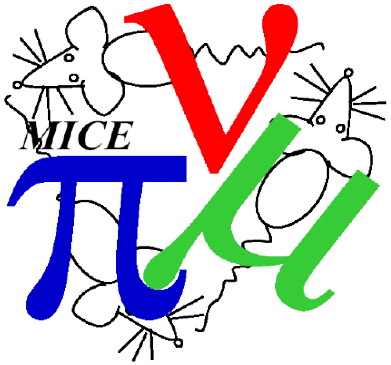
MAUS – Analysis Issues

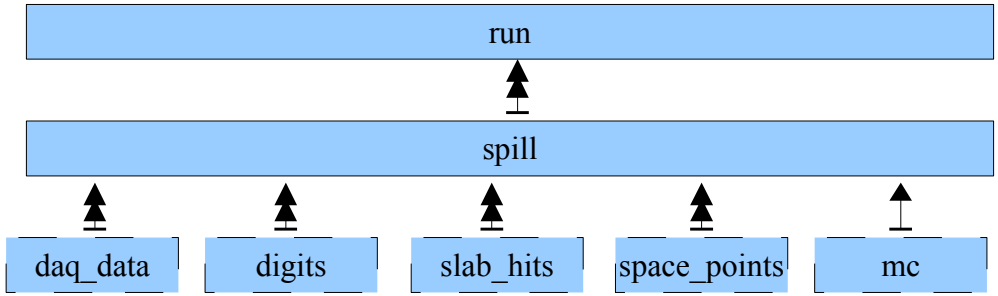


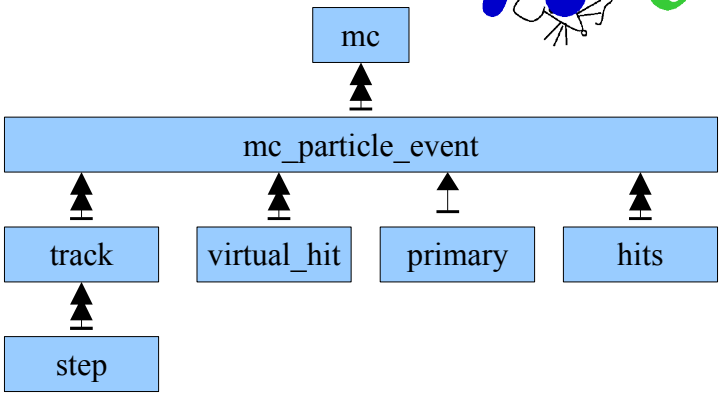
Chris Rogers,
ASTeC,
Rutherford Appleton Laboratory

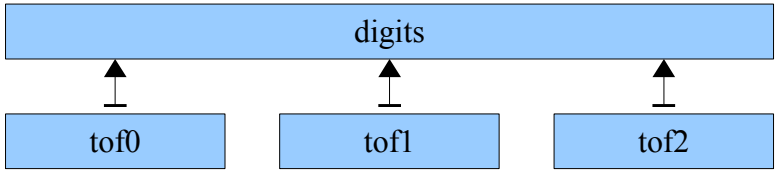
Overview

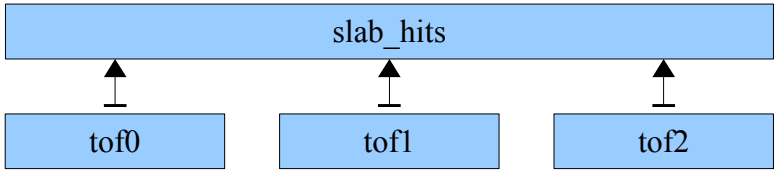
- Trying to specify the “detector integration” task
- Short term
  - Data structure
  - Data representation
  - Batch production
  - Beam input via G4BL
  - Geometry responsibility
- Longer term
  - Global Recon
  - Optics Tool

