Acceleration of the PndSttCellTrackFinder by implementing on a GPU

June 8, 2015

Jette Schumann

Structure of the Trackfinder

PndSttCellTrackletGenerator.cxx

1. Primary tracklets: use of cellular automaton
2. Combination of tracklets: use of Riemann Fit
3. Add remaining hits to best fitting track

Faulty: Fit to center of the tubes

PndSttHitCorrector.cxx

Correct points for Riemann Fit by calculating tangents to the isochrones

PndSttCellTrackFinderData.cxx

For data exchange

Parallelize!

How to parallelize?

Graphics Processing Unit (GPU)

- General Purpose GPUs (GPGPUs)
- much more processing cores than a CPU
- for massively parallel calculations

CUDA

- API model for parallel computing on GPUs created by NVIDIA
- CUDA C: extension of C/C++
- enables task and data parallelism

How to parallelize?

Parallelism at two levels

Algorithm-level

parallelize the trackfinding algorithm in itself

- useful for computation-intensive parts

Event-level

execute the trackfinder for hundreds/thousands of events at the same time

- promises high speedup

Structure of the Trackfinder

PndSttCellTrackletGenerator.cxx

1. Primary tracklets: use of cellular automaton
2. Combination of tracklets: use of Riemann Fit
3. Add remaining hits to best fitting track

Faulty: Fit to center of the tubes

Start

→ Quick reminder

PndSttHitCorrector.cxx

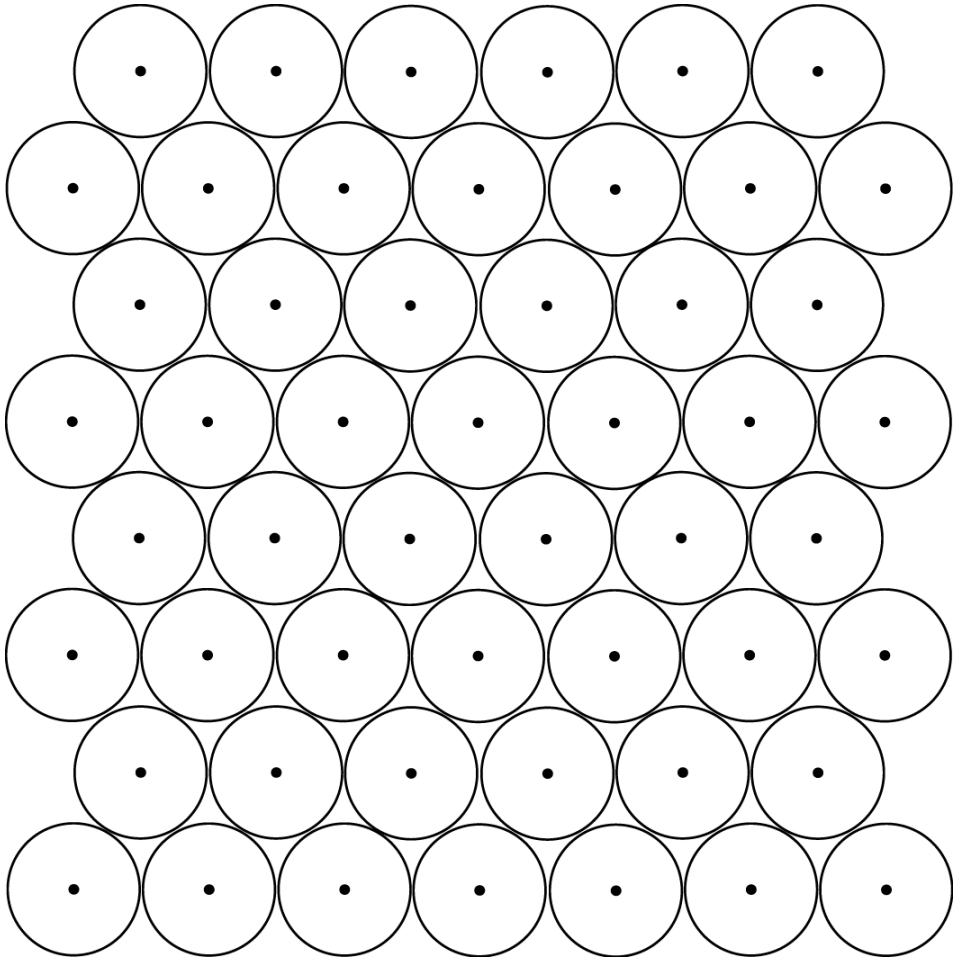
Correct points for Riemann Fit by calculating tangents to the isochrones

PndSttCellTrackFinderData.cxx

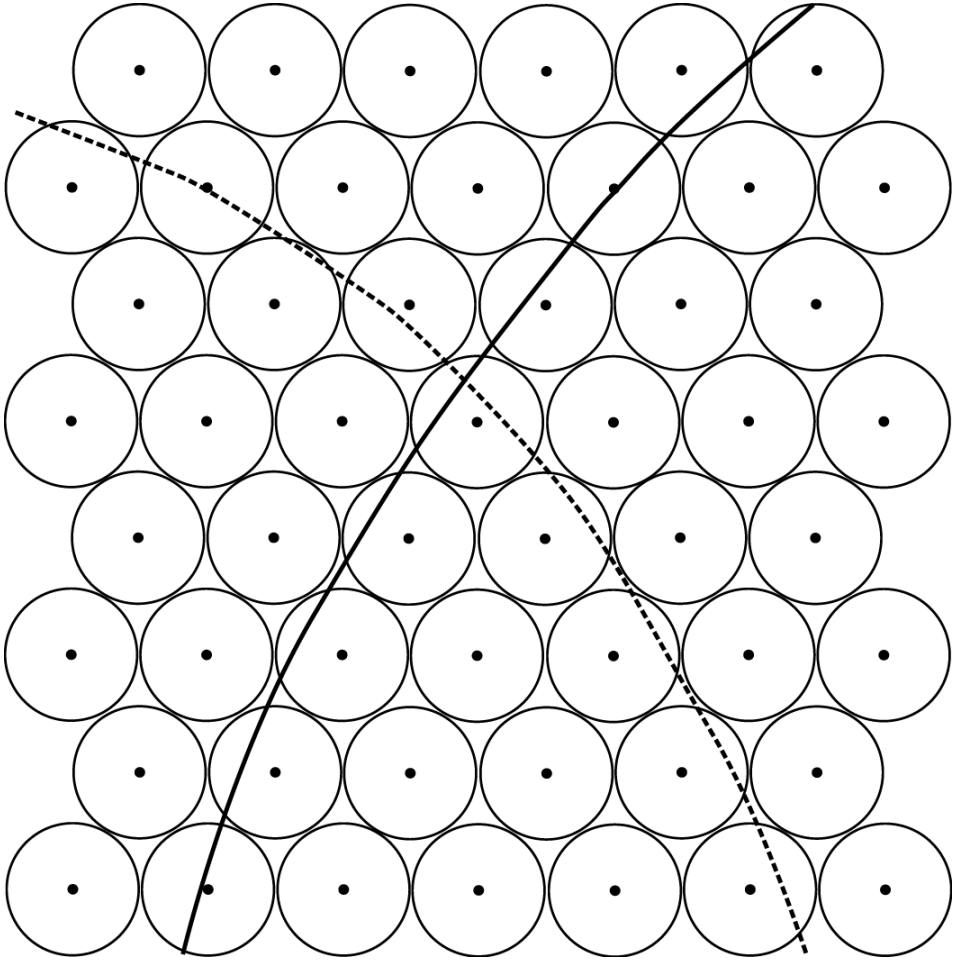
For data exchange

Parallelize!

