Extending the FairRoot framework to allow for simulation and reconstruction of free streaming data

Mohammad Al-Turany

Dennis Klein

Anar Manafov

Alexey Rybalchenko

Florian Uhlig

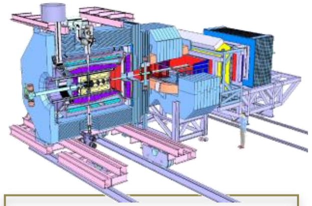
(GSI Darmstadt)

What Does it Mean?

- Introduce pipelined data processing to the current Framework. (This talk!)
- Introduce time based simulation instead of event wise one. (already shown in CHEP 2012)
- Ideally: Keep compatibility to the current offline scheme.

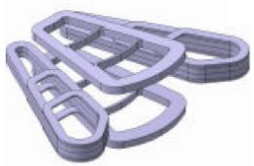
FairRoot

Start testing the VMC concept for CBM

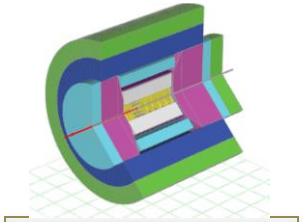


Panda decided to join->

FairRoot: same Base package for different experiments

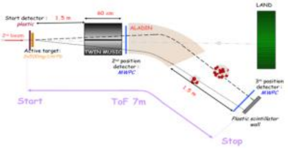


R3B joined



EIC (Electron Ion Collider BNL)

EICRoot



SOFIA (Studies On Fission with Aladin)

2004

2006

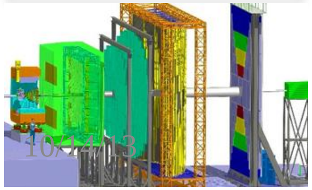
2010

2011

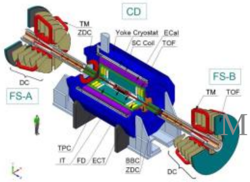
2012

2013

First Release of CbmRoot

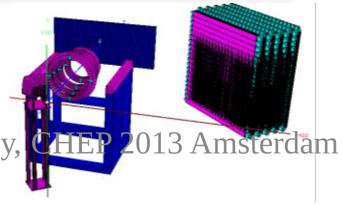


MPD (NICA) start also using FairRoot

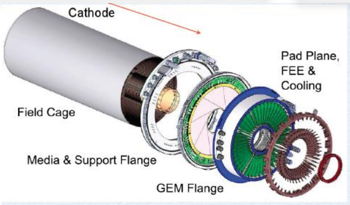


ASYEOS joined

(ASYEOSRoot)



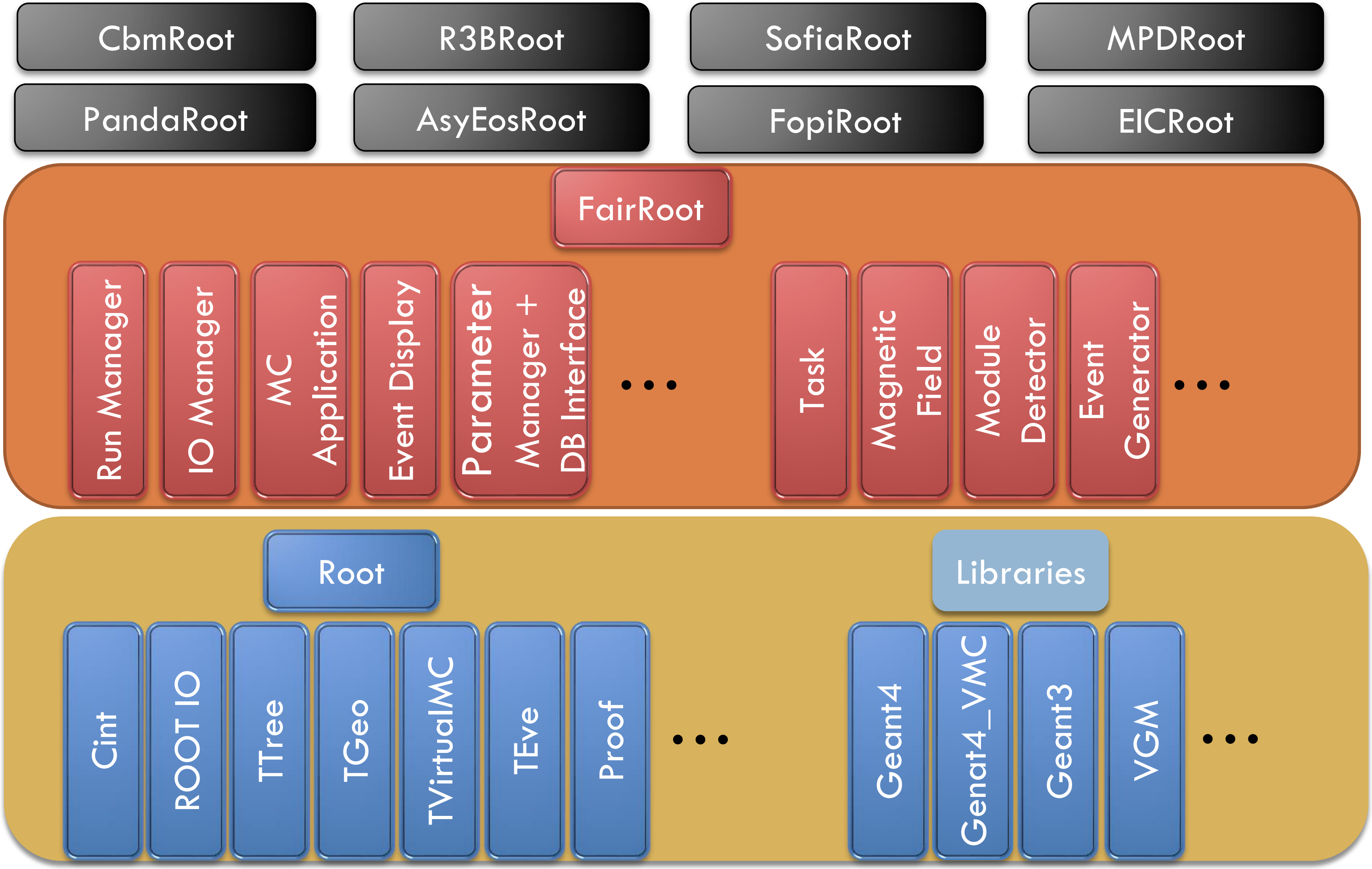
GEM-TPC seperated from PANDA branch (FOPIRoot)



ENSAR-ROOT

Collection of modules used by structural nuclear phsyics exp.

FairRoot: Implementation



CbmRoot
R3BRoot
SofiaRoot
MPDRoot
PandaRoot
AsyEosRoot
FopiRoot
EICRoot

FairRoot

Run Manager
IO Manager
MC Application
Event Display
Parameter Manager + DB Interface
…
Task
Magnetic Field
Module Detector
Event Generator
…

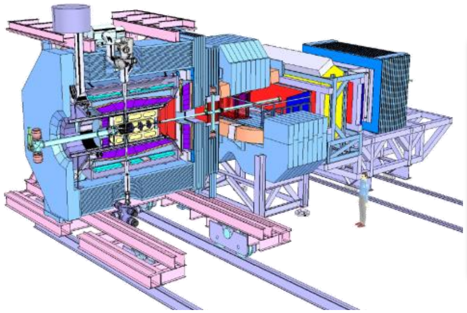
Root

Cint
ROOT IO
TTree
TGeo
TVirtualMC
TEve
Proof
…

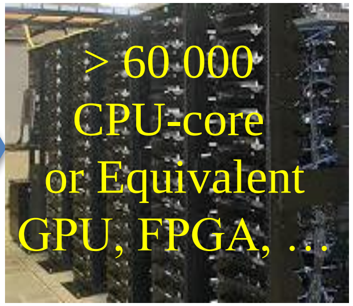
Libraries

Geant4
Genat4_VMC
Geant3
VGM
…

Next challenge is: Online vs. Offline or Online + Offline ?



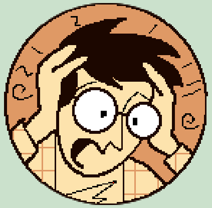
300 GB/s
20M Evt/s

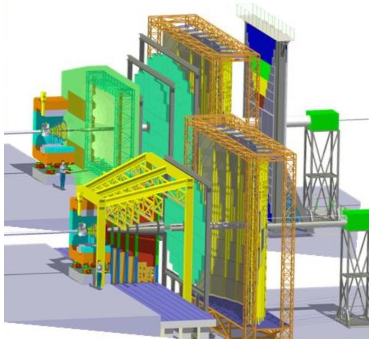


> 60 000
CPU-core
or Equivalent
GPU, FPGA, …

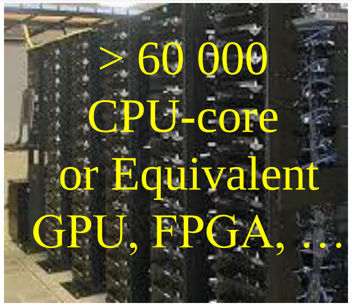
< 1 GB/s
25K Evt/s

How to distribute the processes?
How to manage the data flow?
How to recover processes when they crash?
How to monitor the whole system?
……





1 TB/s



> 60 000
CPU-core
or Equivalent
GPU, FPGA, …

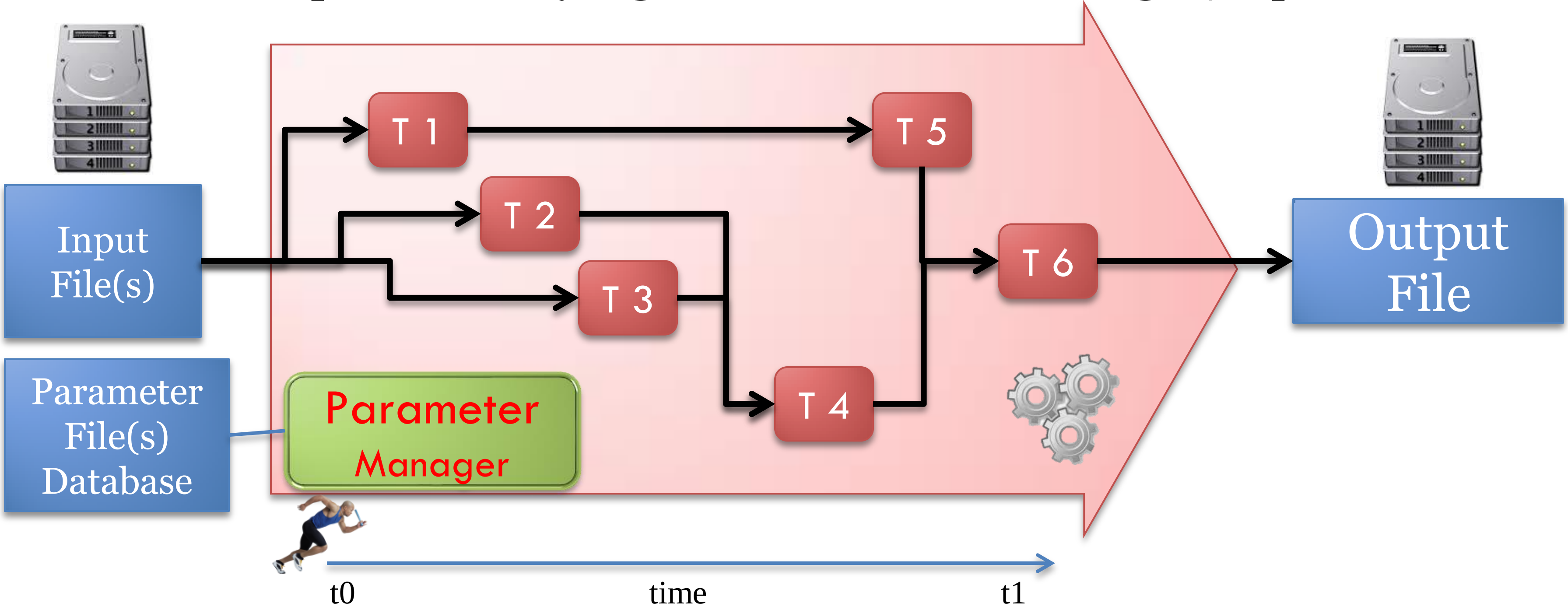
1 GB/s

Design constrains

- Highly flexible:
  - different data paths should be modeled.
- Adaptive:
  - Sub-systems are continuously under development and improvement
- Should work for simulated and real data:
  - developing and debugging the algorithms
- It should support all possible hardware where the algorithms could run (CPU, GPU, FPGA)
- It has to scale to any size! With minimum or ideally no effort.

