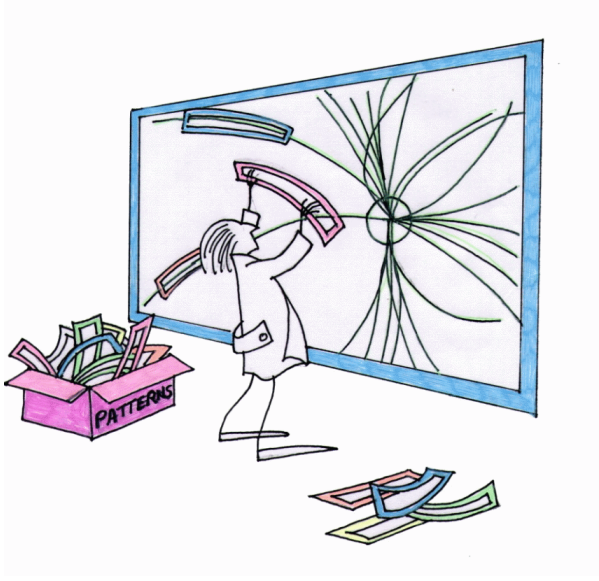
INTERNATIONAL CONFERENCE ON THECHNOLOGY AND INSTRUMENTATION IN PARTICLE PHYSICS '14,
AMSTERDAM, 2-6 JUNE 2014

# A HIGH PERFORMANCE MULTI-CORE FPGA IMPLEMENTATION FOR 2D PIXEL CLUSTERING FOR THE ATLAS FAST TRACKER (FTK) PROCESSOR

**Calliope-Louisa Sotiropoulou** on behalf of the FTK Pixel Clustering Team

Aristotle University of Thessaloniki
AUTH e-LAB
Aristotle University of Thessaloniki-Electronics Laboratory

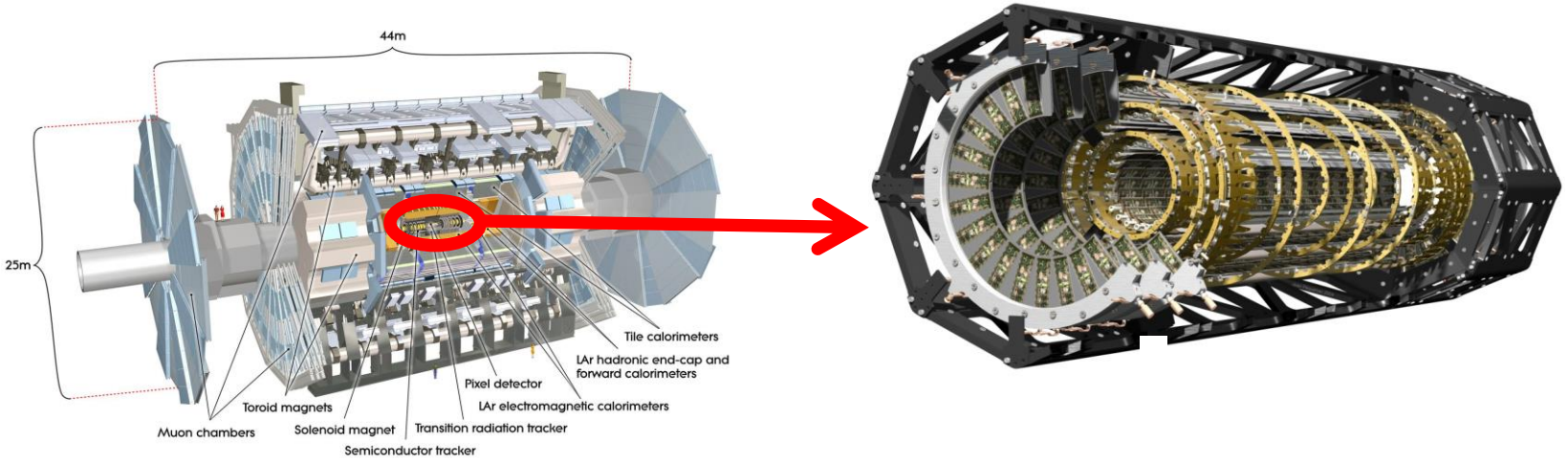
ATLAS EXPERIMENT



# Introduction to 2D Pixel Clustering

- High Resolution Pixel Detectors are used for a variety of applications:
  - High Energy Physics (Particle Tracking)
  - Medical Imaging and Biomedical Applications
  - Astrophysics Image Acquisition
  - Security etc.
- Most of these applications call for real-time data capture at high input rates
- Effective data reduction techniques with minimal loss of information are required
- 2D Pixel Clustering is a well suited choice
  - Our implementation is developed for the ATLAS FTK processor but can be easily adapted to other image processing applications

# The ATLAS Pixel Detector

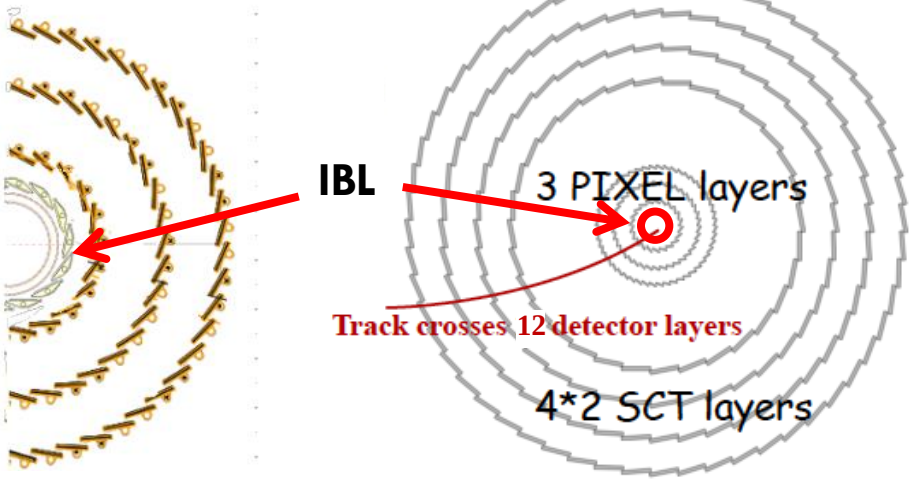



The spatial resolution of individual pixel modules has been measured at normal incidence at 12 $\mu m$

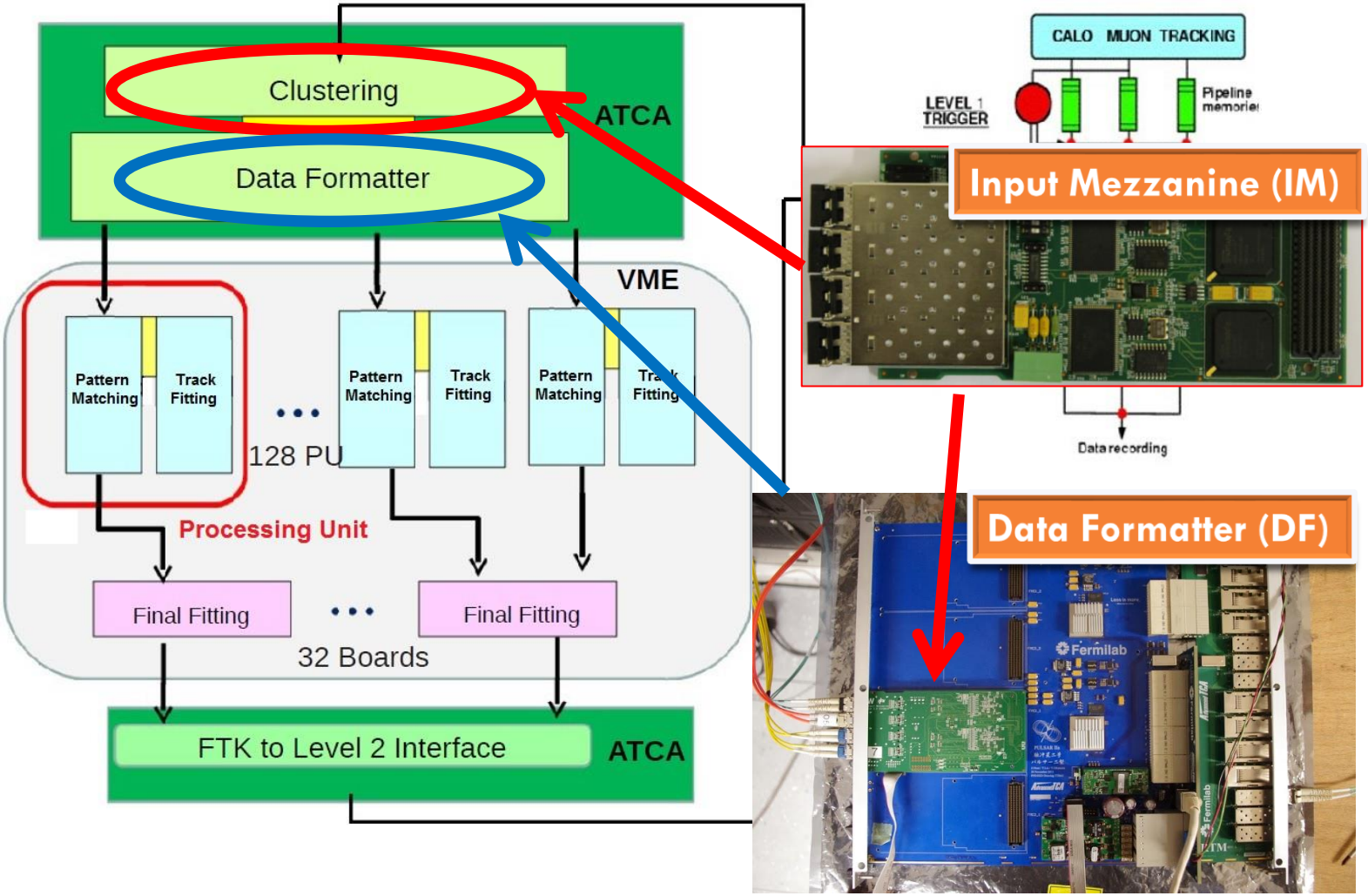
Another pixel layer is being added inside the current 3 pixel layers, the **Insertable B-Layer** (**IBL**)

