CLUSTER OF RESEARCH INFRASTRUCTURES
FOR SYNERGIES IN PHYSICS

Prototype for High-Speed Data Acquisition at European XFEL

CRISP 3rd Annual meeting

June 2-4, 2014

Djelloul Boukhelef

Outline

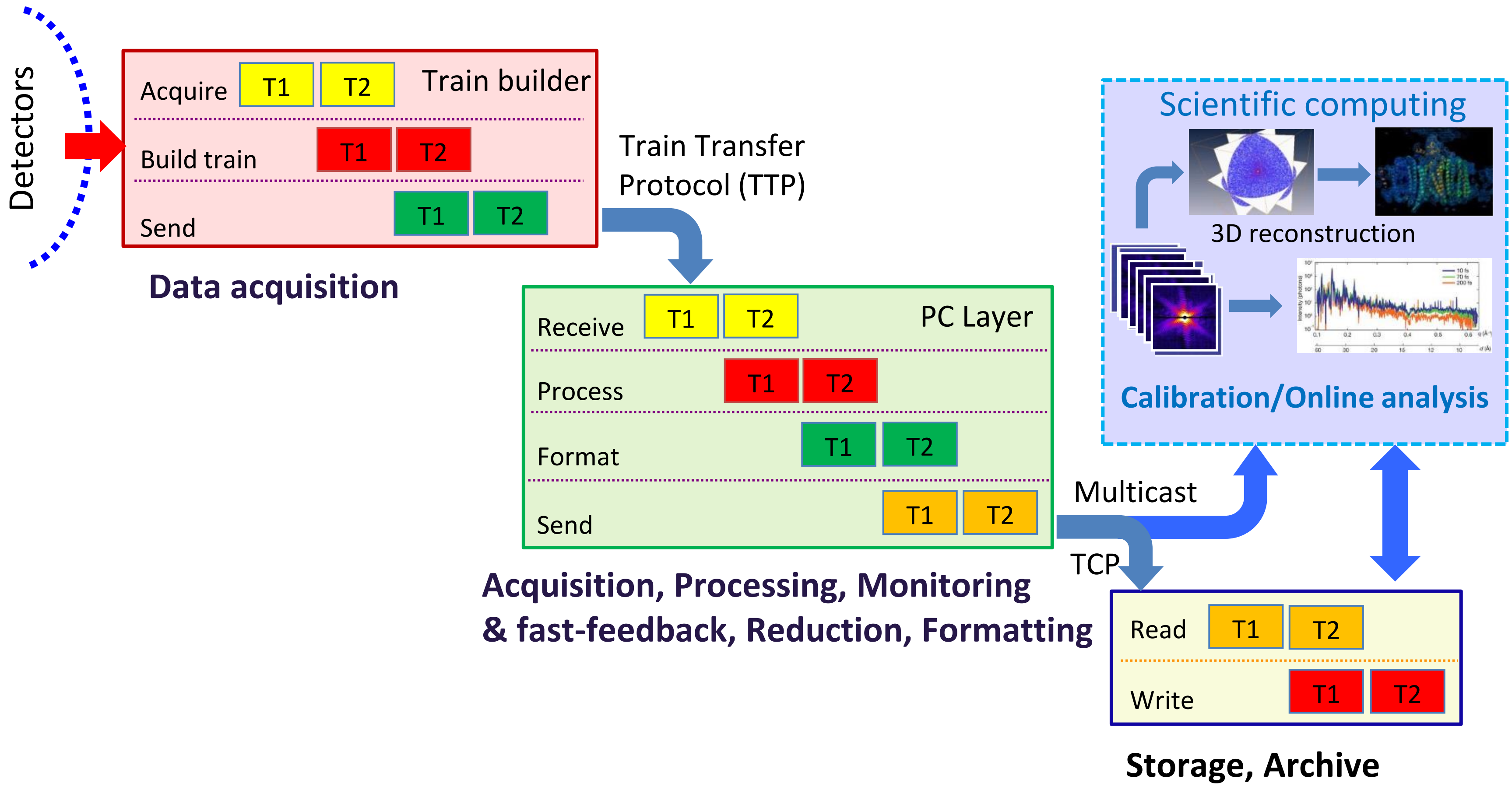
- Introduction
- Hardware and network setup
- Software architecture
  - Performance results (overview)
  - Recent development
- Data format
- Summary

