GPU-based high-performance computing for radiotherapy applications

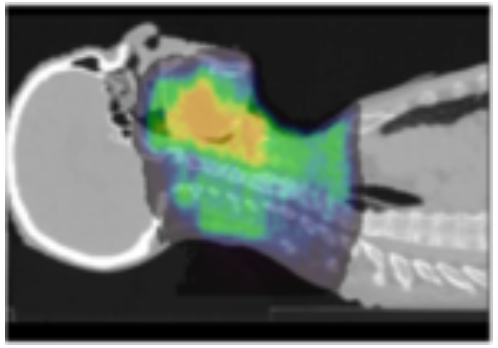
Julien Bert, PhD

CHRU de Brest
LaTIM – INSERM UMR1101

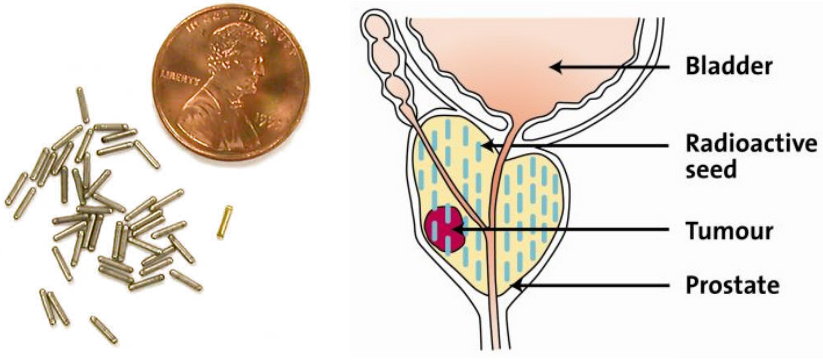
Radiotherapy

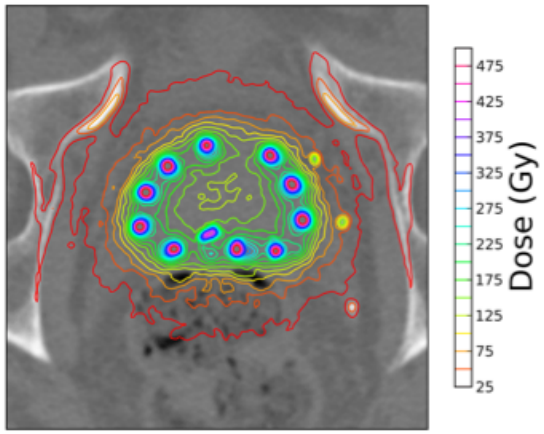
Irradiation of the tumor:

- Maximum dose to the tumor
- Healthy surrounding organs spared



External Beam Radiotherapy





Intra-Operative Radiotherapy

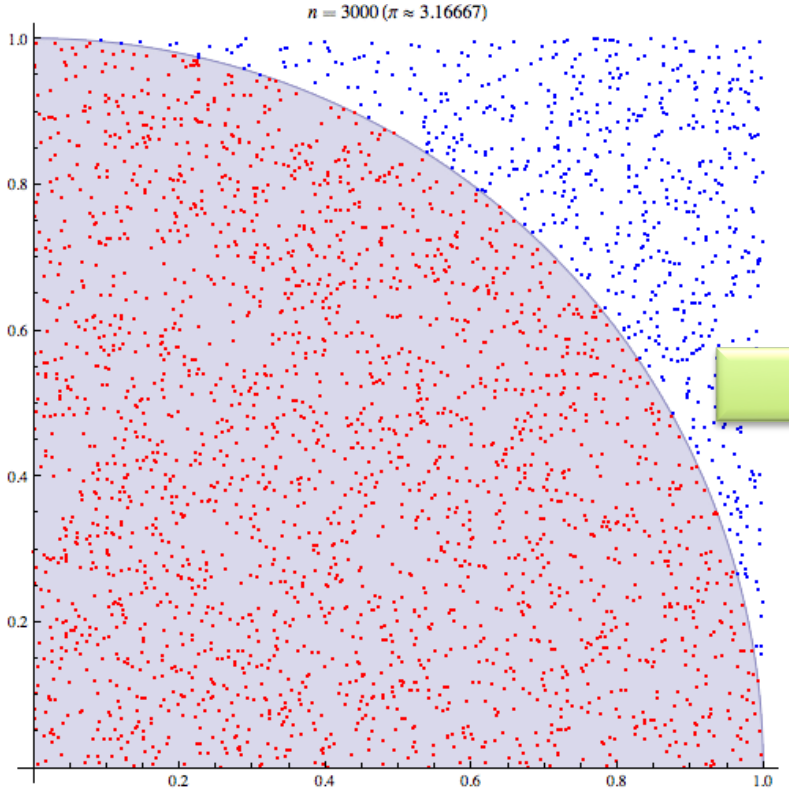
Aims:

- Predict the dose
- Personalized planning
- Ensure quality and safety

Accurate and realistic numerical simulation in radiation physics

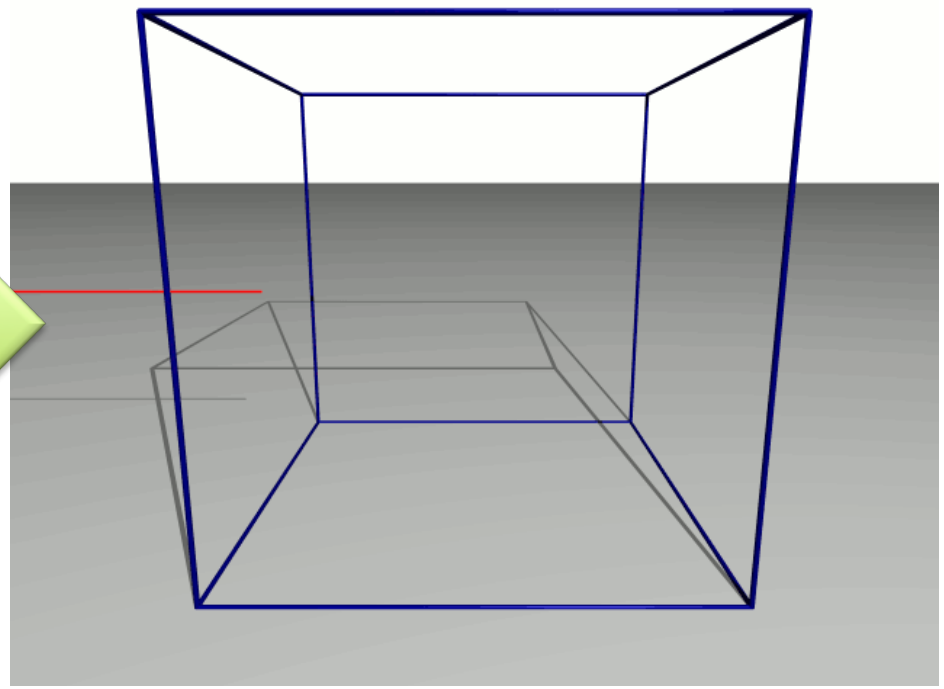
Monte Carlo method

Estimated the value of π



Uniformly scatter some objects of uniform size (rice or sand) over the square

Monte Carlo Simulation



Random sample must follow the laws of physics

Simulate particle interactions with matter

Monte Carlo Simulation

Monte Carlo simulation

Very computationally demanding
limits their applicability in both research and clinical environment applications

Computer cluster
financial burden and availability issues

Cost

Space

- Patient
- Workflow
- Workspace
- Intervention cost

?

h-GATE (2010)

Computational power

4 graphics cards **x2** GPUs[1]

= eq.

249 processors[2] with **x6** cores (**1494** cores)

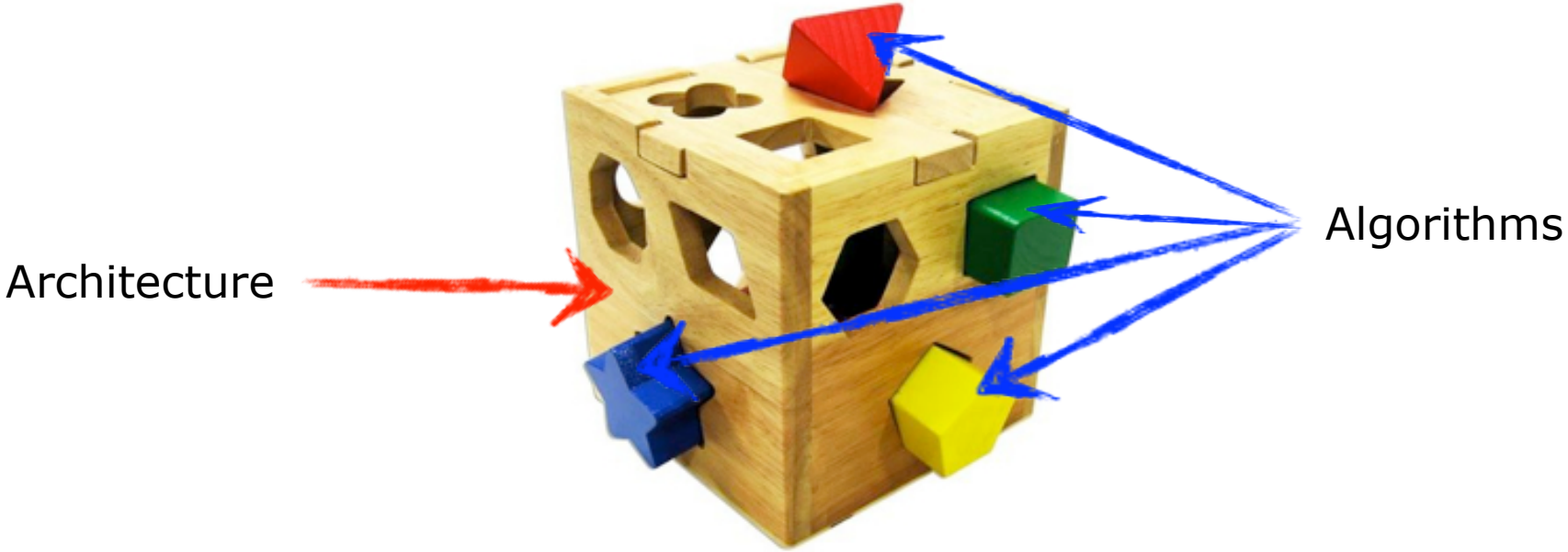
1 GPU core: **4** TFLOPS
1 CPU core: **20** GFL0PS

