GPU Acceleration and EDM Developments for the ATLAS 3D Calorimeter Clustering in the Software Trigger

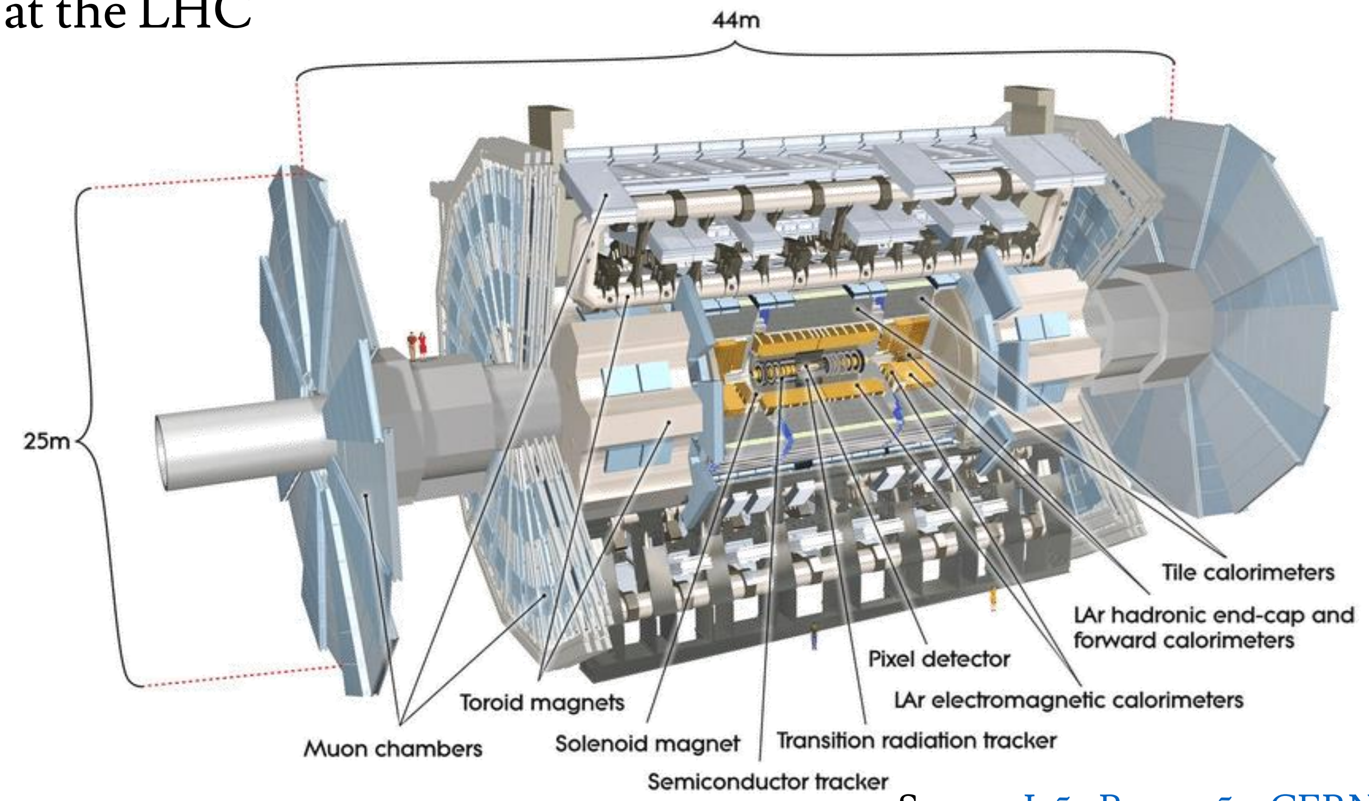
CHEP 2024

Nuno dos Santos Fernandes on behalf of the ATLAS Collaboration

Context

The ATLAS Experiment

- **A** **T**oroidal **L**HC **A**pparatu**s**[1]
- One of the two **general-purpose detectors** at the LHC
- Three layers:
  - Inner Detector
  - **Calorimeters**
  - Muon Spectrometers
- 10^8 electronic channels
  - 187652 calorimeter cells with multiple gain paths to optimize resolution *versus* dynamic range of operation


44m
25m
Tile calorimeters
LAr hadronic end-cap and forward calorimeters
Pixel detector
LAr electromagnetic calorimeters
Toroid magnets
Muon chambers
Solenoid magnet
Transition radiation tracker
Semiconductor tracker

Source: João Pequenão, CERN

[1] – *The ATLAS Experiment at the CERN Large Hadron Collider*, DOI 10.1088/1748-0221/3/08/S08003

The ATLAS Trigger

- The **ATLAS Trigger**[1] is used to **filter the detected events** to ensure a **manageable output rate**
  - **Speed *versus* accuracy trade-offs** can be relevant
- **Two stages:**
  - **Hardware-based** (**Level 1**/Level 0)
  - **Software-based** (**High-Level Trigger**/Event Filter)
- The **High-Luminosity LHC Upgrade** will **increase the luminosity**, making **event reconstruction more computationally demanding**
- The **Phase II upgrade** needed for the **High-Luminosity LHC** increases **event rate** at the software-based stage by **a factor of 10**
- This **higher computational load** requires **more computing power** available for the trigger and/or **better optimization**
- Alternative: **hardware acceleration**
  - Ongoing studies for both FPGA and **GPU acceleration**

[1] – *The ATLAS trigger system for LHC Run 3 and trigger performance in 2022*, DOI 10.1088/1748-0221/19/06/P06029

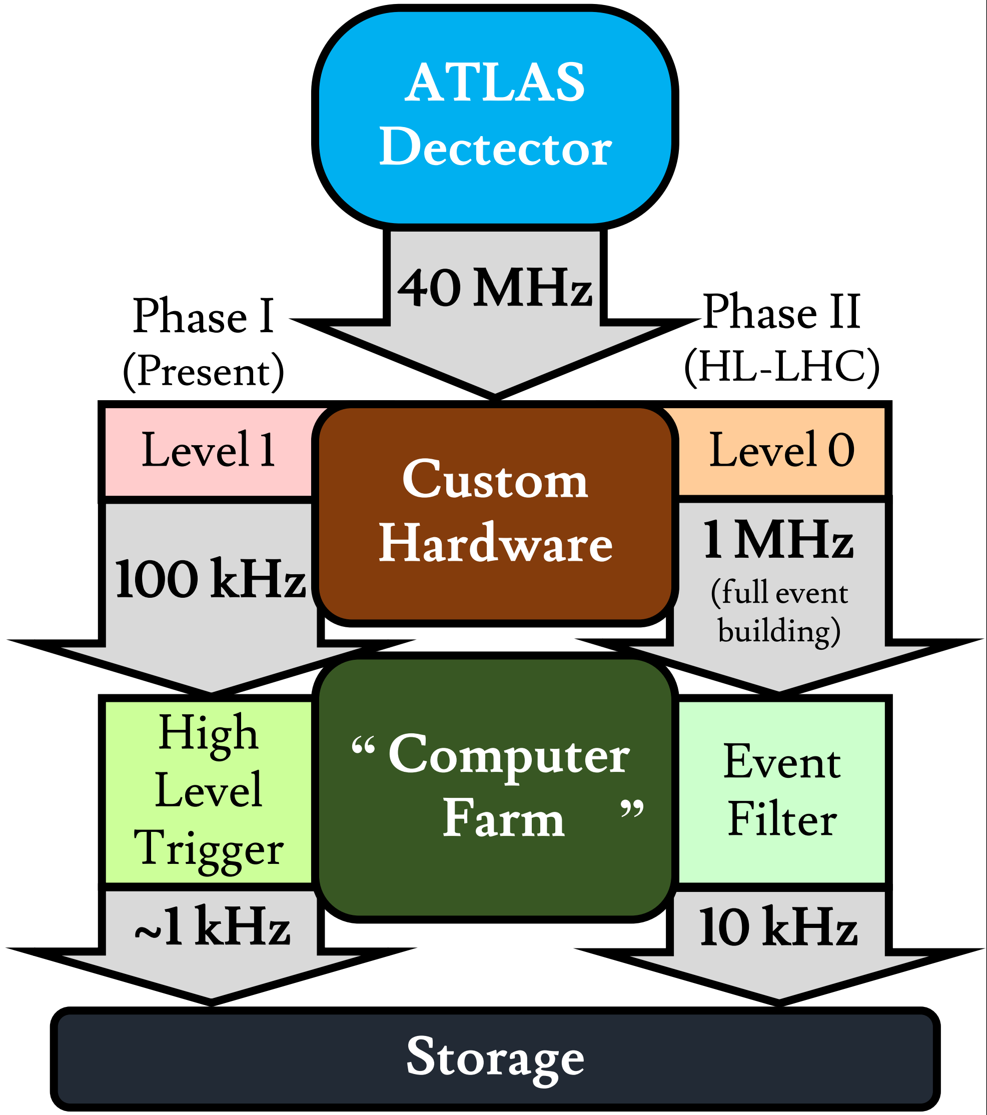
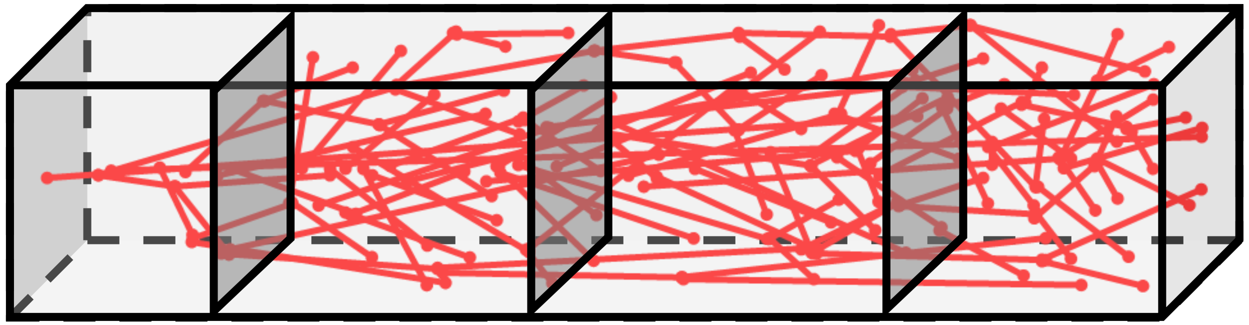


Diagram of the ATLAS Trigger System

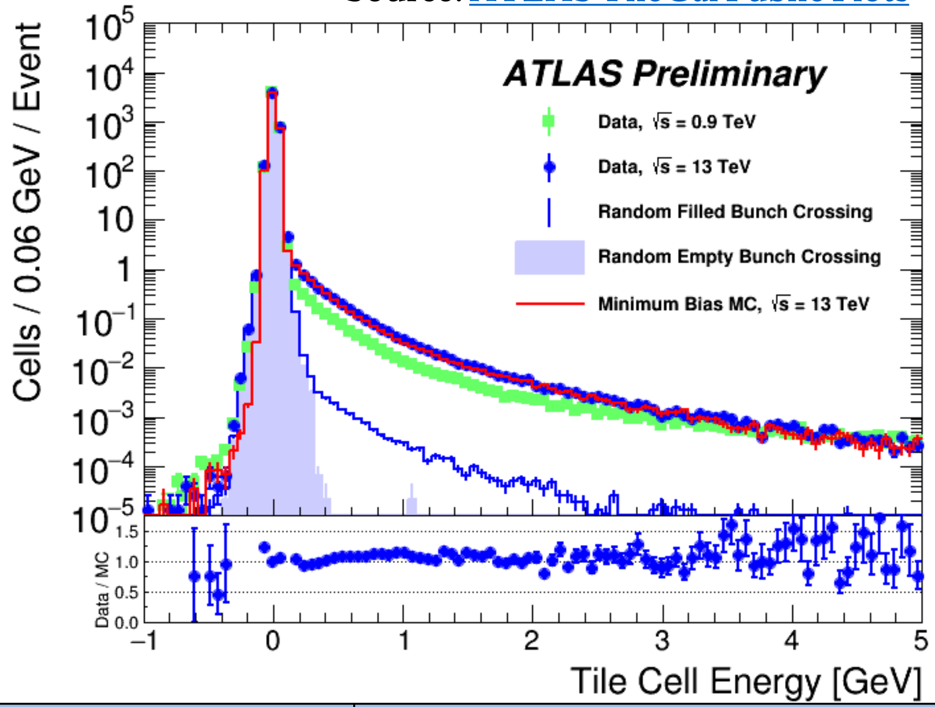
Calorimeter Reconstruction Algorithms

- Reconstruction of **showers** generated by outgoing particles in the calorimeters of the ATLAS experiment



- Showers **deposit their energy** in a finite region of space: a **calorimeter cell**
- Calorimeter cells organized in up to **28 sampling layers/regions**
- Two main sources of **noise**: electronic read-out and pile-up
  - The **noise estimate** is typically a function of the gain of the cell
  - For the **Tile calorimeter**, the electronic noise can be estimated by a **two-Gaussian model**, which involves more sophisticated computations (inverse error function of error functions)

