Trigger

M. de Jong

20/4/2020

“All-data-to-shore”

a) all analogue pulses from all PMTs that pass
a pre-set threshold are timestamped off-shore;

b) all timestamped data are sent to shore;

c) all data are calibrated and filtered on shore
in real time using a farm of commodity PCs;

d) all filtered and some summary data are distributed
and saved on persistent media for further analyses;

Applications (1/3)

- JDataFilter
  - real time filtering of detector data (not this talk)
- JRandomTimesliceWriter
  - produce time slices with random data
- JTriggerProcessor
  - offline filtering of detector data
- JTriggerReprocessor
  - reprocess triggered data

Applications (2/3)

- JTriggerEfficiency
  - processing of Monte Carlo data
    - output of km3, KM3Sim, JSirene, …
    - addition of random background
- JEventTimesliceWriter
  - processing of Monte Carlo data
    - output of K40 simulations
    - addition random background (e.g. singles rates)

Applications (3/3)

- JSummary
  - convert real summary data to summary data suitable for simulations
    - output of ARCA or ORCA detector
- JTriggerMonitor
  - print trigger statistics
- JTrigger
  - provides statistical information about the signal processing during detector simulation of JTriggerEfficiency
    - possible explanation why there are no triggered events

JRandomTimesliceWriter

- command line options

-a <detector file>

-B "R_1 [R_2 [R_3 [...]]]" // background

-o <output file> // time slice data

-P "%.QE=<value>; // PMT simulation

wild card pmt=<module> <address> QE=<value>; "

-P <PMT simulation file>

JTriggerProcessor

- command line options

  -a <detector file>

  -f <input file> // time slice data (see below)

  -@ "<trigger parameter>=<value>; ..."

  -@ "<trigger parameter file>"

JTriggerEfficiency (1/2)

- command line options

  -a <detector file>

  -f <Monte Carlo event file>

  -@ "<trigger parameter>=<value>; ..."

  -@ "<trigger parameter file>"

  -P "%.QE=<value>; // PMT simulation
  pmt=<module> <address> QE=<value>;"

  -P <PMT simulation file>

  -S <seed>

  -O <triggeredEventsOnly>

JTriggerEfficiency (2/2)

- command line options

  -B "R_1 [R_2 [R_3 [R_4]]]" // background rates [Hz]

  -r "file=<file name>" // run-by-run simulation

  -r <run-by-run simulation file>

  N.B.
  singles rates from run-by-run data (option -r) prevails
  but multiples rates are taken from option -B

JSummary

- command line options

  -f <raw data file> // real data

  -a <detector file>// for simulations

  -n <number of slices>

  -o <output file>

JEventTimesliceWriter

- command line options
  - -f <K40 event file> // coincidence data
  - -o <output file> // time slice data
  - -r <event rate [Hz]>
  - -B "R_1 [R_2 [R_3 [R_4]]]" // background rates [Hz]
  - -P "%.QE=<value>; // PMT simulation
    pmt=<module> <address> QE=<value>;"
  - -P <PMT simulation file>

Detector (1/1)

- Data structures
  - JDetector : std::vector<JModule> {};
  - JModule : std::vector<JPMT> {};
  - JPMT : JObjectID, JAxis3D, JCalibration {};
- JModuleRouter
  - O(1) access to module data

    JModule getModule(<module identifier>);
- JPMTRouter
  - O(1) access to PMT data

    JPMT getPMT(<PMT identifier>);
- JDAQHitRouter
  - O(1) access to PMT data

    JPMT getPMT(JDAQKeyHit&);

Data format (1/4)

```
class JDAQHit
{
    getPMT();    // get PMT (0, …, 30)
    getT();   // get time [ns]
    getToT();      // get time-over-threshold [ns]
};
```

```
class JDAQFrame            // low-level data frame
{                   // one per optical module
    const_iterator   begin();
    const_iterator   end();
    bool      empty();
    int   size();
    const JDAQHit& operator[](int);
};
```

mimics std::vector<>

Data format (2/4)

```
class JDAQSuperFrame : // high-level data frame
                       // one per optical module
    ..
    JDAQSuperFrameHeader,     // run number, etc.
    JDAQFrame        // PMT data
{};
```

```
class JDAQTimeslice :      // all data there are
    ..
    JDAQTimesliceHeader,    // run number, etc.
    vector<JDAQSuperFrame>    // PMT data
{};
```

