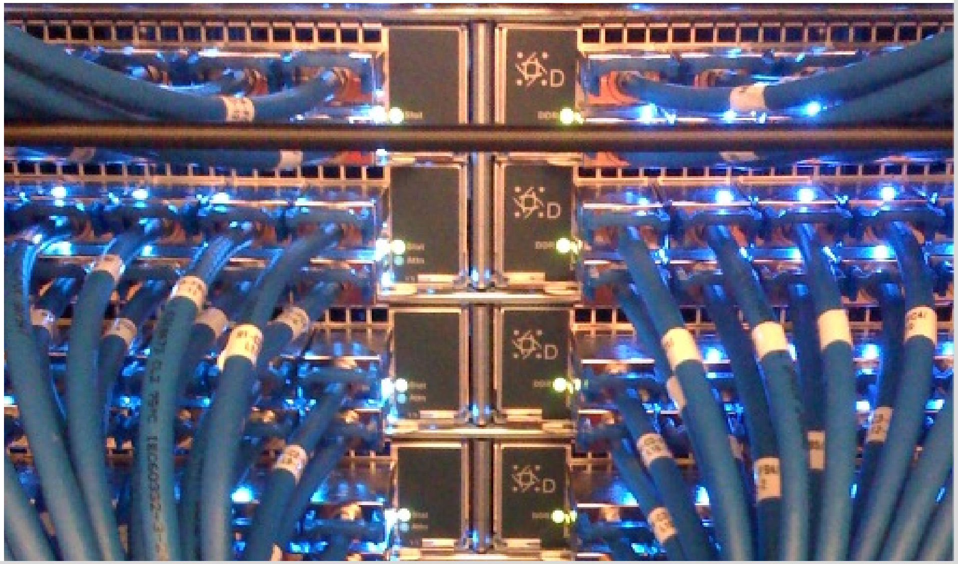
Distributed GPU-computing for scientific applications

Timo Dritschler

IPE, Institute for data processing and electronics



"REFUSING TO ASK FOR HELP WHEN YOU NEED IT IS REFUSING SOMEONE THE CHANCE TO BE HELPFUL."

RIC OCASEK

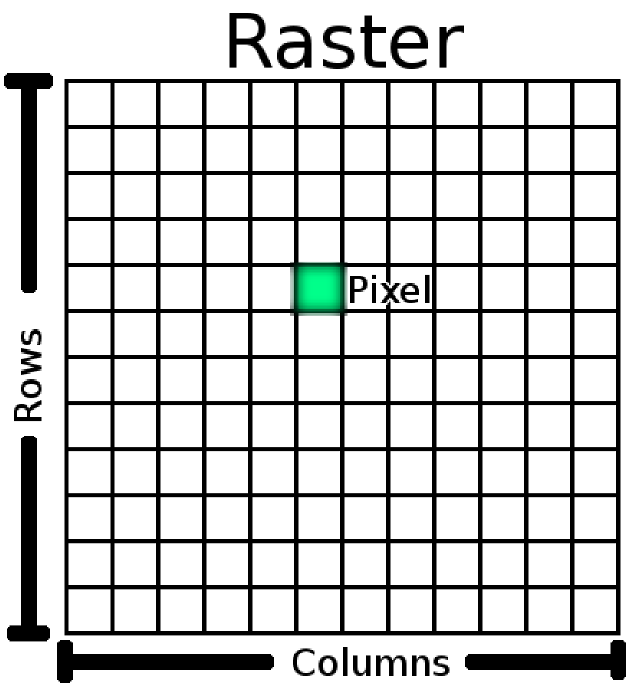
What are GPUs?

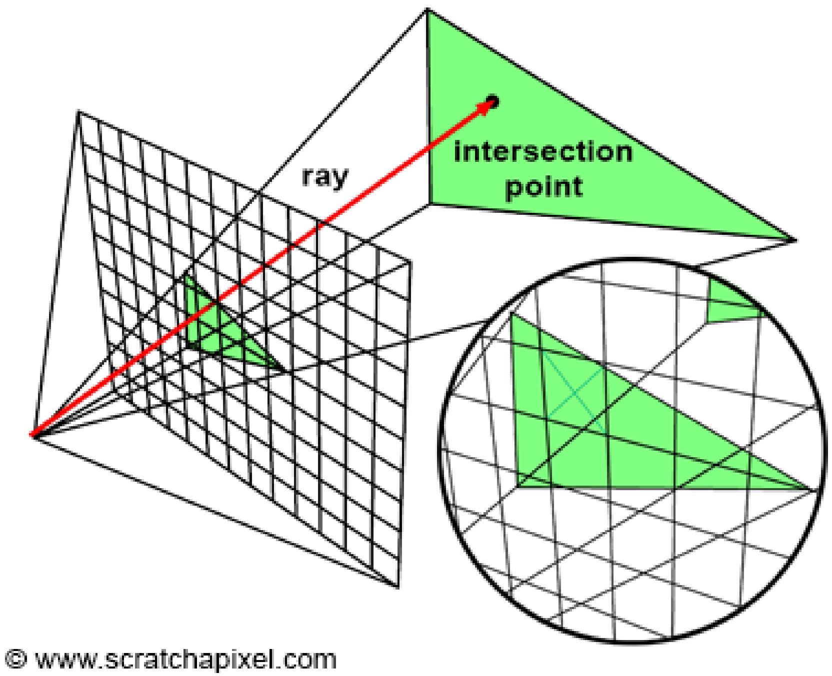
What are GPUs?

- Graphics-Processing-Units
- Co-processors for the computers CPU
- Highly-Parallel computing architectures
- Optimized for numbercrunching and computing throughput
- Programmable!!

What are GPUs?

- Co-processors (mainly) for computer graphics
- Aid the CPU in rasterization computations



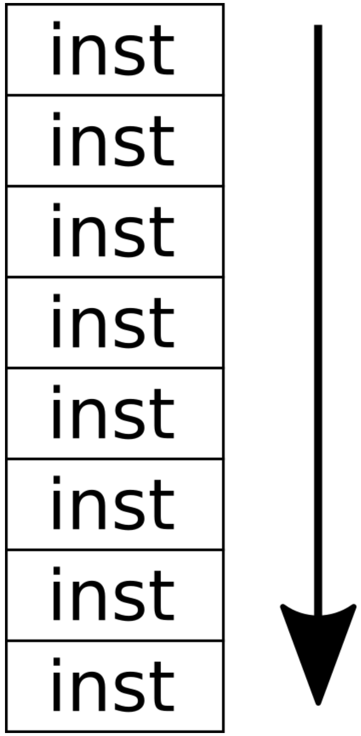


What are GPUs?

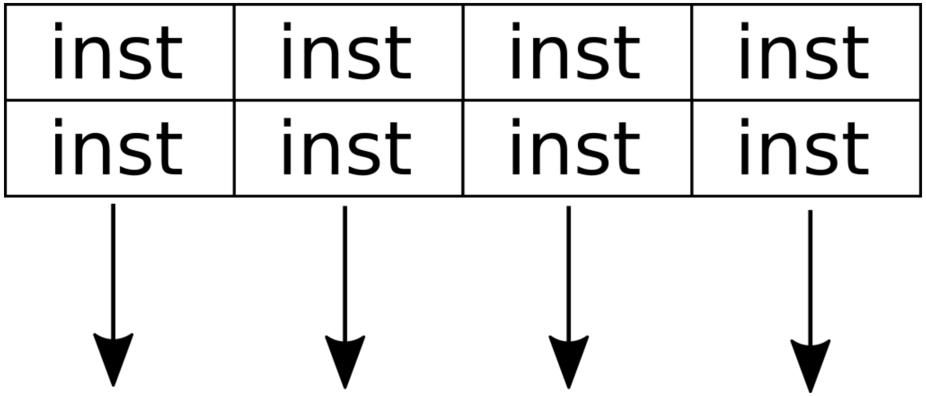
Rasterizing complex 3D geometries onto the 2D viewing-plane takes billions of computing operations!

CPU vs GPU

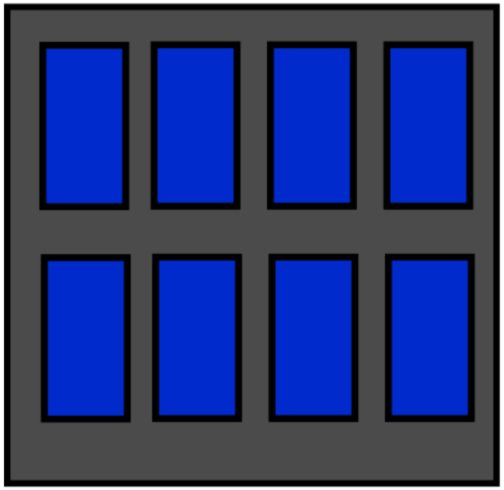
CPU:
Optimized to compute a single stream of instructions as fast as possible



