New trigger for ORCA

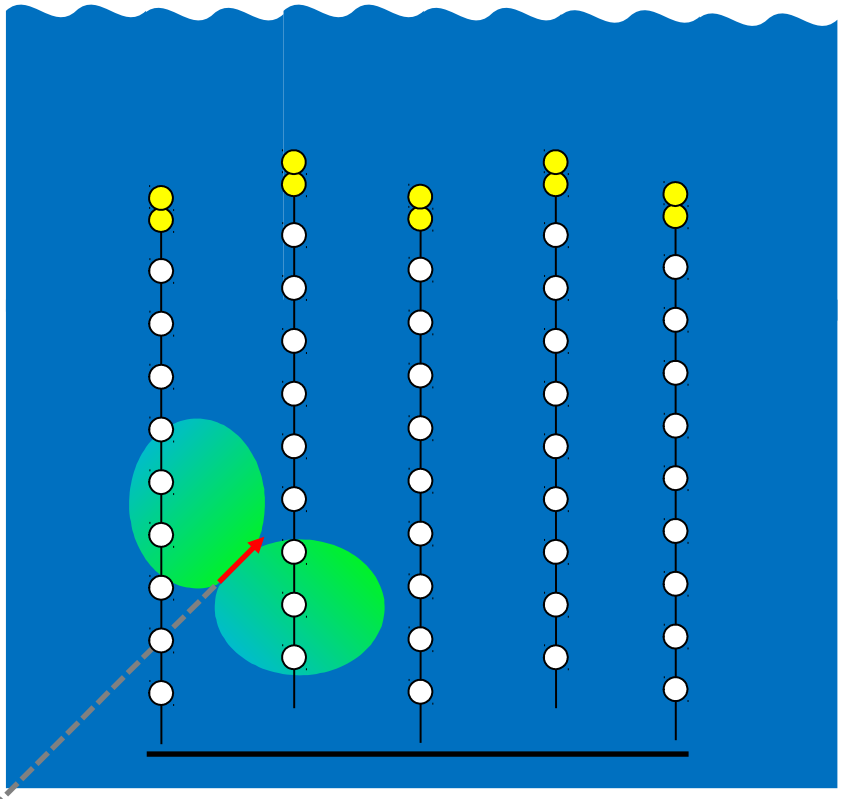
M. de Jong

Introduction (1/2)

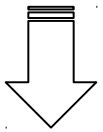
- Shower trigger based on a mix of L1 & L0 hits
- Total rates
  - $R_{L0} \cong 300$ MHz
  - $R_{L1} \cong 2$ MHz
- Typical time window
  - $\Delta T \cong 100$ ns

$$R_{L0} \times \Delta T \gg 1$$

Introduction (2/2)



size of shower
much smaller than
size of detector



limit distance
between
L1/L0 hits

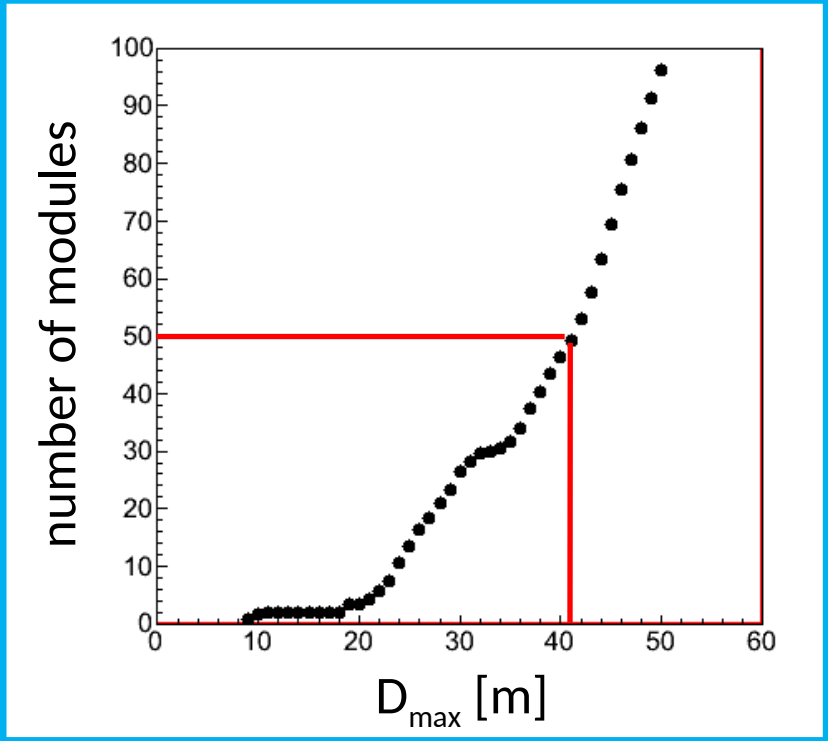
Algorithm (1/4)

