SOFTWARE DEVELOPMENT
for the QA/QC and the DCS of the new ATLAS pixel module prototypes

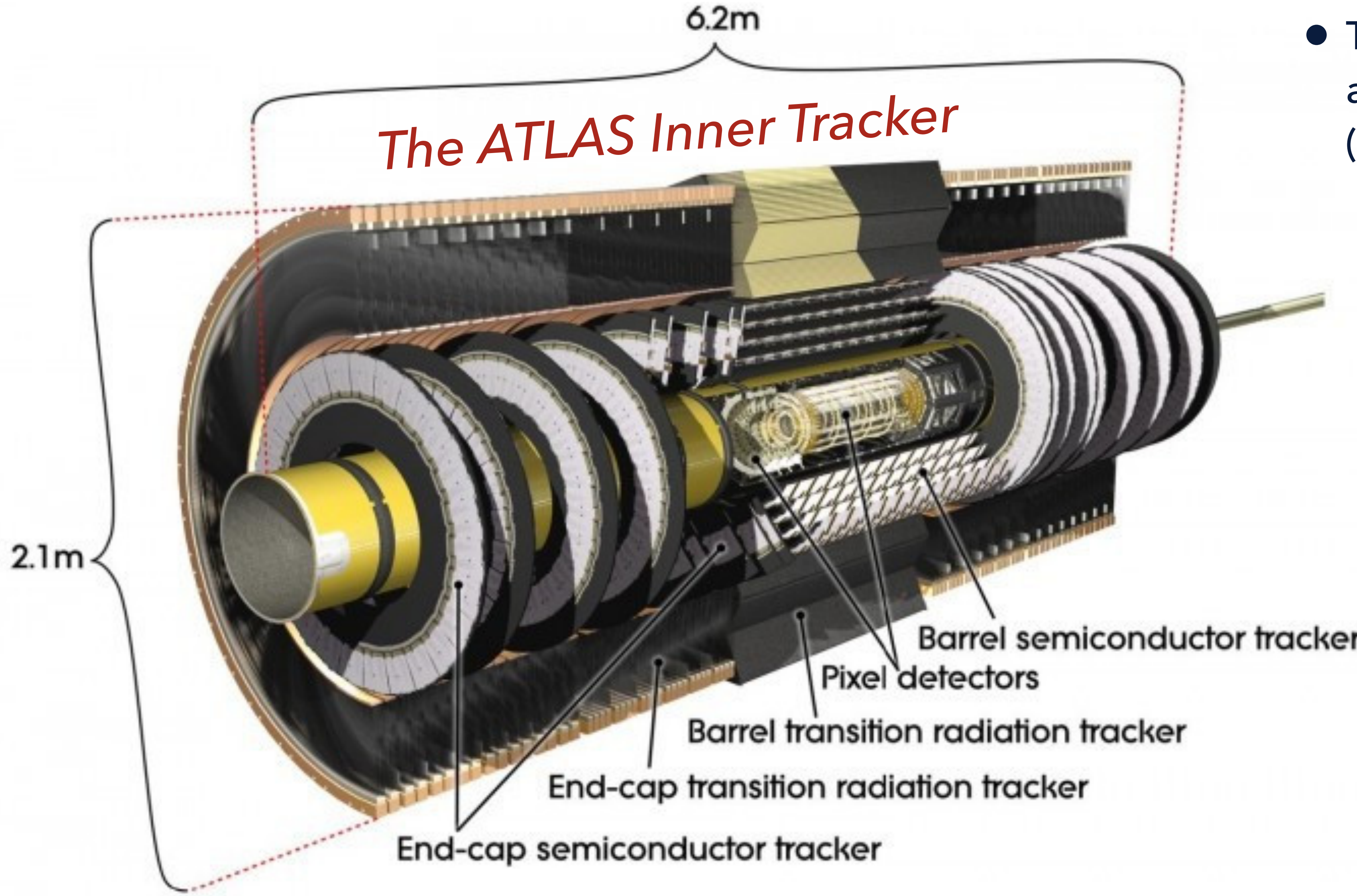
Mario Gonzalez, Yamanaka Lab

23/12/2019

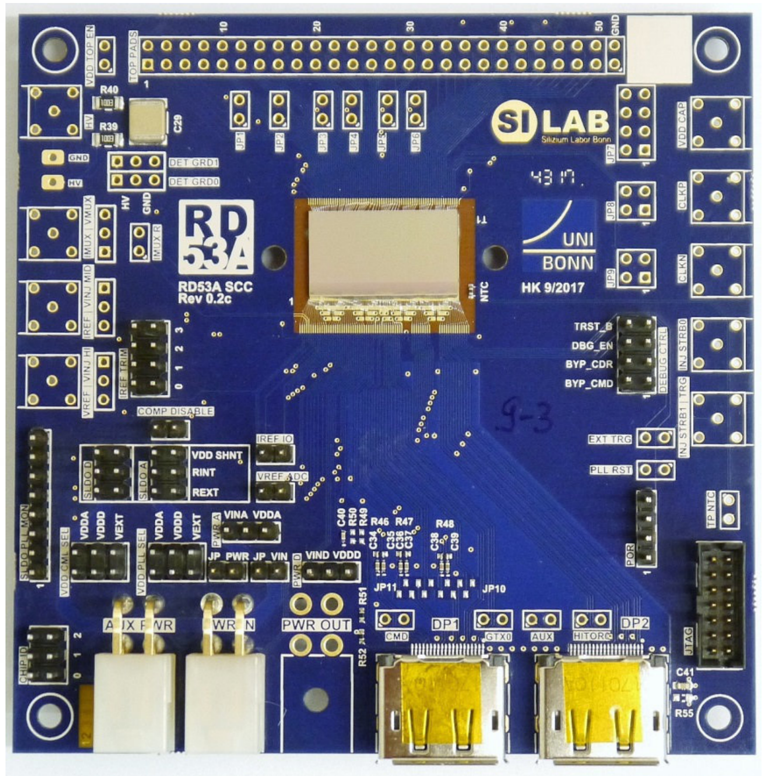
Currently testing the prototype ASIC for the new ATLAS pixel detector

➡ Goal: Provide useful feedback to the ASIC's designers (the Rd53 collaboration).

