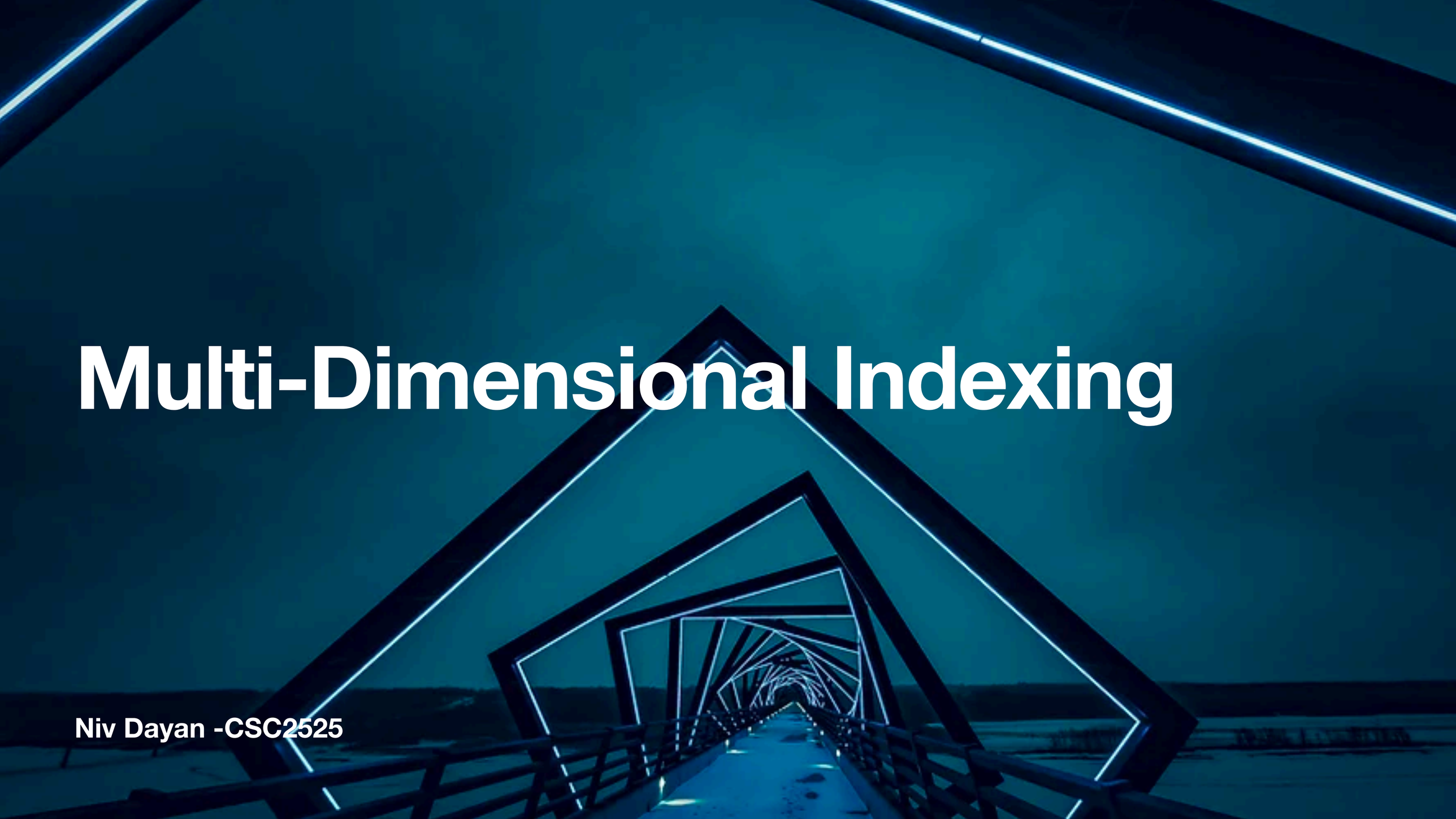
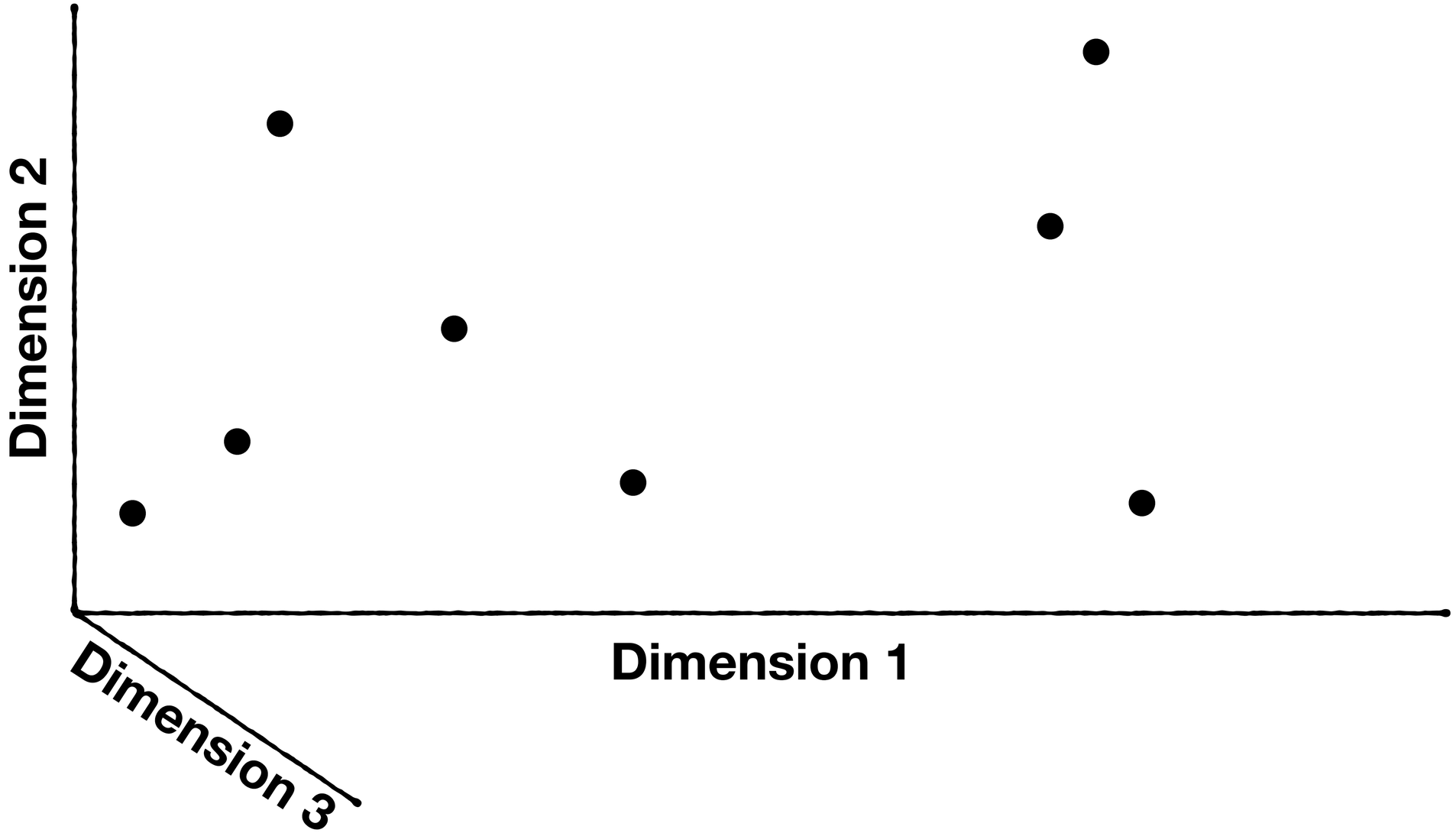
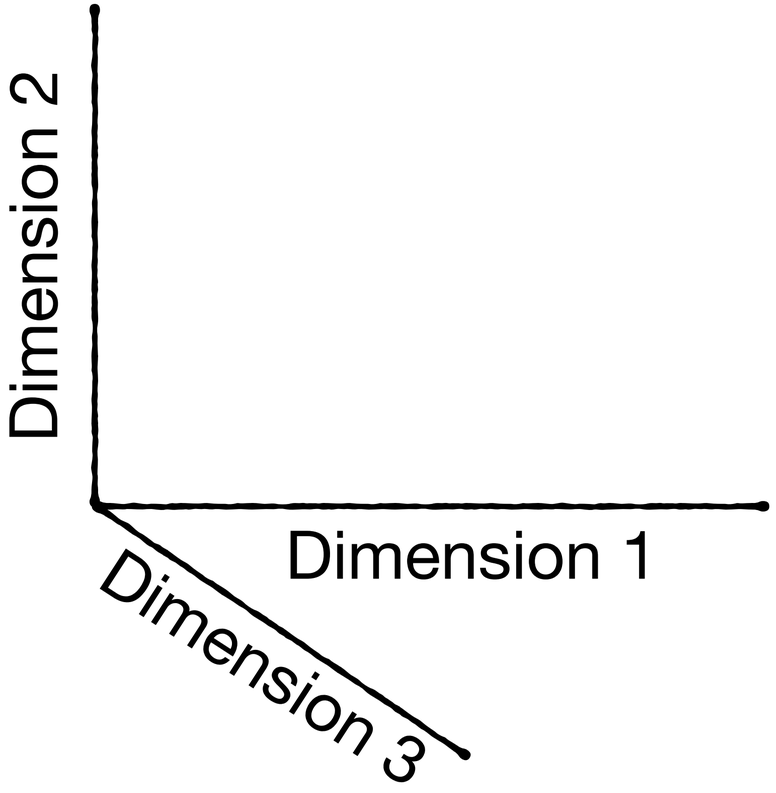


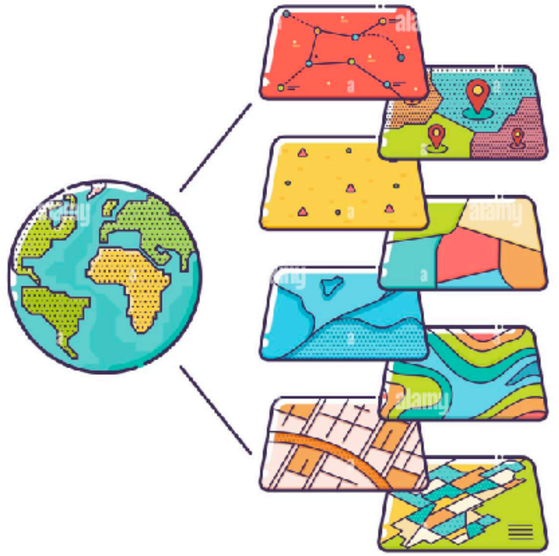
Multi-Dimensional Indexing

Niv Dayan -CSC2525

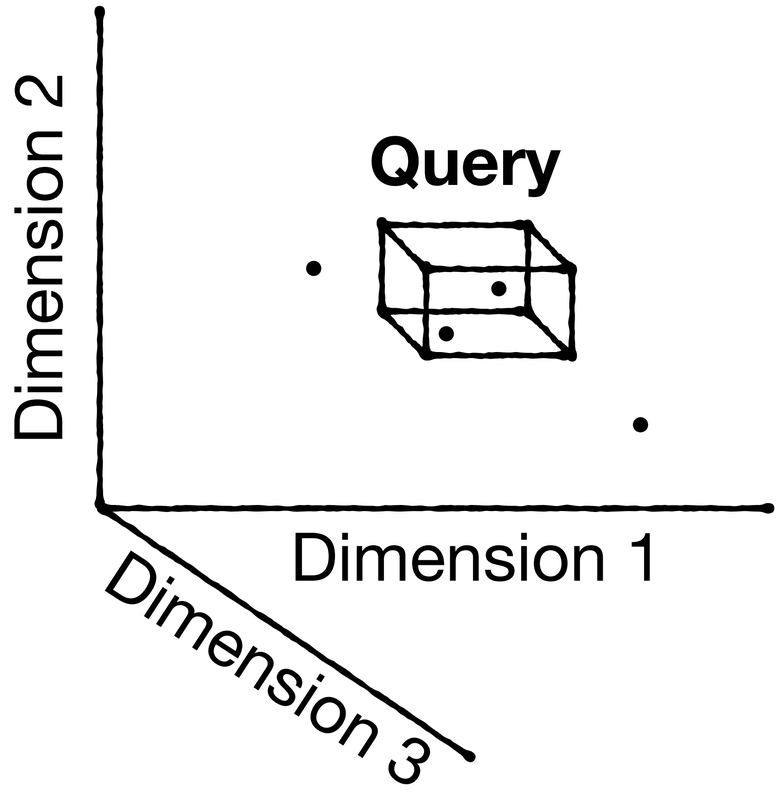

Dimension 2
Dimension 1
Dimension 3

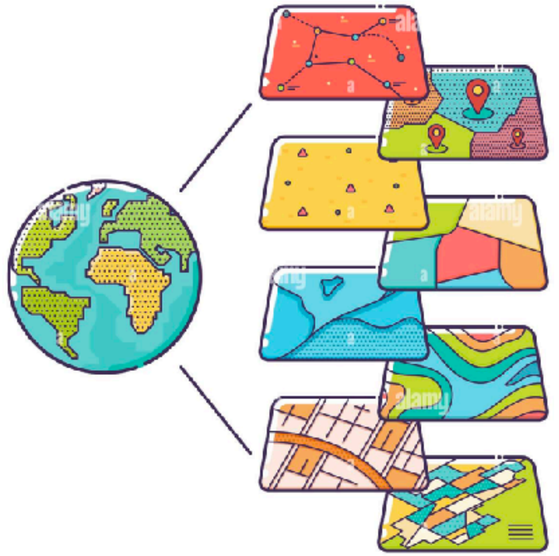
Spatial Data



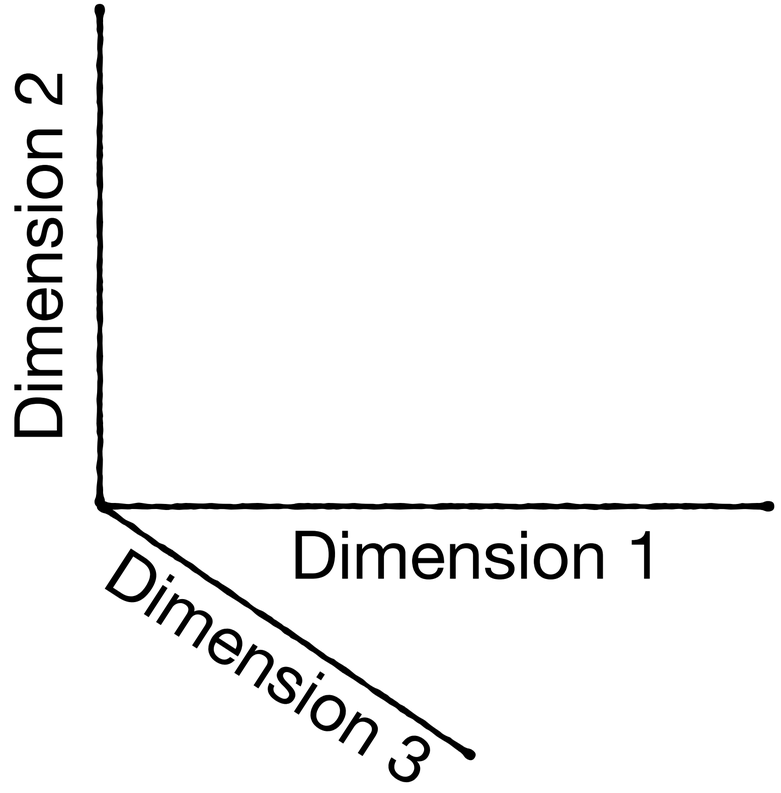


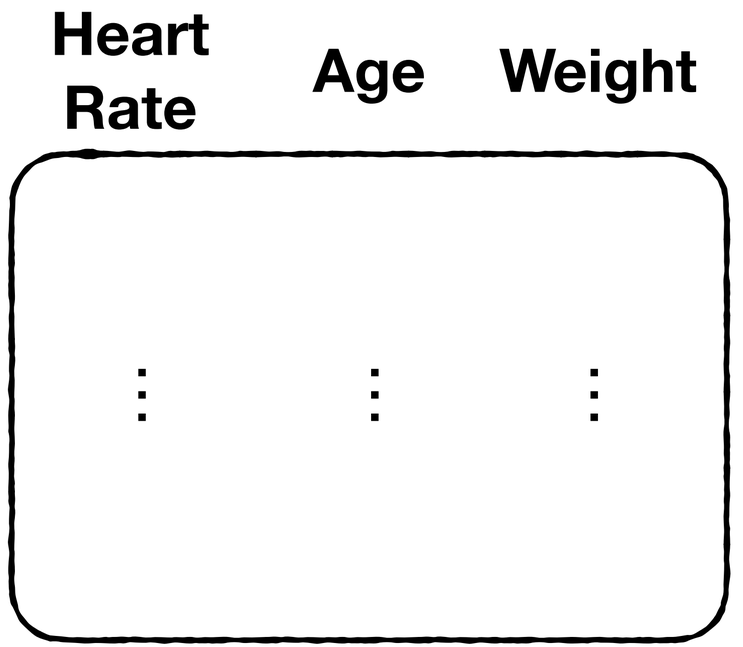
Spatial Data



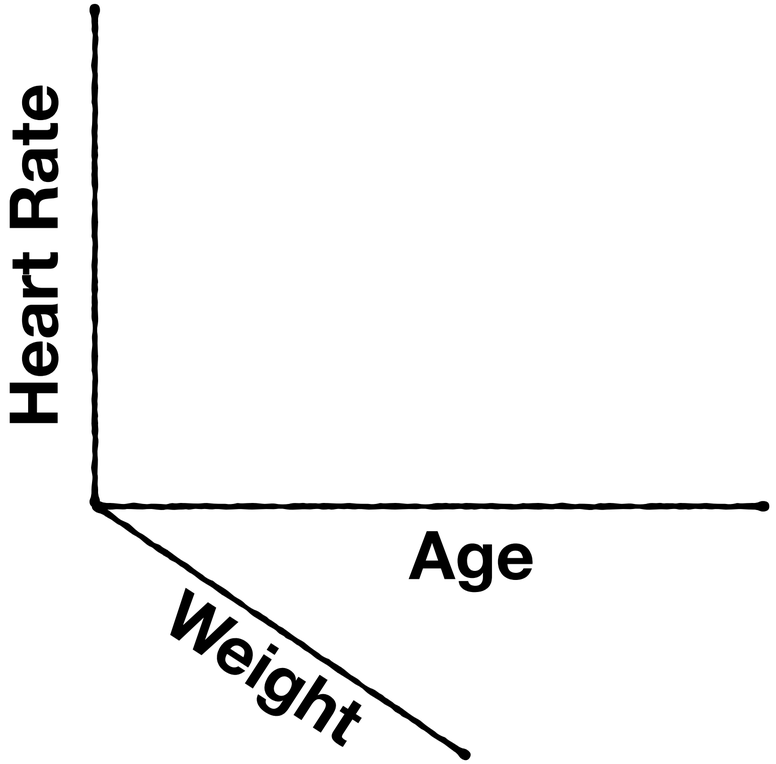


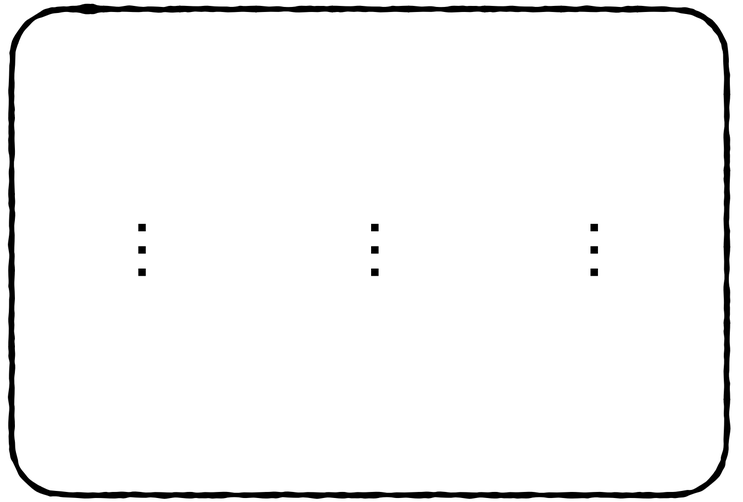
Tables in relational DB



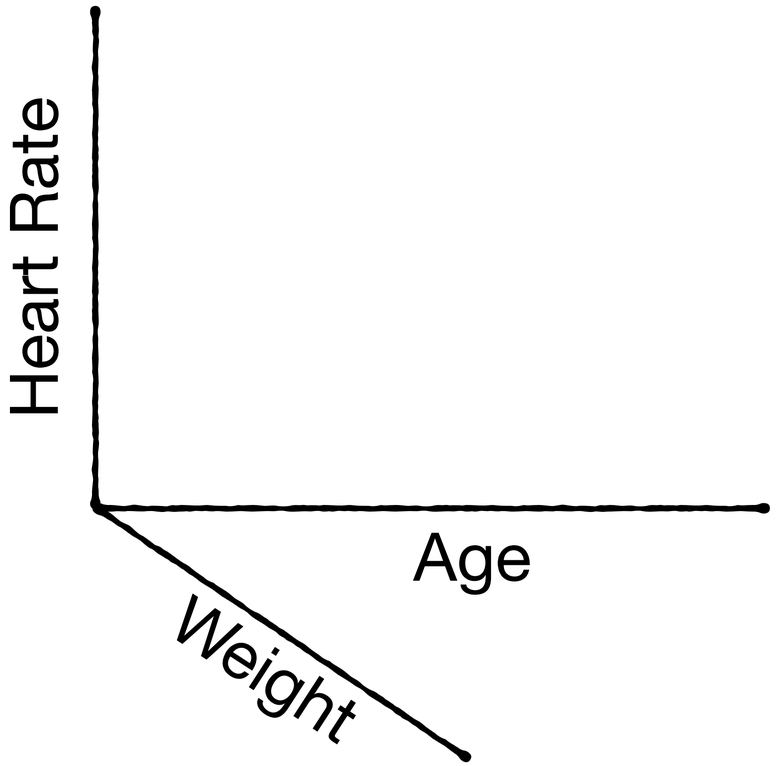


Tables in relational DB





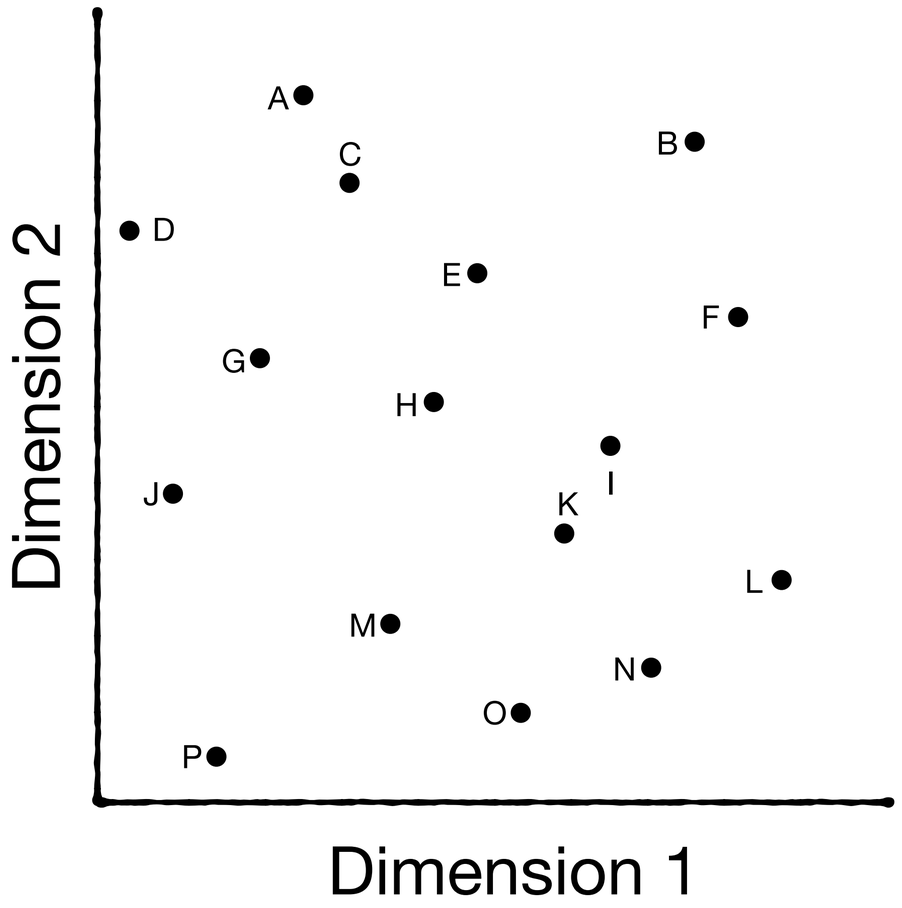
How to index multi-dimensional data?



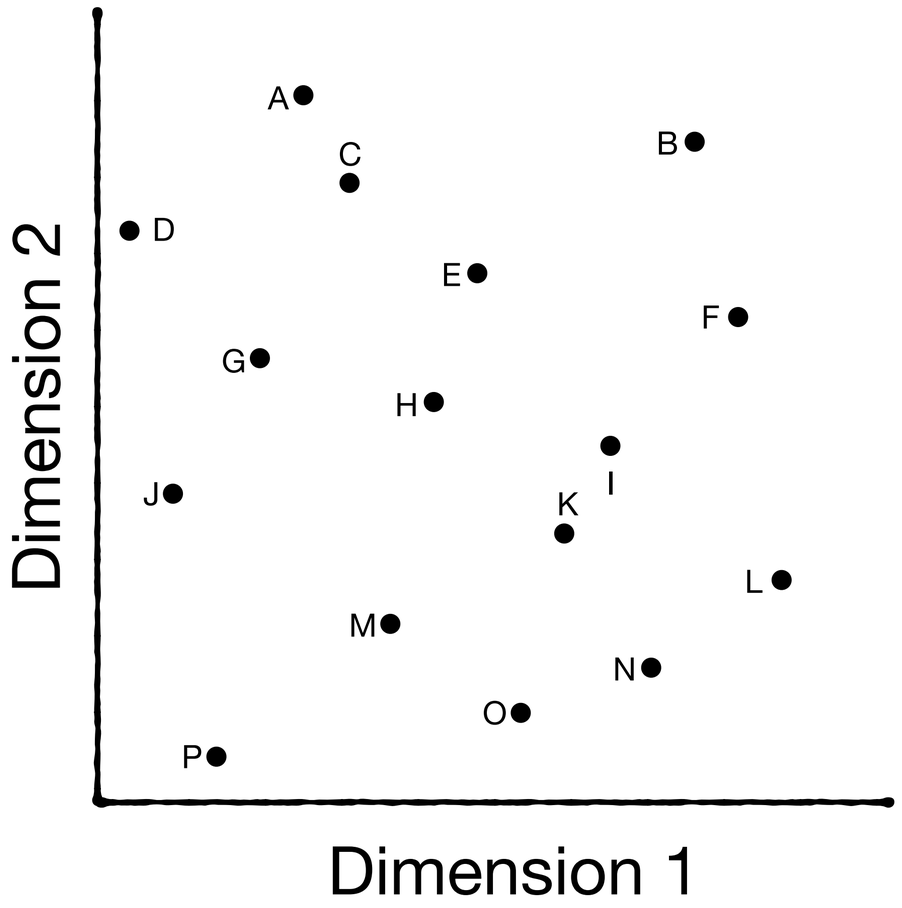
KD-Trees	R-Trees	R+tree	R*-tree	UB-tree
1975	1984	1987	1990	2000

KD-Trees - 1975

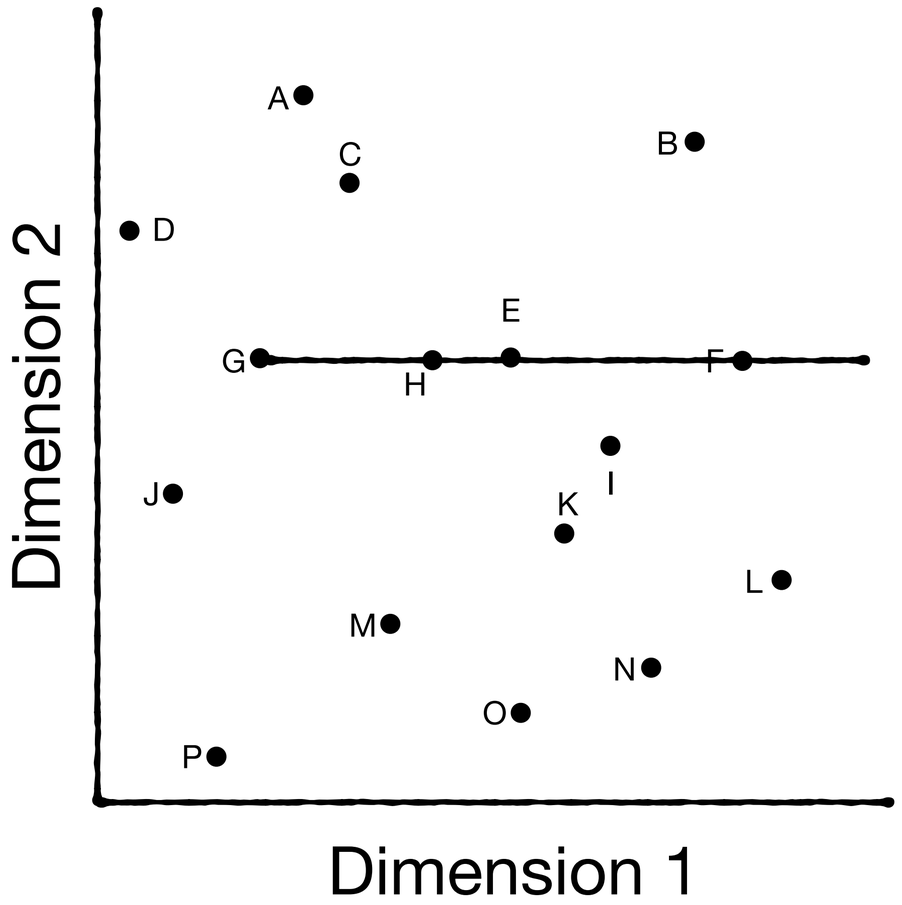
An algorithm for finding best matches in logarithmic expected time. ACM TOMS.
Jerome Harold Friedman, Jon Louis Bentley, and Raphael Ari Finkel.


Dimension 2
Dimension 1
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P

Assume no single line crosses between any 3 points

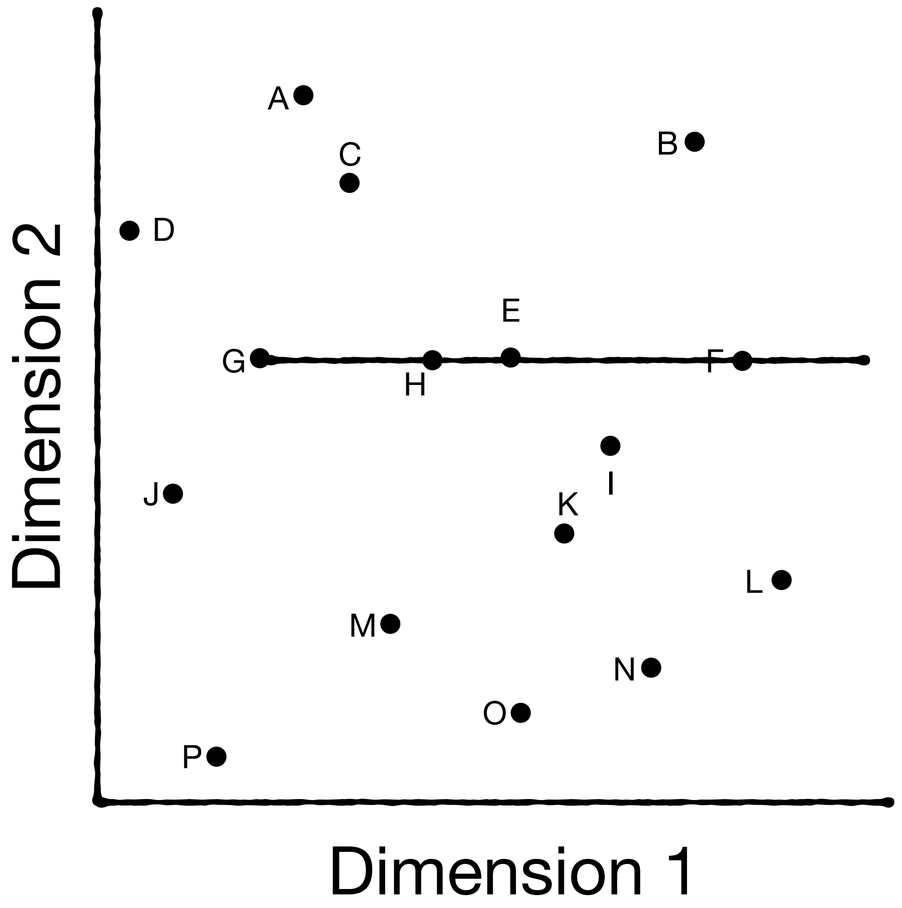


Assume no single line crosses between any 3 points

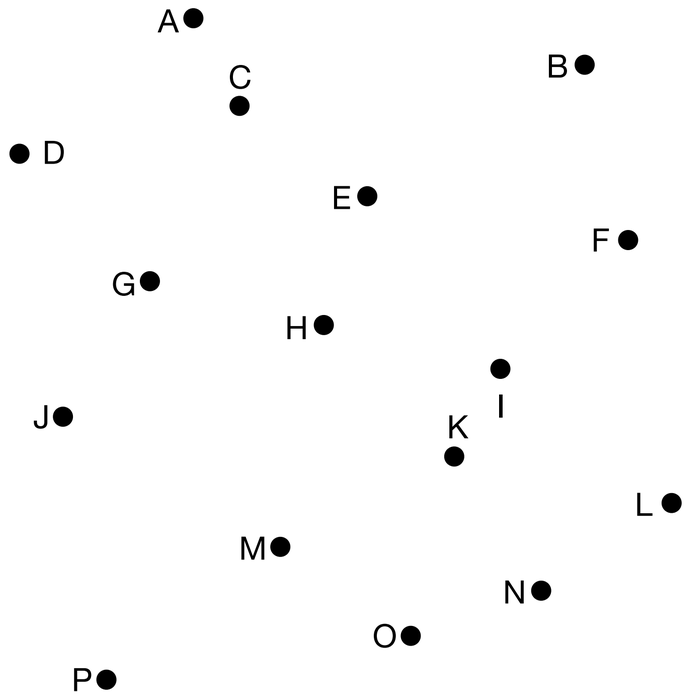


Assume no single line crosses between any 3 points

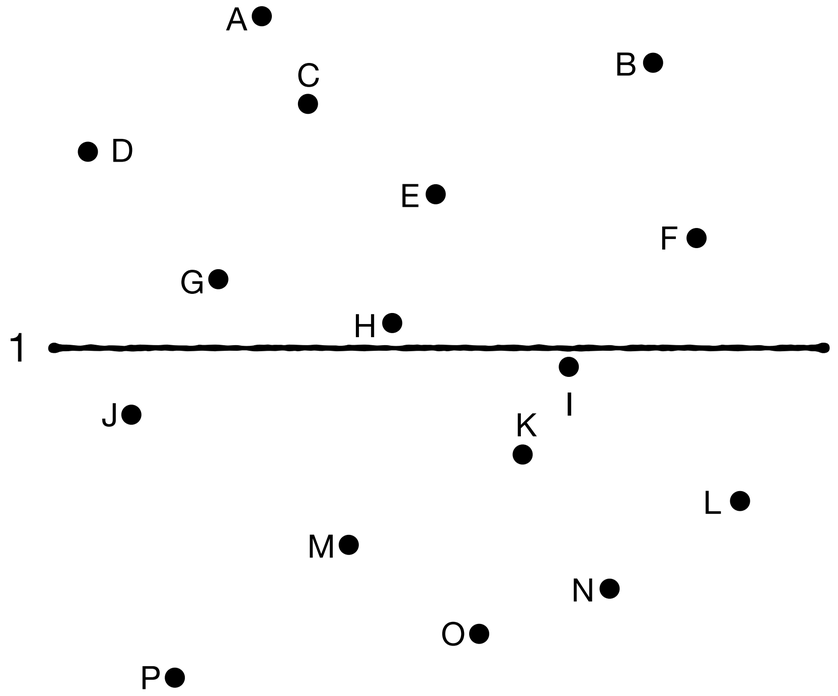
Important for our later analysis

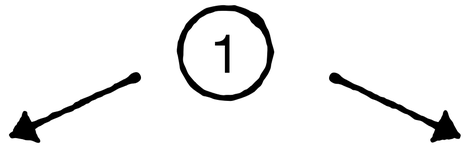


Partition based on median in one dimension - add node to tree



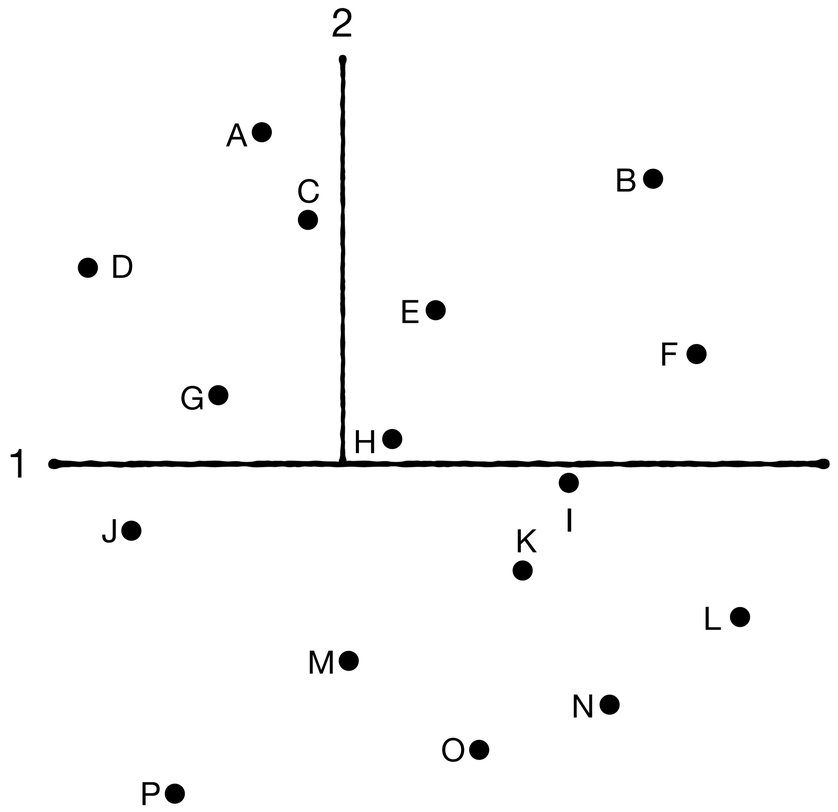
Partition based on median in one dimension - add node to tree

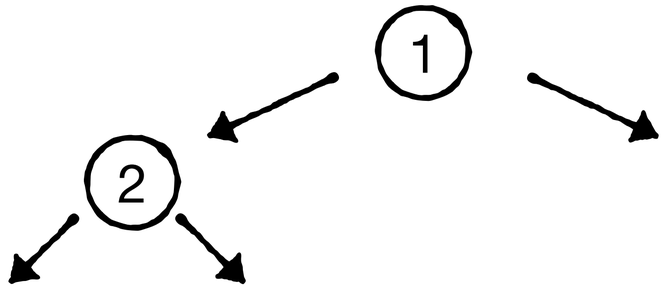




Partition based on median in one dimension - add node to tree

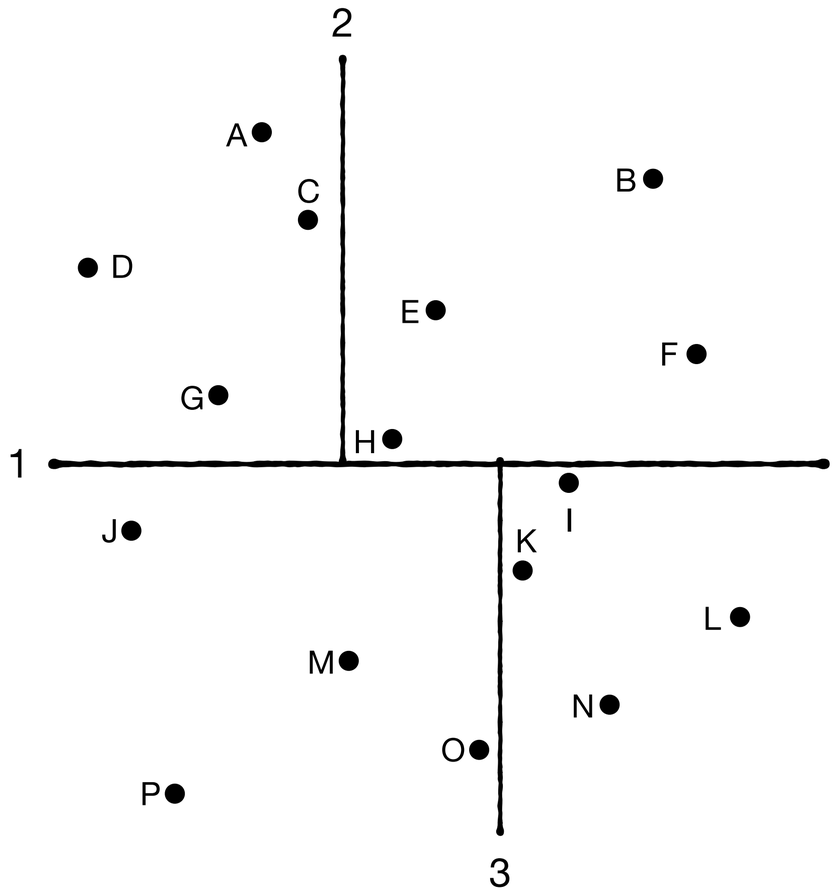
Recurse and alternate the dimensions

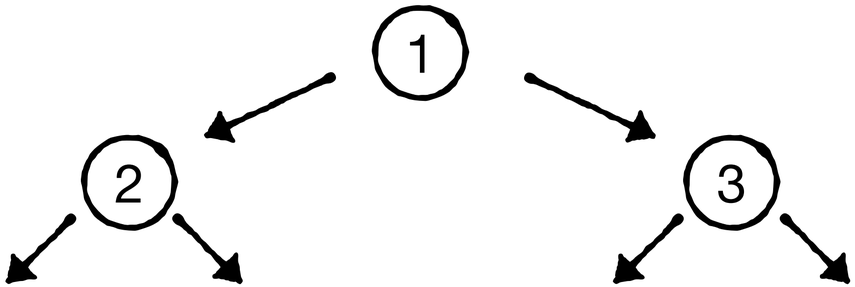




Partition based on median in one dimension - add node to tree

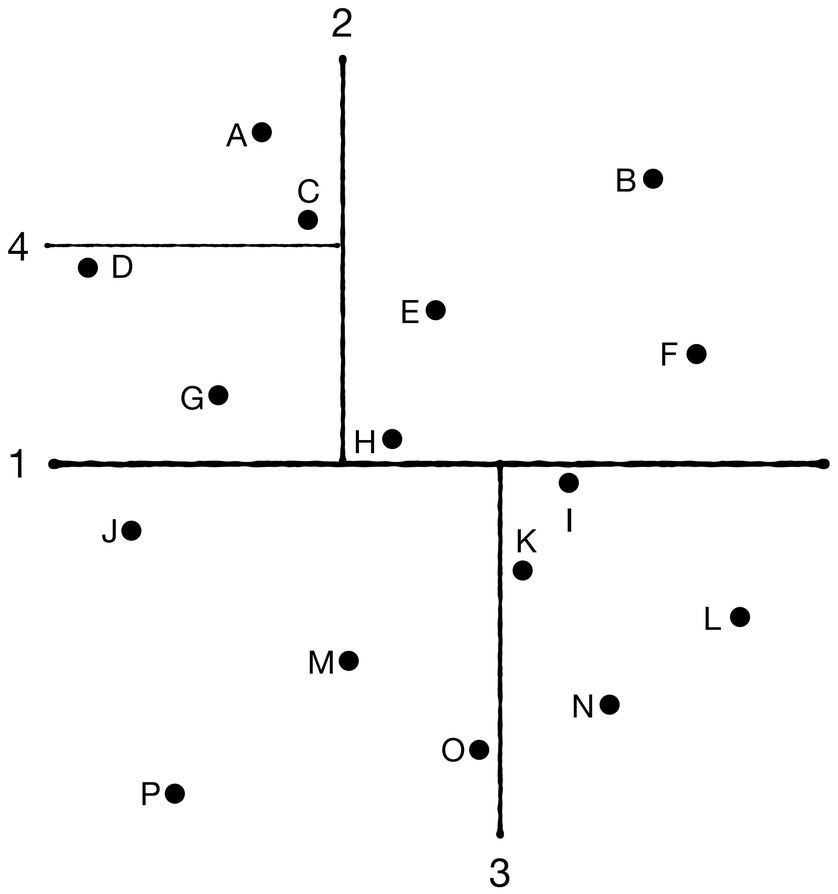
Recurse and alternate the dimensions

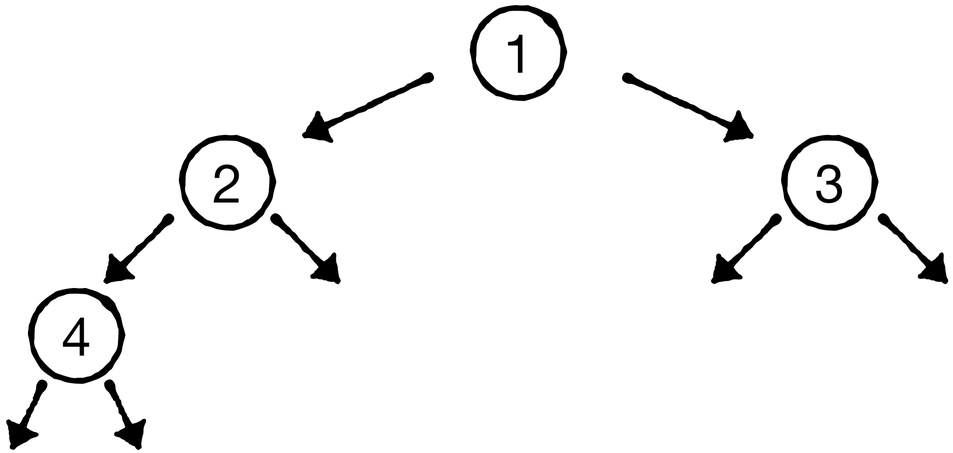


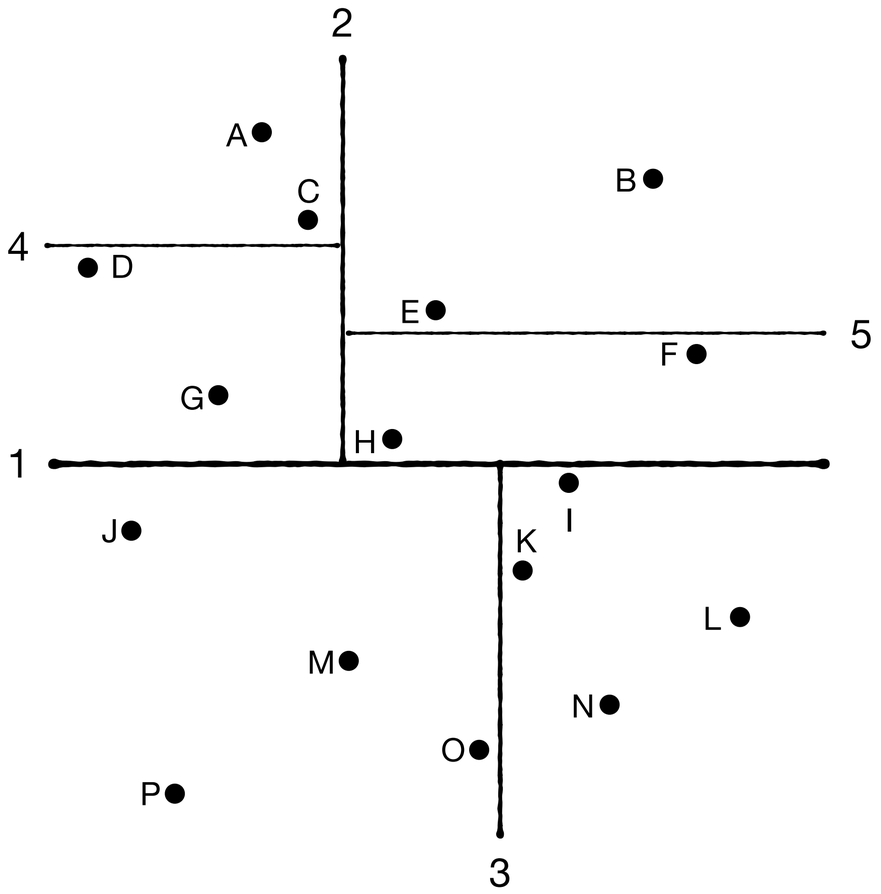


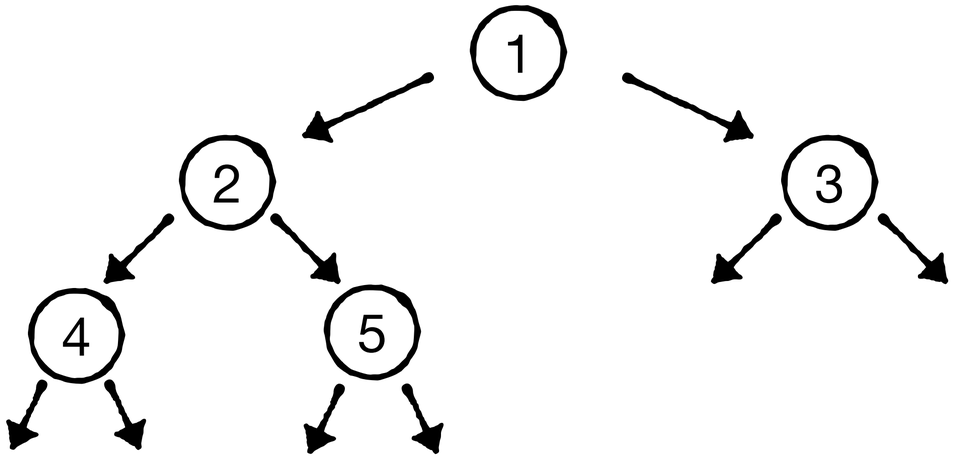
Partition based on median in one dimension - add node to tree

