



Fermi Gamma-ray Space Telescope



PYTHON NELL'ESPERIMENTO FERMI

Luca Baldini

INFN-Pisa

luca.baldini@pi.infn.it

Pisa, March 23, 2011

COMINCIAMO DA FERMI O DA PYTHON?

Astronomical Data Analysis Software and Systems XX
ASP Conference Series, Vol.
I. N. Evans, A. Accomazzi, D. J. Mink and A. H. Rots, eds.
© 2011 Astronomical Society of the Pacific

The True Bottleneck of Modern Scientific Computing in Astronomy

Igor Chilingarian^{1,2} and Ivan Zolotukhin^{3,2}

¹*CDS, Observatoire astronomique de Strasbourg, Université de Strasbourg,
CNRS UMR 7550, 11 rue de l'Université, 67000 Strasbourg, France*

²*Sternberg Astronomical Institute, Moscow State University, 13 Universitetsky
prospect, Moscow, 119992, Russia*

³*Observatoire de Paris, LERMA, UMR 8112, 61 Av. de l'Observatoire, 75014
Paris, France*

Abstract. We discuss what hampers the rate of scientific progress in our exponentially growing world. The rapid increase in technologies leaves the growth of research result metrics far behind. The reason for this lies in the education of astronomers lacking basic computer science aspects crucially important in the data intensive science era.

<http://arxiv.org/abs/1012.4119>

COMINCIAMO DA FERMI O DA PYTHON?

Astronomical Data Analysis Software and Systems XX
ASP Conference Series, Vol.
I. N. Evans, A. Accomazzi, D. J. Mink and A. H. Rots, eds.
© 2011 Astronomical Society of the Pacific

The True Bottleneck of Modern Scientific Computing in Astronomy

Igor Chilingarian^{1,2} and Ivan Zolotukhin^{3,2}

¹*CDS, Observatoire astronomique de Strasbourg, Université de Strasbourg,
CNRS UMR 7550, 11 rue de l'Université, 67000 Strasbourg, France*

²*Sternberg Astronomical Institute, Moscow State University, 13 Universitetsky
prospect, Moscow, 119992, Russia*

³*Observatoire de Paris, LERMA, UMR 8112, 61 Av. de l'Observatoire, 75014
Paris, France*

Abstract. We discuss what hampers the rate of scientific progress in our exponentially growing world. The rapid increase in technologies leaves the growth of research result metrics far behind. The reason for this lies in the education of astronomers lacking basic computer science aspects crucially important in the data intensive science era.

<http://arxiv.org/abs/1012.4119>

FISICI CONTRO INFORMATICI (1/4)

—Codice da Fisici

- ▶ Scritto in FORTRAN (se va bene FORTRAN-95)...
- ▶ ...o in IDL/MATLAB
- ▶ *Script* rudimentali per la compilazione^a
- ▶ Non portabile!

—Codice da Informatici

- ▶ Scritto in un linguaggio vero: C/C++/Java
- ▶ Compilazione con *Makefile*...
- ▶ ...o soluzioni più avanzate
- ▶ Funziona su più di un computer

^aCompilare in un passo? Non è da veri uomini!

FISICI CONTRO INFORMATICI (2/4)

—Codice da Fisici

- ▶ GOTO ogni 10–20 linee
- ▶ Variabili? a1, a2, x, xx, xxx^a...
- ▶ Indentazione casuale
- ▶ Funzioni lunghissime
- ▶ Nomi di *file* e dispositivi *hard-coded*
- ▶ Illeggibile

—Codice da Informatici

- ▶ Organizzato e ben strutturato
- ▶ Convenzioni per i nomi delle variabili
- ▶ Convenzioni per l'indentazione
- ▶ *Coding convention* ben definite
- ▶ Leggibile e comprensibile

^aPossibile che il compilatore non sia abbastanza intelligente da capire che n deve essere un intero e x un numero in virgola mobile?

FISICI CONTRO INFORMATICI (3/4)

—Codice da Fisici

- ▶ Non documentato
- ▶ Soluzioni e algoritmi *intuitivi*...
- ▶ ... *sorting and searching* reinventati^a
- ▶ (e spesso in modo *creativo*, non come insegnano a scuola agli informatici)
- ▶ Per non parlare di *database*...

—Codice da Informatici

- ▶ Documentato? Non sempre...
- ▶ Ma probabilmente lo sviluppatore ha sentito parlare di *The Art of Computer Programming* di Donald E. Knuth!

^aNon so cosa sia un albero binario o una *hash table*. E poi che cosa vuol dire la O in $O(n^2)$ e $O(n \log n)$?

FISICI CONTRO INFORMATICI (4/4)

—Codice da Fisici

- ▶ Funziona!
- ▶ ... perché l'autore sa esattamente quel che vuole che il codice faccia
- ▶ Magari lentamente
- ▶ E magari a volte *crusha* inspiegabilmente

—Codice da Informatici

- ▶ Compila!
- ▶ ... ma non è detto che l'autore padroneggi i principi fisici alla base del problema
- ▶ (stiamo sempre parlando di codice per la ricerca in fisica!)

Il modello “scrivere una volta, vendere molte volte” non si applica necessariamente in Fisica

CHE COS'È PYTHON?

Python è un linguaggio di programmazione dinamico orientato agli oggetti utilizzabile per molti tipi di sviluppo software. Offre un forte supporto all'integrazione con altri linguaggi e programmi, è fornito di una estesa libreria standard e può essere imparato in pochi giorni. Molti programmatori Python possono confermare un sostanziale aumento di produttività e ritengono che il linguaggio incoraggi allo sviluppo di codice di qualità e manutenibilità superiori.

Python gira su Windows, Linux/Unix, Mac OS X, OS/2, Amiga, palmari Palm e cellulari Nokia; è stato anche portato sulle macchine virtuali Java e .NET.

