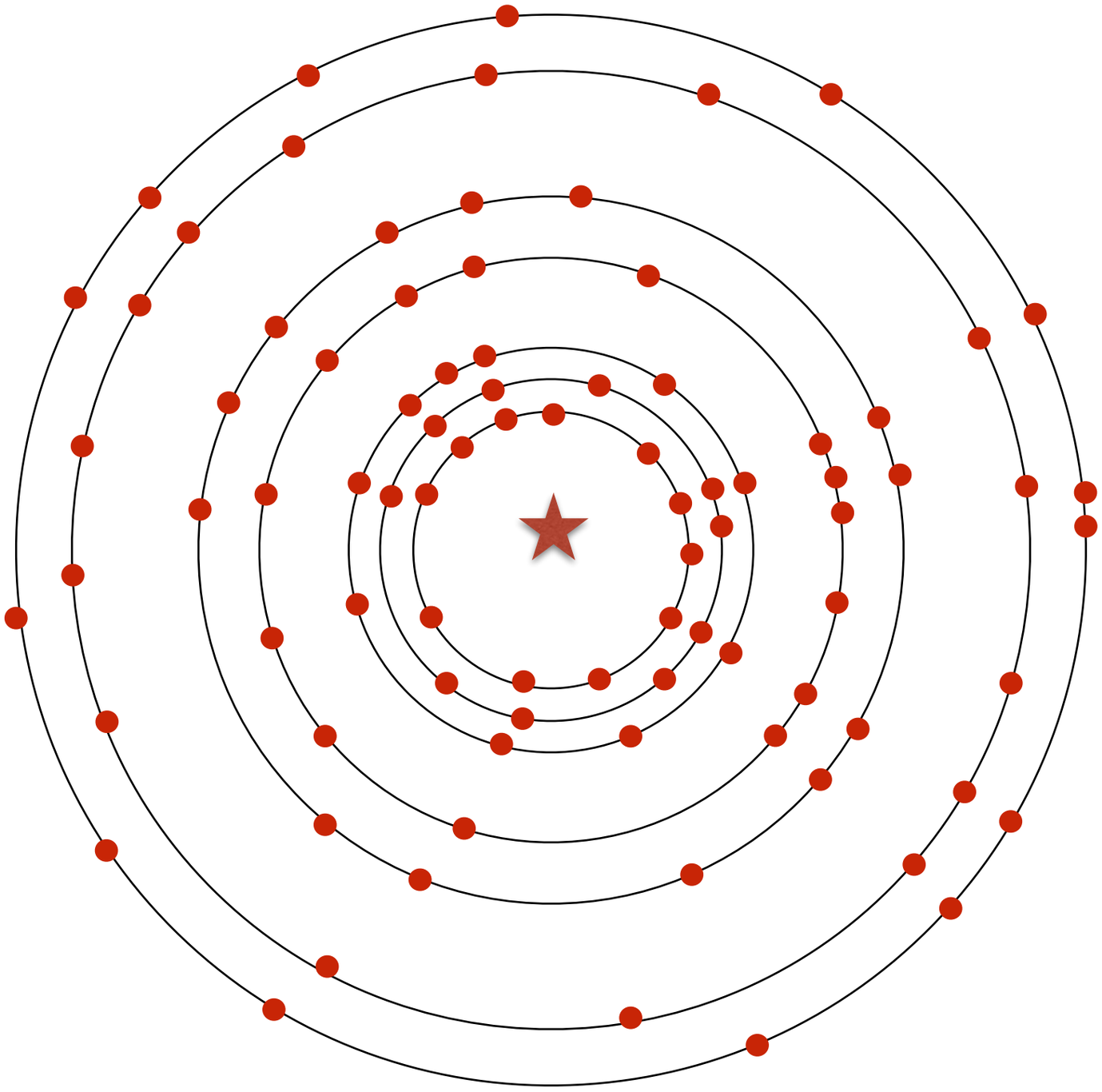
Status of ITS CA track reconstruction

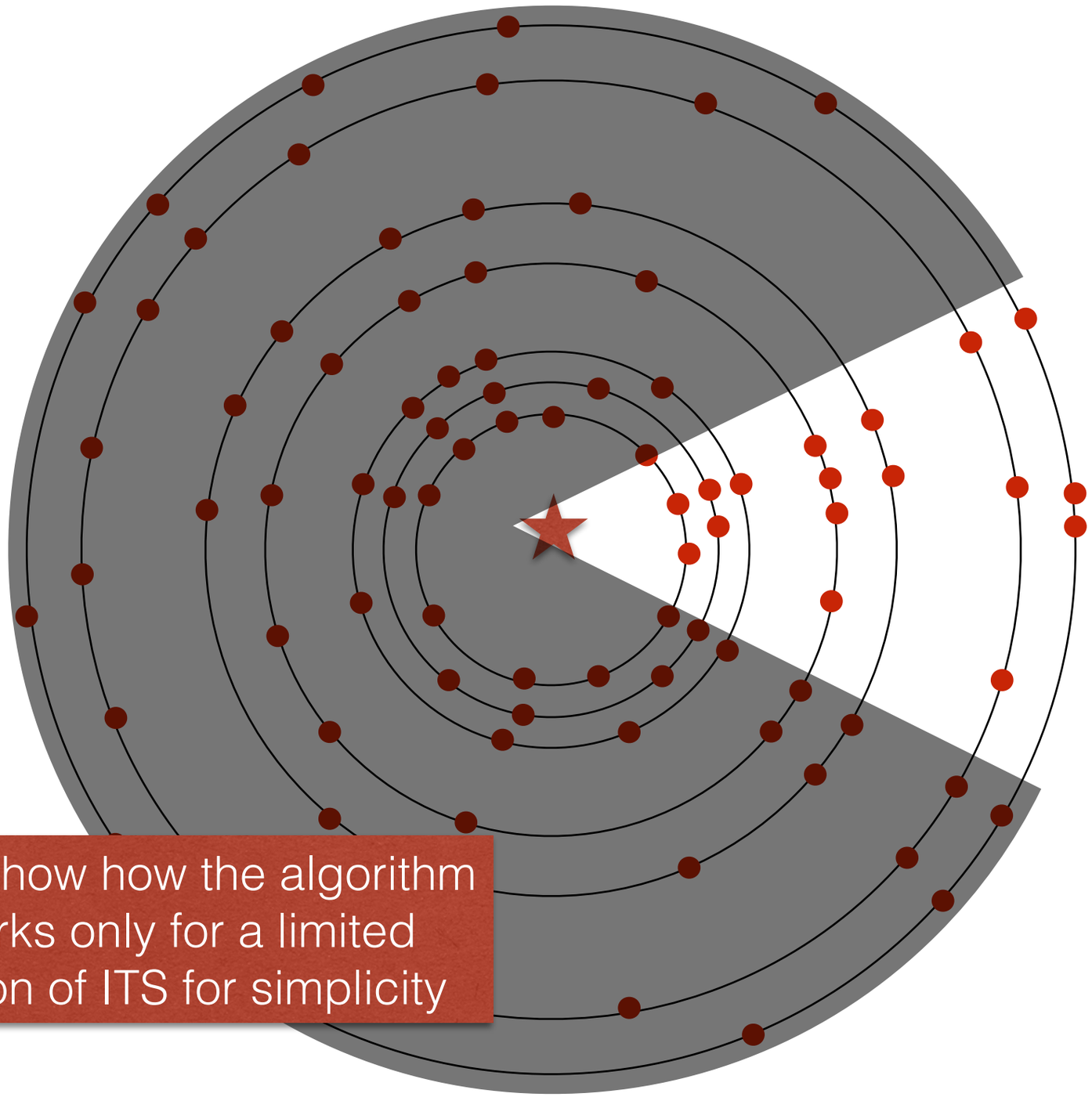
Iacopo Colonnelli[1,3], Matteo Concas[1,3], Maximiliano Puccio[2,3]
Politecnico, University and INFN Torino

