Terascale Monte Carlo School
DESY, March 2011

Jet Physics

Matteo Cacciari
LPTHE - Paris 6,7 and CNRS

Contains original artwork
(as well as many ideas)
from Gavin Salam

Outline

- Basics of jets and jets algorithms
  - Infrared and collinear (un)safety
  - Cone-type (progressive removal, split-merge, SISCone)
  - Recombination-type (k_t, Cambridge/Aachen, anti-k_t)

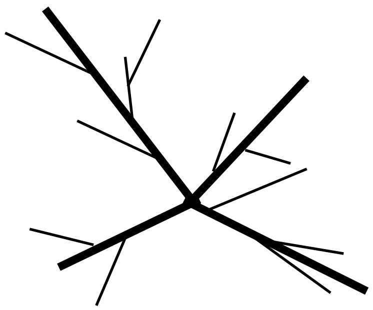
- More fun with jets, physics with jets, and jetography
  - Jet areas and background subtraction
  - Quality measures (mass-peak reconstruction)
  - Third-generation algorithms (taggers, filters,...): jetography

Bibliography

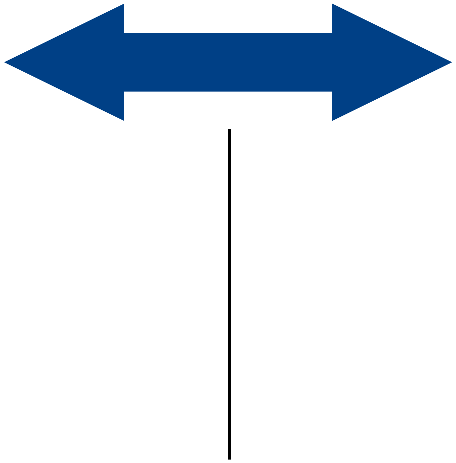
- Les Houches 2007 proceedings, arXiv:0803.0678
- Gavin Salam's lectures at CTEQ-MCnet 2008
- Gavin Salam, 'Towards Jetography', arXiv:0906.1833
- Boosted objects: a probe of beyond the standard model physics, arXiv:1012.5412

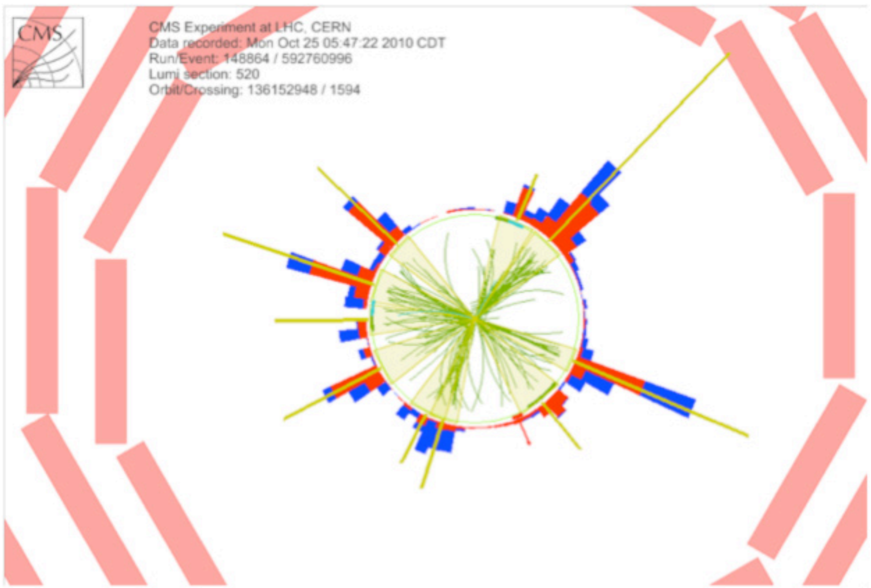
Taming reality

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**

Jet algorithm

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

$$\{p_i\} \xrightarrow{\text{jet algorithm}} \{j_k\}$$

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)

www.fastjet.fr

FastJet

The FastJet package, written by Matteo Cacciari, Gavin Salam and Gregory Soyez, provides a fast implementation of the longitudinally invariant kt [1,2] longitudinally invariant inclusive Cambridge/Aachen [3,4] and anti-kt [7] jet finders and a uniform interface to external jet finders (notably SISCone [5]) via a plugin mechanism. It also includes tools for calculating jet areas [8] and performing background (pileup/UE) subtraction [9].

Native jet-finding is based on the geometrical methods described in Phys. Lett. B **641** (2006) 57 [hep-ph/0512210] and [6].

***** NEW: Alpha preview for 3.0 series: fastjet-3.0alpha2, released March 10, 2011 (release notes) ***** It includes a preliminary subset of the features planned for inclusion in the 3.0 series.

***** Current version: fastjet-2.4.2, released February 26, 2010 (release notes) *****

Main new features in the 2.4.x series

- Addition of several new pp algorithms: D0 run II cone, ATLAS Cone, TrackJet, CMS Iterative cone (all as plugins) and generalised k_t (native).
- Introduction of e^+e^- algorithms: k_t and generalised k_t (native), as well as Jade, Cambridge and spherical SISCone (plugins).
- Also: new way of accessing jet substructure, some improvements in tools related to background estimation, facilities for easier implementation of new sequential-recombination algorithms.
- Backwards compatibility note: for a number of jet definitions, certain misleading default values have been removed from the constructor.

Additionally, version 2.4.2 fixes some mostly minor bugs and an issue of differing results on 32 and 64 bit architectures with the D0RunIICone. For more details, see the release notes.

FastJet resources

- **Main page**
- Download v2.4.2
- Manual
- Doxygen
- All releases
- Jet algorithm list
- Quick start guide
- Tools *(devel)*
- FAQ

Alpha release

- Download 3.0alpha2
- Manual for 3.0alpha2
- Doxygen for 3.0alpha2

A jet algorithm

+

its parameters (e.g. R)

+

a recombination scheme

=

a **Jet Definition**

In FastJet

```
/// JetDefinition constructor
JetDefinition(JetAlgorithm jet_algorithm,
              double R,
              RecombinationScheme recomb_scheme = E_scheme);
```

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

Observables

Snowmass

FERMILAB-Conf-90/249-E
[E-741/CDF]

Toward a Standardization of Jet Definitions *

* To be published in the proceedings of the 1990 Summer Study on High Energy Physics, *Research Directions for the Decade*, Snowmass, Colorado, June 25 - July 13, 1990.

Several important properties that should be met by a jet definition are [3]:

1. Simple to implement in an experimental analysis;
2. Simple to implement in the theoretical calculation;
3. Defined at any order of perturbation theory;
4. Yields finite cross section at any order of perturbation theory;
5. Yields a cross section that is relatively insensitive to hadronization.

Speed

Infrared and collinear safety

[Addition of a soft particle or a collinear splitting should not change final hard jets]

Snowmass set standards, but didn't provide solutions

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, …..

- **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…..

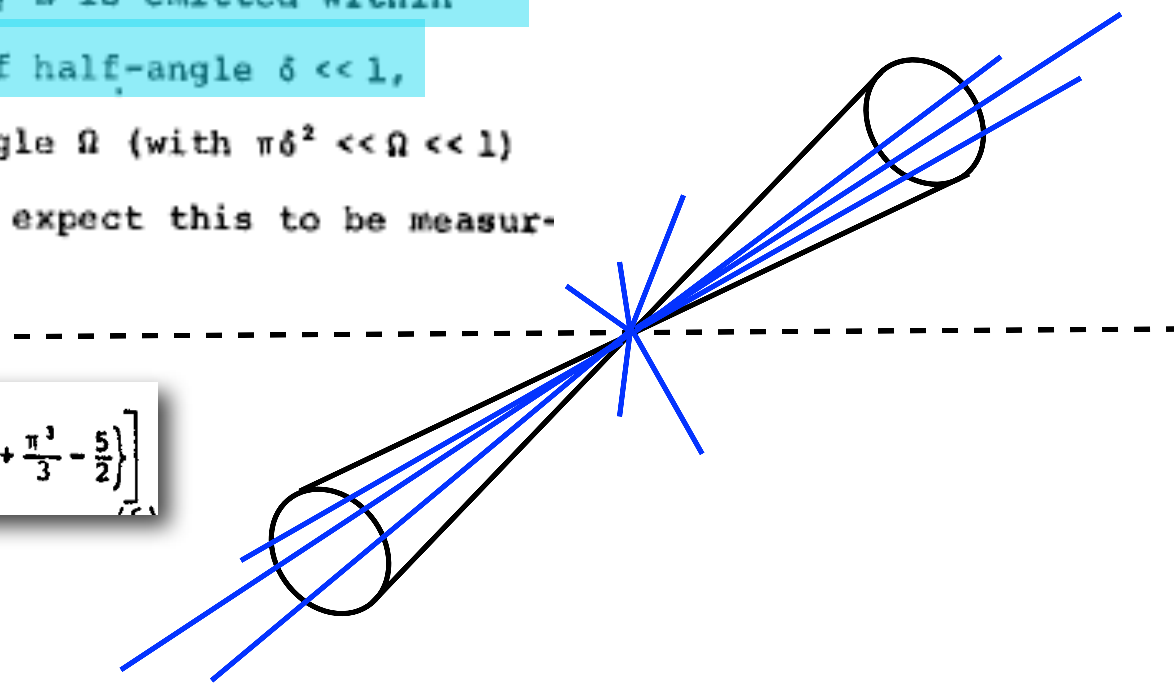
Cone algorithms

The first rigorous definition of cone jets in QCD is due to Sterman and Weinberg

Phys. Rev. Lett. **39**, 1436 (1977)

To study jets, we consider the partial cross section $\sigma(E,\theta,\Omega,\epsilon,\delta)$ for e^+e^- hadron production events, in which all but a fraction $\epsilon \ll 1$ 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-

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



Good for 2 jets and e^+e^- collisions

In more general cases, **where** do we place the cones? **How many**?

Finding cones

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

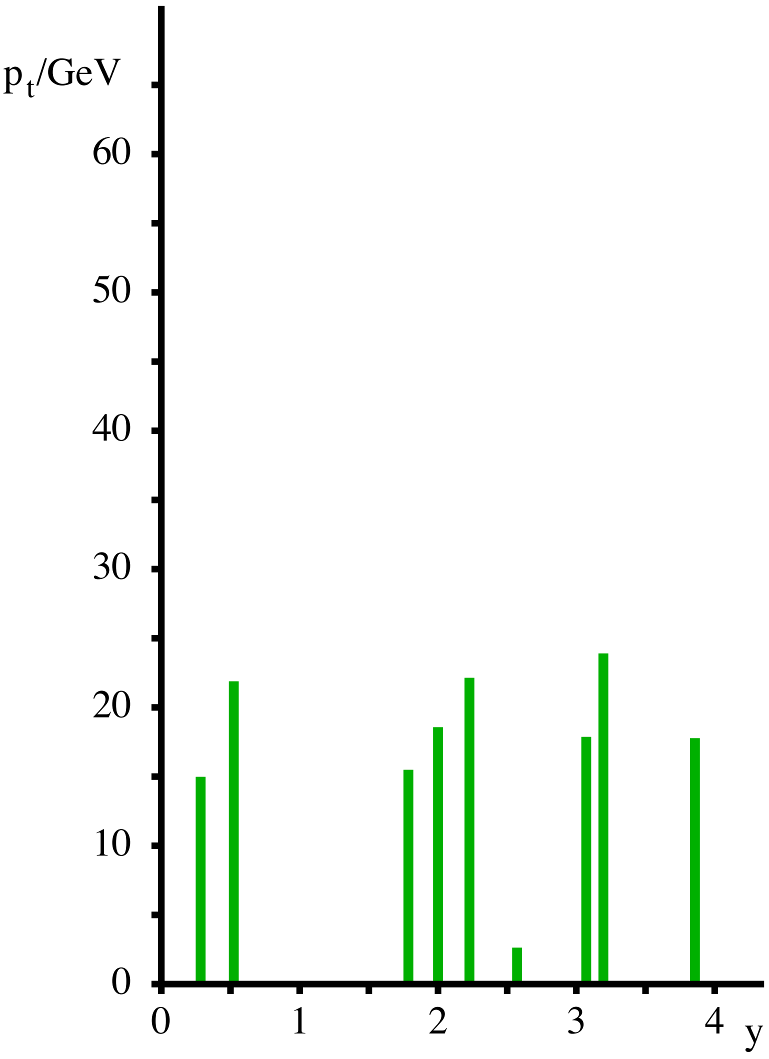
NB: their properties and behaviour can **vastly differ**: there isn't **'a'** cone algorithm, but rather many of them

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)

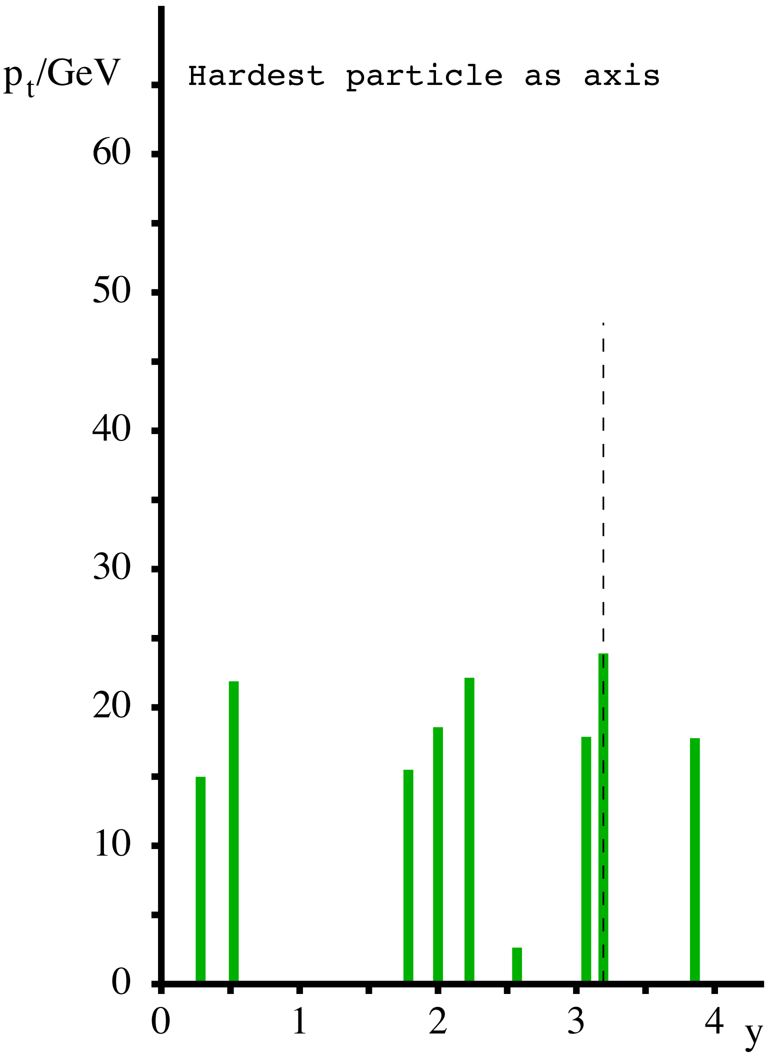
Fixed Cone, Progressive Removal (FC-PR)

Probably the simplest cone algorithm



Fixed Cone, Progressive Removal (FC-PR)

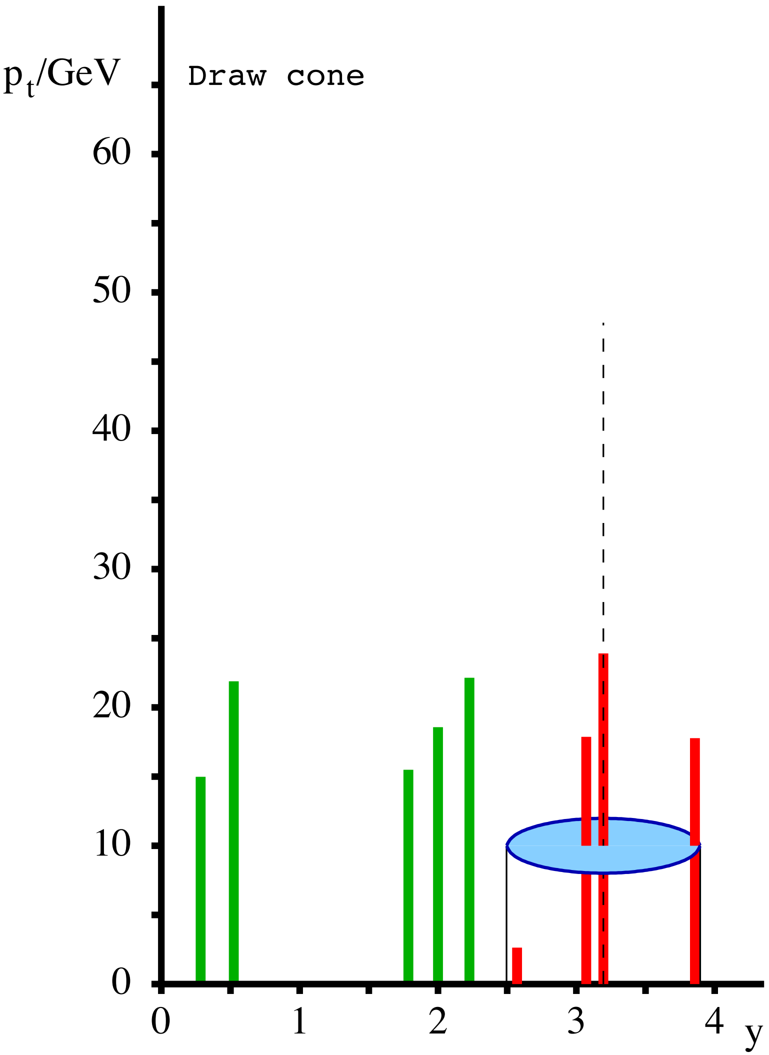
Probably the simplest cone algorithm



Choose hardest particle as seed

Fixed Cone, Progressive Removal (FC-PR)

Probably the simplest cone algorithm

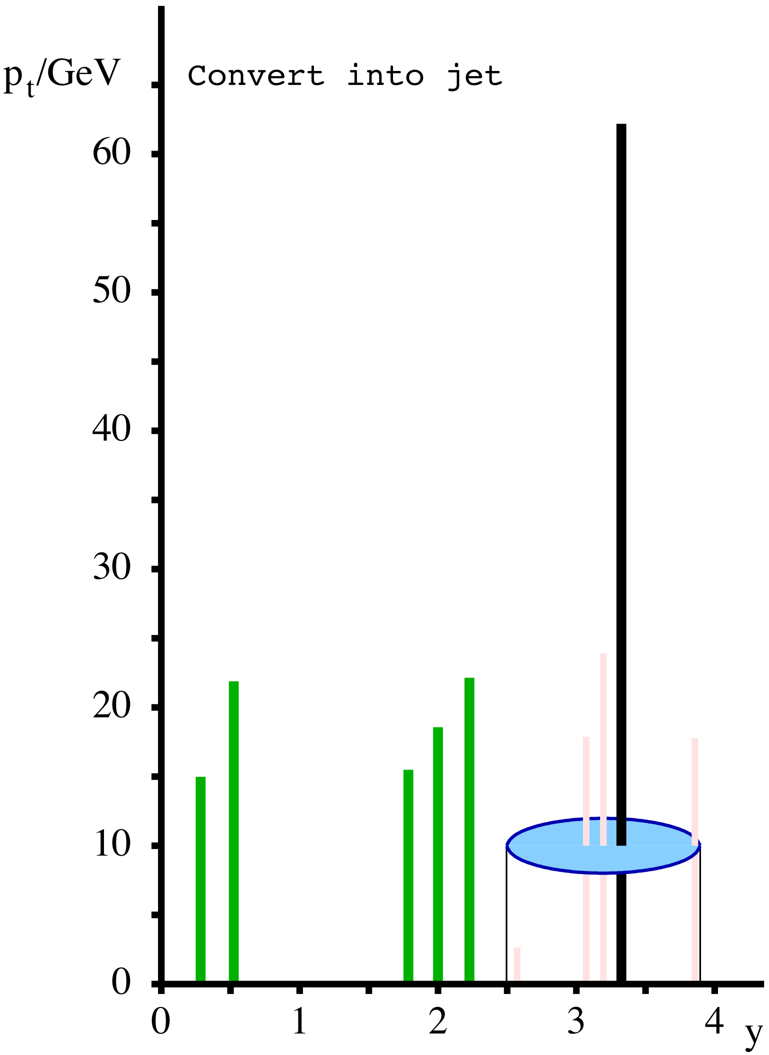


Choose hardest particle as seed

Draw cone around it

Fixed Cone, Progressive Removal (FC-PR)

