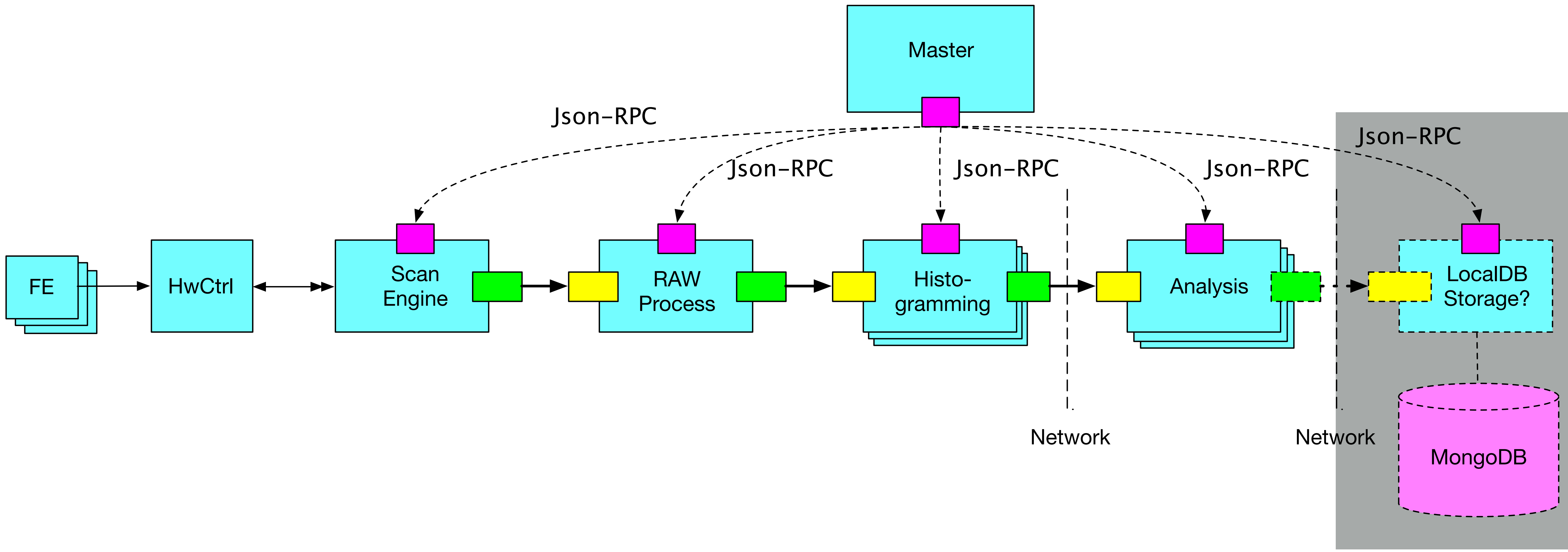
Review of devel_Distributed branch

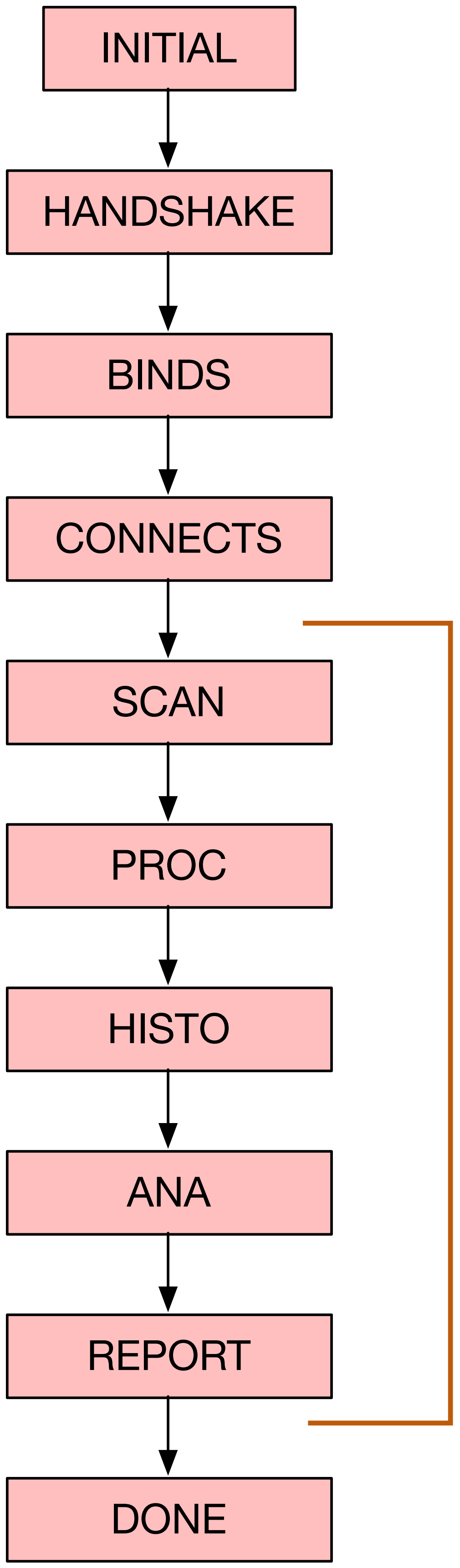
Hide

Communication model



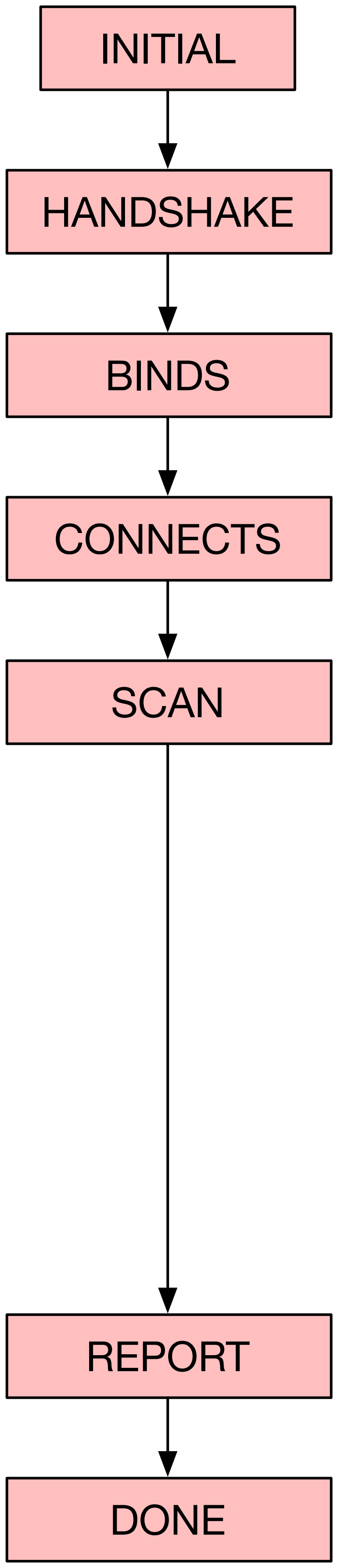
- Master process only lives for a single scan. It terminates for each scan.
- Slave processes do not die — they are always standing once launched. (details are re-configured for each master launching)
- Can insert network boundary for large-scale operation.
- Missing piece: result data accumulation and localDB storaging can be also pipelined (mind the file system boundary).

Master process FSM



- FSM: currently a waterflow model.
- As soon as the previous step is over, the next step is triggered.
- In each step, a signal is emitted from master to slaves, and slaves work for the specified task of the step.
- Each slave responds a finishing message once the task of the state is done.
- This is not a pure pipelining — for instance histogramming slave does not need to wait for master's signal of **`HISTO`** starting — as soon as the data is pushed from upstream the histogramming slave should start the task and send the result to downstream.
- **Need rework here?**

