# Implementation and Performance of FPGA based track fitting for the ATLAS Fast TracKer

Rui Zou, on behalf of the ATLAS Collaboration

University of Chicago

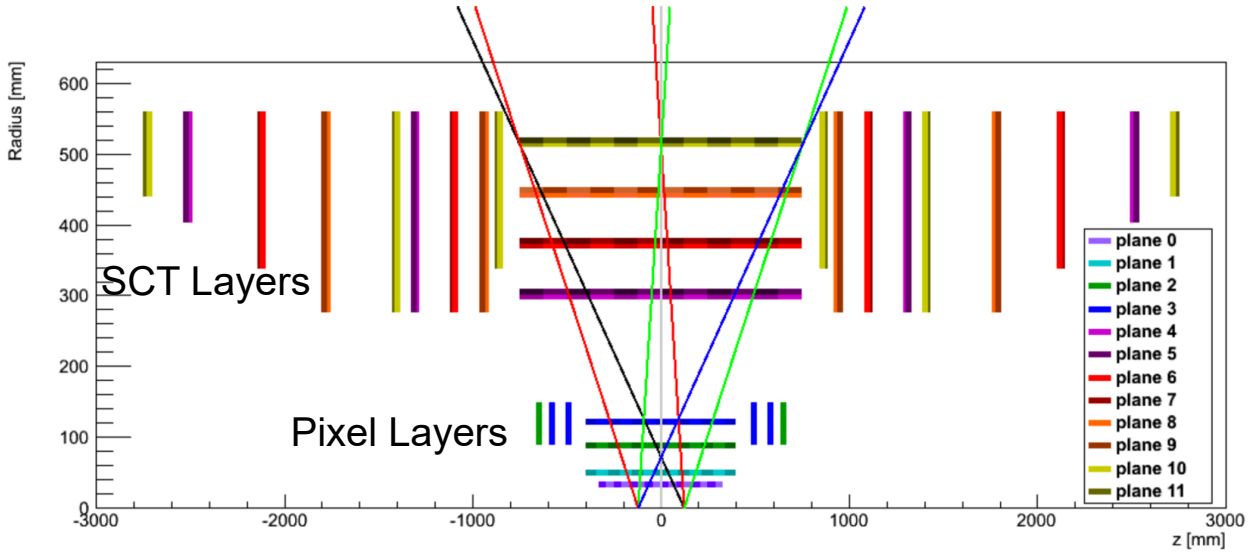
March 21, 2018

# Outline

- Goal and strategy of FastTracker (FTK) in ATLAS
- First stage track fitting of FTK
  - How it works
  - Implementation on FPGA
  - Challenges and strategies
  - Current Performance and Future Plan

# FastTracker (FTK) System in ATLAS

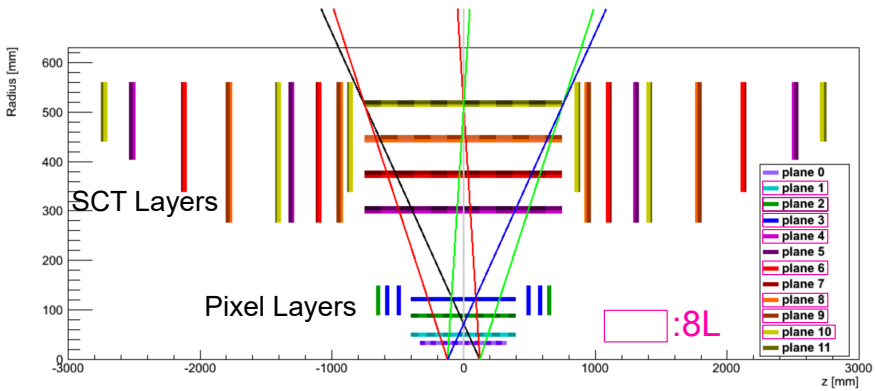
- **A more general FTK talk given yesterday**
- Goal of FTK:
  - Track fitting on 12 layers of hits from ATLAS Inner Detector
  - Total expected latency: 100 $\mu s$
  - Event rate: 100 kHz, thousands of charged particles/event
  - Faster on FPGAs
- Challenges to fit entire detector: massive combinatoric problem
  - Strategy: parallel processing + staging



