

Smart Threshold Scan

Ismet Siral & Laura Jeanty & Timon Heim

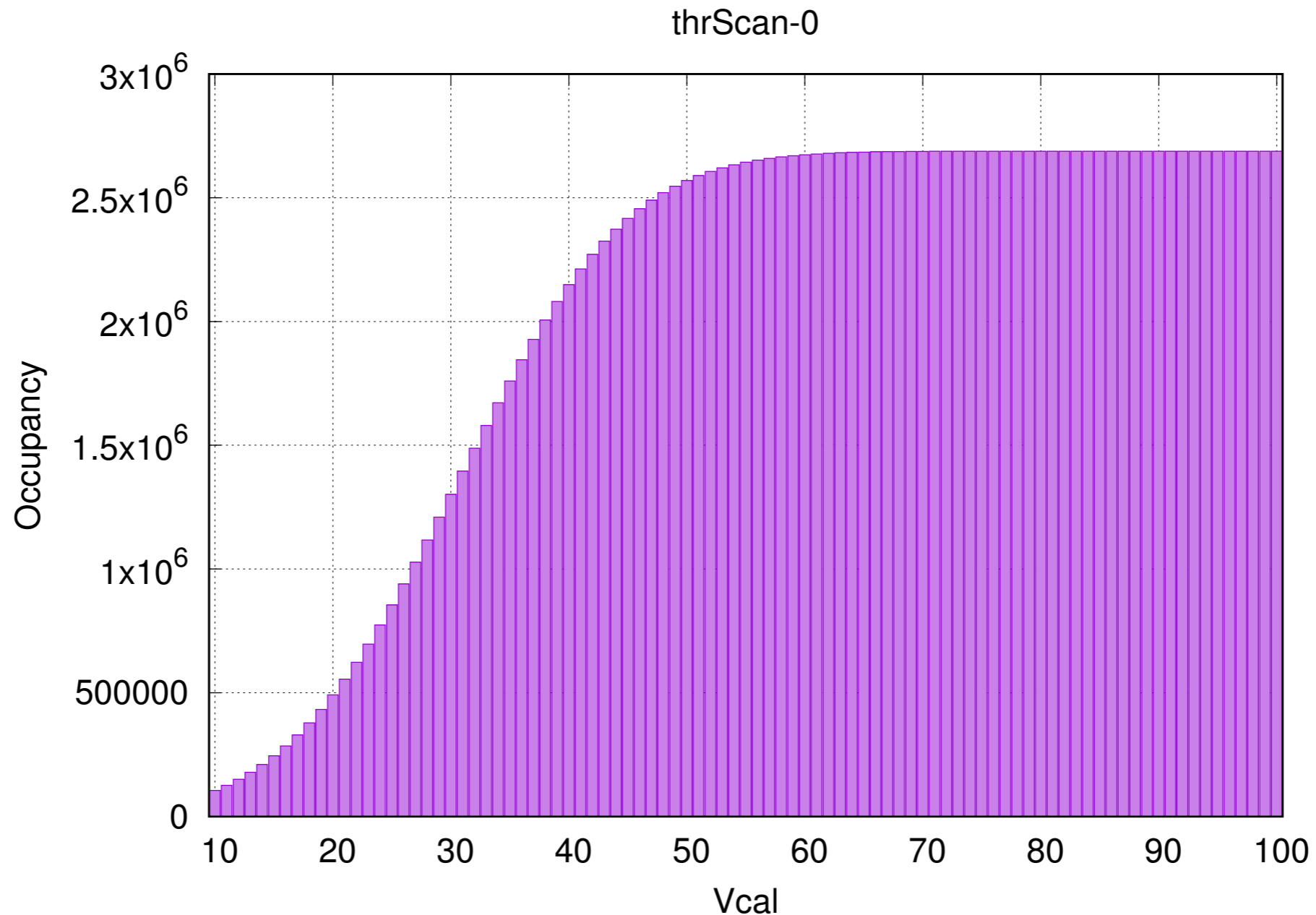
LBL Instrumentation Meeting
17/Oct/2019



UNIVERSITY
OF OREGON

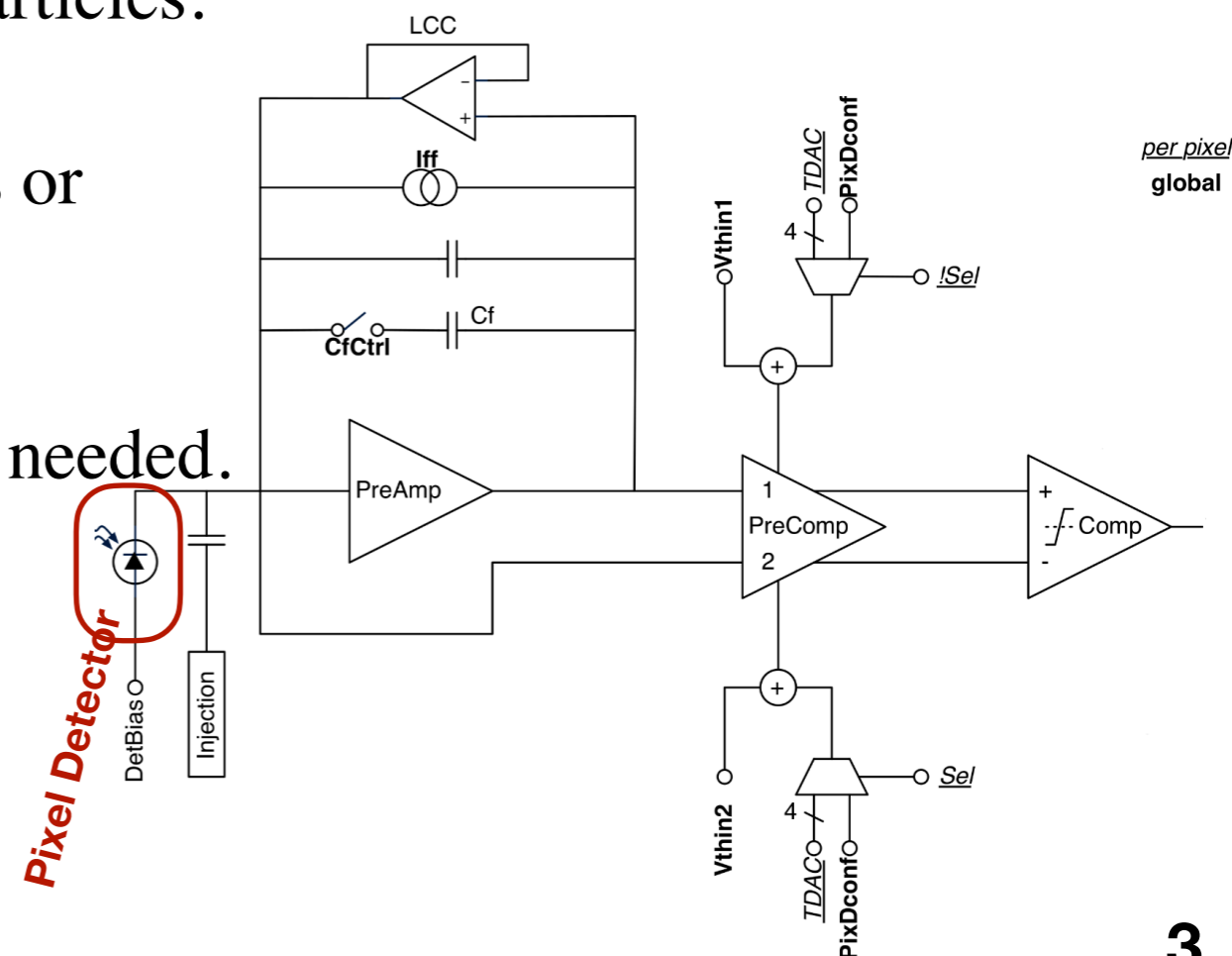
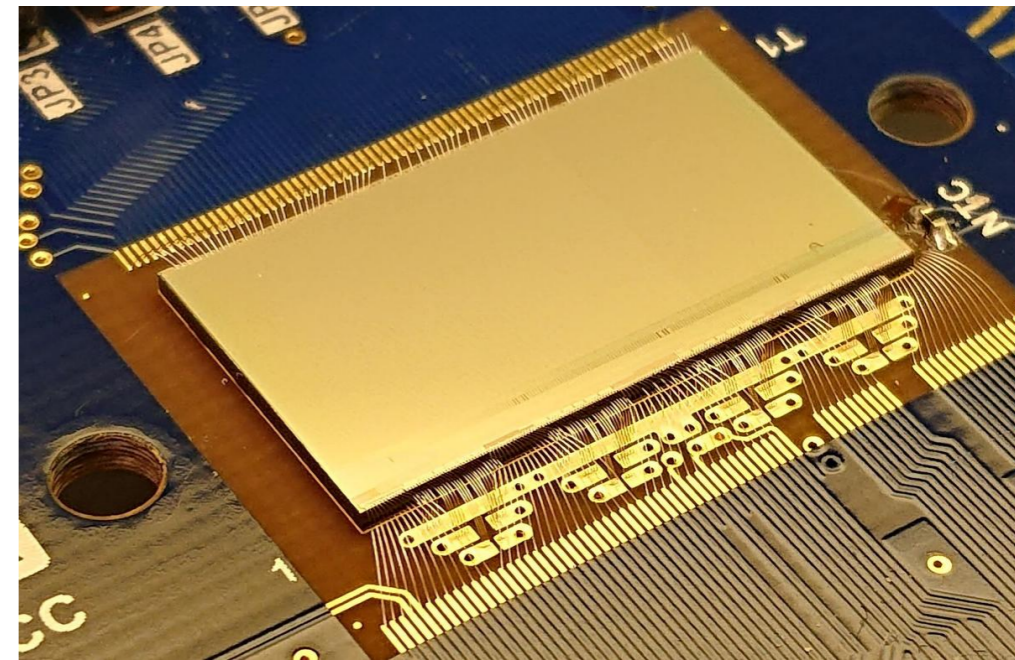
Introduction

- Today, I am going to talk about Smart Threshold Scan.
- What is a threshold scan?

