The plan,
In the Step I
“Looking at the bright side of life...”
In the Step II
Conclusion, plans
Plan

➢ **Motivation:**
  - Revisit the tracking code to try to improve the design
  - Unify code on/off line and define an interface for the clients
    - Define a Track! (for on/off line)
  - Define base classes (data and tools) for and tracking developers

➢ **Method:**
  - Modify the current code adiabatically
  - Reusing almost all the code: “adapting” and not “replacing”

➢ **Organization:**
  - Task Force (G. Raven) to:
    - ‘define the classes, requirements and implementation constrains’

➢ **Plan:**
  - Step I: Interfaces for clients
    - Track, State, ITrackExtrapolator
  - Step II: Tracking interfaces
    - Measurement, Node, FitTrack, ITrackProjector, ITrackFitter

➢ **Scale:**
  - 6 months
Step I: The classes, current view

**A TRACK:**
- flag (bitField) TYPE, HISTORY, FLAG
- chi2/ndof, ndof (quality)
- physics State = “The persistent State”
- <States*> = “the *transient* states”
- <LHCbID> = list of LHCbID

**Methods:**
- Access to physics state: p, pt, slopes, position
- Access states: at z, plane, LOCATION

**A STATE:**
- flag (bitField) TYPE, LOCATION
- vector-state, covariance, z

**Methods:**
- Access to physics contents: pt(), p()

**ITrackExtrapolator**

**A Extrapolator: extrapolate a Track/State**

**Main method:** propagate(State, z)

**Methods:**
- propagate the track
- get directly physics values:
  - i.e momentum(track, z)
  - transportMatrix() (F)
Status, Step I

- **Define the client Interfaces**
  - Track, State, ITrackExtrapolator
  - After long discussions, many compromises, *Dec 04*
    - Track finally ‘controlled’ by persistency, a skeleton of a track
  - A complex and powerful bitfield class

- **Having some Tracks**
  - Converters: *Jan 05*
    - *Tr/TrConverters*, TrStoredTrack -> Track
    - *Trg/TrgConverter* TrgTracks -> Track

- **Extrapolators**
  - Interface:
    - *Kernel/LHCbInterfaces*, *Jan 05*
  - Implementations: in
    - *Tr/TrackExtrapolators*; base tool: TrackExtrapolator
    - Linear: *TrackLinearExtrapolator* *Jan05*
    - Others: Parabolic, FastParabolic, Herab, FirstClever, *Mar05*
      - Propagators work, but more testing needed
      - Intersection with a plane temporally simplified

- **Saving the Tracks:**
  - In progress, we got some problems with persistency, need some help with custom DSTs
Making the Tracks, Tracks!

- In private area:
  - *Event/TrgEvent*, TrgTrack inheriting from Track, **Mar05**
  - A TrgTrack now is Track!, TrgState is in fact just a State
- As an exercise:
  - *Trg/TrgVelo* using the new TrgTrack, Mar05
- *In the yellow light*, waiting the green light: (end Mar 05)
  - Commit the new TrgTrack, update the Trg Packages
- Implications:
  - Trg will get a new version
  - I see no particular problems…, just a delicate work
    - We use a Python tool to help us (see next transparency)
  - It will require revisiting/fixing LHCbID to do:
    - The linking with MC:
    - The Buffer Tampering

Using the new Tracks:

- Ideal Pattern Recognition
  - *In Tr/TrackIdealPR* If we can not do it here, forget it!
  - Minor changes to make independent of old TEM
- In Panoramix
  - In Vis/SoEvent (SoTrackCnv) **Mar05**
  - Of course we need to draw Tracks, (one for all?)
Status, Step I (and cont)

- **Migrating**
  - Updating/replacing Clients code: TrCheck, ParticleMaker, Calorimeter, Rich…

- **With a Python tool**
  
  ```python
  python translate_to_new_tracking.py –f *.cpp *.h –r False
  ```
  - Create new cpp and header files
  - Replacing the old Trg/TrStored code to the new Tracking
  - It works quite nice 😊 but of course do not expect a miracle
  - We tuned the tool with Trg.
  - We need a guinea-pig (some client code) to be replaced and to tune the tool
  - When the tool is tuned up, we advertise it, you run the tool in your package, try to compile…
    - If still too many complicated errors show up…
    - Just contact us and we will try to make the compilation
    - You check them later…
  - It is a general tool to replace any work for another in files

  ```python
  translate_to_new_tracking.py –f *.cpp *.h –i red –o green –r False
  ```

- **Idea:**
  - All code that uses the Track interface is valid for any type of Track!
  - Ie. Drawing in Panoramix
Interactive reconstruction

“How Reconstruction sans frontières”

