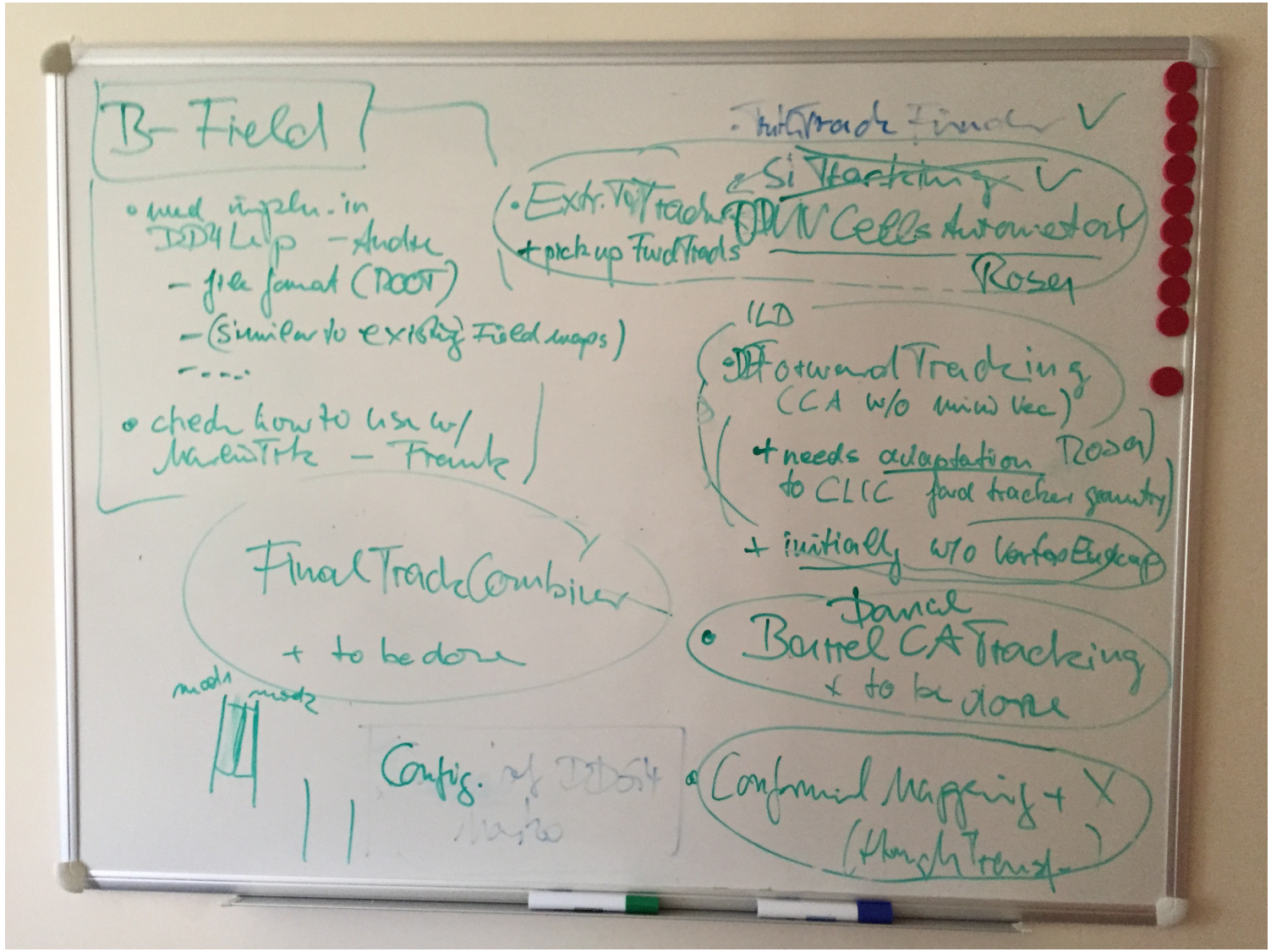
Pattern recognition for the CLIC detector

Daniel Hynds, Rosa Simoniello
with the support of many others!

CLIC tracking code

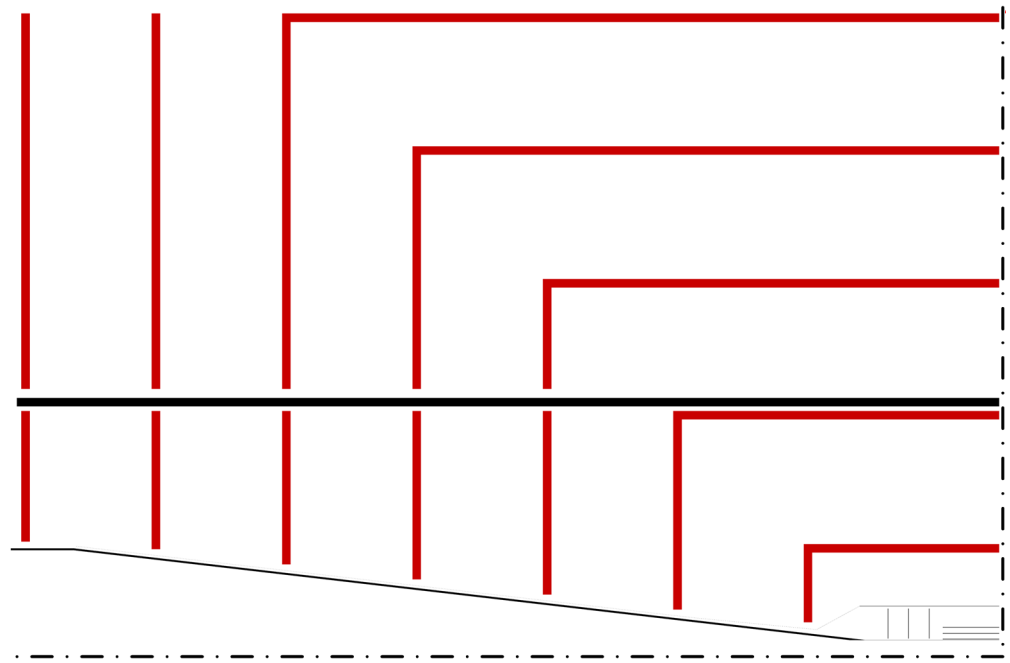
- Brief history since the CDR
  - Decision to move to DD4HEP for geometry description => revamp of the core tracking software required
  - Starting point was to adapt/develop the ILD tracking code:
    - In the barrel region adopt the ILD vertex tracking + custom extrapolation to CLIC tracker
    - In the forward region modify the ILD forward tracking to match the CLIC geometry
    - Combine tracks together at the end
  - Discussion with F. Gaede ~6 months ago to determine how to proceed
    - Continue with ILD algorithms to reach a stage where studies (physics and/or detector) can take place with full reconstruction, inc. background overlay etc.
    - Start developing an alternative tracking, using some more 'exotic' recontruction approach

