Efficiency framework

Giuseppe E Bruno
Università di Bari and INFN - Italy

Outline

- reminder
  - aim of plane efficiency measurement
- frame-work implementation
- ITS implementation
- conclusions

+ a couple of specific items I would like to discuss

Plane efficiencies: what's for ?

- from data to a paper
  - data taking → raw data
  - reconstruction → ESD → standard AOD
  - analysis:
    - standard AOD→ user AOD
    - selection of candidates
    - → computation of corrections for acceptance and reconstruction inefficiencies
    - application (e.g., with a deconvolution) of the corrections (and flux factors for normalisation) to the raw distributions to get the physical distributions
    - → estimate of the systematics errors (here corrections always play a role)
  - writing of the paper

Plane efficiencies: what's for ?

- In order to compute properly the corrections, the Monte Carlo description of our detectors should be as realistic as possible.

Can be used in few cases, and for systematic errors evaluation

- fast simulation (digits created according to tables of probabilities ← "plane efficiencies")

Advantages: fast, history (deterioration) of the detectors reproduced naturally
Drawbacks: poor description of the detector spatial precision, digit correlations ?, some analyses sensitive to the chosen granularity for eff. determination

Standard case

- response models driven by physics (e.g. in ITS, the Gaussian diffusion, plus coupling effects, etc.)

The measured plane efficiencies would be THE REFERENCES of the models

- The aim of the framework is to measure the efficiencies of the layers used for tracking, **with the tracks** themself
  - ITS is the goal
  - Eventually, the framework would be extended to other detectors (TRD, TOF, ...)
  - TPC would do it differently (not using tracks)
- Of course, each layer is sub-divided in basic blocks (e.g., the chip for SPD)
- A rough estimate of the "Plane efficiency" can be done also looking at **the map of hits** (i.e. the distribution of digits)
  - one of the QA/calibration measurements
  - will it give the relative "efficency" within a block ? not for uniformly distributed inefficiencies !

Implementation

- Plane efficiency measured using high quality tracks by removing one layer at the time from the tracker
- The output (efficiency tables) written into the DB for periods of validity
  - bonus from the framework:
    - residuals distributions (i.e. track prediction – cluster position)
      → alignement studies
      → accurate cluster precisions (for tracking)
    - cluster shape distributions:
      → tuning of detector-response simulation models
- Dead and noisy channels:
  - not a goal of this work
  - the framework aware of them (from calibration)

- The measured Eff. will be used as reference for the detector response simulation
  - as an overall factor, if the response model is 100% efficiency
  - as the reference to tune the response model, if not
- For special analyses (e.g. SPD multiplicity?) and (eventually) for systematic errors evaluation
  - fast simulation: the “chip” (module) efficiencies give the probabilities to have a cluster

