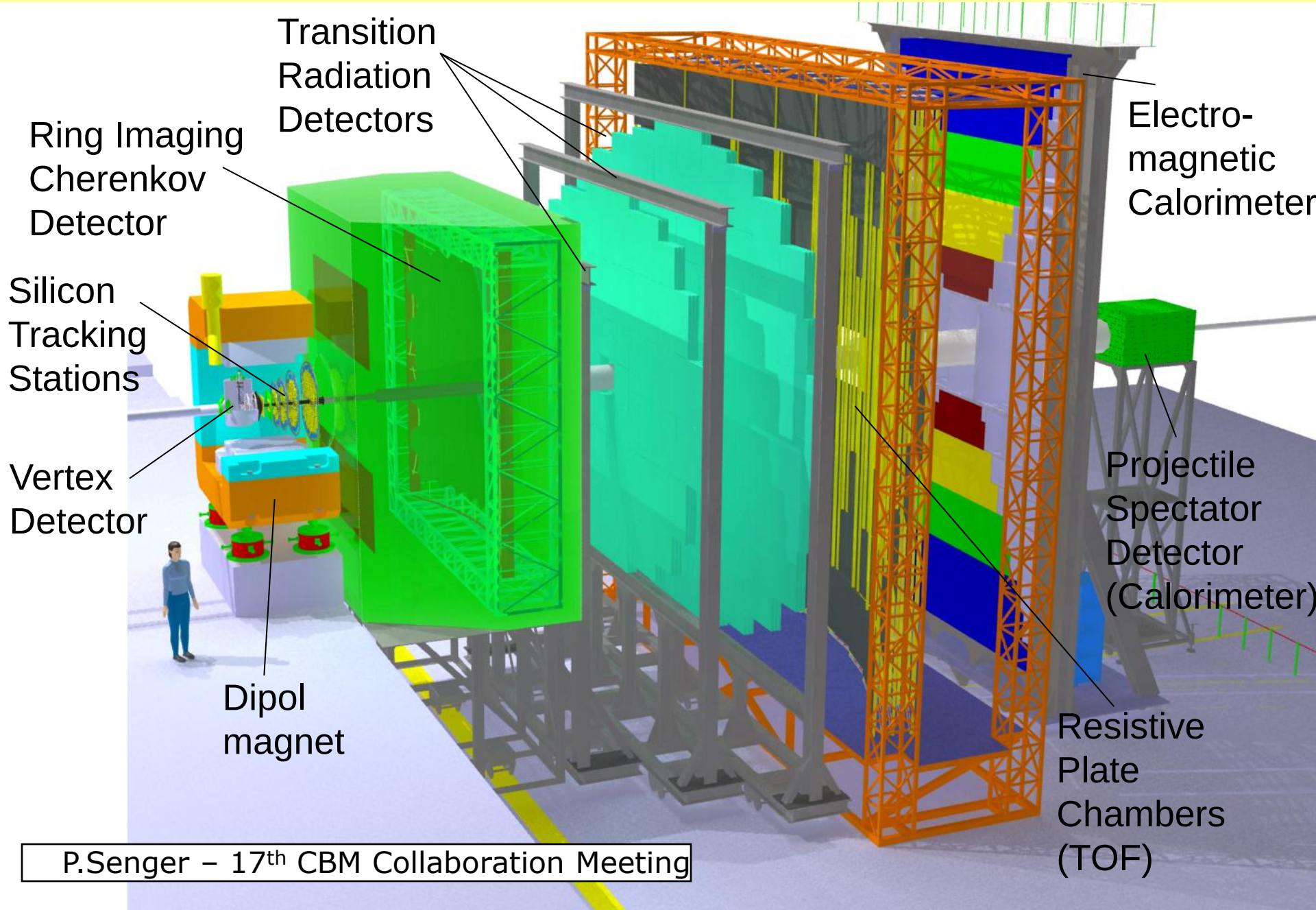
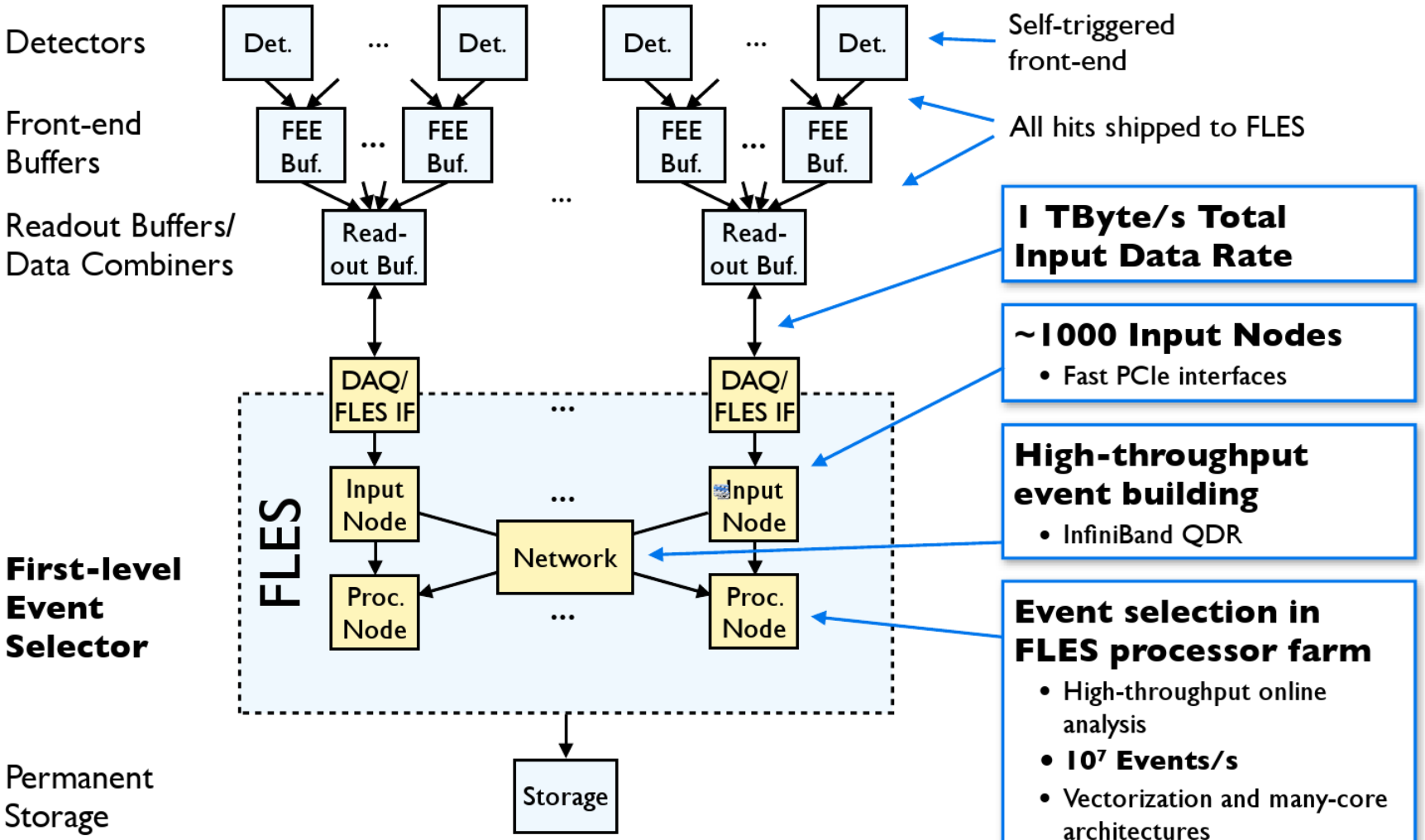
Evaluation of InfiniBand for CBM experiment

