PAUL SCHERRER INSTITUT

PSI

Dominik Werder :: Paul Scherrer Institut

Streaming architecture at ESS

Experiment Control Workshop DMSC Copenhagen
December 8th 2016

PAUL SCHERRER INSTITUT

Streaming architecture at ESS

- Requirements for data streaming
- System components
- Technology
- Status and Performance
- ESSIIP infrastructure and detector streaming

Streaming Requirements

- Stream data from sources
  - EPICS
  - Detectors
  - ...
- Combine streams in Nexus files
- Robust API facing towards ECP
- Bandwidth up to 1.5 GB/s
- Redundant
- Scalable
- Extensible

Stream EPICS sources

EPICS process variables e.g.

- Motors
- Choppers
- Sensors
- Detectors
- Sample Environment
- ...

- Messaging system
- Wire format

Test platform: AMOR-sim

- Simulation of AMOR instrument
  https://bitbucket.org/europeanspallationsource/sinq-amorsim
- Simulation of motor controller
  (EL734 with EPICS IOC)

- Dornier chopper as used at AMOR
- Magnets
- AMOR event stream simulated from recorded histogrammed data files
- to come: Configuration Service
- Test and integration environment for prototyping the full solution
  - Precursor to test at the real instrument

PAUL SCHERRER INSTITUT

Streaming Components

- Data Sources
  - EPICS PVs
  - Neutron event generator
- Message broker: Kafka
- Forward EPICS PVs: as FlatBuffers into Kafka
- Nexus File Writer
- Experiment Control
- Mantid interface

PAUL SCHERRER INSTITUT

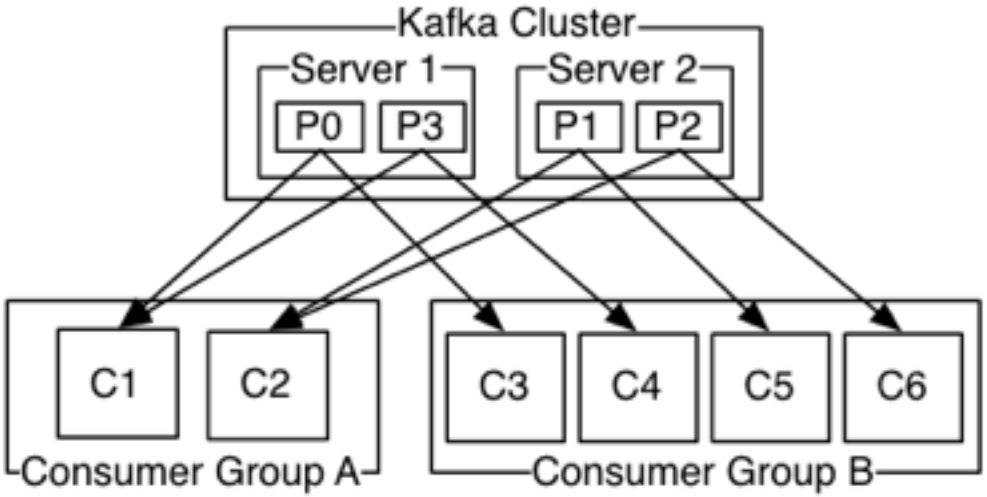
Message Broker: Kafka

- Persistent commit log
- Partitions (logs)
- Topics (sets of partitions)
- Guarantees:
  - Durable writes
  - Producer chooses commit frequency
  - Writes stay ordered within a partition
- Redundancy:
  - Replication of partitions (master, slaves: auto)

Message Broker: Kafka

Scalability

- Many Topics
- Topic's partitions can be on different machines
- Balance over Consumer Groups
- Single Consumer from group for each partition



[Kafka documentation]

PAUL SCHERRER INSTITUT

Message Broker: Kafka

Performance characteristics

- Linear I/O is as fast as it gets
- Independent of log size on disk
- Servers in cluster handle subset of partitions
- Producer responsible to balance over partitions
  - but built-in partitioners available
- Truncate log after time or size is reached
  - but also Compaction, keep last known key
- Scales well to many partitions, trade order
- Load balancing over consumers is dynamic

Message Broker: Kafka

Performance characteristics

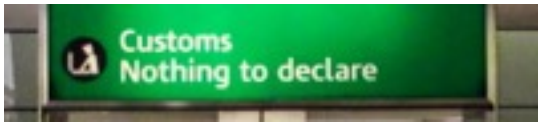
- Balancing over consumers:
  Message is delivered to each ConsumerGroup

Useful for us:

- Handles message passing
- Decouples actors in the system
- Load balancing (I/O, storage, CPU)
- Persistent buffer on disk

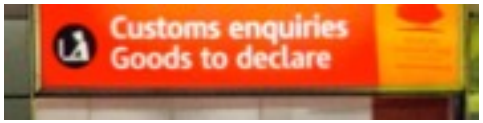
Wire Format: EPICS 4 pvData

- Flexible data types
  - PVStructure
  - PVScalarValue<T>
  - ...
- PVStructure gives no static guarantees
  - but some naming conventions



- Schema-free → introspection on access

Wire Format: FlatBuffers



- Pre-compiled schema
  Allows to trade flexibility vs. efficiency
- Easy to use toolchain (flatc)
- Less introspection needed compared to EPICS
- No allocations on read
- Slightly more verbose serialization code
- Efficient access to trusted buffers
  - No full parse required
- Static schema compile-time checked
- Flexibility via unions, optionals as special case

  → Runtime checks, only if asked for

Wire Format: FlatBuffers

- Access via offset pointers
  (no bounds checks)
- Verify untrusted buffers
  - Check if accesses stay within buffer

```
auto p1 = b->GetBufferPointer();
auto veri = flatbuffers::Verifier(p1, b->GetSize());
if (not VerifyPVBuffer(veri)) {
  throw std::runtime_error("Bad buffer");
}
```

Wire Format: FlatBuffers

- Runtime polymorphism on access:

```
builder.add_x_type(type);
builder.add_x(x.Union());
```

