SuperBigBite
Tracking Simulation Framework

Ole Hansen

Jefferson Lab

Hall A Data Analysis Workshop
December 8, 2010

Simulation Basics

- Goals
  - Feed simulation data into Podd
  - Make it behave exactly like a real analysis
  - Compare analyzed results with Monte Carlo "truth" data
- What you need
  - Event generator (`gRandom`, Pythia, ...)
  - Particle transport and interaction simulator (raytrace, `Geant4`, ...)
  - Digitization (conversion to detector output)
  - Interface software for Podd (this talk)
  - Usual set of databases and scripts

Simulation Basics

- Goals
  - Feed simulation data into Podd
  - Make it behave exactly like a real analysis
  - Compare analyzed results with Monte Carlo "truth" data
- What you need
  - Event generator (`gRandom`, Pythia, ...)
  - Particle transport and interaction simulator (raytrace, `Geant4`, ...)
  - Digitization (conversion to detector output)
  - Interface software for Podd (this talk)
  - Usual set of databases and scripts

Simulation Basics

- Goals
  - Feed simulation data into Podd
  - Make it behave exactly like a real analysis
  - Compare analyzed results with Monte Carlo "truth" data
- What you need
  - Event generator (`gRandom`, Pythia, ...)
  - Particle transport and interaction simulator (raytrace, `Geant4`, ...)
  - Digitization (conversion to detector output)
  - Interface software for Podd (this talk)
  - Usual set of databases and scripts

Simulation Basics

- Goals
  - Feed simulation data into Podd
  - Make it behave exactly like a real analysis
  - Compare analyzed results with Monte Carlo "truth" data
- What you need
  - Event generator (`gRandom`, Pythia, ...)
  - Particle transport and interaction simulator (raytrace, `Geant4`, ...)
  - Digitization (conversion to detector output)
  - Interface software for Podd (this talk)
  - Usual set of databases and scripts

Simulation Basics

- Goals
  - Feed simulation data into Podd
  - Make it behave exactly like a real analysis
  - Compare analyzed results with Monte Carlo "truth" data
- What you need
  - Event generator (`gRandom`, Pythia, ...)
  - Particle transport and interaction simulator (raytrace, `Geant4`, ...)
  - Digitization (conversion to detector output)
  - Interface software for Podd (this talk)
  - Usual set of databases and scripts

Simulation Basics

- Goals
  - Feed simulation data into Podd
  - Make it behave exactly like a real analysis
  - Compare analyzed results with Monte Carlo "truth" data
- What you need
  - Event generator (`gRandom`, Pythia, ...)
  - Particle transport and interaction simulator (raytrace, `Geant4`, ...)
  - Digitization (conversion to detector output)
  - Interface software for Podd (this talk)
  - Usual set of databases and scripts

Podd Simulation Framework

- Standard Analyzer call sequence:

```
Int_t THaAnalyzer::ReadOneEvent()
{
  // Read one event from current run (fRun) and raw-decode it using the
  // current decoder (fEvData)

  fRun->ReadEvent();
  fEvData->LoadEvent( fRun->GetEvBuffer() );
  ...
}
```

- To implement custom input:
  - Use custom simulation run class, inheriting from `THaRunBase`
  - Override `LoadEvent` with custom simulation decoder, inheriting from `THaEvData`

Podd Simulation Framework

- Standard Analyzer call sequence:

```
Int_t THaAnalyzer::ReadOneEvent()
{
  // Read one event from current run (fRun) and raw-decode it using the
  // current decoder (fEvData)

  fRun->ReadEvent();
  fEvData->LoadEvent( fRun->GetEvBuffer() );
  ...
}
```

- To implement custom input:
  - Use custom simulation run class, inheriting from `THaRunBase`
  - Override `LoadEvent` with custom simulation decoder, inheriting from `THaEvData`

Simulation Run Class

- Must inherit from `THaRunBase`
- May read arbitray input, *e.g.* text file, ROOT file, database . . .
- Input is set up in `Open()`. With ROOT file input, this is the place for `SetBranchAddress` and `SetBranchStatus`
- Digitized event data is fetched in `ReadEvent()`. With ROOT file input, call `tree->GetEntry()` here.
- Data must go into some kind of contiguous structure ("event" object). `GetEvBuffer()` must return a pointer to this structure.
- Caveats
  - `Init()` must set `fDate`, and `fAssumeDate = true`
  - `Compare()` should never return 0 for different run objects
- With ROOT file input, `gDirectory` is already properly handled in `THaAnalyzer`. No need to `cd()` explicitly.

