Data model update

M.Krzewicki, FIAS

for CWG4

outlook

- In-memory data layout & transport layer.
  - basic structures, conventions and agreements.
  - FairMQ support of required features.

- Transitional (timeframe-) data formats.

- Status of implementation.

- Milestones

Basic principles

- Constraints:
  - Each processing step (device) independent.
  - API is the message exchange format between devices.
  - Everything is in memory.
  - Many different data types from many sources.
  - In principle every data buffer at a random location in memory.
- Goals:
  - Marry the many data fragments belonging to a single unit (timeframe).
  - Avoid data duplication (and copying).
  - Keep the schema extensible.
  - Hide as much logic as possible behind easy to use interfaces.

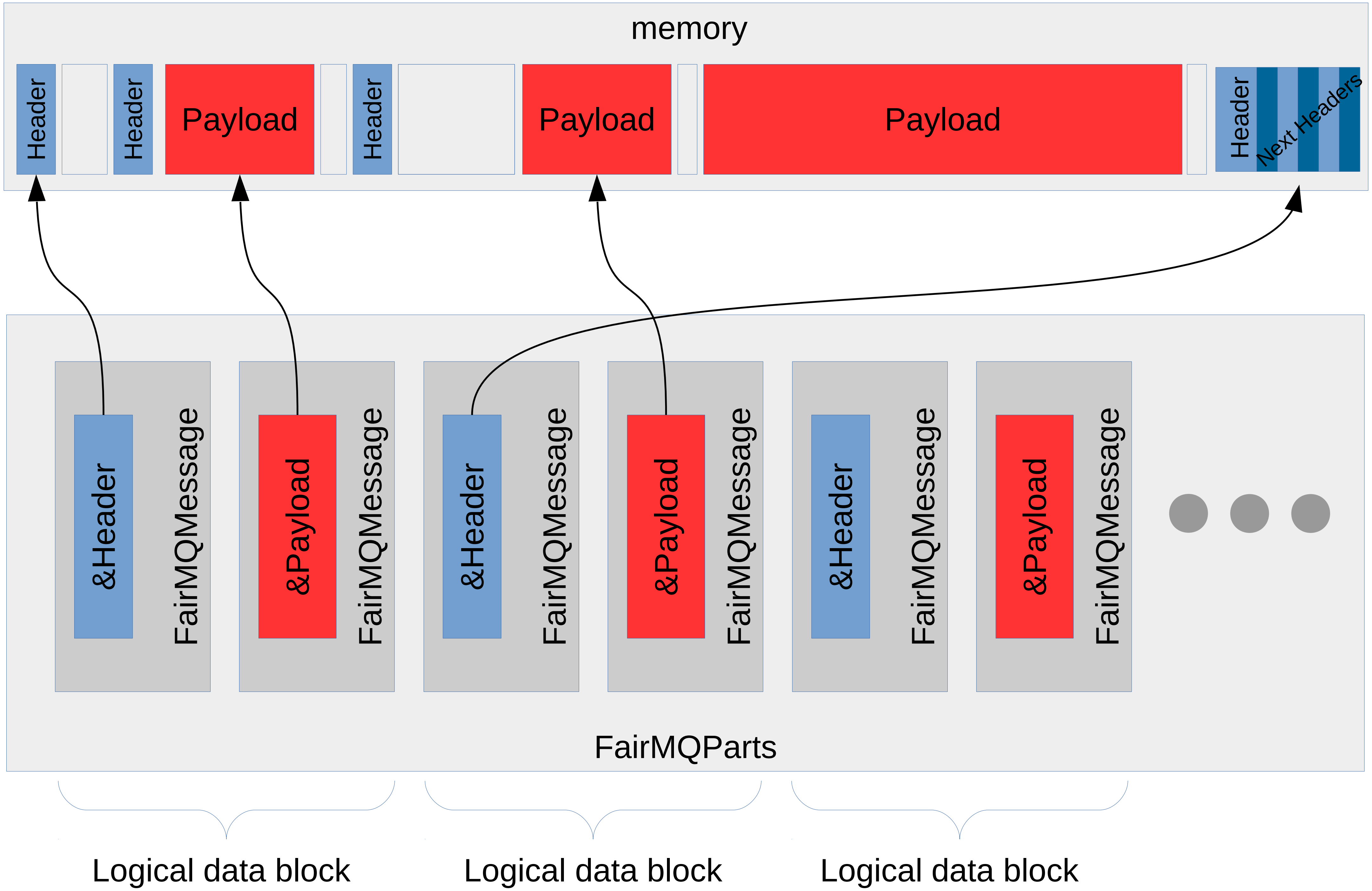
Outcome of discussions so far

- The format consists of header information and payload. The latter is not touched by the framework.
- Multi-part approach is used for the transport, implemented by the transport framework (FairMQ) → Format consists of a sequence of separate parts (header and payload in separate parts).
- Each data part is preceded by a corresponding header part
  Header part supports an extensible header stack.

