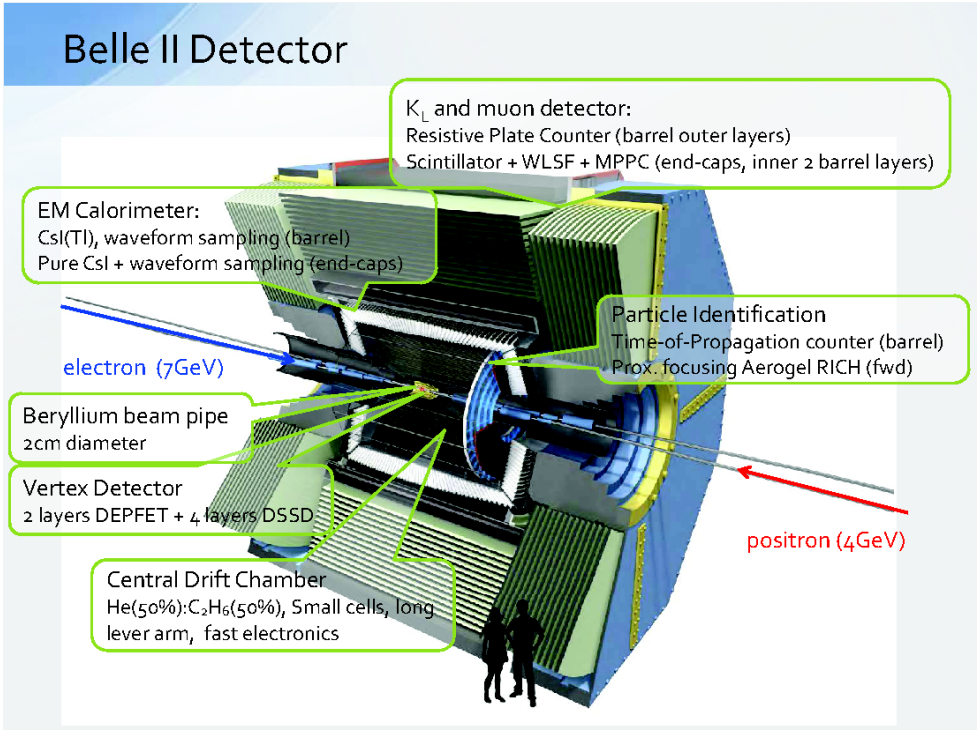
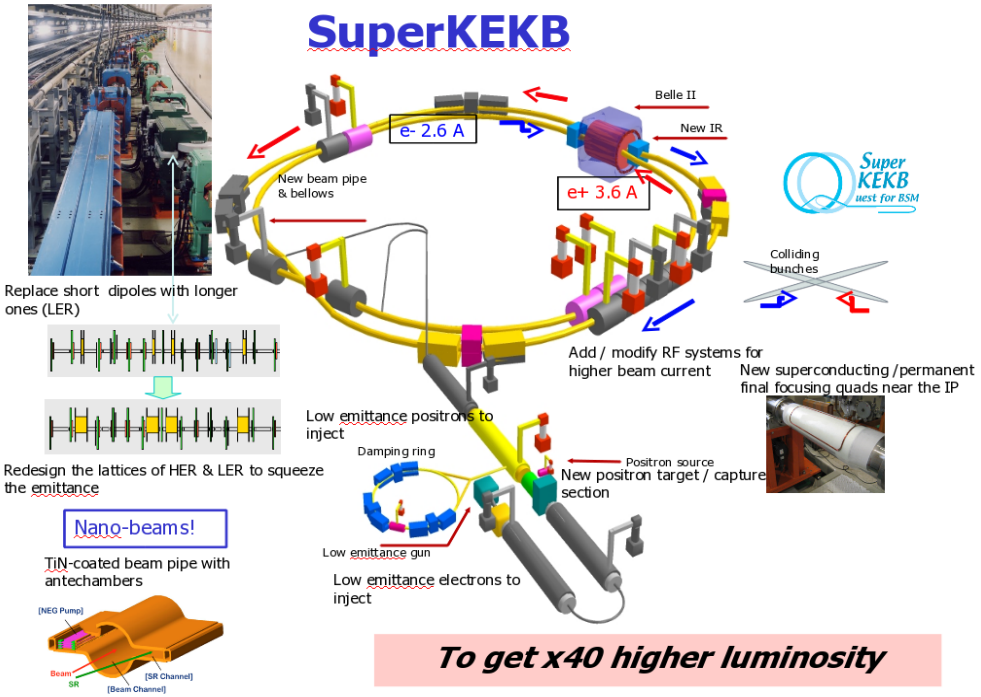
Implementation of parallel processing in Belle II analysis framework (basf2)

Ryosuke Itoh
KEK

SuperKEKB and Belle II





- Belle II experiment @ SuperKEKB is a new generation B-factory experiment aiming at $\times 40$ higher luminosity than that of Belle@KEKB.
- The main purpose of the experiment is the search for New Physics beyond energy scale of LHC in the quantum effect in B meson decays.
- The event size is 100kB(non-PXD) + 1MB(PXD) with the maximum average Level 1 rate of 30kHz resulting in >30GB/sec data flow @L1.
- The data processing is a challenge for computing, which requires a versatile parallel processing of events with multi-core to GRID.

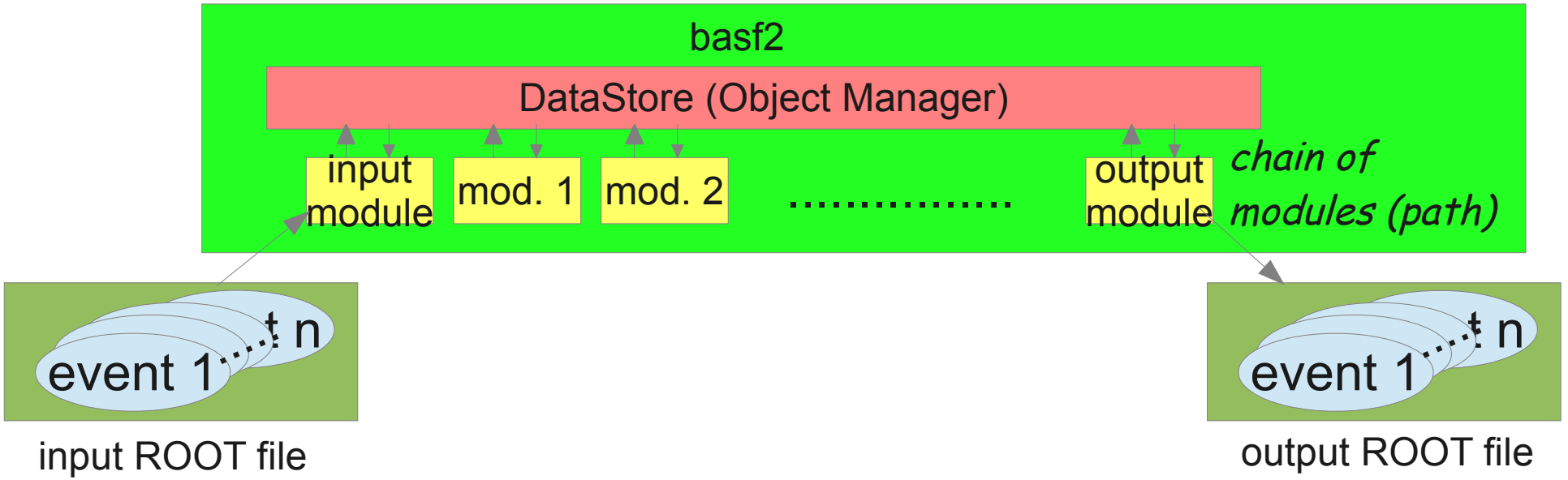
basf2 framework

- ***basf2*** is a software framework developed for the use in Belle II.

- ***basf2*** has a software bus architecture so that a large scale application is implemented by plugging a set of small-functioned modules into the framework (path).

- The data are managed by ROOT and passed among modules by the DataStore object manager.

- The framework is designed to match with a wide range of the usage in MC/DST production, user analysis, and in the DAQ software.



Parallel Processing

- Recent PC servers are equipped with multi-core CPUs and it is desired to utilize the full processing power of them.

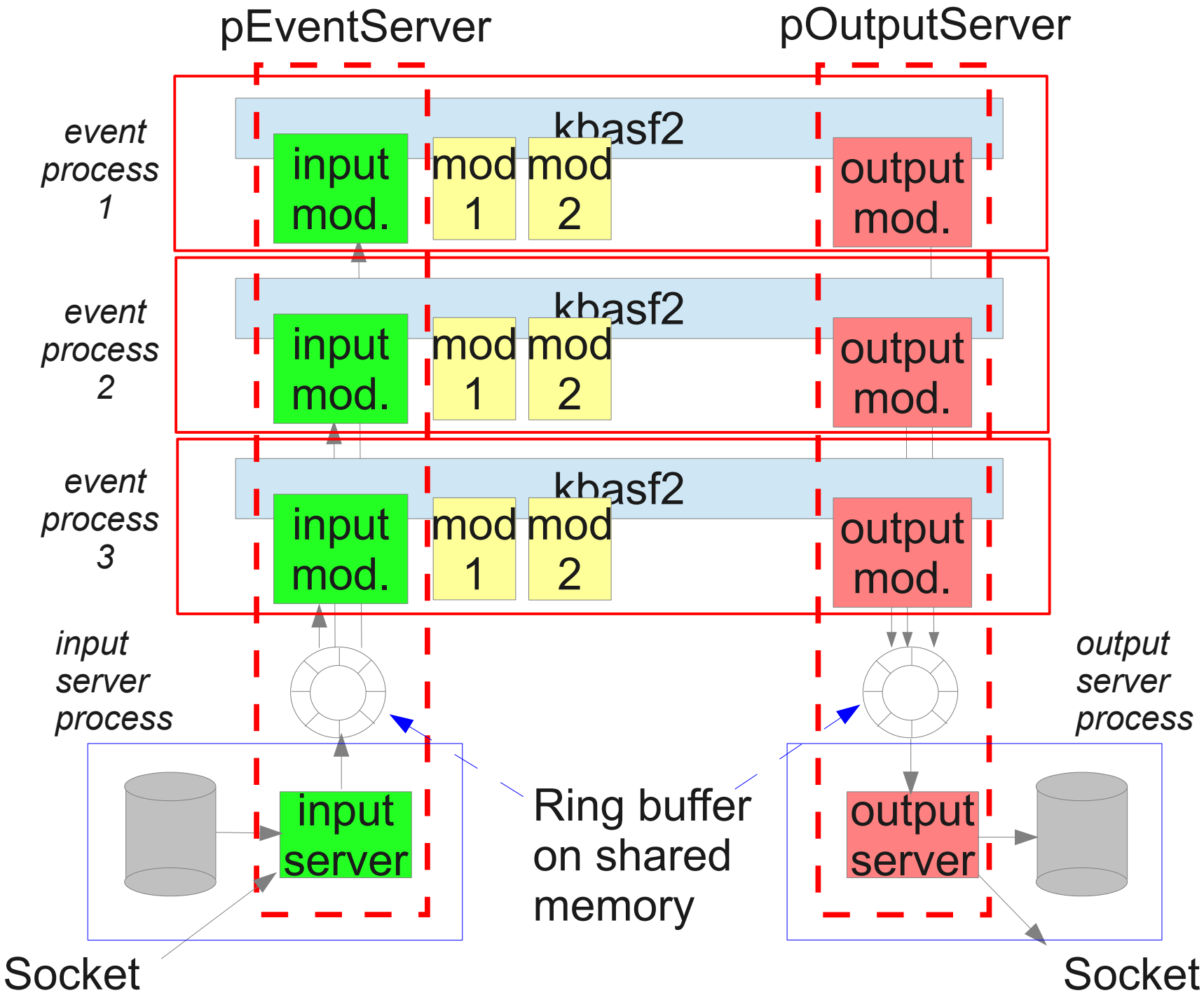
- Trivial “event-by-event” parallel processing is suitable for HEP data processing.

