"GPU in HEP: online high quality trigger processing"

ISOTDAQ

20.2.2018

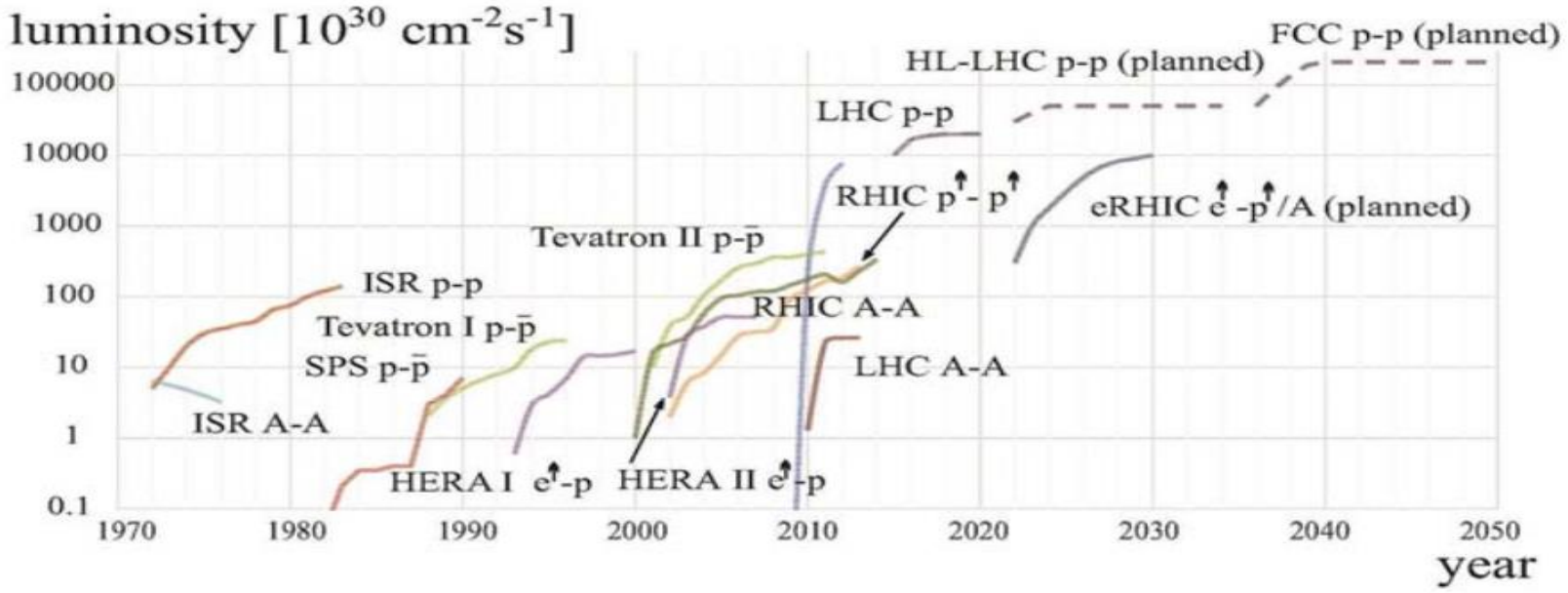
Vienna

Gianluca Lamanna (Univ.Pisa & INFN)

INFN

The World in 2035

The problem in 2035



- FCC (Future Circular Collider) is only an example
  - Fixed target, Flavour factories, … the physics reach will be defined by trigger!
- What the triggers will look like in 2035?

The trigger in 2035…

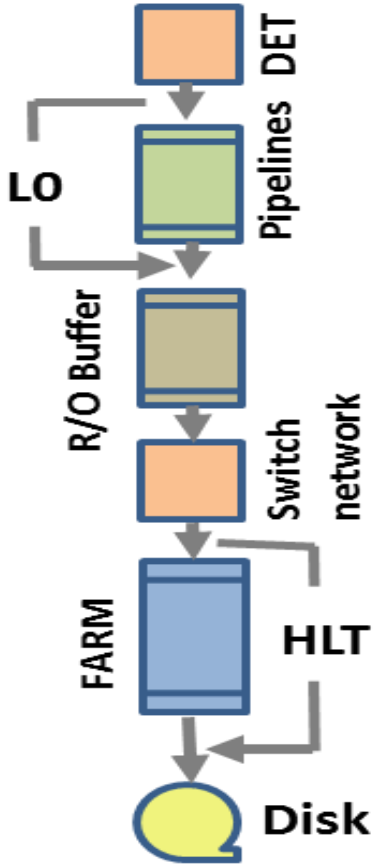
- … will be similar to the current trigger…
  - High reduction factor
  - High efficiency for interesting events
  - Fast decision
  - High resolution
- …but will be also different…
  - The higher background and Pile Up will limit the ability to trigger on interesting events
  - The primitives will be more complicated with respect today: tracks, clusters, rings

The trigger in 2035…

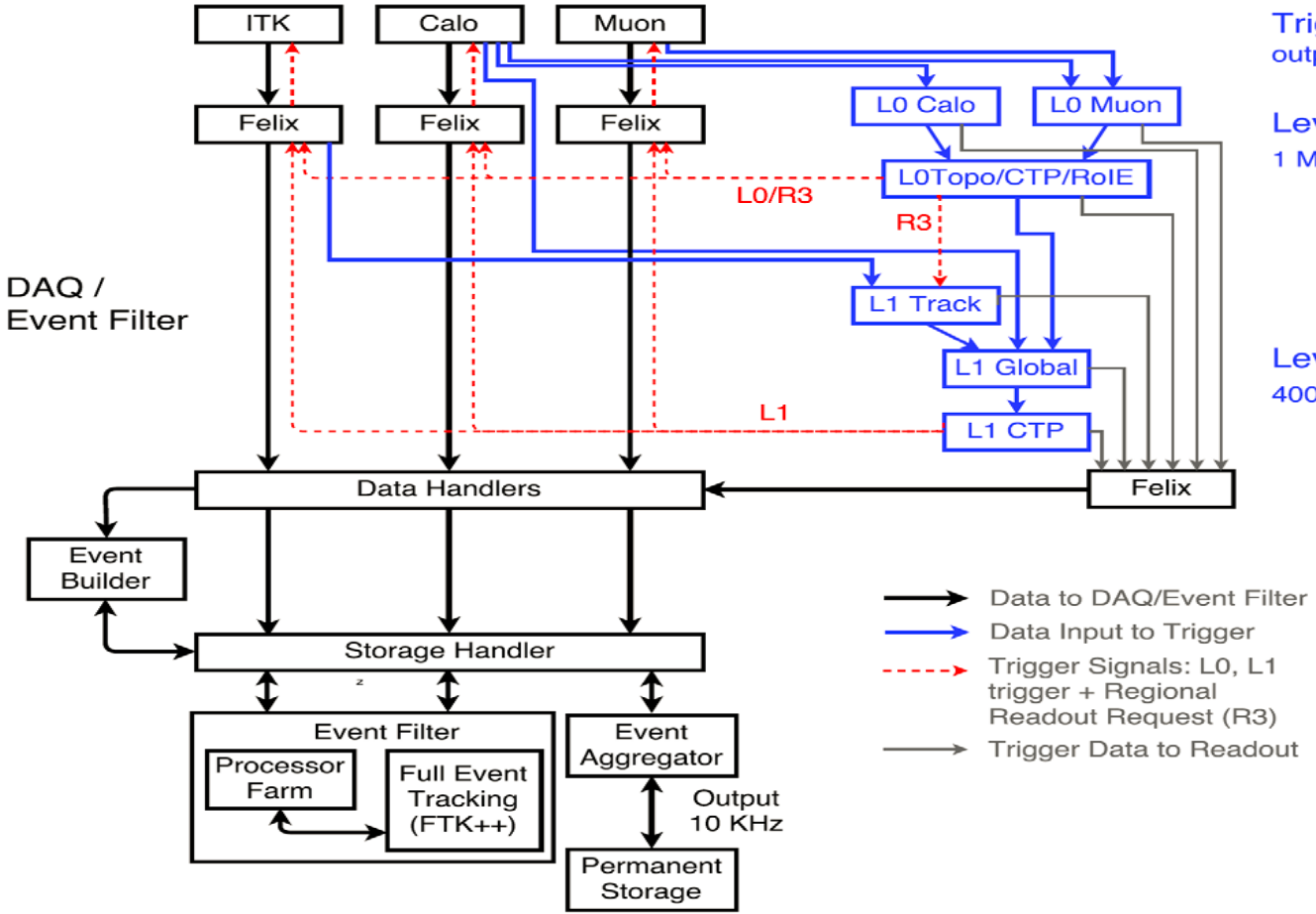
- Higher energy
  - Resolution for high pt leptons → high-precision primitives
  - High occupancy in forward region → better granularity
- Higher luminosity
  - track-calo correlation
  - Bunch crossing ID becomes challenging, pile up
- All of these effects go in the same direction
  - More resolution & more granularity → more data & more processing

Classic trigger in the future?

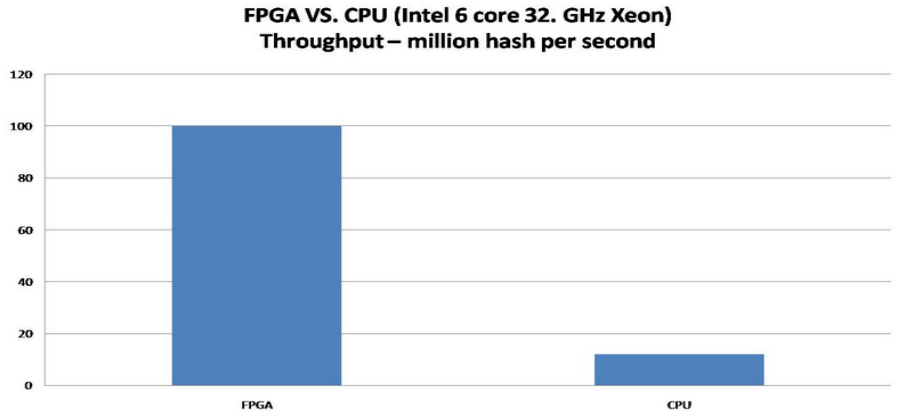
- Is a traditional "pipelined" trigger possible?
  - Yes and no
  - Cost and dimension
  - Getting all data in one place
    - New links -> data flow
    - No "slow" detectors can participate to trigger (limited latency)
  - Pre-processing on-detector could help
    - FPGA: not suitable for complicated processing
    - Software: commodity hw
- Main limitation: high quality trigger primitives generation on detector (processing)

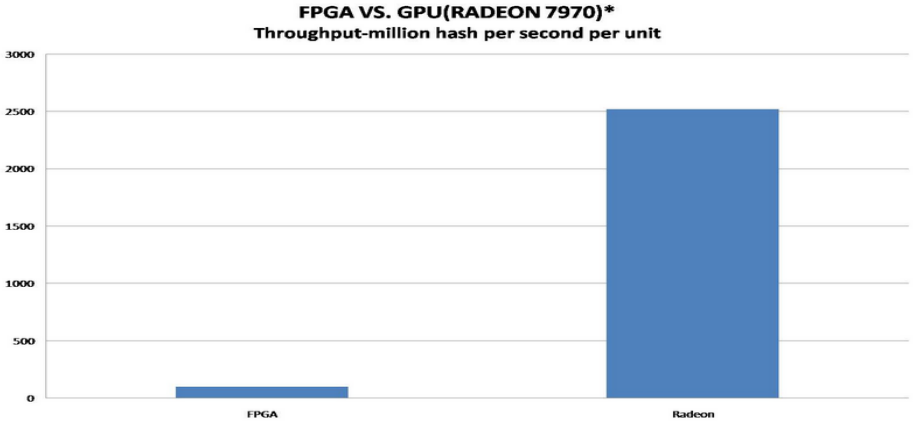


Pipelined trigger in 2025…



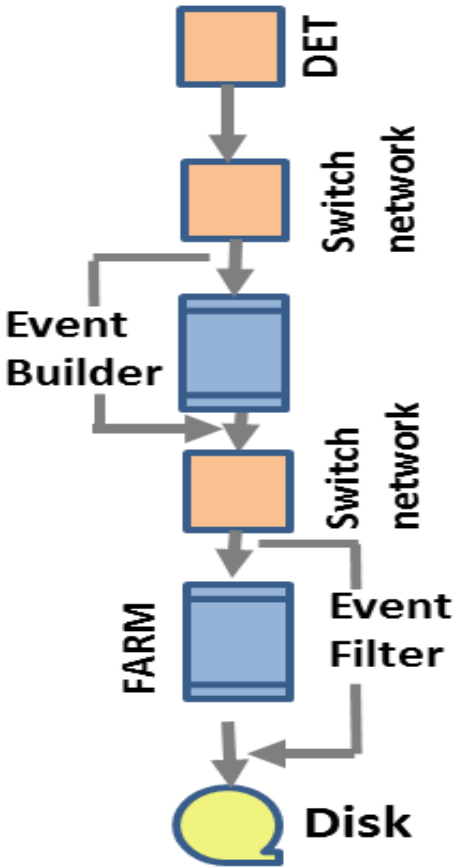
Classic Pipeline: Processing



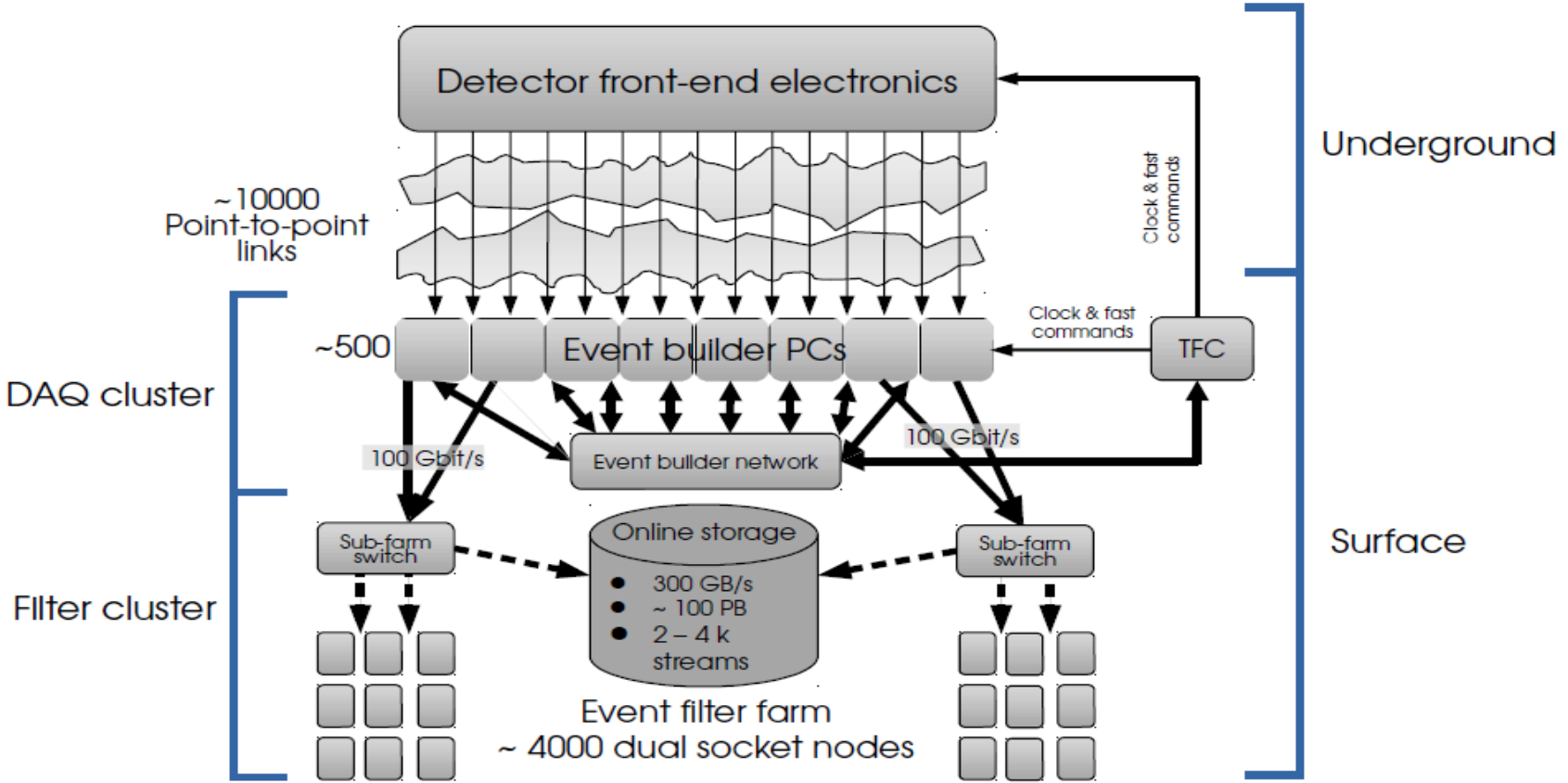


- The performances of FPGA as computing device depends on the problem
- The increasing in computing capability in "standard" FPGA is not as fast as CPU
- This scenario would change in the future with the introduction of new FPGA+CPU hybrid devices

