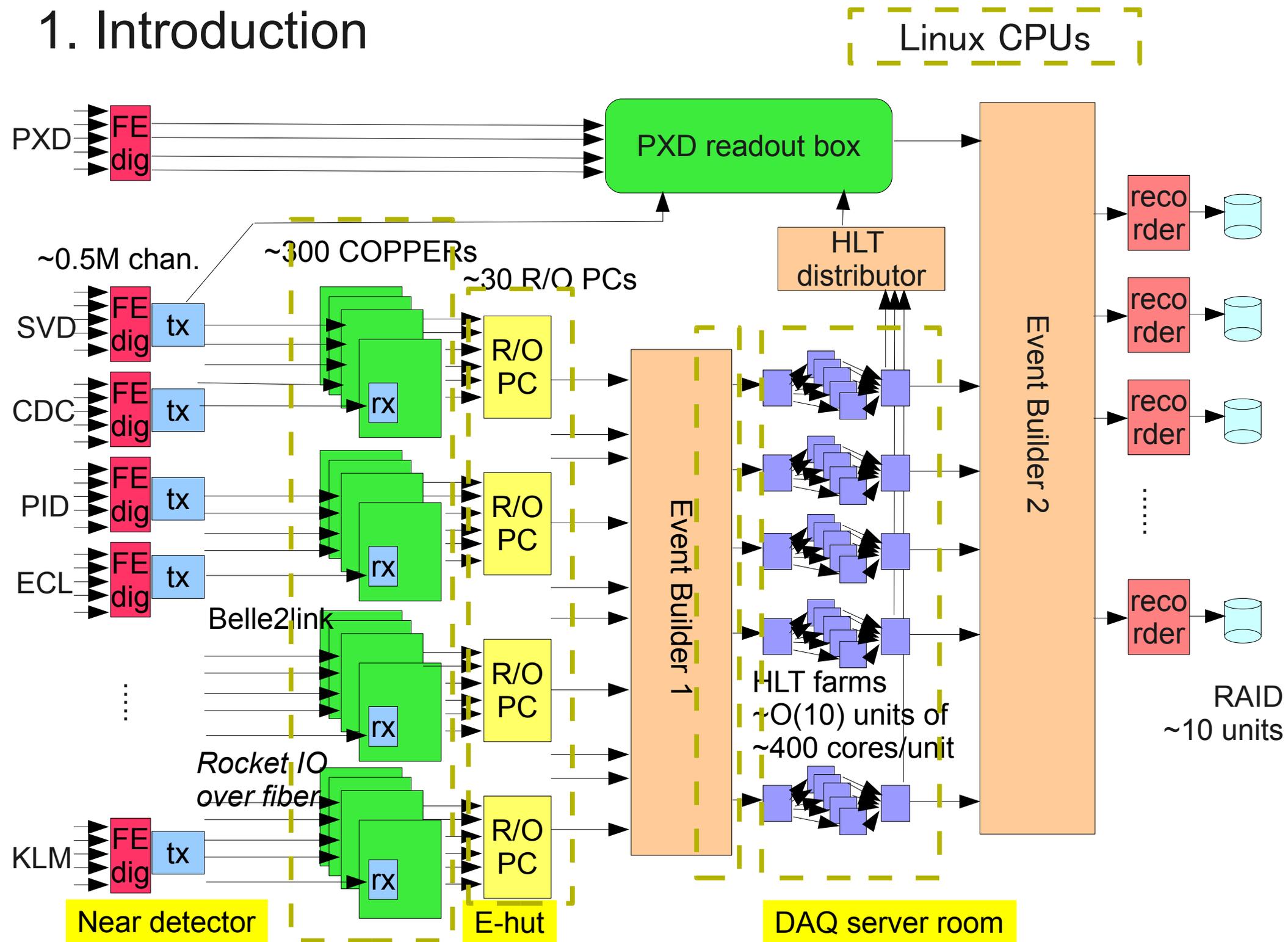


# Data Flow in Belle II DAQ

R.Itoh, KEK

# 1. Introduction



## Requirements

- 30 kHz L1 rate for  $L=8 \times 10^{35} \text{cm}^{-2} \text{sec}^{-1}$  (Max. ave. rate),
- Event size :  $\sim 1 \text{MB}$  from PXD,  $\sim 100 \text{kB}$  from other detectors.
  - > Real-time reduction of this data stream is a challenging task of Belle II DAQ data flow.