- Read:

```
F v_type() const {
  return static_cast<F>(GetField<uint8_t>(VT_V_TYPE, 0));
}
const void *v() const {
  return GetPointer<const void *>(VT_V);
}
switch (type) {
  case F::T: f(reinterpret_cast<const T *>(obj));
  ...
```

Forward EPICS to Kafka

- For general PV:
  Build recursive data structure.
- Very dynamic, introspection

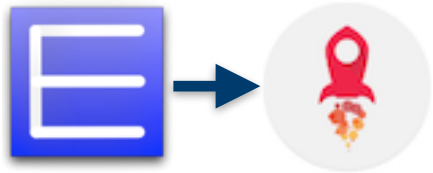
```
table PV { v: F; }

union F { pvByte, pvShort, pvInt, Obj, ... }

table Obj { ms: [ObjM]; }

table ObjM { k: string; v: F; }
```

Forward EPICS to Kafka

- neventGenerator: More static example

```
// Schema for neutron event data according to RITA2
table Event {
  htype: string;
  ts: ulong;
  hws: [ushort];
  ds: [ushort];
  st: ulong;
  pid: ulong;
  data: [ulong];
}
```

EPICS to FlatBuffers



- Message contains:
  - FlatBuffer schema id
  - FlatBuffer payload

- Currently very simple:
  | 16 bit schema id | …payload… |

- Schema id must be unique on the network
  - or indicated in topic settings
  - of course extensible with sub-id if need be

  https://bitbucket.org/europeanspallationsource/streaming-data-types
- Schemas should be able to identify data_source

PAUL SCHERRER INSTITUT

Forward EPICS to Kafka

- Repository:
  https://bitbucket.org/europeanspallationsource/forward-epics-to-kafka
- Monitor EPICS PV's
- Convert PV to FlatBuffer
  - general schema: cover all PV structures
    less optimal, but often good enough and easy
  - specialized schemas:
    more efficient if performance requires

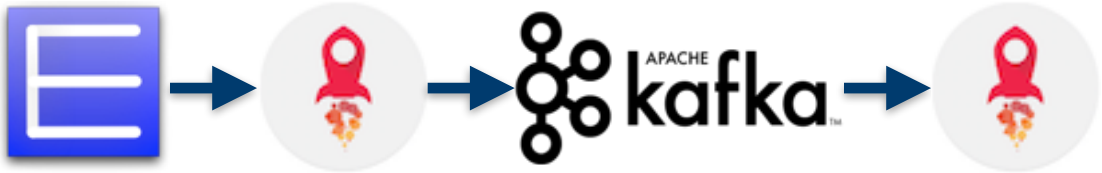
Forward EPICS to Kafka

- Configure via json:
  - file, topic on the broker, config api

```
{
  "broker-data-address": "host",
  "broker-configuration-address": "host",
  "mappings": [
    {
      "channel": "<epics_channel_name>",
      "topic": "<topic_name>",
      "type": "[general/chopper/specialized]"
    },
```

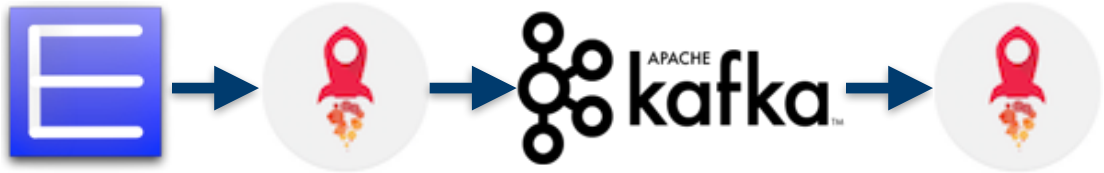
PAUL SCHERRER INSTITUT

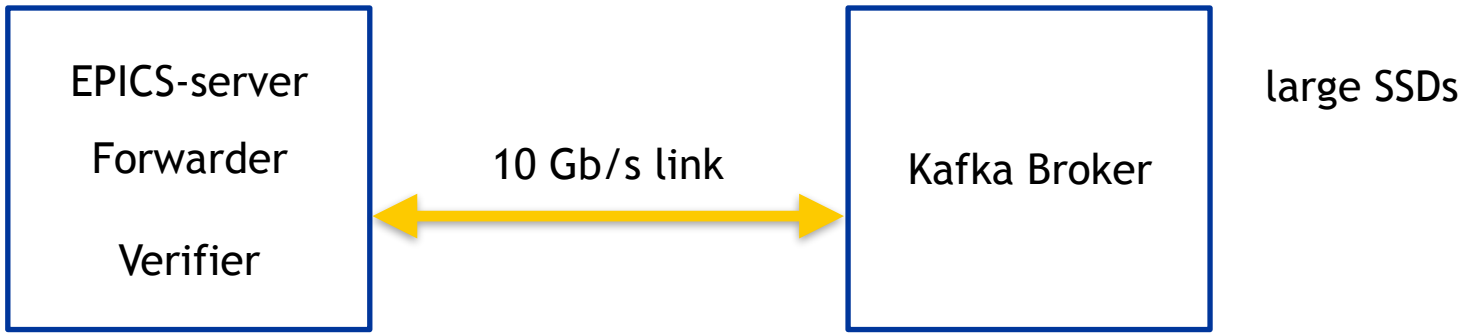
Performance



- EPICS test server
  - Vary PV size and update period
- forwarder-epics-to-kafka
  - Send as FlatBuffer to Kafka broker
- Kafka Consumer and FlatBuffer Verifier
  - Verify all packets arrive
  - Collect statistics
- Partitions: 5 (same number of writer / readers)
- Scheme: general