Code Implementation of the framework

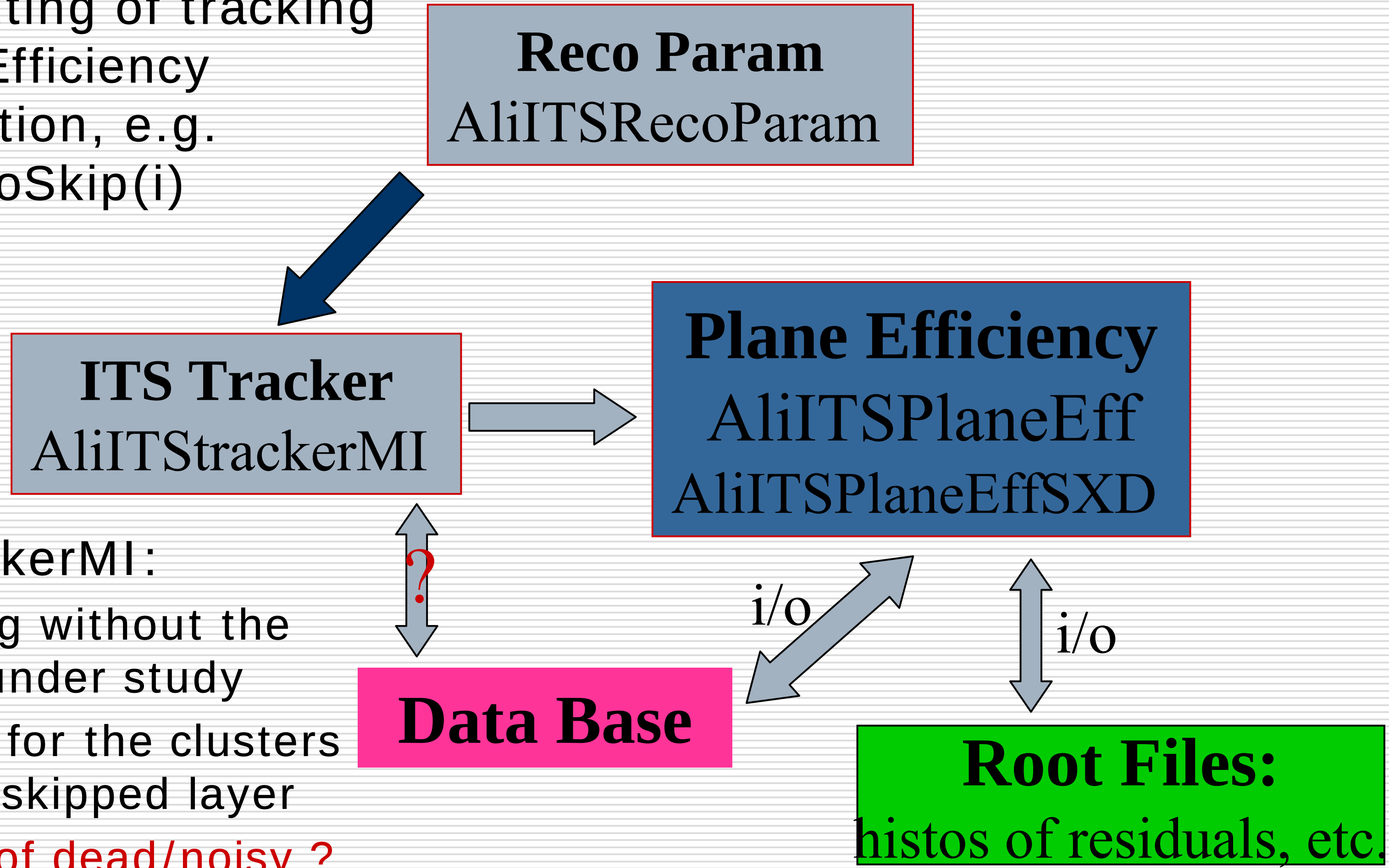
- STEER:
  - new virtual class: AliPlaneEff

    specific implementation for detectors (and subdetector) inherits from it, e.g. :
    - AliITSPlaneEff
      - AliITSPlaneEffSPD
      - AliITSPlaneEffSDD
      - AliITSPlaneEffSSD
  - new methods/data members in AliReconstruction

    to perform (on demand) plane efficiency evaluation
- DET: ITS, ... (TOF ,TRD)
  - AliDetTracker (e.g. AliITStrackerMI) upgraded to perform plane efficiency evaluation

