"GPU in HEP: online high quality trigger processing"

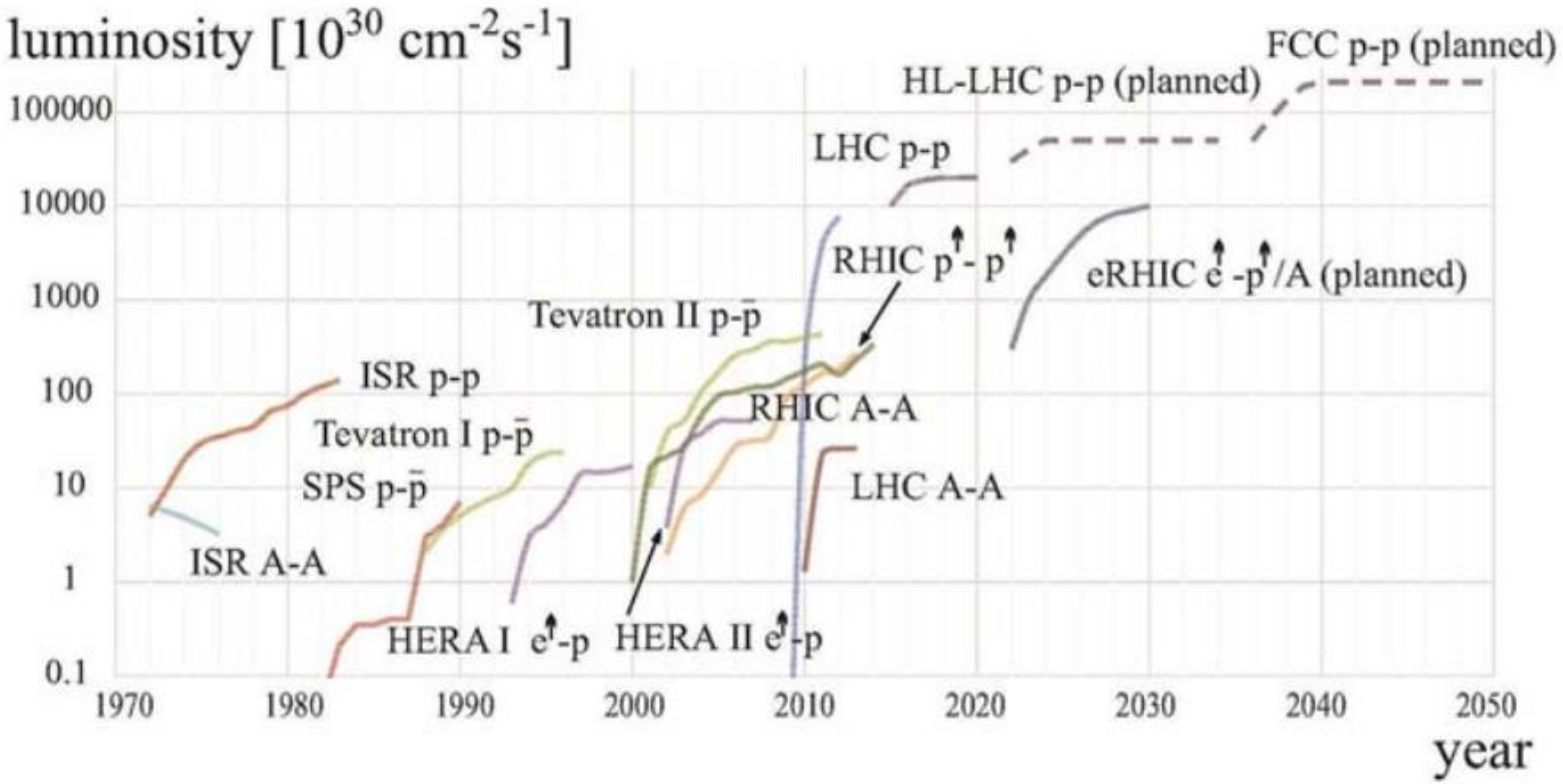
ISOTDAQ
21.6.2024
Hefei

Gianluca Lamanna (Univ.Pisa & INFN)

ISOTDAQ
International School of Trigger and Data Acquisition

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…

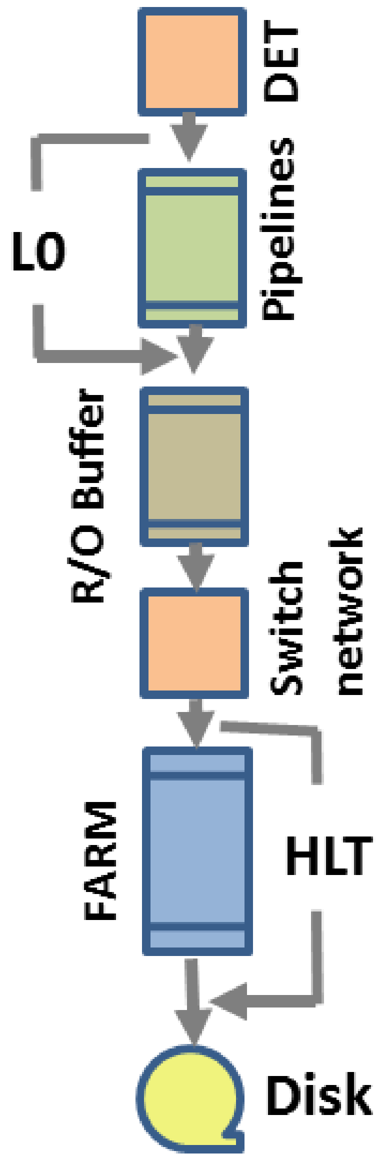
- … will be similar to the current trigger…
  - High reduction factor
  - High efficiency for interesting events
  - Fast decision
  - High resolution
- …but will be also different…
  - The higher background and Pile Up will limit the ability to trigger on interesting events
  - The primitives will be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

The trigger in 2035…

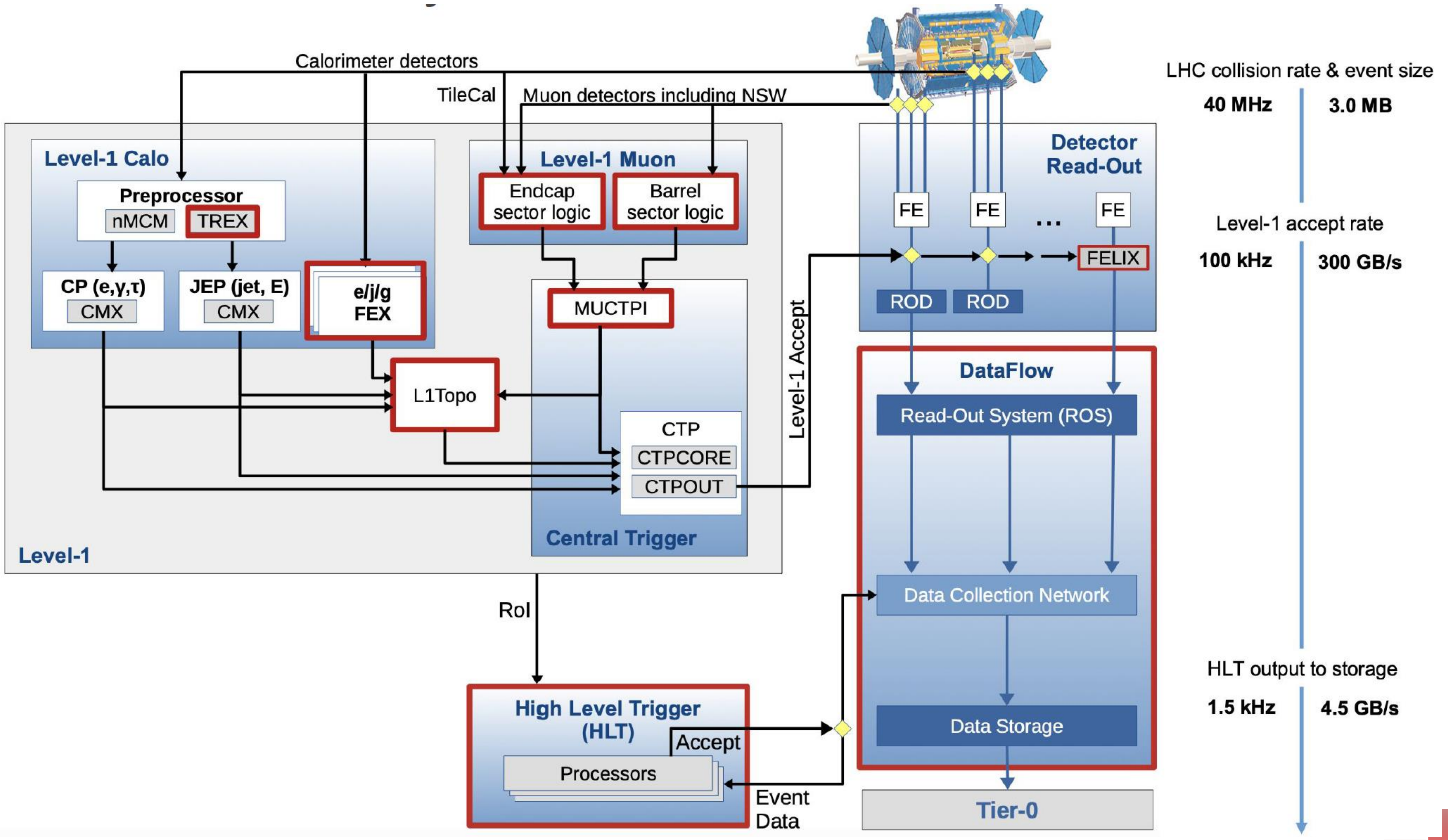
- Higher energy
  - Resolution for high pt leptons → high-precision primitives
  - High occupancy in forward region → better granularity
- Higher luminosity
  - track-calo correlation
  - Bunch crossing ID becomes challenging, pile up
- <u>All of these effects go in the same direction</u>
  - <u>More resolution & more granularity → more data & more processing</u>
- *What previously had to be done in hardware may now be done in firmware; What was previously done in firmware may now be done in software!*

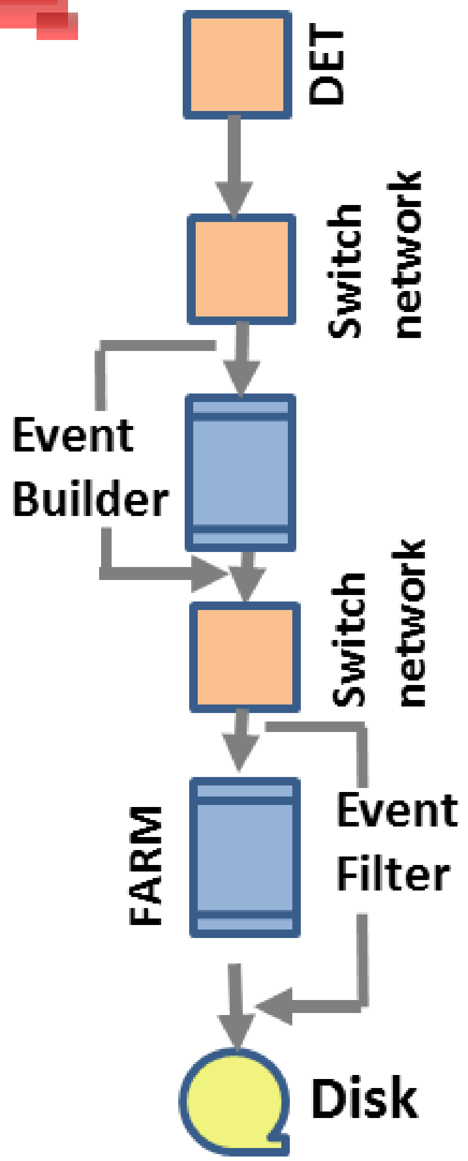
Classic trigger in the future?

- 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
- <u>Main limitation</u>: high quality trigger primitives generation on detector (**processing**)



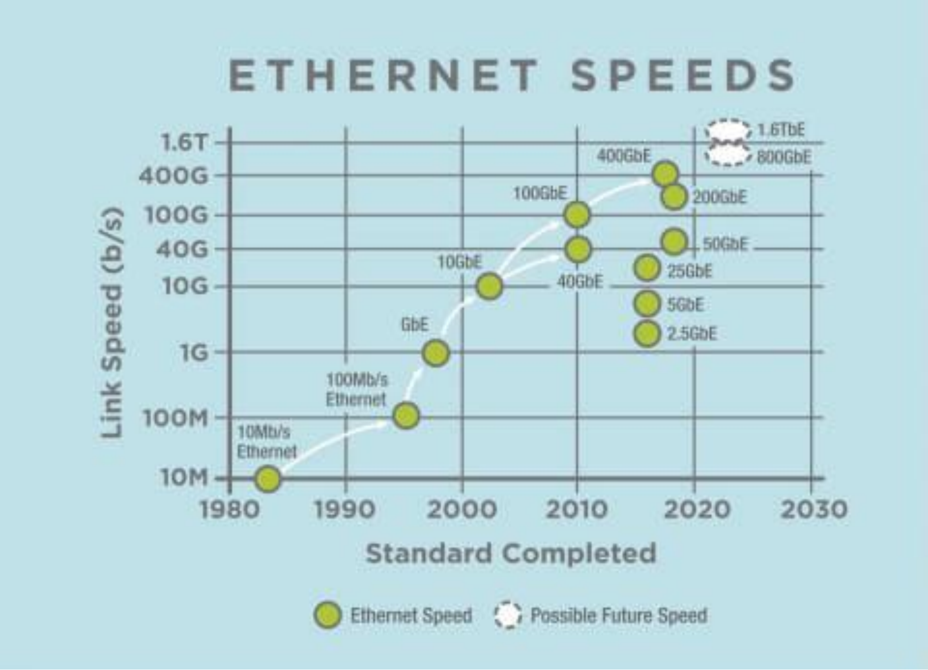
Pipelined trigger in Run3

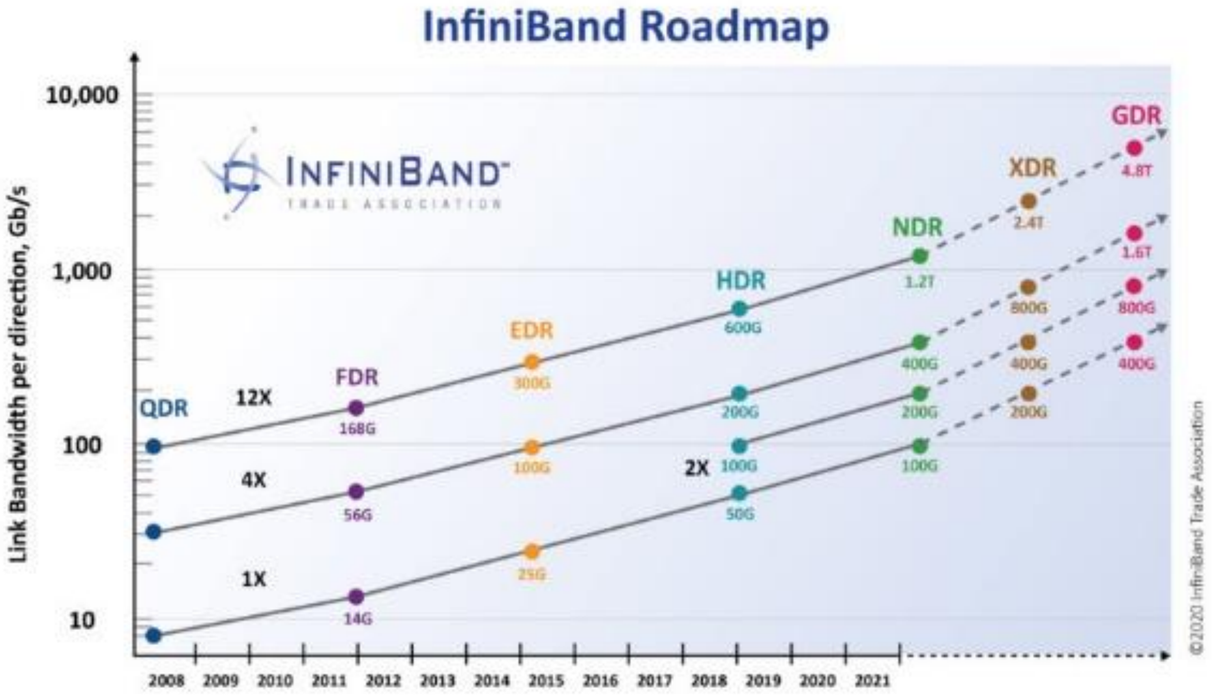




- Is it possible to bring all data on PCs?
  - LHCb: yes in 2022
    - 30 MHz readout, 40 Tb/s data network, 4000 cores, 8800 links
    - (Maybe) No in 2035: track+calo=2PB/s + 5 PB/s ev.building (for comparison largest Google data center = 1 PB/s)
  - CMS & ATLAS: probably no (in 2035)
    - 4 PB/s readout data, 4M links, x10 in performance for switch, x2000 computing
- Main limitation: **data transport**

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

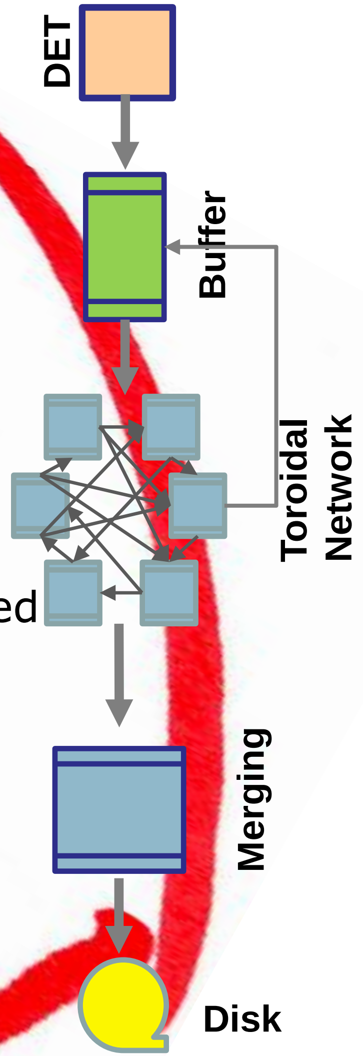
Classic pipeline:

Focus on On-detector processing

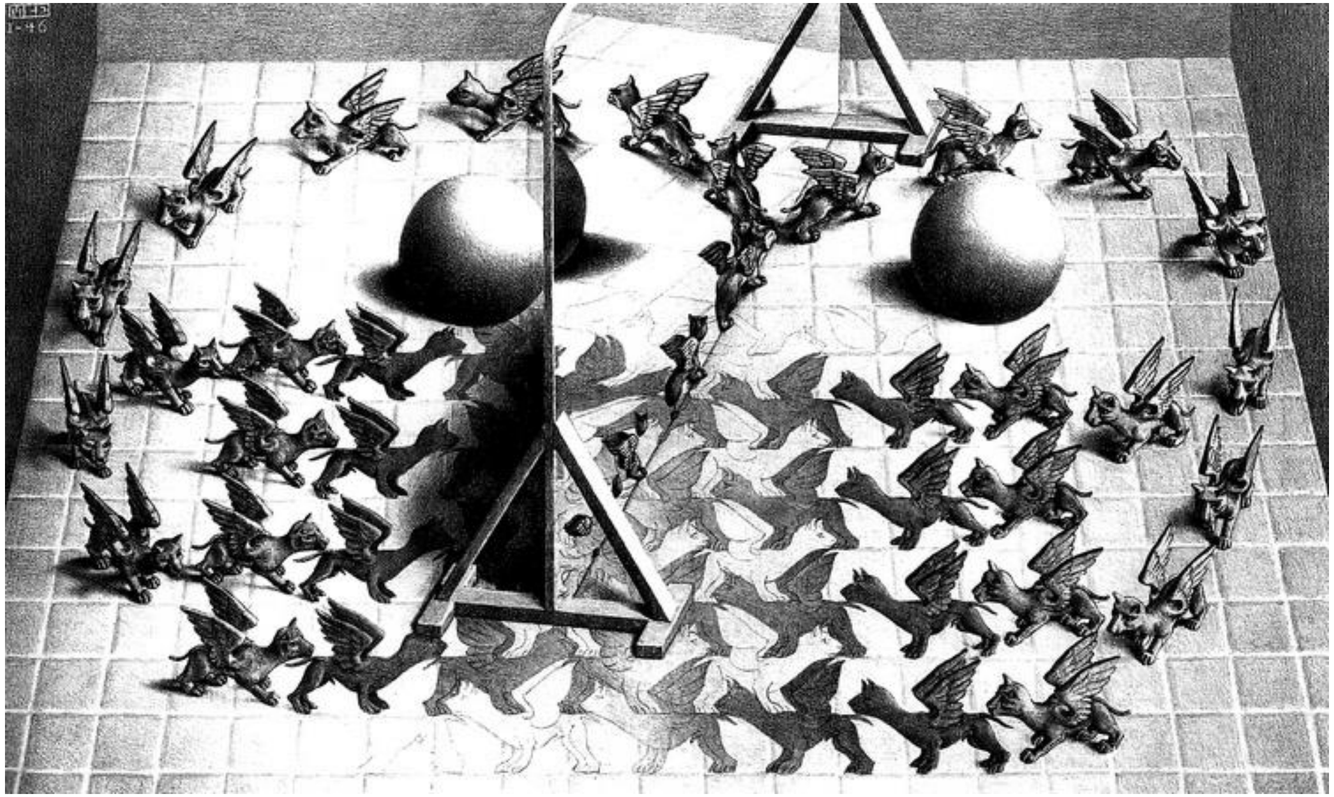
High Latency Trigger:

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

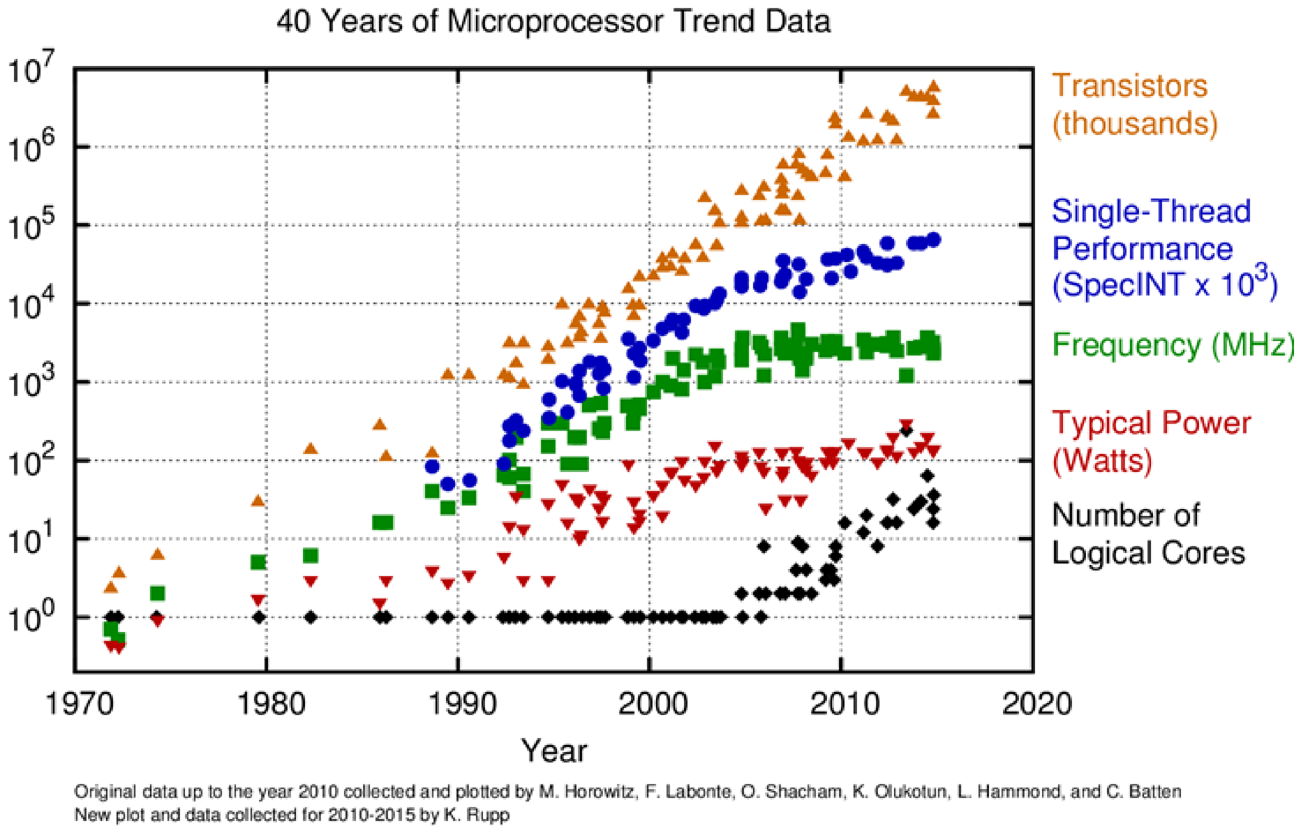
Focus on On-detector Buffers



GPU: Graphics Processing Units



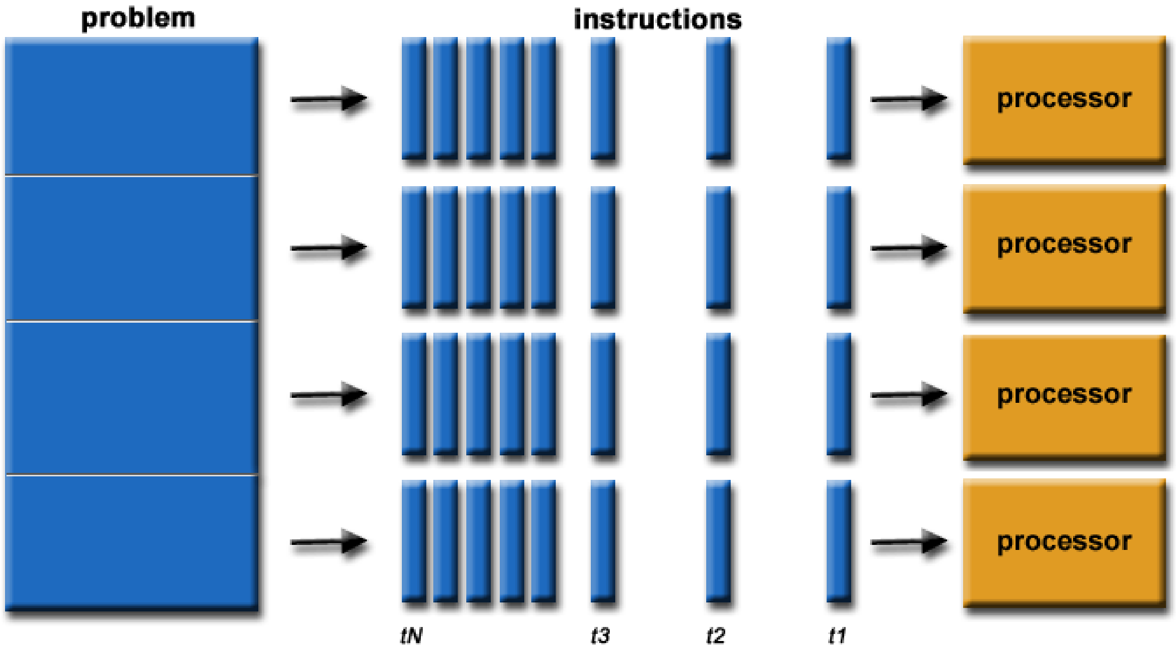
Moore’s Law



- 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

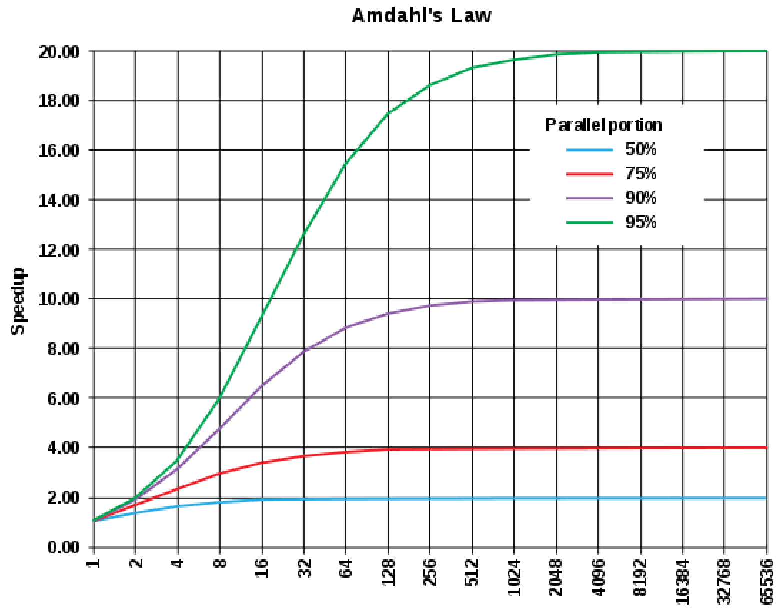
Parallel programming

- 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)