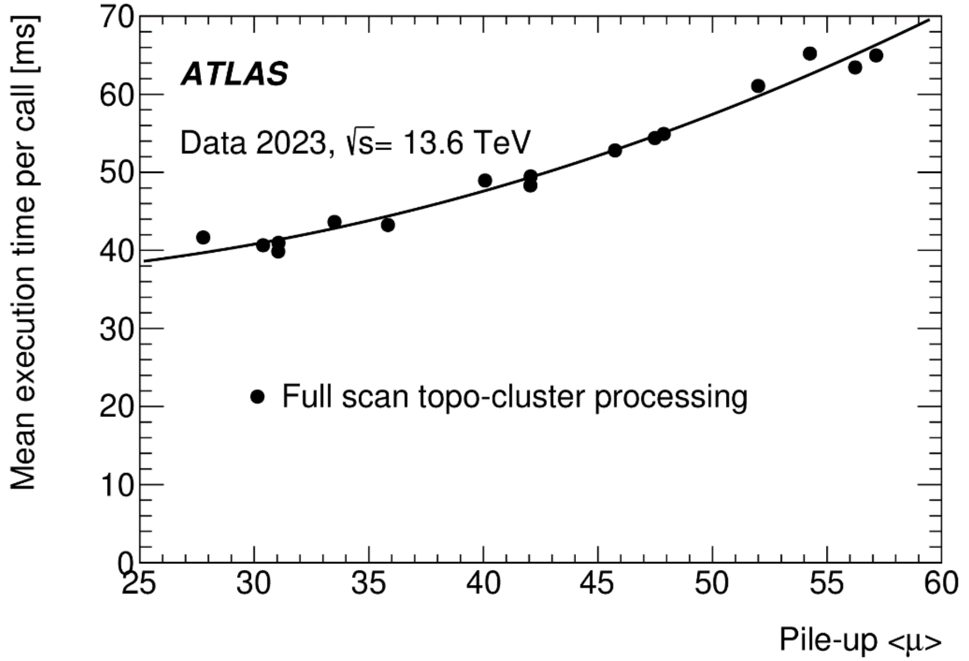
Source: [ATLAS TileCal Public Plots]



Topological and Topo-Automaton Clustering

Topological Clustering

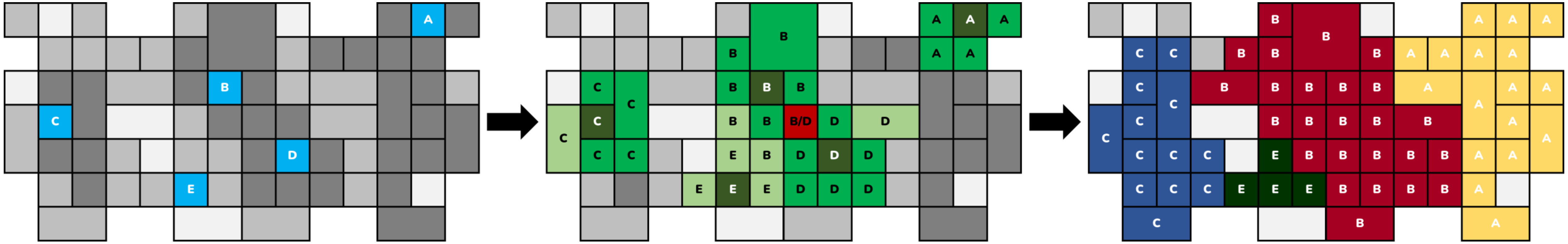
- **Topological Clustering** is the currently used approach for **calorimeter reconstruction** in ATLAS
- **Main criterion** for assigning cells to the clusters is the **signal-to-noise ratio** (SNR) of the **energy deposition**
  - This is essentially the **relevance** of the **contribution** of each cell to the **reconstruction of the underlying physics**
- Clustering typically groups up **several tens of cells**, **some clusters** may be **significantly larger**
- **Several hundred to a few thousand clusters** per event, depending on the **physical process**
- Significant **dependence on the number of collisions per bunch crossing** (μ) in terms of the **execution time**



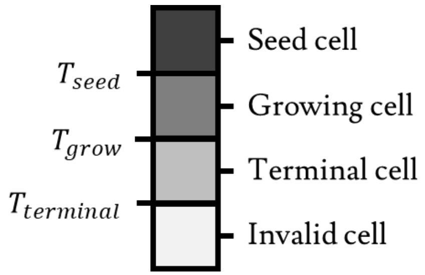
Source: *The ATLAS trigger system for LHC Run 3 and trigger performance in 2022*, DOI 10.1088/1748-0221/19/06/P06029

Topological Clustering

- Two main algorithmic stages:
  - **Cluster growing**: iteratively **assign cells** to clusters based on the **SNR**



➢ Classify cells as **seed**, **growing** or **terminal**

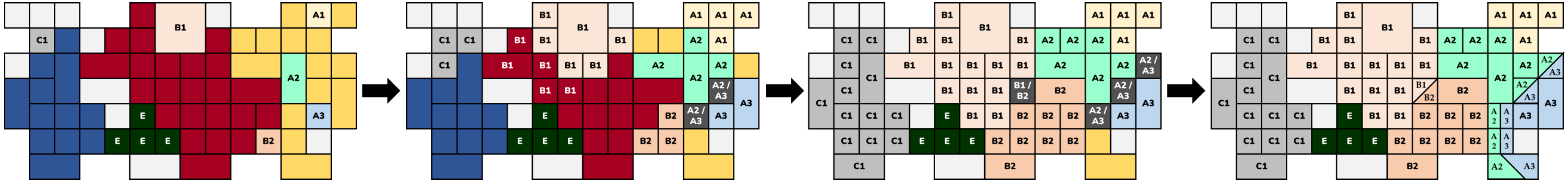


➢ Clusters **grow out from the seeds** to their **neighbouring cells** in an order defined by the **SNR of the seed**

➢ Clusters are **merged** if they **touch through growing cells**

Topological Clustering

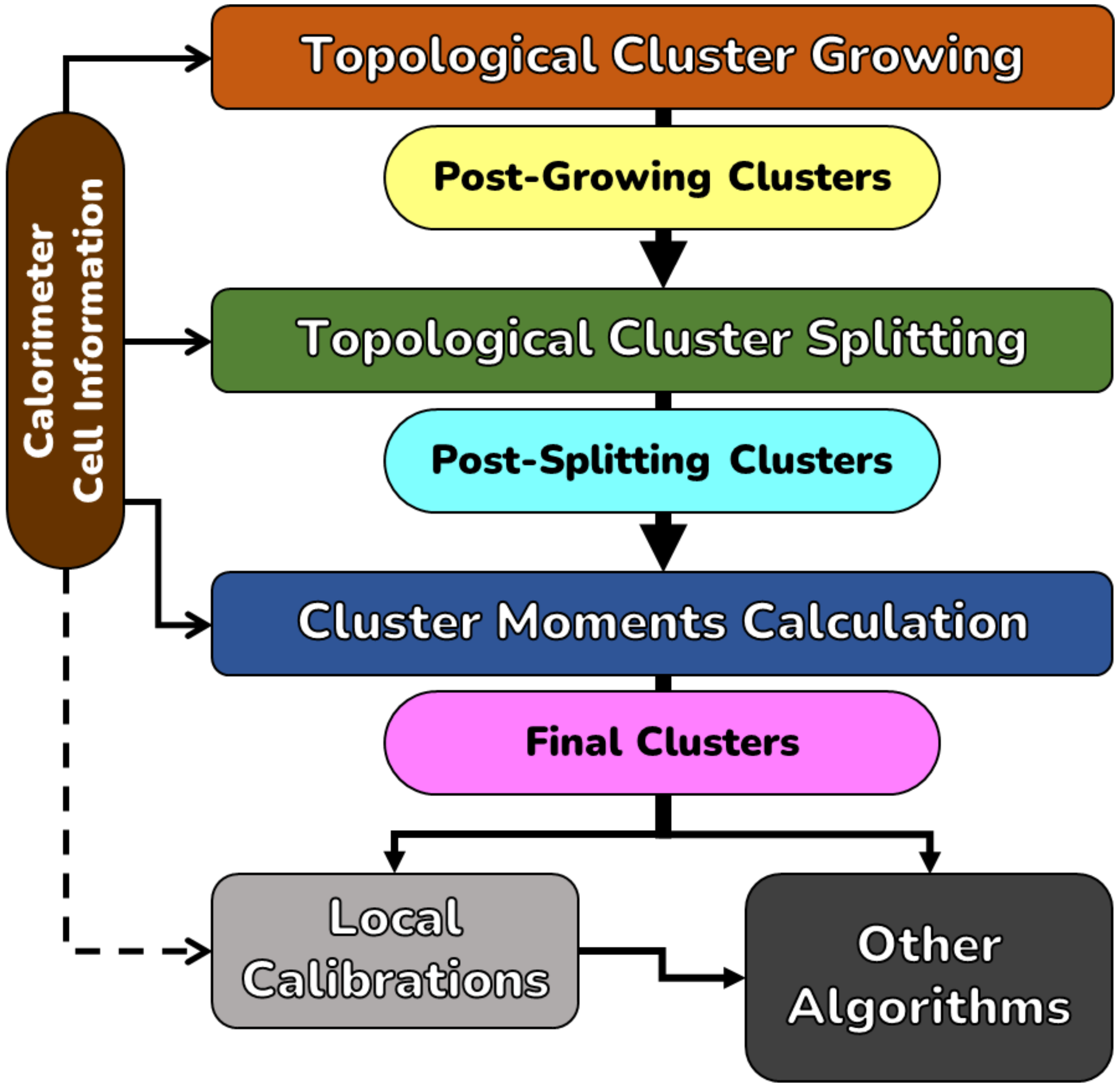
- Two main algorithmic stages:
  - **Cluster splitting**: split the clusters around **local maxima of the energy** to distinguish different objects travelling in the same direction



- ➢ Identify **local maxima**
- ➢ **Exclude maxima** from certain regions of the detector that **overlap** in certain directions to favour **layers with greater radiation depth**
- ➢ Start **growing the clusters** to **neighbouring cells** in an order defined by the **energy of the cells**
- ➢ Cells that can **belong to more than one maximum** are **shared**
- ➢ **Shared cells** grow clusters **only in the end** and are weighted based on the energy and distance to the centroid: $w_1 = \frac{E_1}{E_1 + r\,E_2}$, $w_2 = 1 - w_1$, with $r = e^{d_1 - d_2}$

Topological Clustering

- A final step involves **calculating several cluster moments**, based on weighted sums of the properties of the cells and functions thereof
  - Some of the moments require **calculating eigenvalues and eigenvectors** of a 3×3 matrix
- Some **local calibrations** may be applied to the clusters, based on the cluster moments and the constituent cells
- Topological Clustering is the **most computationally demanding** algorithm of the calorimeter reconstruction and among the **top 20th most computationally demanding algorithms** within the ATLAS trigger
  - The **same implementation** is used for both **online and offline reconstruction** (with potentially different configuration)
- Topological clustering, as described, is **not accelerator-friendly**: a **different algorithmic approach** is needed


Calorimeter Cell Information
Topological Cluster Growing
Post-Growing Clusters
Topological Cluster Splitting
Post-Splitting Clusters
Cluster Moments Calculation
Final Clusters
Local Calibrations
Other Algorithms

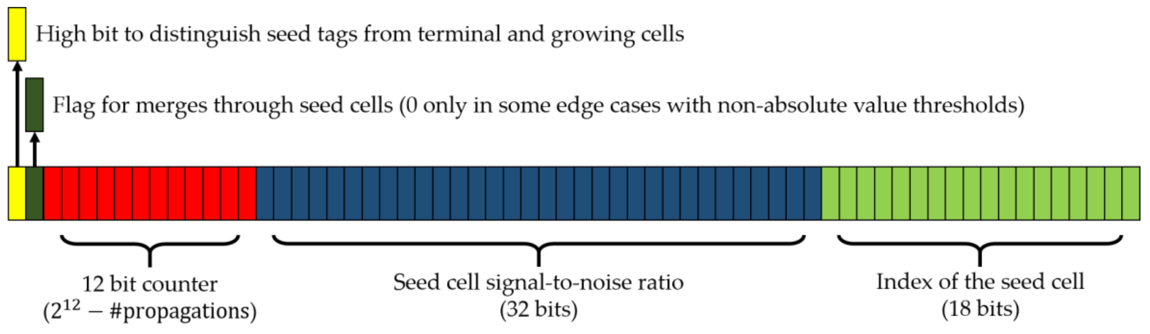
Limitations of Topological Clustering

- The **clusters** are expressed as **lists of cells**[1] which must be **resized** as they grow
- The algorithm itself involves **keeping track of multiple lists of cells**[1], especially for **cluster splitting**
- **Resizing** a container is **difficult to do in parallel**, and it goes **against the memory model** of both GPUs and FPGAs
- For a more **parallel-friendly** implementation, we can instead **mark the cells** that belong to each cluster with a "**tag**"
    - By constructing these **tags** appropriately, the **sorting steps can be skipped** entirely: floating point numbers that follow the **IEEE-754 standard** can be put in a "**total ordering**" where the **bit patterns**, interpreted as integers, are **ordered in the same way** as the original floating point numbers
    - By defining a **set of rules** for how these tags are propagated **from a cell to its neighbours**, one can replicate the entire behaviour of the iterative parts of cluster growing and cluster splitting while only **considering each pair of neighbours independently from each other** (potentially in parallel, as long as tag updates are thread-safe)
- Since we have both a **state** for each cell and can specify the **rules for how that state changes** based on the neighbourhood, this is equivalent to a **cellular automaton**, hence **Topo-Automaton Clustering**

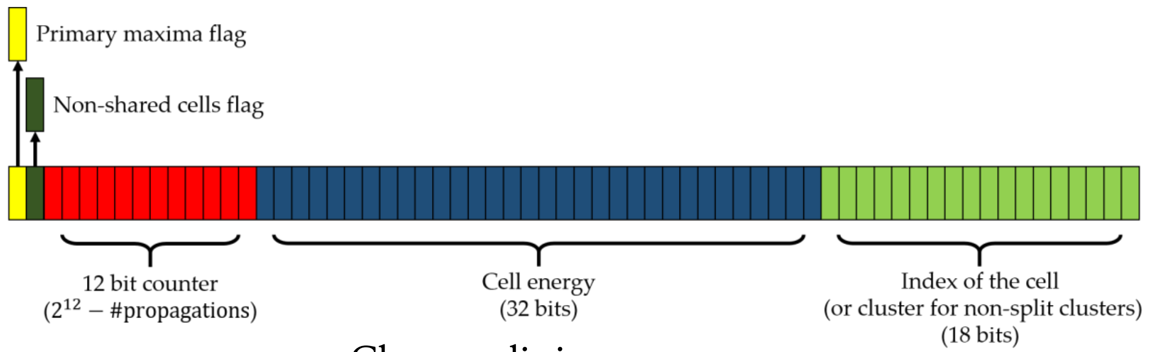
[1] – "List" is used here in the sense of an ordered collection of items; specifically, they correspond to dynamically allocated arrays, or "vectors" in C++.