Simulation Run Class

- Must inherit from `THaRunBase`
- May read arbitray input, *e.g.* text file, ROOT file, database ...
- Input is set up in `Open()`. With ROOT file input, this is the place for `SetBranchAddress` and `SetBranchStatus`
- Digitized event data is fetched in `ReadEvent()`. With ROOT file input, call `tree->GetEntry()` here.
- Data must go into some kind of contiguous structure ("event" object). `GetEvBuffer()` must return a pointer to this structure.
- Caveats
  - `Init()` must set `fDate`, and `fAssumeDate = true`
  - `Compare()` should never return 0 for different run objects
- With ROOT file input, `gDirectory` is already properly handled in `THaAnalyzer`. No need to `cd()` explicitly.

Simulation Run Class

- Must inherit from `THaRunBase`
- May read arbitray input, *e.g.* text file, ROOT file, database . . .
- Input is set up in `Open()`. With ROOT file input, this is the place for `SetBranchAddress` and `SetBranchStatus`
- Digitized event data is fetched in `ReadEvent()`. With ROOT file input, call `tree->GetEntry()` here.
- Data must go into some kind of contiguous structure ("event" object). `GetEvBuffer()` must return a pointer to this structure.
- Caveats
  - `Init()` must set `fDate`, and `fAssumeDate = true`
  - `Compare()` should never return 0 for different run objects
- With ROOT file input, `gDirectory` is already properly handled in `THaAnalyzer`. No need to `cd()` explicitly.

Simulation Run Class

- Must inherit from `THaRunBase`
- May read arbitray input, *e.g.* text file, ROOT file, database ...
- Input is set up in `Open()`. With ROOT file input, this is the place for `SetBranchAddress` and `SetBranchStatus`
- Digitized event data is fetched in `ReadEvent()`. With ROOT file input, call `tree->GetEntry()` here.
- Data must go into some kind of contiguous structure ("event" object). `GetEvBuffer()` must return a pointer to this structure.
- Caveats
  - `Init()` must set `fDate`, and `fAssumeDate = true`
  - `Compare()` should never return 0 for different run objects
- With ROOT file input, `gDirectory` is already properly handled in `THaAnalyzer`. No need to `cd()` explicitly.

Simulation Run Class

- Must inherit from `THaRunBase`
- May read arbitray input, *e.g.* text file, ROOT file, database ...
- Input is set up in `Open()`. With ROOT file input, this is the place for `SetBranchAddress` and `SetBranchStatus`
- Digitized event data is fetched in `ReadEvent()`. With ROOT file input, call `tree->GetEntry()` here.
- Data must go into some kind of contiguous structure ("event" object). `GetEvBuffer()` must return a pointer to this structure.
- Caveats
  - `Init()` must set `fDate`, and `fAssumeDate = true`
  - `Compare()` should never return 0 for different run objects
- With ROOT file input, `gDirectory` is already properly handled in `THaAnalyzer`. No need to `cd()` explicitly.

Simulation Run Class

- Must inherit from `THaRunBase`
- May read arbitray input, *e.g.* text file, ROOT file, database . . .
- Input is set up in `Open()`. With ROOT file input, this is the place for `SetBranchAddress` and `SetBranchStatus`
- Digitized event data is fetched in `ReadEvent()`. With ROOT file input, call `tree->GetEntry()` here.
- Data must go into some kind of contiguous structure ("event" object). `GetEvBuffer()` must return a pointer to this structure.
- Caveats
  - `Init()` must set `fDate`, and `fAssumeDate = true`
  - `Compare()` should never return 0 for different run objects
- With ROOT file input, `gDirectory` is already properly handled in `THaAnalyzer`. No need to `cd()` explicitly.