Code implementation: e.g. ITS

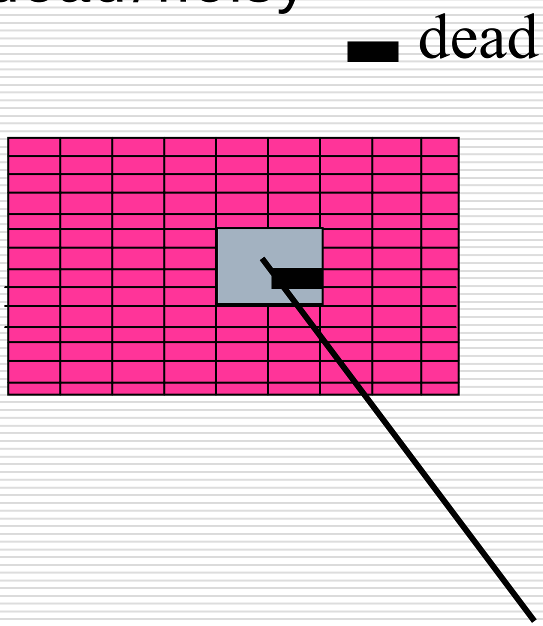
- AliITSRecoParam: steer special setting of tracking for Plane Efficiency determination, e.g. SetLayerToSkip(i)



- AliITStrackerMI:
  - tracking without the plane under study
  - search for the clusters on the skipped layer
  - aware of dead/noisy ?

Which plane efficiency?

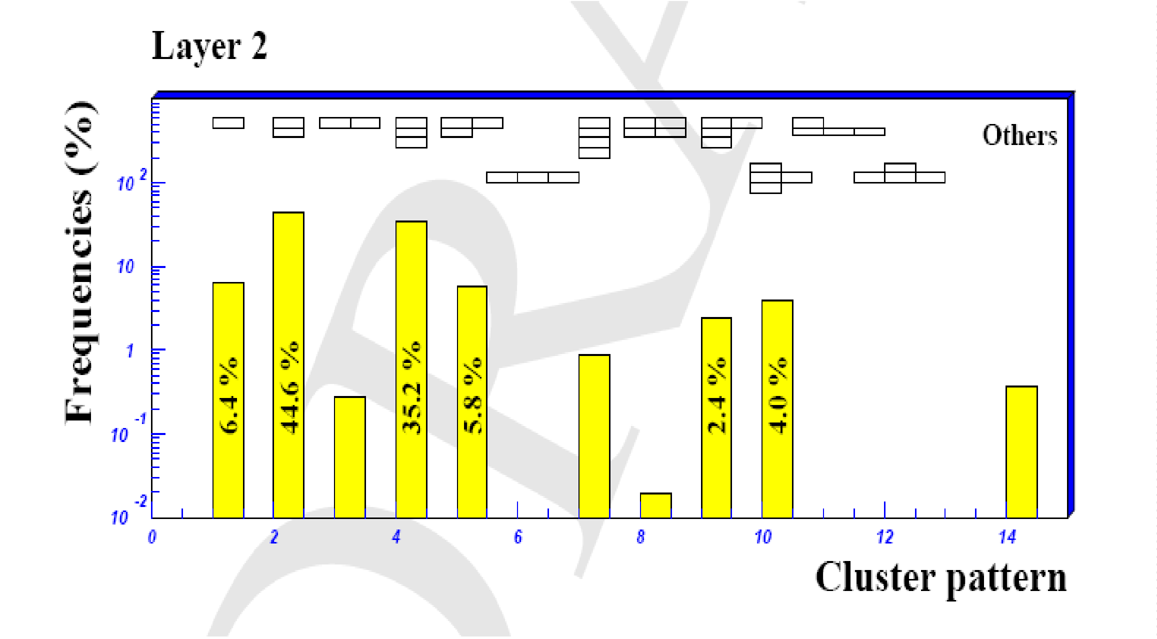
- Efficiency of the live detector ?
  - the tracker should be aware of dead/noisy channels
  - what to do if (some) noisy channels in the track road ?
  - weighted counting ?
- **Overall efficiency**
  - simple counting (binary) statistics with integers
- By the way:



efficiency with tracks ≠ 1 – fraction of bad channels (dead+noisy), even for a digital detector (SPD)

A digression:

- E.g.: an SPD chip on layer 2 with 5% of dead pixels (distribuited uniformly) can be << 1% inefficient since one track fires on average more than 1 pixel



SPD Plane Efficiency for multiplicity measurement

- Availble for day one:
  - raw multiplicity distribution
  - SPD dead/noisy pixel maps

  In principle: plane eff. with tracks after alignement, etc…
- Is this enough to compute corrections?

