



Science and
Technology
Facilities Council

Particle Physics

What's new since last workshop?

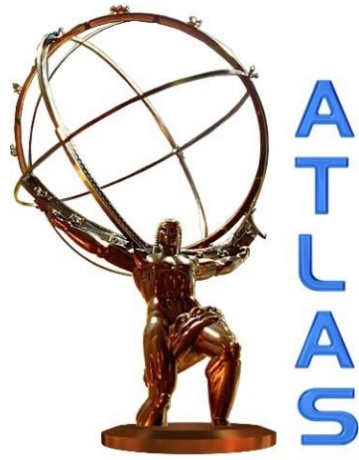
Track Finding

Tim Adye

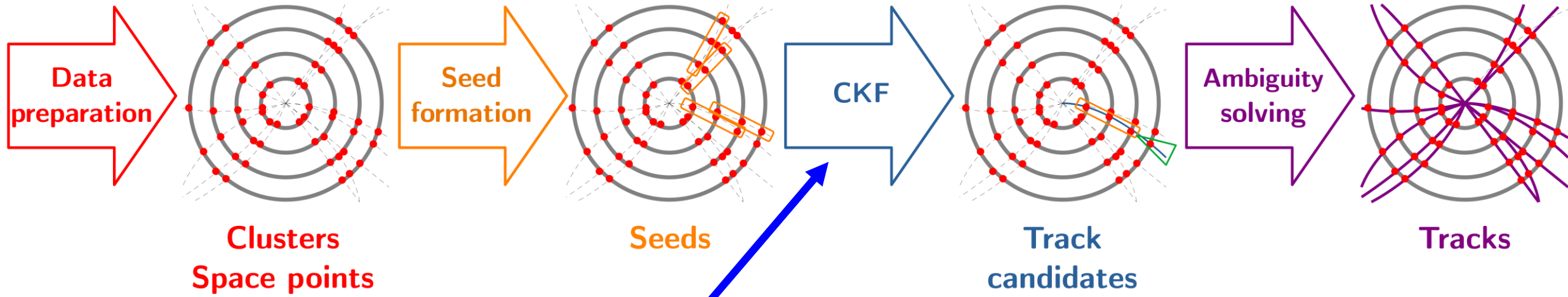
Rutherford Appleton Laboratory

ACTS Developers Workshop

19th November 2024




Track Finding algorithms



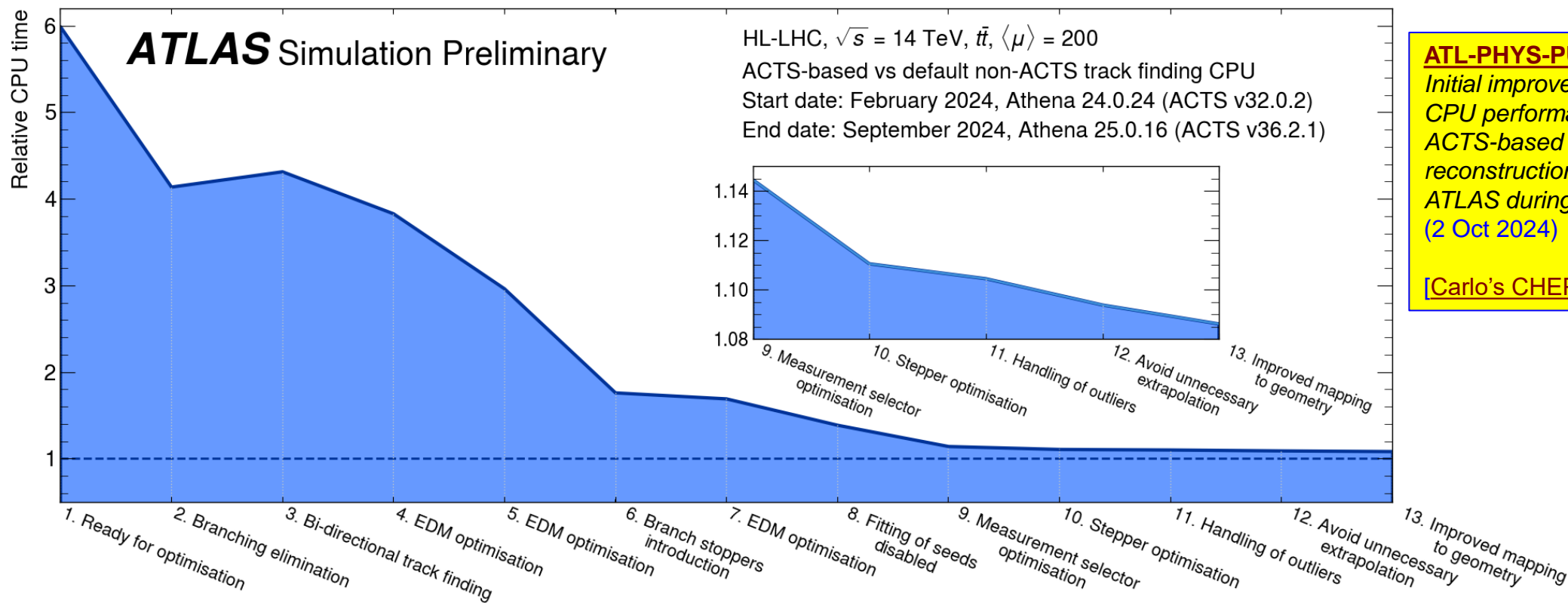
- CKF (Combinatorial Kalman Filter) – optimising
- GNN (Graph Neural Network) – preparing
- NNF (Nearest Neighbour Filter) – anticipating

CKF

CKF

- The **Combinatorial Kalman Filter** is the default ACTS track finding algorithm
 - Discussed by Andreas Stefl at last year's workshop
- This year there has been a concerted campaign of optimisation
 - Targeted at the **ATLAS ITk**, with most challenging speed requirements coming from the online event filter ("EF Tracking")
 - The goal has been to meet or exceed the tracking and CPU performance from the legacy (non-ACTS) offline Athena reconstruction
 - The progress towards this goal are detailed on the **following slides** 
 - One significant difference from the non-ACTS algorithm is that (in many cases) the tracking performance from the ACTS algorithm is good enough **without a final KF fit**. This could buy us up to a factor 2 improvement in CPU.
- We made improvements to both core ACTS and ATLAS Athena steering
 - Here performance measurements are for the ATLAS ITk running in the **Athena** framework
 - Core ACTS tested also with standalone ActsExamples framework, both ATLAS ITk and Open Data Detector
 - **Similar performance improvements seen in both cases**
- **Apologies for the close focus here of ATLAS / Athena applications**
 - Many improvements will have applicability elsewhere
 - I will try to indicate things that really are ATLAS-specific