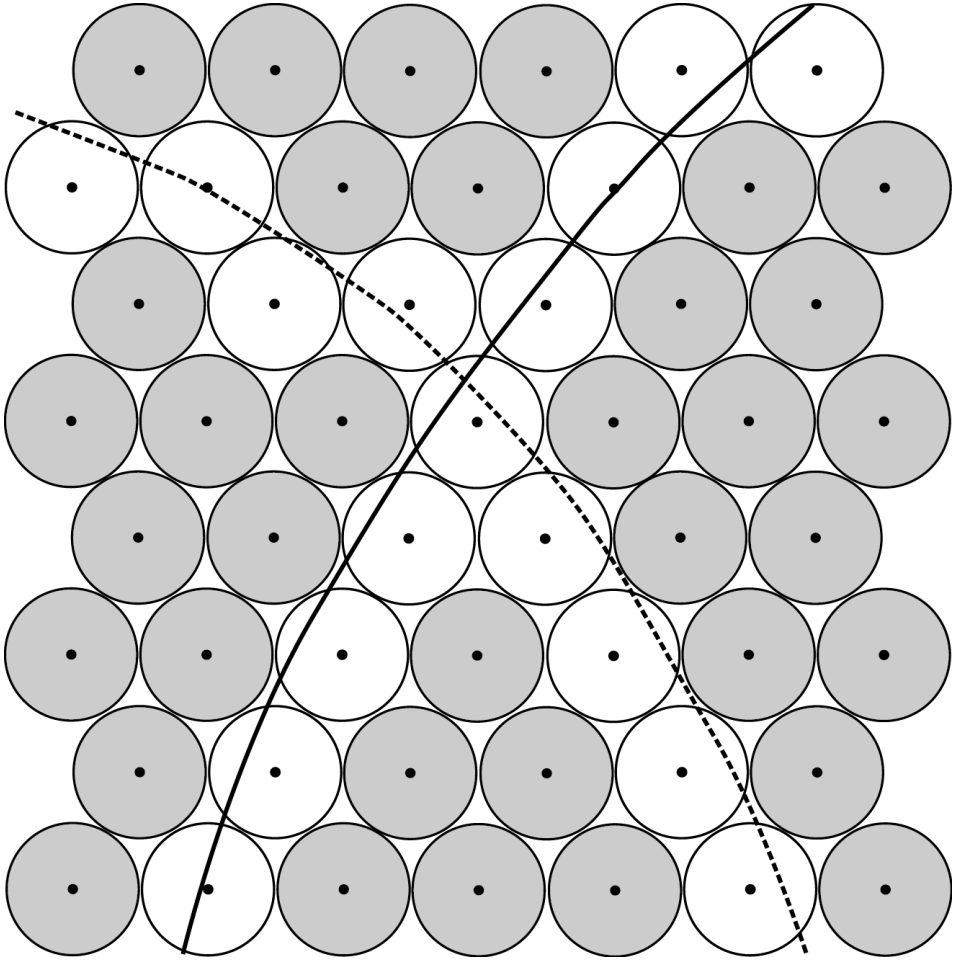
TrackletGenerator
Cellular Automaton



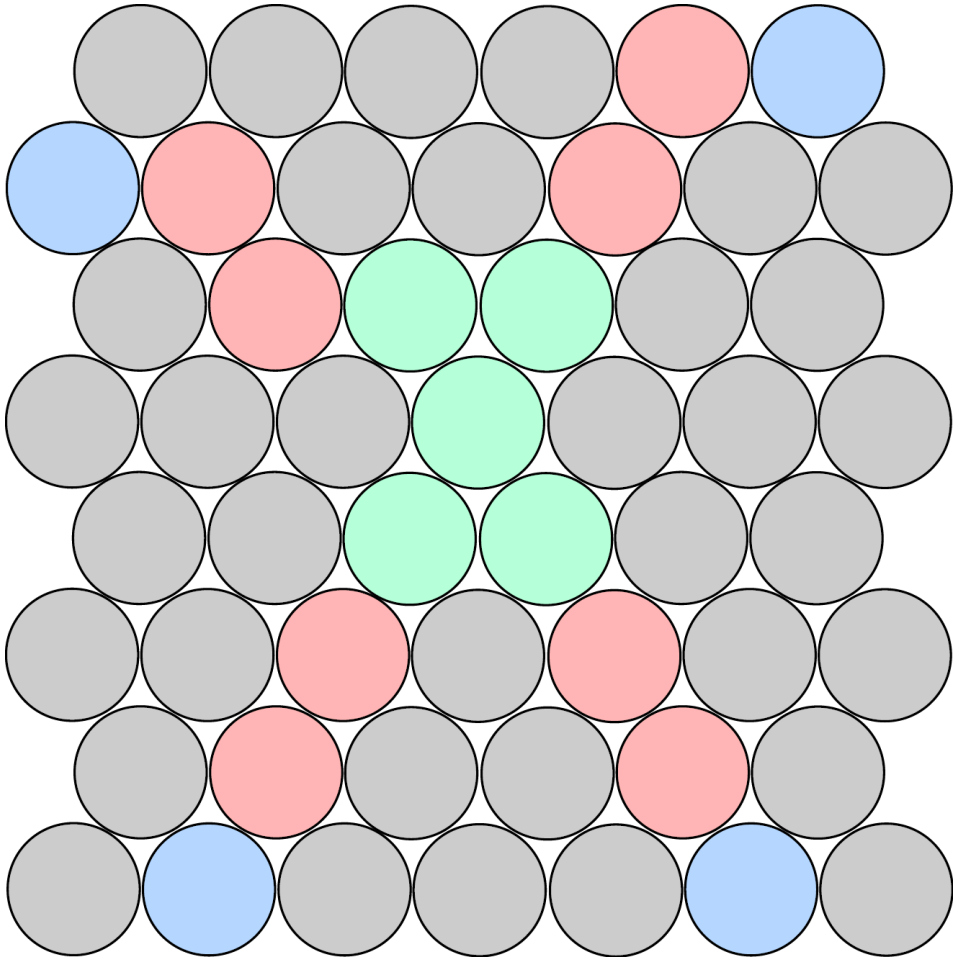
TrackletGenerator
Cellular Automaton



TrackletGenerator
Cellular Automaton



TrackletGenerator
Cellular Automaton



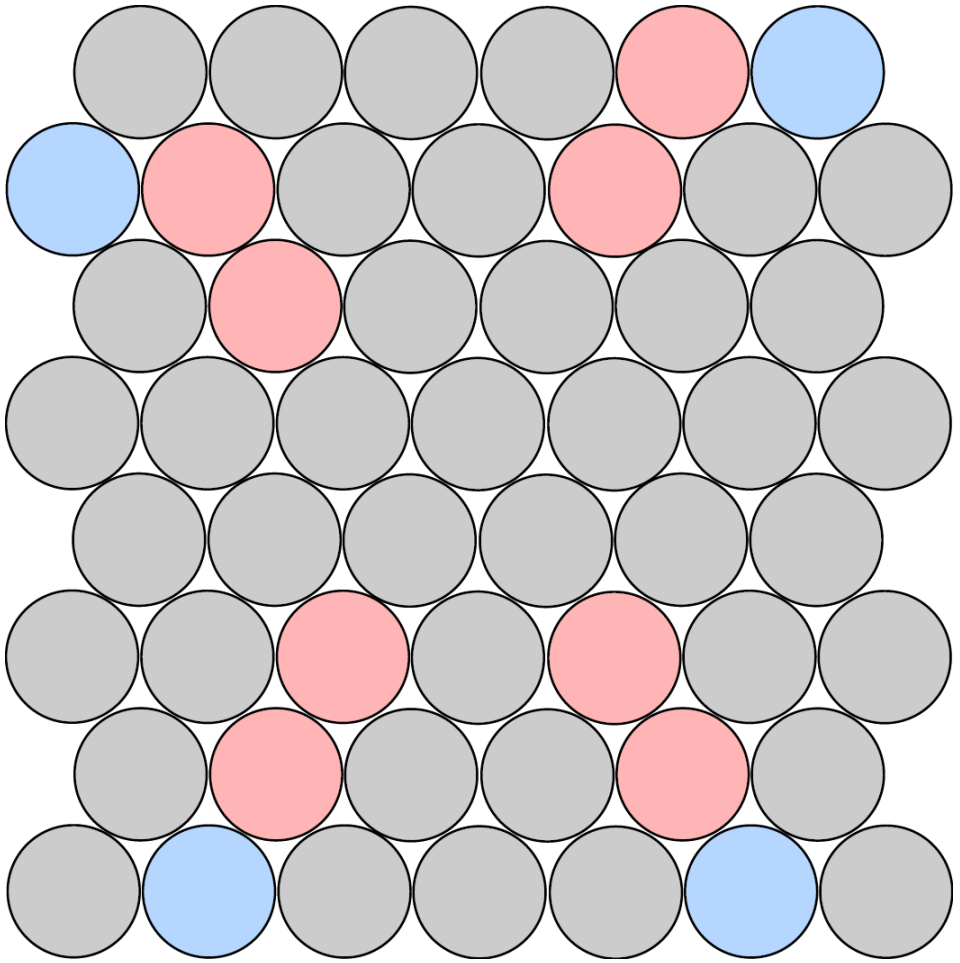
Number of Neighbors:

1 unambiguous

2

> 2 ambiguous

TrackletGenerator
Cellular Automaton – Unambiguous hits



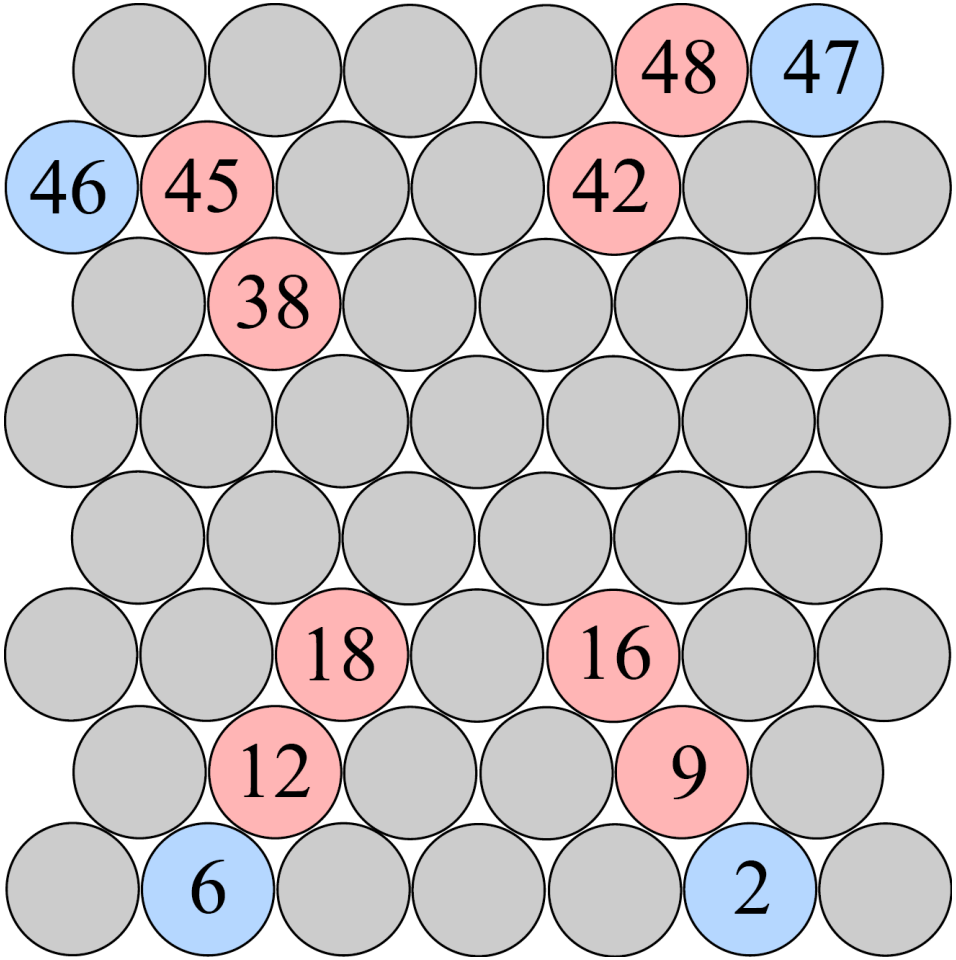
Number of Neighbors:

1 unambiguous

2

> 2 ambiguous

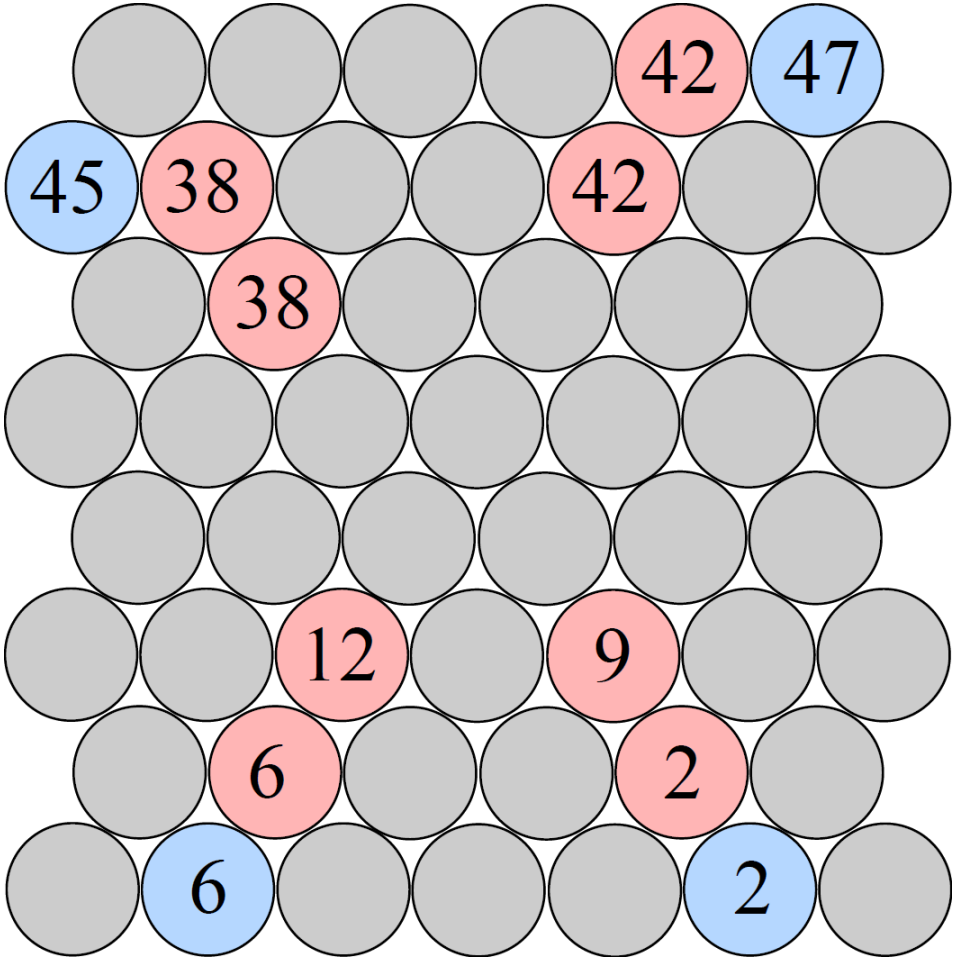
TrackletGenerator
Cellular Automaton – Unambiguous hits



