MCnet-LPCC Summer School

July 2012, CERN

Jets

Lecture 1

Matteo Cacciari

LPTHE Paris

Outline

- Lecture 1: jet clustering
  - *Basic requirements and properties*
  - *Jet clustering algorithms*

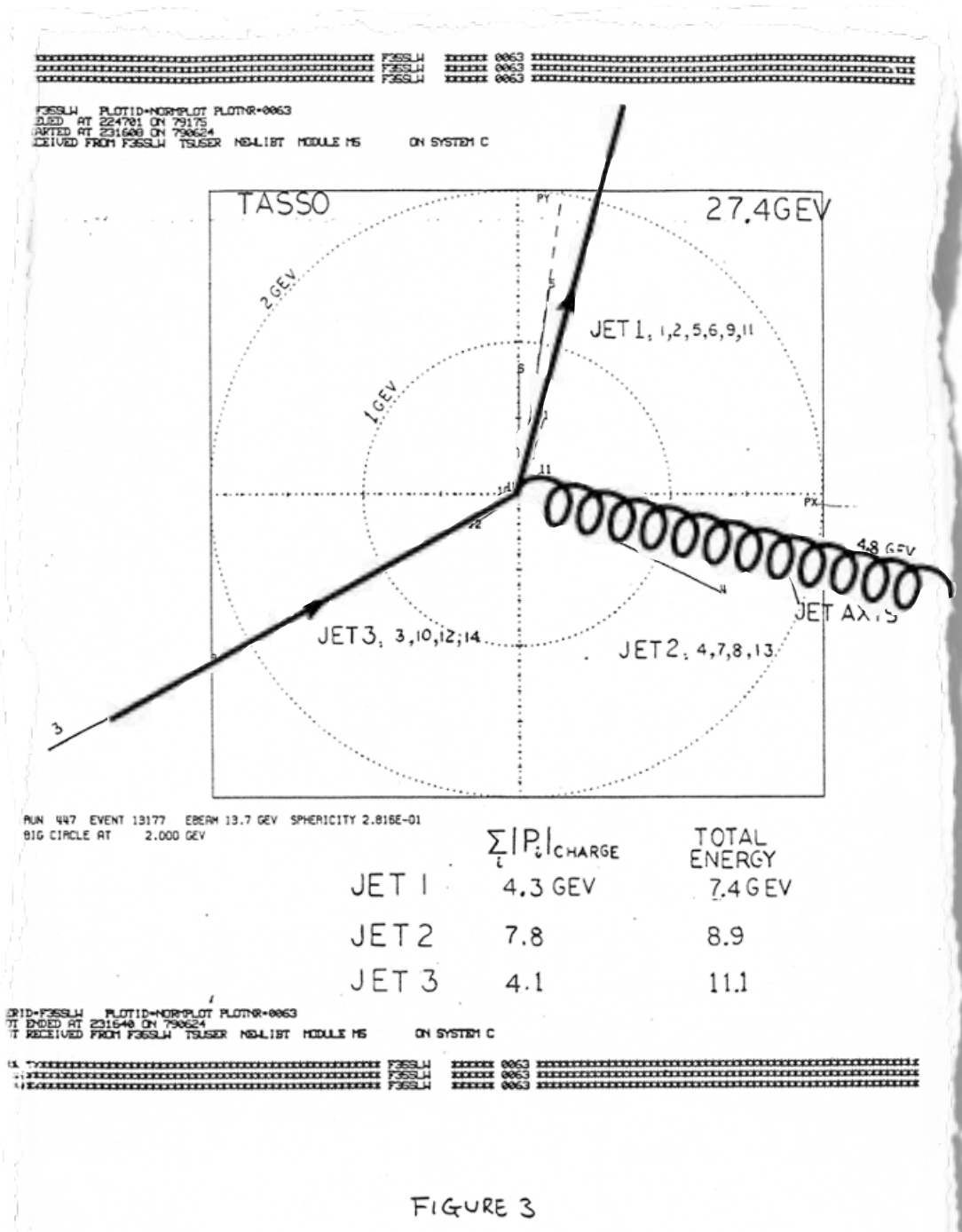
Code snippets detailing FastJet use for some specific cases will be included in the slides. Of course one can obtain equivalent results in a (slightly) different way using SpartyJet (it wraps FastJet and include additional tools, i.e. ROOT interface) or the dedicated software from the experiments, which also tend to wrap FastJet with they own interface.

These lectures include many ideas and contributions by Gavin Salam and Gregory Soyez

Bibliography

- Les Houches 2007 proceedings, arXiv:0803.0678
- Gavin Salam, 'Towards Jetography', arXiv:0906.1833
- Proceedings of BOOST 2010 and 2011: arXiv:1012.5412 and arXiv:1201.0008
- Gavin Salam's lectures at CERN Academic Training (March 2011): http://indico.cern.ch/event/115078
- FastJet user manual, MC, G.P. Salam and G. Soyez, arXiv:1111.6097

Gluon 'discovery'



	$\sum_i \lvert P_i \rvert_{CHARGE}$	TOTAL ENERGY
JET 1	4.3 GEV	7.4 GEV
JET 2	7.8	8.9
JET 3	4.1	11.1

FIGURE 3

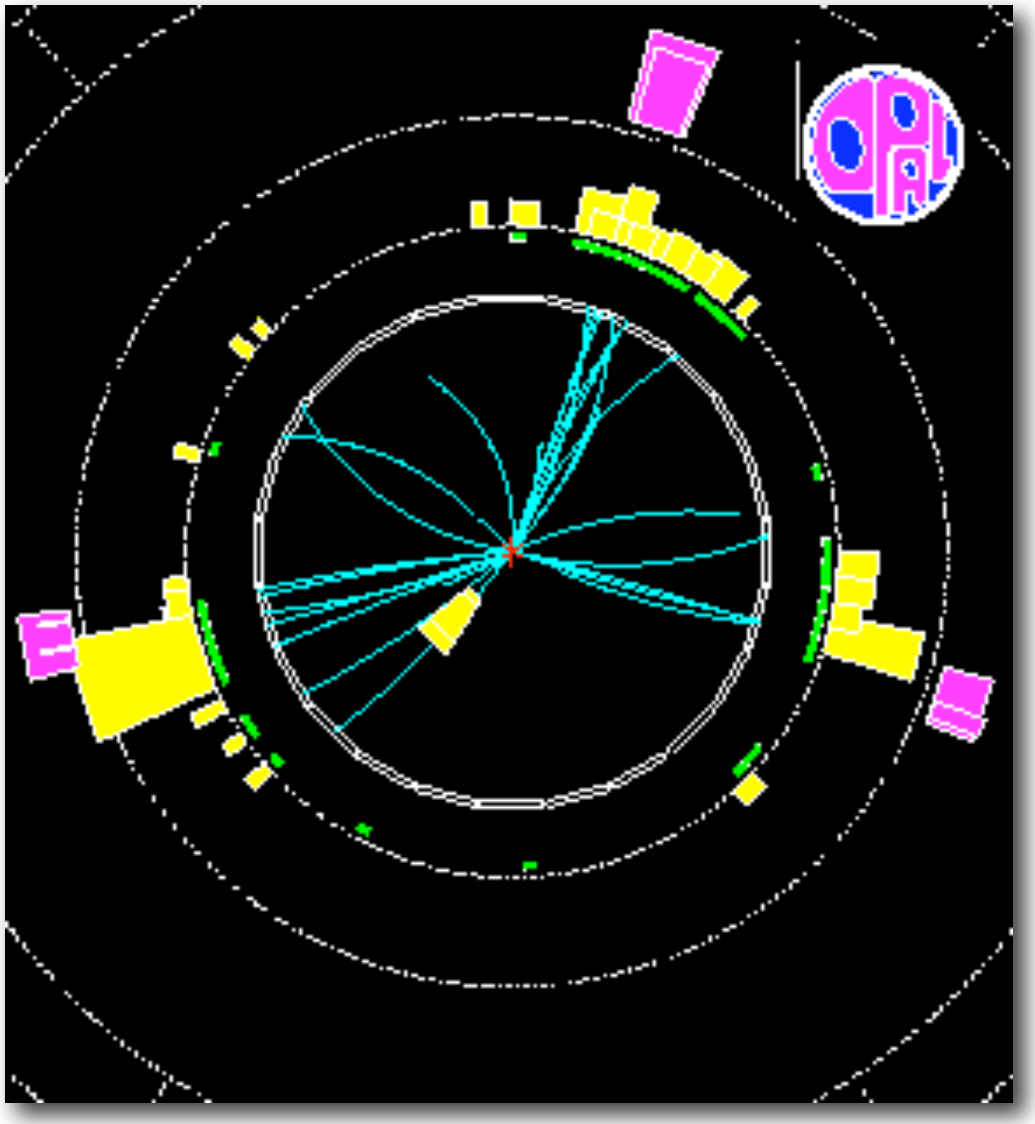
1979:

Three-jet events observed by TASSO, JADE, MARK J and PLUTO at PETRA in e^+e^- collisions at 27.4 GeV

