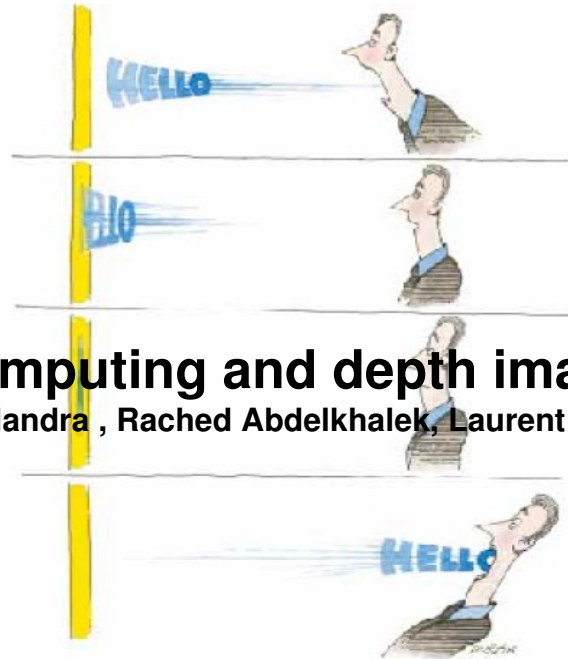


---

## Time Reversal Cartoon.

---



# High performance computing and depth imaging the way to go?

Henri Calandra , Rached Abdelkhalek, Laurent Derrien

From Scientific American, November 1999 (M. Fink).

---

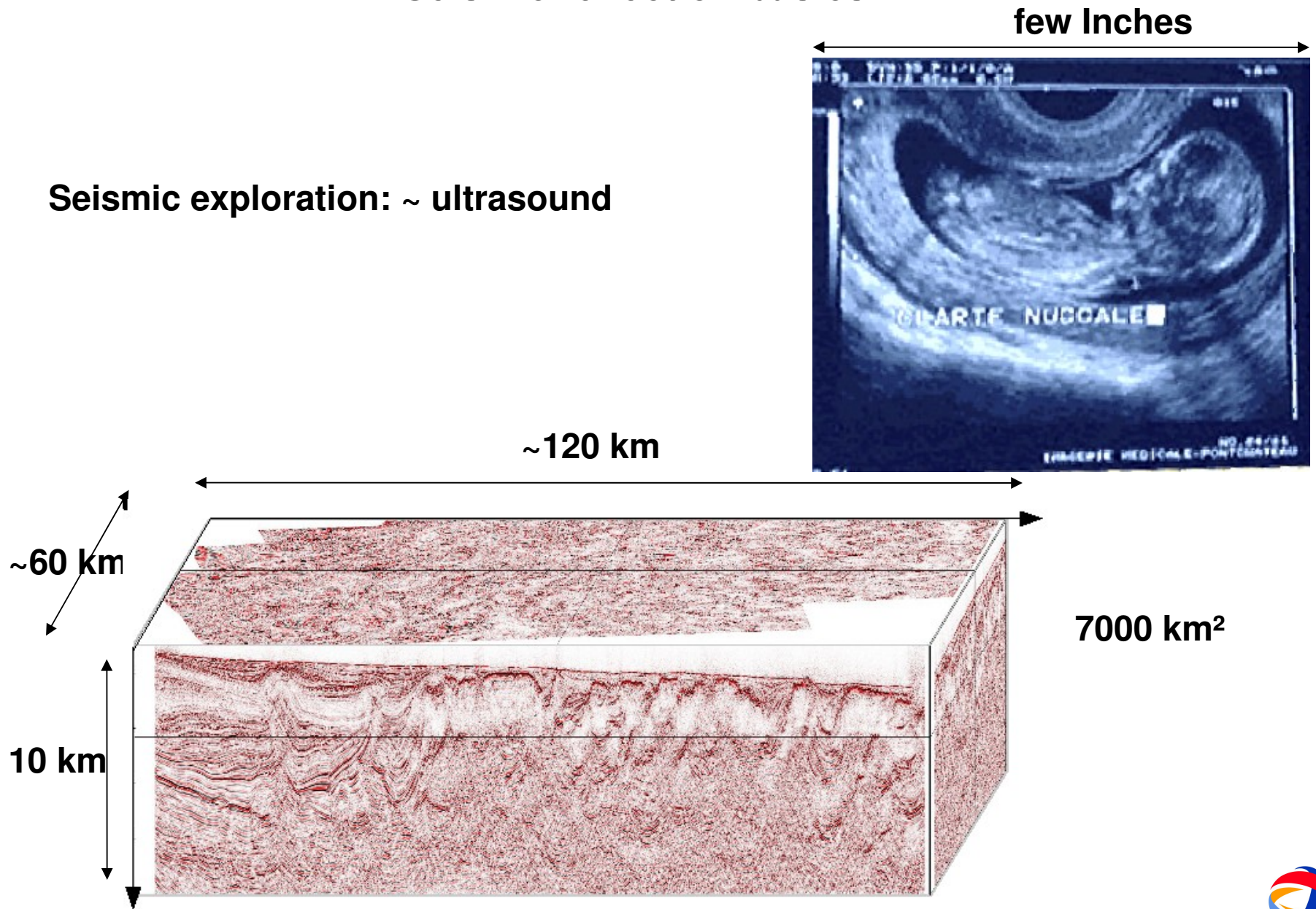
# Outline

- ▶ **introduction to seismic depth imaging**
- ▶ **Seismic exploration Challenges**
- ▶ **Looking for petascale and more ...**
- ▶ **Example: Reverse Time Migration**
- ▶ **Conclusions**

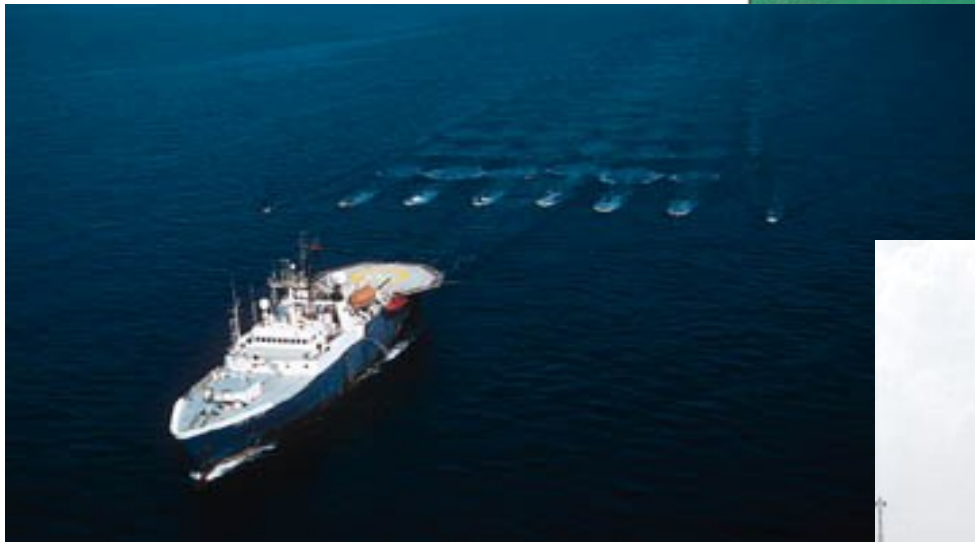
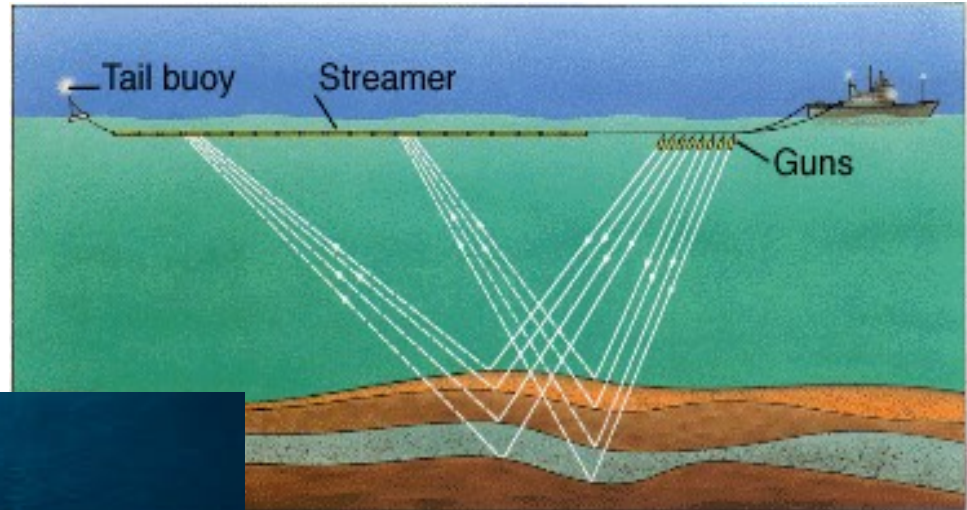
# **introduction to seismic depth imaging**

# Seismic reflection basics

Seismic exploration: ~ ultrasound

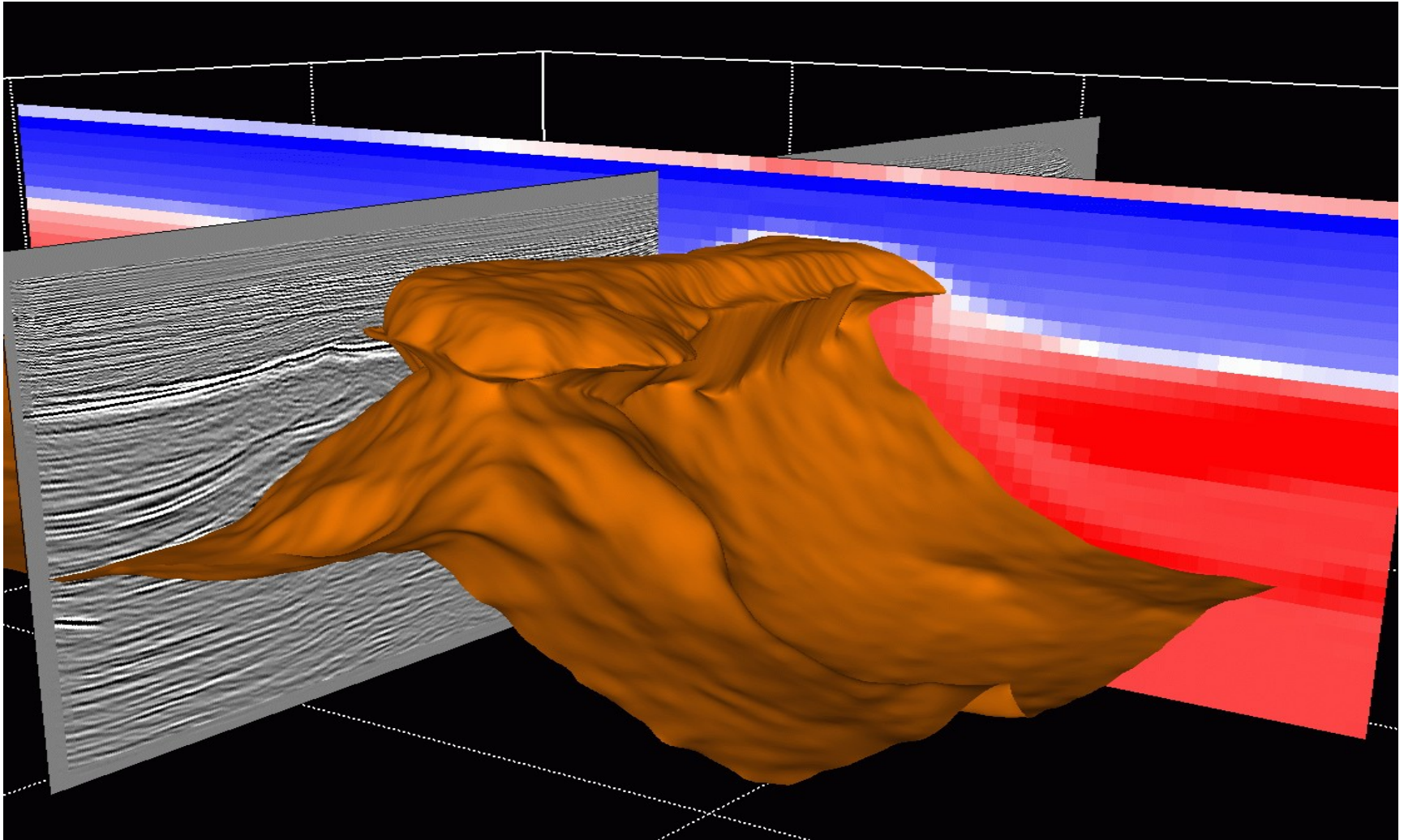


# Seismic reflection basics





# Obtain the most accurate subsurface model representation

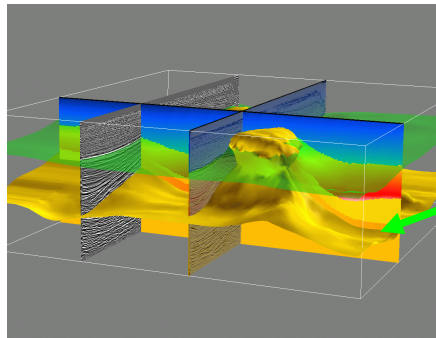
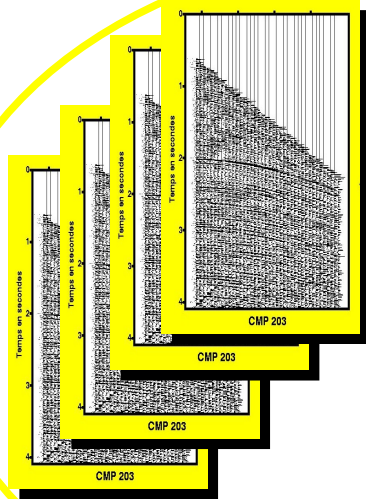


