Use of R-trees to improve reconstruction time in pixel trackers

Albert Pernía Vázquez [1] **Núria Valls Canudas** [2]
Elisabet Golobardes Ribé [1] Alessandro Camboni [1]
Xavier Vilasís Cardona [1]

[1]La Salle - Universitat Ramon Llull

[2]Universitat de Barcelona

April 4, 2019

The context

- Pixel tracker highly inspired in VELO Pixel.
- It outputs a list of hit units:

$$[..., hit_i\_id\{x\_position, y\_position, z\_position\}, ...]$$

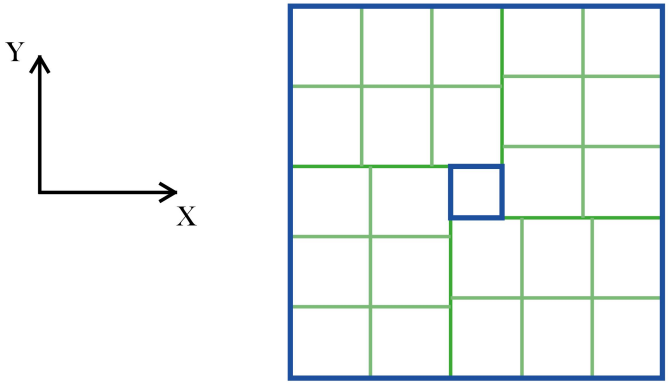


Figure 1: Sensor module

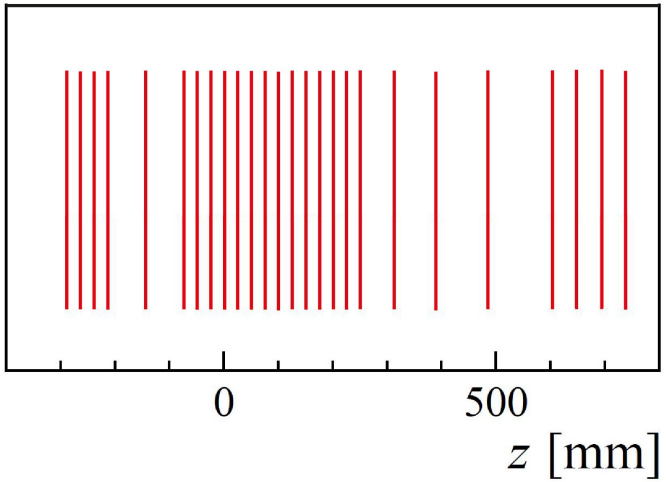


Figure 2: Z positions

Our inspiration

The Cellular Automaton based algorithm [Kisel, 2006]:

- Seed detection.
- Backward reconstruction: look for "compatible" hits in terms of distance.

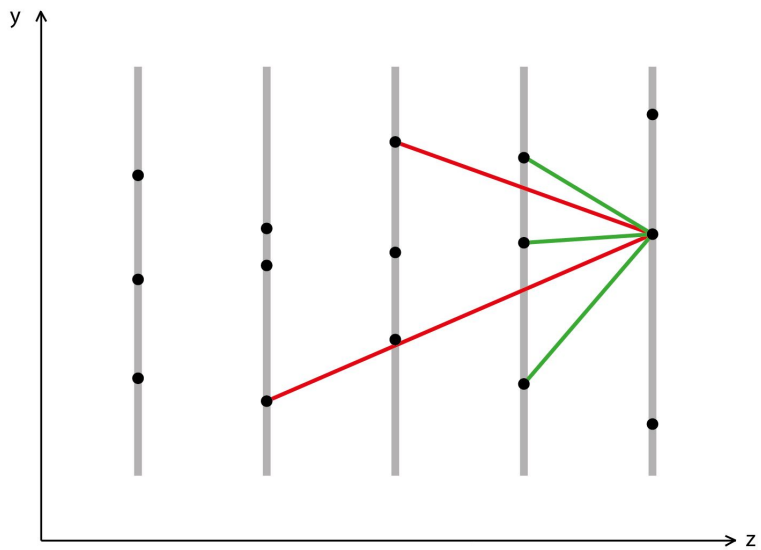


Figure 3: Tracking reconstruction

Problem

Finding the shortest distance implies comparing a specific hit with all the hits from the adjacent module.

- Linear cost O(n).

Globally, it becomes a quadratic cost O(n^2).

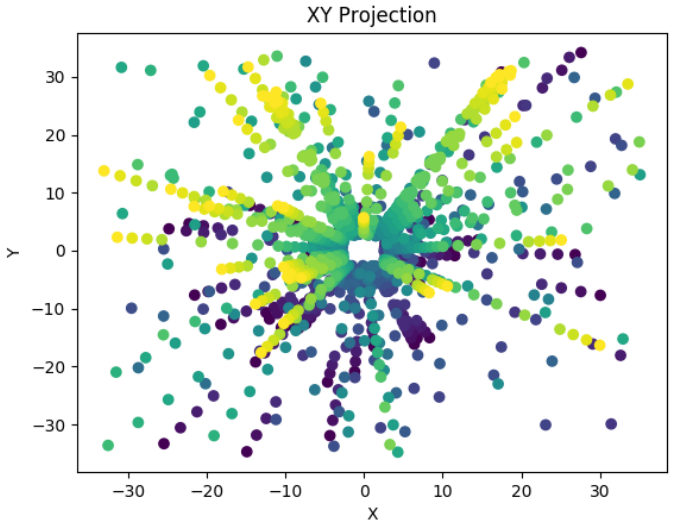


Figure 4: Example event

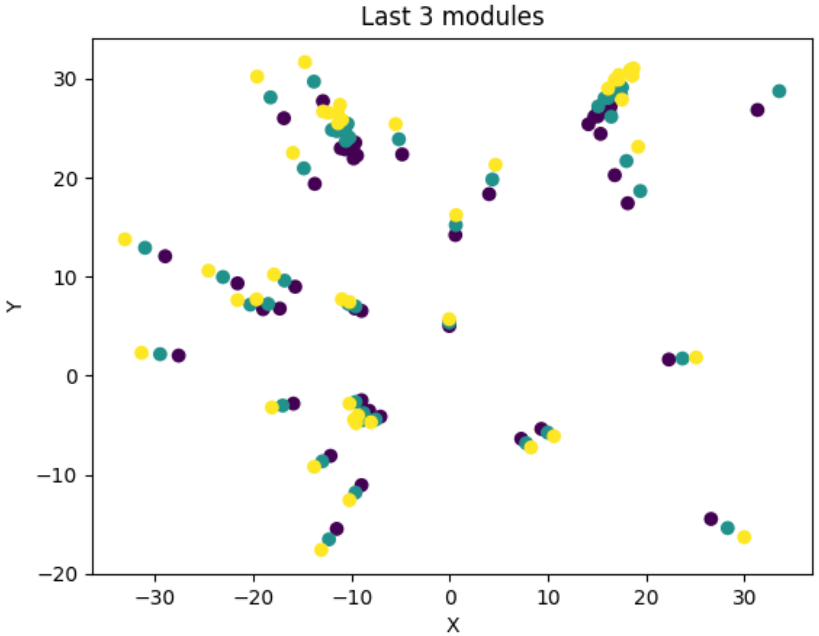


Figure 5: Last 3 modules of the event

Our proposal

Pre-process hit information and insert it into an R-Tree [Guttman, 1984] data structure:

- Gives spatial indexing to the data.
- Compares **only** close hits.
- KNN searches with logarithmic cost $O(log(n))$

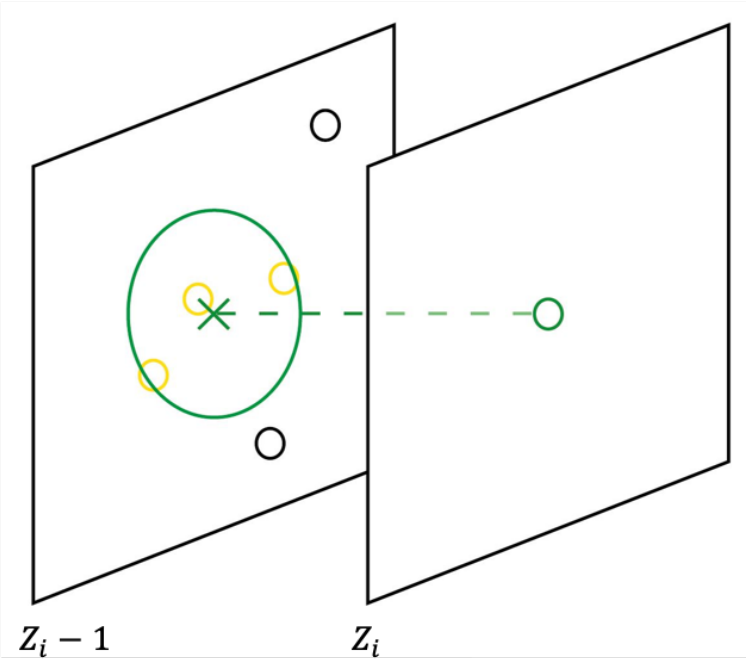


Figure 6: KNN search on the following module

R-Tree: Concept

- Proposed by Antonin Guttman [Guttman, 1984]
- Organize objects from a **D**-Dimensional space.
- Inner nodes contain M entries with M regions.
- Leaf nodes contain M objects. An object can be:
  - A rectangle.
  - A point $\rightarrow$ **hit**.

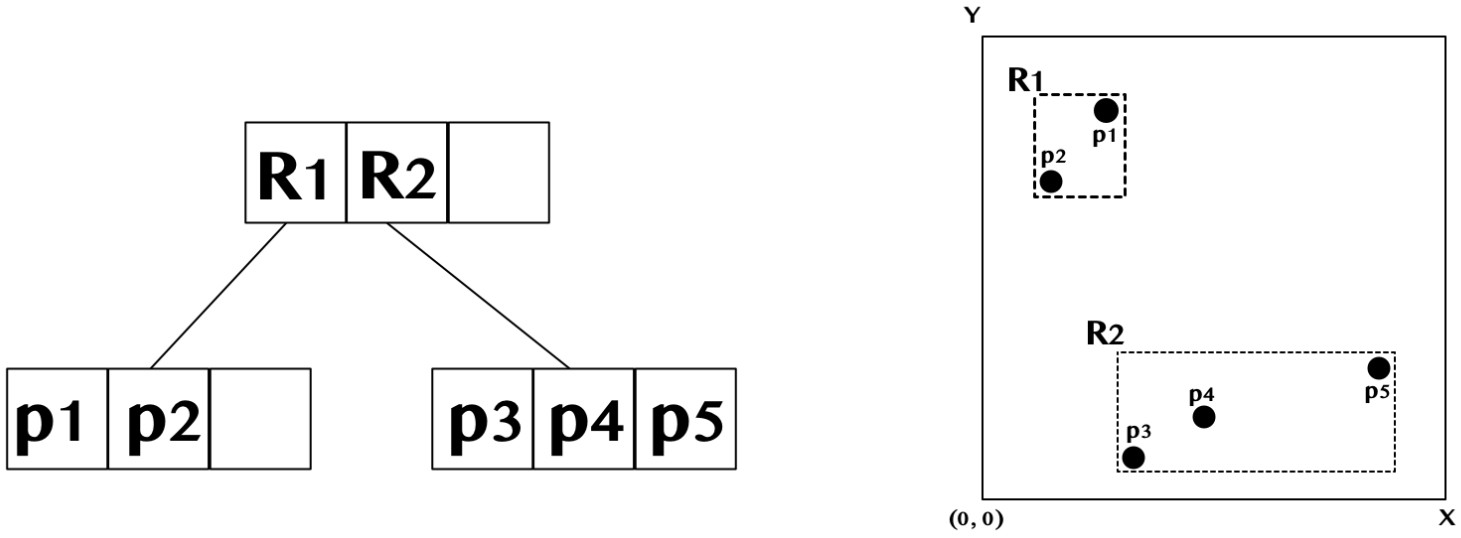


Figure 7: Example of a R-Tree with M = 3

R-Tree: Indexing

- Objects are contained in Minimum Bounding Rectangles (MBR).
- The indexing strategy must be as optimal as possible (figure 8).
- **Bad indexing** (figure 9) will imply **slow searches**.