Data format (3/4)

```
class JDAQSummaryFrame : // summary information
        ..     // one per optical module
        JDAQModuleIdentifier,
        JDAQFrameStatus
{
        bool testDAQStatus();     // UDP packet test
        bool testWhiteRabbitStatus();     // White Rabbit test
        bool testHighRateVeto();       // high-rate veto (do this first)
        bool testHighRateVeto(int tdc);   // high-rate veto

        double getRate(int tdc); // rate [Hz]
};
```

Data format (4/4)

```
class JDAQSummaryslice :          // all summary information  ..            // there is
     JDAQSummarysliceHeader,
     vector<JDAQSummaryFrame>
{
          JDAQSummaryslice( JDAQTimeslice);// build summaries
};
```

Data processing (1/6)

- global methods

```
double getTime(    t [ns],   JCalibration)    {}
double putTime(    t [ns],   JCalibration)    {}
```

sign convention

- auxiliary methods

```
double getTime(JDAQHit,  JCalibration)    {}
```

Data processing (2/6)

- JDAQFrame
  - contains all data from one optical module within pre-set time window ($\equiv$ frame time)

$$\{\cdots,(\text{JDAQHit})_i,(\text{JDAQHit})_{i+1},\cdots\};$$

- specification

$$\forall\ (i,j)\ \text{ where }\ i<j\ \wedge\ \text{JDAQHit::getPMT()}_i=\text{JDAQHit::getPMT()}_j$$

$$\Longrightarrow\ \text{JDAQHit::getT()}_i<\text{JDAQHit::getT()}_j$$

Data processing (3/6)

- first step
  I. convert 1-dimensional mixed array of DAQ hits to collection of time calibrated and time sorted 1-dimensional arrays per PMT

- additional steps
  II. merge collection of 1-dimensional arrays to single time sorted 1-dimensional array
  III. apply coincidence logic

Data processing (4/6)

- first step (2)

```
class JSuperFrame2D :
	JModuleHeader,
	vector<JFrame> // time calibrated data
{
	JSuperFrame1D& operator()(	JDAQSuperFrame&,
	JModule&); I/O operator

	static JSuperFrame2D<>	demultiplex;  // Demultiplexer
};
```

Data processing (5/6)

- additional steps (2)

```
class JSuperFrame1D :
      JModuleHeader,
      vector<JElement_t> // time calibrated data
{
      JSuperFrame1D& operator()(    JSuperFrame2D&);
                  I/O operator

      static JSuperFrame1D<>       multiplex;  // Multiplexer
};
```

Data processing (6/6)

- additional steps (2)
  - JBuildL0<>
    - convert raw data to JHitL0 data
      - JHitL0 : JDAQPMTIdentifier, JAxis3D, JHit {};
  - JBuildL1<>
    - convert raw data to compressed L1 data
    - convert raw data to JHitL1 data
      - JHitL1 : vector<JHitL0> {};
  - JBuildL2<>
    - select subset of L1 data
      - multiplicity of L1
      - maximal angle between PMT axes

Hit clustering (1/2)

- clusterize(.., JMatch&)
  I. count number of friends;
  II. remove element with least number of friends;
  III. stop when least number of friends = number of elements;
- reverse_clusterize(.., JMatch&)
  I. count number of friends;
  II. keep element with most number of friends;
  III. stop when most number of friends = number of elements;
- clusteriseWeight(.., JMatch&)
  - idem using weight of elements

Hit clustering (2/2)

- JMatch3D
  - simple causality
- JMatch3G
  - causality for shower with distance dependent time window (improvement due to J. Hofestädt)
- JMatch3B
  - causality for muon with distance dependent time window (improvement due to B. Bakker)
- JMatch1D
  - causality for muon along z-axis

Accidental coincidence rate (1/2)

number of sources

rate of each source

minimum cluster size

$$R(m) = \binom{N}{m} \times f \times (f \times \Delta T)^{m-1}$$

$$= \frac{N^m}{m!} \times f \times (f \times \Delta T)^{m-1}$$

$$= \frac{1}{m!} \times \underbrace{Nf}_{\text{Total rate}} \times \underbrace{(Nf \times \Delta T)^{m-1}}_{\text{Probability}}$$

Accidental coincidence rate (2/2)

$$R(m) = \frac{1}{m!} \times Nf \times (Nf \times \Delta T)^{m-1}$$

3D

$$\Delta T = \frac{nD}{c} \cong 5\,\mu s$$

$$N_{L1} = 115 \times 18 = 2070$$

$$f_{L1} \cong 1\,kHz$$

1D

$$\Delta T = \frac{nR}{c} \cong 0.5\,\mu s$$

$$N_{L1} \cong 50$$

$$f_{L1} \cong 1\,kHz$$

Trigger logic (1/1)

