Shower analysis

A. Dotti
CALICE-Geant4 Workshop

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Introduction

- Request from March CALICE Collaboration Meeting:
  - Facilitate analysis of shower shape: **utilities to reconstruct primary-secondary relation** of particles.
    - e.g. "Is this particular electron coming from a proton?", "what is the total energy deposit from all electrons produced by this primary?"
- Challenges:
  - Keeping in memory the entire tree of primary-secondary relation would easily make the memory requirement explode
    - If needed this information must be reconstructed/kept by user
  - Very difficult (impossible?) to accomodate all use-cases
- We propose a **general purpose utility** to CALICE to facilitate analysis

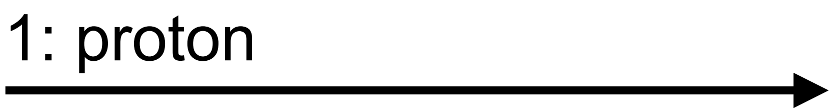
G4ShowerMap

- It’s not a Geant4 utility, I developed it for CALICE, but **we can add it to the toolkit if proved to be useful**
- Basic idea:
  - Create a **map**: ID ↔ (userInfo , parent , secondaries , particle type)
  - Allow for **filtering**: consider only particles matching a given criteria (e.g. type)
  - User is **responsible to add** an interesting particle to the map in a simulation application
    - Trying to minimize memory usage (e.g. do not consider non interesting particles)

Identifying particles relationship

- Each G4Track has two integers variables:
  - **TrackID** : uniquely identify track in current event
  - **ParentID** : the id of the particle that produced this one
- Notes:
  - `min(TrackID)==1`
  - `if (parentID==0) then track is a primary` (e.g. a pythia input)
- Secondaries are simulated after the primary is killed
  - This is not strictly true if you implemented a G4StackingAction, if so we may need to iterate…
  - This is why a gibe G4Track cannot point to secondaries