- Build 1D array of time sorted L1 hits
  - Total L1 rate in detector
    - $R_{L1} \cong 115 \times 18 \times 1\ kH \cong 2\ MHz \implies \Delta T_{L1} \cong 0.5\ \mu s$
- Build 2D array of time sorted L0 hits
  - organised by module index¶
  - Total L0 rate per module
    - $R_{L0} \cong 31 \times 5\ kHz \cong 150\ kHz \implies \Delta T_{L0} \cong 6\ \mu s$

¶ Index refers to position of module in detector data structure

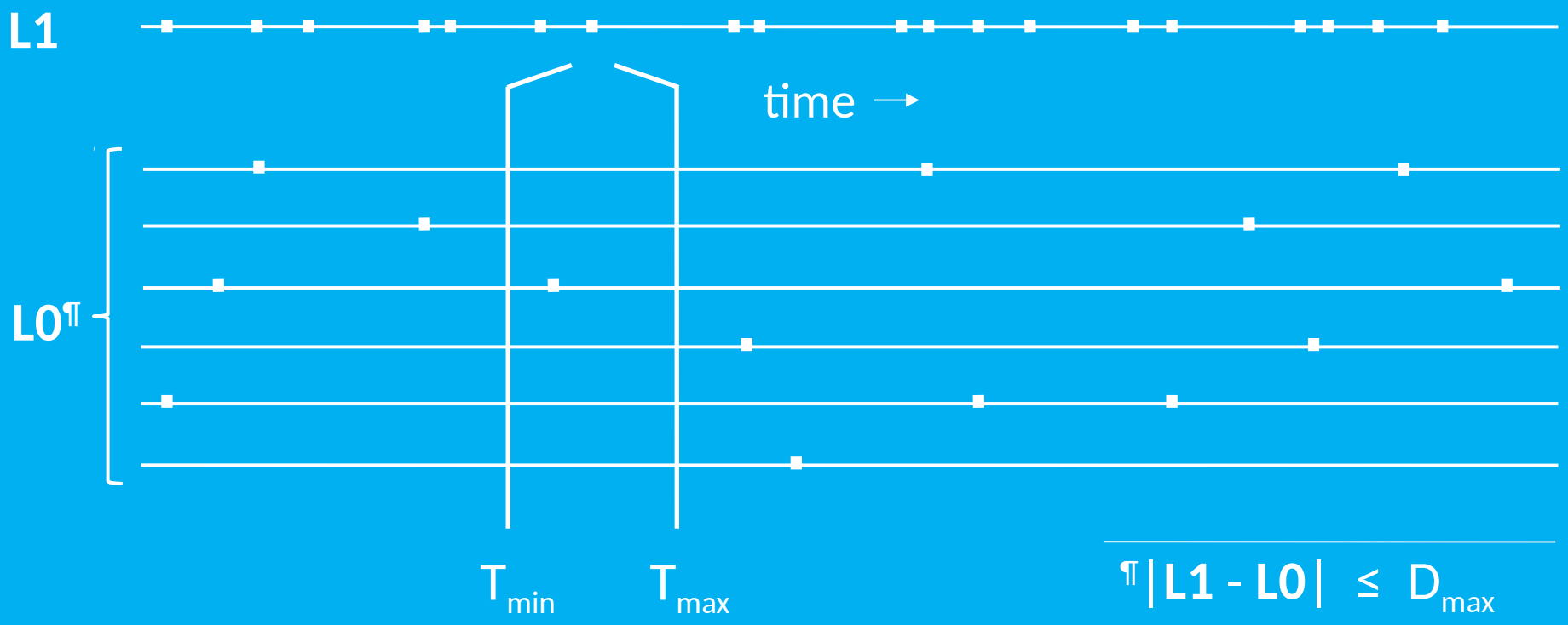
Algorithm (2/4)