# Depth imaging is an inverse problem

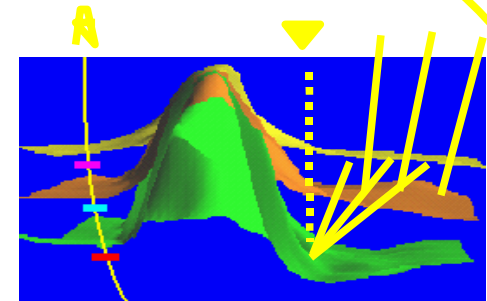
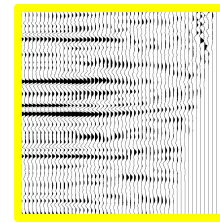
## 3D Pre-stack Depth Migration loop

## Inverse problem

Very CPU intensive



Depth migration



Velocity  
update

Less CPU intensive  
But human time intensive

# Outline

- ▶ introduction to seismic depth imaging
- ▶ **Seismic exploration Challenges**
- ▶ Looking for petascale and more ...
- ▶ Example: Reverse Time Migration
- ▶ Conclusions

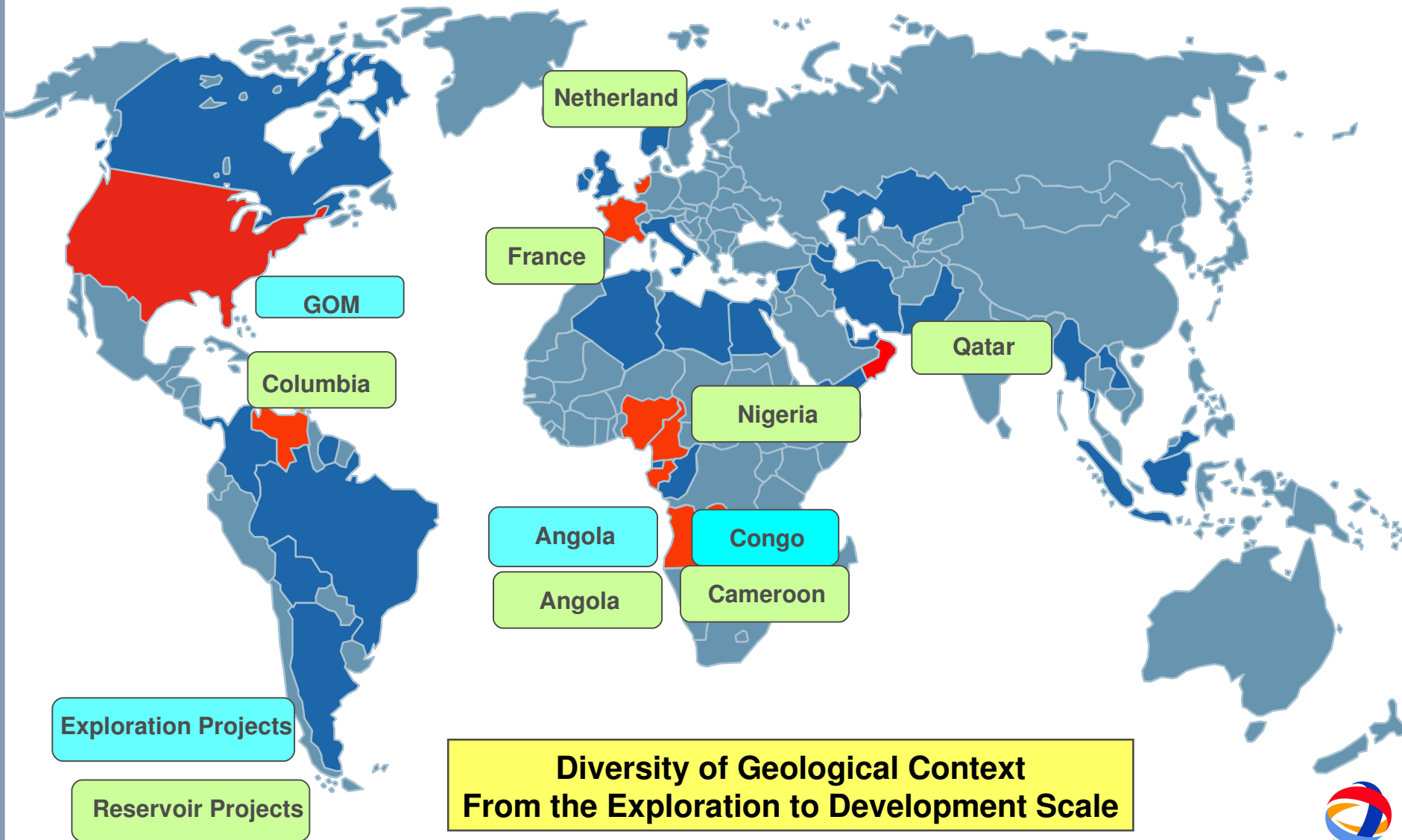


## **Seismic exploration Challenges**

- Business challenges
- Technical challenges
- Change of mind:
  - Size of survey acquisition
  - Computing effort and algorithm design

# Technical Challenges :

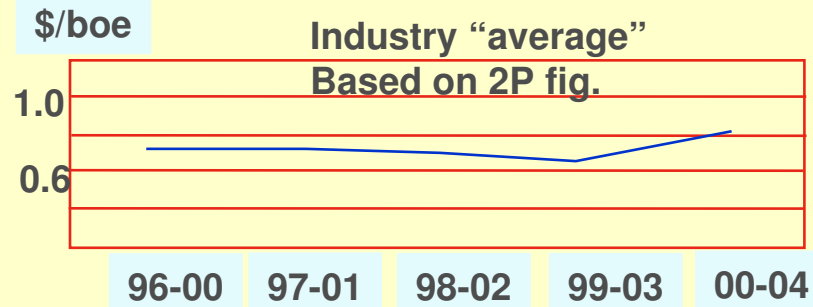
Sept 2005 - Sept 2006 **Total In-House** 3D PSDM Activity



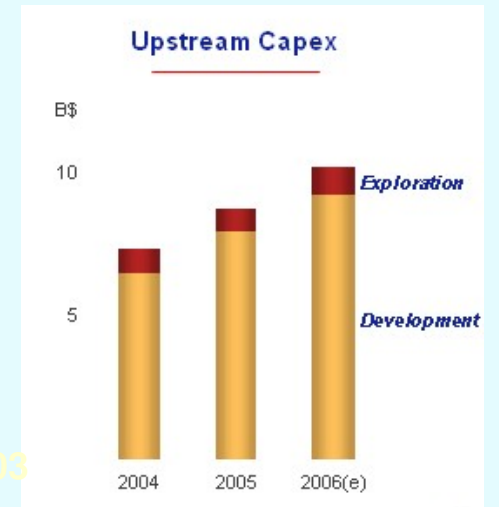
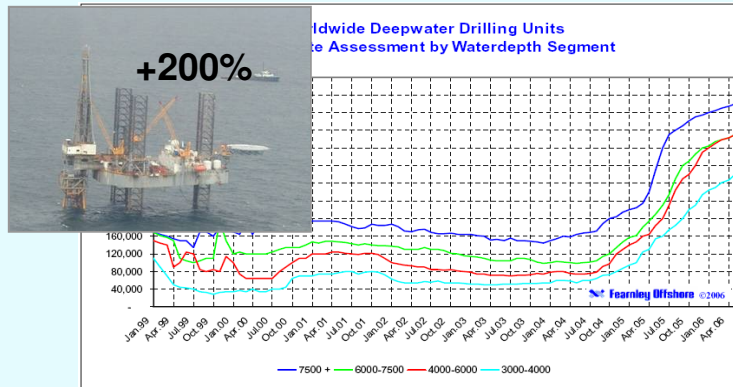
TOTAL

# Business Challenges

## ► Increase of Discovery costs



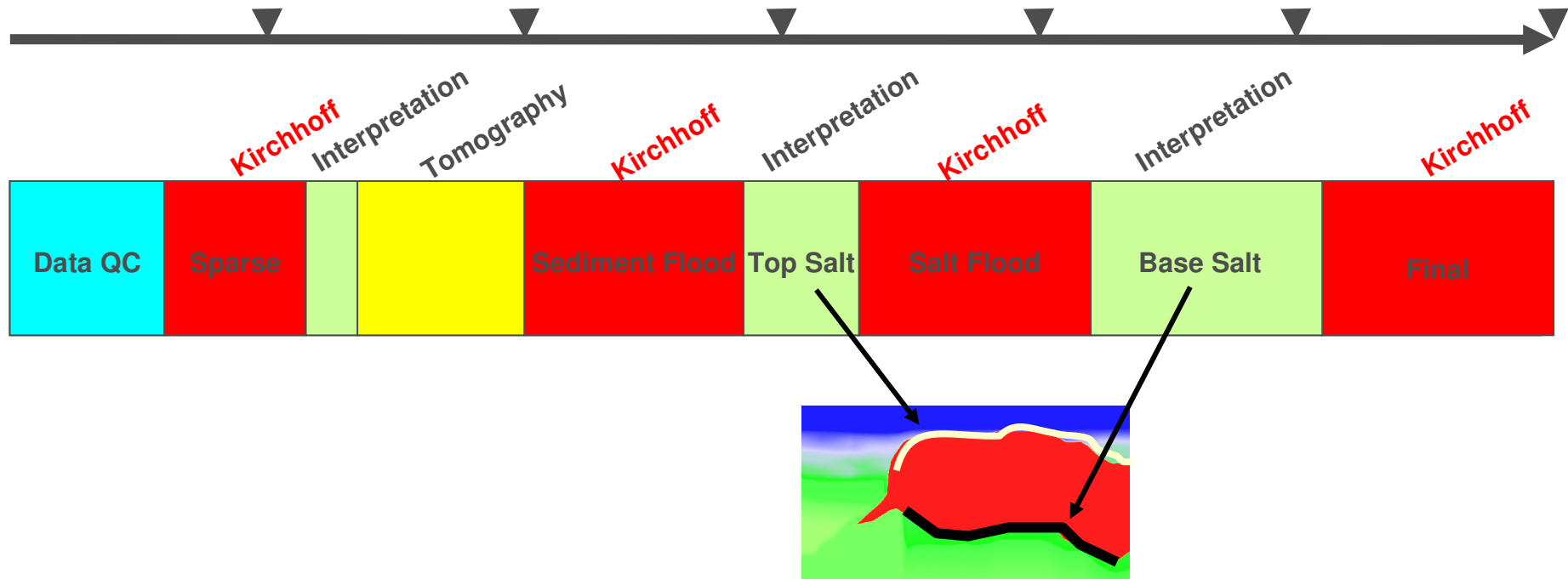
## ► Increase of CAPEX (seismic & drilling)



