First Results with HIJING++ on High-energy Heavy Ion Collisions

Speaker: Gergely Gábor Barnaföldi, Wigner RCP of the H.A.S.

Group: GGB, G. Bíró, Sz.M. Harangozó, W.T. Deng, M. Gyulassy, G.Y. Ma,
O. Nieberl, P. Lévai, G. Papp, X.N. Wang, B.W. Zhang

THOR CA15213 1st WG Meeting 2017, FIAS, Frankfurt am Main, Germany, 19-20th January 2017

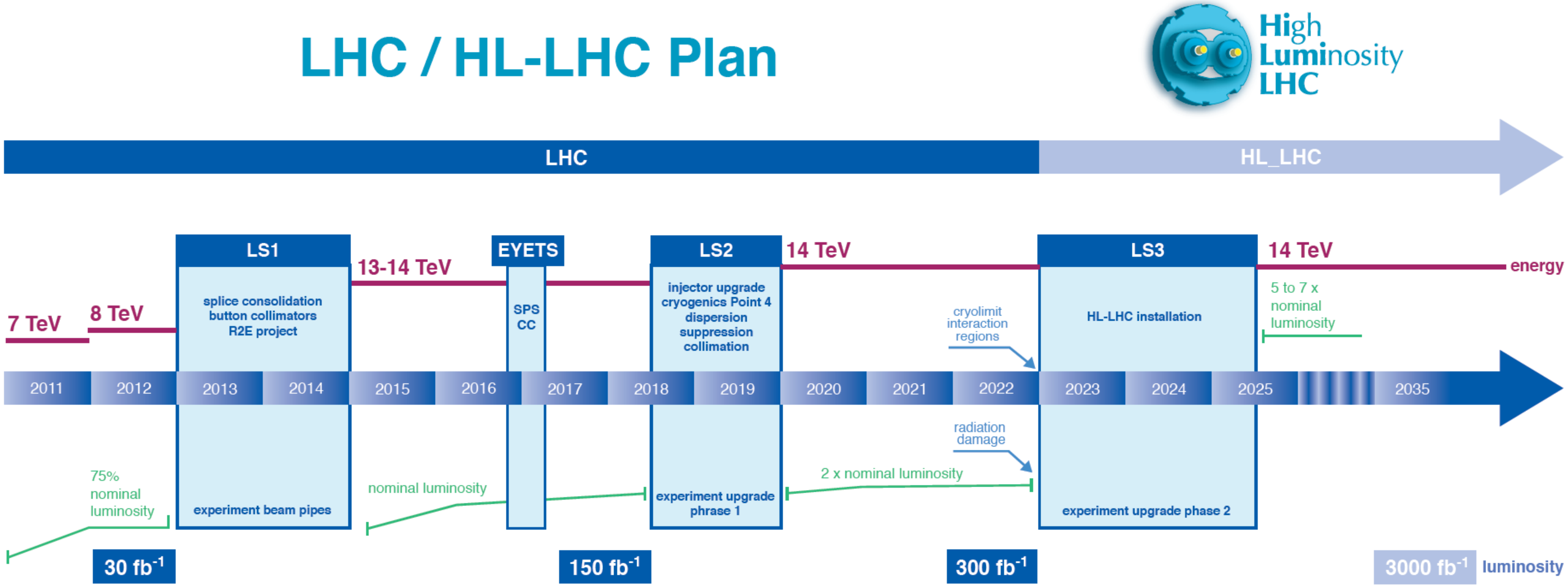
Outline

- Motivation for HIJING++
- Technical details of the HIJING++
  - The structure of the program
  - Simulation framework
- First results
  - Code validation in proton-proton collisions
  - New improvement: Scale-dependent HIJING shadowing
- Outlook...

MOTIVATION

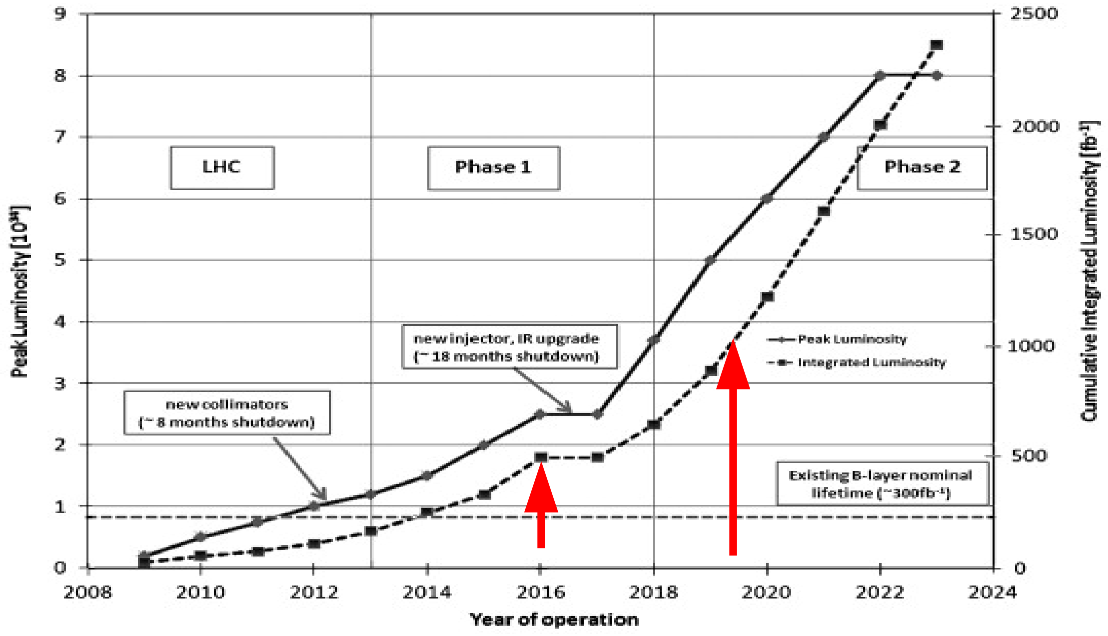
HI data from the Large Hadron Collider

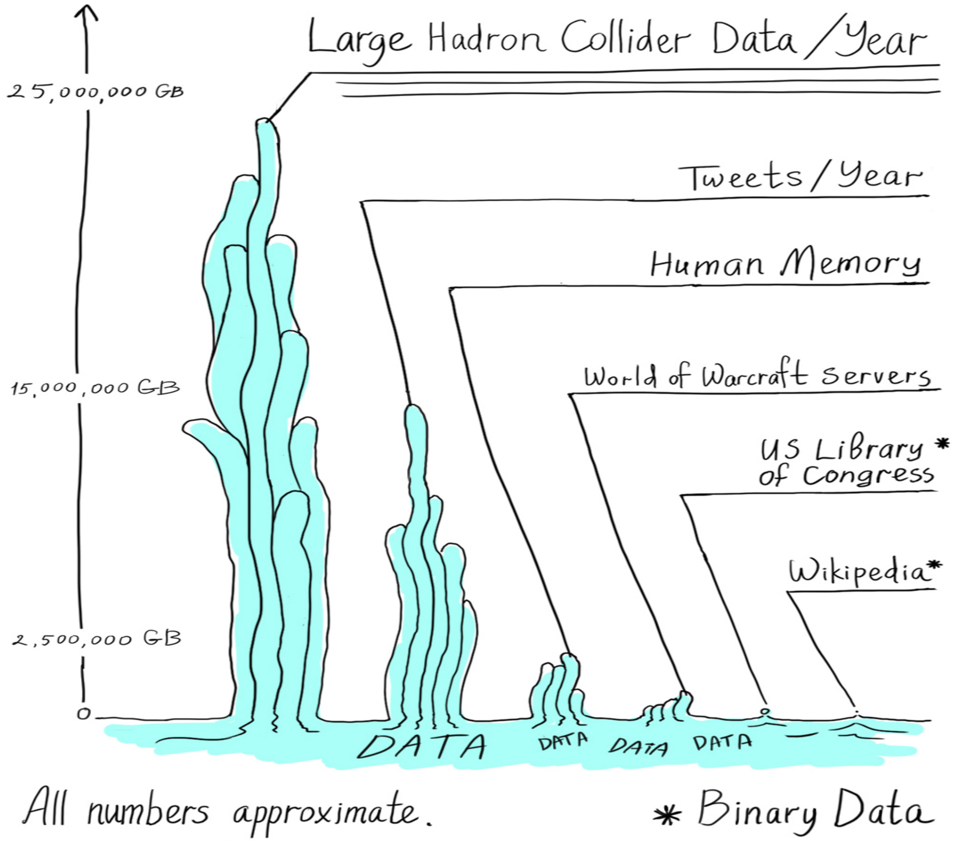
- LHC upgrades & theories required more and faster HI simulations