Sergey Linev
GSI, Darmstadt, Germany

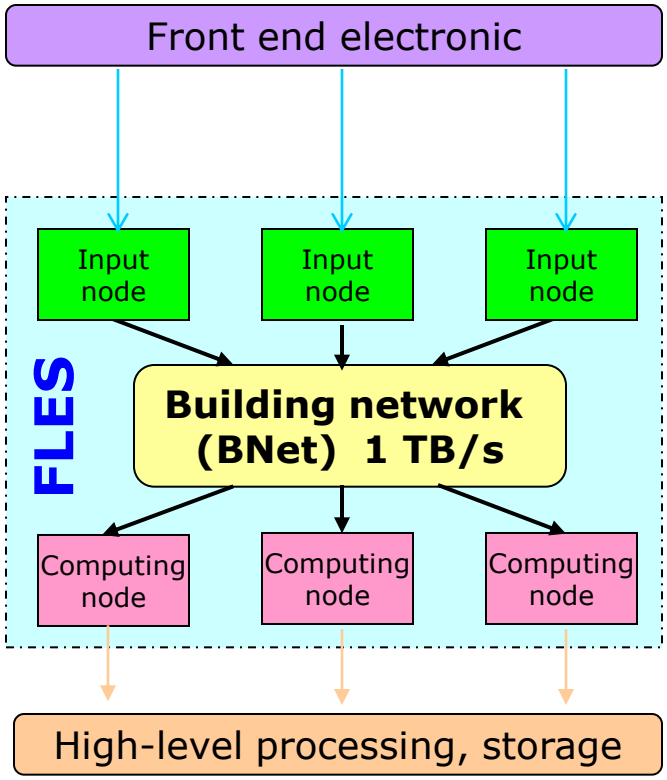
The Compressed Baryonic Matter Experiment