GPU:
Optimized to compute as many instructions in parallel as possible

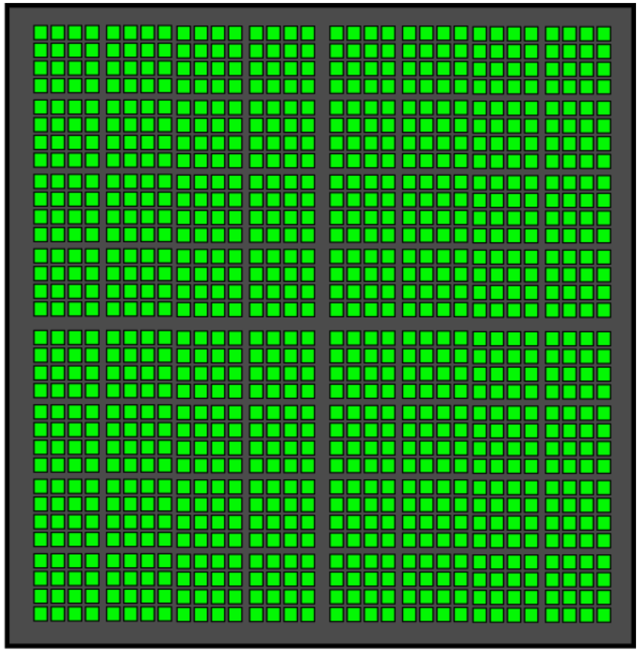


CPU vs GPU



CPU

A few highly-optimized cores

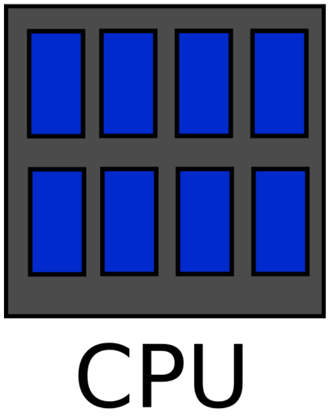


GPU

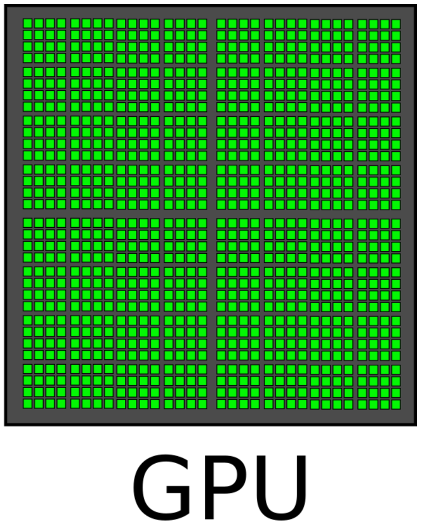
Thousands of simpler cores

Examples

- Good for CPU:
- Calculate the n-th number of a series
- “branching” computation (if-then-else)

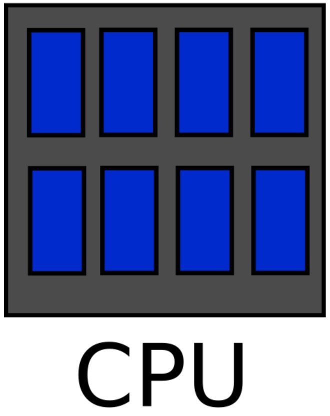


- Good for GPU:
- Calculate n different values of a function
- prime number check

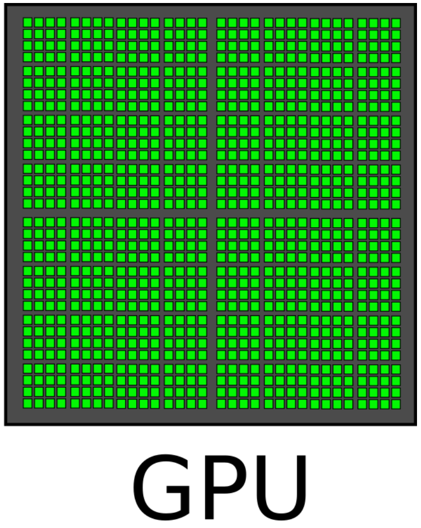


Examples

- Good for CPU:
- Calculate the n-th number of a series
- “branching” computation (if-then-else)

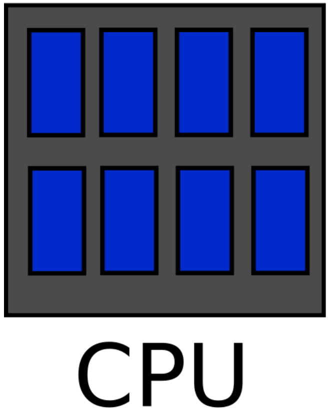


- Good for GPU:
- Calculate n different values of a function
- prime number check
- Bitcoin mining ☺

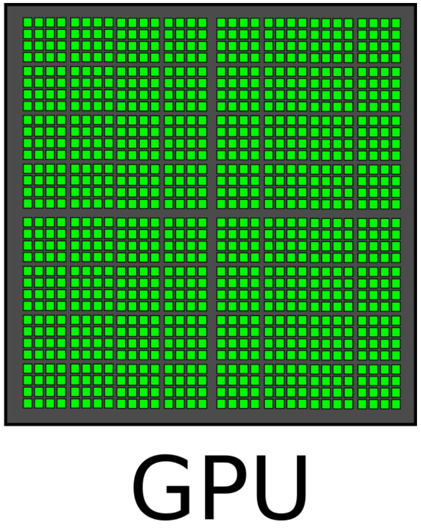


Examples

- Good for CPU:
- Calculate the n-th number of a series
- “branching” computation (if-then-else)



- Good for GPU:
- Calculate n different values of a function
- prime number check
- ~~Bitcoin mining~~ 😵 (ASICs are more energy-efficent)



GPU APIs

- GPU vendors have noticed computing potential of GPUs
- APIs aim to provide easy access for general-purpose computing

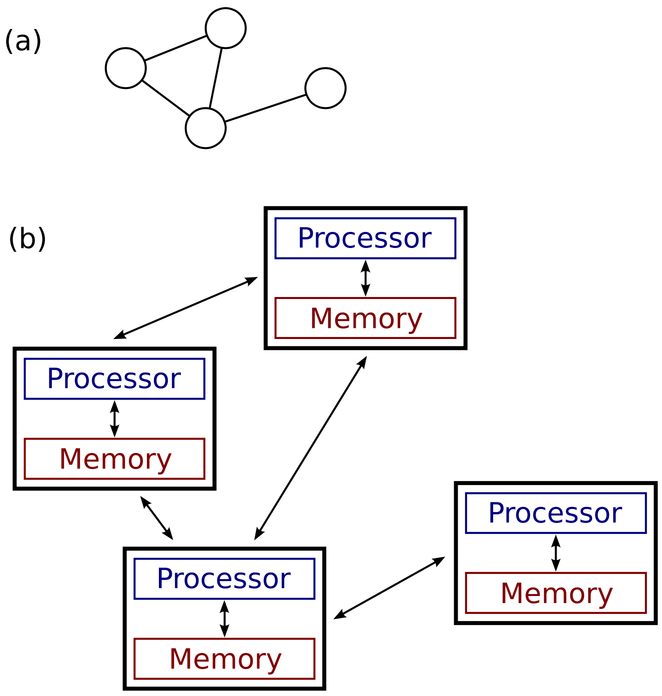
Main Attributes:
- Ease of use
- High computing performance on NVIDIA GPUs only

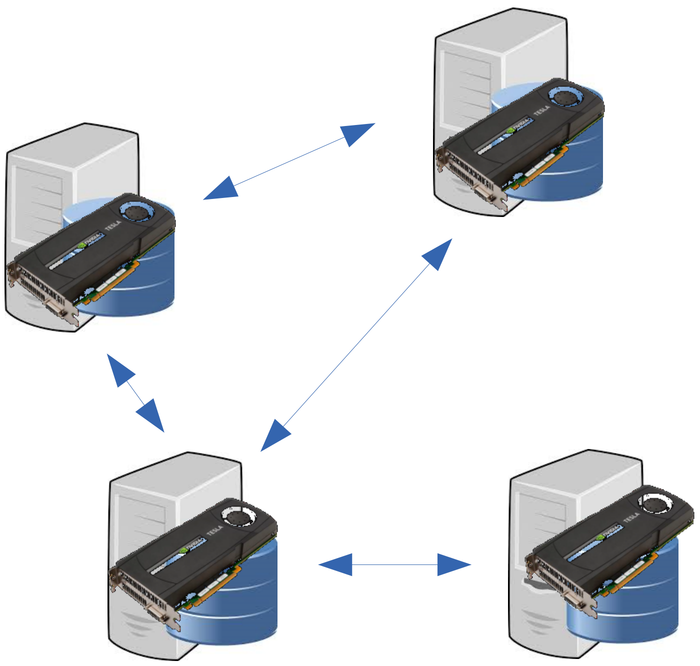
Main Attributes:
- Much more generalized
- Targets a broad band of different accelerators (not only GPUs!)

Distributed Computing

Distributed Computing

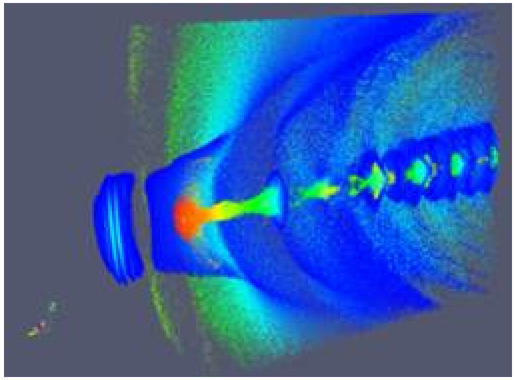
A distributed system is a model in which components located on networked computers communicate and coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal.