More on this topic

CA reconstruction algorithm

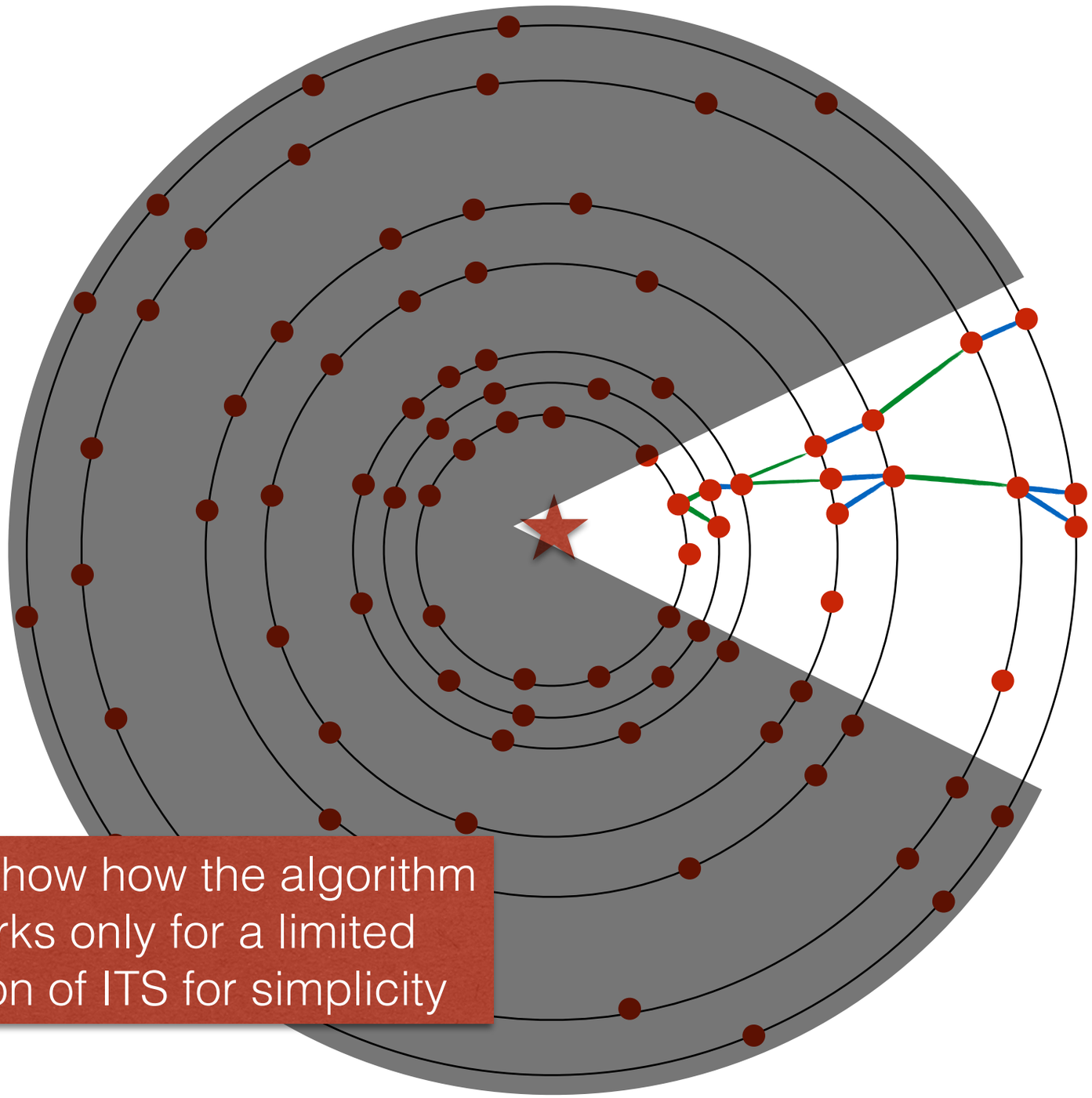


CA reconstruction algorithm



I will show how the algorithm works only for a limited region of ITS for simplicity