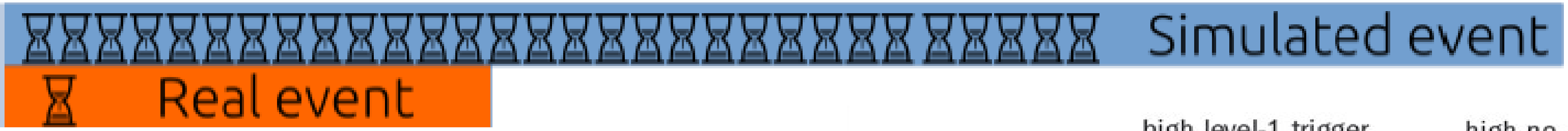
HI data from the Large Hadron Collider

- WLCG – Worldwide LHC Computing GRID:
  - LHC made 15-20 PB data per year
  - ...and now before HL-LHC 2PB/day

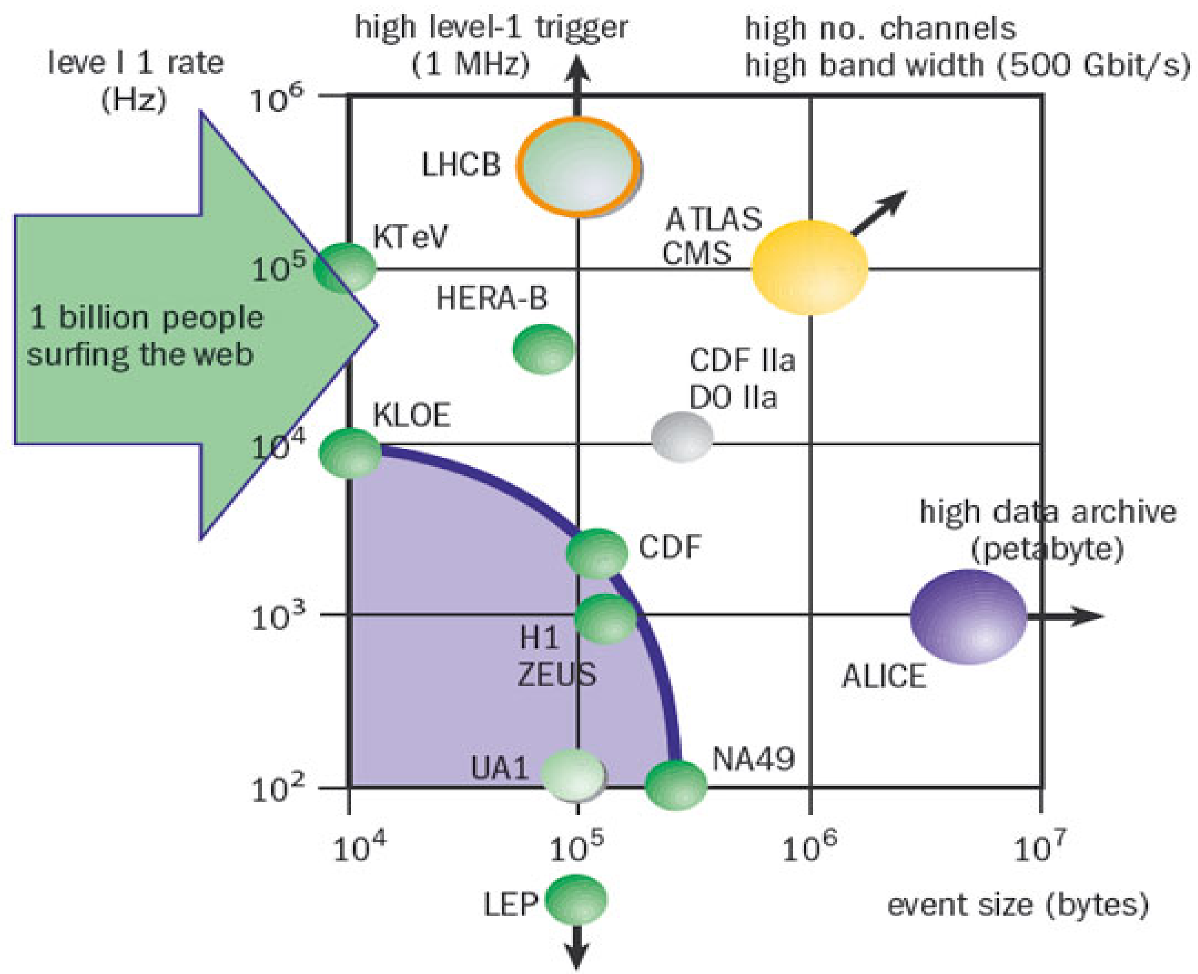




More data: motivation for fast computing at CERN



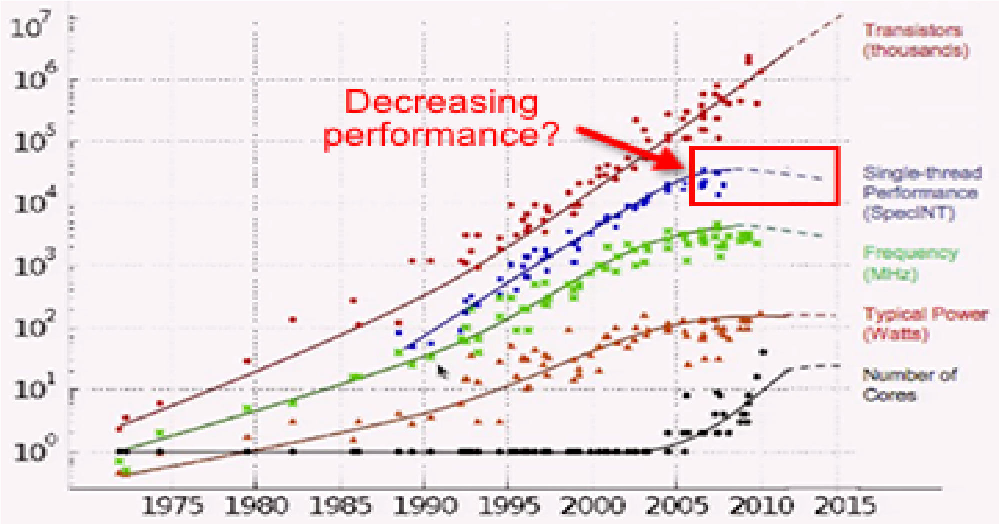
- Ideal: amount of simulated data ≈ real data
  - Number of events at LHC: $\mathcal{O}(10^8)$ / s
  - Necessary time for Monte Carlo with ALICE geometry: $3.8\ ms/track$
- Necessary time to simulate 1 s of ALICE data: $\mathcal{O}(days)$



Fast computing = parallel computing

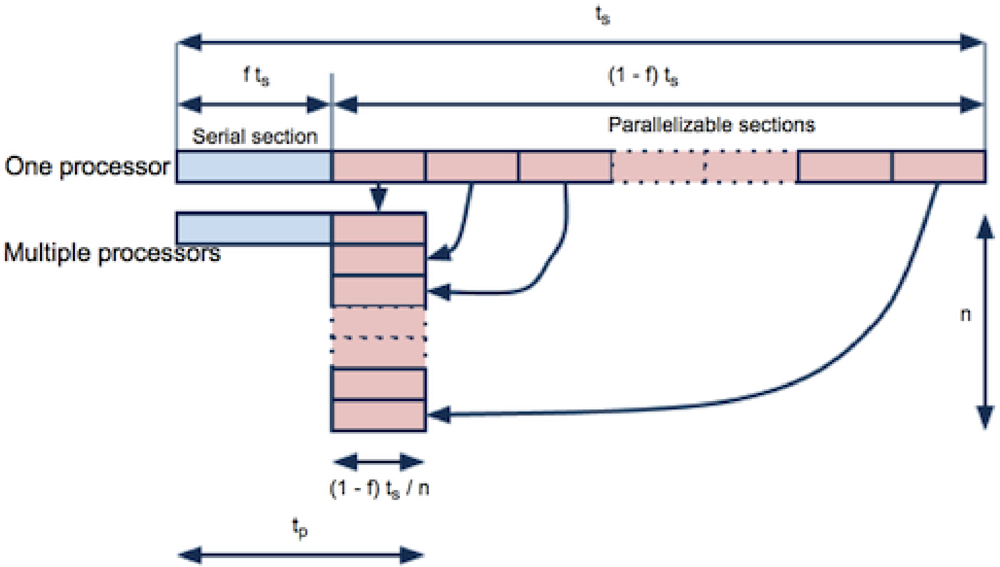
- Moore's law:

Every 2^{nd} year the number of transistors (integrated circuits) are doubled in computing hardwares.