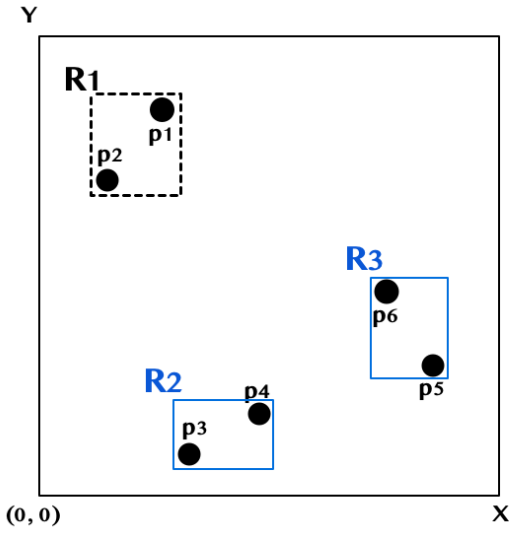


Figure 8: Good indexing

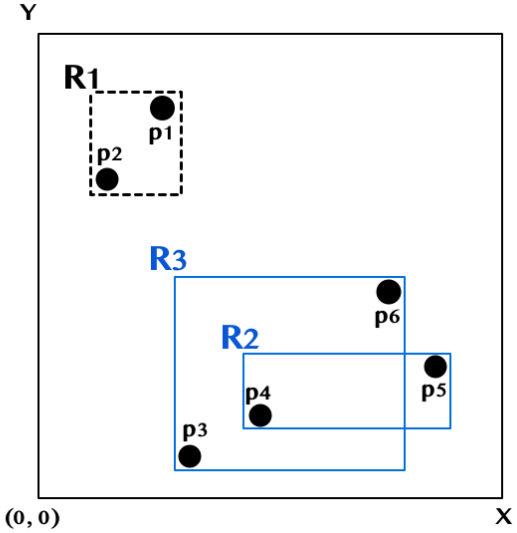
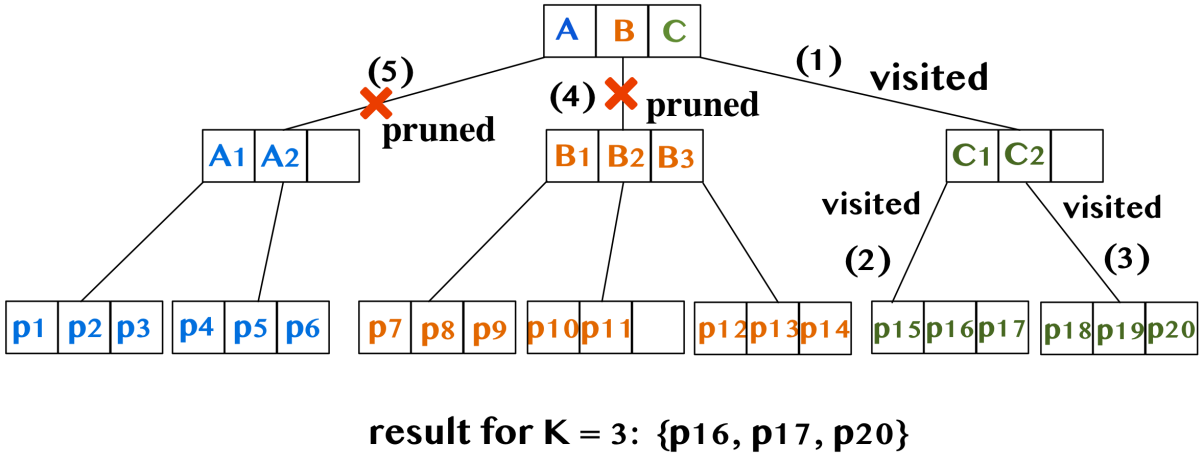


Figure 9: Bad indexing

K-Nearest Neighbour

The R-Tree [Guttman, 1984] allows the execution of KNN queries with the following benefits:

- Height of the tree is at most $log_M(n) - 1$.
- The computational cost becomes $O(log(n))$ [1].
- No need to traverse all the hits.
  - **Significantly faster than a linear search**.



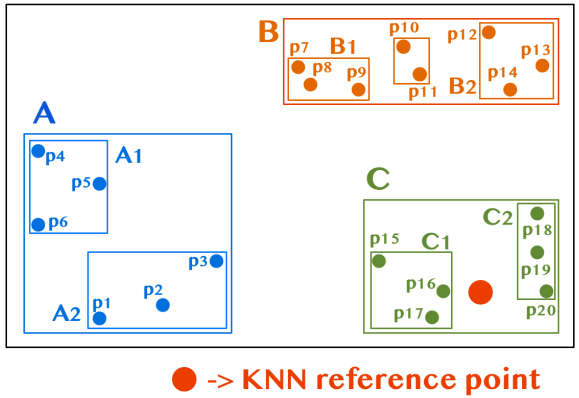


Figure 10: KNN execution example, only 3 nodes visited + root

[1] Assuming a good indexing strategy

Ad-hoc implementation: Idea

A custom implementation in C language, based on the standard R-Tree that supports:

- Choose and add different indexing methodologies.
- Optimal search queries designed for tracking algorithms.
- Application of multiple constraints.

Ad-hoc implementation: Indexing

- Supports custom indexing approaches.
- e.g. Keep hits from the same or near sensors together.

Ad-hoc implementation: K-Nearest Neighbour

Constrained queries are allowed along with the use of a different tree traversing methodology:

1. **Prioritise** those regions that contains the reference point (figures 11 & 12).
2. If needed, visit other near regions (figure 13).