Introduction

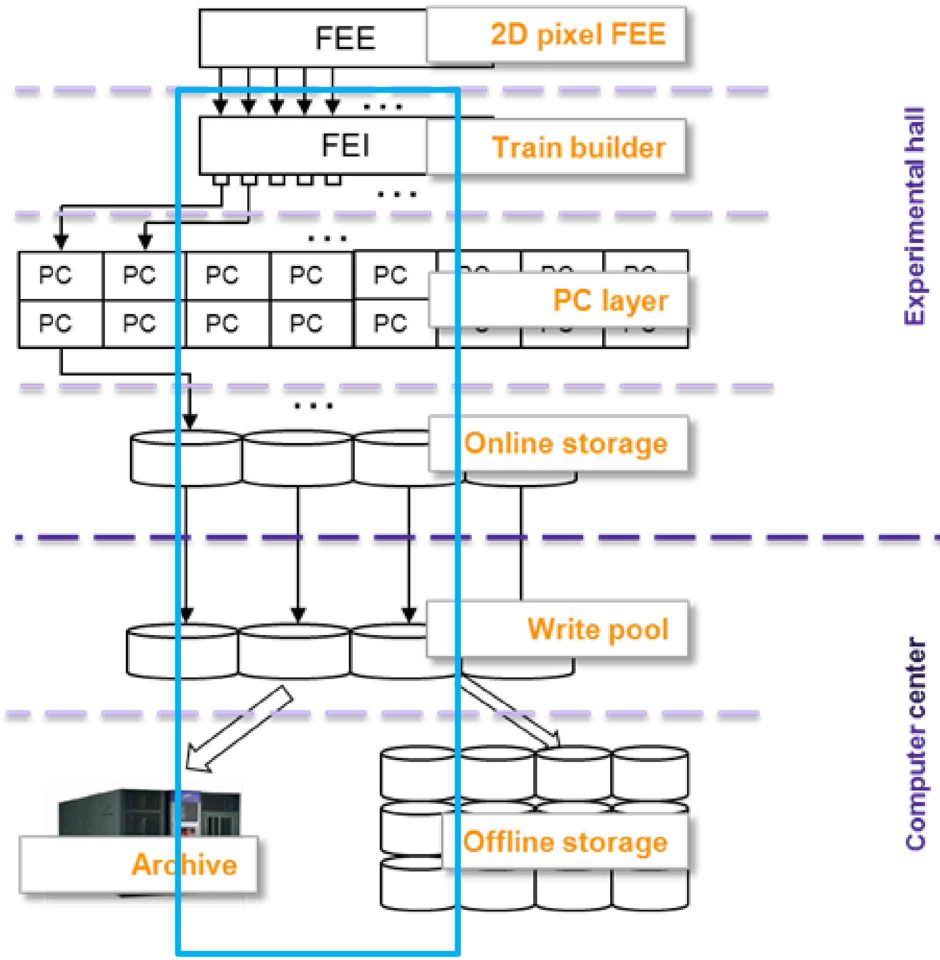
- Several detectors are under construction at the European XFEL
  - 2D detector (LPD, AGIPD, DSSC) will generate several GB/s of raw data
  - Large number of other detectors will be deliver GB/s of fast and slow data
- We are building a prototype of the DAQ/DM/SC system to handle the large volume of data coming at very high-speed
  - Main focus is on data acquisition, pre-processing, monitoring and fast-feedback, formatting and storage
- Purpose of this work
  - Define the software and hardware architecture
  - Select and install adequate hardware components
  - Develop necessary software components: our special design focus is on performance, reliability, scalability, flexibility
  - Perform thorough testing of the full DAQ and DM system, to assess the performance and stability of the h/w + s/w
    - Network: bandwidth (10Gbps), UDP &TCP behavior…
    - Storage: performance of disk (write), concurrent IO operations, …
    - Processing: concurrent read, processing, write operations, …

Big picture

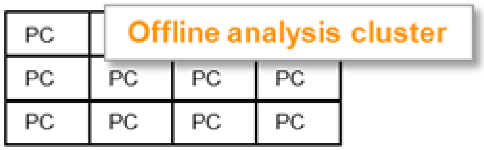
- Data acquisition, processing and management system



