Hadronic Cross-Sections

Witek Pokorski, Alberto Ribon

26.09.2013

Content

- current set of cross sections for physics lists
- improvements in the cross section implementation
- conclusion

FTFP/QGSP_BERT

- Replaced Wellisch nucleon-nucleus inelastic cross sections with Barashenkov-Glauber ones (in G4 9.6)
- Replaced CHIPS gamma-nuclear and electron-nuclear with Bertini + Fritiof&Preco (in G4 9.6)
    - Kept the CHIPS total cross sections
- Replaced CHIPS final-state for hyperon-nucleus and antihyperon-nucleus inelastic interactions with Bertini + Fritiof&Preco (in G4 9.5)
    - Kept the same CHIPS cross sections
- Replaced CHIPS (xsection & final-state) antiproton-nucleus inelastic interactions with Fritiof&Preco (in G4 9.6)
    - Glauber-Gribov-Uzhinsky-Galoyan cross section
- Replaced LEP (xsection & final-state) light ion – nucleus inelastic interactions with Binary + Fritiof&Preco (in G4 9.5)
    - Glauber-Gribov-Grichine cross sections
- Added light anti-ion – nucleus inelastic interactions with Fritiof&Preco (in G4 9.5)
    - Glauber-Gribov-Uzhinsky-Galoyan cross sections

Things to improve

- bad CPU performance of CHIPS-derived cross-sections
- bad design of CHIPS-derived cross-section
  - enormous use of ‘statics’
- use of ‘IsIso/ElementApplicable’ method
  - in my opinion should be entirely removed (redesigned)
- use (calculation) of ‘per isotope’ cross section when the precision is much lower then the dependance on N

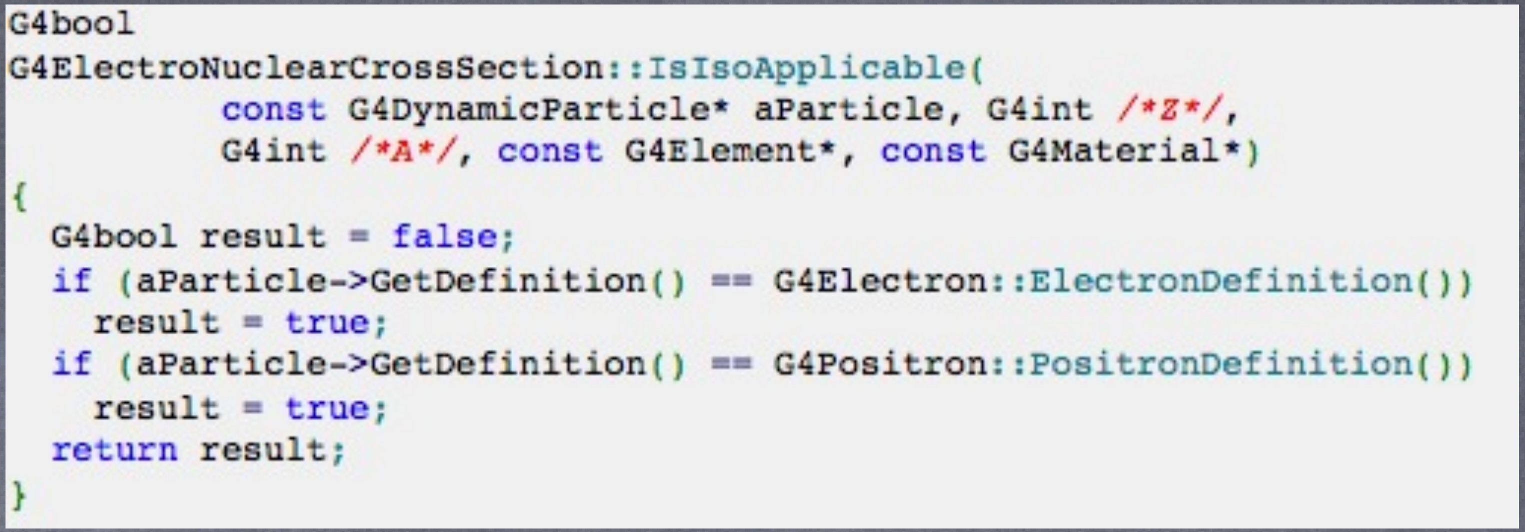
Bad performance of CHIPS-derived code

- redundant and multiple checks (for particle type, etc)
- looping through the cache with no ‘break’ when the entry is found
- very heavy calculation
  - could be approximated
    - use of fast log, etc
  - redundant operations