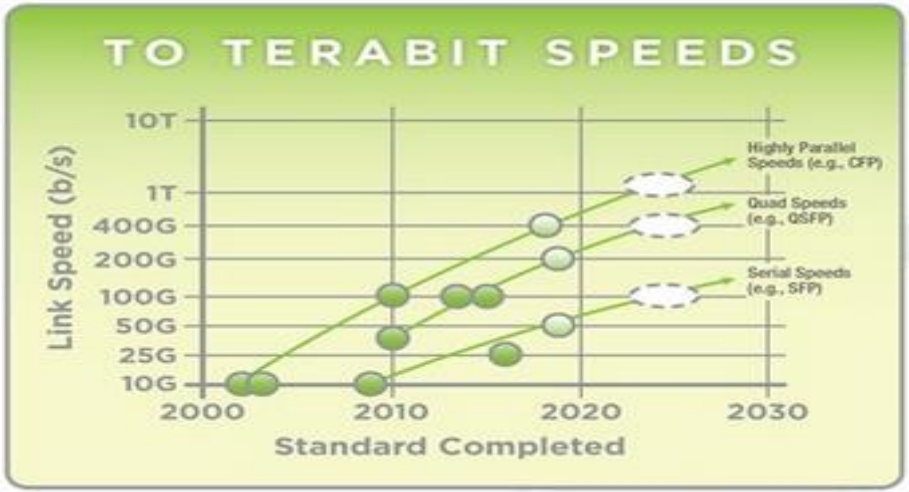
Triggerless?

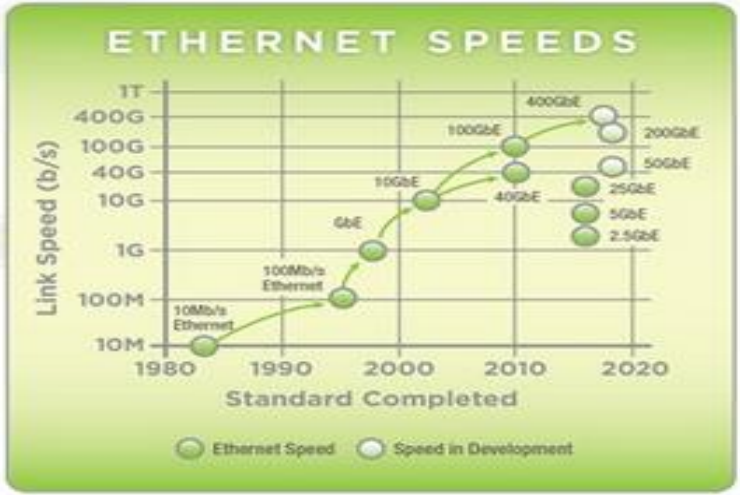


- Is it possible to bring all data on PCs?
  - LHCb: yes in 2020
    - 40 MHz readout, 30 Tb/s data network, 4000 cores, 8800 links
    - No in 2035: track+calo=2PB/s + 5 PB/s ev.building (for comparison largest Google data center = 0.1 PB/s)
  - CMS & ATLAS: probably no (in 2035)
    - 4 PB/s data, 4M links, x10 in performance for switch, x2000 computing
- Main limitation: data transport


Detector front-end electronics
~10000
Point-to-point
links
Clock & fast
commands
Underground
~500
Event builder PCs
Clock & fast
commands
TFC
DAQ cluster
100 Gbit/s
Event builder network
100 Gbit/s
Sub-farm
switch
Online storage
300 GB/s
~ 100 PB
2 – 4 k
streams
Sub-farm
switch
Surface
Filter cluster
Event filter farm
~ 4000 dual socket nodes

Triggerless: Data Links





- The links bandwidth is steadily increasing
- But the power consumption is not compatible with HEP purposes (rad hard serializers):
  - e.g. lpGBT is 500mW per 5Gb/s
  - 4M links → 2 MW only for links on detector
- Nowadays standard market is not interested in this application.

Example: an alternative approach

Triggerless:

Focus on Data Links

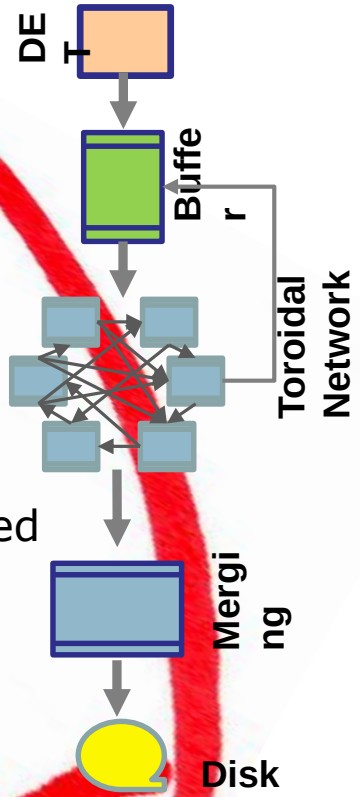
Classic pipeline:

Focus on On-detector processing

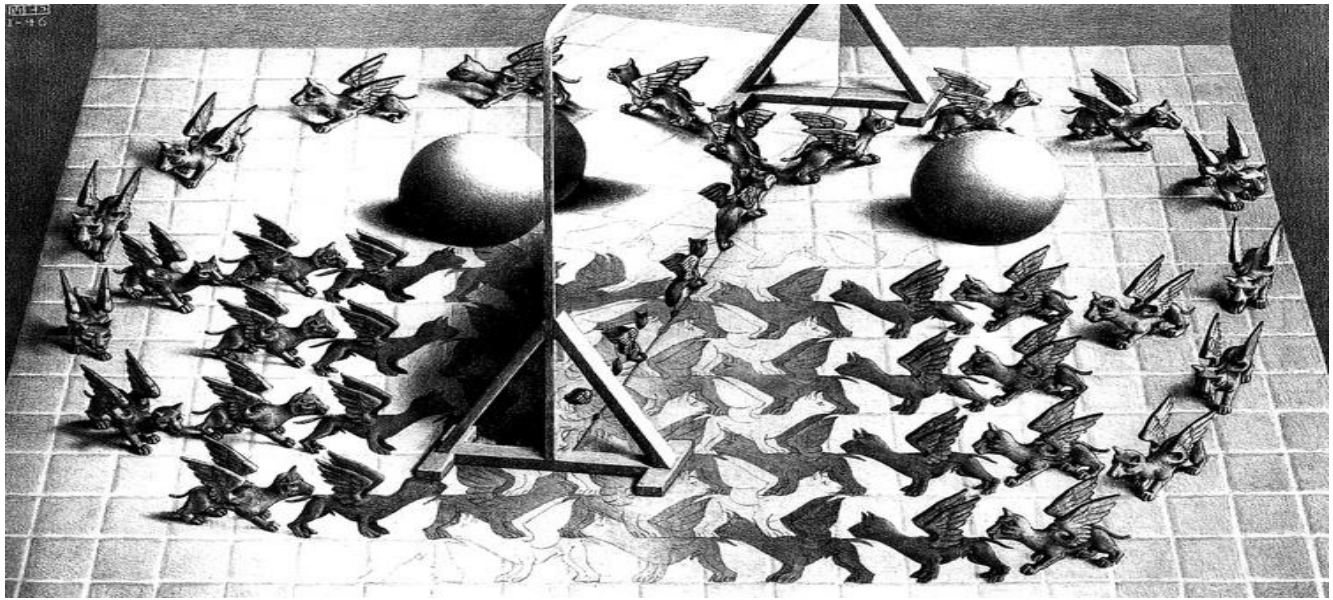
High Latency Trigger:

- Heterogeneous computing nodes
- Toroidal network
- Time multiplexed trigger
- Trigger implemented in software
- Large buffers

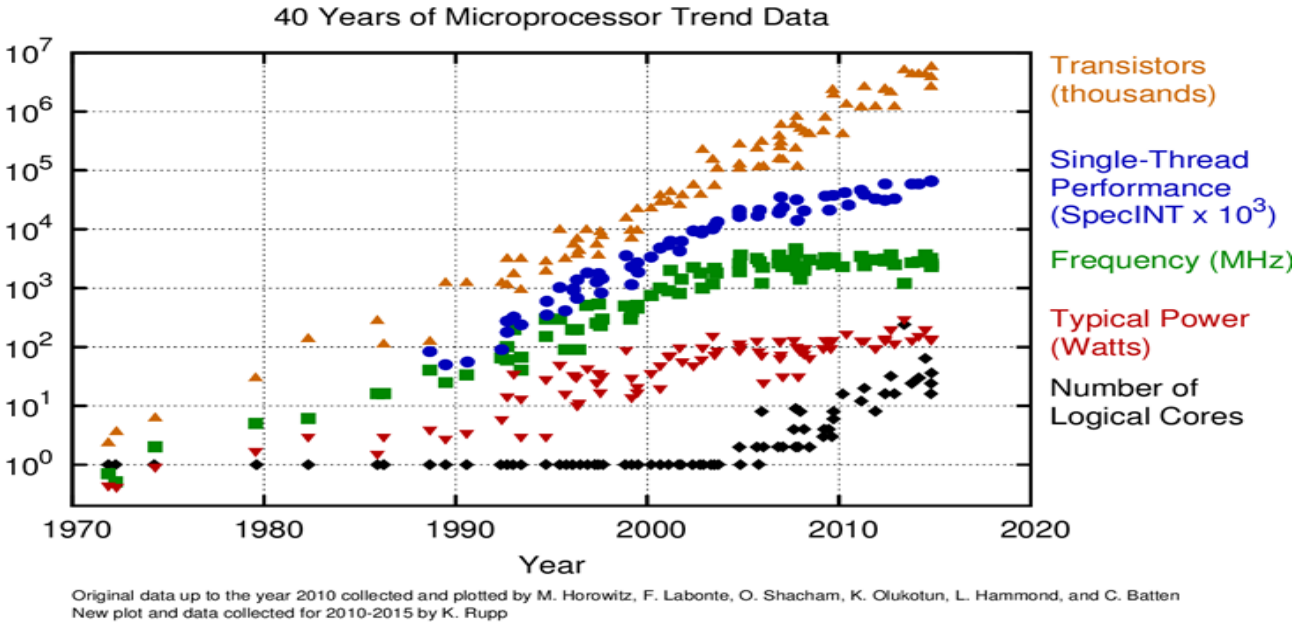
Focus on On-detector Buffers



GPU: Graphics Processing Units



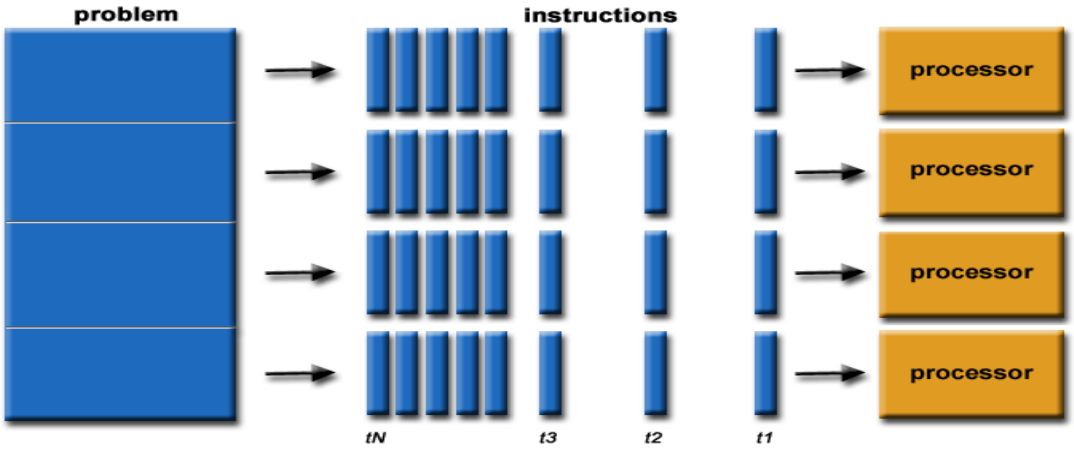
Moore's Law



- Moore's law: "The performance of microprocessors and the number of their transistors will double every 18 months"
- The increasing of performance is related to the clock
- Faster clock means higher voltage → power wall

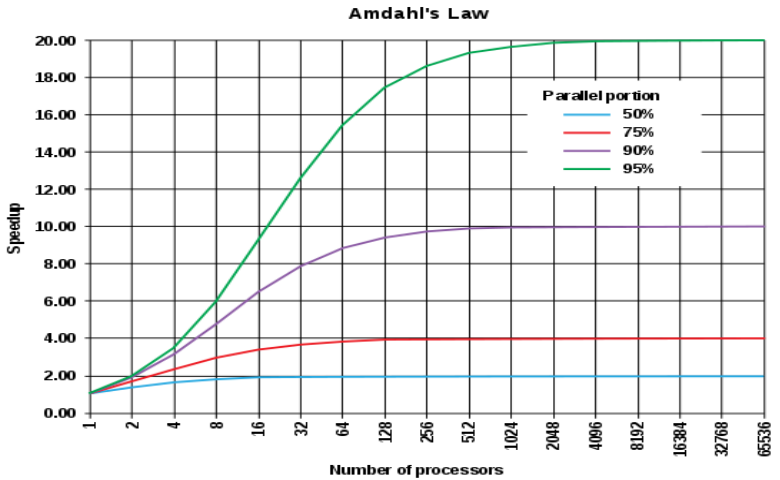
Parallel programming

- Parallel computing is no longer something for SuperComputers
  - All the processors nowadays are multicores
- The use of parallel architectures is mainly due to the physical constraints to frequency scaling



Parallel computing

- Several problems can be split in smaller problems to be solved concurrently
- In any case the maximum speed-up is not linear , but it depends on the serial part of the code (→ Amdahls's law)
- The situation can improve if the amount of parallelizable part depends on the resources (→ Gustafson's Law)



$$S_{latency} = \frac{1}{1 - p + \frac{p}{s}}$$

$$S_{latency} = 1 - p + sp$$

- 

- al
- n

helps a lot.

What are the GPUs?

- The technical definition of a GPU is "a single-chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is capable of processing a minimum of 10 million polygons per second."
- The possibility to use the GPU for generic computing (GPGPU) has been introduced by NVIDIA in 2007 (CUDA)
- In 2008 OpenCL: consortium of different firms to introduce a multi-platform language for manycores computing.

