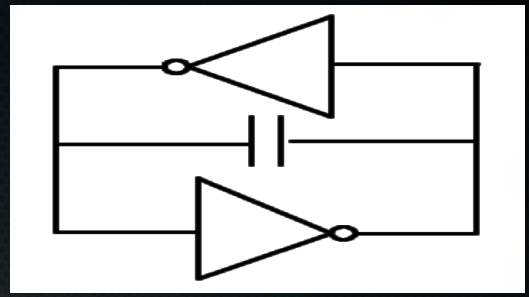
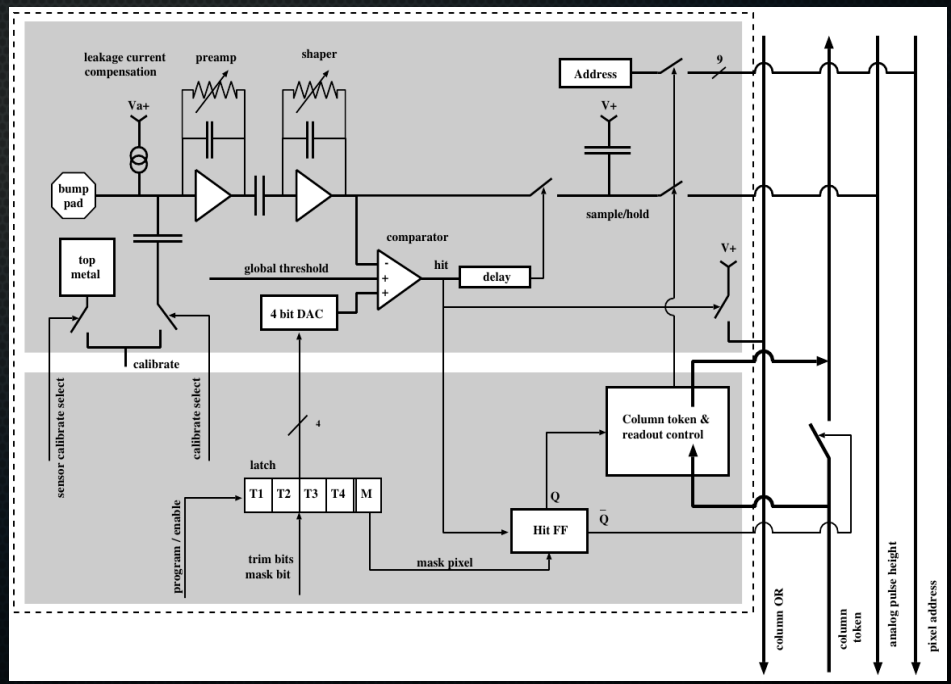
Automatic Detection of Single Event Upsets
In the Read Out Chips of the CMS Pixel Detector

Luigi Calligaris (DESY)

Effects of Single Event Upsets

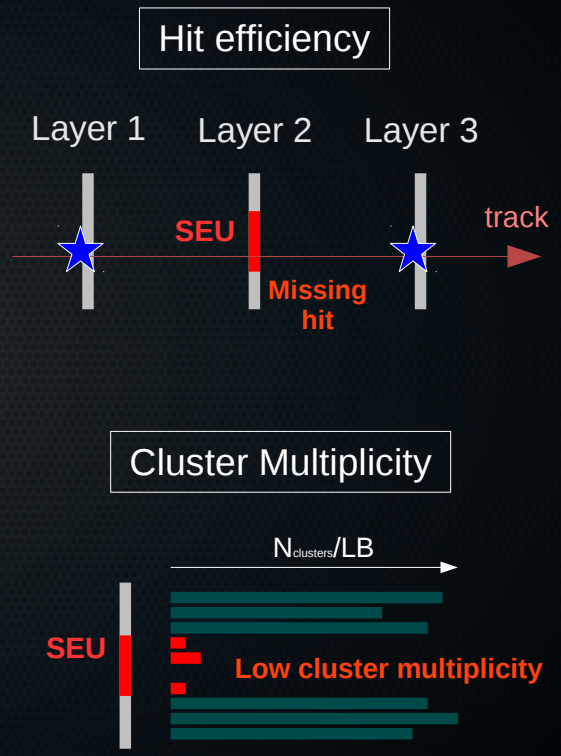
- Pixel Detector: high rate of ionizing particles
- A bit can be flipped (SEU) by injected charge
- ROCs design protects them against SEUs
- Still, sometimes a SEU will happen in a critical bit register
  - Unexpected behaviour
  - Often the chip goes totally/partially silent
- Not possible to read ROC state (design choice)
  - Need for data-driven monitoring





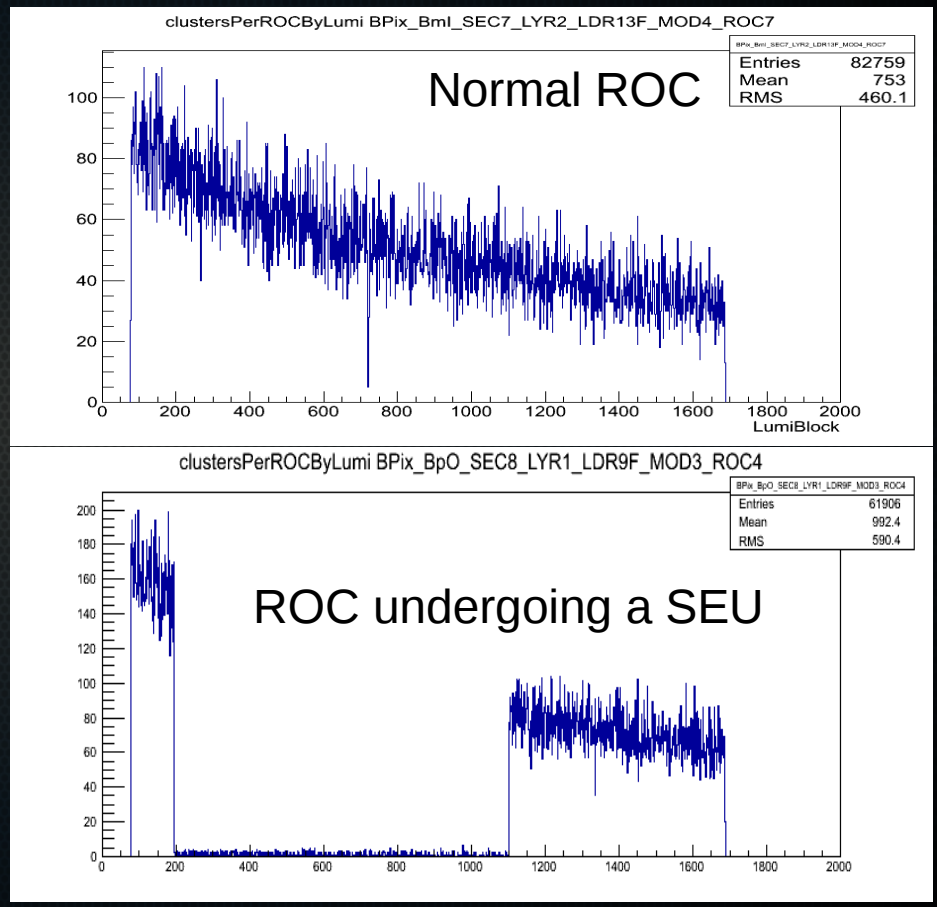
Indicators for a silent ROC

- Hit efficiency for tracks, multiplicity of digis (pixels above threshold), multiplicity of clusters
- Advantages of using clusters
  - Many clusters compared to tracks
  - Do not depend on tracking
  - Computationally cheap compared to tracks
- Aim: determine bad ROCs with a fine time granularity
  - Good data produced by them is not wasted
  - Useful to understand the phenomenon
- Observe: cluster occupancy for each ROC in each LumiBlock
  - Flag LumiBlocks where a ROC was bad



Bare cluster occupancy

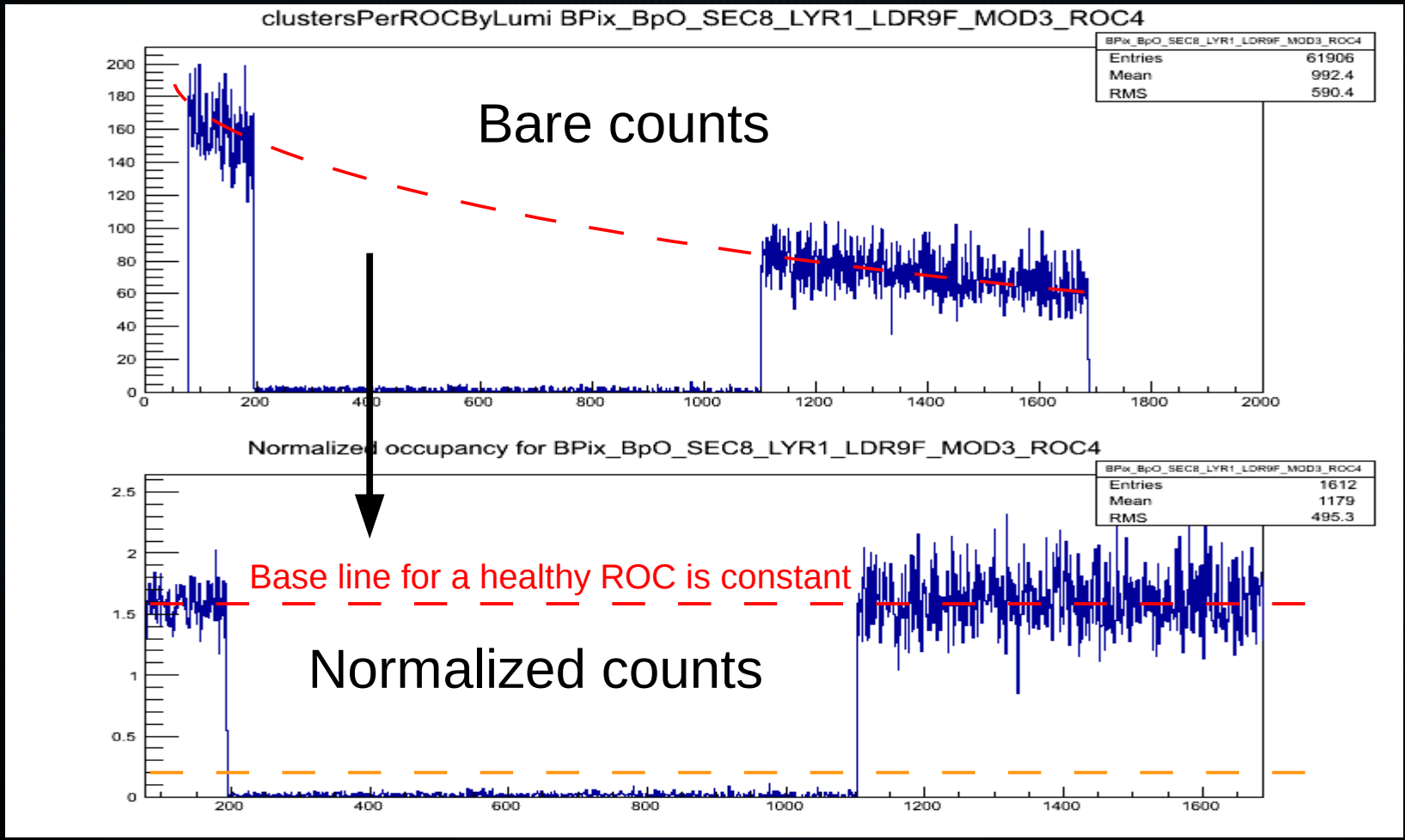
- Count of clusters in that ROC for each LumiBlock (= 23 s)
- Values depend on instantaneous Luminosity
  - Bad for setting a fixed threshold
  - Time dependence
- Need to evolve to a variable that factorizes out
  - Instantaneous luminosity
  - Pile Up



