NestDAQ framework development

T.N. Takahashi
Research Center for Nuclear Physics, Osaka University

Outline

- Motivation
- Implementation
- Future prospect
- Summary

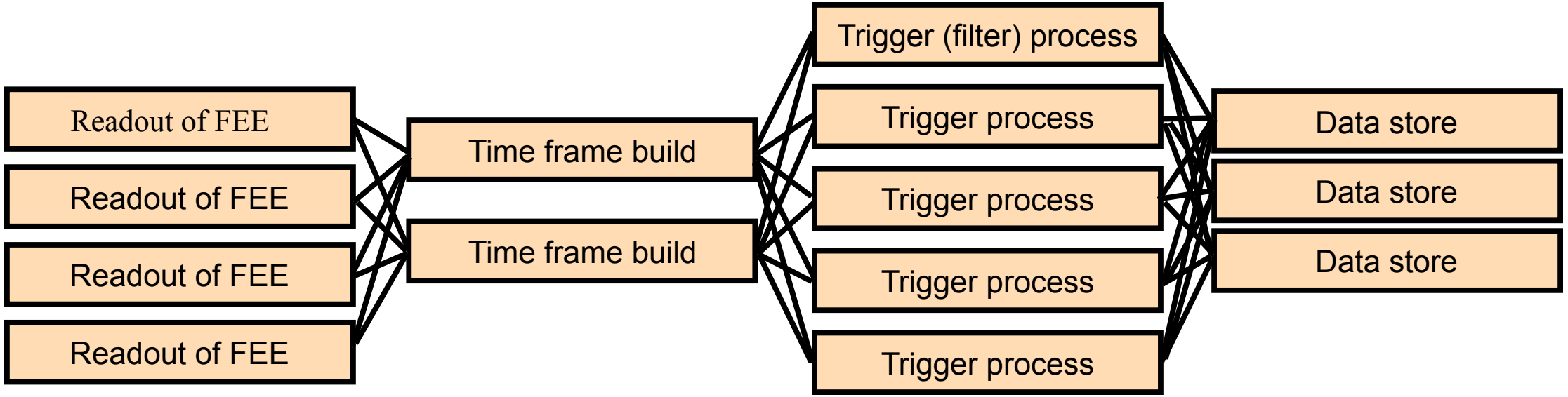
SPADI Alliance

Motivation

- Problems caused by data increase
  - Increase of the numbr of trigger channels, complex trigger logic
  - Lack of budget, time, and manpower required for development
    ➡️ **Difficult to develop hardware triggers**
- The above problems can be solved with streaming readout DAQ.
  - **Data reduction through distributed processing**
  - Relaxed latency requirements → various processors: CPU, GPU, FPGA, ...
  - People who are not experts in circuits or DAQ can be involved in the development.

Streaming type DAQ software : Concept

- Management of various types and numerous processes (tasks)
- Overall control
  - Peer-to-peer, one-to-many, many-to-one, many-to-many
  - Data buffring, load balancing