Python è distribuito con licenza Open-Source approvata dalla OSI: il suo utilizzo è gratuito e libero anche per prodotti commerciali.

<http://www.python.it/>

PYTHON VS. C++: HELLO WORLD!

Python

```
print 'Hello world!'
```

c++

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

Python vince in linee di codice...

```
> python hello.py
```

```
> g++ hello.cpp -o hello.exe
> ./hello.exe
```

... e non va nemmeno compilato!

UN ESEMPIO PIÙ INTERESSANTE

IN 1348 LINGUAGGI DIVERSI SU [HTTP://99-BOTTLES-OF-BEER.NET/](http://99-bottles-of-beer.net/)

99 bottles of beer on the wall, 99 bottles of beer.

Take one down and pass it around, 98 bottles of beer on the wall.

98 bottles of beer on the wall, 98 bottles of beer.

Take one down and pass it around, 97 bottles of beer on the wall.

97 bottles of beer on the wall, 97 bottles of beer.

Take one down and pass it around, 96 bottles of beer on the wall.

...

2 bottles of beer on the wall, 2 bottles of beer.

Take one down and pass it around, 1 bottle of beer on the wall.

1 bottle of beer on the wall, 1 bottle of beer.

Take one down and pass it around, no more bottles of beer on the wall.

No more bottles of beer on the wall, no more bottles of beer.

Go to the store and buy some more, 99 bottles of beer on the wall.

IN PYTHON...

Python

```
for n in range(99, 2, -1):
    print '%d bottles...wall, %d bottles of beer.' % (n, n)
    print 'Take one...around, %d bottles of...wall.\n' % (n-1)
print '1 bottle...wall, 1 bottle of beer.'
print 'Take one...around, no more bottles...wall.\n'
print 'No more bottles...wall, no more bottles of beer.'
print 'Go to the store and buy some more, 99 bottles...wall.'
```

Gamma-ray
Space Telescope

IN C++...

c++

```
#include <iostream>
using namespace std;

template<int i> class Verse{
public:
    static inline void printout() {
        cout << i << " bottles...wall, " << i << " bottles of beer.\n";
        cout << "Take one...around, " << i-1 << " bottles of...wall.\n\n";
        Verse<i-1>::printout();
    };
};

template<> class Verse<1>{
public:
    static inline void printout() {
        cout << "1 bottle...wall, 1 bottle of beer....\n";
    };
};

int main(){
    Verse<99>::printout();
    return 0;
}
```

C'È DI PEGGIO...

HTTP://WWW.LSCHEFFER.COM/MALBOLGE.SHTML

malbolge

b`';\$9!=IlXFtVwwvtP00)pon%IHGFDV|dd@Q+=:(`&Y\$#m!1S|.Q00=v('98\$65aCB}0i.Tw+QPU'7qK#I20jiDVgG
S(bt<%#!7~|4{y1xv.us+rp(om%lj"ig}fd"cx`uz]rwwYnslkTonPfOjikGjeG]\EC_X}@[Z<R;VU7S6QP2N1LK-I
,GF(D'BA#?>7~;:9y16w43s10)p-,l*#(i&e#d~`'{txzPzuXsrTTong0kdMhg'Hd]ba`_~W@[ZYXW9UNSRQPOHMLK
J-++FE`'[A\\$?](#)=<;:387xw43s10/(-&m*)(`&){\$d~}|`zyxwvutmVqpiRQlkjiKafedc\`E`_~@\[ZYX;V9NMRQ42NG
LK.IH*F?DCBA\$#>7~;:{8xx5uu2rr/oo,11)ii&f|e!"aw`{z\r[vXnmVTpongPkNiHgJ_dcFa'B`]\UZ=RWV8TSLQ40
NOLE.IHA>'BA?:!7~5[38y6/v321q].-&m*)i`%{|{d~}_zs\vwutsUqTonPl0jikGJedFbE'_A}@[Z<X;VU7S6Q
P22GL/JIB+FEDC%;@?>7~;:987w5v32r0)p-,+k)(`~g\$#`b`w\uz]xvwutsrqTinQ10jLhgfeH]bE`CB]\>ZSXWVUTS
RQPON1LE.I,++((&&\$"~~|zzxxv4u210/(-n+l)(i&g\$ddy`~u`\]ZvutVlUjSQQQ0dMKgfeG]F[DBB@><<:VU
T6L5J0200EJ-HG*E>'B%\$9=<|4{2y05v321r).o,mlj(igg[#d~]`uz]x[ZotWUUjoRmlkNiBkJIGGEZ_B]\?Z=XW
PU876442NM/KD-B+))'%%##!!){yyw5v32s0q.-&l+j'hff{"caav`~yxwZutsUpSnQ00diLgfHHcb`Y^A\?Z=
;:PU8SRQ40NMLEJ-,+))'%%##!=<:{3z1xvvttrppnnl#j!&g\$#d!b}\{zry[vYtsrTjShQfkNiHgJedcba`Y^A\?
Z=:WV9TSRQPOHMKO.-++)ED&B;\$9"~<;:z2y0wuussqqoom+1jj!&%dzcx`{zy\vwutsrqjSnpNNLhgIedG\EZCA]
\=[S<Q:886644220L/JIHA*)(&&@?!=6)4{yywwuus10/o'n%lj(`&f|ezcaa_)]\wvuWmVkTrnQlkNLlaJiHFbE`_
B@U>Y<;P9775533H1/KJ,HA?(&&\$")=<|4{2ywuu321q)p'n1*k`gg\$["c~a`^z]xwvYtmrUpSRP10MMbK`IGGEE
Z_~]?U>S<;:8866442200.JIH@>C&A@?"=<5|{8y65vtt10/(-n+lk)`%e{dyb`~`~\ZvutVlUjSQmlkMcLaJHHF
bECCX]\=[S<Q:886R5PON1LKJCH+F)(=BAQ"8!6){2y0543s+r)pnlljhhffdubb`_zyx[vutslUTSQQ00MiHgI_
H]FDBB@><XWV8N7L5331MLK-C,A*(D'BA\$"=<;927xwvt2s0/p-n+*)(`~%f#dcaa_)]y\ZzotsrTjShQ0kjiKa
J_HFFDDBB@><X;99NS6QP02MLKJIHA*E('%%:#8=~;:9z7654321*/p-,m*k(hh)\$#dyb}\{zy[qzoXVVTTRnmlNdM
bKIIGGECCAA?[>YXWP9T76K42200.JI+G@)>%A@?!7~5[zx654t,s*qo-n+jj!h%fec!b]\{^s\[vYWlqTonQ10
jchKfIHFFDDB^]\>T=R;P9NS6QP02MLE.-,*FED&<:#!{}){yyw543s+r)pnllkii~%f#"caal\{zsx[ZutVrkTinQ
1kNiLgfe`cFEDYBW\=[YR;P9775530200EJH@>C\$!<;19z7654-tsrrppnnl#(`&f|ezca)\{]s\qZXtsrTjS
hQ00MiHgI_H]FDBB^A\[<<WVUTSLQ43NM/KD-BG+ED'B%@?>=<5:{zy0wuussqqoom\$ki'hff{"c~}`{t}\wvuWmVk
pSmnPNNClKfIGG\aD_~A\?T=<;99775QPO1GOE_,HG)E>`<%#?"~5:98x0w.ussq/pnn/*k(`hff#z!bal\{z\r[puXs
rUpSngl0NiHgI_H]FDDYBW\[Z<R;P977553311//--+))`CBA#9"7<:9z7x54-tirq(ommkkiiggeecca)\{]s\qZX
...
(circa 10%)