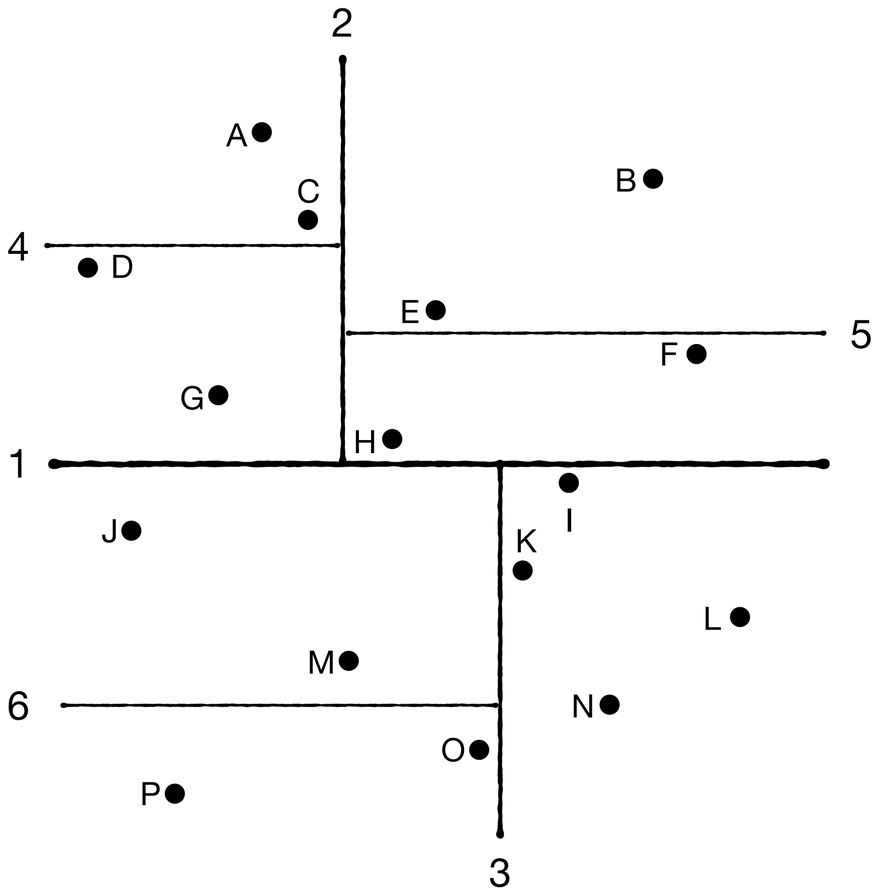
Recurse and alternate the dimensions

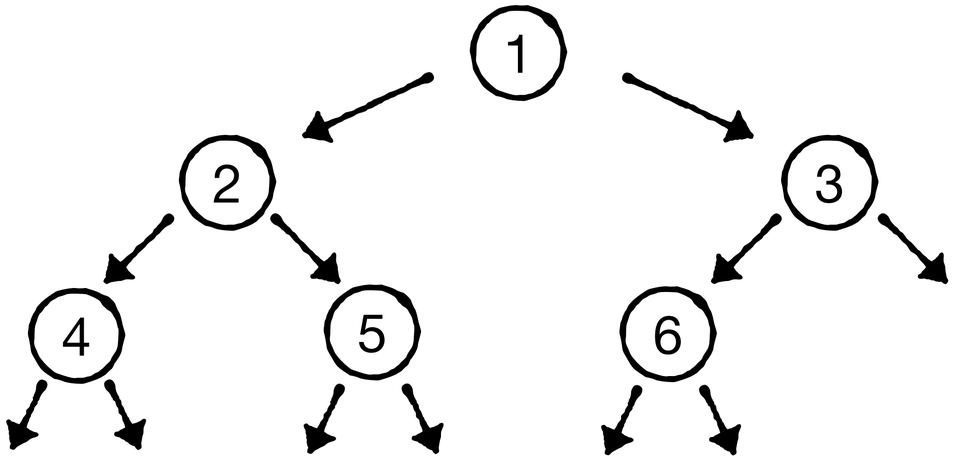


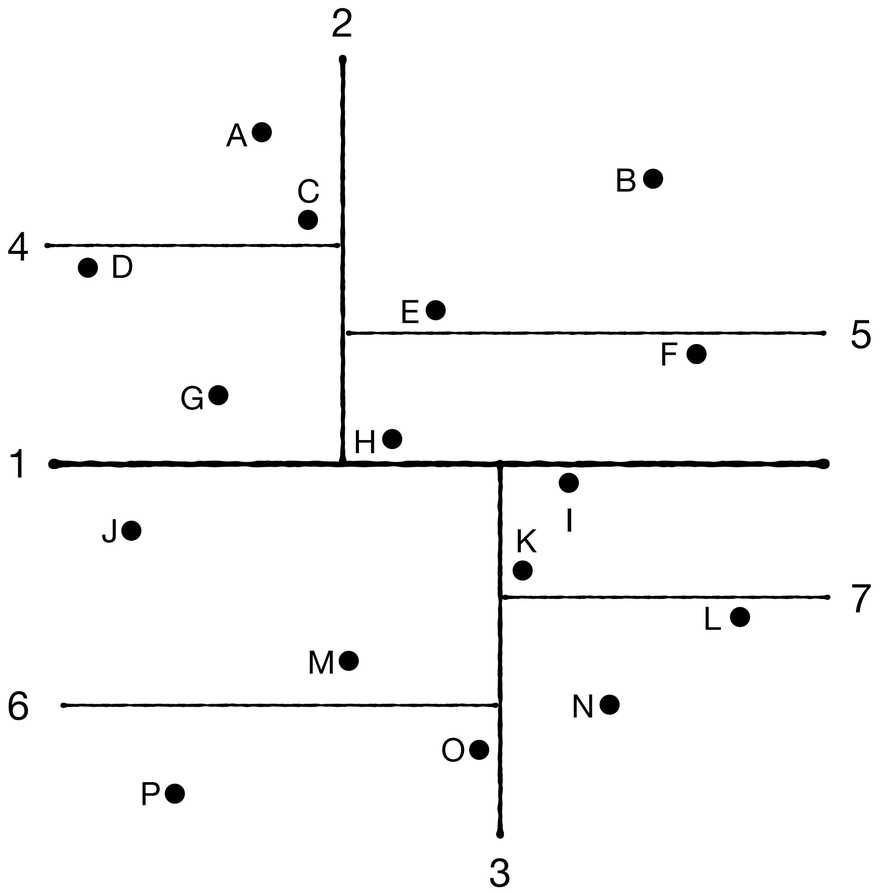


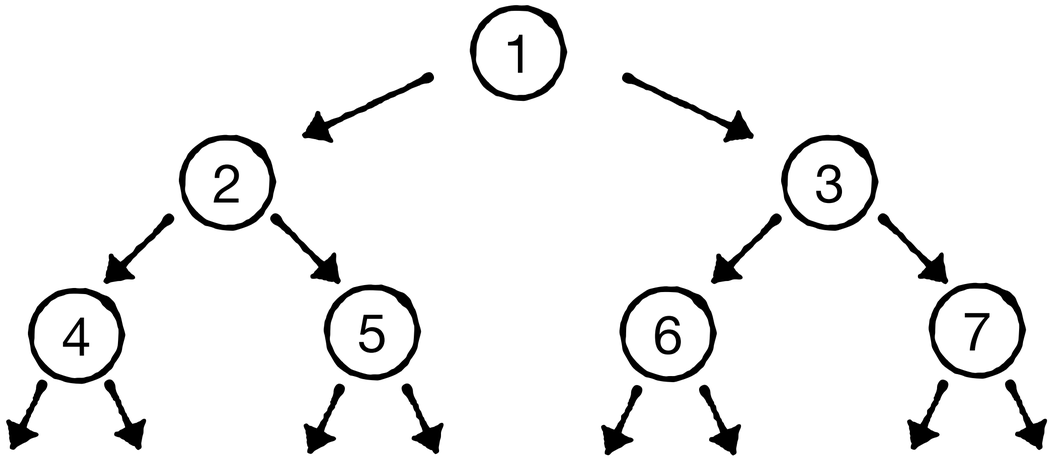

2
A
B
C
4
D
E
5
F
G
H
1
I
J
K
L
M
N
O
P
3

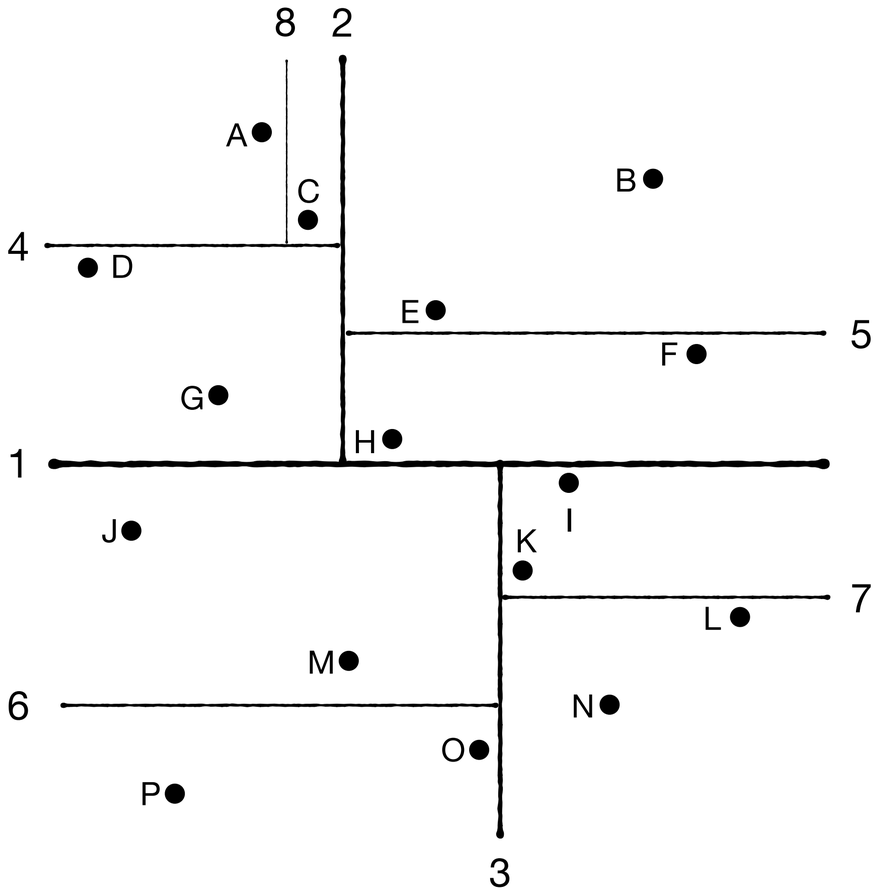

1
2
3
4
5

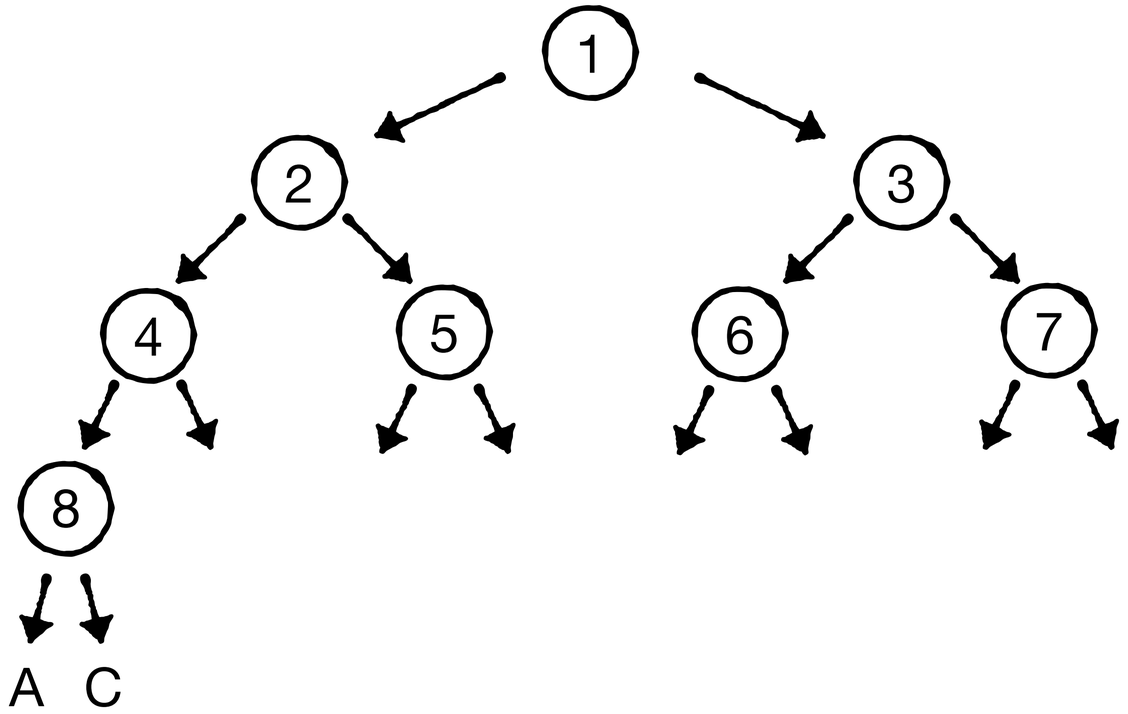

2
A
B
C
4
D
E
5
F
G
1
H
I
J
K
L
M
6
N
O
P
3

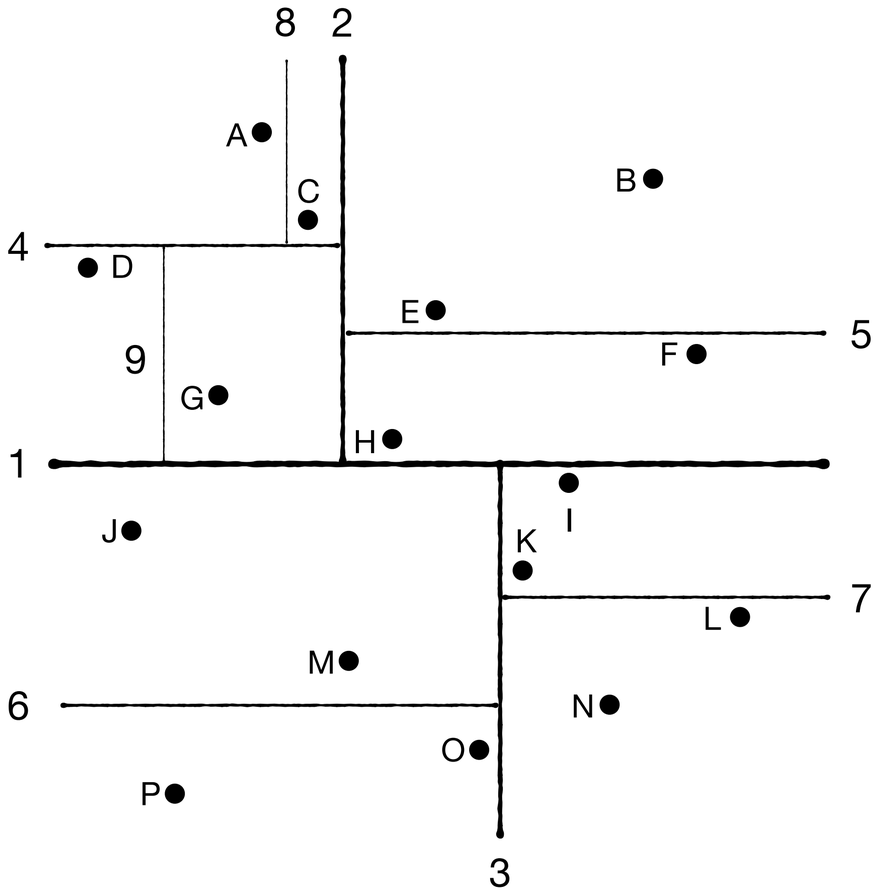

1
2
3
4
5
6

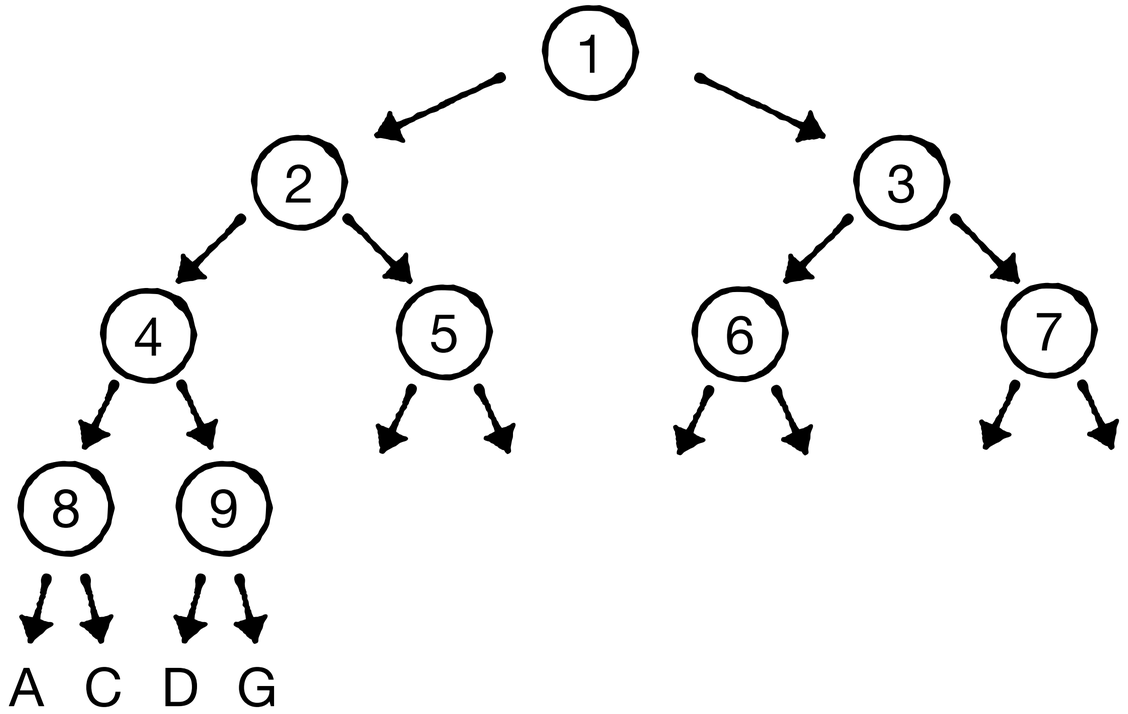

2
A
B
C
4
D
E
5
F
G
1
H
I
J
K
7
L
M
6
N
O
P
3

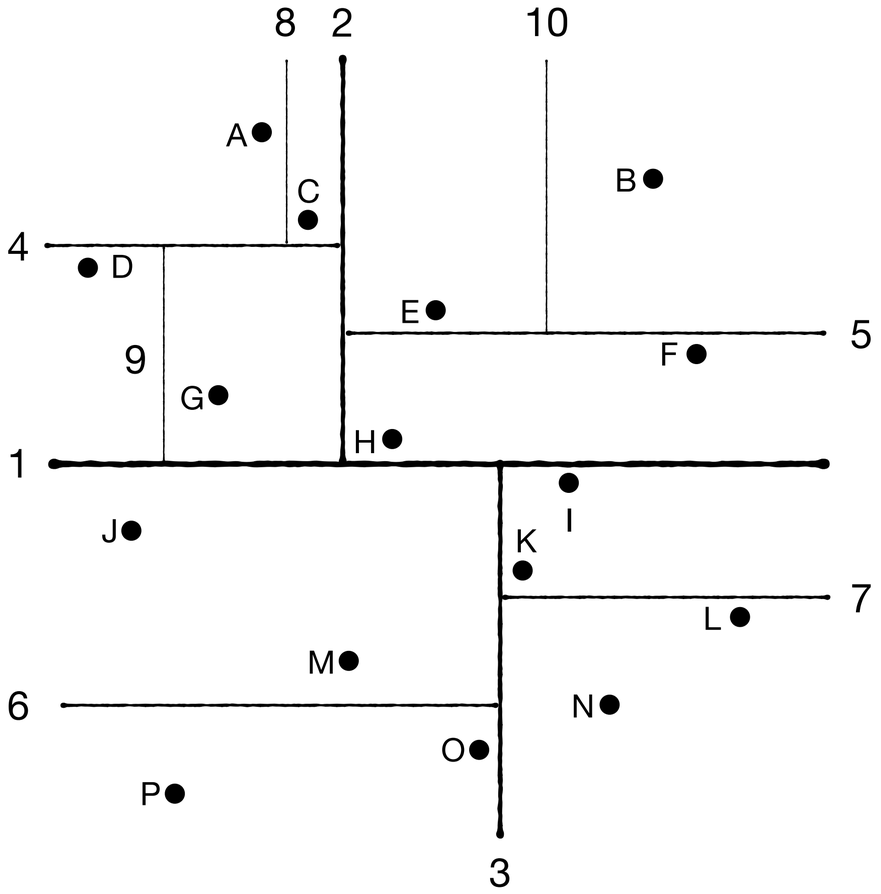

1
2
3
4
5
6
7

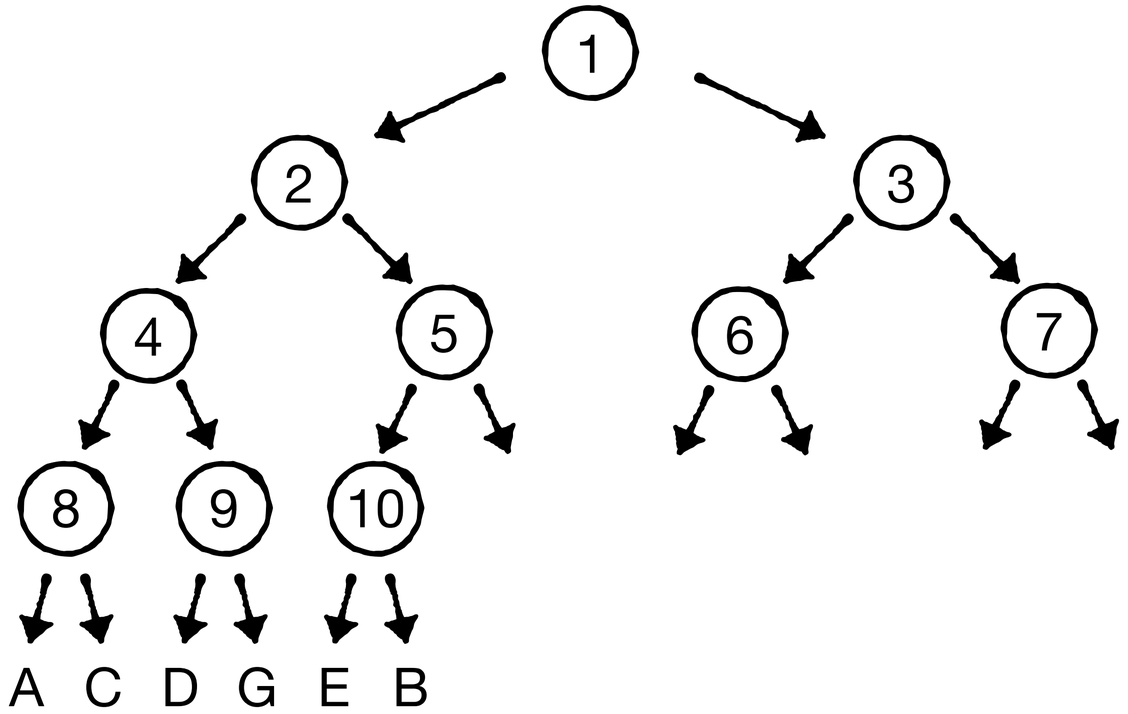

8
2
A
C
B
4
D
E
5
F
G
1
H
I
J
K
7
L
M
6
N
O
P
3

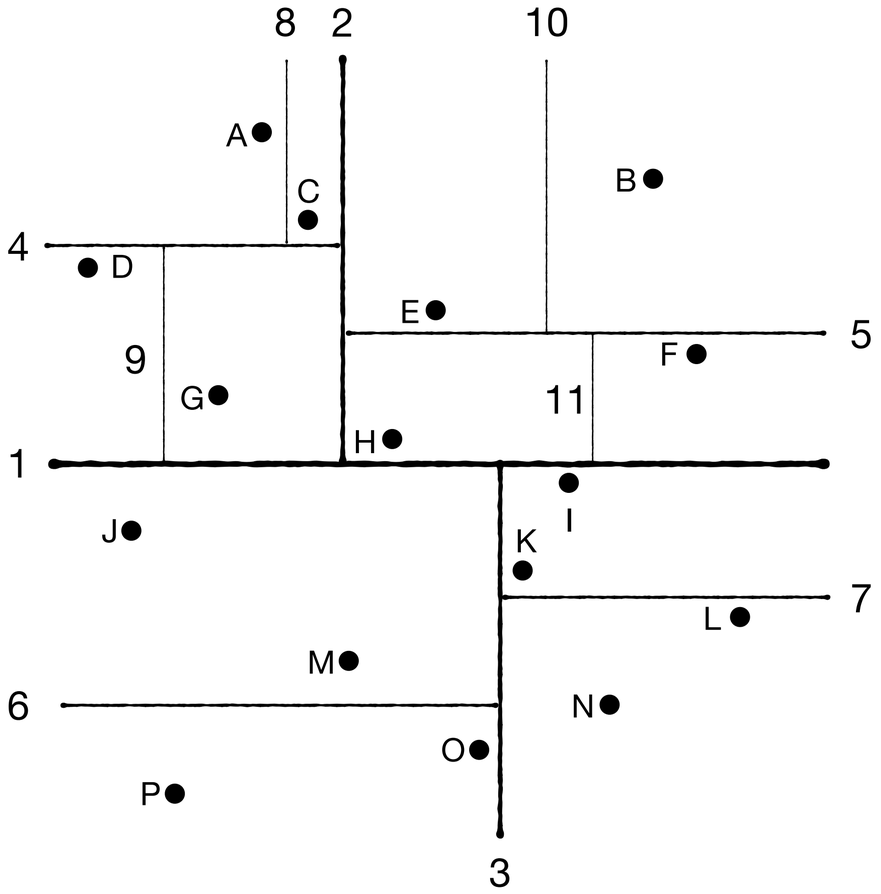

1
2
3
4
5
6
7
8
A
C

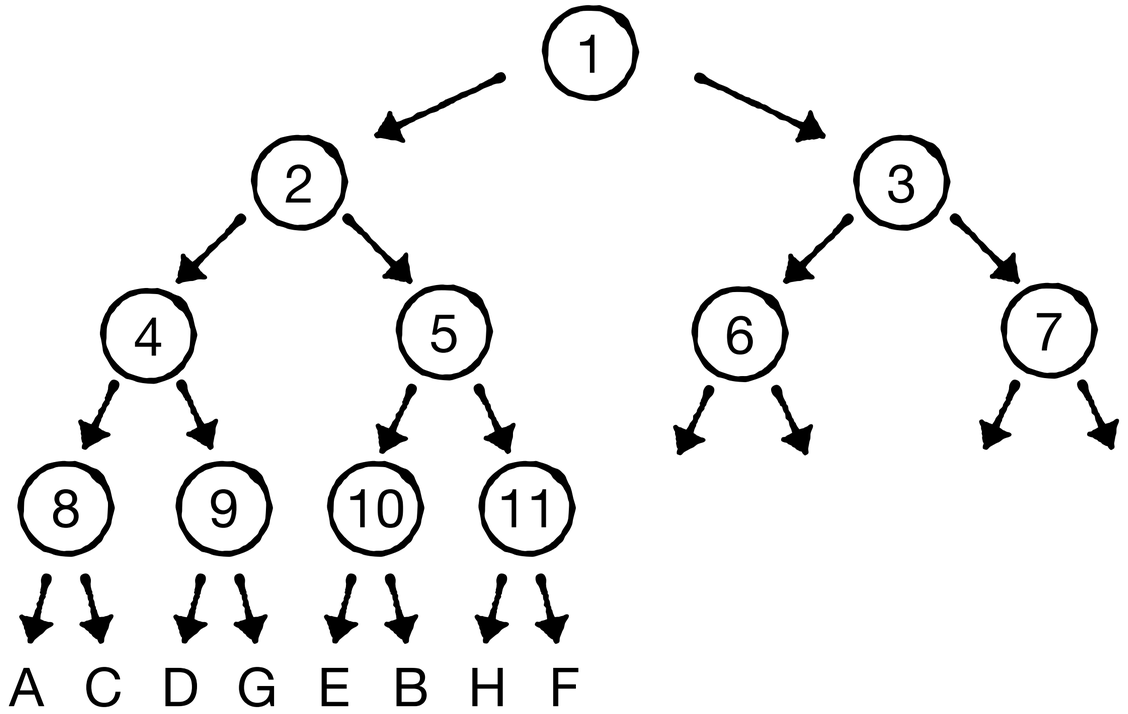

8
2
A
C
B
4
D
E
5
9
F
G
1
H
I
J
K
7
L
M
6
N
O
P
3

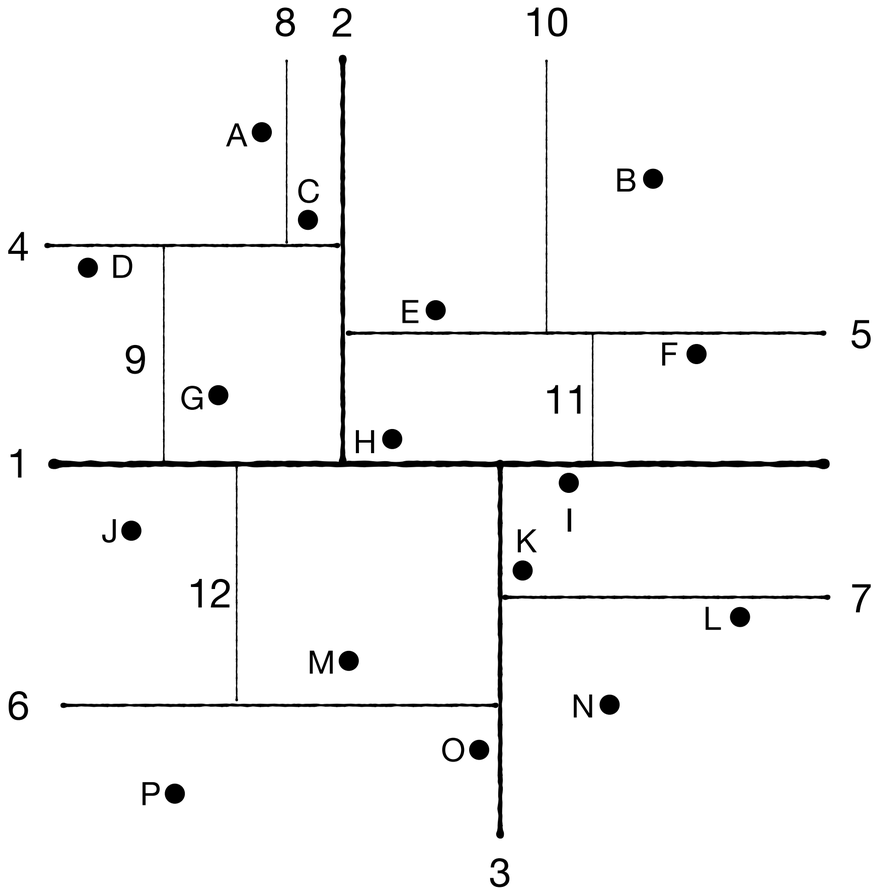

1
2
3
4
5
6
7
8
9
A C D G

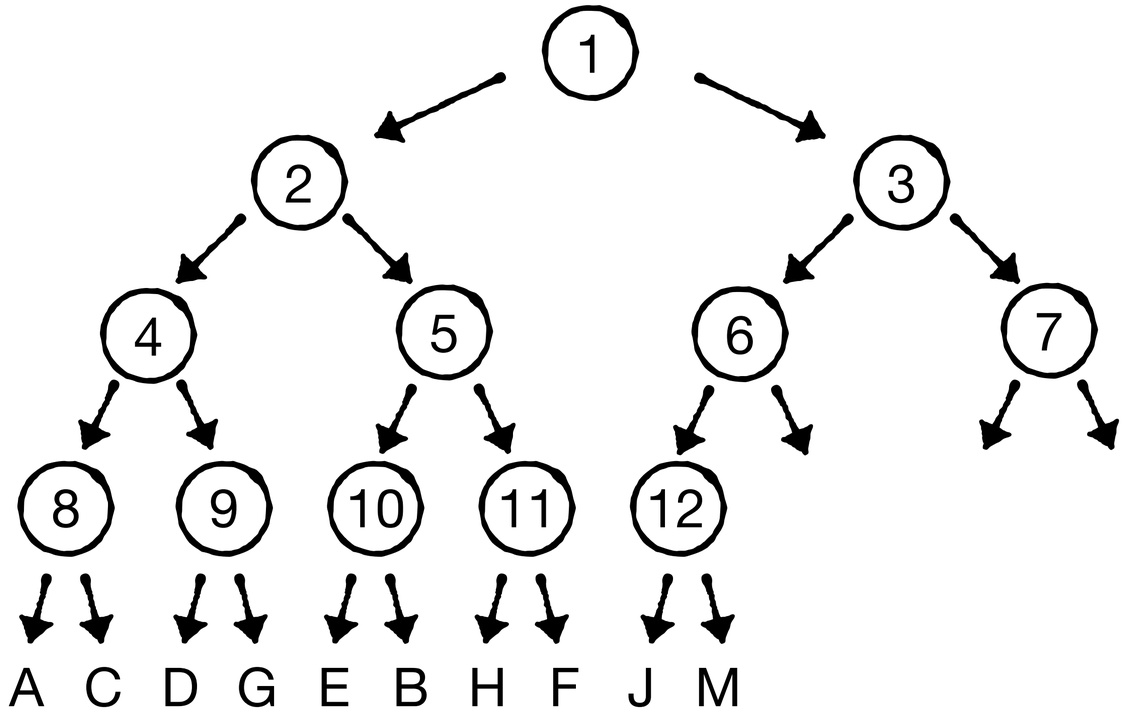

8
2
10
A
C
B
4
D
E
5
9
F
G
1
H
I
J
K
7
L
M
6
N
O
P
3

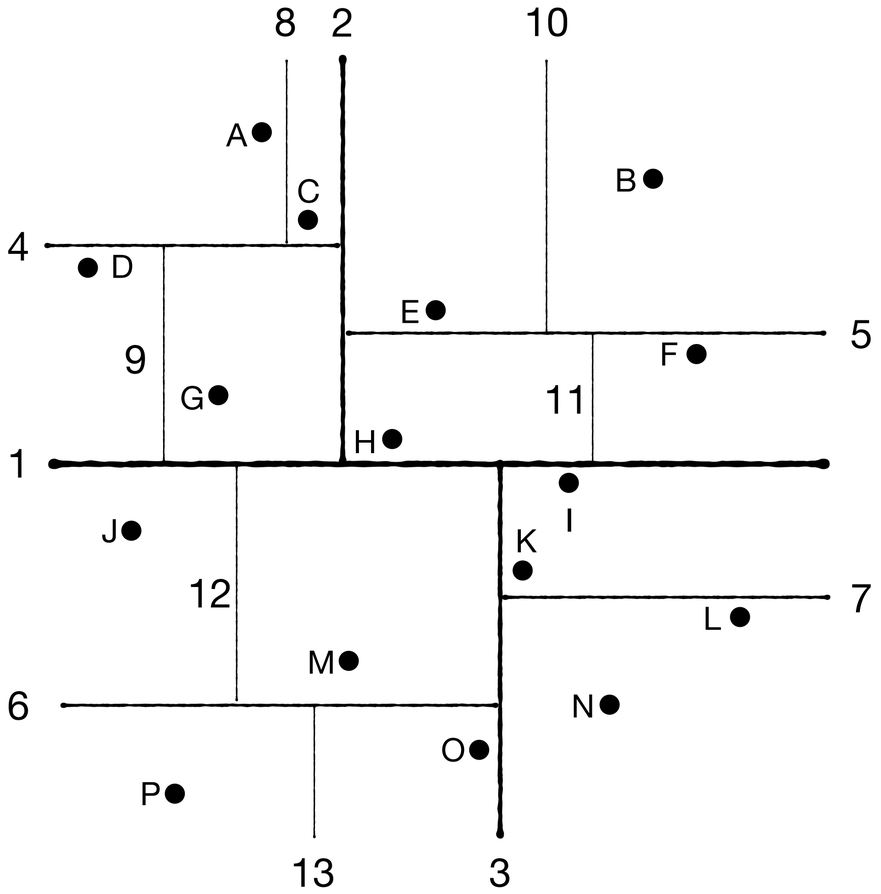

1
2
3
4
5
6
7
8
9
10
A C D G E B

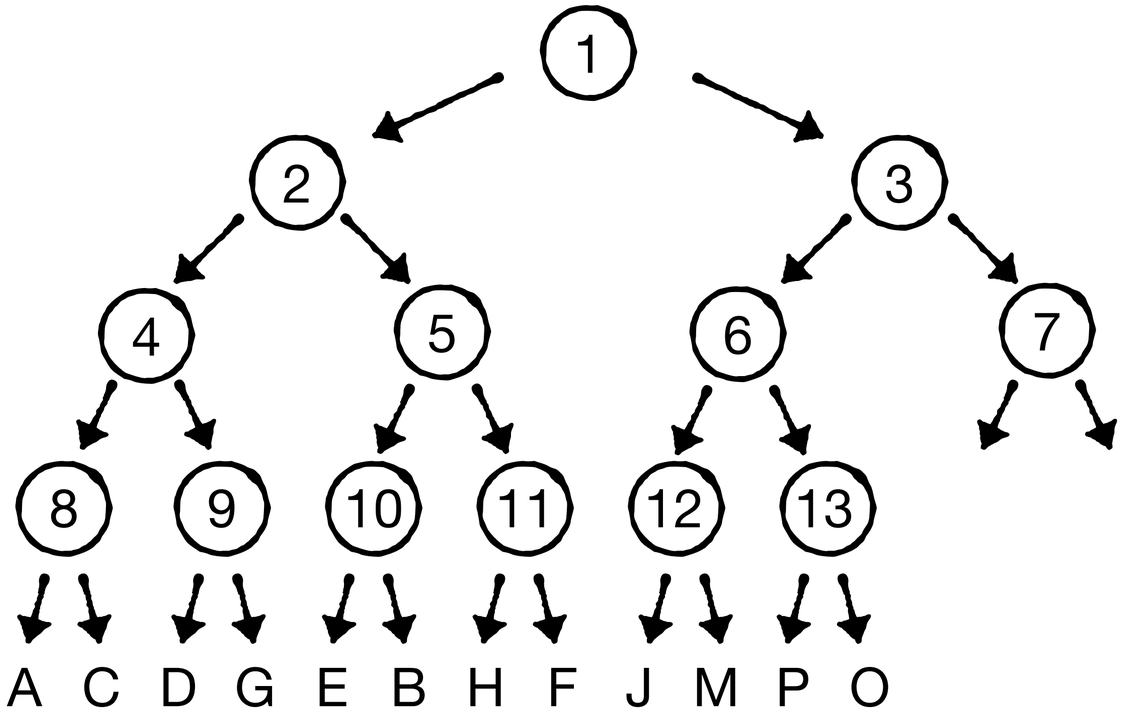

8
2
10
A
C
B
4
D
E
5
9
F
G
11
1
H
I
J
K
7
L
M
6
N
O
P
3

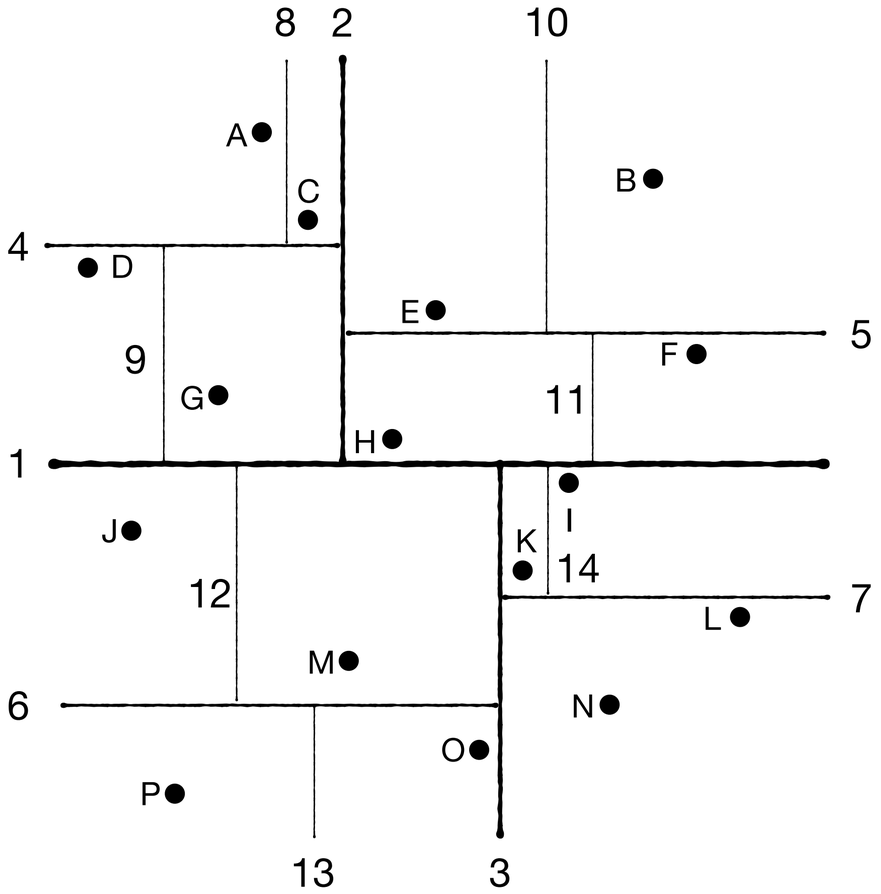

1
2
3
4
5
6
7
8
9
10
11
A C D G E B H F

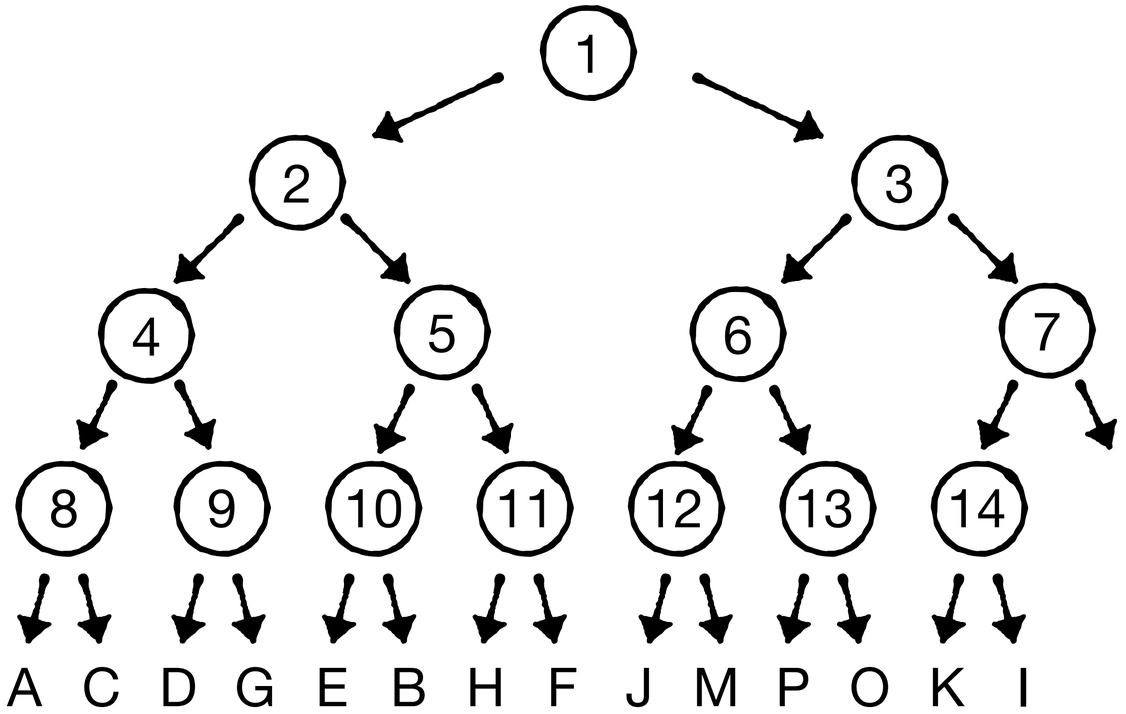

8
2
10
A
C
B
4
D
E
5
9
F
G
11
H
1
I
J
K
12
7
L
M
6
N
O
P
3

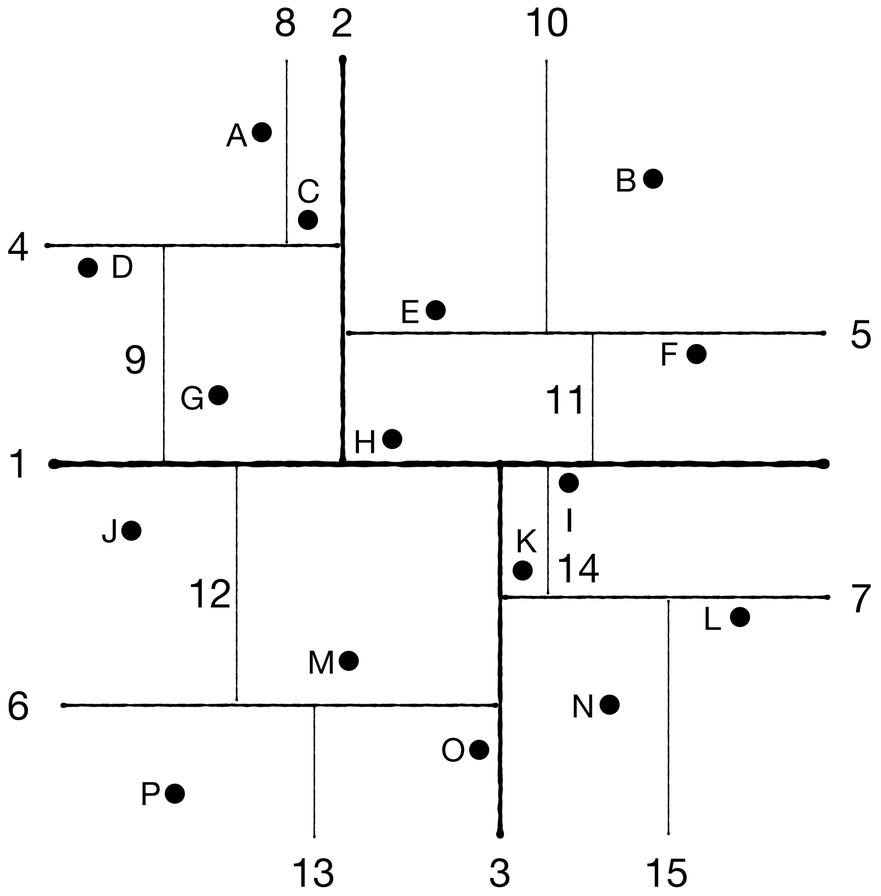

1
2
3
4
5
6
7
8
9
10
11
12
A C D G E B H F J M

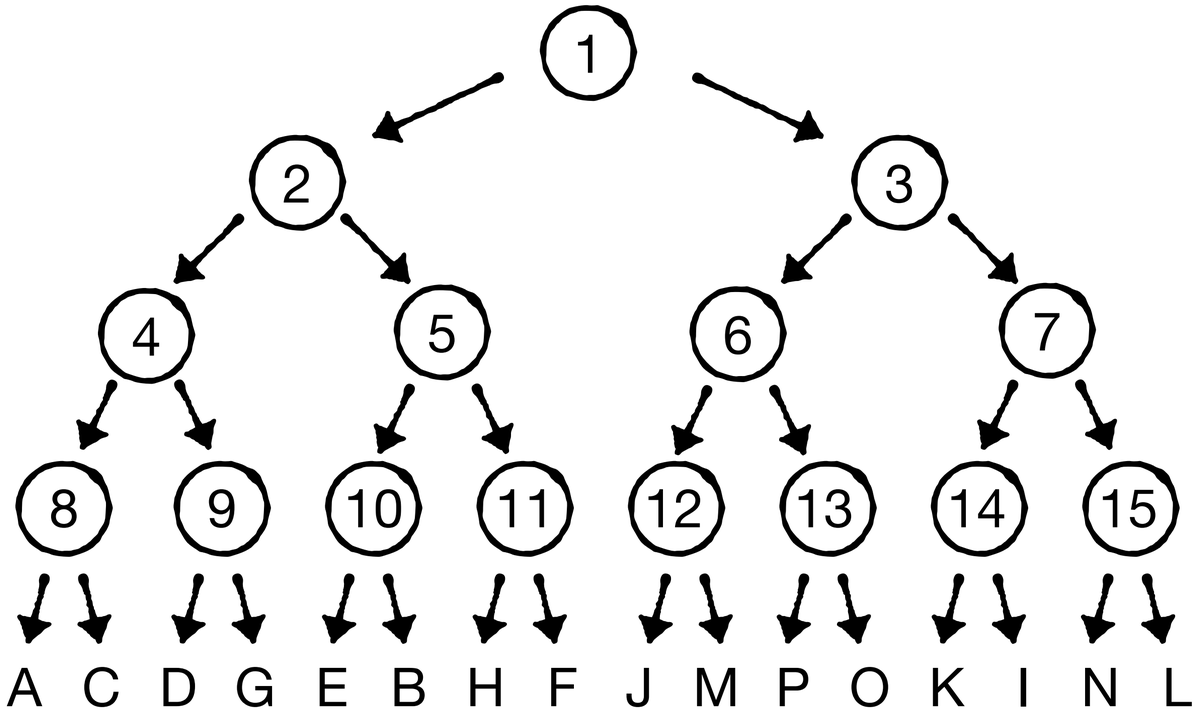

8
2
10
A
C
B
4
D
E
5
9
F
G
11
H
1
I
J
K
12
7
L
M
6
N
O
P
13
3


1
2
3
4
5
6
7
8
9
10
11
12
13
A C D G E B H F J M P O

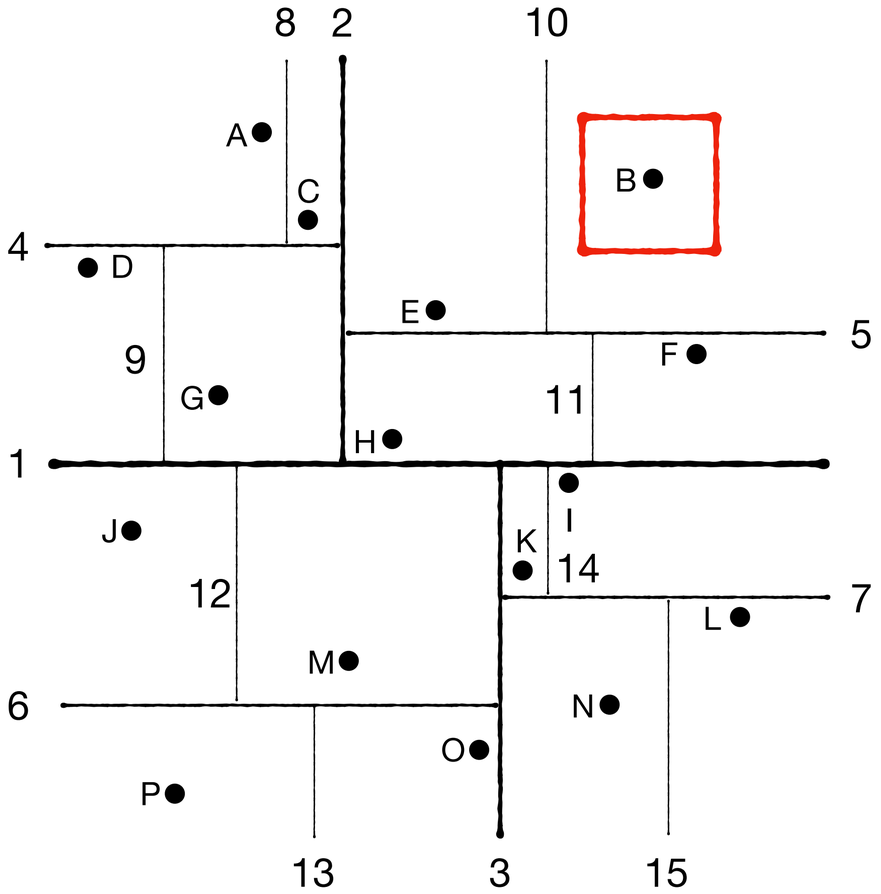

8
2
10
A
C
B
4
D
E
5
9
F
G
11
H
1
J
I
K
14
12
7
L
M
6
N
O
P
13
3

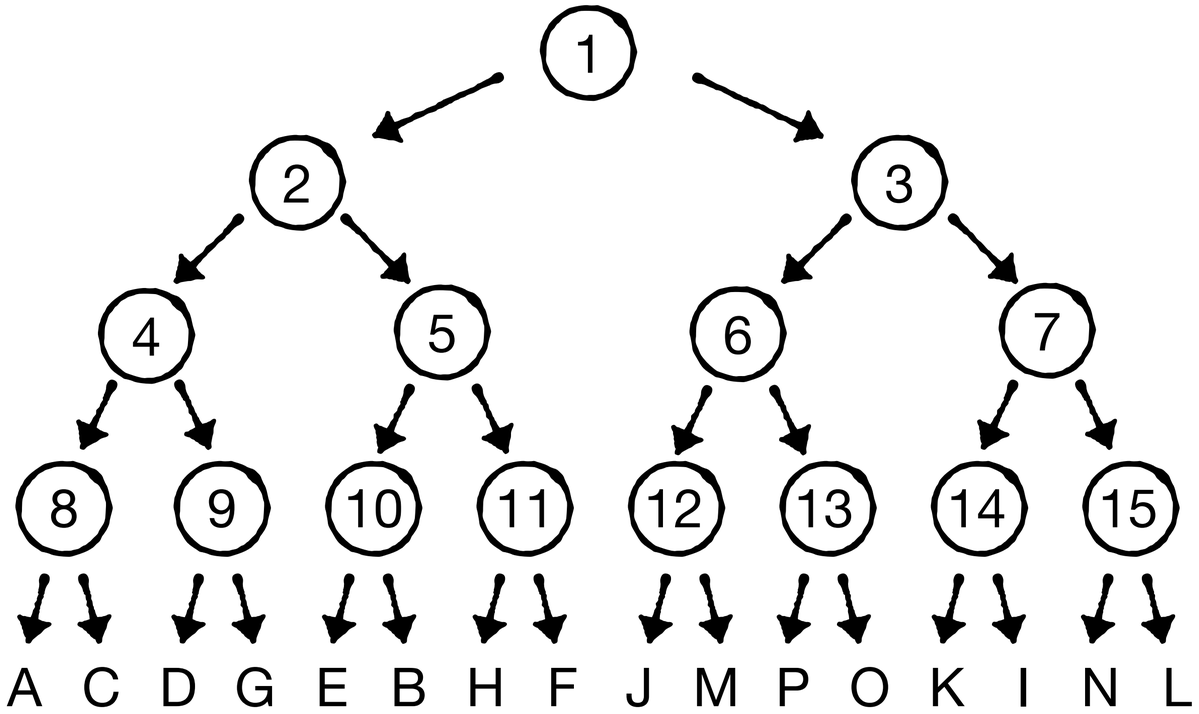

1
2
3
4
5
6
7
8
9
10
11
12
13
14
A C D G E B H F J M P O K I


8
2
10
A
C
B
4
D
E
5
9
F
G
11
H
1
J
I
K
14
12
7
L
M
6
N
O
P
13
3
15

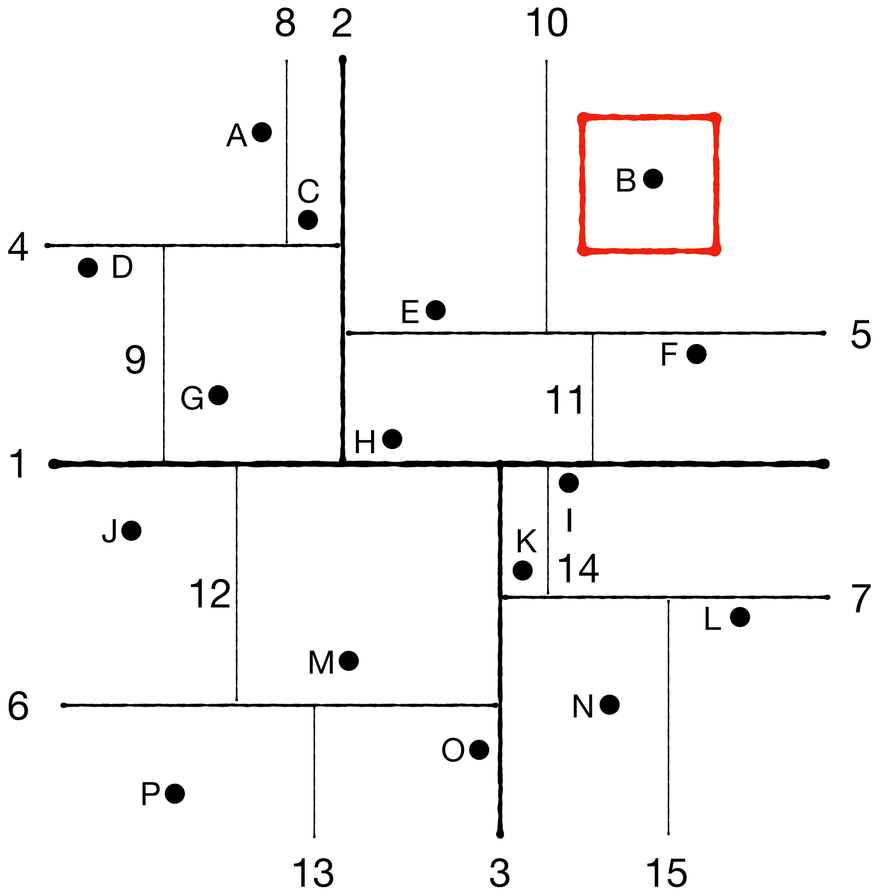

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
A C D G E B H F J M P O K I N L

