Improving Tracking Algorithms with ML: A case for Line Segment Tracking at the HL-LHC

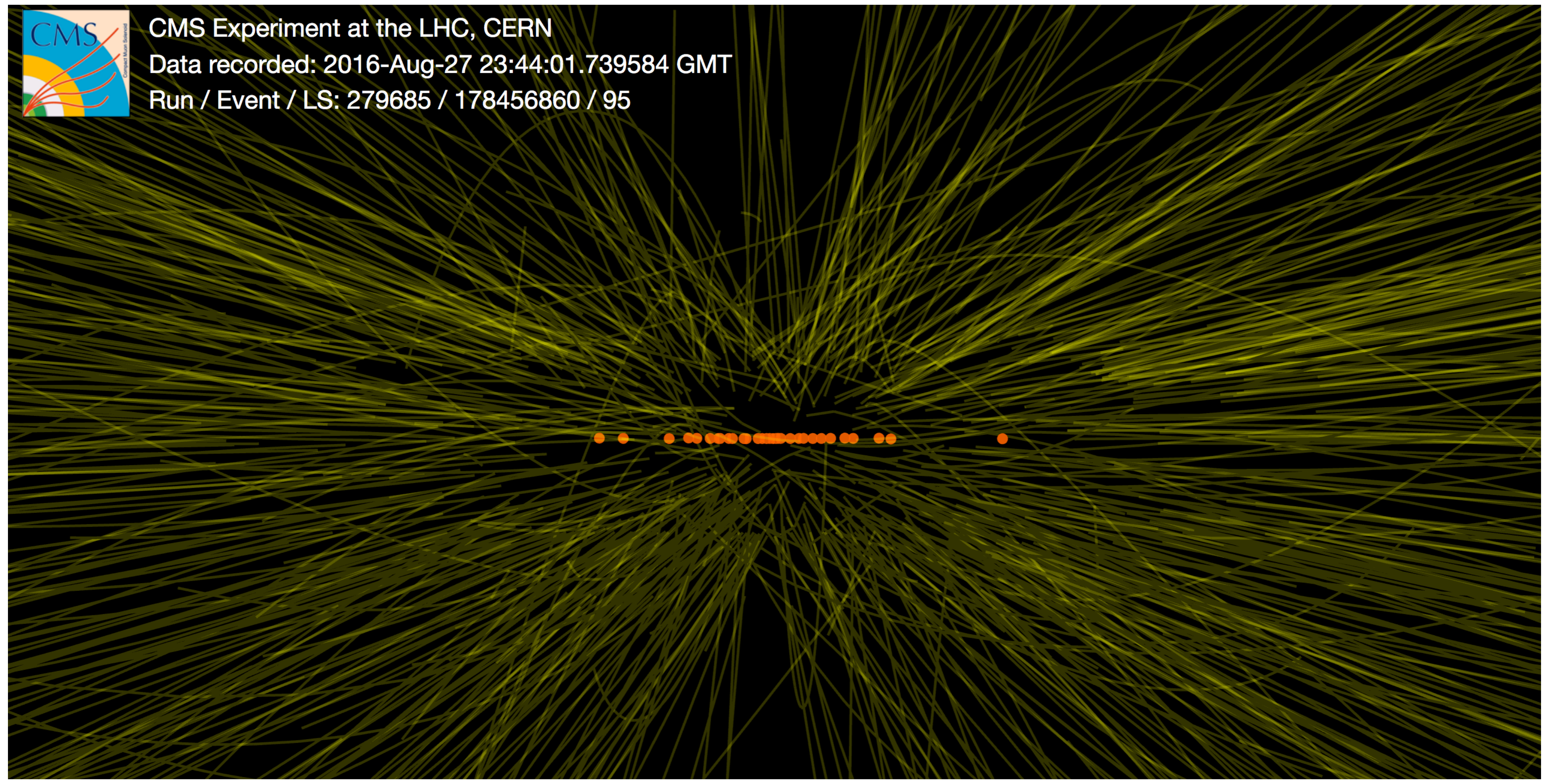
On behalf of the CMS Collaboration

October 12th, 2023

UC San Diego
Cornell University
PRINCETON UNIVERSITY
UF UNIVERSITY of FLORIDA

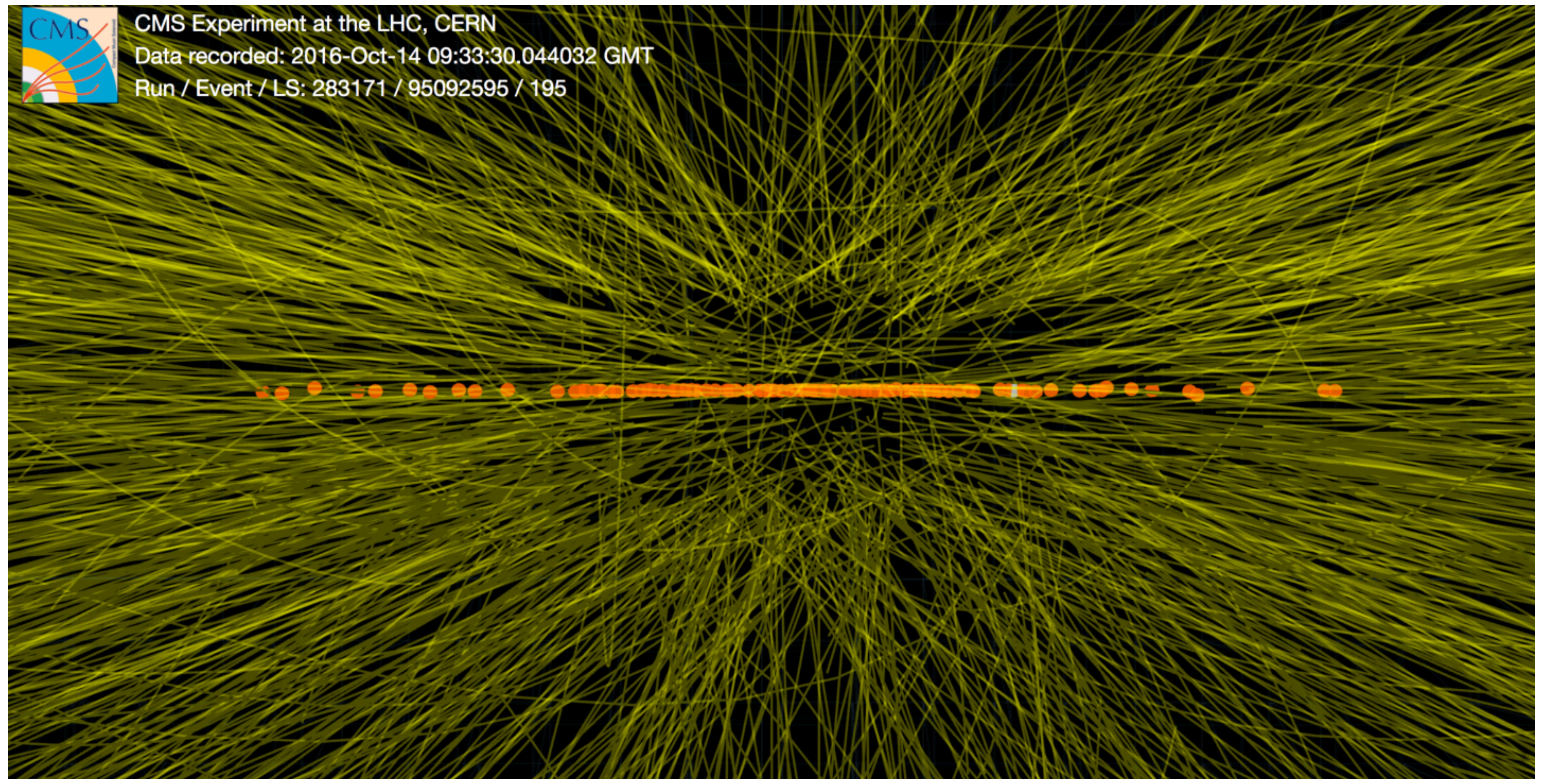
Challenge: HL-LHC Tracking

"High Luminosity" LHC (HL-LHC) planned for 2030s



Nominal Run 2 event (PU 30)

→



HL-like event (PU 130)

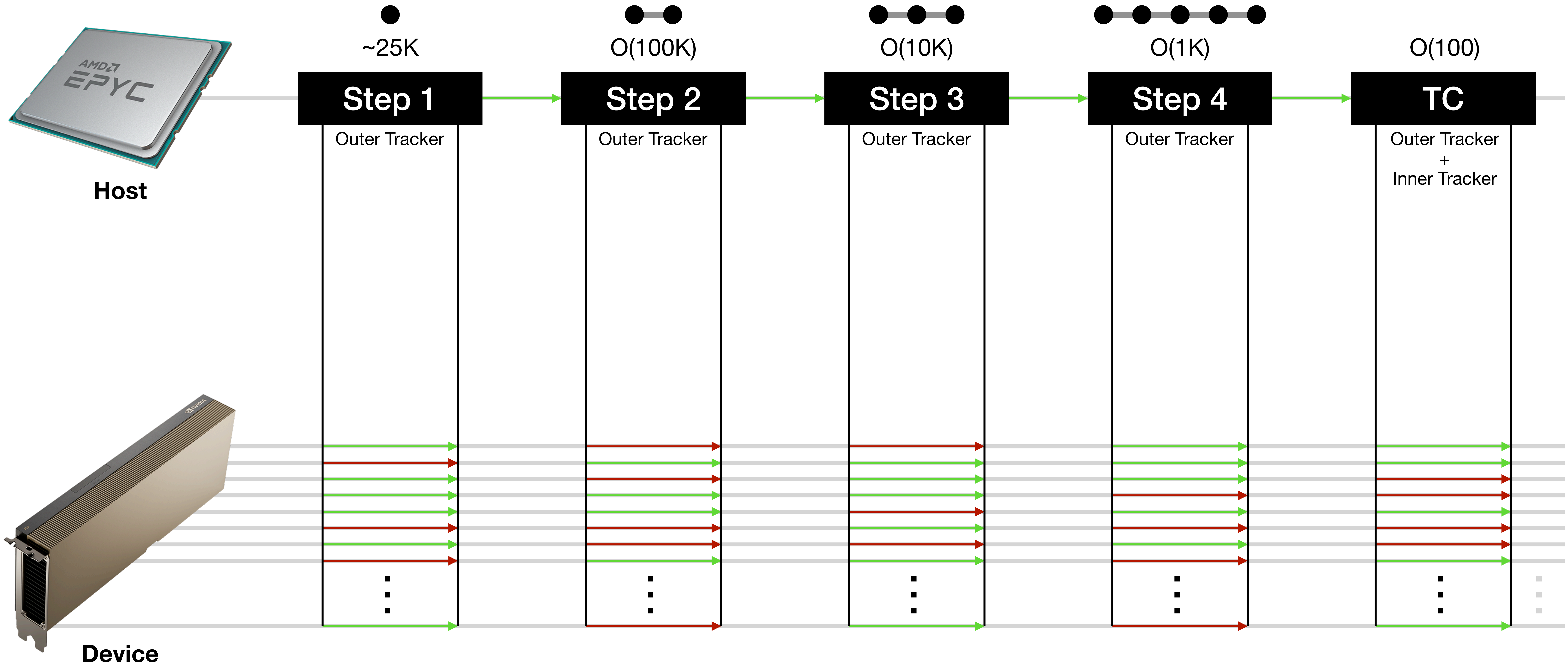
O(10x) concurrent collisions ("pile up") = O(10x) tracks
⇒ **need a fast tracking algorithm** to keep up

Current tracking algorithm is inherently sequential ⇒ poor scaling

We propose **LST: a highly parallelizable tracking algorithm**

Solution: Line Segment Tracking (LST)

At each step, one thread per object: deciding **keep** or **discard**

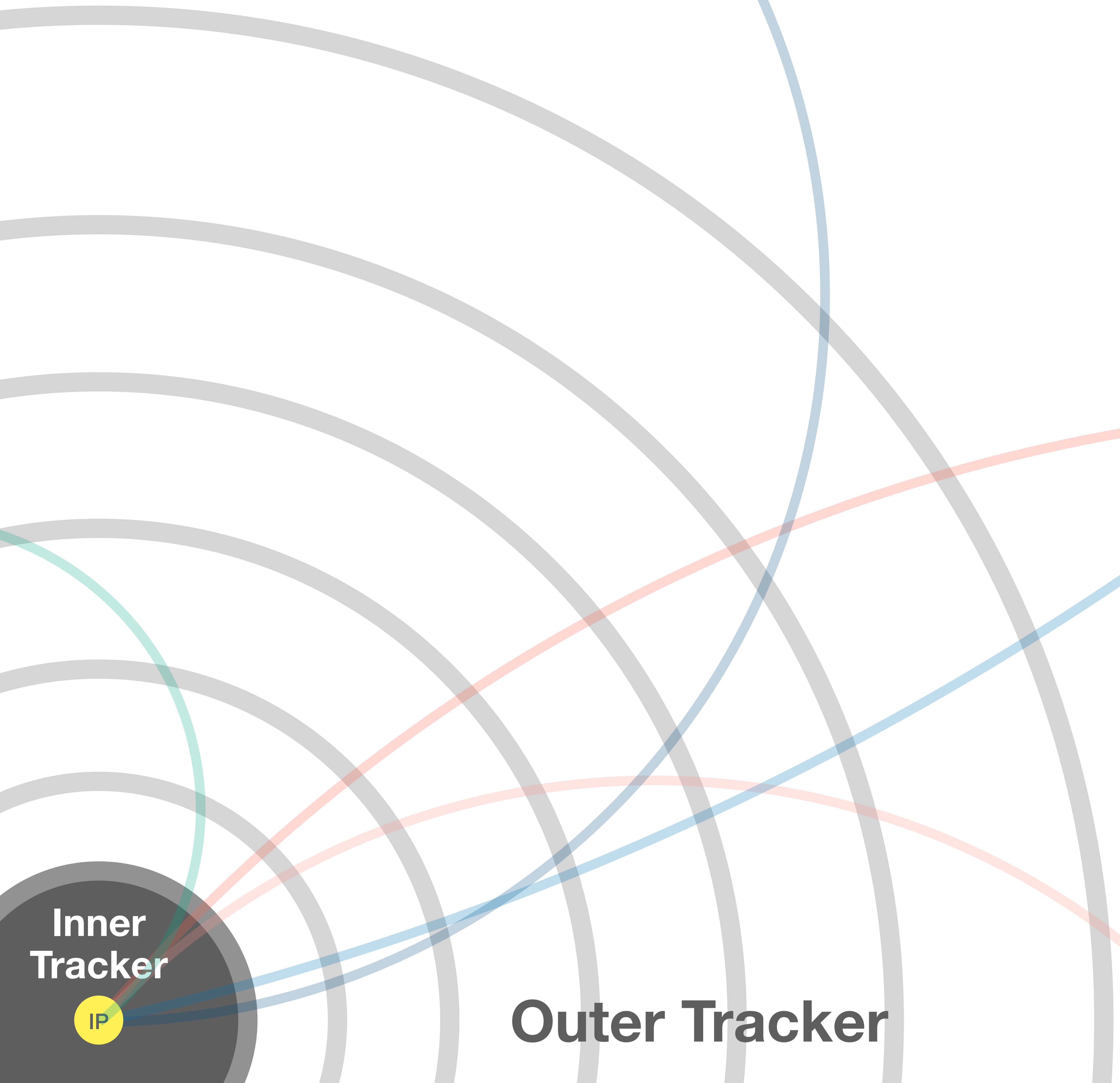


Each step is designed such that objects can be assessed independently

⇒ **massively parallelizable!**

UC San Diego

Solution: Line Segment Tracking (LST)



We will provide a basic description of each step of the LST algorithm here

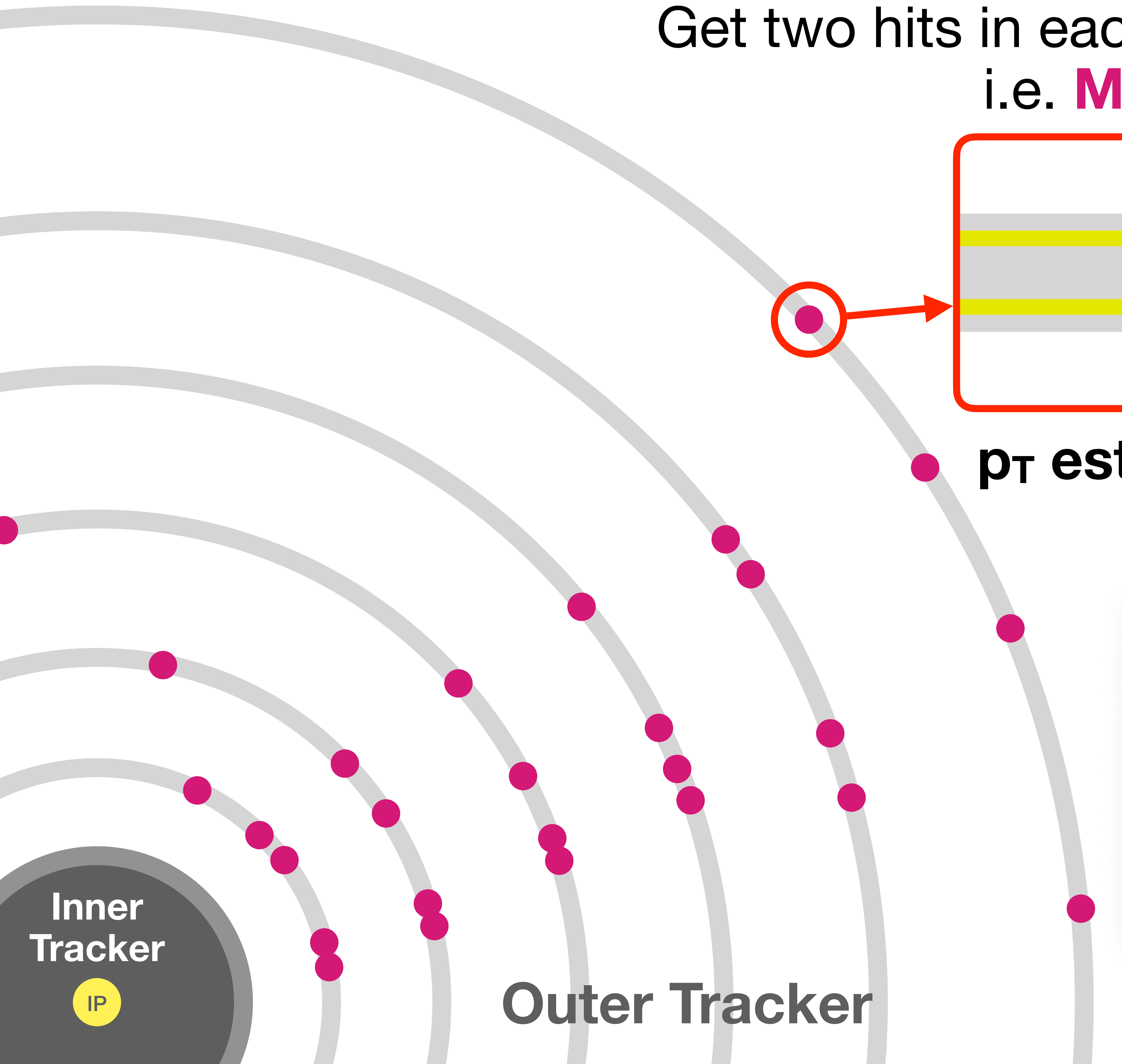
We show the steps here only to **introduce the LST terminology/context**

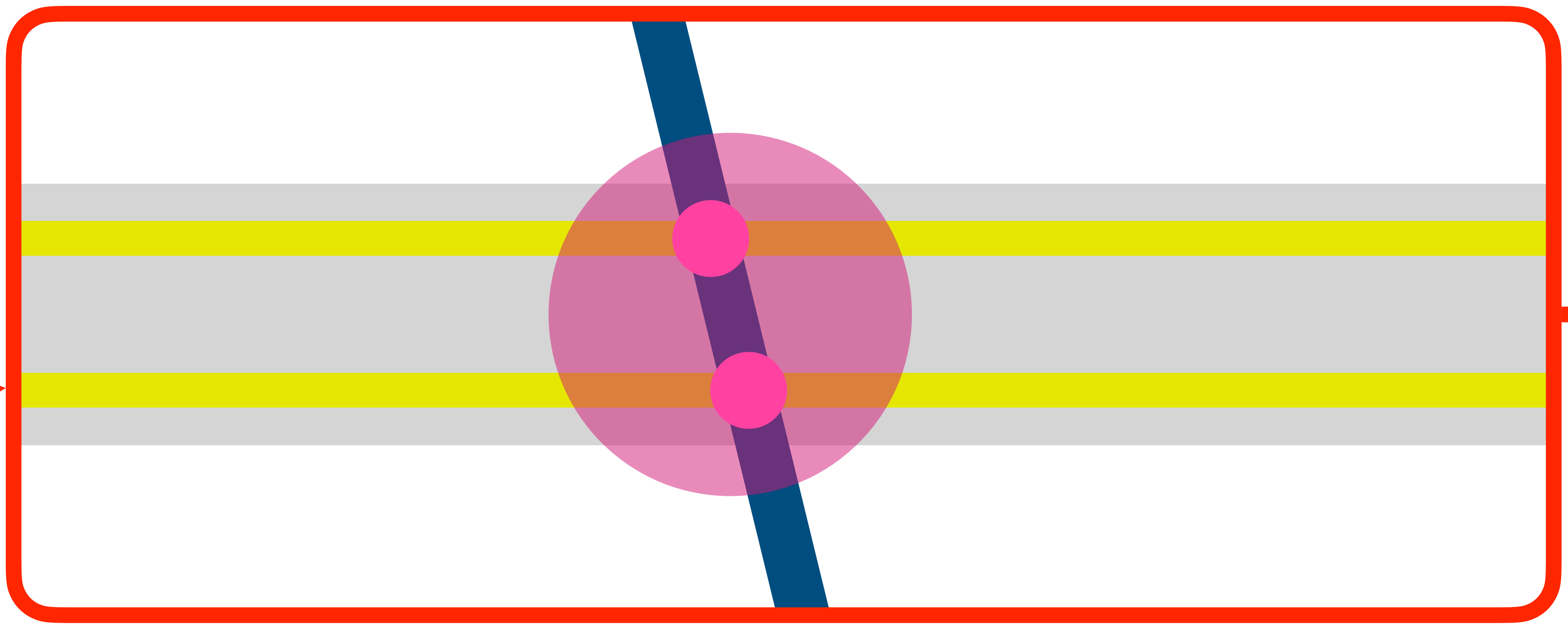
More info on LST can be found here:
CTD 2022, CHEP 2023, CMS DP Note