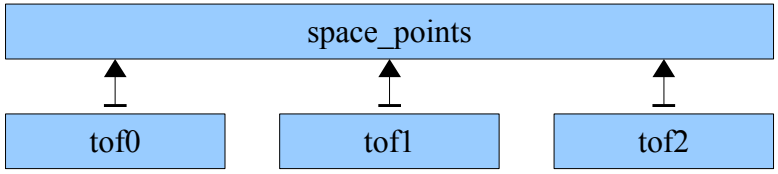
Current data structure

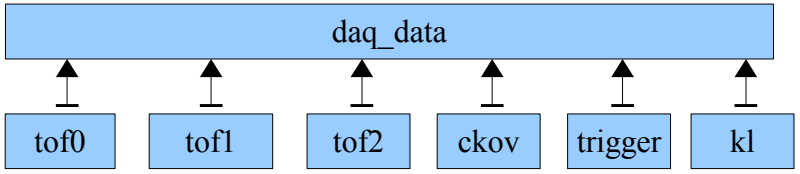




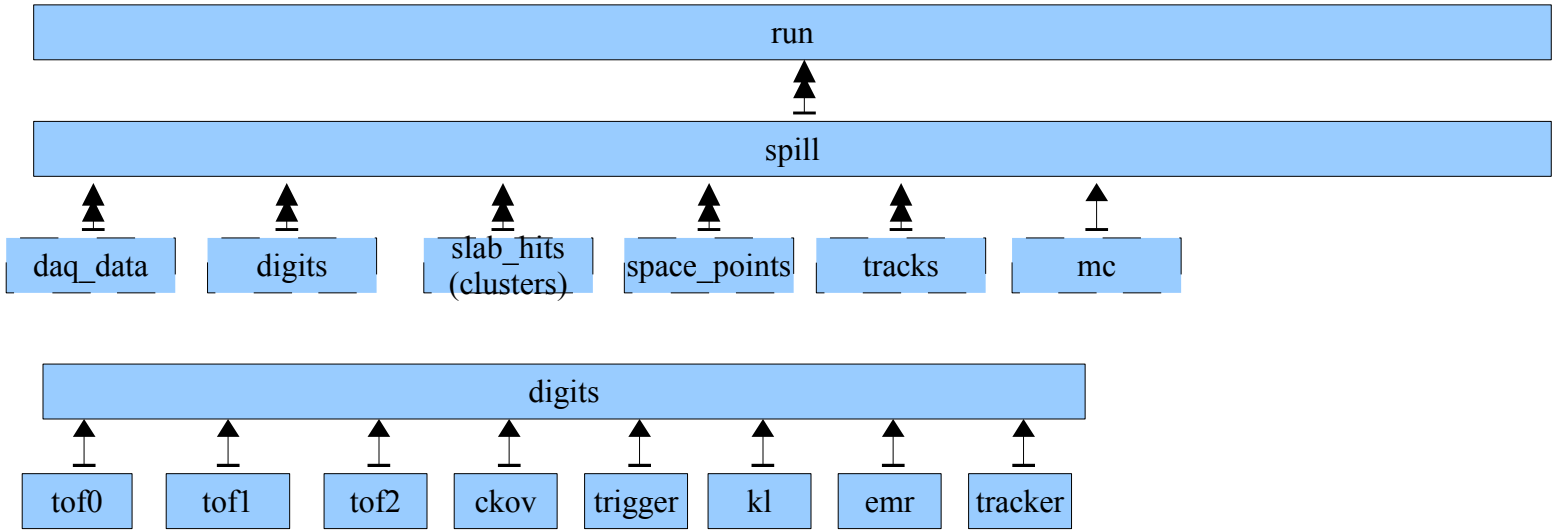








Final data structure

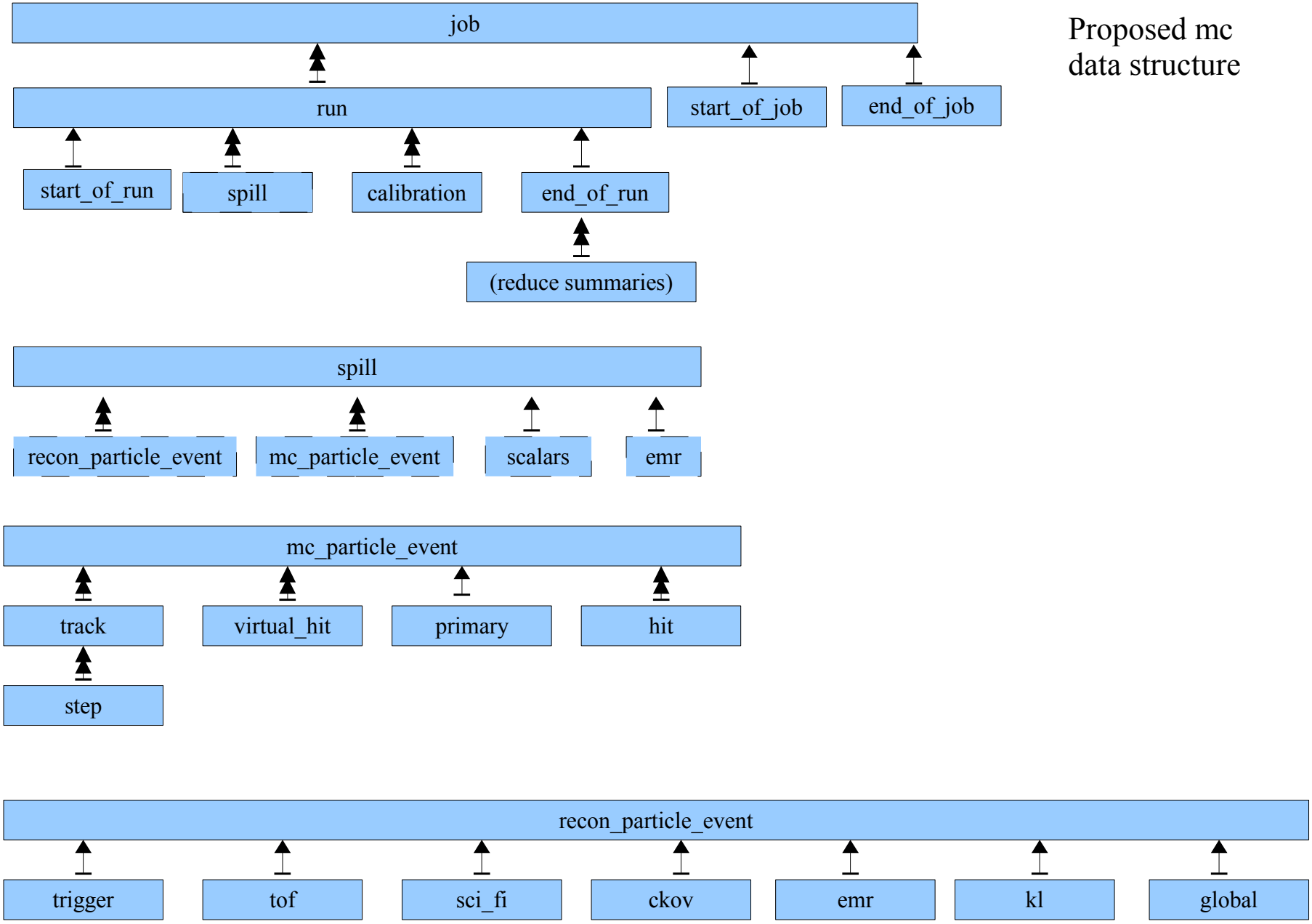


Etc...