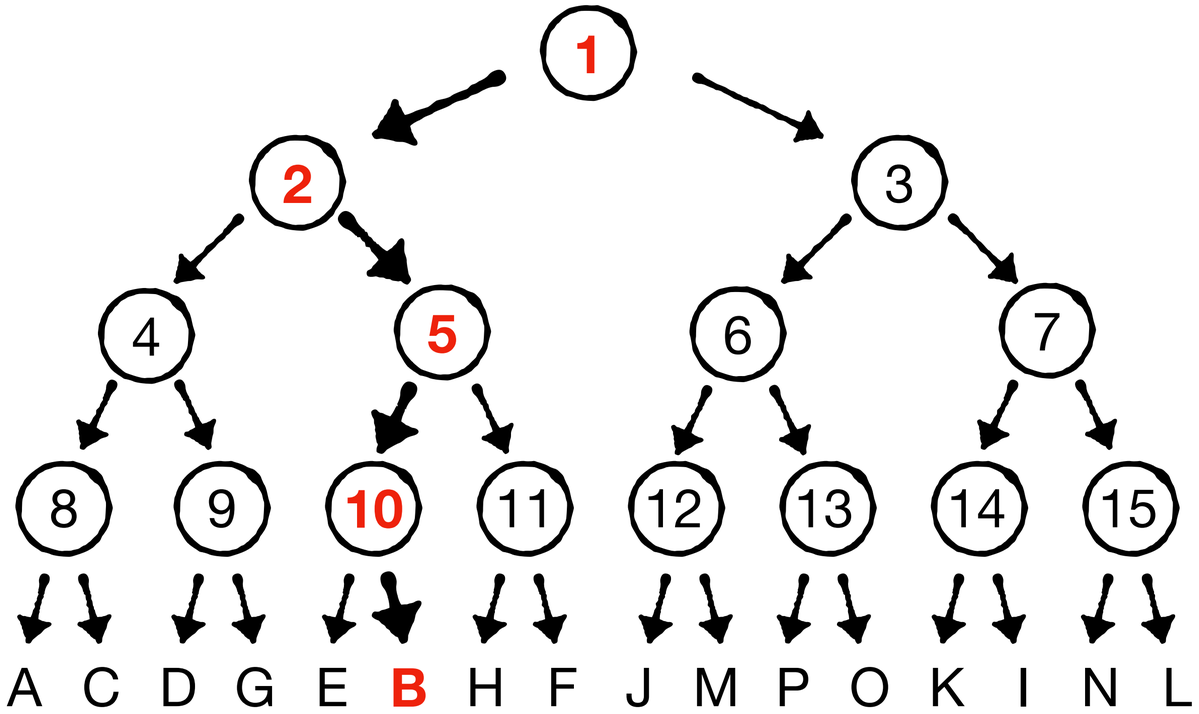
How to process a range query?





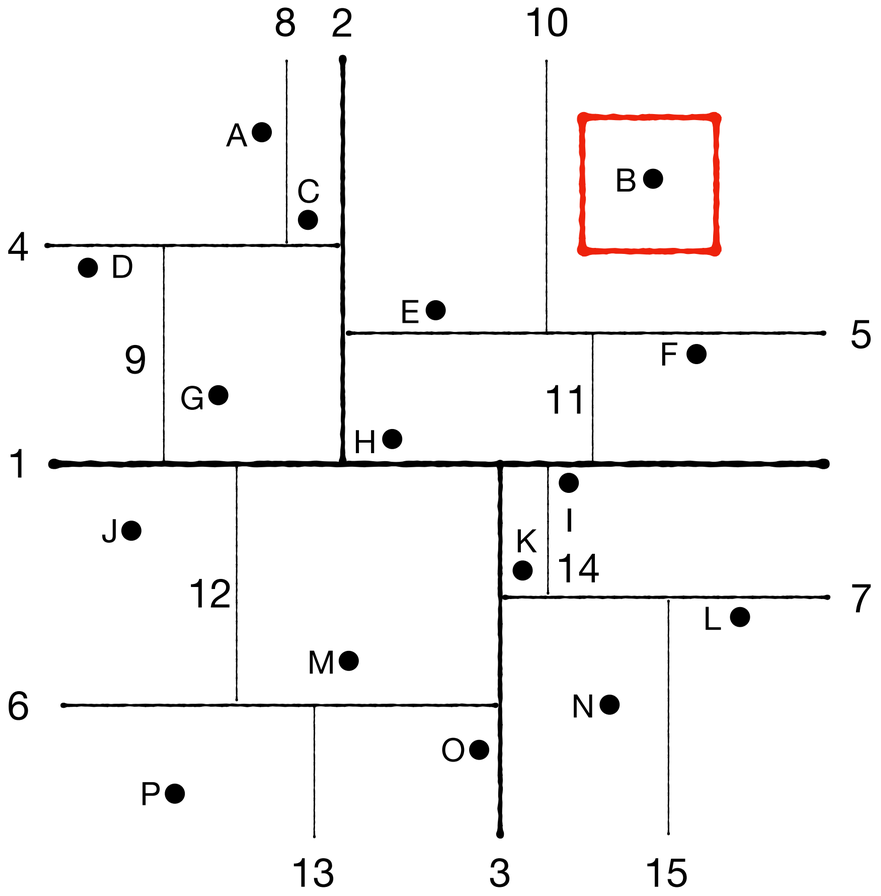
How to process a range query?

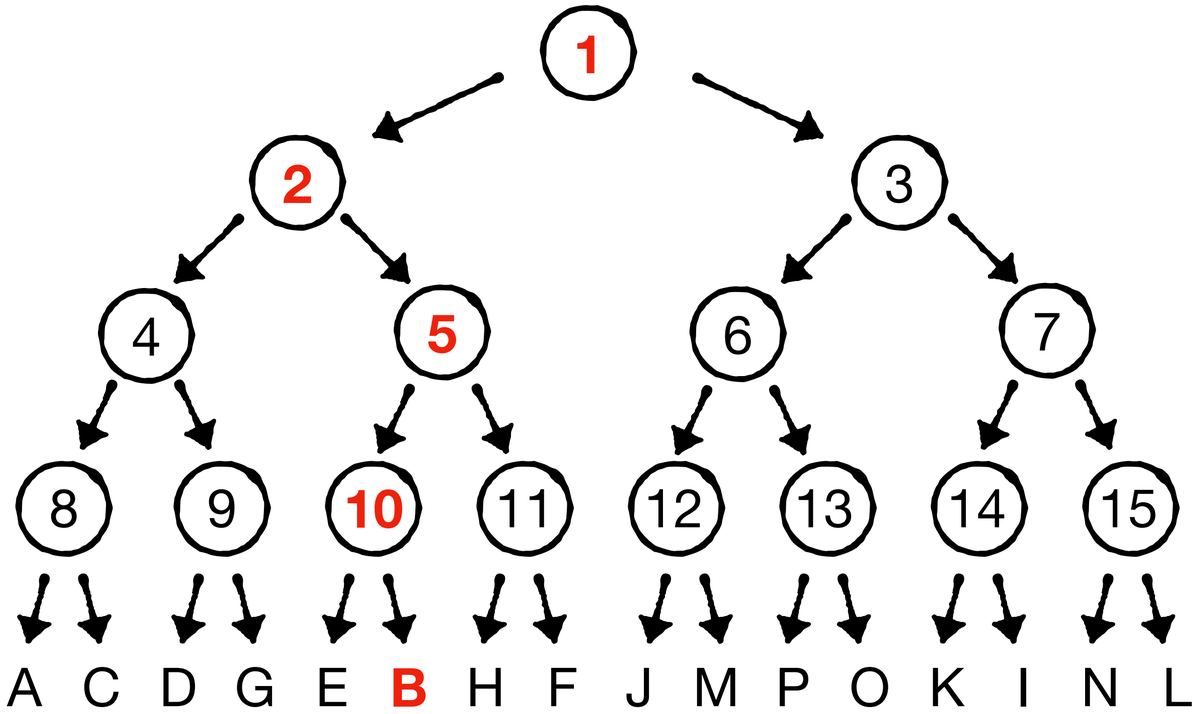




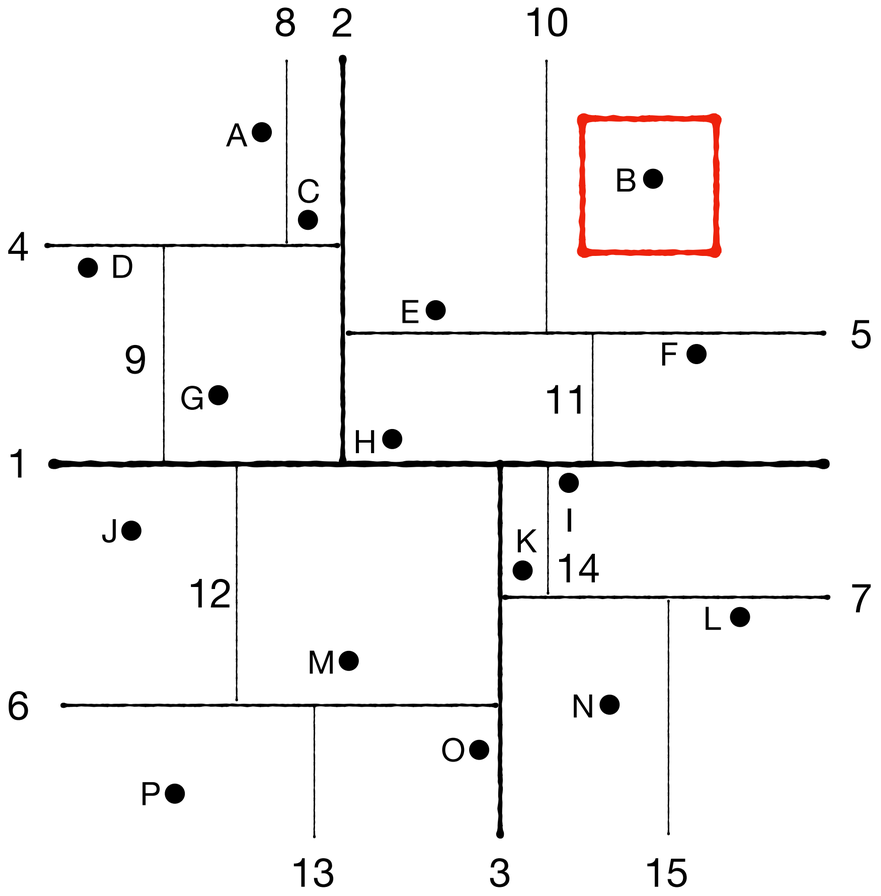
Axis Parallel Range query

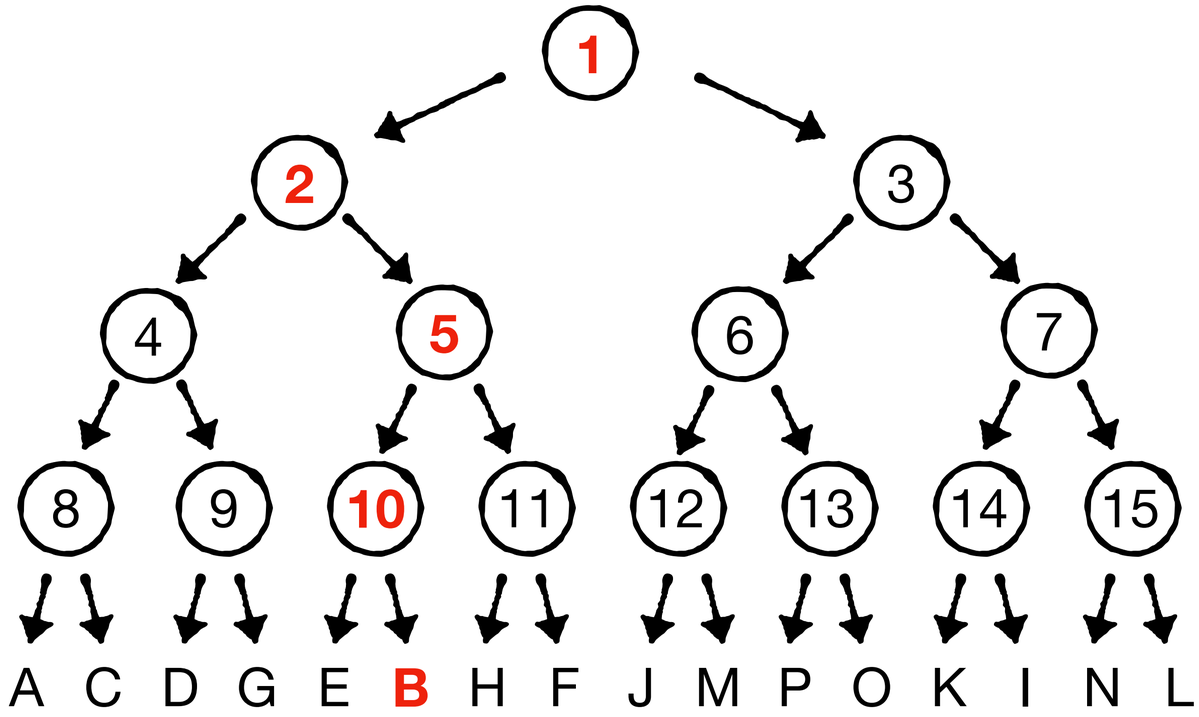
Cost?





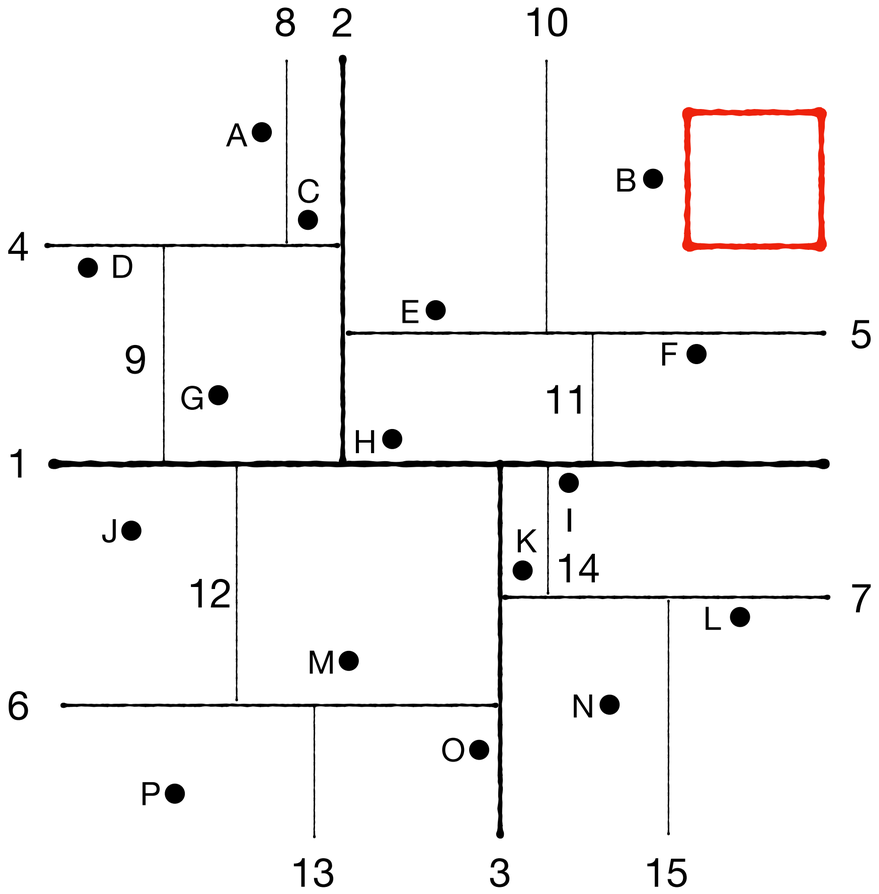
Cost? O(log N) If query is contained in one region

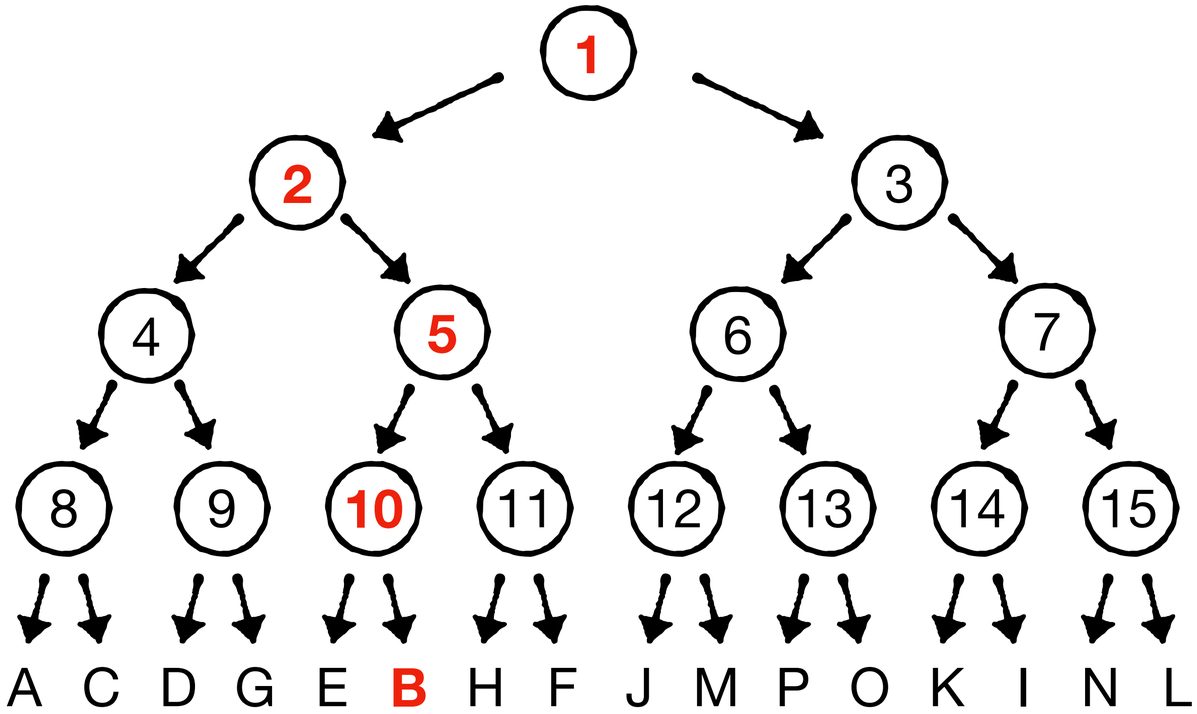




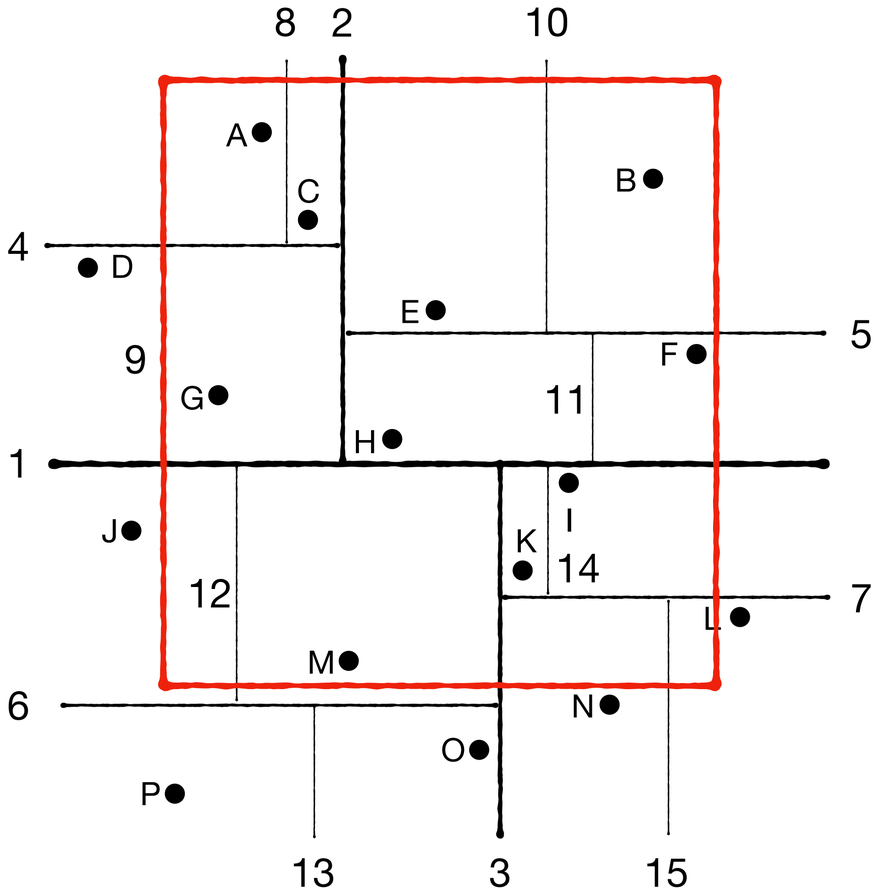
Cost? O(log N) If query is contained in one region

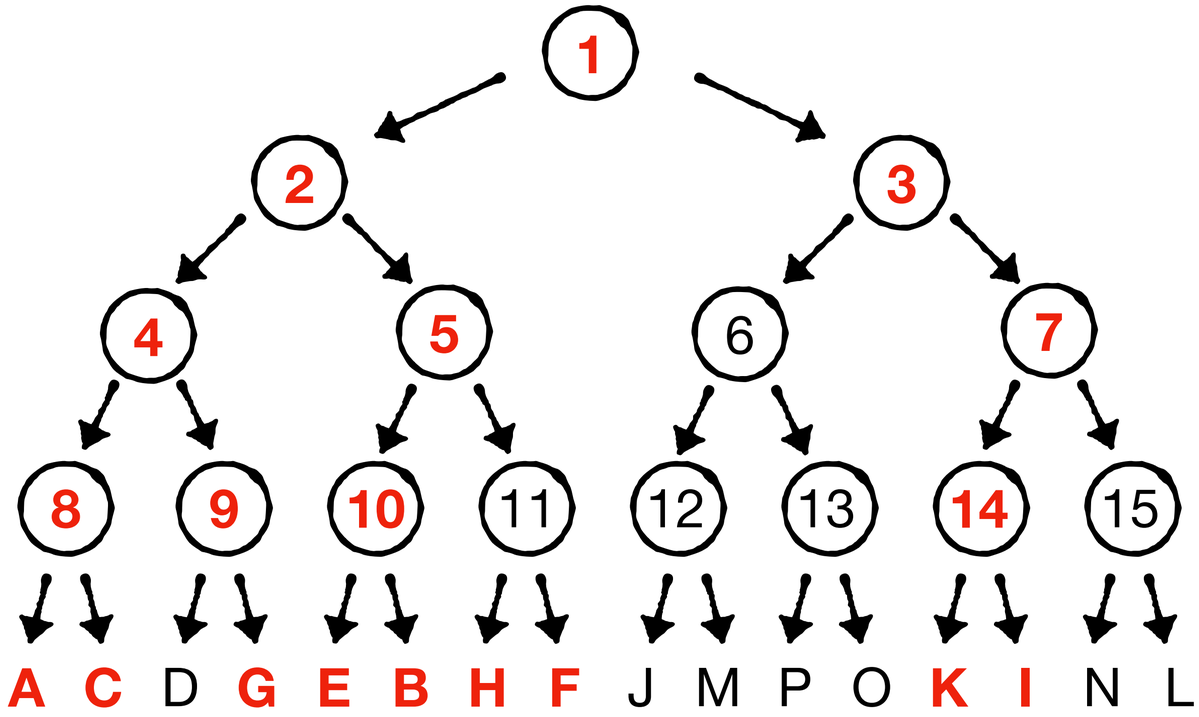
Also applies for empty range query





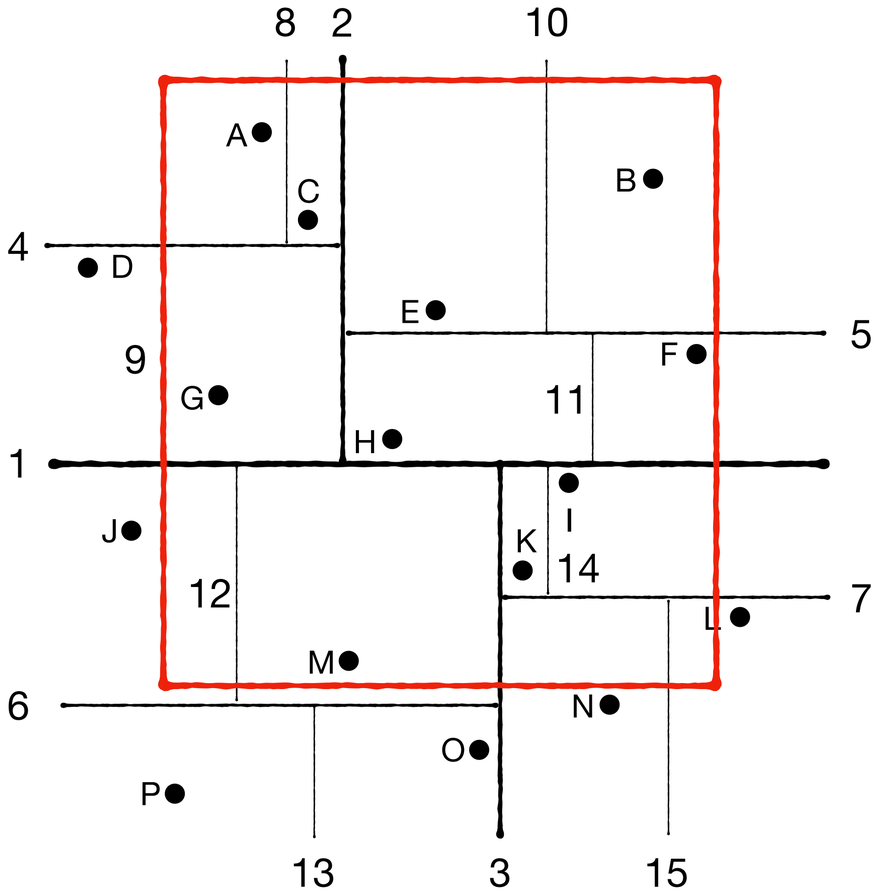
How about a large query?

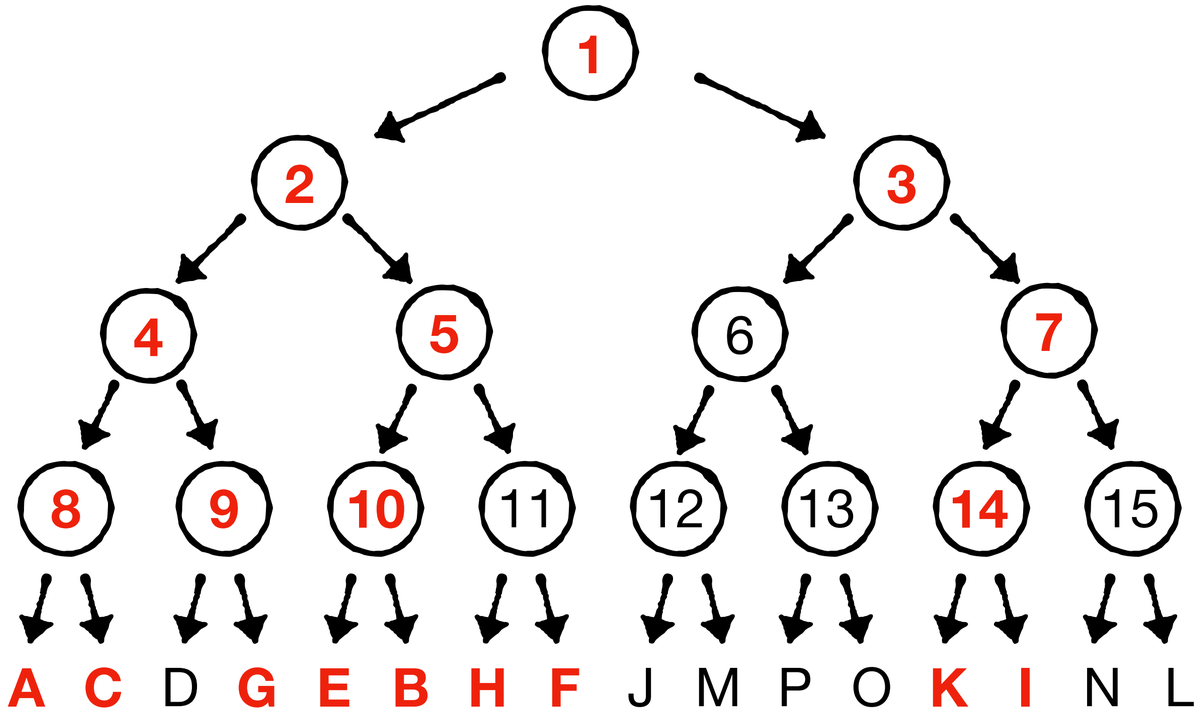




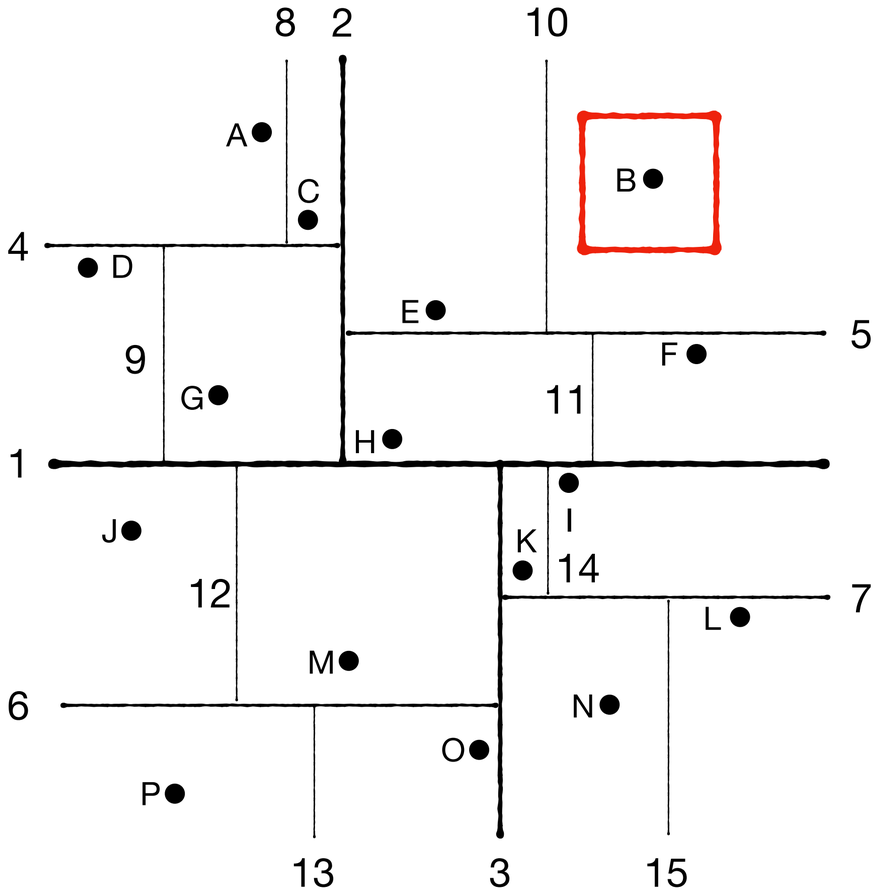
How about a large query?

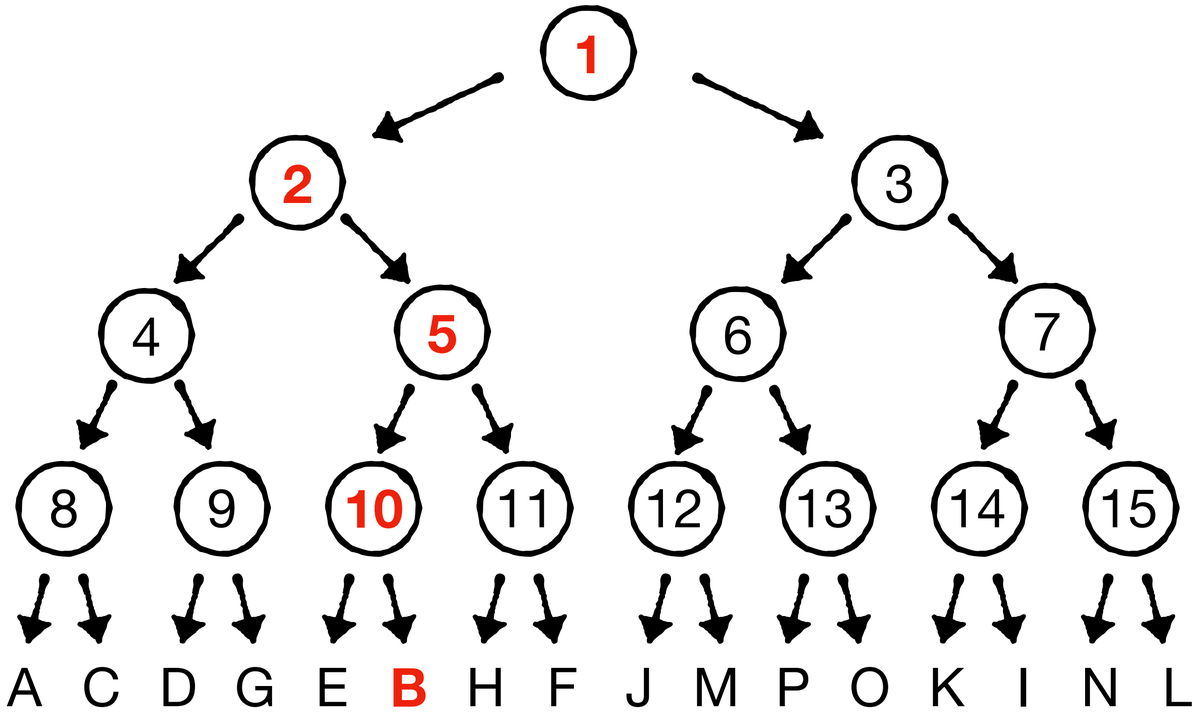
O(N)





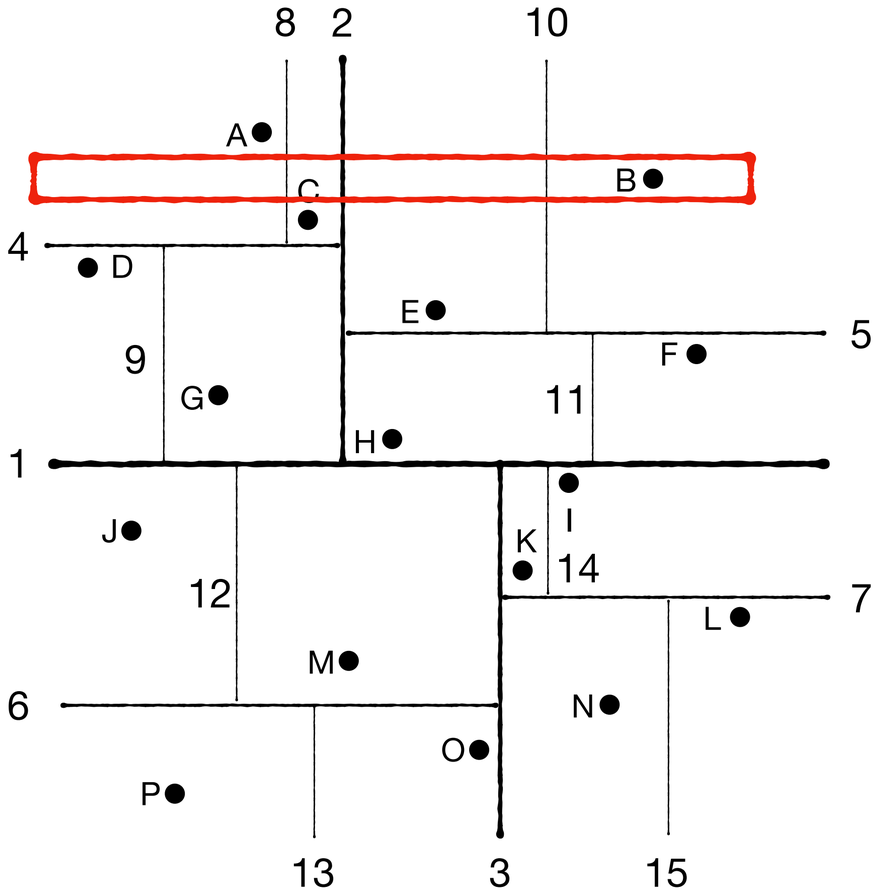
Worst-case for short query? (i.e., one qualifying key)

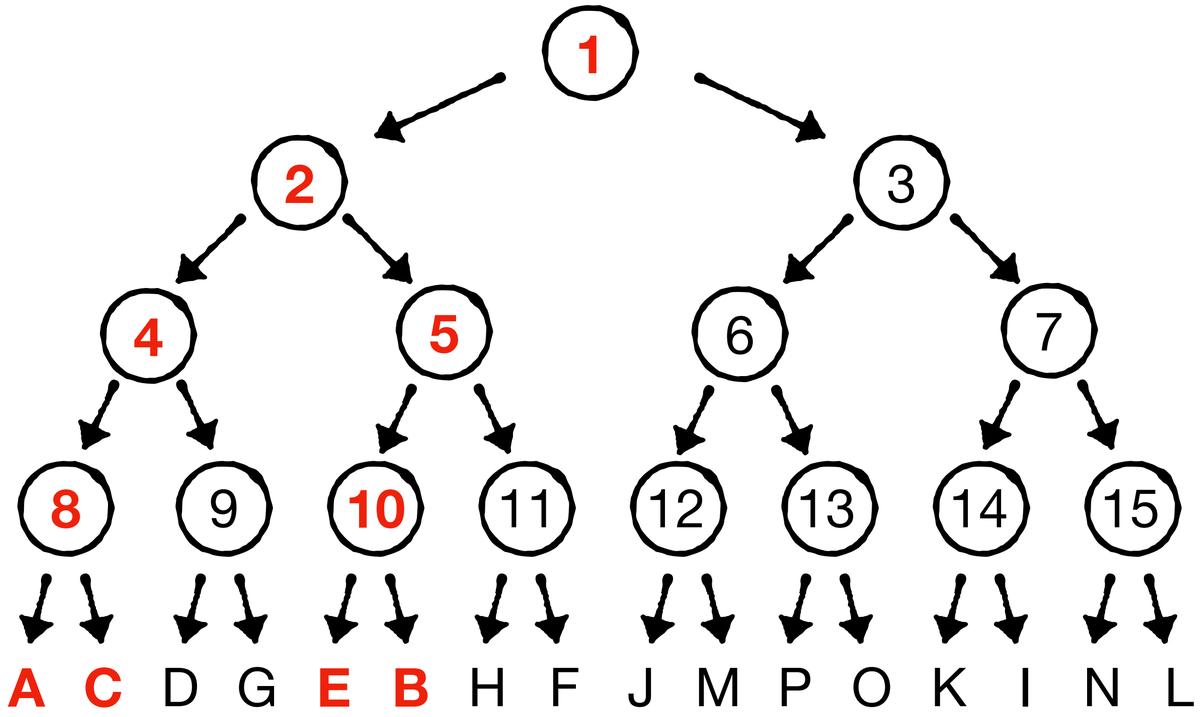




Worst-case for short query? (i.e., one qualifying key)

Maximize intersections across regions

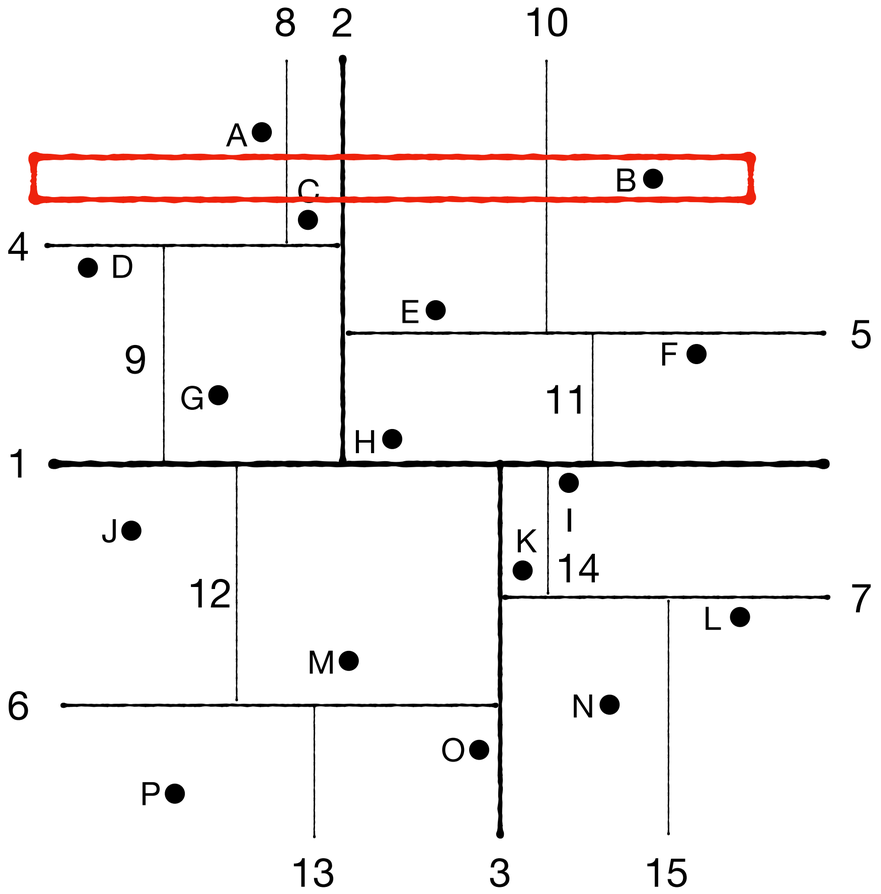


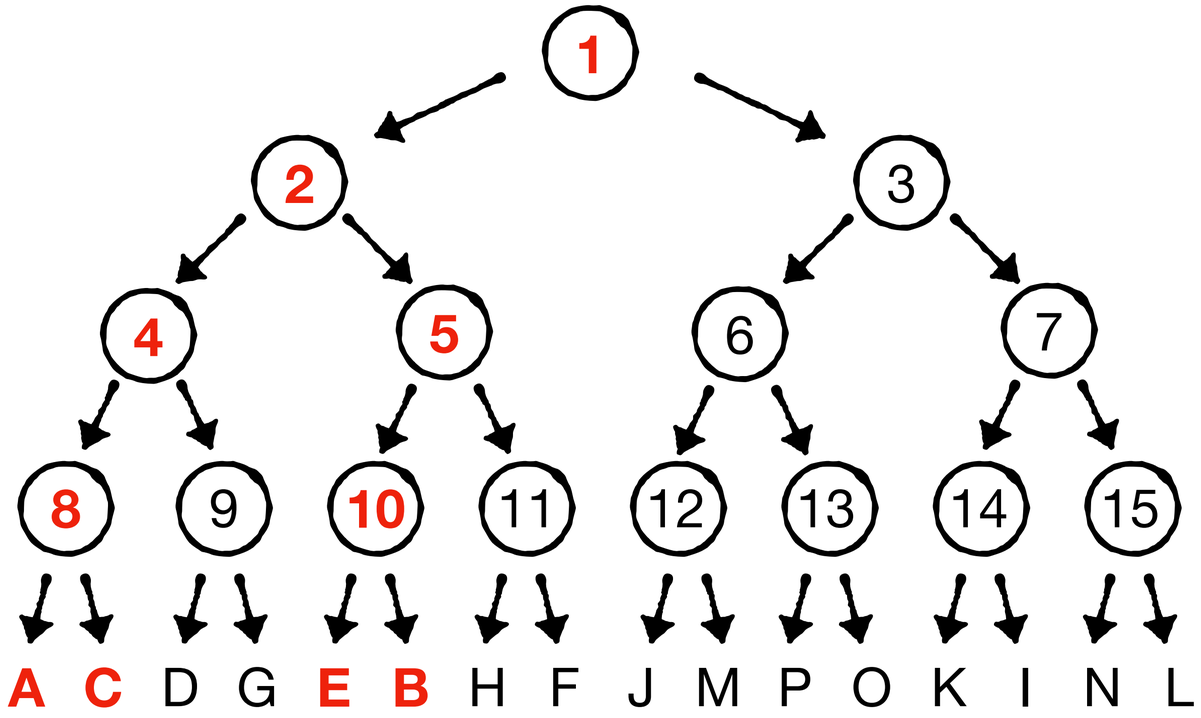


Worst-case for short query? (i.e., one qualifying key)

Maximize intersections across regions

Cost?

