k-Nearest Neighbors Classification using kd-Trees

Design of Parallel and High Performance Computing

Fabian Wermelinger
Johannes de Fine Licht
Prabhakaran Santhanam

November 2, 2015

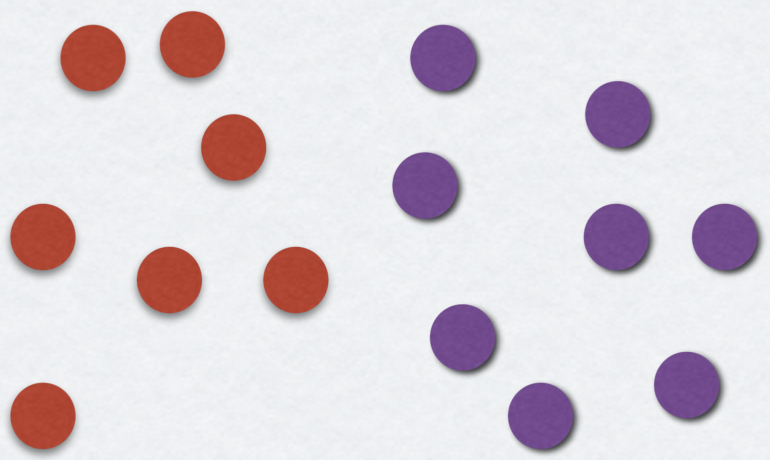
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Introduction

- Algorithm - k-Nearest Neighbours (kNN)
  - Classification of a given Test Point - Output the most occurring class of the k nearest neighbours

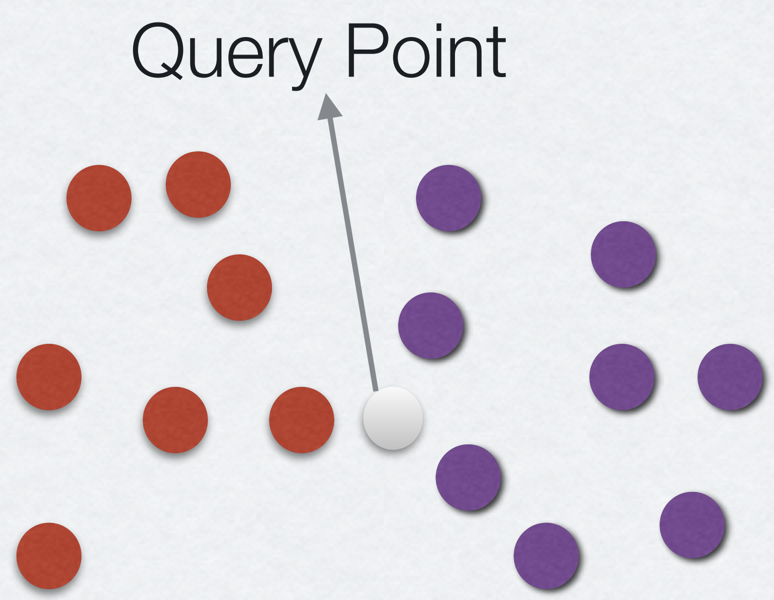
Introduction

- Algorithm - k-Nearest Neighbours (kNN)
  - Classification of a given Test Point - Output the most occurring class of the k nearest neighbours



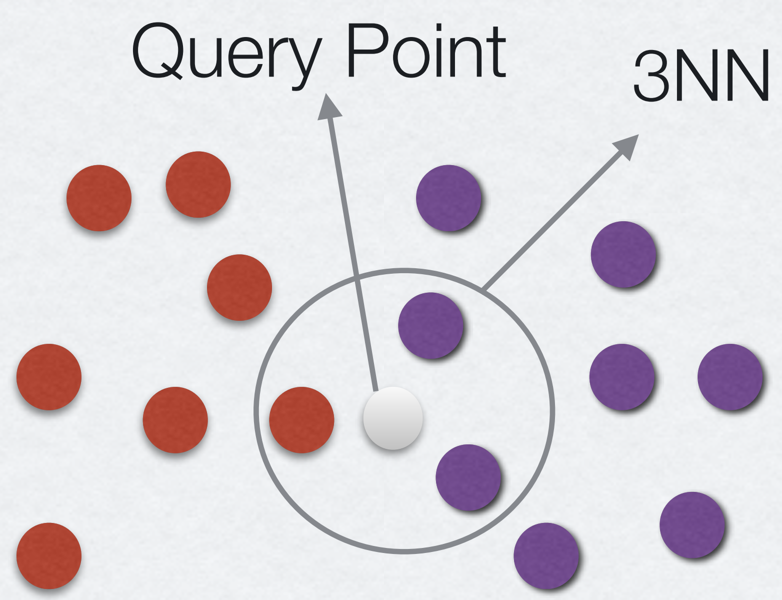
Introduction

- Algorithm - k-Nearest Neighbours (kNN)
  - Classification of a given Test Point - Output the most occurring class of the k nearest neighbours



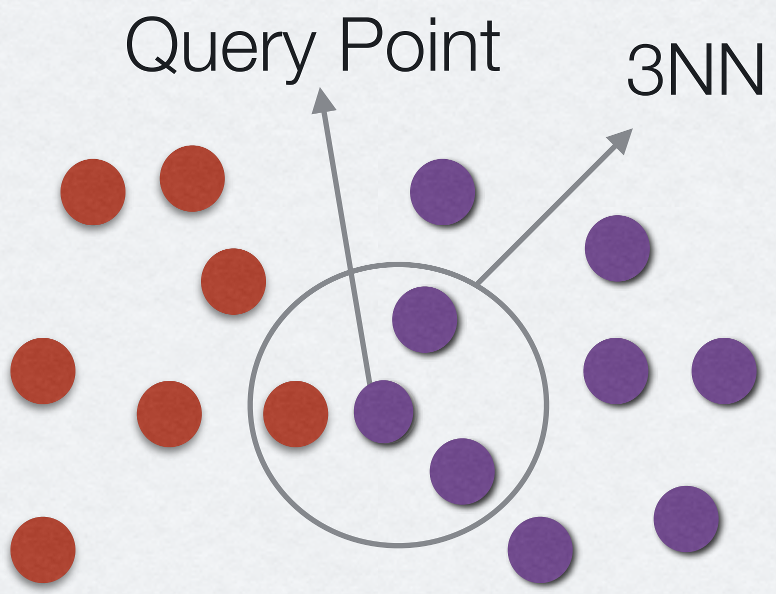
Introduction

- Algorithm - k-Nearest Neighbours (kNN)
  - Classification of a given Test Point - Output the most occurring class of the k nearest neighbours



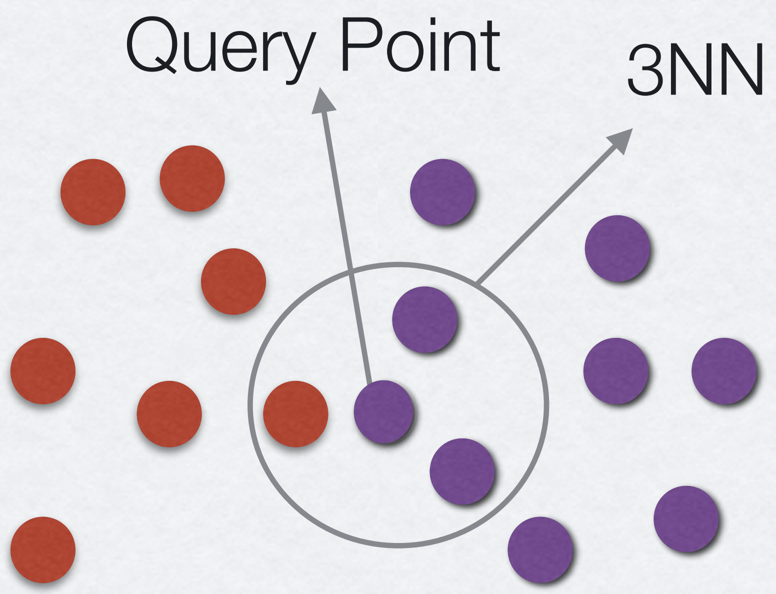
Introduction

- Algorithm - k-Nearest Neighbours (kNN)
  - Classification of a given Test Point - Output the most occurring class of the k nearest neighbours



Introduction

- Algorithm - k-Nearest Neighbours (kNN)
  - Classification of a given Test Point - Output the most occurring class of the k nearest neighbours



- kd-tree Implementation of kNN is used
  - kd-tree can do look up in O(log n), associated with additional overhead due to building the tree.

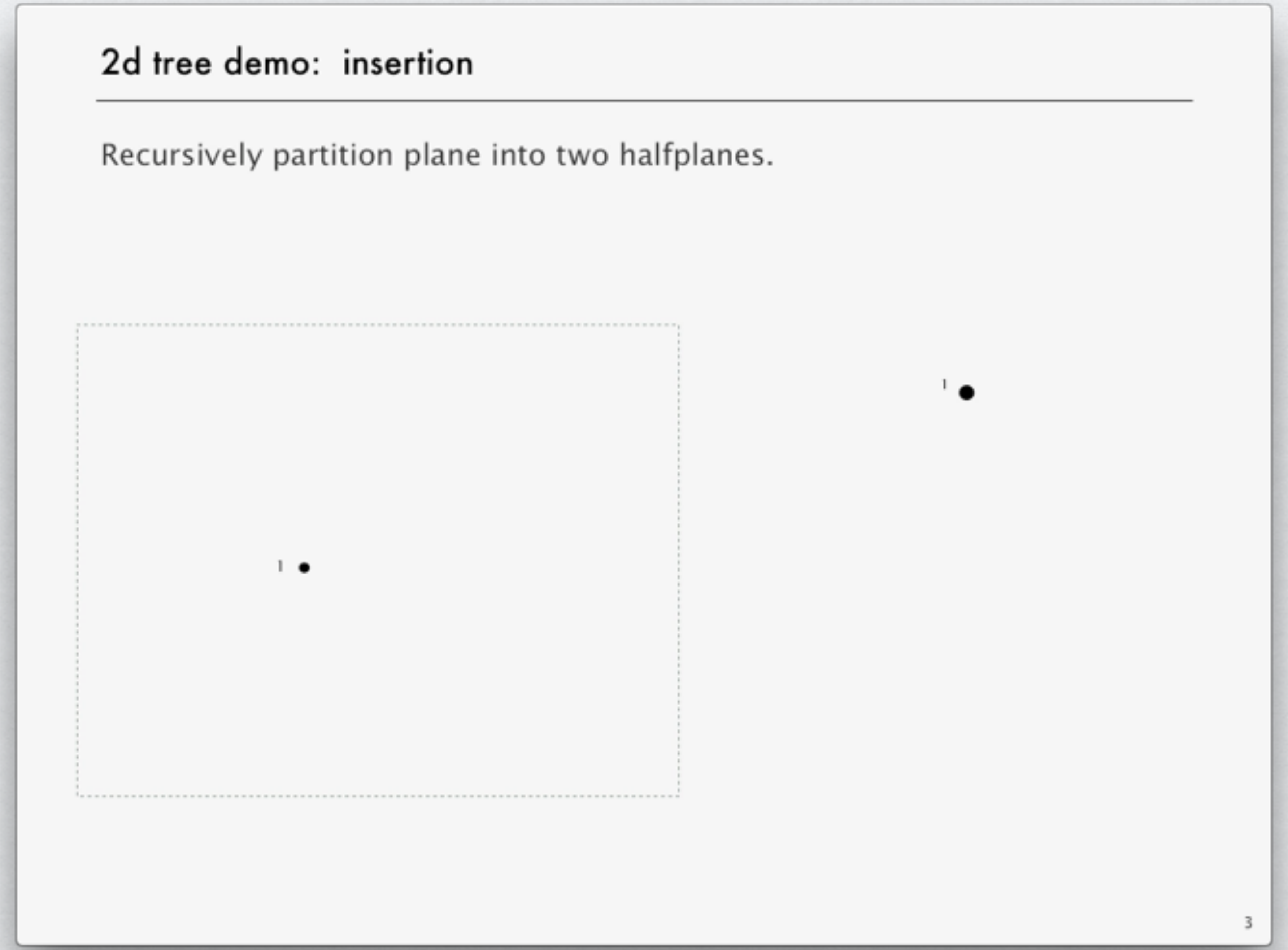
kd-Tree Construction

kd-Tree Construction

- For each level of the tree, a dimension is chosen for splitting the data points.
- For each dimension an average of value of that dimension for all data points are taken and that point is used to split the tree at that level. This process is repeated recursively.