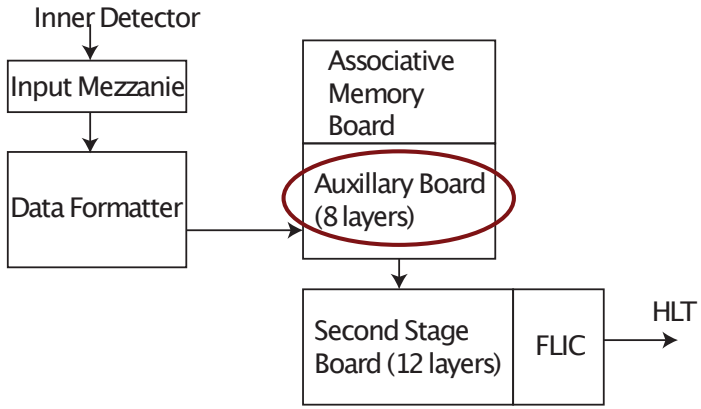

ATLAS inner detector

# Strategy of FTK

- Divide detector into 64 regions for parallel fitting
  - Region small enough that linear approximation is accurate
  - With enough overlap to ensure efficiency
- Pattern recognition + two stages of fitting
- First stage: 8 layers (3 pixel+5 SCT layers)
  - 0.5 trillion linear track fittings/s on Auxillary boards (AUX)
    - Per FPGA: 1 GigaFits(GF)/s
- Second stage: 12 layers linear track fitting