  (may be yes if they are small, I had no time to read the new note by Jan-Fiete)

SPD segmentation

D. Elia

- Plane efficiency to be avaluated chip by chip
- Number of required high quality tracks by assuming Eff=99.0 ± 0.5 %

  and uniform track fluency:
  - layer 1:

    400 chips → 162K
  - layer 2:

    800 chips → 323K

$$Eff = \frac{N_{\text{RecPoints}}}{N_{\text{tracks}}}$$

$$\frac{\sigma_{Eff}}{Eff} = \frac{1}{\sqrt{N_{\text{tracks}}}}\sqrt{\frac{1-Eff}{Eff}}$$

$$N_{\text{tracks}} = \frac{1}{\left(\sigma_{Eff}/Eff\right)^2}\frac{1-Eff}{Eff}$$

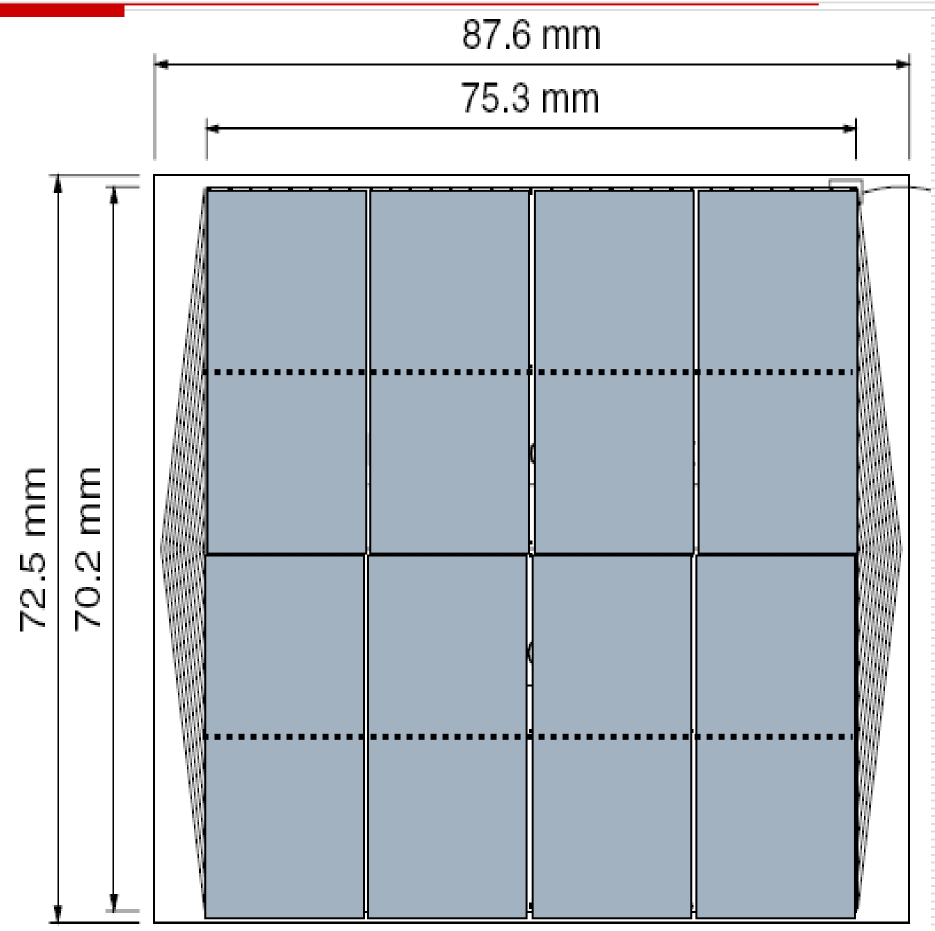
SDD segmentation

Eff=99%

$$\sigma_{eff} = 0.5\%\ eff$$

F. Prino

- layer 3:
  - 14 ladders
  - 1 ladder=6 detector
  - tot. 84 detector
- layer 4:
  - 22 ladders
  - 1 ladder=8 detector
  - tot. 176 detector
- each detector divided in 8(chips) times, eventually, 2 (drift direction)
  - layer 3: 672 zones → 271K tracks
  - layer 4: 1408 zones → 567K tracks



eventually *2

SSD segmentation

- layer 5: 34(ladders)*22=748 modules
- layer 6: 38(ladders)*25=950 modules
- 1 module=6(p-side)+6(n-side)=12 chips
- There is not simple segmentation if we want to go finer than the module
  - chip by chip ?
    - too many elements (layer 6: 11400) !
    - stereo geometry: how to share the inefficiency between n- and p–sides ?
- only possible choice: module by module

Implemented ITS segmentation

$\sigma_{eff} = 0.5\% \, eff$

	Eff=90%	99%
	n. of tracks	
pixel: chip by chip		
layer 1: 400 zones →	1.8M	160K
layer 2: 800 zones →	3.5M	323K
drift: chip by chip (*2)		
layer 3: 672 zones	3.0M	271K
layer 4: 1408 zones	6.2M	567K
strip: module by module		
layer 5: 748 zones	3.3M	302K
layer 6: 950 zones	4.2M	383K

The code is flexible enough to easily reduce/enlarge segmentation

Code implementation

- Just one class:
  - as container for the efficiencies
  - for reading/writing from/to OCDataBase
  - for updating/managing the efficiencies
  - for filling histos of residuals (this week dev.)