Cell: Straw with hit and one or two neighbors

Rule: Compare your status with the status of your neighbors and take the smallest one

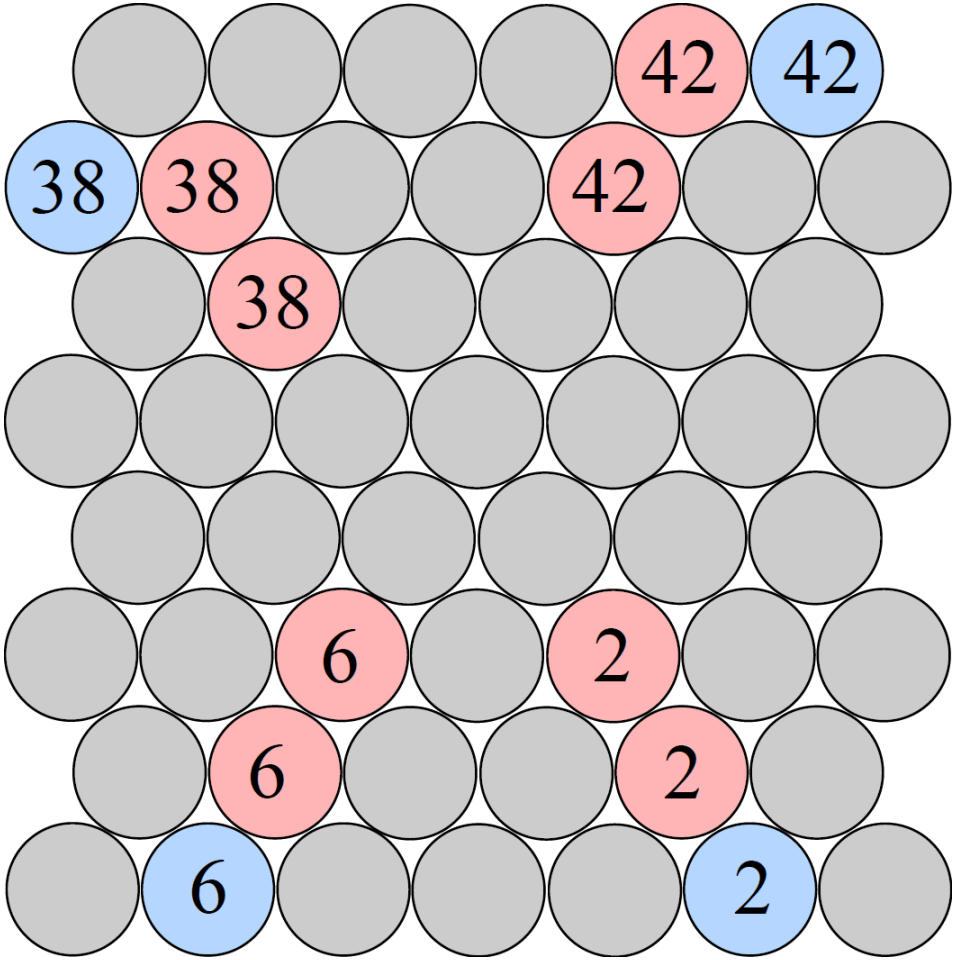
TrackletGenerator
Cellular Automaton – Unambiguous hits



Cell: Straw with hit and one or two neighbors

Rule: Compare your status with the status of your neighbors and take the smallest one

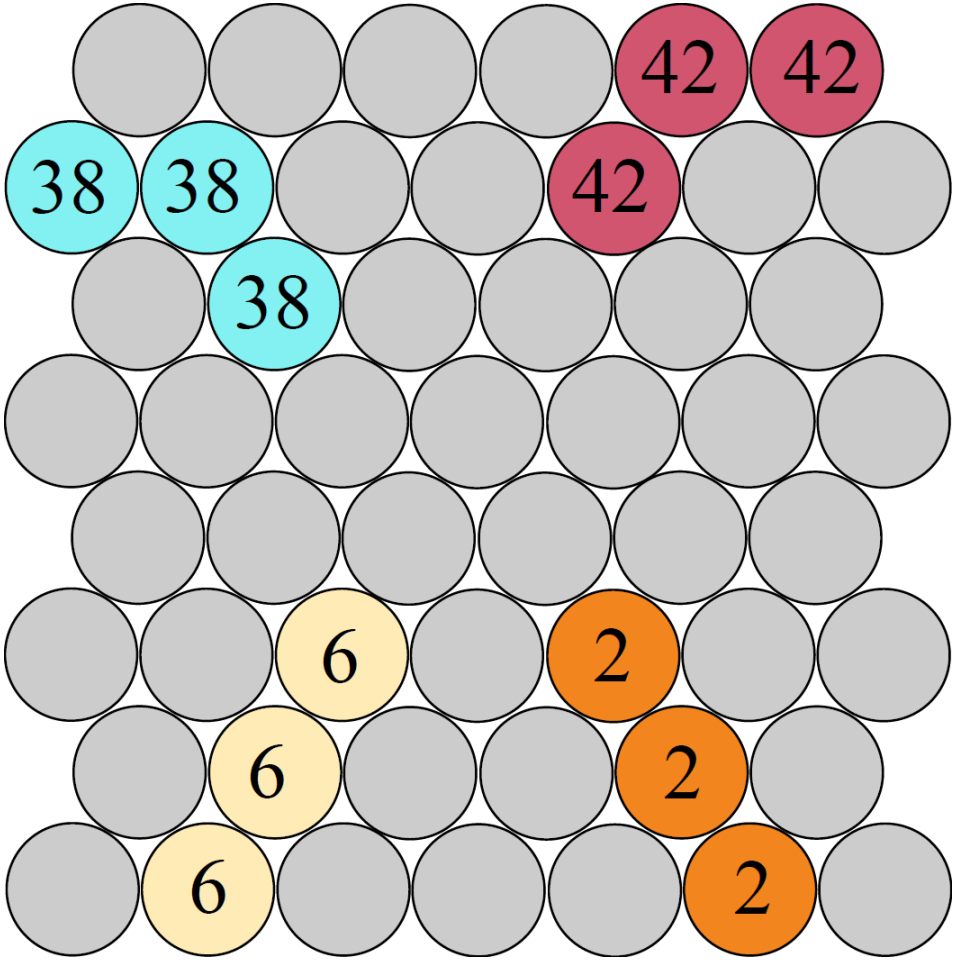
TrackletGenerator
Cellular Automaton – Unambiguous hits



Cell: Straw with hit and one or two neighbors

Rule: Compare your status with the status of your neighbors and take the smallest one

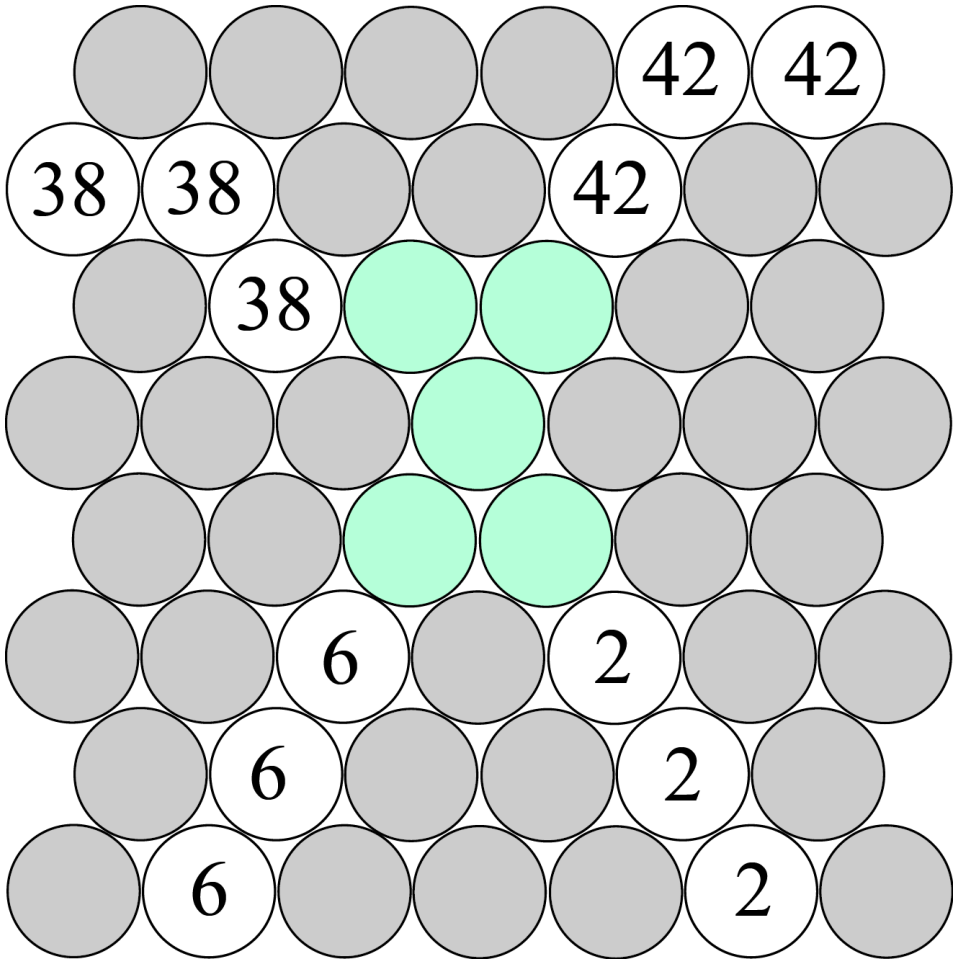
TrackletGenerator
Cellular Automaton – Unambiguous hits



Cell: Straw with hit and one or two neighbors

Rule: Compare your status with the status of your neighbors and take the smallest one