- Amdalh's law:

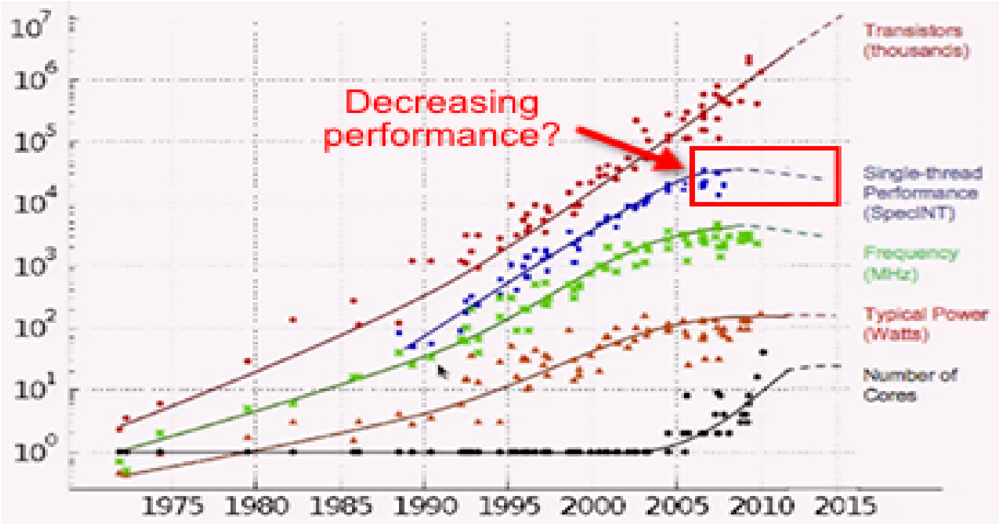
The theoretical speedup is given by the portion of parallelizable program, p, & number of processors, N, is:



Fast computing = parallel computing

- Moore's law:

Every 2nd year the number of transistors (integrated circuits) are doubled in computing hardwares.



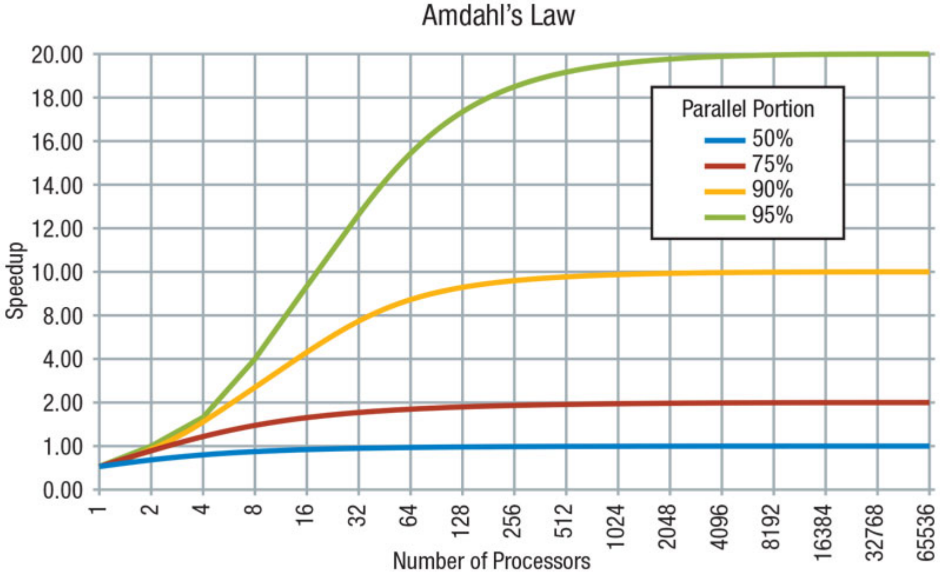
- Amdalh's law:

The theoretical speedup is given by the portion of parallelizable program, p, & number of processors, N, is:

$$\text{Speedup}(N) = \frac{1}{(1-P)+\frac{P}{N}}$$

Serial part of job = 1 (100%) - Parallel part

Parallel part is divided up by N workers



HIJING++

(C++ based HIJING version 3.1 with parallel opportunities)

The HIJING++

HIJING(Heavy-Ion Jet INteraction Generator)

易經

Bagua (eight simbols)

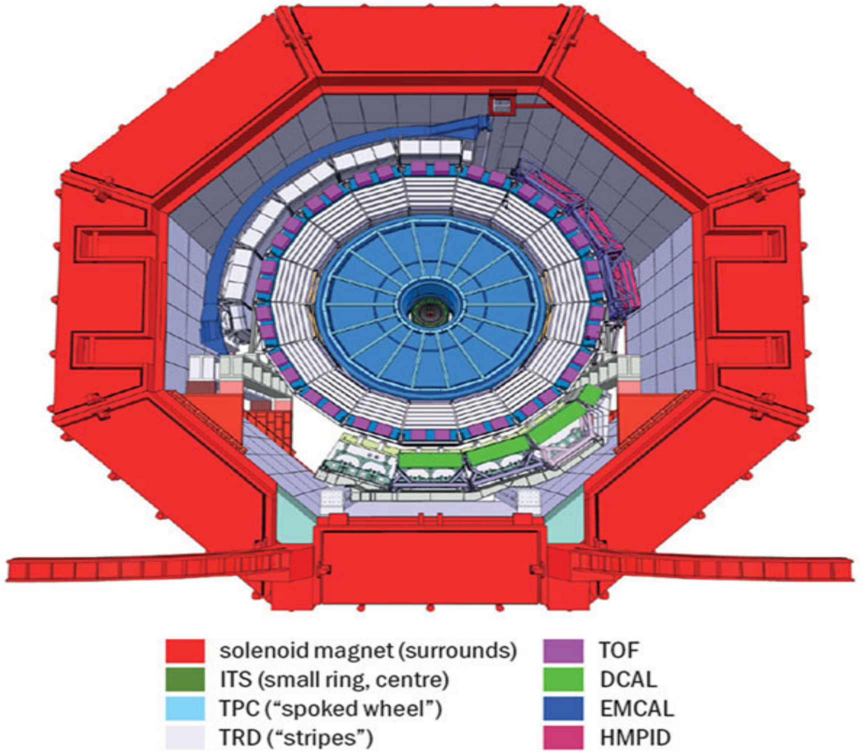
fundamental principles of reality

adjoint representation 8 of *SU(3)*

The HIJING++

HIJING(Heavy-Ion Jet INteraction Generator)

易經



Bagua (eight simbols)

fundamental principles of reality

adjoint representation 8 of *SU(3)*

What is HIJING???

It is a BIG mess....

What is HIJING???