## Challenging cycle times

# Challenging cycle time

## Integrated Flow chart 3D PSDM Sub Salt - 2000-2004

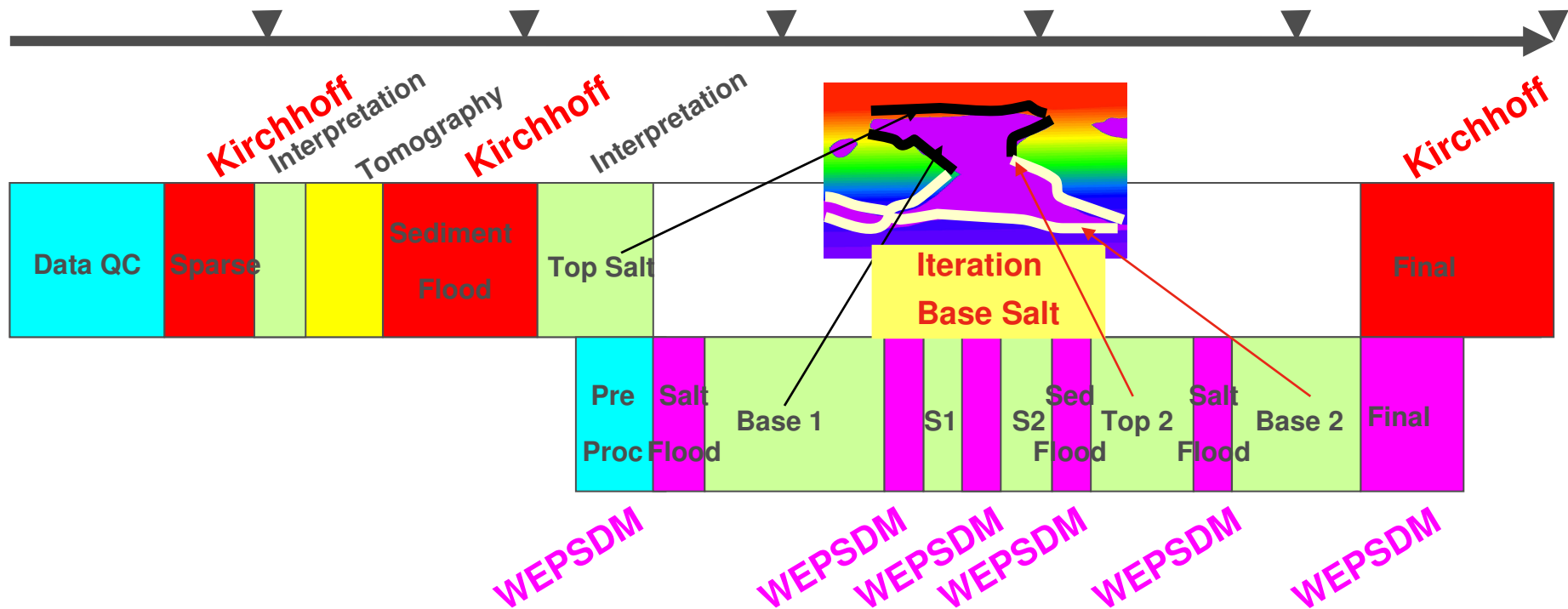


Imaging Project (400 km<sup>2</sup>) : 6 months– 4 migrations

Usually no iteration of Salt bodies model

# Challenging cycle time

## Integrated Flow chart 3D PSDM Sub Salt - 2005-2007



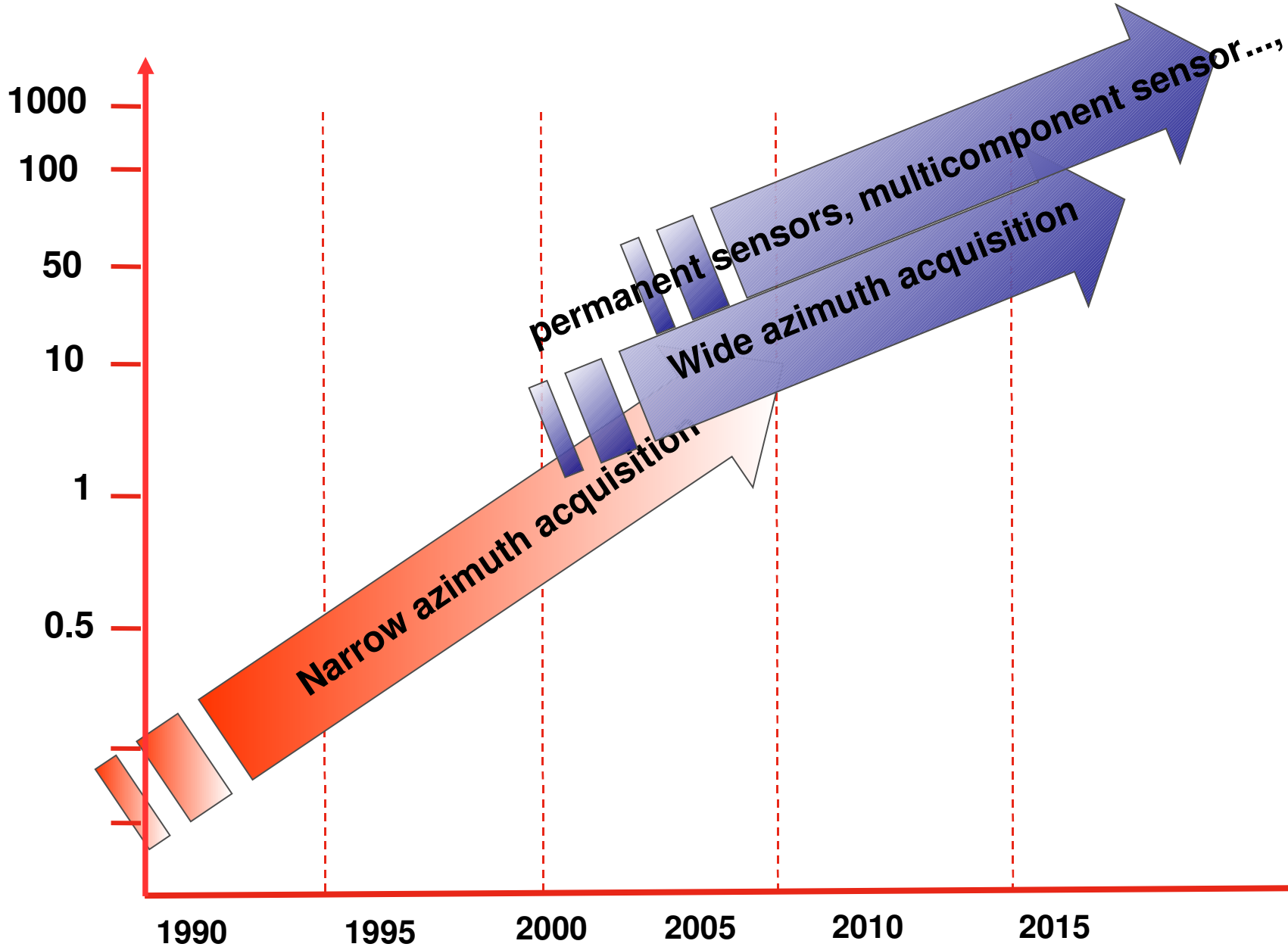
Imaging Project : 6 months (800 km<sup>2</sup>) – 10 migrations