The master plan



The CLIC tracking system

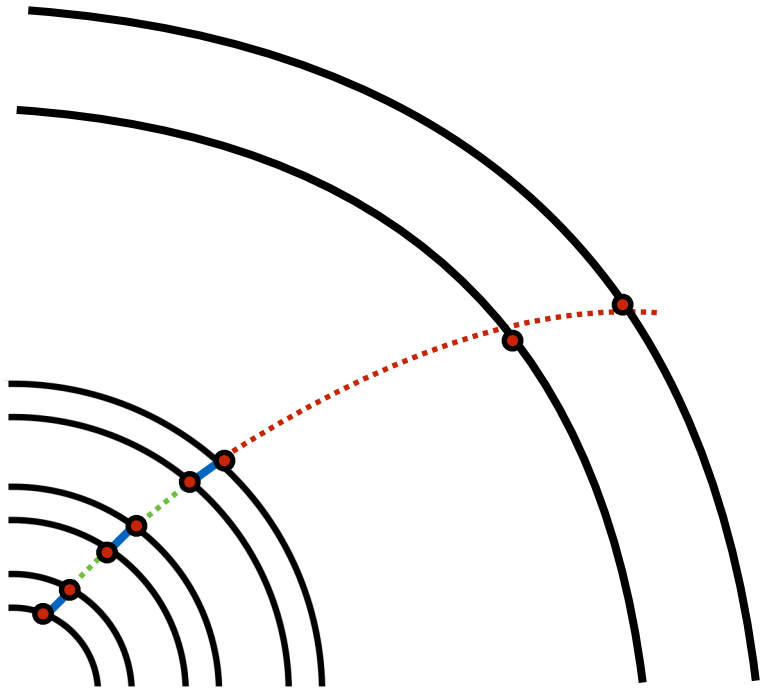
- CLIC tracking system contains a pixellated vertex detector with 3 µm single hit resolution, and a (mostly) large pixel/small strip silicon tracker with 7 µm resolution in the rϕ direction
  - Vertex detector built of double-sided modules: both barrel and endcaps contain 3 layers
  - Inner and outer tracking system separated by mechanical support shell, all single-sided modules with 5 barrel and 7 endcap layers



see talk by M. Petrič

Tracking à la ILD - the barrel

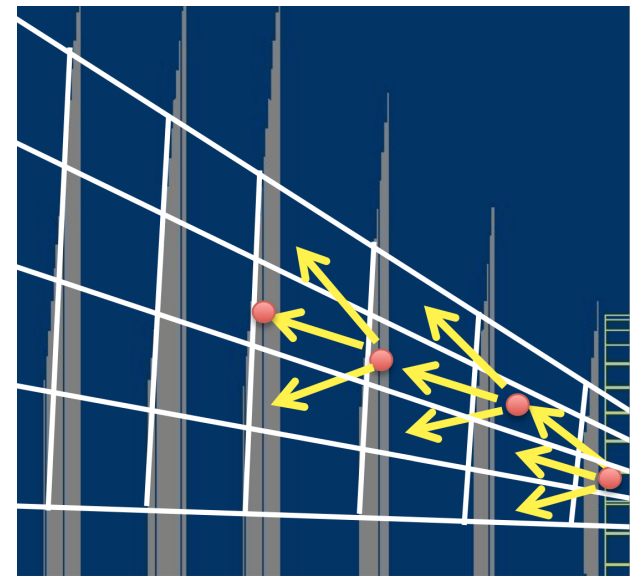
- For tracks passing through the barrel region of the vertex detector, reconstruction is a combination of mini-vector generation, cellular automaton and track extrapolation
  - Mini vectors are created connecting two hits in neighbouring vertex barrel layers (taking advantage of the double-layer modules)
  - Cellular automaton is used to produce tracks from these mini vectors
  - These tracks are extrapolated into the tracker barrels and endcaps, with additional hits added to the track



Tracking à la ILD - the forward direction

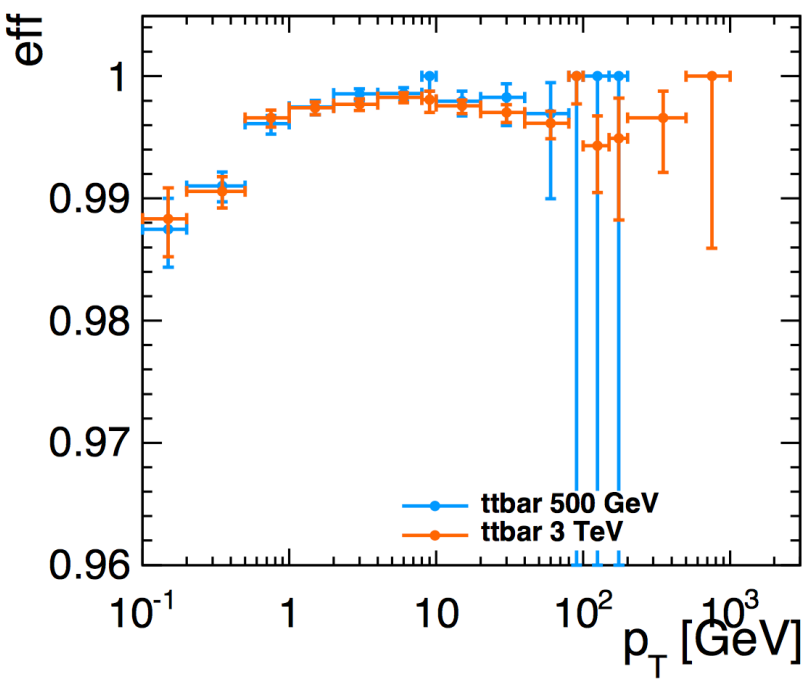
- For tracks passing through the vertex endcap, reconstruction is performed using a pure cellular automaton
  - Based on the ILD forward tracking code
  - All hits from vertex + tracker endcaps
  - Geometry-determined sectors defined in order to reduce combinatorics, take region of $\Delta\phi$, $\Delta\theta$
    - Look for connections between neighbouring sectors
    - Take care of number of layers, neighbour search distance etc.

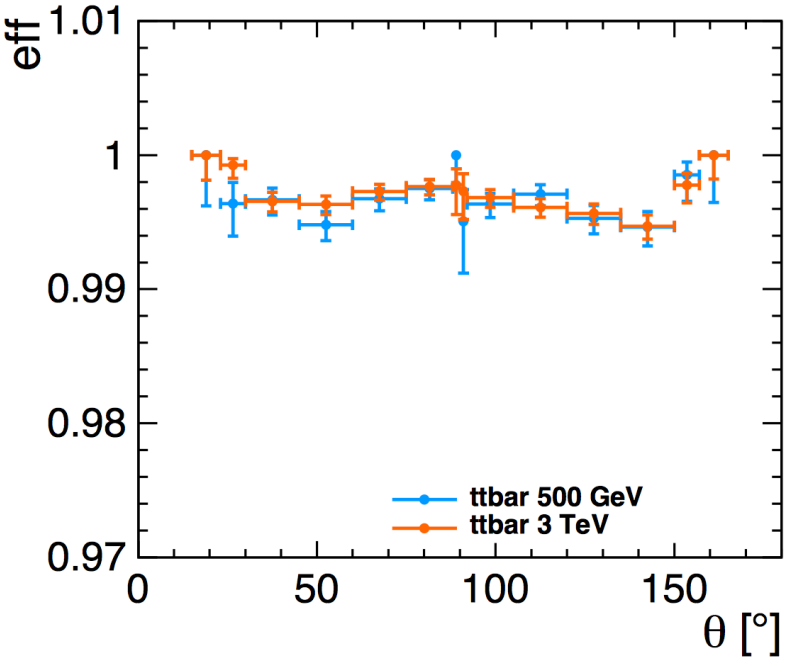
- Searches then made for cloned tracks
  - Track can be reconstructed both in the barrel and as a forward track - check for common hits
  - Currently extending coverage and hits considered to ensure no tracks lost in the transition region (barrel to forward)



Tracking performance

- High tracking efficiency obtained with this approach
  - Results shown for ttbar events (500 GeV and 3 TeV) with no background overlay
  - Particles considered reconstructable if they:
    - Produce more than 4 hits in the detector
    - Are prompt (created within 10 cm of IP)
    - Have $8° < \theta < 172°$ (left) and $p_T > 100$ MeV (right)





