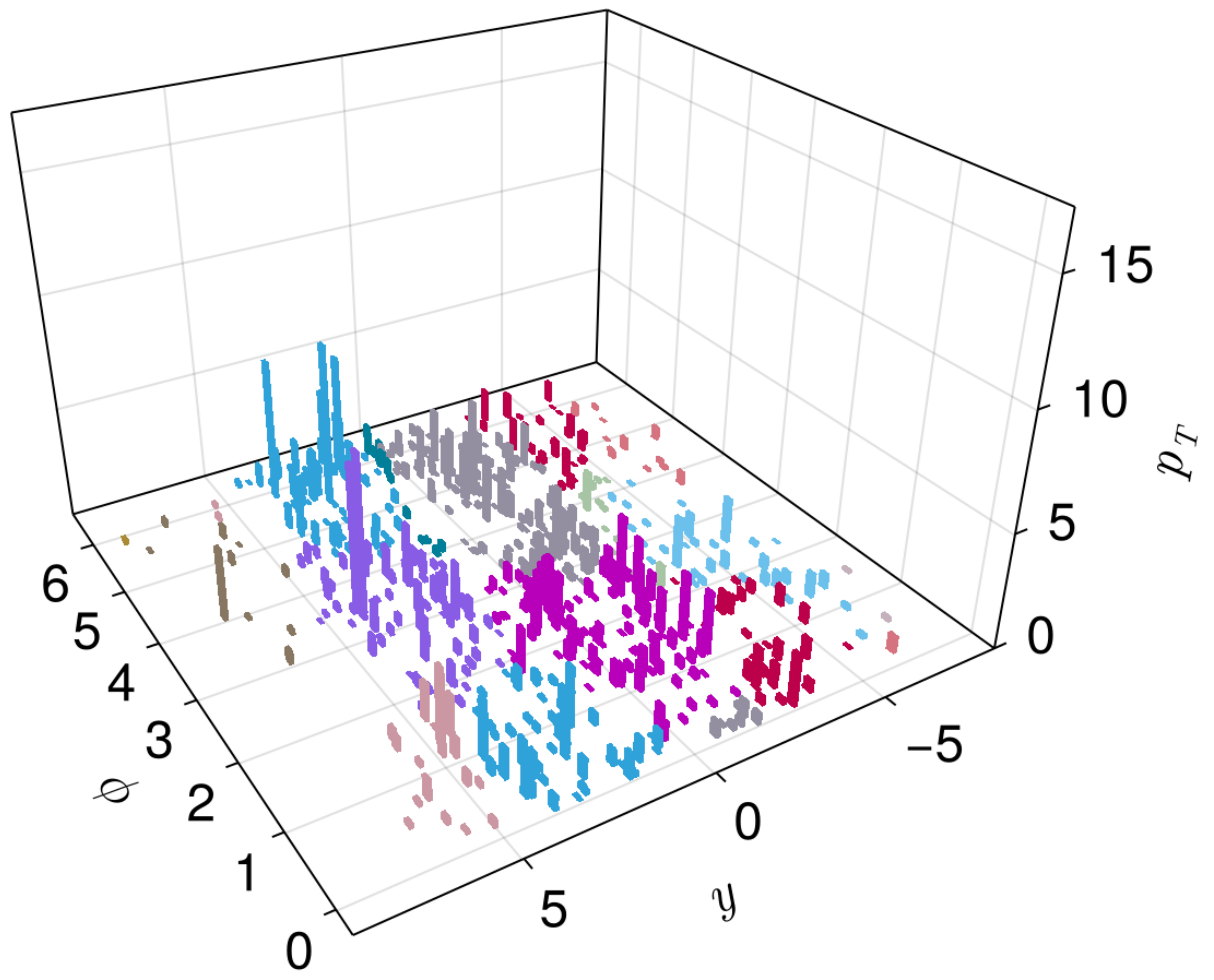
JetReconstruction.jl

JuliaHEP 2024

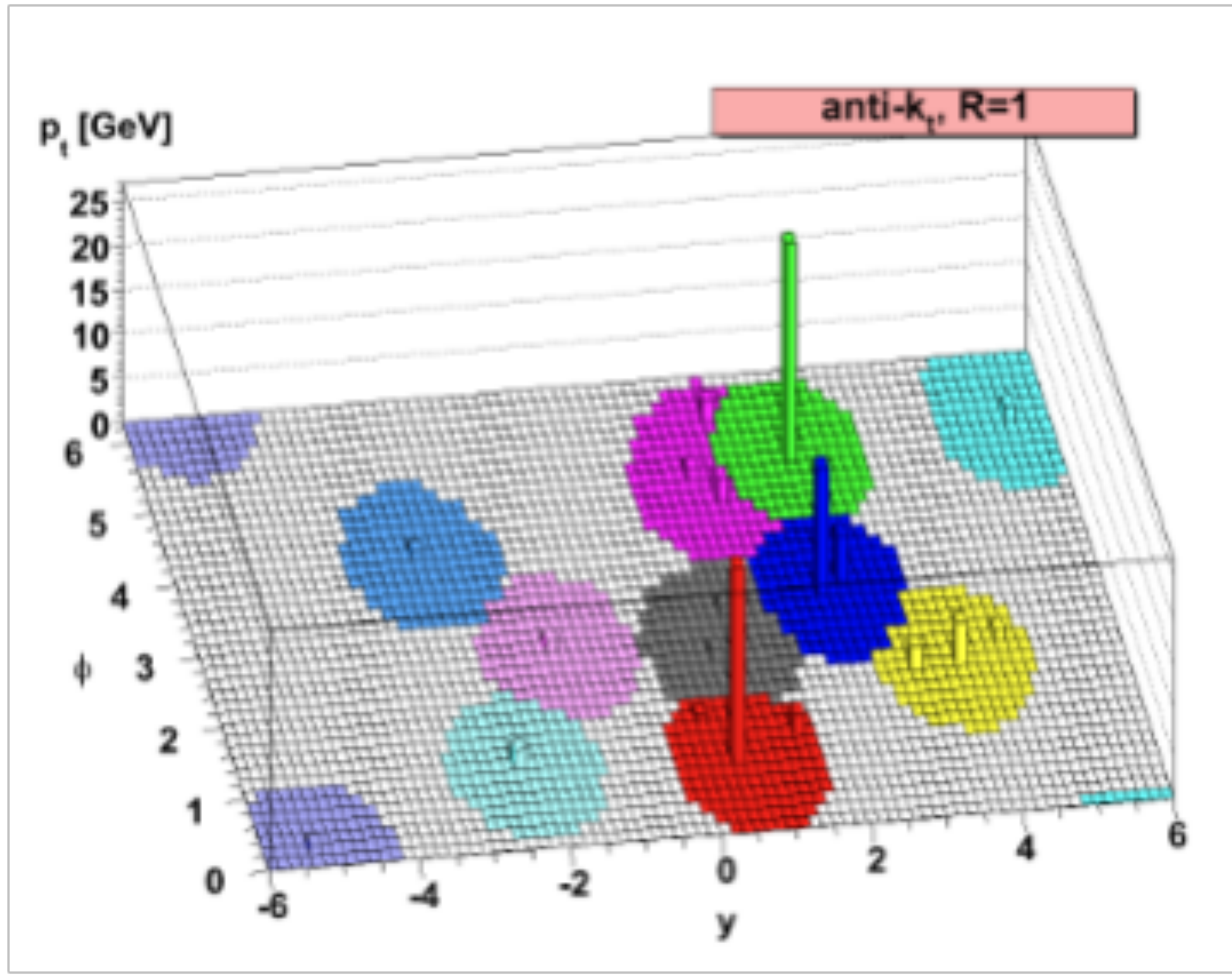
Graeme Stewart (with Philippe Gras and Atell Krasnopolski)

Jet Finding

- Jet finding is a good example of a "goldilocks" algorithm to test Julia
  - The goal is to cluster calorimeter energy deposits into jets
- There exists a highly optimised C++ package, almost ubiquitously used, FastJet
  - There are a number of algorithms for jet clustering
  - AntiKt clustering is popularly used for pp collisions because it is an infrared and co-linear safe [arXiv:0802.1189]
- We get to test language ergonomics and performance



Sequential Jet Reconstruction



- Define parameters p and R
  - Not used by all algorithms, but R is a "cone size" and $2p$ is a metric distance power
- For each active pseudo-jet i (=particle, cluster) measure the geometric distance, d, to the nearest active pseudo-jet j
  - If R is defined, and there are no other pseudo jets with R, then $d = R$
- Define the metric distance, d_{ij}, as
  - $d_{ij} = d \cdot min(p_{Ti}^{2p}, p_{Tj}^{2p})$ or $d_{ij} = d \cdot min(E_i^{2p}, E_j^{2p})$
- Choose the jet with the lowest d_{ij}
  - If this jet has an active partner j, merge these jets
  - If not, this is a final jet
- Repeat until no jets remain active

For pp events:

$p = -1$ AntiKt

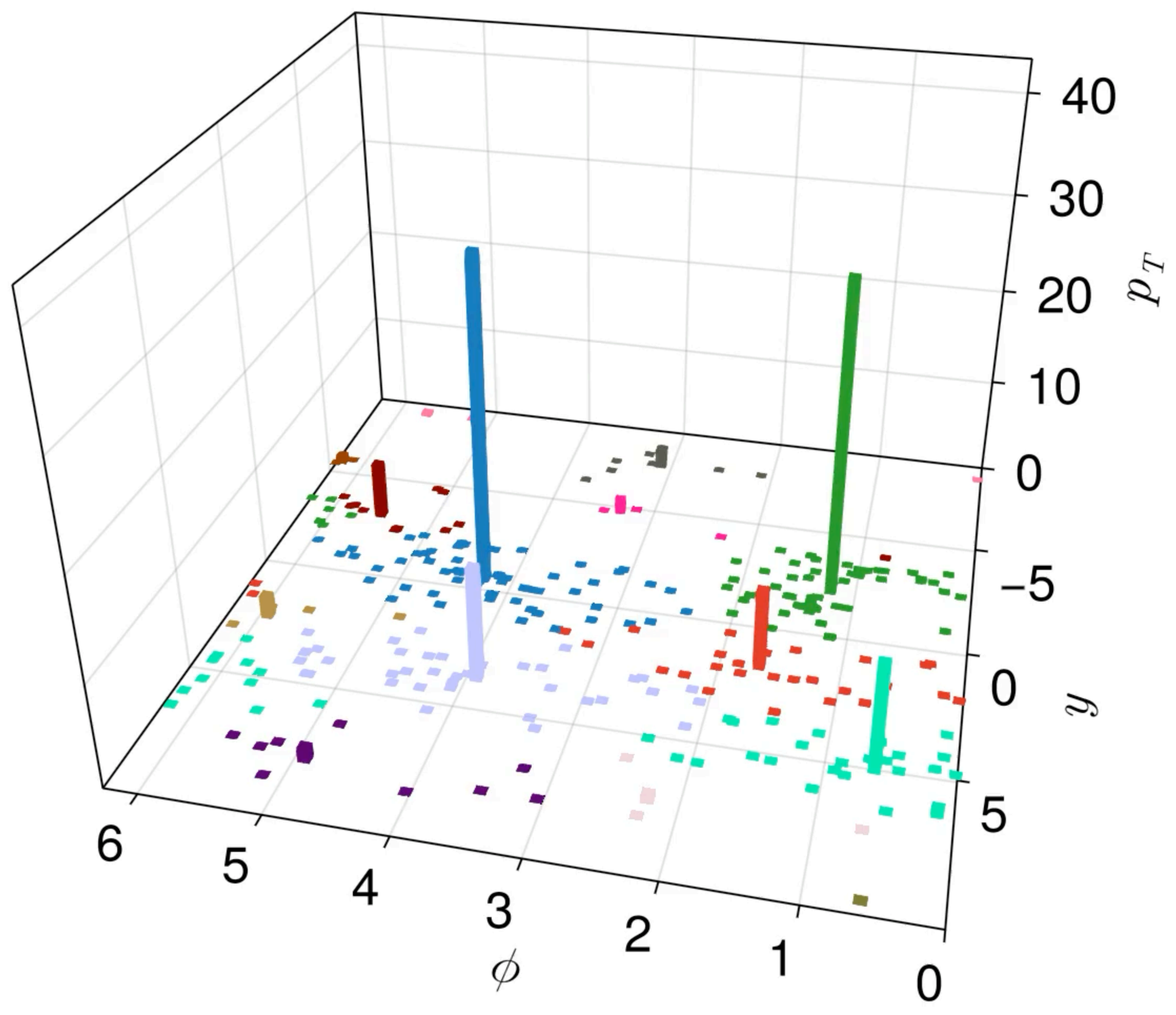
$p = 0$ Cambridge/Achen

$p = 1$ Inclusive Kt

Status Last Year

- Very encouraging results from Julia
    - At least as good as Fastjet for pp event reconstruction
- Decided that this was worth changing from an R&D endeavour to a real package
- Released in June this year as JetReconstruction.jl 🎉

Anti $-$ k_T Jet Reconstruction, 13TeV pp collision



The Release

- A fair amount of refactoring was required to ensure that the two pp strategies (N2P[illegible]d N2Tiled) behaved in the same way
  - Internal restructuring to uniformly use PseudoJets and return ClusterSequence objects
- Implemented exclusive jet selections (n_jet or dij_max cut)
- Implemented *generalised kT algorithm* (i.e. p_T^{2p} for arbitrary p) ✅
- Choice of strategies: N2Plain, N2Tiled and **Best**
- Fixes to visualisation and improved examples
- Significant improvements to documentation (helped by Co-pilot!)
  - Overview, method and structure documentation
    - Documenter.jl setup
    - Published at https://juliahep.github.io/JetReconstruction.jl/dev/

Thanks to Jerry Ling for the beautification tweaks!