Simulation Run Class

- Must inherit from `THaRunBase`
- May read arbitray input, *e.g.* text file, ROOT file, database . . .
- Input is set up in `Open()`. With ROOT file input, this is the place for `SetBranchAddress` and `SetBranchStatus`
- Digitized event data is fetched in `ReadEvent()`. With ROOT file input, call `tree->GetEntry()` here.
- Data must go into some kind of contiguous structure ("event" object). `GetEvBuffer()` must return a pointer to this structure.
- Caveats
  - `Init()` must set `fDate`, and `fAssumeDate = true`
  - `Compare()` should never return 0 for different run objects
- With ROOT file input, `gDirectory` is already properly handled in `THaAnalyzer`. No need to `cd()` explicitly.

Simulation Decoder Class

- Must inherit from `THaEvData`
- Main function: `LoadEvent`
  - takes "event" object from simulation run class as input
  - copies digitized MC data to standard crate/slot/channel structures
  - may fill a list of MC tracks or similar
- should export MC truth information (*e.g.* track data) as global analyzer variables (*e.g.* "MC.track.x")
- You must ensure consistency of cratemap, `LoadData`, and detector map(s). This does not happen automatically.
- Enable with `THaInterface::SetDecoder( TClass* )`

Simulation Decoder Class

- Must inherit from `THaEvData`
- Main function: `LoadEvent`
  - takes "event" object from simulation run class as input
  - copies digitized MC data to standard crate/slot/channel structures
  - may fill a list of MC tracks or similar
- should export MC truth information (*e.g.* track data) as global analyzer variables (*e.g.* "MC.track.x")
- You must ensure consistency of cratemap, `LoadData`, and detector map(s). This does not happen automatically.
- Enable with `THaInterface::SetDecoder( TClass* )`

Simulation Decoder Class

- Must inherit from `THaEvData`
- Main function: `LoadEvent`
  - takes "event" object from simulation run class as input
  - copies digitized MC data to standard crate/slot/channel structures
  - may fill a list of MC tracks or similar
- should export MC truth information (*e.g.* track data) as global analyzer variables (*e.g.* "MC.track.x")
- You must ensure consistency of cratemap, `LoadData`, and detector map(s). This does not happen automatically.
- Enable with `THaInterface::SetDecoder( TClass* )`