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

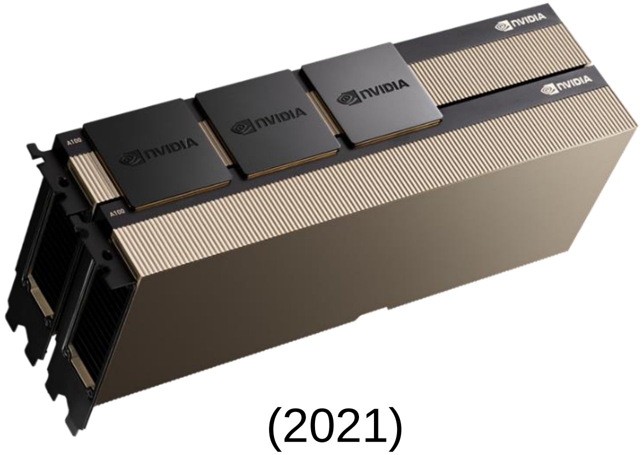
$$S_{latency} = 1 - p + sp$$

- The GPUs progra
- Rende etc. are ty helps

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)

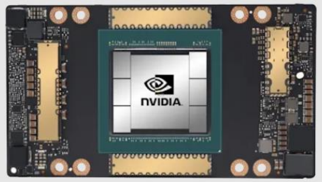

(2021)

Why the GPUs?

- GPU is a way to cheat the Moore's law
  - SIMD/SIMT parallel architecture

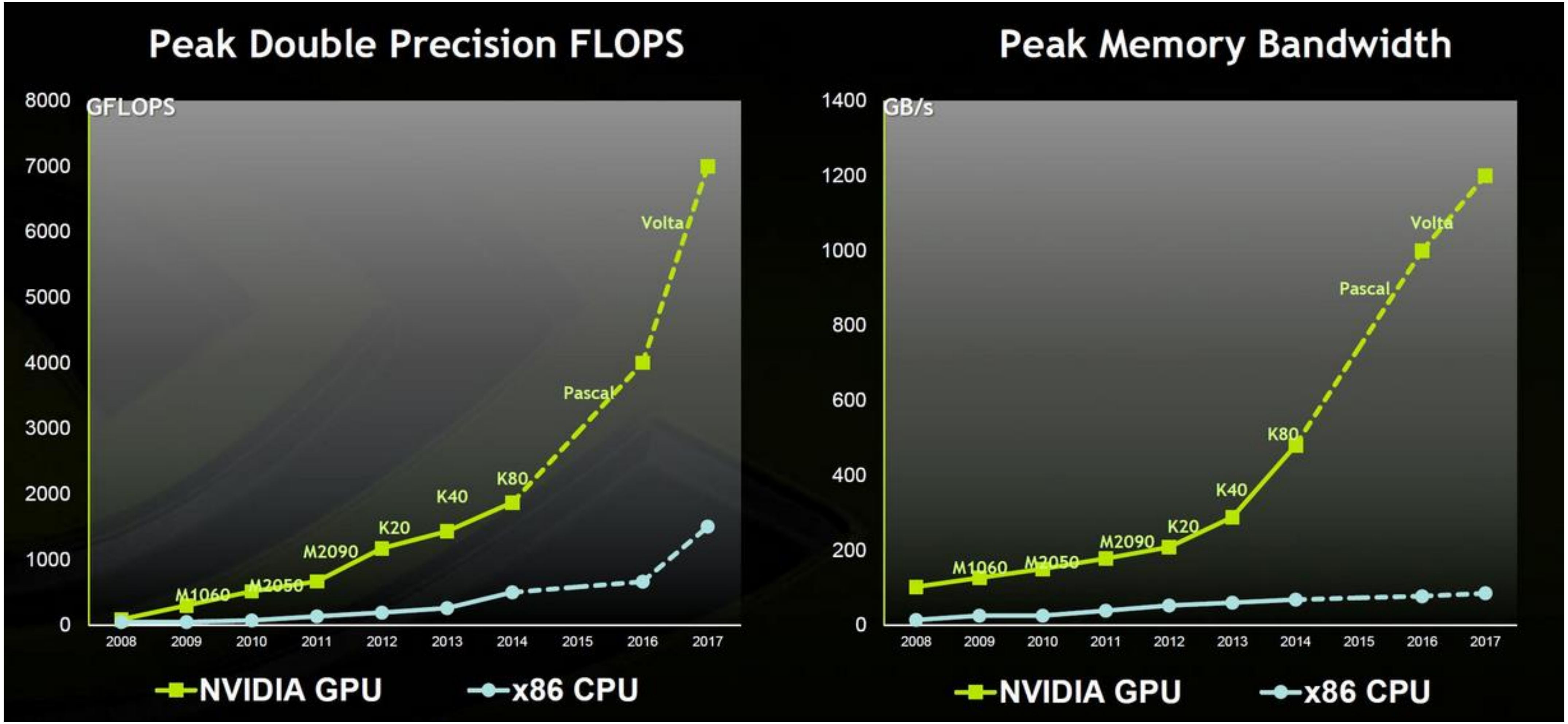
	cores									GFlops
L40 GPU accelerator[388]	Ada Lovelace	October 13, 2022	1× AD102[389]	—	18176	735	2490	GDDR6	384	48	2250	864	362066	90516

NVIDIA Data-Center GPUs Specifications

VideoCardz.com	NVIDIA H100	NVIDIA A100	NVIDIA Tesla V100	NVIDIA Tesla P100
Picture				
GPU	GH100	GA100	GV100	GP100
CUDA Cores	16896/14592*	6912	5120	3584
L2 Cache	50MB	40MB	6MB	4MB
Tensor Cores	528/456*	432	320	–
Memory Bus	5120-bit	5120-bit	4096-bit	4096-bit
Memory Size	80 GB HBM3/HBM2e*	40/80GB HBM2e	16/32 HBM2	16GB HBM2
TDP	700W/350W*	250W/300W/400W	250W/300W/450W	250W/300W
Interface	SXM5/*PCIe Gen5	SXM4/PCIe Gen4	SXM2/PCIe Gen3	SXM/PCIe Gen3
Launch Year	2022	2020	2017	2016

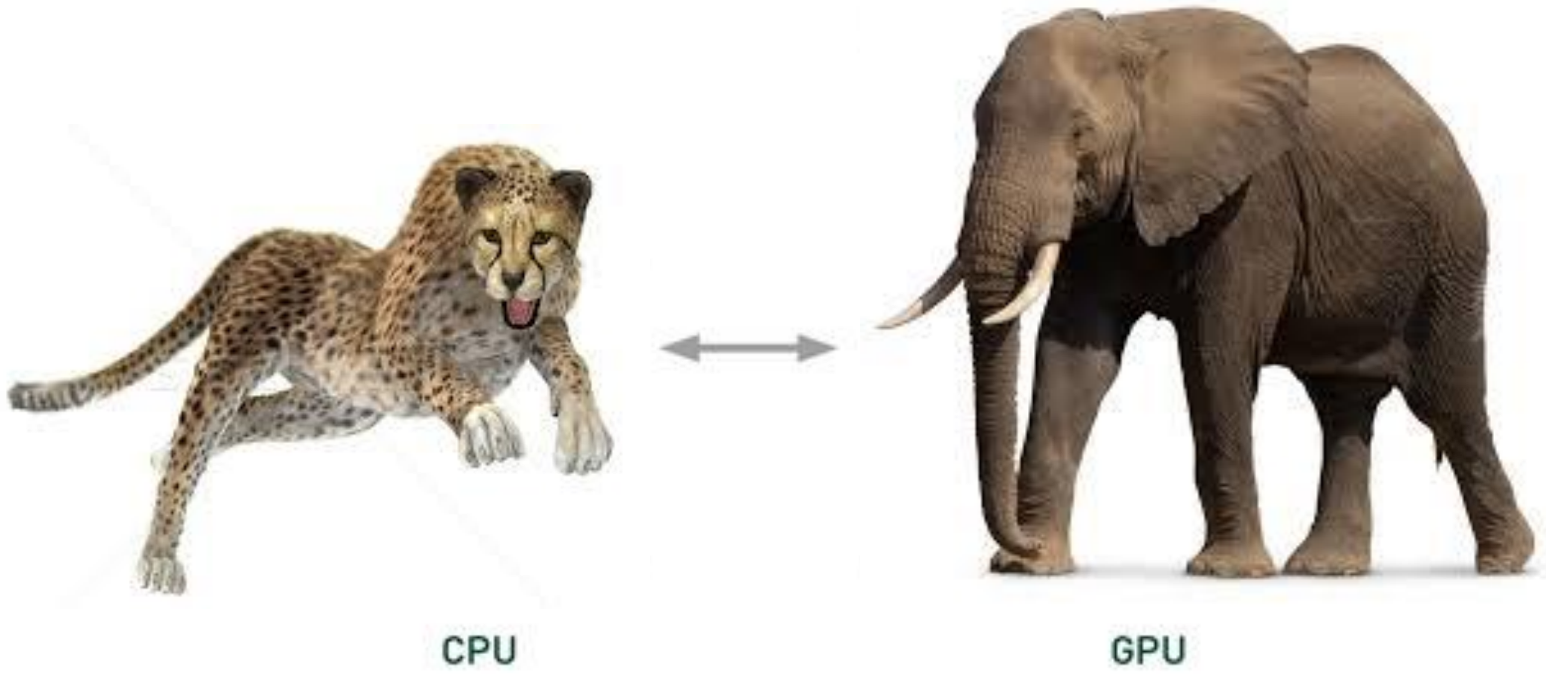
- Several applications in HPC, simulation, scientific computing…

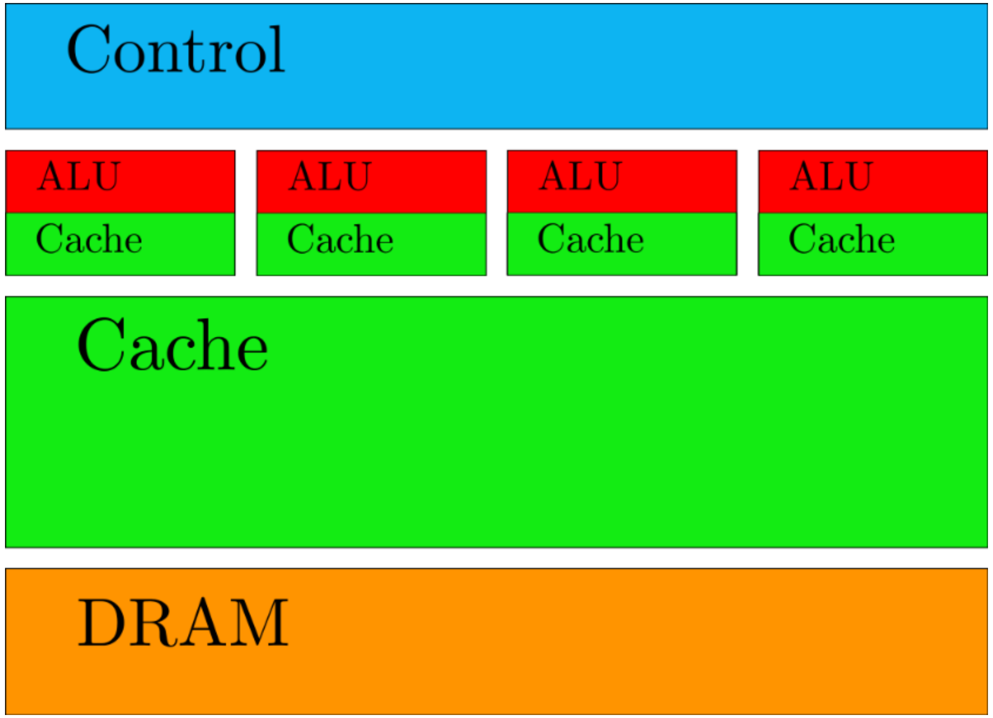
Computing power



- H100: 60 TFlops (30 TFlops), 3TB/s bandwidth
- AD102: 90 TFlops, 0.8 TB/s bandwidth
- Last Intel Processor i9-7980XE Extreme Edition Processor: 1 TFlops , 41.3 GB/s bandwidth

Why?: CPU vs GPU



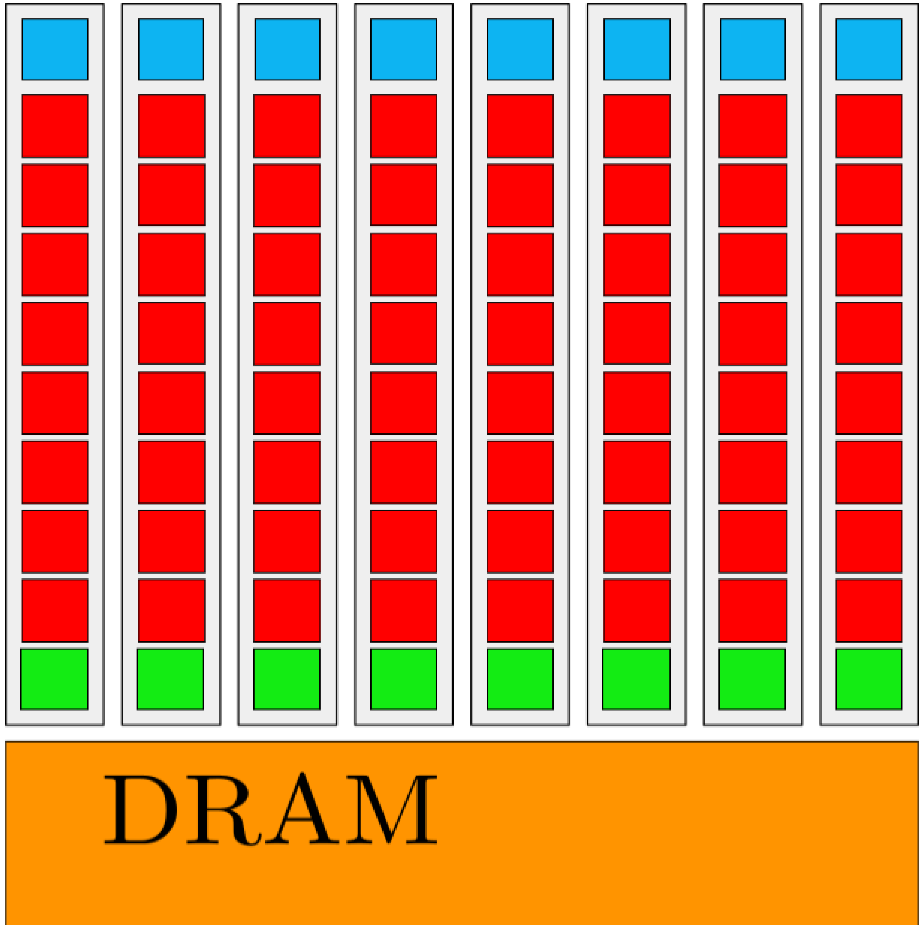


CPU: latency oriented design

- 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/SIMD (Single instruction Multiple Thread/Data) 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

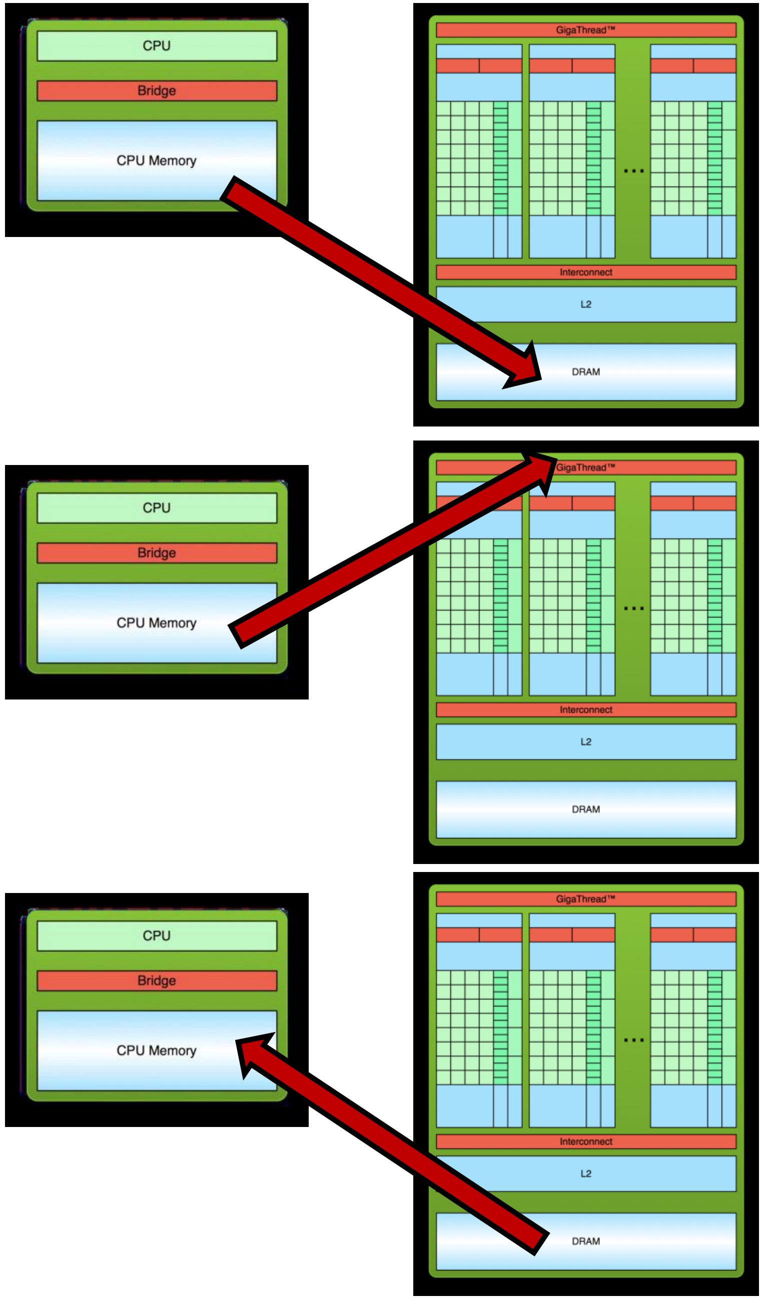
CPU vs GPU

- + Large main memory
- + Fast clock rate
- + Large caches
- + Branch prediction
- + Powerful ALU
- Relatively low memory bandwidth
- Cache misses costly
- Low performance per watt

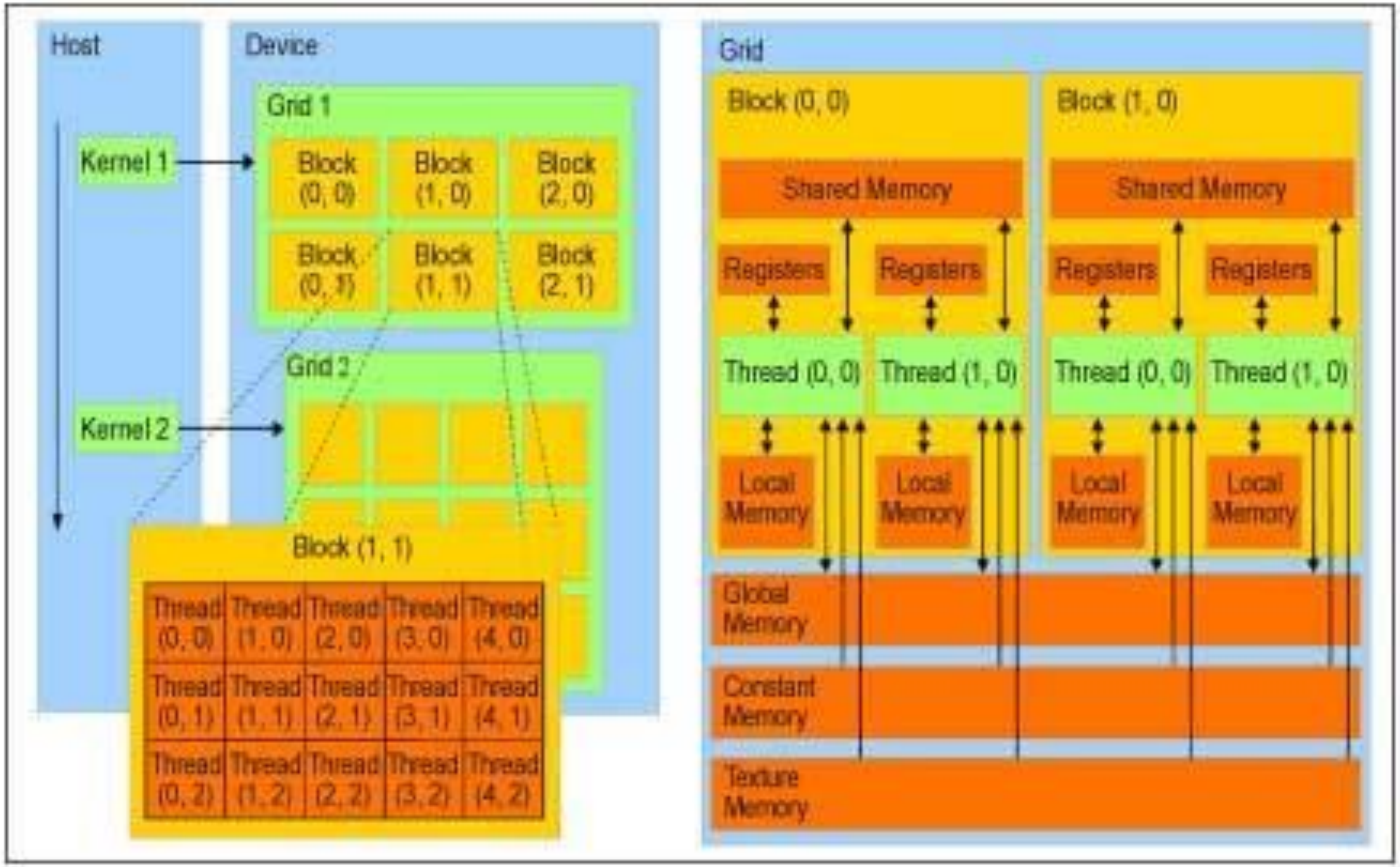
- + High bandwidth main memory
- + Latency tolerant (parallelism)
- + More compute resources
- + High performance per watt
- Limited memory capacity
- Low per-thread performance
- Extension card