- They will use this feedback to build the final version for ATLAS and CMS



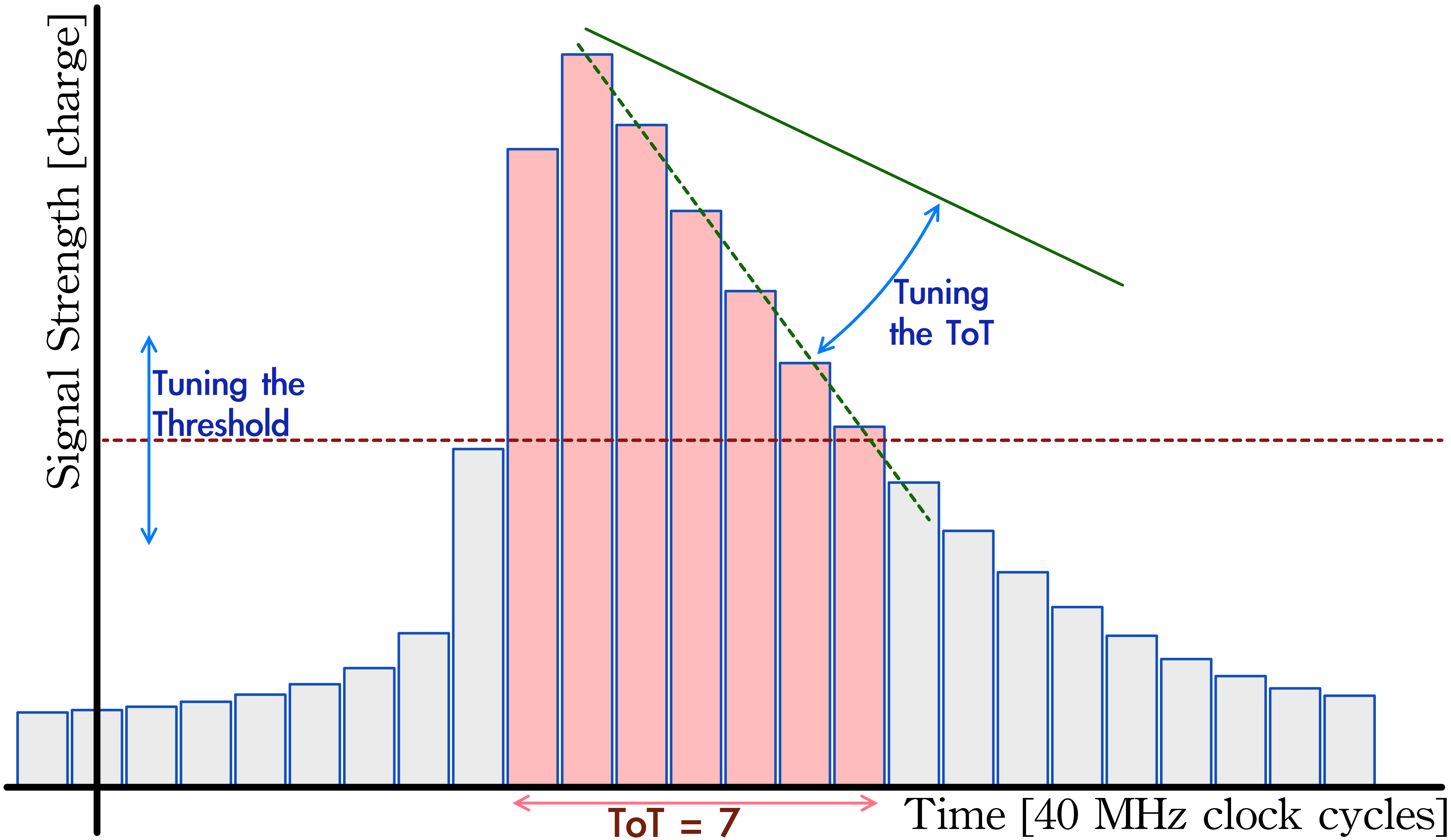
- The framework used to configure and test the ASIC is called **YARR** (Yet Another Rapid Readout).



The ASIC "Rd53a" assembled in a Single Chip Card

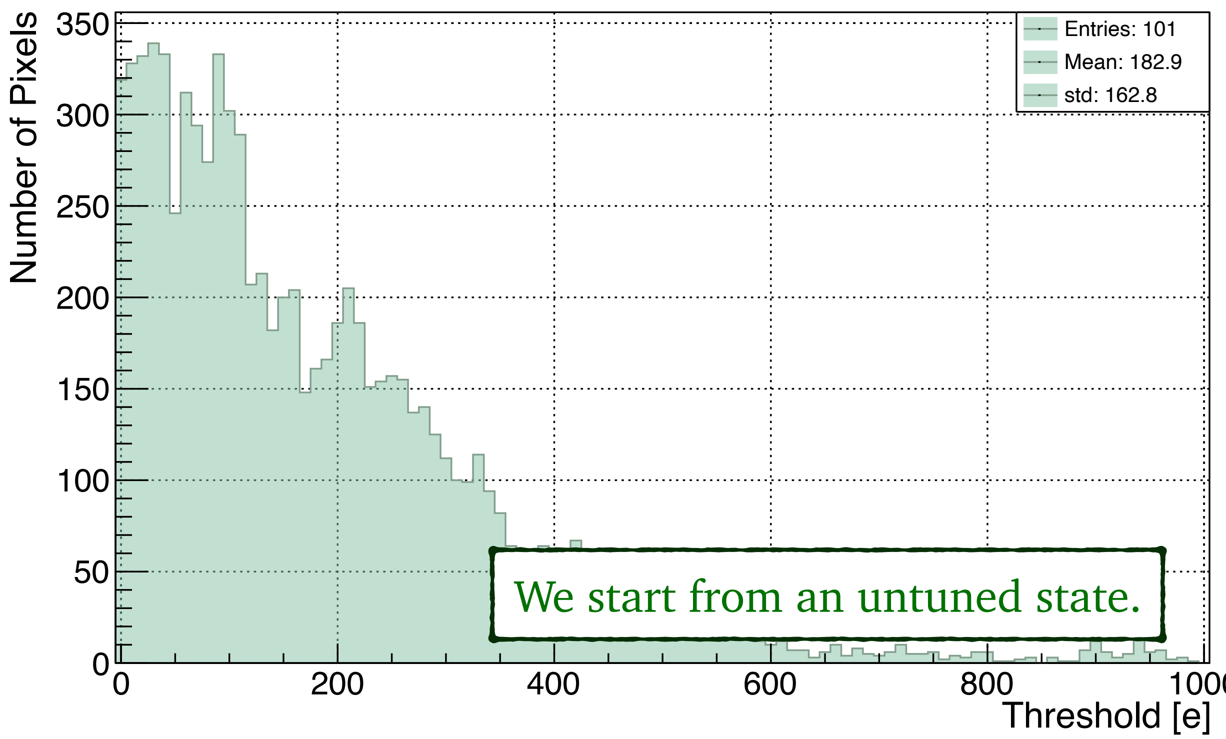
The Threshold and the ToT

- Main function of each pixel: To distinguish an actual hit from the background noise.
- The behaviour of the pixels is mainly defined by the **Threshold** and the **ToT** (Time over Threshold)