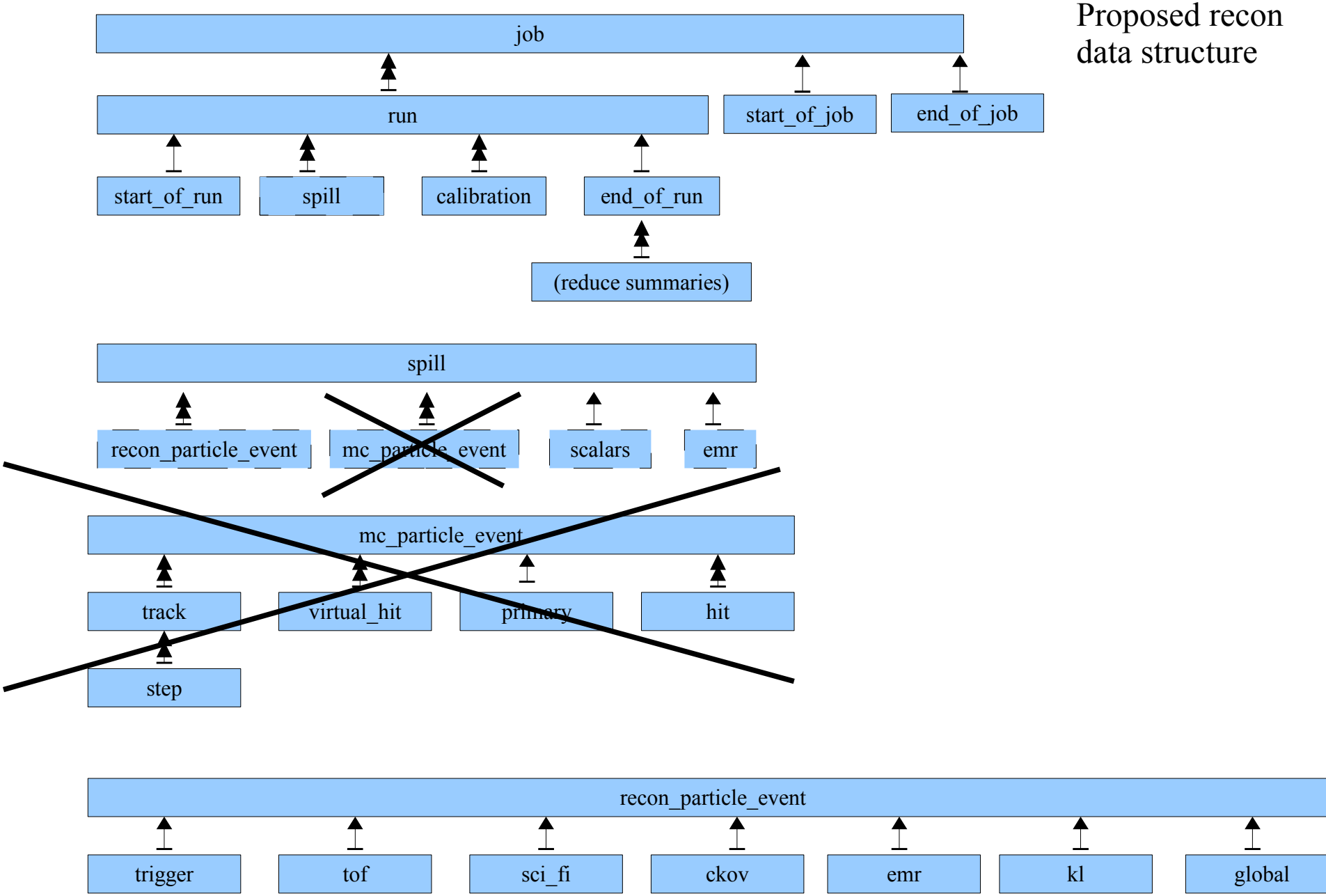
Comments

- Tough to extract the digits/slab hits/cluster for *this* particle event
  - Need to do a search and construct a particle event every time we want to do any reconstruction - madness
- We struggle to know e.g. what digits were used to reconstruct a particular slab hit
  - Can't do noise studies easily
- We have no way to handle per-run calibration data (DAQ calibration event)
- We have no way to store run metadata
- But:
  - Nice for global recon – we can e.g. trivially extract all space points
  - Some existing code in this data structure

Proposed mc data structure



Proposed recon data structure



Data Representation

- Data currently represented in json
  - No way to enforce any structure
  - Developers can e.g. change the data structure during reconstruction
  - No way to enforce conventions
    - Like “all Space Points should have (x,y,z,time) field”
  - Some concerns over file size, access speed
    - Can probably be addressed using native json, e.g. use some compression routine
  - Naturally leads away from good practice – encapsulation, etc
- Propose using a ROOT data tree
  - More conventional
  - Maintain json functionality

Data - Implementation

- Changing data structure needs to be done ASAP and is quite a hard change (i.e. can't phase it in easily)
  - Rogers task? In collaboration with other developers?
- Changing data representation can be phased in slowly – thanks to nice modular structure in MAUS
- At the moment data serialisation goes like
  - PyJson -> Reconstruction module -> PyJson -> string -> JsonCpp -> Reconstruction module -> JsonCpp -> string -> JsonCpp -> Reconstruction module -> ...
  - Choose to use JsonCpp or PyJson by developer preference
  - Conversion to string is just a lazy
- Generalise this serialisation to allow different types
  - Converter(..., DataTypeA) -> Reconstruction module -> Converter(DataTypeA, DataTypeB) -> Reconstruction module -> Converter(DataTypeB, DataTypeC) -> ...
- Then implement Converters for ROOT, JsonCpp, PyJson
  - If DataTypeA == DataTypeB we have a null operation
  - No longer need to go back to strings all the time so should result in code speed up

Batch Production

- Propose a batch production job for MC and Reconstruction
  - After a stage of settling analysis users won't be interested in details of this or that reconstruction routine
  - Nice to have “official” recon data sets for analysis
- Will be done on GRID
  - Automatically trigger as data is taken
  - Use a reference version which may not be the latest – chosen by analysis group
  - Rerun over data sets for new code version/calibration/etc can be requested by analysis group
- Needs:
  - GRID setup
  - Input beam (next slide)
  - Geometry import
  - Digitisation of TOF and other detectors
  - ROOT IO (propose write ROOT data trees)
- Aim was mid-February
  - All looks quite tight

Input Beam (g4bl)

- I think we use G4BL for input beam to batch production
  - Need reasonable rate and time distribution
    - For multiparticle effects (pile up, dead time)
    - For tracker dead time model
  - Extract rate from GVA1
    - Fit to binomial distribution?
  - Extract time distribution in each spill from fit to TOF0 t
    - Fit to some exponential distribution? Saw tooth?
  - Means we need to do reconstruction job and then use reconstructed GVA1 and TOF0 to do the Monte Carlo
    - Circular dependency is unpleasant
      - Use recon as input to MC which is used to understand recon... urg...
    - Is there a better way?

Geometry Responsibility

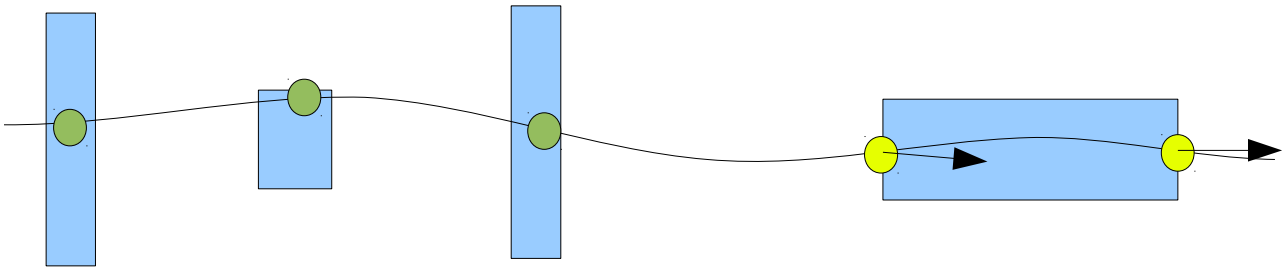
- We have a set of routines for pulling geometry from a reduced CAD model of MICE
- It needs someone to check that the geometry is correct
  - Detectors are responsible for checking that their geometry is correct
  - Beamline expert is responsible for checking geometry and fields in beamline is correct
  - Analysis group is responsible for checking cooling channel geometry and fields are correct
- Geometry release cycle goes like
  - CAD release from Jason Tarrant
  - Converted into Geant4 readable and uploaded to ConfigDB
  - Checked by experts
  - Tagged as a release of new geometry
- First attempt will follow survey being taken right now

Longer Term Items

- Global Recon
- Optics tool

Global Recon

- We have some set of measured space points (TOF), tracks (tracker), Light yields (EMR, KL, Ckov) with some set of errors
- How do we tie these things together?
- Two somewhat separate things:
  - Track fitting
  - PID