Full integrated work within asset interpreter & depth imager

Salt Bodies Interpretation & Migration Iterations

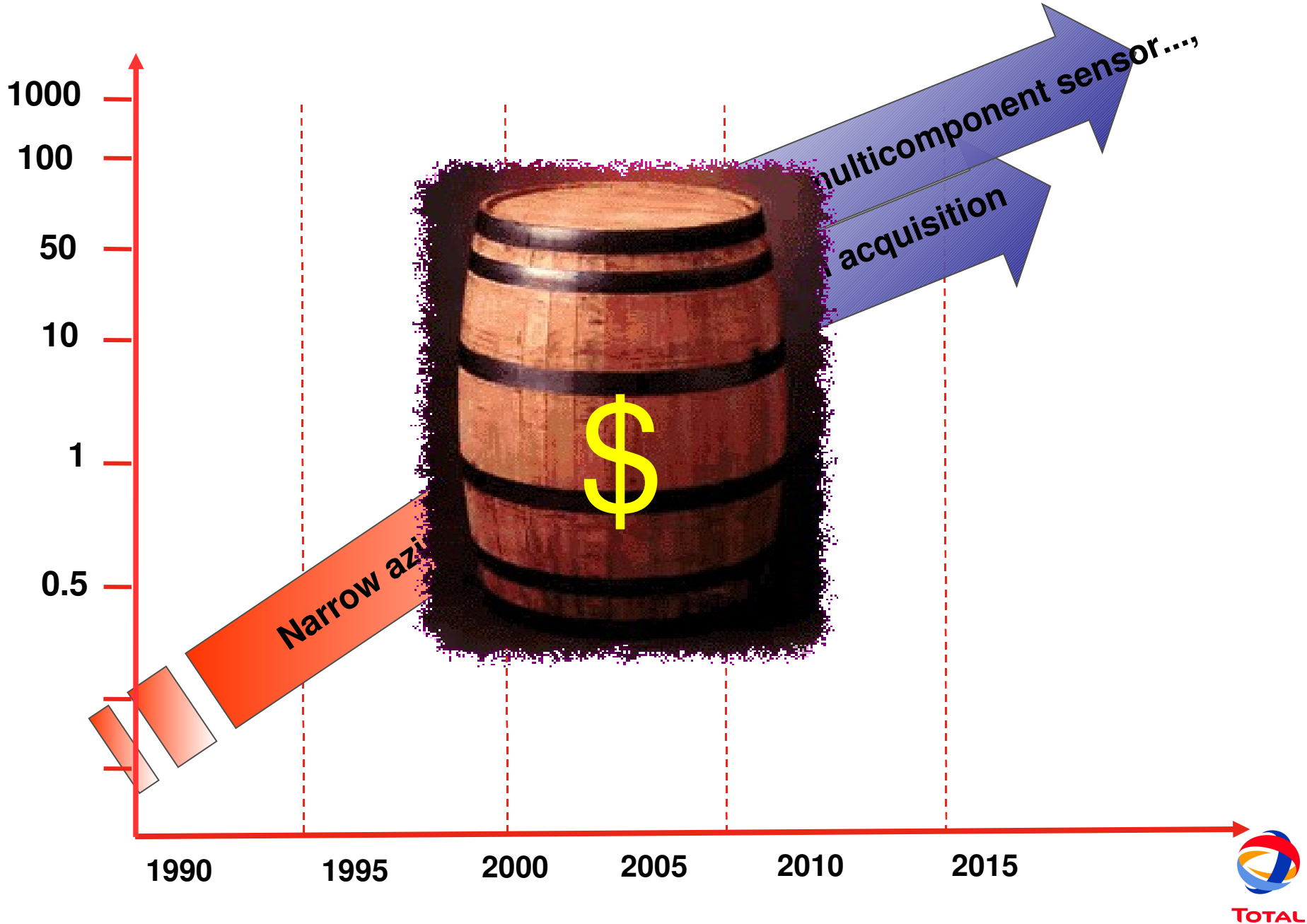


# Seismic survey acquisition size evolution

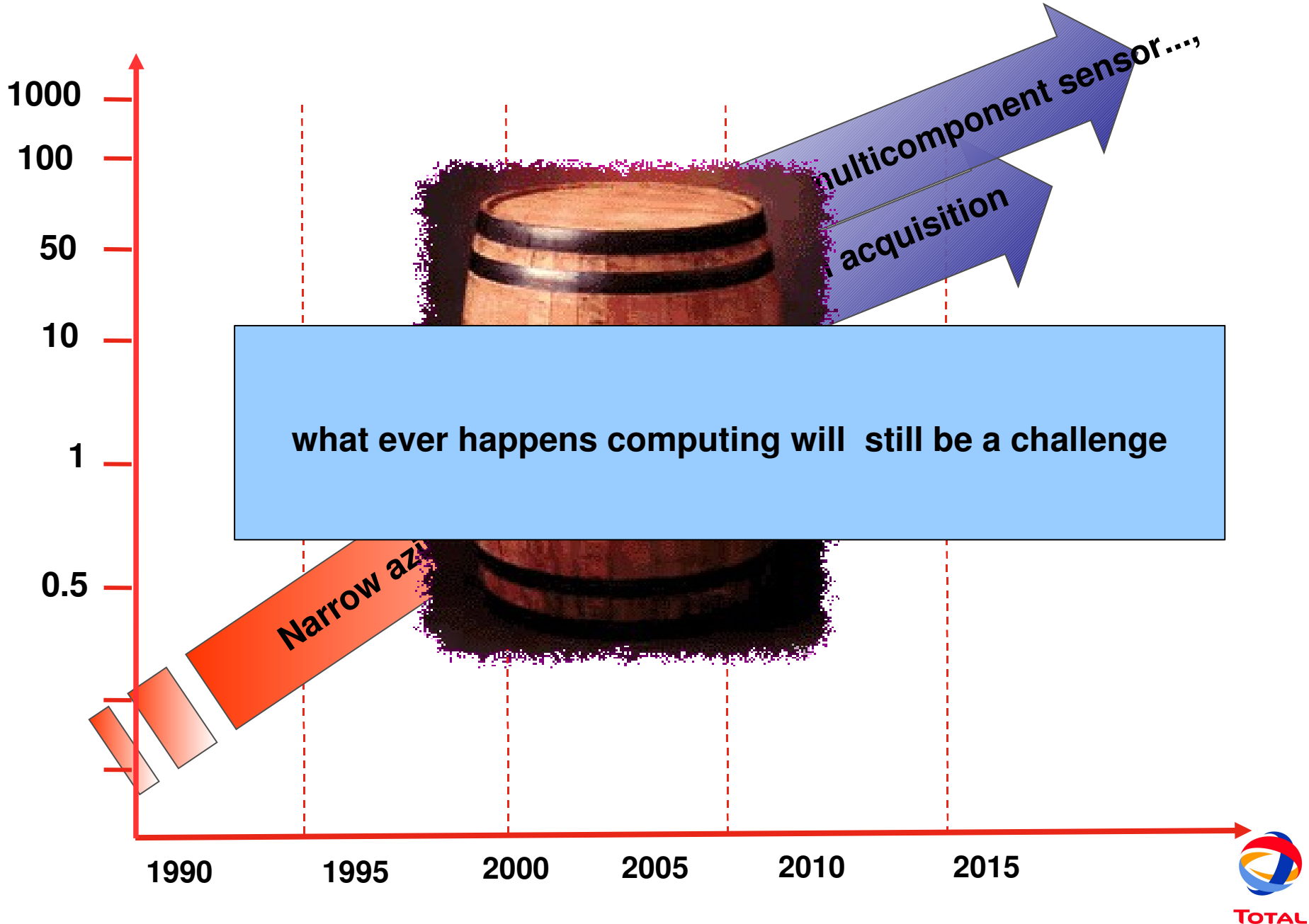


TOTAL

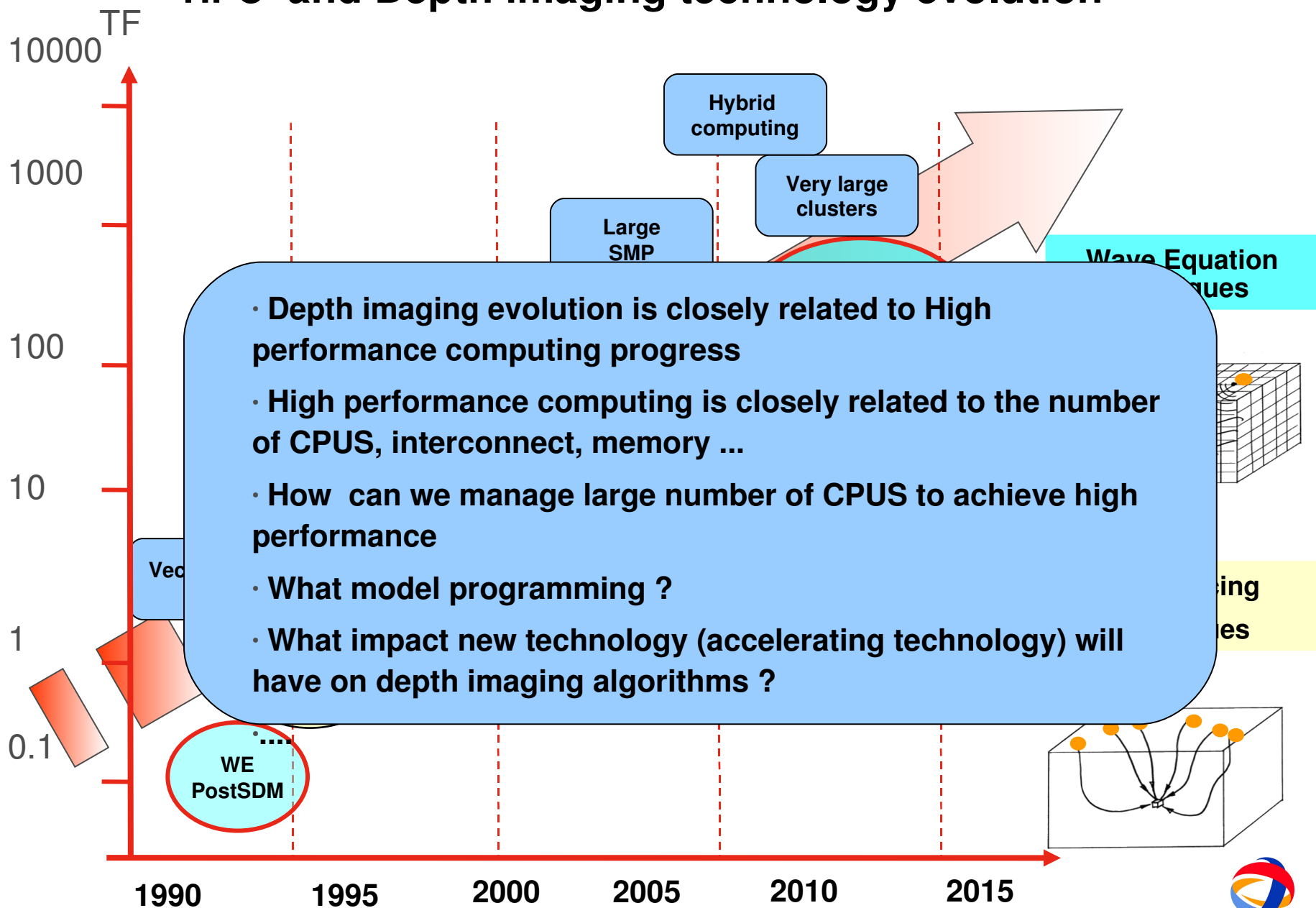
# Seismic survey acquisition size evolution



# Seismic survey acquisition size evolution



# HPC and Depth imaging technology evolution



# Outline

- ▶ introduction to seismic depth imaging
- ▶ Seismic exploration Challenges
- ▶ **Looking for petascale and more ...**
- ▶ Example: Reverse time Migration
- ▶ Conclusions



## **Looking for Petascale and more...**

- MPP technology
- Accelerating technology

# Looking for Peta-Scale and more...

- 1 Peta-flops ~ 100000 cores , 10 Peta-flops ~ 1000000 cores !!!



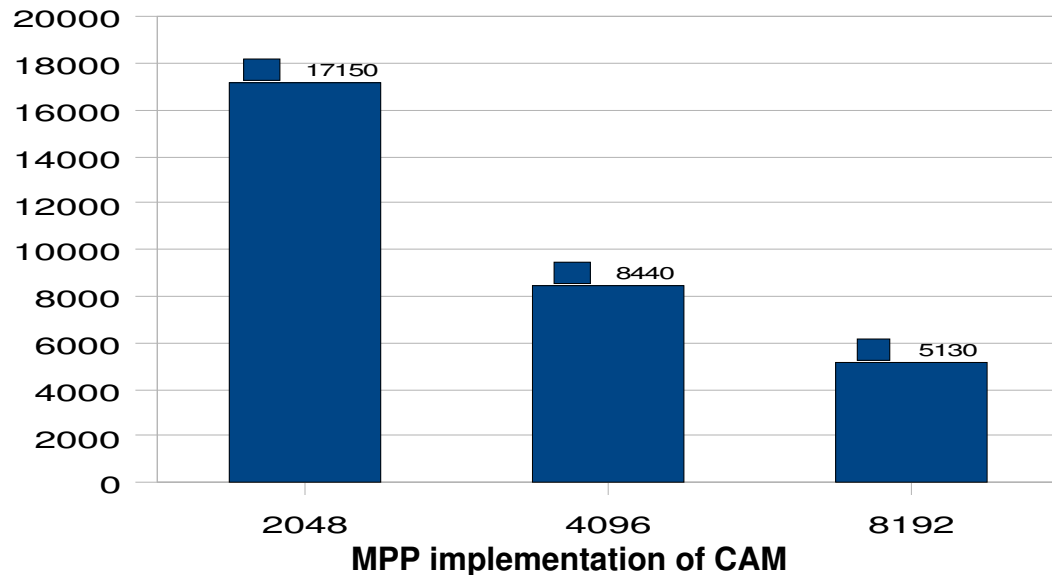
12500 to 125000 nodes !!!

- Q: How to manage so huge number of compute nodes:
  - Heat dissipation
  - MTBF ( Mean Time Between Failure)
  - Scalability: interconnect, OS, I/O...
- “Reasonable” number of compute nodes in terms of Heat dissipation, MTBF,scalability... (1000, 2000, 10000 ?, technology dependant)
- High performance is a trade-off between:
  - the number of compute node
  - the computing power of a compute node
  - Algorithm definition and design
- 2 Solutions:
  - High efficiency interconnect capabilities: MPP
  - High efficiency node computation capabilities: Accelerating technology

# Looking for Peta-Scale and more: MPP technology





## MPP technology

- Get access to Huge number of CPUs
- Scalable interconnect
- Easy to manage, more reliable than clusters
- Take advantage of the fast interconnect:
  - Programming model, data workflow...
  - Efficient numerical implementations,
  - Flexible implementations,
  - Use the fast interconnect as an extra dimension to reduce disk usage...
- Programming model: efficient and well known: MPI ( + OPENMP ), SHMEM
- Compilers extensions: Co-Array Fortran (CAF), Unified parallel C language (UPC) ????



# Looking for Peta-Scale and more: Mass

## Multi core and accelerating technology

- Increasing performances  Perform more operations per clock
  - WE NEED: Tera Flops and more per CPU
  - Solutions:
    - Increase the frequency rate
    - Increase the number of functional units
      -  Technological limitations: heat dissipation, data synch, physics limitations
  - Duplicate the number of computational units inside the same DIE:
    - Multi core technology: Dual core, Quad core ...  massive multi core ?
    - “Accelerating technology”: FPGA, GPGU, CELL
    - Vector Technology
-  Increase parallelism within the CPU

# Looking for Peta-Scale and more: Accelerating technology

## ■ Multi-core solution:

- Moore's law
- large number of cores = large performances > TF per socket
- SMP in one socket: 4, 8,...,80, 128... ?
- Q:
  - what programming model ?
  - data locality and placement,
  - data access,
  - do we have to specialize cores ?

## ■ Accelerating technology:FPGA,CELL,GPGPU..

- Integrating specialized hardware into seismic application to speed up application
- Technology is evolving very fast and still respects Moore's law
- Q: What programming model ?
  - none of the different technologies provides ( until now ) a general and standard programming environment
  - what is the best integration host-accelerator communication ?