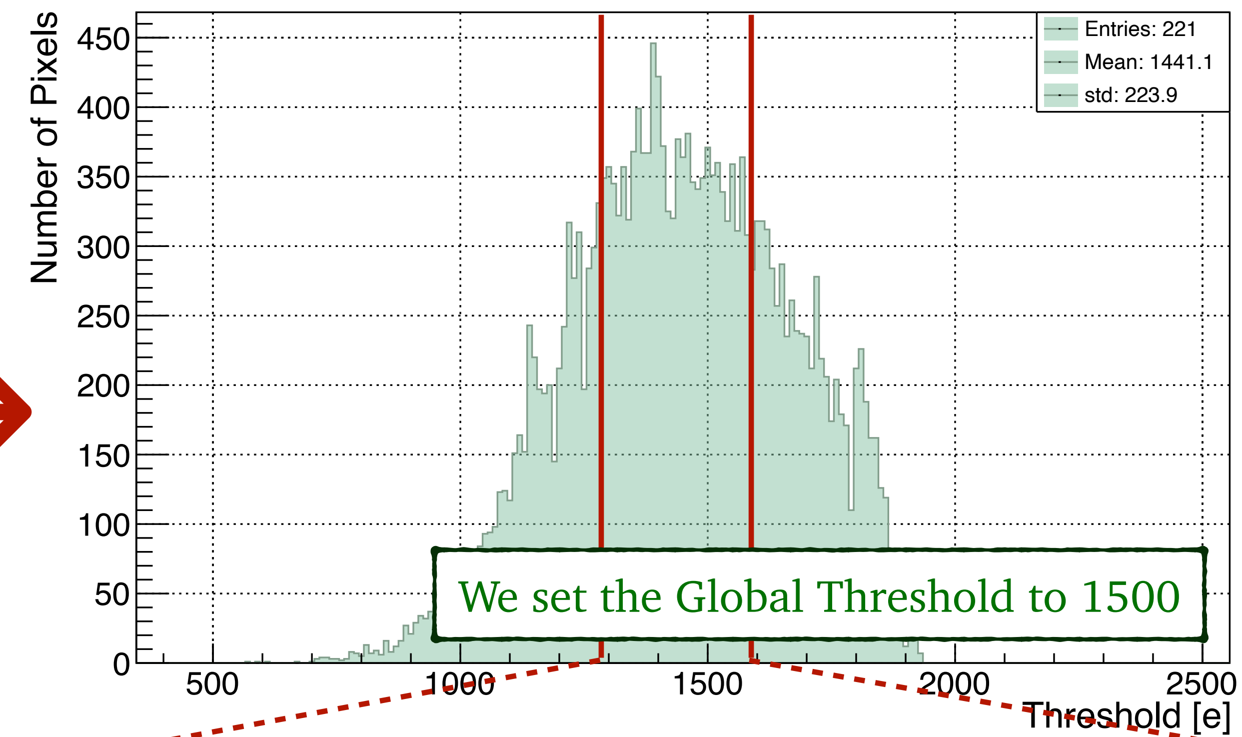
1. Global Threshold

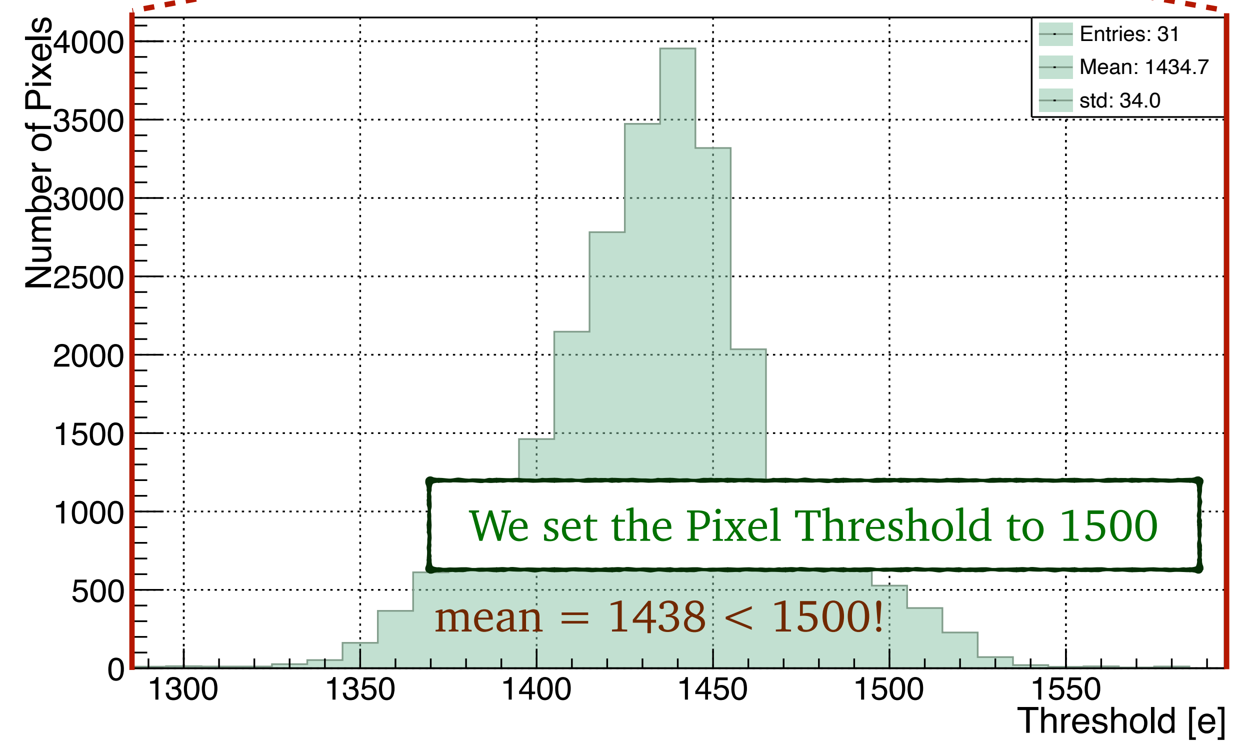
A global register affecting all the pixels. Tuning it returns a wide distribution centered in the target value.