Example



Id	parent	type	First Secondary	Next Sibling
1	-	p	–	-

Example

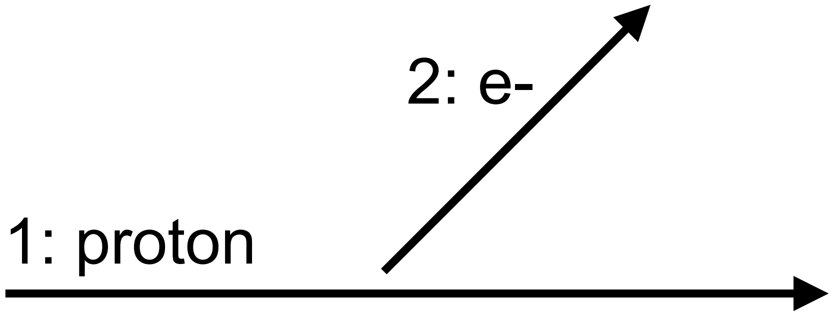
Key

1: proton

Id	parent	type	First Secondary	Next Sibling
1	-	p	–	-

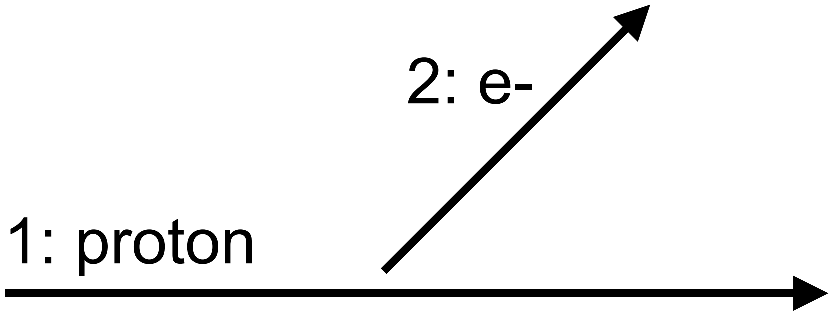
Insert in map

Example



Id	parent	type	First Secondary	Next Sibling
1	-	p	**2**	-
2	1	e-	-	-

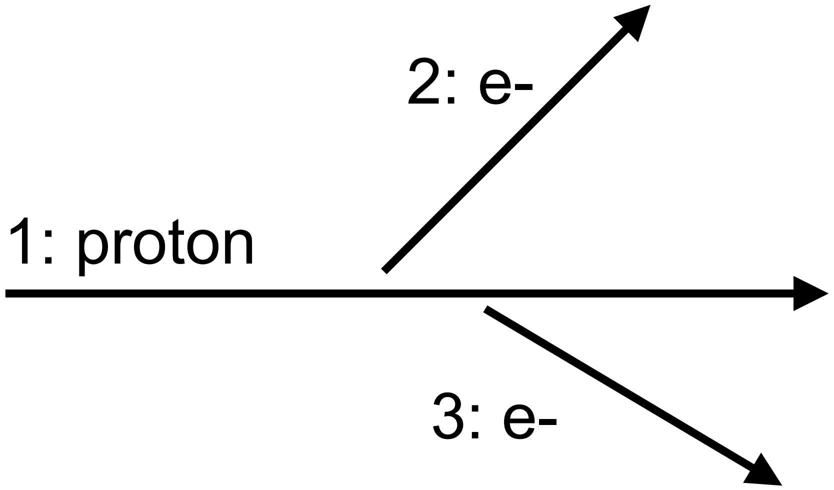
Example