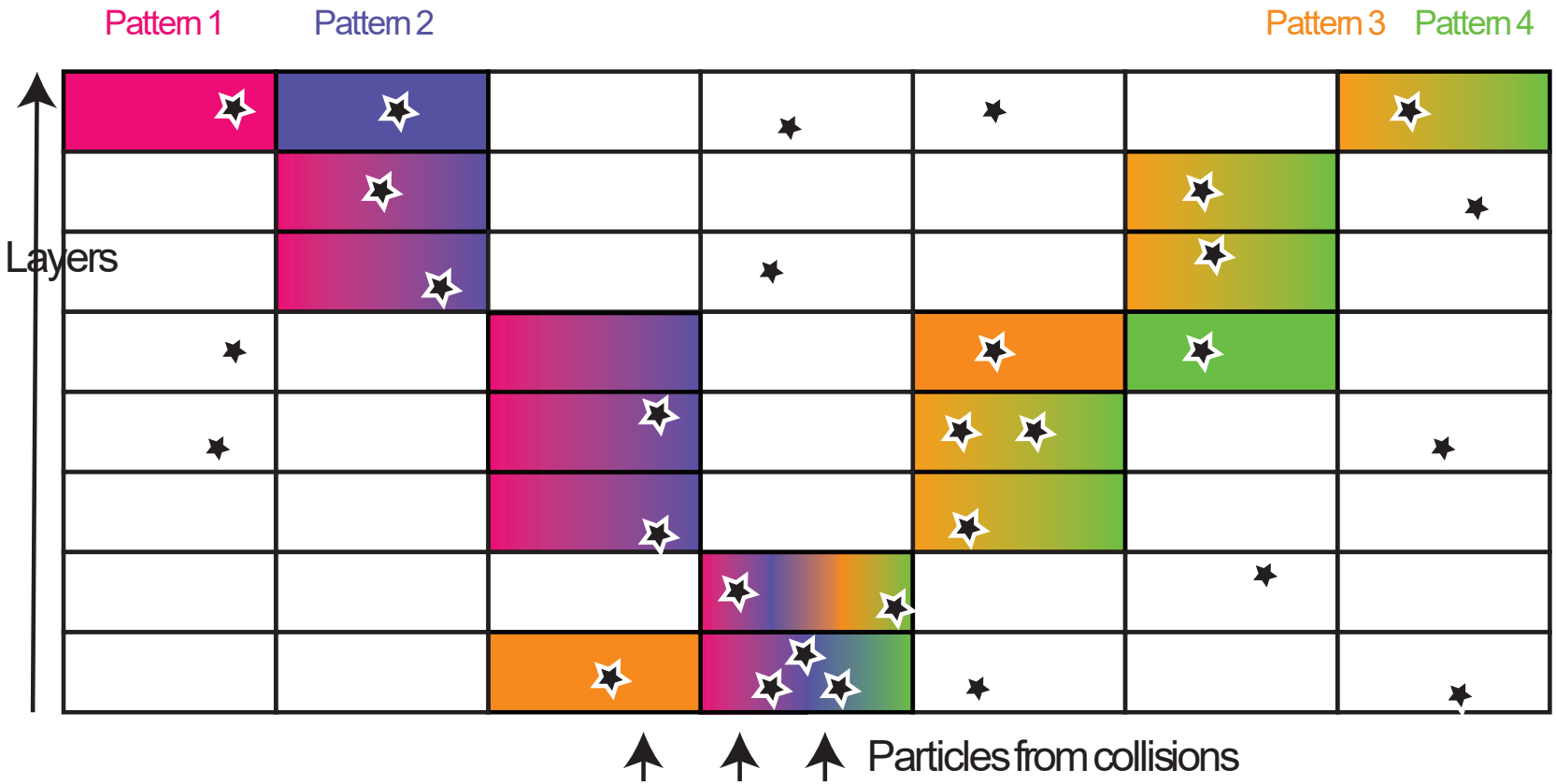

ATLAS inner detector with 8L marked




FTK workflow

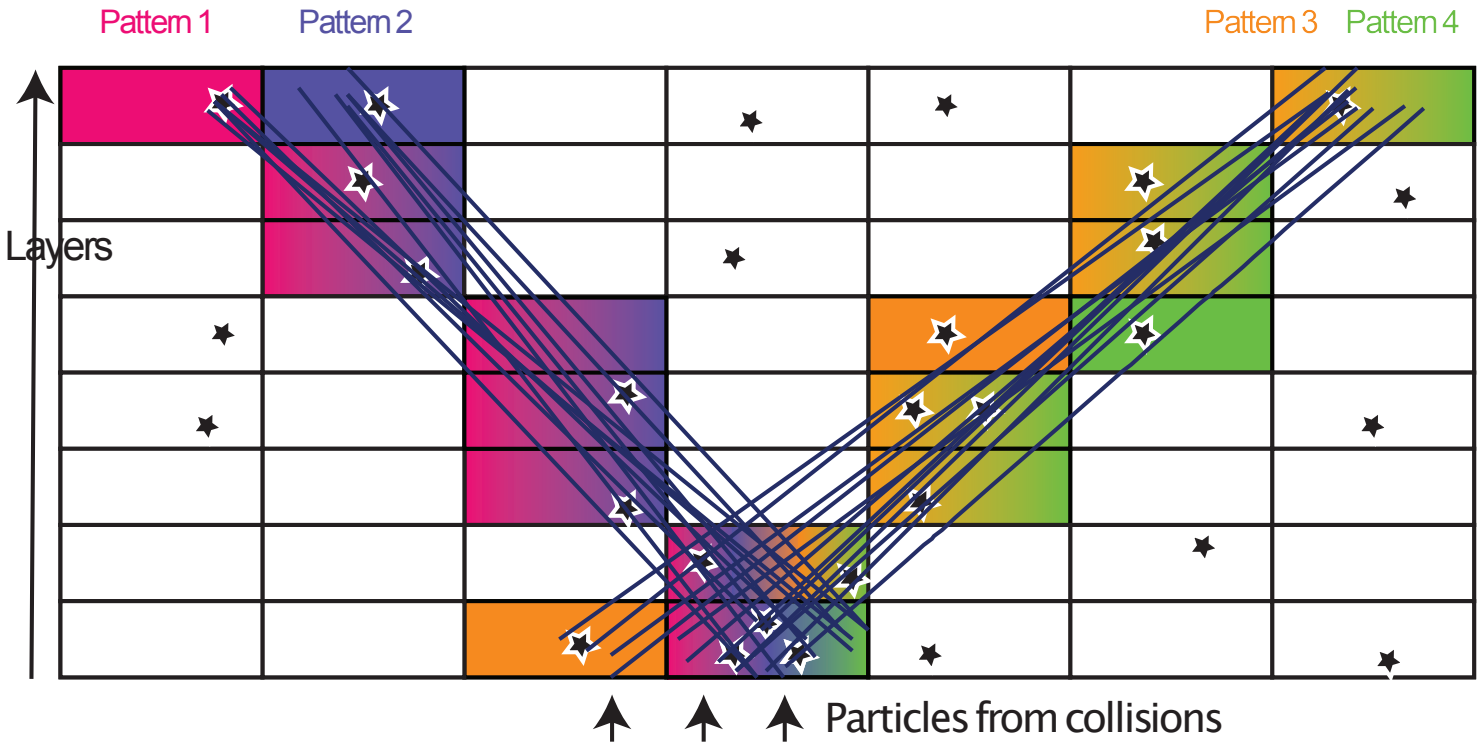
# First Stage Track Fitter

- Pattern recognition board selects all patterns with potential tracks and send to track fitter
  - Majority logic: require at least 7/8 layers match a simulated pattern
- Fit all hit combinations (1 per layer) in each pattern:
  - Fit at least 7/8 layers of hits
  - Pattern 1: 6, Pattern 2: 6, Pattern 3: 4, Pattern 4: 12