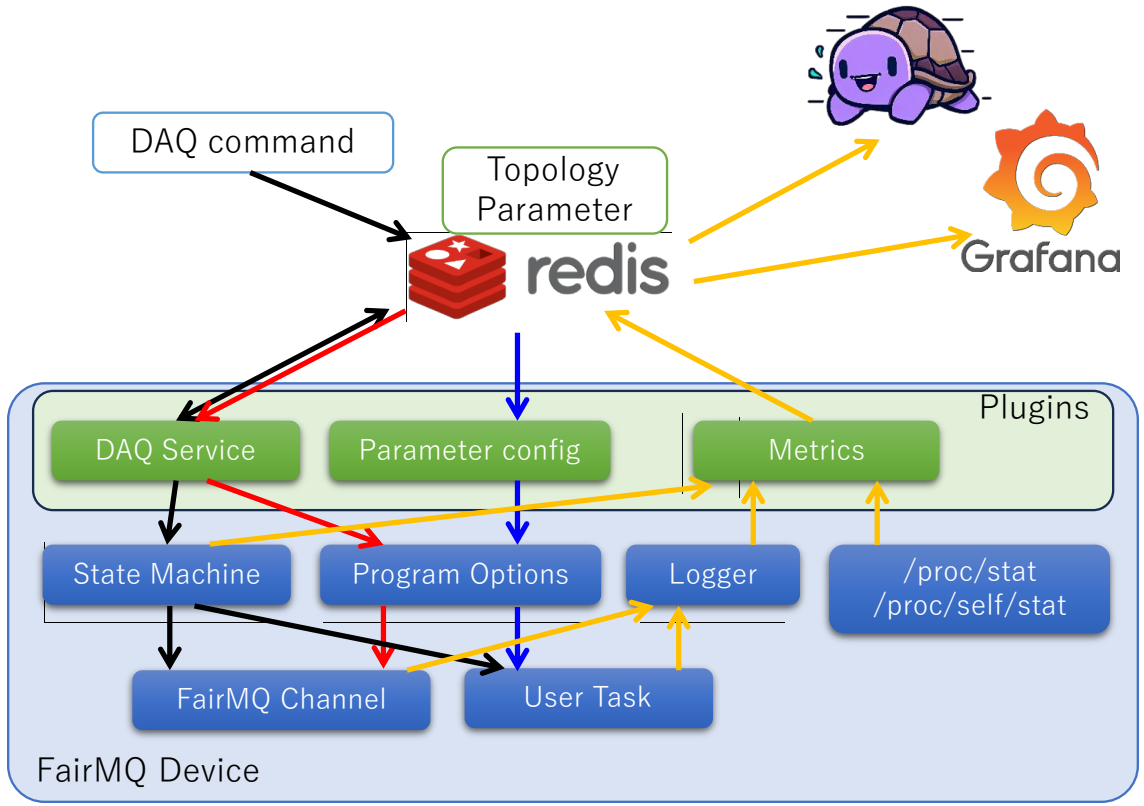
NestDAQ software stack

NestDAQ (**Ne**towrk based **st**reaming **DAQ**)

- **FairMQ**
  - Brolerless communication (ZeroMQ)
  - State machine control for task execution
- **redis**
  - Key-value store
    - In-memory type → fast response
  - Management of a large number of processes
    - Configuration parameters
    - Message queue, Key-space notification → control UI