2. Pixel Threshold

A register that exist for each pixel. This is the finer tuning step to be run after the global threshold tuning.





- The target threshold and the measured one where observed to be always slightly different.
- We have found the origin of this issue, and we have already solved it.
- We have also written a small framework to show on the web an interactive version of these plots. Just click on them!

See Yamagaya's presentation for the details regarding the tuning procedure

Decomposing the overall time when operating the ASIC

- Any operation on the ASIC takes some time:

  Total Time = Time on the **software** side (the YARR framework)
  + the actual **hardware** operations.

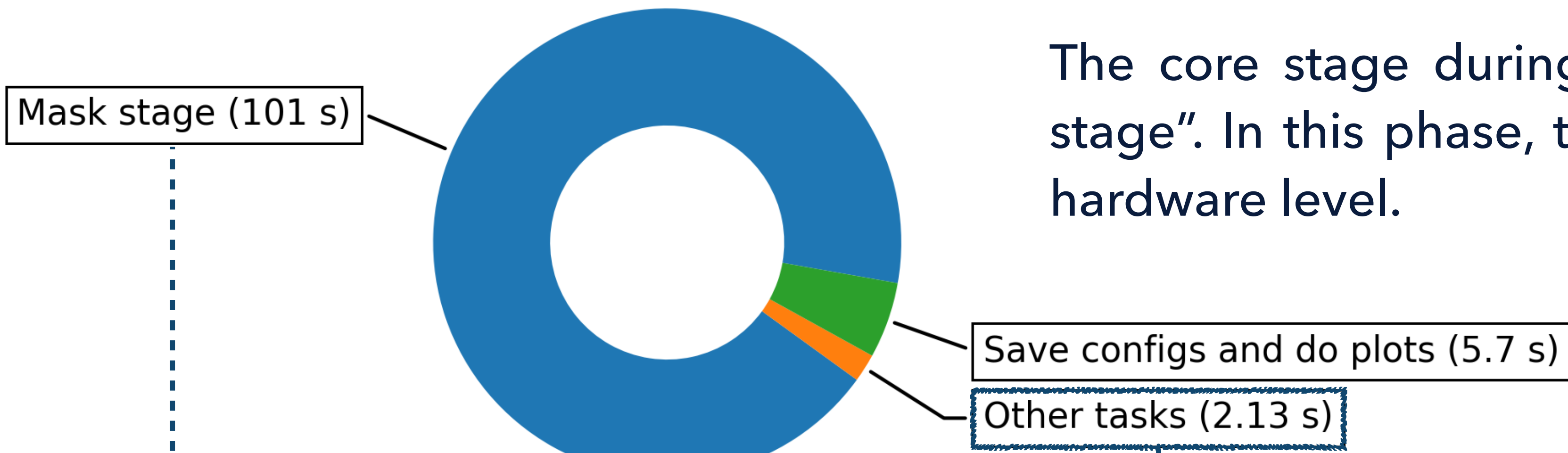
➡ Only the software side is in principle optimizable.

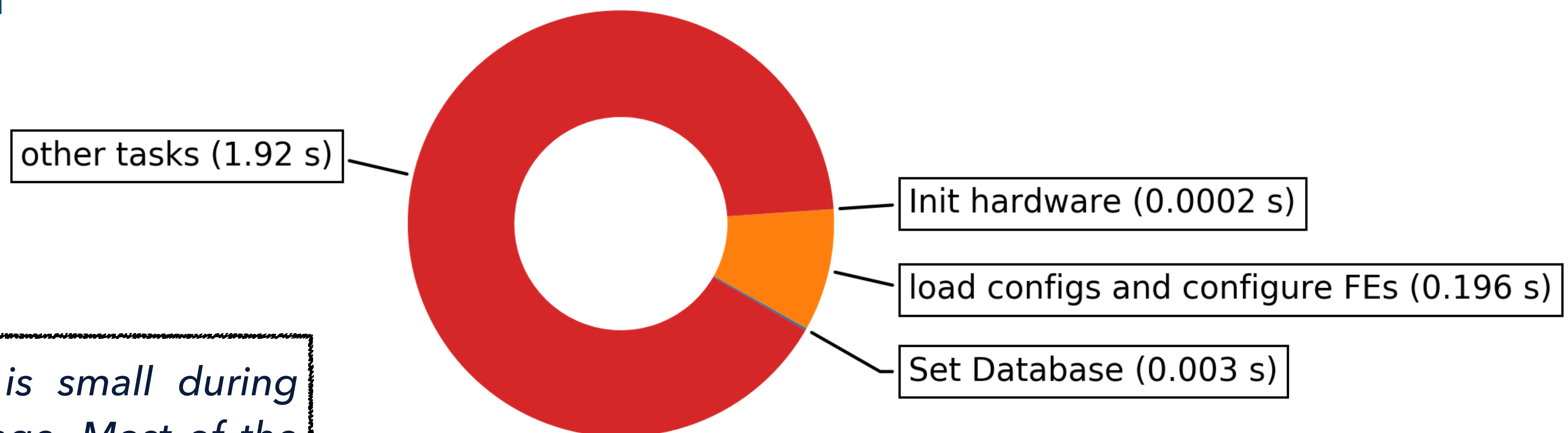
➡ We need to know how worth is to do it.

➡ How big is the ratio $\frac{\text{time taken by the framework}}{\text{overall scan time}}$?

Time consumption during a threshold scan

The core stage during an ASIC operation is the "Mask stage". In this phase, the ASIC is tuned / scanned at the hardware level.





CPU usage is small during the Mask Stage. Most of the time is consumed by the hardware.

There are still some parameters affecting the Mask Stage that we can tune to reduce the overall consumed time.