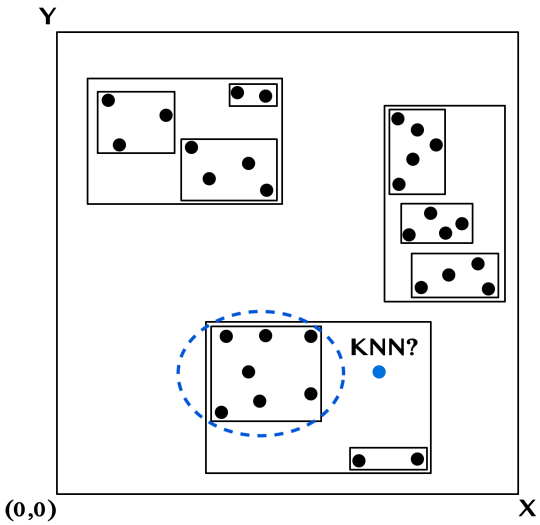


Figure 11: Step 1

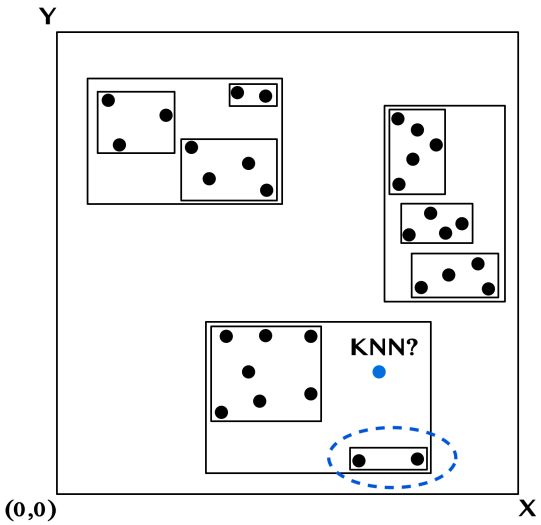


Figure 12: Step 2

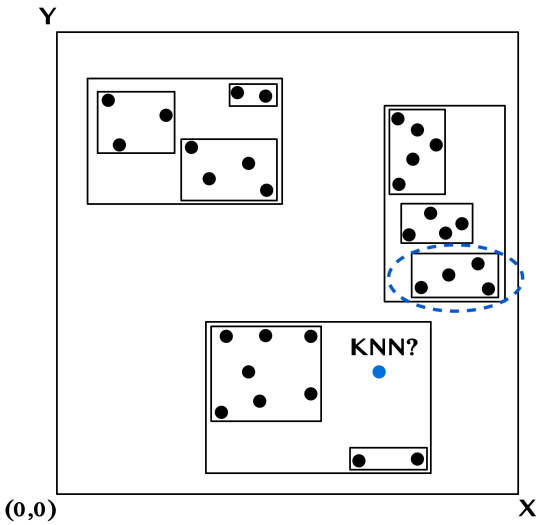
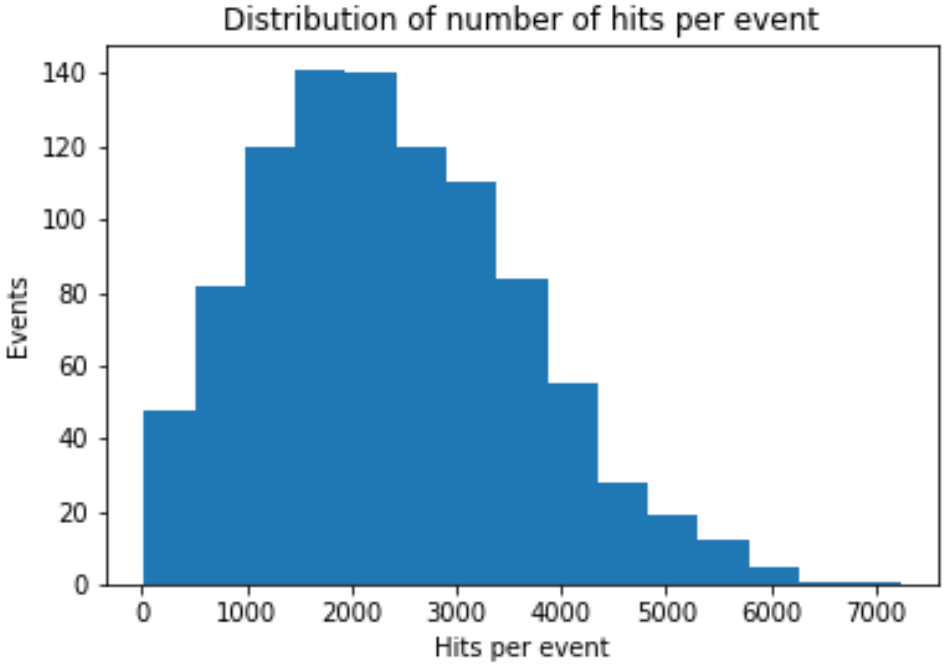


Figure 13: Step 3

Test conditions

- 966 simulated events in LHCb upgrade conditions.
- For each event and for every particle we extracted the interaction points with the detector geometry.
- Hits are grouped in a list.
- Developed in Python, accessing the R-Tree libraries:
  - Standard: programmed in C++.
  - Custom: programmed in C.
- CPU 2.7 GHz Intel Core i7, Memory 16 GB 2133 MHz



