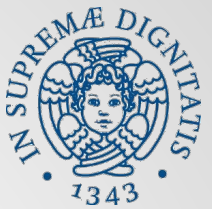


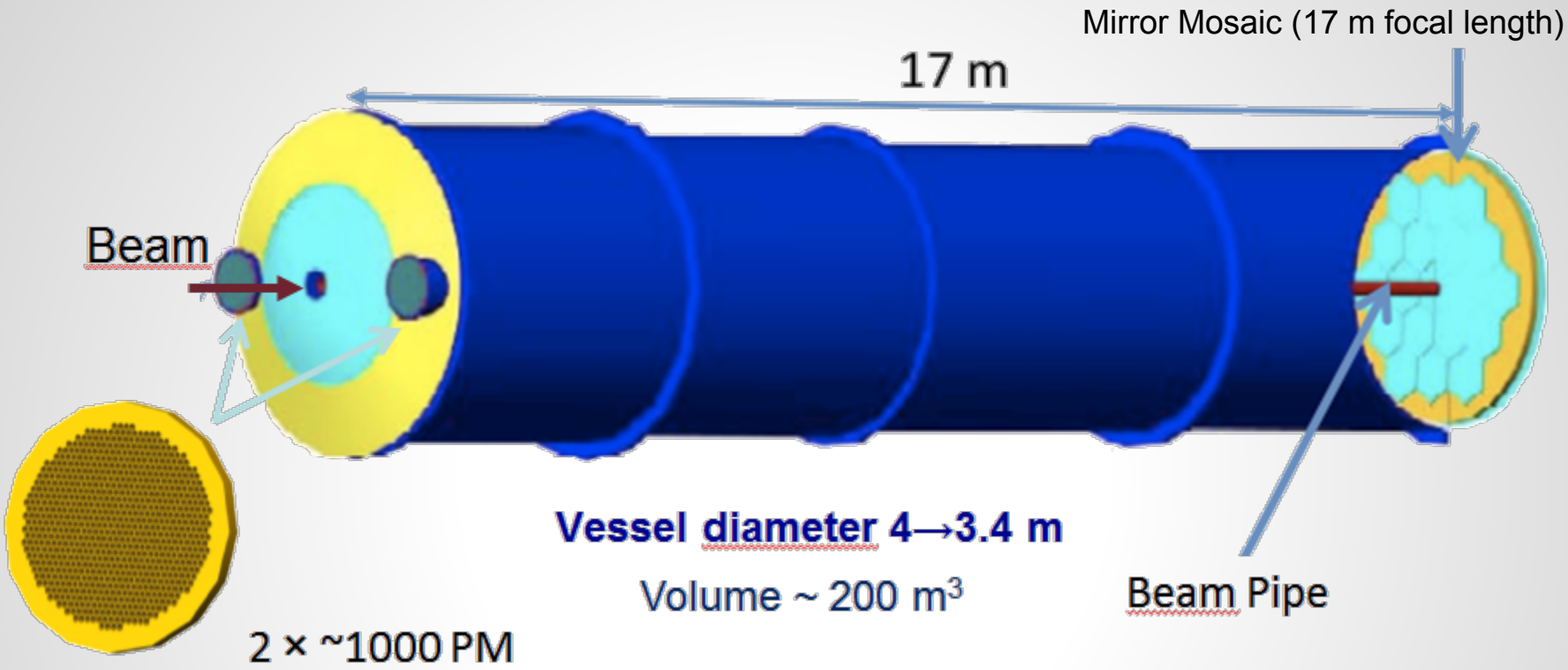
Real-time use of GPUs in NA62 Experiment

Vincenzo Innocente, Felice Pantaleo (PH-SFT)

Meeting on Concurrent Programming Models and Frameworks
10 October 2012

felice.pantaleo@cern.ch

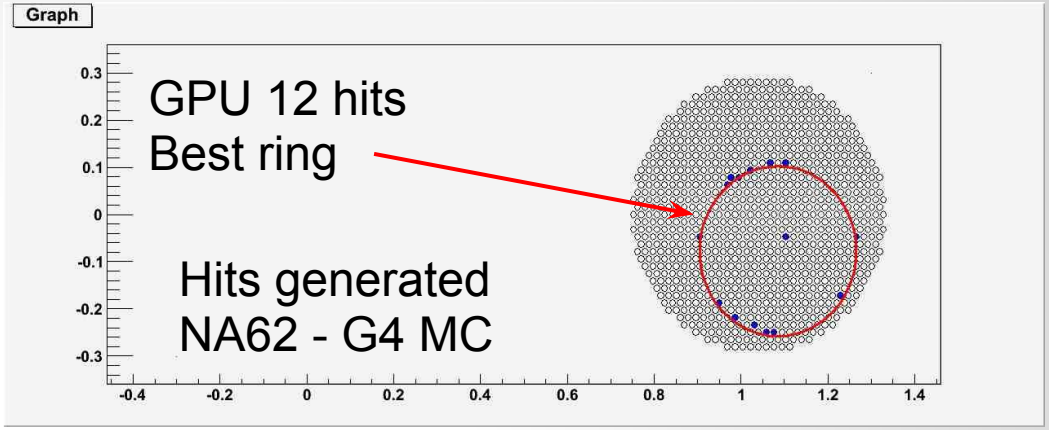
First application: RICH

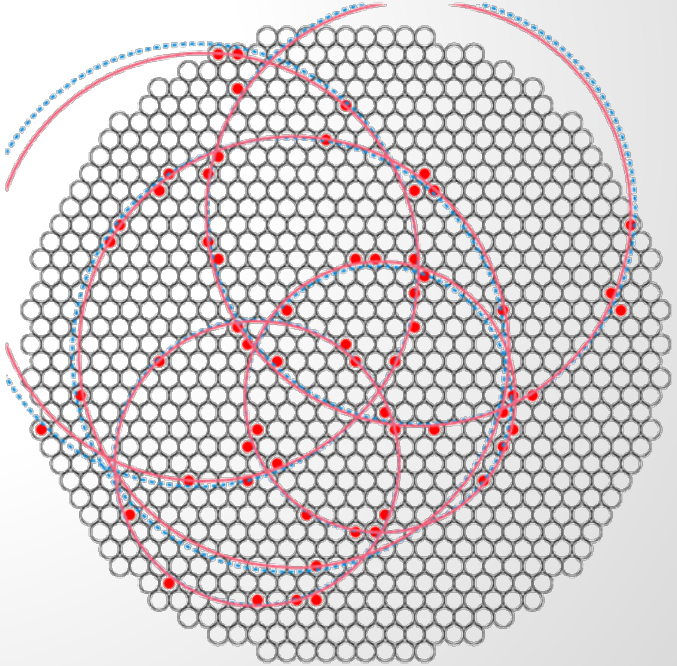


- ~17 m RICH
- 1 atm Neon
- Light focused by two mirrors on two spots equipped with ~1000 PMs each (pixel 18 mm)
- 3s p-m separation in 15-35 GeV/c, ~18 hits per ring in average
- ~100 ps time resolution, ~10 MHz events rate
- Time reference for trigger

GPU trigger for RICH detector

- Natively built for pattern recognition problems
- First attempt: ring reconstruction in RICH detector.

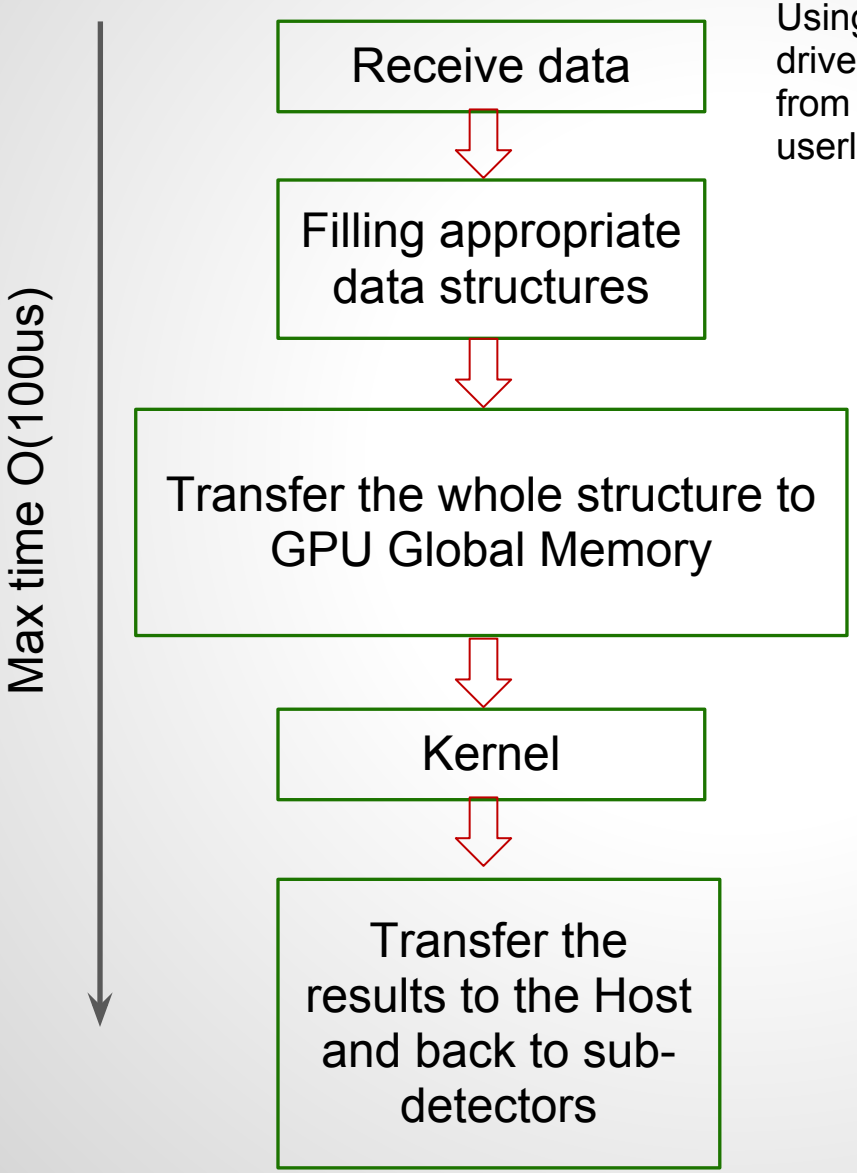




It's a **pilot project**, very promising R&D!

felice.pantaleo@cern.ch

GPU as a L0 Trigger



Using a "zero copy" intel modded driver, we are able to copy data from the NIC directly to the userland

Crawford Algorithm description

Consider a circle of radius R, centered in (x_o , y_o) and a list of points (x_i , y_i).

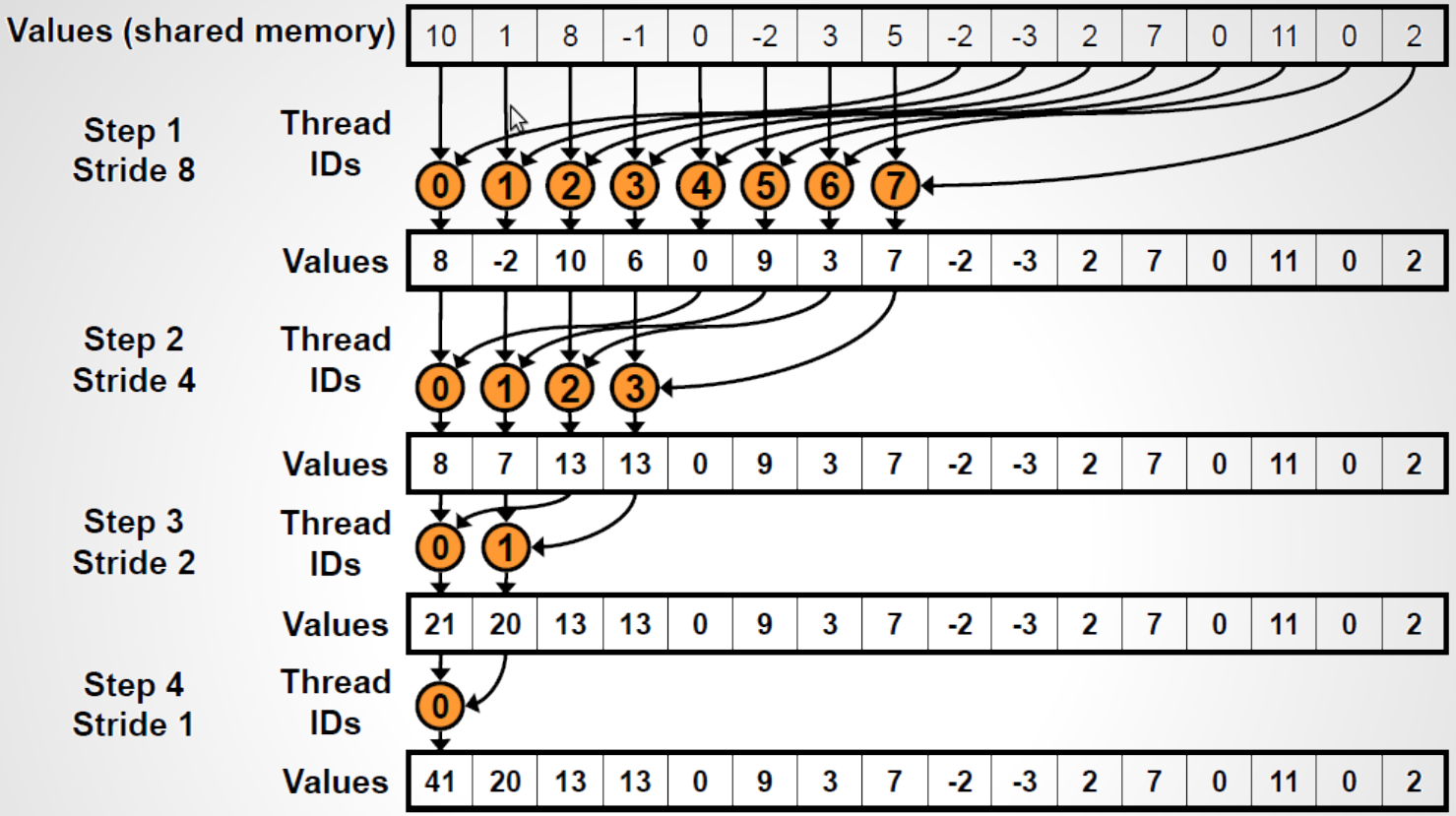
The following relations exist:

$$x_0{}^2 + y_0{}^2 - R^2 = \frac{1}{N}\{2x_0 \sum x_i + 2y_0 \sum y_i - \\ - \sum x_i^2 - \sum y_i^2\}. \qquad (1)$$

$$x_0\{\sum x_i^2 - \frac{(\sum x_i)^2}{N}\} + y_0\{\sum x_i y_i - \\ - \frac{\sum x_i \sum y_i}{N}\} = \frac{1}{2}\{\sum x_i^3 + \sum x_i y_i^2 \\ - \sum x_i \frac{\sum x_i^2 + \sum y_i^2}{N}\}, \qquad (2)$$

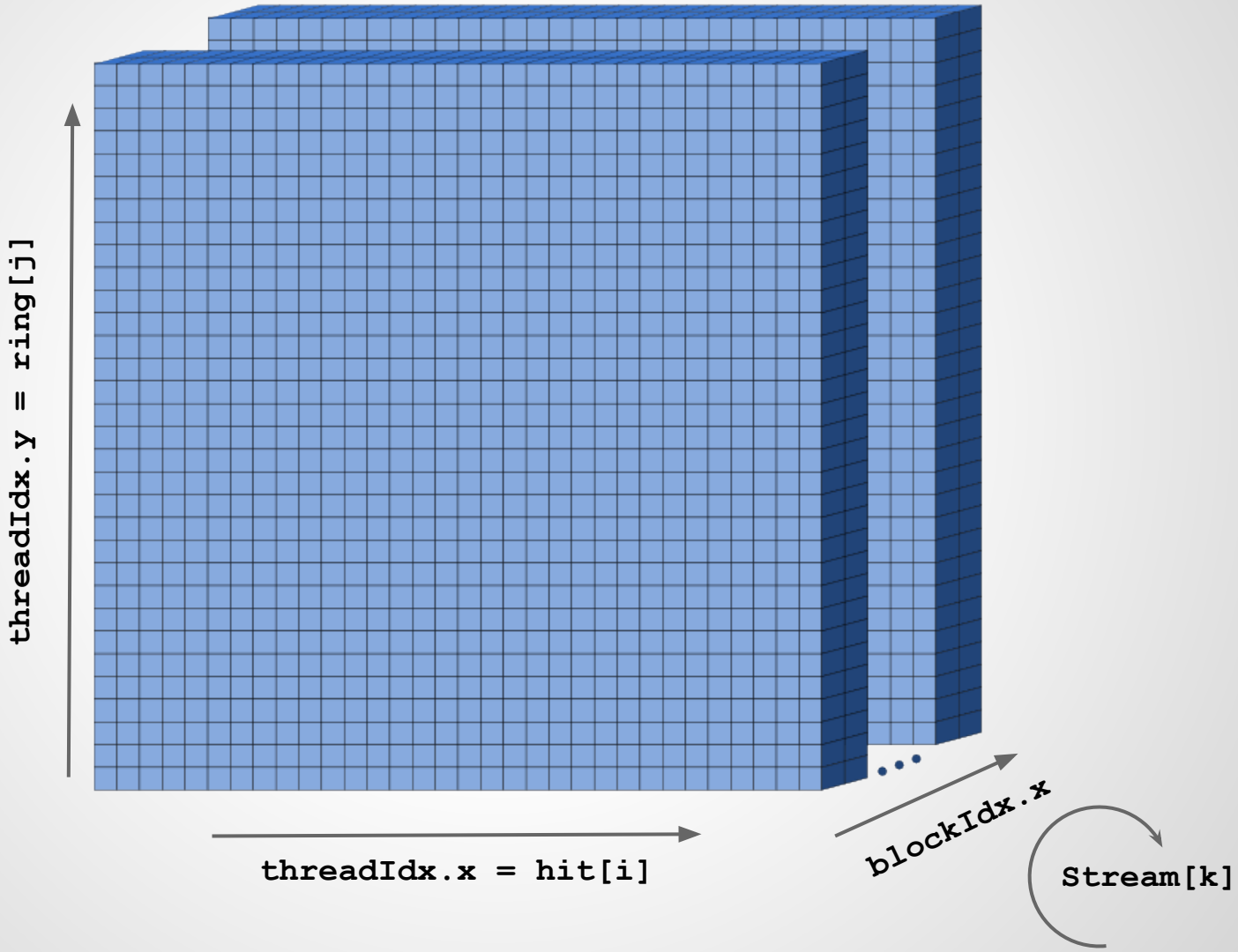
$$x_0\{\sum x_i y_i^2 - \frac{\sum x_i \sum y_i}{N}\} + y_0\{\sum y_i^2 - \\ - \frac{\sum y_i^2}{N}\} = \frac{1}{2}\{\sum x_i^2 y_i + \sum y_i^3 - \\ - \sum y_i \frac{\sum x_i^2 + \sum y_i^2}{N}\}. \qquad (3)$$