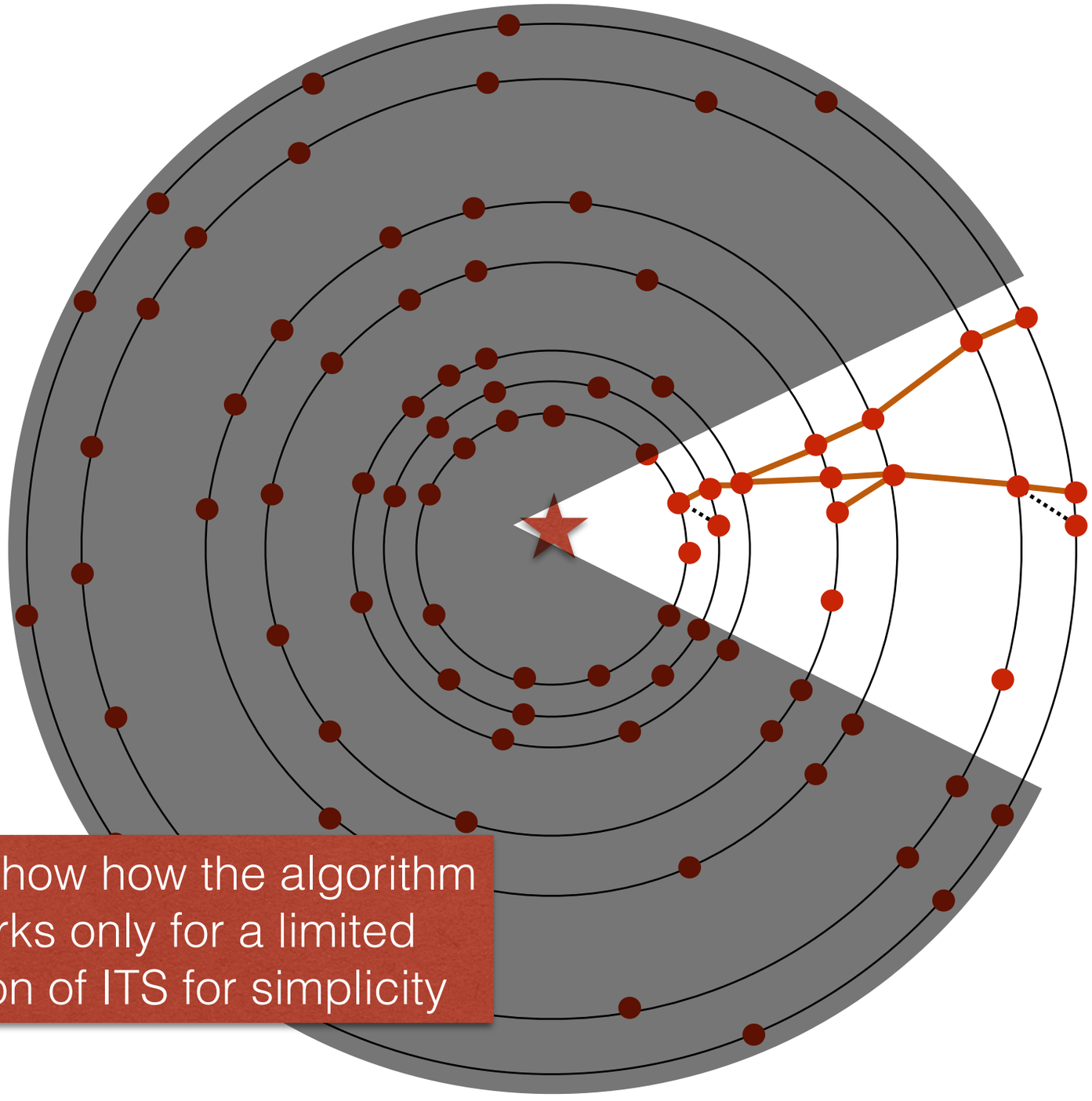
CA reconstruction algorithm



For each cluster on each layer a 2D window is opened. Then the clusters are joined with those on the next layer within the window

I will show how the algorithm works only for a limited region of ITS for simplicity

CA reconstruction algorithm

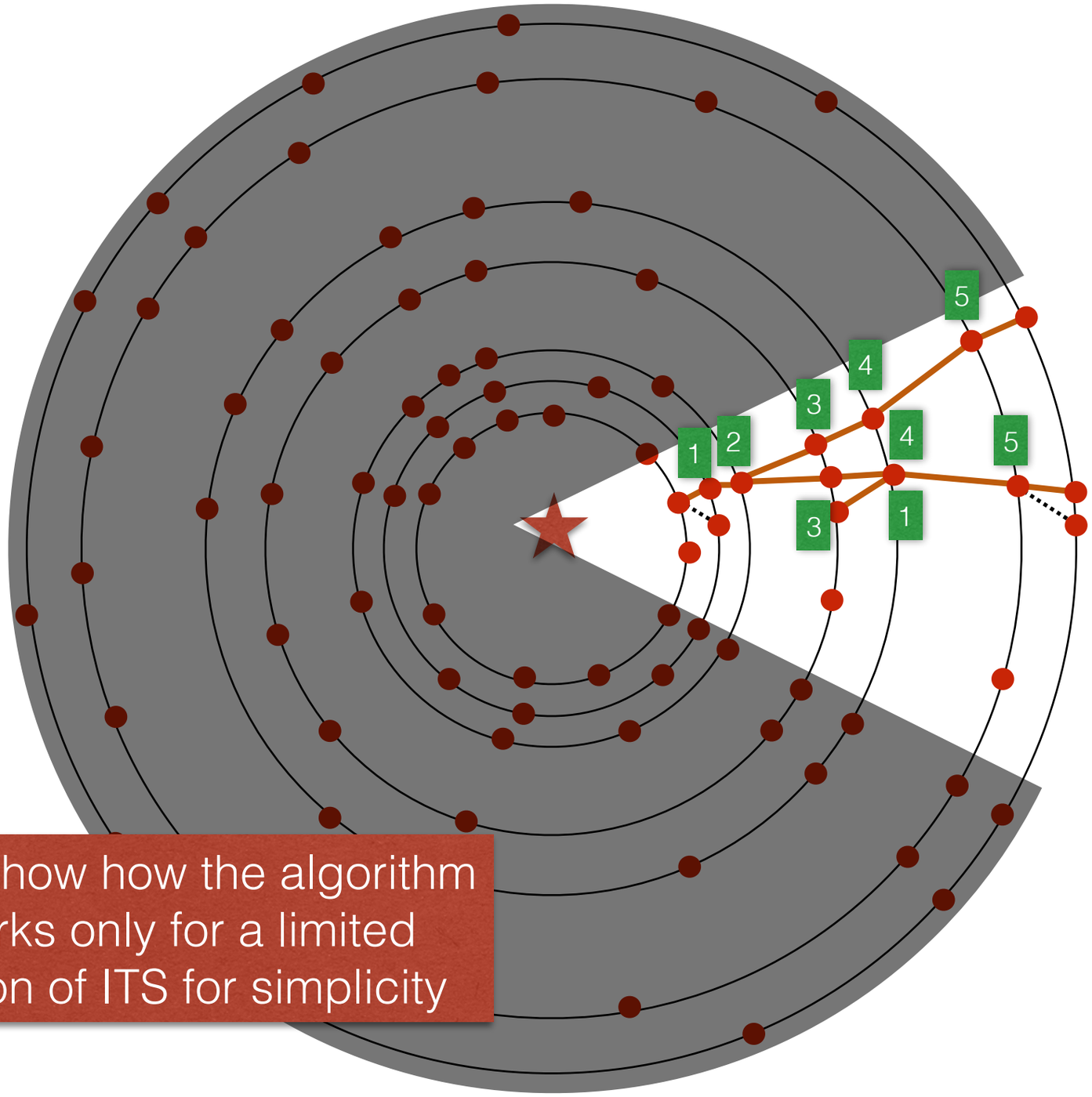


I will show how the algorithm works only for a limited region of ITS for simplicity

For each cluster on each layer a 2D window is opened. Then the clusters are joined with those on the next layer within the window

Subsequent doublets are combined in cells (3 points seed) and track params are computed