Interface

Particles is a vector of 4-momentum objects; specify algorithm and any other parameters

```
Clusters = jet_reconstruct(particles; algorithm = JetAlgorithm.AntiKt, R = 1.0)
```

then

```
jets = inclusive_jets(clusterseq; ptmin = 5.0)
```

or

```
jets = exclusive_jets(clusterseq; njets = 3)
```

Inclusive jets are "finalised" merged jets; exclusive jets rewinds reconstruction to the point where, e.g., there were only 3 jets remaining

and maybe

```
constituents = JetReconstruction.constituents(jet, clusterseq)
```

Constituents usually then passed to a particle ID algorithm (N.B. this API not finalised)

Feedback, Contributions and Extensions

- The package was picked up by an ATLAS summer student
  - Very much successfully used and praised c.f. other student's Python work
  - Needed to add a constituent retrieval function

```
JetReconstruction.constituents(jet, cluster_seq)
```

  - I would not consider this stable quite yet, as there could be a better API
- Also Sattwamo started to implement substructure and taggers (next talk!)
- Started to discuss with our FCCee colleagues how to use this to extend FCC analysis in Julia beyond the purely kinematic examples
  - This required the different set of algorithms favoured for e^+e^- events…

A Tale of Two Particle (Algorithms)

- The main difference between pp and e^+e^- algorithms is that
  - Geometric distance metric is defined in angular space (θ, ϕ) instead of in rapidity space (y, ϕ)
  - d_{ij} metric uses E^{2p} instead of p_T^{2p}

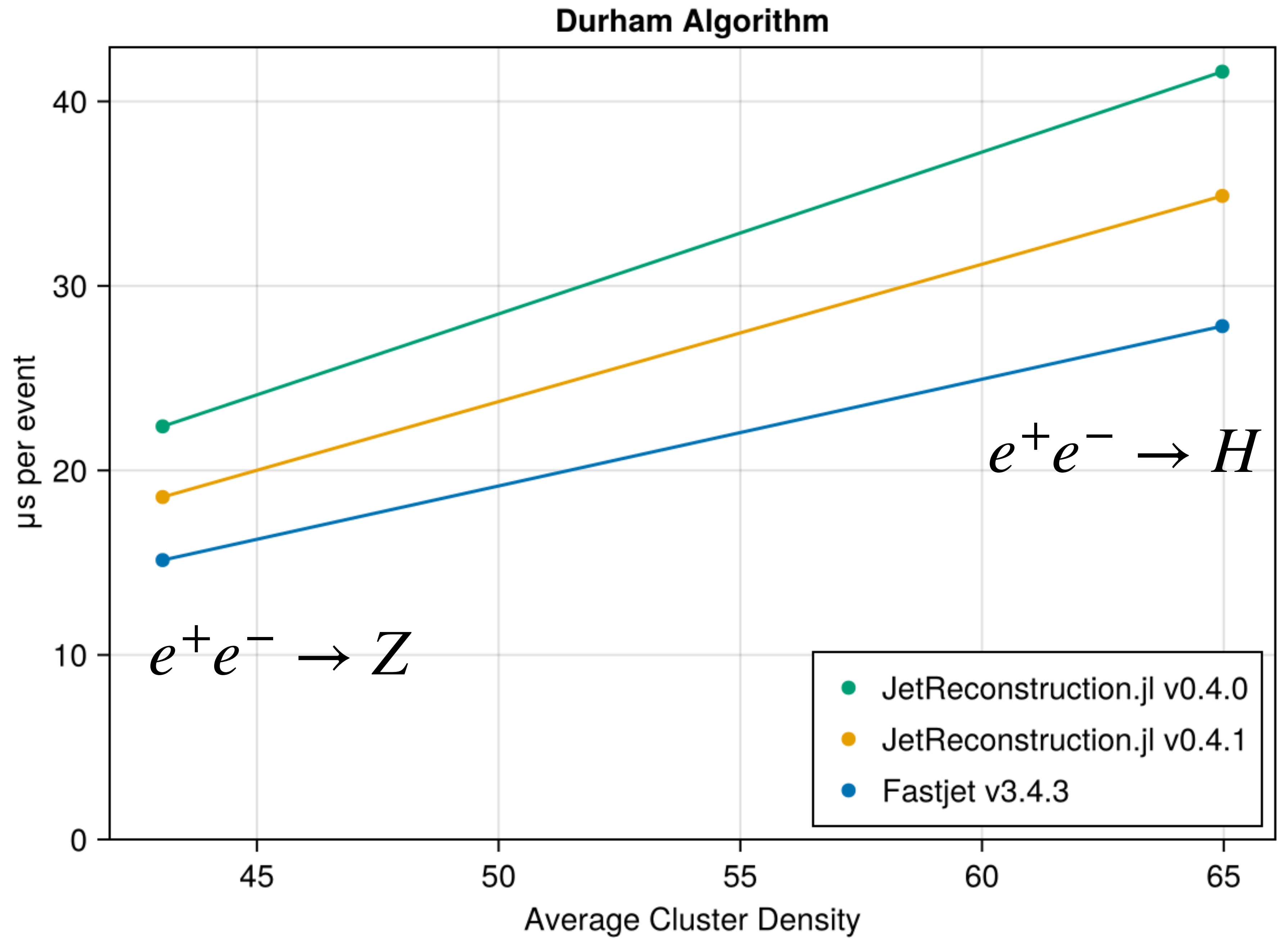
	Durham (e^+e^-)	**Generalised kT (e^+e^-)**	**AntiKt (pp)**
Geometric Distance	$1 - \cos \theta_{ij}$	$1 - \cos \theta_{ij}$	$R_{ij} = \sqrt{(\phi_i - \phi_j)^2 - (y_i - y_j)^2}$
d_{ij}	$2 \min(E_i^2, E_j^2)(1 - \cos \theta_{ij})$	$\min(E_i^{2p}, E_j^{2p})(1 - \cos \theta_{ij})$ $/(1 - \cos R)$	$\min(p_{Ti}^{-2}, p_{Tj}^{-2}) R_{ij}^2 / R^2$
Parameters		p, R	R
Notes		For p=1, $\pi < R < 3\pi$ equivalent to Durham	p=-1 is AntiKt, p=0 is Cambridge Aachen, p=1 inclusive kT

