SOPHYA: a C++ class library for intensive data analysis and scientific computing

R. Ansari

Short introduction - NAOC - May 2013

- SOPHYA development history
  - Choice of language : from fortran to Java
- SOPHYA packages and main features
- Performance issues
- Interactive data analysis and development: **piapp**

SOPHYA

development history

SOPHYA

- We started the development of a C++ class library,in 1999-2000, based on our experience in intensive astrophysical data/image processing EROS/PEIDA++
- Relative “ease” of use for scientists, while maintaining performances comparable with C/Fortran
- Developed at LAL/IN2P3-Orsay and DAPNIA-CEA in France, initially as one of the base components of ESA-Planck CMB mission, with contributions from many scientists :
  - *E. Aubourg, S. Henrot-Versillé, A. Kim, G. Lemeur, C. Magneville, B. Revenu, F. Touze ...*

http://www.sophya.org

EROS and PEIDA++

- PEIDA++ : C++ class library used for EROS-2 microlensing project data processing and analysis
- PEIDA++ (developed in 1994-1998), used for the “on-line” data processing (in ESO/Chili) and off-line processing in France (CC-IN2P3,LAL,DAPNIA)
- Successful and Efficient processing of > 10 TBytes of image data, few TBytes of “light curves” in heterogeneous environmement

http://www.lal.in2p3.fr/recherche/eros/PeidaDoc/

Choice of a programming language and framework

- Not only a technical choice ...
  - Project constraints : Size, time scales, team organization and technical expertise ...
  - Power, flexibility, reliability/maturity of different solutions
  - Avoid multiplication of number of languages / tools used for a given project

For use in scientific computing

- Provide building blocks which can be assembled by scientists
- Selection or development of reliable class libraries covering common needs in the field
- Facilitate memory management
- Keep the object hierarchy simple, seen from the user point of vue
- Avoid as much as possible complexity generated by auxiliary services

SOPHYA : overview

- Data containers covering common needs in data processing and analysis
- Object I/O (persistence) : SOPHYA native (PPF), Exchange formats (FITS , text/ascii, and maybe HDF5 in the future)
- Thread safe Automatic memory management,
- Numerical algorithms : Encapsulate existing (C/Fortran) libraries whenever possible (FFT, Lapack, ...)

Memory management

- On the stack object creation : Memory freed automatically, correct destructor call during exception processing
  - Dynamic allocation should be made in constructors
  - Automatic memory management when classes are used
- On the stack object creation is much more efficient compared to dynamic allocation.
  - A factor ≈10 gain, when “small” objects are created in loops

Large objects : Reference sharing

- To avoid time consuming copies, a reference sharing mechanism must be implemented for objects representing large amount of data
- General rule in SOPHYA :
  - Copy constructor shares the data
  - Equal operator (=) performs a deep copy (duplicates the data)

SOPHYA modules (packages)

- **BaseTools** : Common services and classes (persistence, ref. sharing ...)
- **TArray** : Template multi-dimensional numerical arrays (max 5 D), Matrices, Vectors + basic operations on arrays
- **HiStats** : Data set manipulation, histograms, DataTables ...
- **SkyMap** : spherical (4 π / partial) pixel maps
- **NTools** : Basic numerical algorithms(FFT, linear and non linear fitting ...)
- **Samba, SkyT** : 3D geometry, Spherical harmonic transform, ...
- **SysTools**: Interface with OS services (threads, dynamic load, command interpreter ...)

SOPHYA : External library interface packages

- **FitsIOServer** : c-fitsio wrapper, object import/export in FITS format
- **IFFTW** : Discrete Fourier Transform through FFTW library, conforming to the FFTServerInterface defined in NTools
- **LinAlg** : Partial interface with LAPACK
- **XAstroPack** : Astronomical time/coordinate computation interface to XEphem library

- SophyaLib:
  - 350 files (.cc .h .c) , 95 kl, 2,7 MO
  - > 250 classes, including many templates classes
- SophyaExtLib
  - 60 files (.cc .h .c) , 19 kl, 600 kO
  - > 40 classes, including many templates classes
- PI/PIext (piapp)
  - 240 files (.cc .h .c) , 70 kl, 2 MO
  - 160 classes

http://www.sophya.org/PI

- for PI only , including some specific X11/Motif & MacOS
  - 170 files (.cc .h .c) , 40 kl, 1,3 MO
  - 100 classes

TArray module

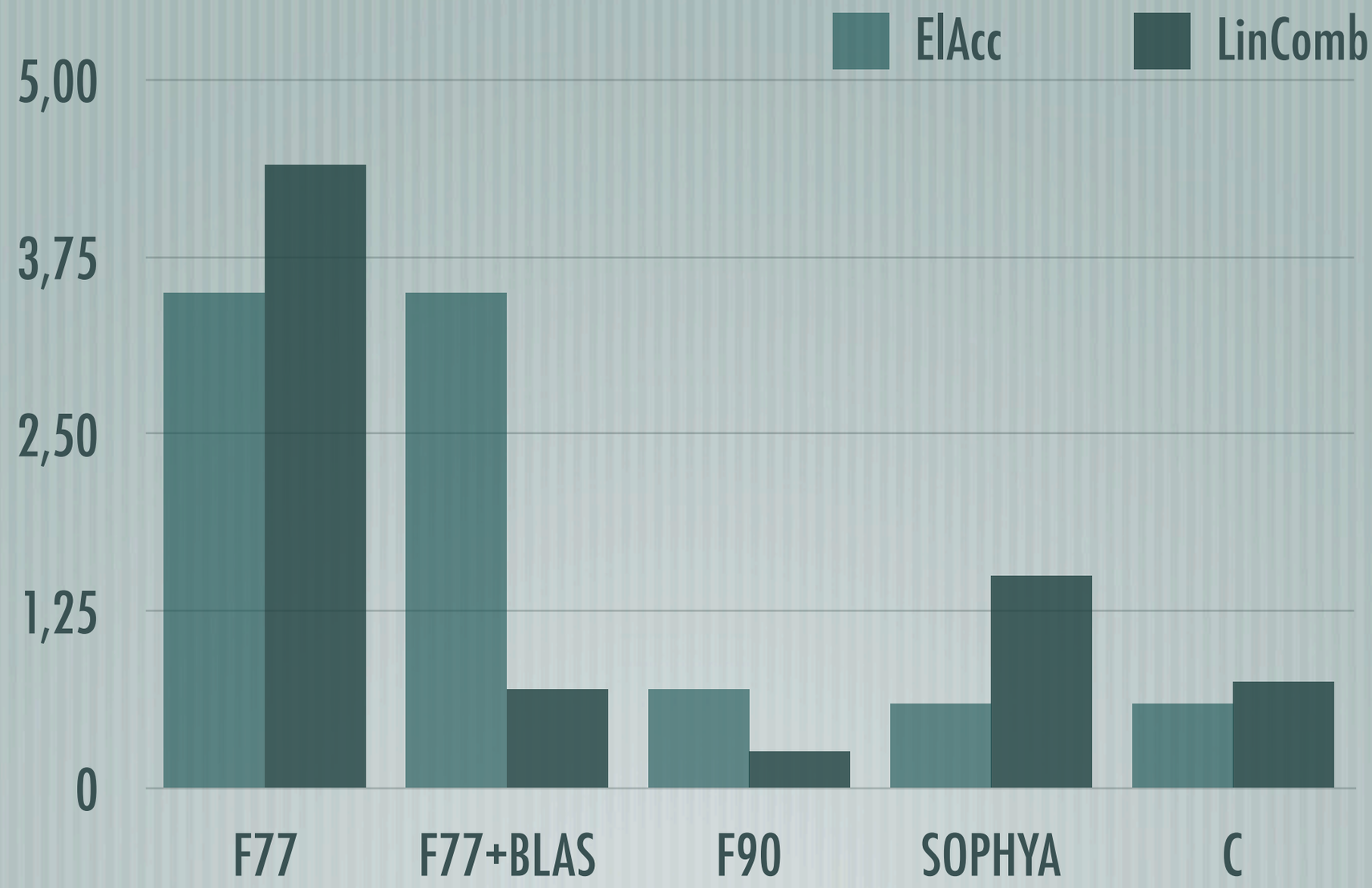
- Handles large, multi-dimensional, dense arrays (max NDim = 5)
- Array shape and size can be defined and changed at run time
- Transparent management of sub-arrays and slices
- Possibility to select memory organization (c-like/fortran-like) for matrices
- Arithmetic operations, conversion, and simple linear algebra
- Partial interface with LAPACK through LinAlg/ module