Now for something completely different…

- Conformal mapping (generally speaking) is a co-ordinate transform which preserves local angles, but not necessarily curvature or size. For HEP there is a particularly useful mapping, that of the *xy* projection onto the *uv* plane

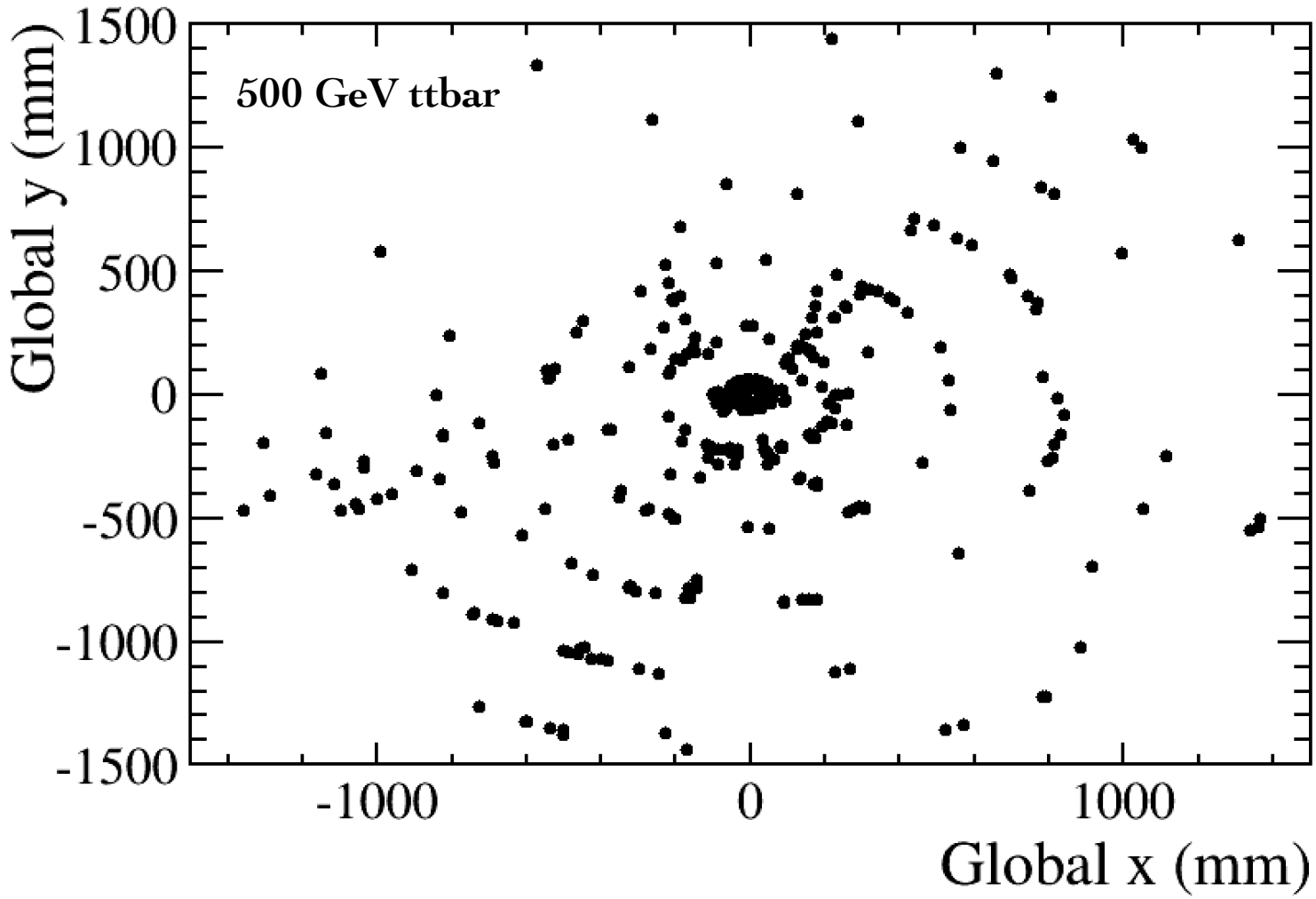
$$u = x / (x^2+y^2)$$
$$v = y / (x^2+y^2)$$

  which has been proposed for tracking algorithms. This has the feature of mapping circles in *xy* space onto straight lines in *uv*

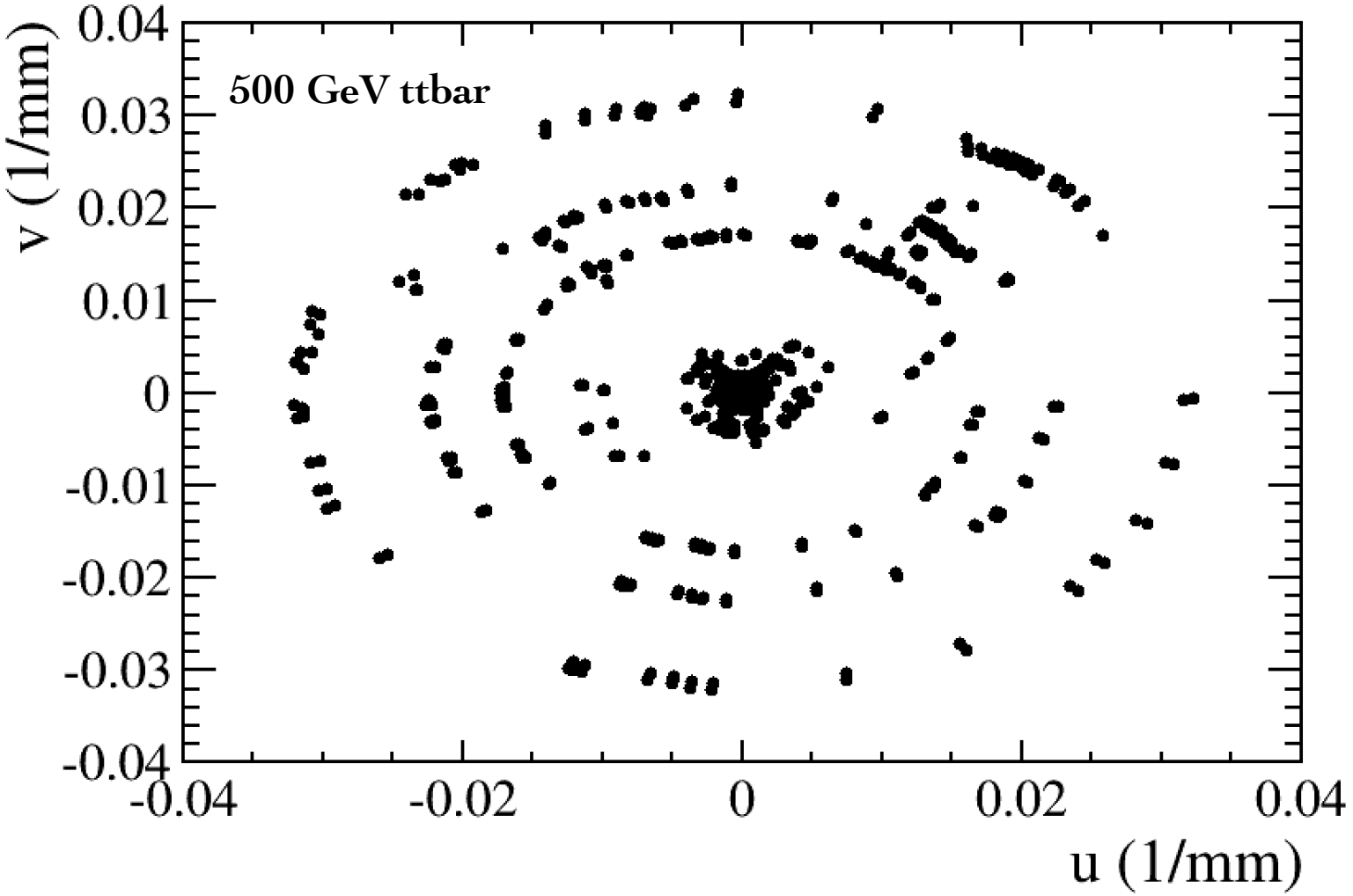
- A conformal map of the detector *xy* plane thus reduces the pattern recognition to a 2D straight line search, with high p_T tracks pointing towards the origin