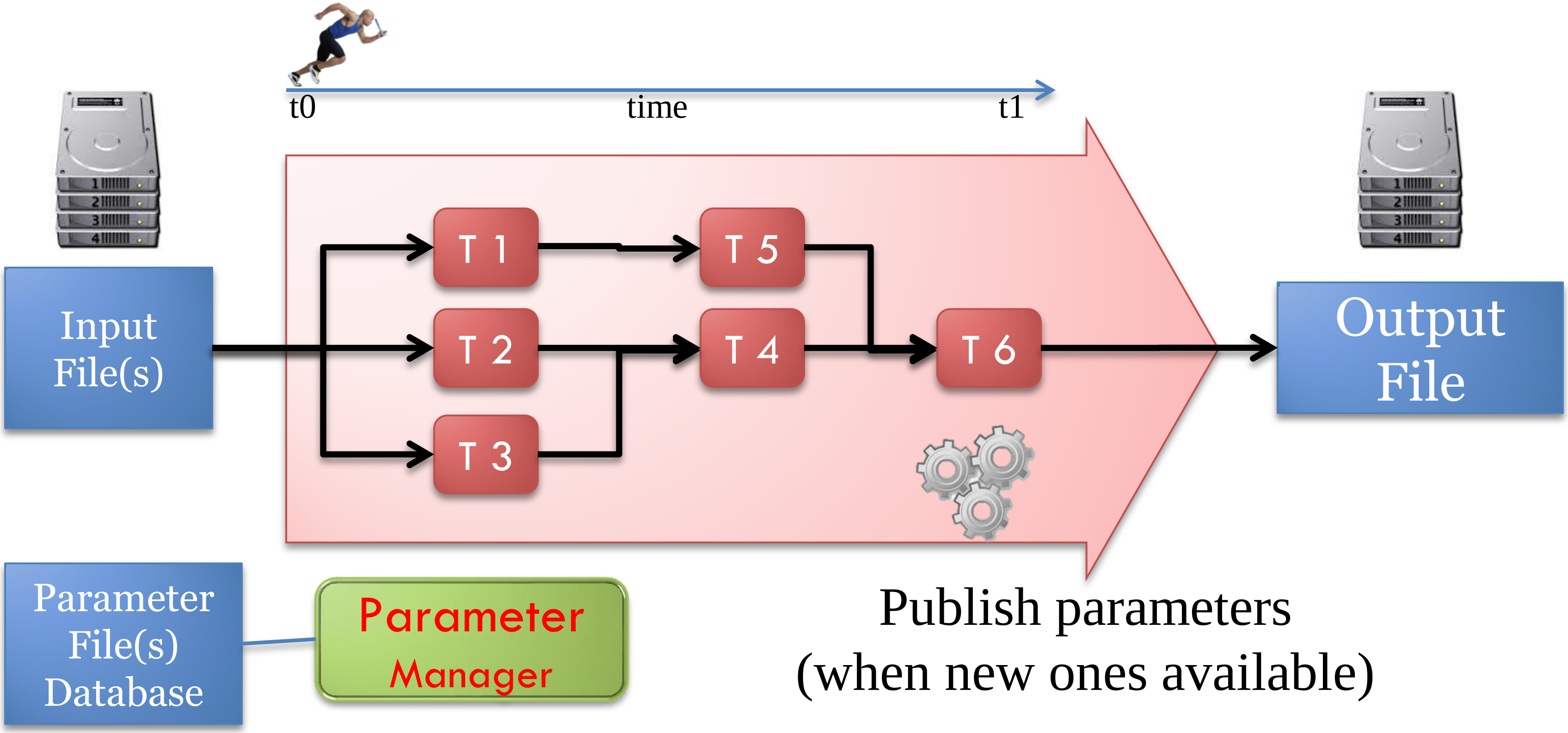
FairRoot: Where we are now?

- ROOT event loop
- User code in Task hierarchy
- Task hierarchy runs sequentially in one process
- Tasks implement only algorithms (can be exchanged/replaced)



FairRoot: Where we are going ? (almost there!)

- Each Task is a process (can be Multi-threaded)
- Message Queues for data exchange
- Support multi-core and multi node



Publish parameters
(when new ones available)

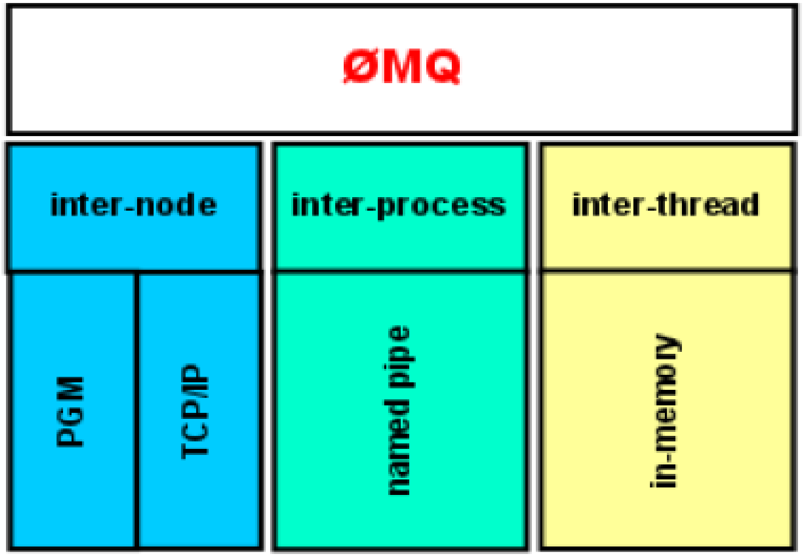
Before Re-inventing the Wheel

- What is available on the market and in the community?
  - A very promising package: ZeroMQ is available since 2011
- Do we intend to separate online and offline? NO
- Multithreaded concept or a message queue based one?
  - Message based systems allow us to decouple producers from consumers.
  - We can spread the work to be done over several processes and machines.
  - We can manage/upgrade/move around programs (processes) independently of each other.

- A messaging library, which allows you to design a complex communication system without much effort
- Abstraction on higher level than MPI (programming model is easier)
- Is suitable for loosely coupled and more general distributed systems
- Multiplatform, multi-language (+30)
- Small (20K lines of C++ code)
- Large and active open source community.
- Open source LGPL free software (large community)

ZeroMQ sockets provide efficient transport options

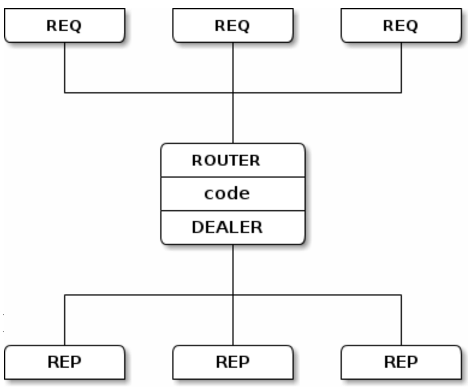
- Inter-thread
- Inter-process
- Inter-node
  - which is really just inter-process across nodes communication

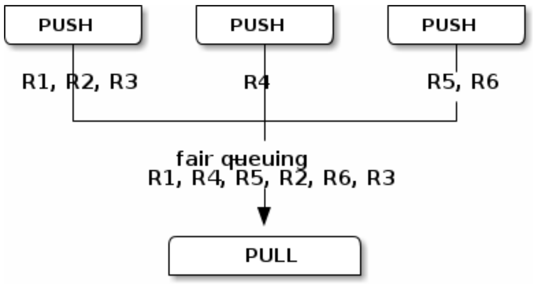


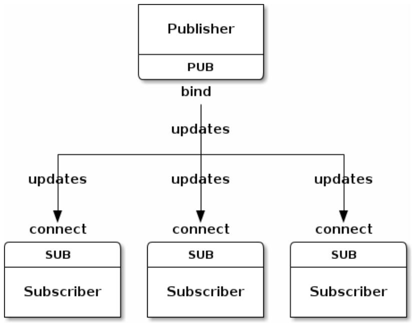
PMG : Pragmatic General Multicast (a reliable multicast protocol)
Named Pipe: Piece of random access memory (RAM) managed by the operating system and exposed to programs through a file descriptor and a named mount point in the file system. It behaves as a first in first out (FIFO) buffer

The built-in core ØMQ patterns are:

- **Request-reply**, which connects a set of clients to a set of services. (remote procedure call and task distribution pattern)
- **Publish-subscribe**, which connects a set of publishers to a set of subscribers. (data distribution pattern)
- **Pipeline**, which connects nodes in a fan-out / fan-in pattern that can have multiple steps, and loops. (Parallel task distribution and collection pattern)
- **Exclusive pair**, which connect two sockets exclusively



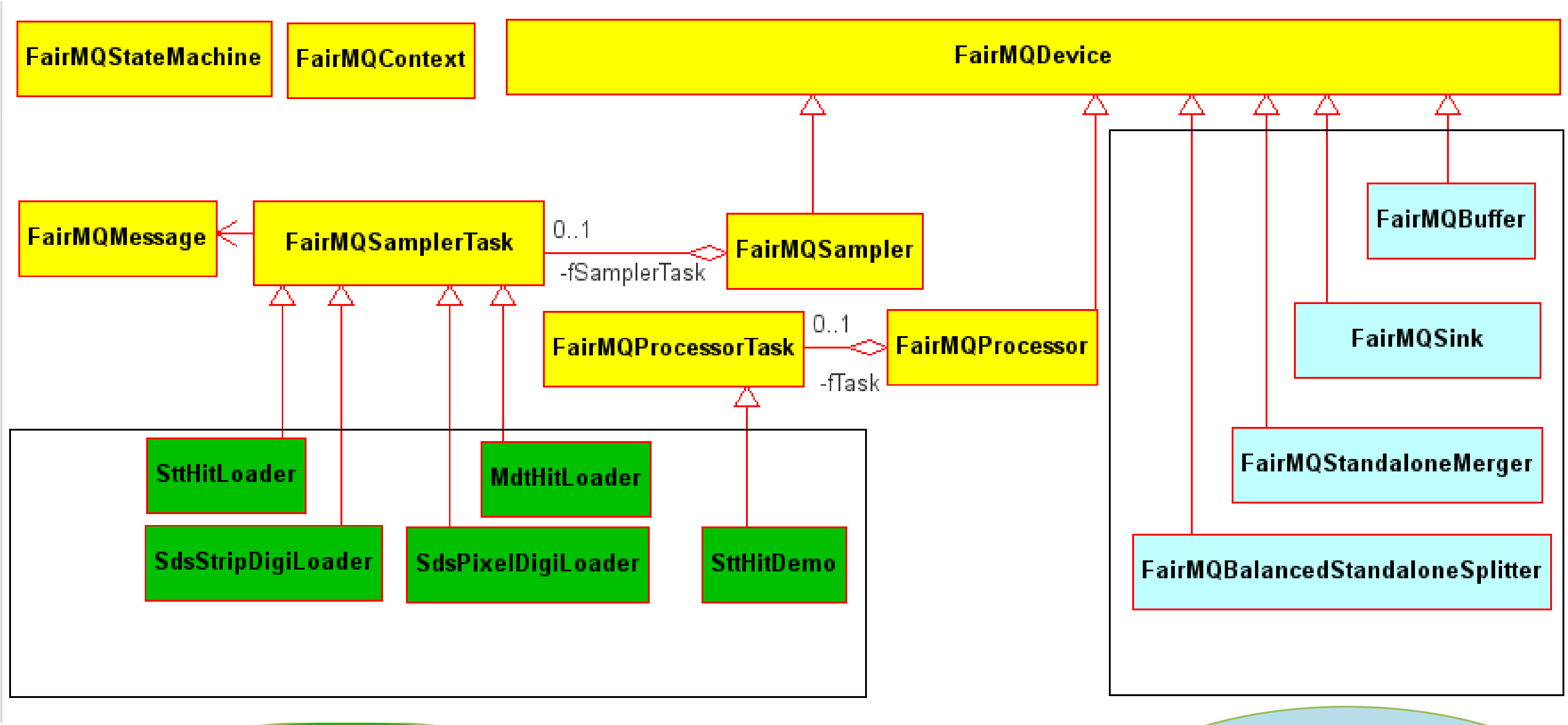




Current Status

- The Framework delivers some components which can be connected to each other in order to construct a processing pipeline(s).
- All components share a common base called Device (ZeroMQ Class).
- Devices are grouped by three categories:
  - Source:
    - Data Sampler
  - Message-based Processor:
    - Sink, Splitter, Merger, Buffer, Proxy
  - Content-based Processor:
    - Processor