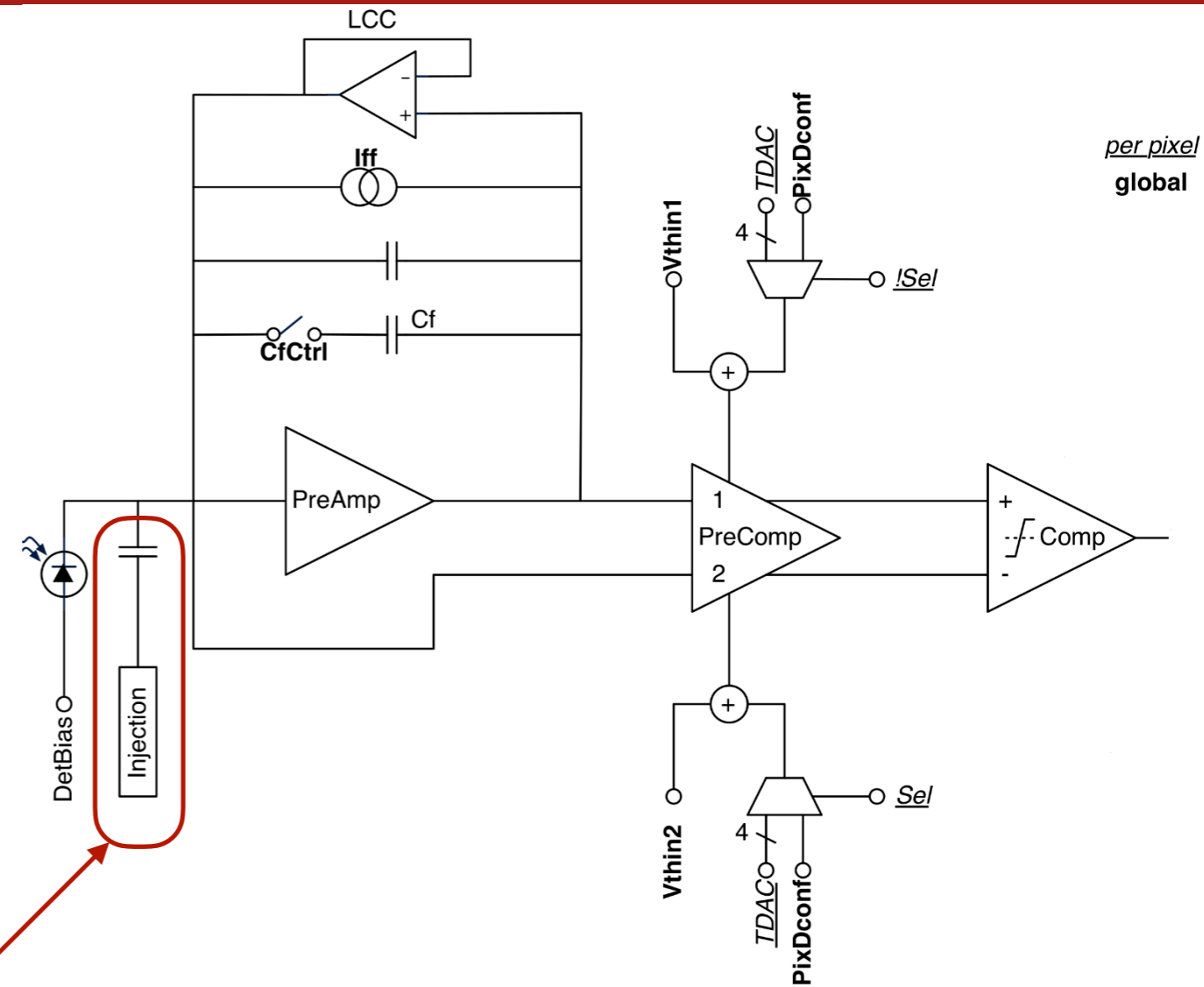


Threshold Scan

- As charged particles pass through silicon detectors, they leave some amount of charge.
- These charges are collected through an analog circuit, where they are amplified and then they are digitized.
- We (will) use a large array of these silicon detectors (Pixel Detector) in ATLAS to track these particles.
- Due to the fabrication processes, no two pixels or analog circuits are identical.
- For this reason calibration of these circuits are needed.



Analog Circuit



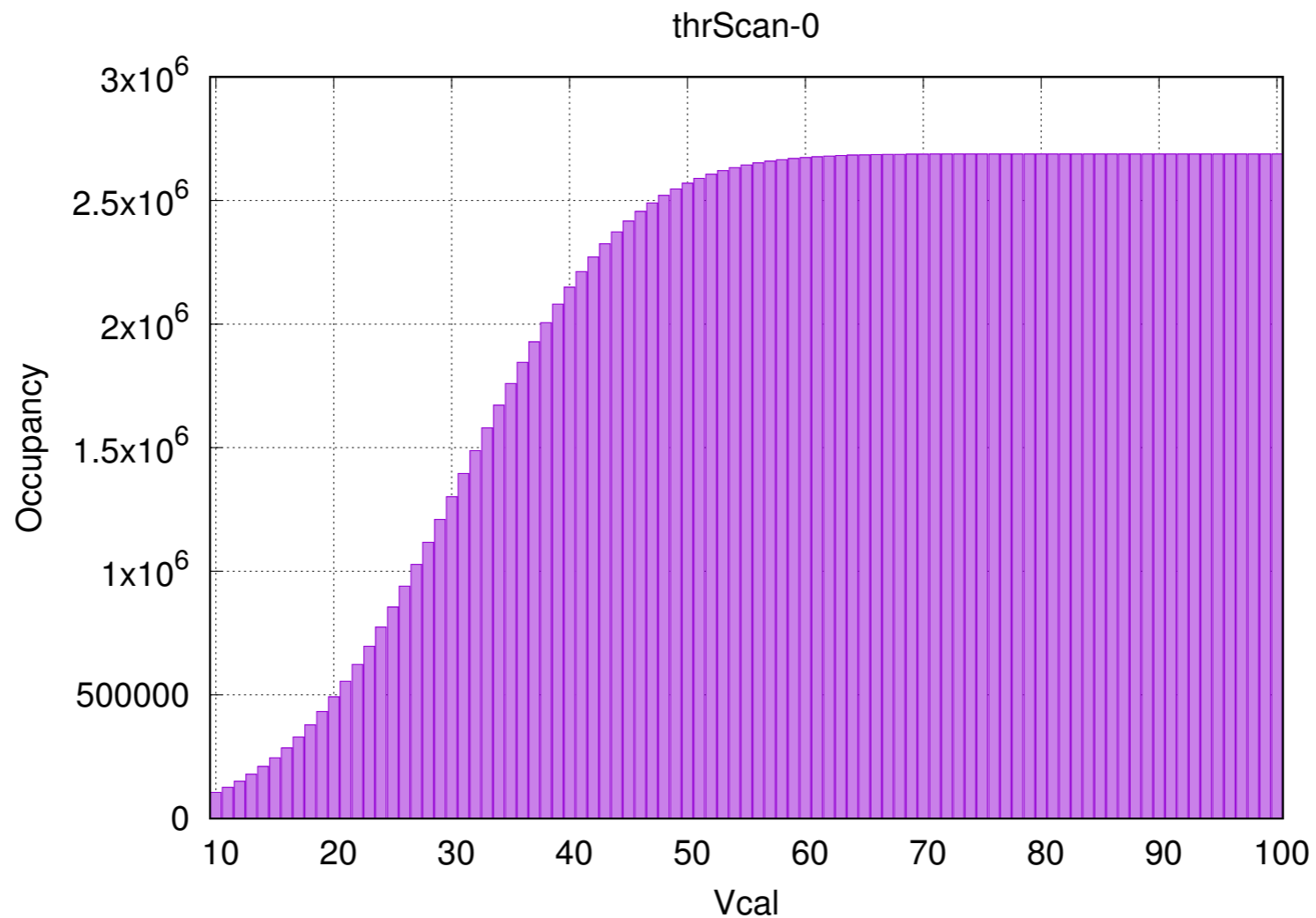
per pixel
global

- Each analog readout circuit has a small capacitor at the input that can be externally charged. (Injected)
 - Charging this capacitor mimics a silicon detector hit for the readout circuit.
 - This makes it possible for us to test and calibrate the readout circuit.