Interpretation:
large angle emission of a hard gluon

Jets viewed as a proxy to the initial partons

Why jets



From PETRA to LEP

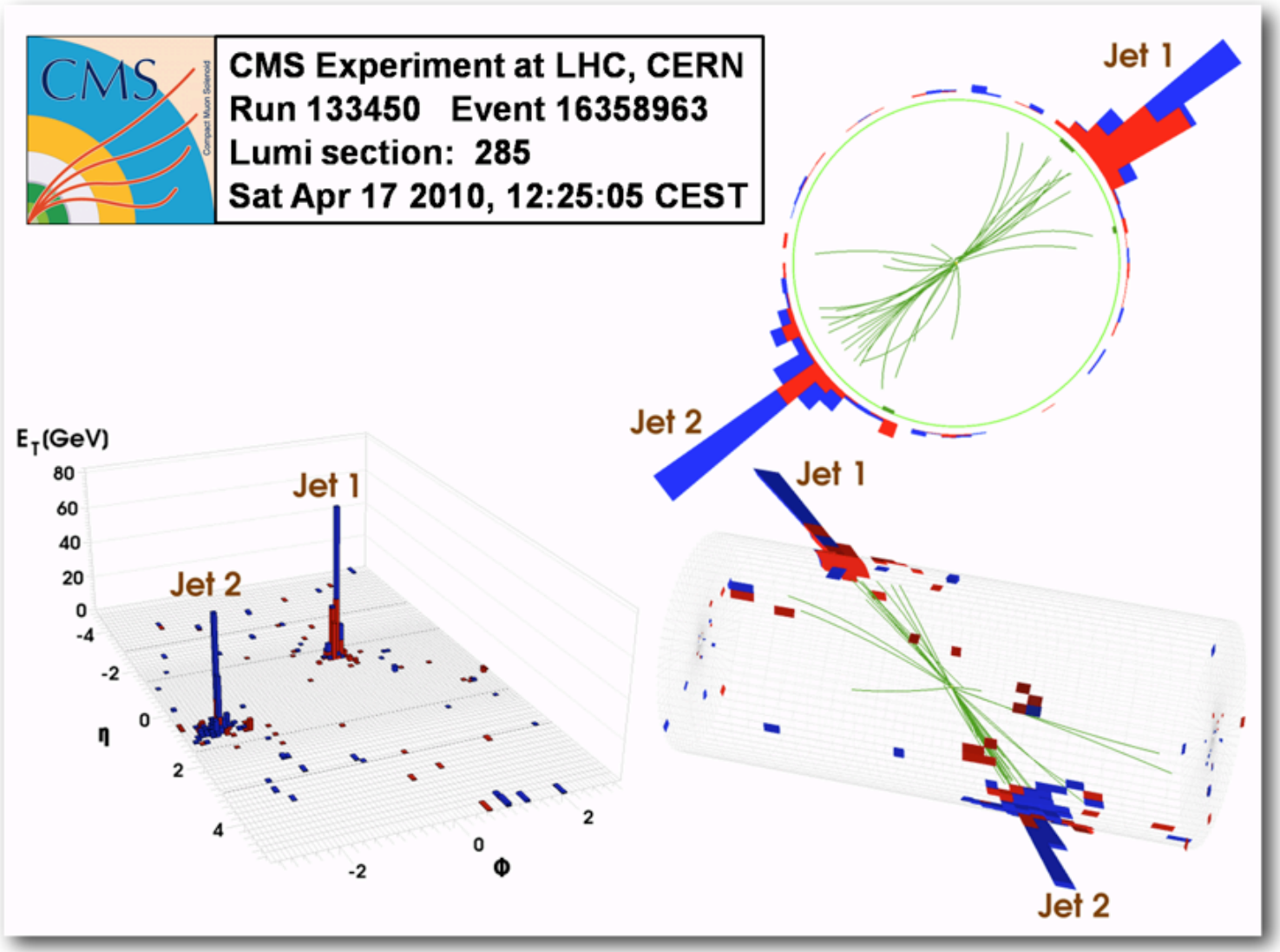
A **jet** is something that happens in high energy events:
a collimated bunch of hadrons flying roughly in the same direction

We could eyeball the collimated bunches, but it becomes impractical with millions of events

The classification of particles into jets is best done using a **clustering algorithm**

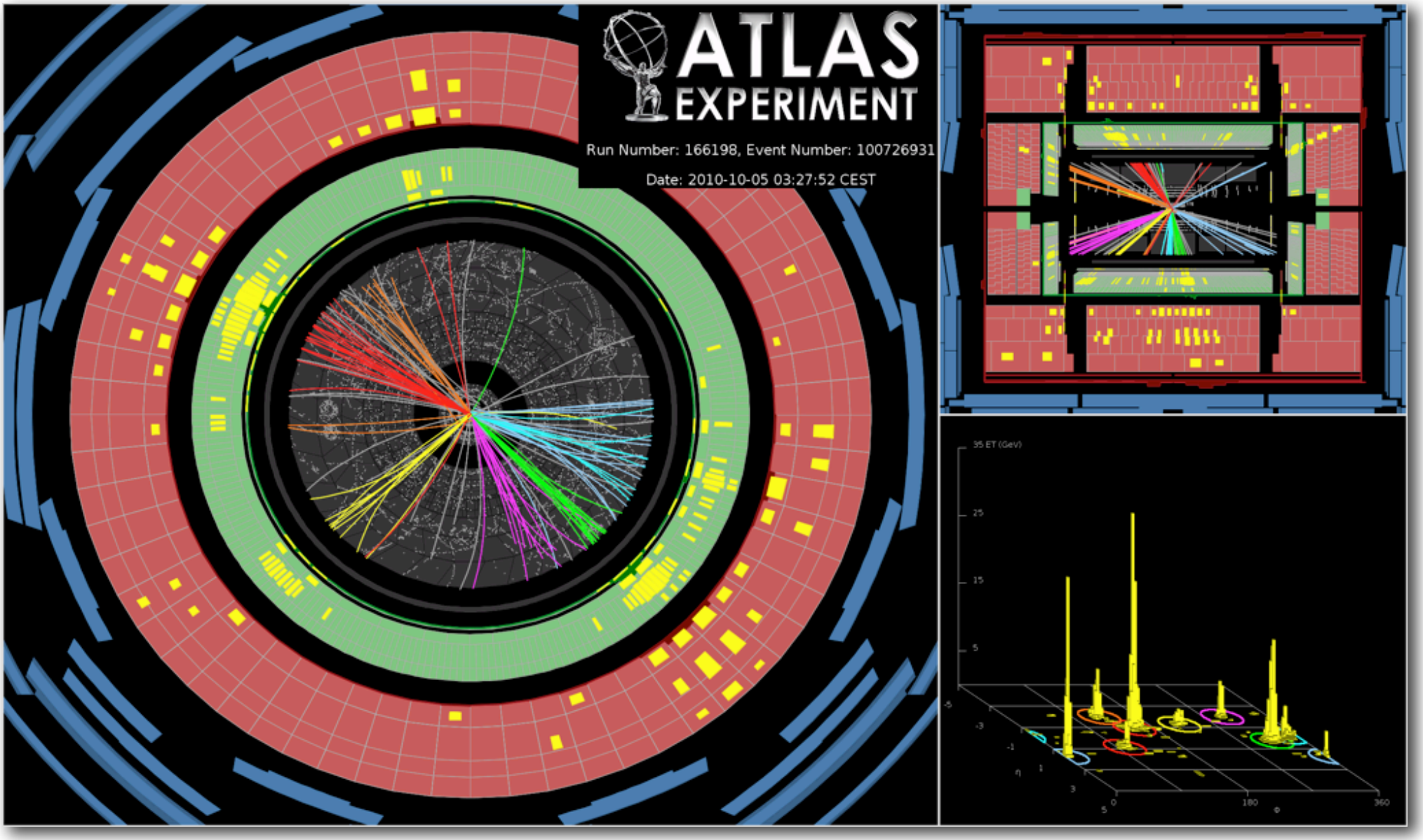
Jets @ LHC

A few decades after PETRA and LEP, the event displays got prettier, but jets are still pretty much the same



Dijet event from CMS

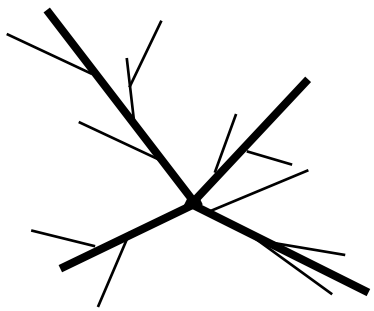
Jets @ LHC



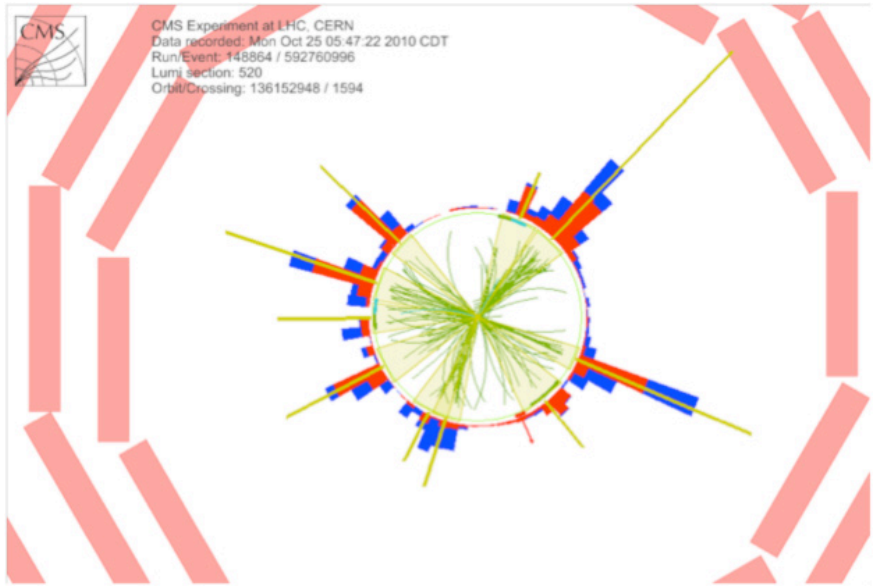
8(!) jets event from ATLAS

Taming reality

Multileg + PS



??



QCD predictions

Real data

Jets

One purpose of a 'jet clustering' algorithm is to
reduce the complexity of the final state, simplifying many hadrons
to **simpler objects** that one can hope to **calculate**

Clustering goes well beyond particle physics.

It is used in many fields (genetics, data mining, information retrieval, ...) as a form of ***unsupervised learning****: it is meant to* ***group similar objects together****, and therefore suggest a classification*

- *What does "similar" mean?*
  - *We need to define a dissimilarity function between two objects.*
    Almost infinite freedom of choice, as long as

    $D(A,B) = D(B,A)$ (symmetry)
    $D(A,B) = 0$ iff $A=B$ (self-similarity)
    $D(A,B) \geq 0$ (positivity)
    $D(A,B) \leq D(A,C) + D(B,C)$ (triangle inequality)

- *How do we group the objects together?*
  - *This is specified by the choice of a clustering algorithm*
    - *partitional*
    - *hierarchical*

Clustering algorithms

Partitional: construct partitions of the set of objects and evaluate them by some criterion.