(1997)

(2016)

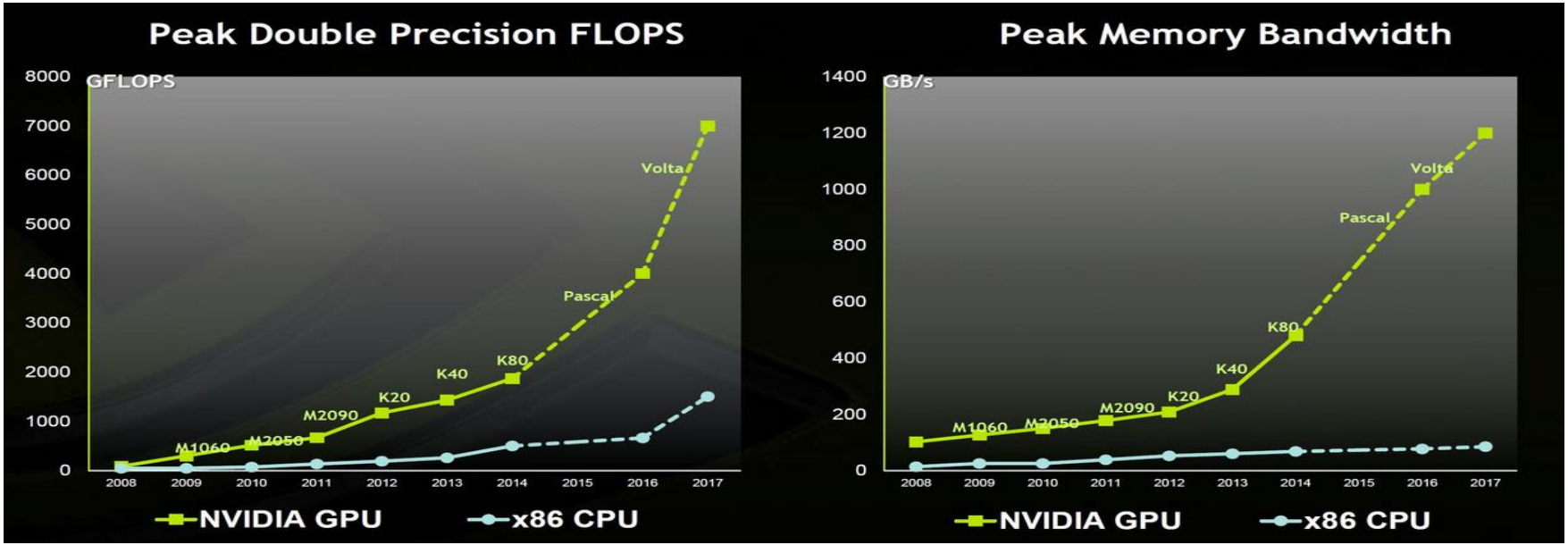
- GPU is a way to cheat the Moore's law

	"Fermi"	"Fermi"	"Kepler"	"Kepler"	"Maxwell"	"Pascal"
Tesla GPU	**GF100**	**GF104**	**GK104**	**GK110**	**GM200**	**GP100**
Compute Capability	2.0	2.1	3.0	3.5	5.3	6.0
Streaming Multiprocessors (SMs)	16	16	8	15	24	56
FP32 CUDA Cores / SM	32	32	192	192	128	64
FP32 CUDA Cores	512	512	1536	2880	3072	3584
FP64 Units	–	–	512	960	96	1792
Threads / Warp	32	32	32	32	32	32
Max Warps / Multiprocessor	48	48	64	64	64	64
Max Threads / Multiprocessor	1536	1536	2048	2048	2048	2048
Max Thread Blocks / Multiprocessor	8	8	16	16	32	32
32-bit Registers / Multiprocessor	32768	32768	65536	65536	65536	65536
Max Registers / Thread	63	63	63	255	255	255
Max Threads / Thread Block	1024	1024	1024	1024	1024	1024
Shared Memory Size Configurations	16 KB	16 KB	16 KB	16 KB	96 KB	64 KB
	48 KB	48 KB	32 KB	32 KB		
			48 KB	48 KB		
Hyper-Q	No	No	No	Yes	Yes	Yes
Dynamic Parallelism	No	No	No	Yes	Yes	Yes
Unified Memory	No	No	No	No	No	Yes
Pre-Emption	No	No	No	No	No	Yes

computing power, with thousand of cores.

- Several applications in HPC, simulation, scientific computing…

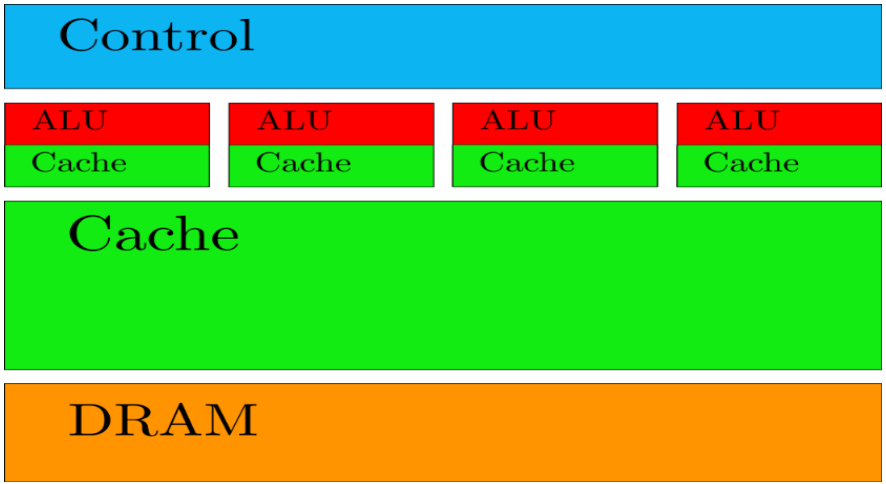
Computing power



Why?: CPU vs GPU

CPU

GPU

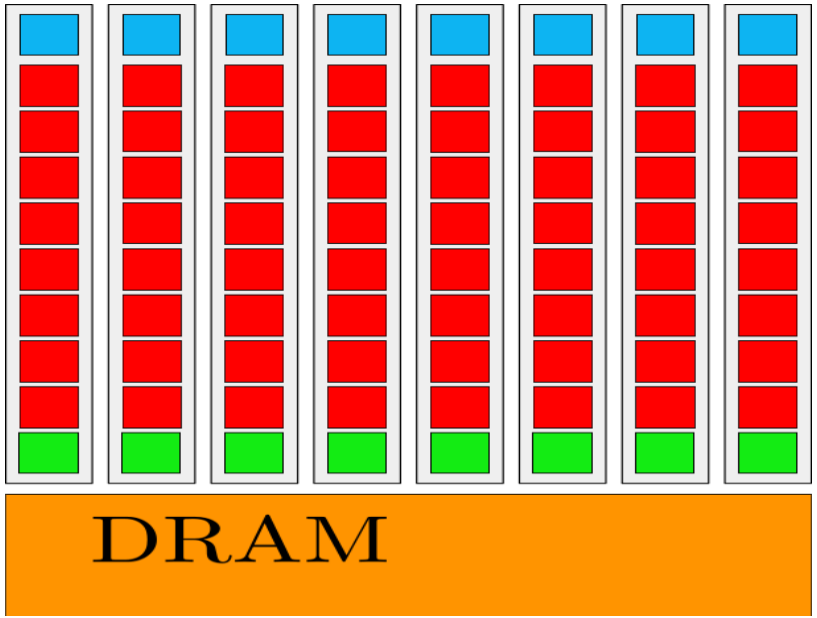


CPU: latency oriented design

- Multilevel and Large Caches
  - Convert long latency memory access
- Branch prediction
  - To reduce latency in branching
- Powerful ALU
- Memory management
- Large control part

GPU

- SIMT (Single instruction Multiple Thread) architecture
- SMX (Streaming Multi Processors) to execute kernels
- Thread level parallelism
- Limited caching
- Limited control
- No branch prediction, but branch predication

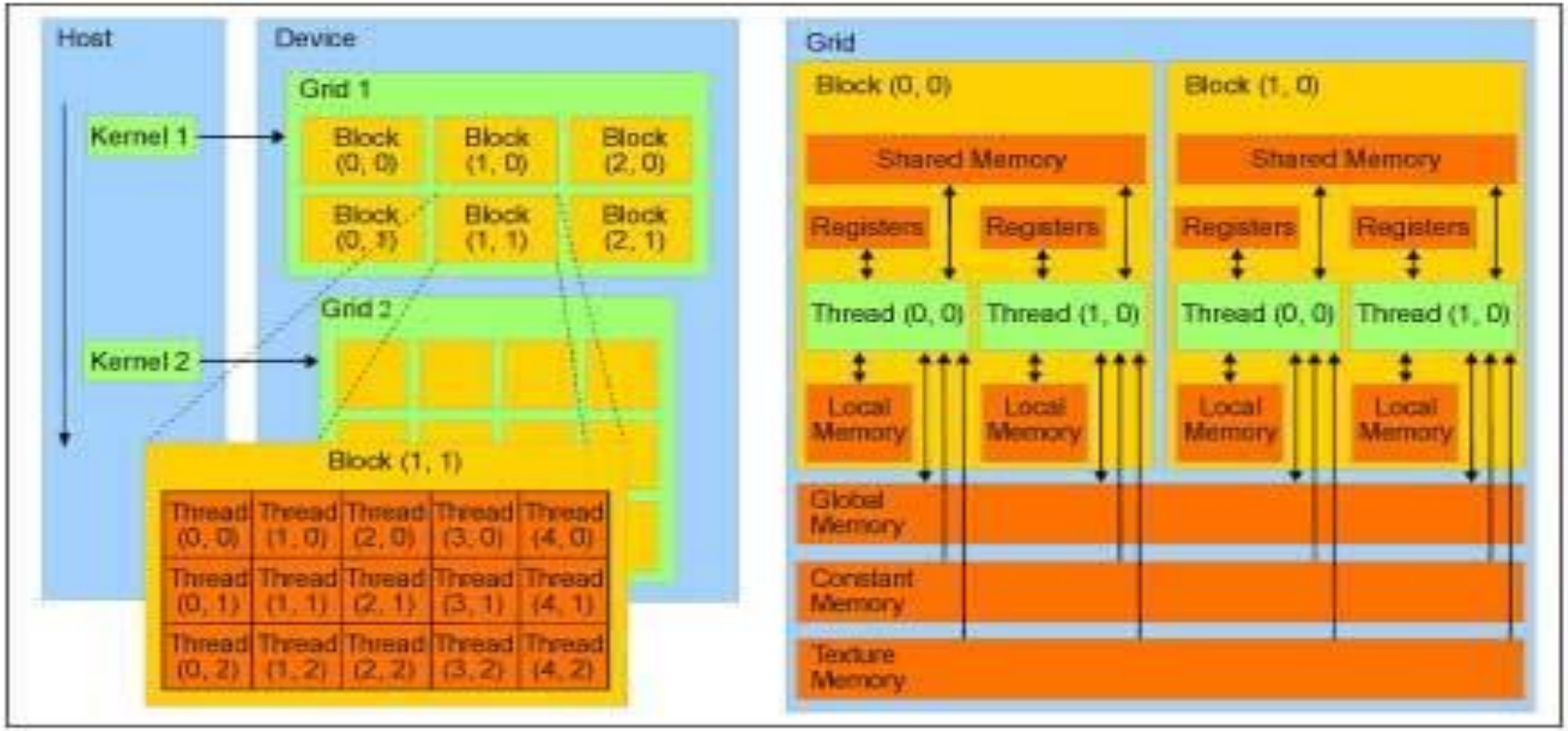


GPU: throughput oriented design

- The winning application uses both CPU and GPU
  - CPUs for sequential parts (can be 10X faster than GPU for sequential code)
  - GPUs for parallel part where throughput wins (can be 100X faster than CPU for parallel code)

- What is CUDA?
  - It is a set of C/C++ extensions to enable the GPGPU computing on NVIDIA GPUs
  - Dedicated APIs allow to control almost all the functions of the graphics processor
- Three steps:
  - 1) copy data from Host to Device
  - 2) copy Kernel and execute
  - 3) copy back results

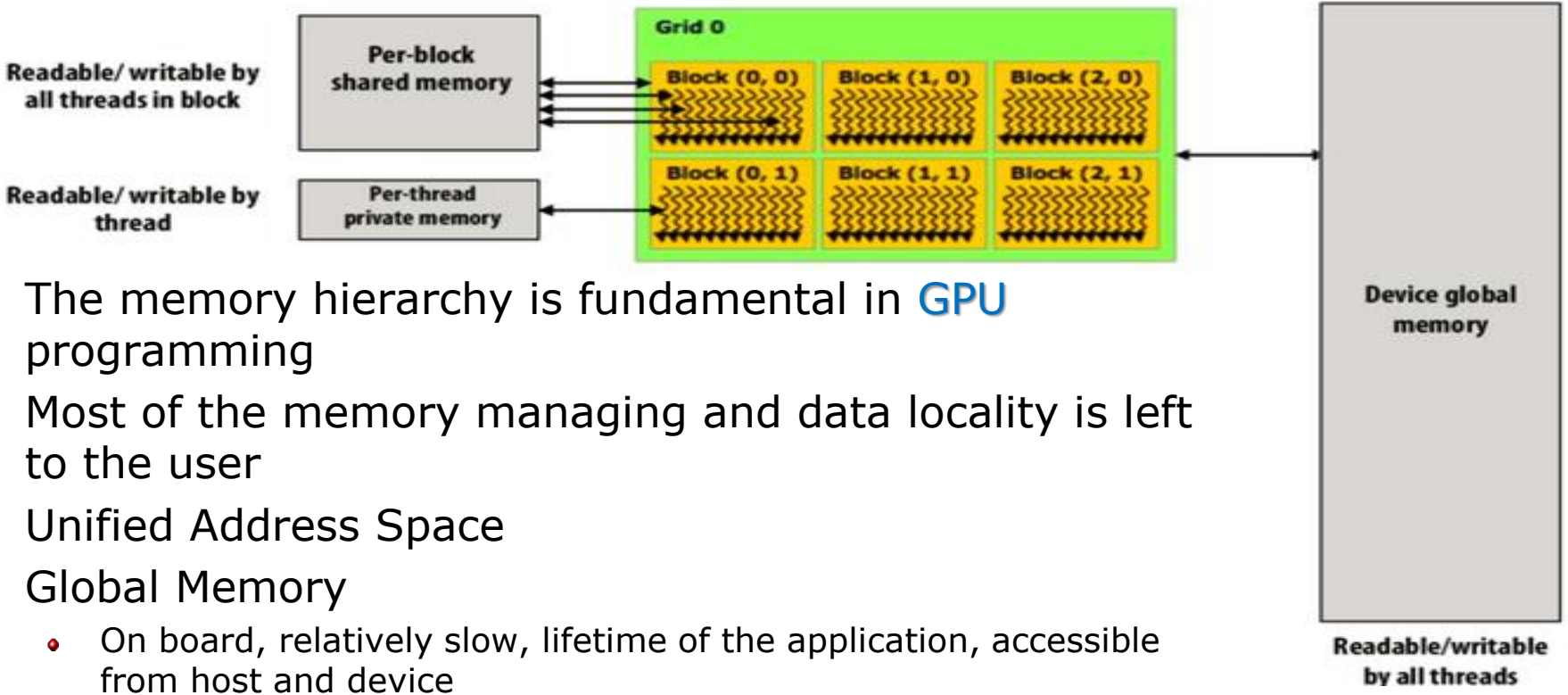
Grids, blocks and threads



launch time

Mapping on the hardware

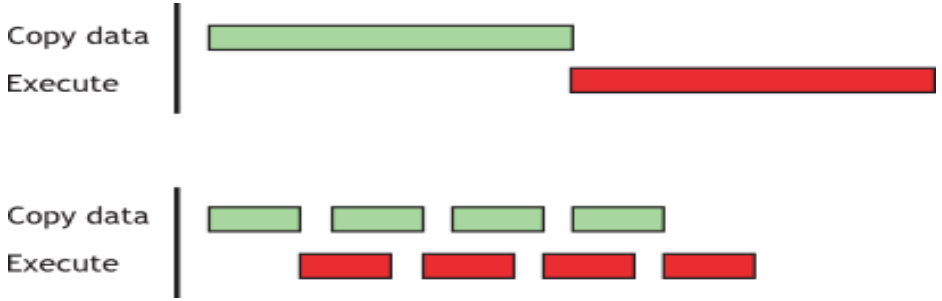
Memory



- The memory hierarchy is fundamental in GPU programming
- Most of the memory managing and data locality is left to the user
- Unified Address Space
- Global Memory
  - On board, relatively slow, lifetime of the application, accessible from host and device
- Shared memory/registers
  - On Chip, very fast, lifetime of blocks/threads, accessible from kernel only