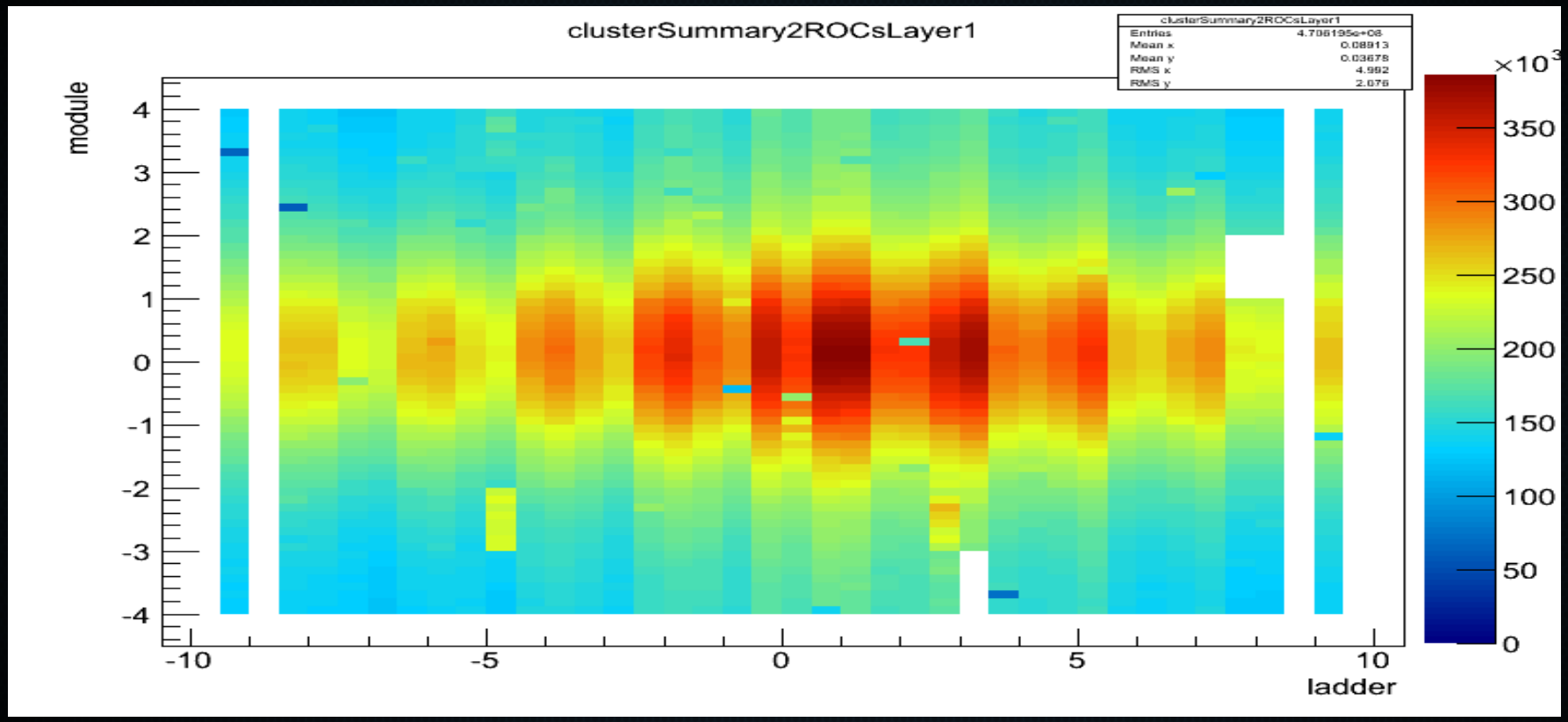
Normalized cluster occupancy

- $O_{normalized}$ = ($N_{clusters}$ in a ROC / $N_{clusters}$ in all ROCs) * N_{ROCs}
- Factorizes out the luminosity dependence
- Differences in θ and layer expected
  - Acceptance changes
  - Particle flux changes
  - Beam spot not centered w.r.t. Barrel pixel
- Solution: calculate normalized occupancy in regions
  - Dividing detector along Z
  - Dividing detector in octants

Normalized cluster occupancy

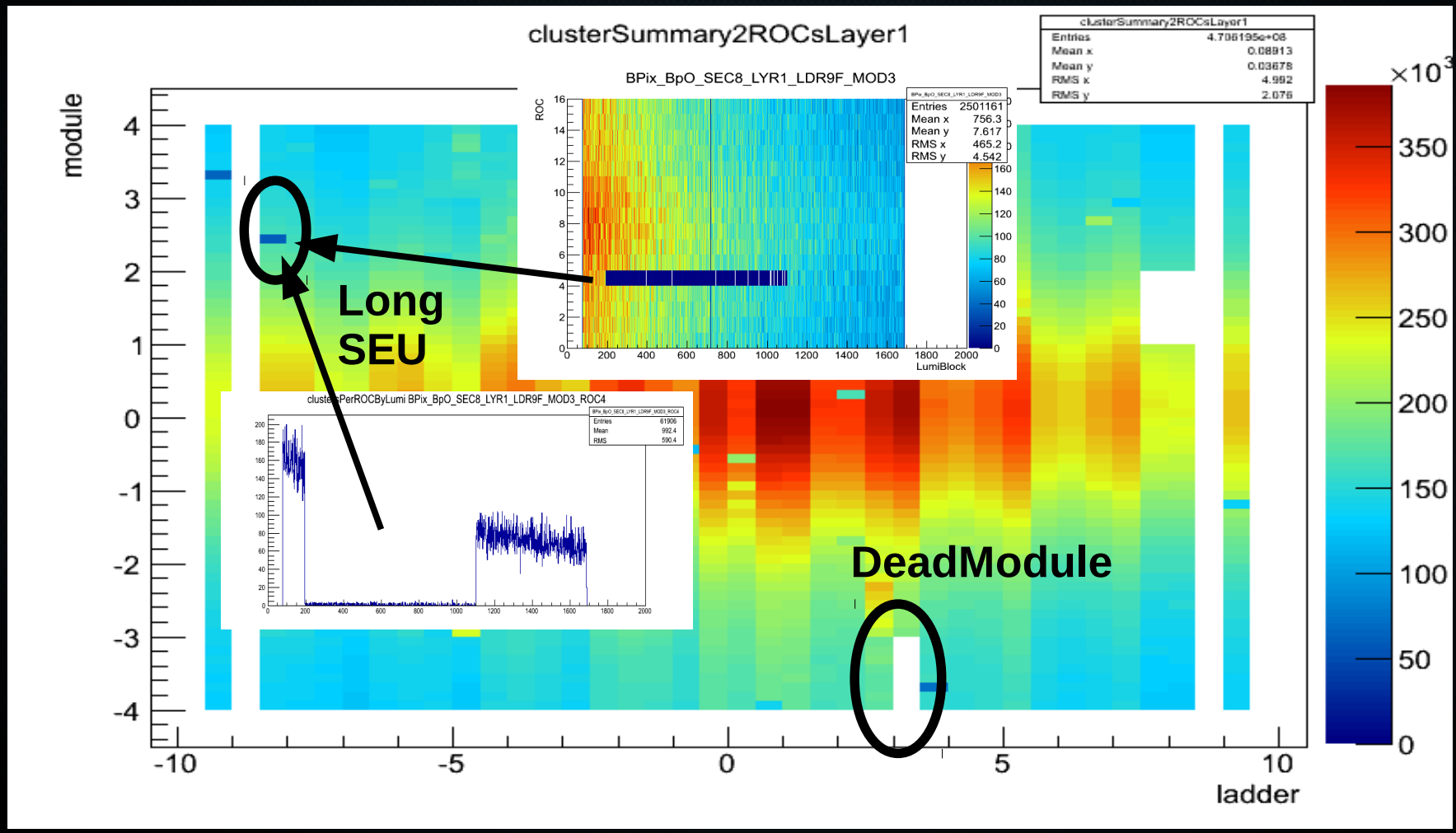


Bare occupancy map (LYR1) Run 191226

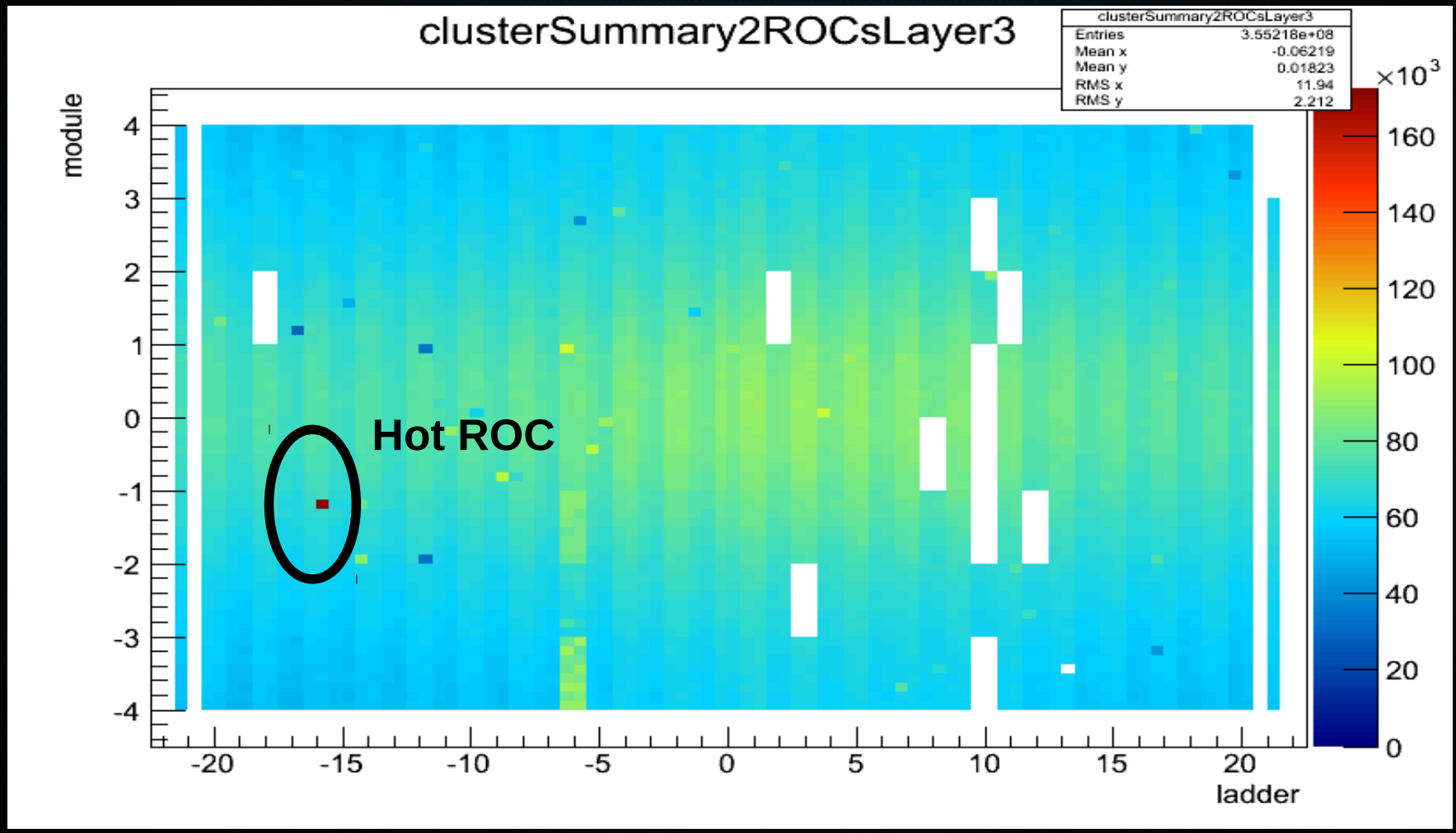


This summary map shows the cluster occupancy for each ROC, integrated over the whole run