X Number of clusters must be provided in advance

✓ Easy to implement, fast

Example: **k-means**. Choose K centroids at random, assign objects to them, recalculate centroids iterate until clusters are stable (i.e. they are all closest to the centroid of the cluster they belong too)

Hierarchical: create a hierarchical decomposition of the set of objects by some criterion

X Computationally heavy

✓ No need to specify number of clusters in advance

✓ 'In depth' view of structure (hierarchy).
Can decide a posteriori on a criterion for number of clusters.

Example: **hierarchical agglomerative clustering**. Combine objects iteratively starting from those with the smallest distance (i.e. largest similarity). Stop when a single cluster is left.

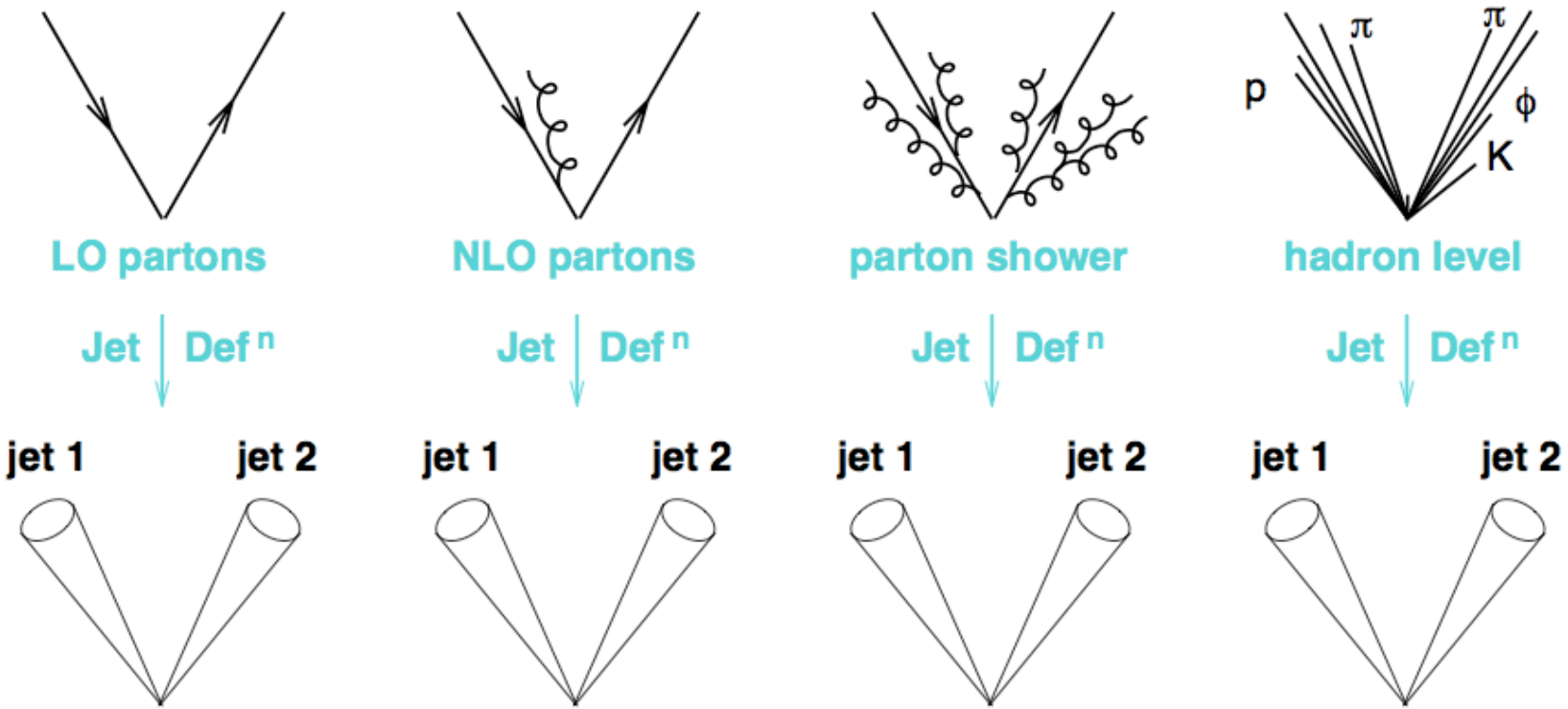
Jets in physics

- While we could take almost any clustering algorithm and, with a reasonable distance, use it to construct jets, i.e. clusters of hadrons, the result may not be particularly useful.
  We must also be guided by physics, so that
  - *the procedure leads to calculable results* → *infrared and collinear safety*
  - *the results are robust with respect to dynamics that we cannot calculate in detail* → *resiliency to hadronisation effects*

This puts strong constraints on the distances and algorithms that we can use

Jets as proxies

A good jet definition should be resilient to QCD effects



NB. 'Resiliency' does not mean 'total insensitivity'

A 'hadron jet' is **not** a parton

Most definitions will give very similar results (especially for inclusive observables), but it is important to be aware of potential differences, and not to compare apples with oranges.

IRC safety

Cancellation of singularities in total cross section

Born Virtual Real

$$\sigma_{tot} = \int_n |M_n^B|^2 d\Phi_n + \int_n |M_n^V|^2 d\Phi_n + \int_{n+1} |M_{n+1}^R|^2 d\Phi_{n+1}$$

CANCELLATION

A generic observable

$$\frac{d\sigma}{dX} = \int_n |M_n^B|^2 O(X; p_1, \ldots, p_n) d\Phi_n + \int_n |M_n^V|^2 O(X; p_1, \ldots, p_n) d\Phi_n + \int_{n+1} |M_{n+1}^R|^2 O(X; p_1, \ldots, p_n, p_{n+1}) d\Phi_{n+1}$$

In order to ensure the same cancellation existing in σ_{tot}, the definition of the observable $O(X; p_1, \ldots p_{n+1})$ **must not affect** the soft/collinear limit of the real emission term $|M^R_{n+1}|^2$, because it is there that the real/virtual cancellation takes place

IRC safety

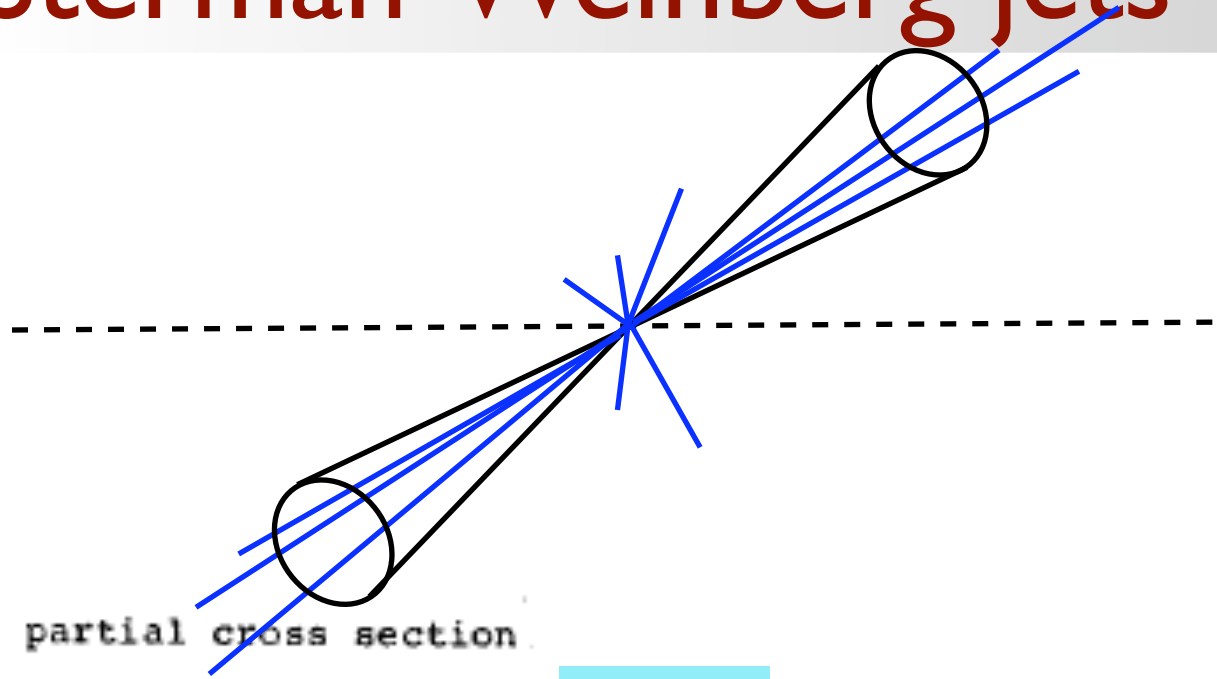
An observable is **infrared and collinear safe** if, in the limit of a **collinear splitting**, or the **emission of an infinitely soft** particle, the observable remains **unchanged:**

$$O(X; p_1, \ldots, p_n, p_{n+1} \to 0) \to O(X; p_1, \ldots, p_n)$$
$$O(X; p_1, \ldots, p_n \parallel p_{n+1}) \to O(X; p_1, \ldots, p_n + p_{n+1})$$

If we wish to be able to calculate a jet rate in perturbative QCD the jet algorithm that we use must be IRC safe:
soft emissions and collinear splittings must not change the hard jets

Sterman-Weinberg jets

The first rigorous definition of an **infrared and collinear safe** jet in QCD is due to Sterman and Weinberg, Phys. Rev. Lett. **39**, 1436 (1977):