➡ Useful when we know the specific working conditions

➡ Can increase the uncertainty of the results in other case.

What I have done so far?

Solved the issue that led to a mismatch between the target and the measured thresholds

- ➡ And reported it to the developers.

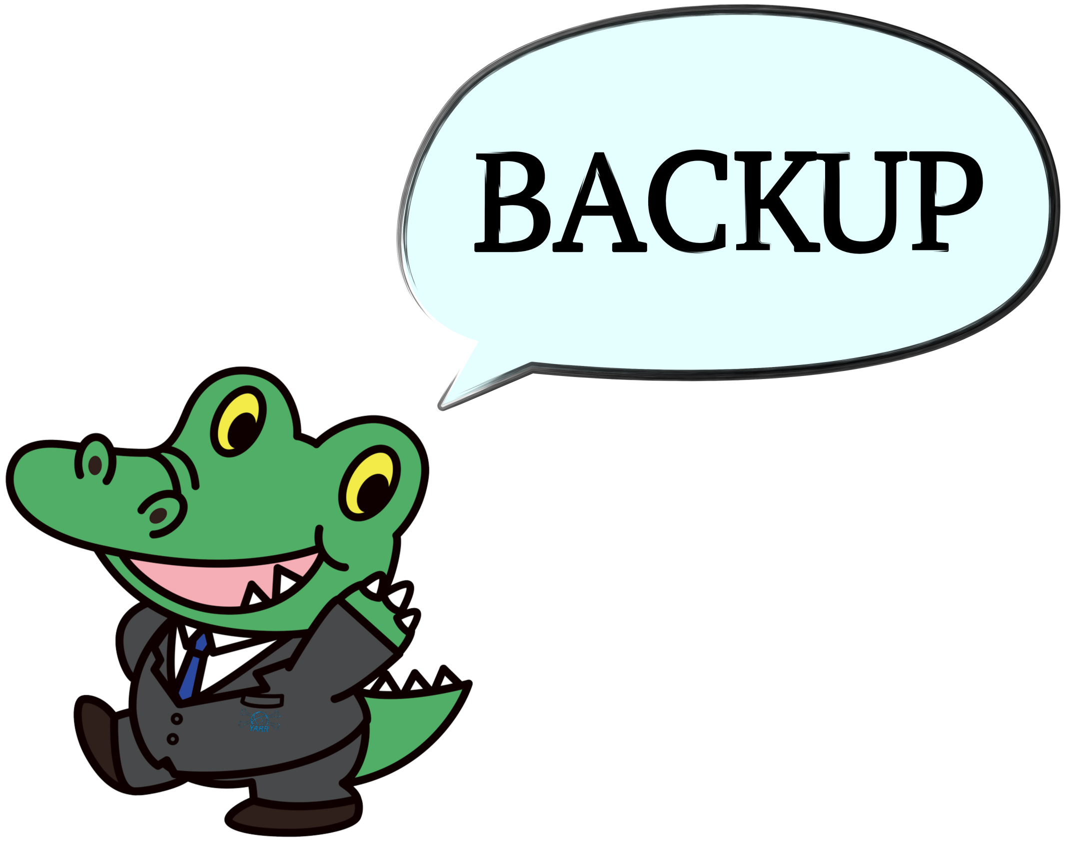
Measured the overall scan performance and reduced its consumed time under specific conditions

- ➡ Finding the appropriate configuration parameters for each scan can significantly increase its speed without compromising precision.
- ➡ I will continue working on it next year.

Completely timed a full electrical test on the ASIC

- ➡ An electrical test consist on a sequence of scan / tuning phases to check whether the ASIC is working properly after making a QA/QC test on it. My results are indeed helpful to estimate the overall time needed for the whole QC procedure.

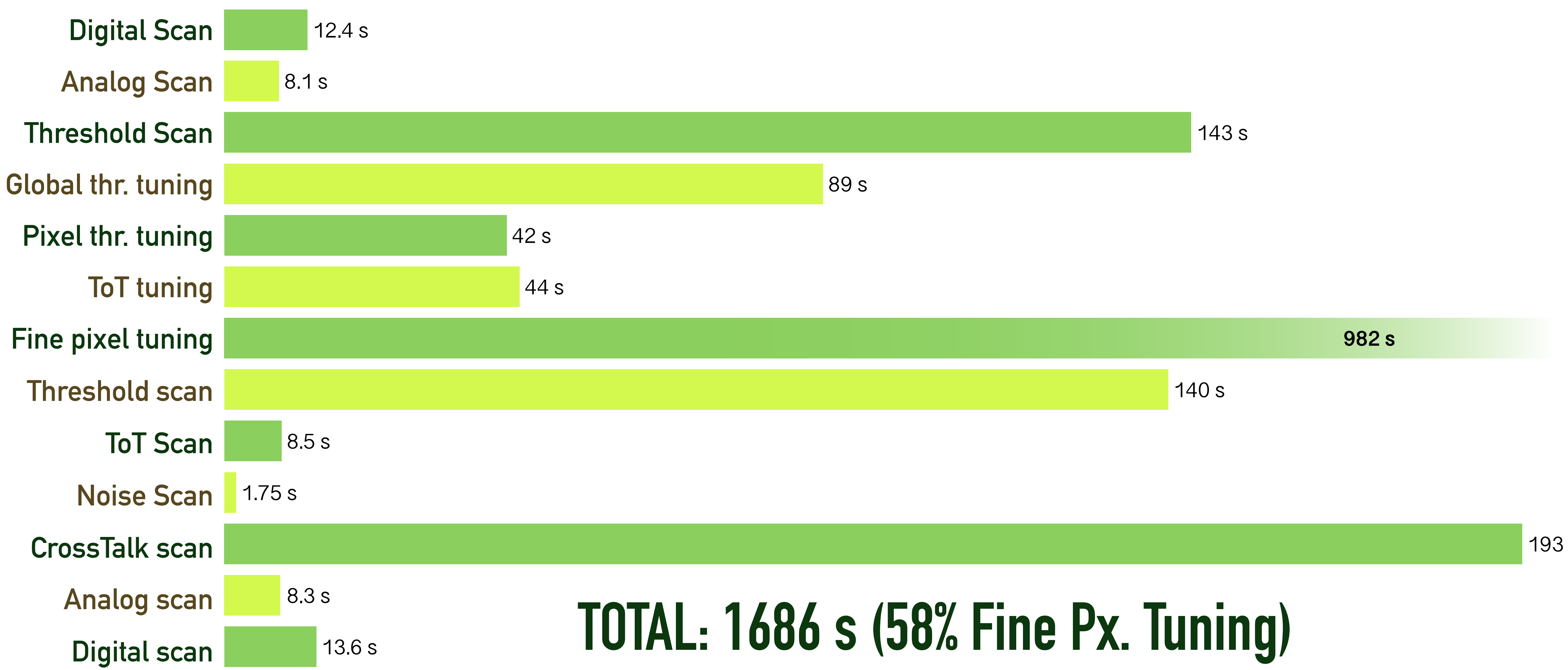
Also, I got used to work in a collaboration, and I learnt a lot from the work of my mates. Also improved my Japanese, although is still one of the main TODOs for the next year. Let's keep doing our best!