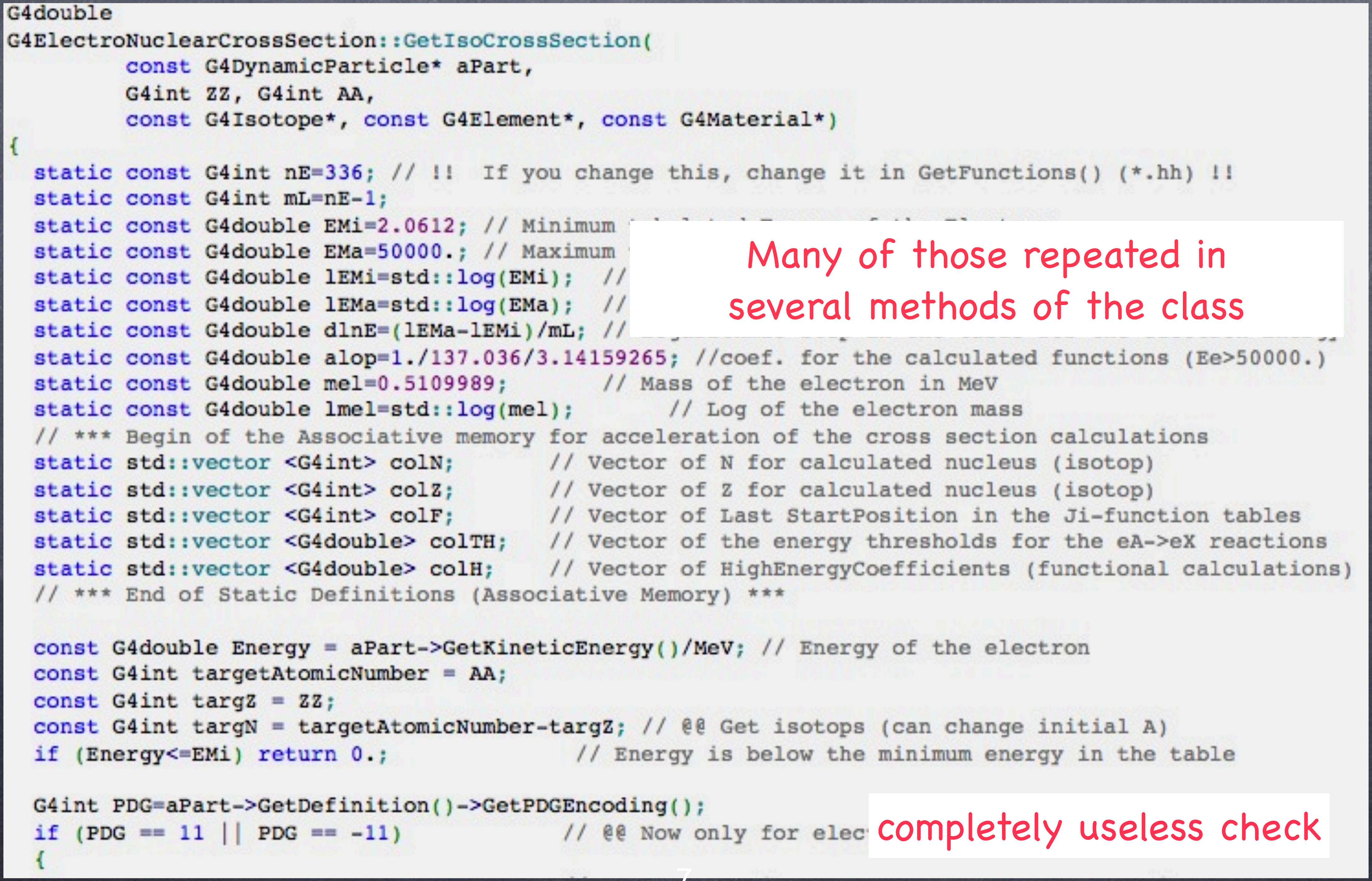
Bad design

- enormous use of ‘statics’
  - completely not needed
  - creating problems for MT
- obscure code to implement cache
  - inefficient, error-prone

G4ElectroNuclearCross Section



```
G4bool
G4ElectroNuclearCrossSection::IsIsoApplicable(
          const G4DynamicParticle* aParticle, G4int /*Z*/,
          G4int /*A*/, const G4Element*, const G4Material*)
{
  G4bool result = false;
  if (aParticle->GetDefinition() == G4Electron::ElectronDefinition())
    result = true;
  if (aParticle->GetDefinition() == G4Positron::PositronDefinition())
    result = true;
  return result;
}
```