- Module map¶
  - list of modules within maximal distance



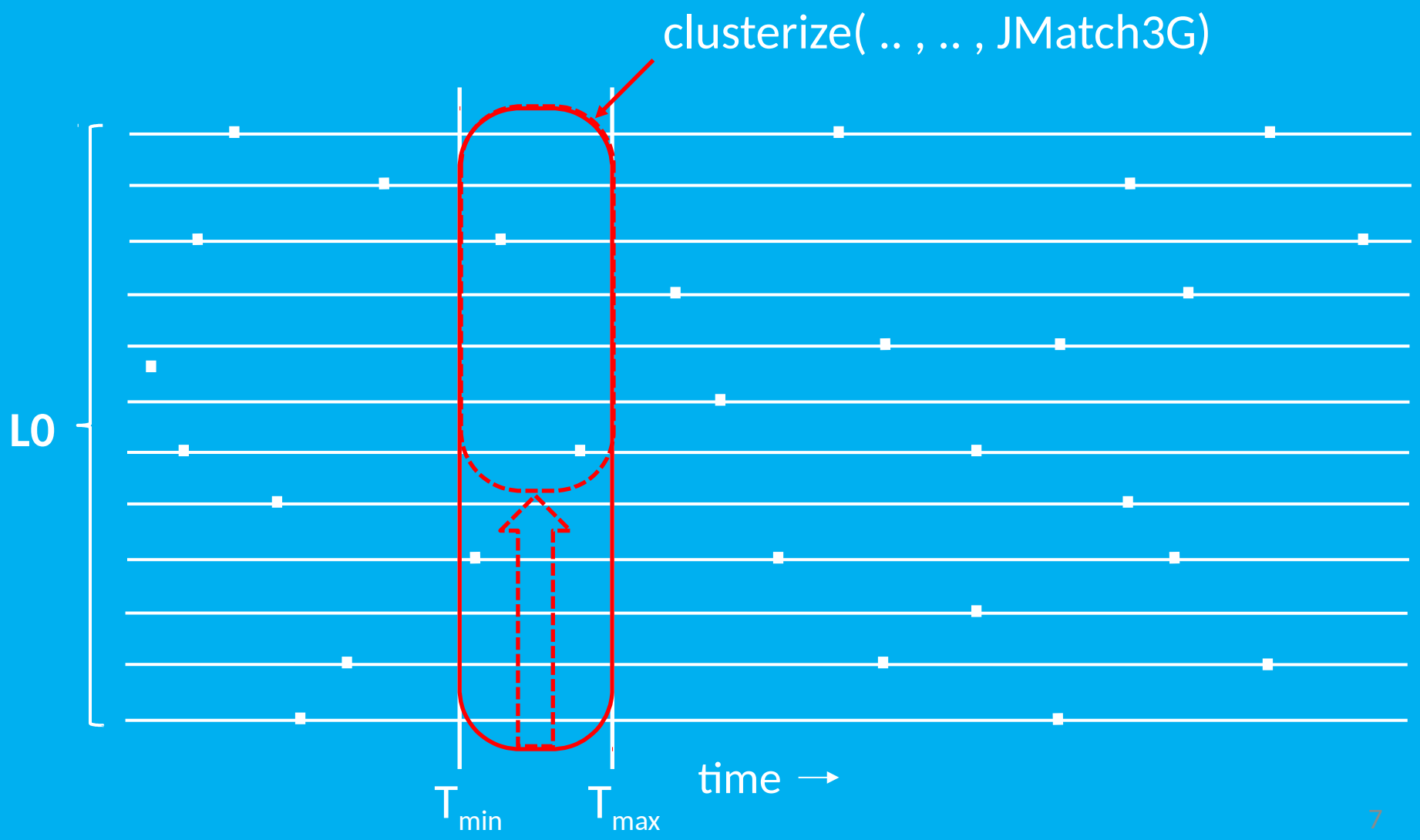
¶ see <Jpp>/examples/JDetector/JModuleMapper