High computational power at a reduce cost (1/20)

[1] *NVIDIA GTX Titan Z = 8.1 TFLOPS*
[2] *Intel Core i7 980 x6 3.33 GHz = 130 GFLOPS*

Programming language

Writing a "non-graphics" program with a graphics API



2003 Cg NVIDIA (Mark et al. ACM Trans. Graph.)

2004 Brook ATI (Buck et al. SIGGRAPH)

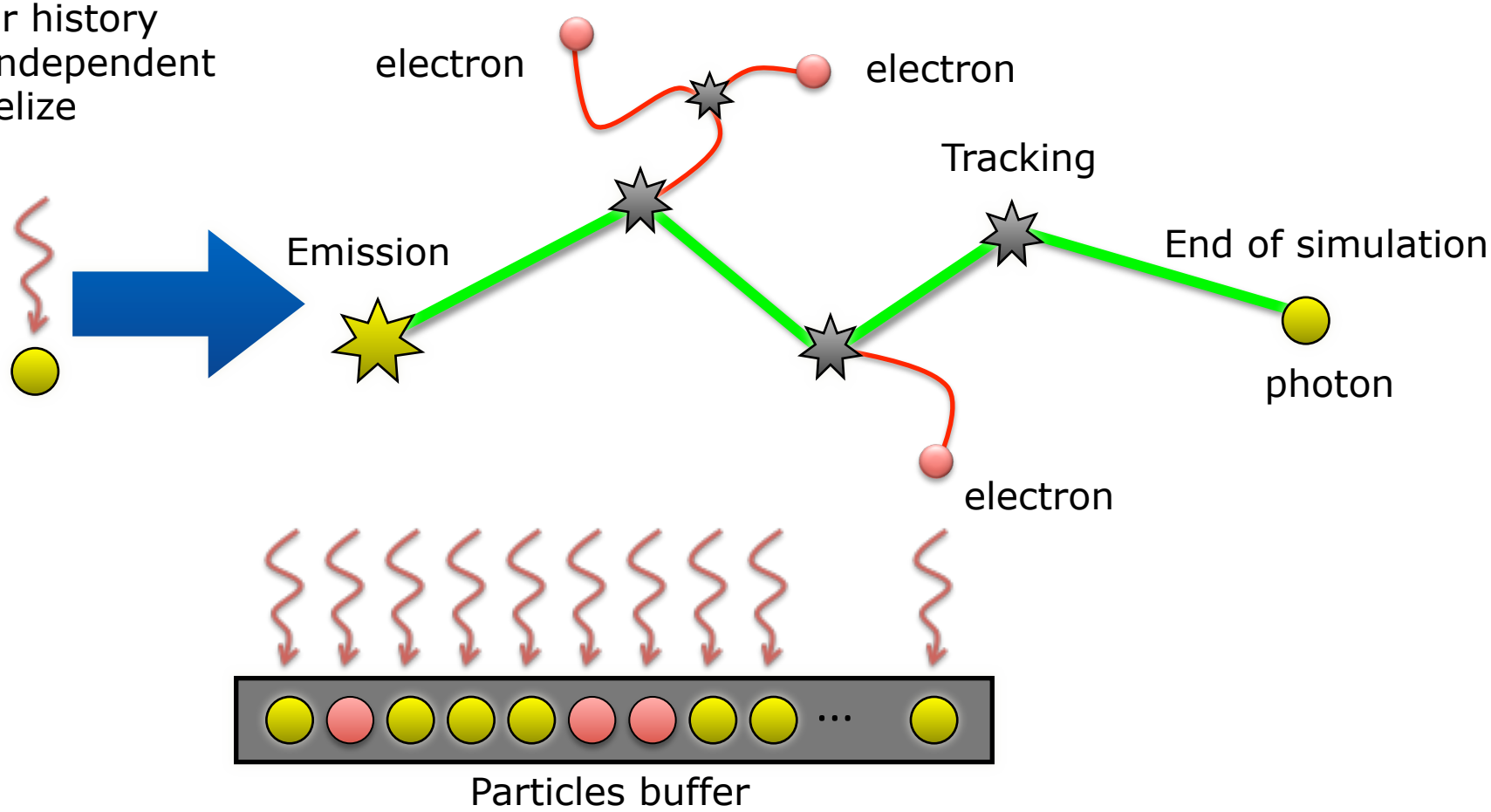
2004 OpenGL shading language (Kessenich et al. http://www.opengl.org)

2007 CUDA NVIDIA (Compute Unified Device Architecture)

Monte Carlo simulation on GPU

Paradigm

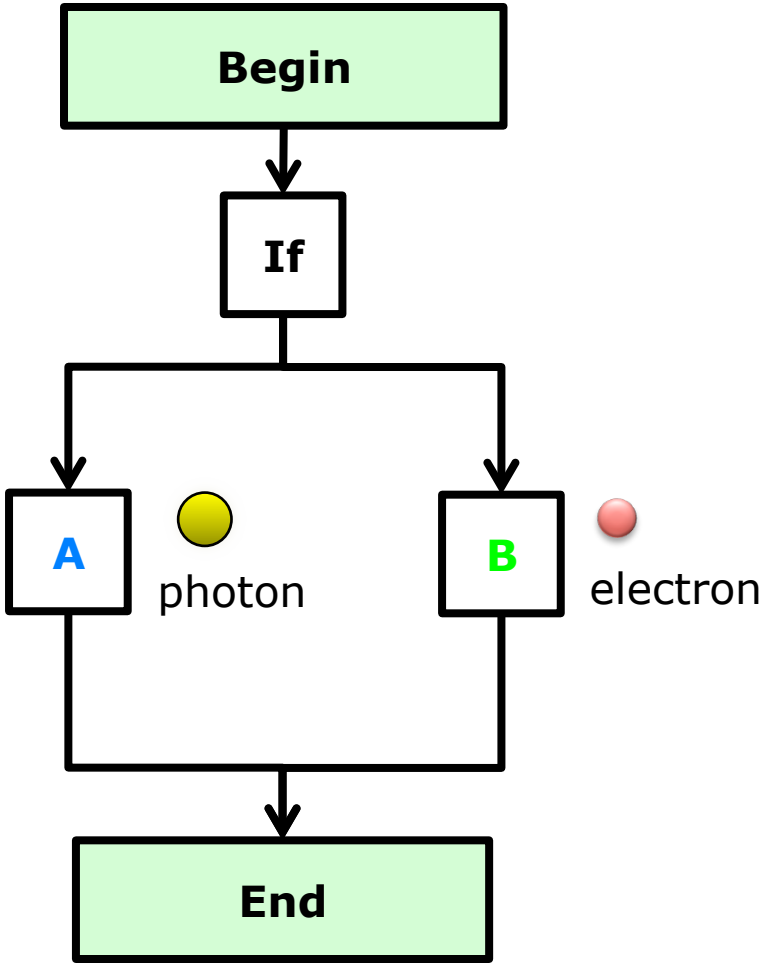
- one thread per history
- particles are independent
- easy to parallelize