Probably the simplest cone algorithm

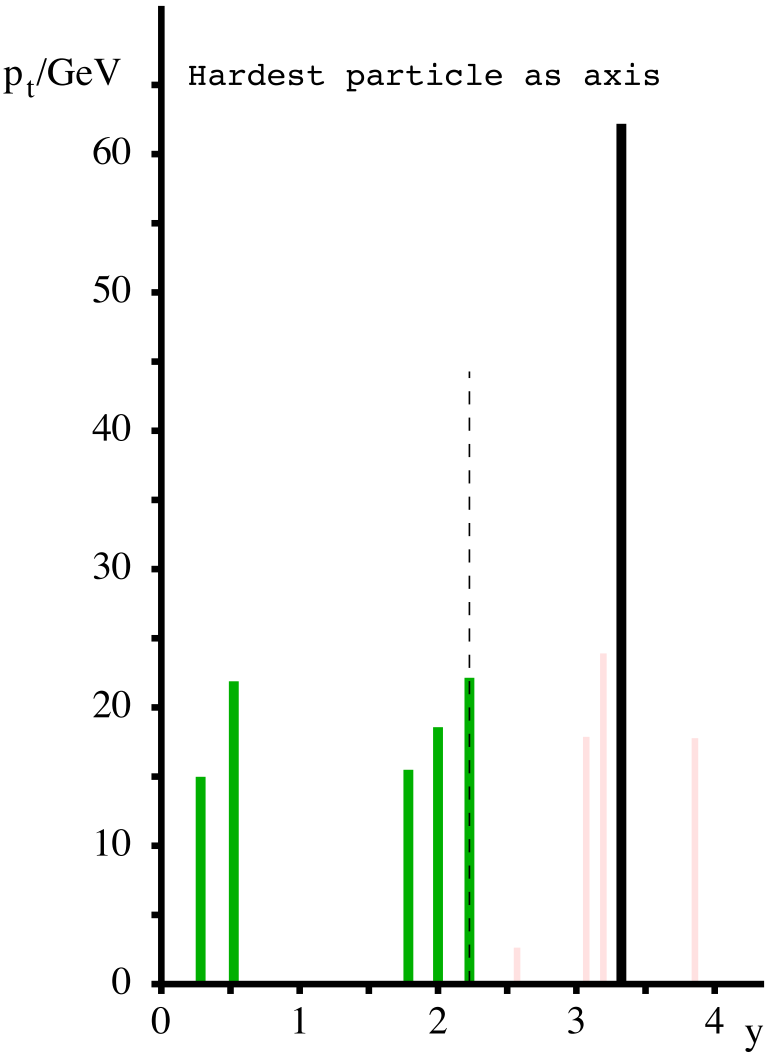


Choose hardest particle as seed

Draw cone around it

Call it a jet, remove constituents from set of particles

Fixed Cone, Progressive Removal (FC-PR)



Probably the simplest cone algorithm

Choose hardest particle as seed

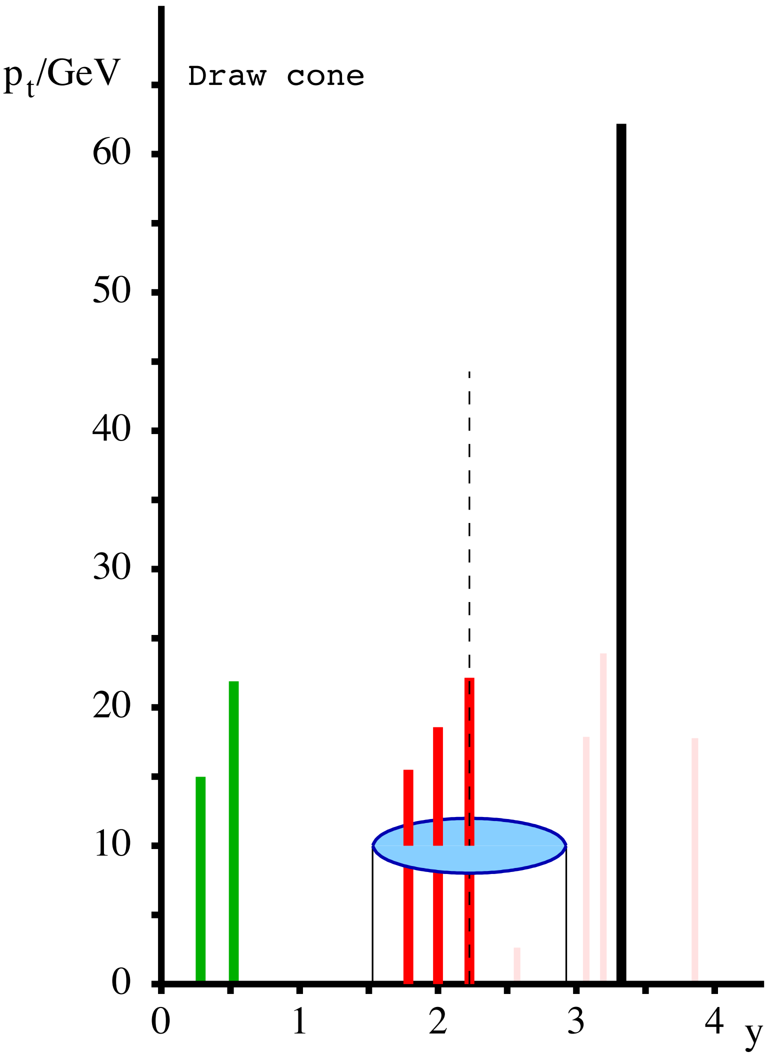
Draw cone around it

Call it a jet, remove constituents from set of particles

Repeat using hardest particle left

Fixed Cone, Progressive Removal (FC-PR)

Probably the simplest cone algorithm



Choose hardest particle as seed

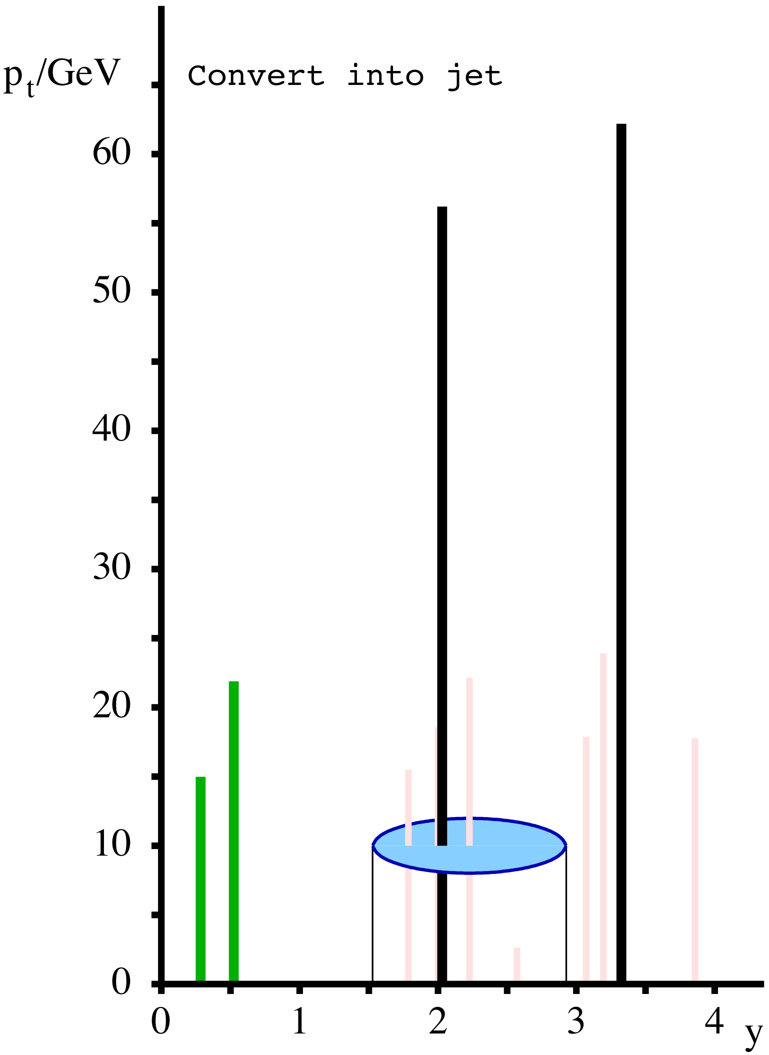
Draw cone around it

Call it a jet, remove constituents from set of particles

Repeat using hardest particle left

Etc, etc

Fixed Cone, Progressive Removal (FC-PR)



Probably the simplest cone algorithm

Choose hardest particle as seed

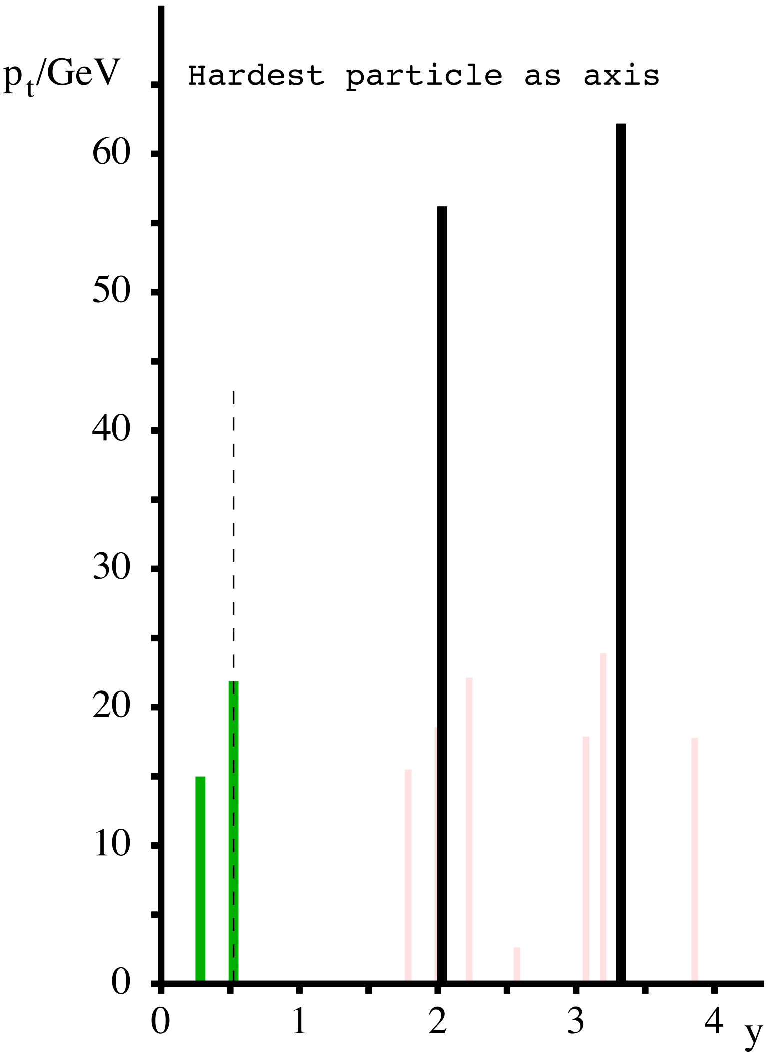
Draw cone around it

Call it a jet, remove constituents from set of particles

Repeat using hardest particle left

Etc, etc

Fixed Cone, Progressive Removal (FC-PR)



Probably the simplest cone algorithm

Choose hardest particle as seed

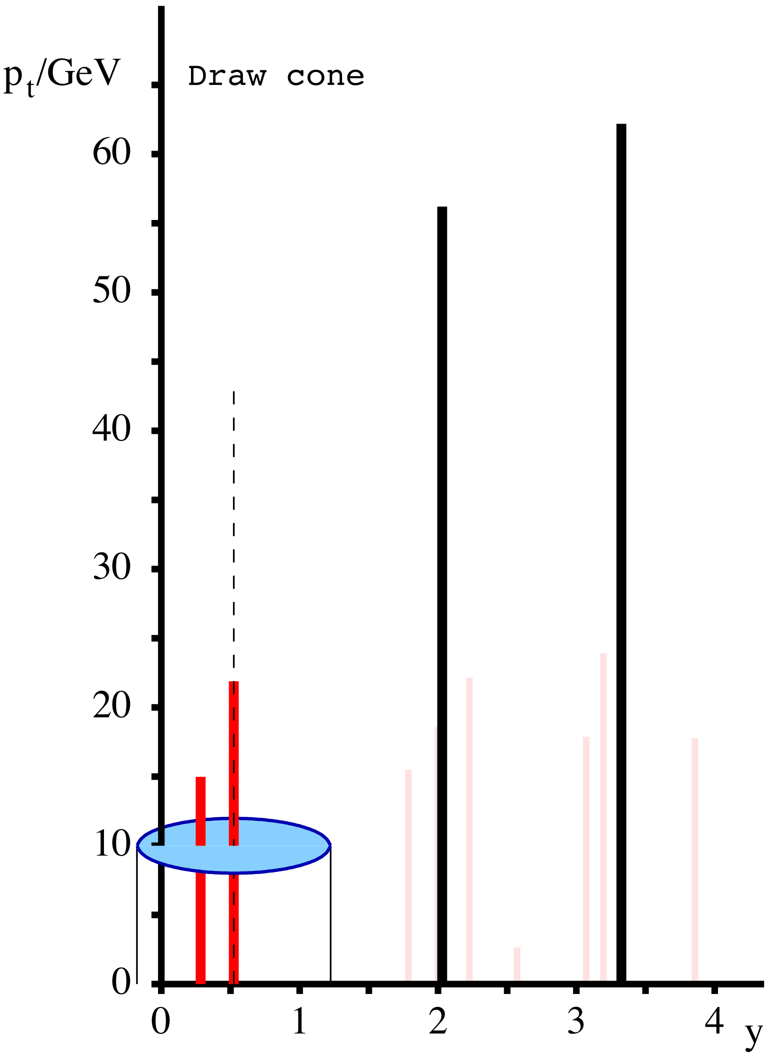
Draw cone around it

Call it a jet, remove constituents from set of particles

Repeat using hardest particle left

Etc, etc

Fixed Cone, Progressive Removal (FC-PR)



Probably the simplest cone algorithm

Choose hardest particle as seed

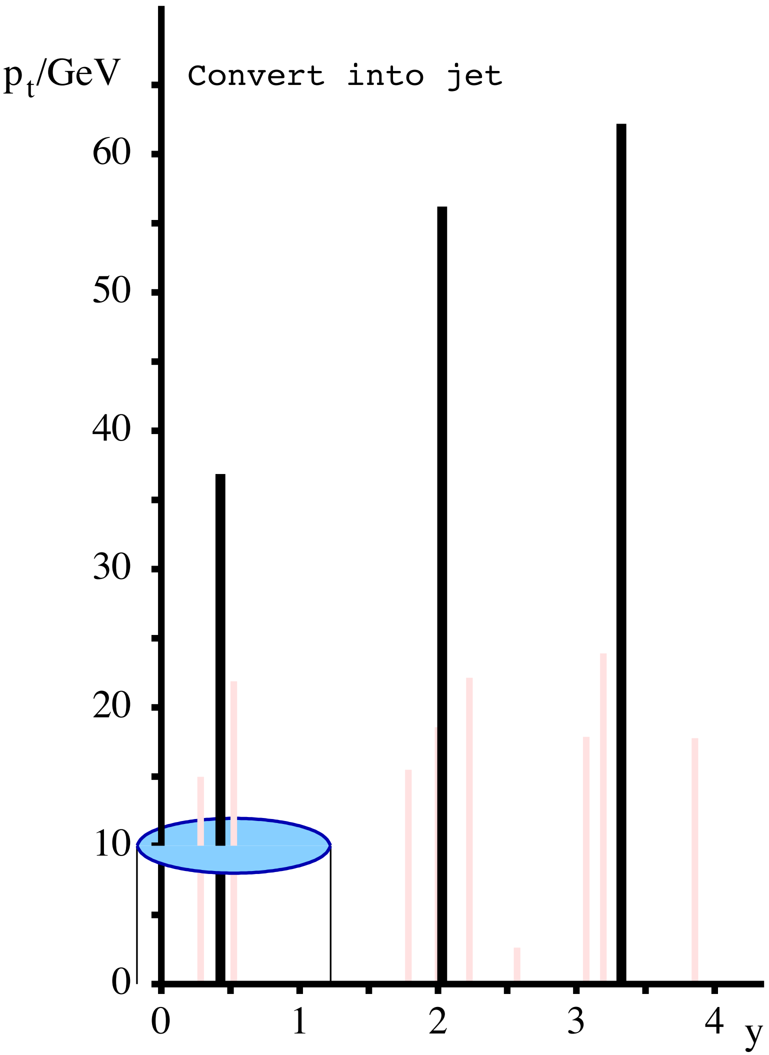
Draw cone around it

Call it a jet, remove constituents from set of particles

Repeat using hardest particle left

Etc, etc

Fixed Cone, Progressive Removal (FC-PR)



Probably the simplest cone algorithm

Choose hardest particle as seed

Draw cone around it

Call it a jet, remove constituents from set of particles

Repeat using hardest particle left

Etc, etc

Until no particles left

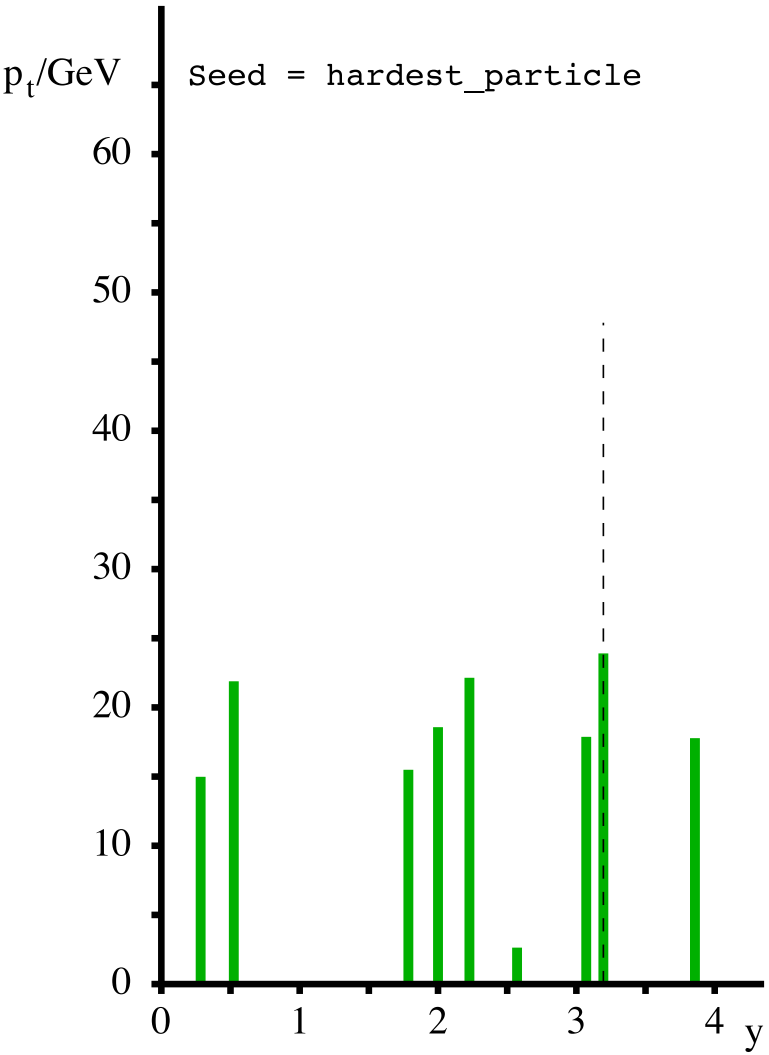
FC-PR v. IC-PR

Seed and cone axis may **not coincide**.
Making them do can lead to different jets

Let us try an **Iterative** Cone with Progressive Removal (IC-PR)
(e.g. the CMS Iterative Cone)

