(Nearest) Neighbors You Can Rely On

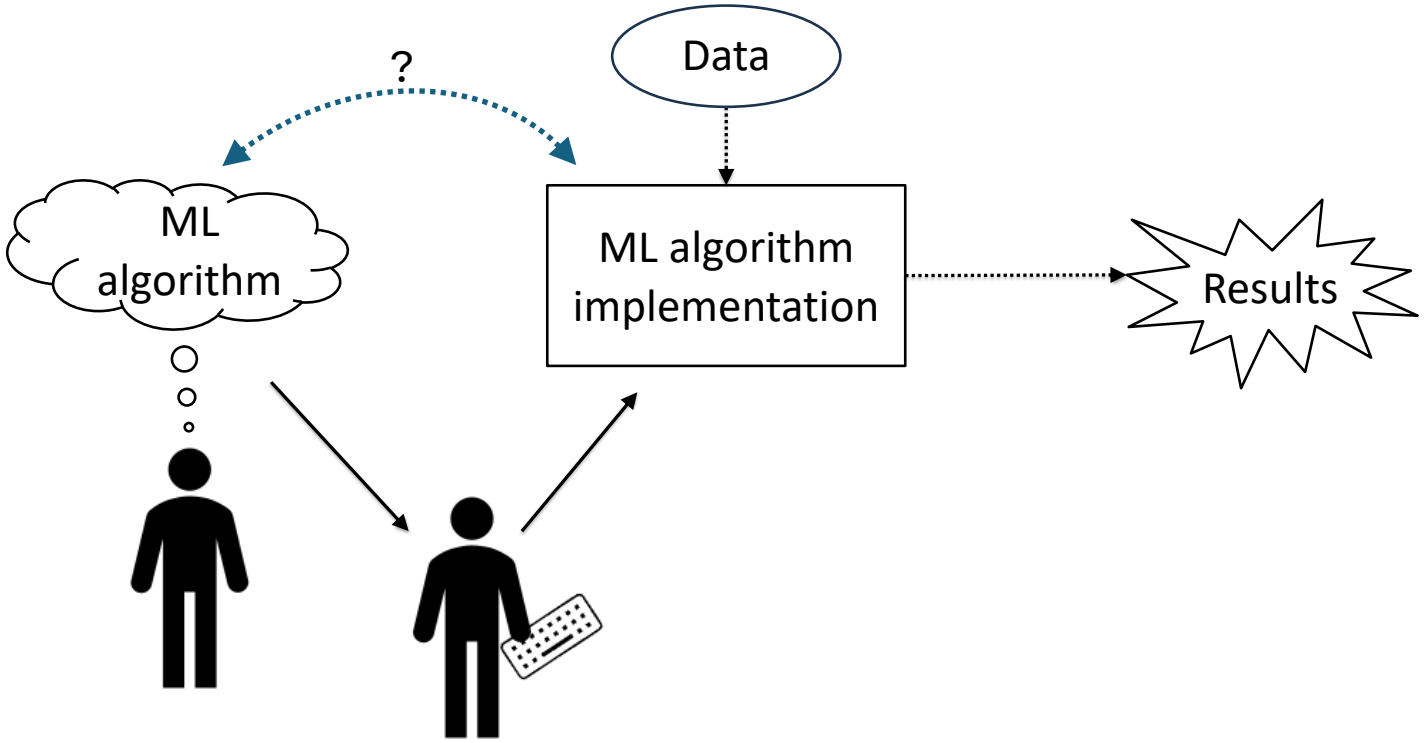
Formally Verified k-d Tree Construction and Search in Coq

