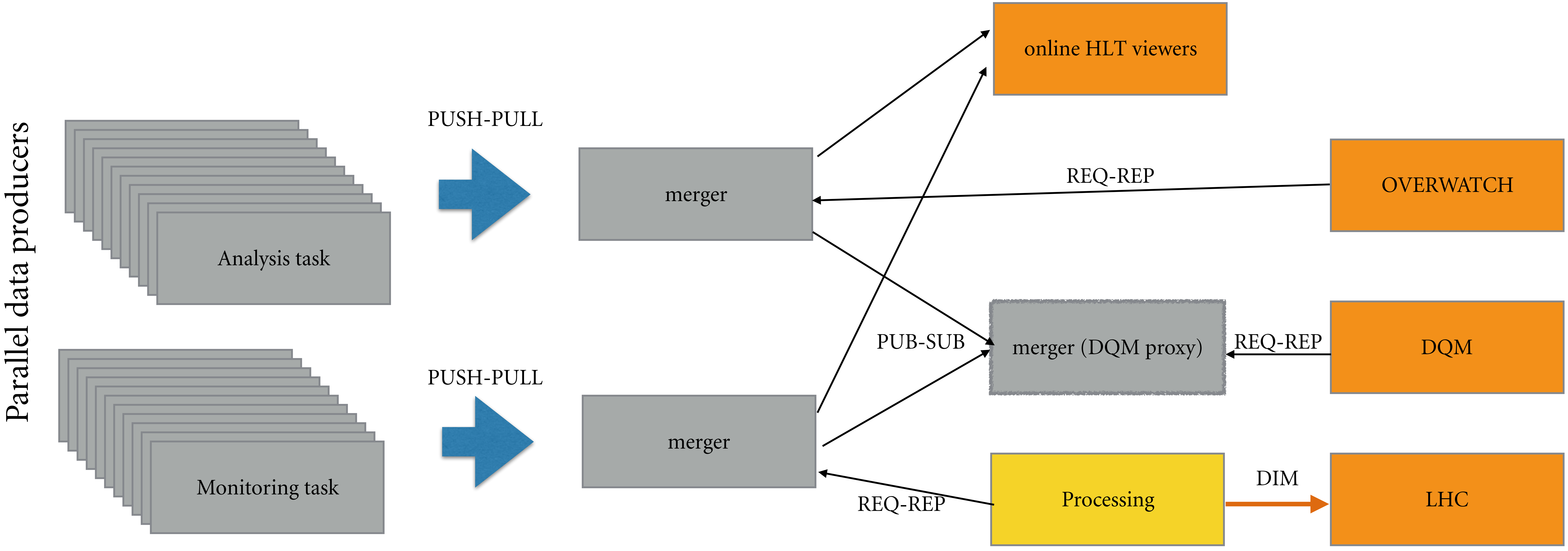
Real-time ROOT object merging in the HLT

M.Krzewicki

FIAS

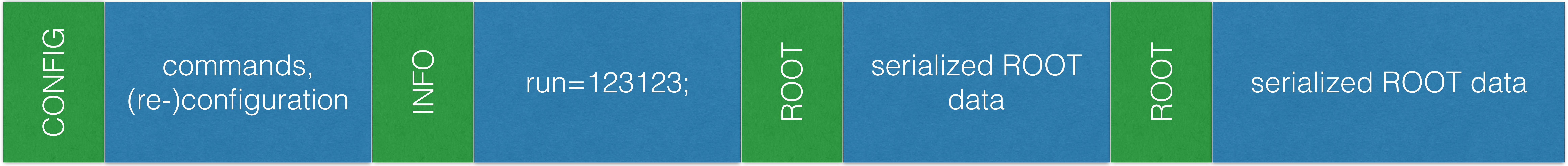
Simplified dataflow schema



- Data producers push data to mergers continuously (almost).
  - Analysis tasks are offline QA/calibration tasks running in AnalysisManager-like env @HLT (*)
  - Monitoring tasks is usually code written to run synchronously in the HLT framework.
- All communication is handled by ZMQ (via AliZMQhelpers lib) - following a simple data model.
- O2-like devices exchanging data asynchronously.