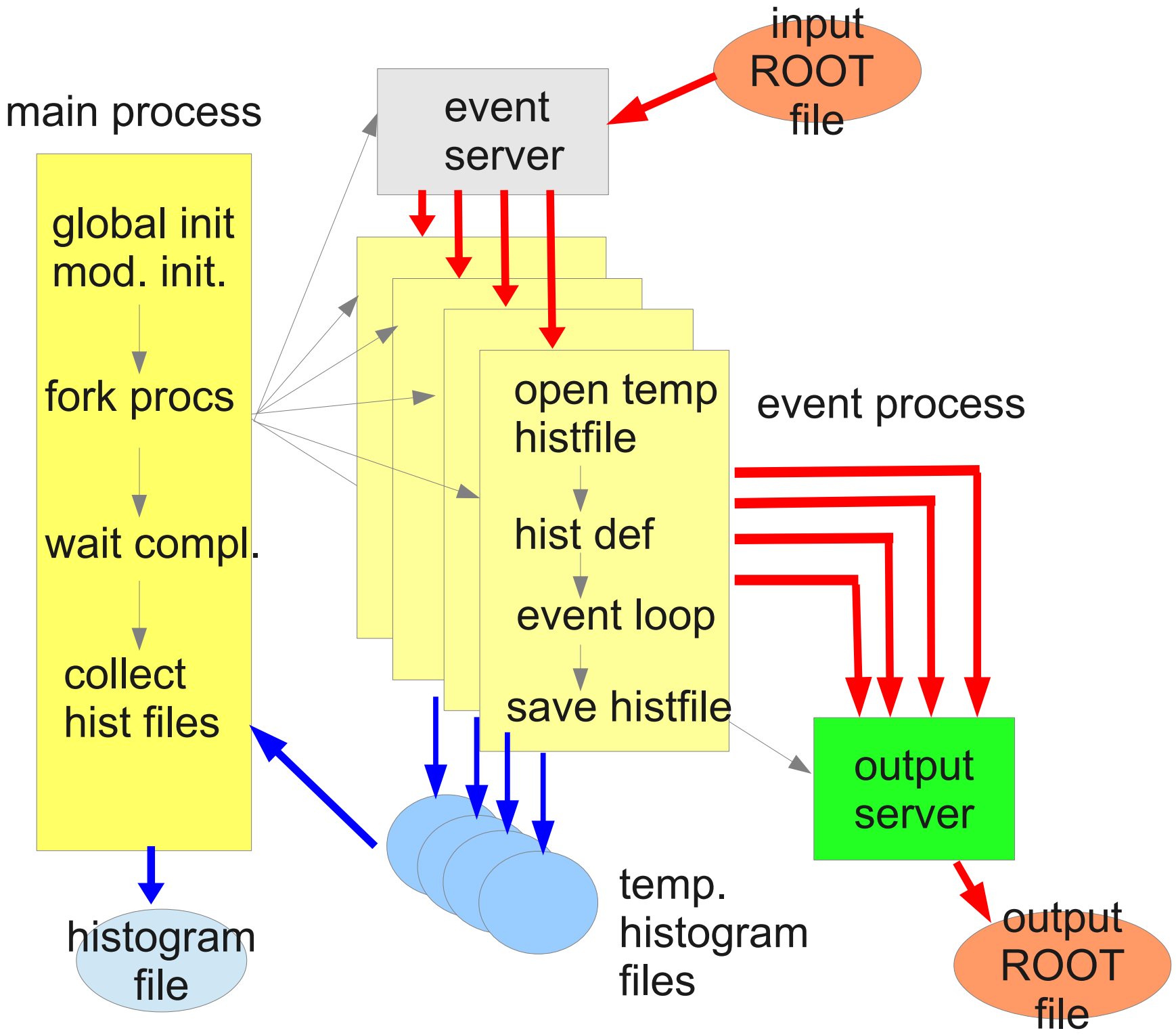
- The parallel processing utilizing “multi-process” (=using `fork()`) was already implemented in previous ***basf*** (Belle AnalysiS Framework) more than 15 years ago *(first presented at CHEP97)*.

“Why multi-process? Why not thread?”
* No special treatment for parallel processing is needed in user-programming.
* Separation of memory space for different event is ensured.
  - Common context is initialized before fork()
  - Processing of events in different memory space after fork()
    -> It was extensively used in Belle successfully.

- The design was inherited in ***basf2*** at the beginning of basf2 development.

Original implementation of basf2 parallel processing




input
ROOT
file
main process
event
server
global init
mod. init.
fork procs
wait compl.
collect
hist files
open temp
histfile
hist def
event loop
save histfile
event process
output
server
histogram
file
temp.
histogram
files
output
ROOT
file

However,

- This implementation needs special input/output modules for parallel processing.
  * I/O was managed inside framework in previous ***basf*** and it was not a user-concern.

  * I/O is provided by “modules” in ***basf2*** and a special treatment is not desired.

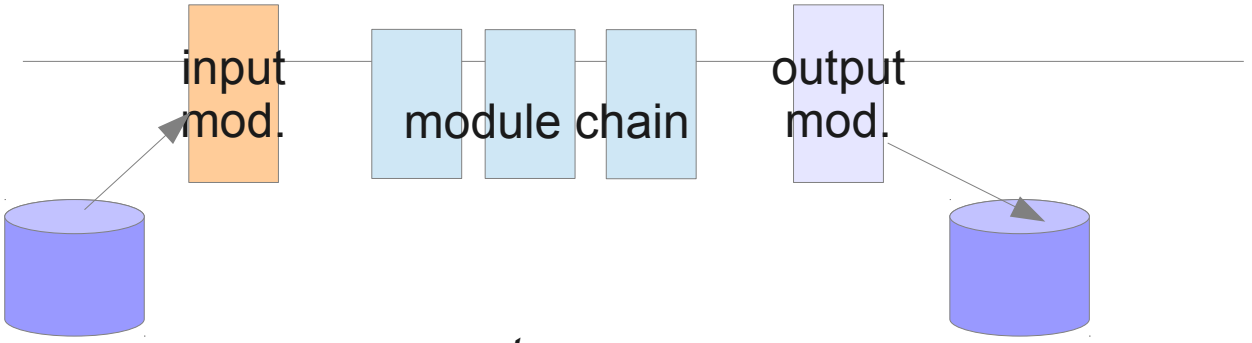
- When using event generators as an input module, a consistent management of random number seed in different processes was a problem in ***basf***.

Better implementation was proposed.
-> Pallarelize a part of path = “parallel path”

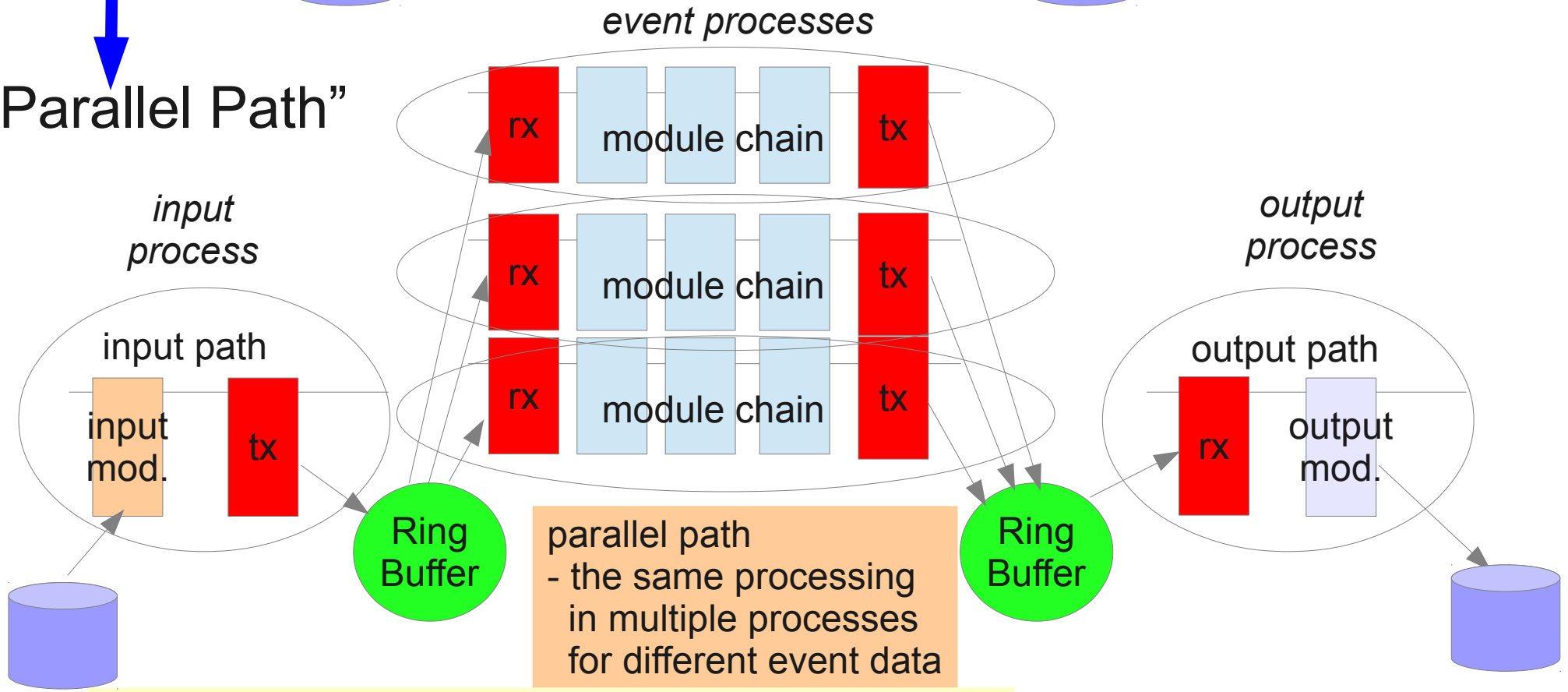
Partial Parallel Processing

Ryosuke Itoh et al 2012 J. Phys.: Conf. Ser. 396 022026

basf2 path



"Parallel Path"



parallel path
- the same processing
  in multiple processes
  for different event data

** Input module can also be an event generator module*
→ Unique random number sequence is ensured.

Object passing between processes

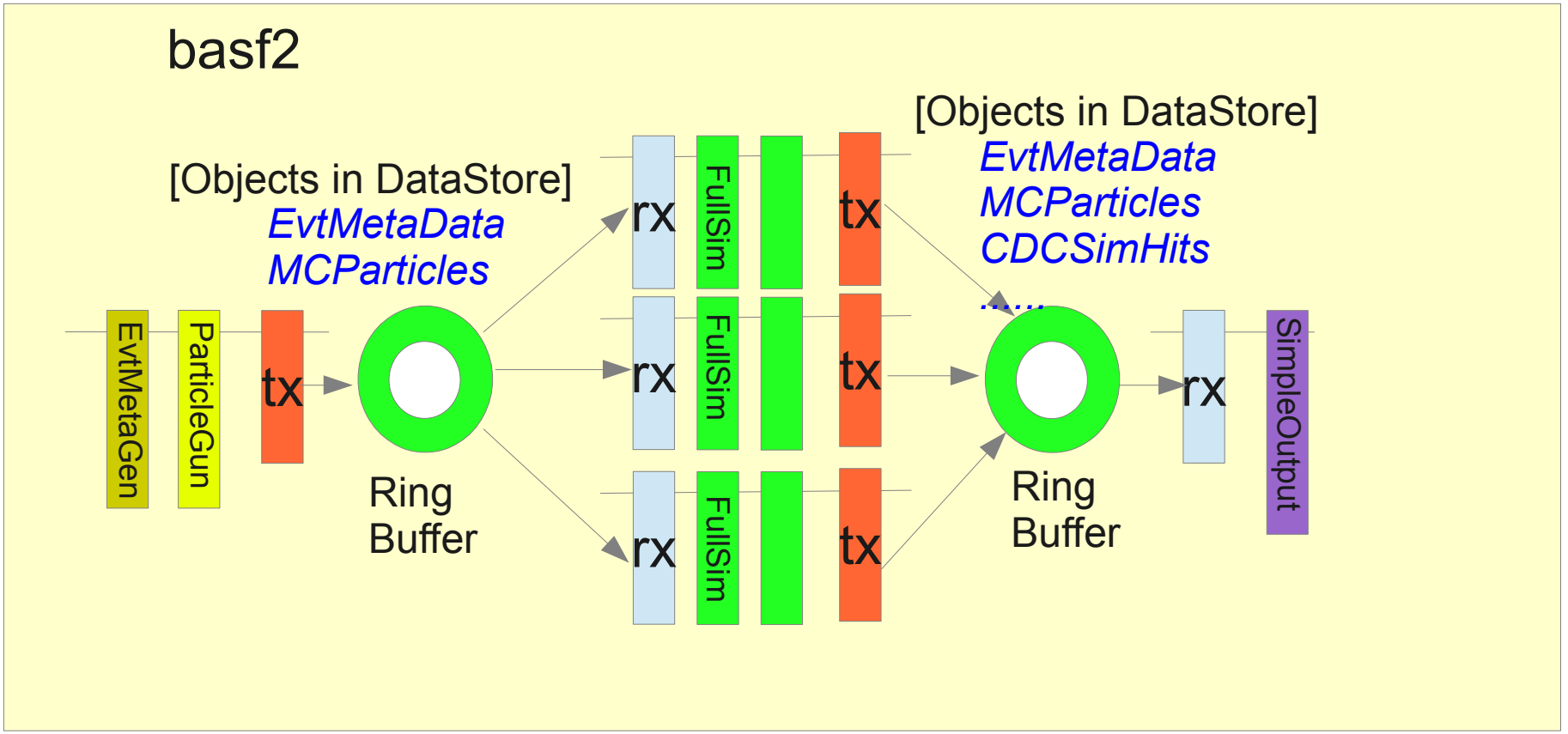
- To pass objects between processes, objects in DataStore are once streamed in a byte stream record by TMessage event by event.