Id	parent	type	First Secondary	Next Sibling
1	-	p	**2**	-
2	1	e-	-	-

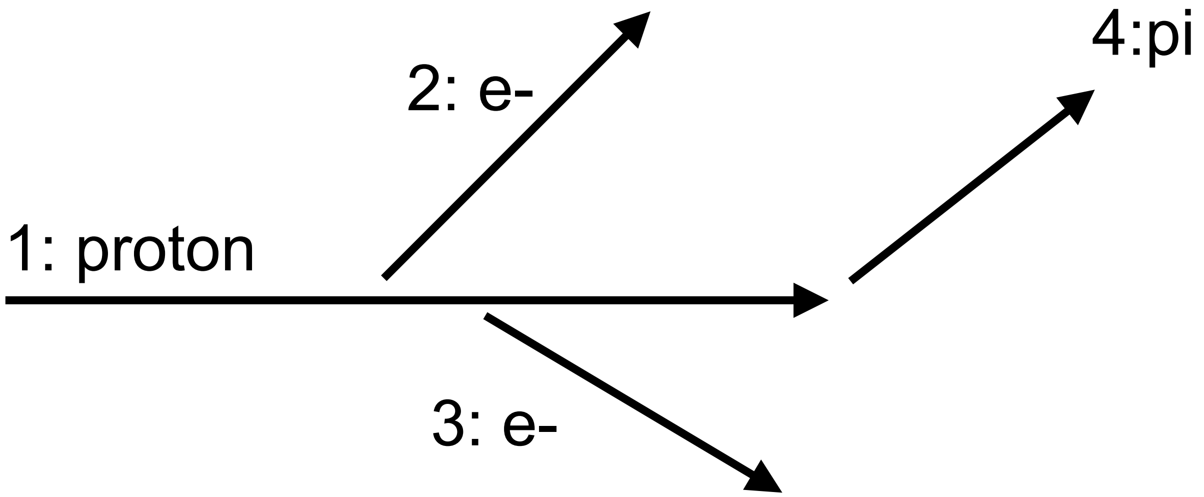
Update navigation pointers

Example



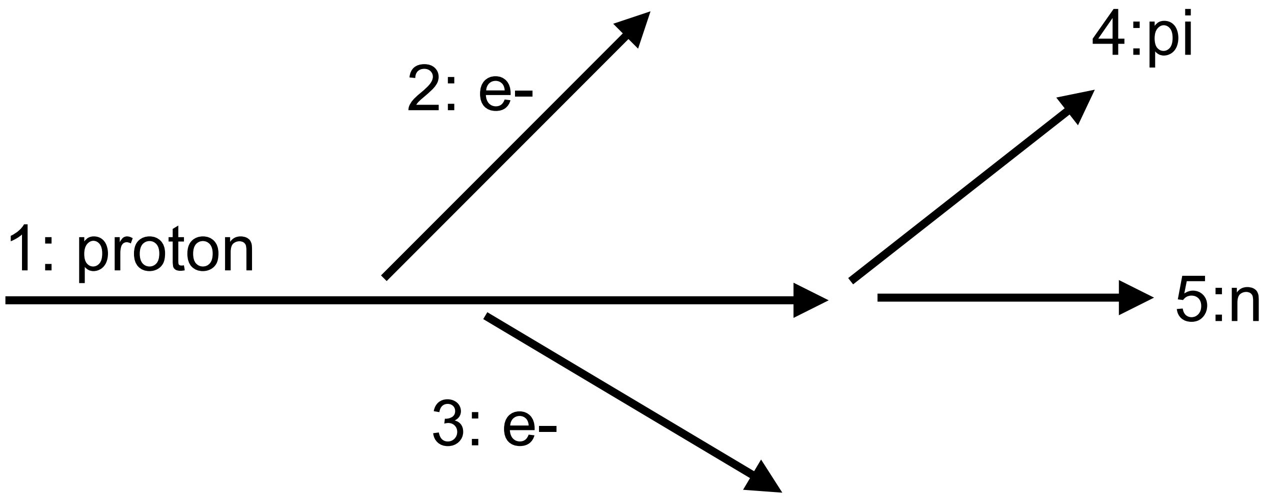
Id	parent	type	First Secondary	Next Sibling
1	-	p	2	-
2	1	e-	-	**3**
3	1	e-	-	-

Example



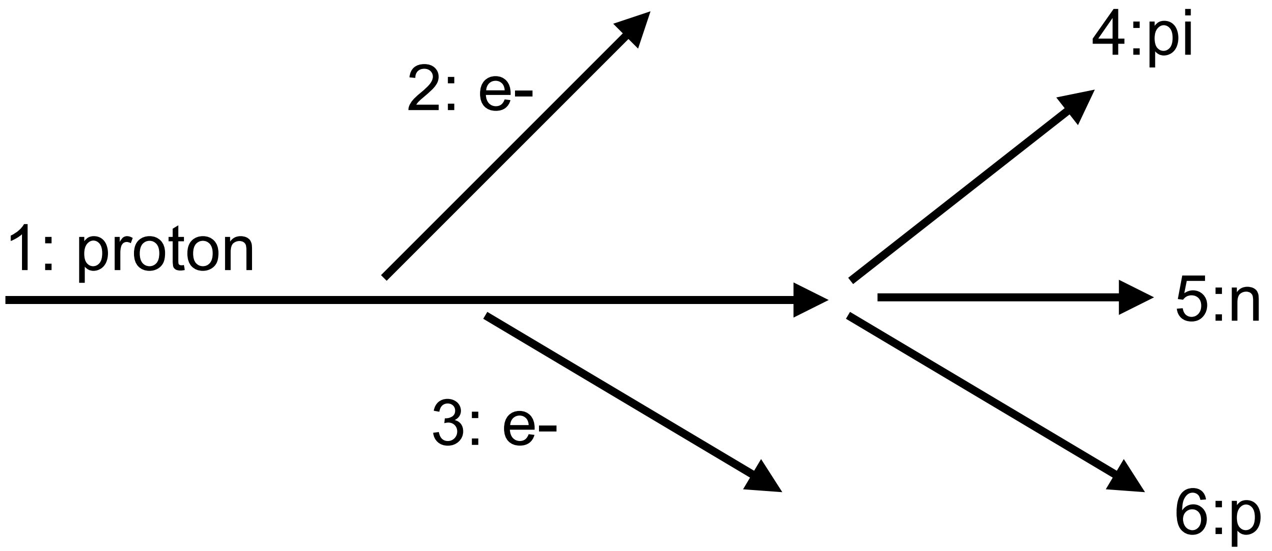
Id	parent	type	First Secondary	Next Sibling
1	-	p	2	-
2	1	e-	-	3
3	1	e-	-	**4**
4	1	pi	-	-