CA reconstruction algorithm



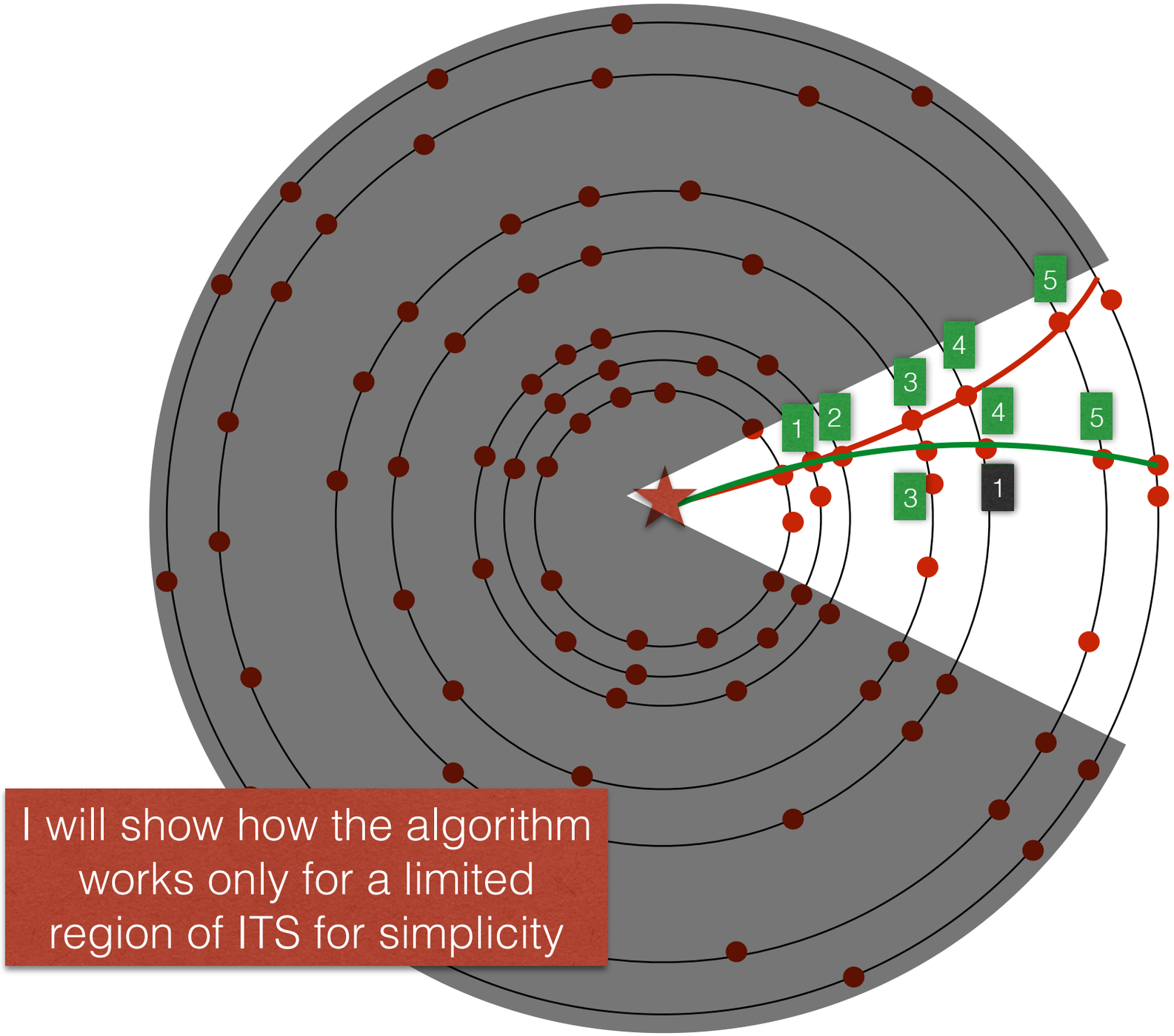
I will show how the algorithm works only for a limited region of ITS for simplicity

For each cluster on each layer a 2D window is opened. Then the clusters are joined with those on the next layer within the window

Subsequent doublets are combined in cells (3 points seed) and track params are computed

Each cell has an index representing the number of connected inner cells + 1

CA reconstruction algorithm



I will show how the algorithm works only for a limited region of ITS for simplicity

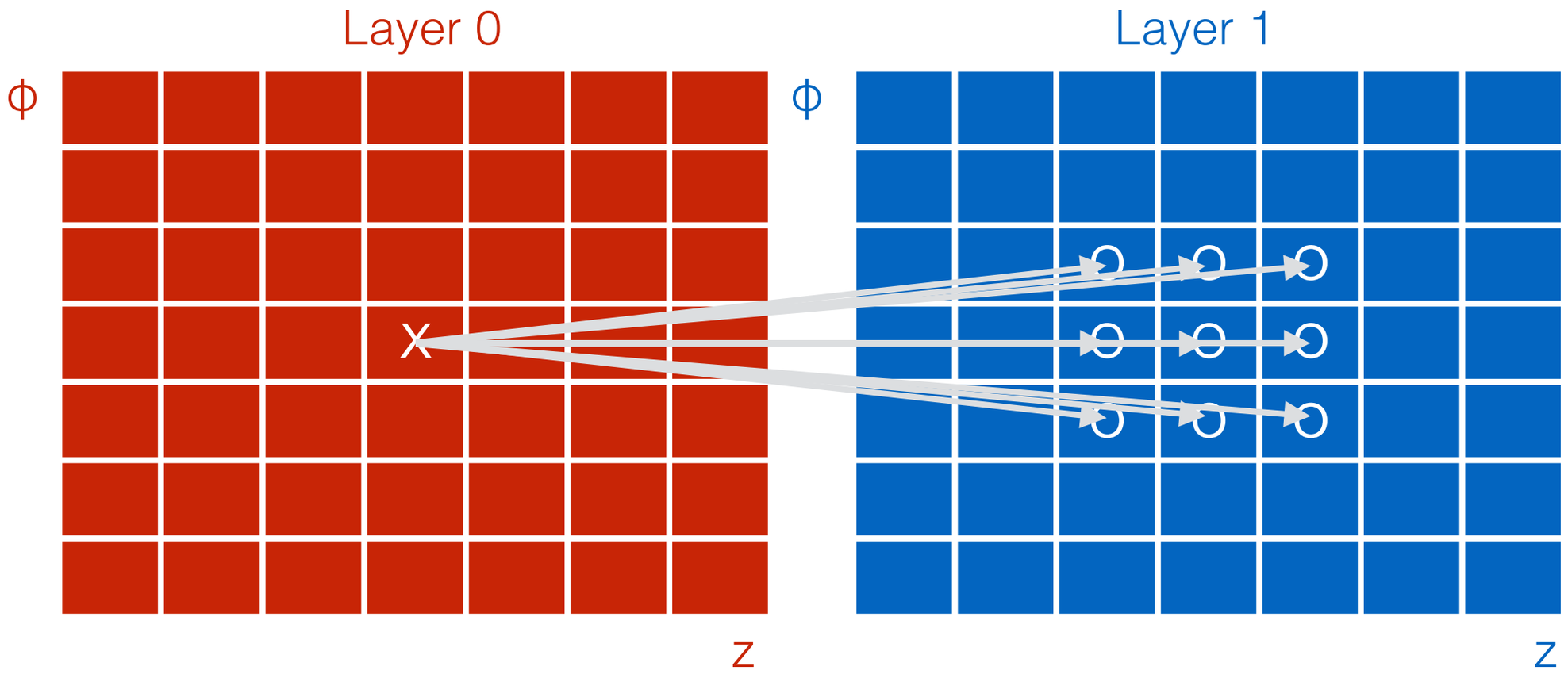
For each cluster on each layer a 2D window is opened. Then the clusters are joined with those on the next layer within the window

Subsequent doublets are combined in cells (3 points seed) and track params are computed

Each cell has an index representing the number of connected inner cells + 1

Longest, continuous sequences of indices represent candidates

Data layout



- Hits are sorted and stored according their azimuthal angle and their z coordinate
- An index table is filled to quickly fetch the hits in the region of interest of the detector
  - ✓ Increase data locality
  - ✓ Possible parallelisation