# Looking for Peta-Scale and more...

## 3 programming directions

### ▪ Libraries:

- Design efficient libraries( FFTS, Trigonometric functions, stencils convolution..)
- Use these functions from your C or Fortran Code
- Easy to modify the original code to get advantage of these libraries ( work on vector or Matrix)
- OK in principle for GPGPU and CELL technology
- NOT efficient for FPGA technology

### ▪ Low level language programming,

- Very difficult to get good performances if not expert
- Poor flexibility

### ▪ High level language

- Better control of the evolution of the program
- More flexible than using libraries
- Requires very high bandwidth between host and accelerator card
- No standard language or programming model

## • Algorithm design has to respect High Performance Computing Rules

# Looking for Peta-Scale and more...

## ► Research orientations

- Stop working on FPGA
- Continue working on GPGPU and CELL:
  - CELL:
    - multi core technology: SPE (powerpc) +PPE (simple SIMD units)
    - PPE program with f90,C (PPE can be seen as the host)
    - SPE programming can be: independent task scheduled on each unit, pipelined parallelism or data parallelism
    - model programming is evolving fast, intrinsic, compiling directives (OMP)..
    - PPE, SPE communication via DMA ( stream computing)
    - Still on going technology
    - need more time to be really efficient, will be it still competitive compare to multi cores, GPGPU or vector technology?
  - GPGPU:
    - Interesting solution
    - Evolving very fast , double precision, larger local memory
    - “SIMD” like model programming
    - Asynchronous communications between host and GPGU (coming soon)
    - Still remains the question of interconnection between host and GPGPU

## ► 2 main directions: model programming and hardware configuration

# Looking for Peta-Scale and more...

## ► Research orientations

### ▪ Model programming

- Standard language: F90, C
- An OpenMP-like extension: HMPP
  - Express task parallelism whose codelets are executed/distributed over the stream cores:
    - Homogeneous: pthread
    - Heterogenous: CTM, CUDA, Mitrion, ...
  - Define a single interface between application and runtime
    - Data transfers, synchronization, execution
    - A computation can be split over different HW cores
  - Define a standardized HW specific interface between runtime and codelet implementation

## ► Develop “real application” on GPGPU technology based on this model programming

## ► Establish close relations with vendors

# HMPP : Hybrid Multicore Parallel Programming

- **Codelet declaration**

```
!$hmpp all codelet, target GPU, inout=phi, inout=u, inout=v, inout=partialu, inout=partialv,  
  subroutine rtm_update_all_layer_2d_f90( ...)
```

- **data transfer management:**

```
  subroutine rtm_solve_fwd_2d  
!$hmpp all advancedload, &  
!$hmpp all calleeArg=n1, const, &  
!$hmpp all calleeArg=v, const, &  
.....
```

data transfert to the  
GPGPU in one shot

- **kernel execution:**

```
!$hmpp all callsite, &  
!$hmpp all advancedload:calleeArg=n1, &  
!$hmpp all advancedload:calleeArg=v, &  
...  
  call rtm_update_all_layer_2d_f90(...)  
!$hmpp all delegatedstore, calleeArg=partialu
```

- **Introduce OMP like directives,**
- **Automatic code generation for specific hardware ( C: Nvidia, ATI , Fortran coming soon)**
- **Dynamic execution of accelerated kernels.**

# Looking for Peta-Scale and more...

## ► Research orientation

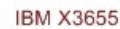
### ▪ Hardware configuration

- Define the best hardware configuration: Compute node definition, interconnect....
- Test different solutions: NVIDIA ATI,
- Perform tests on realistic Hardware configuration
- Verify that the general programming model ( domain decomposition over the nodes, multi parallelism level) is still valid
- Host-GPGU strategy ?
  - HOST 1 GPGPU
  - HOST 2 GPGPU
  - Multi GPGU implmentation ....



Cluster Caïman  
10 Tflops

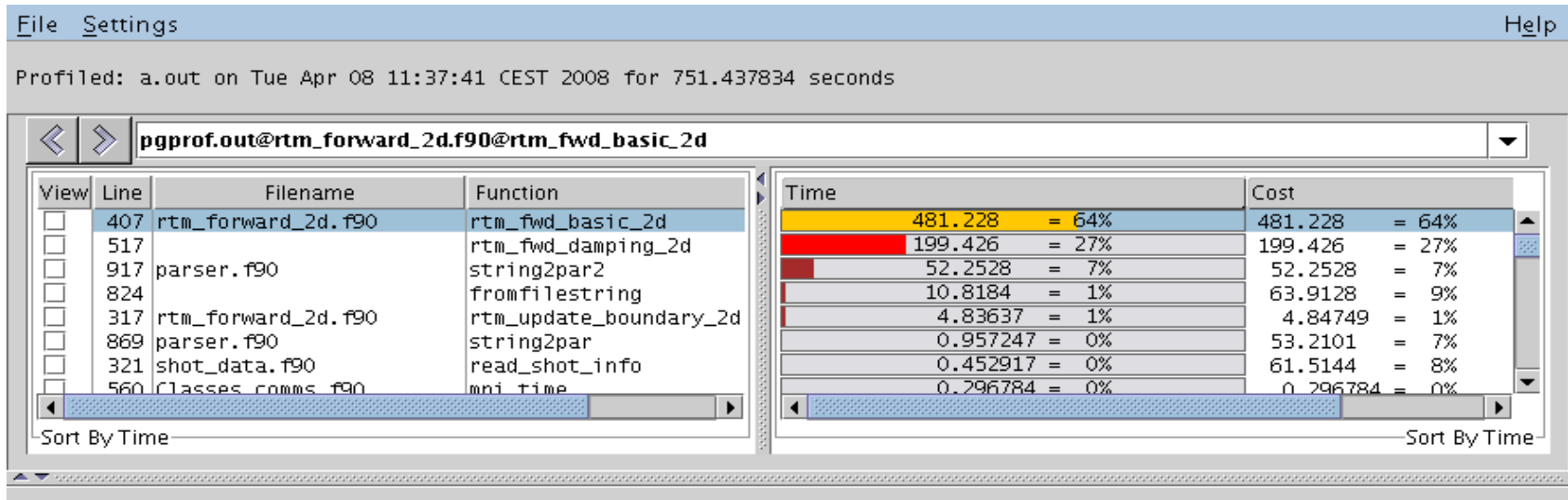
PCI-Express



# Outline

- ▶ introduction to seismic depth imaging
- ▶ Seismic exploration Challenges
- ▶ Looking for petascale and more ...
- ▶ **Example: Reverse time Migration**
- ▶ Conclusions

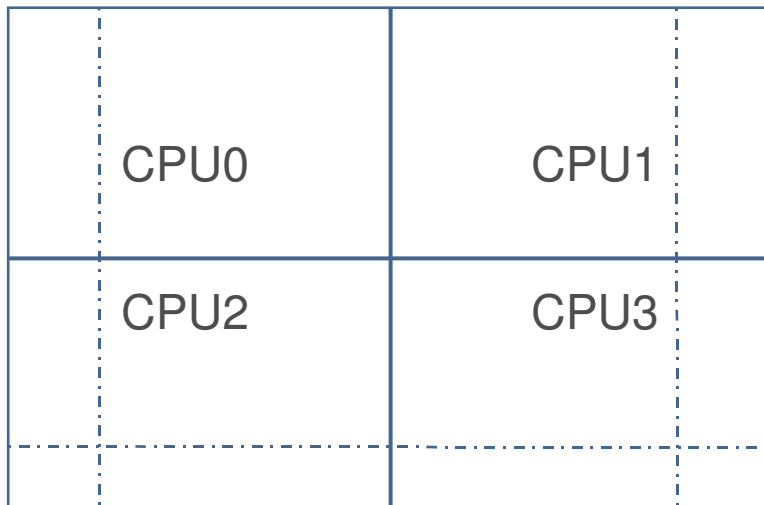
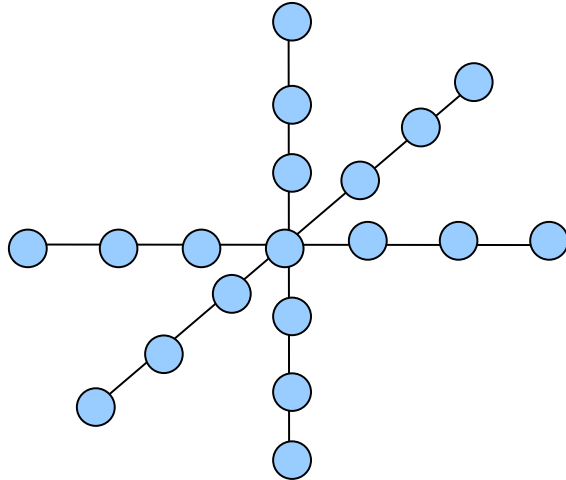
# RTM: PreSDM leading edge technology



**91% of cpu time is spent on solving acoustic wave equation based on explicit time-space finite difference discretisation**