- A transmitter(tx) module placed at the end of a path in a process streams the objects and place it in the ring buffer.

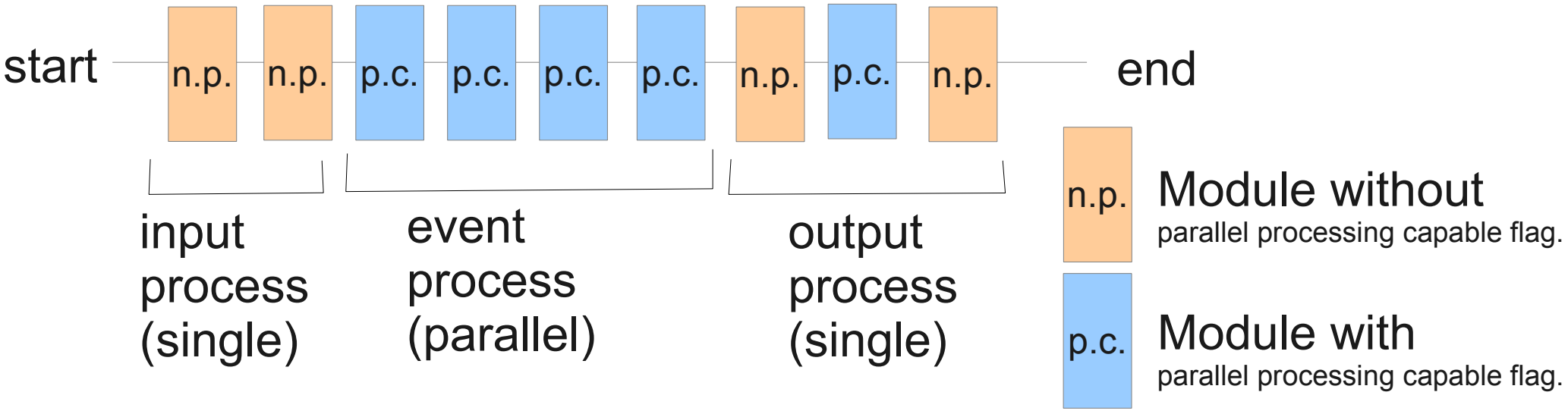
- The ring buffer is implemented using Linux IPC shared memory which is accessible by multiple processes.

- A receiver module(rx) picks up one event record from the ring buffer and then restores objects in DataStore.

- Multiple event processes are connected to a single ring buffer so that events are distributed/collected for the parallel processing.
  -> The load balancing of event processes is ensured by the ring buffer automatically.


basf2
[Objects in DataStore]
EvtMetaData
MCParticles
EvtMetaGen
ParticleGun
tx
Ring
Buffer
rx
FullSim
tx
rx
FullSim
tx
rx
FullSim
tx
[Objects in DataStore]
EvtMetaData
MCParticles
CDCSimHits
.....
Ring
Buffer
rx
SimpleOutput

Parallelizing a path: Path Analysis



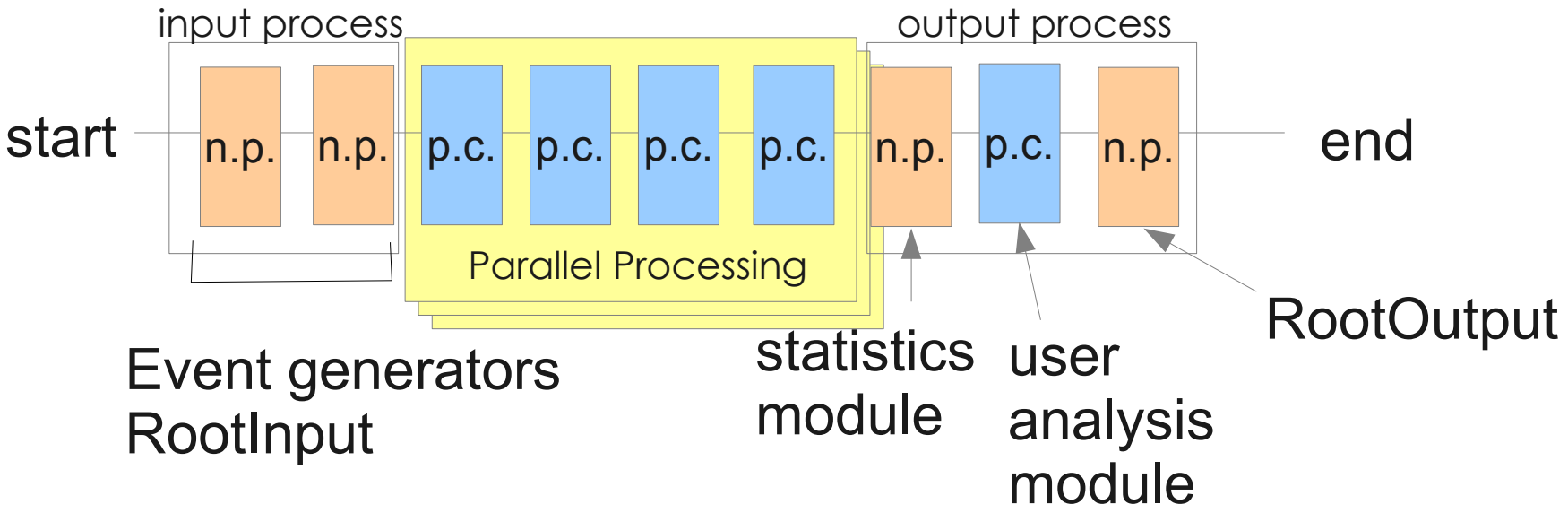
Each module has a flag telling framework that it is parallel-processing capable or not.

- Modules are examined from the beginning of a path.
- If a parallel-processing-capable module is found, the modules before it are placed in “input path” executed in a process.
- The module examination is continued and module w/ the flag are all placed in the “parallel path” until a module w/o the flag is found.
- The modules after the non-certified module are all placed in “output path”.

- Input path (Output path) is executed in a separate process.

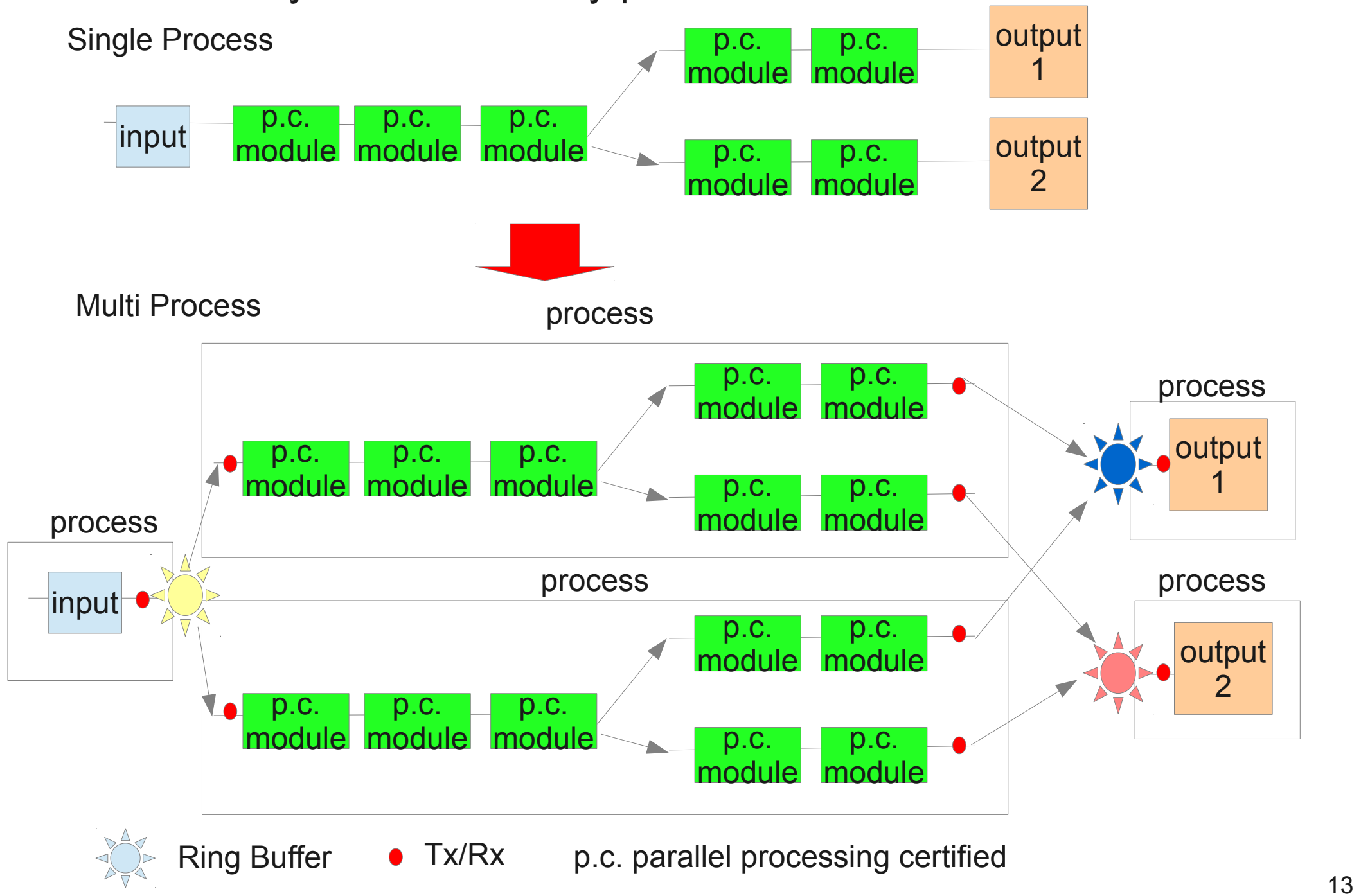
- Parallell path is executed in multiple processes whose number is specified by script

- Modules w/o parallel processing flag:
  * Event Generator
  * Input Module : RootInput, SeqRootInput
  * Output Module : RootOutput, SeqRootOutput
  * Analysis module, Statistics module.....



