: PVAccess protocol

- Very complex nested datatypes can be defined
- Can include multiplicity at any level
- Structure leafs have additional metadata



Example

Channel Access archiving: PON archiver

- **SWMR model**
- Aggregates the maximum number of PVs in one file
- Implements flush timeout (to warranty maximum read latency)
- HDF5 backend files rotate in some size / time conditions
- Any PV can change on fly:
  - Type
  - Enum labels
  - Display metadata

PON archiver (Channel Access archiver)

- HDF5 model
  - One group per PV
    - Payload dataset: timestamp, status, severity, value
    - Dynamic metadata attributes
      - Enum labels
      - Display metadata
  - Until 80K PVs / file

PON archiver (Channel Access archiver)

- Challenges
  - Memory problem
    - HDF5 caches must be correctly managed
  - Expensive file rotation
    - Creation of all objects have big CPU usage and takes significant time
    - File rotation in asyn mode (separate thread)
  - If a change in PV property: datatype, units, enum labels
    - New individual file is created until next rotation
  - Performance problem for flushing updated PVs
    - One flush for all file at the end of the loop (avoiding flush per dataset)

PVaccess archiving: PVA archiver

- ✓ **SWMR model**
- ✓ Manages big nested datatypes
- ✓ Autodiscovery PV datatype
- ✓ PV datatype can change

Big nested datatype: some alternatives

- Use opac datatype + datatype definition
  - Breaks our current model based on HDF5 types
  - Mandatory to read all data structure just for one field
- Flat nested structure: 1 composite field
  - Cases of 16K fields (10 DAQ boards)
  - **HDF5 limitations: maximum about 1300 fields**
- One dataset per field
  - Good read performance
  - Poor write performance: 1 write -> 16K writes

Big nested datatype: first implementation

- **Flat structure break algorithm**
- Iterates flatten structure trying to:
  - Find the longest common path (trying to group as much as possible)
    - Until aggregation limit (number of fields or size limit) is achieved
  - Check if this path already exists in the file
    - If exists -> necessary to force a new break with 1 level longer paths (less aggregation)
  - Add a group for the found path name
    - With a composite fields payload dataset that aggregates all data under the found path name

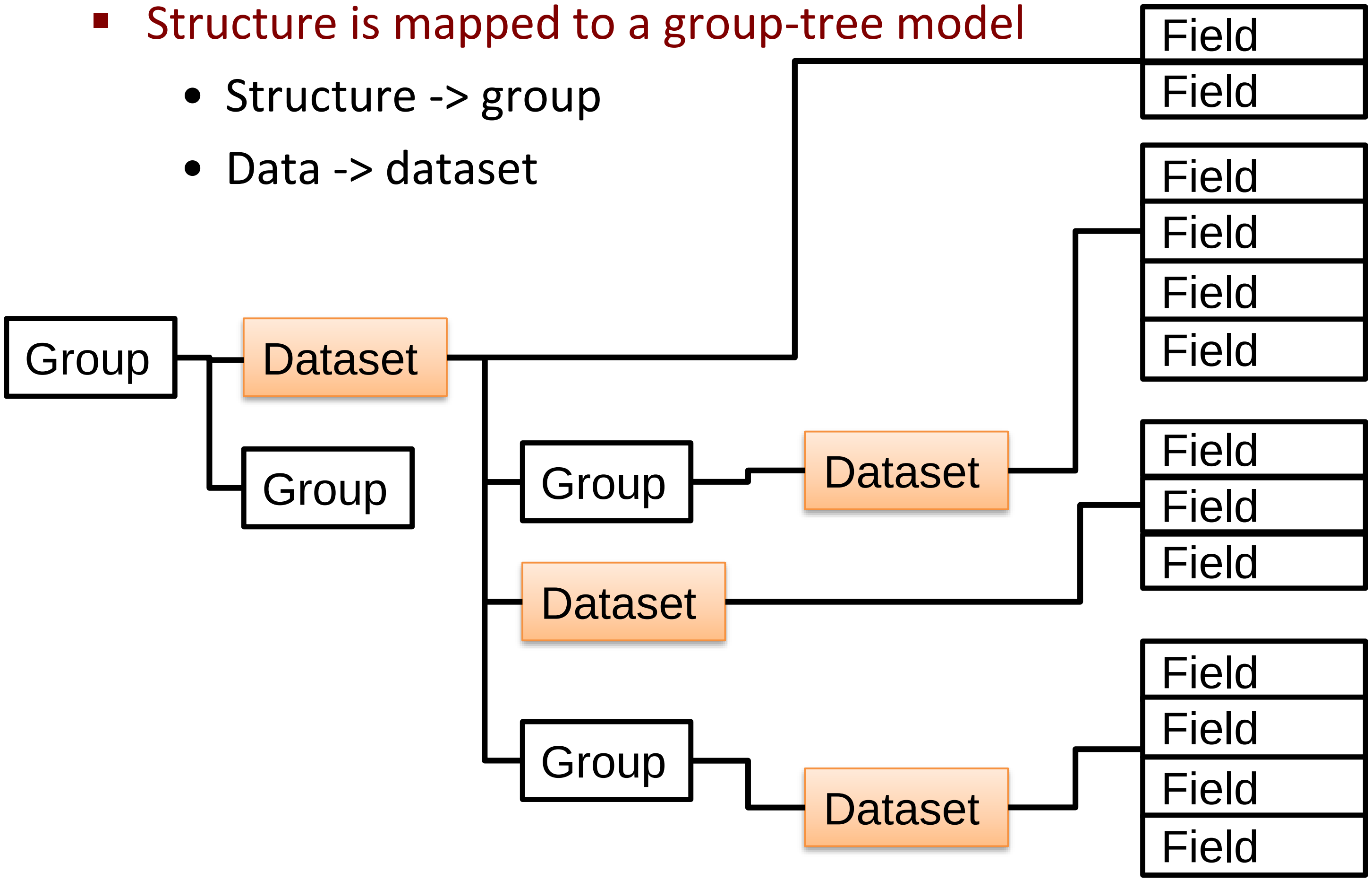
Big nested datatype: first implementation