Example Uses

- Calculate change in 4D emittance, amplitude
- Calculate change in longitudinal (energy-time) emittance, amplitude
- Detector alignment in the presence of fields
- Physics process studies in the presence of fields (e.g. multiple scattering in the absorber)
- PID (esp tracker vs TOF)

Basic specification

- We seek to get the following variables
  - x, y, z, time
  - px, py, pz, energy
  - Preferred pid (gives charge, mass, etc)
  - On user-defined planes – typically spacial planes, probably not z-planes
    - Engineering geometry has ***x*** as the beam axis
- We seek errors
  - Covariance matrix of errors in (x, y, time, px, py, energy)
    - E.g. say we know x exactly but px we have some error
    - Propagate that measurement and we find we introduce uncertainty in x
    - In fact there is a correlation between the x-px uncertainty
    - Encode as a “covariance matrix”
  - p(electron), p(muon), p(pion)? sigma(mass)? Not sure how to write down error in pid

Aim of track fitting

- Basic aim is to minimise χ^2 of the measurement errors
  - So in 1D minimise sum$[(X_i - x_i)^2 / Var(dx_i)]$ where
    - x_i is a measurement made at detector i,
    - X_i is the globally reconstructed measurement at detector i
    - $Var(dx)_i$ is the error on x_i
  - In ND this becomes
    - sum$[(U_i - u_i) V^{-1} (U_i - u_i)^T]$
    - u_i is a vector of measurements made at detector i,
    - U_i is a vector of globally reconstructed measurements at detector i
    - V_i^{-1} is the error covariance matrix at detector i
- Might alternatively try to minimise determinant of the error covariance matrix

Tools

- Need to propagate tracks through our equipment
  - Runge Kutta
  - Accelerator transport matrix for online
- Need to propagate errors through our equipment
  - Accelerator transport matrices – this transports like a beam envelope
  - Required for some minimisation routines e.g. Kalman
- Need minimisation routines
  - Minuit is quick and dirty
  - Kalman is more elegant but needs some work
  - TOF recon algorithm is also nice
    - Need to generalise
- Some seed finding routines
  - Normally we just use the tracker
  - Might want something for if tracker is not there/cross checks

Track/Error propagation

- We can already do Runge Kutta through arbitrary field maps, either using G4 or custom tracking routines
- We can already calculate transfer matrices through arbitrary field maps
  - Numerical differentiation of the tracking
    - Generate a few different tracks
    - Do $u_{out} = M\, u_{in}$
    - We can use G4 to calculate u_{out} from u_{in}
    - We can solve as a simultaneous equation for **M**
  - Second order correction
    - Instead of solving as a simultaneous equation, can do linear least squares (it's all nice linear sums of terms)
    - Gets a more correct answer
    - Costs more processing - tracking more particles
  - Direct integration (not implemented)
- All implemented in Optics routines
  - **Needs significant clean up and testing**

Other things

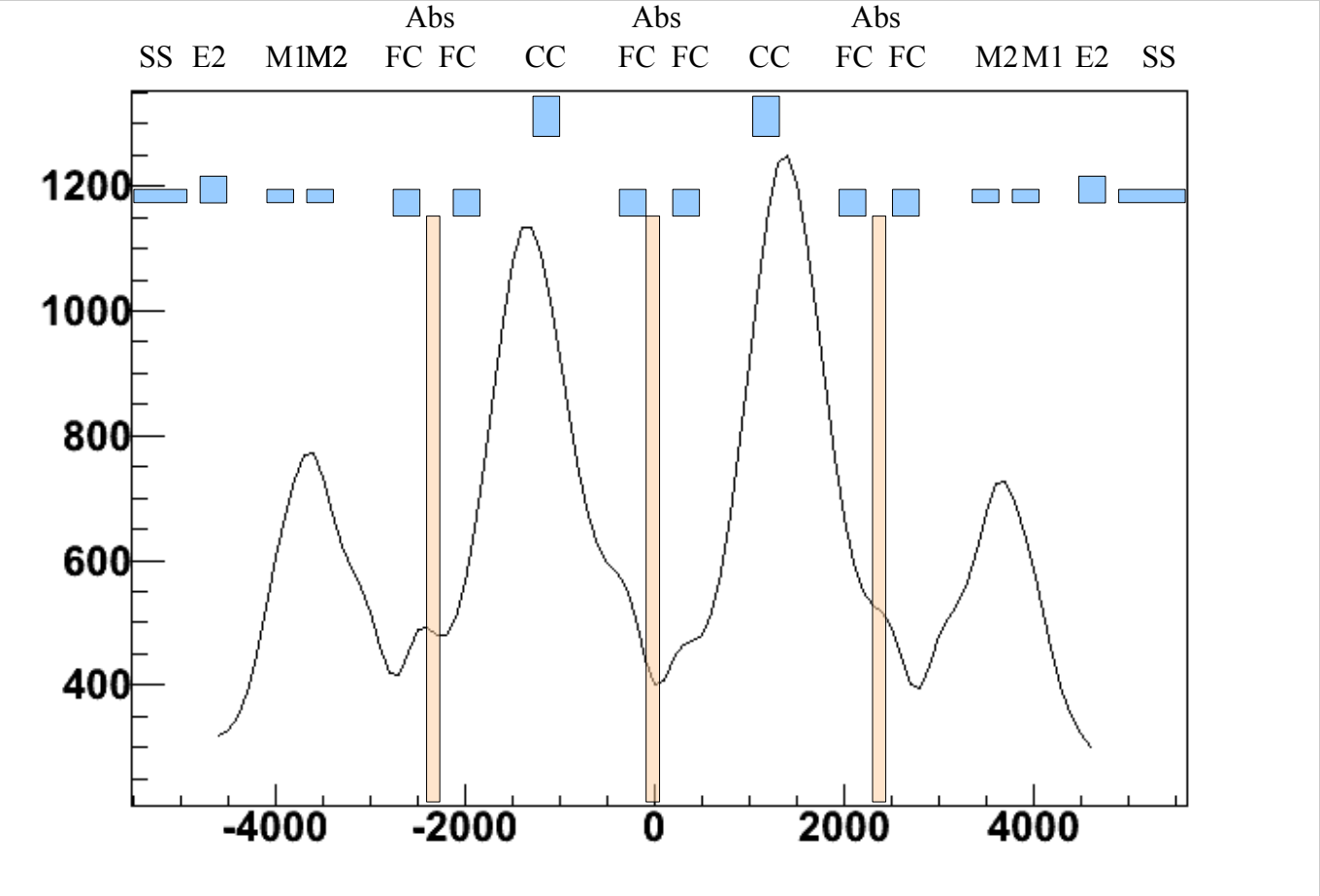
- Transport through materials
  - Tracks should lose energy
  - Error matrices should get a scattering/straggling kick
- PID
  - Basic routine I think we try candidate particles and look for well-fitted tracks
    - E.g. try to reconstruct assuming muon, electron, pion
    - Look at errors and take the best one
  - Not sure how to do something more sophisticated
    - Include mass as a tracking parameter?
    - How to include Ckov and KL/EMR?