Example (1)

```
---------- Writing ---------
// We create a integer array SizeX=7, SizeY=5
TArray<int_4> ia(7,5);
// We fill it ....
// We extract a subarray ib
TArray<int_4> ib = ia(Range(0,3), Range(3,4), Range(0));
TArray<r_4> fc(10,8), fd(5,5);
// Fill the arrays ....
cout << " >>>>>> Writing in arrt.ppf <<<<<<< " << endl;
POutPersist pos("arrt.ppf");
pos << ia << ib;
pos << fc;
pos << PPFNameTag("FD") << fd ;
```

Performance issues

T1 on asc (TCPU in seconds)



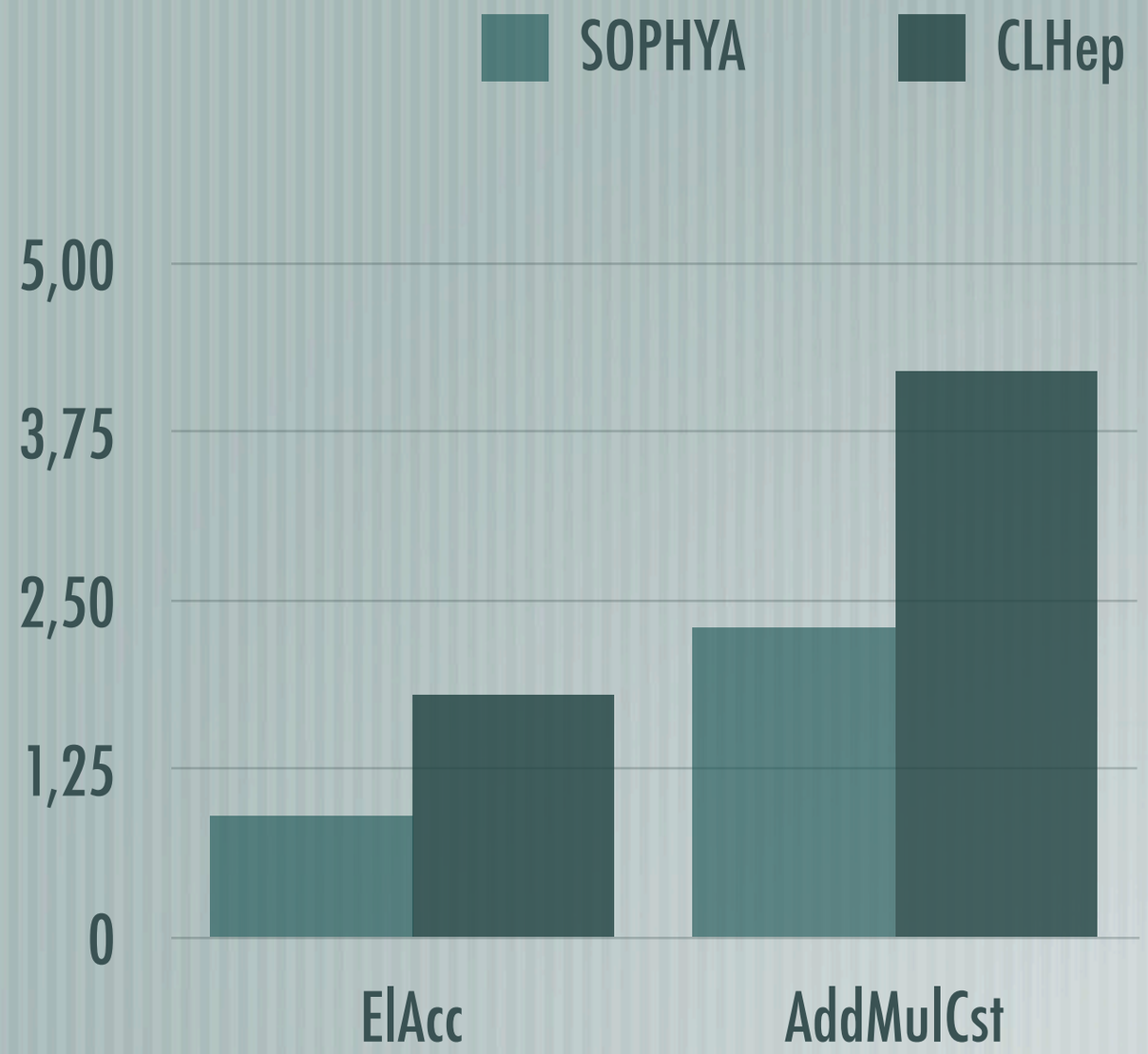
Comparison SOPHYA::TMatrix<r_8> / Fortran

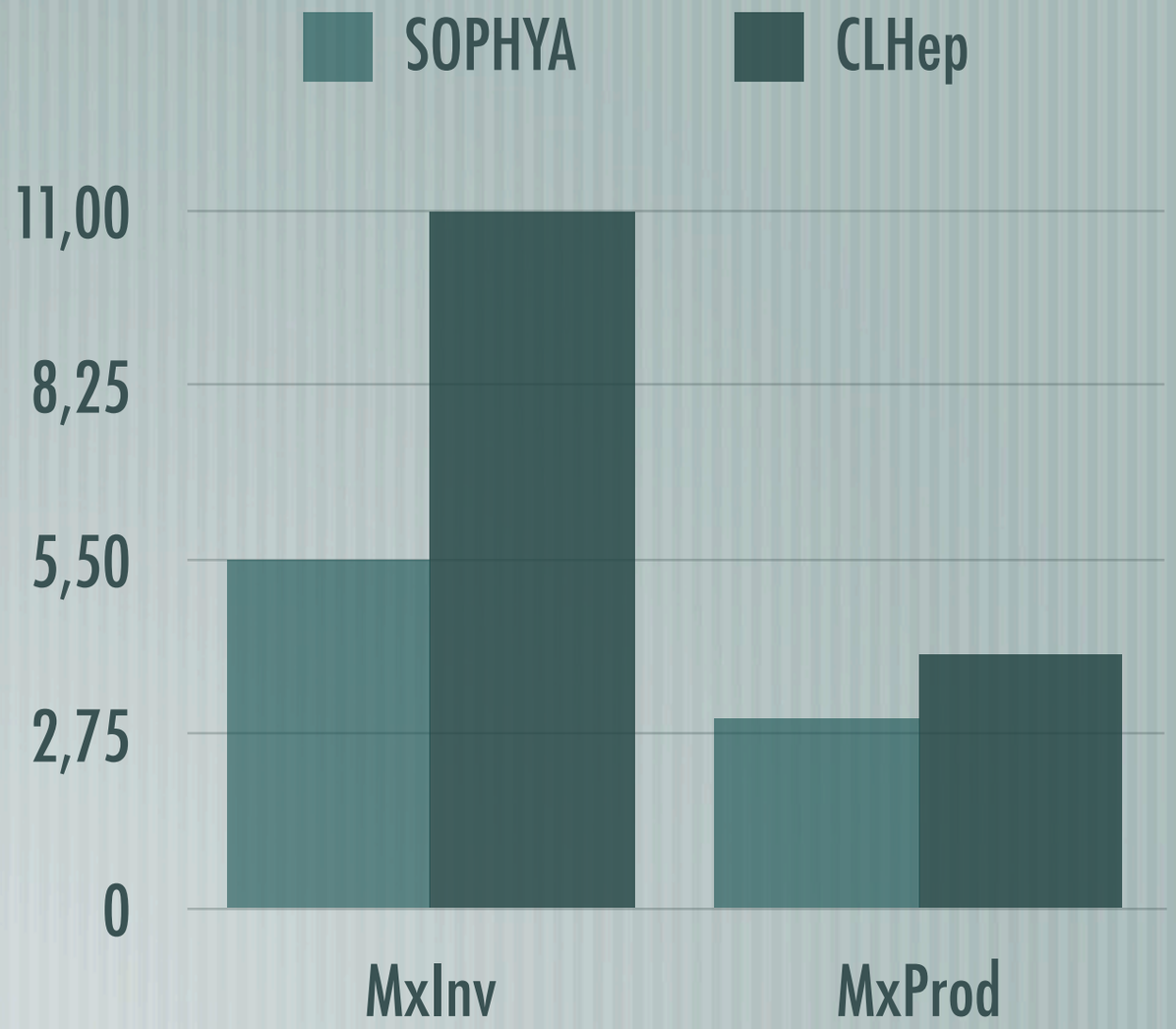
(cxx/f90/f77 -fast)