***r-φ* view of pixel detector**

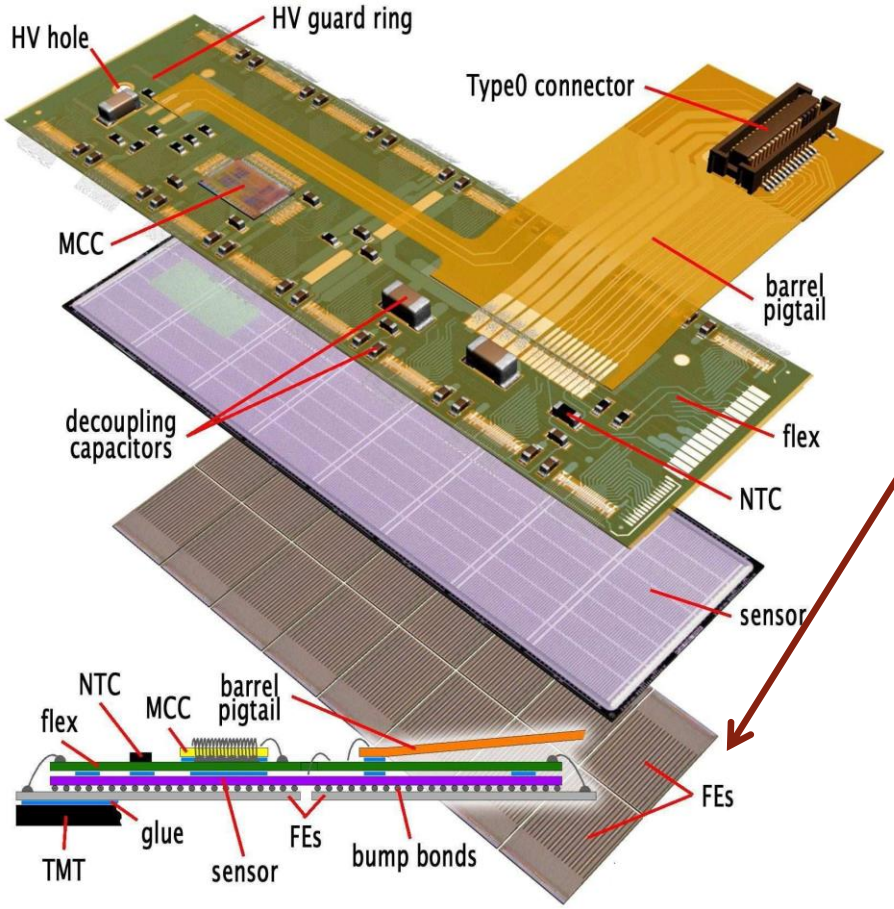
***r-φ* view of barrel region**



# FTK Architecture



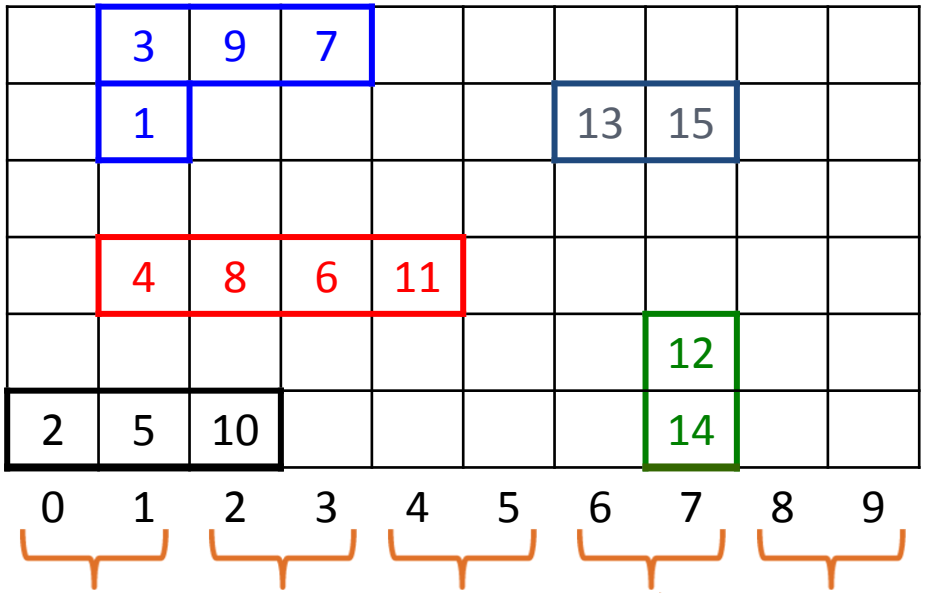
# The ATLAS Pixel Module



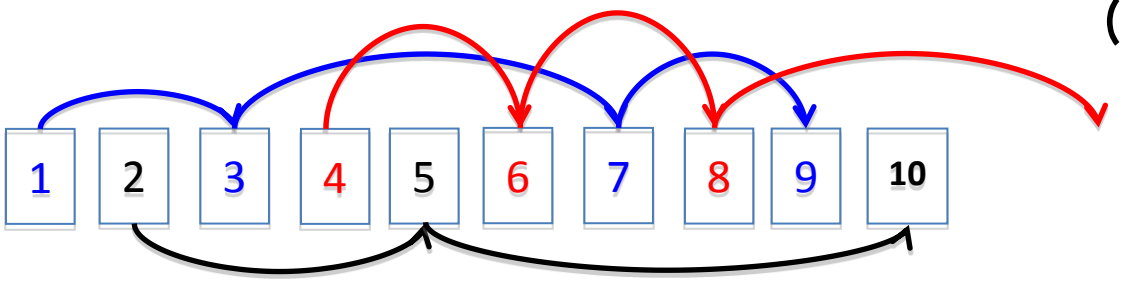

G. Aad et al., "ATLAS pixel detector electronics and sensors", Journal of Instrumentation, Vol. 3, P07007, July 2008.

- Each pixel module has 144 x 328 pixels
- The pixels are readout by 16 Front End (FE) chips
- Each module is comprised of 2 rows of 8 FE
- The pixels are $400\ \mu m$ x $50\ \mu m$ or $600\ \mu m$ x $50\ \mu m$ (FE edges)

# The Clustering Problem




- The Cluster Hits arrive scrambled
- The hits are additionally scrambled by dual column

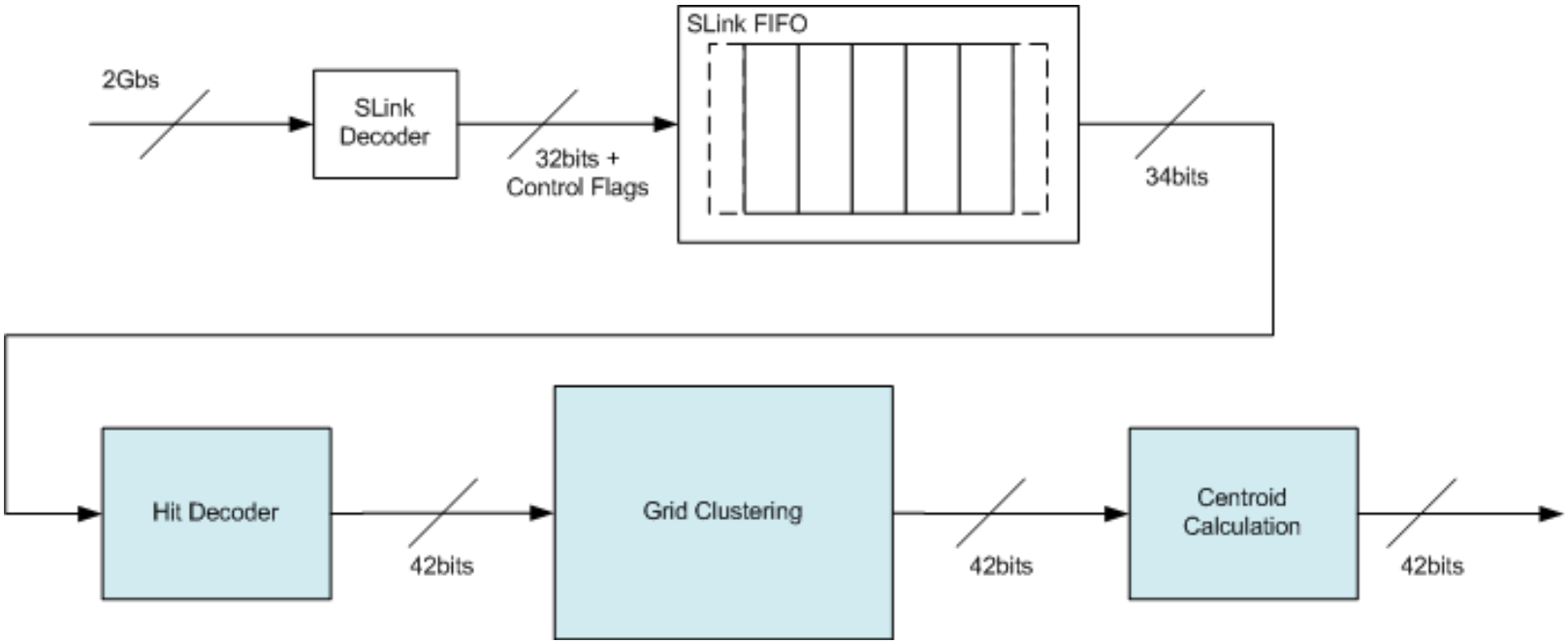



- Associate hits from same cluster
  - Loop over hit list
  - Time increases with occupancy & instantaneous luminosity
  - Non linear execution time
- **Goal: Keep up with 40 Mhz hit input rate**
- **Total input rate 308 Gb/s (coming from Serial Links)**