Parallelizing path with conditional branch

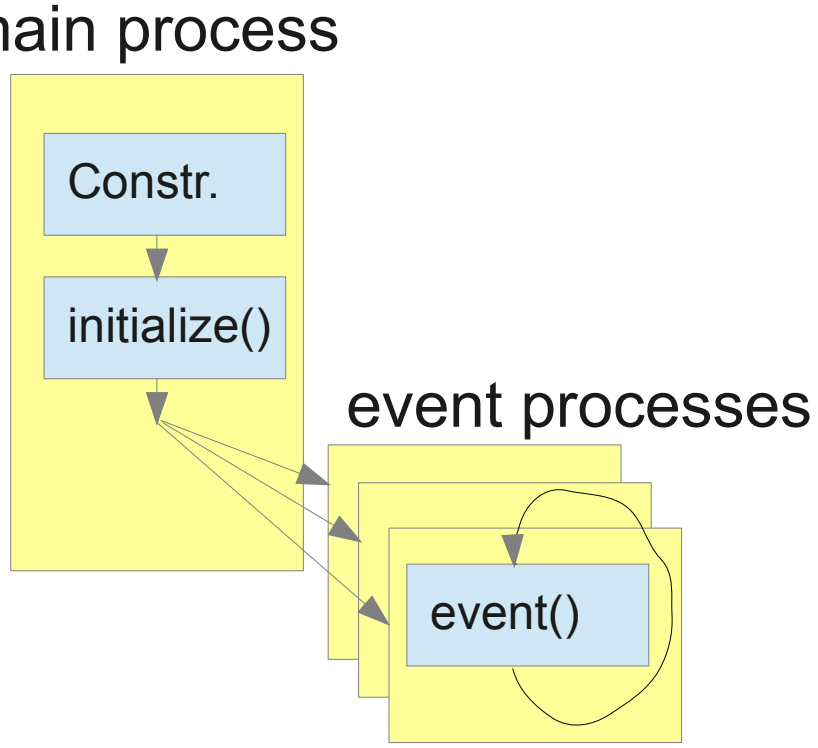
* Path analysis is recursively performed.



Initialization of module

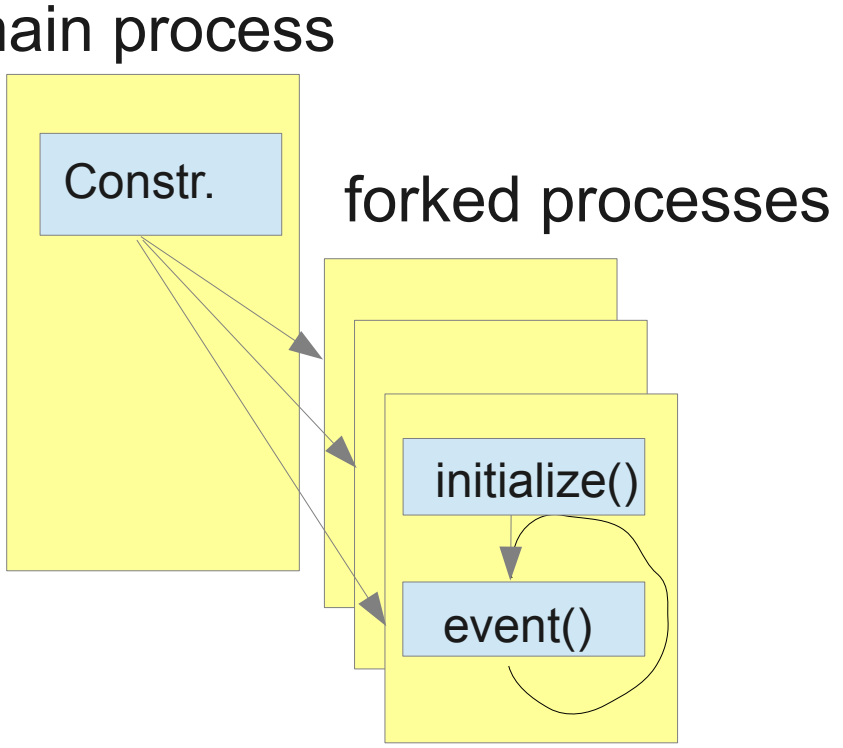
- Two cases for module initialization, before or after `fork()`
- It can be switched by setting a module flag
  `Module::c_InitializeInProcess / c_InitializeInMain` flag

w/ `c_InitializeInMain`



forked out after
initialize() is called

w/ `c_InitializeInProcess` (default)



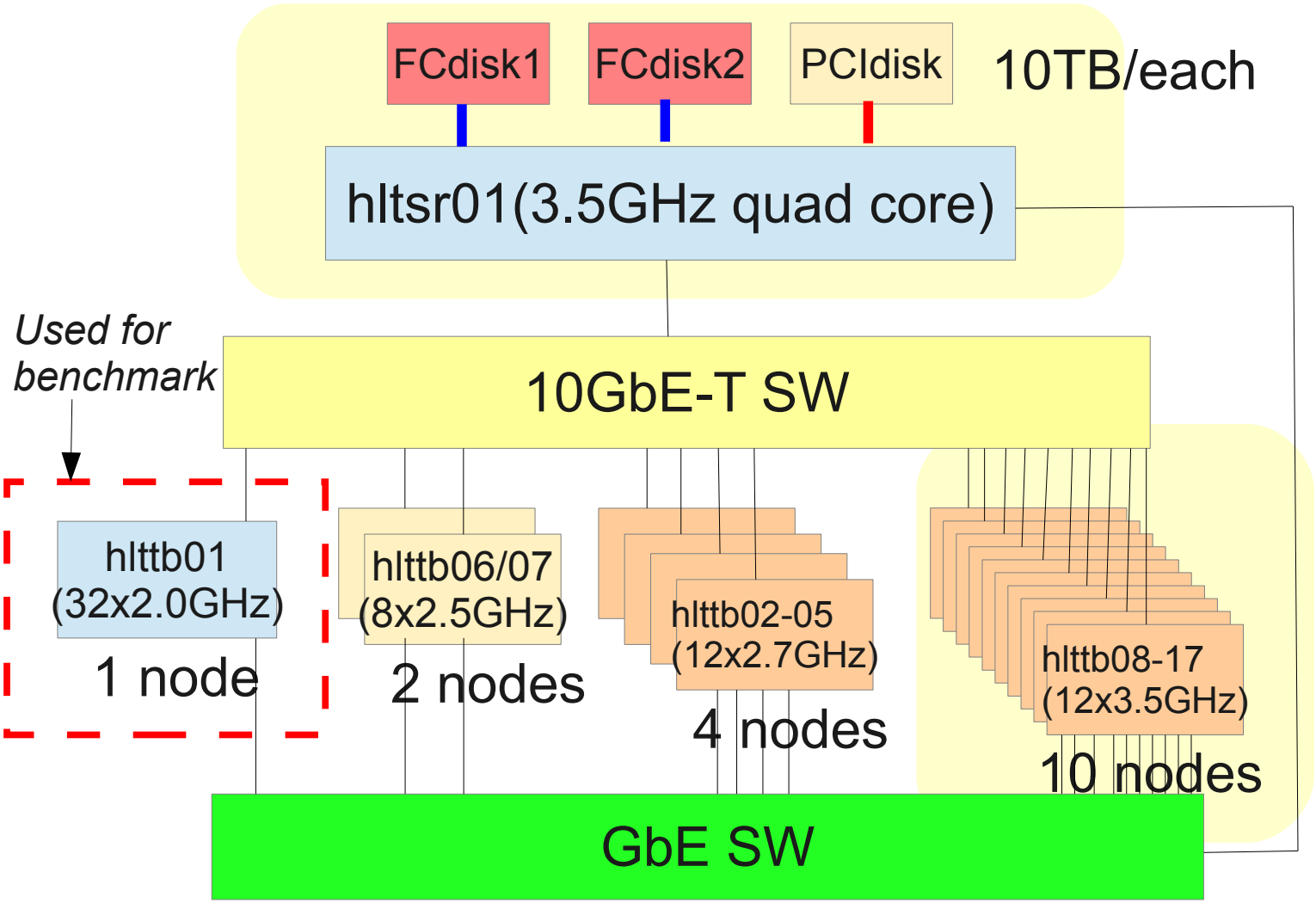
forked out after
initialize() is called

Example Script : Gen + Sim + Reco

```
# Histogram Manager
histoman = register_module ( 'HistoManager' );

# EvtMetaGen - generate event meta data
evtmetagen = register_module('EvtMetaGen')
evtmetagen.param('EvtNumList', [1000])

# Particle gun
particlegun = register_module('PGunInput')
particlegun.param('nTracks', 10)
particlegun.param('PIDcodes', [11, -11])
particlegun.param('pPar1', 0.05)
particlegun.param('pPar2', 3)

# Generator monitor
genmon = register_module ( 'GenMonitor' )

# Geometry parameter loader
paramloader = register_module('ParamLoaderXML')
paramloader.param('InputFileXML',
os.path.join(basf2datadir,'simulation/Belle2.xml'))

# Geometry builder
geobuilder = register_module('GeoBuilder')

# Full Geant4 simulation
g4sim = register_module('FullSim')

# Reco modules
# PXD
pxddigi = register_module ('PXDDigi')
pxdhit = register_module ('PXDRecoHitMaker')
# SVD
svddigi = register_module ('SVDDigi')
svdhit = register_module ( 'SVDRecoHitMaker')
vxdspace = register_module ( 'VXDSpacePointMaker' )

# Root file output
simpleoutput = register_module('SimpleOutput')
simpleoutput.param('outputFileName', 'simrec.root')

# Analysis Modules
simmon = register_module ( 'SimMonitor' )

# Create main path
main = create_path()

# Generator modules
main.add_module(evtmetagen)
main.add_module(histoman)
main.add_module(particlegun)
main.add_module(genmon)

# Simulation modules (parallel processing capable)
main.add_module(paramloader)
main.add_module(geobuilder)
#main.add_module(dump)
main.add_module(g4sim)
main.add_module(simmon)

# Reconstruction modules
main.add_module(pxddigi)
main.add_module(pxdhit)
main.add_module(svddigi)
main.add_module(svdhit)
main.add_module(vxdspace)

# Output path
main.add_module(simpleoutput)

# Process 1000 events with 64 CPUs
nprocess(64)
process(main)
```

Performance Measurement

- Belle II HLT test bench is used which is equipped with a number of multi-core PC servers



* SL5.5
* Network boot

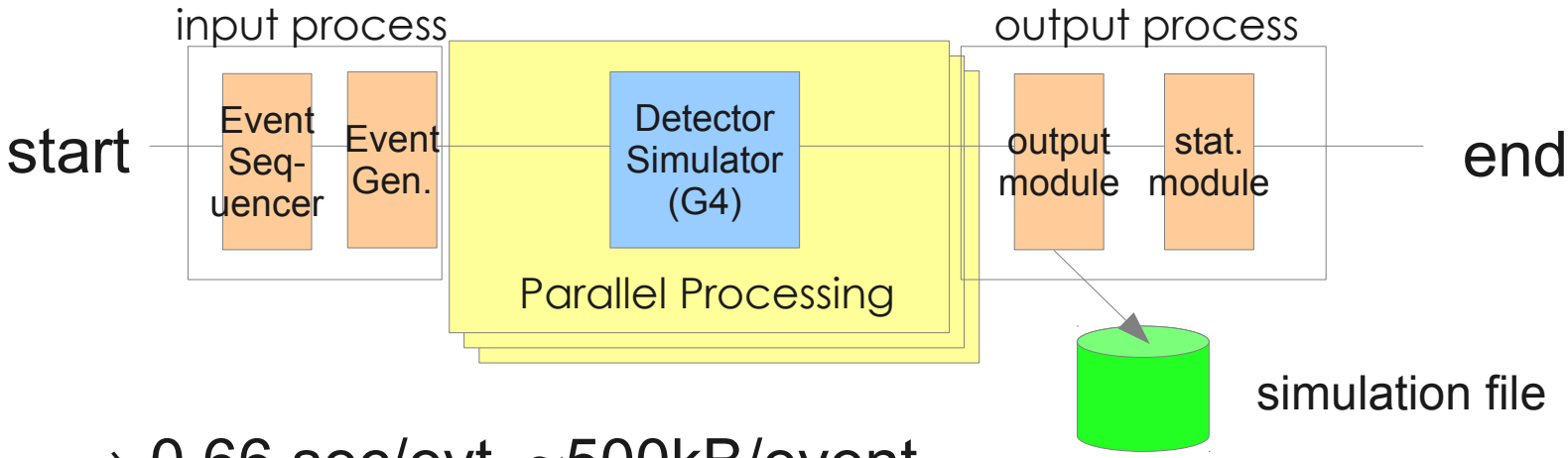
/home is placed in FCdisk2 and NFS mounted via GbE link.

