# SiW-ECAL 2018 CERN Beam Test:

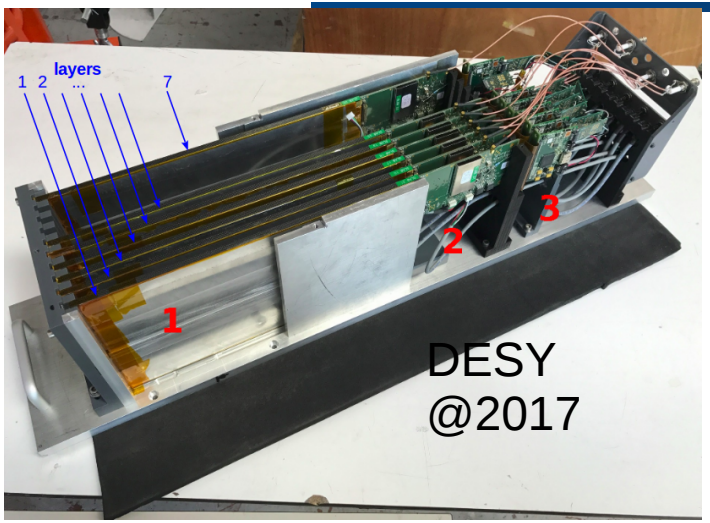
## beam test summary

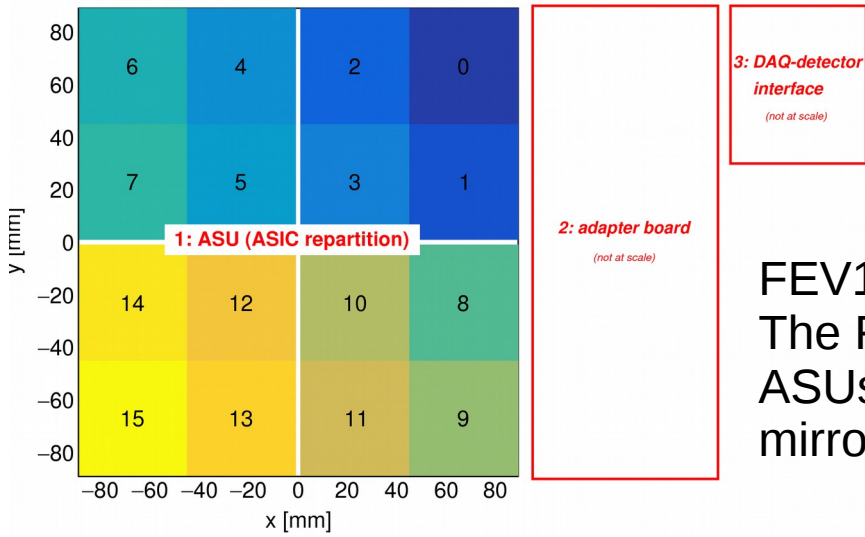
A. Irles, LAL-CNRS/IN2P3

19th December 2018

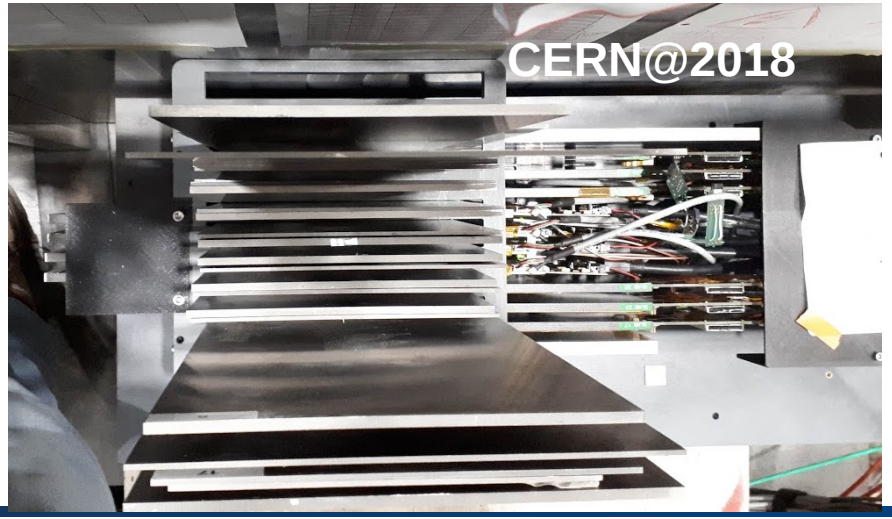
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 654168