Data model

- Data is contained in a multi-part ZeroMQ message.
- Each part is a binary buffer, annotated by a header (headers are separate parts).
- Header determines the data type:
  - ROOT - a ROOT object, to be deserialized and merged.
  - CONFIG - A configuration/command string.
  - INFO - some metadata, currently run number, HLT running state
    - formatted as a “;” delimited string of “<key>=<value>” pairs (is a subset of the ECS string).
  - ROOTSTRI - ROOT schema information.
- Binary compatible with current O2 data model - compatible with O2 devices at data exchange level!
- Single message can contain any number of ROOT objects, other data etc.



CONFIG	commands, (re-)configuration	INFO	run=123123;	ROOT	serialized ROOT data	ROOT	serialized ROOT data

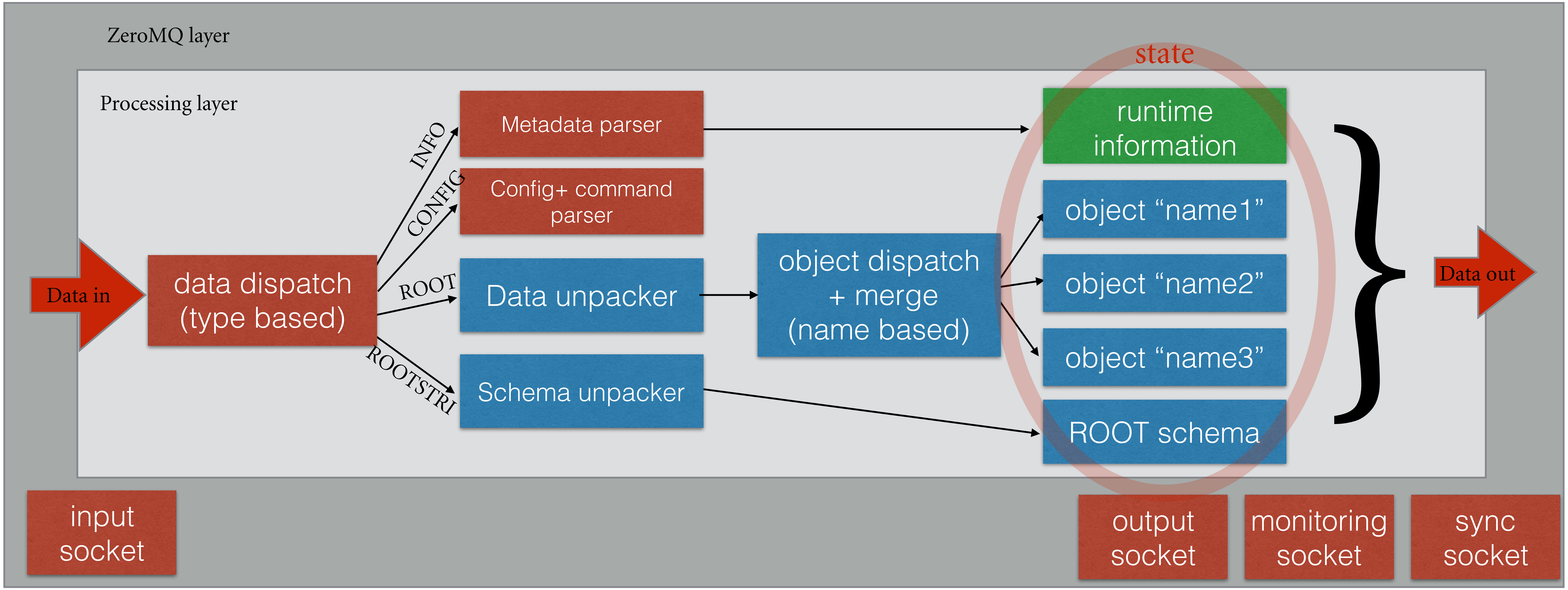
ZMQROOTmerger

- Fully asynchronous, data driven architecture.
  - main event loop steered by ZeroMQ events (data in).
  - Everything is data, including configuration and commands.

- Usually 4 ZMQ sockets defined (messaging pattern chosen at configuration time):
  - data in (usually PULL or SUB)
  - data out (usually PUSH or PUB)
  - monitoring (usually REP or PUB)
  - {*sync (PUB or SUB only), redundancy for condition changes (EOR,SOR etc.)*}
- Data, configuration and commands can come in(or out) on any of these (except sync).

- Acts as a server either replying to requests or pushing data at specified intervals.