Algorithm (3/4)



- maintain hit iterator for each L0 frame
- increment hit iterator until hit->getT() >= T_{min}

Algorithm (4/4)



JTriggerMXShower

```
JTimesliceClone<..> clone(inputL0, mapper);

for (JTriggerInput::const_iterator root = inputL1.begin(); root != inputL1.end(); ++root) {

    buffer.clear();

    const JList_t& zip = mapper.getList(root->getModuleID());


    for (typename JList_t::const_iterator mod = zip.begin(); mod != zip.end(); ++mod) {

        const double Tmin   = root->getT() + mod->Tmin_ns;
        const double Tmax   = root->getT() + mod->Tmax_ns;


        JFrame_t&    frame  = clone[mod->first];


        frame.fast_forward(root->getT() - TMaxEvent_ns);

        :
```

start iterators

L0 buffer

associated modules

module attributes

direct indexing

increment iterator

strictly time ordered

```
JTimesliceClone<..> clone(inputL0, mapper);

for (JTriggerInput::const_iterator root = inputL1.begin(); root != inputL1.end(); ++root) {
    :
    for (typename JList_t::const_iterator mod = zip.begin(); mod != zip.end(); ++mod) {
        :
        for (typename JFrame_t::const_iterator i = frame.get(); *i <= Tmax; ++i) {
            if (*i >= Tmin) {
                buffer.push_back( JHitR1( frame.getModuleID(),
                                          frame.getPosition(),
                                          frame.getJHit(*i)));
            }
        }
    }
    :
```

fill L0 buffer

```
JTimesliceClone<..> clone(inputL0, mapper);

for (JTriggerInput::const_iterator root = inputL1.begin(); root != inputL1.end(); ++root) {
    :
    if (buffer.size() >= parameters.numberOfHits - 1) {
        if (buffer.size() < parameters.factoryLimit) {
            iterator q = clusterize(buffer.begin(), buffer.end(), match);
            if (distance(buffer.begin(),q) >= parameters.numberOfHits - 1) {
                // Trigger
            }
        } else {
            // Trigger
        }
```

← *test L0 buffer*

← *factory limit*

← *JMatch3G*

JTriggerMXShower

- parameters (with default values)