- We all prepared well…
  to follow the ruler/z
  to add new things as well…

The HIJING++

HIJING(Heavy-Ion Jet INteraction Generator)

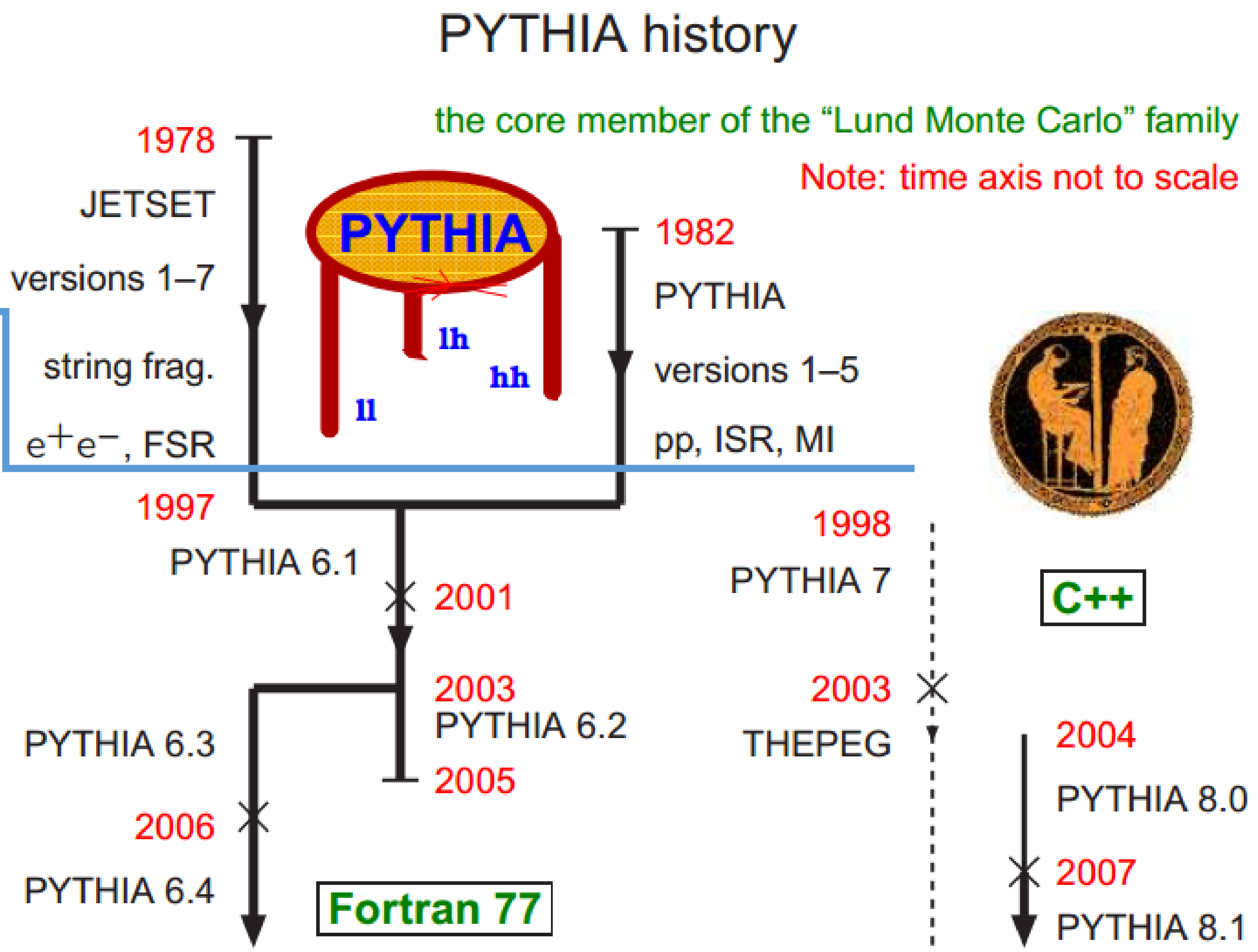
- HIJING versions
  - FORTRAN v1.36, v2.553
  - C++ v3.0

Reasons to use C++

Object oriented language:
Hierarchy, Modularity

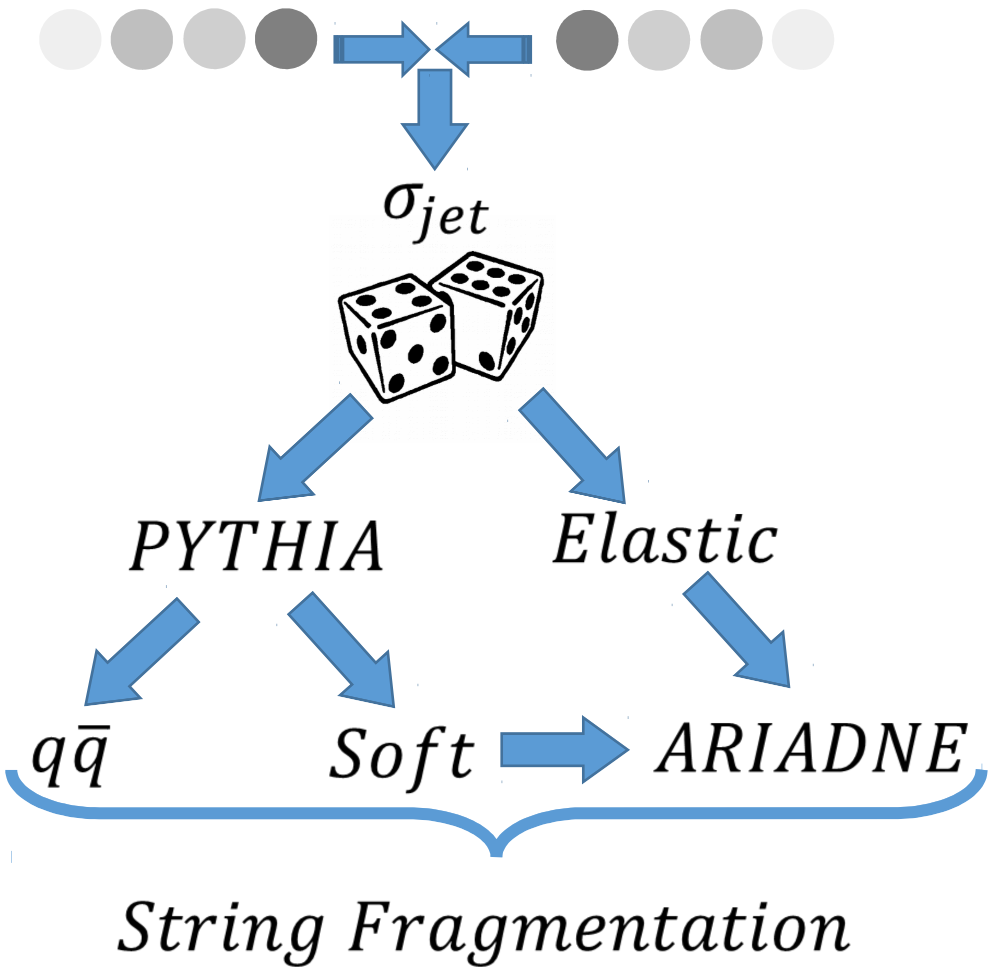
C++11/14 has thread support
and compatibility with OpenCL