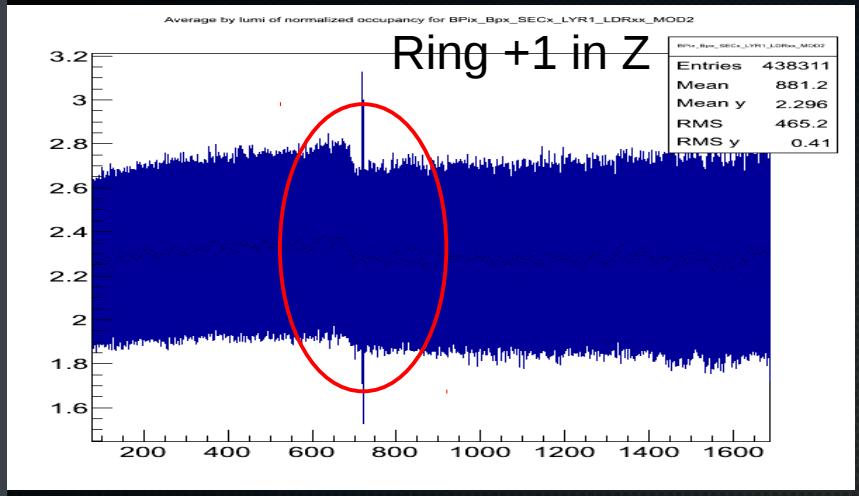
Bare occupancy map (LYR1) Run 191226

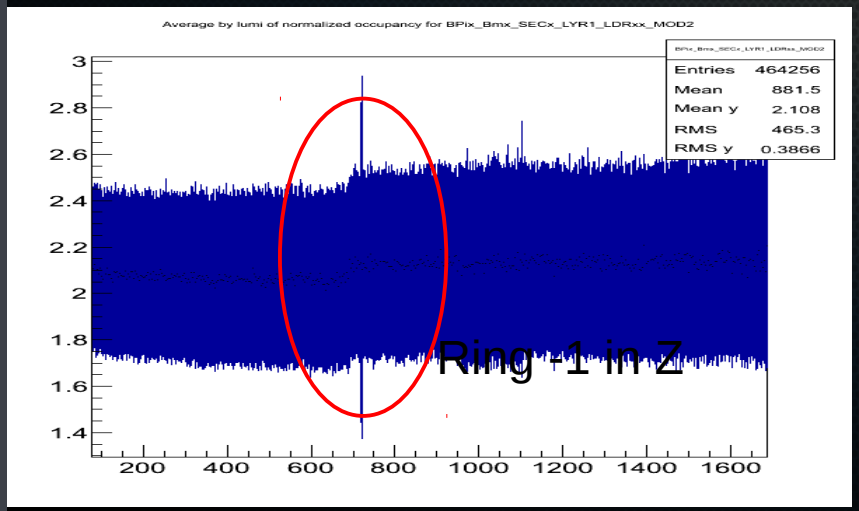


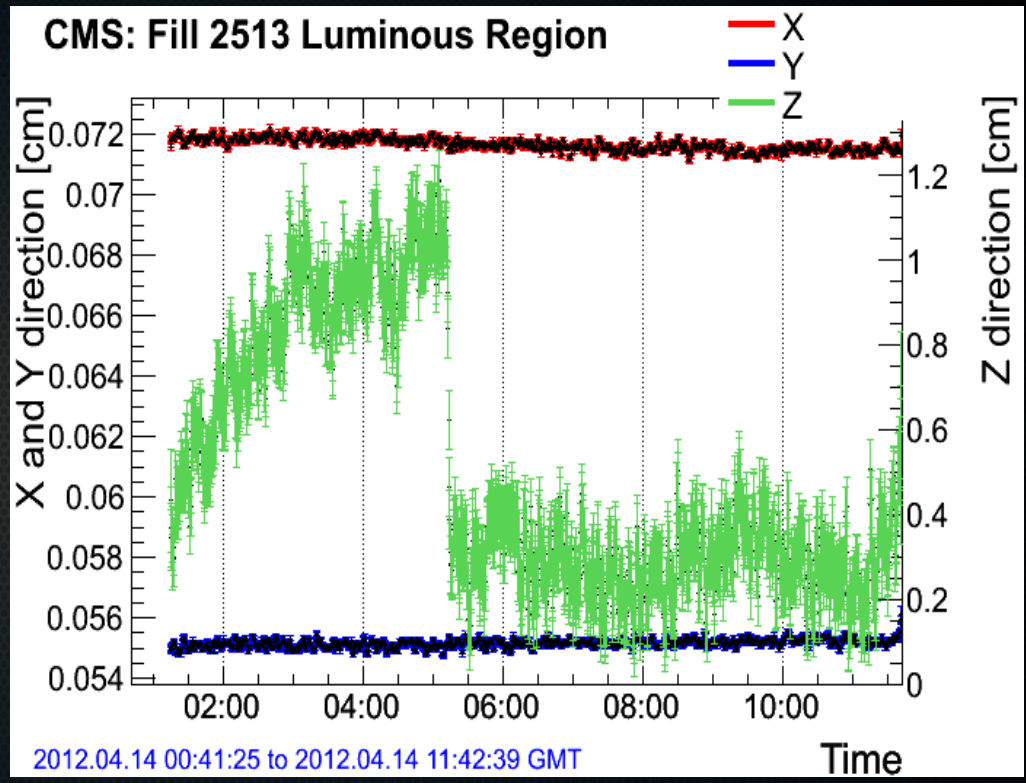
Bare occupancy map (LYR3) Run 191226



Cluster occupancy is quite sensitive...