Merger architecture



- Asynchronous, data driven processing layer is always in a consistent state (no need for an explicit state machine).
- Incoming objects are merged into the state (merger state is metadata + merged data) one-by-one using name matching.
- Data OUT must be triggered:
  - externally by a CONFIG data block with a send command (ex. a thread that triggers a send on an OUT socket periodically).
  - by sending a REQuest - by default the reply will contain the full merger state (unless it contains a CONFIG block with other instructions).

Constraints for data producers

- Data accumulated at producer level for a prescribed time duration (~10s)
- When pushing data out, the producer NEEDS to reset (drop all data) - data is MOVED to the merger, not copied.
  - otherwise we would be merging same data many times (or need some complicated logic).

- For stuff deriving from AliAnalysisTask user must overload ResetOutputData().
  - Tasks ported to use the "V" virtual interfaces to ESD/AOD data (there is no AliESDEvent in the HLT).
  - Steered by the HLT framework (AliHLTAnalysisManager) for QA and calibration tasks.
  - this is not yet in master, has been running online for over a year, code in ALICEHLT AliRoot and AliPhysics dev and prod branches.

- HLT framework always adds run metadata to each message.

Input/output data considerations (for merging)

- Input data is:
  - TH1, any number of those.
  - TH1 wraped in (nested) TCollections (TList, TObjArray).
  - User objects (e.g. outputs of AliAnalysisTask, people put anything in there), usually in a structure of TCollections.
  - We always merge like-named objects - object names UNIQUE!

- Output data:
  - Depends on the use case:
    - Same structure as input data (default).
    - Unpacked histograms (and other drawable objects) - makes life on the (QA) processing end easier
      - also performance benefits (see later slides).

ROOT object merging caveats

- Built-in ROOT merging mechanism using TMethodCall slow, using the interpreter (also reported slow on ROOT 6).
- Better solution: use RTTI (we use dynamic_cast, although builtin ROOT RTTI is a bit faster!).
  - dynamic_cast<TH1*>, then call TH1::Merge()
  - What to do with custom objects?
    - Derive from AliMergeable and overload Merge().

- TCollection::Merge() falls back to slow TMethodCall!
  - solution: unpack first (that is fast) then merge unpacked objects (ideally all TH1 and AliMergeable).
  - PROBLEM: TCollection ownership! A non-owning collection stays non-owning after transport -> in general this means mem leak.
  - SOLUTION: AliHLTList and AliHLTobjArray - become owner after deserialize.
    - they are just TList and TObjArray, but with safe streaming behaviour.

- Custom objects NEED to be streaming safe AND not leak memory - it is unfortunately out of our control.
  - Policy: if you leak, you're out!

Unpacking objects

- In reality we always unpack QA objects to have a consistent and easy to visualize data set (histograms only).
- User objects overload AliMergeable::GetListOfDrawableObjects() to aid unpacking (if wanted).
- when unpacking we rename objects:
    - TCollections have a name.
    - unpacked objects renamed to: “<collection name 1>/<collection name 2>/…/<object name>”
    - Path-like, easy to parse.
    - Easier to ensure name uniqueness (only unpacked and renamed objects are dispatched to mergers).
    - When unpacking (recursively) we get to all levels TCollections and can clean up properly.
        - Without unpacking this mostly leads to mem leaks.
        - Better to use special stream safe HLT variants (AliHLTList, AliHLTObjArray).

Other features

Bunch of useful features which proved best to be implemented in the merger itself:

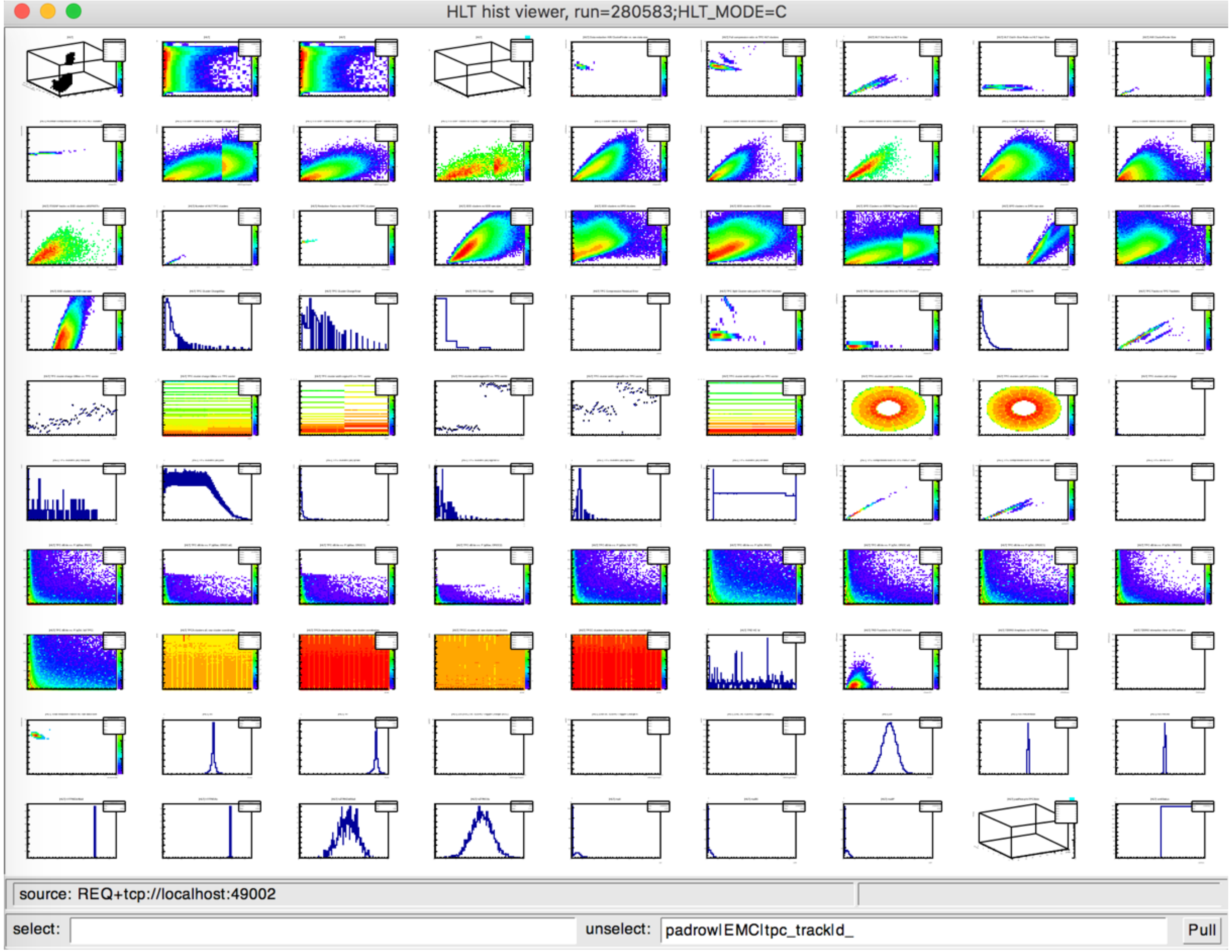
- Proxy mode: incoming objects replace old ones instead of being merged.
  - Used for DQM proxy - DQM can not dynamically adjust during run, needs full list of histograms at SOR.
  - Histograms are cleared at some condition (e.g. change of run) instead of deleted.

- State is persistent:
  - When killed or restarted the state is persisted on disk and loaded automatically - no loss of statistics.

- Regex object selection - a subset of objects can be sent (on per-request basis), no need to eat too much bandwidth all the time.

Example

- Subset of data available in the DQM merger (proxy) - selection regex visible below in the window.
- Metadata displayed in window title.



HLT hist viewer, run=280583;HLT_MODE=C

source: REQ+tcp://localhost:49002

select:

unselect: padrow|EMC|tpc_trackId_

Code

- The ZMQ infrastructure (including merger, histogram viewer, dummy histogram producer, examples, etc.) is in current master (needs zeromq to compile).
    - helper lib: HLT/ZMQ/AliZMQhelpers.h
    - merger: HLT/BASE/util/ZMQROOTmerger.cxx
    - viewer: HLT/BASE/util/ZMQhistViewer.cxx
    - dummy histogram producer: HLT/BASE/util/ZMQhistSource.cxx
    - binaries installed in $PATH: ZMQhistSource, ZMQROOTmerger, ZMQhistViewer
        - run without arguments for options.
    - example script showing the async features: $ALICE_ROOT/HLT/exa/exampleZMQchain.sh
        - starts a number of data sources, a proxy, a merger and a viewer.