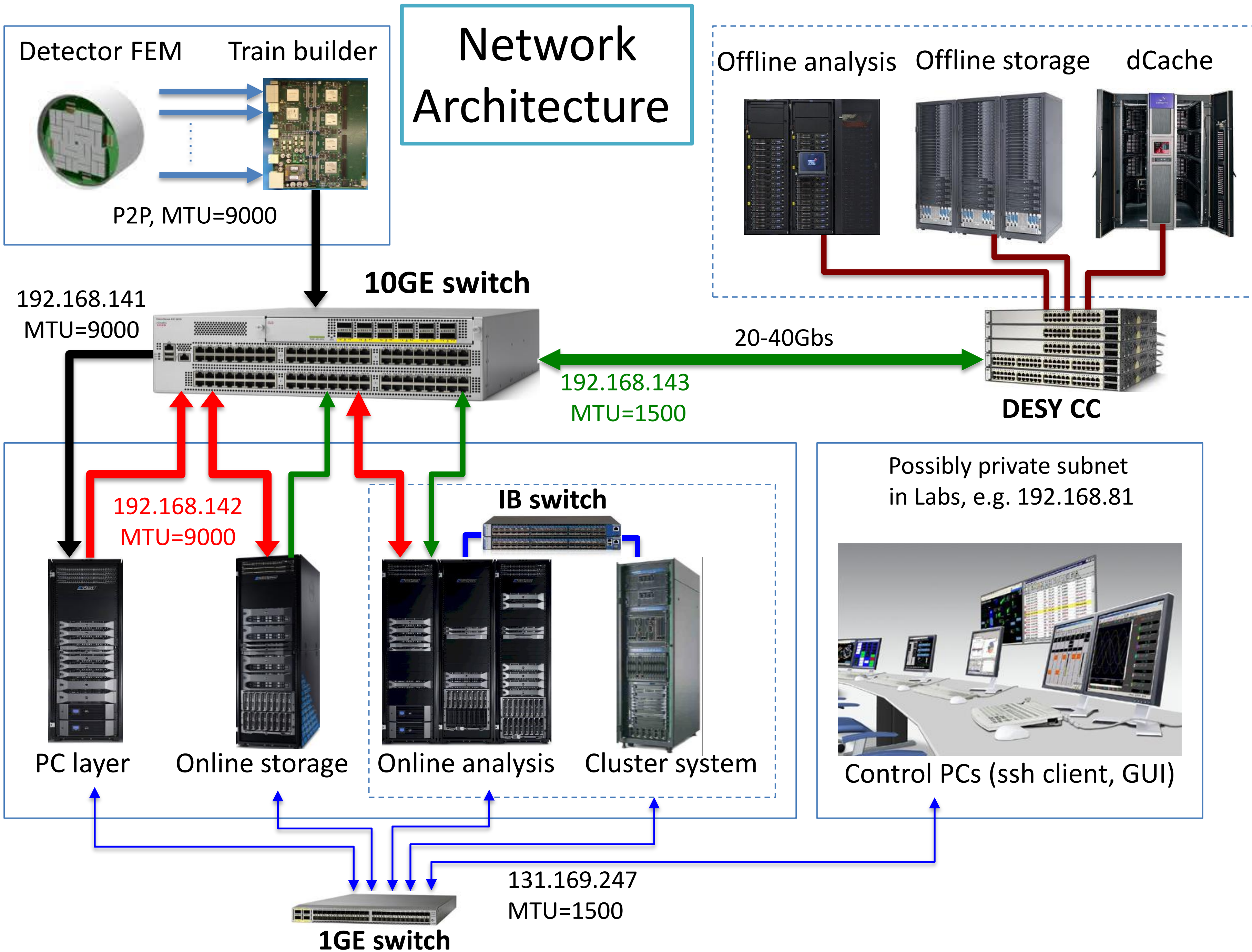
Hardware setup

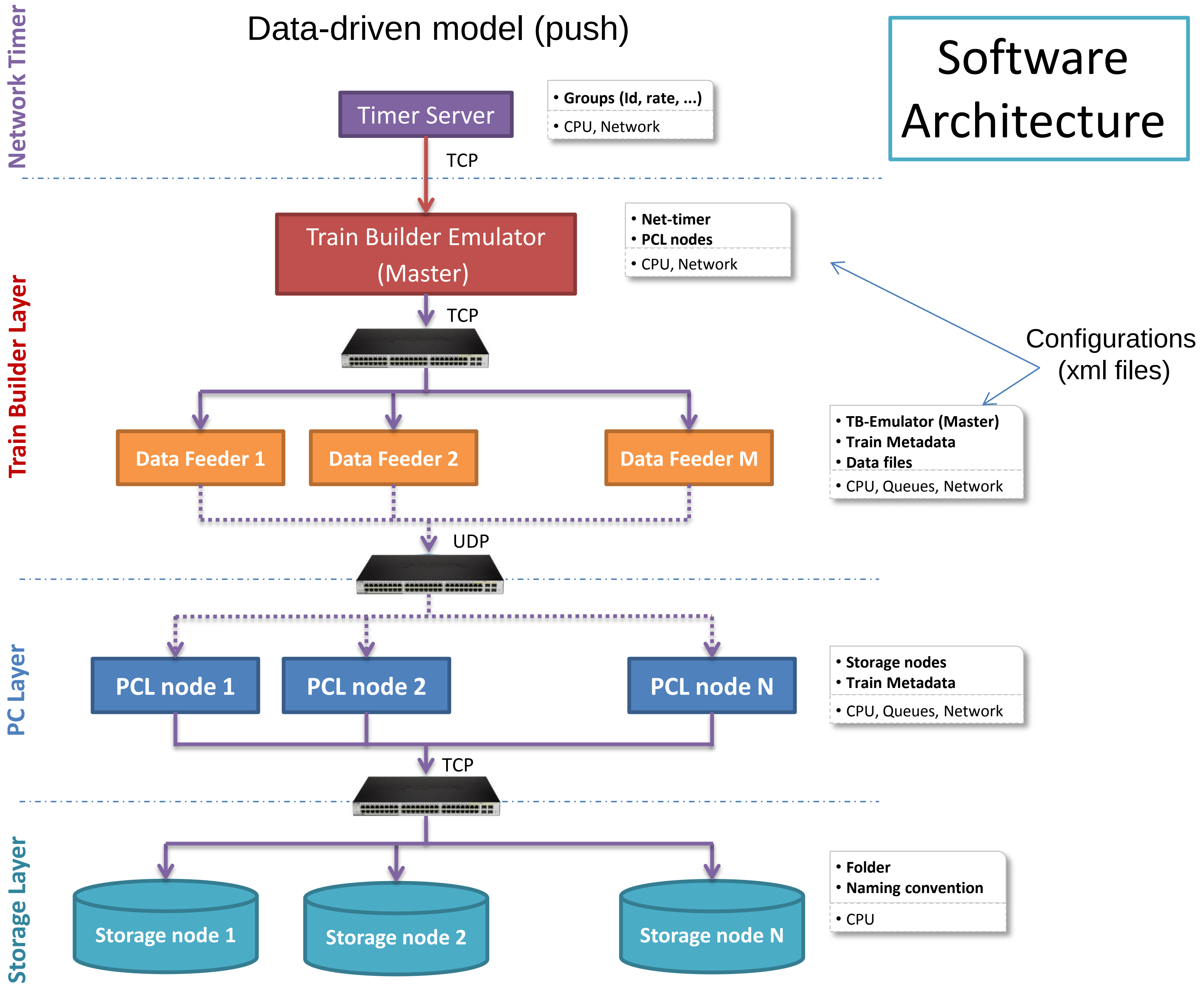


- 2D detector generates up to 10GB of image data per second
  - Data is multiplexed on 16 channels (10GbE)
  - 1GB per 1.6s per channel
- Lots of other fast/slow data streams