Threshold Scan

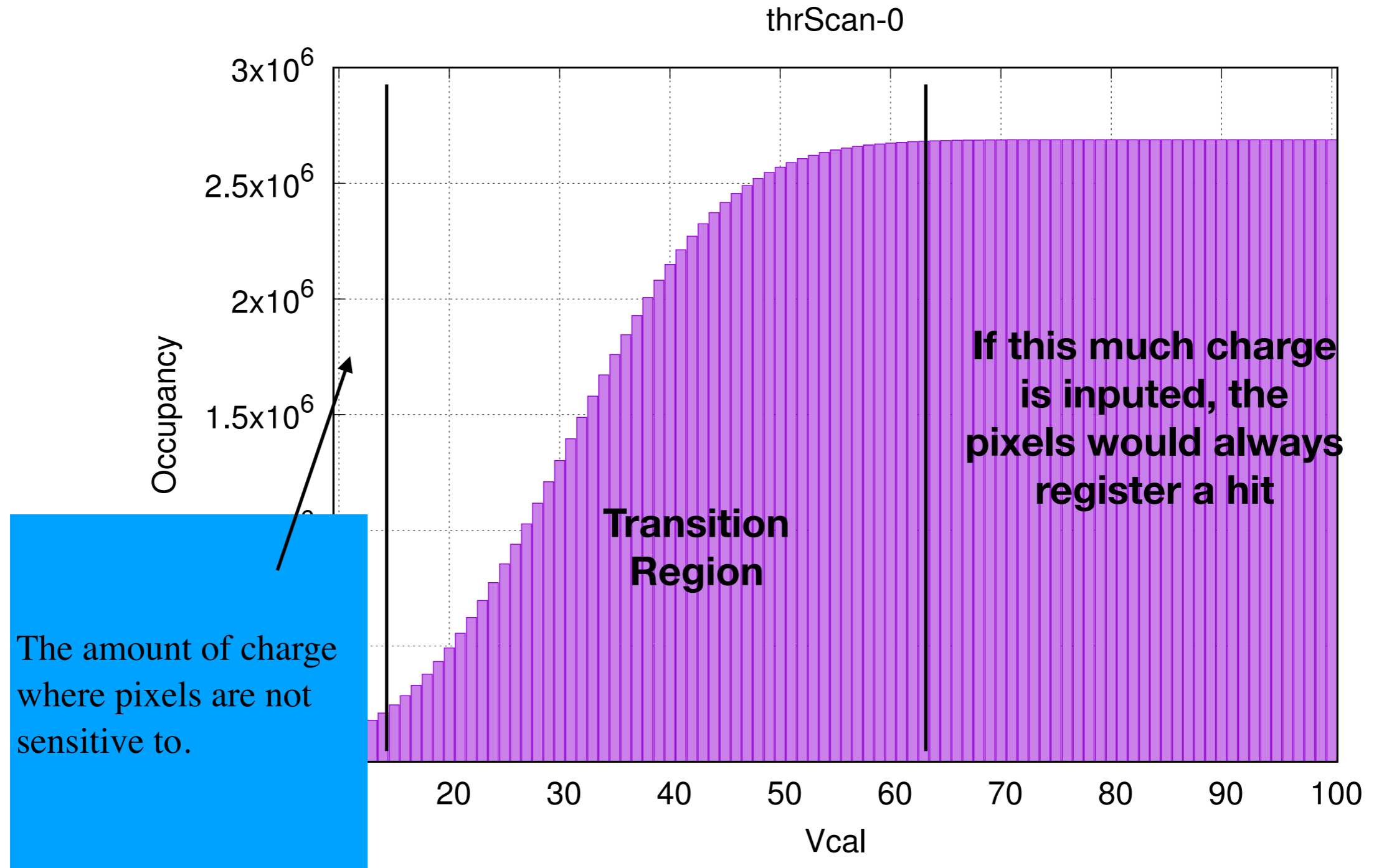
- Threshold scan is a basic calibration processes.
- Different amount charge are injected to a large array of readout circuits to observe it's behavior.
- This is use-full for identifying the average behavior of an analog circuit.

Total number of readout hits registered, for different amount of injected charge.



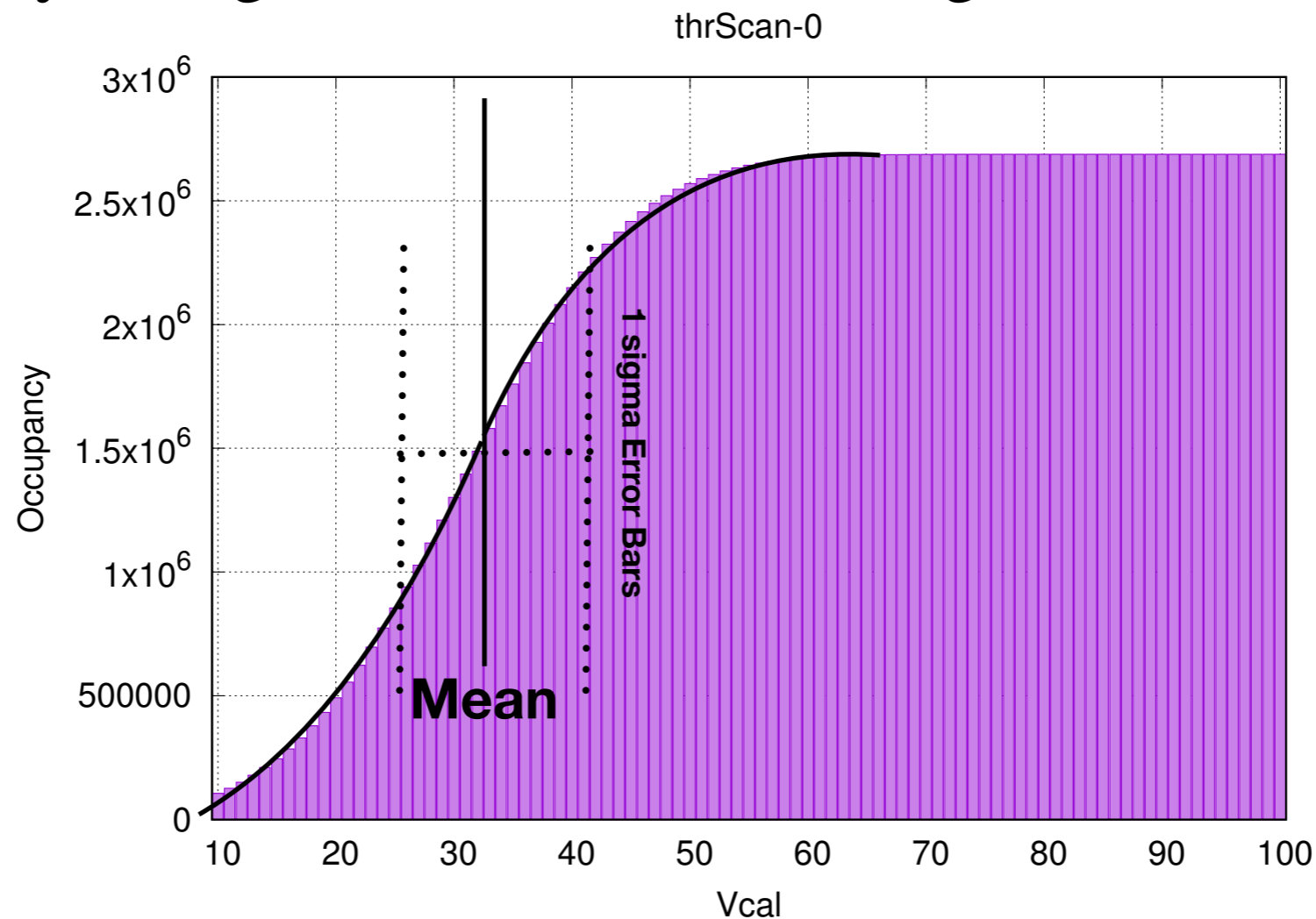
X: Different amount of injected charge.