Example



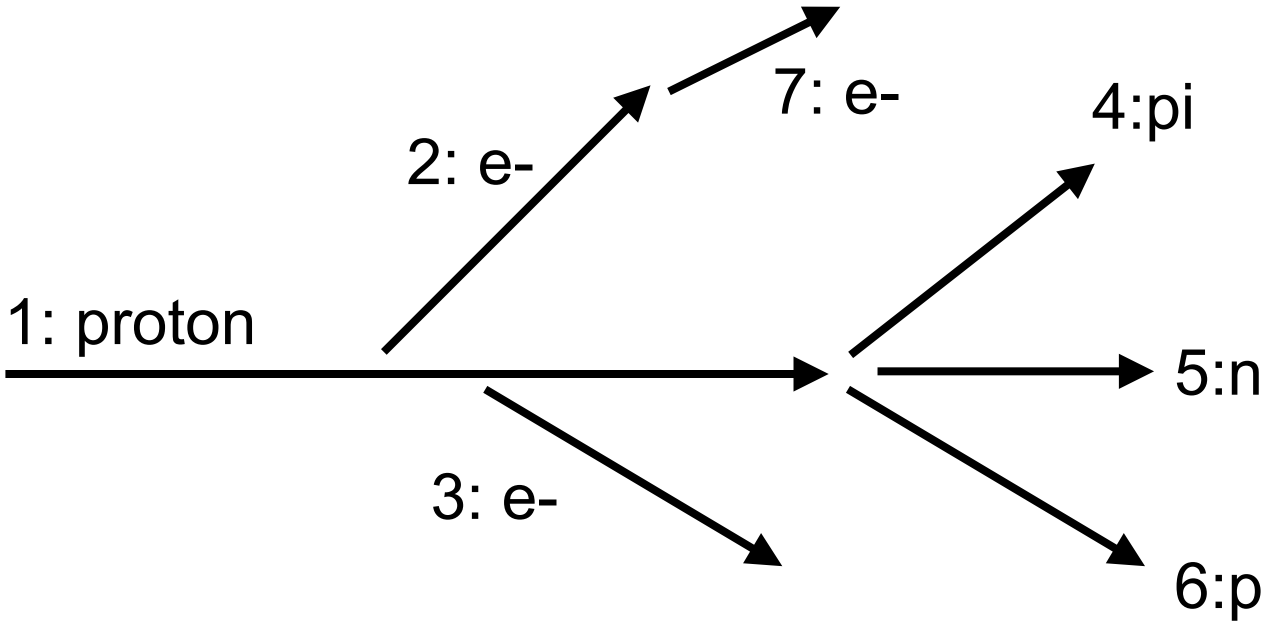
Id	parent	type	First Secondary	Next Sibling
1	-	p	2	-
2	1	e-	-	3
3	1	e-	-	4
4	1	pi	-	**5**
5	1	n	-	-