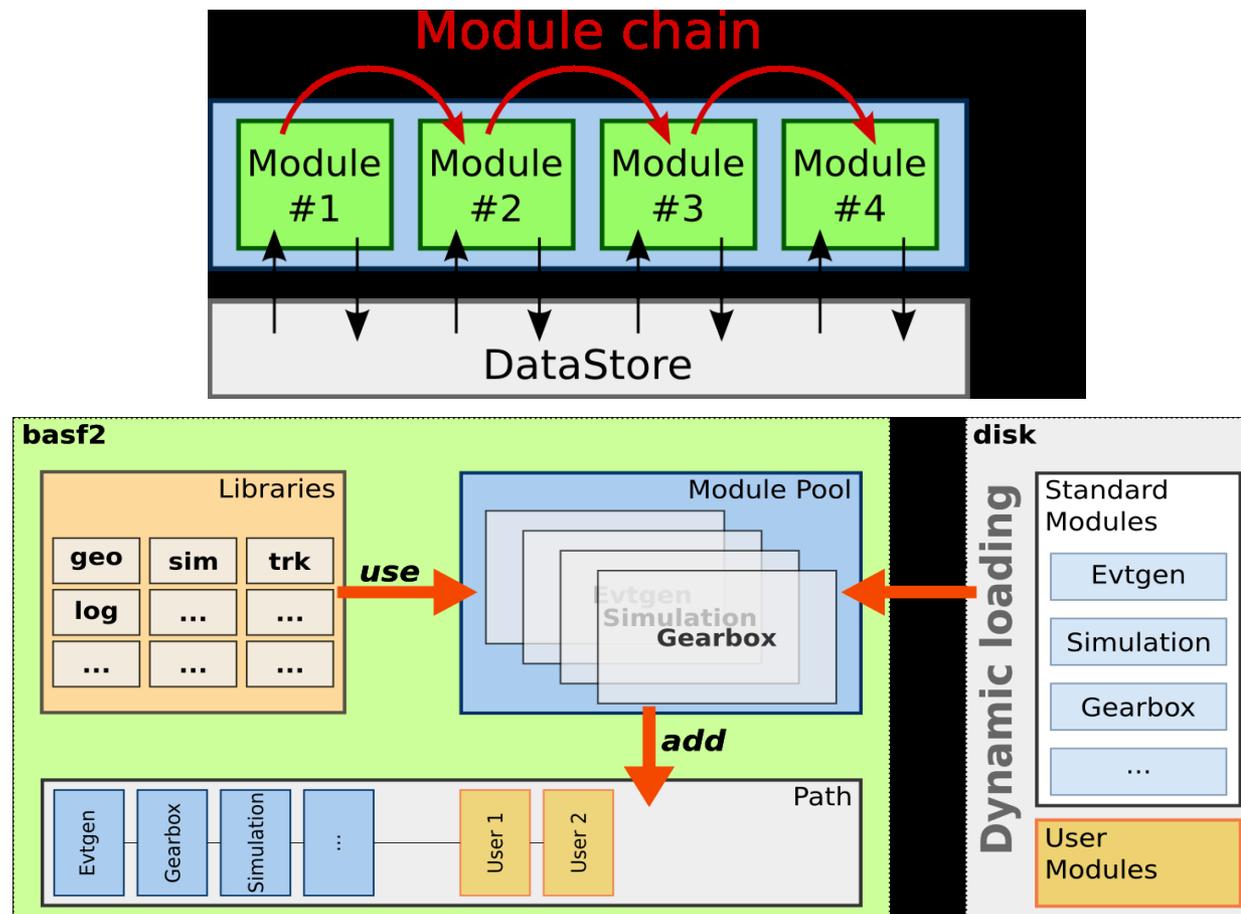
## Design concept

1. No “software” boundary between DAQ and offline
  - HLT uses the same reconstruction software used in offline and the “physics event selection” is used as software trigger.
  - The software for data reduction and monitoring are supposed to be developed in offline environment and is ported in DAQ w/o any modifications.
2. Multi-level data reduction
  - Having CPUs at every step of DAQ to perform the data reduction step by step.

- \* A common data processing framework shared with offline
- \* Unified data flow based on the framework
- \* ROOT based data management

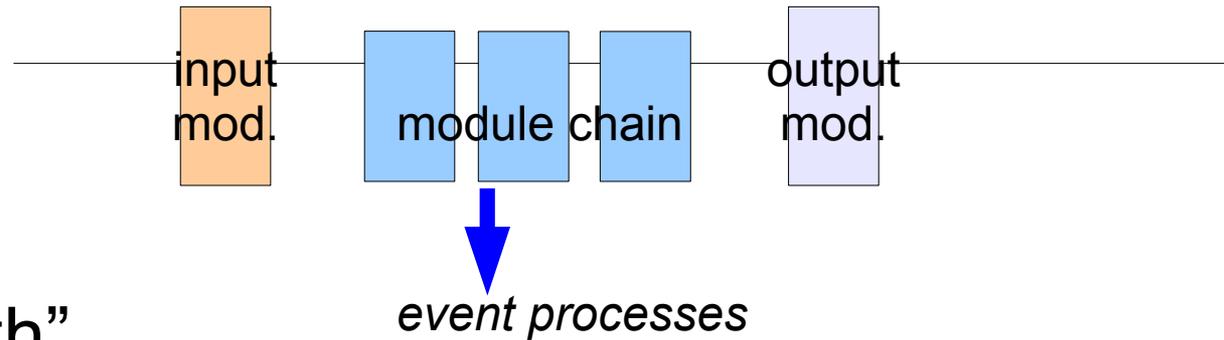
## 2. Common data processing framework : basf2

- “basf2” is a Belle II standard analysis framework designed considering the use in DAQ.
- It has the “module and path” structure which is derived from Belle's BASF framework.
- I/O is implemented as a module. ROOT I/O, Socket I/O, etc can all be implemented as the I/O module.