Benchmarks

- The parallel processing performance is tested using a PC server with
  * 32 core CPUs (4 x Intel Xeon X7550 @ 2GHz),
  * 65GB memory,
  * Scientific Linux 5.5 (Kernel 2.6.18, 64bit).

- Two realistic benchmarks.
  * Elapsed time for 10,000 events is measured varying no. of event processes.

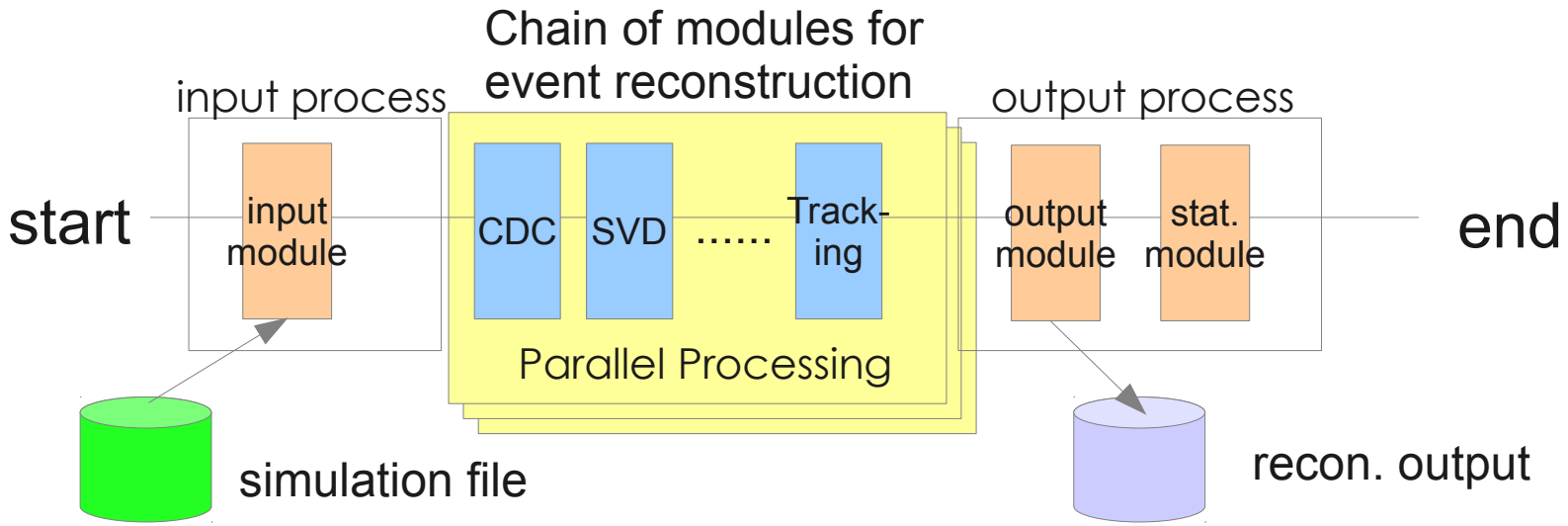
1) Event generation + Belle II detector simulation (tracker only)



$\Rightarrow$ 0.66 sec/evt, ~500kB/event

$\rightarrow$ max output rate = 23.2 MB/sec @ 28 processes

2) Belle II event reconstruction (tracking only)



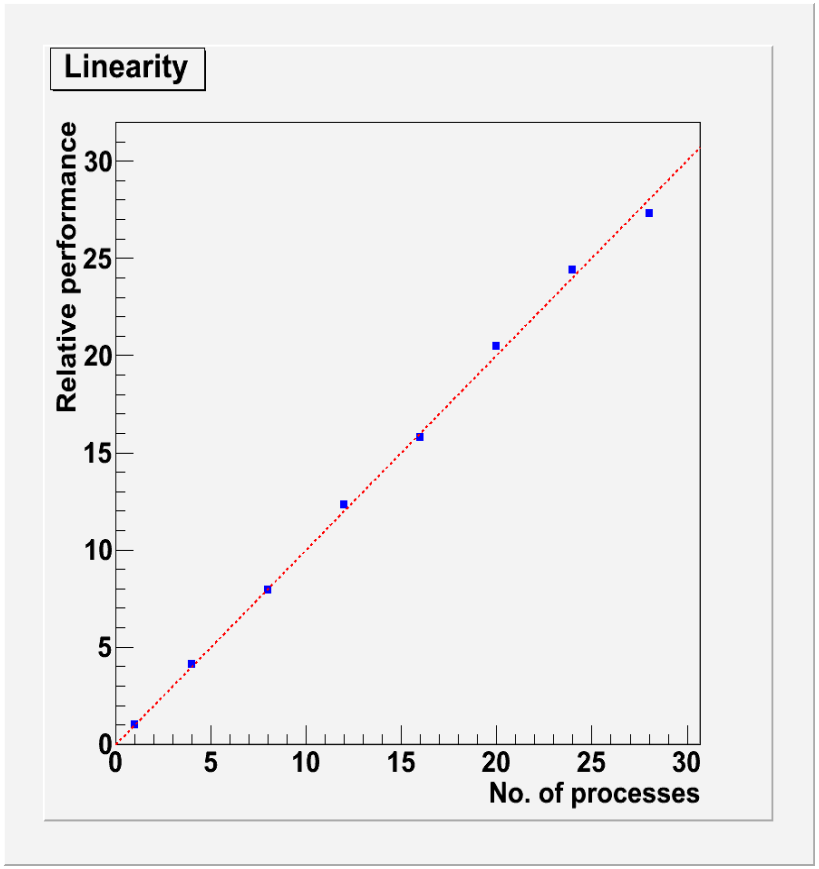
$\Rightarrow$ 0.29 sec/evt, input ~500kB/ev, output ~750kB/ev

$\rightarrow$ max output rate = 48.5MB/sec @ 28 processes

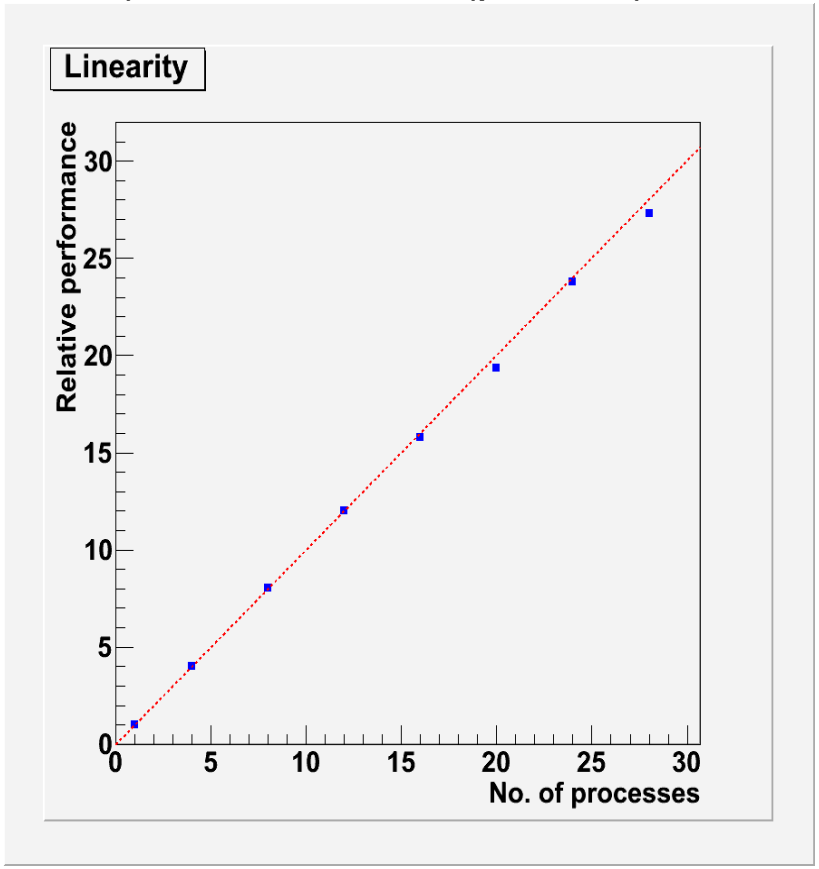
- The performance increase by parallel processing is studied by measuring the elapsed time for 10000 events varying the number of processes.
- The performance was measured using a 32-core server.

Performance linearity

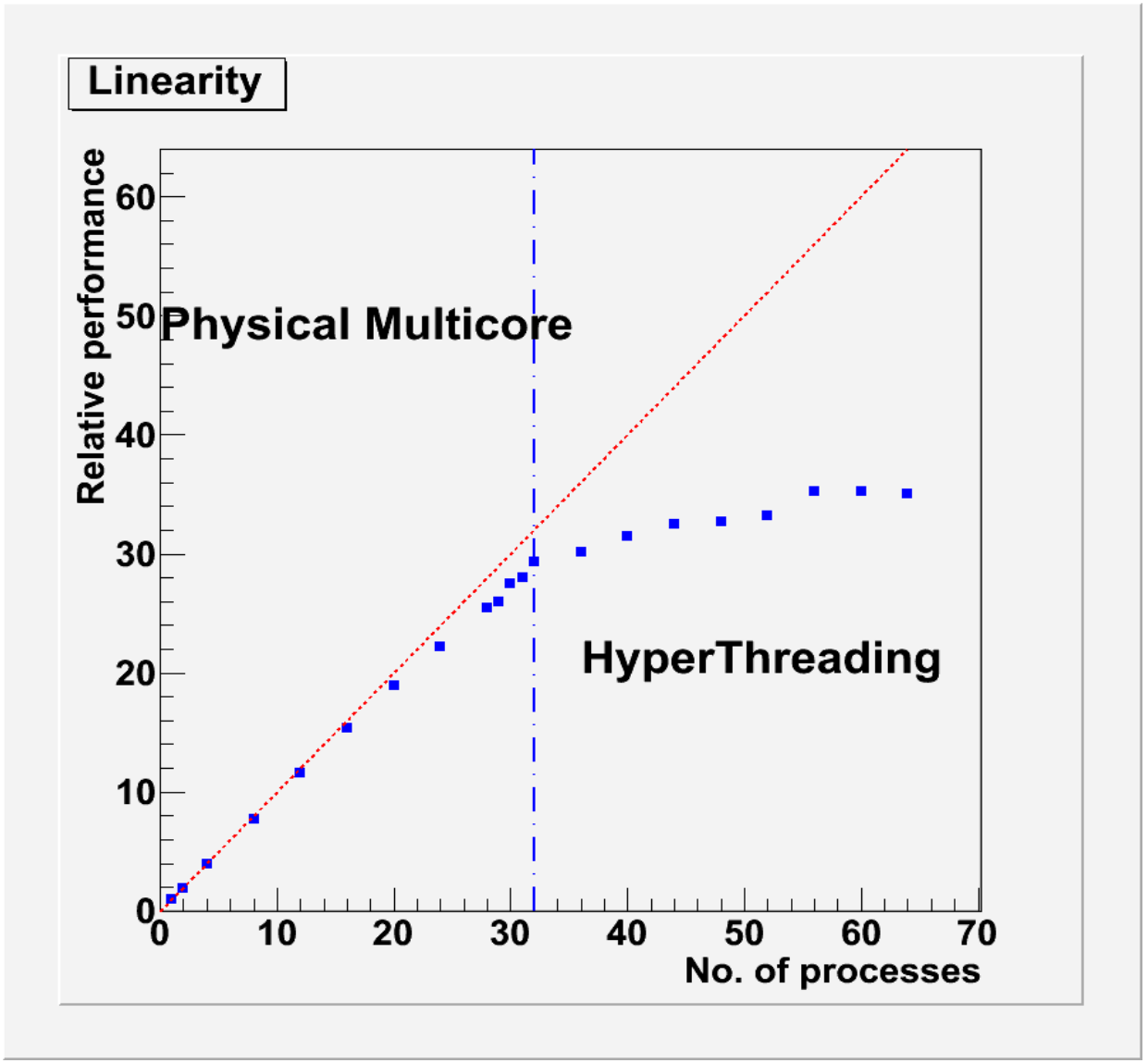
1) Event gen(single) + Simulation(parallel)



2) Event reconst (parallel)



Performance of HyperThreading



* Good linearity up to max. number of cores.