Test and results

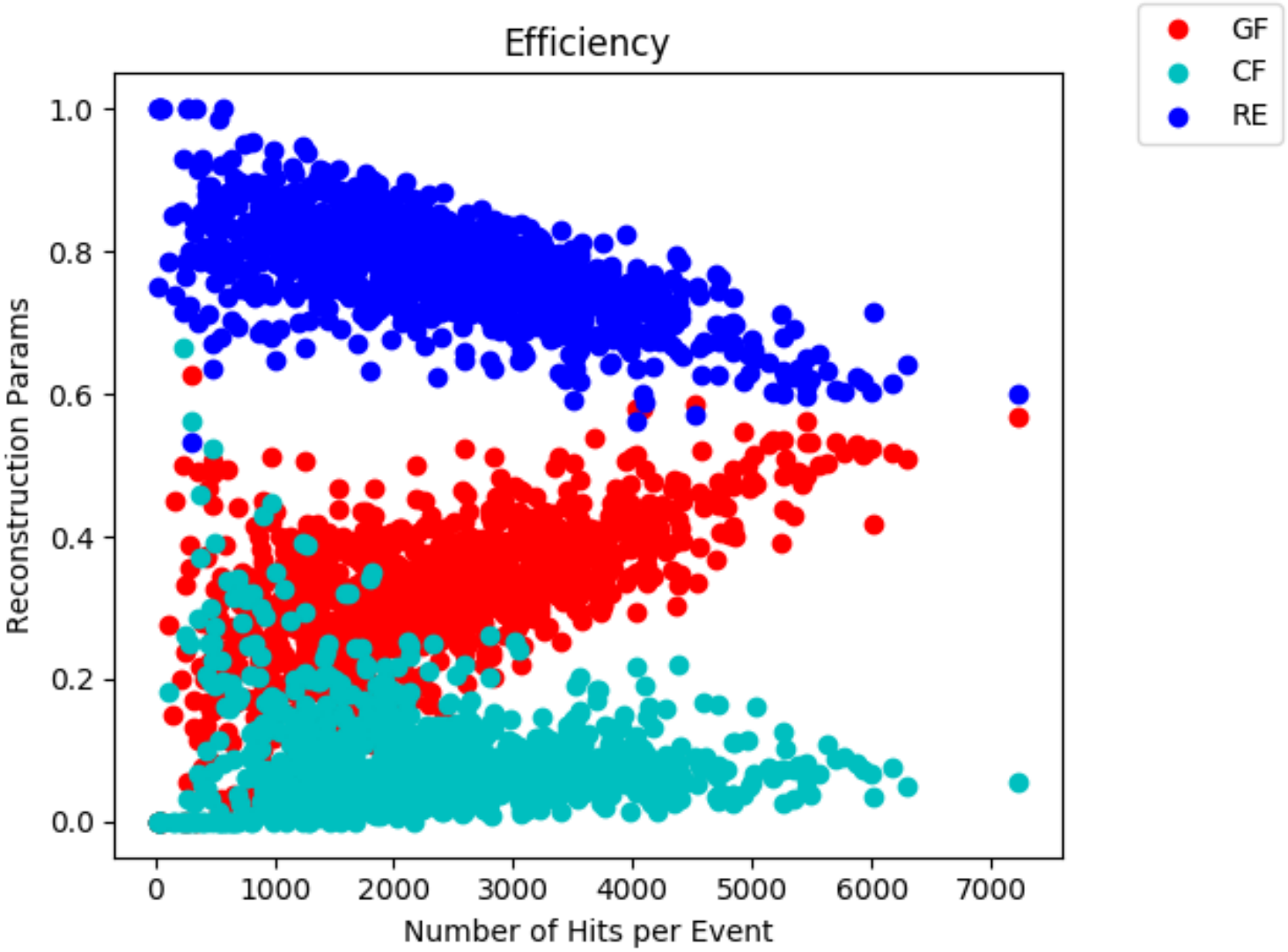


Figure 15: Using list of hits

Test and results

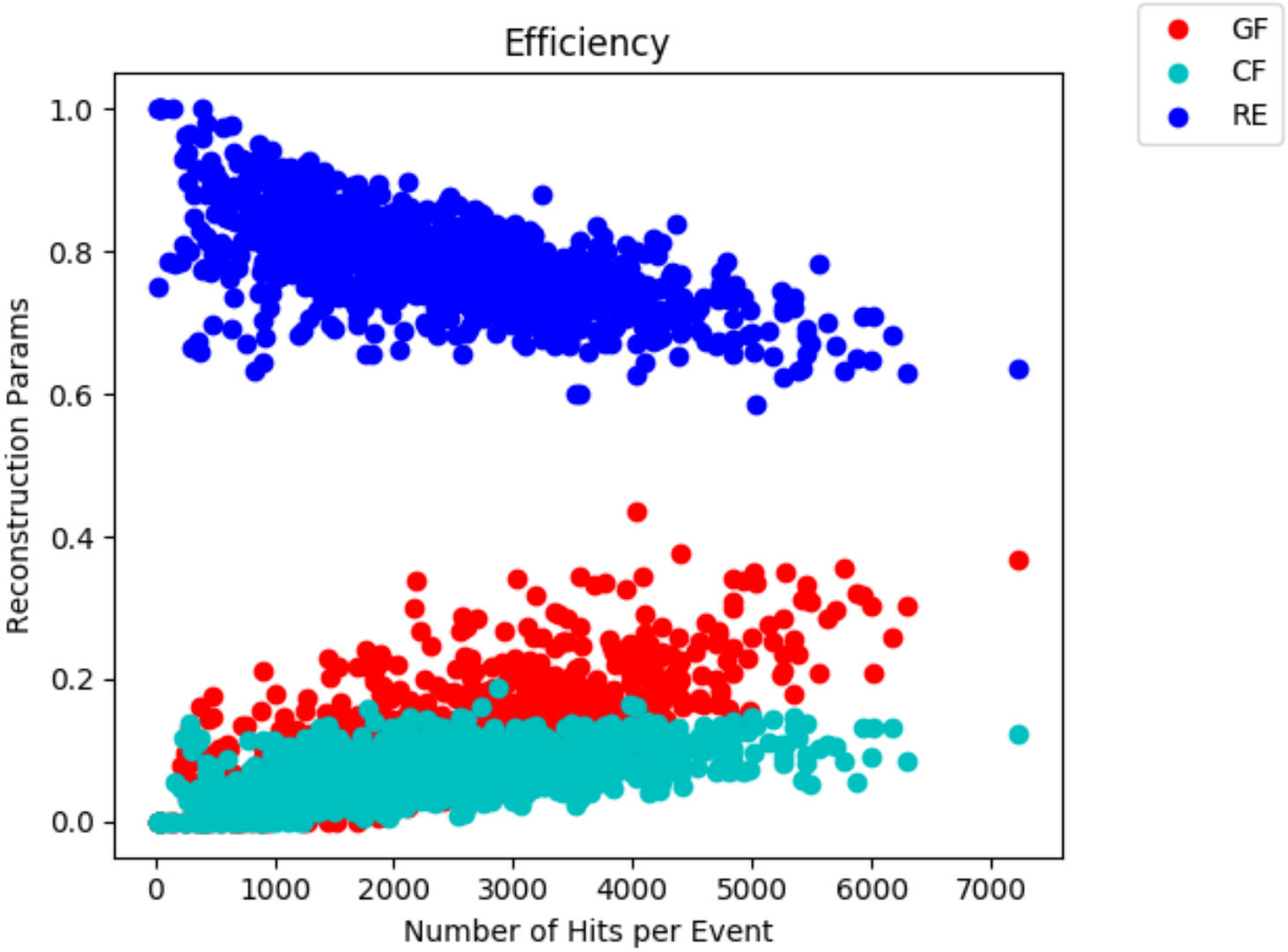


Figure 16: Generical purpose R-Tree results

Test and results

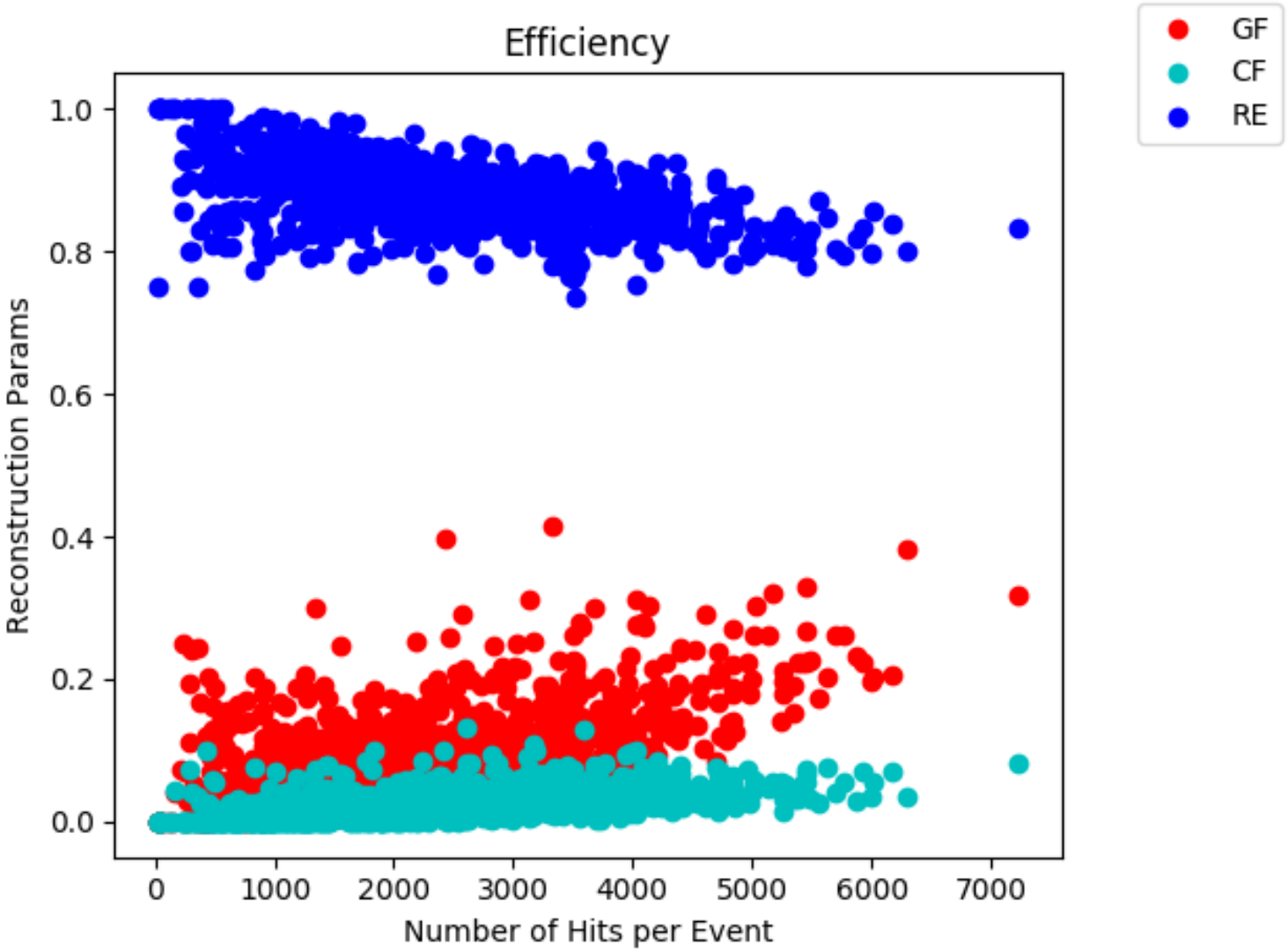


Figure 17: Ad-hoc R-Tree results

Test and results

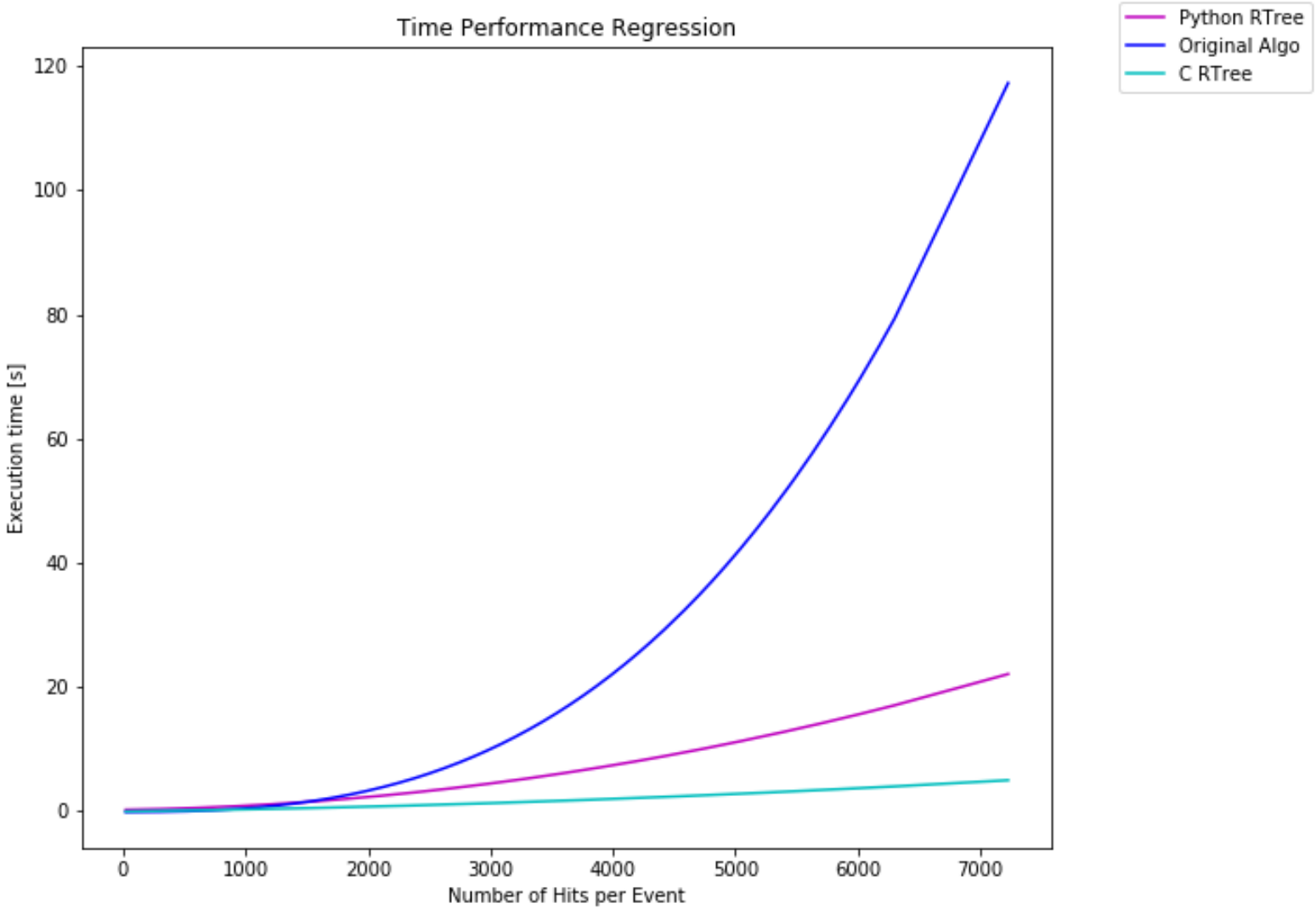


Figure 18: Time performance

Test and results

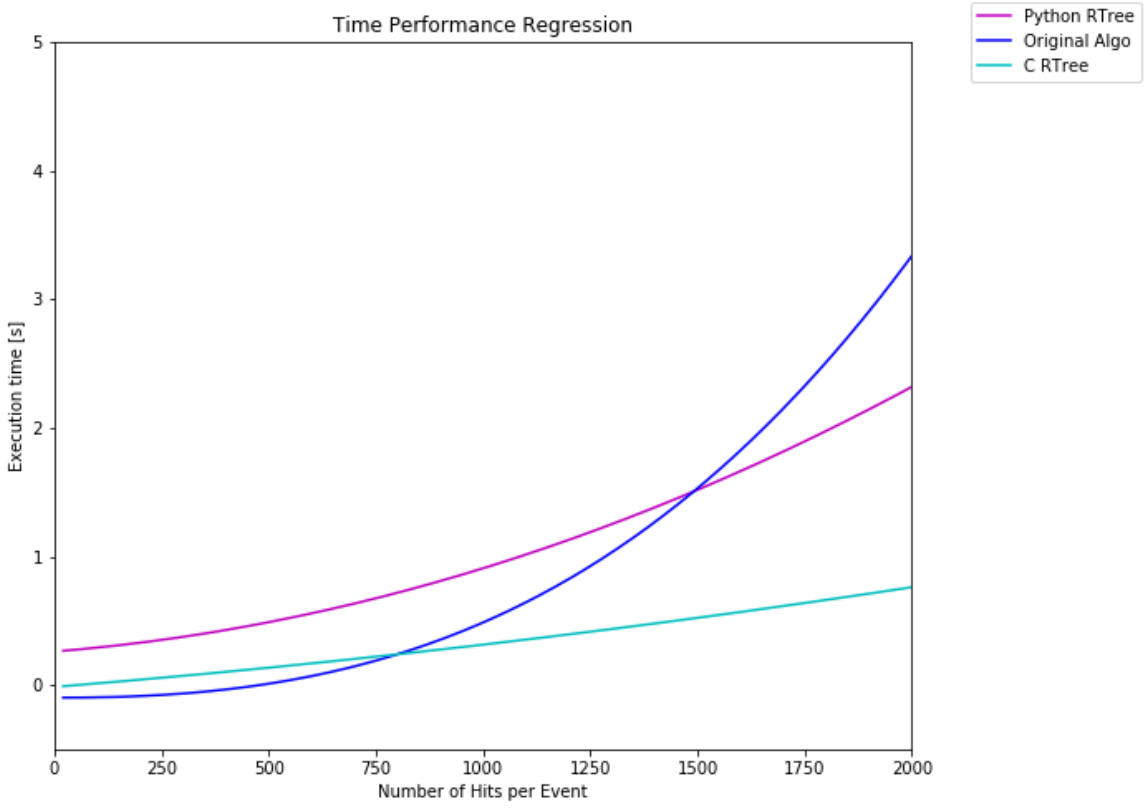


Figure 19: Time performance