BACKUP

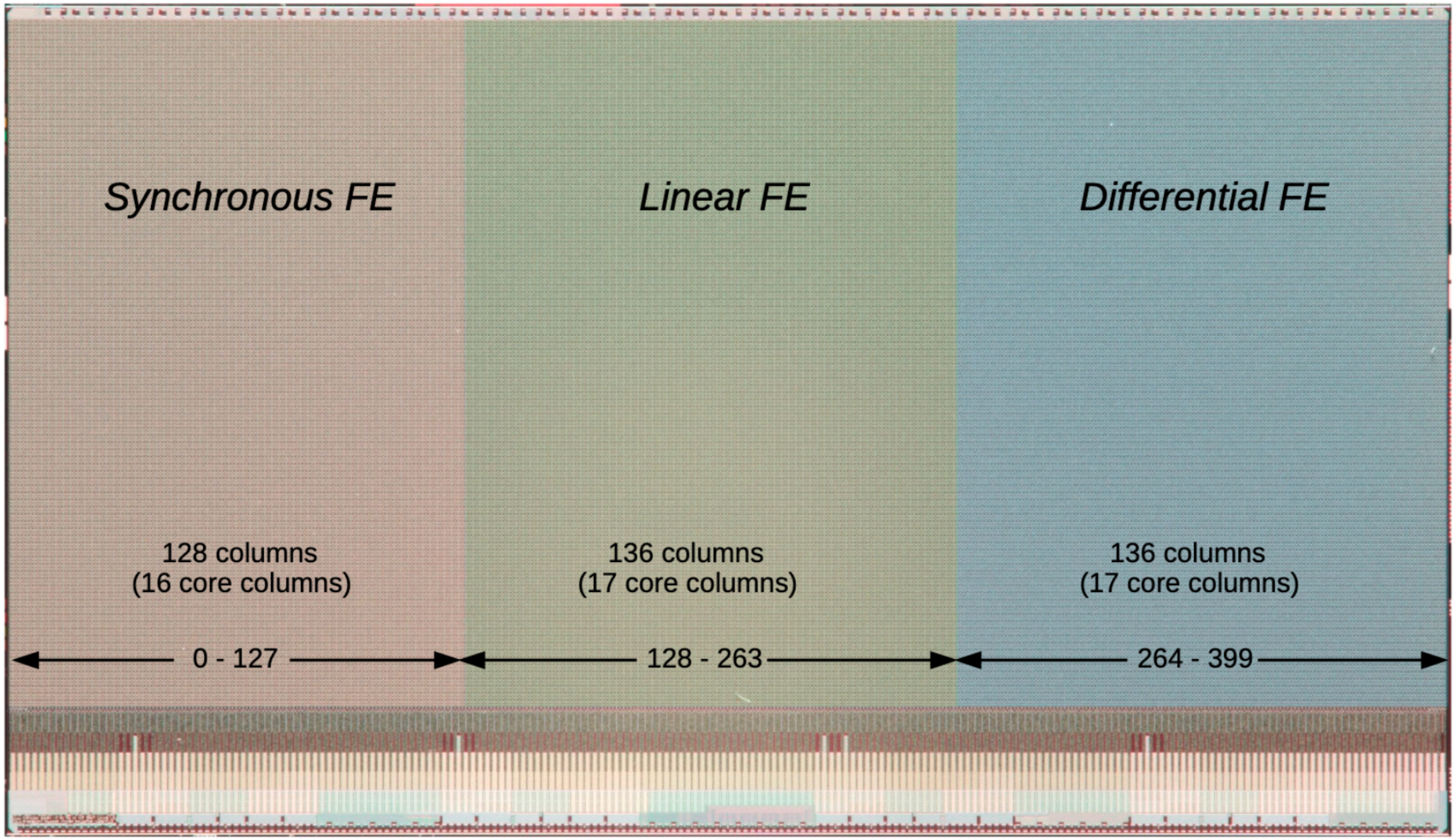
The QC Flow during production

Different QC tests (such as visual inspections, thermal cycles or electrical tests) are currently being performed on the modules to control their quality. The whole QC flow takes a lot of time (in the order of days), and it's therefore important to have a rough estimation on how much time it could take to complete each phase of it.

An electrical test consists on sequential scans that we have already timed. The total needed time is roughly 30 minutes.



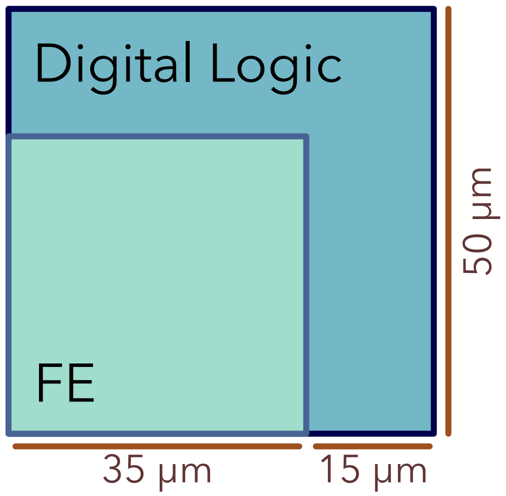
TOTAL: 1686 s (58% Fine Px. Tuning)



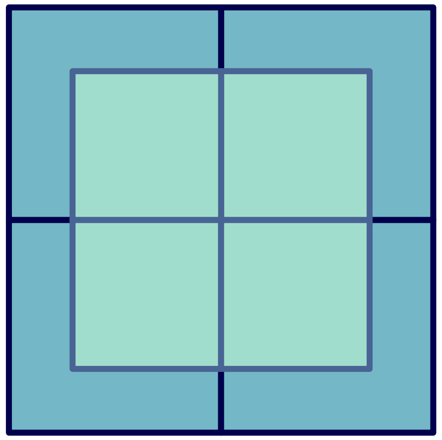
- 400 columns and 192 rows of pixels
- A total of 76800 px in a 11.8 x 20 mm matrix
- Three different Front Ends built for testing purposes
- The Differential FE has been decided to be the most efficient under the real working conditions

- Four pixels form an Analog Island. A matrix of 4x4 analog Islands is grouped under a Digital Core, that configures the islands and handles all the processing of the pixels.