Threshold Scan



Threshold Scan

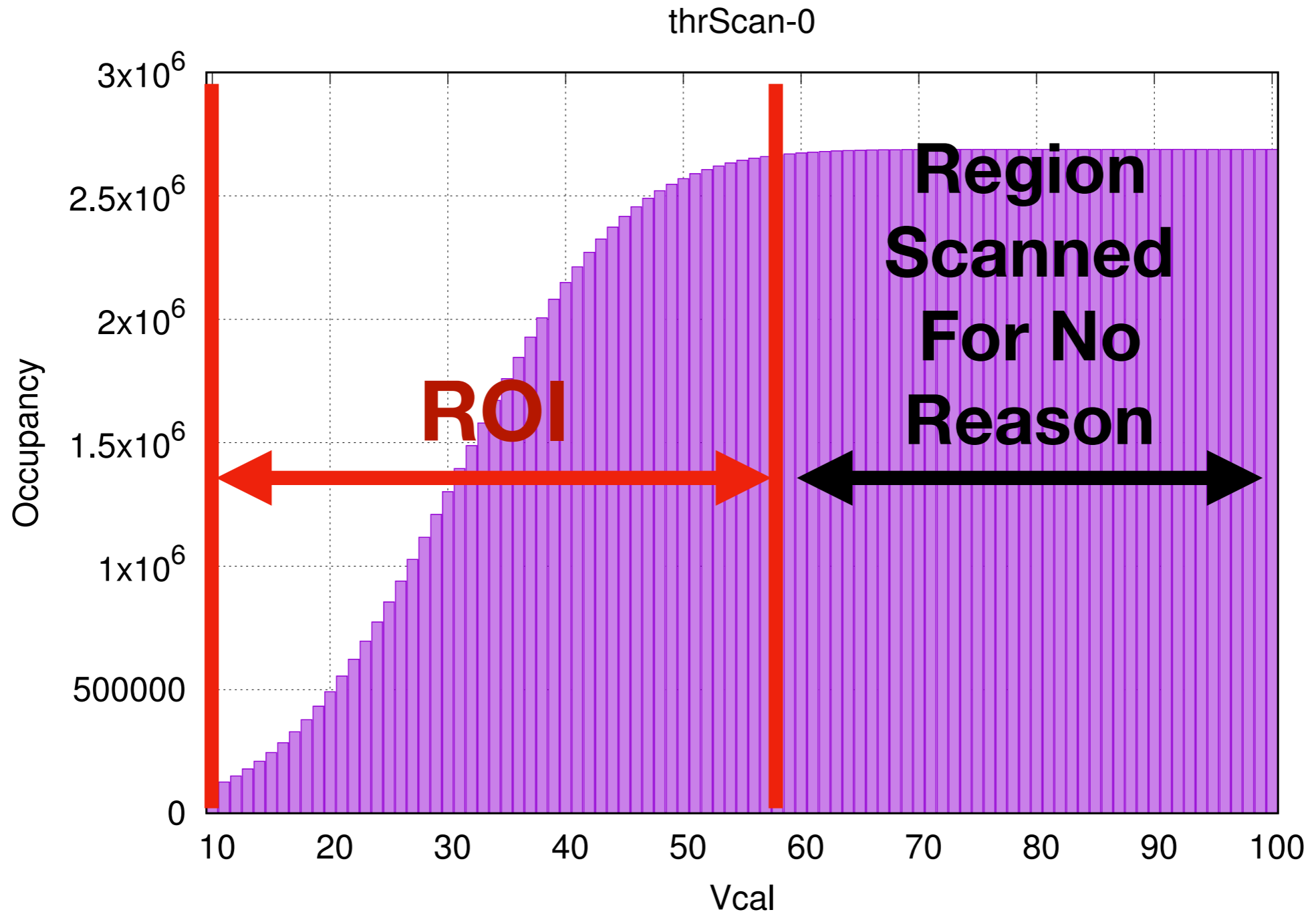
- We use threshold scan to identify the average amount of charge that is needed to trigger a pixel and the spread.
- We do that by fitting the threshold scan range: (SCurve)



- This process is slow since we have to scan a large injection range.