Reduction



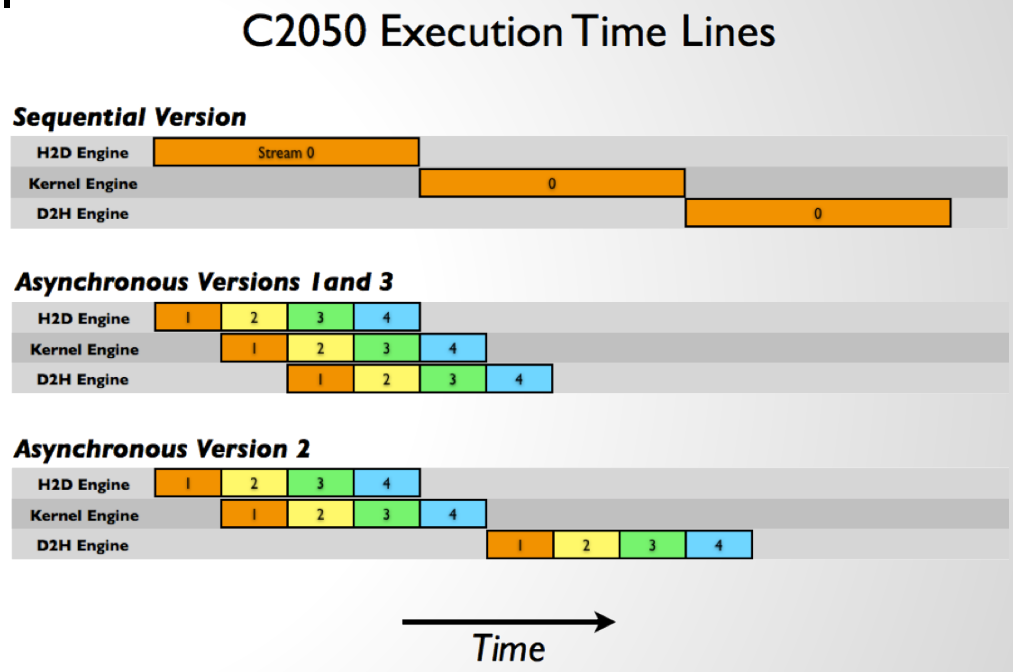
- One reduction kernel is called per block, giving an array of results (one for each event)
- Must use sequential addressing instead of interleaved addressing to avoid Shared Memory bank conflicts
- Time complexity is O(logN), cost is O(N*logN): not cost efficient
- Brent's theorem (algorithm cascading) suggests O(N/logN) threads:
  - Each thread does O(logN) sequential work
  - All O(N/logN) threads cooperate for O(logN) steps
  - New cost = O(N/logN * logN) = O(N)

Algorithm description - ctd



Stream Scheduler

- Exploit the instruction-level parallelism (i.e. pipelining streams)
- This is usually done by interlacing one stream instructions with another stream ones
- This cannot be done in real-time without the introduction of other **unknown** latencies
- CPU hw-thread-level parallelism is the solution



Hardware on the workbench

First Machine

- GPU: NVIDIA Tesla C2050
  - 448 CUDA cores @ 1.15GHz
  - 3GB GDDR5 ECC @ 1.5GHz
  - CUDA CC 2.0 (Fermi Architecture)
  - PCIe 2.0 (effective bandwidth up to ~5GB/s)
  - CUDA Runtime v4.2, driver v295.20 (Feb '12)
- CPU: Intel® Xeon® Processor E5630 (released in Q1'10)
  - 2 CPUs, 8 physical cores (16 HW-threads)
- SLC6, GNU C compiler v4.6.2

Hardware on the workbench

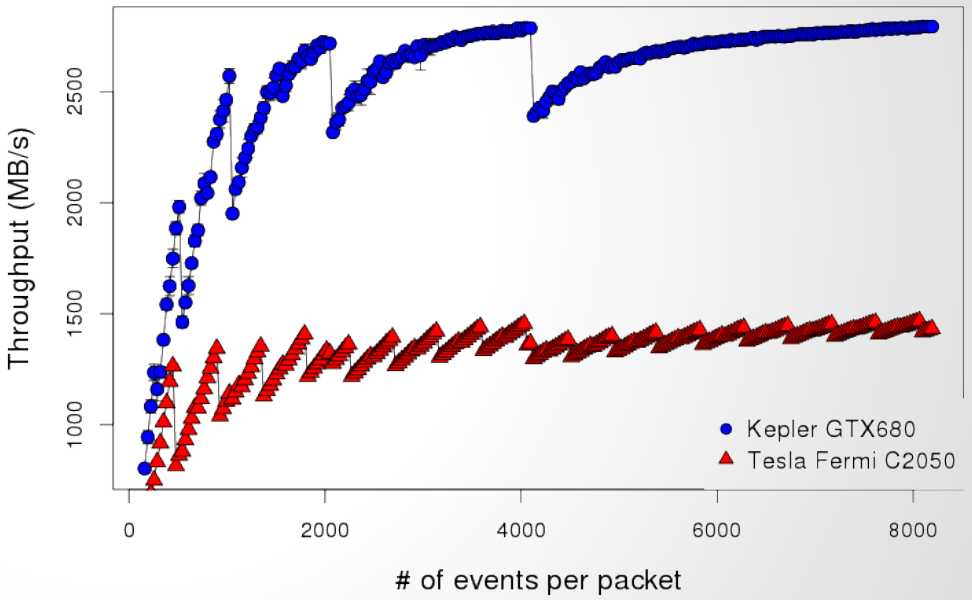
Second Machine

- GPU: NVIDIA GTX680
  - 1536 CUDA cores @ 1.01GHz
  - 2GB GDDR5 ECC @ 1.5GHz
  - CUDA CC 3.0 (Kepler Architecture)
  - PCIe 3.0 (effective bandwidth up to ~11GB/s)
  - CUDA Runtime v4.2, driver v295.20 (Feb '12)
- CPU: Intel® Ivy Bridge Processor i7-3770 (released in Q2 '12)
  - 1 CPUs, 4 physical cores (8 hw-threads) @3.4GHz
- Fedora 17, GNU C compiler v4.6.2

Results - Throughput

The throughput behaviour for a varying number of events inside a packet is a typical many-core device behaviour:

- constant time to process a varying number of events, activating more SMs as the packet size increases
- discrete oscillations due to the discrete nature of the GPU
- saturation plateau (**1.4GB/s** and **2.7 GB/s**)



The right choice of packet dimension is not unique.
It depends on the maximum latency we don't want to exceed and on the input rate of events.

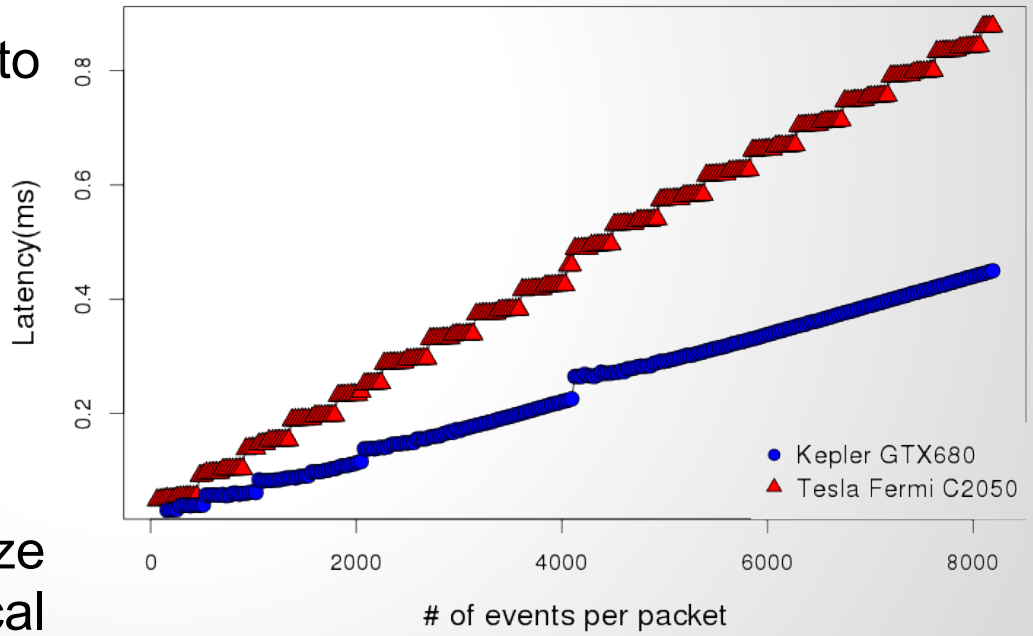
Considering that the maximum rate per sub-detector (@10MHz particles rate) for NA62 experiment is ~500MB/s, I would consider the throughput test **PASSED**

felice.pantaleo@cern.ch

Results - Latency

Latency pretty stable wrt event size.

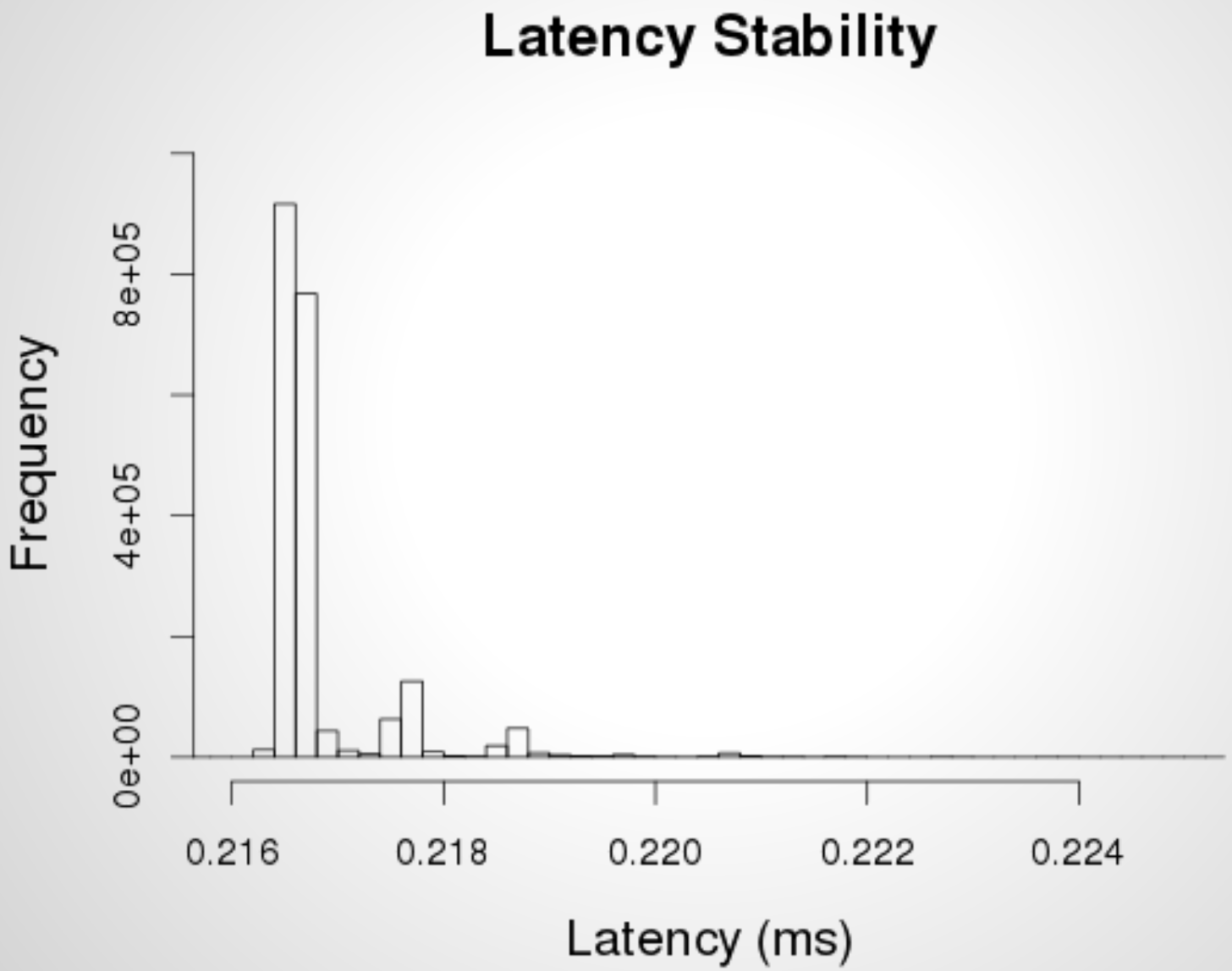
- A lower number of event inside a package is better to achieve a low latency.
- A larger number of event guarantees a better performance and a lower overhead.



The choice of the packet size depends on the technical requirements.

felice.pantaleo@cern.ch

Results - Latency Stability



Technical Run GPU Demonstrator - CHOD

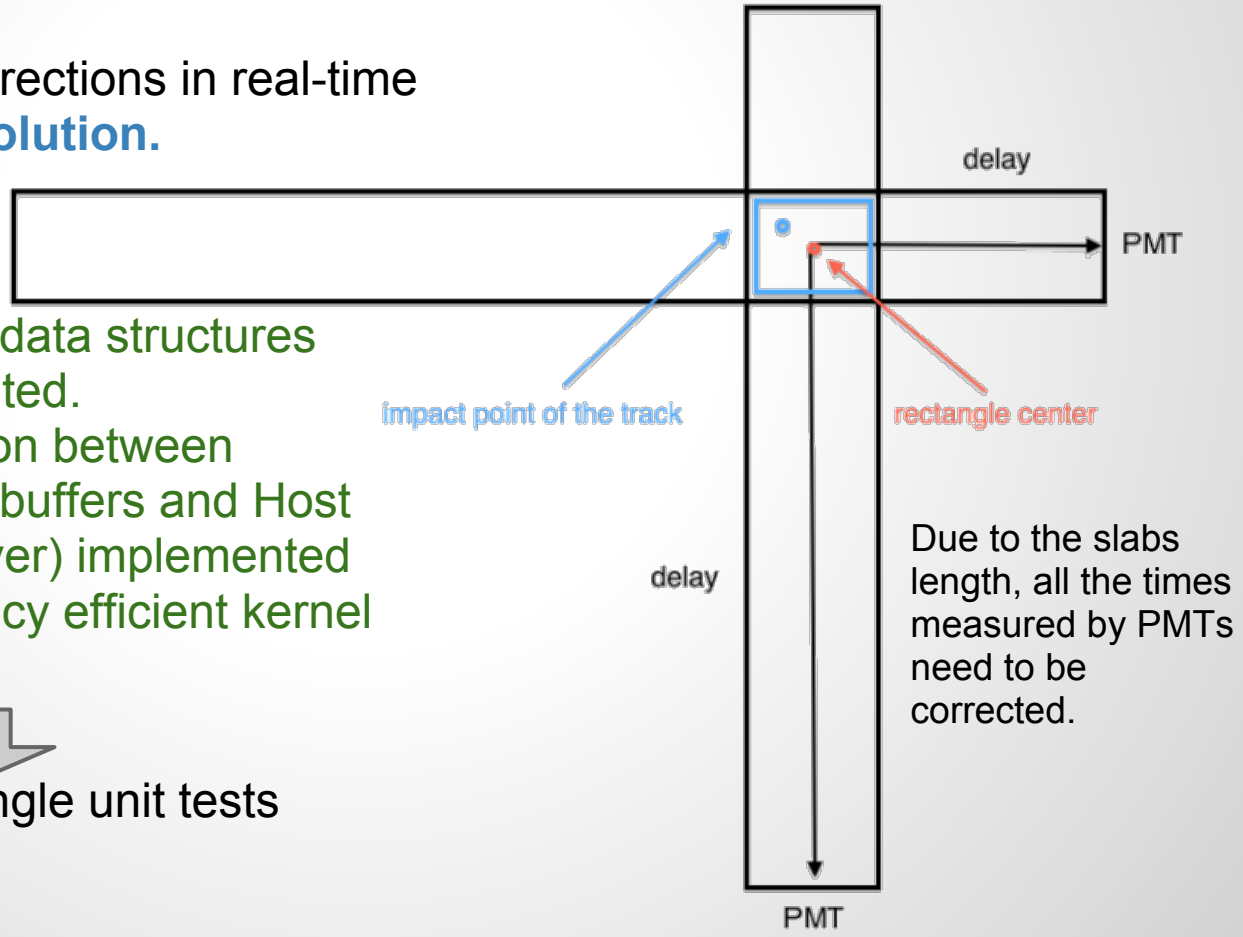
Opportunity to run test on a running sub-detector.

Aim: Applying time corrections in real-time can **improve time resolution.**

Divide et Impera:

- Real-Time-Ready data structures receiver implemented.
- Fast communication between Network Interface buffers and Host Memory (DNA driver) implemented
- A throughput/latency efficient kernel implemented

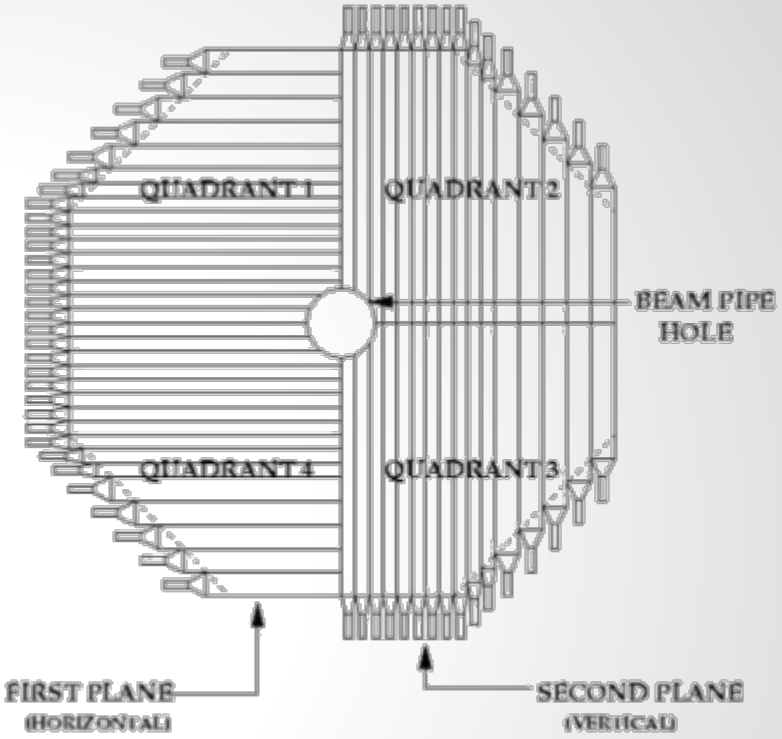
Integrate the single unit tests



Due to the slabs length, all the times measured by PMTs need to be corrected.

CHOD - Kernel

- Assuming the conditions of the detector to be stable, the kernel is an increment by a value taken from a lookup table.
- Each element of the lookup table contains the correction specific of each rectangular zone.
- Each zone correction is the central point one.



Due to the slabs length, all the times measured by PMTs need to be corrected.