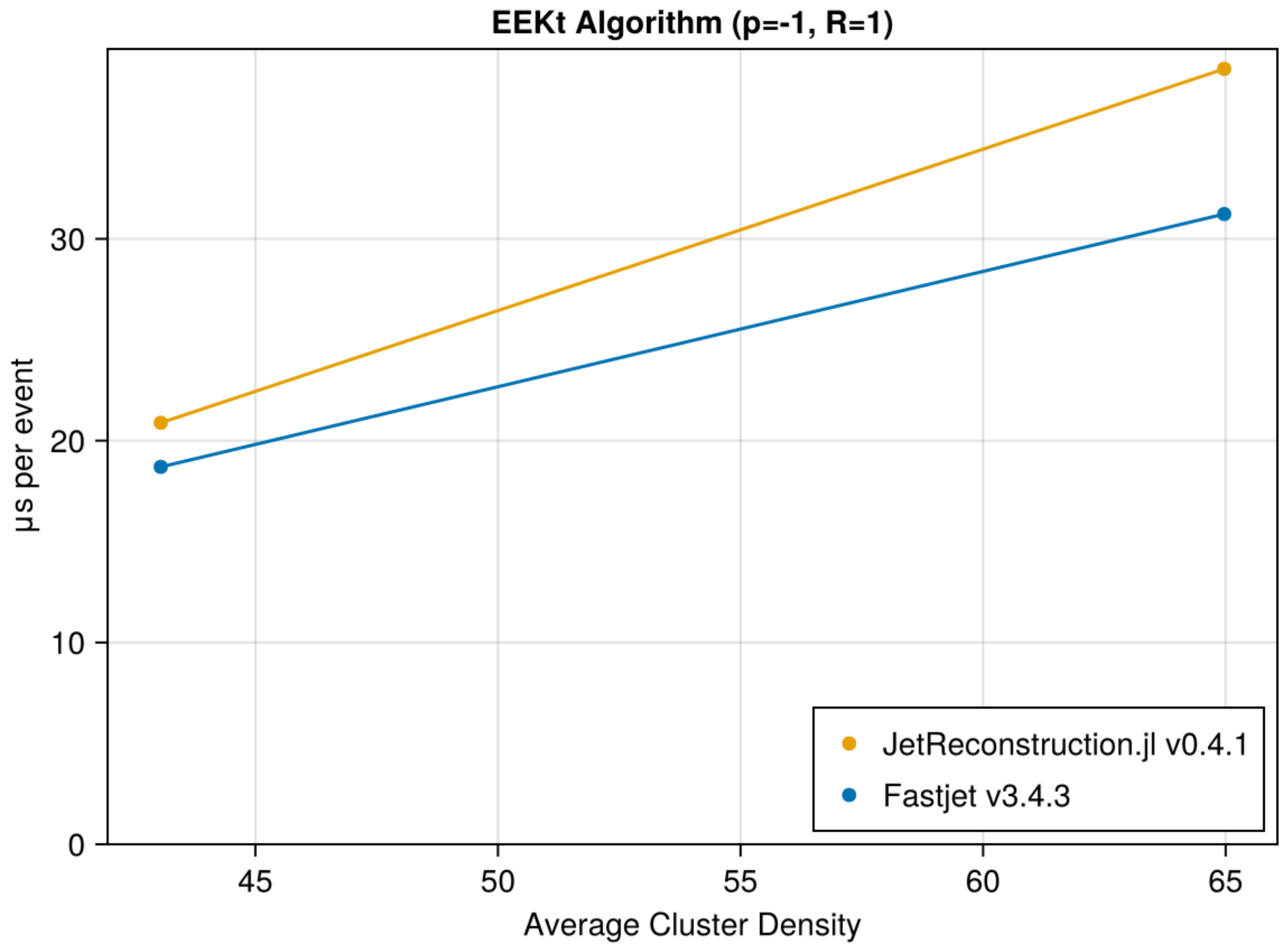
A Few Implementation Details…

- The PseudoJet class used in the pp reconstruction wasn't very suitable for e^+e^-
    - It is working in (y, ϕ) space not (θ, ϕ) space
    - Want to cache normalised momenta to calculate θ_{ij} from a dot product
- There's a new `EEJet` class
    - Concrete subtype of abstract `FourMomentum` (as is `PseudoJet`)
    - However, `LorentzVectorHEP` isn't…
        - Wouldn't it be nice to have a `FourMomentumBase` (hackathon!)
- The tiled strategy is *not* implemented here
    - Particle densities are too low to make this worthwhile

Performance

- Initial performance was pretty disappointing
  - Slower than FastJet (which has a very optimised implementation for these geometric reconstructions)
- Did a rewrite using StructureOfArrays layout for all quantities used in the reconstruction sequence