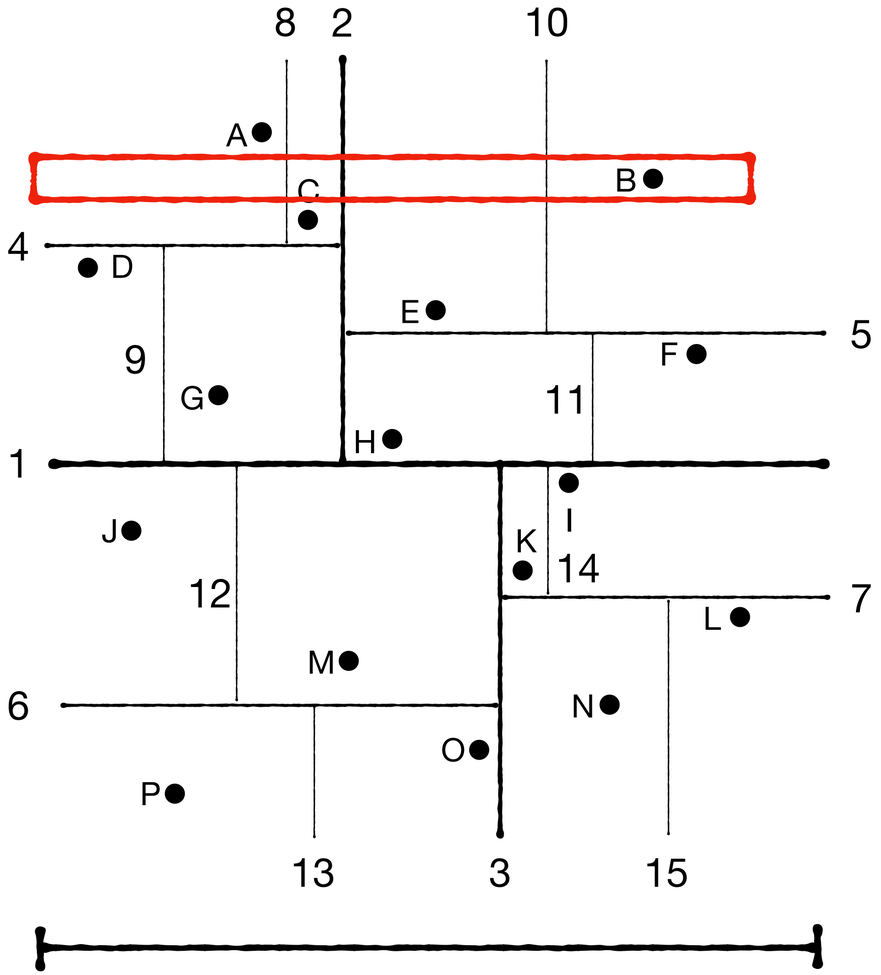




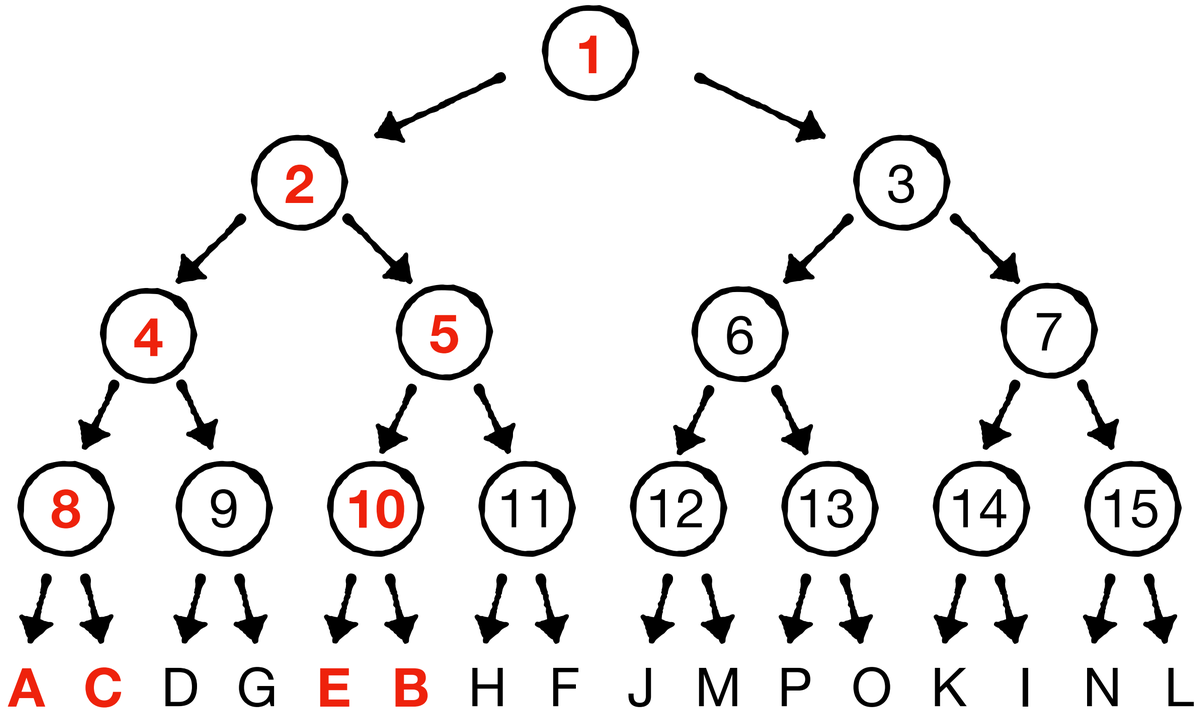
Worst-case for short query? (i.e., one qualifying key)

Maximize intersections across regions

Cost?



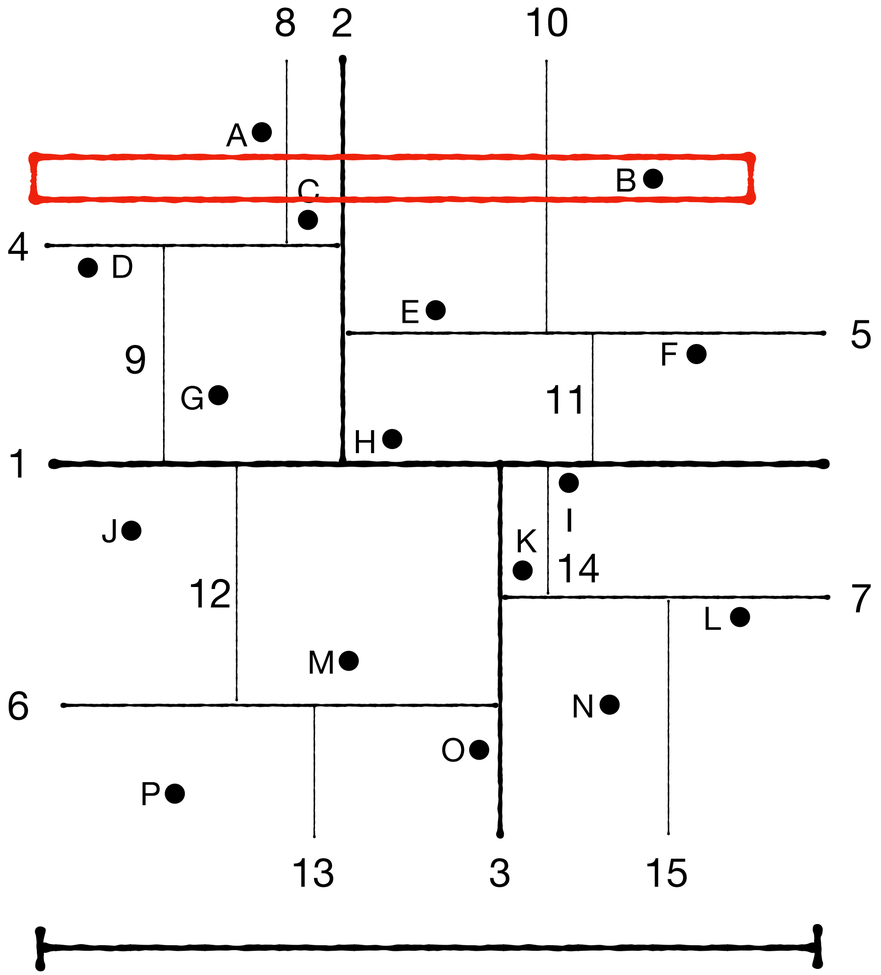
Max regions along one dimension?

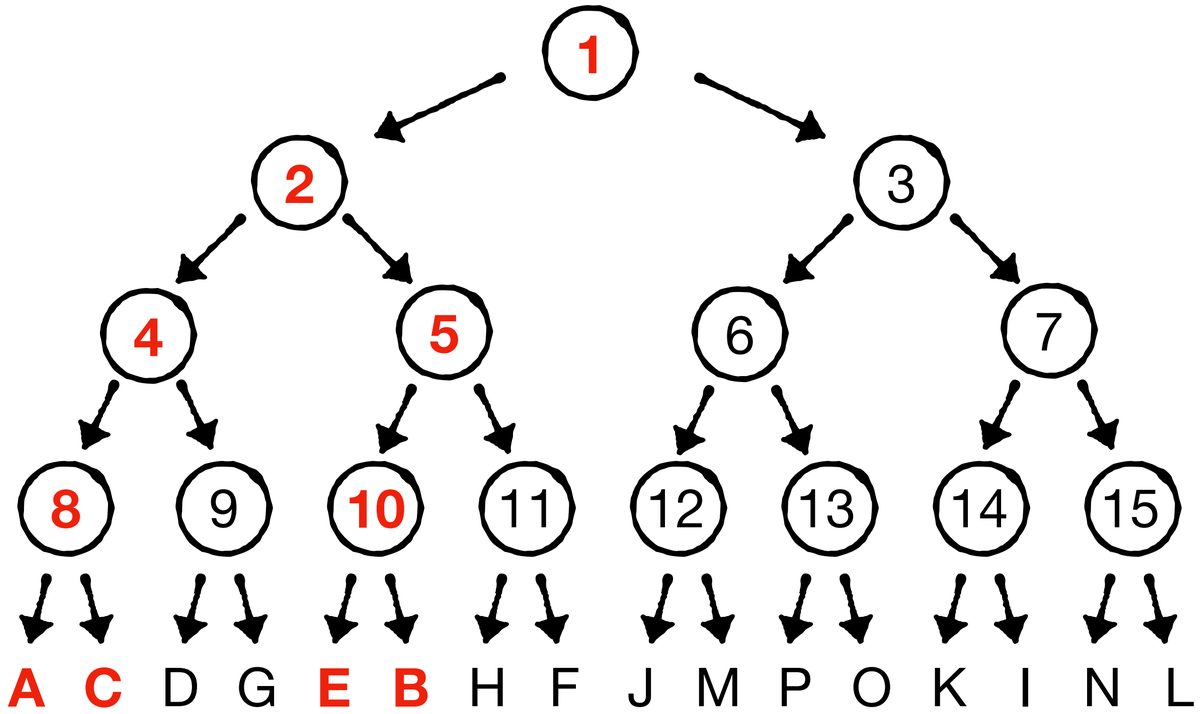


Worst-case for short query? (i.e., one qualifying key)

Maximize intersections across regions

Cost?



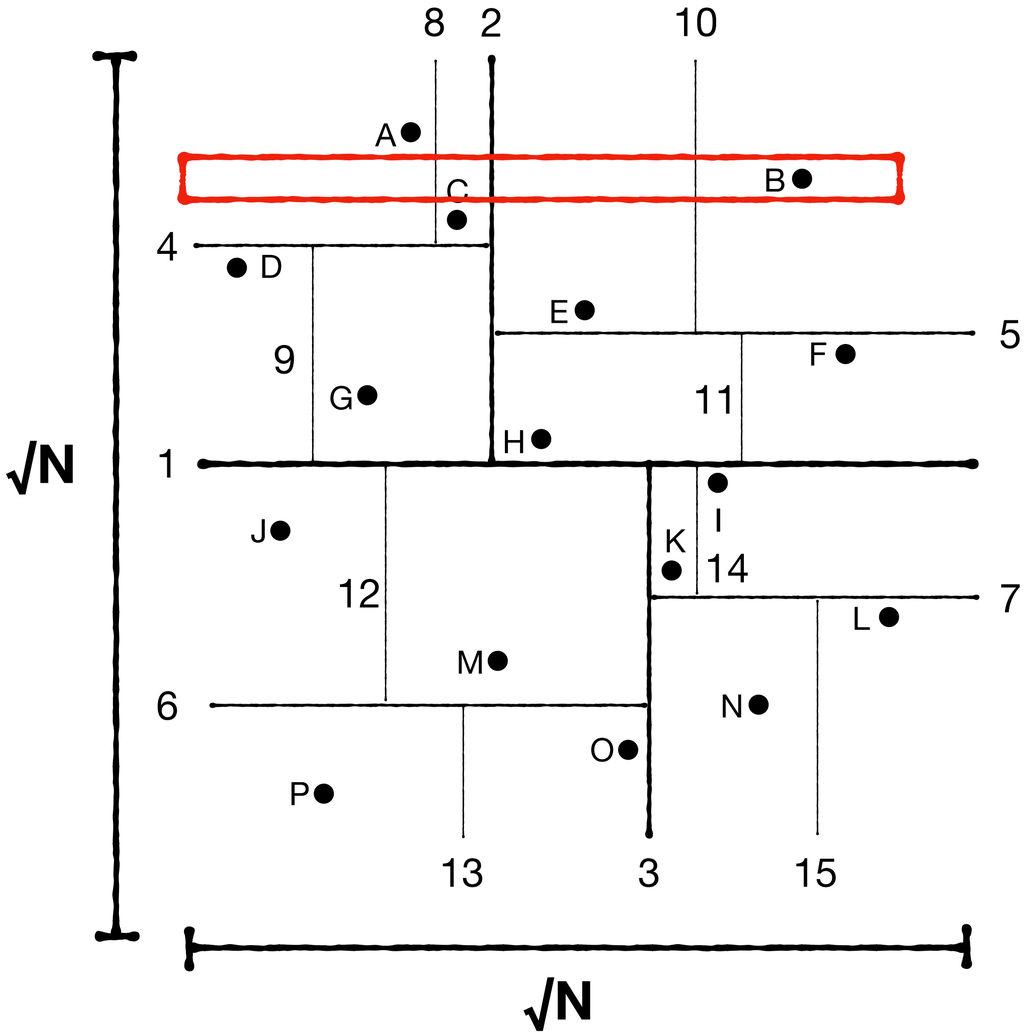


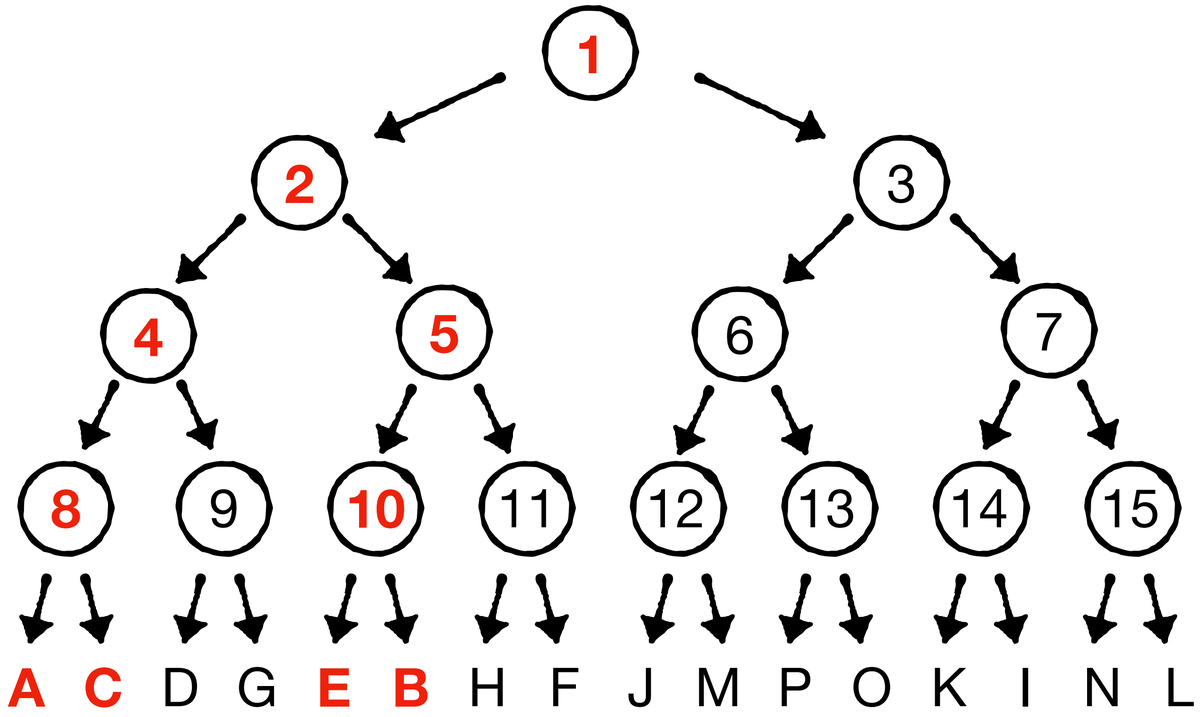
Max regions along one dimension? √N

Worst-case for short query? (i.e., one qualifying key)

Maximize intersections across regions

Cost?

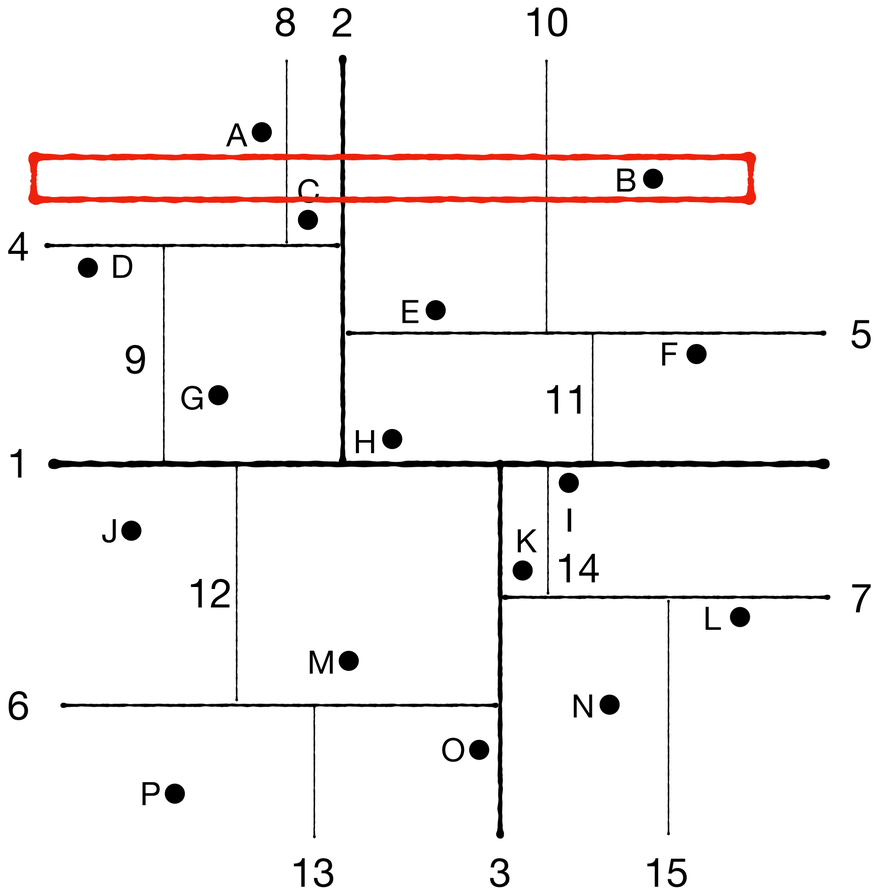


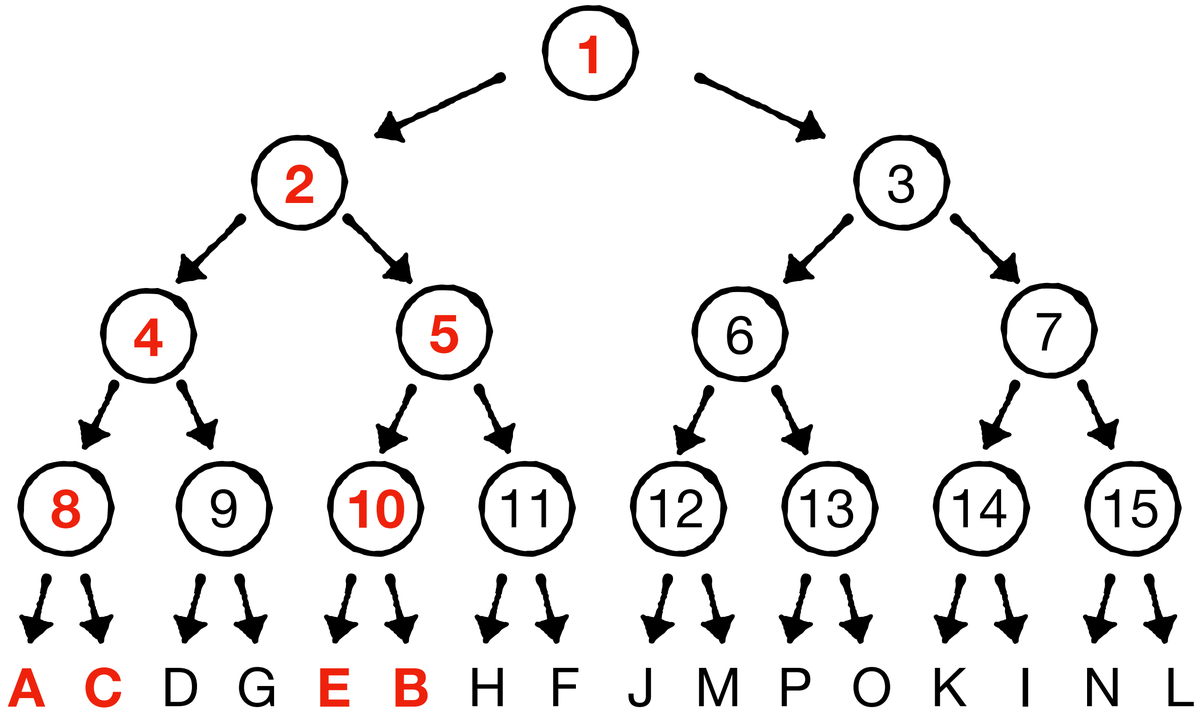


Worst-case for short query? (i.e., one qualifying key)

Maximize intersections across regions

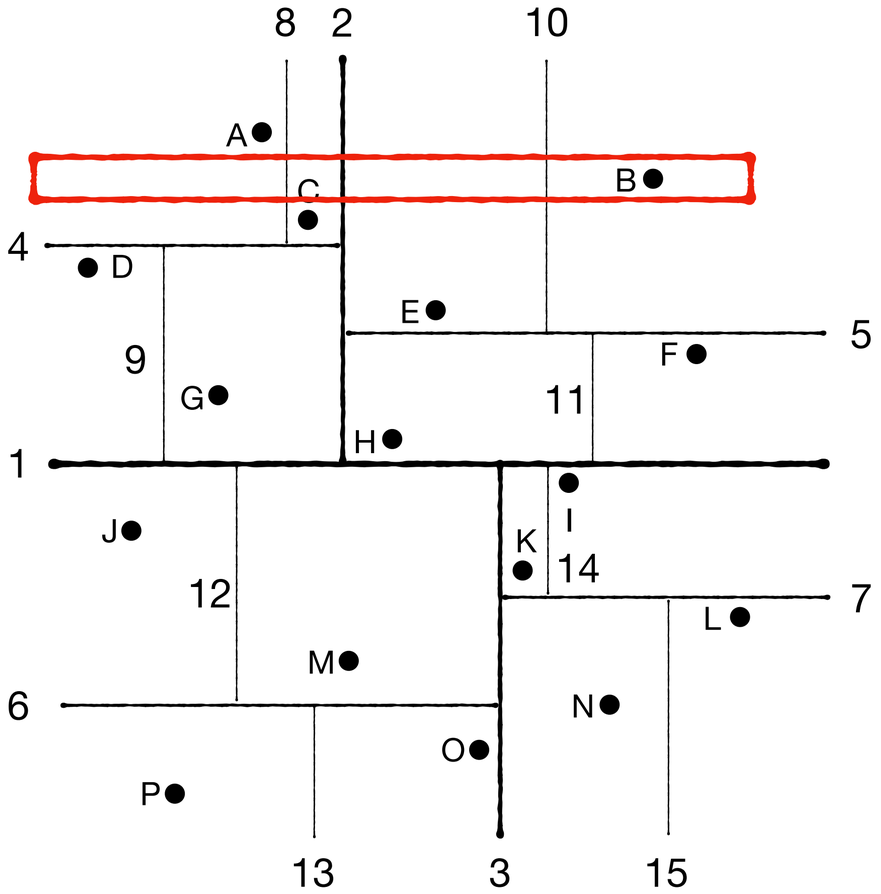
Cost? O(√N)

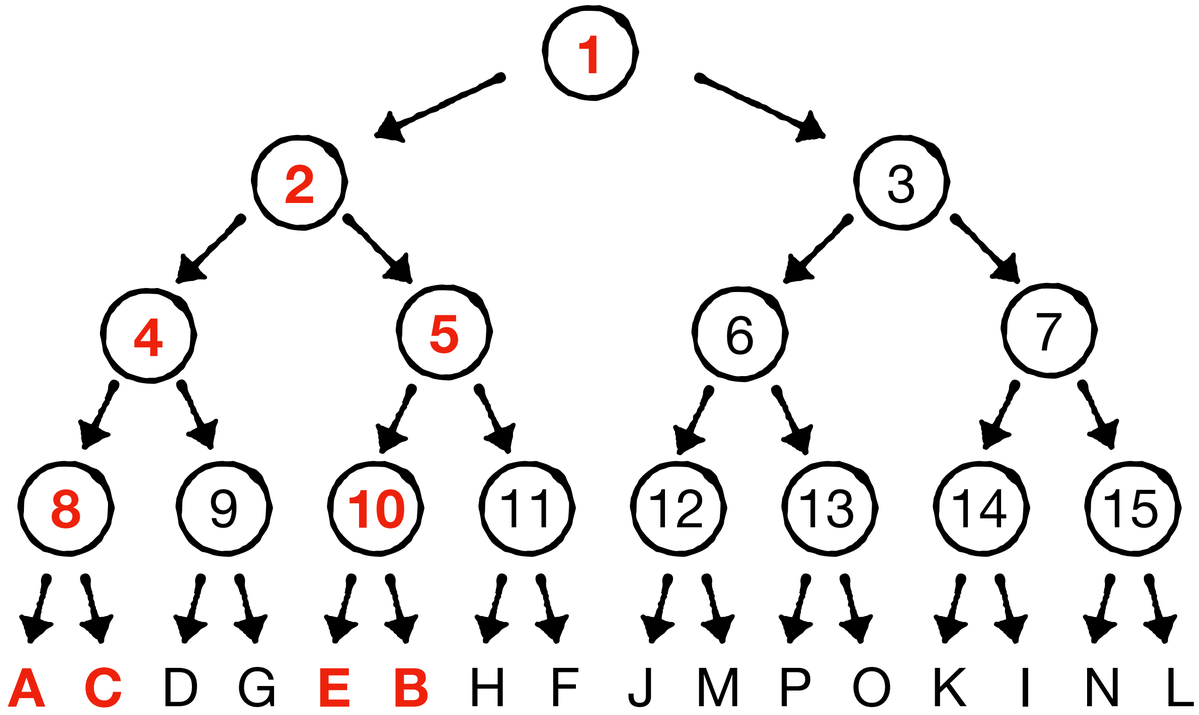




Cost? O(√N)

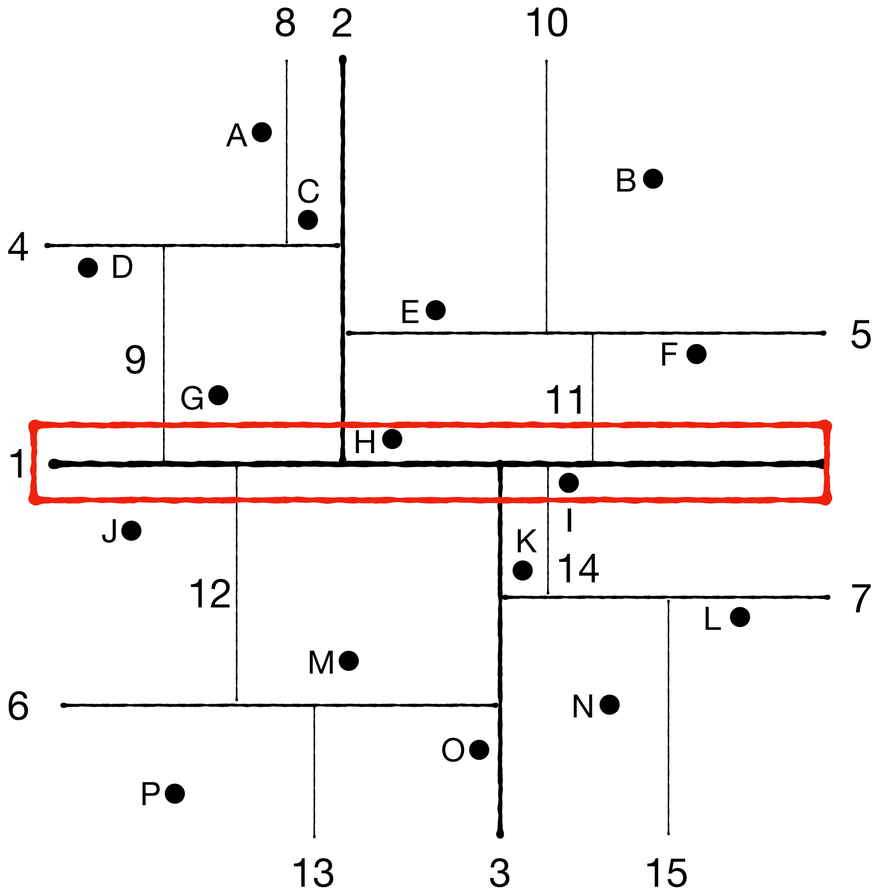
Can we make this even worse?

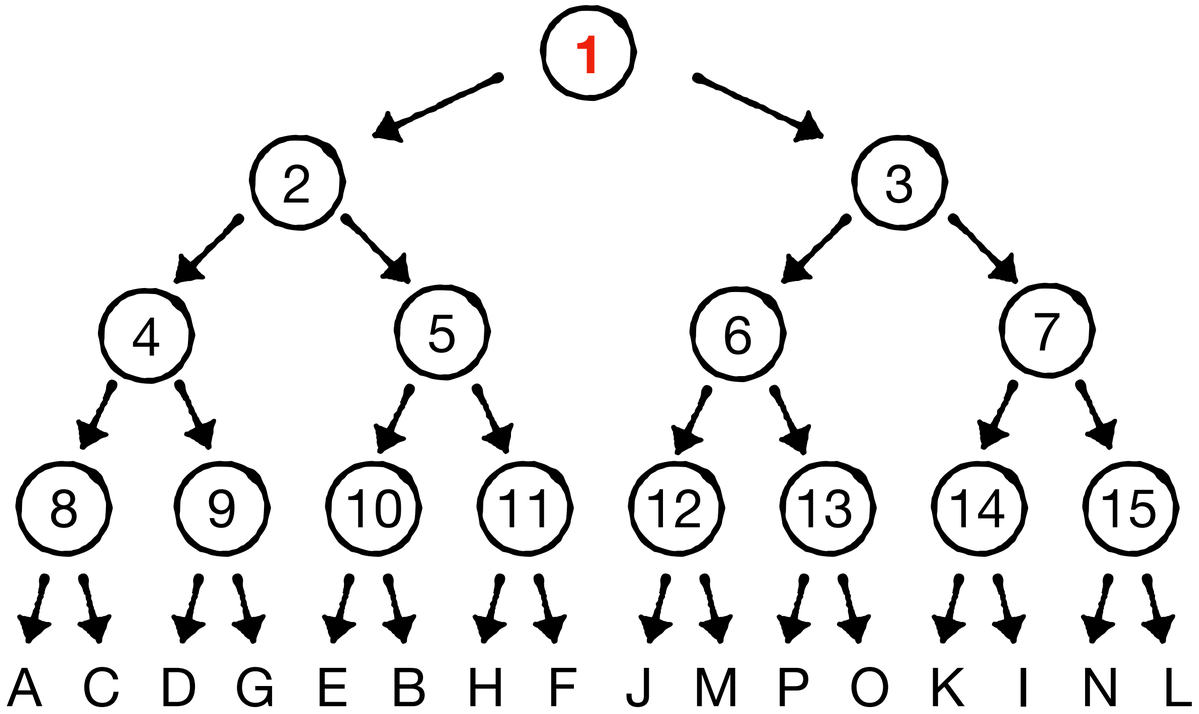




Cost? O(√N)

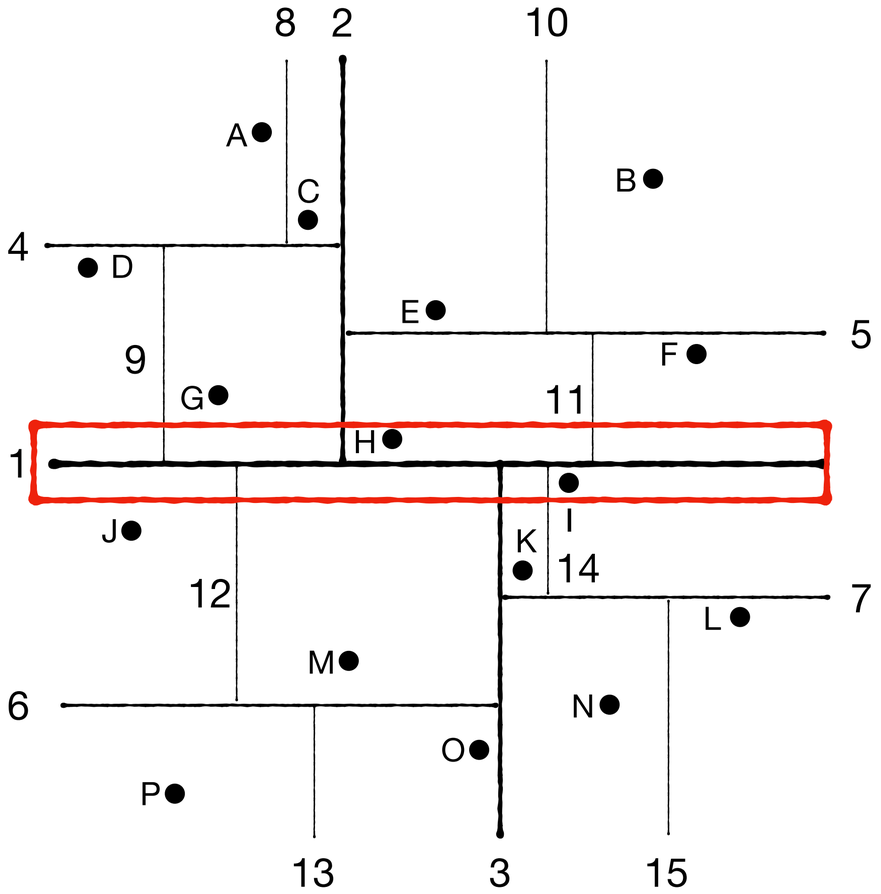
Can we make this even worse? **Intersect across dimensions**

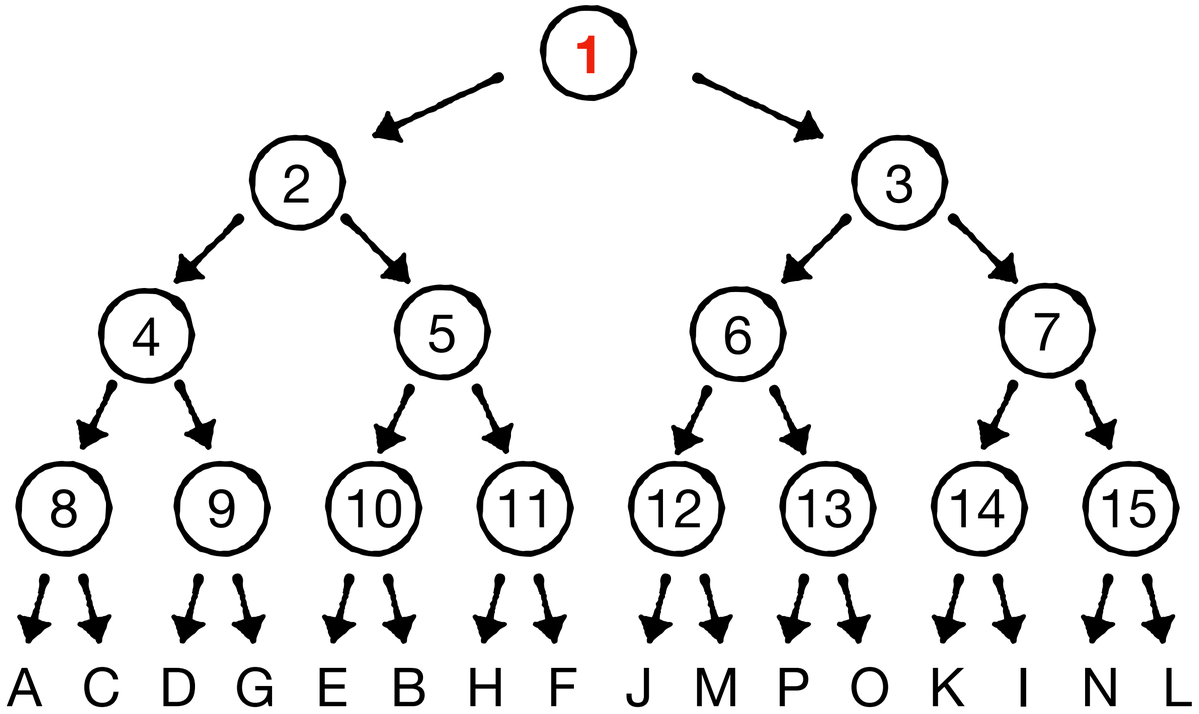




Cost? $O(\mathbf{2} \cdot \sqrt{N})$

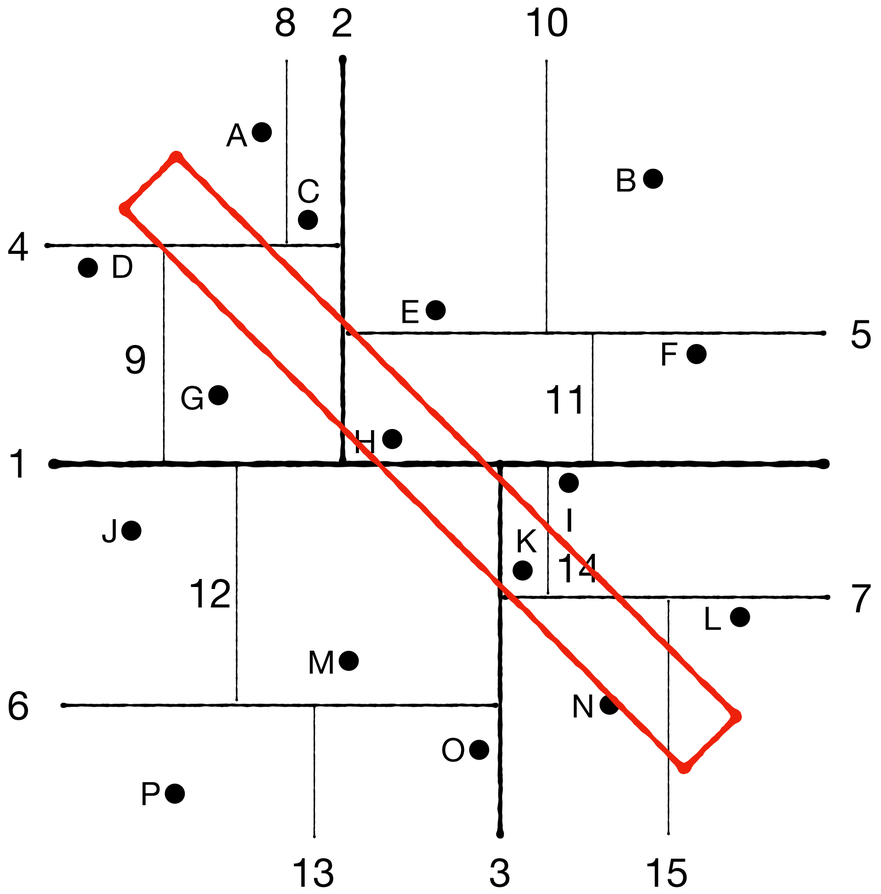
Can we make this even worse? Intersect across dimensions

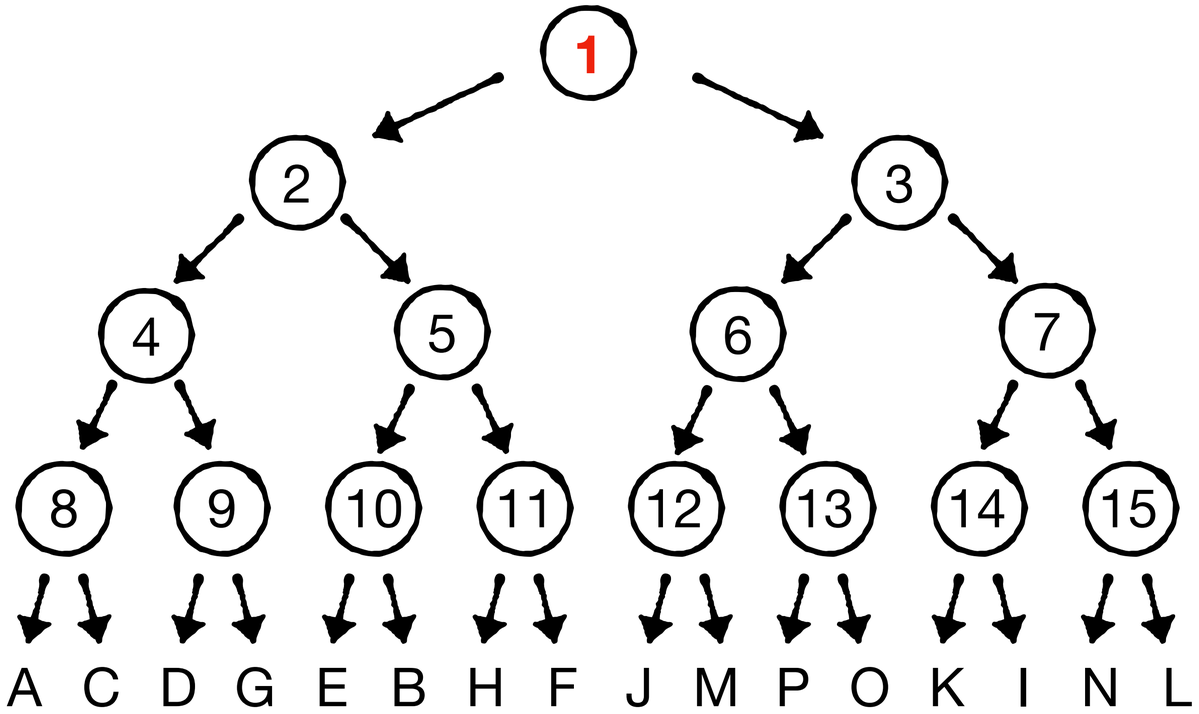




Cost? $O(\mathbf{2} \cdot \sqrt{N})$

:)

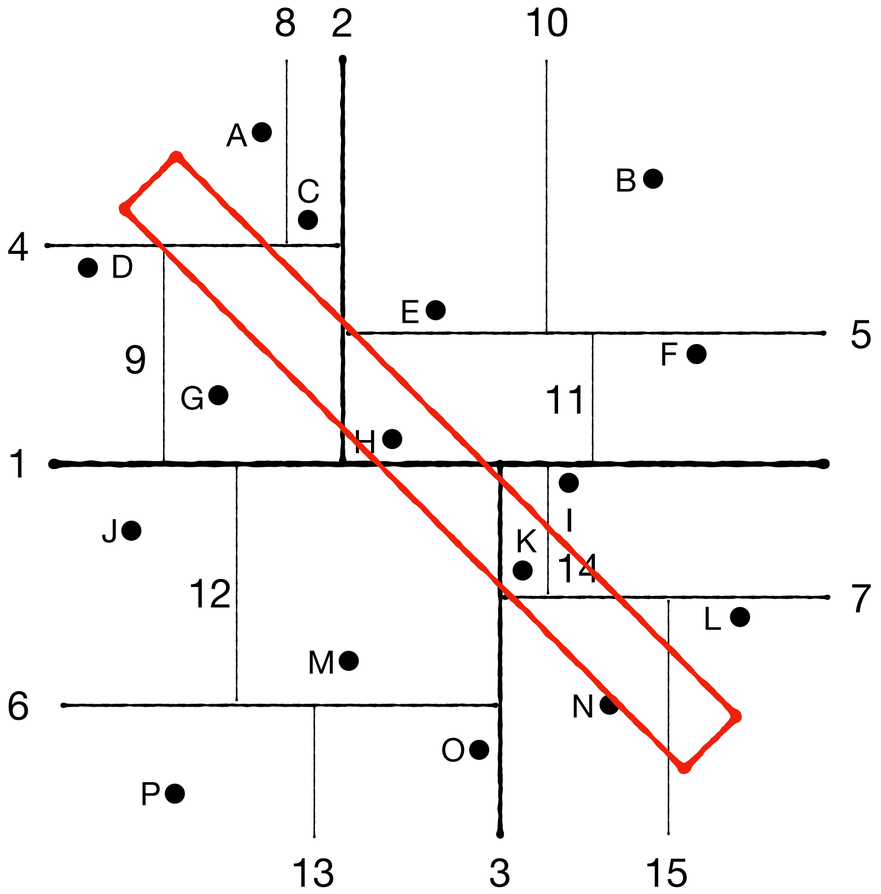


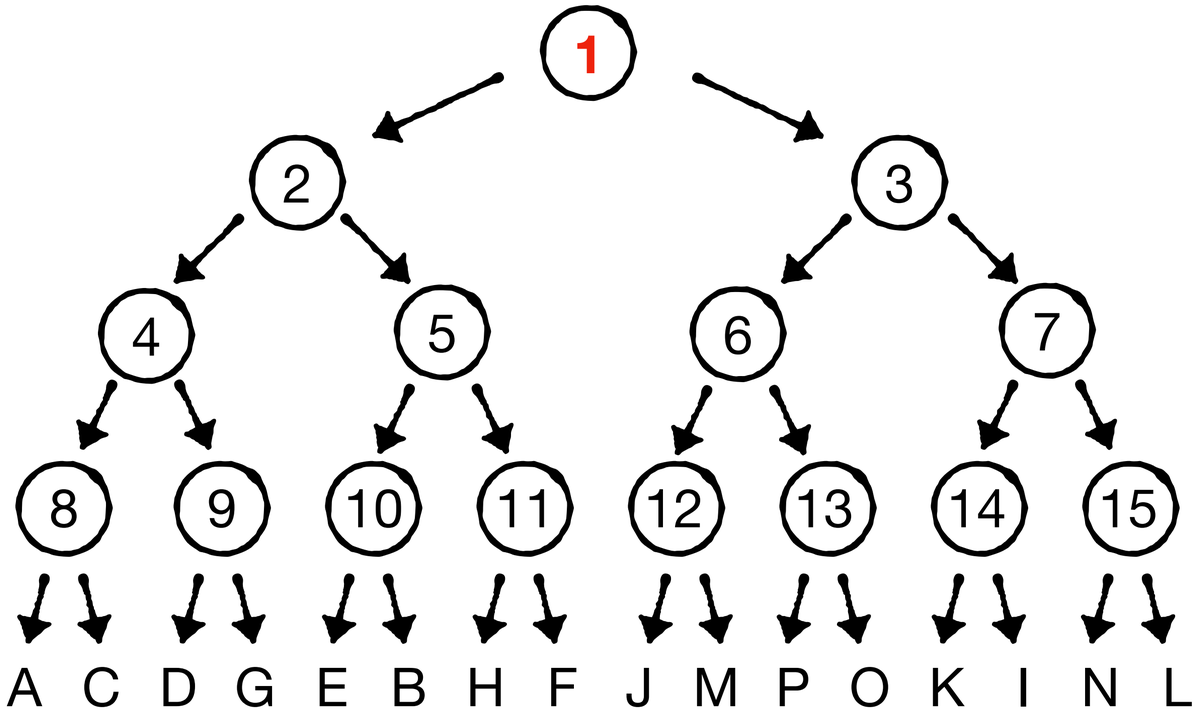


Cost? $O(\mathbf{2} \cdot \sqrt{N})$

:)

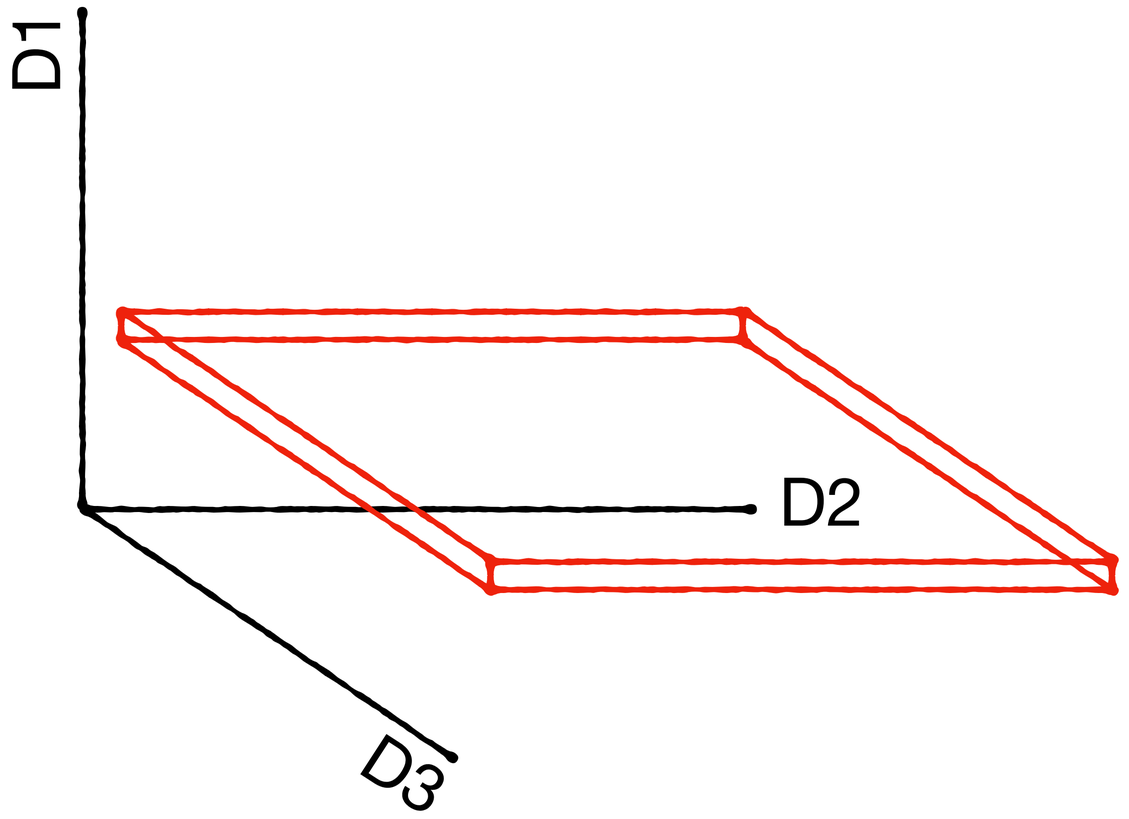
Maybe O(N)… but let's stick to being axis-aligned queries





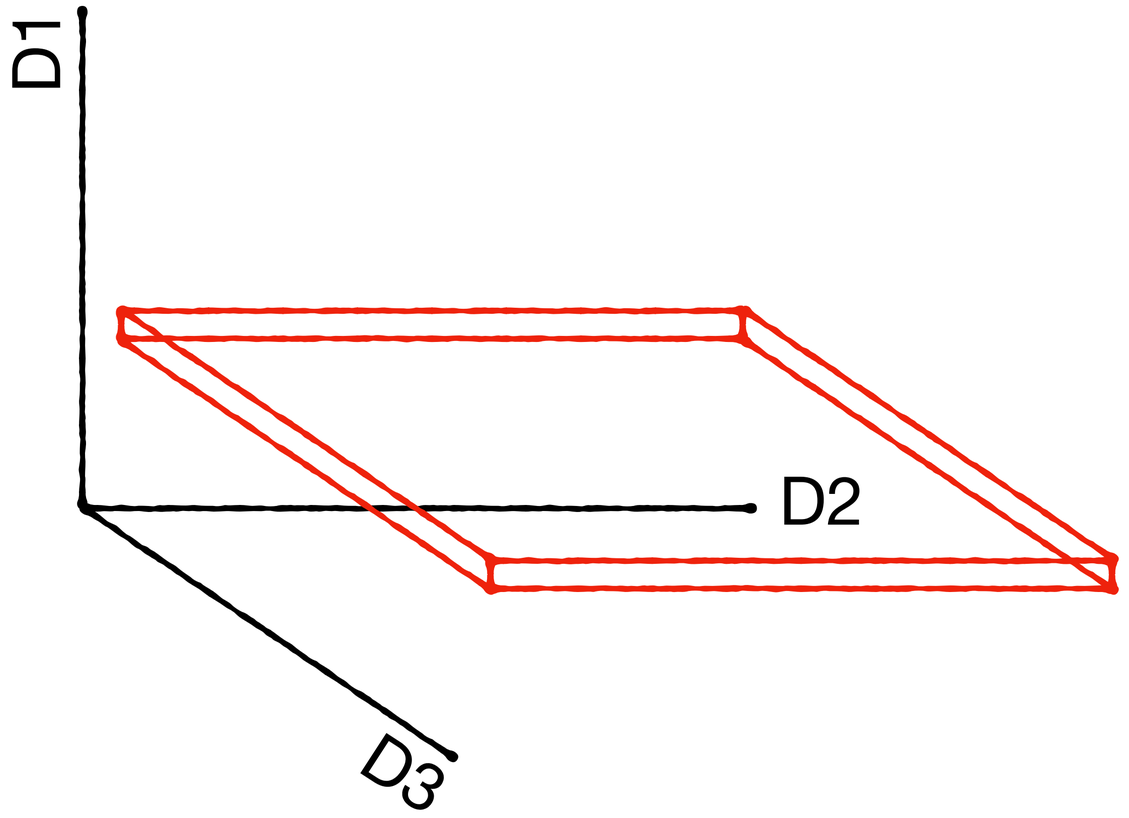
What if we increase dimensionality?