CUDA Architecture

Investigation on which memory to use to store this matrix:

Global memory (read and write)

- Slow, but now with cache
- L1 cache designed for spatial re-usage, not temporal (similar to coalescing)
- It benefits if compiler detects that all threads load same value (`LDU` PTX ASM instruction, load uniform)

Texture memory

- Cache optimized for 2D spatial access pattern

Constant memory

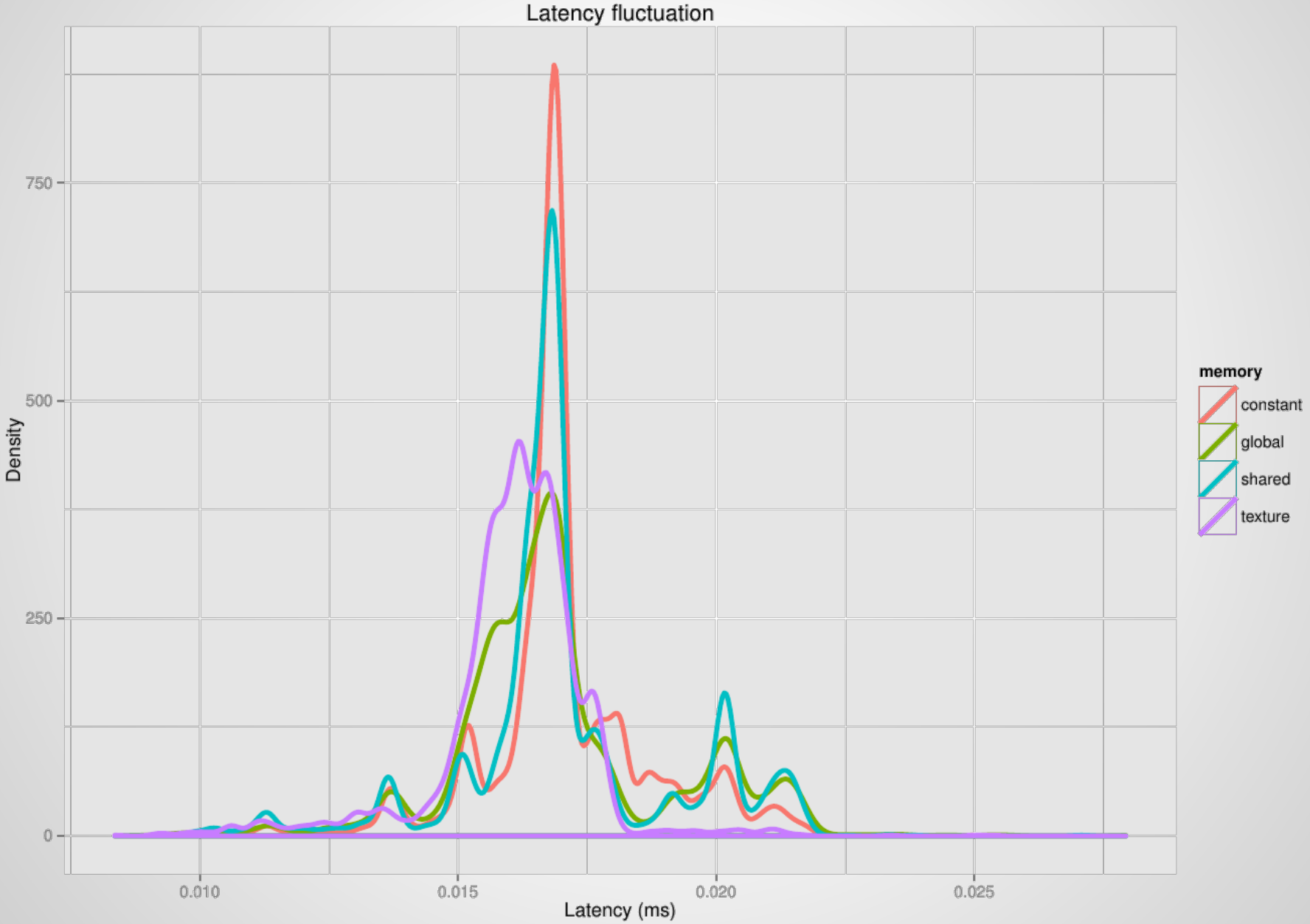
- Slow, but with cache (8 kb)
- Special “LoaD Uniform” (LDU) instruction

Shared memory (48kB per SMX)

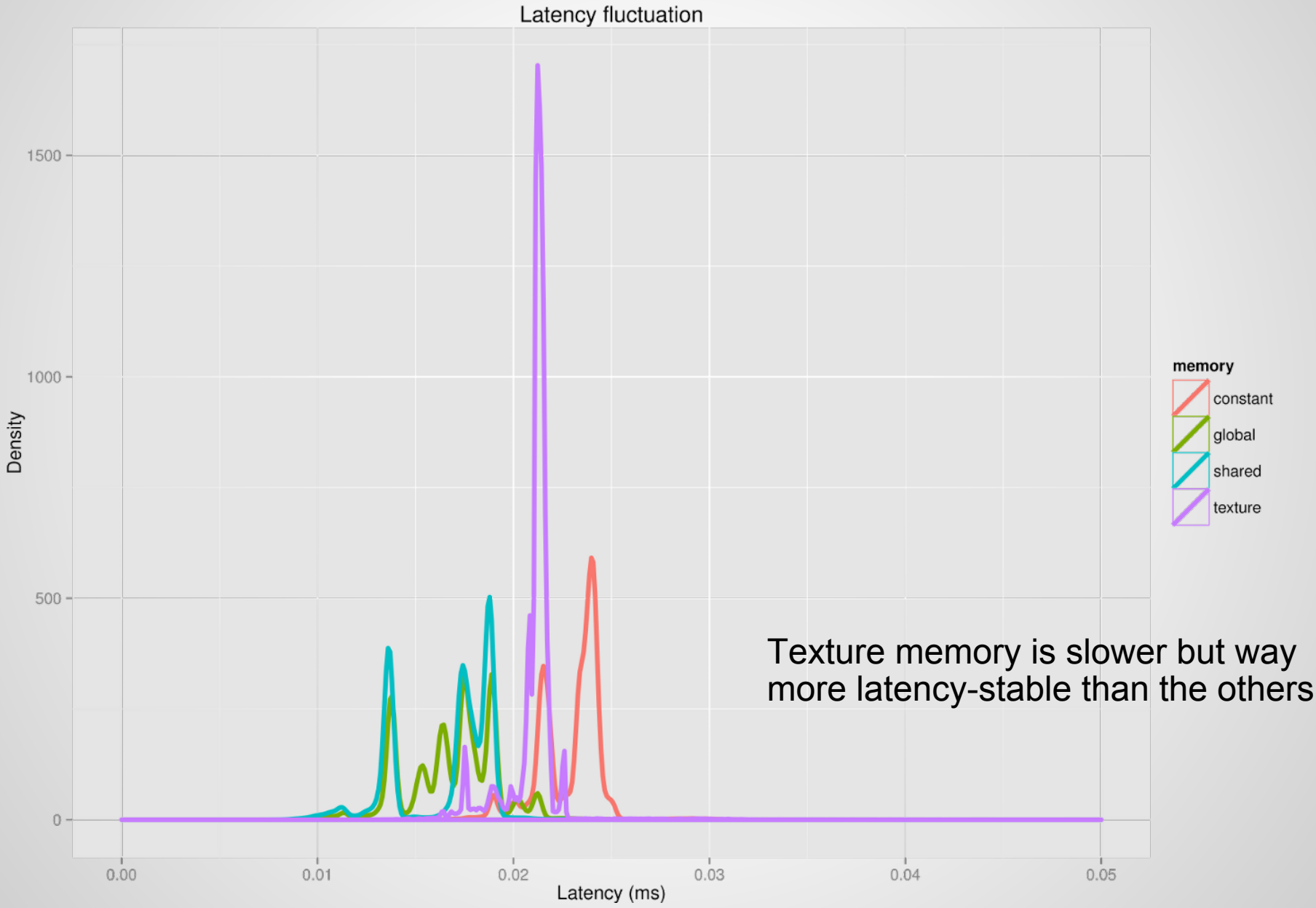
- Fast, but slightly different rules for bank conflicts now

Registers (65536 32-bit registers per SMX)

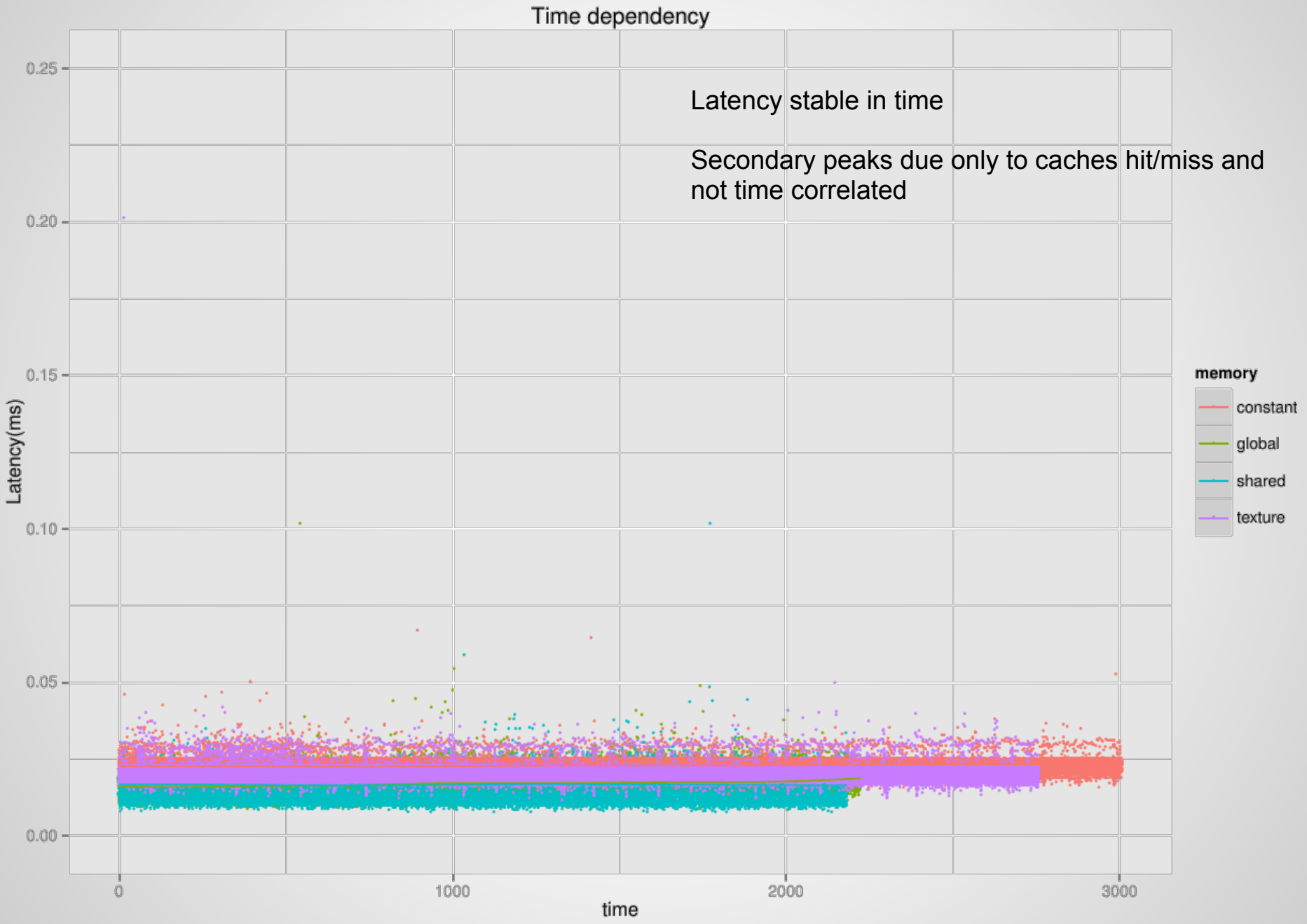
CHOD - Kernel 8x8

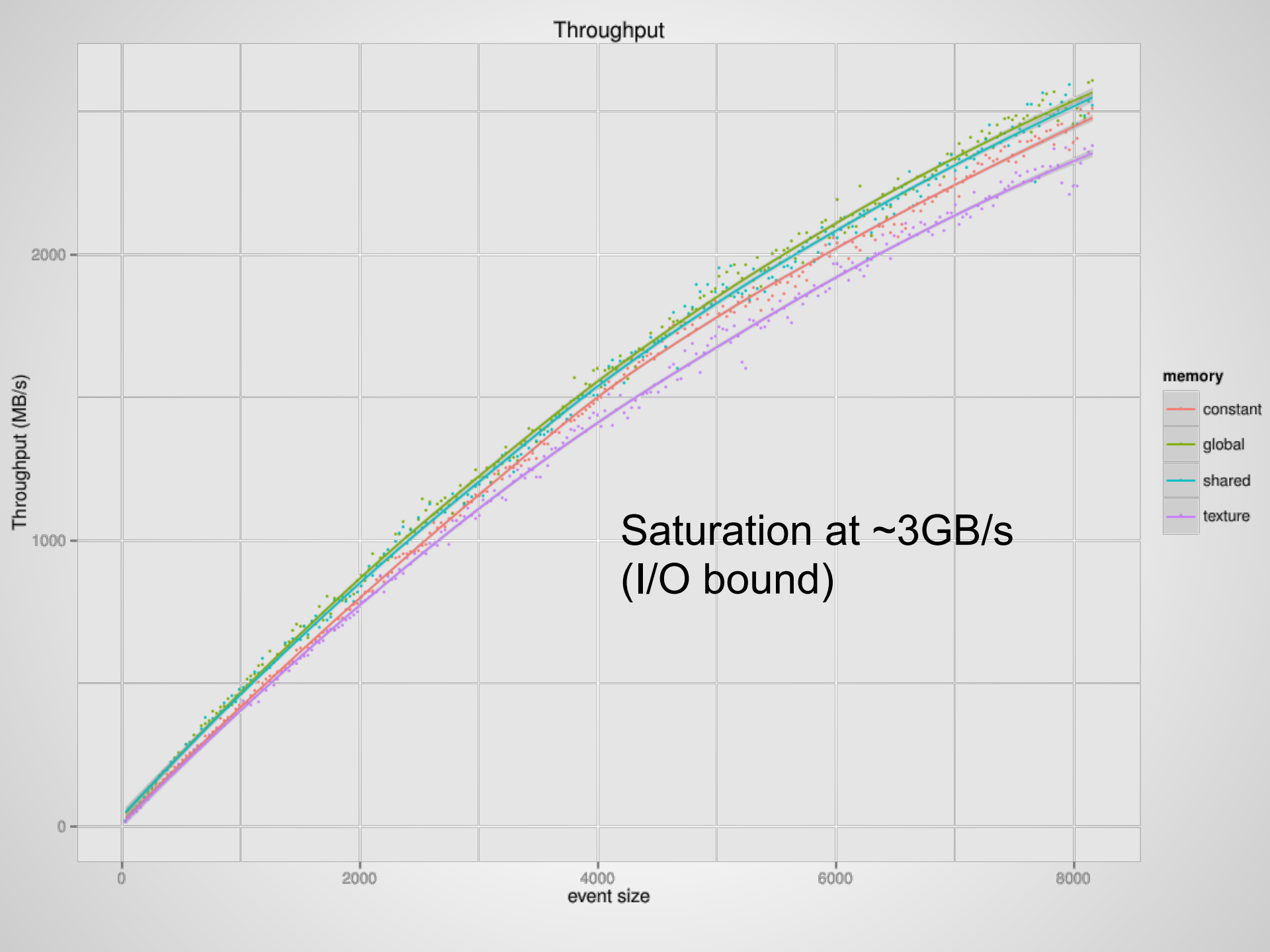


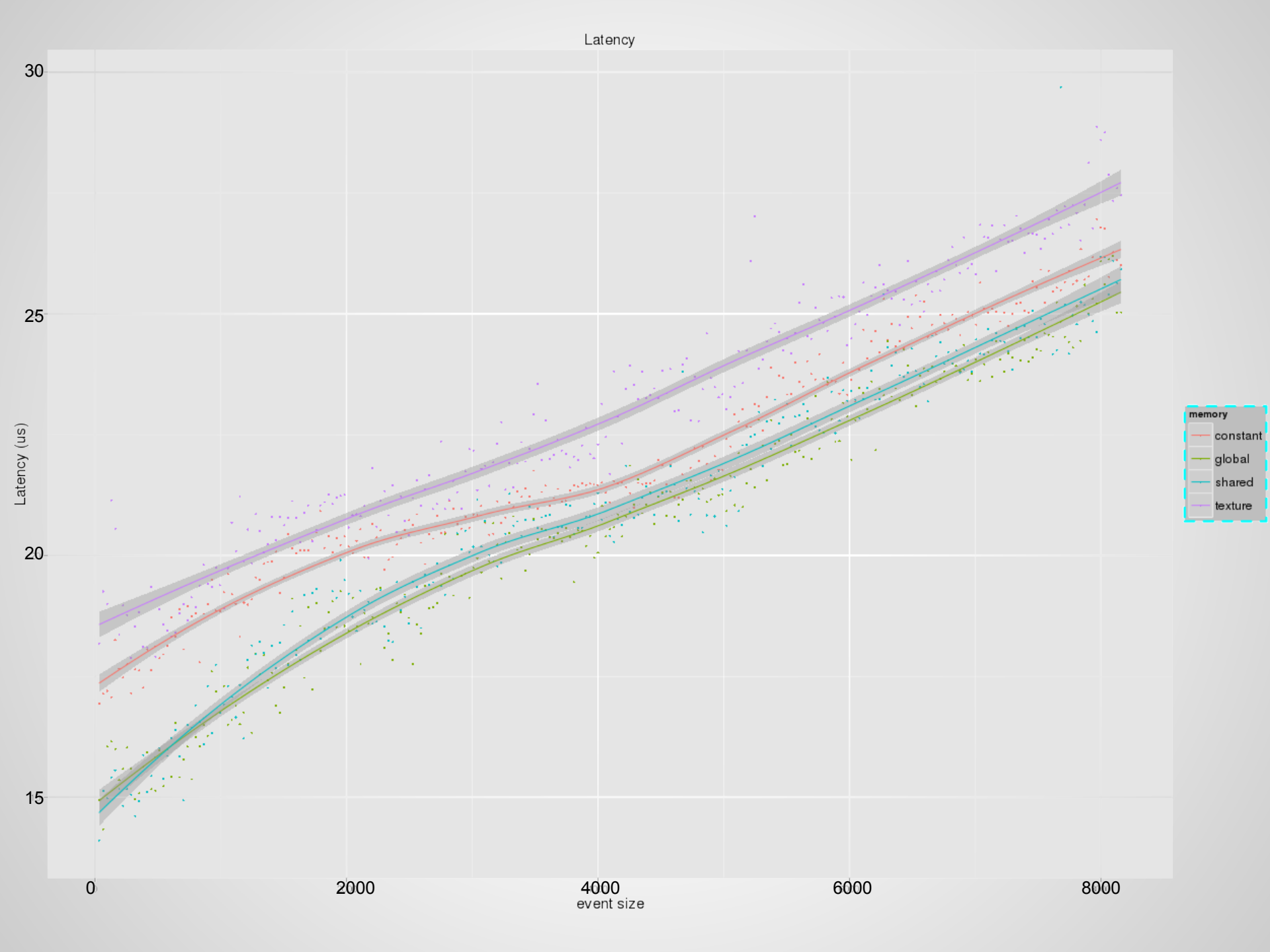
CHOD - Kernel 64x64



Texture memory is slower but way more latency-stable than the others


Time dependency
Latency stable in time
Secondary peaks due only to caches hit/miss and not time correlated
Latency(ms)
0.25
0.20
0.15
0.10
0.05
0.00
0
1000
2000
3000
time
memory
constant
global
shared
texture


Throughput
Throughput (MB/s)
2000
1000
0
0
2000
4000
6000
8000
event size
memory
constant
global
shared
texture
Saturation at ~3GB/s
(I/O bound)


Latency
30
25
20
15
Latency (us)
0
2000
4000
6000
8000
event size
memory
constant
global
shared
texture

Conclusion

- Connect the stand-alone parts together:
  - PF_RING receiver: IP stack removed from kernel. Data goes directly to userland
  - Scheduler: a thread-safe CUDA queue takes data from the NIC ring buffer
  - Kernel: the tests succeeded with excellent results for both the RICH and the CHOD.
- The complete system is expected to work during the Technical Run in November

Reference

- Previous Concurrency Forum slides

- **Real-Time Use of GPUs in NA62 Experiment**

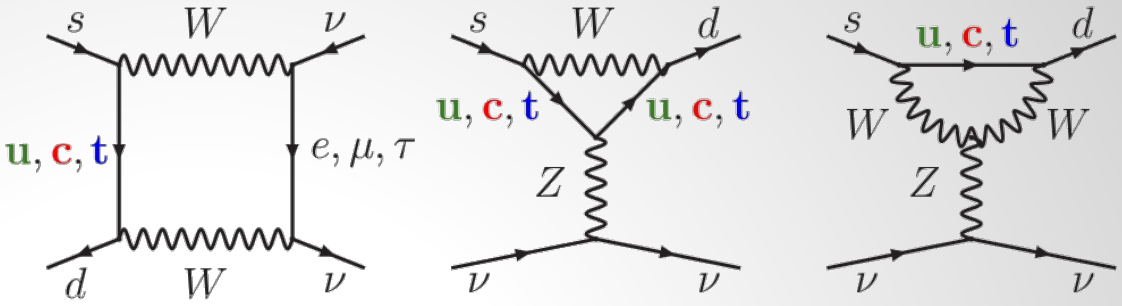
  F. Pantaleo et al., 13th International Workshop on Cellular Nanoscale Networks and their Applications (CERN-PH-EP-2012-260)