- Some examples in literature (use in ALICE), but typically after the mapping is applied the track finding is either simple track following or looking for groups of hits with the same azimuthal co-ordinates

A view in the *xy* detector plane

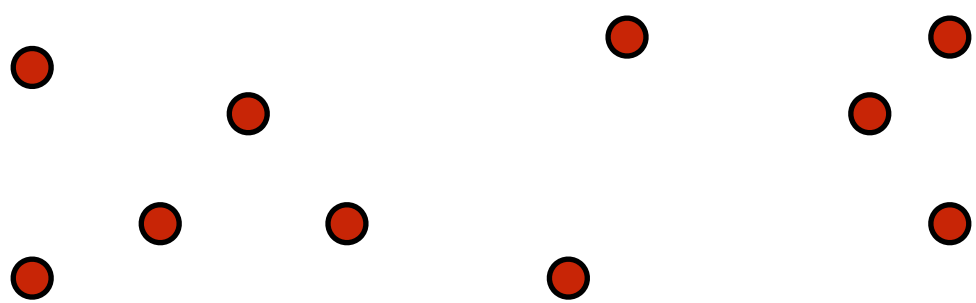


A view in the *uv* detector plane



Cellular automaton in practice

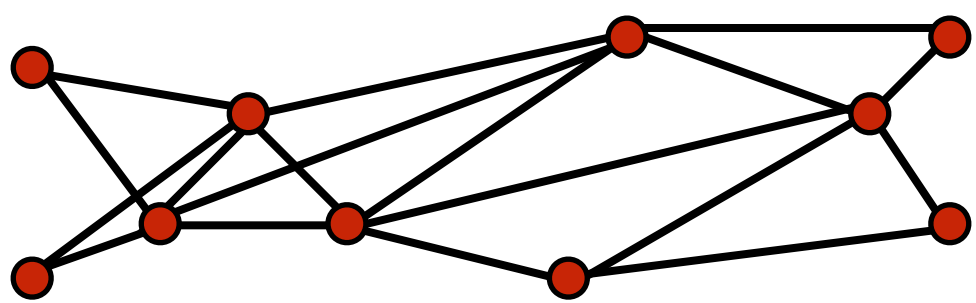
- CA can be implemented in many ways, but all feature the same basic blocks
  - A *cell* is a line connecting two points
  - Each cell contains a *weight*. Starting from some seeding point, all succeeding cells which pass the given criteria are incremented



Cellular automaton in practice

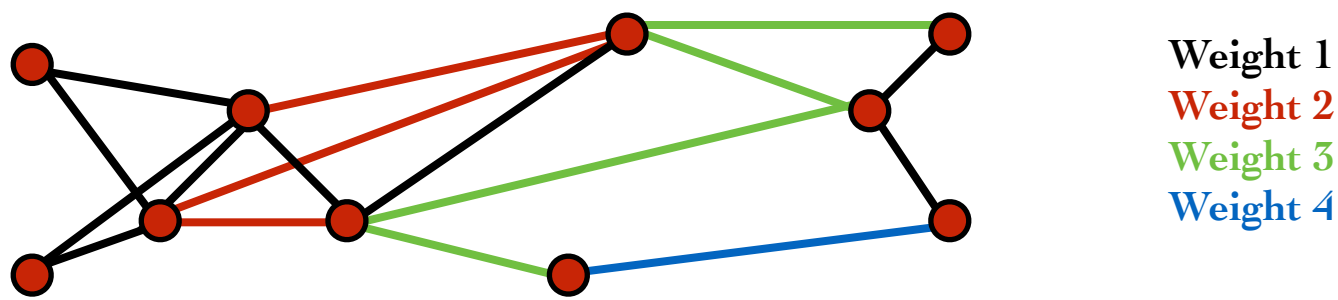
- CA can be implemented in many ways, but all feature the same basic blocks
  - A *cell* is a line connecting two points
  - Each cell contains a *weight*. Starting from some seeding point, all succeeding cells which pass the given criteria are incremented

- In principle each point may be connected to any other, but combinatorics explode => criteria added (such as only to hits nearby in next detector layer)



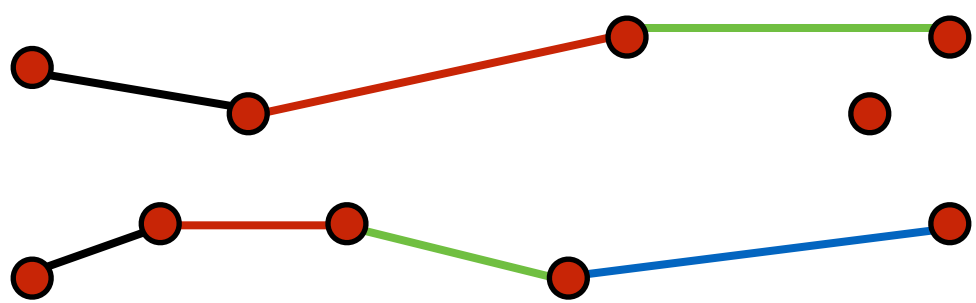
Cellular automaton in practice

- CA can be implemented in many ways, but all feature the same basic blocks
  - A *cell* is a line connecting two points
  - Each cell contains a *weight*. Starting from some seeding point, all succeeding cells which pass the given criteria are incremented
- In principle each point may be connected to any other, but combinatorics explode => criteria added (such as only to hits nearby in next detector layer)
- Weight increased for cells which have an angle < θ to the previous cell



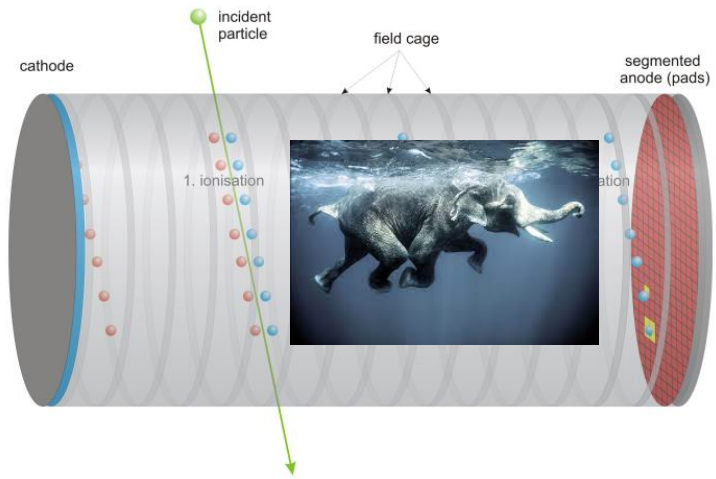
Cellular automaton in practice

- CA can be implemented in many ways, but all feature the same basic blocks
  - A *cell* is a line connecting two points
  - Each cell contains a *weight*. Starting from some seeding point, all succeeding cells which pass the given criteria are incremented
- In principle each point may be connected to any other, but combinatorics explode => criteria added (such as only to hits nearby in next detector layer)
- Weight increased for cells which have an angle < θ to the previous cell
- Create valid tracks (typically start with highest weight)