- 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



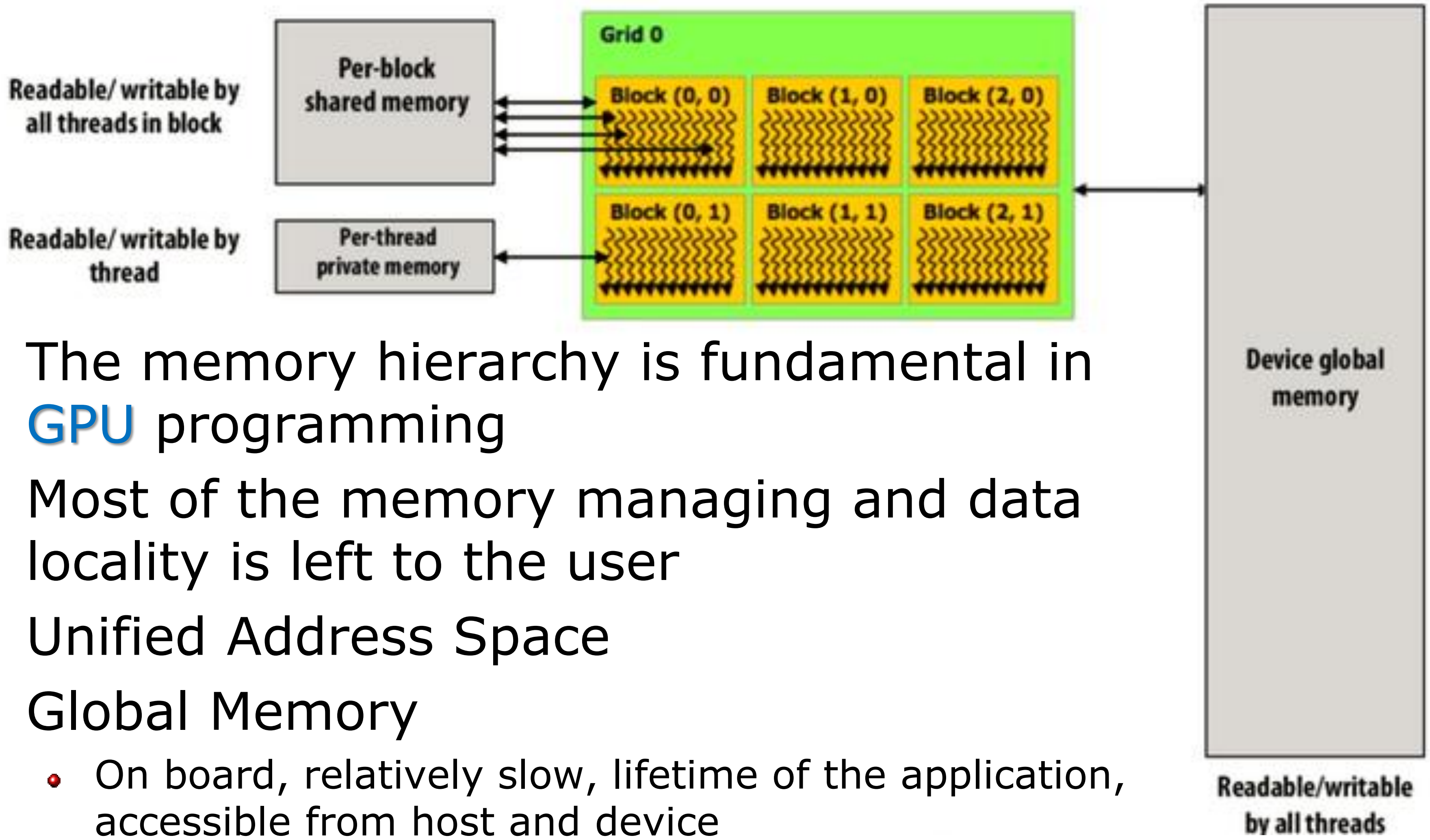
Grids, blocks and threads



kernel launch time

Mapping on the hardware

Memory



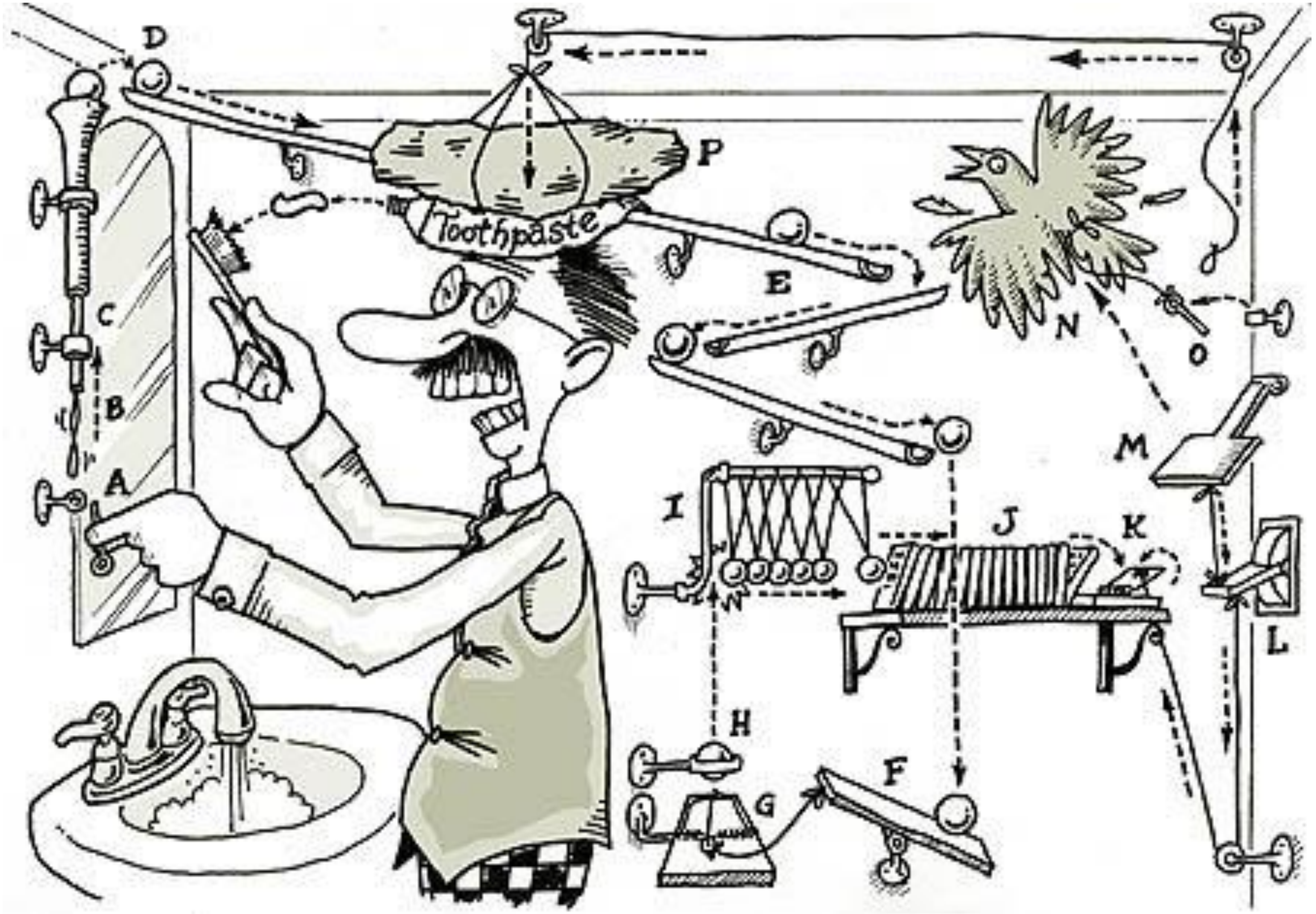
- The memory hierarchy is fundamental in GPU programming
- Most of the memory managing and data locality is left to the user
- Unified Address Space
- Global Memory
  - On board, relatively slow, lifetime of the application, accessible from host and device
- Shared memory/registers
  - On Chip, very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

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.
- HIP, SYCL, Kokkos, …

More on GPU programming in Lab 14!

Triggers 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?

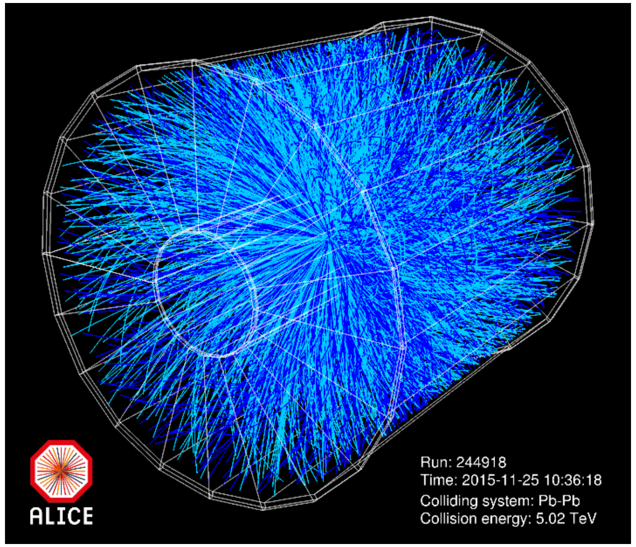
- In High Level Trigger
  - It is the "natural" place. If your problem can be parallelized (either for events or for algorithm) you can gain factor on speed-up → smaller number of PC in Online Farm

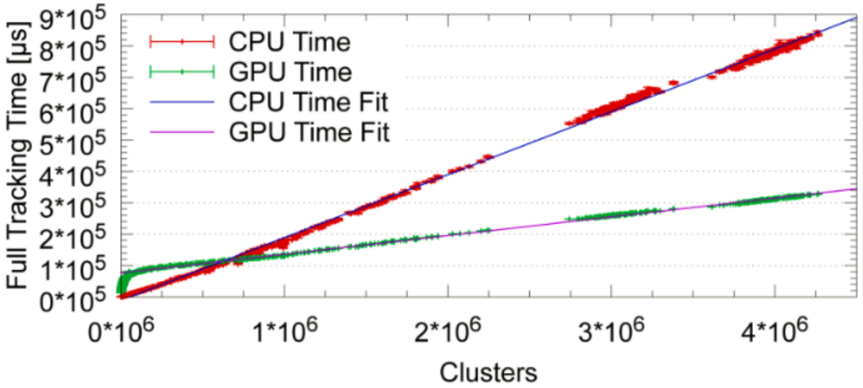
Alice: Rohr@CHEP2021
Rohr@VCI2022
LHCb: Aaji et al. «Computing and Software for Big Science» (2022)
Aaji et al. «Computing and Software for Big Science» (2020)
CMS: Acosta@PITT-PAC2021
Bocci@HepSeminar(caltech) 30.11.2020
ATLAS: Wynne@ECHEP2020
Krasznahorkay@CHEP2019

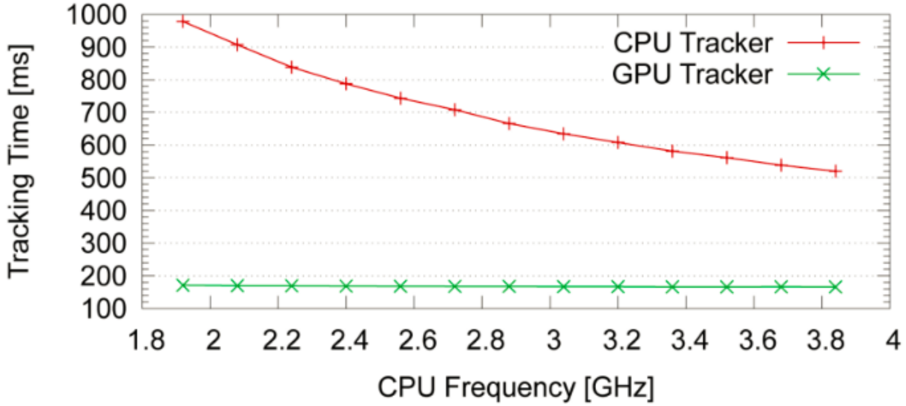
- In Low Level Trigger
  - Bring power and flexibility of processors close to the data source → more physics

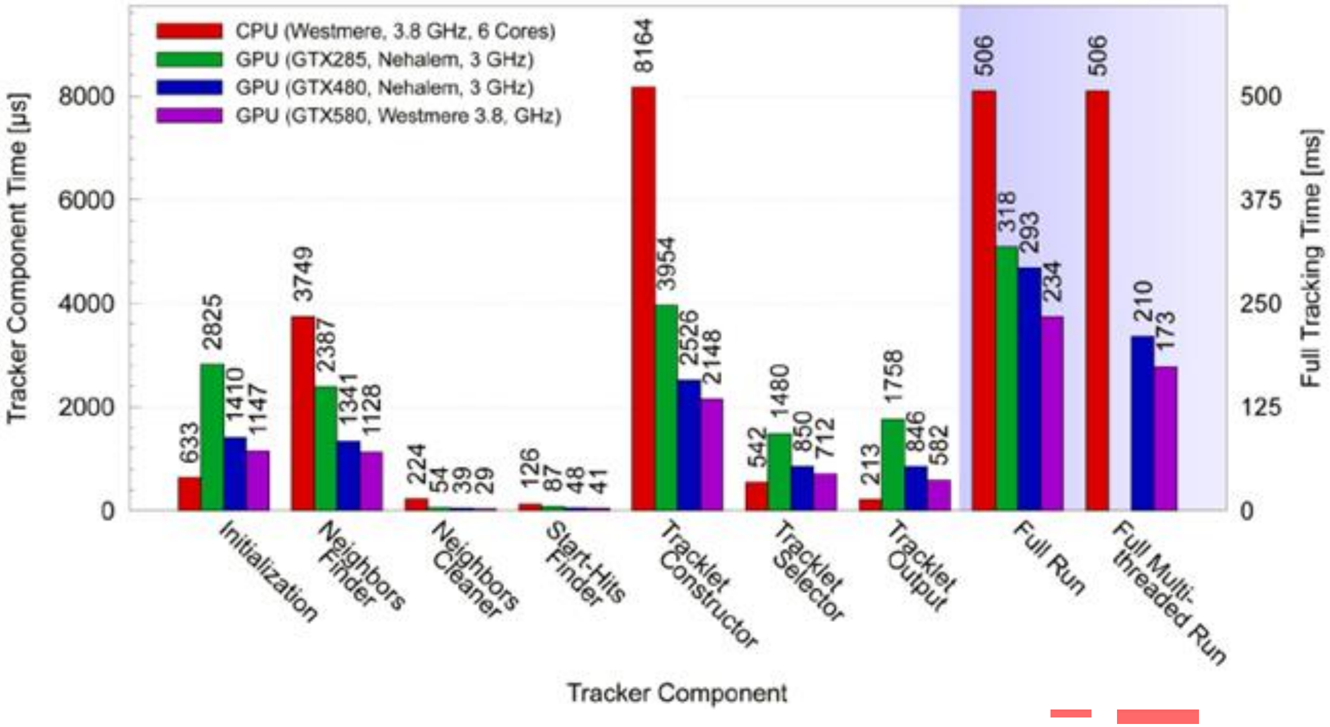
ALICE: HLT TPC online Tracking in RUN1

- 2 kHz input at HLT, 5×10^7 B/event, 25 GB/s, 20000 tracks/event
- TPC
- Cellular automaton + Kalman filter
- GTX 580 (in 2011) and AMD S9000 (2015) → GPUs halves the number of computer nodes (1.5 MCHF cheaper than full CPU)



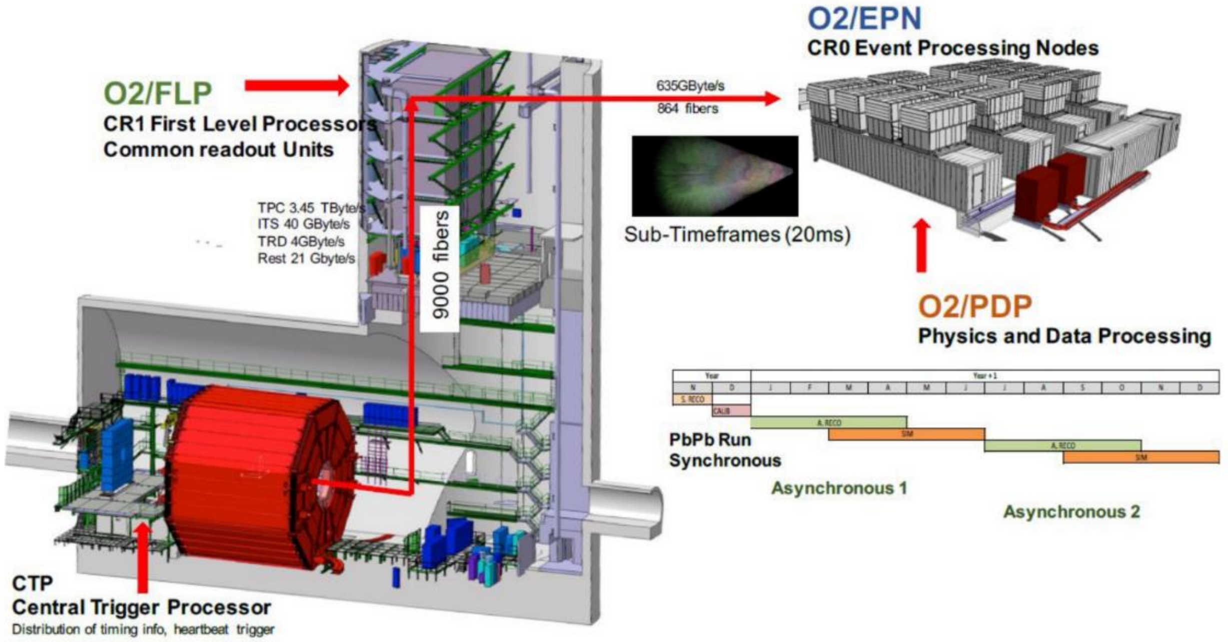






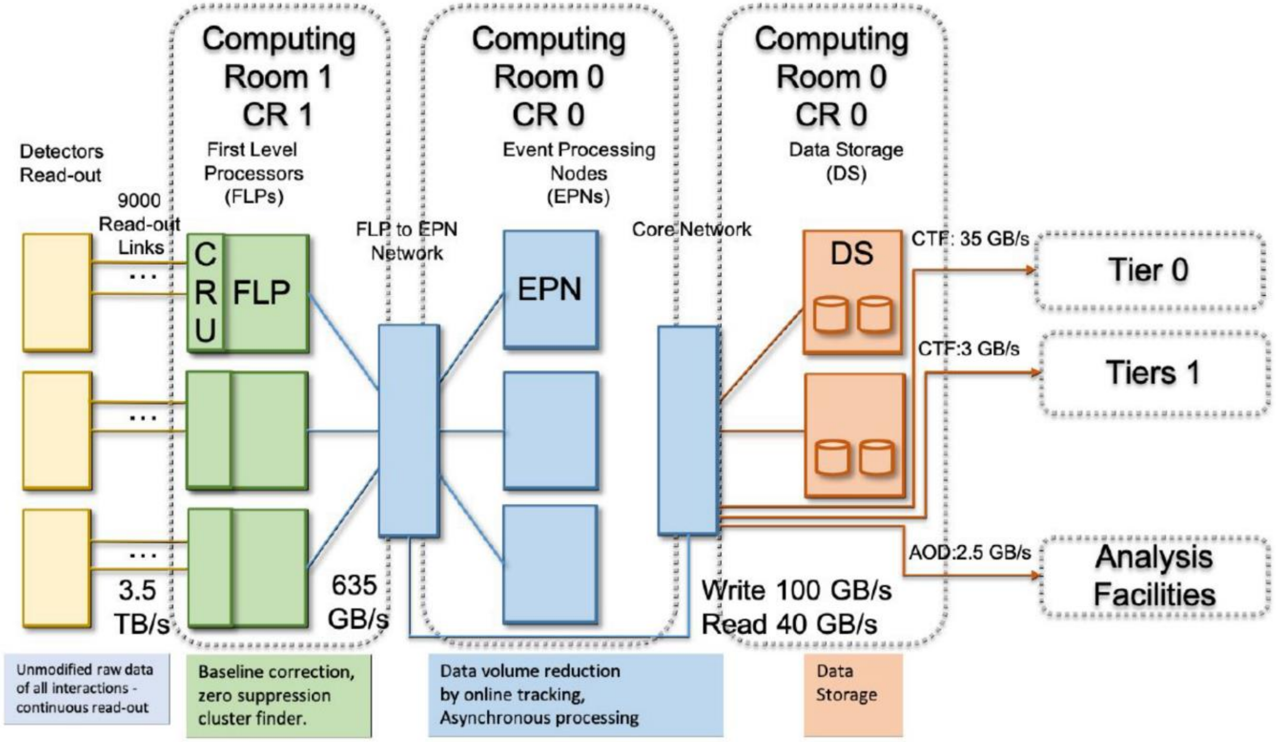
ALICE: the O^2 readout

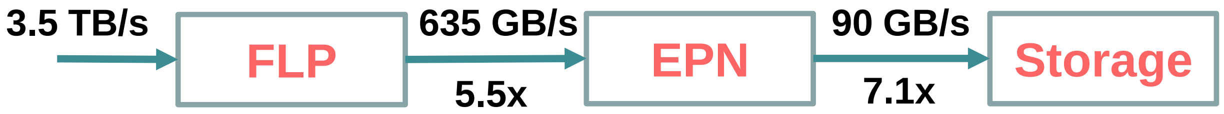
- Run3
  - Detector modified wrt Run1/2
  - x50 events rate and time frames (TF) instead of bunch crossing (1 TF = 10 ms is about 500 events)
  - Continuous readout
- New O^2 (online+offline) trigger-less readout concept
  - Synchronous: calibration and data compression during data taking
  - Asynchronous: final reconstruction , when no beam



ALICE: the O^2 readout

- FLP (Firts level processor) receives data from detectors read-out
  - 9000 read-out links
  - 3.5 Tbyte/s (mainly from TPC)
  - FLP assembles SFT (sub-time frames)
- EPN (Event Processing Node) applies calibration, runs reconstruction and builds the final events
- Data are transfered on disk
  - Not trigger selection applied at any stage
  - Only data compression: ~40x in the full chain



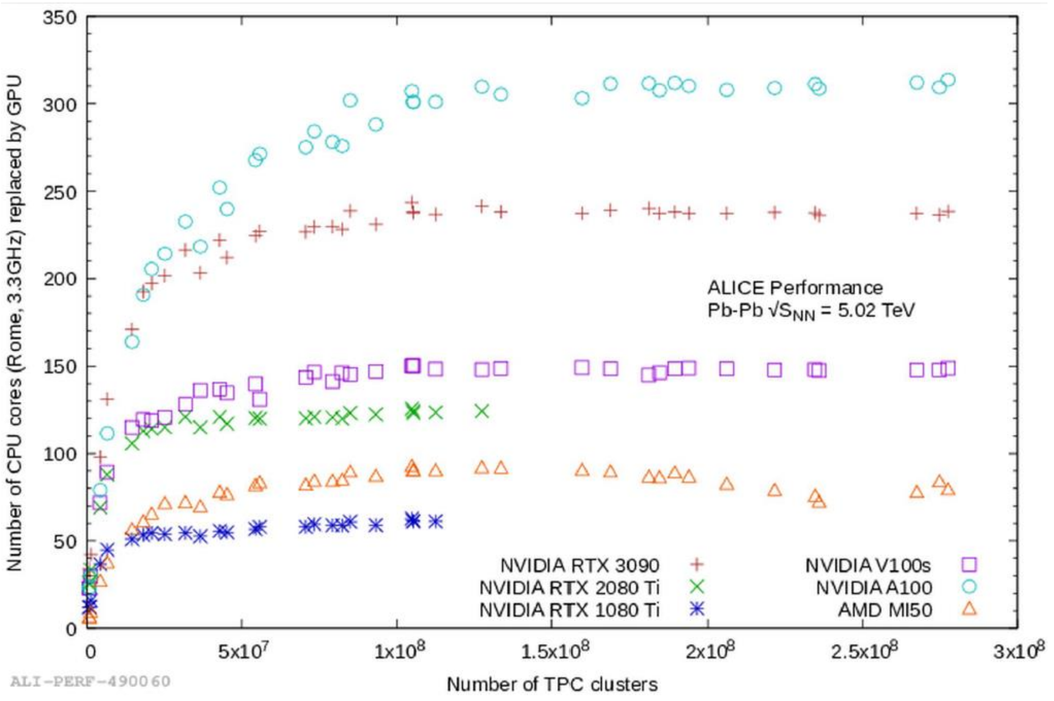


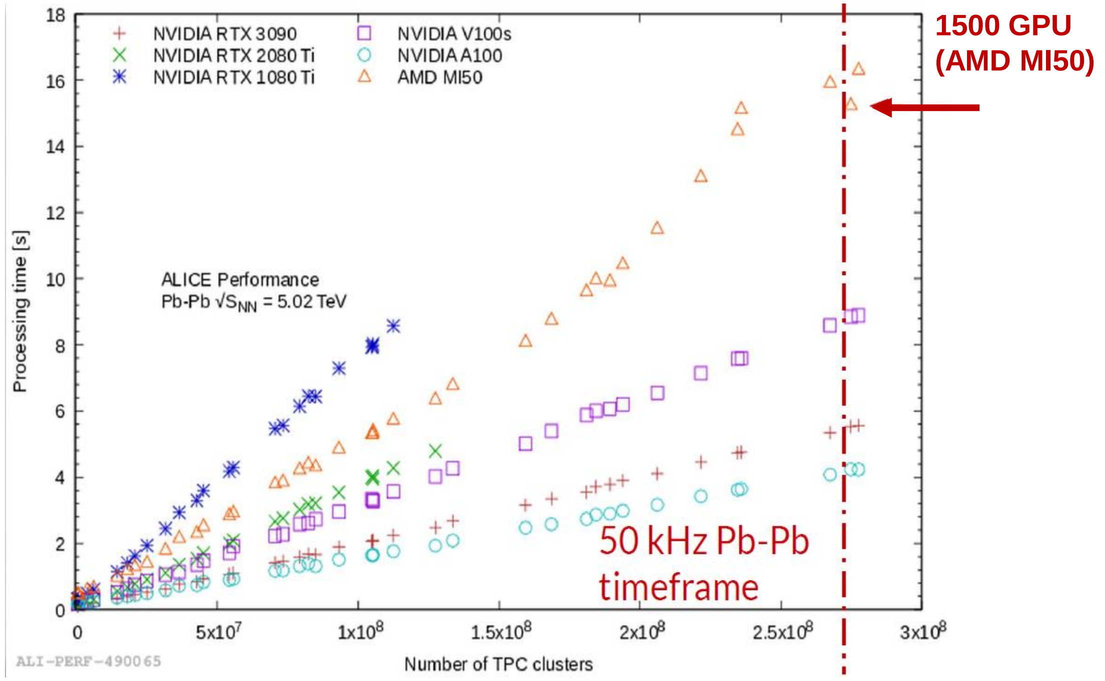
ALICE: GPU in Run3

- GPU can help both in sync and async phase
  - During synchronous phase 99% processing time dominated by TPC
    - Time frame/event definition, calibration, compression, …
    - Factor 20-25 speed-up
  - Other detectors reconstruction during asynchronous phase on EPN farm
    - TPC async (72%), TRD tracking (13%), TOF-TPC matching (10%)