# Slab mapping



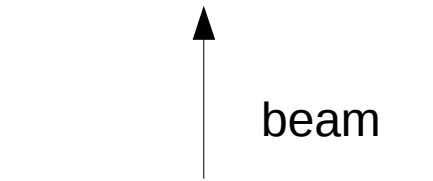




FEV11,
The FEV13 ASUs are mirrored in Y



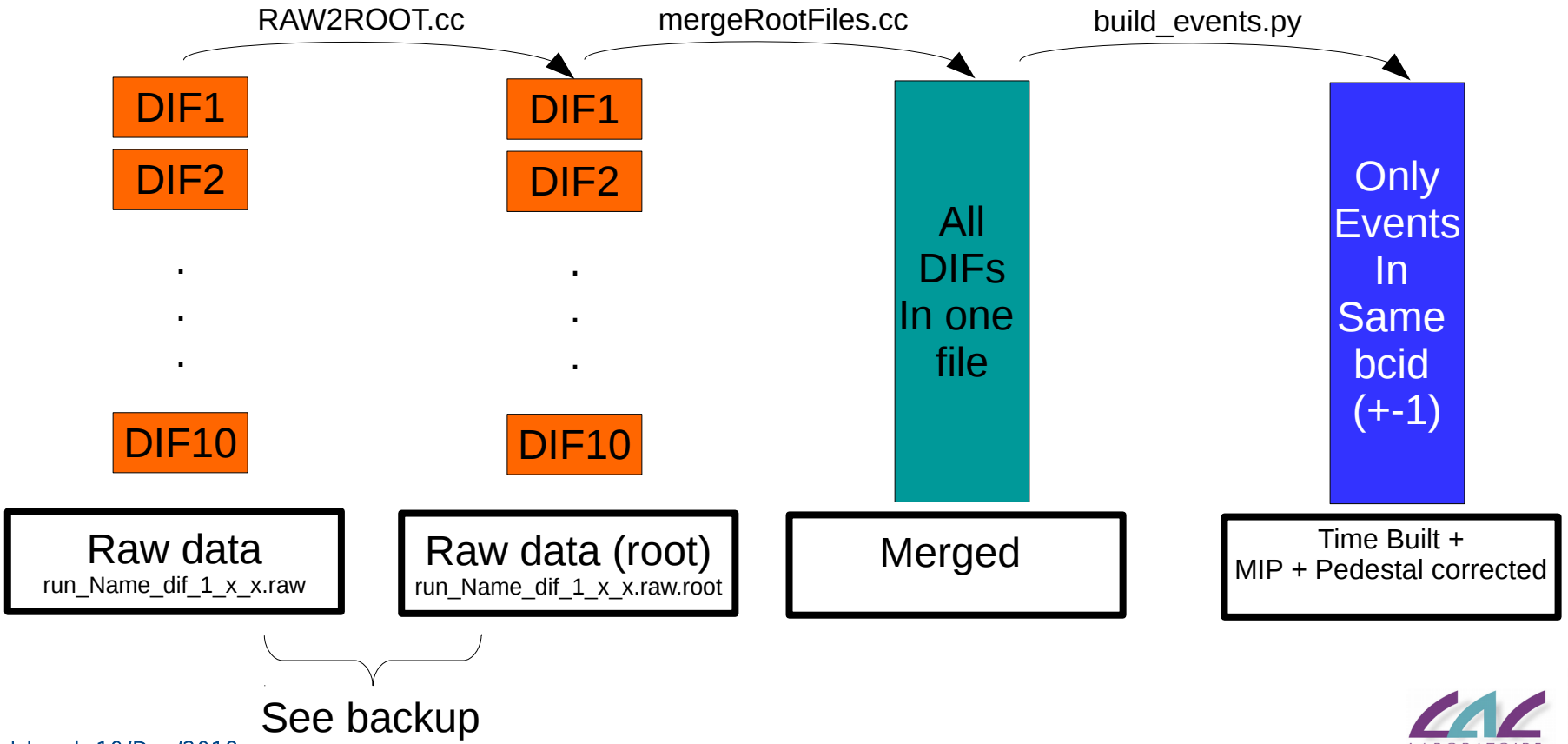
# ECAL setup

| | | | |
|---|---|---|---|
| dif_1_1_1 | FEV11, slab 16 | FW_v0: 2.5MHz, offset (in bcid) = 0 (±1) | bcid x 2 to get the equivalent of 5MHz |
| dif_1_1_4 | FEV11, slab 19 | FW_v0: 2.5MHz, offset (in bcid) = 0 (±1) | bcid x 2 to get the equivalent of 5MHz |
| dif_1_1_3 | FEV11, slab 18 | FW_v0: 2.5MHz, offset (in bcid) = 0 (±1) | bcid x 2 to get the equivalent of 5MHz |
| dif_1_2_3 | FEV13-Jp, K2 | FW_v2: 5MHz, offset (in bcid) = 2260 (±1) | |
| dif_1_2_4 | FEV13-Jp, P3 | FW_v2: 5MHz, offset (in bcid) = 2260 (±1) | |
| dif_1_2_5 | FEV13-Jp, P2 | FW_v2: 5MHz, offset (in bcid) = 2260 (±1) | |
| dif_1_1_5 | FEV13-Jp, K1 | FW_v2: 5MHz, offset (in bcid) = 2260 (±1) | |
| dif_1_2_1 | FEV11, slab 20 | FW_v0: 2.5MHz, offset (in bcid) = 0 (±1) | bcid x 2 to get the equivalent of 5MHz |