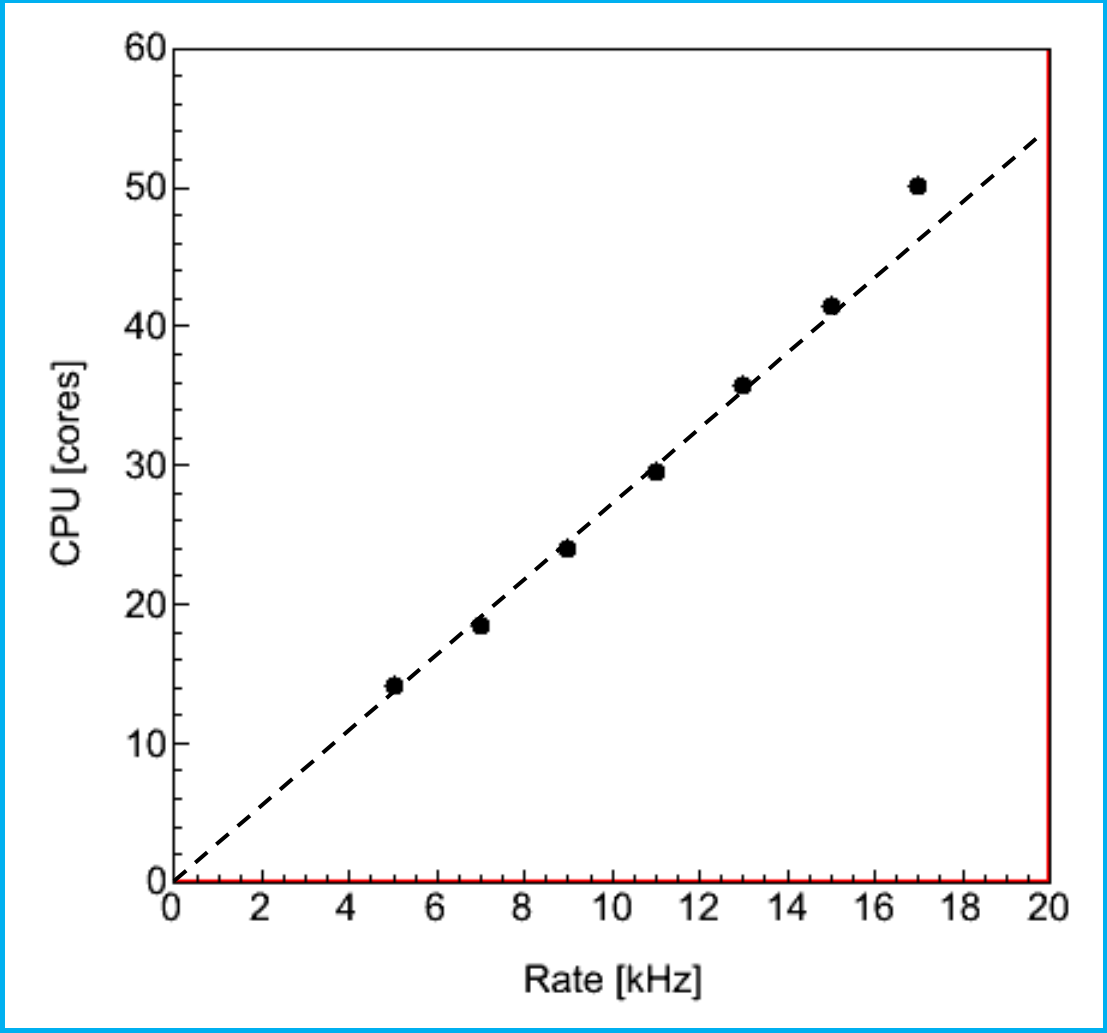
  triggerMXShower.enabled = 0;

  triggerMXShower.numberOfHits = 7;

  triggerMXShower.DMax_m = 43;

  triggerMXShower.TMaxExtra_ns = 20;

CPU◊ usage (1/2)



