HIJING++, OOP concepts and simple parallelization

[1,6]Sz. M. Harangozó, [2,3]G. Y. Ma

Supervisors:
[1]G. G. Barnaföldi, [6]G. Papp , [2,3]B. W. Zhang

Founders:
[2,3,4]X-N. Wang, [5]M. Gyulassy

Contributor:
[7]W. T. Deng

1、Wigner Research Centre for Physics, Hungarian Academy of Sciences
2、Institute of Partical Physics, Central China Normal University
3、Key Laboratory of Quark & Lepton Physics, China
4、Lawrence Berkeley National Laboratory
5、Columbia University in the City of New York.
6、Department of Theroretical Physics, Eötvös Loránd University
7、Huazhong University of Science and Technology.

Introduction

- HIJING(Heavy-Ion Jet INteraction Generator)

易經

Bagua (eight simbols)

fundamental principles of reality

adjoint representation 8 of *SU(3)*

Introduction

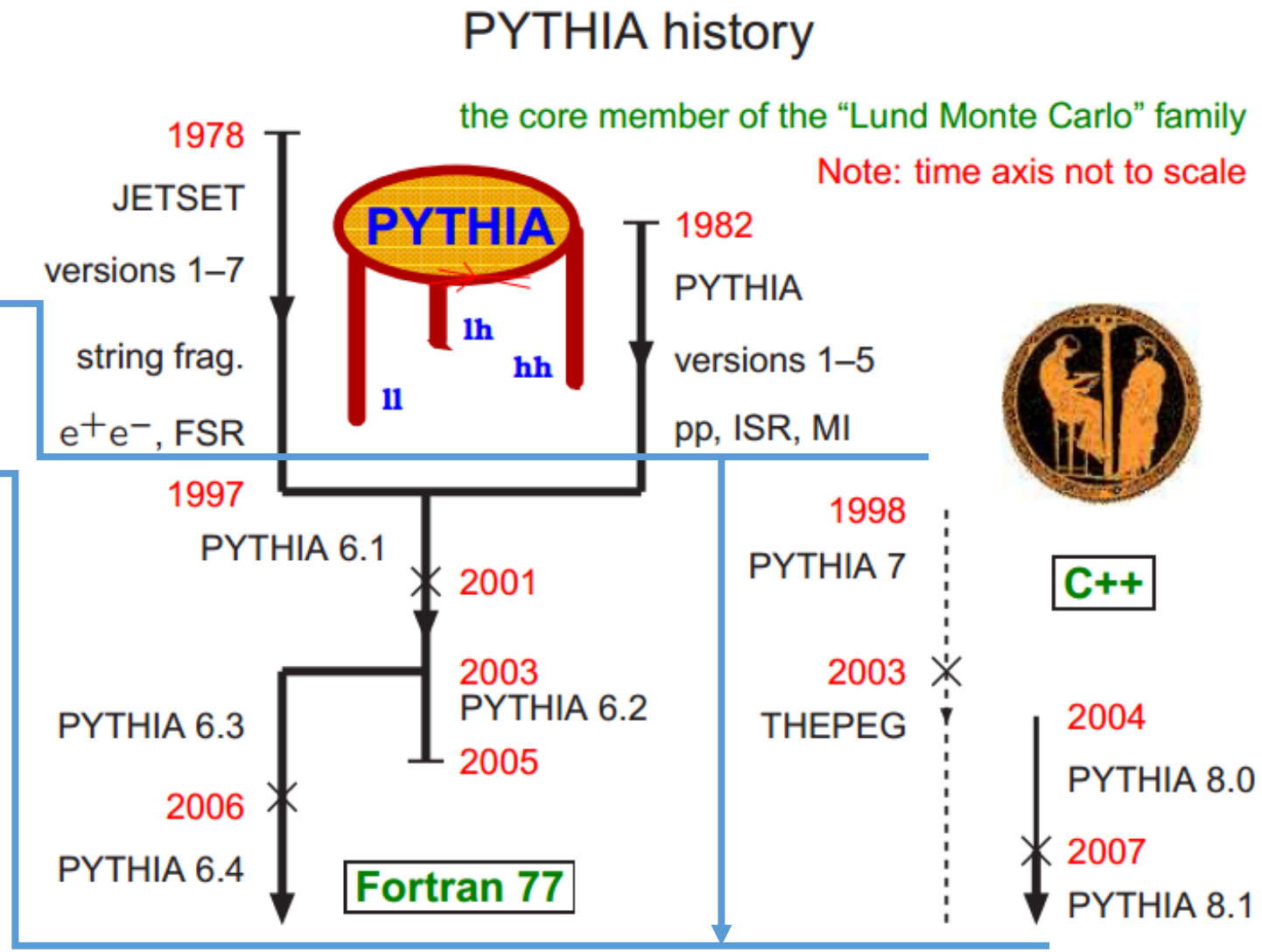
- HIJING(Heavy-Ion Jet INteraction Generator)