Nadeem Abdul Hamid
Berry College, Georgia, USA

SAC-SVT 2024, Ávila, Spain

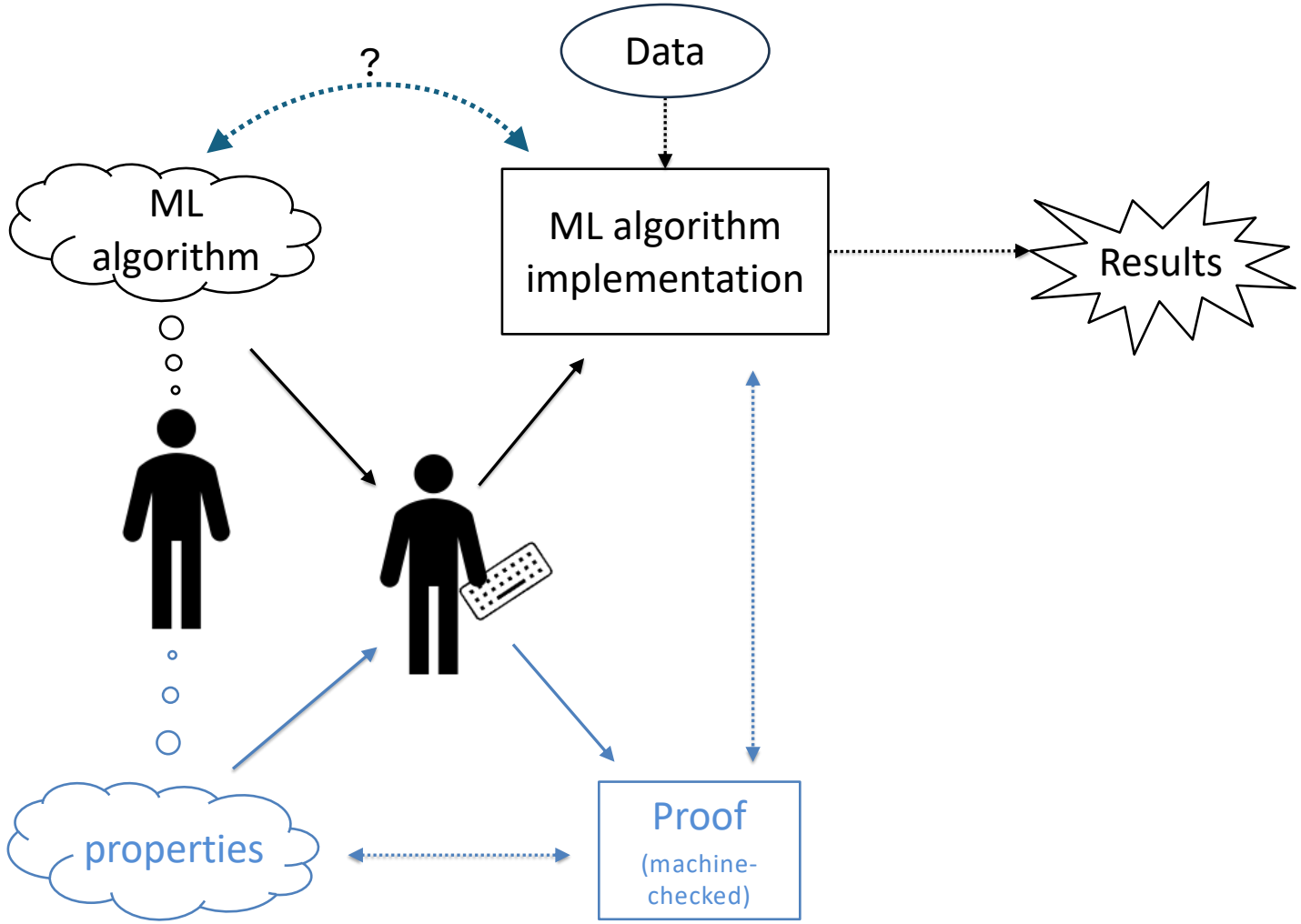
Implementing Machine Learning Algorithms

- Gap between the mathematical model and mechanics of implementation



Implementing Machine Learning Algorithms

- Gap between the mathematical model and mechanics of implementation
- (Big Picture) Context for this work: *Development of verified implementations of ML systems*



Focus: KNN (*K-nearest-neighbors*) Search

- Does the program code for an ML algorithm faithfully implement the mathematical description?
- Focus on the mechanics of the algorithm, not meta-theoretical properties
  - That an implementation correctly finds the closest neighbors to a query
  - Not that those closest neighbors have some statistical properties (Future work)