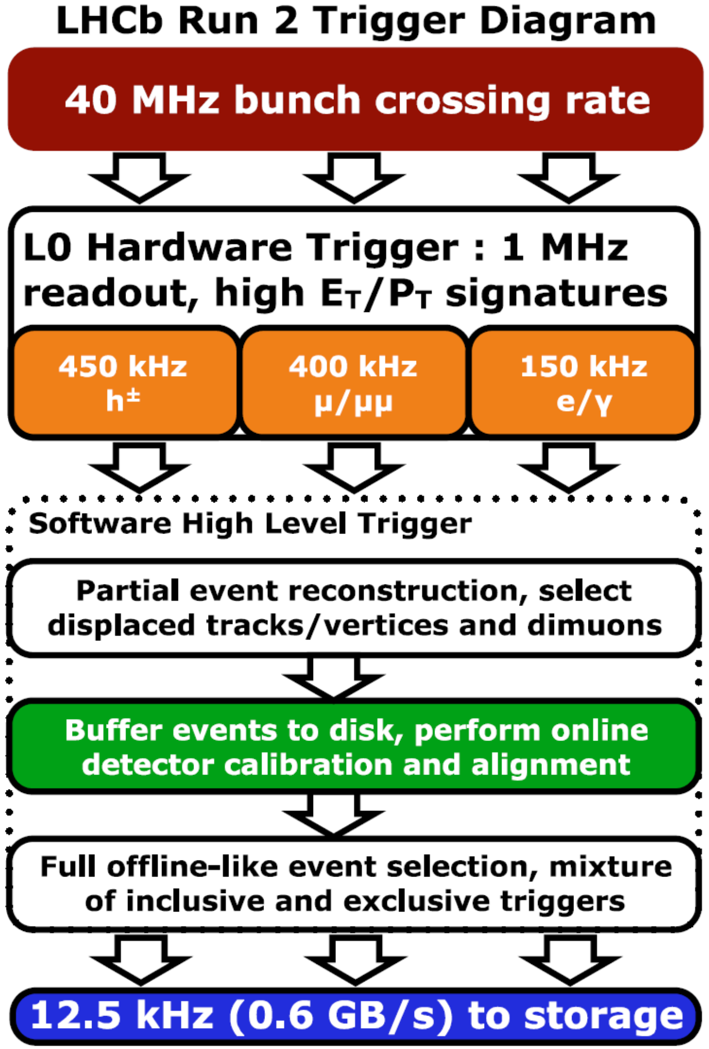
Task name	CPU Time [s]	GPU Time [s]
TPC sector track finding	706	11
TPC track merging	40	2
TPC track fit	300	6
TPC looping track following	150	6
TPC data track-based compression	100	2
Sum	1296	27
ITS clustering	10	
TPC-ITS track matching	1	
Global track matching to TRD	1	
Global track matching to TOF	1	
ITS tracking	10	
ITS tracklet vertexer (seeding)	1	
ITS (MFT) data compression	3	
TPC data entropy compression	35	
TPC gain calibration	10	
TPC distortions calibration with residuals	20	
Sum	92	
Total	1388	

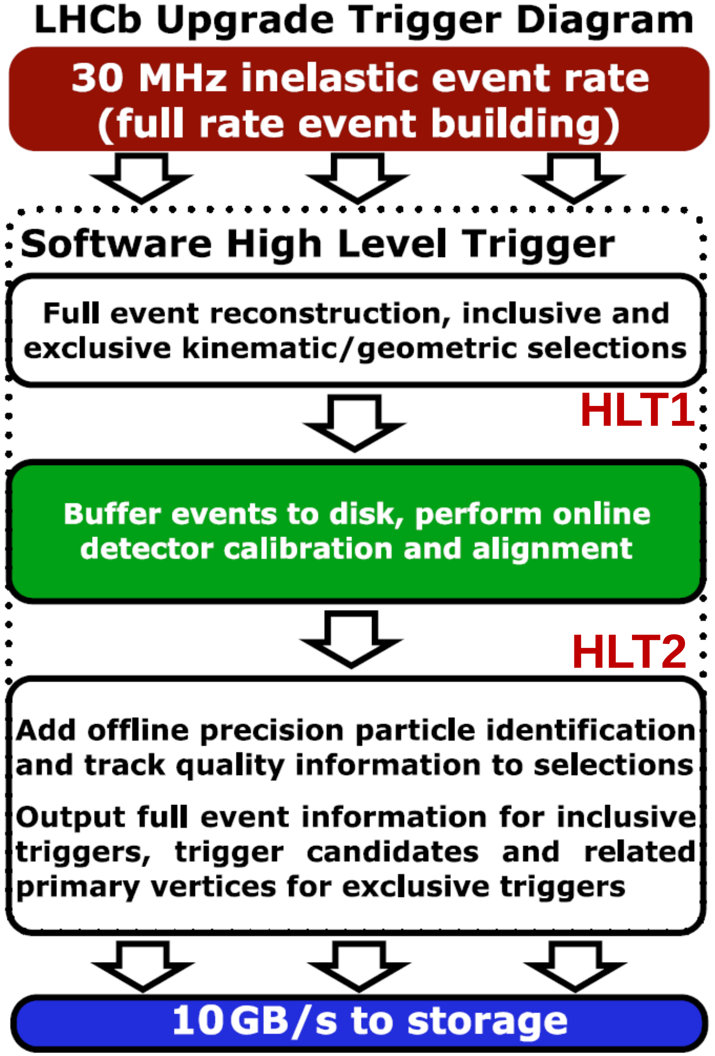
ALICE: GPU in Run3





- Several GPUs tested
  - One AMD MI50 GPU is equivalent to about 55 standard CPU cores
  - Computing farm consists of 250 servers with 8 AMD MI50 GPU each (and 2x32 cores Rome AMD-CPU and 512 GB ram)
  - Tested on full 50 kHz Pb-Pb collisions with a 20% margin

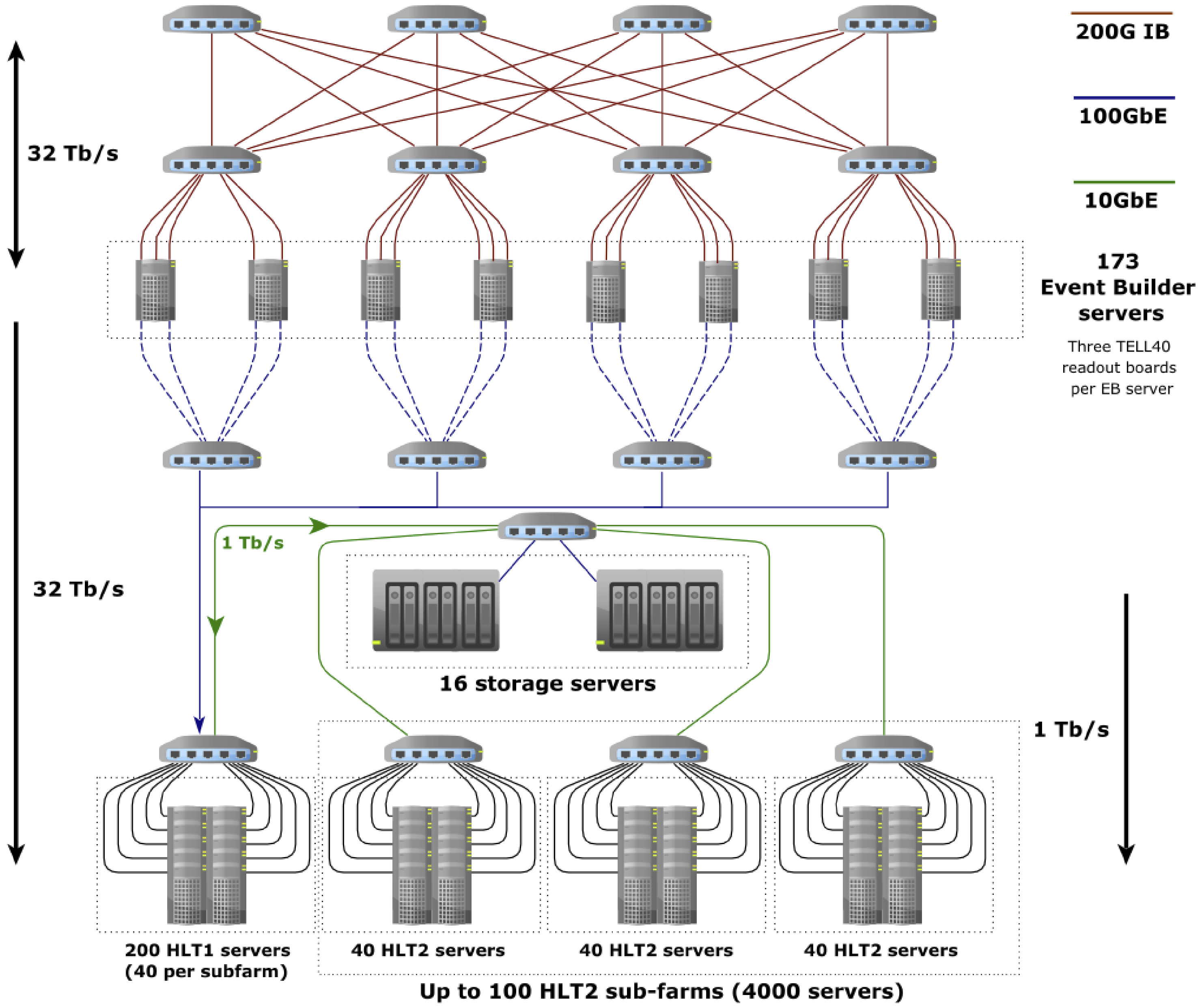




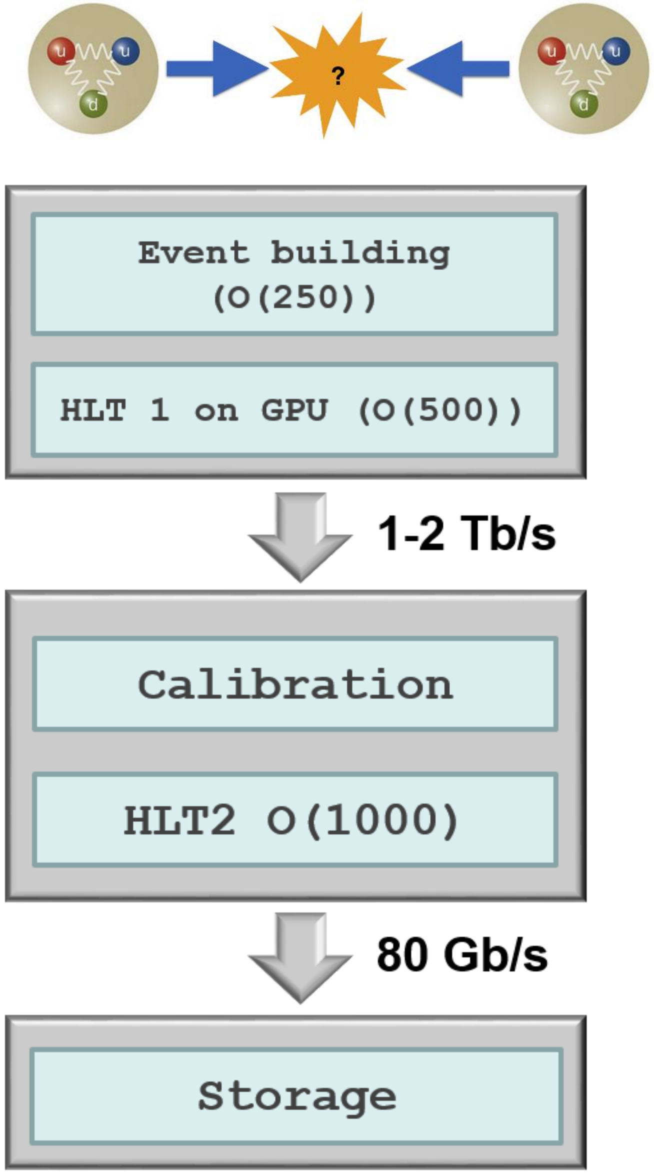
HLT1 and HLT2 are ideal places where to use GPUs!

- L0 Hardware Trigger removed
- 30x higher rate and 5x more pile-up
- HLT1:
  - Full charged track reconstruction (@30 MHz!!!!)
  - Reduce the rate by a factor 30x
- HLT2:
  - Detector calibration and offline track quality reconstruction
  - PID, vertices, exclusive triggers,…

LHCb: Standard Online Farm

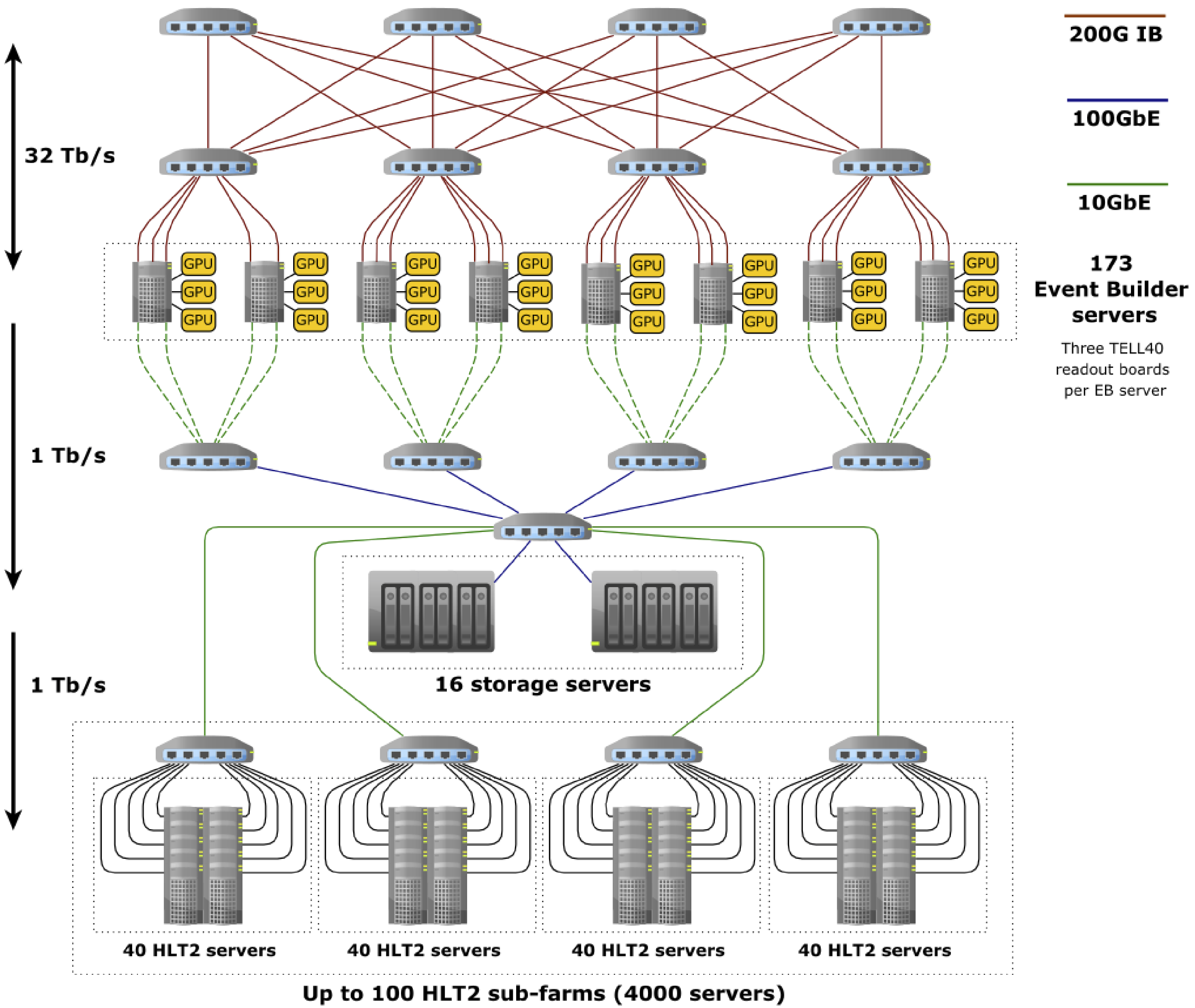


LHCb: the Allen project



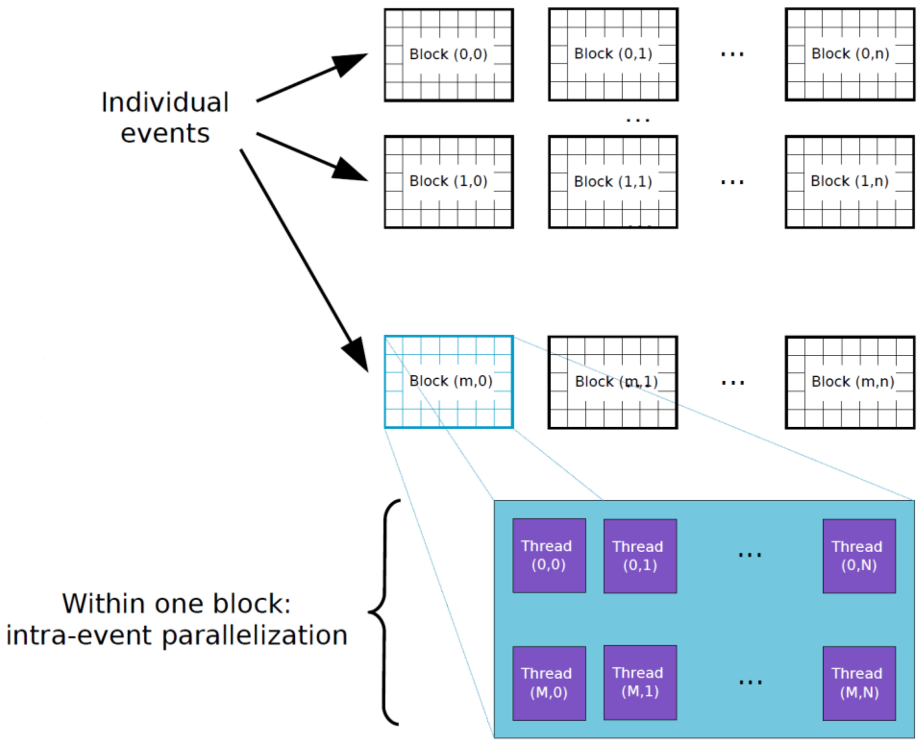
- Move the HLT1 before the switch, in the Event Bulding farm
  - Full reconstruction at HLT1
- Use GPU to increase the computing power
  - Natural Parallel processing on events
  - Parallelize algorithms
- Reduction of data bandwidth
  - From 32 Tb/s to 1-2 Tb/s
- Next step: use GPU in HLT2

LHCb: Online Farm with GPU@HLT1



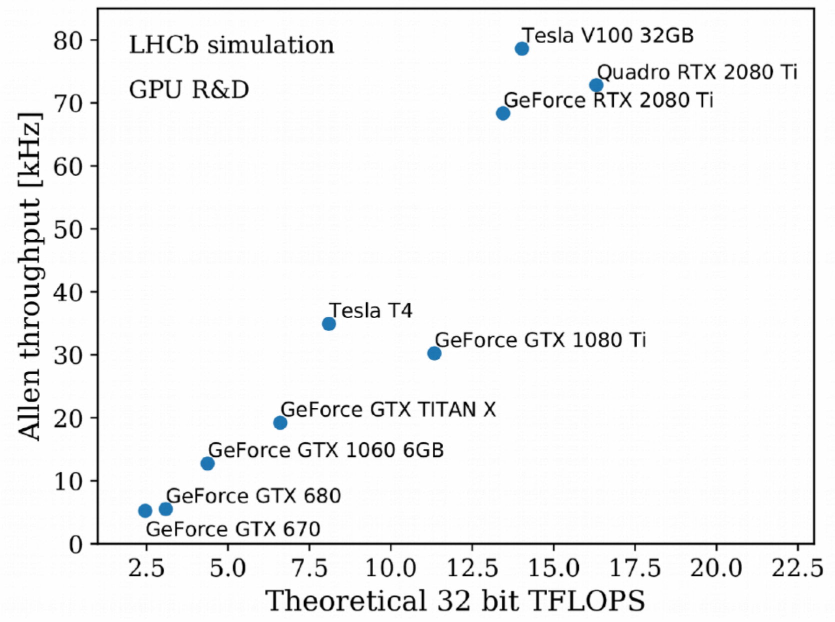
LHCb: the Allen project

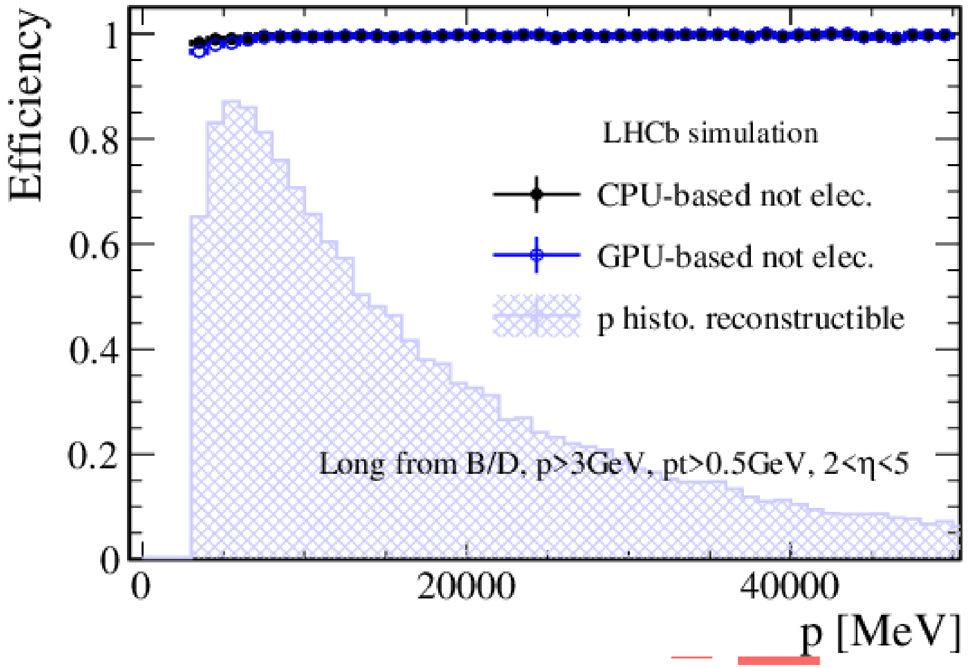
- All the HLT1 primitives produced on GPU
  - Velo: clustering, tracking, vertexing
  - UT: Tracks reconstruction
  - SciFi: Tracks reconstruction
  - Muon: particle identification
- Selection:
  - 1 Track, 2 Tracks, High-pt muons, muon identification, …
- Event rate reduced from 30 MHz to 1 MHz with physics performace consistent with TDR