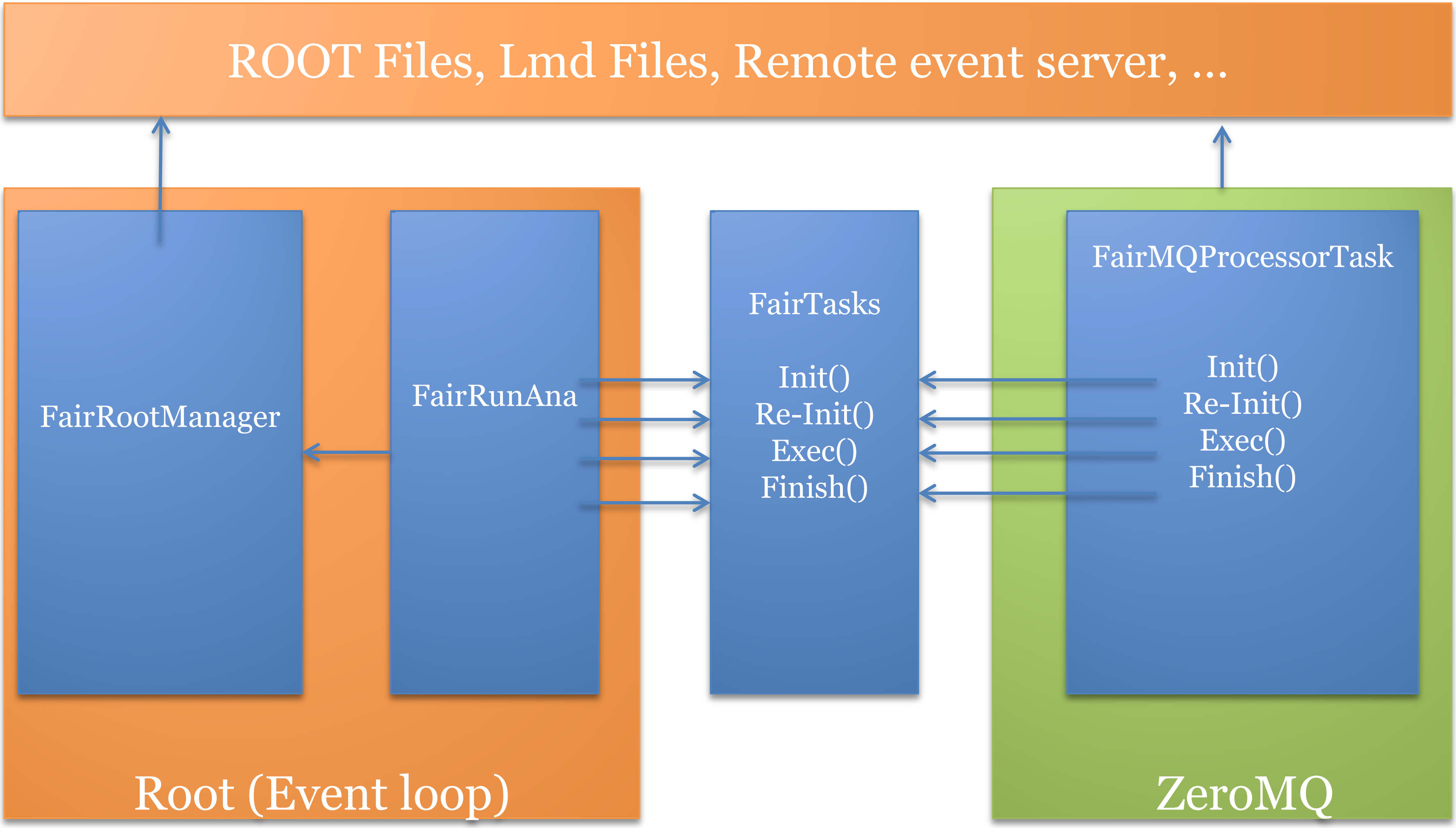
Design



Experiment/detector specific code

Framework classes that can be used directly

Integrating the existing software:



ROOT Files, Lmd Files, Remote event server, …

FairRootManager

FairRunAna

FairTasks

Init()
Re-Init()
Exec()
Finish()

FairMQProcessorTask

Init()
Re-Init()
Exec()
Finish()

Root (Event loop)

ZeroMQ

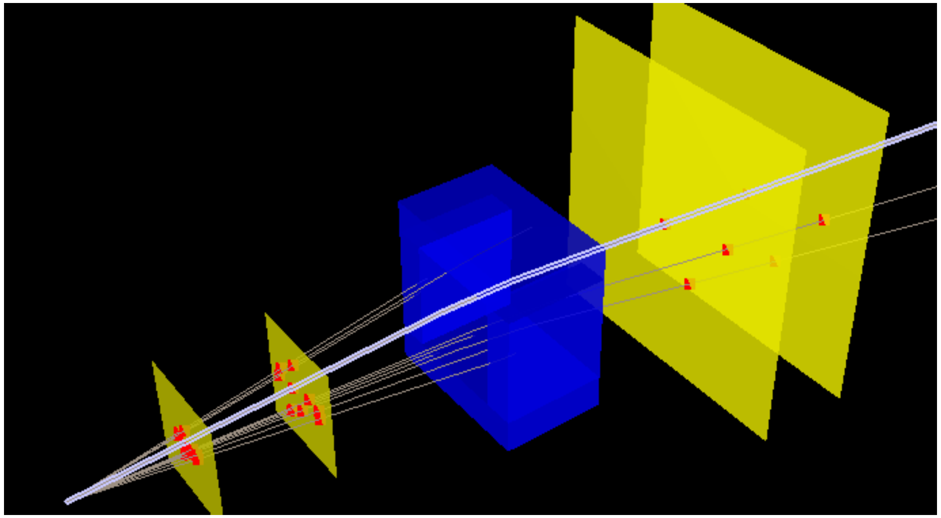
FairRoot: Example 3

4 -Tracking stations with
a dipole field

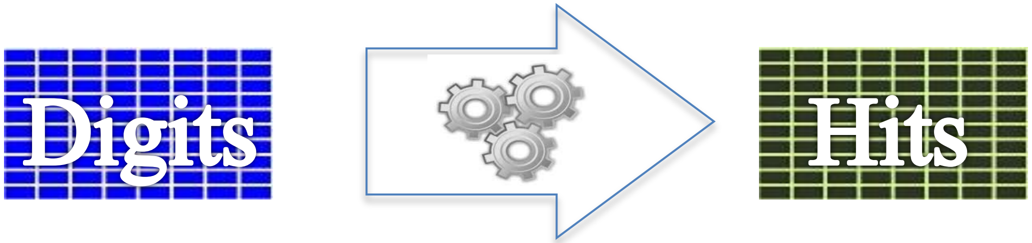
Simulation:
A) 10k event: 10 Protons/ev
B) 20k event: 300 Protons/ev

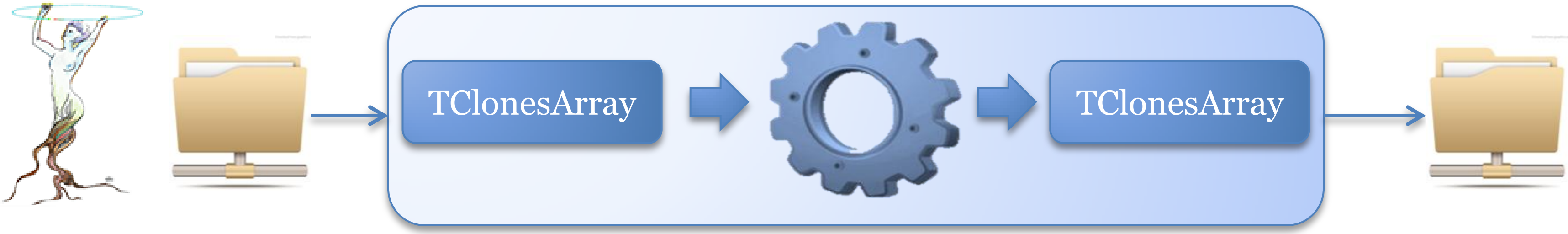
Digitization

Reconstruction:
Hit/Cluster Finder



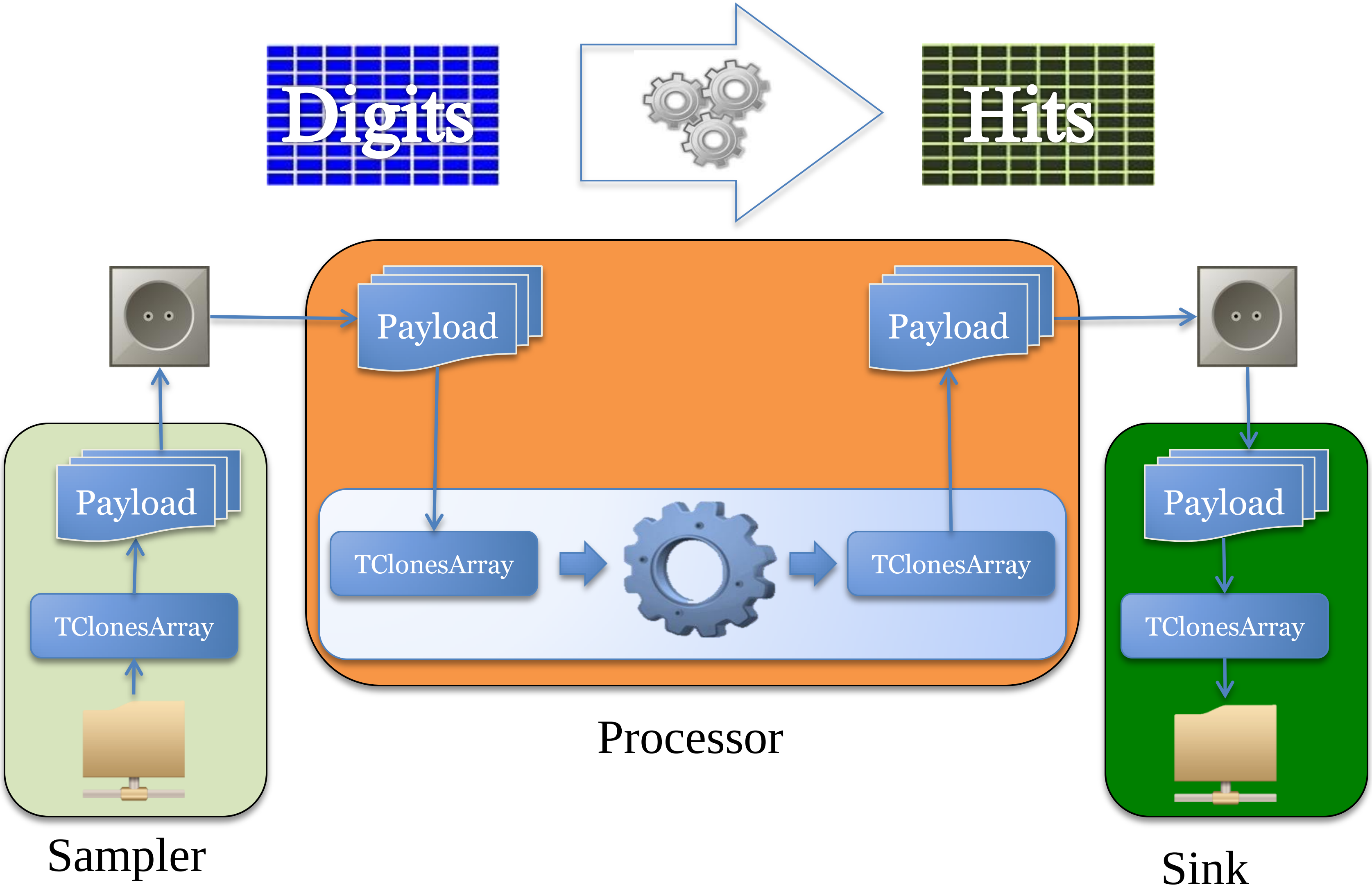
From digits to hits with ROOT:





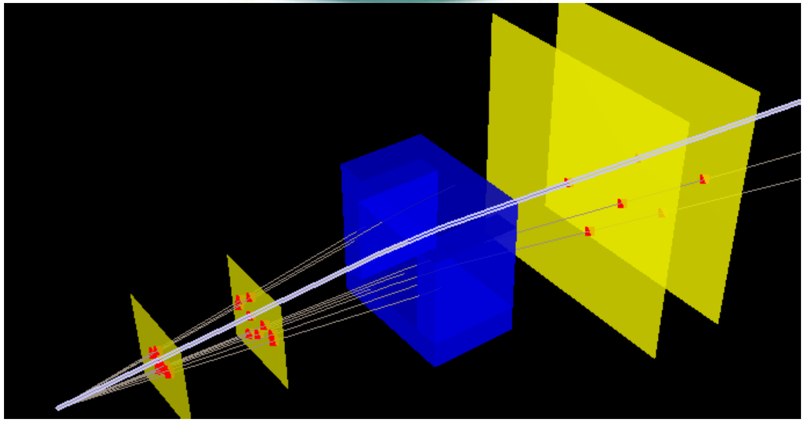
RUN	CPU Time (s) (Wall time)	Memory (Mbyte)
10k Events, 10 Protons/event	12	143
20k Events, 300 Protons/event	162	241