Example



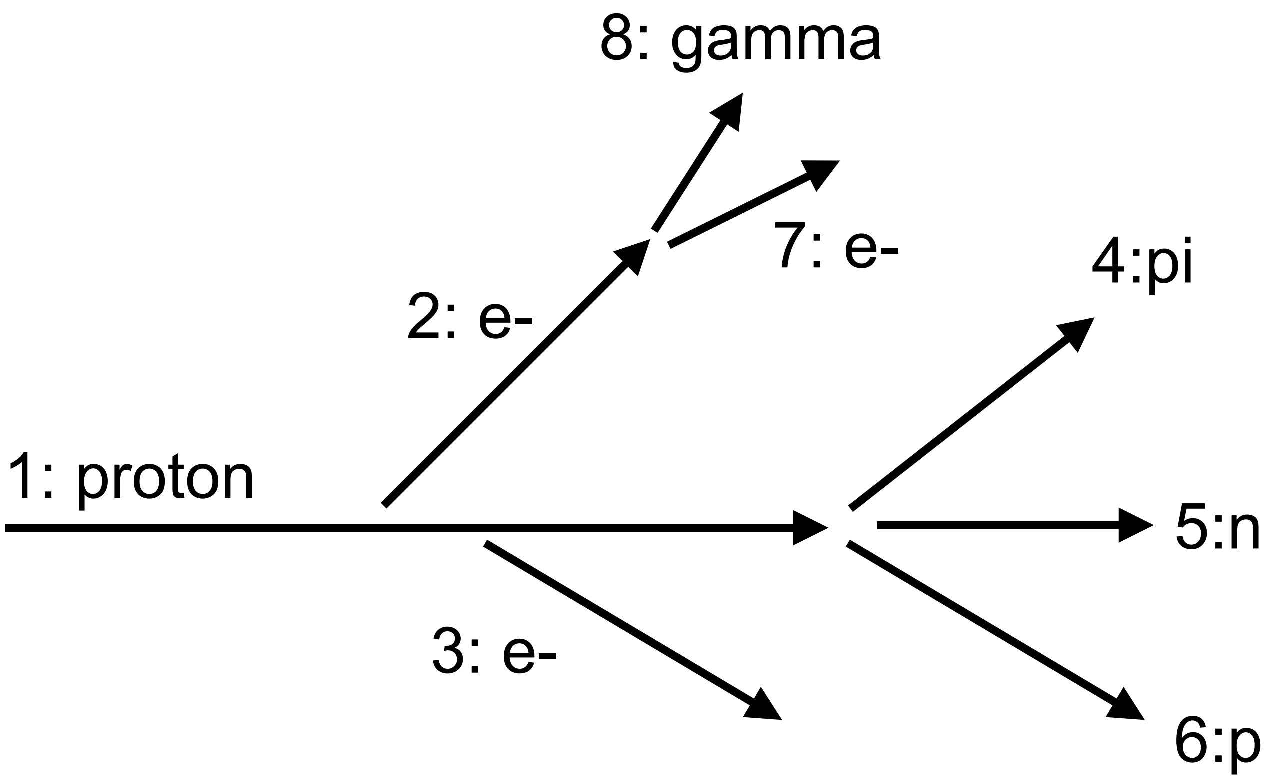
Id	parent	type	First Secondary	Next Sibling
1	-	p	2	-
2	1	e-	-	3
3	1	e-	-	4
4	1	pi	-	5
5	1	n	-	**6**
6	1	p	-	-