PERCHÉ PYTHON È COSÌ BELLO?

- ▶ È *facile!*
 - ▶ Si impara velocemente
 - ▶ È relativamente facile anche fare cose complicate
 - ▶ Alcuni esempi nelle trasparenze successive
- ▶ Ha una libreria standard estremamente vasta
 - ▶ Manipolazione di stringhe, xml, comunicazione di rete, espressioni regolari, numeri pseudo-random etc...
- ▶ È uno dei linguaggi di scripting più usato (il sesto)
 - ▶ Un numero sconfinato di pacchetti in aggiunta alla libreria standard
 - ▶ Un numero sconfinato di programmi con *wrapper* in Python
- ▶ Permette un ciclo di sviluppo estremamente veloce
 - ▶ Tipicamente più snello in termini di linee di codice (non si dichiarano le variabili, i contenitori sono estremamente flessibili)
 - ▶ Non va compilato
 - ▶ *Traceback* completo al *crash* (anziché *segfault*)
 - ▶ **Python's interactive. It's not edit-compile-link-execute-break-debug. It's edit-debug.**

ALLORA: DOV'È LA FREGATURA?

LA VELOCITÀ

loop.py

```
import time

NUM_CYCLES = 100000000

startTime = time.time()
total = 0
for i in xrange(NUM_CYCLES):
    total += 1
elapsedTime = time.time() - startTime
print '%d cycles done in %.2f s.' %\
    (NUM_CYCLES, elapsedTime)
```

loop.cpp

```
#include <iostream>
#include <time.h>
#define NUM_CYCLES 100000000

int main()
{
    int startTime = clock();
    int total = 0;
    for (int i=0; i<NUM_CYCLES; i++){
        total += 1;
    }
    double elapsedTime = double(clock() -
        startTime)/CLOCKS_PER_SEC;
    std::cout << NUM_CYCLES <<
        " cycles done in " <<
        elapsedTime << " s." << std::endl;
    return 0;
}
```

```
> python loop.py
100000000 cycles done in 19.30 s.
```

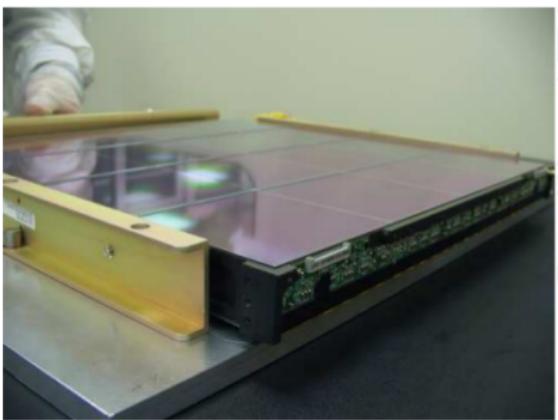
```
> g++ loop.cpp -o loop.exe
> ./loop.exe
100000000 cycles done in 0.33 s.
```

CHE COS'È FERMI

- ▶ Esperimento di (astro)Fisica su satellite
- ▶ Progettato per la rivelazione di raggi gamma da sorgenti celesti
- ▶ Lanciato nel giugno 2008, durata della missione: 5-10 anni



NUMEROLOGIA



► Il Large Area Telescope

- Dimensioni: $\approx 2 \times 2 \times 1.5 \text{ m}^3$
- Massa: ≈ 3 tonnellate
- Consumo: $\approx 650 \text{ W}$

► Il tracciatore

- Assemblato completamente in Italia (in questo edificio)
- ≈ 1 milione di canali di elettronica indipendenti
- ≈ 10000 rivelatori al silicio
- ≈ 15000 chip di elettronica di lettura (ASIC) dedicati
- ≈ 70000 registri indipendenti per la gestione della configurazione.

Consuma meno di un tostapane e gli parliamo su una linea telefonica (Bill Atwood)

IL LANCIO



IL LANCIO



COME FUNZIONA?