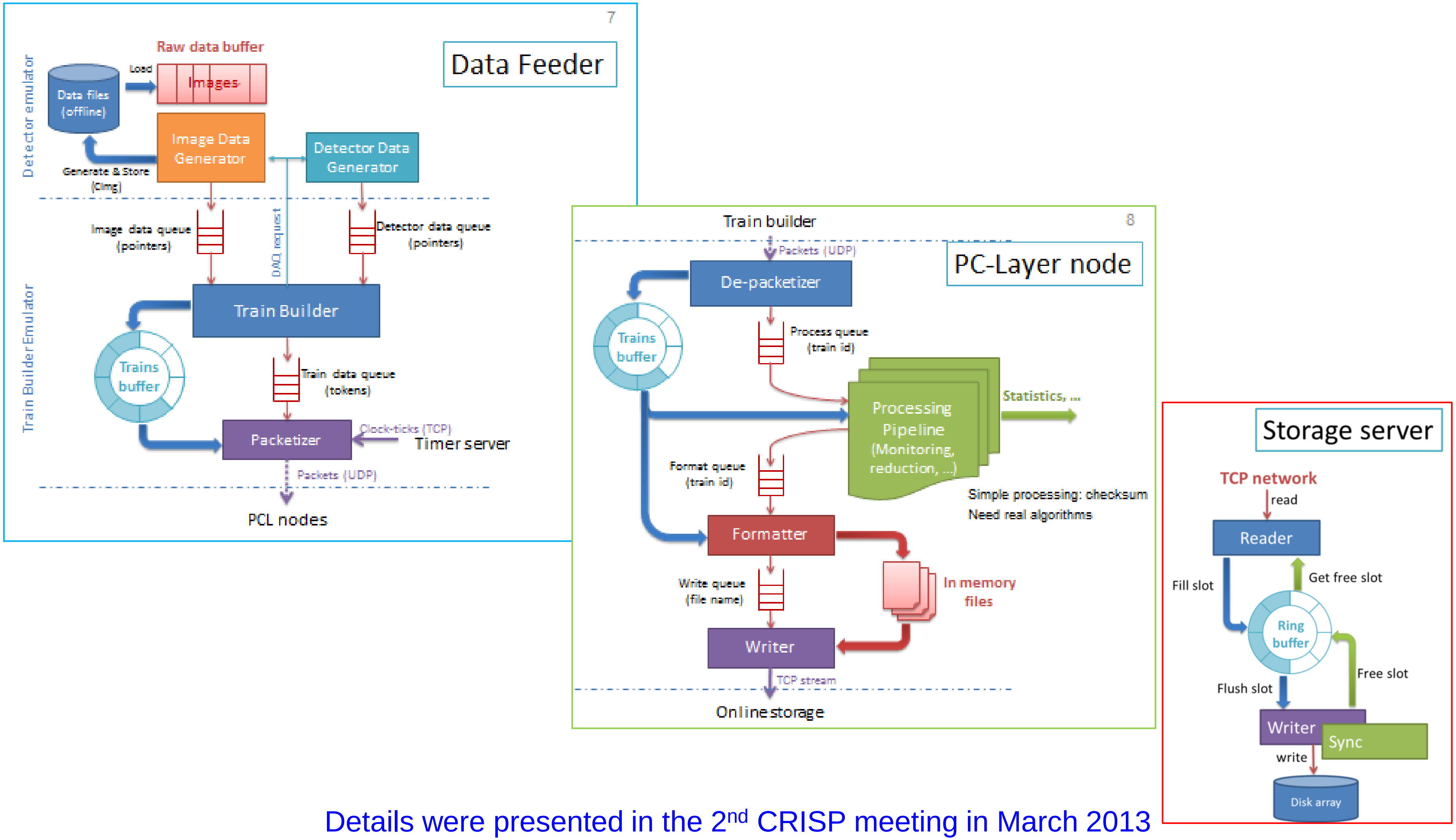


Details were presented in the IT&DM meeting in October 2012


Network Architecture
Detector FEM
Train builder
P2P, MTU=9000
Offline analysis
Offline storage
dCache
10GE switch
192.168.141
MTU=9000
20-40Gbs
192.168.143
MTU=1500
DESY CC
192.168.142
MTU=9000
IB switch
Possibly private subnet
in Labs, e.g. 192.168.81
PC layer
Online storage
Online analysis
Cluster system
Control PCs (ssh client, GUI)
131.169.247
MTU=1500
1GE switch


Data-driven model (push)
Software Architecture
Network Timer
Timer Server
• Groups (Id, rate, ...)
• CPU, Network
TCP
Train Builder Layer
Train Builder Emulator (Master)
• Net-timer
• PCL nodes
• CPU, Network
TCP
Configurations (xml files)
Data Feeder 1
Data Feeder 2
Data Feeder M
• TB-Emulator (Master)
• Train Metadata
• Data files
• CPU, Queues, Network
UDP
PC Layer
PCL node 1
PCL node 2
PCL node N
• Storage nodes
• Train Metadata
• CPU, Queues, Network
TCP
Storage Layer
Storage node 1
Storage node 2
Storage node N
• Folder
• Naming convention
• CPU

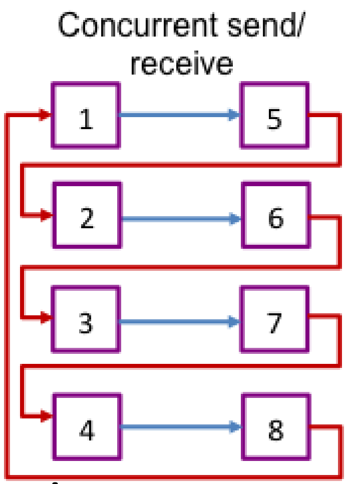
PC Layer software prototype



Details were presented in the 2nd CRISP meeting in March 2013

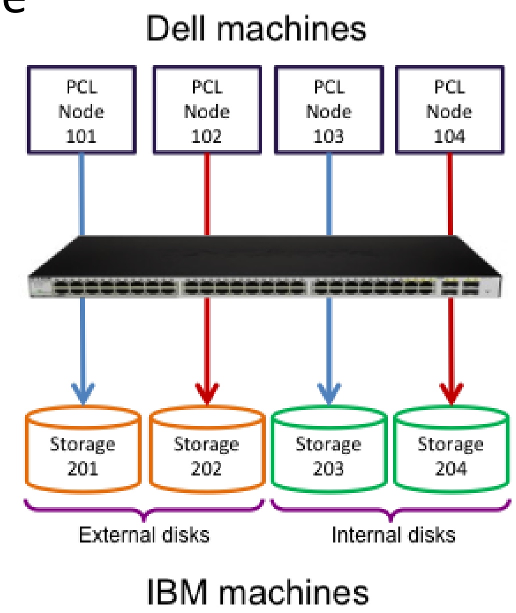
Performance tests - Network

- Network transfer speed
  - ~1GB train (131202 packets) → 0.87sec ≈ 9.9Gbps
- CPU usage (i.e. receiver core) ≤ 40%
- Packet loss
  - Few packets lost at the start of some runs, not each train
  - Happened sometimes on all machines, on some machines only, and sometimes no packets loss on any machine
- Ignoring the first packets lost which affect only the first train
  - Typical run (3.5×10-8) → 3.7 out of 10000 trains
  - Long run (5×10-9) → 26 out of million trains
- Test of dynamic packets switching, i.e. send train data from one source to different target PC each time.
  - Runs take up to 18 hours ~81657 trains ≈ 80TB
  - Same network performance and behavior like in the static mode.