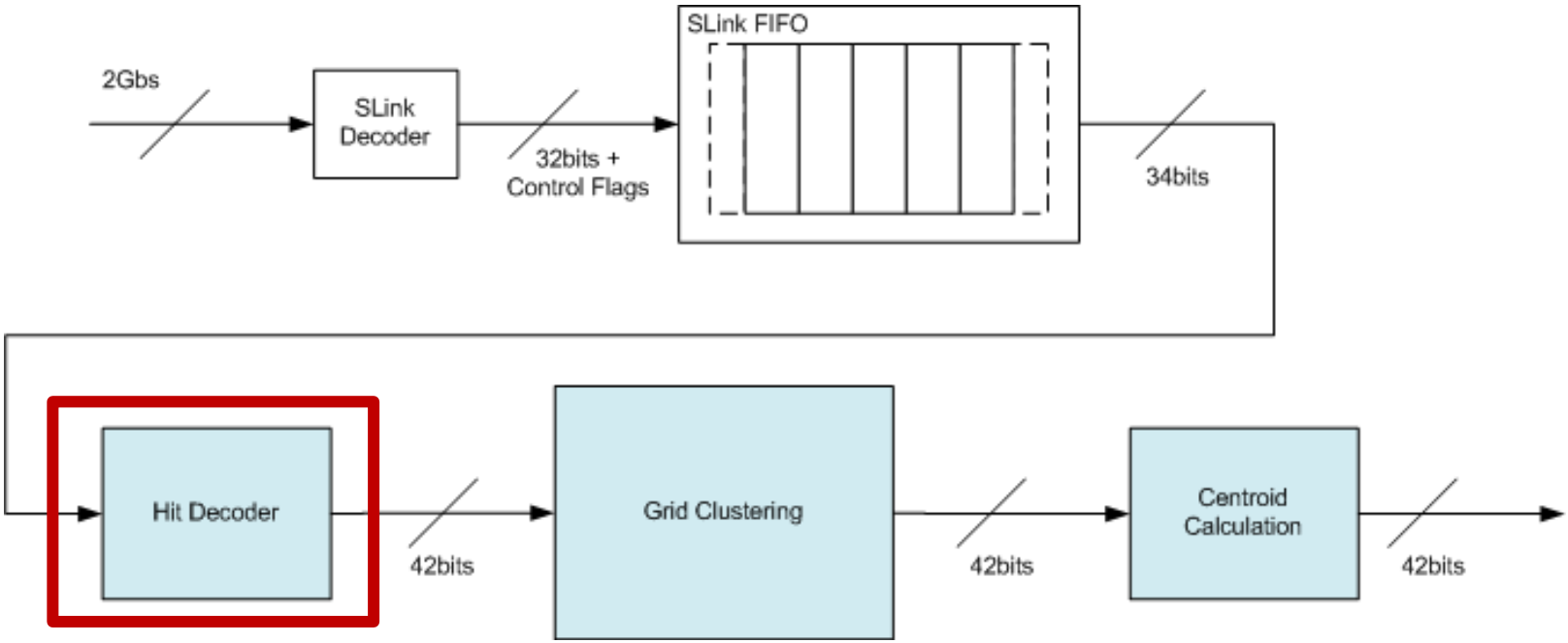
# 2D-Clustering for the ATLAS FastTracKer Processor

- What is needed is a **fast** 2D-Pixel Clustering implementation
- **Generic** enough to allow a design space exploration to adapt to the ATLAS detector needs
- With a prospect to be used for **different image processing applications**