CKF improvements overview

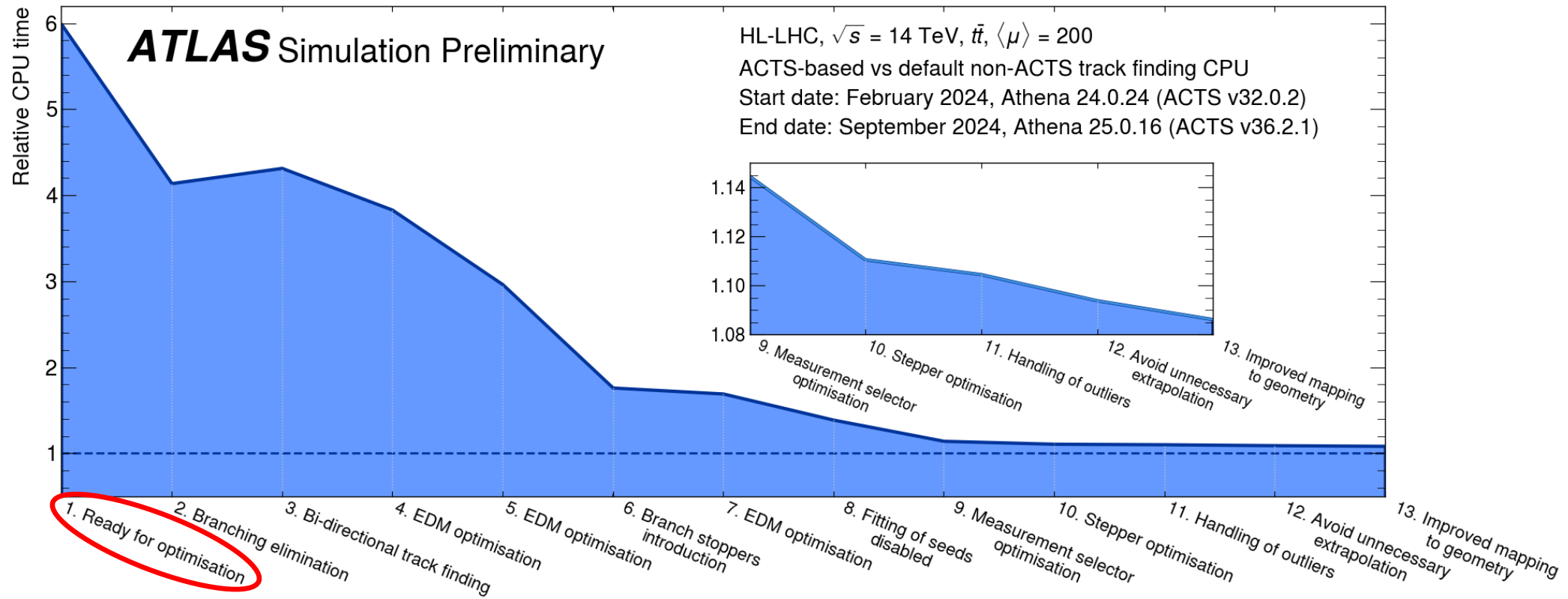


ATL-PHYS-PUB-2024-017
Initial improvements to the CPU performance for the ACTS-based track reconstruction software for ATLAS during the HL-LHC (2 Oct 2024)

[Carlo's CHEP 2024 talk]

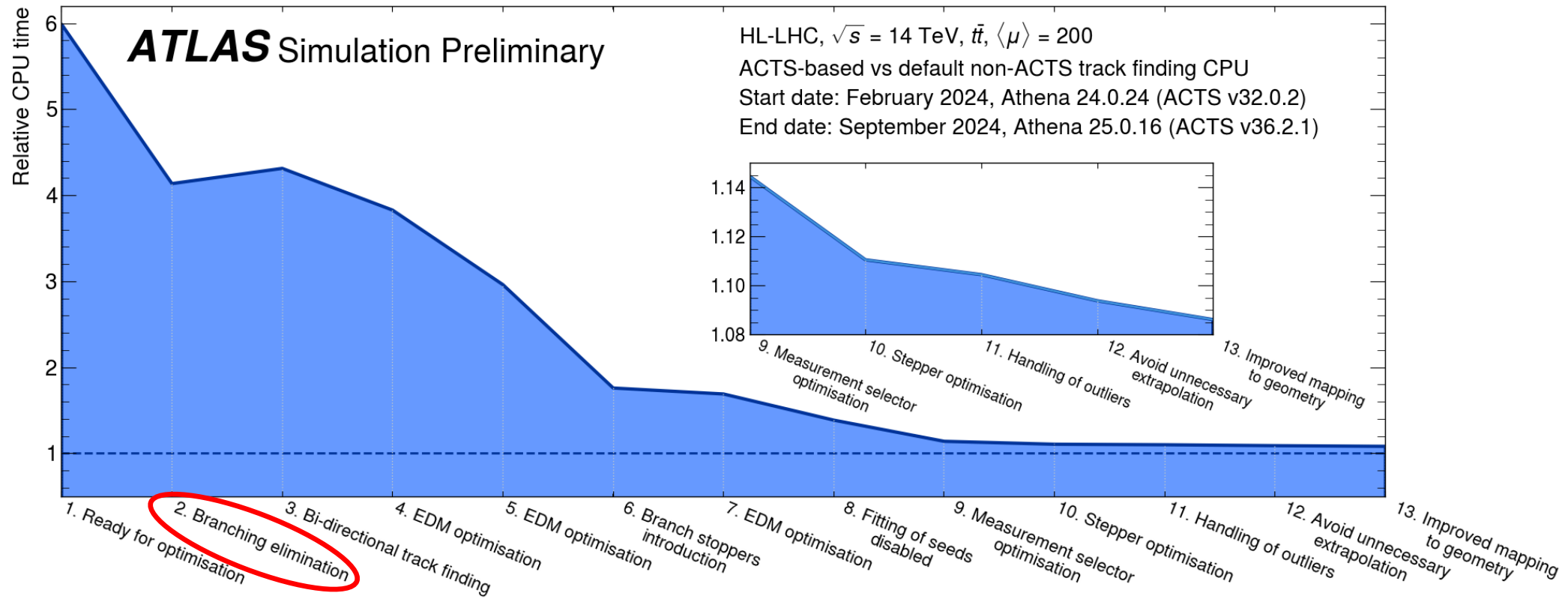
- **Figure 1:** Incremental decrease of the CPU time to reconstruct $t\bar{t}$ events at $\langle\mu\rangle = 200$ when adding improvements to the ACTS-based ITk track reconstruction deployed in the ATLAS software framework Athena.
 - The CPU time is for the track finding alone.
 - This is relative to the legacy non-ACTS algorithm. Seeding is included in the timing, but did not change.
 - This comparison is for the default offline configuration. Similar relative performance obtained with EF “fast tracking”.
 - The relevant improvements are reported in chronological order on the horizontal axis of the plot and are described in the following slides. →

CKF improvements 1



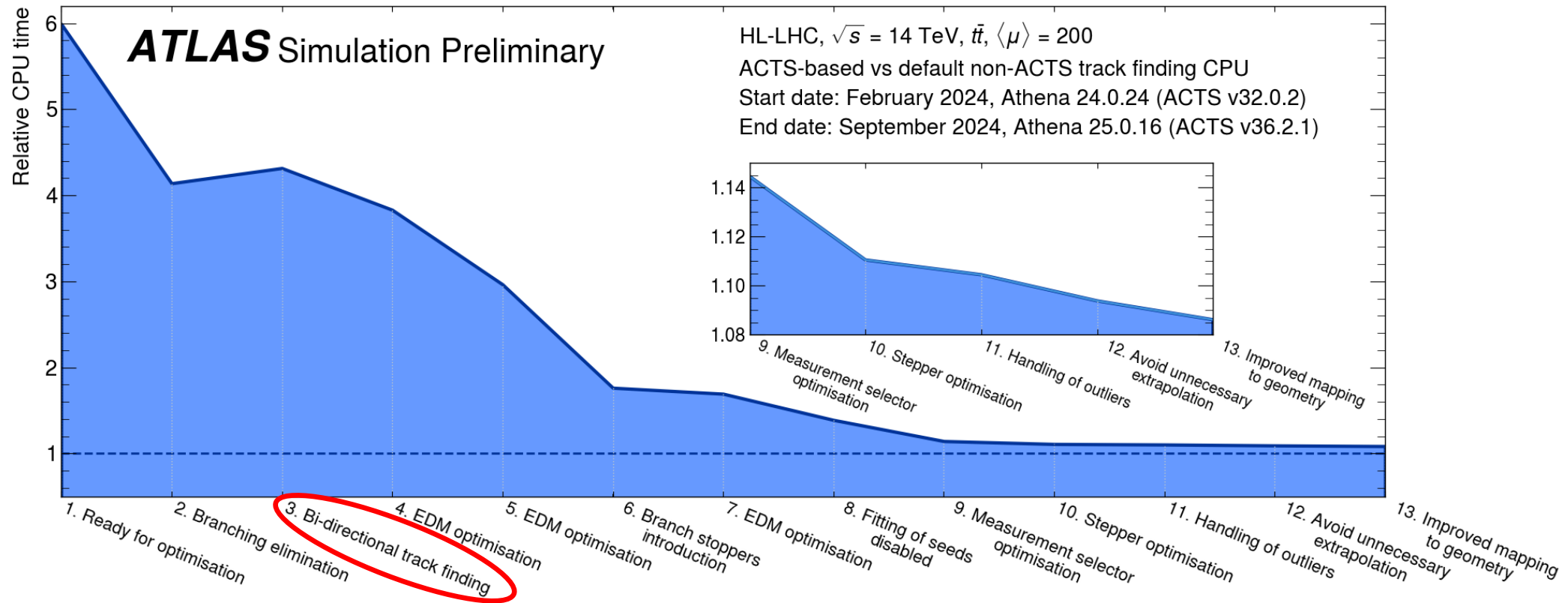
- 1. Ready for optimisation:** the starting point for the optimisation corresponding to the first fully-functional version of the ACTS-based ITk track reconstruction deployed in Athena 24.0.24 (ACTS v32.0.2) on 15 February 2024. This version of the code was seen to be six times slower than the default non-ACTS ITk track finding