Performances - T3

- Tests: Inversion and matrix multiplication
  - MxInv : 1000x1000 matrix (double) inversion
  - MxProd : Multiplication of two 1000x1000 matrices (2 GFLOP)
- Inversion through SOPHYA::LapackServer<T> class
- Matrix multiply SOPHYA::TMatrix<T>, mx3 = mx1*mx2;

SOPHYA vs CLHep : T1 , T3 on ccali

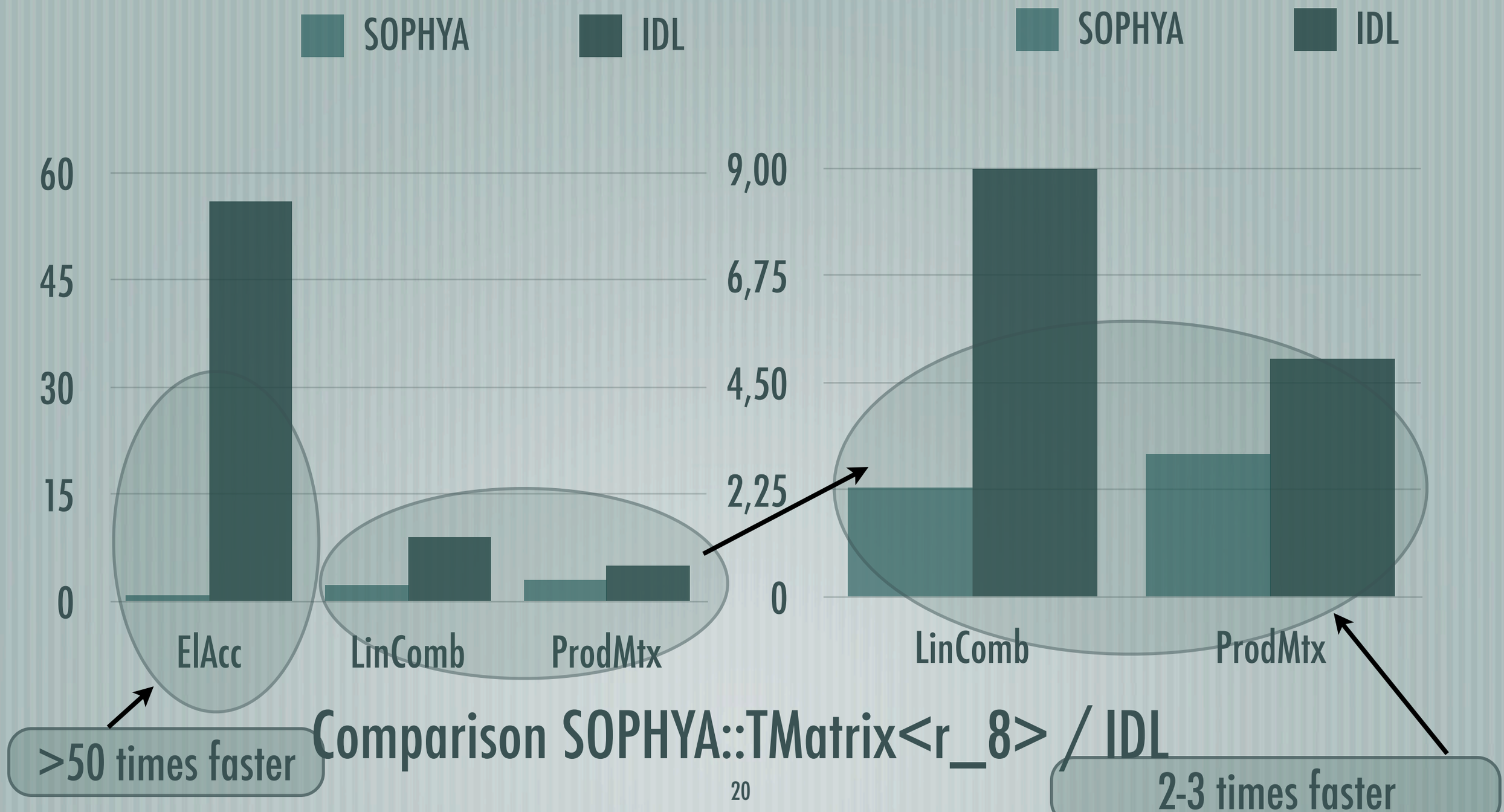




Comparison SOPHYA::TMatrix<r_8> / CLHepMatrix

SOPHYA vs IDL (ccali)

C++ vs specialized array oriented interpreted language



Comparison SOPHYA::TMatrix<r_8> / IDL

Interactive data analysis

(s)piapp

Integrated technical and scientific computing frameworks

- Interpreted language, and integrated visualization/technical computing framework (array oriented) - few examples :
  - Matlab (http://www.mathworks.com/)
  - Scilab (http://scilabsoft.inria.fr/)
  - IDL (http://www.rsinc.com/)
- Computer algebra systems :
  - Mathematica (http://www.wolfram.com/)
  - Maple (http://www.maplesoft.com/)

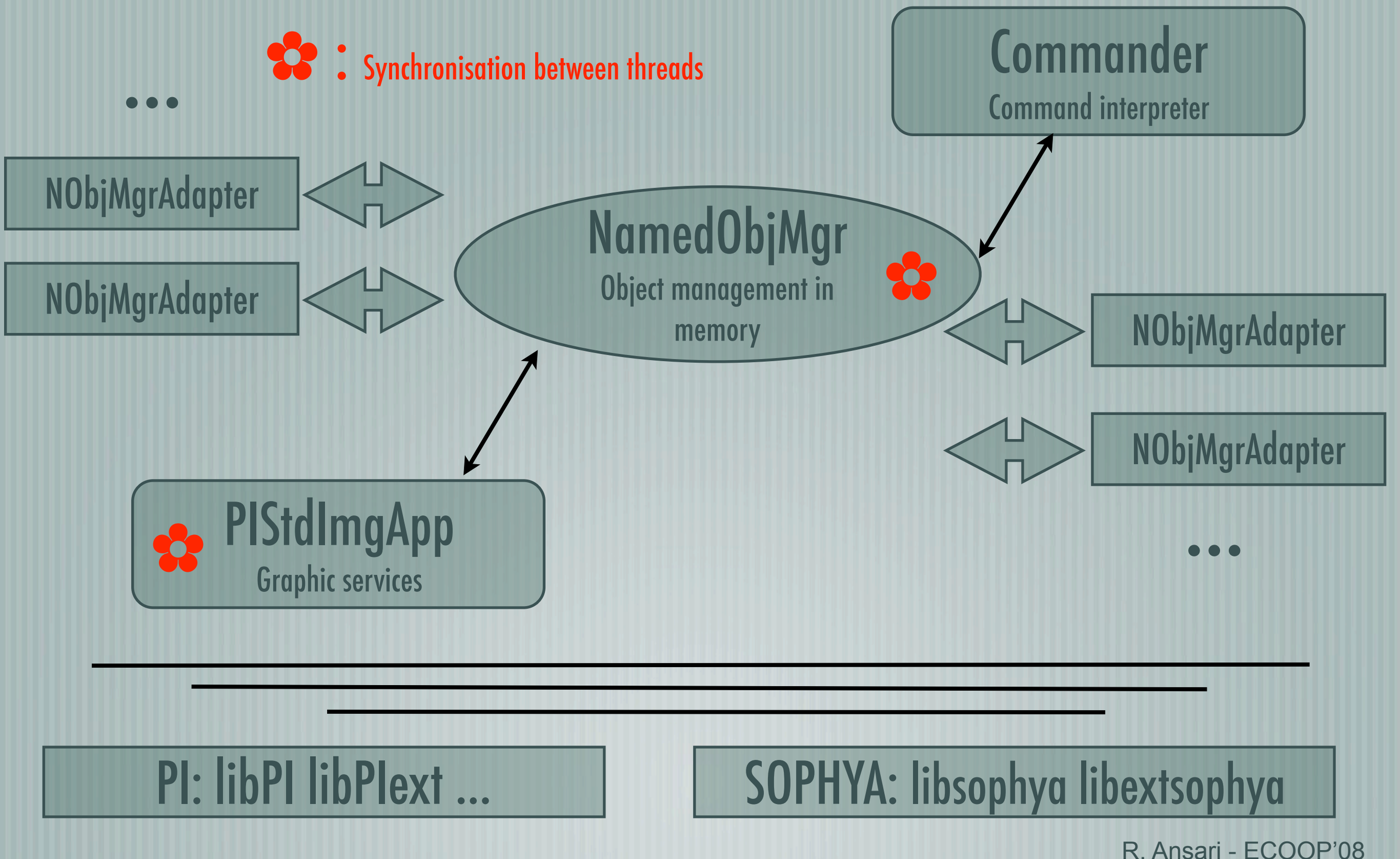
Some other, more specialized data analysis systems ...