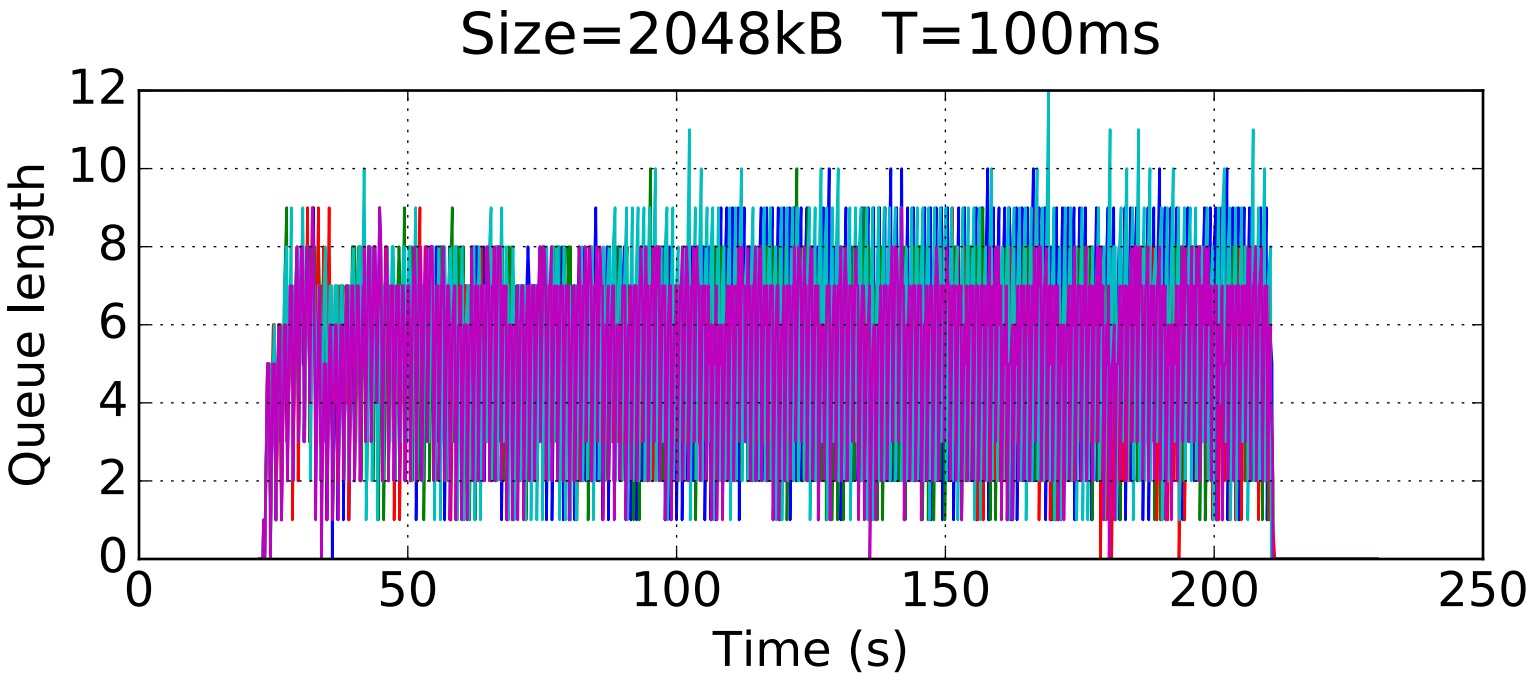
Performance





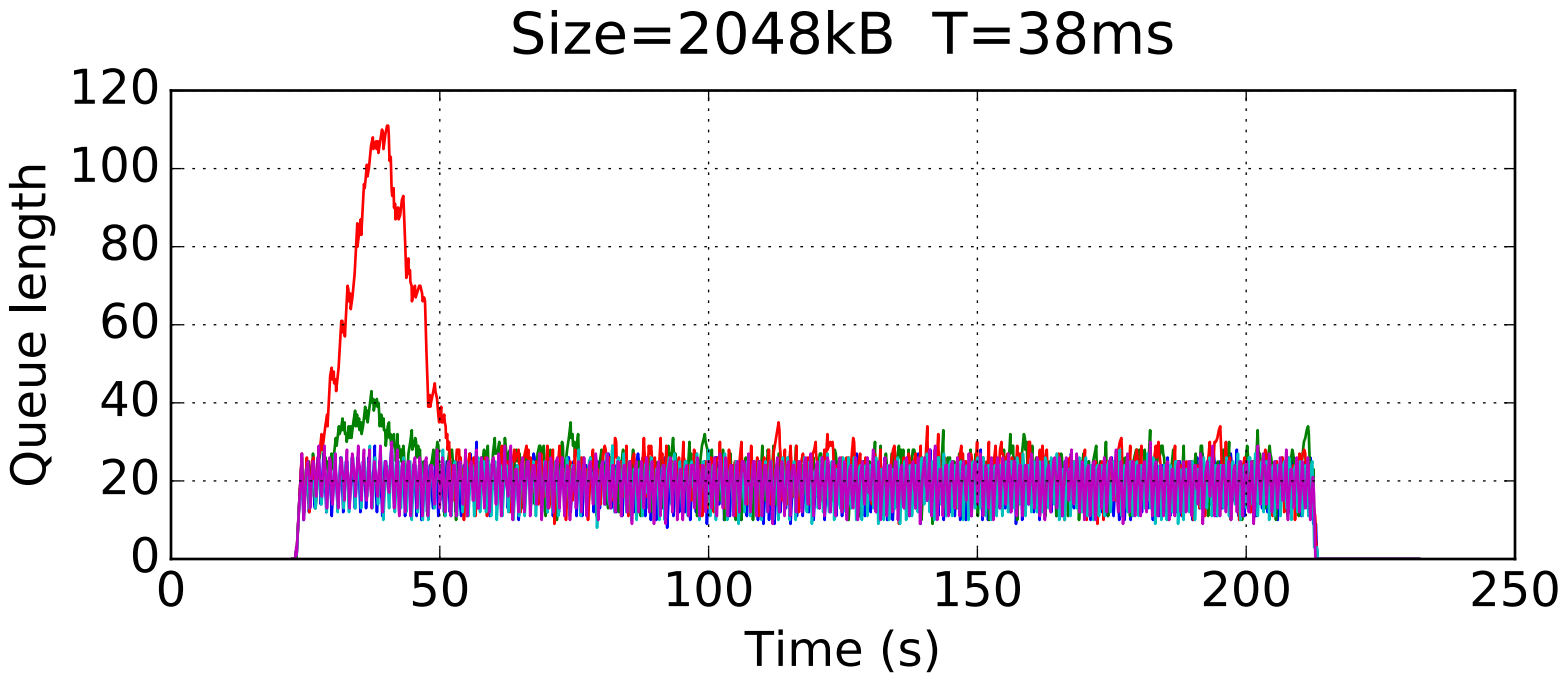
Performance

- Write: 100 MB/s
- All writers make progress
- Queue sizes stable



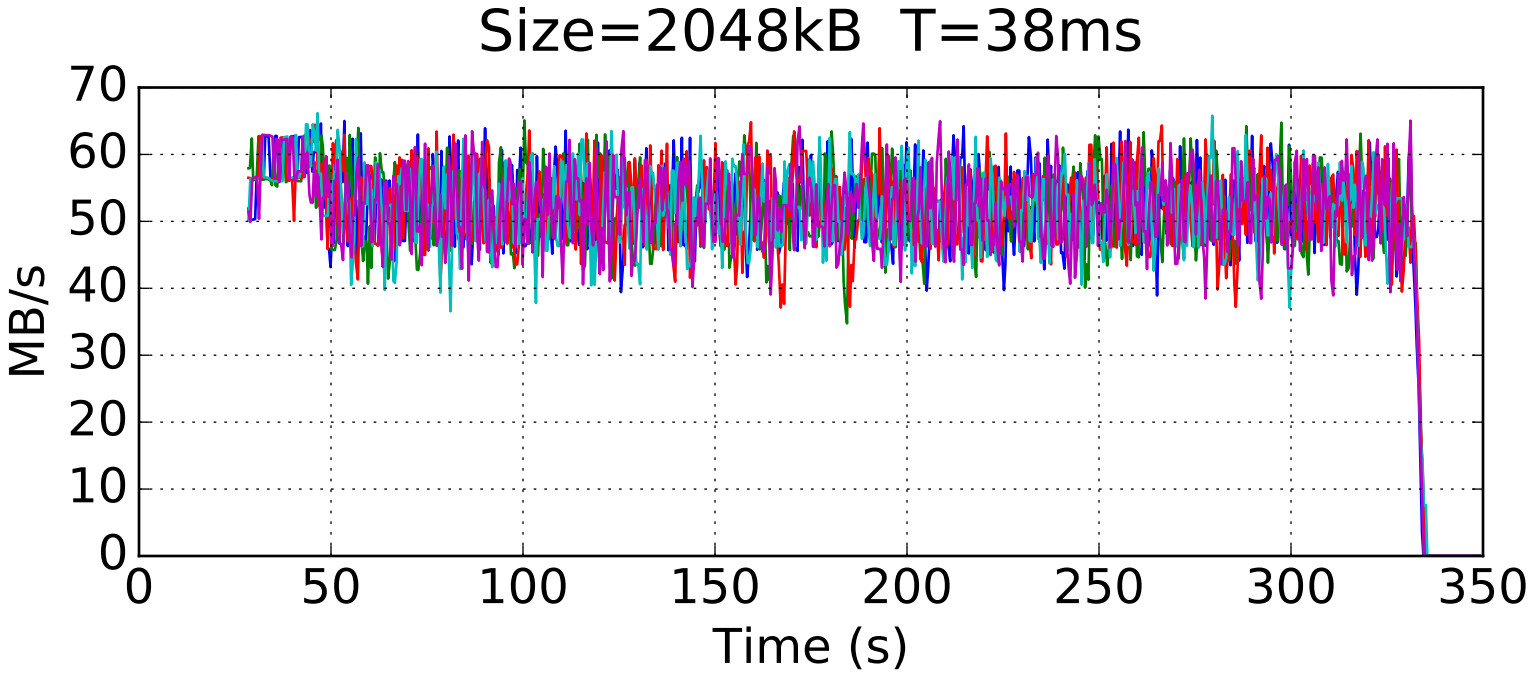