LHCb: Allen performaces

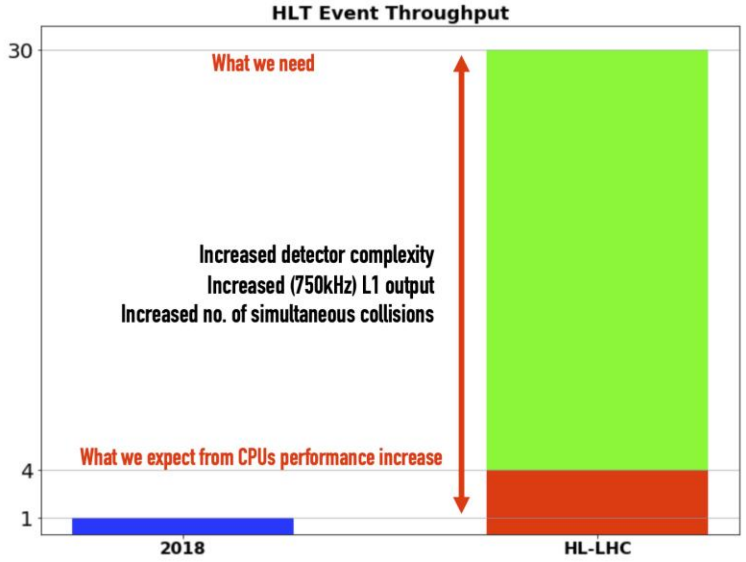
- Several GPUs tested
- Switch a 1 Tb/s commutate network is easier and cheaper with respect to a 40 Tb/s
- Runs on ~250 servers with ~500 A5000 GPU
  - Allen software support CPU, CUDA and HIP capable GPUs
- Reconstruction efficiency pratically equal between CPU and GPU version

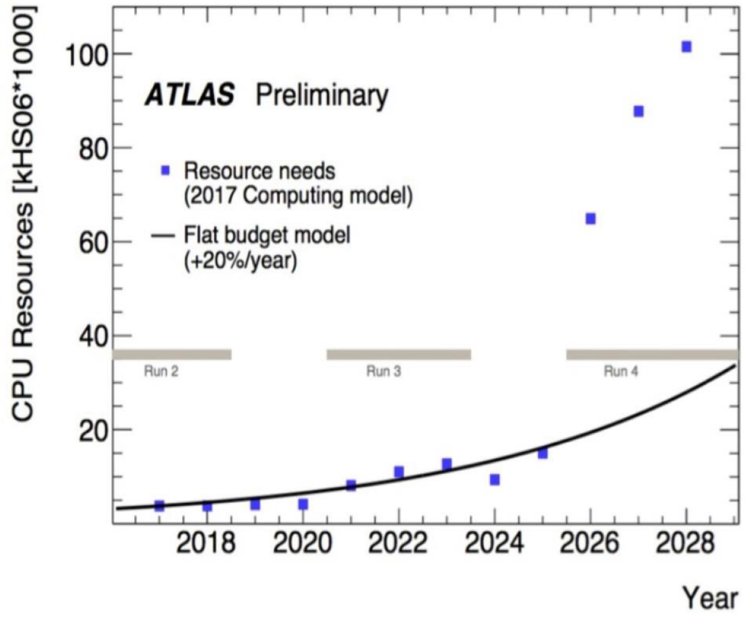




CMS: heterogenous computing in the high level trigger

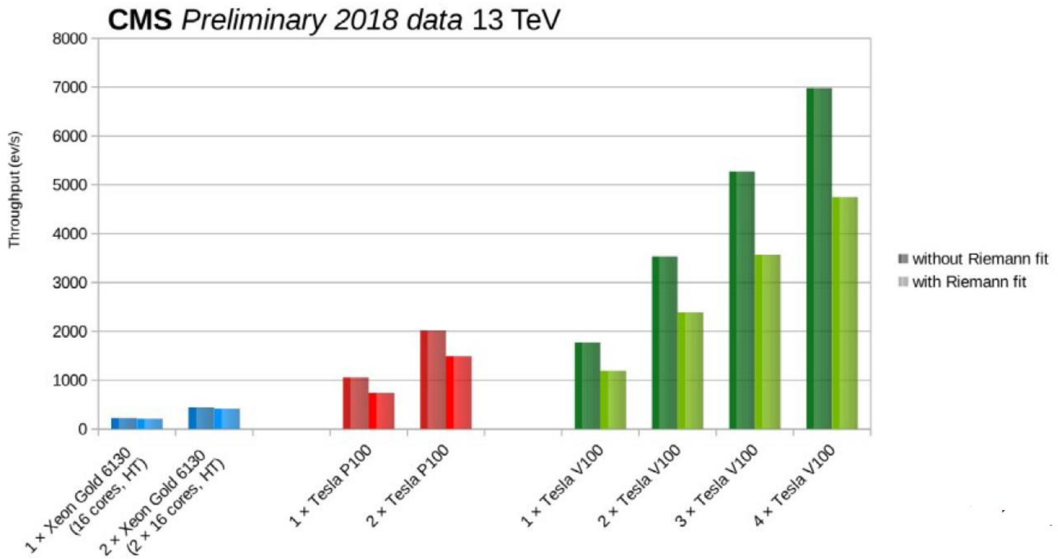
- In HL-LHC era CMS expects 20x computing load in HLT
  - ~1.3x from detectors upgrade, ~3x from higher pile-up, ~7.5x from event rate
- The foreseen CPUs increase in performance can account only for a 4x
  - Similar for ATLAS
- Heterogeneous computing (with GPU and other co-processors) can be a solution
  - The most important constraint is the computing time per event in HLT



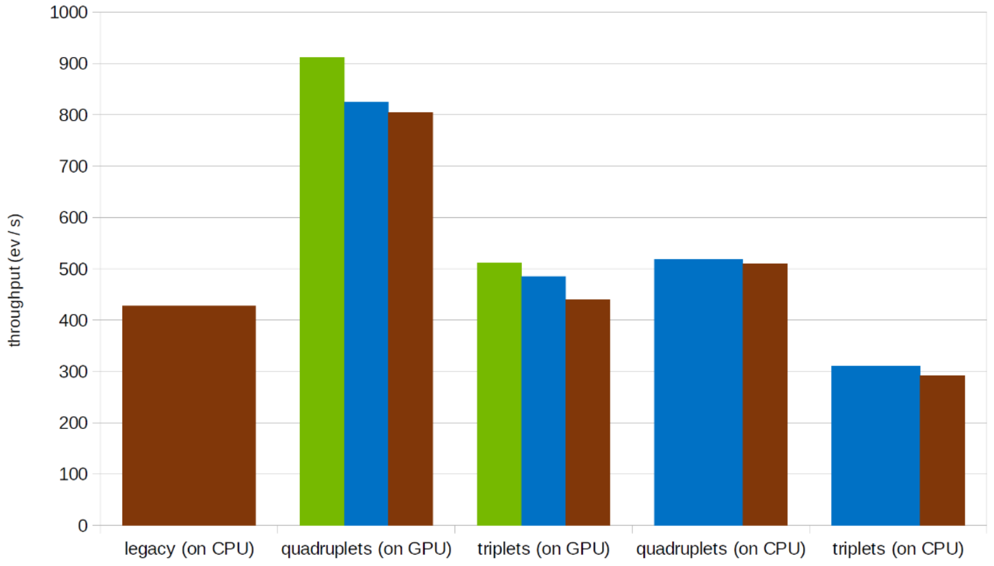


CMS: Patatrack Throughput performances

- CPU
  - Dual socket Xeon Gold 6130
  - 2x16 cores
  - 4 jobs with 16 threads
- GPU
  - Single NVIDIA Tesla T4 (2560 cuda cores)
  - 10/16 concurrent events

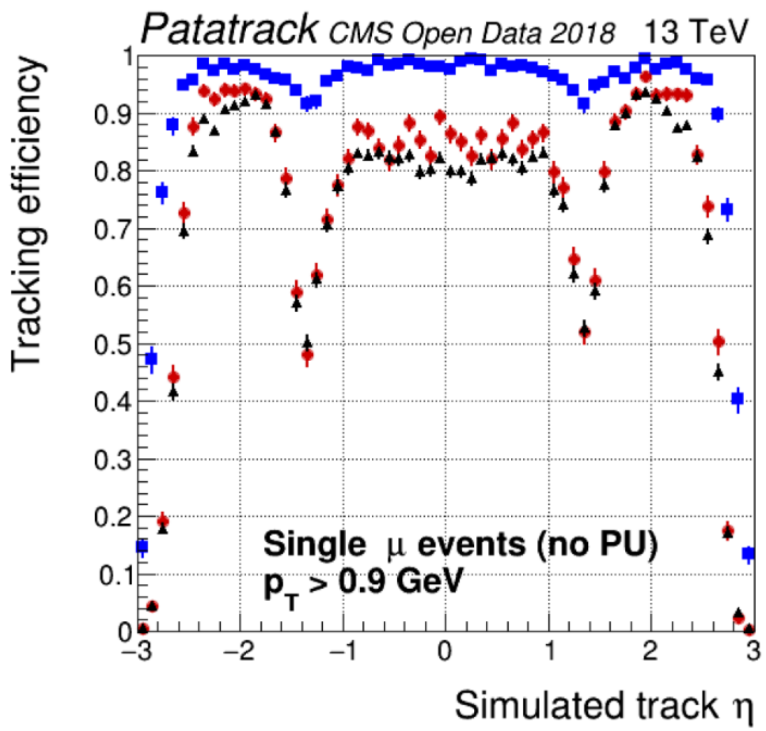


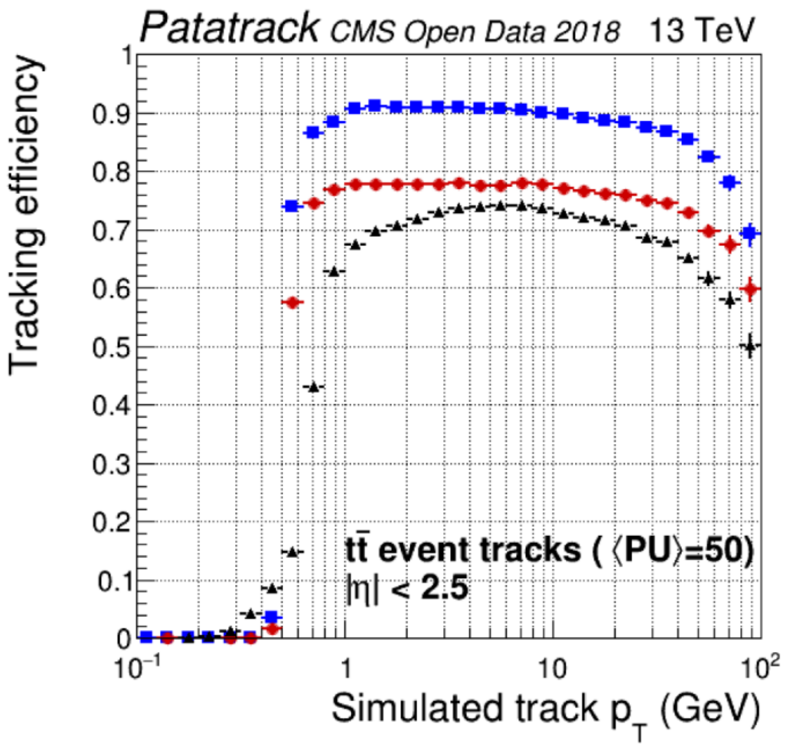
Tested during Run2 data taking



CMS: GPU in RUN3

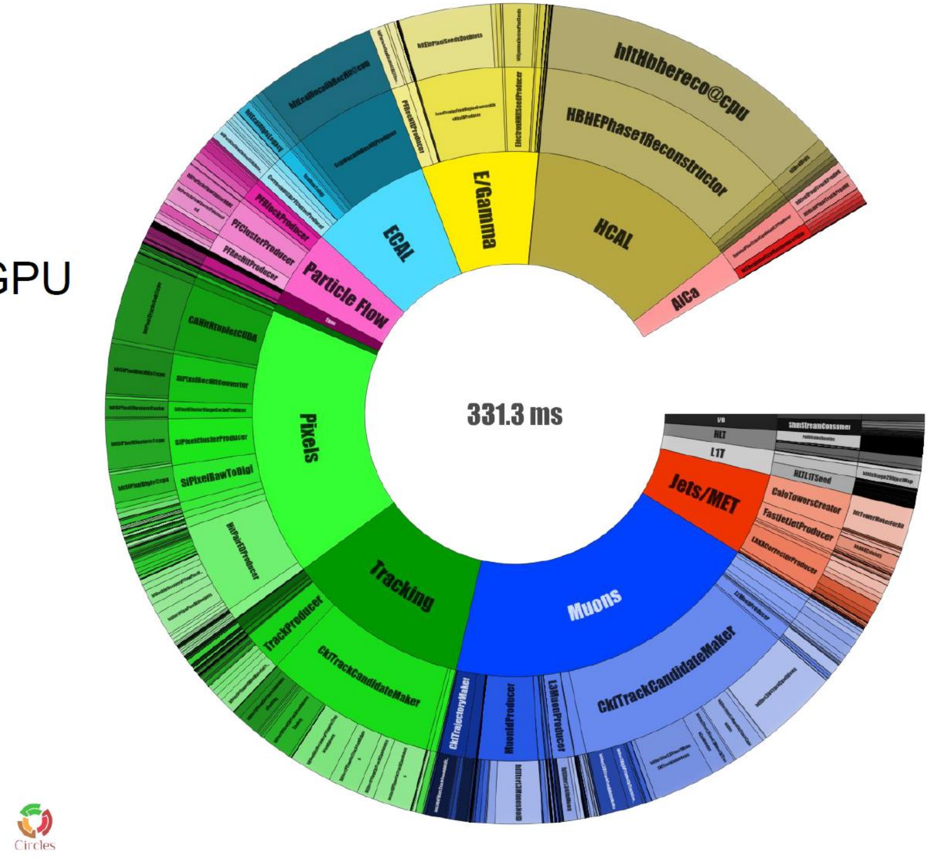
- Pixel tracking on GPU ready for HLT in Run3 (Patatrack)
  - Reconstruction of tracks and vertices in the pixels detector
- Offload various steps of the reconstruction algorithm on GPU
  - Cellular automaton
  - Improve the fitting quality exploiting the GPU computing power
- Use the CPU for interaction with the software framework



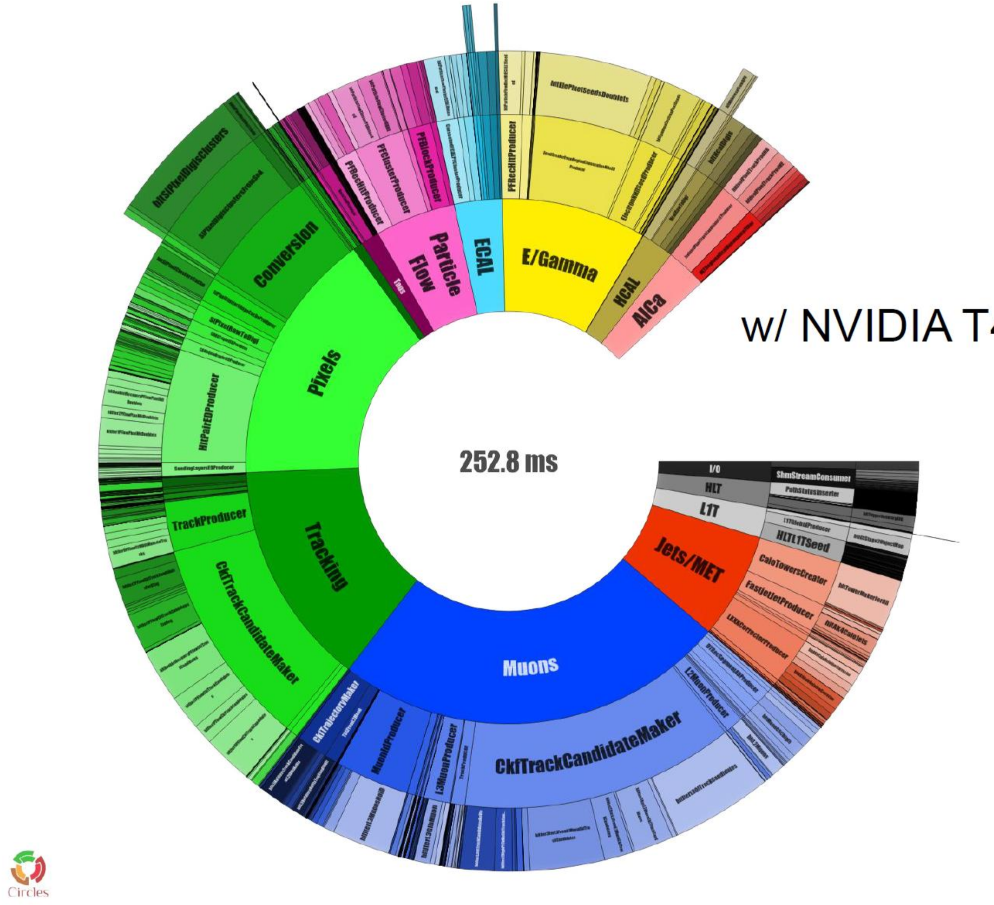


CMS: GPU timing results

w/o GPU



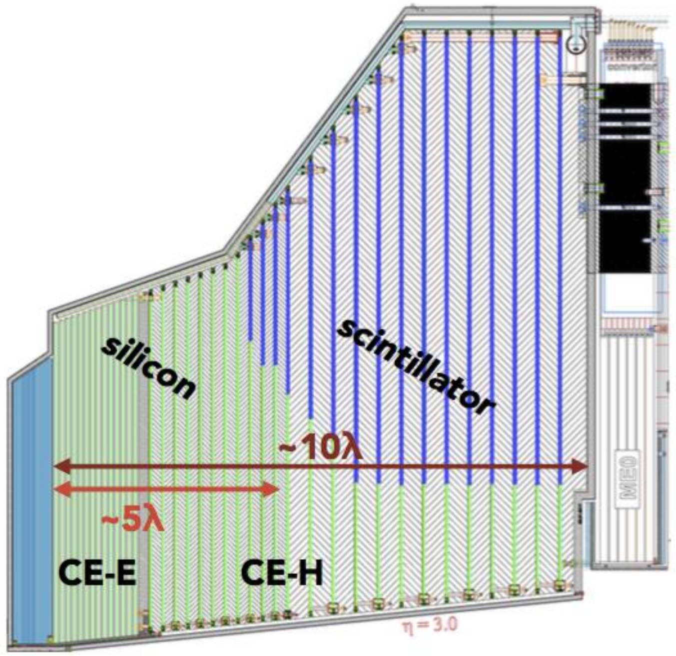
w/ NVIDIA T4 GPU

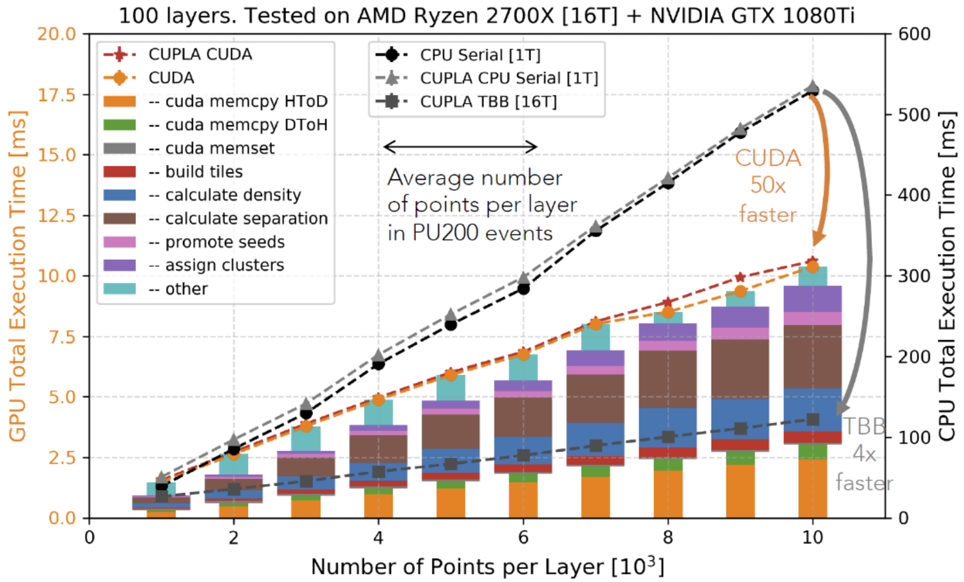


About 22% of GPU off-loading only for pixel tracking

CMS: clustering on GPU

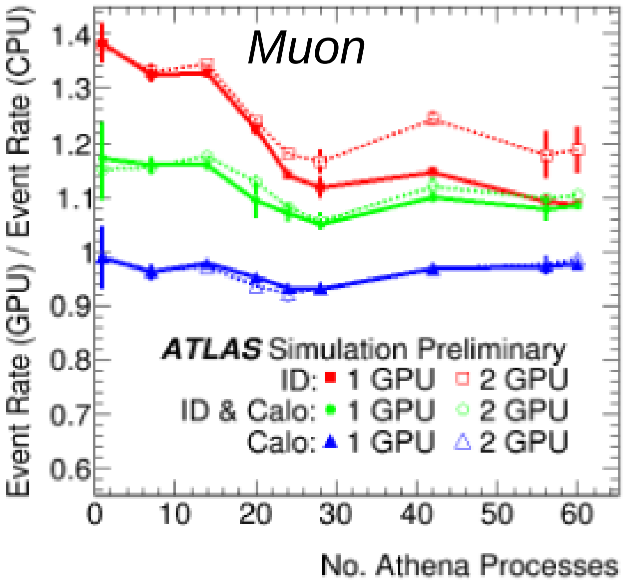
- Clustering by Energy (CLUE) on GPU:
  - Both electromagnetic and hadronic
- New algorithm designed for the HGCAL
  - Parallelizable 5 steps algorithm
- CLUE on CPU is a factor ~30x faster than the present clustering algorithm
- CLUE on GPU is an additional factor 6x
  - Factor 30x if exclude data transfer time
  - The data transfer time can be reduced by using streams and multiple GPUs
- Vertex reconstruction is another HLT component that can be offloaded to GPU

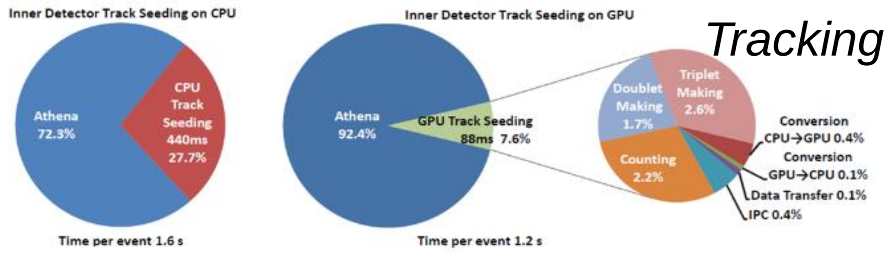


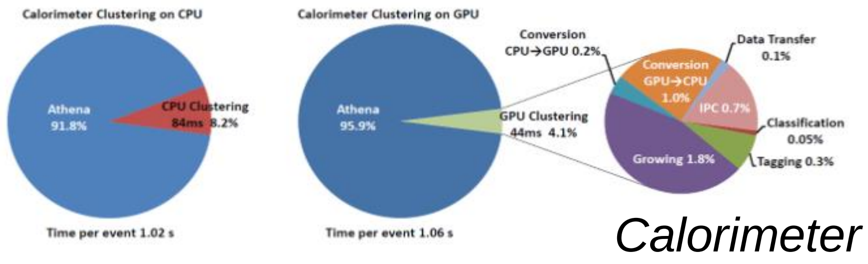


ATLAS: GPU in HLT in Run1

- Demonstrators in Run1
- Accelerator Process Extension(APE) Framework
- Inner Detector, tracking based on Cellular Automata(CA)
- Calorimeter, jet finding and clusterization based on CA
- Muon, tracking based of hough transforms
- Best result: x28 in tracking seeding algorithm







ATLAS: GPU for the future

- The conclusion of this study was <u>not to use the GPU in ATLAS</u>
  - The gain was marginal
- The reason is related to the use of "Athena", the ATLAS software that was not able to manage concurrency and multithreading
- New studies are on going to study the interaction of aynchrounous run of heterogeneous accelerators in the "Athena MT" framework (based on TBB)

- Recent effort in ATLAS to include accelerators support in software by using AthenaMT/Gaudi vs OpenACC, SYCL, OpenCL, Cuda,…
- Search for Hardware Trigger Tracking alternative is motivating a new round of GPU devolpment in ATLAS

Low level trigger:Different Solutions

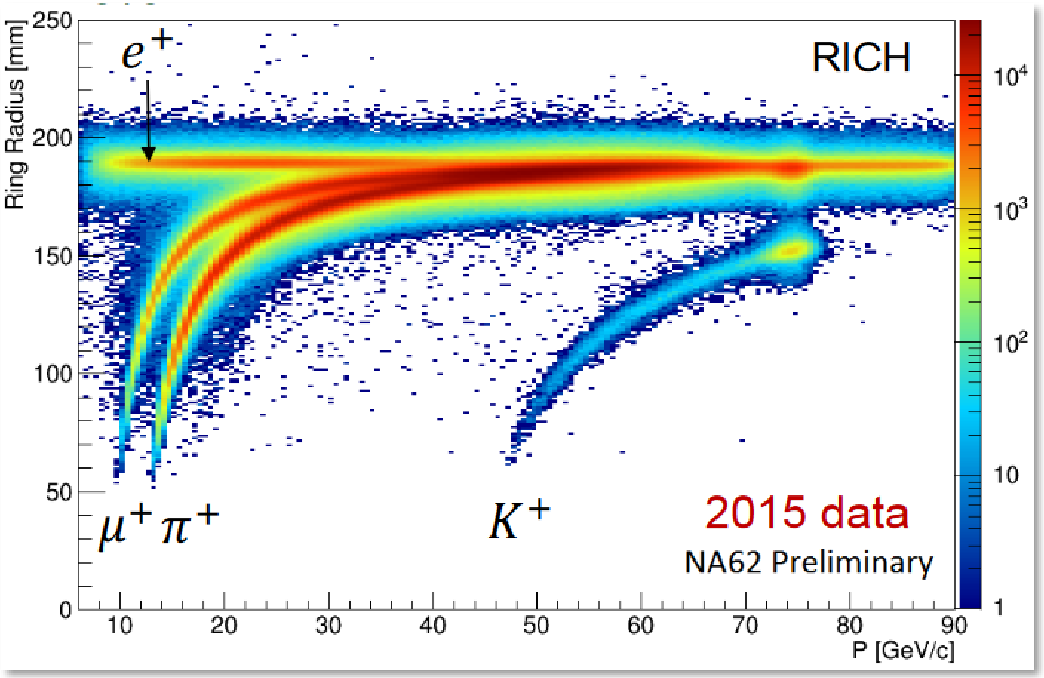
- Brute force: PCs
  - Bring all data on a huge pc farm, using fast (and eventually smart) routers.
  - Pro: easy to program, flexibility; Cons: very expensive, most of resources just to process junk.
- Rock Solid: Custom Hardware
  - Build your own board with dedicated processors and links
  - Pro: 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

- 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?