Topo-Automaton Clustering

- **Cluster tags** are **64-bit integers** with **specific structure**:


High bit to distinguish seed tags from terminal and growing cells
Flag for merges through seed cells (0 only in some edge cases with non-absolute value thresholds)
12 bit counter (2^{12} – #propagations)
Seed cell signal-to-noise ratio (32 bits)
Index of the seed cell (18 bits)

Cluster growing tag structure

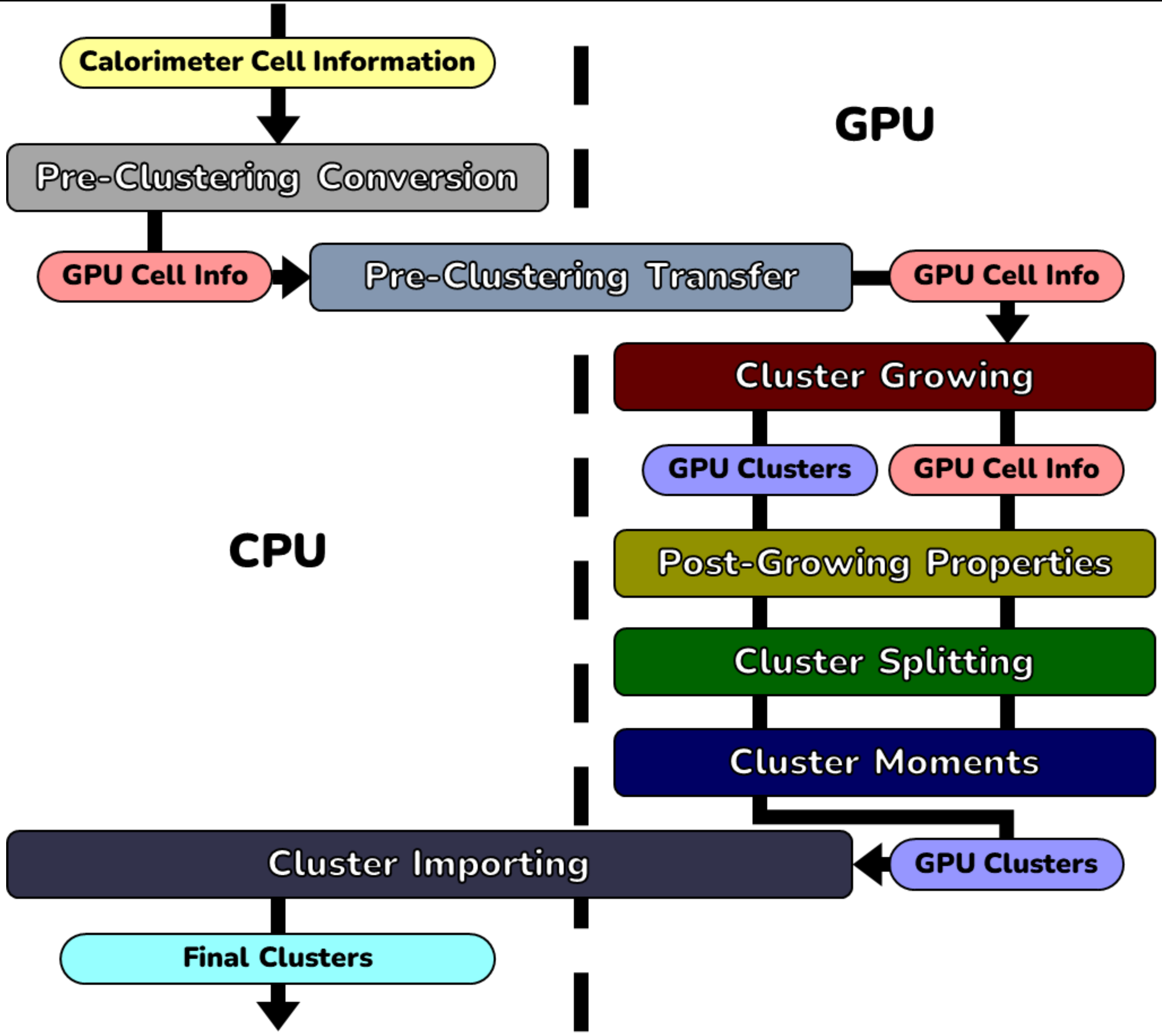

Primary maxima flag
Non-shared cells flag
12 bit counter (2^{12} – #propagations)
Cell energy (32 bits)
Index of the cell (or cluster for non-split clusters) (18 bits)

Cluster splitting tag structure

- The tags are **propagated through pairs of neighbouring cells** satisfying the conditions for clusters to expand
  - We handle each **pair of cells in parallel**, using appropriate **atomic operations** when needed
- **Additional logic** (e. g. keeping a cell to cluster index table) **reduces the number of iterations**
- All necessary **temporary information stored** in the same block of memory meant to hold the **cluster moments** (calculated only at the end), **everything can be pre-allocated**
  - Total **per event memory** footprint is **~80 MB** (per CPU thread)
  - Cell geometry, neighbourhood relations and noise constants represent **~100 MB** of **constant information**

Topo-Automaton Clustering

- Growing, splitting and moments calculation fully implemented in the GPU using CUDA
- **100% agreement in cell assignment** can be achieved between CPU and GPU with appropriate options (e. g. not using the two-Gaussian noise model)
  - Differences without these options are also **fully understood** (due to **indeterminacies** in the CPU and **floating point accuracy** issues, mostly)
- **Basic cluster properties** (e. g.: energy, η, φ) yield **similar values** (within floating point accuracy)
- Some **cluster moments** have **greater differences** due to accumulated and **compounded floating point errors** (there are calculated values that depend on calculated values that depend on calculated values…)
- The **data structures** used in the CPU part of the code **cannot be used directly** in the GPU, so we need to **convert between the two representations**

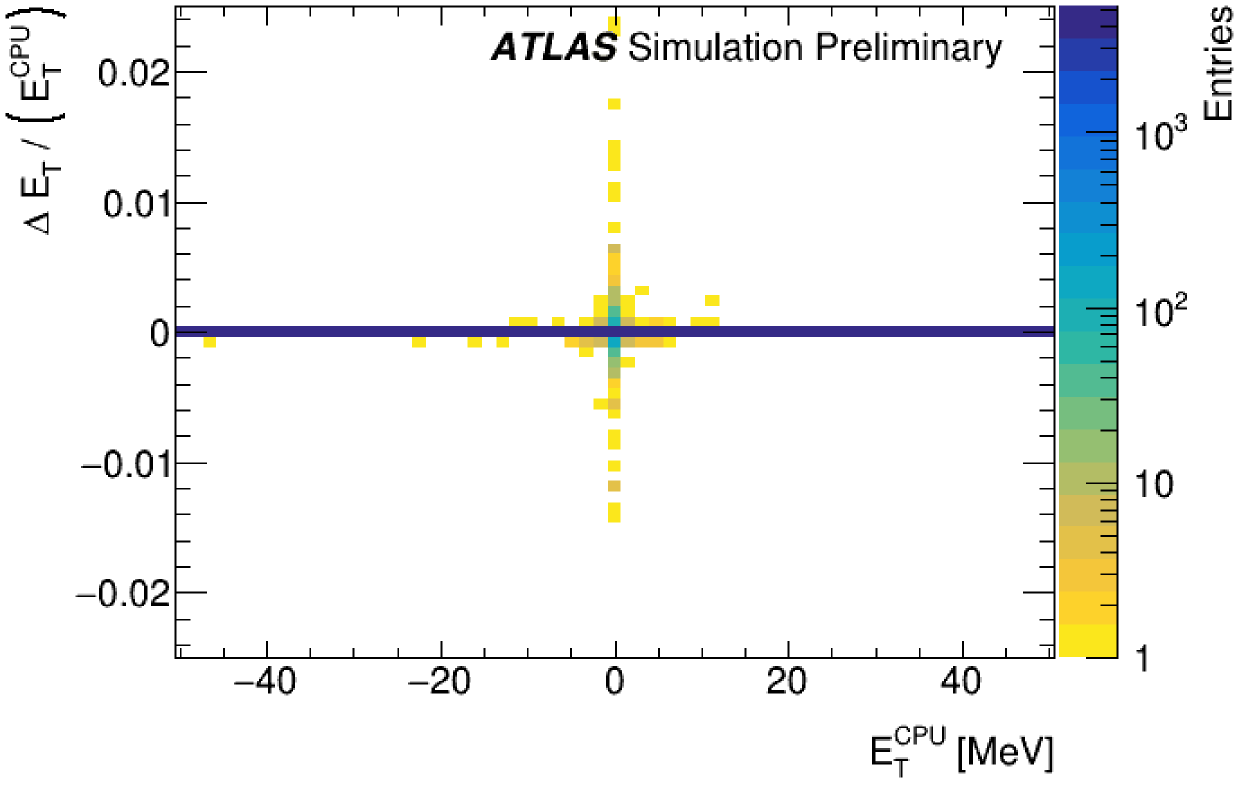


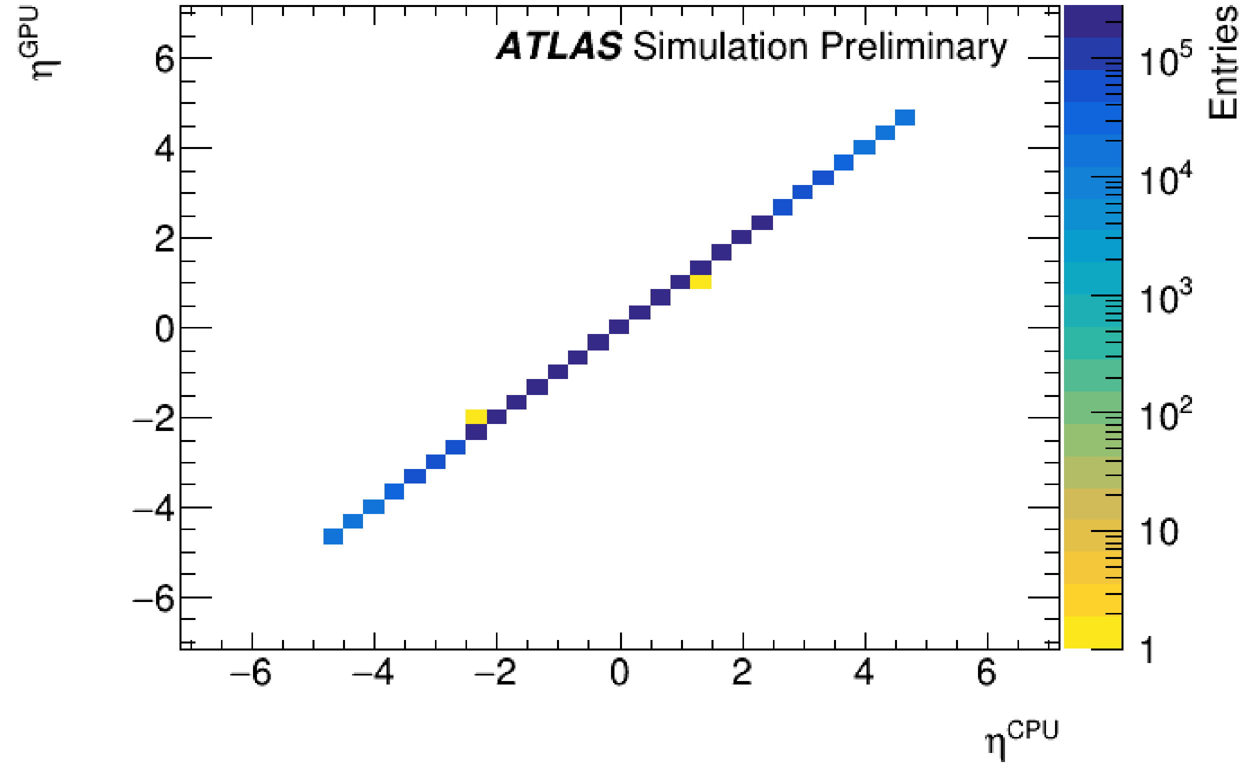
Comparison of the Cluster Properties

Relative error in the calculated transverse energy of the clusters as a function of the CPU reference value