A few comments on implementation

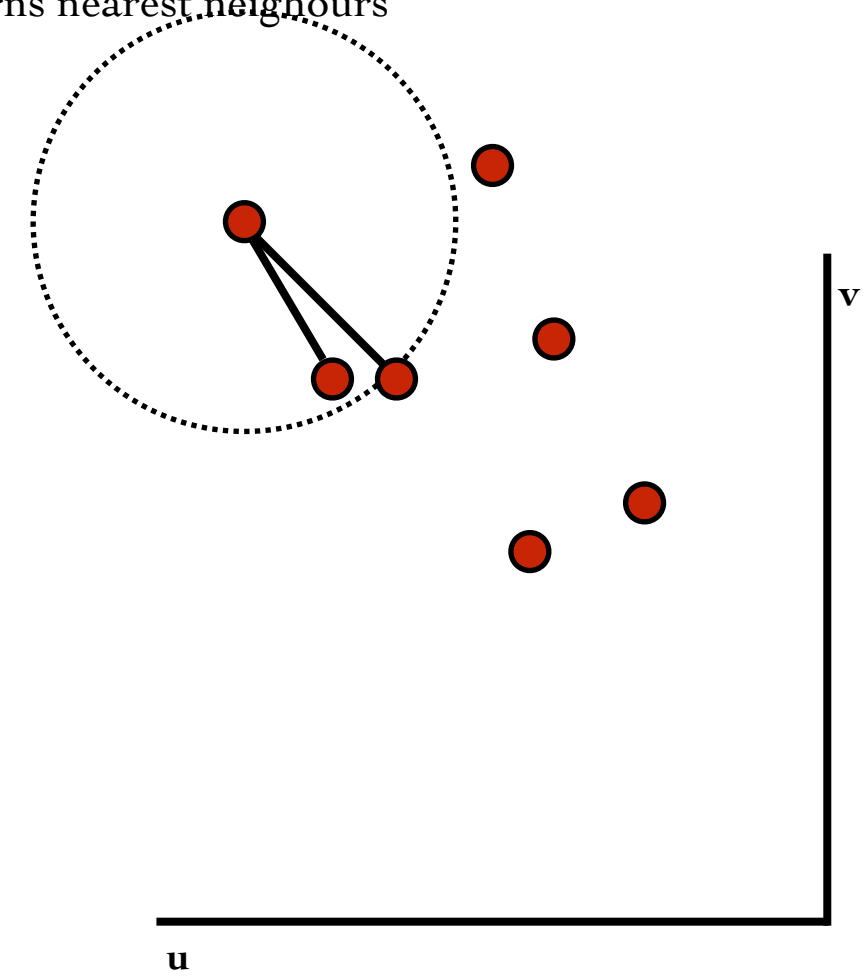
- Many lessons can be learned from existing tracking code
  - Ignoring the elephant in the ~~TPC~~ room
  - Something always goes wrong!
  - Hits not present in one layer => code has to extrapolate to other detector layers, leads to hard-coding of geometry information, inelegant handling…
  - Tracks can be lost in transition from barrel to endcap (or more generally between different tracking detectors) => need to consider hits together…



- In the end, all geometry information was removed from the tracking
  - All neighbour searches occur on 2D space points (using z-information as additional cut)
  - No detector specifications are made - no knowledge of sub-detector, layout, etc.
  - Only concession is to not make cells between hits on the same detection surface
  - All hits considered together

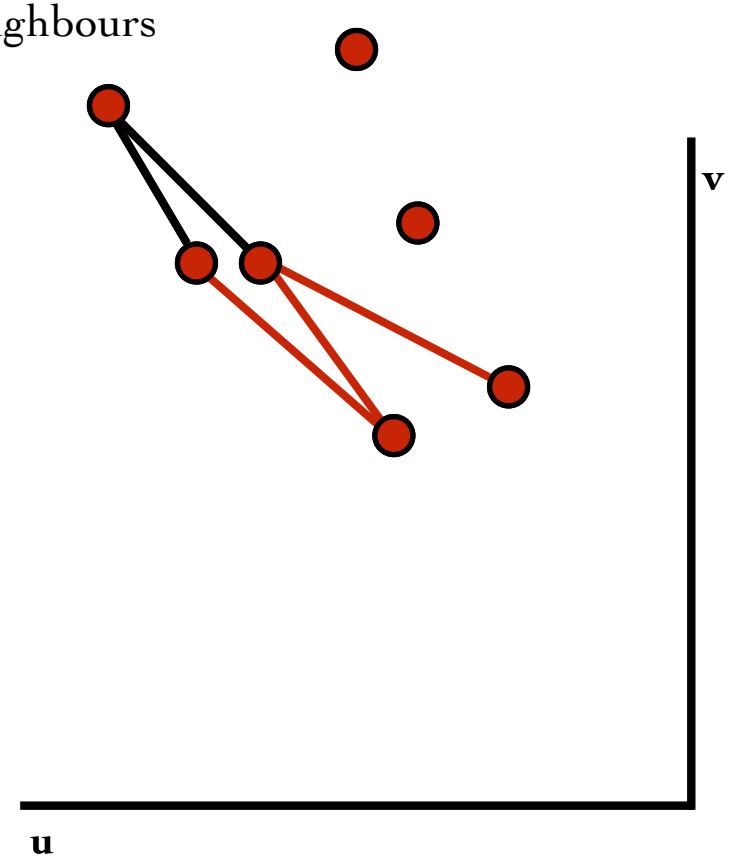
Conformally mapped cellular automaton

- Take a seed hit and build sensible starting cells (high p_T point towards the origin)
  - Binary search tree (KDTree) used for speed, returns nearest neighbours



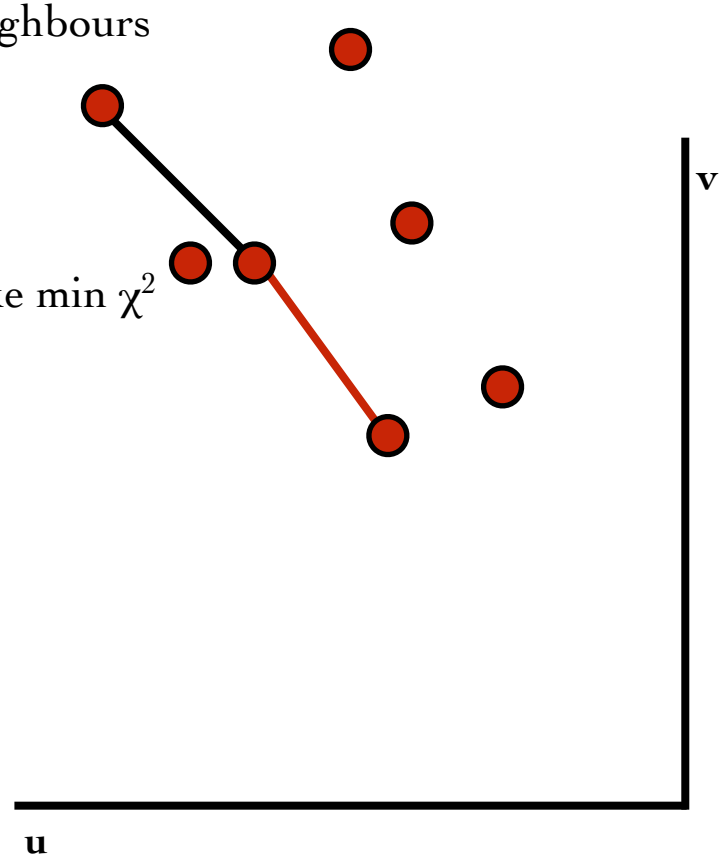
Conformally mapped cellular automaton

- Take a seed hit and build sensible starting cells (high p_T point towards the origin)
  - Binary search tree (KDTree) used for speed, returns nearest neighours
- For each cell, extrapolate along cell direction and look for neighbours
  - Form new cells
  - Throw away cells with angles > α