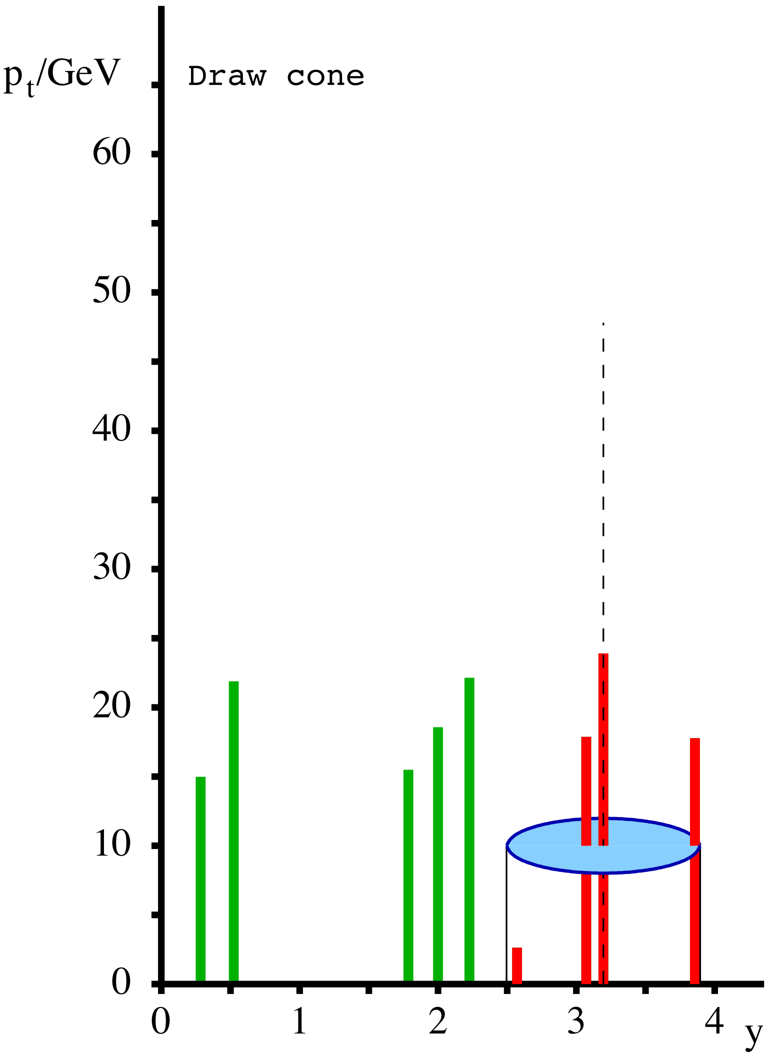
- Begin with **hardest particle** as seed
- Cluster particles into cone if $\Delta R < R$
- **Iterate** until stable (i.e. axis coincide with sum of momenta) cones found
- Eliminate constituents of jet and start over from hardest remaining particle