| dif_1_2_2 | FEV11, slab 22 | FW_v0: 2.5MHz, offset (in bcid) = 0 (±1) | bcid x 2 to get the equivalent of 5MHz |
| dif_1_1_2 | FEV11, slab 17 | FW_v1: 5MHz, offset (in bcid) = -8 (±1), | |



# Data Format + software

- All software is based in the BT-software developped during last 2 years (based on previous software)
- https://github.com/SiWECAL-TestBeam/SiWECAL-TB-analysis/ → Branch TB201809_10slabs



# Data Format + software

All DIFs In one file

Merged

```
int bcid[NSLABS][NCHIP][MEMDEPTH];
int badbcid[NSLABS][NCHIP][MEMDEPTH];
int charge_low[NSLABS][NCHIP][MEMDEPTH][NCHANNELS];
int charge_high[NSLABS][NCHIP][MEMDEPTH][NCHANNELS];
int gain_hit_low[NSLABS][NCHIP][MEMDEPTH][NCHANNELS];
int gain_hit_high[NSLABS][NCHIP][MEMDEPTH][NCHANNELS];
int numCol[NSLABS][NCHIP];
int chipID[NSLABS][NCHIP];
int acqNumber;
int corrected_bcid[NSLABS][NCHIP][MEMDEPTH];
int nhits[NSLABS][NCHIP][MEMDEPTH];
```