Comparison of the calculated pseudo-rapidity (η) of the clusters in the CPU and GPU implementations

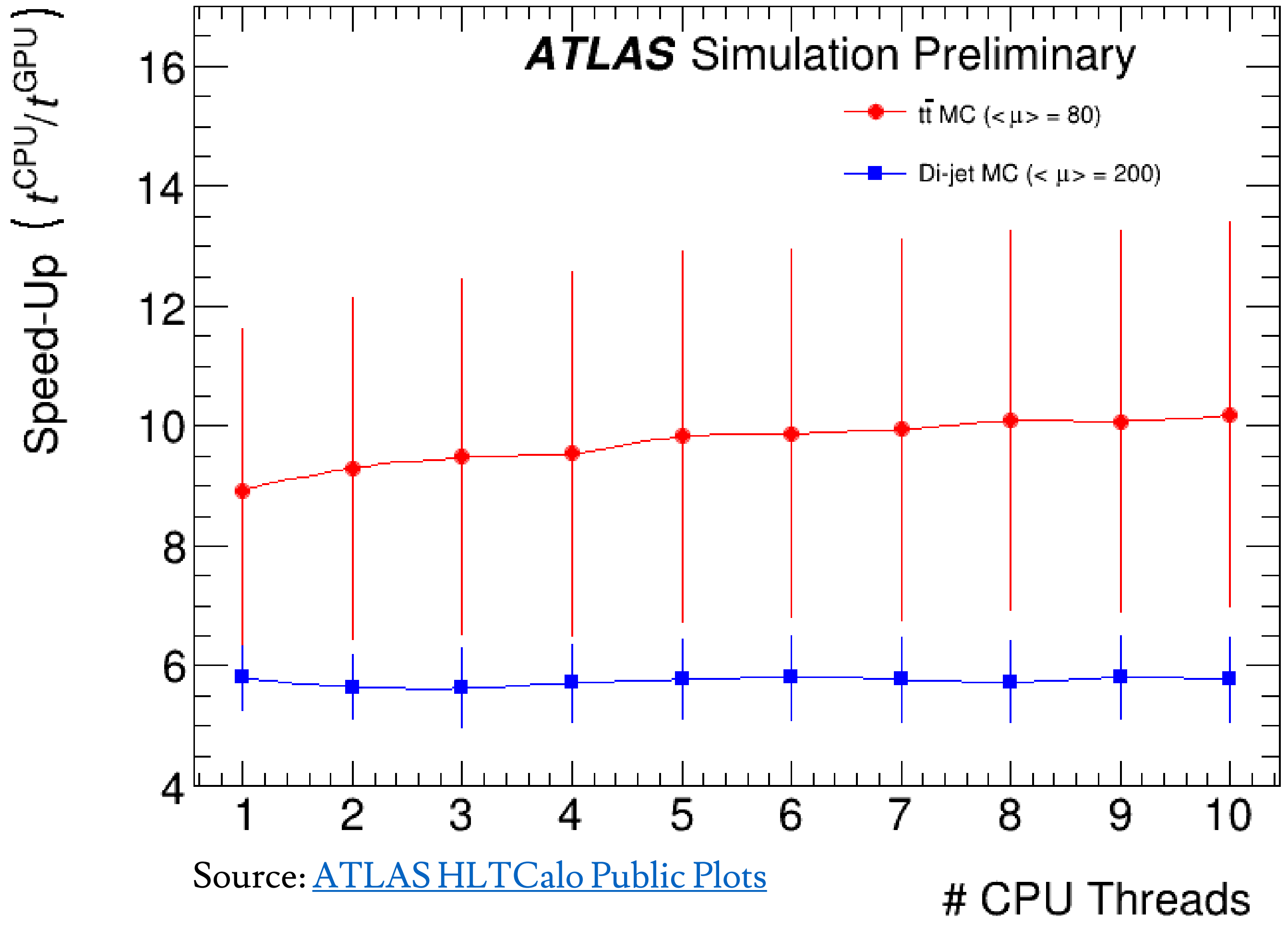
Source: ATLAS HLTCalo Public Plots





Results show an **excellent agreement** between CPU and GPU, with only **floating point accuracy issues** caused by the **different order of operations** due to the inherently unpredictable timing of the parallel execution on a GPU.

Speed-up from GPU Acceleration in Relation to the CPU Implementation



Source: ATLAS HLTCalo Public Plots

We currently achieve a **speed-up of ~5.9 for di-jets, ~8.9 for $t\bar{t}$**, considering all data conversions and transfers.
The speed-up depends on the complexity of the event (number and size of the clusters), mostly due to CPU scaling.

Breakdown of GPU Execution Times

- **Main bottleneck: converting the GPU data structures** representing the clusters **back to CPU-compatible structures**

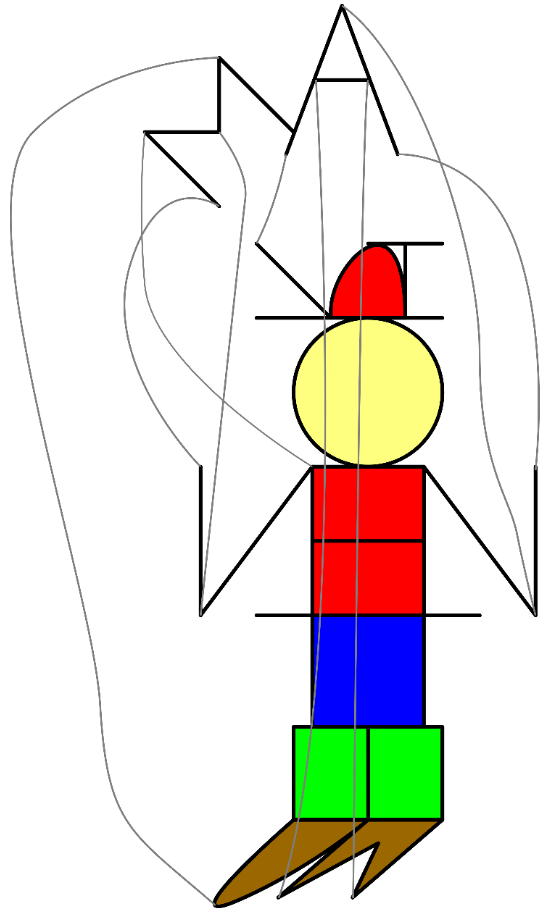
Step		t-$\bar{t}$ Events			Jet Events		
		Time (µs)	Fraction of Total Time		Time (µs)	Fraction of Total Time	
Pre-Clustering Conversion		1441 ± 225	$9.24 \pm 1.58\%$		1128 ± 88	$13.24 \pm 1.14\%$	
Pre-Clustering Transfer		266 ± 8	$1.71 \pm 0.18\%$		248 ± 15	$2.92 \pm 0.30\%$	
Growing	Cell Classification	61 ± 2	$15.76 \pm 1.77\%$	$2.53 \pm 0.35\%$	56 ± 3	$20.38 \pm 1.30\%$	$3.24 \pm 0.25\%$
	Neighbour Pair Creation	159 ± 8	$40.68 \pm 3.23\%$		114 ± 6	$41.45 \pm 1.98\%$	
	Tag Propagation	175 ± 46	$43.55 \pm 4.87\%$		106 ± 13	$38.17 \pm 2.59\%$	
	Total	396 ± 53	—		276 ± 16	—	
Post-Growing Property Calculation		74 ± 22	$0.47 \pm 0.14\%$		55 ± 2	$0.65 \pm 0.05\%$	
Splitting	Neighbour Pair Creation	409 ± 28	$29.23 \pm 2.73\%$	$9.02 \pm 1.16\%$	287 ± 14	$33.80 \pm 1.62\%$	$9.97 \pm 0.57\%$
	Local Maxima Identification	88 ± 7	$6.25 \pm 0.53\%$		57 ± 3	$6.77 \pm 0.33\%$	
	Secondary Maxima Exclusion	229 ± 16	$16.48 \pm 2.25\%$		230 ± 14	$27.15 \pm 2.11\%$	
	Main Tag Propagation	642 ± 194	$44.44 \pm 5.46\%$		236 ± 49	$27.53 \pm 3.54\%$	
	Finalization	50 ± 5	$3.60 \pm 0.31\%$		40 ± 3	$4.75 \pm 0.27\%$	
	Total	1417 ± 226	—		851 ± 67	—	
Cluster Moments		1422 ± 134	$9.07 \pm 0.58\%$		889 ± 52	$10.43 \pm 0.51\%$	
Post-Clustering Transfer + Conversion		10679 ± 137	$67.77 \pm 2.59\%$		5094 ± 690	$59.27 \pm 2.26\%$	
Total		15724 ± 1630	—		8565 ± 847	—	

- Potential improvements by **offloading ~3 ms of the conversion** to the GPU, but **the bottleneck would remain** given the constraints of **pre-existing CPU data structures** used in **other portions of the code**
- Ongoing effort to develop a **general solution** for an **Event Data Model** (EDM) with **CPU and GPU compatibility** requiring **minimal boilerplate**, with the possibility of providing an **API that matches pre-existing code**

EDM Developments

EDM Developments: Marionette

- Marionette: **M**emory **A**bstracted **R**epresentation with **I**nterfaces in **O**bjects **N**ecessitating **E**xtensively **T**emplated **T**ypes **E**DM
- Goal: provide a **more general solution** to **handle data structures** that are meant to be **usable** on a **CPU** and a **GPU** (or a **hardware accelerator** in general) with a "**single source of truth**"
  - User-friendly, **array-of-structs interface** over the **underlying struct-of-arrays**
  - **Arbitrarily extensible interface** to enable **compatibility with existing code**
  - All the interface composition and data description handled at **compilation time: no runtime polymorphism**, the basic data structures are (almost) trivially copyable
  - No need to explicitly define CPU and GPU data structures and transfers "by hand", but specializations by the end user are fully possible
- Basic idea: **decouple** the description of the **data to be stored** from the way it which it will be **laid out in memory**: the **data description is used to define the final data representation**
  - The user simply provides a **list of "properties"** and specifies the **layout**
  - Strategic usage of macros allows **injecting the name of the property** as a part of the interface generated from it, for **intuitive getter/setter** behaviour: `clusters[i].eta()`
- Current status: the implementation **works as intended** and **generates the same assembly** as the equivalent handwritten structures, even for a complex example with GPU usage
  - Work in progress to reduce compilation times and further extend functionality



Marionette GPU Usage Example

```
//Declare two float properties with up to 256 entries, called 'energy' and 'time', associated with each entry of the collection.
MARIONETTE_DECLARE_PER_ITEM_PROPERTY_SIZED(energy, Energy, 256, float);        //provides x.energy() and x.setEnergy()
MARIONETTE_DECLARE_PER_ITEM_PROPERTY_SIZED(time, Time, 256, float);            //provides x.time() and x.setTime()
struct Foo : Marionette::InterfaceDescription::NoObject {
  template <class Final, class Layout> struct ObjectFunctions {                //Define a property that adds a function
    int foo() const { return 42; }                                            //to objects, but not the collection.
  };
};
struct Bar : Marionette::InterfaceDescription::NoObject {
  template <class Final, class Layout> struct CollectionFunctions {            //These functions can refer to other functions
    void bar() { static_cast<Final *>(this)->energy()[0] *= 1.21e9f; }        //of the final collection or object via CRTP...
  };
};
//Define the list of properties to be used (the ones we defined earlier)
using ExampleProperties = Marionette::InterfaceDescription::PropertyList<Energy, Time, Foo, Bar>;
//Define the actual classes that hold memory as a struct-of-arrays with a fixed maximum size
template <class Context> using OurCollection =
    Marionette::Collections::Collection<Marionette::LayoutTypes::DynamicStructInContext<Context, int>, ExampleProperties>;
using CPUCollection = OurCollection<Marionette::MemoryContexts::CUDAHostPinned>;   //Pinned CPU memory for faster transfers
using GPUCollection = OurCollection<Marionette::MemoryContexts::CUDAStandardGPU>;  //Normally allocated GPU memory
//Finally, the implementation:
CPUCollection coll(42);                                                        //Instantiate a collection of 42 elements
std::vector<float> desired_times(50, 10.f);                                    //Instantiate a vector for initialization
coll.time() = desired_times;                                                   //coll.time() behaves as a vector of times
GPUCollection gpu_coll = coll;                                                 //Copy-construct a new collection, held on the GPU
some_kernel<<<4, 64>>>(Marionette::Collections::pass_by_value(gpu_coll));      //Pass that collection to a GPU kernel
coll = gpu_coll;                                                               //Copy-assign back to the CPU collection
```

Summary and Future Efforts

Summary and Future Efforts

- Topo-Automaton Clustering **fully implemented** and working, for **cluster growing**, **cluster splitting** and **cluster moments calculation**, with configurability on a par with the CPU implementation (essentially, **drop-in replacement**)
- A very significant **speed-up** was found (factor of **~5.9 for di-jet events, ~8.9 for denser $t\bar{t}$ events**)
  - This despite a significant portion of the GPU event processing time **(60~70%)** is spent in **data conversions**
  - Efforts to improve this bottleneck are under way, but it is **a complex issue** due to the well-established CPU structures and the nature of the underlying EDM approach
- **Marionette** may provide a **general solution** to **mitigate the data structure conversion overhead**
  - Some **technical hurdles** still to be overcome, but the current iteration of **the Marionette framework *works***
  - **Integration** with the **current implementation** of Topo-Automaton Clustering to follow
- Preliminary work started on assessing the feasibility of performing at least some **cluster calibrations** on the GPU
- **Lessons learned** and **experience gained** from this development have fed back into general hardware acceleration-related development within ATLAS and in particular the ATLAS Trigger
- A **final decision** on **using this approach in the ATLAS Trigger** depends on a **general technical assessment** of the feasibility and/or performance of **GPU-accelerated algorithms**, being scheduled for **next year**
  - The approach is **also being considered** for **offline reconstruction** on **grid sites where GPUs are available**