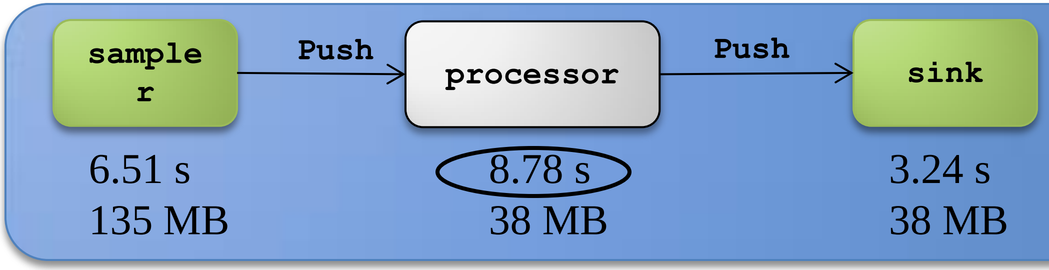
From digits to hits with ØMQ:



Test 1: Reconstruction
10k Event 10 Tracks/event

root 11.85s 143MB





sample
r

Push

processor

Push

sink

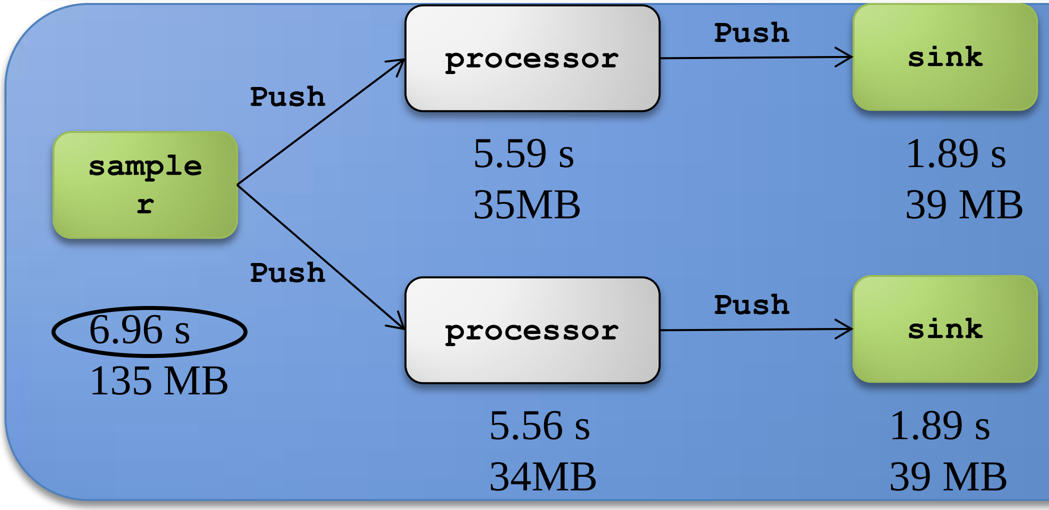
6.51 s
135 MB

8.78 s
38 MB

3.24 s
38 MB

CPU Time 25% less time

48 % more memory



processor

Push

sink

Push

sample
r

Push

5.59 s
35MB

1.89 s
39 MB

processor

Push

sink

6.96 s
135 MB

5.56 s
34MB

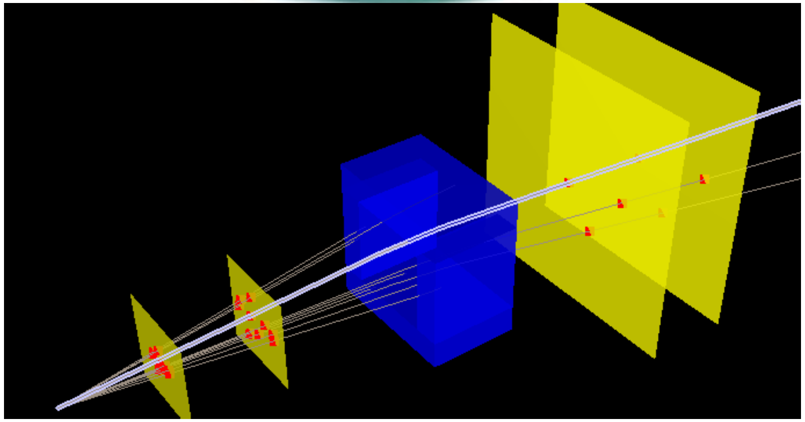
1.89 s
39 MB

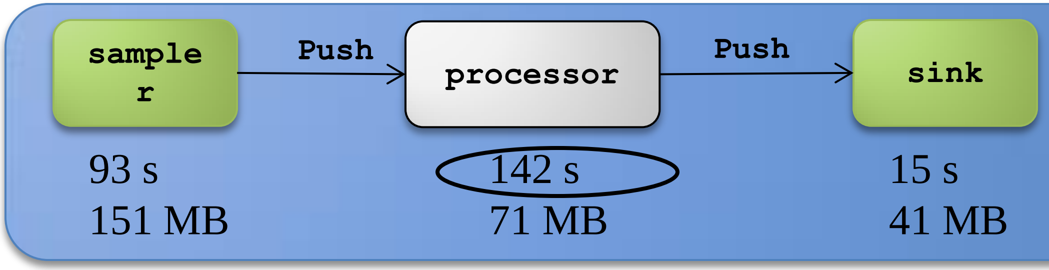
CPU Time 40% less time

97 % more memory

Test 1: Reconstruction
20k Event 300 Tracks/event

root 162 s 241MB





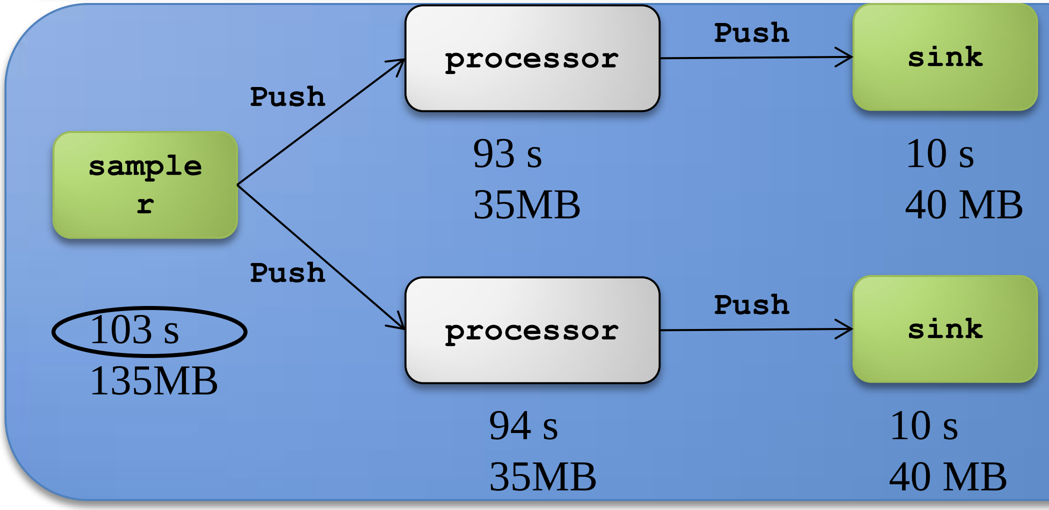
93 s
151 MB

142 s
71 MB

15 s
41 MB

CPU Time 14% less time

10 % more memory



93 s
35MB

10 s
40 MB

103 s
135MB

94 s
35MB

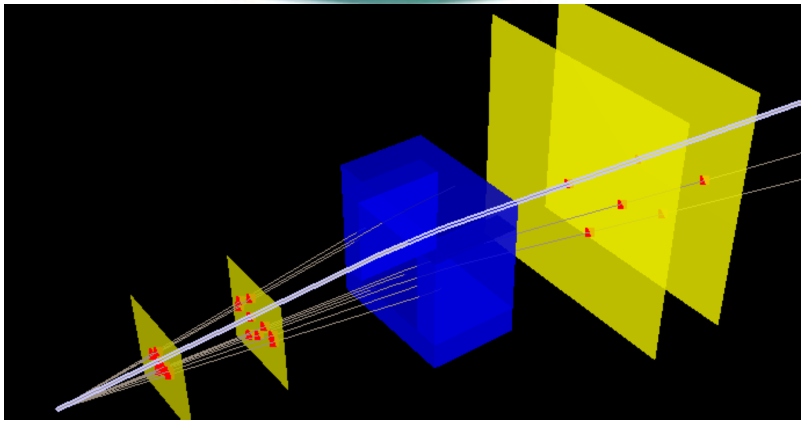
10 s
40 MB

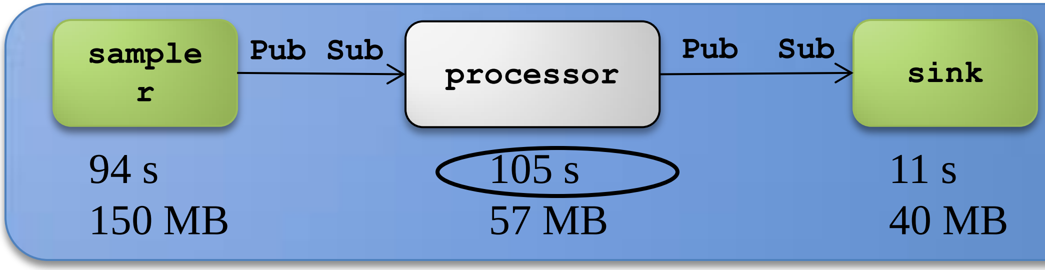
CPU Time 36 % less time

18 % more memory

Test 1: Reconstruction
20k Event 300 Tracks/event

root 162 s 241MB





sample
r

Pub Sub

processor

Pub Sub

sink

94 s
150 MB

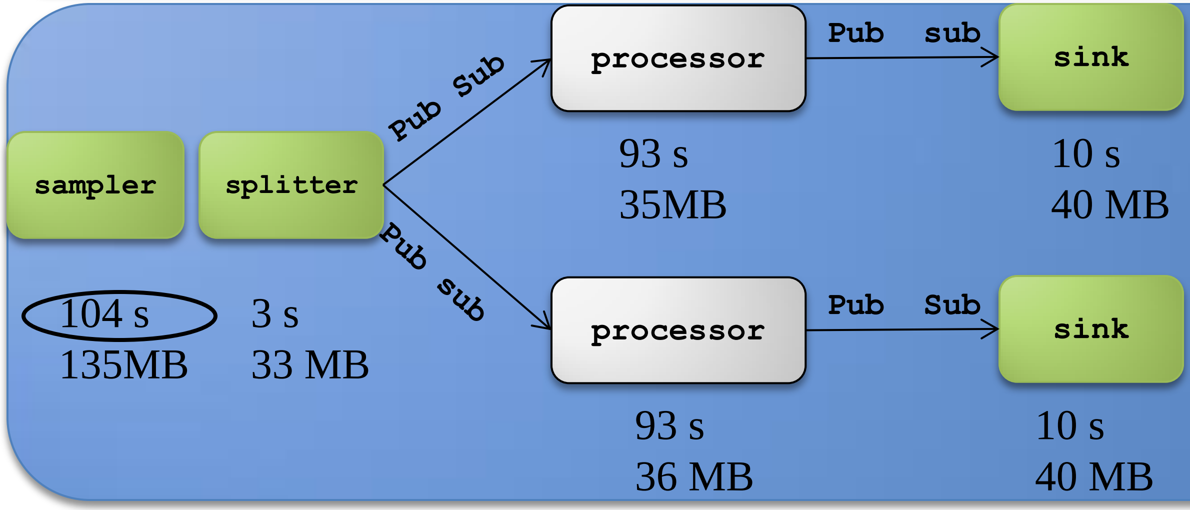
105 s
57 MB

11 s
40 MB

CPU Time 35% less time

2 % more memory

32 % events lost !