- Dalla progettazione al lancio

- ▶ Primo articolo scientifico: Space Science Rev. 75: 109–125, 1996
- ▶ Fase di R&D (inclusi test su fascio ed un volo su pallone): 1999–2004
- ▶ Costruzione ed integrazione dello strumento: 2004–2006
- ▶ Integrazione con lo spacecraft: 2006–2007
- ▶ Test su fascio di una unità di calibrazione: 2006
- ▶ Lancio: giugno 2008
- ▶ Analisi dei dati scientifici: 2008–in corso

- Python è onnipresente in Fermi:

- ▶ Prototipizzazione
- ▶ Costruzione e test
- ▶ Test su fascio
- ▶ Monitoraggio dei dati di volo
- ▶ Analisi scientifica

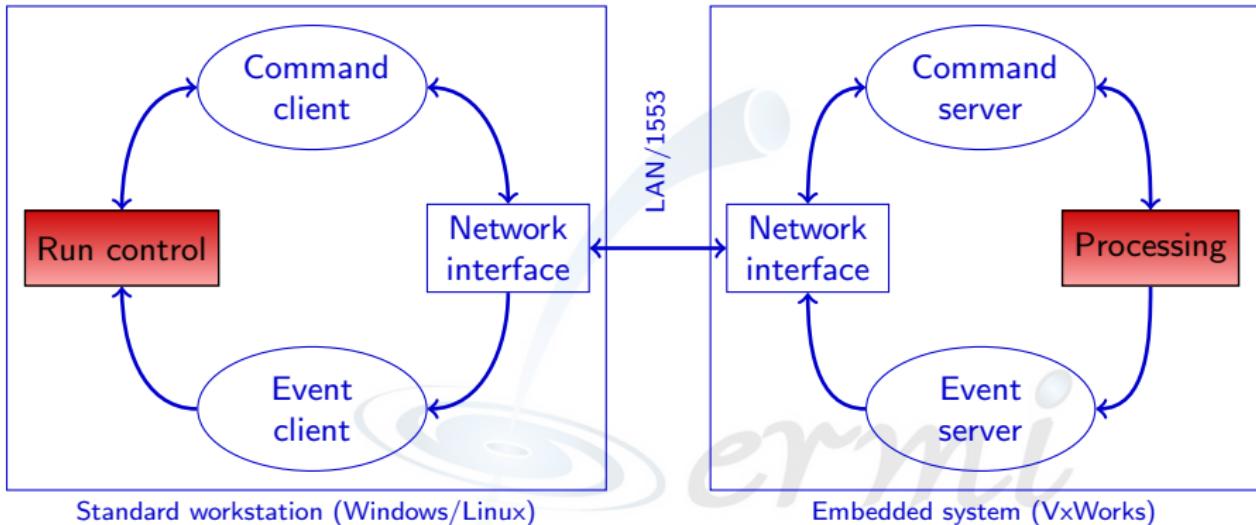
- Python è stata un'ottima scelta per Fermi

- Python è un'ottima scelta per Fisici

CHE COSA È UN DAQ?

- ▶ DAQ: *Data AcQuisition System*
 - ▶ Qui e nel seguito inteso in senso allargato (DAQ in senso stretto ed eventuale interfaccia)
- ▶ Tipicamente una cosa difficile da implementare
 - ▶ Si interfaccia con hardware estremamente specializzato o completamente *custom* (con le proprie idiosincrasie)
 - ▶ Deve operare in **tempo reale**
 - ▶ Deve essere ottimizzato e **stabile**
 - ▶ **Multithread** (interfaccia grafica utente)
- ▶ Vasto assortimento di soluzioni sul mercato
 - ▶ Un singolo file in FORTRAN con interfaccia *command-line*
 - ▶ Con o senza Interfaccia Grafica Utente (GUI)?
 - ▶ Scheda di acquisizione NI e LabVIEW (normalmente non adeguata per un esperimento di grandi dimensioni)
 - ▶ Monitor dei dati assente, integrato o esterno?

FERMI: IL DAQ PER I TEST A TERRA



- ▶ Struttura modulare
 - ▶ Parte *real time* e interfaccia utente completamente disaccopiate
- ▶ Basso livello: in C; alto livello: quasi interamente in Python
 - ▶ Con l'eccezione di alcune funzionalità critiche (spacchettamento dei dati) scritte in C++ ed esposte in Python come librerie (SWIG)
- ▶ Ottimizzare ogni singola parte del codice è inutile (e quindi dannoso)

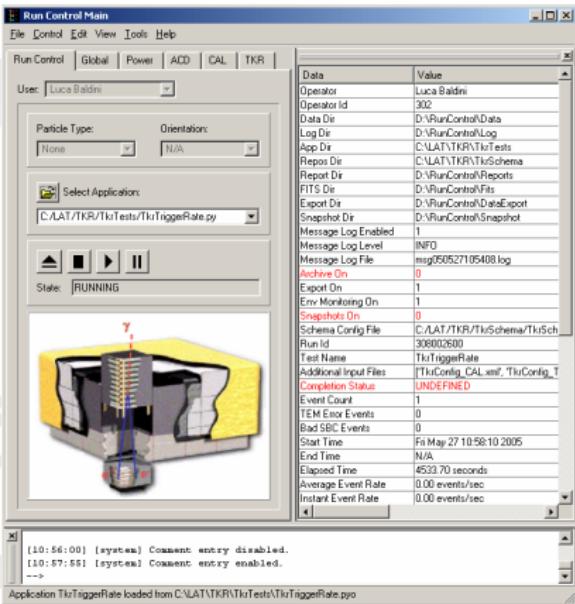
IL SISTEMA EMBEDDED



RUN CONTROL (LA PARTE IN PYTHON)