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

Reasons to use C++

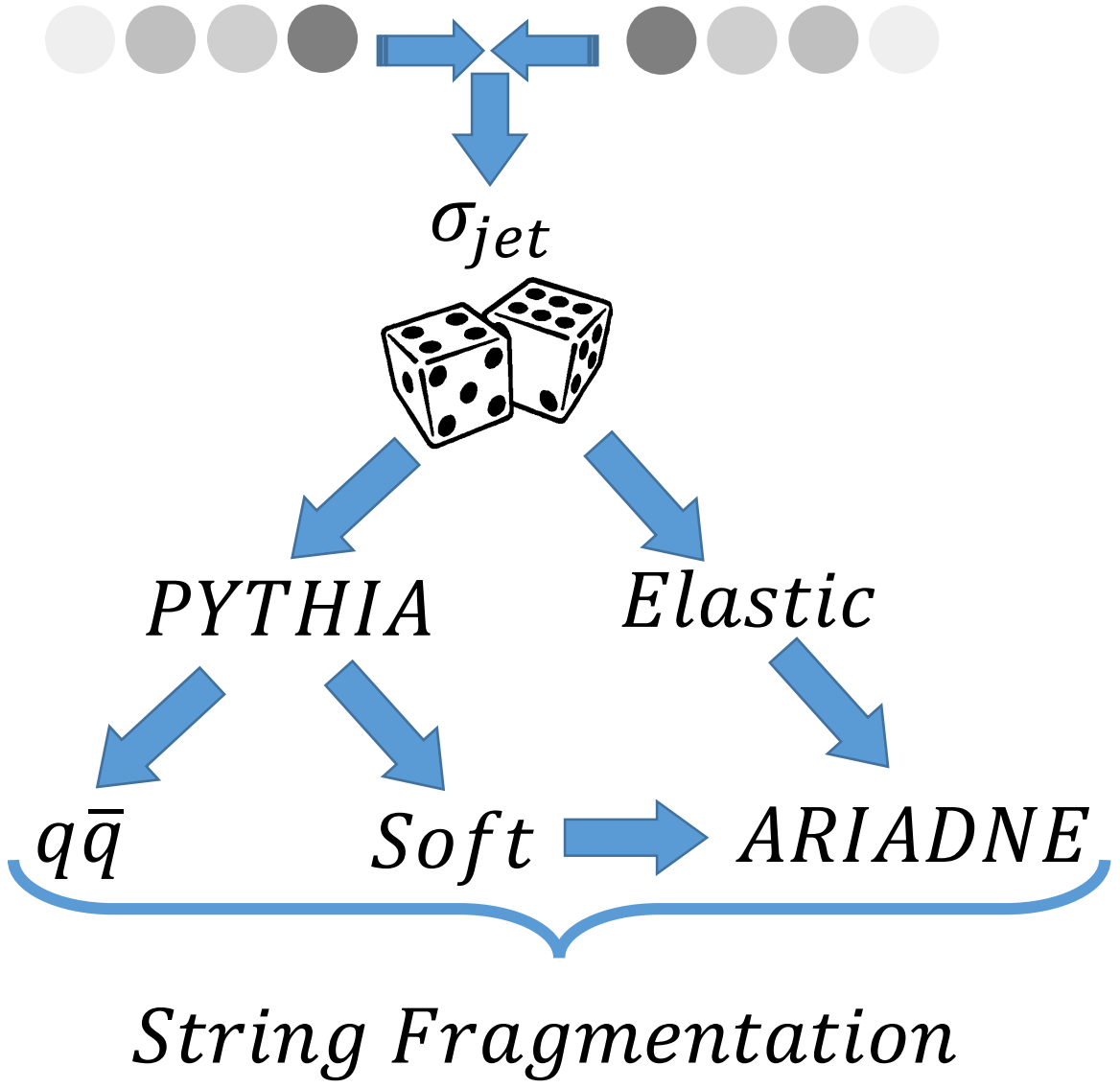
Object oriented language: Hierarchy, Modularity