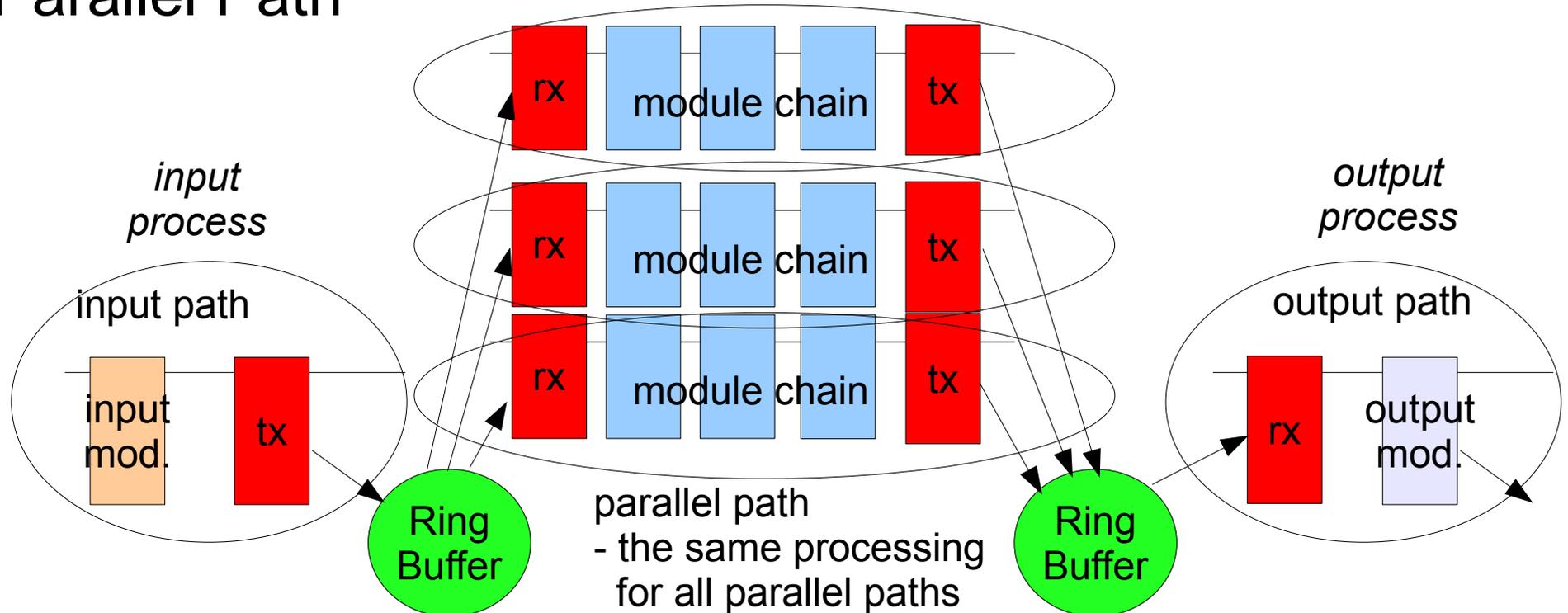


# Parallel processing implementation

- Event-by-event parallel processing mechanism is implemented in basf2 inheriting BASF's implementation
- “Partial parallel processing” is newly introduced to manager I/O modules. ➔ For the use in HLT

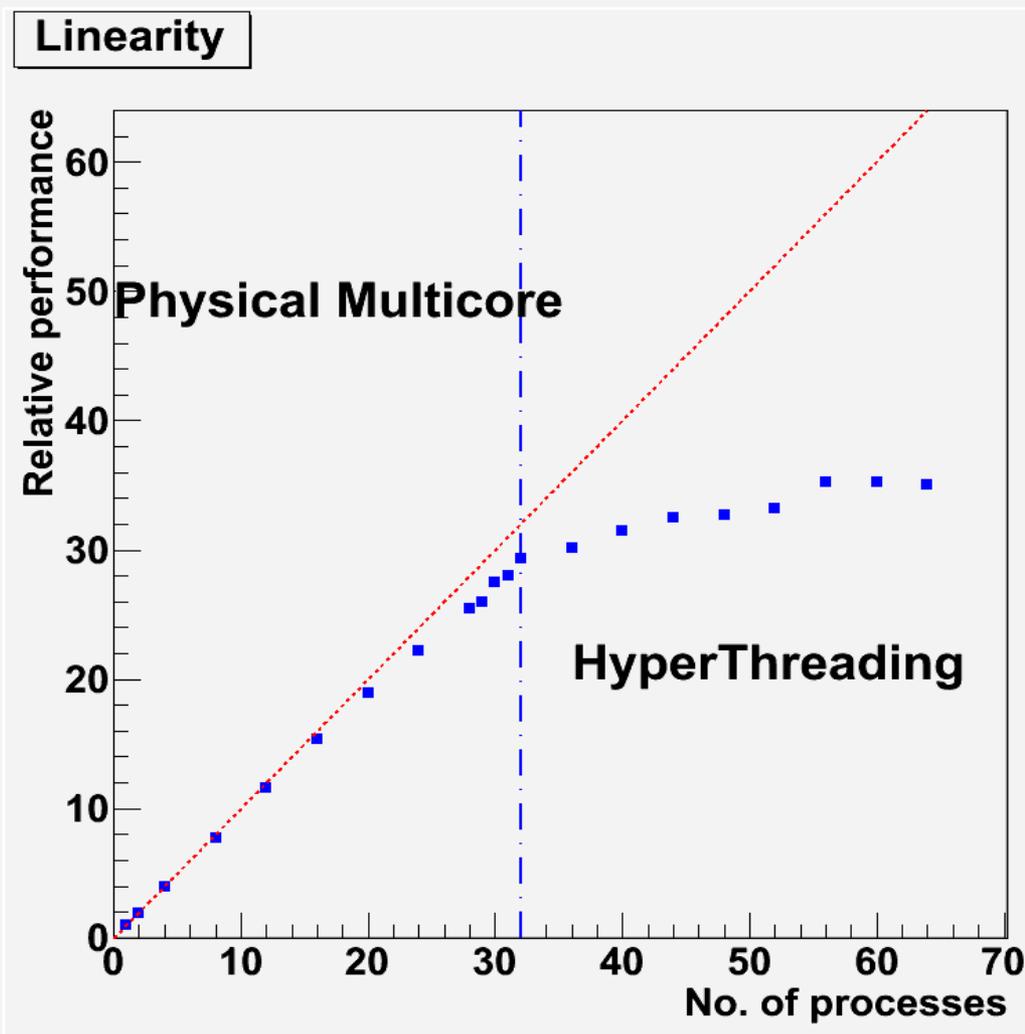


## “Parallel Path”



## Performance test of parallel processing

- \* Script in previous page is used (gen+sim+rec).
- \* No. of processes is scanned and the elapsed time for 1000 events is measured.
- \* Platform : 2GHz x 32 cores (4 Xeon X7550), 65GB memory



elapsed time:

1 core = 15557 sec.

16 core = 1011 sec.

32 core = 555 sec.