- PAW : HBOOK/NTuple + interpreted languages (kumac+fortran) - "event" oriented, developed in HEP
- ROOT : C++ class library and C++ interpreter, from HEP (http://root.cern.ch)
- ESO-MIDAS (http://www.eso.org/projects/esomidas/)
- IRAF (http://iraf.noao.edu)
  - Processing of astronomical image/data - Unix shell like command interpreted language

piapp (1)

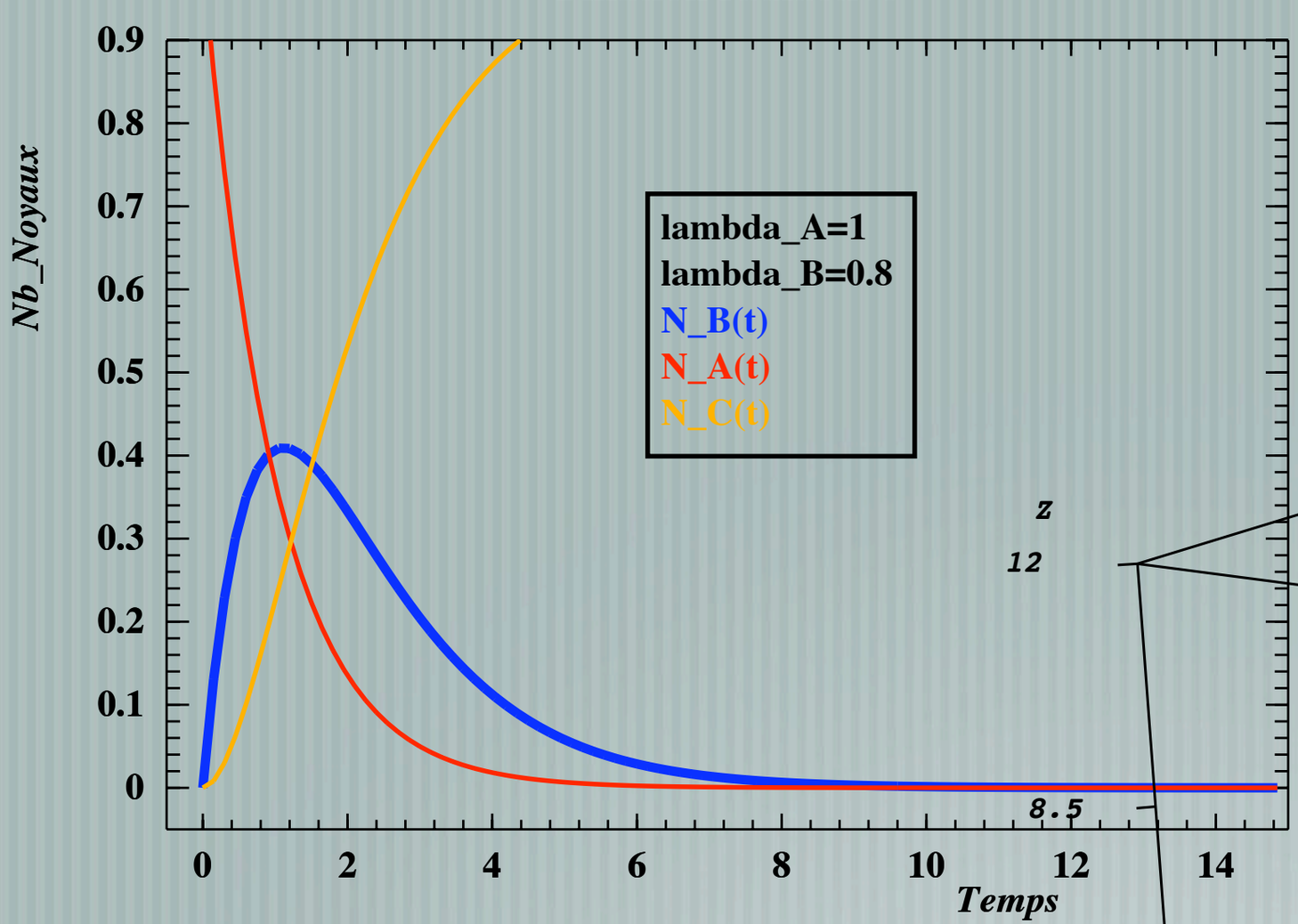
- Interactive data analysis, based on SOPHYA and the C++ GUI-graphics class library PI
- Definition of an OO interactive data analysis framework, build around a class managing objects in memory (NamedObjMgr) a class providing the graphic services (PIStdImgApp)
- Adapter classes, inheriting from NObjMgrAdapter handles data object classes (from SOPHYA or other libraries) and provide the interface between the NamedObjMgr and data object classes.
- Registration of Adapter object classes at run time (use RTTI)

Simplified (s) piapp architecture

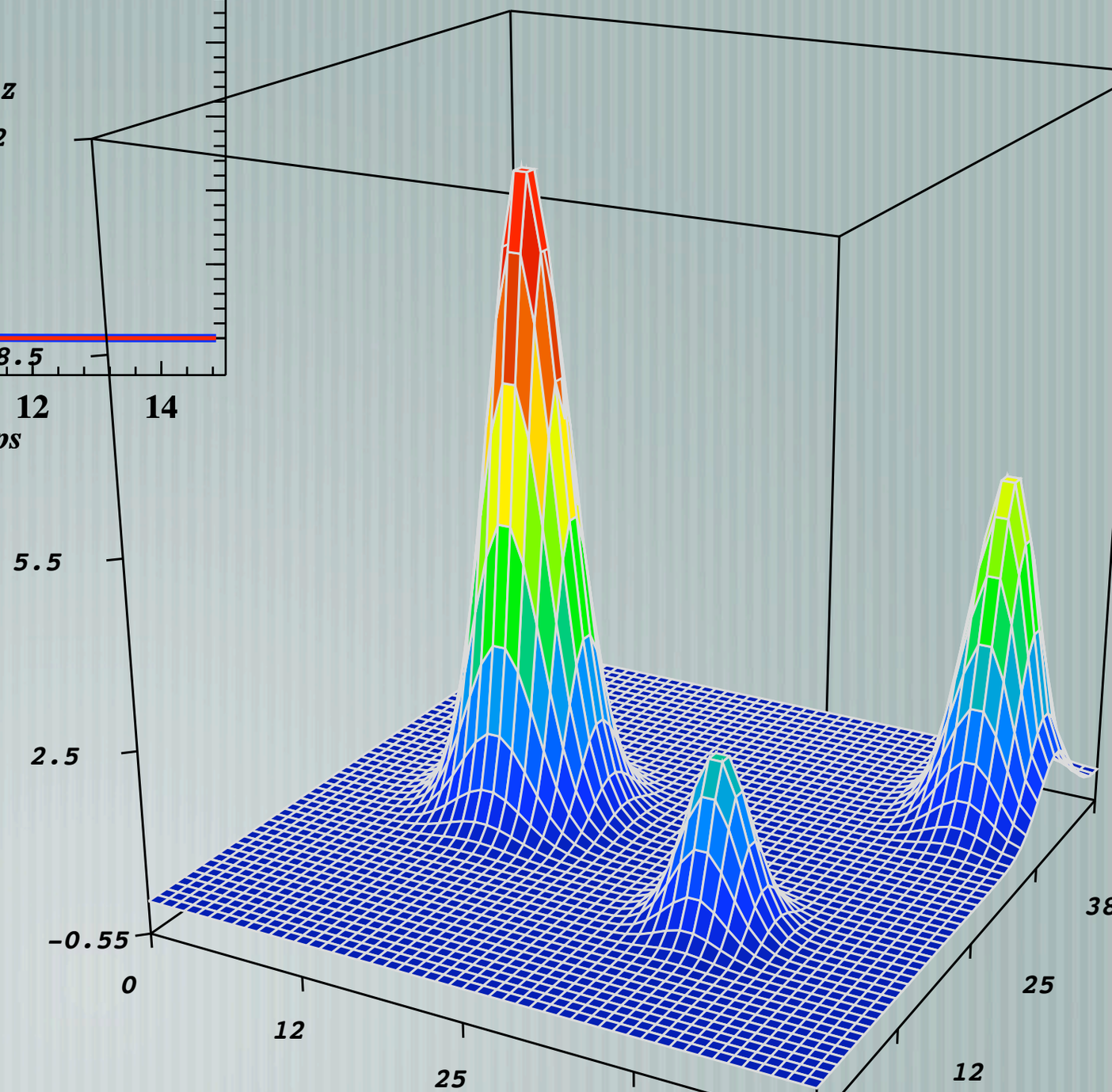


piapp (2)