- 13,6% of events under 1000 hits per event
- 9,1% of events under 800 hits per event

Summary

R-trees provide an efficient spatial indexing that improves time performance of CA-like tracking algorithms

Future prospects

- Better understanding of clone and ghost tracks
- Parellelization and GPGPU approach
- Intersection queries

References

Guttman, A. (1984).
R-trees: A dynamic index structure for spatial searching.
SIGMOD Rec., 14(2):47–57.

Kisel, I. (2006).
Event reconstruction in the cbm experiment.
Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 566:85–88.

Backup

R-Tree: Insertion

- Regions increases to cover all the hits.
- The region implying the minimum increase is chosen to cover the new hit.

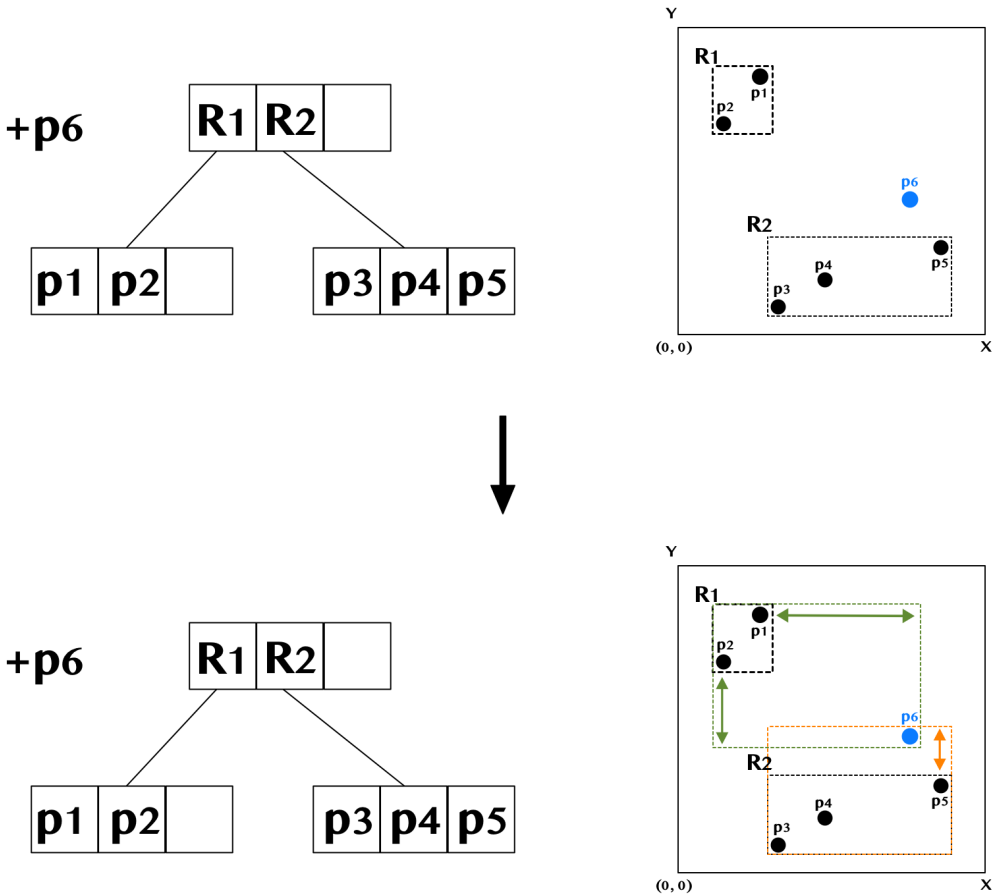


Figure 20: Insertion example

R-Tree: *Overflow*

- The node where the hit should be stored is full.
- It is needed to perform a node **split** (figure 21).