Streams

- The main purpose of all the GPU computing is to hide the latency
- In case of multiple data transfer from host to device the asynchronous data copy and kernel execution can be superimposed to avoid dead time



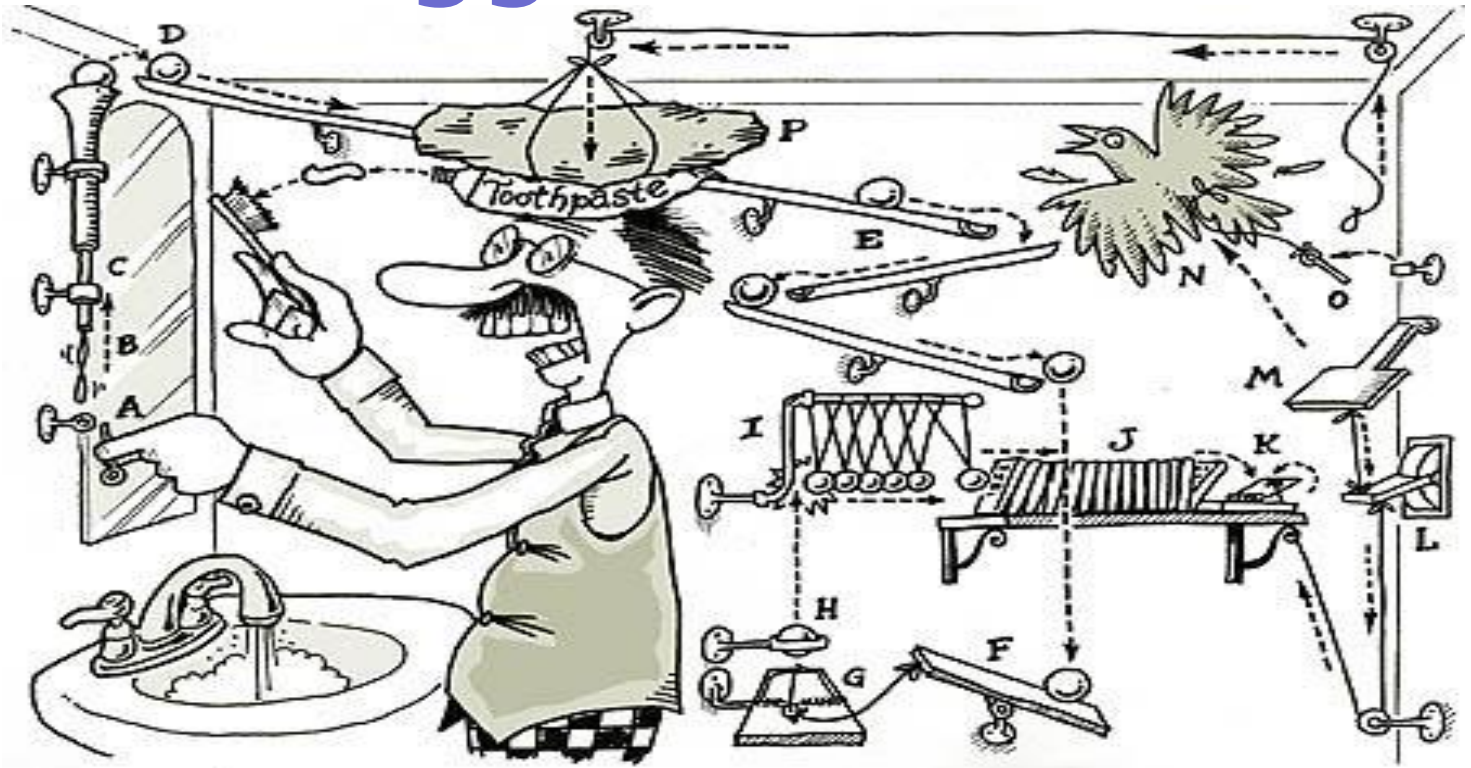

```
kernel<<< blocks, threads, bytes >>>();     // default stream
kernel<<< blocks, threads, bytes, 0 >>>(); // stream 0
```

Other ways to program GPU

- CUDA is the "best" way to program NVIDIA GPU at "low level"
- If your code is almost CPU or if you need to accelerate dedicated functions, you could consider to use
  - Directives (OpenMP, OpenACC, …)
  - Libraries (Thrust, ArrayFire,…)
- OpenCL is a framework equivalent to CUDA to program multiplatforms (GPU, CPU, DSP, FPGA,…).
  - NVIDIA GPUs supports OpenCL.

JOIN SECRET LAB!
This evening at 6.30 PM

Trigger and GPUs



Next generation trigger

- Next generation experiments will look for tiny effects:
  - The trigger systems become more and more important
- Higher readout band
  - New links to bring data faster on processing nodes
- Accurate online selection
  - High quality selection closer and closer to the detector readout
- Flexibility, Scalability, Upgradability
  - More software less hardware

Where to use GPU in the trigger?

- In High Level Trigger
  - It is the "natural" place. If your problem can be parallelized (either for events or for algorithm) you can gain factor on speed-up → smaller number of PC in Online Farm
  - Few examples in backup slides
- In Low Level Trigger
  - Bring power and flexibility of processors close to the data source → more physics

Different Solutions

- Brute force: PCs
  - Bring all data on a huge pc farm, using fast (and eventually smart) routers.
  - Pro: easy to program, flexibility; Cons: very expensive, most of resources just to process junk.

- Rock Solid: Custom Hardware
  - Build your own board with dedicated processors and links
  - Pro: power, reliability; Cons: several years of R&D (sometimes to re-rebuild the wheel), limited flexibility

- Elegant: FPGA
  - Use a programmable logic to have a flexible way to apply your trigger conditions.
  - Pro: flexibility and low deterministic latency; Cons: not so easy (up to now) to program, algorithm complexity limited by FPGA clock and logic.

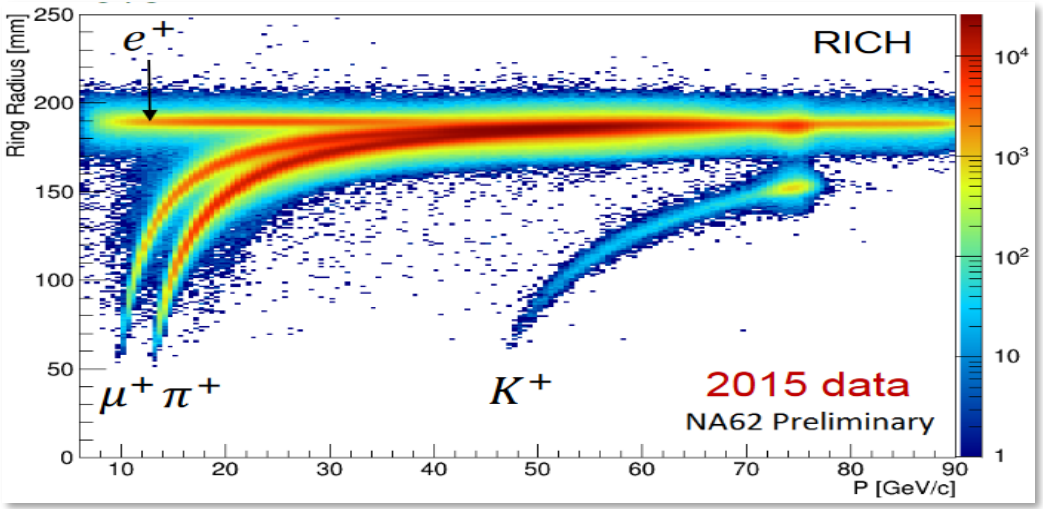
- Off-the-shelf: GPU
  - Try to exploit hardware built for other purposes continuously developed for other reasons
  - Pro: cheap, flexible, scalable, PC based. Cons: Latency

GPU in low level trigger

- **Latency:** Is the GPU latency per event small enough to cope with the tiny latency of a low level trigger system? Is the latency stable enough for usage in synchronous trigger systems?

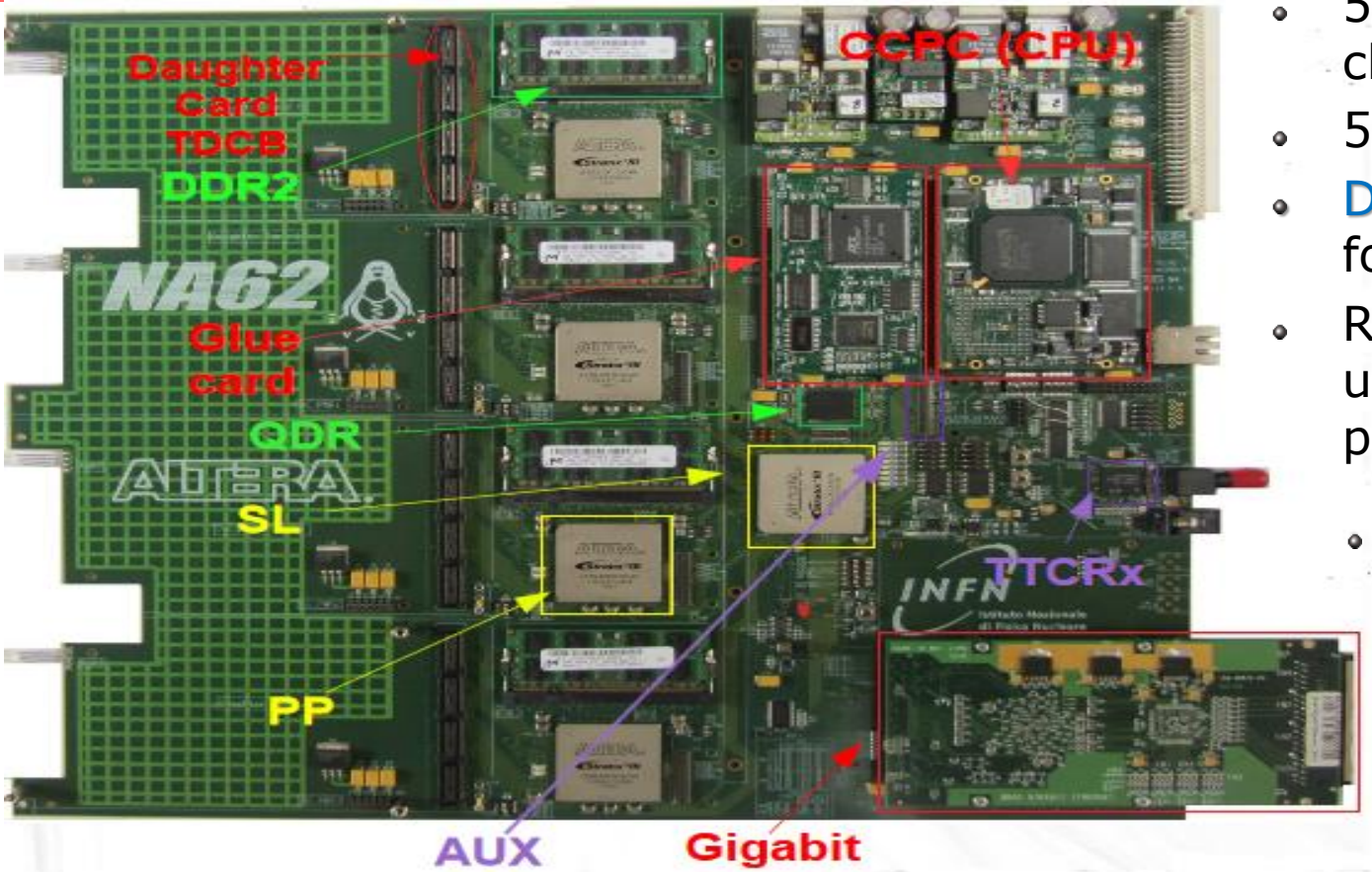
- **Computing power:** Is the GPU fast enough to take trigger decision at tens of MHz events rate?

Low Level trigger: NA62 Test bench



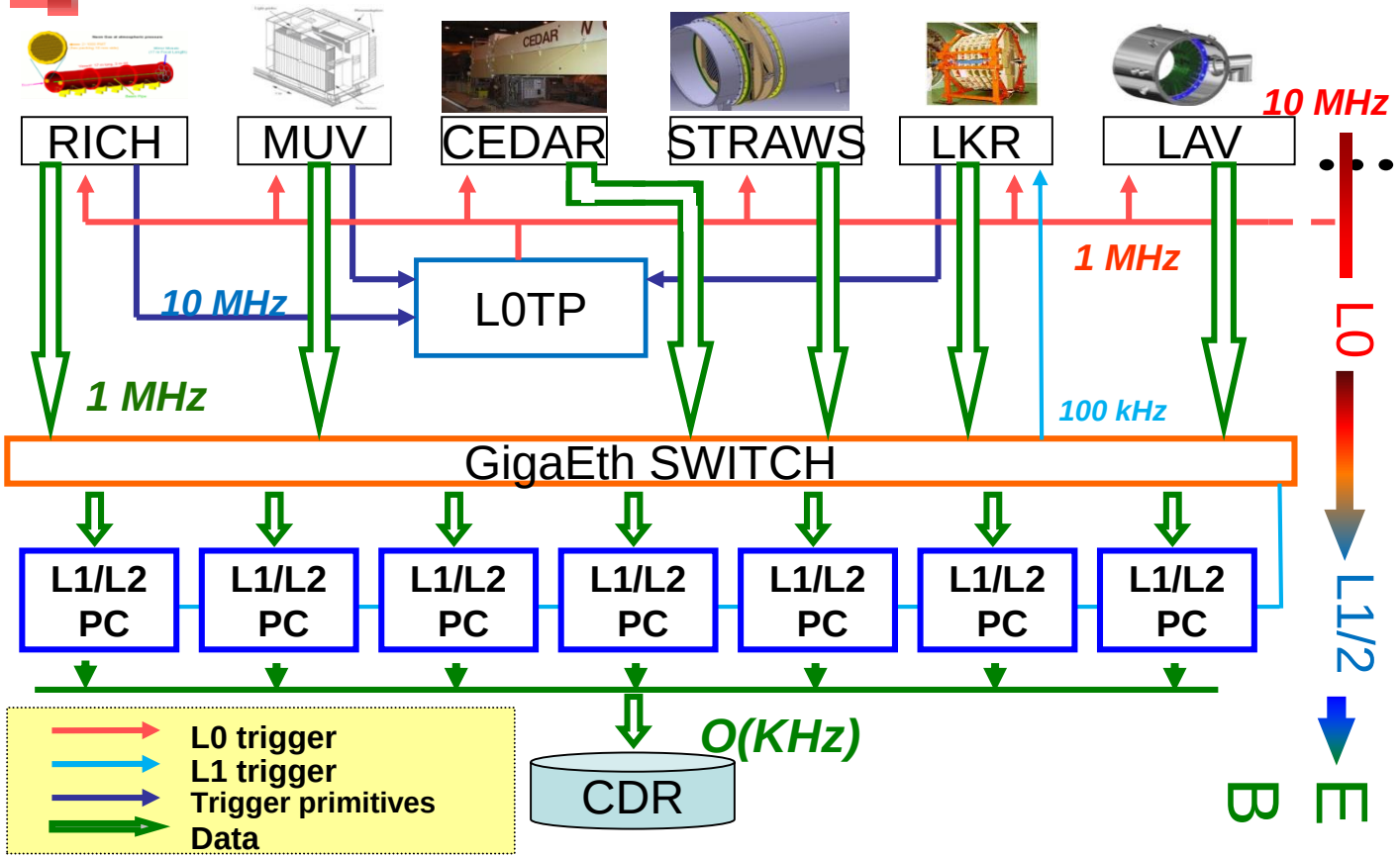
- NA62:
  - Fixed target experiment on SPS (slow extraction)
  - Look for ultra-rare kaon decays (K->pi nu nubar)
- RICH:
  - 17 m long, 3 m in diameter, filled with Ne at 1 atm
  - Reconstruct Cherenkov Rings to distinguish between pions and muons from 15 to 35 GeV
  - 2 spots of 1000 PMs each
  - Time resolution: 70 ps
  - MisID: 5×10^{-3}
  - 10 MHz events: about 20 hits per particle