To study jets, we consider the partial cross section $\sigma(E,\theta,\Omega,\varepsilon,\delta)$ for e^+e^- hadron production events, in which all but a fraction $\varepsilon \ll 1$ of the total e^+e^- energy E is emitted within some pair of oppositely directed cones of half-angle $\delta \ll 1$, lying within two fixed cones of solid angle Ω (with $\pi\delta^2 \ll \Omega \ll 1$) at an angle θ to the e^+e^- beam line. We expect this to be measur-

$$\sigma(E,\theta,\Omega,\varepsilon,\delta) = (d\sigma/d\Omega)_0\,\Omega\left[1 - (g_E^2/3\pi^2)\left\{3\ell n\,\delta + 4\ell n\,\delta\;\ell n\,2\varepsilon + \frac{\pi^3}{3} - \frac{5}{2}\right\}\right]$$

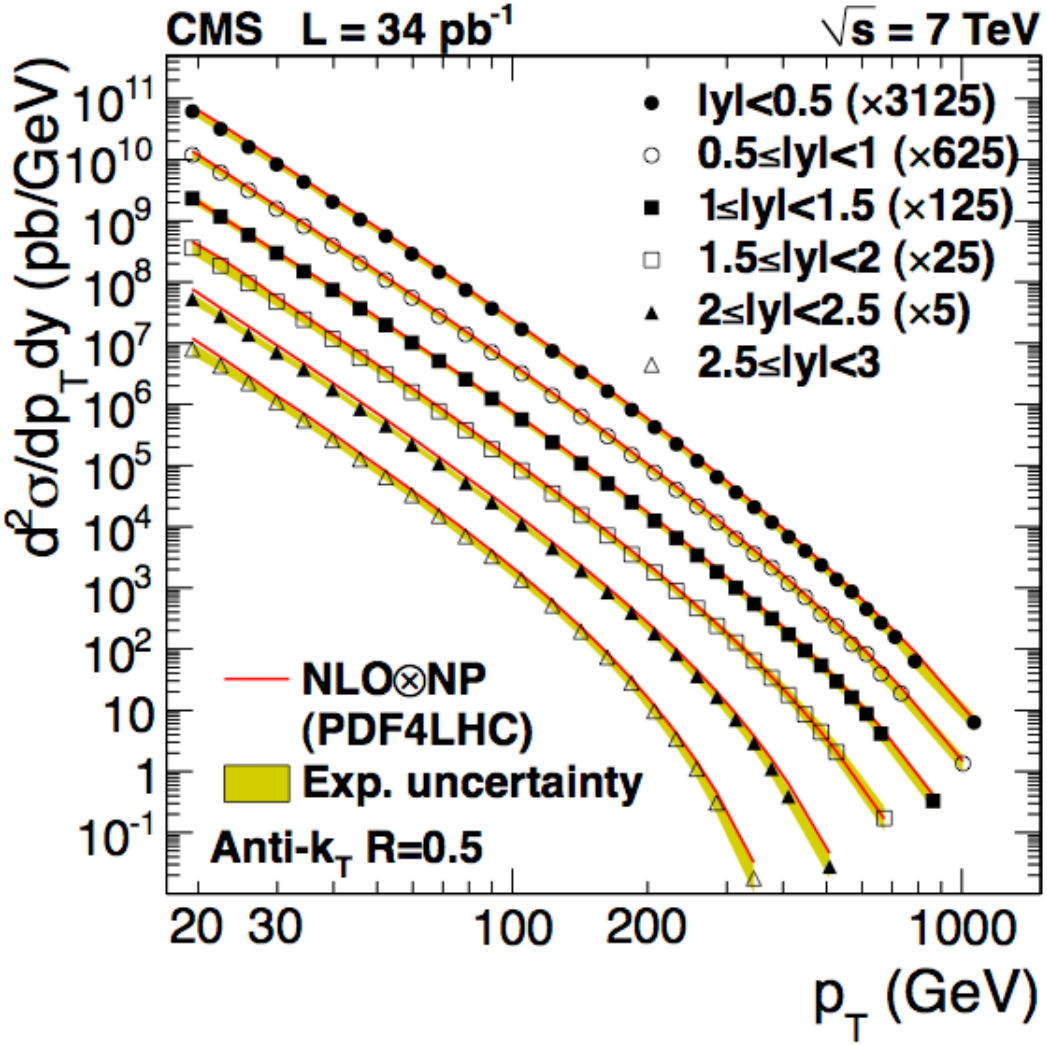
Calculable in pQCD (here is the result) but notice the soft and collinear large logs

Jets can serve two purposes

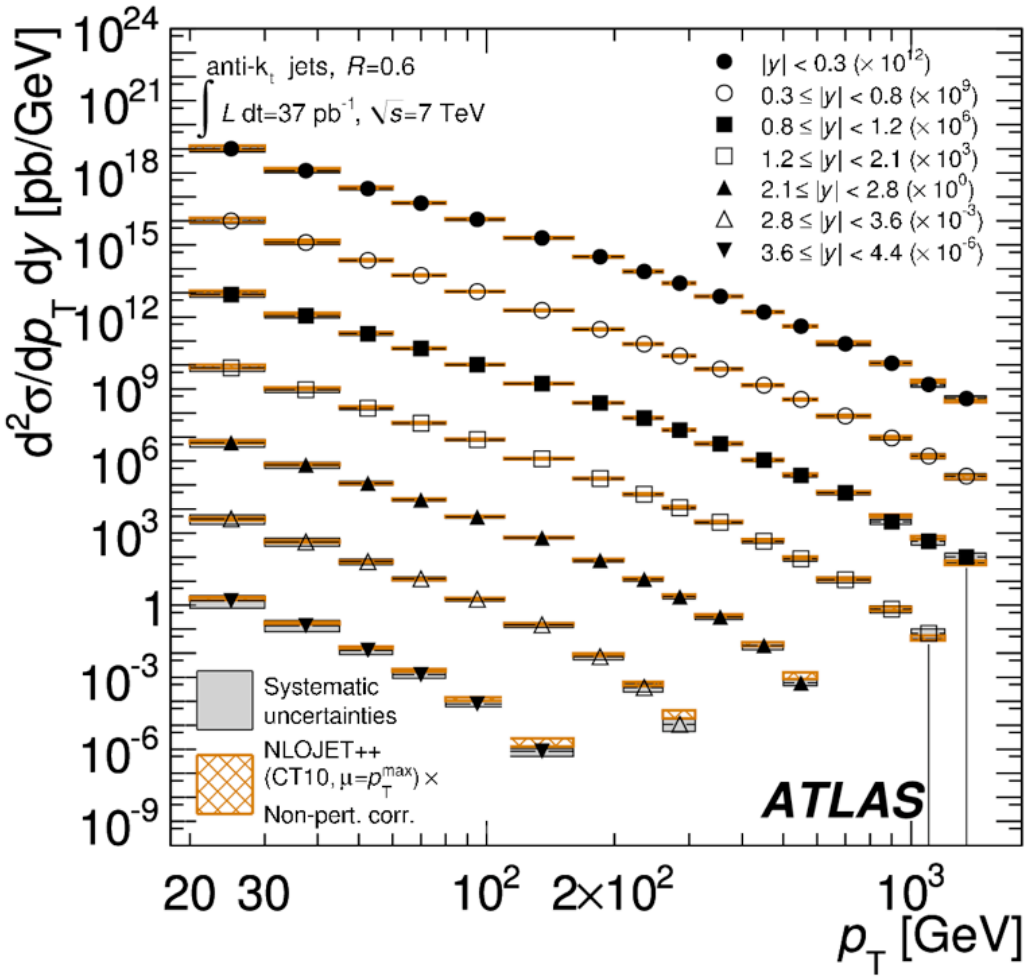
- They can be ***observables***, that one can measure and calculate
- They can be ***tools***, that one can employ to extract specific properties of the final state

Example of a jet observable

arXiv:1106.0208



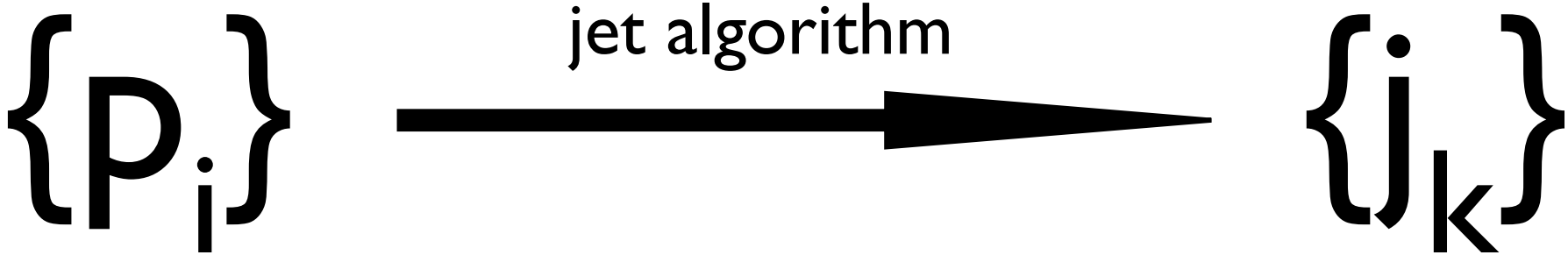
arXiv:1112.6297



Jets extensively measured in hadronic collisions.
Very good agreement with pQCD predictions over 10 orders of magnitude

Jet algorithm

A **jet algorithm** maps the momenta of the final state particles into the momenta of a certain number of jets:



particles,
4-momenta,
calorimeter towers, ...

jets

Most algorithms contain a resolution parameter, **R**, which controls the extension of the jet
(more about this later on)

Jet Definition

A jet algorithm
+
its parameters (e.g. R)
+
a recombination scheme
=
a **Jet Definition**

"*Jet [definitions] are legal contracts between theorists and experimentalists*"
-- MJ Tannenbaum

Two main classes of jet algorithms

- **Sequential recombination algorithms**

  Bottom-up approach: combine particles starting from **closest ones**

  How? Choose a **distance measure**, iterate recombination until few objects left, call them jets

  Works because of mapping closeness ⇔ QCD divergence

  Examples: Jade, k_t, Cambridge/Aachen, anti-k_t, …..