sampler

splitter

Pub Sub

processor

Pub sub

sink

93 s
35MB

10 s
40 MB

Pub sub

processor

Pub Sub

sink

104 s
135MB

3 s
33 MB

93 s
36 MB

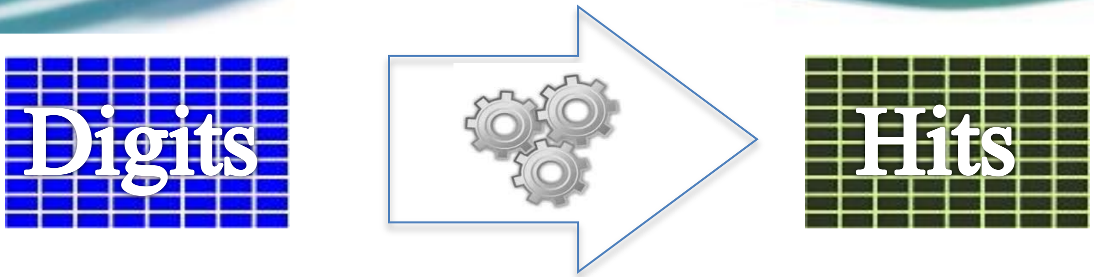
10 s
40 MB

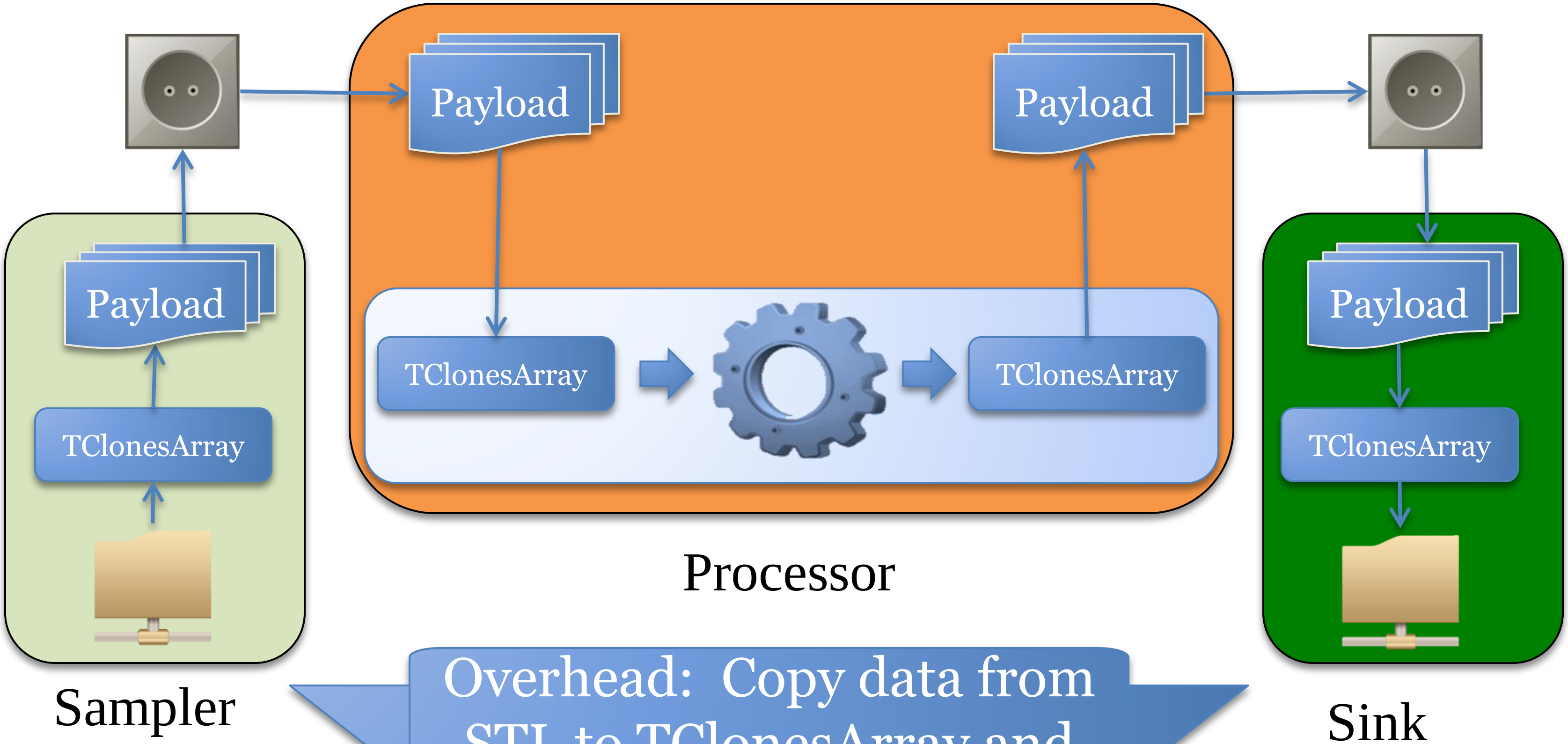
CPU Time 36 % less time

32 % more memory

0 % events lost !