CBM Online Computing and Readout

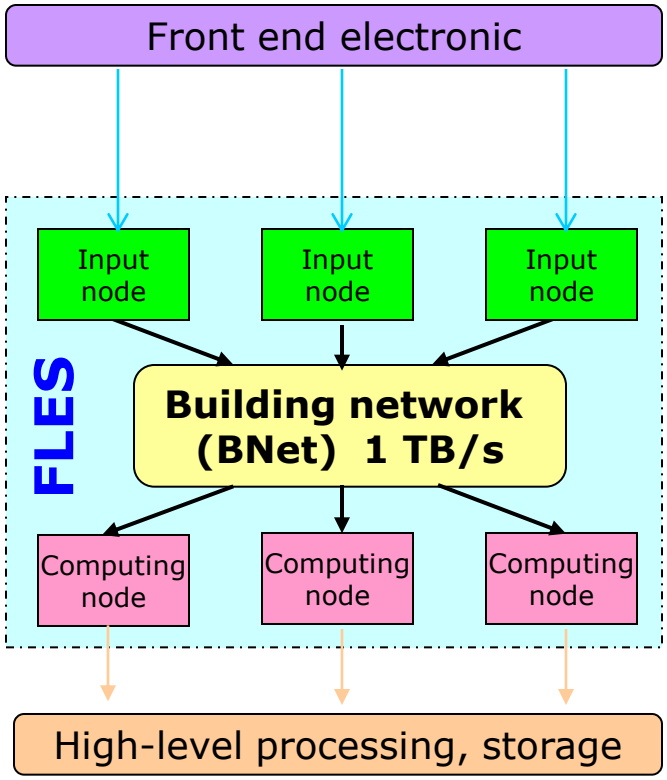


FLES - First Level Event Selection



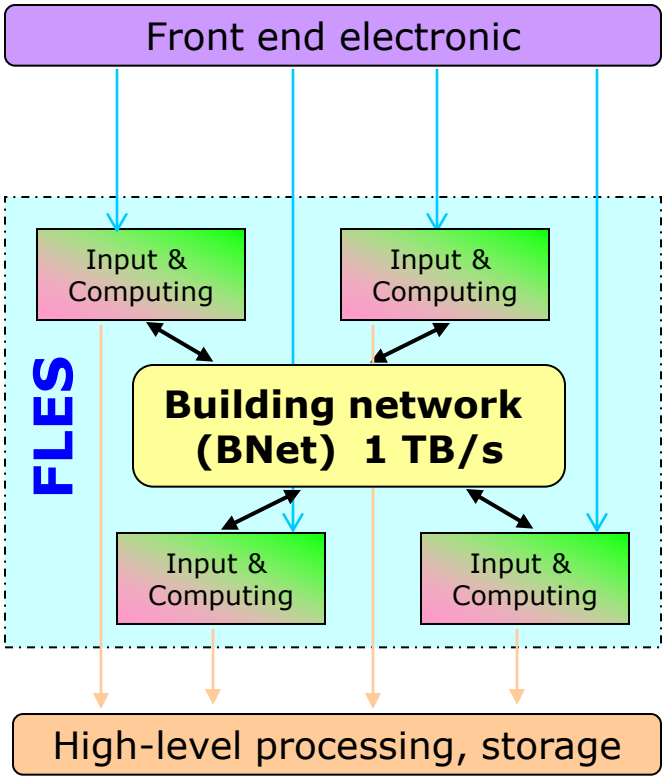
unidirectional

FLES - First Level Event Selection



unidirectional

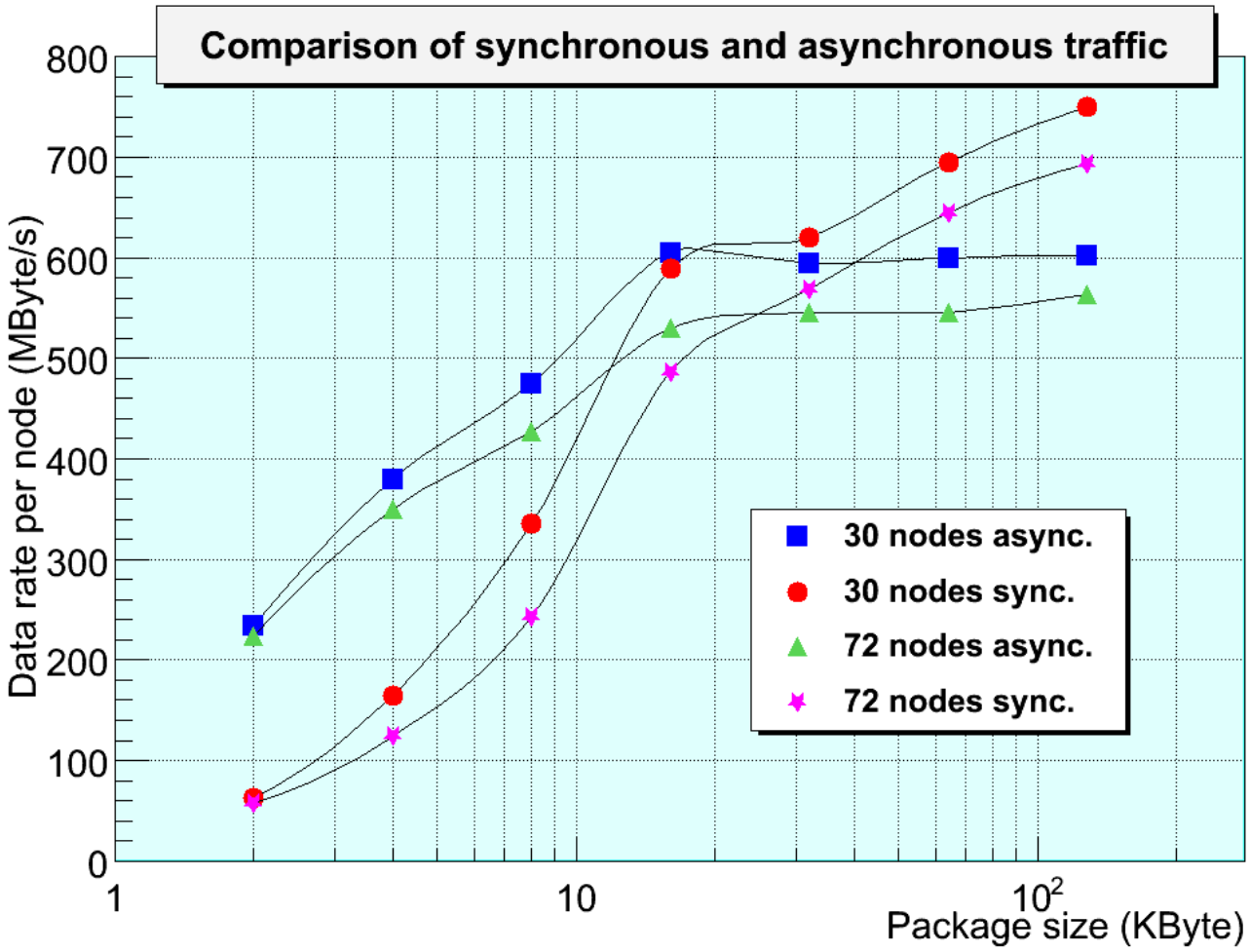
bidirectional



Scheduled transfer approach

- Data flow in FLES
  - ~1000 input nodes, ~1GB/s/node
  - input nodes could be used as computing nodes (bidirectional approach)
  - data, belonging to the same time interval, should be collected on the same computing node
  - all-to-all traffic ~1 TB/s
- Data rates are huge – one should help network to coupe with such rates
- Scheduled transfer
  - defines when node can transfer data to other nodes
  - could (must?) avoid congestions in the network and balance transfers between available links
  - very much depends from network topology and routing

First IB tests (2006-2007)



LOEWE-CSC cluster

https://csc.uni-frankfurt.de

Hardware:

- 832 nodes in 34 water-cooled racks
- 20,928 CPU cores
- 778 GPGPU hardware accelerators
- 56 TB RAM
- over 2 PB aggregated disk capacity
- QDR InfiniBand interconnects
- 46 Mellanox InfiniScale IV switches

Installed in late 2010 in Industriepark Höchst

Cluster performance:

- CPUs performance (dp): 176 TFlop/s (peak)
- GPUs performance (sp): 2.1 PFlop/s (peak)
- GPUs performance (dp): 599 TFlop/s (peak)
- Cluster performance HPL: 299.3 TFlop/s
- Energy efficiency Green500: 740.78 MFlop/s/Watt

First results on LOEWE

- Use OFED VERBs for test app

- Point-to-point:
  - one-to-one 2.75×10^9 B/s
  - one-to-many 2.88×10^9 B/s
  - many-to-one 3.18×10^9 B/s

- all-to-all scheduled transfer:
  - avoids congestion on receiving nodes
  - about 2.1×10^9 B/s/node
  - scales good up to 20 nodes
  - BUT - performance degrading with nodes increase

- Same problem as before
  - should one take into account network topology?
  - LOEWE-CSC cluster uses ½ fat tree topology