  → hierarchical clustering

- **Cone algorithms**

  Top-down approach: find coarse regions of energy flow.

  How? Find **stable cones** (i.e. their axis coincides with sum of momenta of particles in it)

  Works because QCD only modifies energy flow on small scales

  Examples: JetClu, MidPoint, ATLAS cone, CMS cone, SISCone…...

  → partitional clustering

Finding stable cones

In partitional-type algorithms, one wishes to find
the **stable configurations**:
axis of cones coincides with sum of 4-momenta of the particles it contains

The 'safe' way of doing so is to test
all possible combinations of N objects

Unfortunately, this takes $N2^N$ operations:
the age of the universe for only 100 objects

An approximate way out is to use seeds (e.g. à la k-means)
However, the final result can depend on the choice of the seeds and, such jet algorithms usually turn out to be **IRC unsafe**

Finding cones

Different procedures for placing the cones lead to **different cone algorithms**

NB: their properties and behaviour can **vastly differ**: there isn't **'a'** cone algorithm, but rather many of them

The main sub-categories of cone algorithms are:

- **Fixed** cone with **progressive removal** (FC-PR) (PyJet, CellJet, GetJet)
- **Iterative** cone with **progressive removal** (IC-PR) (CMS iterative cone)
- **Iterative** cone with **split-merge** (IC-SM) (JetClu, ATLAS cone)
- **IC-SM** with **mid-points** (IC_{mp}-SM) (CDF MidPoint, D0 Run II)
- **IC_{mp}** with **split-drop** (IC_{mp}-SD) (PxCone)
- **Seedless** cone with **split-merge** (SC-SM) (SISCone)

All, except SISCone, are infrared or collinear unsafe

Example of iterative cone

(This algorithm was used at the Tevatron, first under the name JetClu and, after a modification, MidPoint)

- Use **all particles** as seed
- Cluster particles into cone if $\Delta R < R$
- Iterate until stable (i.e. axis coincide with sum of momenta) cones found
- Split-merge step (see next slide)

Split-Merge

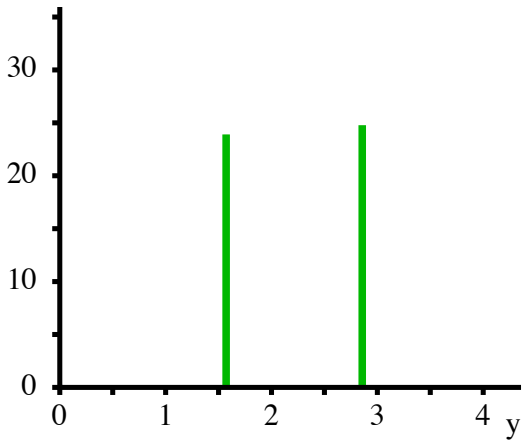
'Split-merge' is a further algorithm aimed at disentangling overlapping protojets.

The Tevatron Run II implementation goes like this:

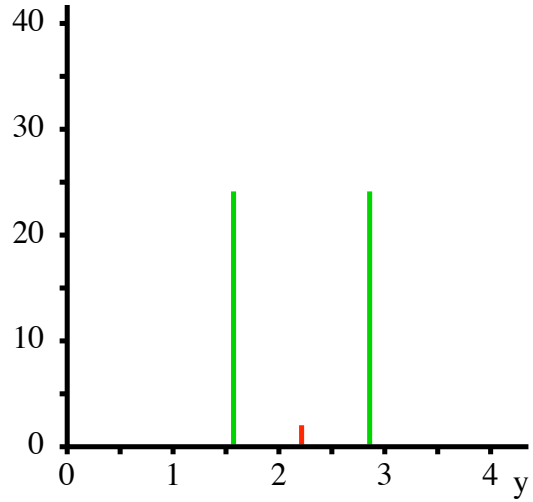
- Choose an **overlap threshold *f***
- Find hardest protojet
- Find hardest other protojet overlapping with it
- Merge is they share a fraction of momentum larger than *f*, split along axis at centre otherwise
- (Call protojet a jet if there are no overlapping protojets)

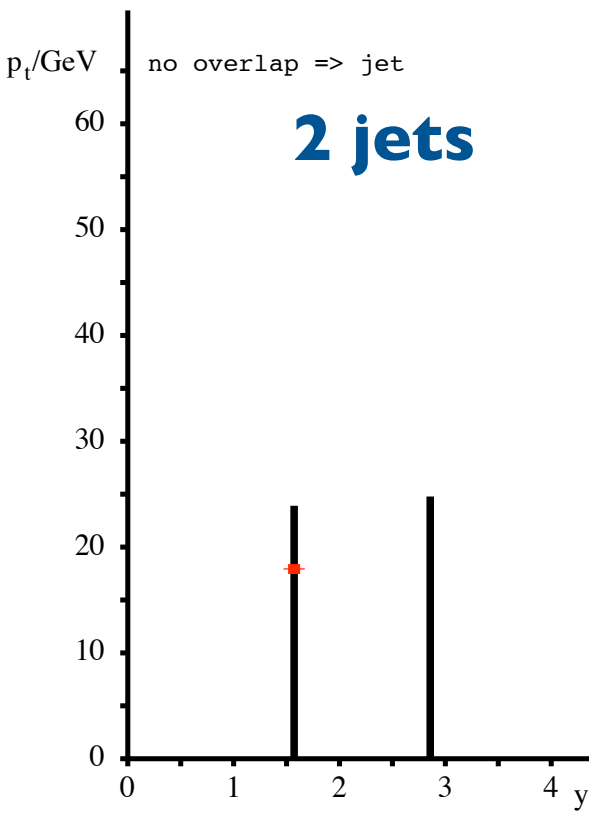
Call this an "**IC-SM**" type of cone
(iterative cone with split-merge step at the end)