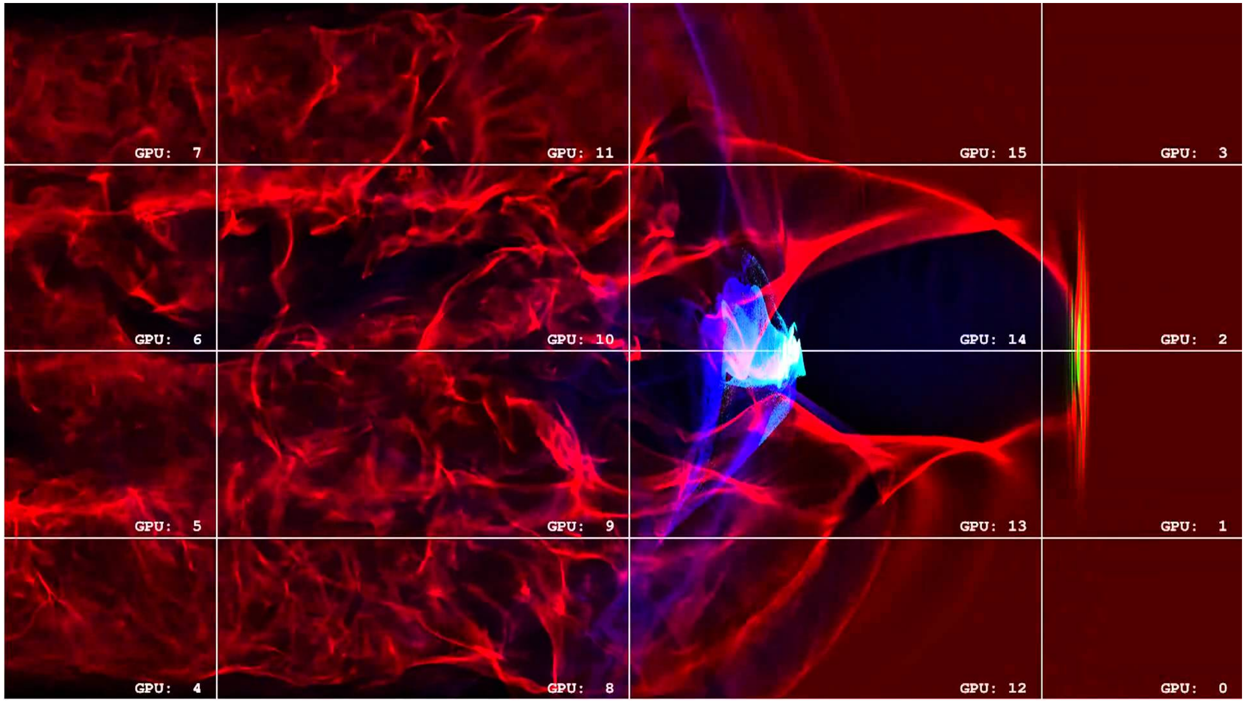




Example: PIConGPU

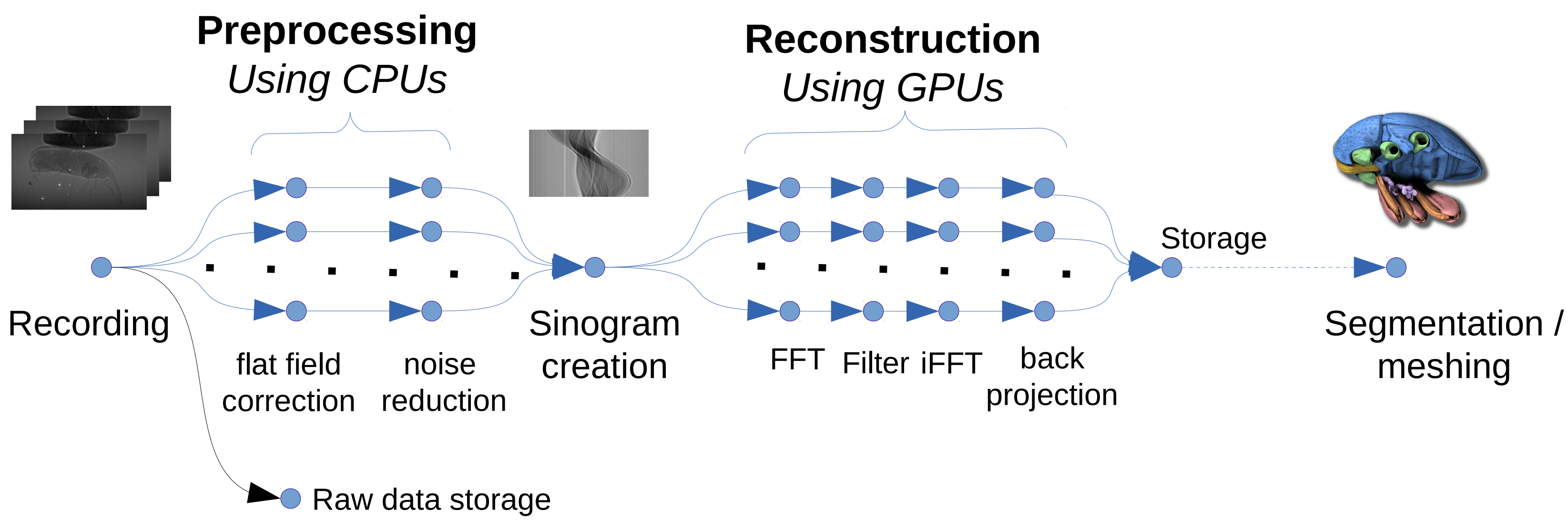
- Particle-In-Cell, mainly used for laser wakefield acceleration simulation
- Runs massivle parallel on GPUs





PIConGPU is developed byt the HZDR (Hemholtz-Zentrum Dresden-Rossendorf)

Example: KIT UFO Framework



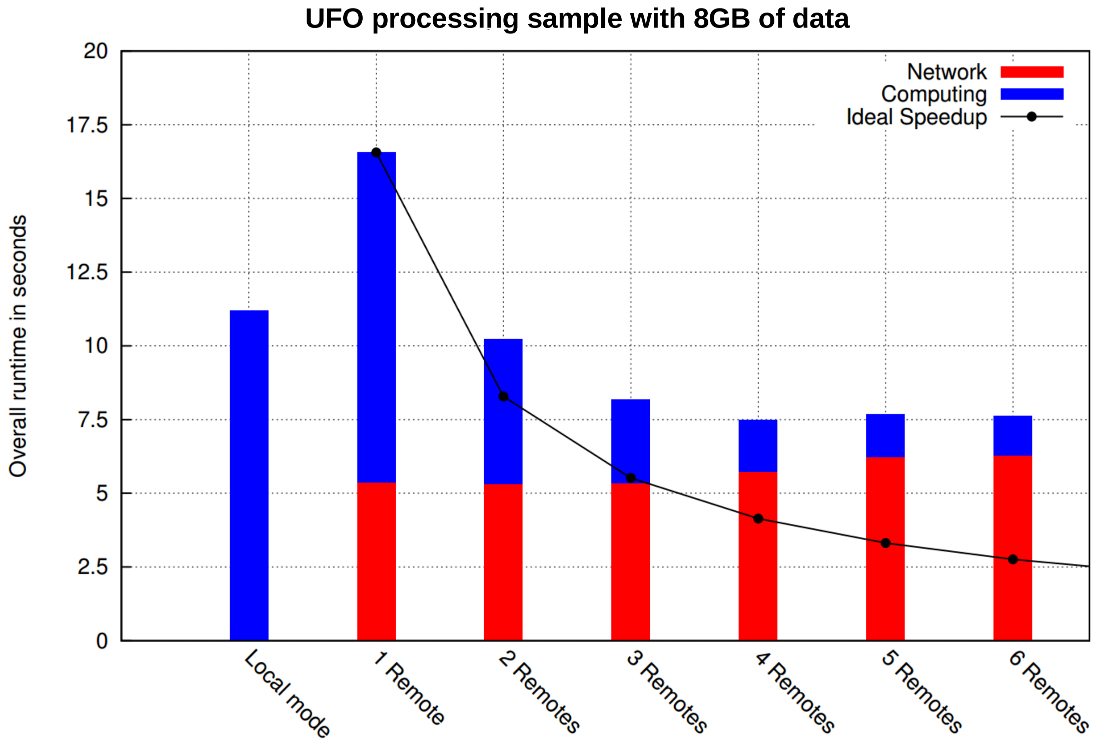
KIT UFO Framework

- Distributed computing framework
- Plug-in based and extensible algorithms
- GPU-enabled
- Automatically distributes work across the network
- Simple markup-file-based configuration

Available on Github:
https://github.com/ufo-kit

KIT UFO framework scalability

- Early test with MPI based communication
- Computing part scaled nicely
- However, network communication did not scale

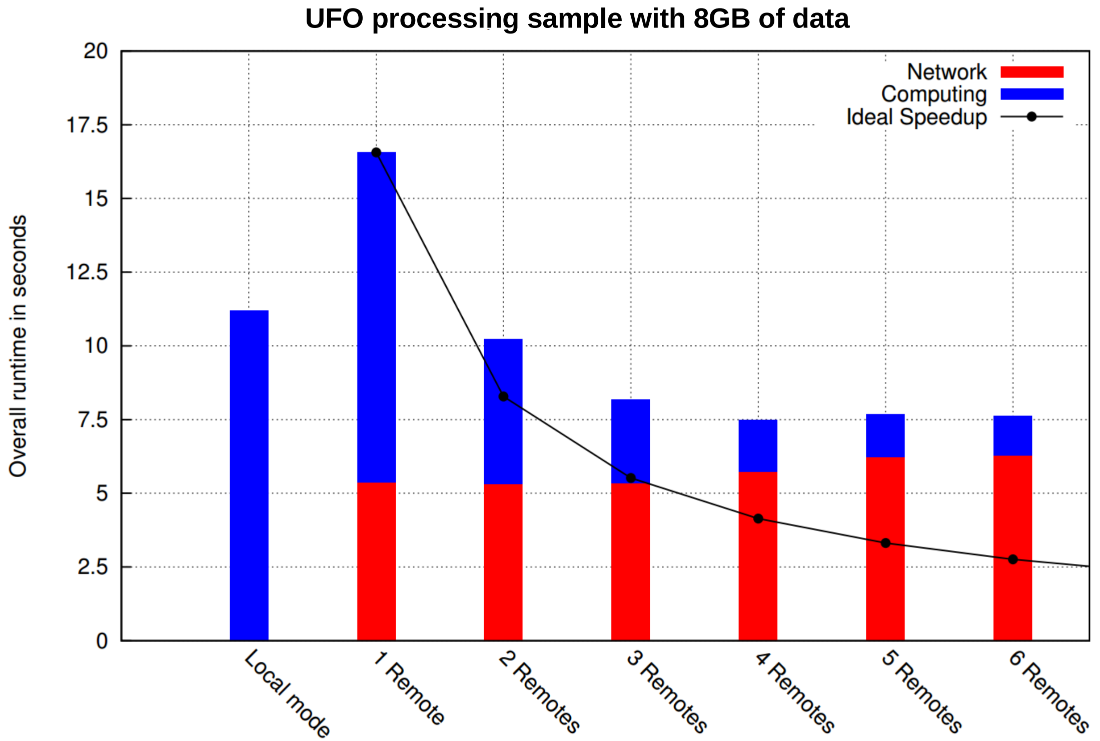


Graphic by Timo Dörr, *'Concepts and evaluation of communication patterns for digital image processing in heterogeneous distributed systems',* 2014