UC San Diego

LST in a Nutshell: Mini-Doublets

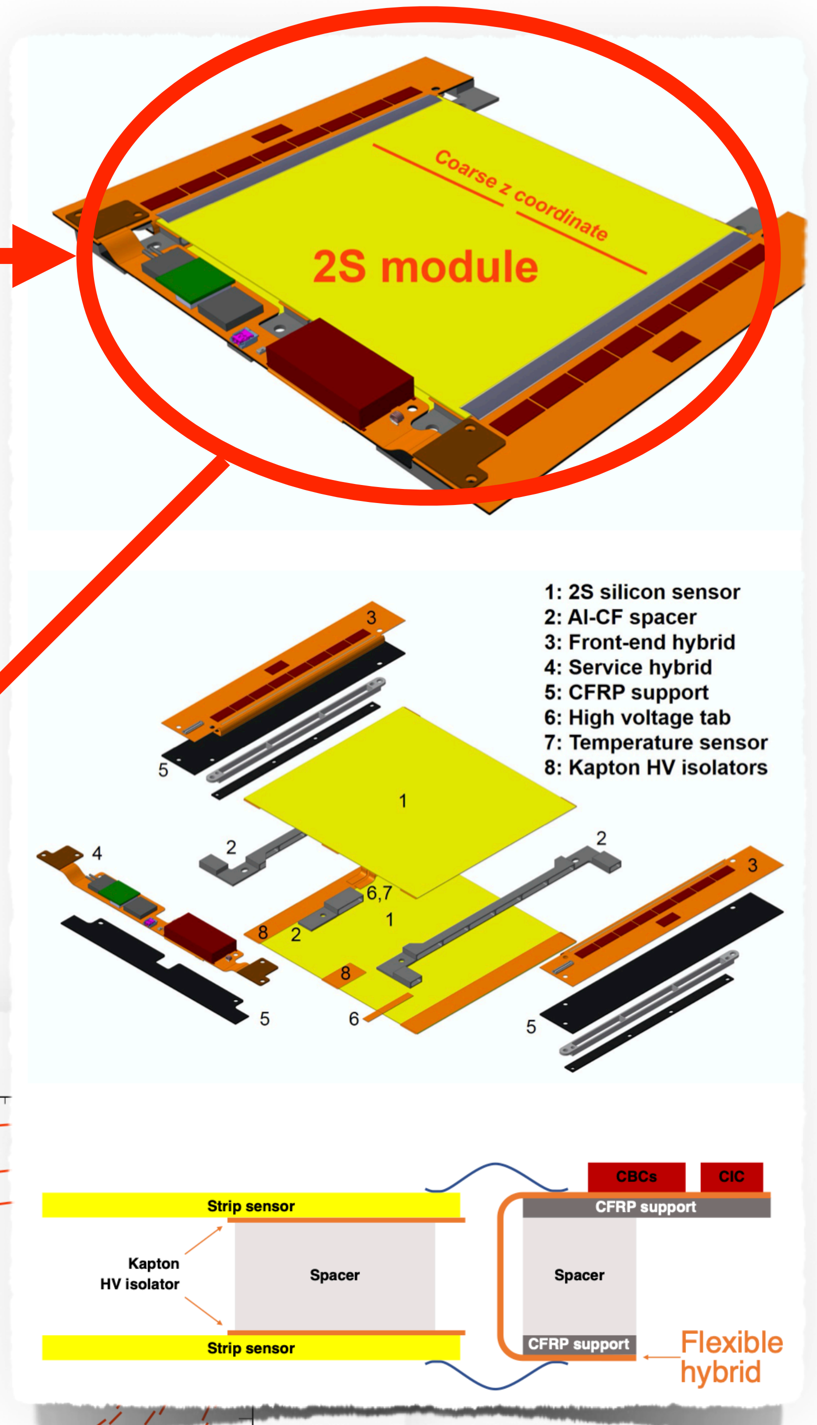
Get two hits in each layer in Phase 2 Outer Tracker:
i.e. **Mini-Doublets (MDs)**





p_T estimate for each MD

“p_T module”



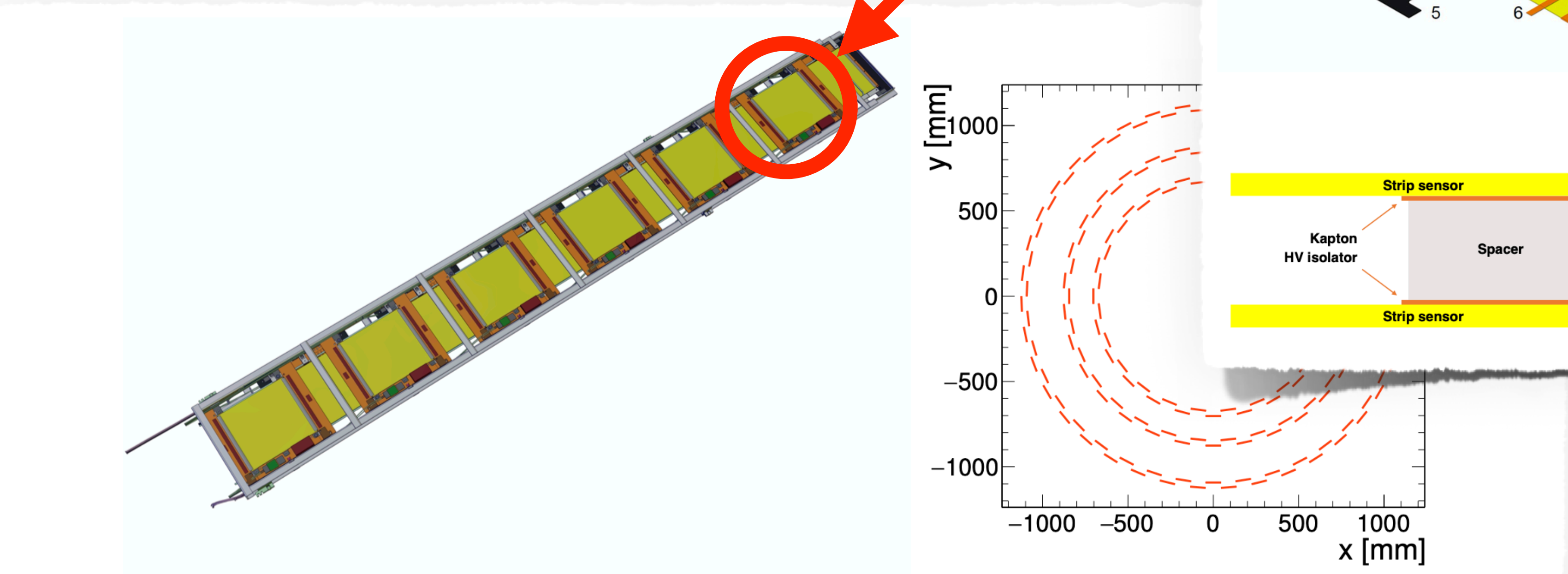
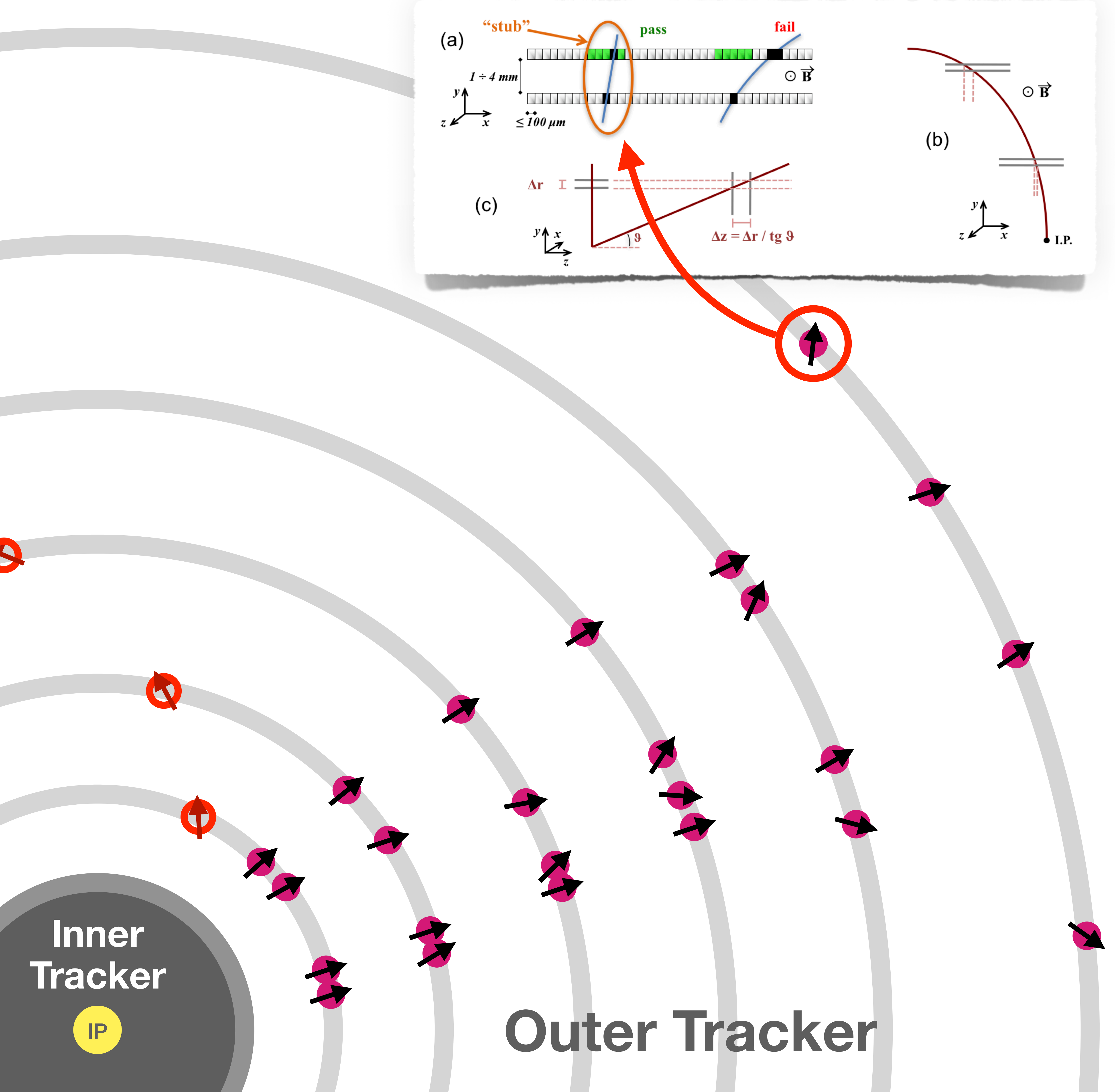


Figure 3.3: Left: model of a TB2S ladder, housing twelve 2S modules. Right: x-y view of the TB2S, showing the staggering of neighbouring ladders.

UC San Diego

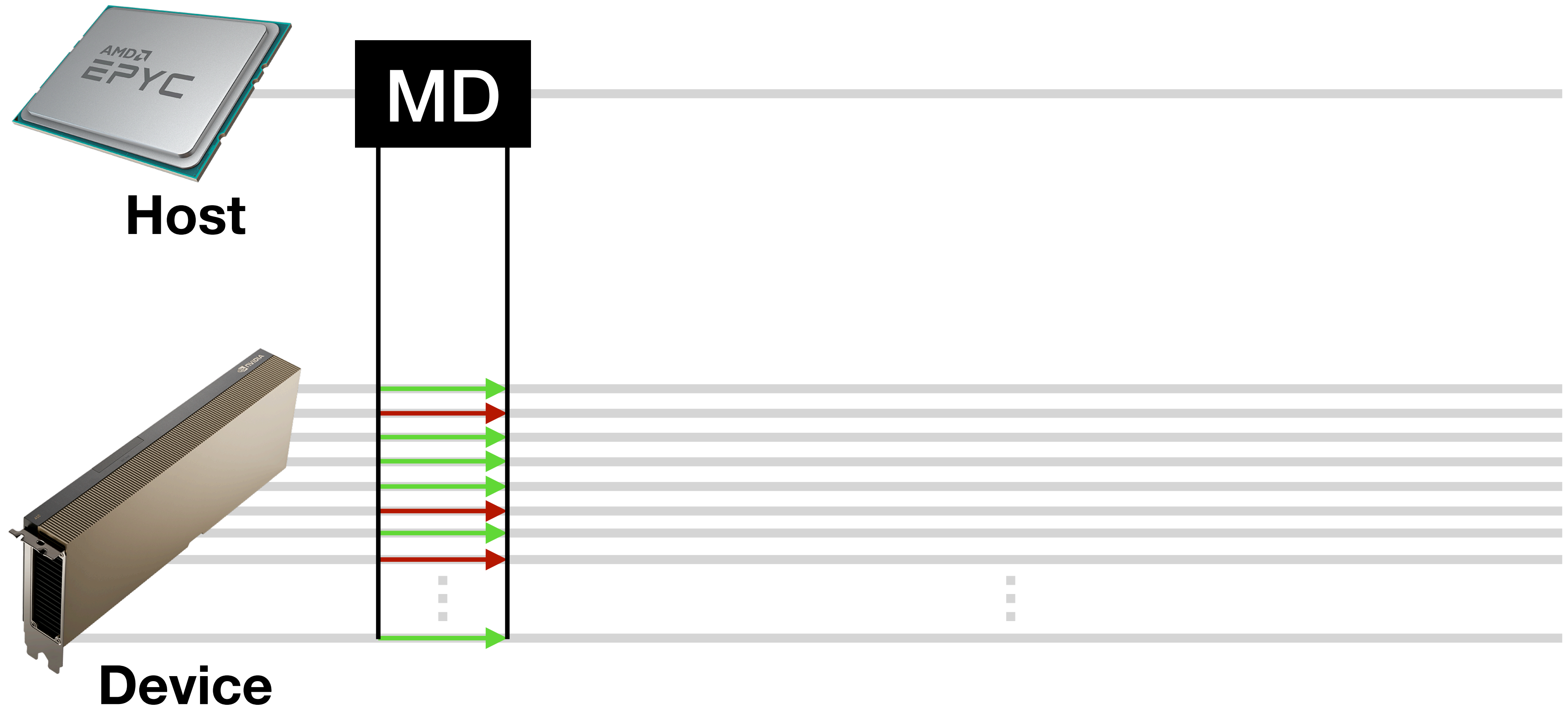
LST in a Nutshell: Mini-Doublets



Build all **good MDs**

good = **p_T > 0.8**

One thread per MD

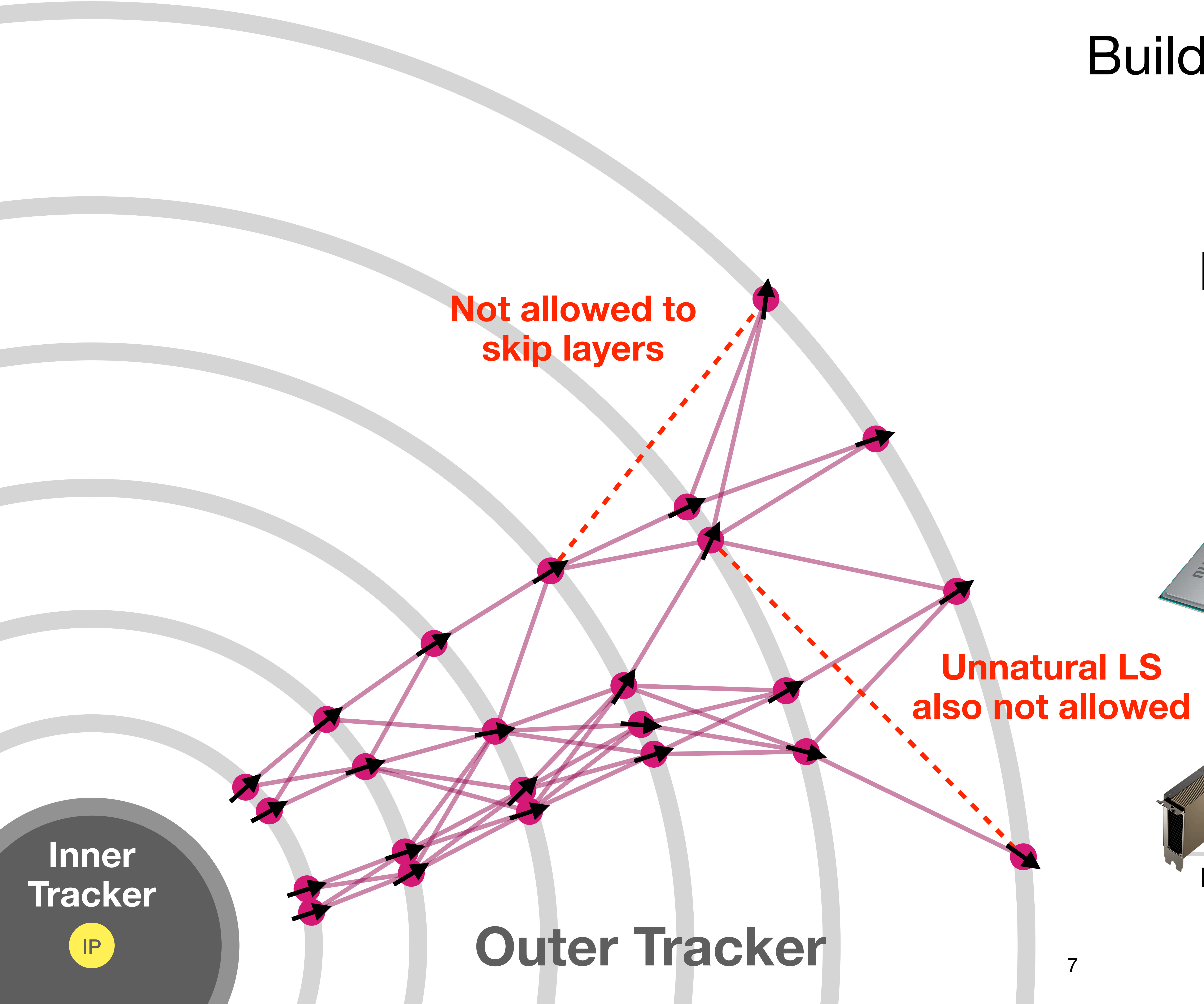


UC San Diego

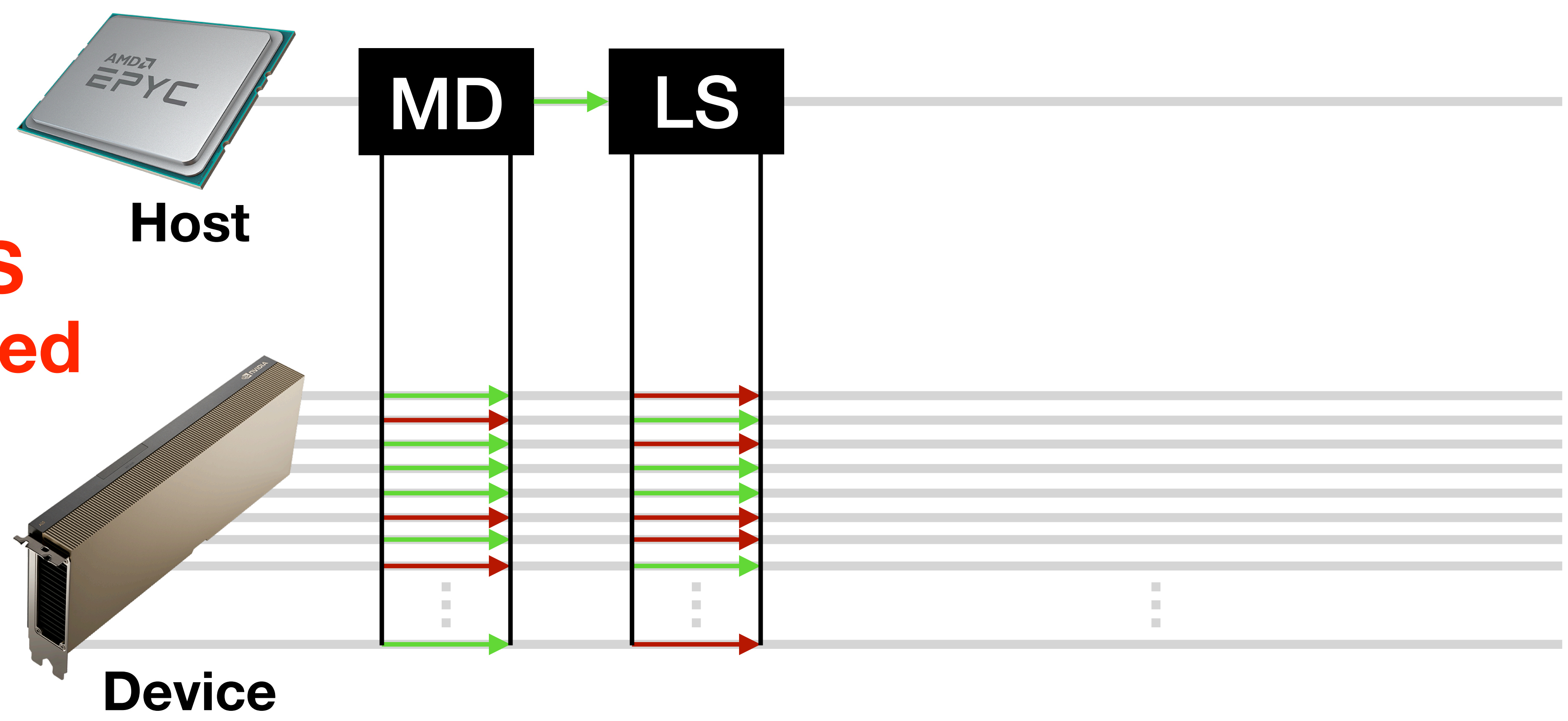
LST in a Nutshell: Line Segments

Build all **valid** connections of two MDs:
i.e. **Line Segments (LSs)**

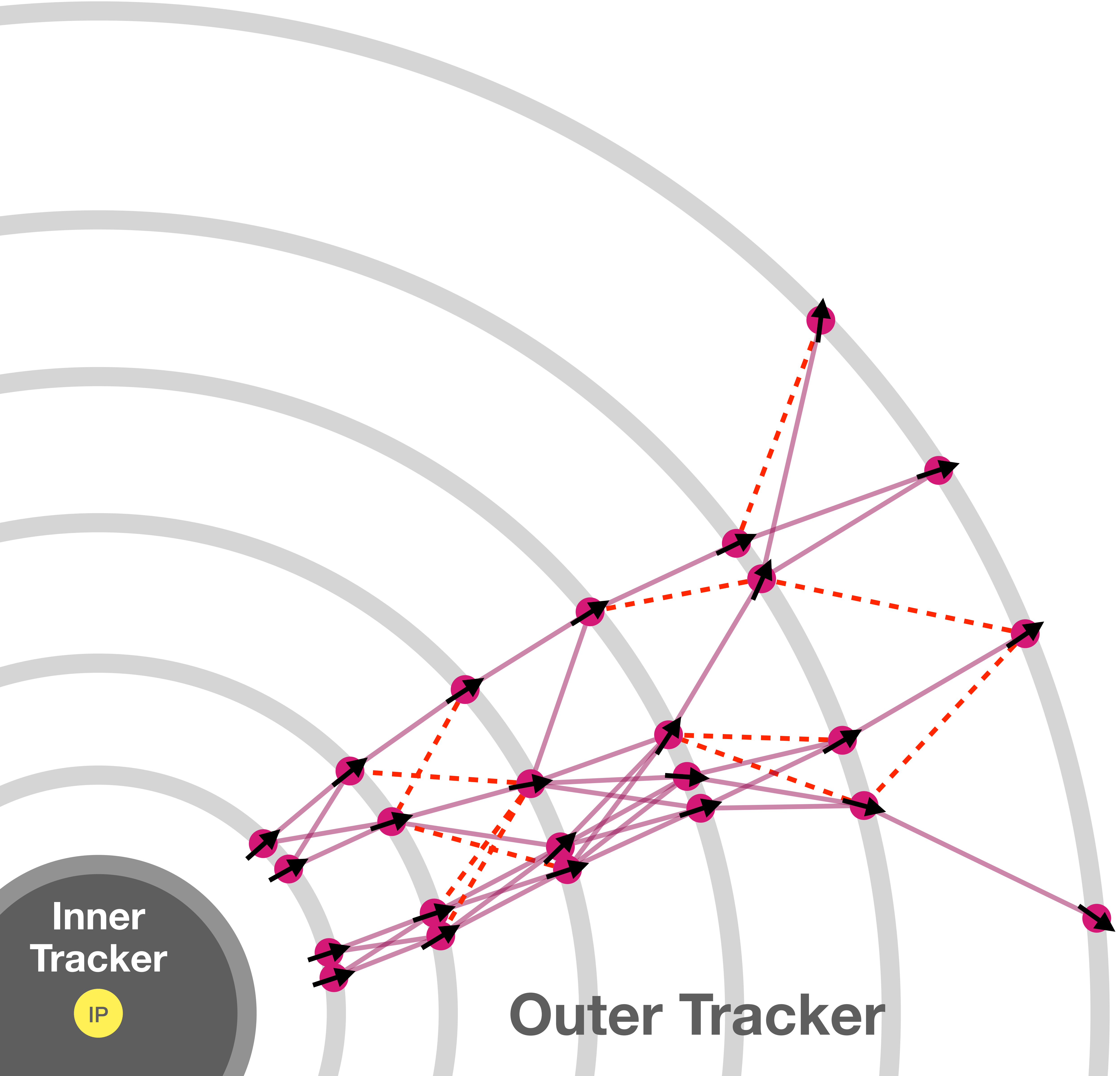
Derived a “**module map**” that
pre-determines valid LSs