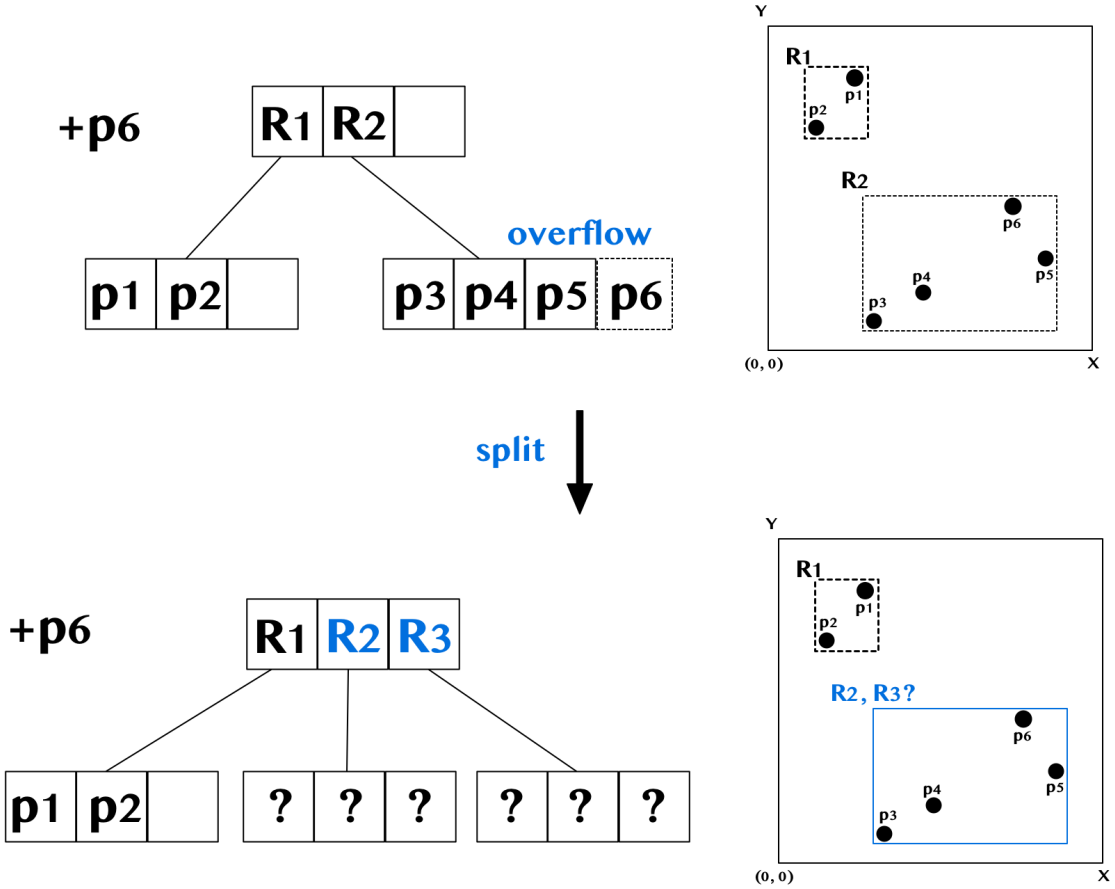


Figure 21: Hits of the node split must be distributed over the new two resulting nodes

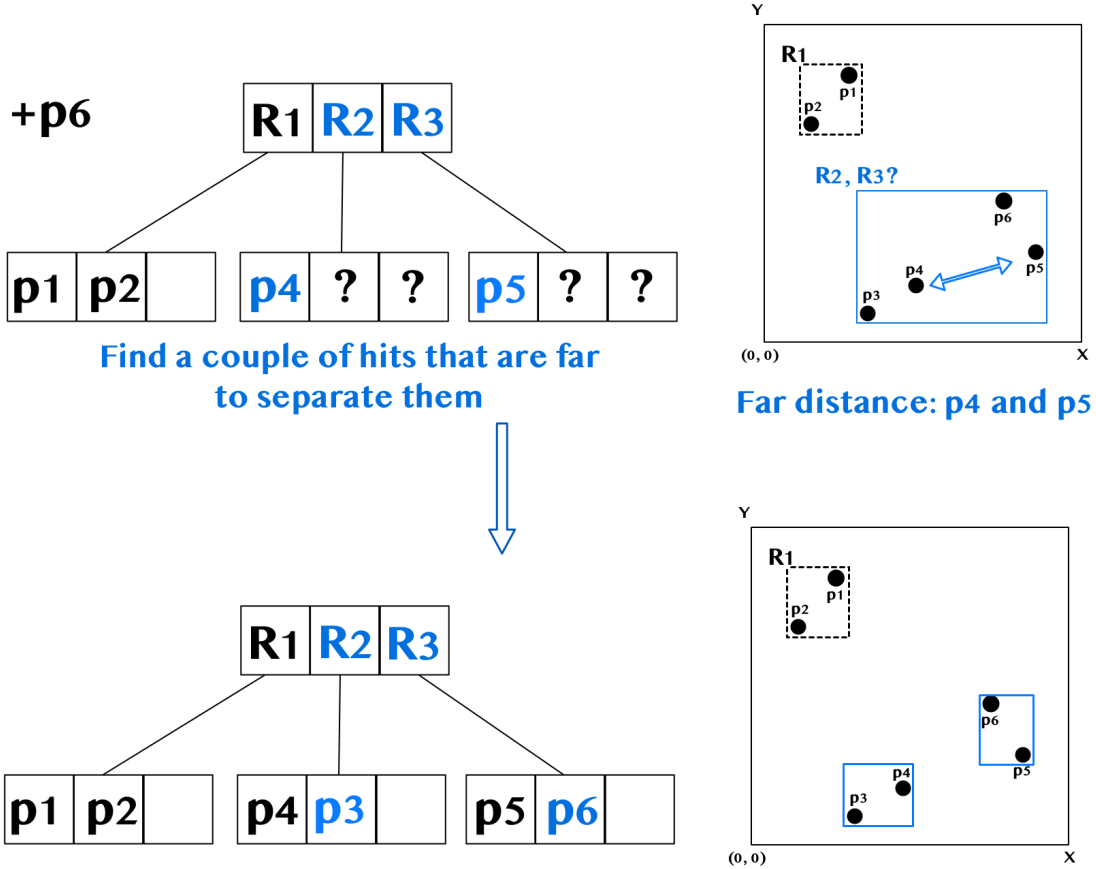
R-Tree: Split algorithms

There are many split algorithms:

- Exponential.
- Quadratic.
- Linear.

R-Tree: Linear split

1. Find and separate two hits that are far $\rightarrow \theta(n)$
2. Distribute the rest of the hits following the insertion procedure.



R-Tree: Quadratic split

1. Find and separate the farthest hits $\rightarrow \theta(n^2)$
2. Distribute the rest of the hits following the insertion procedure.