  - I took advantage of StructArrays.jl to do this
  - Make it look like an ArrayOfStructs, but it's SoA underneath
- Provided quite some speed-up for Apple M2
  - Made no difference on x86 (AMD Ryzen and Intel i7)!?
- Fastjet is ~20% faster at the moment





What's Next

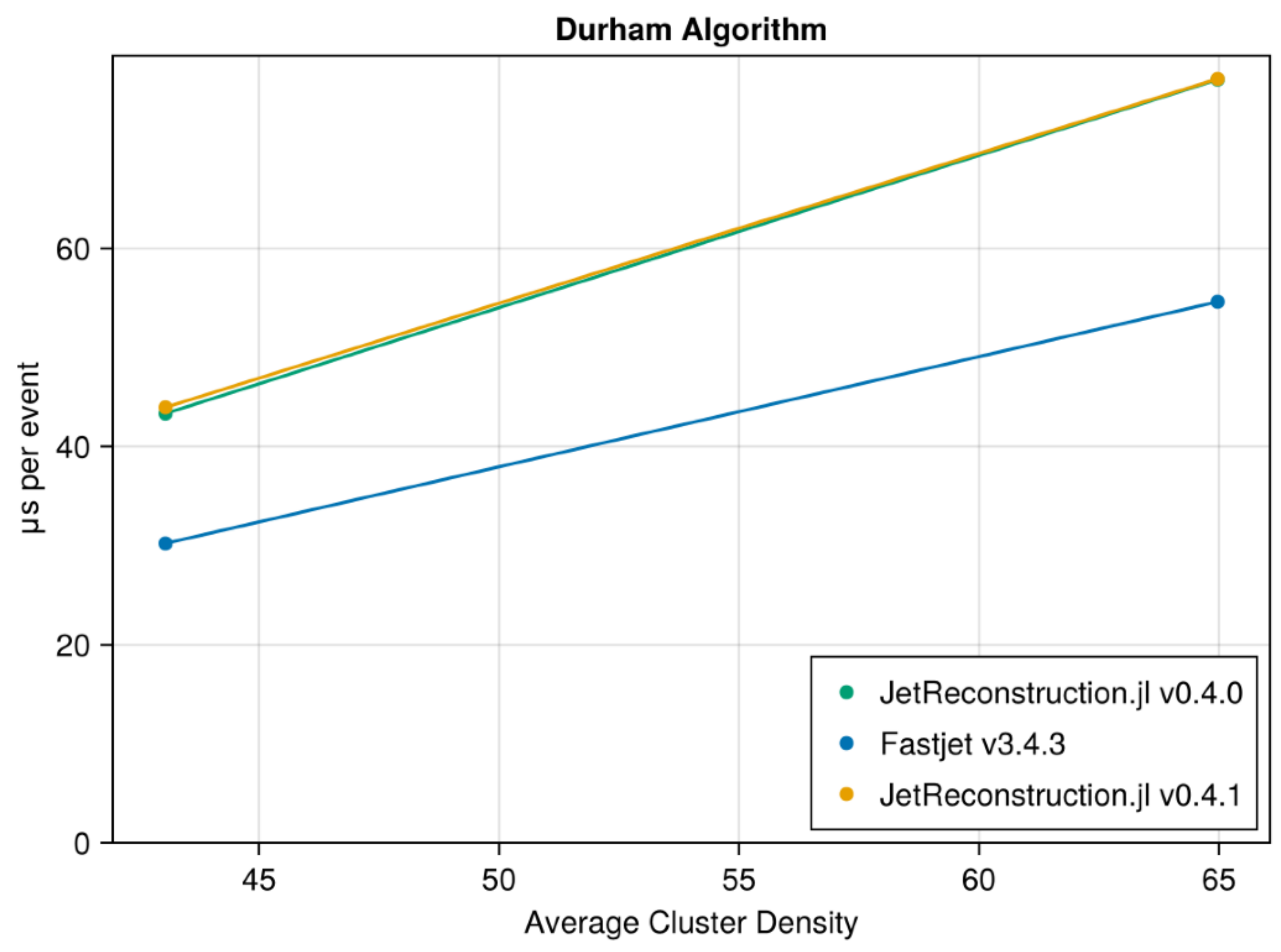
- ee
  - Be able to smoothly handle `EDM4hep` data
    - Should not be very hard to do
      - Bonus points: `EDM4HepReconstructedParticle <: FourMomentumBase`
  - Stabilise constituents interface
    - Needed for the next reconstruction step, particle identification
- pp
  - Integrate Sattwamo's work on substructure and taggers
- General care and feeding
  - Proper parameterised internal types (`Float32`, `Float64`)
  - Take another look at optimisation…?