- **Flat structure break algorithm**
- Pros:
  - Current HDF5 archiving model (reading, indexing) is valid
  - Level of aggregation (size of datasets) is configurable
  - Good aggregation results / universal algorithm : Breaks 8K fields (5 DAQ boards) -> 220 datasets
- Cons:
  - HDF5 structure is not visually a 1-to-1 map of the original nested structure

Big nested datatype: second implementation

❑ Group-tree data model

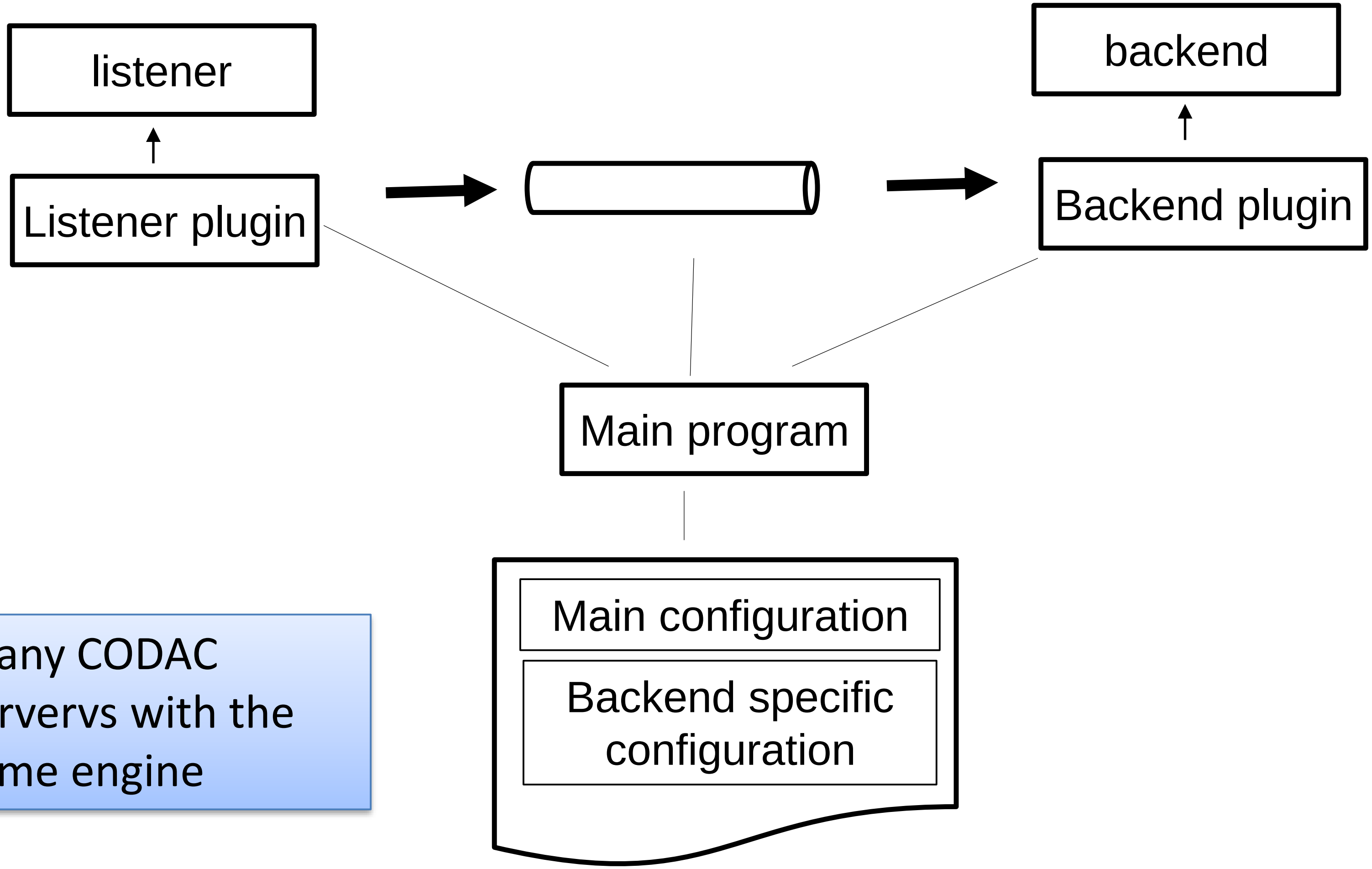
- Structure is mapped to a group-tree model
  - Structure -> group
  - Data -> dataset