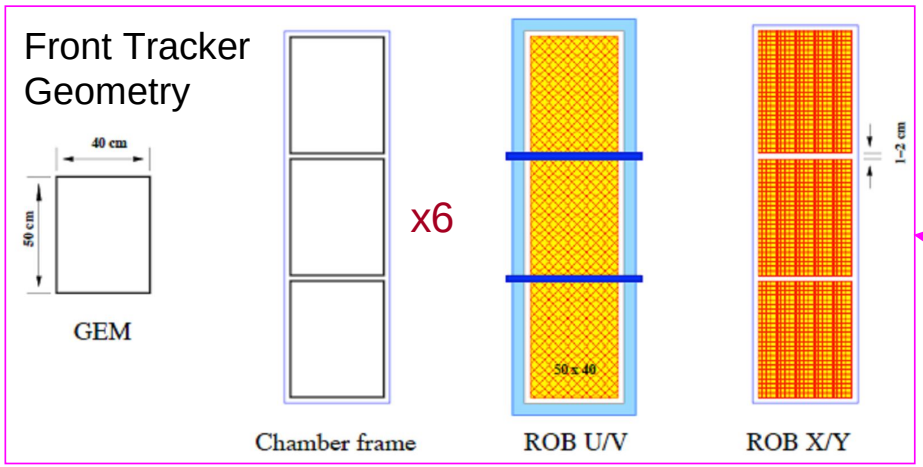
Simulation Decoder Class

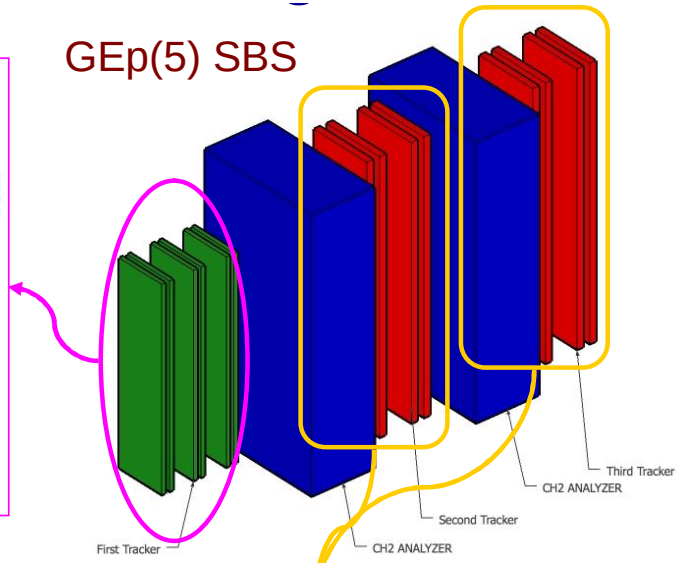
- Must inherit from `THaEvData`
- Main function: `LoadEvent`
  - takes "event" object from simulation run class as input
  - copies digitized MC data to standard crate/slot/channel structures
  - may fill a list of MC tracks or similar
- should export MC truth information (*e.g.* track data) as global analyzer variables (*e.g.* "MC.track.x")
- You must ensure consistency of cratemap, `LoadData`, and detector map(s). This does not happen automatically.
- Enable with `THaInterface::SetDecoder( TClass* )`

Simulation Decoder Class

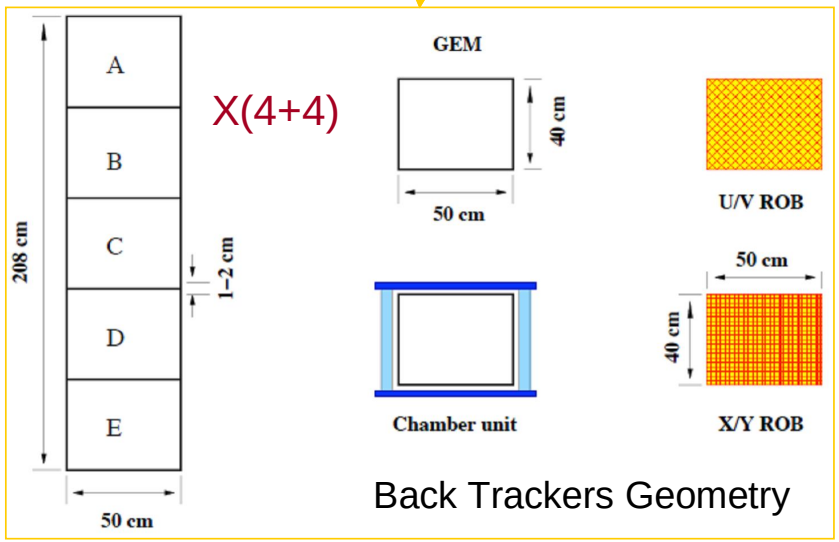
- Must inherit from `THaEvData`
- Main function: `LoadEvent`
  - takes "event" object from simulation run class as input
  - copies digitized MC data to standard crate/slot/channel structures
  - may fill a list of MC tracks or similar
- should export MC truth information (*e.g.* track data) as global analyzer variables (*e.g.* "MC.track.x")
- You must ensure consistency of cratemap, `LoadData`, and detector map(s). This does not happen automatically.
- Enable with `THaInterface::SetDecoder( TClass* )`

Preliminary SBS Chamber Configuration (from E. Cisbani)