NestDAQ process structure

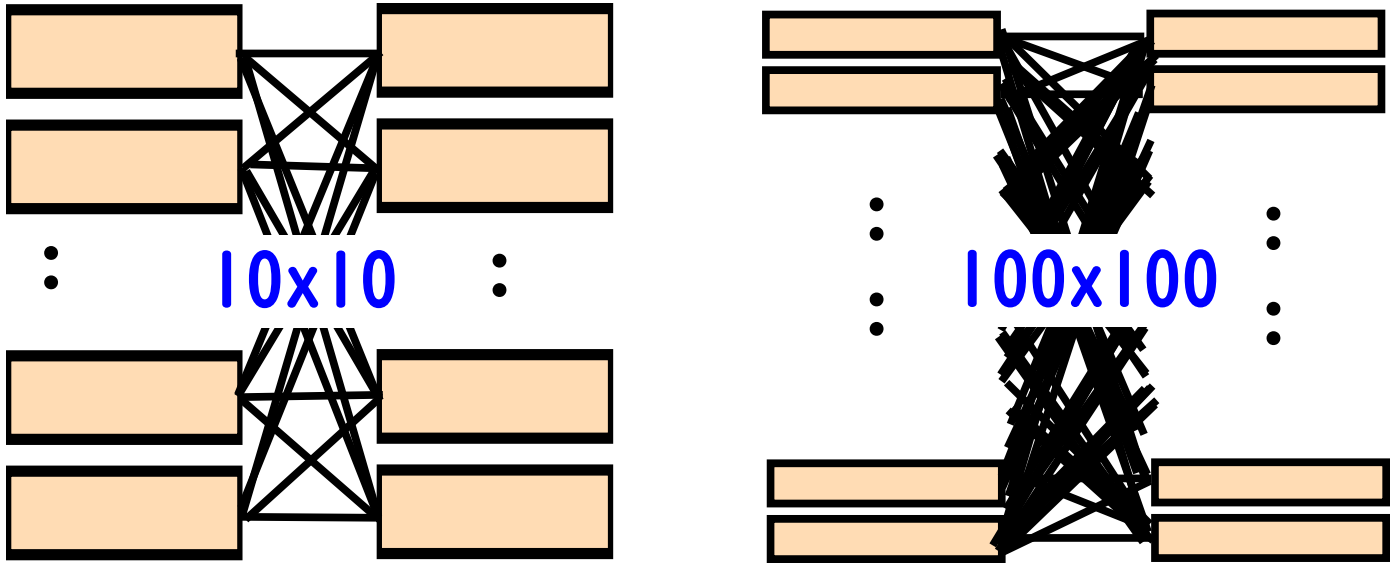
- **DAQ Service Plugin**
  - **Service discovery**
    - semi-automatic connection configuration
  - Run control
- **Parameter Config Plugin**
  - Read program options from the command line or the data base
- **Metrics Plugin**
  - Monitroing processes
  - Export to dashboard tools (Grafana, SlowDash)
- **User task**
  - Override virtual functions in a derived class of FairMQ Device



NestDAQ itself is a data-agnostic framework that allows the implementation of the transmission and processing of any type of data.

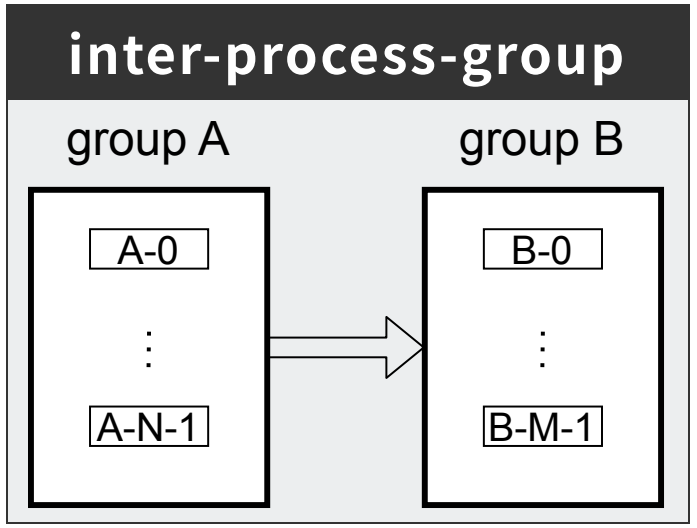
Handling connections between many processes

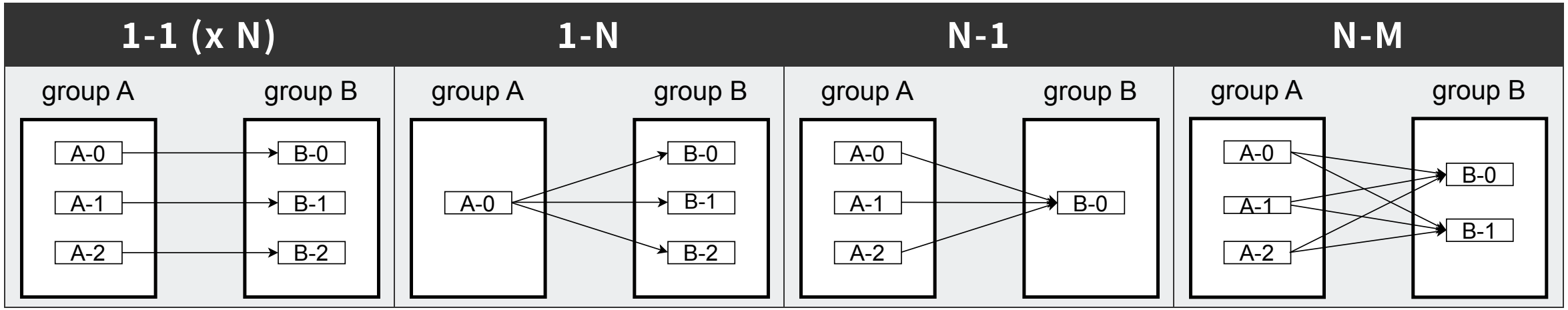
- It is a pain to have to change the configuration file of the connection settings when the number of processes changes.
- The number of peer-to-peer connections increases exponentially with the number of processes.
  - 10x10, 100x100, 1000x1000, ...
- Can we manage the connection settings of a large DAQ system with **fewer lines**?