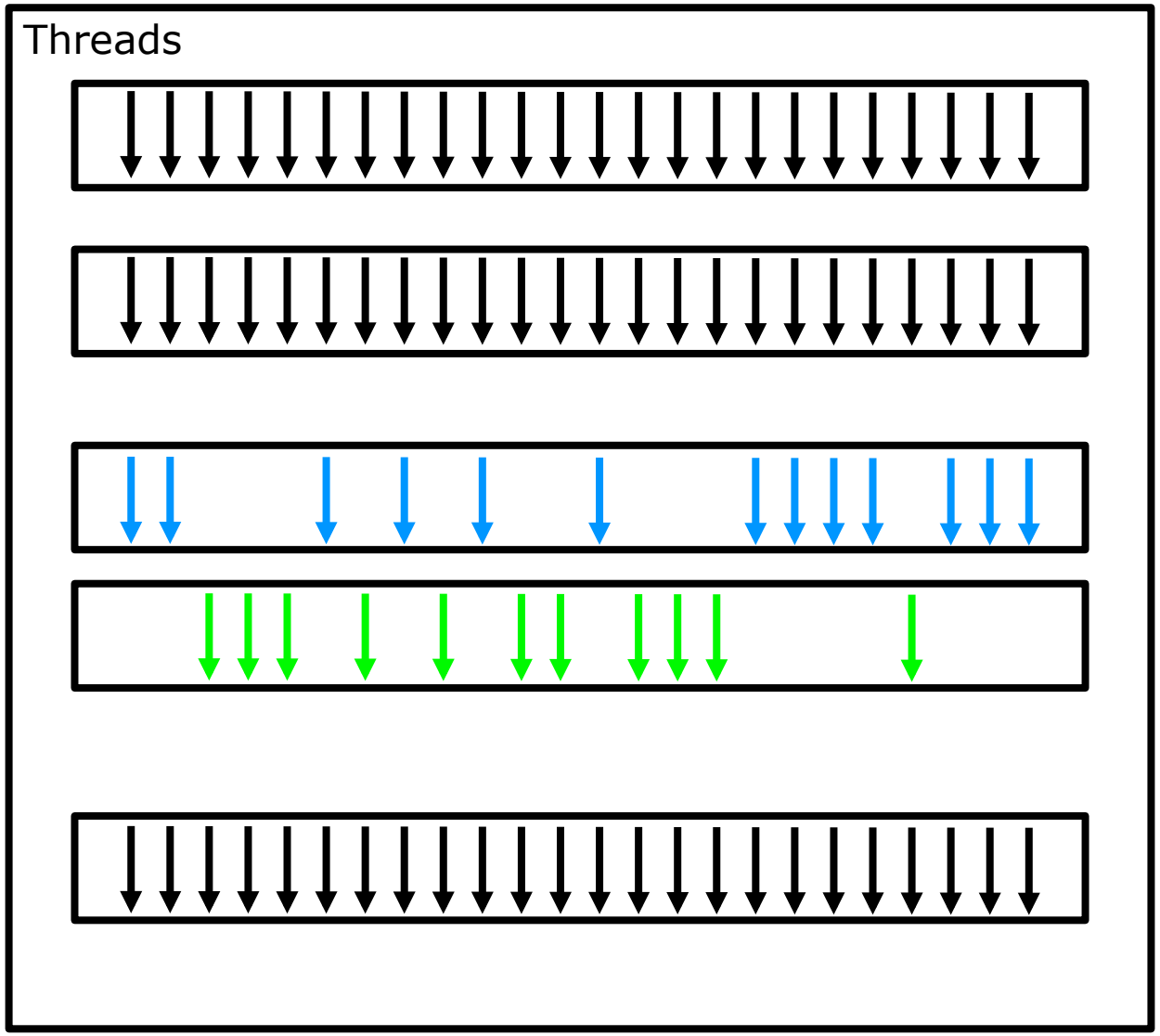


Thousands of particles are simulated in parallel

Divergence

If statement





Physics Model

Geant 4

Conseil européen pour la recherche nucléaire (CERN, Genève, Suisse)

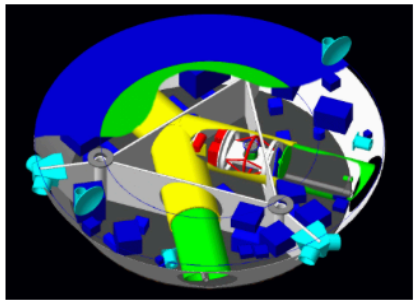
CERN: LHC, ATLAS **Fermilab**: MINOS **SLAC**: BaBar

Geant4 is a toolkit for the simulation of the passage of particles through matter. Its areas of application include high energy, nuclear and accelerator physics, as well as studies in medical and space science. The two main reference papers for Geant4 are published in *Nuclear Instruments and Methods in Physics Research* A 506 (2003) 250-303, and *IEEE Transactions on Nuclear Science* 53 No. 1 (2006) 270-278.

Applications

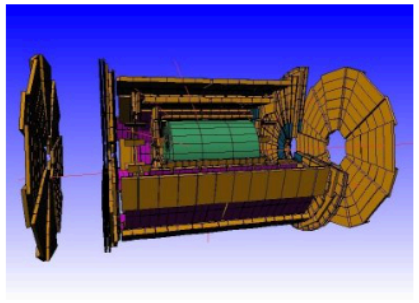
A sampling of applications, technology transfer and other uses of Geant4

User Support



Getting started, guides and information for users and developers

Publications



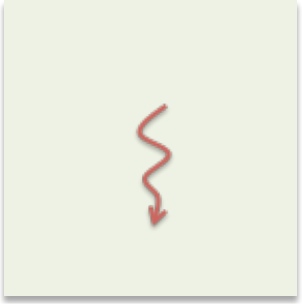
Validation of Geant4, results from experiments and publications

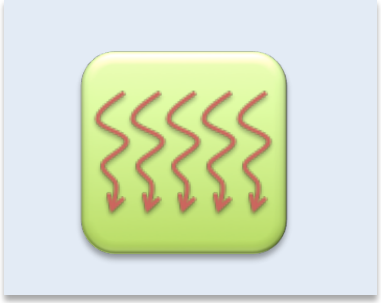
Collaboration

Who we are: collaborating institutions, members, organization and legal information

- Geant4 code on GPU (C++ ➔ C ➔ CUDA)

Types of memory

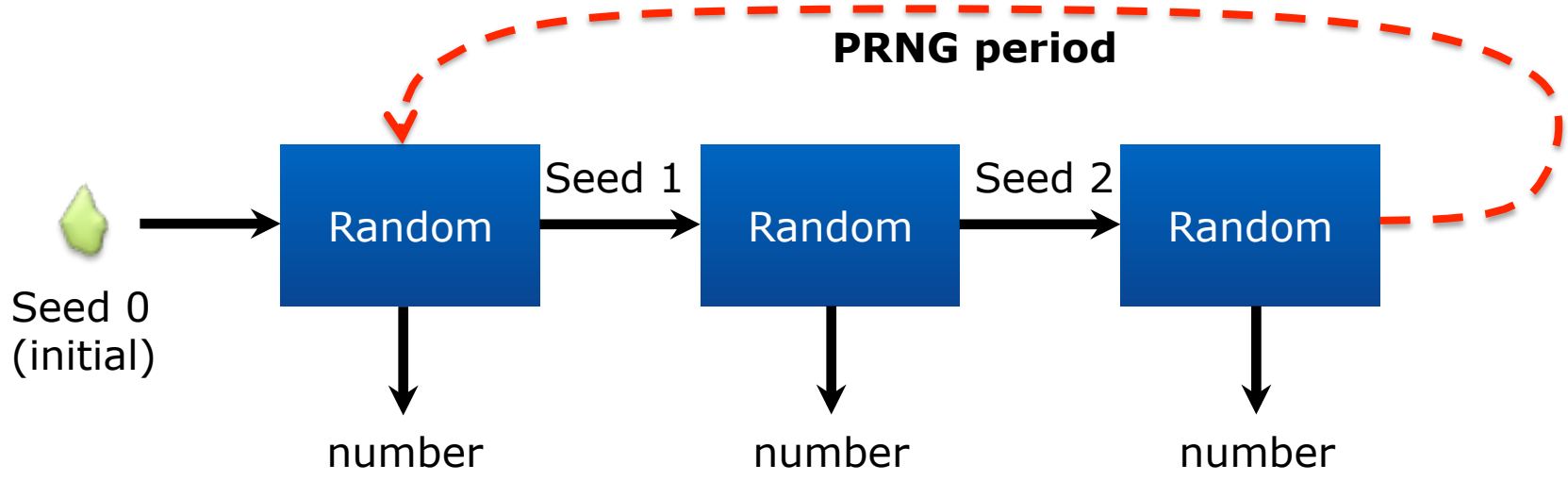




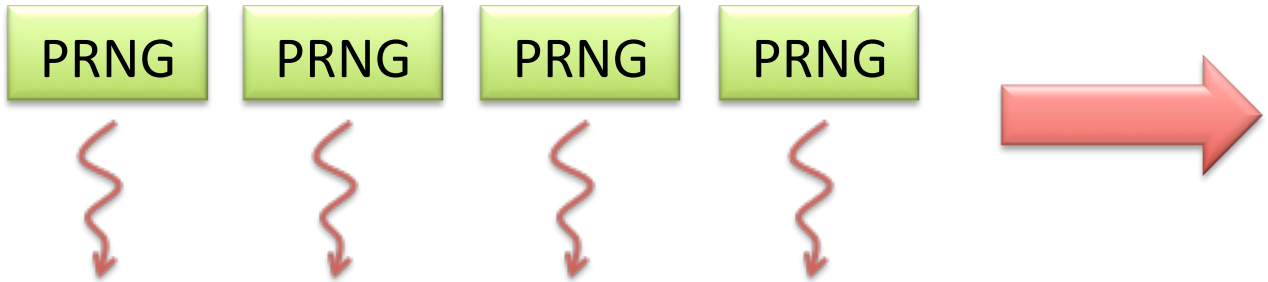
Storage type	Register	Shared memory	Texture memory	Constant memory	Global memory
Bandwith	~8 TB/s	~1.5 TB/s	~200 MB/s	~200 MB/s	~200 MB/s
Latency	1 cycle	1 to 32 cycles	~400 to 600	~400 to 600	~400 to 600
Lifetime	Function	kernel	application	application	application
Range	Thread	Block	All threads	All threads	All threads
Size					rd
Read/ Write	R/W	R/W	R	R	R/W
Cached	No	N/A	Yes	Yes	No
	limited variables		**geometry and physical tables**	**physical constants**	**particles**