CKF improvements 2



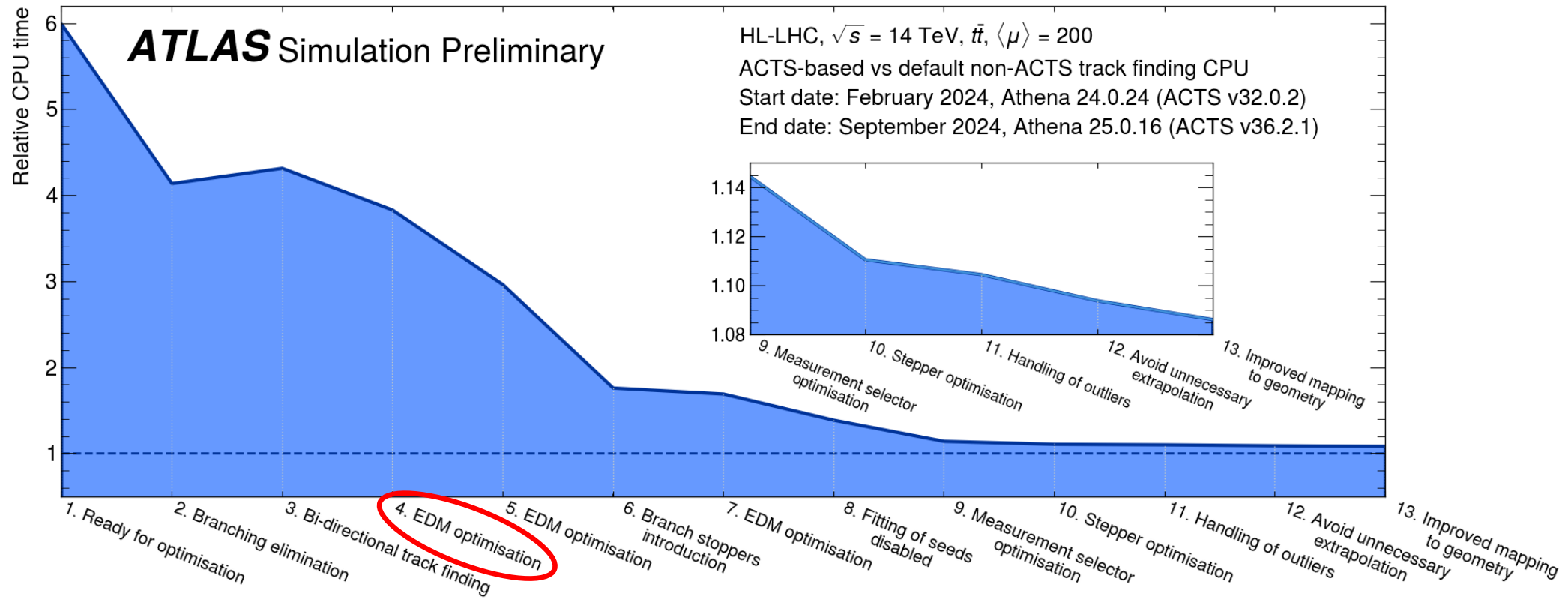
2. Branching elimination: the combinatorial Kalman filter is configured to only consider a single branch, hence providing one track candidate per input seed

CKF improvements 3



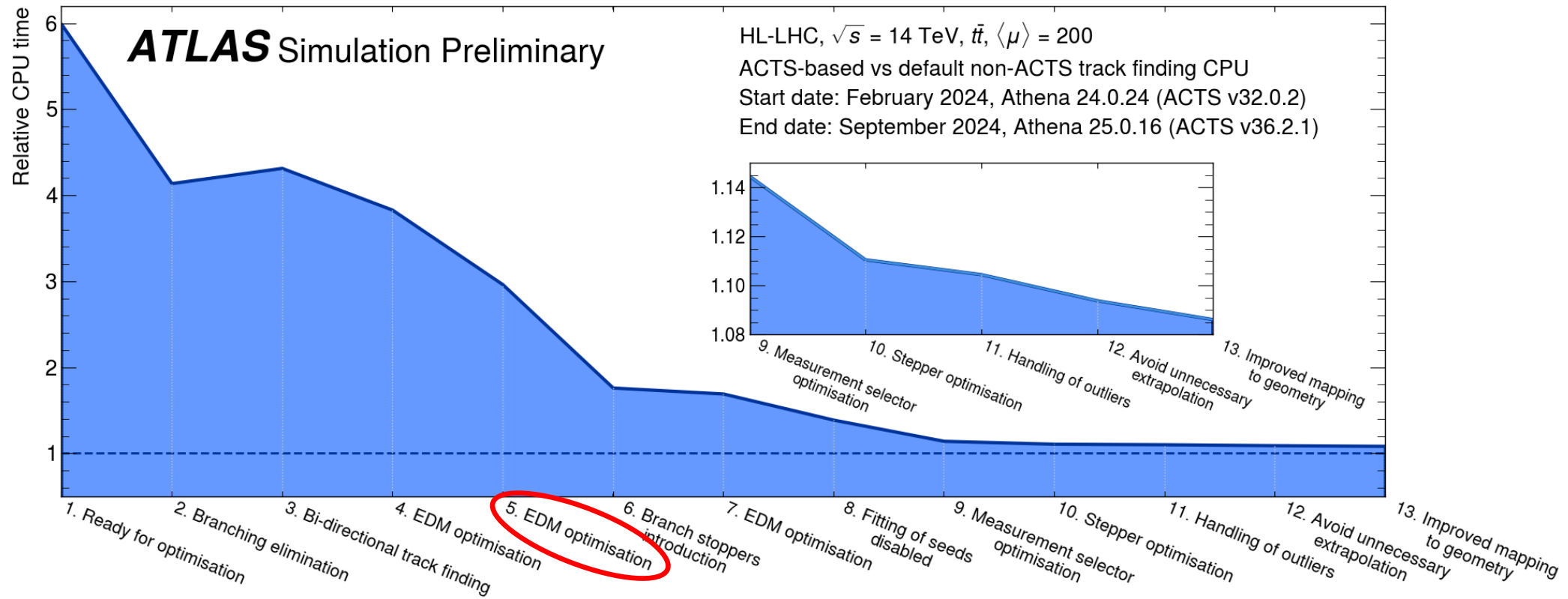
3. Bi-directional track finding: enabling extension of the seed segment both inward (i.e. toward the interaction point) and outward

CKF improvements 4



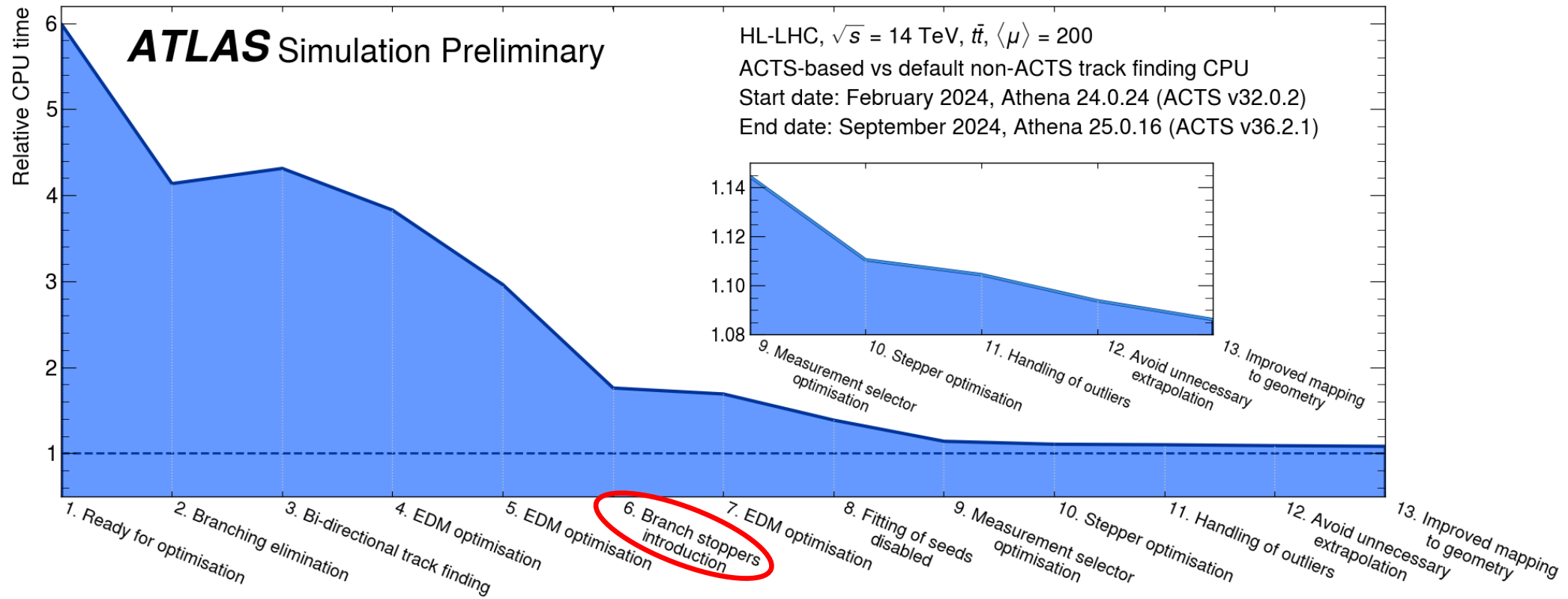
4. EDM optimisation: improved storage and access of the seed containers (Athena-only improvement)