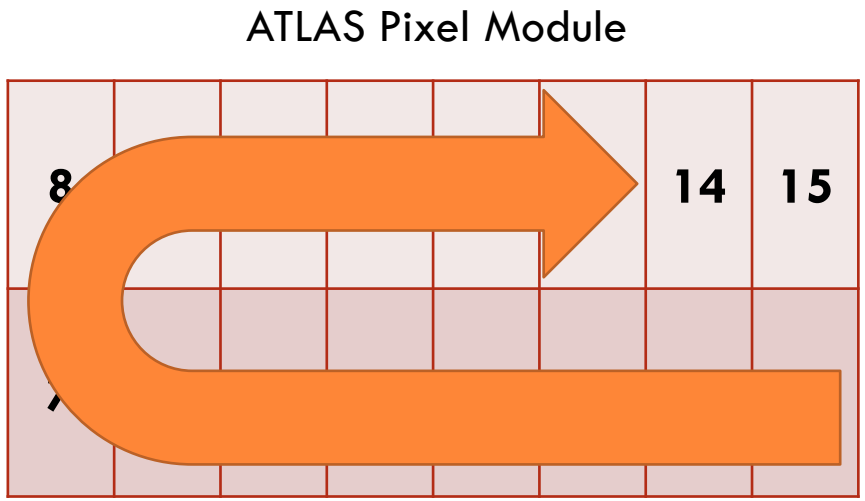
# 2D-Pixel Clustering



# Clustering Modules

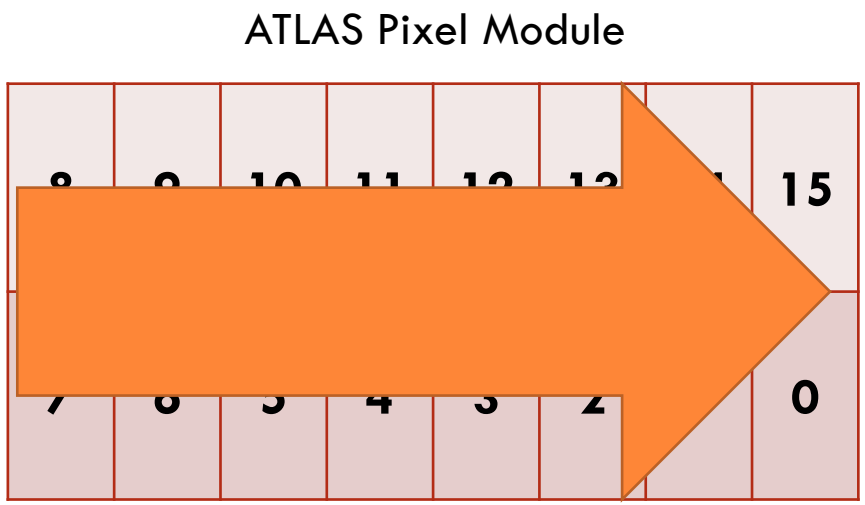


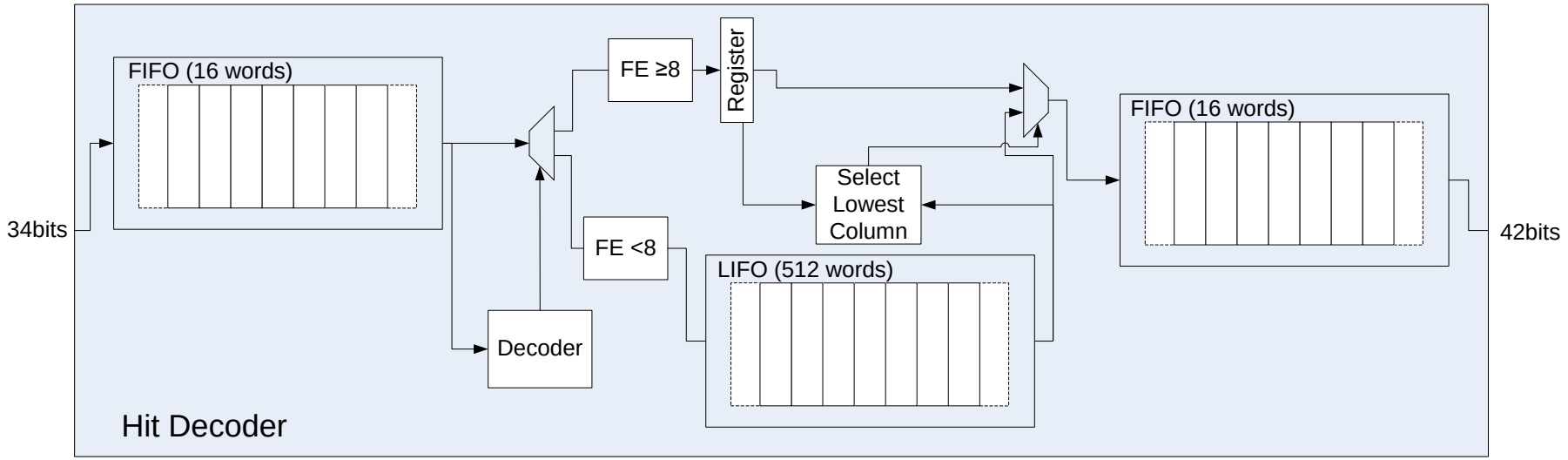
# Hit Decoder – Issues addressed



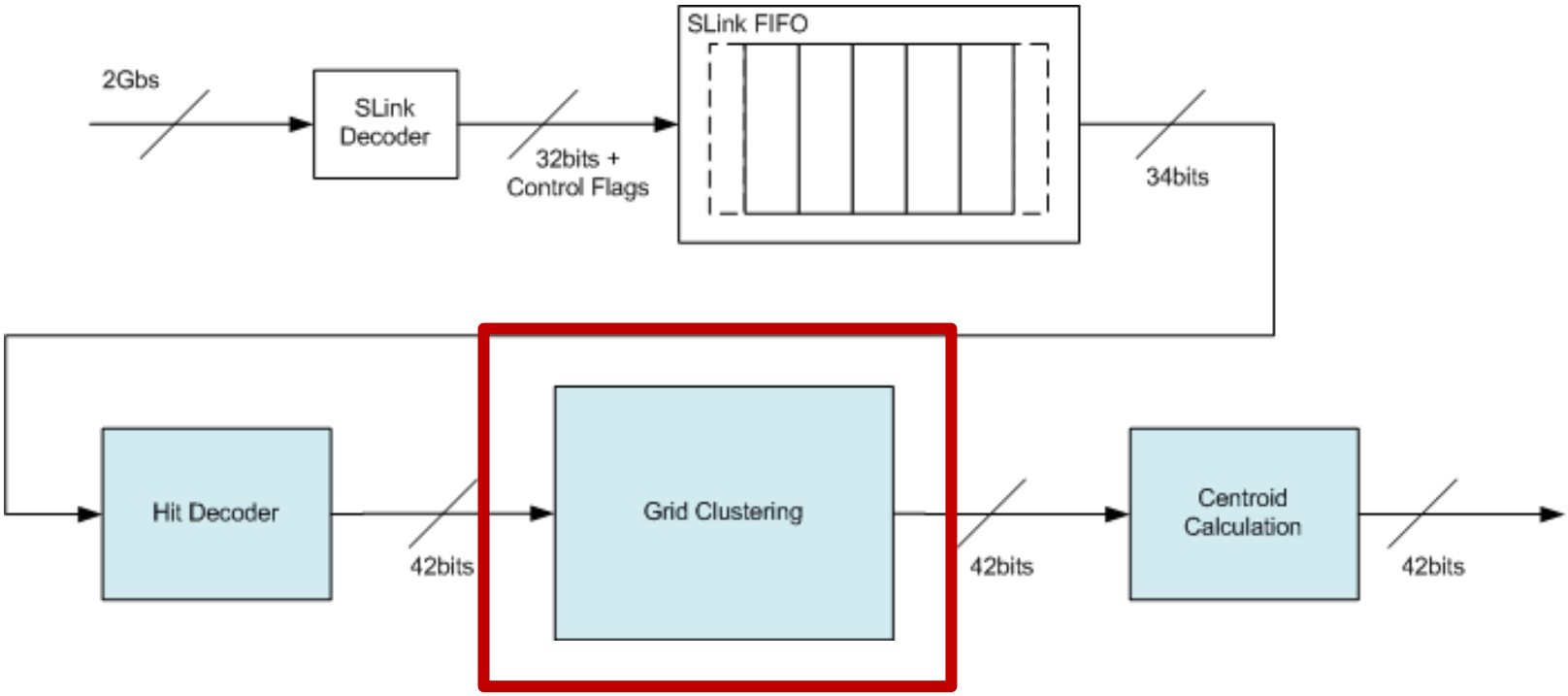

- The Hit Decoder outputs a **robust** and **aligned** data stream
- FE chip numbering is circular
- FE chips are readout in the order of their numbering →
- Incoming data is not in sequence:
  - Hits from FE <8 come in the reverse column order
  - Hits from FE>=8 come in the correct column order

# Hit Decoder – Issues addressed

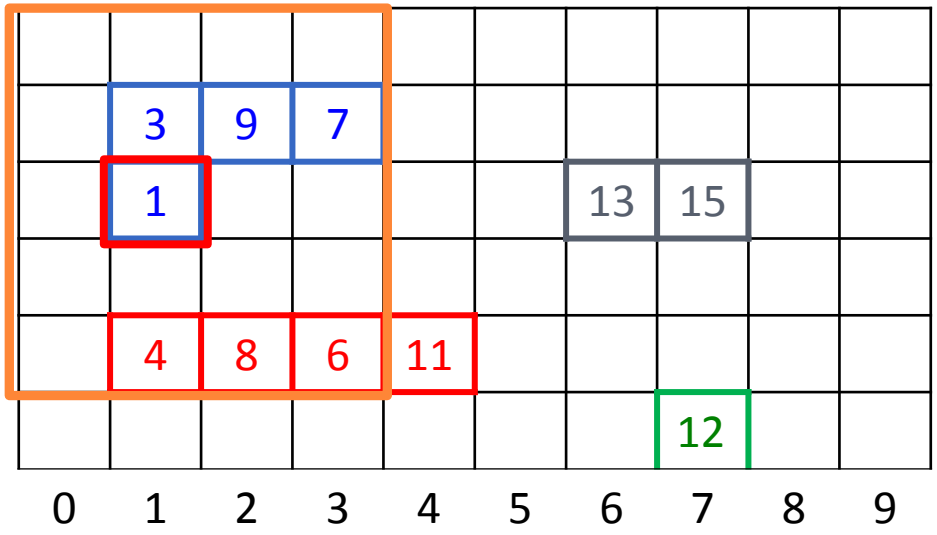


# Clustering Modules



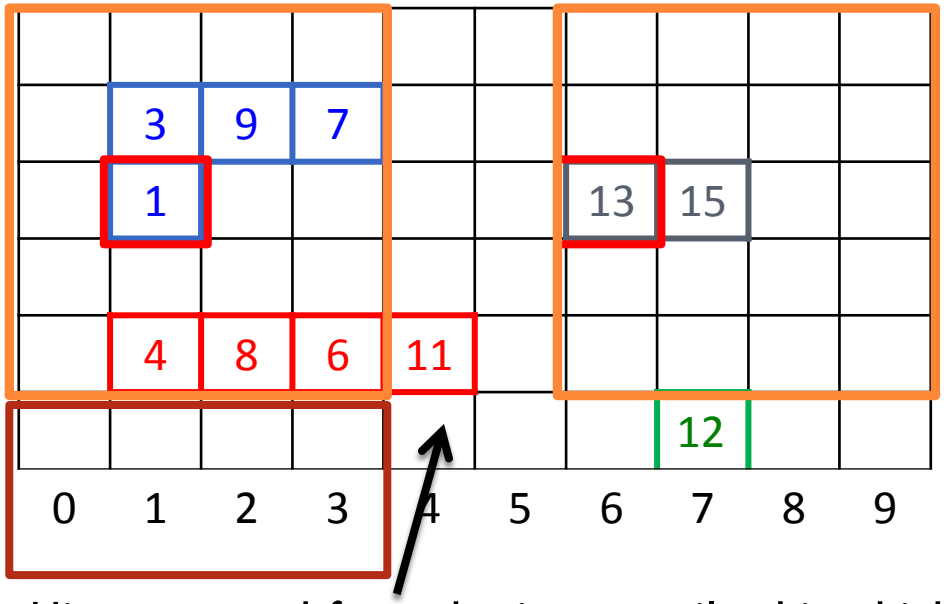
# Grid Clustering – Generate Window




- Generate a **cluster window** (e.g. 4x5 pixels) around a reference hit
- The reference hit is located on the **middle row** of the window and
  - **Column 1** of the window if it belongs to an **odd** column

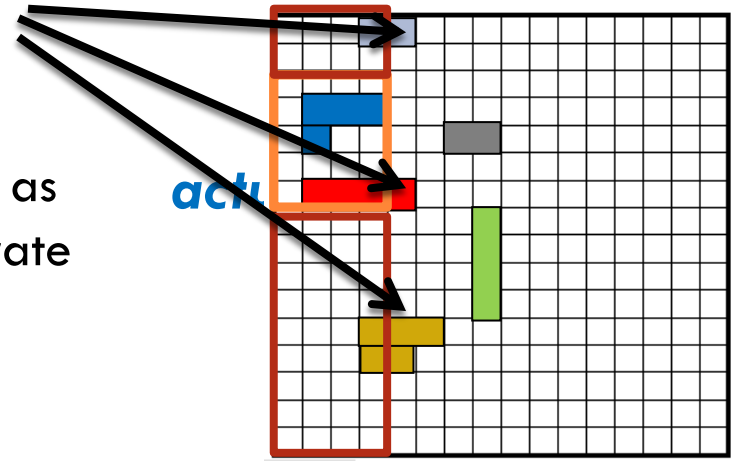
***actual grid size is 8x21***

# Grid Clustering – Generate Window

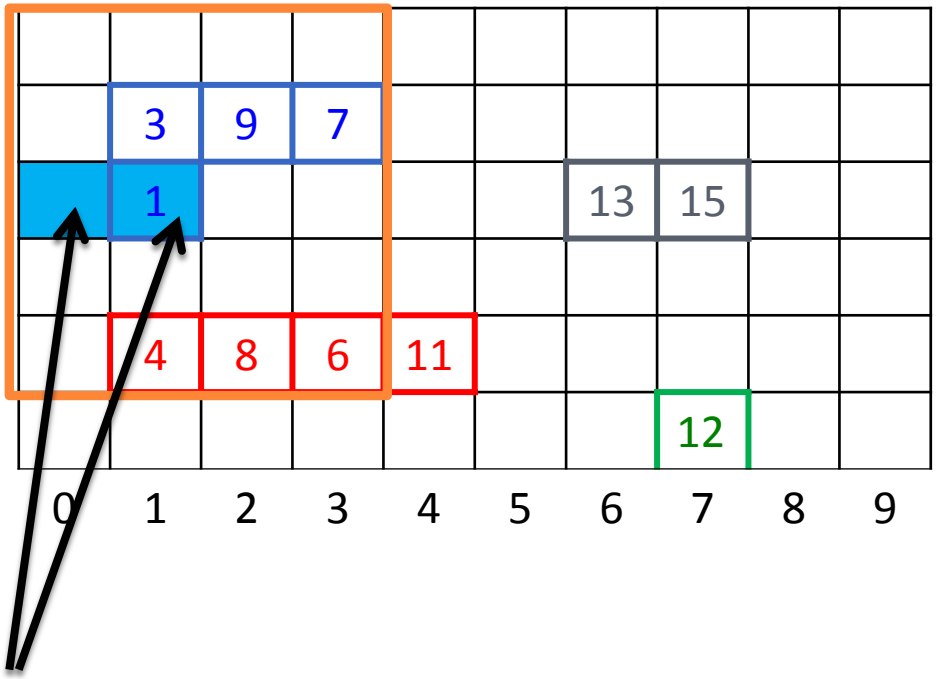



- Generate a **cluster window** (e.g. 4x5 pixels) around a reference hit
- The reference hit is located on the **middle row** of the window and
  - **Column 1** of the window if it belongs to an **odd** column
  - **Column 0** of the window if it belongs to an **even** column

- Hits are read from the input until a hit which belongs to a column beyond the **cluster window** is identified
- The hits that belong to the same columns as the cluster window are stored in a separate **circular buffer**