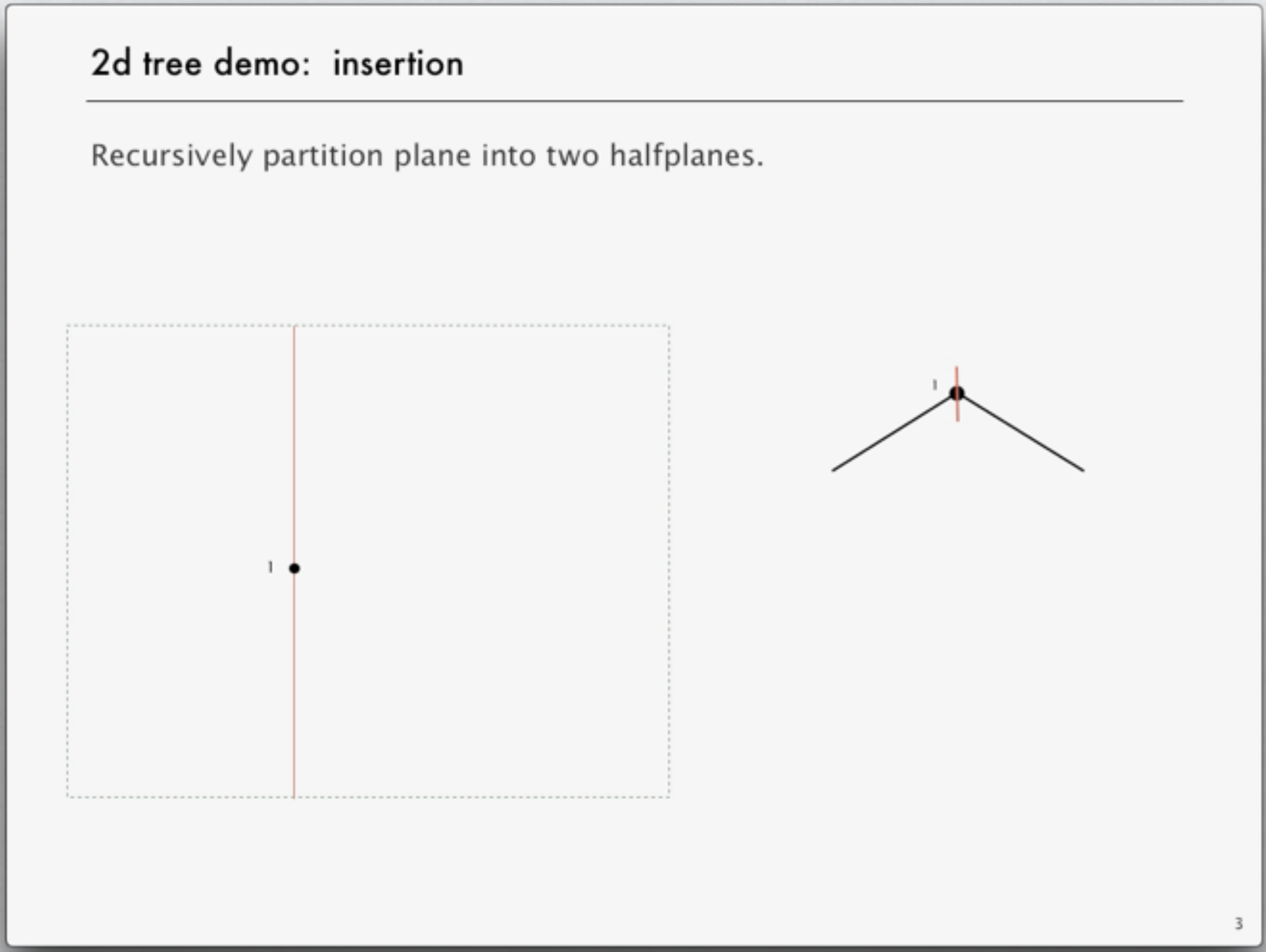
kd-Tree Construction

- For each level of the tree, a dimension is chosen for splitting the data points.
- For each dimension an average of value of that dimension for all data points are taken and that point is used to split the tree at that level. This process is repeated recursively.



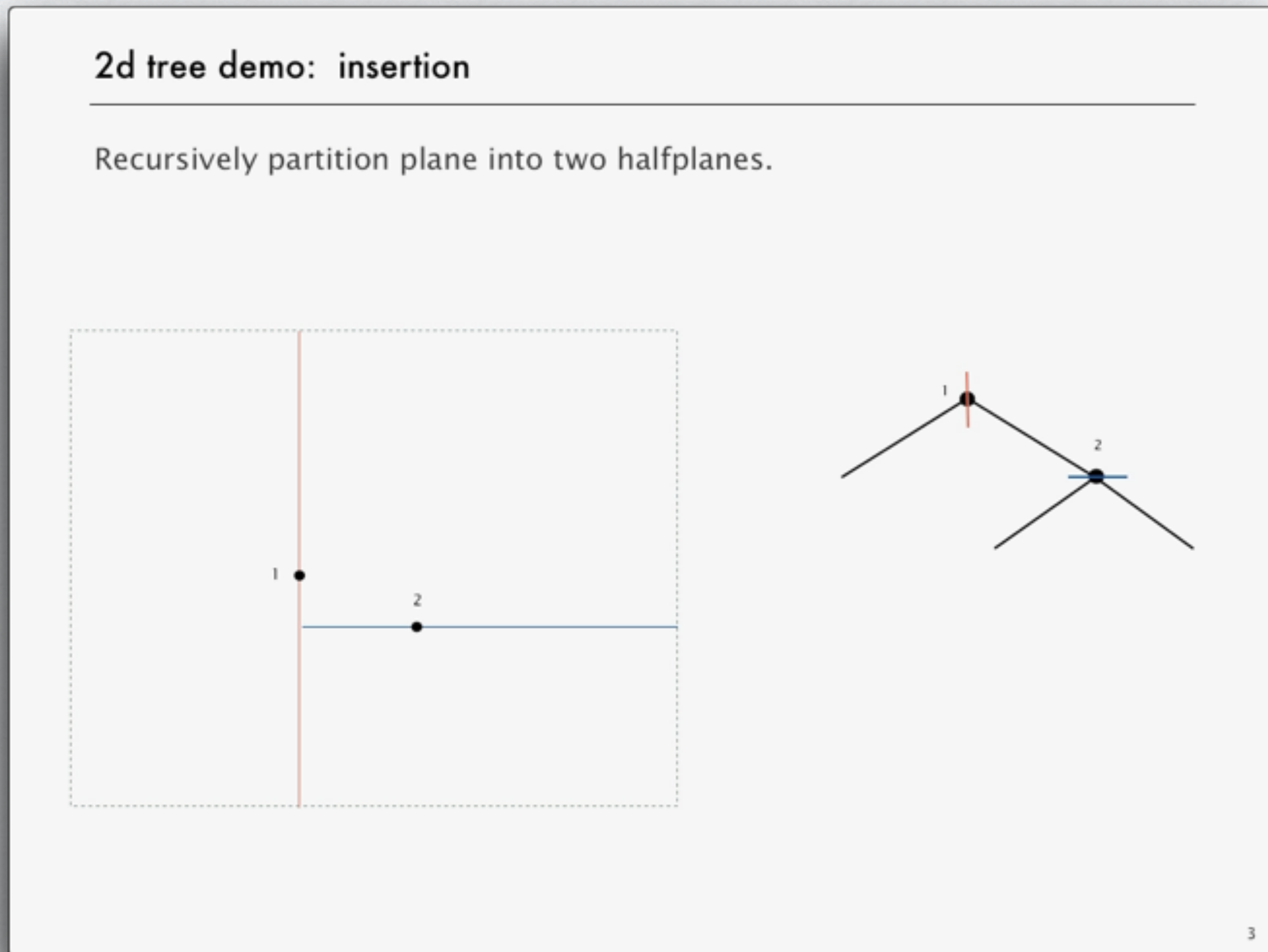
kd-Tree Construction

- For each level of the tree, a dimension is chosen for splitting the data points.
- For each dimension an average of value of that dimension for all data points are taken and that point is used to split the tree at that level. This process is repeated recursively.



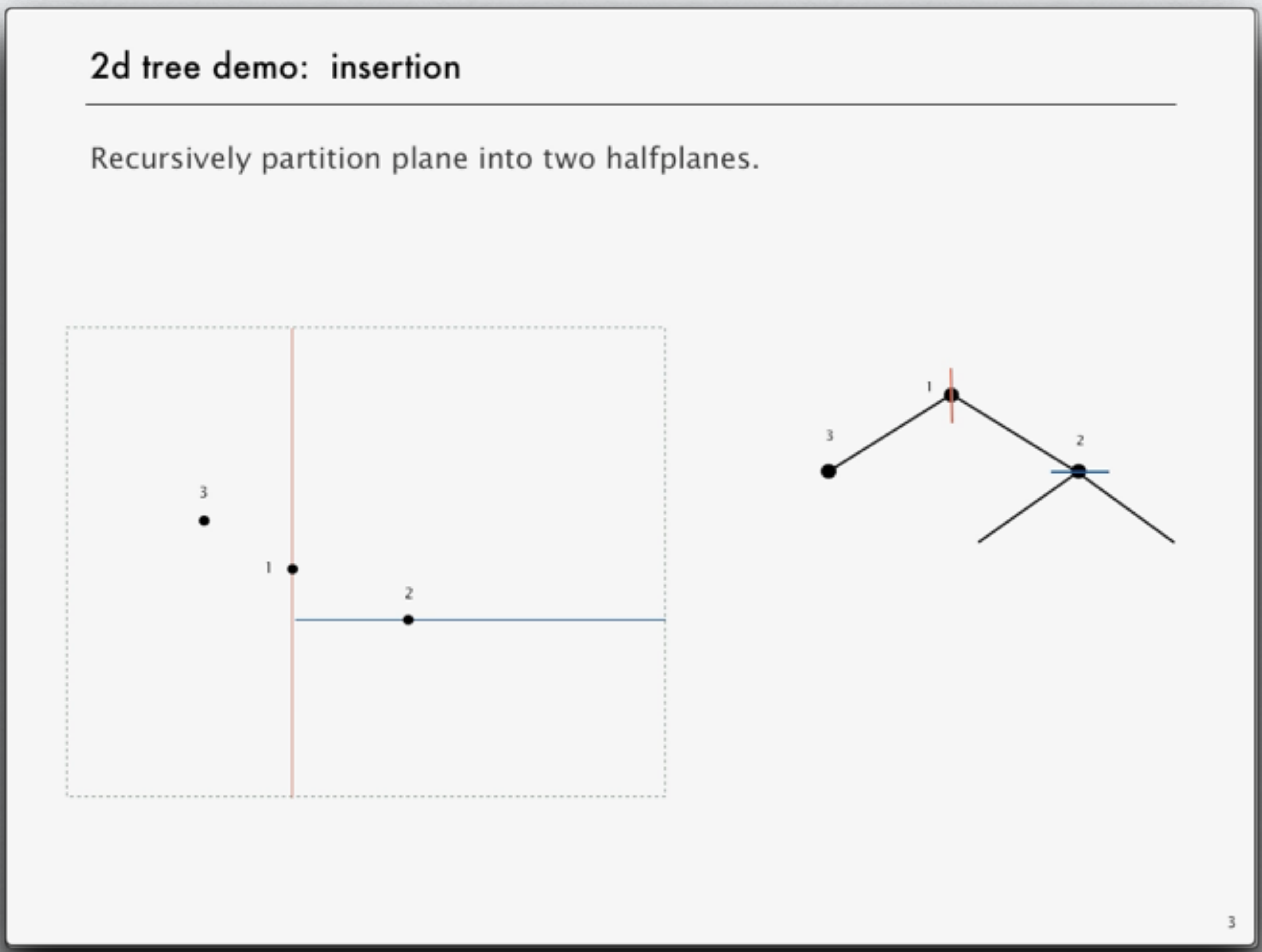
kd-Tree Construction

- For each level of the tree, a dimension is chosen for splitting the data points.
- For each dimension an average of value of that dimension for all data points are taken and that point is used to split the tree at that level. This process is repeated recursively.