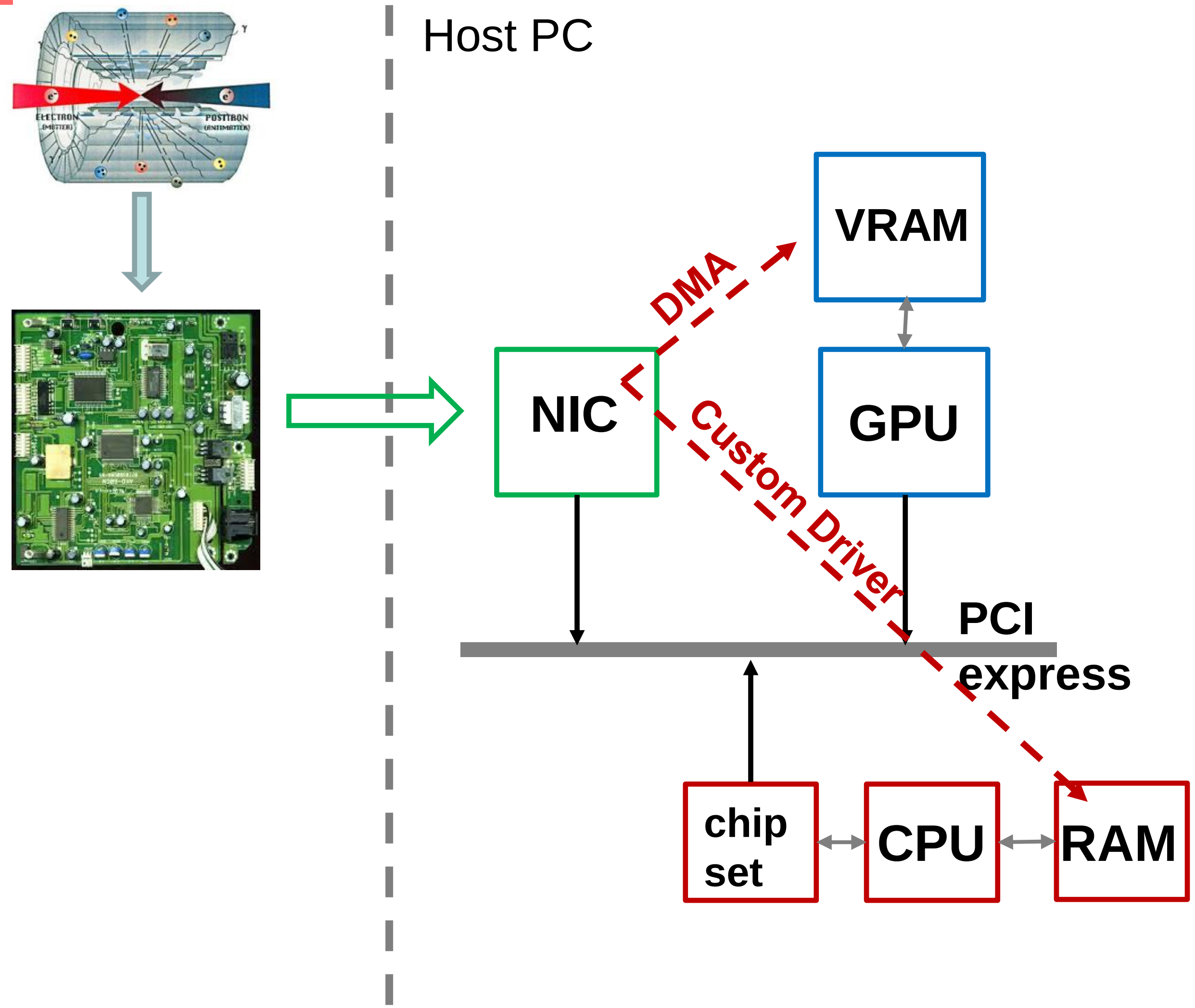
- Computing power: Is the GPU fast enough to take trigger decision at tens of MHz events rate?

Low Level trigger: NA62 Test bench



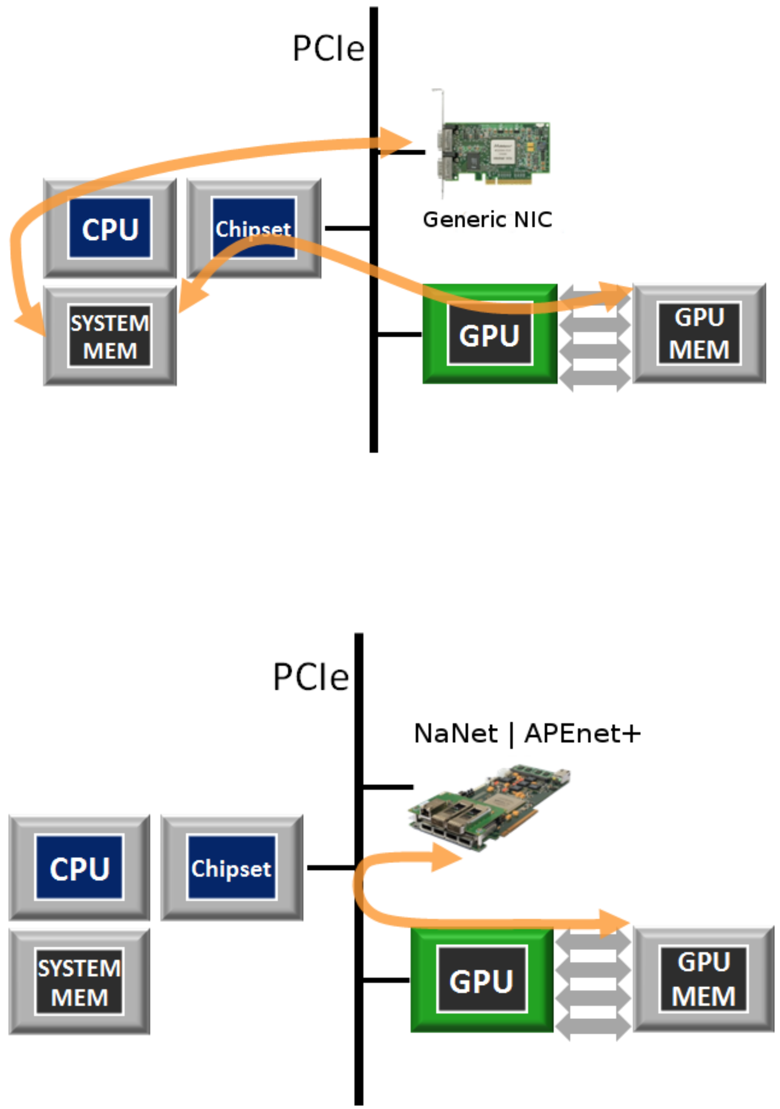
- NA62:
  - Fixed target experiment on SPS (slow extraction)
  - Look for ultra-rare kaon decays (K→πνν)
- RICH:
  - 17 m long, 3 m in diameter, filled with Ne at 1 atm
  - Reconstruct Cherenkov Rings to distinguish between pions and muons from 15 to 35 GeV
  - 2 spots of 1000 PMs each
  - Time resolution: 70 ps
  - MisID: $5x10^{-3}$
  - 10 MHz events: about 20 hits per particle

Latency: main problem of GPU computing



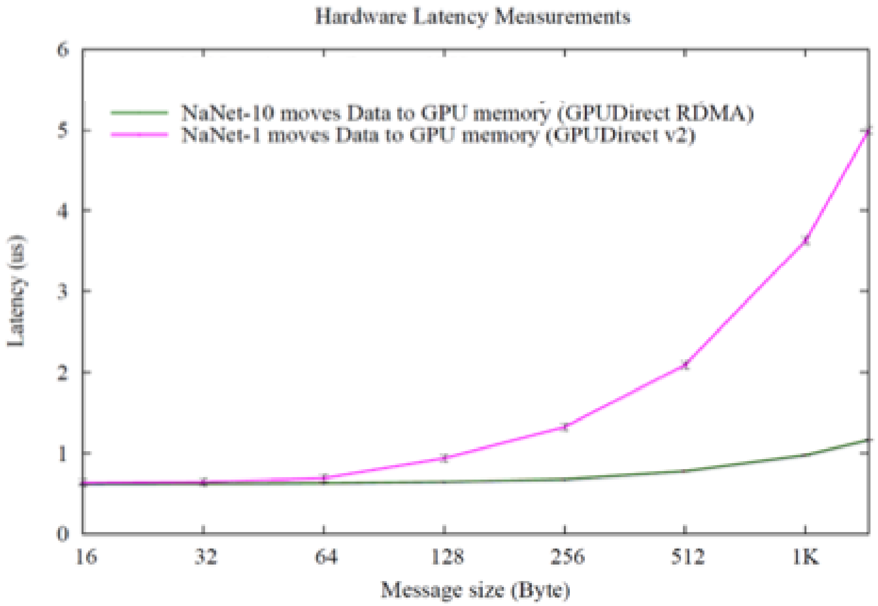
- Total latency dominated by double copy in Host RAM
- Decrease the data transfer time:
  - DMA (Direct Memory Access)
  - Custom manage of NIC buffers
- *"Hide"* some component of the latency optimizing the multi-events computing

NaNet-10

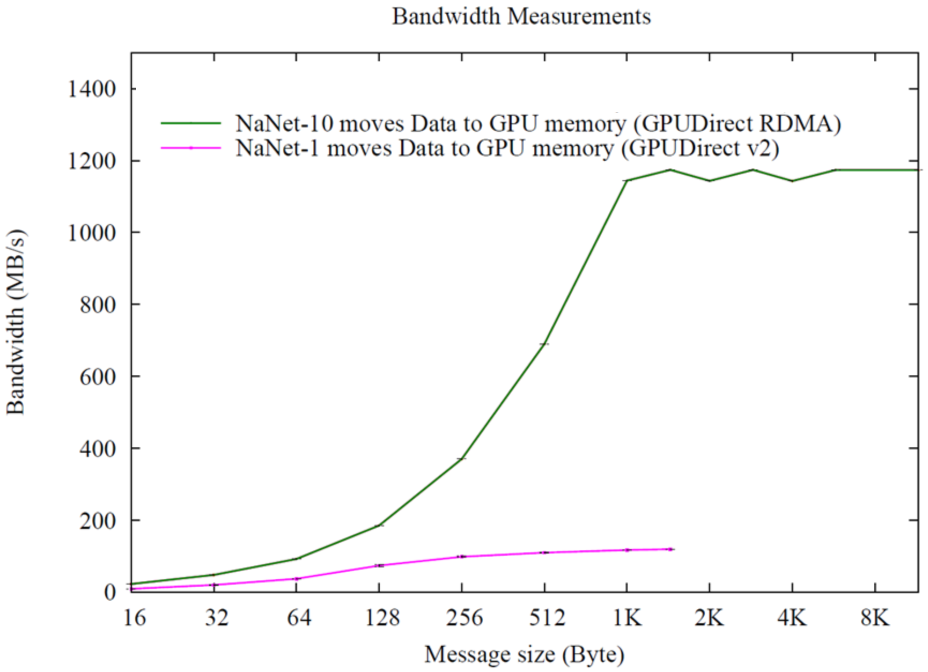


- ALTERA Stratix V dev board (TERASIC DE5-Net board)
  - PCIe x8 Gen3 (8 GB/s)
  - 4 SFP+ ports (Link speed up to 10Gb/s)
- GPUDirect /RDMA
- UDP offload support
- 4x10 Gb/s Links
- Stream processing on FPGA (merging, decompression, …)
- Working on 40 GbE (foreseen 100 GbE)

NaNet-10

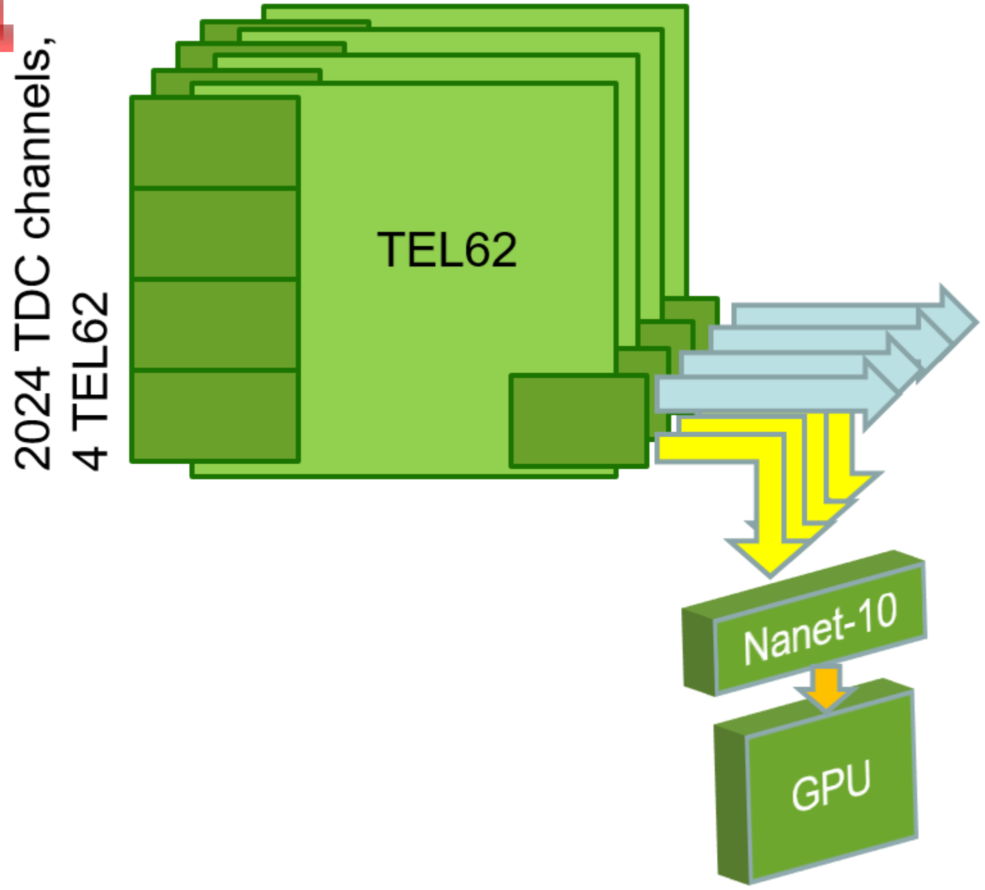


Hardware Latency Measurements
NaNet-10 moves Data to GPU memory (GPUDirect RDMA)
NaNet-1 moves Data to GPU memory (GPUDirect v2)
Latency (us)
0
1
2
3
4
5
6
16
32
64
128
256
512
1K
Message size (Byte)



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

NA62 GPU trigger system



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

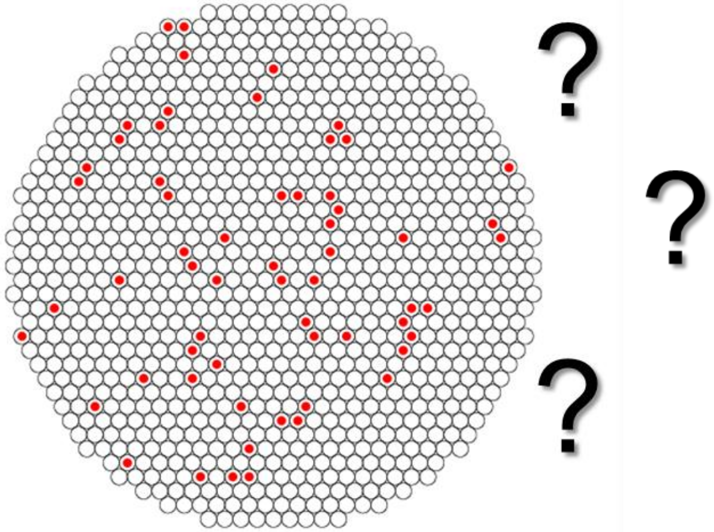
8x1Gb/s links for data readout
4x1Gb/s Standard trigger primitives
4x1Gb/s GPU trigger

GPU NVIDIA K20:
- 2688 cores
- 3.9 Teraflops
- 6GB VRAM
- PCI ex.gen3
- Bandwidth: 250 GB/s

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

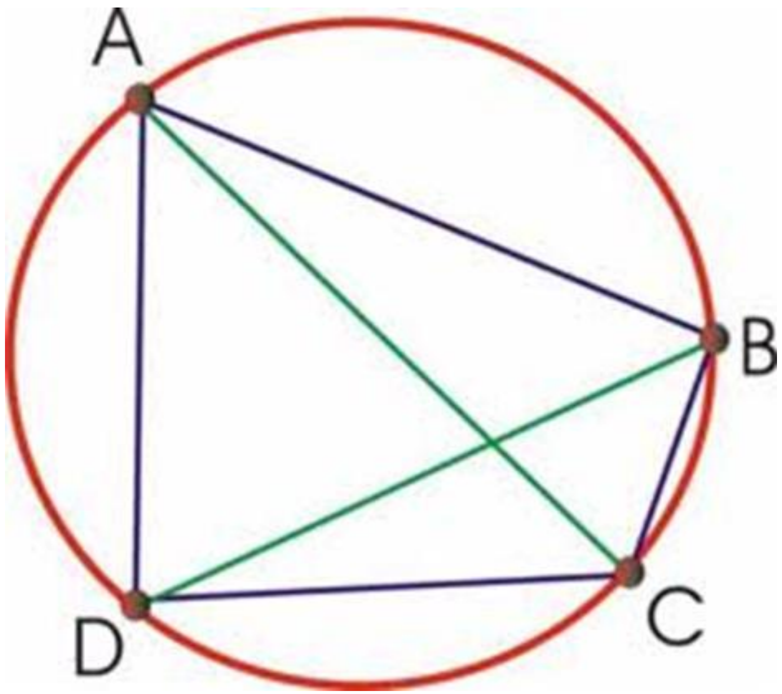
Ring fitting problem

- 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

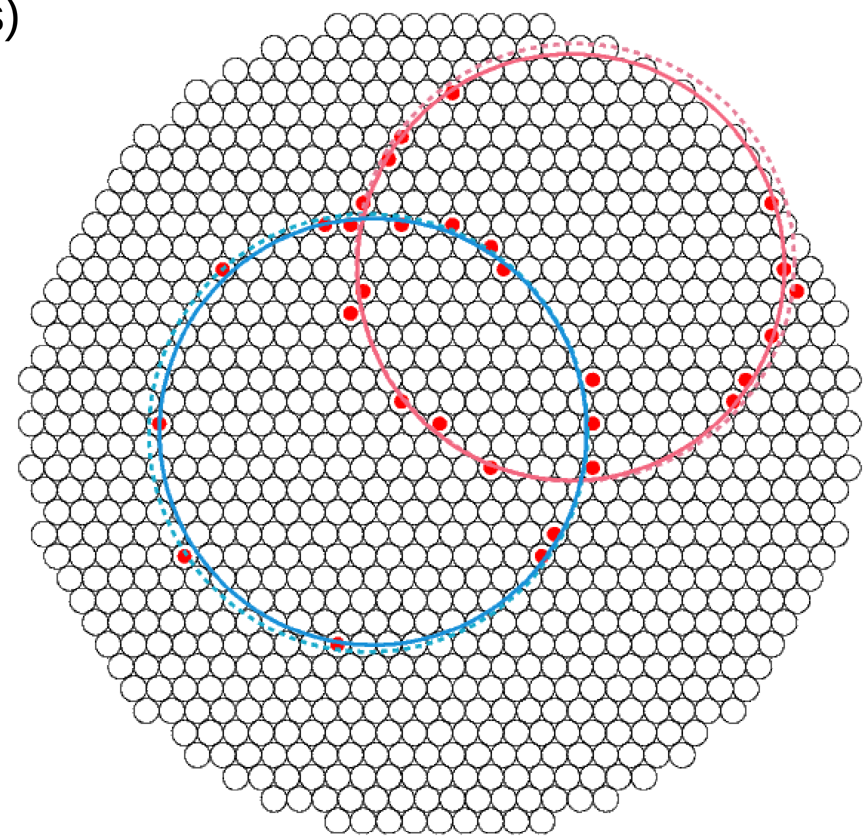


- New algorithm (Almagest) based on Ptolemy's theorem: *"A quadrilater is cyclic (the vertex lie on a circle) if and only if is valid the relation: AD*BC+AB*DC=AC*BD "*
- Design a procedure for parallel implementation