Q ∧ A

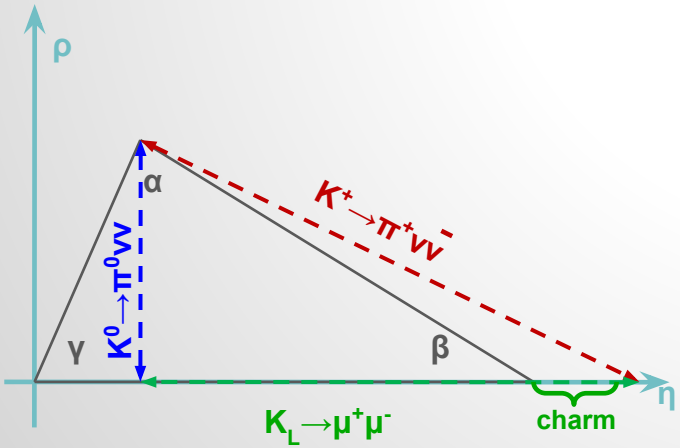
Backup

$K^+ \to \pi^+ \nu\bar{\nu}$ in the Standard Model

. FCNC process forbidden at tree level
. Short distance contribution dominated by Z penguins and box diagrams
. Negligible contribution from u quark, small contribution from c quark
. Very small BR due to the CKM top coupling → λ^5

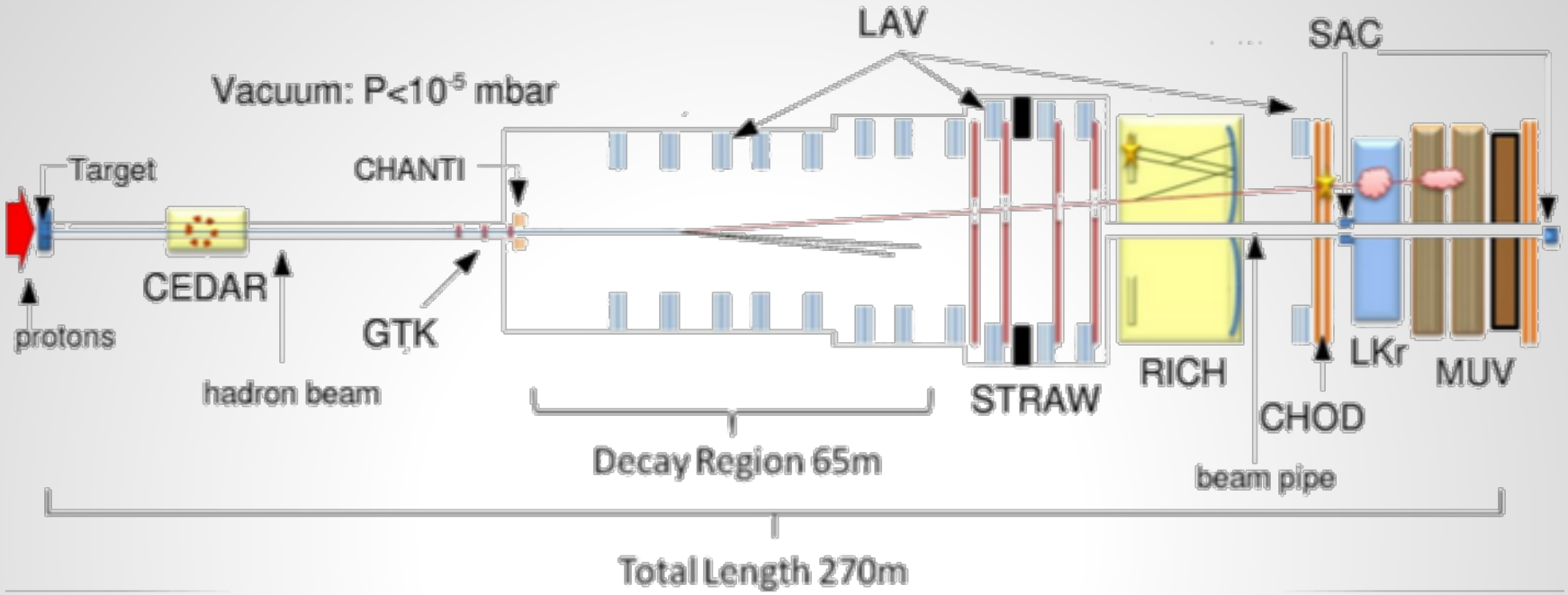


. Amplitude well predicted in SM (measurement of V_{td}) [see E.Stamou]
. Residual error in the BR due to parametric uncertainties (mainly due to charm contributions): ~7%
. Alternative way to measure the Unitarity Triangle with smaller theoretical uncertainty



	G_{SD}/G	Irr. theory err.	BR x 10^{-11}
$K_L \to \pi\nu\nu$	>99%	1%	3
$K^+ \to \pi^+\nu\nu$	88%	3%	8
$K_L \to \pi^0 e^+ e^-$	38%	15%	3.5
$K_L \to \pi^0 \mu^+ \mu^-$	28%	30%	1.5

Experimental technique



. Kaon decay in-flight from an unseparated 75 GeV/c hadron beam, produced with 400 GeV/c protons from SPS on a fixed berilium target

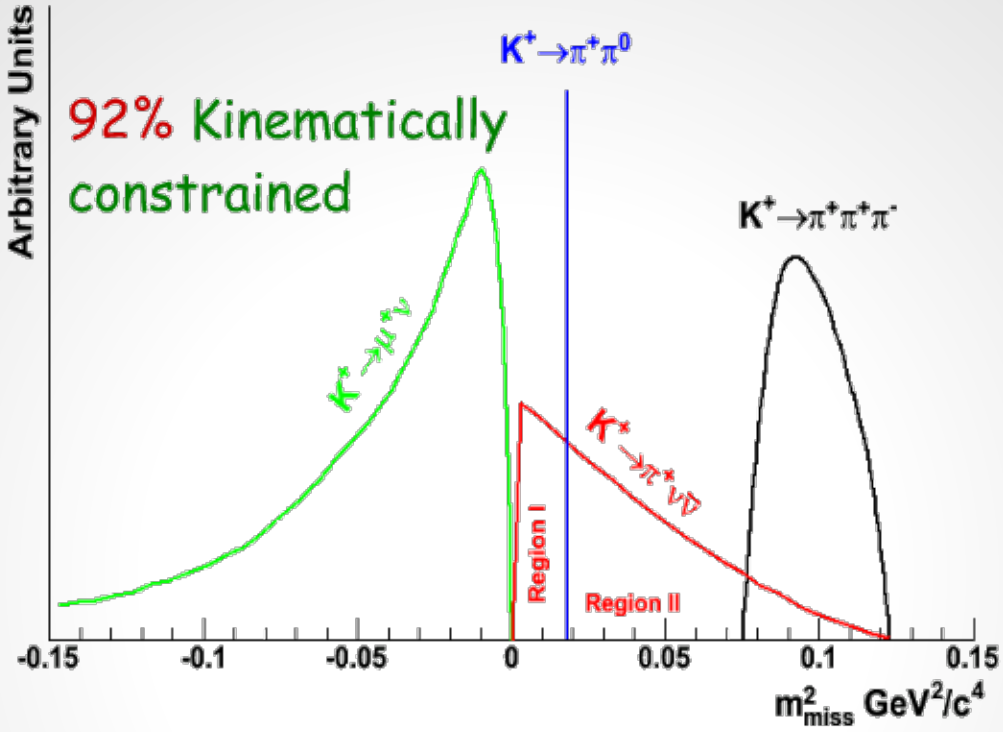
. ~800 MHz hadron beam with ~6% kaons

. The pion decay products in the beam remain in the beam pipe

. **Goal:** measurement of **O(100)** events in two years of data taking with % level of systematics

. Present result (E787+E949): 7 events, total error of ~65%.

Kinematic rejection



. The missing mass will be used to identify **two regions** with lower background level

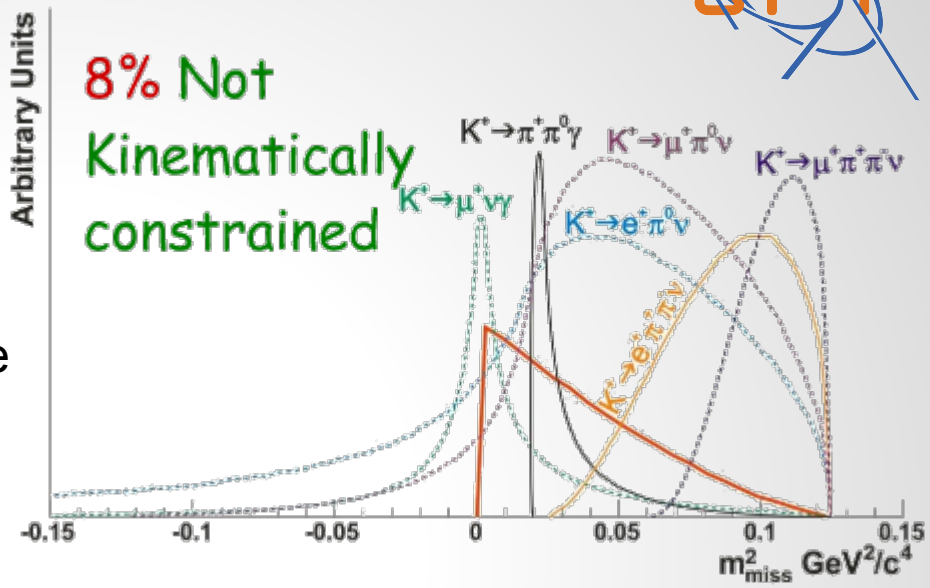
. Very important to have **high resolution** missing mass reconstruction

. Measurement of kaon and pion momenta

. Very light spectrometers to keep the **multiple scattering** as low as possible.

Not kinematically constrained background

. ~8% of the Kaon decays is not kinematically constrained.
. Rejection is based solely on veto and particle identification.
. The veto and PID are exploited to reach the 10^8 rejection factor in the kinematical constraint background



Veto system requirements:

. Large angle (8.5-50 mrad): inefficiency $<10^{-4}$ for gamma between 100 MeV and 35 GeV
. Forward veto (1-8.5 mrad): inefficiency $<10^{-5}$ for E>10 GeV
. Small angle (<1 mrad): $<10^{-3}$ for E>10 GeV

PID system requirements:

. Positive kaon identification in the hadron beam
. π-μ separation: 10^{-3} mis-identification probability

NA62 Trigger

- L0 selection: RICH+!LKR+!MUV3
- RICH: hit multiplicity positive signal
- !LKR: no 2 clusters more than 30 cm apart
- !MUV3: no signal in MUV3

- Very good time resolution is required to avoid random veto
- At the software levels a more complete analysis will be performed (missing mass, Z vertex,…)

	Initial rate (MHz)	After L 0 (MHz)
$\pi\pi^0$	1.9	0.22
$\mu\nu$	5.7	0.04
$\pi\pi\pi$	0.5	0.1
$\pi\pi\pi\pi^0$	0.16	0.002
$\pi^0 e\nu$	0.3	0.05
$\mu\nu\pi^0$	0.2	0.002
TOT	6.7	0.4
$\pi\nu\nu$ (eff.)		82%

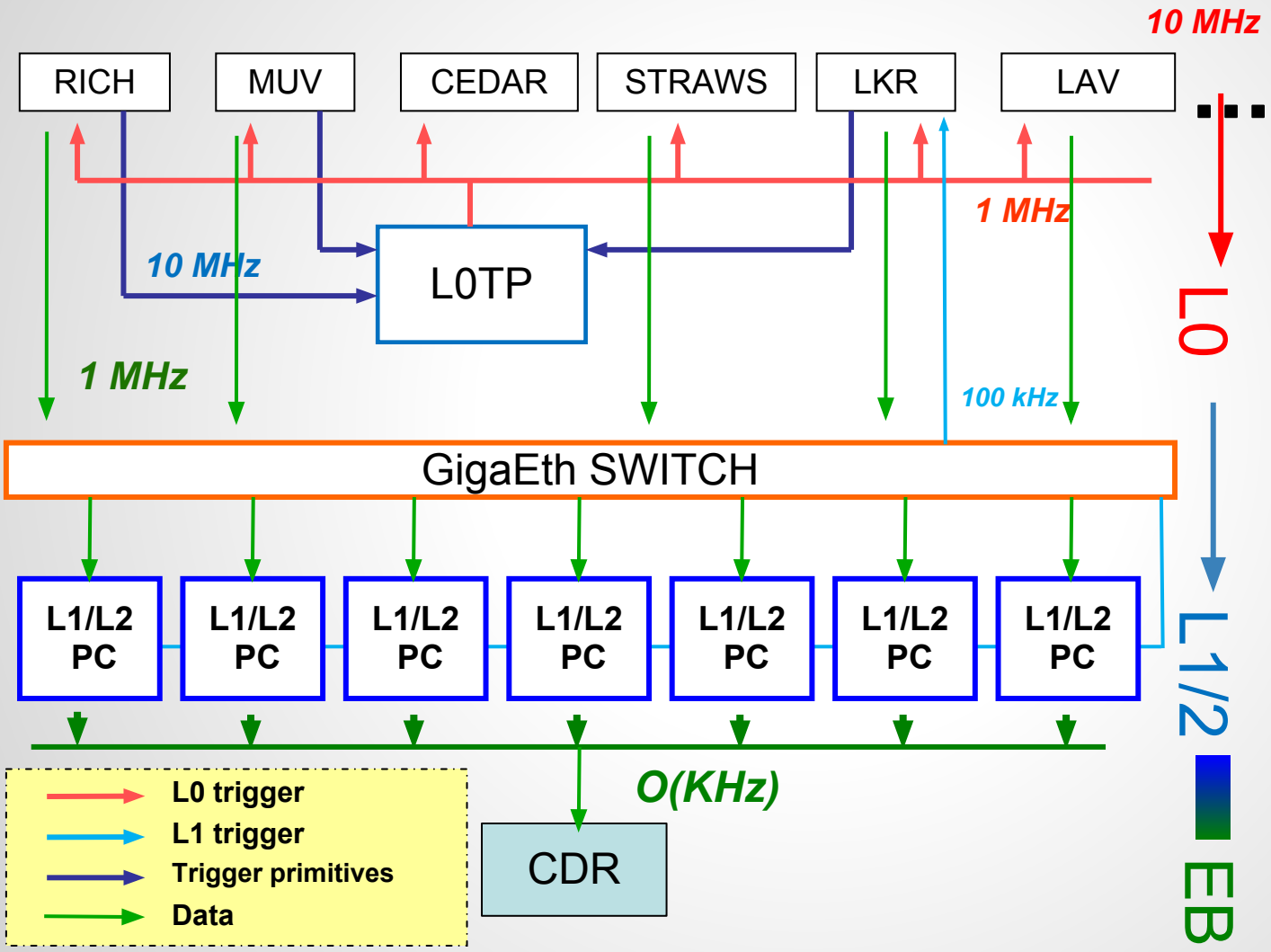
		Input (max)	Output (max)	latency
L0	hw,sync	~10 MHz	~ 1 MHz	1 ms
L1	soft,async	~ 1 MHz	~ 100 kHz	undefined
L2	soft,async	~ 100 kHz	O(kHz)	undefined

NA62 sensitivity

$K^+ \to \pi^+ \nu\nu$ (signal)	55 events/year
$K^+ \to \pi^+ \pi^0$	4.3% (2.3 evts)
$K^+ \to \mu^+ \nu$	2.2% (1.2 evts)
$K^+ \to \pi^+ \pi^- e \nu$	<3% (1.7 evts)
3 tracks	<1.5% (0.8 evts)
$K^+ \to \pi^+ \pi^0 \gamma$	2% (1.1 evts)
$K^+ \to \mu^+ \nu \gamma$	0.7% (0.4 evts)
others	negligible
Expected bkg	<13.5% (7.4 evts)

- $4.8 \cdot 10^{12}$ decays per year
- x50 wrt NA48 flux (same amount of protons from SPS)
- π^0 rejection $2 \cdot 10^8$
- O(10%) signal acceptance
- 100% trigger efficiency assumed

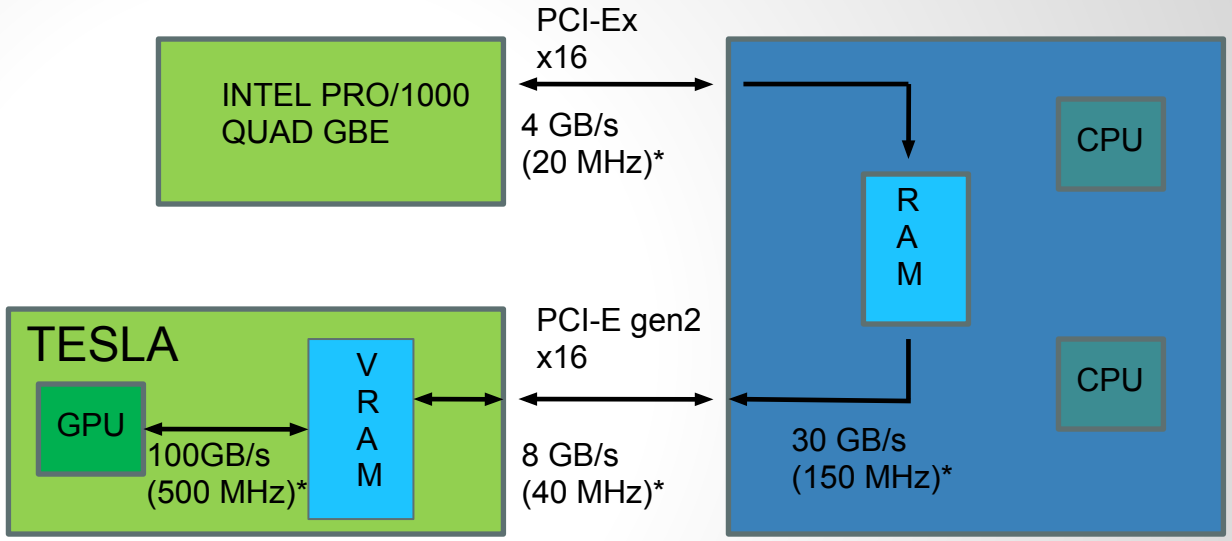
The NA62 TDAQ system

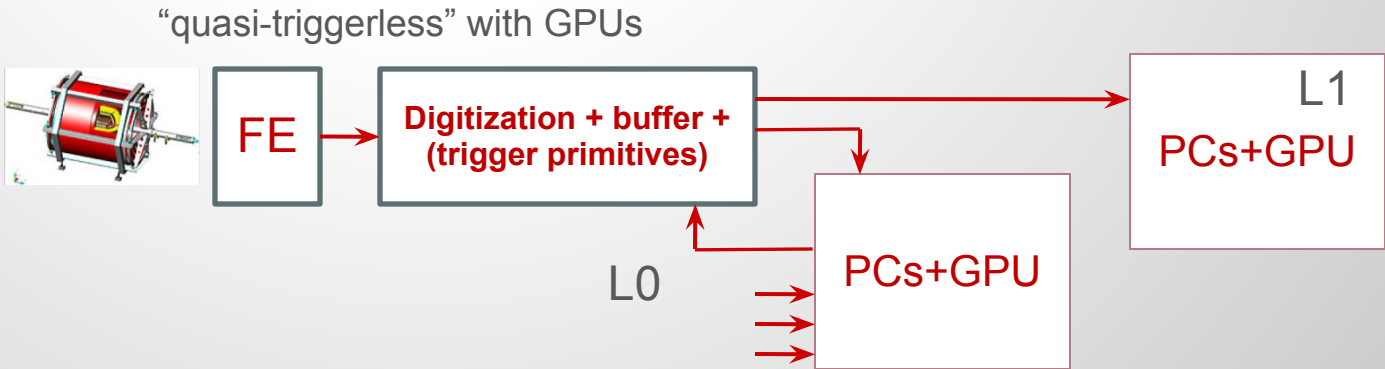


L0: Hardware synchronous level. 10 MHz to 1 MHz. Max latency 1 ms.
L1: Software level. “Single detector”. 1 MHz to 100 kHz
L2: Software level. “Complete information level”. 100 kHz to few kHz.