* Improvement by HyperThreading : ~ 1.26

Note: the benchmark used for the measurement is different from previous study.

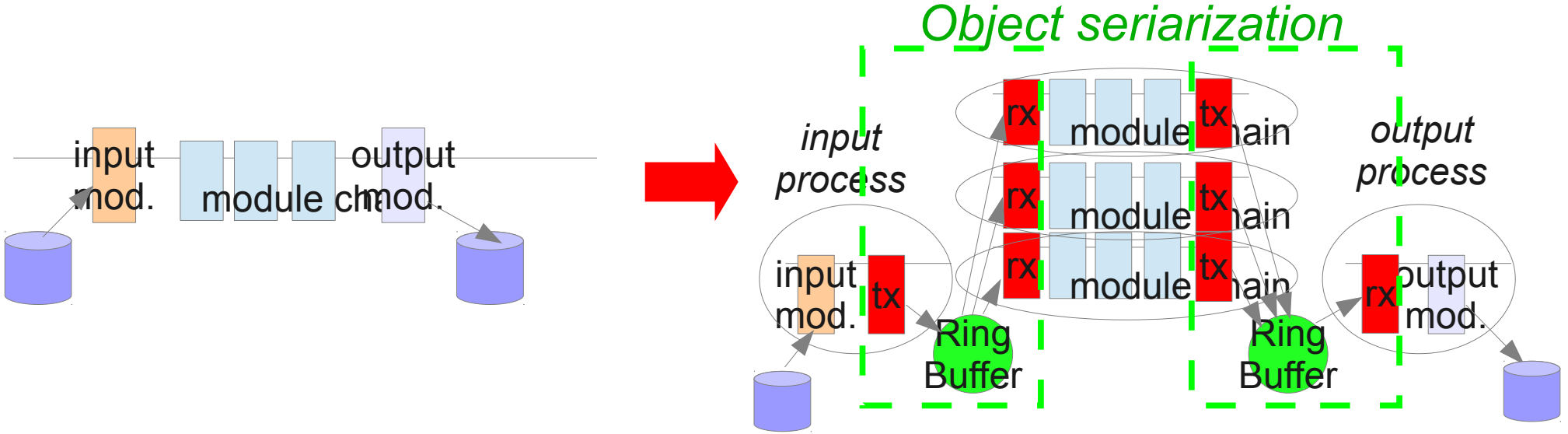
- Bottleneck in absolute performance :

Object streaming/destreaming for the ring buffer

a) Max. flow rate w/o parallel processing : 253.3MB/sec

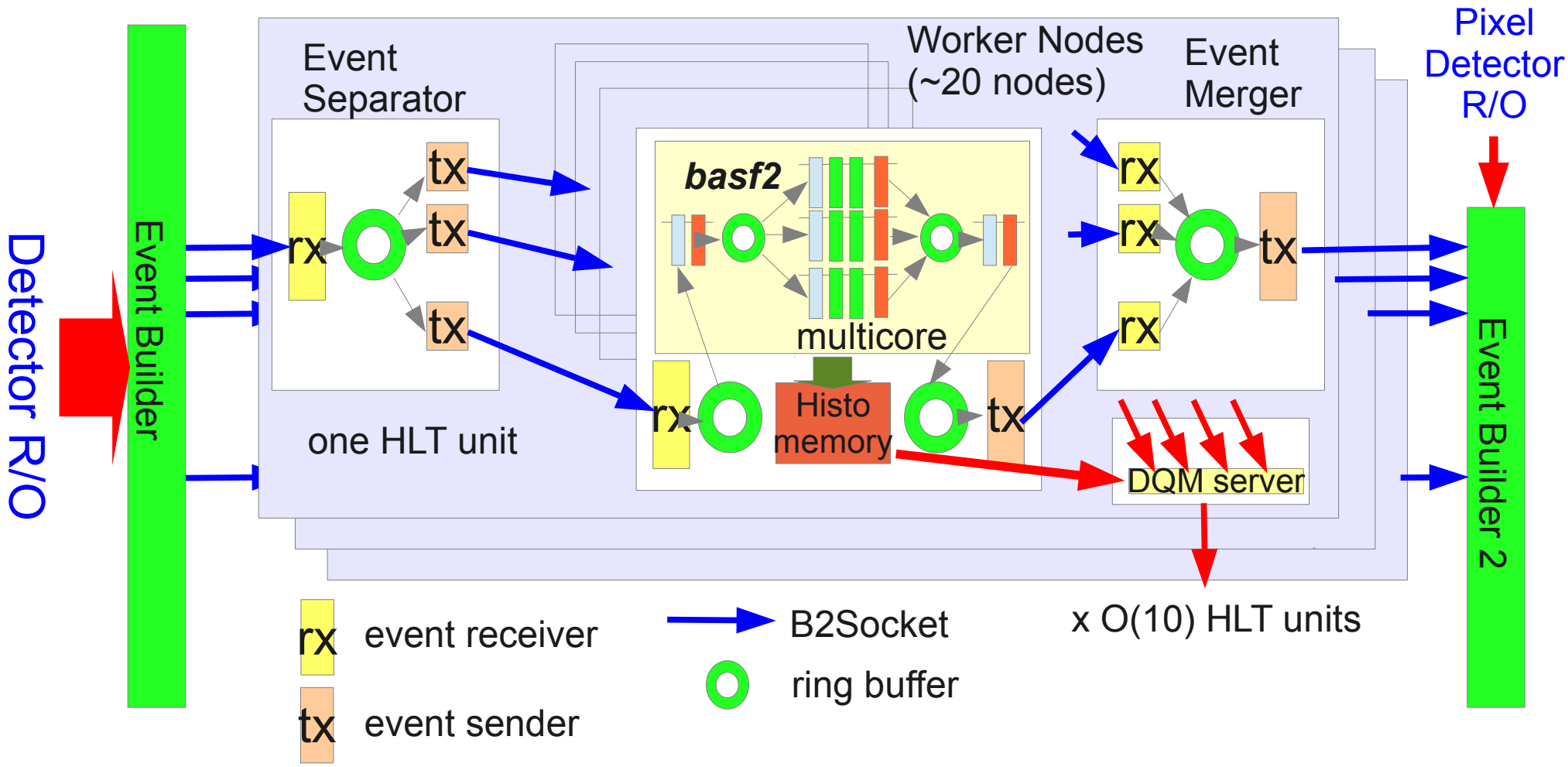
b) w parallel processing : 112.4MB/sec

→ Maximum flow rate becomes a half because of object streaming overhead but still good enough in “many core” parallel processing.

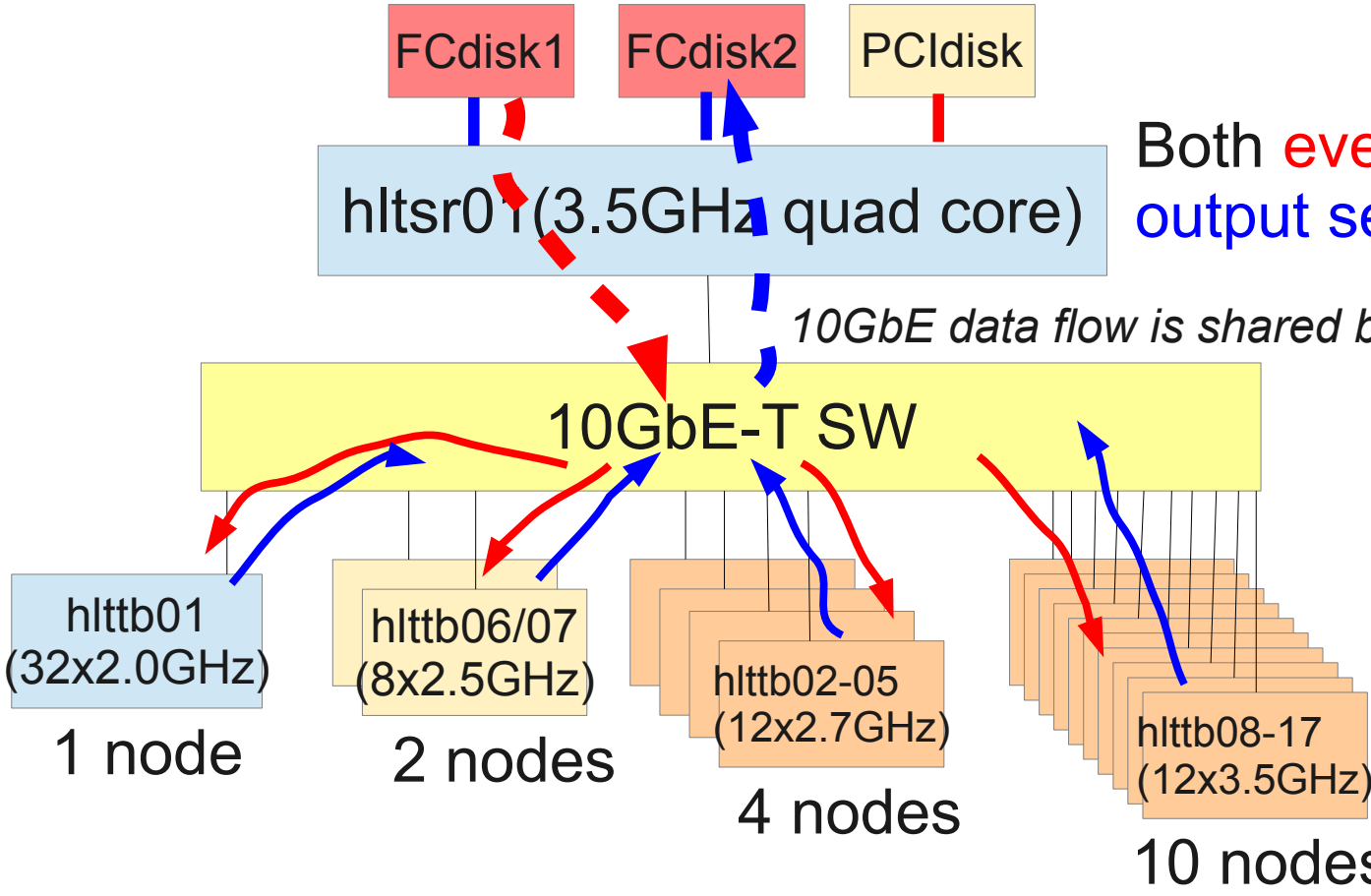


Application for High Level Trigger (HLT)

- Parallel processing is extended over network connected PC cluster.
- Based on the same object transfer technique developed for basf2
  = Object streaming + RingBuffer combined with network socket.