FC-PR v. IC-PR



Choose hardest particle as seed

FC-PR v. IC-PR

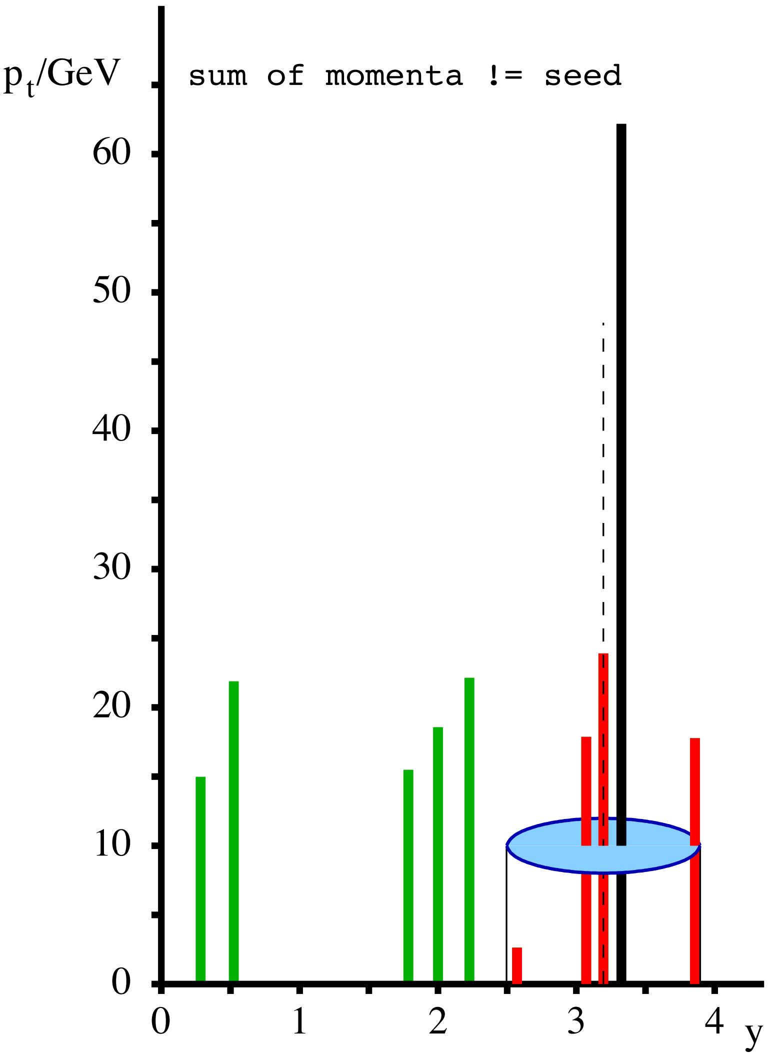


Choose hardest particle as seed

Draw cone around it

So far, identical to FC-PR

FC-PR v. IC-PR

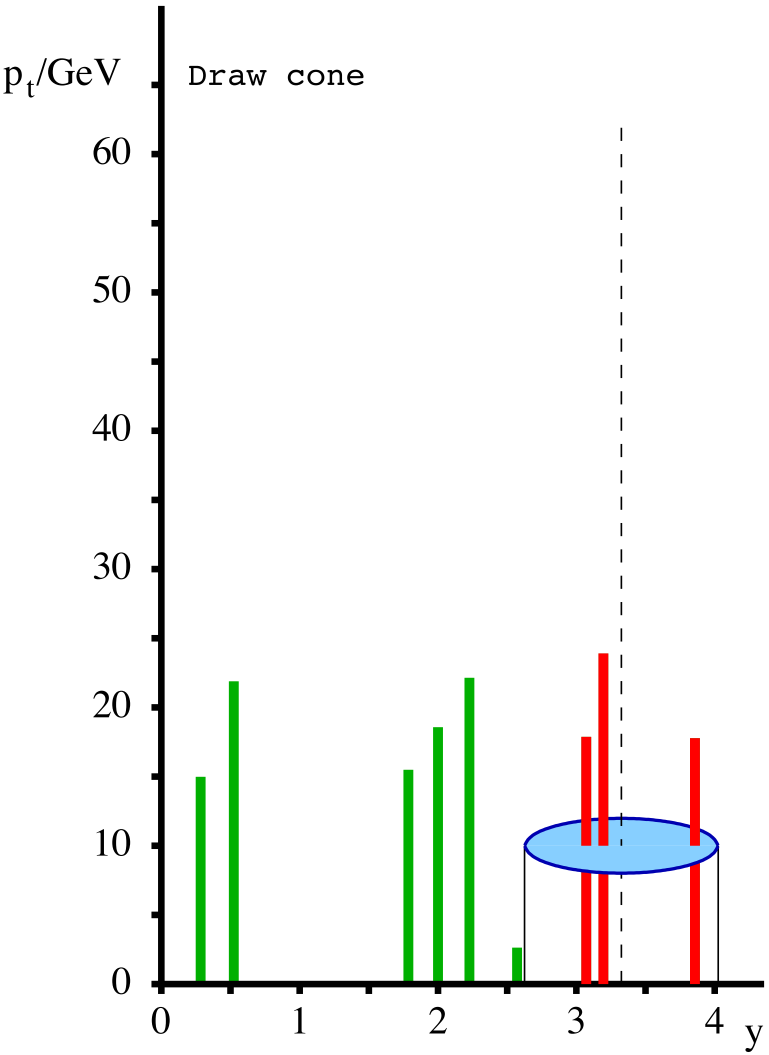


Choose hardest particle as seed

Draw cone around it

Jet axis not centred on seed

FC-PR v. IC-PR



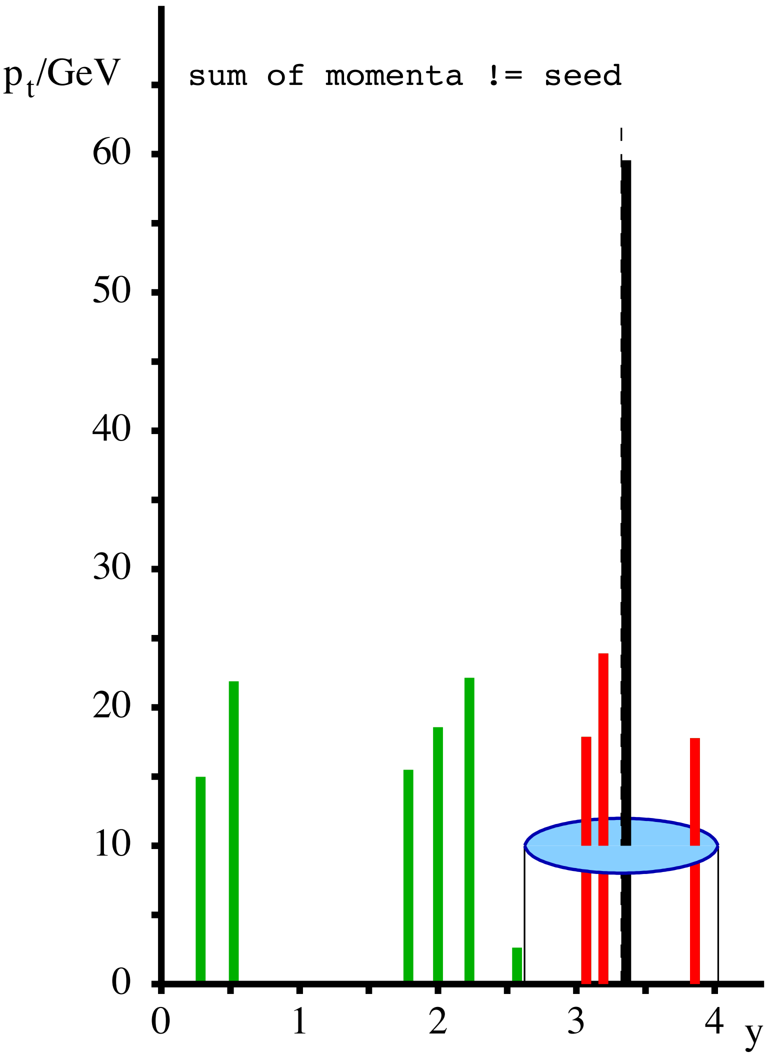
Choose hardest particle as seed

Draw cone around it

Jet axis not centred on seed

Redraw cone around new axis

FC-PR v. IC-PR



Choose hardest particle as seed

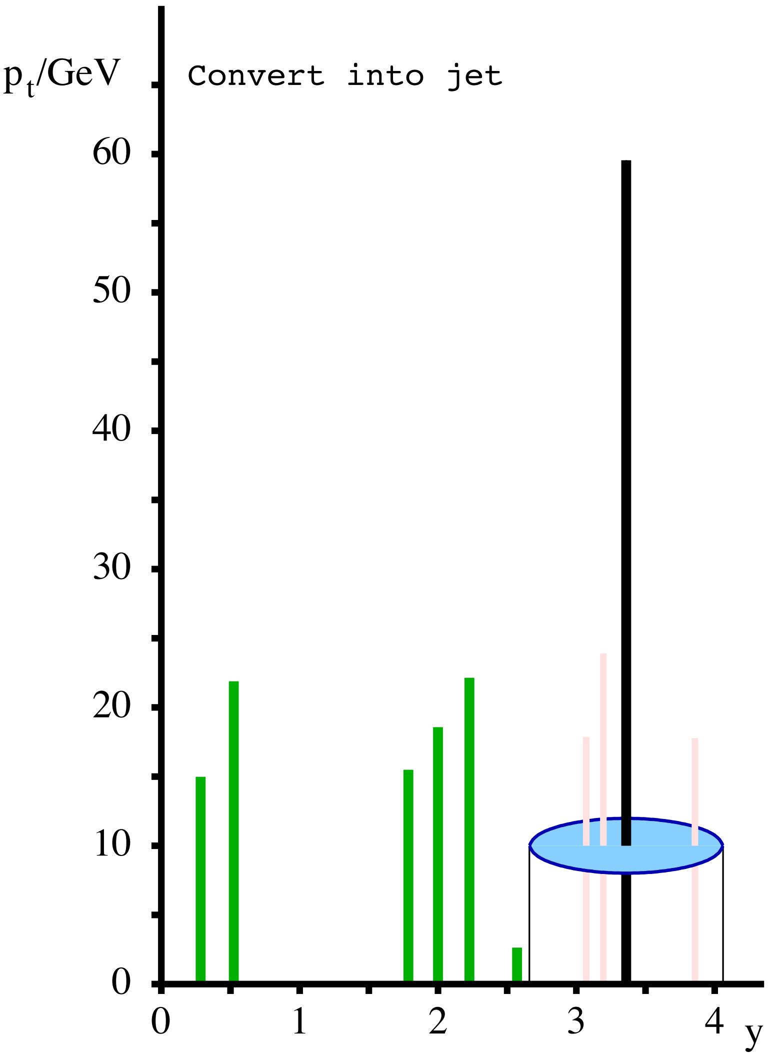
Draw cone around it

Jet axis not centred on seed

Redraw cone around new axis

Still, jet axis not centred on seed

FC-PR v. IC-PR



Choose hardest particle as seed

Draw cone around it

Jet axis not centred on seed

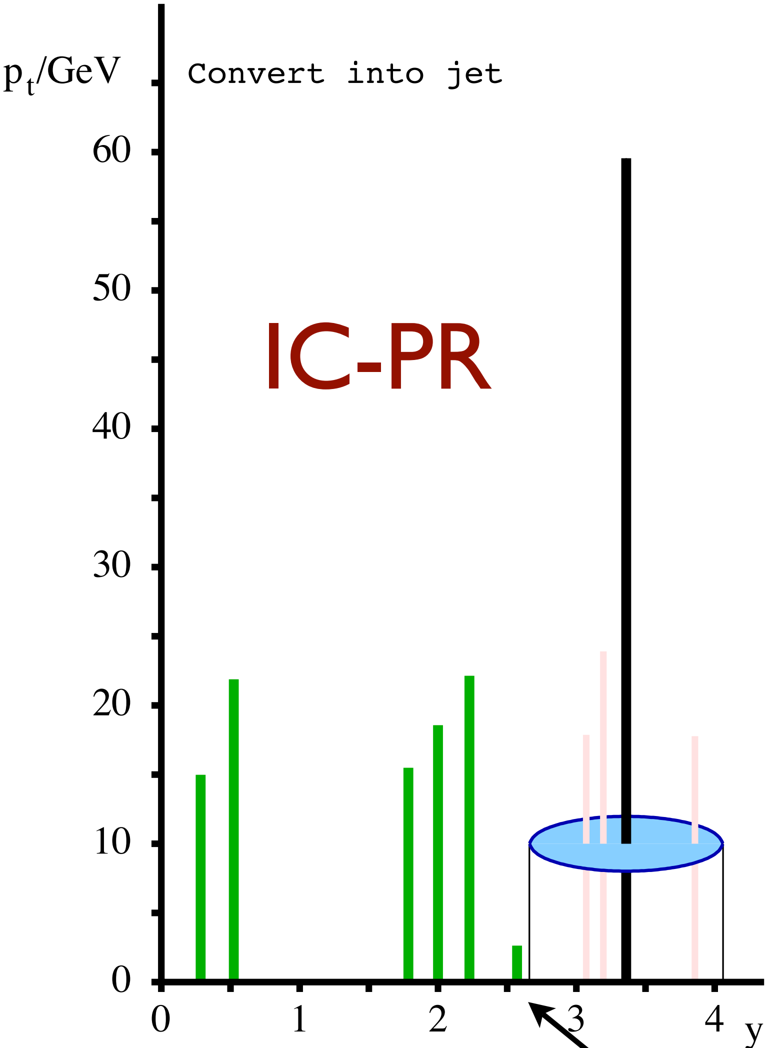
Redraw cone around new axis

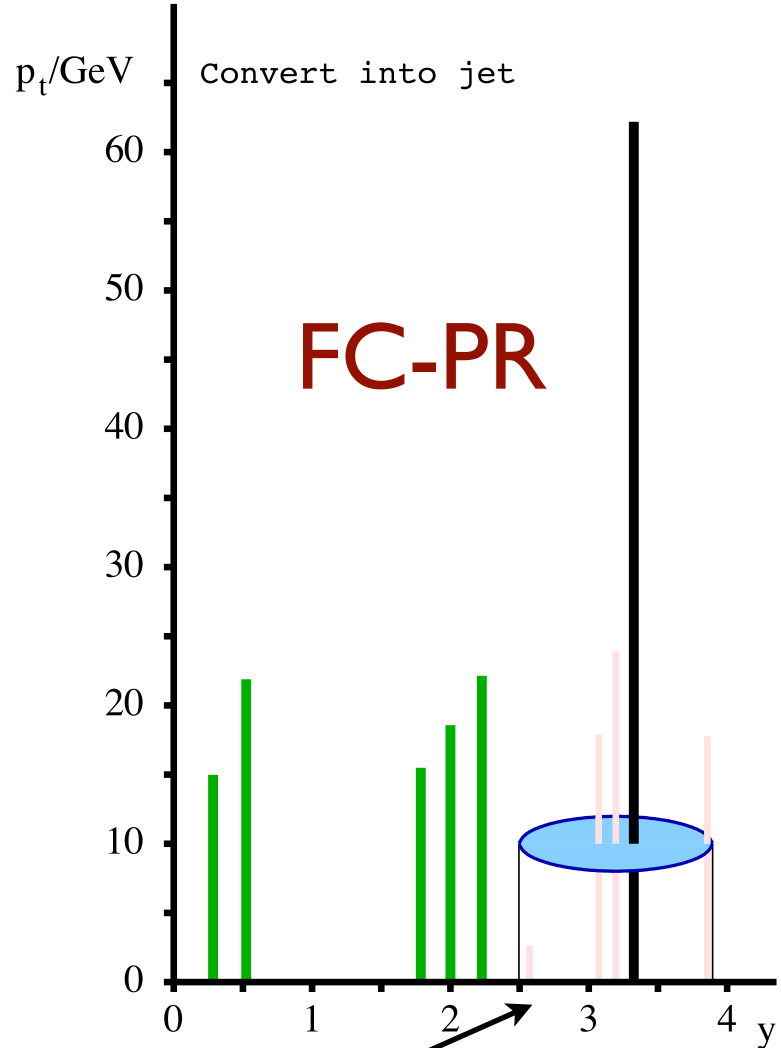
Still, jet axis not centred on seed

Repeat until it is, finally get to this

FC-PR v. IC-PR

This jet differs from the corresponding FC-PR one

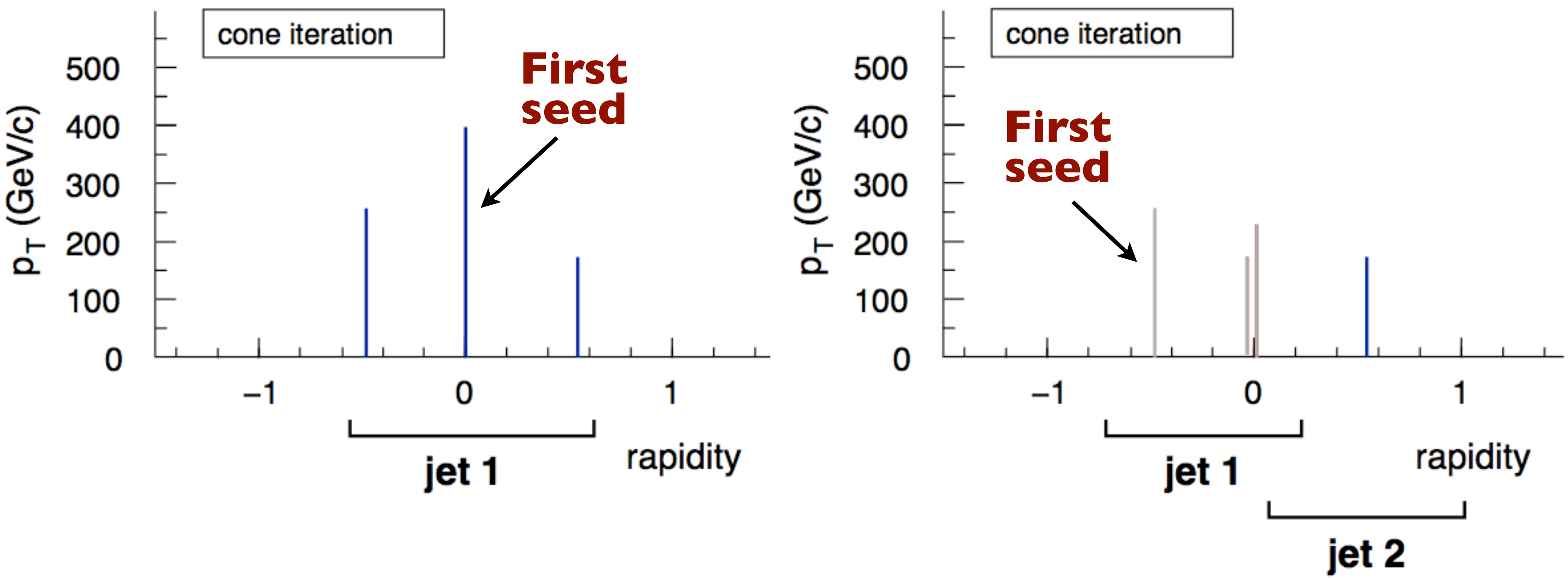




Different constituents

IC-PR cone collinear unsafety

A collinear splitting can change the final state

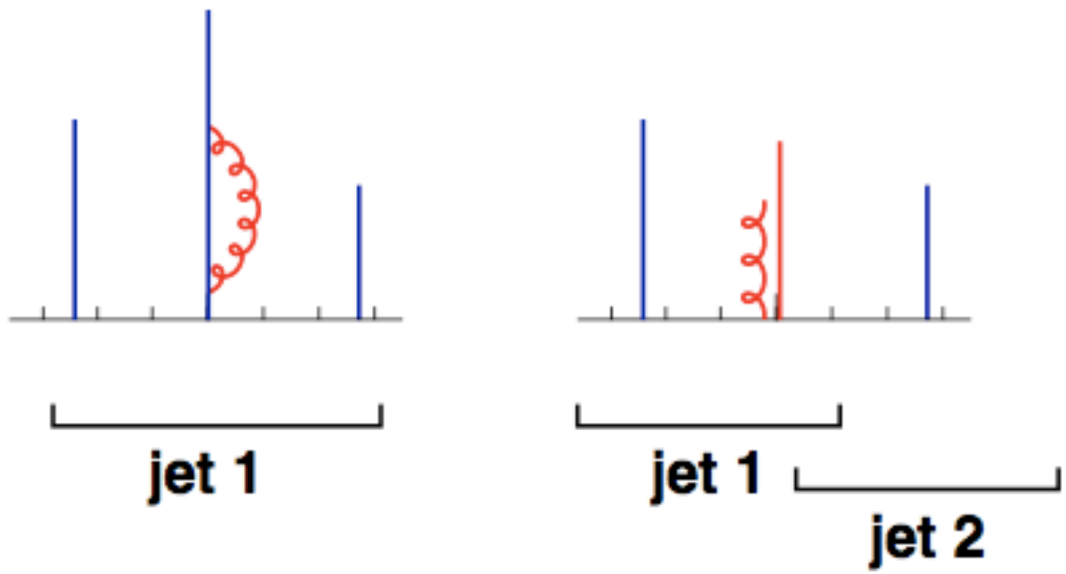


Splitting the hardest particle **collinearly** has changed the number of final jets

Consequences of collinear unsafety

In QCD perturbation theory, virtual and soft/collinear real configurations can only cancel if they lead to the **same** final state

In this example with IC-PR, we have seen that the final state can differ:



⇒ no cancellation of divergencies, no convergence of perturbation theory

Jet algorithms using hardest particles as seeds will generally be susceptible to collinear unsafety

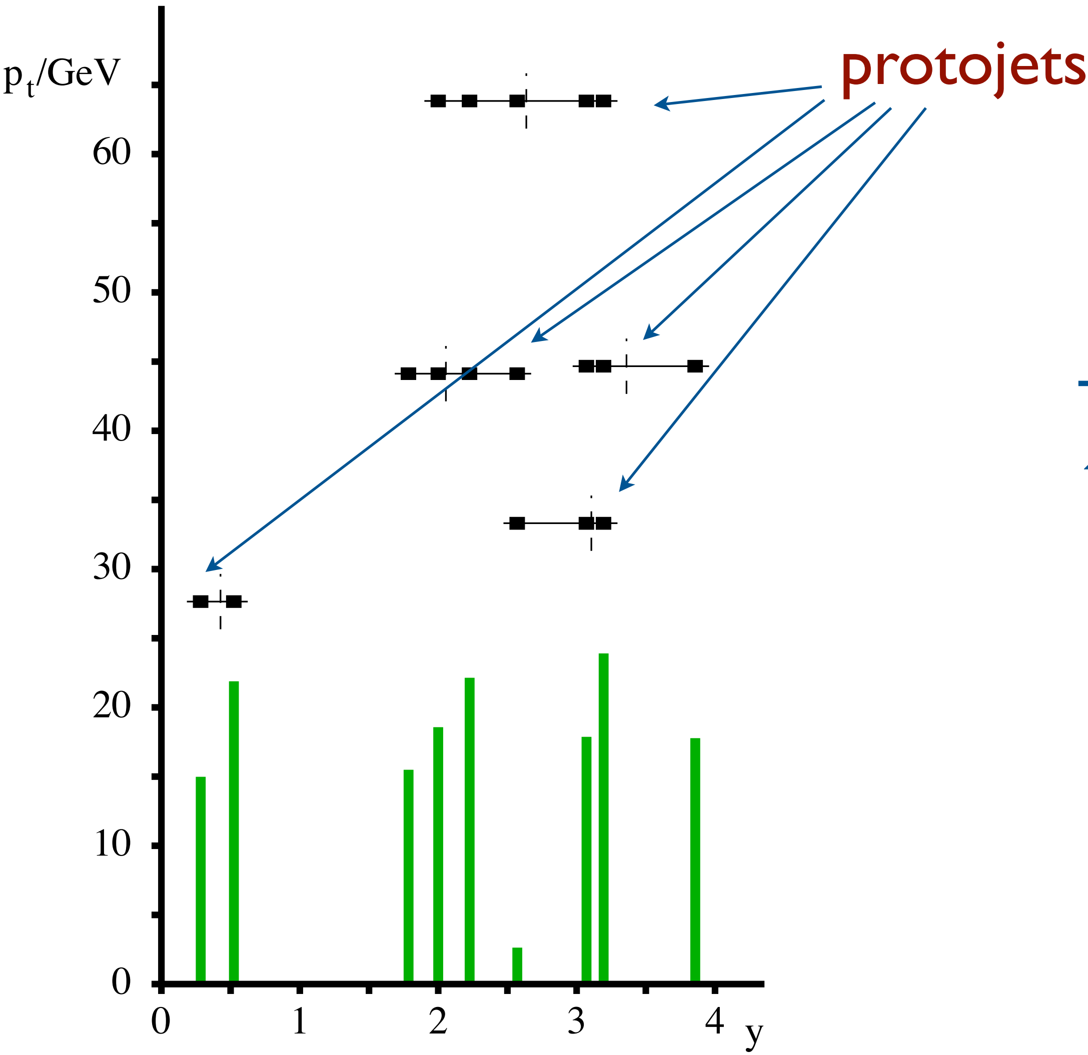
Iterative Cone with Split-Merge (IC-SM)

Choosing hardest particles as seed was an issue (collinear unsafety).
Let us therefore try taking **all particles**

- 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 later on)

Examples of this algorithm are JetClu and the ATLAS Cone

Iterating the cones over all particles as seeds returns 5 stable protojets



The lack of 'progressive removal' means that some protojets can be overlapping (i.e. contain the same particles). Must deal with this: **split-merge**

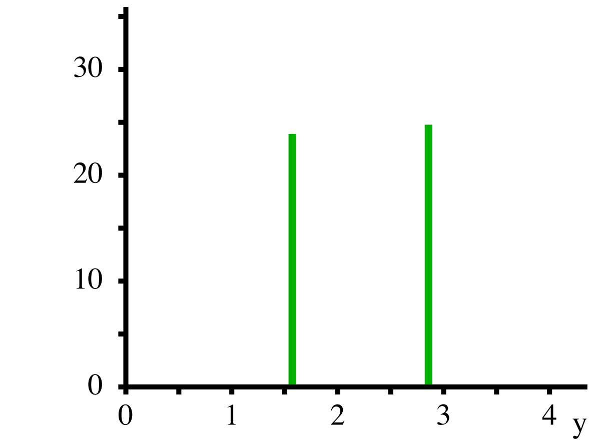
Split-Merge

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

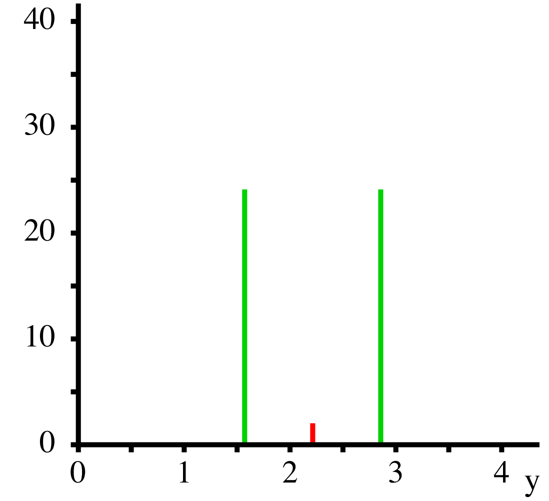
The Tevatron Run II implementation goes like this:

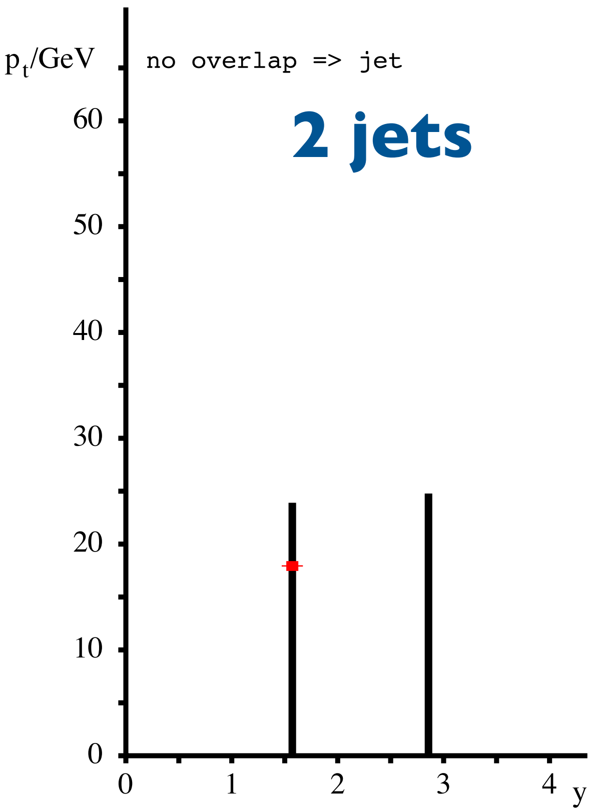
- Choose an **overlap threshold *f***
- Find hardest protojet
- Find hardest other protojet overlapping with it
- Merge is they share a fraction of momentum larger than *f*, split along axis at centre otherwise
- (Call protojet a jet if there are no overlapping protojets)

IC-SM infrared unsafety

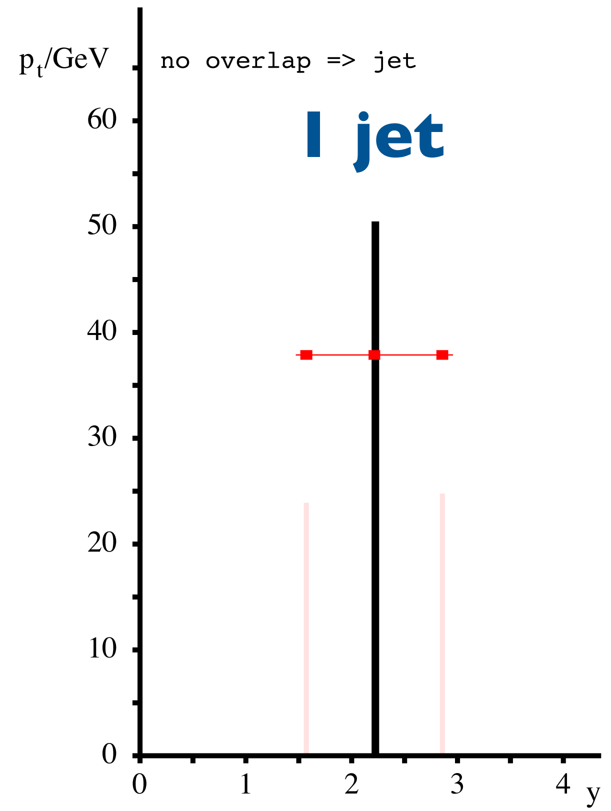


Add a
soft particle





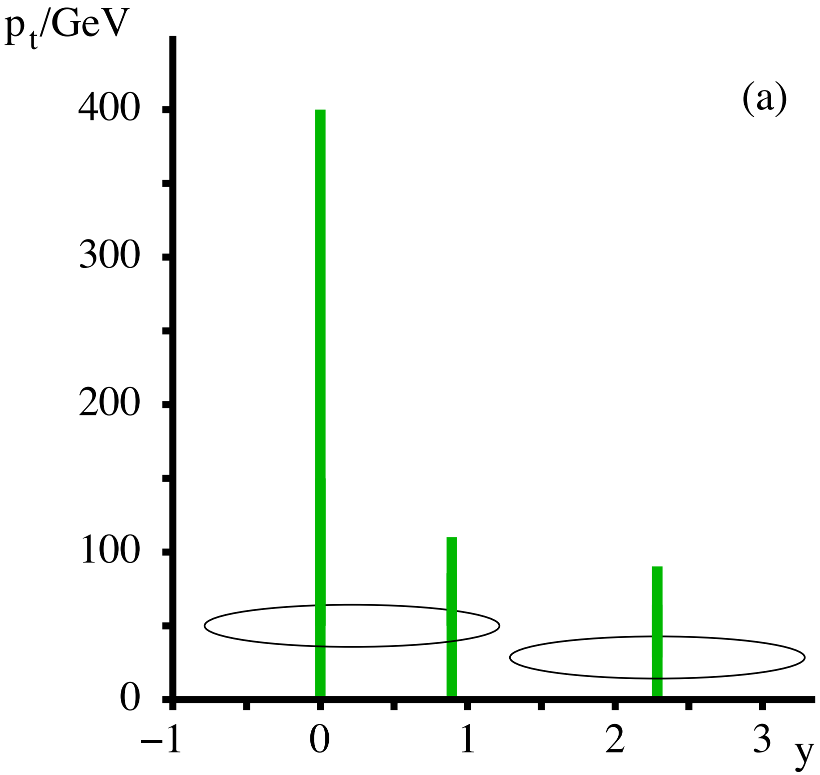
Final state jets differ



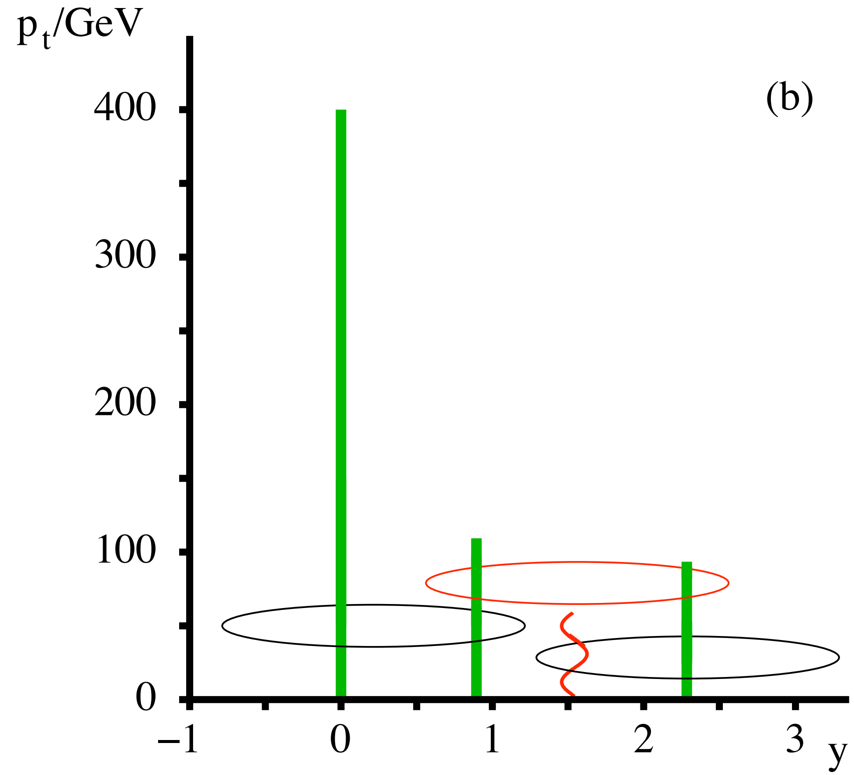
MidPoint (IC_{mp}-SM) infrared unsafety

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

A long list of cones (all eventually unsafe)

Les Houches 2007 proceedings, arXiv:0803.0678

'First-generation' algorithms

CDF JetClu	IC_r-SM	IR_{2+1}
CDF MidPoint cone	IC_{mp}-SM	IR_{3+1}
CDF MidPoint searchcone	$IC_{se,mp}$-SM	IR_{2+1}
D0 Run II cone	IC_{mp}-SM	IR_{3+1}
ATLAS Cone	IC-SM	IR_{2+1}
PxCone	IC_{mp}-SD	IR_{3+1}
CMS Iterative Cone	IC-PR	$Coll_{3+1}$
PyCell/CellJet (from Pythia)	FC-PR	$Coll_{3+1}$
GetJet (from ISAJET)	FC-PR	$Coll_{3+1}$

IC = Iterative Cone
SM = Split-Merge
SD = Split-Drop
FC = Fixed Cone
PR = Progressive Removal

type of algorithm

safety issue

IR_{n+1} : unsafe when a soft particle is added to n hard particles in a common neighbourhood

$Coll_{n+1}$: unsafe when one of n hard particles in a common neighbourhood is split collinearly

IRC safety does matter

The best cones seen so far fail at (3+1) partons, others already at (2+1)

	Last meaningful order			
	JetClu, ATLAS cone [IC-SM]	MidPoint [IC$_{mp}$-SM]	CMS it. cone [IC-PR]	Known at
Inclusive jets	LO	NLO	NLO	NLO ($\rightarrow$ NNLO)
$W/Z + 1$ jet	LO	NLO	NLO	NLO
3 jets	none	LO	LO	NLO [nlojet++]
$W/Z + 2$ jets	none	LO	LO	NLO [MCFM]
m_{jet} in $2j + X$	none	none	none	LO

Calculations cost real money: ~ 100 theorists ×15 years ≈100 M€

Using unsafe jet tools essentially renders them useless

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 thory/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)

Seedless IRC-safe Cone (SC-SM): SISCone

Salam, Soyez, arXiv:0704:0292

Seeds are a problem:
they lead to finding only some of the stable cones

Obvious solution:
find ALL stable cones, testing all possible combinations of N particles