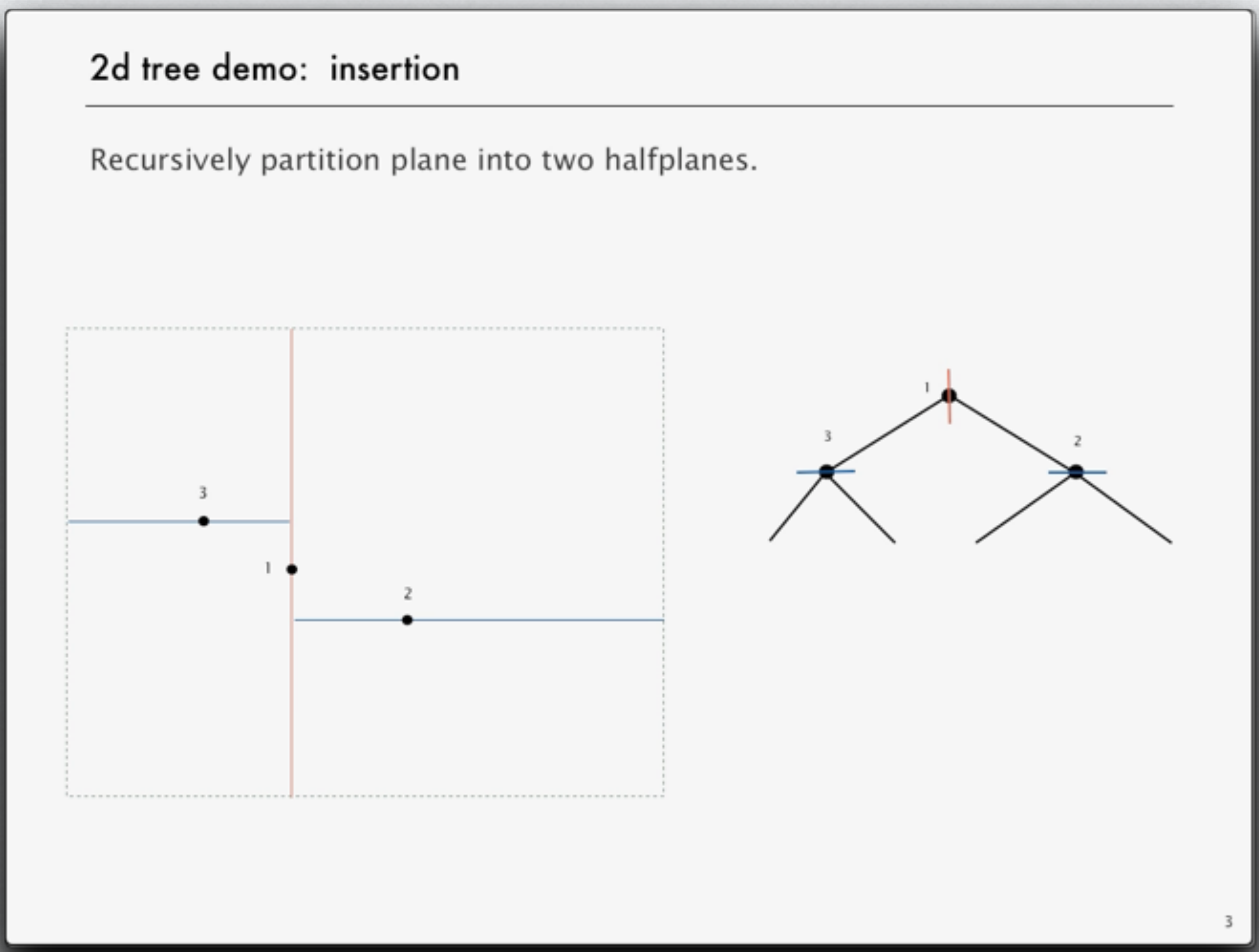
kd-Tree Construction

- For each level of the tree, a dimension is chosen for splitting the data points.
- For each dimension an average of value of that dimension for all data points are taken and that point is used to split the tree at that level. This process is repeated recursively.



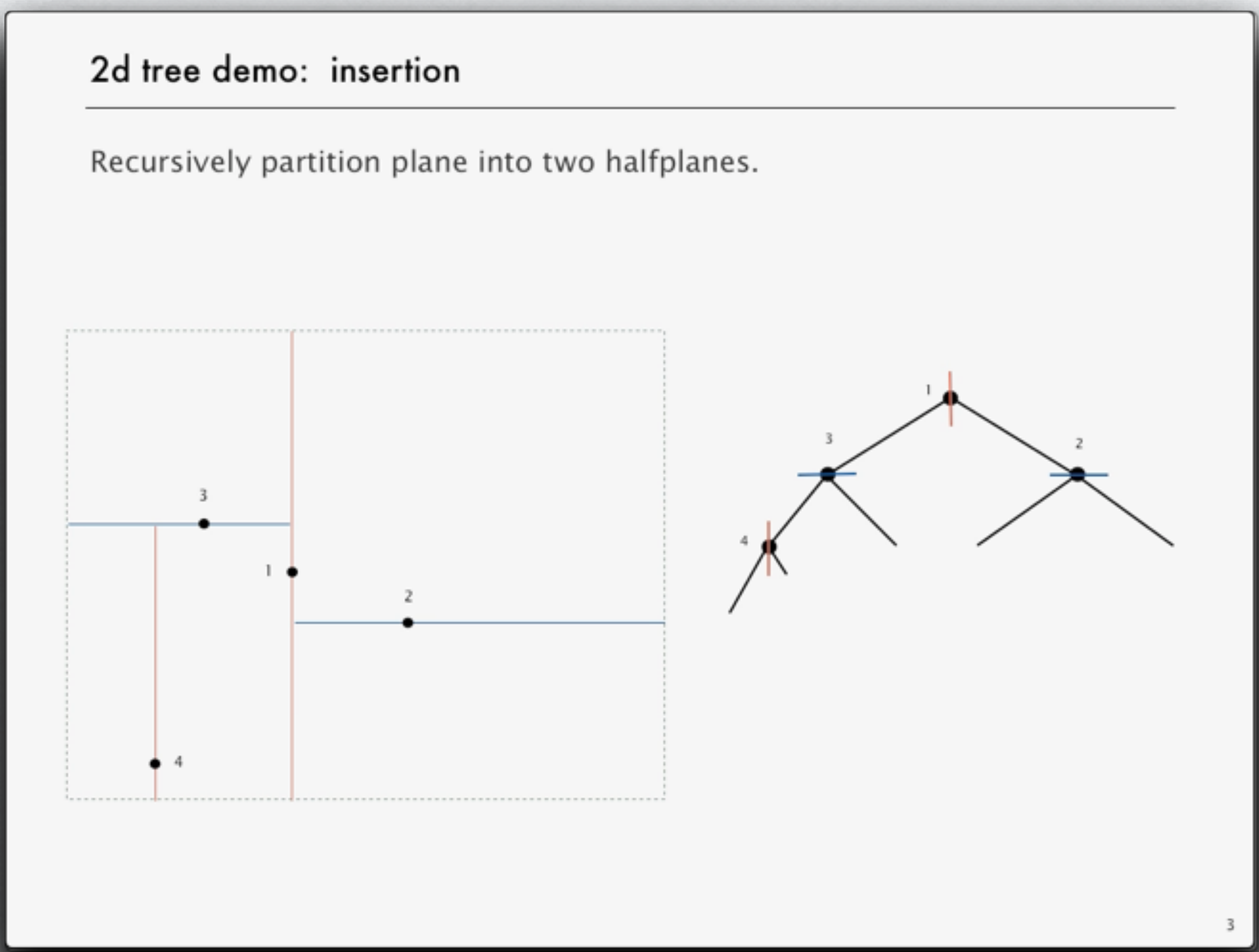
kd-Tree Construction

- For each level of the tree, a dimension is chosen for splitting the data points.
- For each dimension an average of value of that dimension for all data points are taken and that point is used to split the tree at that level. This process is repeated recursively.



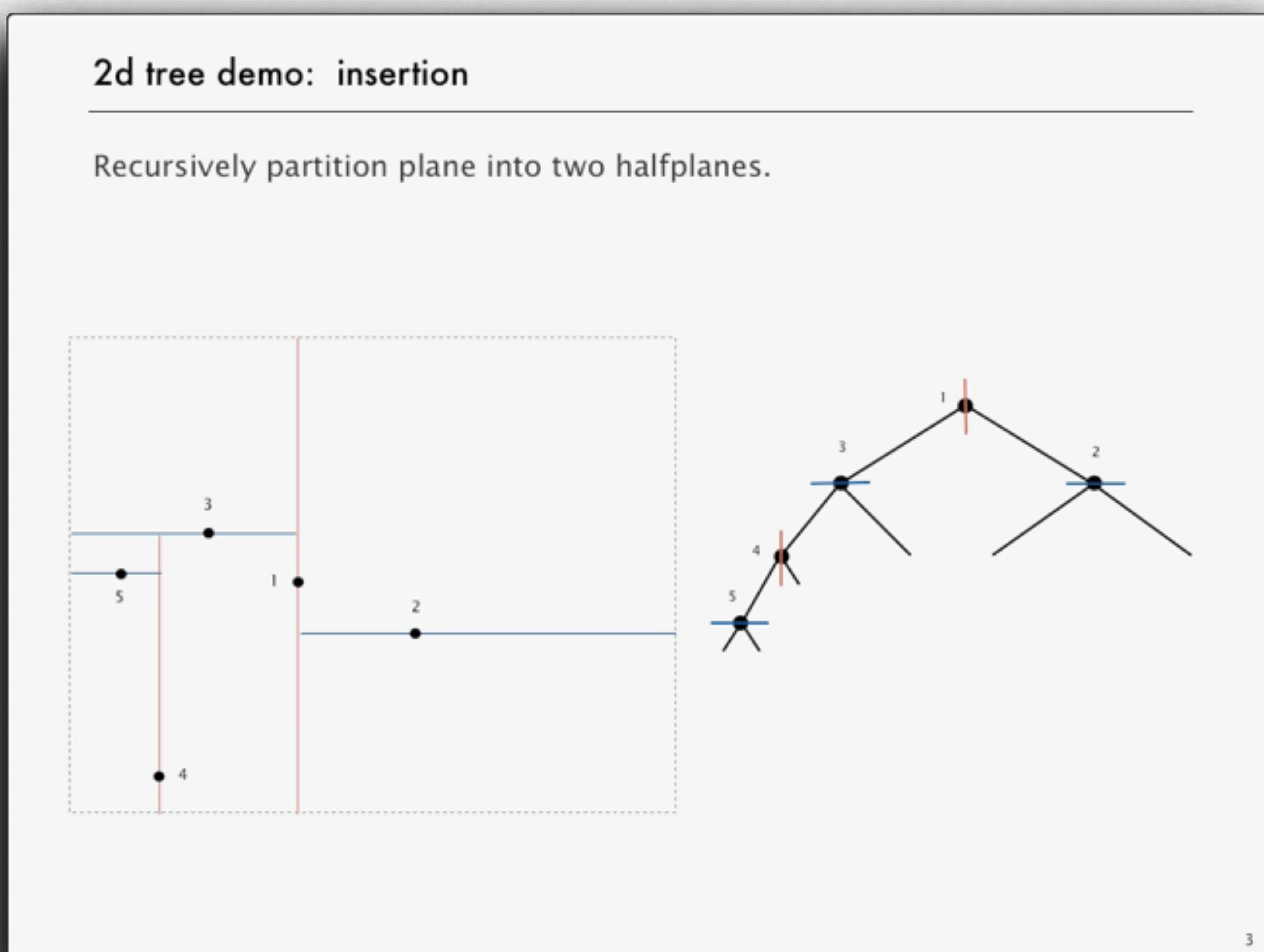
kd-Tree Construction

- For each level of the tree, a dimension is chosen for splitting the data points.
- For each dimension an average of value of that dimension for all data points are taken and that point is used to split the tree at that level. This process is repeated recursively.