2-layers of connection settings

- **inter-process-group**
  - Specified by the user
- **inter-process**
  - 4 types: 1-1 (x N), 1-N, N-1, N-M
  - **Automatically generated** from **the number of processes** registered in the database and "**hints**" given by the user

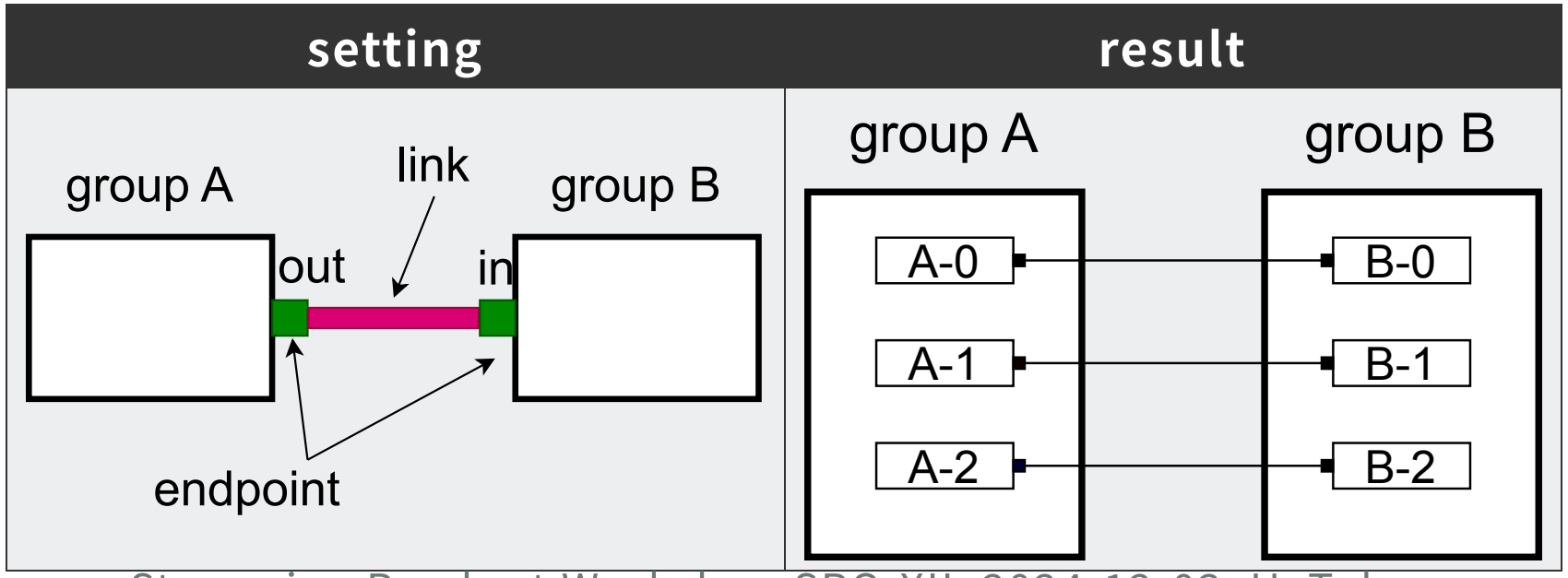




Example: N one-to-one connections

- The number of processes of A and B in the registry are the same ➡️ N one-to-one connections in parallel
- One socket (sub-channel) per channel