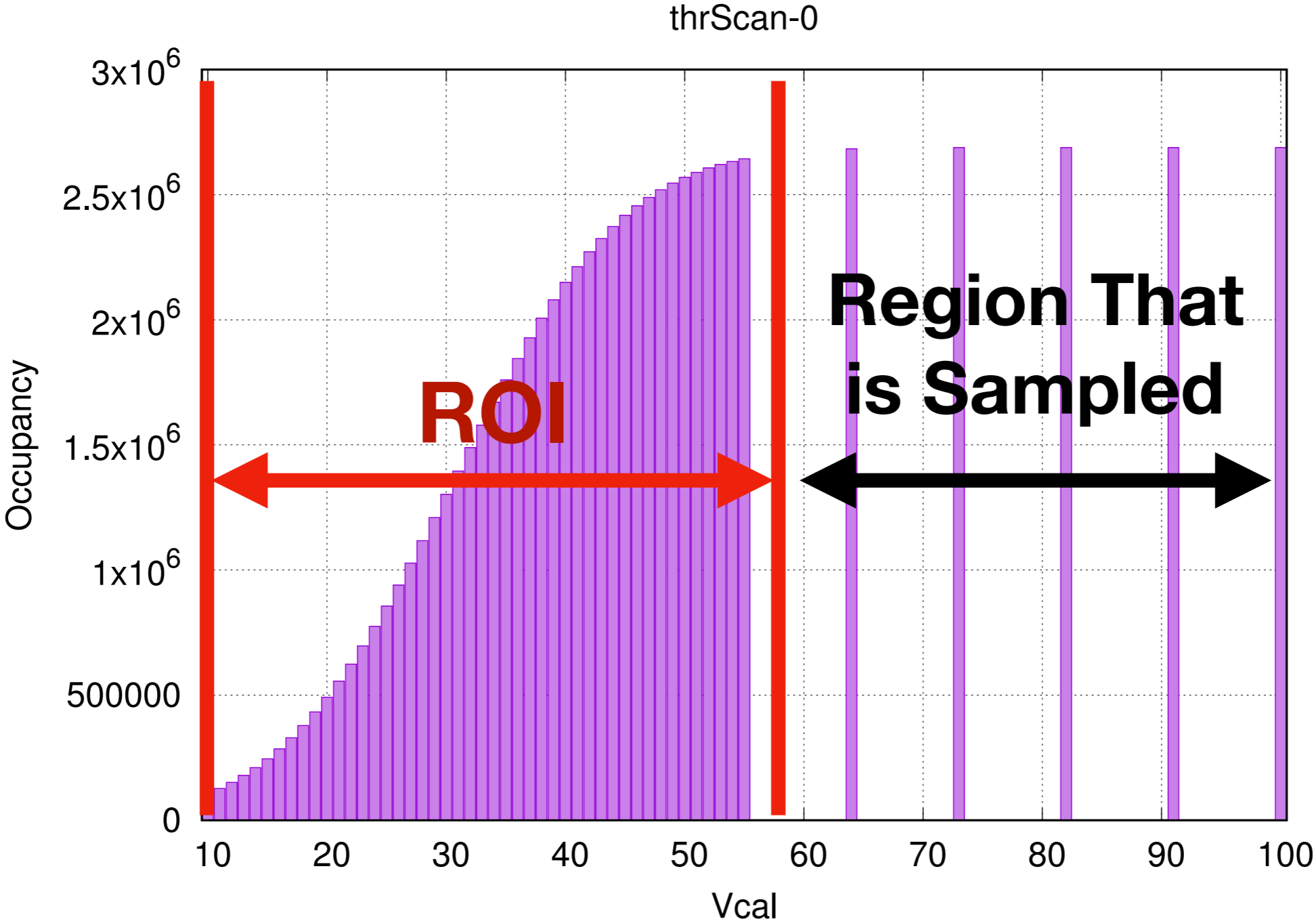
Region Of Interest

Basic Threshold Scan With Parameter Loop



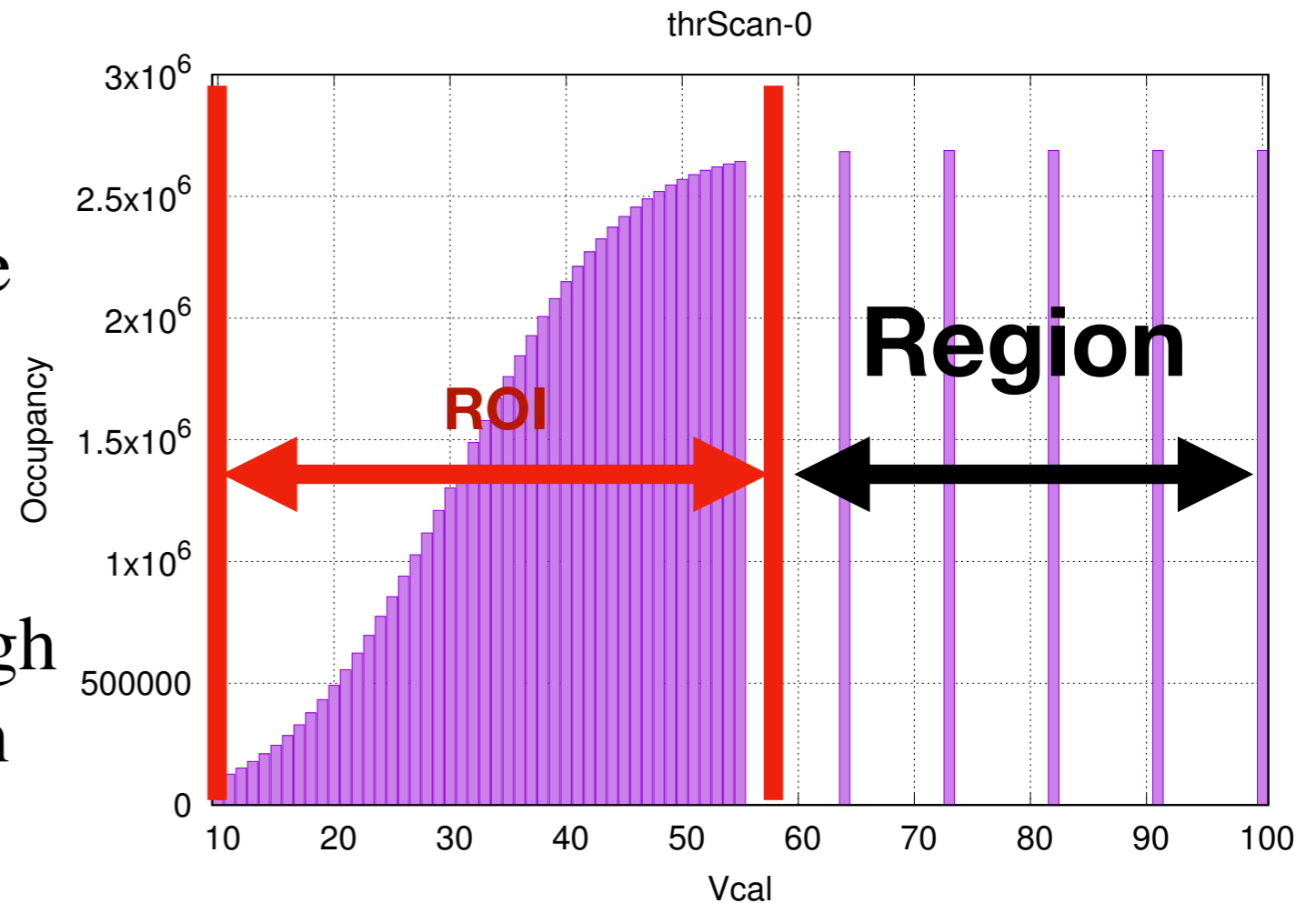
Smart Threshold Scan

The same threshold scan done with Smart Scan



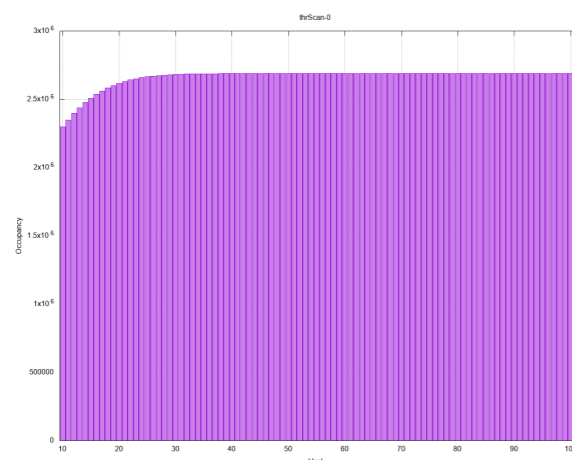
Smart Threshold Scan

- The aim of the smart threshold scan is to reduce the time a threshold scan takes.
- This is achieved by sampling the whole threshold region first, to identify a region of interest.
- Then by sampling the ROI in high granularity to calculate the mean and the spread.

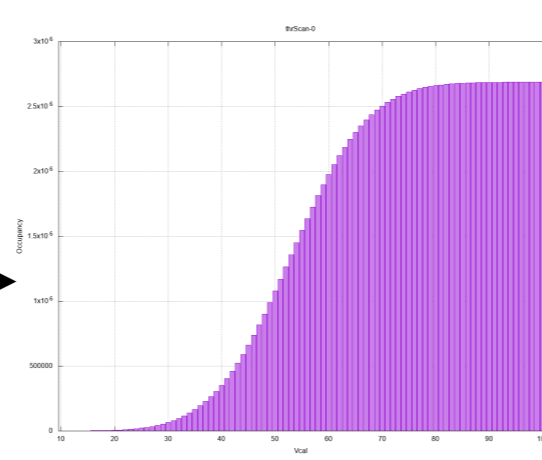


Simulation Performance

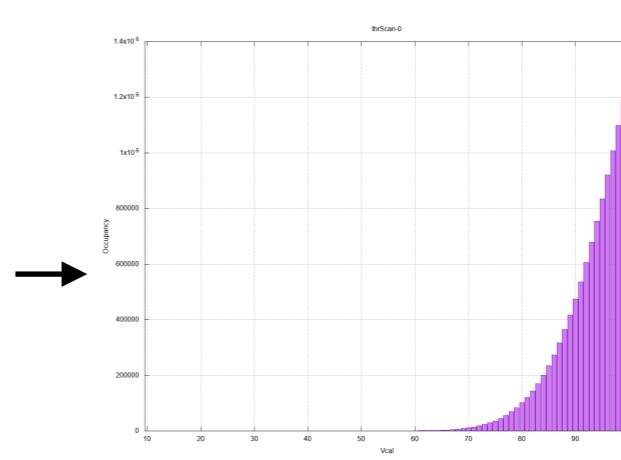
- The tool was first designed for the Fei4 Simulator and later migrated to Rd53 hardware.
- The simulation tests were done at different the voltage thresholds which can be easily varied in a simulator. This is parameterized by an arbitrary parameter.
(I don't know the direct conversion to volt values of this parameter)



Voltage Par 2



Voltage Par 22 (Default)