Hackathon Topics

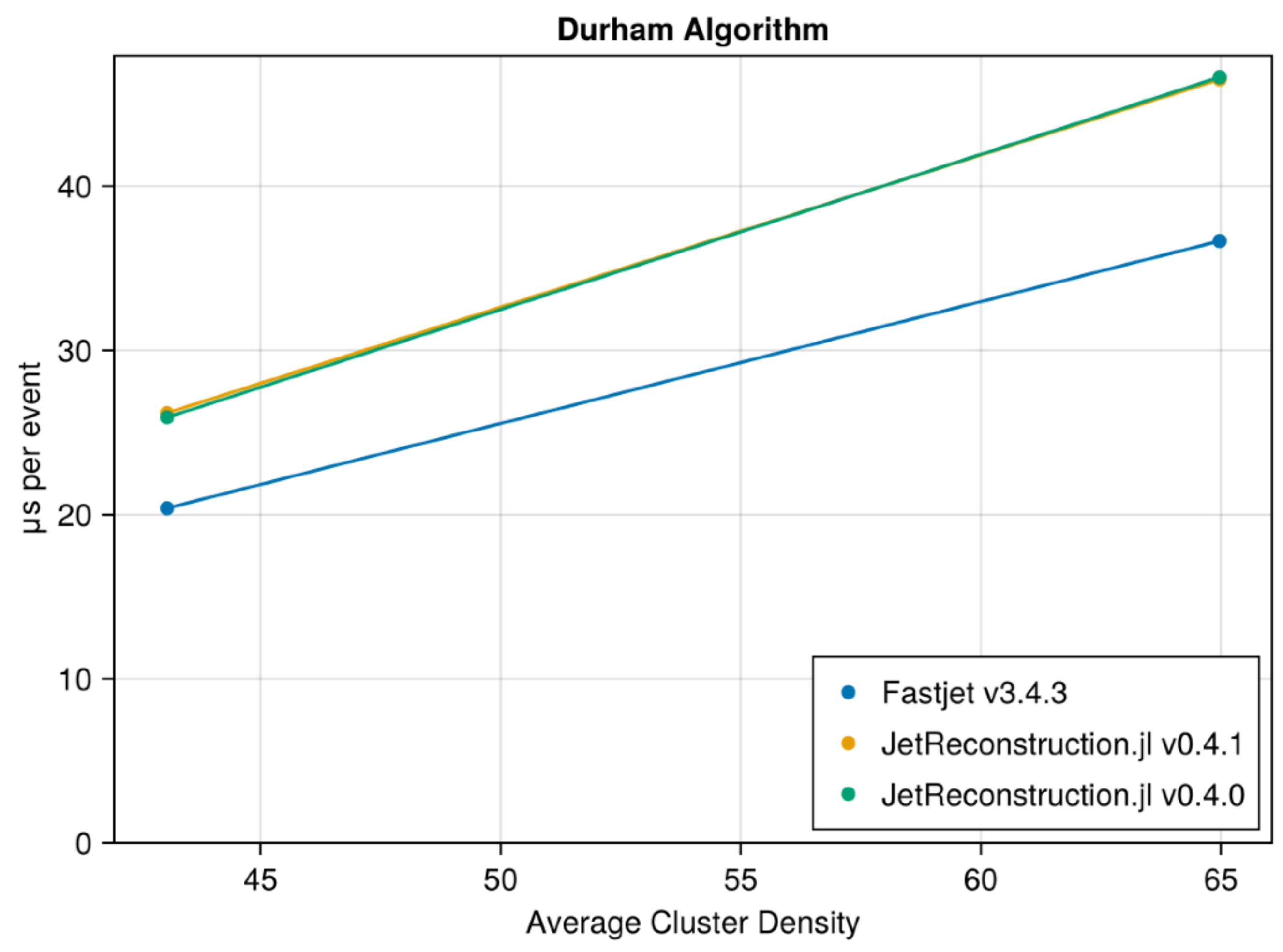
- FourMomentumBase
- Static compilation

Backup