- AliITSPlaneEff ::AliPlaneEff
  - virtual base class for ITS Plane Efficiency
- AliITSPlaneEffSXD
  - specific classes for subdetectors

Code implementation

- The class as container:

```
Int_t fRunNumber;    //! run number (to access CDB)
TString fCDBUri;     //! Uri of the default CDB storage
Bool_t fInitCDBCalled;  //! flag to check if CDB storages
                        // are already initialized
```

```
Int_t fFound[kNModule*kNChip];  // n. of associated
                                //clusters in a given chip
Int_t fTried[kNModule*kNChip];  // n. of tracks used for
                                // chip efficiency evaluation
```

↑

Data members to be streamed from/to Data Base,
specific of the derived class (only for dimension: e.g. SPD 1200)

Code implementation

- Reading and writing into the DataBase:

```
virtual Bool_t WriteIntoCDB() const;
virtual Bool_t ReadFromCDB(); // this method reads Data Members
                              // (statistics) from DataBase
virtual Bool_t AddFromCDB() // this method updates Data Members
                            // (statistics) from DataBase
void SetDefaultStorage(const char* uri);
void InitCDB();         // activate a default CDB storage
                        // First check if we have any CDB storage set
```

Differently from other ITS Calibration/Response objects, this class refers to the full subdetector, and it is written as a whole to the OCDB. Main motivations: (i) when analyzing a bunch of events, all the chips/modules of a layer can be involved; (ii) very small-size objects

Code implementation

- other functionalities related to DataBase:

```
// Compute the fraction of  live/bad (i.e. dead and noisy) area
(of the CHIP for the SPD)
virtual Double_t GetFracLive(const UInt_t key) const;
virtual Double_t GetFracBad(const UInt_t key) const;

// Plane efficiency for active detector (excluding dead/noisy channels)
virtual Double_t LivePlaneEff(UInt_t) const
```

etc ...