Pseudo Random Generator Number (PRNG)

Return a random number uniformly distributed between 0 and 1

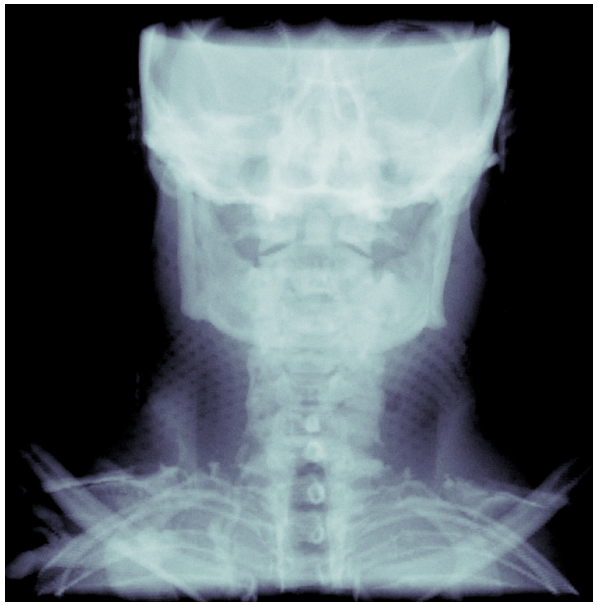


Need to store **generator state**



Find a suitable algorithm:
- small state size
- large period

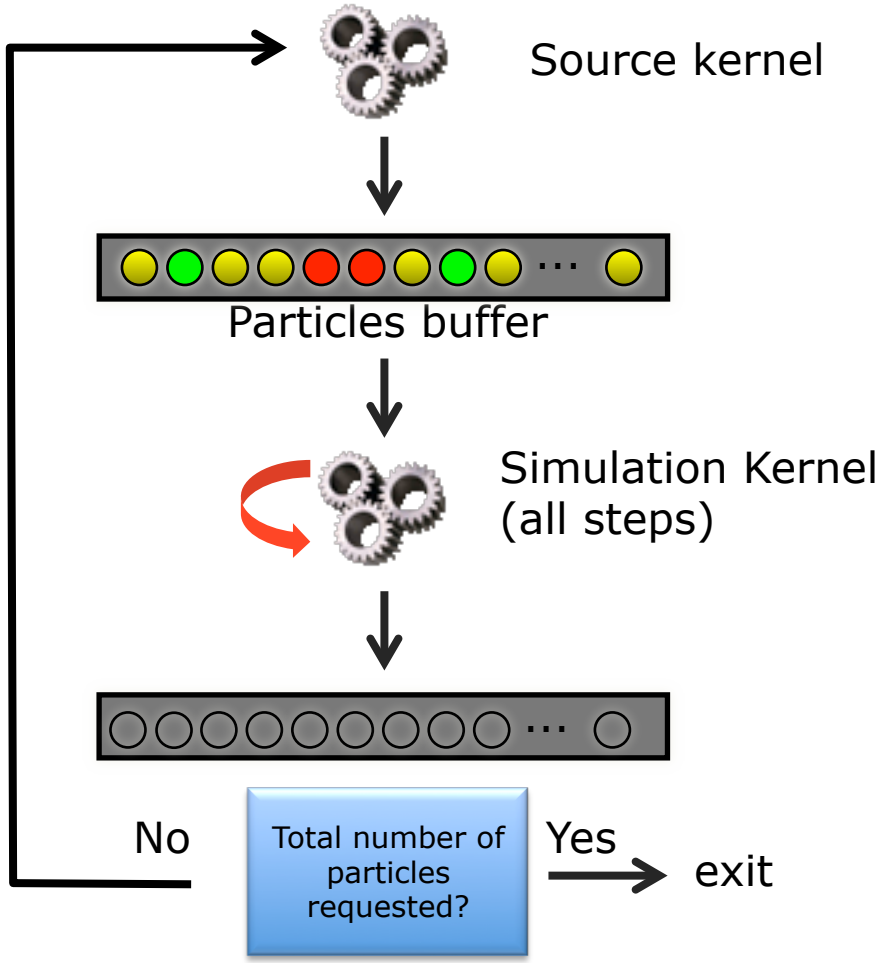
Simulation loop



14 billions of particles

~ 2-4 GB

Main simulation loop



Data structure

Array of Structure (AoS)

```
struct Point {
    float x;
    float y;
    float z;
};
```

- list of points
- regular access is **not optimized**

Structure of Array (SoA)

```
struct Point {
    float *x;
    float *y;
    float *z;
};
```

- Point contains a list of x, y, z values
- coalesced access is **optimized**

Particle data structure

```
20  // Stack of gamma particles, format
21  struct StackGamma{
22      float* E;
23      float* dx;
24      float* dy;
25      float* dz;
26      float* px;
27      float* py;
28      float* pz;
29      float* t;
30      unsigned int* seed;
31      unsigned char* interaction;
32      unsigned char* live;
33      unsigned char* endsimu;
34      unsigned char* ct_cpt;
35      unsigned char* ct_pe;
36      unsigned char* ct_ray;
37      unsigned int size;
38      unsigned long* table_x_brent;
39  }; //
40
```

Energy

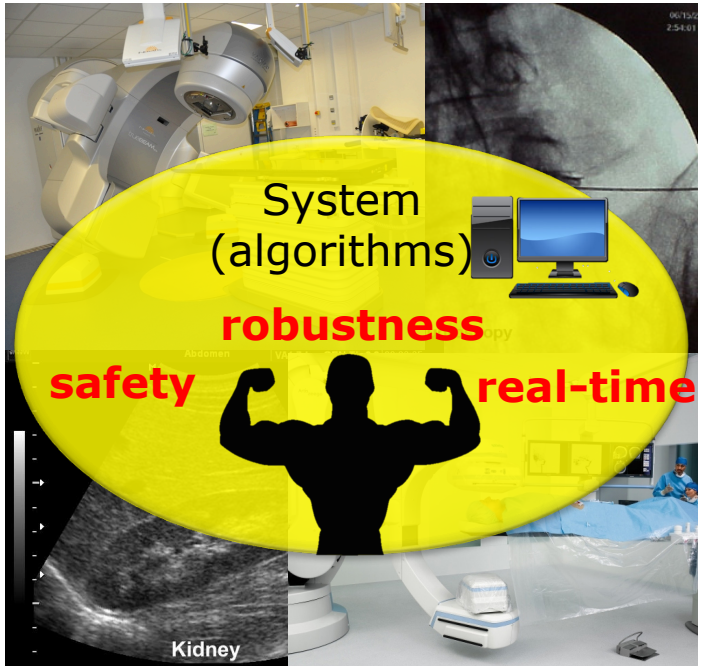
Direction

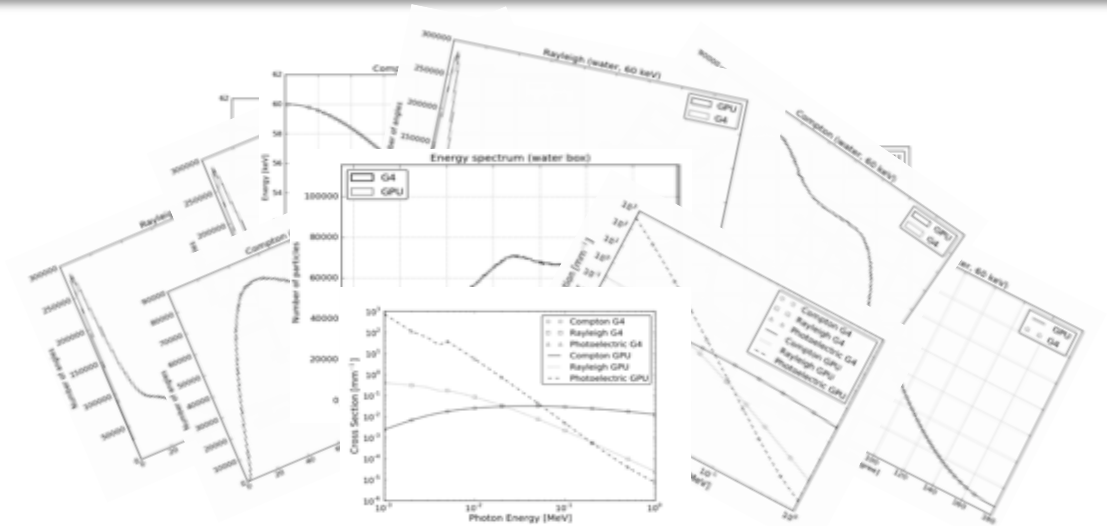
Position

Validation

Monte Carlo simulation on GPU (2012):