—Un test in poche parole

- ▶ Configurazione dello strumento
- ▶ Invio comandi
- ▶ Lettura dati
- ▶ Analisi dati
- ▶ Generazione di un *report*

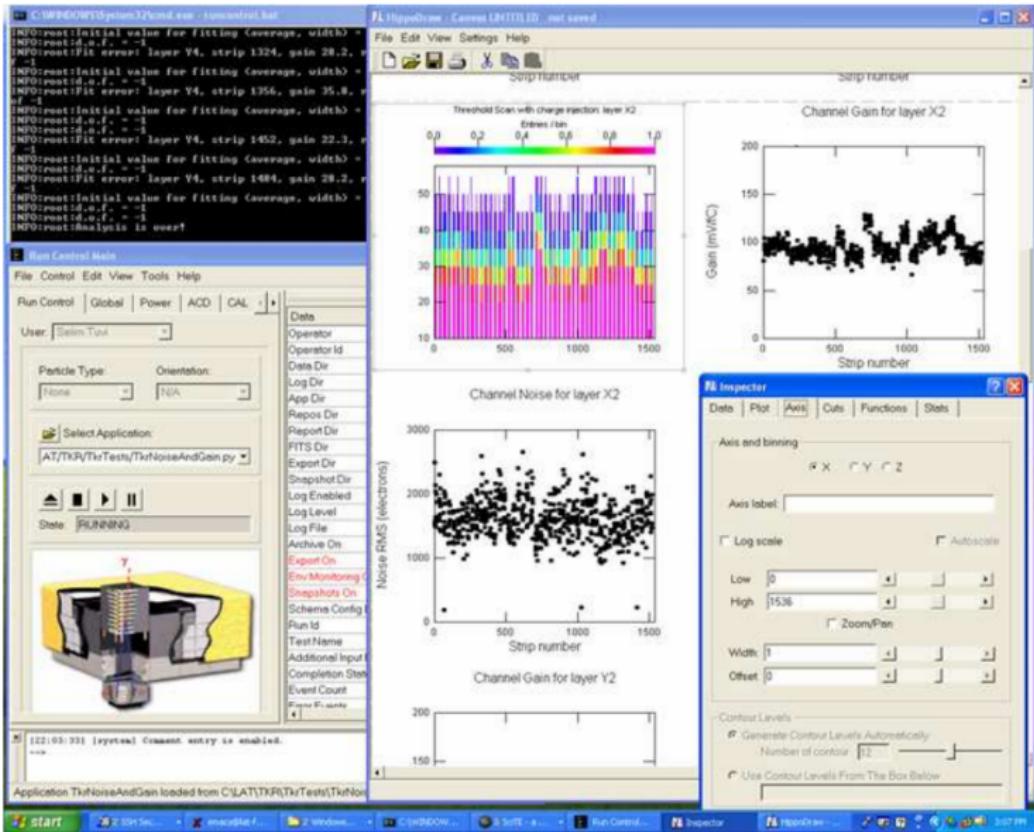


—Run control fa tutto questo

- ▶ Basato su una macchina a stati finiti
- ▶ Monitor dello stato dello strumento
- ▶ Distribuzione ed archiviazione dei dati
- ▶ Visualizzazione online dei dati

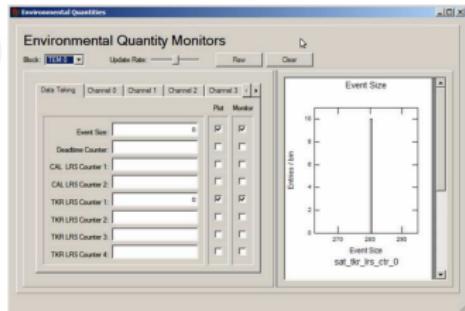
Se è vero che *Python è facile*, perché non fare in Python tutto ciò che non incide significativamente su tempo totale?

UN ESEMPIO DI APPLICAZIONE



DAQ E MICRO-THREADING

- ▶ Python supporta il *multi-threading*
- ▶ Utile per piccole applicazioni da lanciare dalla GUI principale
 - ▶ *Browsing dei registri, monitor ambientale*
- ▶ In polling sull'hardware



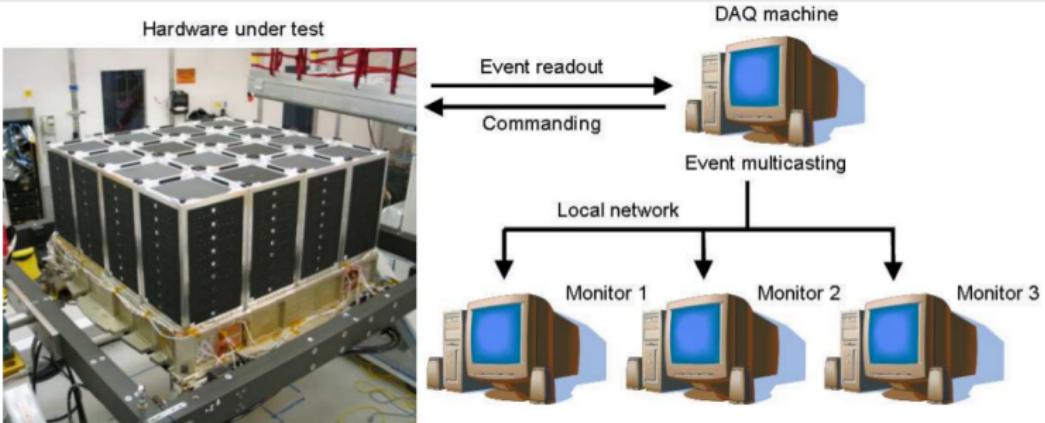
```
import threading
from time import sleep
new_thread = threading.Thread(None, loop, 'loop')
self.new_thread.start()

def loop():
    read_registers()
    sleep(1)

def read_registers():
    ...
```