IC-SM infrared unsafety

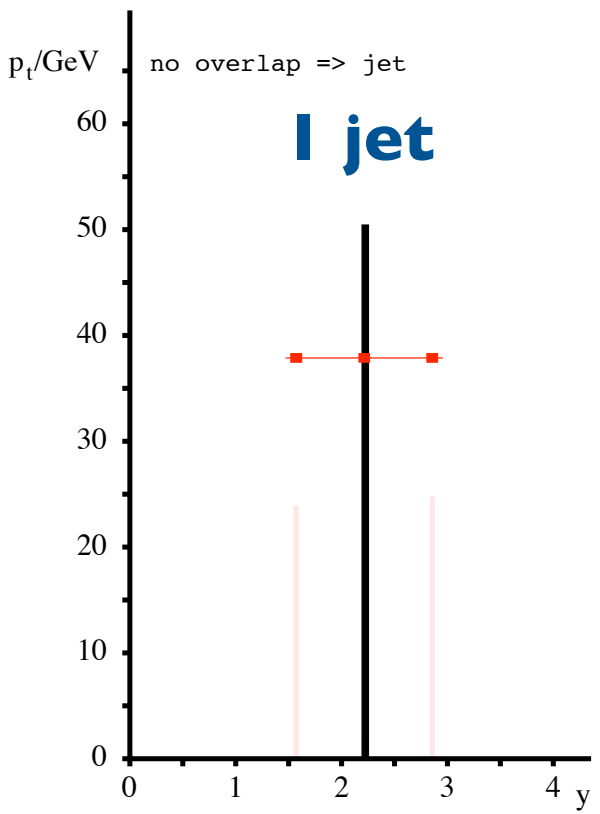


Add a
soft particle





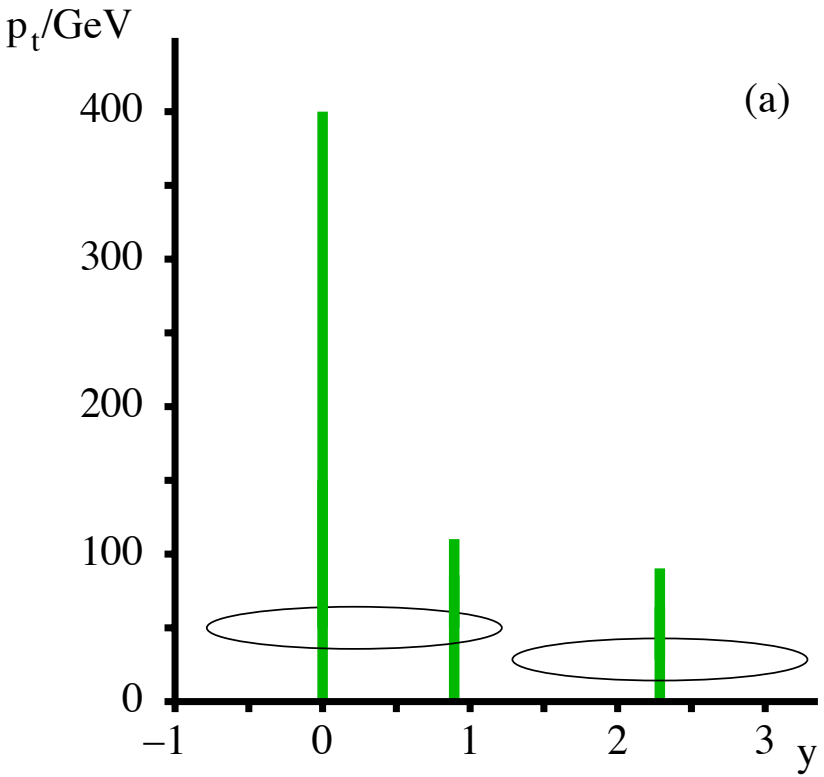
Final state jets differ



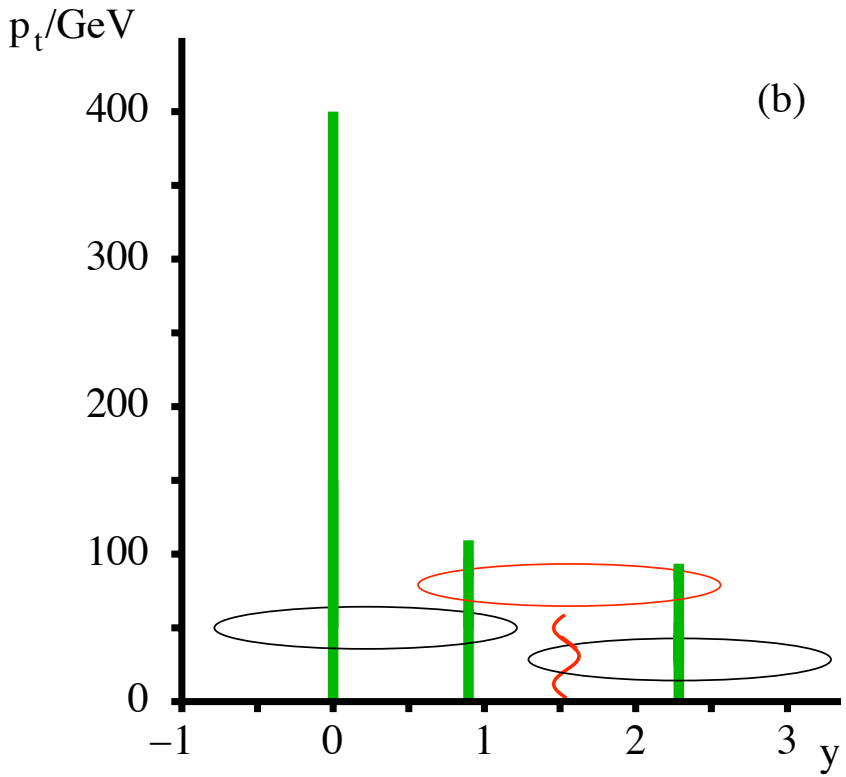
MidPoint (IC_{mp}-SM) infrared unsafety

MidPoint fixes the two-particle configuration IR-safety problem by **adding midpoints** to list of seeds.

But this merely **shifts the problem to three-particle configurations**



Three hard particles clustered into **two** cones by the MidPoint algorithm



Addition of a **soft** particle changes the hard jets: **three** stable cones are now found

The problem is that the stable-cone search procedure used by seeded IC algorithms often cannot find **all** possible stable cones

IRC safety in real life

Strictly speaking, one needs IRC safety not so much to find jets, but to be able to calculate them in pQCD

If you are not interested in theory/data comparisons, you may think of doing well enough with an IRC-unsafe jet algorithm

However

- Detectors may split/merge collinear particles, and be poorly understood for soft ones
- High luminosity (or heavy ions collisions) add a lot of soft particles to hard event

IRC safety provides resiliency to such effects
(plus, at some point in the future you may wish to compare your measurement to a calculation)

Finding stable cones

In partitional-type algorithms, one wishes to find the **stable configurations**:

axis of cones coincides with sum of 4-momenta of particles it contains.

The 'safe' way of doing so is to test **all possible combinations** of N objects

Unfortunately, this takes $N2^N$ operations: the age of the universe for only 100 objects

~~An approximate way out is to use seeds (e.g. à la k-means) However, the final result can depend on the choice of the seeds and, such jet algorithms usually turn out to be **IRC unsafe**~~

SISCone is guaranteed to find **ALL stable cones** (via a **fast** geometric implementation, G. Salam and G. Soyez 0704.0292) and therefore leads to an **IRC-safe** cone jet algorithm

Cones Infrared (un)safety

Q: How often are the hard jets changed by the addition of a soft particle?

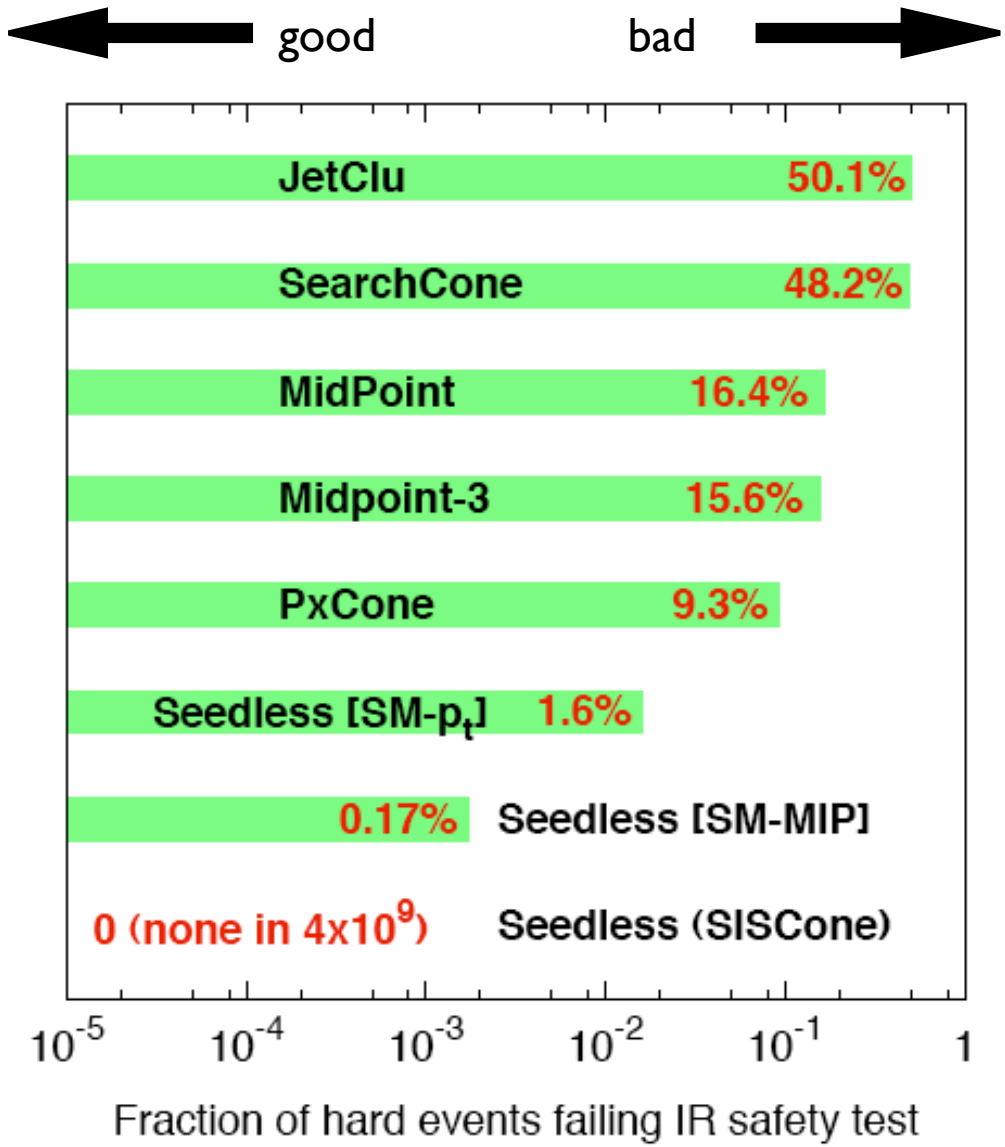
- Generate event with $2 < N < 10$ hard particles, find jets
- Add $1 < N_{soft} < 5$ soft particles, find jets again

A: [repeatedly]

- If the jets are different, algorithm is IR unsafe.

Unsafety level	failure rate
2 hard + 1 soft	$\sim 50\%$
3 hard + 1 soft	$\sim 15\%$
SISCone	IR safe !

Be careful with split–merge too



Recombination algorithms

- First introduced in e^+e^- collisions in the '80s
- Typically they work by calculating a **'distance'** between particles, and then recombine them pairwise according to a given order, until some condition is met (e.g. no particles are left, or the distance crosses a given threshold)

IRC safety can usually be seen to be trivially guaranteed

JADE algorithm

distance:

$$y_{ij} = \frac{2E_iE_j(1-\cos\theta_{ij})}{Q^2}$$

- Find the minimum y_{min} of all y_{ij}
- If y_{min} is below some jet resolution threshold y_{cut}, recombine i and j into a single new particle ('pseudojet'), and repeat
- If no $y_{min} < y_{cut}$ are left, all remaining particles are jets

Problem of this particular algorithm:
two soft particles emitted at large angle get easily recombined into a single jet:
counterintuitive and perturbatively troublesome

e^+e^- k_t (Durham) algorithm

[Catani, Dokshitzer, Olsson, Turnock, Webber '91]

Identical to JADE, but with distance:

$$y_{ij} = \frac{2\min(E_i^2, E_j^2)(1-\cos\theta_{ij})}{Q^2}$$

In the collinear limit, the numerator reduces to the **relative transverse momentum** (squared) of the two particles, hence the name of the algorithm