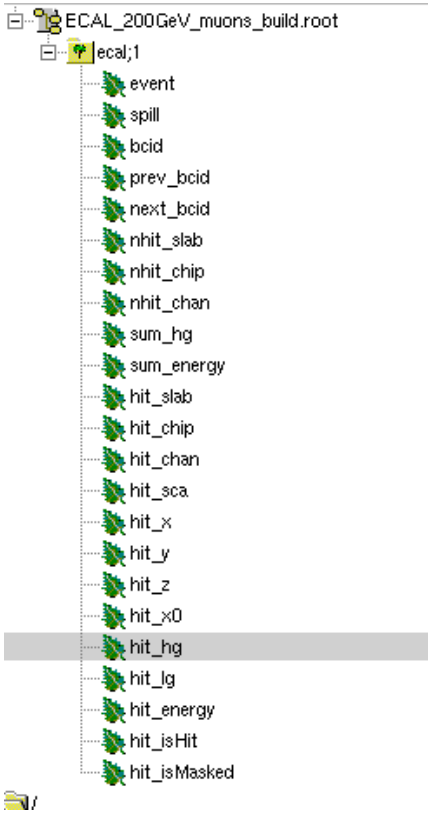
- Bcid = bcid that corresponds to 5MHz
  - The overrunning of the bcid counter (12 bits) is accounted. One loop of 12Bits at 5MHz is 0.819ms and we were open during 25ms.
- Badbcid == 0 if the event is not a retrigger

# Data Format + software

- See the examples attached in the indico agenda.
- A dummy MIP calibration and pedestal calibration is applied for slabs 1_1_5, 1_2_4 and 1_2_3 since they are not calibrated yet (the data are not yet in the cern folder)

Only Events In Same bcid (+-1)

Time Built + MIP + Pedestal corrected



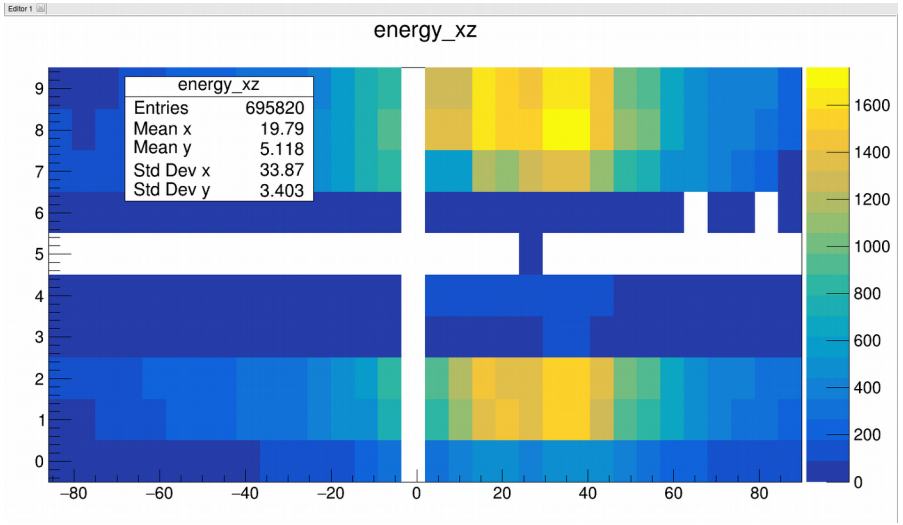

- Only hits are saved (hit_isHit is always 1). Retriggers are filtered.
- Each event has nhit_chan cells triggered.
- The hit_energy is pedestal subtracted and MIP calibrated.
- The hit_hg is only pedestal subtracted.
- (x,y)=(0,0) in the center of the detector
- (x,y)=(+max,+max) as seen from the beam pipe.
  - Attention! FEV13 are mirrored in y
  - see back up for schematic picture