One thread per LS



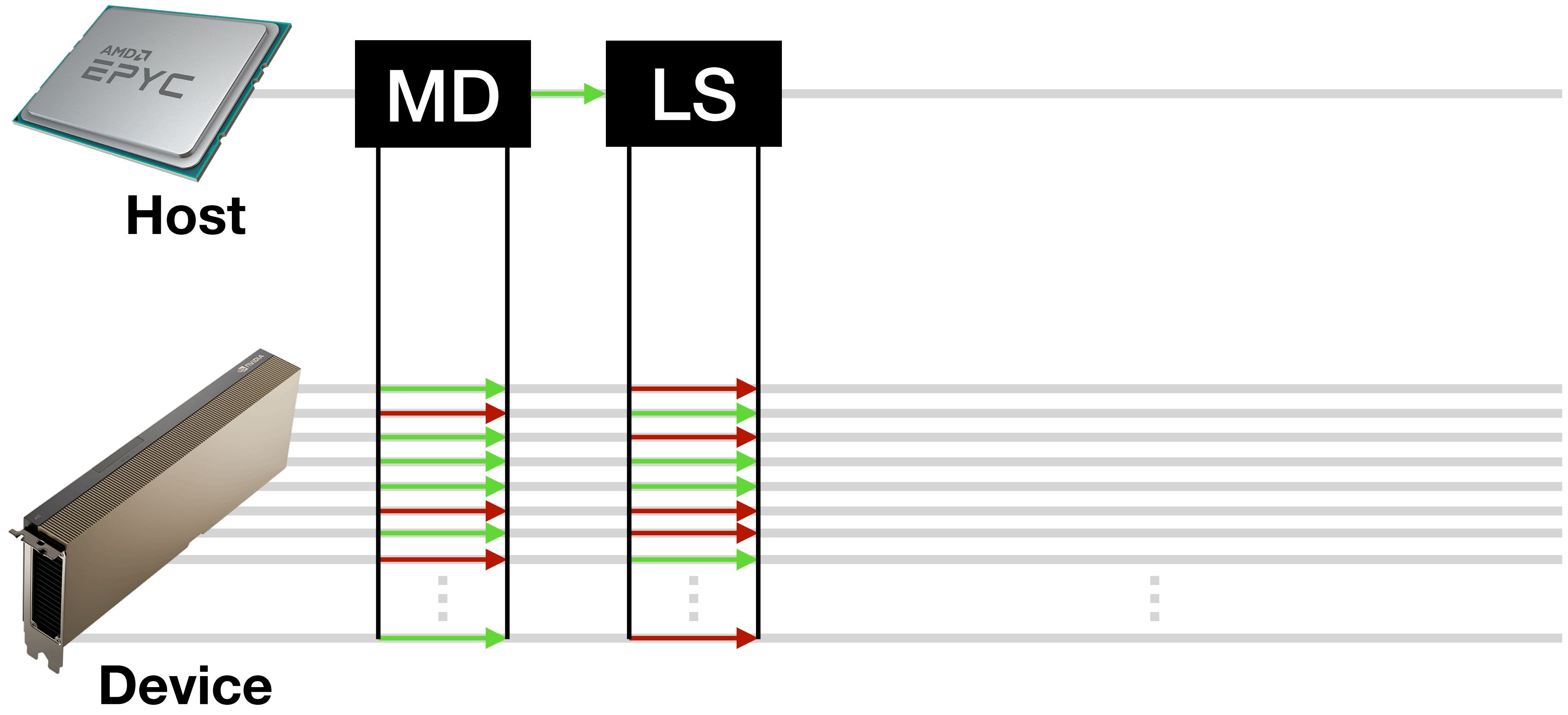
LST in a Nutshell: Line Segments



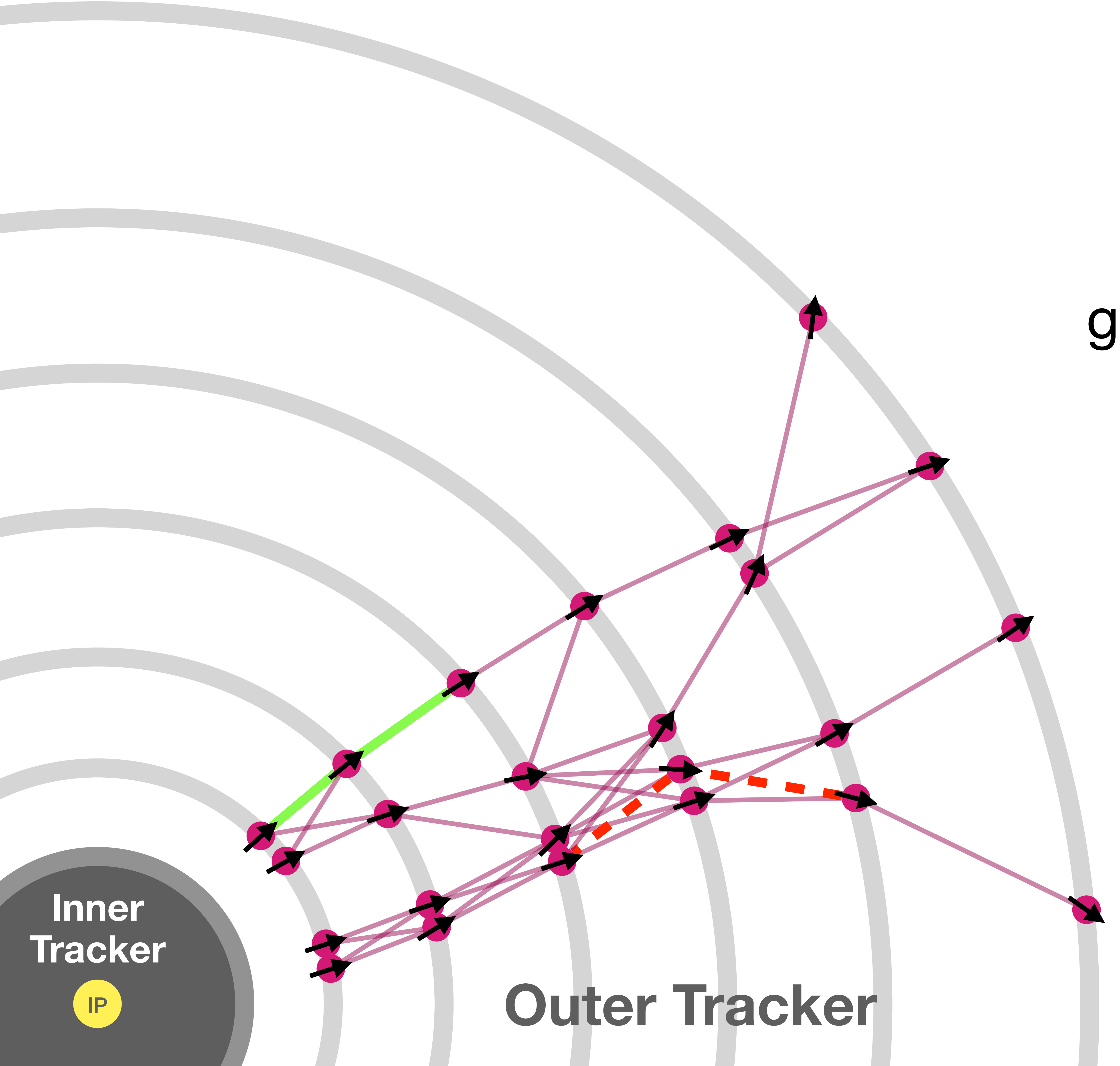
Keep **good** **LSs**

good = **consistency between MD p_T**

One thread per LS



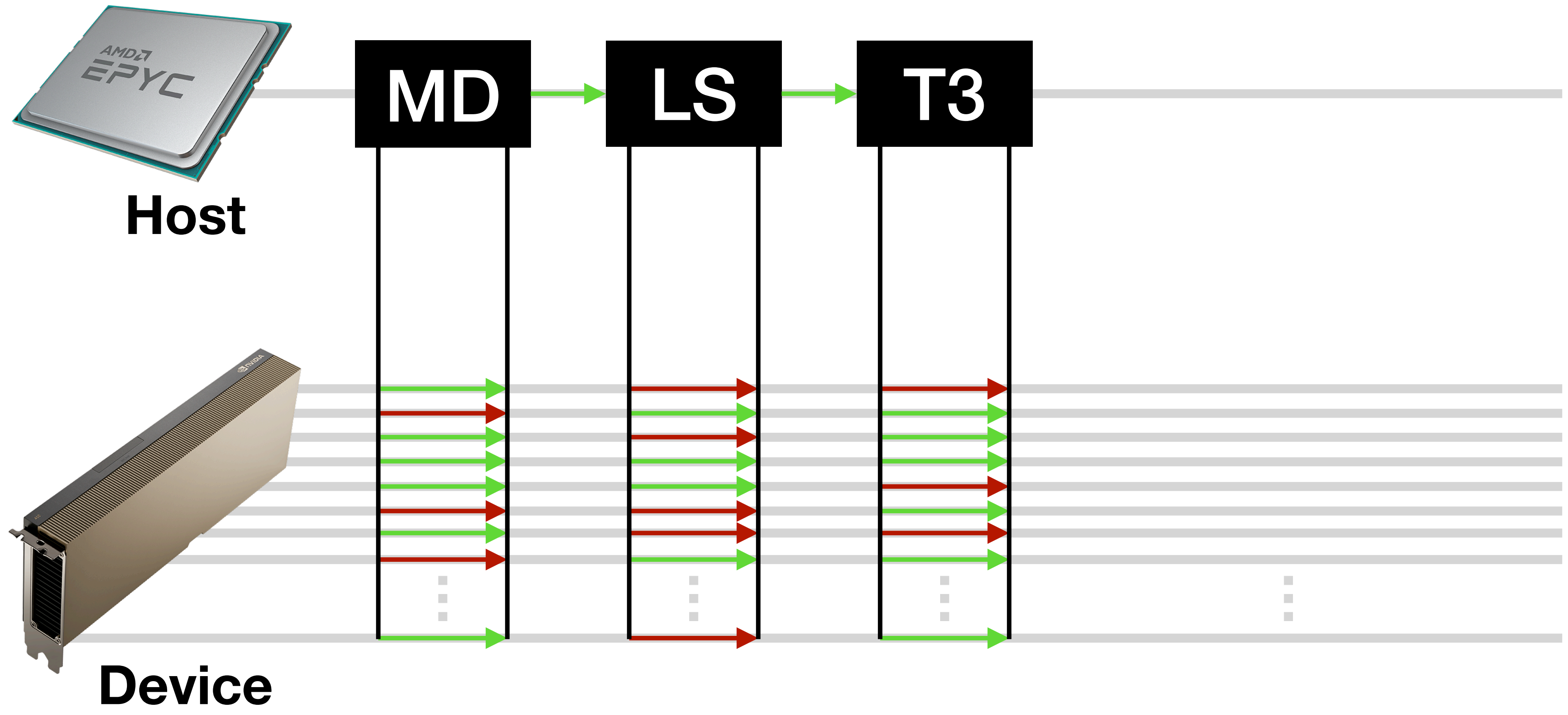
LST in a Nutshell: Triplets



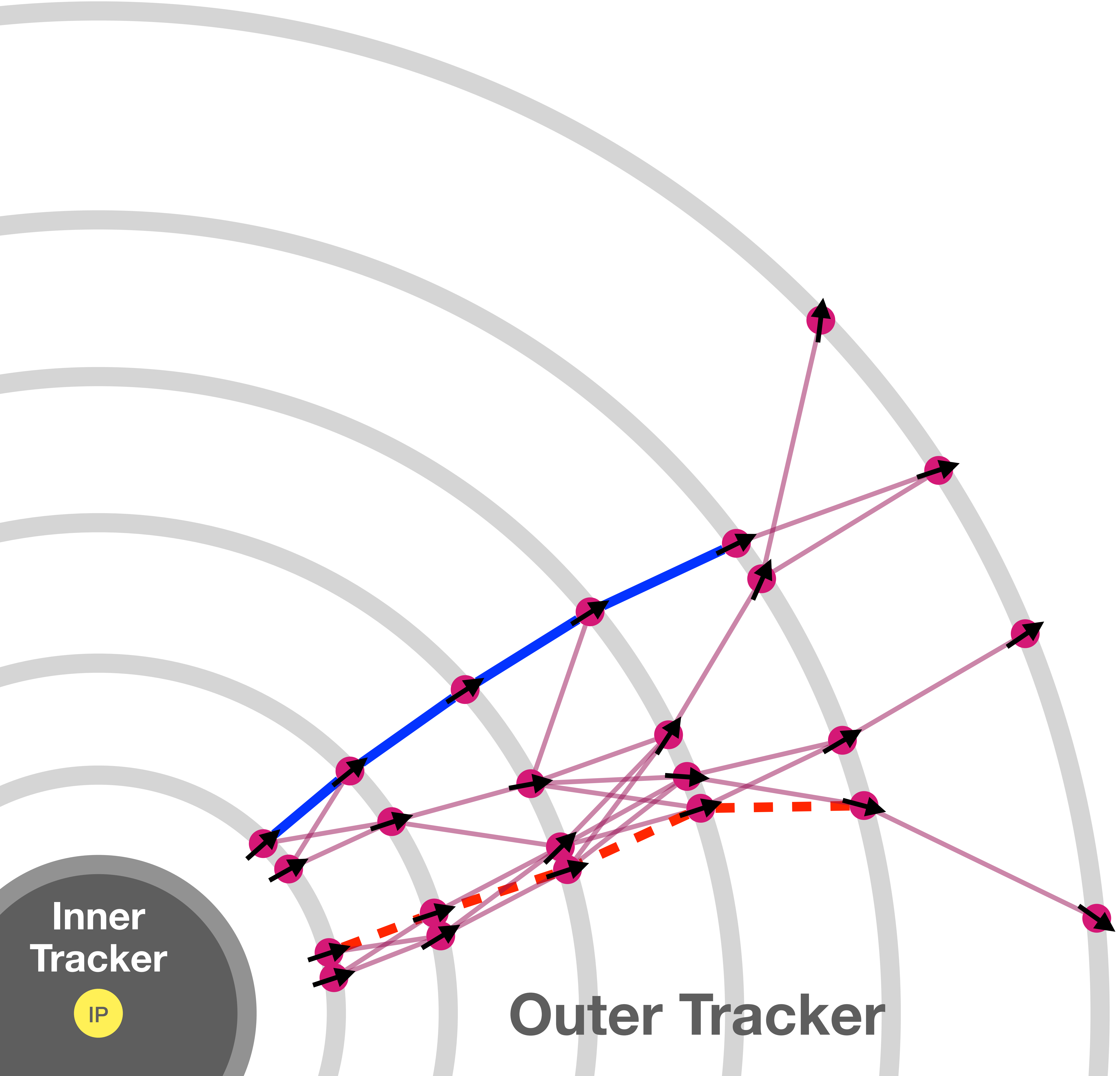
Keep **good** pairs of LSs that share a MD:
i.e. **Triplets (T3s)**

good = **p_T consistency + other constraints**

One thread per T3



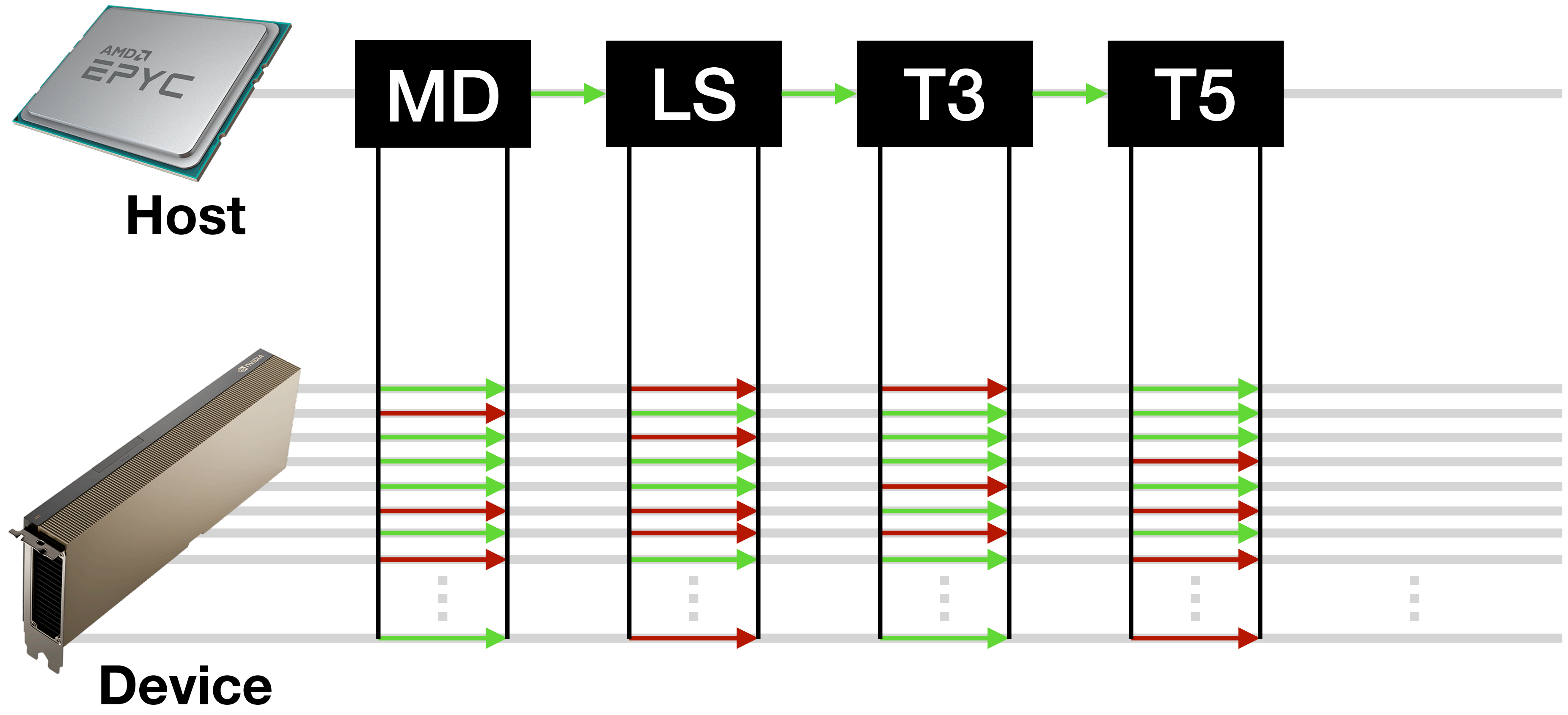
LST in a Nutshell: Quintuplets



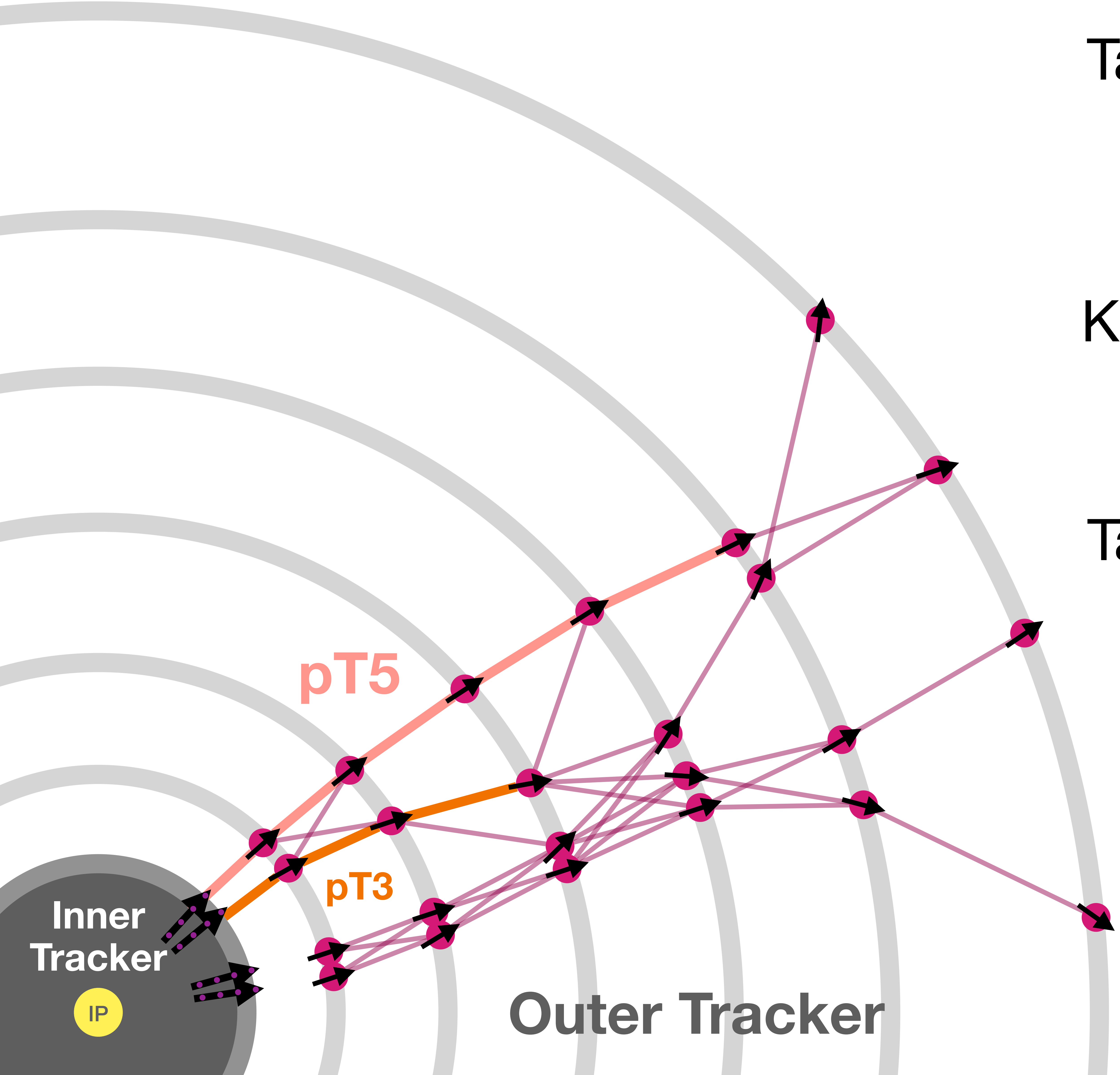
Keep **good** pairs of T3s that share a MD:
i.e. **Quintuplets (T5s)**

good = **p_T consistency + circle fit quality**

One thread per T5



LST in a Nutshell: Track Candidates



Take all **T5s** and match to **pixel seeds (pLS)**: i.e. pT5s

Keep **good** pT5s as **Track Candidates (TCs)**

Take unmatched **T5s**, **good pT3s** (pLS + T3), and unmatched **pLS** also as **TCs**