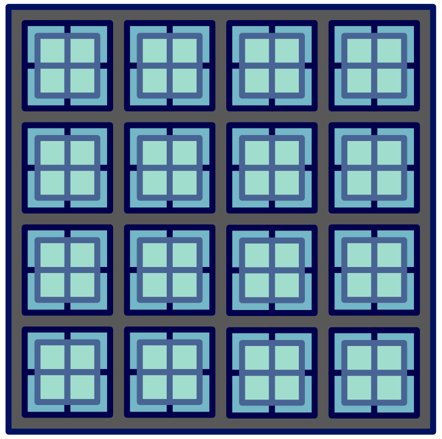
Pixel



Analog Island



Pixel Core



From **Charge** to **Vcal**, and from **Vcal** to **Charge**

Rd53aCfg.cpp::toVcal (charge)

```
V = (charge * ElectronCharge) / (m_injCap);
vcal = (V)/(m_vcalPar[1])  // Note: no offset applied
return vcal
```

➡ The offset is not applied when converting charge to Vcal

m_vcalPar is declared in Rd53aCfg.h

```
std::array<double, 4> m_vcalPar;
//mV, [0] + [1]*x + [2]*x^2 + [3]*x^3
```

The “offset” is defined by m_vcalPar[0]

Rd53aCfg.cpp::toCharge (vcal)

```
V = (m_vcalPar[0] + m_vcalPar[1]*vcal)/ElectronCharge;
return V*m_injCap;
```

➡ The offset is applied when converting Vcal to charge

default_rd53a.json

```
"Parameter": {
            "ChipId": 0,
            "InjCap": 8.2,
            "Name": "JohnDoe_0",
            "VcalPar": [-1.0,0.195,0.0,0.0]
        }
```

Fei4Cfg.h::toVcal (charge)

```
V = (charge * ElectronCharge) / (sCap + lCap)
vcal = (V - vcalOffset)/(vcalSlope)
return vcal
```

default_fei4b.json

```
"Parameter": {
            "chipId": 0,
            "lCap": 3.8,
            "sCap": 1.9,
            "vcalOffset": 0.0,
            "vcalSlope": 1.5
        }
```

toVcal and toCharge are not symmetric. We should either remove the offset from toCharge or include it in toVcal: `vcal = (V)/(m_vcalPar[1])` → `vcal = (V - m_vcalPar[0])/(m_vcalPar[1])`

Before and after our modification in Yarr

Given threshold: 1000 e

Before the modification

OFFSET	Measured THRESHOLD
2	1100 ± 33
1	1049 ± 35
0	998 ± 32
-1	947 ± 33
-2	897 ± 34

After the modification

OFFSET	Measured THRESHOLD
2	1001 ± 32
1	997 ± 30
0	999 ± 33
-1	997 ± 35
-2	996 ± 34

Given Offset: 0 mV

Before / After our modification

Given threshold:	500	900	1300	1700
Measured threshold:	498	899	1294	1695

The YARR's configuration files

std_analogscan.json

```
{
  "scan": {
    ...
    "loops": [
      ...
      {
        "config": {
          "max": 50,
          "min": 0,
          "step": 1,
          "nSteps": 5
        },
        "loopAction": "Rd53aCoreColLoop"
      },
      ...
    ],
    "name": "AnalogScan",
    "prescan": {
        "InjEnDig": 0,
        "LatencyConfig": 48,
        "GlobalPulseRt": 16384,
        "InjVcalHigh": 2500,
        "InjVcalMed": 500,
        "InjVcalDiff": 0
    }
  }
}
```