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



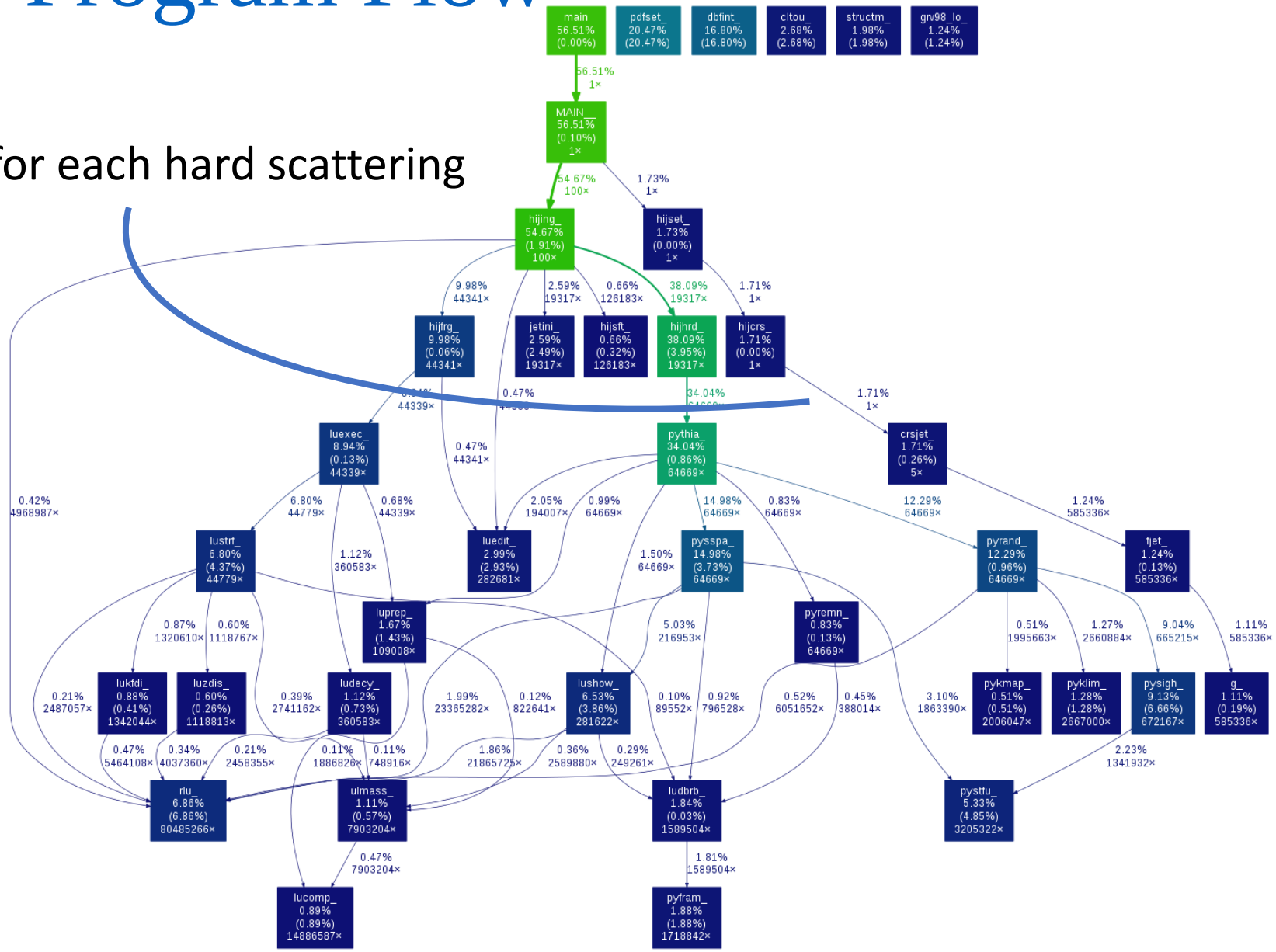
Program Flow

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



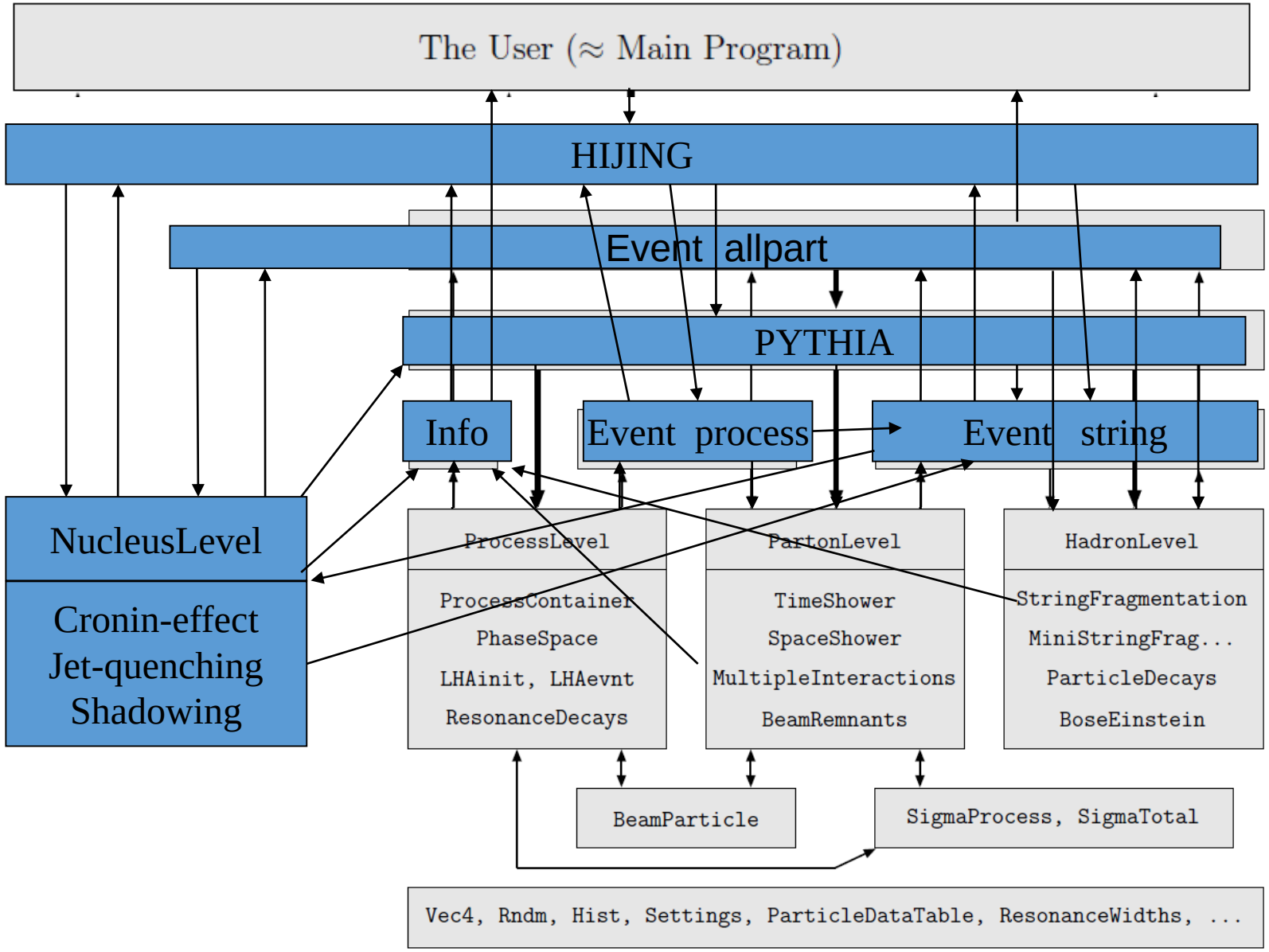
(Old) Program Flow

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



(New) Program Structure

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



How should a class look like?

- Single responsibility principle
- Open/closed principle
- Liskov substitution principle
- Interface segregation principle
- Dependency inversion principle

Design Patterns

```
class HijPDF : public PDF {
public:
  HijPDF(int idBeamIn, string setName, Info* infoPtr) :
  PDF(idBeamIn), id(-1), pdf(0), extrapol(false)
  {init(setName, (int)0, infoPtr);}
  ~HijPDF();
  void setExtrapolate(bool extrapolIn){extrapol = extrapolIn;}
  void setb(double bIn);
  void setA(double AIn);
  void shadow(bool do_shadowIn);
  void xfUpdate(int idIn, double xIn, double Q2In);

private:
  bool do_shadow, extrapol;
  double aax,abx,sax;
  double sq,sq, rrg, rrq;
  int id;