# Common + last standalone muon runs

- Location of the converted + standalone event built data /eos/project/s/siw-ecal/TB2018-09/Common/ECAL
- Scripts for conversion in /eos/project/s/siw-ecal/TB2018-09/converter
  - Main script: *build_script.sh*
  - Script with the selection of runs from the e-log: *launch_build.sh*
  - Instructions and comments are in the script and in the README in the github (for the use of the root building event script).
- Still some data are to be copied:
  - Common electron runs
  - Last muon runs for calibration of the 3-4 FeV13s

# Common + last standalone muon runs

- Full **common muon run** hit map (x vs z). Only ECAL data.
- Selection: nslabs_with_hit≥3

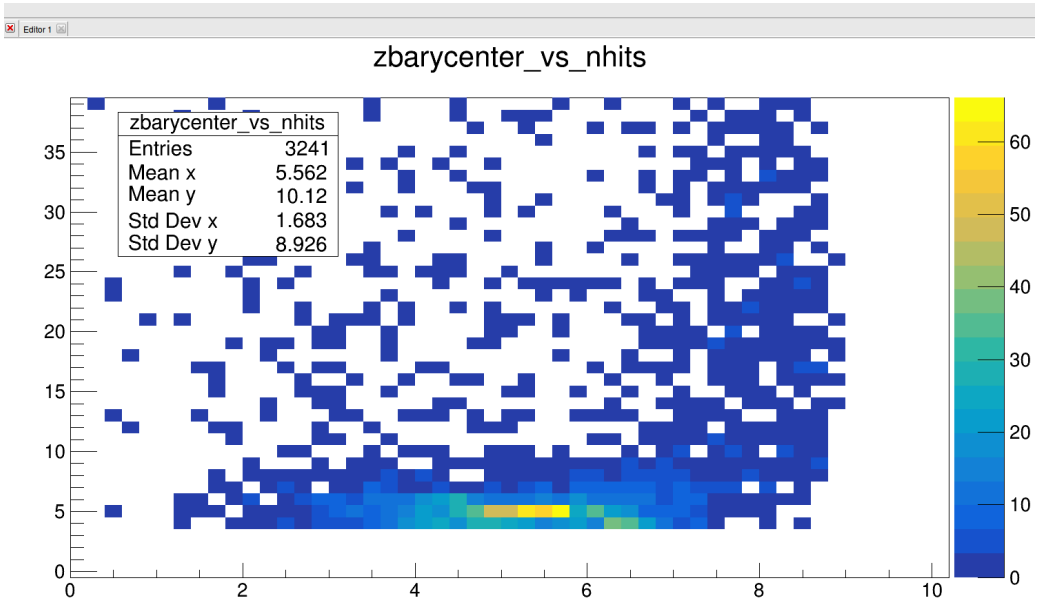



- Some optimization of the event building + offsets management may be needed
- But ...
- If the offsets event building is the issue... the selection will still accept two groups of events:
  - the events where the FEV11 are synchronized and the events where the 4 central slabs are synchronized

# Common + last standalone muon runs

- Selection: nslabs_with_hit≥3
- Plot for PiPlus_50GeV

| Common runs (selection = nslabs with hit >3) | |
|---|---|
| **run** | **events (offsets elog)** |
| PiPlus_40GeV | 28299 |
| PiPlus_50GeV | 3241 |
| PiPlus_60GeV | 2365 |
| PiPlus_70GeV | 12727 |
| PiPlus_80GeV | 5484 |
| Muon_200GeV | 108729 |
| Electron 150 GeV | not copied to the cern eos |