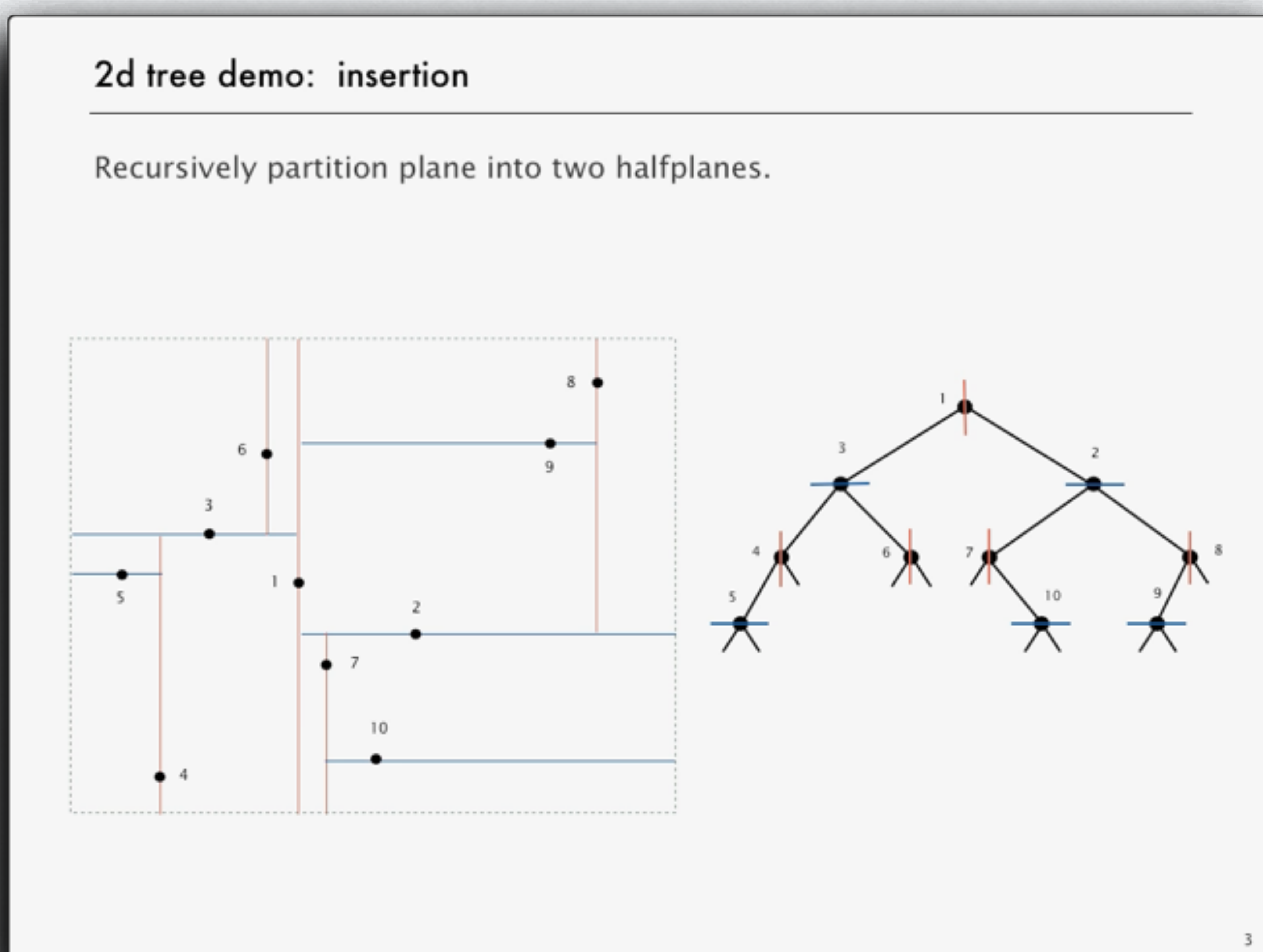
kd-Tree Construction

- For each level of the tree, a dimension is chosen for splitting the data points.
- For each dimension an average of value of that dimension for all data points are taken and that point is used to split the tree at that level. This process is repeated recursively.



kd-Tree Construction

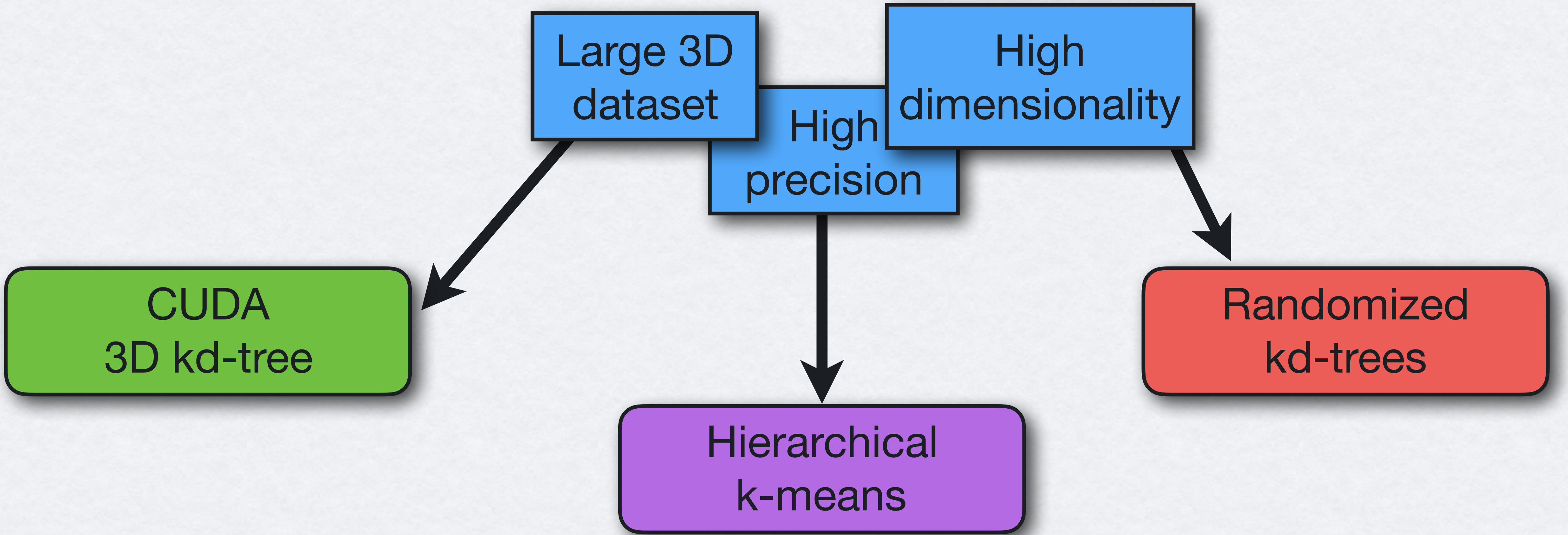
- For each level of the tree, a dimension is chosen for splitting the data points.
- For each dimension an average of value of that dimension for all data points are taken and that point is used to split the tree at that level. This process is repeated recursively.



FLANN

Fast **L**ibrary for **A**pproximate **N**earest **N**eighbor by Marius Muja and David G. Lowe (http://www.cs.ubc.ca/research/flann/).

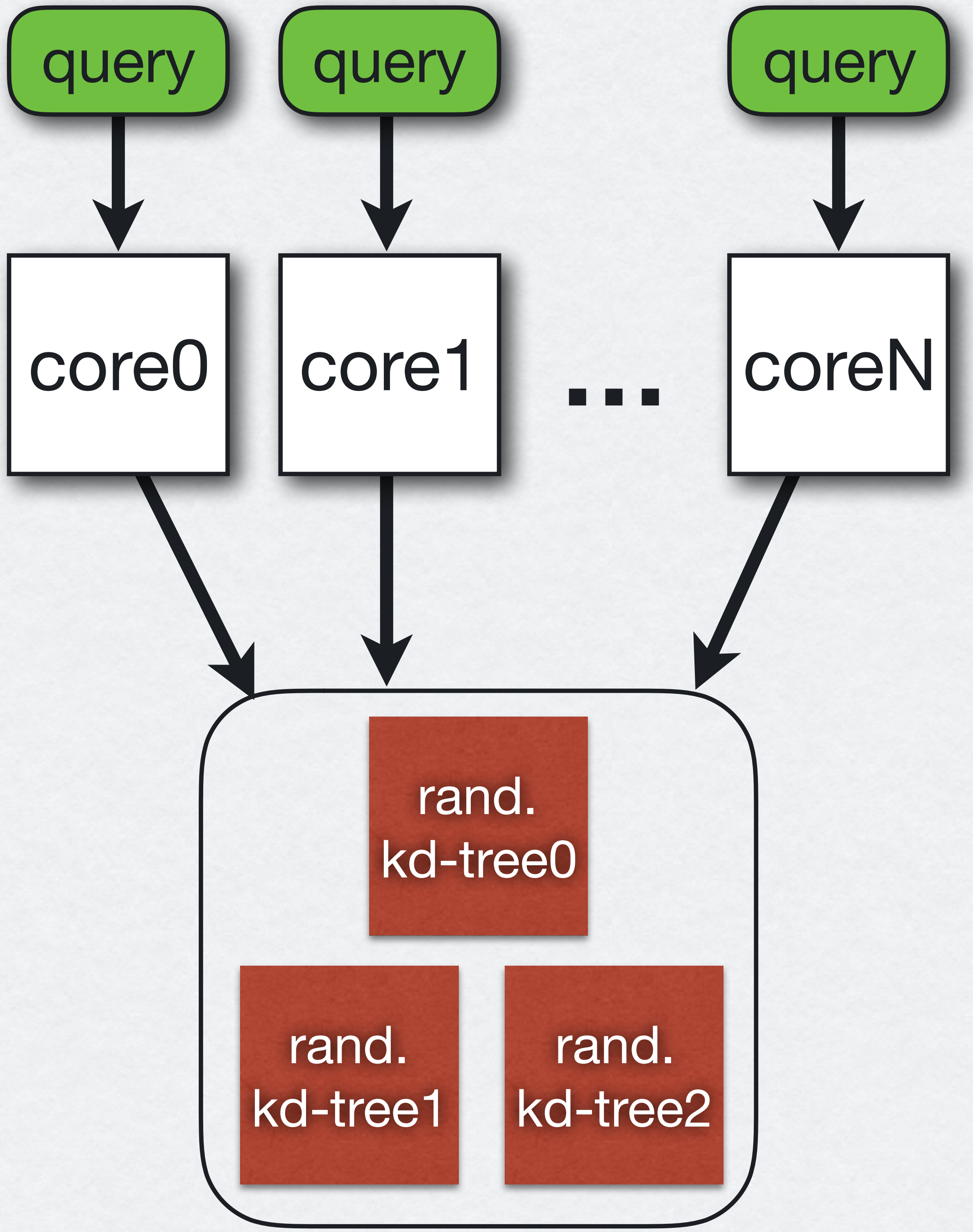
Implements different NN algorithms suitable for different input:



Widely used, in particular in Computer Vision application.