- Level 1
  - JBuildL1 local coincidences (Δt)
- Level 2
  - JBuildL2 local coincidences (Δt; $\cos(\theta)$; $M \geq 2$)
- Level 3
  - trigger3DMuon
  - trigger3DShower
  - …

multiple trigger algorithms can be applied to the same data

- Merge events
  - overlap in time
    - JDAQEvent::getOverlays()

JTriggerEfficiency

- initialisation
  - detector geometry and calibration
  - setup detector simulation
- event loop
  - input Monte Carlo event
  - add background
  - create JDAQTimeslice
  - calibrate data
  - application of trigger(s)
  - output JDAQEvent
  - output JDAQSummaryslice
- termination
  - close files

handled via interfaces, see next slides

same as JDataFilter and JTriggerProcessor

Interfaces

- JK40Simulator
  - generation of random background

    virtual void generateHits(JModule, JTimeRange, JModuleData& output);

    JModuleData : vector< JPMTData<JPMTSignal> > {};
- JPMTSimulator
  - simulation of PMT

    virtual void processHits(JPMTIdentifier, JCalibration, JStatus,

    JPMTData<JPMTSignal>& input,

    JPMTData<JPMTPulse>& output);
- JCLBSimulator
  - Simulation of CLB

    typedef vector< JPMTData<JPMTPulse> > JCLBInput;

    virtual void processData(JModuleIdentifier, JCLBInput input, JDAQSuperFrame& output);

JK40DefaultSimulatorInterface

- implements
  - JK40Simulator
- interface methods

```
virtual double getSinglesRate(JPMTIdentifier);
virtual double getCoincidenceRate(JModuleIdentifier, multiplicity);
virtual double getProbability(cos(θ));
```

JPMTDefaultSimulatorInterface

- implements
  - JPMTSimulator
- interface methods

```
virtual bool getPMTstatus(JPMTIdentifier,
        JTimeRange,      // time dependence
        JStatus);      // on/off
virtual JPMTSignalProcessorInterface      // see next slide
getPMTSignalProcessor(JPMTIdentifier);
```

JPMTSignalProcessorInterface

- interface methods

```
virtual bool applyQE();
virtual double getRandomTime(double t_ns);
virtual bool compare(JPhotoElectron,  JPhotoElectron);
virtual double getRandomCharge(int NPE);
virtual JChargeDomains applyThreshold(double npe);
virtual double getTimeOverThreshold(double npe);
virtual double getSurvivalProbability(int NPE);
virtual void merge(JPMTData<JPMTHit>&);
```

JCLBDefaultSimulatorInterface

- implements
  - JCLBSimulator
- interface methods

```
virtual JDAQHit  JTDC::makeHit(JPMT_t, double, JTOT_t);        // non-linearity
virtual bool  JStateMachine::maybeSwapped(JDAQHit, JDAQHit); // state machine

virtual int getUDPNumberOfReceivedPackets(JModuleIdentifier);
virtual int getUDPMaximalSequenceNumber(const JModuleIdentifier);
virtual bool hasUDPTrailer(const JModuleIdentifier);
virtual bool getHighRateVeto(JPMTIdentifier);                    // high-rate veto
```

Available implementations (1/3)

- JK40DefaultSimulator
  - ✓single rates
  - ✓multiple rates
  - ✓angular dependence of coincidence rates
- JPMTAnalogueSignalProcessor
  - ✓transition time spread
  - ✓gain spread
  - ✓threshold
  - ✓time-over-threshold
  - ✓hit merging

Available implementations (2/3)

- JCLBDefaultSimulator
  - ✓ TDC non-lenarity
  - ✓ hit re-shuffling

Available implementations (3/3)

- JK40RunByRunSimulator :
  - JK40DefaultSimulator
    - ✓ single rates from real data
    - ✓ multiple rates
    - ✓ angular dependence of coincidence rates
- JPMTRunByRunSimulator :
  - JPMTDefaultSimulator
    - ✓ PMT status from real data
- JCLBRunByRunSimulator :
  - JCLBDefaultSimulator
    - ✓ high-rate veto from real data

Detector simulation

```
JDetectorSimulator :
    JPMTRouter,      // fast access to PMT data
    JK40Simulator,
    JPMTSimulator,
    JCLBSimulator
{
    JDetectorSimulator(JDetector);   // input detector

    protected:
        std::auto_ptr<JK40Simulator>    k40Simulator;
        std::auto_ptr<JPMTSimulator>    pmtSimulator;
        std::auto_ptr<JCLBSimulator>    clbSimulator;
}
```

set your favourite simulator

PMT simulation (1/1)