- FairMQ supports all of the above.
  - still some tweaking of APIs, as expected.

- Discussions on memory management:
  - proposal to allow FairMQ to do full buffer management to allow transparent use of e.g. shared memory.

In memory layout of the O^2 message (timeframe)



- Buffers with data at random locations in memory.
- A zero-copy approach (with some transports, handled transparently by FairMQ)
- Order of message parts is preserved, we rely on convention to group the header-payload pairs.

General header format

- Starts with *basic header information*, never serialized, with unique version number (detailed format in the following slides).
- Enforce *strict policy*: no changes to members (e.g. width) or sequence of members, new members can be appended, new version number.
- All basic header structs are defined with fixed endianness and padding.
- Handlers for inhomogeneous systems will be provided at compile time.
  - Strategy: "keep concept open for new ideas but do not solve problems we don't have at the moment."

- Header-stack concept: optional headers can follow the basic header.
- A next header is indicated in a flag member of preceding header
- Optional headers consist of a fixed NextHeaderDescription followed by the NextHeaderContent

Header definition

```
struct DataHeader
{
  //a magic string
  char      magicString[3+1];
  //origin of the data (originating detector)
  char      dataOrigin[3+1];
  //serialization method
  char      payloadSerialization[7+1];
  //data type descriptor
  char      dataDescription[15+1];
  //sub specification (e.g. link number)
  uint64_t  subSpecification;
  //flags, first bit indicates that a sub header follows
  uint32_t  flags;
  //version of this header
  uint32_t  headerVersion;
  //size of this header
  uint32_t  headerSize;
  //size of the associated data
  uint32_t  payloadSize;
}
```

- extensions: only new members at the end OR header stack
- for more details see e.g. https://indico.cern.ch/event/491190/

Header stack

- Design an open concept of headers which can accommodate extension requests like those we had in the past (e.g. more trigger bits, new detectors, more detector links).
- Indicate in the header flags that there will be a next header coming immediately after the current header.
- Additional headers consist of basic header information NextHeaderDescription and following that NextHeaderContent (in a single buffer)
- Header payload can be serialized

```
//____________________________________________________________________________
struct NextHeaderDescription
{
  // size of this next header description
  int32_t size;
  // size of the next header payload
  int32_t payloadSize;
  // serialization method
  char    serializationMethod[7+1];
  // header contents description
  char    headerDescription[15+1];
  //first bit indicates there is a next one after this
  int32_t flags;
};
```

Header and payload navigation

- The logical structure of the message (timeframe) depends on framing and convention (separate headers and payloads)
  - Tools for easy access and navigation are being developed
    - Using standard C++/stl tools and concepts (iterators etc.).
    - Similar approach to navigate the timeframe and the header stack.
    - First useable interfaces should be there soon.
    - development happens here:

      https://github.com/mkrzewic/AliceO2/tree/dev
      https://github.com/matthiasrichter/AliceO2/tree/dev-format/format

Transitional (test-) timeframe format

- Definition of the actual persistent format postponed.
- User code (reconstruction, calibration) implemented as FairMQDevice only sees the message, everything is already in memory.
    - A file reader (sampler) device can easily be written with support for any on-disk format.
    - First idea is to just use separate raw data files for every detector (link) - a la derooted raw files.
    - Implementation after the message navigation API ready.
    - Detectors can provide the raw data already now.

CWG4 milestones

- Jan 2016:
  - ~~Assessment of available manpower.~~
  - ~~Task definitions and assignments.~~
  - ~~First proposal AOD format.~~
  - ~~First proposal QA formats.~~
- Q1 2016
  - ~~In memory data layout.~~ (agreements reached, implementation ongoing)
  - ~~Transitional file format for test time frames.~~
- Q2 2016:
  - Prototype persistent format time frame and compressed time.
  - Prototype AOD format.
  - Prototype of auxiliary formats (QA etc.)
- Q3: Data model/interface implemented and used in further prototyping.