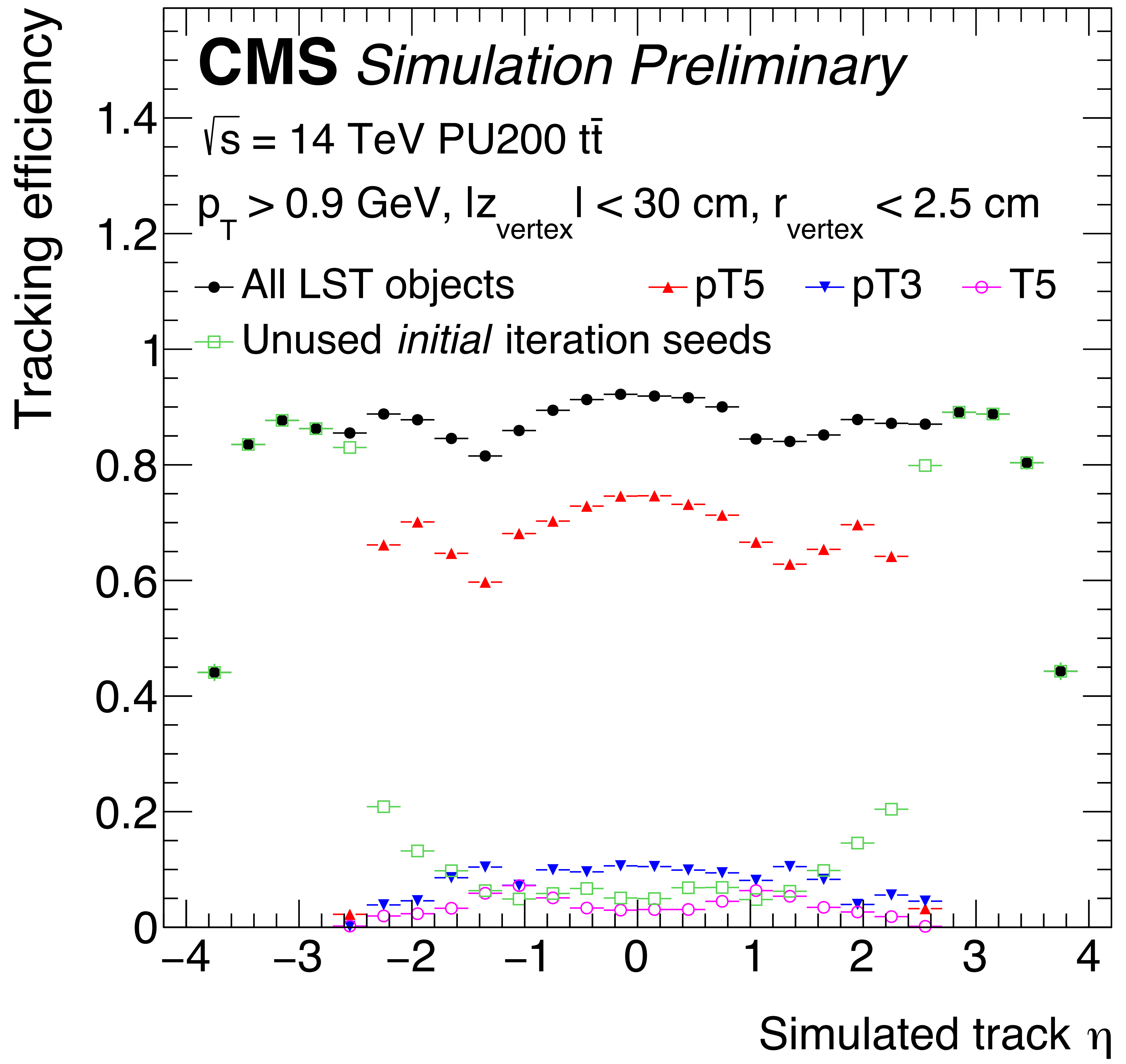
good = **p_T consistency + circle fit quality**

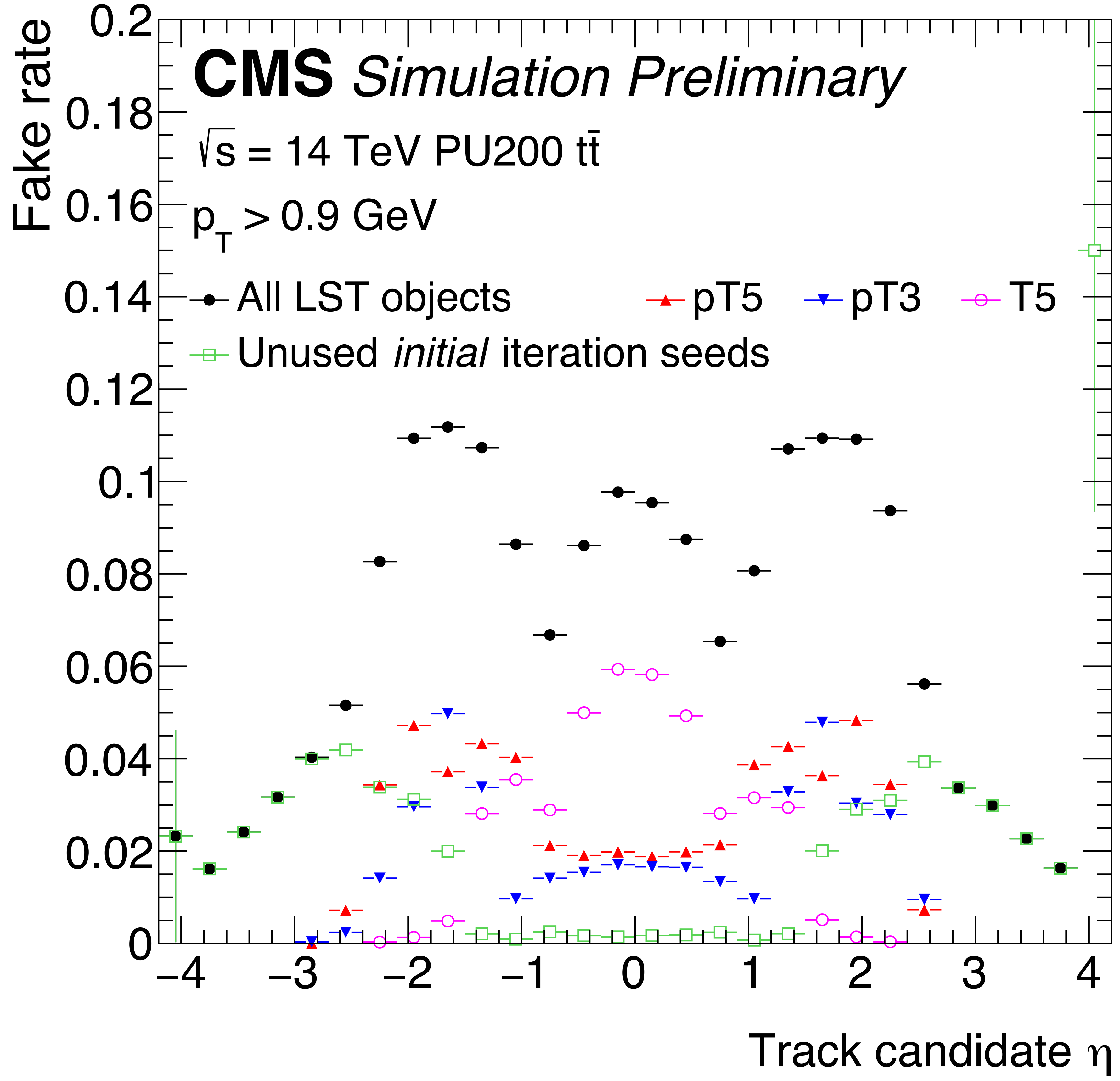
UC San Diego

Improving LST with ML

- Line Segment Tracking (LST) is already highly performant and parallelizable
- Central question: **where can Machine Learning (ML) *realistically* be used to improve LST?**
- In this talk we will…
    - Outline a suitable step in LST to try a **simple ML solution**
    - Show significant **improvements to LST!**
    - Present a prospectus for **more ambitious ML solutions/algorithms**

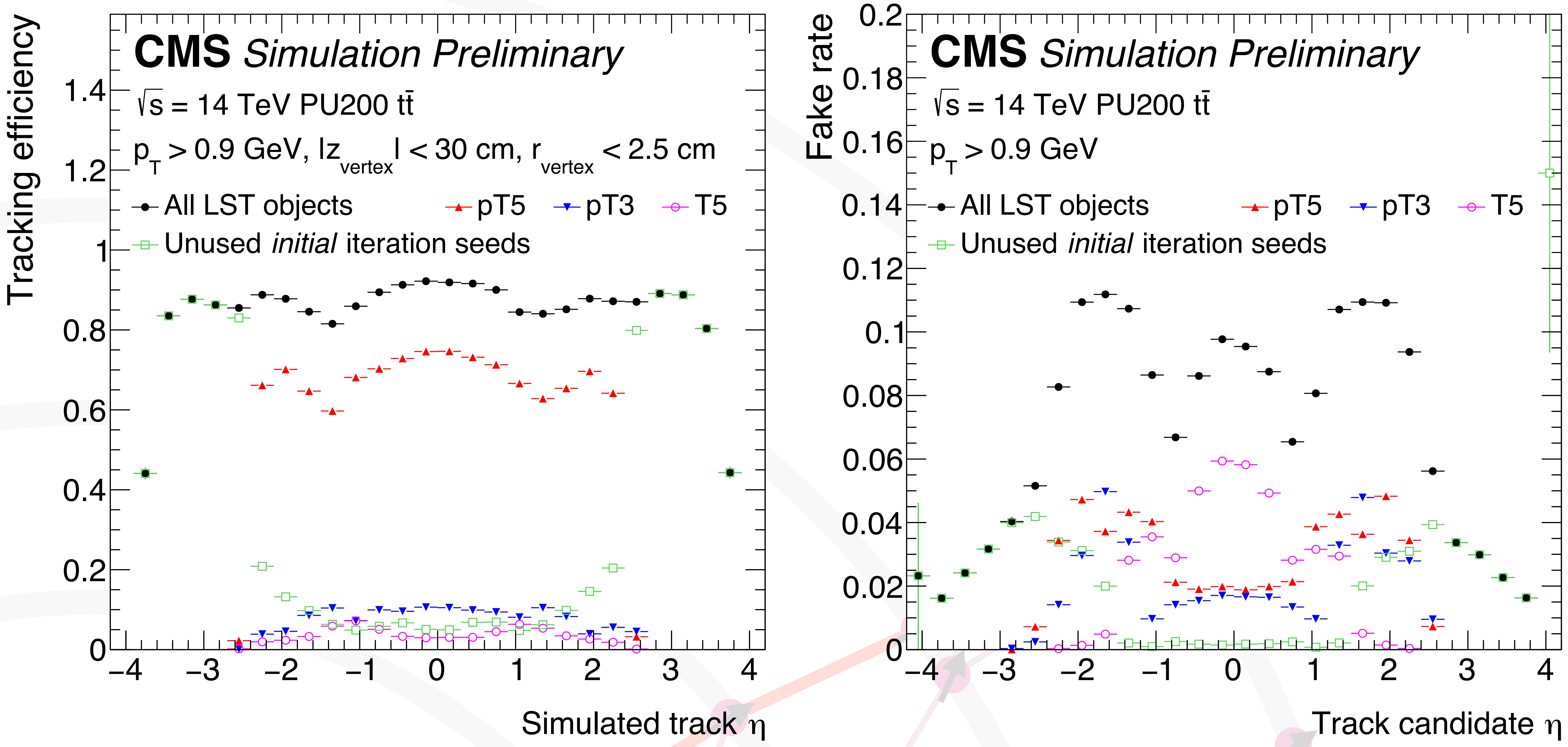
ML Opportunity: Quintuplets





pT5s (pLS + T5) + T5s give most of the TC efficiency
T5s have a high fake rate

ML Opportunity: Quintuplets



pT5s + T5s give most of the TC eff.
T5s have a high fake rate
⇒ there is room for improvement

Can ML do better without heavily impacting the total LST runtime?

Next: we train/deploy a small neural network for classifying real vs. fake **T5s**

UC San Diego

T5 DNN Training Data

Objective
Train DNN to classify
"real" vs. "fake" **T5s**

Real: > 75% of hits are from the same sim track
Fake: not "real"

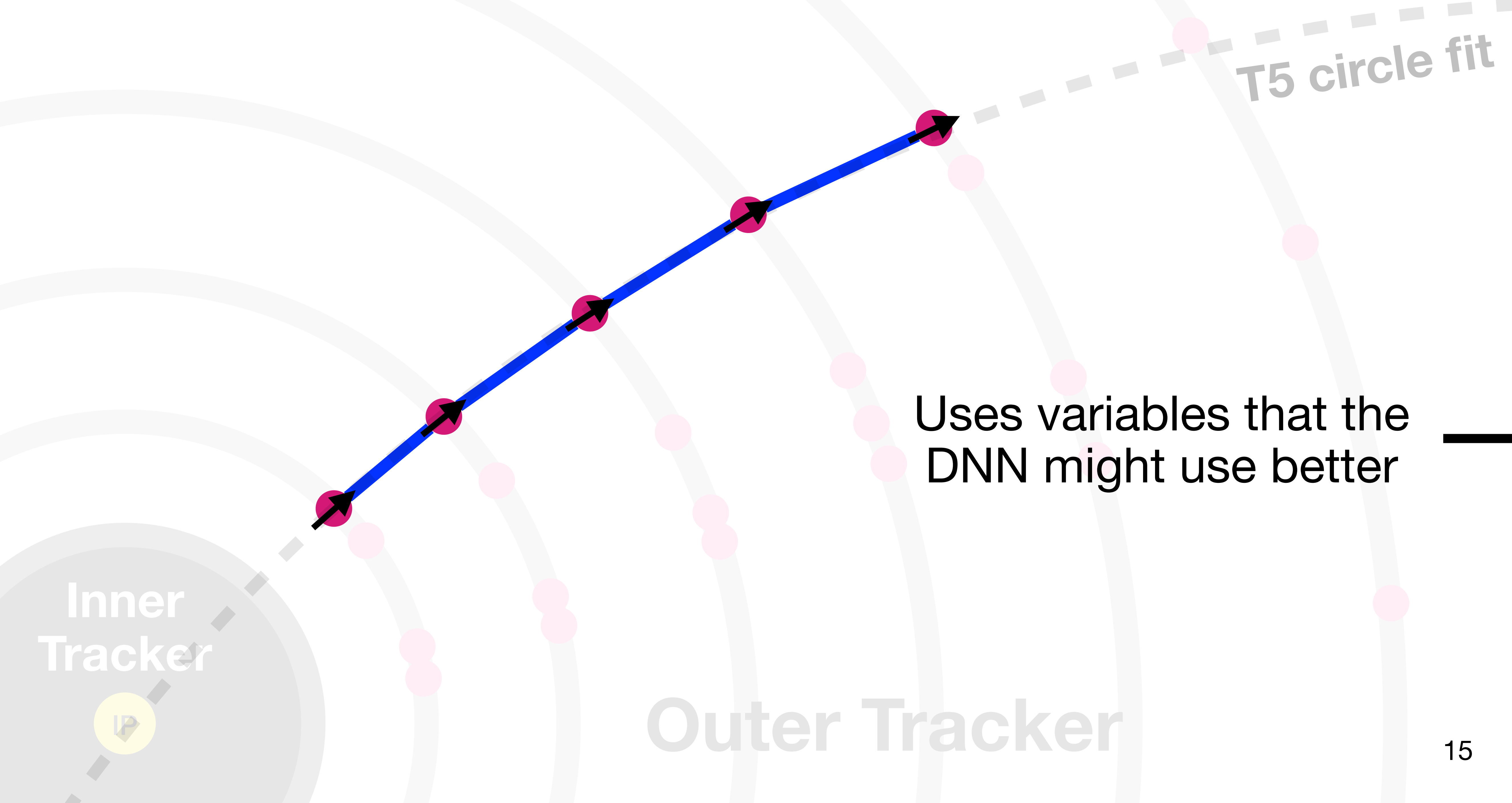
Baseline LST T5s
Pass basic quality cuts
Pass r-z χ^2 cut
Pass r-ϕ χ^2 cut
Quality of circle fit