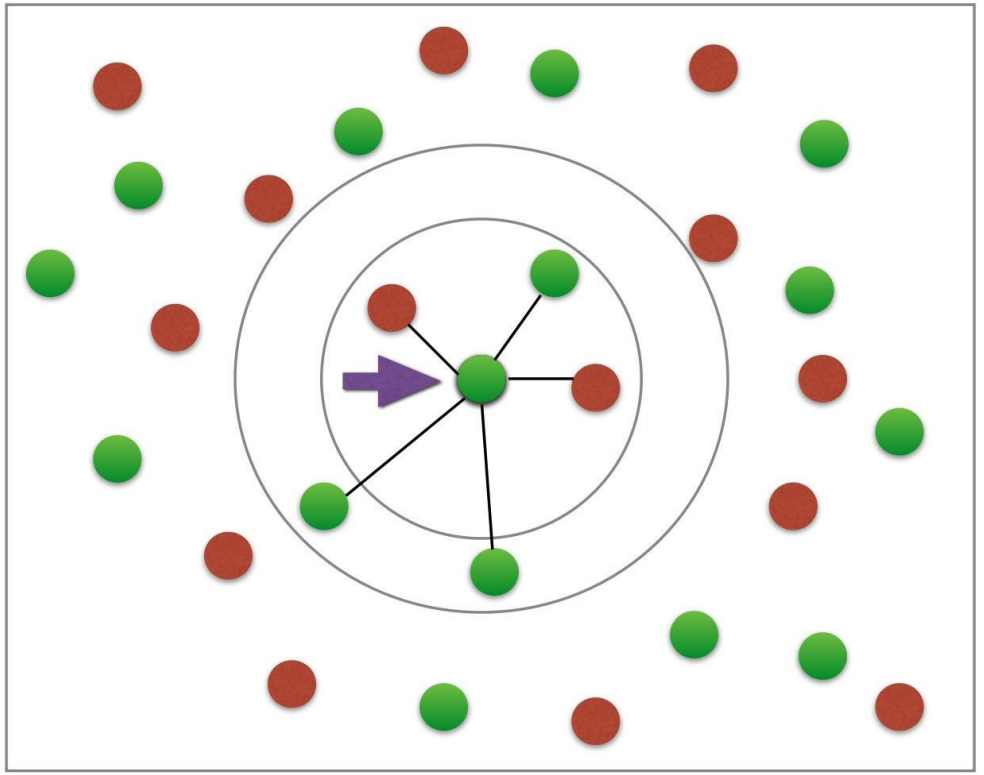


Image: datacamp.com

KNN Search

- One of the oldest, well-known, widely used classification algorithms
  - Assigns class labels to observations based on previously seen data
  - Can also be used for regression
- Applied in a wide variety of domains (not just ML)
- Popularity can be attributed to its simplicity, ease of implementation, and high accuracy rates
- Although, there are known limitations of KNN search
  - (curse of dimensionality; scaling to large data sets)