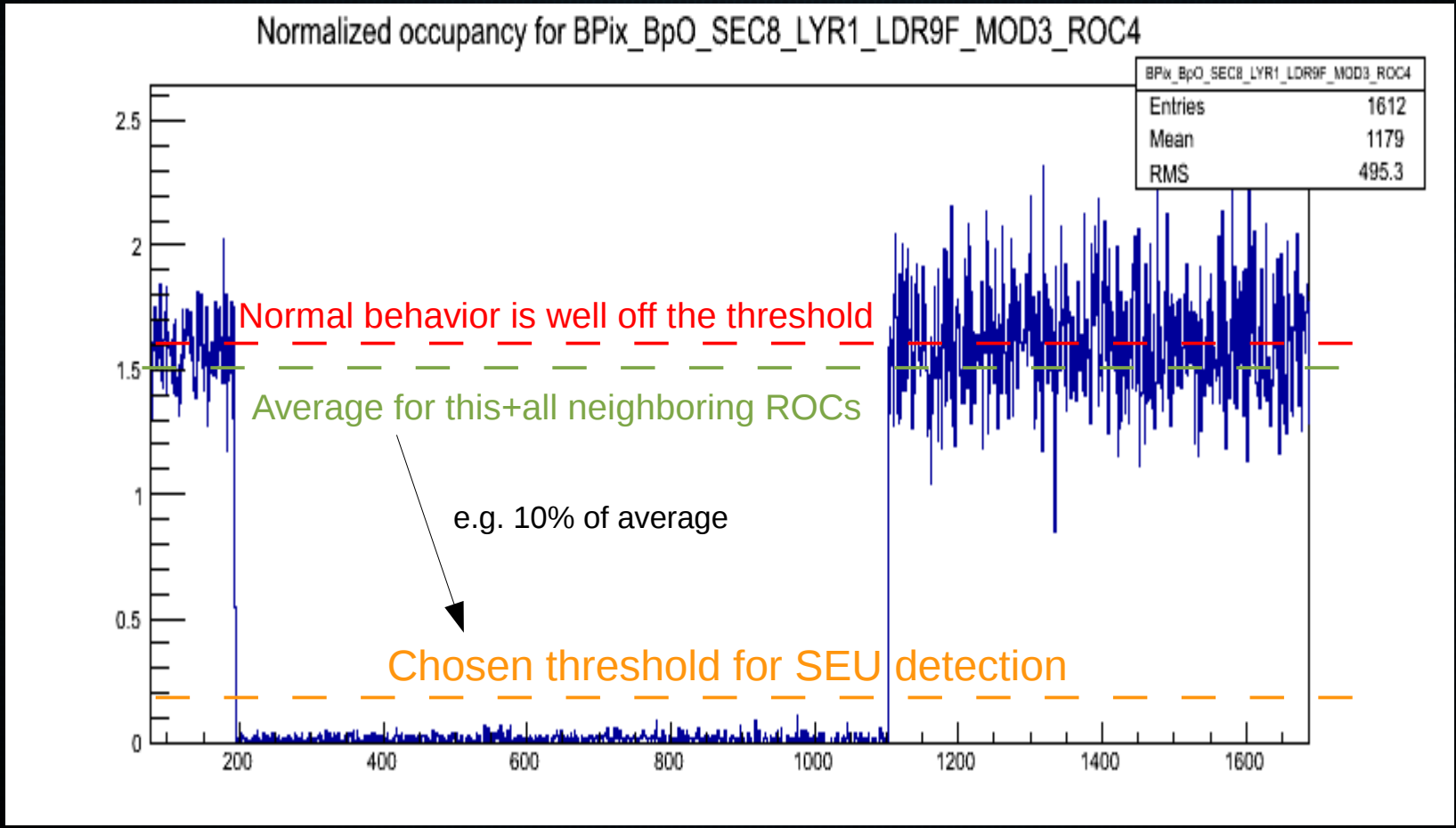


Sees the beam spot
Moving in z by 110 μm

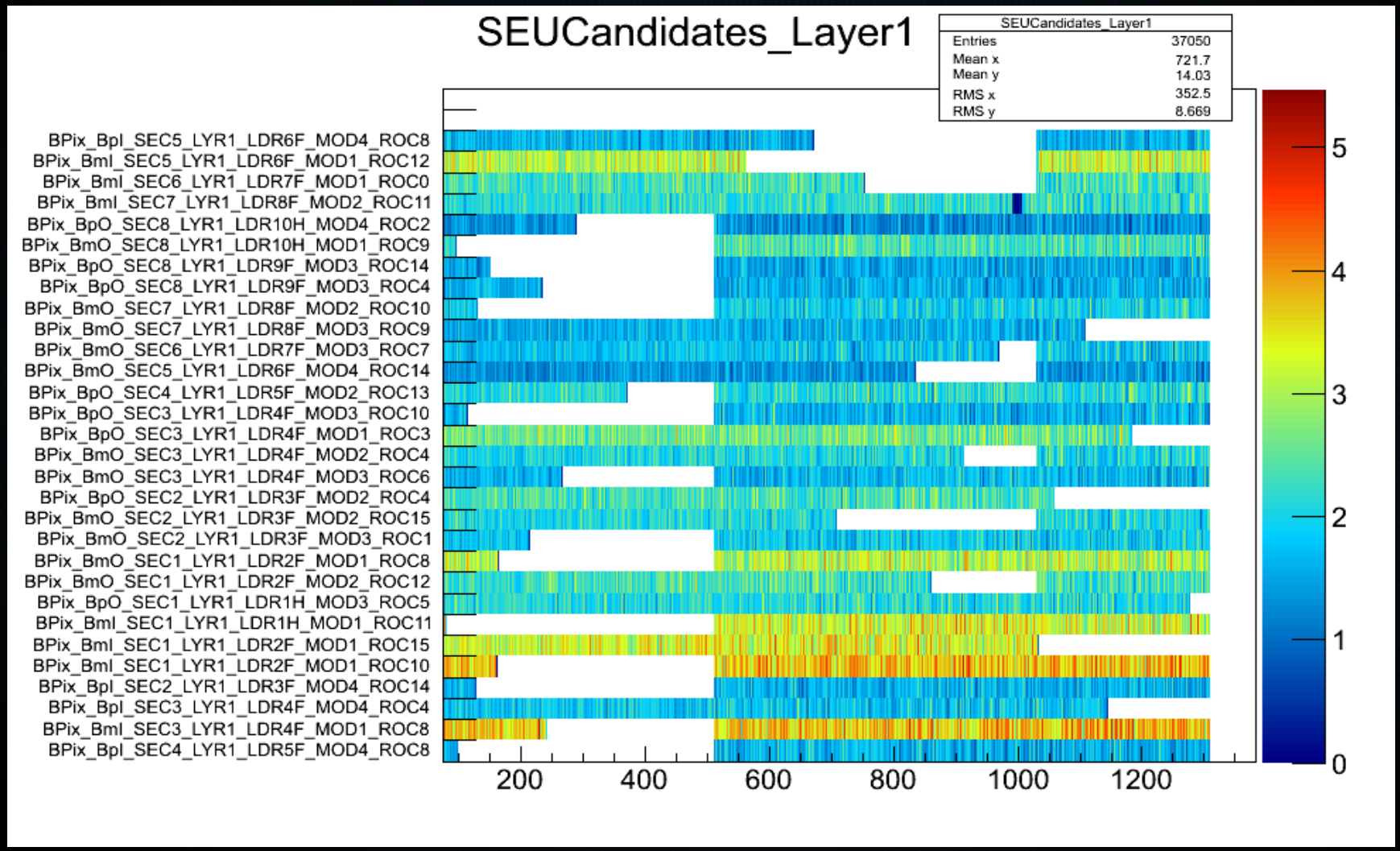
Algorithm to find SEU

- For each LumiBlock
  - Compute the average of the normalized occupancy
  - Choose a fraction of this value (e.g. 10%) as threshold
  - Compare with the normalized occupancy of each ROC
- If the ROC is flat zero, put it in the list of dead ROCs
- If the ROC stays under the threshold for more than N (e.g. 6) consecutive LumiBlocks
  - Put in the list of SEU
  - Plot its evolution as a function of LumiBlock

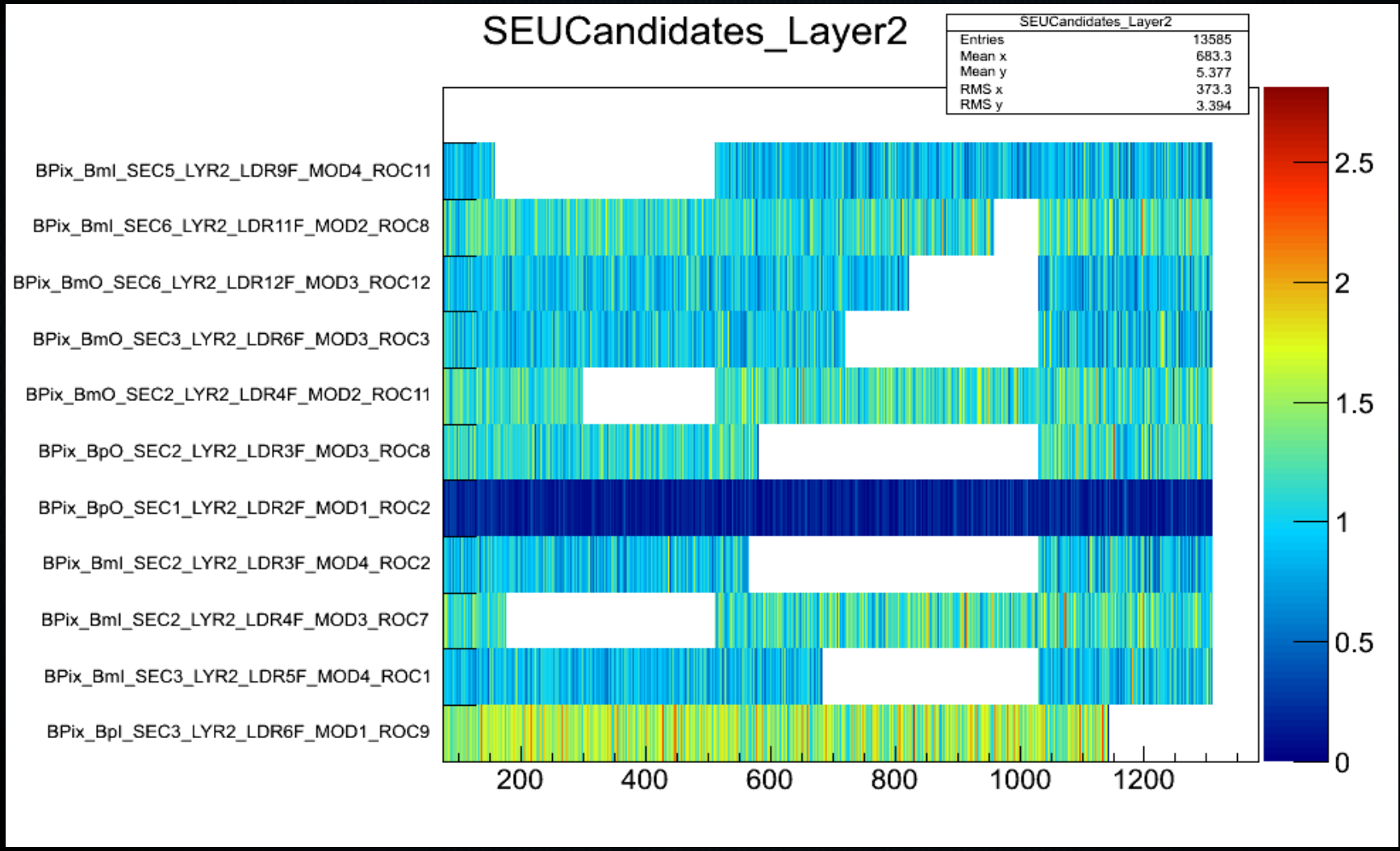
SEU seen from normalized occupancy



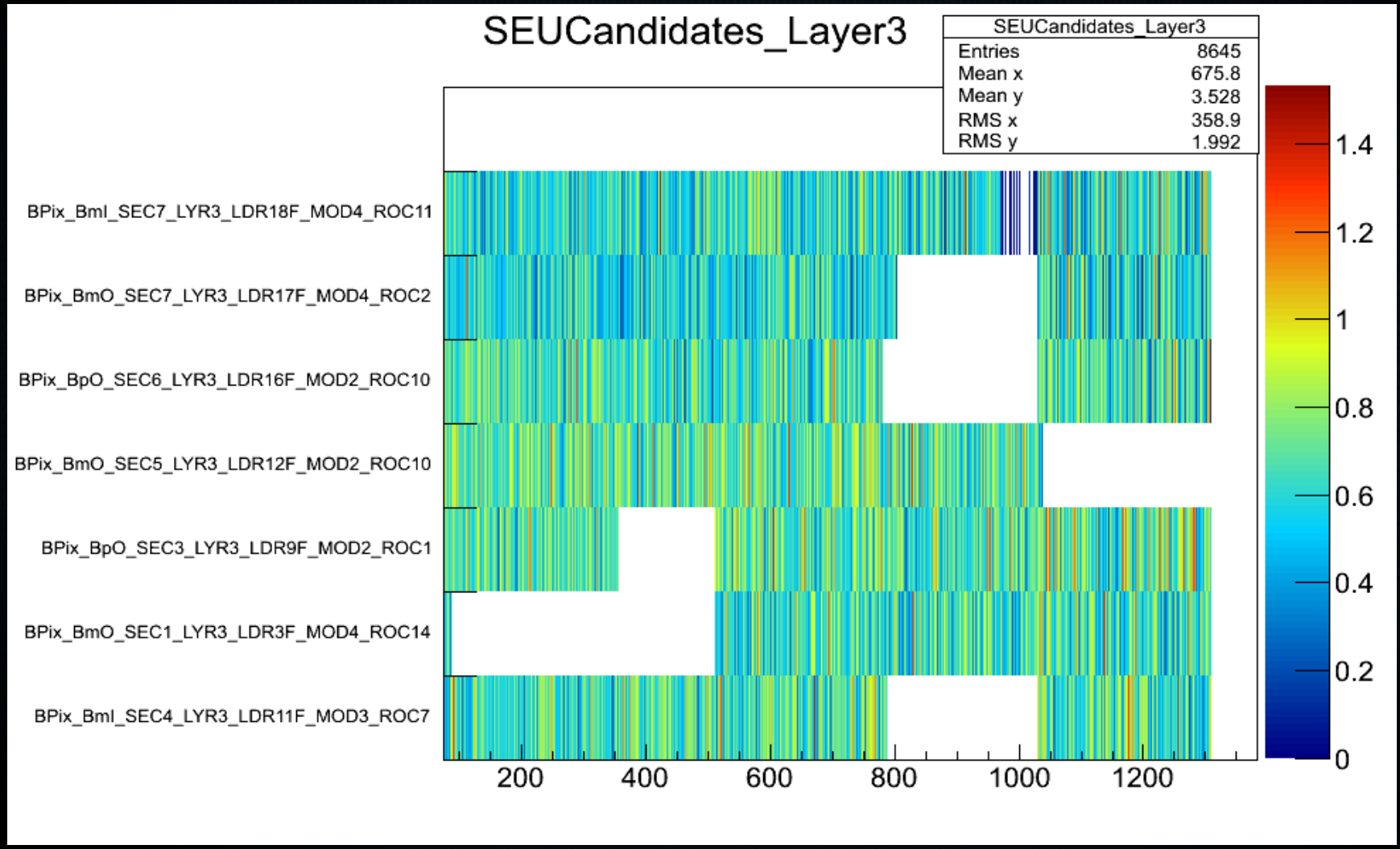
Run 201191 – Layer1



Run 201191 – Layer2



Run 201191 – Layer3



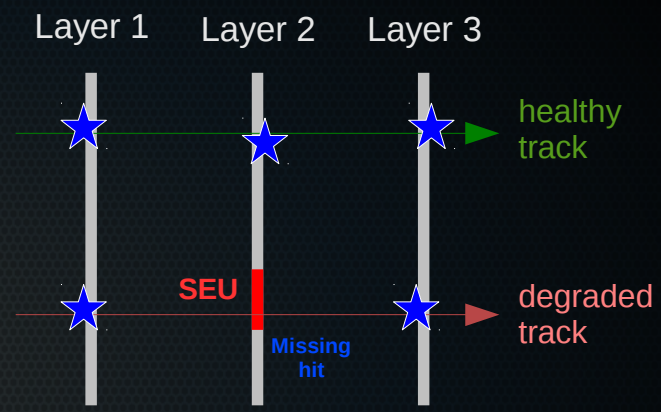
Summary & Outlook

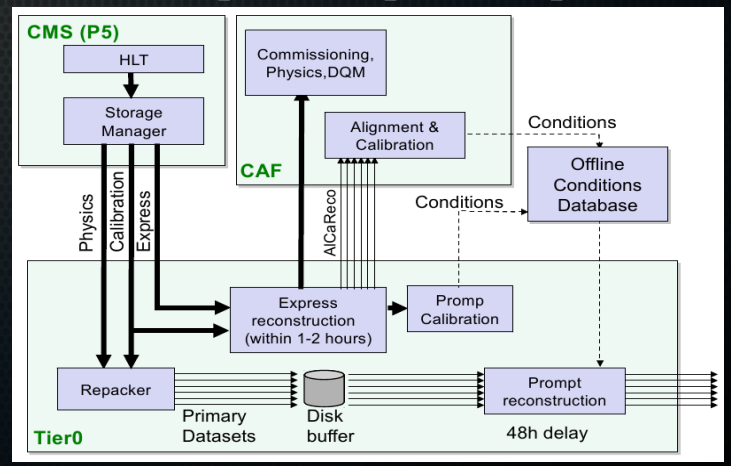
- Tested a proof-of-concept setup to find SEUs
  - SEUs are identified by the algorithm
- Next:
  - Re-write the code as a bundle of CMSSW modules (ongoing), aimed at running on ExpressStream
- Goal:
  - Identify automatically SEUs in the Pixel detector
  - Provide a list of dead ROCs and modules, to ease their book-keeping