Cost? $O(2 \cdot \sqrt{N})$



What if we increase dimensionality?

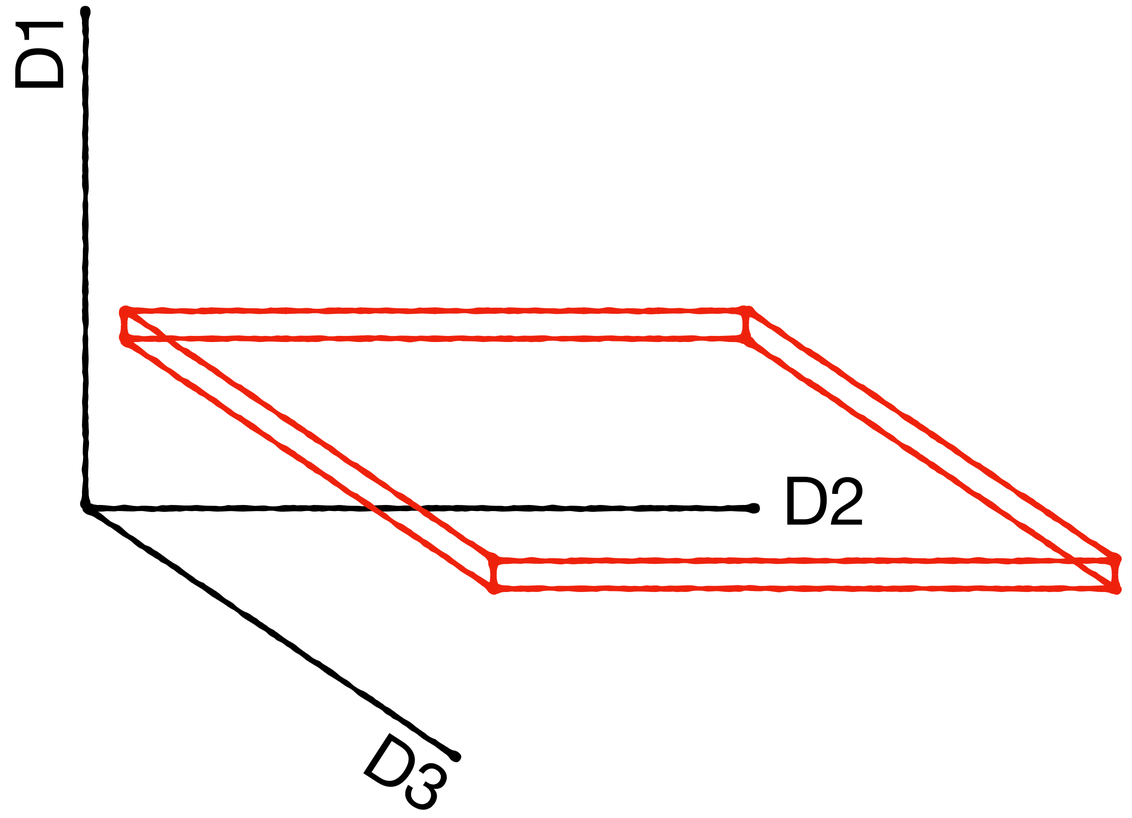
Cost? $O(2 \cdot N^{(D-1)/D})$



What if we increase dimensionality?

Cost? $O(2 \cdot N^{(D-1)/D})$

Seems to only occur with low selectivity over most dimensions

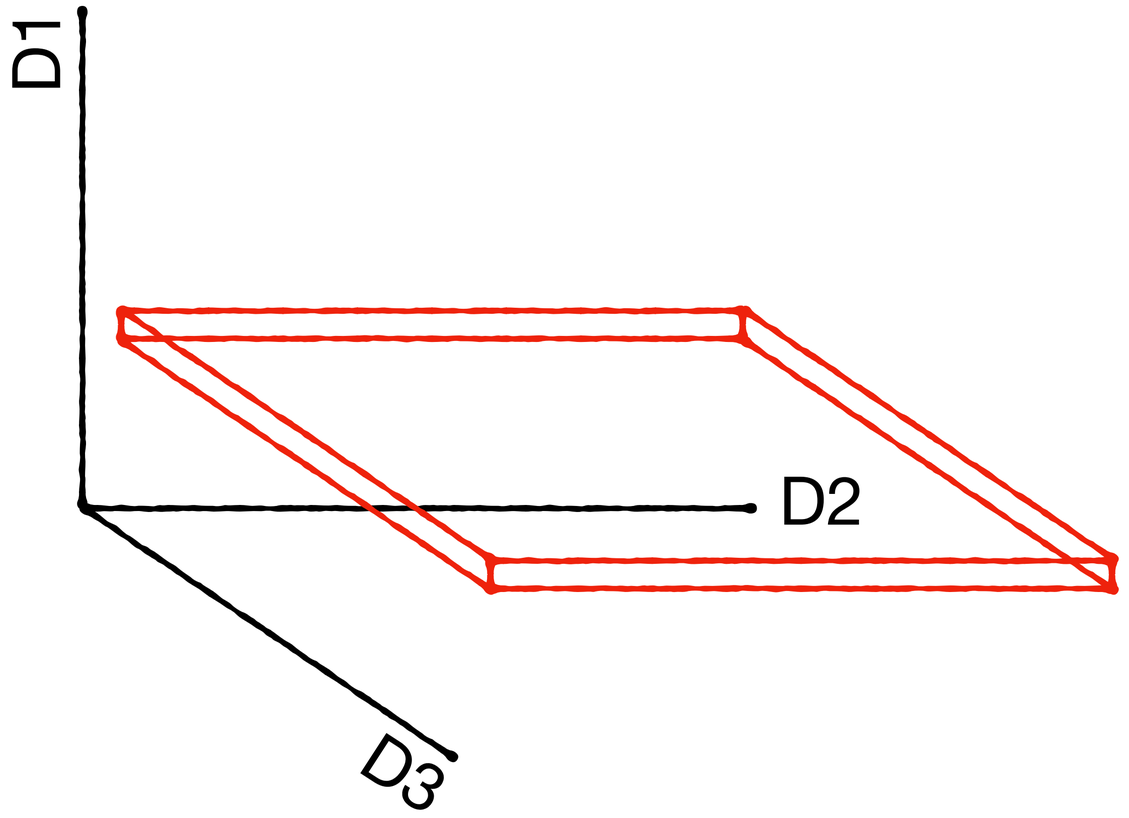


What if we increase dimensionality?

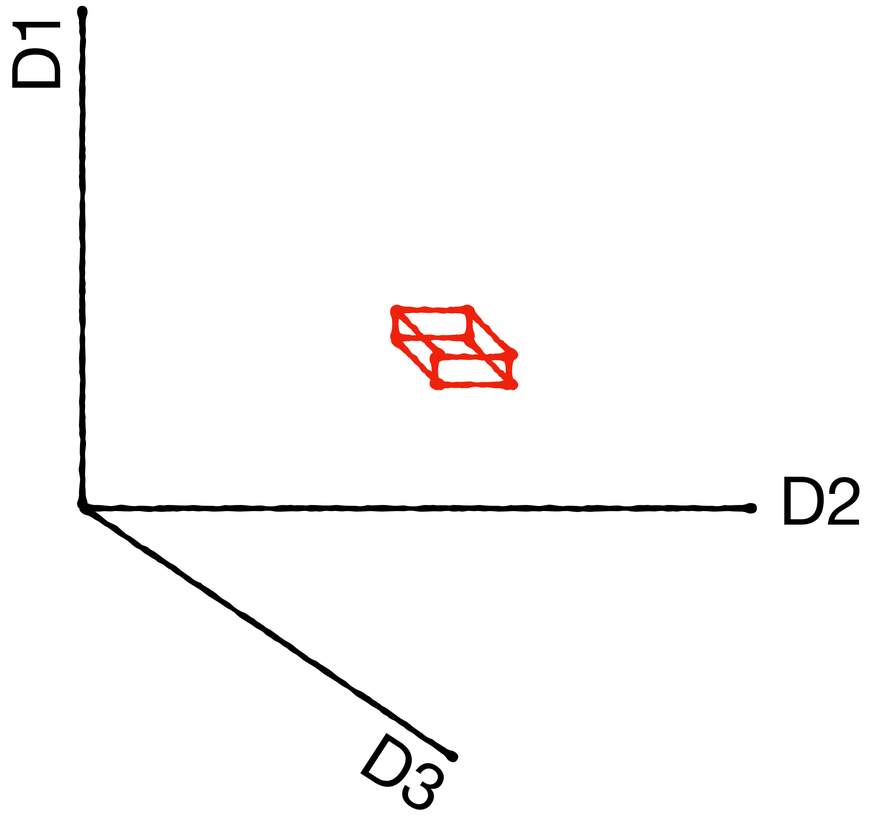
Cost? $O(2 \cdot N^{(D-1)/D})$

Seems to only occur with low selectivity over most dimensions

Each dimension that we're not selective over contributes: $N^{1/D}$

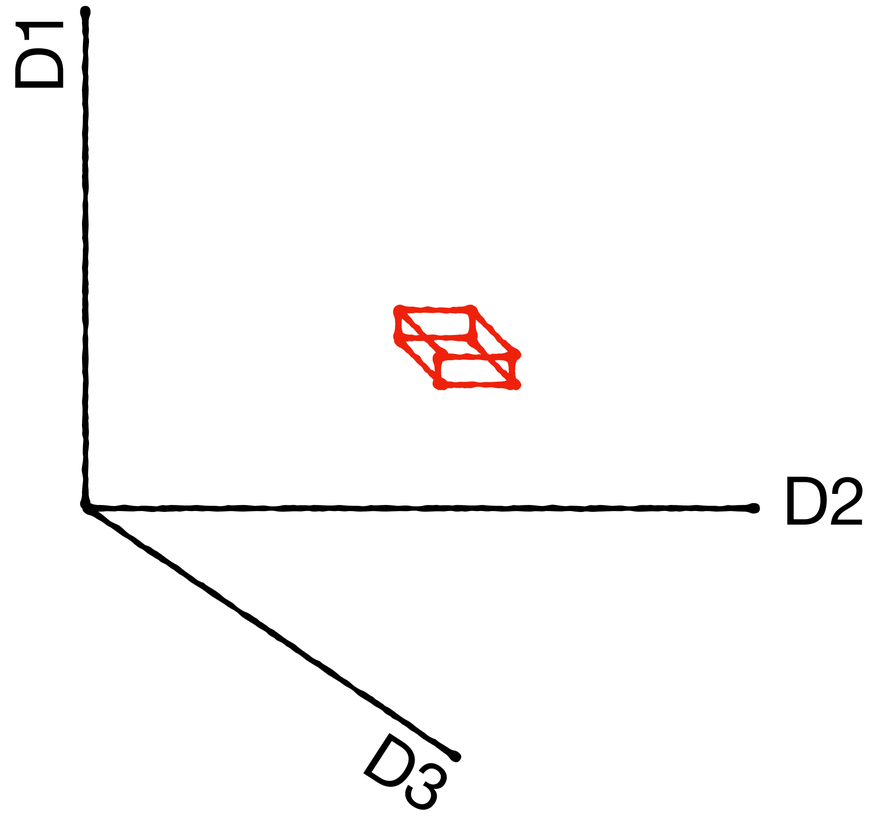


Selective queries across all dimensions are likely not so bad

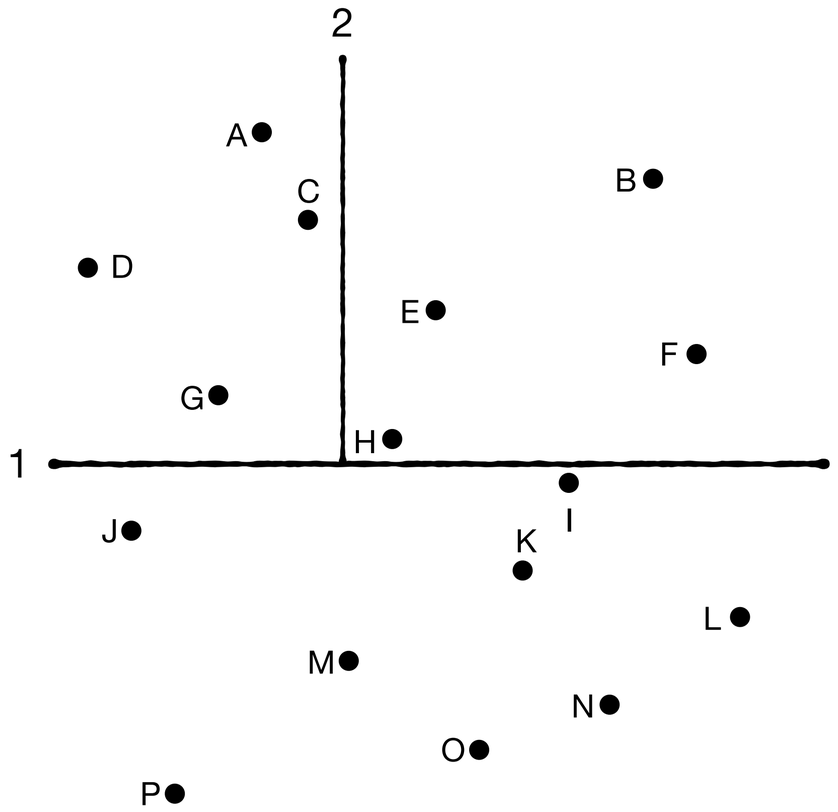


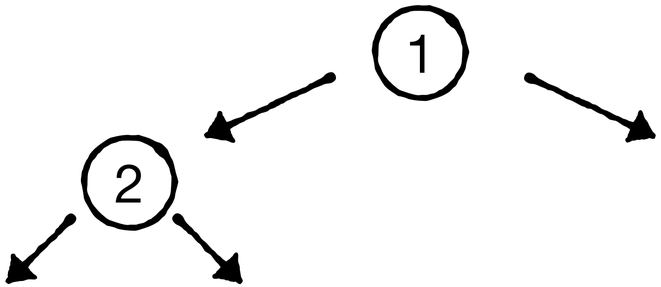
Selective queries across all dimensions are likely not so bad

My guess: O(2 · log(N))



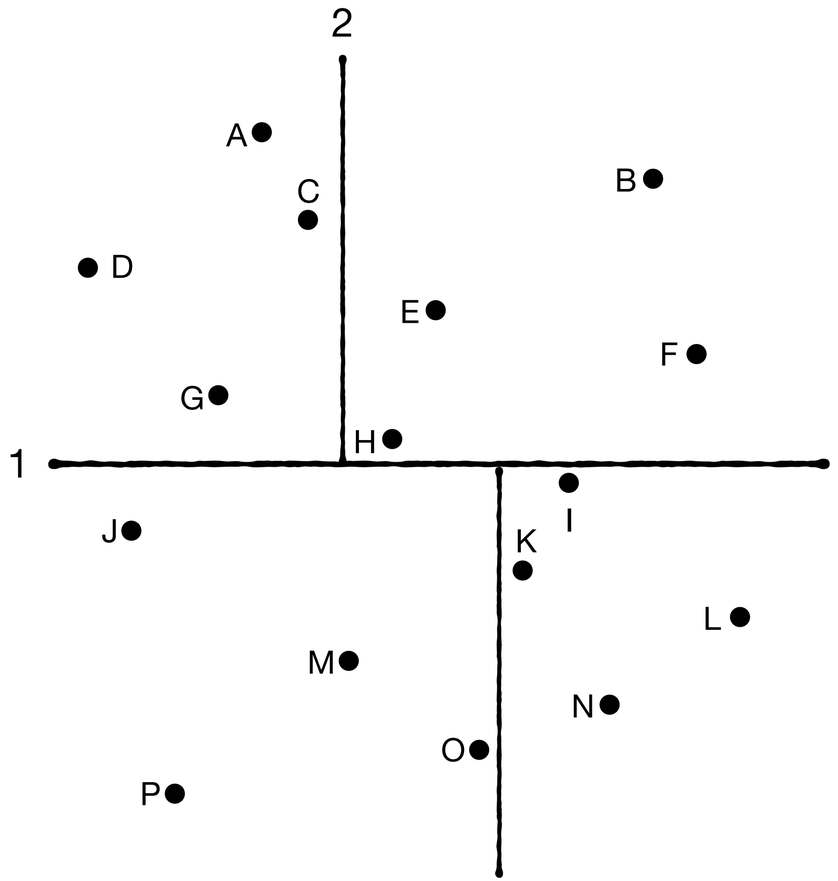
Efficient constructions

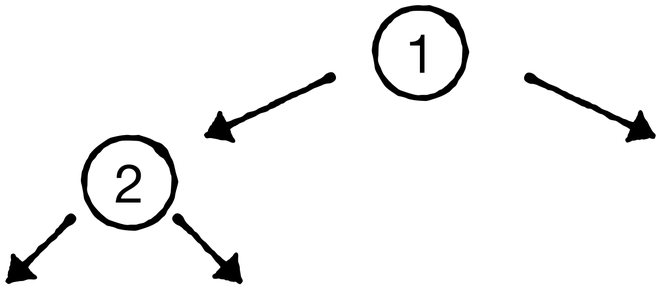




Efficient constructions

How to find median in each iteration?

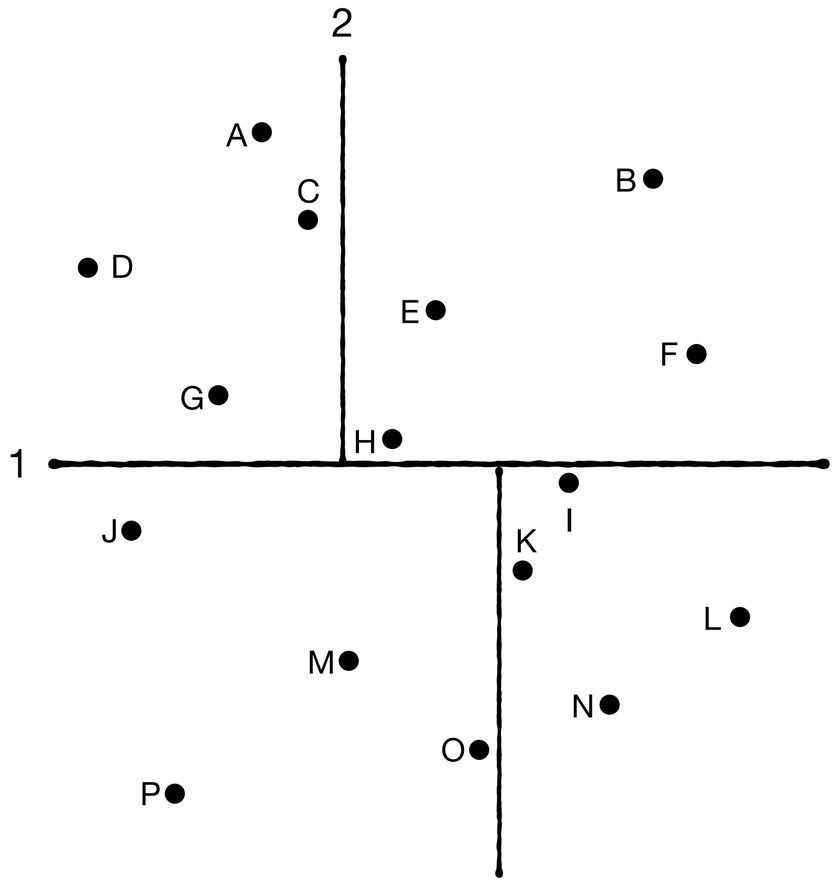


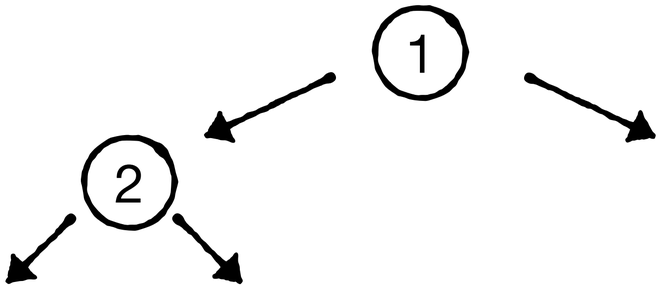


Efficient constructions

How to find median in each iteration?

Sort based on each dimension: $O(D \cdot N \cdot \log_2(N))$



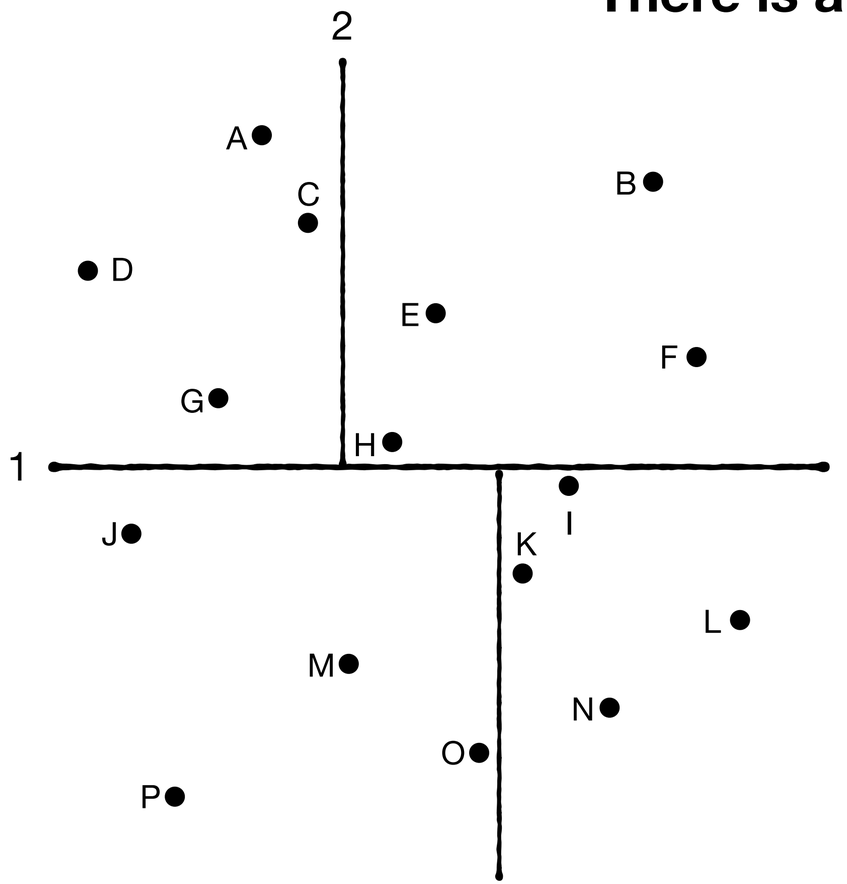


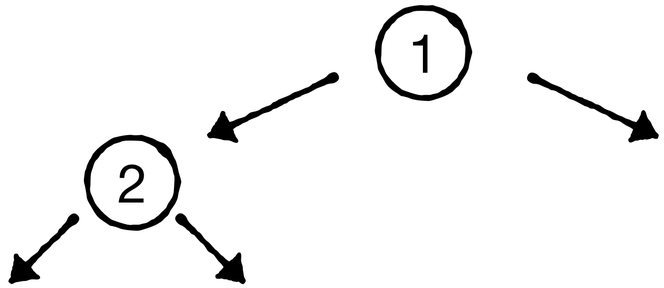
Efficient constructions

How to find median in each iteration?

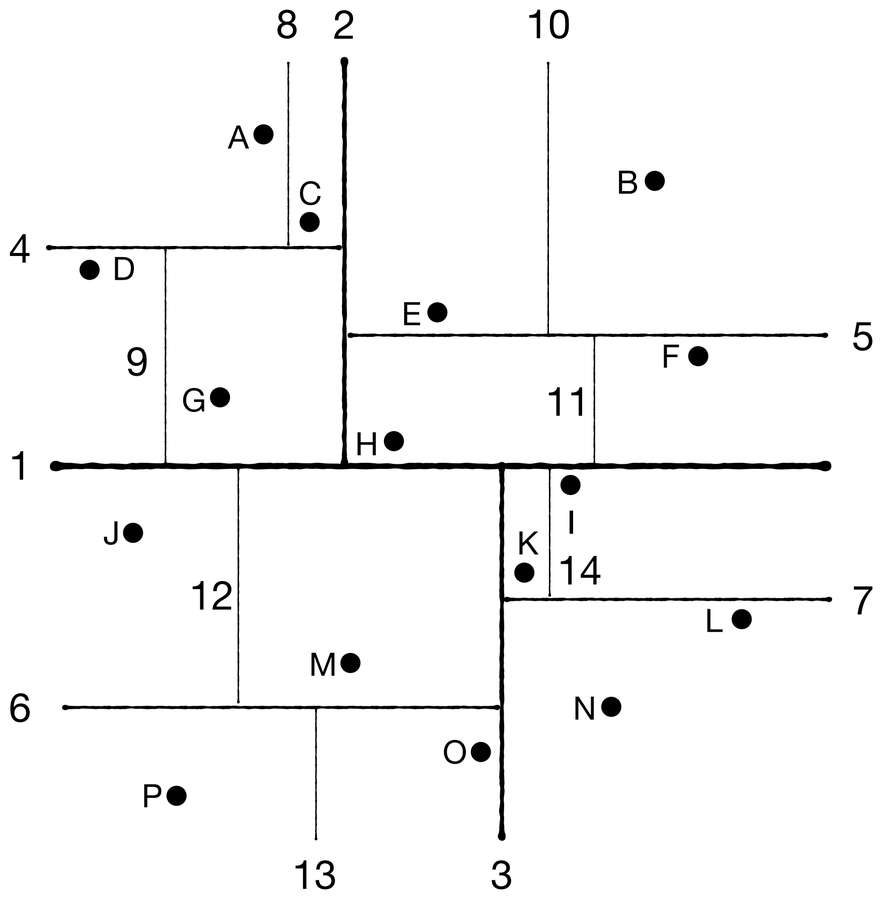
Sort based on each dimension: $O(D \cdot N \cdot \log_2(N))$

There is an $O(N \cdot \log_2(N))$ algorithm :)

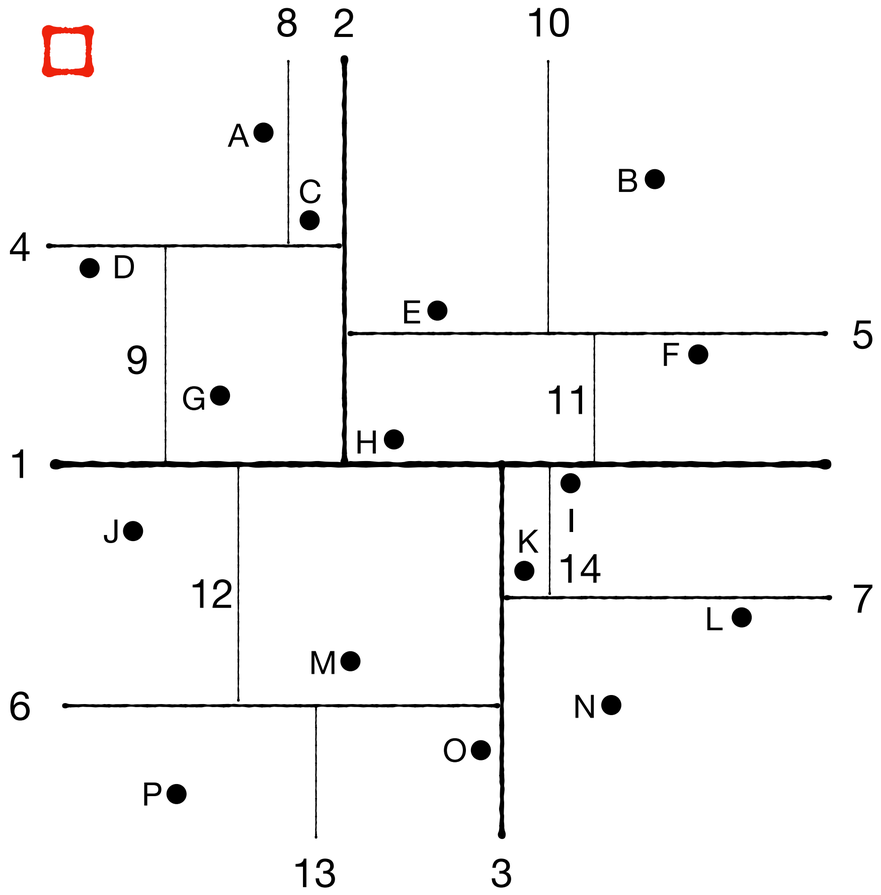




Is partitioning the space the best approach?



Is partitioning the space the best approach?



1

Some nodes need to be accessed even if query is far from entry

KD-Trees	R-Trees	R+tree	R*-tree	UB-tree
1975	1984	1987	1990	2000

R-Trees: A Dynamic Index Structure for Spatial Searching

ACM SIGMOD Record Journal, 1984

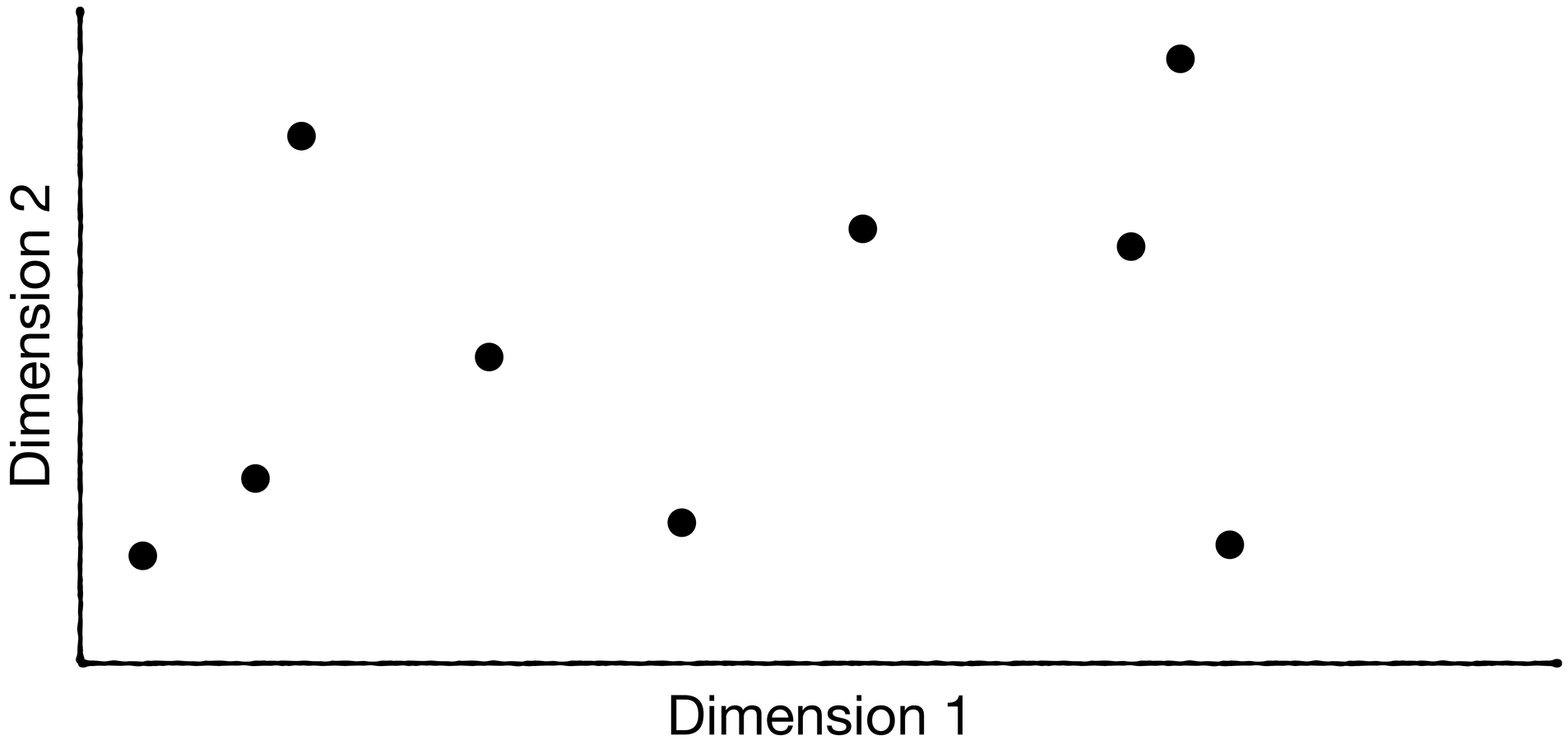
Antonin Gutman

Rectangle-Trees: A Dynamic Index Structure for Spatial Searching

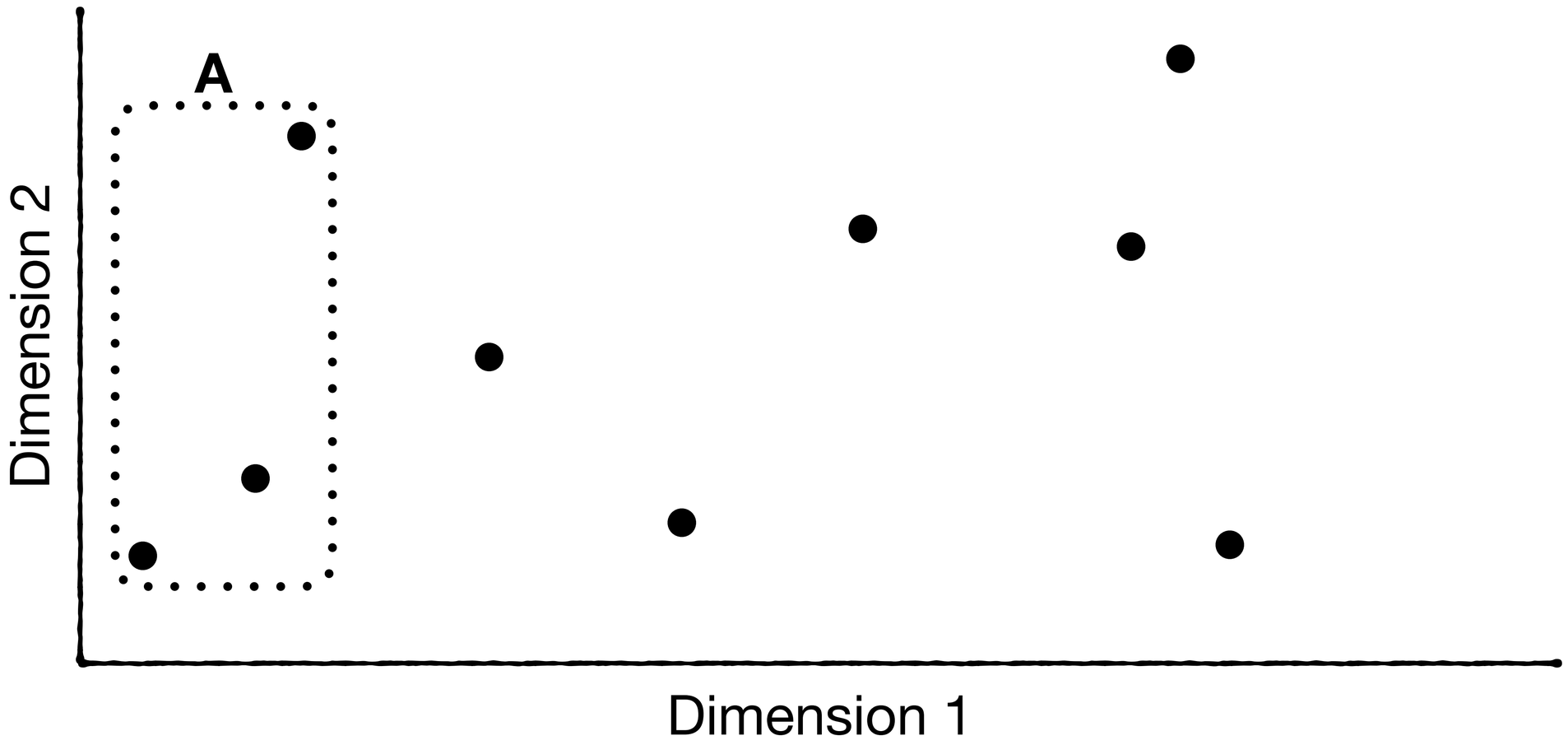
ACM SIGMOD Record Journal, 1984

Antonin Gutman

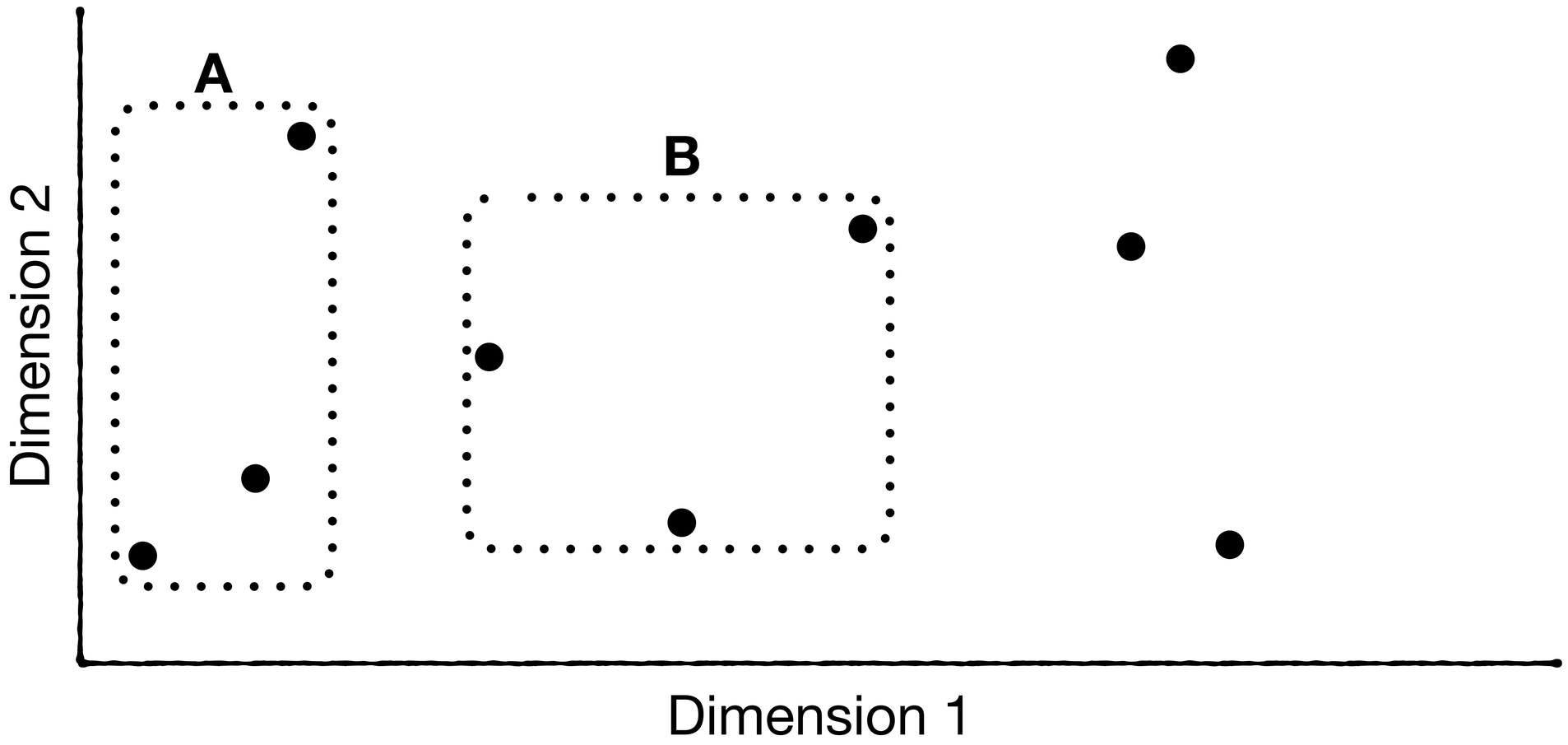
A hierarchy of minimum bounding rectangles



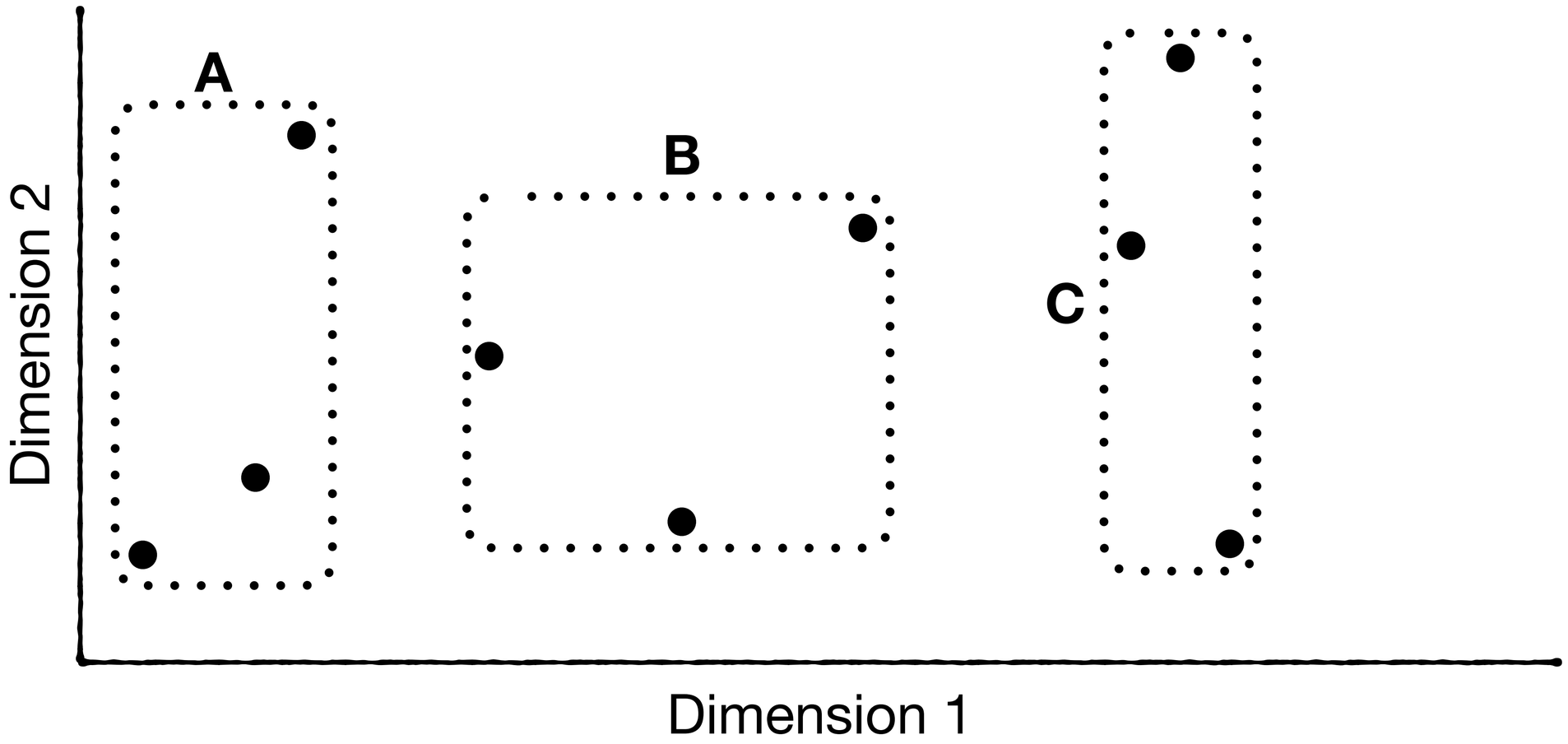
A hierarchy of minimum bounding rectangles



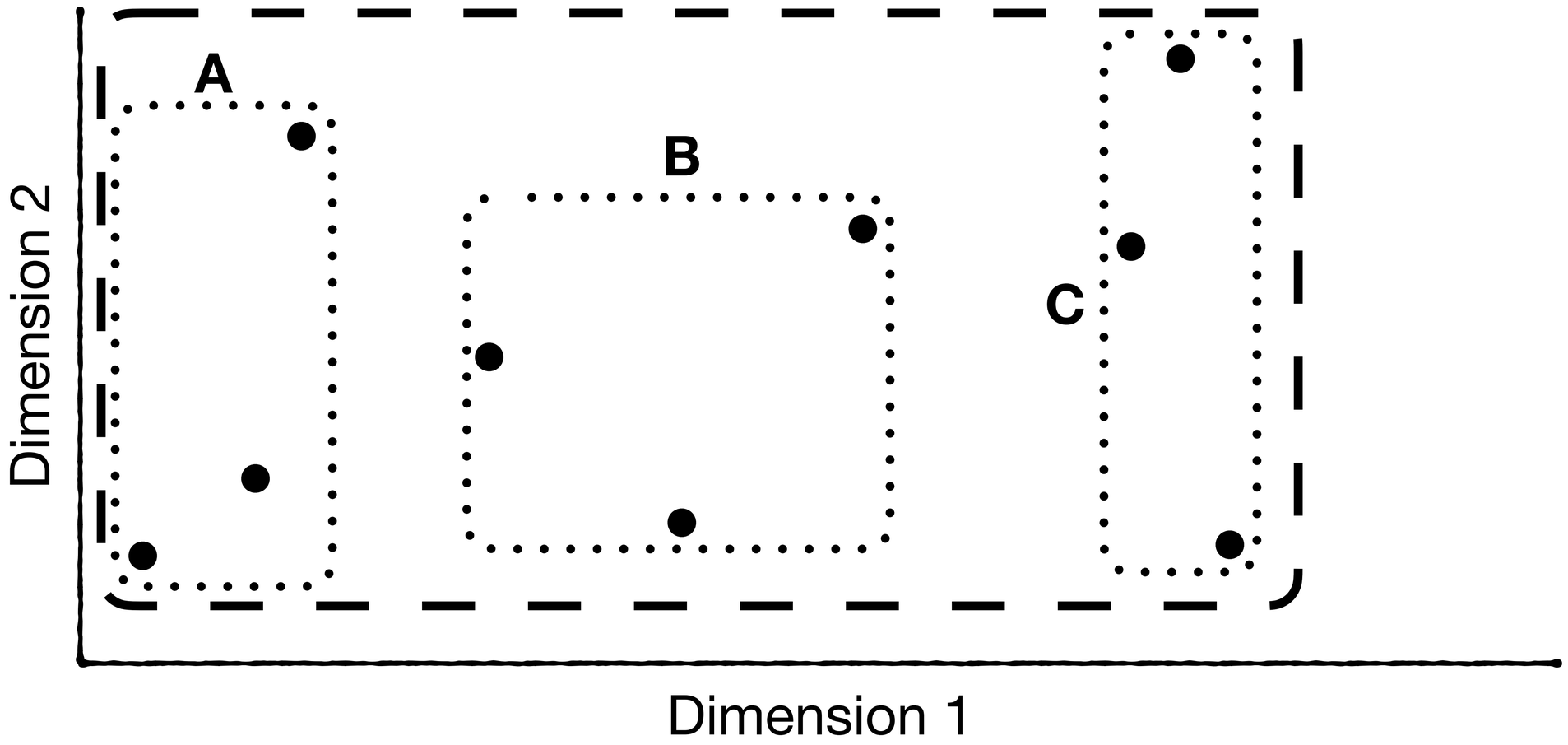
A hierarchy of minimum bounding rectangles