- How to make the reconstruction interactive?: via Python
  - From Python you can execute and use C++ code
  - Python is an interactive language, has introspection
    - ```python
dir(track)
    ```
  - Other pros: Python is very intuitive, dynamically typed, no pointers, heterogenic containers, dictionaries…
  - One develop code ~4 times faster than in C++
- GaudyPython and Bender
  - Pere already exposed Gaudi framework to Python
    - ```python
    gaudi.run(1)
    ```
  - Vanya has exposed most of DaVinci tools/data for analysis, including his ‘meta-language’ LoKI in Python
- Idea:
  - Expose the base track data classes and interfaces tracking tools to Python
  - The base tracking classes allow to write code in a base level for reconstruction
    - You will be able to do this code in Python and check it interactively
  - You can debug/test and develop tracking code with the Base classes in Python
    - That is what we are doing already!
Looking at the bright side of life…

- **Already done:**
  - Expose ITrackExtrapolator, Track/State to Python, thanks to Vanya and Pere, Mar 05
  - Example:
    - `pol = extrapolator("TrackParabolicExtrapolator")`
    - `state = track.physicsState().clone()`
    - `pol.propagate(state, z=1000.)`

- **And Panoramix?:**
  - Panoramix has methods exposed to Python (Guy also was in the business :)
    - So we can ‘use’ Panoramix from Python
  - The other direction is needed (and can be made), nice requirement
    - If you click in a Track in Panoramix you can get the Object in your Python prompt!

- **In the future:**
  - We will expose to Python:
    - Measurement and ITrackProjector, Measurement/FitTrack and IFitter
  - Some things that will be possible to do *interactively*:
    - Pattern Recognition algorithms:
      - extrapolate this track to ‘here’, get the best measurement, update the track
    - Refitting
      - Replace/Remove this measurement and refit
      - Change the fitter and refit, change the extrapolator (this has better error estimate…) and refit
    - Alignment:
      - I want to try this new set of parameters, replace the Projector, refit the track or the Event
- *Python:*
  - Just import modules: PyROOT, Hippys

```python
pol = extrapolator("TrackParabolicExtrapolator")
state = track.physicsState().clone()
z = state.z(), xx = [], zz = []
for i in range(50):
    z = z + 10
    pol.propagate(state,z)
    xx.append(state.x())
    zz.append(z)
hxy(z,x)
```

**Preliminary: Tracks in Panoramix**

**A scatter plot from a Python prompt**
Step II: The classes

**A FitTrack:**
- Measurement*
- Nodes*

**Methods:**
- Add/remove measurements

Nodes are the ‘Transient’ state of the Track while fitting it

**A Measurement:**
- type (i.e. RVelo)
- z, measure, error (double)
- LHCbID

**ITrackProjector:**

**A Projector:** Project a state into a measurement

**Main method:** project(State, Measurement)

*Internally deals with the Alignment/Calibration*

**Methods:**
- residual, chi2, node, ProjectionMatrix (H)

**A Node:**
- type (i.e. RVelo)
- Measurement* (“refined”)
- State*
- residual, error

**Methods:**
- chi2(), ...

**IFitter**
Step II, Status and Plans

- Define the client Interfaces:
  - Measurement, Node, FitTrack, ITrackProjector, Ifitter
    - http://cern.ch/eduardo.rodrigues/lhcb/tracking
  - Please contact us if you want to discuss them…
  - A draft version in:
    - Event/TrackEvent and Event/FitTrackEvent

- Next steps:
  - Creating Measurements from Clusters
  - Coding the projectors
  - Expose them to Python
  - Check how they work…
  - Make the Kalman Filter work with Projectors and Extrapolator
    - The present code already has almost the ‘same’ philosophy.
    - A delicate work from Tr/TrFitter to Tr/TrackFitter
  - Study of how to refit the Track starting only from Track
    - It could imply to write in persistency some extra info (Marcel, Matt)
Conclusions and Plans

- **Status:**
  - Steady work, many fronts, small forces (E.R, JA, Edwin Bos – Nikhef-)
  - Guide by G. Raven as a Task Force.

- **In the Plans**
  - **Step I**
    - TrgTracks to be Track
    - Tune the Python tool to migrate code to the new tracking
    - To have Ideal Pattern Recognition
  - **Step II**
    - Code Measurement, Projectors,
    - Adapt Fitter package
    - PR packages will follow accordingly with Task Force
  - **In the Python front:**
    - Expose the base classes inside *Bender*
      - “Bender”: exposing LHCb code (DaVinci, LoKi, Brunell) into Python
    - Interact with Panoramix

- **This A C++ chirurgic operation:**
  - For the moment the patient behaves fine, no anesthesia applied yet