Code implementation

- main methods for updating/managing the efficiencies, i.e. **fFound**[] and **fTried**[]

```
Double_t PlaneEff(Int_t nfound,Int_t ntried) const;
Double_t ErrPlaneEff(Int_t nfound,Int_t ntried) const;
```

$$Eff = \frac{fFound}{fTried}$$

$$\sigma_{Eff} = \frac{1}{fTried}\sqrt{fFound\frac{1-fFound}{fTried}}$$

```
// Estimate of the number of tracks needed
// for measuring efficiency within RelErr
virtual Int_t GetNTracksForGivenEff(Double_t eff, Double_t RelErr) cons
```

$$N_{\text{tracks}} = \frac{1}{\left(\sigma_{Eff}/Eff\right)^2}\frac{1-Eff}{Eff}$$

```
// Update the statistics of a given module
virtual Bool_t UpDatePlaneEff(const Bool_t b, const UInt_t k)
```

```
fTried[k]++;  if(b) fFound[k]++;
```

Code implementation

- main methods for updating/managing the efficiencies, i.e. **fFound**[] and **fTried**[]

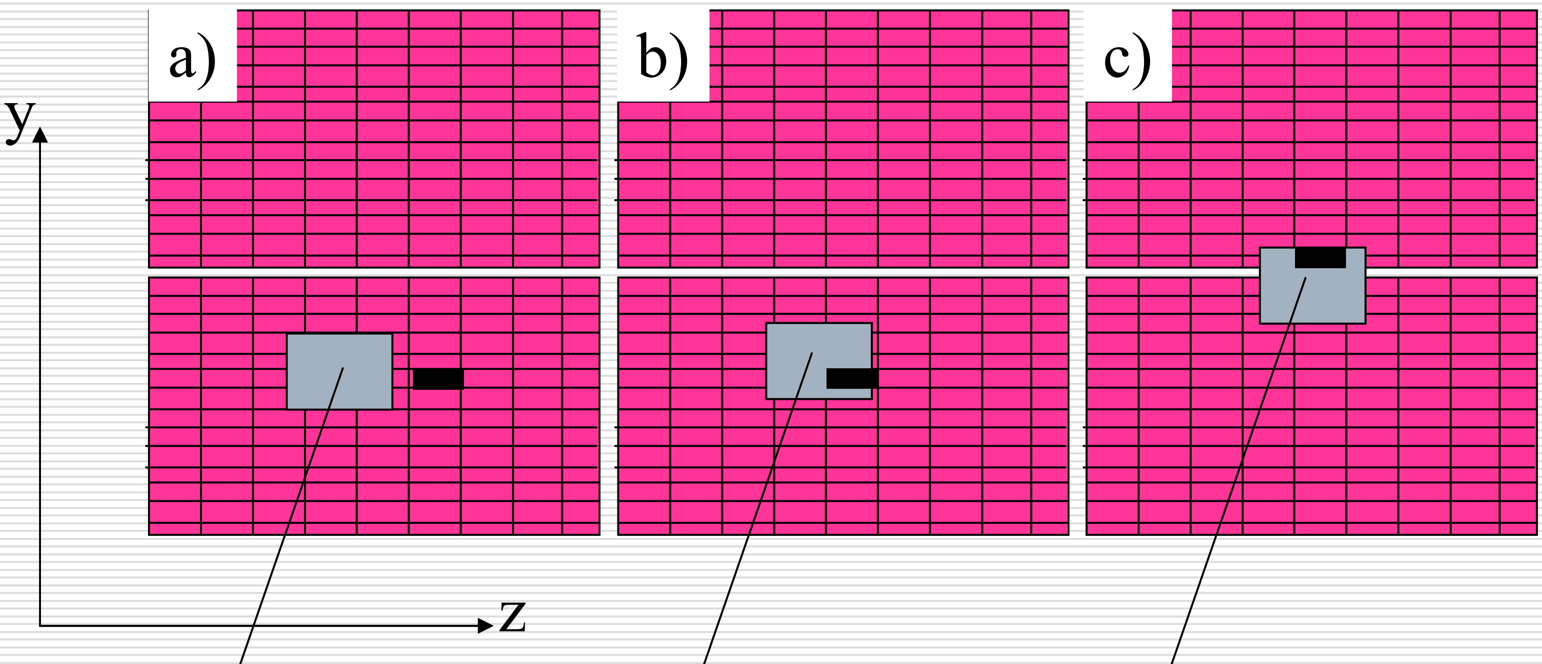
```
// ass. operator
AliITSPlaneEffSPD& operator=(const AliITSPlaneEffSPD &s);
virtual AliITSPlaneEff& operator=(const AliITSPlaneEff &source);


// Simple way to add another class (i.e. statistics).
AliITSPlaneEffSPD& operator +=( const AliITSPlaneEffSPD &add);
```