- **Pseudo random number generator**
- Electromagnetic effects for **photon**
  - Compton scattering
  - Rayleigh scattering
  - Photoelectric effect
- Voxelized **geometry navigation**
- **Materials properties**
- Single precision (float number)





2013 - Full agreement between **GPU code** and Geant4 (**CPU**)

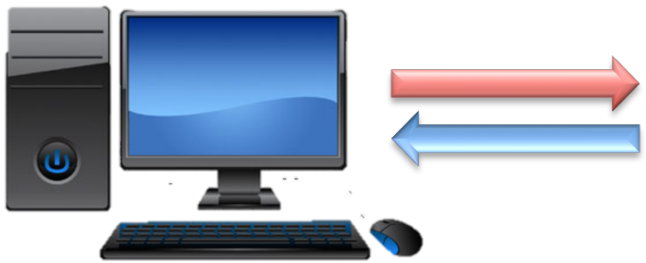
IOP Publishing — Physics in Medicine and Biology
Phys. Med. Biol. **58** (2013) 5593–5611 doi:10.1088/0031-9155/58/16/5593

Geant4-based Monte Carlo simulations on GPU for medical applications

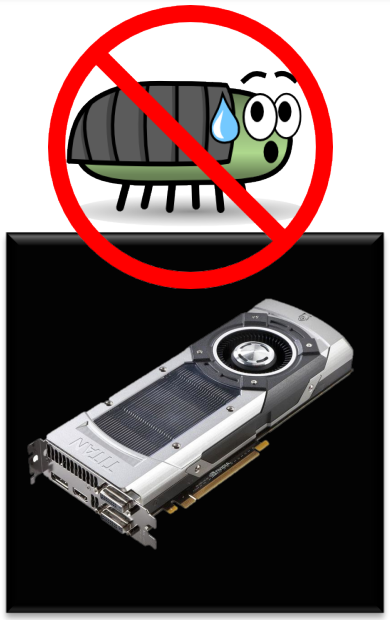
Julien Bert[1,5], Hector Perez-Ponce[2,5], Ziad El Bitar[3], Sébastien Jan[4], Yannick Boursier[2], Damien Vintache[3], Alain Bonissent[2], Christian Morel[2], David Brasse[3] and Dimitris Visvikis[1]

[1] LaTIM, UMR 1101 INSERM, CHRU Brest, Brest, France
[2] CPPM, Aix-Marseille Université, CNRS/IN2P3, Marseille, France
[3] IPHC, UMR 7178—CNRS/IN2P3, Strasbourg, France
[4] DSV/I2BM/SHFJ, Commissariat à l'Energie Atomique, Orsay, France

Bugs

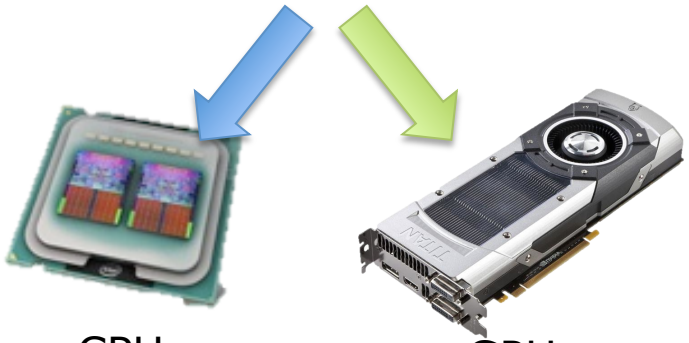


Tracking device bugs



Embedded system

```
__device__ __host__ void MyFunction()
```



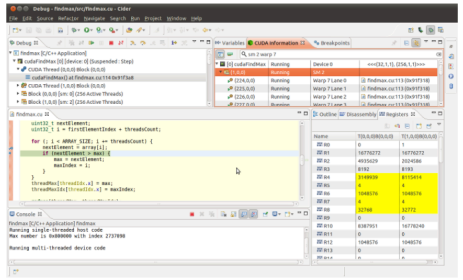
CPU GPU

Tracking bugs:

- printf (device code, cuda 3.1 2010)
- Nsight debugger

Nsight Debugger

- Seamless and simultaneous debugging of both CPU and GPU code
- View program variables across several CUDA threads
- Examine execution state and mapping of the kernels and GPUs
- View, Navigate and filter to selectively track execution across threads
- Set breakpoints and single-step execution at both source-code and assembly levels
- Includes cuda-memcheck to help detect memory errors



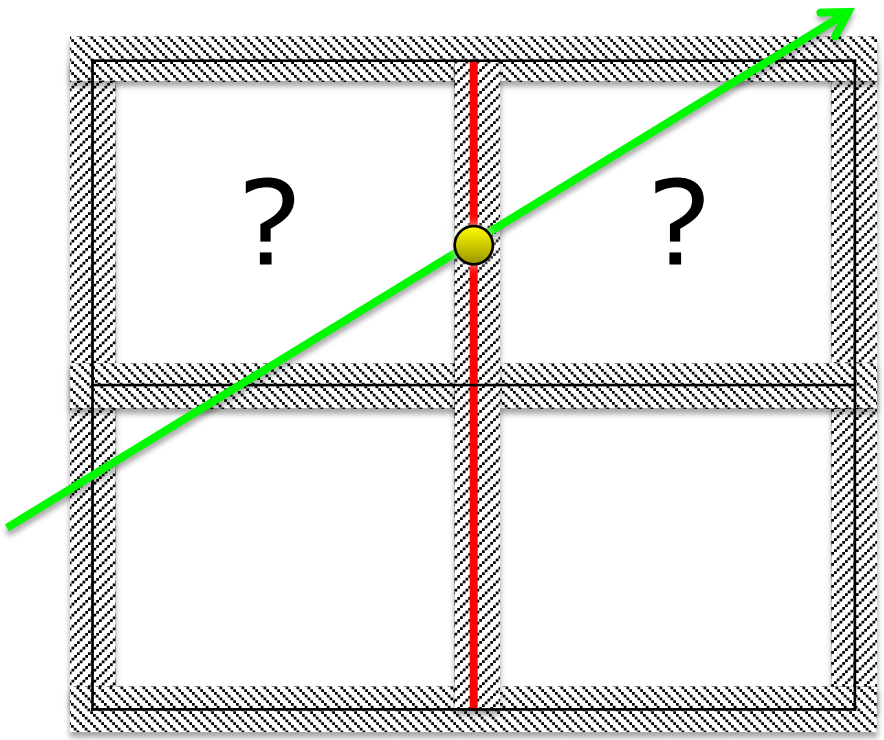
Science applications

Floating point IEEE 754 (*single precision*)

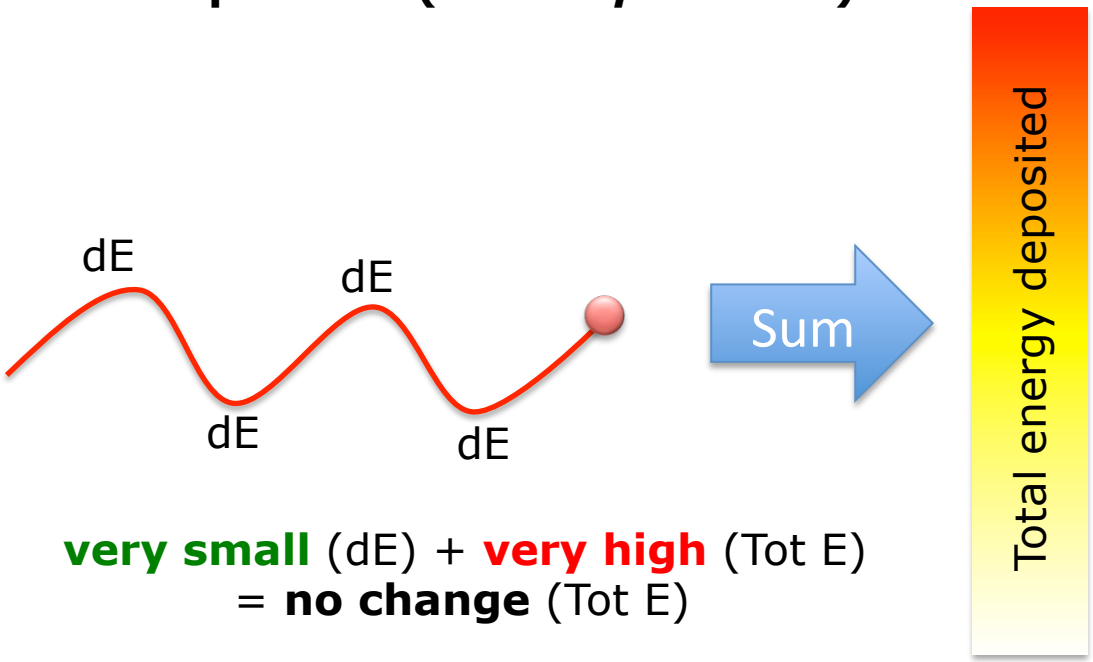
Double precision available on GPU:

- slower computation

Navigation (*single precision*)



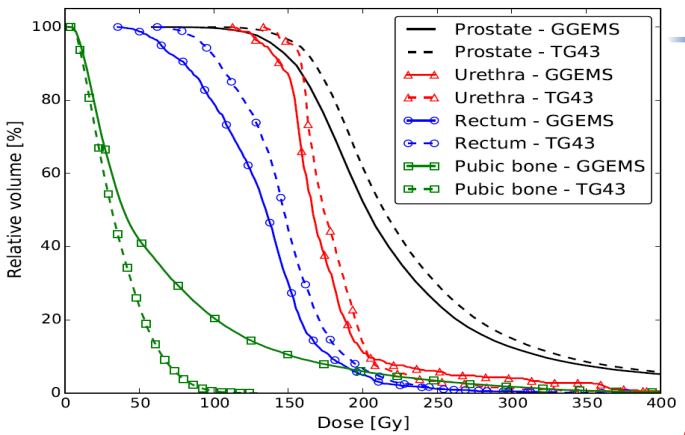
Dose deposition (*double precision*)

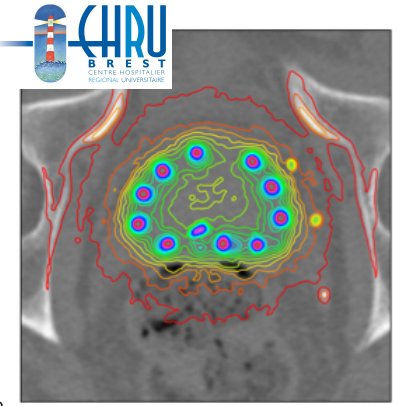


very small (dE) + **very high** (Tot E)
= **no change** (Tot E)

Clinical context

2015 - Radiotherapy (*brachytherapy*)





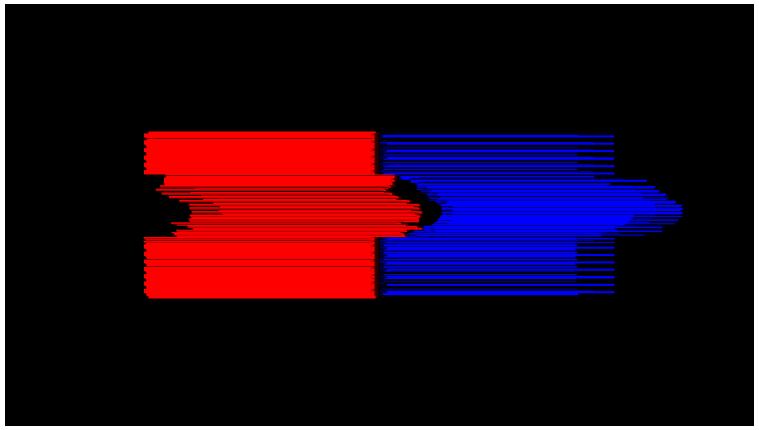
dosimetry in 2 s

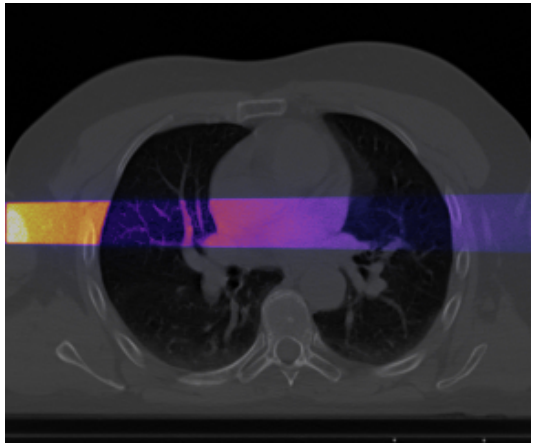
IOP Publishing | Institute of Physics and Engineering in Medicine — Physics in Medicine & Biology
Phys. Med. Biol. **60** (2015) 4987–5006 — doi:10.1088/0031-9155/60/13/4987

GGEMS-Brachy: GPU GEant4-based Monte Carlo simulation for brachytherapy applications

Yannick Lemaréchal[1], Julien Bert[1], Claire Falconnet[2], Philippe Després[3,4], Antoine Valeri[4,5], Ulrike Schick[1,2], Olivier Pradier[1,2], Marie-Paule Garcia[1], Nicolas Boussion[1,2] and Dimitris Visvikis[1]

Radiotherapy (*external beam*)



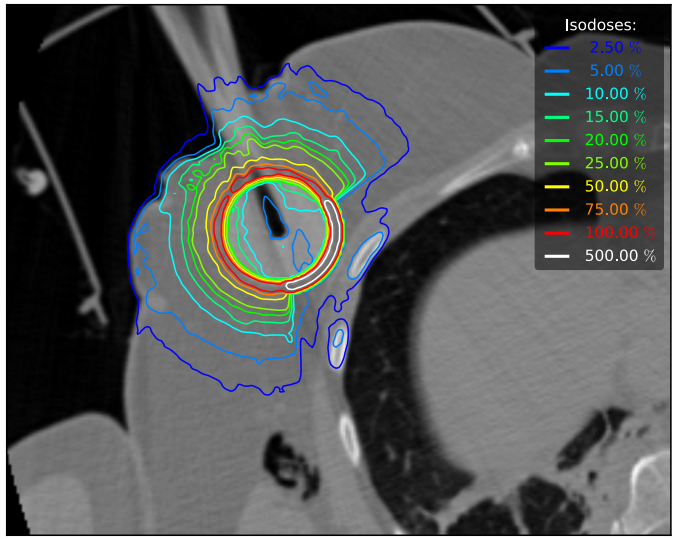


GGEMS

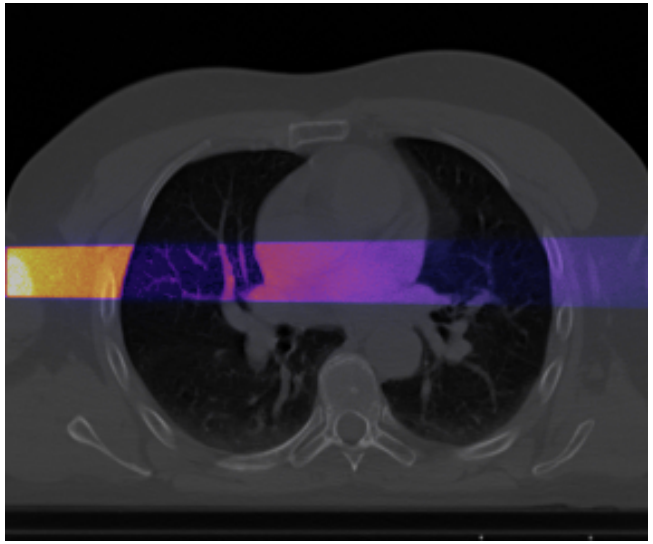
GPU GEant4-based Monte Carlo Simulations

http://www.ggems.fr

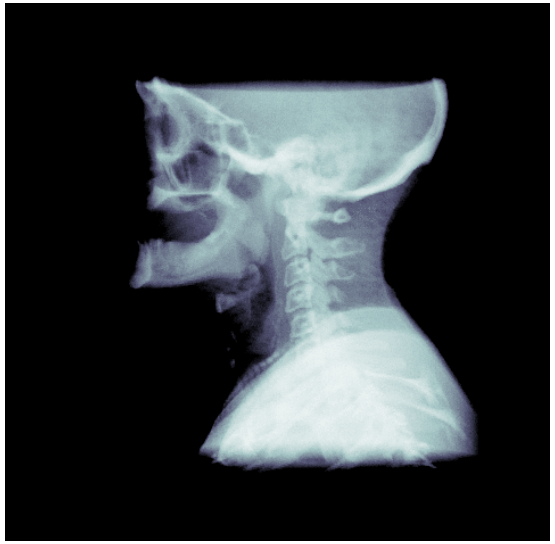
GGEMS is a library not a software.