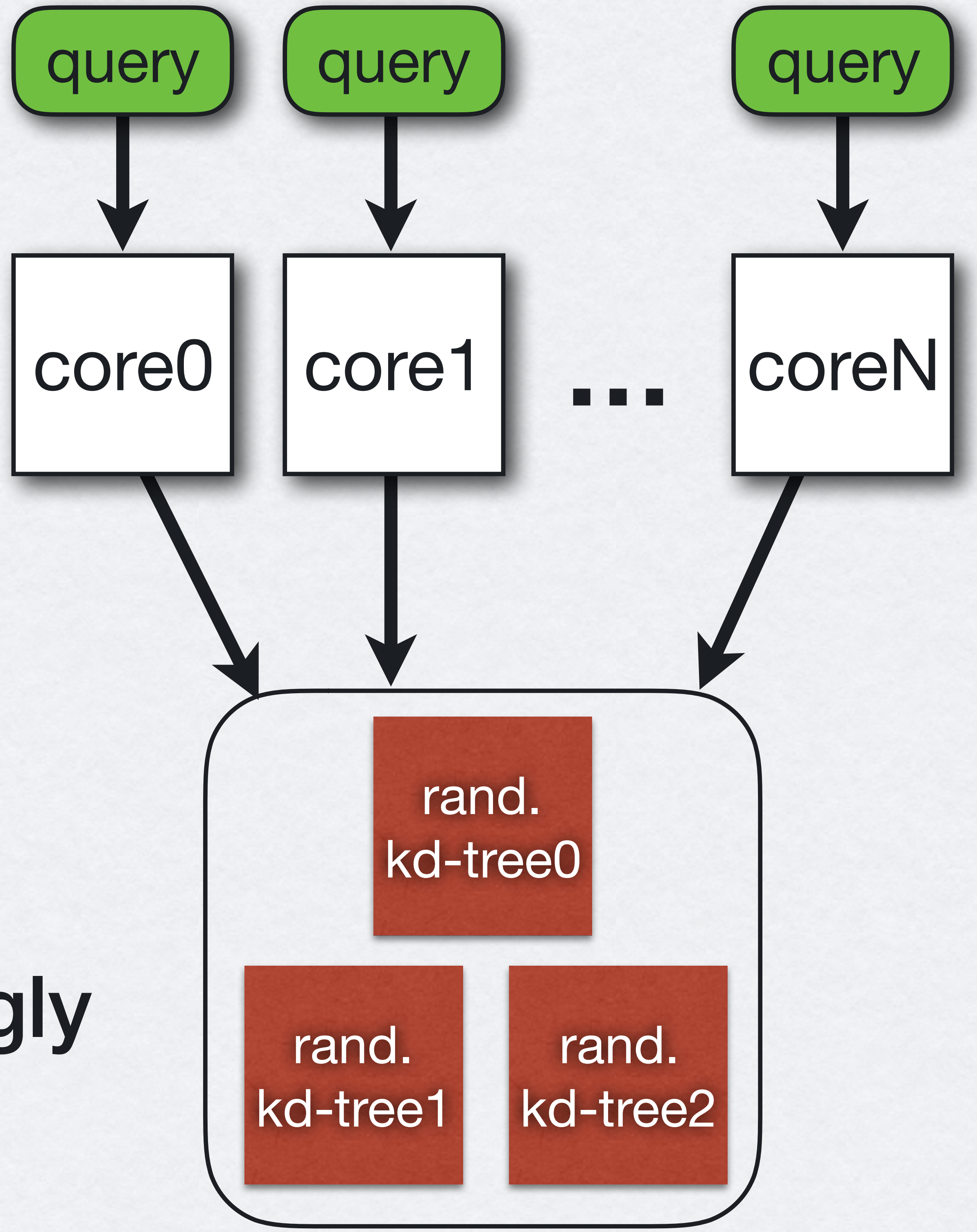
Xeon Phi

All cores query one set of random kd-trees



Xeon Phi

All cores query one set of random kd-trees

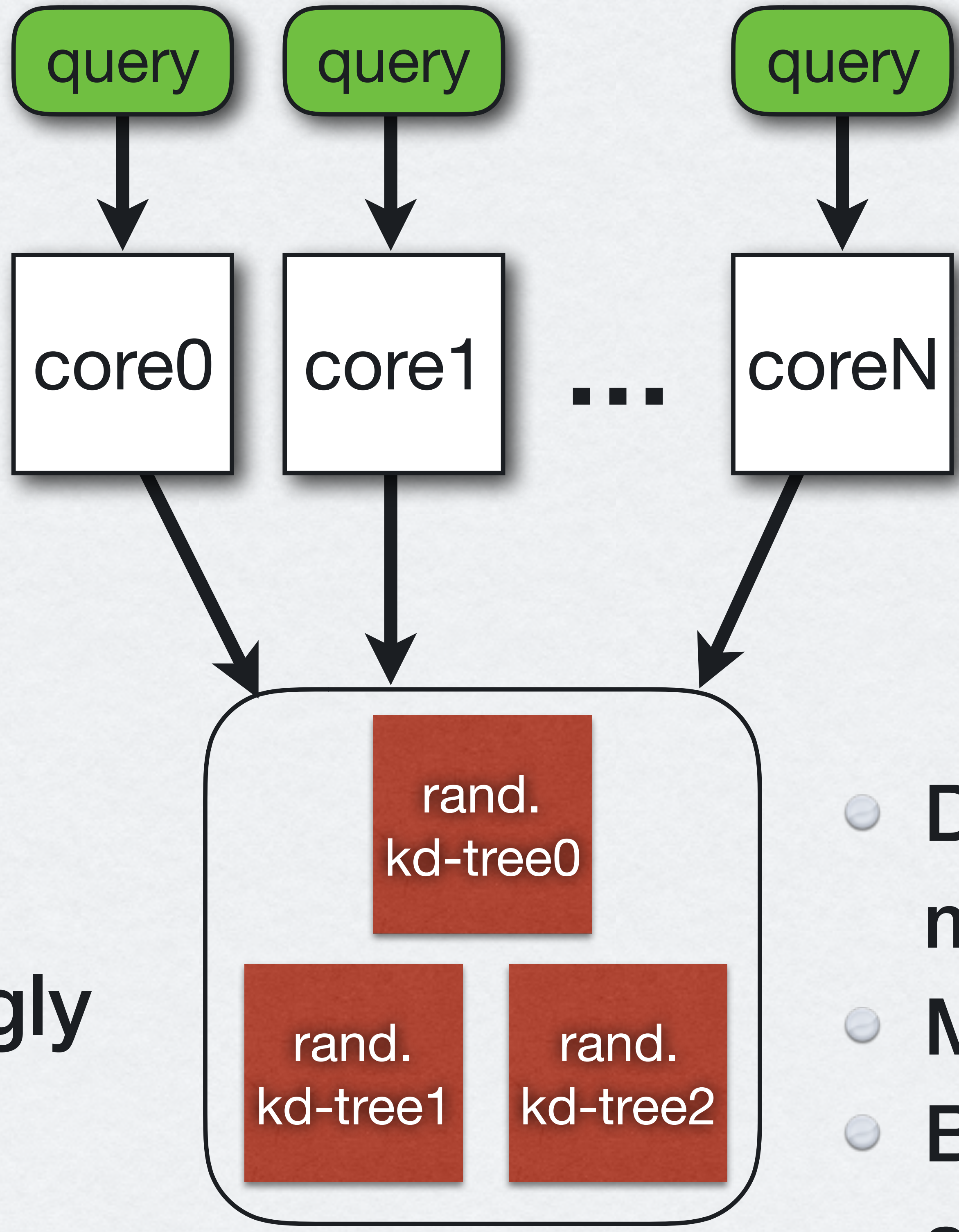


Pros

- O(log n) complexity
- Embarrassingly parallel

Xeon Phi

All cores query one set of random kd-trees



Pros

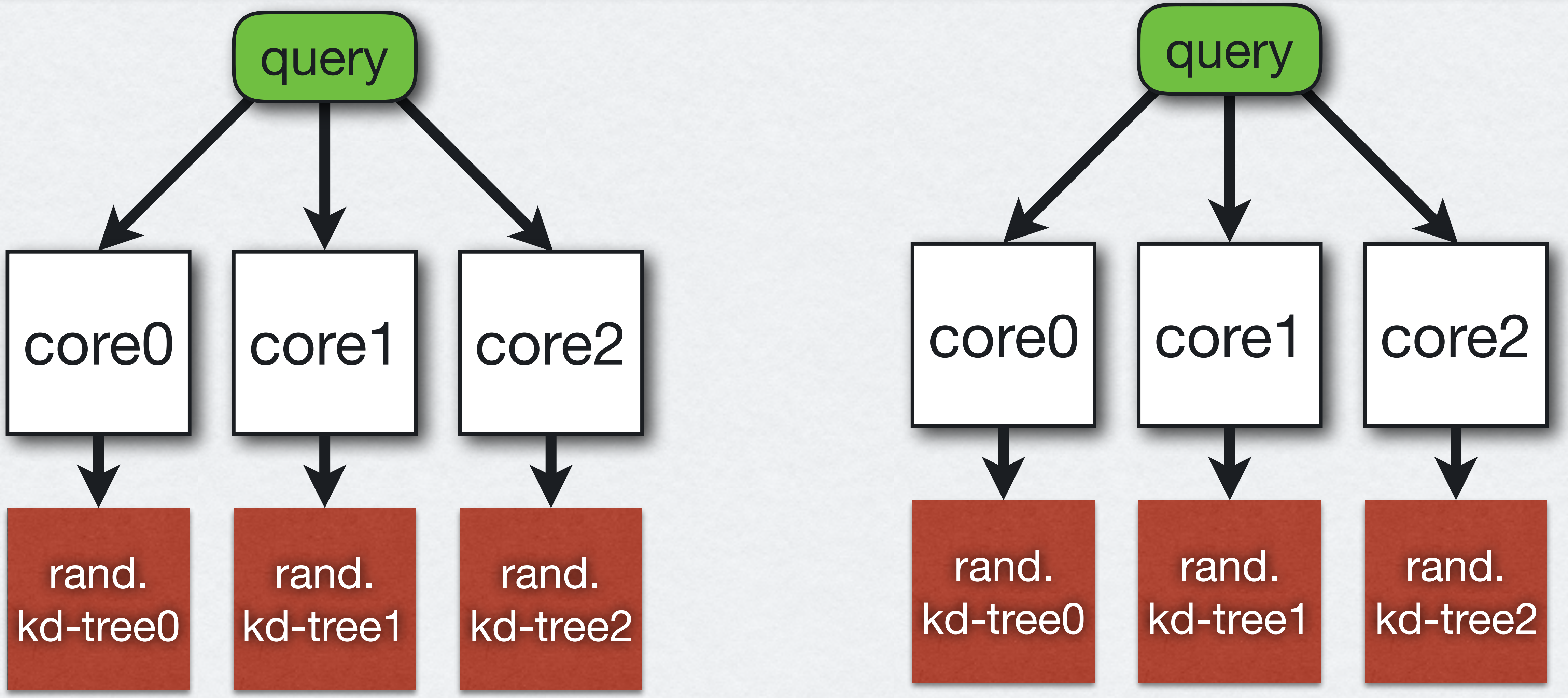
- O(log n) complexity
- Embarrassingly parallel