KIT UFO framework scalability

Karlsruher Institut für Technologie

- Early test with MPI based communication
- Computing part scaled nicely
- However, network communication did not scale

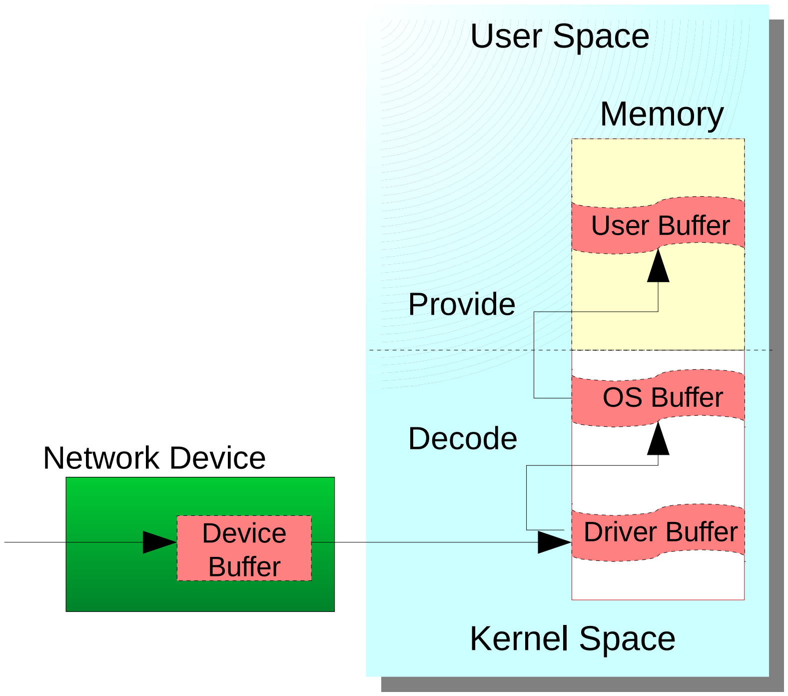


Graphic by Timo Dörr, *'Concepts and evaluation of communication patterns for digital image processing in heterogeneous distributed systems'*, 2014

Just throwing more computers at the problem will not always help!

Key Technology RDMA (Remote Direct Memory Access)

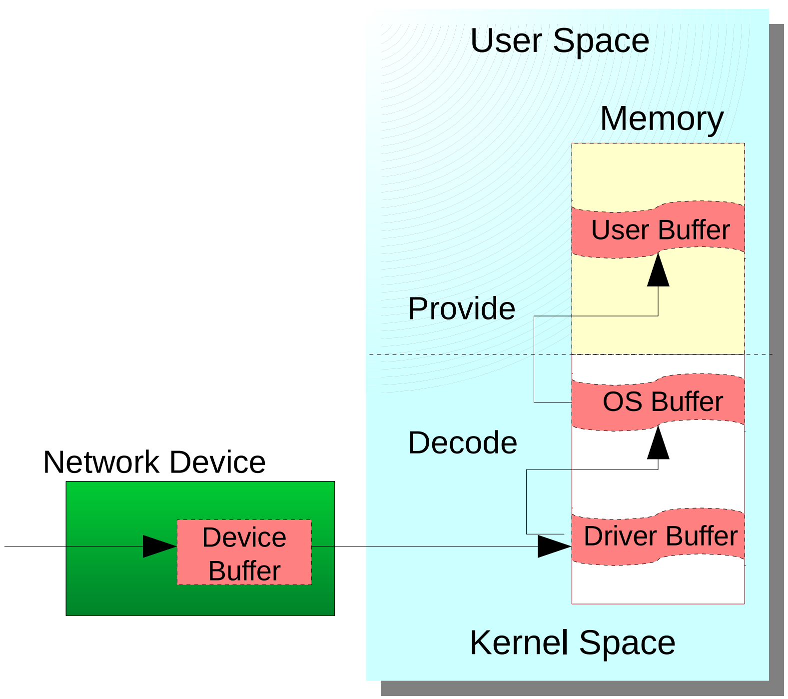
Classical network transfer



- Up to four copies
- High latency

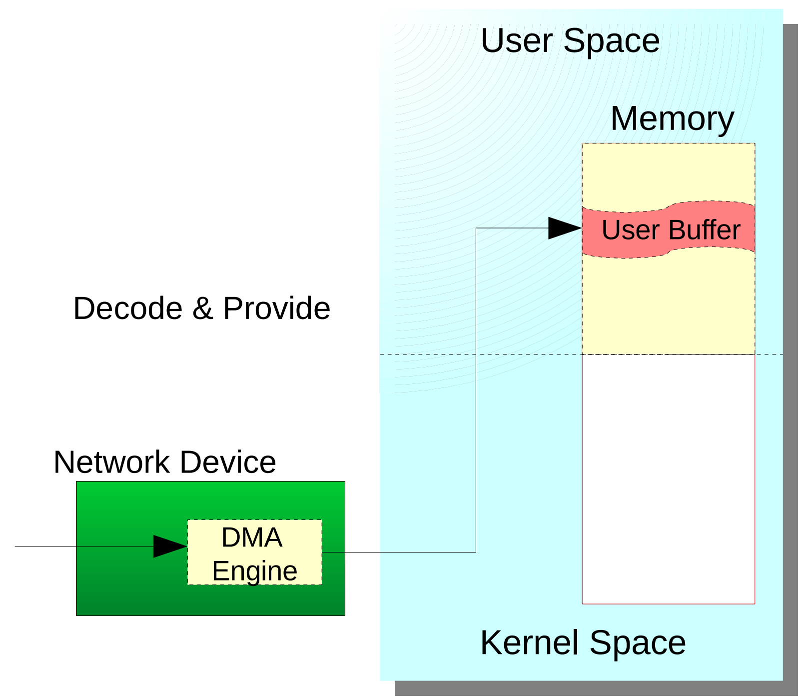
Key Technology RDMA (Remote Direct Memory Access)

Classical network transfer



- Up to four copies
- High latency

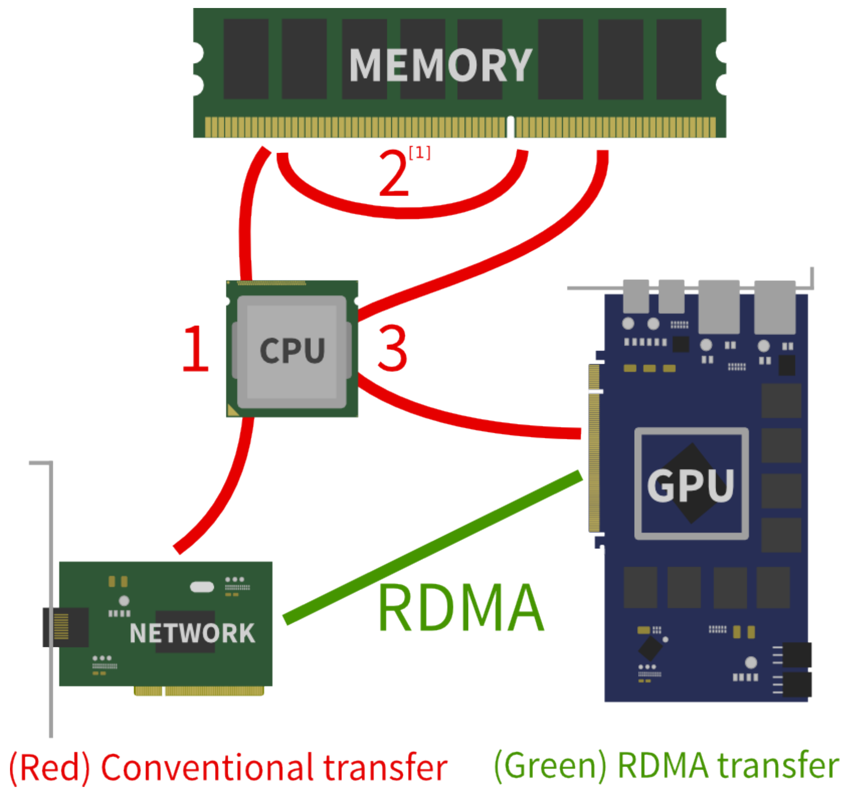
RDMA (Remote Direct Memory Access)