Fat-tree topology

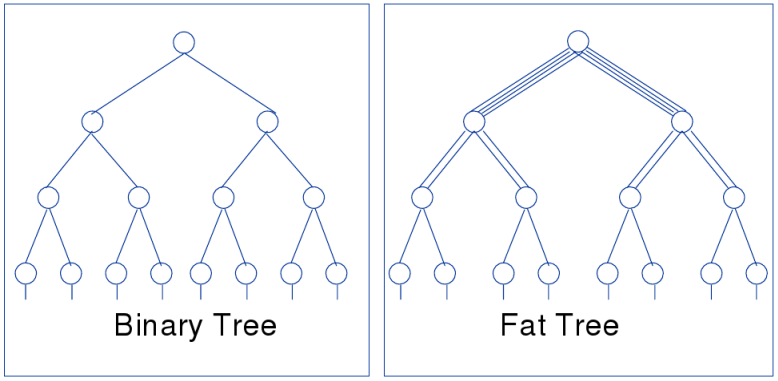


Figure 1. Binary and Fat Tree Topologies

CBB – constant bisectional bandwidth

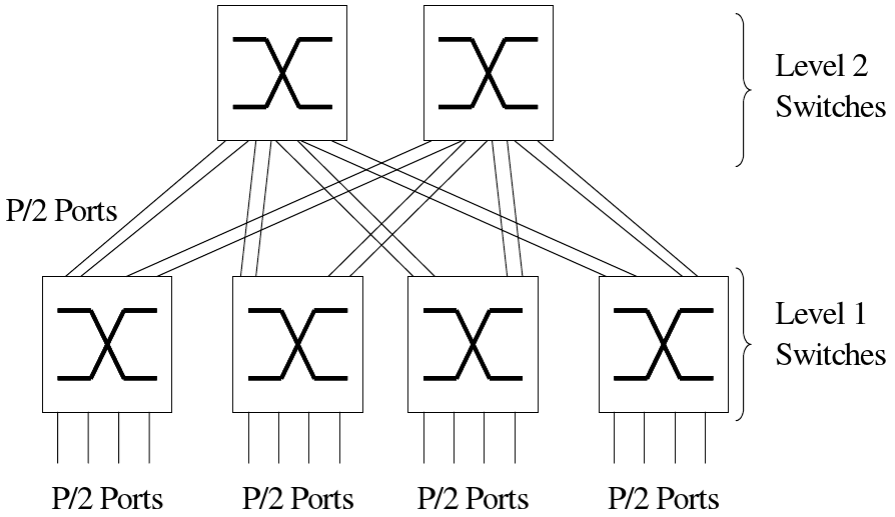
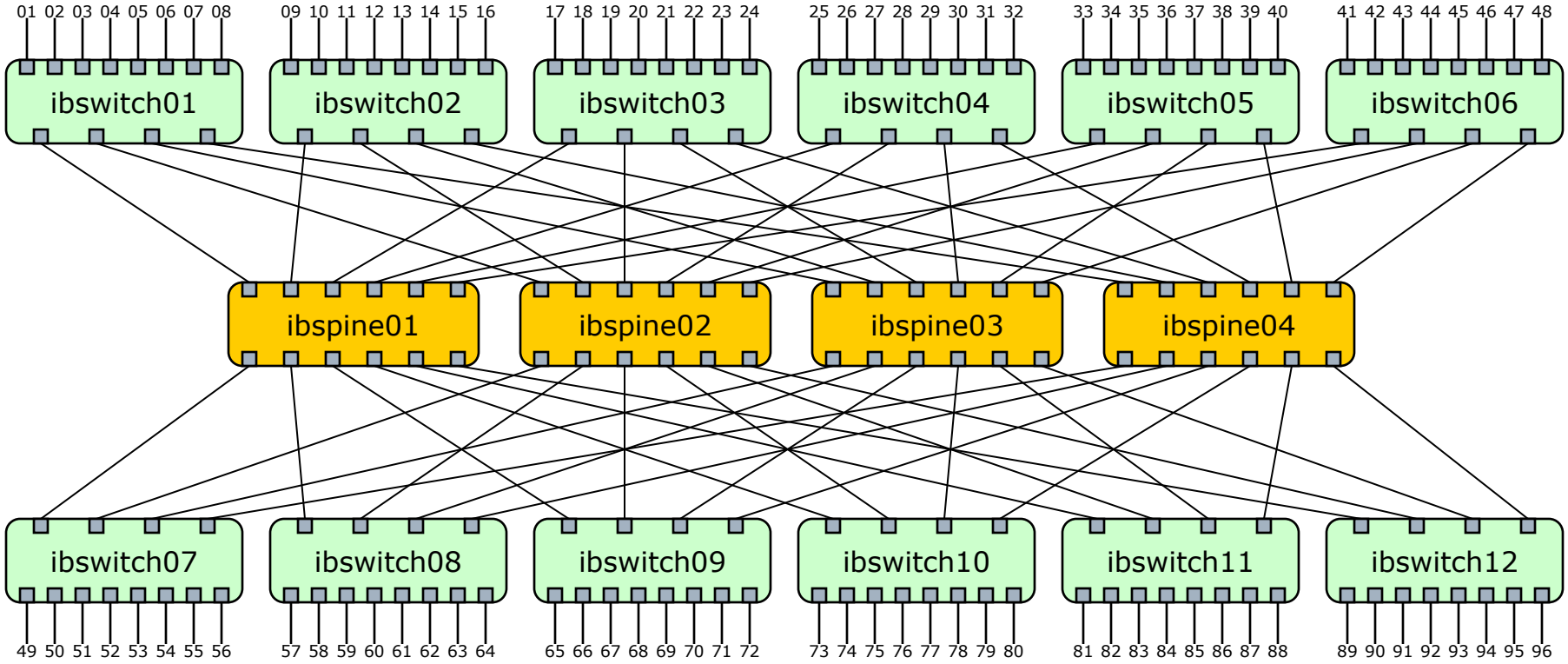


Figure 2. Example of a simple two level CBB switch topology

Figures from Mellanox whitepaper:

http://www.mellanox.com/pdf/whitepapers/IB_vs_Ethernet_Clustering_WP_100.pdf

½ fat tree topology



On the example half-fat-tree topology with 12-port switches is shown. It has 12 leaf switches, 4 core switches and 96 end nodes

With 36-port InfiniScale IV switches one can build ½ fat-tree fabric for maximum 36x24=864 end-nodes. In normal fat-tree 36x18=648 nodes would be possible.

Routing* exploration

- *ibnetdiscover* produces full list of nodes and switches in subnet
- *ibtracert* gives route between two LIDs

```
[linev@login02]$ ibtracert 5024 4464
From ca {0x002590ffff16039c} portnum 1 lid 5024-5039 "login02 HCA-1"
[1] -> switch port {0x0002c90200421930}[1] lid 29-29 "MF0;ibswitch15:IS5030/U1"
[18] -> switch port {0x0002c9020041dc28}[25] lid 17-17 "MF0;ibspine08:IS5035/U1"
[14] -> switch port {0x0002c90200421a30}[20] lid 119-119 "MF0;ibswitch02:IS5030/U1"
[6] -> ca port {0x002590ffff161de5}[1] lid 4464-4479 "node1-036 HCA-1"
To ca {0x002590ffff161de4} portnum 1 lid 4464-4479 "node1-036 HCA-1"
```