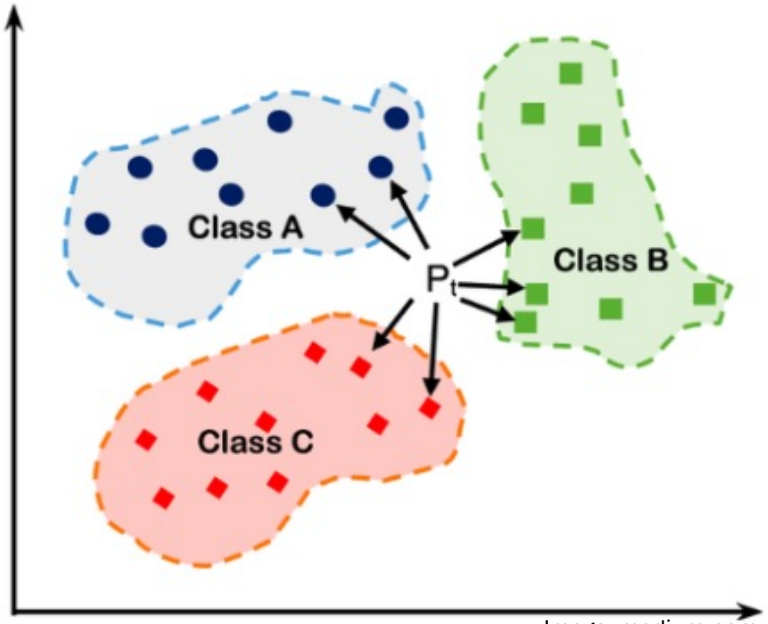


Image: medium.com

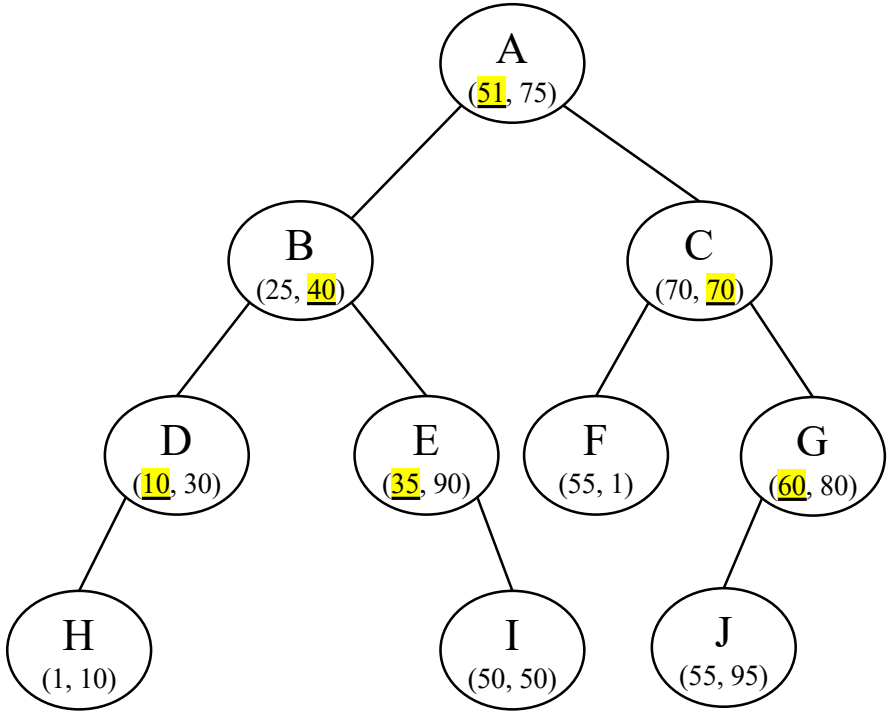
Our Results

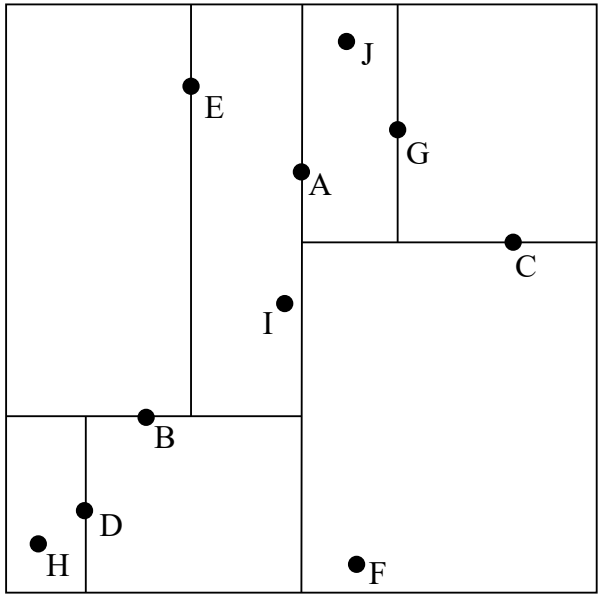
Formally verified (machine-checkable) implementation of a **KNN search algorithm** in the **Coq proof assistant**

- Implementing/Integrating previously-verified data structures
  - ***k*-d trees** (new)
  - bounded priority queue (adapted)
- And algorithms
  - **Quick-select** median finding
  - Generalized ***K*-nearest neighbors search**

k-d trees

- Binary tree
- Nodes: *k*-dimensional data points
- Each level partitioned based on one of *k* dimensions
- Each subtree associated with an (implicit) bounding box
- Enables sub-linear NN search complexity through branch-and-bound






```
I: (50, 50)
A: (51, 75)
B: (25, 40)
C: (70, 70)
J: (55, 95)
H: (1, 10)
G: (60, 80)
F: (55, 1)
E: (35, 90)
D: (10, 30)
```