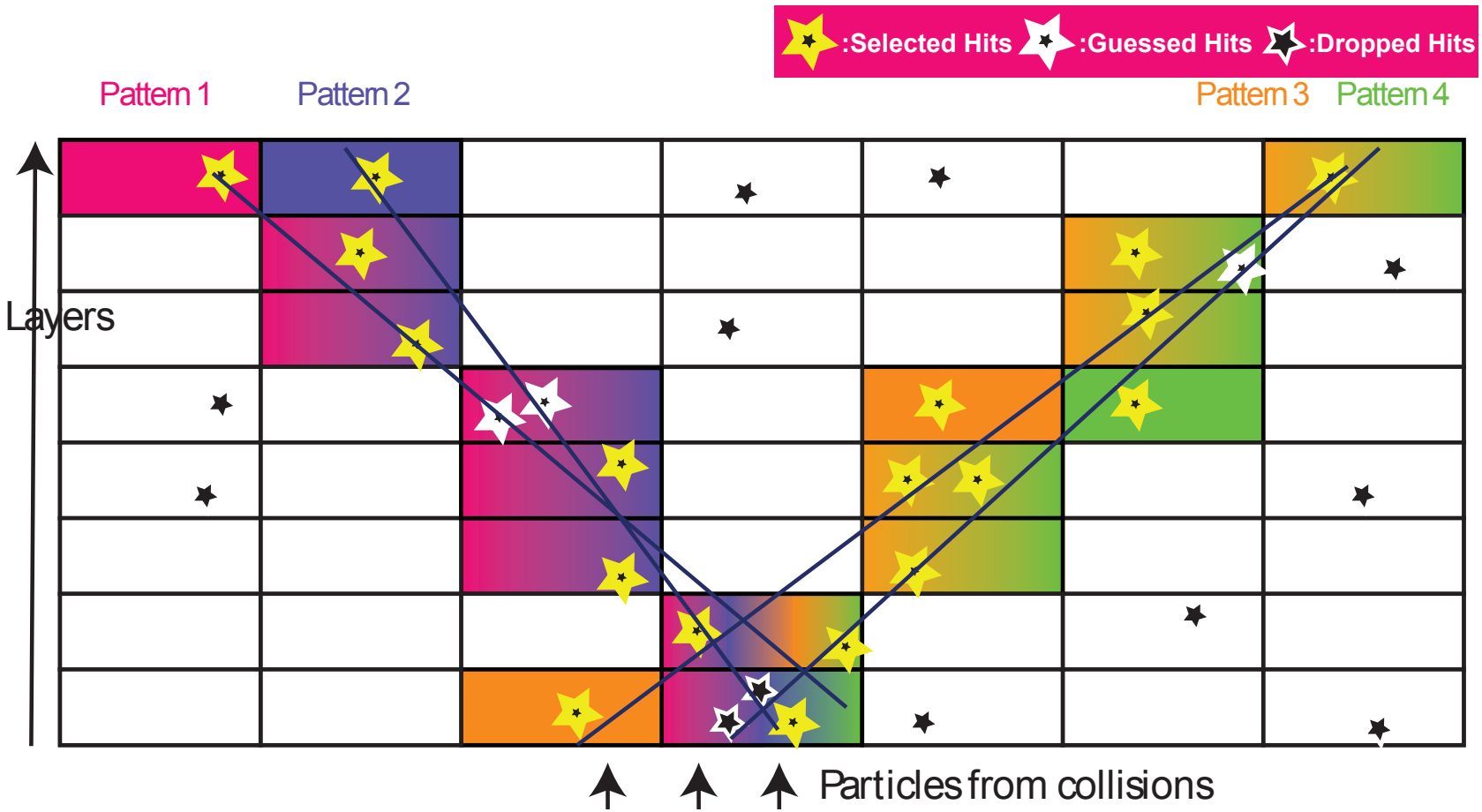
# First Stage Track Fitter

- Pre-simulated constants stored in RAMs for calculations
  - 4096 sets of constants per FPGA
- Calculate $\chi^2$ for all the hit combinations for first 8 layers
  - $\chi^2 = \sum_i^{N_\chi} (h_i + \sum_j^{N_{\text{hits}}} S_{ij} \times \text{coord}_j)^2$, $h_i$, $S_{ij}$ constants
- Calculate the best possible hit for layer with missing hit
  - $x_j = C_{jj}^{-1} t_j$, where $C_{jj}^{-1} \xleftarrow{\text{offline}} S_{jj}$, $t_j \xleftarrow{\text{on chip}} S_{ij} + \text{coords}$