Performance tests - Storage

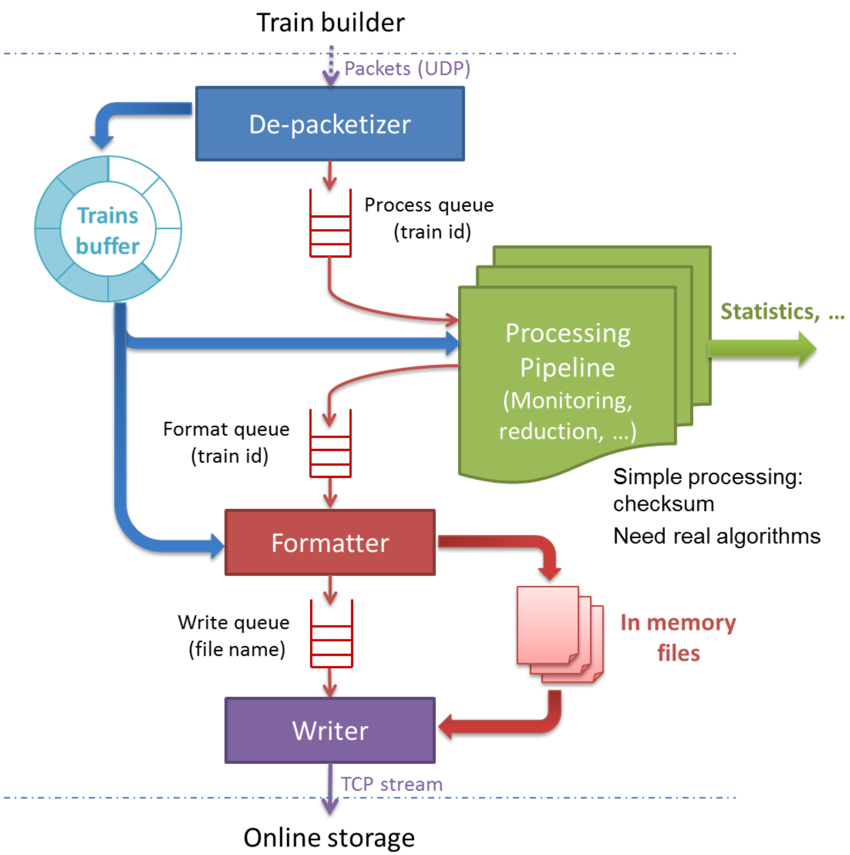
- We need to write 1GB data file within 1.6s per storage box
  - Both internal and external storage configurations are able to achieve this rate (1.1GB/s, 0.97GB/s, resp.)
  - 16 storage boxes are needed to handle 10GB/s train data stream
  - High network bandwidth and low CPU load
- Direct IO
  - Network read and disk write operations are overlapped at 97% → Low overall time per file



- Conclusion:
  - Prototype of a Full DAQ chain for ½ Mpxl detector, i.e. from train builder to online data storage
  - Sustainable and stable network bandwidth and storage performance

PC layer node – current version

- PC layer node performs series of well-defined and strictly under-control tasks
- Data receivers
  - Read train data and store it into memory buffer.
  - UDP for big train data (2D detectors)
  - TCP for slow and small data
- Processing pipeline
  - Users' algorithms that perform data monitoring, rejection, and analysis.
- Aggregator, formatter, writer
  - Filter data and merge results,
  - Format data into HDF5 files,
  - Send files to data cache and analysis.

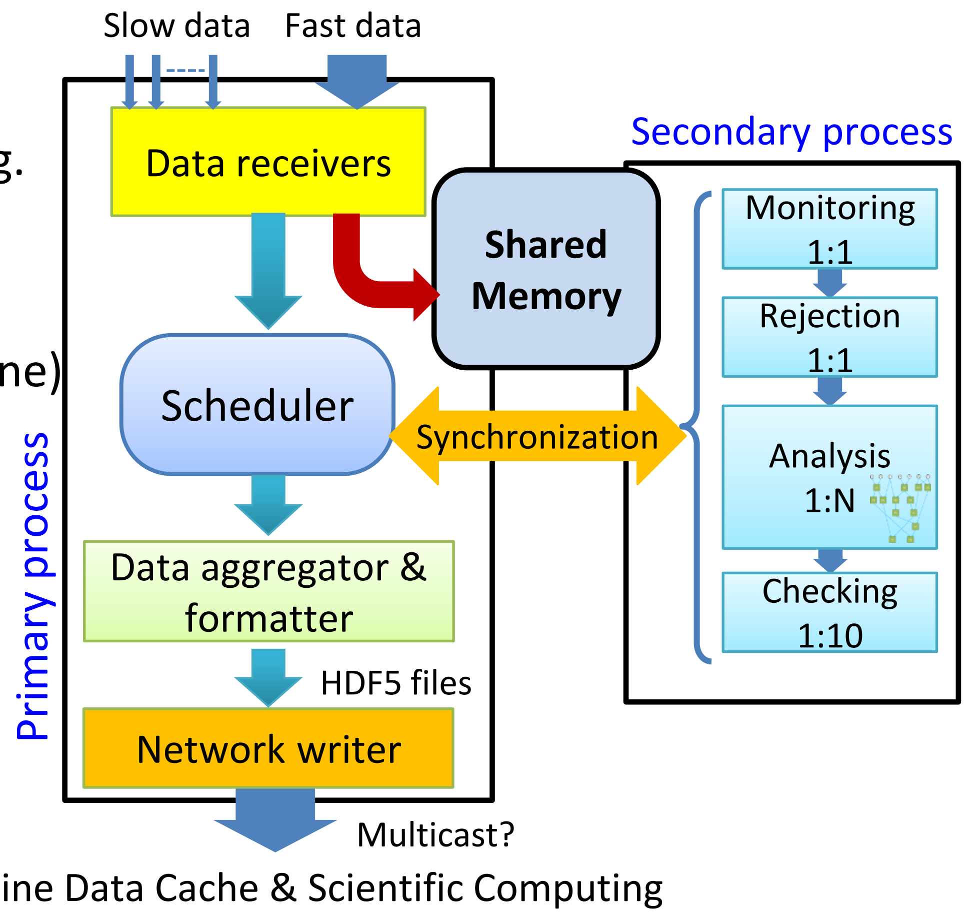


Recent developments

- Revamped the internal architecture of the PC layer software for better reliability, scalability and flexibility
  - Thread-pool and runnable objects, fast inter-thread communication queue, memory management, etc.
- Extend the concept of PC layer to cope with data coming from different data sources (2D detectors, fast digitizers, etc.)
- Introduce the concept of processing pipeline to enhance the capabilities of the PC layer
  - Runtime loading of users' fast feedback algorithms (plug-and-play)
  - Composition of users' algorithms into pipeline to perform complex tasks on the incoming data, such as data monitoring and rejection
  - Advanced tuning of the processing tasks at the PC layer
  - Increased reliability and better error recovery strategies