Remove r-ϕ χ^2 cut

DNN Training T5s
Pass basic quality cuts
Pass r-z χ^2 cut
~~Pass r-ϕ χ^2 cut~~

Uses variables that the DNN might use better

~2.1 million T5s
40% real



UC San Diego

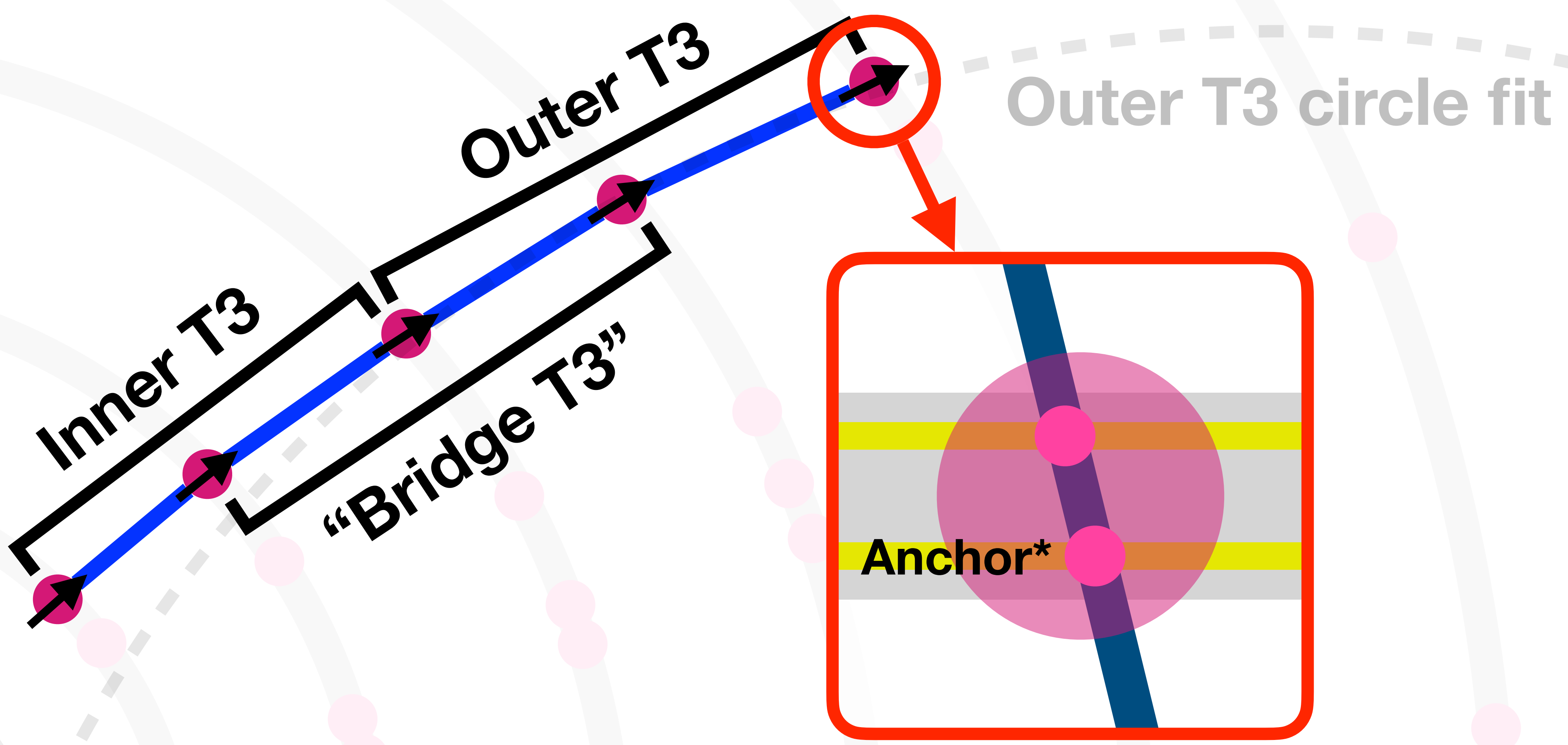
T5 DNN Features and Configuration

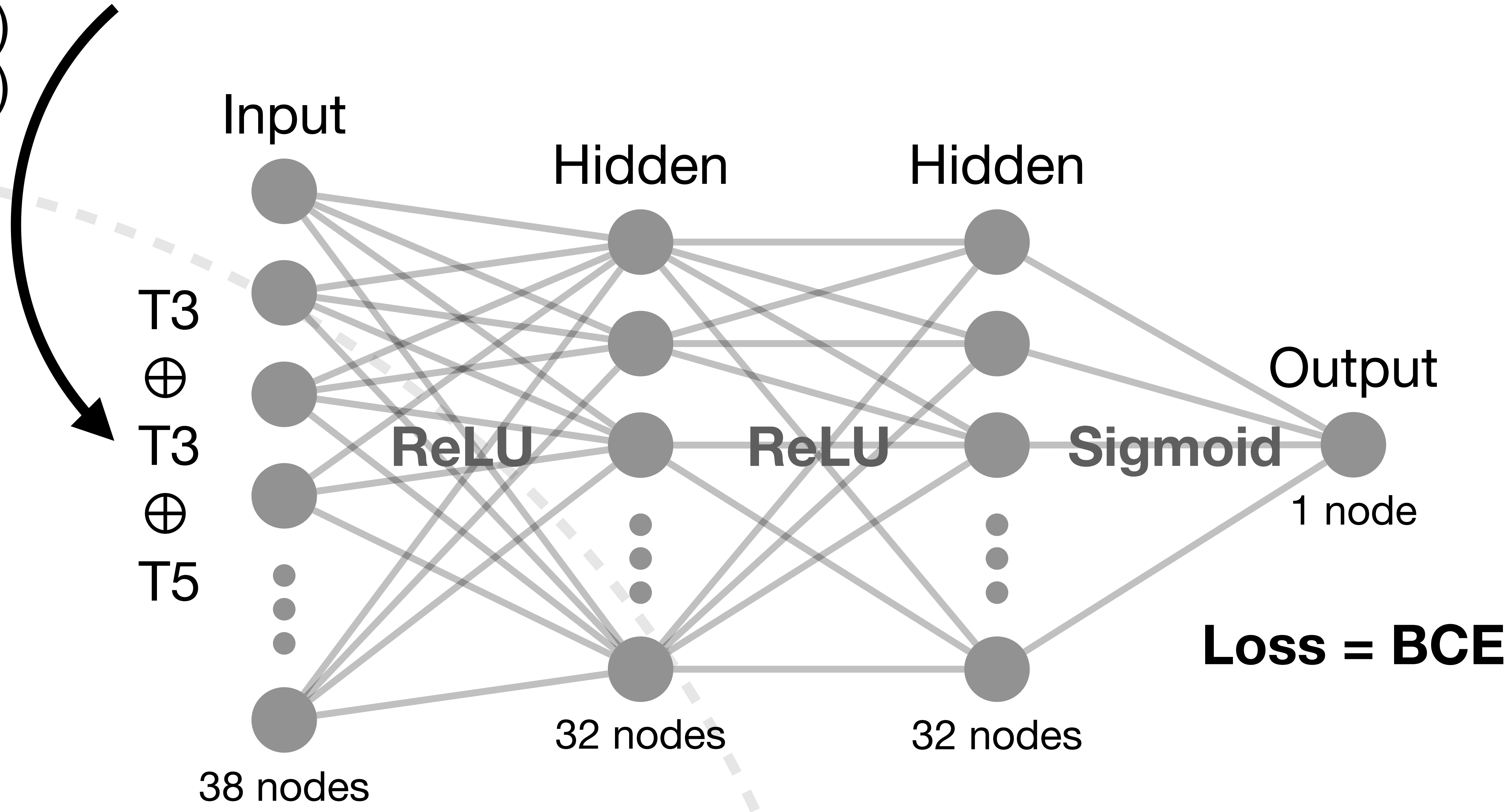
Objective
Train DNN to classify
"real" vs. "fake" **T5s**

Real: > 75% of hits are from the same sim track
Fake: not "real"

Object	Feature
T3 (x2)	p_T
	Inner anchor hit r, z, ϕ, η, layer
	Middle anchor hit r, z, ϕ, η, layer
	Outer anchor hit r, z, ϕ, η, layer
	Radius of circle fit
T5 candidate	p_T, η, ϕ
	Radius of circle fit for Bridge "T3"

$p_T \rightarrow \log_{10}(p_T)$
radius → $\log_{10}$(radius)

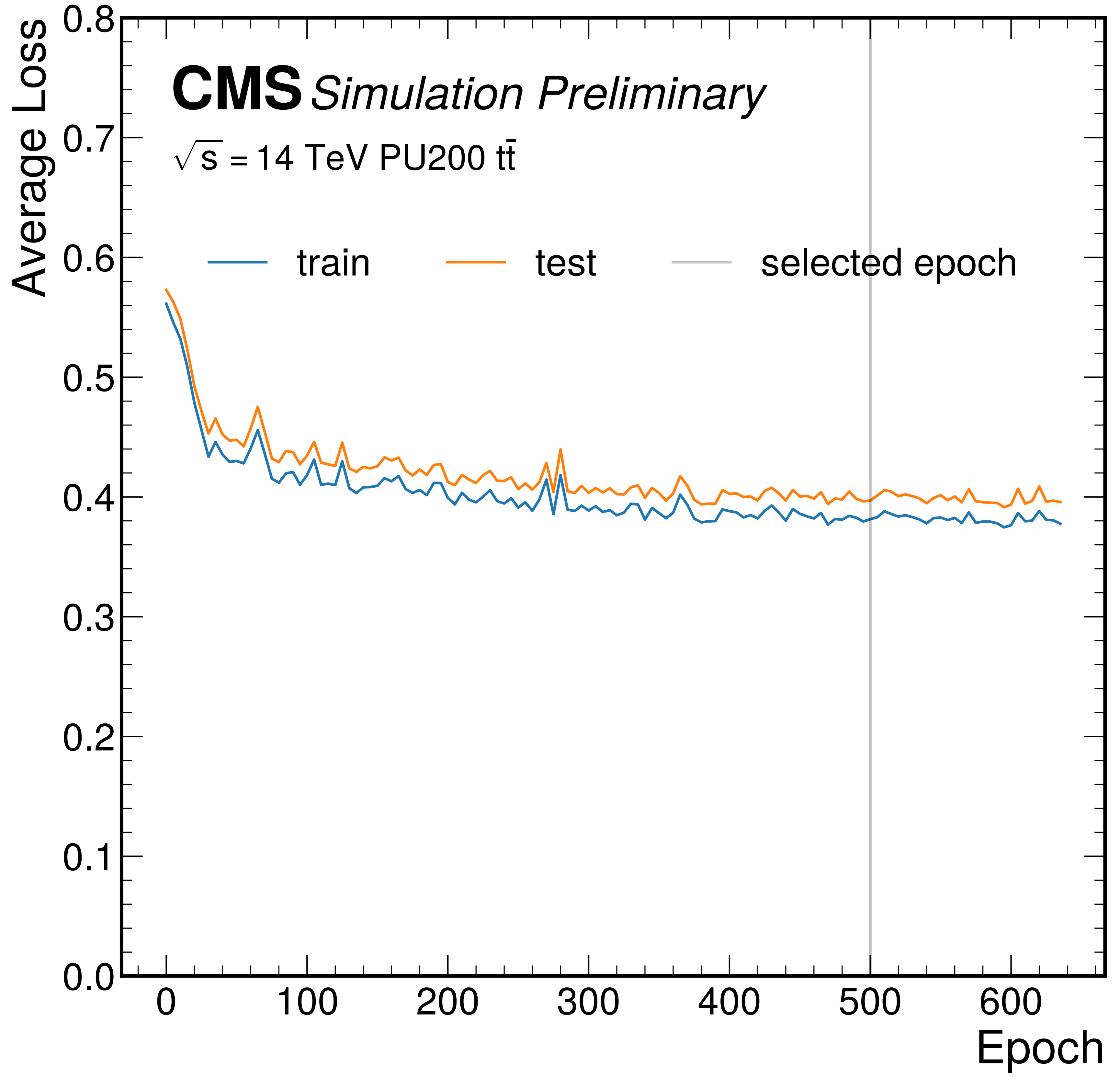


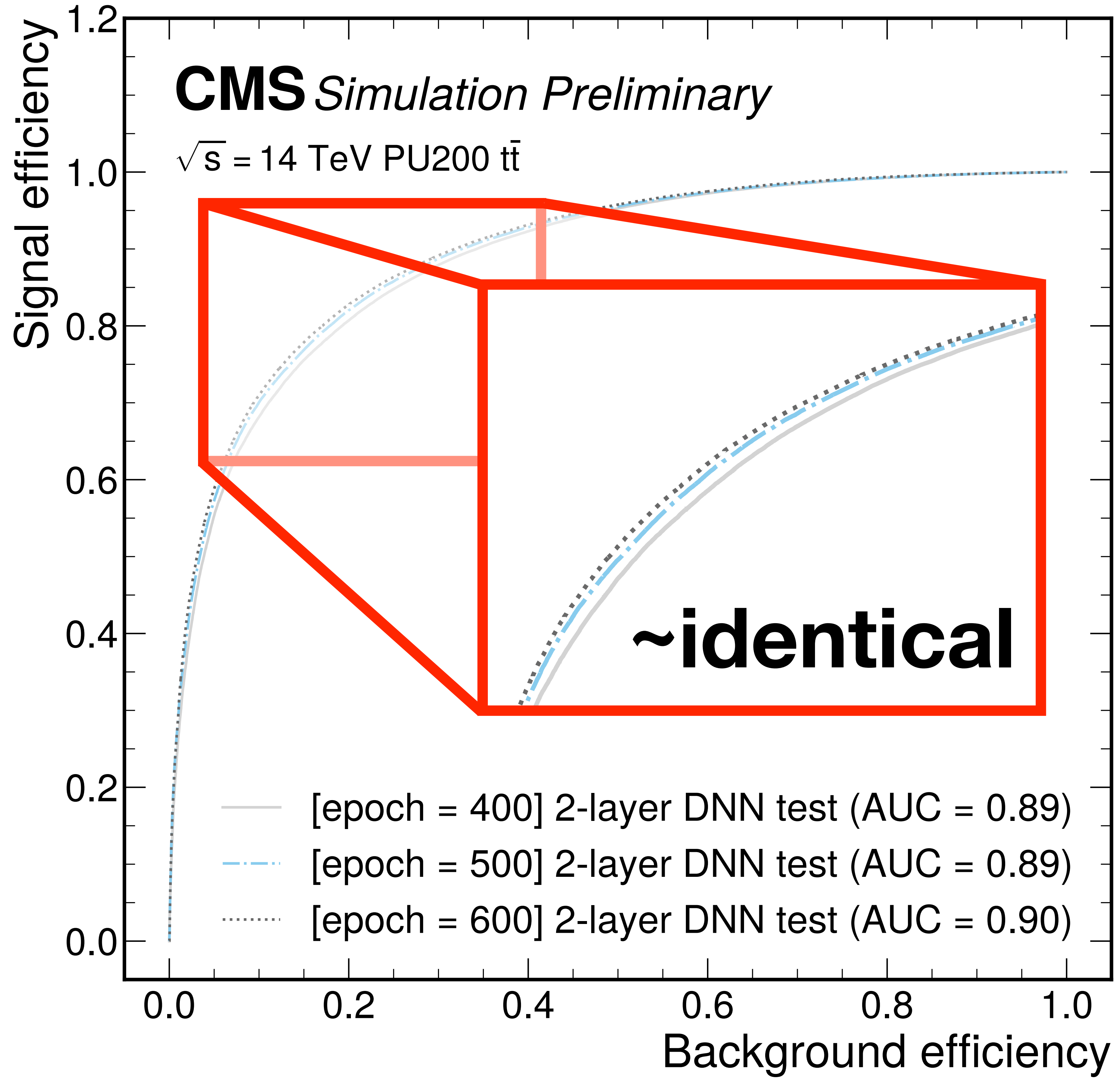


*Not always the inner hit: for PS modules, it is always the pixel (P) hit, which is not necessarily the hit that is closest to the beamline

UC San Diego

T5 DNN Training and Selection



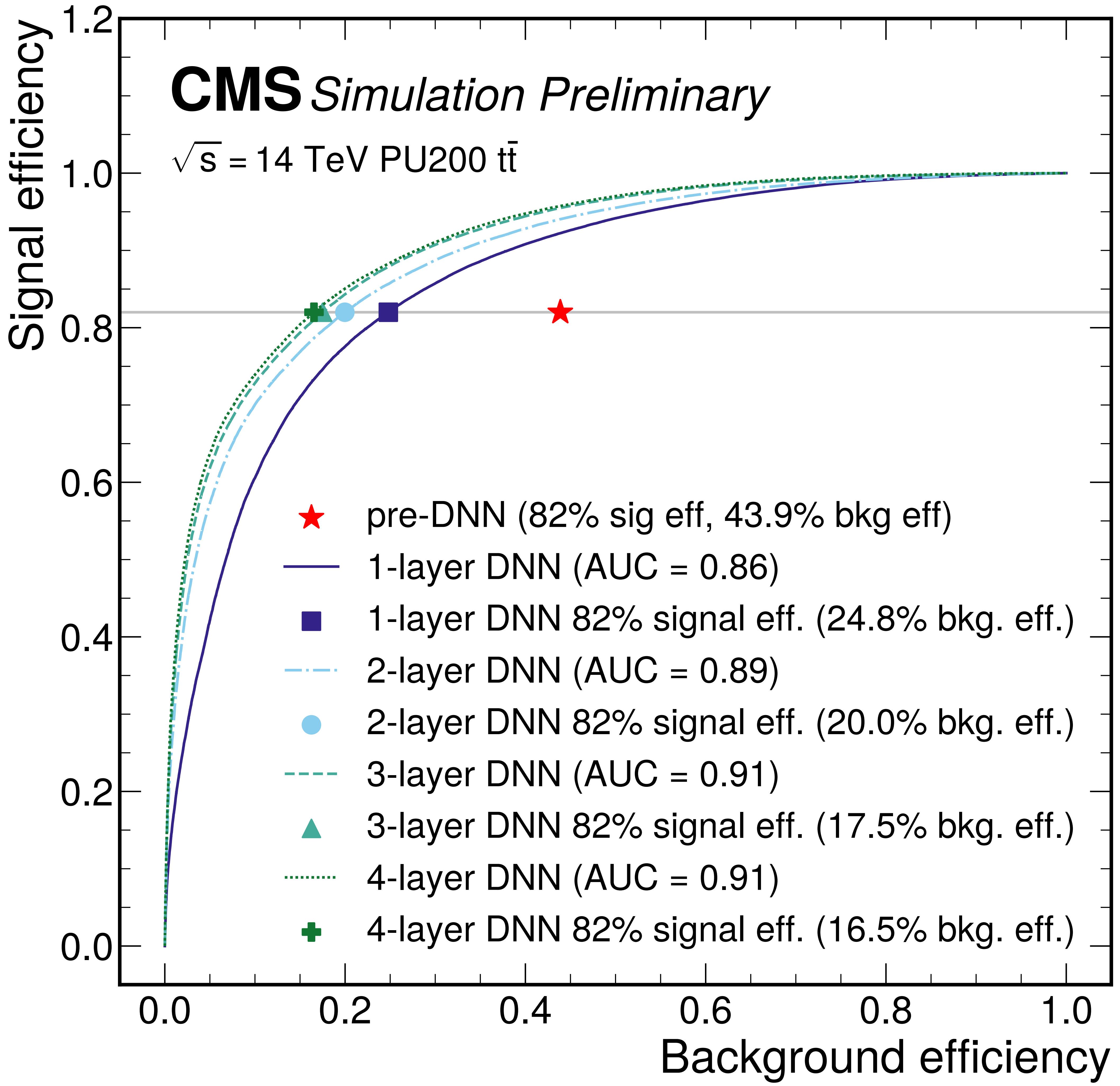


Choice of epoch is arbitrary after 400 (we choose 500)

UC San Diego

Aside: Other T5 DNN Architectures

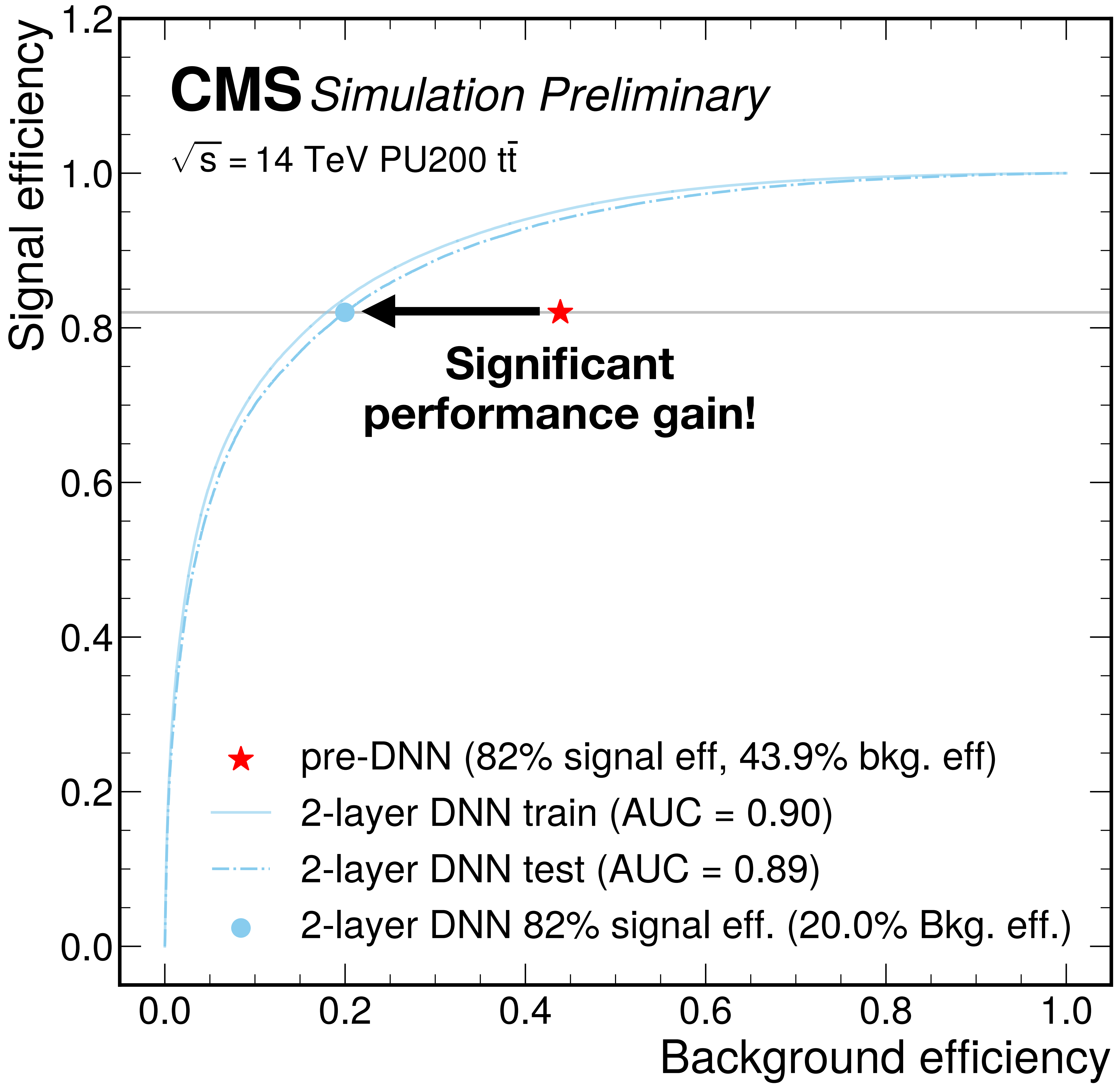
- Additional layers (up to 4) gives an extra ~3% decrease in bkg. efficiency (i.e. fake rate)
- Selected 2-layer DNN as a balance between performance and computational complexity
- A larger DNN could be used, but seems to be diminishing returns



UC San Diego

T5 DNN Performance

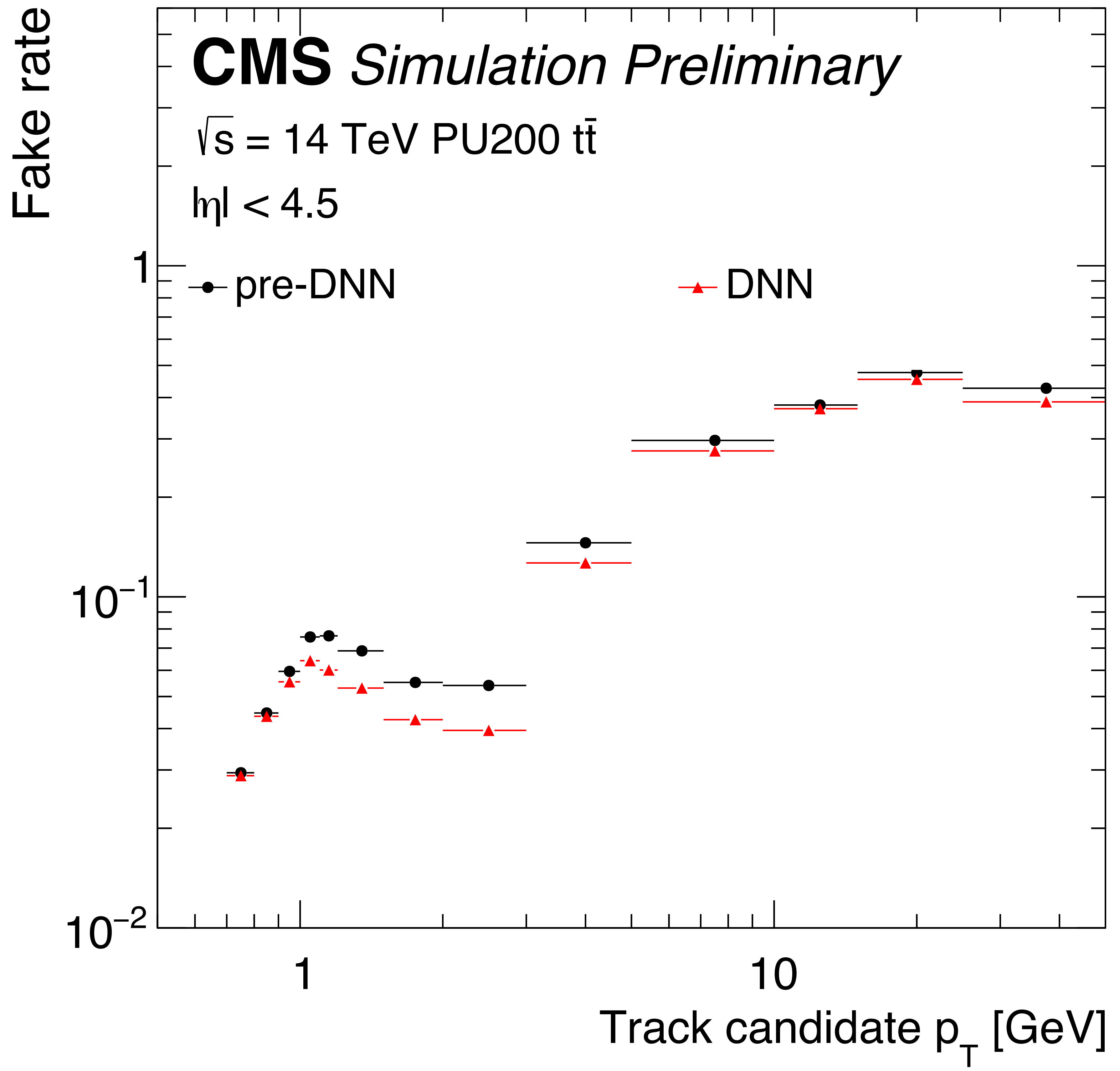
- Performance gain over pre-DNN baseline (★)
  - Background efficiency = Fake rate
- **Cannot get full picture of DNN in LST from this ROC curve alone:**
  - Only looking at T5s, but the TCs are more diverse
  - Effects of duplicate removal, TC selection, etc. are non-trivial
- **Next:** implement DNN inference in LST and compare to pre-DNN performance
  - Details of implementation are in the backup