Performance

- Write: 260 MB/s
- Some queue during warm up
- Stable operation



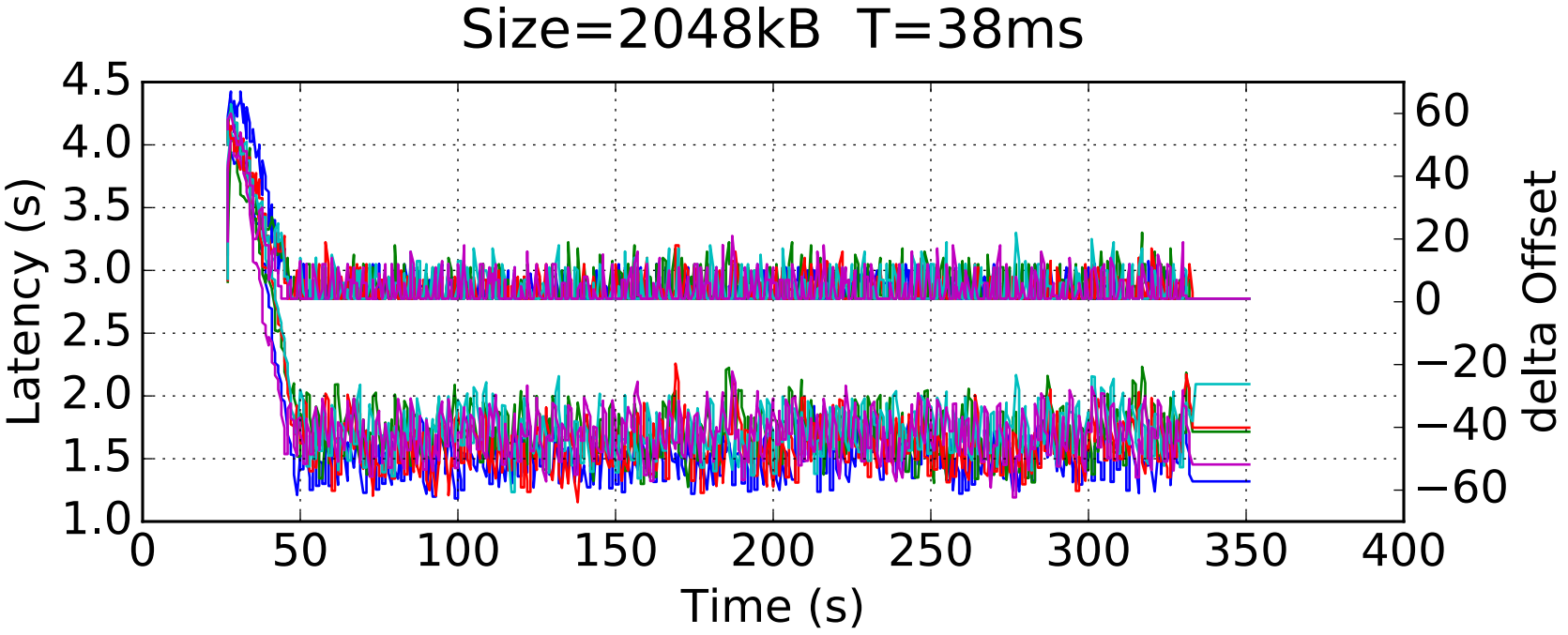
Performance

- Verification
- Load balanced over consumers