  ::LHAPDF::PDF *pdf;
  ::LHAPDF::Extrapolator *ext;

  void init(string setName, int member, Info* infoPtr);
};
```

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

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

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

- AliRoot compatibility (planned)

```
unsigned int thread_num = ::std::thread::hardware_concurrency();
// INITIALIZATION
::std::vector<::std::thread> consumer_threads;
  for (unsigned int i = 0; i < thread_num; ++i)
  consumer_threads.emplace_back
  (&ParaHijing<int,Selector>::init,async_hijing.at(i).get());
for(auto consumer : consumer_threads) consumer.join();
// CLEAR THREAD WORKLOAD
consumer_threads.clear();
// RUN EVENT GENERATION
for (unsigned int i = 0; i < thread_num; ++i)
  consumer_threads.emplace_back
  (&ParaHijing<int,Selector>::run,async_hijing.at(i).get());
for (auto consumer : consumer_threads) consumer.join();
consumer_threads.clear();
```

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

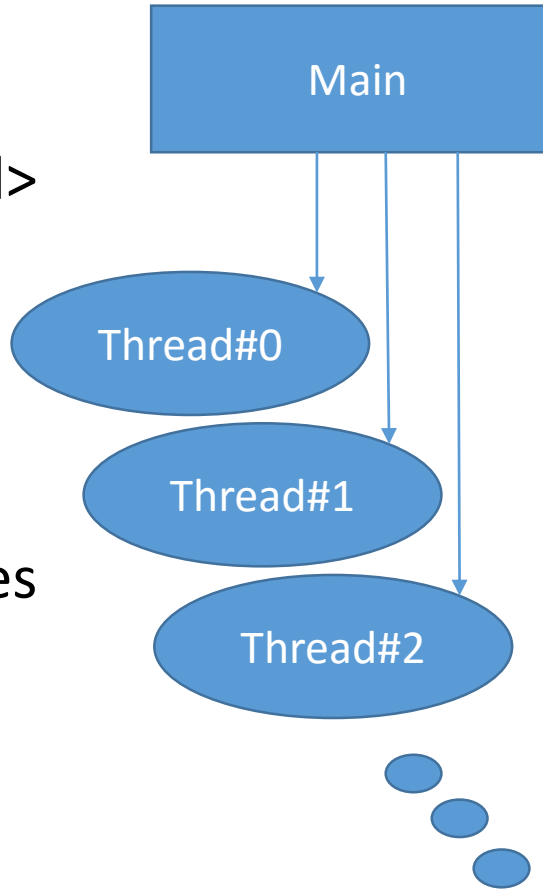
Hist::fill(double Input);