GPU as a Level0 trigger

- The idea: exploit GPUs to perform high quality analysis at trigger level
- GPU architecture: massive parallel processor SIMD
- "Easy" at L1/2, challenging at L0
- Real benefits: increase the physics potential of the experiment at very low cost!
- Profit from continuative developments in technology for free (Video Games,…)





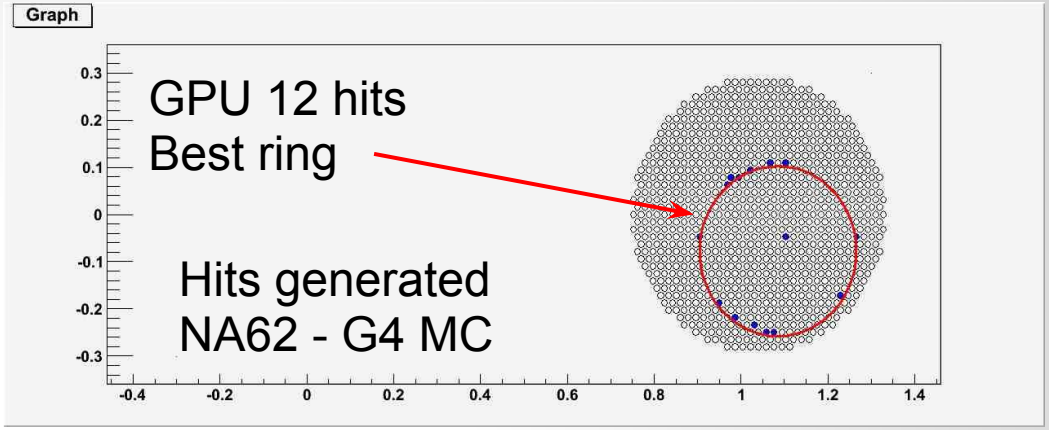
Hardware on the workbench

- GPU: NVIDIA Tesla C2050
  - 448 CUDA cores @ 1.15GHz
  - 3GB GDDR5 ECC @ 1.5GHz
  - CUDA CC 2.0 (Fermi Architecture)
  - PCIe 2.0 (effective bandwidth up to ~5GB/s)
  - CUDA Runtime v4.1, driver v295.20 (Feb '12)
- CPU: Intel® Xeon® Processor E5630 (released in Q1 '10)
  - 2 CPUs, 8 total cores (16 threads SMT) @2.53GHz
- SLC6, GNU C compiler v4.4.6

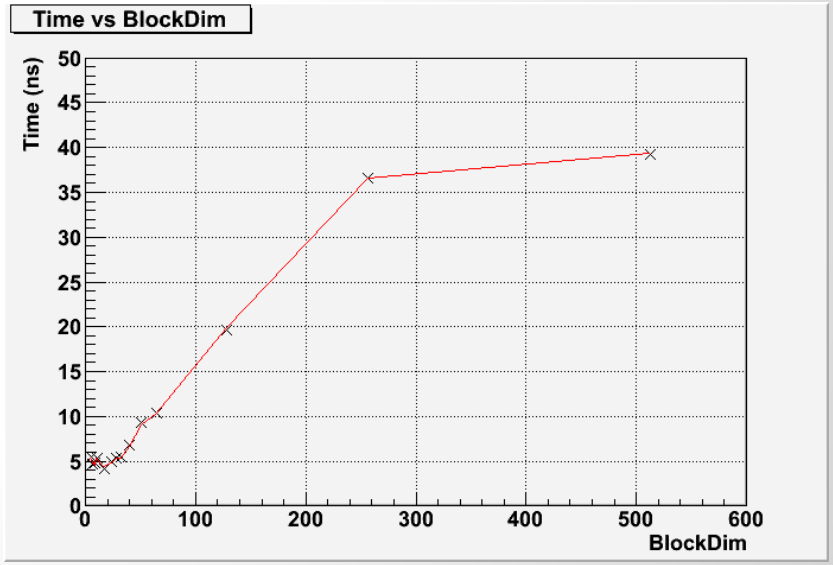
Check if possible to fake the ECC on GTX card between spills

GPU trigger

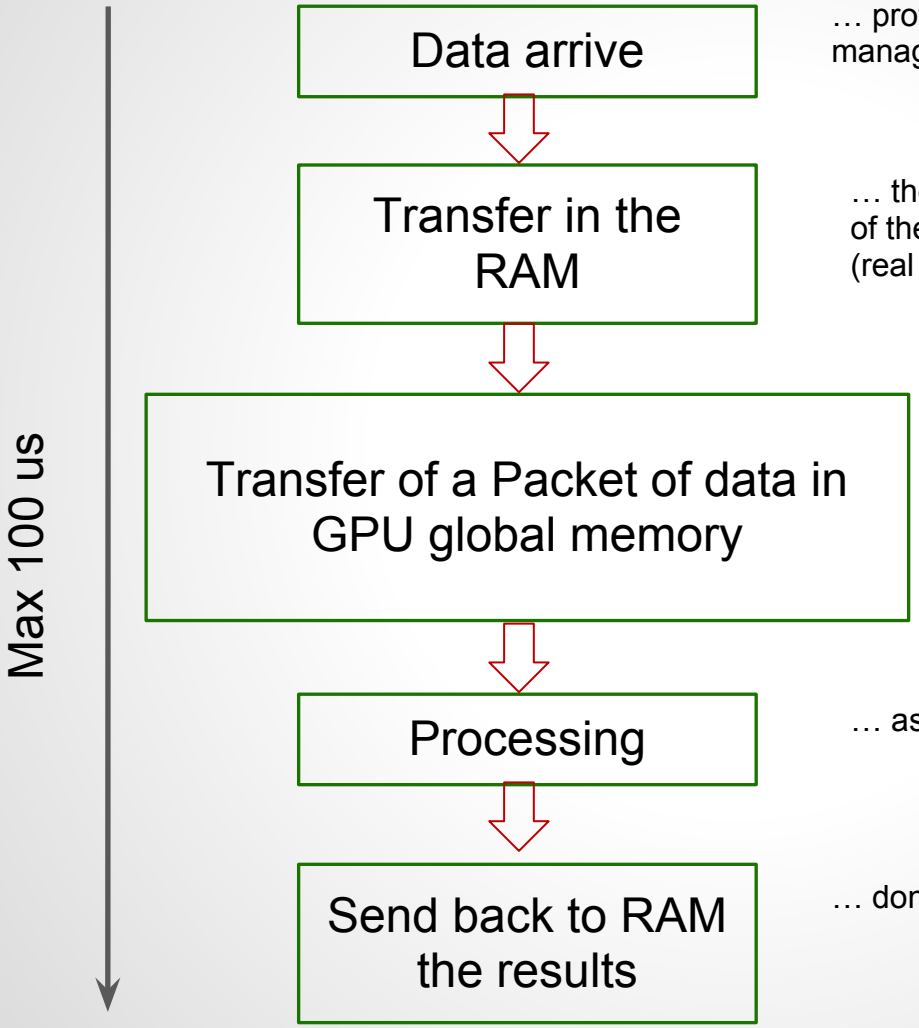
- Natively built for pattern recognition problems
- First attempt: ring reconstruction in RICH detector.



- Several algorithms tested → best result 5 ns/per ring .
- Long latency in data transfer from PC to Video Card → avoided by transferring a packet of 1000 events.
- Total processing time well below 1 ms (per 1000 events) in a single PC!
- Pilot project, very promising R&D.



GPU



… protocol stack possibly managed in the receiver card

… the non deterministic behavior of the CPU should be avoided (real time OS)

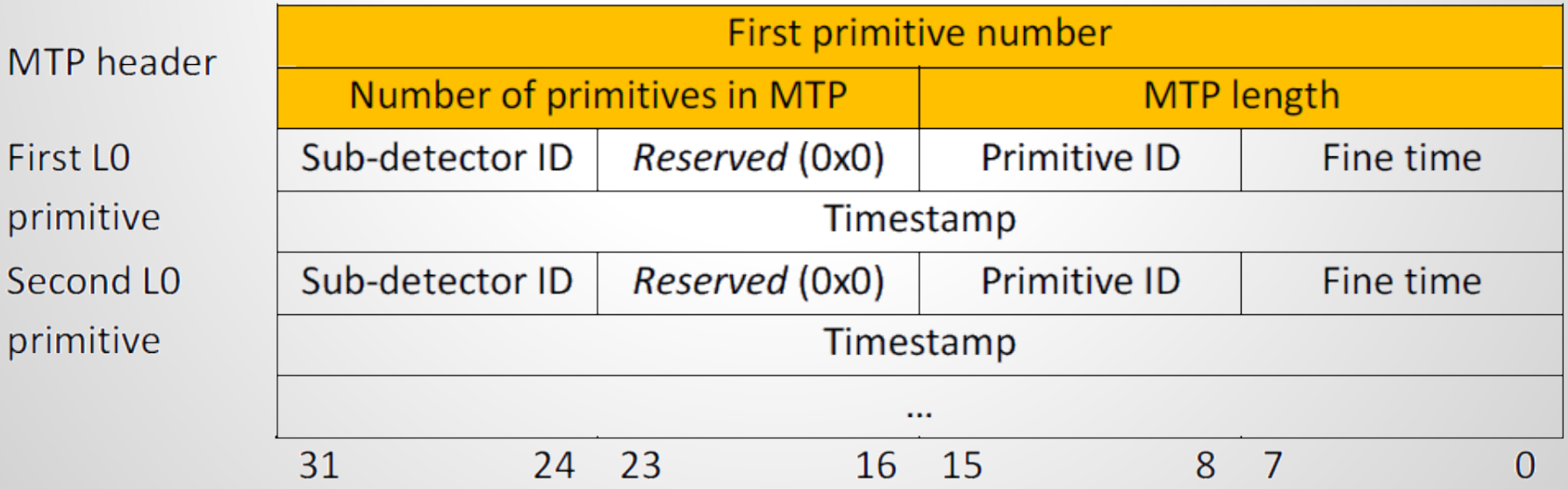
… the PCI-ex gen2 is fast enough. Concurrent transfer during processing.

… as fast as possible!!!

… done!

Multi-trigger packets

- Sources asynchronously produce L0 Trigger Primitive information
- L0 Trigger Processor time-matches the primitives and possibly issues (synchronous) L0 triggers with a fixed latency
- Maximum latency 1ms (possible extension in the future)
- L0 Trigger Primitives are sent via dedicated ethernet links, packed into MTPs (see below for a possible solution)
- Expected peak bandwidth from single sub-detector ~500MB/s (@10MHz expected particle rate)



Are the GPUs ready for this?

Main issues:

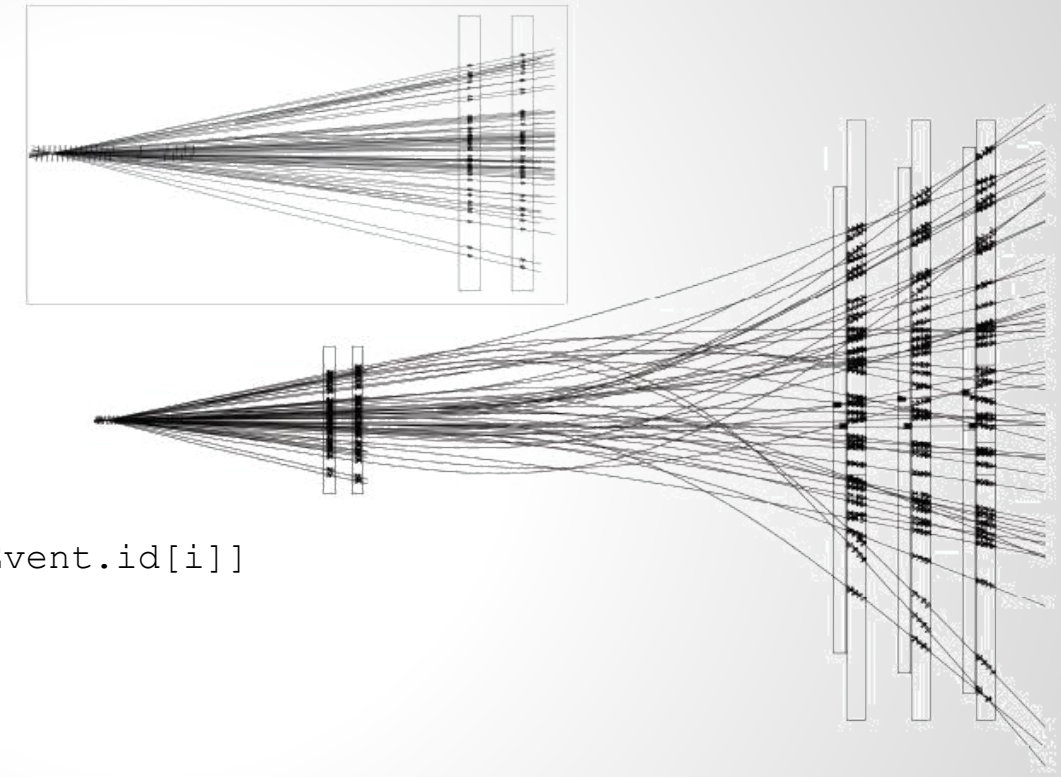
1. Memory throughput sufficient?
2. How to solve overhead problems?
3. What about the maximum latency?

Hit Counting Problem

Suppose we have a detector with `NCHANS` channels, each one with a fixed `threshold[i]`.

A particle crosses the detector (`Event`) activating the channel `id` and releasing the energy `adc`

Each channel is then activated iff the threshold is exceeded, i.e.:



```
ok[i] = Event.adc[i] > threshold[Event.id[i]]
```

Then, the `Event` is considered "triggered" if the number of `"OKs"` exceeds a second threshold `TH`.

We don't know, until an event is generated, how many and which channels are hit and how much energy has been released in that channel.
The minimal output from the GPU is the identification of the Event if it triggered, nothing if not.
This is a first very (very) simple algorithm to use as benchmark, actually the most trivial possible!

Hit Counting Problem - 2

We have to make the first choice:

How do we transfer data on the GPU?

X a. shall we move everything as it comes into the buffer?

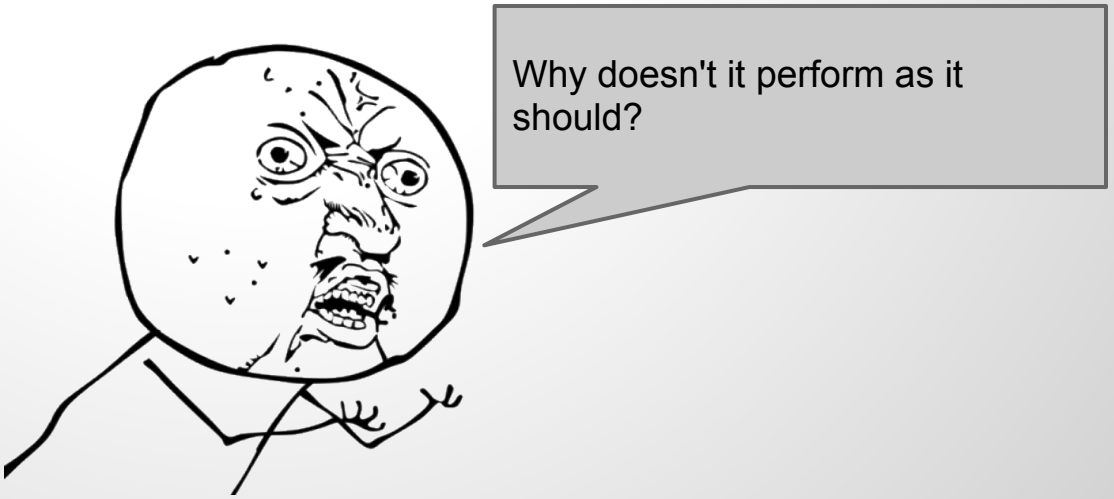
X b. shall we wait until a "good" size has been reached, send everything to the GPU and then unpack there?

V c. shall we organize data first so that they are easier to read, and elaborate? (see memory coalescence, SoA vs AoS.....)

Hit Counting Problem - 3

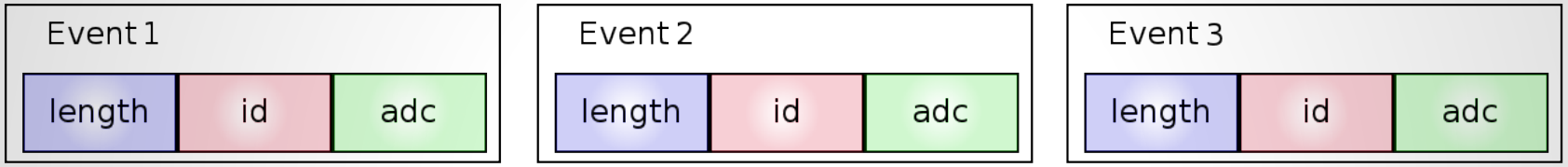
We cannot move data as they come for several reasons:

- overhead (reduced collecting more package, i.e. option b)
- not coalesced access pattern
- many threads idle due to matrix-shaped thread organization in the 3-dimensional grid