- JMonitorK40

  -f <input file> // JDAQTimeslice

  -o <output file> // histograms

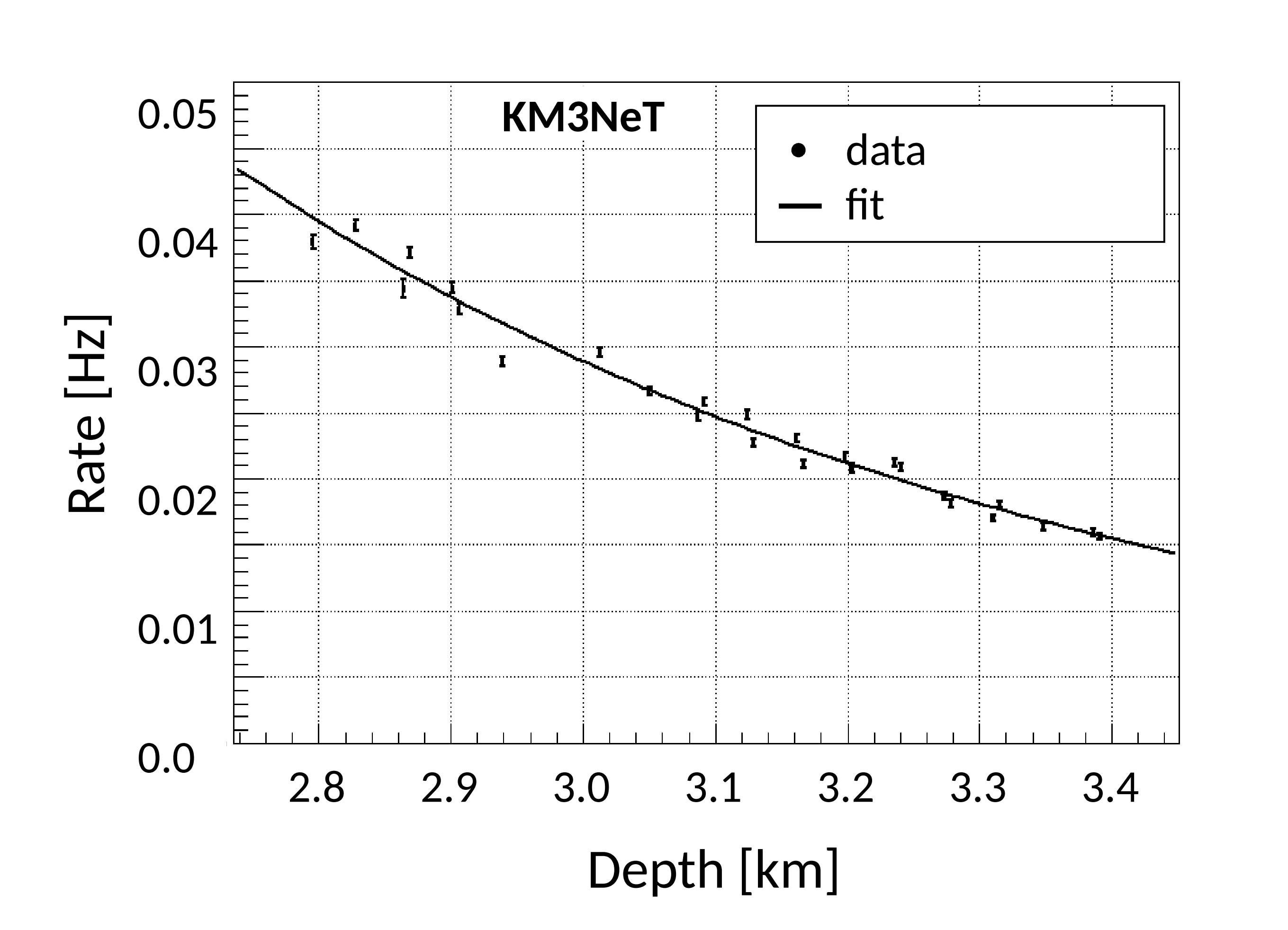
- JFitK40

  -f <input file> // histograms

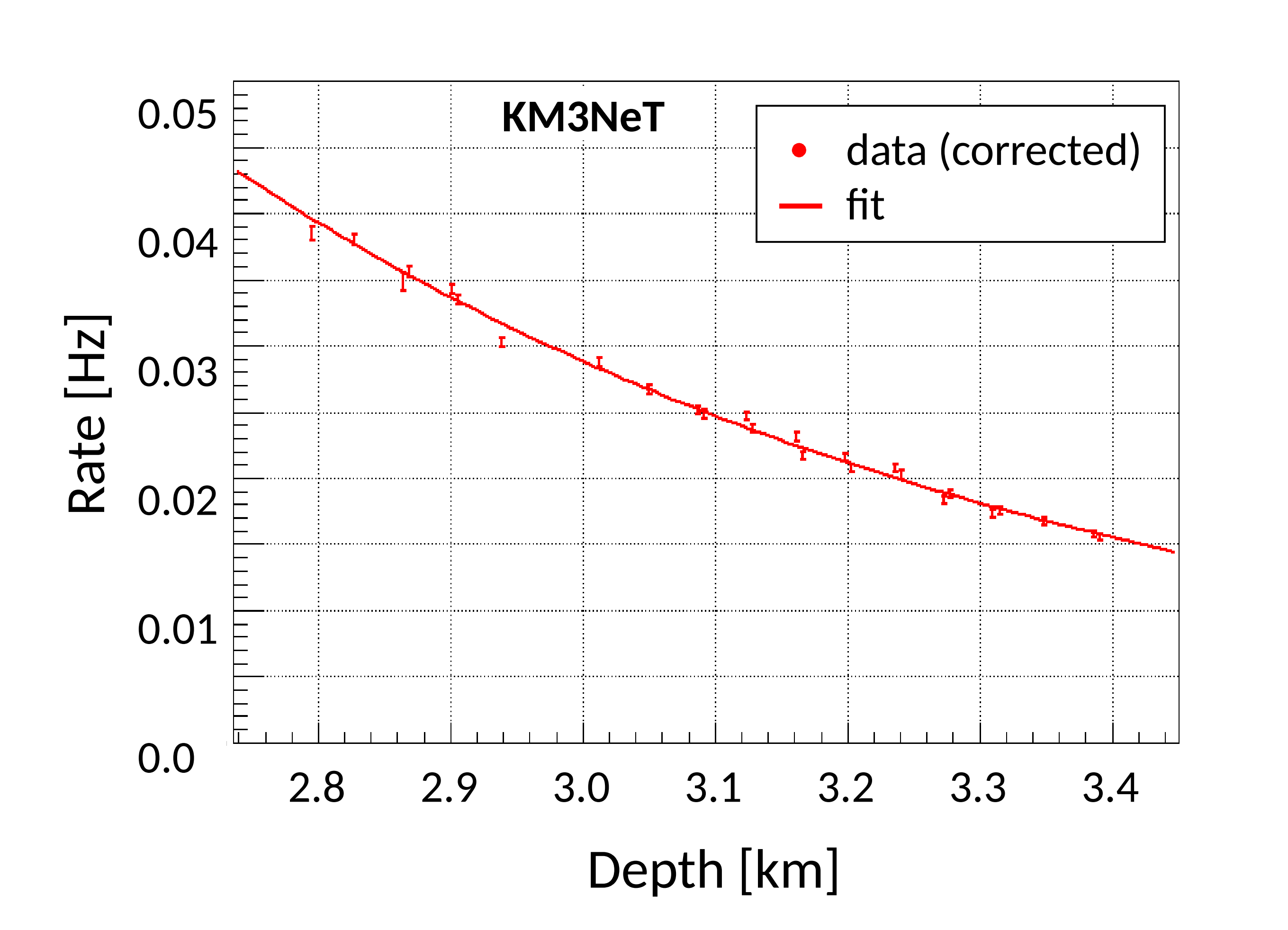
  -P <PMT efficiency file> //

PMT efficiency file can be input to JTriggerEfficiency, JEventTimeslice, JRandomTimesliceWriter

Muon depth dependence (1/1)



Muon depth dependence (1/1)



Time slice (1/1)

```
class JEventTimeslice :
    JDAQTimeslice   // as raw data (!)
{
    JEventTimeslice(      JDAQChronometer,  // RUN number, etc.
    JDetectorSimulator, // detector simulation
    Event,    // Monte Carlo event
    [JTimeRange]);   // time window
};
```

Event (1/6)

```
JDAQEvent :
    JDAQChronometer,
    JDAQTriggerCounter,   //  real data:   unique per thread
    //  Monte Carlo:  index to TTree
    JDAQTriggerMask    //  trigger bits of event
{
    const_iterator<..> begin<..>();
    const_iterator<..> end<..>();
    unsigned int size<..>();
    unsigned int getOverlays();
protected:
    std::vector<JDAQTriggeredHit>  triggeredHits;
    std::vector<JDAQSnapshotHit>  snapshotHits;
};
```