# implementation

## 1. general algorithm ( host implementation)

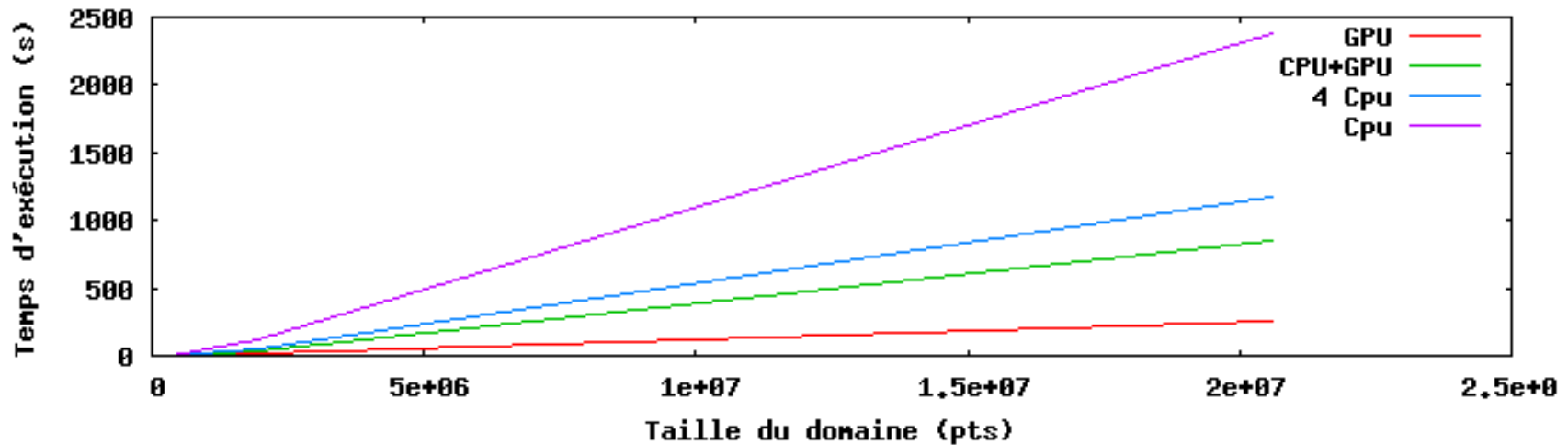


**general domain decomposition implementation**

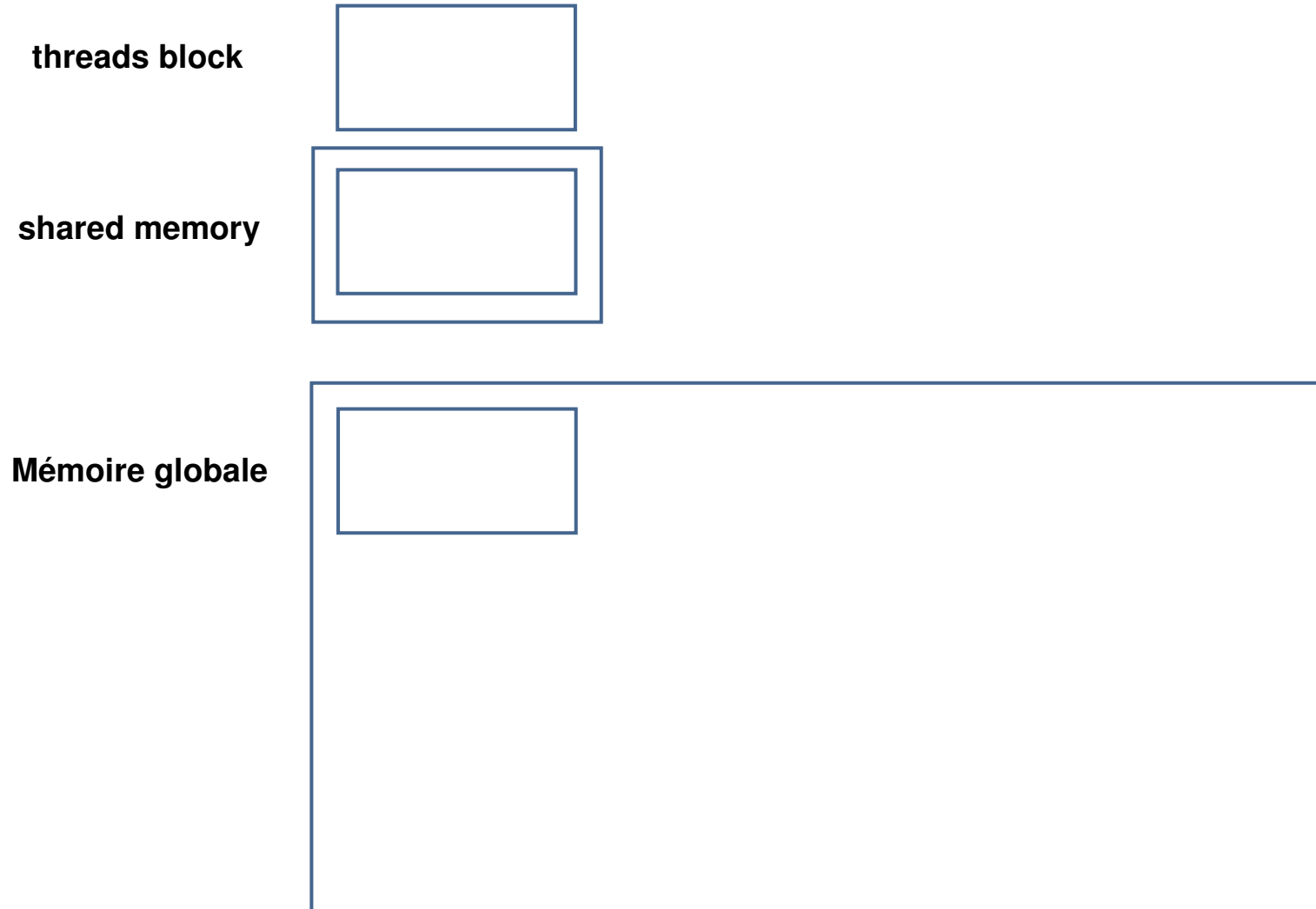
# Performances (results still in progress)

- Dual Core AMD Opteron Processor 280 - 16Go
- NVIDIA Quadro FX4600 (12 Multiproc- 768Mo)

	grid size	CPU	4CPUs	CPU+GPU	GPU
model1 (688*489, 1346 time steps)	336 432	7,29	6,18	4,78	2,9
model2 (2421*811, 4131 time steps)	1 963 431	118	52,22	29,37	14,9
model3 (4720*4361, 9011 time steps)	20 583 920	2384	1169	846,2	258

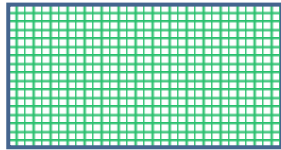


# FD kernel implementation



# FD kernel implementation

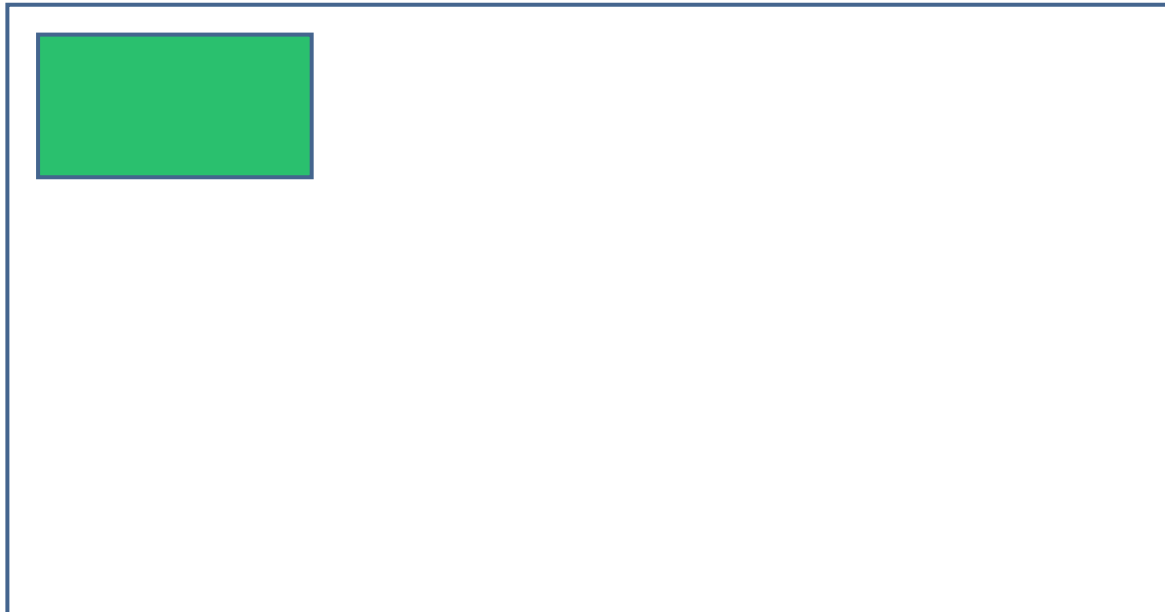
threads block



shared memory



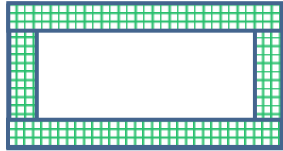
global memory



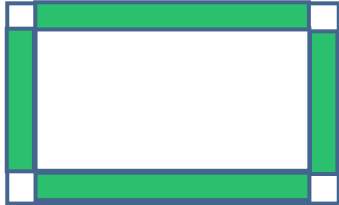


# FD kernel implementation

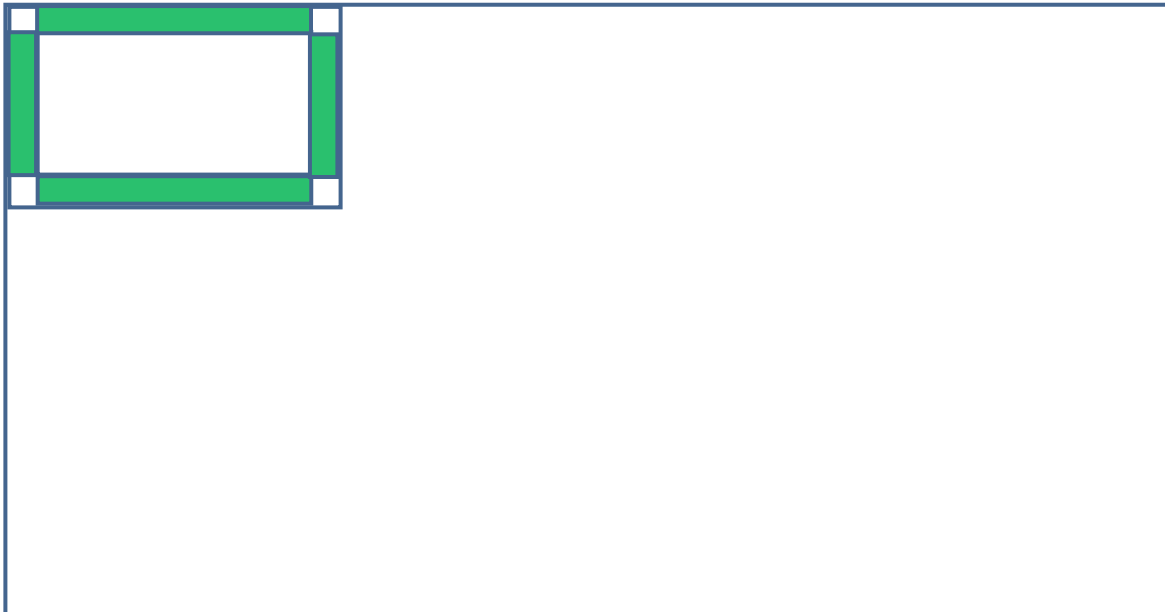
threads block



Shared memory



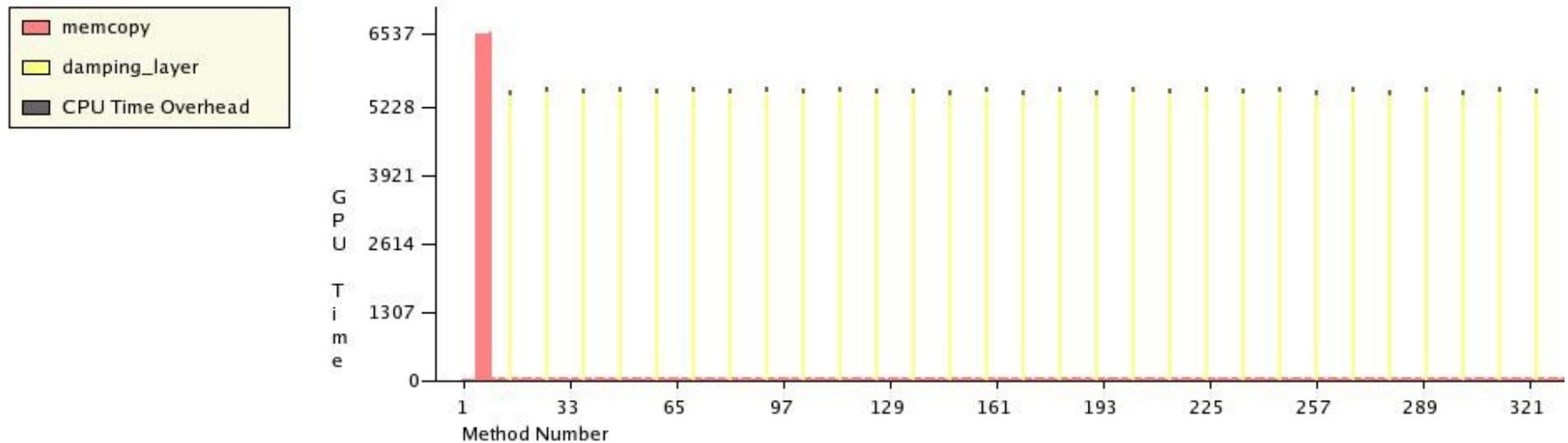
global memory



# Minimize Host-GPU communication

► First iteration: send all the data to GPGPU

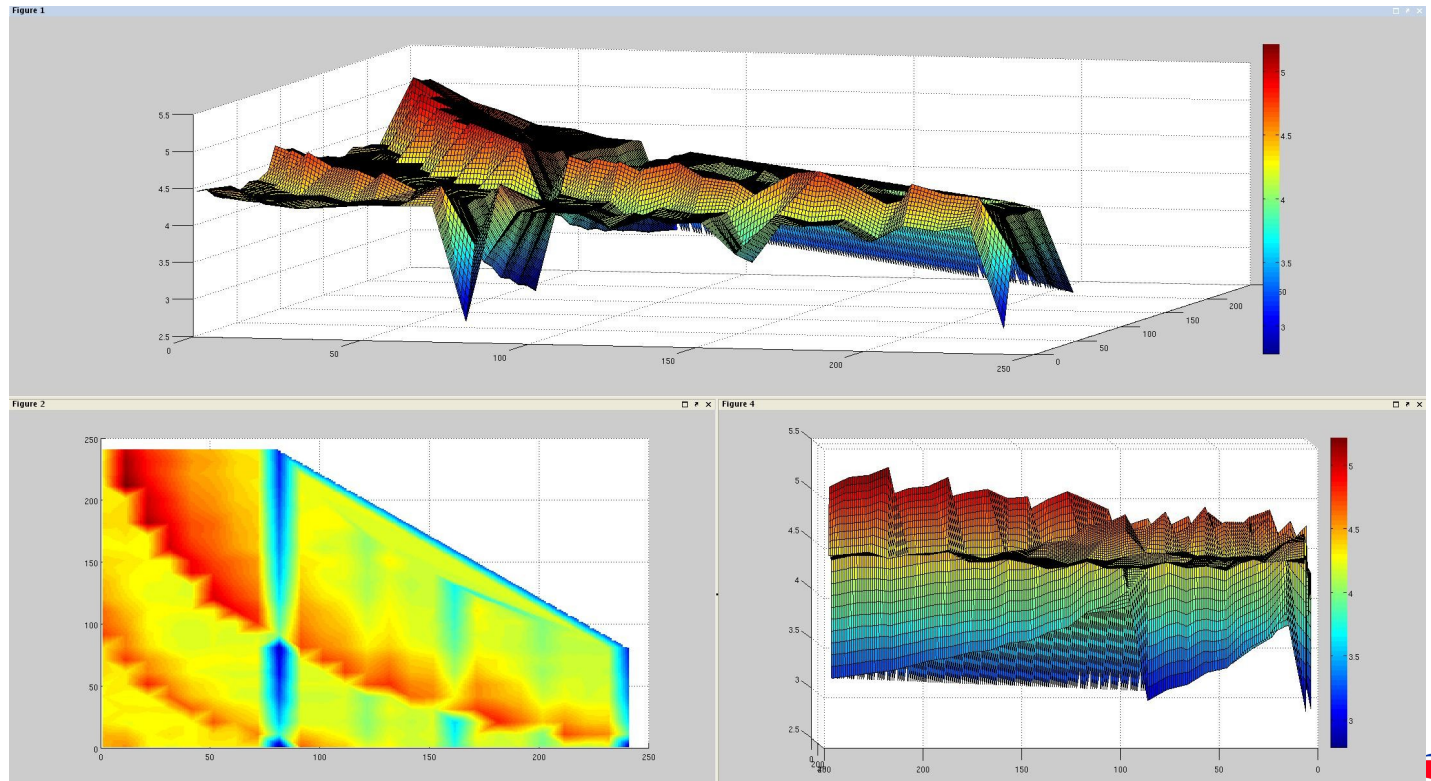
► all other iteration exchange only few data