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

Way out: a geometrical solution → SISCone

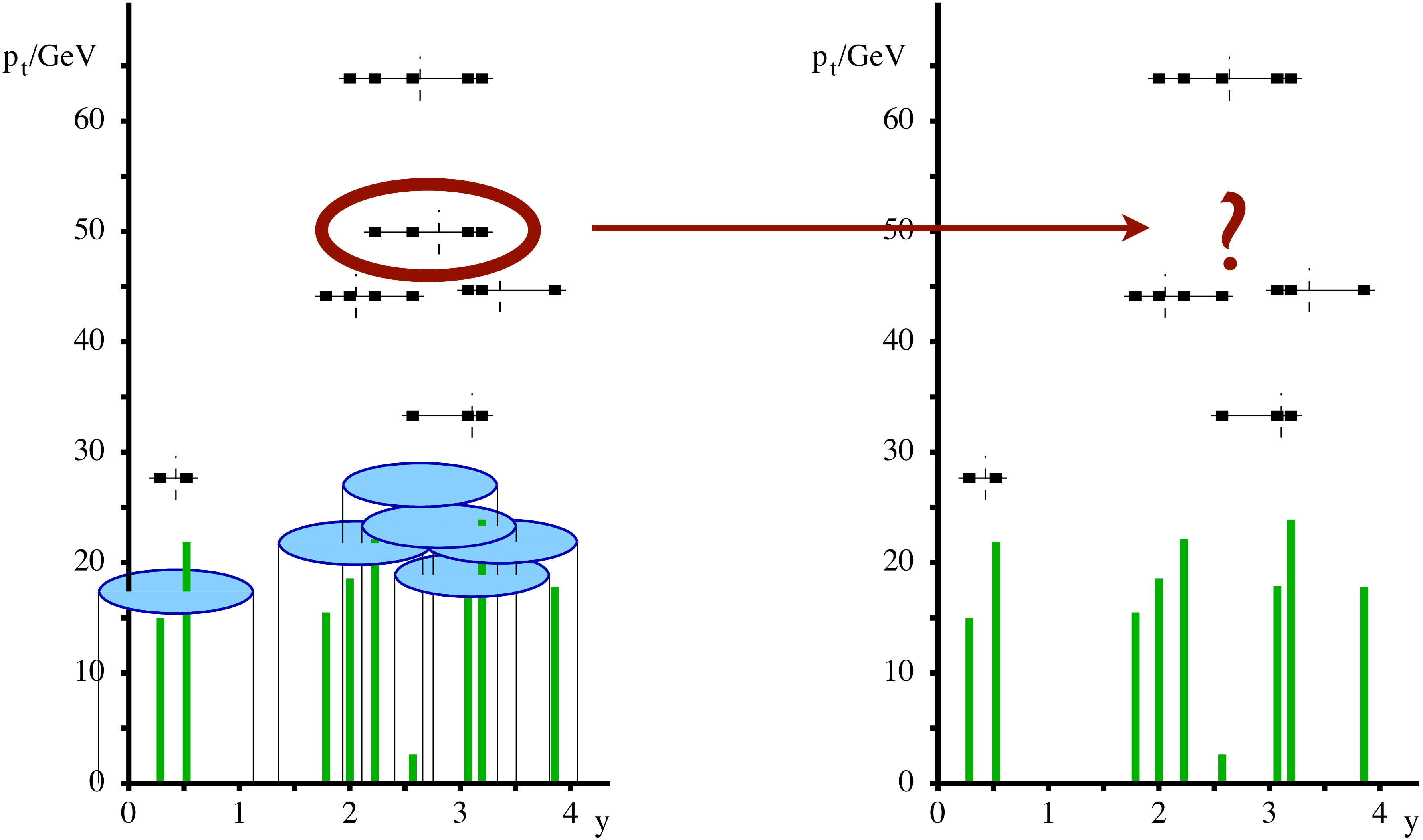
The first (and only?) IRC-safe cone algorithm for hadronic collisions

SISCone is guaranteed to find ALL the stable cones

SISCone v. IC-SM

These are ALL the stable cones

Compare to those found by IC-SM: one is missing



SISCone

IC-SM

Cones Infrared (un)safety

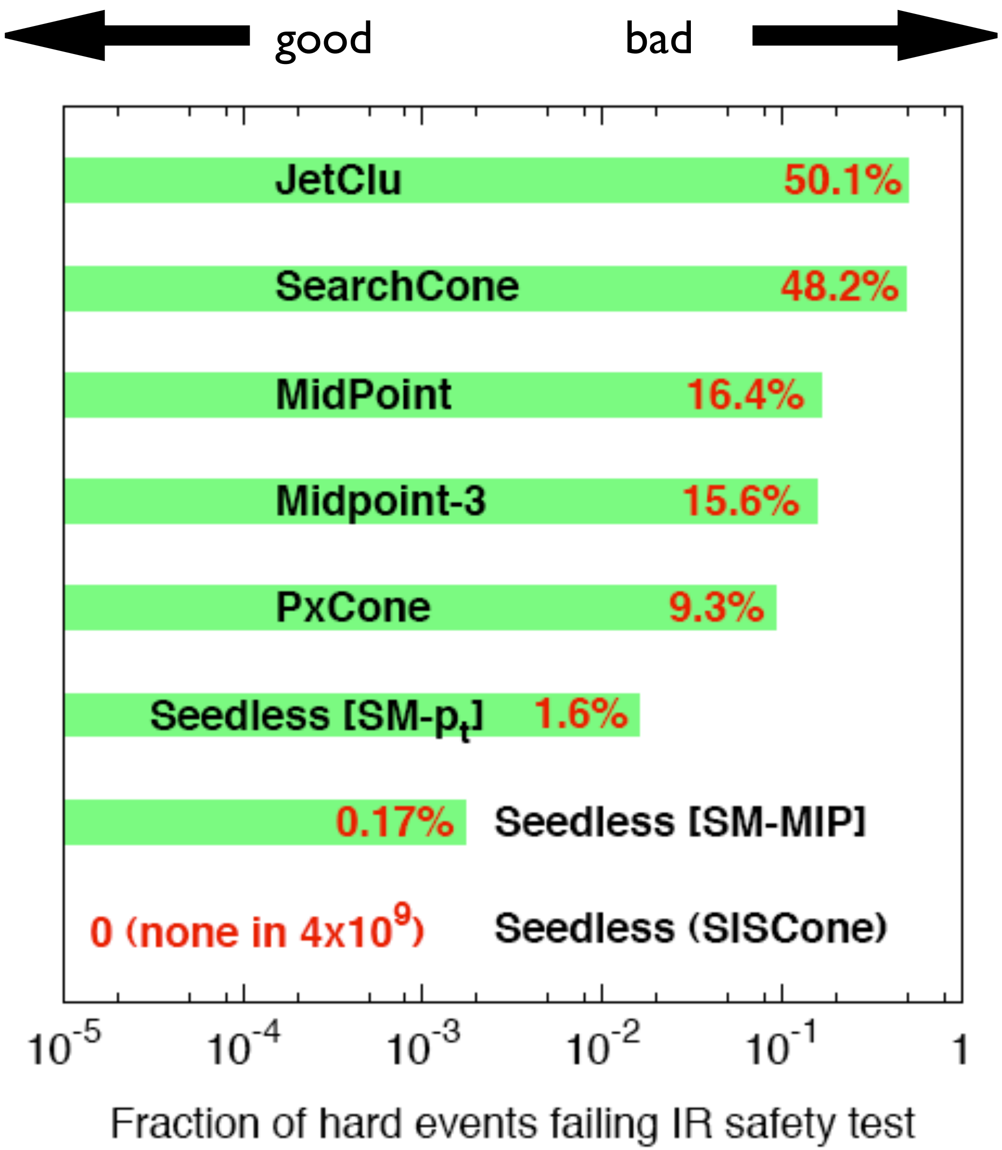
Q: How often are the hard jets changed by the addition of a soft particle?

A:

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



Salam & Soyez

Recombination algorithms

(Inclusive version)

- Calculate the distances between the particles: $\mathbf{d_{ij}}$
- Calculate the beam distances: $\mathbf{d_{iB}}$
- Combine particles with **smallest distance** or, if d_{iB} is smallest, call it a jet
- Find again smallest distance and repeat procedure until no particles are left

IRC safety can usually be seen to be trivially guaranteed

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

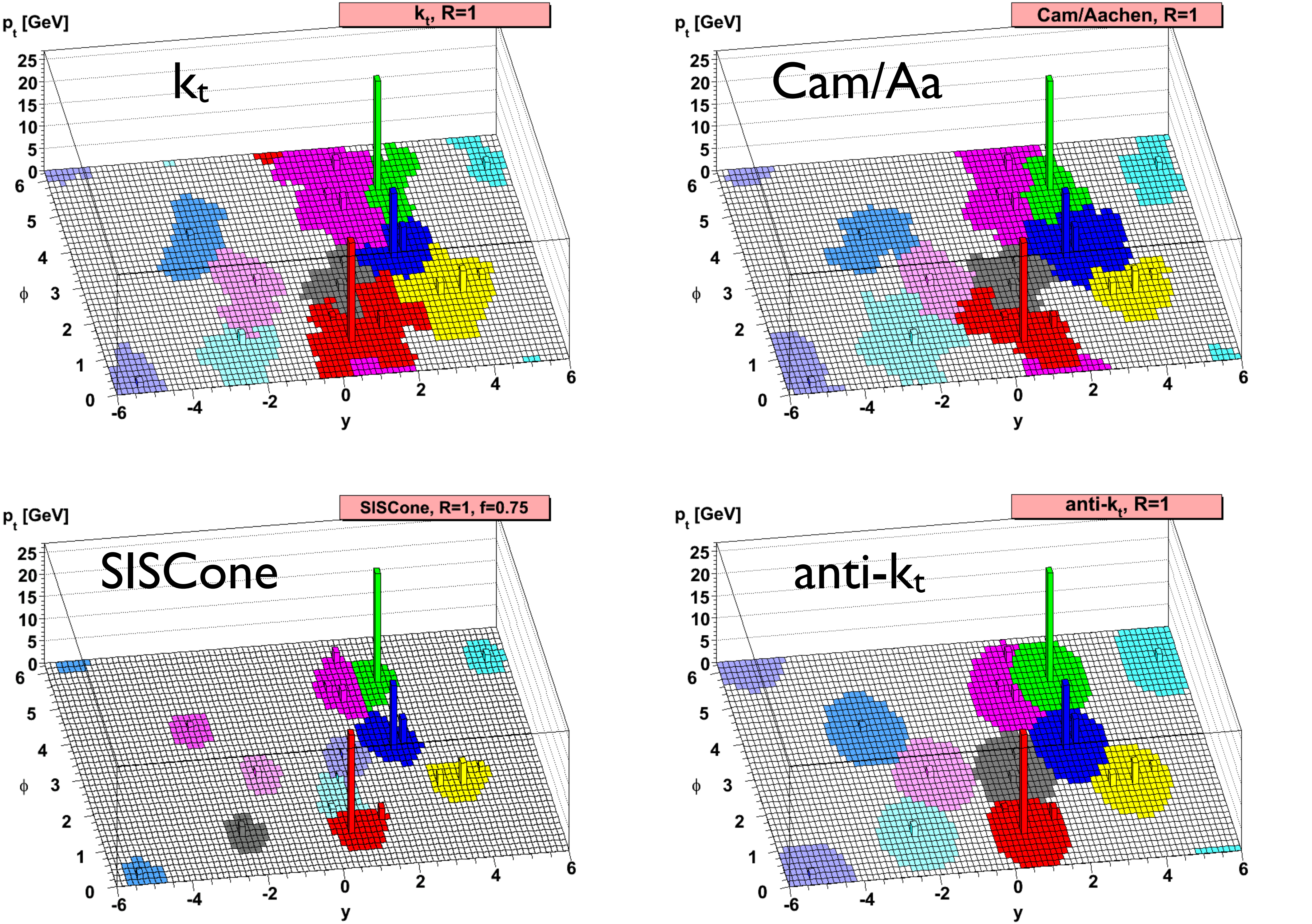
The IRC safe algorithms

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	$N^2\ln N$

We call these algs '**second-generation**' ones

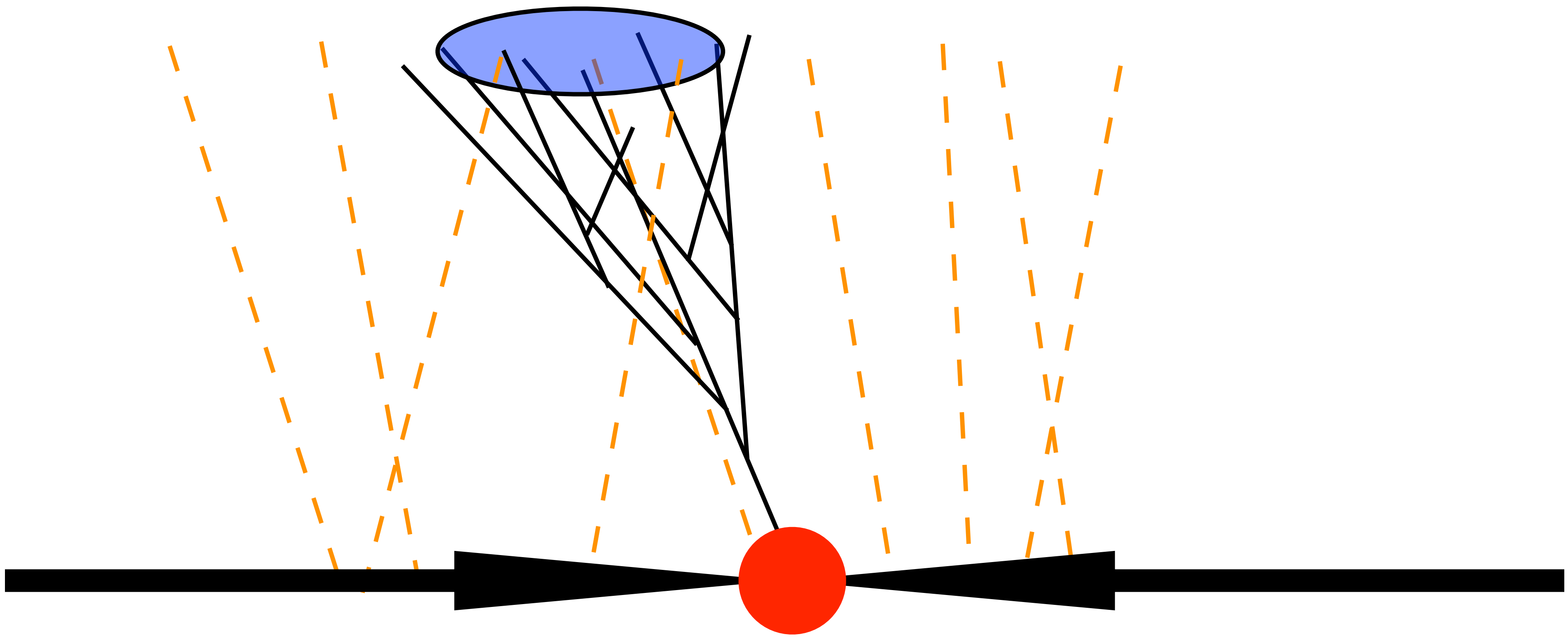
All are available in FastJet, http://fastjet.fr

(As well as many IRC unsafe ones)



kt, R=1
kt
Cam/Aachen, R=1
Cam/Aa
SISCone, R=1, f=0.75
SISCone
anti-kt, R=1
anti-kt
pt [GeV]
φ
y

Hard jets and background



In a realistic set-up underlying event (UE) and pile-up (PU) from multiple collisions produce many soft particles which can 'contaminate' the hard jet

Hard jets and background

How are the hard jets modified by the background?

Susceptibility (how much bkgd gets picked up)

Resiliency (how much the original jet changes)

Susceptibility: jet area

MC, Salam, Soyez, arXiv:0802.1188

Operational definition of **active jet area:**

Add many **ghost-particles** of infinitesimally small momentum to the hard event.
Cluster them together with the real particles,
and count how many on average get clustered within a given jet.

$$A(J \mid \{g_i\}) = \frac{N_g(J)}{\nu_g}$$

Number of ghosts in jet J

Active area of a single ghosts configuration

Ghost density

$$A(J) = \lim_{\nu_g \to \infty} \langle A(J \mid \{g_i\}) \rangle_g$$

Active area

Susceptibility: jet area

The definition of **active area** mimics the behaviour of the jet-clustering algorithms in the presence of a **large number of randomly distributed soft particles,** like those due to **pileup or underlying event**

Tools needed to implement it:

1. An **infrared safe jet algorithm** (the ghosts should not change the jets)
2. A reasonably **fast implementation** (we are adding thousands of ghosts)

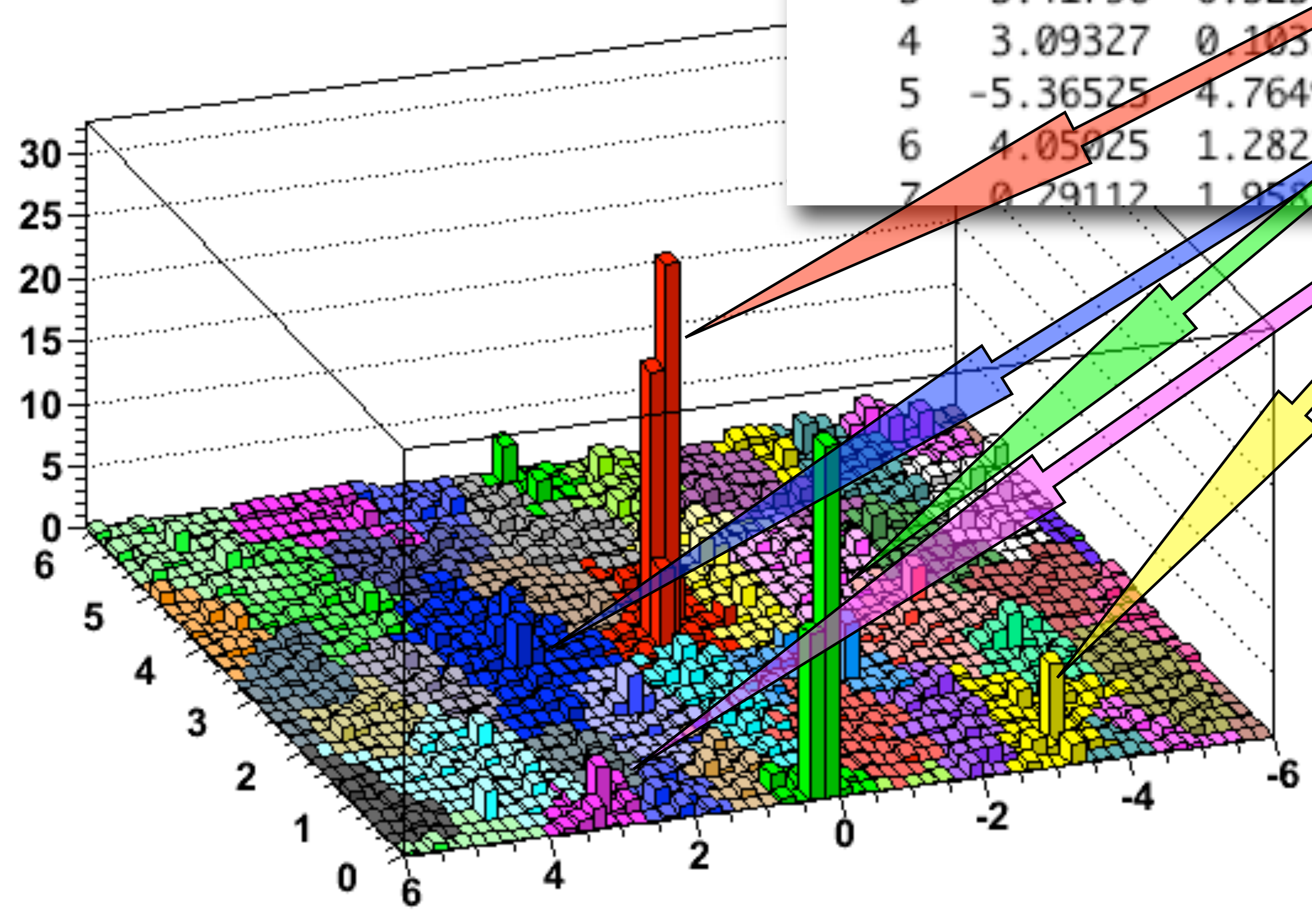
Both are available

In FastJet

```
/// constructor for an area definition based on an area type and a
/// ghosted area specification
AreaDefinition(AreaType type, const GhostedAreaSpec & spec);
```

Jet active areas

```
iev 0 (irepeat 24): number of particles = 1428
strategy used =  NlnN
number of particles = 9051
Total area: 76.0265
Expected area: 76.0265
 ijet     eta        phi         Pt          area  +-   err
    0   0.15050  3.24498     69.970     2.625 +-  0.020
    1   0.18579  0.13150     59.133     1.896 +-  0.020
    2   2.33840  3.23960     31.976     4.749 +-  0.028
    3  -3.41796  0.52394     26.595     3.084 +-  0.021
    4   3.09327  0.10350     20.072     2.688 +-  0.023
    5  -5.36525  4.76491     19.598     2.780 +-  0.012
    6   4.05025  1.28279     15.361     3.592 +-  0.028
    7   0.29112  1.95835     14.566     2.114 +-  0.018
```



The ghost can also give you a visual impression of the reach of each jet

A jet is not (always) a cone

The typical area of a jet around a hard particle is **not** necessarily πR^2

	k_t	Cam/Aa	SISCone	anti-k_t
$\langle A\rangle/\pi R^2$	0.81	0.81	1/4	1

Only anti-k_t has the behaviour one would naively expect

A jet is not (always) a cone

Also, the area can change with the p_t:

$$\langle \Delta A \rangle = \mathrm{D} \, \frac{C_1}{\pi b_0} \ln \frac{\alpha_s(Q_0)}{\alpha_s(R p_{t1})}$$