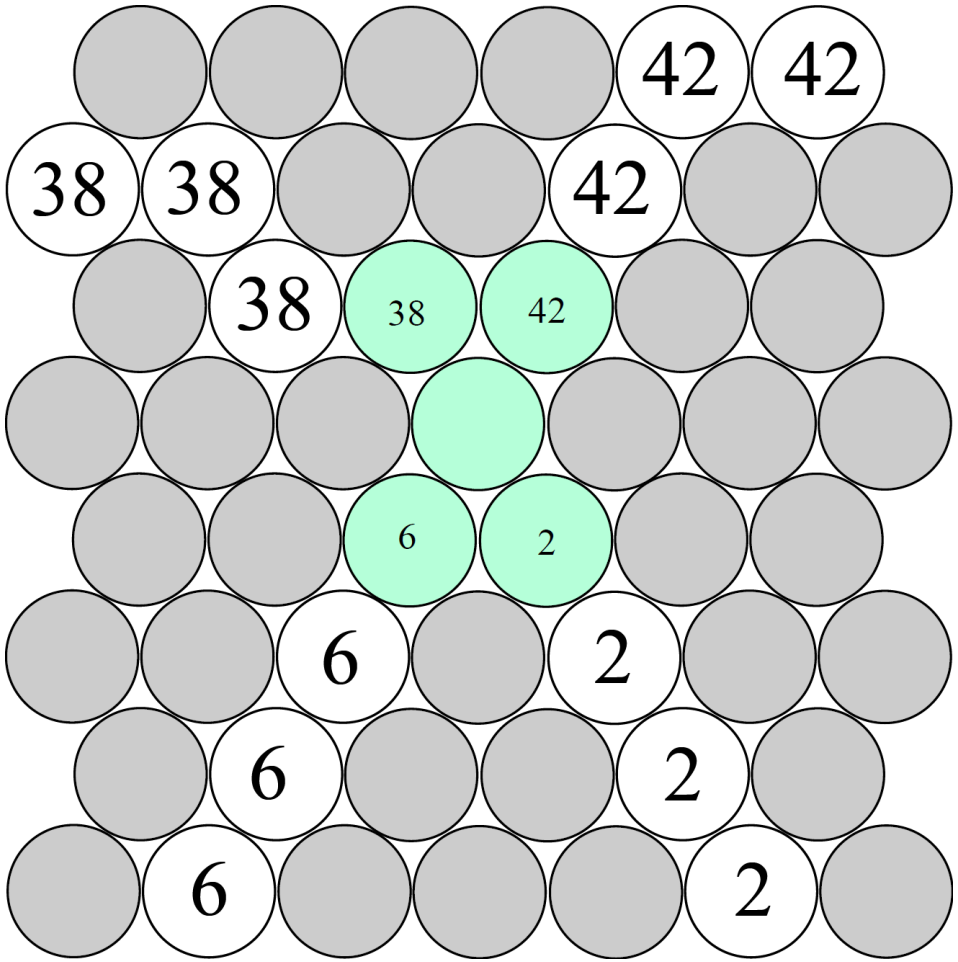
TrackletGenerator
Cellular Automaton – Ambiguous hits



Cell: Straw with hit and more than two neighbors

Rule: Copy the status of all your neighbors into your status

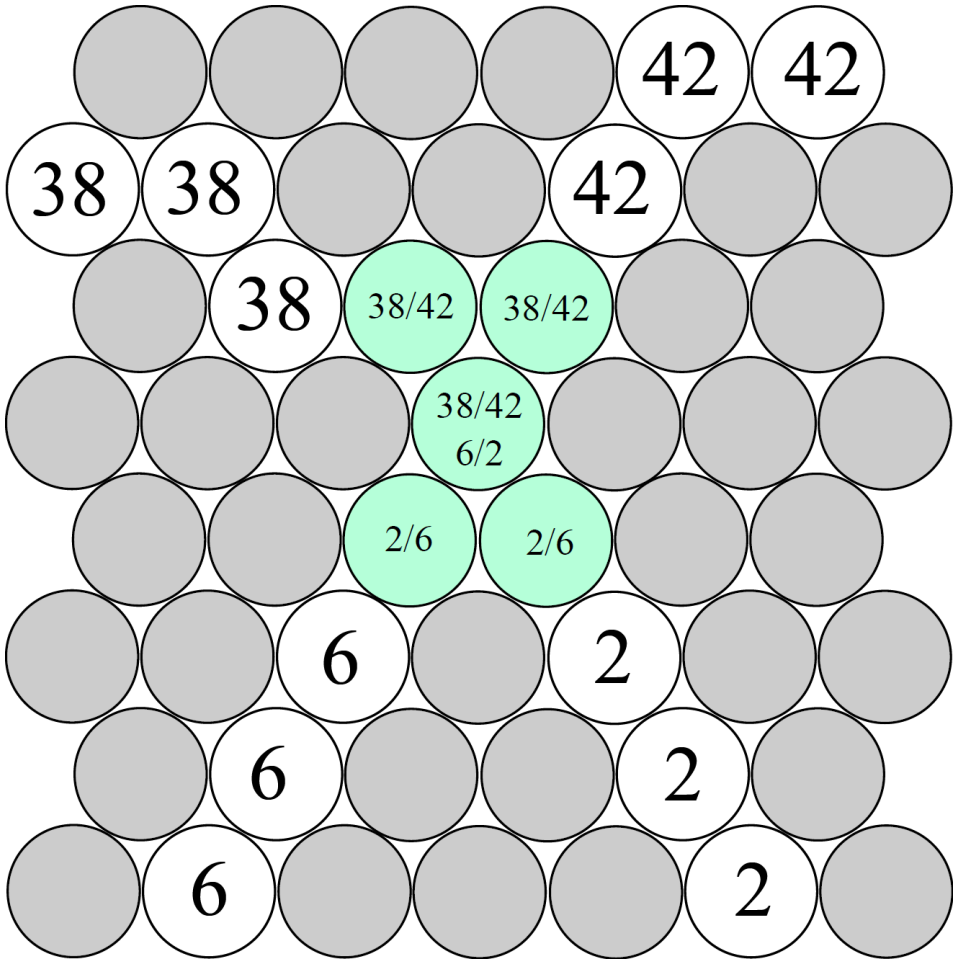
TrackletGenerator
Cellular Automaton – Ambiguous hits



Cell: Straw with hit and more than two neighbors

Rule: Copy the status of all your neighbors into your status

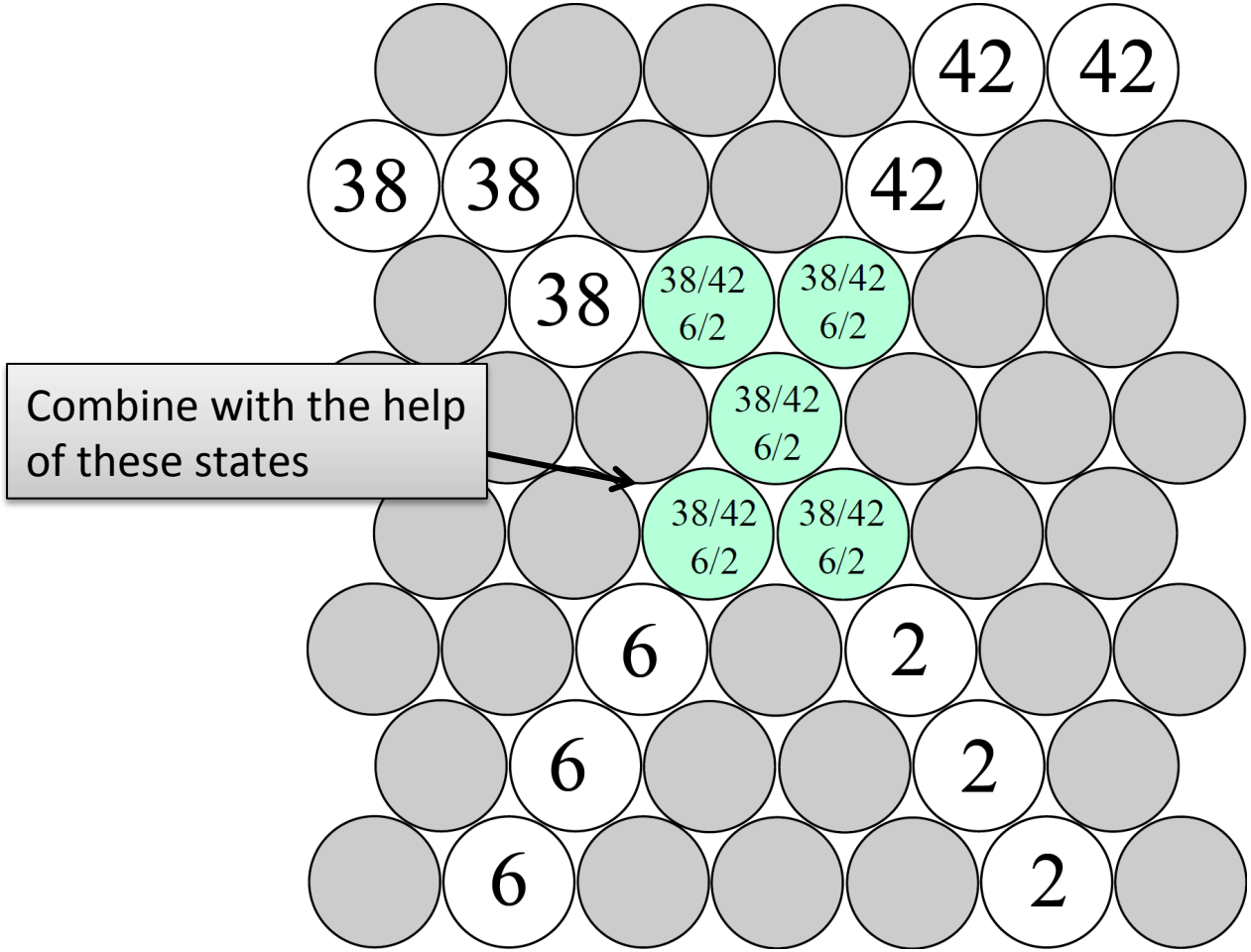
TrackletGenerator
Cellular Automaton – Ambiguous hits



Cell: Straw with hit and more than two neighbors

Rule: Copy the status of all your neighbors into your status

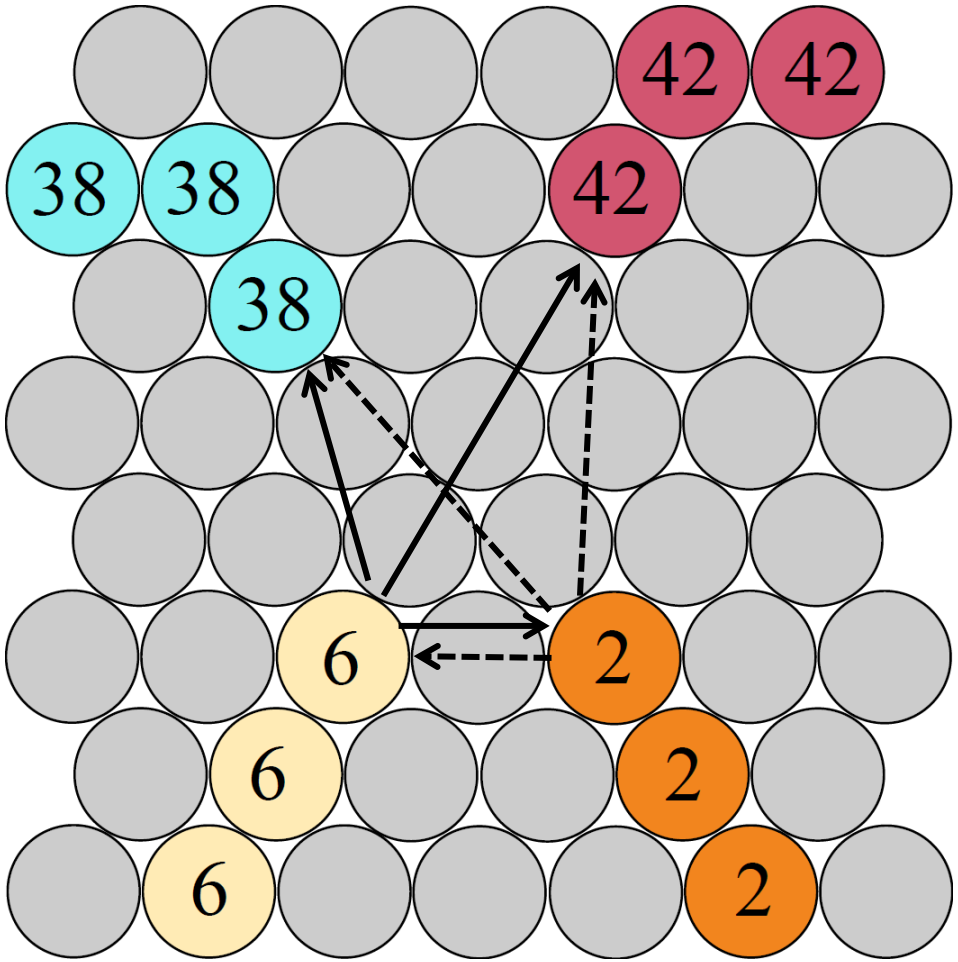
TrackletGenerator
Cellular Automaton – Ambiguous hits