- FlatBuffer verify and check all payload arrives
- Collection of statistics



Performance

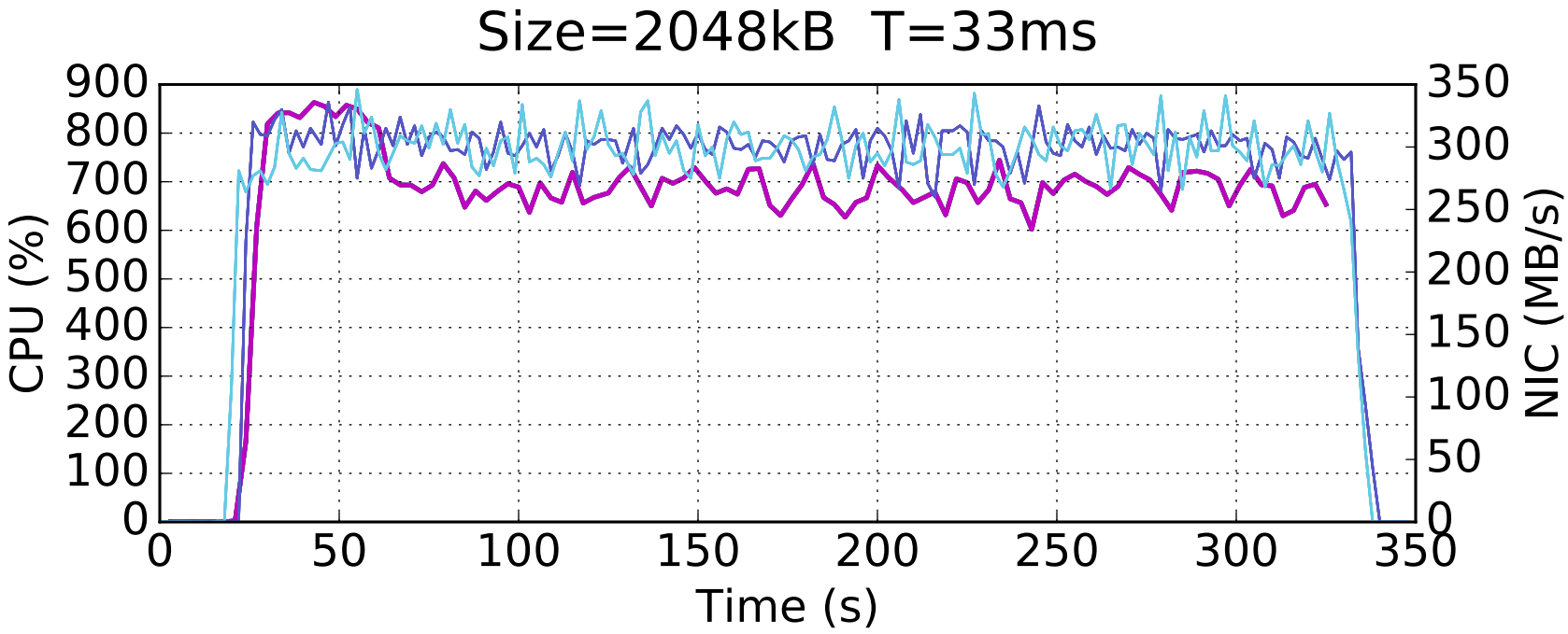
- Latency from EPICS production to verification
- Backlog on verification side