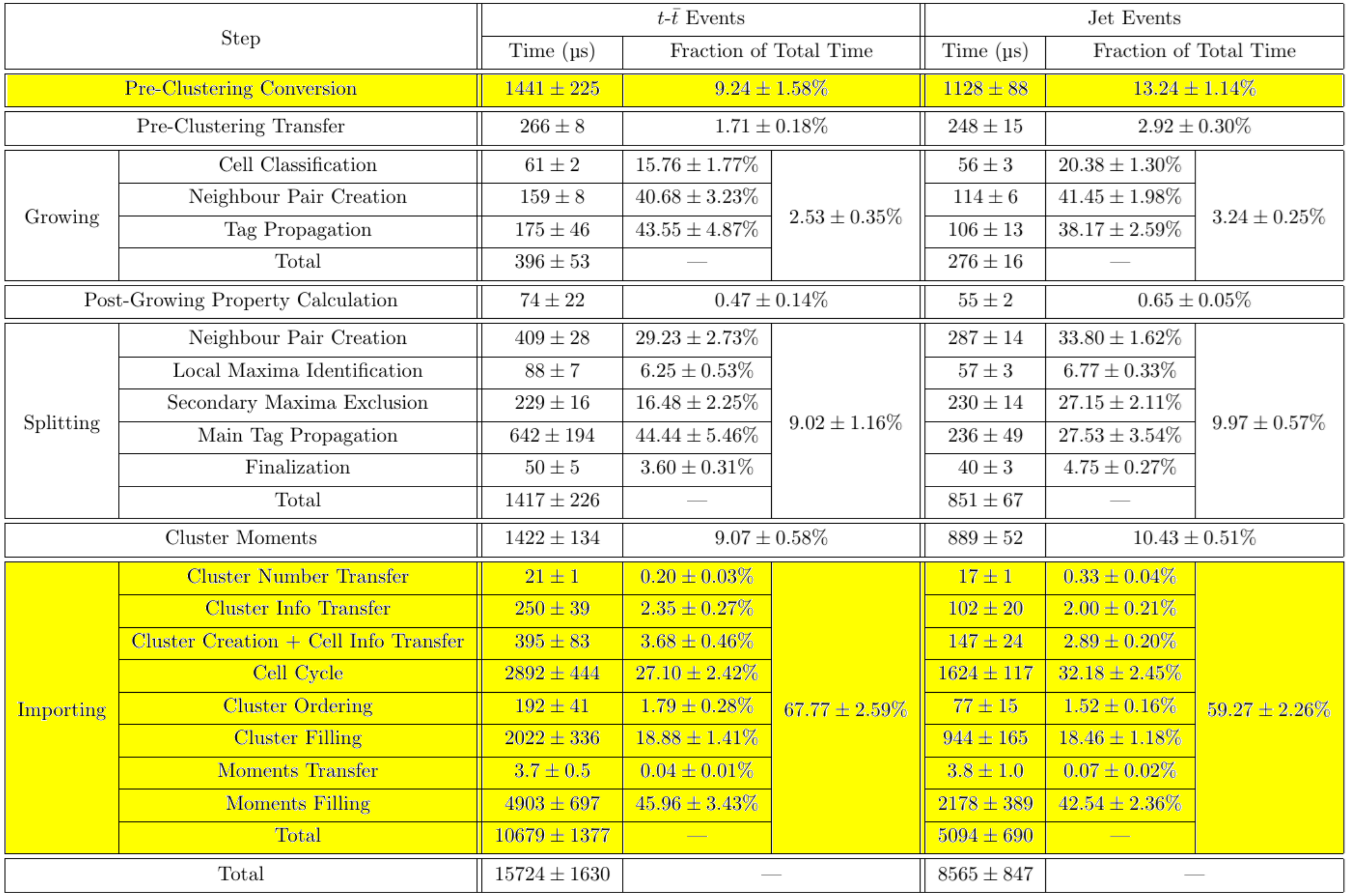
Thank you for your attention!

Backup Slides

Conditions of the Benchmarking

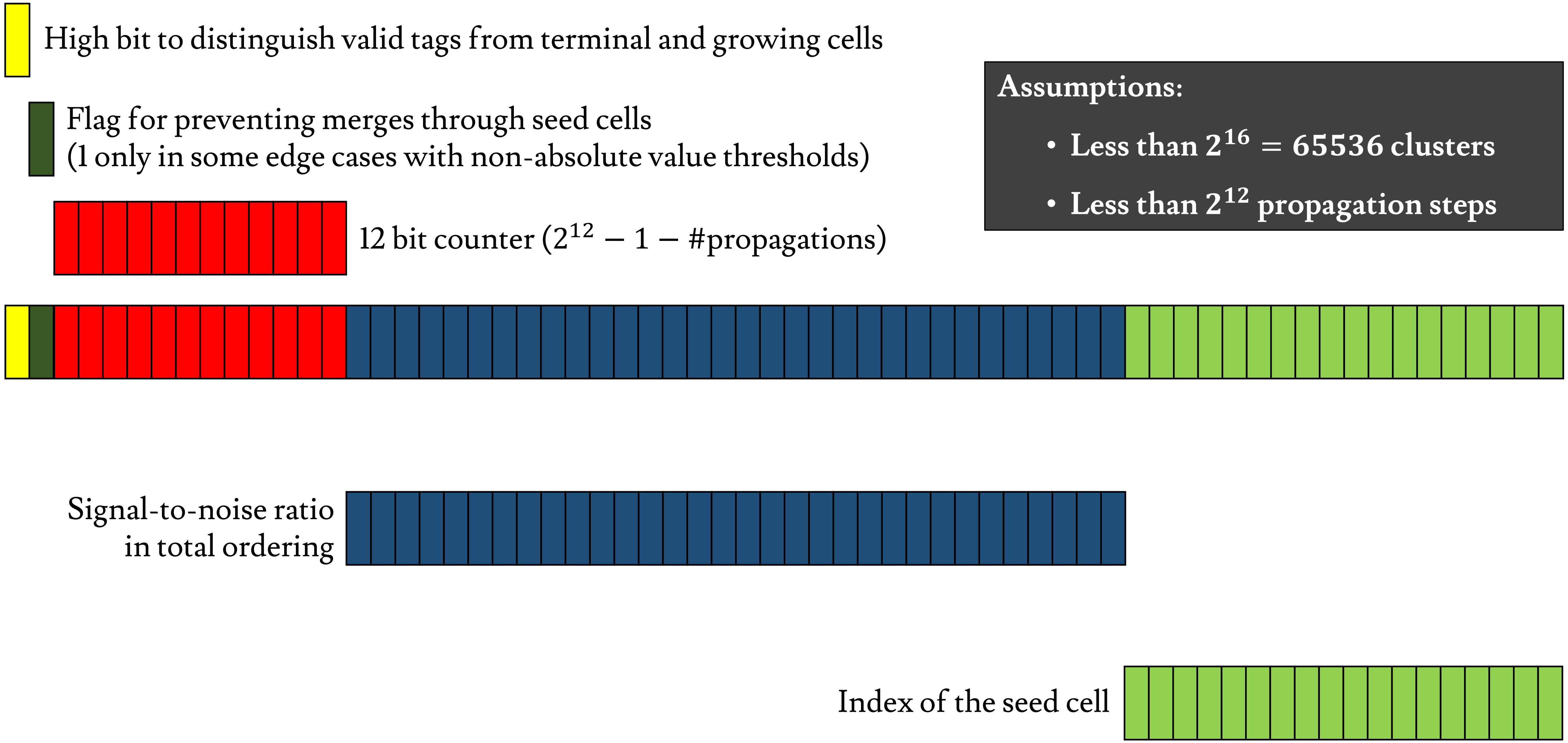
- Samples correspond to two kinds of **Monte-Carlo simulated** events:
  - $t\bar{t}$ **events**: 3000 events, $\mu = 80$
  - **di-jet events**: 10000 events, $\mu = 200$
- Results were obtained on a remote server provided by the Brookhaven National Laboratory:
  GPU is a Tesla P100, CPU is a Xeon E5-2695 v4
- Time measurements were based on a **per-thread clock**
  - For a single thread, "any clock" would work
    - The CPU – GPU comparison is a bit lopsided, though...
  - For more threads, **timing and speed-up are representative, but throughput is a best-case estimate**
    - Essentially, we are assuming everything is always running in parallel
  - This is due to **several limitations** when trying to **benchmark within the ATLAS software**

Breakdown of GPU Execution Times



Step		t-$\bar{t}$ Events			Jet Events		
		Time (μs)	Fraction of Total Time		Time (μs)	Fraction of Total Time	
Pre-Clustering Conversion		1441 ± 225	$9.24 \pm 1.58\%$		1128 ± 88	$13.24 \pm 1.14\%$	
Pre-Clustering Transfer		266 ± 8	$1.71 \pm 0.18\%$		248 ± 15	$2.92 \pm 0.30\%$	
Growing	Cell Classification	61 ± 2	$15.76 \pm 1.77\%$	$2.53 \pm 0.35\%$	56 ± 3	$20.38 \pm 1.30\%$	$3.24 \pm 0.25\%$
	Neighbour Pair Creation	159 ± 8	$40.68 \pm 3.23\%$		114 ± 6	$41.45 \pm 1.98\%$	
	Tag Propagation	175 ± 46	$43.55 \pm 4.87\%$		106 ± 13	$38.17 \pm 2.59\%$	
	Total	396 ± 53	—		276 ± 16	—	
Post-Growing Property Calculation		74 ± 22	$0.47 \pm 0.14\%$		55 ± 2	$0.65 \pm 0.05\%$	
Splitting	Neighbour Pair Creation	409 ± 28	$29.23 \pm 2.73\%$	$9.02 \pm 1.16\%$	287 ± 14	$33.80 \pm 1.62\%$	$9.97 \pm 0.57\%$
	Local Maxima Identification	88 ± 7	$6.25 \pm 0.53\%$		57 ± 3	$6.77 \pm 0.33\%$	
	Secondary Maxima Exclusion	229 ± 16	$16.48 \pm 2.25\%$		230 ± 14	$27.15 \pm 2.11\%$	
	Main Tag Propagation	642 ± 194	$44.44 \pm 5.46\%$		236 ± 49	$27.53 \pm 3.54\%$	
	Finalization	50 ± 5	$3.60 \pm 0.31\%$		40 ± 3	$4.75 \pm 0.27\%$	
	Total	1417 ± 226	—		851 ± 67	—	
Cluster Moments		1422 ± 134	$9.07 \pm 0.58\%$		889 ± 52	$10.43 \pm 0.51\%$	
Importing	Cluster Number Transfer	21 ± 1	$0.20 \pm 0.03\%$	$67.77 \pm 2.59\%$	17 ± 1	$0.33 \pm 0.04\%$	$59.27 \pm 2.26\%$
	Cluster Info Transfer	250 ± 39	$2.35 \pm 0.27\%$		102 ± 20	$2.00 \pm 0.21\%$	
	Cluster Creation + Cell Info Transfer	395 ± 83	$3.68 \pm 0.46\%$		147 ± 24	$2.89 \pm 0.20\%$	
	Cell Cycle	2892 ± 444	$27.10 \pm 2.42\%$		1624 ± 117	$32.18 \pm 2.45\%$	
	Cluster Ordering	192 ± 41	$1.79 \pm 0.28\%$		77 ± 15	$1.52 \pm 0.16\%$	
	Cluster Filling	2022 ± 336	$18.88 \pm 1.41\%$		944 ± 165	$18.46 \pm 1.18\%$	
	Moments Transfer	3.7 ± 0.5	$0.04 \pm 0.01\%$		3.8 ± 1.0	$0.07 \pm 0.02\%$	
	Moments Filling	4903 ± 697	$45.96 \pm 3.43\%$		2178 ± 389	$42.54 \pm 2.36\%$	
	Total	10679 ± 1377	—		5094 ± 690	—	
Total		15724 ± 1630	—		8565 ± 847	—	

Topo-Automaton Cluster Growing – Anatomy of a Tag



High bit to distinguish valid tags from terminal and growing cells

Flag for preventing merges through seed cells
(1 only in some edge cases with non-absolute value thresholds)

12 bit counter ($2^{12} - 1 - \#\text{propagations}$)