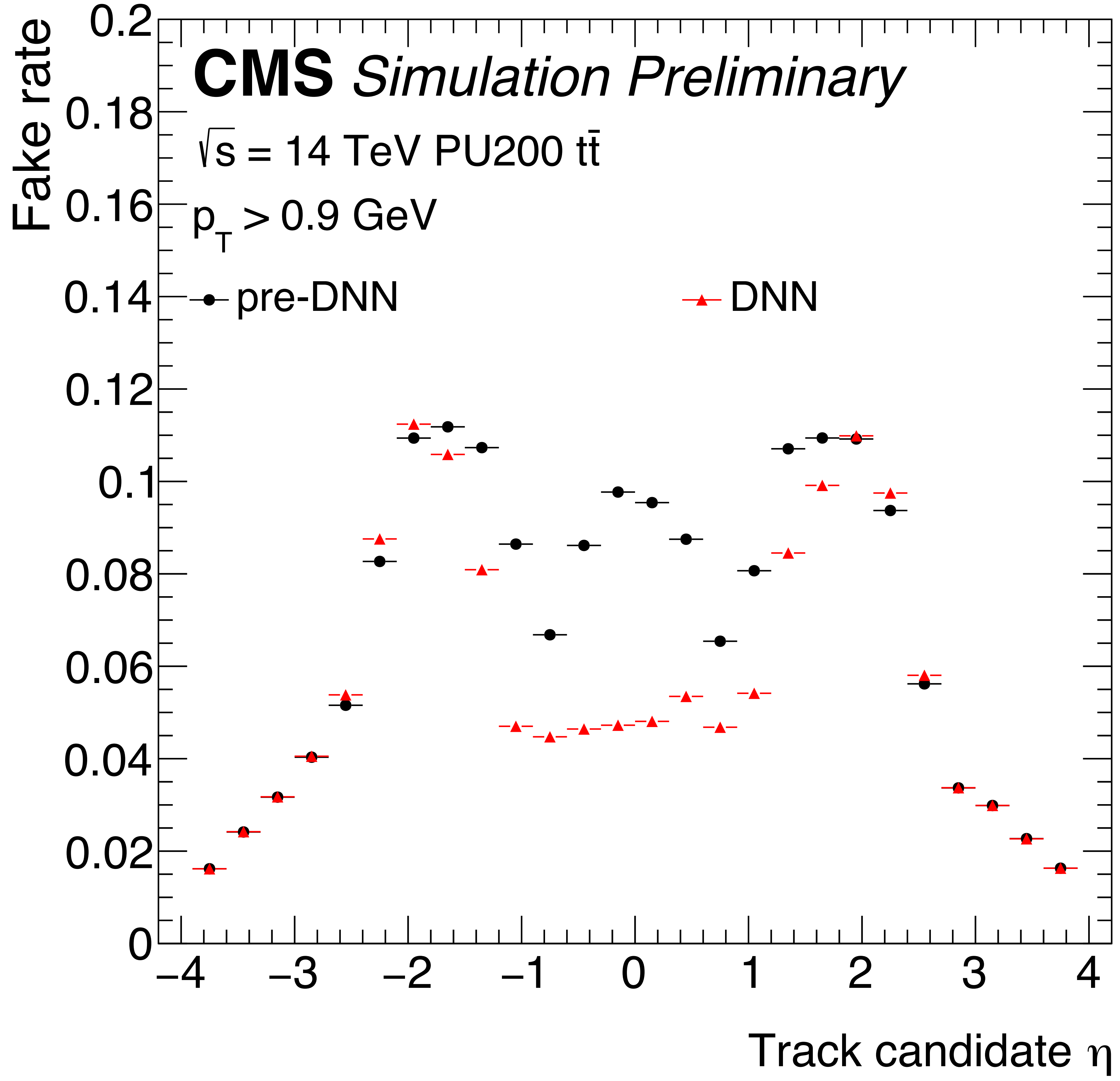


UC San Diego

Standalone

Fake Rate Comparison



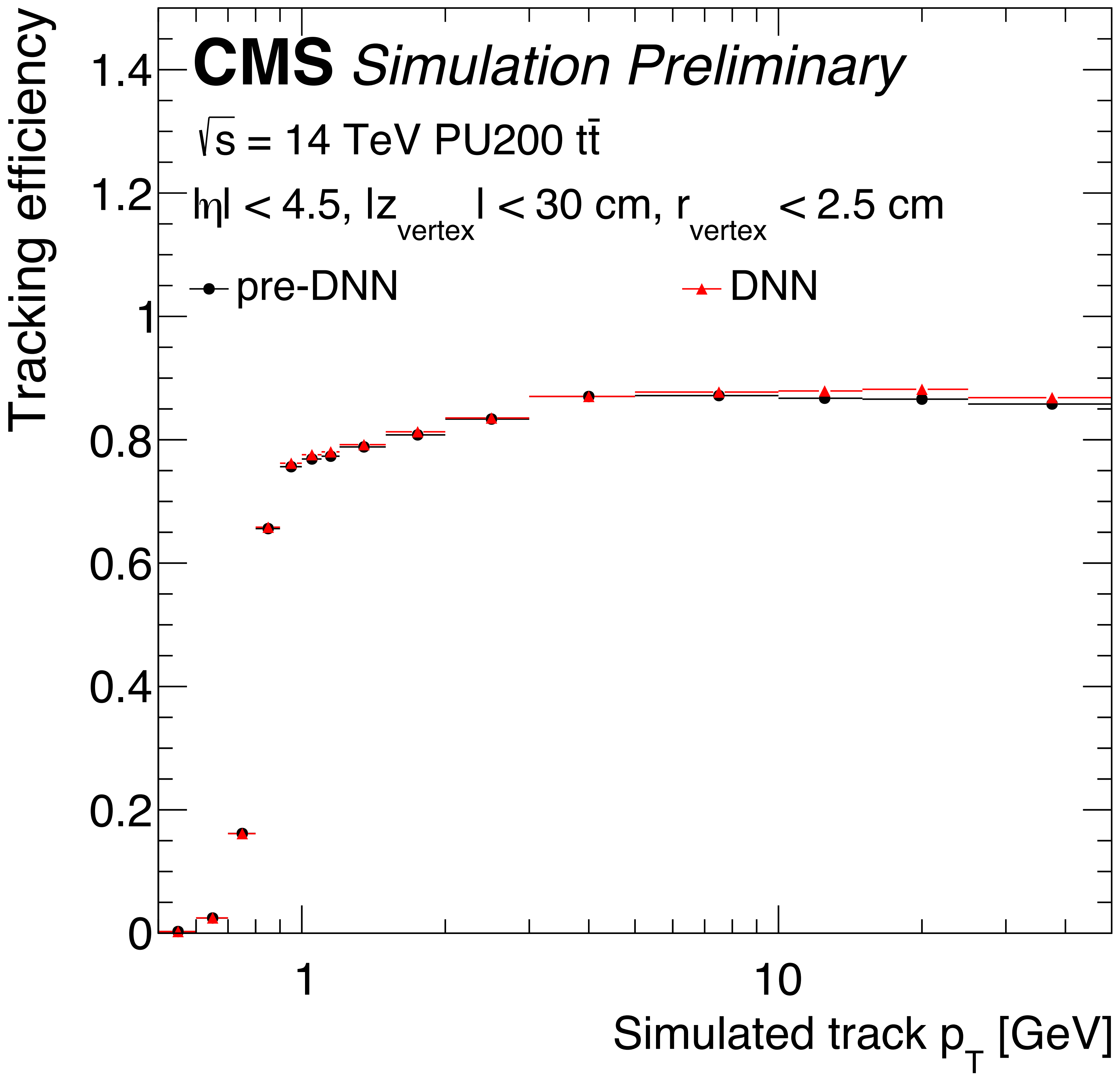


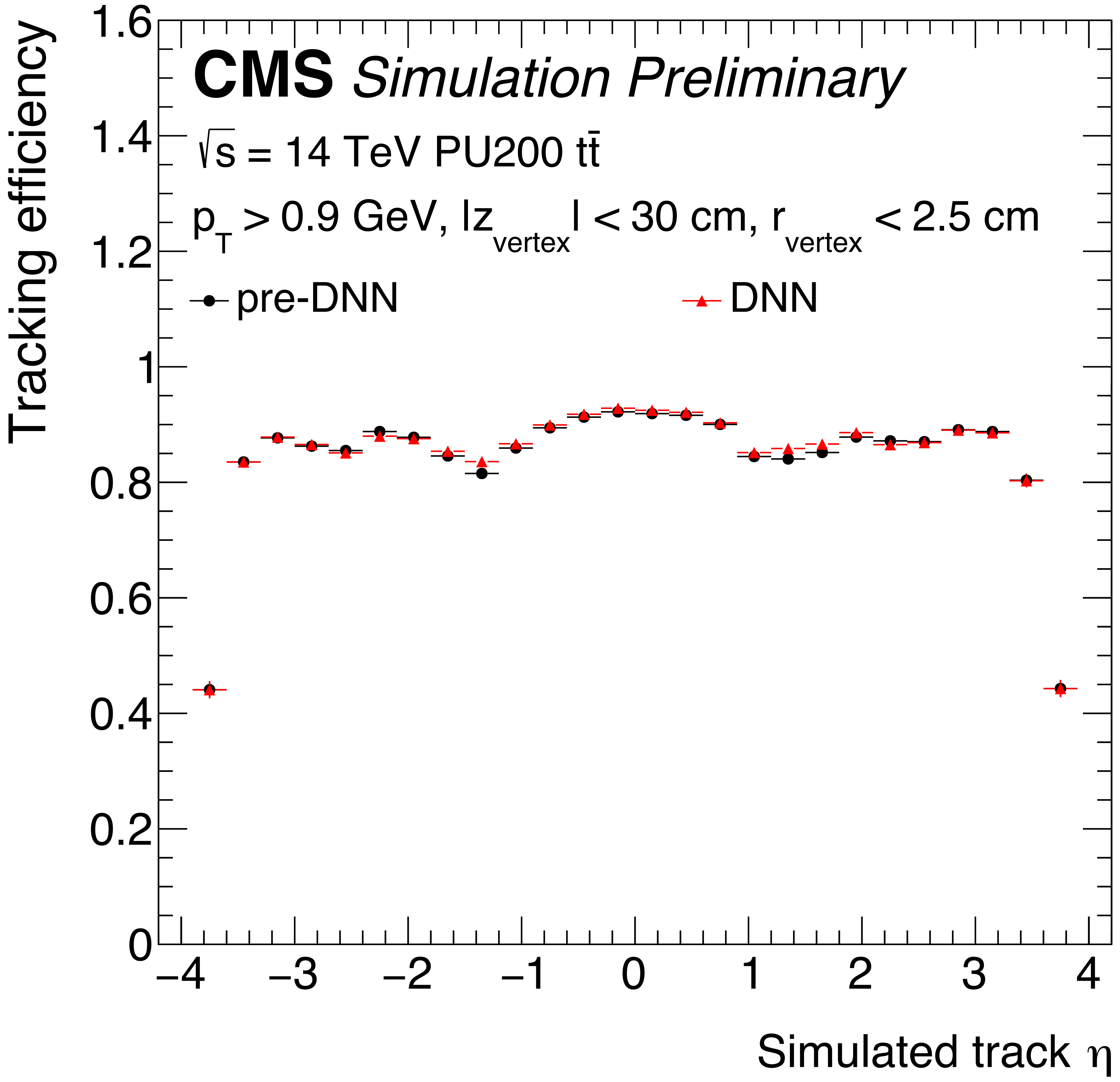
DNN gives ~40% reduction of fake rate in the barrel

UC San Diego

Standalone

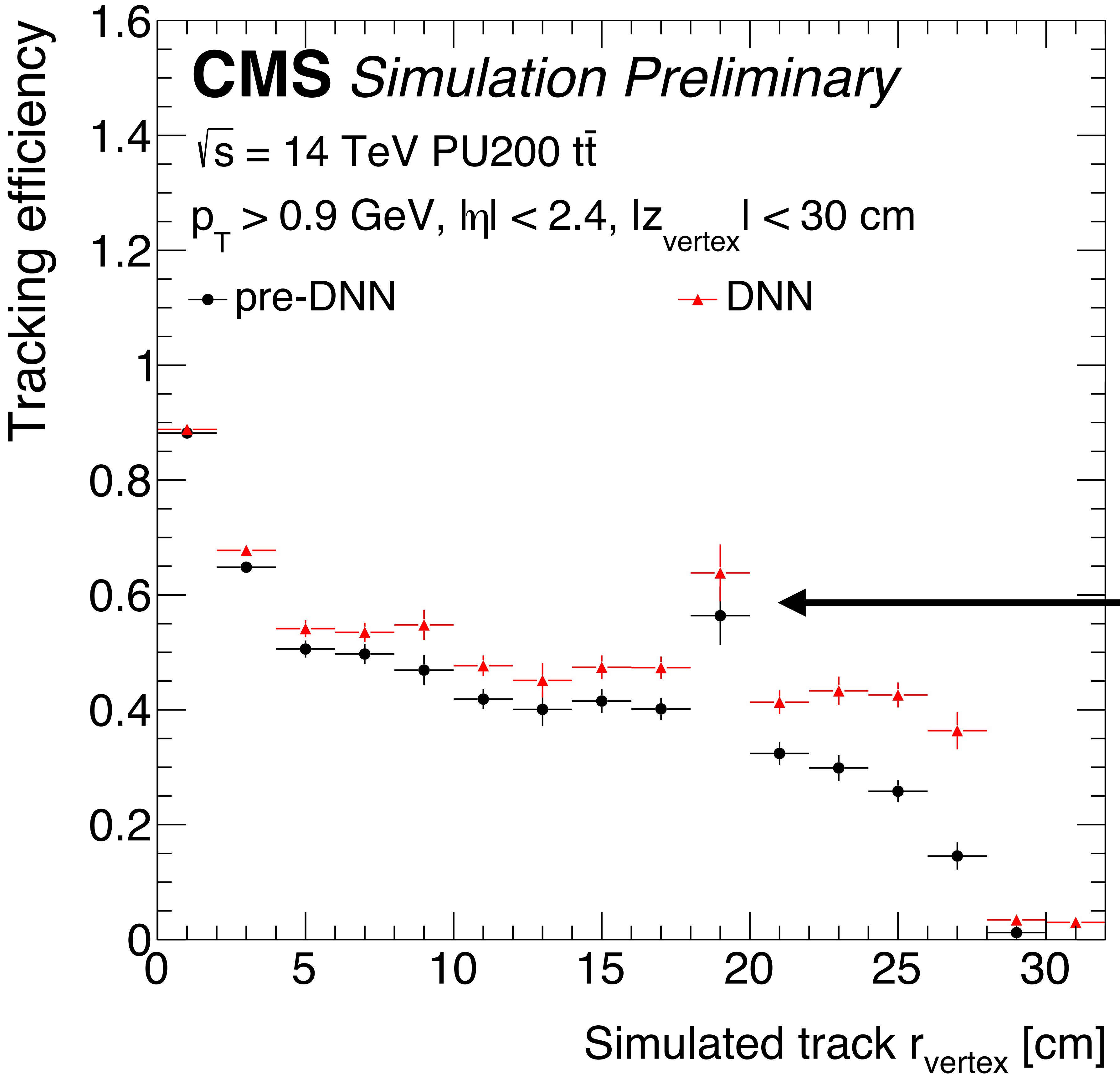
Efficiency Comparison





DNN gives no loss in efficiency
Using DNN WP that matches LST signal efficiency

Efficiency vs. r_{vertex} Comparison

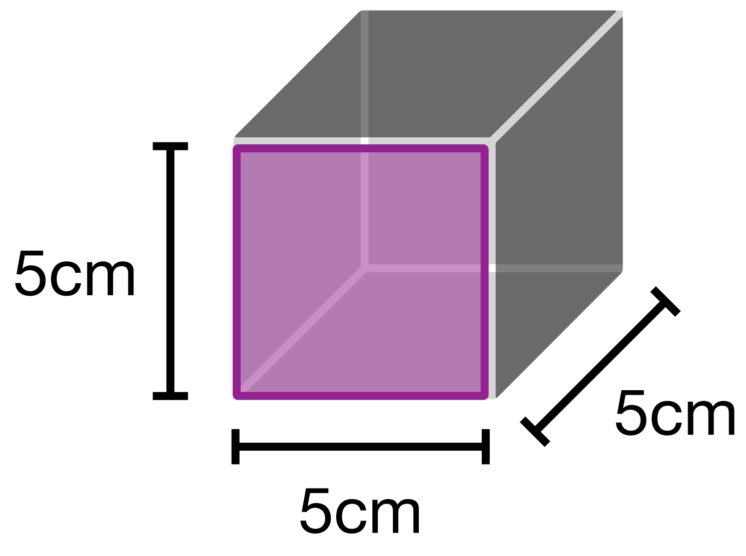


Contributions from material interactions are lower than neighboring bins due to the detector geometry

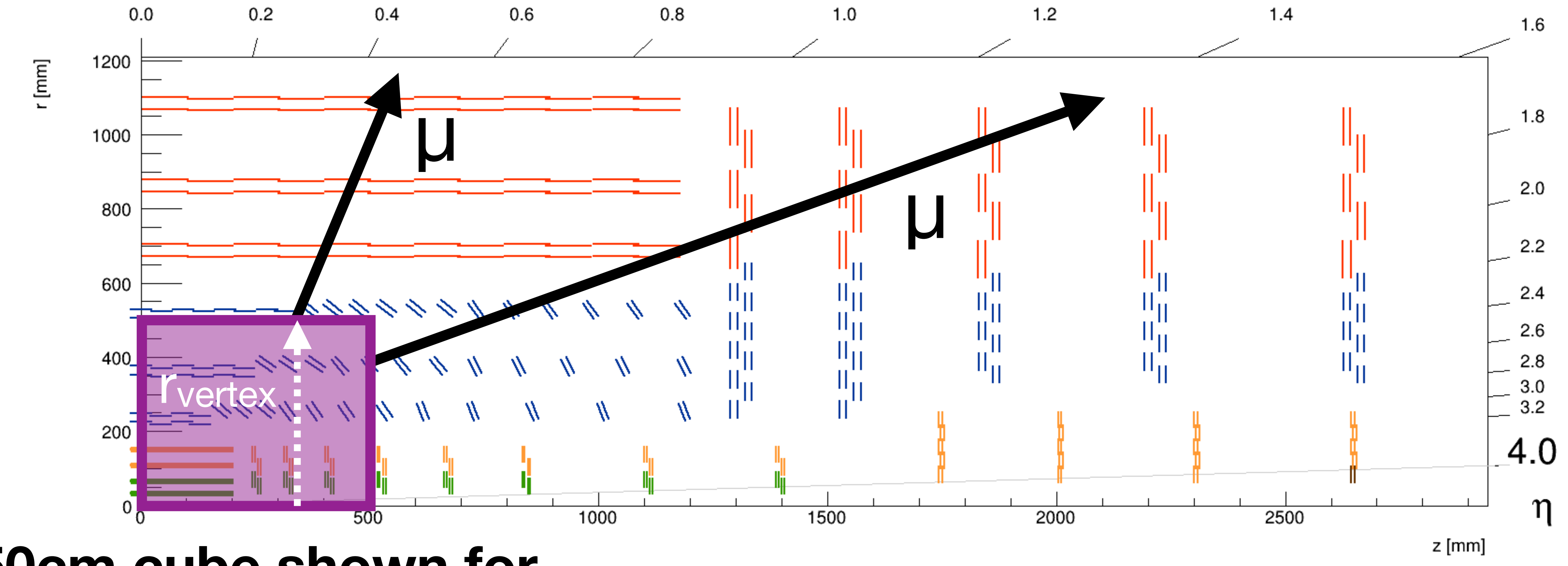
Significant gain in efficiency for displaced tracks