BACKUP

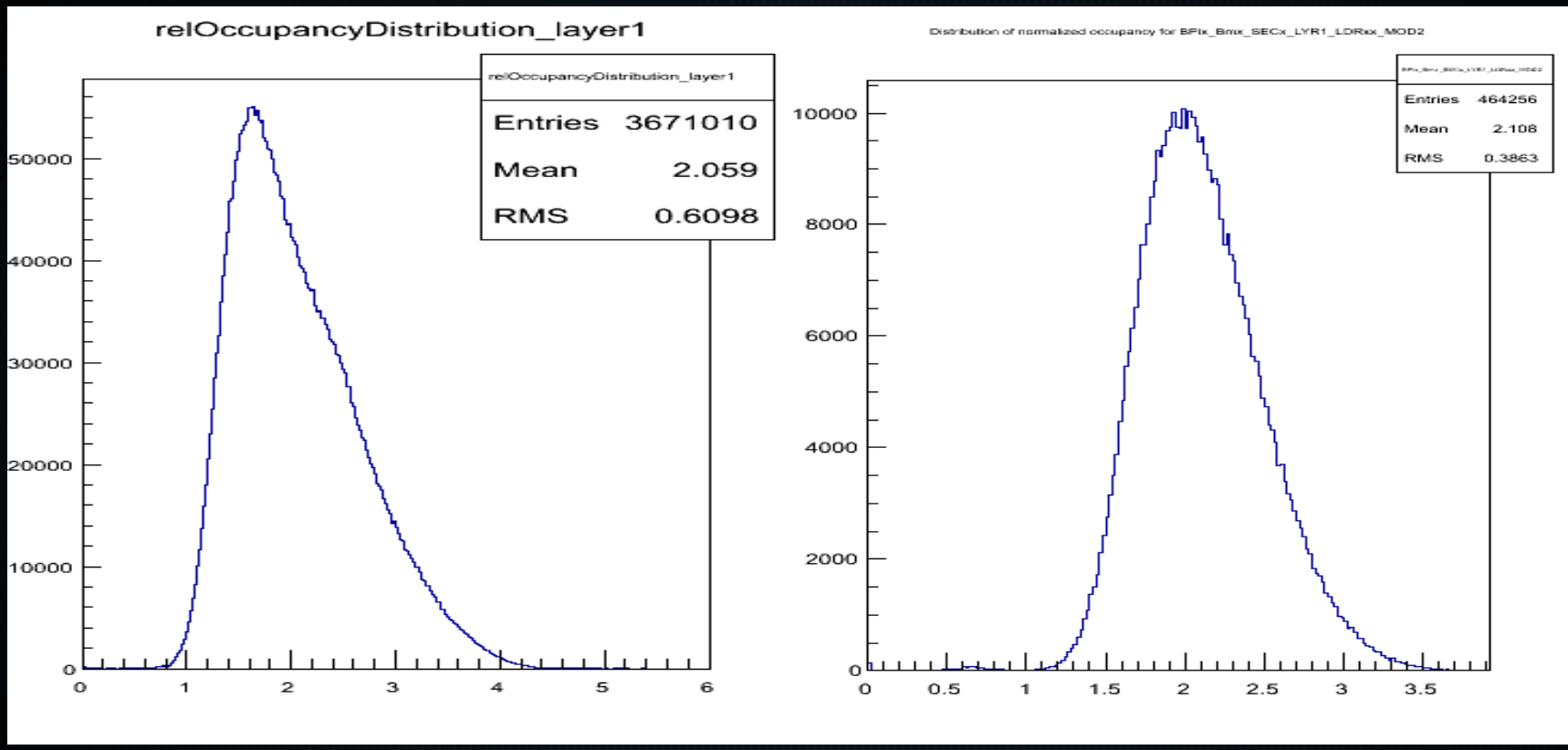
Tracking benefits from SEU detection

- Number of missing hits is a figure of merit for a track
  - A silent ROC pollutes good tracks with missing hits
- PromptReconstruction begins 48h after the end of run
- ExpressStream is available before for calibration, DQM, …
- → use this data to find silent ROCs, and upload to DB
  - PromptReco: improved tracking