Cons

- Discontiguous memory access
- Memory bound
- Best suited for approximate

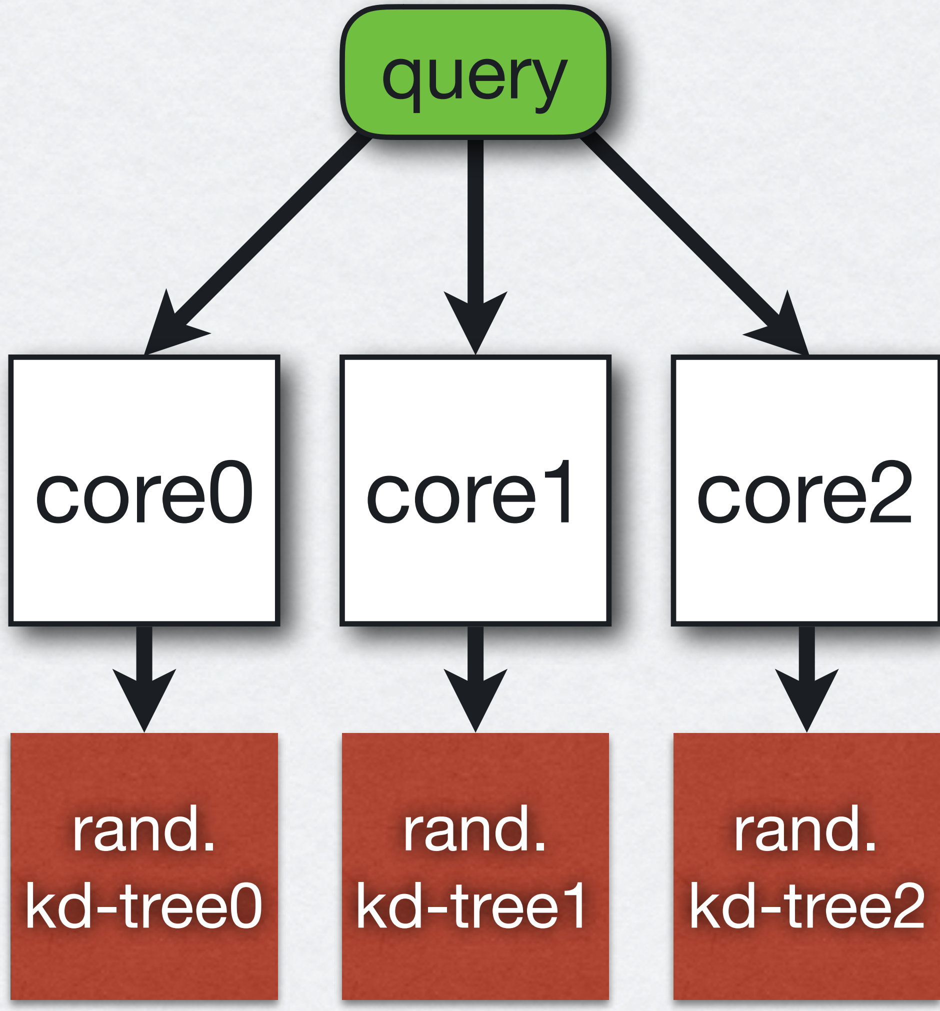
Xeon Phi

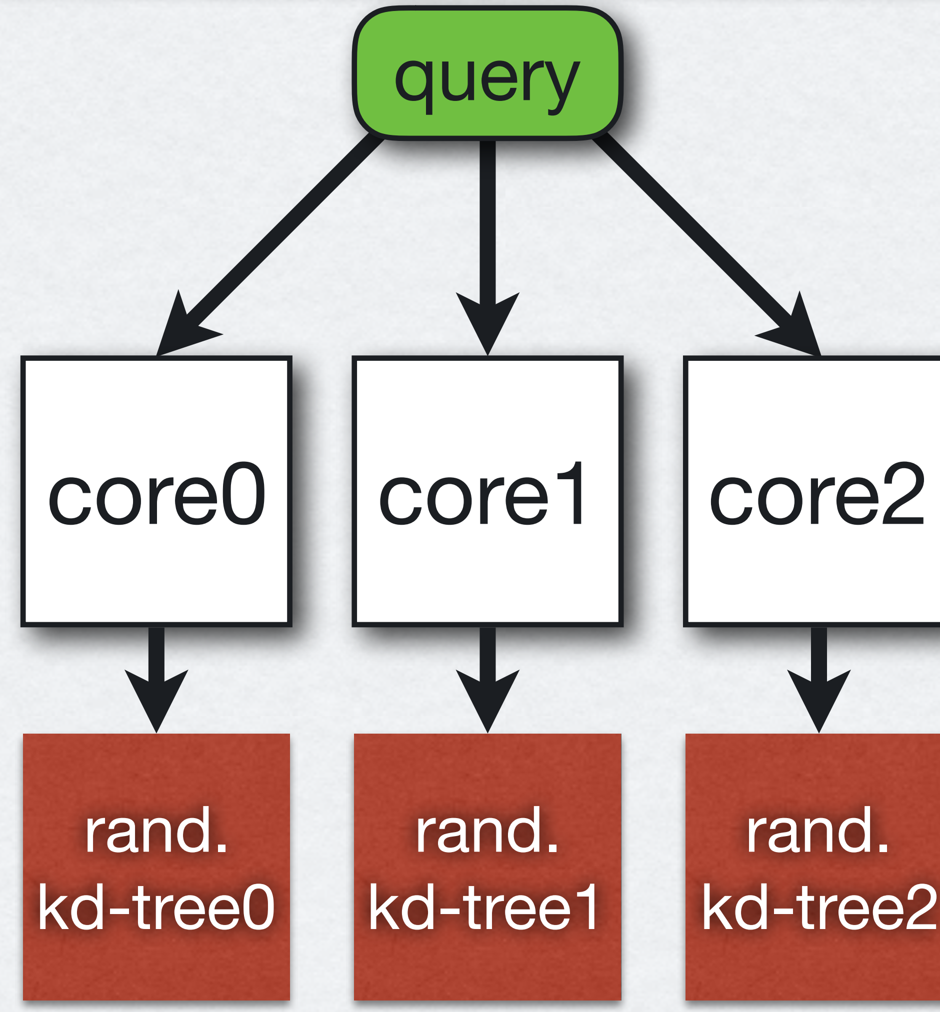
Random kd-trees distributed on cores



Xeon Phi

Random kd-trees distributed on cores



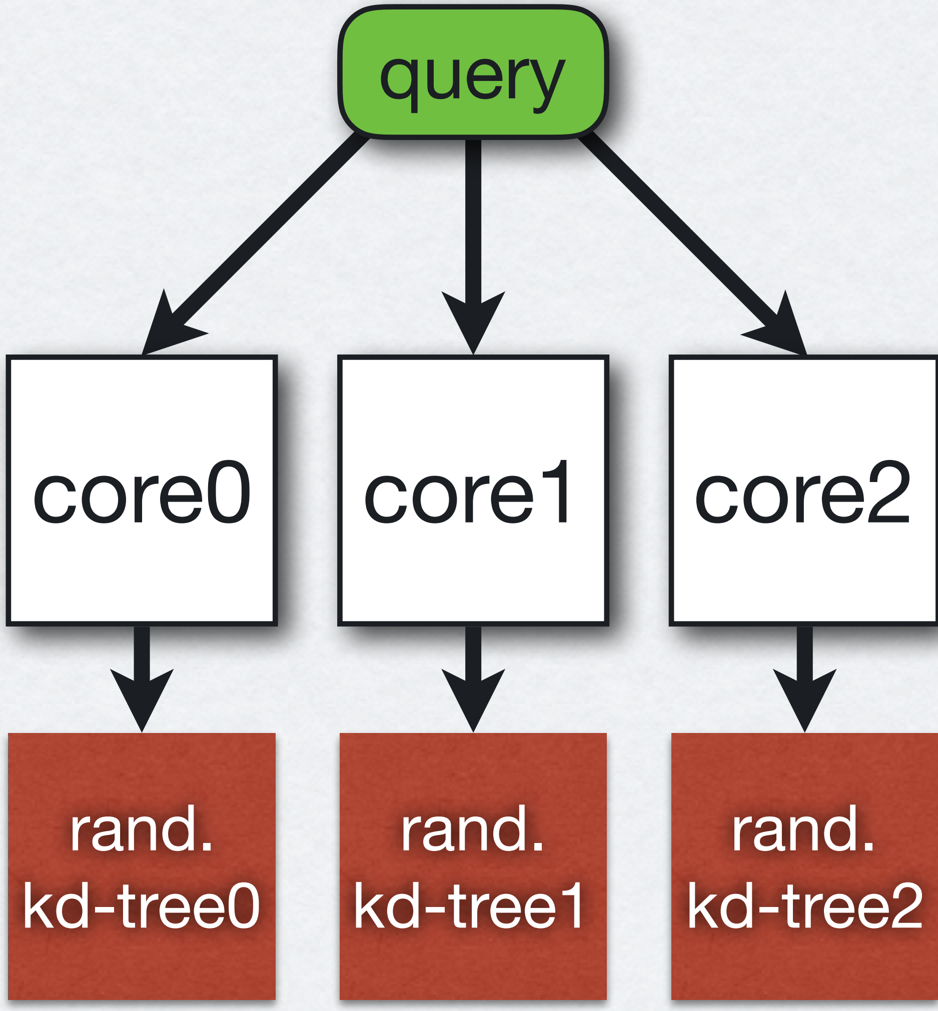


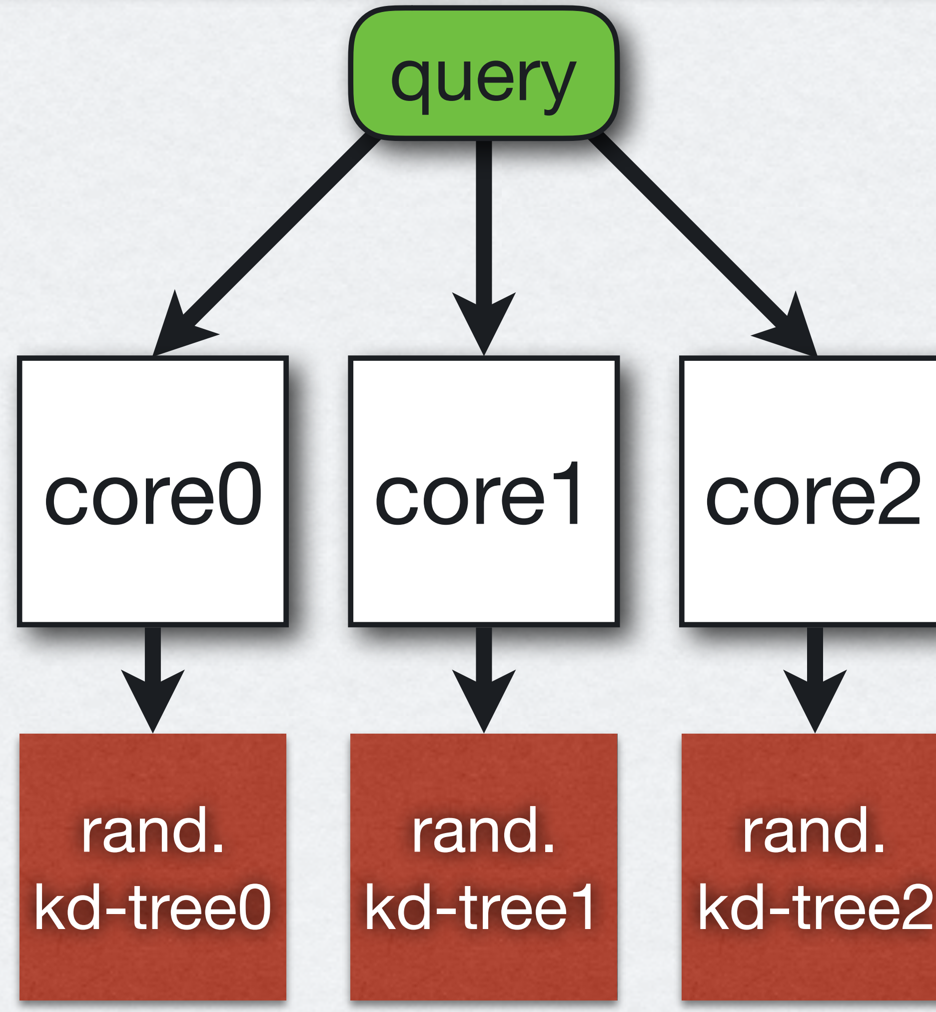
Pros

- Keep entire tree in L1 or L2 cache
- Still O(log n)