Normalization

standard output and file output both provided

Multithreading

- The Naive Way
  - STL <future> <thread>
  - Hijing instance encapsulated
  - Each thread calculates the **same** number of events
  - λ-expression can be confusing

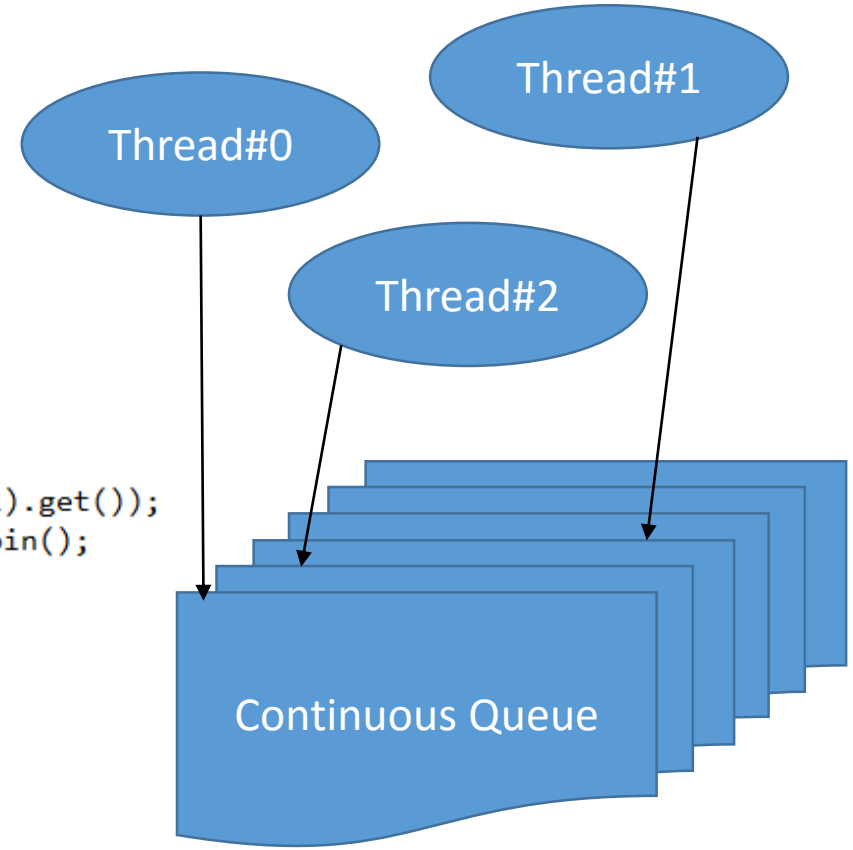



```
const std::size_t num_threads = std::thread::hardware_concurrency();
int numEvent = MaxEvent / (int)num_threads;
MaxEvent = numEvent * (int)num_threads;

std::vector<std::future<void>> handles;
std::vector<std::unique_ptr<Hijing>> async_hijing(num_threads);
for (std::size_t i = 0u; i < num_threads; ++i) {
    async_hijing.at(i) = std::unique_ptr<Hijing>(new Hijing);
    ... // PARAMETRIZATION
}
for (std::size_t I = 0; I < num_threads; ++I)
    handles.push_back(std::async(std::launch::async,
        [I, num_threads, &async_hijing ... &hist]() {
        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){
            ... // FILLING HISTOGRAMS, SUM AT END
            }
        }
}));
for (auto& handle : handles) handle.wait();
```

Multithreading

- The Clever Way
  - STL <mutex> <thread>
  - Producer – Consumer design
  - New ParaHijing class with parameterless functions
  - Singleton Queue

```
struct Selector {
public:
    void selector(Event &event, HijHist &hist) { ... }
};
// MAIN
vector<::std::unique_ptr<ParaHijing<int,Selector>>> async_hijing(thread_num);
HijHist hist( ... );
WorkQueue<int> que(mu,cond,hist);
Selector selector;

que.fill(MaxEvent,NumEvent);

for (unsigned int i = 0 ; i < thread_num; ++i){
async_hijing.at(i) = std::unique_ptr<...>(new ...)
  async_hijing.at(i)->giveValues( ... );
}
::std::vector<::std::thread> consumer_threads;
for (unsigned int i = 0; i < thread_num; ++i)
  consumer_threads.emplace_back
(&ParaHijing<int,Selector>::run, async_hijing.at(i).get());
for (auto consumer : consumer_threads) consumer.join();
consumer_threads.clear();
```



Parallel Computing

- Message Passing Interface
  - Boost should be compiled with MPI support
  - HijHist class should be serialized
  - Serialization provides **save** opportunity

```
#include "HijHist.h"
#include <boost/mpi.hpp>

::boost::mpi::environment env;
::boost::mpi::communicator world;

world.barrier();
if (world.rank() == 0) {
    vector<::Pythia8::HijHist> hists;
    gather(world, hist, hists, 0);
    HijHist sum = ::std::accumulate(hists.begin(), hists.end(), 0);
    sum *= 1.0 / ((double)EventPerNode*world.size());
    cout << sum << endl;
} else {
    gather(world, hist, 0);
}
```

Runtime comparison

For 1e5 Events, 200 cores.