- Only one implicit 'copy' from DMA engine
- Low latency

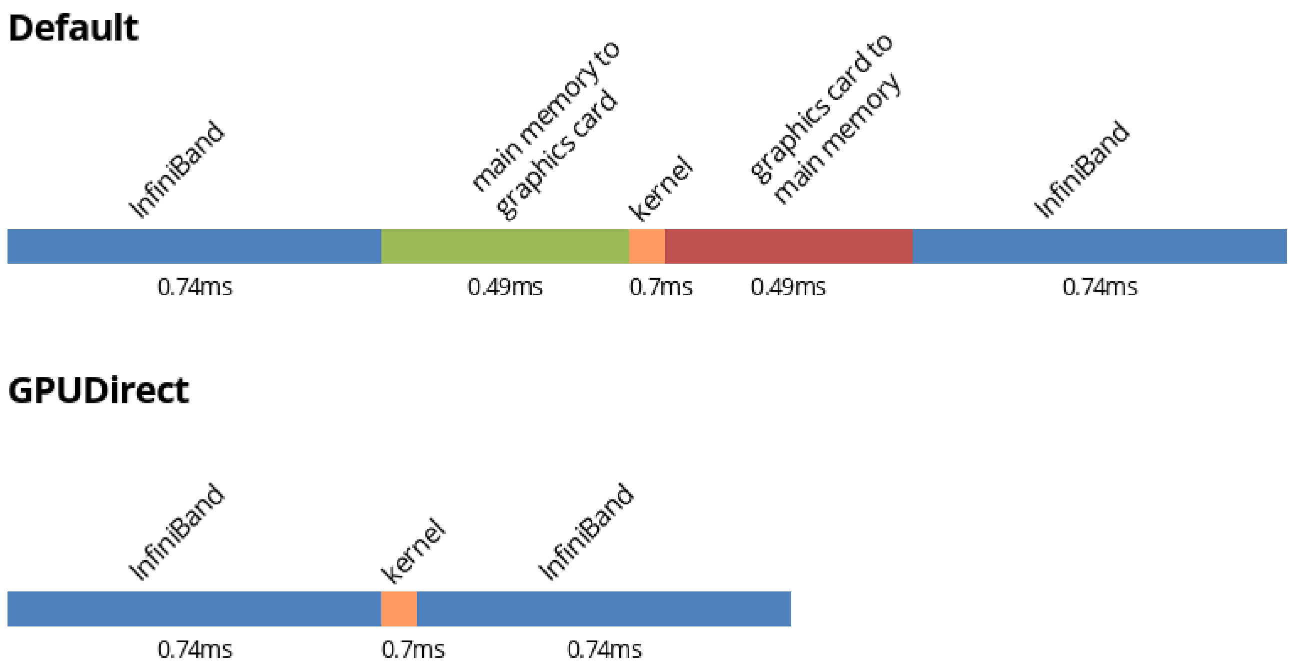
Key technology: GPU RDMA

When doing conventional data transfer that is meant for GPU computation, at least two copy operations are required. GPU RDMA circumvents this problem.