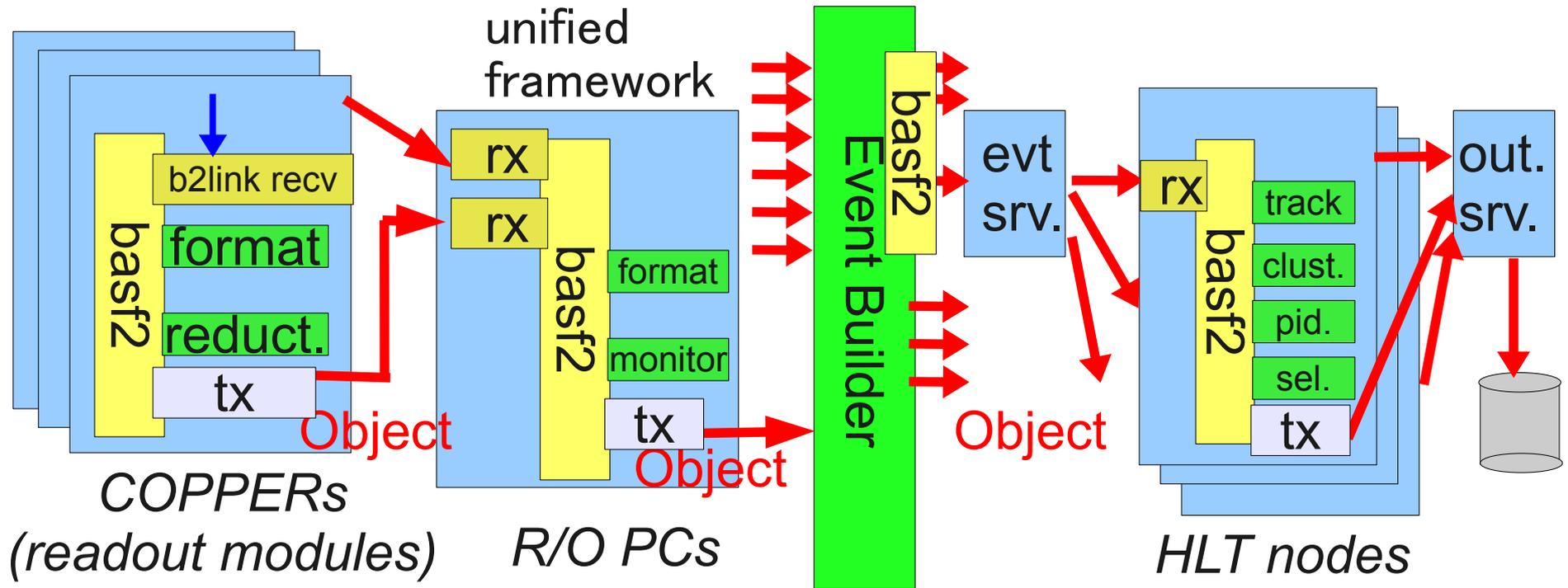
56 core = 440 sec.

(best)

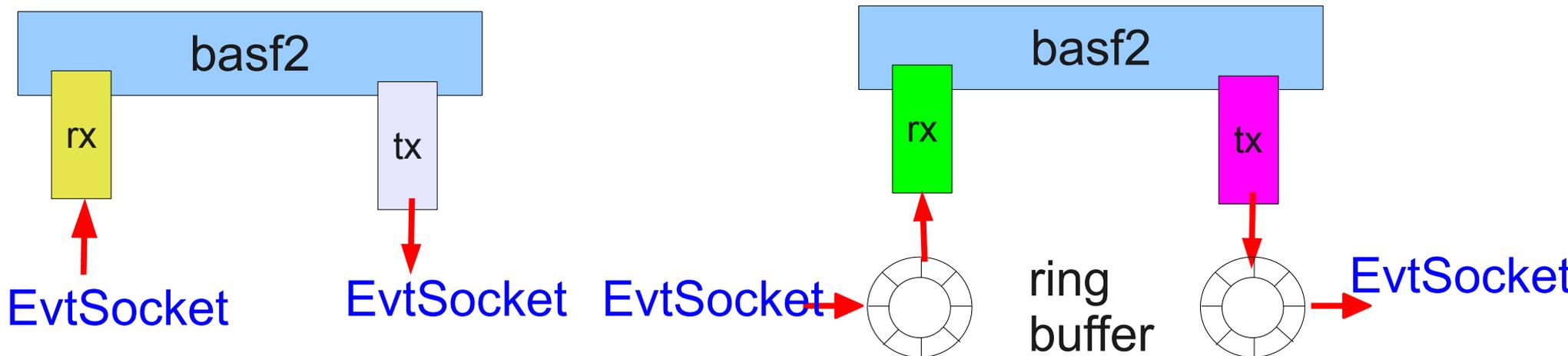
\* Good linearity up to max. number of cores.

\* Improvement by HyperThreading :  
~ 1.26

### 3. Data flow : implementation



- Raw data are placed in ROOT "object" by COPPER CPU.
- Objects are streamed and transferred between nodes using "EvtSocket" class



# EvtMessage

- Streamed object format used in DAQ data transfer

Header (16words) (tentative, something similar to d2packet\_header)

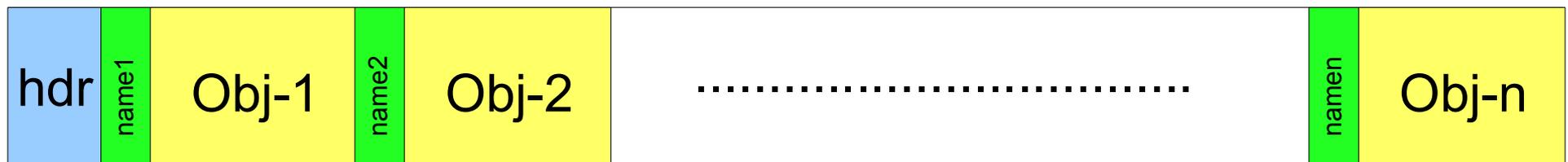
word 0 : Number of bytes in this record  
word 1 : RECORD\_TYPE (begin\_run, event, end\_run, others)  
word 2,3 : Time stamp (gettimeofday() format)  
word 4 : source of this message  
word 5 : dest. of this message  
word 6-15 : Reserved (used to store durability, nobjs and narrays for DataStore objects)

List of serialized objects (word 16-)

Streamed object :