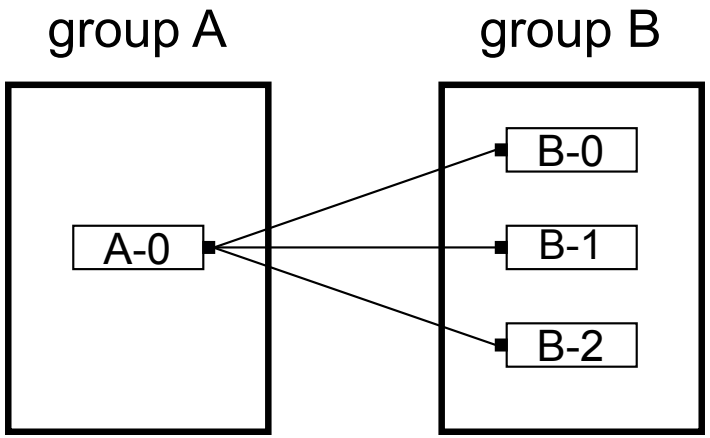
```
endpoint A out  type push method bind
endpoint B in   type pull method connect
link     A out B in
```



Example: 1-N connections (or N-1)

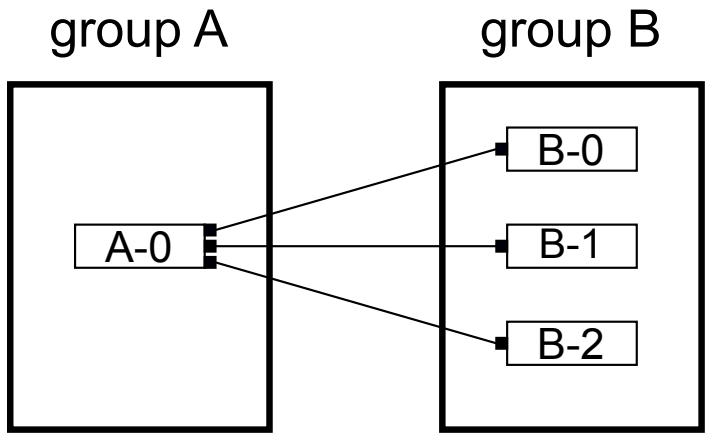
- The number of processes of A and B in the registry = 1-N (N-1) ➡ Uses ZeroMQ's built-in 1-N (N-1) connection
- With "hint"(`autoSubChannel true`) ➡ **Creates a socket (sub-channel) per connection**

```
endpoint A out  type push method bind
endpoint B in   type pull method connect
link     A out B in
```