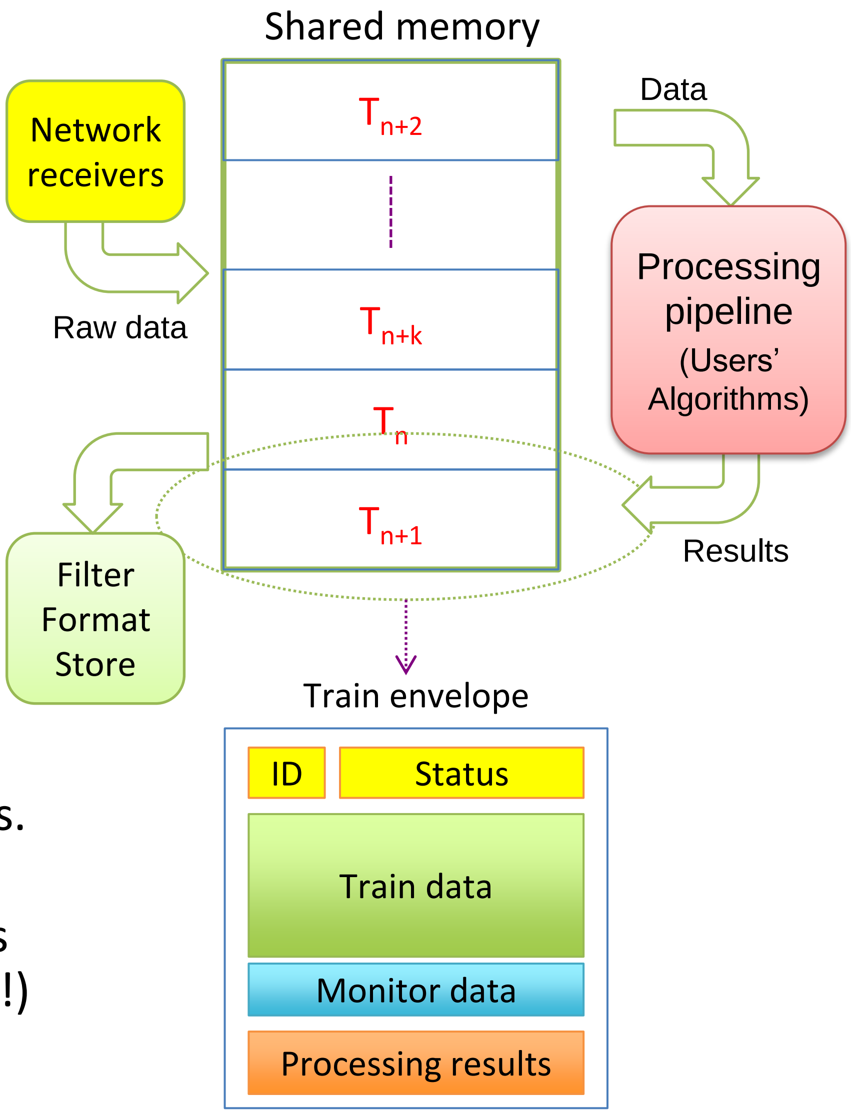
PC layer node – next version

- Master-slave pattern: PC layer node software consists of one primary process and many secondary processes
- Primary process
  - Performs critical tasks, i.e. data receiving, storing, and scheduling.
  - May requires super-user mode
- Secondary processes
  - Run users' algorithms (PCL pipeline)
  - Run at normal user-mode
- Data exchange is done through inter-process shared memory
- Scheduler
  - Monitors tasks and data status
  - Coordinates threads activities



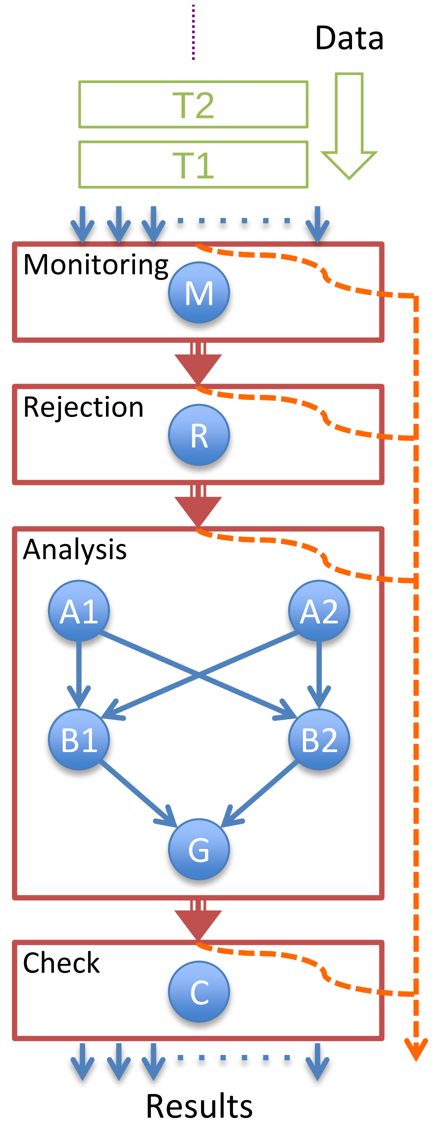
Inter-process data exchange

- Shared memory (managed buffer)
  - Manages a pool of memory buffers.
- Train envelope
  - Identified by train number
  - Contains raw data and produced results associated with one train.
  - Keeps track of data and tasks status.
- Policy
  - Only receivers can write raw data.
  - Secondary process can only read raw data and cannot alter it, but can produce and store intermediate results.
  - Aggregator and formatter consolidate rejection decisions and remove images marked as to-be-rejected (irreversible!)