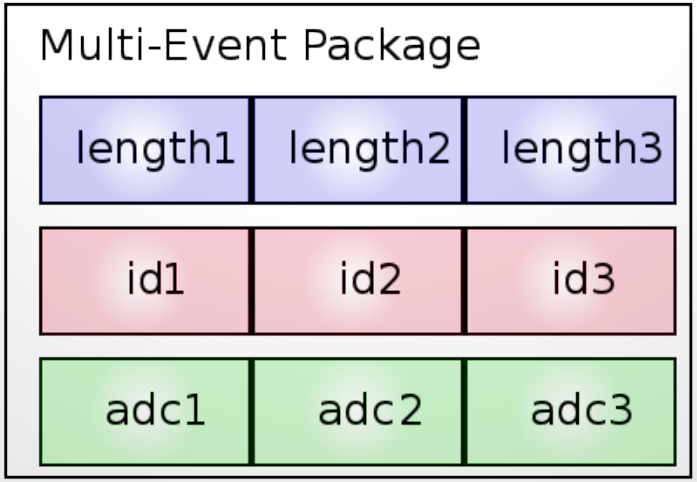


Organization of data: AoS vs SoA

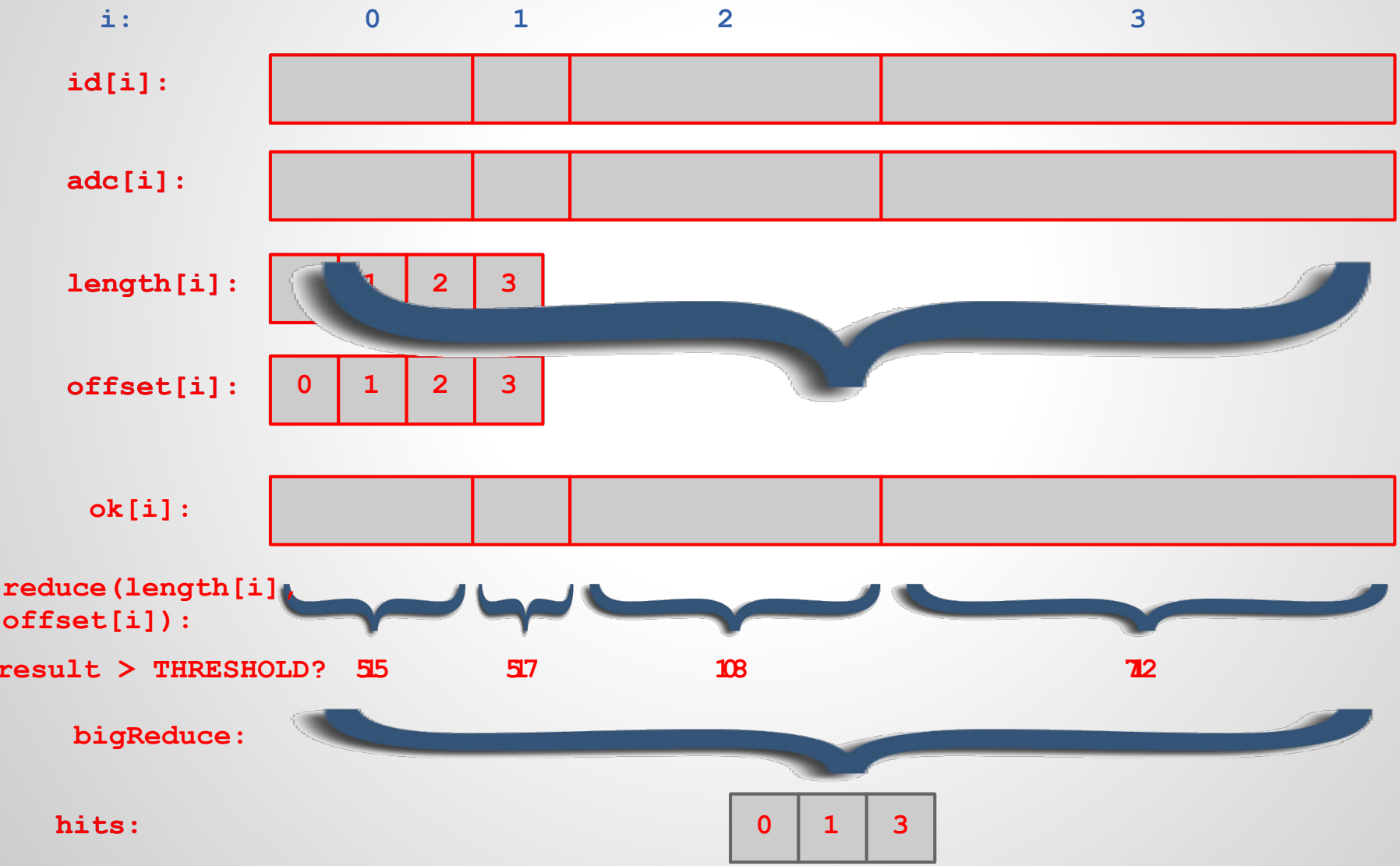
- Data come in Arrays of Structures:



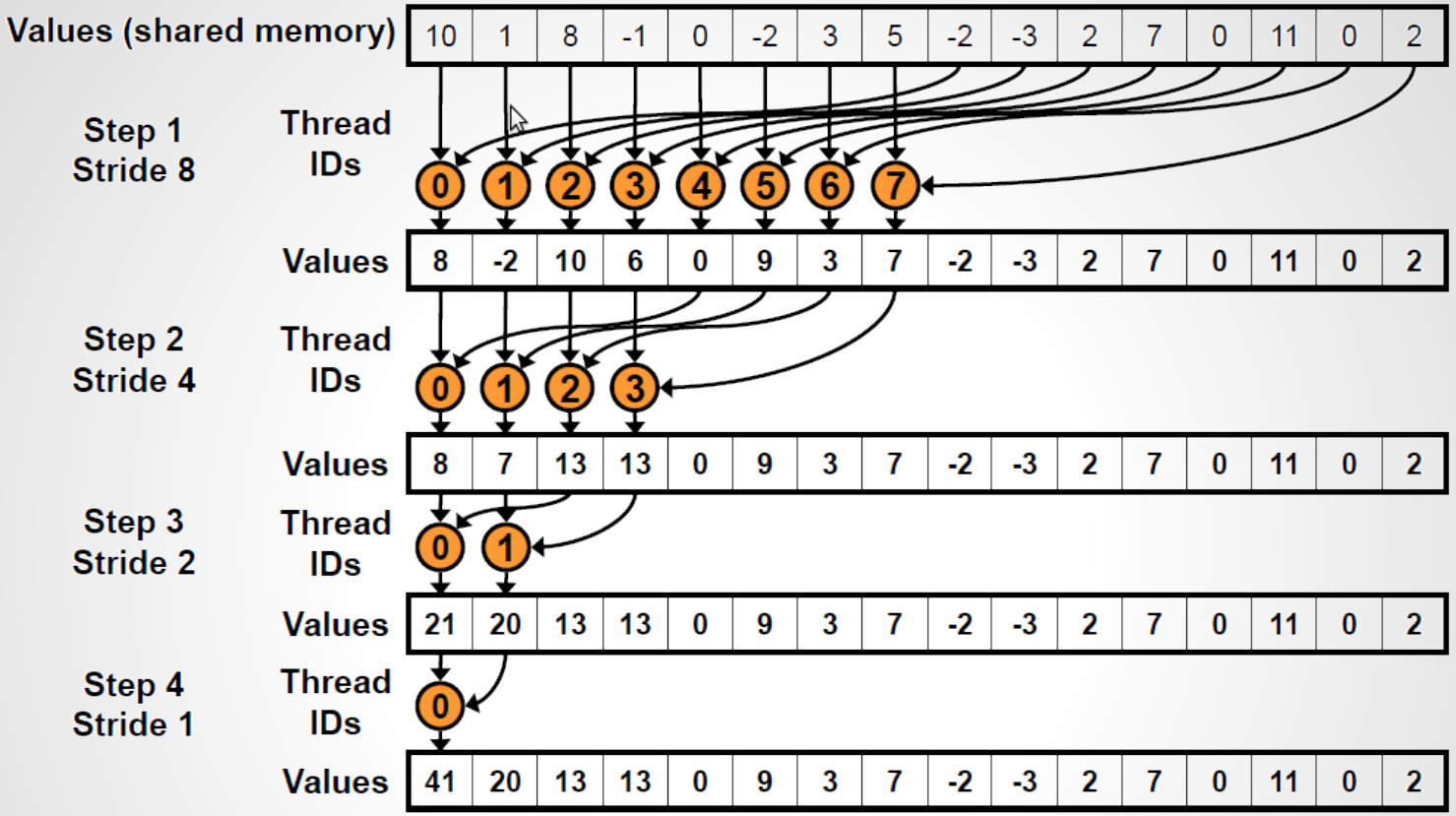
- Compilers and Hardware digest Structure of Arrays a lot better than AoS (+60% gain in hcp):



Layout


i:
0
1
2
3
id[i]:
adc[i]:
length[i]:
1
2
3
offset[i]:
0
1
2
3
ok[i]:
reduce(length[i],
offset[i]):
result > THRESHOLD?
515
517
108
712
bigReduce:
hits:
0
1
3

Reduction



- Must use sequential addressing instead of interleaved addressing to avoid Shared Memory bank conflicts
- Time complexity is O(logN), cost is O(N*logN): not cost efficient
- Brent's theorem (algorithm cascading) suggests O(N/logN) threads:
  - Each thread does O(logN) sequential work
  - All O(N/logN) threads cooperate for O(logN) steps
  - New cost = O(N/logN * logN) = O(N)

Concurrent Memcpy - Kernel

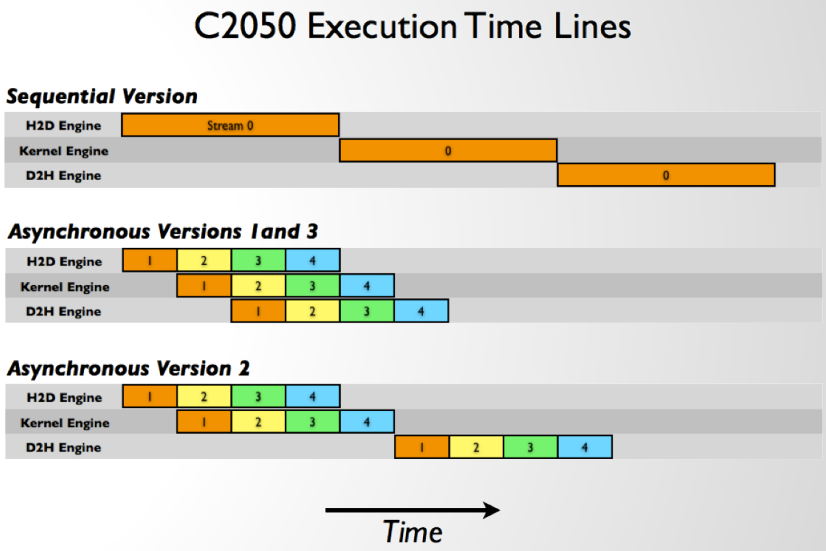
The main bottleneck is the PCIe 2 (real bandwidth is 5GB/s vs theoric 8GB/s)

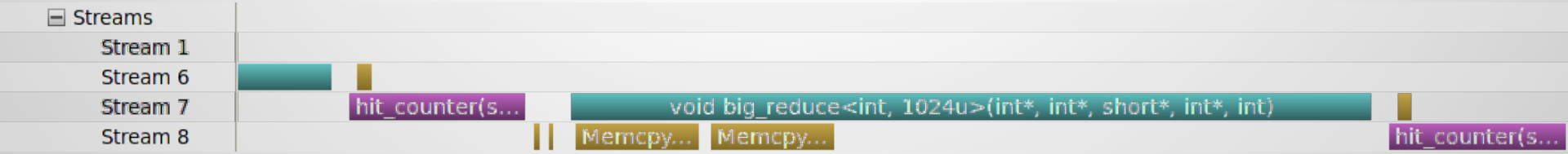
While waiting for PCIe 3 (double bandwidth), we can exploite streaming.

Streaming allows overlapping between kernel execution time and transfer time of independent data between CPU and GPUs.

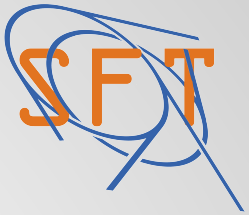
The flow of the application has to be reorganized using 3 concurrent streams (3 is the minimum: 1 for reading, 1 for kernel execution, 1 for reading. We can actually use more but it's just an exercise and we wanted to be pessimistic):

- 3X memory occupation (memory does not seem to be a problem)
- Events must be packed in Multi-Event Packets, and partitioned in 3 buffers (circular buffers or whatever)





Memory Allocation



Avoid, when possible, repeated dynamic allocation/deallocation of memory:
the time spent doing `cudaMalloc()` or `cudaFree()` is t ~ 0.1ms.

Allocate the maximum possible size at the beginning of the program and then just copy from/to memory locations using data inside the arrays `length[]` and `offset[]`.

Modern NVIDIA Tesla Cards can have up to 6GB GDDR5 on board.

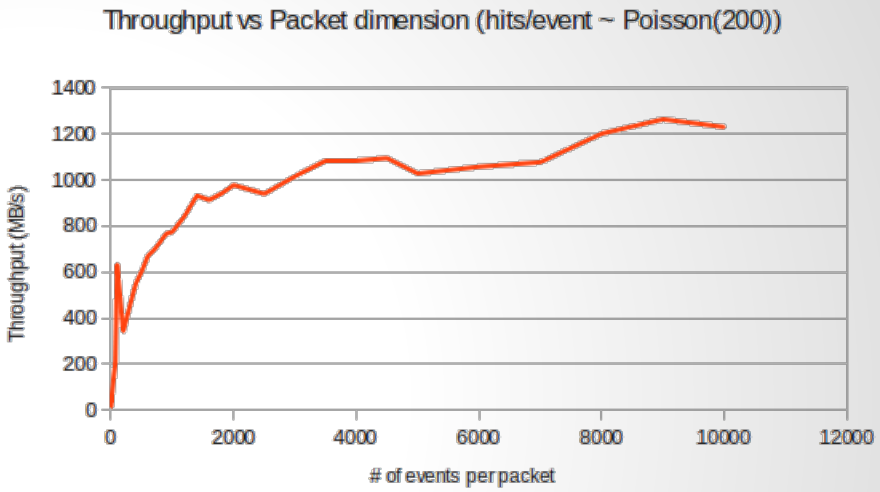
Preliminary Results - Throughput

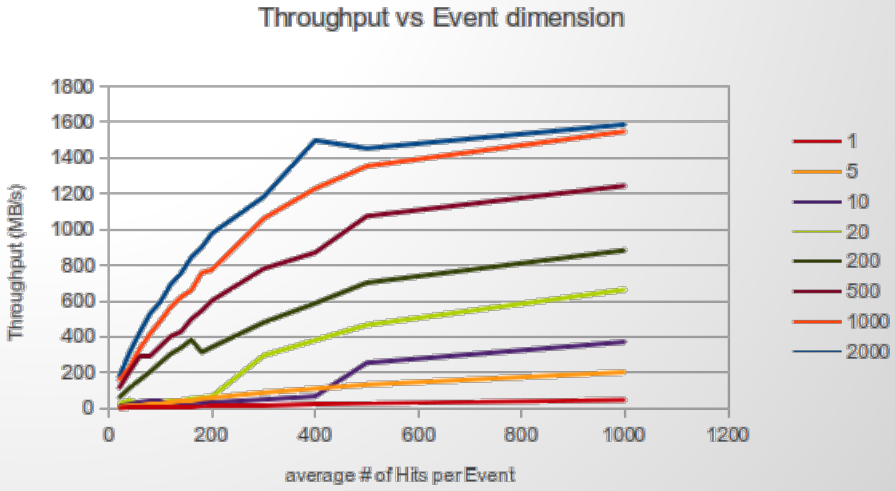
The throughput behaviour for a varying number of events inside a packet is a typical many-core device behaviour:

- constant time to process a varying number of events, activating more SMs as the packet size increases
- discrete oscillations due to the discrete nature of the GPU
- saturation plateau (1.2GB/s)

The right choice of packet dimension is not unique.
It depends on the maximum latency we don't want to exceed and on the input rate of events.

Considering that the maximum rate per subdetector (@10MHz particles rate) for NA62 experiment is O(500MB/s), I would consider the throughput test **PASSED**



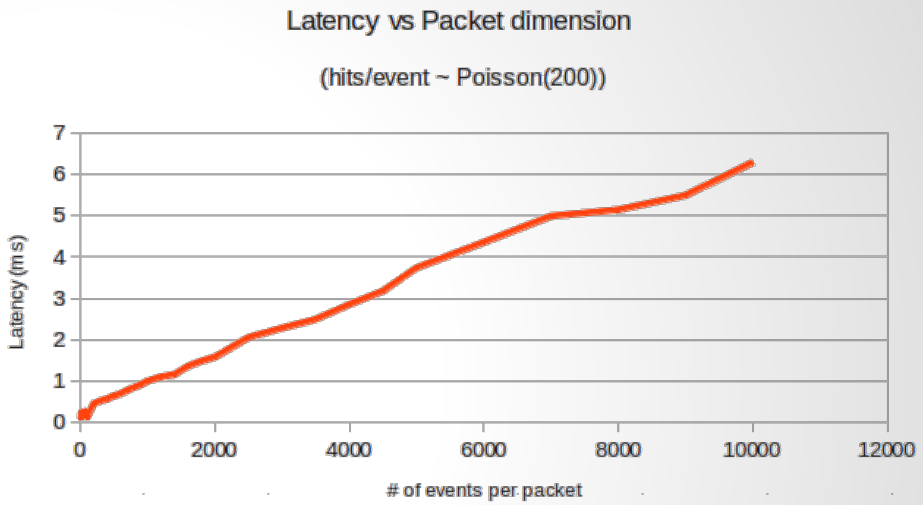


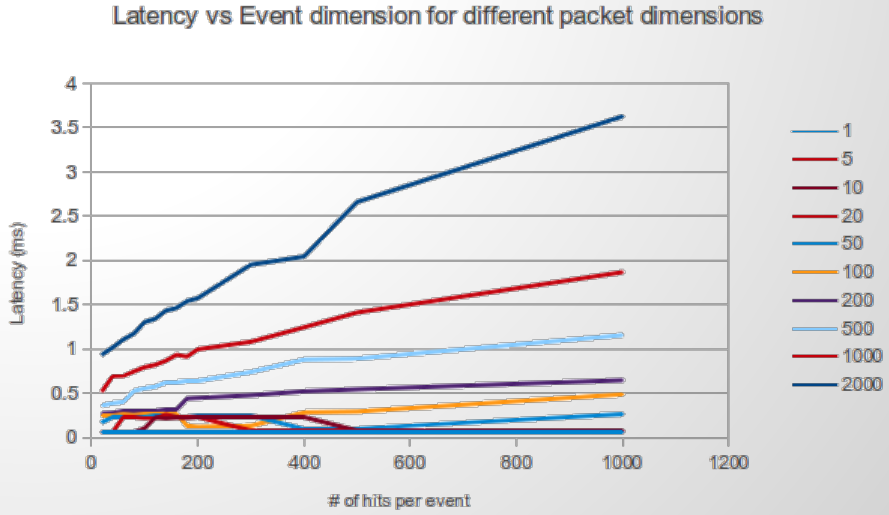
Preliminary Results - Latency

Latency pretty stable wrt event size.

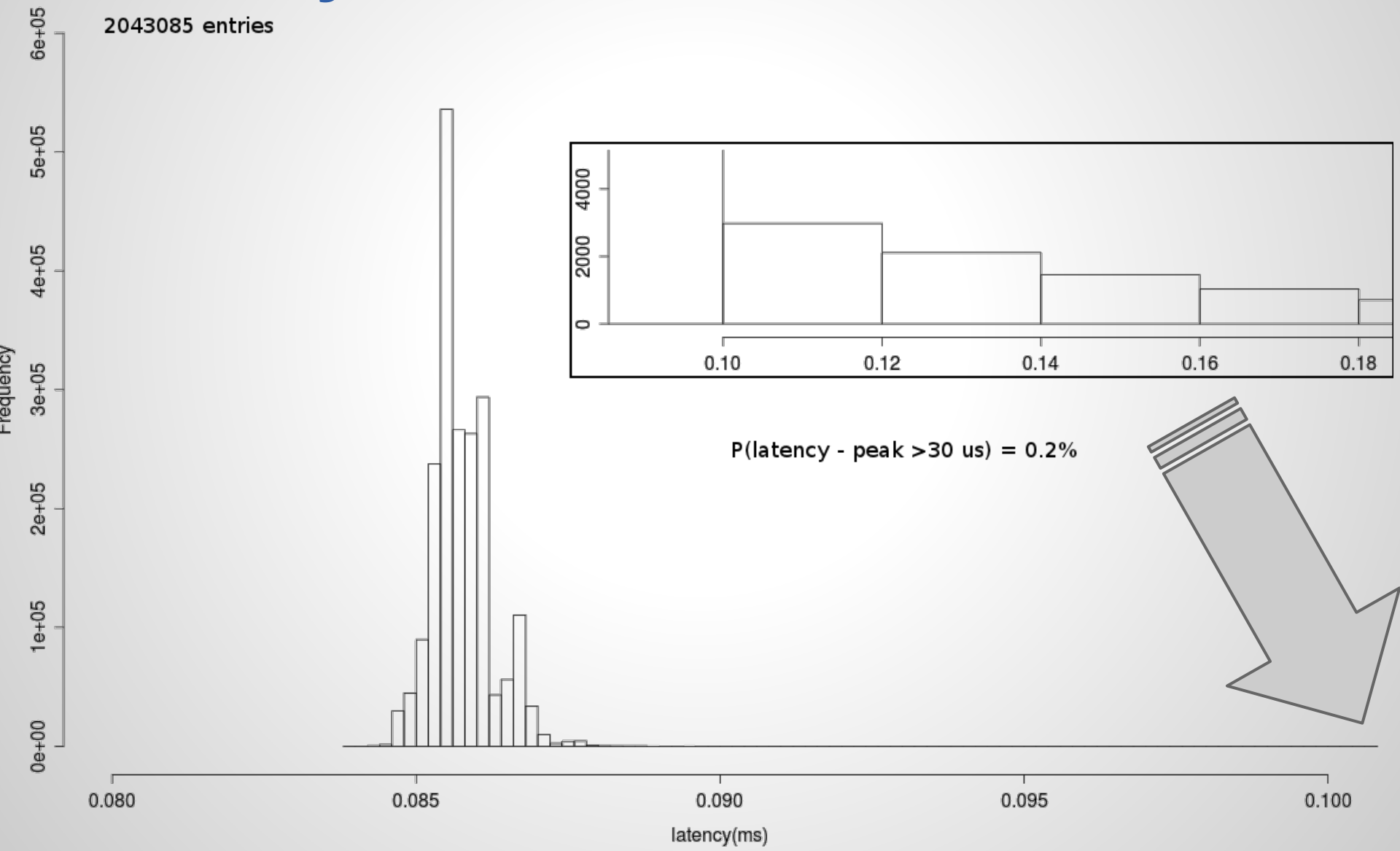
A lower number of event inside a package is better to achieve a low latency. A larger number of event guarantees a better performance and lower overhead.