Next Steps

- Specification document is in progress
- Definition of data structure
  - PID output might have to be vague until I understand the right way to do it
- Definition of class structure
  - We already have some optics infrastructure in place, currently in a tidy-up phase
- Seek comments early and often
  - I am a novice – detector experts in MICE have done it all before

Optics Tool

- Request was made for a light weight tool to evolve beam envelopes for use in control room
- Propose following tool
  - (powerpoint prototype of a GUI)
  - Allows to define a set of magnet currents and input beam
  - Then plot resultant beam envelope, momenta, etc down the accelerator
  - Geometry shown is Stage 6 geometry
  - In principle we should be able to handle any geometry e.g. would hope to include Q4-9, etc
- Seek comments



Beam

Z (mm) -2750

definition penn

beta (mm) 420

alpha 0

em. (mm) 6

Beamline

E1 (A) 149.05

M1 (A) 149.05

SS (A) 222.01

M2 (A) 222.01

E2 (A) 109.12

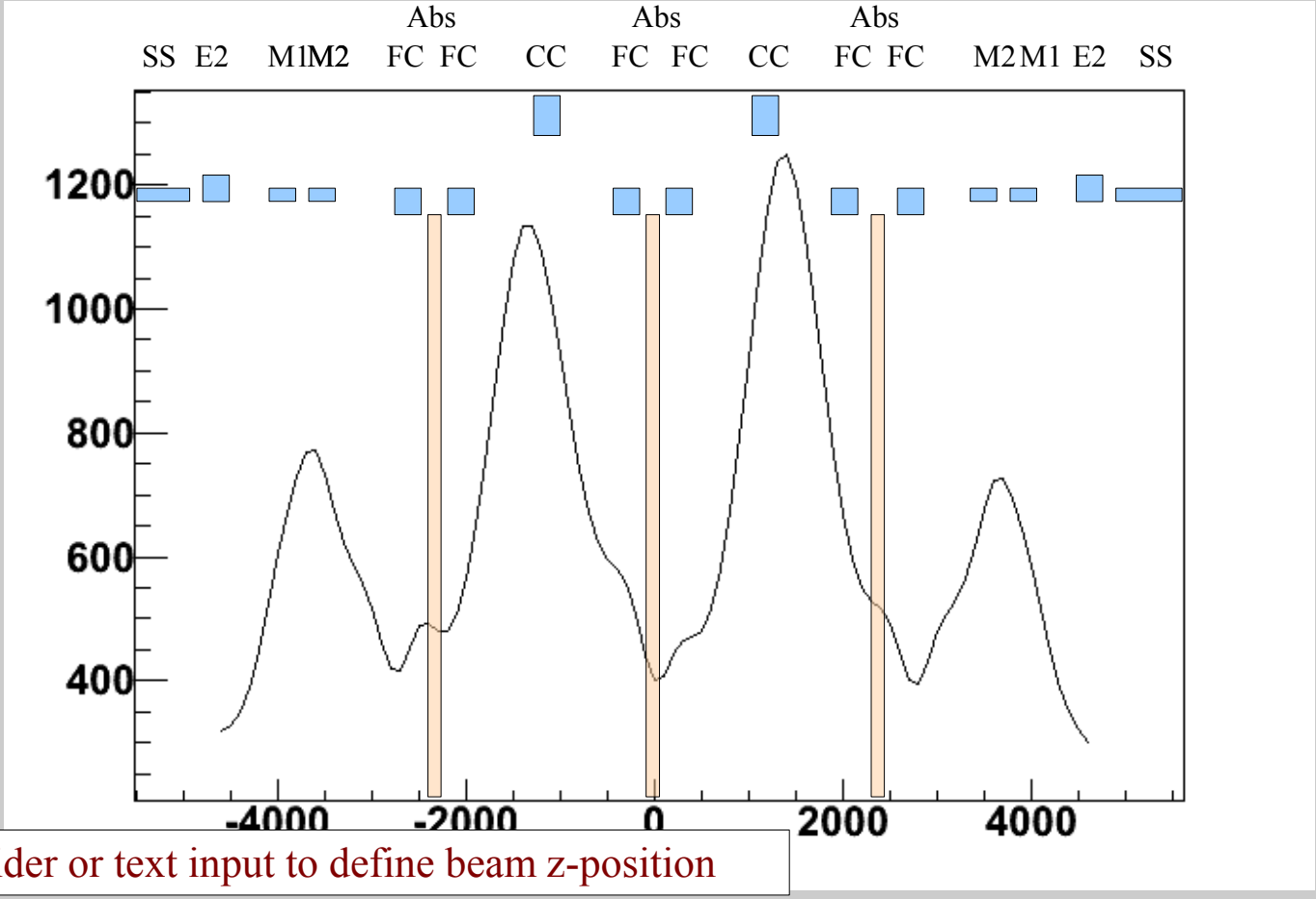
FC (A) 109.12

CC (A) 109.12

Plot

ordinate beta_perp

Show optical elements



Slider or text input to define beam z-position

Beam

Z (mm) -2750 definition penn

beta (mm) 420

alpha 0

em. (mm) 6

Beamline

E1 (A) 149.05 M1 (A) 149.05

SS (A) 222.01 M2 (A) 222.01

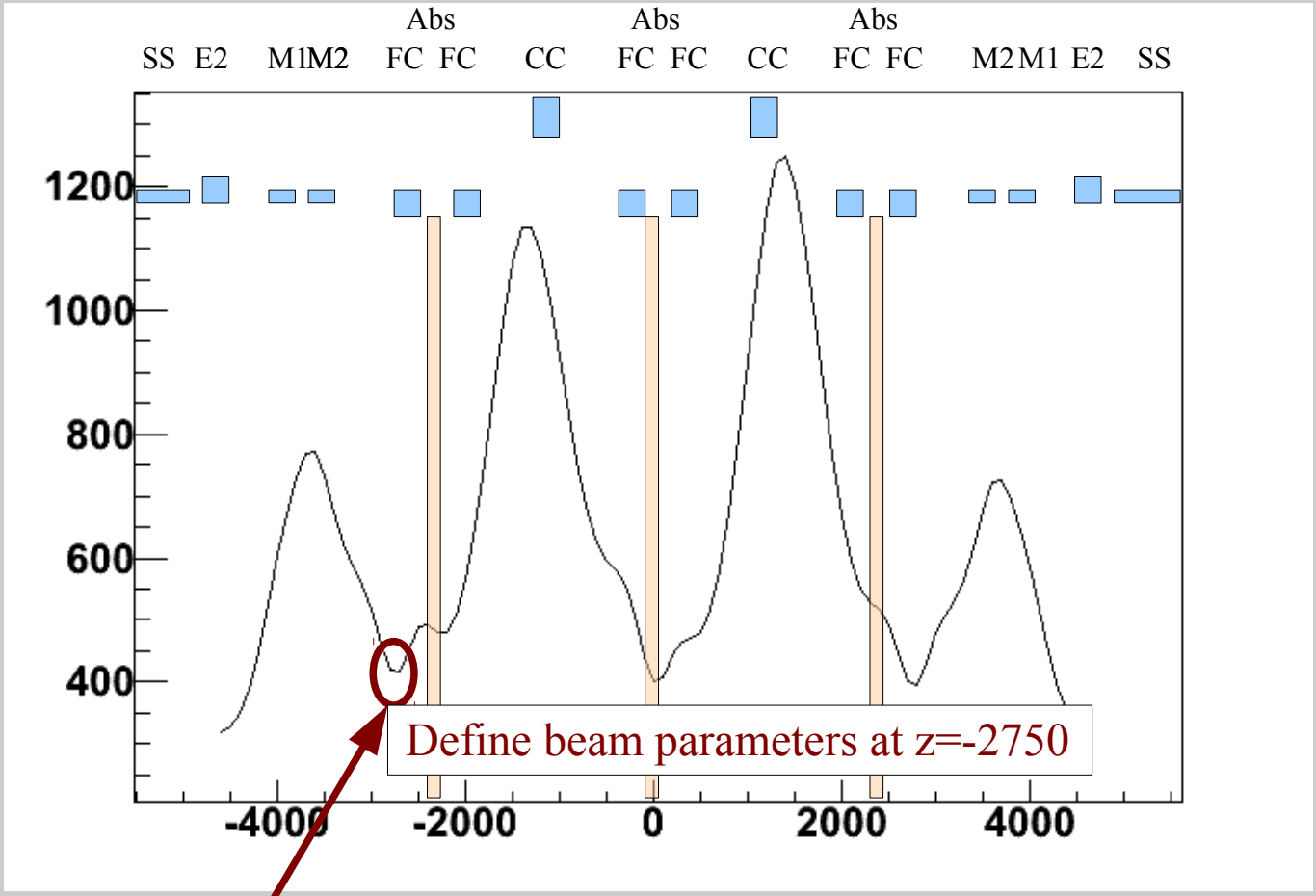
E2 (A) 109.12 FC (A) 109.12

CC (A) 109.12

Plot

ordinate beta_perp

Show optical elements



Beam

Z (mm) -2750

definition penn