<..> select data type

Event (2/6)

```
JDAQTriggerMask {};        // 64 bit coded word, 1 bit per trigger
        // macro setTriggerBit(<trigger>, <bit>);

JDAQKeyHit :
        JDAQModuleIdentifier,   // module
        JDAQHit        // PMT hit
{};

typedef  JDAQKeyHit       JDAQSnapshotHit;    // snap shot hit

JDAQTriggeredHit :          // triggered hit
        JDAQKeyHit,       // same as snapshot hit
        JDAQTriggerMask     // trigger bits of hit
{};
```

Event (3/6)

trigger(s)

snapshot

{JDAQSnapshotHit,	{JDAQTriggeredHit}	,JDAQSnapshotHit}

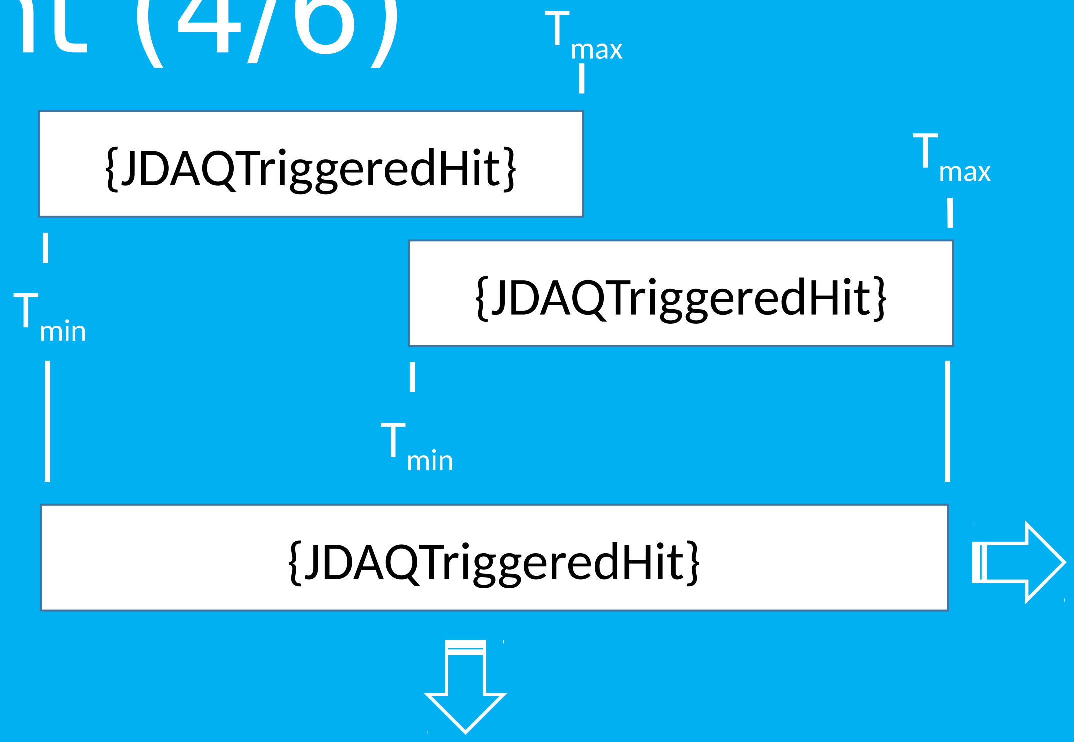
T_{min} - ΔT $\qquad$ T_{min} $\qquad$ T_{max} $\qquad$ T_{max} + ΔT

time ⟶

snapshot¶ = All raw data between $[T_{min} - \Delta T, T_{max} + \Delta T]$

¶ ΔT = nD/c (where D corresponds to size of detector)

Event (4/6)




```
JDAQEvent :
      ..
      JDAQTriggerMask     //  ORed
{
      unsigned int getOverlays();   //  incremented
};
```

```
JDAQTriggeredHit :
      ..
      JDAQTriggerMask     // ORed
{}
```

Event (5/6)

```
class JTriggeredEvent :
    JDAQEvent   // I/O
{
    JTriggeredEvent(    JEvent,  // internal to trigger
    JTimesliceRouter,    // see next slide
    TMaxLocal_ns,   // L1 time window
    [snapshot]);// option
};
```