Cell: Straw with hit and more than two neighbors

Rule: Copy the status of all your neighbors into your status

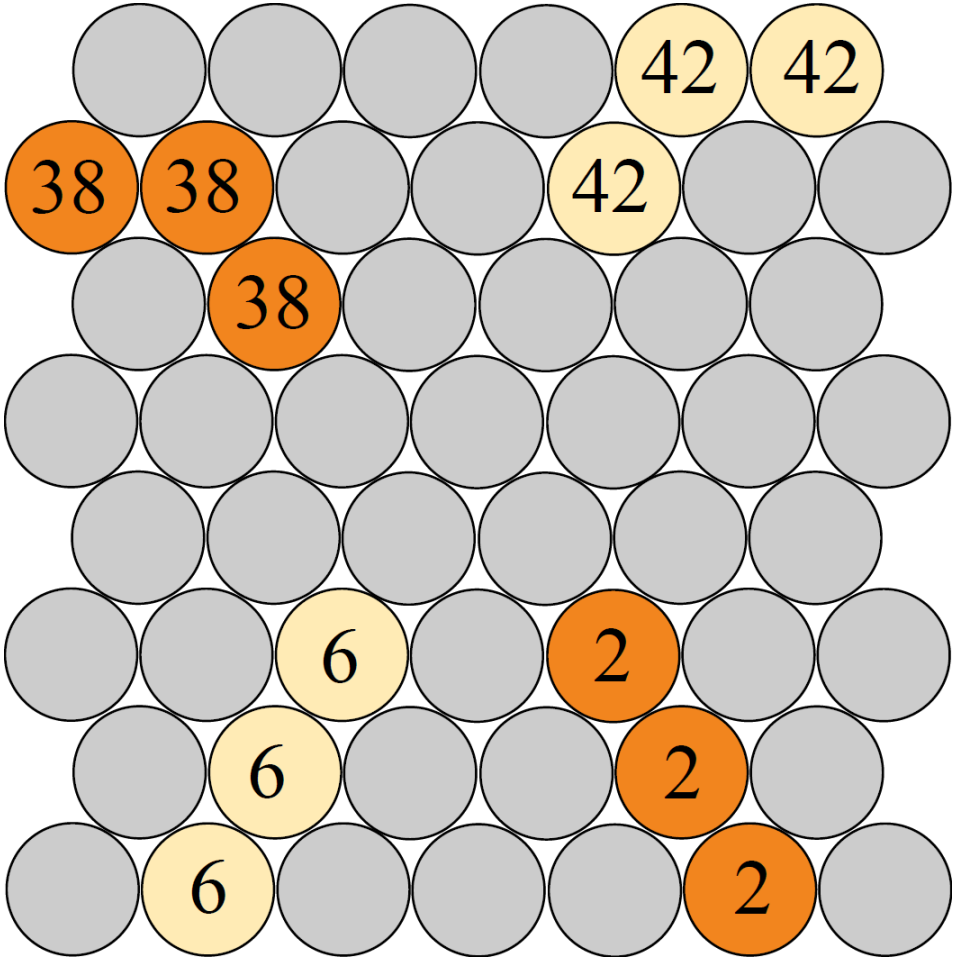
TrackletGenerator
Cellular Automaton – Combination



For inner tracklets:
test 3 combinations
with Riemann Fit

→ take the best one

TrackletGenerator
Cellular Automaton – Result



For inner tracklets:
test 3 combinations
with Riemann Fit

→ take the best one

CPU-Version
Data structures

- tubeID of active cells <-> tubeIDs of neighboring tubes that signals a hit

```
map<int, vector<int> > fHitNeighbors;
```

- tubeID of unambiguous cell <-> state of the tube

```
map<int, int> fStates;
```

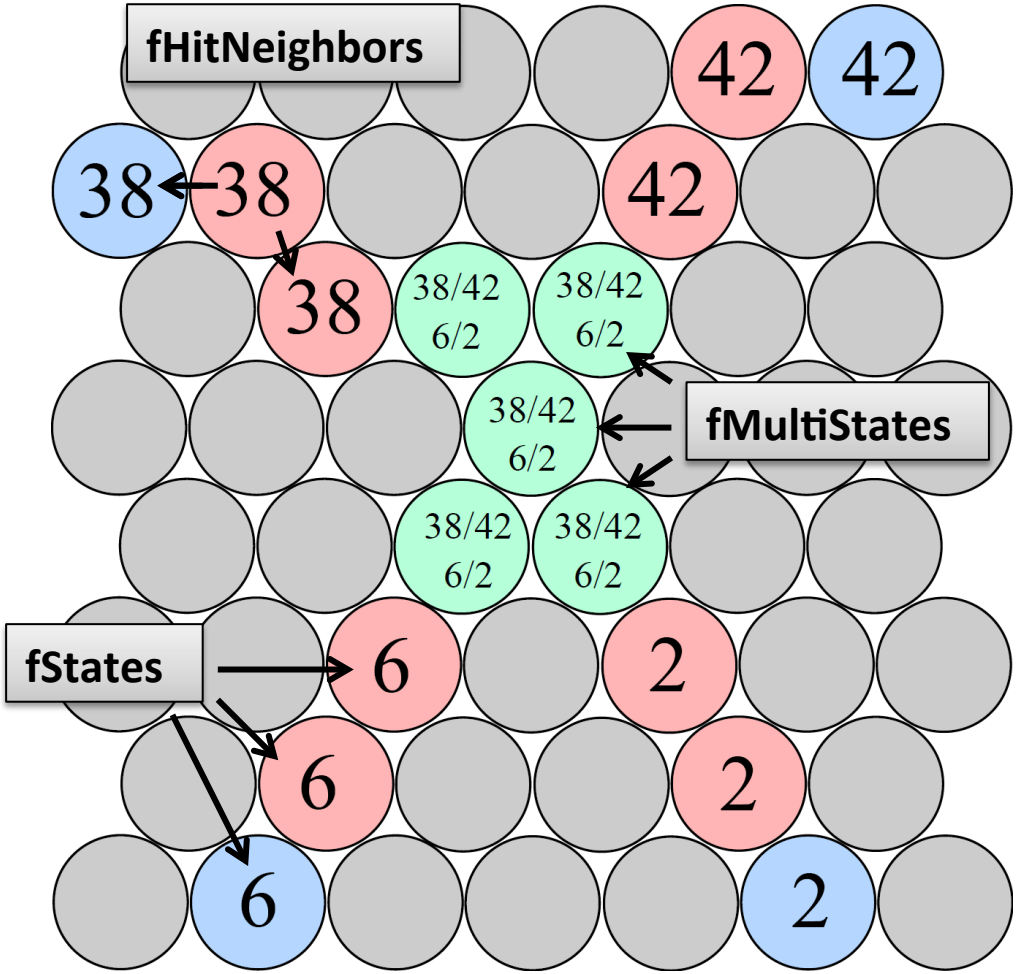
- tubeID of ambiguous cell <-> multistate of the

```
map<int, set<int> > fMultiStates;
```

Dynamic data structures:
- STL
- Flexible size
- Implicit memory management
- Easy in usage

Not available on the GPU!

CPU-Version
Data structures



Number of Neighbors:

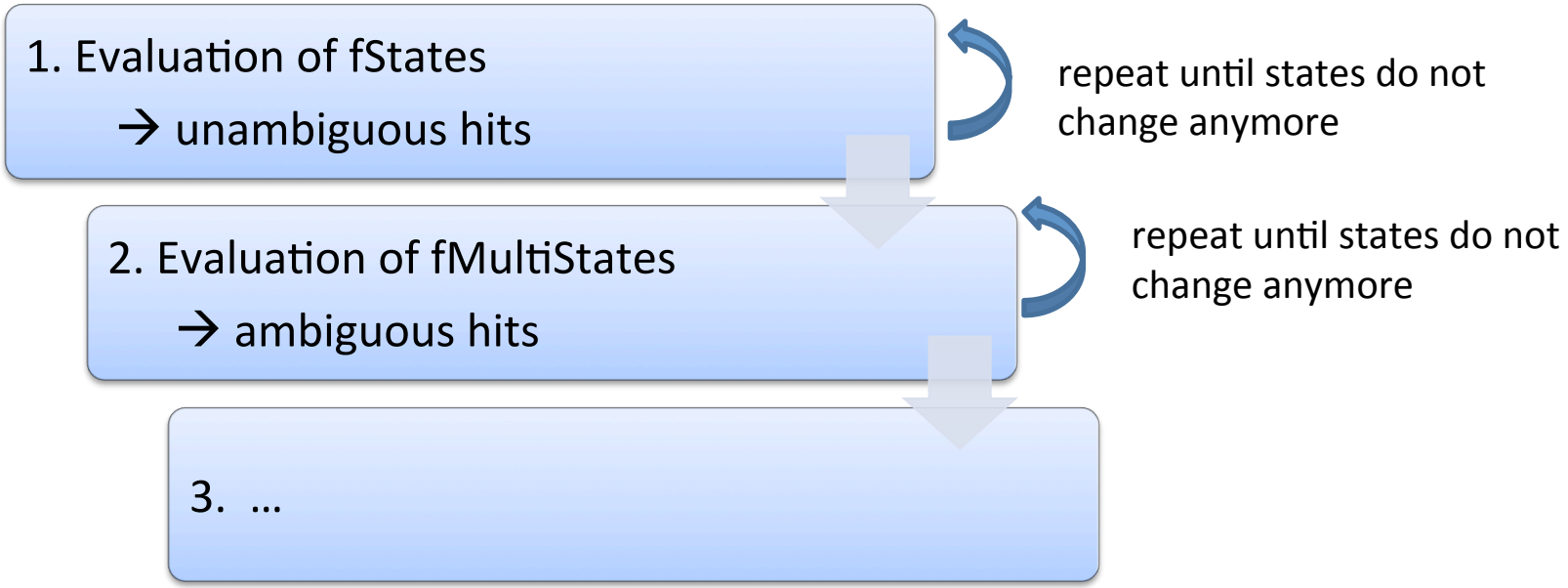
1 unambiguous

2

> 2 ambiguous

CPU-Version
Cellular automaton

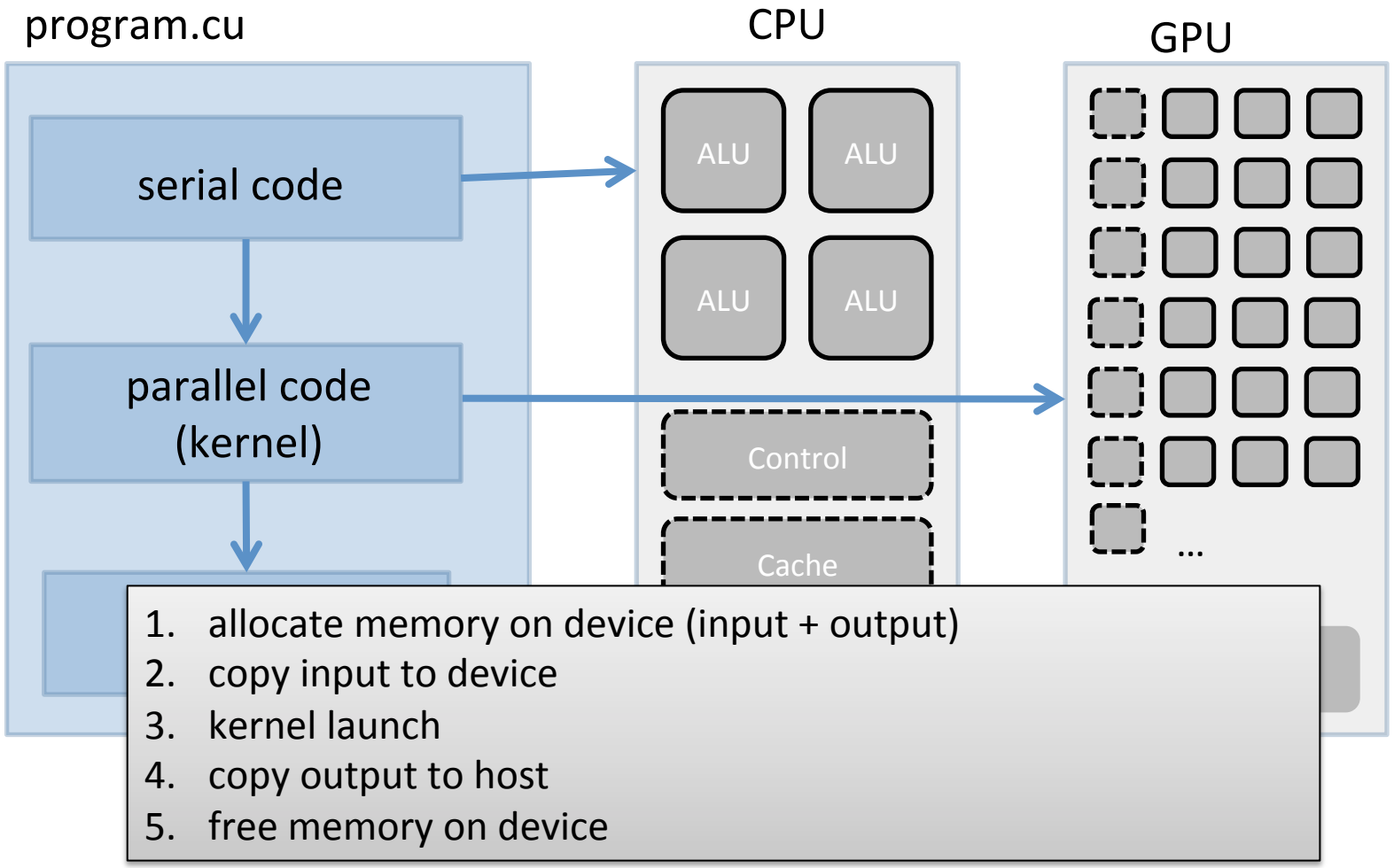
Serially executed steps:



→ Try to update all states in parallel

fMultistates depend on final fStates

GPU-Version
Programming model



GPU-Version
Data structures

static event data

- Number of tubes and tubeIDs
- Neighboring tubes
- skewed or not skewed

→ Copy once

dynamic event data

- SttHits
  - fHitNeighbors
  - fStates
  - fMultiStates

→ Copy for each event
→ calculate derived data on device

use **arrays with fixed sizes** instead of maps

balancing: memory usage <-> memory management

GPU-Version
Data structures

static event data

- variable number of neighbors: 2-22 (parallel/skewed)

→ wasting memory if one assumes that each tube has 22 neighbors

→ split information into 2 arrays

- skewed/unskewed: range request

```
#define NUM_STRAWS 4542
#define MAX_SKEWED_NEIGHBORS 22
#define MAX_UNSKEWED_NEIGHBORS 6
#define START_TUBE_ID_SKEWED 855
#define END_TUBE_ID_SKEWED 2956

int skewedTubes[NUM_SKEWED_STRAWS*MAX_SKEWED_NEIGHBORS]={0};
int unskewedTubes[NUM_UNSKEWED_STRAWS*MAX_UNSKEWED_NEIGHBORS]={0};
```

GPU-Version
Data structures

dynamic event data

- variable number of hit neighbors: 0-22

→ handle like static neighbor data

- memory for fState/fMultiState is allocated for each tube (hit/no hit)

→ easy access, but much memory

- number of multi-states for one tube is unknown

→ define max number, handle exception

```
#define MAX_MULTISTATE_NUM 20

cudaMalloc((void**) &dev_states, sizeof(int)*NUM_STRAWS);
cudaMalloc((void**) &dev_multiStates,
                    sizeof(int)*NUM_STRAWS*MAX_MULTISTATE_NUM);
```

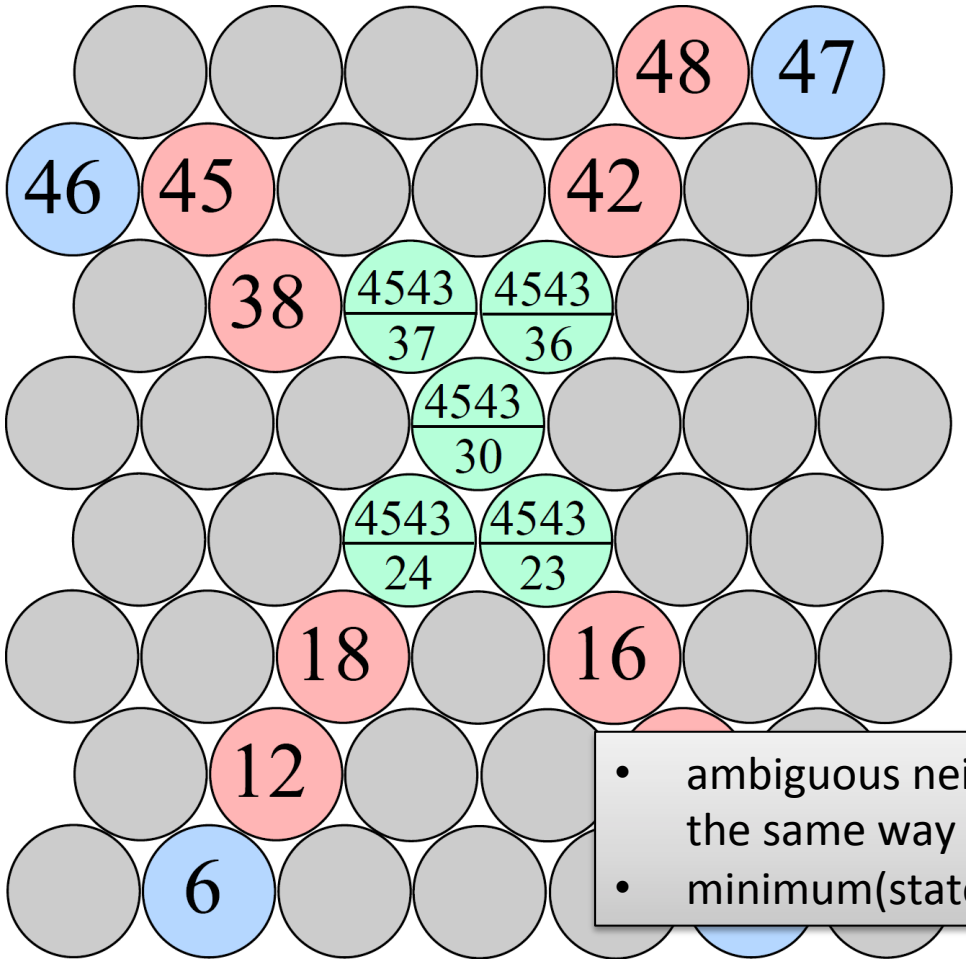
GPU-Version
Cellular automaton

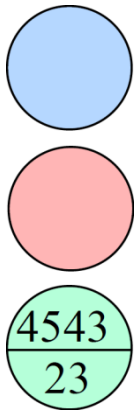
Evaluation of states

- kernel launch: one thread for each hit
- determine minimum in parallel for all hits
- calculate sum of all states in parallel
  → termination condition

- initialize state of ambiguous tubes with NUM_STRAWS+1
  → avoid unneccessary branching

GPU-Version
Cellular automaton – Unambiguous hits





unambiguous with state

ambiguous with max state / tubeID

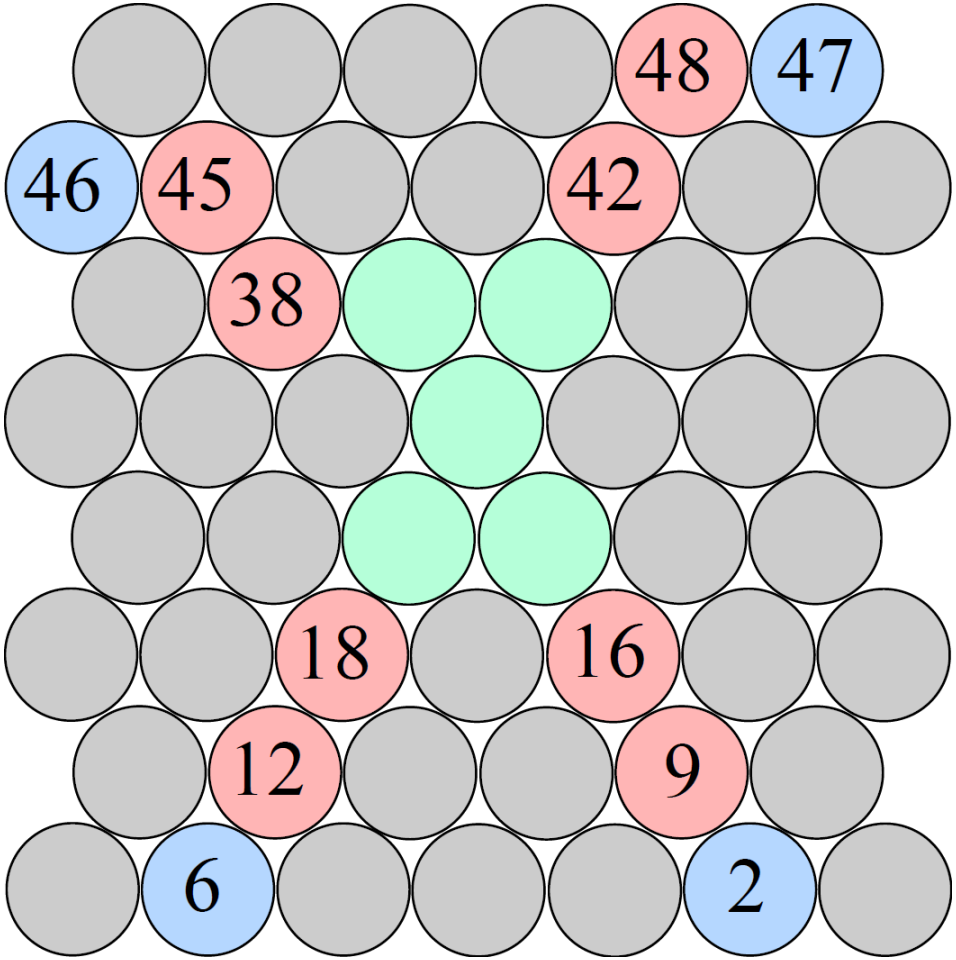
- ambiguous neighbors are treated in the same way as unambiguous ones
- minimum(state, 4543)=state