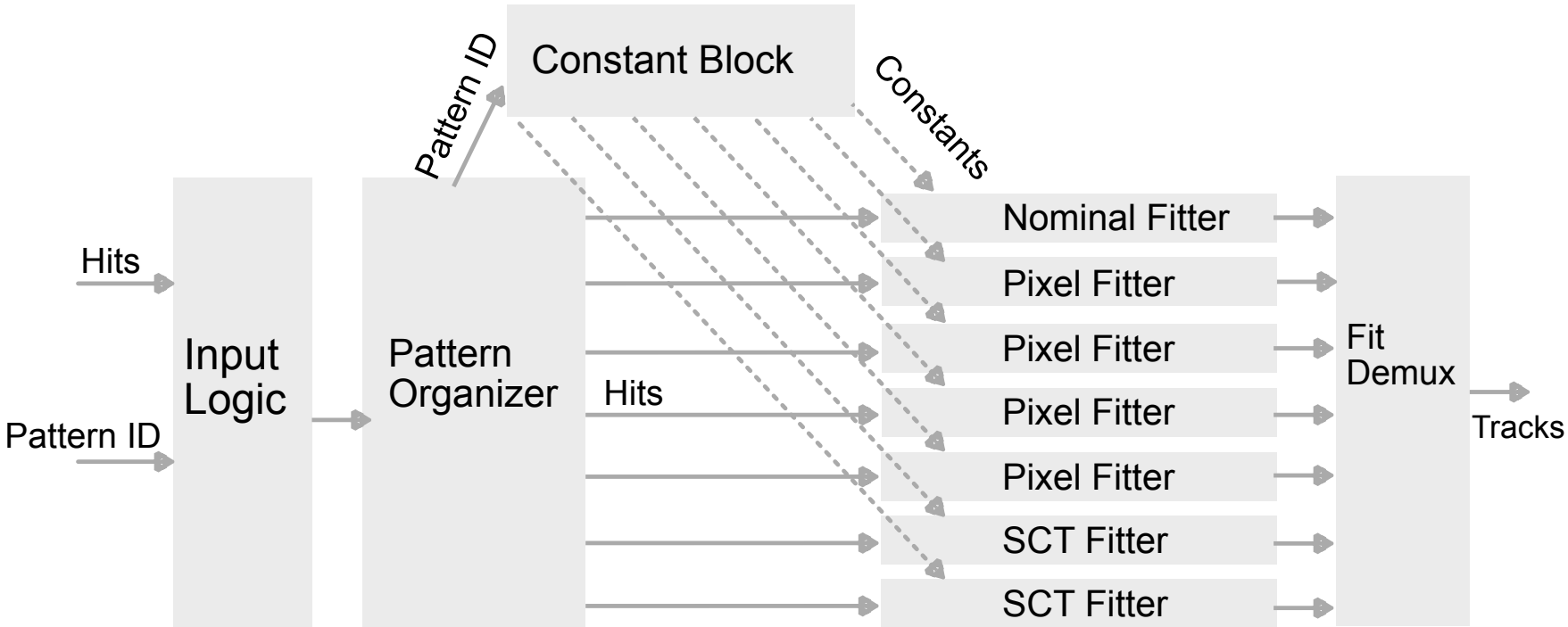
# First Stage Track Fitter

- Only the best $\chi^2$ track is kept if multiple tracks share at least 6 hits



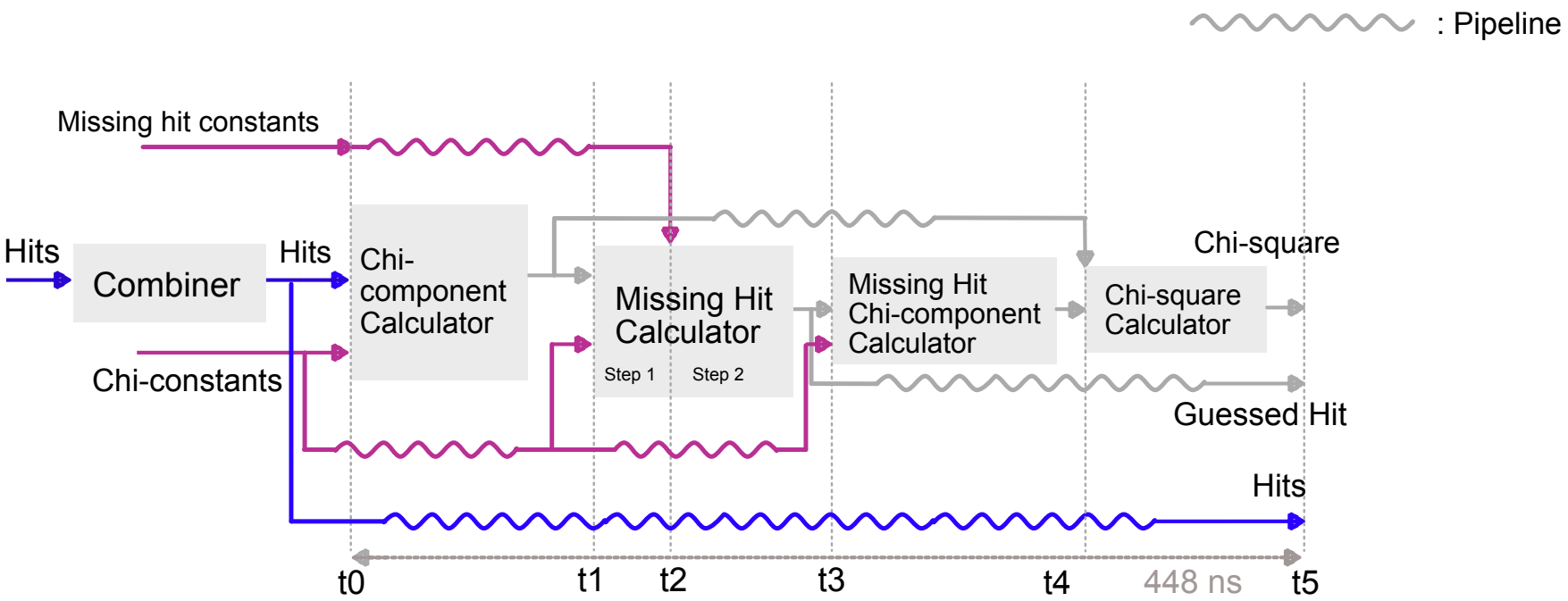
# Implementation Of Track Fitter In Firmware