G.(



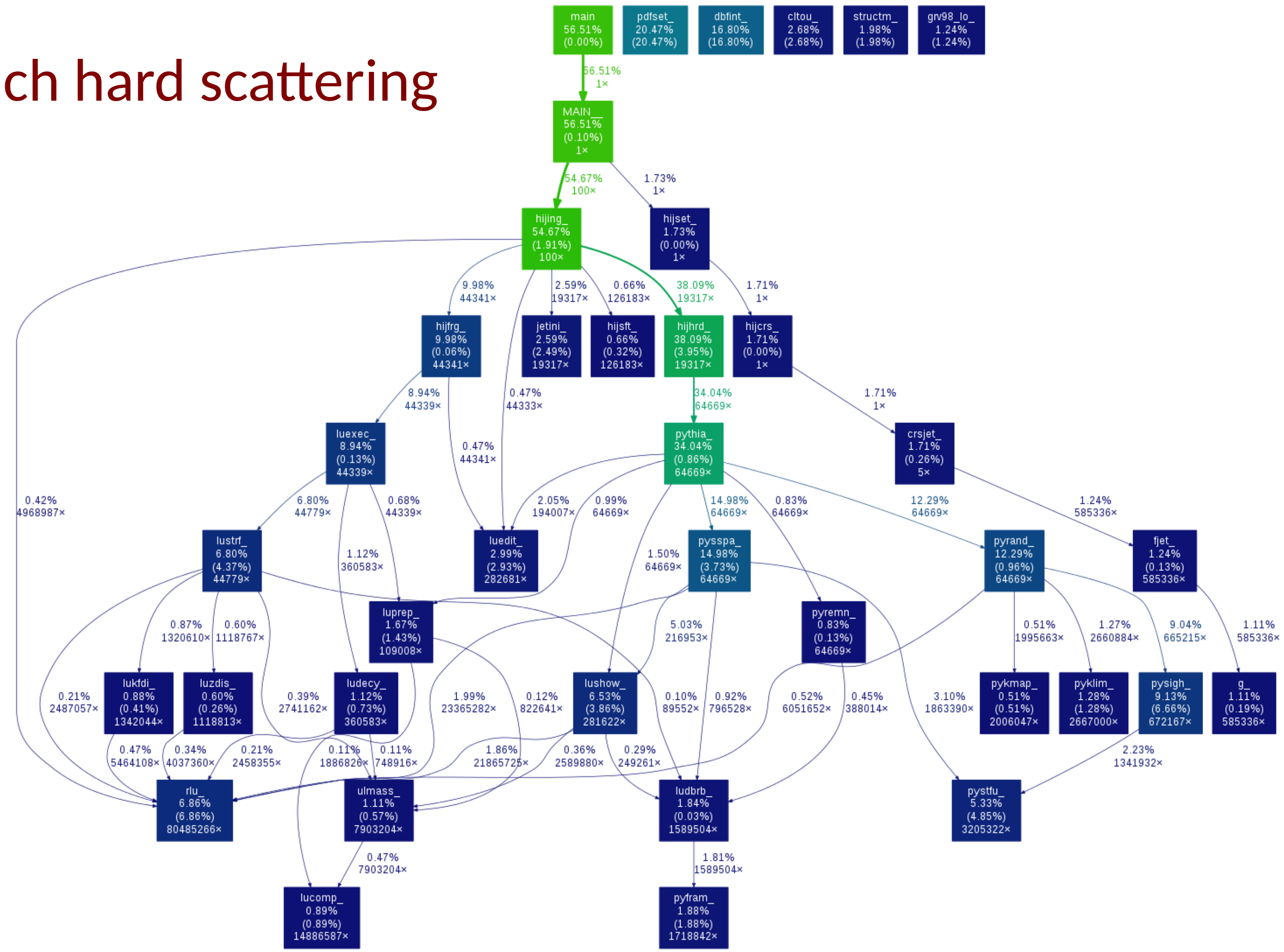
Program Flow – in general

- Pair-by-pair nucleon-nucleon events
- Multiple soft gluon exchanges between valence- and di-quarks
- String hadronization according to Lund fragmentation scheme



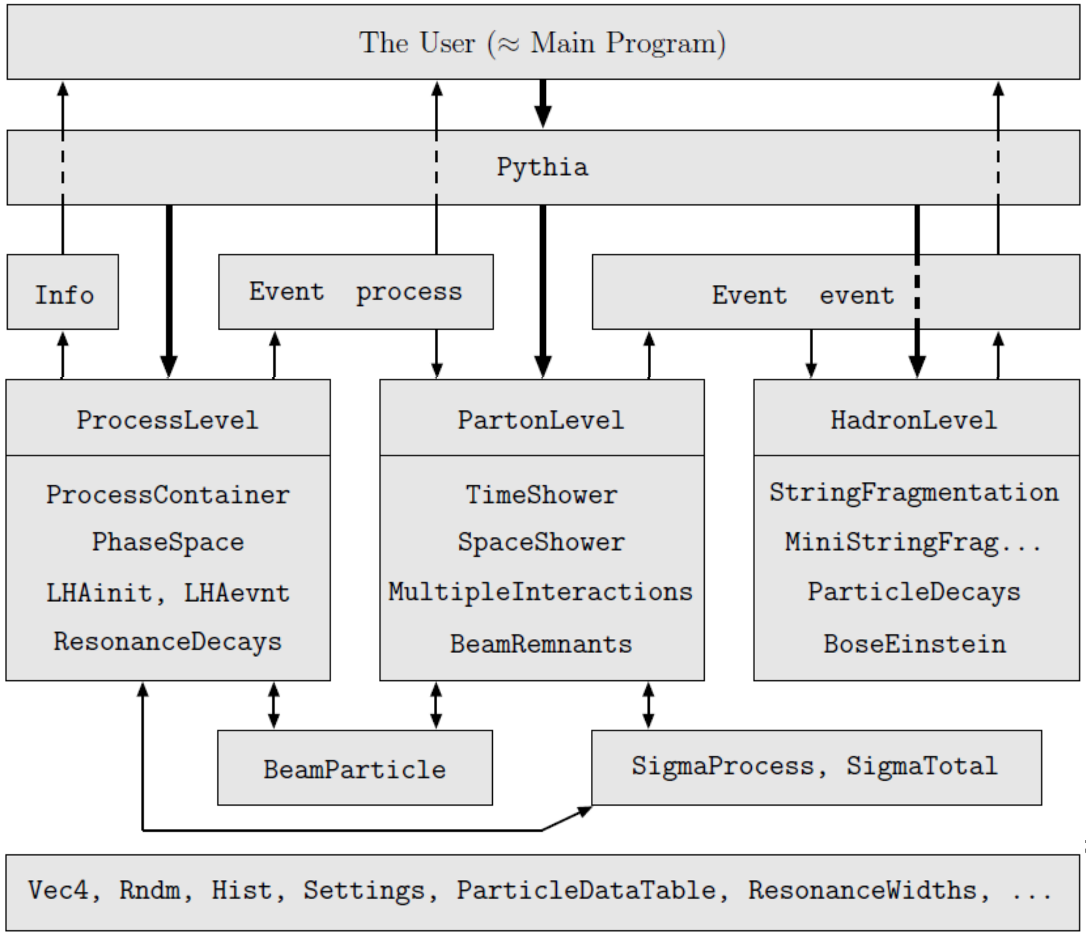
Program Flow – old one

- Generation of kinetic variables for each hard scattering with Pythia 5.3
- Multiple soft gluon exchanges between valence- and di-quarks
- String hadronization according to Lund fragmentation scheme