GPU-Version
Cellular automaton

Evaluation of multi-states

- some multi-states depend on final states
- copy tubeIDs instead of states:
  - outer tracklets: tubeID of first tube = state of this tracklet
  - inner tracklets: tubeID of ending tube of the tracklet is copied
- tubeIDs do not change and are independend of the states
  → parallel execution

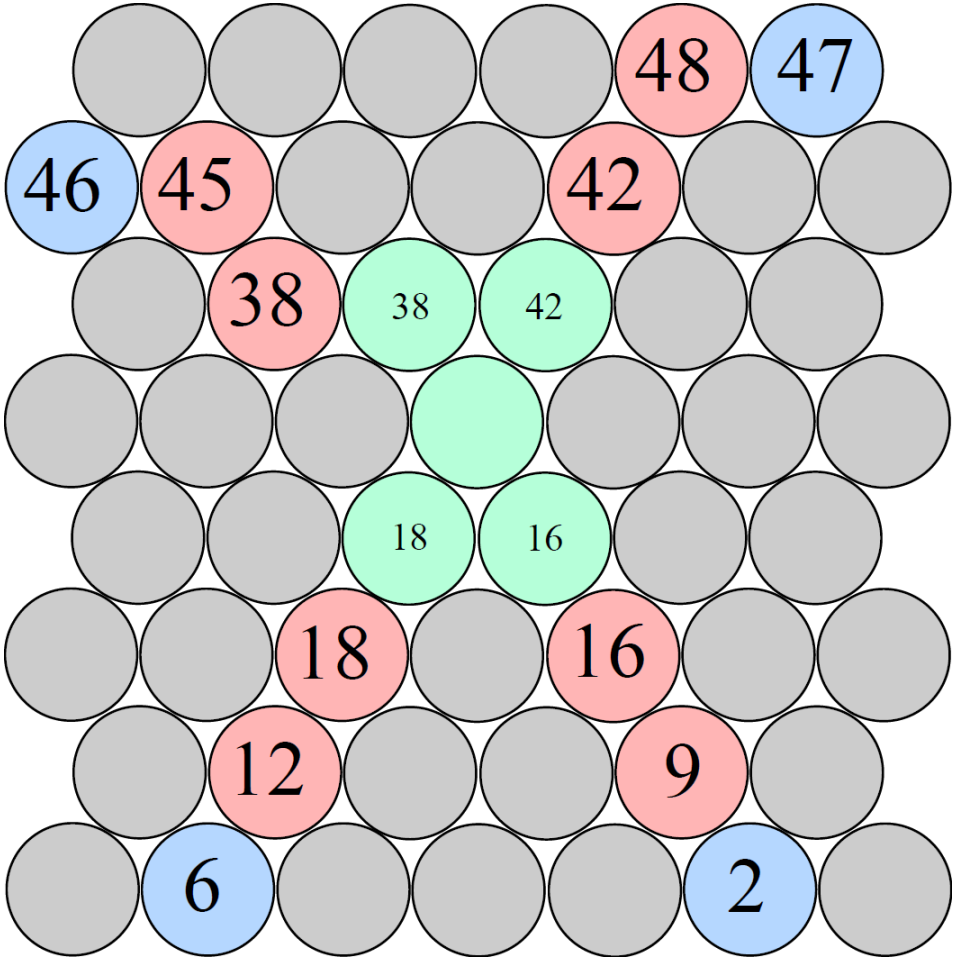
GPU-Version
Cellular automaton – Ambiguous hits



Cell: Straw with hit and more than two neighbors

Rule: Copy the **tubeID** of all your neighbors into your status

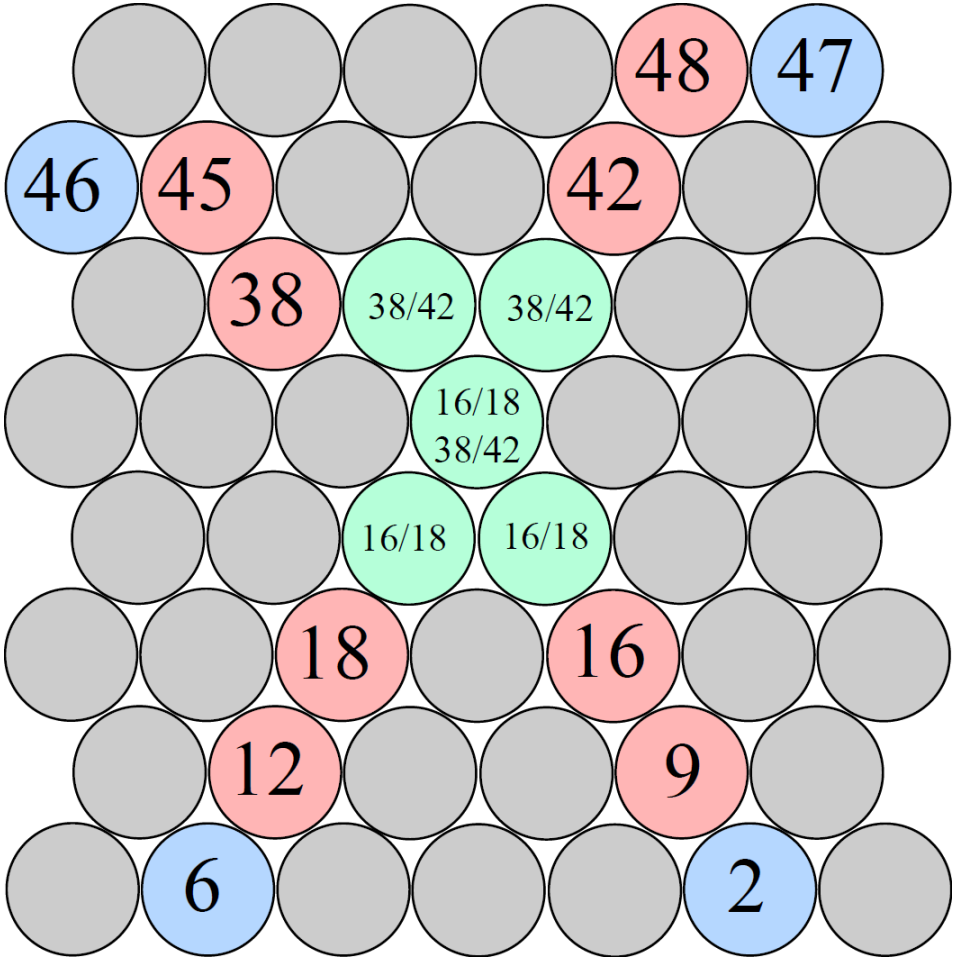
GPU-Version
Cellular automaton – Ambiguous hits



Cell: Straw with hit and more than two neighbors

Rule: Copy the **tubeID** of all your neighbors into your status

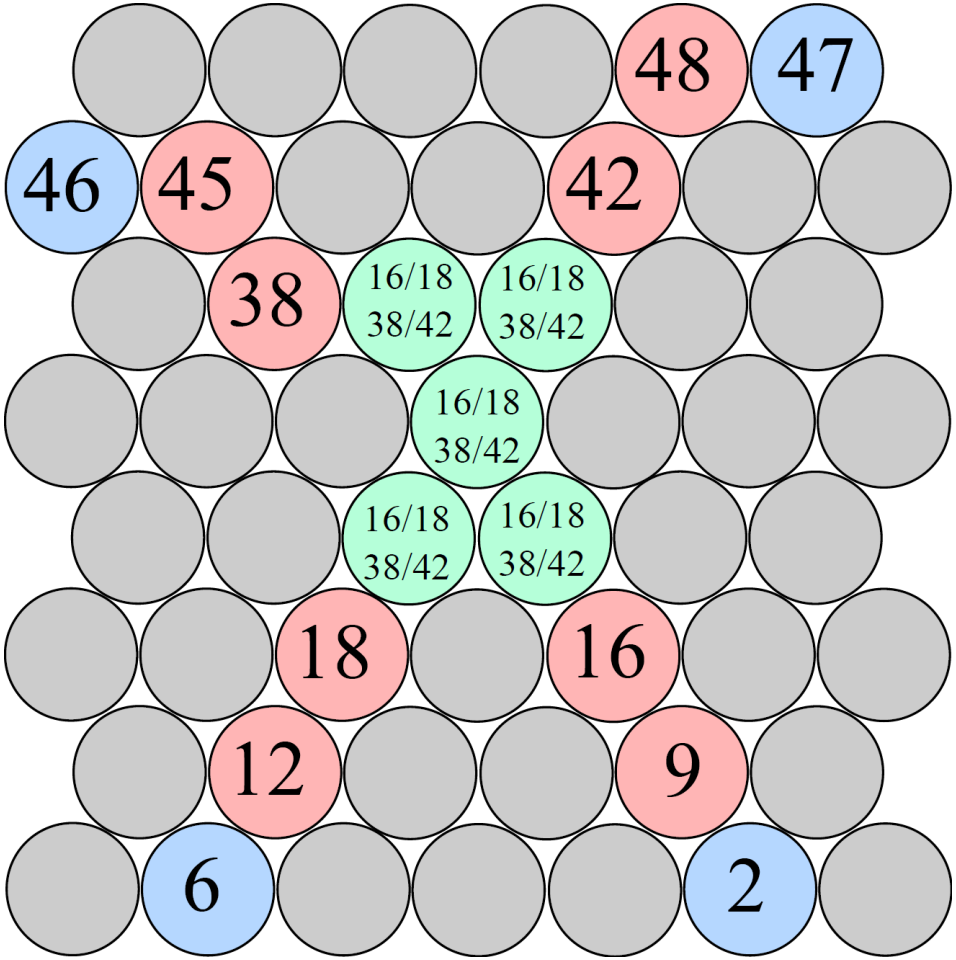
GPU-Version
Cellular automaton – Ambiguous hits