TEL62



- 512 HPTDC channels
- 5 FPGAs
- DDR2 memories for readout buffer
- Readout data are used for trigger primitives
  - Data and primitives transmission through eth (UDP)

The NA62 "standard" TDAQ system

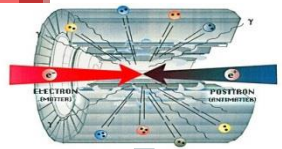


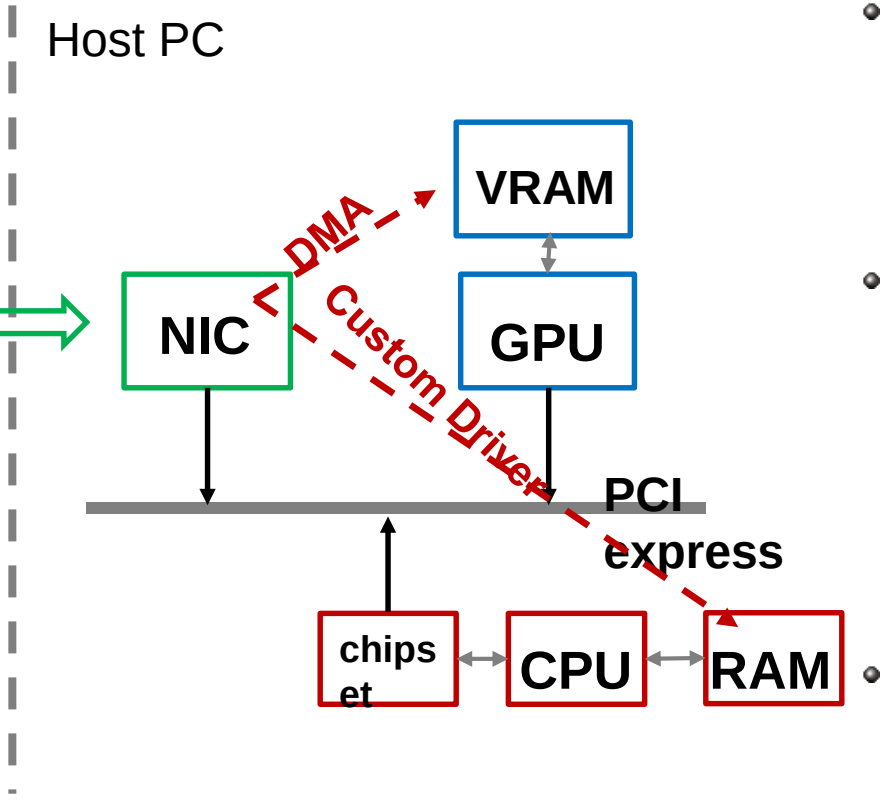
L0: Hardware synchronous level. 10 MHz to 1 MHz. Max latency 1 ms.

L1: Software level. "Single detector". 1 MHz to 100 kHz

L2: Software level. "Complete information level". 100 kHz to few kHz.

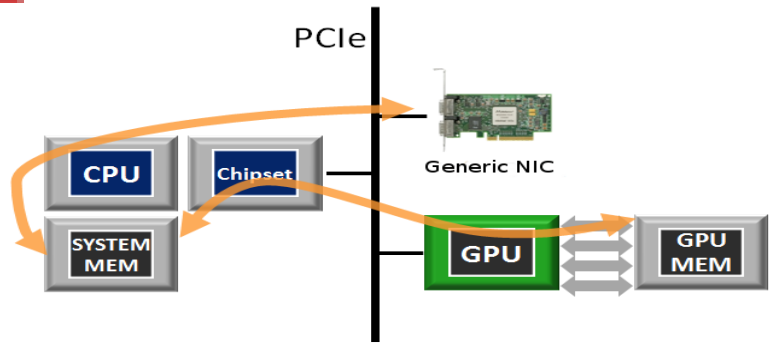
Latency: main problem of GPU computing

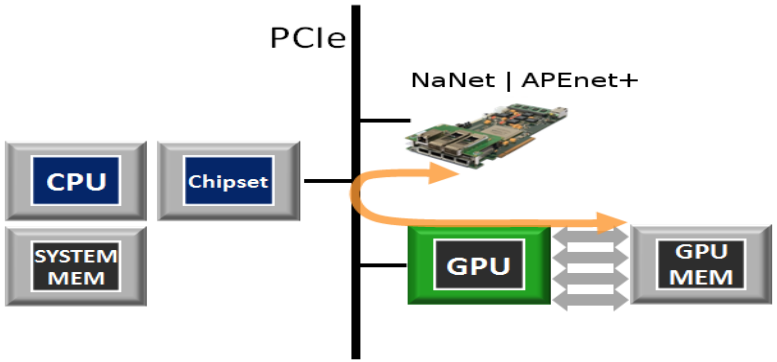




- Total latency dominated by double copy in Host RAM
- Decrease the data transfer time:
  - DMA (Direct Memory Access)
  - Custom manage of NIC buffers
- "*Hide*" some component of the latency optimizing the multi-events computing

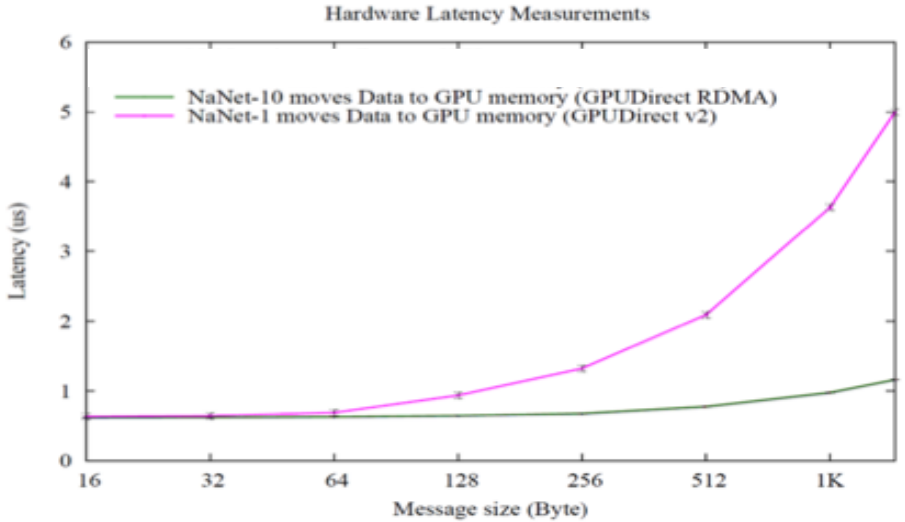
NaNet-10

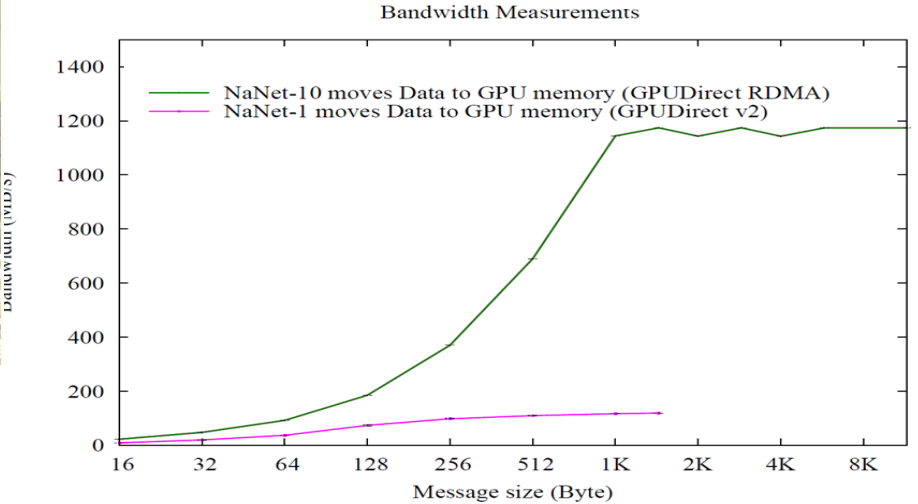




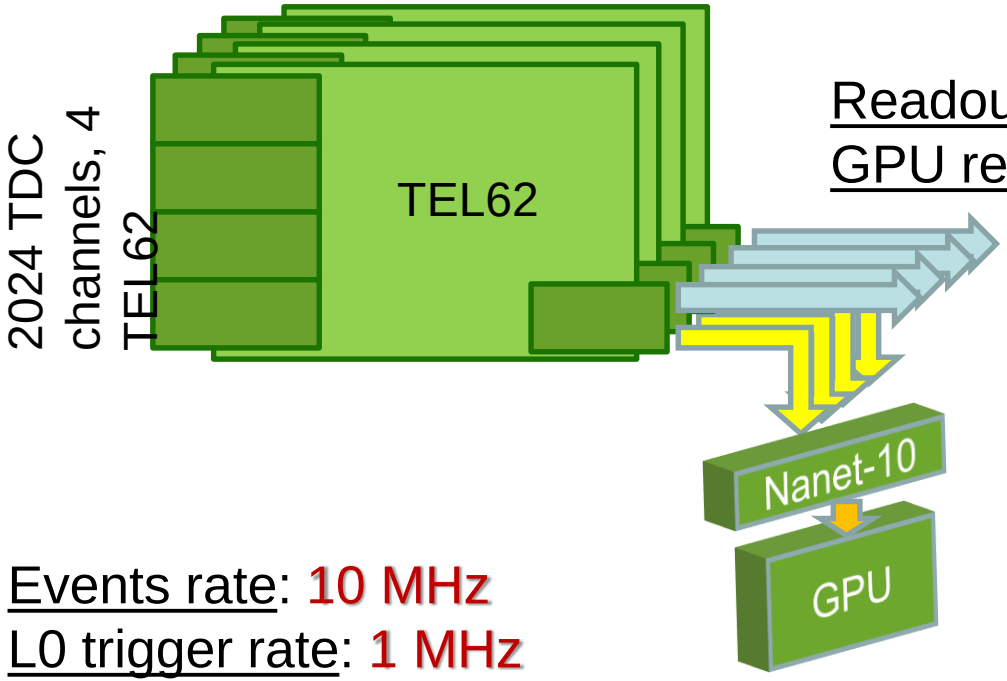
- ALTERA Stratix V dev board (TERASIC DE5-Net board)
  - PCIe x8 Gen3 (8 GB/s)
  - 4 SFP+ ports (Link speed up to 10Gb/s)
- GPUDirect /RDMA
- UDP offload support
- 4x10 Gb/s Links
- Stream processing on FPGA (merging, decompression, …)
- Working on 40 GbE (foreseen 100 GbE)

NaNet-10


Hardware Latency Measurements
NaNet-10 moves Data to GPU memory (GPUDirect RDMA)
NaNet-1 moves Data to GPU memory (GPUDirect v2)
Latency (us)
Message size (Byte)
0
1
2
3
4
5
6
16
32
64
128
256
512
1K


Bandwidth Measurements
NaNet-10 moves Data to GPU memory (GPUDirect RDMA)
NaNet-1 moves Data to GPU memory (GPUDirect v2)
Message size (Byte)
0
200
400
600
800
1000
1200
1400
16
32
64
128
256
512
1K
2K
4K
8K

NA62 GPU trigger system



Readout event: 1.5 kb (1.5 Gb/s)
GPU reduced event: 300 b (3 Gb/s)

8x1Gb/s links for data readout
4x1Gb/s Standard trigger primitives
4x1Gb/s GPU trigger

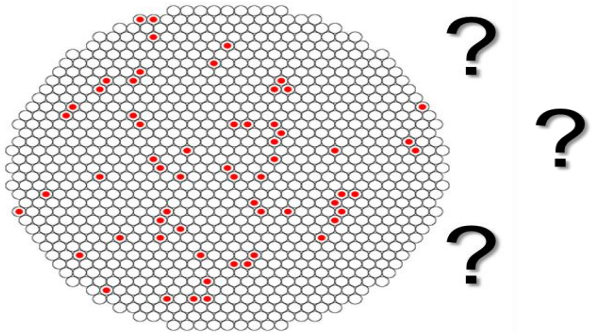
Events rate: 10 MHz
L0 trigger rate: 1 MHz
Max Latency: 1 ms
Total buffering (per board): 8 GB
Max output bandwidth (per board): 4 Gb/s

GPU NVIDIA K20:

- 2688 cores
- 3.9 Teraflops
- 6GB VRAM
- PCI ex.gen3
- Bandwidth: 250 GB/s

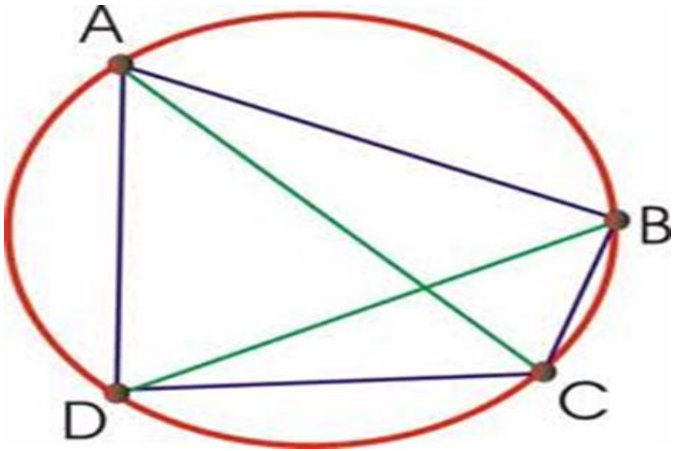
Ring fitting problem

- Trackless
  - no information from the tracker
  - Difficult to merge information from many detectors at L0
- Fast
  - Not iterative procedure
  - Events rate at levels of tens of MHz
- Low latency
  - Online (synchronous) trigger
- Accurate
  - Offline resolution required



- Multi rings on the market:
  - With seeds: Likelihood, Constrained Hough, …
  - Trackless: fiTQun, APFit, possibilistic clustering, Metropolis-Hastings, Hough transform, …

Almagest



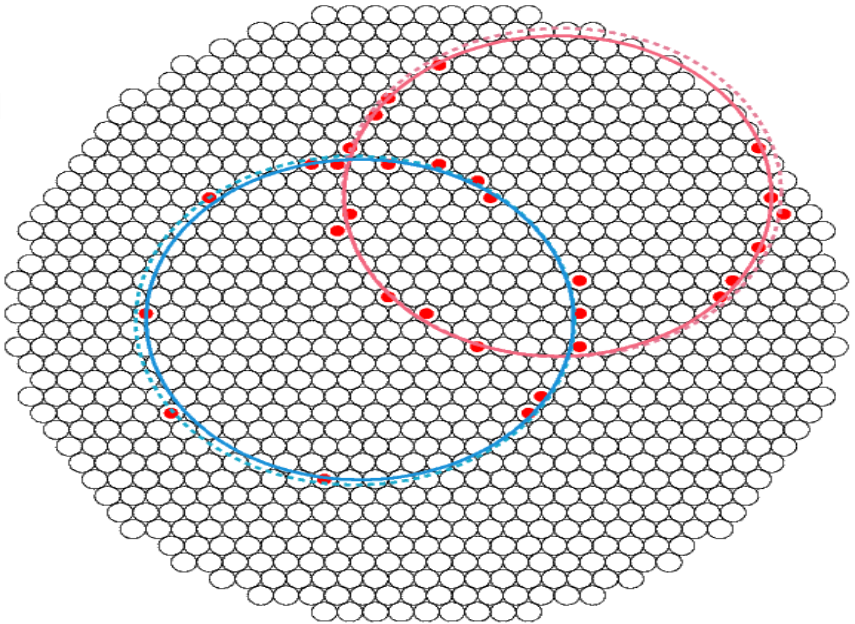
- New algorithm (Almagest) based on Ptolemy's theorem: *"A quadrilater is cyclic (the vertex lie on a circle) if and only if is valid the relation: AD*BC+AB*DC=AC*BD "*
- Design a procedure for parallel implementation