ZeroMQ's built-in round robin selects the destination

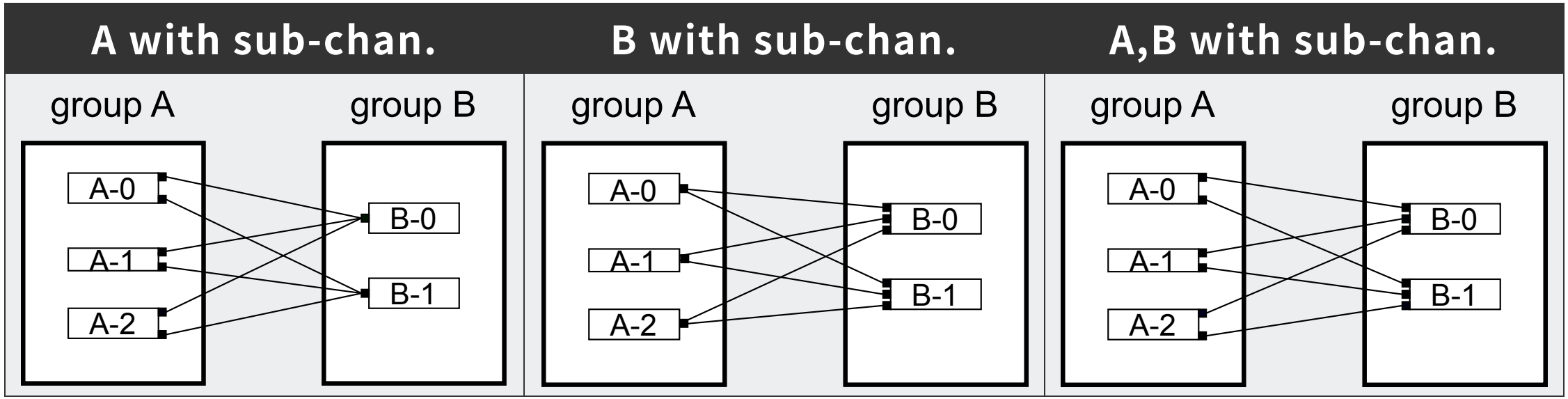
```
endpoint A out  type push method connect autoSubChannel true
endpoint B in   type pull method bind
link     A out B in
```



One socket per connection
User appl. selcects destination.

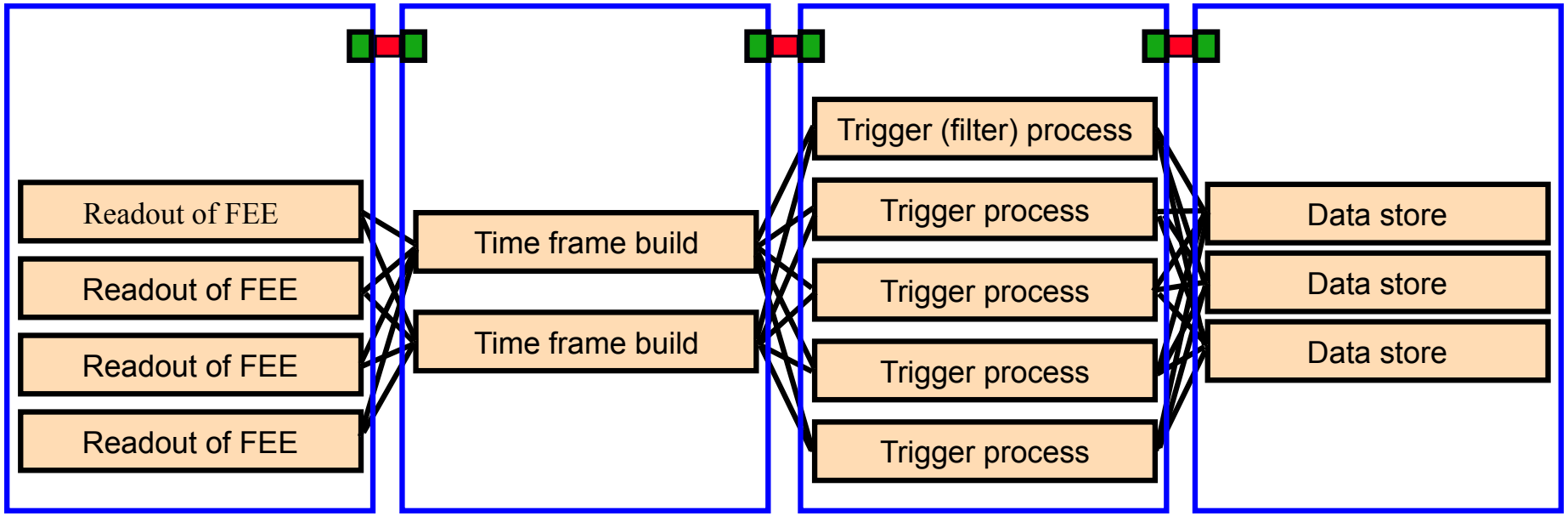
Example: N-M connections

- Multiple processes of both A and B ➡ N-M
- The process with the "hint" creates a socket (sub-channel) for each connection.