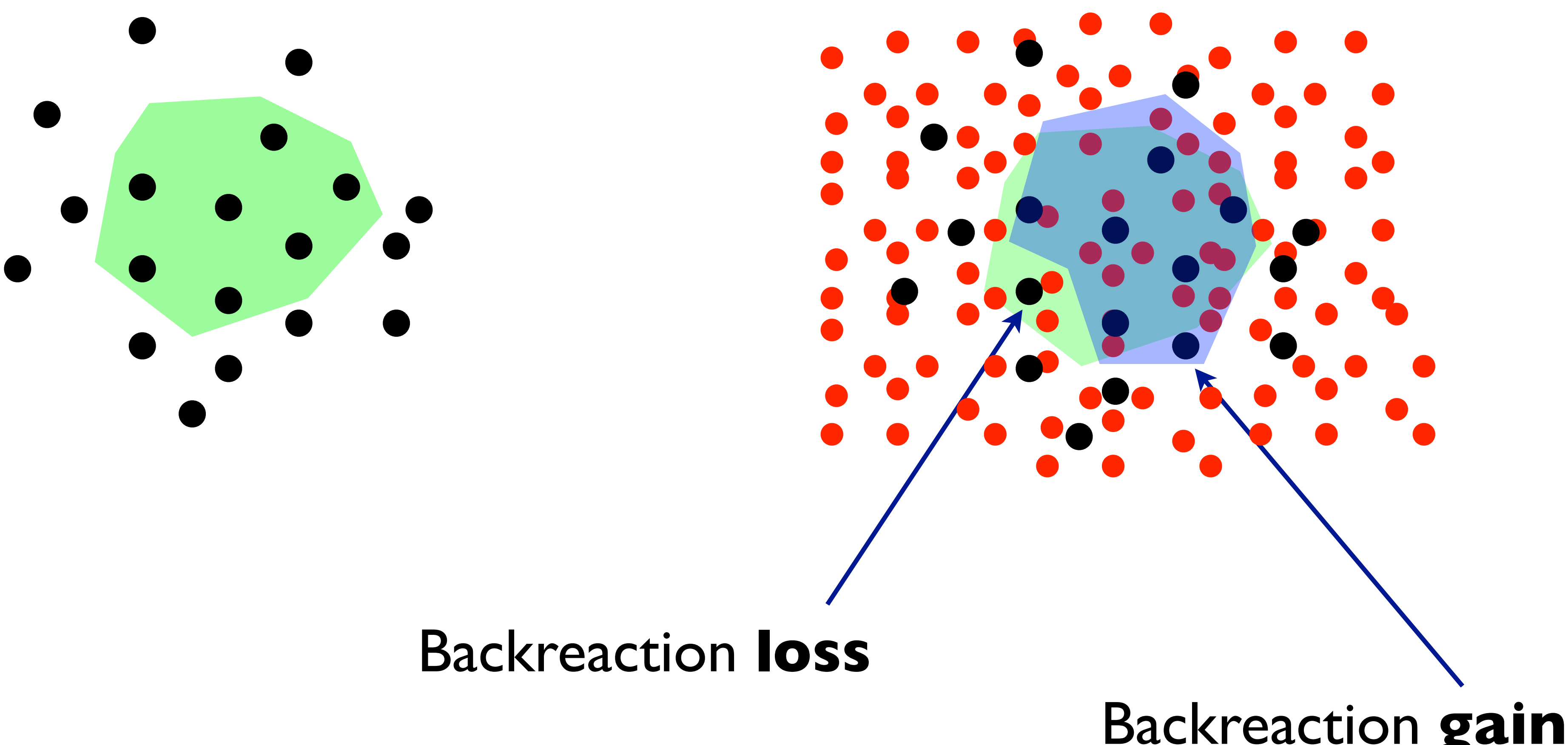
	k_t	Cam/Aa	SISCone	anti-k_t
D	0.52	0.08	0.12	0

Again, only anti-k_t has a typical area that does **not** increase with p_t

Resiliency: backreaction

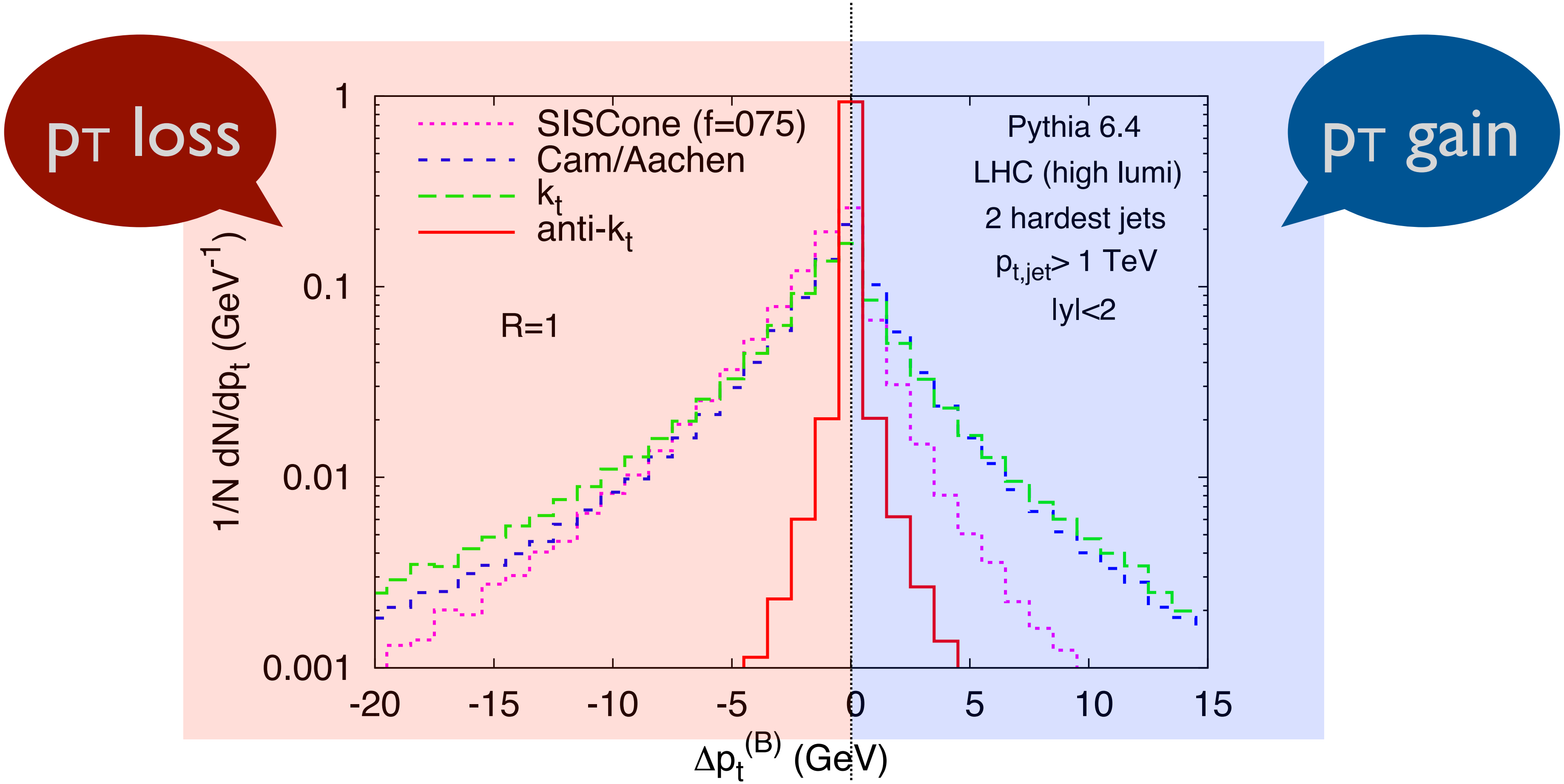
"How (much) a jet changes when immersed in a background"

Without background | With background



Resiliency: backreaction

MC, Salam, Soyez, arXiv:0802.1188



Anti-k_t jets are much more resilient to changes from background immersion

The IRC safe algorithms

	Speed	Regularity	UE	Backreaction	Hierarchical substructure
k_t	☺ ☺ ☺	☂	☂☂	☁☁	☺ ☺
Cambridge /Aachen	☺ ☺ ☺	☂	☂	☁☁	☺ ☺ ☺
anti-k_t	☺ ☺ ☺	☺ ☺	☁/ ☺	☺ ☺	✗
SISCone	☺	☁	☺ ☺	☁	✗

Tools

Background characterisation and subtraction

Mass reconstruction

Remove soft contamination from a hard jet

Tag heavy objects originating the jet

Eventually leading to ‘**third-generation**’ jet algorithms

Tools

Background characterisation and subtraction

Mass reconstruction

Remove soft contamination from a hard jet

Tag heavy objects originating the jet

Eventually leading to '**third-generation**' jet algorithms

UE characterisation

Jet algorithms like k_t or Cambridge/Aachen allow one to determine
on an event-by-event basis
the **"typical" level of transverse momentum density**
of a **roughly uniform background noise**:

$$\rho \equiv \underset{\text{(over a \underline{single} event)}}{\text{median}} \left[\left\{ \frac{p_t^{jet}}{\text{Area}_{jet}} \right\} \right]$$

MC, Salam, 2007

This ρ value can, in turn, be used to characterise the UE

Since this measurement is done with the jets, it is alternative/complementary to the usual analyses done using charged tracks (à la R. Field)

Hard jets and background

MC, Salam, arXiv:0707.1378
MC, Salam, Soyez, arXiv:0802.1188

Modifications of the hard jet

$$p_{t,jet}^{AA} = p_{t,jet}^{pp} + \boxed{\rho A_{jet} \pm \sigma \sqrt{A_{jet}}} + \boxed{\Delta p_t^{BR}}$$

hard jet

background → 'susceptibility'

back-reaction → 'resiliency'

Background subtraction

In FastJet (v3 only)

```
BackgroundEstimator bkgd(const std::vector<PseudoJet> &jets,
                         const Selector &rho_range);

double rho = bkgd.rho(jet);

PseudoJet area_4vector jet.area_4vector();

PseudoJet subtracted_jet = jet - rho*area_4vector;
```

NB. BackgroundEstimator still preliminary in FastJet 3.0alpha2

Tools

Background characterisation and subtraction

Mass reconstruction

Remove soft contamination from a hard jet

Tag heavy objects originating the jet

Eventually leading to ‘**third-generation**’ jet algorithms

R-dependent effects

Perturbative radiation: $$\Delta p_t \simeq \frac{\alpha_s(C_F, C_A)}{\pi} p_t \ln R$$

Hadronisation: $$\Delta p_t \simeq \frac{(C_F, C_A)}{R} \times 0.4 \text{ GeV}$$

Underlying Event: $$\Delta p_t \simeq \frac{R^2}{2} \times (2.5 \text{ — } 15 \text{ GeV})$$

Tevatron LHC

Analytical estimates,
Dasgupta, Magnea, Salam, arXiv:0712.3014

Which R to choose?

The value of R matters because it affects, in opposite ways, a number of things:

Small R:

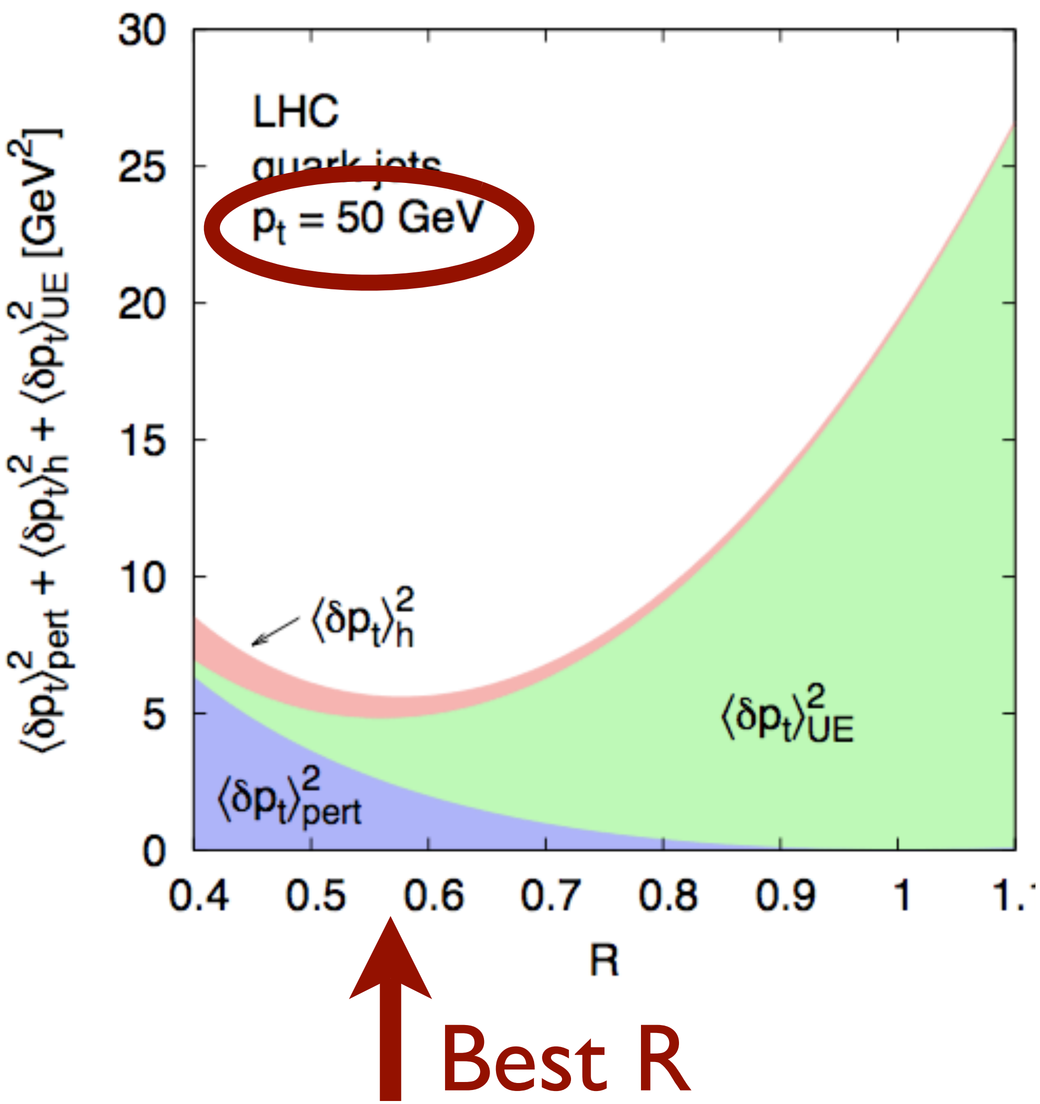
- Limit underlying event and pileup contamination
- Better resolve many-jets events

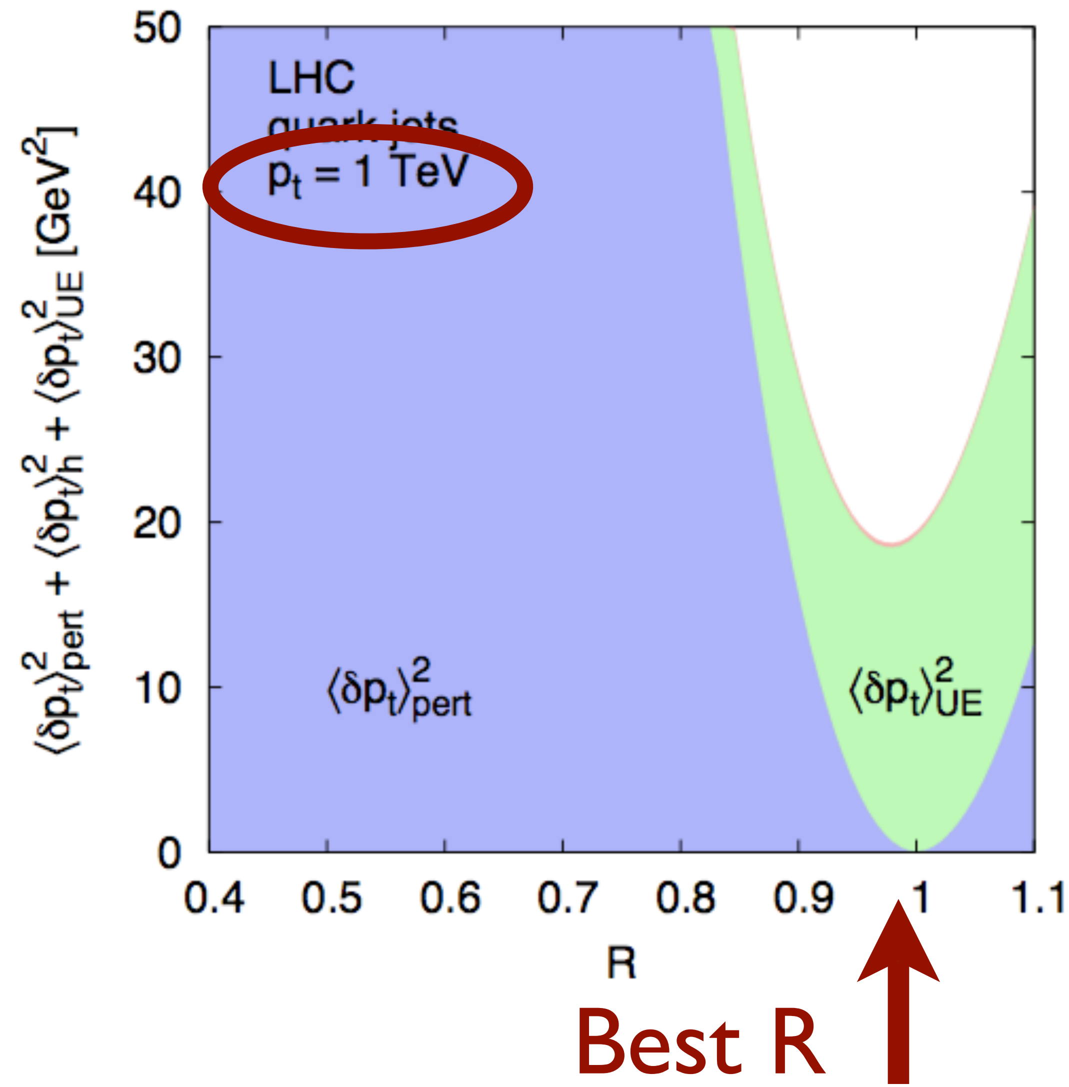
Large R:

- Limit perturbative radiation loss (‘out-of-cone’)
- Limit non-perturbative hadronisation effects