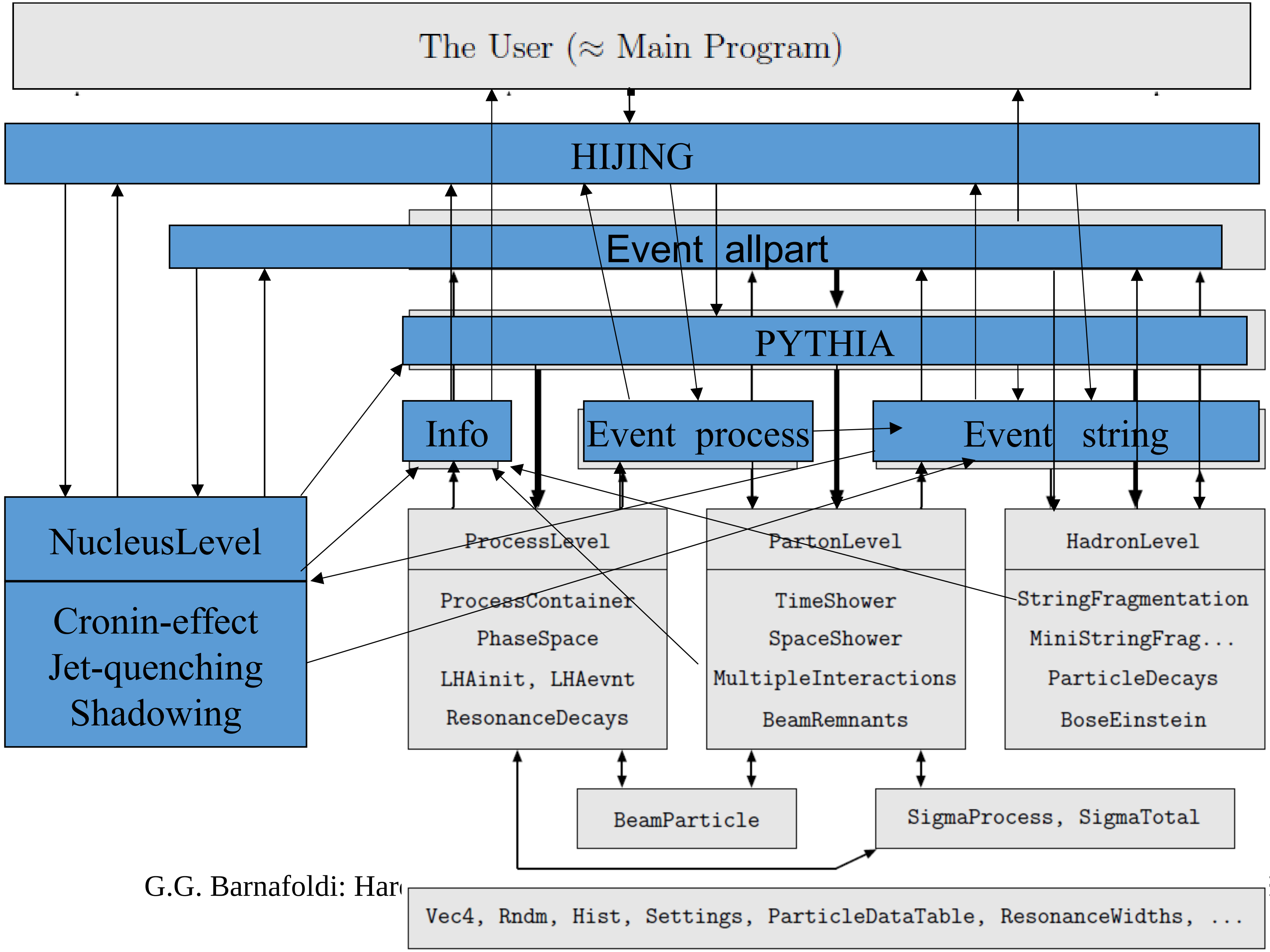
Program Structure

- Pythia8 namespace containers
- Structure similarities
- Actual program flow is more complicated



Program Structure

- Pythia8 namespace containers
- Structure similarities
- Actual program flow is more complicated



Program Structure
Hijing class

```
namespace Pythia8 {

    class Hijing {

    public:

        Info            info;
        Rndm            rndm;
        Settings        settings;
        ...

    private:

        HardCollision   hijhard;

        SoftScatter     hijsoft;

        Fragmentation   fragmentation;

        NucleonLevel    nucleonlevel;

        ...
    }
}
```

- Processes ordered in class hierarchy
- Former common blocks ✉ class variables
- Processes called through object functions

// Class for handling the hard collisions

// Class for handling the soft interactions

// Class for handling the Lund string fragmentation

// Class for the nuclear effects

The 'main' example

Usual form kept for regular users

FORTRAN

```
PROGRAM TEST
...
PARM(1) = 'DEFAULT'
VALUE(1) = 80060
CALL PDFSET(PARM, VALUE)
CALL GetDesc()
...

CALL HIJSET(EFRM, FRAME, PROJ, TARG, IAP, IZP, IAT, IZT)

N_EVENT=1E6
DO 200 IE = 1, N_EVENT
    CALL HIJING(FRAME, BMIN, BMAX)
200 CONTINUE

STOP
END
```

Form also similar to Pythia 8.x

C++

```
#include "Hijing.h"

using namespace Pythia8;

int main() {
    Hijing hijing("../xmldoc", true);
    hijing.readString("PDF:pSet = LHAPDF6:GRV98lo");

    bool okay = hijing.init(200.0, frame,
                "A", "A", 197, 79, 197, 79);
    if (!okay) return 1;

    int MaxEvent = 1e6;
    for (int iEvent = 0; iEvent < MaxEvent; ++iEvent)
        hijing.next(frame, 0.0, 0.0);
}
```

Program Features

- Calculation by improved models
- Pythia like prompt Histogram creation
- CPU level Parallel computing

```
const std::size_t num_threads = std::thread::hardware_concurrency();
for (std::size_t i = 0u; i < num_threads; ++i){
    async_hijing.at(i) = std::unique_ptr<Hijing>(new Hijing);
}
for (std::size_t I = 0; I < num_threads; ++I){
    ...async run...
    okay[I] = async_hijing[I]->init(...);
    for (int iEvent = 0; iEvent < numEvent; ++iEvent)
        async_hijing[I]->next(...);
    for (int i = 0; i < async_hijing[I]->event.size(); ++i)
        if(...) hist[I]->fill(...);
}
```

- AliRoot compatibility (planned)

Dependencies & External packages

- Boost

  sudo apt-get install libboost-all-dev

- LHAPDF 6

  ./configure –prefix=$HOME/.../share/LHAPDF

  make all

  insert downloaded PDF library to $HOME/.../share/LHAPDF

  optionally modify **pdfsets.index**, add set if needed

  export LD LIBRARY PATH=<library path>

- Pythia 8

  *./configure --with-lhapdf6-lib=$HOME/.../lib *
  --with-boost-lib=/usr/lib/x86_64-linux-gnu
  make –j4

- GSL (optional)

  HIJING **make** option

Data Analysis

```
#include "Hijing.h"
using namespace Pythia8;

int main() {
    Hist dndpT("dn/dpT for charged particles", 100, 0., 10.);
    ofstream ch_file("ch_hist.dat");
    ...
    bool okay = hijing.init(efrm, frame, proj, targ,
                            aproj, zproj, atarg, ztarg);
    if (!okay) return 1;
    int MaxEvent = 1e6;
    for (int iEvent = 0; iEvent < MaxEvent; ++iEvent) {
        hijing.next(frame, bmin, bmax);
    for (int i = 0; i < hijing.event.size(); ++i)
    if (hijing.event[i].isFinal() && hijing.event[i].isCharged())
        dndpT.fill(hijing.event[i].pT());
    }
    dndpT *= 1.0 / MaxEvent;
    cout << dndpT;
    dndpT.table(ch_file);
    ...
    return 0;
}
```

Pythia 8 Histogram class available

Selection has to be made for every particle

Hist::fill(double Input);

Normalization

standard output and file output both provided

FIRST TEST & RESULTS WITH HIJING++

Performance tests: runtime

- Runtime new vs. old

  Single core run:

  - PP is slower, but this is the effect of PHYTHIA8
  - Difference getting less as more complex HIC we have.

  Multi-core run:

  - Due to the MPI support several times faster
  - Better performance in HIC than in small systems (10^5 evts)

(gain)	FORTRAN	C++ single core		C++ parallel	
pp	0.2640s	0.5055s	-91.5%	0.0044s	5055%
pA	3.5090s	6.274s	-46.4%	0.0514s	6826%
AA	397.96s	482.28s	-21.2%	5.688s	6896%

```
integer::beg, end, rate
call system_clock(beg,rate)

(end - beg)/real(rate)
```

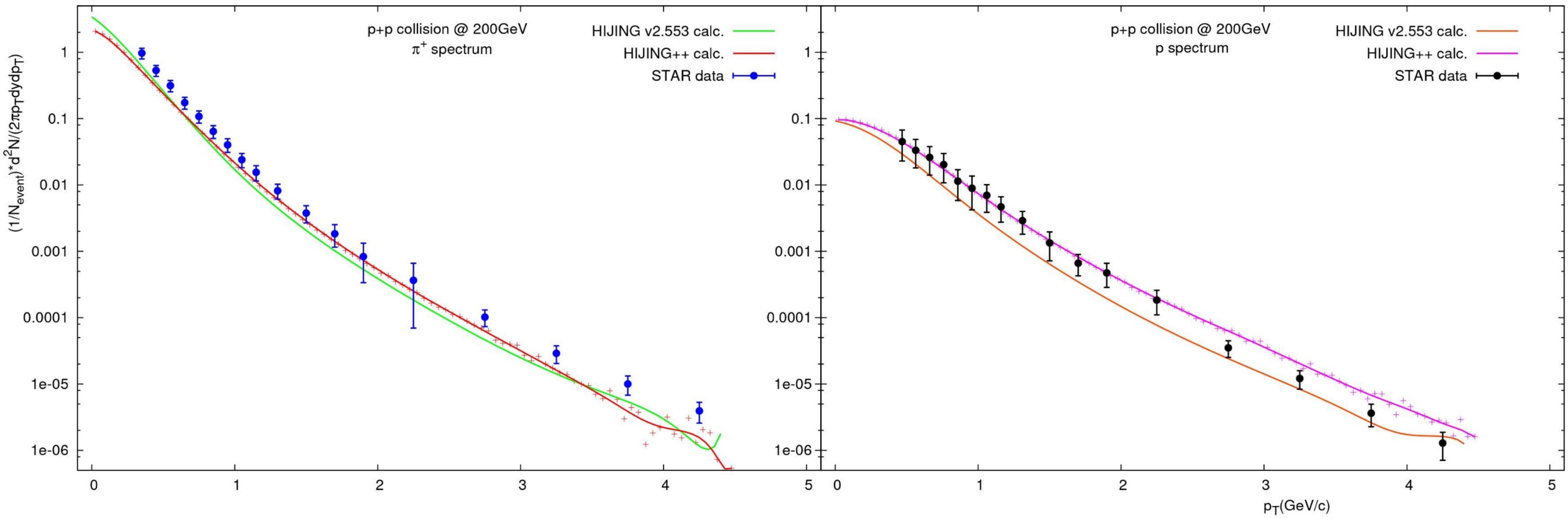
```
#include <chrono>

auto start = std::chrono::high_resolution_clock::now();

double runtime = std::chrono::duration_cast<std::chrono::milliseconds>
(end.time_since_epoch() - start.time_since_epoch()).count();
```

