3A: Improving interfaces

- G.Folger "Integer Z and A"
- K.Tatsumi "Cross section interface redesign"
- M.Kelsey "Change of Bertini and other interfaces"
- V.Ivanchenko "Update of pre-compound/de-excitation/photon evaporation"
- D.Wright "Problems of hadronic interfaces"

- Working session in which details of Geant4 hadronic interfaces have been discussed
  - First 3 subjects were covered during the session
  - A part of the last subject have been discussed

Integer Z and A

- Migration was planned 2 years ago
- Few places not yet migrated
- Responsible persons are identified
- Agreed fix this before the release

Cross section interface redesign

- Main goals of migration are achieved:
  - Implementation of material dependent cross sections for low-energy neutron sub-libraries now is possible
  - Code is optimized and number of run time calls to cross section is reduced

One on one comparison

Current interface of

```
virtual
G4double GetCrossSection(const G4DynamicParticle*,
                         const G4Element*,
                         G4double aTemperature = 0.) = 0;
```

be replaced by

```
virtual
 G4double GetElementCrossSection(const G4DynamicParticle*, G4int Z,
                                 const G4Material* mat = 0);
```

However we still provides

```
inline G4double GetCrossSection(const G4DynamicParticle*, const G4Element*,
                                const G4Material* mat = 0);
```

Next steps to improve cross section interface

- G4Element construction should be extended:
  - G4Isotopes should be created even if user does not specify set of isotopes
  - Material group members agree with the proposal
  - This can be done after 9.5
- When all G4Elements will have G4Isotope vector the interfaces to cross section may be changed to be more elegant

Summary

- Interface changes within Bertini
  - Return "do nothing" state if cascade fails
  - Use of **G4PreCompoundModel** vs. internal de-excitation
  - Use of **Propagate()** for string rescattering
- Issues during development
  - Different and confusing interfaces for different stages
  - Significant memory fragmentation, churn
- Hadronic infrastructure changes
  - Improve memory use in **G4VParticipants**, **G4V3DNucleus**
  - Improve buffer (re)use in **G4HadFinalState**
  - Utility to decay **G4KineticTrackVector** contents
- Recommendations for further work

Memory Fragmentation

Significant inefficiencies in memory fragmentation, churn

G4VParticipants	64 kB per cascade	*Eliminated*
G4Fancy3DNucleus	47 kB per cascade	*Eliminated*
G4ReactionProductVector	10 kB per cascade	*Reduced 90%*
G4VFermiFragments, G4FermiFragmentsPool	3.5 kB per cascade	
G4DiffractiveSplitableHadron	2.5 kB per cascade	
G4HadFinalState, G4HadSecondary	2 kB per cascade	*Eliminated*
G4KineticTrack(Vector)	0.5 kB per cascade	

All due to vectors of pointers, and vector objects being created and deleted on every interaction

Recommendations

- ⋆ Systematic use of **IgProf** or other memory profiling tool to identify inefficiencies
- Class data members for vectors, lists, etc. to allow reuse
- Vectors of objects, not pointers, to reuse allocations
- Improve constness of interfaces
- Clarify ownership: don't pass/return pointers if receiving code shouldn't delete
- ⋆ **G4Allocator** model for **G4ReactionProduct**

Friday discussion

- G4Exception migration – should be completed
- Introduction of automatic documentation
  - Description() method inside each process, model, cross section
  - DumpHtml(), PrintHtml() methods added to G4HadronicProcessStore class
- Request should be formulated for run category to add public interface allowing printout of Physics List
  - Also UI command enabling printout
- Requirement to Physics List task force to add name to G4VUserPhysicsList and method to activate printout