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

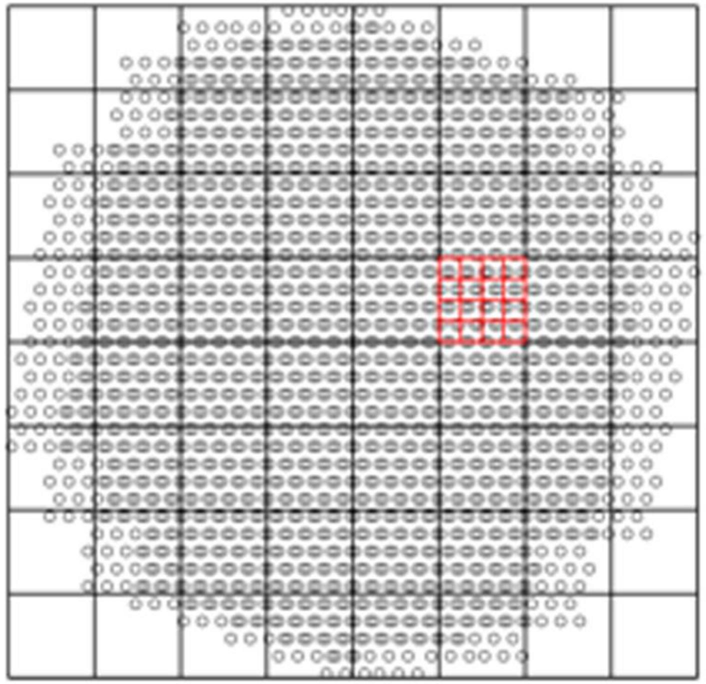


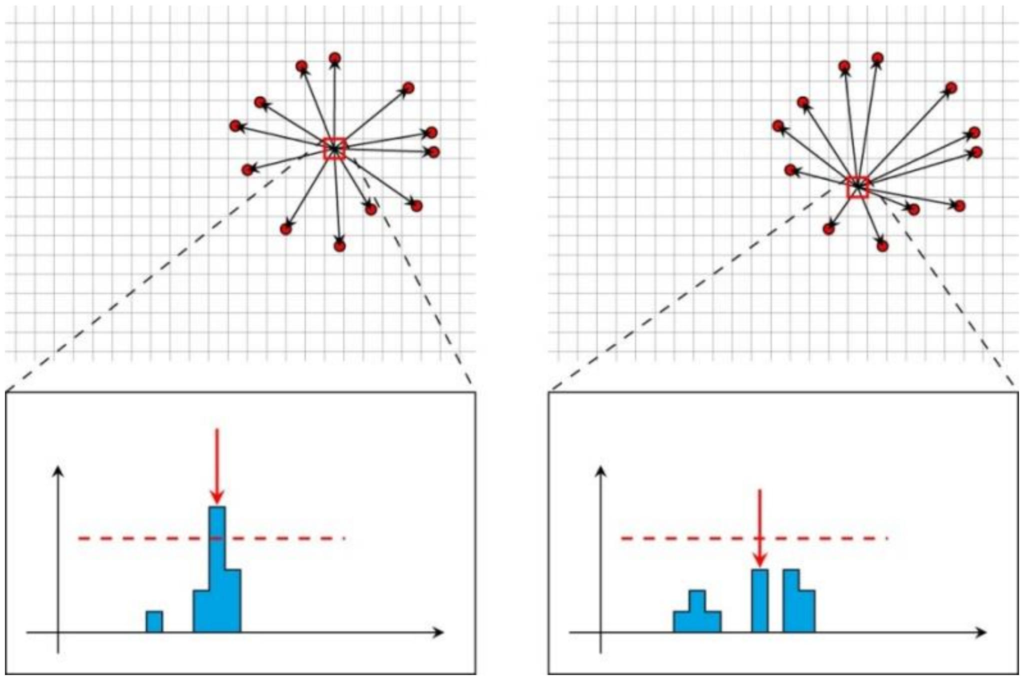
iv) …again…

v) Perform a single ring fit

vi) Repeat by excluding the already used points

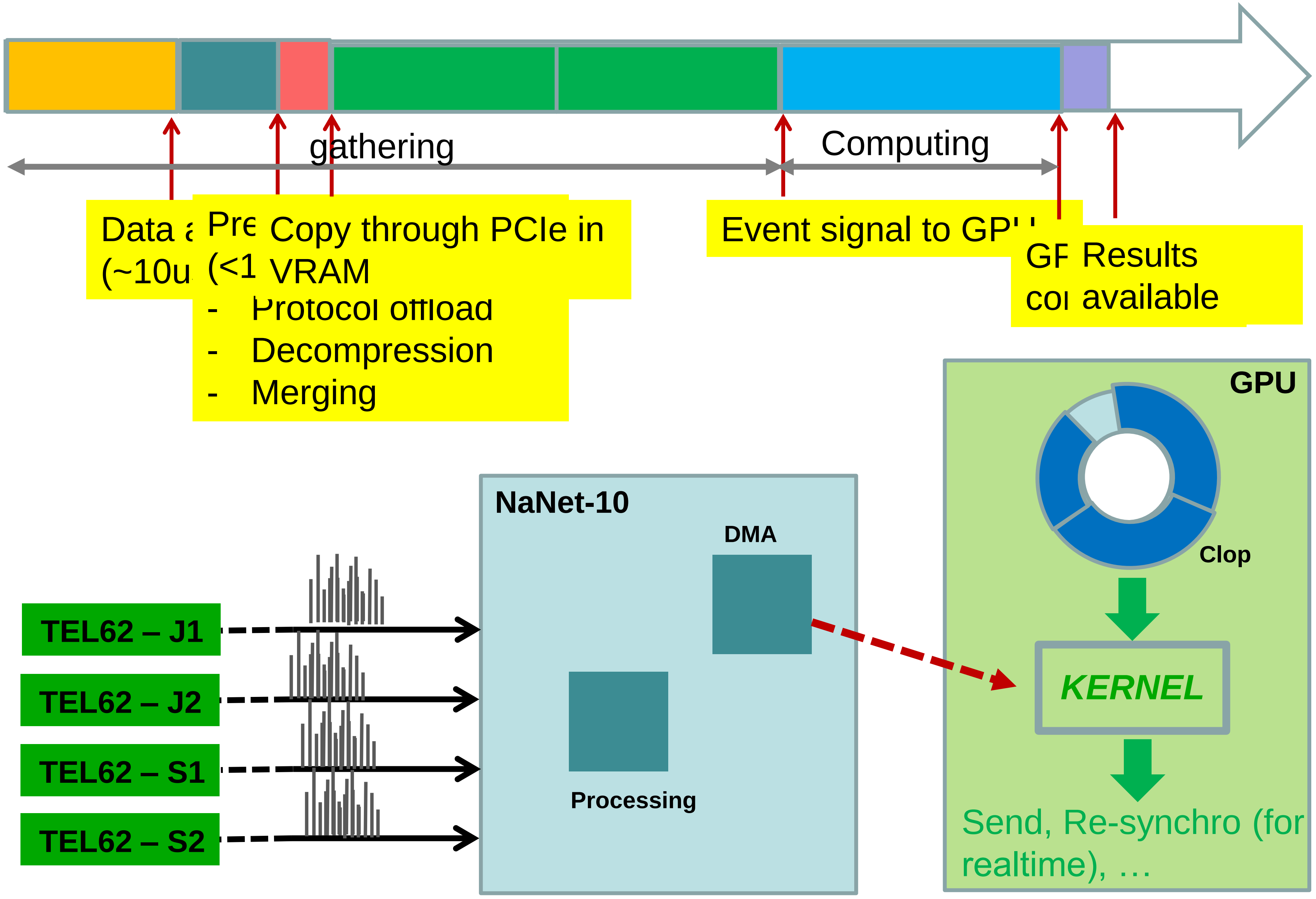
Histogram





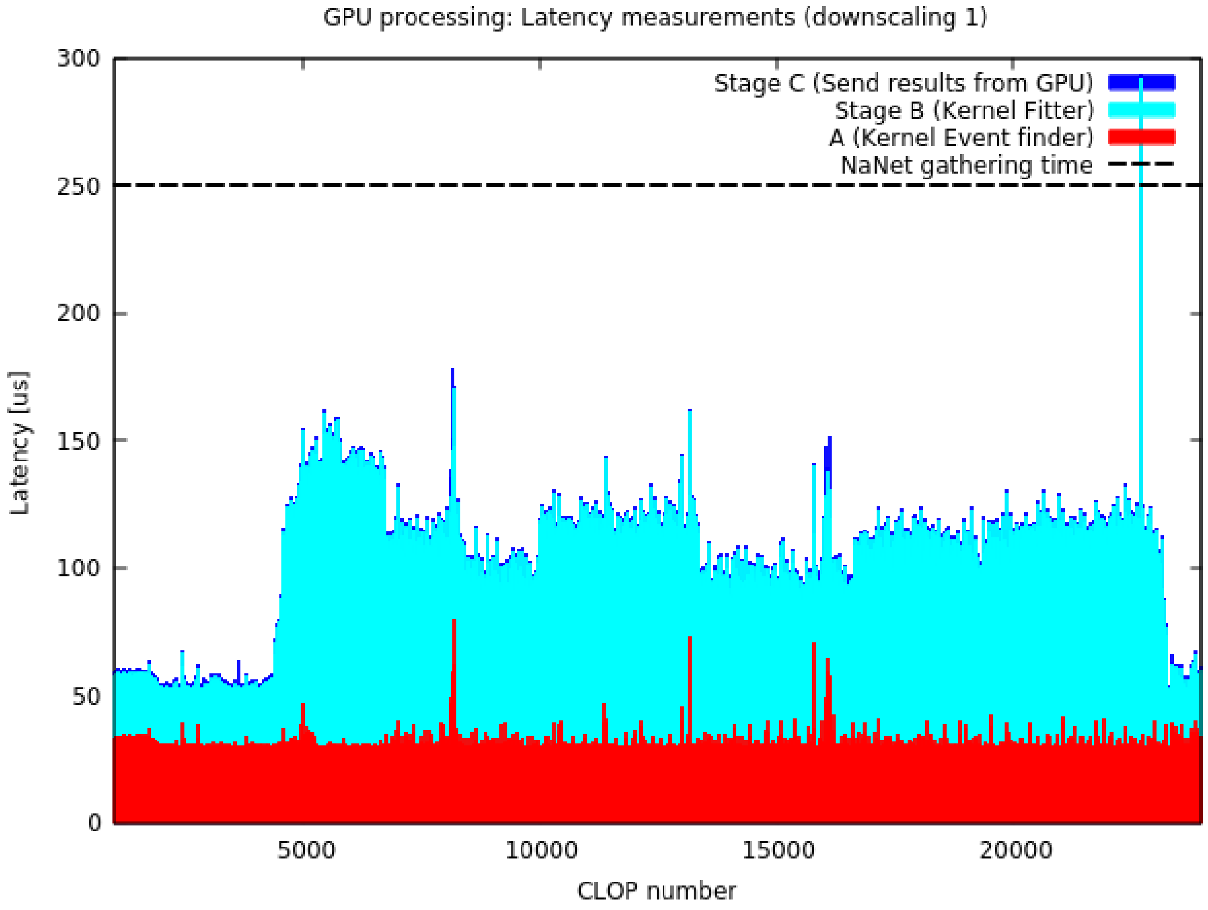
- The XY plane is divided in a Grid
- The histograms of the distances is created for each point in the grid

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)

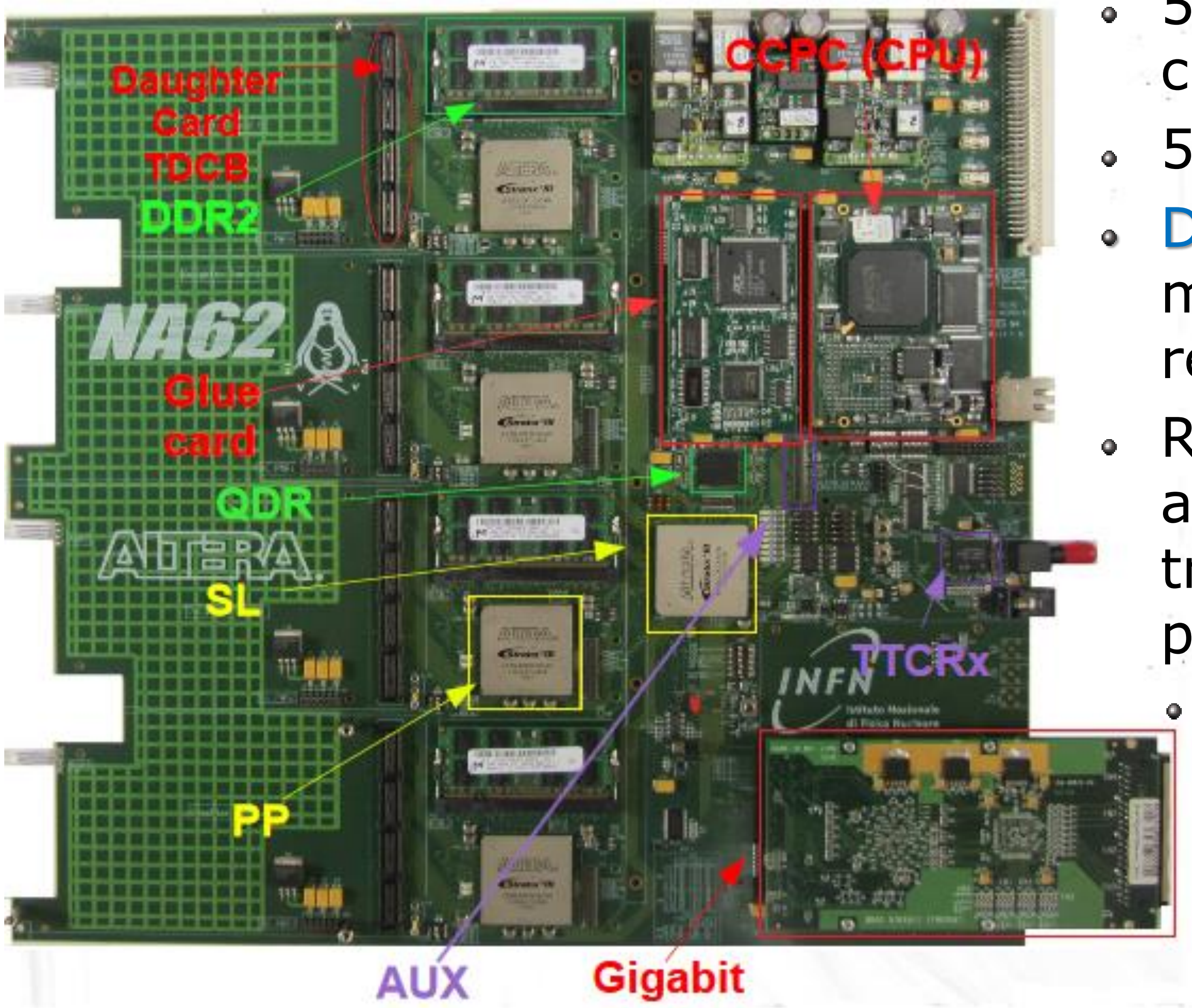
Where to use GPU in HEP?

- 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 generators
  - Fast linear algebra
- DNN and ML
  - Training and (maybe) inference
  - Data quality
  - Jet reconstruction

Conclusions

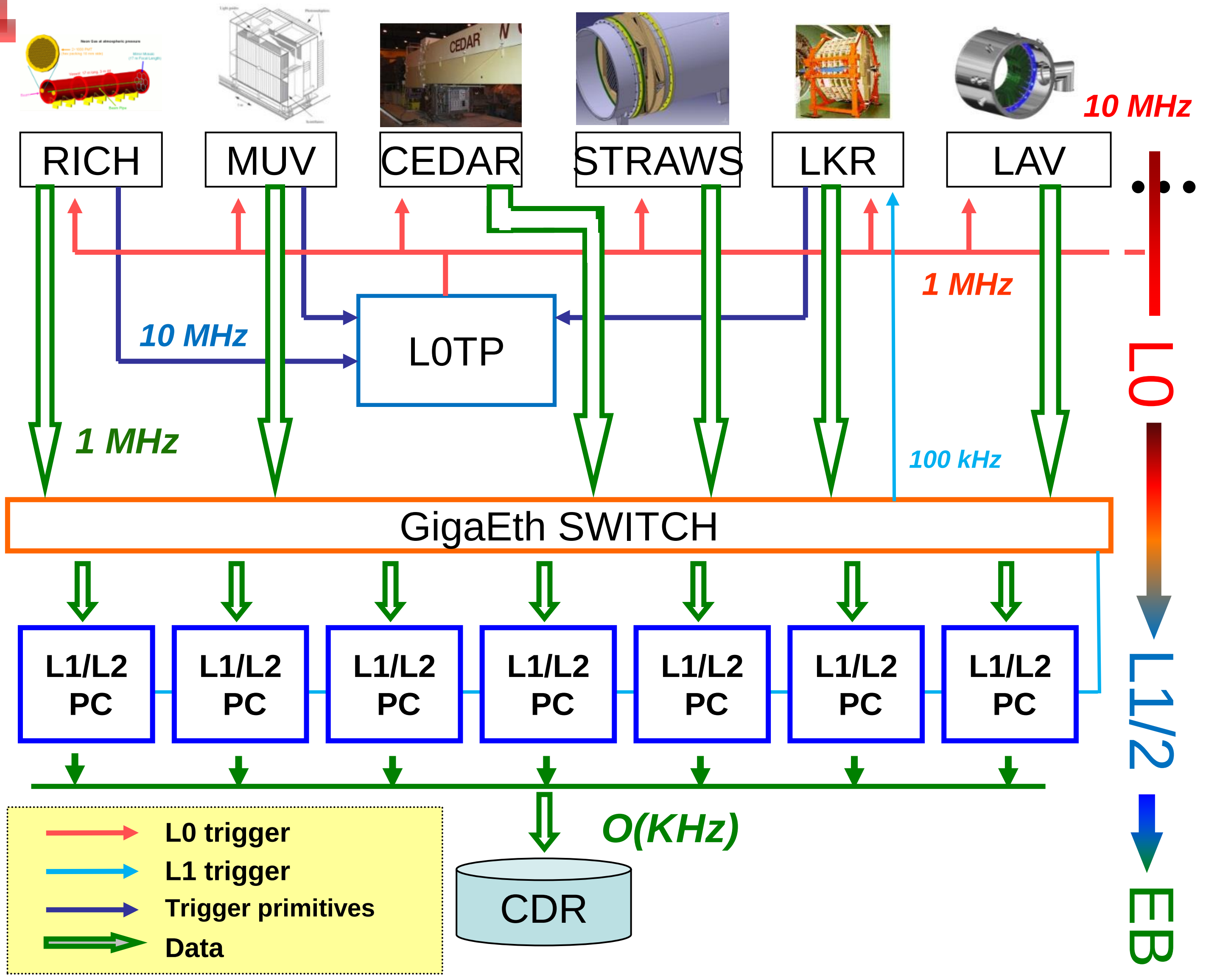
- be

SPARES



- 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.

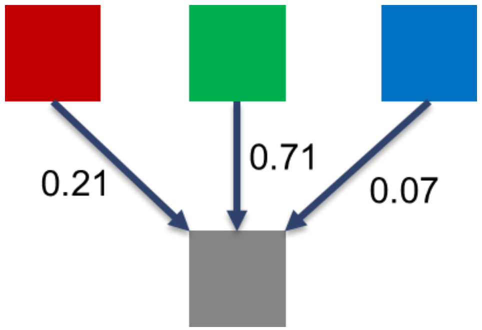
Example: RGB to gray scale conversion

- Assume you want to convert an image in which you have the rgb code for each pixel in greyscale
  - Rgb is a standard to define the quantity of red, green and blue in each pixel
  - A greyscale image is an image in which the value of each pixel carries only intensity information.

Example: RGB to grayscale conversion

- Conversion formula: For each pixel (I, J) do:

grayPixel[I,J] = 0.21*r + 0.71*g + 0.07*b

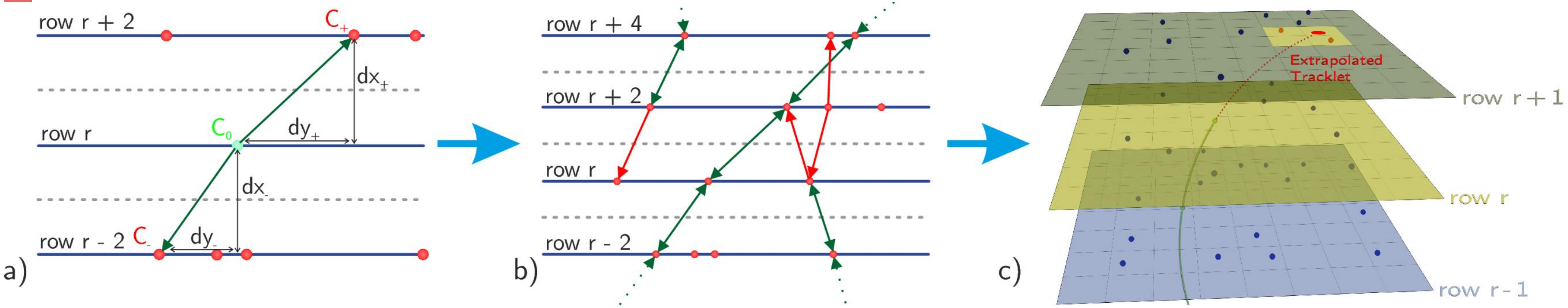



```
                  // we have 3 channels corresponding to RGB
// The input image is encoded as unsigned characters [0, 255]
__global__ void colorConvert(unsigned char * grayImage,
                                            unsigned char * rgbImage,
                               int width, int height) {
 int x = threadIdx.x + blockIdx.x * blockDim.x;
 int y = threadIdx.y + blockIdx.y * blockDim.y;

 if (x < width && y < height) {
    // get 1D coordinate for the grayscale image
    int grayOffset = y*width + x;
    // one can think of the RGB image having
    // CHANNEL times columns than the gray scale image
    int rgbOffset = grayOffset*CHANNELS;
    unsigned char r =  rgbImage[rgbOffset      ]; // red value for pixel
    unsigned char g = rgbImage[rgbOffset + 2]; // green value for pixel
    unsigned char b = rgbImage[rgbOffset + 3]; // blue value for pixel
    // perform the rescaling and store it
    // We multiply by floating point constants
    grayImage[grayOffset] = 0.21f*r + 0.71f*g + 0.07f*b;
 }
}
```

More on GPU programming in Lab 14!

Cellular Automaton for track seeding



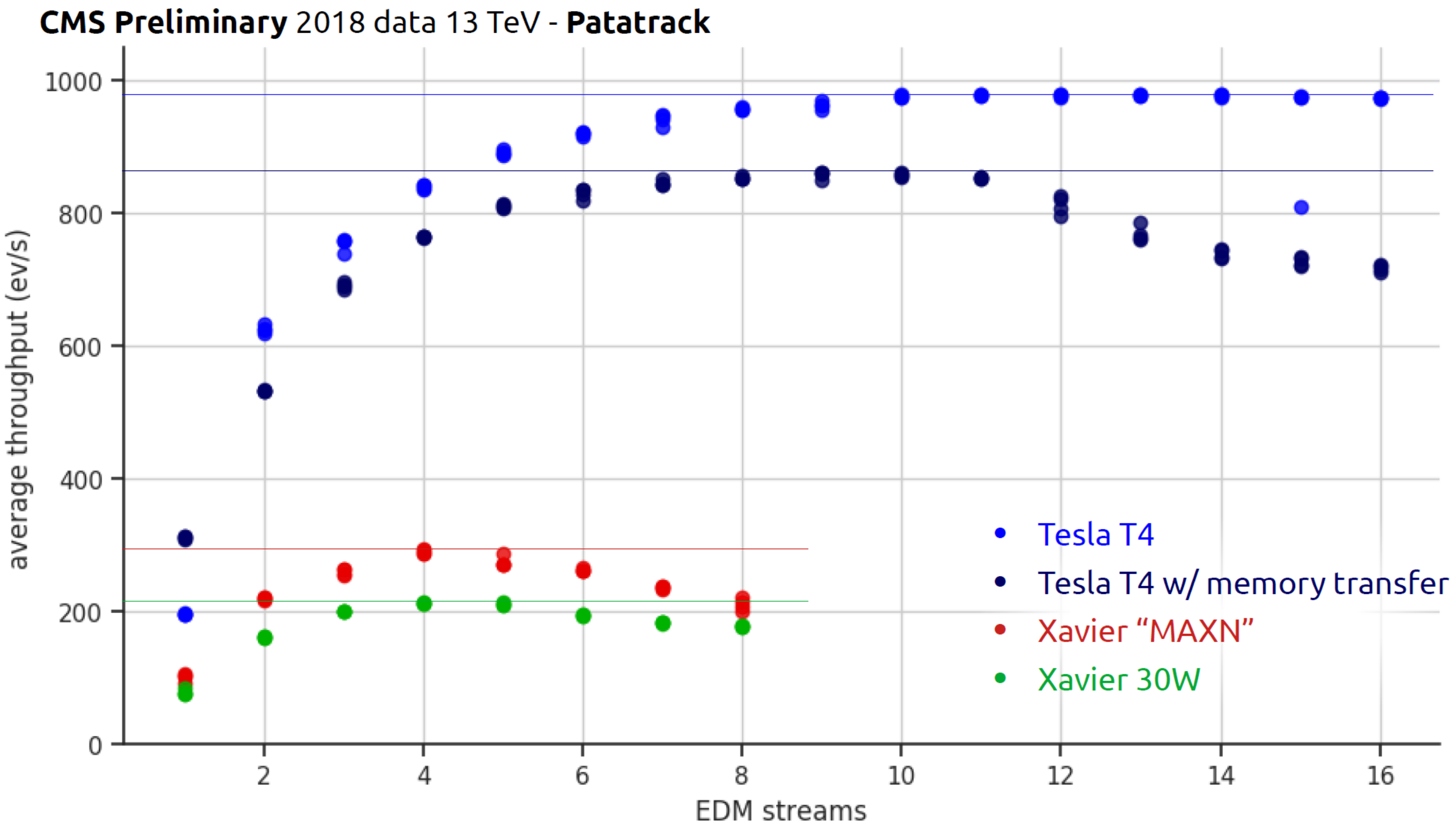
- Build local tracks segments from detector layers
  - Highly parallelizable
- Connect the possible segments
- Apply some rule to find the real track among all the possible tracks
- Design from scratch for parallel application

#	Phase	Task	Method	Locality	Time	Device
1	I	Seeding	Cellular Automaton	Very local	30 %	CPU & GPU
2		Track following	Simple Kalman filter	Sector-local	60 %	CPU & GPU
3	II	Track Merging	Matching Covariance	Global	2 %	CPU
4		Final Fit	Kalman filter	Global	8 %	CPU (or GPU)

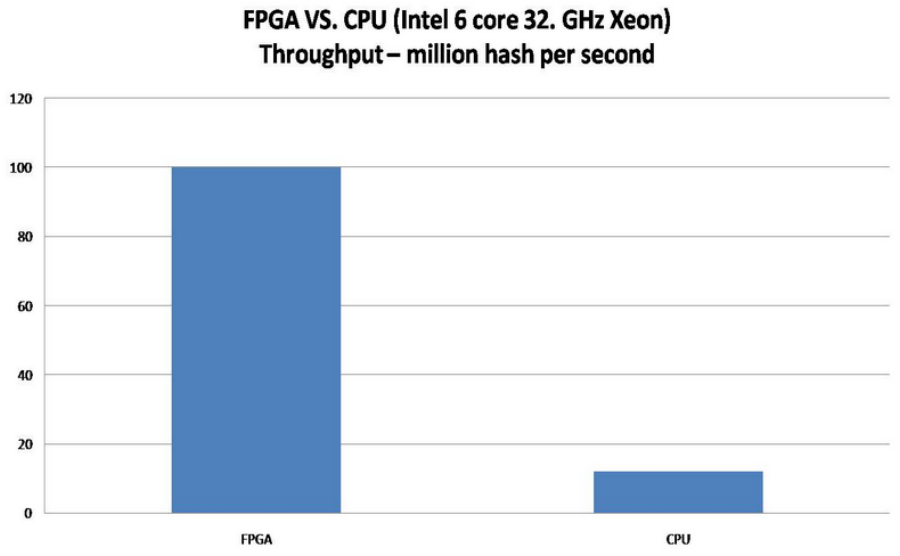
CMS: Heterogenous Computing

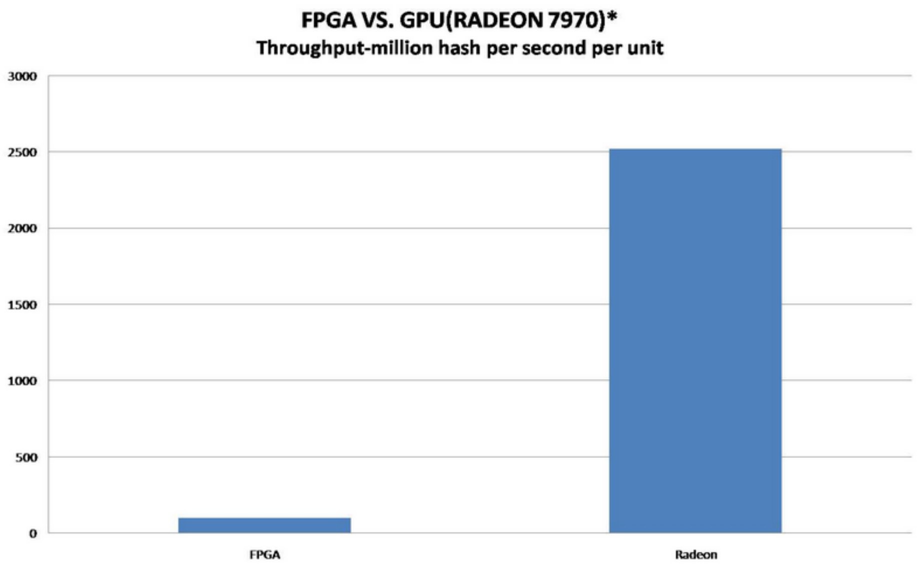
- NVIDIA Jetson AGX Xavier
  - Single board computer with 8 ARMv8 cores with an integrated Volta GPU (512 cores)
  - Rediced power consumption: 30 W