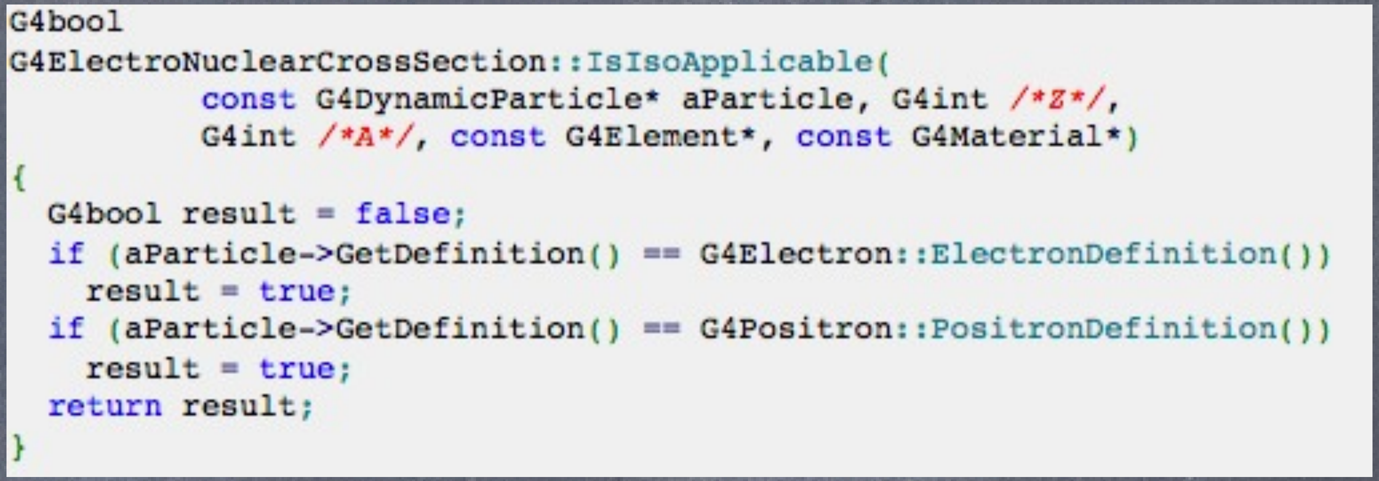
```
G4double
G4ElectroNuclearCrossSection::GetIsoCrossSection(
         const G4DynamicParticle* aPart,
         G4int ZZ, G4int AA,
         const G4Isotope*, const G4Element*, const G4Material*)
{
  static const G4int nE=336; // !!  If you change this, change it in GetFunctions() (*.hh) !!
  static const G4int mL=nE-1;
  static const G4double EMi=2.0612; // Minimum
  static const G4double EMa=50000.; // Maximum
  static const G4double lEMi=std::log(EMi);  //
  static const G4double lEMa=std::log(EMa);  //
  static const G4double dlnE=(lEMa-lEMi)/mL; //
  static const G4double alop=1./137.036/3.14159265; //coef. for the calculated functions (Ee>50000.)
  static const G4double mel=0.5109989;        // Mass of the electron in MeV
  static const G4double lmel=std::log(mel);         // Log of the electron mass
  // *** Begin of the Associative memory for acceleration of the cross section calculations
  static std::vector <G4int> colN;        // Vector of N for calculated nucleus (isotop)
  static std::vector <G4int> colZ;        // Vector of Z for calculated nucleus (isotop)
  static std::vector <G4int> colF;        // Vector of Last StartPosition in the Ji-function tables
  static std::vector <G4double> colTH;    // Vector of the energy thresholds for the eA->eX reactions
  static std::vector <G4double> colH;     // Vector of HighEnergyCoefficients (functional calculations)
  // *** End of Static Definitions (Associative Memory) ***

  const G4double Energy = aPart->GetKineticEnergy()/MeV; // Energy of the electron
  const G4int targetAtomicNumber = AA;
  const G4int targZ = ZZ;
  const G4int targN = targetAtomicNumber-targZ; // @@ Get isotops (can change initial A)
  if (Energy<=EMi) return 0.;                // Energy is below the minimum energy in the table

  G4int PDG=aPart->GetDefinition()->GetPDGEncoding();
  if (PDG == 11 || PDG == -11)               // @@ Now only for elec
  {
```

Many of those repeated in several methods of the class

completely useless check

(Mis)use of Is(...)Applicable method

- called before each call to ‘get cross section’ method (!!)



```
G4bool
G4ElectroNuclearCrossSection::IsIsoApplicable(
          const G4DynamicParticle* aParticle, G4int /*Z*/,
          G4int /*A*/, const G4Element*, const G4Material*)
{
  G4bool result = false;
  if (aParticle->GetDefinition() == G4Electron::ElectronDefinition())
    result = true;
  if (aParticle->GetDefinition() == G4Positron::PositronDefinition())
    result = true;
  return result;
}
```

- these checks are really not needed!
- cross section is called for the particle you have assigned it to
  - if any check needed it should be at initialization

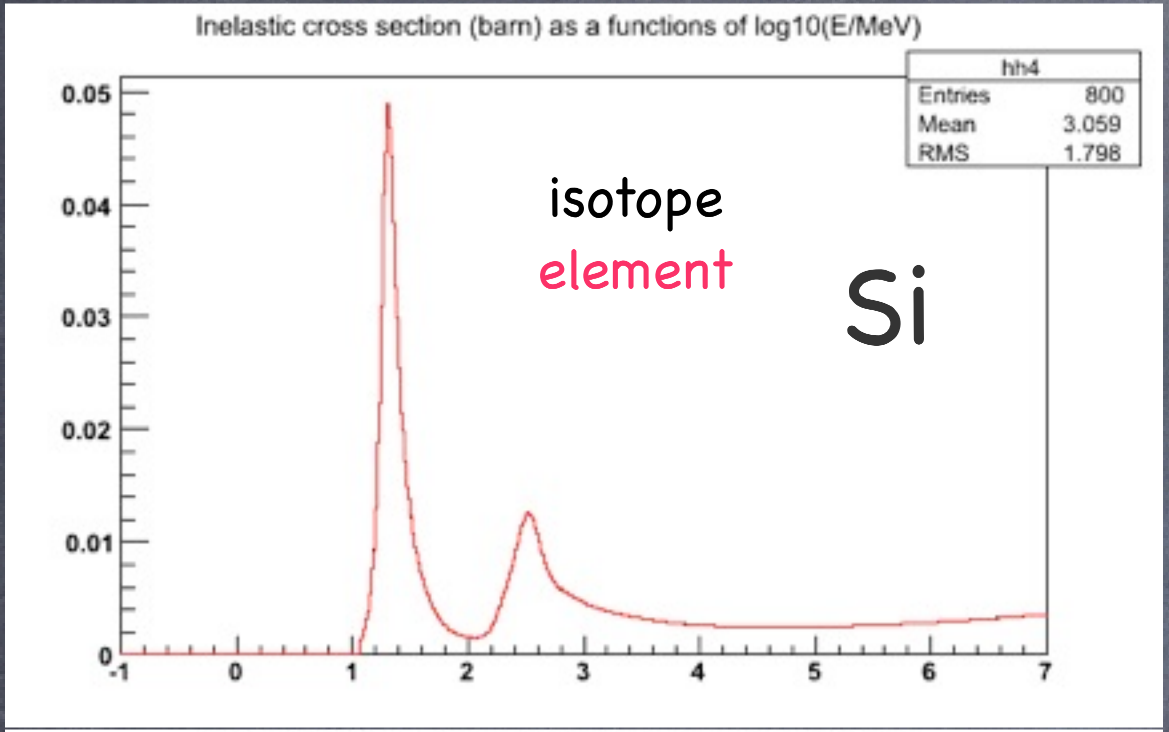
Is(...)Applicable needed?