Standalone

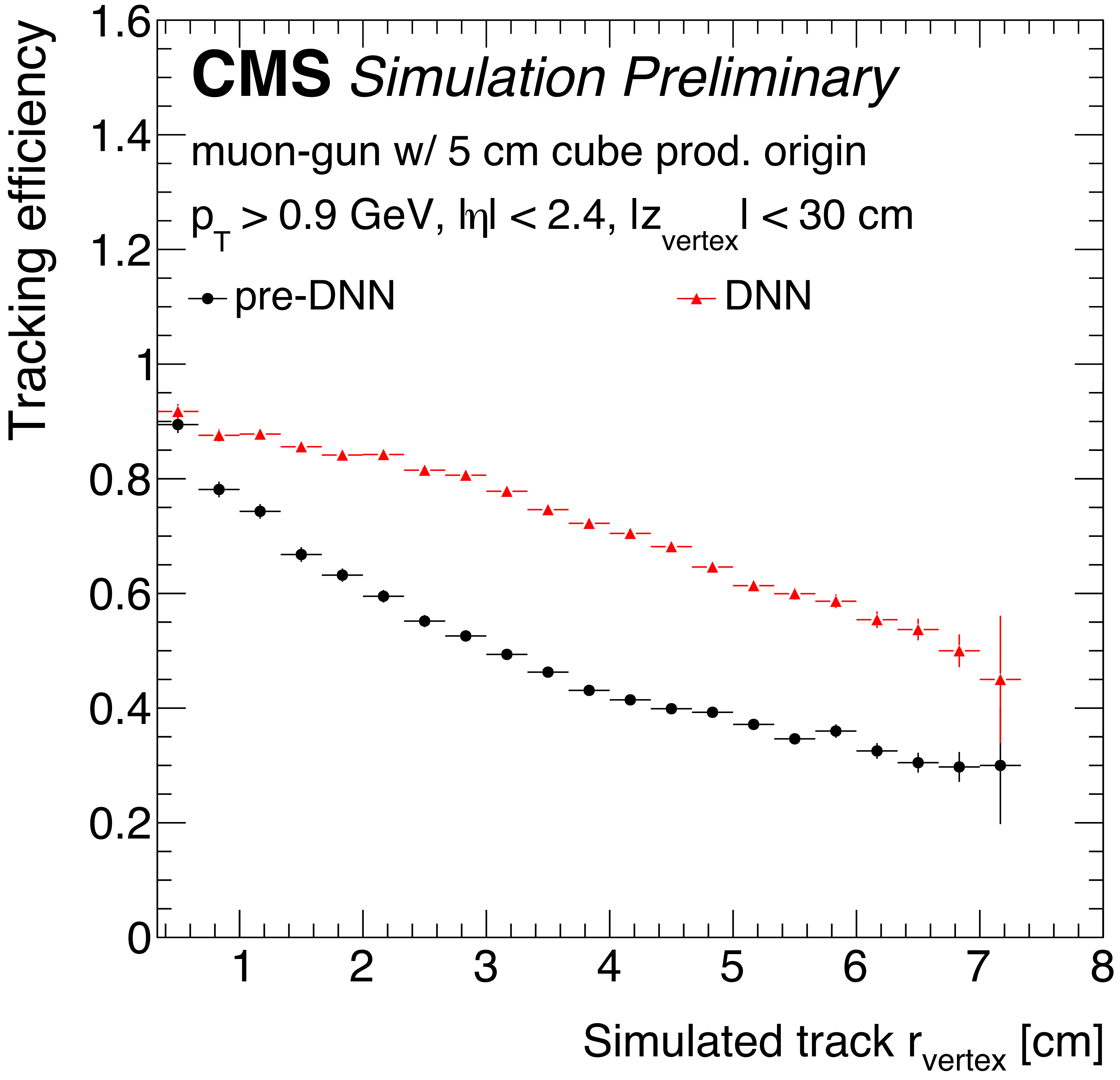
Efficiency Comparison: Muon Cube



Muon cube sample:
Production vertex uniformly distributed across 5cm cube (no pileup),
i.e. displaced tracks



50cm cube shown for visualization purposes



Significant recovery of efficiency for displaced tracks

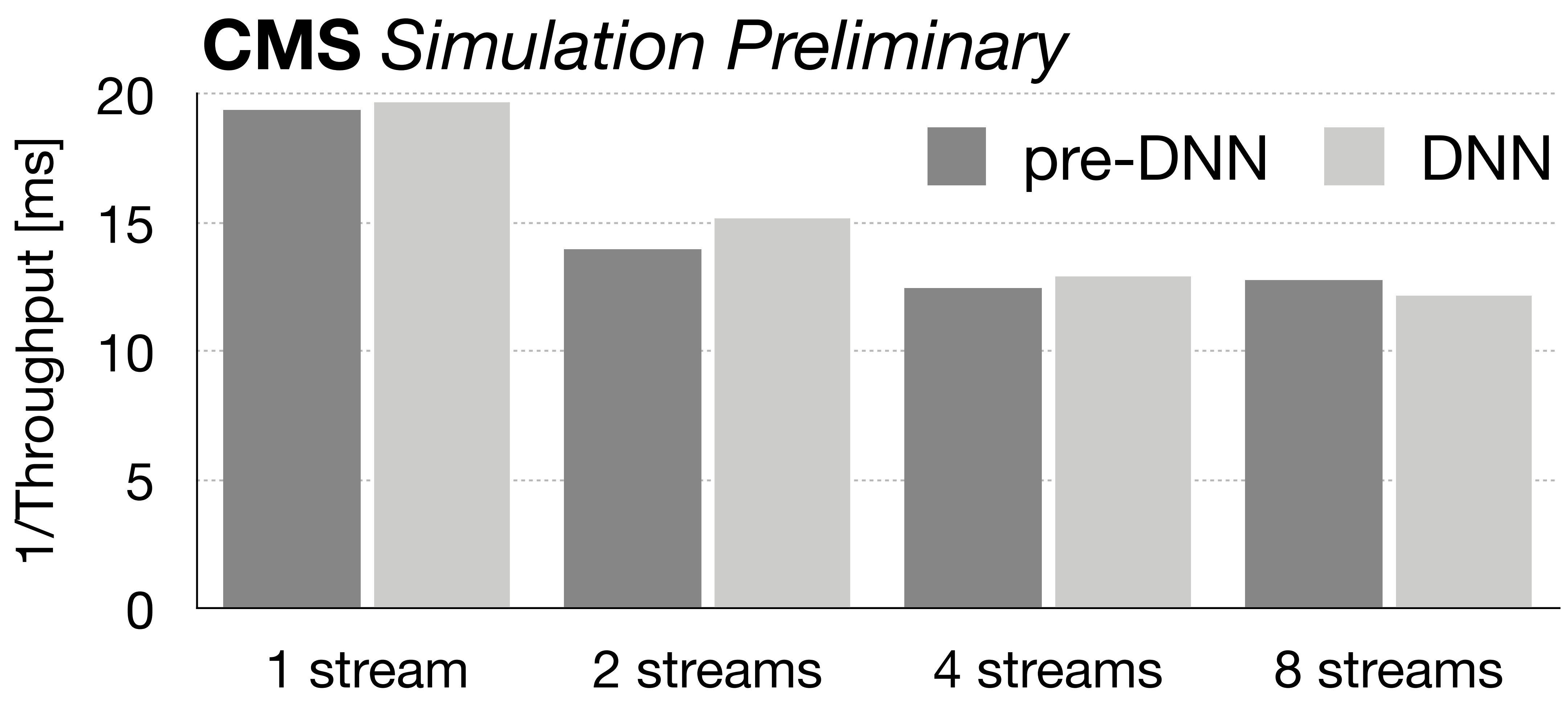
T5 DNN Timing Impact

Standalone

Units of milliseconds (ms)

	T5	1/Throughput	N streams
pre-DNN	**3.37 ± 0.13**	**28.4 ± 1.5**	1
DNN	**3.39 ± 0.07**	**28.7 ± 1.1**	1

Error is 1 standard deviation for 10 trials

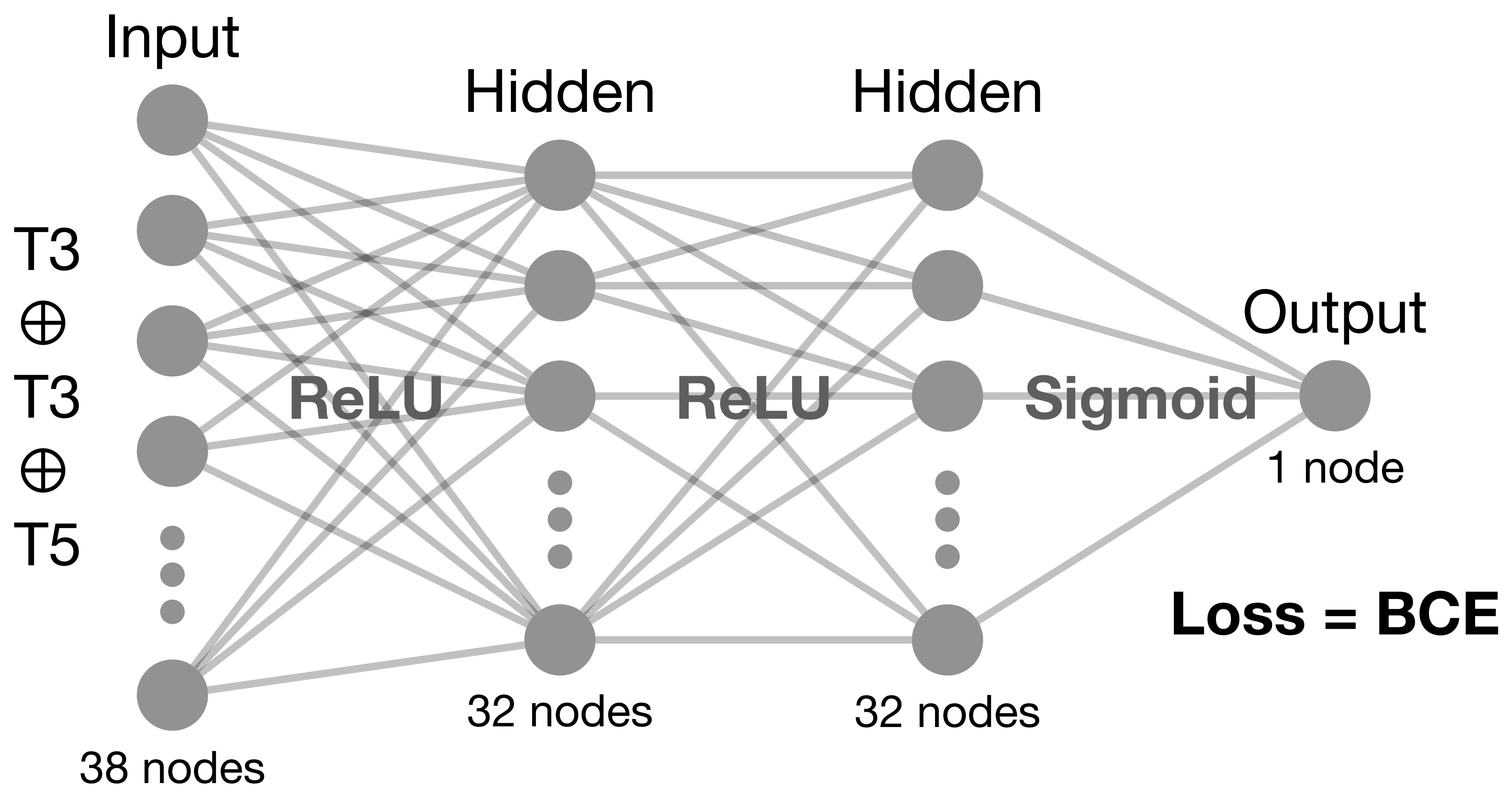


DNN has no measurable impact on LST runtime
(Measured on an NVidia A30 GPU)

UC San Diego

T5 DNN Summary

- We have trained a lightweight DNN to classify real vs. fake LST T5s
- We have shown that **the T5 DNN improves LST w/ no impact on runtime**
- We have **established a pipeline for training ML algorithms on LST data**
    - Includes a full simulation of the CMS detector



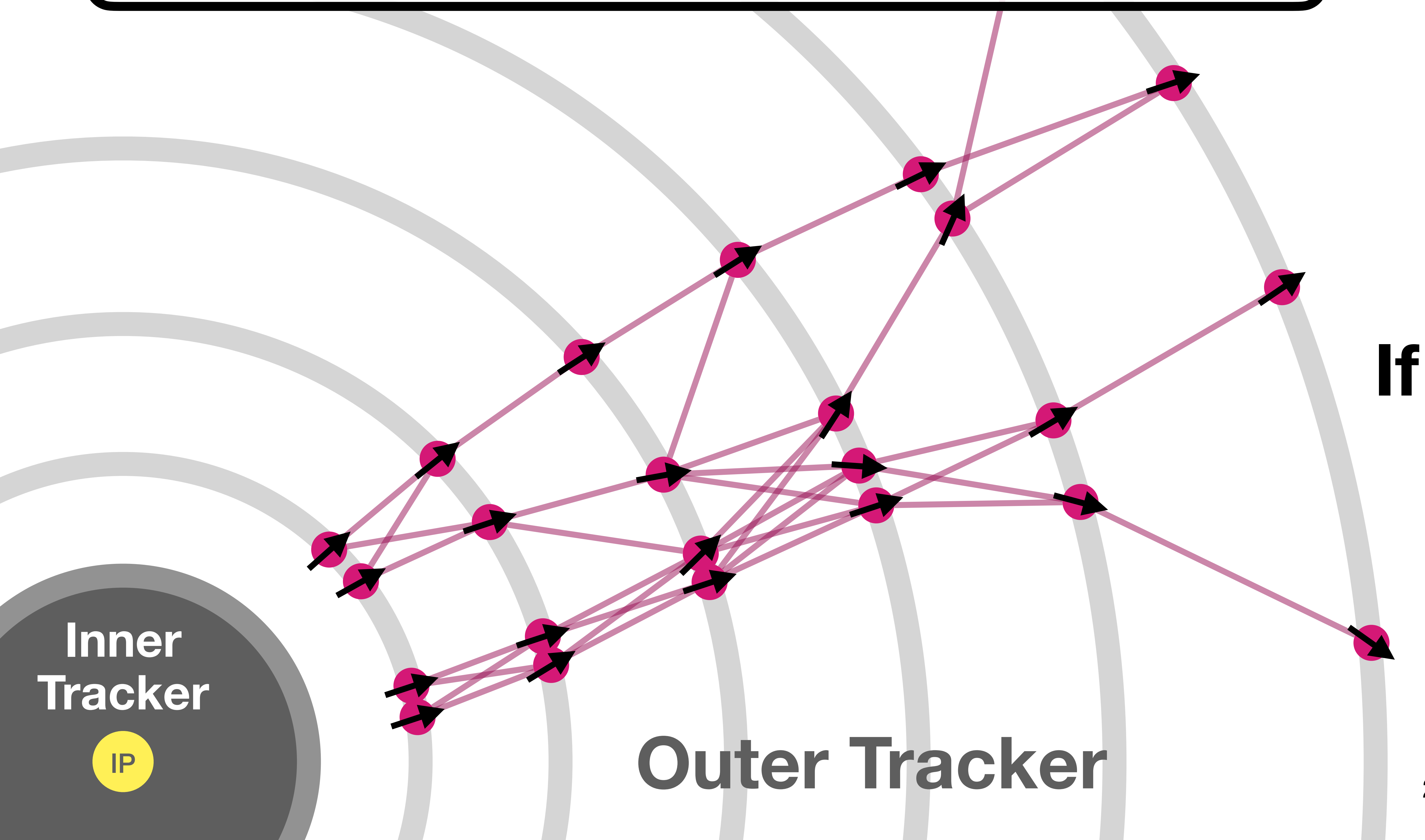
GNN Prospectus

- Interaction Networks (DeZoort, Thais, Duarte et al.) https://doi.org/10.1007/s41781-021-00073-z
- Exa.TrkX (Ju, Murnane, Calafiura et al.) https://doi.org/10.1140/epjc/s10052-021-09675-8
- GNNs (review by DeZoort, Battaglia, Biscarat et al.) https://doi.org/10.1038/s42254-023-00569-0
- And much more (see CTD 2023 agenda)!

Promising results from TrackML GNNs
No single best graph-building algorithm
No results using full CMS simulation

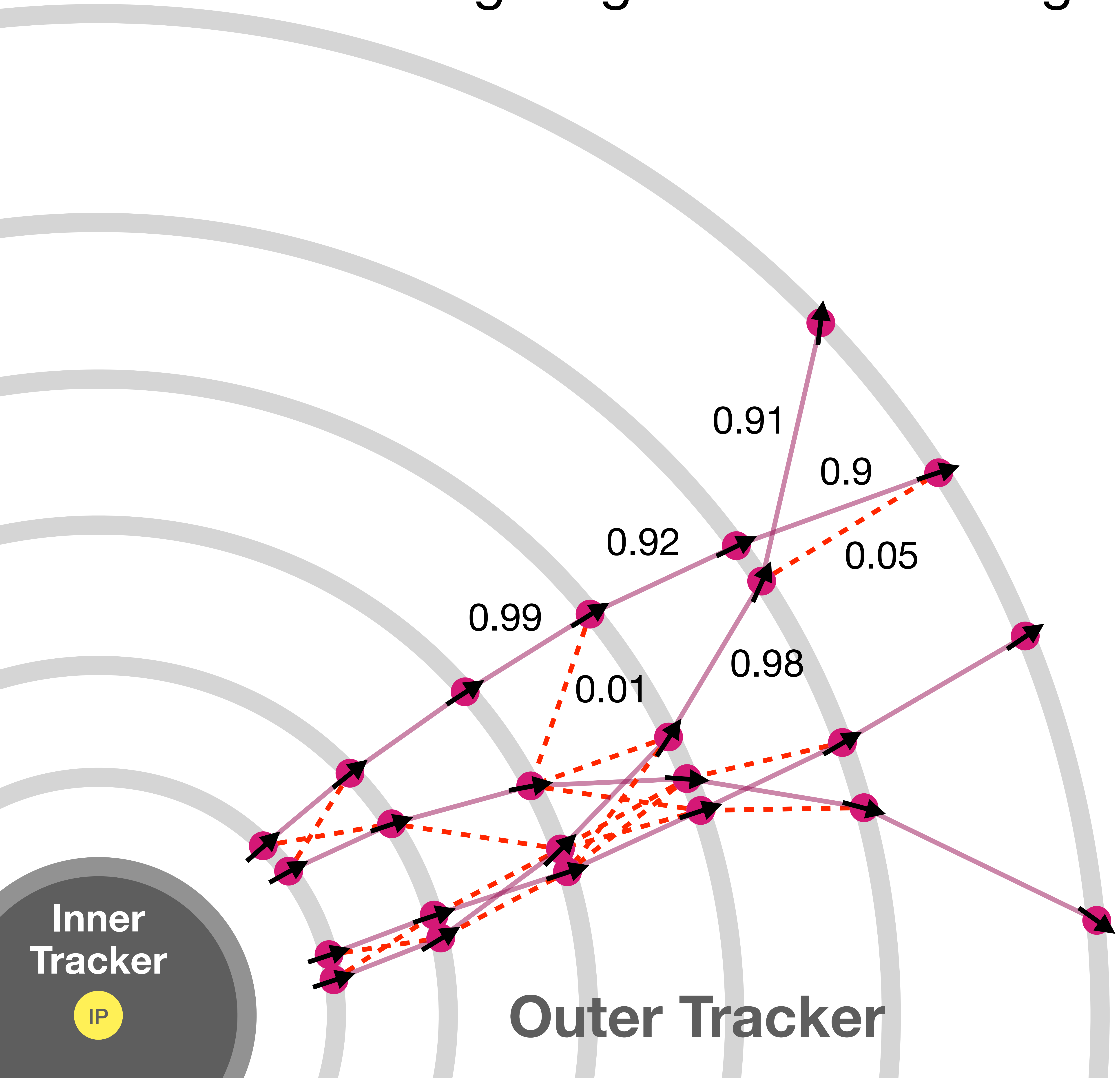
Use **LST to build input graph,**
most obvious: **MD nodes**, **LS edges**

If GNN can build good TCs: LST becomes a *very fast* graph building algorithm!
O(ms/event)



GNN Prospectus: Learning Objective

Targeting Exa.TrkX-like algorithm: https://exatrkx.github.io/



Develop some edge classifier that can **efficiently select real LSs**

Cut on classifier score and connect all remaining LSs (+ pLS) as TCs

Working on bringing pixel seeds into LS graph built by LST → full end-to-end tracking algorithm

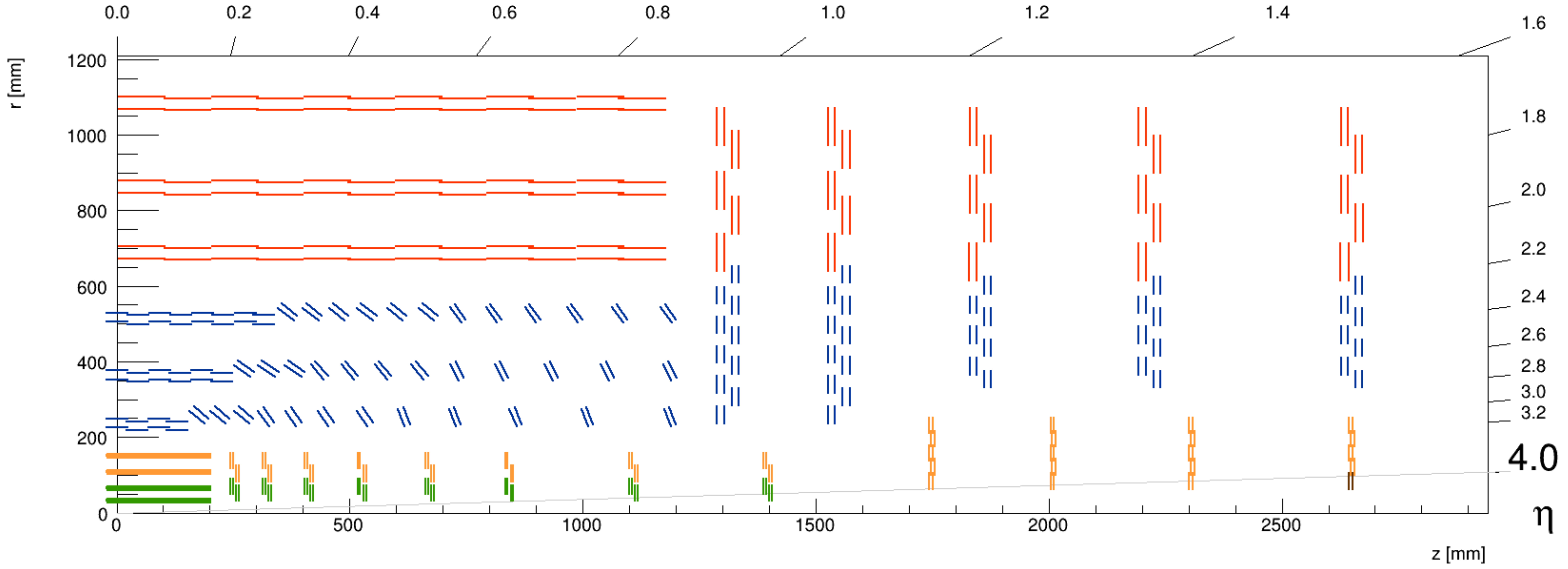
Conclusion

- LST is a highly performant and parallelizable tracking algorithm
- We are investigating how LST could be improved with ML
- We have **improved LST with a lightweight DNN**
  - The DNN has **no impact on the runtime**
- We are working towards training a LST GNN→track candidate pipeline

Thank you!