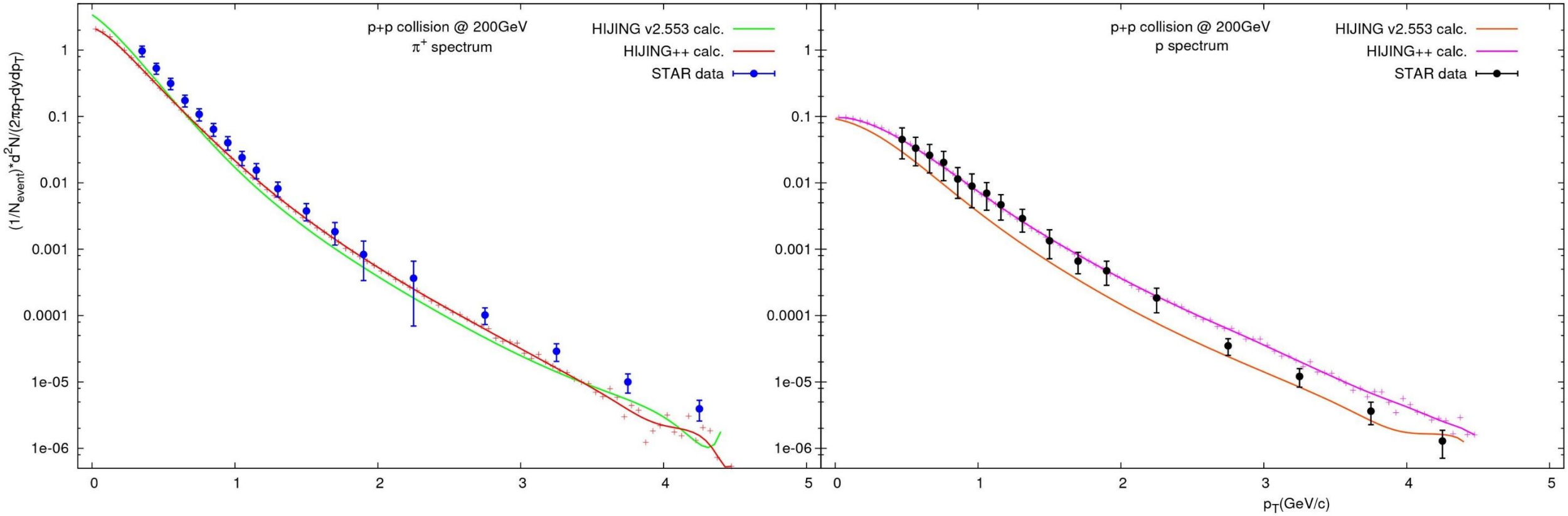
```
integer::beg, end, rate
call system_clock(beg,rate)

(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();
```

(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%

Apetizer plots for the RHIC era

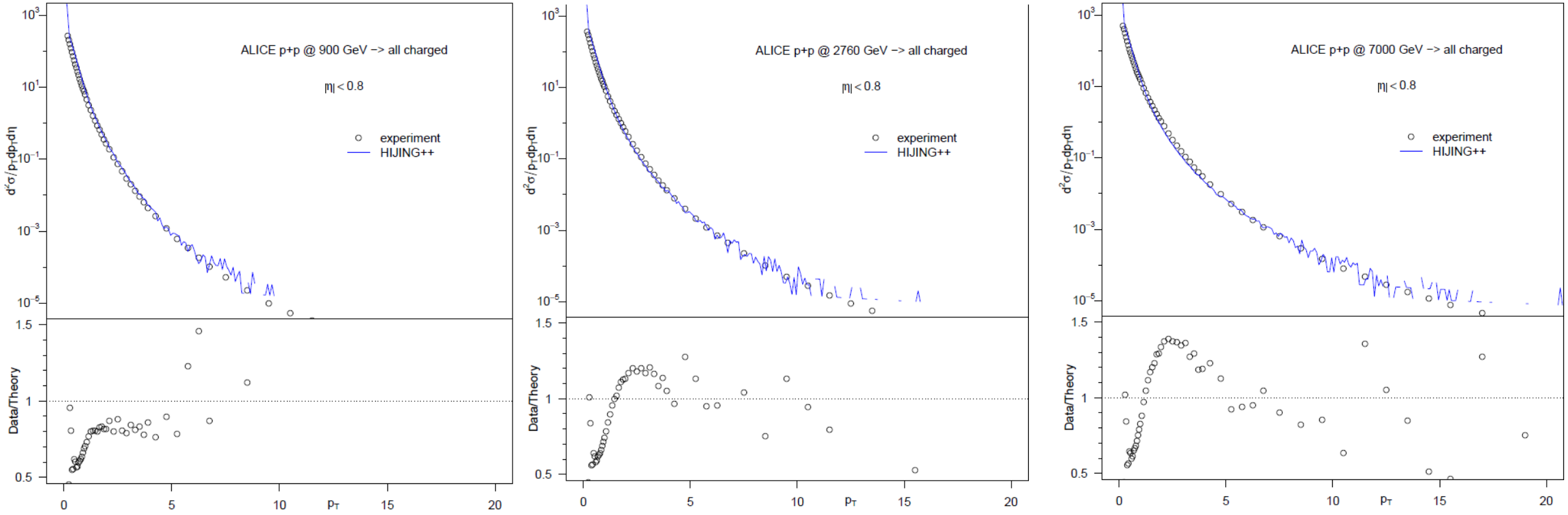
Code validation with „old” version and RHIC data