GPUDirect performance

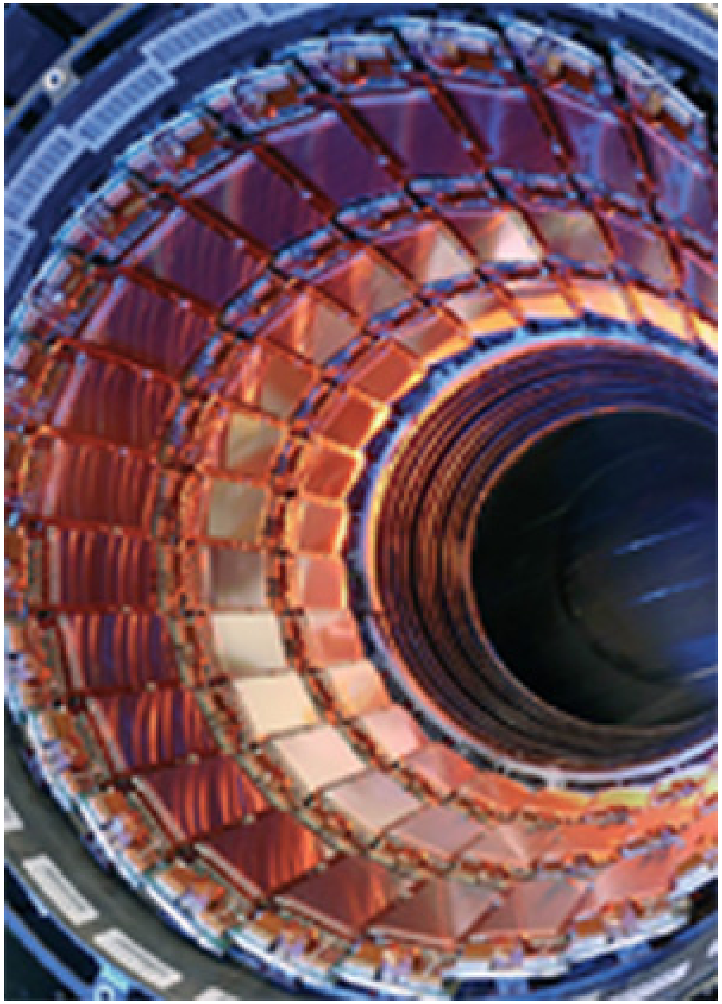
BENCHMARK PROXY



Exemplary latency for a block size of 4MB

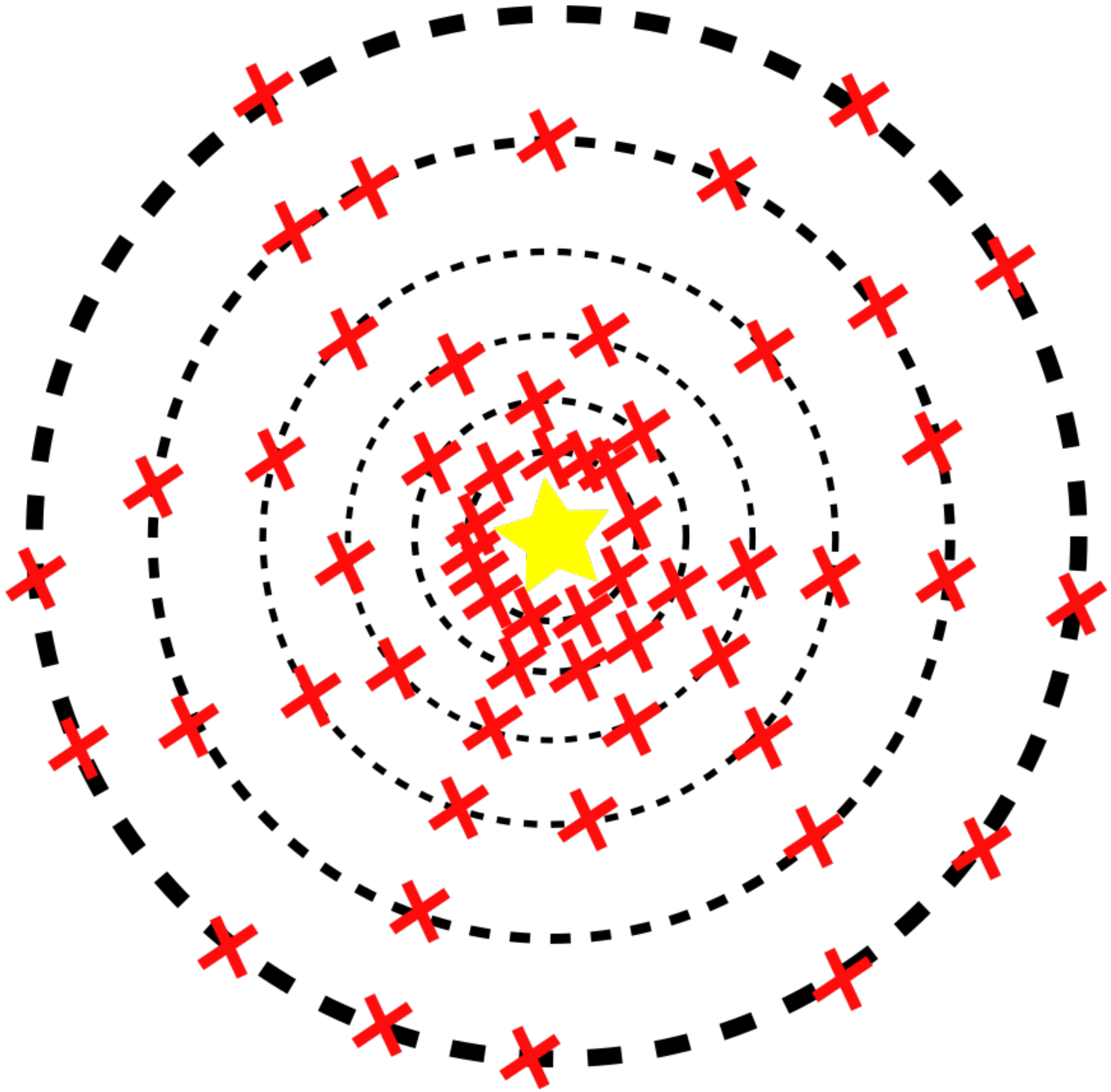
Putting it all together

Application: CMS Track Trigger on GPUs

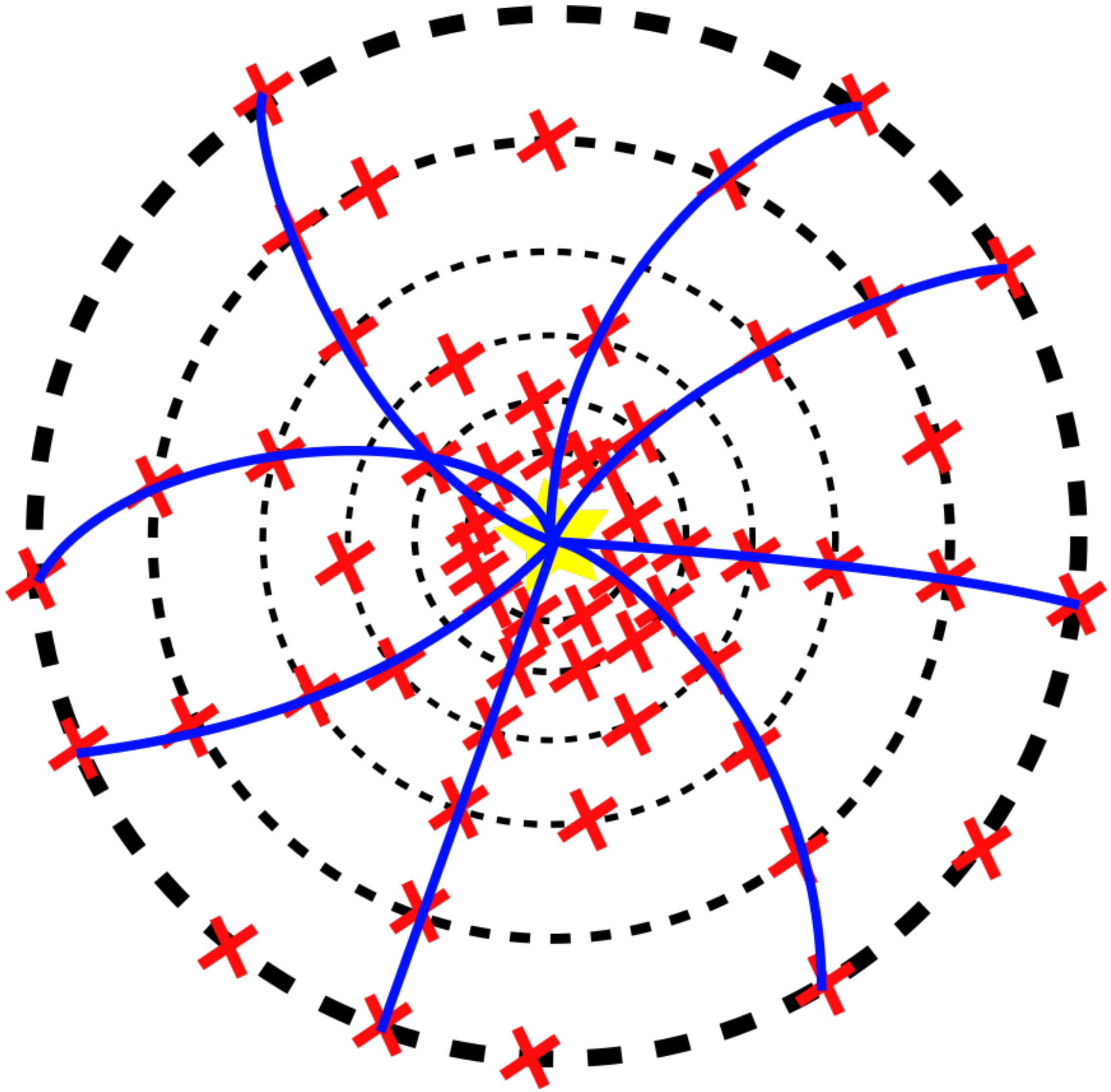


- Collisions every 25ns (approx. **100TBit/s**)
- Pipeline gives us **6µs** for Track-Finding
- Detector is split into sections at approx. **150Gbit/s** data rate