Almagest

i) Select a *triplet* (3 starting points)

ii) Loop on the remaining points: if the next point does not satisfy the Ptolemy's condition then reject it

iii) If the point satisfy the Ptolemy's condition then consider it for the fit

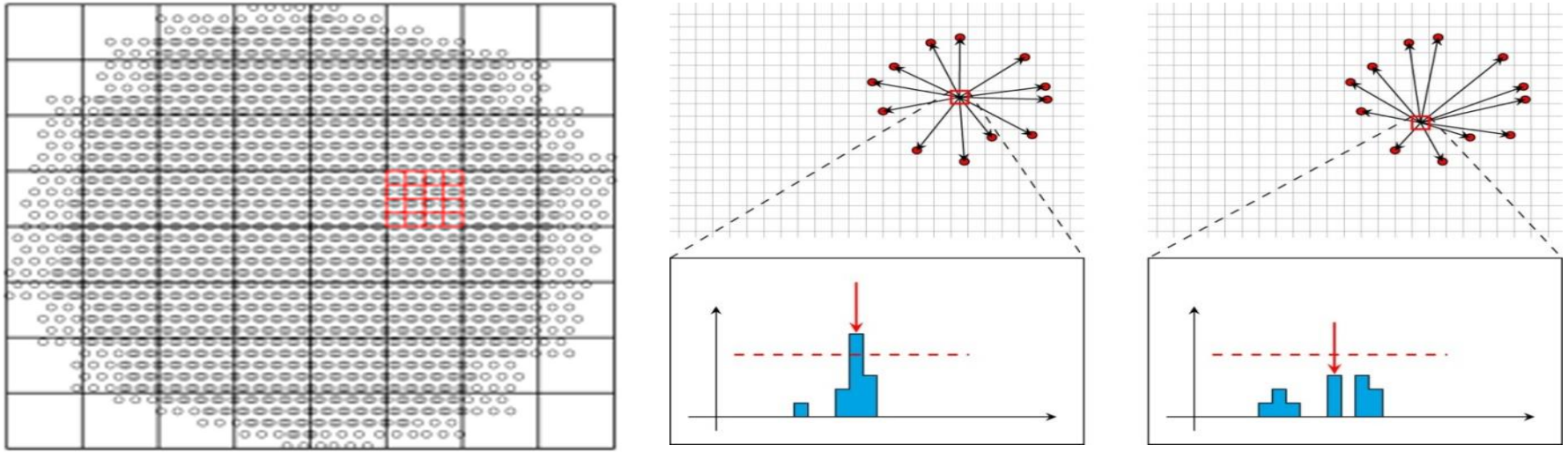


iv) …again…

v) Perform a single ring fit

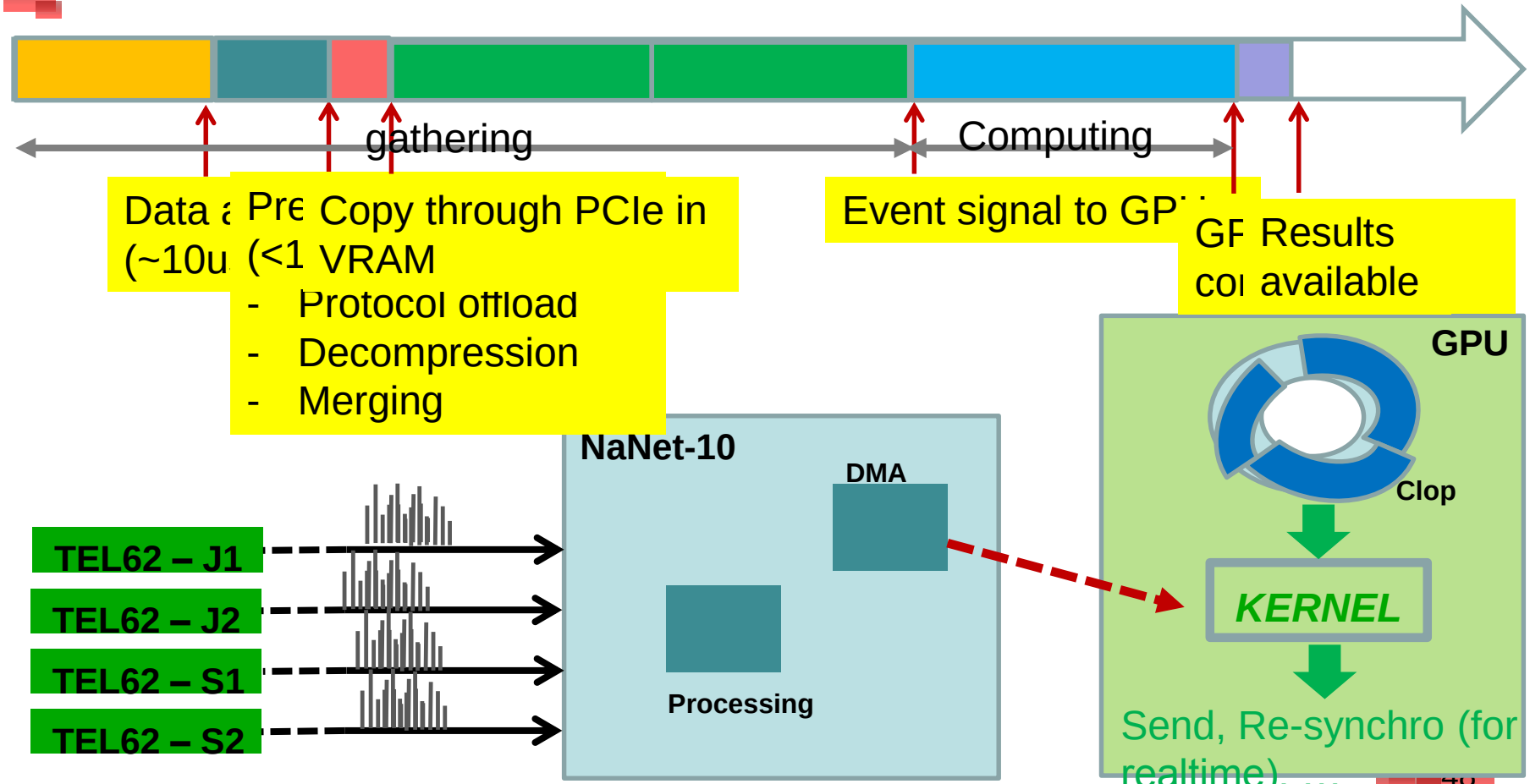
vi) Repeat by excluding the already used points

Histogram



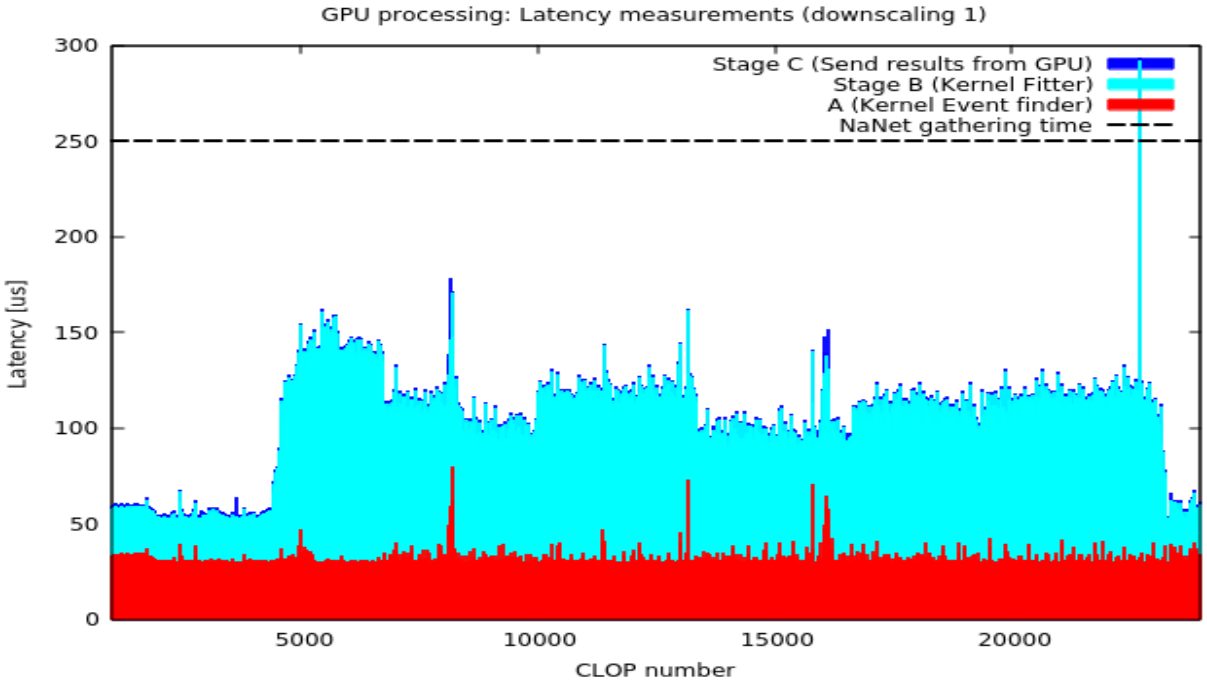
- The XY plane is divided in a Grid
- The histograms of the distances is created for each point in the grid

Processing flow



Results in 2017 NA62 Run

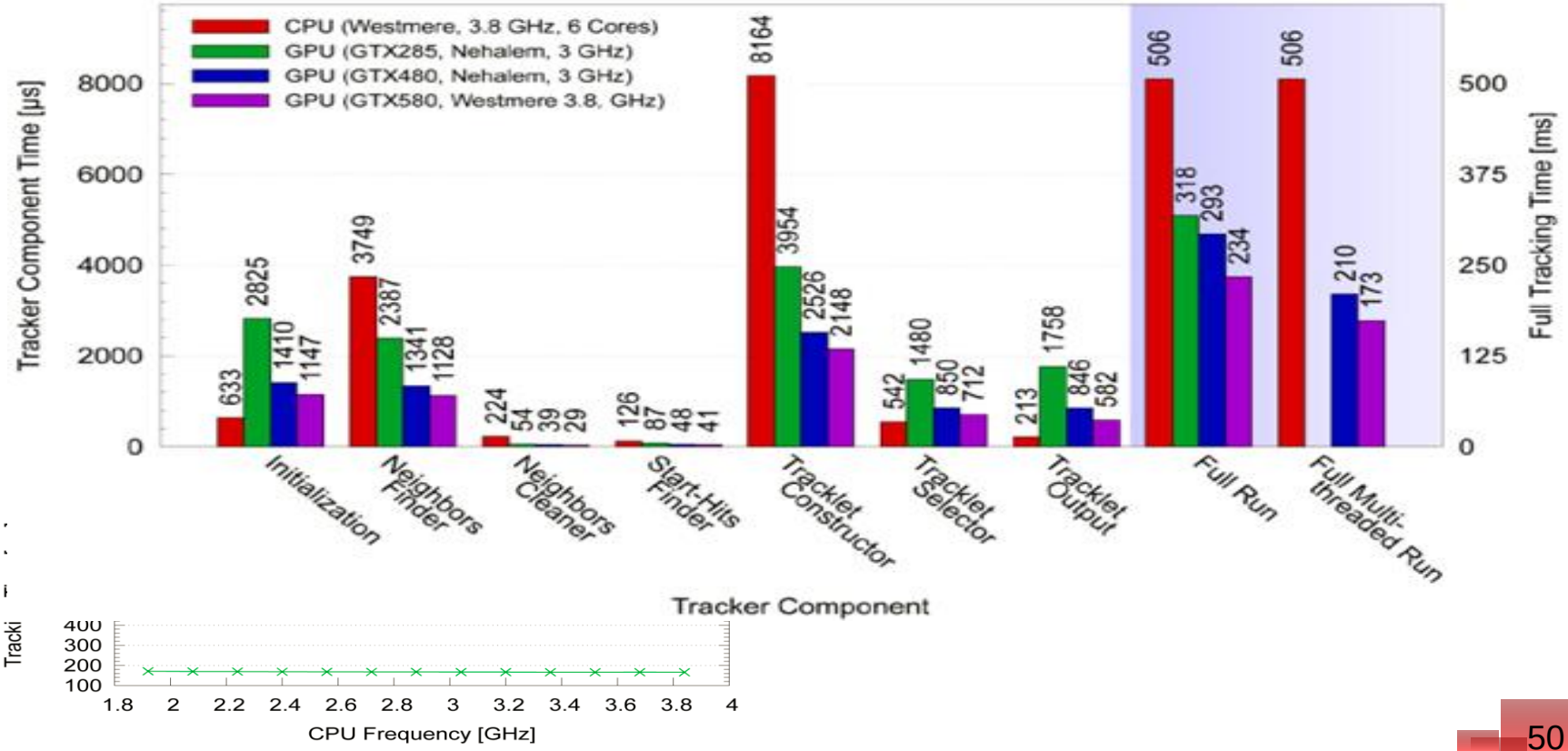
- Testbed
  - Supermicro X9DRG-QF Intel C602 Patsburg
  - Intel Xeon E5-2602 2.0 GHz
  - 32 GB DDR3
  - nVIDIA K20c and P100
- ~ 25% target beam intensity ($9*10^{11}$ Pps)
- Gathering time: 350μs



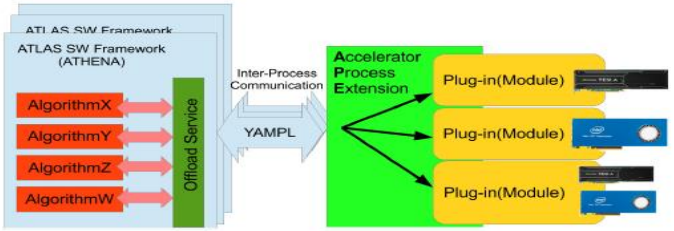
- Processing time per event: 1 us (K20c), <0.20 us (P100)
- Processing latency: below 200 us (compatible with the NA62 requirements)

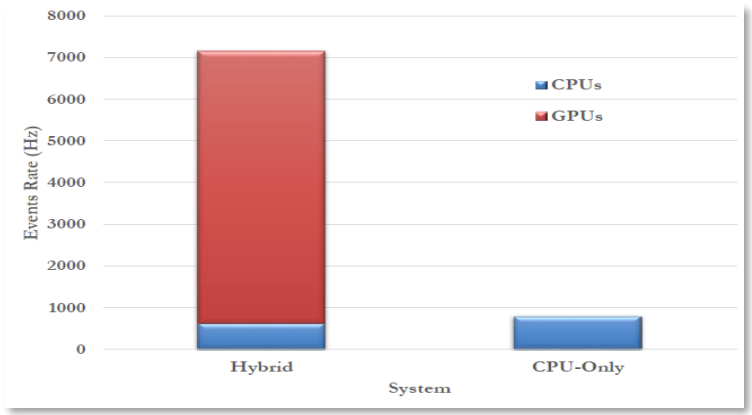
Alice: HLT TPC online Tracking

- 2 kHz input at HLT, 5×10^7 B/event, 25



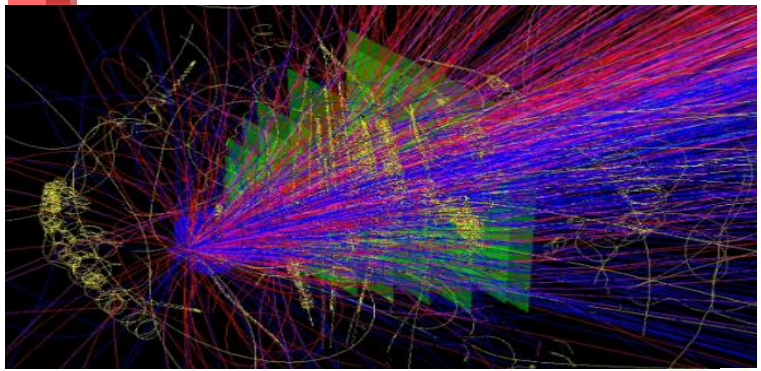
- ATLAS
  - Demonstrators in Run1
  - Accelerator Process Extension(APE) Framework
  - Inner Detector, tracking based on Cellular Automata(CA)
  - Calorimeter, jet finding and clusterization based on CA
  - I Muon, tracking based of hough transforms