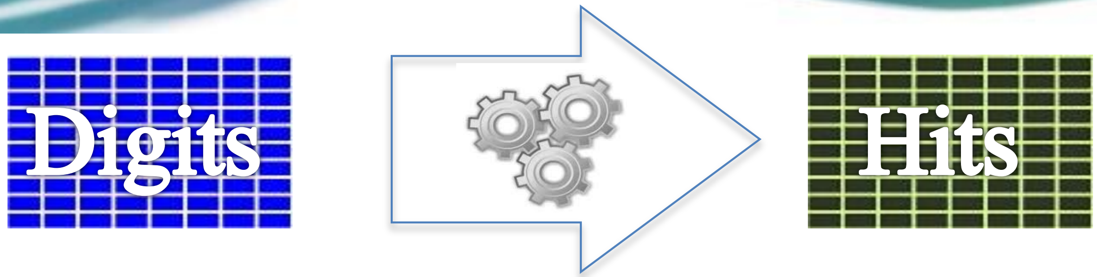
ØMQ

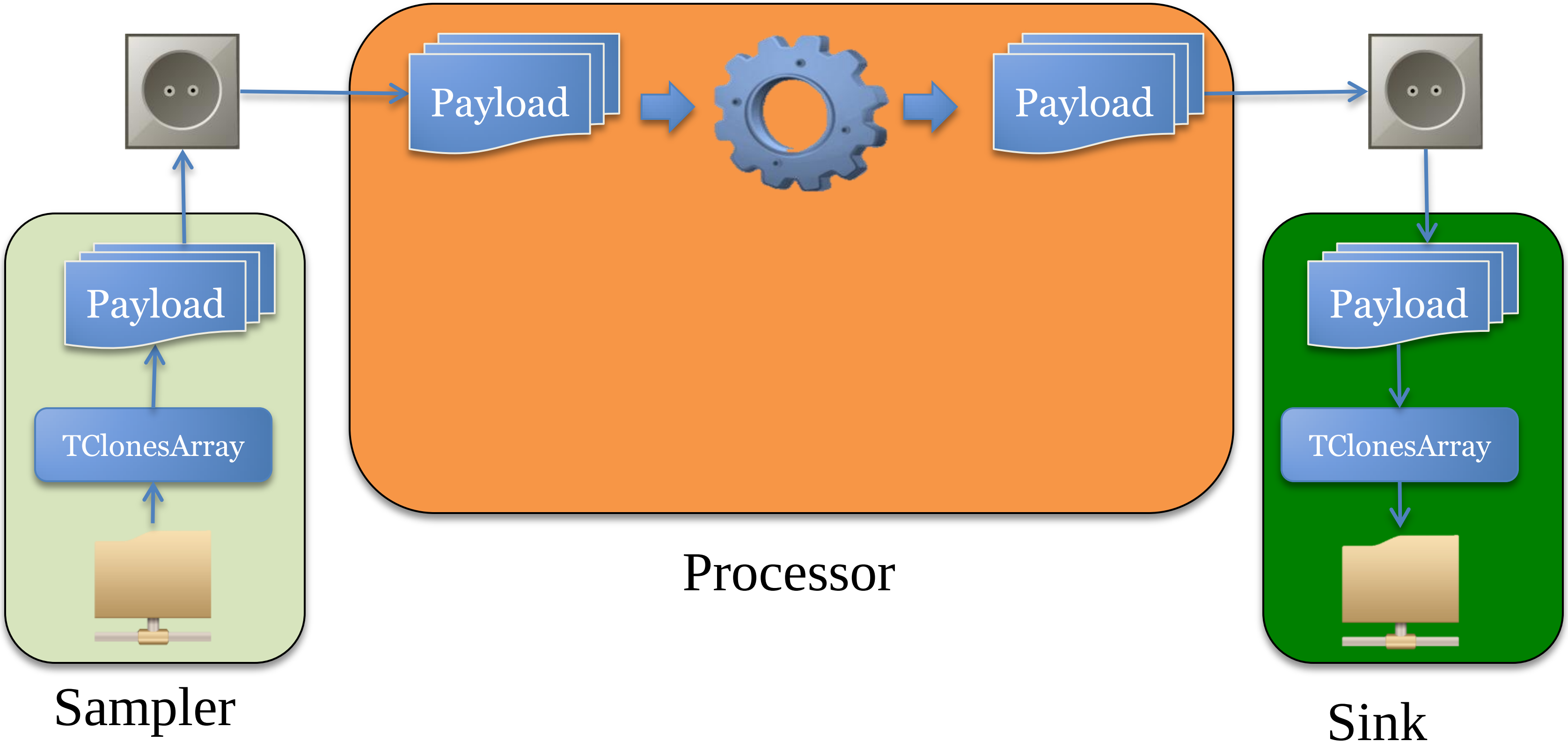




Overhead: Copy data from STL to TClonesArray and back

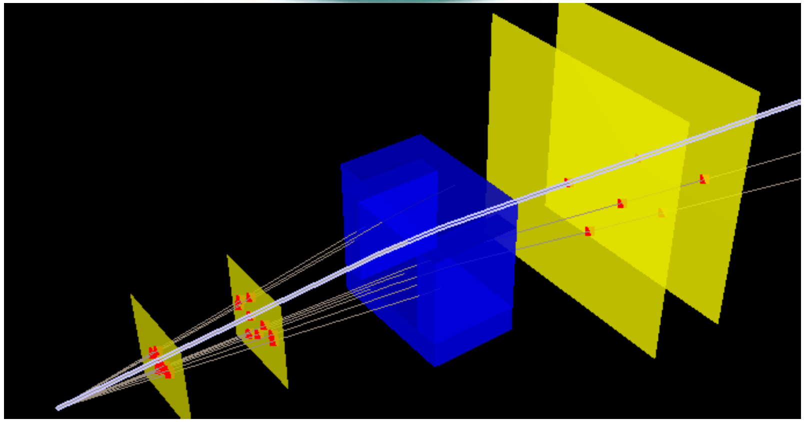
ØMQ

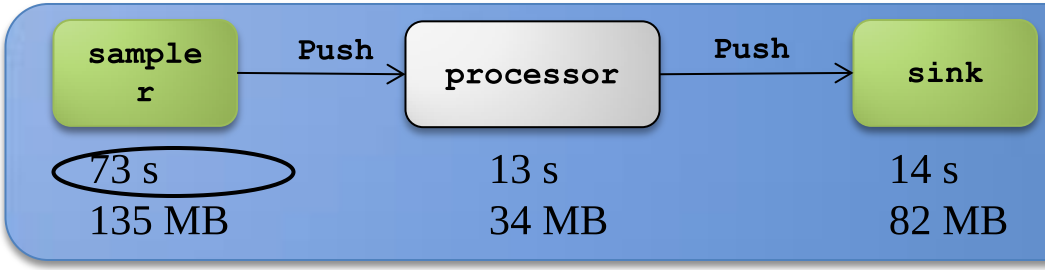

Digits
Hits


Payload
Payload
Payload
TClonesArray
Payload
TClonesArray
Processor
Sampler
Sink

Test 1: Reconstruction
20k Event 300 Tracks/event

root 162 s 241MB





sampler → Push → processor → Push → sink

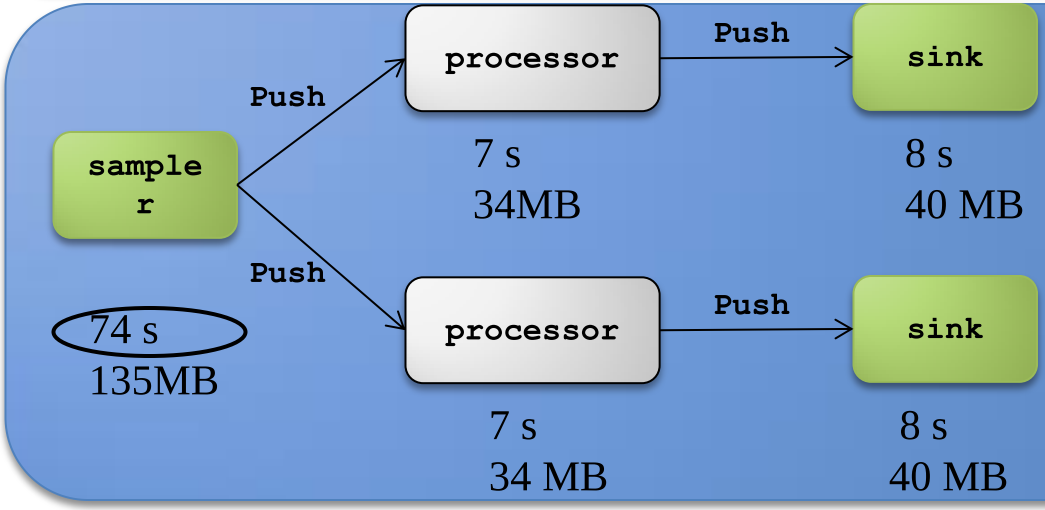
73 s
135 MB

13 s
34 MB

14 s
82 MB

CPU Time 55% less time

4 % more memory



sampler → Push → processor → Push → sink

sampler → Push → processor → Push → sink

74 s
135MB

7 s
34MB

8 s
40 MB

7 s
34 MB

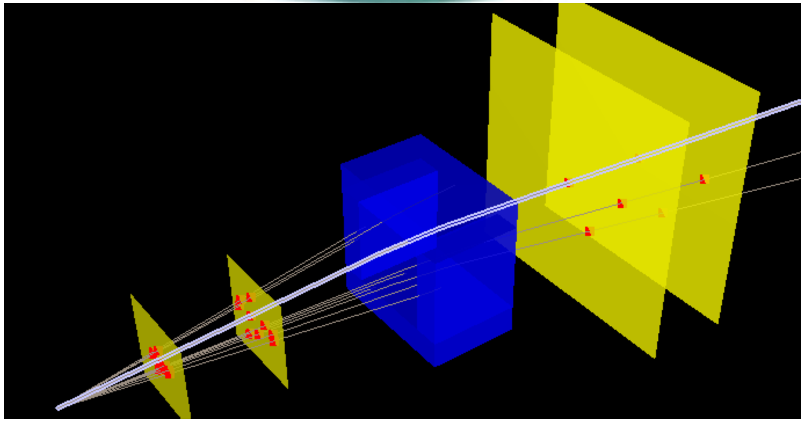
8 s
40 MB

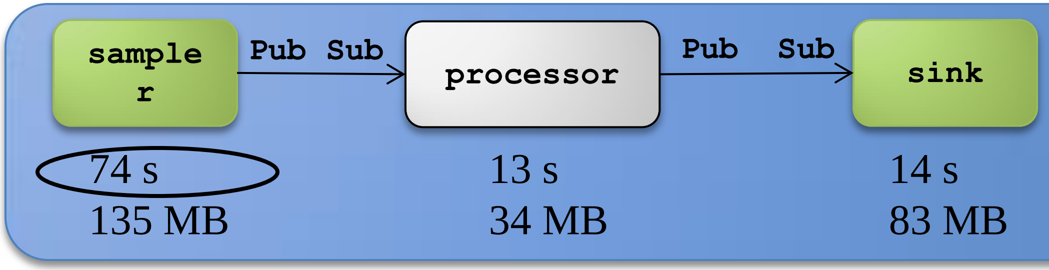
CPU Time 55 % less time

17 % more memory

Test 1: Reconstruction
20k Event 300 Tracks/event

root 162 s 241MB





sampler → Pub Sub → processor → Pub Sub → sink

sampler: 74 s, 135 MB

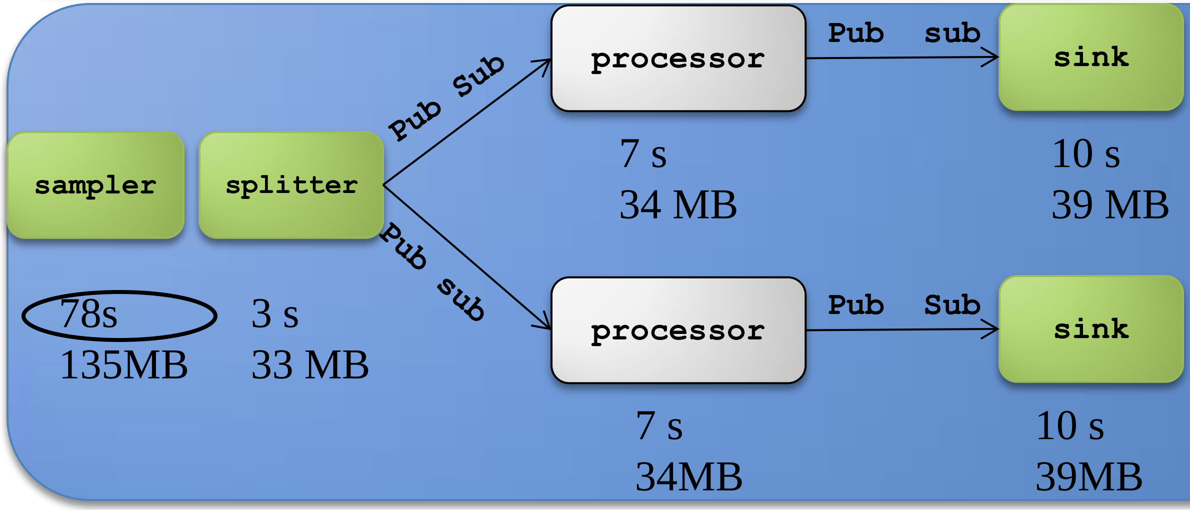
processor: 13 s, 34 MB

sink: 14 s, 83 MB

CPU Time 54% less time

2 % more memory

0 % events lost !



sampler → splitter → Pub Sub → processor → Pub sub → sink

splitter → Pub sub → processor → Pub Sub → sink

sampler: 78s, 135MB

splitter: 3 s, 33 MB

processor: 7 s, 34 MB

sink: 10 s, 39 MB

processor: 7 s, 34MB

sink: 10 s, 39MB

CPU Time 52 % less time

30 % more memory

0 % events lost !

Summary

- ZeroMQ communication layer is integrated into our offline framework (FairRoot).
- On the short term we will keep both options: ROOT based event loop and concurrent processes communicating with each other via ZeroMQ.
- On long term we are moving away from single event loop to distributed processes.

Next Step: Design and development of a dynamic deployment system (DDS)

- STORM is very attractive but no native support for C++ !
- We need to utilize any RMS (Resource Management system)
- Support different topologies and process dependencies
- Device (process) is a single entity of the system
  - Each device has its own watchdog process
  - Devices are defined by a set of props and rules,
  - All devices are statically inherited (should support) 3 interfaces: IDDSConfig, IDDSStatus, and IDDSLog
- …..

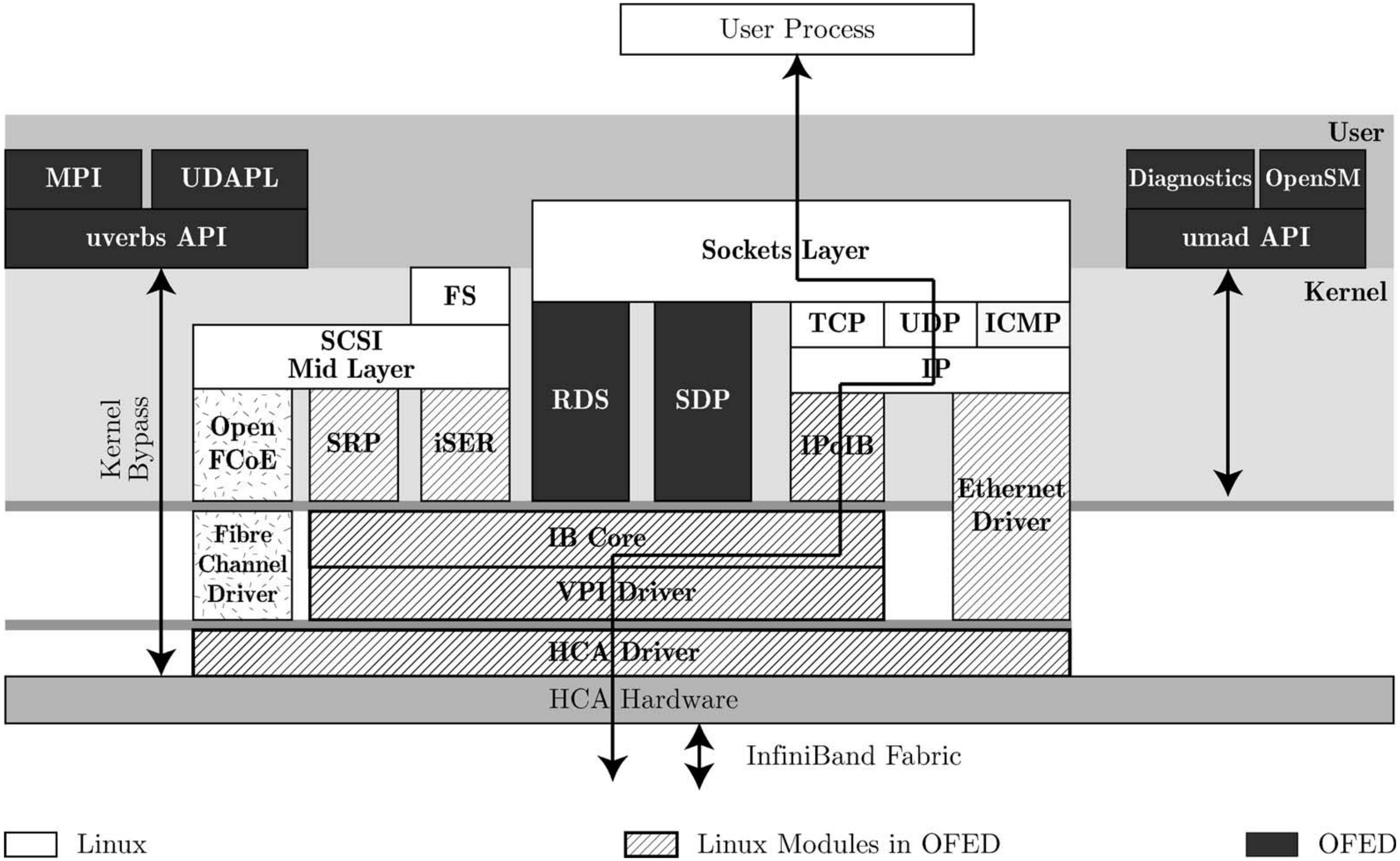
Thank you

Backup

Message format (Protocol)

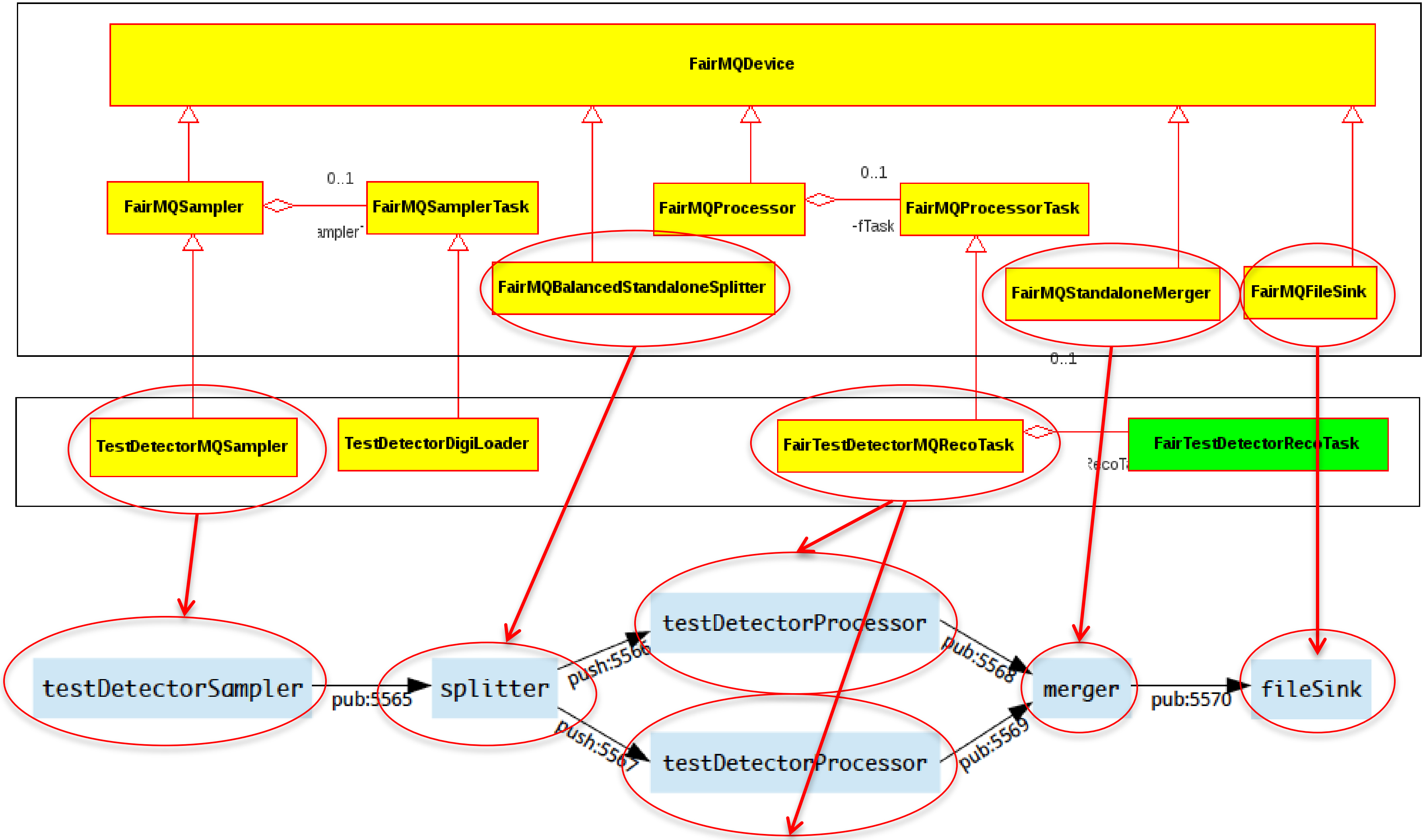
- Potentially any content-based processor or any source can change the application protocol. Therefore, the framework provides a generic Message class that works with any arbitrary and continuous junk of memory (FairMQMessage).
- One has to pass a pointer to the memory buffer, the size in bytes, and can optionally pass a function pointer to a destructor, which will be called once the message object is discarded.