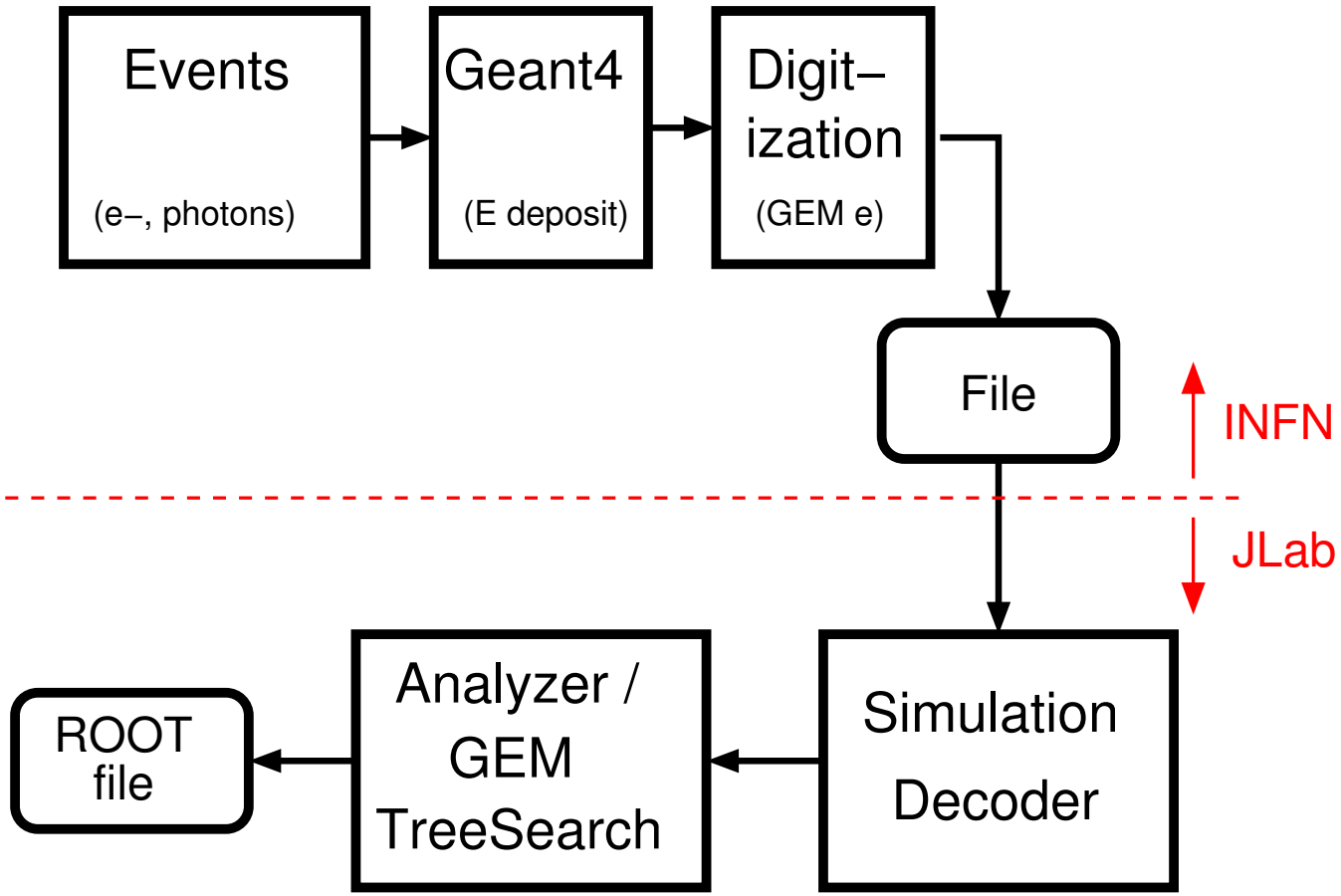
- Modules are composed to form larger chambers with different sizes
- **Electronics along the borders and behind the frame (at 90°) – cyan and blue in drawing**
- **Aluminum support frame around the chamber (cyan in drawing); dedicated to each chamber configuration**