Performance

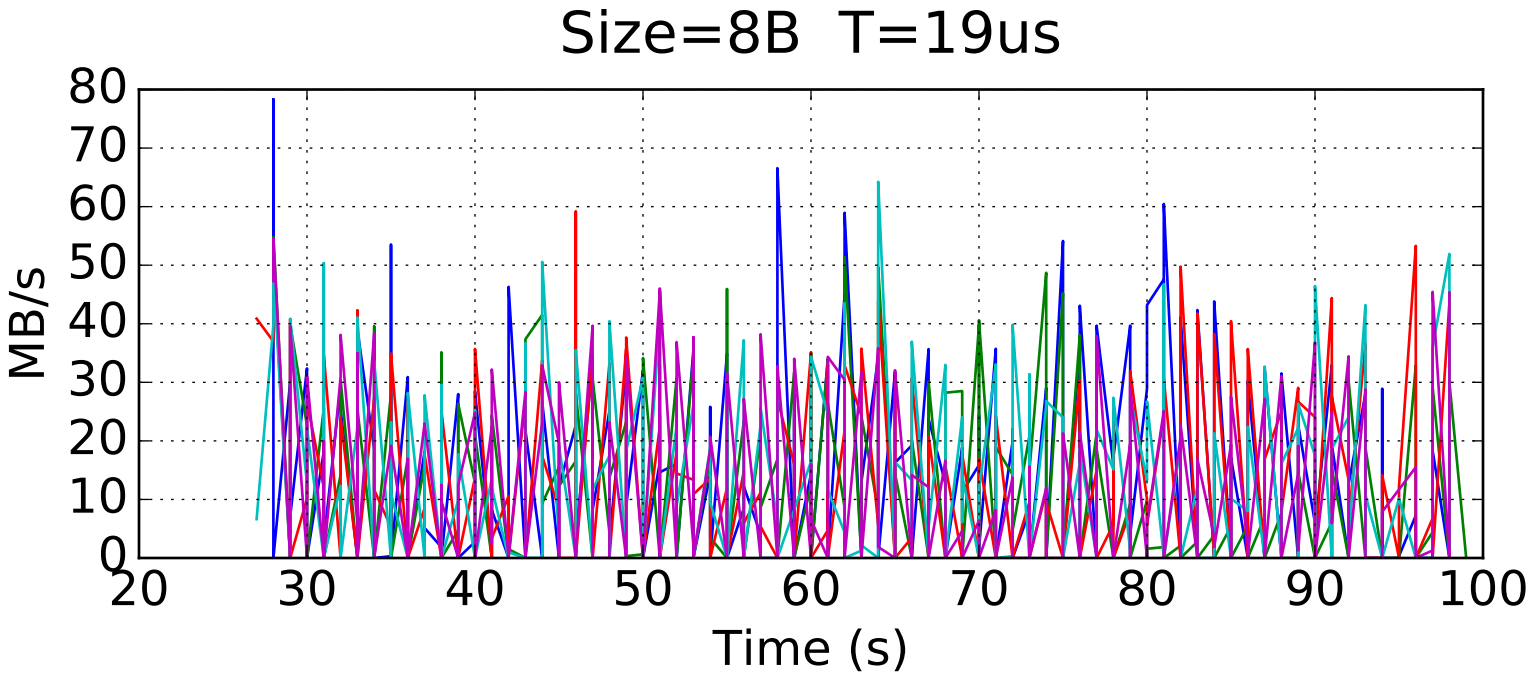
- How does the broker feel about that?

Note: Kafka broker is a plain vanilla install, could probably use some tuning...



Performance

- Small messages, 52 kHz updates from EPICS
- Processing in batches more pronounced
- Tunable via min/max on queues



PAUL SCHERRER INSTITUT

Forward EPICS to Kafka

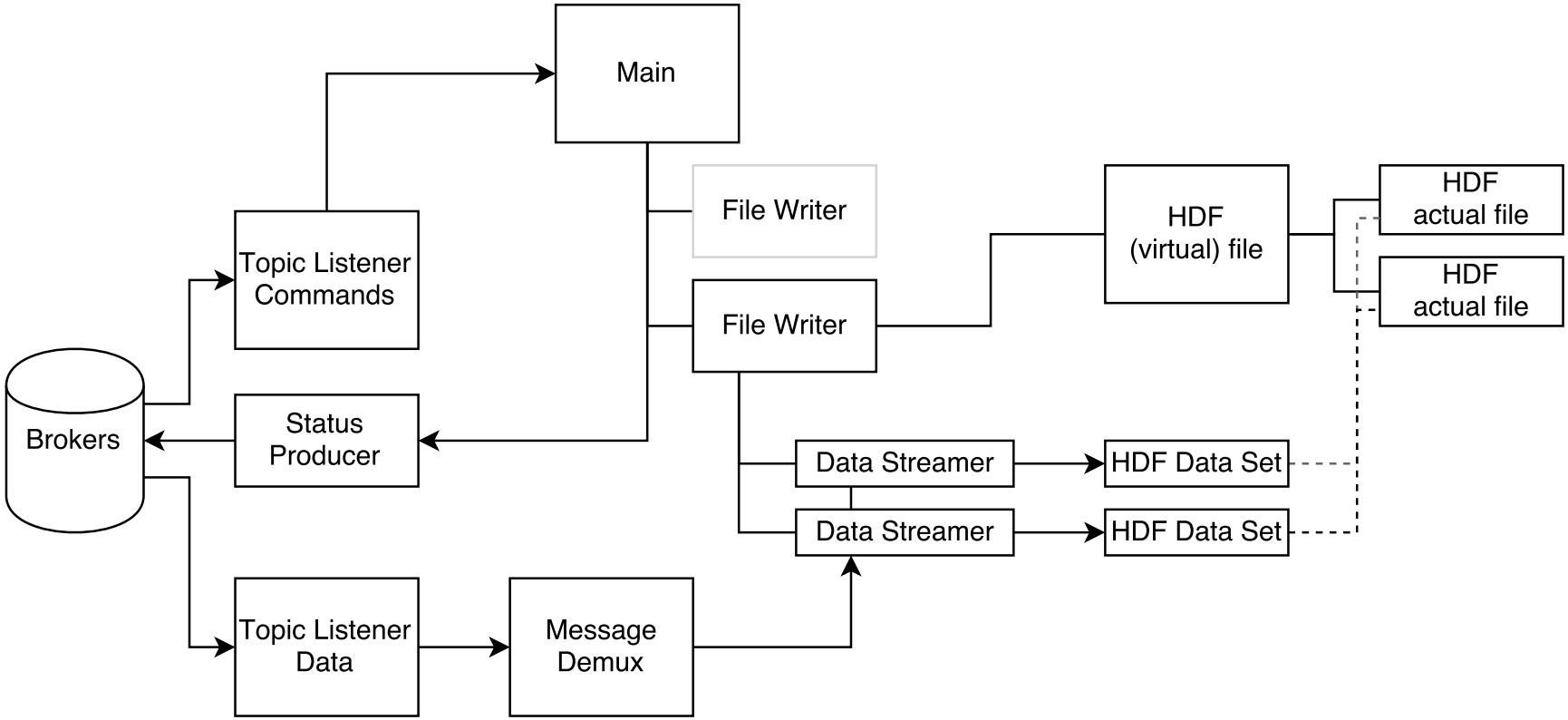
- Try hard to stay alive and make progress
  - Reconnect EPICS and Kafka
- Good performance even with general schema
- Future
  - Use multiple broker connections
  - Make all features available through config
  - Make features designed for testing optional
  - Test on real data sources