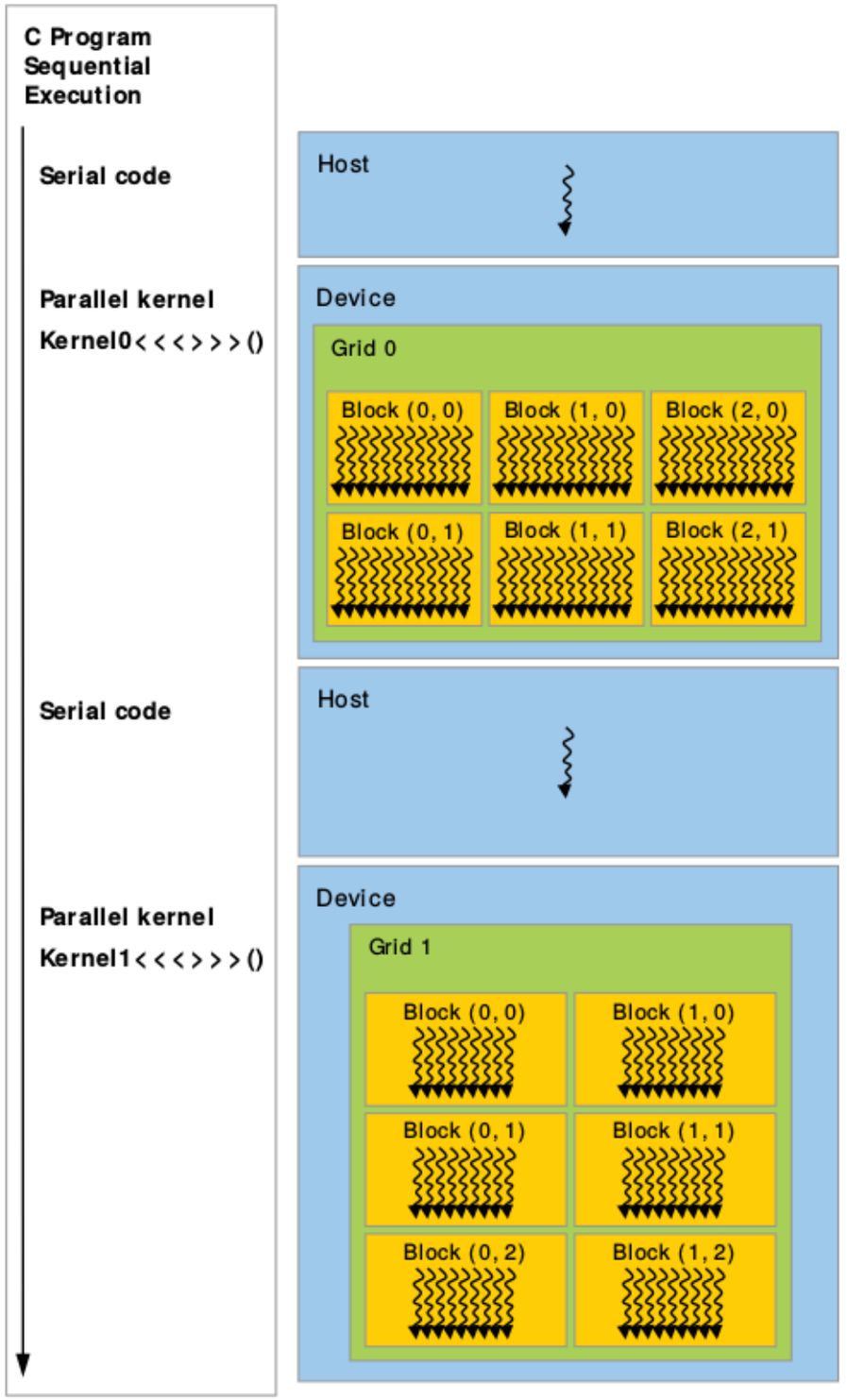
Computational hotspots

Incl.	Self	Called	Function
100.00	0.00	1	(below main)
100.00	0.00	(0)	0x0000000000000cc0
100.00	0.00	1	_dl_runtime_resolve_avx
100.00	0.00	1	_start
100.00	0.00	1	main
99.94	0.00	100	CATracker::clustersToTracks()
66.06	49.31	100	CATracker::computeTracklets(CAPrimaryVertexContext&)
25.77	25.63	100	CATracker::computeCells(CAPrimaryVertexContext&)
12.57	2.09	63 874 276	atan2f
10.48	5.02	63 874 276	__atan2f_finite
5.47	5.47	63 873 978	atanf
5.23	0.48	100	CAPrimaryVertexContext::CAPrimaryVertexContext(CAEvent const&, int)
4.28	3.10	6 987 953	CAIndexTable::selectBins(float, float, float, float)
2.58	0.80	8 008 004	CACluster::CACluster(int, std::array<float, 3ul> const&, CACluster const&)
2.18	0.18	9 847 129	operator new(unsigned long)
2.09	1.24	700	void std::__introsort_loop<__gnu_cxx::__normal_iterator<CACluster*, std::ve
2.00	0.70	52 418 686	malloc

- Tracklet and cell finding part are responsible for ~90% of the computing time: try to bring them to GPUs!
- `atan2f` is another computational hotspot that can be optimised using a fast approximate algorithm.