- Very good graphic interactivity
- Various 2D graphic views / representations
- Image visualization
- Some 3D representations
- All graphic views are based on abstract interfaces : Graphic representation for new objects can easily be added



← 2D (functions) + text



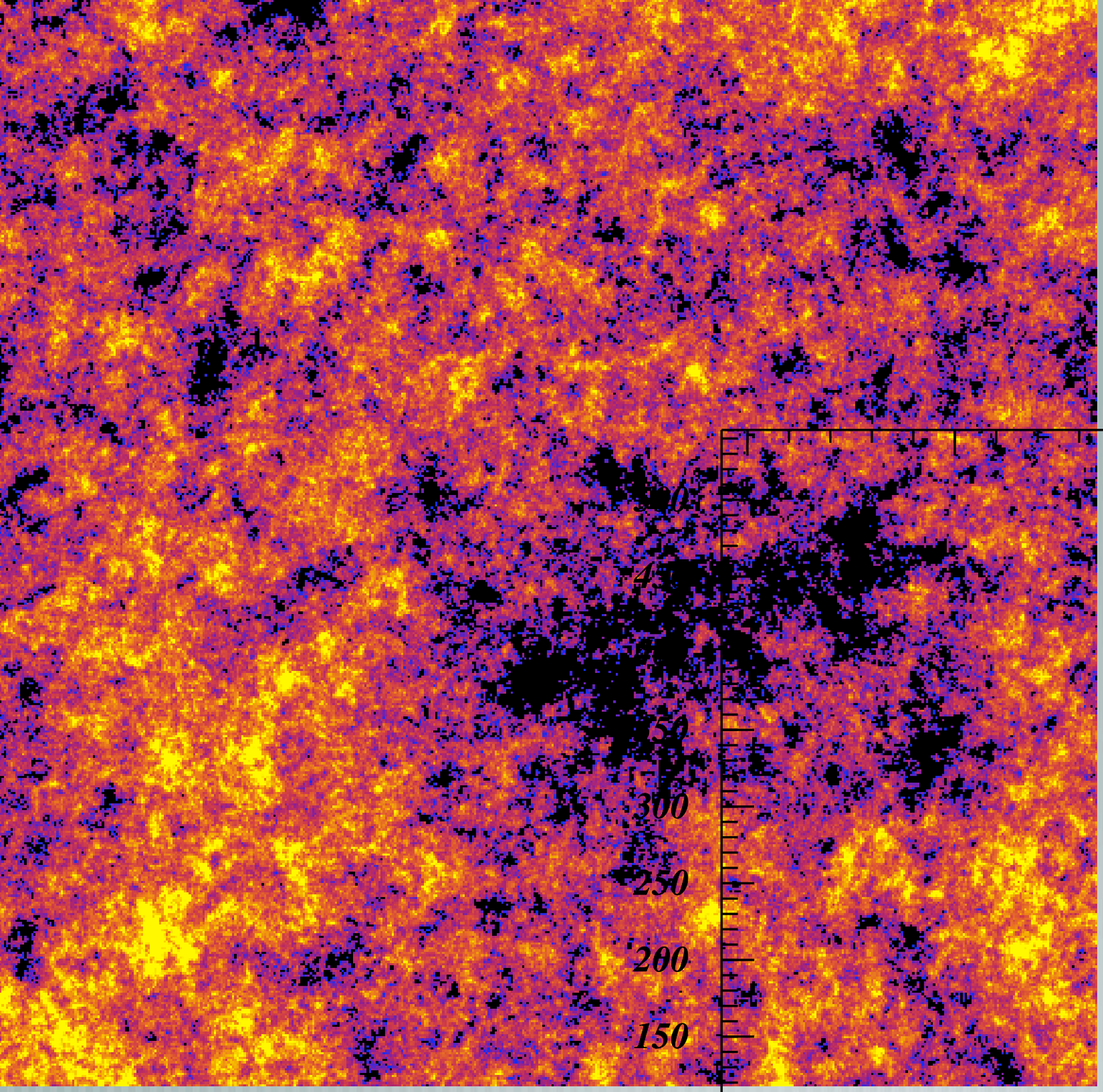
3D (surface) representation

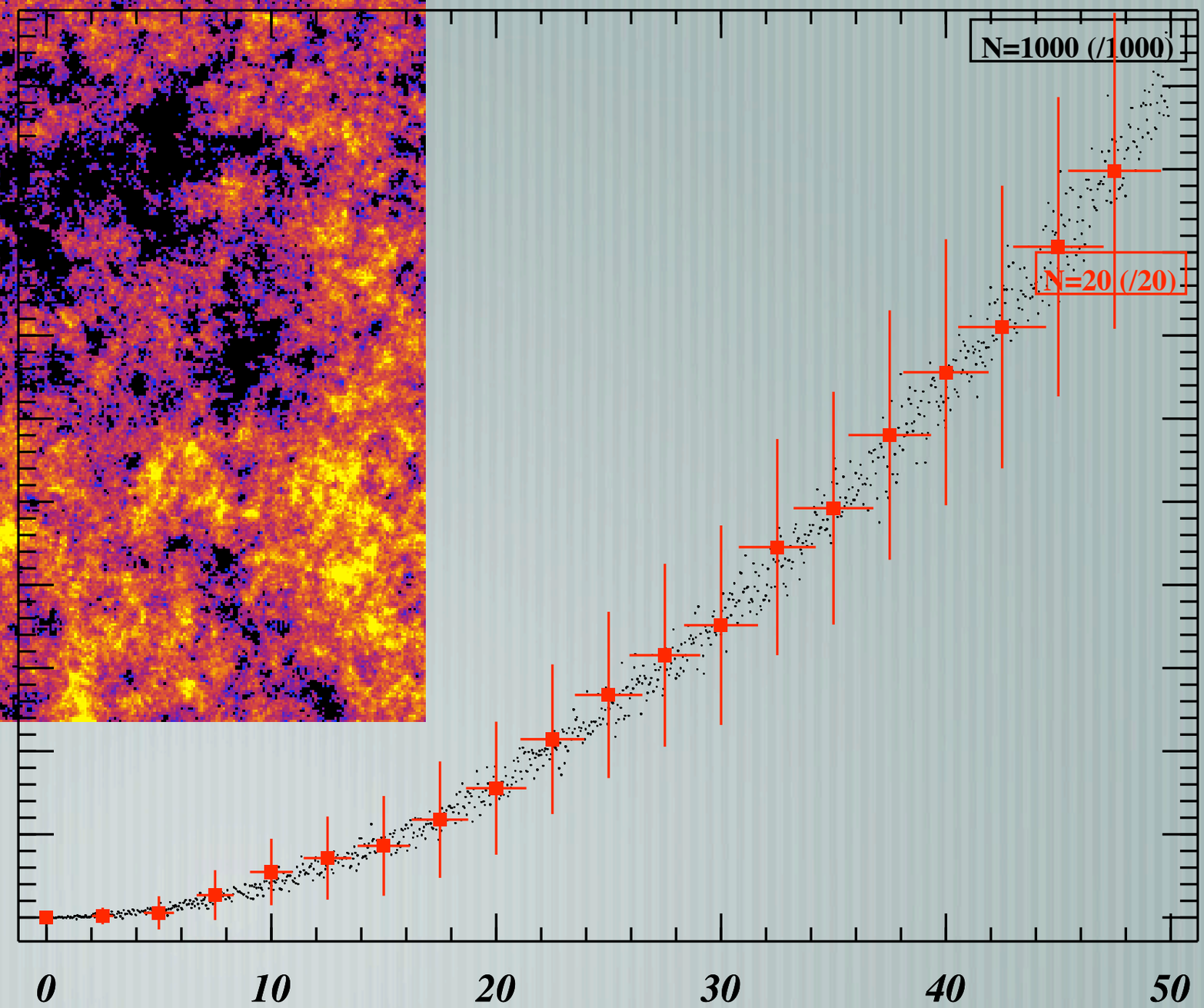
PI graphical view examples (1)