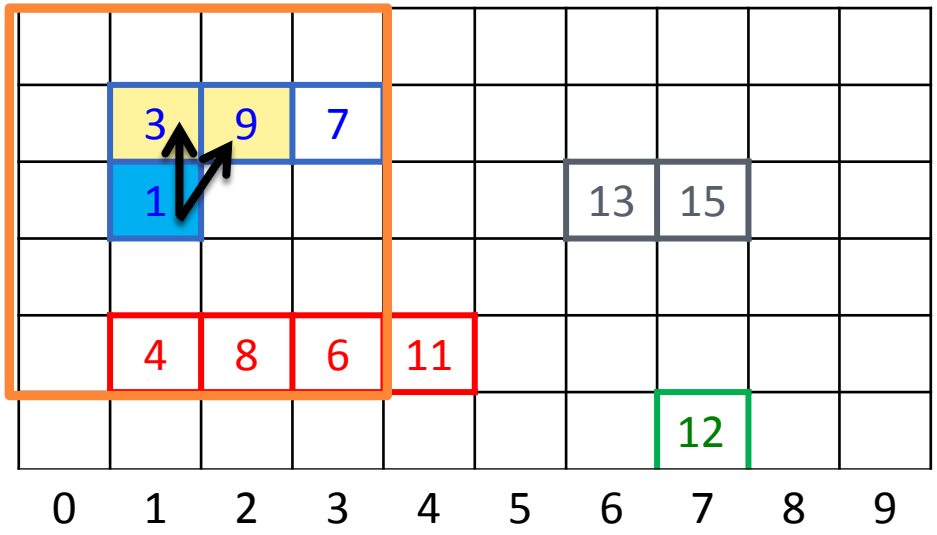
# Grid Clustering – Select and Readout




Selected grid cells

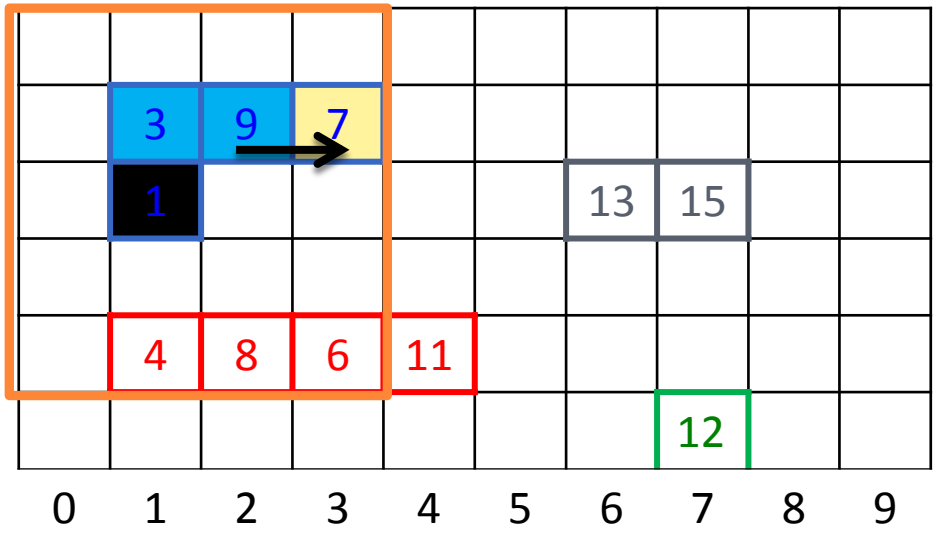
- Two cells in the **cluster window** are selected as "seeds"
  (0, mid-row) and (1, mid-row)

# Grid Clustering – Select and Readout



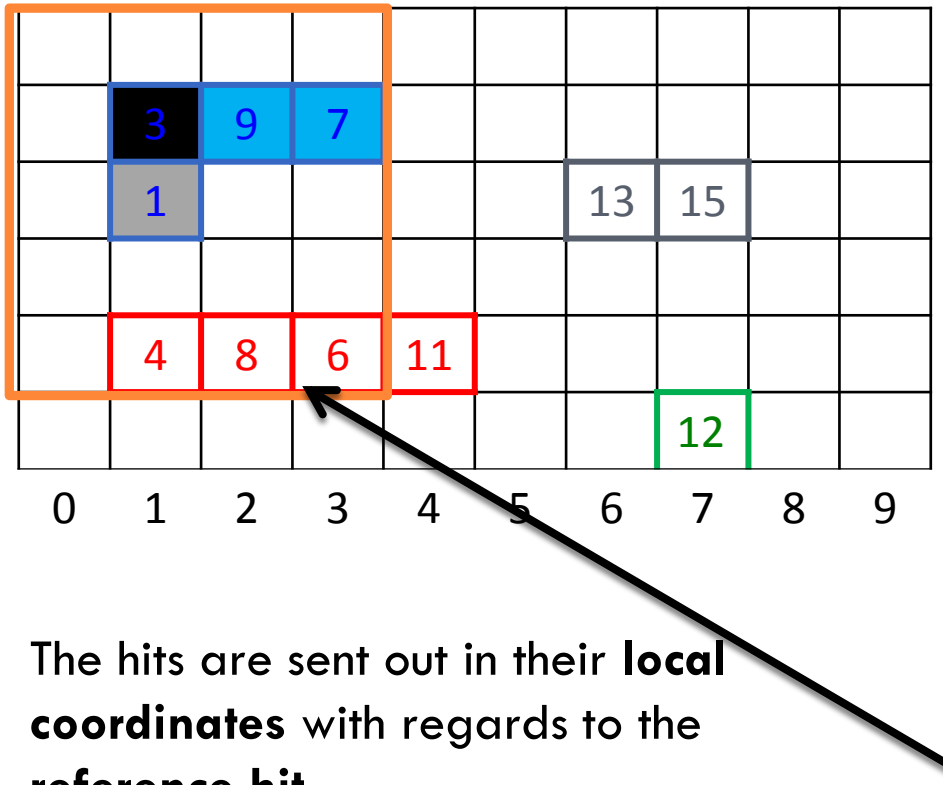

- Two cells in the **cluster window** are selected as "seeds"
  (0, mid-row) and (1, mid-row)
- The "selected" state is propagated to the neighboring hits

# Grid Clustering – Select and Readout




- Two cells in the **cluster window** are selected as "seeds"
  (0, mid-row) and (1, mid-row)
- The "selected" state is propagated to the neighboring hits
- One of the previous hits is now readout

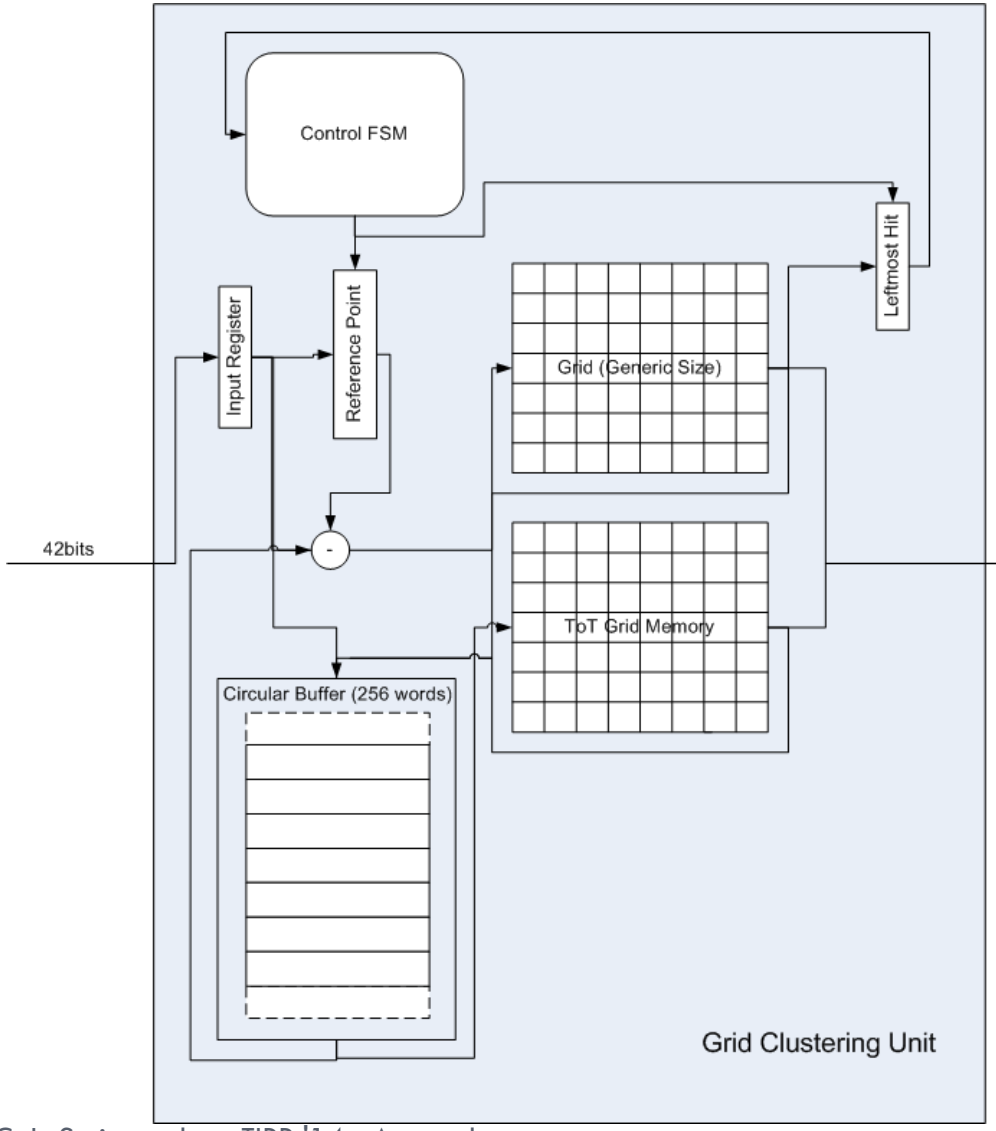
# Grid Clustering – Select and Readout




- Two cells in the **cluster window** are selected as "seeds"
  (0, mid-row) and (1, mid-row)
- The "selected" state is propagated to the neighboring hits
- One of the previous hits is now readout
- Selecting and reading out the cluster is executed in parallel
- The hits that do not belong to the cluster are recovered and written to the circular buffer in their absolute coordinates