PAUL SCHERRER INSTITUT

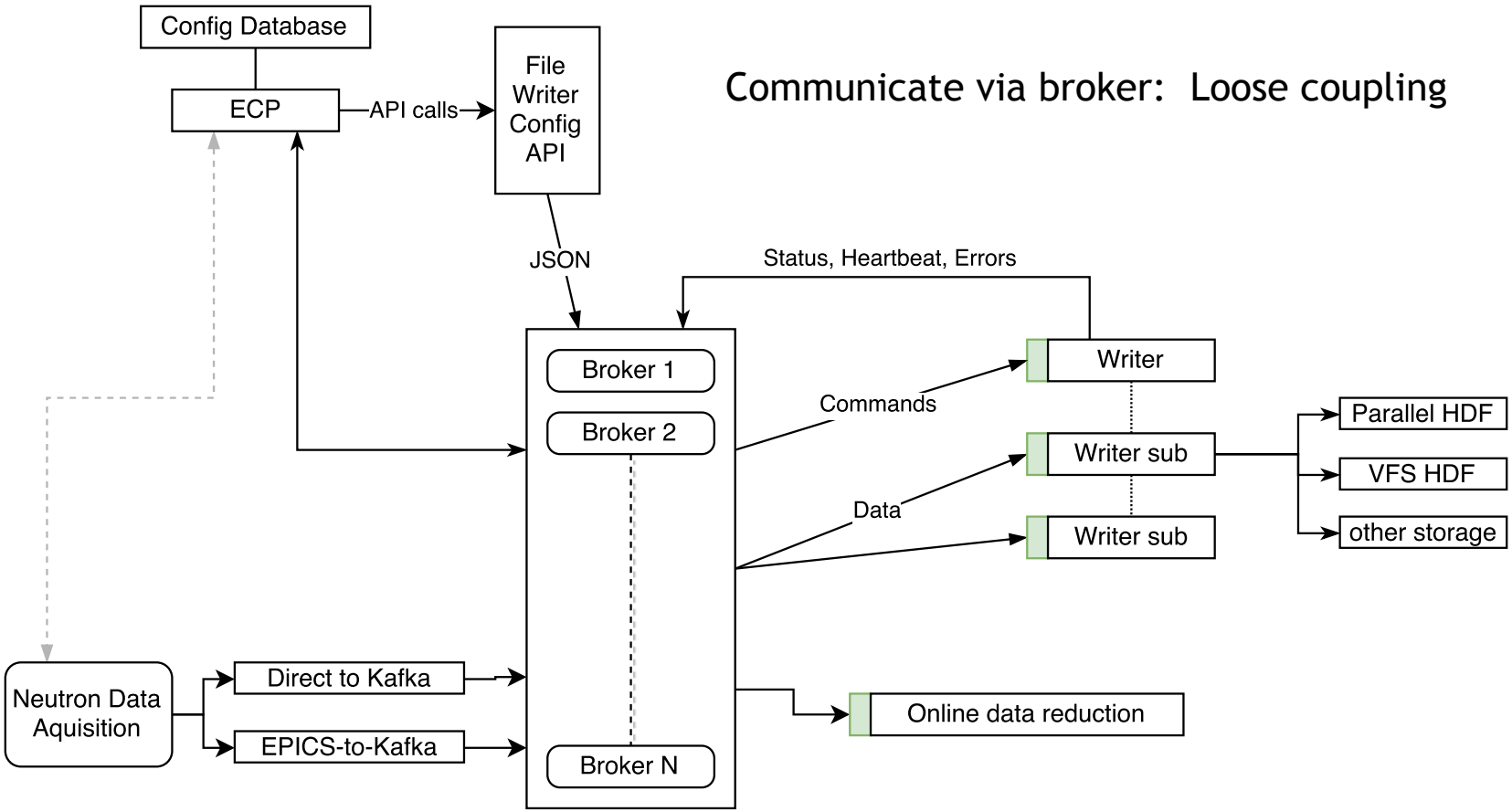
Nexus File Writer

- Start (logical) file
  - Metadata: directly from ECP
  - Create "DataStreams" and friends to subscribe to topics, care about HDF data set handles...
- Streaming
  - Introspect incoming for type, "data_source" and hand off to the streams
  - Write, emit status and metrics as heartbeat
- Next file: With changed metadata
- Stop from ECP, preset timeout if ECP fails
- Scalability, support from HDF