Lowercase ***k*** = dimension of data points;
Uppercase ***K*** = number of neighbors

Bounding boxes

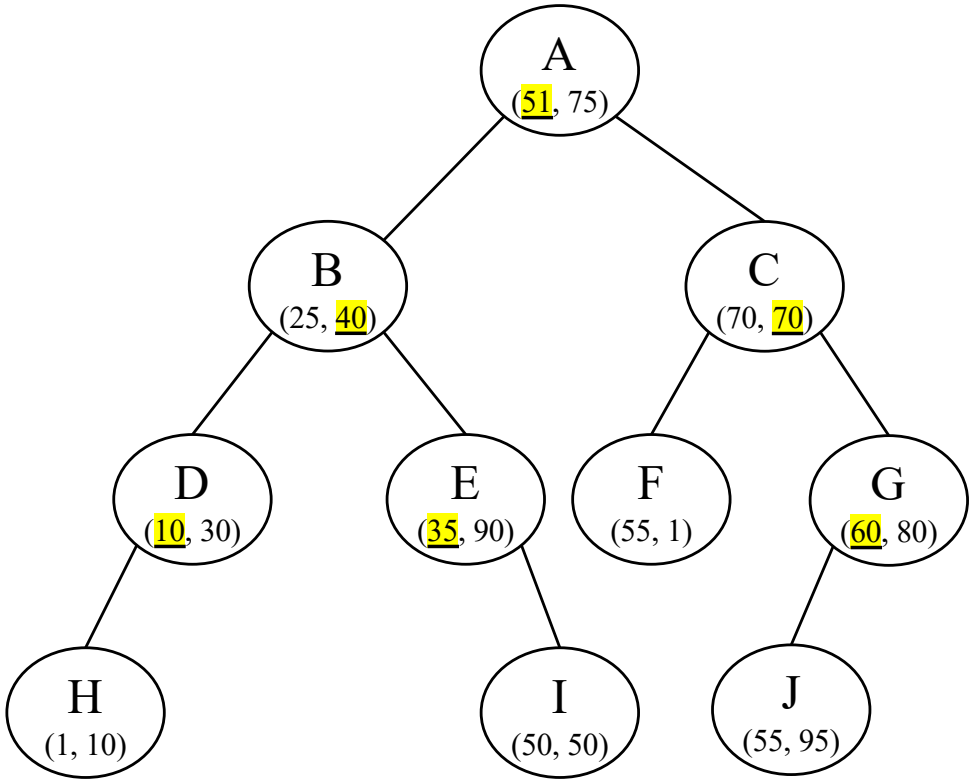
- Implicit in implementation; Crucial to correctness

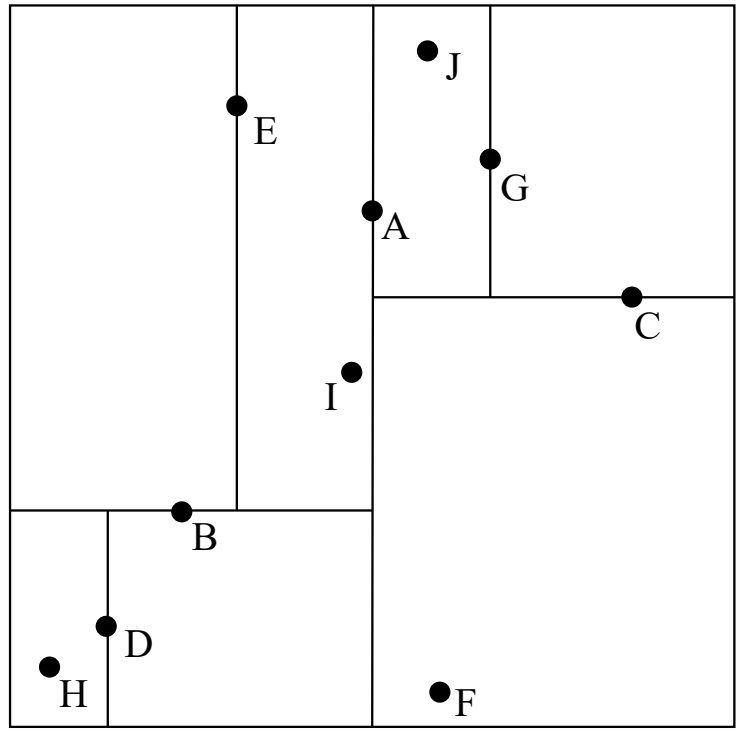
$[(x_{min}, y_{min}), (x_{max}, y_{max})]$