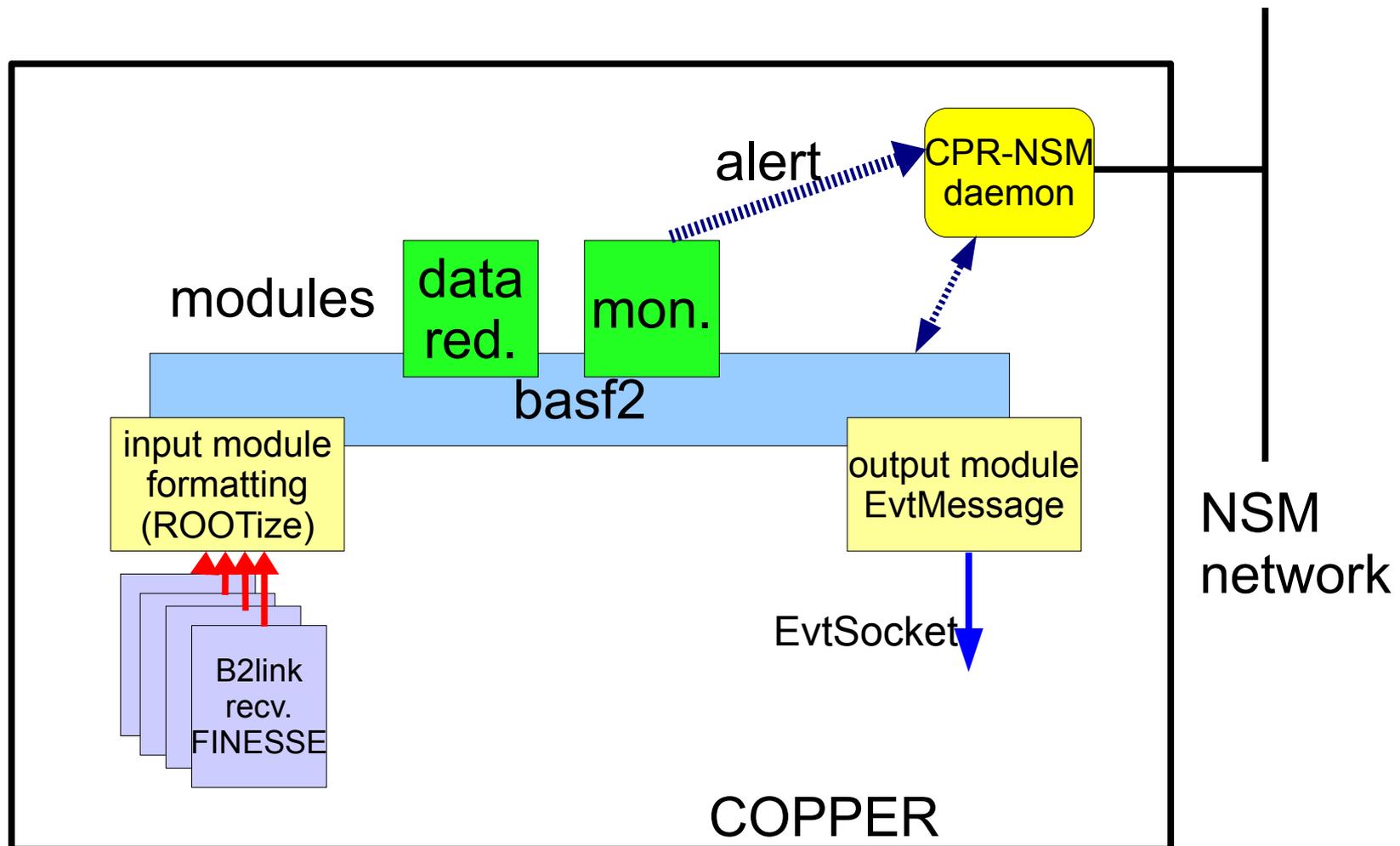
word 1 : nbytes of object name  
bytes : object name

word n : nbytes of streamed object  
bytes : streamed object



# COPPER

- Format raw data read from Belle2link to ROOT object.
- Perform data reduction and monitoring if necessary.
- Transfer raw data object to Readout PC through EvtSocket.



## Streaming raw data

- \* Raw event data have to be stored in “DataStore” as an object so as to be managed by basf2.
- \* To transfer the raw data to different node, the DataStore has to be streamed (serialized) and destreamed(deserialized).
- \* Streaming using ROOT is reported to be CPU consuming.  
-> could be an issue for COPPER CPU (ATOM@1.6GHz).