The choice depends on the technical requirements.

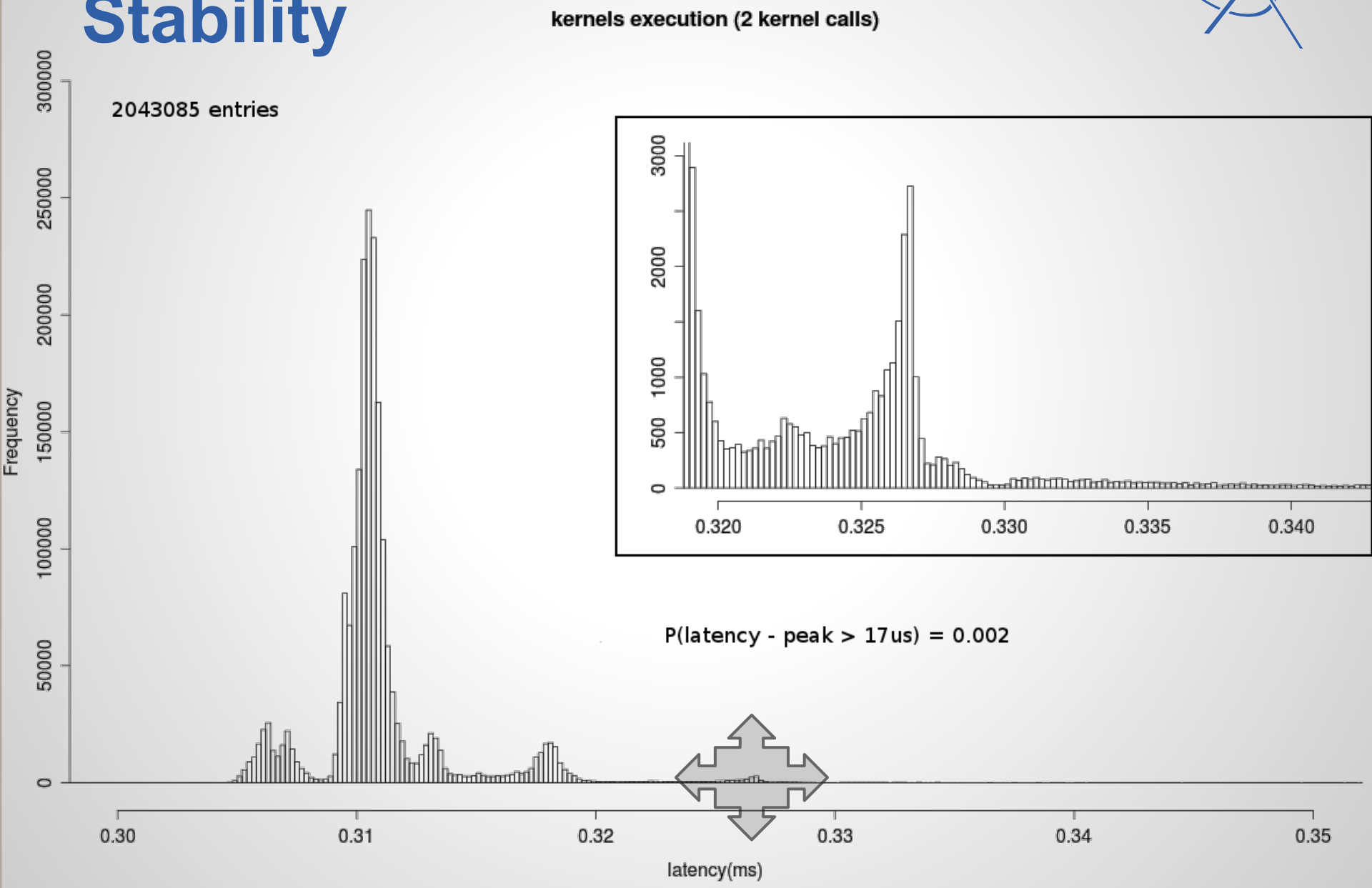




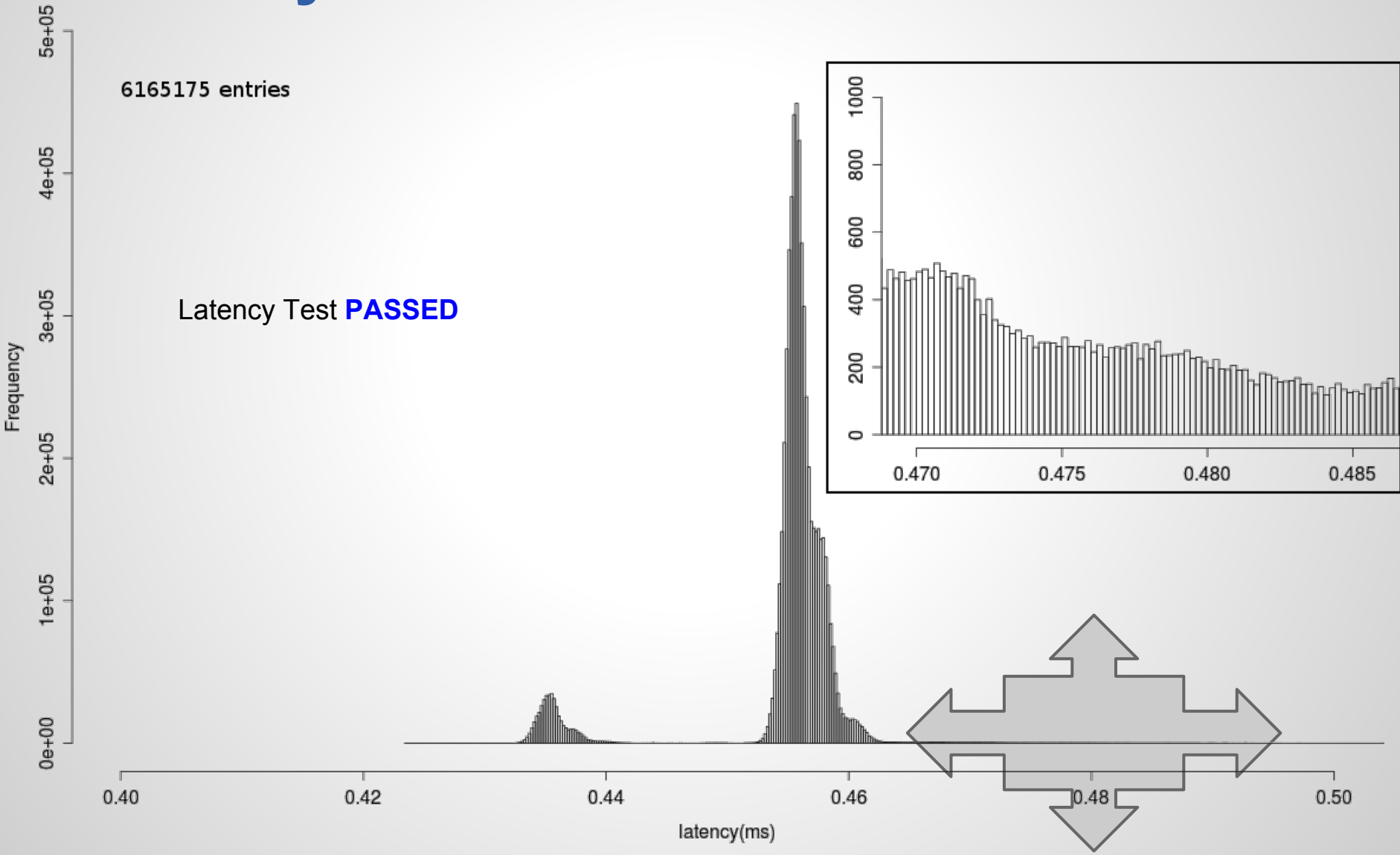
Preliminary Results - Latency Stability



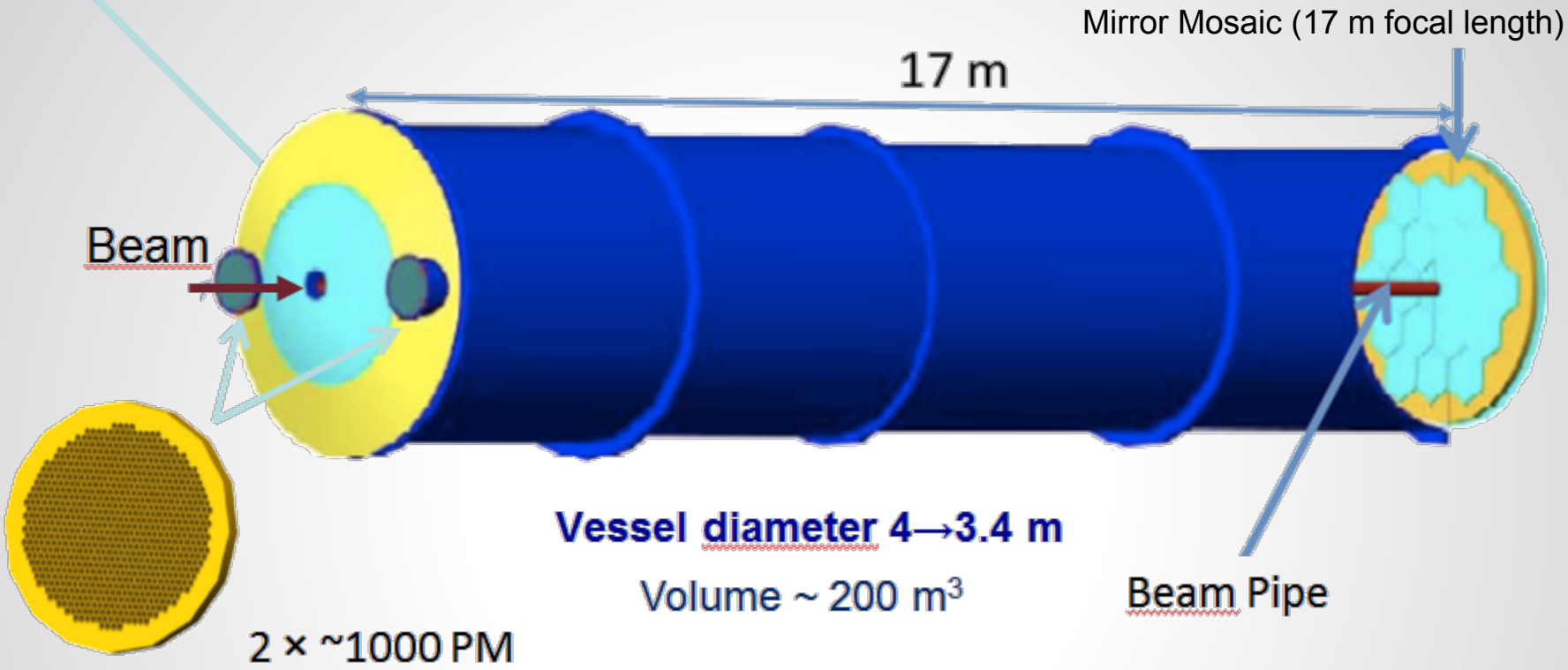
Preliminary Results - Latency Stability



Preliminary Results - Latency Stability



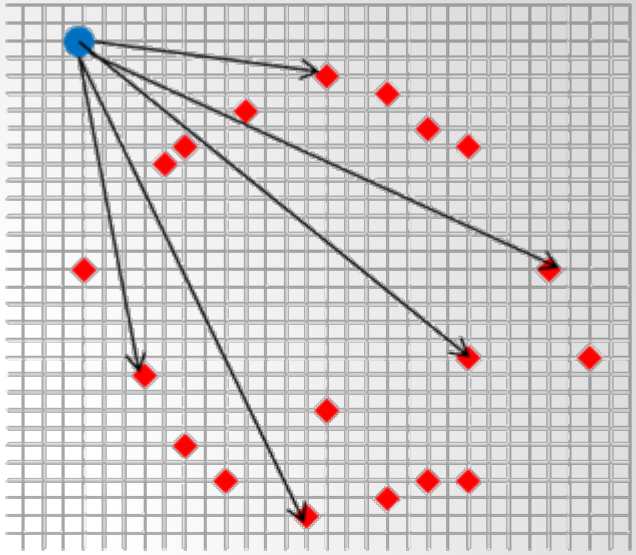
First application: RICH



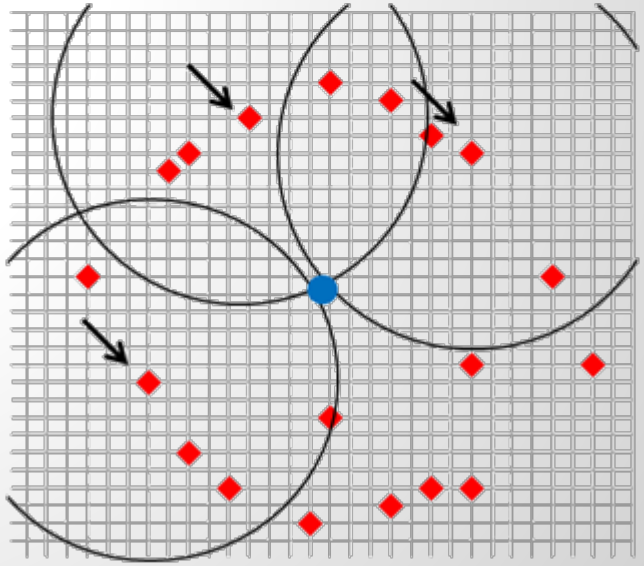
- ~17 m RICH
- 1 Atm Neon
- Light focused by two mirrors on two spots equipped with ~1000 PMs each (pixel 18 mm)
- 3s p-m separation in 15-35 GeV/c, ~18 hits per ring in average
- ~100 ps time resolution, ~10 MHz events rate
- Time reference for trigger

Algorithms for single ring search (1)

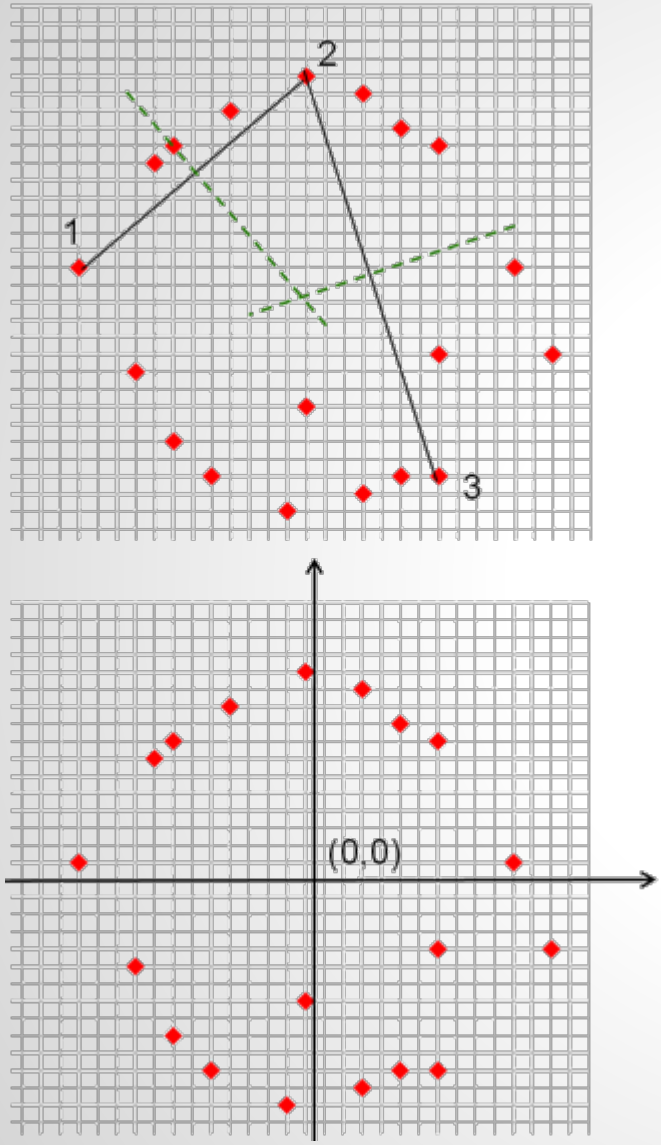
- DOMH/POMH: Each PM (1000) is considered as the center of a circle. For each center an histogram is constructed with the distances between center and hits.



- HOUGH: Each hit is the center of a test circle with a given radius. The ring center is the best matching point of the test circles. Voting procedure in a 3D parameters space



Algorithms for single ring search (2)

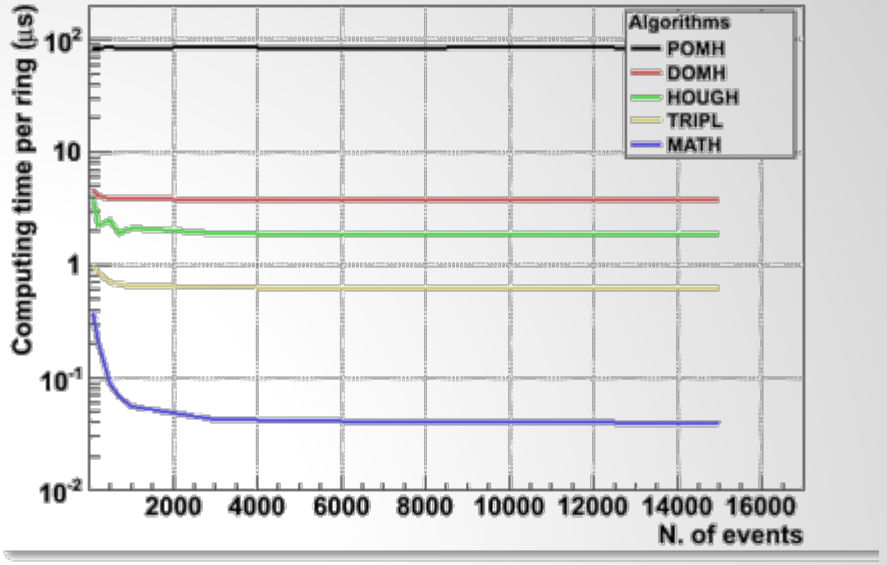


- TRIPL: In each thread the center of the ring is computed using three points (“triplets”). For the same event, several triplets (but not all the possible) are examined at the same time. The final center is obtained by averaging the obtained center positions.