◊ Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz

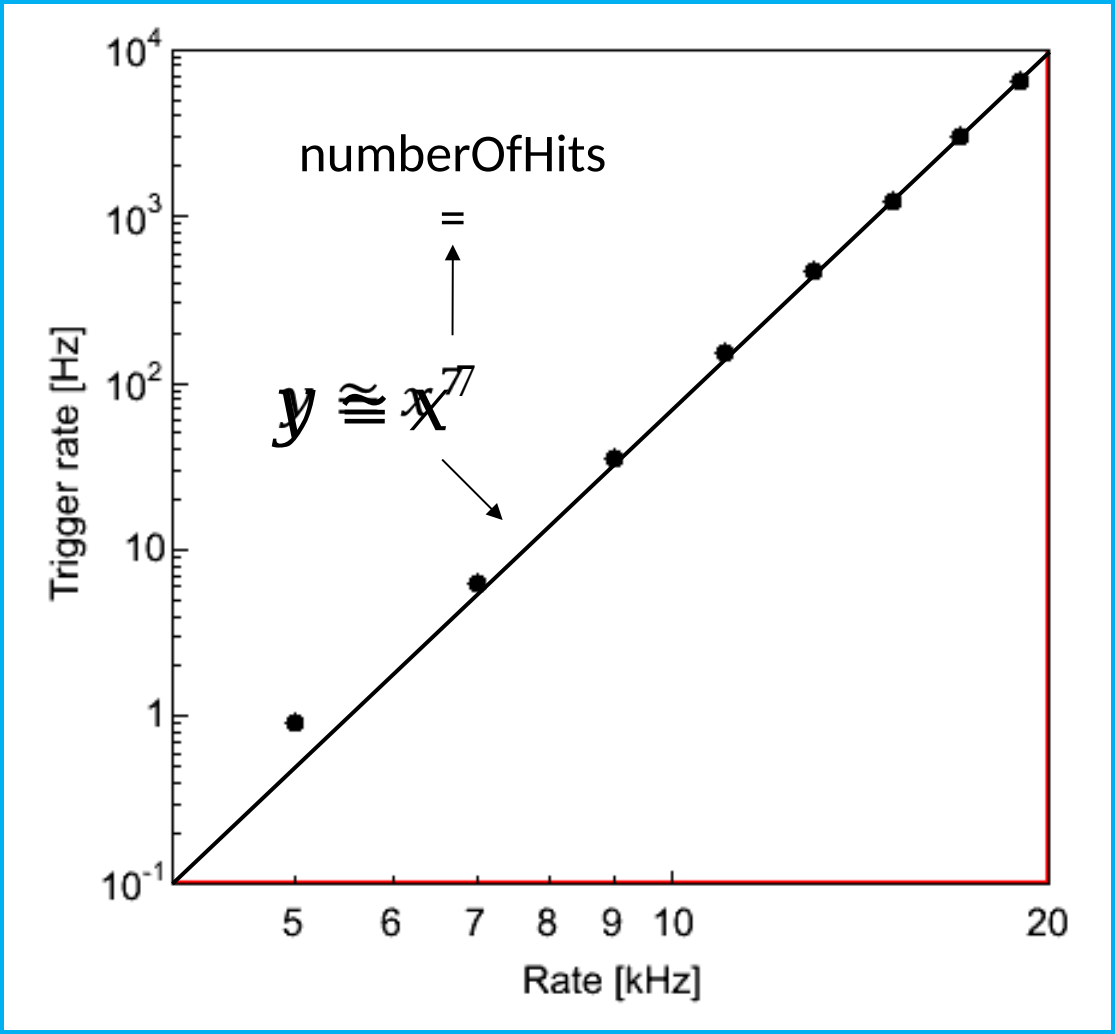
CPU◊ usage (2/2)

Level	Live time [ms]	CPU time [ms]	Ratio CPU/Live
calibration	100	65	< 1
L0 (sort)	100	800	8
L1	100	25	< 1
L2	100	70	< 1
Time slice router	100	100	1
Trigger	100	400	4
Total	100	1460	15