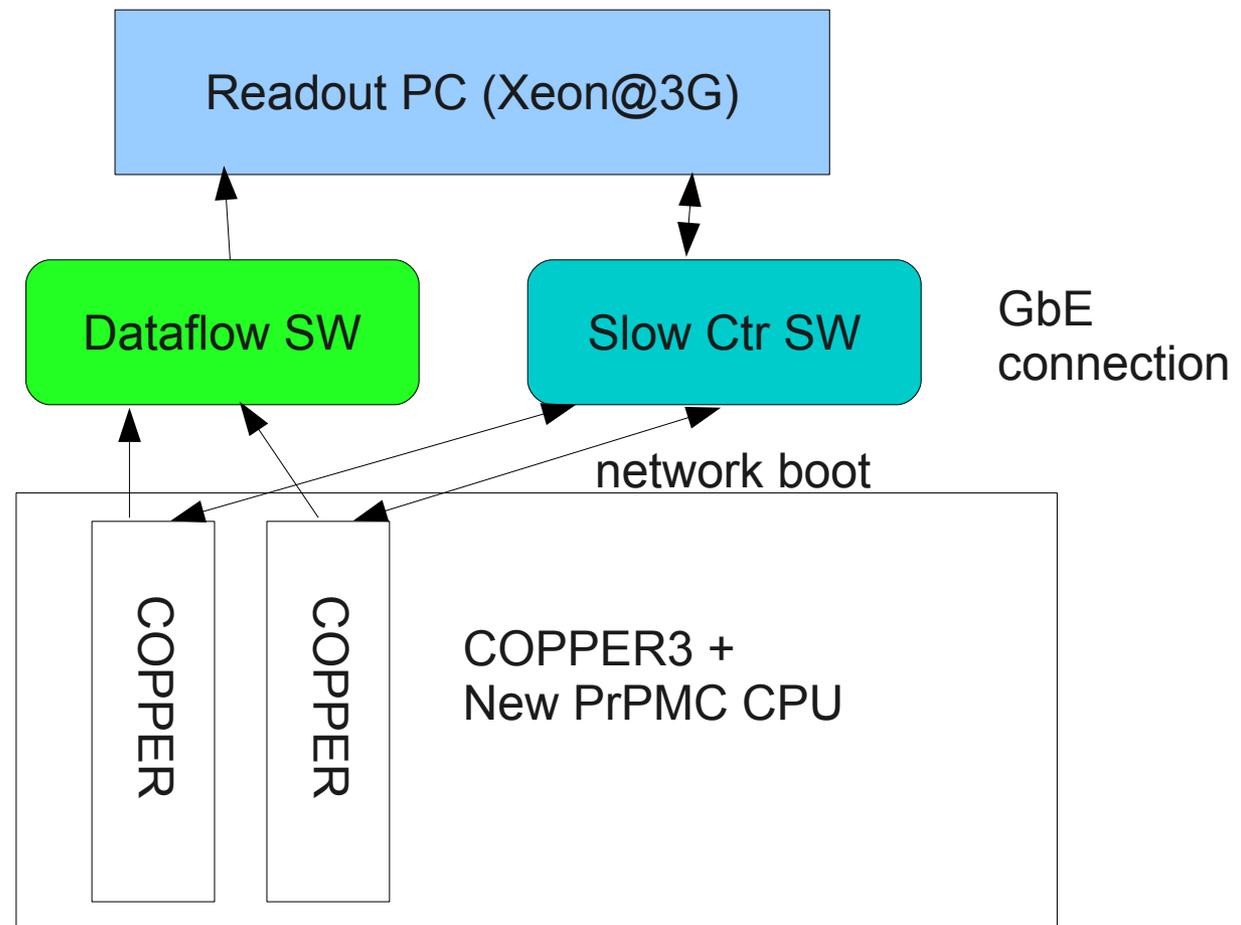
- \* Use the simplest structure of raw data :  
“variable sized array of integer”
  - Overhead of ROOT streaming can be minimized.
  - If still slow, optimized streamer (=handwritten streamer) will be implemented.
  - Data access through accessing class (just like Belle's TdcUnpacker class)

# Performance test of streaming

Environment to run basf2 on COPPER

Test bench at Tsukuba B3 floor

- Scientific Linux 5.7
- Belle2 library (svn head on Dec.8 + development ver of externals)
- Network boot (set up by using system\_config\_netboot)



# Class to manage raw data : RawCOPPER

```
class RawCOPPER : public TObject {
public:
    //! Default constructor
    RawCOPPER();
    //! Constructor using existing pointer to raw data
    buffer
    RawCOPPER( RawHeader& hdr, int nwords, int* );
    //! Destructor
    virtual ~RawCOPPER();
    //! copy rawdata into internal buffer
    virtual void copy(RawHeader& hdr, int nwords,
int*);
    //! allocate buffer
    virtual int* allocate_buffer(int nwords);
    //! get buffer
    virtual int* buffer();
    //! set buffer
    virtual void buffer(int, int*);
    //! get header
    virtual RawHeader& header ( void );
    virtual void header ( RawHeader& hdr );

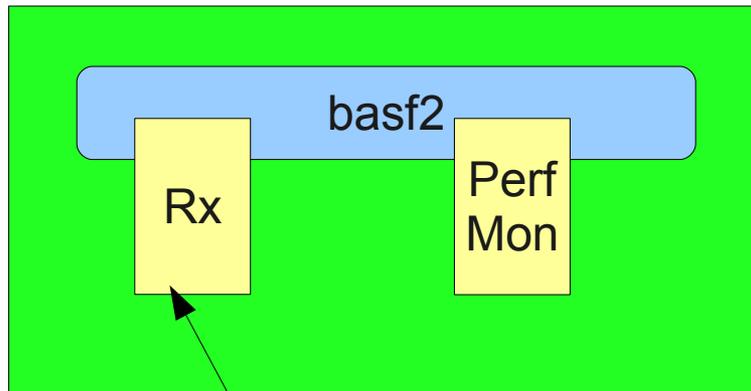
private:
    int m_nwords;
    int* m_buffer; //[m_nwords]
    RawHeader m_header; //|| do not split the header
    bool m_allocated;

    ClassDef(RawCOPPER, 1);
};
```

Raw data from Belle2link are treated as variable length integer array. The dummy data are generated and filled on COPPER.

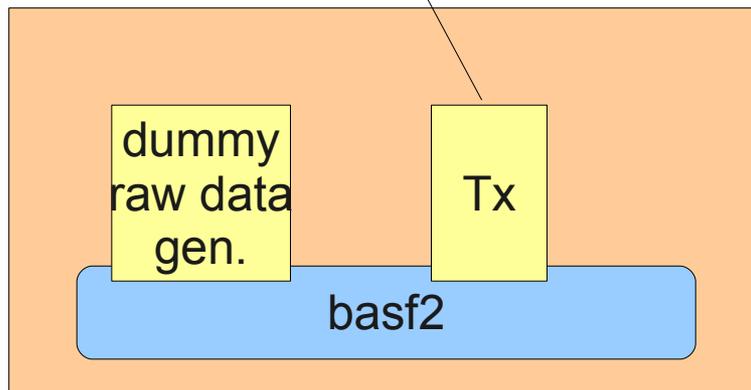
8 words fixed header

## Readout PC



EvtSocket

## COPPER



## [COPPER]

Dummy raw data generator:

- Generate raw data and put them in RawCOPPER object.
  - <- Pregenerated raw data are memcpied.
- Put RawCOPPER in *DataStore*

## Tx

- Retrieve RawCOPPER from *DataStore*
- Serialize the object to a byte stream and send it out to network

## [Readout PC]

### Rx

- Destream received byte stream and restore it in RawCOPPER object
- Put RawCOPPER in *DataStore*

### Perf

- Performance monitor module

# Performance

Data size : 32 words

1. Raw Data Generator only on COPPER : 30.2 kHz
2. Full data flow. Rate measured on R/O PC : **5.1 kHz**
  - > **Veeerrrrry Sllllllowwwww even though full CPU consumption**
  - <- Previous measurement by Higuchi-san using simple algorithm on EPC6315
  - > 30kHz transfer rate with ~160 bytes event size.