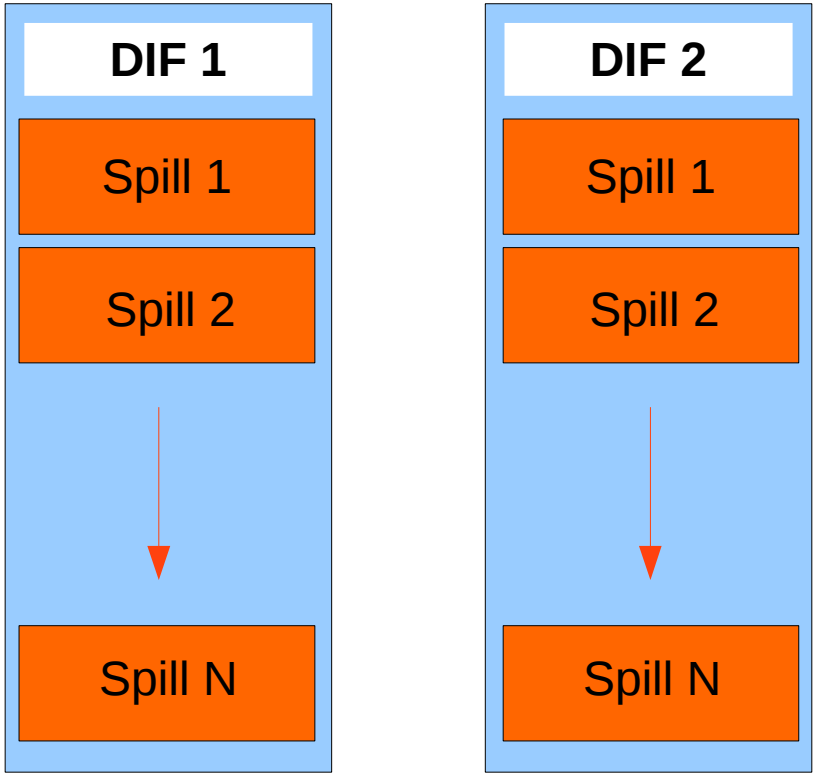

- The selection is very loose, a proper selection may easily apply a substantial reduction
  - And there is the issue with the middle slabs…

- Details on the raw data format

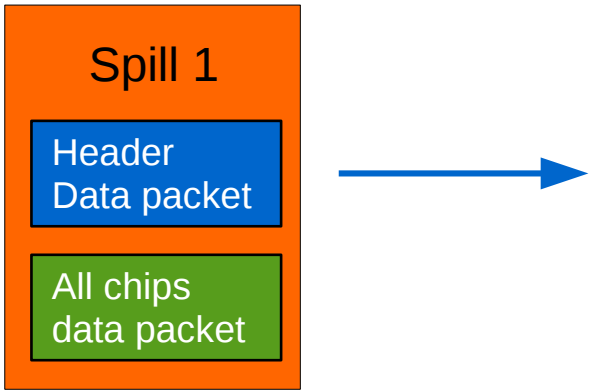
# Raw data output format

Pyrame and/or the DIF fw are introducing small changes into the SKIROC data format.

- The data is grouped in block of spills with all chip info inside
  - → common to all data files
- Spill number is increased by a counter (GDCC, DIF firmware, Pyrame?) and it is reset when a new configuration of the detector is done.
- The output are saved in independents data files (one per DIF)



# Raw data output format




0xfffc → header tag
0x0
0x1
0x5053 → footer tag
0x4c49 → footer tag
0x2020 → footer tag

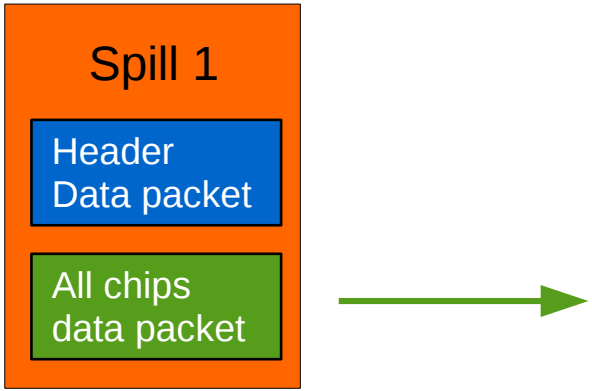
We use the header and footer tags to identify the data packet as spill info packet.