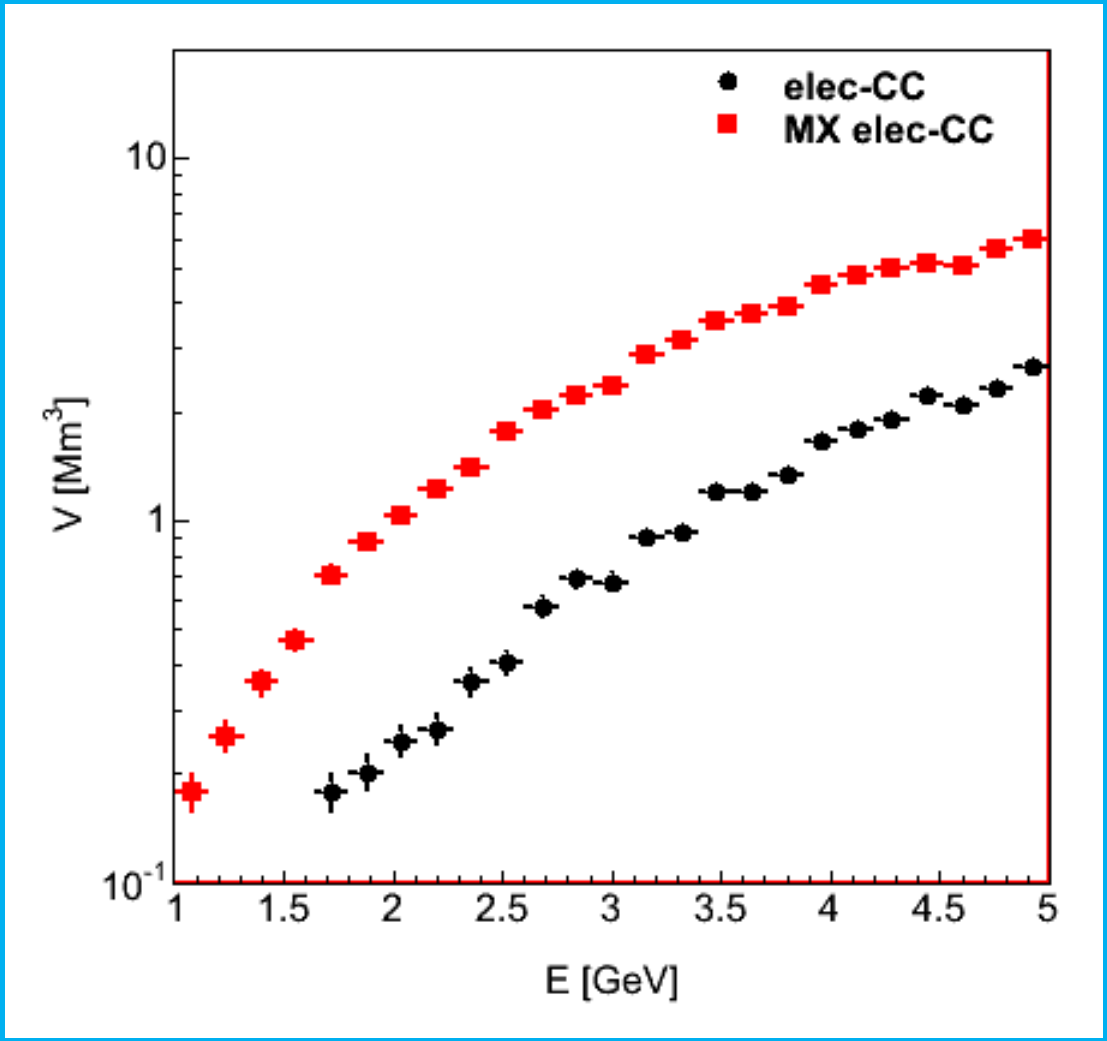
!