The best compromise will in general depend on the specific observable

Minimize $\Sigma(\Delta p_t)^2$

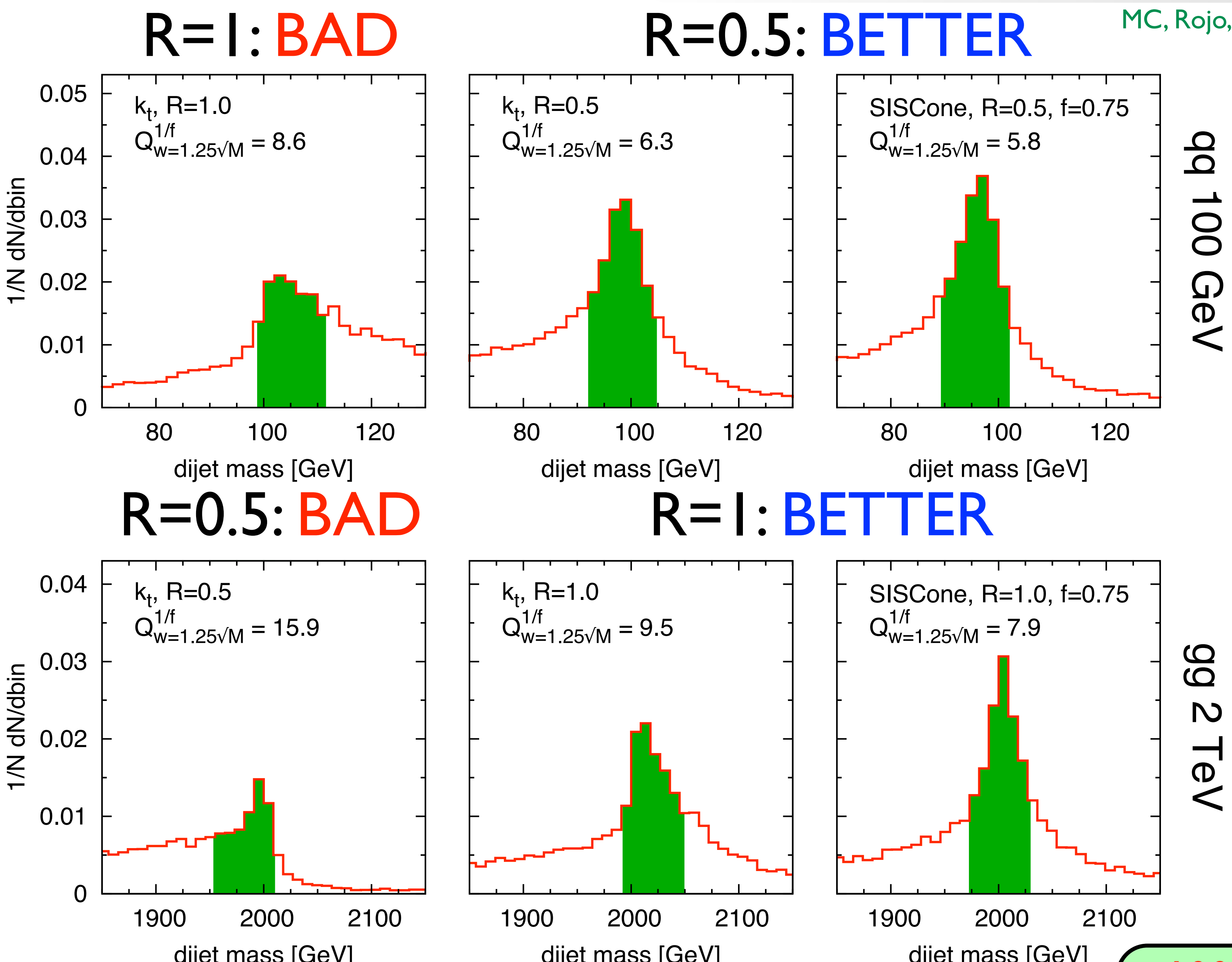




Dasgupta, Magnea, Salam, arXiv:0712.3014

Reconstruction of a di-jet mass peak

MC, Rojo, Salam, Soyez, arXiv:0810.1304



qq jets at 100 GeV

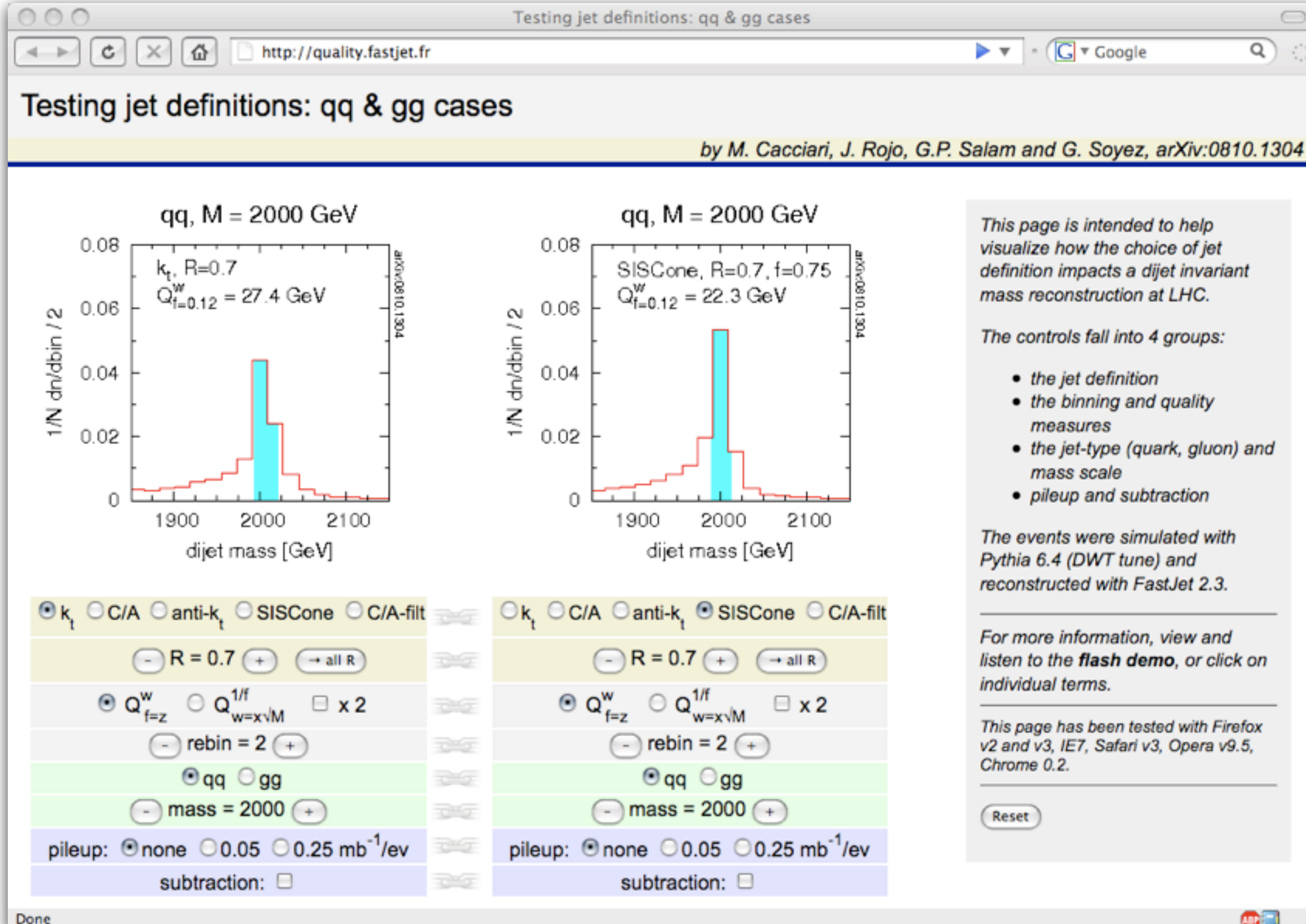
gg jets at 2 TeV

Gluons (and heavy objects) prefer larger R

~ 100000 similar plots at http://quality.fastjet.fr

quality.fastjet.fr

Do it yourself



Tools

Background characterisation and subtraction

Mass reconstruction

Remove soft contamination from a hard jet

Tag heavy objects originating the jet

Eventually leading to '**third-generation**' jet algorithms

‘Jet substructure’ in SPIRES

Out of 20, most came after:

15. **Jet substructure as a new Higgs search channel at the LHC.**
Jonathan M. Butterworth, Adam R. Davison (University Coll. London), Mathieu Rubin, Gavin P. Salam (Paris, LPTHE).
Published in **Phys.Rev.Lett. 100 (2008) 242001**
e-Print: **arXiv:0802.2470 [hep-ph]**

Jet substructure as tagger

Studying the ***jet substructure***

(i.e. the subjets obtained by undoing the clustering of a sequential recombination algorithm)

can lead to **identification capabilities** of specific objects

(as opposed to 'standard' QCD background)

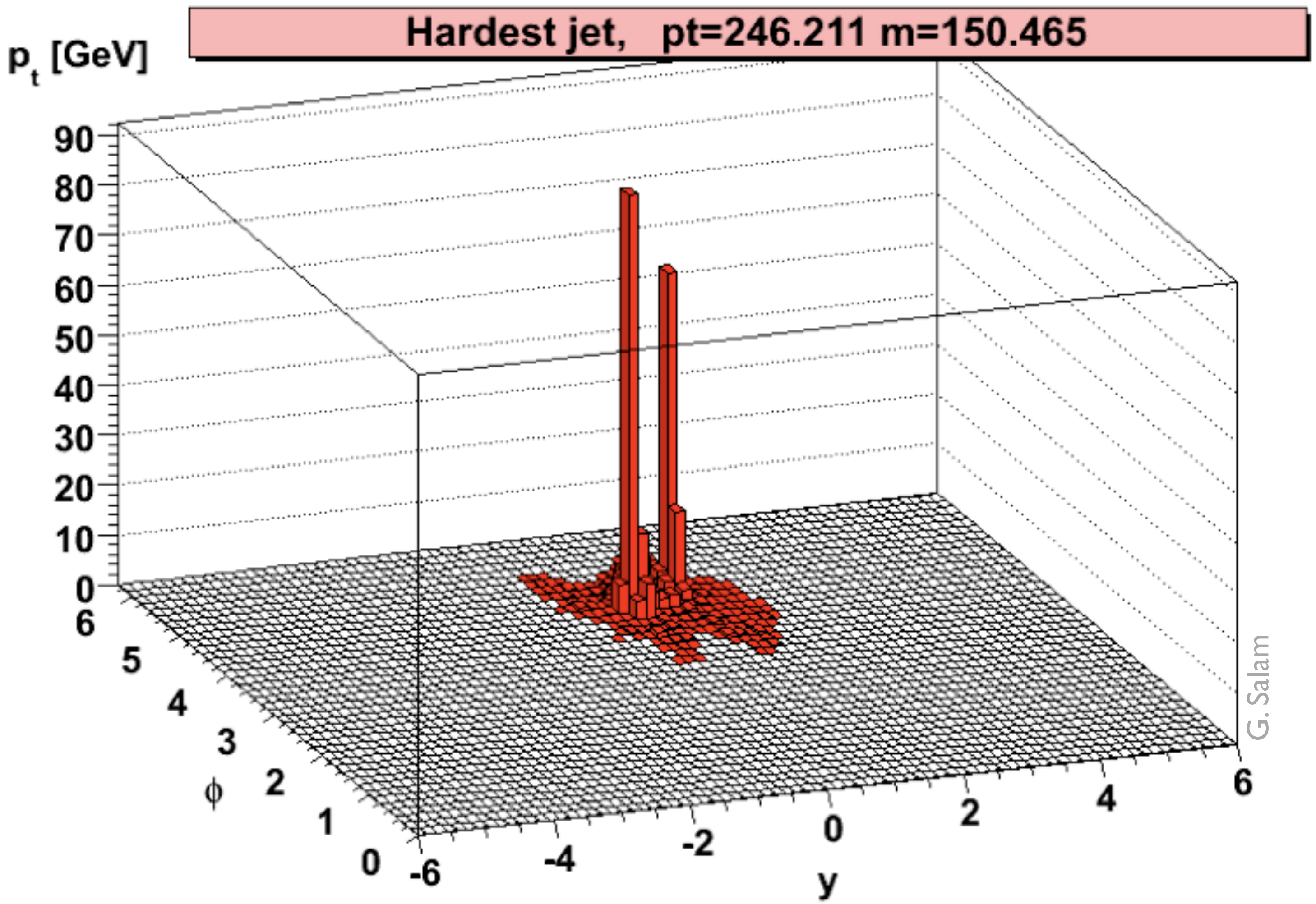
- Boosted Higgs tagger — Butterworth, Davison, Rubin, Salam, 2008
- Boosted top tagger — Kaplan, Rehermann, Schwartz, Tweedie, 2008; Thaler, Wang, 2008; G. Broojmans, ATLAS 2008
- Moderately boosted top and Higgs tagger — Plehn, Salam, Spannowsky, 2009
- + others

Common feature: start with a 'fat jet', decluster it and check if it contains a complex 'hard' substructure

Boosted Higgs tagger

$pp \to ZH \to \nu\bar{\nu}b\bar{b}$

Butterworth, Davison, Rubin, Salam, 2008



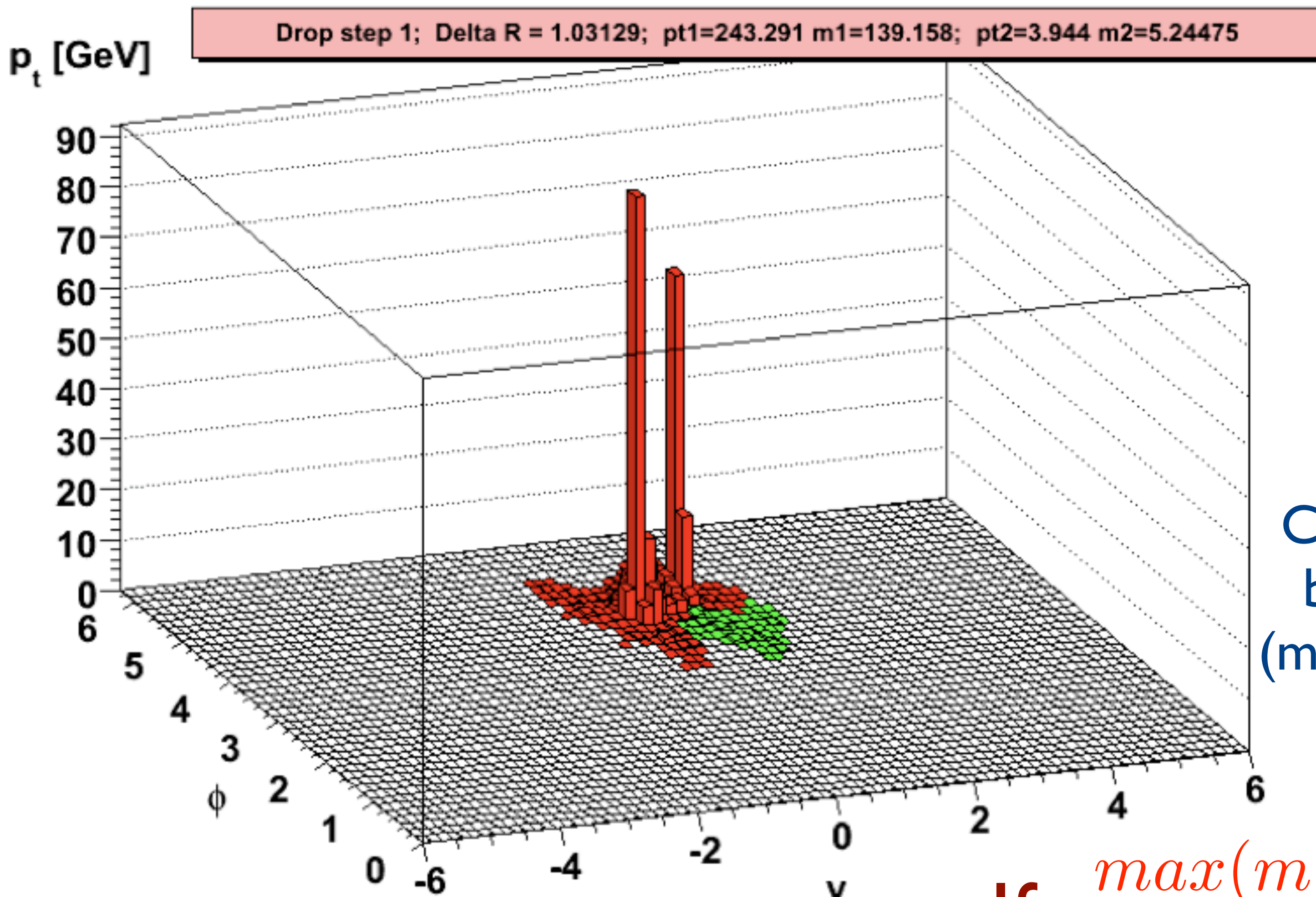
Start with the hardest jet

Use C/A with large R=1.2

m = 150 GeV

Boosted Higgs tagger

pp →ZH → ννbb



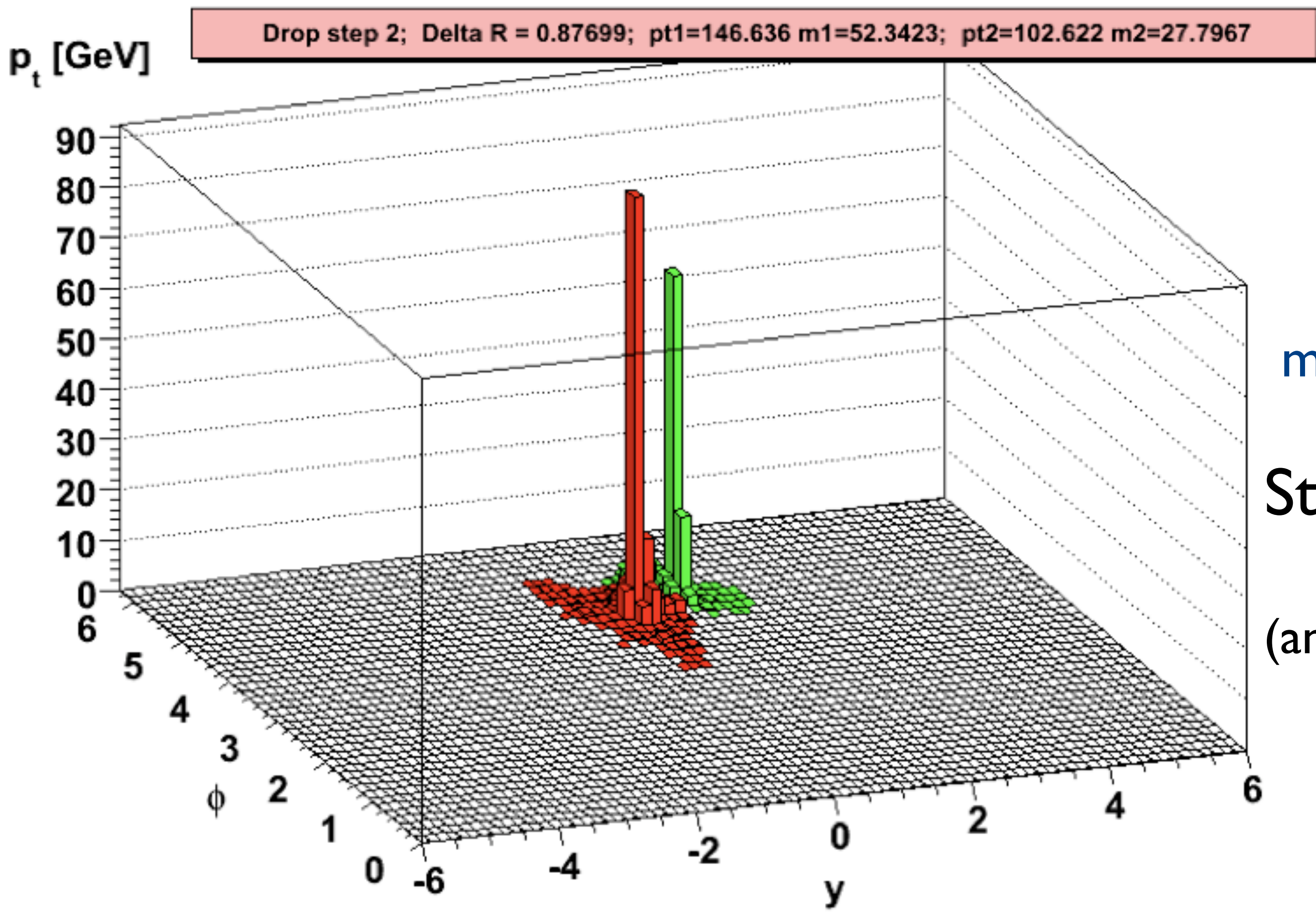
Undo last step of clustering

Check how the mass splits between the two subjects (m₁ = 139 GeV, m₂ = 5 GeV)

If $\frac{max(m_1, m_2)}{m} > \mu$ repeat

Boosted Higgs tagger

pp →ZH → ννbb



m_1 = 52 GeV, m_2 = 28 GeV

Stop when a large mass drop is observed

(and recombine these two jets)

Jet substructure as filter

The ***jet substructure***
can be exploited to help **removing contamination**
from a soft background

- Jet 'filtering' — Butterworth, Davison, Rubin, Salam, 2008
- Jet 'pruning' — S. Ellis, Vermilion, Walsh, 2009
- Jet 'trimming' — Krohn, Thaler, Wang, 2009

Aim: limit sensitivity to background while retaining bulk of perturbative radiation

(Filtering, trimming and pruning are actually quite similar)