We rextract the spill number as:

packetData[packetData.size()-5]*65536+packetData[packetData.size()-4]

The packet has variable length… why?

# Raw data output format




0xfffd → beginning of chip block
0xff01 → block ID (reset every spill)
0x4843 → header tag (why 4843?)
0x5049 → header tag (why 5049?)
0x2020 → header tag (why 2020?)
SKIROC-Data (contains the chip ID)
0xfffe → end of block chip block
0xff01 → block ID (reset every spill)
0x2020 → footer tag
0x2020 → footer tag

1

0xfffd
0xff02
0x4843
0x5049
0x2020
SKIROC-Data
0xfffe
0xff02
0x2020
0x2020

2

0xfffd
0xff10
0x4843
0x5049
0x2020
SKIROC-Data
0xfffe
0xff10
0x2020
0x2020
0xffff → end of spill block

last

# Raw data output format

Spill 1

Header
Data packet

All chips
data packet

SKIROC-Data

1

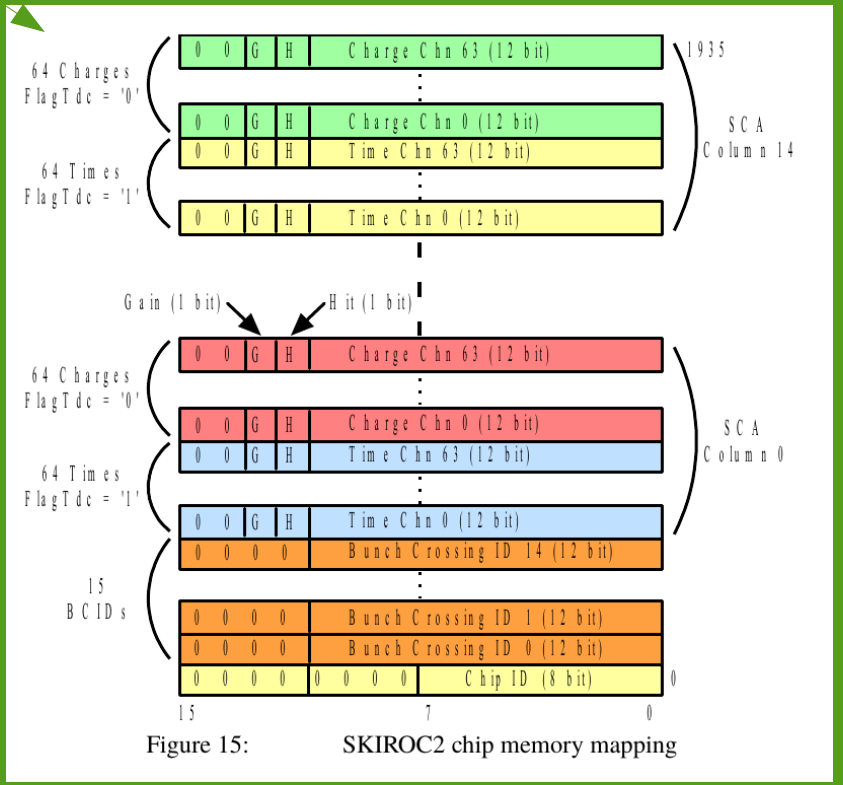



Figure 15: SKIROC2 chip memory mapping

# Raw to ROOT decoder

- The ROOT decoder uses the tags to identify the different data packets types.
- If a spill packet is found, the spill is decoded and saved only if the next packet is a data packet.
- When a data packet is found (after a spill packet), the length is checked
  - It has to be comaptible with the number of chips in the ASU (can be less but not more!)
  - It has to be compatible with the SKIROC data structure: chip ID, a maximum of 15 SCA high gain + low gain (or  auto gain + TDC), 15 bcids
- The data is converted to ROOT format without any event building (nor time, nor merge of DIFs files)
  - The event building is done afterwards, if needed.