CKF improvements 5



5. EDM optimisation: better usage of internal representation of track containers

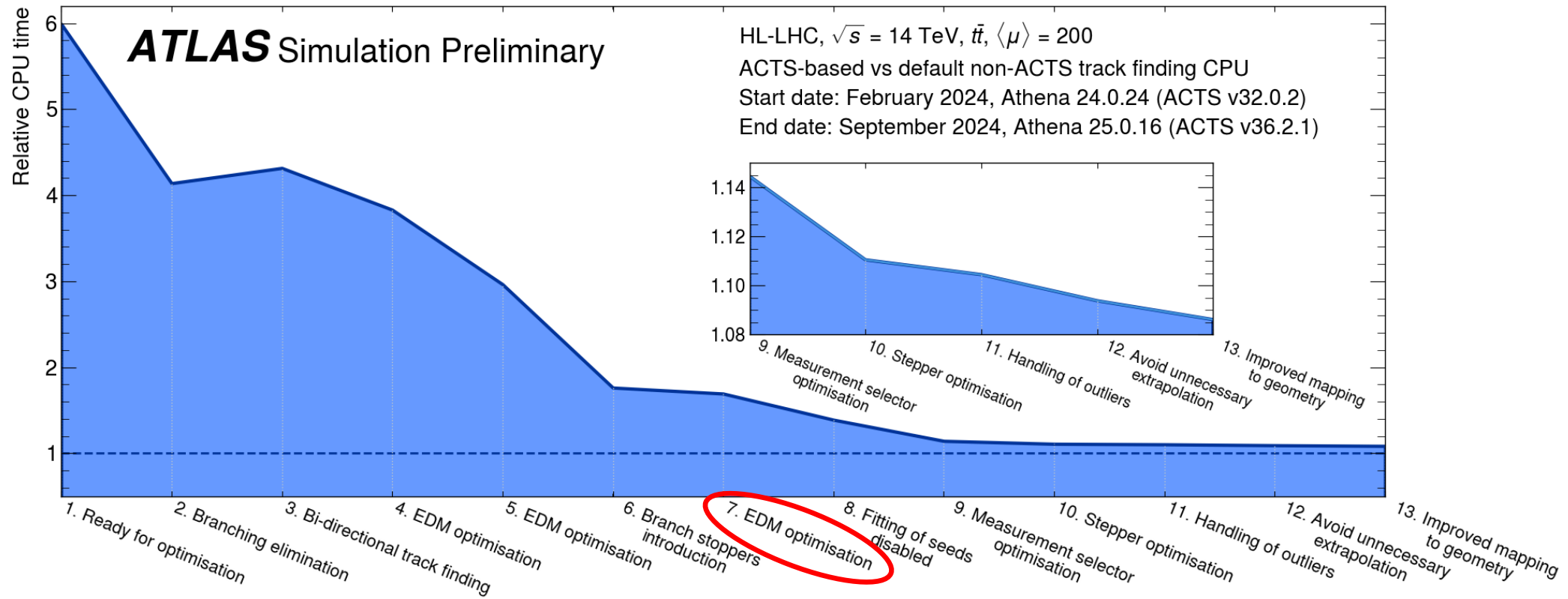
CKF improvements 6



6. Branch stoppers introduction: usage of early branch-aborting conditions to stop the track finding and decide whether to keep the track candidate

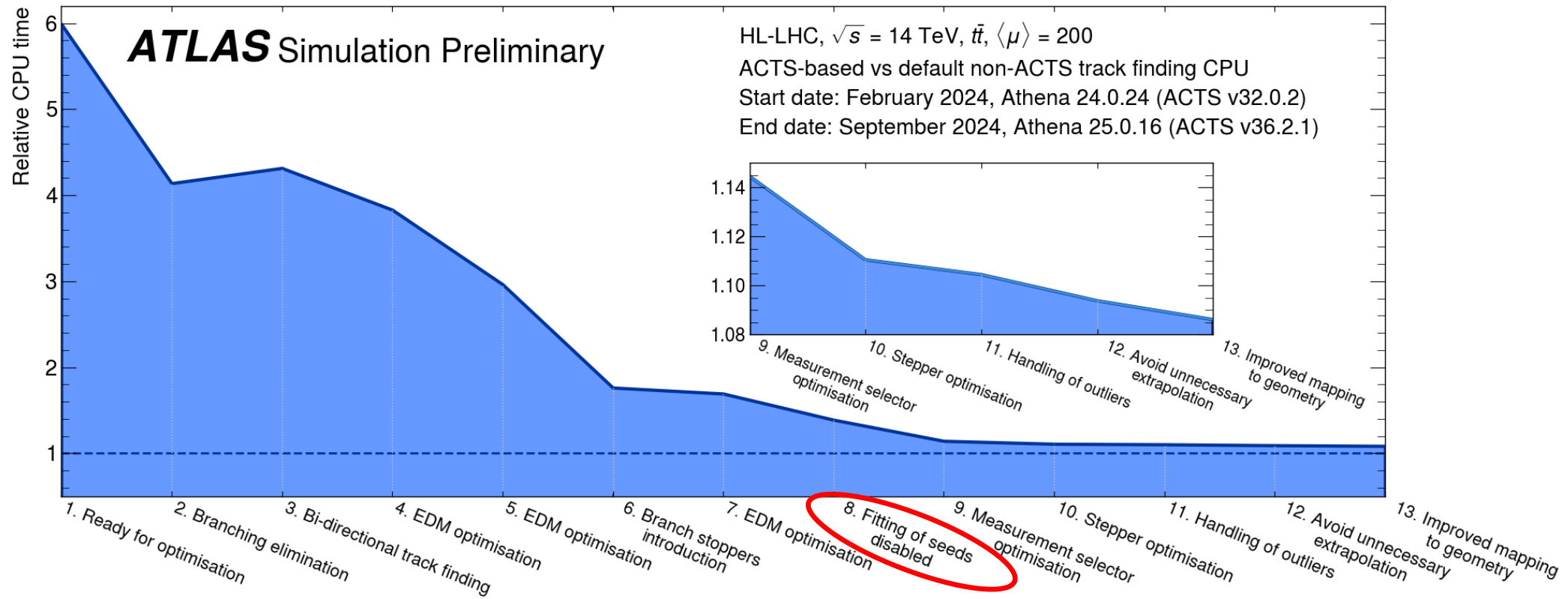
- Can stop track finding when there are too many holes
 - To prevent holes at the end of the track rejecting otherwise-good tracks, keep the track if there are already enough measurements