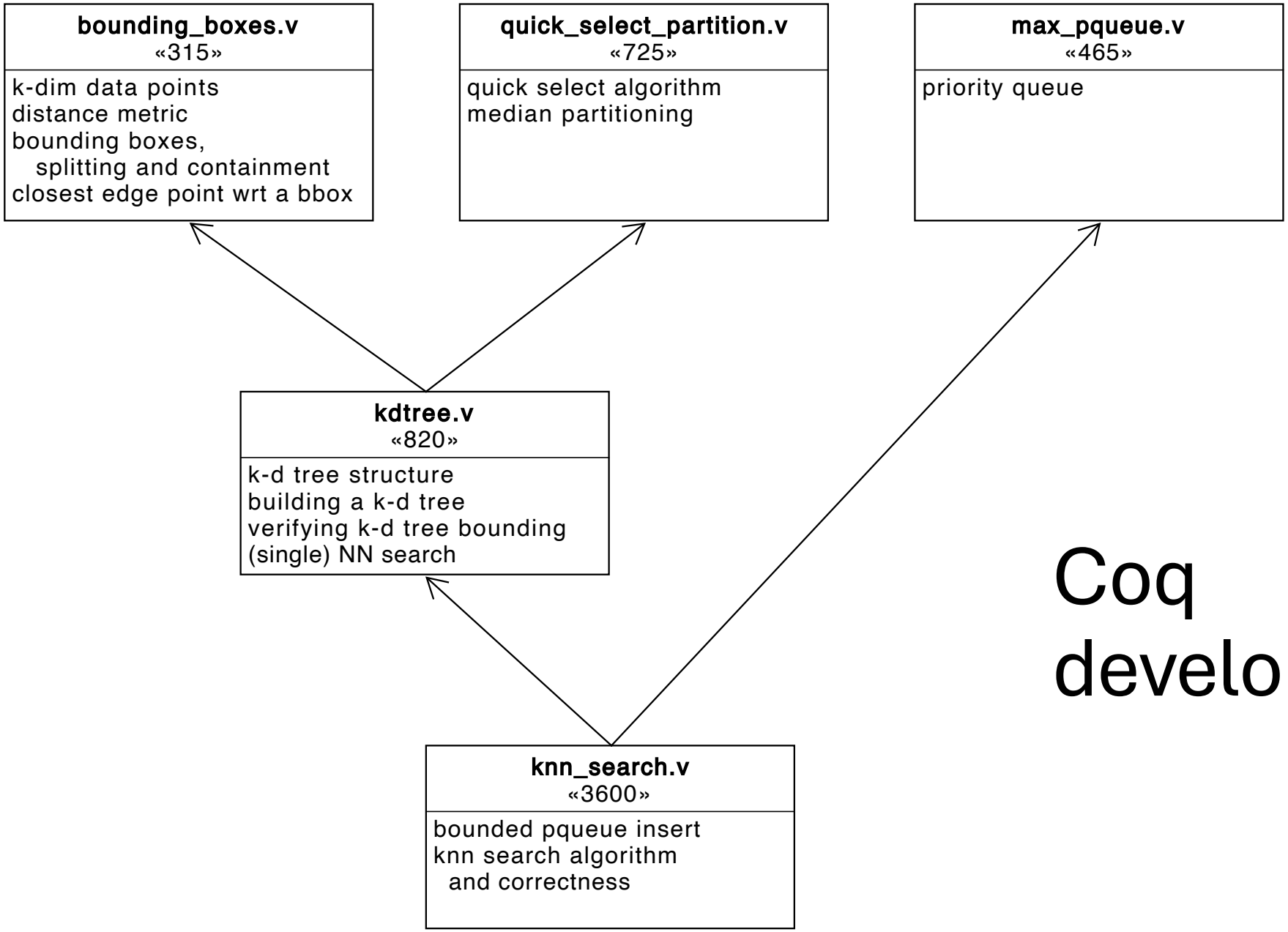
A - $[(-\infty, -\infty), (+\infty, +\infty)]$