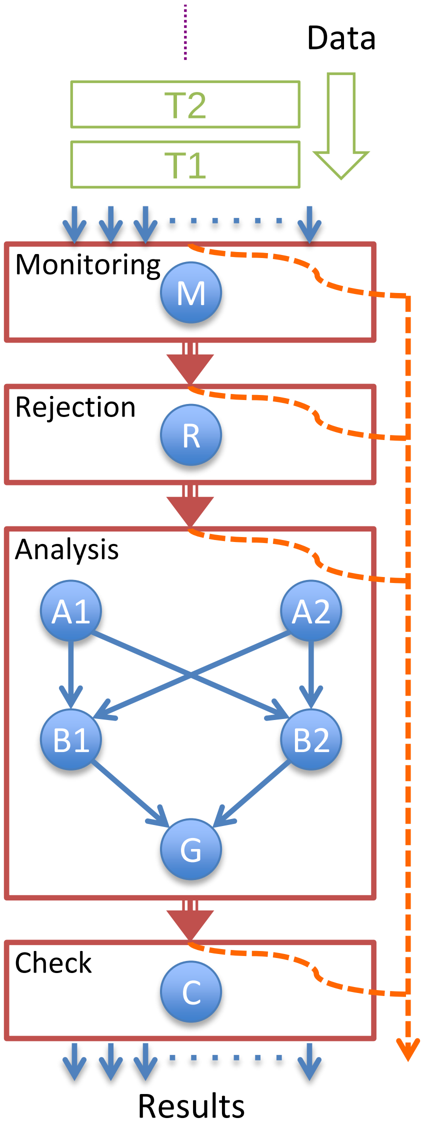
Processing pipeline

- Pipeline is fed with trains data sequentially, that's, according to their arrival order.
- Monitoring
  - Check train data before starting any processing task, e.g. data availability, validity, etc.
- Rejection
  - Fast algorithms that find and mark bad quality images, i.e. do not contain desired information or pattern.
  - Efficiency: avoid processing non-useful images.
- Analysis
  - Sophisticated algorithms that perform deep analysis.
  - Compute values, data rejection, pattern search, etc.
- Results checking
  - Monitor results, compile decisions and results, etc.
- Data and result monitoring tasks produce and publish monitor data for scientists and operators to visualize.



Policies and features

- Scheduler must perform some housekeeping tasks in order to restore the stable regime in case of errors
  - Bypass stale nodes: processing activities of current stale train and few other immediate trains are simply skipped
  - Skip processing incoming trains (i.e. shortcut pipeline) and store raw data as is to prevent any data loss
  - Instruct stale nodes to abort the processing immediately
  - In the worst case, shutdown the whole processing pipeline, and restart it again at a normal processing state
- Tuning
  - Fraction of trains to handle by a pipeline section or node
  - Number of trains handled concurrently in one section
- Pass-through
  - Fraction of data that should pass regardless of the algorithms' decisions. E.g. store every 5000th image

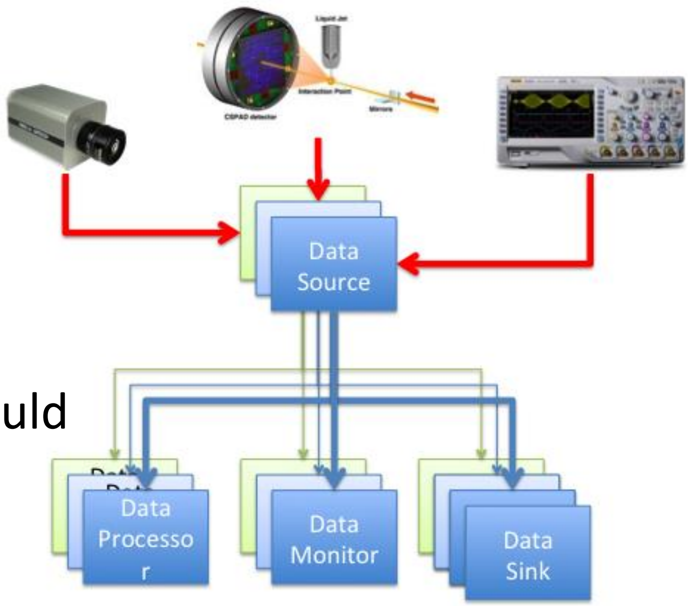


Simulation model

- We developed a simulation model of the PC layer pipeline
  - Pipeline specification using XML: sections, nodes, dependencies
  - Topological ordering of nodes
    - Detect source nodes and link them to data producers, and detect sink nodes and link them to results aggregator
    - Automatically find out input data and output data for each node and section, and detect unused or undefined data variables.
  - Probabilistic model for node execution times, error rates, etc.
  - Statistics about train residence time per node/section/pipeline, section/nodes execution time, process/skip/error rates etc.
- Next …
  - Implement a prototype of the processing pipeline using processes and real algorithms, partial/complete shutdown and restart, etc.
  - Define a common API and good practices for writing users' algorithms and data exchange

Data format description language

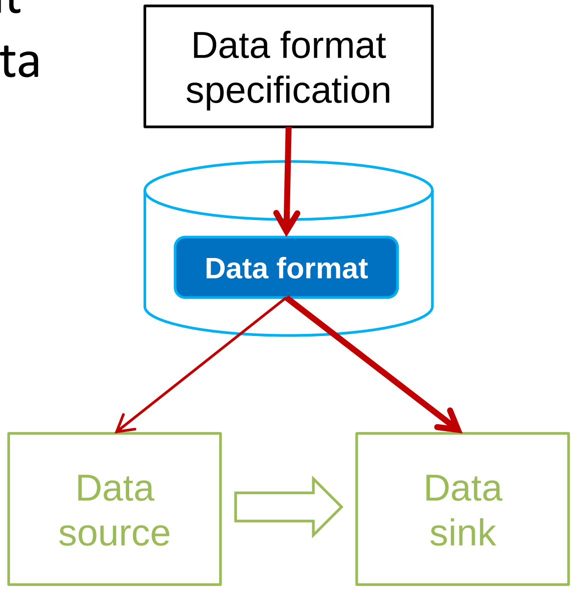
- Motivations
  - We have several data sources producing similar data contents, which are handled in the same way.
  - We are creating generic devices (PC layer, sink devices) that should be able to assimilate and handle data from multiple homo/heterogeneous data sources
  - Accessing data block sections and fields should not be hard-coded; especially in generic tools such as formatter, serializer, etc.