CKF improvements 7



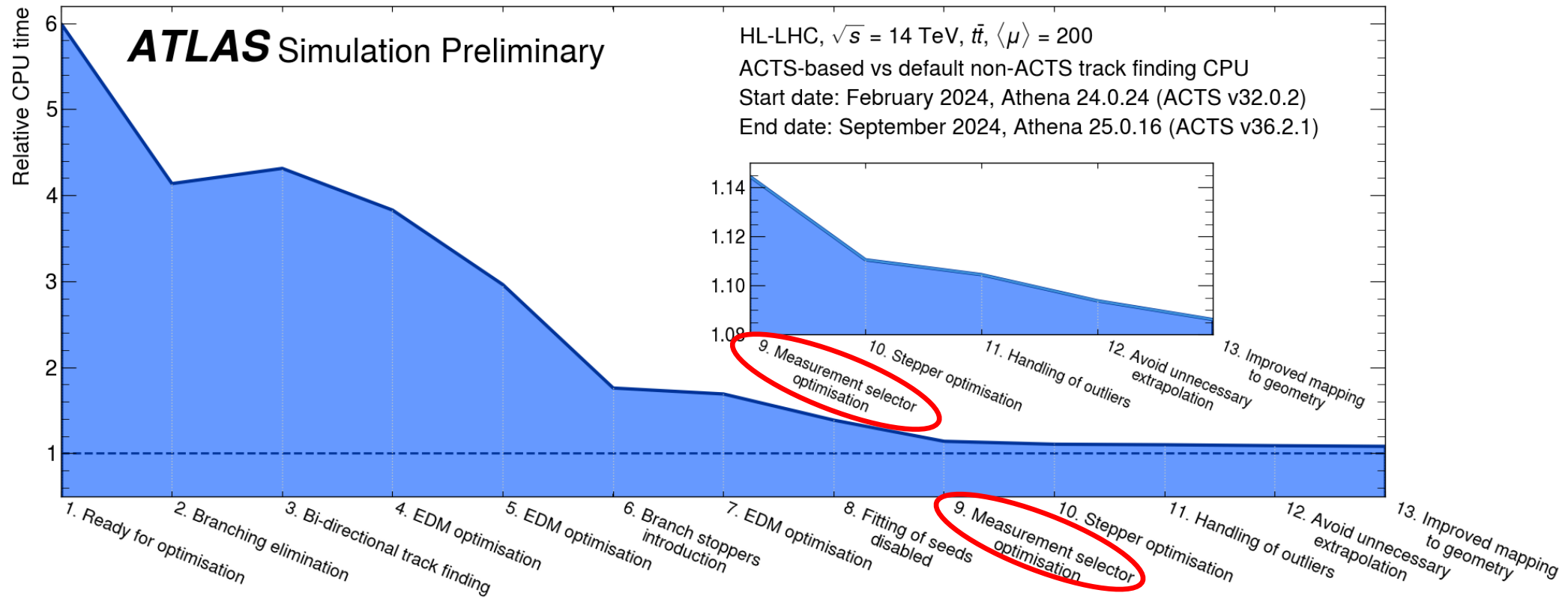
7. EDM optimisation: simplification related to navigation between track states and measurements

CKF improvements 8



8. Fitting of seeds disabled: avoiding refinement of seed parameters using the Kalman filter, as this was shown not to be critical for tracking performance (Athena-only improvement)

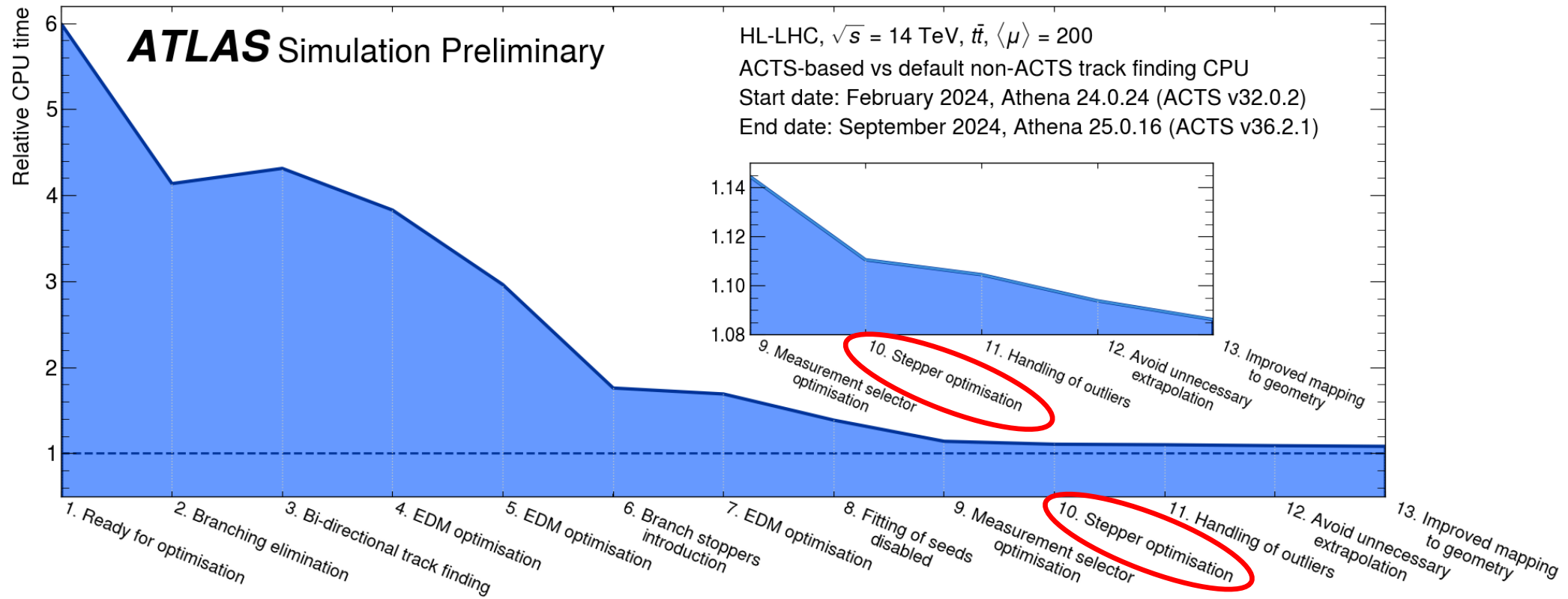
CKF improvements 9



9. Measurement selector optimisation: improved selection and calibration of measurements used in the combinatorial Kalman filter

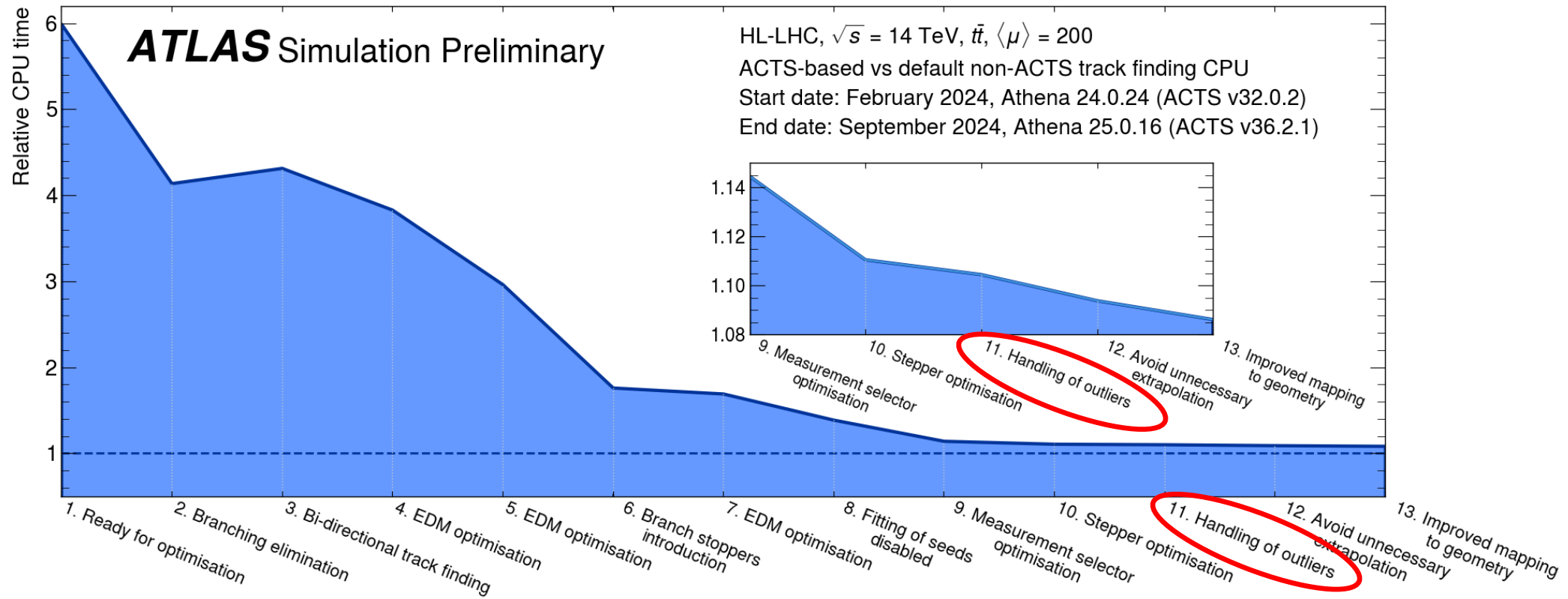
- Added custom measurement selector in Athena and improvements in the Core ACTS API.