The differential FE starts at the 33th core column

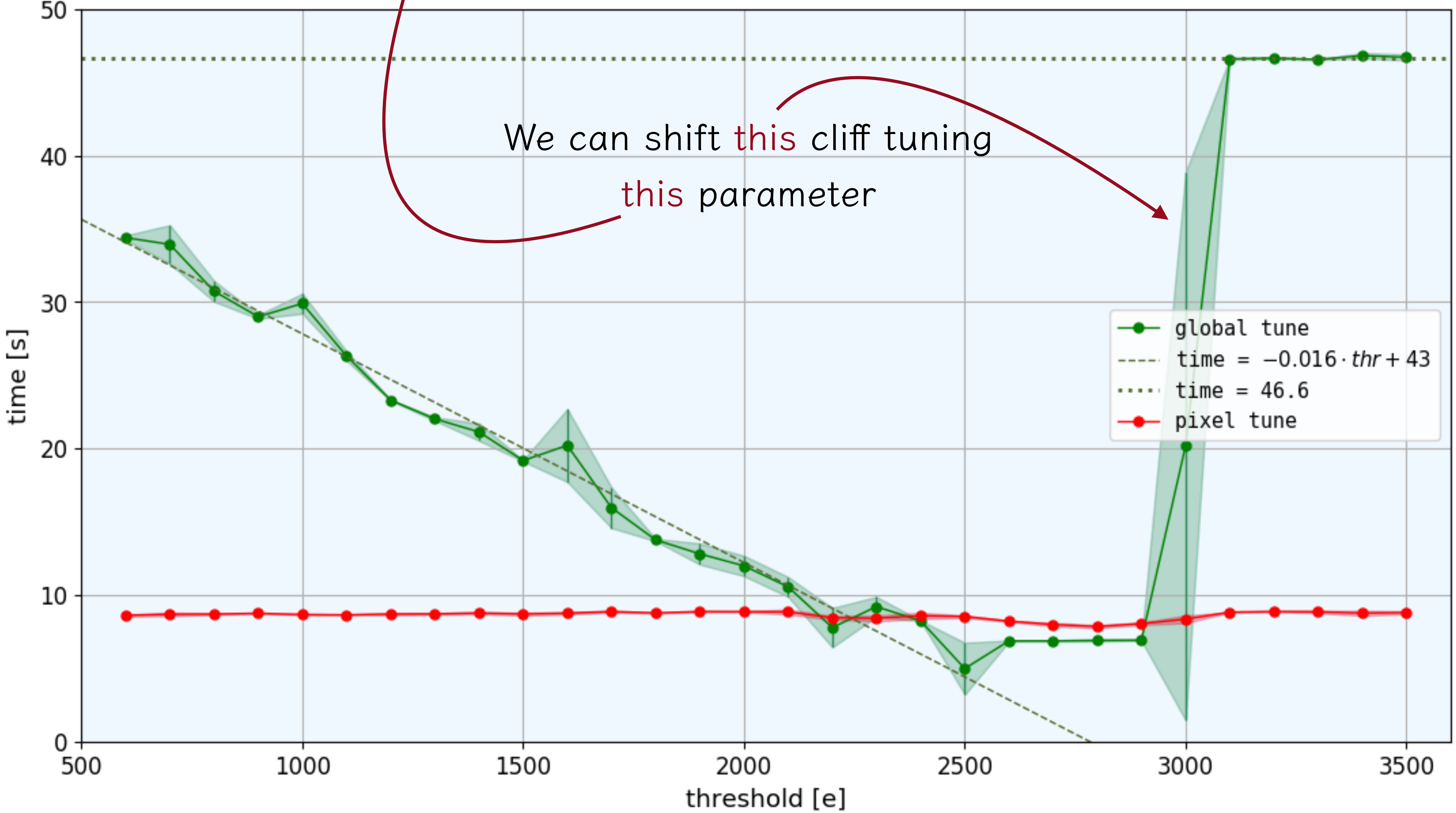
diff_analogscan.json

```
{
  "scan": {
    ...
    "loops": [
      ...
      {
        "config": {
          "max": 50,
          "min": 33,
          "step": 1,
          "nSteps": 2
        },
        "loopAction": "Rd53aCoreColLoop"
      },
      ...
    ],
    "name": "AnalogScan",
    "prescan": {
        "InjEnDig": 0,
        "LatencyConfig": 50,
        "GlobalPulseRt": 0,
        "InjVcalHigh": 2500,
        "InjVcalMed": 500,
        "SyncVth": 500,
        "LinVth": 500,
        "EnCoreColLin1": 0,
        "EnCoreColLin2": 0,
        "EnCoreColSync": 0
    }
  }
}
```

Tuning time consumption as a function of the Target Threshold

diff_tune_globalthreshold.json > scan > loops > DiffVth1 > max = 500
diff_thresholdscan.json > scan > loops > InjVcalDiff > max = 400

We can shift this cliff tuning this parameter



Measurements to be done for the QC document I

We will include the following scans in the Sequential Operator:

- Threshold scan
- Tune global threshold
- Tune pixel threshold
- Tot tuning
- Fine tune pixel
- ToT scan
- Noise occupancy scan
- Disconnected bump scan
- Stuck pixel scan
- Crosstalk scan
- Analog scan
- Digital scan

Stuck pixel scan: There is no implementation in the Master branch, but Yarr has a branch called "stuck_pixel_scan"

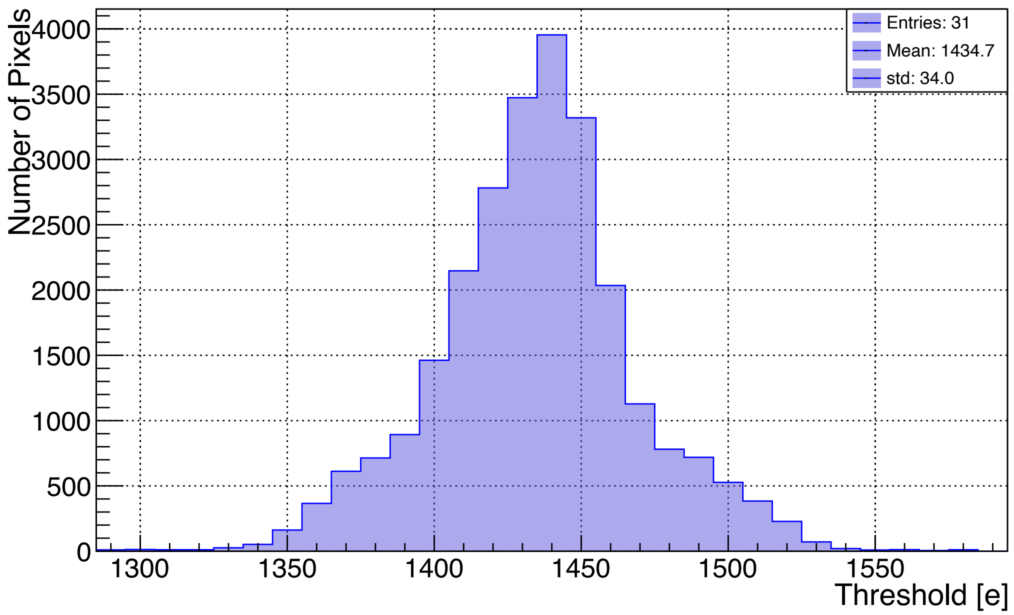
Crosstalk scan: We have std_crosstalk_scan but not diff_crosstalk_scan

Digital scan: We have std_digitalscan but not diff_digitalscan

Pixel Failure Test (Tuning), sec. 4.4.4

Std deviation calculation in Yarr

The starting point is an histogram where we know the width and content of each bin



The variance of a set of data is defined as

$$\sigma^2 = \frac{1}{N}\sum(x_i - \mu)^2$$

And the standard deviation as

$$\sigma = \sqrt{\sigma^2}$$

Yarr computes the sum as

```
sum += data[i] * pow(((i*binWidth)+xlow+(binWidth/2.0)) - mu, 2);
```

- `data[i]`: All the events in the same bin
- `((i*binWidth)+xlow+(binWidth/2.0))`: x_i
- `mu`: μ
- `pow(...)`: All the events in the same bin have the exact same contribution to the variance

And then the standard deviation as

```
std += sqrt(mu/(double)sum);
```

This value is returned as uncertainty by Yarr after a threshold or a ToT scan.