PI graphical view examples (2)

← Images

Distrib 2D

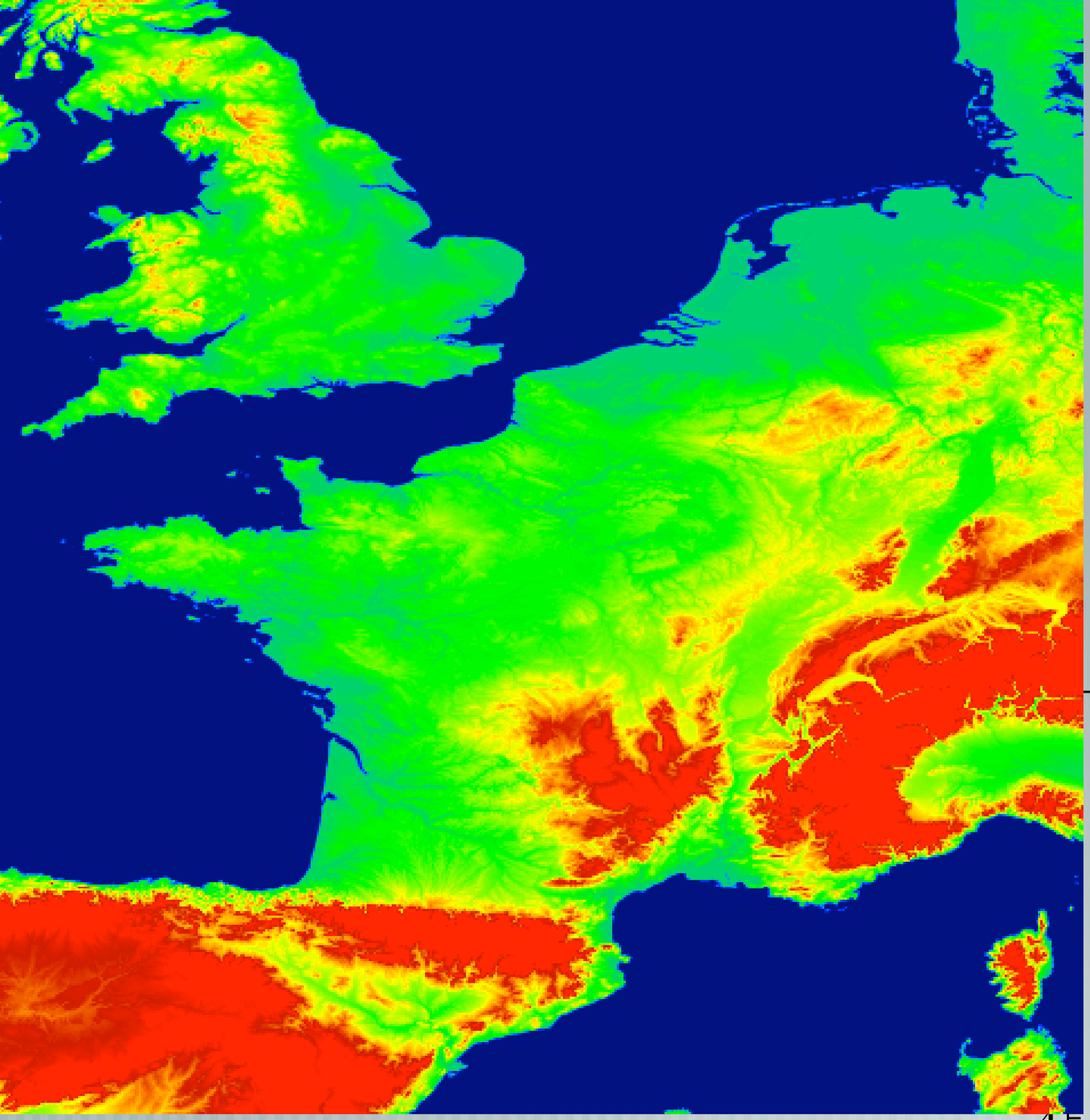




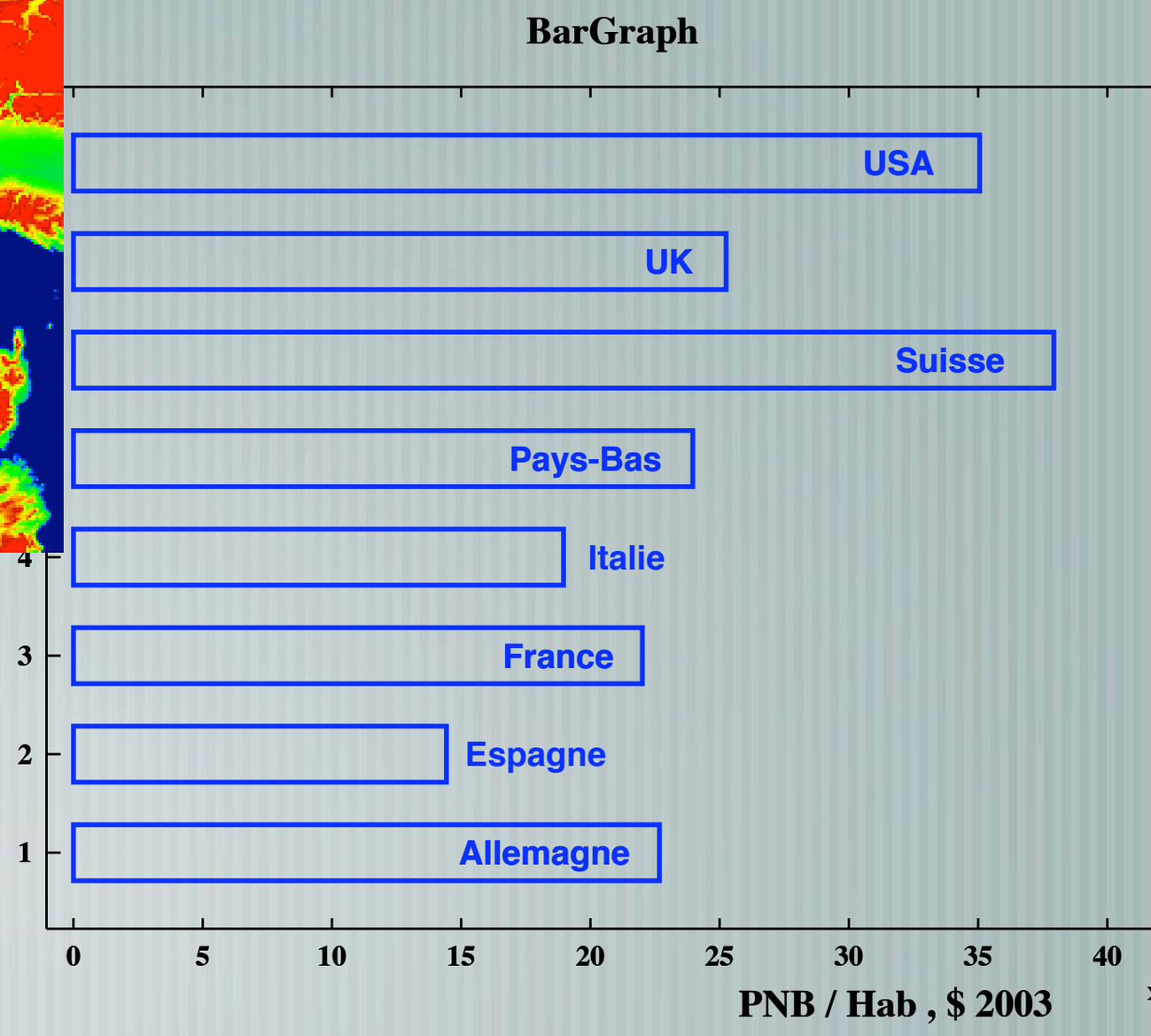
2D data points →

PI graphical view examples (2)