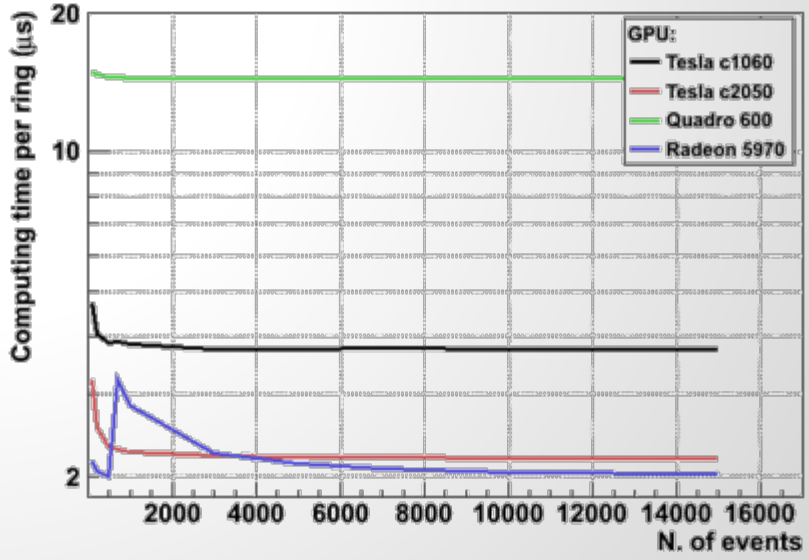
- MATH: Translation of the ring to centroid. In this system a least square method can be used. The circle condition can be reduced to a linear system , analytically solvable, without any iterative procedure.

Processing time

- Using Monte Carlo data, the algorithms are compared on Tesla C1060
- For packets of >1000 events, the MATH algorithm processing time is around 50 ns per event

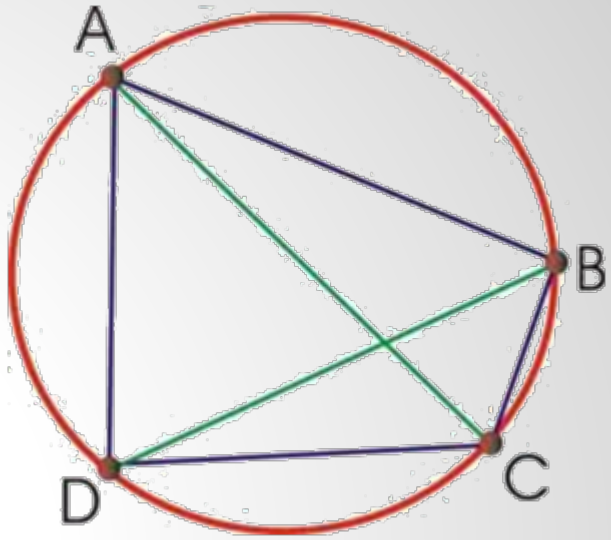


- The performance on DOMH (the most resource-dependent algorithm) is compared on several GPUs
- The gain due to different generation of video cards can be clearly recognized.



GPU@ L0/L1 RICH trigger

- After the L0: ~50% 1 track events, ~50% 3 tracks events
- Most of the 3 tracks, which are background, have max 2 rings per spot
- Standard multiple rings fit methods are not suitable for us, since we need:
- Trackless
- Non iterative
- High resolution
- Fast: ~1 us (1 MHz input rate)

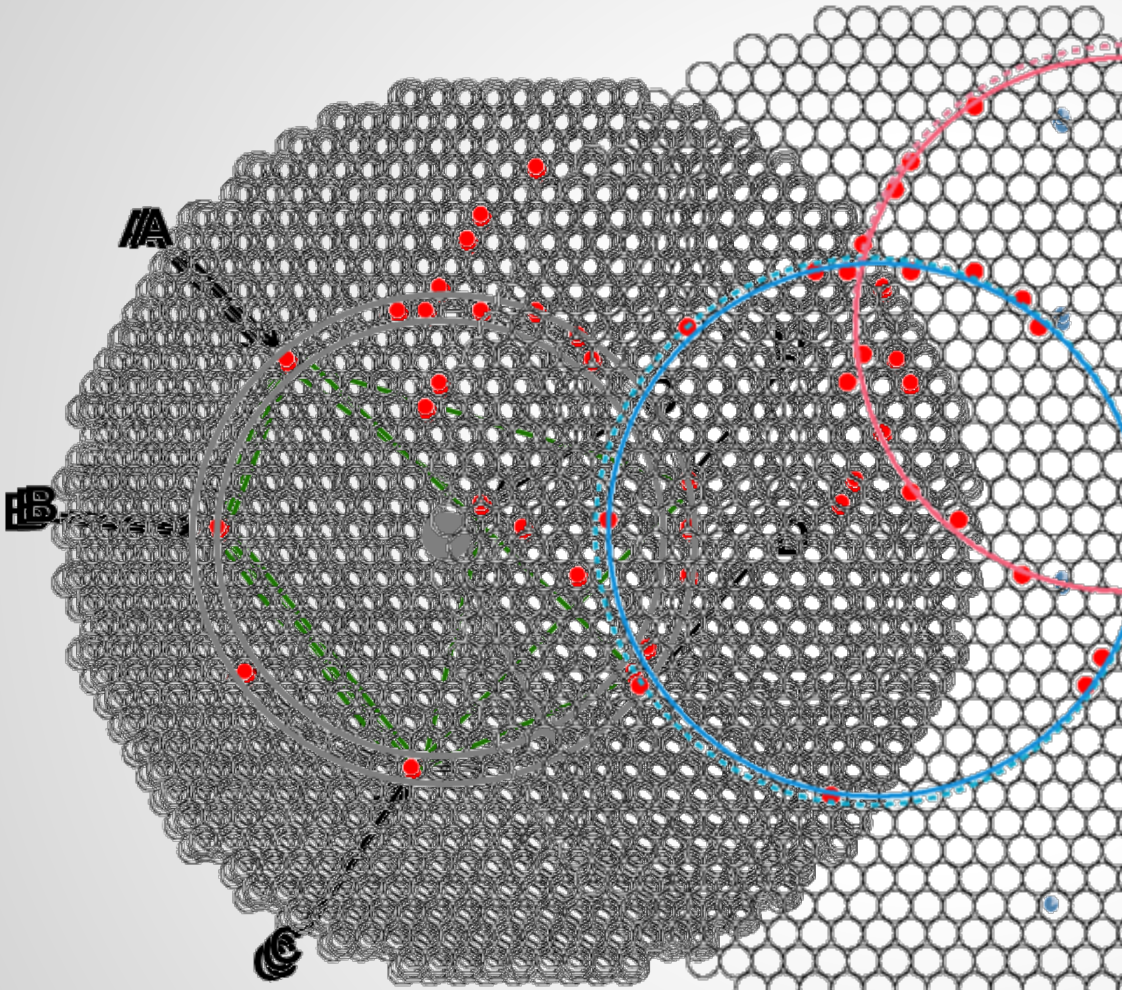


- New approach→ use the Ptolemy's theorem (from the first book of the Almagest)

"A quadrilater is cyclic (the vertex lie on a circle) if and only if is valid the relation:

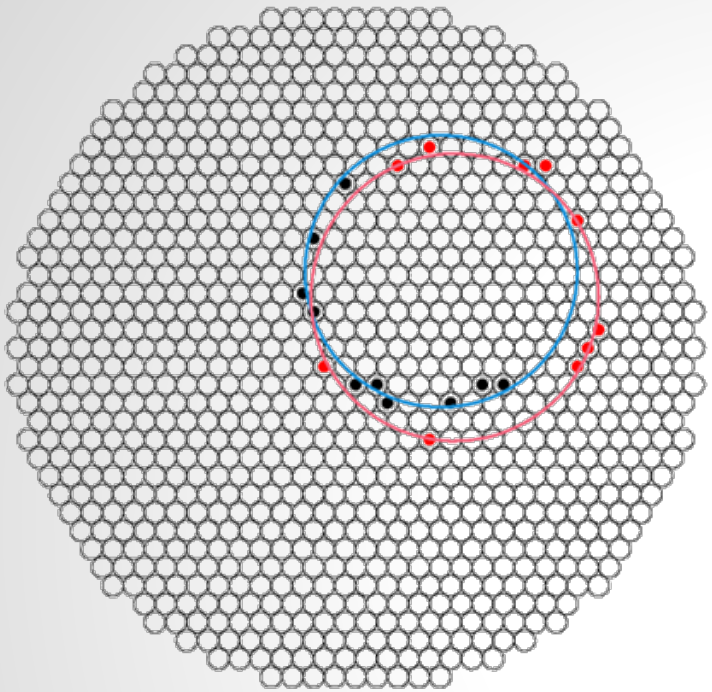
$$AD*BC+AB*DC=AC*BD$$ "

Almagest algorithm description



- Select a triplet randomly (1 triplet per point ≡ N+M triplets in parallel)
- Consider a fourth point: if the point doesn't satisfy the Ptolemy theorem reject it
- If the point satisfy the Ptolemy theorem, it is considered for a fast algebraic fit (i.e. math, pernarun sphere, tobin, ...).
- Each thread converges to a candidate center point. Each candidate is associated to Q quadrilaters contributing to his definition
- For the center candidates with Q greater than a threshold, the points at distance R are considered for a more precise re-fit.

Almagest algorithm results



- The real position of the two generated rings is:

1→ (6.65, 6.15) R=11.0

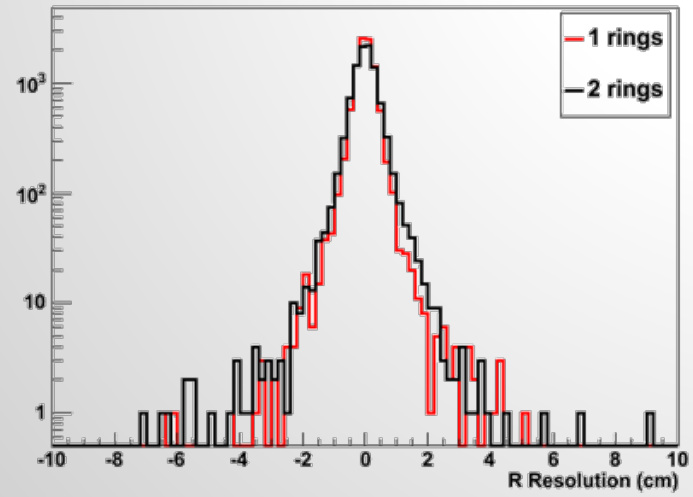
2→ (8.42,4.59) R=12.6

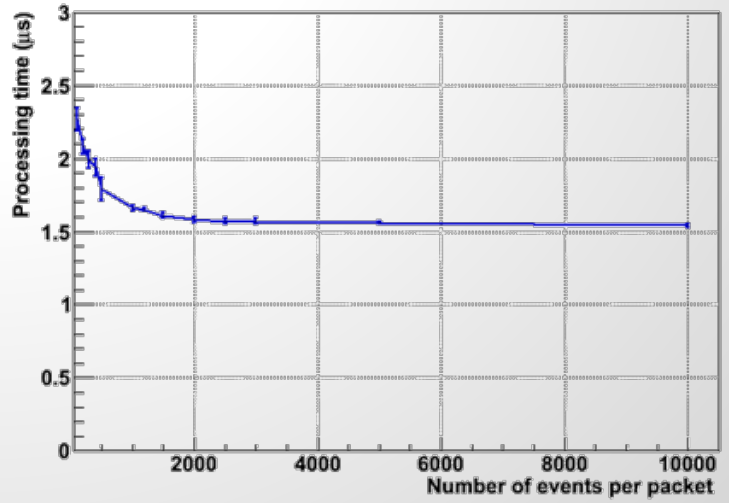
- The fitted position of the two rings is:

1→ (7.29, 6.57) R=11.6

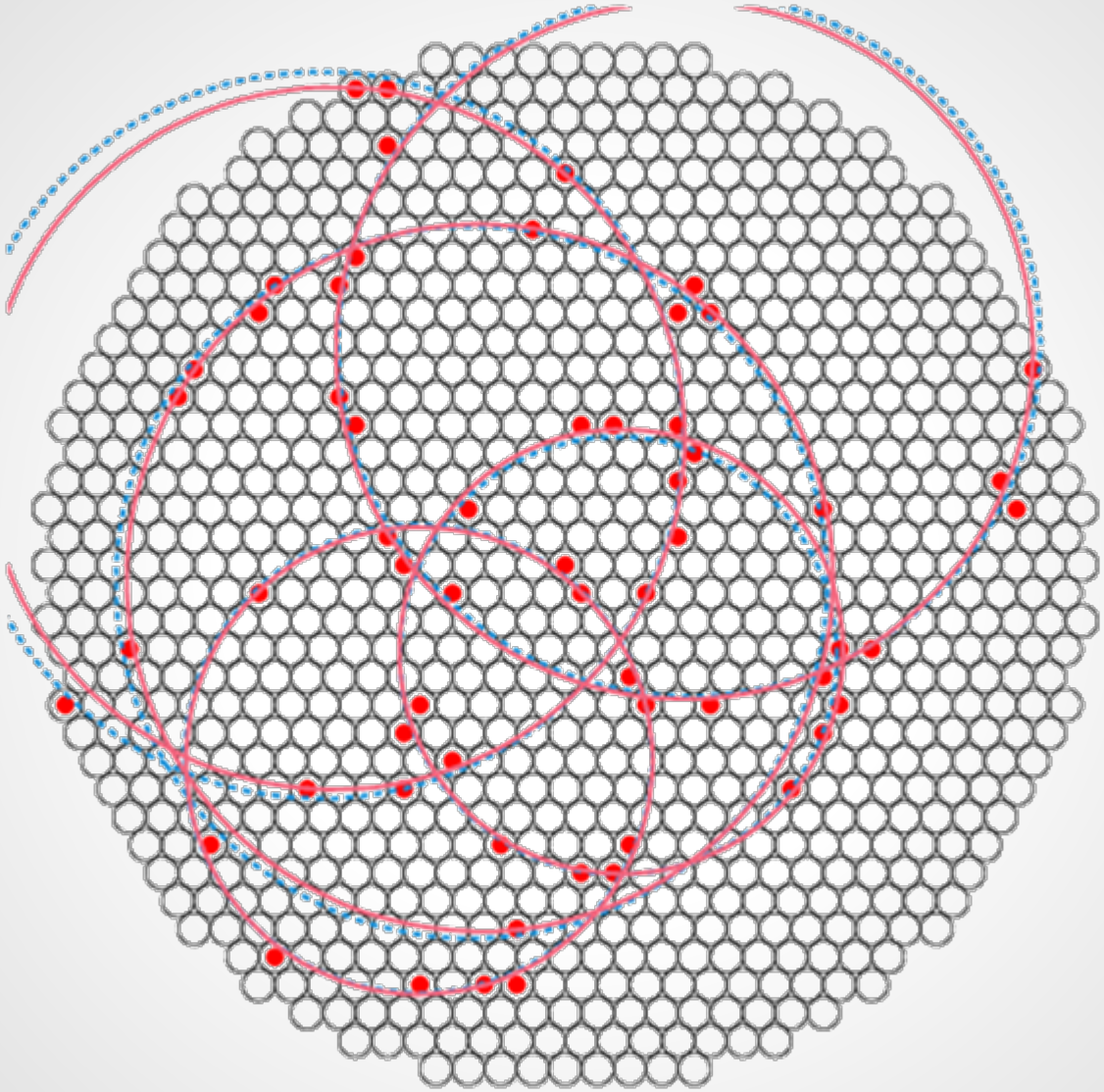
2→ (8.44,4.34) R=12.26

- Fitting time on Tesla C1060: 1.5 us/event

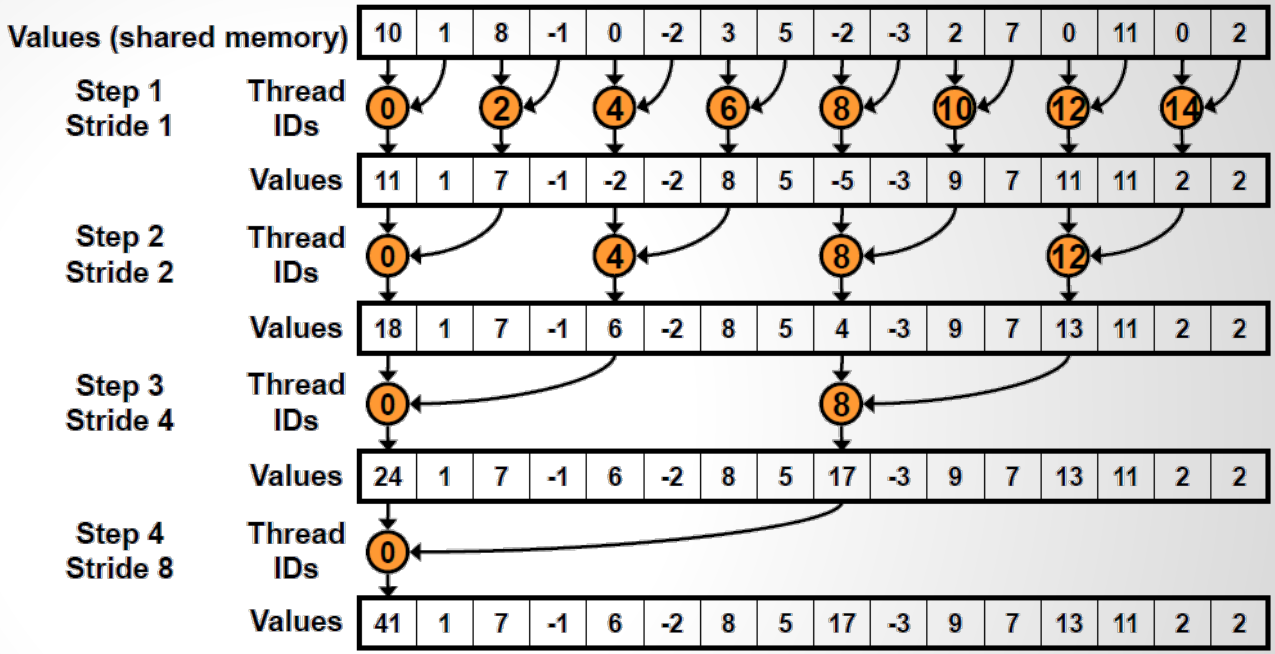




Almagest for many rings



Reduction - Interleaved addressing




```
// do reduction in shared mem
for (unsigned int s=1; s < blockDim.x; s *= 2) {
    if (tid % (2*s) == 0) {
        sdata[tid] += sdata[tid + s];
    }
    __syncthreads();
}
```

Problem: highly divergent warps are very inefficient, and % operator is very slow

Brent's theorem

Brent's theorem specifies for a sequential algorithm with:

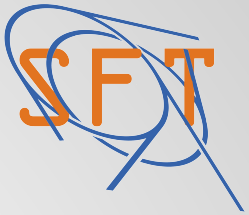
- *t* time steps
- a total of *m* operations,

that a run time *T* is definitely possible on a shared memory machine with *p* processors.

$$T = t + \frac{(m-t)}{p}$$

Note: The Brent's theorem does not take into account the possibility of additional improvements due to level shift in the original schedule.

Brent's Theorem



Suppose we want to sum an array.

We could step through the array, adding each value in turn to an accumulator. In pseudocode:

```
for (i=0;i < length(a); i++) { sum = sum + a(i); }
```

Using this algorithm, each add operation depends on the result of the previous one, forming a chain of length *n*, thus:

$$t = n$$

There are *n* operations, so

$$m = n$$

No matter how many processors are available, this algorithm will take time n.

Brent's Theorem

Now consider the adding the array recursively:

`sum(a) = ((A0 + A1) + (A2 + A3)) + ((A4 + A5) + (A6 + A7))...`

We can add **A_2** to **A_3** without needing to know what **A_0 + A_1** is.

To calculate the sum from **0-3**, we only need the results of **A_0 + A_1**, and **A_2 + A_3**.

Instead of one chain of length 4, we have many chains of length 2.

For an array of length *n*, the longest chain(s) will be of length log (n).

t = log(n). m = n (as before).

$$T = log(n) + \frac{n - log(n)}{p}$$

Brent's Theorem

$$T = log(n) + \frac{n - log(n)}{p}$$

This tells us many useful things about the algorithm:

- No matter how many processors used, there can be no implementation of this algorithm that runs faster than O(log(n)).
- If we have n processors, the algorithm can be implemented in log(n) time
- If we have log(n) processors, the algorithm can be implemented in 2log(n) time (asymptotically this is the same as log(n) time)
- If we have one processor, the algorithm can be implemented in n time.