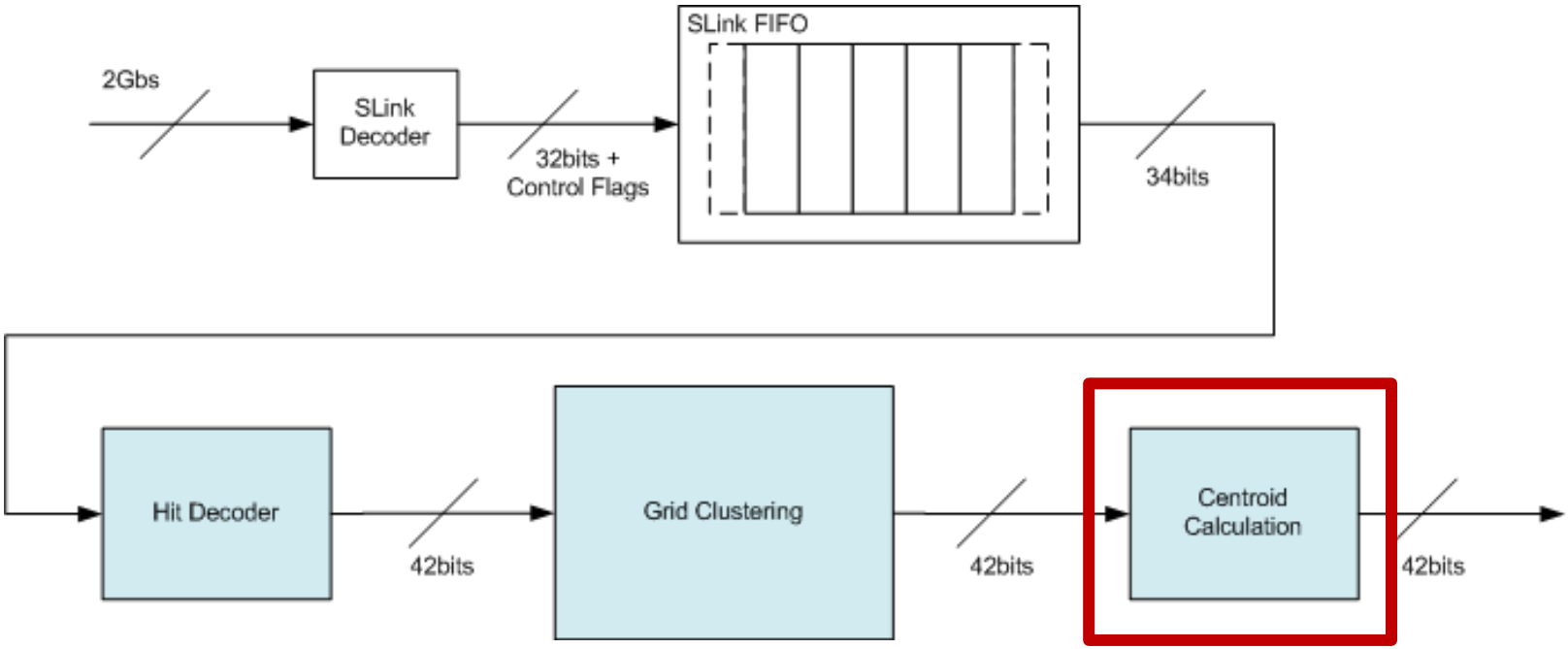
- The hits are sent out in their **local coordinates** with regards to the **reference hit**
- The **reference hit** is sent out in absolute coordinates in the **end cluster word**

# Grid Clustering – Block Diagram




- It is fundamentally **a control intensive data manipulation module**
- The pixel data is moved from the grid to the circular buffer and back to the grid until all clusters are identified
- This is why **execution time is data dependent**

# Clustering Modules



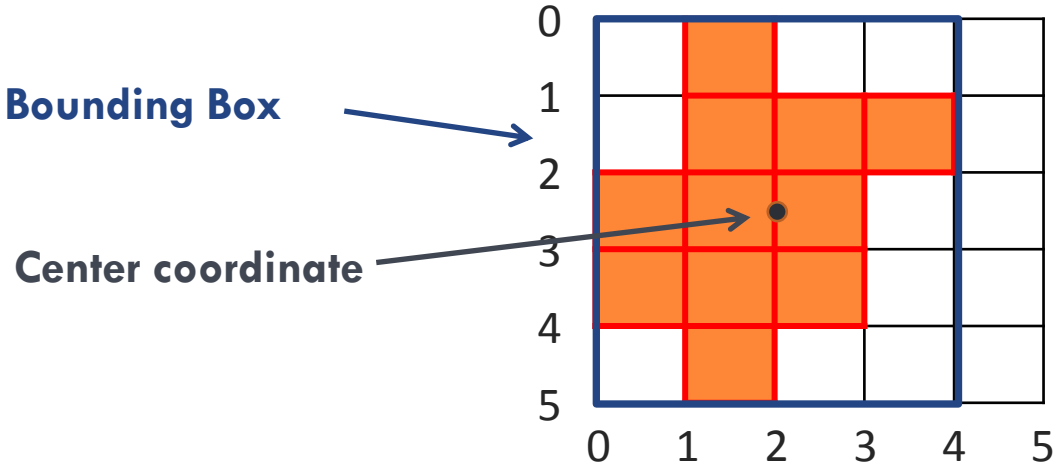
# Centroid Calculation

- The Centroid Calculation step will output the cluster centroid and also the cluster size in two coordinates
- This is were the actual data reduction takes place→ One cluster is reduced to one single word
- To have an accurate centroid calculation the actual pixel size must be taken into account:
  - Pixels have different sizes in the *x* or *z* direction (400 $\mu m$ and 600 $\mu m$)
  - We transform the pixel coordinates to normalized units of **25 $\mu m$** for ***x*** direction and **5 $\mu m$** for ***y*** direction
- The Centroid Calculation is under development
- Current version calculates the centroid as the "**center of the cluster bounding box**"

# Centroid Calculation

- The bounding box is calculated with respect to the **center** of the **edge pixels** of the cluster

  (given example is in pixel units, actual implementation is in normalized units of 25 $\mu m$ for columns and 5 $\mu m$ for rows)




- Min_col = 0
  Min_col_pix_center = 0.5
- Max_col = 4
  Max_col_pix_center = 3.5
- Min_row = 0
  Min_row_pix_center = 0.5
- Max_row= 5
  Max_row_pixel_center = 4.5

- Center_of_bounding_box_column_coordinate = (Min_col_pix_center+Max_col_pix_center)/2 = 2
- Center_of_bounding_box_row_coordinate = (Min_row_pix_center+Max_row_pix_center)/2 = 2.5

- Future version will take into account the charge distribution of the cluster and apply it as a correction to the centroid value

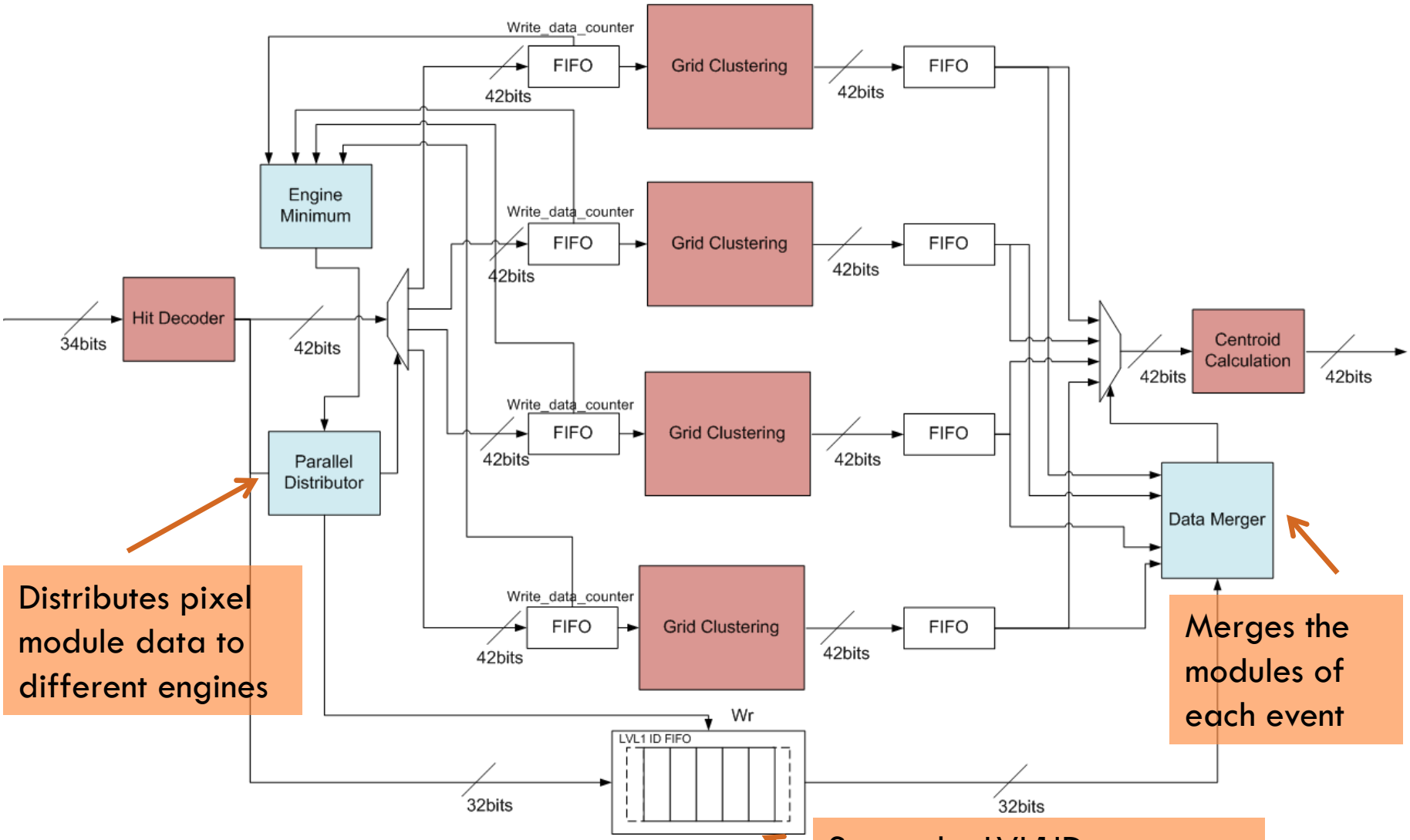
# 2D Pixel Clustering Parallel Version

- The fundamental characteristic of the 2D-Pixel Clustering implementation is that **different clustering engines can work independently in parallel on different data** →