Benchmark Data Flow

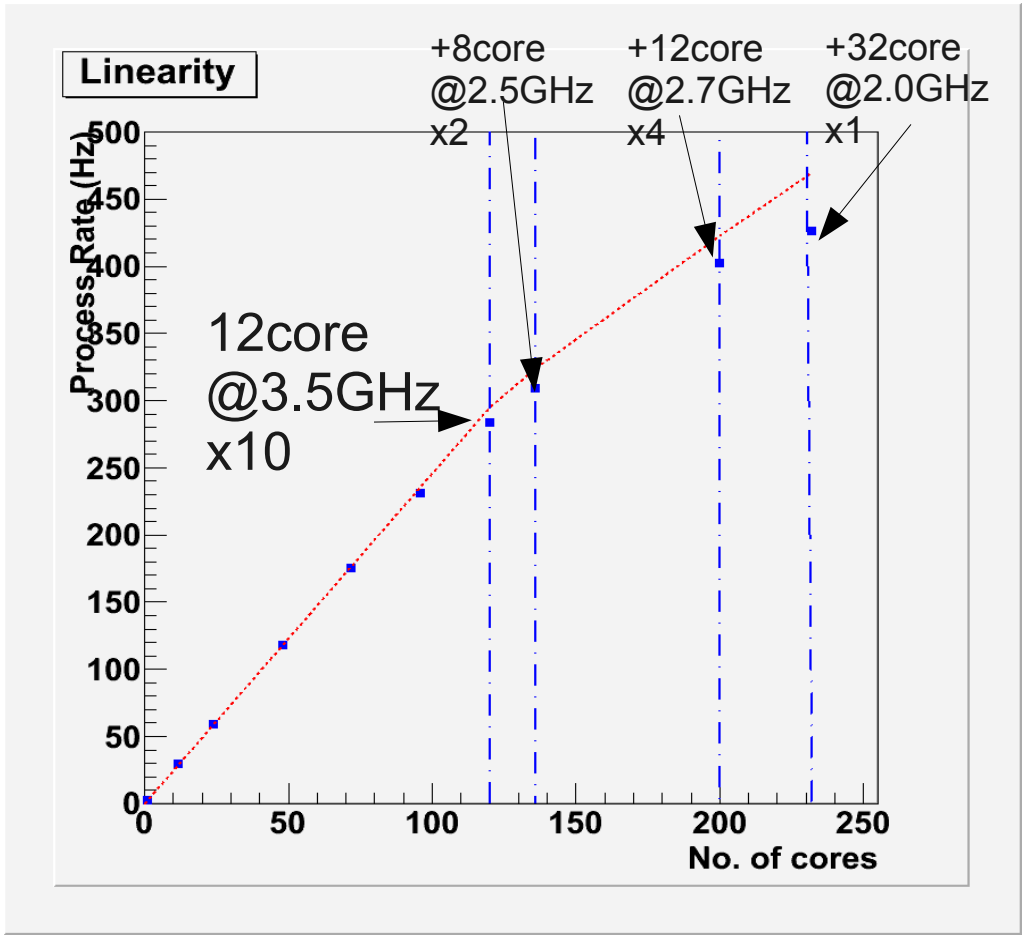


Both event server and output server run here.

10GbE data flow is shared by both input and output

basf2 runs on all nodes utilizing max. number of cores

Performance with all servers/cores in test bench



Benchmark : Belle2 recon.
input:500kB/ev, output:700kB/ev

Expected linear performance increase

* Linearity is kept for <200 cores.

* Some deterioration observed for a higher number of cores

* I/O bandwidth is reaching at maximum. (Total input/output rate is already reaching at >600MB/sec which is 10GbE max. BW).

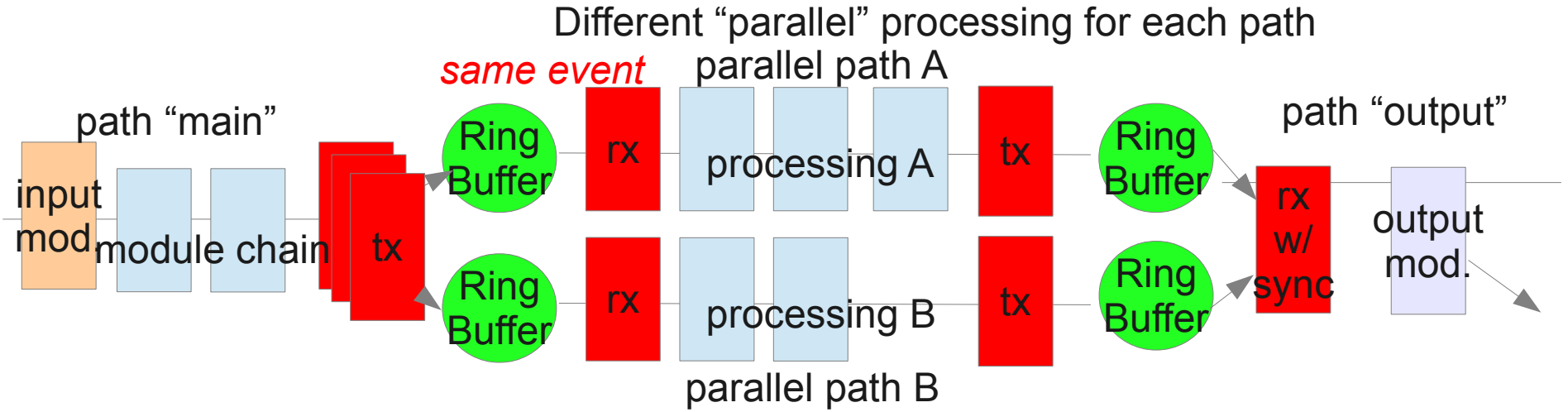
* Further study required.

Summary

- Event-by-event parallel processing is implemented in basf2 to utilize multi-core CPUs based on multi-process approach.
- The implementation enables the partial parallelization of processing chain.
- The performance linearity was studied up to ~30 cores and and the linearity was confirmed.

Plan

- Generalized implementation of parallel path :
  parallelized-pipeline processing within one event
  -> to reduce the latency per event



* Dynamic layout of modules/paths to optimize the data flow.
* Extension of pipeline over network PC-cluster

Backup Slides

Histogram Management

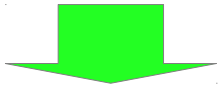
- Histograms(incl. N-tuples, user-defined TTrees) accumulated in a basf2 session can be consistently managed using "HistoManager" module.

- Histograms are all stored in a single ROOT file whose file name is specified by a parameter of HistoManager.

- The usage is unchanged from previous version.
  Module:
  * Modules to accumulate histograms are required to be derived from "HistoModule" class instead of "Module" class.
  * Histogram are supposed to be defined in a function defineHisto()
  Script:
  * Place "HistoManager" module right after "Master Module" (i.e. SimpleInput, EvtMetaGen, etc.)

- The histograms are automatically merged for the case that parallel processing is turned on.

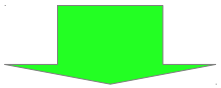
```
# Generator modules
main.add_module(evtmetagen)
main.add_module(histoman)
main.add_module(particlegun)
main.add_module(genmon)

# Simulation modules (parallel processing capable)
main.add_module(paramloader)
main.add_module(geobuilder)
#main.add_module(dump)
main.add_module(g4sim)
main.add_module(simmon)

# Output module
main.add_module(simpleoutput)
```



```
[INFO] process : inlistpath size = 1
[INFO] process : bodypathlist size = 1
[INFO] process : outpathlist size = 1
[INFO] Input Path (0x26f901a0) : EvtMetaGen -> HistoManager -> PGunInput -> GenMonitor ->
Tx52658189
[INFO] Main Path (0x26f8fed0) : Rx52658189 -> HistoManager -> ParamLoaderXML -> GeoBuilder
-> FullSim -> SimMonitor -> Tx52690958
[INFO] Output Path (0x26f90080) : Rx52690958 -> HistoManager -> SimpleOutput
```

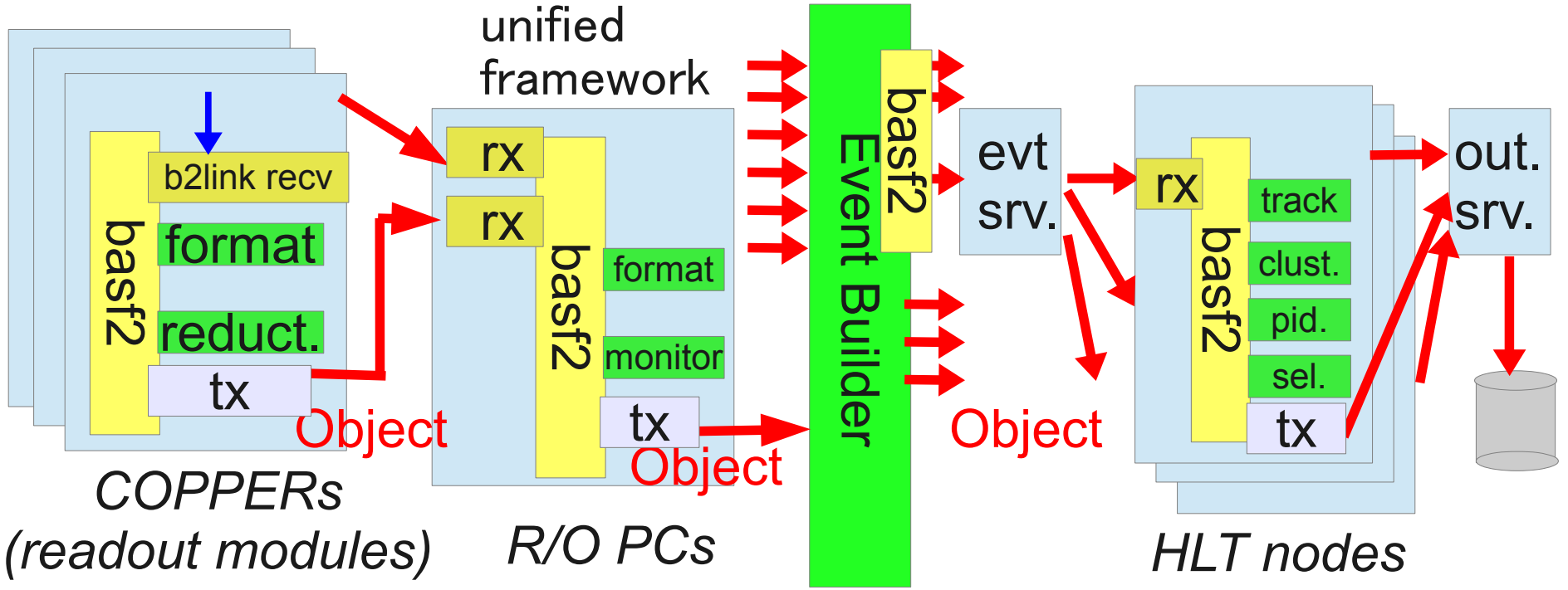
```
# Create Condition Path
condpath = create_path()
cond.condition ( condpath )
pcondpath = create_path()
pcond.condition ( pcondpath )

# Add modules to main path
main.add_module(evtmetagen)
main.add_module(histoman)
main.add_module(particlegun)
main.add_module(genmon)
main.add_module(cond)
```

```
# Parallel processing path from here
main.add_module(paramloader)
main.add_module(geobuilder)
#main.add_module(dump)
main.add_module(g4sim)
main.add_module(simmon)
main.add_module(pcond)

# output path
main.add_module(simpleoutput)

# Add modules to cond path
condpath.add_module(dump)

# Add module to pcondpath
pcondpath.add_module(seqout)
pcondpath.add_module(simmon)
```