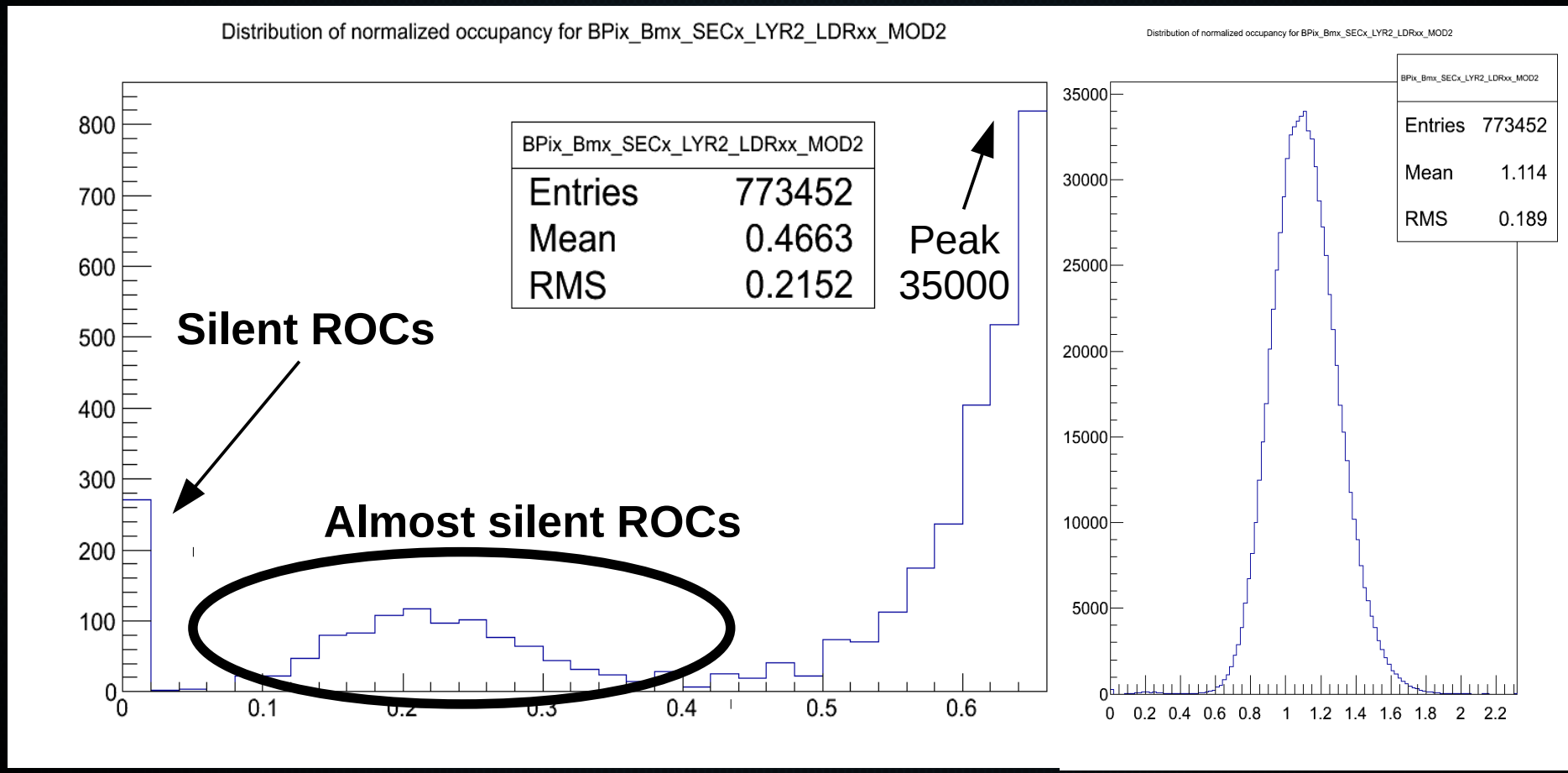
Distribution of normalized occupancy



All Layer 1
Integrated over run

Bmx_SECx_LYR1_LDRxx_MOD2
Integrated over run
Specific layer and ring in z

Low tail of the distribution for a single ring

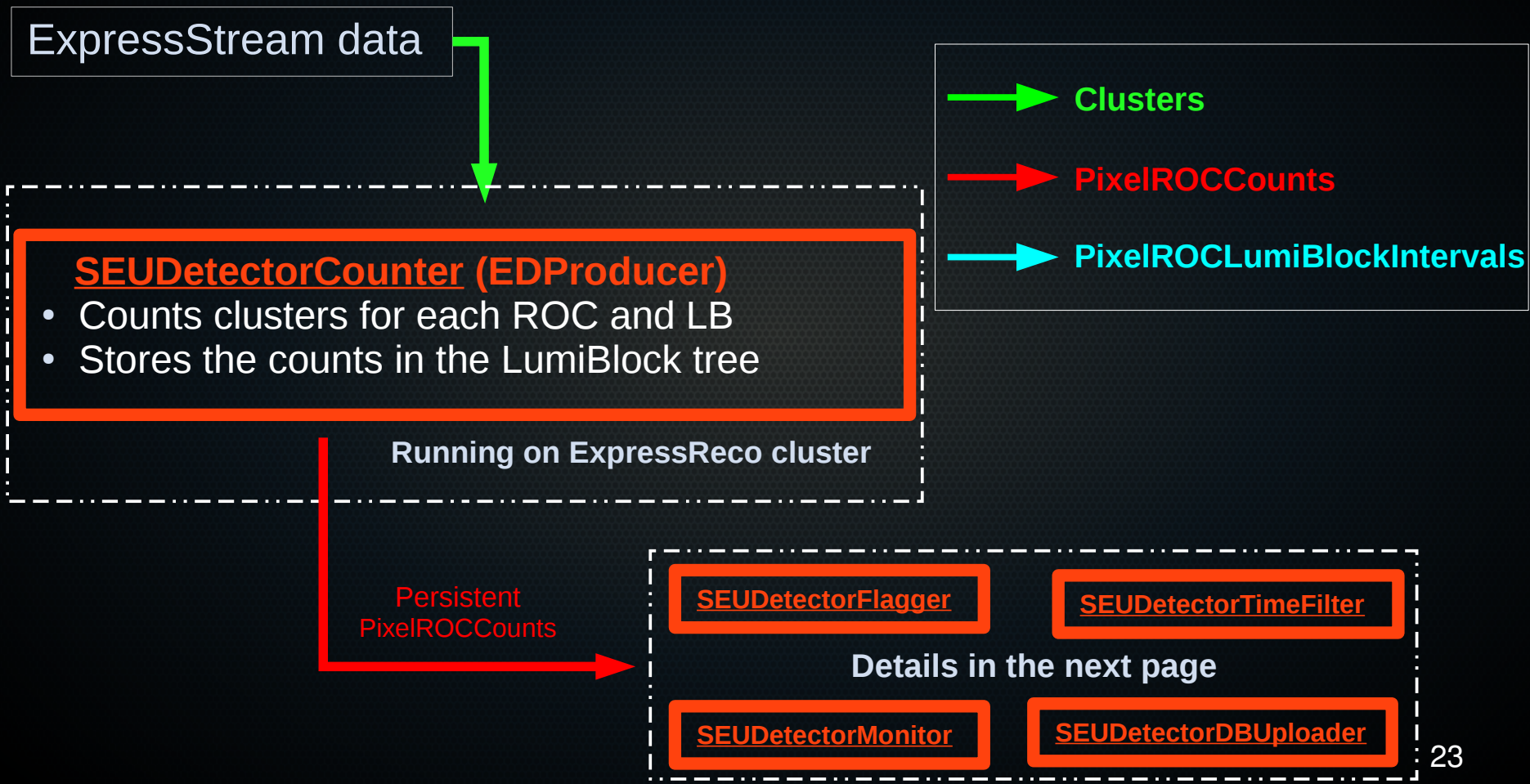


Implementation in CMSSW

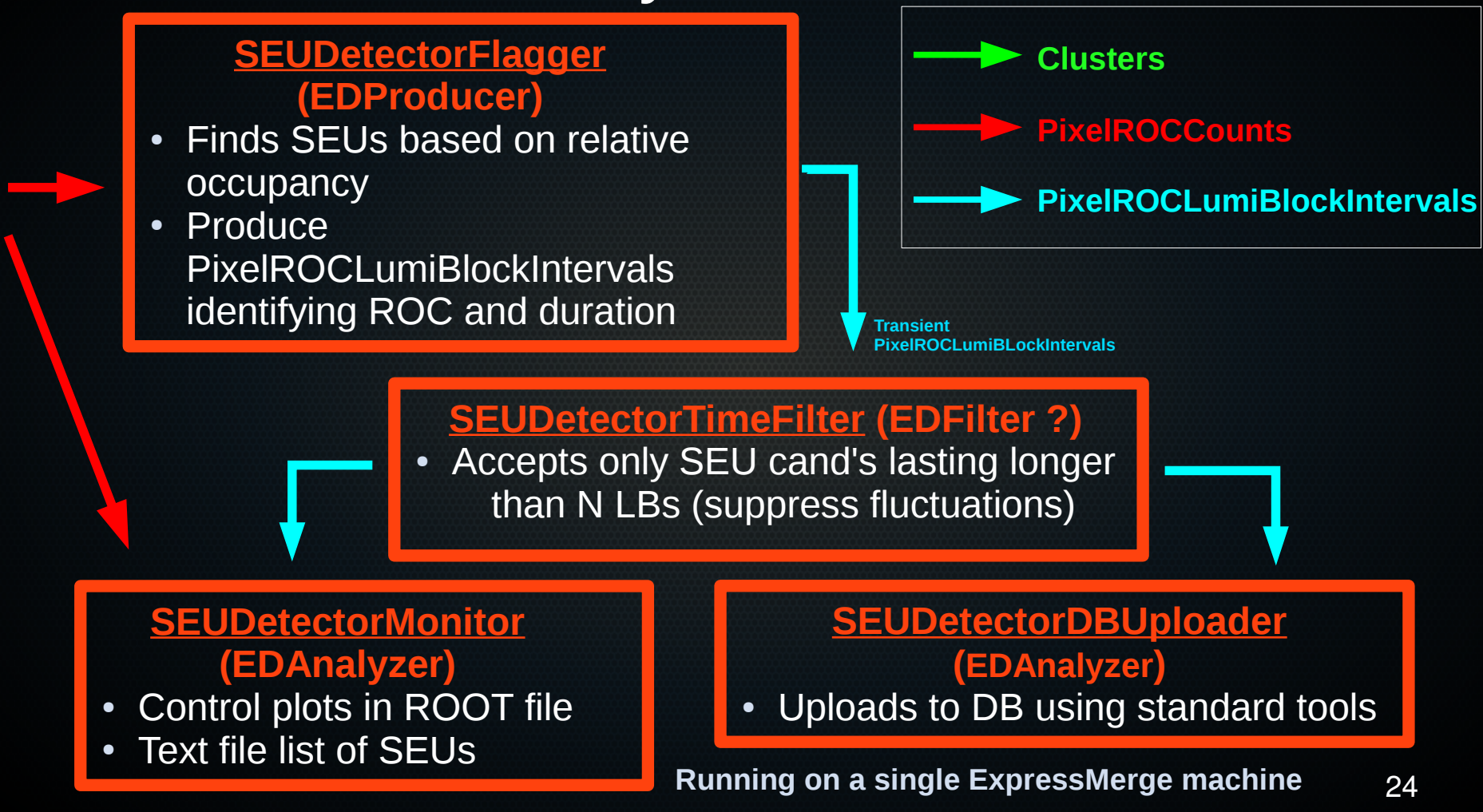
Foreseen Data Objects

- PixelROCId
  - detId (uint32_t) + roc number (uint32_t)
  - Identifies uniquely a ROC
- PixelROCCounts
  - PixelROCId + counts (uint32_t)
  - Transports the counts across CMSSW modules
- PixelROCLumiBlockInterval
  - PixelROCId + FirstSEULB + LastSEULB (both unsigned int)
  - Describes the interval in which a SEU was in effect

Analisys workflow



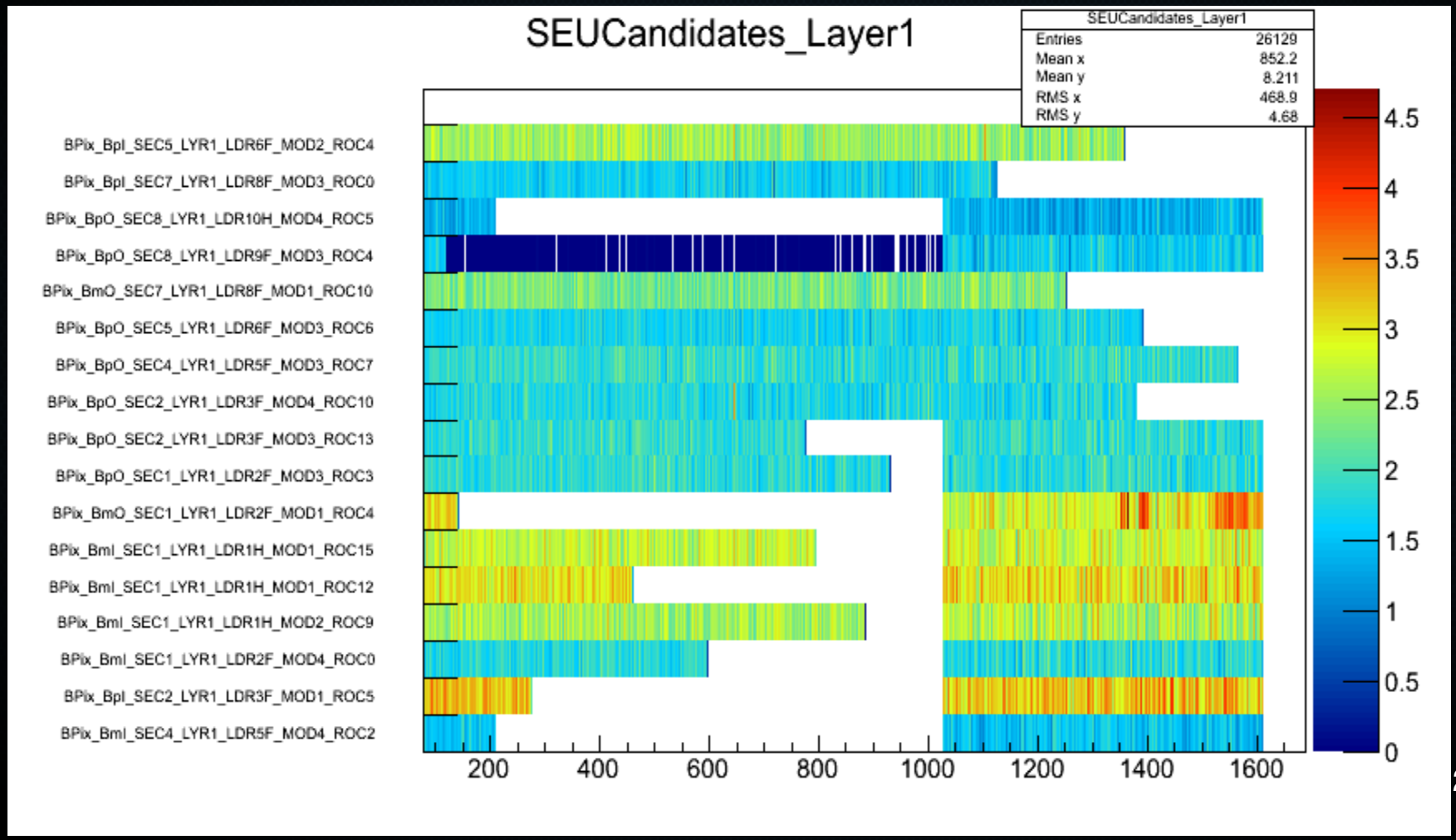
Analisys workflow



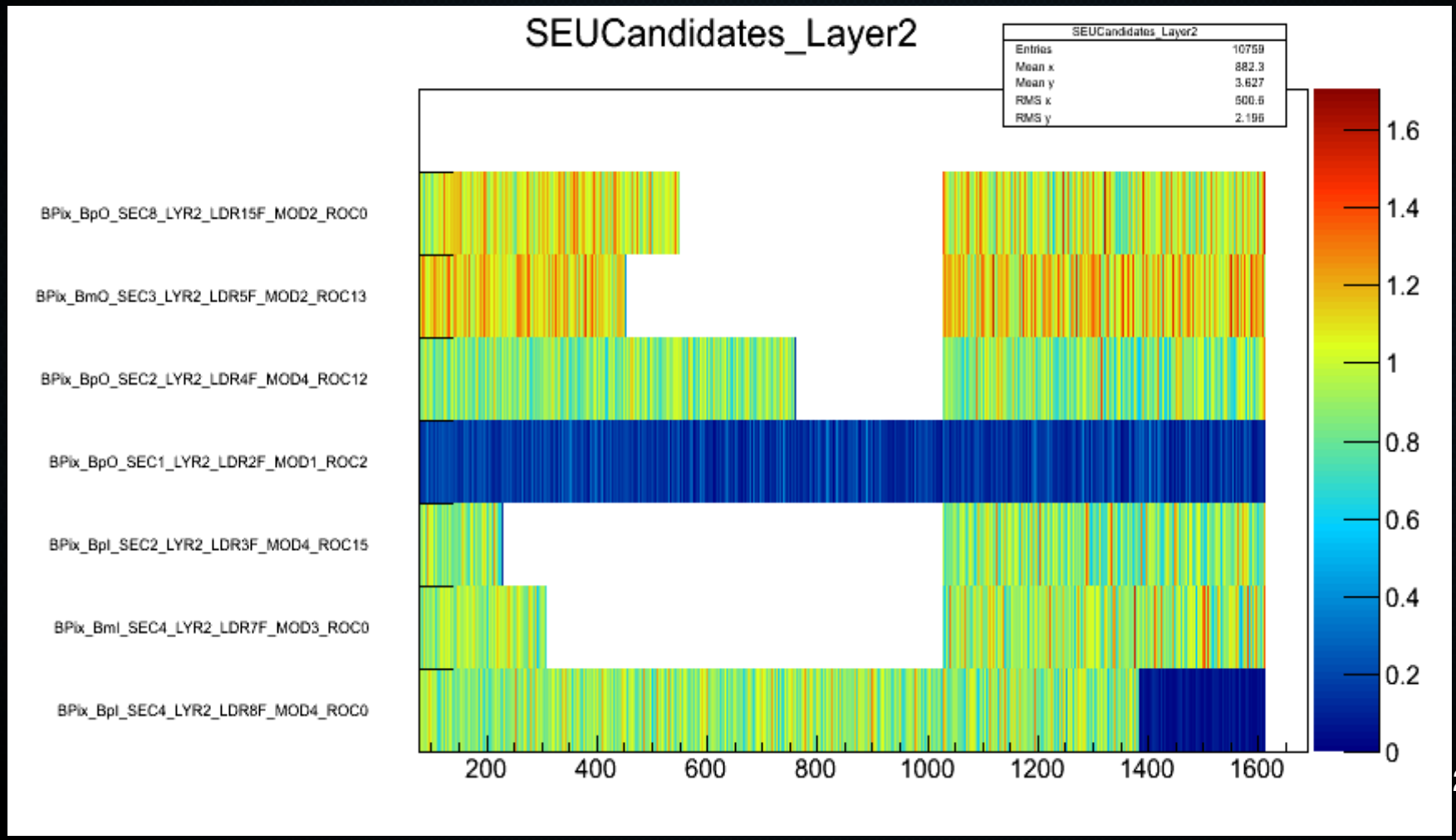
Running on a single ExpressMerge machine