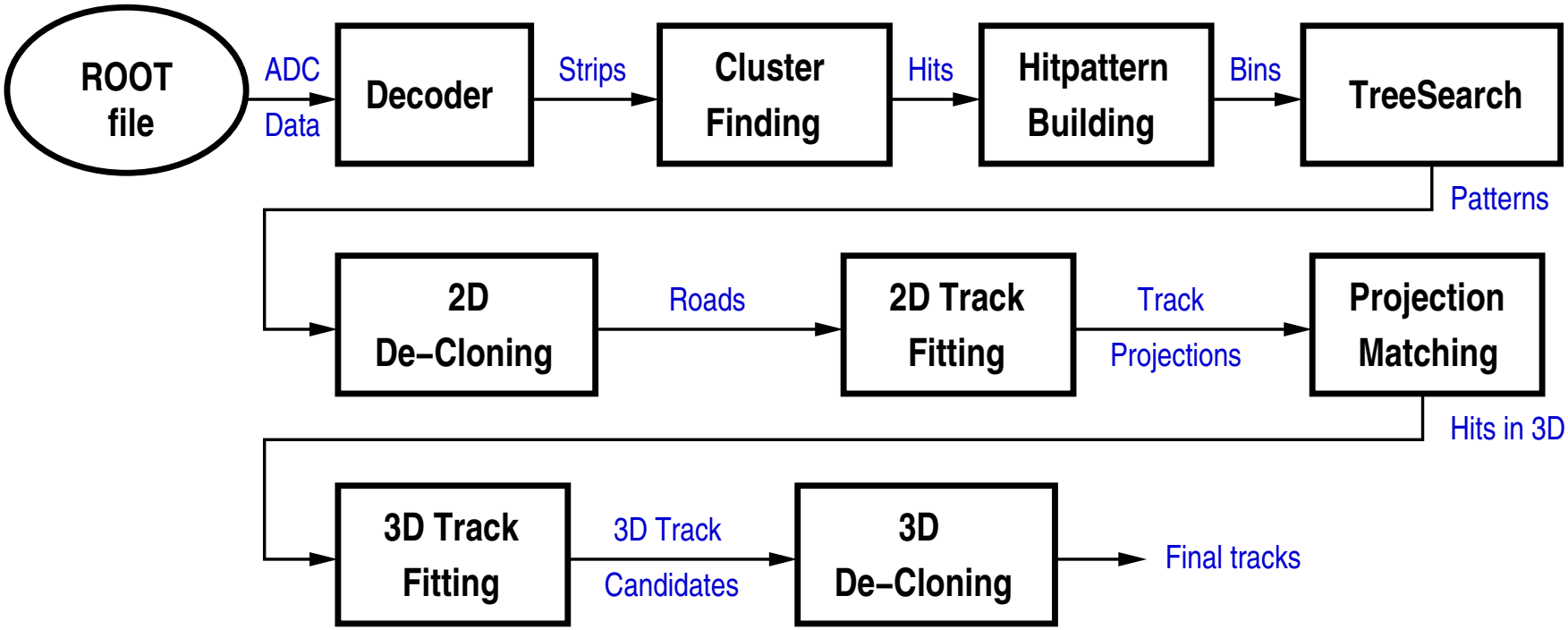
SBSsim Implementation

- Event generator: pre-computed particle distributions based on a certain geometry and published cross-section data
- Interaction simulator: `Geant4` with most things turned on
- Digitization: custom model of GEM response, based on charge diffusion
- Custom Podd interface: `SBSsim`
- GEM-specific track reconstruction, using TreeSearch recursive template matching algorithm

SBS Tracking Monte Carlo Components



Track Reconstruction Flow



SBSsim Analysis Script Example

replay.C

```
void replay_xy() {

  // Prepare for simulation input
  gSystem->Load( libSBSsim.so );
  THaInterface::SetDecoder( SBSsimDecoder::Class() );

  // Proceed as usual
  gSystem->Load( libTreeSearch-GEM.so );
  THaApparatus* SBS = new TreeSearch::SBS( "SBS", "SBS apparatus" );
  THaTrackingDetector* gem = new TreeSearch::GEM( "gem_xy", "GEM trackers with xy-only readouts" );
  SBS->AddDetector( gem );
  gHaApps->Add( SBS );
  ...
  THaRunBase* run = new SBSsimRun( fname );
  run->Init();
  ...
  analyzer->Process( run );
  ...
  analyzer->Close();
}
```

Summary

- Podd provides a convenient interface for arbitrary data input, specifically for simulation data
- Used extensively for SuperBigBite Tracking simulations. Working code library available in CVS, soon on Web
- See Friday afternoon's talk for analysis results