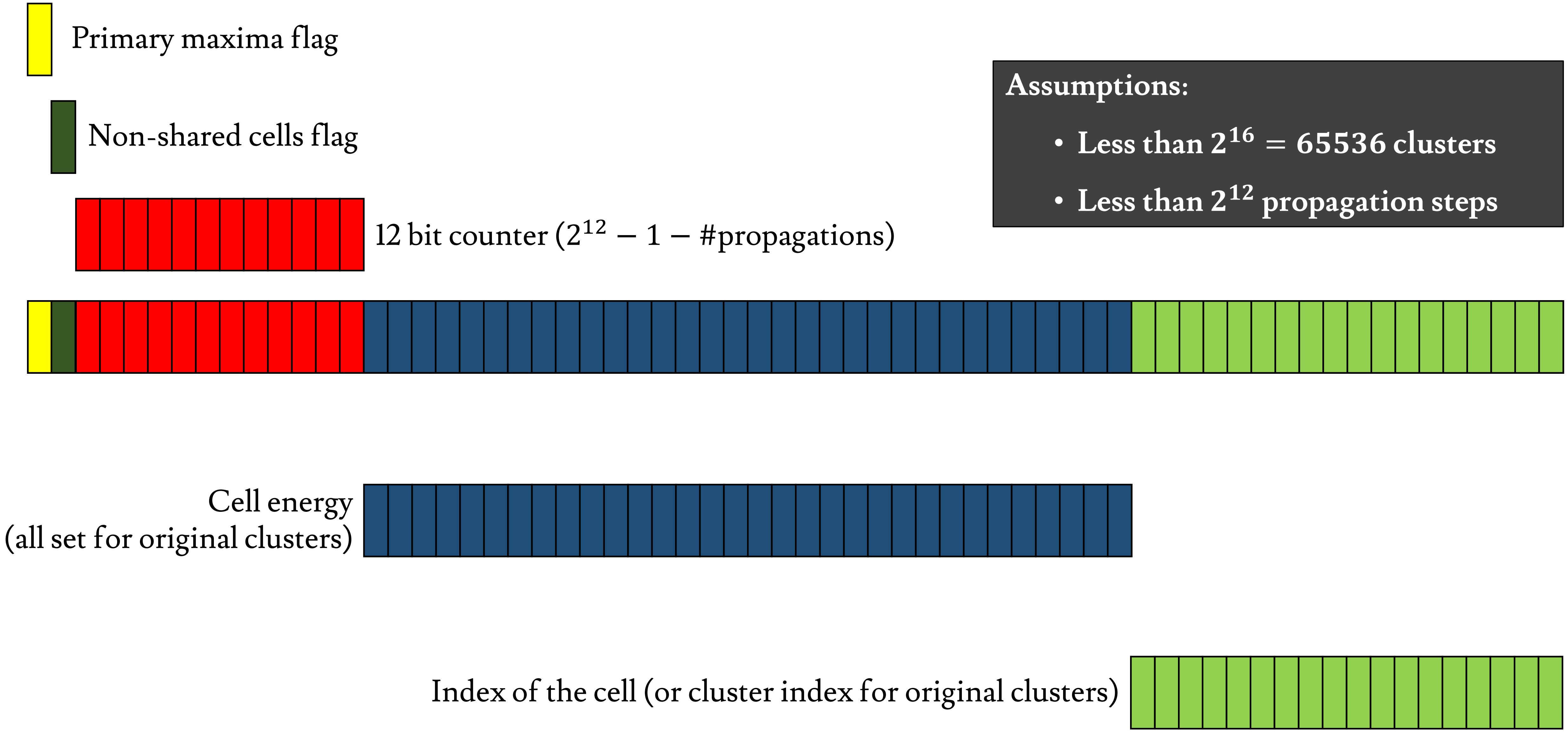
Signal-to-noise ratio
in total ordering

Index of the seed cell

Assumptions:

- Less than $2^{16} = 65536$ clusters
- Less than 2^{12} propagation steps

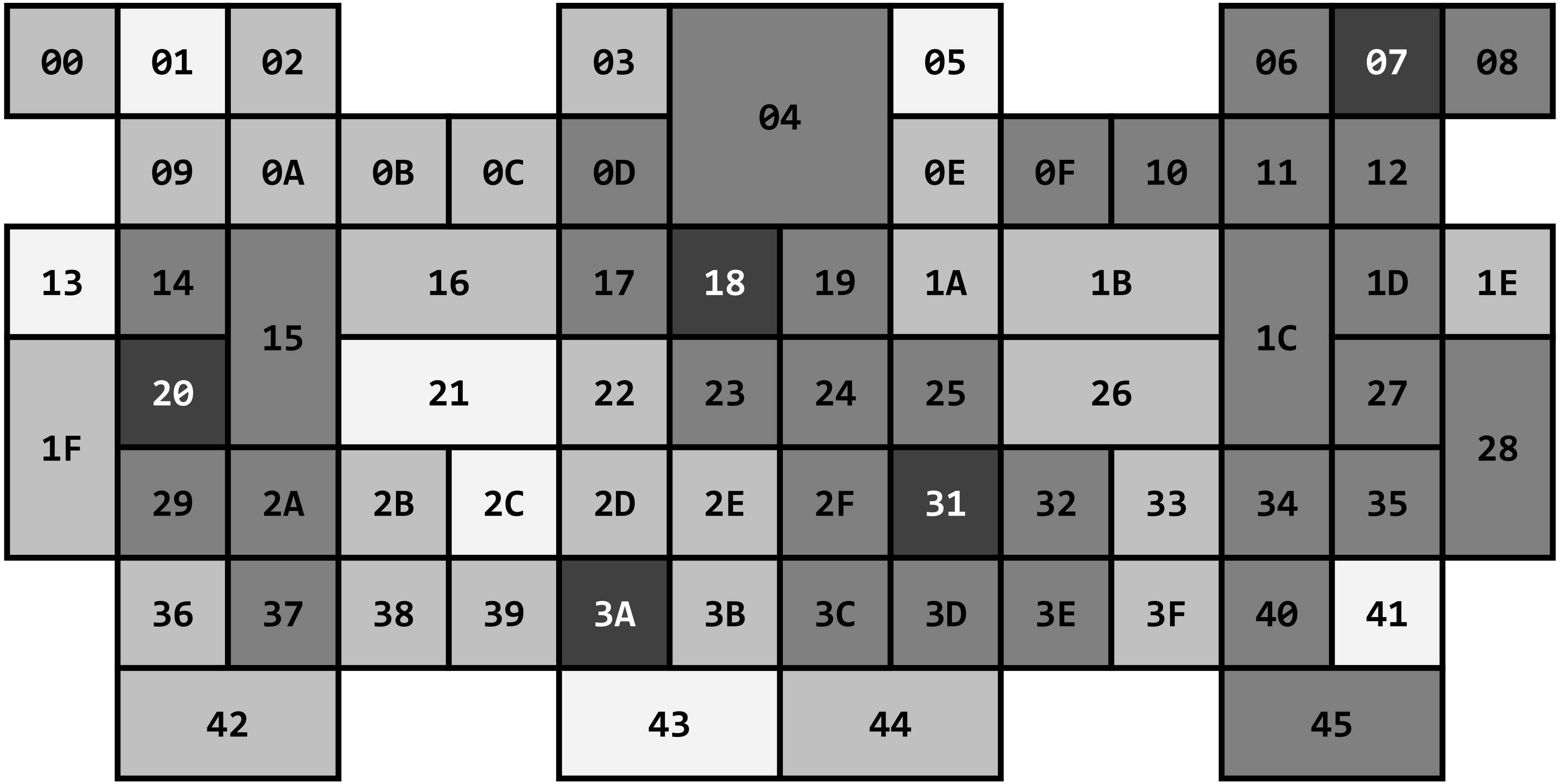
Topo-Automaton Cluster Splitting – Anatomy of a Tag


Primary maxima flag
Non-shared cells flag
12 bit counter ($2^{12} - 1 - \#\text{propagations}$)
Cell energy
(all set for original clusters)
Index of the cell (or cluster index for original clusters)

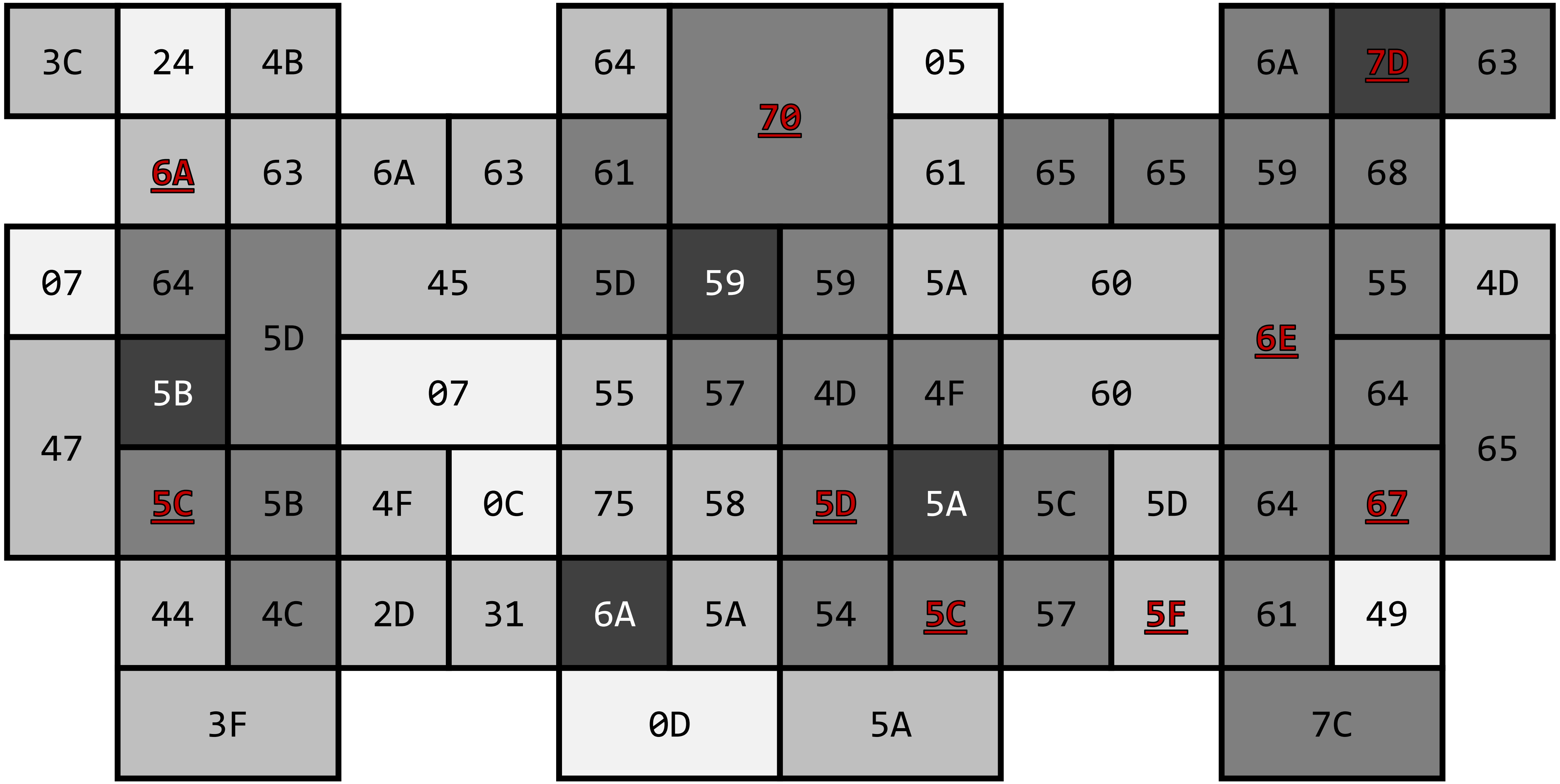
Assumptions:

- Less than $2^{16} = 65536$ clusters
- Less than 2^{12} propagation steps

Example Cell Indices

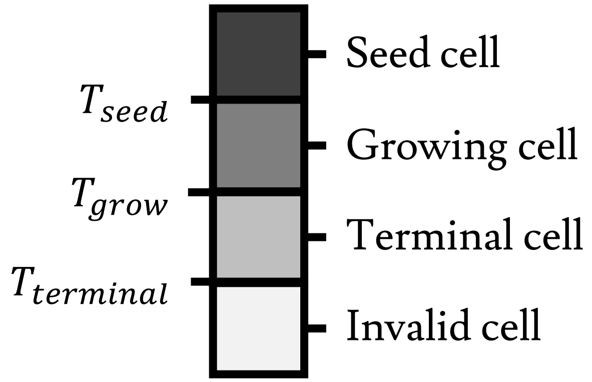


Example Cell Energies Bit Patterns


3C 24 4B 64 70 05 6A 7D 63
6A 63 6A 63 61 61 65 65 59 68
07 64 5D 45 5D 59 59 5A 60 6E 55 4D
47 5B 07 55 57 4D 4F 60 64 65
5C 5B 4F 0C 75 58 5D 5A 5C 5D 64 67
44 4C 2D 31 6A 5A 54 5C 57 5F 61 49
3F 0D 5A 7C