- route between two nodes than

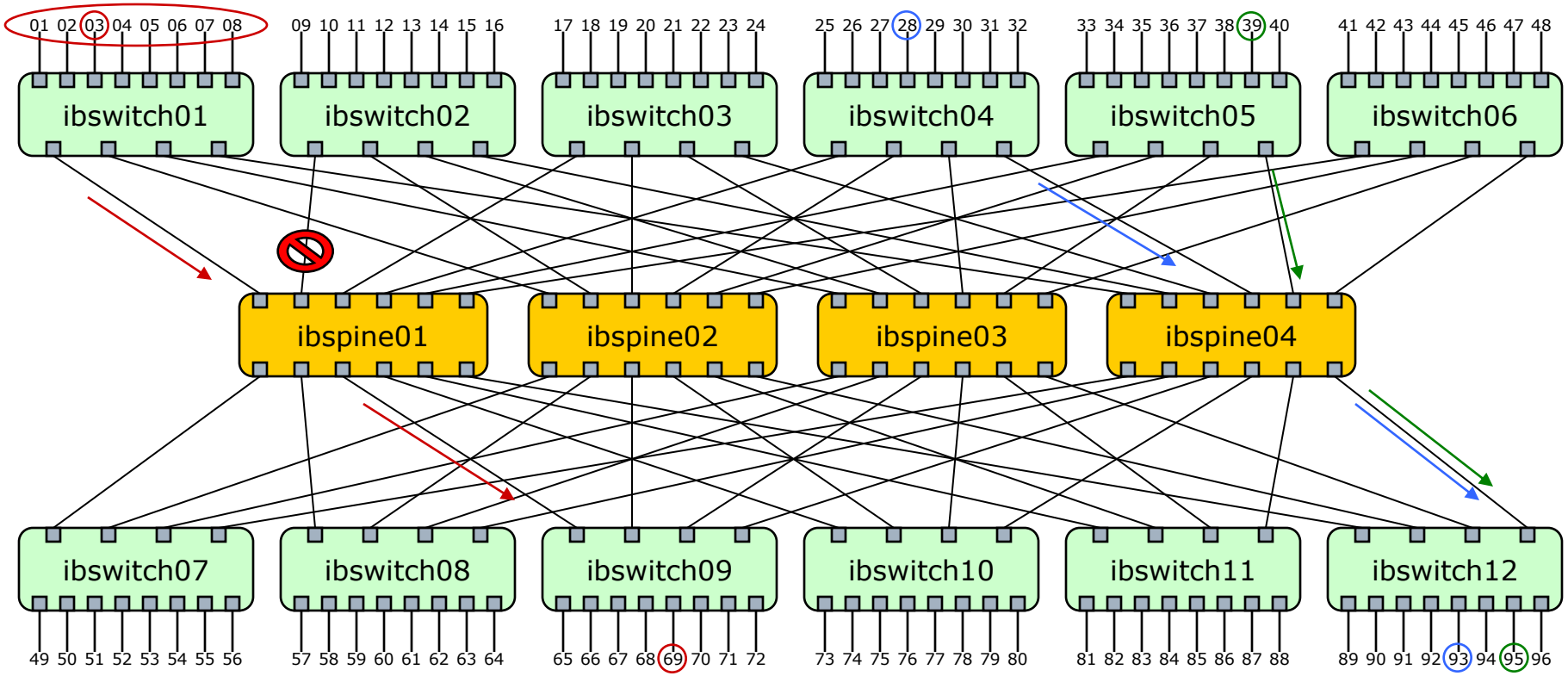
```
login02   -> ibswitch15 -> ibspine08 -> ibswitch02 -> node1-036
node1-036 -> ibswitch02 -> ibspine02 -> ibswitch15 -> login02
```

- small shell script to scan all combination of ports pairs
- first scan took ~8 hours

*According to IB specs packet transport in subnet called forwarding.
Term routing in IB used to indicate packet transport between subnets via routers.

Routing – first observations

- Route depends only from DLID
- Routes distributed non-uniform between spines switches
- There were broken links
- Routing tables can be changed on-the-fly
- Measured link speed 3.89×10^9 B/s



Subnet manager

- Runs on one of the core switches
  - can be configured on any host PC
- Tasks (not all) of subnet manager are:
  - discover nodes in the net
  - assign LID (Local IDentifier) to the ports
  - set routing tables for the switches
- According to IB specs, route between two ports defined by source (SLID) and destination (DLID) identifiers
- Open questions – easy possibility of:
  - fixed LID assignment?
  - fixed (regular) routing tables?

Routing – properties

- Using obtained routing tables, one can estimate number of congestions for different kind of traffics
- In ½ fat tree congestion means that more than 2 transfer goes via the same link
- For simple round-robin transfer
  - 1.8 transfer/link average, but
  - 6 transfer at maximum per link
  - all the time more than 10% of transfers with congestions
- One could try to optimize schedule
  - take into account routing tables to avoid congestions
- Main problem
  - there are many physical paths between two nodes but
  - only single path is available for node1 -> node2 transfer
  - no real optimization is possible