…

I - $[(35, 40), (51, +\infty)]$





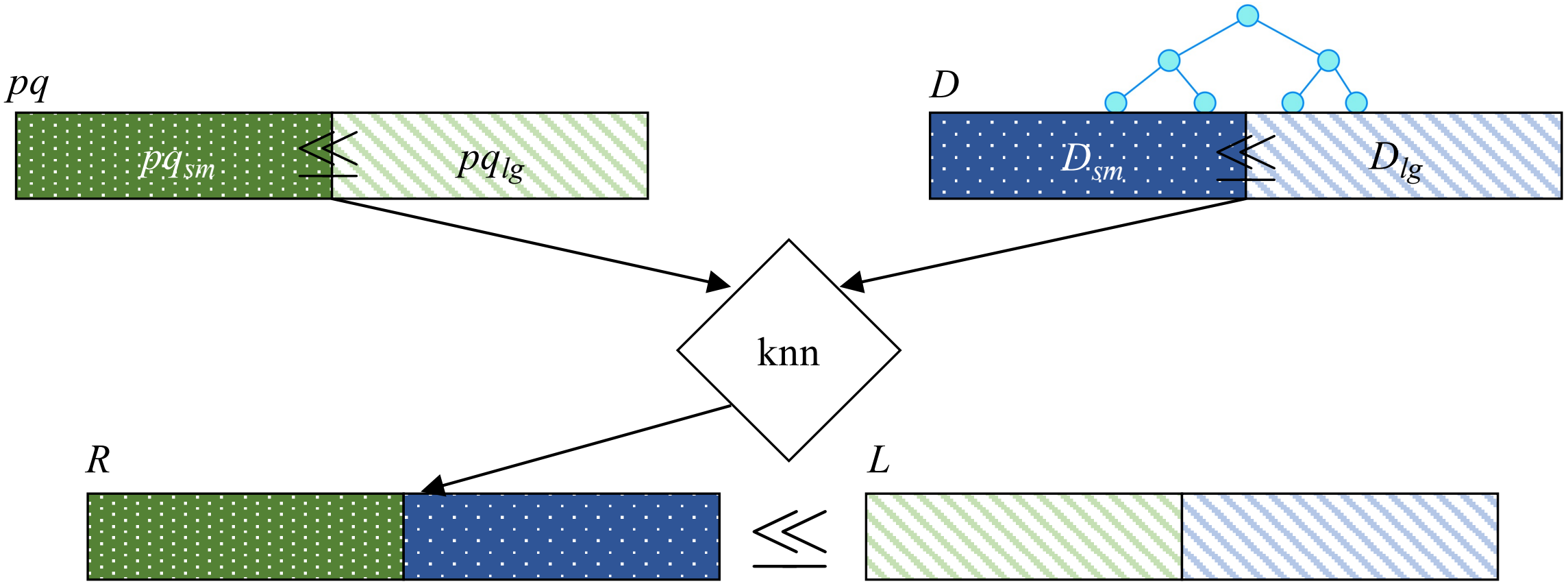


Coq developments

Final Theorem

```
Theorem knn_search_build_kdtree_correct :
 forall (K:nat) (k : nat) (data : list datapt),   // Preconditions:
   0 < K ->                                       // at least one neighbor sought
   0 < length data ->                             // data is non-empty
   0 < k ->                                       // dimension space is non-empty
   (forall v' : datapt,                           // all data points well-formed (k-dim)
          In v' data -> length v' = k) ->
   forall tree query result,
      tree = (build_kdtree k data) ->             // If: the k-d tree built from data
      knn_search K k tree query = result ->       // produces result for a query point,
      exists leftover,                            // Then:
         length result = min K (length data)      // the result is length (at most) K,
         /\ Permutation data (result ++ leftover) // and is a sub-list of data,
         /\ all_in_leb (sum_dist query) result leftover.
                                                  // and everything in
                  // result is closer in distance to the query than all the leftover part of data.
```