  Increase performance with greater FPGA resource usage

- A parallelization strategy was chosen:
  - Instantiate multiple clustering engines (grid clustering modules) that work independently on data from separate pixel modules
  - To achieve this **data parallelizing** (**demultiplexing**) and **data serializing** (**multiplexing**) modules are necessary

# 2D Pixel Clustering Parallel Version



# 2D Pixel Clustering Implementation Results

- **Successful integration on hardware** (the FTK_IM board) for both Single Flow and 4 Parallel Engine Flow implementations
- Testing input files of events of 80 overlapping proton-proton collisions, which is the maximum LHC luminosity planned until 2022
- Output was compared with bit-accurate simulation with excellent accuracy

Xilinx Spartan-6 LX150T-3

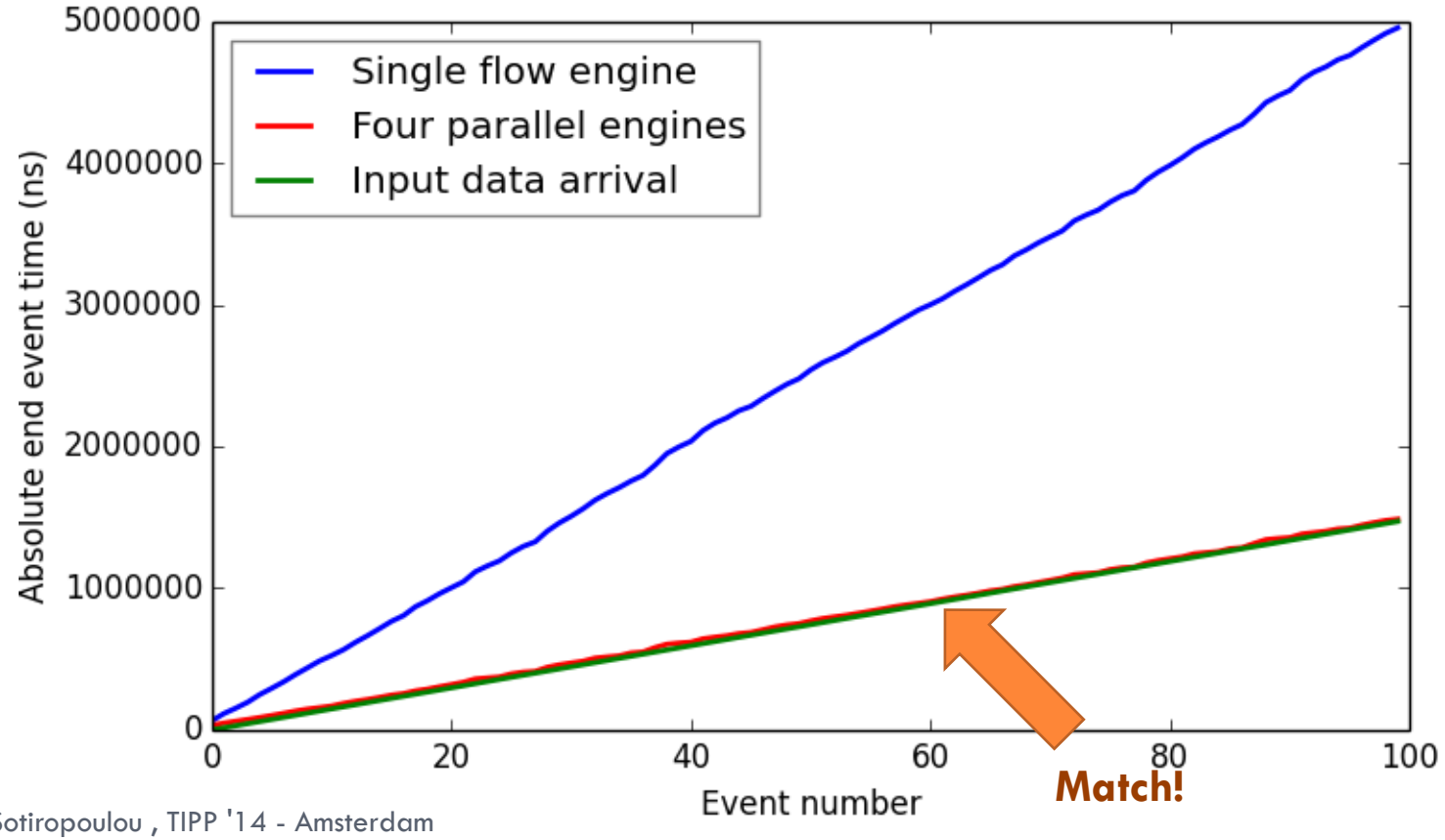
| | FPGA Area (%) | Max Frequency (MHz) |
|---|---|---|
| Single Flow | 2.5 | 81.5 |
| 4x Parallel Flow | 11.1 | 80.5 |
| 16x Parallel Flow | 38.4 | 65 |

# 2D Pixel Clustering Testing Results

- The maximum input data rate is one word per 25 ns (40 MHz)
- For the **Single Flow** the average processing time is one word per 84 ns (post place and route simulation measurements)
- 84 / 25 = 3.4 clock cycles of 40MHz are required for one data word processing
- Therefore it was anticipated that 4 engines would be sufficient for the worst case scenario (Pixel B Layer)
- A very important aspect was the data buffering (used FIFOs)
  A 4x Parallel implementation with small data buffering (FIFOs) failed to meet the requirements
- The 4x Parallel implementation with a sufficient buffer size achieved the required target

# 2D-Pixel Clustering Performance Plot

- The 4x Parallel Engine Flow fully covers pixel specifications for worst case (Pixel B Layer)
- The sample is 100 events where each event has 80 overlapping proton-proton collisions




**84ns / word**

**25ns / word (input rate)**

# 2D-Pixel Clustering - Conclusions

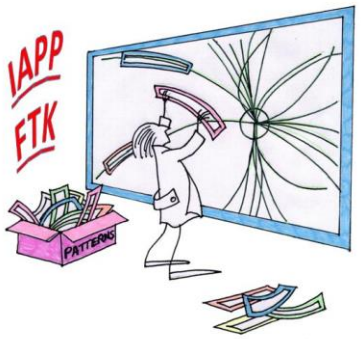
- A high performance FPGA-based implementation
- Uses a moving window technique to identify the clusters
  - Reduces required resources for clustering identification
- Fully generic
  - Detection window size, memory size, bus size
- Can be parallelized
  - Generic number of parallel engines
- For the ATLAS FTK processor: data with an input rate of 40MHz
- Target achieved with a 4x Parallelization
- More exciting things to be explored…
  - Next targets (short term) → IBL, full centroid calculation implementation (charge distribution)
  - Next targets (long term) → Biomedical Applications etc.
- **FTK** → Partial system installation late 2015
  Full detector coverage 2016

# Acknowledgements


- The presented work was supported by Istituto Nazionale di Fisica Nucleare, and European community FP7 funds (Marie Curie IAPP Project 324318 FTK).


- FTK Pixel Clustering Team:
  *C.-L. Sotiropoulou, A. Annovi, M. Beretta, S. Gkaitatzis, K. Kordas, S. Nikolaidis, C. Petridou and G. Volpi*

Thank you!




INFN
Istituto Nazionale
di Fisica Nucleare

SEVENTH FRAMEWORK
PROGRAMME

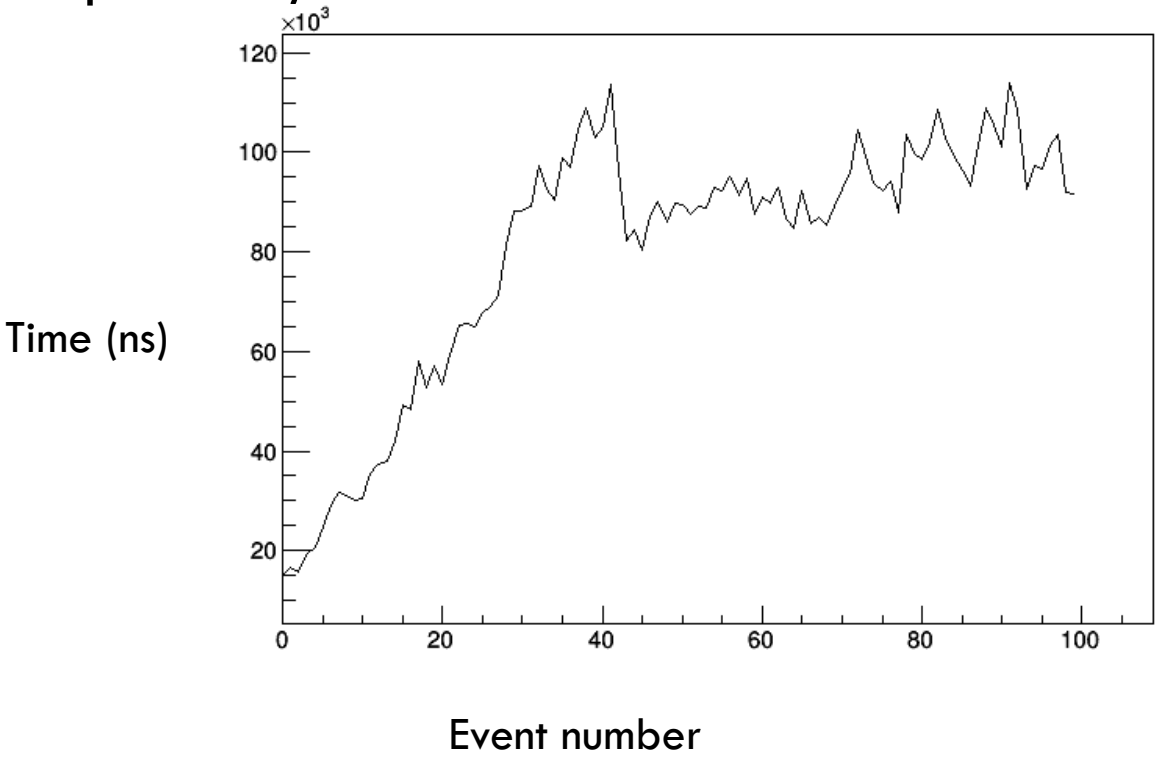
# Back-Up Slides

# Clustering – Testing Results

- Successful integration with the rest of the FTK_IM firmware
- Testing with ZH → nunubb files (80pu)
- Big event files (50 events: 7 modules per event) multiple times – this is what each pixel clustering module will receive from the RODs