Native InfiniBand/RDMA is faster than IP over IB

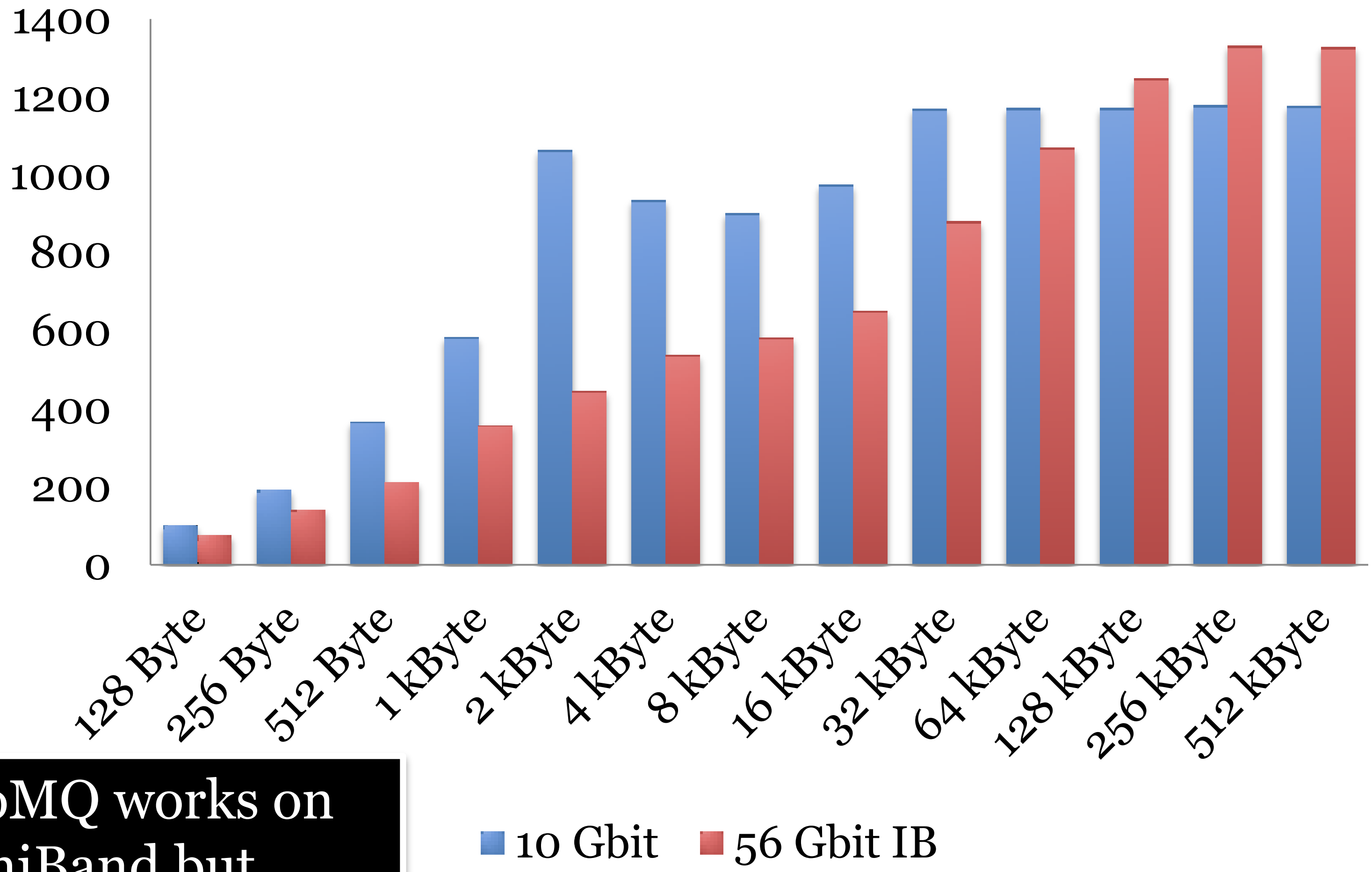


Implementing ZeroMQ over IB verbs will improve the performance.

Fairbase/example/Tutorial3



Payload in Mbyte/s as function of message size



ZeroMQ works on InfiniBand but using IP over IB

Results

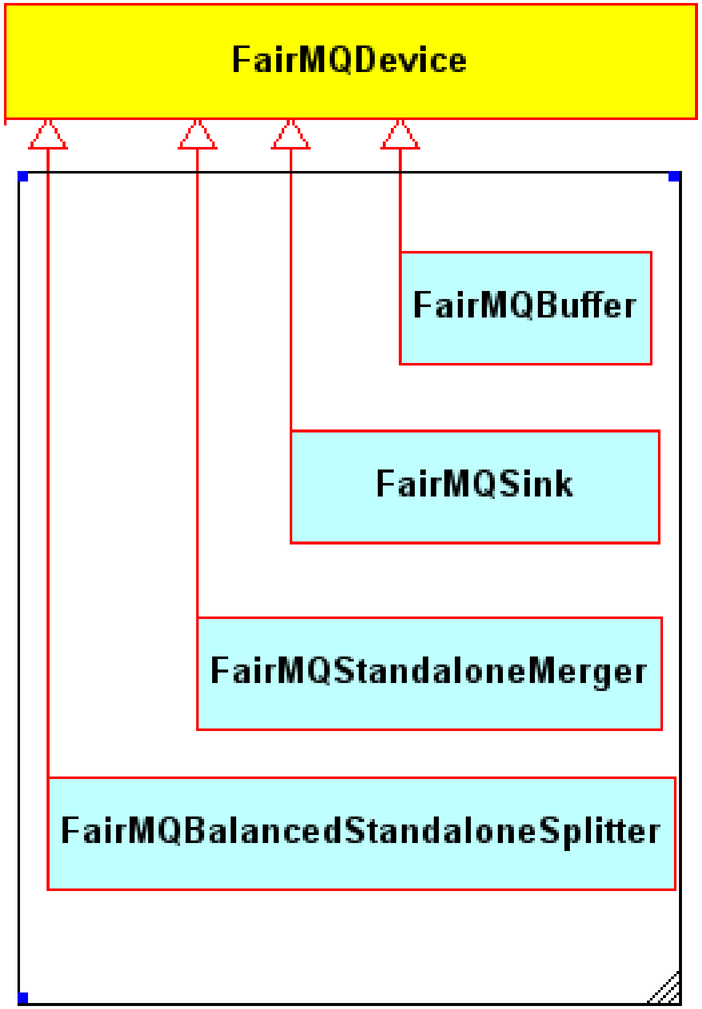
- Throughput of 940 Mbit/s was measured which is very close to the theoretical limit of the TCP/IPv4/GigabitEthernet

- The throughput for the named pipe transport between two devices on one node has been measured around 1.7 GB/s

Each message consists of digits in one event for one detector, with size of few kBytes

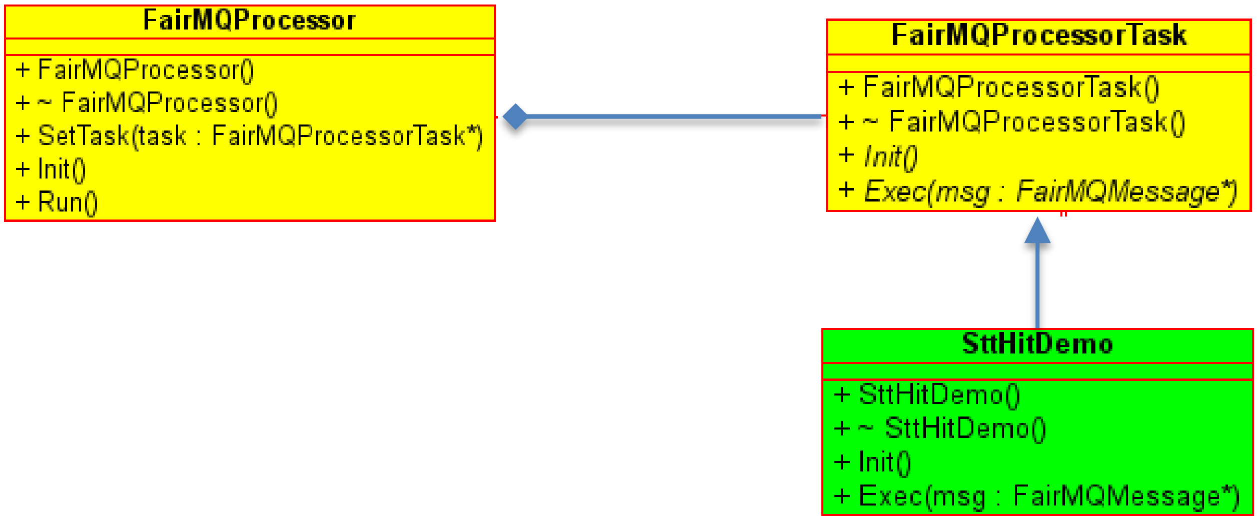
Message-based Processor

- All message-based processors inherit from Device and operate on messages without interpreting their content.
- Four message-based processors have been implemented so far



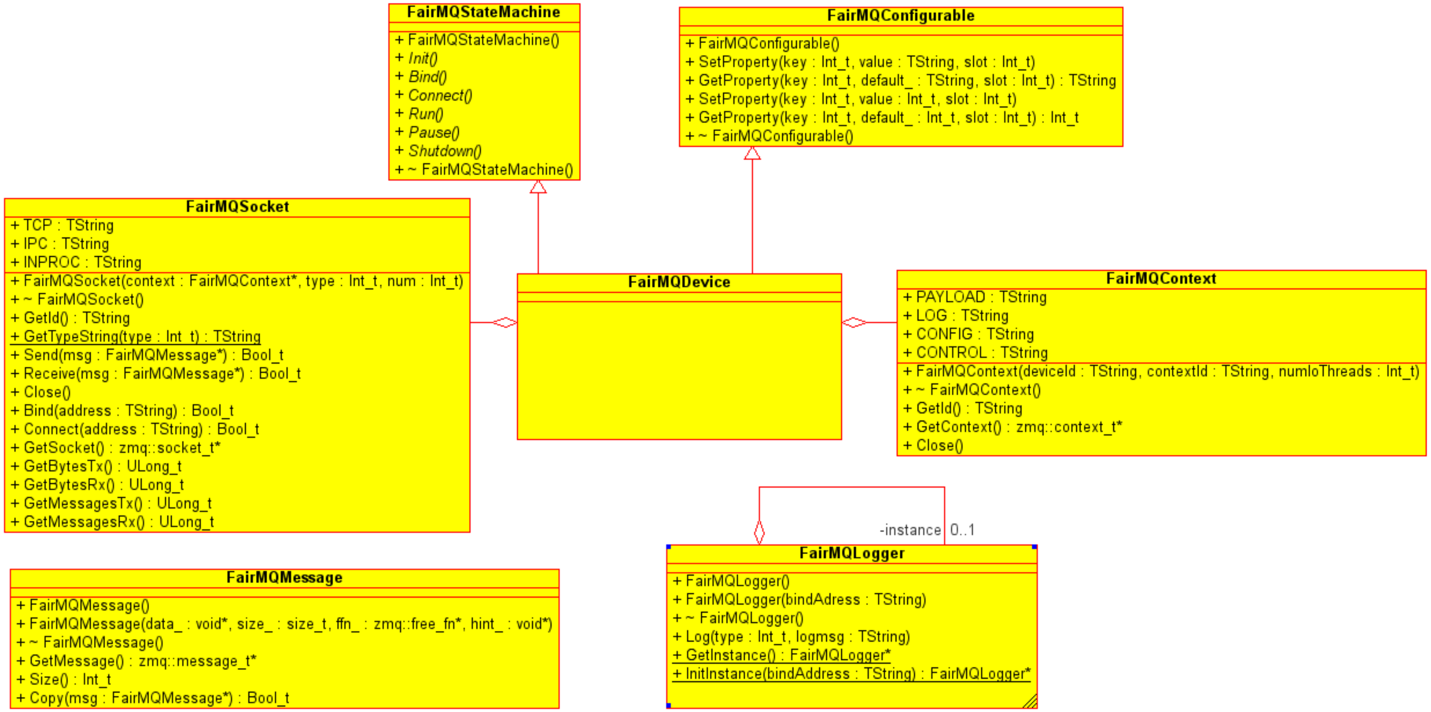
Content-based Processor

- The Processor device has at least one input and one output socket.
- A task is meant for accessing and potentially changing the message content.