Event (6/6)

```
class JTimesliceRouter :
      JPointer<const JDAQTimeslice>   // pointer to raw data
{
            configure(JDAQTimeslice);    // input

            JDAQFrameSubset getFrameSubset( JDAQModuleIdentifier,
                        JTimeRange);
};
```

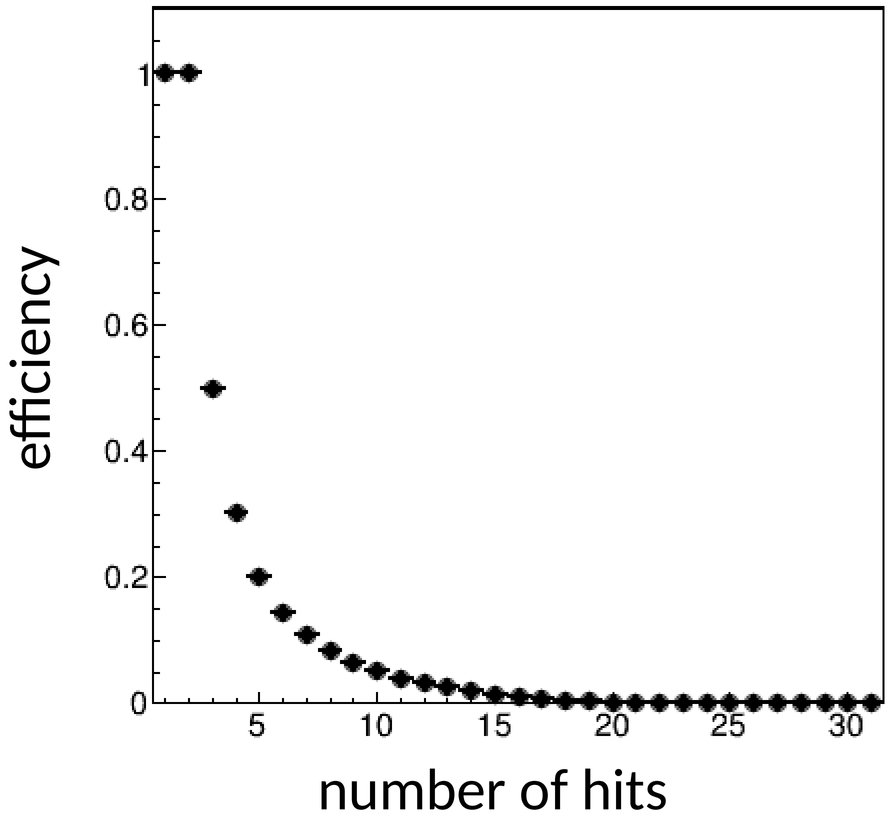
subset of JDAQSuperFrame

Examples (1/4)

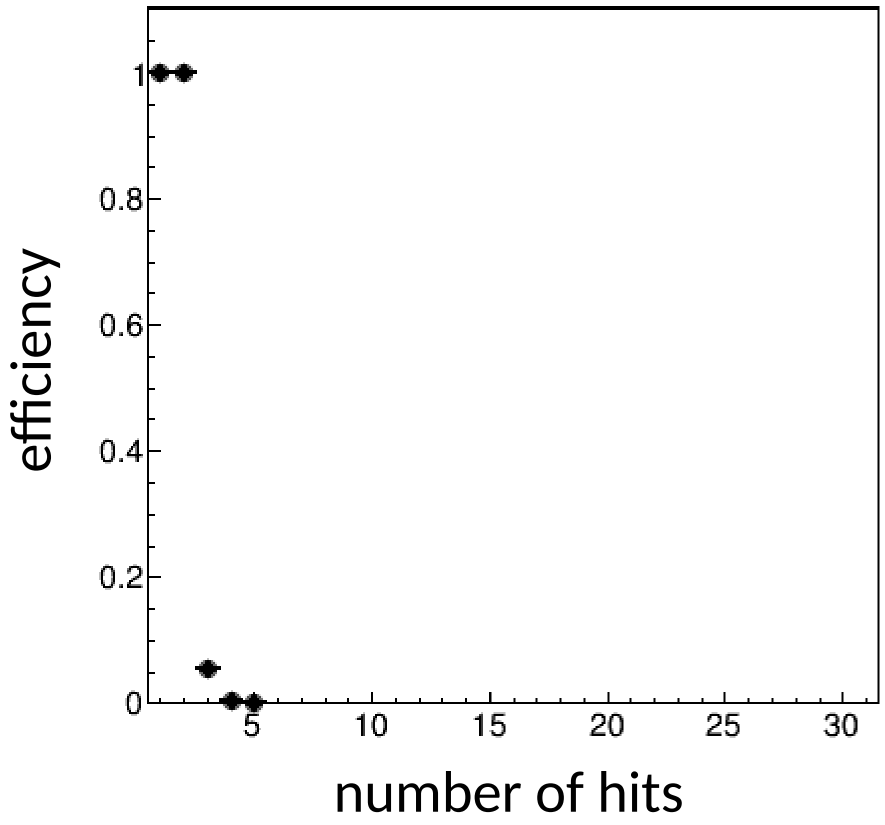
- JSignalL1 & JRandomL1
  - test efficiency and purity of L1-L2 coincidence logic
- JFilter
  - test efficiency and purity of cluster methods
- JVolume1D
  - plot trigger effective volume