- Encouraging results
  - Comparison with T4: 2560 cores
- Preliminar results on Cavium ThunderX2+Volta (5120 cores) give about 1800 ev/s
  - 150 W



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

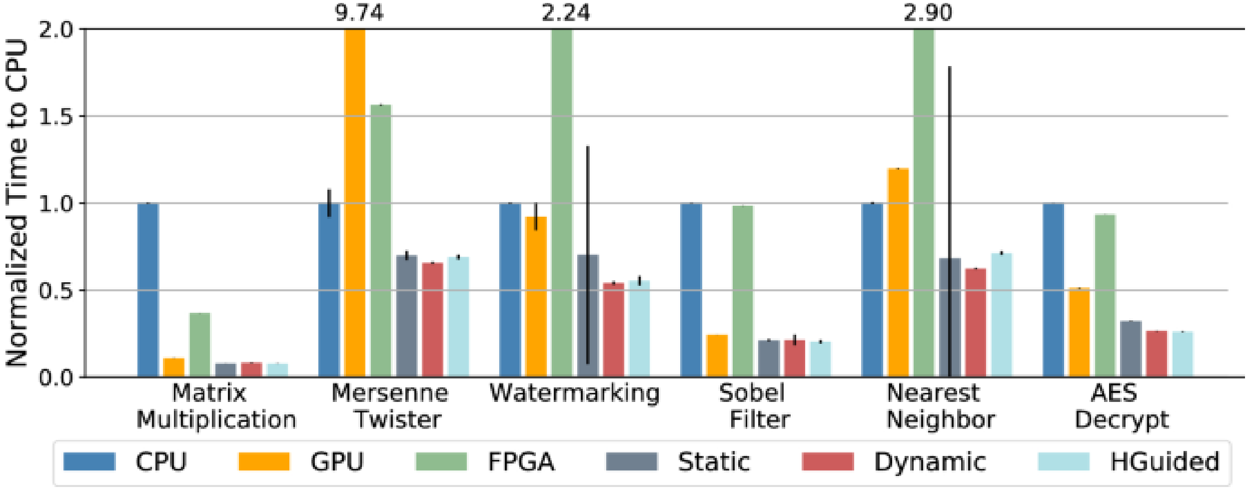
GPU vs CPU vs FPGA

FPGA	Multicore Processor	GPU
Parallel	Sequential	Limited parallel
Extremely fast real time processing	Varies with dependency	Fast real time processing
Not good in floating point operations	Versatile	Excellent in floating point operations
HDL programming (complicated)	Easy to program(C++)	OpenCV libraries (easy to program)
Need to interface the software	No need of special interfacing	No need of special interfacing
Comparatively less flexible but better performance	-	Programming flexibility

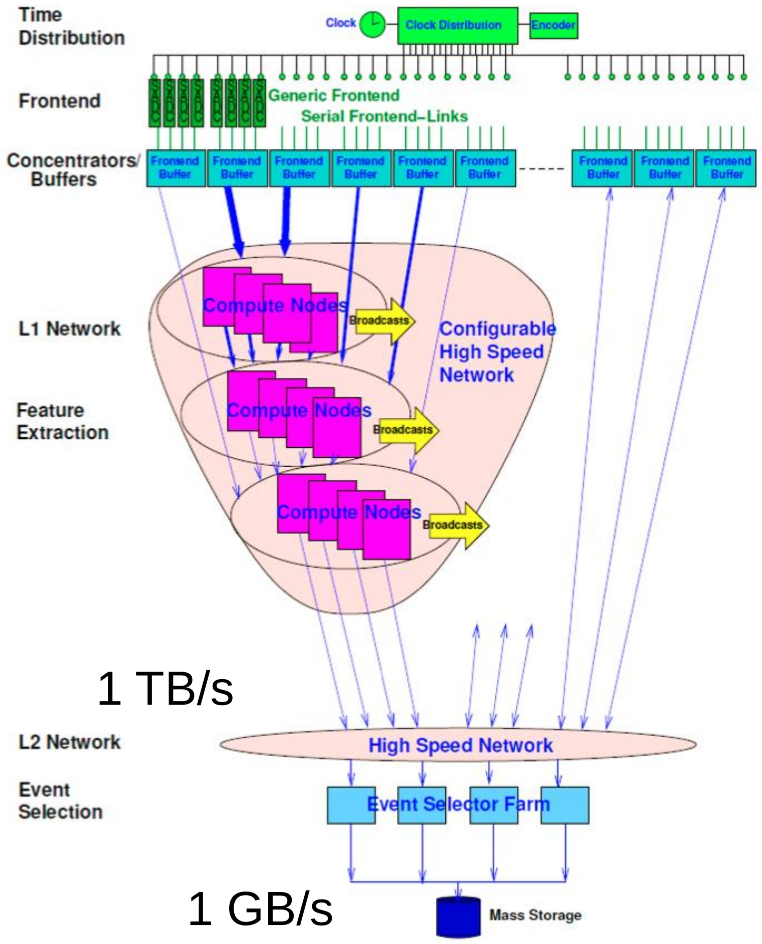
Table 1: Comparison of FPGA, GPU and CPUs

IV. CONCLUSION

Platform	Number of cores	Serial or parallel	Clock frequency	Development time	Power efficiency	Portability
CPU	Low	Mainly serial	High	Short	Average	Straightforward
GPU	High	Parallel	High	Average	Low	Less challenging
FPGA	High	Parallel	Low	Long	High (less power consumption, depending on implementation)	Difficult when vendor-specific IP-cores are used
DSP	Low	Mainly serial	Average to high	Average	Average	Simple (from low- to higher-performance)
Xeon Phi/ ClearSpeed	Average	Serial with many-core parallel	Average	Short	Average	Average
Heterogeneous SoC, e.g., FPGA + CPU	High	Serial and parallel	Low	Long	High	Platform dependent



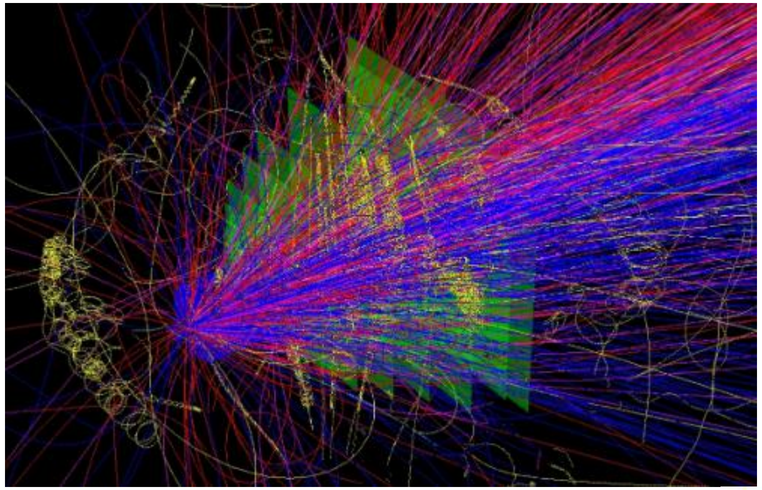
PANDA



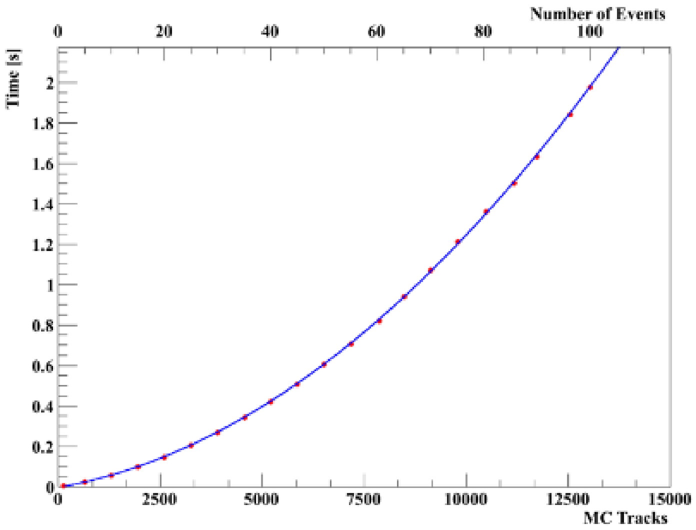
- 10^7 events/s
- Full reconstruction for online selection: assuming 1-10 ms → 10000 – 100000 CPU cores
- Tracking, EMC, PID,…
- First exercice: online tracking
- Comparison between the same code on FPGA and on GPU: the GPUs are 30% faster for this application (a factor 200 with respect to CPU)

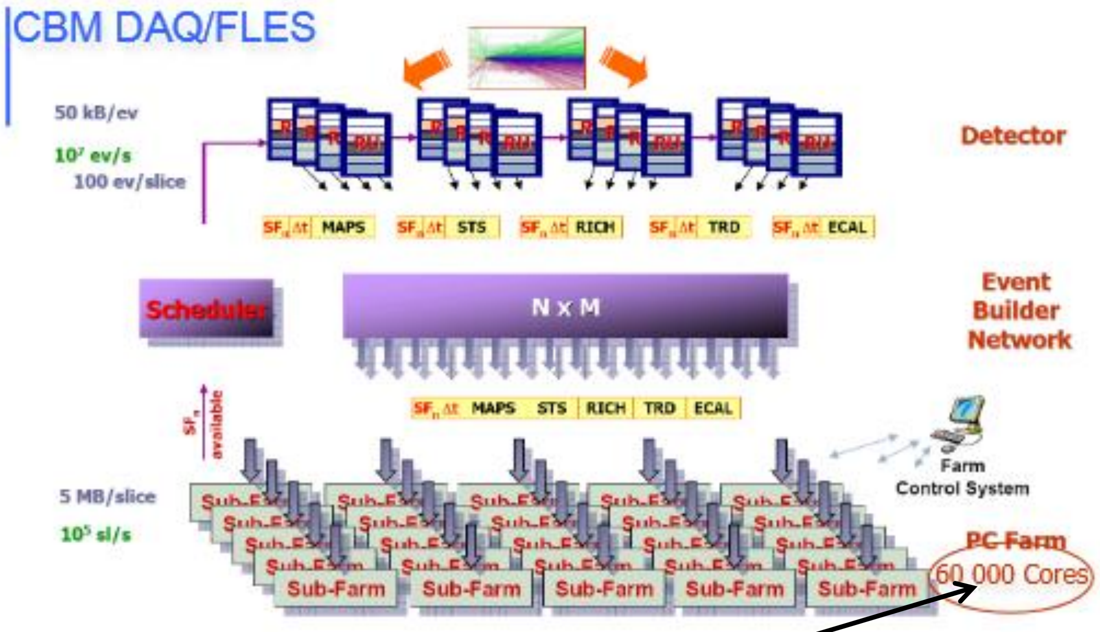
	CPU (ms)	GPU (ms)	Improvement	Occupancy	Notes
total runtime (without Z-Analysis)	117138	590	199		
startUp()	0.25	0.0122	20	2%	runs (num_points) times
setOrigin()	0.25	0.0119	21	25%	runs (num_points) times
clear Hough and Peaks (memset on GPU)	3	0.0463	65	100%	runs (num_points) times
conformalAndHough()	73	0.8363	87	25%	runs (num_points) times
findPeaksInHoughSpace()	51	0.497	103	100%	runs (num_points) times
findDoublePointPeaksInHoughSpace()	4	0.0645	62	100%	runs (num_points) times
collectPeaks()	4	0.066	61	100%	runs (num_points) times
sortPeaks()	0.25	0.0368	7	2%	runs (num_points) times
resetOrigin()	0.25	0.0121	21	25%	runs (num_points) times
countPointsCloseToTrackAndTrackParams()	22444	0.9581	23426	33%	runs once
collectSimilarTracks()	4	2.3506	2	67%	runs once
collectSimilarTracks2()				2%	runs once
getPointsOnTrack()	0.25	0.0187	13	33%	runs (num_tracks) times
nullifyPointsOfThisTrack()	0.25	0.0106	24	33%	runs (num_tracks) times
clear Hough space (memset on GPU)	2	0.0024	833	100%	runs (num_tracks) times
secondHough()	0.25	0.0734	3	4%	runs (num_tracks) times
findPeaksInHoughSpaceAgain()	290	0.2373	1222	66%	runs (num_tracks) times
collectTracks()	0.25	0.0368	7	2%	runs (num_tracks) times

CBM & STAR



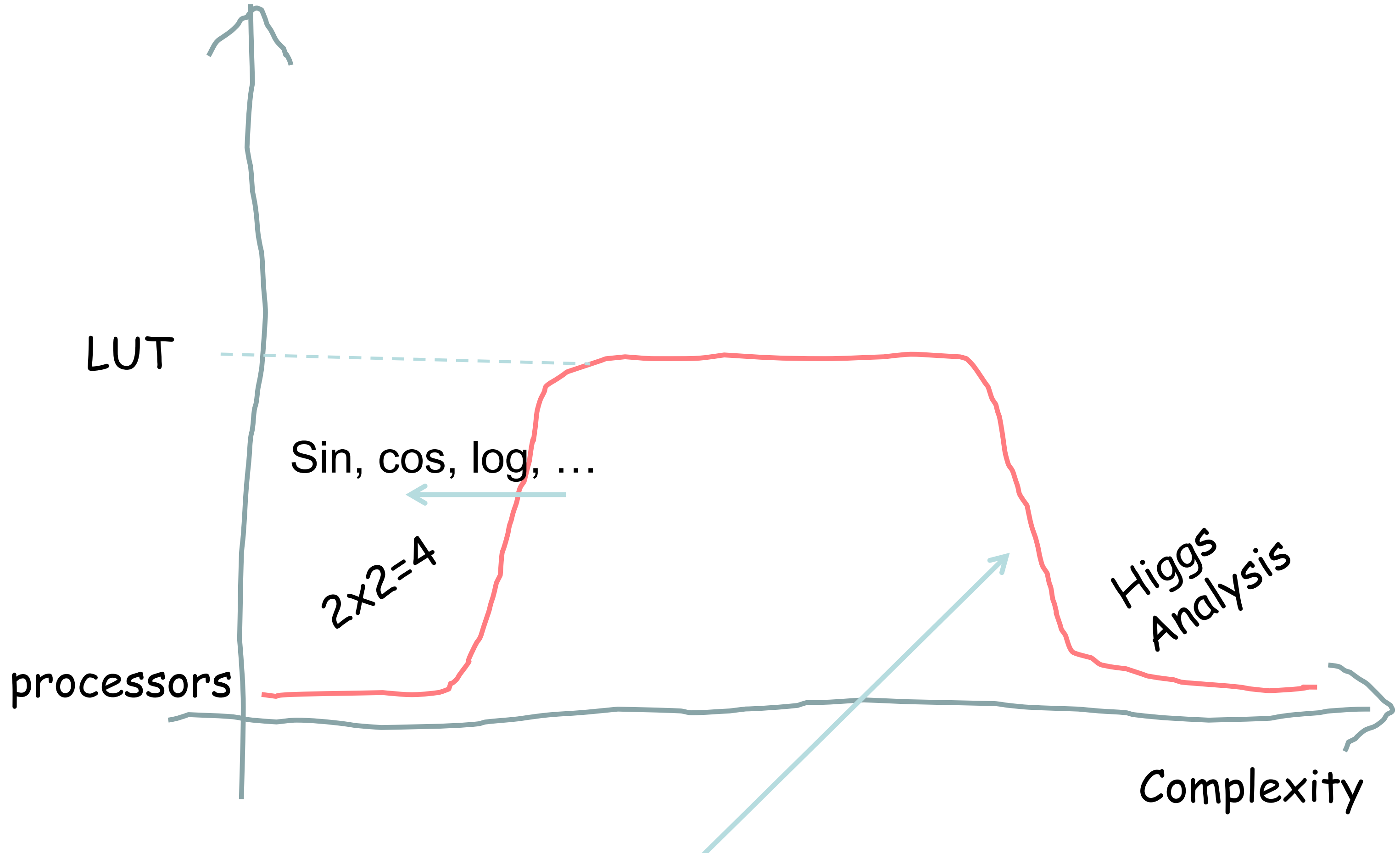
- 10^7 Au+Au collisions /s
- ~1000 tracks/event
- trigger-less
- Since the continuos structure of the beam ~10000 tracks/frame
- Cellular automaton+KF





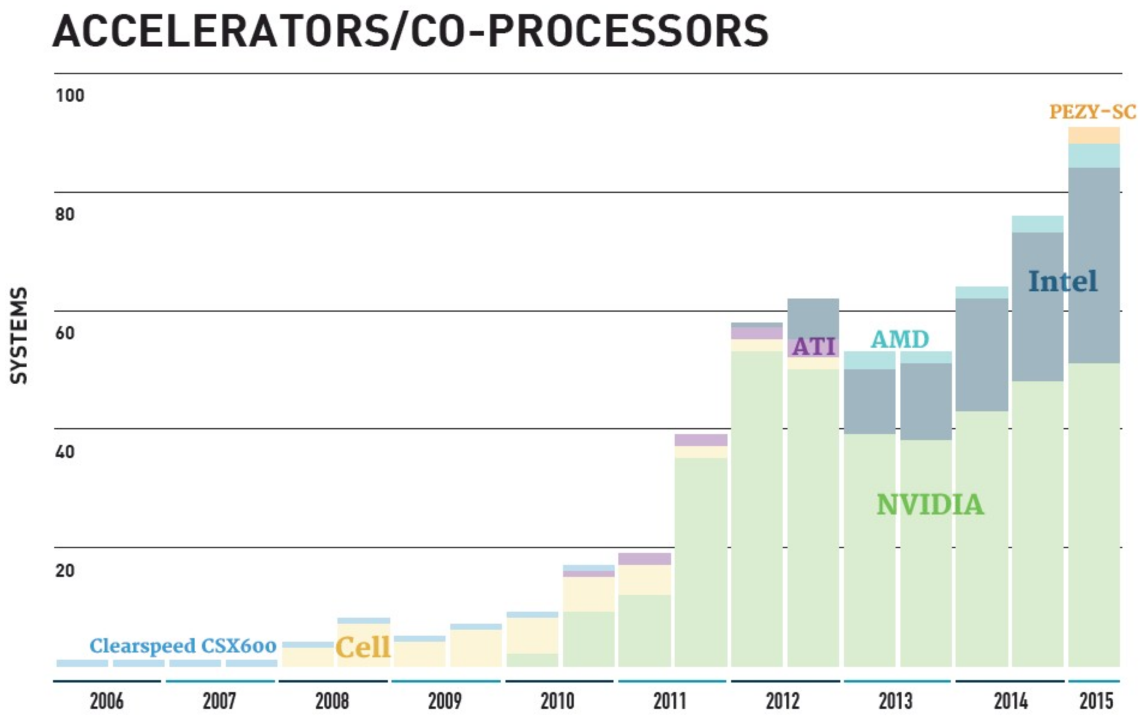
Grid=100000 cores

Computing vs LUT



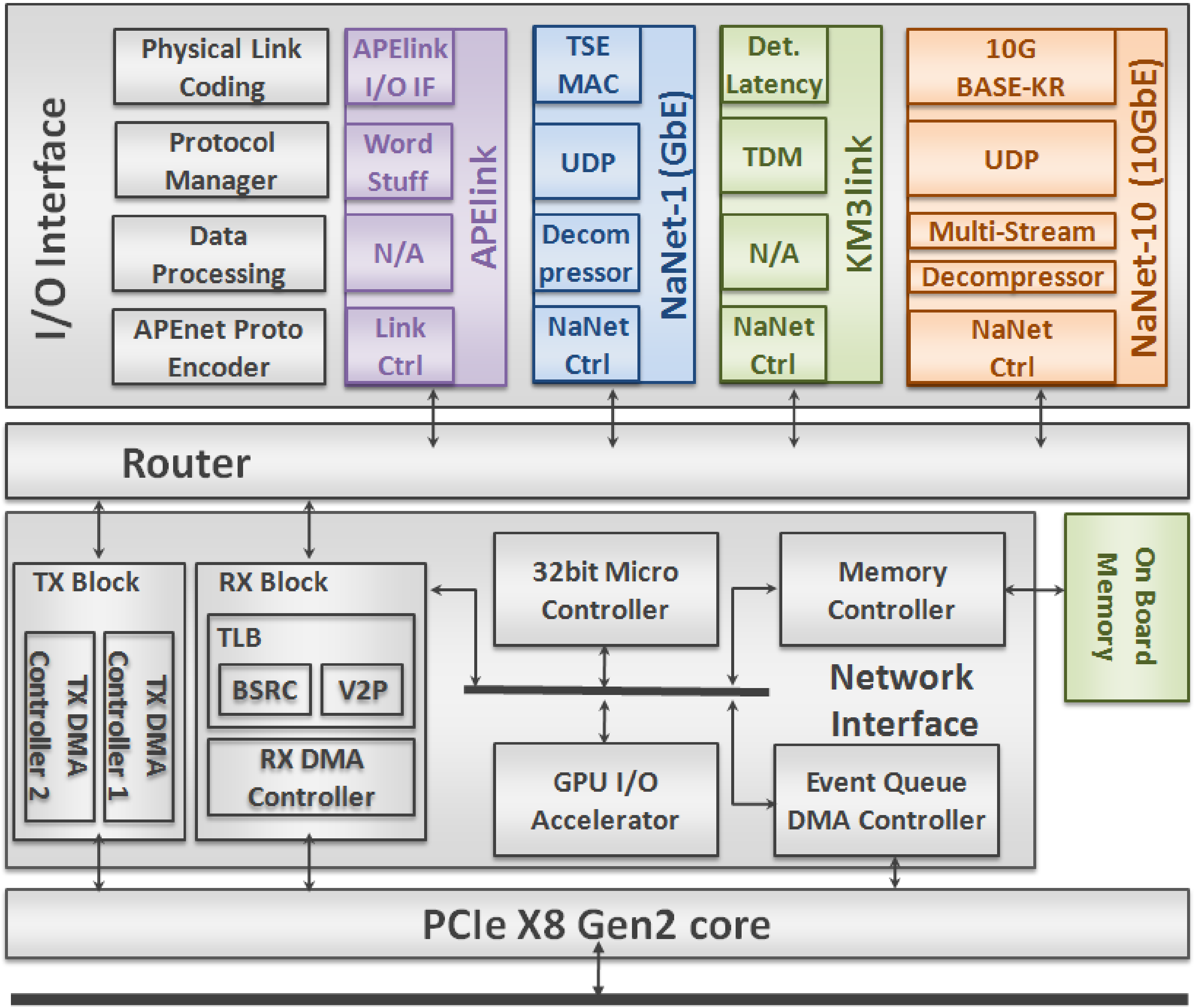
Where is this limit?
It depends …
In any case the GPUs
aim to shrink this space

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



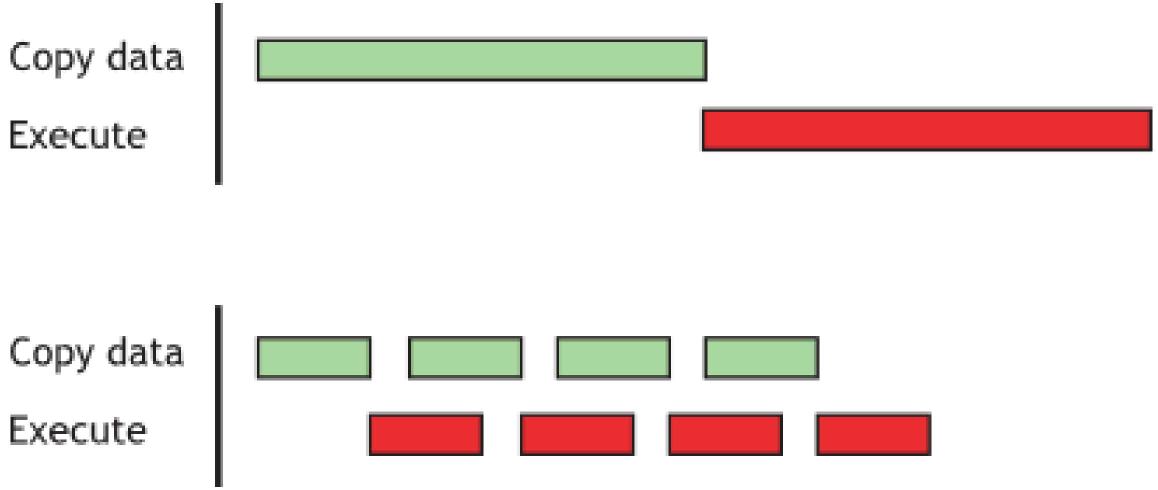
CPU v GPU

	Intel Core E7-8890 v3	GeForce GTX 1080
Core count	18 cores / 36 threads	20 SMs / 2560 cores
Frequency	2.5 GHz	1.6 GHz
Peak Compute Performance	1.8 TFLOPs	8873 GFLOPs
Memory bandwidth	Max. 102 GB/s	320 GB/s
Memory capacity	Max. 1.54 TB	8 GB
Technology	22 nm	16 nm
Die size	662 mm^2	314 mm^2
Transistor count	5.6 billion	7.2 billion
Model	Minimize latency	Hide latency through parallelism


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
Decom pressor
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

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
```