Big nested datatype: second implementation

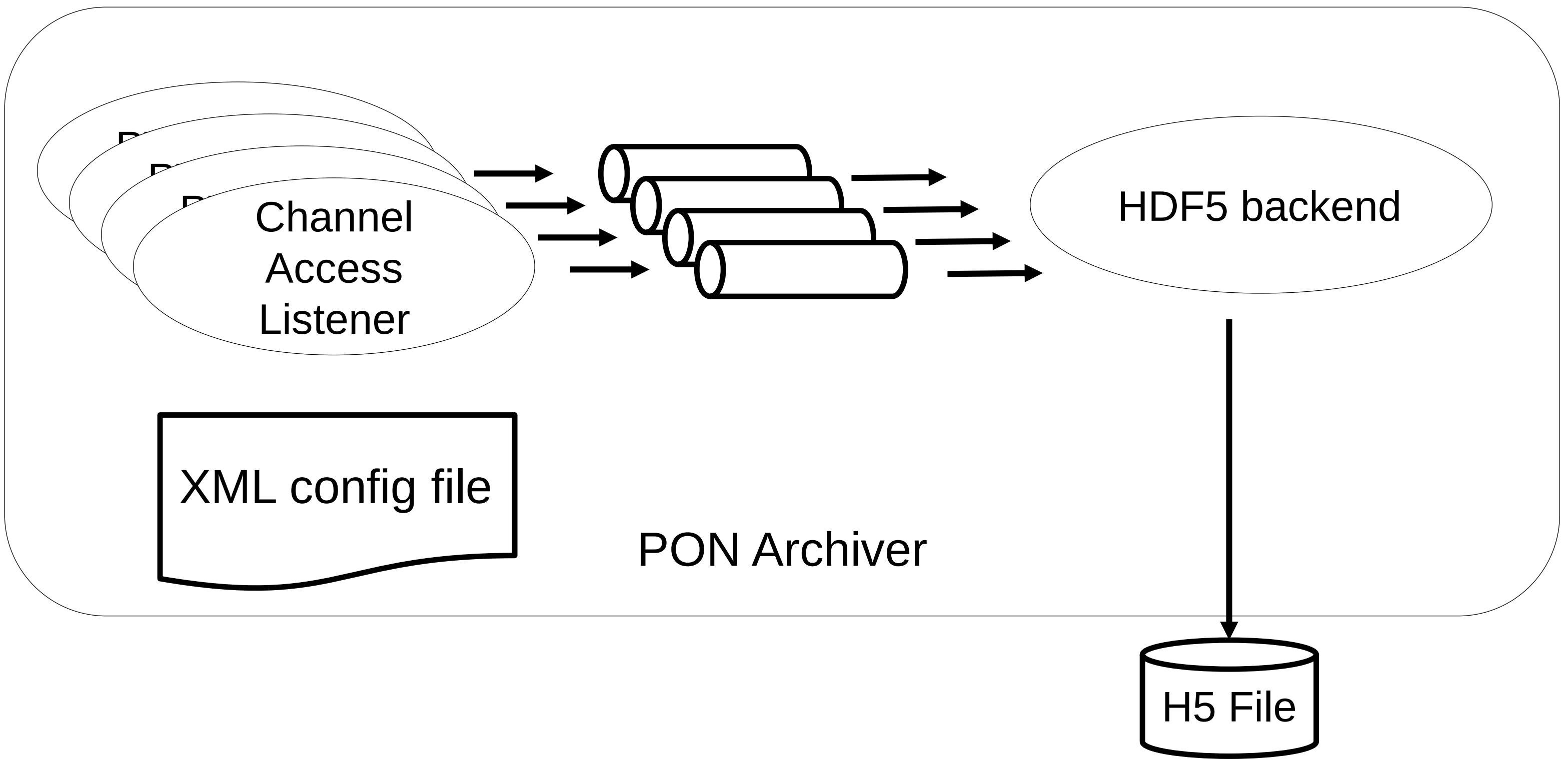
- **Group-tree data model**
- Pros:
  - HDF5 structure maps 1-to-1 the original nested structure
  - Easy to extract a subtree of data
  - Read performance
- Cons:
  - Not already a complete solution in case of leaf composite datatype with more than 1300 fields (is really a limitation?)