- Synchronizes hits + pattern ID from pattern recognition board (Input)
- Assigns the to-be-fit pattern to right fitter (Pattern Organizer)
- Sends pre-simulated constants to right fitter (Constant block)
- 7 parallel fitters (ratio of fitter type optimized on simulation studies)



# Fitter Algorithm

- Pixel fitter shown as an example
- Lots of pipelines between calculators
  - All input variables arrive at each calculator in the same time
  - No buffer during the massive calculations
    - Ouput rate = input rate
  - Ensures ability to fit a new track every 7 ns

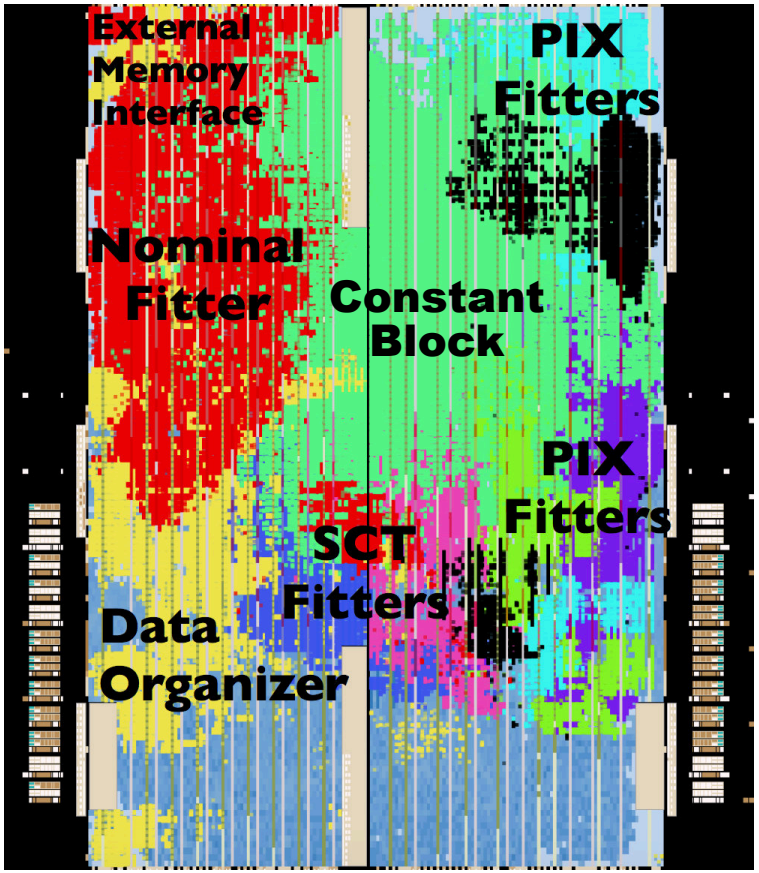


# Firmware Optimizations

- Balance among resources, fit rate, and efficiency
  - Smaller resource usage $\rightarrow$ faster processing rate
  - Faster firmware algorithm for fit rate $\rightarrow$ bigger resource usage
  - Better efficiency $\rightarrow$ more resource usage
  - Faster fit rate $\rightarrow$ better efficiency
- Dedicated effort has gone into this

# Reducing Resource Usages

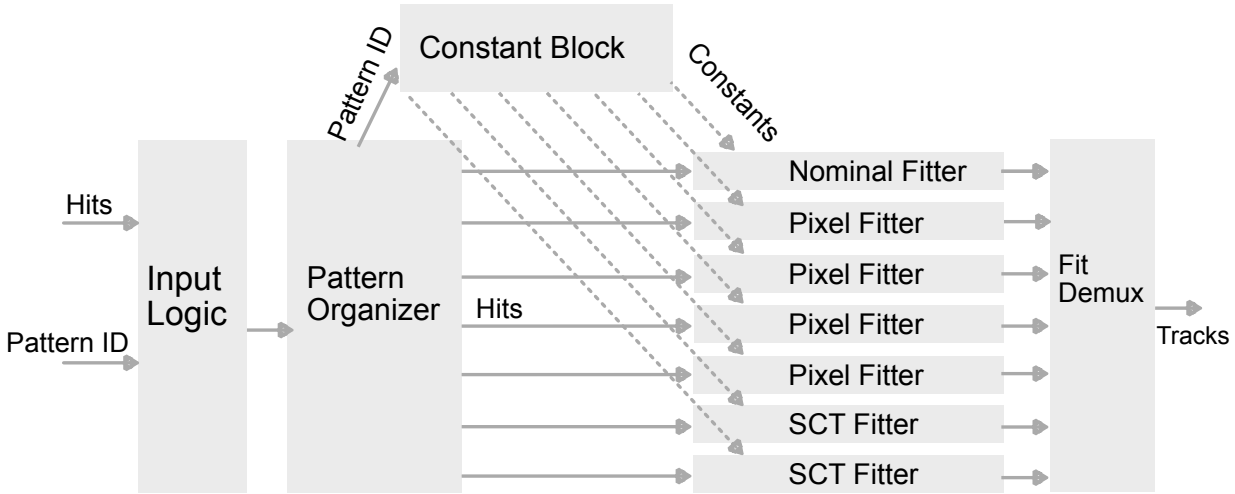
- Smaller resource usage→faster processing rate
  - Current: 140 MHz(7ns)
  - Goal: 200MHz(5ns)
- Compiler places fitters around constants
- Fits other components in left over space
- Routing is the biggest challenge:
  - Fan out of constants bus despite staging and multi cycling
  - 1200 bits fan out to each fitter (optimized for efficiency)