beta (mm) 420

alpha 0

em. (mm) 6

Beamline

E1 (A) 149.05

M1 (A) 149.05

SS (A) 222.01

M2 (A) 222.01

E2 (A) 109.12

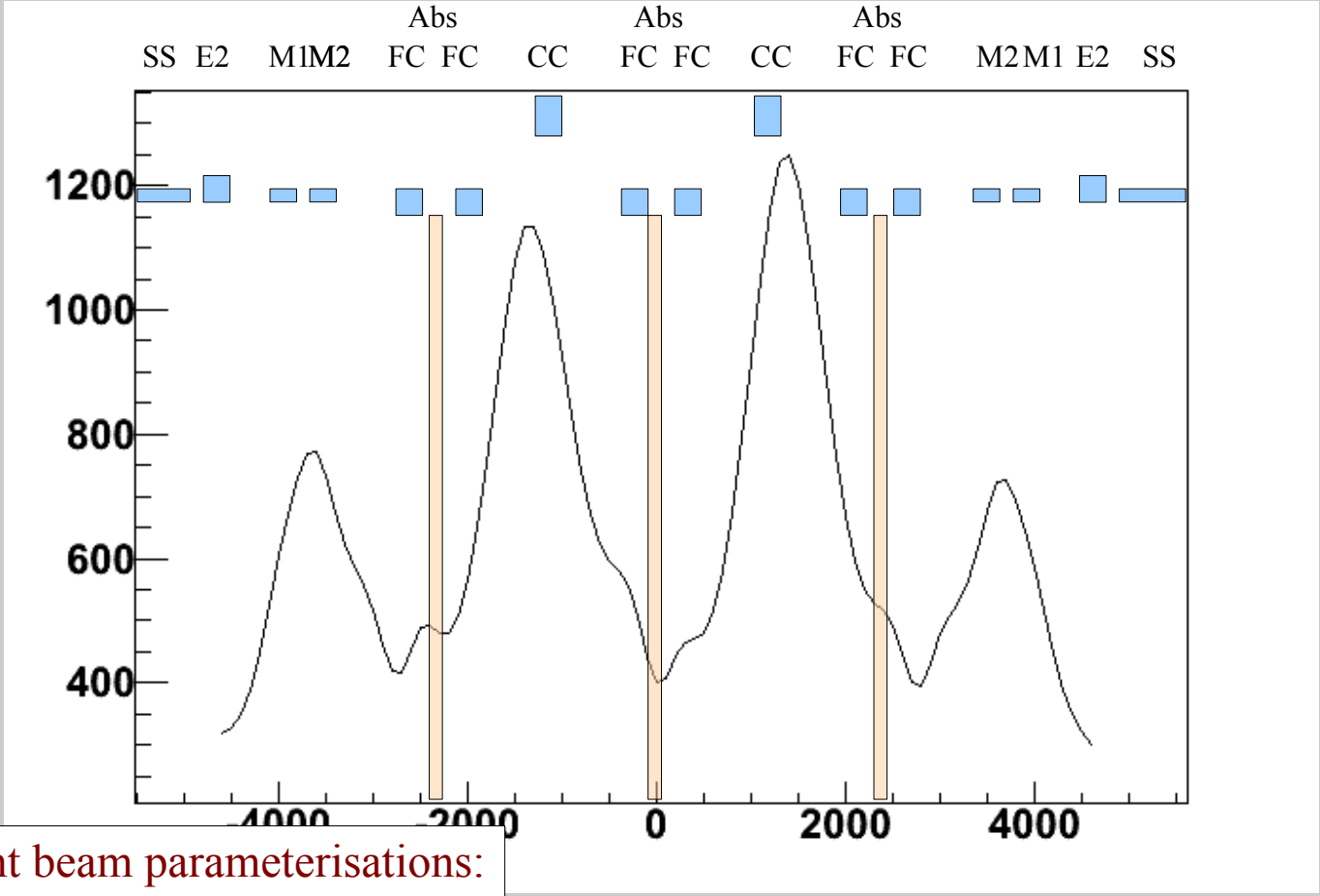
FC (A) 109.12

CC (A) 109.12

Plot

ordinate beta_perp

Show optical elements



Different beam parameterisations:
penn
twiss

Z (mm) -7000 definitio twiss

beta_x (mm) 2700 beta_y (mm) 350

alpha_x 0 alpha_y 0

em_x (mm) 2 em_y (mm) 4

Beamline

E1 (A) 149.05 M1 (A) 149.05

SS (A) 222.01 M2 (A) 222.01

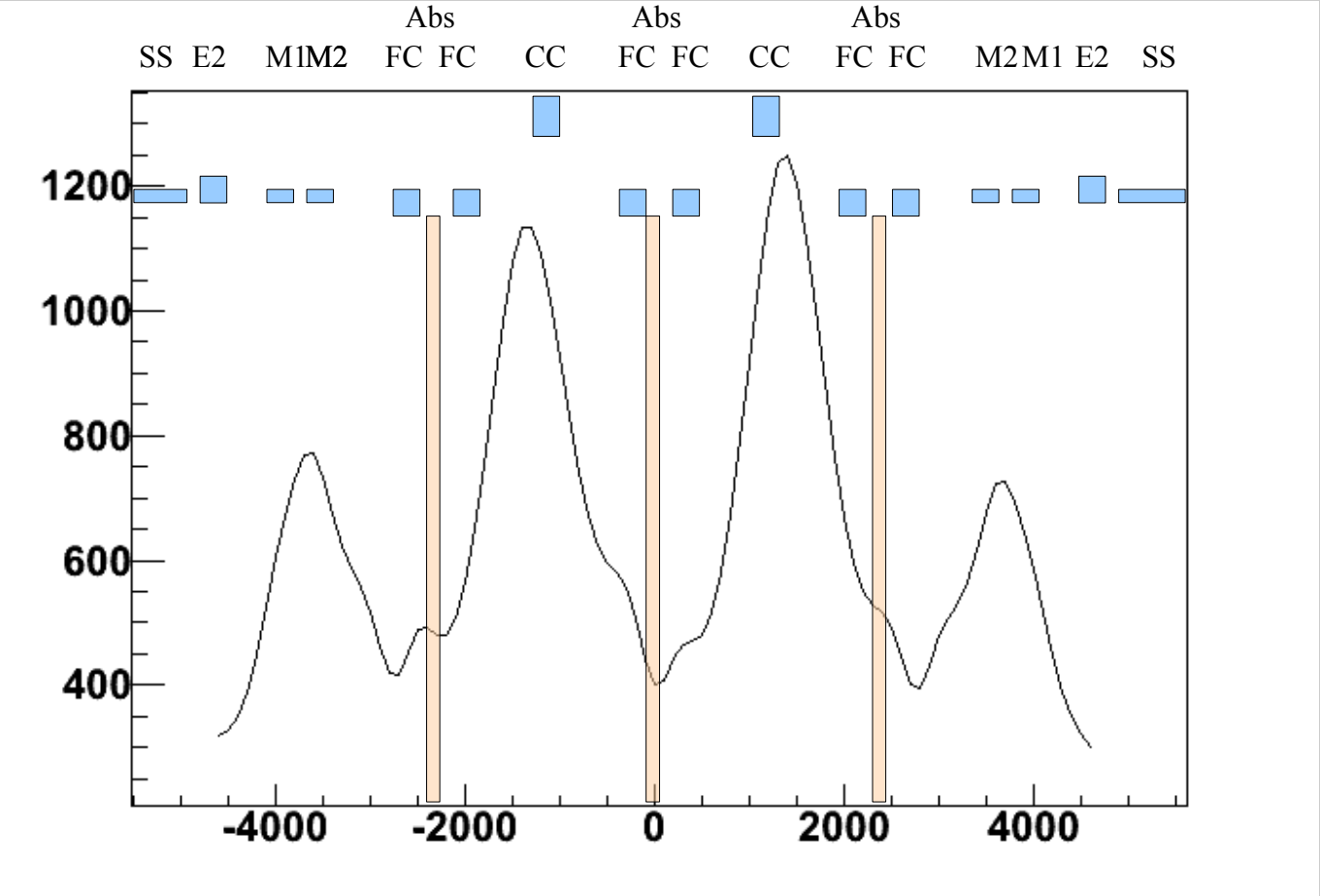
E2 (A) 109.12 FC (A) 109.12

CC (A) 109.12

Plot

ordinate beta_perp

Show optical elements



Beam

Z (mm)

beta (mm)

alpha

em. (mm) 6

Control magnet currents
•Should use currents like the ones typed into controls PC