(semi-) Open point: Strategy

area: $N_y \sigma_y^{track} \times N_z \sigma_z^{track}$

"Search road" for associating a cluster can be larger/smaller



- usable track
- **fTried**++

- usable track

fTried++

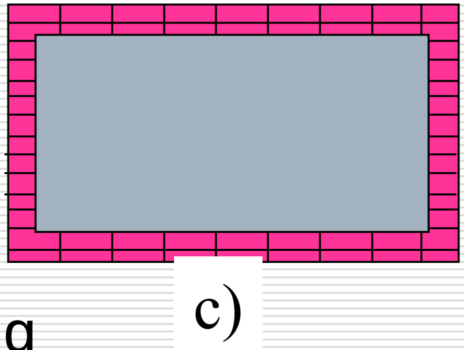
fFound++

- reject track

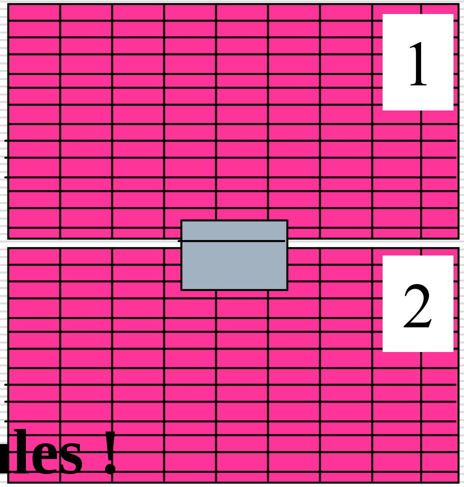
Case c) can be treated differently → next slide

(semi-) Open point: Strategy

- With the method indicated in previous slide, the border of the basic block (chip for SPD) would not weight in the total block efficiency
  - this is the actual implementation
  - same as CMS (but sharp cuts)



- Another possible method ?
  - use each track, assigning the counting of efficiency/inefficiency to one of the blocks with a probability proportional to the overlaps: 2→80%
    1→20%



I'm in favour of the 1st method: **spacing among modules !**

Conclusions

- The framework will provide the ITS overall plane efficiency, not the efficiency of the live detector.
  - functionality to get the fraction of bad/noisy channels implemented

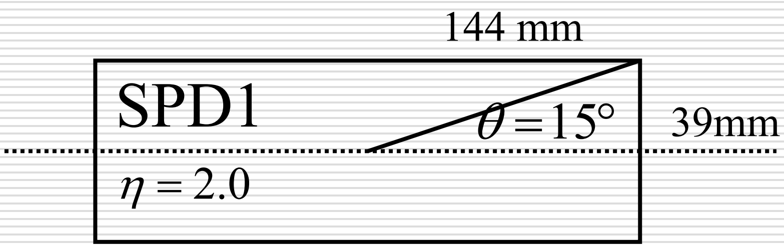
    (from DB Calibration files)
- The framework has been thought in such a way to be extensible to other detectors (e.g. TRD, TOF)