Physical location of each component on one FPGA

# Improving Fit Rate

- Two ways: better algorithm or faster processing rate
- 500 fits per pattern maximum to avoid long event latency
- Firmware components developed to output fastest possible:
  - Input Logic, Pattern Organizer: output a hit in each layer every 7ns ✓
  - Constant Block: retrieve a new set of constants every 14ns ✓
  - Combiner: output a new combination every 7ns
- Coming up with better strategy over time:
  - Group patterns with the same set of constants together
    - Less frequent retrieval of constant
  - Clever usage of available fitters

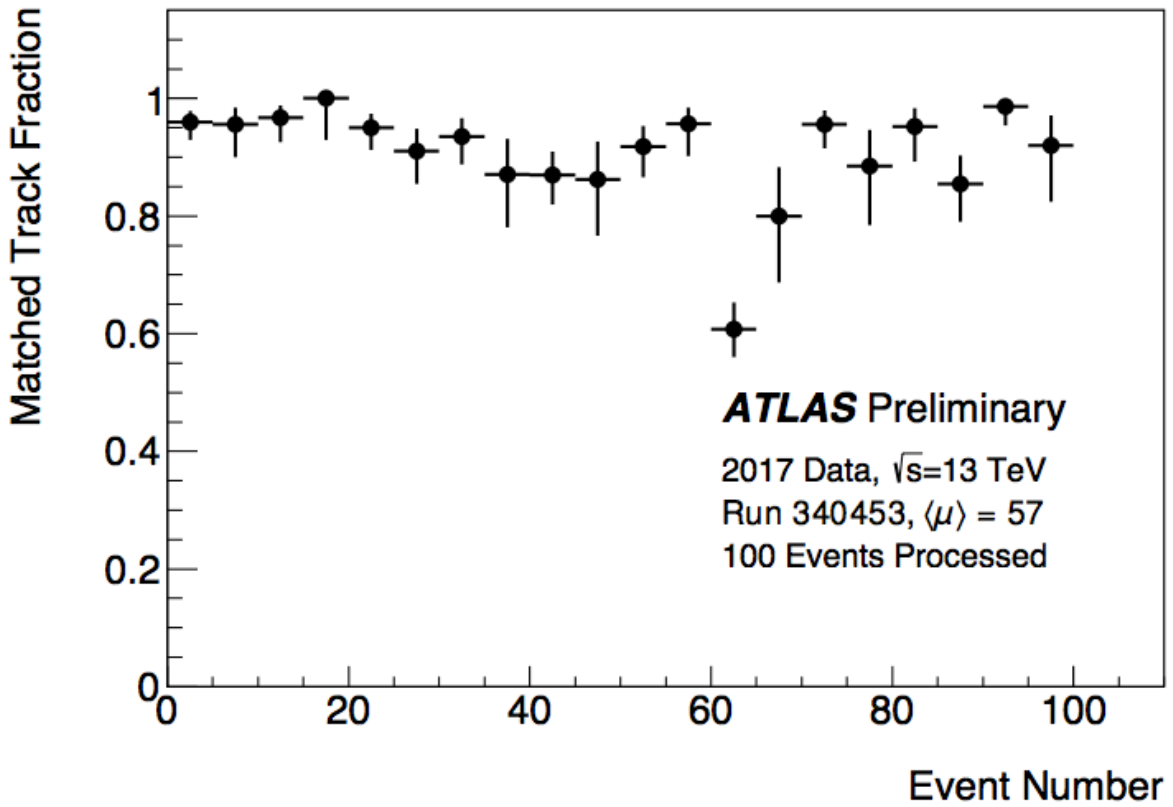


# Optimizing Efficiency

- Efficiency: with respect to ATLAS offline reconstructed track
- Better efficiency requires more precision in calculations
- Careful software studies done to determine precisions of the intermediate products and pre-simulated constants
- Constants are non integer values of a wide range
  - Left shift the constants until absolute value right below 4096 then round to integer
    - Retaining the most significant bits of smaller constants

# Preliminary Results

- FTK ran with ATLAS late 2017 up to first stage fitting
  - With only one first stage board out of 128
- Recorded board output and ran the same input data in simulation
- Firmware largely behaved as expected with issues being investigated
  - Need to run with more data this year