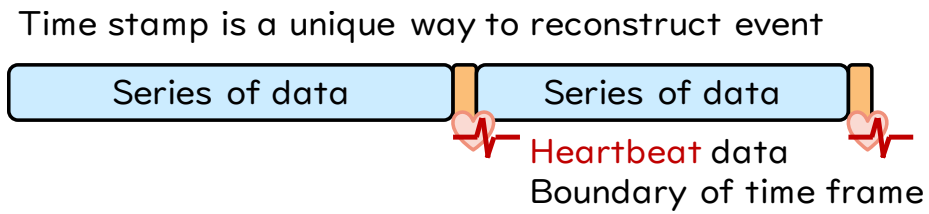


Example: 4-stage pipeline (4 groups)

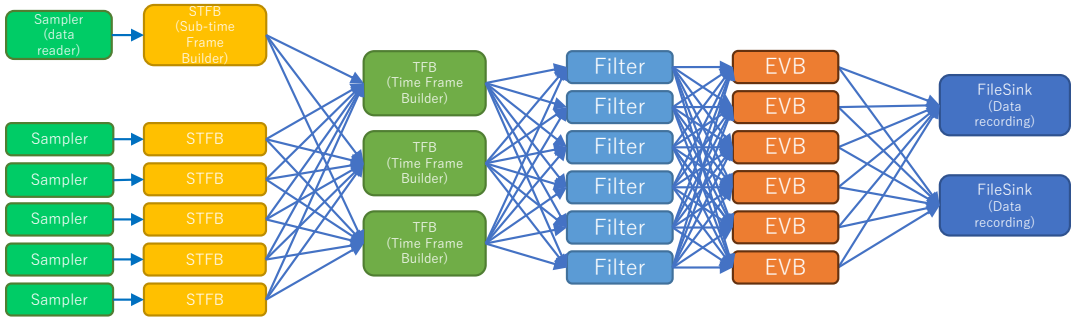
- 6 **endpoints** and 3 **links**
- 6+3 = 9 lines in the configuration → any number of processs in each group



Typical DAQ configuration



Common DAQ configuration



3 stage N column parallel configuration

- Front-end electronics (FEE)
  - outputs self-triggered data with heartbeat frames (HBFs) that are the time interval separators and common to all FEEs.
- Sampler
  - reads (and buffers) data from the FEE.
- Sub-Time Frame Builder
  - slices data from the Sampler at the HBFs and makes a fragment of one time slice (sub-time frame, STF).
- Time Frame Builder
  - merges STFs in the same time slice.
- Filter/Online Trigger
  - finds the good event in the time frames.
- Event builder (for Streaming Read Out)
  - extracts the data in the time near the good events.
- File Sink
  - records data to a file

Future prospects

In progress

- Increase **robustness**
  - Discarding data that cannot be processed
- Handling telemetry data
  - Metrics, **logs**

Under discussion

- **Remote management (deployment) of DAQ processes**
  - NestDAQ processes are launched as native executables.
    - Command line / shell scripts + SSH
  - Introducing container orchestration

Summary

- Streaming DAQ software is highly demanded for online data filtering as an alternative to hardware triggers.
- **NestDAQ**
  - **FairMQ** is used as the basis for data transport and task execution.
  - We have developed **redis**-based plugins of FairMQ
    - **Service discovery, run control, parameter conficuration, process monitoring**
    - **Semi-automated connection configuration** makes it easy to set up multiple inter-process connections.
- Future prospects
  - More robust
  - Log collection
  - Container orchestration is under discussion.