Some additional examples from different runs

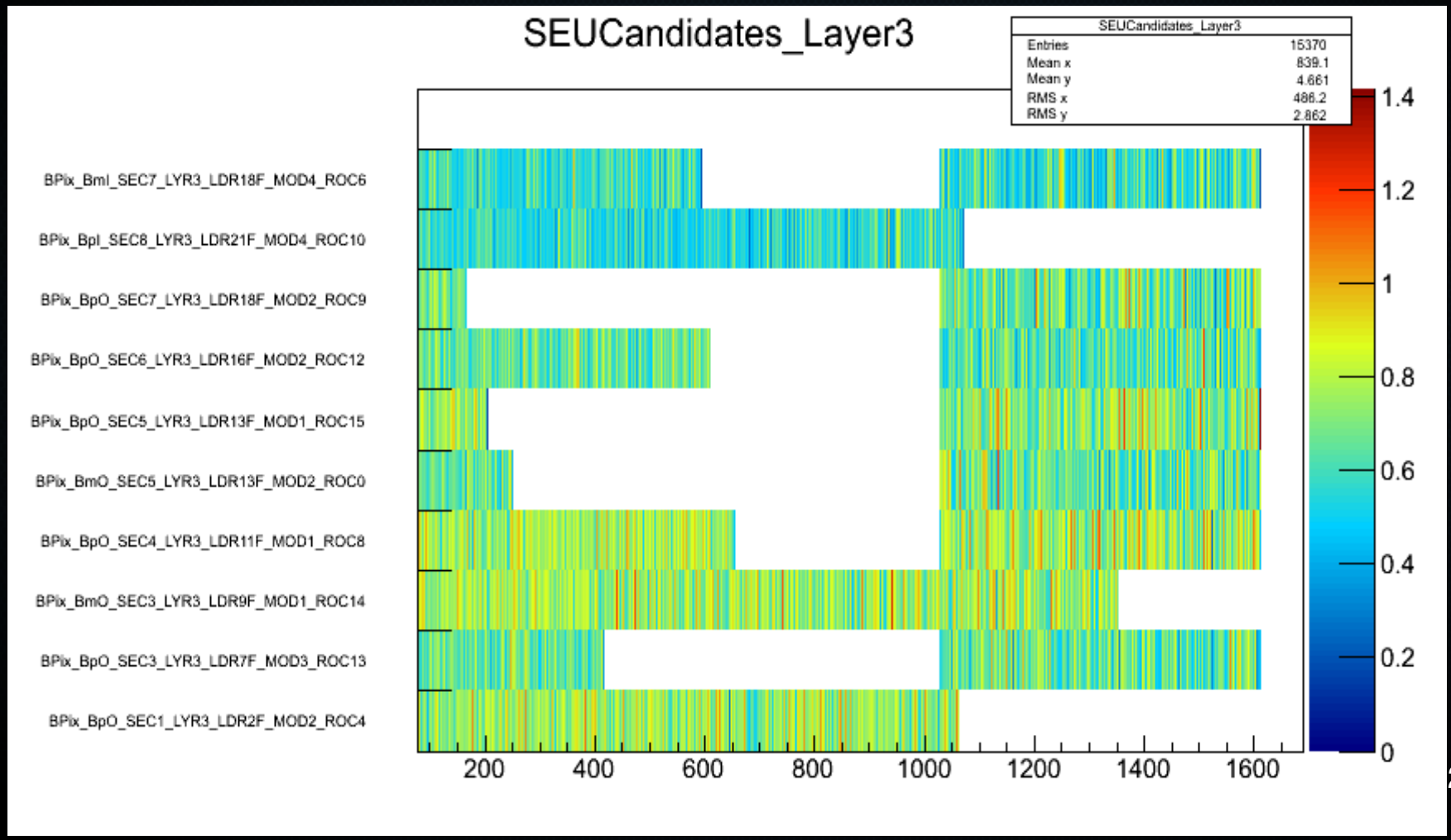
Run191226 Layer1



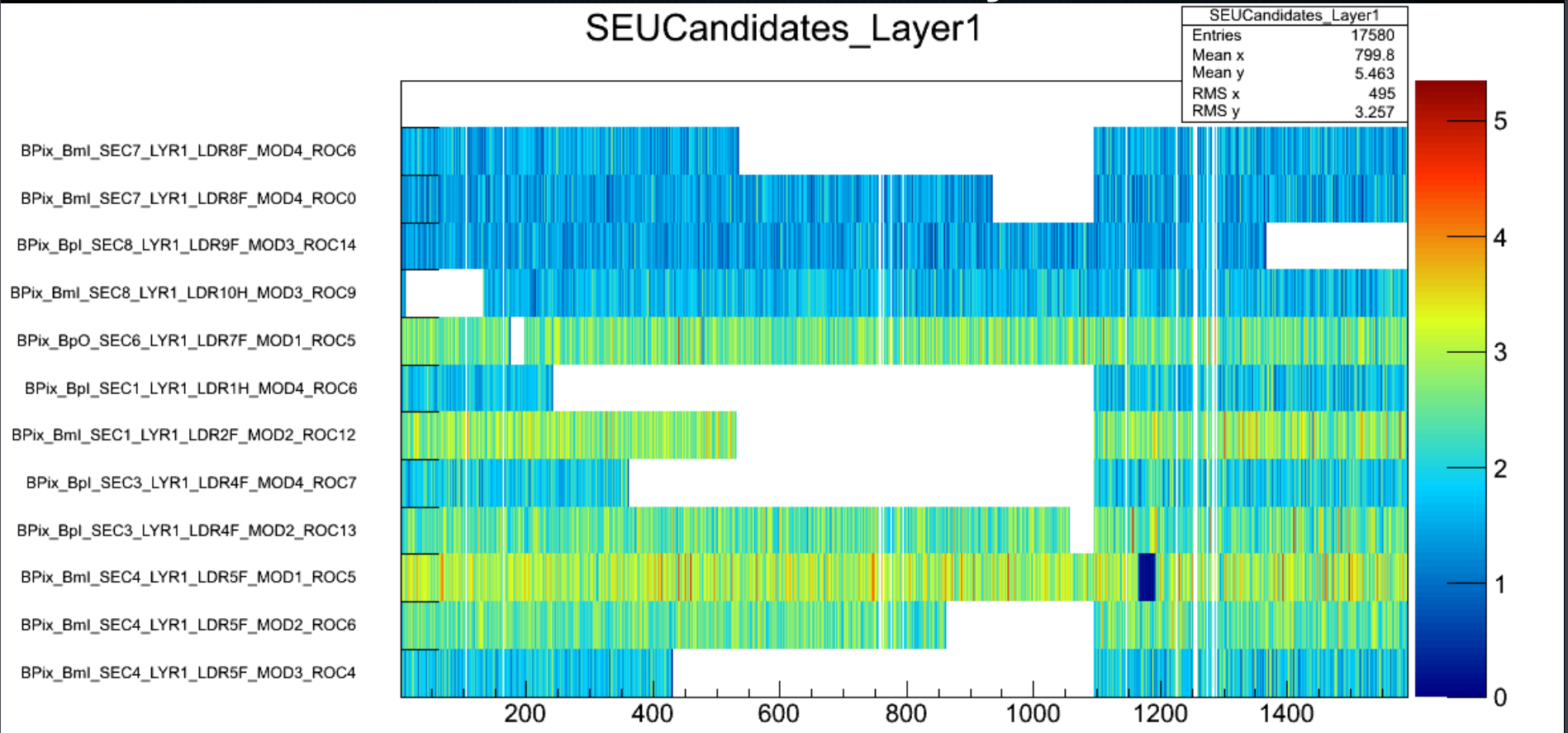
Run191226 Layer2



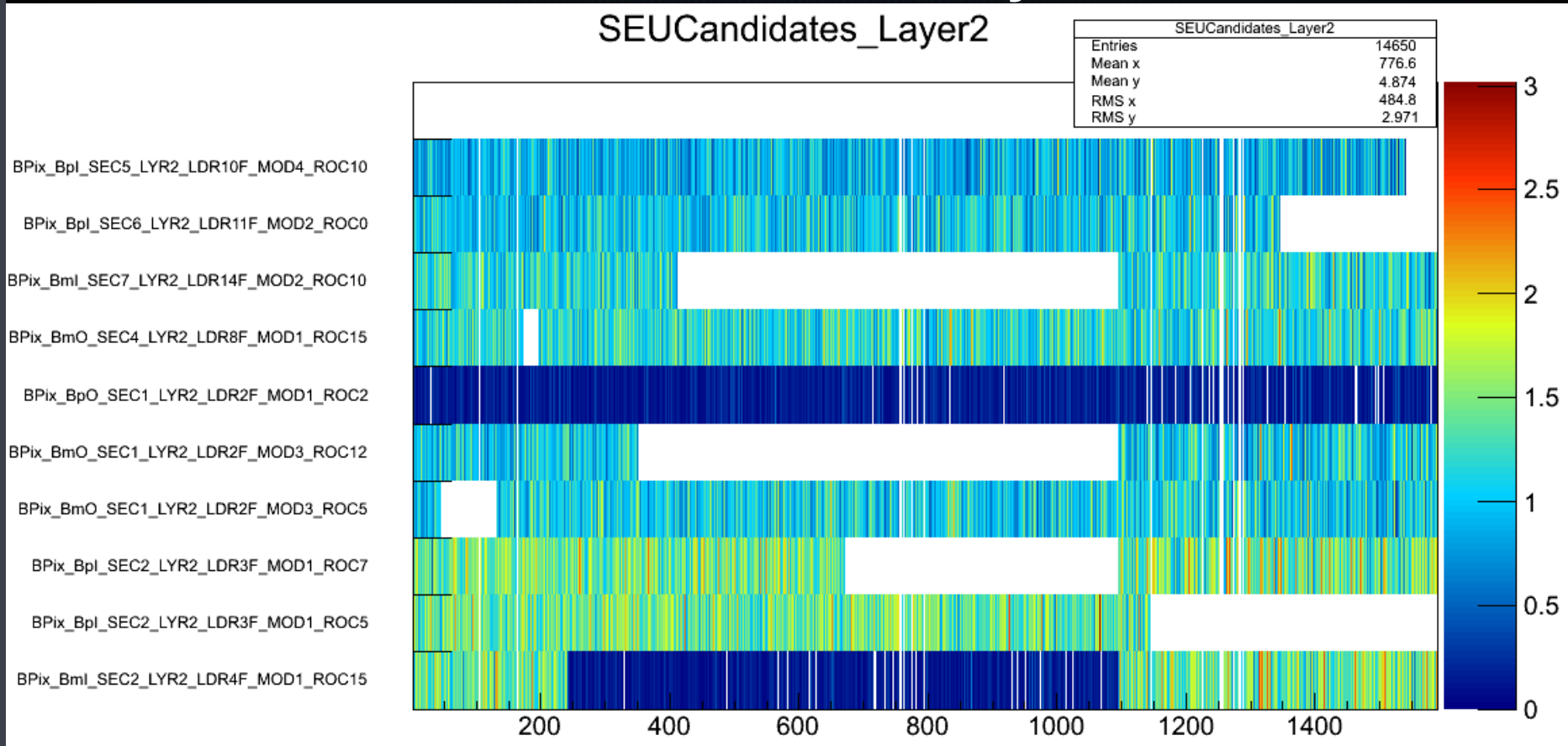
Run191226 Layer3