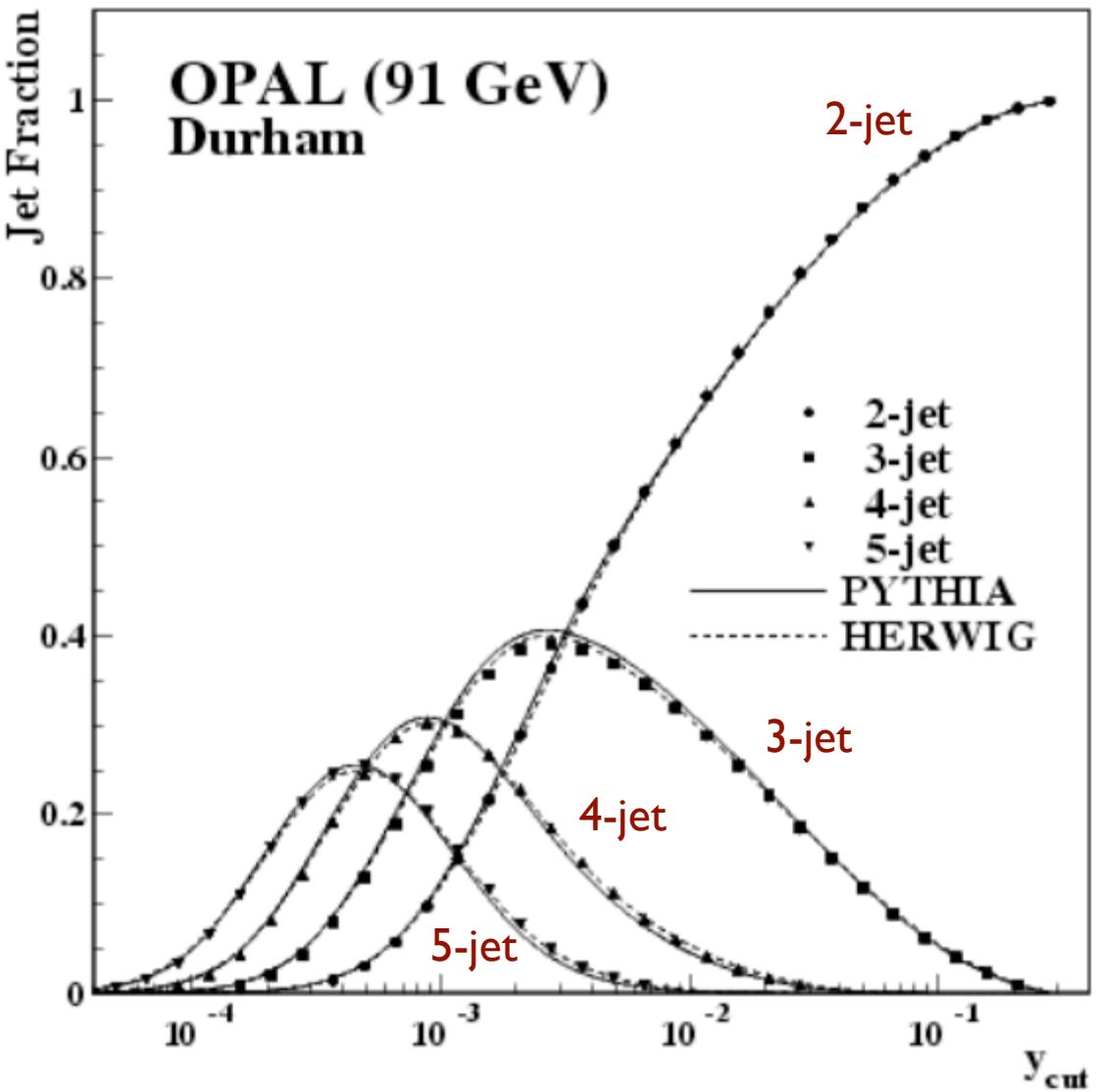
The use of the min() avoids the problem of recombination of back-to-back particles present in JADE: a soft and a hard particle close in angle are 'closer' than two soft ones at large angle

One key feature of the k_t algorithm is its relation to the structure of QCD divergences:

$$\frac{dP_{k\to ij}}{dE_i d\theta_{ij}} \sim \frac{\alpha_s}{\min(E_i, E_j)\theta_{ij}}$$

The k_t algorithm inverts the QCD branching sequence (the pair which is recombined first is the one with the largest probability to have branched)

e^+e^- k_t (Durham) algorithm in action



Characterise events
in terms of number of jets
(as a function of y_{cut})

Resummed calculations for distributions of y_{cut} doable with the k_t algorithm

k_t algorithm in hadron collisions

(Inclusive and longitudinally invariant version)

$$d_{ij} = \min(p_{ti}^2, p_{tj}^2)\frac{\Delta R_{ij}^2}{R^2} \qquad d_{iB} = p_{ti}^2$$

- Calculate the distances between the particles: $\mathbf{d_{ij}}$
- Calculate the beam distances: $\mathbf{d_{iB}}$
- Combine particles with **smallest distance** d_{ij} or, if d_{iB} is smallest, call it a jet
- Find again smallest distance and repeat procedure until no particles are left (this stopping criterion leads to the *inclusive* version of the k_t algorithm)

- Given N particles this is, naively, an $O(N^3)$ algorithm: calculate N^2 distances, repeat for all N iterations. 1 second to cluster 1000 particles: too slow for practical use.
- An NlnN implementation exists: 1ms for 1000 particles. Can even use it in the trigger.

The k_t algorithm and its siblings

One can generalise the k_t distance measure:

$$d_{ij} = \min(k_{ti}^{2p}, k_{tj}^{2p}) \frac{\Delta y^2 + \Delta\phi^2}{R^2} \qquad d_{iB} = k_{ti}^{2p}$$

p = 1 k_t algorithm — S. Catani, Y. Dokshitzer, M. Seymour and B. Webber, Nucl. Phys. B406 (1993) 187; S.D. Ellis and D.E. Soper, Phys. Rev. D48 (1993) 3160

p = 0 Cambridge/Aachen algorithm — Y. Dokshitzer, G. Leder, S.Moretti and B. Webber, JHEP 08 (1997) 001; M. Wobisch and T. Wengler, hep-ph/9907280

p = -1 **anti-k_t algorithm** — MC, G. Salam and G. Soyez, arXiv:0802.1189

NB: in anti-kt pairs with a **hard** particle will cluster first: if no other hard particles are close by, the algorithm will give **perfect cones**

Quite ironically, a sequential recombination algorithm is the 'perfect' cone algorithm

IRC safe algorithms

k_t	SR $d_{ij} = \min(k_{ti}^2, k_{tj}^2)\Delta R_{ij}^2/R^2$ hierarchical in rel p_t	Catani et al '91 Ellis, Soper '93	NlnN
Cambridge/ Aachen	SR $d_{ij} = \Delta R_{ij}^2/R^2$ hierarchical in angle	Dokshitzer et al '97 Wengler, Wobish '98	NlnN
anti-k_t	SR $d_{ij} = \min(k_{ti}^{-2}, k_{tj}^{-2})\Delta R_{ij}^2/R^2$ gives perfectly conical hard jets	MC, Salam, Soyez '08 (Delsart, Loch)	$N^{3/2}$
SISCone	Seedless iterative cone with split-merge gives 'economical' jets	Salam, Soyez '07	N^2lnN

'second-generation' algorithms

All are available in FastJet, http://fastjet.fr

(As well as many IRC unsafe ones)

Jet clustering in FastJet

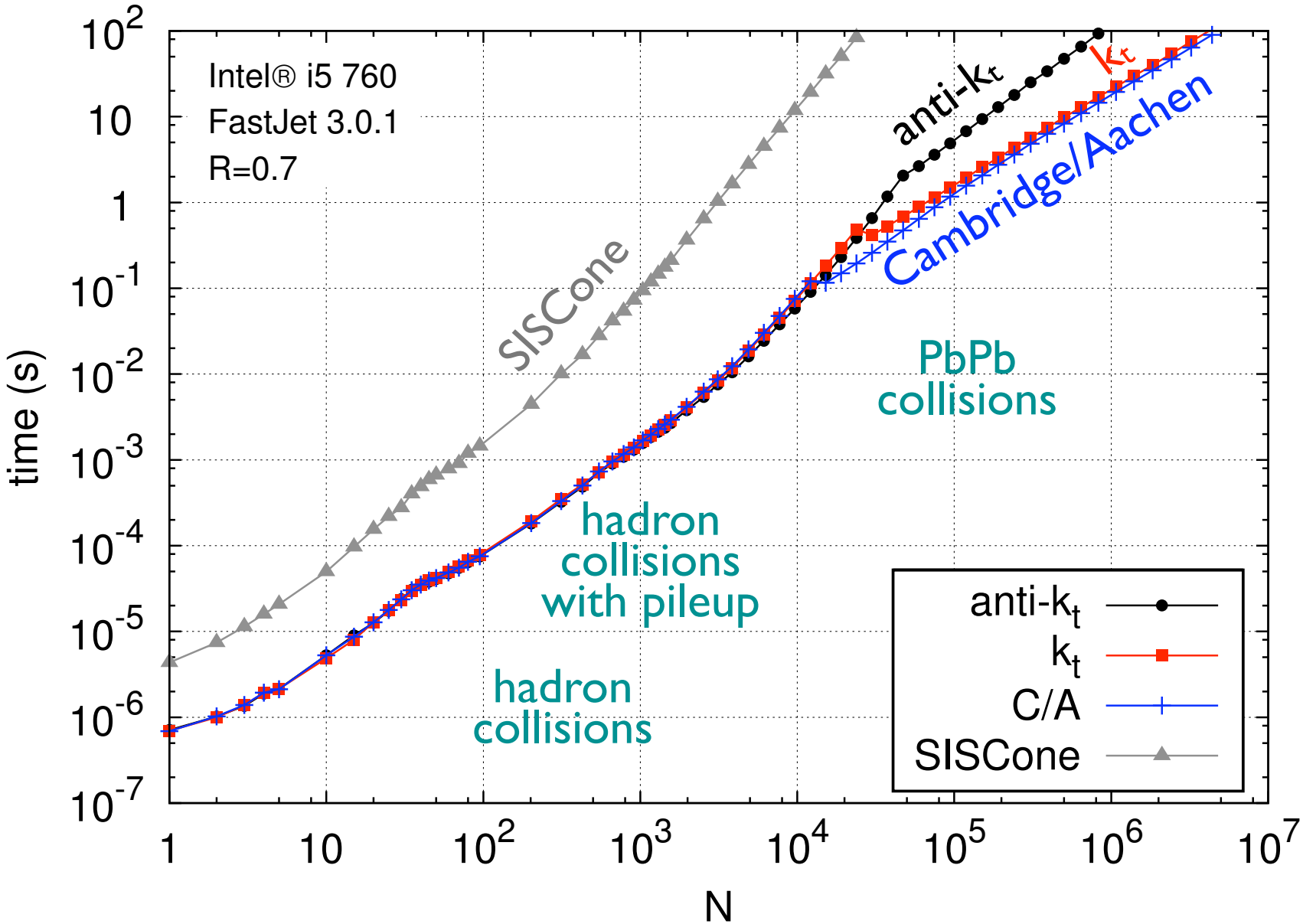
```
/// define a jet definition
JetDefinition jet_def(JetAlgorithm jet_algorithm,
                      double R,
                      RecombinationScheme rec_sch = E_scheme);
```

jet_algorithm can be any one of the four IRC safe algorithms, or also most of the old IRC-unsafe ones, for legacy purposes

```
/// create a ClusterSequence, extract the jets
ClusterSequence cs(input_particles, jet_def);
vector<PseudoJet> jets = sorted_by_pt(cs.inclusive(jets));
...
// pt of hardest jet
double pt_hardest = jets[0].perp();
...
// constituents of hardest jet
vector<PseudoJet> constits = jets[0].constituents();
```