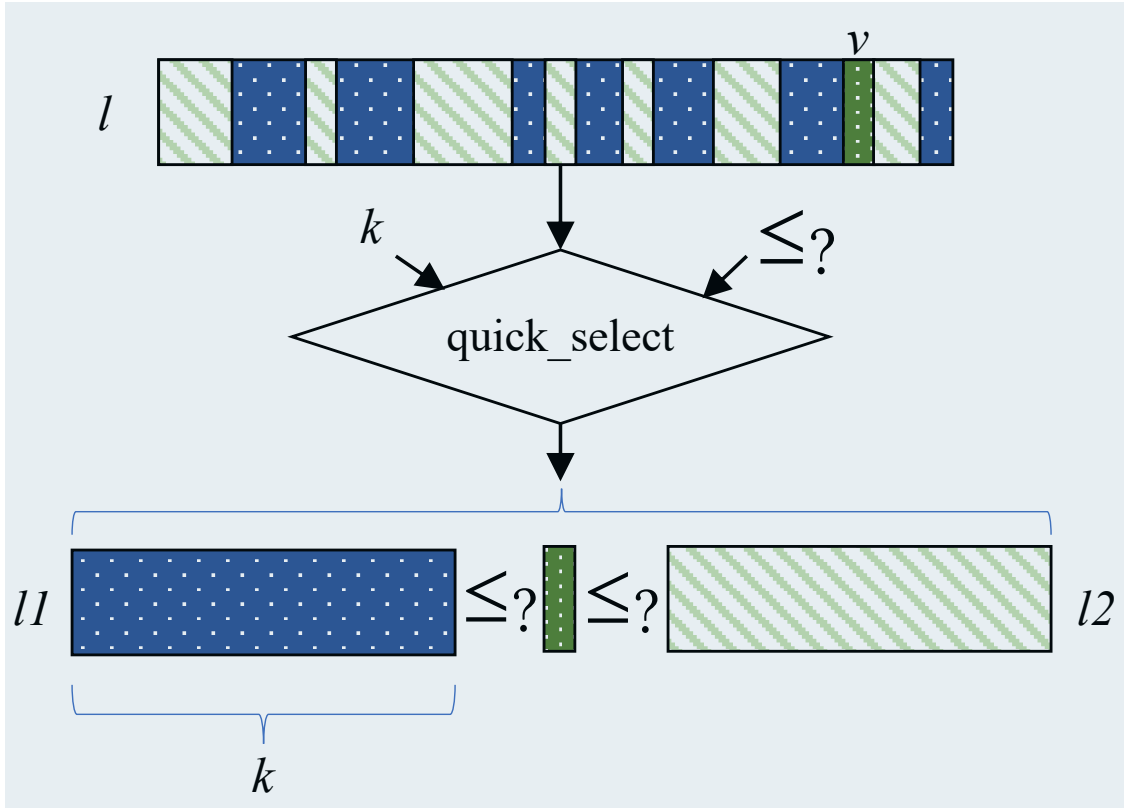
Partitions Induced by the *knn* Function



Quickselect

- Used to build the initial *k*-d tree

```
Theorem quick_select_exists_correct :
 forall (X:Set) (k:nat) (l:list X)
        (le:X -> X -> bool),
   le_props le ->
   k < length l ->
   exists l1 v l2,
     quick_select k l le = Some (l1,v,l2) /\
     Permutation l (l1 ++ v :: l2) /\
     length l1 = k /\
     (forall x, In x l1 -> le x v = true) /\
     (forall x, In x l2 -> le v x = true).
```



Future Work

- Automate permutations reasoning
- Abstract the distance metric
- *Implement, specify, & verify a full classification system*
- Port to verified C implementation
- Extend to modern variants of KNN (e.g. approximation, etc.)

Acknowledgements

Matthew Bowker

Jessica Herring

Bernny Velasquez

Thank you!

Questions?

nadeem@acm.org