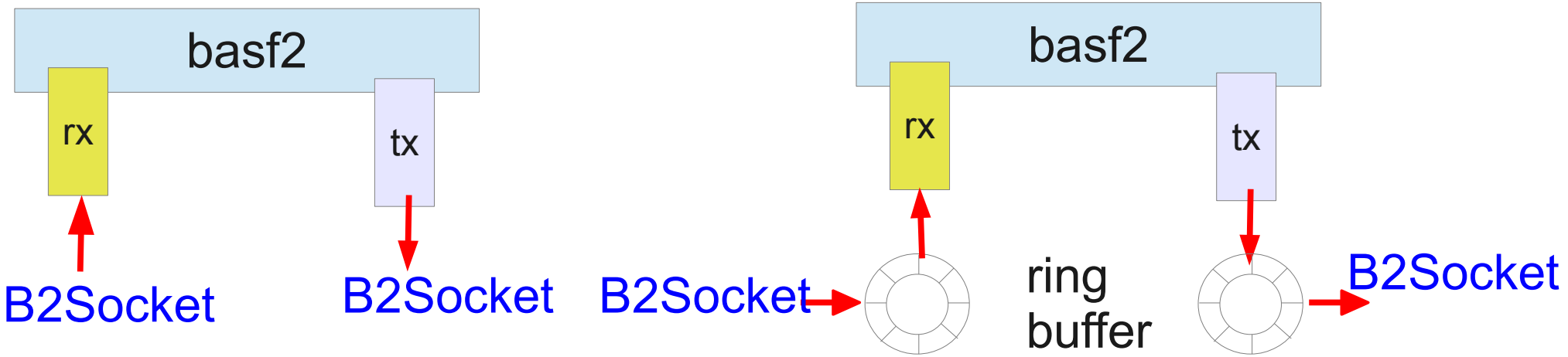


```
[INFO] process : inlistpath size = 1
[INFO] process : bodypathlist size = 3
[INFO] process : outpathlist size = 2
[INFO] Input Path (0x10001af0) : EvtMetaGen -> HistoManager -> PGunInput -> GenMonitor ->
TestCondition[->0xeceae60]  -> Tx7241730
[INFO] Main Path (0xeceae60) : PrintCollections
[INFO] Main Path (0x10001850) : Rx7241730 -> HistoManager -> ParamLoaderXML -> GeoBuilder
-> FullSim -> pTestCondition[->0xfc2fd50]  -> Tx7307269
[INFO] Main Path (0xfc2fd50) : Tx7274500
[INFO] Output Path (0x100019e0) : Rx7307269 -> HistoManager -> SimpleOutput
[INFO] Output Path (0xeceaea0) : Rx7274500 -> HistoManager -> SeqRootOutput -> SimMonitor
```

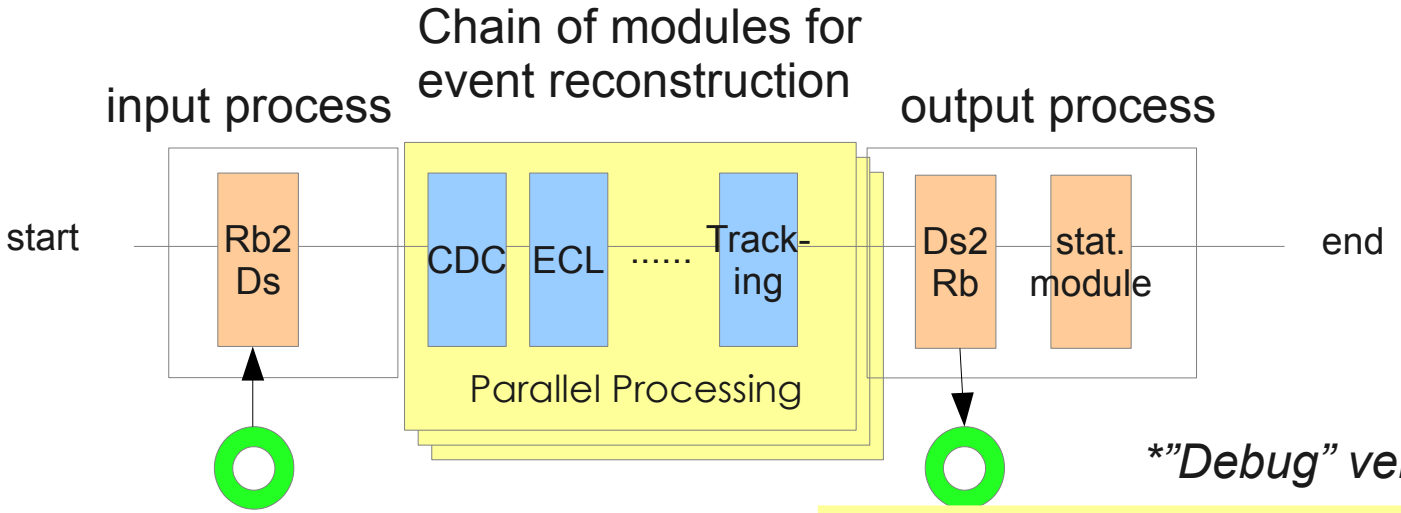
Data flow : Belle2 software framework(basf2) + OO data transfer



- Raw data are stored in ROOT objects by COPPER CPU.
- Objects are streamed and transferred between nodes using "B2Socket" class



Performance of parallel processing with PC cluster



*"Debug" version library was used

```
main = create_path()

main.add_module(input)

main.add_module(gearbox)
main.add_module(geometry)

main.add_module(trasan)
main.add_module(mcmatching)
main.add_module(cdcfitting)
main.add_module(topdigi)
main.add_module(topreco)
main.add_module(arichDigi)
main.add_module(arichRec)

main.add_module(output)
main.add_module(progress)
main.add_module(elapsed)

nprocess(12)
process(main)
```

Input file : (500kB/ev, 50000 events)
* generated by ParticleGun (10 pions)
* FullSim + CDCDigi + CDCBG

Processing chain:
trasan + mcmatching + genfit+
topdigi/reco + arichdigi/reco

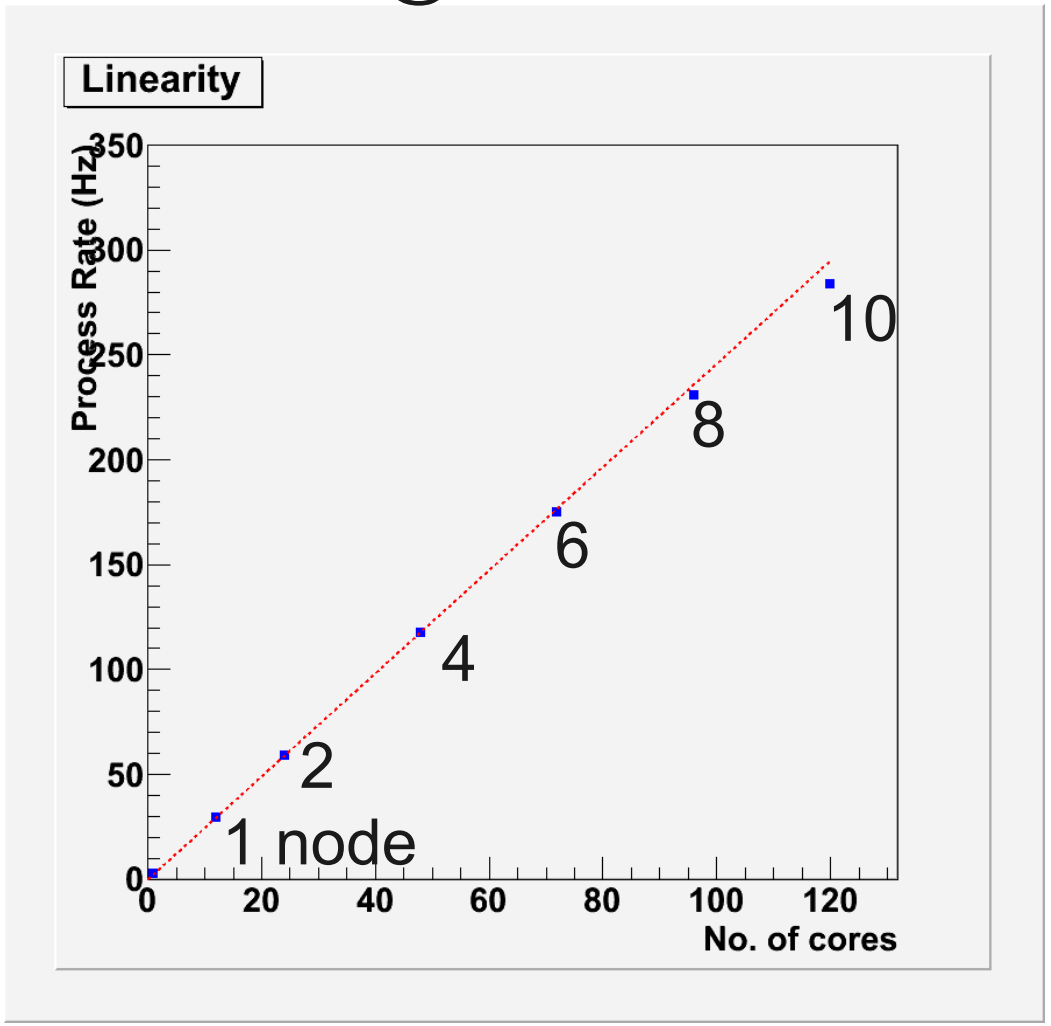
Output file : 700kB/ev

Processing speed :
1.5Hz with one 2.0GHz core

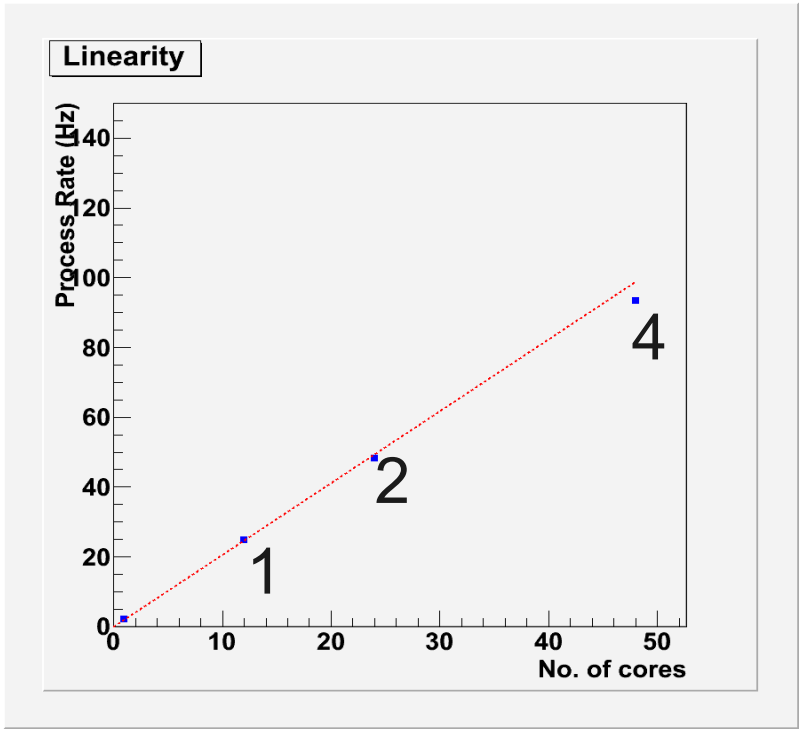
* Parallel processing on multicore is turned on.
* nprocess() is set to be the no. of maximum cores of each node

Measured performance

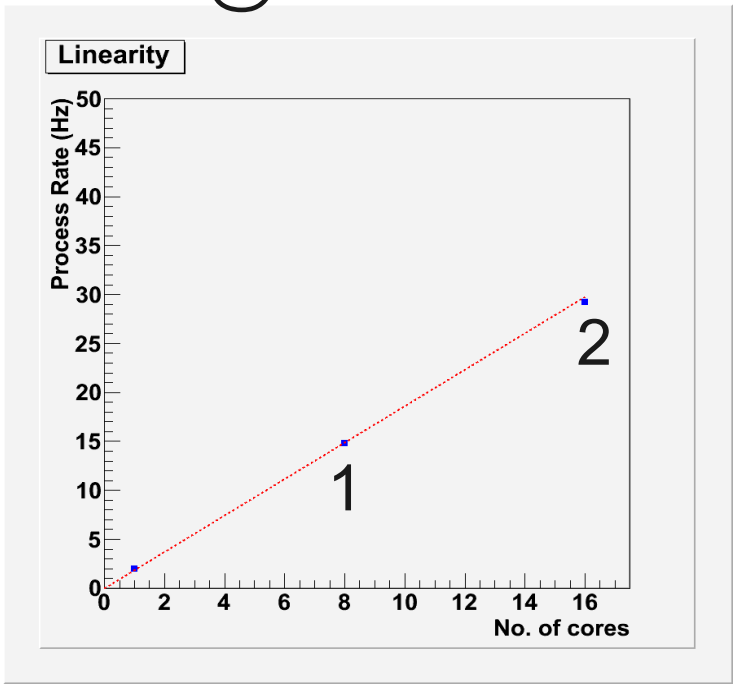
12 cores @ 3.5GHz / server



12 cores @ 2.7GHz / server



8 cores @ 2.5GHz / server



All available cores in each node are used for the processing (except HT).