Xeon Phi

Random kd-trees distributed on cores





Pros

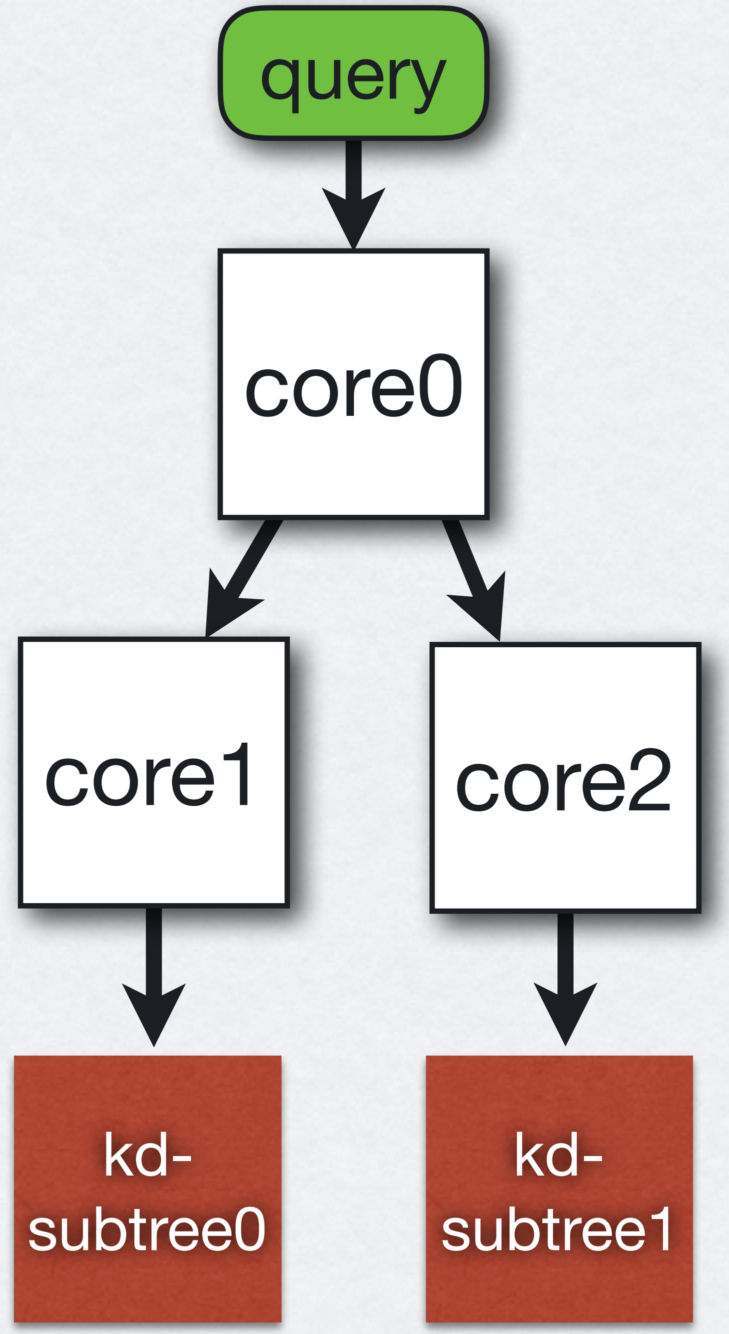
- Keep entire tree in L1 or L2 cache
- Still O(log n)