CKF improvements 10



10. Stepper optimisation: faster implementation of the propagation stepping code

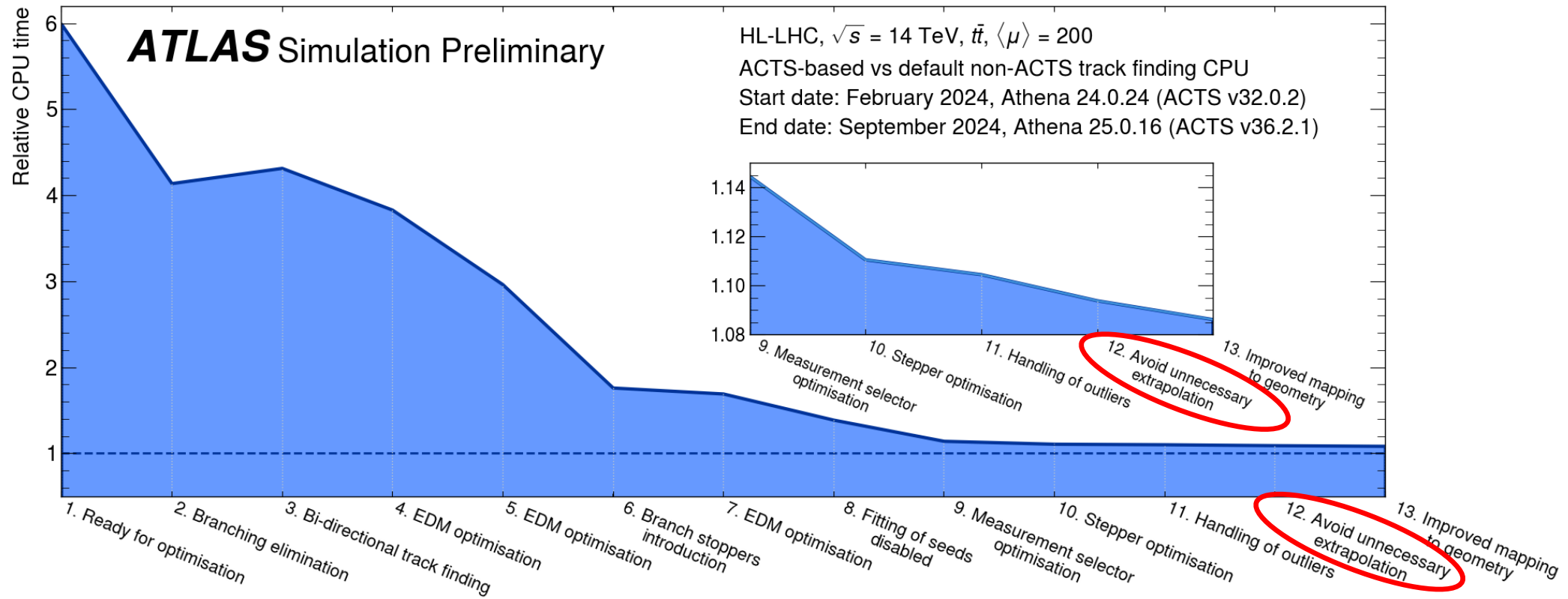
CKF improvements 11



11. Handling of outliers: implementation of a dedicated outlier compatibility criterion to reduce the number of outliers allowed per track and to stop track finding earlier

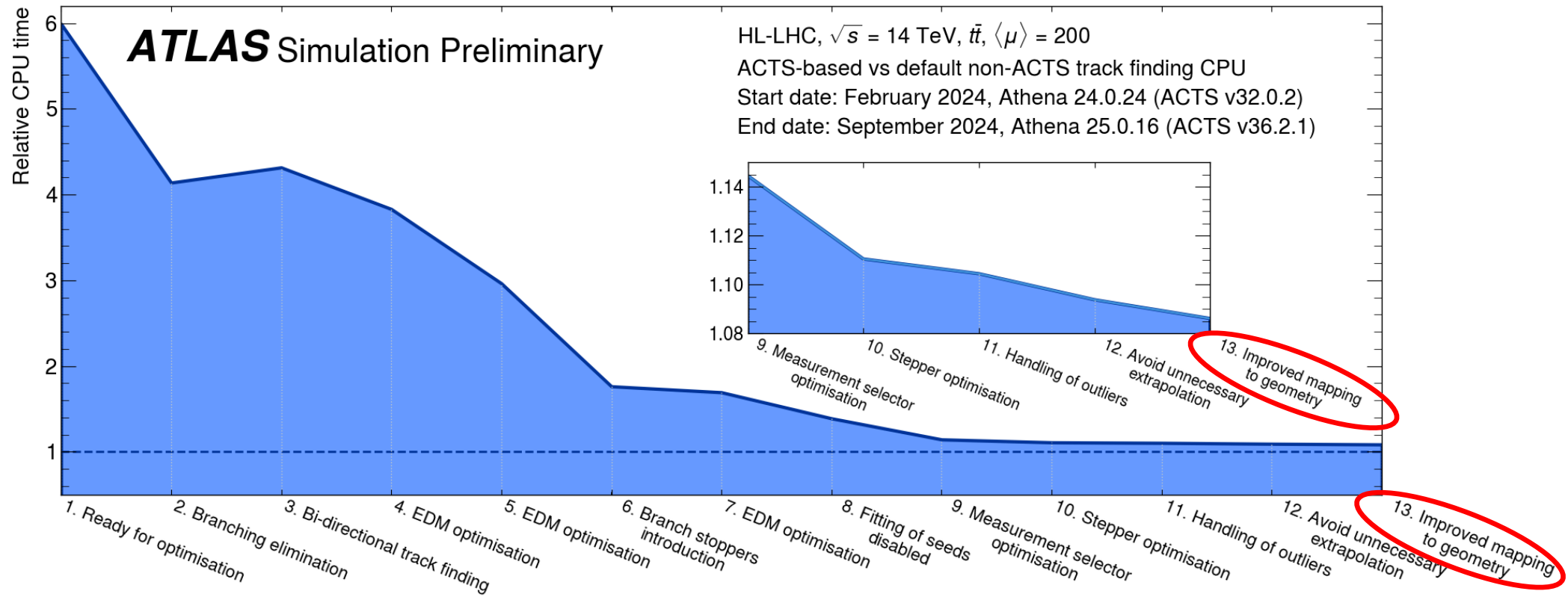
- Measurements with low compatibility with the predicted trajectory (based on a χ^2 -based criterion) are flagged as **outliers**. They are attached to the track but not used in the CKF's filtering step.

CKF improvements 12



12. Avoid unnecessary extrapolation: avoiding unnecessary extrapolation the end of track finding, in case this is already performed

CKF improvements 13



13. Improved mapping to geometry: improved mapping of detector elements and measurements to tracking geometry surfaces (Athena-only improvement)

Other recent CKF updates

- Initial parameters from seeds: can use
 1. helix interpolation between 3 spacepoints
 - Simpler and faster approach adopted since step 8
 - Updates: adapted to work with reverse search; optimised initial uncertainties
 2. KF fit through seed spacepoints
 - Doesn't much help with 3 pixel SPs, but probably better for strip seeds
 - Updates: remove χ^2 cut – don't want to drop hits
 - To do: adapt to work with reverse search
- Add separate pixel and strip hit / hole / outlier counts and cuts
 - Implemented with dynamic columns track container
 - example in ActsExamples; added to Athena

But already, for some time now, I think we can declare that...

FILTER IS GO



An early use of the **Kalman Filter** was for the US Space Programme. This is **Flight Dynamics Officer**, Jay Greene, 35 minutes before the **Apollo 11** moon landing on 20th July 1969. The Kalman Filter was used to calculate the LEM's trajectory. During the descent, "**Filter is go**" was regularly reported to confirm that the computer was keeping up with the calculations.