# Clustering - Performance Results (Parallel Flow)

- **4 Parallel Engine Flow with Small Buffers**
  Event Processing Latency (end_event output – end_event input time)




End_event input time delayed by backpressure

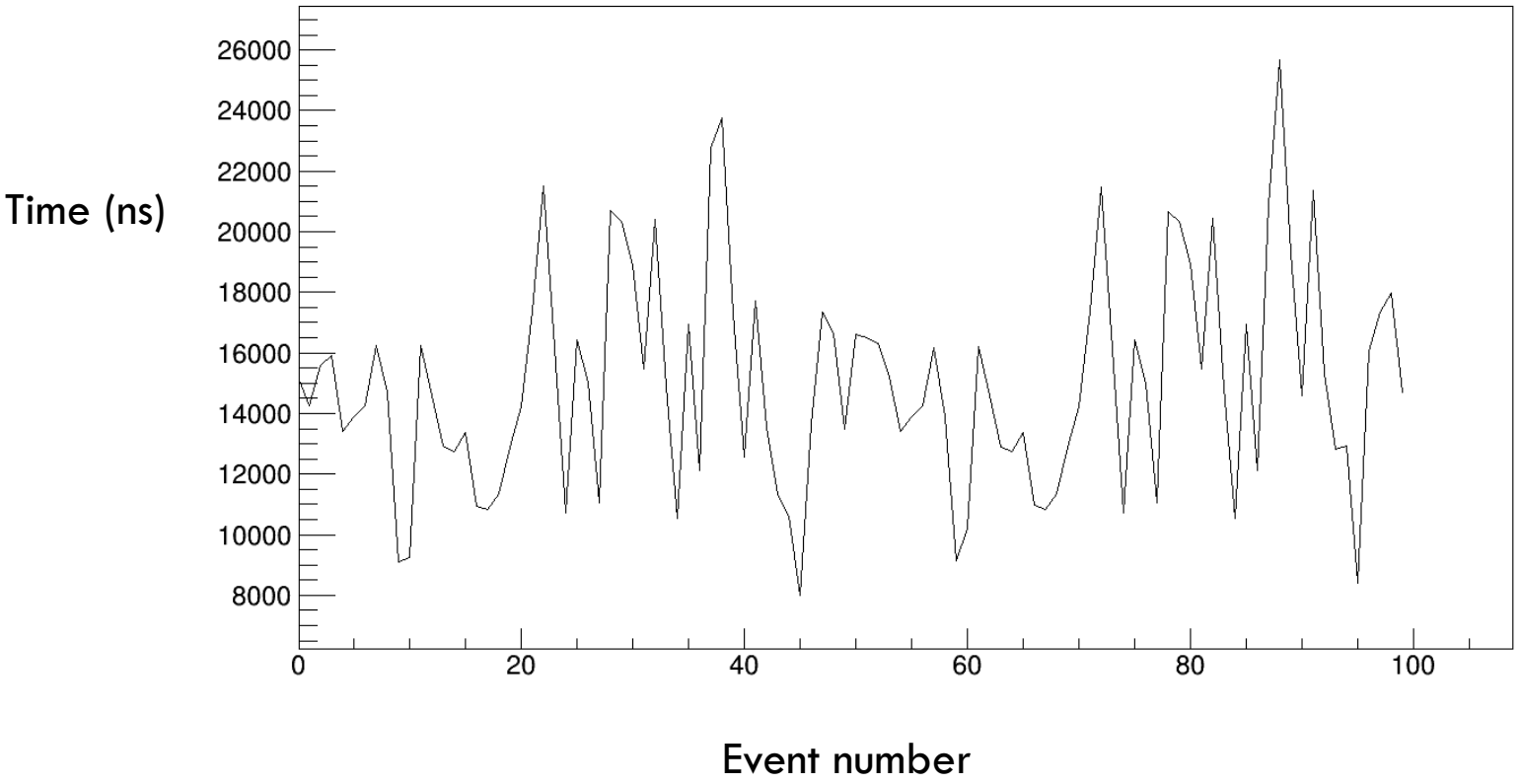
The actual delay is bigger and increasing with event number

# Clustering - Performance Results (Parallel Flow)

- The **4 Parallel Engine Flow with Small Buffers** saturates after 30 events
- This is because of the **backpressure** applied by the Data Merger
- We increased the buffering before the Data Merger to reduce this effect
- Bigger buffering → processing time per word reduced to **25ns** which is the 8 Parallel Engine Flow performance without the extra buffering (next slides)

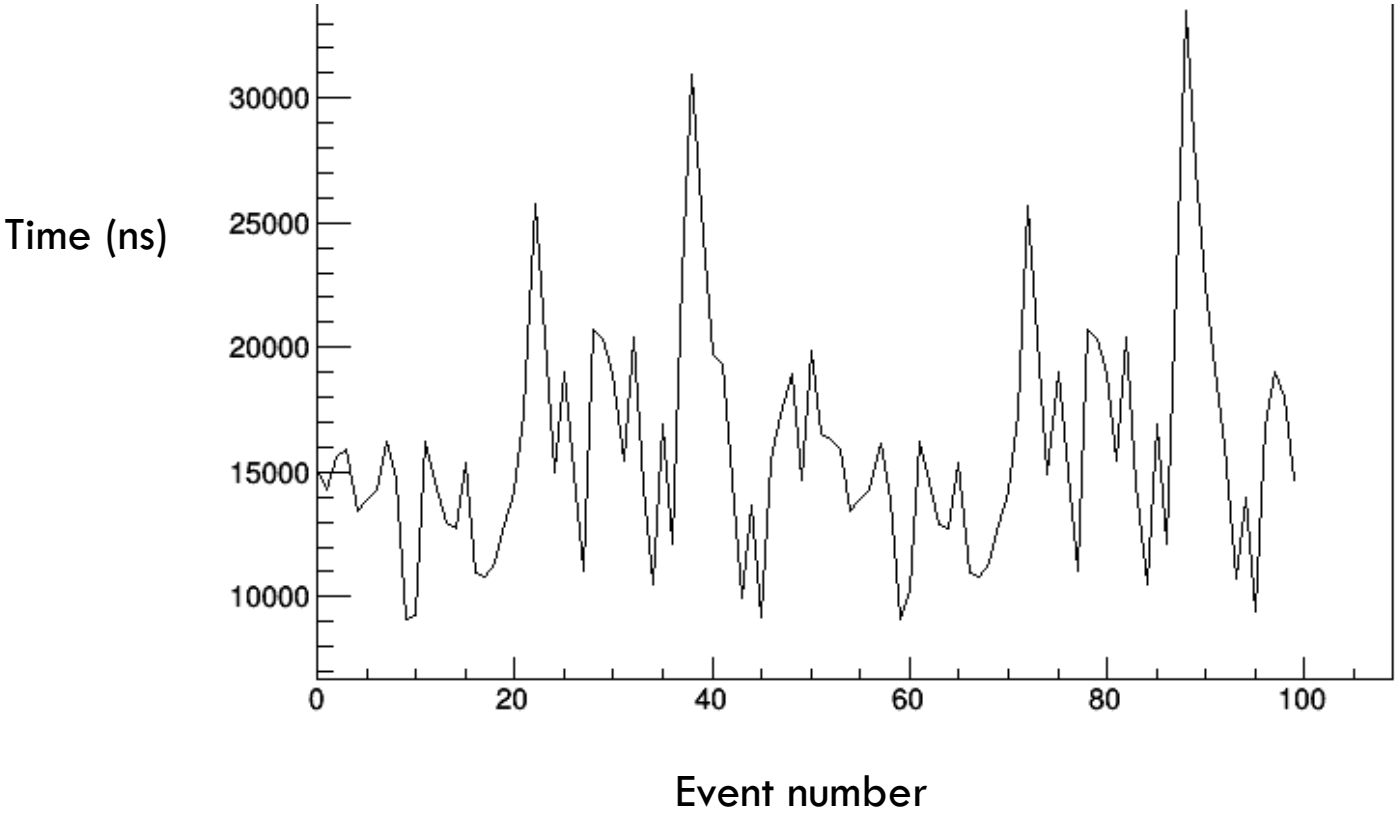
# Clustering - Performance Results (Parallel Flow)

- **4 Parallel Engine Flow with Big Buffers** Event Processing Latency (end_event output – end_event input time)
  ← No saturation and no backpressure seen at the input



# Clustering - Performance Results (Parallel Flow)

- **8 Parallel Engine Flow** Event Processing Latency (end_event output – end_event input time)
  ← No saturation and no backpressure seen at the input



# Clustering - Performance Results (Parallel Flow)

- The **8 Parallel Engine Flow** fully covers pixel specifications for 80pu events even **without the big buffering** (data from 7 pixel modules are easily distributed over 8 engines)

- A **25 ns** processing time was calculated per data word as a worst case

- The **4x Parallel Engine Flow with Big Buffers** is the best solution: Smaller device area occupation, smaller architecture fan out, easier place and route