Physics tests: pp collisions

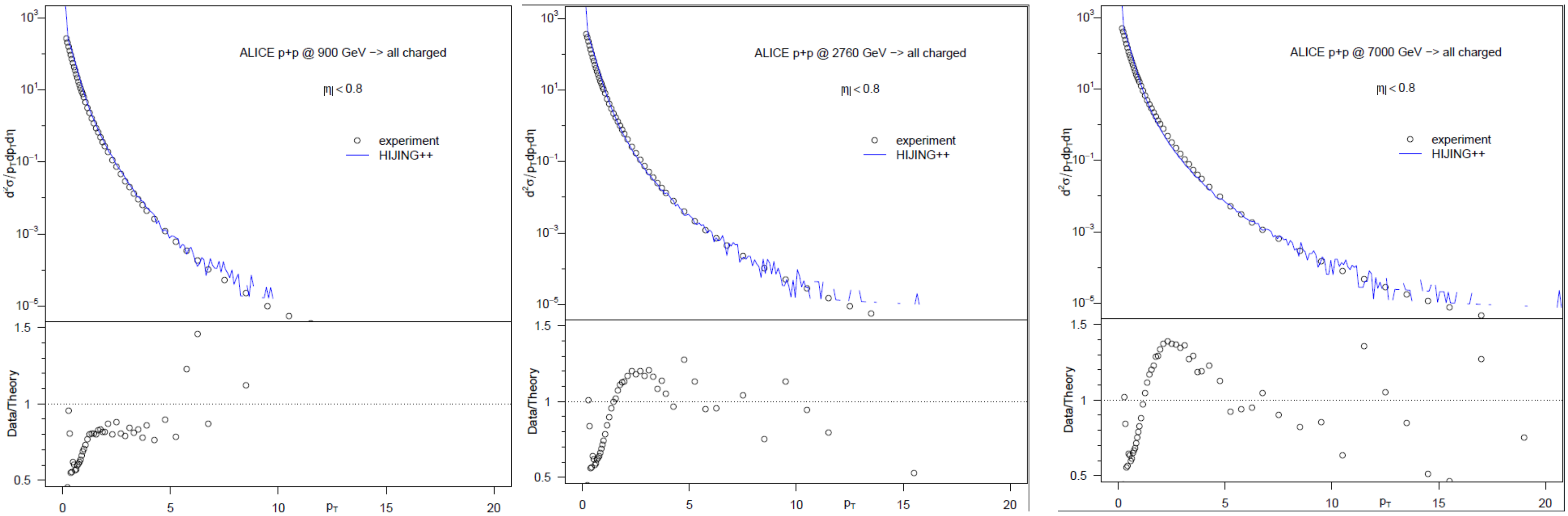
Code validation with „old” version and data in pp at RHIC energies



Data: STAR Collaboration, Phys.Lett. B637 page 161-169 (2006)

Physics tests: pp collisions

Code validation with „old" version and data in pp at LHC energies

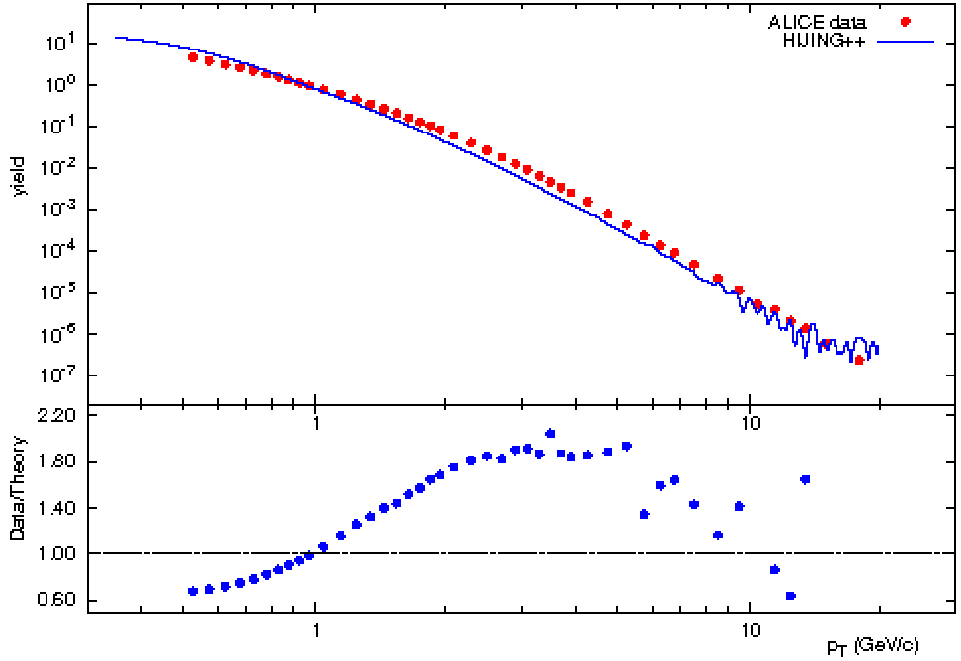


Data: ALICE Collaboration, Eur. Phys. J. C73 2662 (2013)

Physics tests: pA (ongoing)

The pA features

- PYTHIA5.x vs PYTHIA8.x plenty of new physics: try to avoid "double counting" in HIJING++
- GRV98 were include to LHAPDF6 for backward compatibility
- Nuclear shadowing: many kinds available, new Q-dependent version of the HIJING shadowing parametrization.
- Jet Quenching: several models
- Soft QCD radiation (incl. since v1.36), new calls for ARIADNE



Data: ALICE data @ 5.02 TeV p+Pb

Physics tests: shadowing with *Q*²

Old HIJING shadowing (nPDF)

- PDF are based on GRV98lo
- A & Isospin effects: averaging p/n PDFs
- Geometry (b-dependence) is given by a simple geometry factor
- NO Multiple scattering included!
- NO Q^2-scale parametrization for the shadowing function, $S_{a/A}(x)$
- QCD scale were factorized in PDF ONLY
- NO flavor dependence included only q/g

$$f_{a/A}(x,Q^2) = S_{a/A}(x)\left[\frac{Z}{A} f_{a/p}(x,Q^2) + \left(1-\frac{Z}{A}\right) f_{a/n}(x,Q^2)\right]$$

$$\begin{aligned} Sq(x,A) &= 1 + 1.19\log^{1/6} A(x^3 - 1.2x^2 + 0.21x) \\ &- s_q(A^{1/3}-1)^{0.6}(1-3.5\sqrt{x})e^{-x^2/0.01} \\ S_g(x,A) &= 1 + 1.19\log^{1/6} A(x^3 - 1.2x^2 + 0.21x) \\ &- s_g(A^{1/3}-1)^{0.6}(1-1.5x^{0.35})e^{-x^2/0.00} \end{aligned}$$