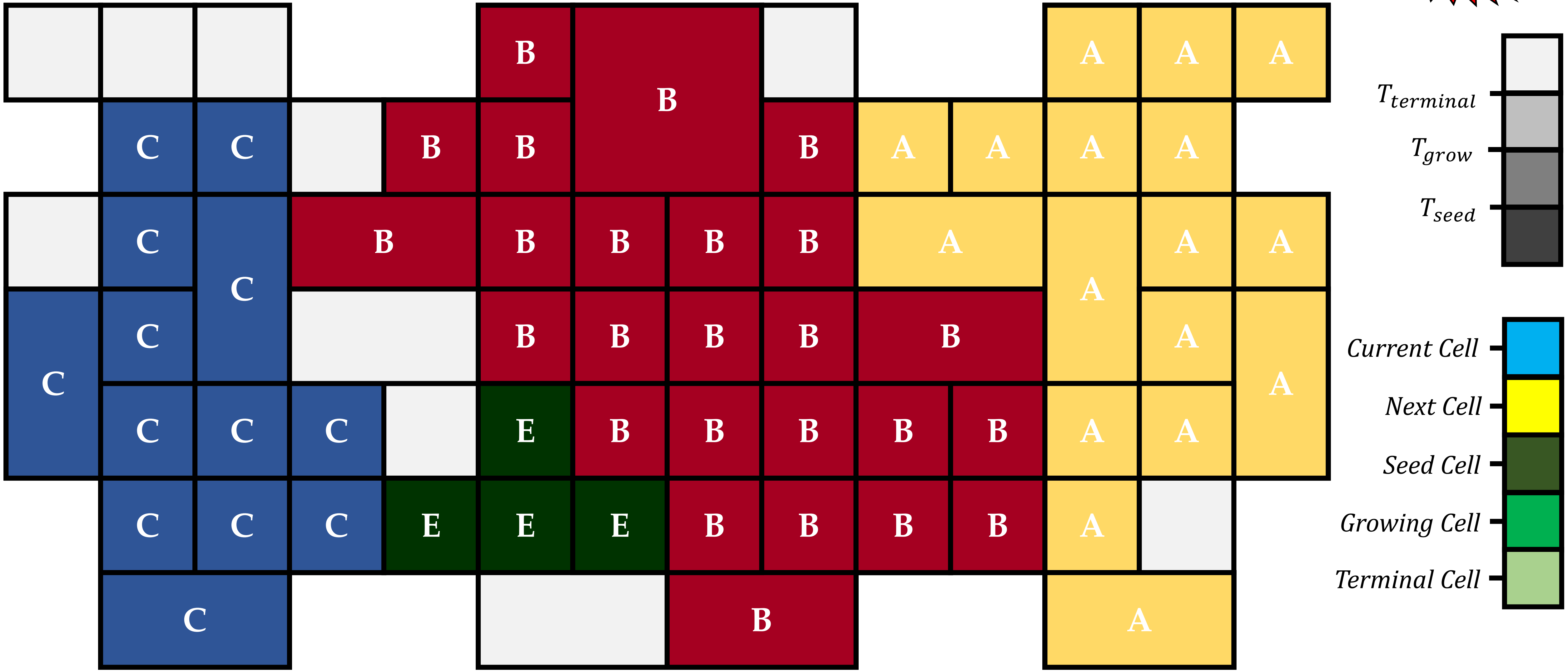
Topological Cluster Growing – Description

- Cluster growing starts by **classifying the cells** using the **signal-to-noise ratio** of the energy deposition, according to **three thresholds**:



- Seed cells are the **origin of the clusters** we construct, so we **sort them by the signal-to-noise ratio** and add them to the *list of current cells*, and we now **iterate until that list is empty**:
  - For every cell in the *list of current cells*, we will **check each of its direct neighbours**:
    - If the neighbour does not belong to a cluster and its signal-to-noise ratio is above $T_{terminal}$,
      - **Add** it to the currently considered cell's **cluster**
      - If its signal-to-noise ratio is above T_{grow}, **add** it to a ***list of next cells***
    - Else, if it already belongs to a cluster and its signal-to-noise ratio is above T_{grow}, merge the two clusters: **all cells of the smallest cluster get added to the largest cluster**
  - Once all cells have been checked, the ***list of next cells*** becomes the new ***list of current cells***

Topological Cluster Growing – Example



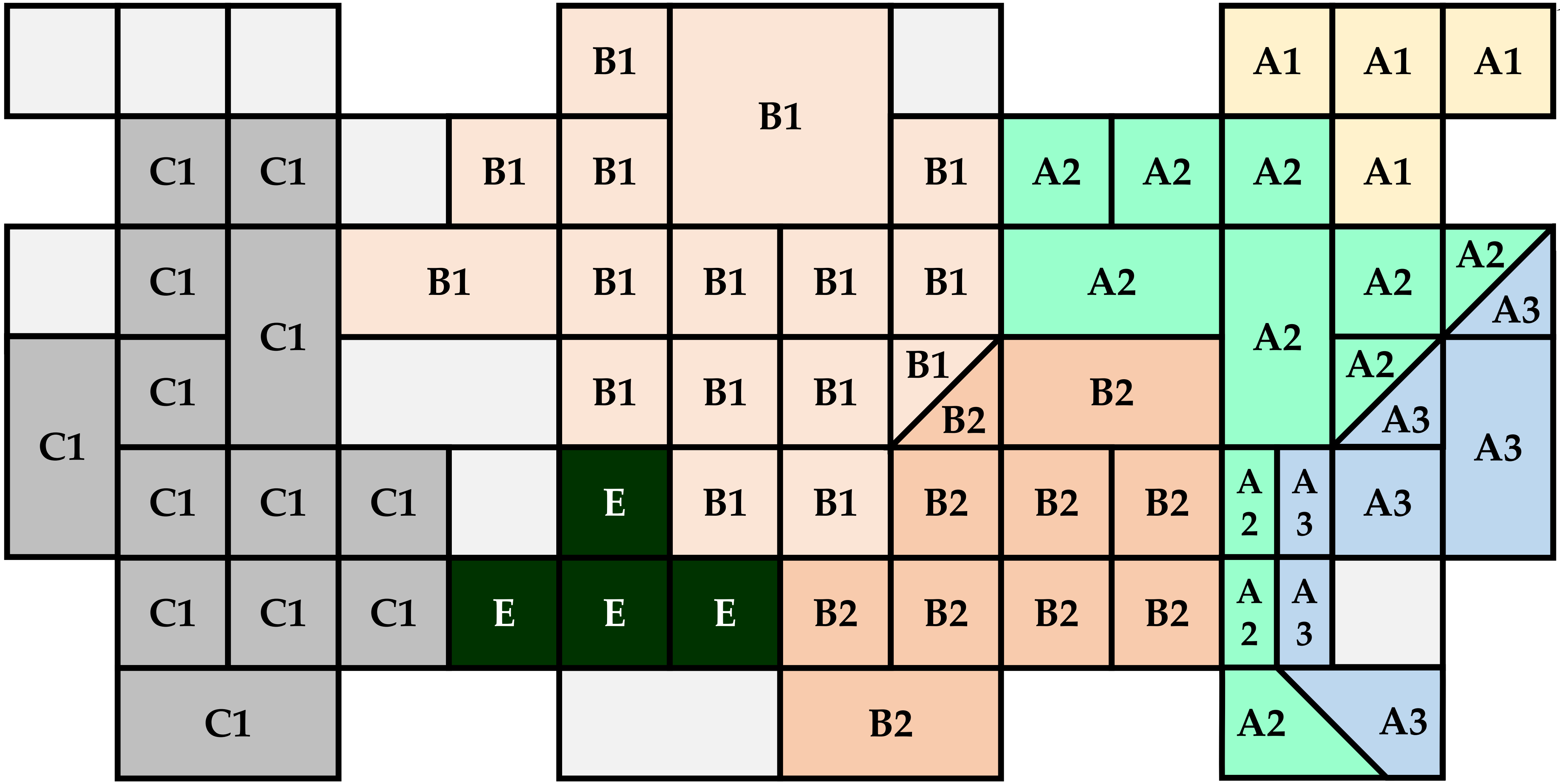
Topological Cluster Splitting – Description I

- Starting from the post-growing clusters, we **find the local maxima** of the deposited energy checking **only within the post-growing cluster** (and also taking into account some **thresholds in energy and number of neighbours**)
- Some calorimeter sampling layers give rise to **secondary local maxima**; secondary maxima that **overlap** with others through (`next`/`prev`)(`InSamp`/`InCalo`) neighbours are **excluded**
- Create a cluster for every local maximum, add the cells to the *list of current cells*, and **iterate until the list is empty**:
  - **Sort** the *list of current cells* by their **energy**
  - For every cell in the list, **check each of its direct neighbours** that is **part of the same post-growing cluster**:
    - If it does not belong to a cluster, **add** it to the current cell's **cluster** and to a ***list of next cells***
    - Else, if it already belongs to a cluster, check if it is part of the *list of next cells*; if it is, **remove** it from that list, add it to a ***list of next shared cells*** and add it to the current cell's **cluster** as well
  - The ***list of next cells*** becomes the new ***list of current cells***
  - **Sort** the *list of next shared cells* by **energy** and add it to the ***list of shared cells***
- Add all elements of the ***list of shared cells*** to the (now empty) ***list of current cells***

Topological Cluster Splitting – Description II

- **Iterate** again **until the *list of current cells* is empty**:
  - For every cell in the list , **check each of its direct neighbours** that is **part of the same post-growing cluster**:
    - If it does not yet belong to a cluster, **add** it to both of the current cell's **clusters**, add it to the ***list of shared cells*** and add it to a ***list of next cells***
  - The ***list of next cells*** becomes the new ***list of current cells***
  - **Sort** the *list of current cells* by **energy**
- For every cell in the *list of shared cells*, **calculate the weight** of its contribution to each cluster: $w_1 = \frac{E_1}{E_1 + r\ E_2},\ w_2 = 1 - w_1$, with $r = e^{d_1 - d_2}$ and d_i being the **distance** from the cell **to the centroid** of cluster i in units of typical shower scale (~ 5 cm)
- The resulting clusters are the **final clusters**
  - Post-growing clusters that had **no local maxima** are also part of the final clusters, **unchanged** by cluster splitting

Topological Cluster Splitting – Example



Cluster Moments Calculation

- While the calculation of cluster moments is **not the focus of the current discussion**, it is useful to provide a few remarks, especially to better **understand some of the comparisons** we will show later
- Many moments are **weighted averages of cell properties** (typically with $n \in \{1, 2\}$):

$$\langle \Xi^n \rangle = \frac{\sum_{c \in \text{cluster}:E_c > 0} w_c E_c (\Xi_c)^n}{\sum_{c \in \text{cluster}:E_c > 0} w_c E_c}$$

- Some moments depend on the **shower axis** $\vec{S}$ (the **direction of flight** of the particle responsible for the shower):
  - Find the **centroid of the cluster** taking into account **only cells with positive energy**, $\vec{C} = (C_x, C_y, C_z)$
  - Define a matrix **M** such that its components are:

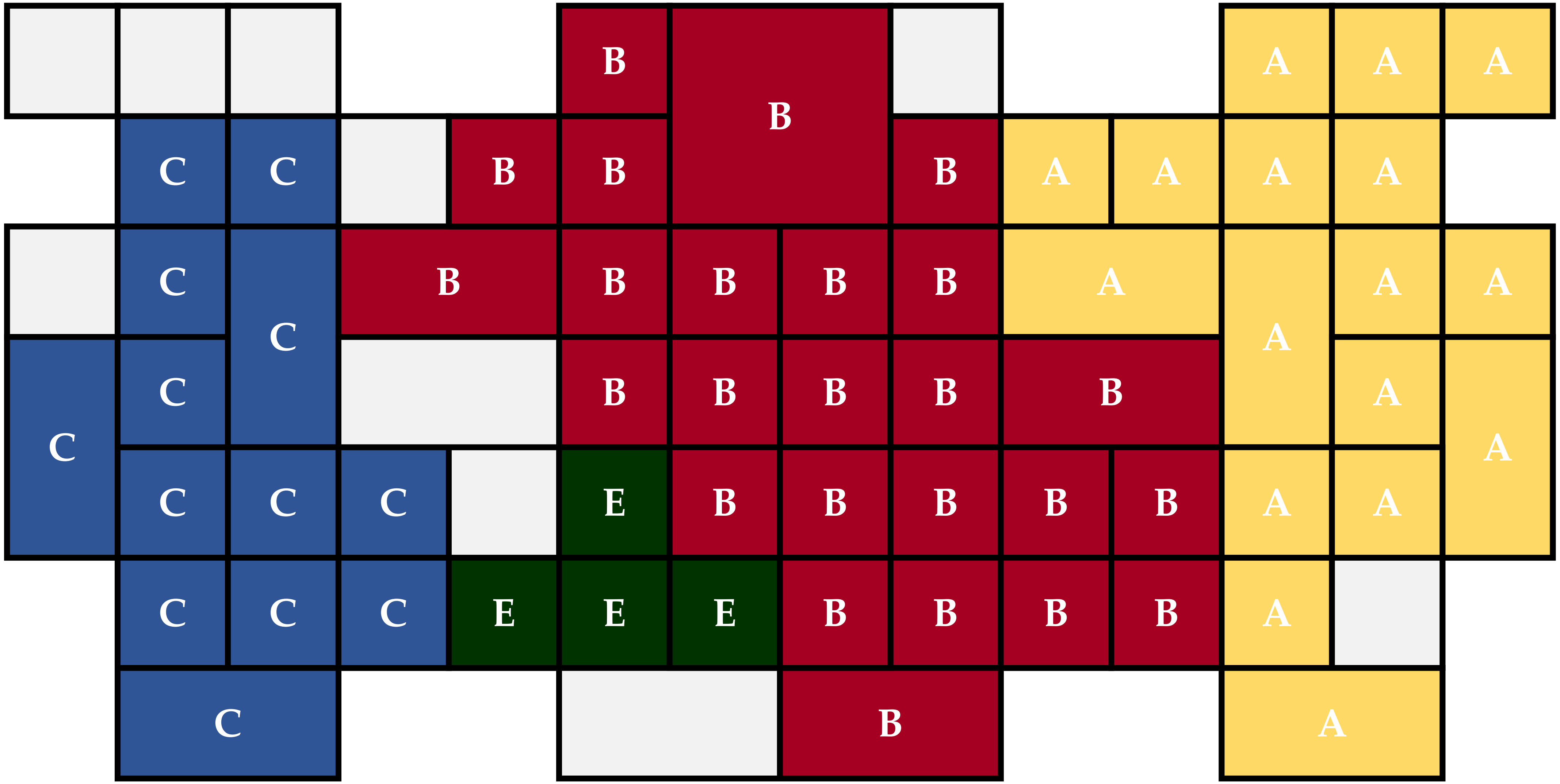
$$\mathbf{M}_{i,j} = \frac{\sum_{c \in \text{cluster}:E_c > 0} w_c^2 E_c^2 (x_i(c) - C_i)(x_j(c) - C_j)}{\sum_{c \in \text{cluster}:E_c > 0} w_c^2 \, E_c^2}$$