Cambridge/Aachen with filtering

Butterworth, Davison, Rubin, Salam, arXiv:0802.2470

An example of a **third-generation** jet algorithm

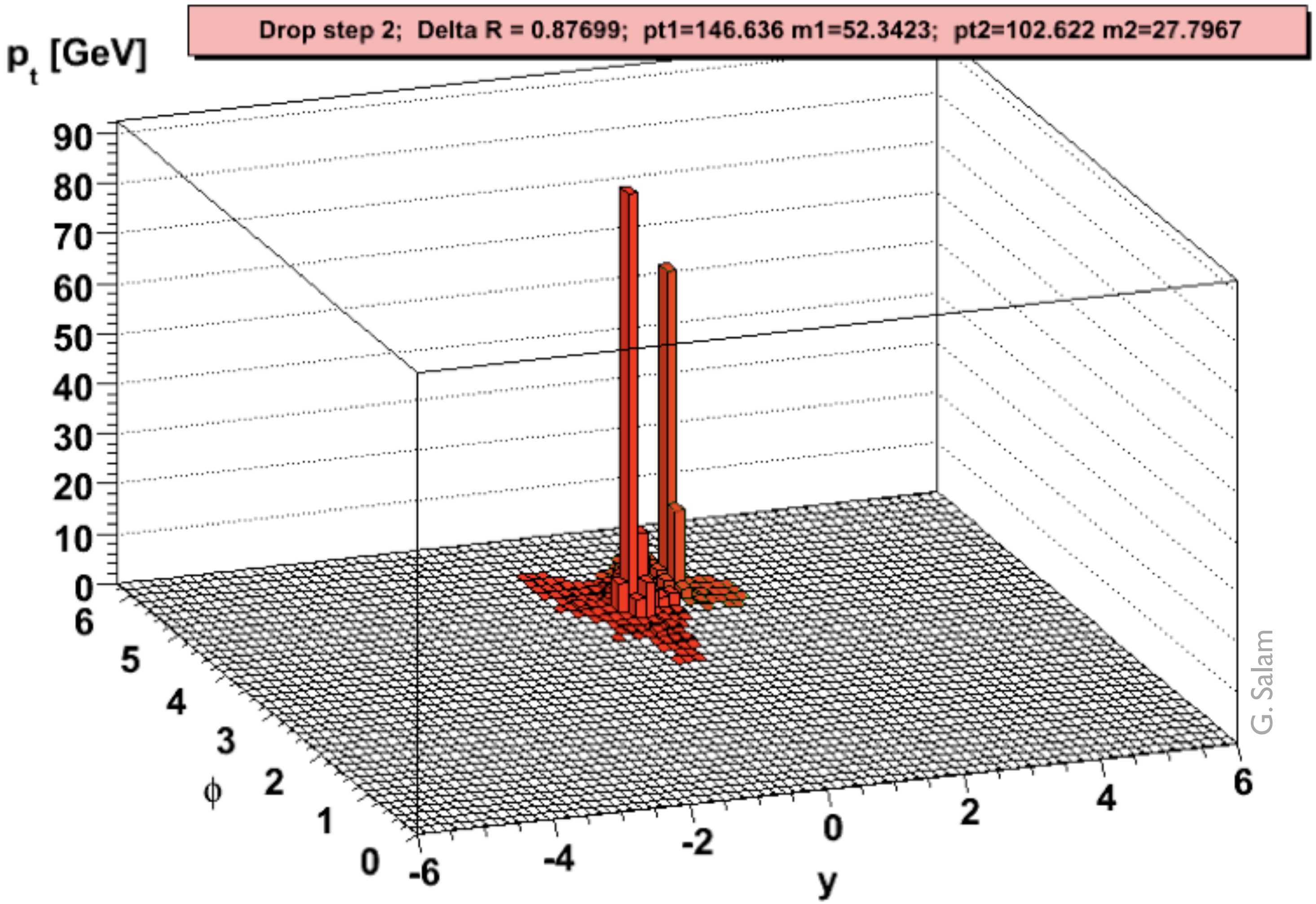
- Cluster with C/A and a given R
- Undo the clustering of each jet down to subjets with radius $\mathbf{x_{filt}R}$
- Retain only the $\mathbf{n_{filt}}$ hardest subjets

In FastJet (v3 only)

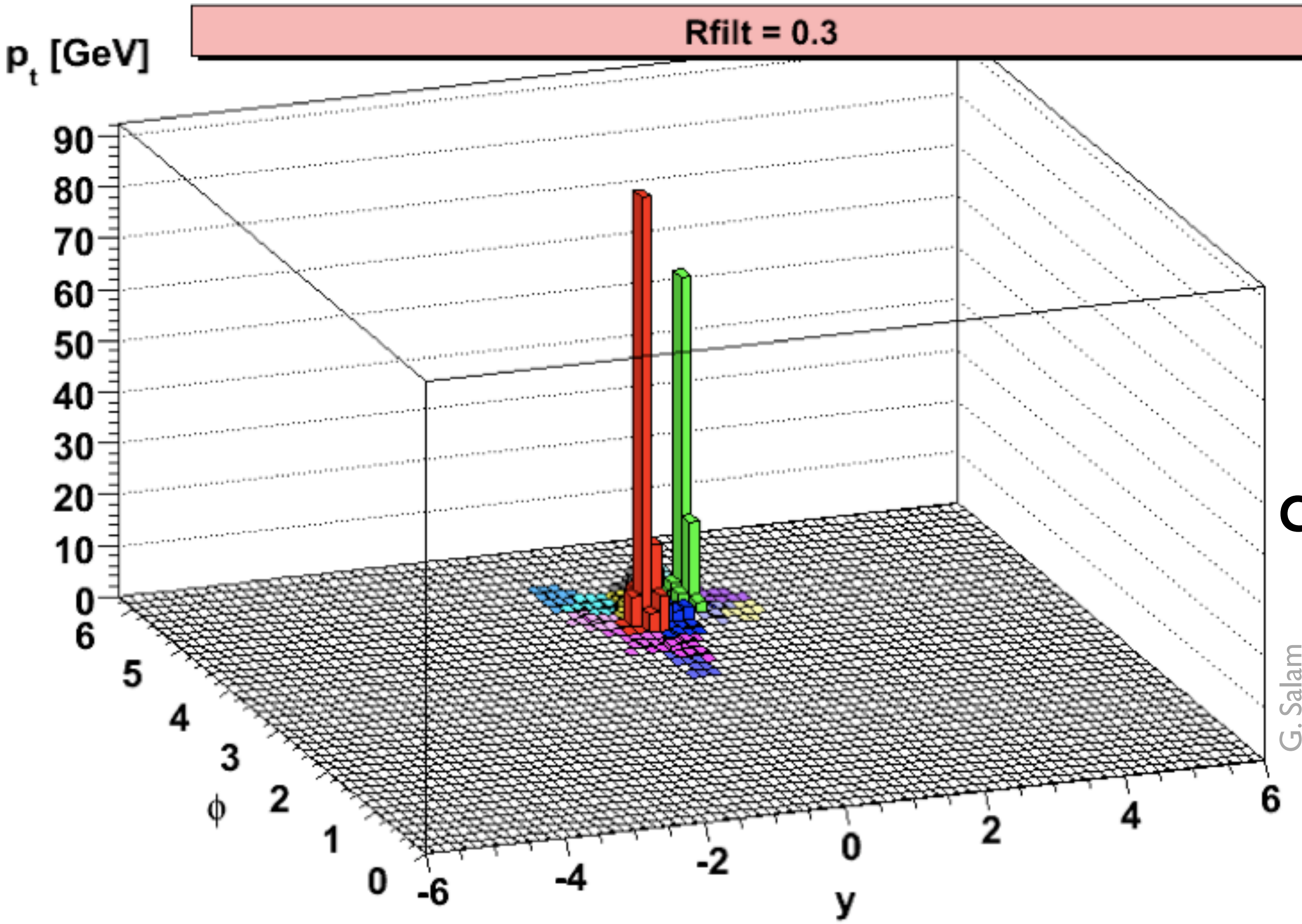
```
Filter filter(JetDefinition(cambridge_algorithm,xfilt*R),
              SelectorNHardest(nfilt));

PseudoJet filtered_jet = filter(jet);
```

Filtering in action

Butterworth, Davison, Rubin, Salam, arXiv:0802.2470

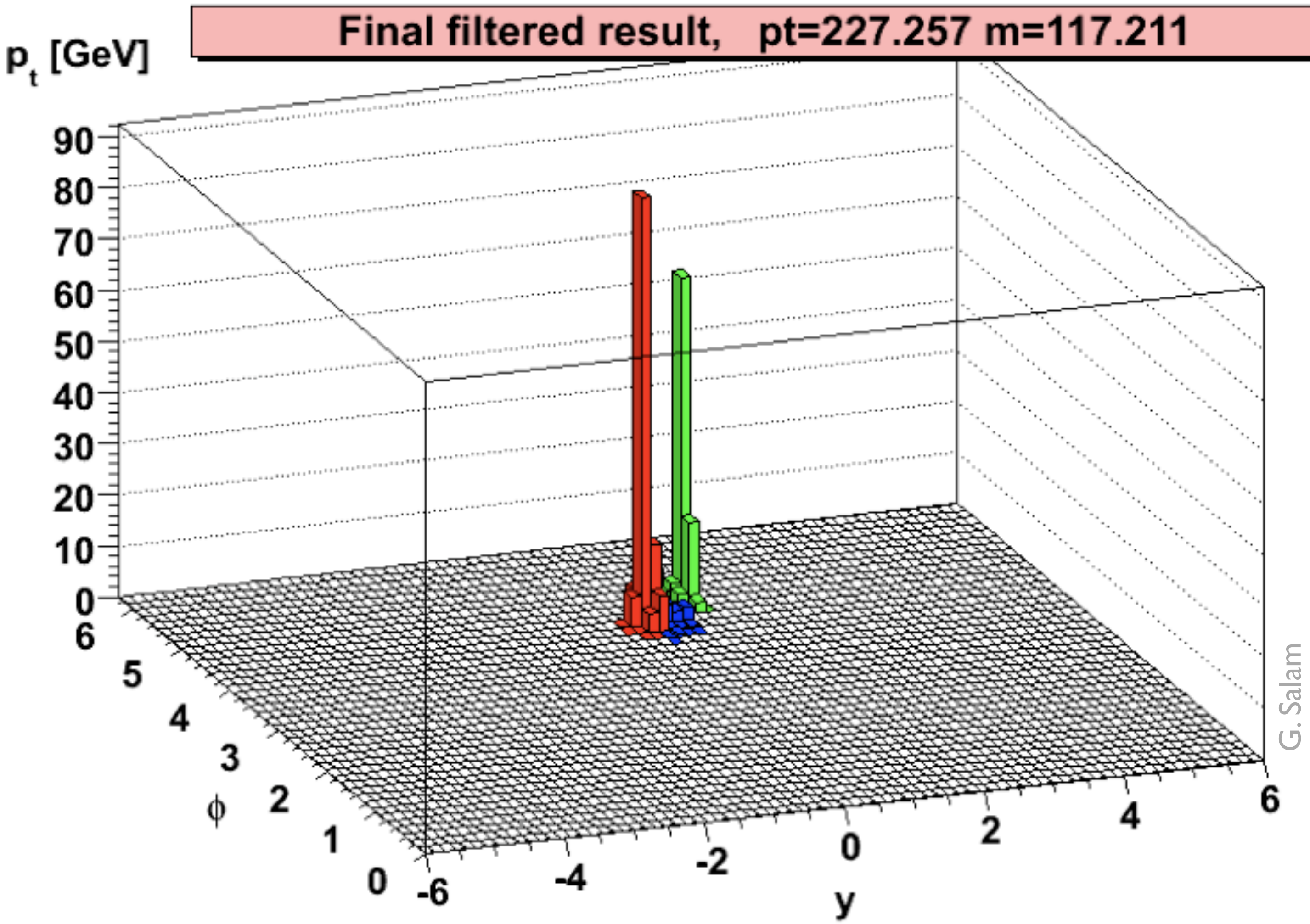


Start with a jet



Recluster the contituents with R_{filt}

Filtering in action



Only keep the n_{filt} hardest jets

The low-momentum stuff surrounding the hard particles has been removed

Jet trimming

Krohn, Thaler, Wang, 2009

1. Cluster all cells/tracks into jets using any clustering algorithm. The resulting jets are called the seed jets.

2. Within each seed jet, recluster the constituents using a (possibly different) jet algorithm into subjets with a characteristic radius $R_{\rm sub}$ smaller than that of the seed jet.

3. Consider each subjet, and discard the contributions of subjet i to the associated seed jet if $p_{Ti} < f_{\rm cut} \cdot \Lambda_{\rm hard}$, where $f_{\rm cut}$ is a fixed dimensionless parameter, and $\Lambda_{\rm hard}$ is some hard scale chosen depending upon the kinematics of the event.

4. Assemble the remaining subjets into the trimmed jet.

Different condition for retaining jets (p_T-cut rather than $n_{\rm filt}$ hardest) with respect to filtering

In FastJet (v3 only)

```
Filter trimmer(JetDefinition(cambridge_algorithm,xfilt*R),
               SelectorPtMin(fcut*Lambdahard));

PseudoJet trimmed_jet = trimmer(jet);
```

Jet pruning

S. Ellis, Vermilion, Walsh, 2009

0. Start with a jet found by any jet algorithm, and collect the objects (such as calorimeter towers) in the jet into a list L. Define parameters $D_{\rm cut}$ and $z_{\rm cut}$ for the pruning procedure.

1. Rerun a jet algorithm on the list L, checking for the following condition in each recombination $i, j \to p$:

$$z = \frac{\min(p_{Ti}, p_{Tj})}{p_{Tp}} < z_{\rm cut} \quad \text{and} \quad \Delta R_{ij} > D_{\rm cut}.$$

   This algorithm must be a recombination algorithm such as the CA or $\mathrm{k_T}$ algorithms, and should give a "useful" jet substructure (one where we can meaningfully interpret recombinations in terms of the physics of the jet).

2. If the conditions in 1. are met, do not merge the two branches 1 and 2 into p. Instead, discard the softer branch, i.e., veto on the merging. Proceed with the algorithm.

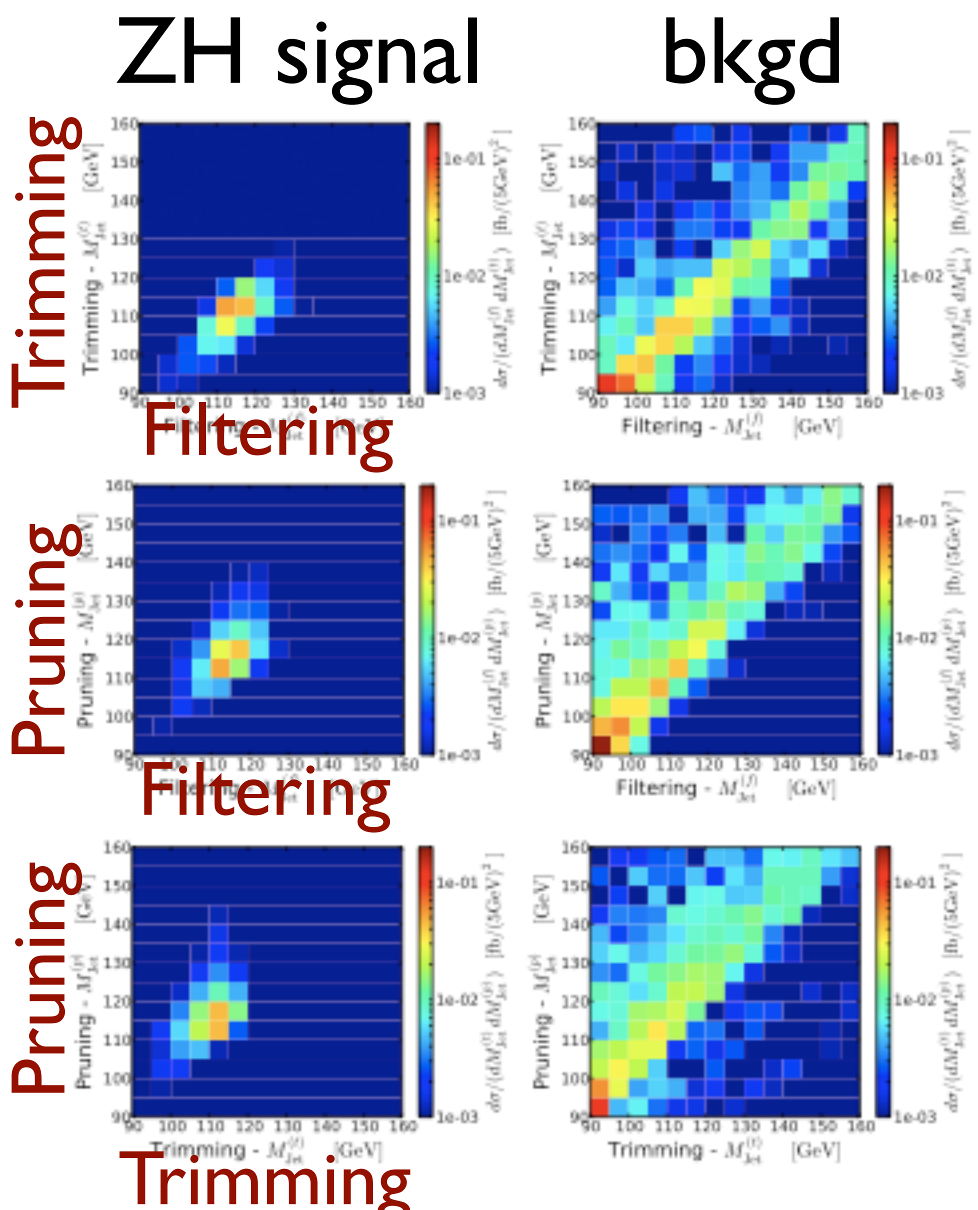
3. The resulting jet is the *pruned jet*, and can be compared with the jet found in Step 0.

True in general for substructure studies

Exclude soft stuff and large angle recombinations from clustering

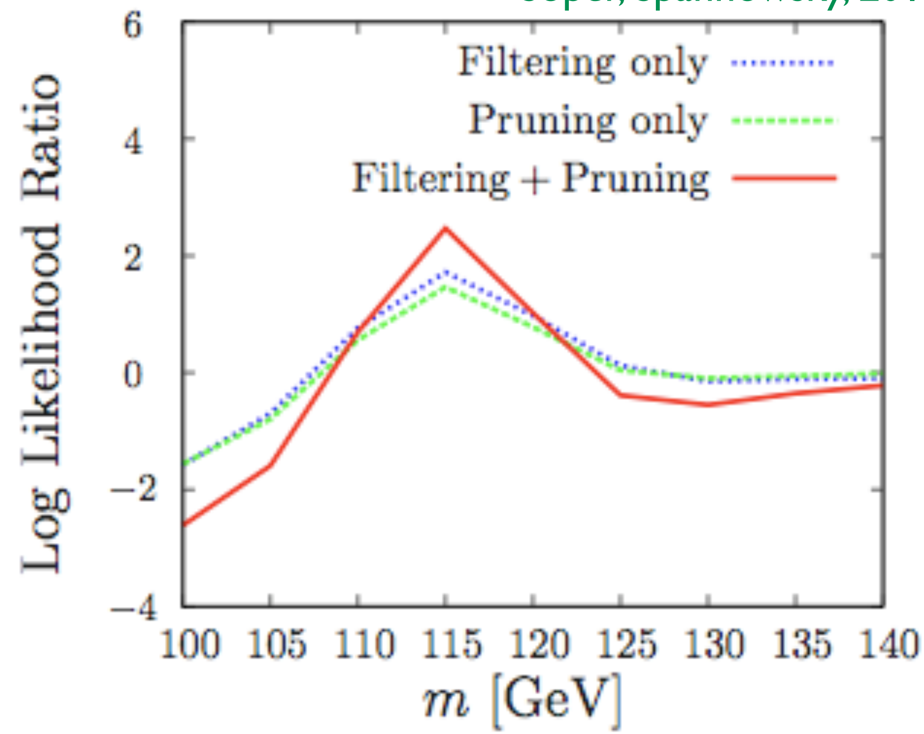
filtering + pruning + trimming

Filtering, trimming and pruning are identical in aim and spirit ('clean up' a jet, keeping the hard core but getting rid of soft contamination') but differ in details



This allows for combining them, obtaining an enhanced signal/ background significance

Soper, Spannowsky, 2010



Summary

- At least four IRC-safe algorithms exist: k_t, Cambridge/Aachen, SISCone, anti-k_t (the default algorithm of all LHC Collaborations), all available in FastJet
- A jet algorithm complemented by its parameters and the recombination scheme is called a **jet definition**
    - The proper choice of the parameters of a jet definition can considerably improve the sensitivity of an analysis
- Third-generation algorithms (e.g. tagging/filtering) appear promising for analyses where the jet substructure plays a relevant role
    - Jet areas, background subtraction and especially **jet substructure** are tools whose full potential has probably not yet been explored and exploited