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

Apetizer plots for the LHC era

Code validation with LHC pp data at 900, 2760, 7000 GeV c.m. energies.



ALICE collaboration, Eur. Phys. J. C 73 2662 (2013)

Model Improvements

- Shadowing

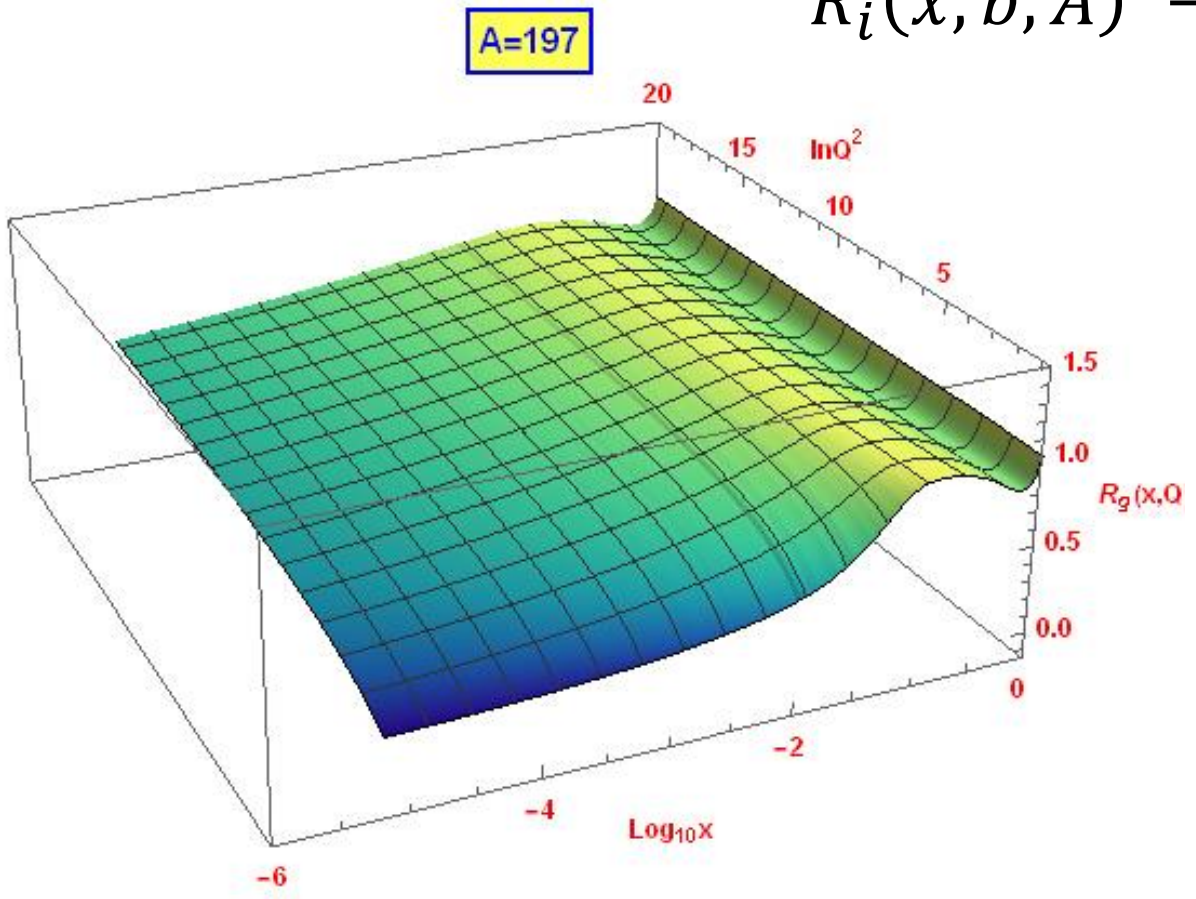
  HIJING 2.0 fits RHIC data well
  Improvements are needed for LHC energy