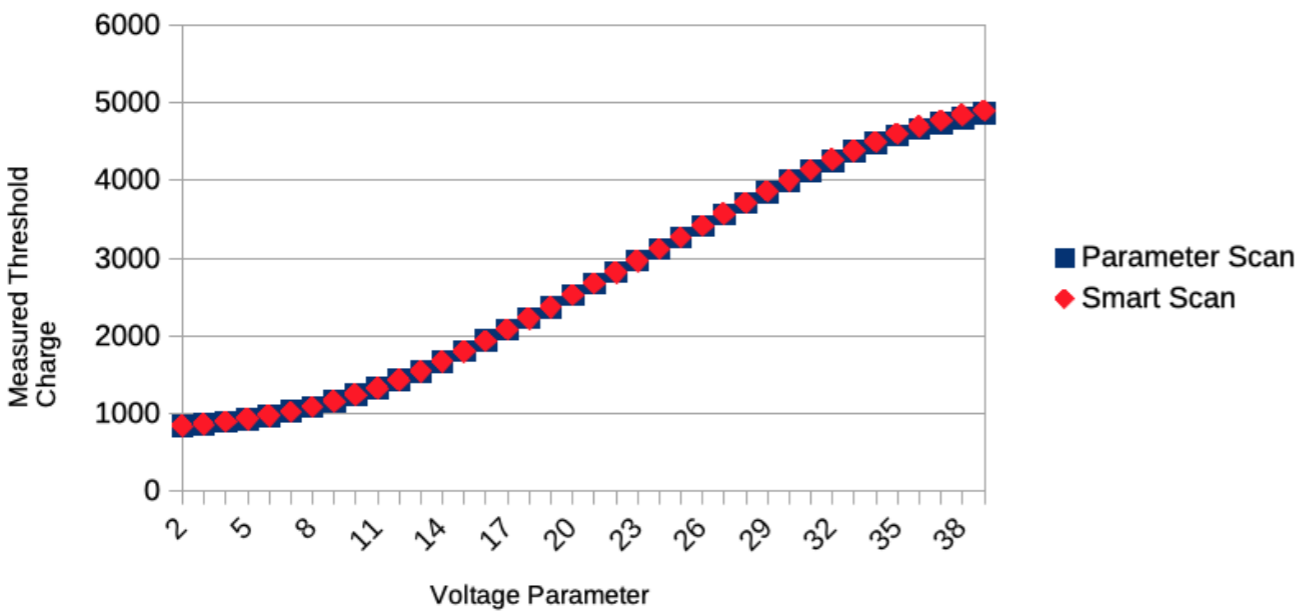


Voltage Par 39

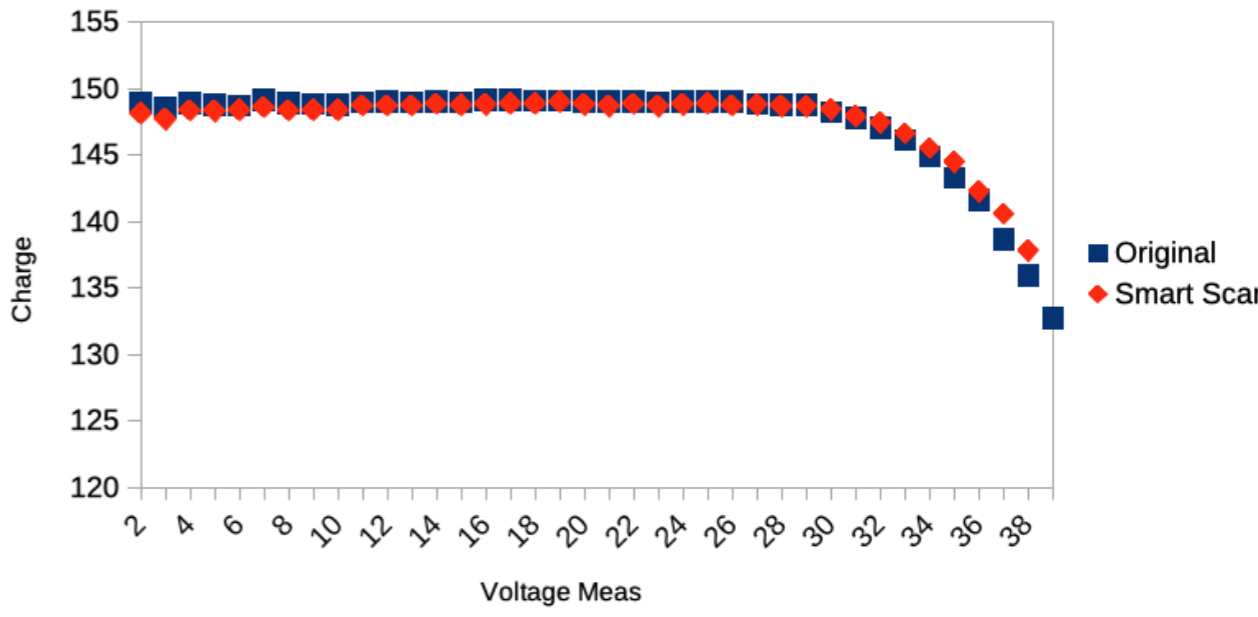
Simulation Performance

Measured/Fitted Threshold and Total Time

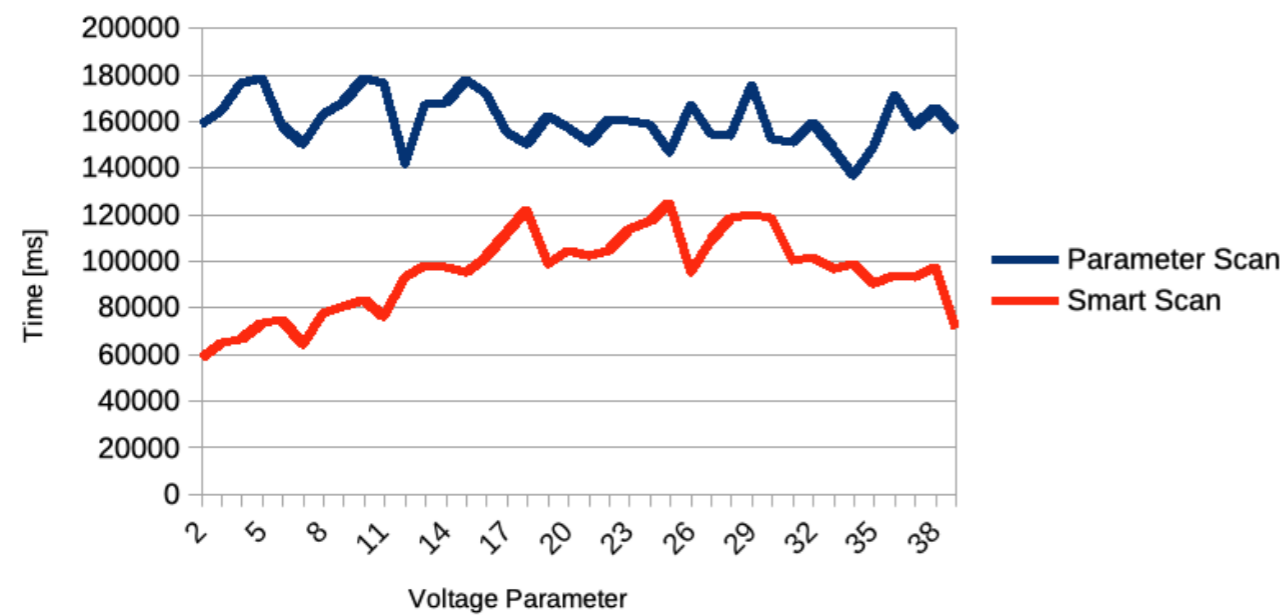
Measured Threshold



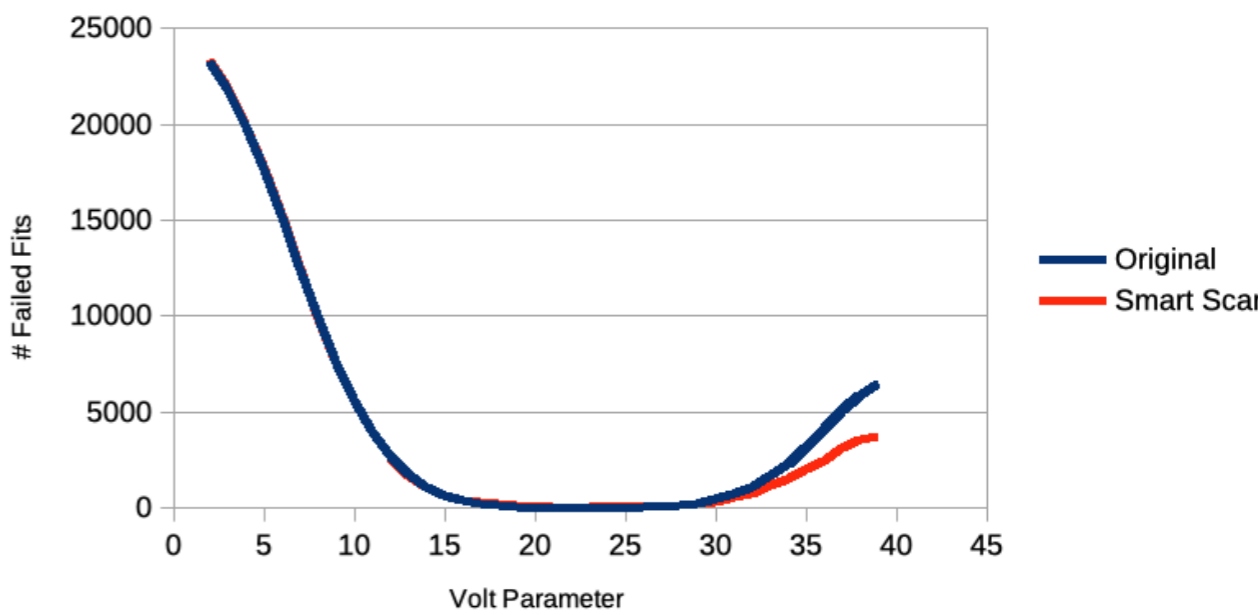
Noise Measurement



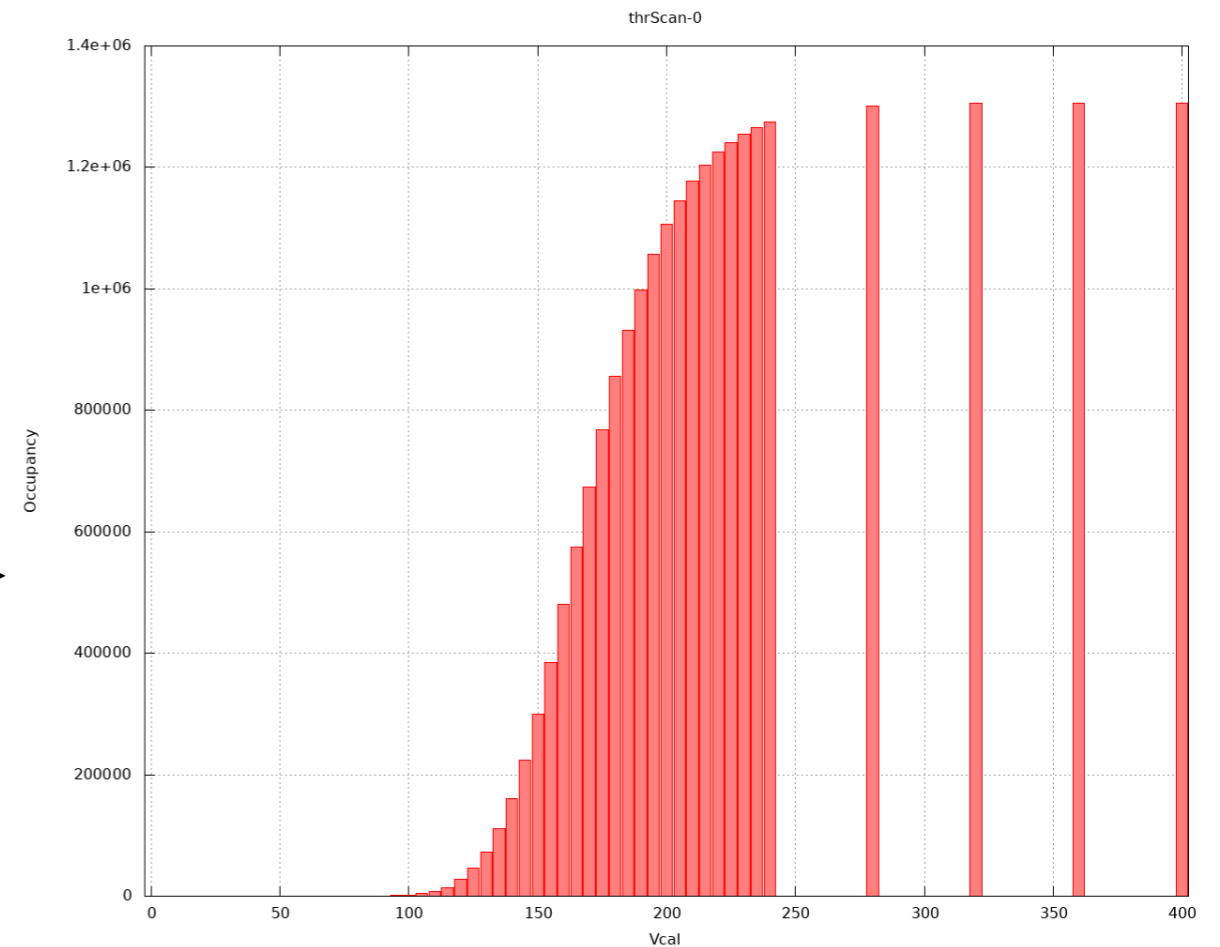
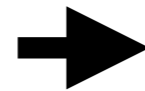
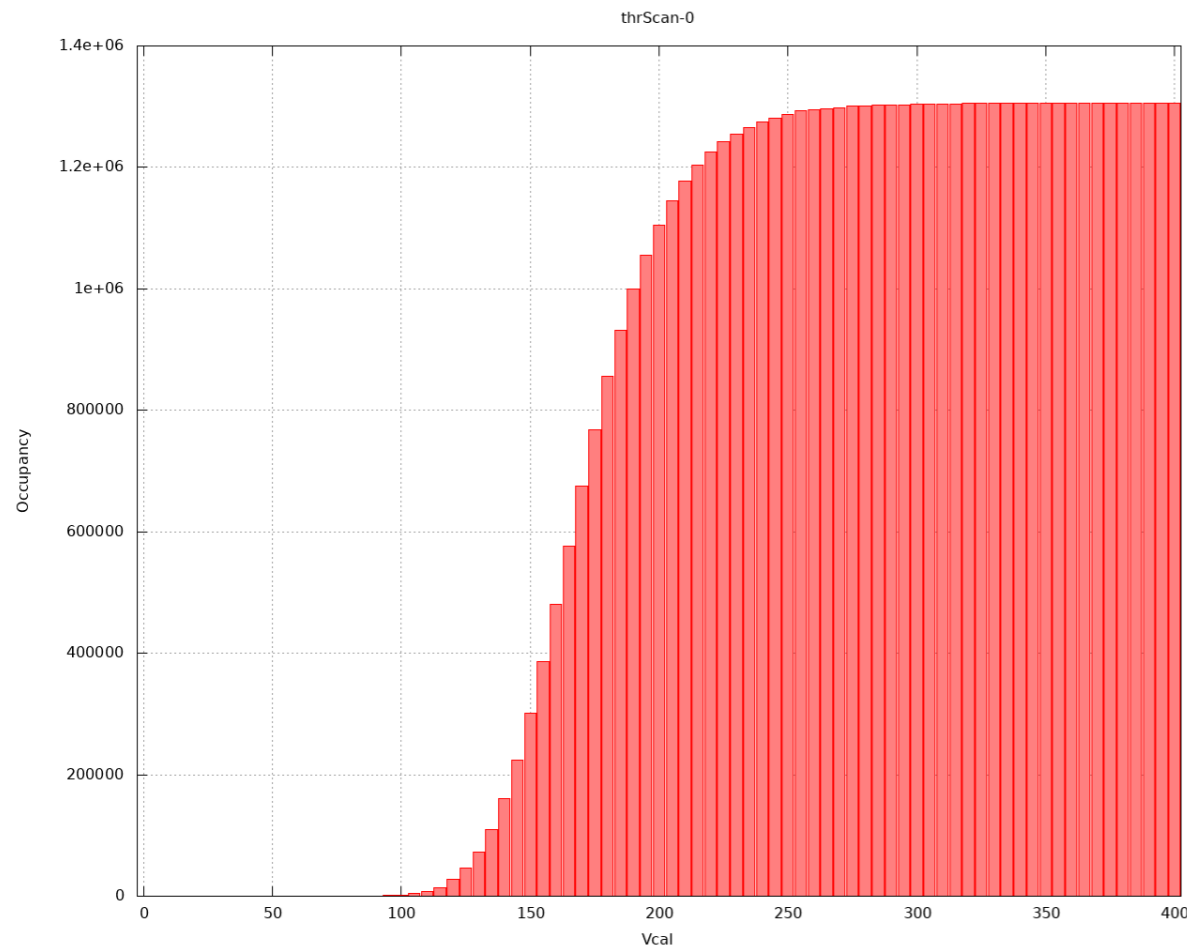
Total Time of the Threshold Scan



Number Of Failed Fits



Real Life - RD53A Performance



Standard Threshold Scan

Smart Threshold Scan

- As it should be the scans look nearly identical but the low energy regions are not scanned and a few high energy regions.

Real Life - RD53A Performance

Performance Gain By Numbers

	Single RD53A		Double RD53A's	
	Threshold Scan	Smart Scan	Threshold Scan	Smart Scan
Threshold Mean	1660.5	1645.87		
Noise Mean	40.6091	40.13		
# of failed fits	10	6		
Config Time	41	41	72	72
Scan Time	52647	25459	58474	32205
Proc. Time	2	2	2	2
Analy. Time	661	682	800	774
Total. Time	53351	26184	59348	33053