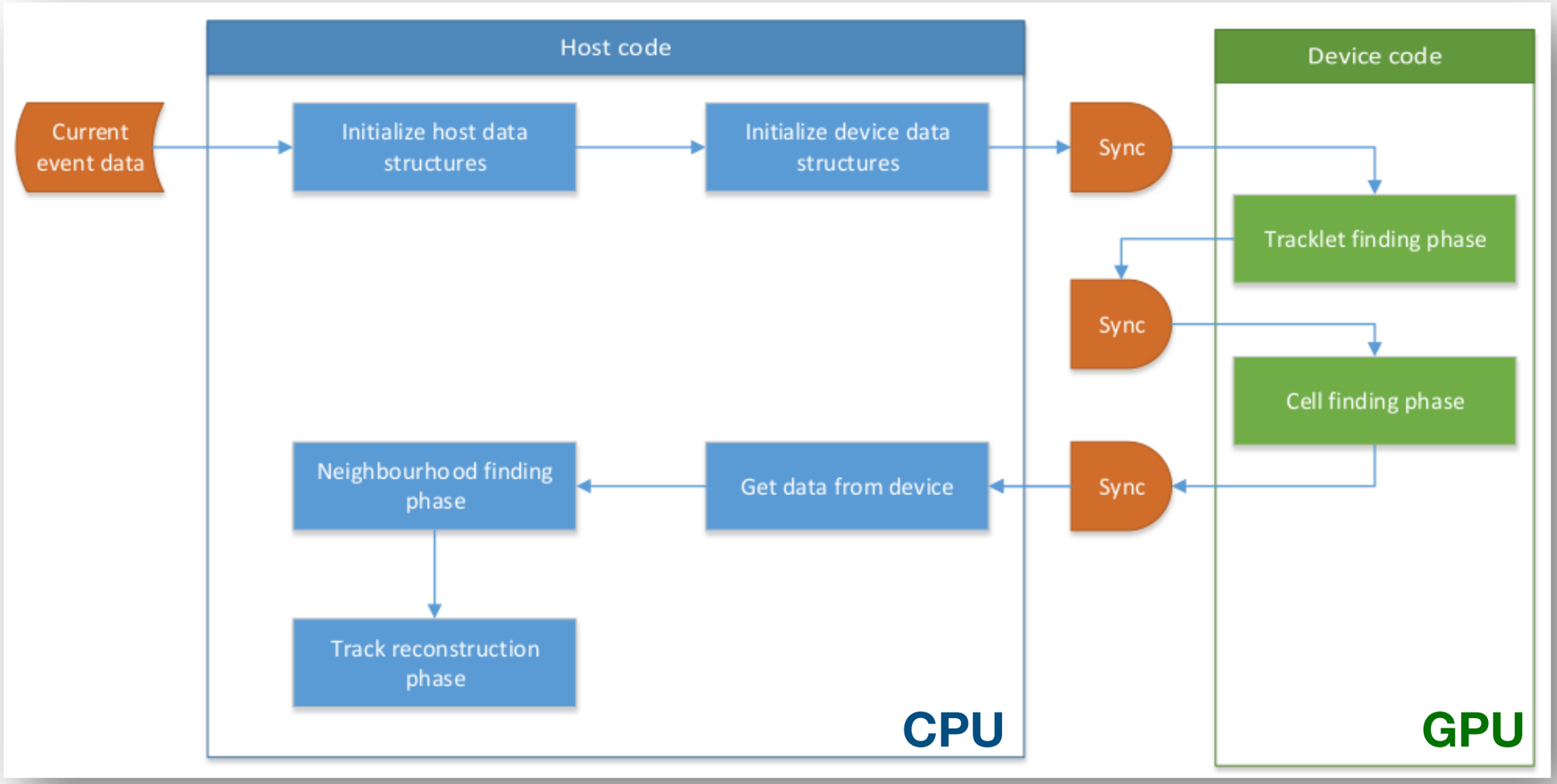
Using CUDA

- CUDA programming model assumes that GPU threads execute on a physically separated device that operates as a coprocessor to the host
- CUDA GPU parallelisation is managed by specific C/C++ functions called kernels that, when called, are executed N times in parallel by N different CUDA threads
- Both the host and the device maintain their own separate memory spaces in DRAM, referred to as host memory and device memory, respectively
- The presence of two separate memory spaces means that data must be transferred from host to device memory, to be processed by kernels, and back to host memory, to be processed by the host



GPU version structure

I.Colonnelli master thesis



- Initialisation of data structure is performed on the host side (CPU)
- At least/Currently three device-host synchronisation barriers
- Final processing of cells to reconstruct track candidates on the CPU