Unfortunately, for most mortals, thread programming is just Too Hard to get right.... Even in Python

DAQ E MONITORING



- ▶ RunControl può *forwardare* i dati su un socket UDP con un *overhead* minimo
- ▶ DAQ e monitoring completamente disaccoppiati
 - ▶ Una macchina dedicata esclusivamente all'acquisizione dei dati
 - ▶ Una (o più di una, magari configurate diversamente) macchina per il monitor dei dati (lavoro intenso in termini di CPU)

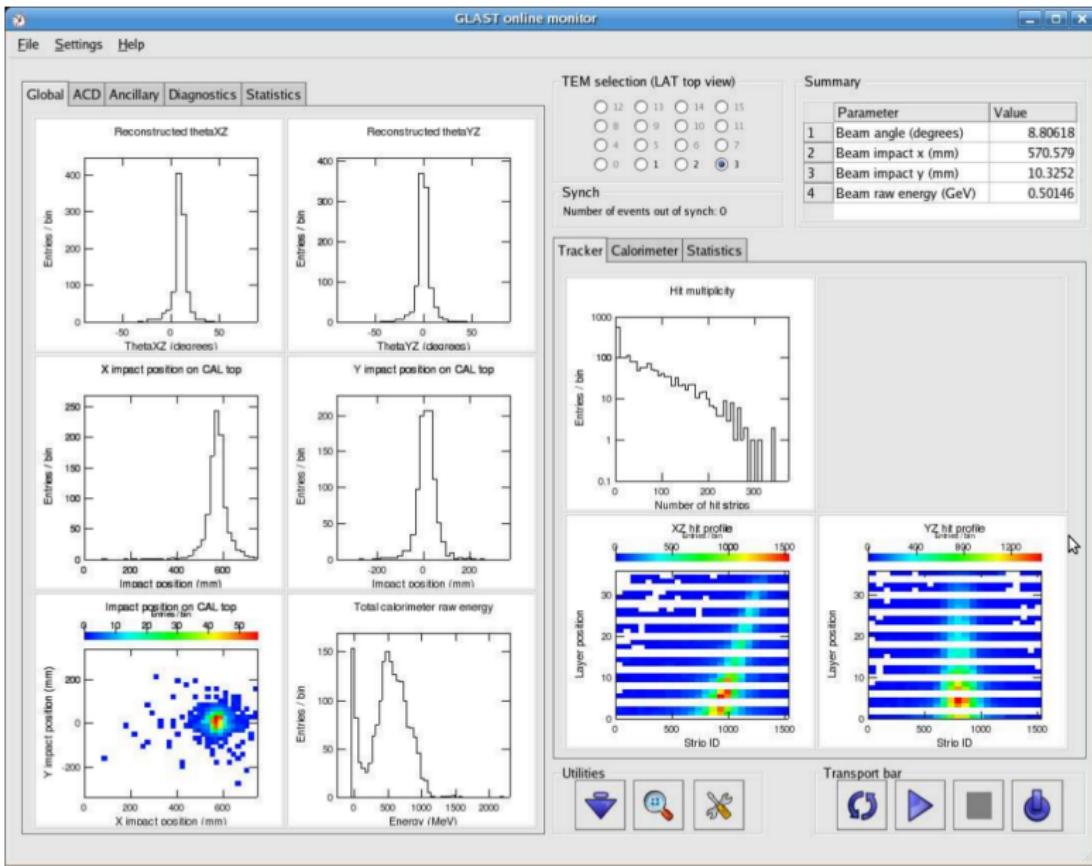
UN APPLICAZIONE SPECIFICA: IL BEAM TEST (1/3)

- ▶ Letteralmente: test su fascio (di particelle prodotto da un acceleratore)
 - ▶ Slot di tempo ben precisi (e non rinegoziabili) assegnati
 - ▶ Il tempo di fascio è prezioso e non recuperabile: non è permesso sbagliare
 - ▶ La capacità di adattarsi all'ambiente è un punto cruciale
- ▶ Il software di acquisizione e monitor dei dati deve essere abbastanza flessibile da adattarsi alle esigenze (difficilmente prevedibili a priori)
- ▶ Due campagne estensive di test per la calibrazione dello strumento: CERN (Ginevra) nell'estate 2006 e GSI (Darmstadt) nell'autunno 2006
 - ▶ Necessario per caratterizzare le prestazioni dello strumento con sorgenti note...
 - ▶ ... e per validare la simulazione dello strumento stesso
 - ▶ Un gruppo di ~ 40 persone impegnato per un mese e mezzo (tempo biblico per un *beam test*, modesto per il ciclo di utilizzo di un pacchetto *software*)

UN APPLICAZIONE SPECIFICA: IL BEAM TEST (2/3)



UN APPLICAZIONE SPECIFICA: IL BEAM TEST (3/3)



DUE PAROLE SU ROOT

- ▶ Framework di analisi dati utilizzato da tutti i grandi esperimenti di Fisica delle alte energie
- ▶ *The backbone of the ROOT architecture is a layered class hierarchy with, currently, around 1200 classes grouped in about 60 frameworks (libraries) divided in 19 main categories (modules).*
- ▶ I/O estremamente ottimizzato
 - ▶ Tipicamente il formato per l'archiviazione dei dati
- ▶ Tre modalità principali di utilizzo:
 - ▶ Libreria C++
 - ▶ Interpret C++ compliant
 - ▶ Python wrapper