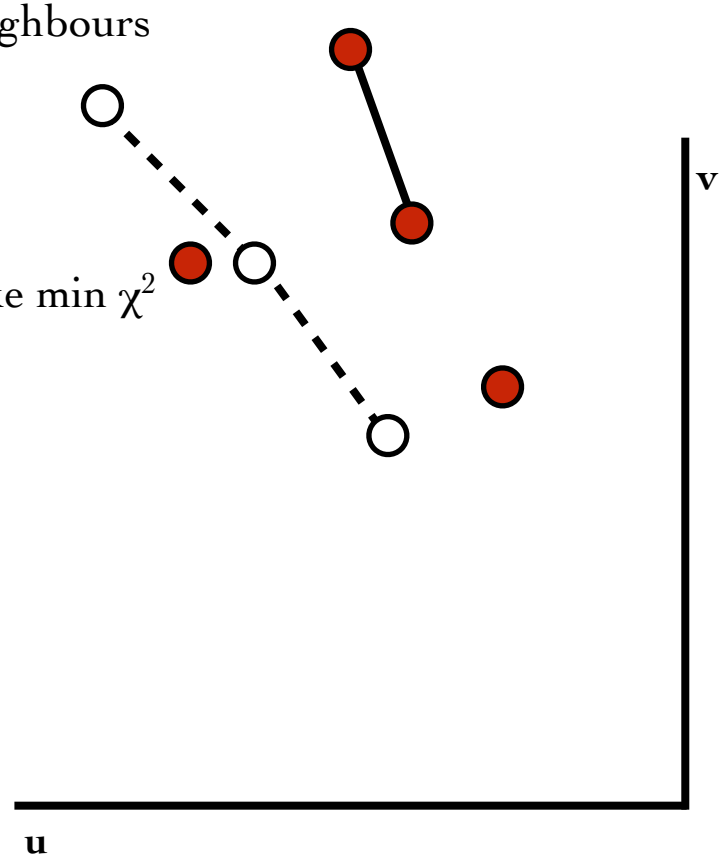
Conformally mapped cellular automaton

- Take a seed hit and build sensible starting cells (high p_T point towards the origin)
  - Binary search tree (KDTree) used for speed, returns nearest neighours
- For each cell, extrapolate along cell direction and look for neighbours
  - Form new cells
  - Throw away cells with angles > α
- Where several track candidates possible, use linear fit and take min χ^2



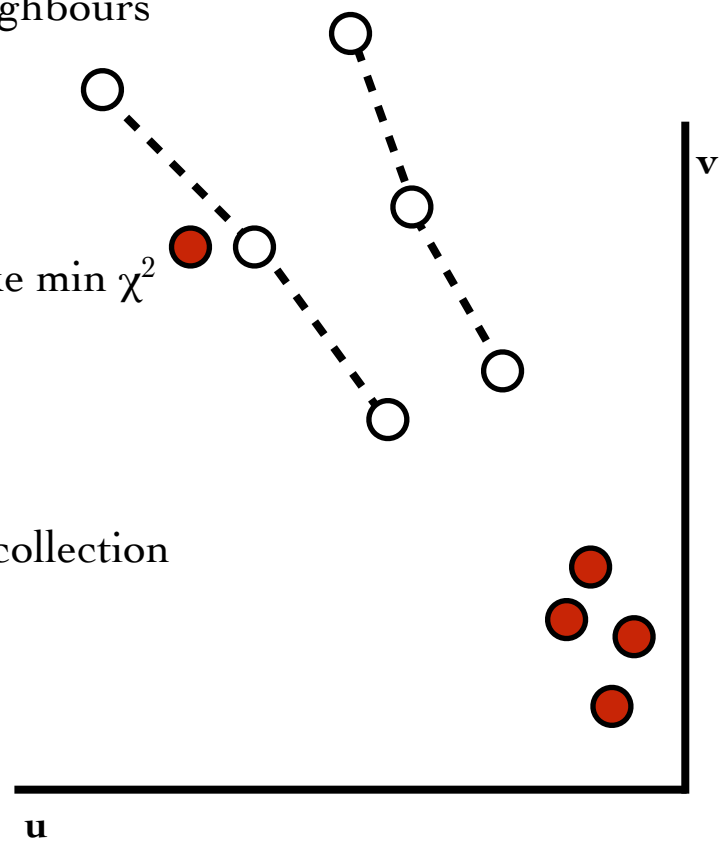
Conformally mapped cellular automaton

- Take a seed hit and build sensible starting cells (high p_T point towards the origin)
  - Binary search tree (KDTree) used for speed, returns nearest neighours
- For each cell, extrapolate along cell direction and look for neighbours
  - Form new cells
  - Throw away cells with angles > α
- Where several track candidates possible, use linear fit and take min χ^2
- Exlude used hits from collection and continue



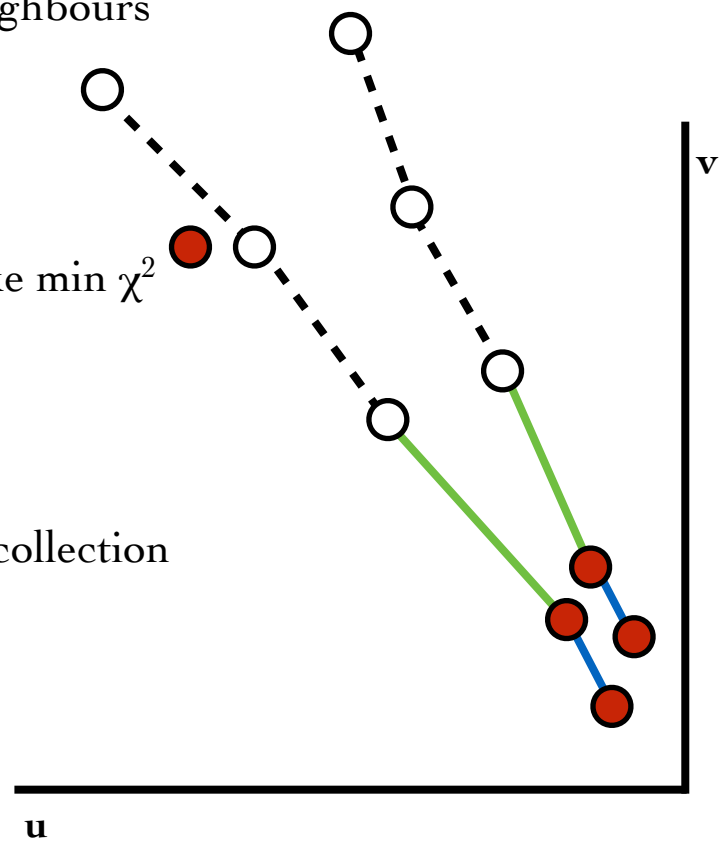
Conformally mapped cellular automaton

- Take a seed hit and build sensible starting cells (high p_T point towards the origin)
  - Binary search tree (KDTree) used for speed, returns nearest neighours
- For each cell, extrapolate along cell direction and look for neighbours
  - Form new cells
  - Throw away cells with angles > α
- Where several track candidates possible, use linear fit and take min χ^2
- Exlude used hits from collection and continue
- Once all hits have been considered as seeds, add the next hit collection