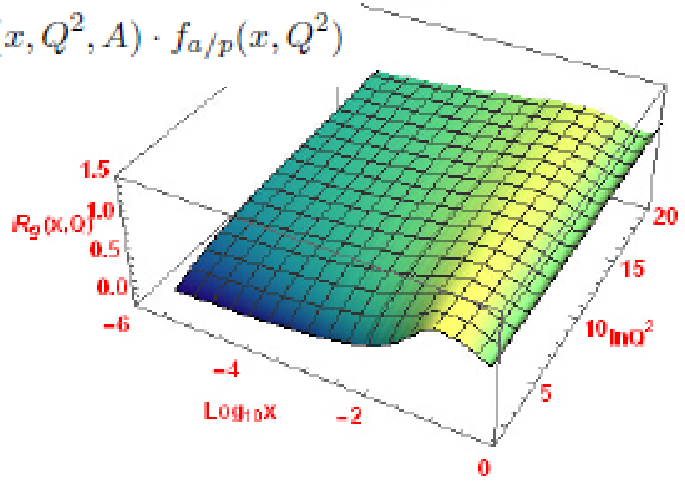
$$s_a(b) = s_a\frac{5}{3}\left(1-\frac{b^2}{R_A^2}\right)$$

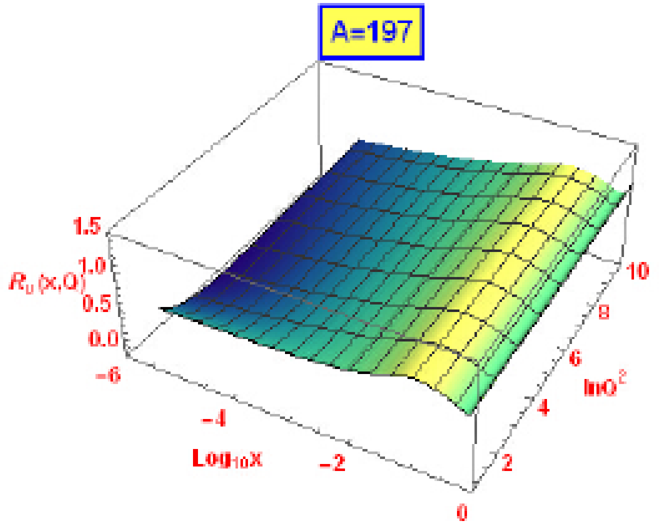
$$R_A = 1.12A^{1/3}$$

Physics tests: shadowing with Q^2

NEW HIJING shadowing (nPDF) $f^{(s)}_{a/p}(x,Q^2,A) = S_a(x,Q^2,A)\cdot f_{a/p}(x,Q^2)$

- PDF are based on GRV98lo
- A & Isospin effects: averaging p/n PDFs
- Geometry (b-dependence) is given by a simple geometry factor
- NO Multiple scattering included!
- Old Q^2-scale parametrization is fixed for Q^2=2 GeV^2 , DGLAP evolution is calculated by HOPPET code, $S_{a/A}(x,Q^2)$
- DGLAP Q^2 evolution were parametrized
- ONLY g/q difference in the corrections





Physics tests: shadowing with Q^2

NEW HIJING shadowing (nPDF)

- PDF are based on GRV98lo
- A & Isospin effects: averaging p/n PDFs
- Geometry (b-dependence) is given by a simple geometry factor
- NO Multiple scattering included!
- Old Q^2-scale parametrization is fixed for Q^2=2 GeV^2 , DGLAP evolution is calculated by HOPPET code, $S_{a/A}(x,Q^2)$
- DGLAP Q^2 evolution were parametrized
- ONLY g/q difference in the corrections

$$f^{(s)}_{a/p}(x,Q^2,A) = S_a(x,Q^2,A) \cdot f_{a/p}(x,Q^2)$$

$$S_q(x,Q^2,A) = f^{(s)}_{u/p}(x,Q^2,A) / f_{u/p}(x,Q^2)$$

$$\begin{aligned}
S_g(x,y,A) &= \left(1 + f_1(y)x^{0.35} + \frac{f_2(y)}{-0.8 + \exp f_3(y)\sqrt{x}}\right) e^{-\frac{x^{\cdot}}{0.0}} \\
f_1(y) &= -1.42 - \frac{0.7314}{y^2} + \frac{1}{y} + 0.0011y^2 \\
f_2(y) &= -0.1628 e^{\frac{0.006}{y^{10}}} (y - \log 2)^{0.03} \\
f_3(y) &= 0.126 + 0.0096y + \frac{0.78}{(y - \log 2)^{0.8}}
\end{aligned}$$

$$\begin{cases} f_1(x) = x^3 1.2x^2 + 0.21x \\ f_2(x) = (11.5x^0.35)e^{x/0.004} \end{cases}$$

$$y = \log Q^2$$

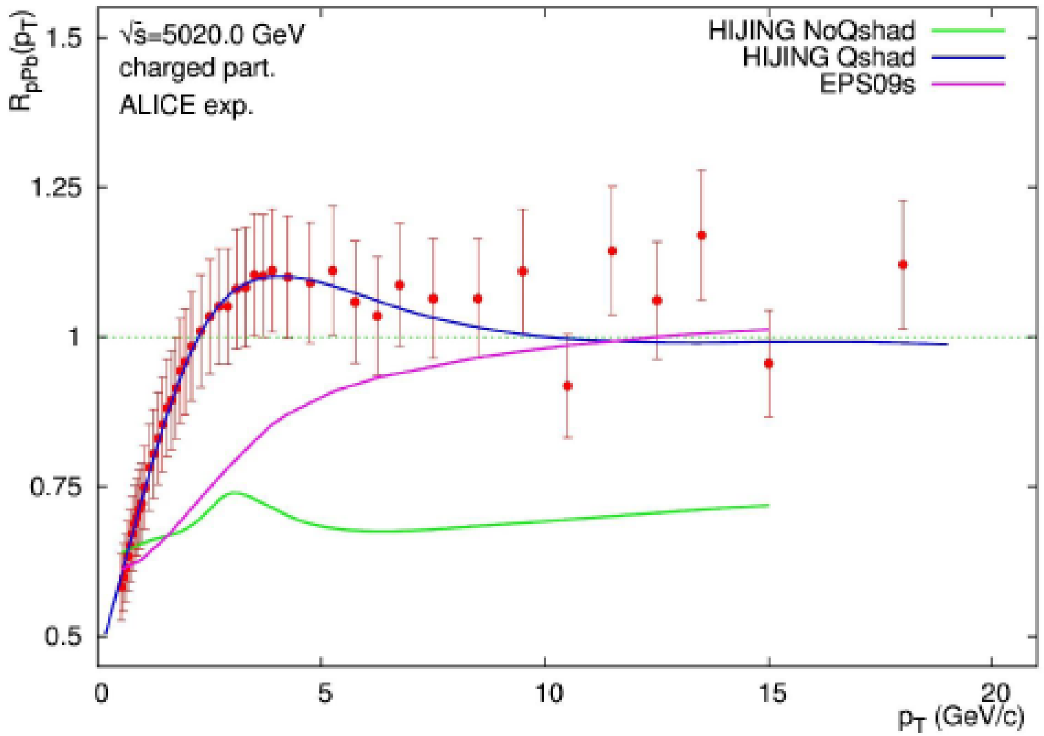
Physics tests: $R_{pPb}(p_T)$

NEW HIJING shadowing (nPDF)

- Test: Nuclear Modification Factor

$$R_{pA}(p_T) = \frac{dN_{pA}/dyd^2p_T}{\langle N_{\rm bin}\rangle dN_{pp}/dyd^2p_T}$$

- Theory (HIJING 2.553)
  - Q^2-dependent HIJING shadowing with multiple scattering
  - EPS09s with NO multiple scattering
- Data:
  - ALICE data @ 5.02 TeV p+Pb



Summary

- Big Data era is here: it is time for parallel computing in HIC
  - High Luminosity LHC will come after 2018
  - Simulation and theory need faster MC

- HIJING++
  - Coding from FORTRAN → C++ has been done
  - Performance (parallel) tests are ongoing and promising
  - Physics tests has been started (preliminary results)
  - Step-by-step reconsidering of nuclear effect (shadowing with Q^2, jet quenching)

stay tuned….

Proposed Events

- 12th High-pT Physics for the RHIC and LHC Era – “HpT4LHC”
  - Date: 2-5 October, 2017
  - Organizers: Bergen, at the University of Bergen (UiB), Western Norway University of Applied Sciences (HVL)
  - Web: under construction – Maybe WG3 meeting?

- New perspectives on Neutron Star Interiors
  - Date: 9-13 October, 2017
  - ECT*, Trento, Italy
  - Web: http://www.ectstar.eu/node/2230