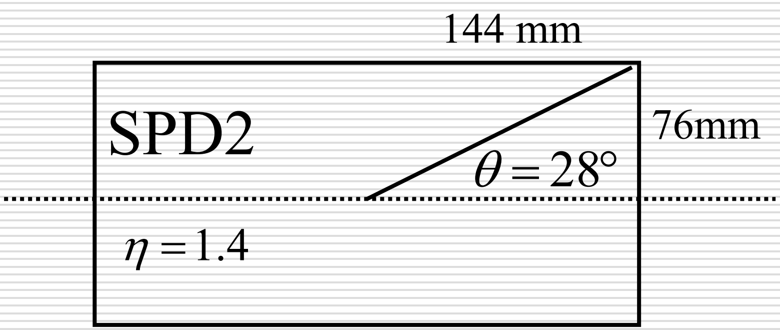
SPD: relative track occupancy

$$\frac{dN}{d\eta} = C \Rightarrow \frac{dN}{d\theta} = -\frac{C}{\sin\theta} \Rightarrow \frac{dN}{dz} = \frac{C}{r}\sin\theta$$

Assuming Vertex_z=0

$$\eta = -\ln\tan\frac{\theta}{2}\ , d\eta = -\frac{d\theta}{\sin\theta},\quad dz = -r\frac{d\theta}{\sin^2\theta}$$



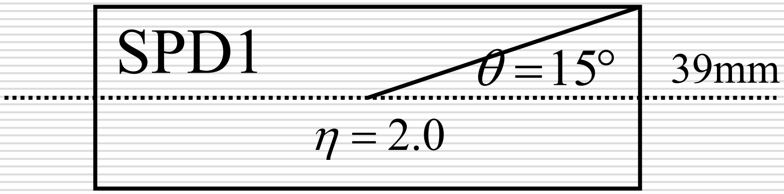


At the border:

SPD1 $$\frac{dN}{dz}(\theta = 15°) = \frac{dN}{dz}(\theta = 0°)\sin 15° = 0.26\frac{dN}{dz}(\theta = 0°)$$

SPD2 $$\frac{dN}{dz}(\theta = 28°) = \frac{dN}{dz}(\theta = 0°)\sin 28° = 0.46\frac{dN}{dz}(\theta = 0°)$$

SPD regions uncovered by tracking



$$dz = \frac{r}{\sin\theta} d\eta = r\frac{1+e^{-2\eta}}{2e^{-\eta}} d\eta$$

$$dz = r\cosh\eta\, d\eta$$

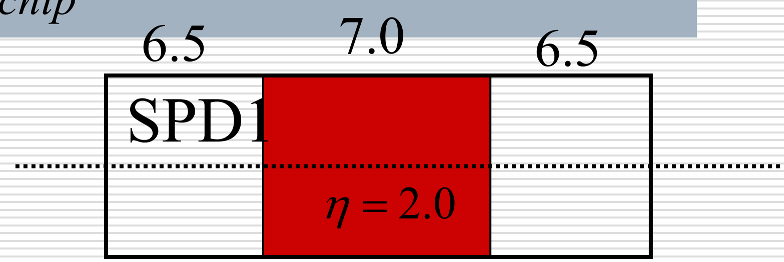
$$\int_{z_{\min}}^{z_{\max}} dz = \int_{\eta_1}^{\eta_2} r\cosh\eta\, d\eta \Rightarrow \Delta z = r(\sinh\eta_2 - \sinh\eta_1)$$

$$\Delta z_{chip} = 1.41cm$$

$$\frac{\Delta z}{\Delta z_{chip}} = \frac{1}{\Delta z_{chip}} r(\sinh\eta_2 - \sinh\eta_1)$$

SPD1: r=3.9cm, $\eta_1 = 1$, $\eta_2 = 2$

$$\frac{\Delta z}{\Delta z_{chip}} = 6.7$$



SPD2: r=7.6cm, $\eta_1 = 1$, $\eta_2 = 1.4$

$$\frac{\Delta z}{\Delta z_{chip}} = 3.9$$