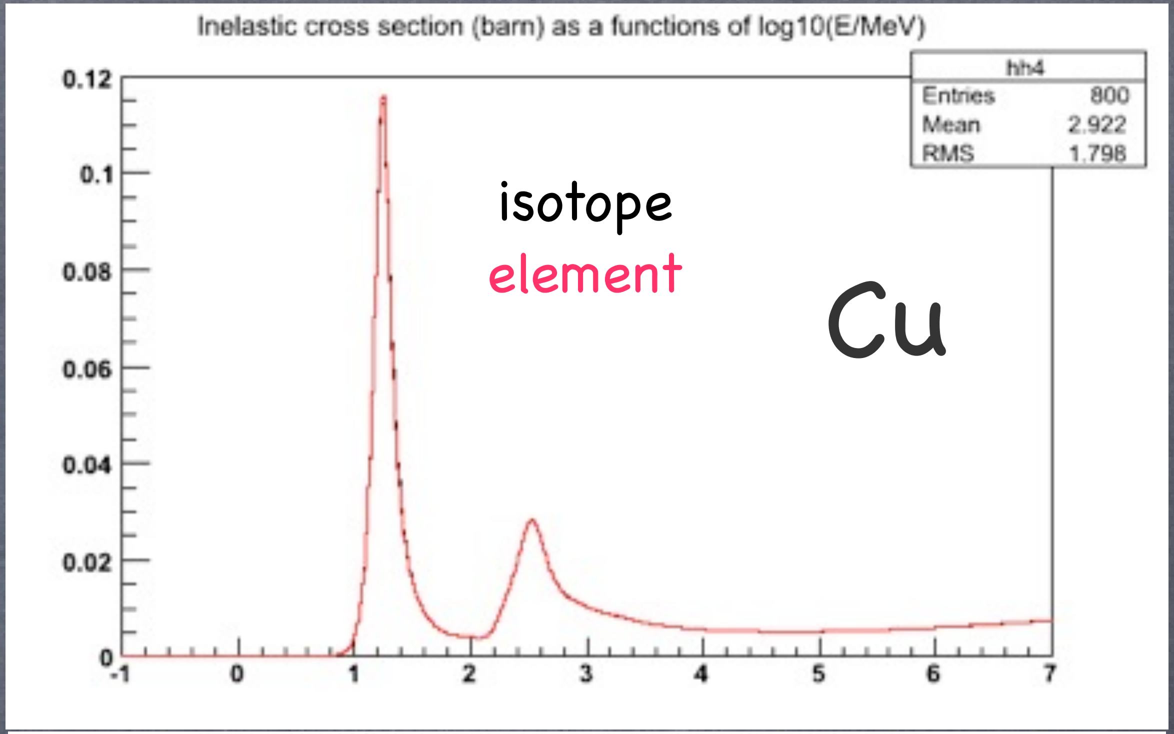
- only current use to have IsIso/ElementApplicable is to check for energy range if two cross sections are ‘glued’ together
  - if cross section used in the whole energy range (most of the cases) the method returns only true
- proposal: get rid of Is(...)Applicable method
  - easy to redesign in `G4CrossSectionDataStore::GetCrossSection`
- if a cross section does not cover the whole energy range and needs to be glued with another one, it can be handled with a ‘wrapper’ cross-section (calling two different cross-sections behind)