◊ Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
Random background as a function of multiplicity 5000, 500, 50, 5, 0.5 Hz, respectively

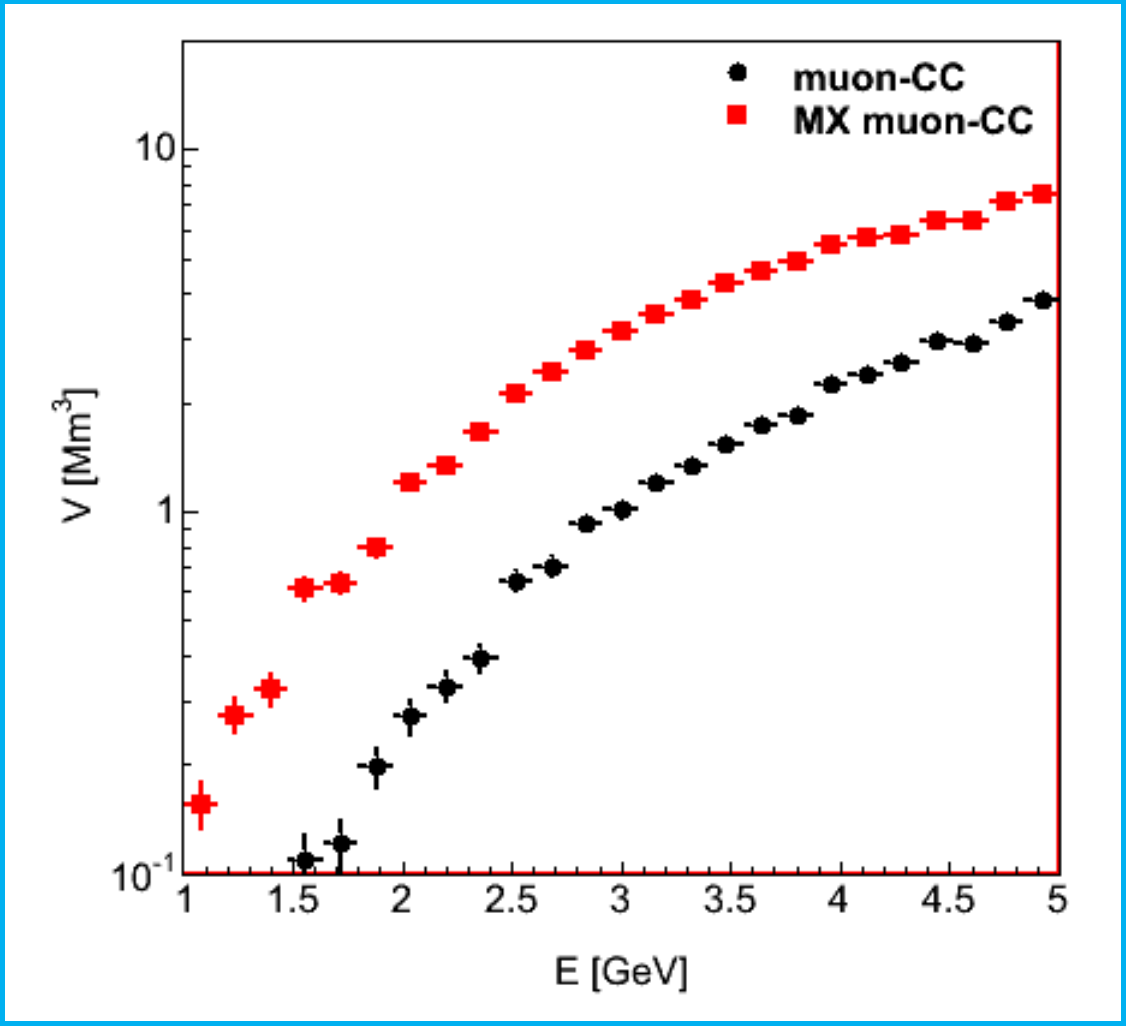
Random background



Effective volume ν_e



Effective volume ν_μ



Status & Outlook

- New ORCA trigger available
  - CPU power consumption very low
  - reasonable purity
  - improved efficiency
  - input parameters to be (fine) tuned

Note that normalisation of ORCA Monte Carlo data requires to process gSeaGen files