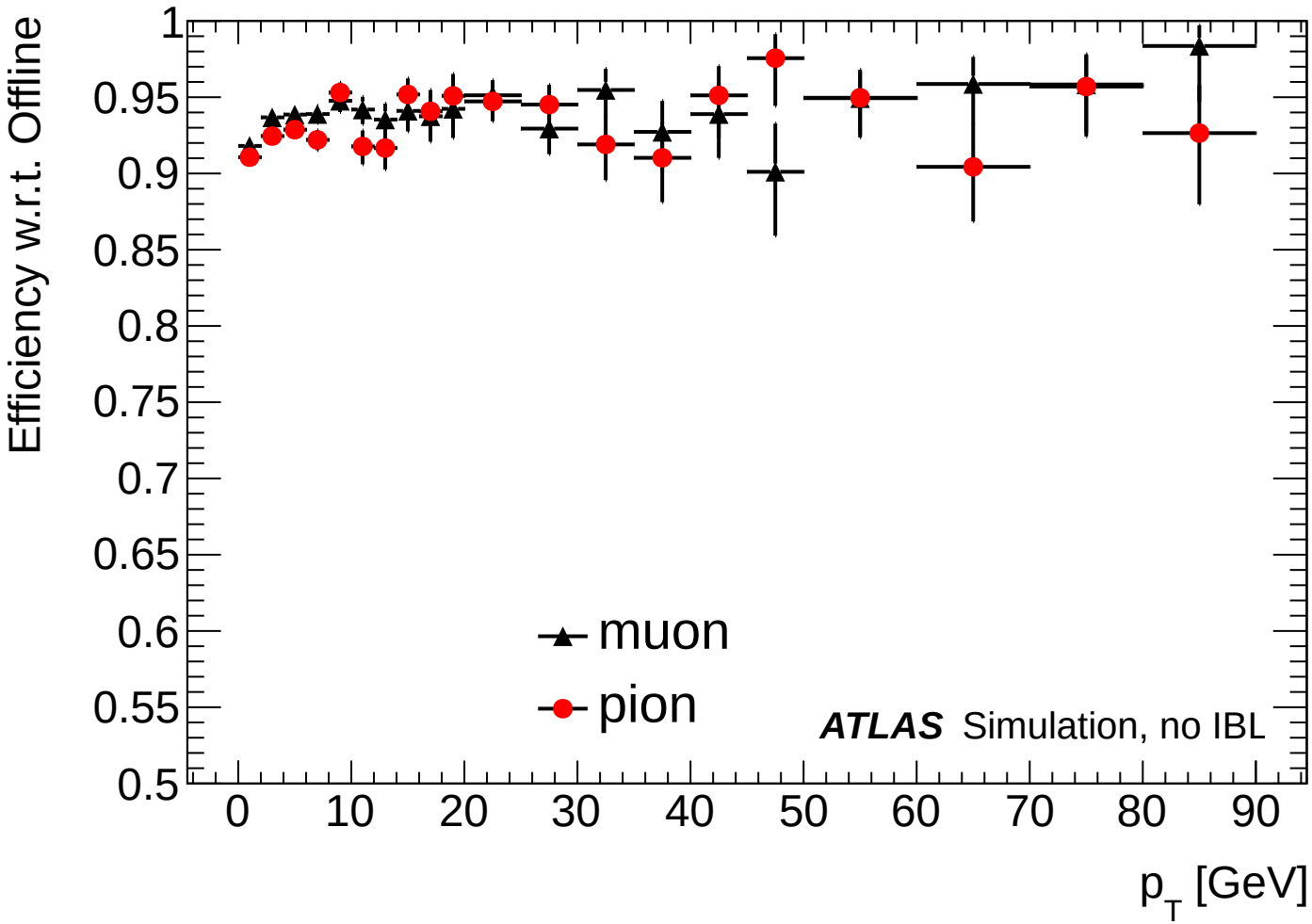
# Status And Plan

- Current fit rate: 0.5 GF/s
- Future Plan
  - Focus on testing running FTK in ATLAS this year
    - Protections against errors in the input data
  - Build software that simulates the firmware processing time
    - Performance optimization
  - Improve inefficiencies in firmware algorithm

# Backup Slides

# Expected Performance Of FTK

- Efficiency: with respect to offline in muon and pion samples
  - 12 layer tracks




Efficiency vs transverse momentum

# Current Resource Usage

- Resource usage glossary:
  - **ALM**: Adaptive Logic Module
  - **M10k**: 10kb dual port block memory
  - **DSP**: Digital Signal Processing blocks, 2x 18x19 multipliers

| Processor - Arria V: 5AGXFB7H4F35C4 | | |
|---|---|---|
| Resource | Usage | % |
| ALM | 168k | 66 |
| M10k | 1.7k | 71 |
| DSP | 617 | 53 |
| Ave. Interconnect | - | 45 |
| Peak Interconnect | - | 78 |