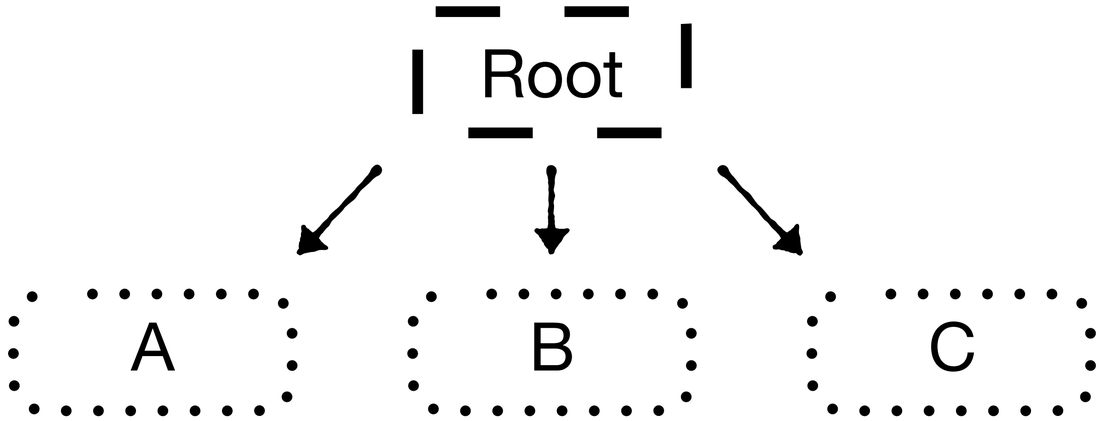
A hierarchy of minimum bounding rectangles



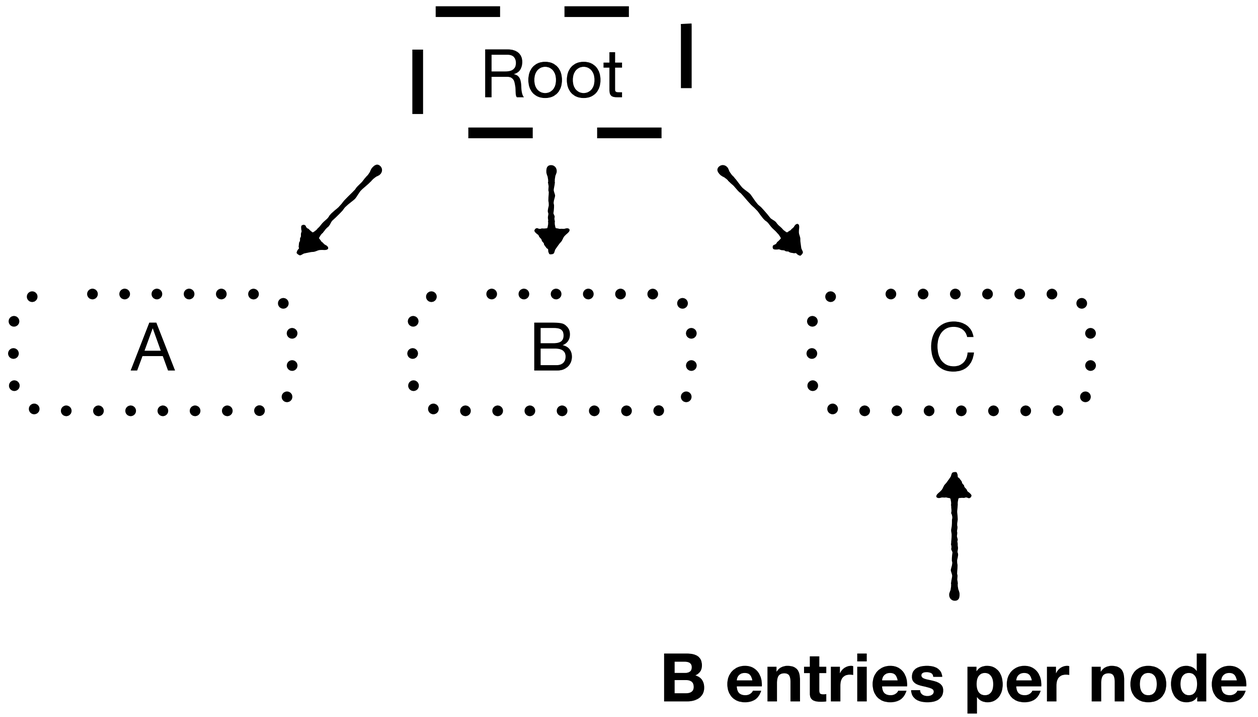
A hierarchy of minimum bounding rectangles



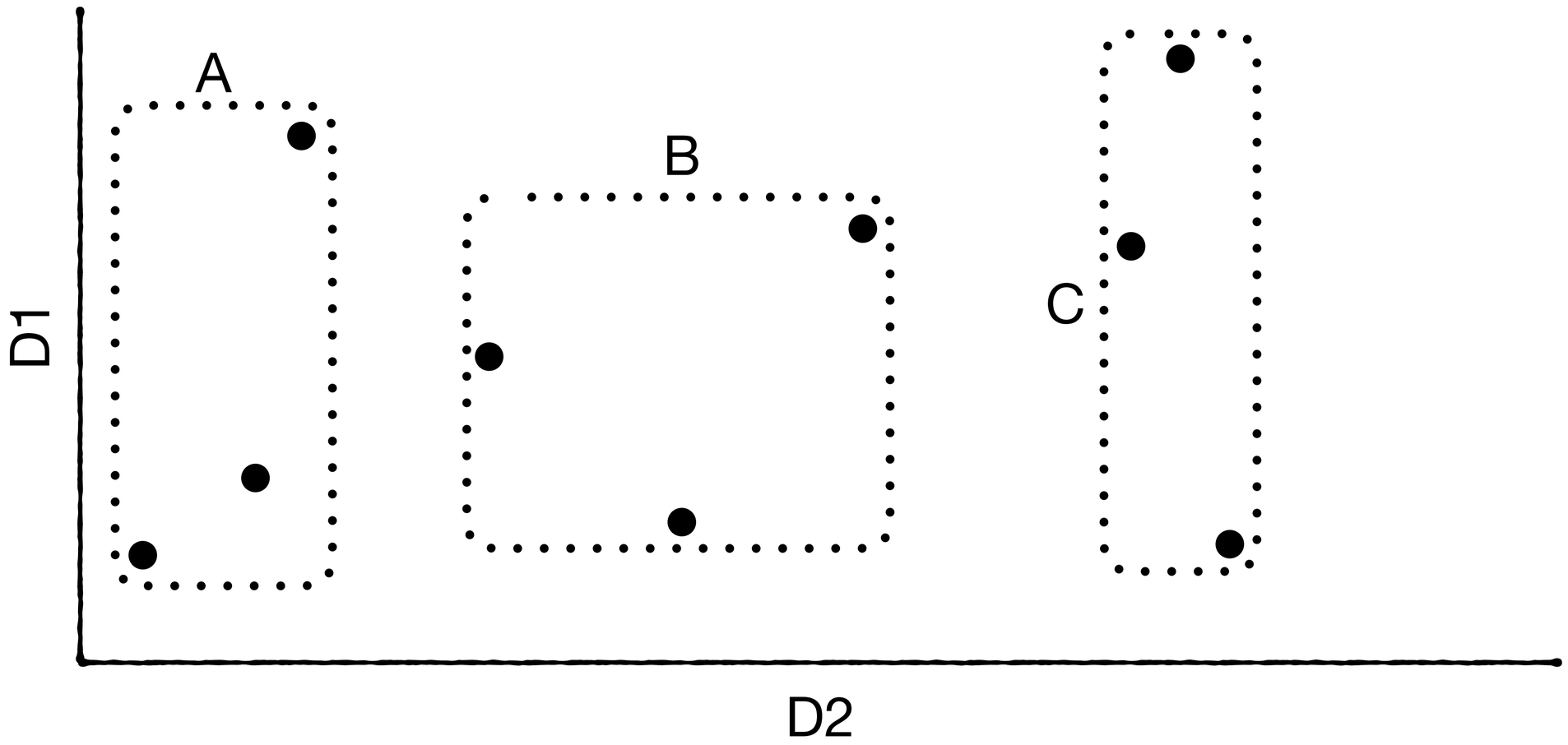
Physically Structured as a B-Tree



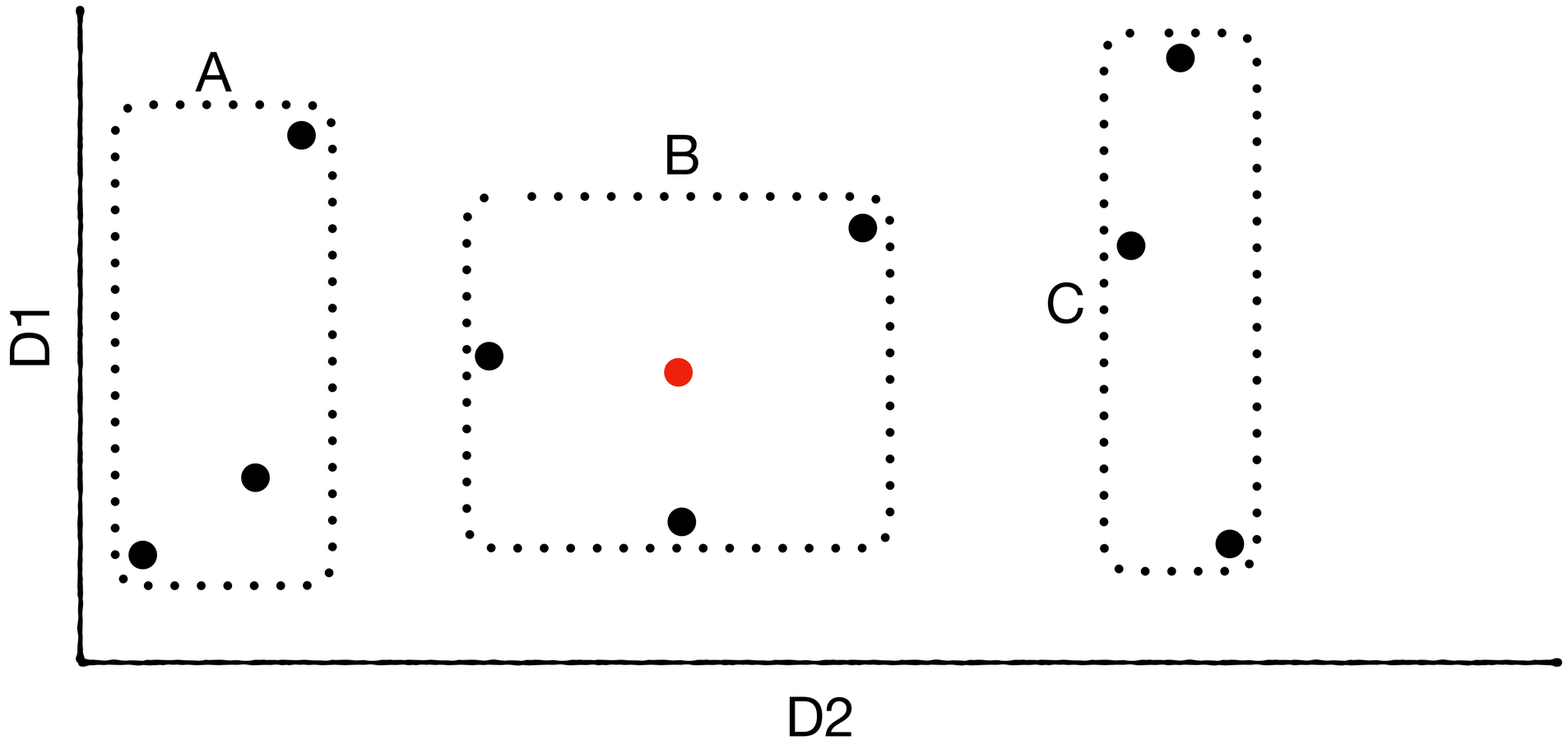
Physically Structured as a B-Tree



How to handle inserts?

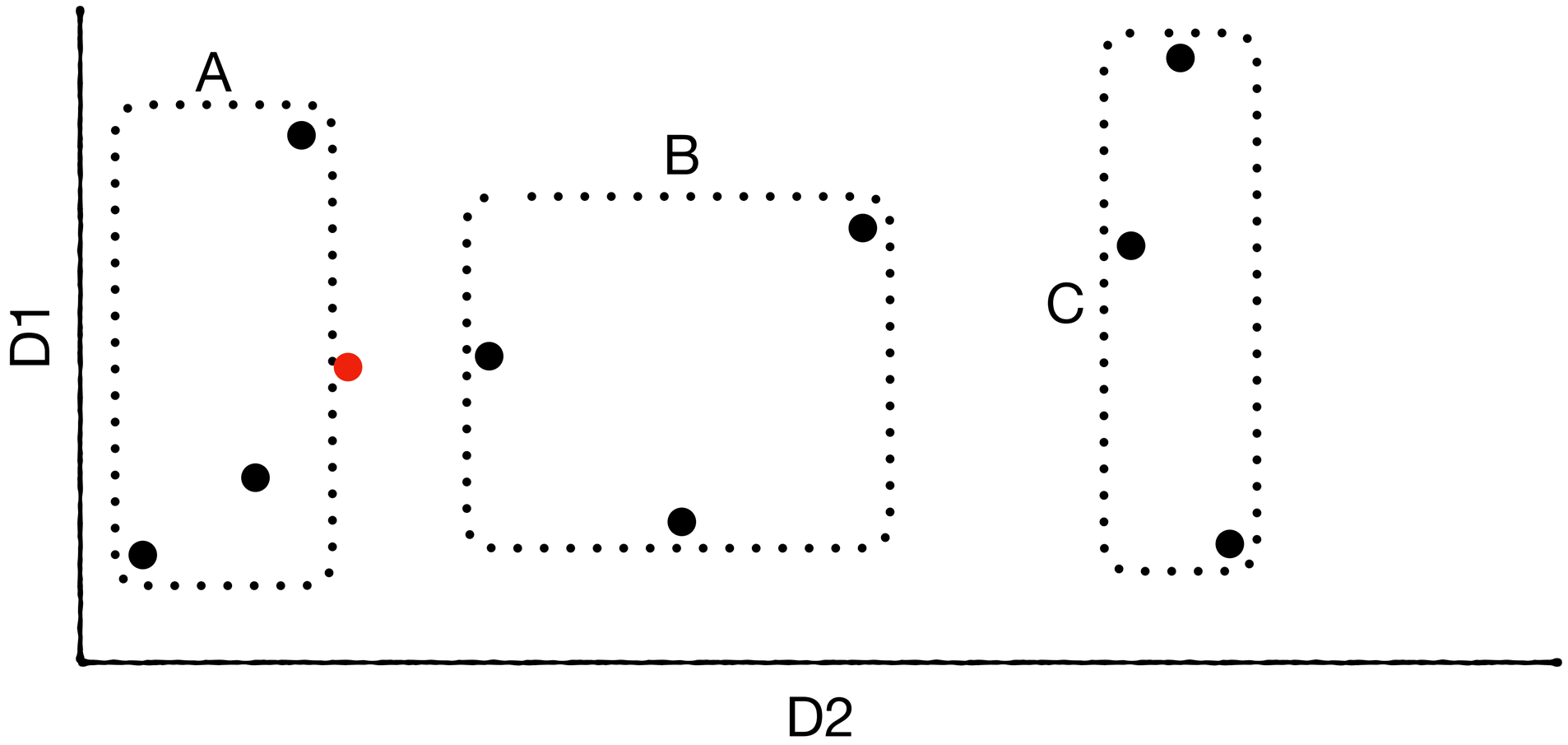


If point is in existing rectangle, add it to that rectangle



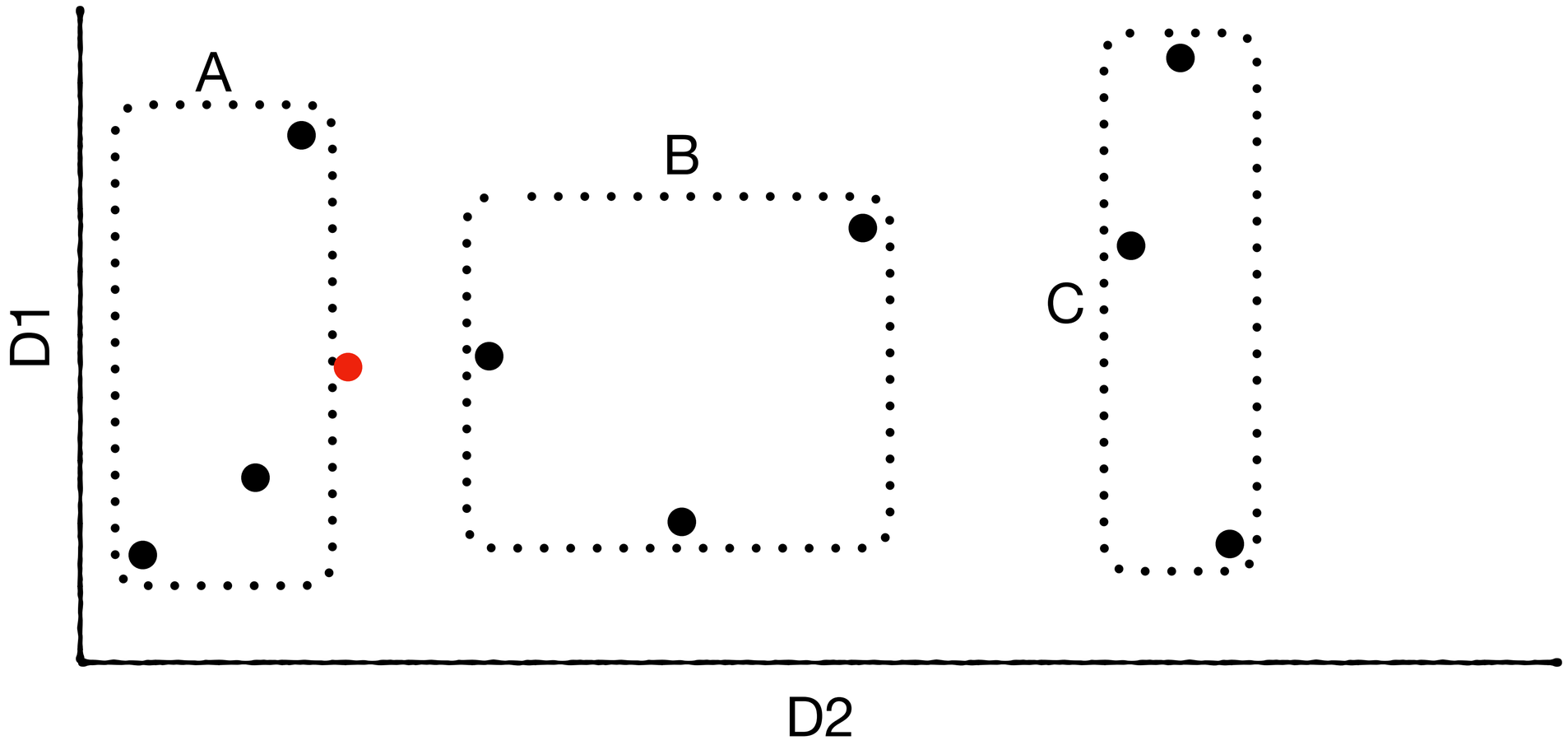
If point is in existing rectangle, add it to that rectangle

Otherwise?



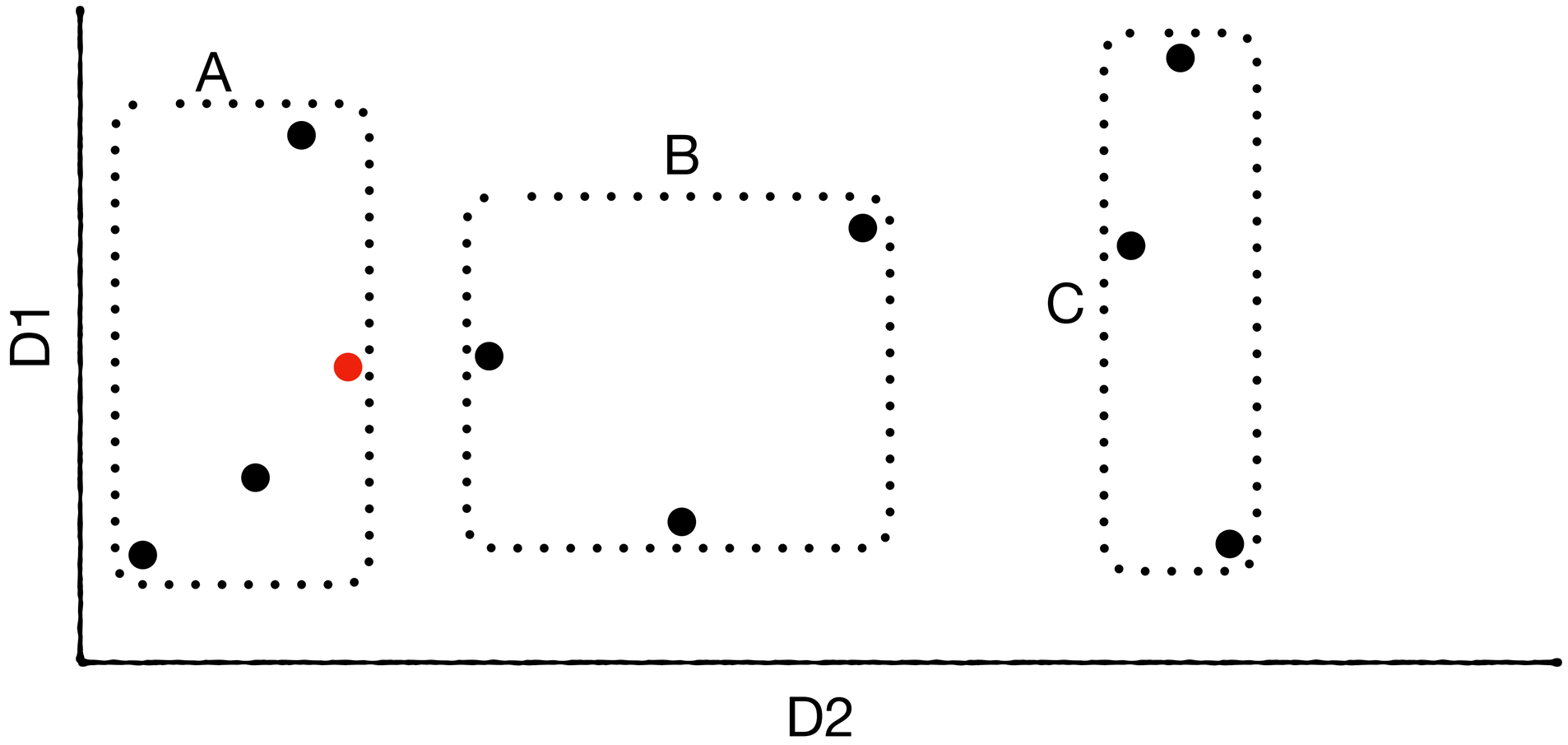
If point is in existing rectangle, add it to that rectangle

Otherwise, add to rectangle we need to expand the least



If point is in existing rectangle, add it to that rectangle

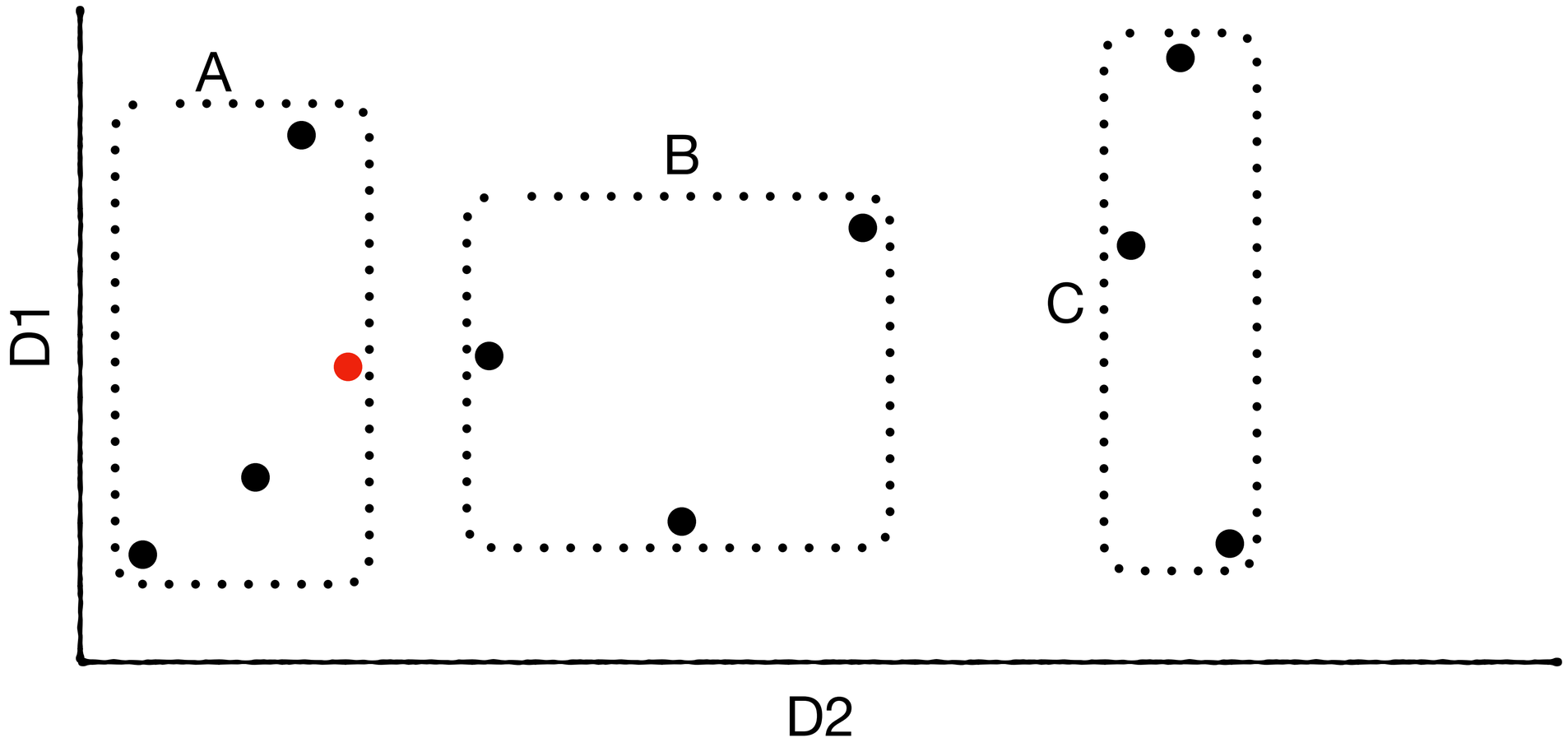
Otherwise, add to rectangle we need to expand the least



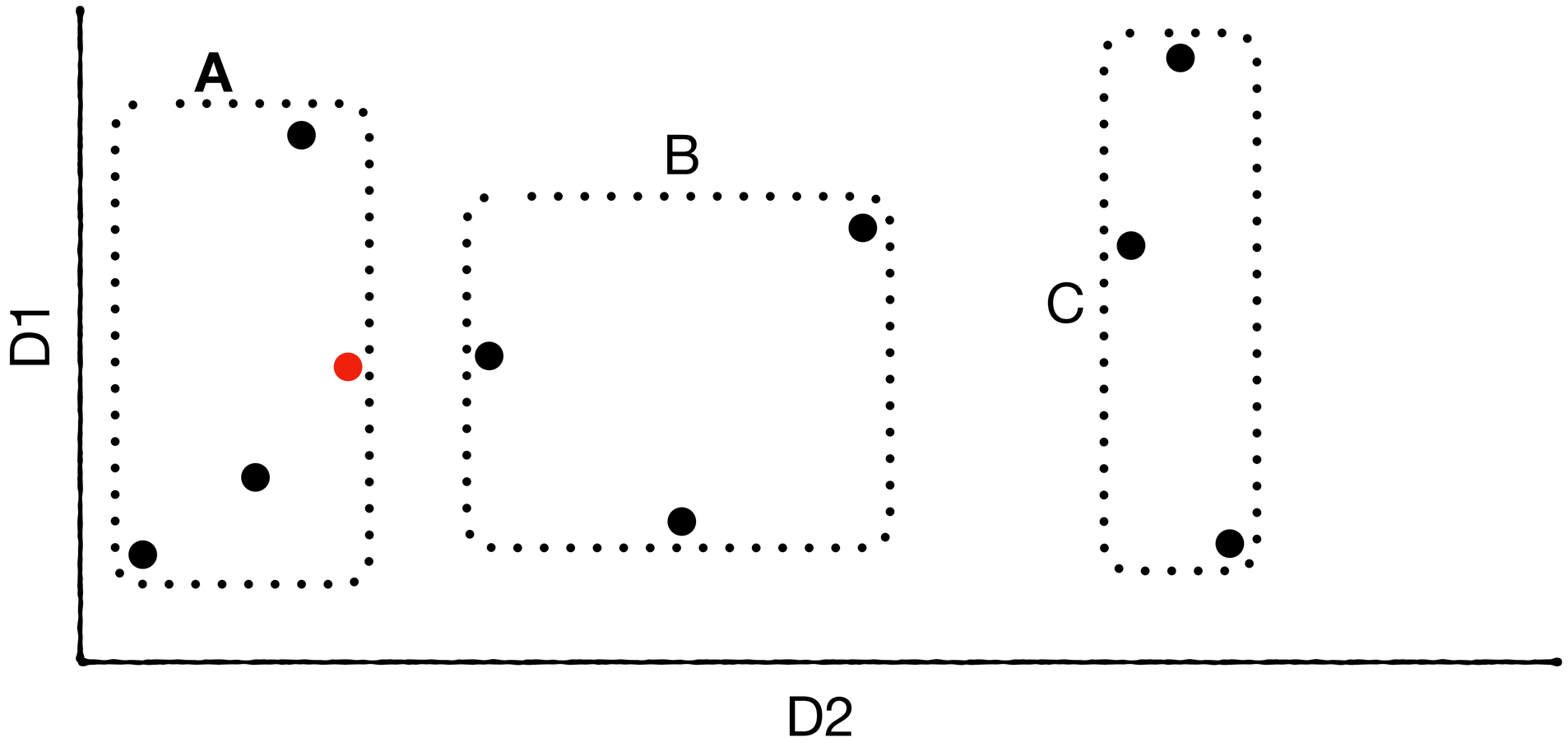
If point is in existing rectangle, add it to that rectangle

Otherwise, add to rectangle we need to expand the least

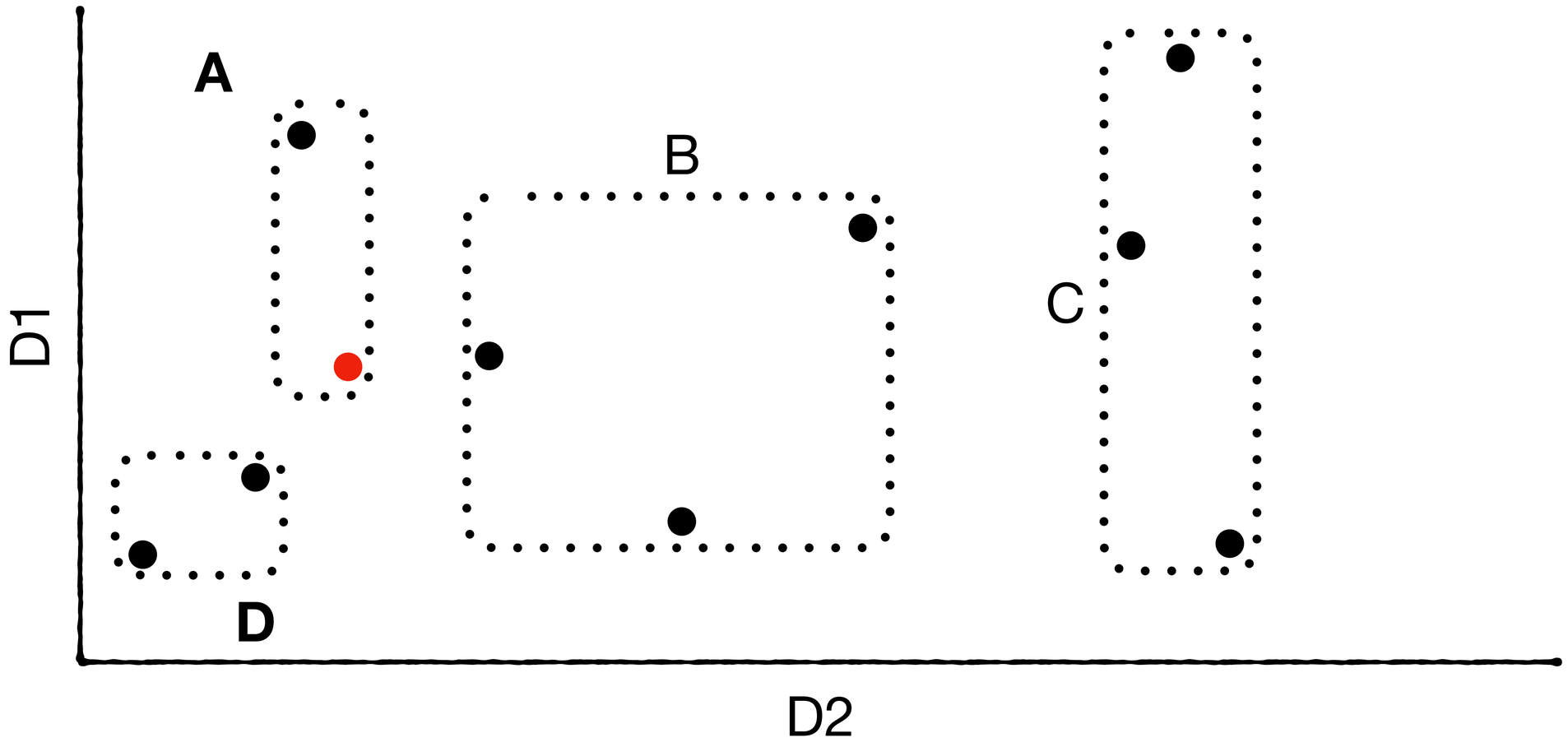
Why? To minimize coverage



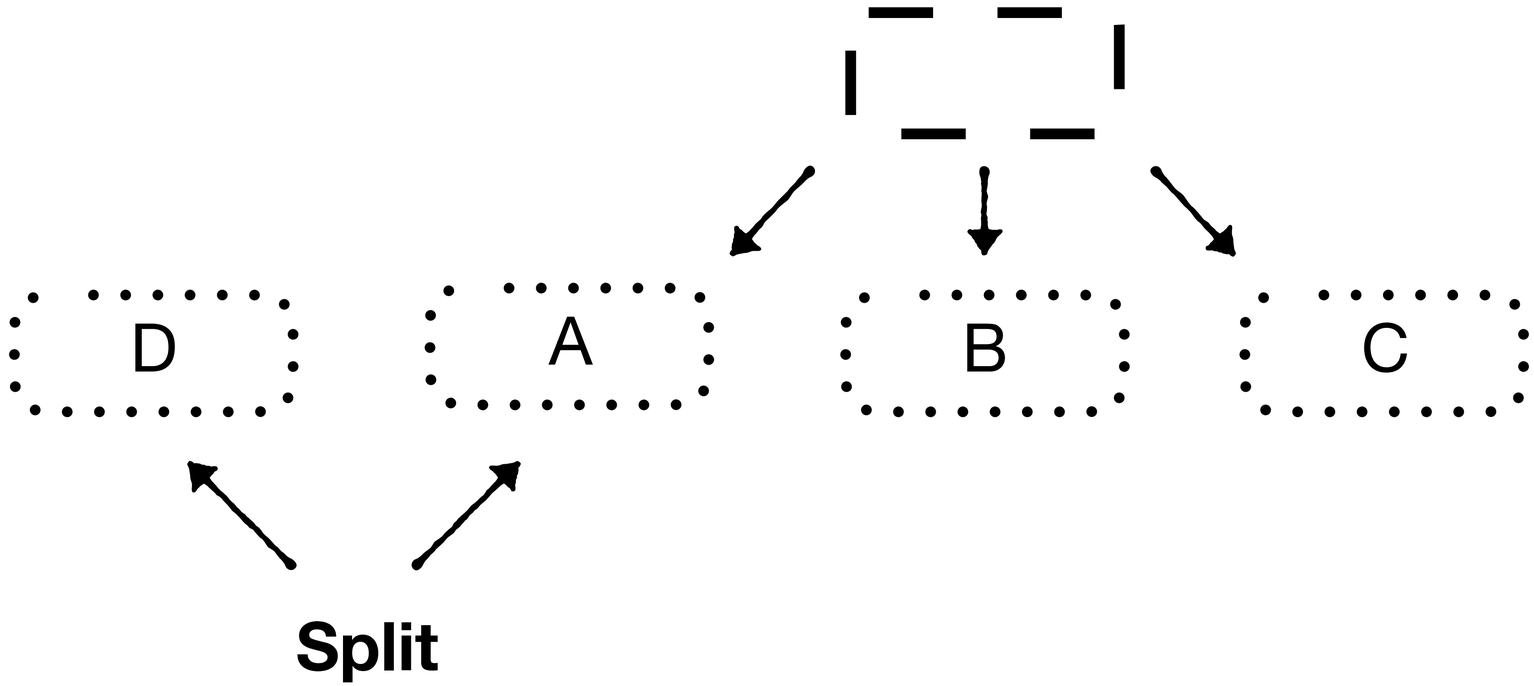
If a rectangle now contains B entries, split it (e.g., B=4)



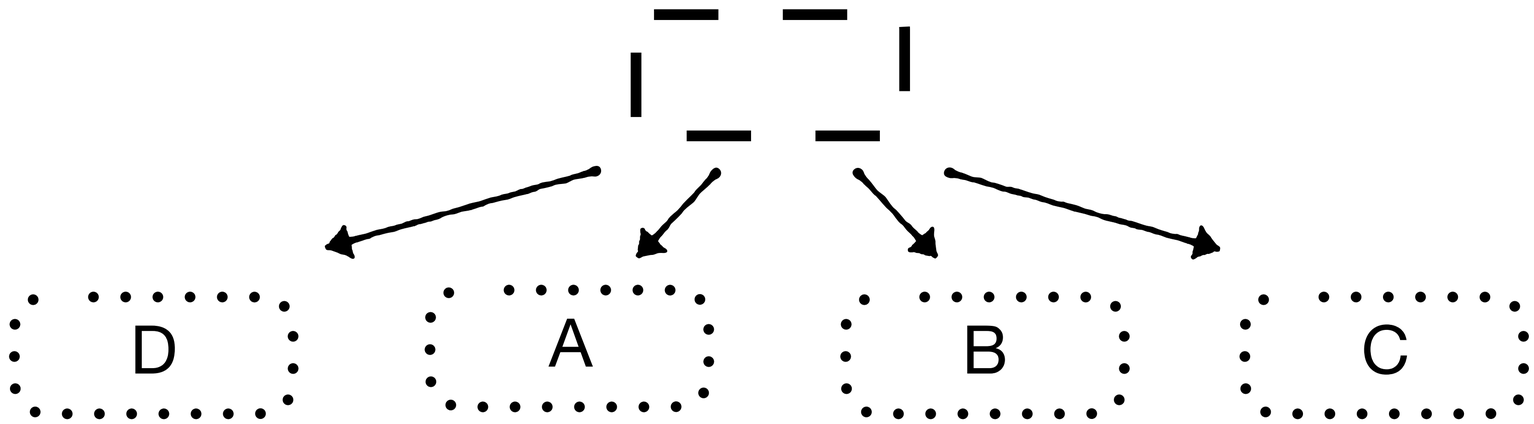
If a rectangle now contains B entries, split it (e.g., B=4)