CKF next steps

1. We are missing hits near the edge of the modules
 - Possibly due to navigation finding neighbouring module, while both compatible within errors
 - This leads to holes, and if too many will drop track
2. Calculate shared hit count (and cuts?) immediately after track finding
 - Prototype in Athena, then move to ACTS core
 - Also add other detector-specific track summary info: strip double-holes, ganged pixel hits (ATLAS-specific)
3. Start with strip seeds with reverse search, then do pixel seeds with forward search
 - The machinery exists for this, but need to gauge performance before enabling
4. Is it useful to do road building in CKF?
 - If so, could be done with the DirectNavigator
5. Can we stop extrapolating sooner, e.g. when we know this is the last sensitive surface?
6. We suspect that further large improvements in the CKF track finding can come from starting with better seeds
 - Fewer, higher purity

GNN

- The **ATLAS GNN4ITk group** has recently started developing the GNN-based chain to suit the needs of ATLAS EF tracking

- Aim is to have full GPU chain:

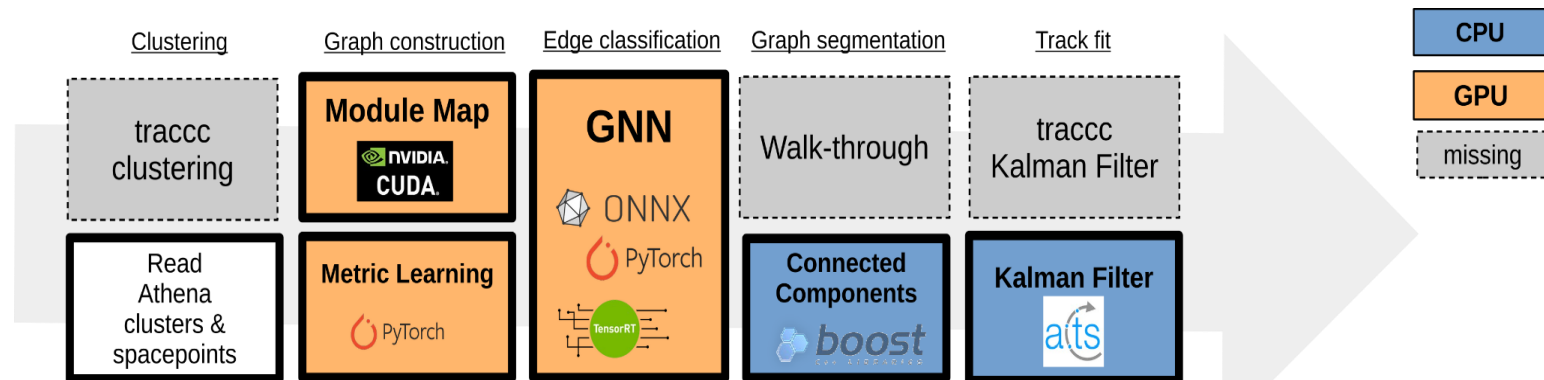
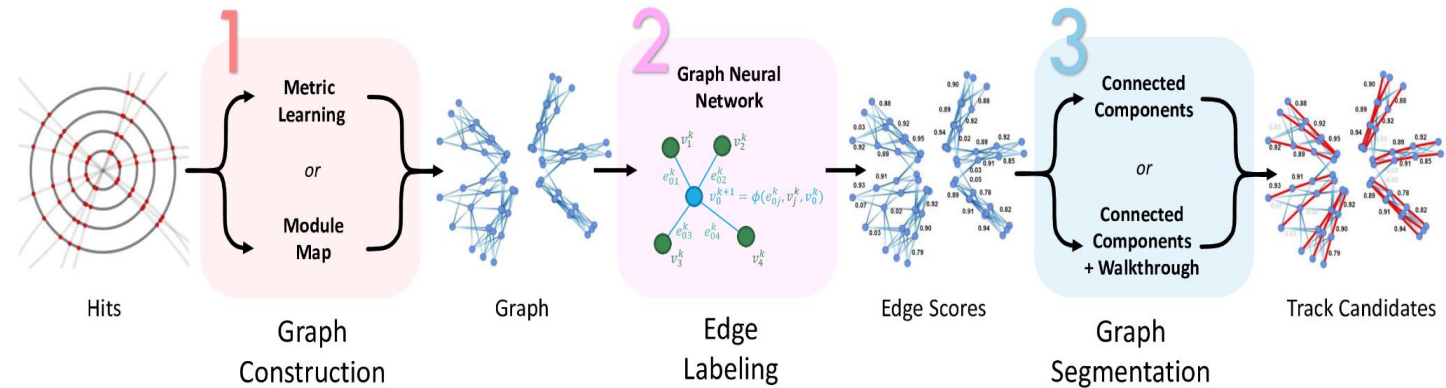
- tracc clustering + GNN track finding + tracc track fitting

- Currently track fitting performed on CPU

- Ongoing work on **designing the infrastructure** for performance studies

- Standalone **ACTS-based GNN workflow**

- ATLAS simulation is used as input data to ensure realistic conditions

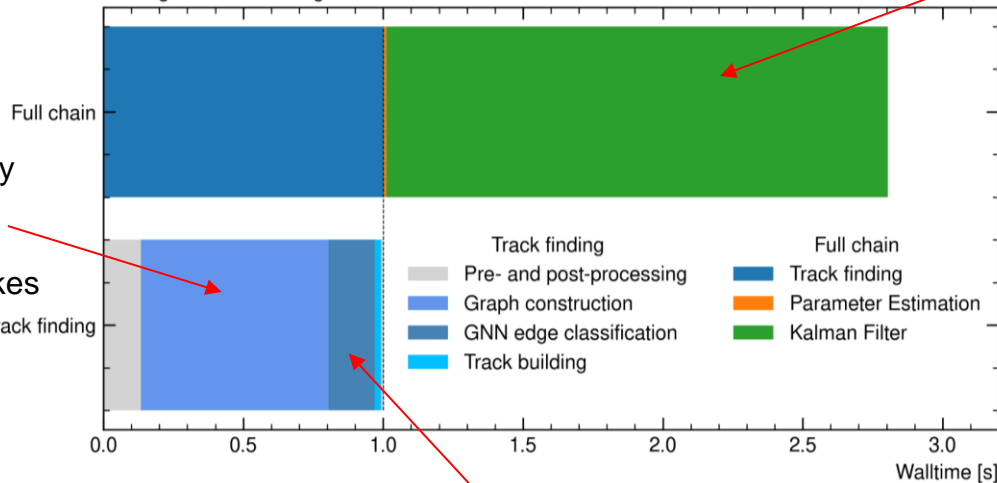


Work ongoing on implementing the missing components

- Preliminary tracking performance is obtained using a **standalone ACTS-based GNN workflow**
 - Graphs are constructed using the “Module Map” approach
 - The GNN assigns a classification score s to each edge
 - Track candidates are built considering edges with score $s > 0.5$ and using a connected-components algorithm

Work in progress

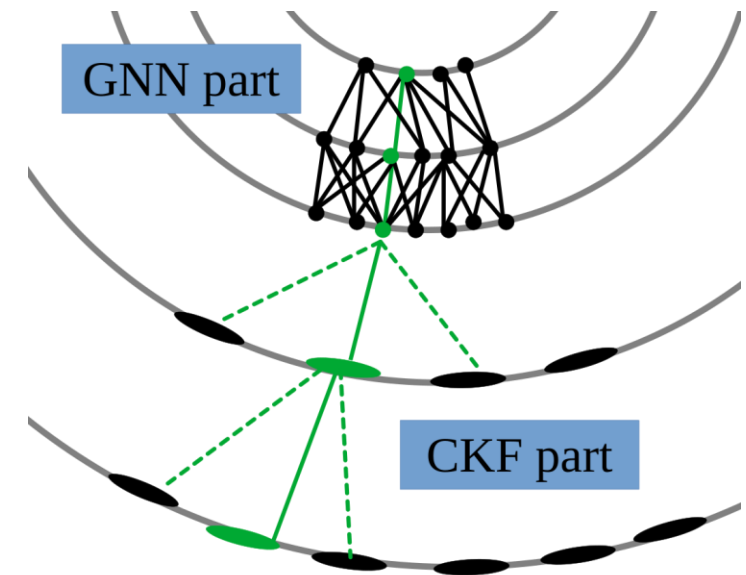
HL-LHC, ITk Layout: 03-00-00, $t\bar{t}$, $\langle\mu\rangle = 200$, $\sqrt{s} = 14$ TeV
Average full chain timing of the ACTS standalone GNN workflow