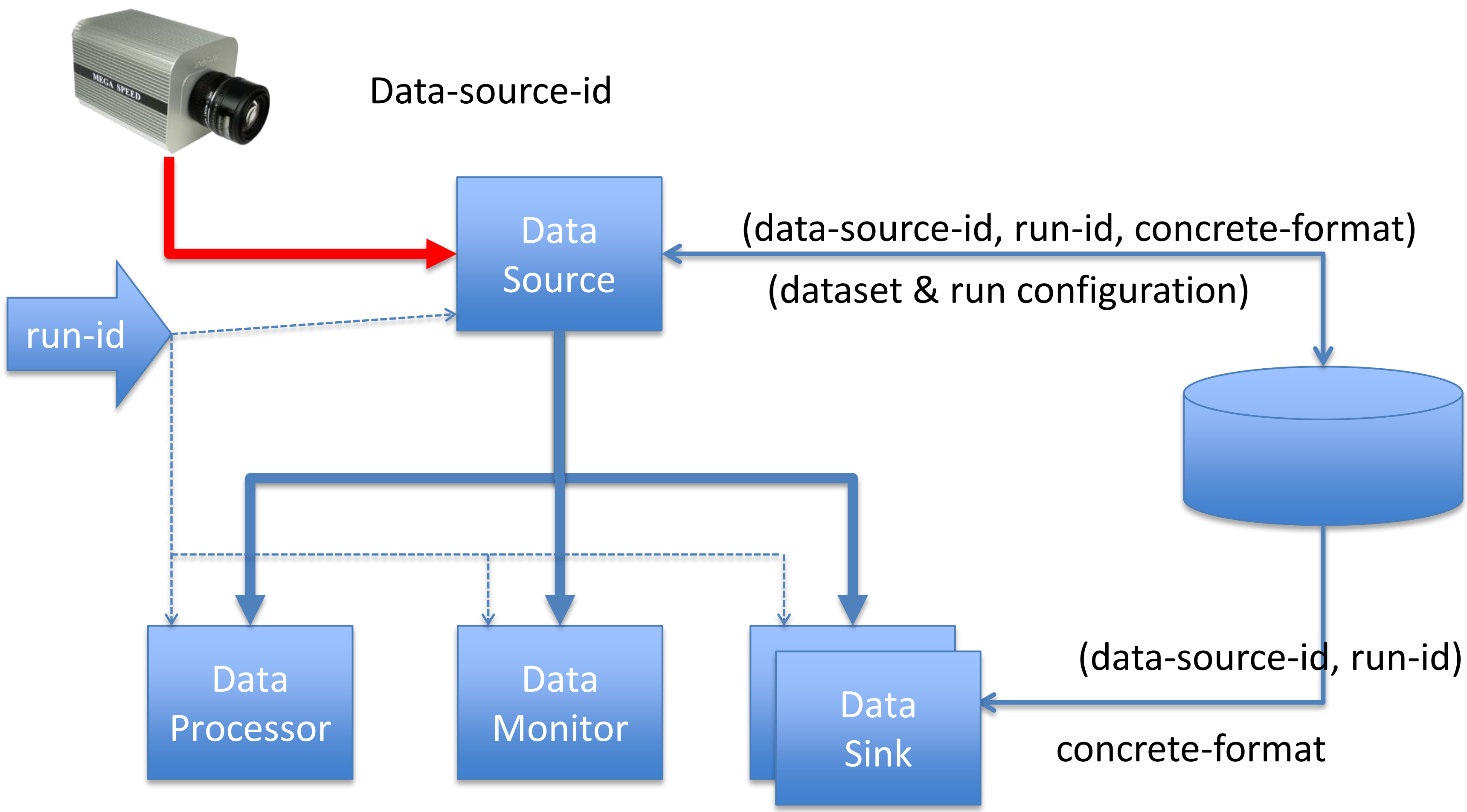
- We need a standard way to define the layout and format of binary data: structure, fields, etc.

What do we want?

- Purpose
  - Define a generic data format to organize data blocks exchanged between data sources and data sinks within our DAQ system.
  - Create a data format description formalism that allows generic tools to correctly understand data contents produced by different data sources.
  - Provide API for encoding binary data at the data sources and decoding it by data sinks.
- Three levels API:
  - Format: field size, type, …
  - Structure: order, layout, …
  - Representation: transformation, …

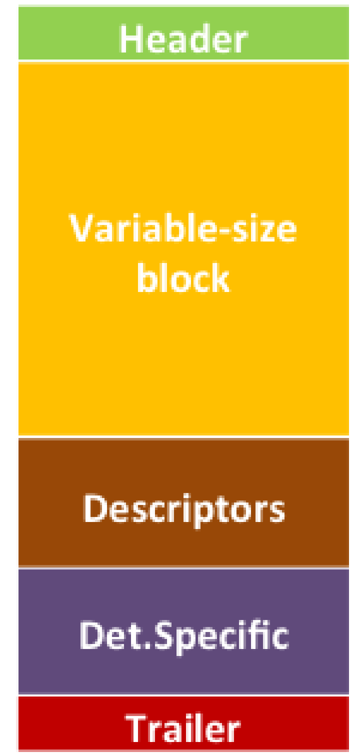


How does it work?



Features

- Features
  - Header and trailer sections are mandatory.
  - Specify multiple variable-size data sections within the same block, each section has different fields, and count.
  - Support for data fields with fixed, variable, and parametric length.
  - External and self referencing
- API
  - Append data function automatically arranges different data fields at the right place.
  - Implicit injection of length and pulse-id as descriptors
- For performance reasons, we still provide access to internal data in the buffer using native pointers.



Summary

- We built a slice test-stand that can handle a ½ MPxl detector
  - Select hardware and develop the DAQ software stack
  - Network and storage performance results were very acceptable
- Extend the concept of PC layer to cope with data coming from different data sources (2D detectors, monitors, fast digitizers, etc.)
  - Improve internal architecture that led to higher levels of reliability, scalability and flexibility
- PC layer pipeline concept
  - Plug-and-play data monitoring and rejection routines, and fast-feedback users' algorithms at runtime
- Data format description language and API for exchanging binary data between data sources and generic data digestion tools (serializers, data monitoring algorithms, etc.)
  - Proof of concept was done, now we are selecting use cases (1d digitizer, commercial camera, etc.)
  - Format specification document and API will be released by this summer