CMS Track Finding

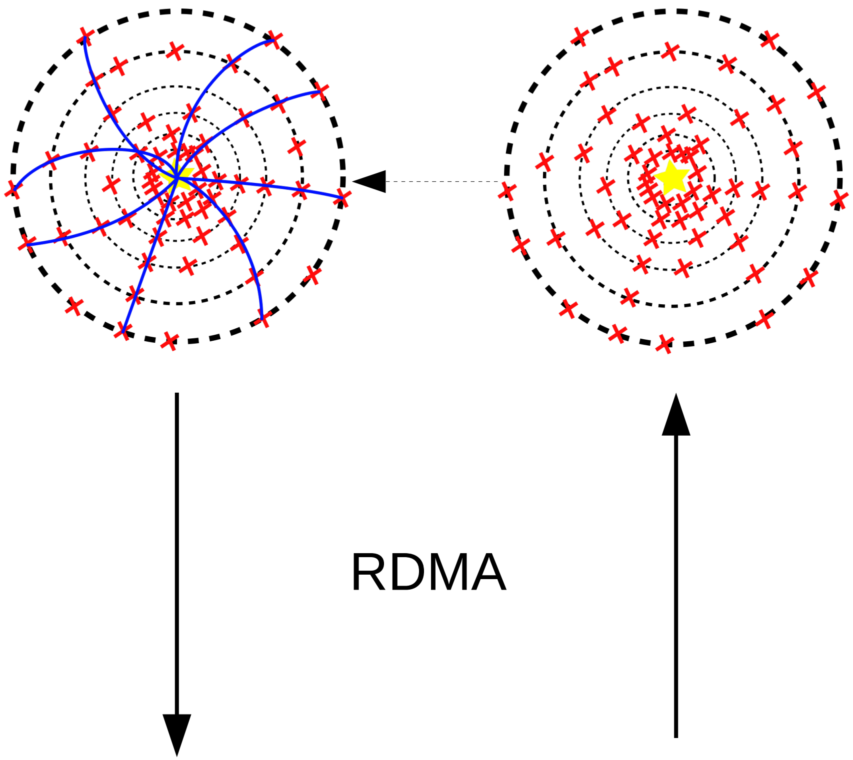


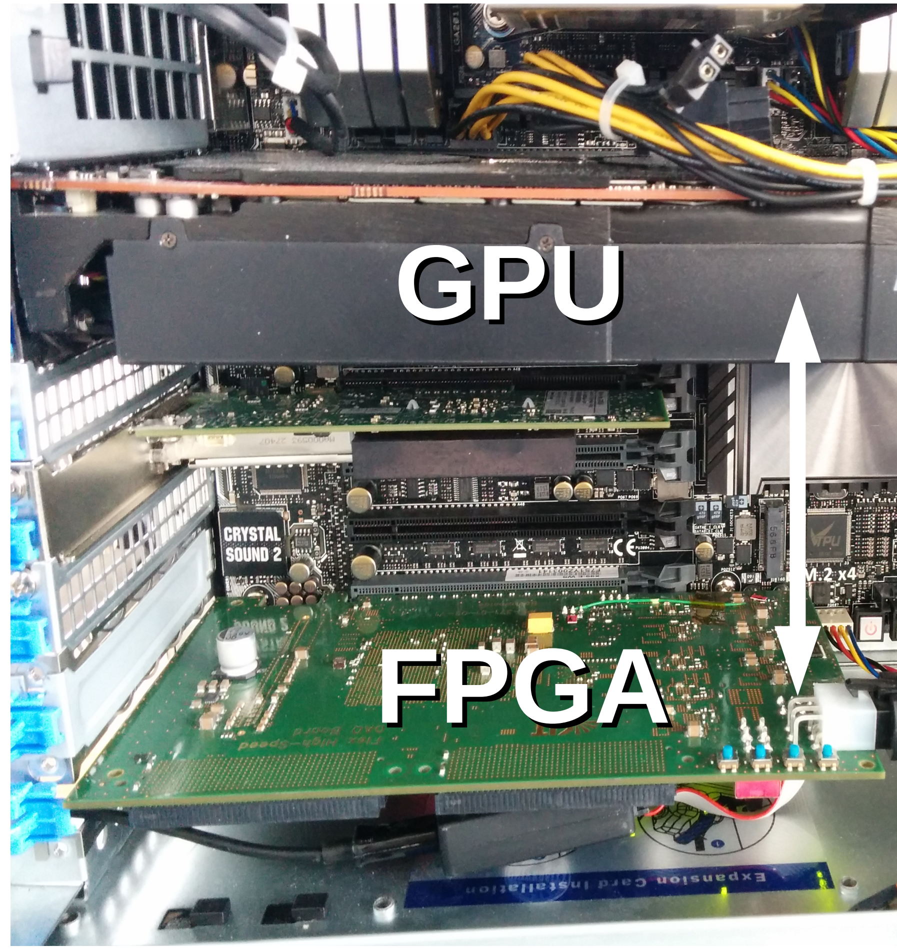
CMS Track Finding



Karlsruher Institut für Technologie

GPU ↔ FPGA using RDMA

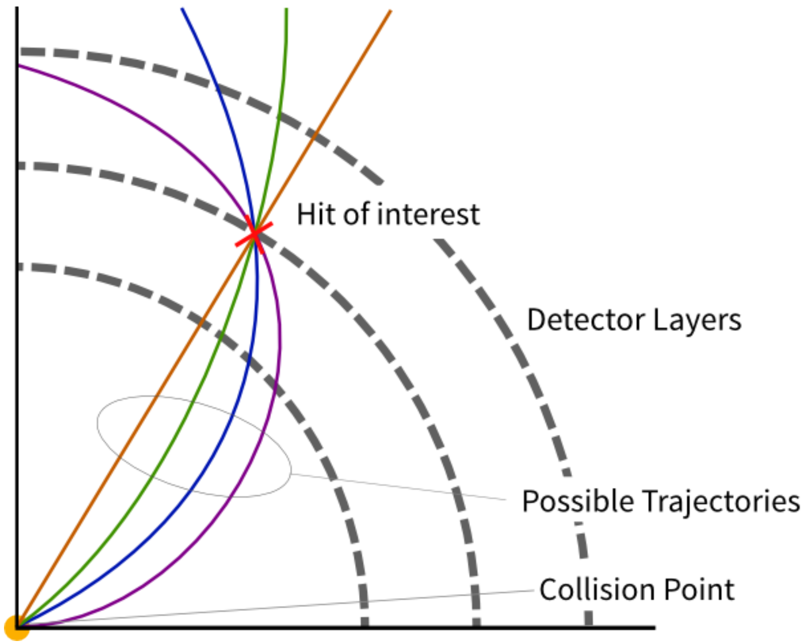


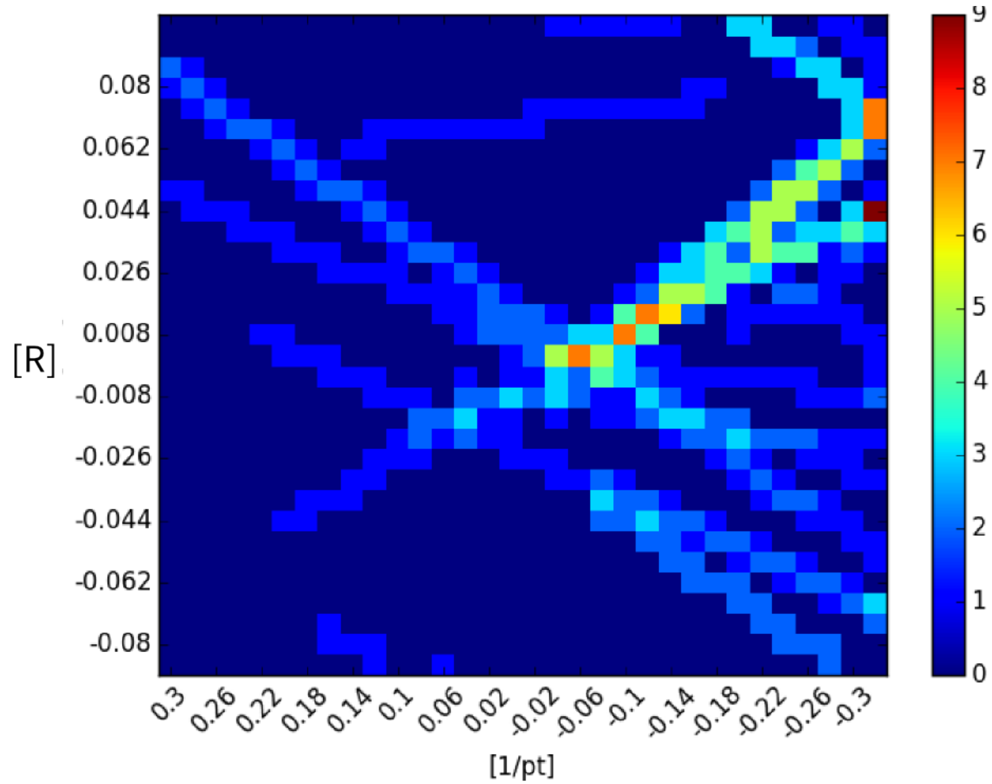


4 kB data transfer (one sector):
Conventional: 120 µs
RDMA: **2 µs!**

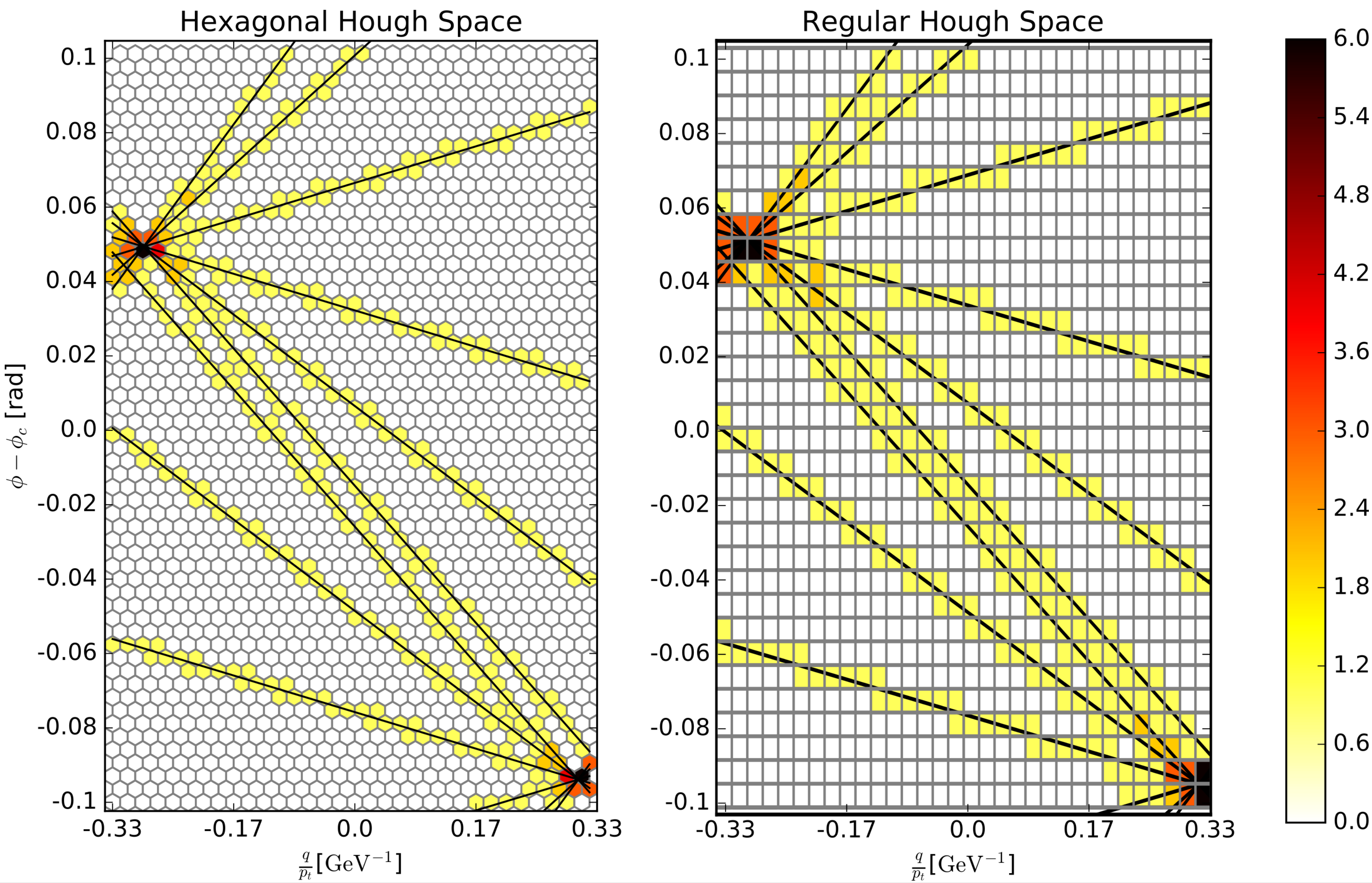
Hough Transformation

- Transform possible track-parameters into lines (in Hough-space)
- In plpaces where multiple lines intersect, we have possible candidates for a track!

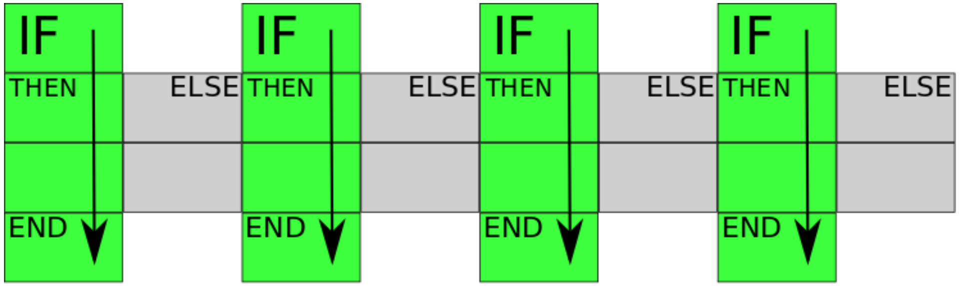




Hough Transformation

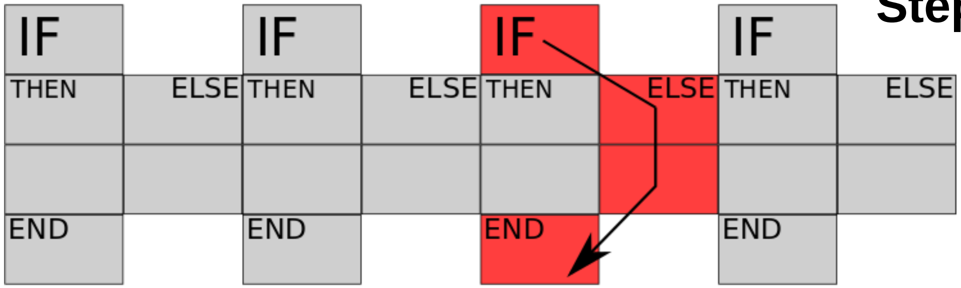


Things to keep in mind: Algorithmic Branching

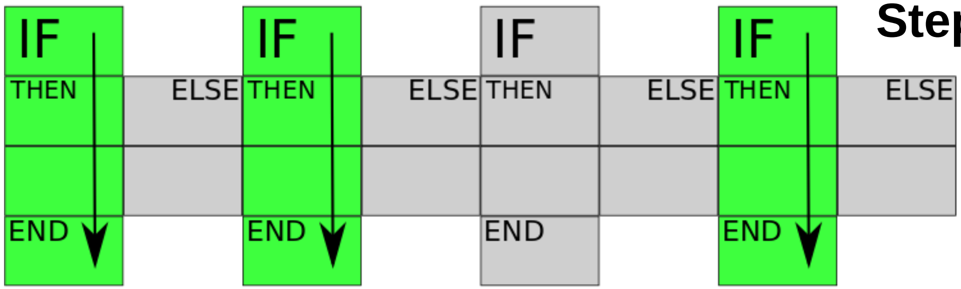


No branching
All threads perform the same operation.