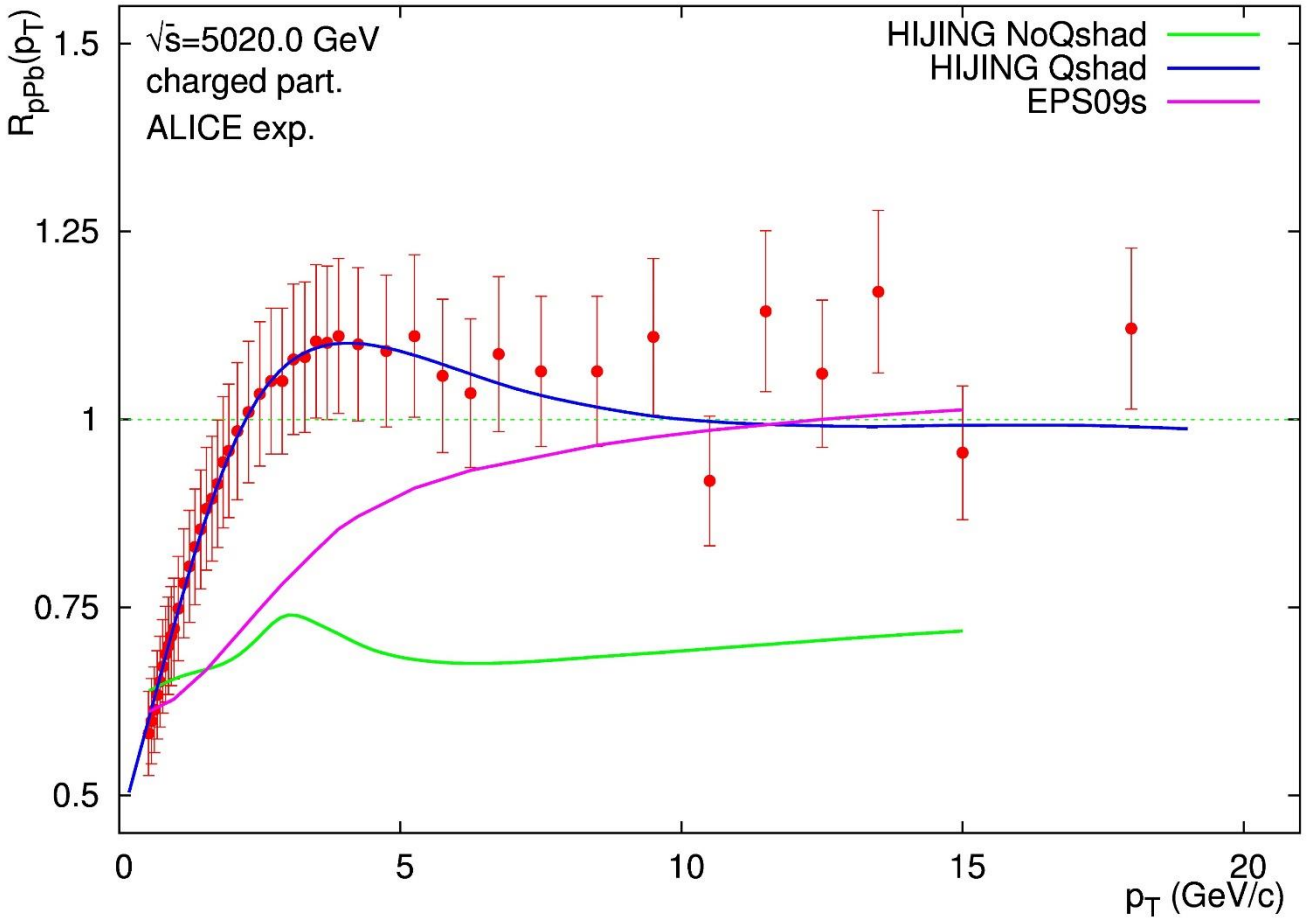
  $R_i(x, b) \rightarrow R_i(Q, x, b)$

- Jet-Quenching — Various models: accuracy ←→ speed
- Soft QCD radiation — updated ARIADNE calls
- (already implemented improvements since v1.36)

Updated Shadowing

$$R_i(x, b, A) \rightarrow R_i(Q, x, b, A)$$





- High-p_T region – Still investigated

Ongoing activities and future plans

- Ongoing activities (HIJING++ v3.0)
  - code/compatibility tests & tuning
  - performance test
  - new physics (Shadowing, Quenching)
  - parallel version

- Future plans (HIJINGv3.x)
  - online access – documentation
  - AliRoot compatibility
  - multi thread / GPU support
  - GUI

Thank you!

This work is supported by the Hungarian-Chinese cooperation grant No TéT 12 CN-1-2012-0016 and No. MOST 2014DFG02050, Hungarian National Research Fund (OTKA) grant NK106119.
We acknowledge the support of Wigner GPU laboratory, and discussions with Miklós Gyulassy, Xin-Nian Wang and Wei-Tian Deng.