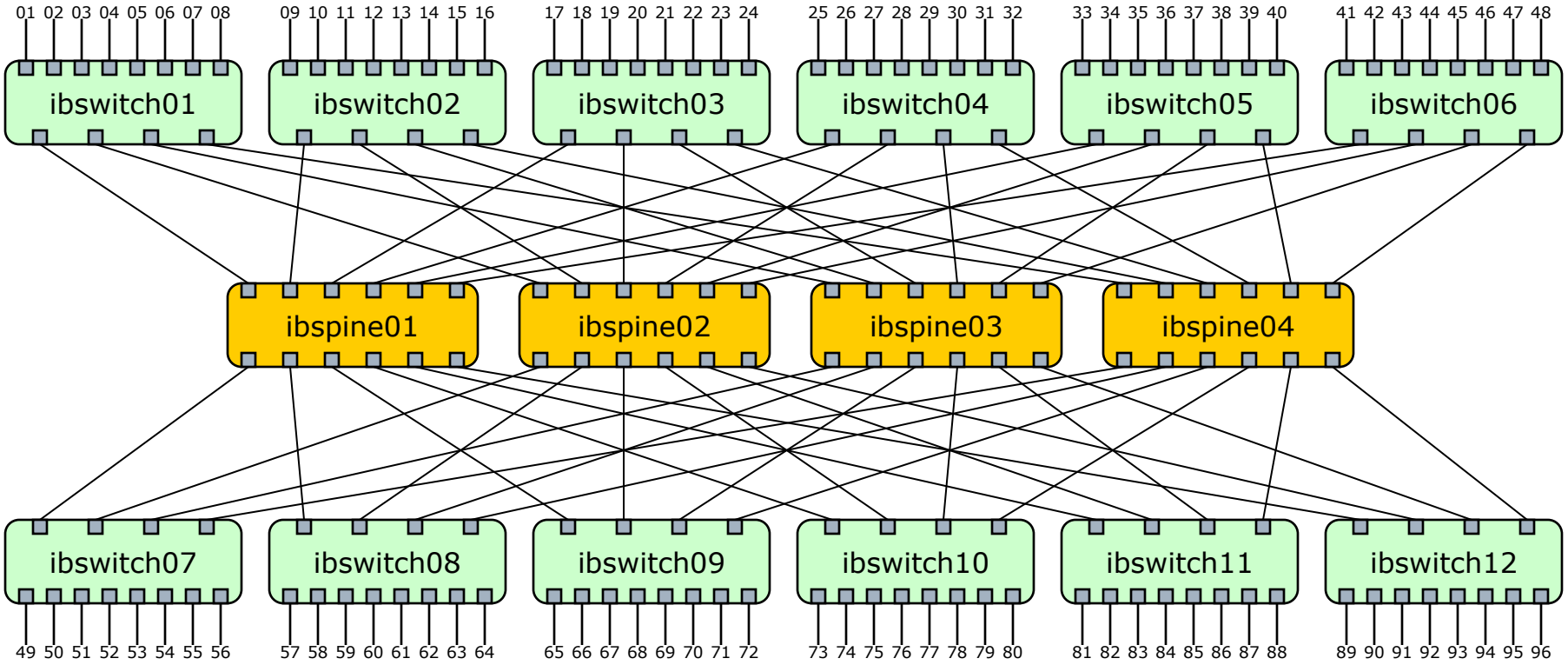
Multiple LIDs

- Problems with single LID
  - there is always the only route between two nodes
  - no possibility to optimize transfers, doing routing between nodes via different spines ourselves
- Solution – LMC (LID Mask Control)
  - When LMC=4, lower 4 bits of host LID are reserved for routing
  - Subnet Manager can assign up to 16 routes to that node
  - Not always smoothly works
- Problem – scan all these routes
  - 8h x 16 = ~5 days
- Solution
  - modified version of *ibtracert* program with cashing of intermediate tables and excluding scan of similar routes
  - reduce scanning time to about 4 minutes

Routing–aware schedule

Main motivation:
- avoid congestions in all links at any moment of time

Two approaches to build such schedule:
- select route with unused link (better for small number of nodes)
- using regular structure of the network (better for bigger number of nodes)



ib-test application

- implemented with dabc2 (beta quality)

- dabc2
  - multithreaded application environment
  - classes to working with OFED verbs
  - udp-based command channel between nodes
  - configuration and startup of multi-node application
  - used as DAQ framework in many CBM beam tests

- ib-test
  - master-slave architecture
  - all actions are driven by master node
  - all-to-all connectivity
  - time synchronization with master
  - scheduled transfers with specified rate
  - transfers statistic

https://subversion.gsi.de/dabc/trunk/applications/ib-test

Time synchronization

- Scheduled transfer means submit send/receive operations at predefined time
- With $2x10^9$ Bytes/s and 1MB buffers one requires time precision of several μs
- One can use small IB round-trip packet, which should have very small latency
- On LOEWE-CSC cluster such round-trip packet takes about 3.5 μs. Measuring time on both nodes, one can calculate time shift and compensate it
- Excluding 30% of max. variation due to system activity, one can achieve precision below 1 μs

all-to-all IB performance

- In May 2011 before planned cluster shutdown I get about 4 hours for IB performance tests
- By CPU load 774 nodes were selected for tests
- different transfer rates were tested
  - 1.5×10^9 B/s/node – 0.8% packets skipped
  - 1.6×10^9 B/s/node – 4.4% packets skipped
  - 0.5×10^9 B/s/node – with skip disabled
- Means at maximum: 1.25×10^{12} B/s
- Main problem here: skipped transfers and how one could avoid them

Skipped transfers

- Already with first tests on 150-200 nodes I encounter a problem, that some transfers were not completed in reasonable time (~100 ms)
- Simple guess – there was other traffic
  - first tests were performed parallel to other jobs
- Very probable, that physical-layer errors and many retransmission also causing that problem
- To coupe with such situation, simple skip was implemented – when several transfers to the some node are hanging, following transfers just skipped
- Would it be better to use unreliable transport here?

IB + GPU

- InfiniBand is just transport
  - one need to deliver data to computing entity for data analysis and selection
- All LOEWE-CSC nodes equipped with GPUs
  - use GPU as data sink for transfer
  - use GPU also as source of data
- GPU -> host -> IB -> host -> GPU
- With small 4x nodes setup
  - 1.1 GB/s/node for all-to-all traffic pattern

Outlook and plans

- MPI performance
- Use of unreliable connections (UC)
- RDMA to GPU memory
  - GPU -> IB -> GPU?
  - NVIDIA GPUDirect?
- Multicast performance/reliability
- Subnet manager control

Conclusion

- One must take into account IB fabric topology
- Only with scheduling one could achieve 70-80% of available bandwidth
- Nowadays IB fulfill CBM requirements
- A lot of work need to be done before real system will run

Wishlist for Mellanox

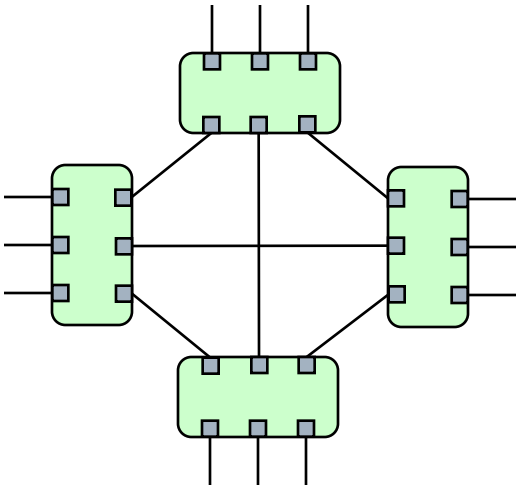
- By-time execution
  - perform operation not immediately but at specified time
- operation cancellation
  - how one could remove submitted operation from the queues
- No LMC for switch ports
  - waste of very limited address space
- RDMA-completion signaling for slave side
- How 36-port switch build inside?