Nexus File Writer



Infrastructure Overview



Communicate via broker: Loose coupling

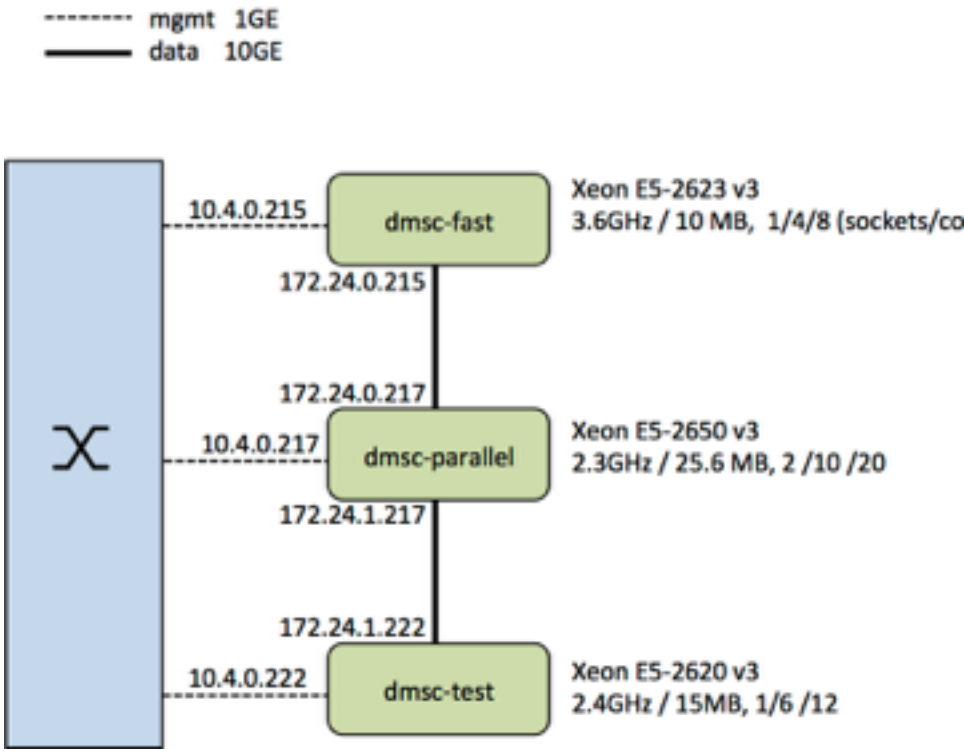
ESSIIP infrastructure

on behalf of Afonso Mukai et al.

- Infrastructure at DMSC:

  https://ess-ics.atlassian.net/wiki/display/DMSC/ESSIIP

  https://ess-ics.atlassian.net/wiki/display/IS/ESSIIP+Operations+Status
- Integration of projects from different groups
- 10 Gb/s pairwise
- Kafka broker
- Nexus streamer
- AMOR-sim
- EPICS forwarder
- HDF test writer
- Fast sample IOC

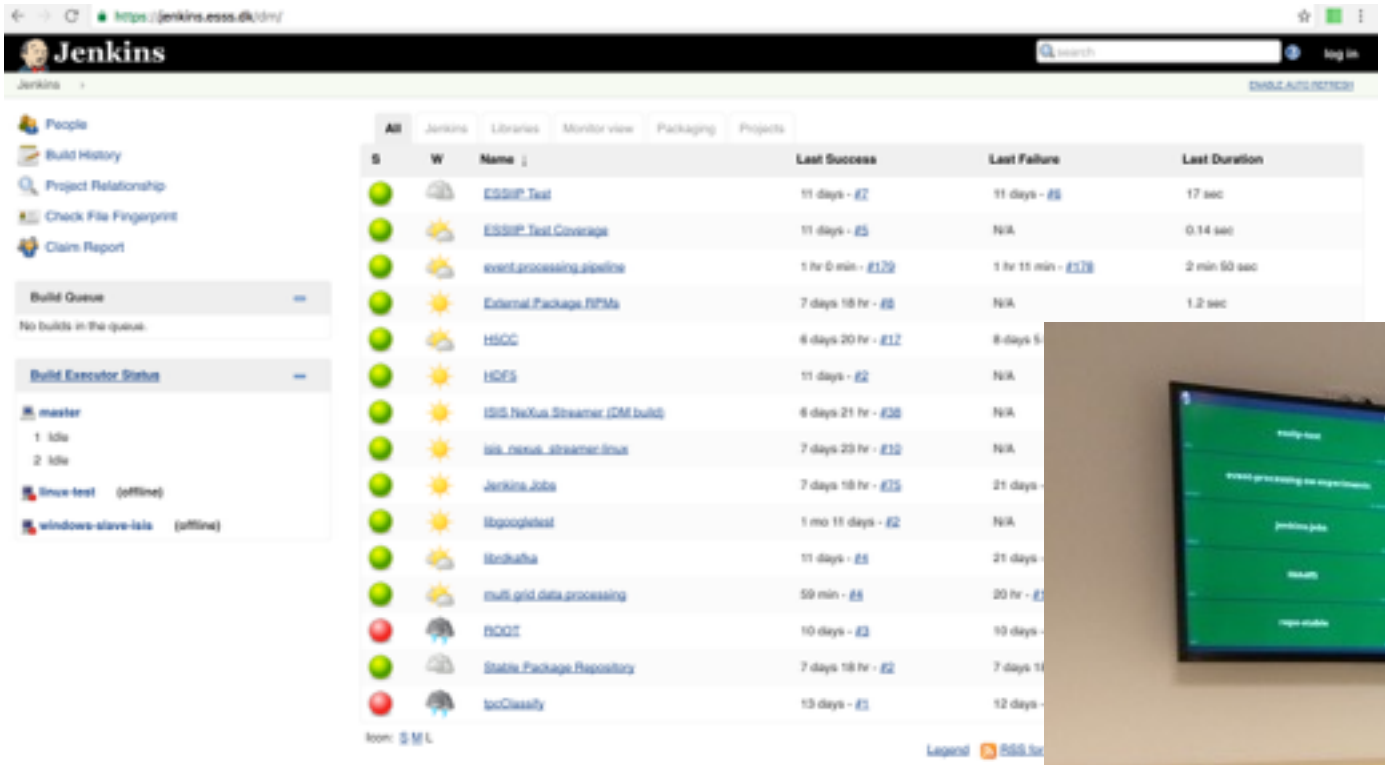


ESSIIP infrastructure

on behalf of Afonso Mukai et al.

- Jenkins build server:

  https://jenkins.esss.dk/dm/



PAUL SCHERRER INSTITUT

Area detector streaming

on behalf of Jonas Nilsson et al.

- Simulated areaDetector at ESSIIP-lab
  https://ess-ics.atlassian.net/wiki/display/IS/ESSIIP+Operations+Status
- Expected data rates for detector up to 500MB/s
- Plugin for the EPICS areaDetector architecture
- Serializes data using flatbuffers.
- Currently write speed to Kafka up to ~100MB/s
- Requires unit tests and better error handling before it is ready for a production environment.
- Current version here:
  https://bitbucket.org/europeanspallationsource/m-epics-kafkaplugin