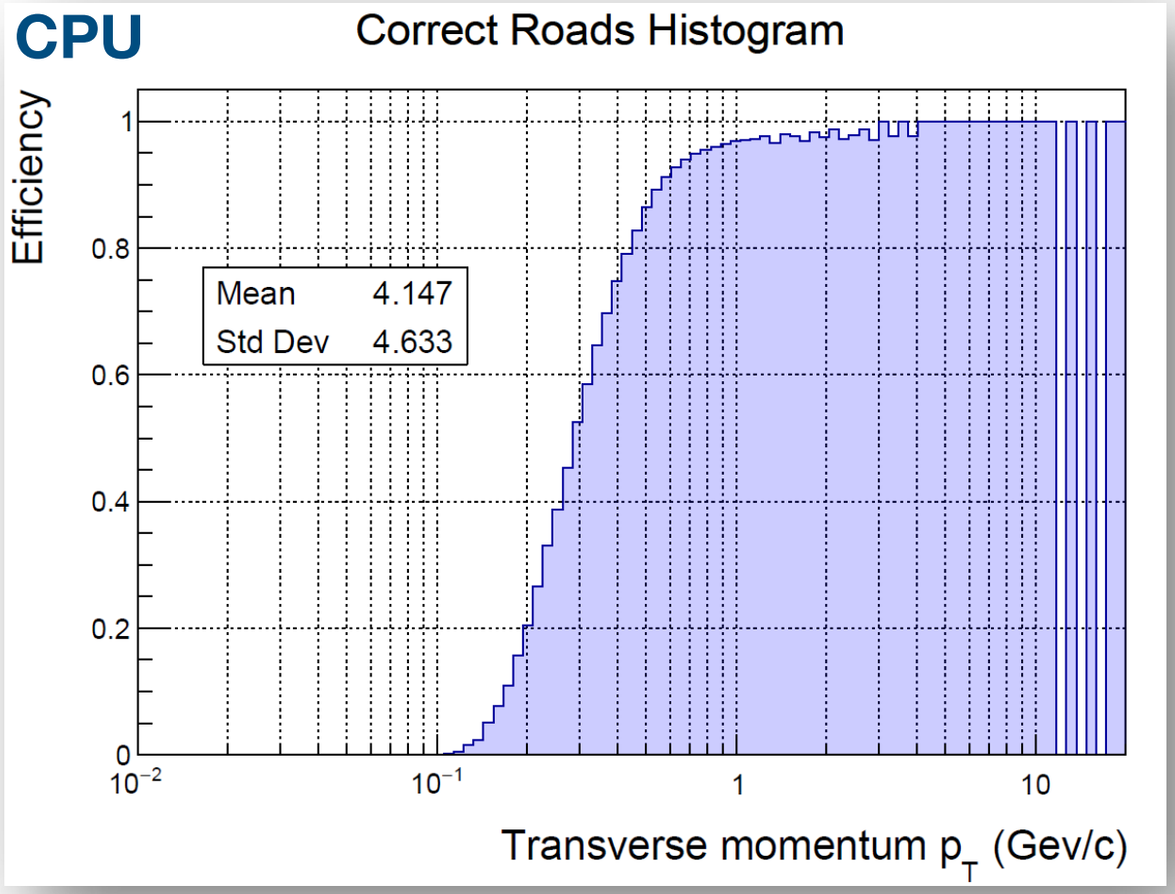
Parallel tracklet (cell) finding

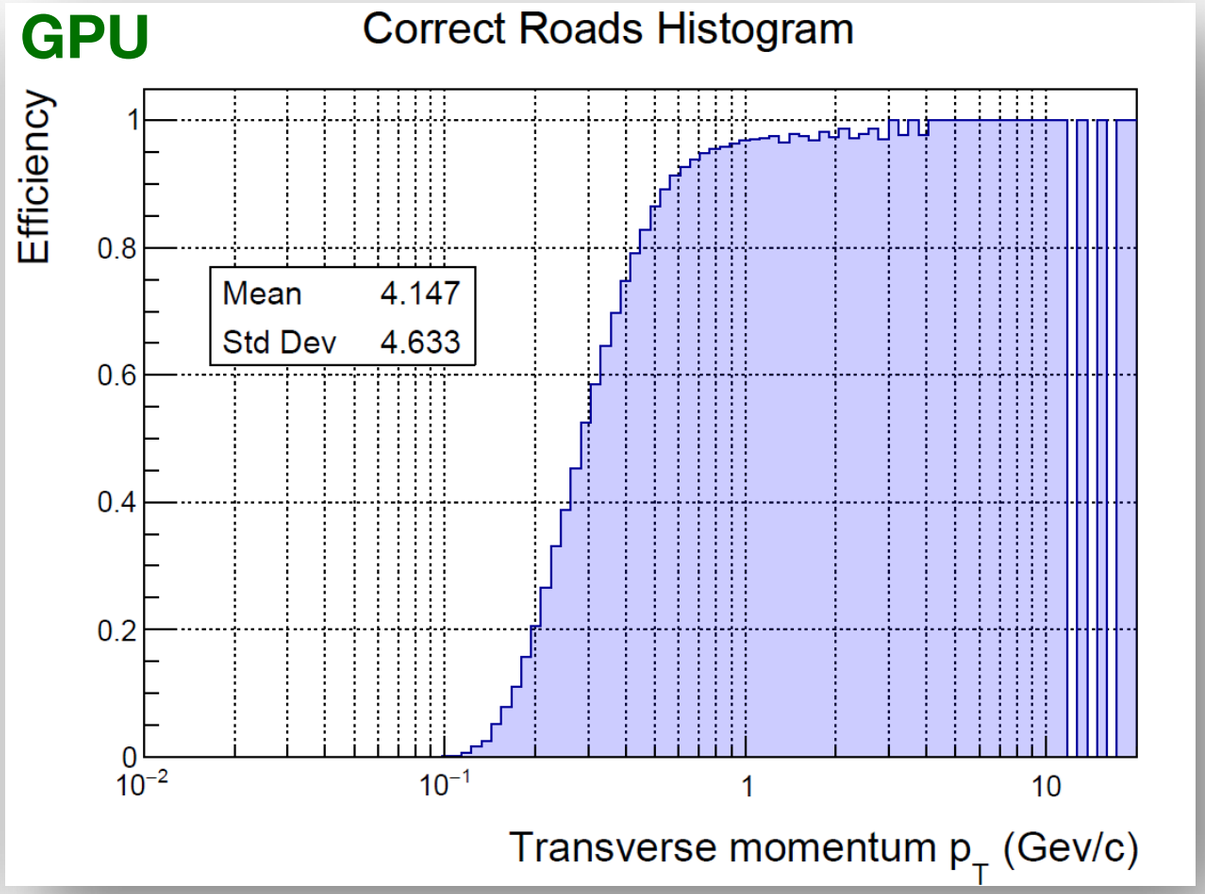
Tracklet finding and cell finding are dual, thus the same strategy is adopted in both cases. Here the parallel tracklet finding description:

- Each CUDA thread processes one cluster on Layer n looking for good matches on Layer n+1
- Whenever a good match is found a tracklet is added atomically in the first free position in the output tracklet array
- The number of valid reconstructed tracklets for each cluster is stored in a separate array
  - Once the tracklet finding is finished, the prefix sum (scan) algorithm is used on this data structure to create an index table used for the subsequent cell finding step
- The counting sort is then used to reorganise the tracklets

Pattern recognition efficiency

I.Colonnelli master thesis





- Track candidate reconstruction efficiency for pions with 7 hits on the ITS is the same for both the CPU (serial) and the GPU version of the algorithm.
- The performance is compatible with that obtained with a single tracking iteration with the AliRoot CA prototype

Timing performance

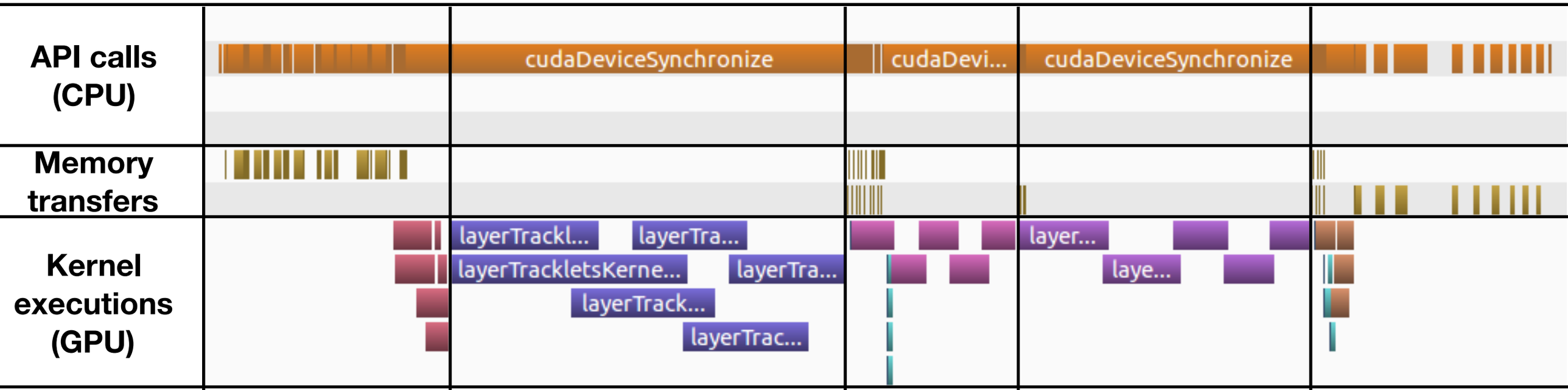
I.Colonnelli master thesis

# Vertices		1	2	4	5
Context init	Min	5.3	9.1	19.3	24.9
	Mean	6.7	13.0	25.1	32.5
	Max	11.2	22.2	63.1	94.3
Tracklet finding	Min	1.8 (18.2)	6.3 (76.4)	27.0 (329.1)	44.2 (517.3)
	Mean	3.1 (39.4)	9.0 (139.0)	35.2 (549.6)	55.1 (810.0)
	Max	5.3 (70.6)	13.3 (269.3)	47.2 (1094.0)	70.5 (1641.0)
Cell finding	Min	2.0 (10.2)	6.5 (50.9)	35.4 (350.3)	60.6 (646.9)
	Mean	5.0 (28.5)	15.0 (134.5)	72.5 (862.4)	123.1 (1520.3)
	Max	7.9 (44.4)	20.9 (188.7)	104.4 (1213.6)	166.0 (2140.7)
Total	Min	9.6 (33.4)	23.4 (135.0)	84.2 (696.2)	133.3 (1186.2)
	Mean	16.7 (75.0)	40.2 (285.4)	139.5 (1437.7)	219.9 (2362.8)
	Max	27.0 (110.3)	59.9 (446.5)	220.7 (2326.1)	317.9 (3810.5)

- Serial CPU implementation timing in parenthesises
- Speed-up of factor 12.7 in the tracklet finding
- Speed-up of factor 5.7 in the cell finding
- Overall speedup 4.5
  - Trivial CPU parallelisation can already reduce the gap

- Test on 100 Pb-Pb central (0-5%) HIJING events simulated with AliRoot
- Pile-up obtained stacking several Pb-Pb events
- GPU speed-up increases with increasing pile-up (up to factor 10 with 5 vertices)
- Total times account also for the cells post processing (i.e. track candidate reconstruction)