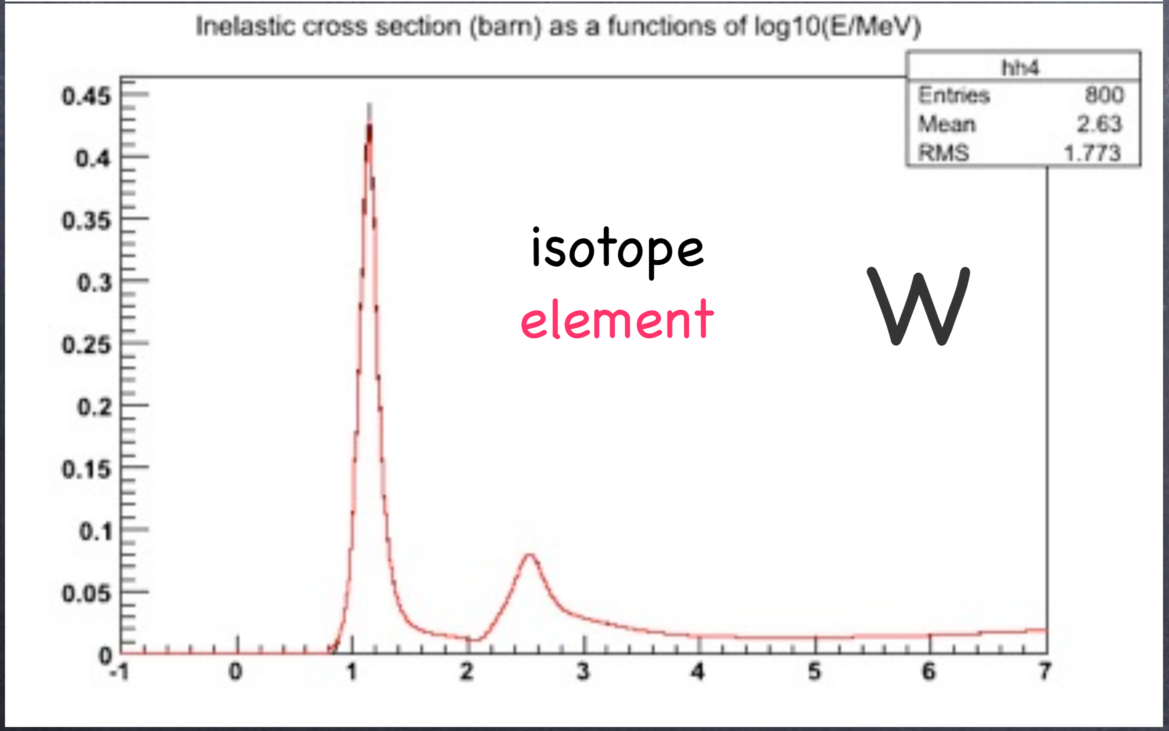
Per isotope vs per element

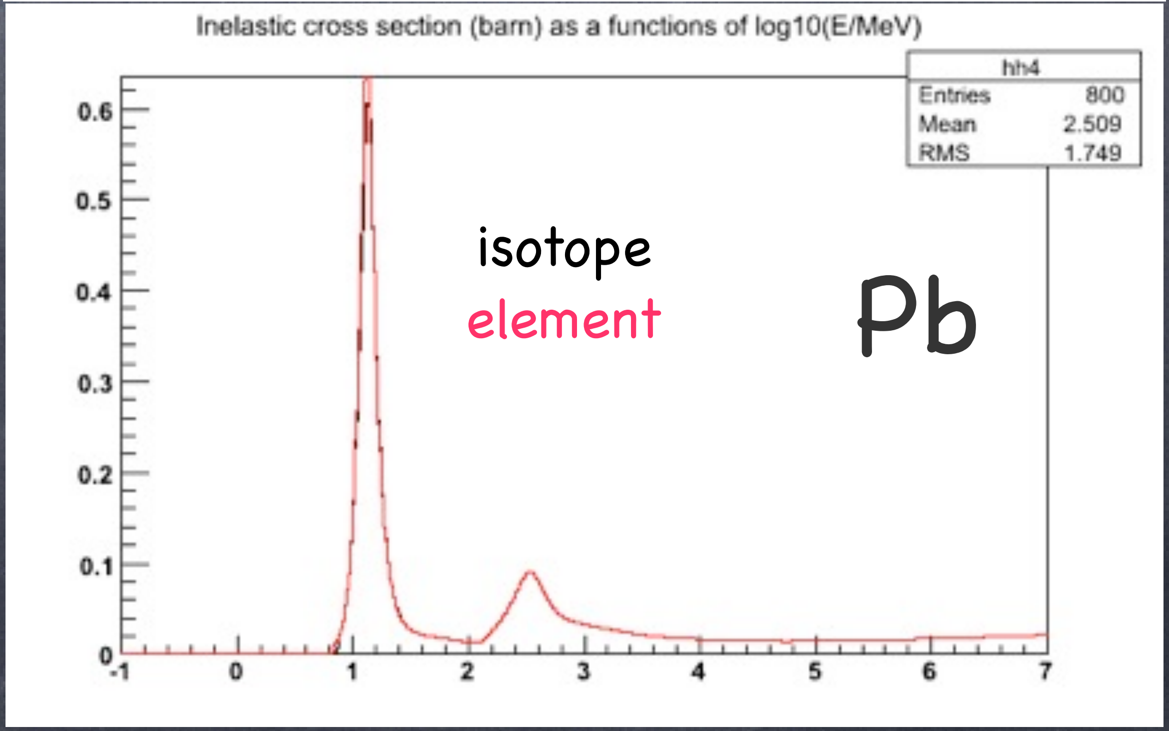
- per isotope cross section means that we need check every time Z and N, and calculate it
  - in case of CHIPS cross section there is a huge CPU penalty for it
- per element cross section means that it depends only on Z (we take average N)
  - huge (5 times!!!) gain in speed
  - we loose in accuracy, but is it relevant?
    - in most (all HEP?) cases no!