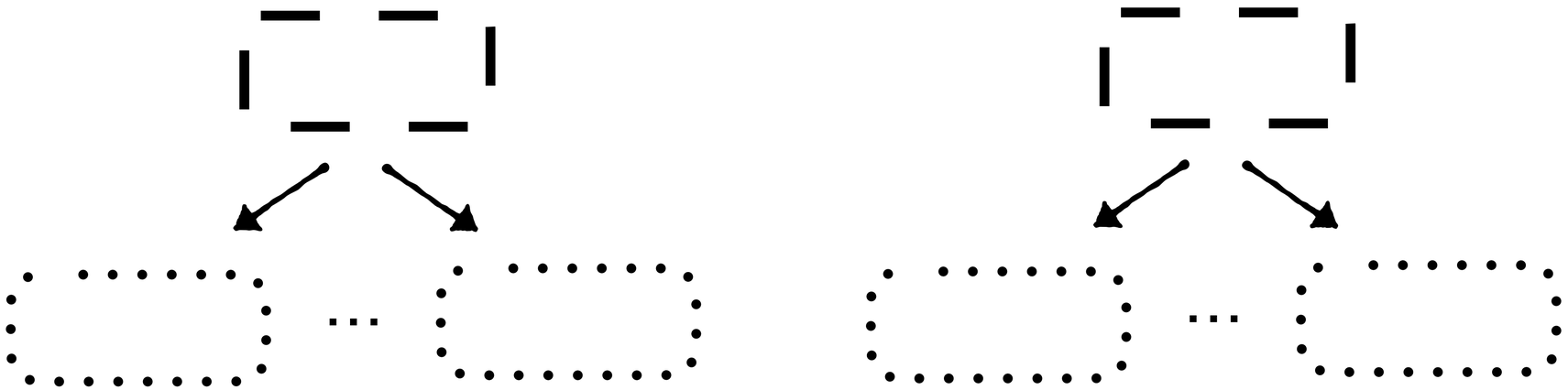
Physically Structured as a B-Tree



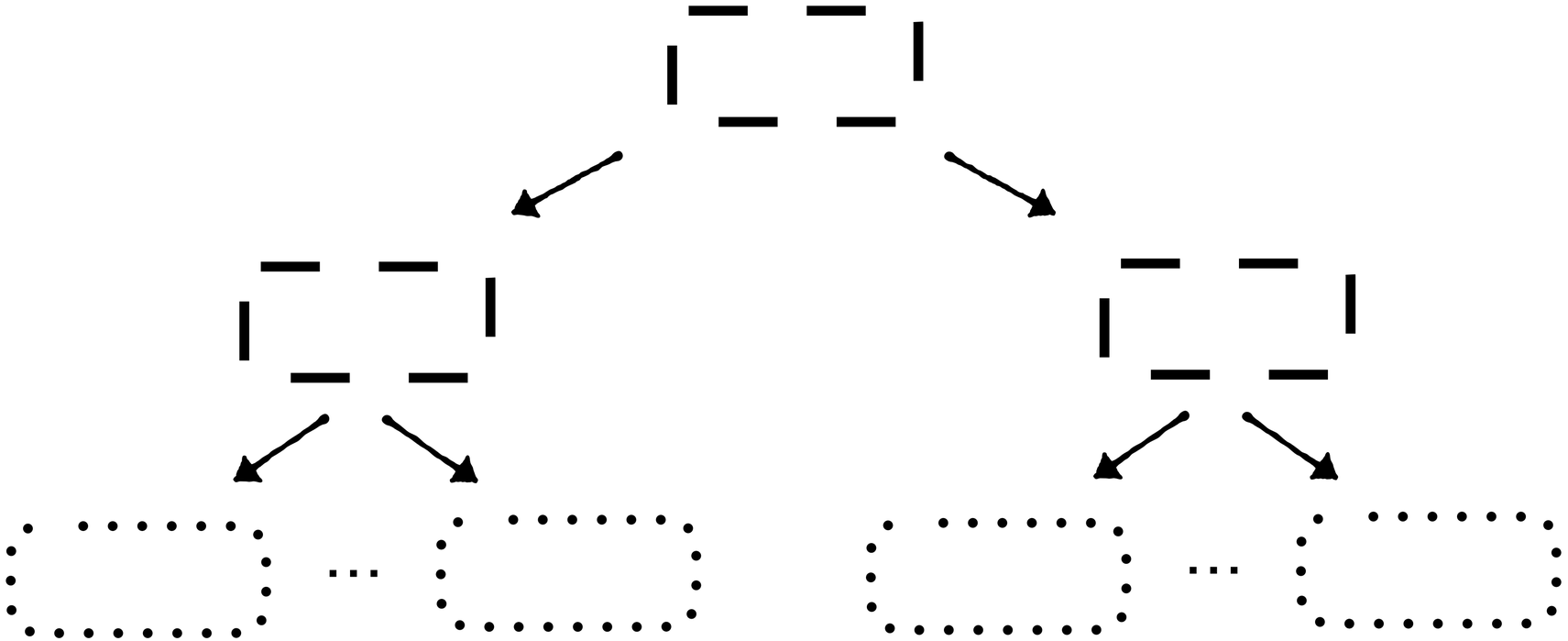
Add pointer in parent



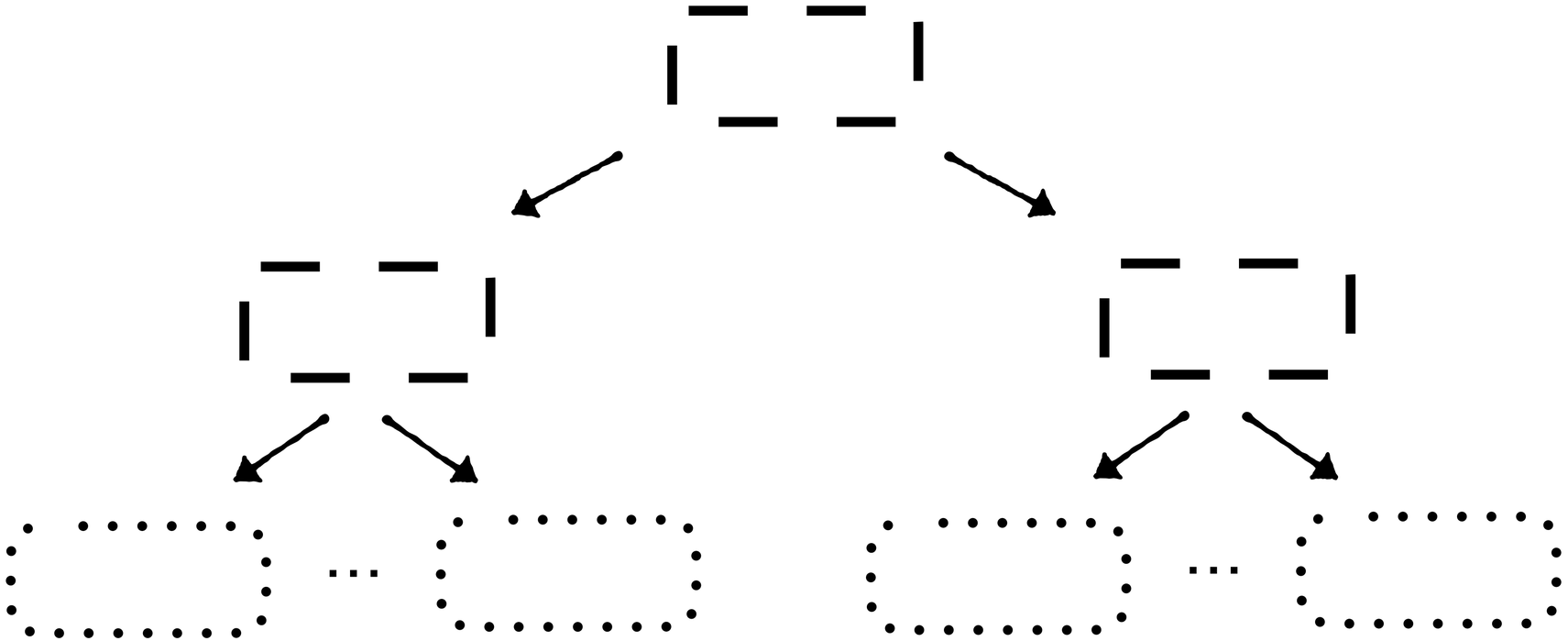
Parent may split as well



Add new parent, causing depth to grow

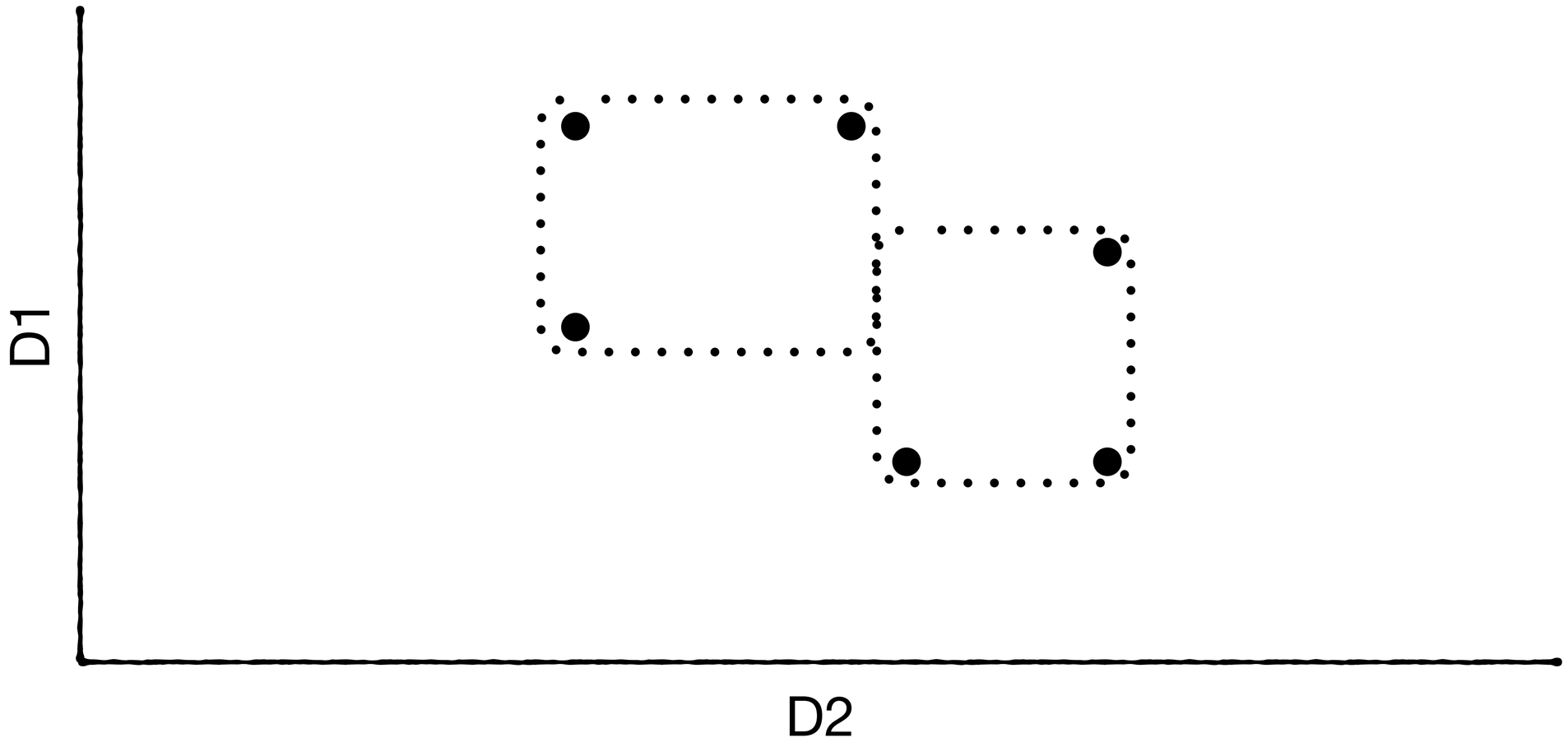


Add new parent, causing depth to grow

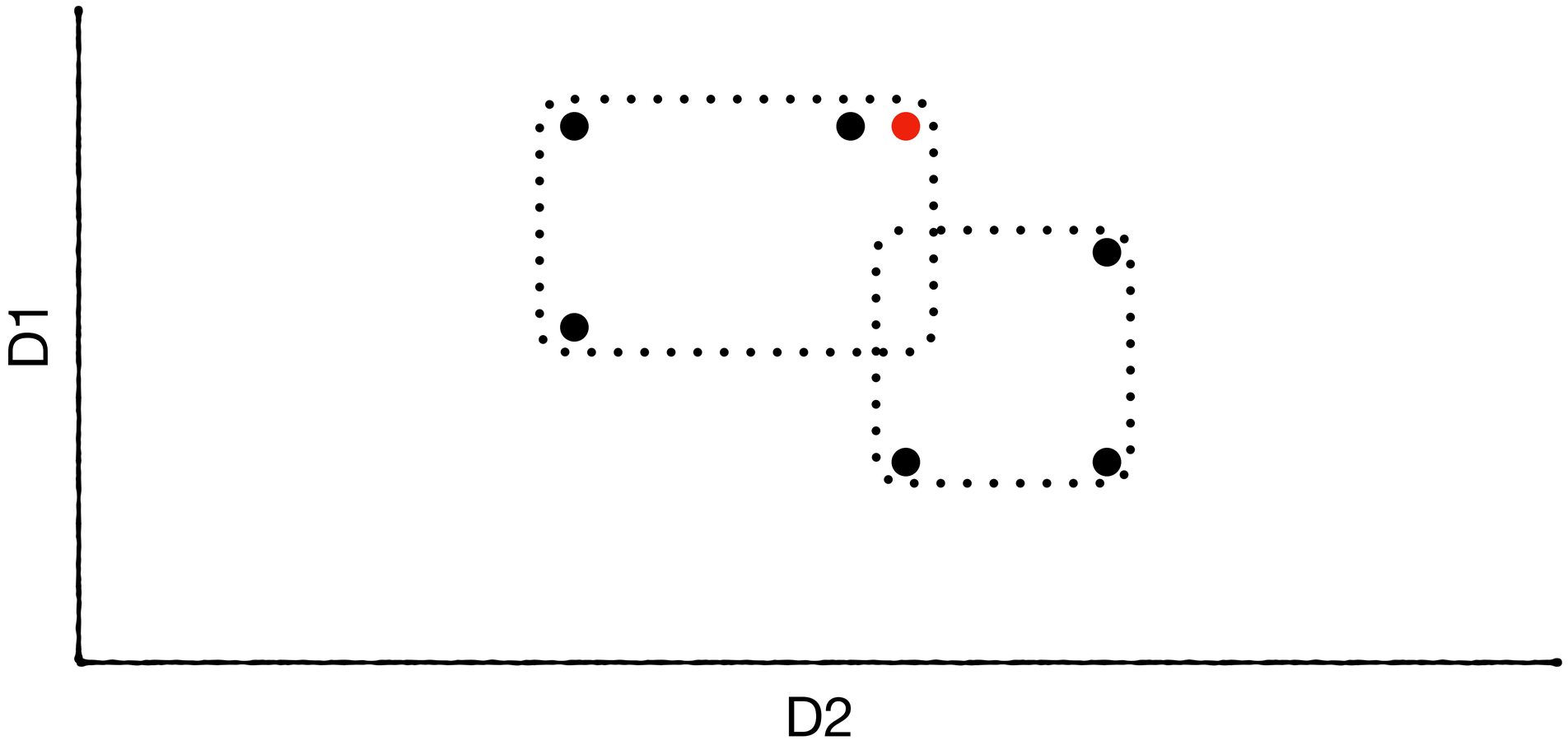


The tree stays perfectly balanced

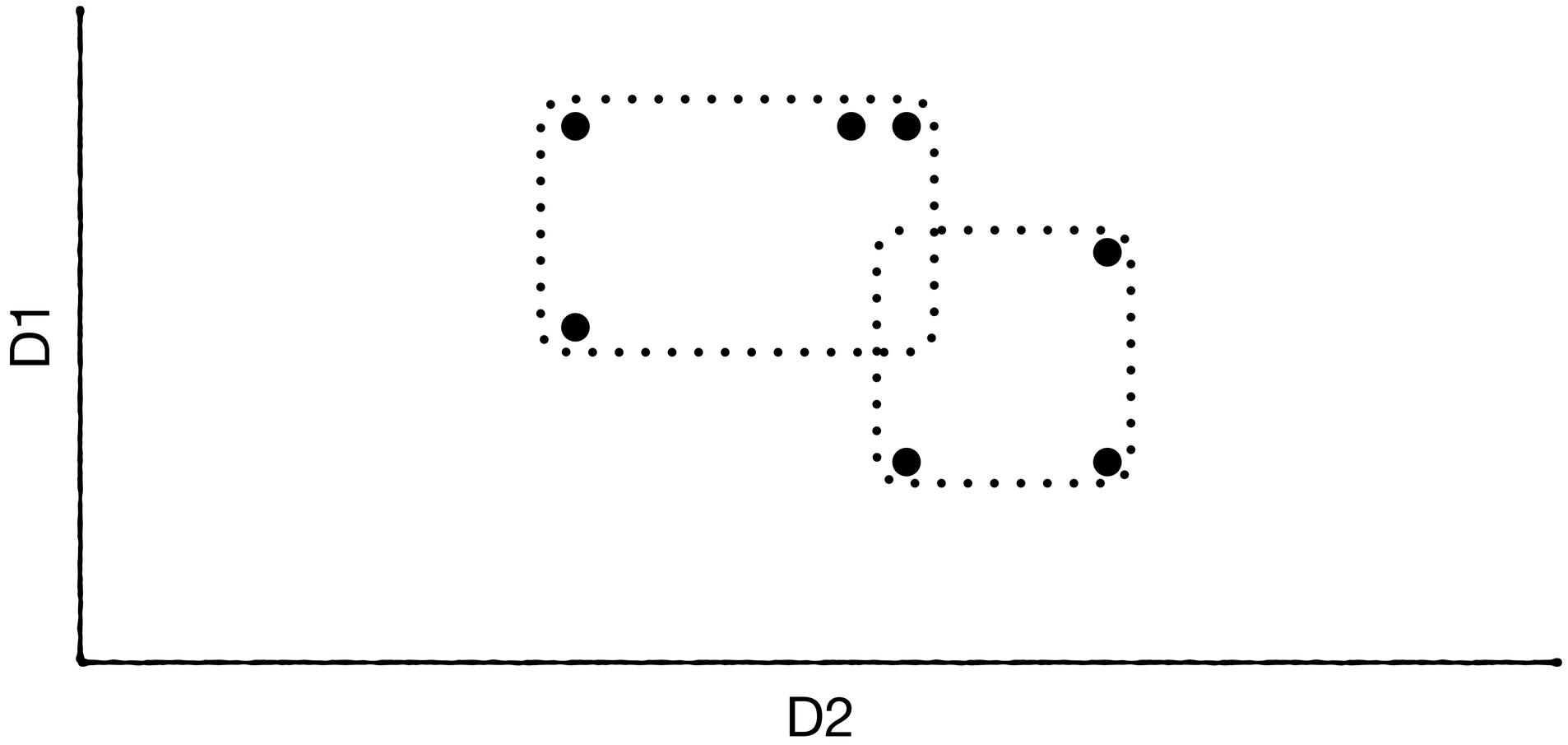
Insertions may cause overlap across rectangles



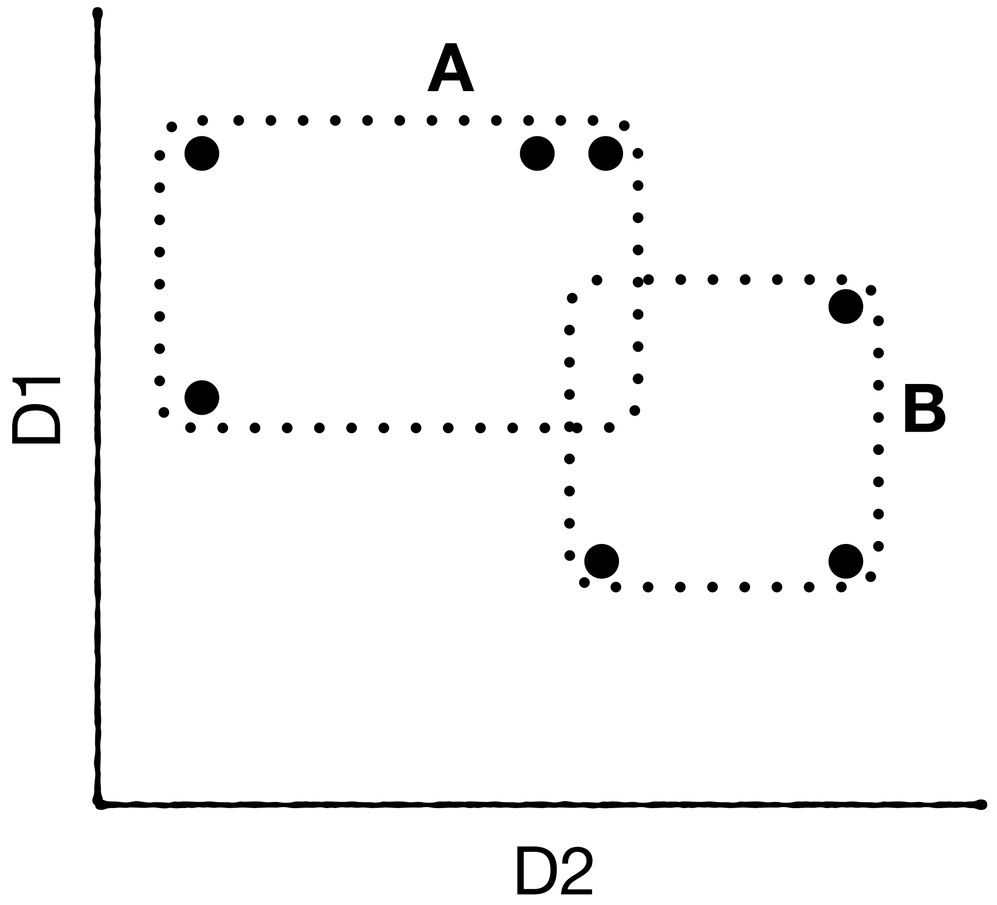
Insertions may cause overlap across rectangles

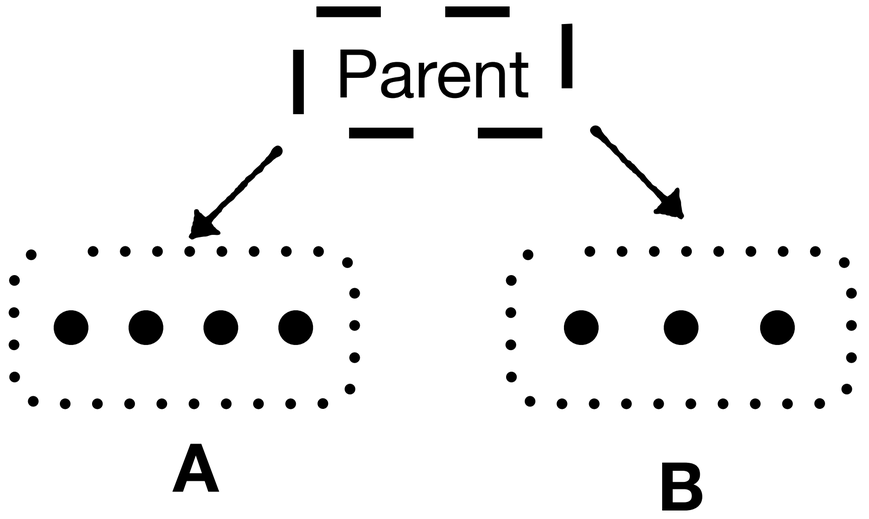


An entry may be inserted into an intersection

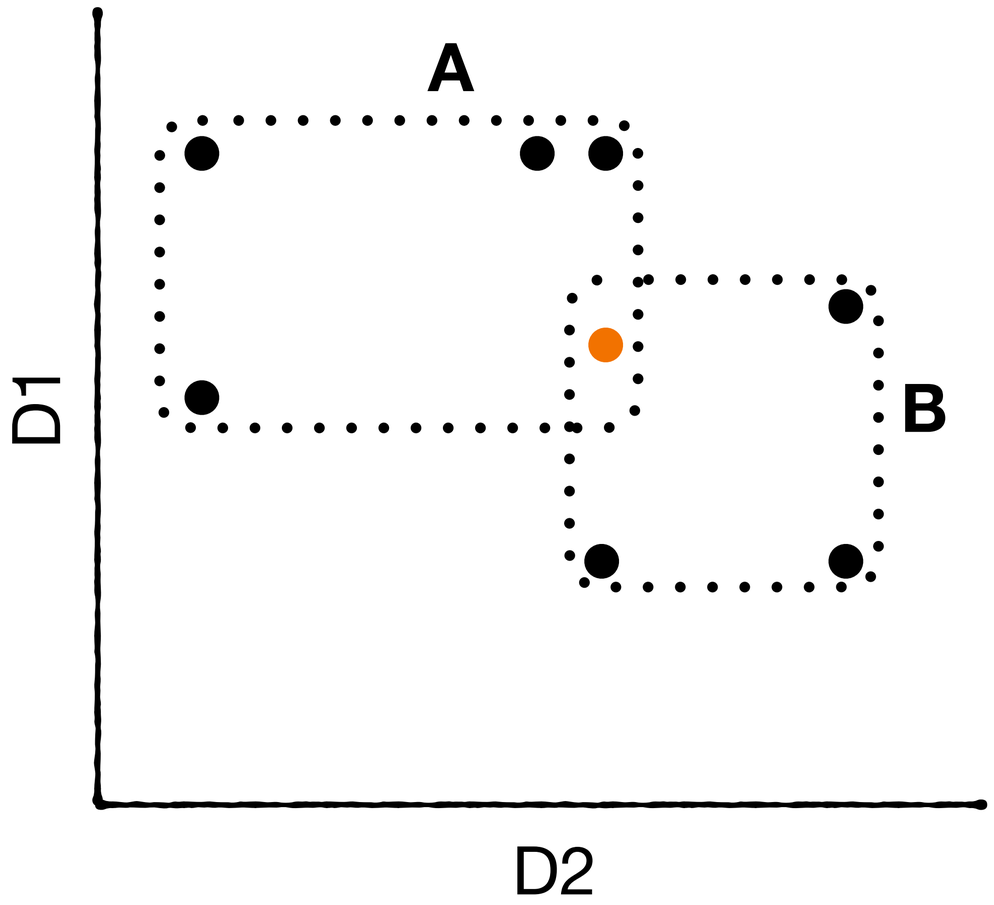


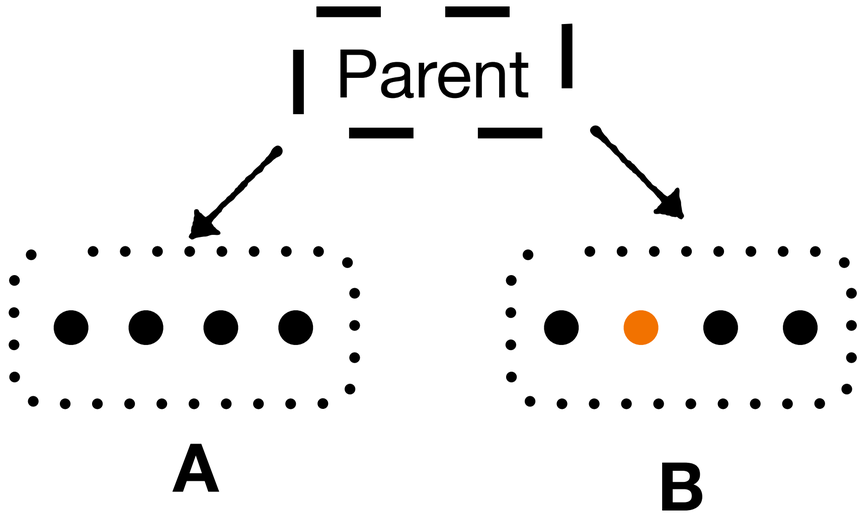
An entry may be inserted into an intersection

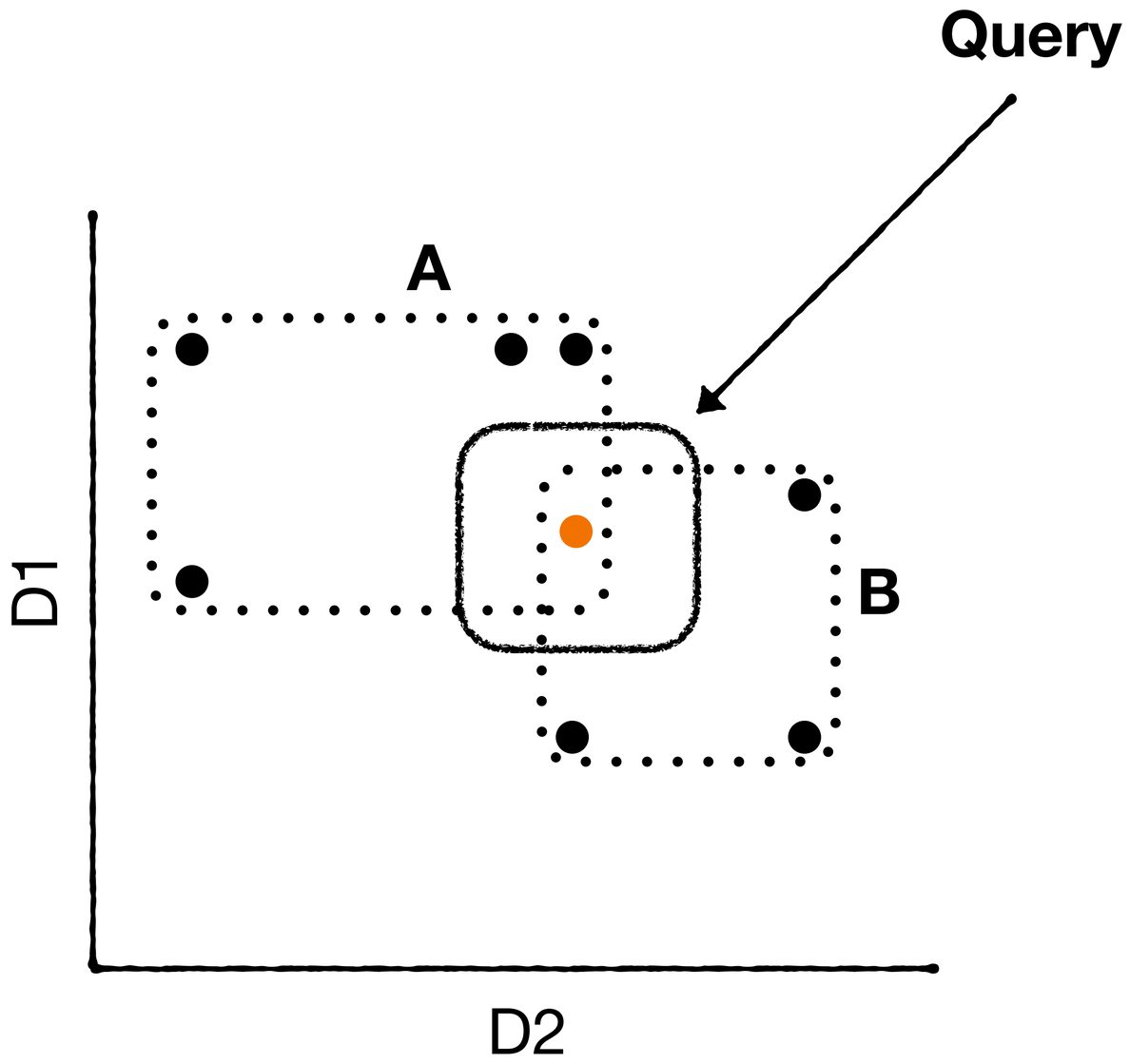


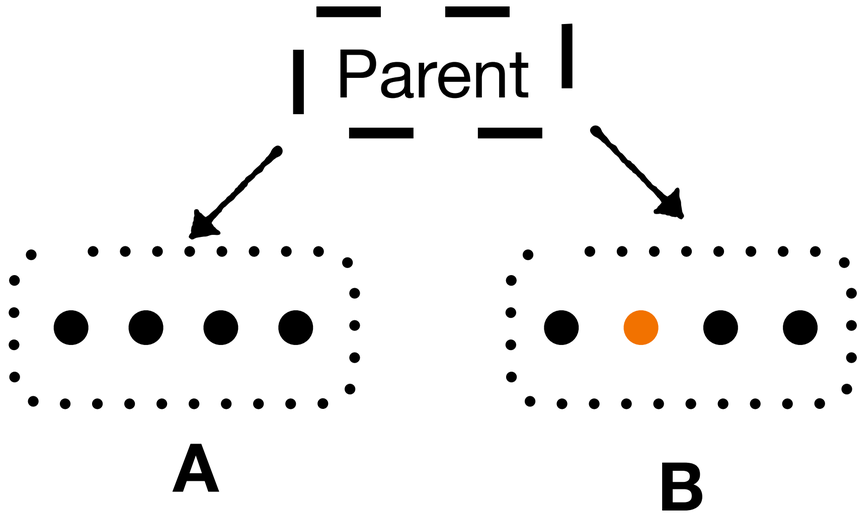


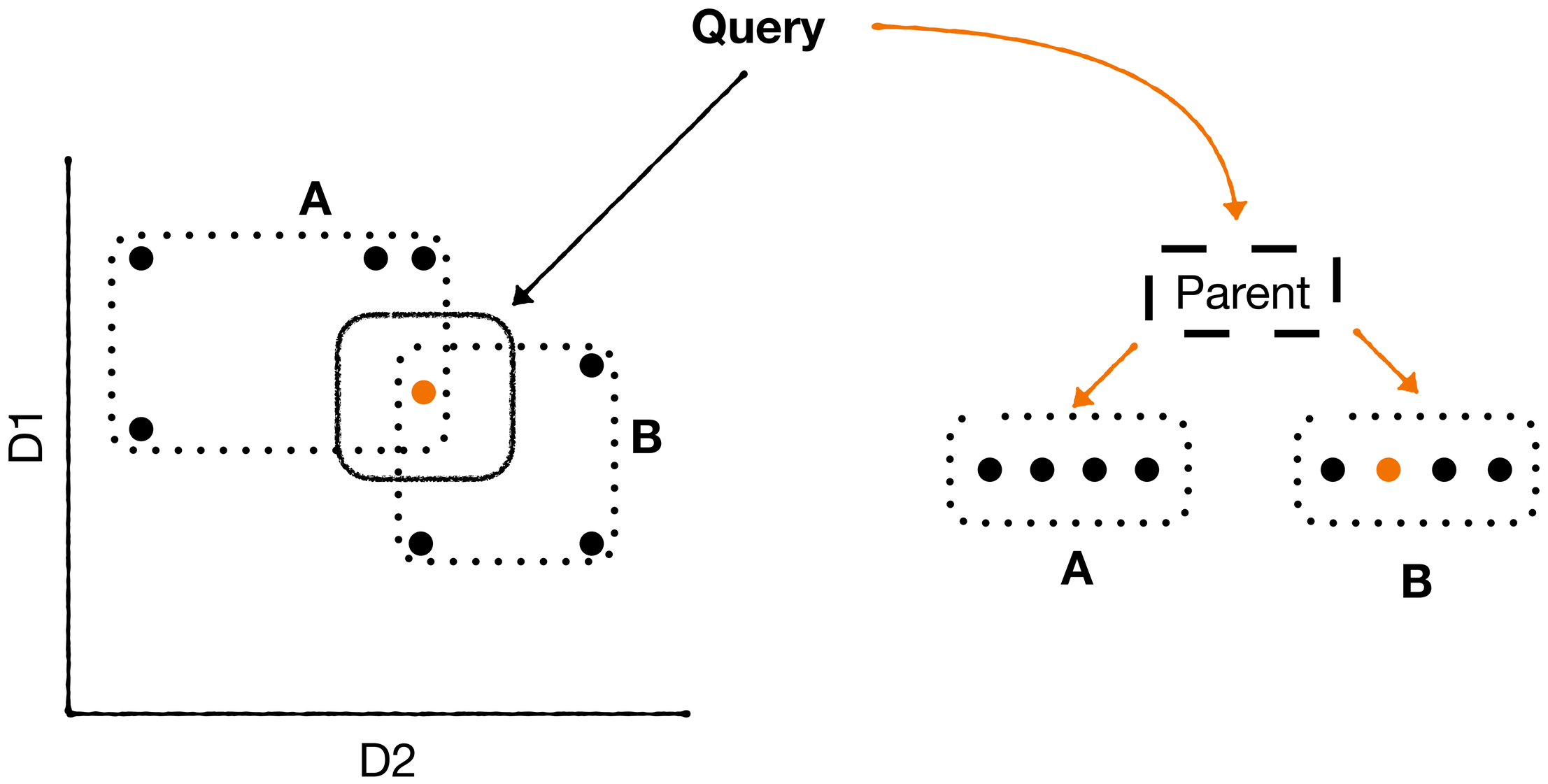
An entry may be inserted into an intersection

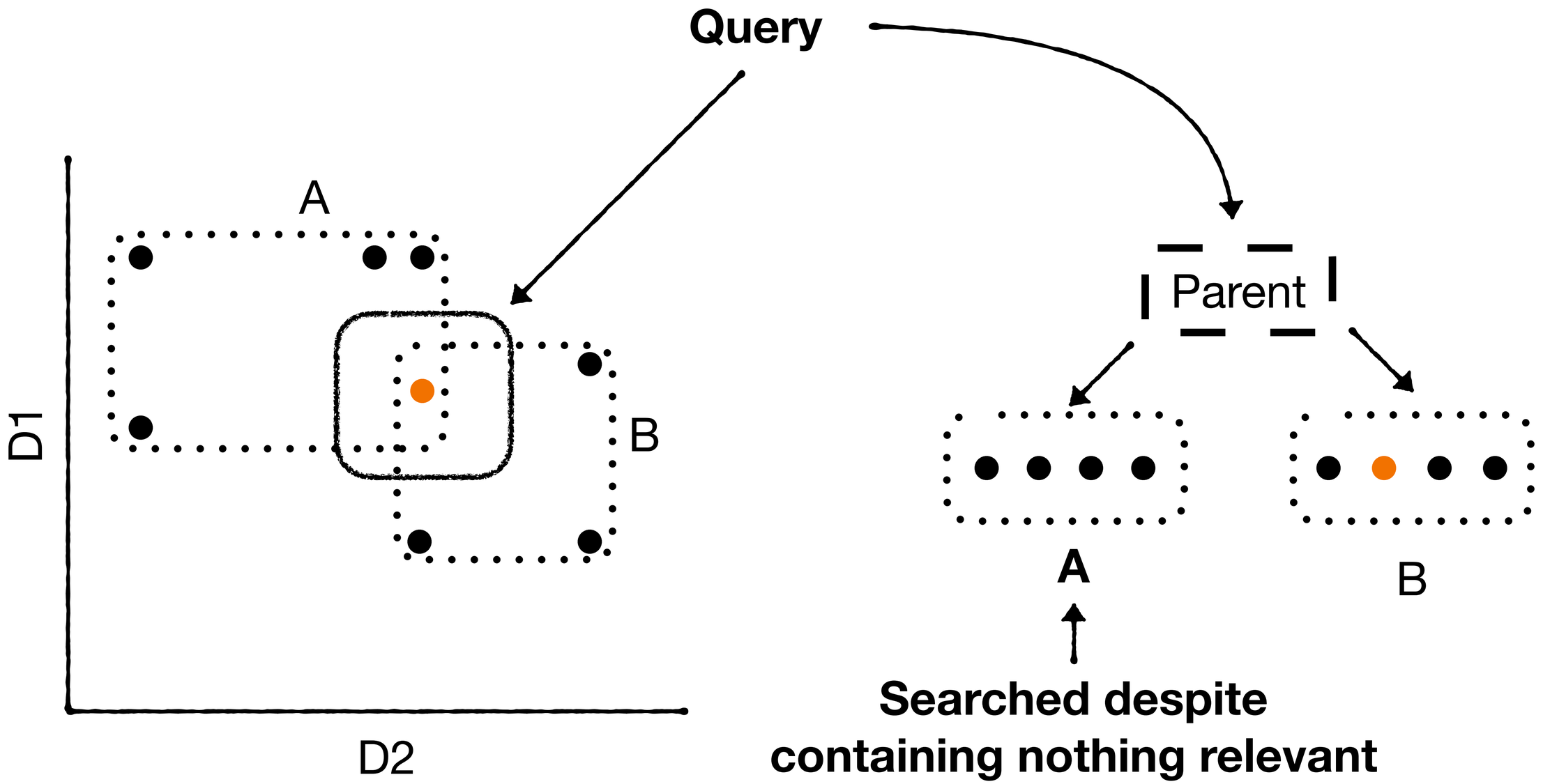





Query
A
B
D1
D2

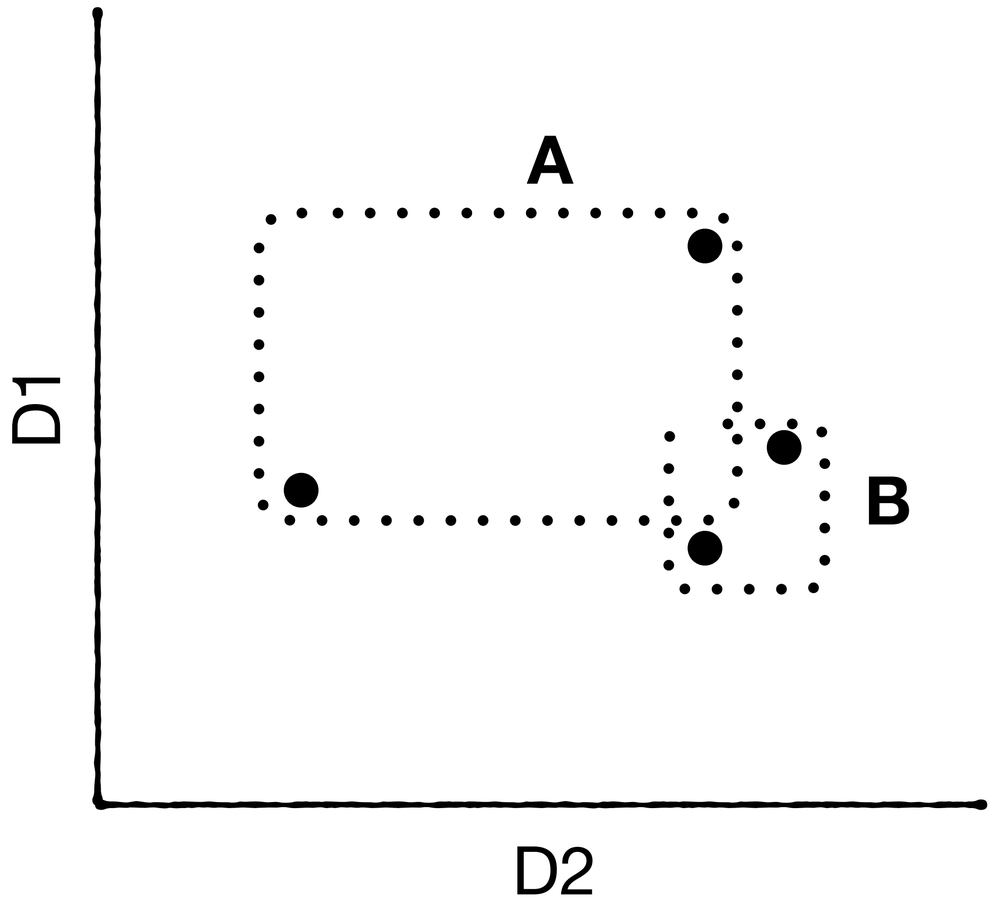

Parent
A
B

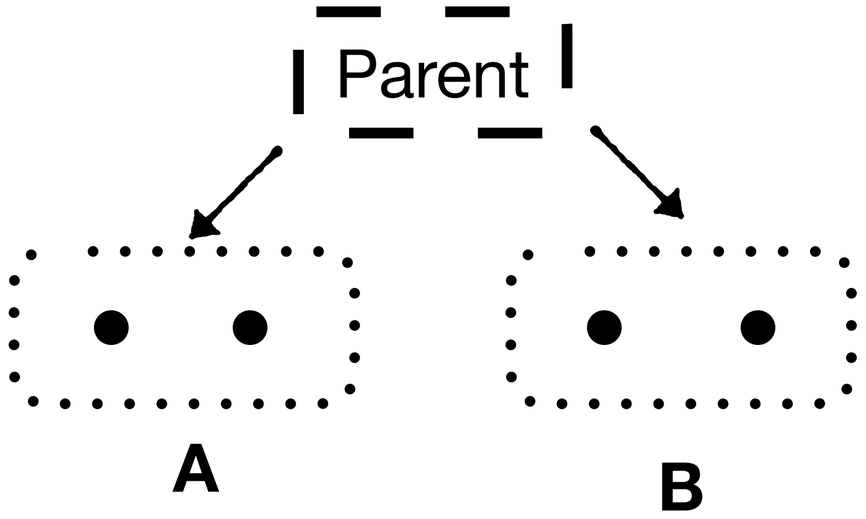

Query
A
B
D1
D2
Parent
A
B


Query
A
B
D1
D2
Parent
A
B
Searched despite
containing nothing relevant

Dual goal: maintain R-tree with least coverage & intersections :)

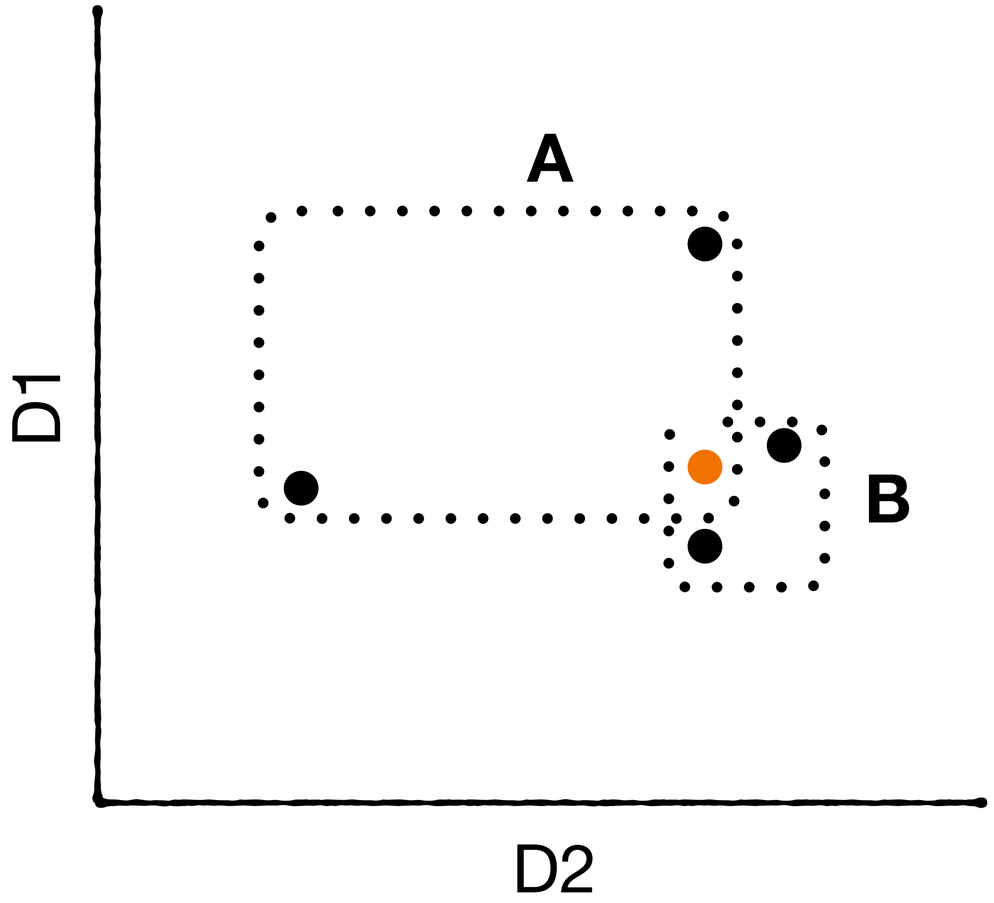
Suppose we insert entry into an intersection

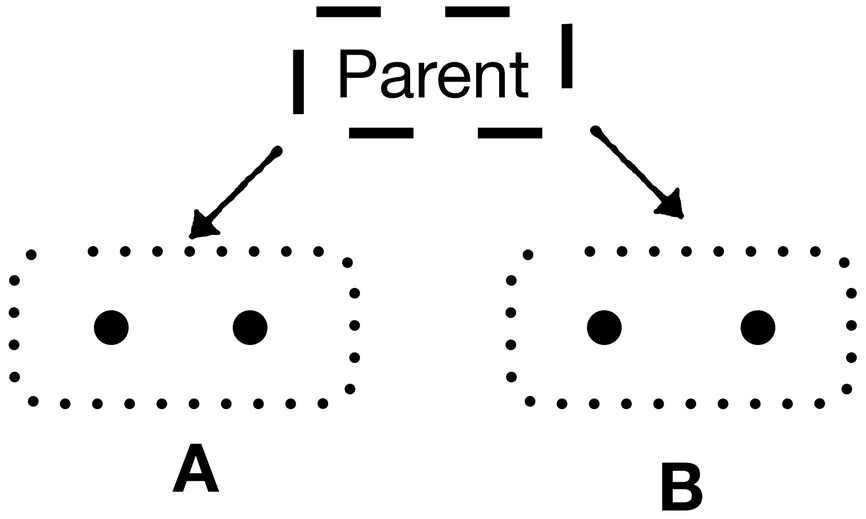




Suppose we insert entry into intersection

To which node should we add it?

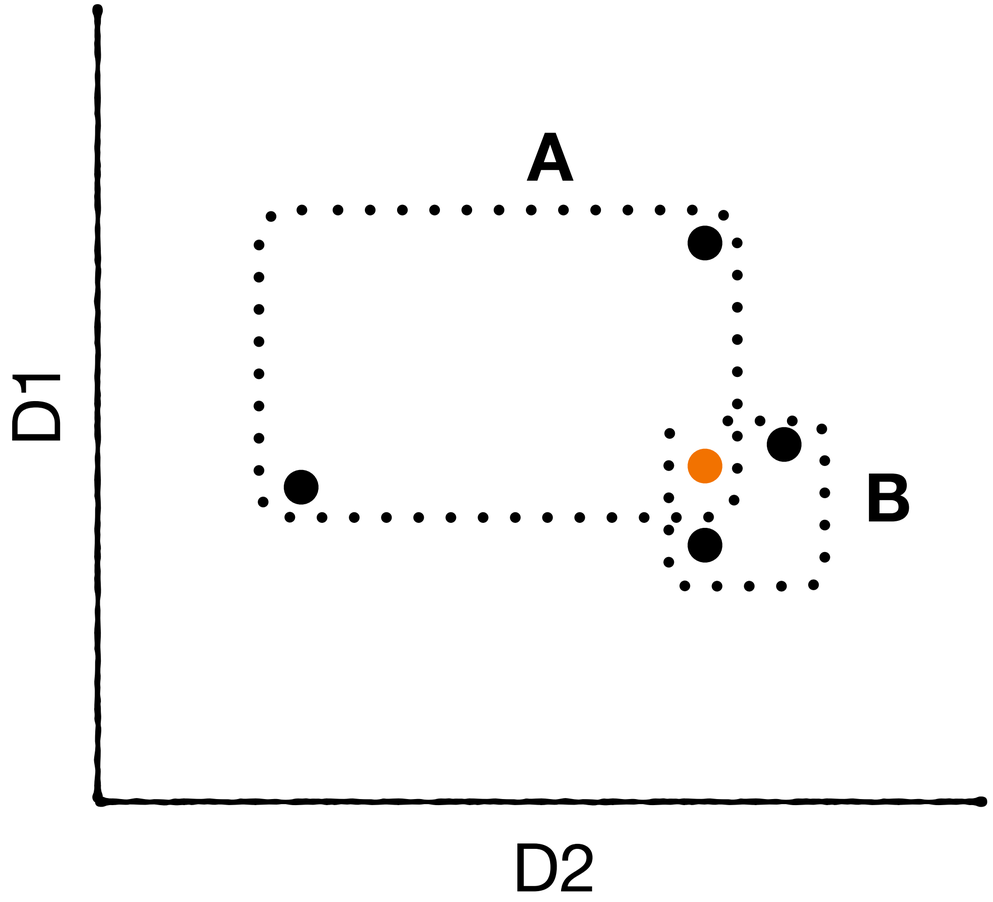


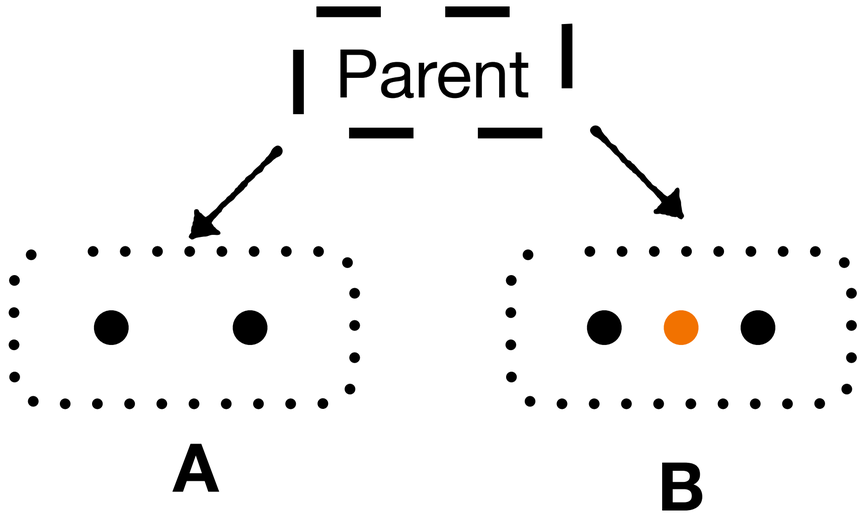


Suppose we insert entry into intersection

To which node should we add it?

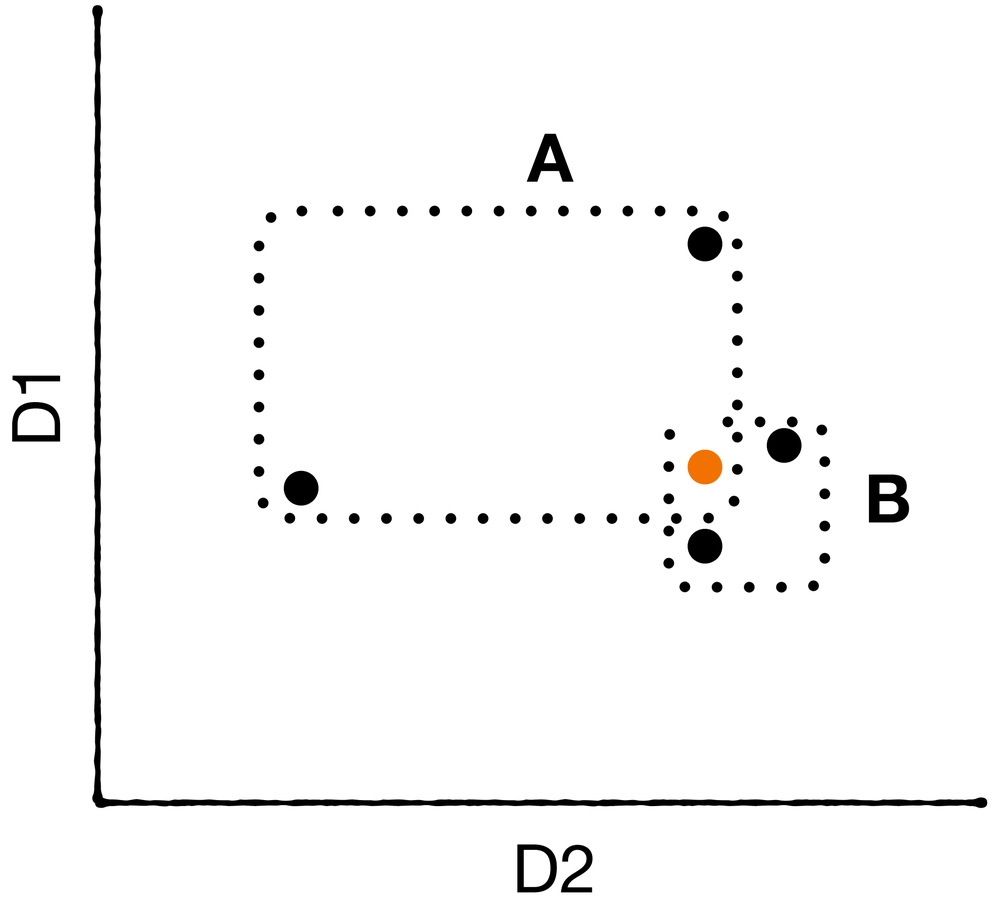
smaller intersecting rectangle

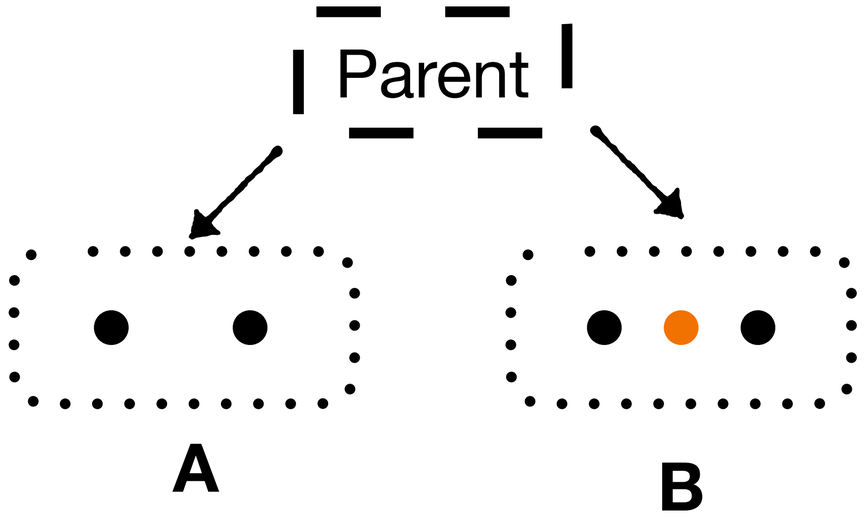




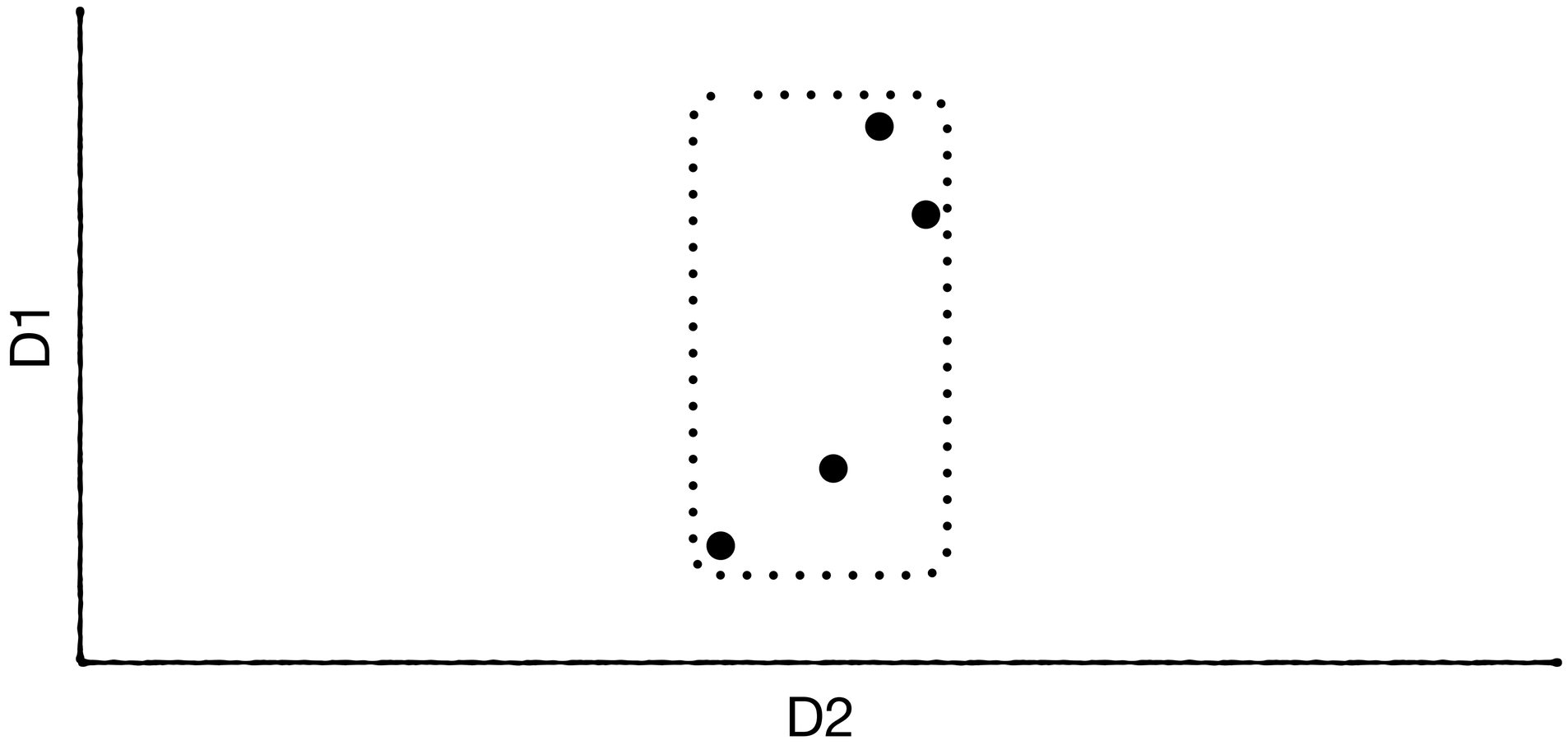
smaller intersecting rectangle

keeps points densely packed



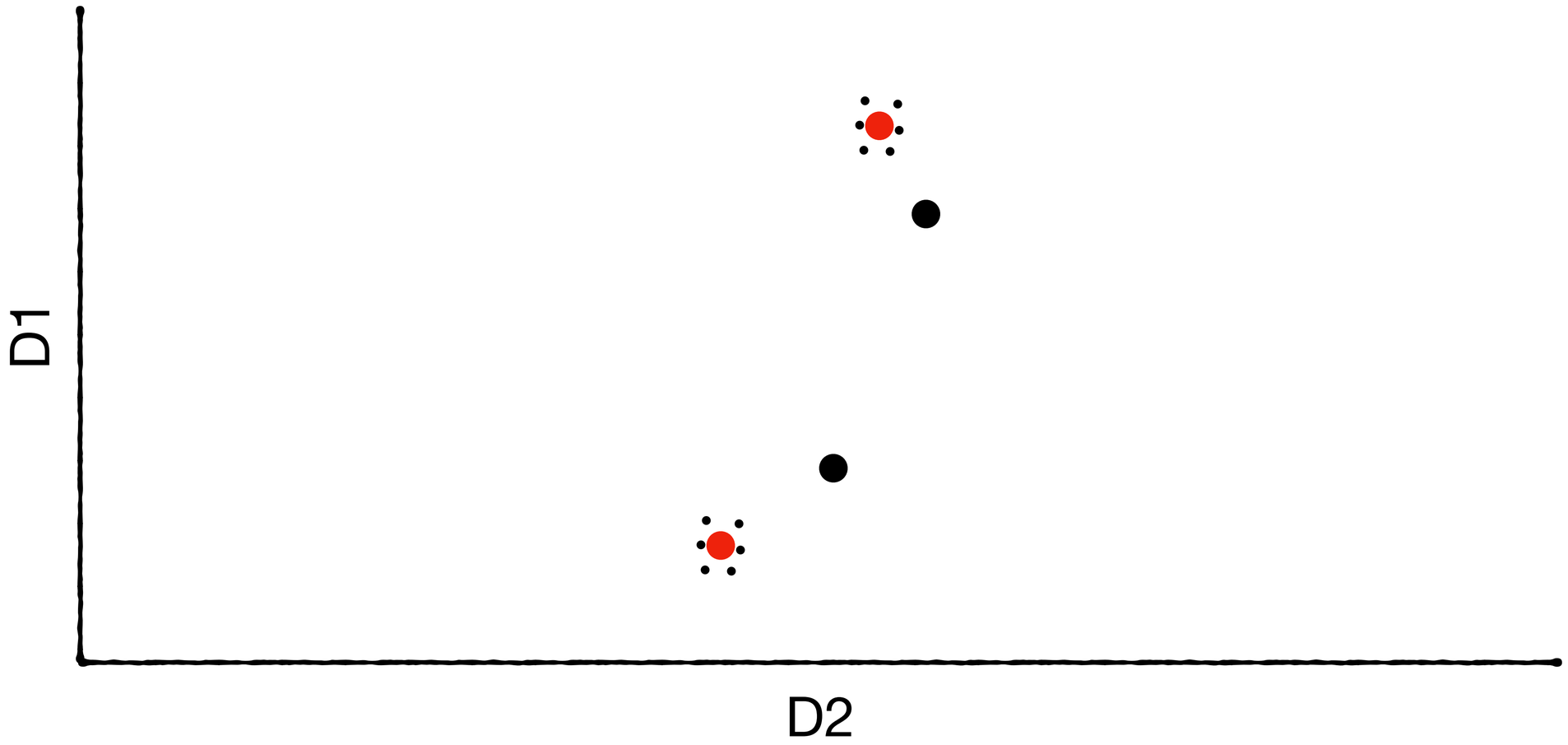


How to split a node?