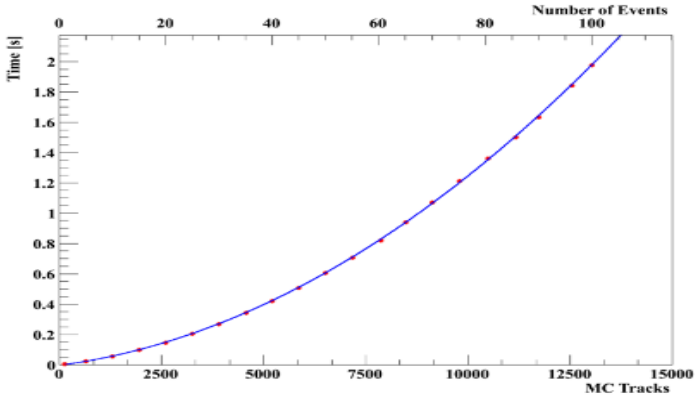


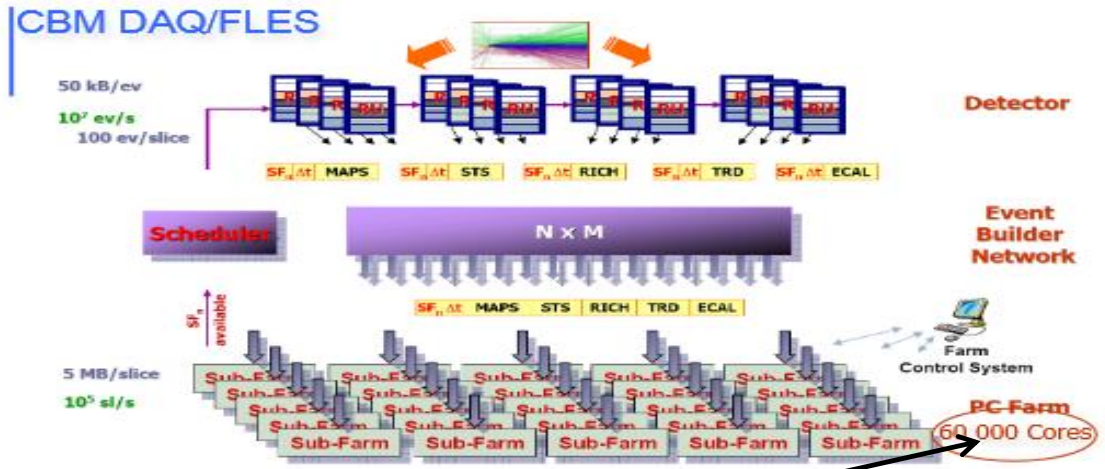
- CMS
  - PATATRACK
  - Cellular Automaton for pixels detector
  - Test with 8xGPUs (GTX 1080)+24 CPU →1 box
  - Integration in the HLT CMS farm under study

CBM & STAR

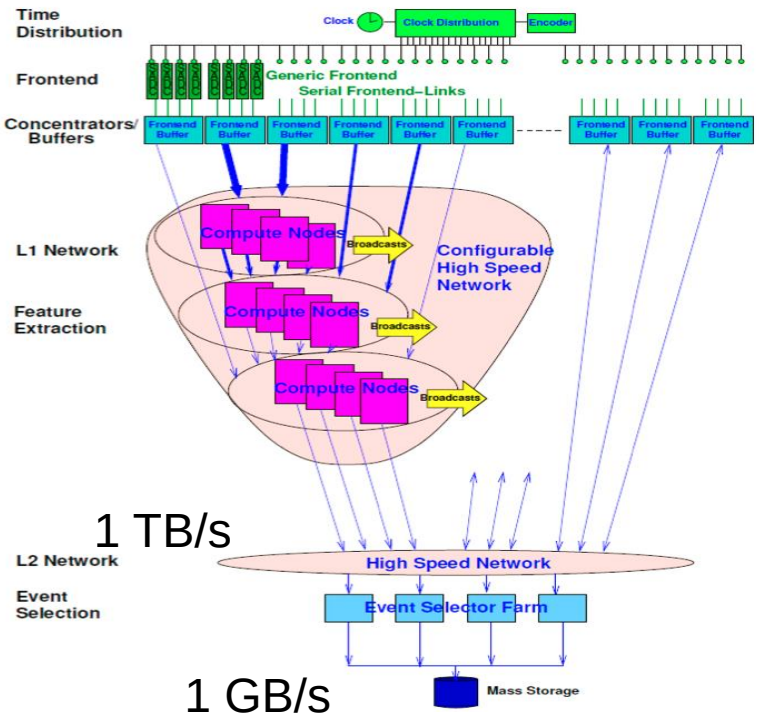


- 10^7 Au+Au collisions /s
- ~1000 tracks/event
- trigger-less
- Since the continuos structure of the beam ~10000 tracks/frame





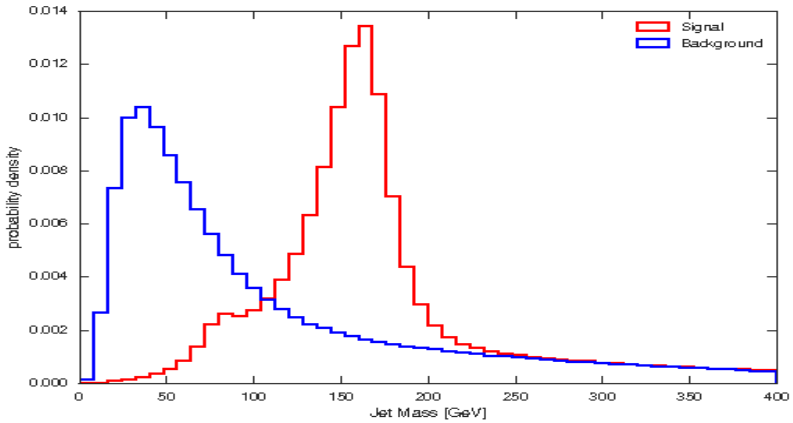
Grid=100000 cores



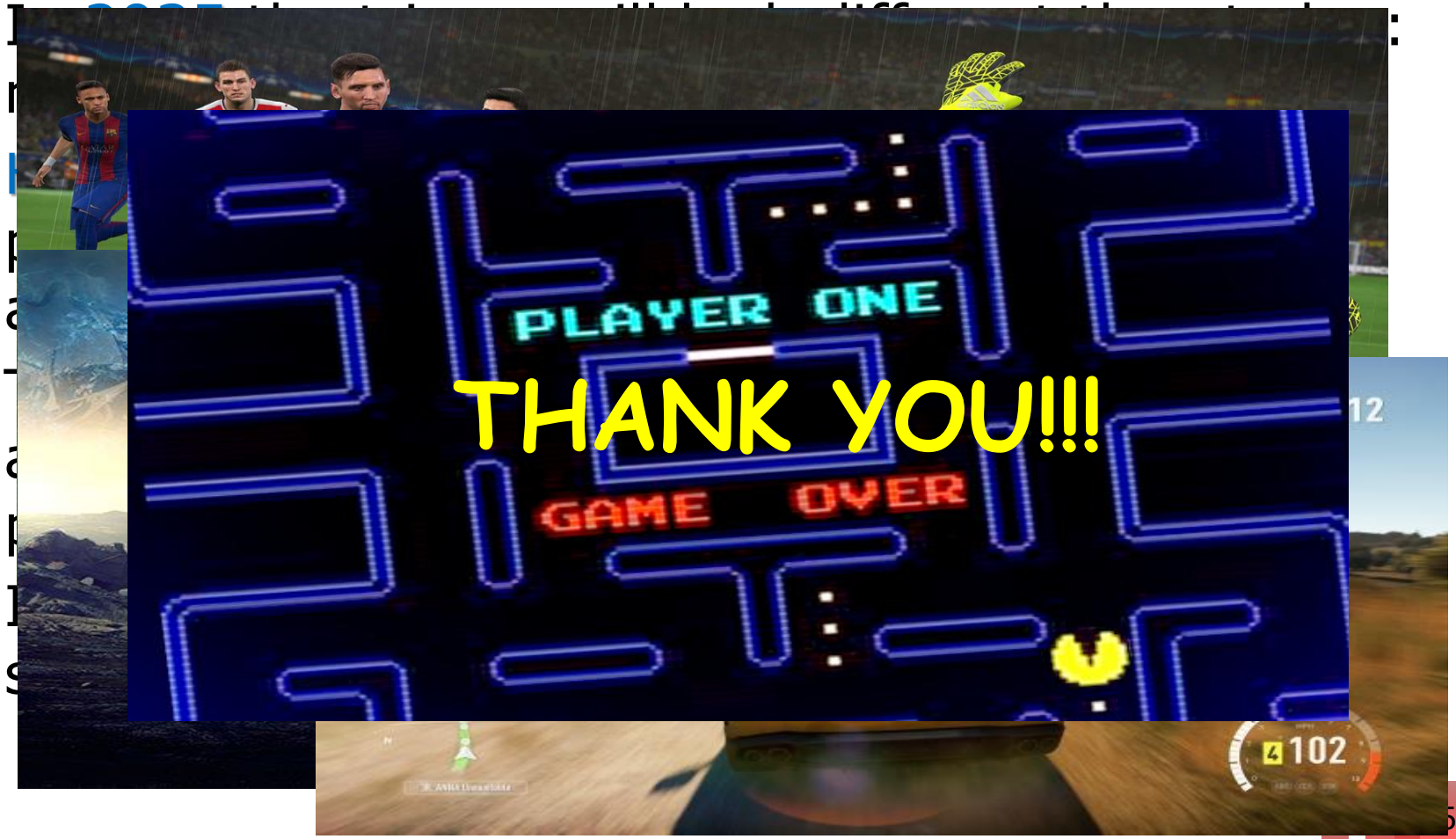
- 10^7 events/s
- Full reconstruction for online selection: assuming 1-10 ms → 10000 – 100000 CPU cores
- Tracking, EMC, PID,…
- First exercice: online tracking
- Comparison between the same code on FPGA and on GPU: the GPUs are 30% faster for this application (a factor 200 with respect to CPU)

Where to use GPU in HEP?

- Trigger
  - Latency order from 10-1000 us
  - Rate up to O(10 MHz) (per board)
  - Tracking, Calorimeters, Pattern recognition
- Simulation & Analysis
  - Geant V
  - Random number generator
  - Fast linear algebra
- DNN
  - Data quality
  - Jet reconstruction