# Block size optimization ?

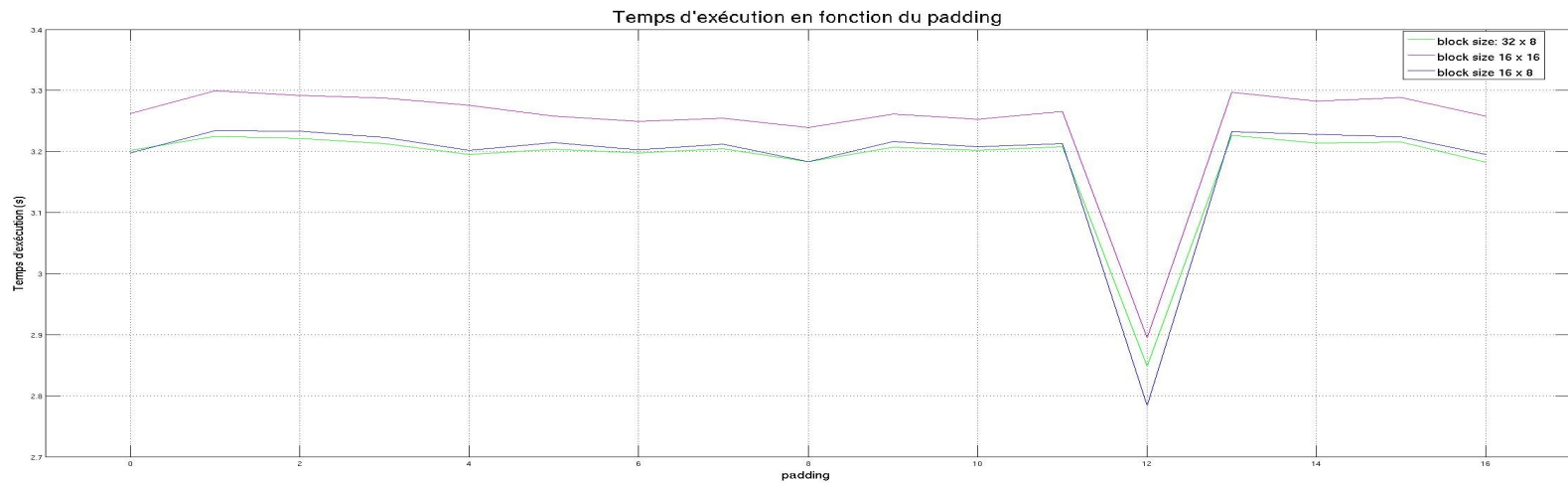
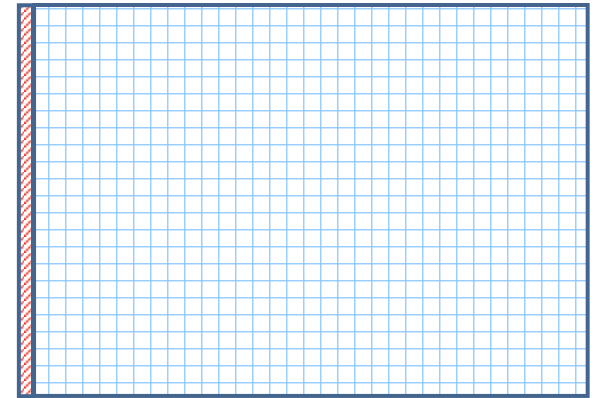
- how to choose the optimal block size ?

- Only choice: testing



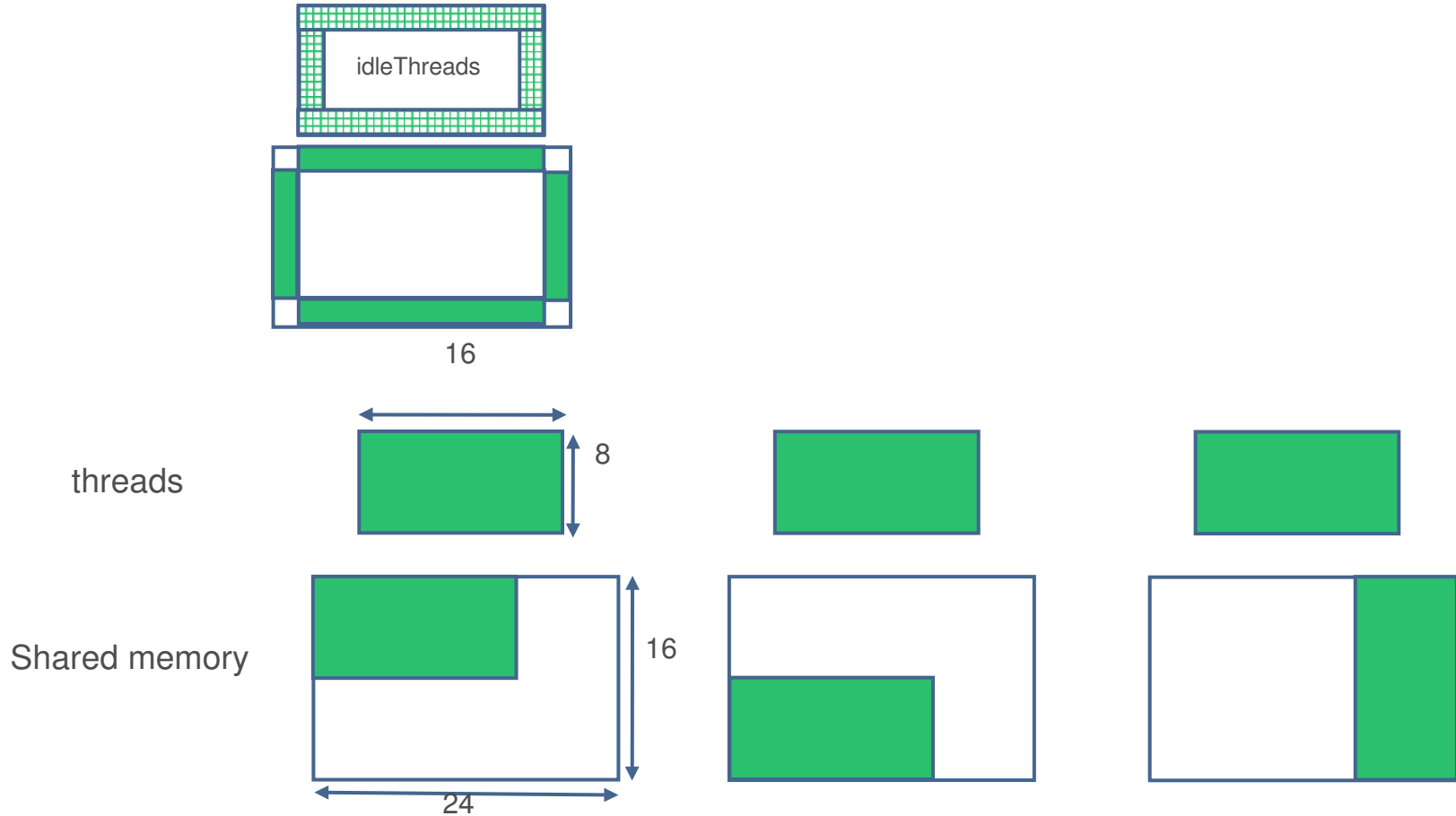
# Memory padding

method=[ damping\_layer ] gputime=[ 4047.520 ] cputime=[ 4101.000 ] occupancy=[ 0.667 ] gst\_incoherent=[ 737696 ]



method=[ damping\_layer ] gputime=[3404.984] cputime=[3466.478] occupancy=[ 0.667 ] gst\_incoherent=[ 0 ]

# Optimization: avoid branch statements



# Resolve memory bank conflicts

Same active threads access the same  
memory bank

threads



shared memory



method=[damping\_layer] gputime=[ 3405.984 ] cputime=[ 3461.000 ] occupancy=[ 0.667 ] warp\_serialize=[ **30286** ]

Solution: padding

threads



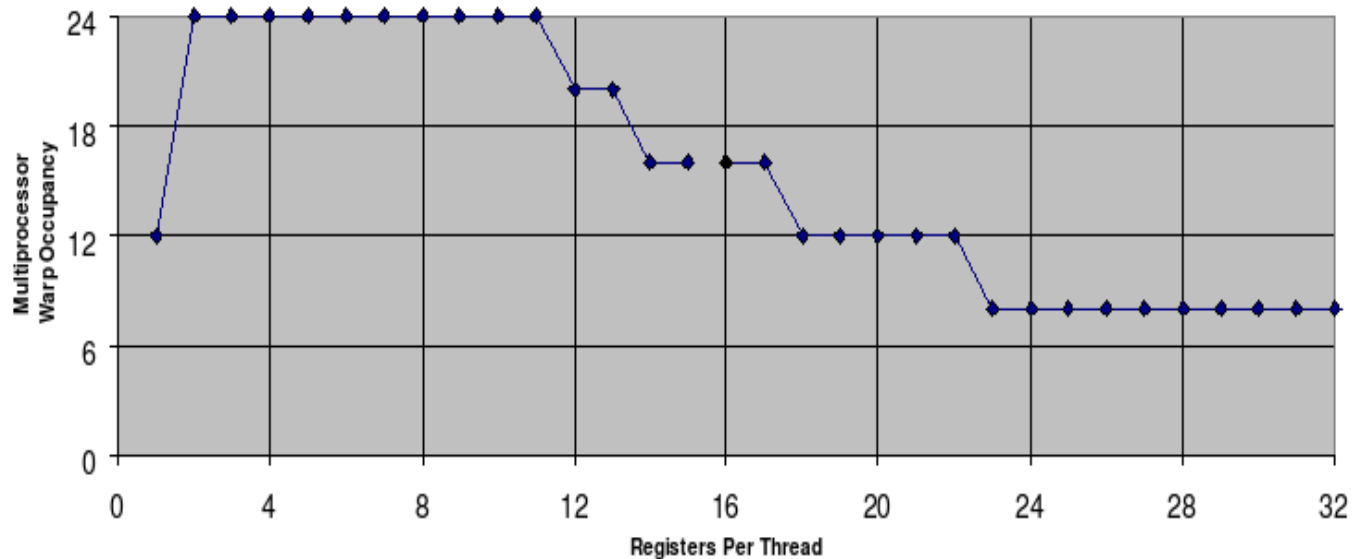
shared memory



method=[damping\_layer] gputime=[ 3367.104 ] cputime=[ 3420.000 ] occupancy=[ 0.667 ] warp\_serialize=[ **0** ]