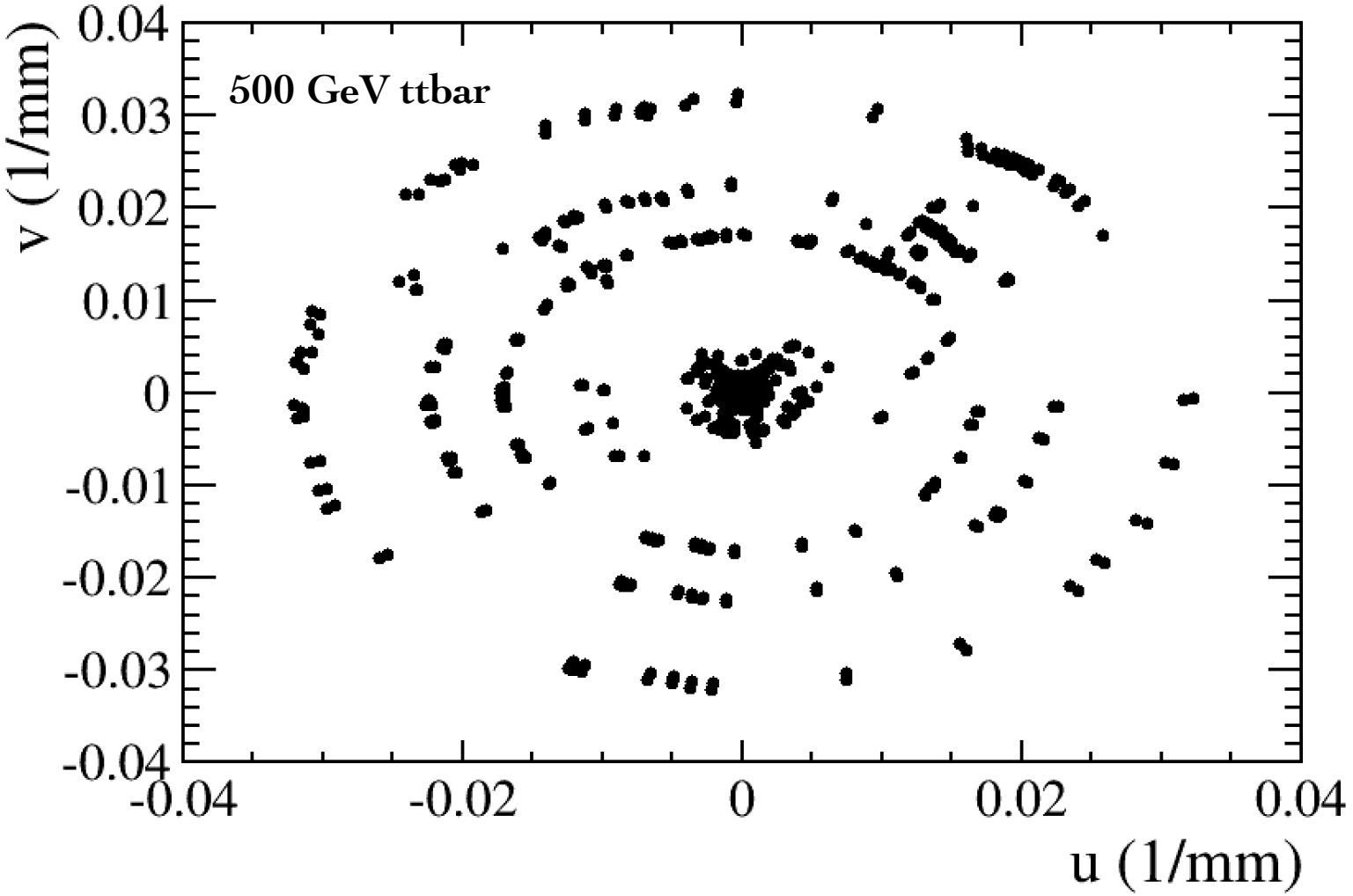


Conformally mapped cellular automaton

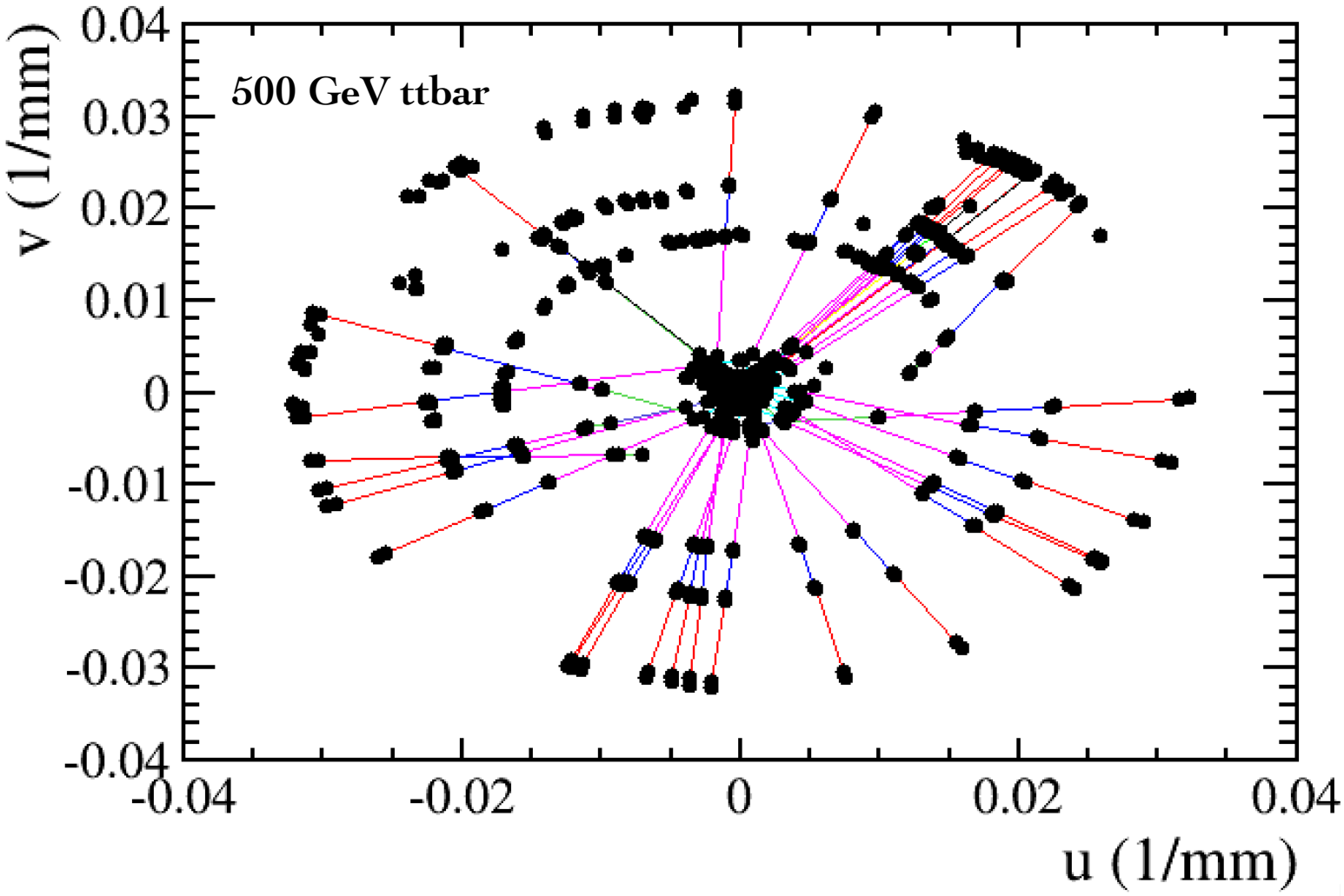
- Take a seed hit and build sensible starting cells (high p_T point towards the origin)
  - Binary search tree (KDTree) used for speed, returns nearest neighours
- For each cell, extrapolate along cell direction and look for neighbours
  - Form new cells
  - Throw away cells with angles > α
- Where several track candidates possible, use linear fit and take min χ^2
- Exlude used hits from collection and continue
- Once all hits have been considered as seeds, add the next hit collection
  - Try to extend existing tracks
  - Try to build new tracks from leftover hits