Possible reasons of slowness:

- |                               |   |
|-------------------------------|---|
| a) ROOT streaming performance | <- as is expected.....                      |
| b) DataStore access           | <- Object management on basf2 may cost some |
| c) basf2 itself               | <- basf2 is still not optimized             |
| d) Data transfer over socket  | <- Extra CPU cost?                          |

3. Remove socket data transfer : 7.8kHz
4. Replace DataStore access with direct passing : **12.4kHz**
5. + Remove ROOT streaming : **25.5 kHz**

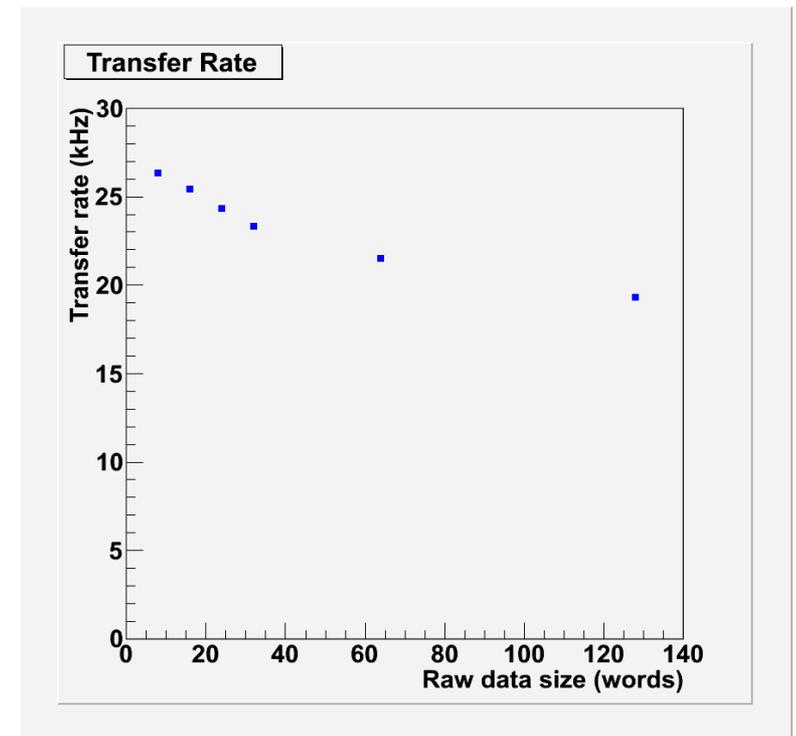
=> Re-design of RawCOPPER class

## New implementation of RawCOPPER class

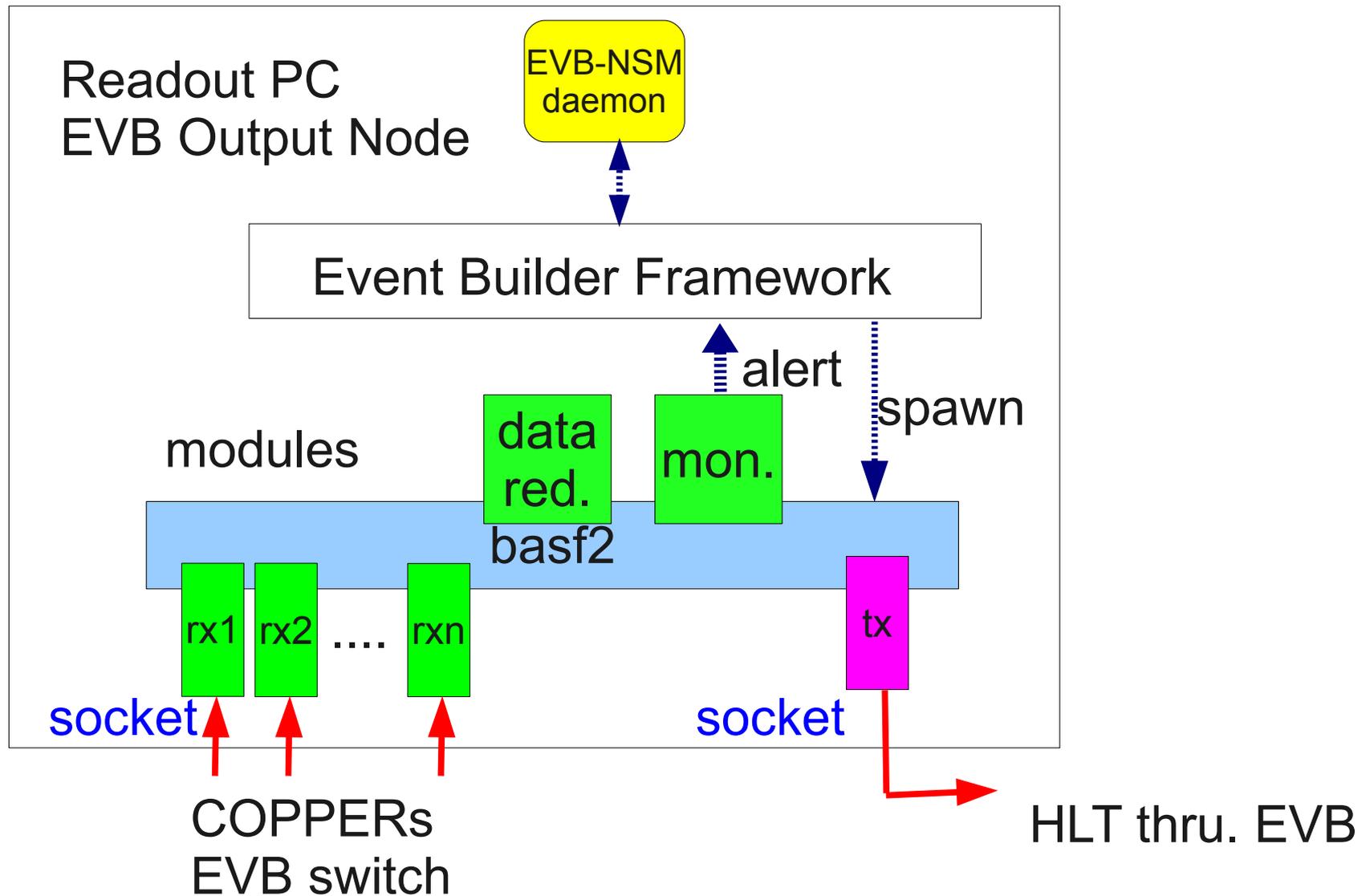
- \* Class to manage single variable length buffer containing fixed word header+ raw data (array of interger)
- \* **No streaming for data transmission. The buffer is sent directly using socket.**
- \* The buffer can be accessed from outside to fill/retrieve the contents.
- \* No "class" access to the contents.