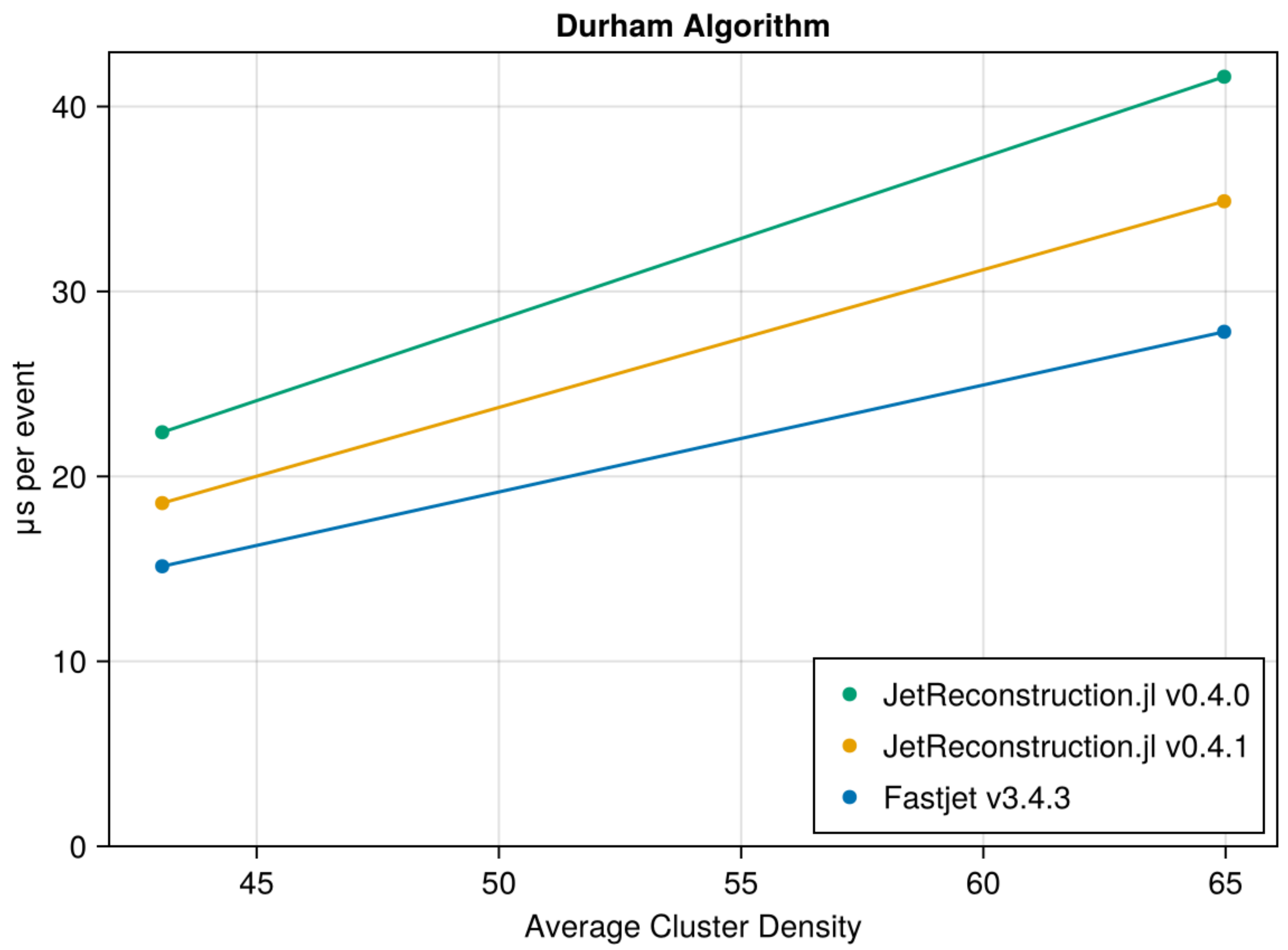
Durham Algorithm - Different CPUs



Intel i7-3770, 3.7GHz



AMD Ryzen 7 5700G
3.8GHz



Apple M2Pro 3.5GHz