Common Architecture

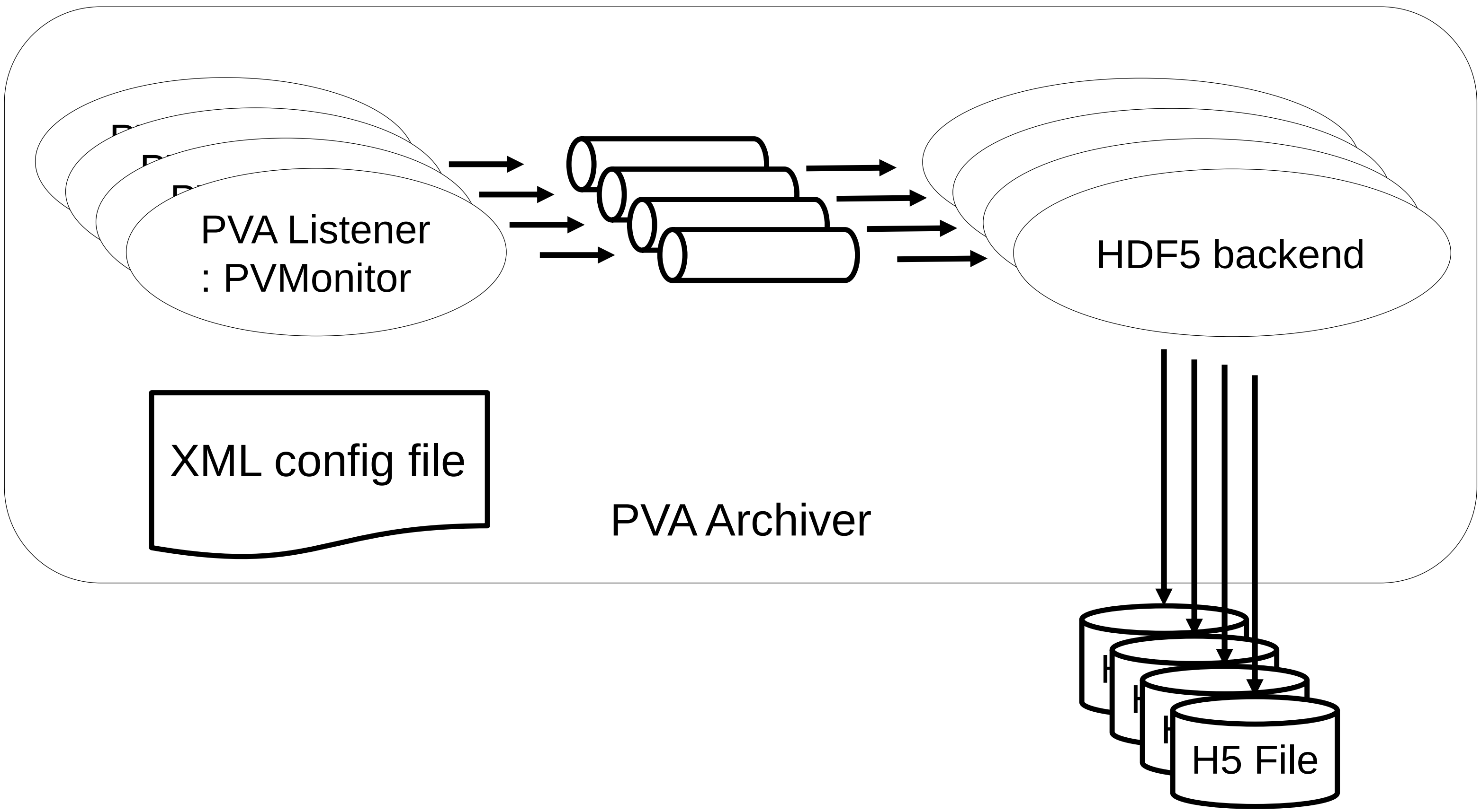


Many CODAC servervs with the same engine

PON archiver implementation



PVA archiver implementation



Thank you for your attention

Questions?