### Performance

- 8 words header + 16words data
  1. Raw data generator only : 37.9 kHz
  2. Full data flow measured at R/O PC : **25.5 kHz**
  3. Data size dependence
    - Could be usable with some improvements/optimization in the framework (basf2).
    - “Hand-written” streamer for RawCOPPER may be tried to restore original design.....



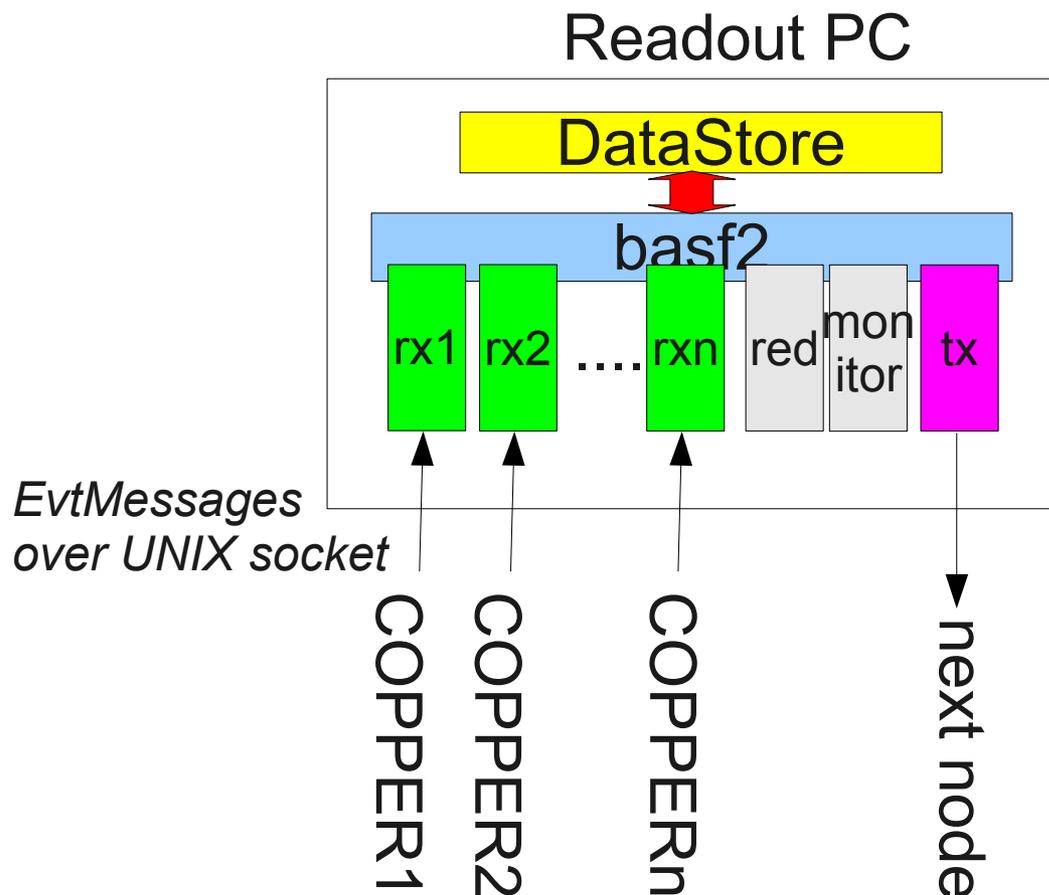
# Event Building on Readout PC



Event sender(tx) and receiver(rx) are coded as “modules” for basf2.

## Event Building with “DataStore” object

- On each node of Readout PC and Event Builder output PC, software event building in has to be implemented in basf2.
  - <-> Transport of event fragment is taken care by “Event Builder” (Yamagata-san)
- Event builder interface (“tx” and “rx”) is provided as **basf2 modules**.



-  Input module of basf2
  - \* Read one event fragment from a socket and save it in DataStore
-  Output module of basf2
  - \* Stream DataStore and send it to a socket.

- By reading all sockets and placing them in DataStore, the event building is automatically ensured.

## Event building using *DataStore*

- DataStore in basf2 is a manager of objects
- Two managers
  - \* Simple objects
  - \* Arrays (TObjArray)
- Event building can be done by treating collection of RawCOPPER objects in a TObjArray:

```
StoreArray<RawCOPPER> RawCDC  
int ncpr = RawCDC->GetLast()+1;  
RawCOPPER* rawdata = new(RawCDC->AddrAt(ncpr))  
int stat = EvtSocket.recv ( rawdata );  
.....
```

- Some worries about the performance
  - \* Need to loop over all “rx” modules for corresponding COPPERs
  - \* Is StoreArray fast enough to fill up to 15 COPPER data?
    - > will be tested soon.
  - \* If performance is not enough, need to consider parallel processing of COPPER data collection.

## 4. Backend processing

- Event builder -> Yamagata-san's talk
- HLT -> next talk
- Post-recording processing = “Prompt Reco” -> next talk
  
- Data reduction strategy : as discussed yesterday

# Expected data rate/size reduction in “early time” operation