Step 1

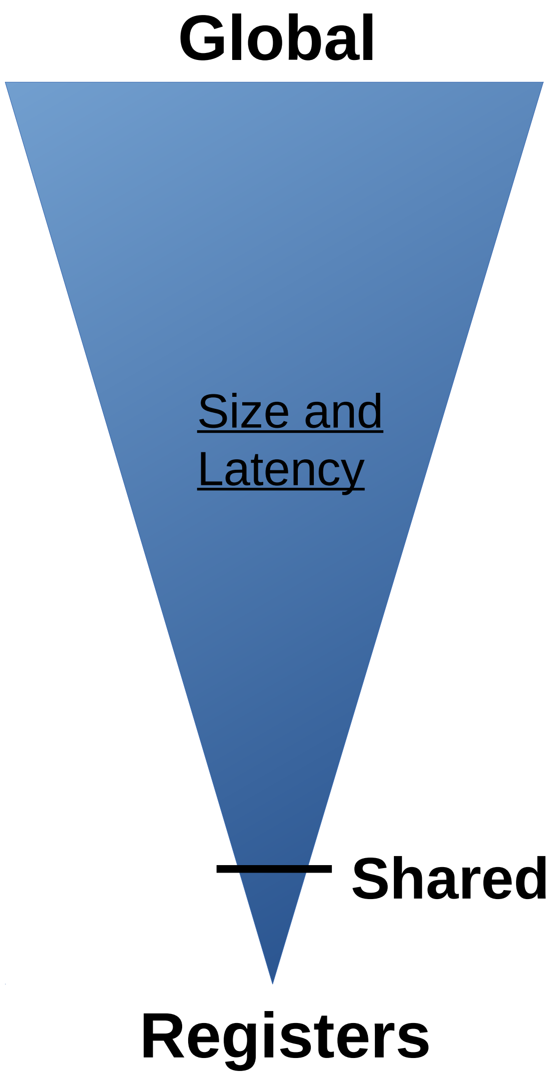


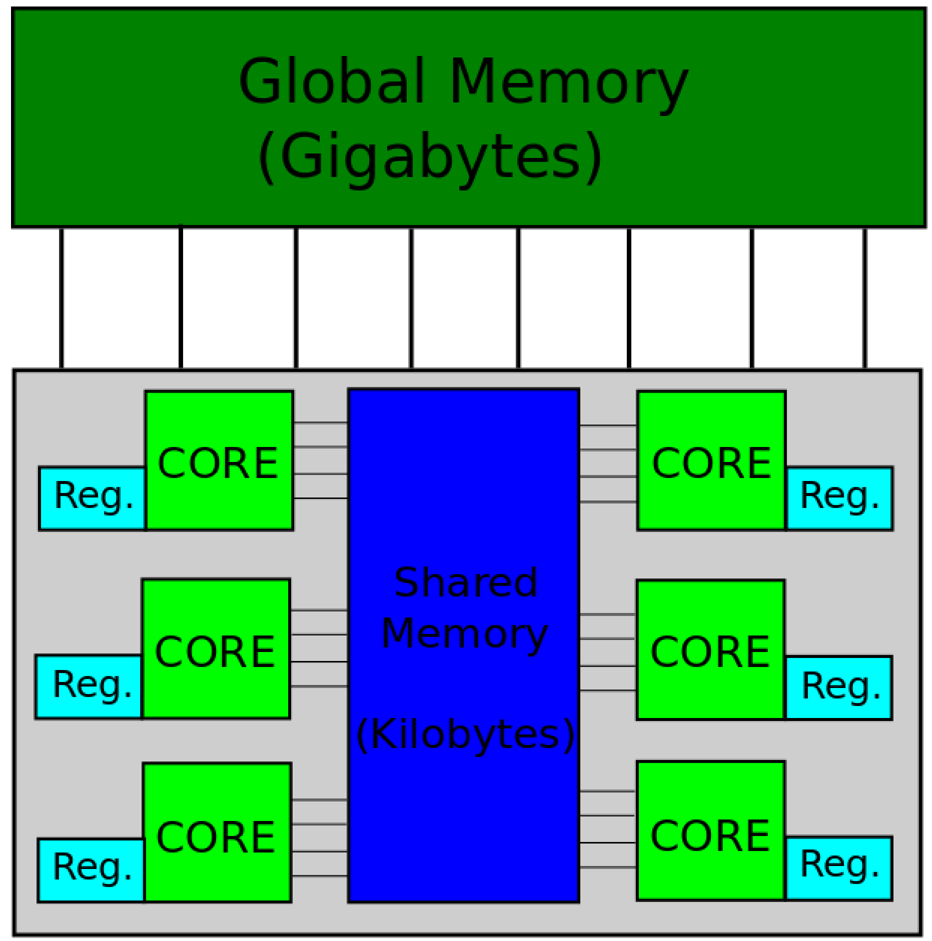
Step 2



At least one thread branches
The two different branches are computed in sequence. All threads of the opposing branches sleep during execution.

Things to keep in mind: Memory layers





Registers: Several Bytes

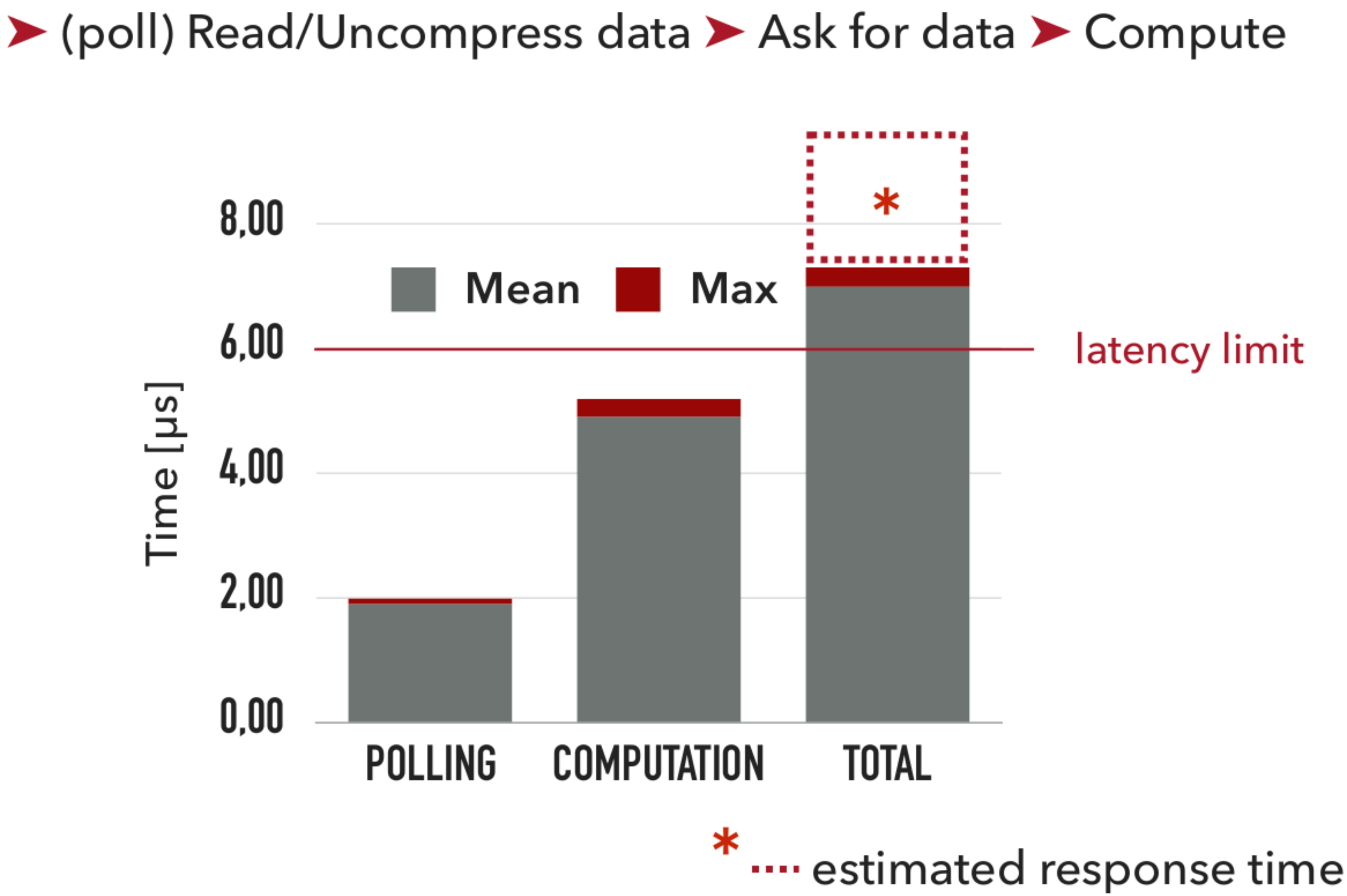
Optimization Steps

- Naive (unoptimized) approach

- After Branching optimization

- After Memory optimization

Current performance



Conclusion

- GPUs are very well suited for parallel number crunching
- Programming interfaces make it (relatively) easy to utilize GPUs computing power
- Modern networking techniques make distributed GPU computing scalable
- Opztimization can be tricky. There is no “One size fits all”-solution.

  Ask your friendly Computer-Scientists. They are happy to help!