Intra-Operative Radiotherapy



External Beam Radiotherapy



Medical Imaging

NVIDIA

x80-x150

Heterogeneous architecture

	Compute Capability							
Technical Specifications	**1.0**	**1.1**	**1.2**	**1.3**	**2.x**	**3.0**	**3.5**	**5.0**
Number of 32-bit registers per multiprocessor	8 K		16 K		32 K	64 K		
Maximum number of 32-bit registers per thread	128				63		255	
Maximum amount of shared memory per multiprocessor	16 KB				48 KB			64 KB

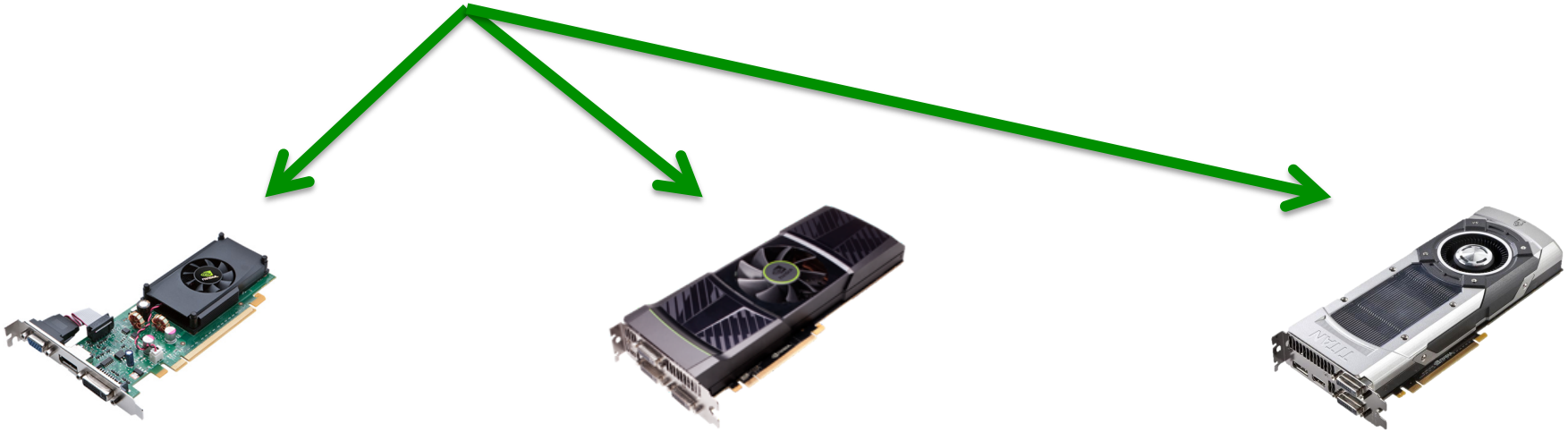
GTX580 cc 2.0 512 cores (Fermi)
GTX680 cc 3.0 1536 cores (Kepler)

Feature Support	Compute Capability						
(Unlisted features are supported for all compute capabilities)	**1.0**	**1.1**	**1.2**	**1.3**	**2.x**	**3.0**	**3.5, 5.0**
Warp vote functions (Warp Vote Functions)							
Double-precision floating-point numbers	No			Yes			
Atomic functions operating on 64-bit integer values in shared memory (Atomic Functions)	No				Yes		
Atomic addition operating on 32-bit floating point values in global and shared memory (atomicAdd())							

GTX Titan cc 3.5 2688 cores (Kepler)

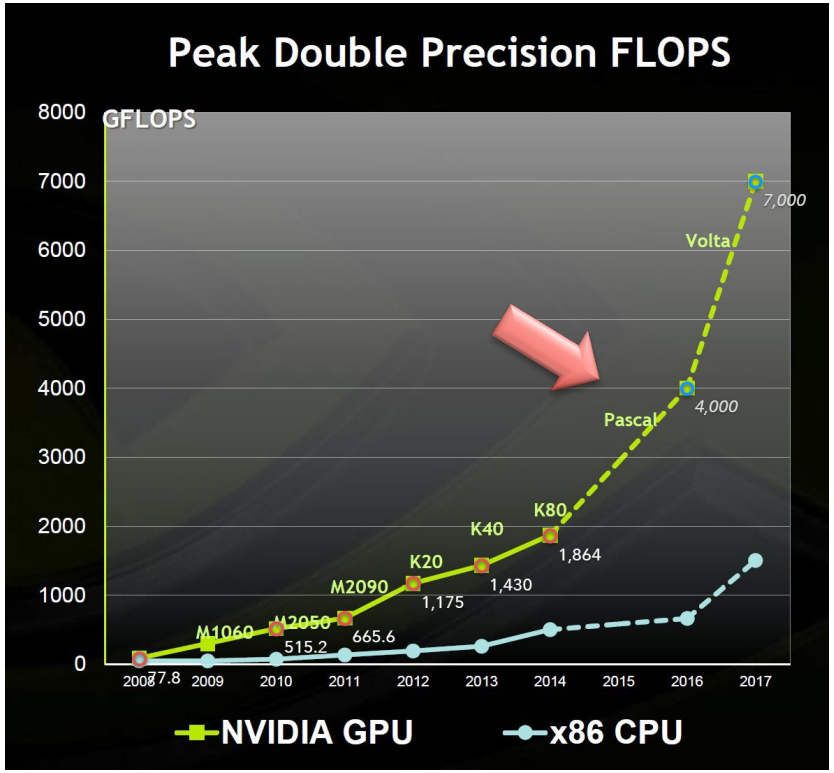
Heterogeneous architecture

```
CUDAHOME := /usr/local/cuda/lib64
CUDALIBS := -L$(CUDAHOME)/lib -lcudart
CUDAFLAGS := -gencode=arch=compute_30,code=sm_30 \
             -gencode=arch=compute_32,code=sm_32 \
             -gencode=arch=compute_35,code=sm_35 \
             -gencode=arch=compute_37,code=sm_37 \
             -gencode=arch=compute_50,code=sm_50 \
             -gencode=arch=compute_52,code=sm_52 \
             --compiler-options -w --std c++11
```



Heterogeneous architecture

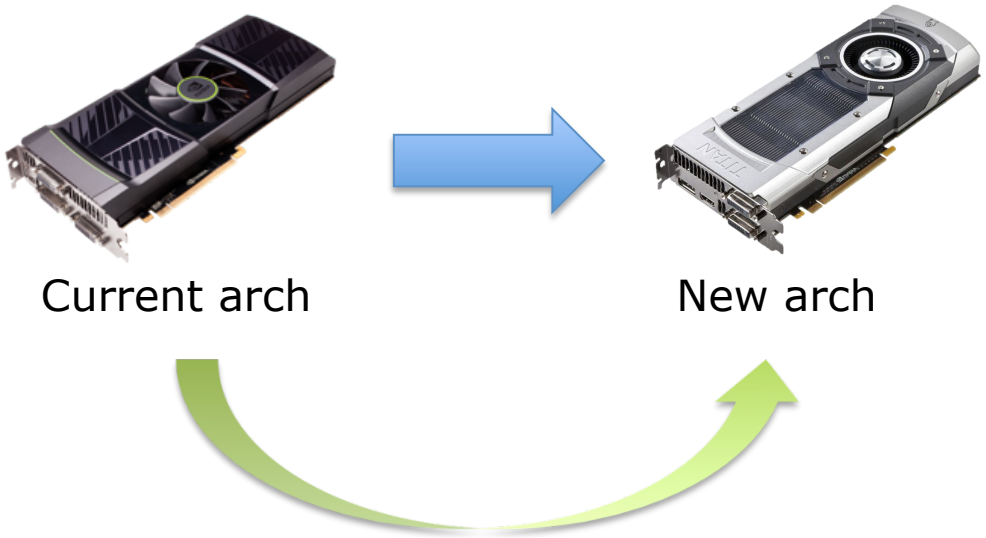
Hardware evolution:



Hardware upgrade have a cost!

CUDA, new features:

- updating the code
- using new graphics cards



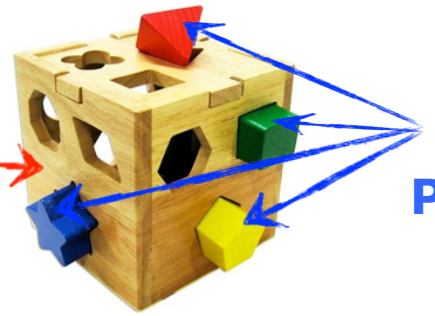
Instantly speed up your application

Conclusion

Main CUDA code is similar to the original one (C++)

```
// Compton Cross Section Per Atom (Standard - Klein-Nishina)
__device__ float Compton_CSPA(float E, unsigned short int Z) {
    float CrossSection = 0.0;
    if (Z<1 || E < 1e-4f) {
        return CrossSection;
        }

    float p1Z = Z*(2.7965e-23f + 1.9756e-27f*Z + -3.9178e-29f*Z*Z);
    float p2Z = Z*(-1.8300e-23f + -1.0205e-24f*Z + 6.8241e-27f*Z*Z);
    float p3Z = Z*(6.7527e-22f + -7.3913e-24f*Z + 6.0480e-27f*Z*Z);
    float p4Z = Z*(-1.9798e-21f + 2.7079e-24f*Z + 3.0274e-26f*Z*Z);
    float T0 = (Z < 1.5f)? 40.0e-3f : 15.0e-3f;
    float d1, d2, d3, d4, d5;

    d1 = __fdividef(fmaxf(E, T0), 0.510998910f); // X
    CrossSection = __fdividef(p1Z*__logf(1.0f+2.0f*d1), d1) + __fdividef(p2Z + p3Z*d1 + p4Z*d1*d1, 1.0f +
    20.0f*d1 + 230.0f*d1*d1 + 440.0f*d1*d1*d1);

    if (E < T0) {
        d1 = __fdividef(T0+1.0e-3f, 0.510998910f); // X
        d2 = __fdividef(p1Z*__logf(1.0f+2.0f*d1), d1) + __fdividef(p2Z + p3Z*d1 + p4Z*d1*d1, 1.0f + 20.0f*d1 +
        230.0f*d1*d1 + 440.0f*d1*d1*d1); // sigma
        d3 = __fdividef(-T0 * (d2 - CrossSection), CrossSection*1.0e-3f); // c1
        d4 = (Z > 1.5f)? 0.375f-0.0556f*__logf(Z) : 0.15f; // c2
        d5 = __logf(__fdividef(E, T0)); // y
        CrossSection *= __expf(-d5 * (d3 + d4*d5));
        }

    return CrossSection;
    }
```