Cell: Straw with hit and more than two neighbors

Rule: Copy the **tubeID** of all your neighbors into your status

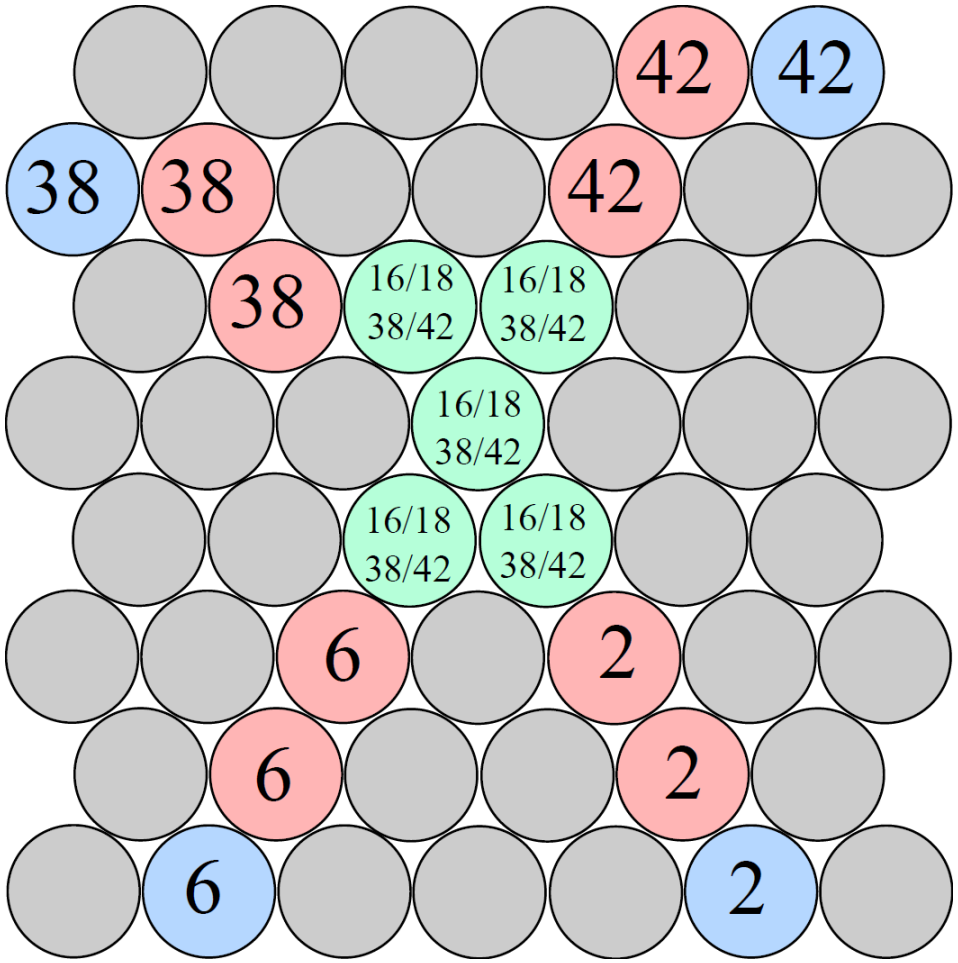
GPU-Version
Cellular automaton – Ambiguous hits



Cell: Straw with hit and more than two neighbors

Rule: Copy the **tubeID** of all your neighbors into your status

GPU-Version
Cellular automaton - Result



Final result of the parallel execution of EvaluateState() and EvaluateMultiState()

38/42: states of outer tracklets

16/18: tubeIDs of ending tubes of inner tracklets

GPU-Version
Cellular automaton

Number of multi-states

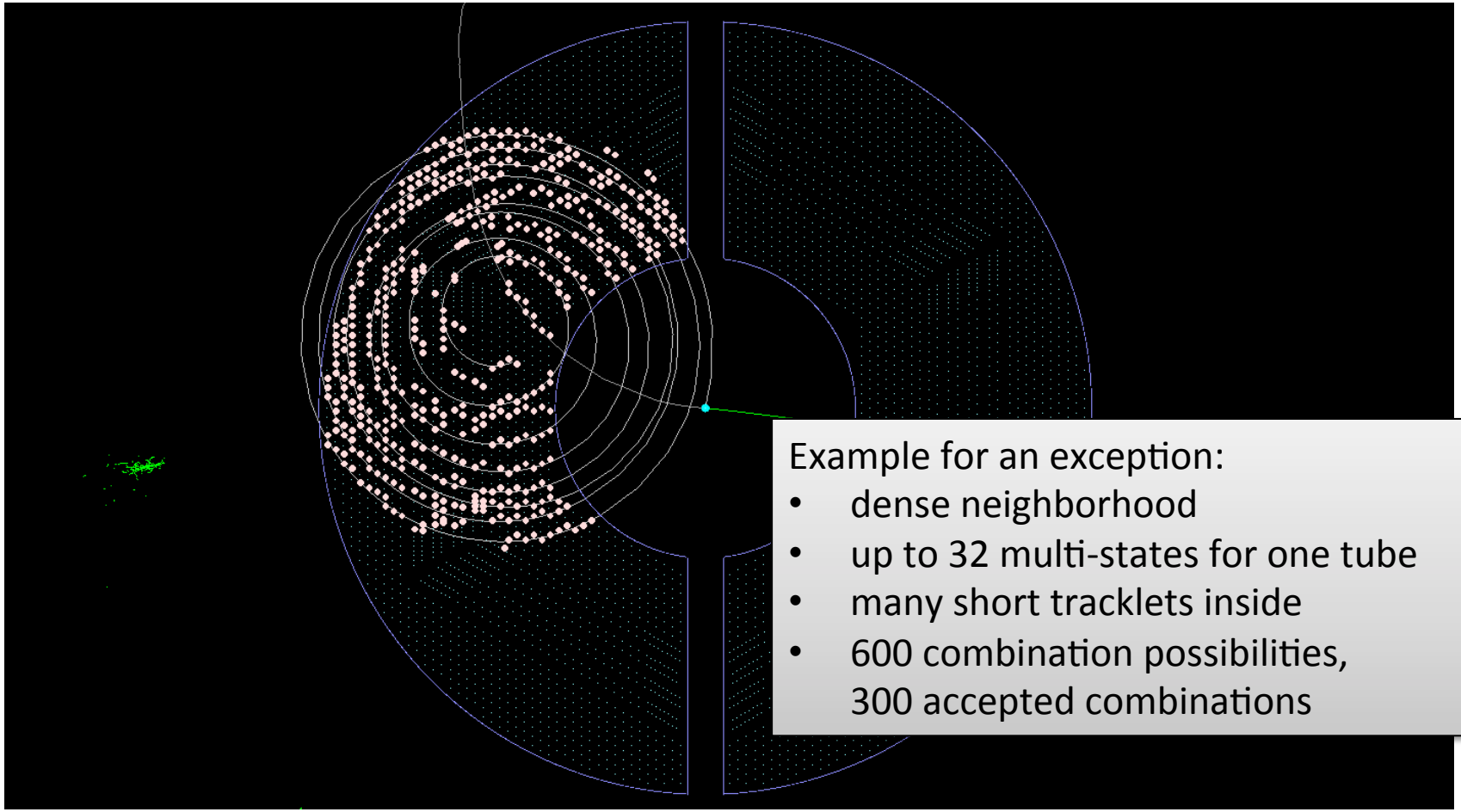
- unknown, variable for each event
- more memory results in longer execution time (looping over the data)

0≤MAX#≤5	5<MAX#≤10	10<MAX#≤20	20<MAX#
≈74%	≈20%	≈5%	≈1%

(first estimate, based on 1000 events)

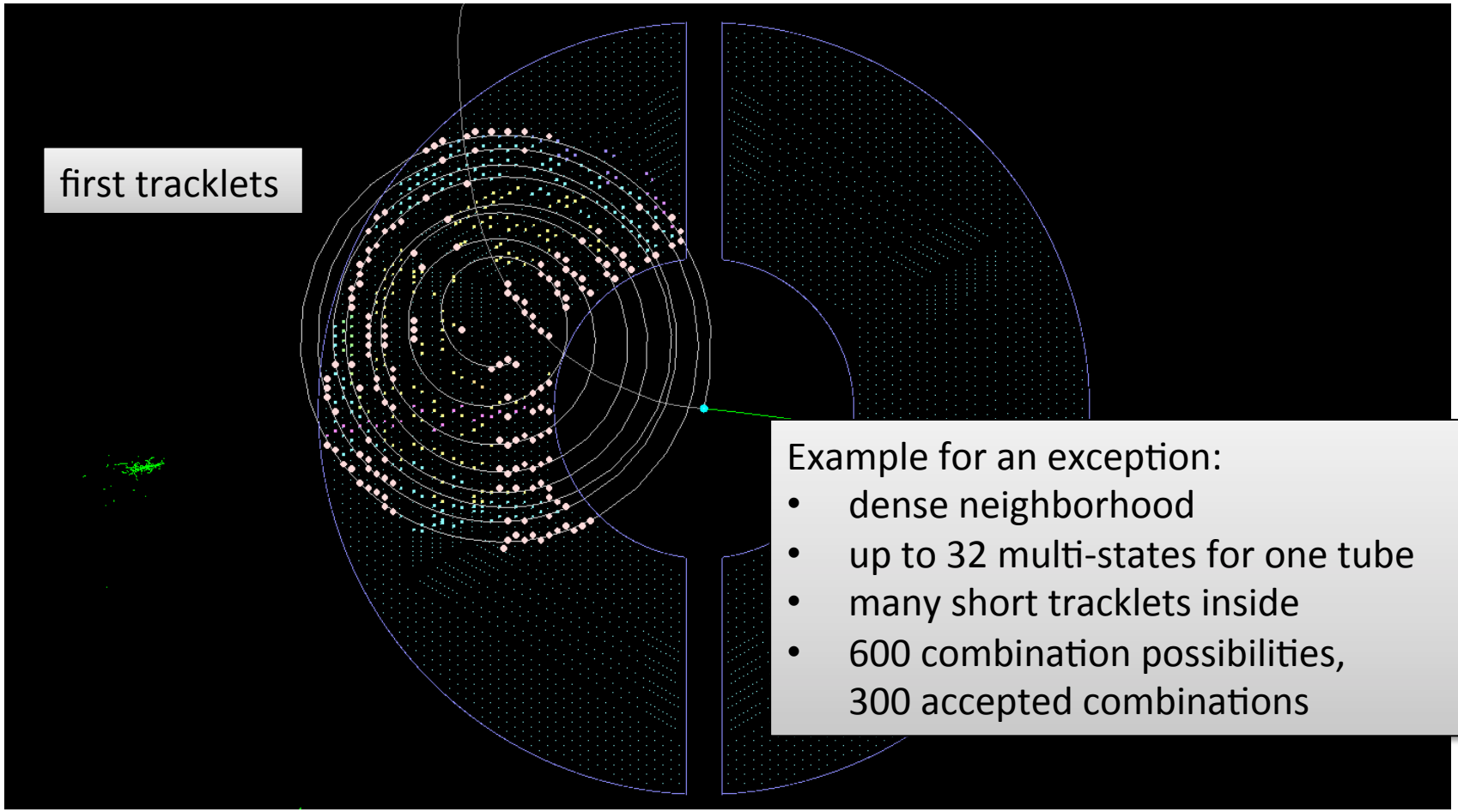
→ set max to 20
→ stop evaluation for exceptions

GPU-Version
Cellular automaton



Example for an exception:
- dense neighborhood
- up to 32 multi-states for one tube
- many short tracklets inside
- 600 combination possibilities, 300 accepted combinations

GPU-Version
Cellular automaton



Example for an exception:
- dense neighborhood
- up to 32 multi-states for one tube
- many short tracklets inside
- 600 combination possibilities, 300 accepted combinations

Outlook

- parallelism at event-level!
- parallelize other parts of the trackfinder
- optimization: memory management + access, loops, …

Thank you very much for your attention!