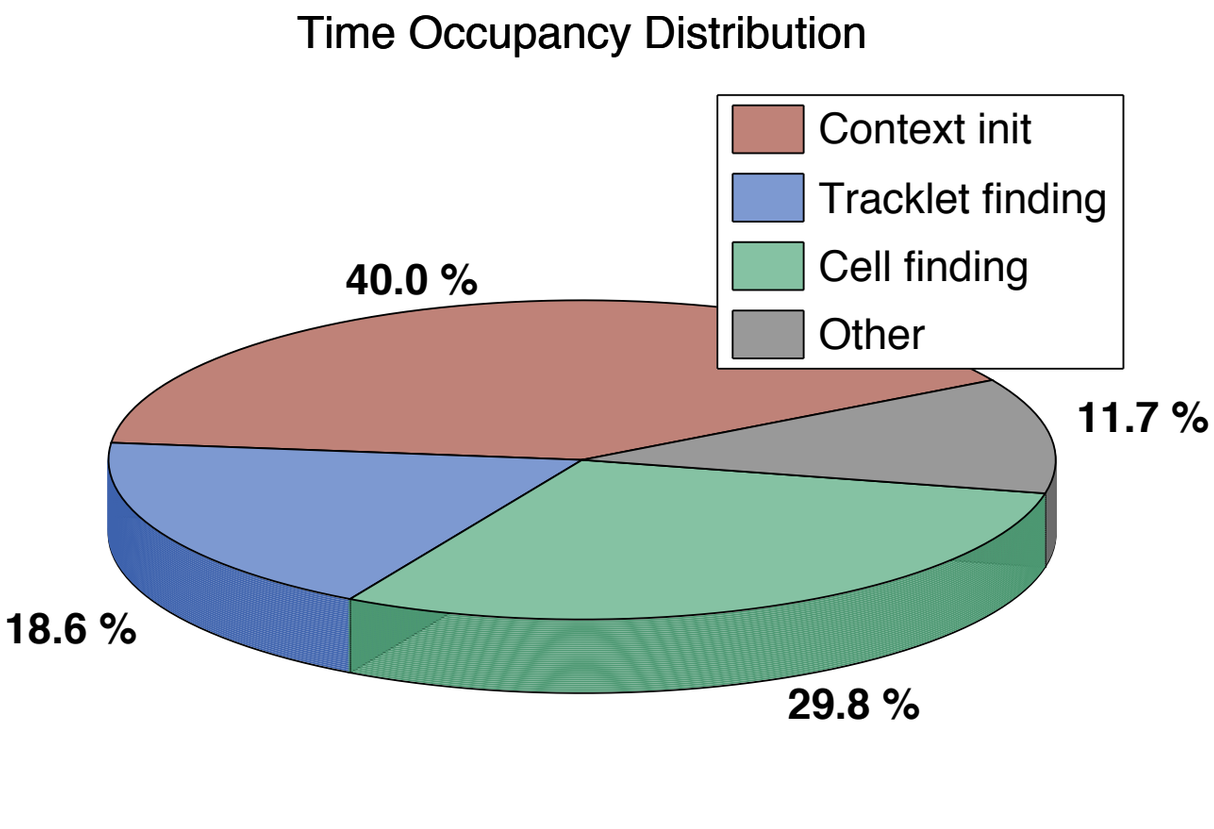
GPU version computational hotspots



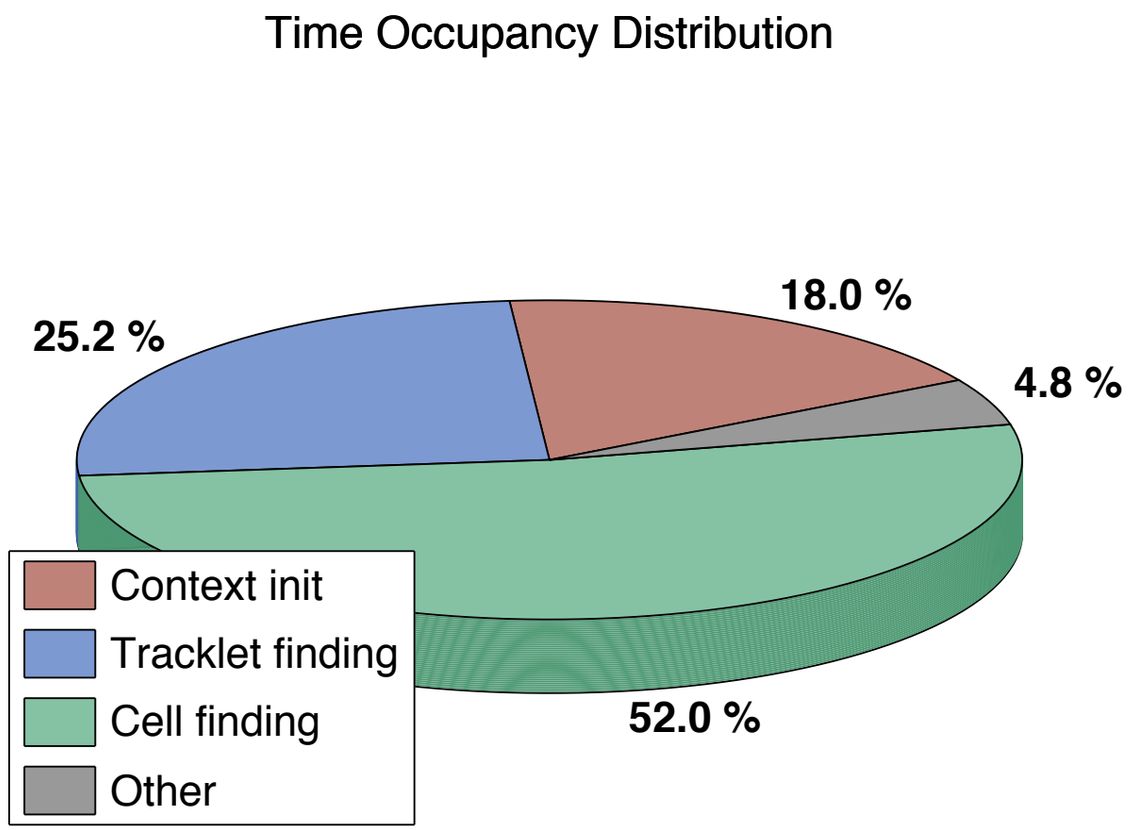
	Data to GPU + index table compilation	Tracklet finding	Scan + Counting sort	Cell finding	Scan + Counting sort + copy to host

- GPU full exploitation in the tracklet finding and cell finding phase is limited by the register pressure
  - Further optimisation of the usage of the shared memory is needed to improve the performance
- Asynchronous copy of data between host and device will improve the kernel initialisation time

GPU version computational hotspots 2



1 vertex



4 vertices

- Context initialisation on CPU and for GPU memory allocation is now a significant contribution to overall processing time for single events
  - Vectorisation, change of the data structures and multi-threading will be investigated to improve that
- Track fitting figures of merit will be available as soon as the porting in the O2 framework will be ready

Current status and outlook

- Standalone version of the pattern recognition algorithm fully tested and working
  - Both CPU (serial) and GPU (CUDA) implementation there
  - OpenCL support coming up…
- O^2 framework integration ongoing (code in private forks)
  - Track fitting and MC truth handling inherited by the framework
  - Testing will start soon and when successful PR will come.
- Future optimisations for GPU
  - Asynchronous copy of data between host and device
  - Better use of shared memory
  - Move full processing there? Some algorithm challenges…
- Future optimisations for CPU
  - Vectorisation using *Vc* and multi-threading

Testbed platform

- CPU: Intel i7-7700K (4.2GHz, 8MB Cache)
- RAM: Corsair DDR4 32GB (2×16) 2133MHz
- GPU: NVIDIA GeForce GTX 970 (SM 52, 1664 CUDA cores)
- OS: Linux Ubuntu 16.04 LTS

- C++ Host Compiler: Clang 3.8.0-2ubuntu4, with -O3 optimisation flag
- Device Toolkit: CUDA Toolkit V8.0.61