Thank you!

BACKUP SLIDES

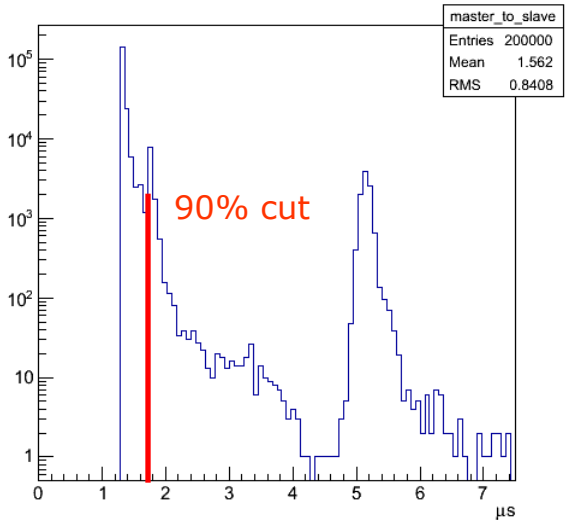
Topology limitations

- Single hop
  - 36 nodes
- Two hops (not a CBB)
  - 18 x 18 = 324 nodes
- Three hops
  - 36 x 18 = 648 nodes (fat tree)
  - 36 x 24 = 864 nodes (½ fat tree)
- Four hops (not a CBB)
  - 36 x 36 = 1296 nodes
  - 72 x 72 = 5184 nodes
  - …
- Five hops
  - 72 x 36 = 2592 nodes (fat tree)
  - 72 x 48 = 3458 nodes (½ fat tree)
  - …

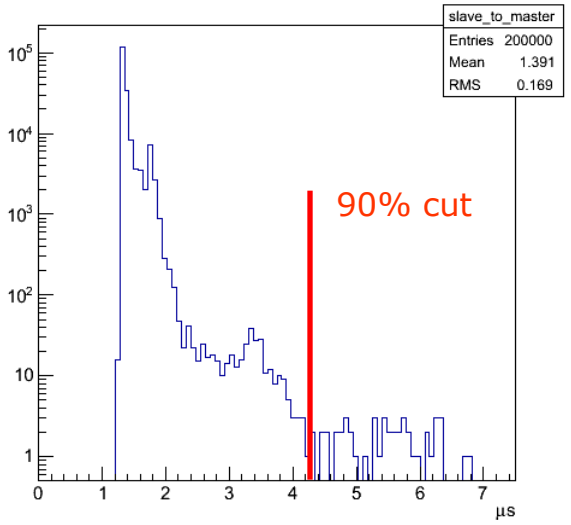


- Practical limitation
  - only ~48000 LIDs in subnet, including all switch ports

Latency distribution



A)



B)

Distribution of transfer (packet latency) time for master-to-slave (A) and slave-to-master (B) communication

Time sync with many nodes

```
09:40:37 133.348335 Round trip to  1:  3.50 microsec
09:40:37 133.348366    Master -> Slave  :  1.57  +- 0.03 (max = 10.45 min =  1.47)
09:40:37 133.348387    Slave  -> Master :  1.93  +- 0.04 (max = 10.20 min =  1.85)
09:40:37 133.348403    GET: Shift =  0.18
09:40:37 133.350986 Round trip to  2:  3.27 microsec
09:40:37 133.351008    Master -> Slave  :  2.08  +- 0.03 (max = 10.75 min =  2.03)
09:40:37 133.351026    Slave  -> Master :  1.20  +- 0.04 (max = 10.37 min =  1.14)
09:40:37 133.351087    GET: Shift = -0.44
09:40:37 133.353597 Round trip to  3:  3.29 microsec
09:40:37 133.353620    Master -> Slave  :  1.61  +- 0.04 (max =  9.27 min =  1.57)
09:40:37 133.353638    Slave  -> Master :  1.68  +- 0.03 (max =  9.45 min =  1.63)
09:40:37 133.353654    GET: Shift =  0.03
…
09:40:39 135.309515 Round trip to 721:  3.49 microsec
09:40:39 135.309535    Master -> Slave  :  2.07  +- 0.04 (max = 12.46 min =  2.02)
09:40:39 135.309555    Slave  -> Master :  1.42  +- 0.05 (max = 10.39 min =  1.37)
09:40:39 135.309571    GET: Shift = -0.33
09:40:39 135.312352 Round trip to 722:  3.63 microsec
09:40:39 135.312380    Master -> Slave  :  1.70  +- 0.03 (max =  9.69 min =  1.59)
09:40:39 135.312420    Slave  -> Master :  1.93  +- 0.03 (max = 11.06 min =  1.87)
09:40:39 135.312437    GET: Shift =  0.12
09:40:39 135.312458 GET shift = 0.090629 +- 0.208242 (min = -2.035549, max = 0.929582)
09:40:39 135.312476 Tyme sync done in 1.9669 sec
```

Clocks skew after ~60 sec: 0.09±0.21 μs