Device

- Each processing stage of a pipeline is occupied by a process which executes an instance of the Device class



FairMQStateMachine

+ FairMQStateMachine()
+ *Init()*
+ *Bind()*
+ *Connect()*
+ *Run()*
+ *Pause()*
+ *Shutdown()*
+ ~ FairMQStateMachine()

FairMQConfigurable

+ FairMQConfigurable()
+ SetProperty(key : Int_t, value : TString, slot : Int_t)
+ GetProperty(key : Int_t, default_ : TString, slot : Int_t) : TString
+ SetProperty(key : Int_t, value : Int_t, slot : Int_t)
+ GetProperty(key : Int_t, default_ : Int_t, slot : Int_t) : Int_t
+ ~ FairMQConfigurable()

FairMQSocket

+ TCP : TString
+ IPC : TString
+ INPROC : TString
+ FairMQSocket(context : FairMQContext*, type : Int_t, num : Int_t)
+ ~ FairMQSocket()
+ GetId() : TString
+ GetTypeString(type : Int_t) : TString
+ Send(msg : FairMQMessage*) : Bool_t
+ Receive(msg : FairMQMessage*) : Bool_t
+ Close()
+ Bind(address : TString) : Bool_t
+ Connect(address : TString) : Bool_t
+ GetSocket() : zmq::socket_t*
+ GetBytesTx() : ULong_t
+ GetBytesRx() : ULong_t
+ GetMessagesTx() : ULong_t
+ GetMessagesRx() : ULong_t

FairMQDevice

FairMQContext

+ PAYLOAD : TString
+ LOG : TString
+ CONFIG : TString
+ CONTROL : TString
+ FairMQContext(deviceId : TString, contextId : TString, numIoThreads : Int_t)
+ ~ FairMQContext()
+ GetId() : TString
+ GetContext() : zmq::context_t*
+ Close()

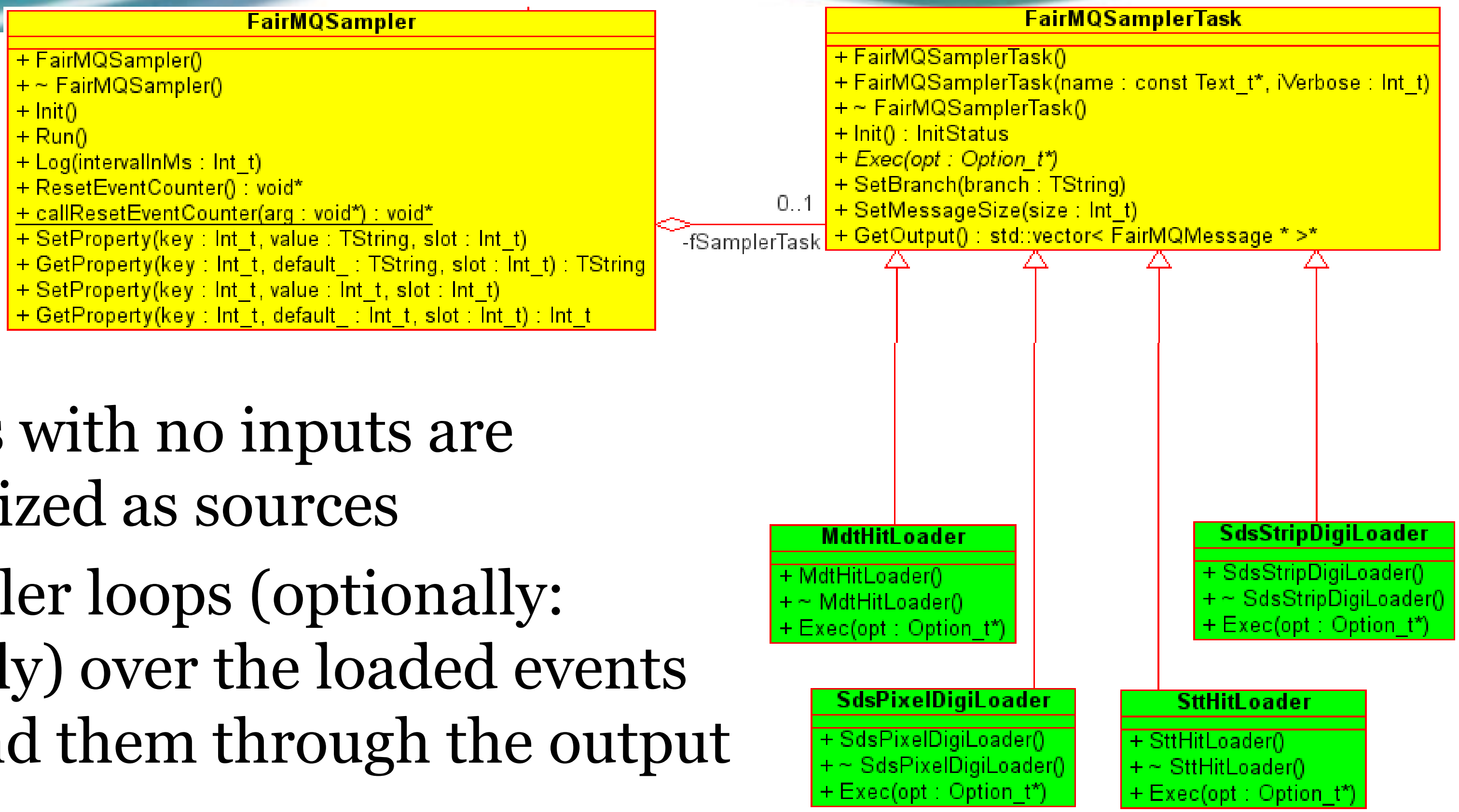
FairMQMessage

+ FairMQMessage()
+ FairMQMessage(data_ : void*, size_ : size_t, ffn_ : zmq::free_fn*, hint_ : void*)
+ ~ FairMQMessage()
+ GetMessage() : zmq::message_t*
+ Size() : Int_t
+ Copy(msg : FairMQMessage*) : Bool_t

FairMQLogger

+ FairMQLogger()
+ FairMQLogger(bindAdress : TString)
+ ~ FairMQLogger()
+ Log(type : Int_t, logmsg : TString)
+ GetInstance() : FairMQLogger*
+ InitInstance(bindAddress : TString) : FairMQLogger*

Sampler



FairMQSampler

+ FairMQSampler()
+ ~ FairMQSampler()
+ Init()
+ Run()
+ Log(intervalInMs : Int_t)
+ ResetEventCounter() : void*
+ callResetEventCounter(arg : void*) : void*
+ SetProperty(key : Int_t, value : TString, slot : Int_t)
+ GetProperty(key : Int_t, default_ : TString, slot : Int_t) : TString
+ SetProperty(key : Int_t, value : Int_t, slot : Int_t)
+ GetProperty(key : Int_t, default_ : Int_t, slot : Int_t) : Int_t

0..1

-fSamplerTask

FairMQSamplerTask

+ FairMQSamplerTask()
+ FairMQSamplerTask(name : const Text_t*, iVerbose : Int_t)
+ ~ FairMQSamplerTask()
+ Init() : InitStatus
+ *Exec(opt : Option_t*)*
+ SetBranch(branch : TString)
+ SetMessageSize(size : Int_t)
+ GetOutput() : std::vector< FairMQMessage * >*

MdtHitLoader

+ MdtHitLoader()
+ ~ MdtHitLoader()
+ Exec(opt : Option_t*)

SdsStripDigiLoader

+ SdsStripDigiLoader()
+ ~ SdsStripDigiLoader()
+ Exec(opt : Option_t*)

SdsPixelDigiLoader

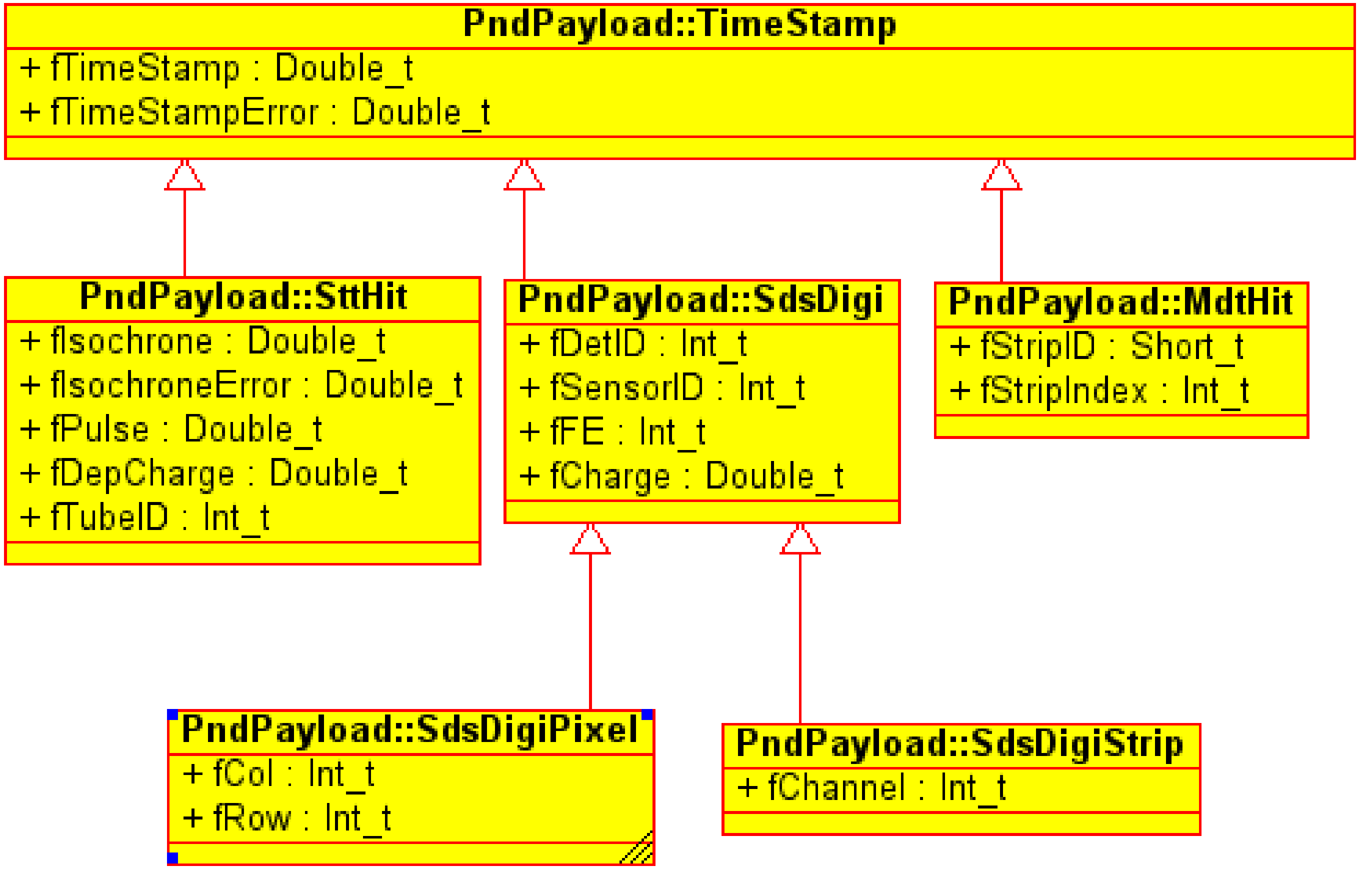
+ SdsPixelDigiLoader()
+ ~ SdsPixelDigiLoader()
+ Exec(opt : Option_t*)

SttHitLoader

+ SttHitLoader()
+ ~ SttHitLoader()
+ Exec(opt : Option_t*)

- Devices with no inputs are categorized as sources
- A sampler loops (optionally: infinitely) over the loaded events and send them through the output socket.
- A variable event rate limiter has been implemented to control the sending speed

New simple classes without ROOT are used in the Sampler (This enable us to use non-ROOT clients) and reduce the messages size.



PndPayload::TimeStamp

+ fTimeStamp : Double_t
+ fTimeStampError : Double_t

PndPayload::SttHit

+ fIsochrone : Double_t
+ fIsochroneError : Double_t
+ fPulse : Double_t
+ fDepCharge : Double_t
+ fTubeID : Int_t

PndPayload::SdsDigi

+ fDetID : Int_t
+ fSensorID : Int_t
+ fFE : Int_t
+ fCharge : Double_t

PndPayload::MdtHit

+ fStripID : Short_t
+ fStripIndex : Int_t

PndPayload::SdsDigiPixel

+ fCol : Int_t
+ fRow : Int_t

PndPayload::SdsDigiStrip

+ fChannel : Int_t

ØMQ Features

- Message blobs of Zero to N bytes
- One socket connect to many sockets
- Queuing sender and receiver
- Automatic TCP (re)connect
- Zero-copy for large messages