Cons

- Requires synchronization
- Limited tree size

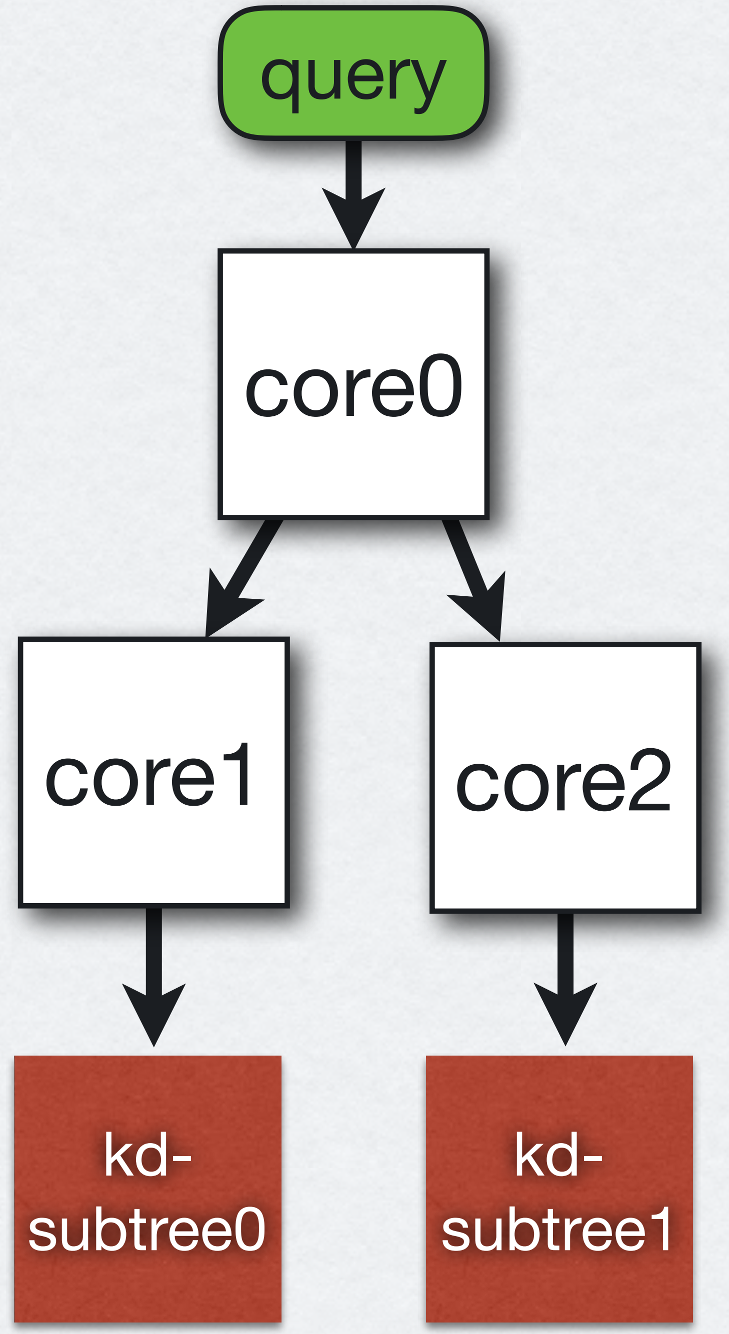
Xeon Phi

Single distributed complete tree



Xeon Phi

Single distributed complete tree

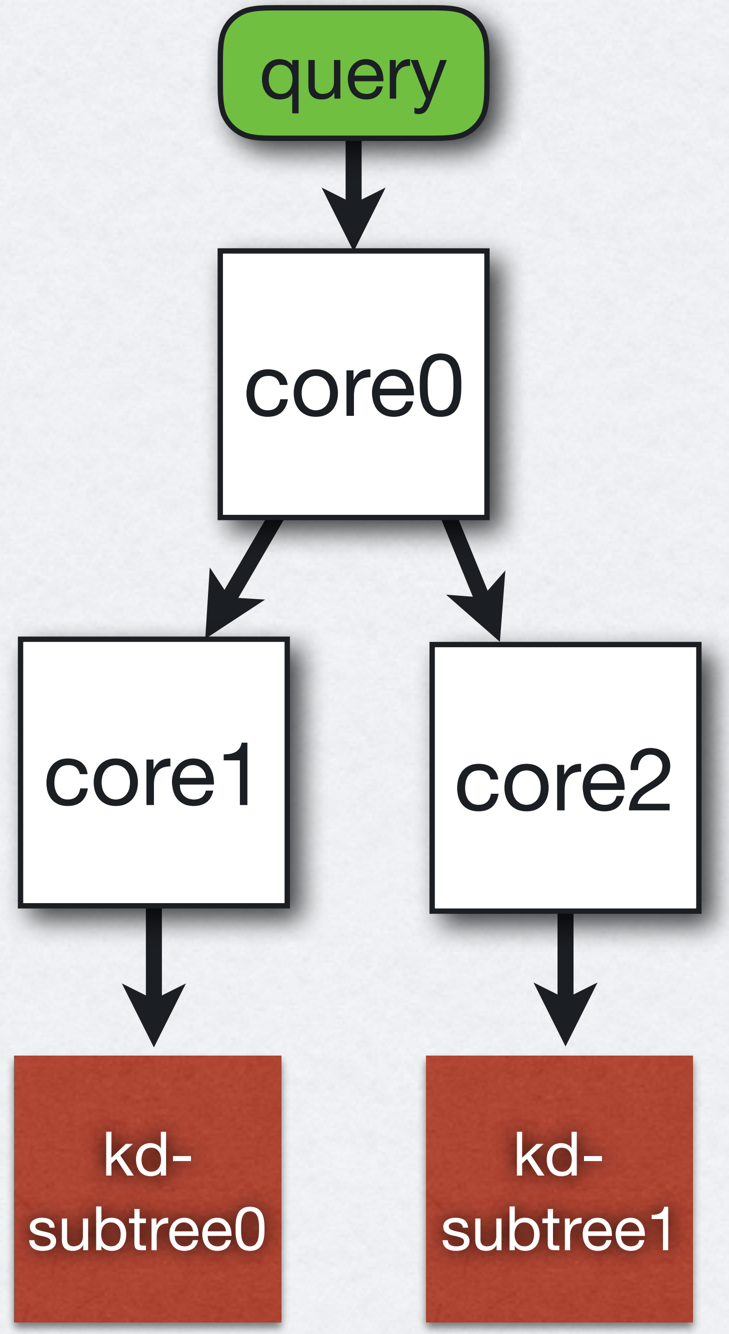


Pros

- Exact solution
- O(log n)
- Keep subtrees in L1/L2 cache

Xeon Phi

Single distributed complete tree



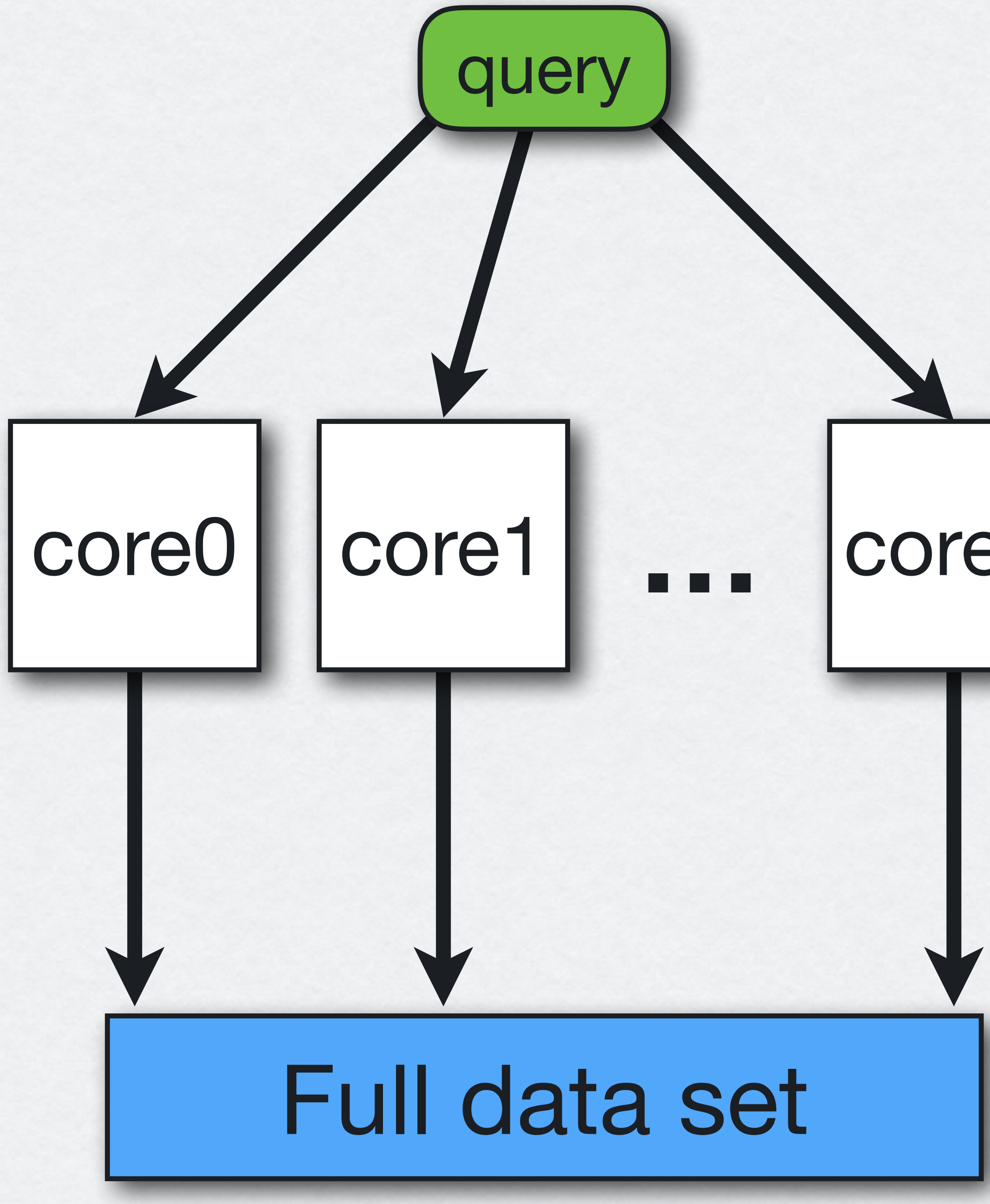
Pros

- Exact solution
- O(log n)
- Keep subtrees in L1/L2 cache

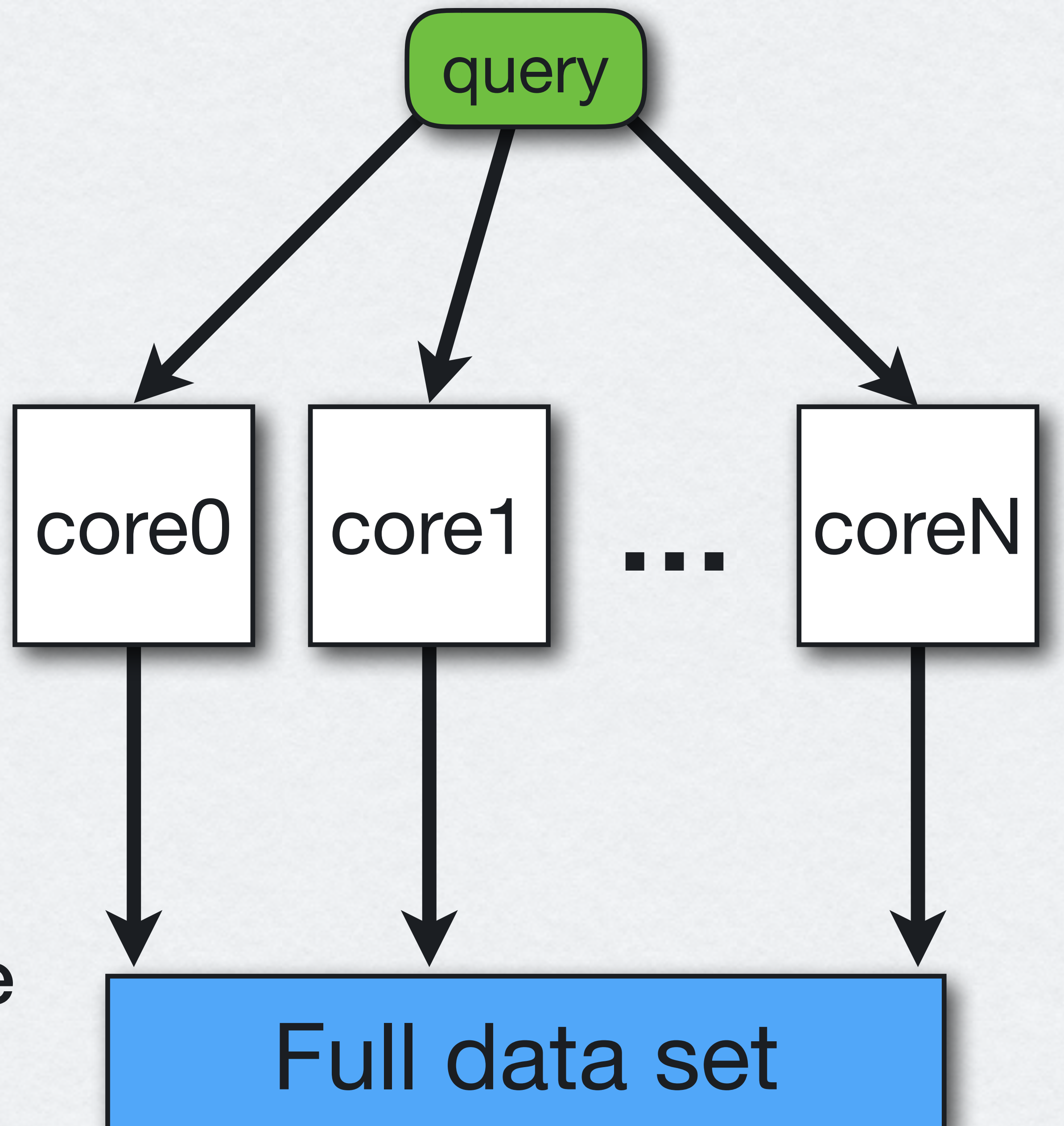
Cons

- Communication costs
- Load imbalance
- Requires extensive tweaking

Xeon Phi



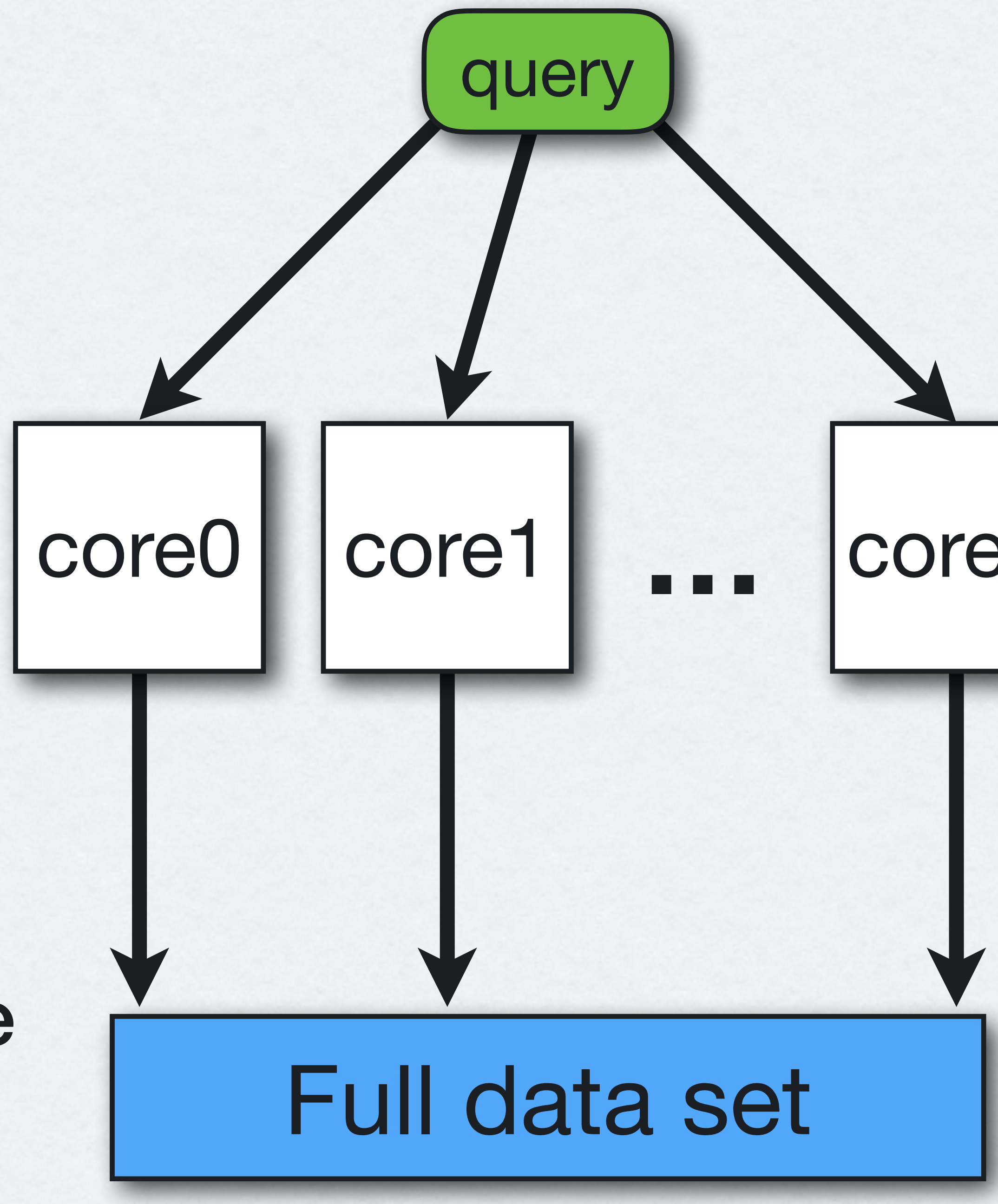
Xeon Phi



Pros

- Exact solution
- High hardware utilization
- Use highly optimized matrix library

Xeon Phi



Pros

- Exact solution
- High hardware utilization
- Use highly optimized matrix library

Cons

- O(n) complexity

Results of Current Tests

SIFT Data Set

- Linear search vs. kd-tree search is identical (using 128-dimensional SIFT data, compared to exact k-neighbors)

- Speed up of kd-tree search using median splitting compared to linear search (1’000’000 base vectors, 10’000 query vectors)

3-Dimensional Data			
k	t_{linear}	t_{kd}	Improvement
10	16.89s	0.21s	78.6x
50	18.70s	1.20s	15.6x
100	16.02s	2.89s	5.6x
500	20.40s	15.82s	1.3x
1000	31.63s	32.51s	1.0x

128-Dimensional Data			
k	t_{linear}	t_{kd}	Improvement
10	356.7s	446.0s	0.8x
50	368.5s	621.1s	0.6x
100	310.0s	526.4s	0.6x
500	713.3s	744.5s	1.0x
1000	721.2s	563.0s	1.3x

Run on 1 Brutus node with four 12-core AMD Opteron 6174 CPUs and 64 GB of RAM
GCC 4.8.2 -std=c++11 -O3 -g

Milestones

- Naive kNN search ☑
- kd-Tree kNN search ☑
- Randomized approximate kd-Trees on CPU ☐
- Parallelize tree build and kNN search ☐
- Tuning to Xeon Phi ☐