Gamma nuclear XS

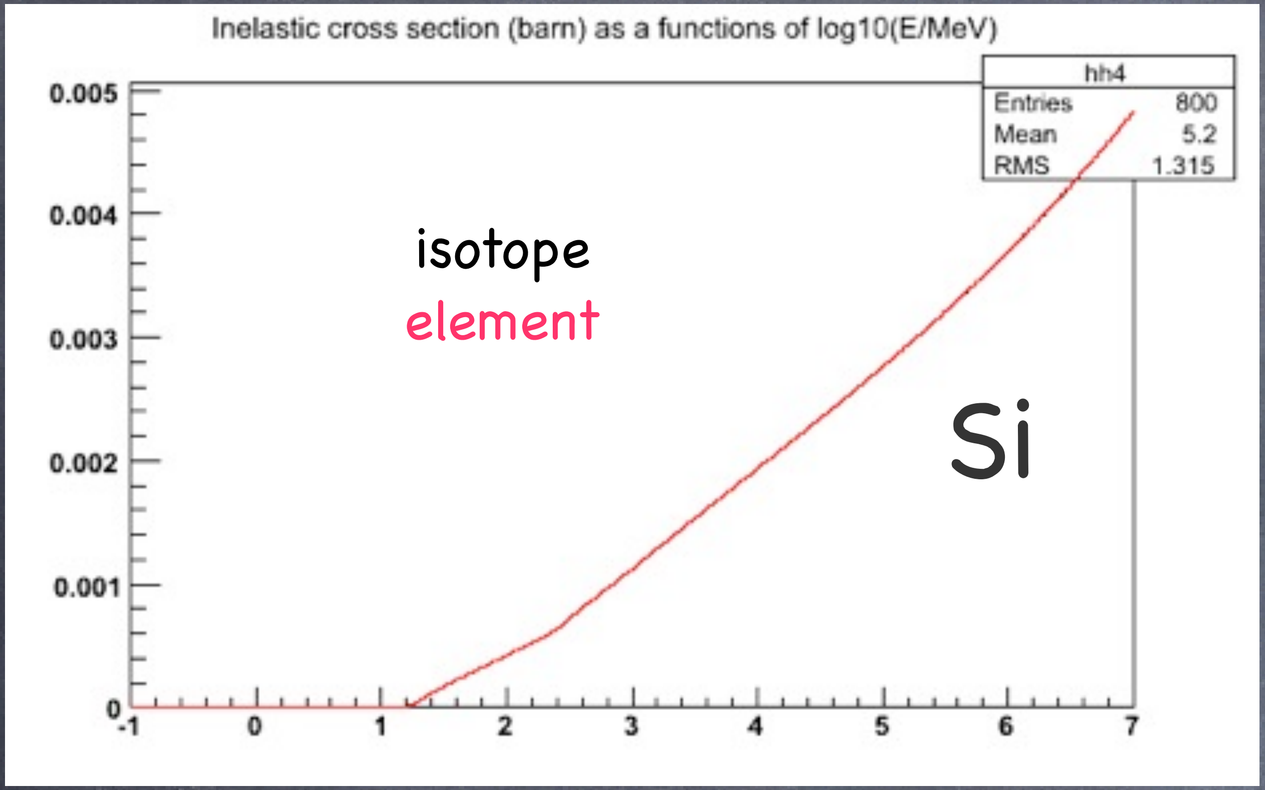


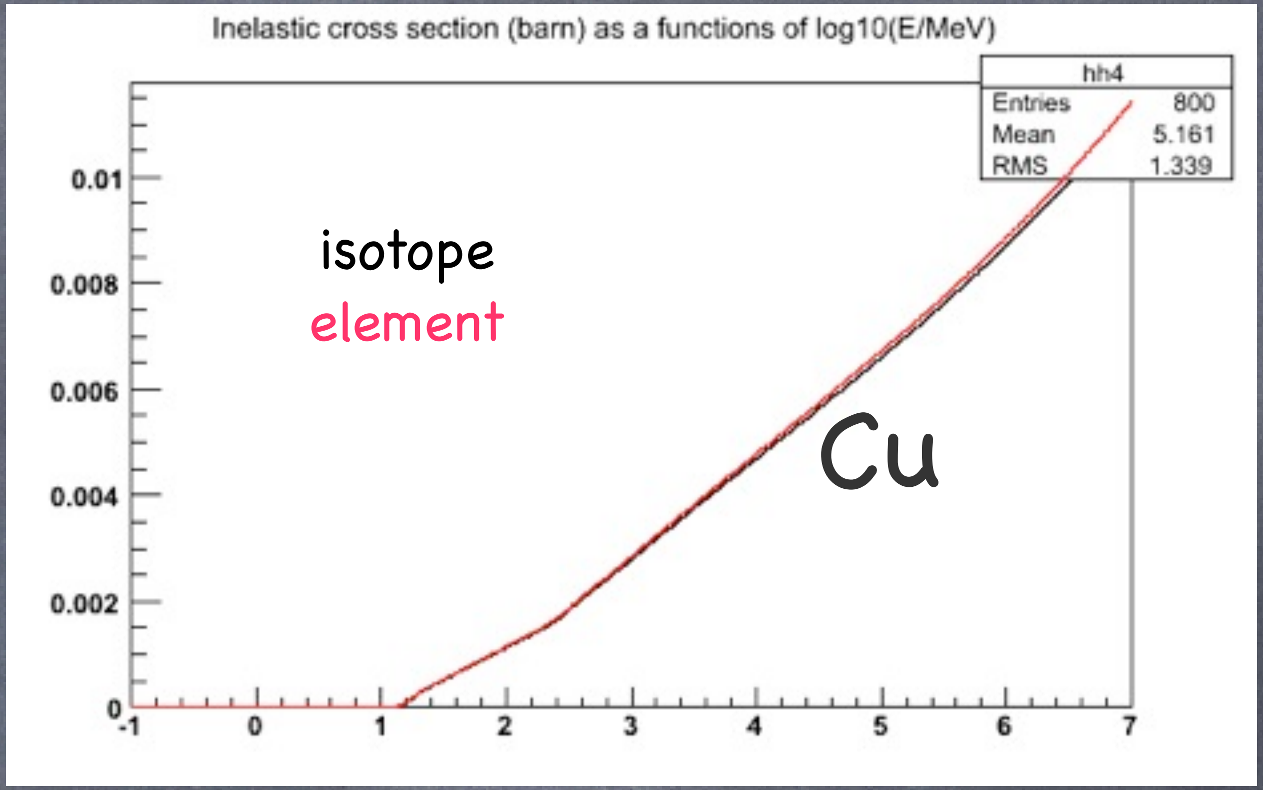


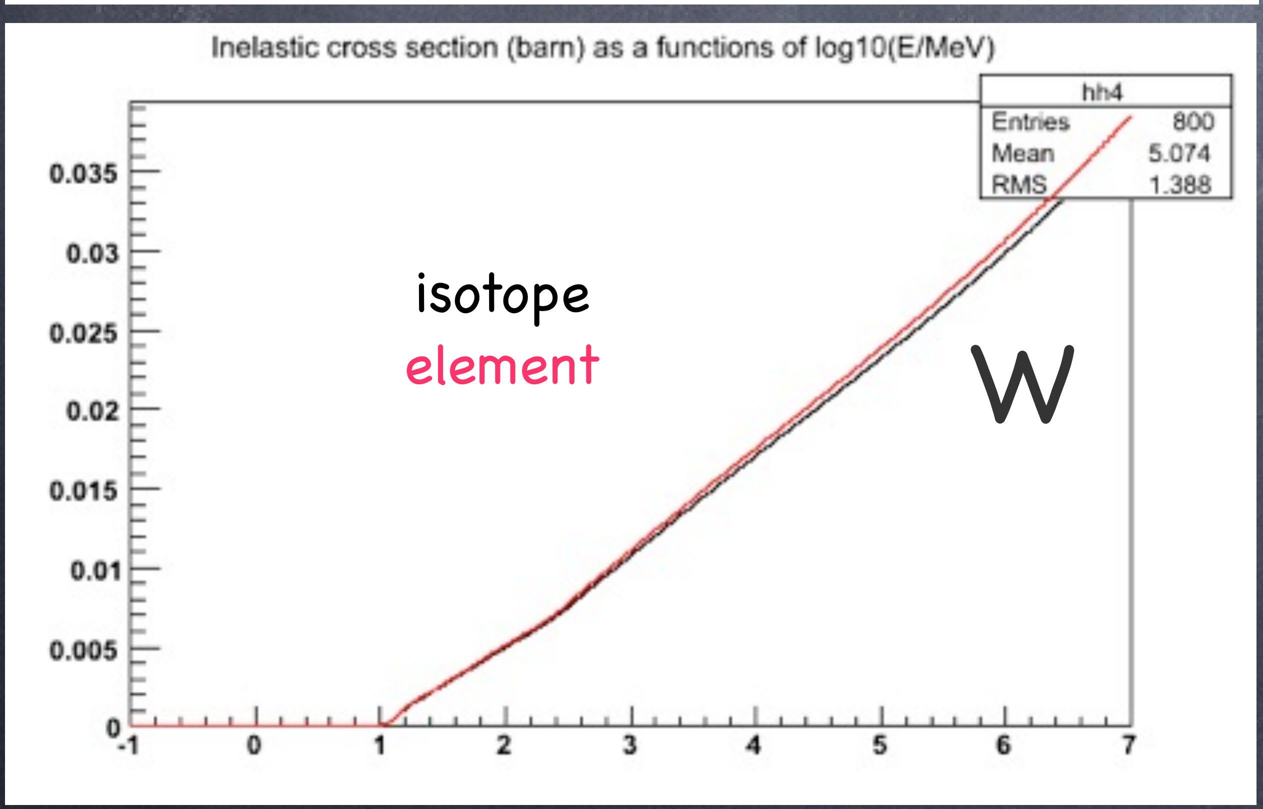


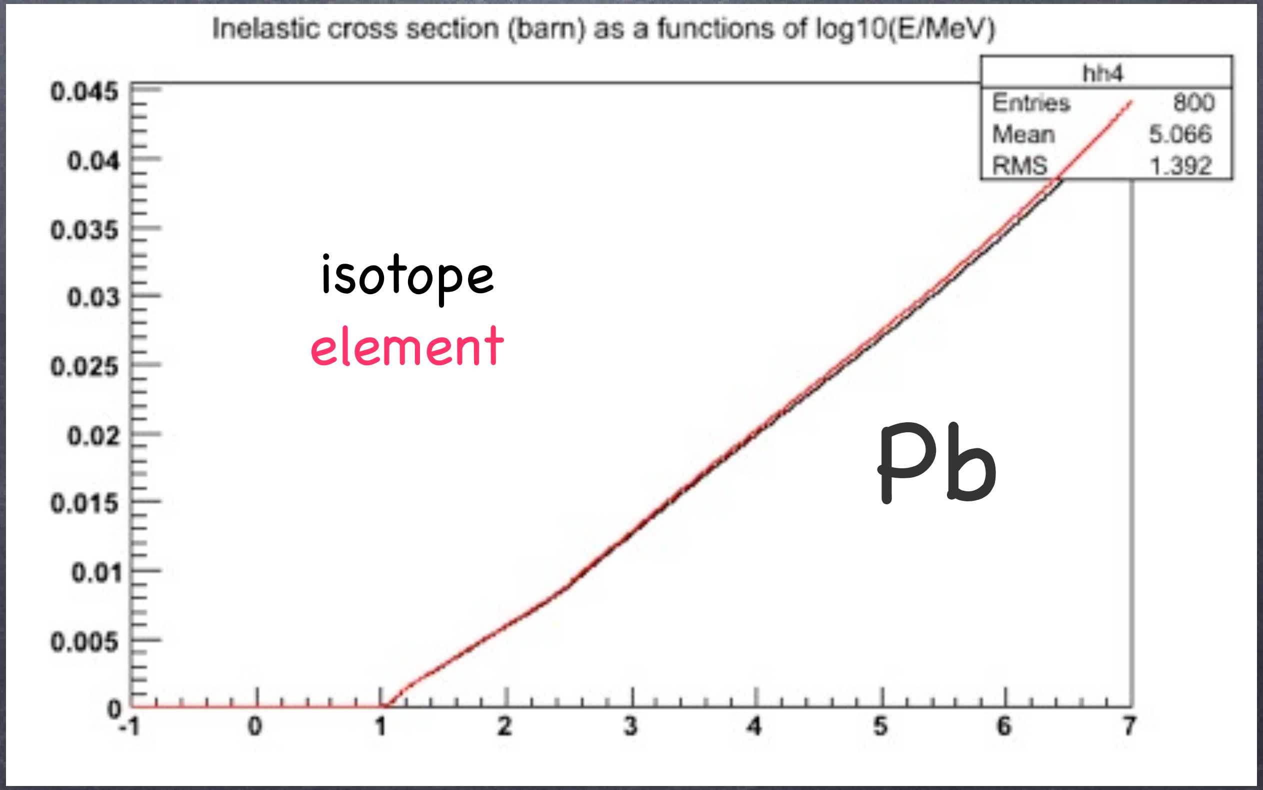


Electronuclear XS

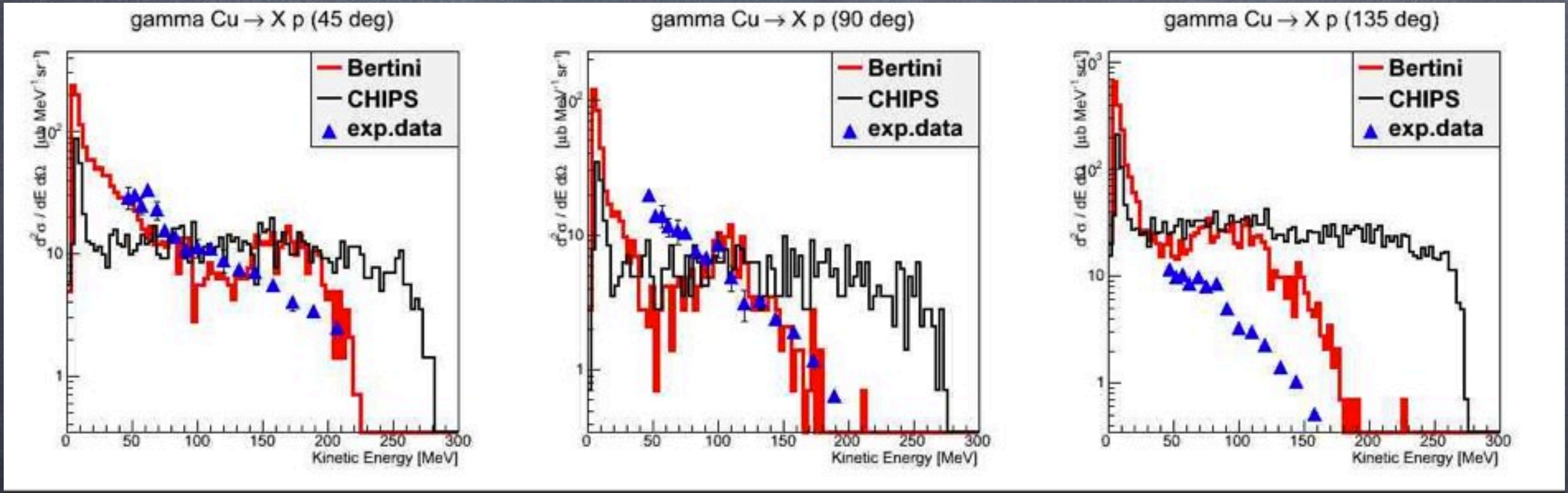








For reference: gamma-nuclear validation



Cleanup of CHIPS XS

- major clean up of gamma- and electro-nuclear cross sections done
  - no more statics
  - removed all redundant checks
  - per element calculation
    - in simplified calorimeter from 0.6% CPU to 0.1%
  - moved to 'IsElementApplicable'
- all 'automatic' migration to MT could be removed

Plan

- review the implementation of all cross sections
- get rid of not needed ‘statics’ (especially in former CHIPS code)
- remove checks for particle from Is(...)Applicable methods
- move from per isotope to per element cross sections where it is justified (everywhere for HEP?)
- revise the hadronic framework to eventually get rid of Is(...)Applicable methods?

Conclusion

- thanks to V. Uzhinski, A. Galloyan and V. Grichine new (better) cross sections added to physics lists
- significant improvement in CPU performance of electro- and gamma-nuclear XS
  - 5-fold for electro-nuclear in simplified calorimeter
  - clean up and reorganization of the code done
  - ‘per isotope’ replaced by ‘per element’
- plan to do the same with other cross-section
- proposal to re-discuss the hadronic cross-section framework to improve the performance

Factory approach to cross-sections

- extended functionality of G4CrossSectionDataSetRegistry
  - responsible for instantiating cross sections
  - user should never 'new' cross section object
  - registry (singleton) provides the method GetCrossSectionDataSet(const G4String& name)
- unique cross-sections objects (for a give cross section) shared across the application