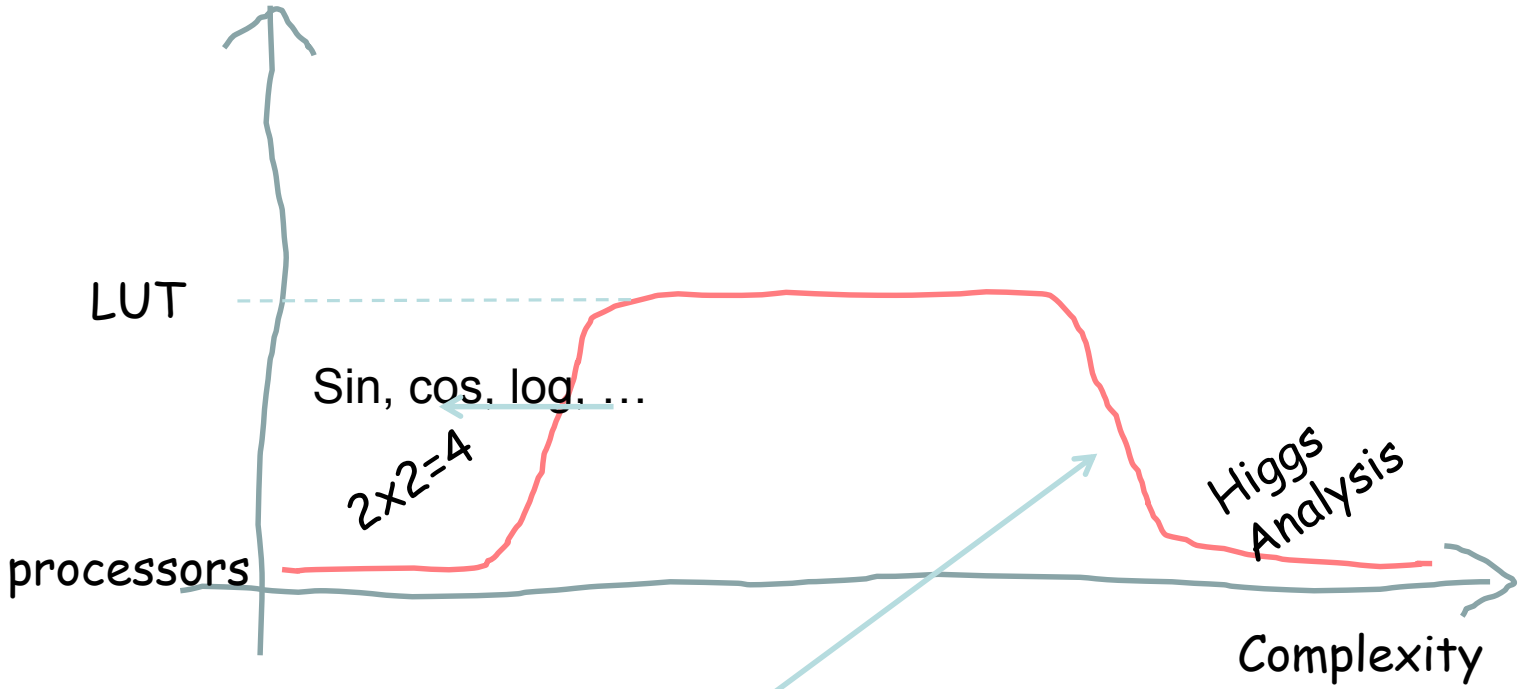


Conclusions



SPARES

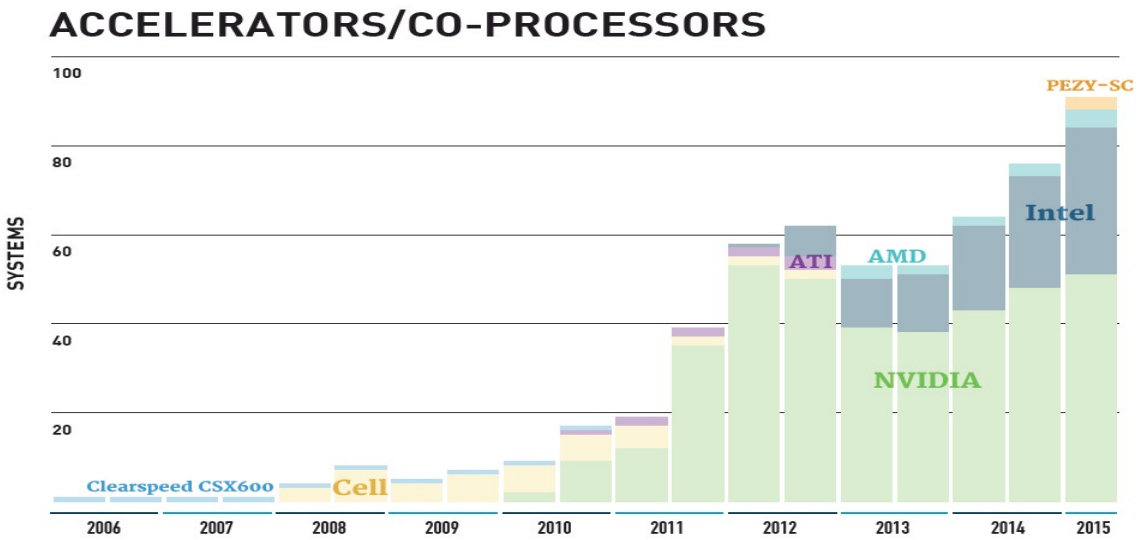
Computing vs LUT



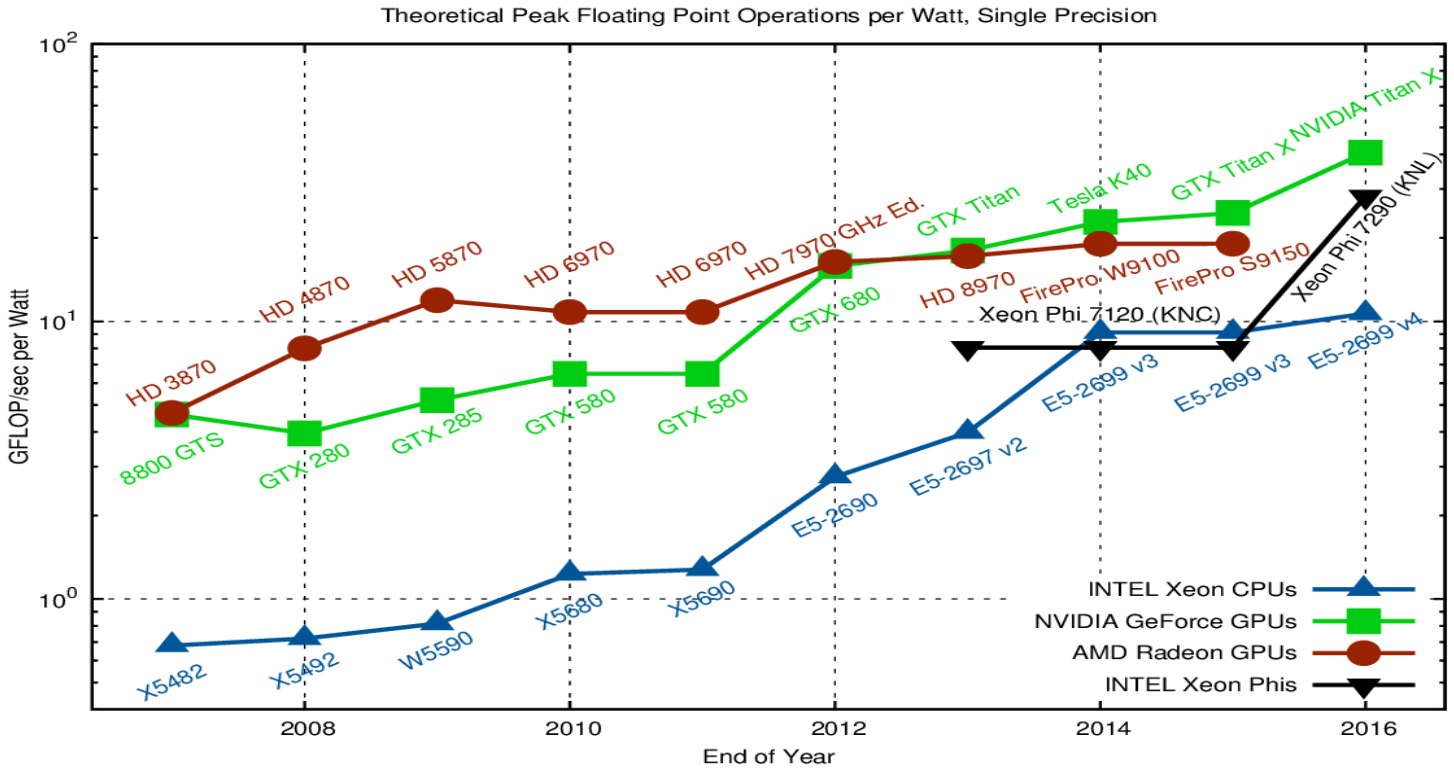
Where is this limit?
It depends …
In any case the GPUs
aim to shrink this space

Accelerators

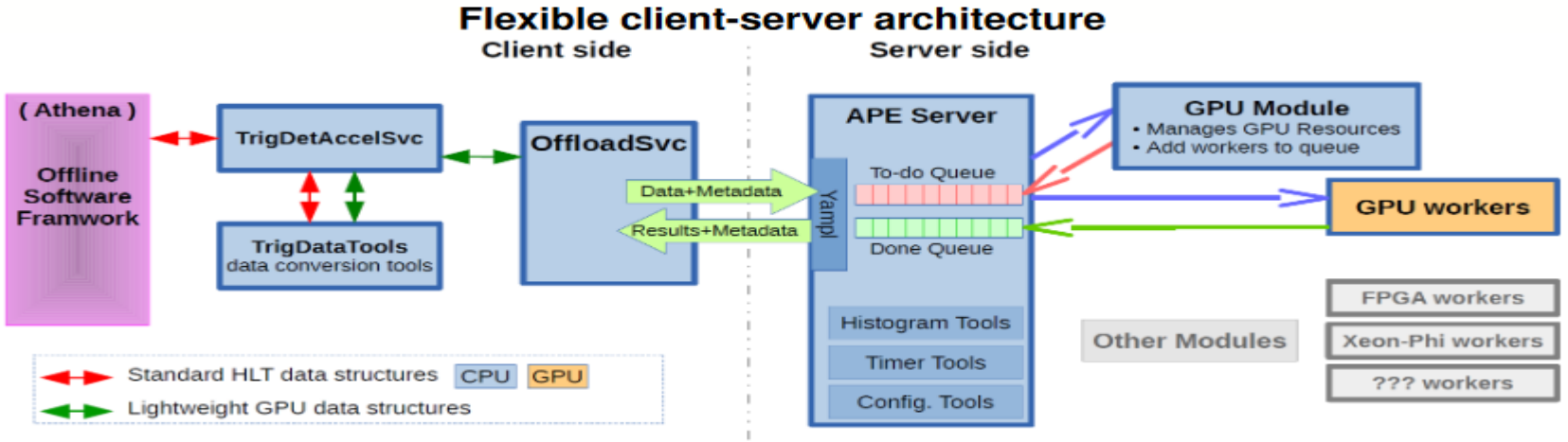
- Accelerators: co-processors for intensive computing
- Nowdays co-processors are connected through standard bus



Computing power



GPU in ATLAS HLT



Client:

- One HLT processing unit per core
- Offline & Online framework (Athena)
  - manage data
  - execute chains of algorithms
  - monitors data-processing

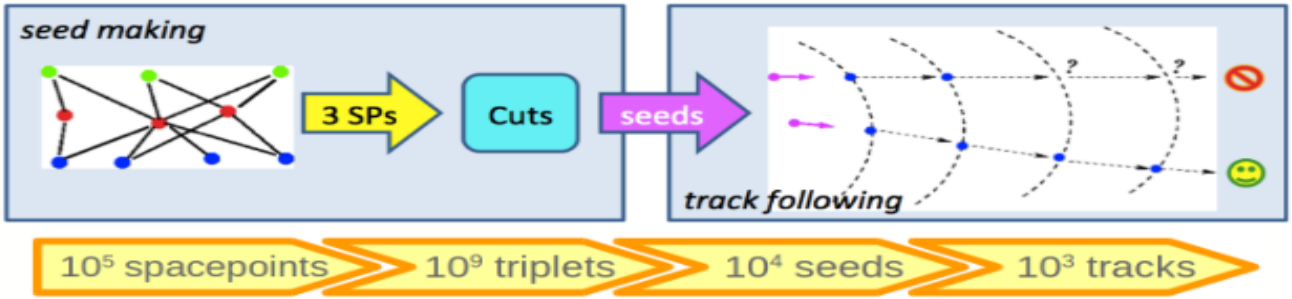
Server:

- Independent from Client framework
- Flexible hardware resources management (multi-devices)
- Preallocate memory for data and store constants

GPU in ATLAS HLT

Tracking is the most time consuming algorithm

- Sequential steps: silicon hit clustering, seeds creation, track following
- Parallelization on GPU of **track-seeding**
- Huge data multiplicity for a full-detector scan tracking: a GPU makes it feasible

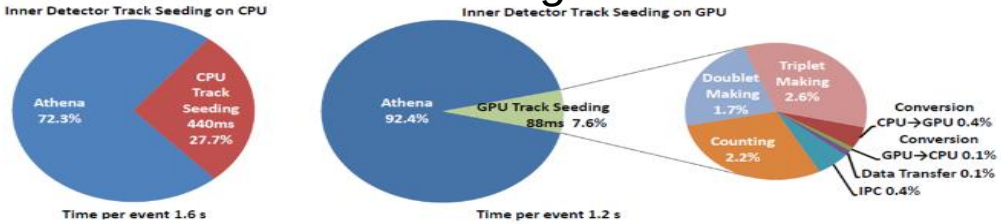


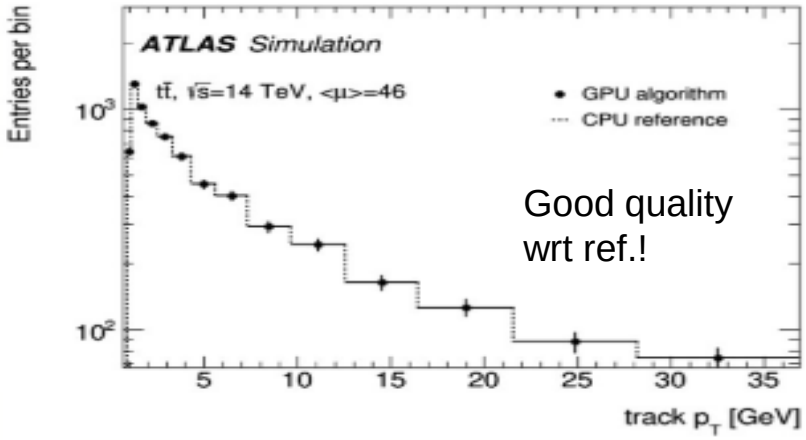
Pair formation: 2D thread array checking for pairing conditions

Triplet formation through 2D thread block

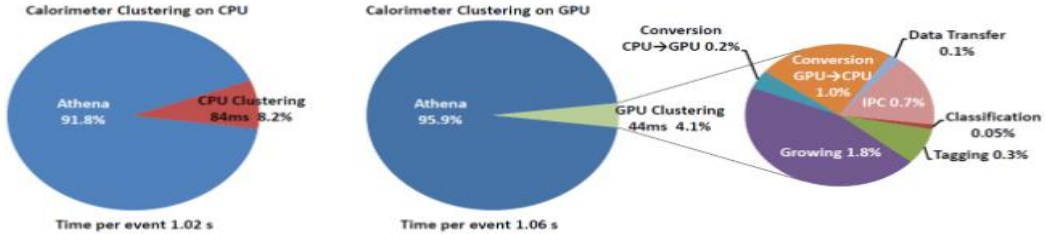
GPU in ATLAS HLT: results

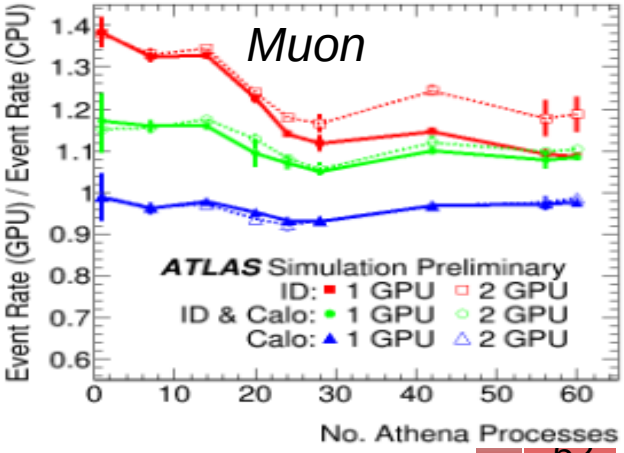
Tracking

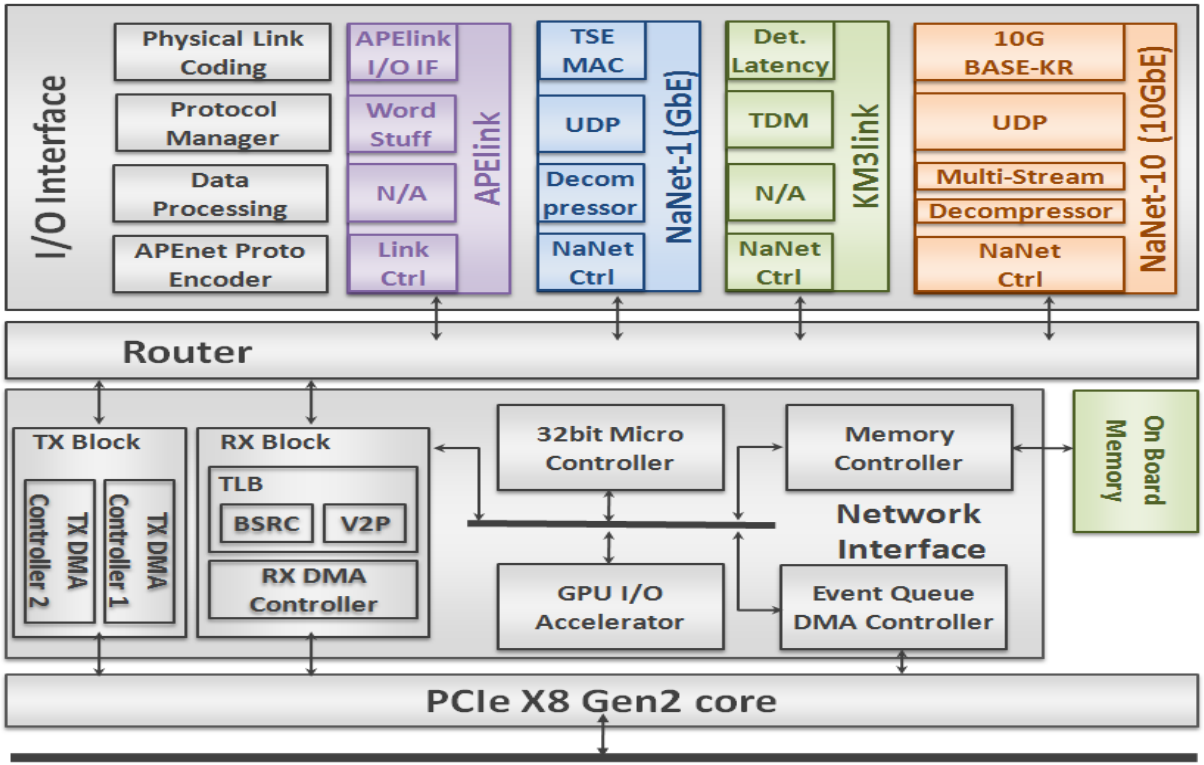




Calorimeter






I/O Interface
Physical Link Coding
Protocol Manager
Data Processing
APEnet Proto Encoder
APElink I/O IF
Word Stuff
N/A
Link Ctrl
APElink
TSE MAC
UDP
Decompressor
NaNet Ctrl
NaNet-1 (GbE)
Det. Latency
TDM
N/A
NaNet Ctrl
KM3link
10G BASE-KR
UDP
Multi-Stream
Decompressor
NaNet Ctrl
NaNet-10 (10GbE)
Router
TX Block
TX DMA Controller 2
TX DMA Controller 1
RX Block
TLB
BSRC
V2P
RX DMA Controller
32bit Micro Controller
GPU I/O Accelerator
Memory Controller
Network Interface
Event Queue DMA Controller
On Board Memory
PCIe X8 Gen2 core