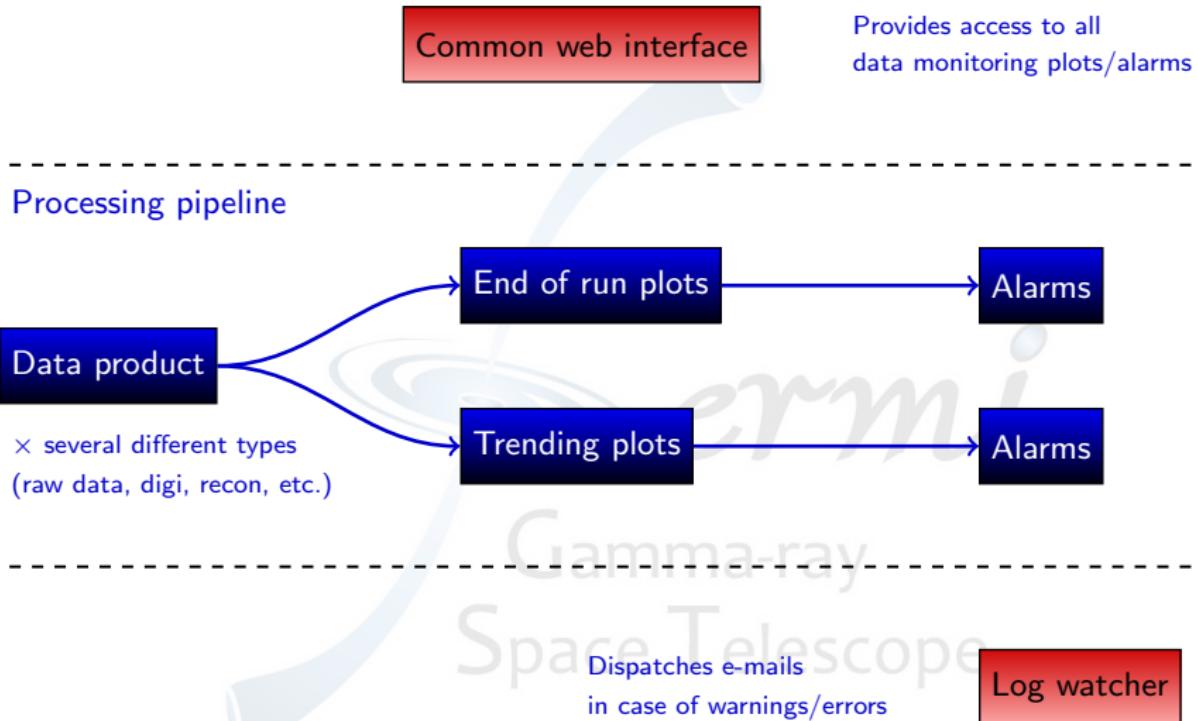
C++

```
#include TChain.h  
  
TChain *t = new TChain("MeritTuple");  
t->Add("somefile.root");  
t->Draw("CalEnergyRaw");
```

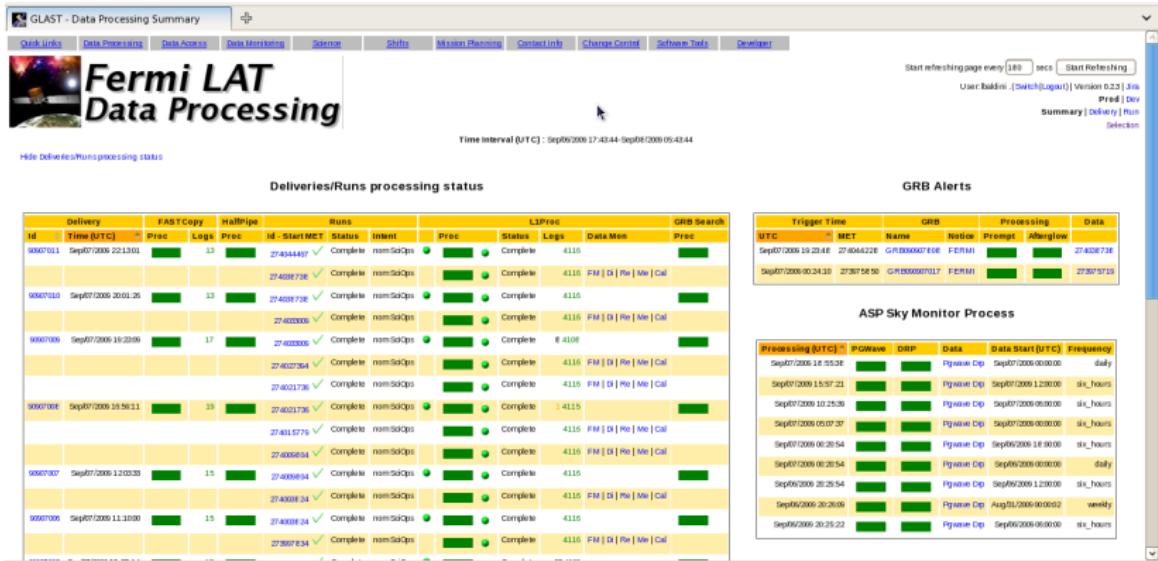
Python

```
import ROOT  
  
t = ROOT.TChain("MeritTuple")  
t.Add("somefile.root")  
t.Draw("CalEnergyRaw")
```

IL MONITORING DEI DATI (1/3)



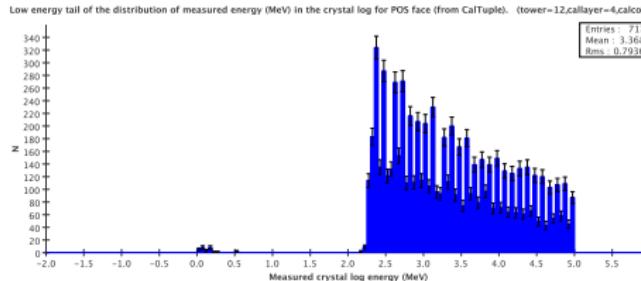
IL MONITORING DEI DATI (2/3)