← Topographic data (france)



Economic data →



piapp (3)

- c-shell inspired command interpreter (Other interpreter, for example Tcl can be used)
- On the fly (C/C++) compilation, and dynamic load for functions and modules (application extension)
- PAW-like expression evaluation and representation, extended to all objects (uses adapter objects and on-the-fly C-compilation)
- C++ codelet execution, operating on data objects managed by the application (interactive code development)
- Multi-thread application, thread-safe commands can be executed on separate threads

macro crevec.pic

```
# C++ codelet execution within spiapp
# Creation of a vector with noise
c++exec \
 Vector in(1024); \
 in = RandomSequence(RandomSequence::Gaussian, 0., 1.); \
 for(int kk=0; kk<in.Size(); kk++) \
    in(kk) += 2*sin(kk*0.05); \
 KeepObj(in);
# Displaying the vector
disp in
```

macro filtvec.pic

```
#  Cleaning (filtering) the in filter
c++exec \
  Vector out(1024); \
   int w = 2; \
   for(int k=w; k<in.Size()-w; k++) \
     out(k) = in(Range(k-w, k+w)).Sum()/(2.*w+1.); \
  KeepObj(out);
#  Displaying the out vector
disp out 'same red'
```

Conclusions

- C++ classes with performances comparable to fortran/C/f90 , while offering high level abstraction and versatility
- Scientists / physicists can assemble blocks, by using classes
- A good understanding of both the scientists needs, AND the technical constraints from OO/C++ are needed when designing and implementing the class library
- Reliability, performance and ... documentation
- Need also to have “experts” , understanding coding issues in the team

Additional slides

- Fortran : First (?) high “level language” (**For**mula **Tran**slator)
  - Introduced by IBM in, 1954/1955, Fortran IV en 1962, Fortran 77, Fortran 90/95
- C++ : OO-extension of C (C-with-classes) by B. Stroustrup in the early 1980’s (AT&T)
  - ISO/IEC 14882 (1998)
- Java: Language/framework introduced Sun in the early ‘1990
  - Used initially for embedded systems Oak/green OS 1992-1993), for web/internet in 1994-1996, as a general purpose environment from 2000

C++ : Advantages

- Mature language
- Compilers reliability, performance and availability
- Foreseeable long term support
- Integration with "system" libraries (C/C++)
- Existing libraries in C and Fortran can easily be used

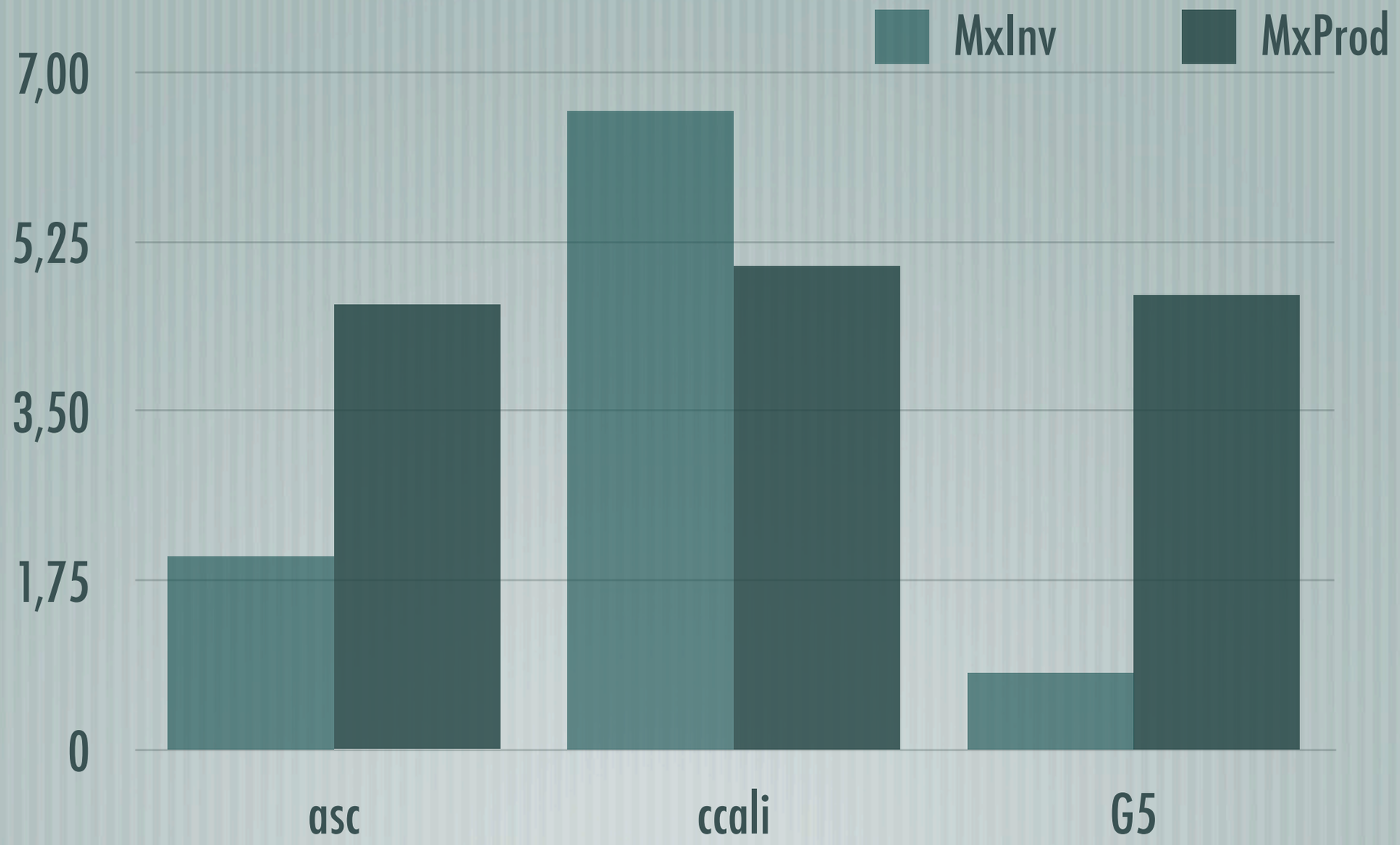
C++ : disadvantages...

- C++ is a powerful language, but rather difficult to learn
- Ambiguous syntax
- Powerful and versatile standard library (stdc++) , but lack of coverage of many useful domains in scientific computing
- No numerical library - no “standard” persistence (Object I/O) service

Platform/compilers

- Linux / g++ (3.x, 4.x)
- MacOSX (Darwin) 10.3-10.4-10.5 / g++
- HP Tru64 (OSF1) / cxx (6.x)
- SGI IRIX64 / CC (7.x)
- Linux / icc (8.0) compilateur Intel
- IBM AIX / xlC (8.x)