# Increase the multi processor warp occupancy

Maximum occupancy is limited by the maximum number of registers per thread (16)



method=[ damping\_layer ] gputime=[ 3425.056 ] cputime=[ 3476.000 ] occupancy=[ **0.667** ]

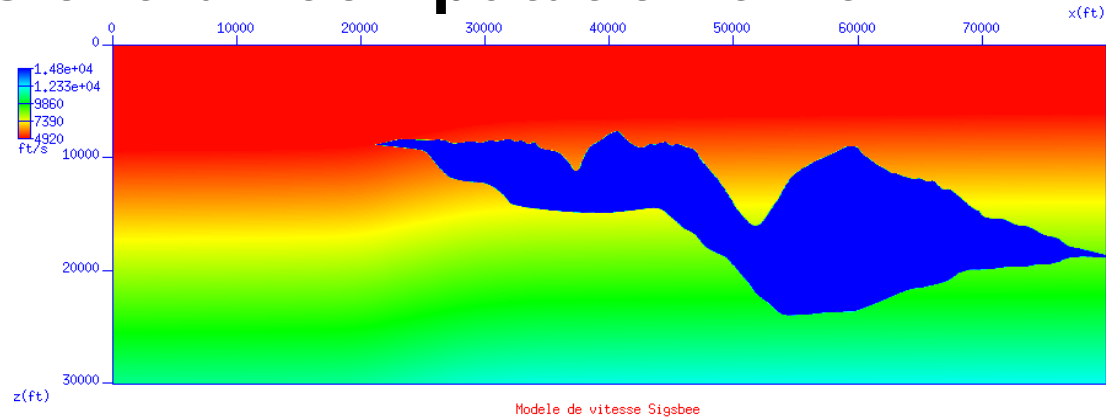
EX: decompose kernel in two parts.

method=[ damping\_layer ] gputime=[ 2294.816 ] cputime=[ 2351.000 ] occupancy=[ **1.000** ]

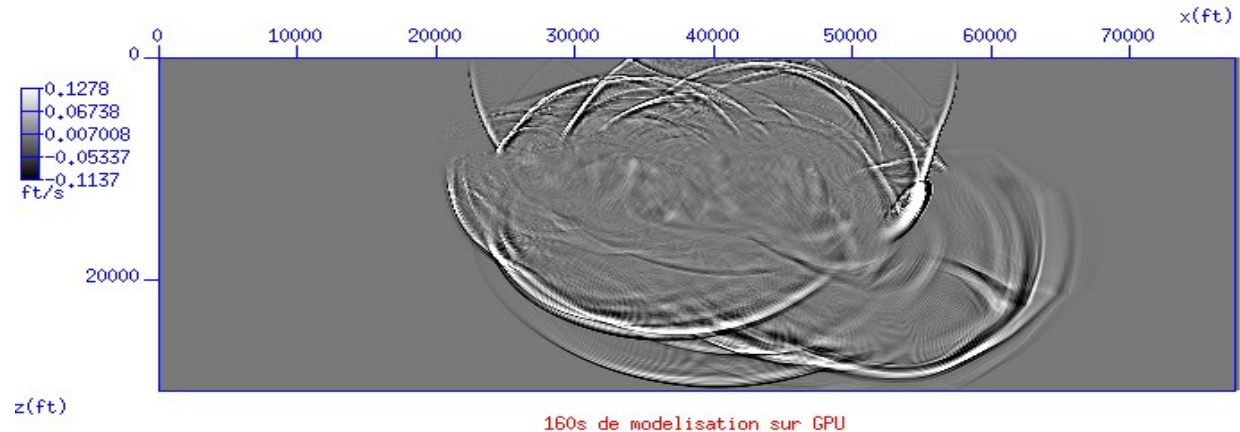
method=[ damping\_layer\_update\_pml ] gputime=[ 1833.824 ] cputime=[ 1887.000 ] occupancy=[ **1.000** ]

# 160s overall computation time

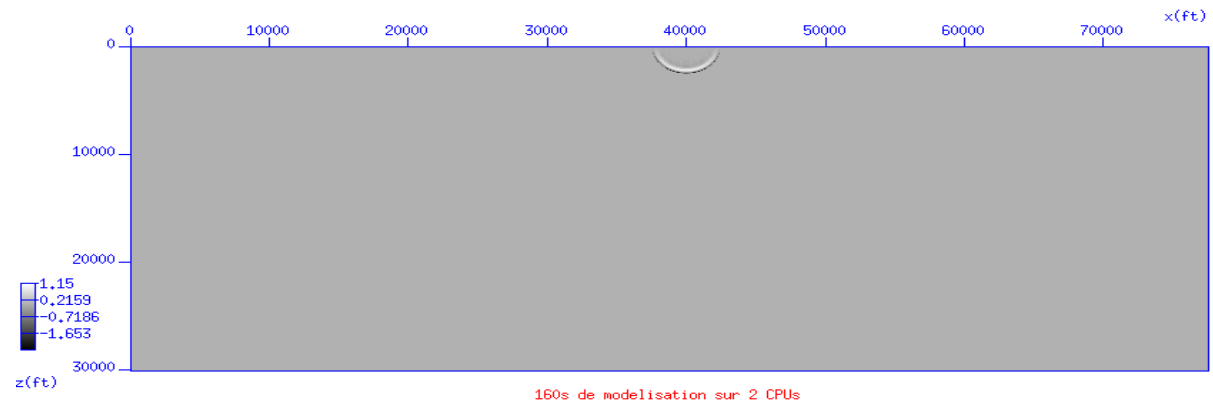
- Velocity model:  
6400\*1500 grid  
size



- GPU: 3.6s  
(12808 time steps)



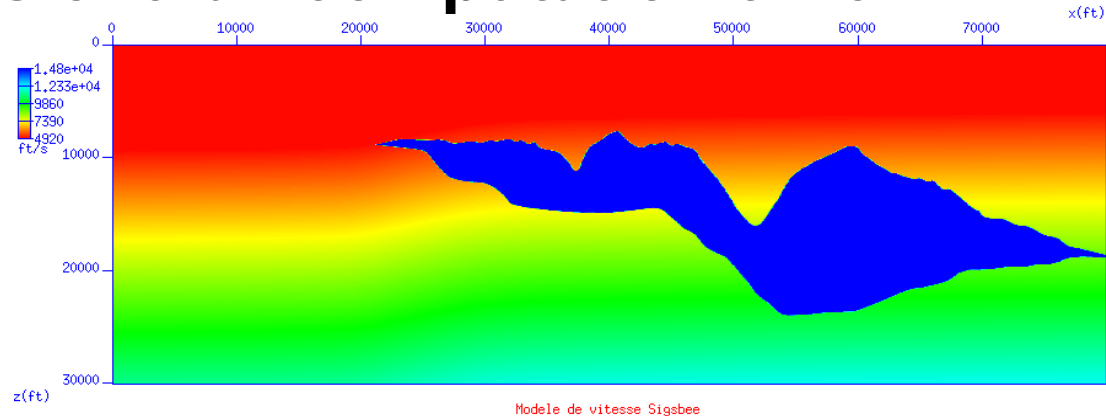
- 2 CPUs: 0.5s  
(1778 time steps).



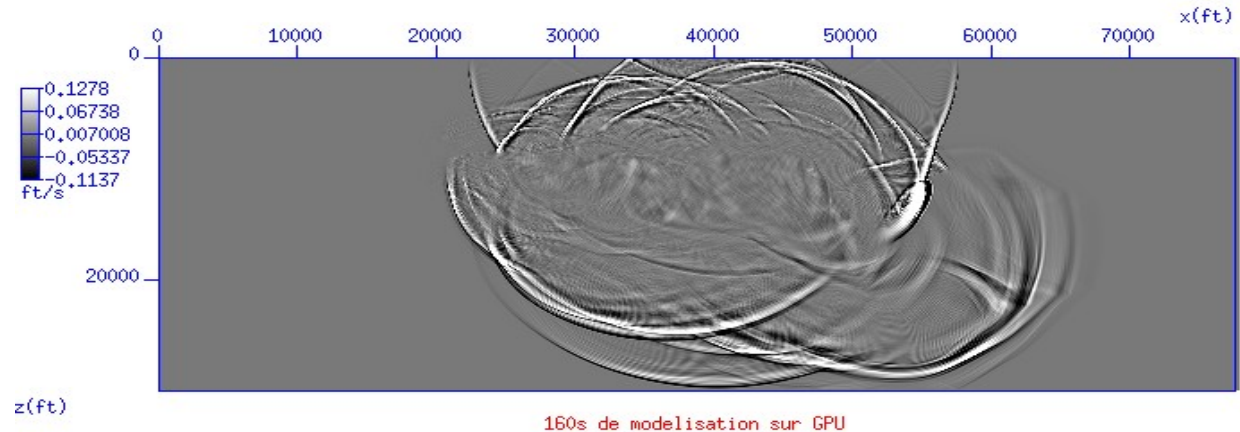


# 160s overall computation time

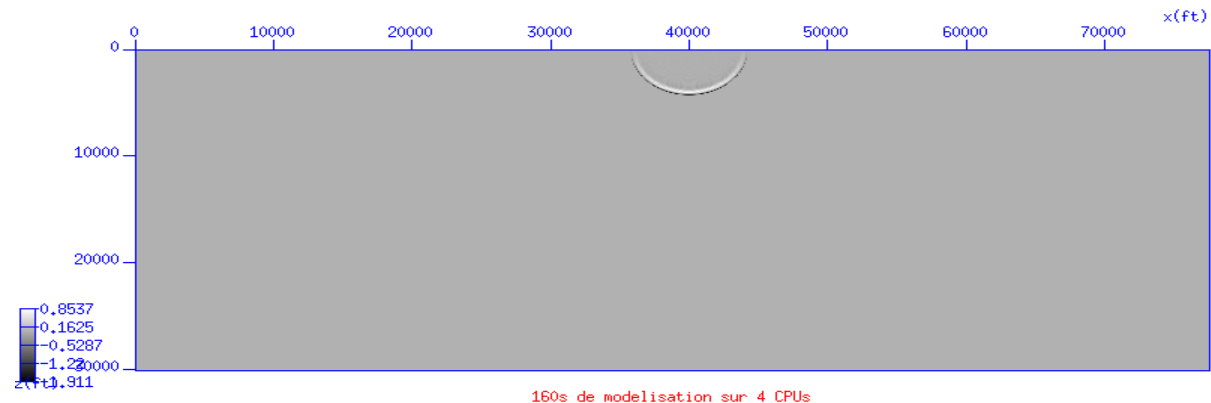
- Velocity model:  
6400\*1500 grid  
size



- GPU: 3.6s  
(12808 time steps)

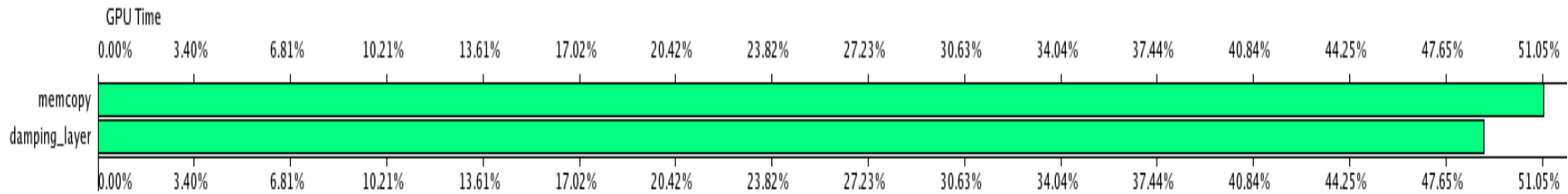


- 4 CPUs: 0.85s  
(3054 time step).



# actual limitations

## Host-GPGPU communication



## improvement ?

- Host-GPGPU asynchronous communication
- Higher bandwidth between host GPGPU
- more integrated solution
- global memory access ?

# Conclusion

- ▶ Depth imaging is very challenging
- ▶ Explore new directions to achieve high performance computing
- ▶ Accelerating technology is one way to be investigated
- ▶ GPGPU can be one way to accelerate
- ▶ but still progress need to be achieved in integration, communication...
- ▶ what % of theoretical peak performance can we obtain ?
- ▶ what impact accelerated compute node on interconnect and load balance..
- ▶ Test on large configuration ( summer 2008)