Example



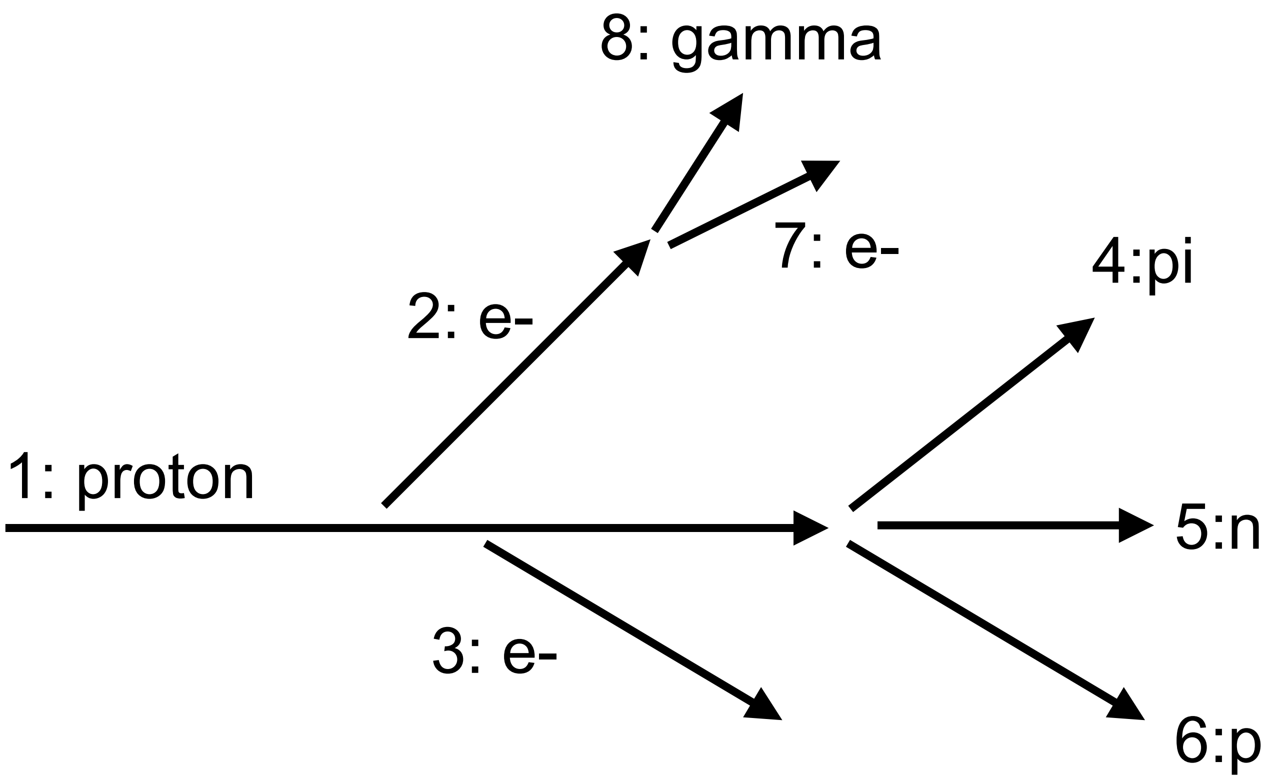
Id	parent	type	First Secondary	Next Sibling
1	-	p	2	-
2	1	e-	**7**	3
3	1	e-	-	4
4	1	pi	-	5
5	1	n	-	6
6	1	p	-	-
7	2	g	-	-

Example



Id	parent	type	First Secondary	Next Sibling
1	-	p	2	-
2	1	e-	7	3
3	1	e-	-	4
4	1	pi	-	5
5	1	n	-	6
6	1	p	-	-
7	2	g	-	**8**
8	2	e-	-	-

Example



Id	parent	type	First Secondary	Next Sibling
1	-	p	2	-
2	1	e-	7	3
3	1	e-	-	4
4	1	pi	-	5
5	1	n	-	6
6	1	p	-	-
7	2	g	-	8
8	2	e-	-	-

Given a key
it is possible to navigate up-down via
(parent,FirstSecondary,NextSibling) triplets

Implementation

- A general "parent-children" container has been created
- Concrete implementations to allow for storing of a simple G4double as user information (e.g. energy deposit, track length)
- Added utility to filter on particle type
  - Can be extended to allow for filtering on more complex conditions
- The "complexity" is in the mechanism to navigate the tree up and down (given a particle get the parent or the secondaries)

An efficiency consideration

- Container class is a singleton
  - To minimize memory
- If you need to store more than one G4double per particle, extend code:

```
class Analysis : public TShowerMap<G4double> {…}
```