Master process FSM

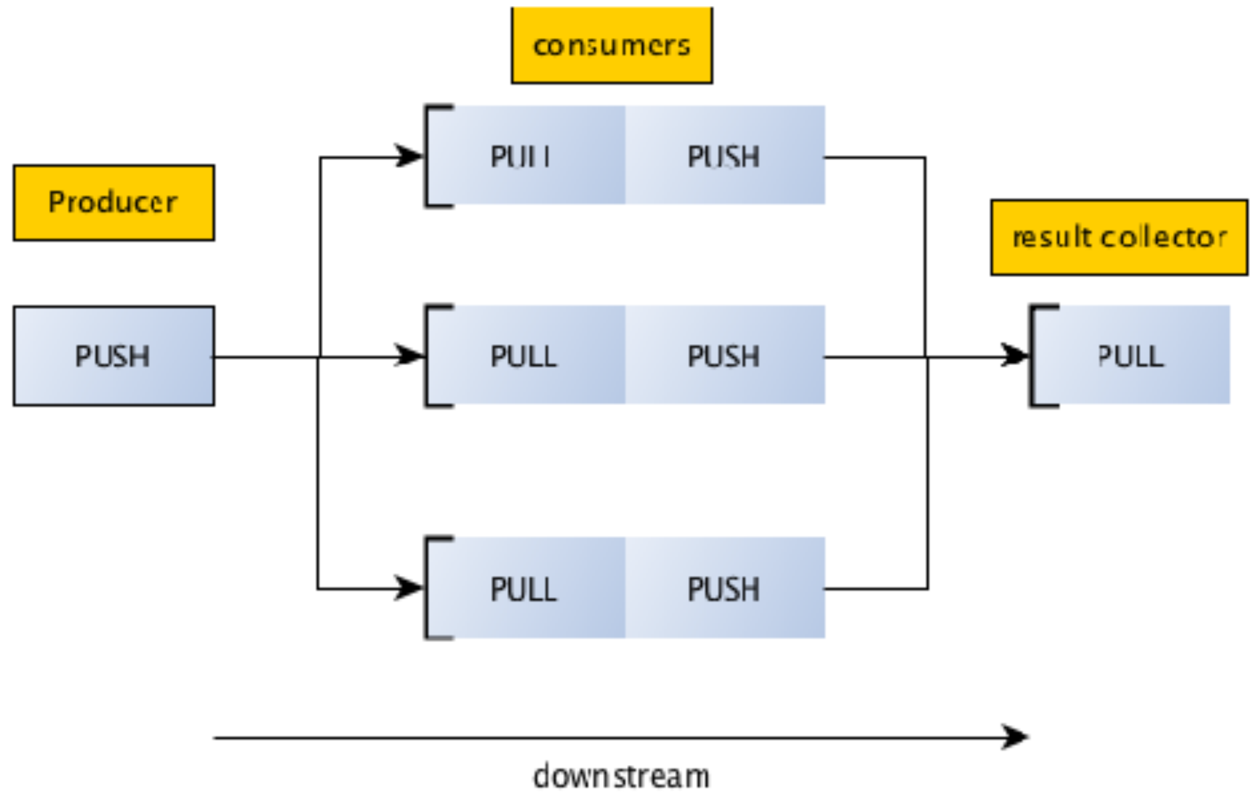


- FSM: currently a waterflow model.
- As soon as the previous step is over, the next step is triggered.
- In each step, a signal is emitted from master to slaves, and slaves work for the specified task of the step.
- Each slave responds a finishing message once the task of the state is done.
- This is not a pure pipelining — for instance histogramming slave does not need to wait for master's signal of **`HISTO`** starting — as soon as the data is pushed from upstream the histogramming slave should start the task and send the result to downstream.
- **Need rework here?**

15:40 **Pipelined version works!**

SlaveProcess

- The base class of the pipeline component.
- Support `{Scan, DataProcess, Histogramming, Analysis}` types so far.
- Not a static templated class → ideally wish to generalize?
  (e.g. we may want to combine Scan+DataProcess in the same process).
- Pipeline is implemented by `ZMQ TCP PUSH/PULL` model (by default).
  - According to the `ZMQ PUSH/PULL` model, one could potentially think of having multiple histogrammers per FE — need to insert histogram merger afterwards, but can distribute Event data to multiple processes via the network?
- Each slave process communicates with the `master` process.
  This is implemented by `ZMQ TCP DEALER` model (by default).



Issues

- For the moment, histogramming and analysis slaves handle all FEs in the scan. We should distribute processes and otherwise there's no scale merits of distributing.
- Serialization can be a bottleneck. Minimizing the serialization goes to the direction of monolithic and passing data by pointers. We should be able to have a more flexible embedding of the functions in the distributed scheme.
  - One possibility is to expand the `Connectivity` class so that it can be also used for direct pointer transfer in a monolithic process — so far it only supposes serialized communication.
  - Meanwhile we need to devise a very efficient serialization to reduce the bottleneck (e.g. histogram data format)
- Master would not need to die for each single scan… think of primlist of scans, then one could pipeline scans while launching many different histogramming/analysis instances for scans. The actual data-taking can finish earlier while waiting for back-end processes to finish. This does not apply for scans with feedback.

Step-by-step ToDo's from here…

- Rebasing to the latest `devel` branch (tedious!)
- Run with `Rd53aEmu`
- Distribute processes over multiple PCs
- Run multiple `Rd53aEmu` "chips" and split histogramming/analysis processes for each FE.
- Run multiple `Rd53aEmu` scan engines simultaneously.
- Add LocalDB slave and hook it up at the `mongoDB` server.