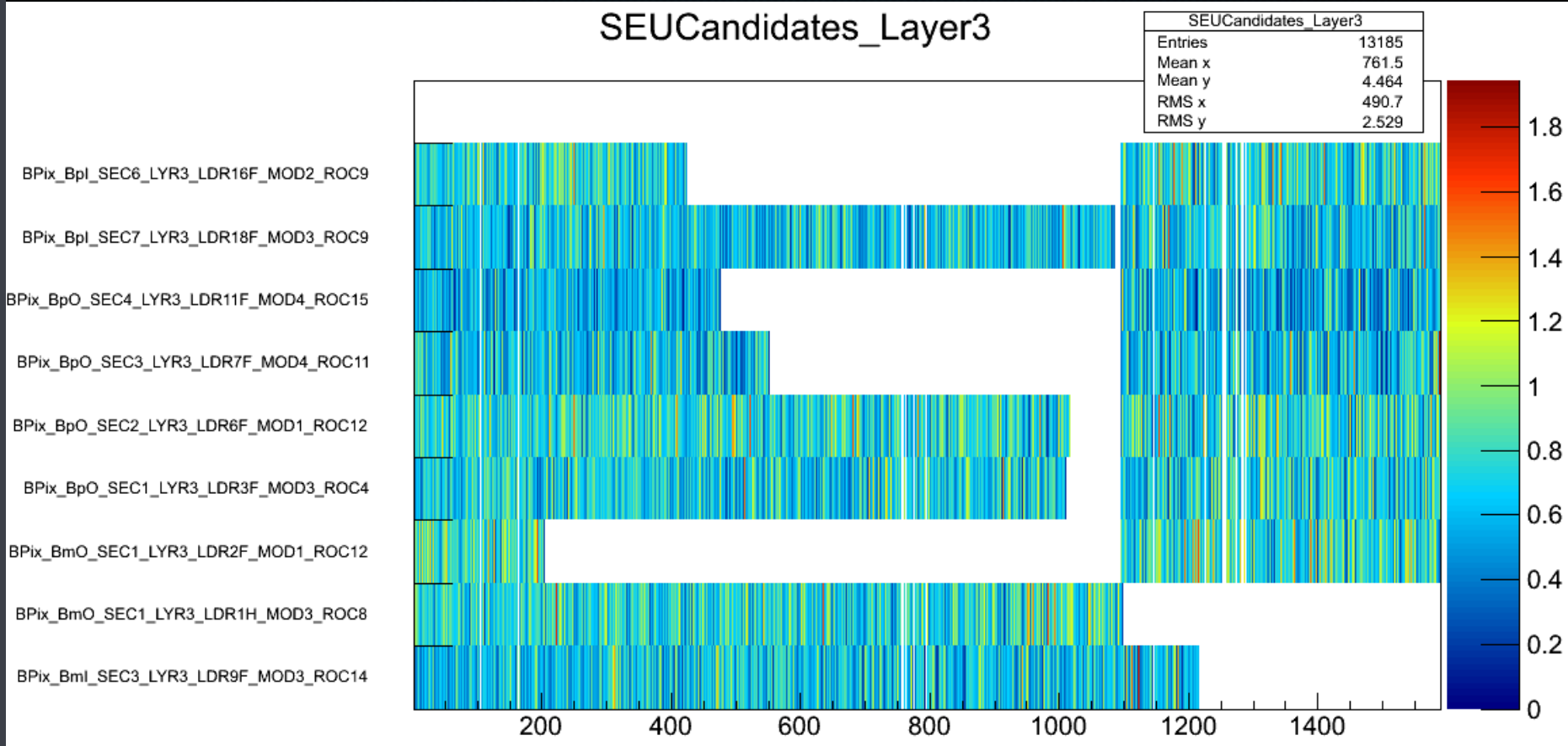
Run 195950 - Layer1



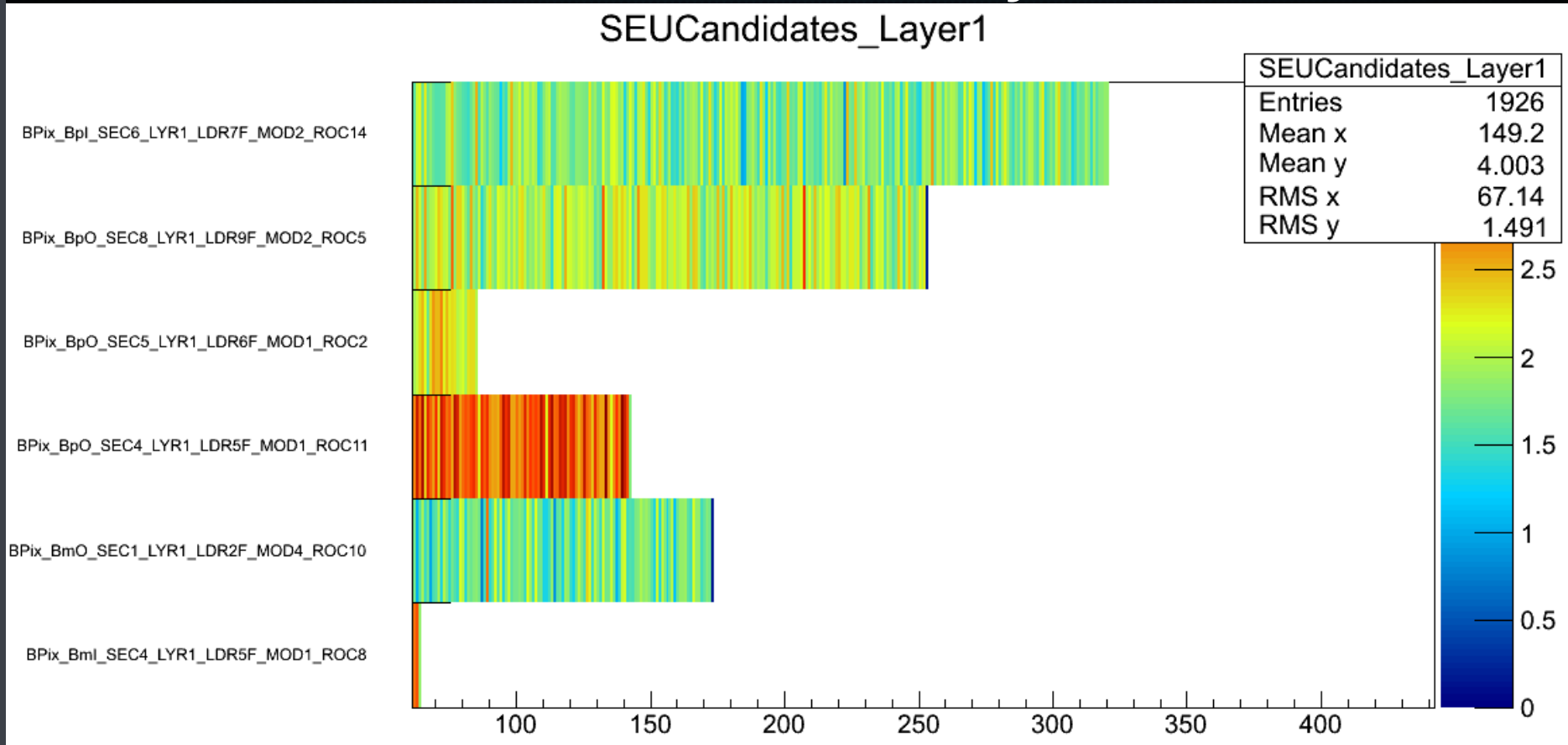
Run 195950 - Layer2



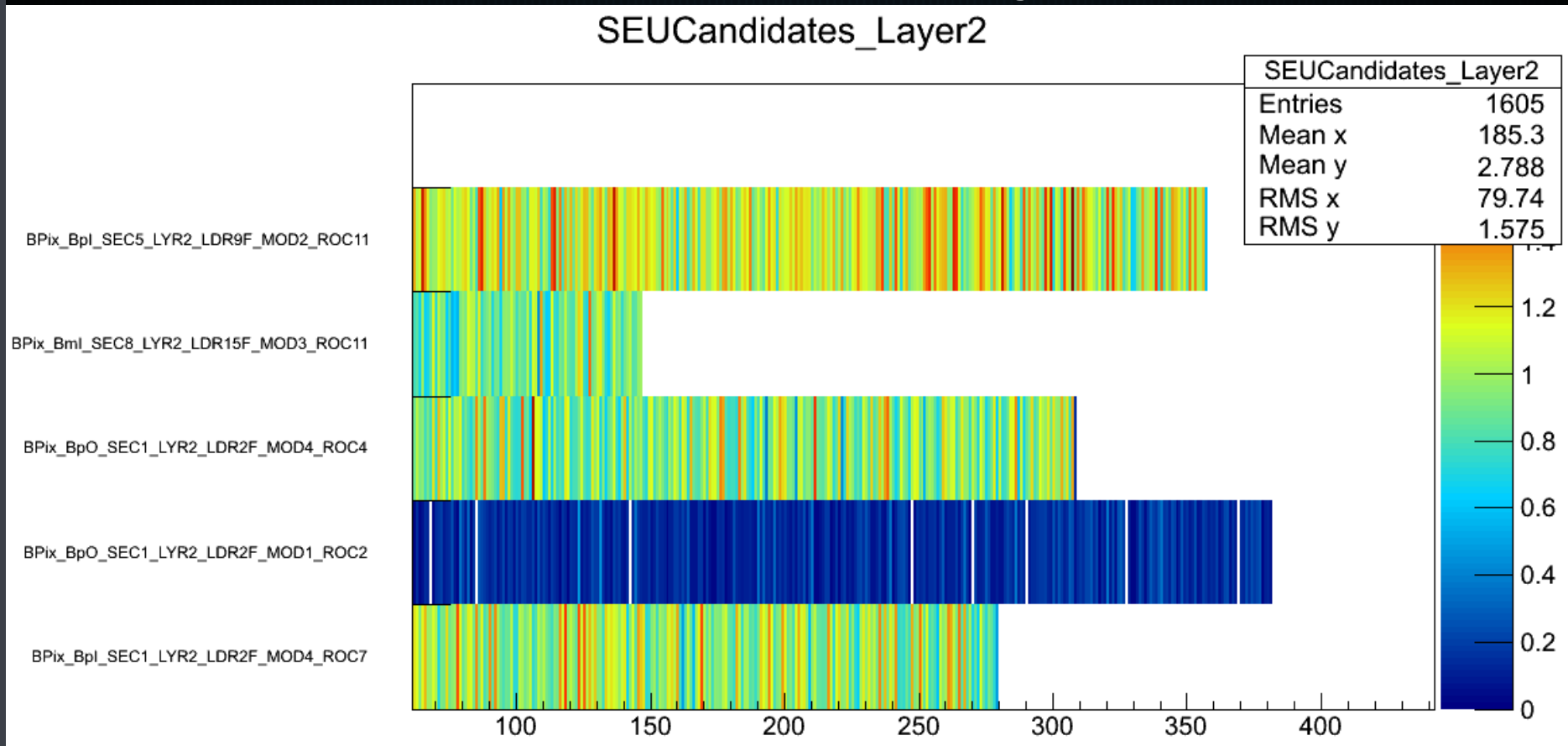
Run 195950 – Layer3



Run 196531 - Layer1



Run 196531 – Layer2



Run 196531 – Layer3