- Modify to:

```
struct myData {
    G4double edep;
    G4double trackL;
    …
    myData& operator+=(const myData& rhs) {…}
    myData() { … }
};
class MyAnalysis : public TShowerMap<MyData> { … }
```

An efficiency consideration

- Container class is a singleton
  - To minimize memory
- If you need to store more than one G4double per particle, extend code:

```
class Analysis : public TShowerMap<G4double> {…}
```

- Modify to:

```
struct myData {
    G4double edep;
    G4double trackL;
    …
    myData& operator+=(const myData& rhs) {…}
    myData() { … }
};
class MyAnalysis : public TShowerMap<MyData> { … }
```

Required!

Adding information

Example (Adding energy deposited at each step):

```
MySteppingAciton::UserSteppingAction(const G4Step* step) {
    G4double eDep = step->GetTotalEnergyDeposit();
    G4Track* track = step->GetTrack();
    G4int trackId = track->GetTrackId();
```

Adding information

Example (Adding energy deposited at each step):

```
MySteppingAciton::UserSteppingAction(const G4Step* step) {
    G4double eDep = step->GetTotalEnergyDeposit();
    G4Track* track = step->GetTrack();
    G4int trackId = track->GetTrackId();
    G4ShowerMap::Analysis* instance = G4ShowerMap::Analysis::Instance();
```

Adding information

Example (Adding energy deposited at each step):

```
MySteppingAciton::UserSteppingAction(const G4Step* step) {
    G4double eDep = step->GetTotalEnergyDeposit();
    G4Track* track = step->GetTrack();
    G4int trackId = track->GetTrackId();
    G4ShowerMap::Analysis* instance = G4ShowerMap::Analysis::Instance();
    if ( instance->Exists(trackId) ) {
        G4double oldValue = 0;
        instance->GetValue( trackId, oldvalue );
        instance->Update( trackId , oldvalue+eDep );
    }
```

Adding information

Example (Adding energy deposited at each step):

```
MySteppingAciton::UserSteppingAction(const G4Step* step) {
    G4double eDep = step->GetTotalEnergyDeposit();
    G4Track* track = step->GetTrack();
    G4int trackId = track->GetTrackId();
    G4ShowerMap::Analysis* instance = G4ShowerMap::Analysis::Instance();
    if ( instance->Exists(trackId) ) {
        G4double oldValue = 0;
        instance->GetValue( trackId, oldvalue );
        instance->Update( trackId , oldvalue+eDep );
    } else {
        G4int parentId = track->GetParentId();
        G4ParticleDefinition* def = track->GetDefinition();
        instance->AddSecondary( trackId, parentId , def , eDep);
    }
}
```

Retrieving Information

- Several methods are provided to extract information
  - For example at the end-of-event

- All "getters" have an optional parameter:
  - A **condition** to consider only particles matching the selection criteria

- Example, to get sum of value for all secondaries:

```
G4double result = 0;
G4bool success = instance->GetSumSecondaries( 345, result );
```

- Consider only secondary electrons:

```
G4ShowerMap::conditions::ptype eleFilter(G4Electron::Definition());
G4double result = 0;
G4bool success = instance->GetSumSecondaries(345,result,eleFilter);
```

Main analysis “getters”

- **`Matches( id, condition)`**: Does “id” match condition?
- **`ParentMatches(id,cond)`**: Go back into tree starting from “id” and check if a parent matches condition (returns first one to match).
- **`GetValue(id,cond)`**: Value associated with “id” if matches
- **`GetSumParents(id,cond)`**: Go back in tree and sum values of matching parents
- **`GetSumSecondaries(id,cond)`**: sum of direct matching secondaries
- **`GetSecondariesIds(id,cond)`**: the track-ids of matching secondaries

Test

- Check test file included for a detailed example of all interfaces (it does not require G4 installation):

```
make
./test
```

Next Steps

- Please try out the code (attached tar ball to the agenda)
- It should be able to modify it for your specific needs