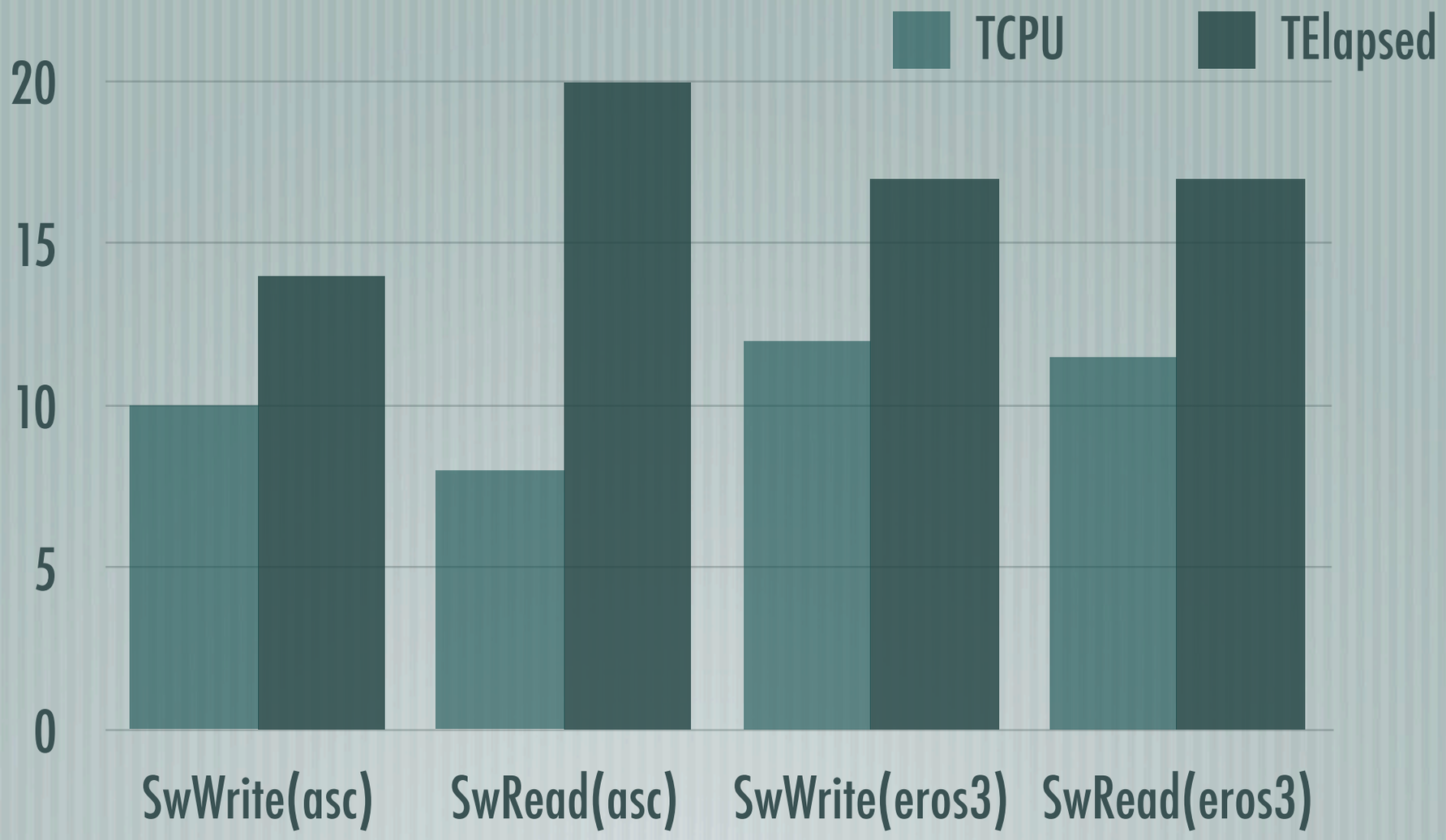
T3 sur asc,ccali,G5 (TCPU)



Inversion et multiplication de matrices

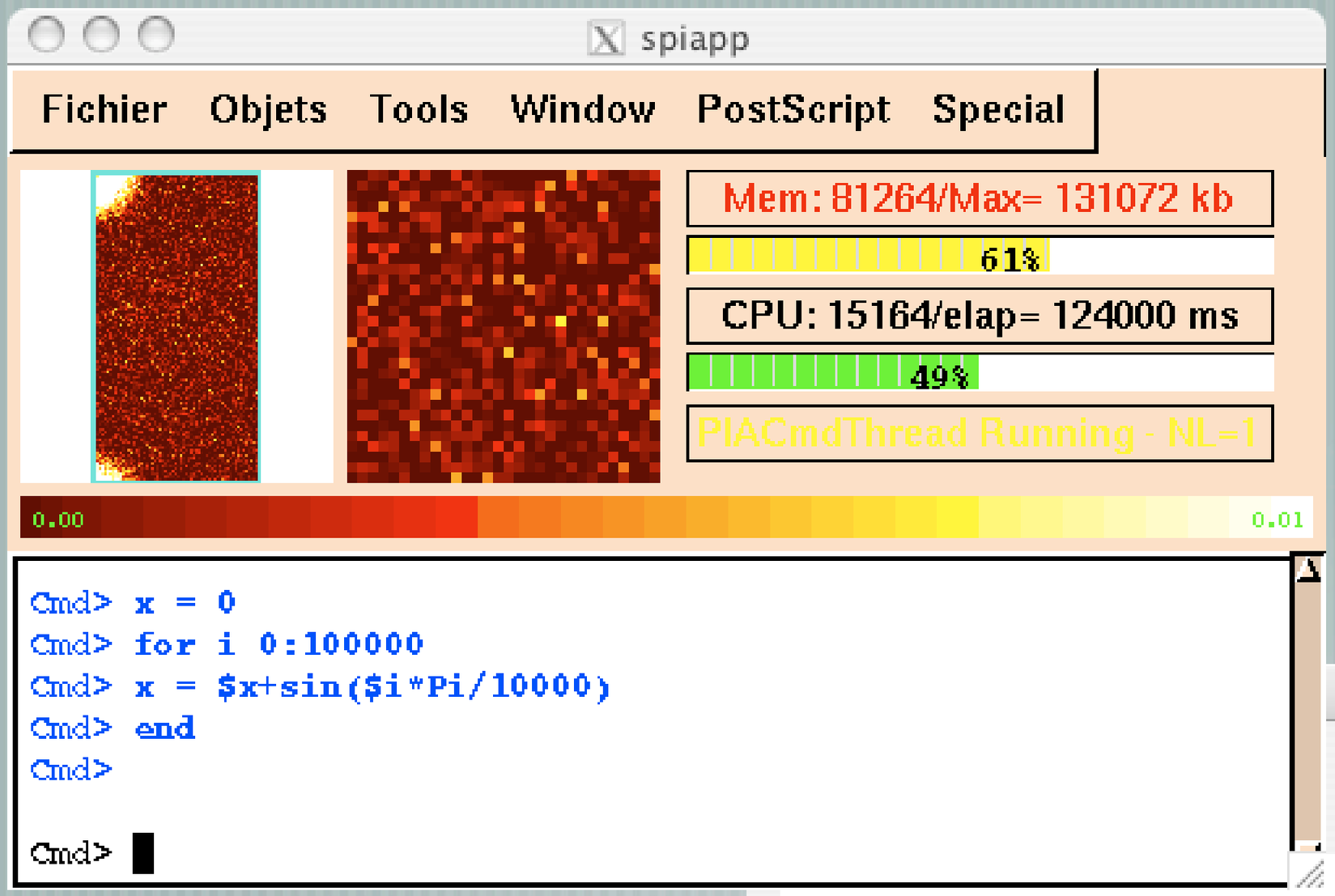
cxx/icc/g++ -O

T2 on asc and eros3 (basic discs)



Data rearrangment/conversion and I/O

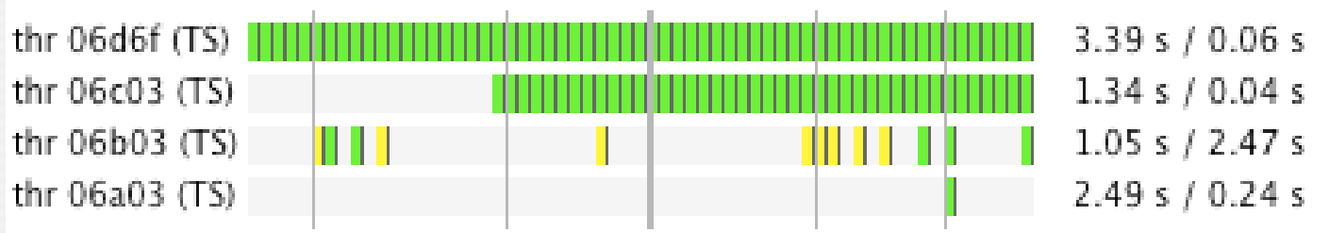
Overall data rate : 30-50 MO/sec ~ raw disk I/O rate



← Fenêtre principale
(s)piapp

Threads (s)piapp →
- Gestion des événements (2)
- Execution des commandes (2)

ThreadViewer (Mac/OS X)
Vue des threads piapp



Quelques bibliothèques numériques

- Quelques projets actifs actuellement (2006)
  - ATLAS (pas du C++) http://math-atlas.sourceforge.net/
  - C++/BOOST http://www.boost.org/
  - IML++ (Iterative Methods Library) http://math.nist.gov/iml++/
  - CLHEP http://wwwasd.web.cern.ch/wwwasd/lhc++/clhep/
  - Blitz++ (?) http://www.oonumerics.org/blitz/
  - POOMA (?) http://acts.nersc.gov/pooma/
- Portail pour librairies numériques / information http://math.nist.gov/