  - The shower axis is the **normalized eigenvector** of **M** that has the **smallest angle** to the direction of $\vec{C}$, as long as that angle is smaller than a threshold (typically 20º) and the cluster has **at least 3 cells with positive energy**
  - Otherwise, the shower axis is taken to be the direction of $\vec{C}$: $\vec{S} = \vec{C} / \|\vec{C}\|$
- Some **other moments** (e. g. isolation, significance, second time) **have different definitions**

Topo-Automaton Cluster Growing – Description

- **Classify each cell** according to the signal-to-noise ratio, as in standard topological clustering:
  - **Invalid**, **terminal** and **growing cells** get assigned values that do not correspond to valid tags (0, 1 and $2^{63} - 1$, respectively, though the choice is arbitrary as long as they are **ordered** and **compare lower than any valid tags**)
  - **Seed cells** are assigned a tag constructed as previously shown, with a counter for the cluster index being used (and incremented afterwards) to **ensure the indices are sequential**, and the corresponding entries of the *seed cell index to cluster map* and the *cluster index to seed cell map* are **updated** with the appropriate values
- Create two lists of **pairs of neighbouring cells**: the ***list of growing pairs*** for pairs of cells where both are **growing or seed**, and the ***list of terminal pairs*** where one is **growing or seed** and the other is **terminal**
  - Given that the **neighbourhood relations are not necessarily symmetric**, this is a list of **ordered pairs**: we will only consider **propagation in a particular direction** (by convention, from the second to the first cell of the pair)
- **Keep iterating** over all pairs of cells in the *list of growing pairs* **until there are no tag changes**:
  - If the second element of the pair is **not part of a cluster**, **ignore** this pair
  - **Decrement the counter** of the second element's tag and **unset the cell merging flag** to get the **propagated tag**
  - If the first element **does not have a valid cluster tag**, it **gets the propagated tag** (e. g. via an atomic maximum)
  - Else, if it is **part of a different cluste**r, assign **the highest of the two cluster indices** to the entries of the *seed cell index to cluster map* and take the **maximum** between the entries of the *cluster index to seed cell map*
- For every pair of cells in the ***list of terminal pairs***, **propagate the tag** from the second element to the first

Topo-Automaton Cluster Growing – Example



Topo-Automaton Cluster Splitting – Description I

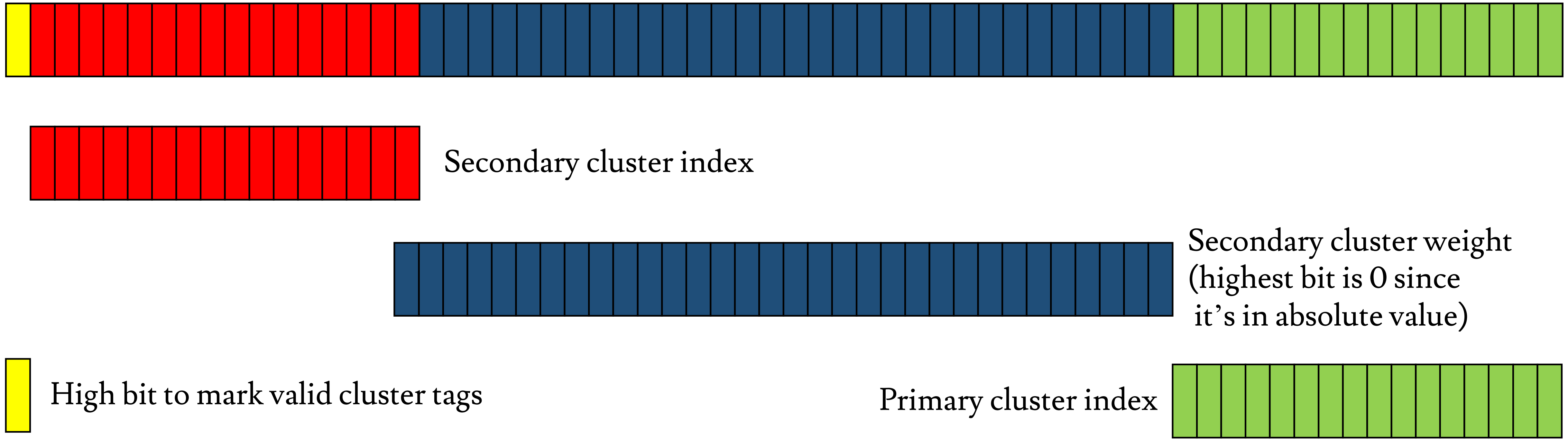
- **Create four lists of ordered pairs of neighbours** (similar to cluster growing) that will be used within the algorithm:
  - **Pairs** within the **same post-growing cluster** that will be **used to expand the split clusters**
  - **Additional pairs** used for **checking the local maxima** (due to certain neighbour options being ignored)
  - Pairs of `next(InSamp/InCalo)` neighbours **regardless of the original post-growing cluster**
  - Pairs of `prev(InSamp/InCalo)` neighbours **regardless of the original post-growing cluster**
- Use the first two lists of pairs to **check for local maxima**: exclude every cell that has a **neighbour with equal or greater energy**, check the remaining ones for the **thresholds in energy and number of neighbours**
- **Local maxima** are assigned a tag calculated as shown before (again using a counter and incrementing afterwards to get an index for every cluster) and the original cluster is stored in the *cluster index to original cluster map*; other cells belonging to clusters are assigned a tag that has the **cluster index in the 18 lowest bits**, and **the first bit of the energy** set if the **post-growing cluster has local maxima**, or **all other bits and flags set** if **it does not**
- Consider **two separate sets of tags**, one for `next` neighbours, the other for `prev`; every primary local maximum will be assigned the value of $2^{64} - 1$, the secondary local maxima get their regular tags, all other cells are initialized to 0
- **Keep iterating** through the pairs of `next` and `prev` neighbours **until there are no tag changes**: if the second element of the pair has a **non-zero tag**, **propagate it to the first**
- Any **secondary maxima** that got their **tag replaced** in either set of tags should be **excluded** and their tags set to one that identifies a **cell belonging to the original cluster**, for all others **set the primary flag** (as there are **no further distinctions** between primary and secondary maxima in later stages of the algorithm)

Topo-Automaton Cluster Splitting – Description II

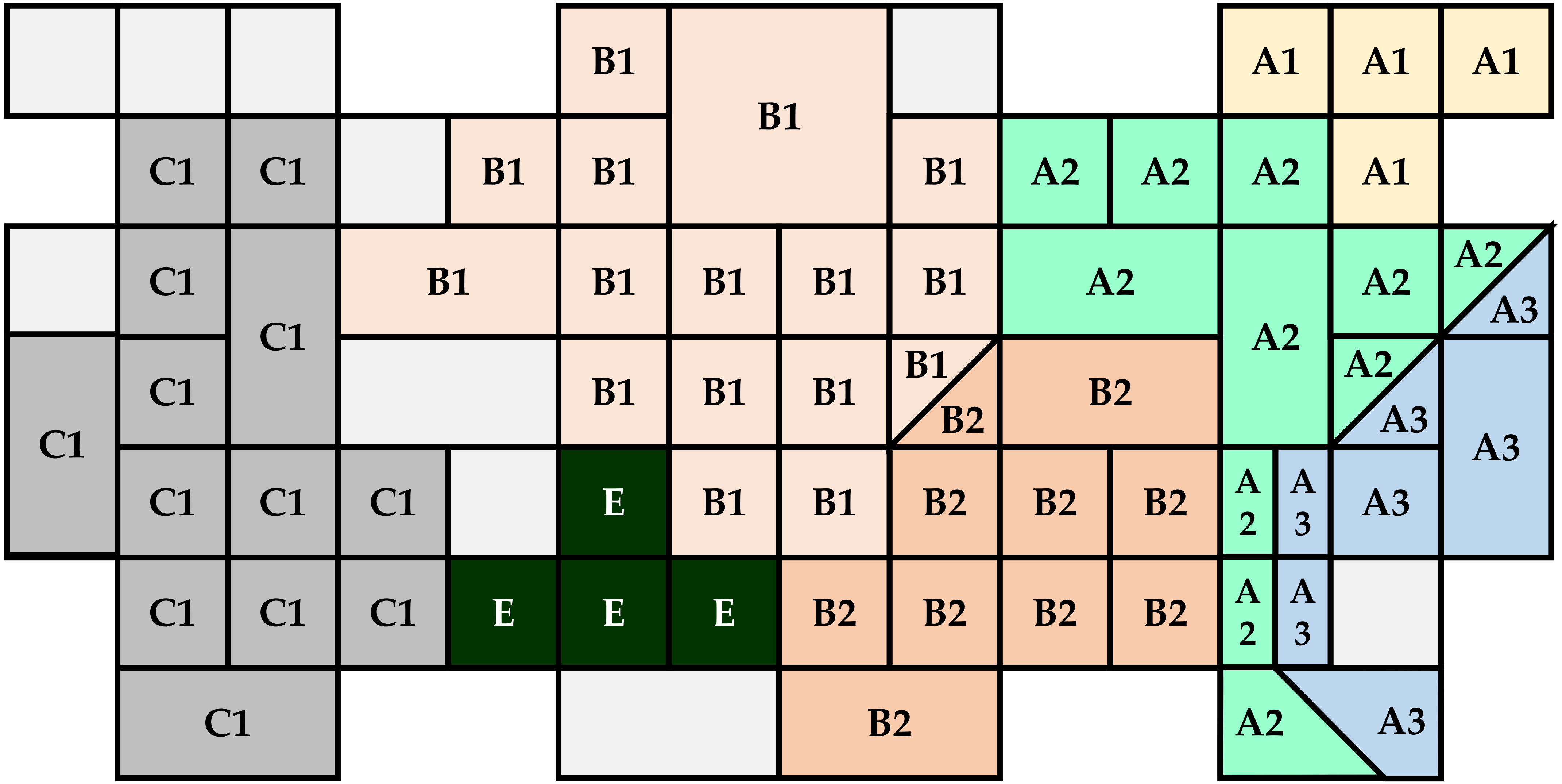
- **Keep iterating until there are no tag changes**:
  - Set a **reset counter** to 0
  - Iterate through the **first list of pairs** of neighbouring cells:
    - If the second element of the pair **is not part of a cluster** created from a local maximum, **ignore this pair**
    - **Decrement the counter** of the second element's tag (or **set it to $2^{12} - 1$** if the second element is a **shared cell** and the counter is **higher than this value**) and **unset the primary flag** to get the **propagated tag**
    - If the first element **does not have a valid cluster tag**, it gets the **propagated tag** (e. g. via atomic maximum)
    - Else, if its **reverse propagation counter** is the **same as the propagated tag's** and it belongs to a **different cluster**, assign to it a tag that signals this is to become a **shared cell: first flag and all bits of the counter set**, **second flag unset**, and the energy and index the same as the propagated tag; also **set the reset counter** to the **maximum** between its **current value** and the **reverse propagation counter** of the original tag
  - **Iterate through all cells** that are **part of a cluster** created from a local maximum:
    - If the cell's **reverse propagation counter** is **lower than the reset counter**, assign to it the tag of **a cell that belongs to its original post-growing cluster** (and thus not to a local maximum)
    - **Update the *cell index to cluster map*** with the new cluster assignment (in the case of shared cells, the **highest cluster index** is in **the most significant bits** and the **lowest** in the **least significant bits**)
    - **Update the tag** with the **energy and index of this cell**

Topo-Automaton Cluster Splitting – Description III

- **Calculate the centroid** of the split clusters: the **absolute-energy-weighted sum** of the **non-shared cells' coordinates**
- **Assign** appropriate **weights** to the contribution of the **shared cells** to each of the clusters:
  - As before, the weight is calculated by: $w_1 = \frac{E_1}{E_1 + r\,E_2}$, $w_2 = 1 - w_1$, with $r = e^{d_1 - d_2}$ and d_i being the distance from the cell to the centroid of cluster i in units of typical shower scale ($\sim$ 5 cm)
  - The choice of indices is such that $w_1 \leq w_2$, to **minimize potential precision loss**
- We express the **final cluster assignment** with a **different tag format**:


Secondary cluster index
Secondary cluster weight (highest bit is 0 since it's in absolute value)
High bit to mark valid cluster tags
Primary cluster index

Topo-Automaton Cluster Splitting – Example


B1
B1
A1
A1
A1
C1
C1
B1
B1
B1
A2
A2
A2
A1
C1
B1
B1
B1
B1
B1
A2
A2
A2
A3
C1
C1
B1
B1
B1
B1
B2
B2
A2
A3
C1
A3
C1
C1
C1
E
B1
B1
B2
B2
B2
A 2
A 3
A3
C1
C1
C1
E
E
E
B2
B2
B2
B2
A 2
A 3
C1
B2
A2
A3