Examples (2/4)

JSignalL1.sh

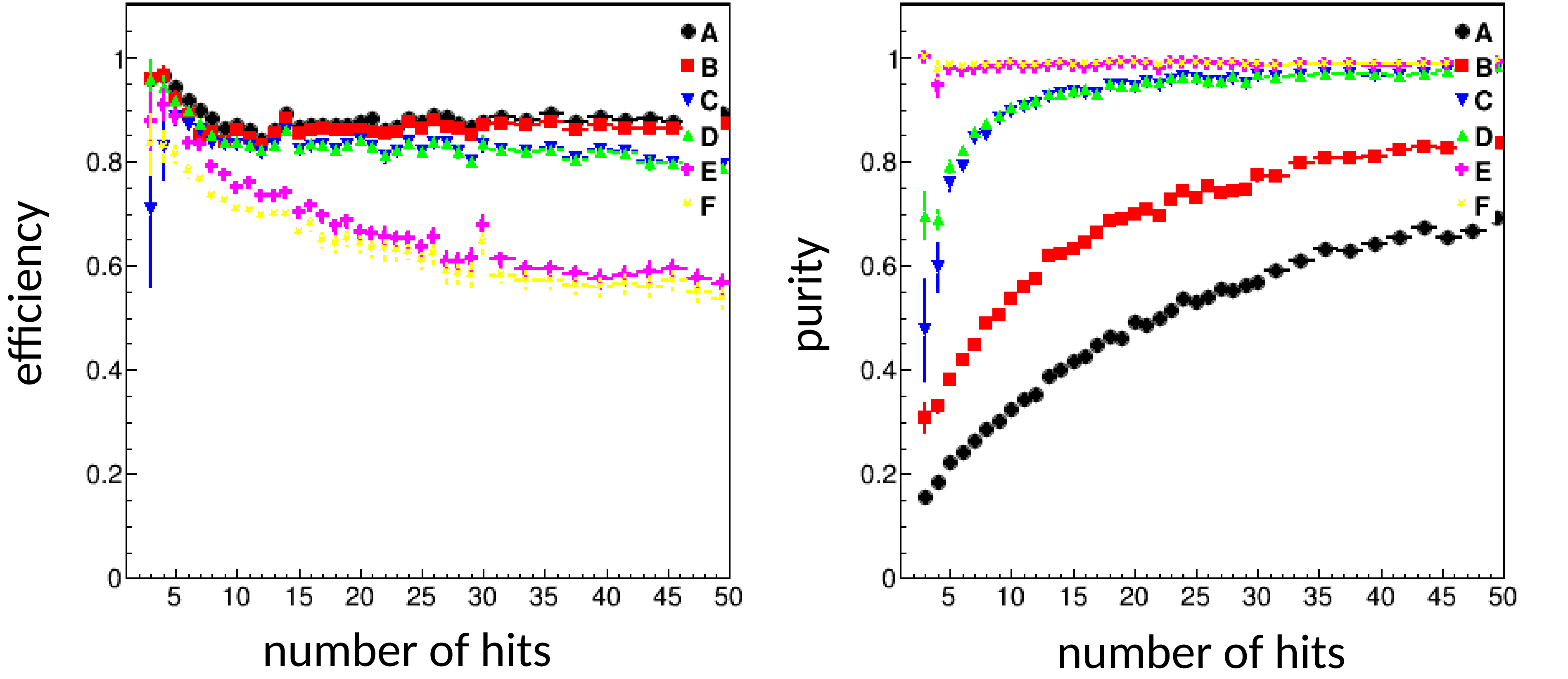


JRandomL1.sh



Examples (3/4)

JFilter.sh



Examples (4/4)

JVolume1D.sh