Kalman Filter on single CPU core

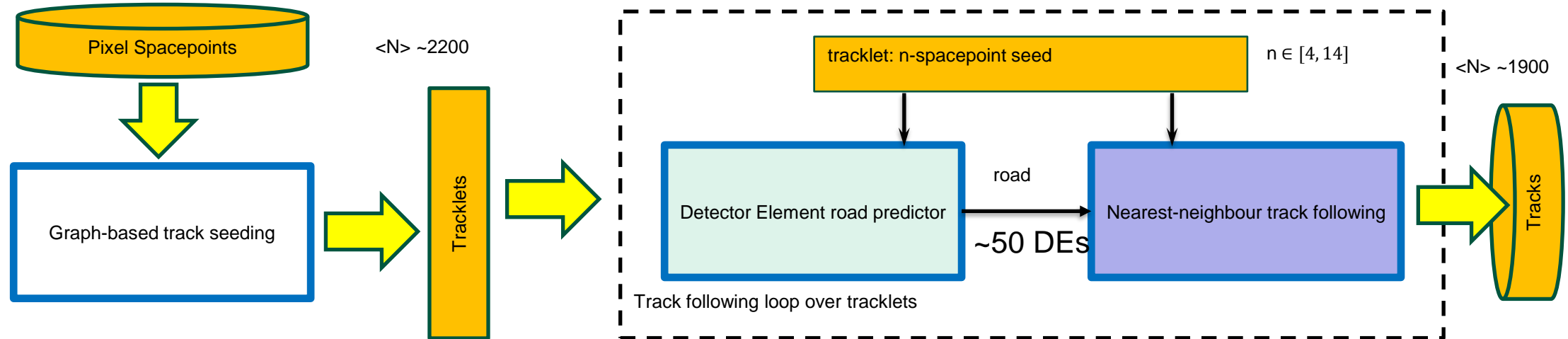
- First example of timing measurements with Nvidia A100
 - Promising timing for individual steps
 - but far from ready to judge if the pipeline is viable or not
 - Chain not yet fully on-device
 - data copied to host after each sub-step
 - GPU device not saturated
 - still need to get some throughput measurements and to try to saturate the devices

- More and more developments to come
 1. Enhance ACTS framework to support throughput measurements
 - Widespread discussion in ATLAS on how to compare different technologies
 2. Integrate missing components (walkthrough, tracc fitter)
 - We need to be able to use these functionalities directly in Athena
 3. GNN performance optimisation
 - Investigate using GNNs on FPGAs
 4. Investigate pixel-only GNN tracking with CKF extension
 - there was already a study done on the ODD with this concept, and it should now be extended to the ITk.
 - An implementation of the modified CKF is already available [upstream](#).
- This is very much in-flight
 - interested parties are very welcome to approach the **GNN4ITk group** and try it out



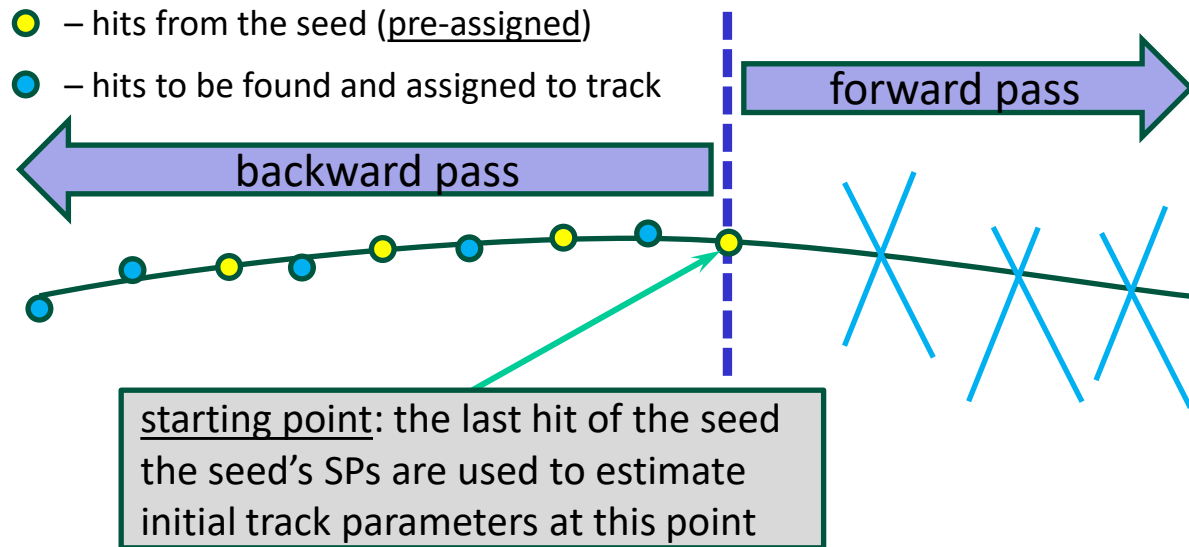
NNF

- Pixel Seeding (**GBTS**) and Track Finding (**NNF**) algorithms have been developed for the ITk within the legacy Athena trigger framework (**TrigFastTrackFinder**)
 - Both show very promising CPU performance with good tracking performance
 - Would like to take advantage of this within ACTS:
 - Porting GBTS seeding is already well advanced (see [Rosie's talk](#)). Next step is GBTS v2.0.
 - Plans to port the NNF Track Finding (AKA Seed Extension) also to ACTS



- A big factor in improving the track finding CPU performance is the **seed quality**
 - the Track Seeding is basically a fast Track Finding algorithm operating on pixel spacepoints
 - it produces **high-purity pixel seeds** with 75-95% (η -dependent) chance to result in a good track
 - The track finding step extends the pixel seed, especially to pick up (for the first time) strip measurements

- The new track finding algorithm is based on a few techniques:
 - non-combinatorial, nearest-neighbour filter
 - Pixel seed precision is good enough to resolve hit ambiguities without track splitting, branching, etc.
 - road-based (rather than navigation-based) search for next hit on track
 - third-order Runge-Kutta method (Heun's R-K scheme) for track state propagation
 - using fixed-point(s) smoother instead of the usual fixed-interval smoothing algorithm:



- The track state has two parts – head and tail:
 - each is 5-dim: $(x_{loc}, y_{loc}, \varphi, \theta, \frac{q}{p})$
 - they have a joint 10x10 covariance matrix
 - the head is extrapolated, the tail is stationary
 - the head state is updated via the standard Kalman filter mechanism
 - the tail is updated too thanks to the joint covariance matrix
- Once the forward pass is finished, head and tail are swapped and we ready for the backward pass!

- Times per $t\bar{t}$ at $\langle \mu \rangle = 200$ event on Intel Xeon Gold 6430 3.4GHz CPU (HS23 score 25.7)

Track seeding	Track following	SpacePoint conversion	Total seeding + TF time
166 ms	171 ms	17 ms	358 ms

Work in progress

- Converting to HS23 seconds, the GBTS+NNF time is **9.2 s** vs **28.8 s** for legacy “fast tracking” (similar to ACTS)
 - 3.1 × faster!

- The performance of the GBTS+NNF pipeline looks good in terms of both CPU time and tracking quality
- The pipeline is fully integrated into Athena as an Athena Algorithm
 - dedicated ART tests are in place
- The ongoing work is focussed on further CPU time improvement:
 1. simplifying spacepoint loading path
 - seems like (deceptively?) low-hanging fruit
 2. possible speed-up of the track finding tool using vectorization helpers from [Athena CxxUtils](#)

- Once the algorithm development within Athena is mostly complete, will look at porting to ACTS
 - Can then compare ACTS called from Athena against the Athena-native implementation
 - Expect identical tracking performance, hope for similar CPU performance
 - In the meantime, look at mixing the components, e.g. Athena GBTS v2.0 + ACTS CKF
 - This can show how much improvement comes from better seeds, or faster track finding

- CKF optimisation has given us a $\times 6$ speedup
 - Optimisation continues of tracking and CPU performance
- GNN pipeline aiming for the ITk full pipeline running on GPU
 - Track Finding component being tested now
- NNF suggests a $\times 3$ speedup compared to the CKF
 - Aim to have GBTS v2.0 seeding and NNF Track Finding available within ACTS

Backup

Recent CKF updates

- More recent improvements (@andiwand), incorporated in [v37.1.0](#)
 - Combine material, measurement and hole handling in Core CKF [#3723](#)
 - Physmon for KF and GSF refitting [#3733](#)
 - Implement DirectNavigator direction handling [#3702](#)
 - Allow reflection of track parameters [#3682](#)
 - Add estimateTrackParamCovariance to Core [#3683](#)

- The NNF algorithm doesn't require storing intermediate track parameter estimates for subsequent smoothing
 - much smaller memory footprint, and no dynamic memory allocation
 - basically, the filter handles a single instance of `TrigFTF_ExtendedTrackState` which at the end contains the smoothed estimates at the end of the track and at the perigee point
- A set of detector elements to search for hits is given by `SiDetElementsRoadTool_xk`
 - their order is re-arranged so that the forward and backward passes can be conveniently wrapped into a single for-loop
- The NNF track finding tool can be used straight from `SiTrackMaker_xk`
 - as a replacement for `SiCombinatorialTrackFinder_xk`

```
//tracks = m_tracksfinder->getTracks(data.combinatorialData(), *Tp, Sp, Gp, DE, data.clusterTrack(),ctx);
```

```
Trk::Track* newTrack = m_trigInDetTrackFollowingTool->getTrack(Sp, DE, ctx);  
if(newTrack!=nullptr) tracks.push_back(newTrack);
```