Beamline

E1 (A) 149.05 M1 (A) 149.05

SS (A) 222.01 M2 (A) 222.01

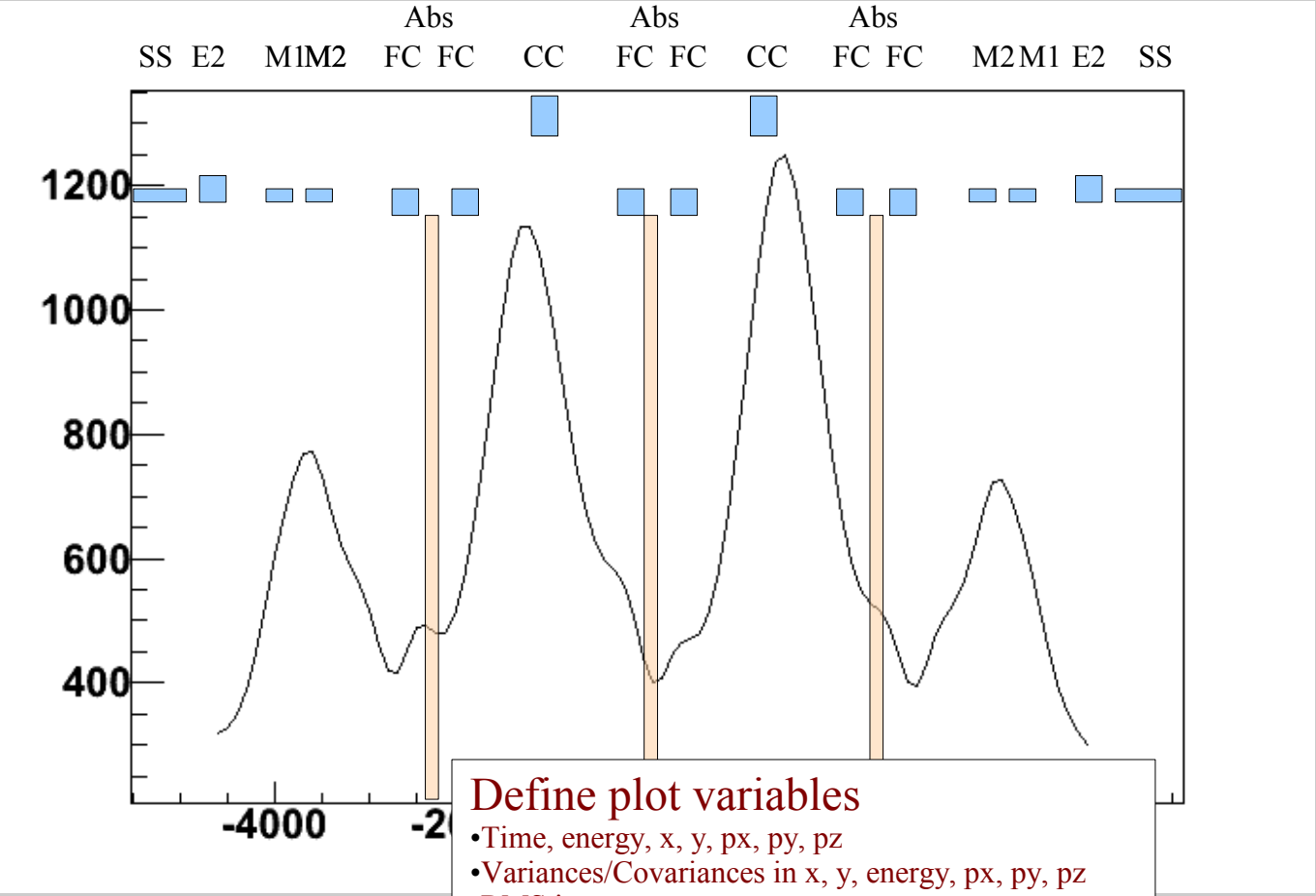
E2 (A) 109.12 FC (A) 109.12

CC (A) 109.12

Plot

ordinate beta_perp

Show optical elements



Define plot variables

- Time, energy, x, y, px, py, pz
- Variances/Covariances in x, y, energy, px, py, pz
- RMS in x, y, energy, px, py, pz
- Beta function in x, y, transverse 4D
- Alpha function in x, y, transverse 4D
- Gamma function in x, y, transverse 4D
- Emittance in x, y, transverse 4D
- Kinetic angular momentum
- Amplitude momentum correlation
- Dispersion in x, y
- Dispersion prime in x, y

Beam

Z (mm) -2750

definition penn

beta (mm) 420

alpha 0

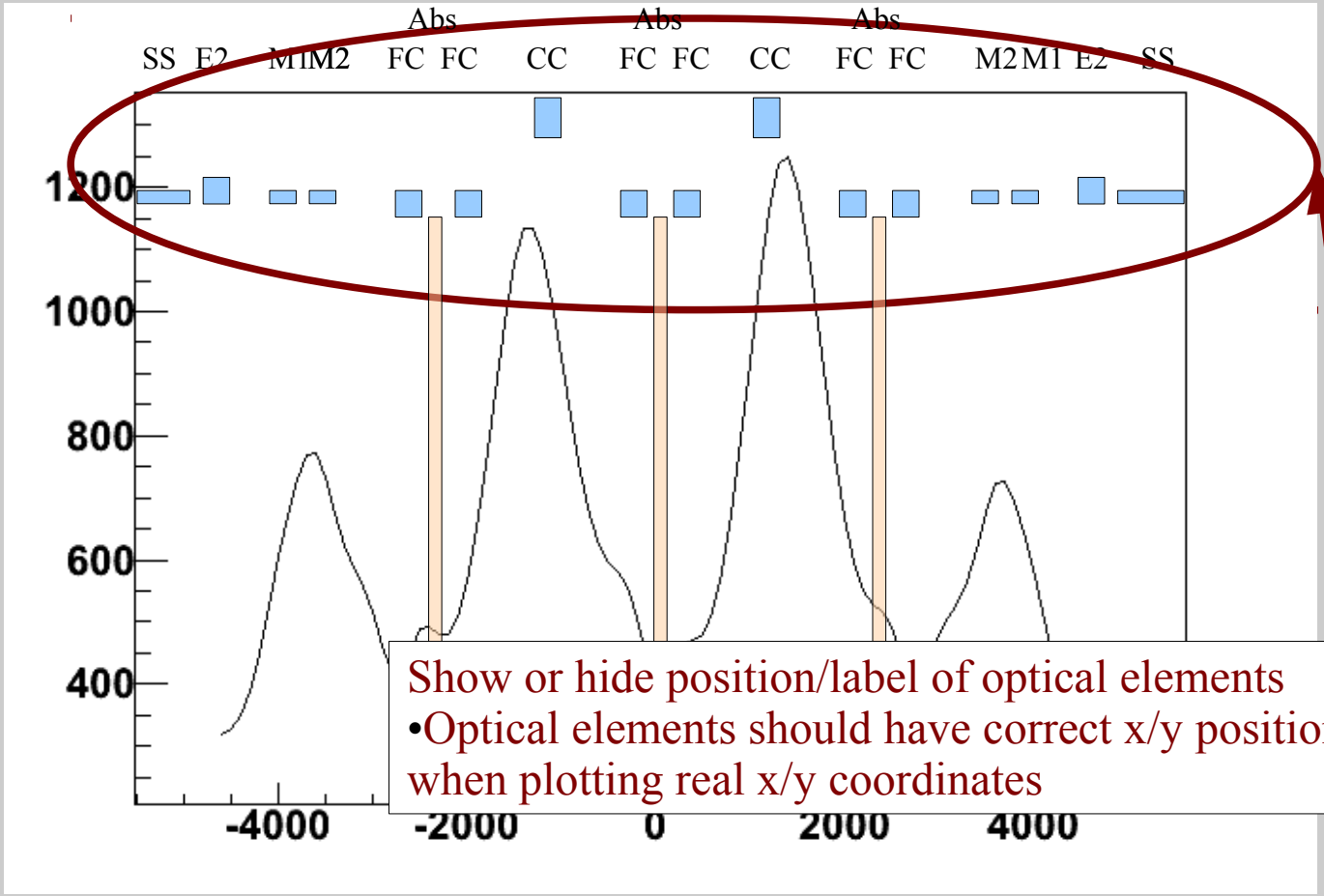
em. (mm) 6

CC (A) 109.12

Plot

ordinate beta_perp

Show optical elements



Show or hide position/label of optical elements
•Optical elements should have correct x/y position when plotting real x/y coordinates

Beam

Z (mm) -2750

definition penn

beta (mm) 420

alpha 0

em. (mm) 6

Beamline

E1 (A) 149.05

M1 (A) 149.05

SS (A) 222.01

M2 (A) 222.01

E2 (A) 109.12

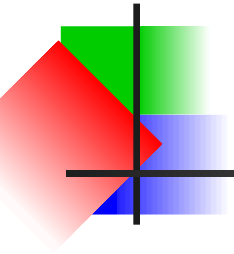
FC (A) 109.12

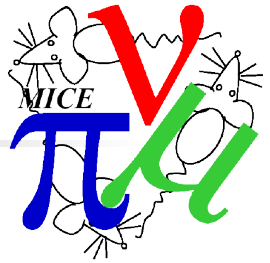
CC (A) 109.12

Plot

ordinate beta perp

Show optical elements





FIN