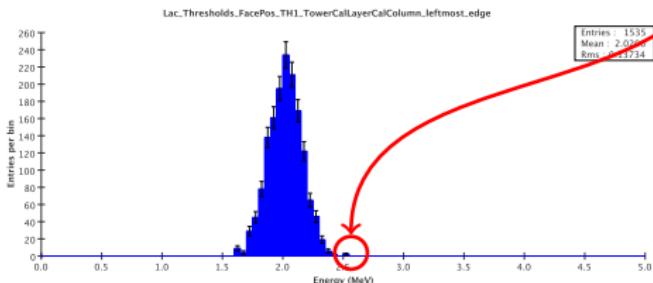
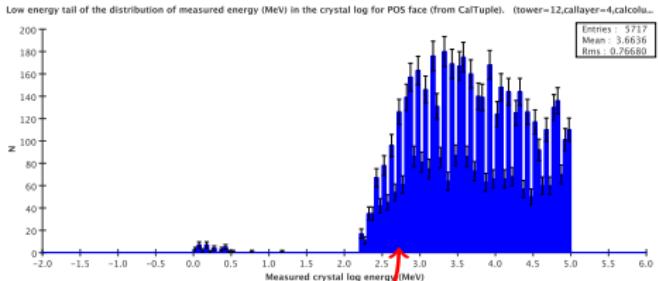
- Facile accesso a tutte le informazioni necessarie per il monitoring dei dati
- Circa 120,000 grafici e 4,000 allarmi sulle relative quantità a portata di click

IL MONITORING DEI DATI (3/3)

Good channel



Bad channel



- ▶ 3072 istogrammi come questo con un allarme sulla posizione del fronte di salita
- ▶ Il sistema di allarmi è interamente scritto in Python

NON SOLO ROOT...

Plain python

```
A = [[1,1], [0,1]]  
B = [[2,0], [3,4]]  
  
print A  
print  
print B  
print  
for i in range(2):  
    for j in range(2):  
        print A[i][j] > B[i][j]
```

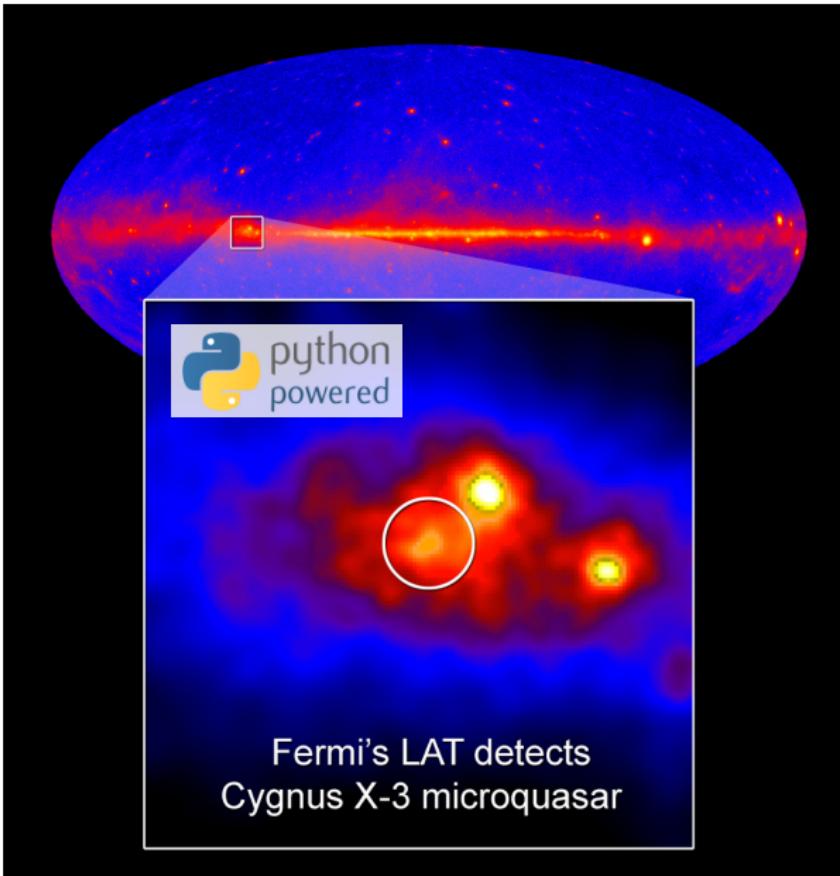
numpy

```
import numpy  
  
A = numpy.array( [[1,1], [0,1]] )  
B = numpy.array( [[2,0], [3,4]] )  
  
print A  
print  
print B  
print  
print A > B
```

```
[[1, 1], [0, 1]]  
[[2, 0], [3, 4]]  
  
False  
True  
False  
False
```

```
[[1 1]  
 [0 1]]  
  
[[2 0]  
 [3 4]]  
  
[[False True]  
 [False False]]
```

E LO SCOPO DEL GIOCO È...



CONCLUSIONI

- ▶ I Fisici non sono necessariamente buoni programmati
 - ▶ (Molto spesso il contrario)
 - ▶ Un linguaggio di *scripting* semplice e moderno come Python è potenzialmente una buona scelta
- ▶ I Fisici amano fare cose estremamente complicate
 - ▶ In generale non si può rinunciare a: velocità, efficienza ottimizzazione del codice
- ▶ Python è omnipresente in Fermi
 - ▶ DAQ e monitoring durante la costruzione
 - ▶ Beam test
 - ▶ Monitoring dei dati
 - ▶ Analisi scientifica
- ▶ Scrivere in C/C++ le parti critiche del codice, utilizzare librerie in C/C++ esistenti (ROOT, Qt, numpy) usare Python (od un altro linguaggio di scripting) come *glue language*
 - ▶ Si è rivelato un buon modello per Fermi

COLOPHON



This work is licensed under a
Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported
License