FastJet speed test

Time needed to cluster an event with N particles

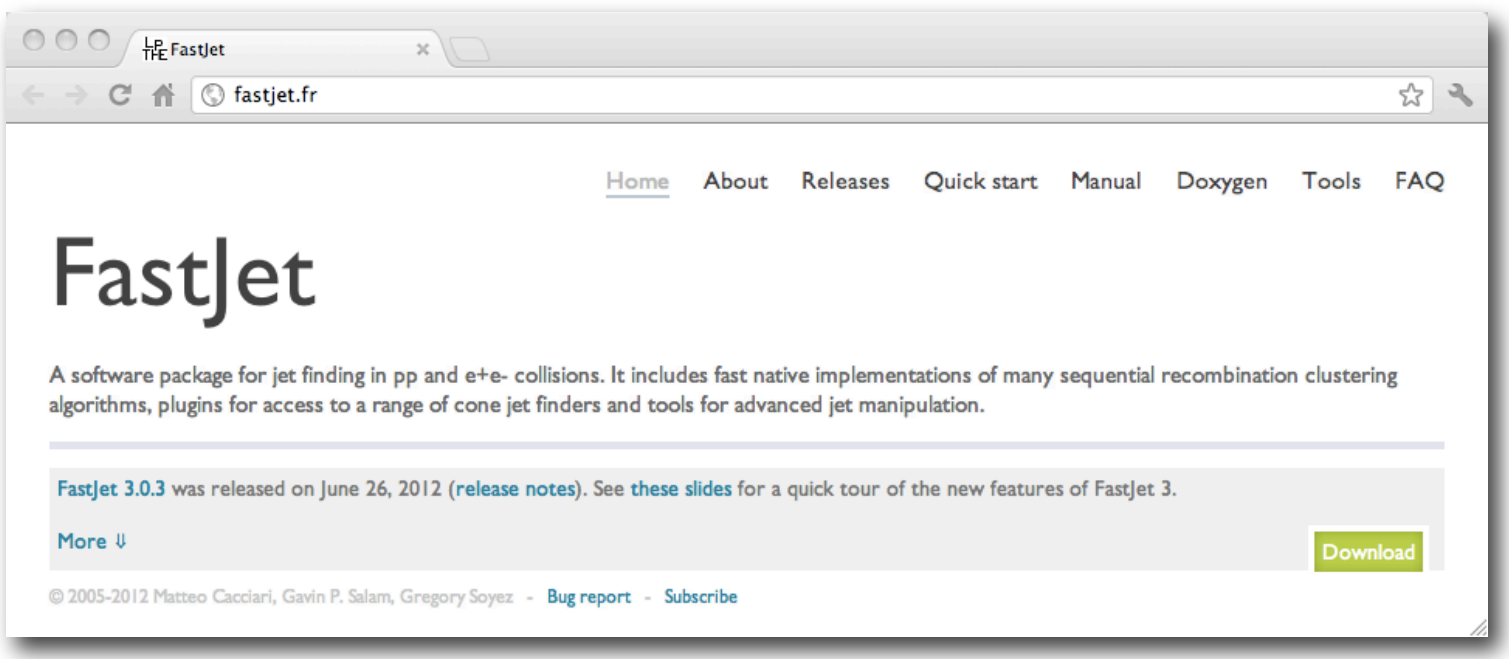


Summary

- A vast zoology of jet algorithms has been reduced in the past few years to **4 infrared and collinear safe algorithms**
  - All are implemented in an efficient and fast way
  - Of these, **anti-k_t** is used by all the LHC collaborations as their main algorithm for "finding" jets and measuring inclusive cross sections
- The four algorithms have quite different characteristics, which makes them non easily swappable when specific properties are needed for specific tasks. On the other hand, chances are that one can chose the algorithm which is most appropriate for a specific job

Beware, advert

www.fastjet.fr

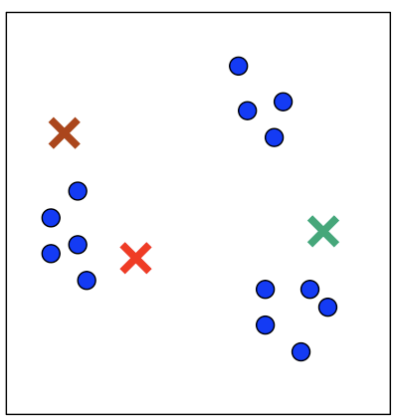


Backup slides

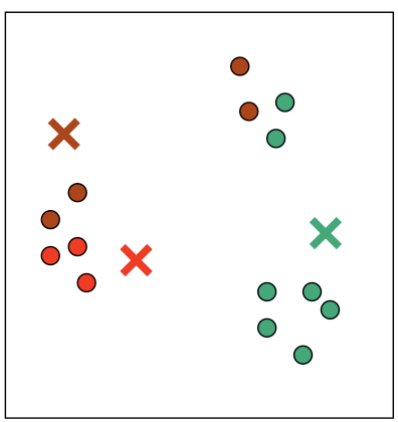
k-means

Example of a partitional algorithm

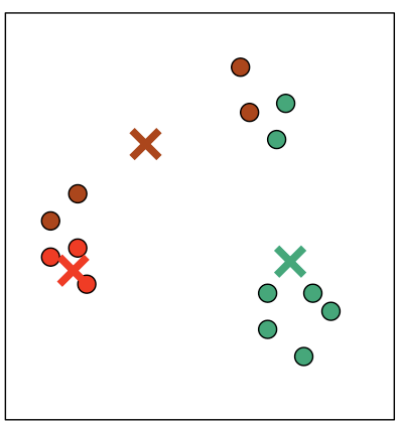
1) Choose K centroids at random
2) Assign objects to closest centroid, forming K clusters
3) Calculate centroid (mean of distances) of each cluster, update centroids
4) Check if an object in a cluster is closer to another centroid.
   Reallocate in case.
5) Repeat from step 3 until no object changes cluster anymore.



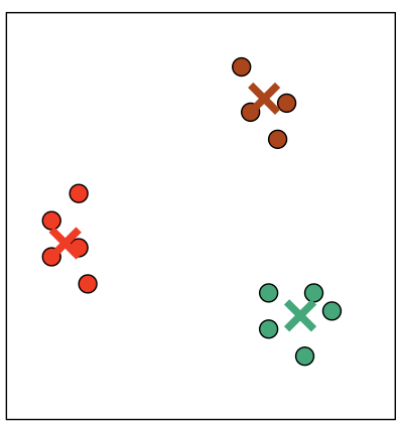
Step 1
(random centroids)



Step 2
(allocate objects)



Step 3
(move centroids)



Step 5
(end of iteration)

[Figures by E. Garrett-Mayer]

One of the main shortcomings:

result of final convergence can be highly sensitive to choice of initial seeds.

Also, the concept of 'mean distance' (to calculate the centroid) must be defined.

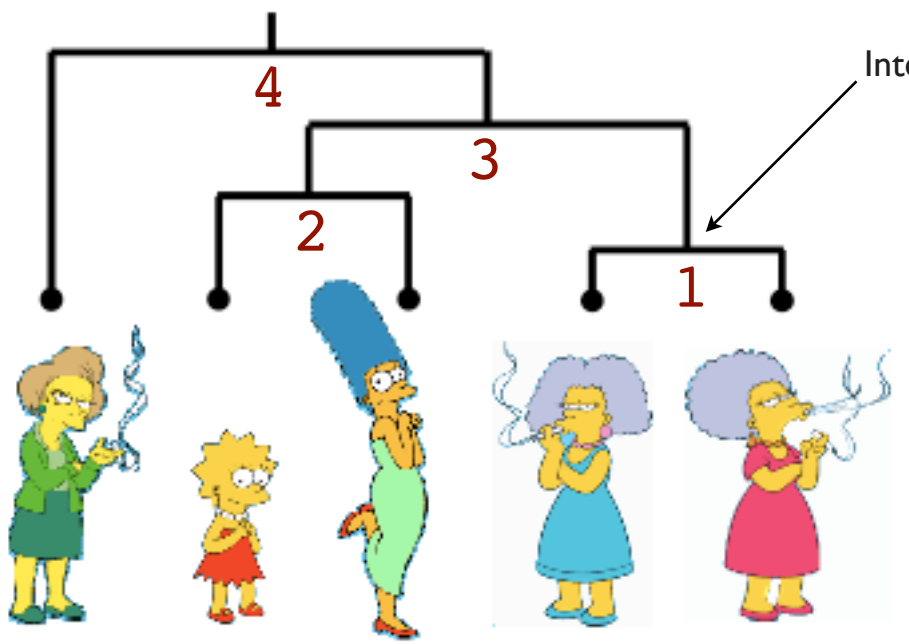
Agglomerative clustering

Example of a hierarchical algorithm

(often denoted HAC: Hierarchical Agglomerative Clustering)

1) Choose a dissimilarity function, calculate distance matrix between all objects
2) Choose a linkage method
3) **Cluster** two objects with **smallest** dissimilarity
4) Update the distance matrix
5) Repeat from step 3 until a single cluster is left
6) Look at the resulting hierarchy, and decide what 'best' number of clusters is

Dendrogram



Similarity between two objects is represented by the **height** of the lowest internal node that they share.

Order of clustering is 1,2,3,4