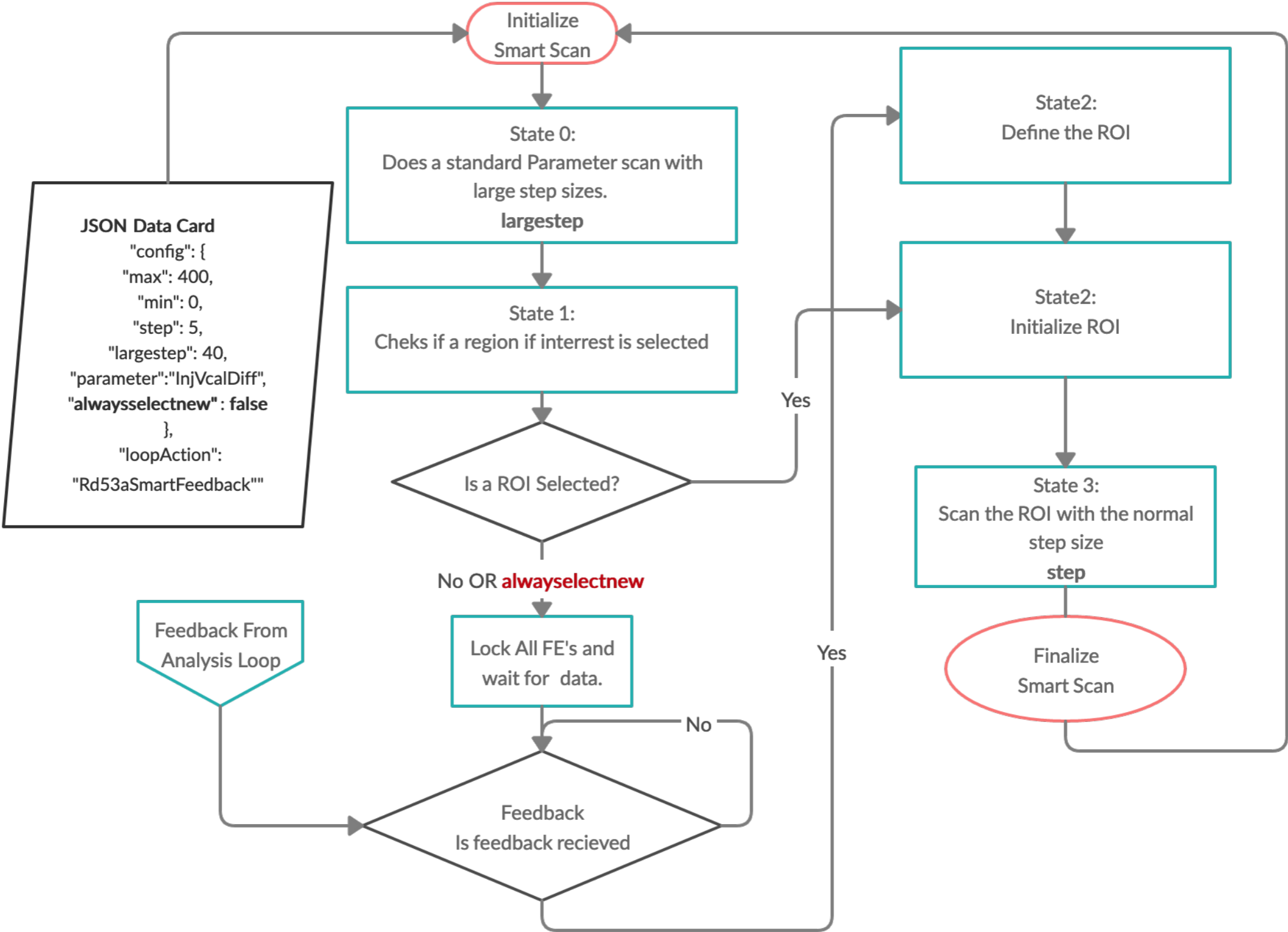
Conclusion

- The smart threshold scan is ready for use.
 - For now: ScurveFit only.
 - We want to introduce this for different fitting algorithms as well.
- The algorithm that selects the RIO is interest is easily modifiable, so we are open for any ideas.
- You can checkout the code in: SmartScan Branch.

Selecting the ROI

- The way to select the ROI in this algorithm can be easily modified.
- Currently the method is to search for the initial and final plateaus.
- This is done by identifying the first/last bin that fullfills:
 - $(\text{Bin}[\text{Last}] - \text{Bin}[\text{Current}]) / \text{Bin}[\text{Last}] < \%1$
 - $(\text{Bin}[\text{Current}] - \text{Bin}[\text{First}]) / \text{Bin}[\text{Last}] < \%1$
- Other algorithms are always welcome.

Smart Scan State Machine




Locking Mechanism

- **Inside Smart Scan:**

- When lock is called:

- All FE Mutex's are locked. Also additional lock boolean is set per FE.

```
for (auto fe : keeper->feList) {
    if (fe->getActive()) {
        std::cout<<"Locking things up now"<<std::endl;
        unsigned rx = dynamic_cast<FrontEndCfg*>(fe)->getRxChannel();
        keeper->mutexMap[rx].try_lock();
        m_LockStatus[m_RxMap[rx]]=true;
    }
}
m_StateMach=1;
```



Locking Mechanism

- **Inside Smart Scan:**

- When lock is called:
 - All FE Mutex's are locked. Also additional lock boolean is set per FE.
 - The g_stat is set to max unsigned value (4294967295).

```
for (auto fe : keeper->feList) {  
    if (fe->getActive()) {  
        std::cout<<"Locking things up now"<<std::endl;  
        unsigned rx = dynamic_cast<FrontEndCfg*>(fe)->getRxChannel();  
        keeper->mutexMap[rx].try_lock();  
        m_LockStatus[m_RxMap[rx]]=true;  
    }  
}  
m_StateMach=1;
```

```
else if (m_StateMach==1){  
    g_stat->set(this, 4294967295);  
    m_LockCount++;  
    if(verbose)  
        std::cout<<"Lock Count is at "<<m_LockCount<<std::endl;
```

Locking Mechanism

- **Inside Smart Scan:**

- When lock is called:

- All FE Mutex's are locked. Also additional lock boolean is set per FE.
- The g_stat is set to max unsigned value (4294967295).

- When system is locked:

- For every new loop, smart threshold scan just checks if all FE's booleans are unlocked.
- If they are unlocked, change the state to the new one and unlock Mutex's.

```
for (auto fe : keeper->feList) {  
    if (fe->getActive()) {  
        std::cout<<"Locking things up now"<<std::endl;  
        unsigned rx = dynamic_cast<FrontEndCfg*>(fe)->getRxChannel();  
        keeper->mutexMap[rx].try_lock();  
        m_LockStatus[m_RxMap[rx]]=true;  
    }  
}  
m_StateMach=1;
```

```
else if (m_StateMach==1){  
    g_stat->set(this, 4294967295);  
    m_LockCount++;  
    if(verbose)  
        std::cout<<"Lock Count is at "<<m_LockCount<<std::endl;
```

Locking Mechanism

- **Inside Smart Scan:**

- When lock is called:

- All FE Mutex's are locked. Also additional lock boolean is set per FE.
- The g_stat is set to max unsigned value (4294967295).

- When system is locked:

- For every new loop, smart threshold scan just checks if all FE's booleans are unlocked.
- If they are unlocked, change the state to the new one and unlock Mutex's.

- **Inside the analysis loop:**

- If (g_stat=4294967295) send feedback.

```
for (auto fe : keeper->feList) {  
    if (fe->getActive()) {  
        std::cout<<"Locking things up now"<<std::endl;  
        unsigned rx = dynamic_cast<FrontEndCfg*>(fe)->getRxChannel();  
        keeper->mutexMap[rx].try_lock();  
        m_LockStatus[m_RxMap[rx]]=true;  
    }  
}  
m_StateMach=1;
```

```
else if (m_StateMach==1){  
    g_stat->set(this, 4294967295);  
    m_LockCount++;  
    if(verbose)  
        std::cout<<"Lock Count is at "<<m_LockCount<<std::endl;
```

```
if (SmartFb!=nullptr && vcal==4294967295)// - int(vcalMax-vcalMin)%SmartFb->Ge  
{  
  
    SmartFb->feedback(this->channel,thrScan[outerIdent].get()); //Numbers don'  
    return;  
}
```

Locking Mechanism

- **Inside Smart Scan:**

- When lock is called:

- All FE Mutex's are locked. Also additional lock boolean is set per FE.
 - The g_stat is set to max unsigned value (4294967295).

```
for (auto fe : keeper->feList) {  
    if (fe->getActive()) {  
        std::cout<<"Locking things up now"<<std::endl;  
        unsigned rx = dynamic_cast<FrontEndCfg*>(fe)->getRxChannel();  
        keeper->mutexMap[rx].try_lock();  
        m_LockStatus[m_RxMap[rx]]=true;  
    }  
}  
m_StateMach=1;
```

- When system is locked:

- For every new loop, smart threshold scan just checks if all FE's booleans are unlocked.
 - If they are unlocked, change the state to the new one and unlock Mutex's.

```
else if (m_StateMach==1){  
    g_stat->set(this, 4294967295);  
    m_LockCount++;  
    if(verbose)  
        std::cout<<"Lock Count is at "<<m_LockCount<<std::endl;
```

- **Inside the analysis loop:**

- If (g_stat=4294967295) send feedback.

```
if (SmartFb!=nullptr && vcal==4294967295)// - int(vcalMax-vcalMin)%SmartFb->Ge  
{  
  
    SmartFb->feedback(this->channel,thrScan[outerId].get()); //Numbers don't  
    return;  
}
```

- **When feedback is called (Per FE):**

- Retrieve the Histogram
 - Set the lock boolean for that FE to false.

Locking Mechanism

- **Inside Smart Scan:**

- When lock is called:

- All FE Mutex's are locked. Also additional lock boolean is set per FE.
 - The g_stat is set to max unsigned value (4294967295).

```
for (auto fe : keeper->feList) {  
    if (fe->getActive()) {  
        std::cout<<"Locking things up now"<<std::endl;  
        unsigned rx = dynamic_cast<FrontEndCfg*>(fe)->getRxChannel();  
        keeper->mutexMap[rx].try_lock();  
        m_LockStatus[m_RxMap[rx]]=true;  
    }  
}  
m_StateMach=1;
```

- When system is locked:

- For every new loop, smart threshold scan just checks if all FE's booleans are unlocked.
 - If they are unlocked, change the state to the new one and unlock Mutex's.

```
else if (m_StateMach==1){  
    g_stat->set(this, 4294967295);  
    m_LockCount++;  
    if(verbose)  
        std::cout<<"Lock Count is at "<<m_LockCount<<std::endl;
```

- **Inside the analysis loop:**

- If (g_stat=4294967295) send feedback.

```
if (SmartFb!=nullptr && vcal==4294967295)// - int(vcalMax-vcalMin)%SmartFb->Ge  
{  
  
    SmartFb->feedback(this->channel,thrScan[outerIdent].get()); //Numbers don't  
    return;  
}
```

- **When feedback is called (Per FE):**

- Retrieve the Histogram
 - Set the lock boolean for that FE to false.

In built counter is implemented to prevent permanent lockdown.