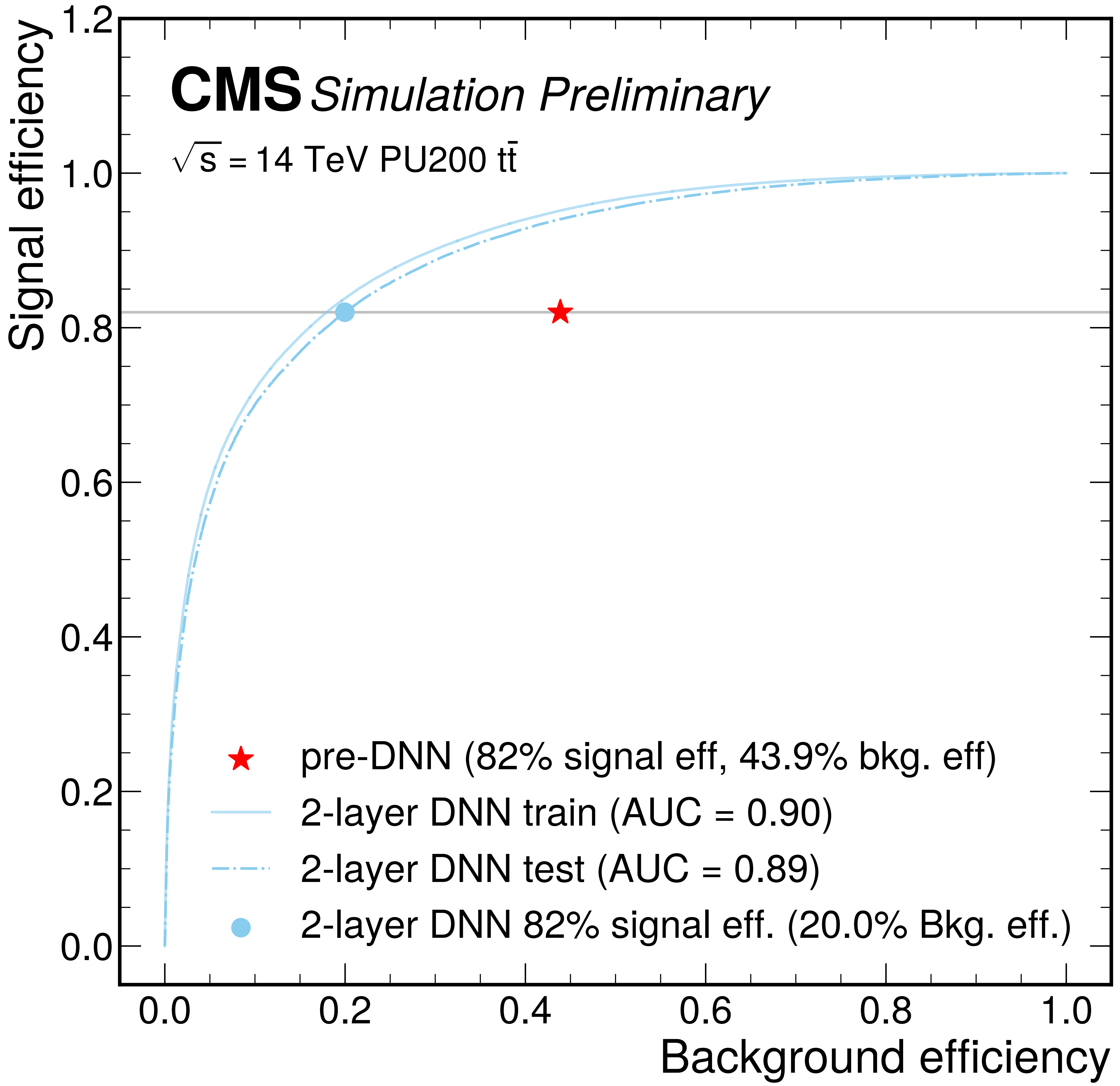
Backup

Phase 2 Tracker Layout



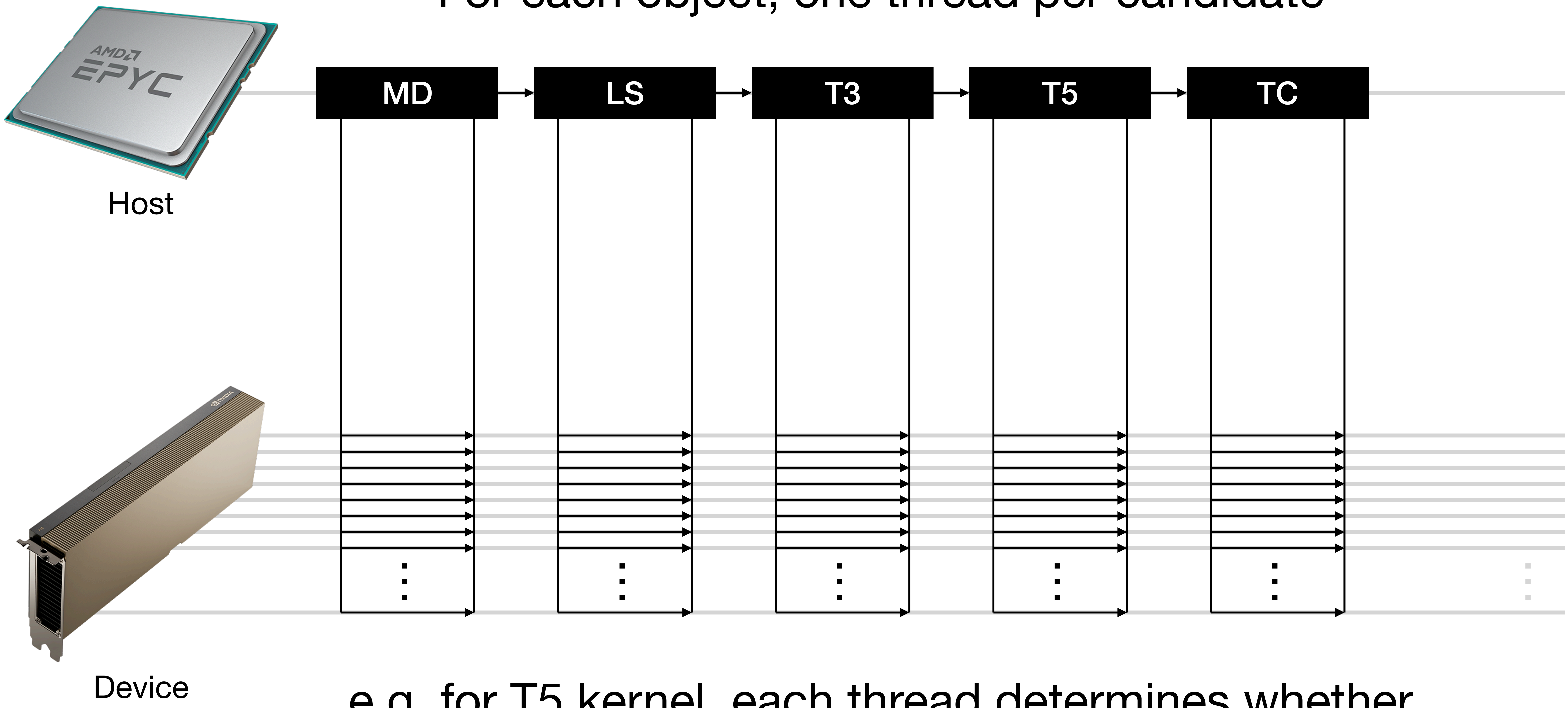
T5 DNN Training and Testing

- Background efficiency = FPR = FP/N
  - i.e. Fake rate
- Signal efficiency = TPR = TP/P
- For DNN ROC curve:
  - TP|FP = # real|fake T5s after DNN cut
  - P|N = # real|fake T5s before DNN cut
- For pre-DNN performance (★):
  - TP|FP = # real|fake T5s after r-ϕ χ^2 cut
  - P|N = # real|fake T5s before r-ϕ χ^2 cut



T5 DNN in LST

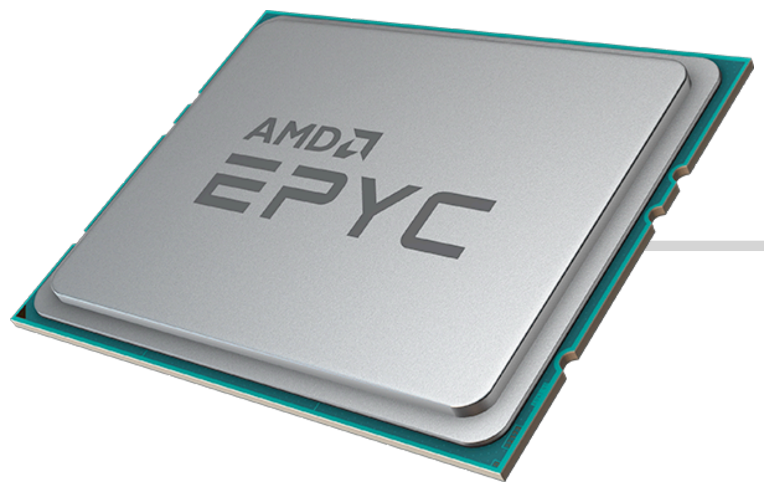
For each object, one thread per candidate



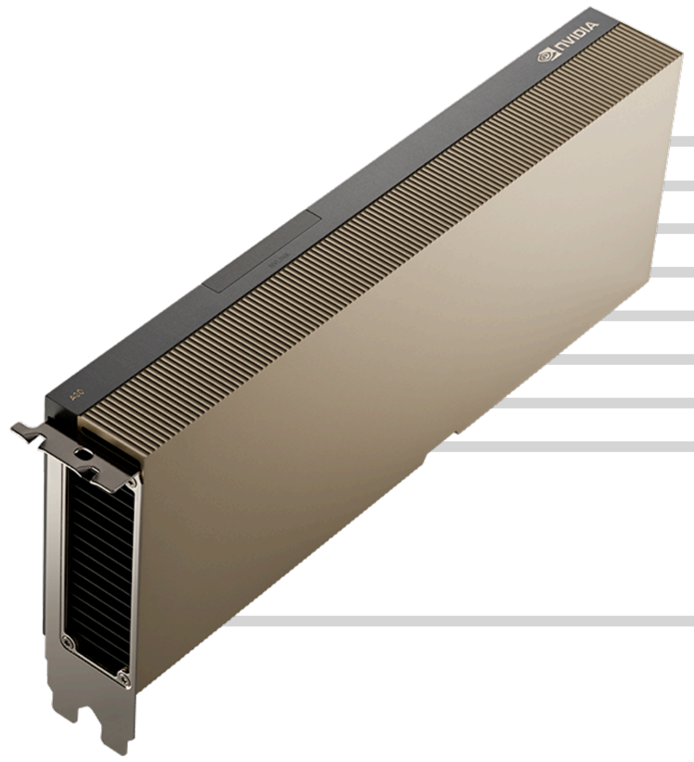
e.g. for T5 kernel, each thread determines whether or not to pass a given T5 candidate to the next step

T5 DNN in LST

Each thread (one per T5) runs the ML inference!

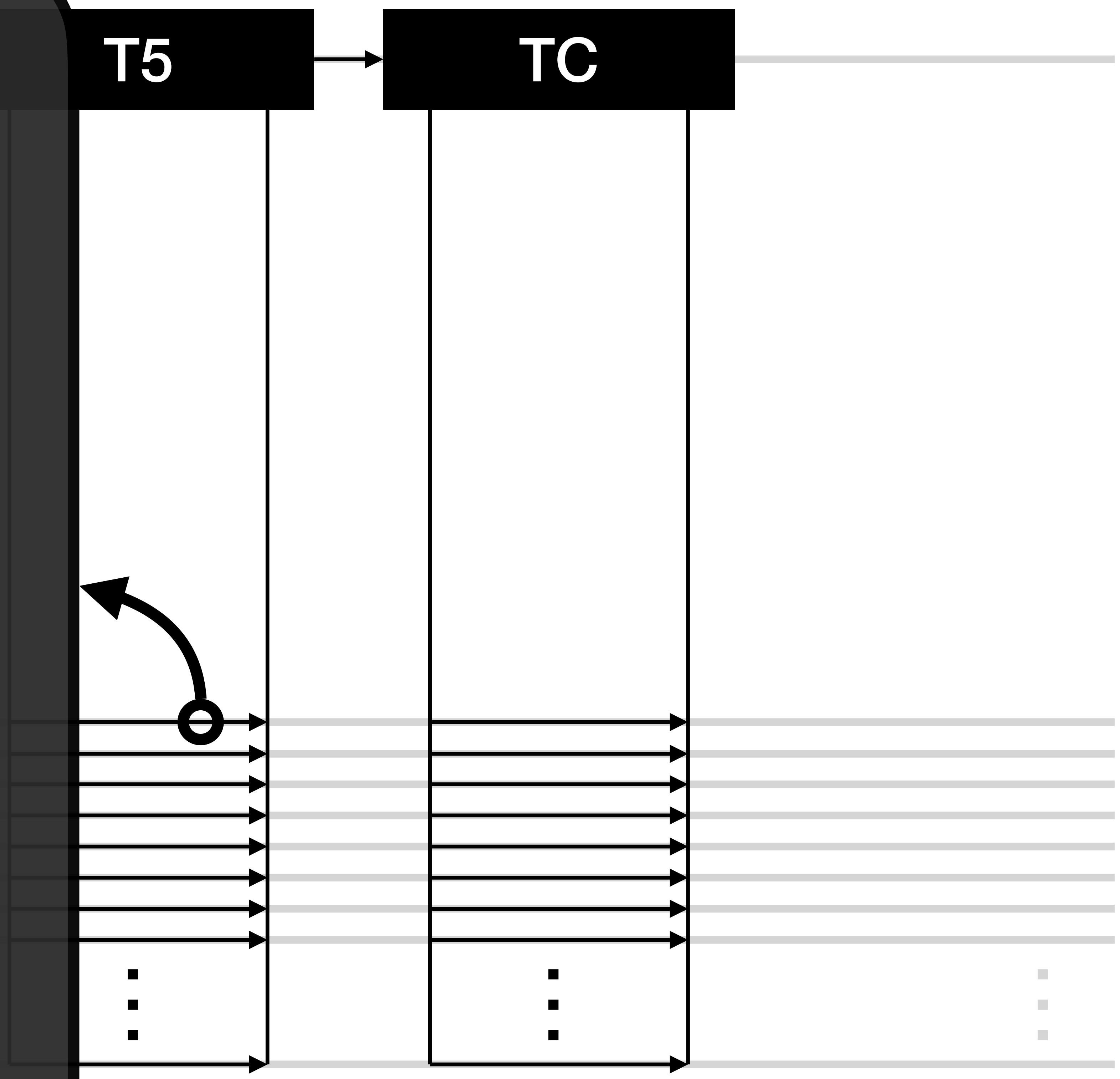


Host



Device

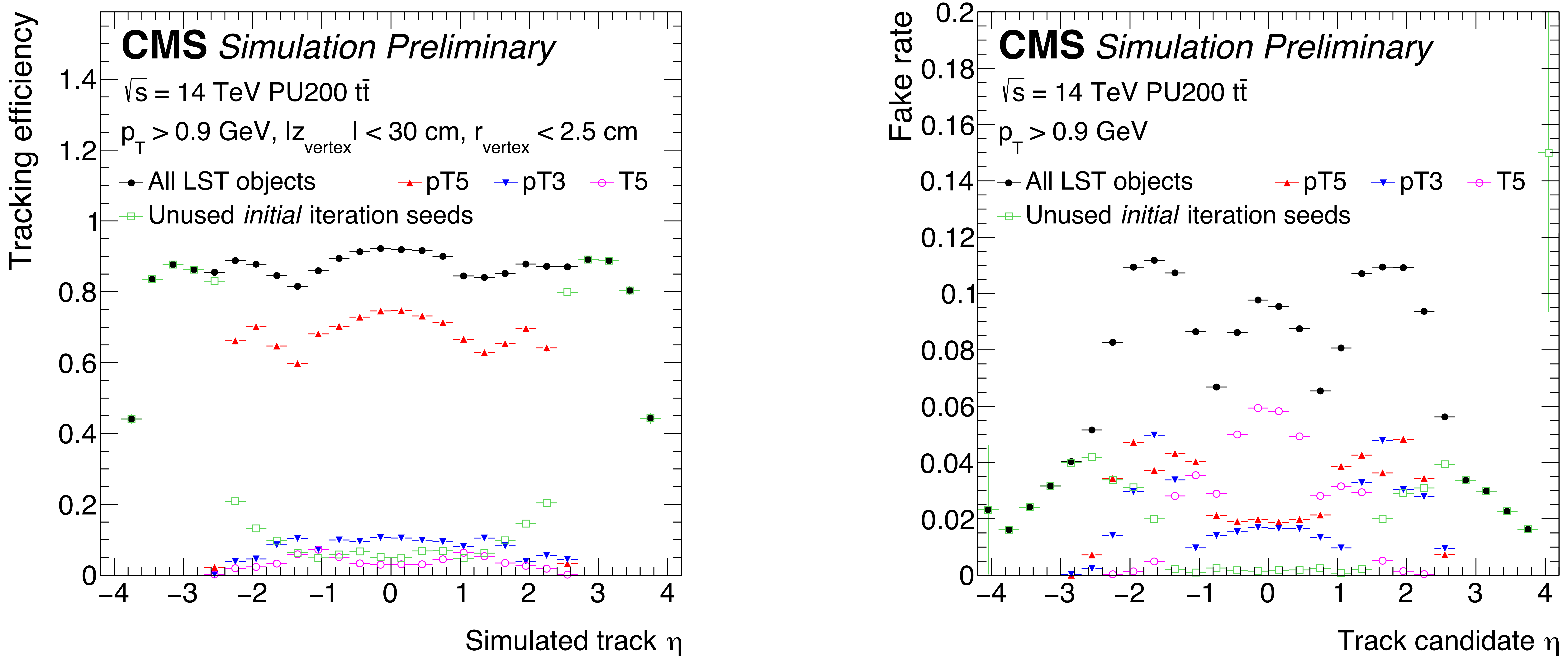
```
def passT5QualityCutsPseudoCode(...)
{
    // Original LST cuts
    if (!passBasicT5QualityCuts(...)) { return false; }
    if (!passRZChi2T5Cut(...)) { return false; }

    // Build DNN input features vector
    float x[38] = {
        log10(innerT3pT),
        innerT3eta,
        ...
    };

    // Input -> first hidden layer
    float hidden0[32];
    for (int col = 0; col < 32; ++col) {
        hidden0[col] = 0.f;
        for (int inner = 0; inner < 38; ++inner) {
            hidden0[col] += x[inner]*wgts0[inner][col];
        }
    }
    hidden0 = leakyReLU(hidden0);
    ... // and so on…

    // Last hidden layer -> output
    float inference = 0.f;
    for (int i = 0; i < 32; ++i) {
        inference += hidden1[i]*wgts4[i][0];
    }
    inference = sigmoid(inference);

    if (inference < LSTDNN::WP95) { return false; }

    return true;
}
```



Very simple implementation
(certainly not optimal)

Standalone

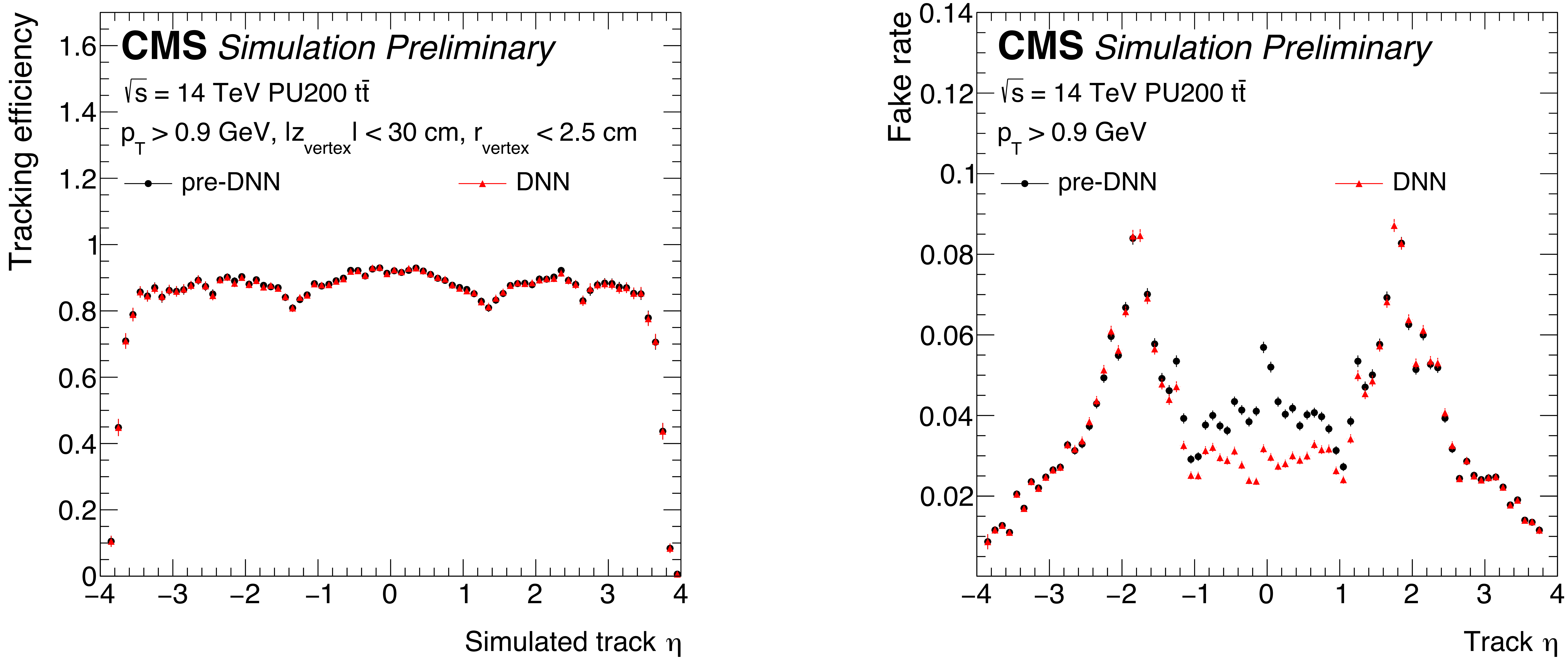
Pre-DNN Performance



T5s account for much of the LST efficiency
T5s have a high fake rate

Performance Comparison

Standalone + CMSSW



Significant reduction of fake rate in the barrel
No loss in efficiency