Architecture



~~Algorithms~~

Programming model

within this context

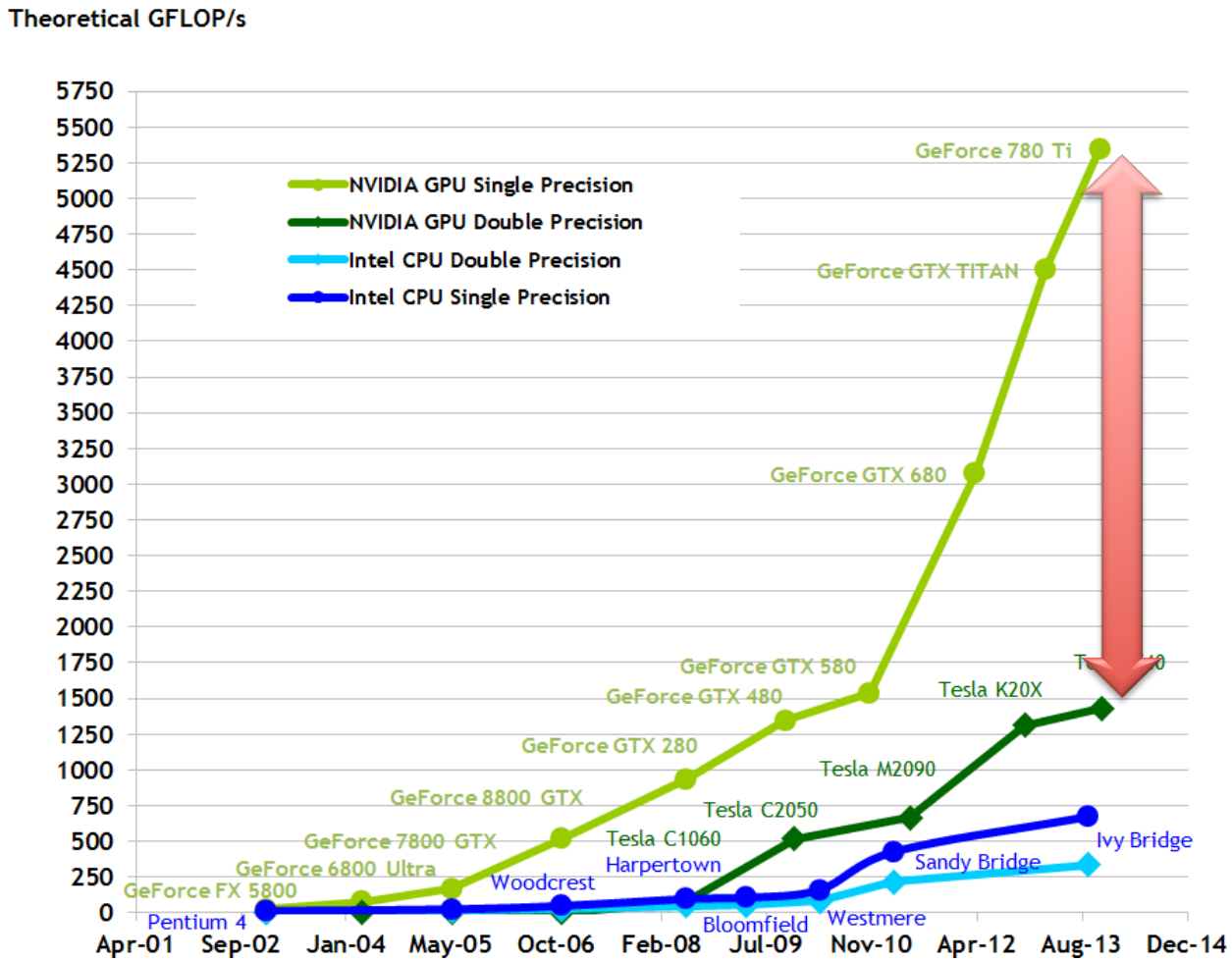
Conclusion

GPU-Accelerated Libraries

- cuFFT – Fast Fourier Transforms Library
- cuBLAS – Complete BLAS library
- cuSPARSE – Sparse Matrix library
- cuRAND – Random Number Generator
- NPP – Thousands of Performance Primitives for Image & Video Processing
- Thrust – Templated Parallel Algorithms & Data Structures
- CUDA Math Library of high performance math routines

- PhysX – scalable multi-platform game physics solution
- VisualFX – solutions for complex, cinematic visual effects
- OptiX – Fast ray tracing engine

Conclusion

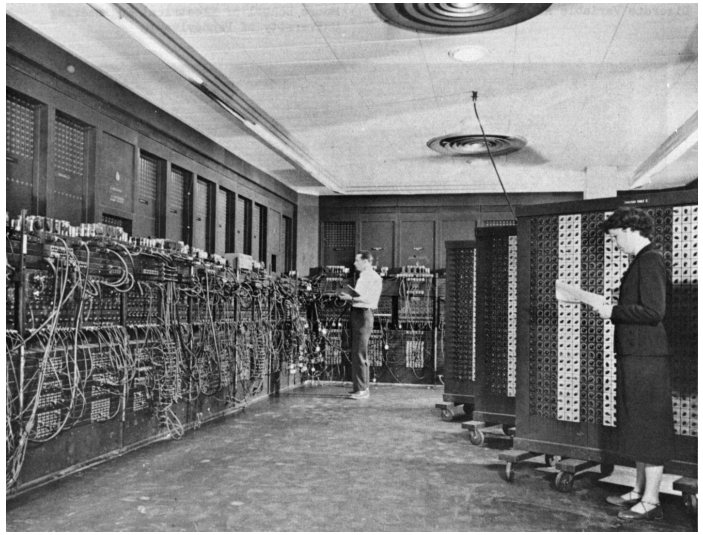


Specifications:

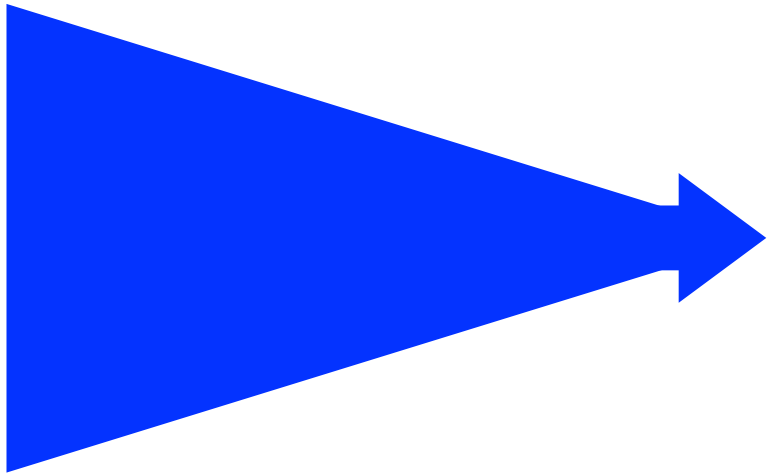
Number of 32-bit registers per multiprocessor	8 K	16 K	32 K	64 K
Maximum number of 32-bit registers per thread	128		63	255
Maximum amount of shared memory per multiprocessor	16 KB		48 KB	64 KB

Developer friendly:
- debugging
- thread exception error

Conclusion

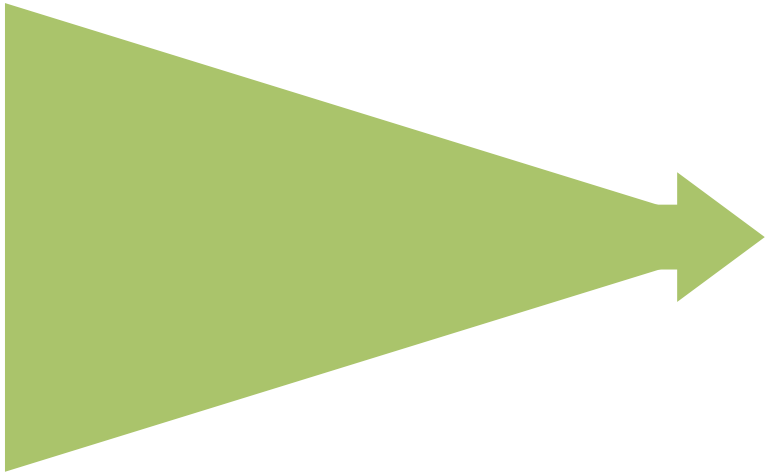


Computer



Personal **C**omputer

Cluster



Personal **S**uper **C**omputer

Conclusion

- Computer assisted medical intervention
- Image-guide radiotherapy
- Treatment planning
- …

Questions?

Julien Bert, PhD

CHRU de Brest
LaTIM – INSERM UMR1101