Back to the view in the *uv* detector plane

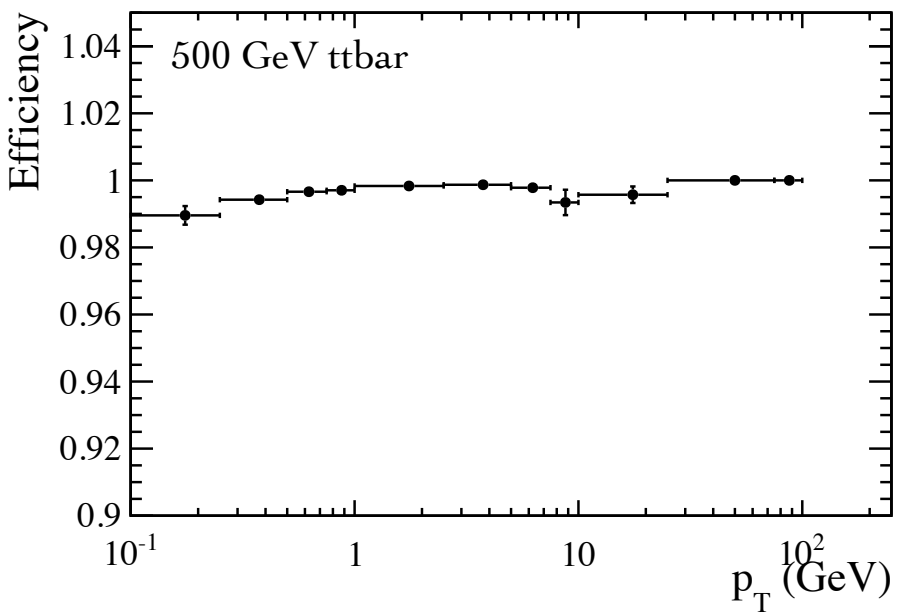


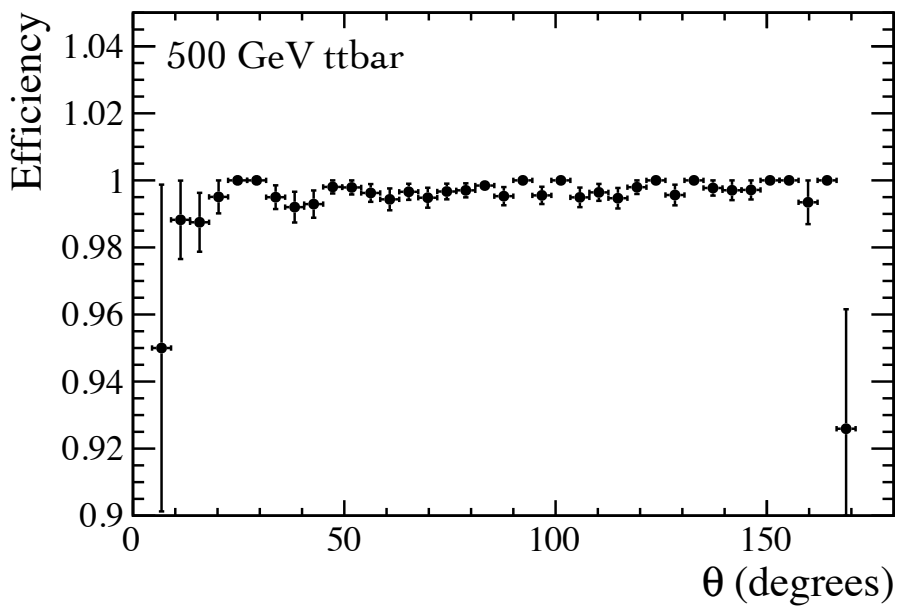
Tracks in the *uv* detector plane



Tracking performance

- High tracking efficiency obtained for conformally mapped CA
  - Many possible further developments - track fit in sz? Psuedo-kalman filter?
  - Approach should be robust to geometry changes, since no knowledge of layers/subdetectors/ layout
  - Advert: intel profiling tools at cern! Significant improvements in speed achieved through profiling (obvious tip: be careful using ROOT…)





Next steps

- The **track fit:**
  - The most critical piece still under development (see talk by R. Simoniello)
  - In addition to enabling further downstream analysis (vertex finding and fitting, etc.) the track fit will be necessary to get a handle on ghosts, improve timing, overall performance of reconstruction, etc.
  - Significant impact on the rest of the pattern recognition strategy
    - For ILD-style tracking, obvious effect in extrapolating vertex tracks to the tracker, will greatly influence missed hit rates etc.
    - For Conformal Mapping CA ghost rate reduced by χ^2 cut currently in conformal space - ghosts overlap with low p_T tracks where no material effects considered
  - Once the fit is correctly working, it must be optimised for timing in order to be more useful in the pattern recognition

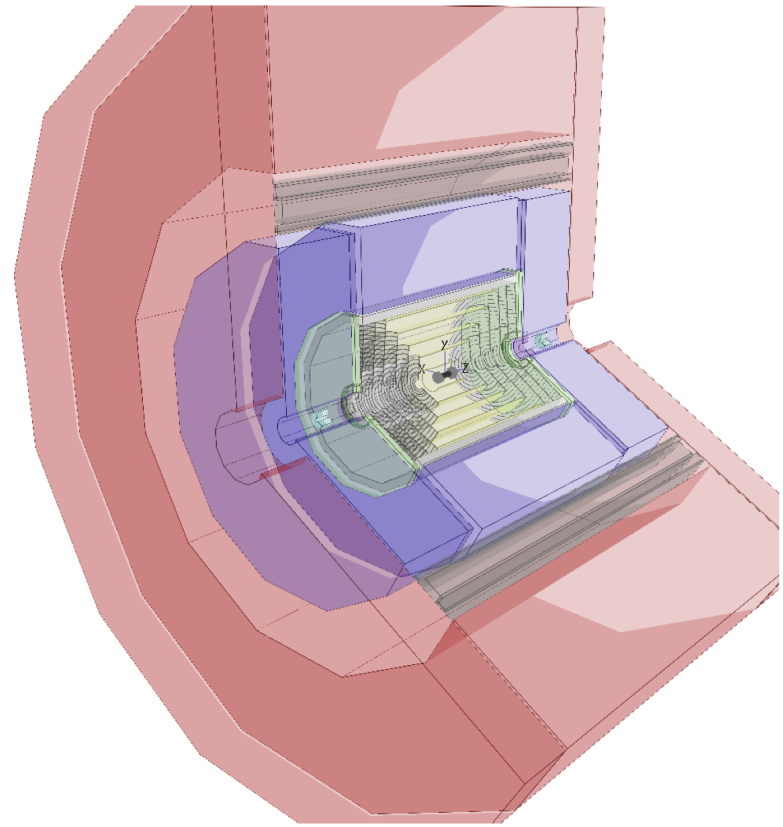
- The digitiser:
  - In parallel with heading towards production-readiness, a realistic description of the silicon must be implemented (where possible and where a preference on detector technology exists)

Summary

- Track finding algorithms for CLIC detector working well and still under development
  - High tracking efficiency over full angular and p_T range
  - Detailed study of ghosts, clones, missing hits, algorithm timing underway

- Next steps:
  - Completion and optimisation of the track fitting
  - Overlay of background events

- Much to be optimistic about!



BACKUP