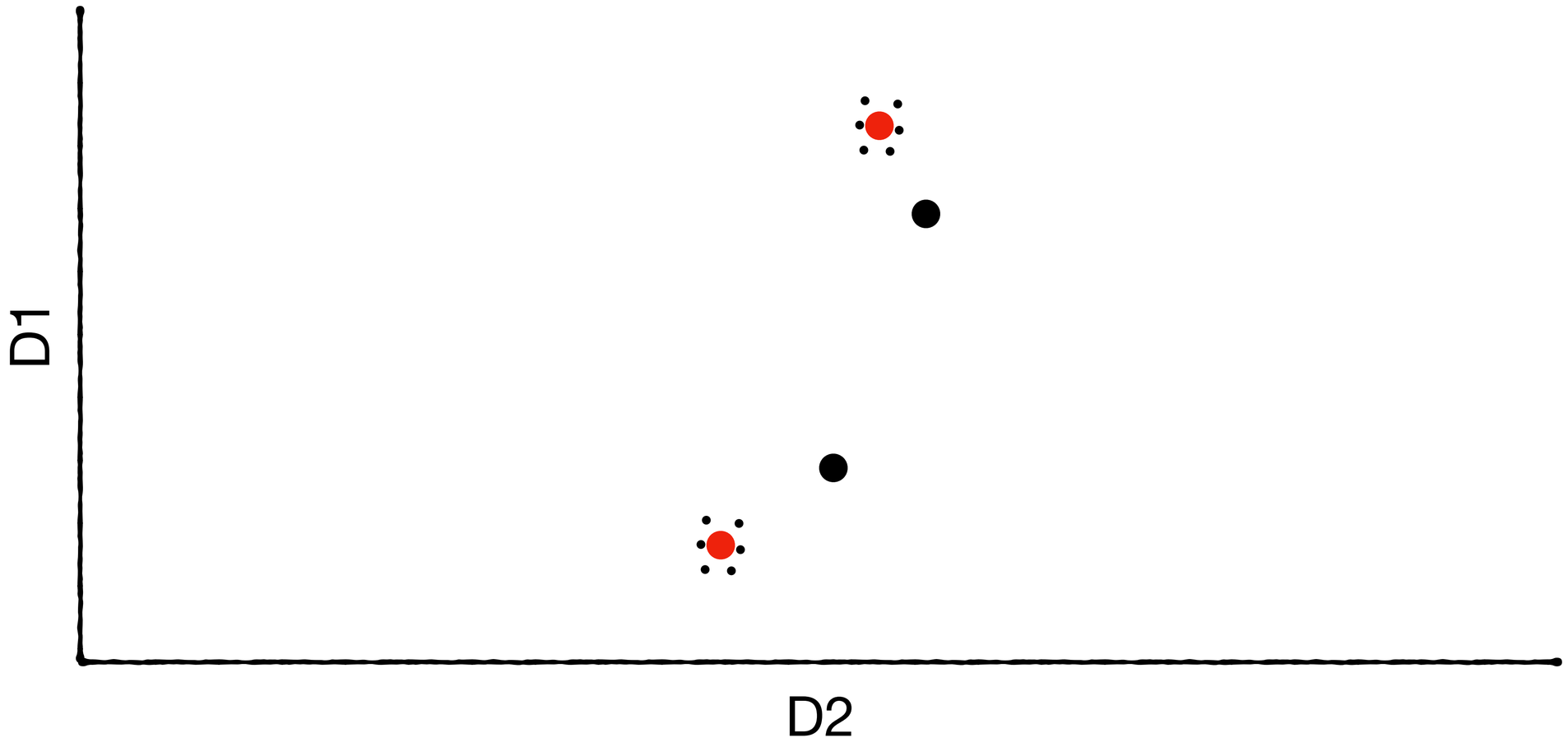
How to split a node?

Pick farthest apart pair of entries as new rectangles



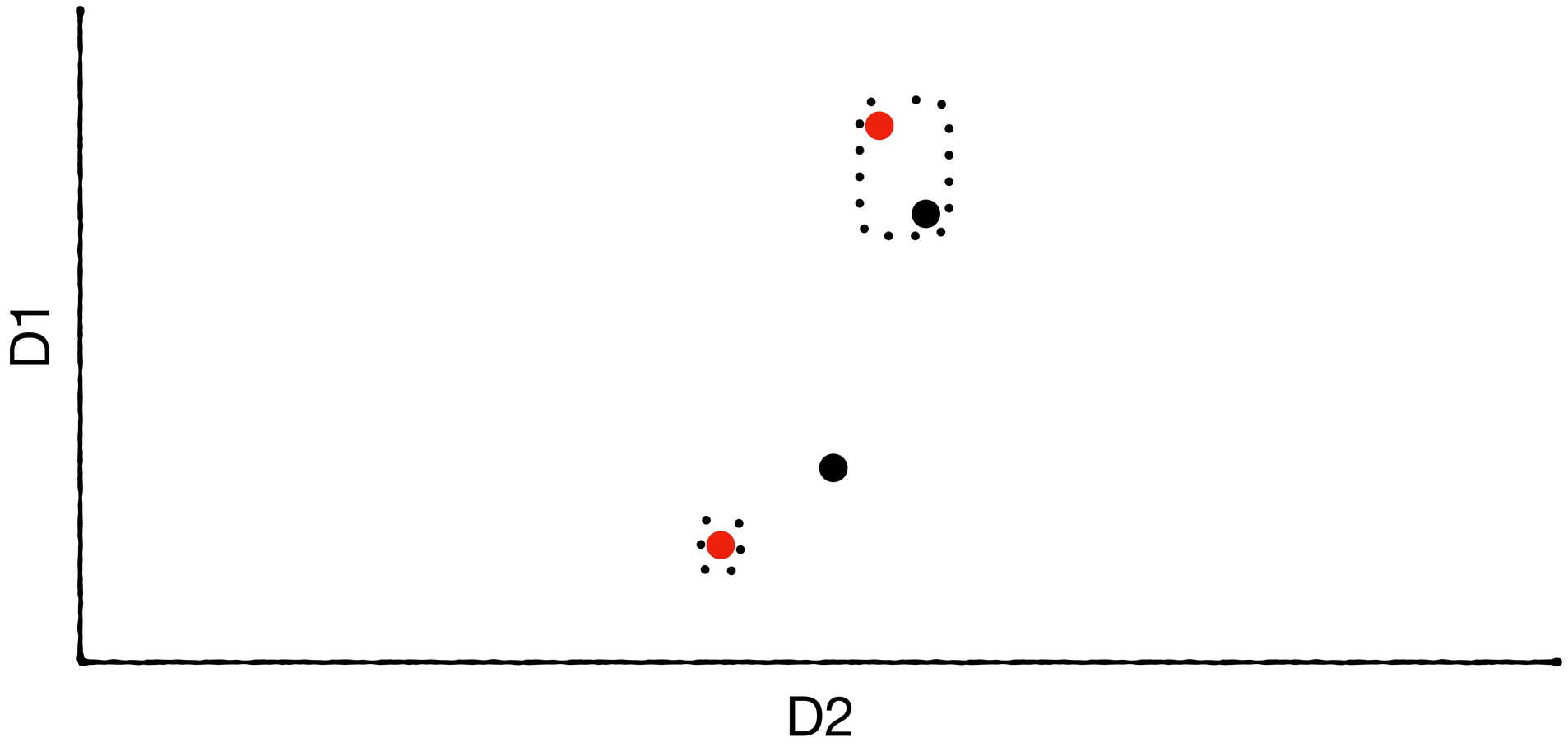
Pick farthest apart pair of entries as new rectangles

Identify next point that would result in minimum expansion for one of the rectangles



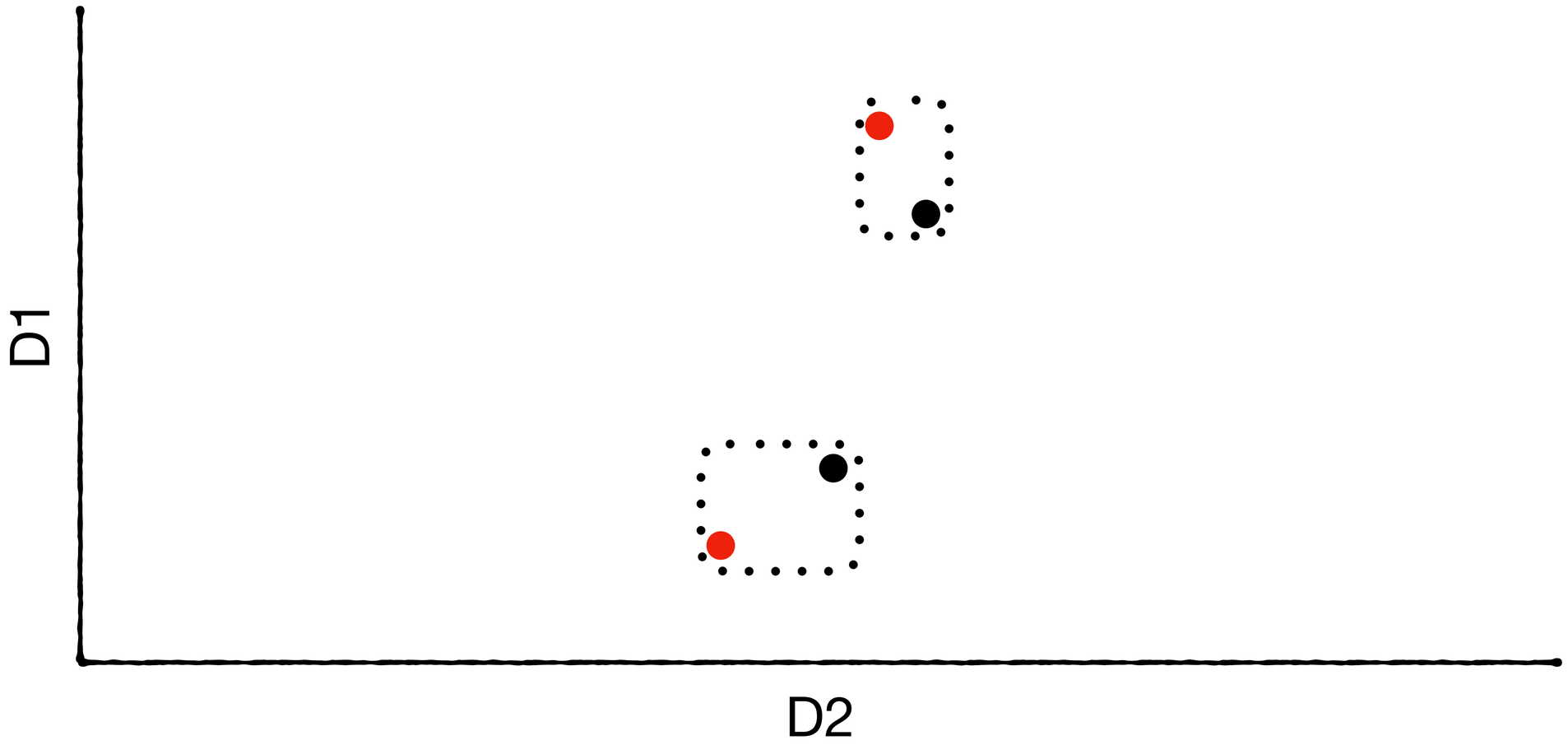
Pick farthest apart pair of entries as new rectangles

Identify next point that would result in minimum expansion for one of the rectangles



Pick farthest apart pair of entries as new rectangles

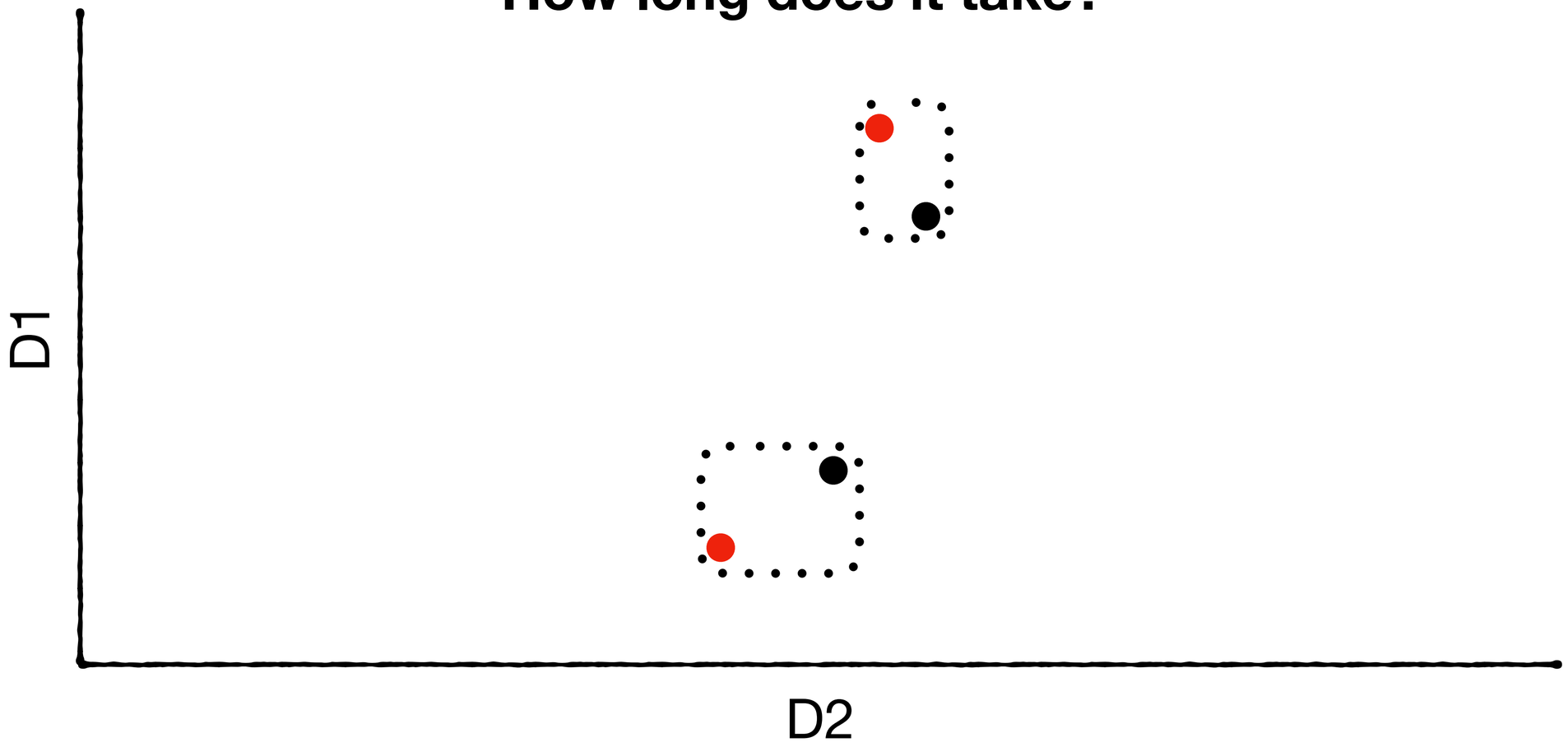
Identify next point that would result in minimum expansion for one of the rectangles



Pick farthest apart pair of entries as new rectangles

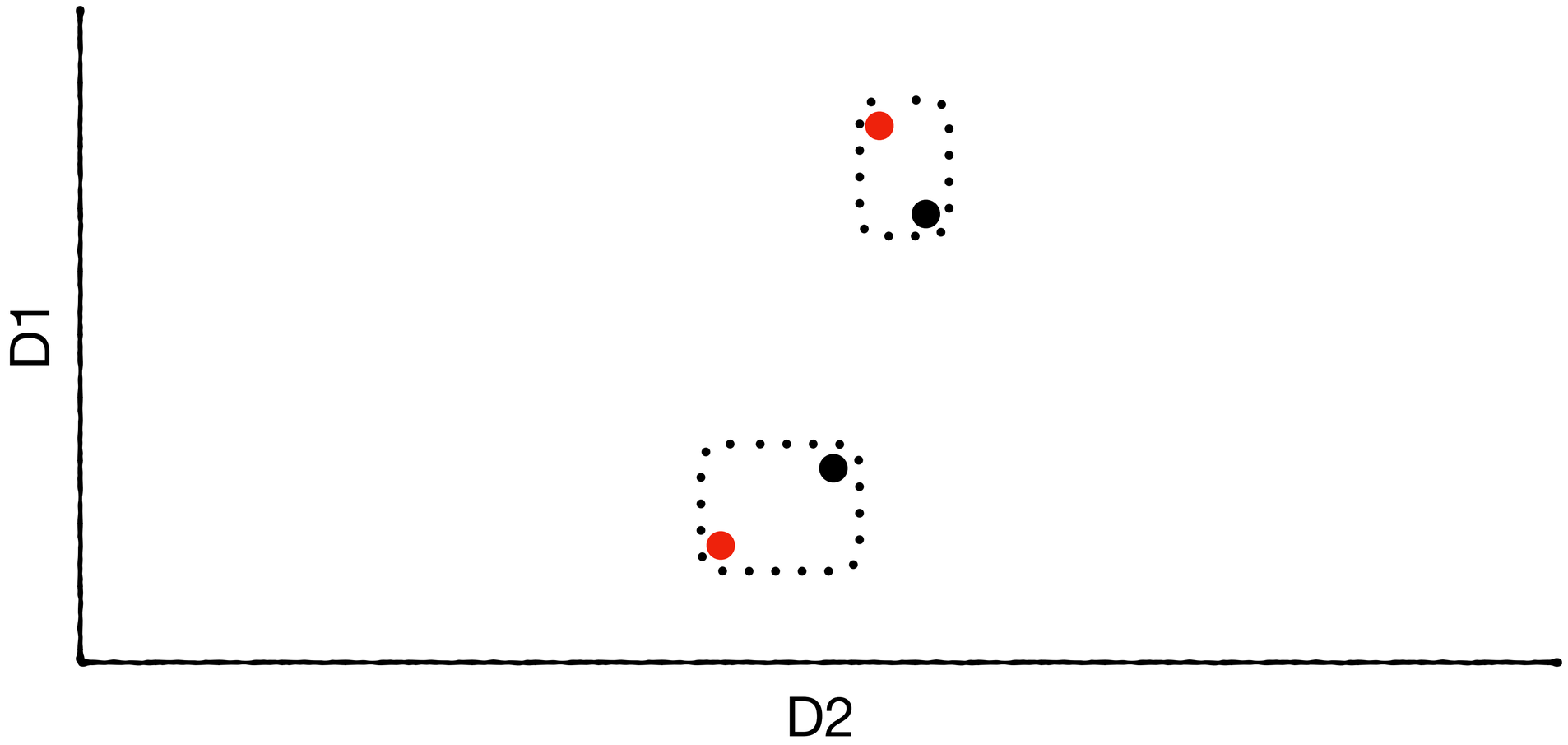
Identify next point that would result in minimum expansion for one of the rectangles

How long does it take?



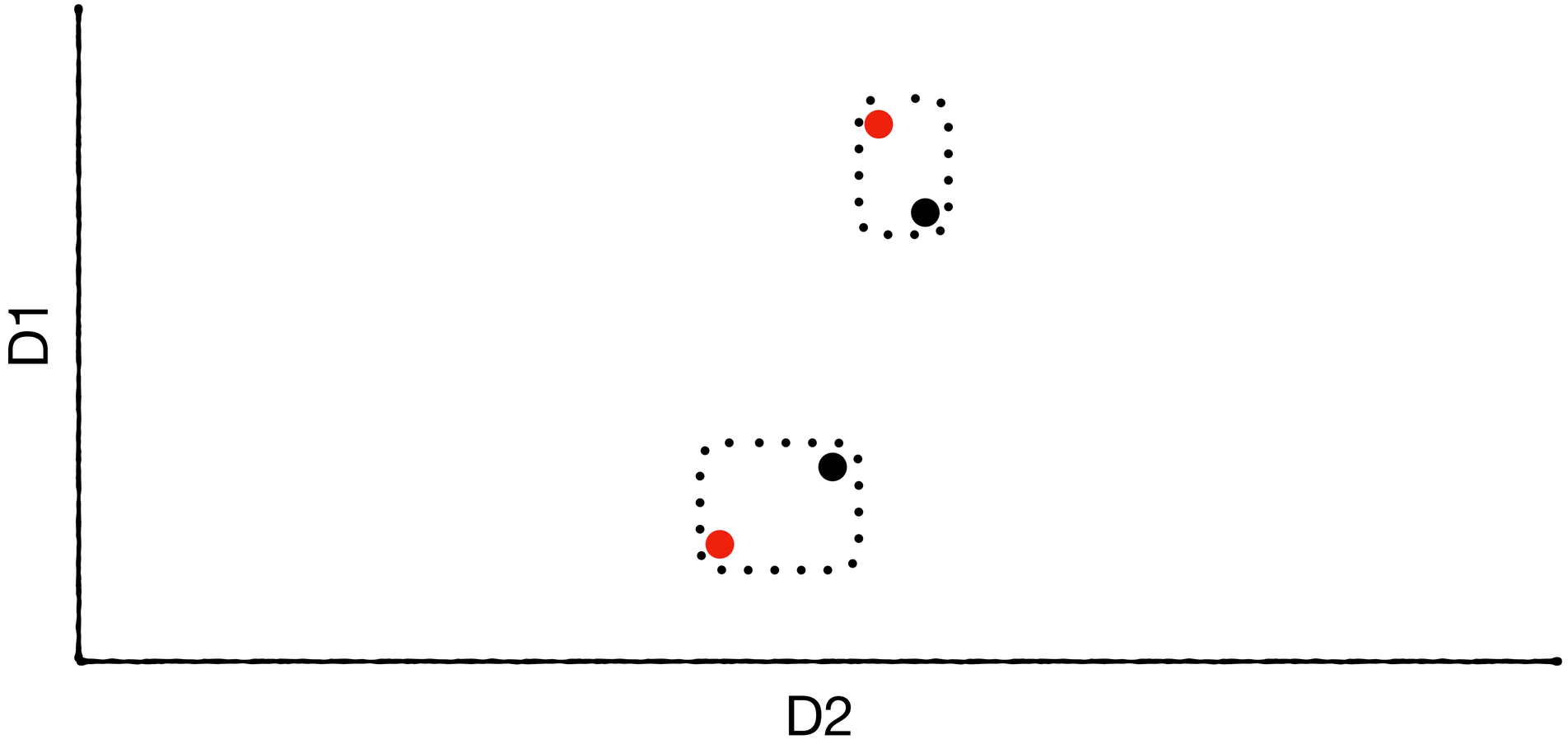
How long does it take?

$O(B^2)$ CPU cycles

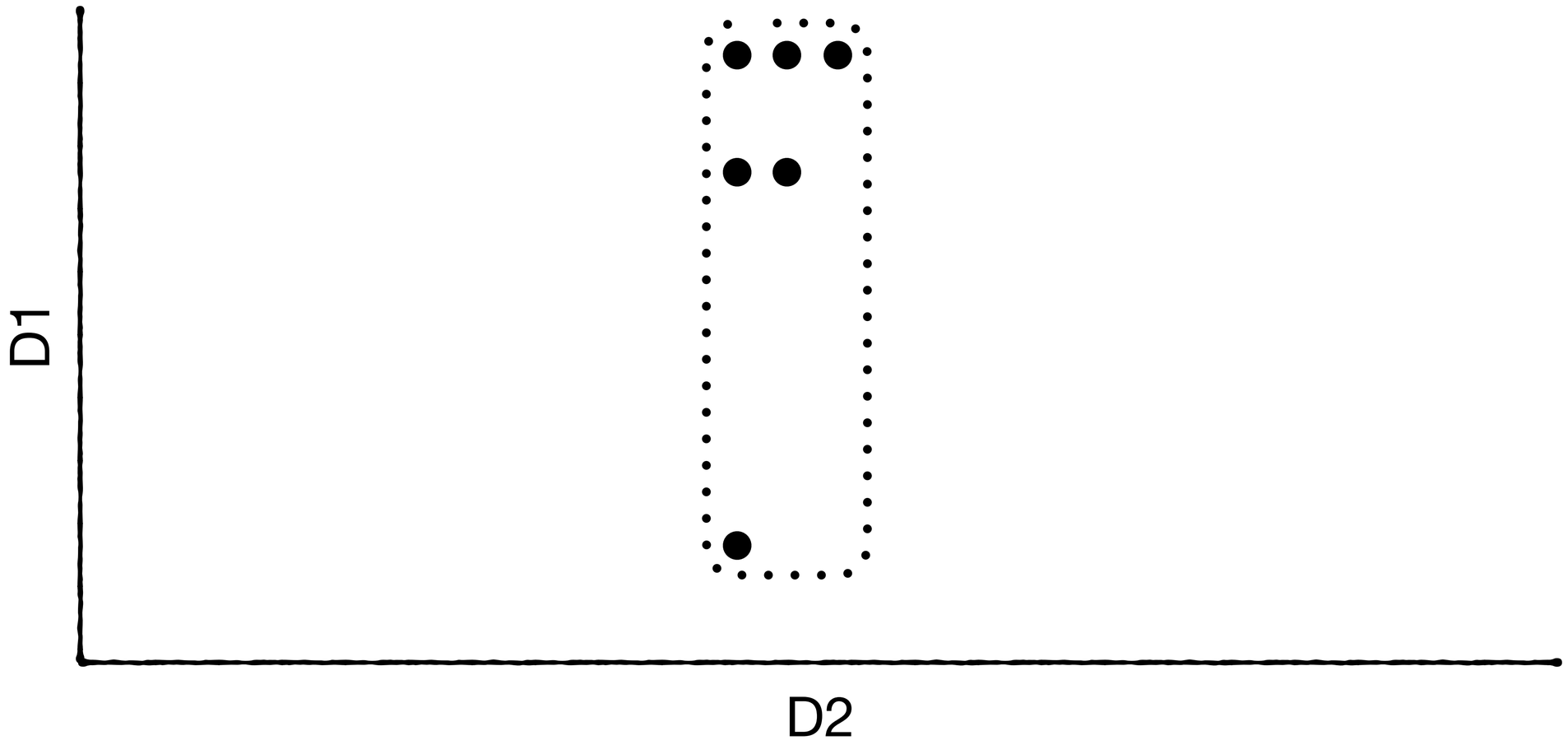


How long does it take?

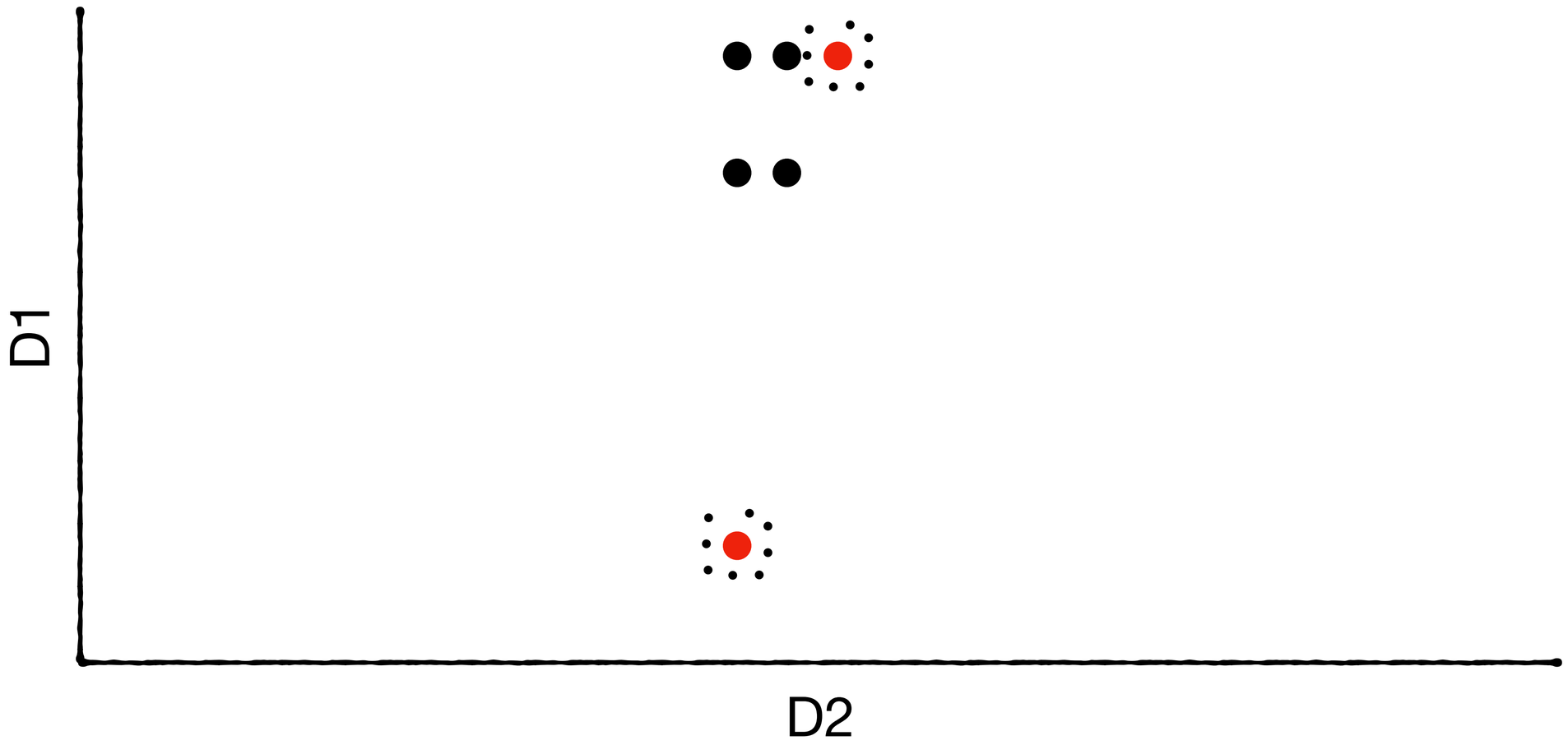
O(B^2) CPU cycles - **referred to as Quadratic Split**



Consider a more nuanced split example

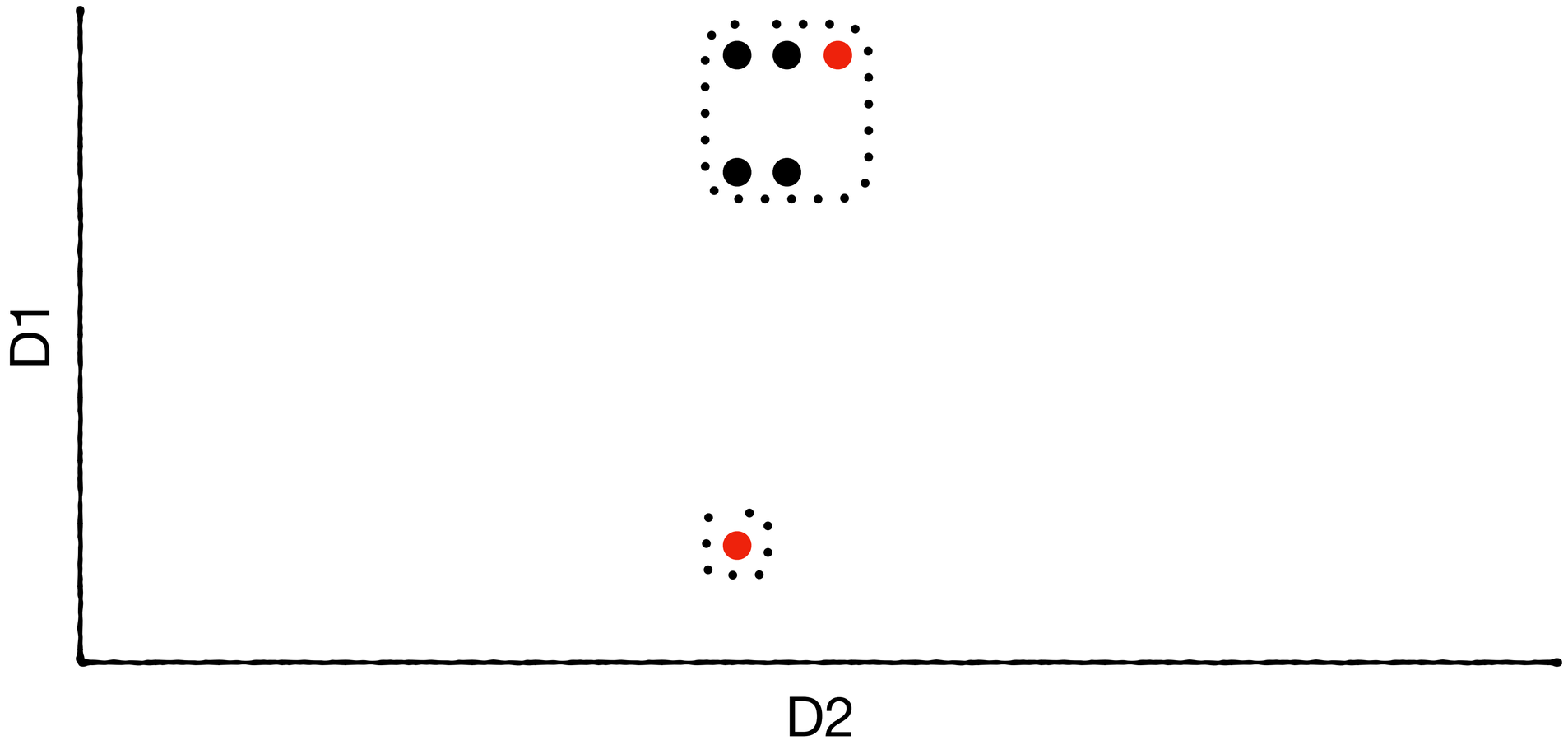


Find farthest apart points



Find farthest apart points

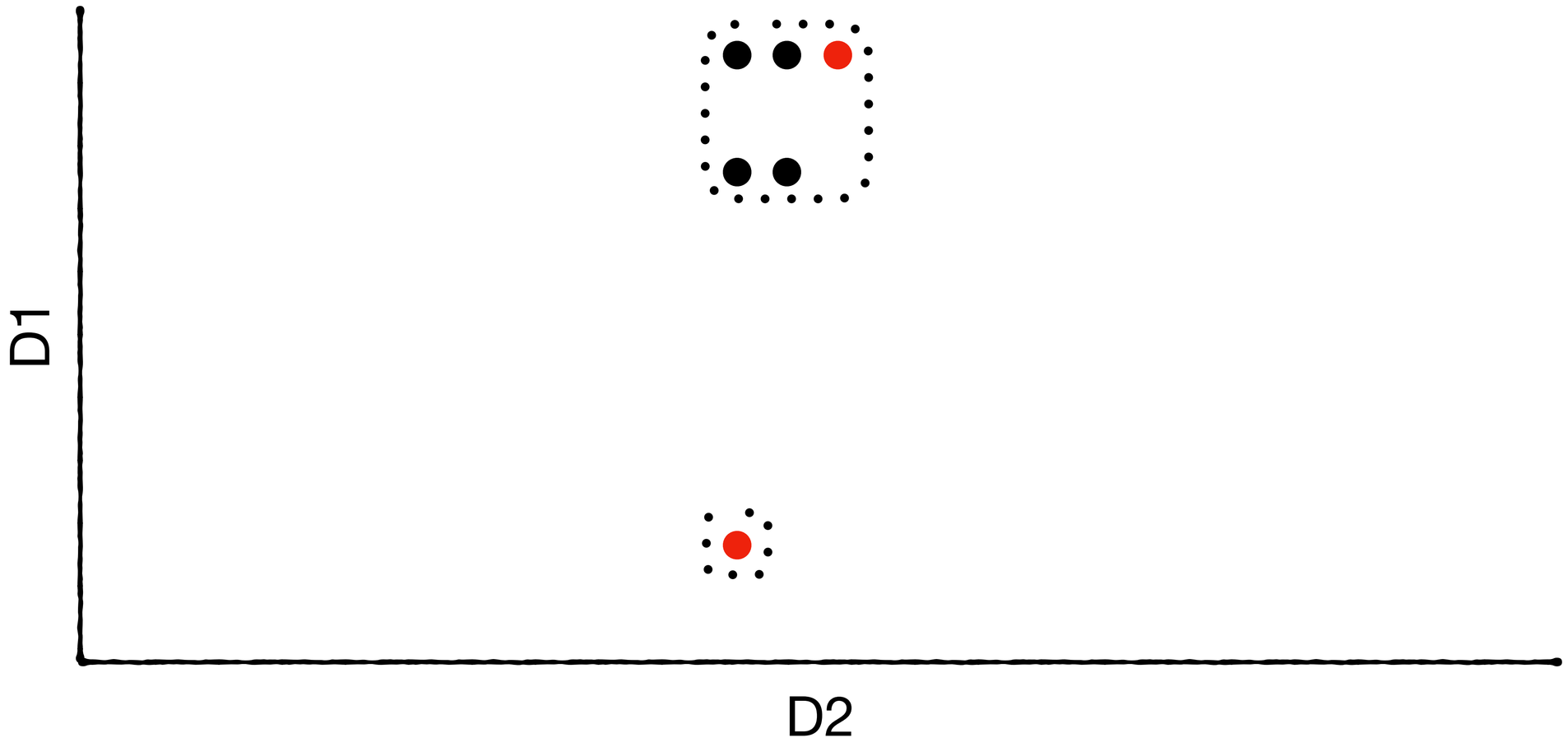
If we apply simple quadratic split, all points would go in one node



Find farthest apart points

If we apply simple quadratic split, all points would go in one node

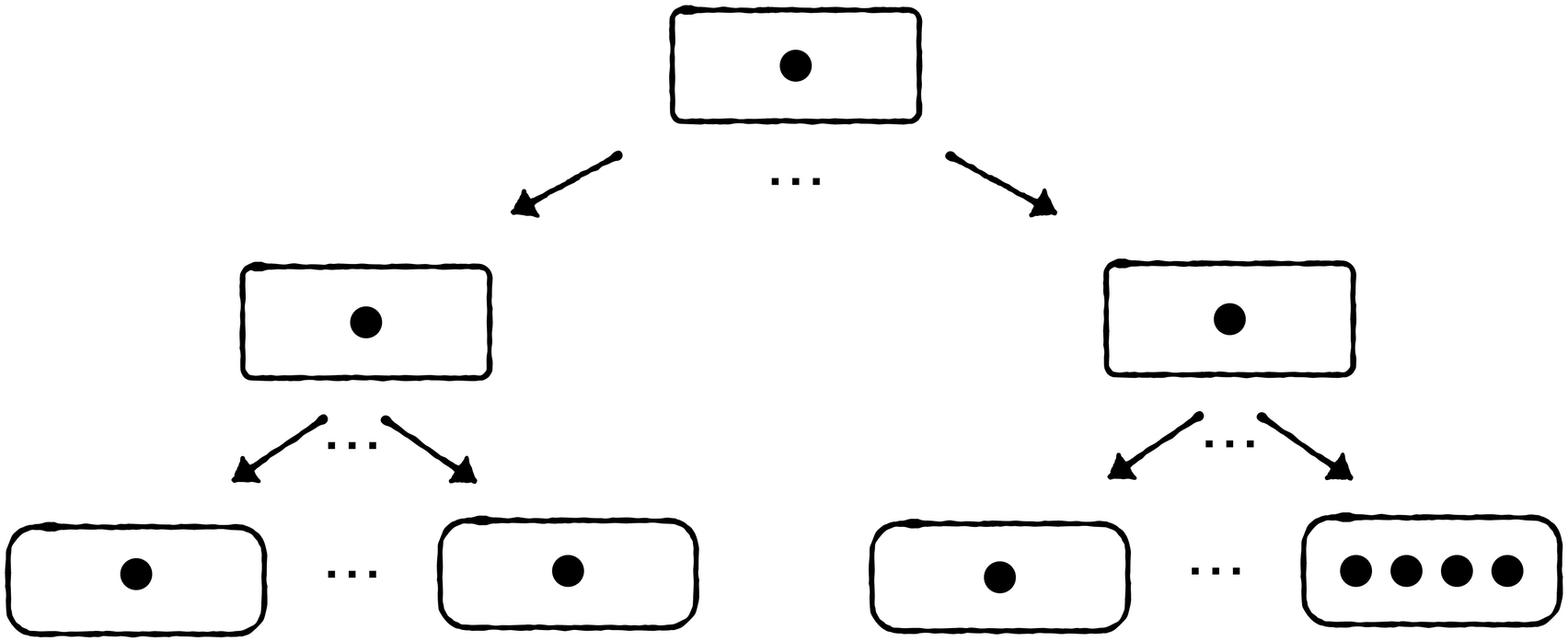
Why is this bad?



Find farthest apart points

Should we insert remaining points to closest node?

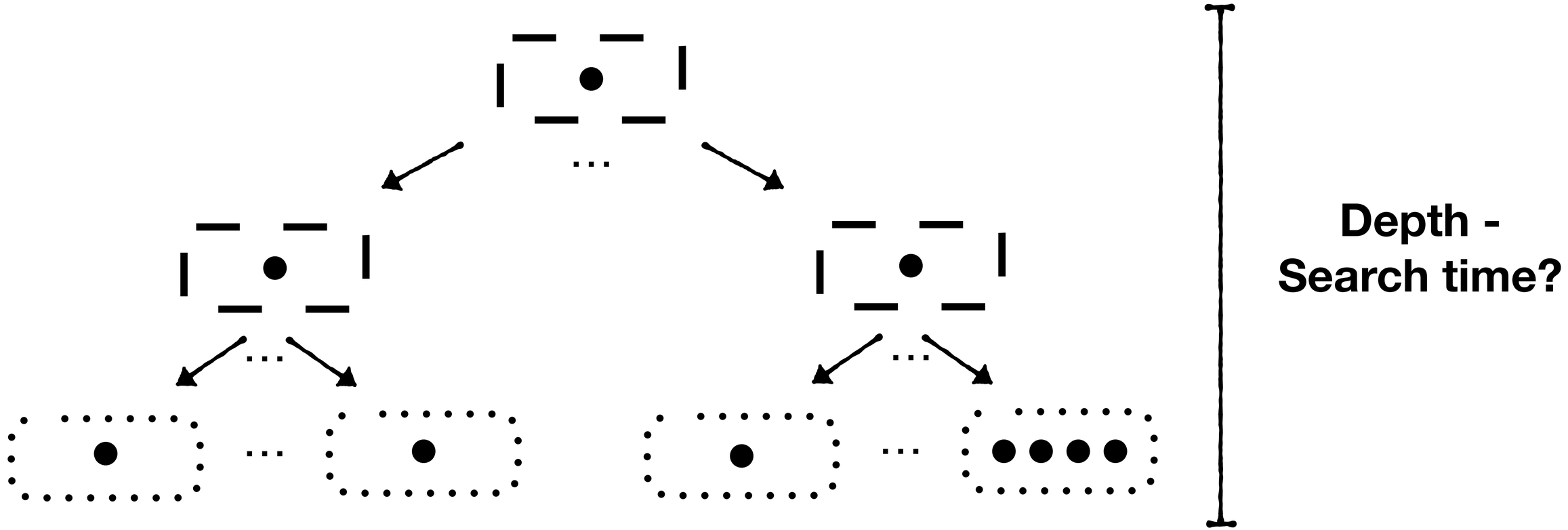
No as this would lead to an unbalanced tree.



Find farthest apart points

Should we insert remaining points to closest node?

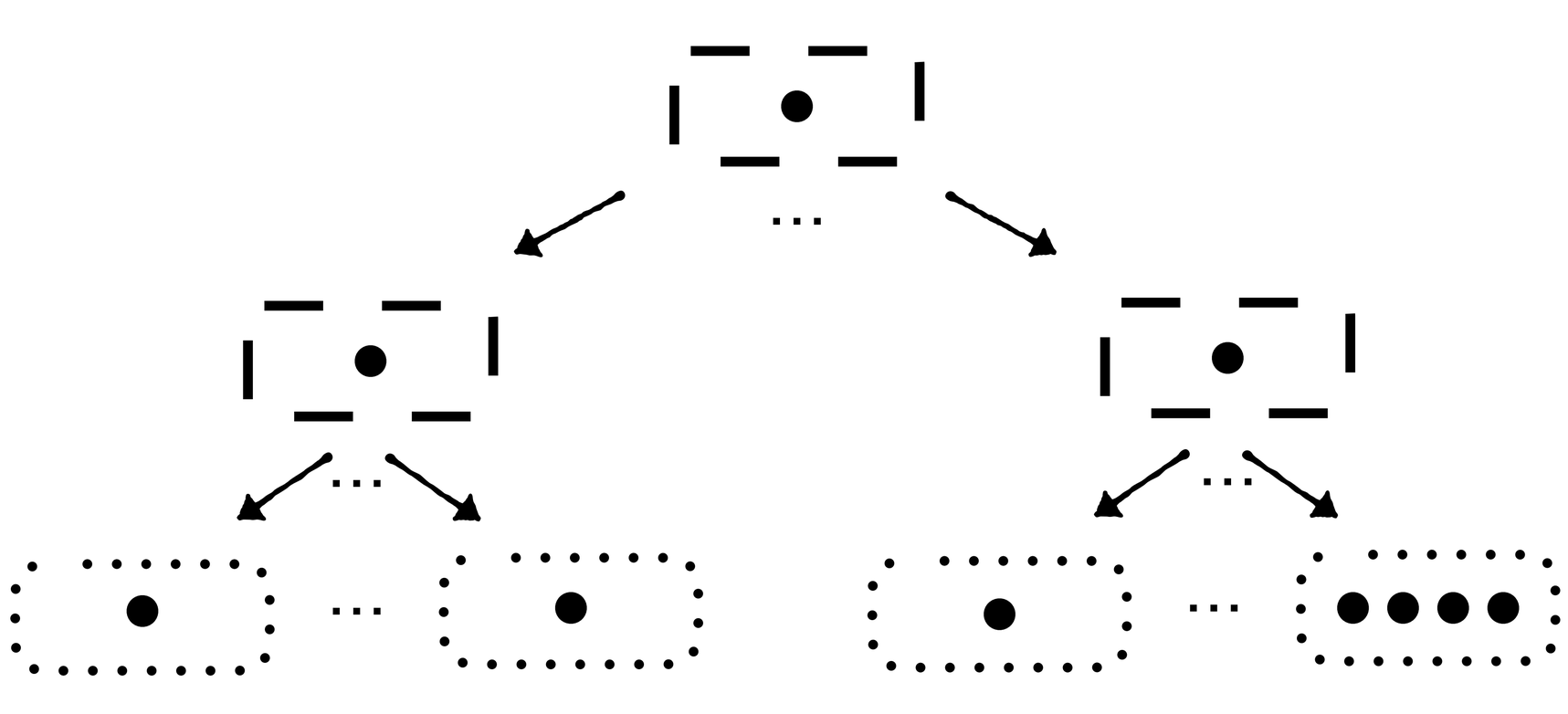
No as this would lead to an unbalanced tree.



Find farthest apart points

Should we insert remaining points to closest node?

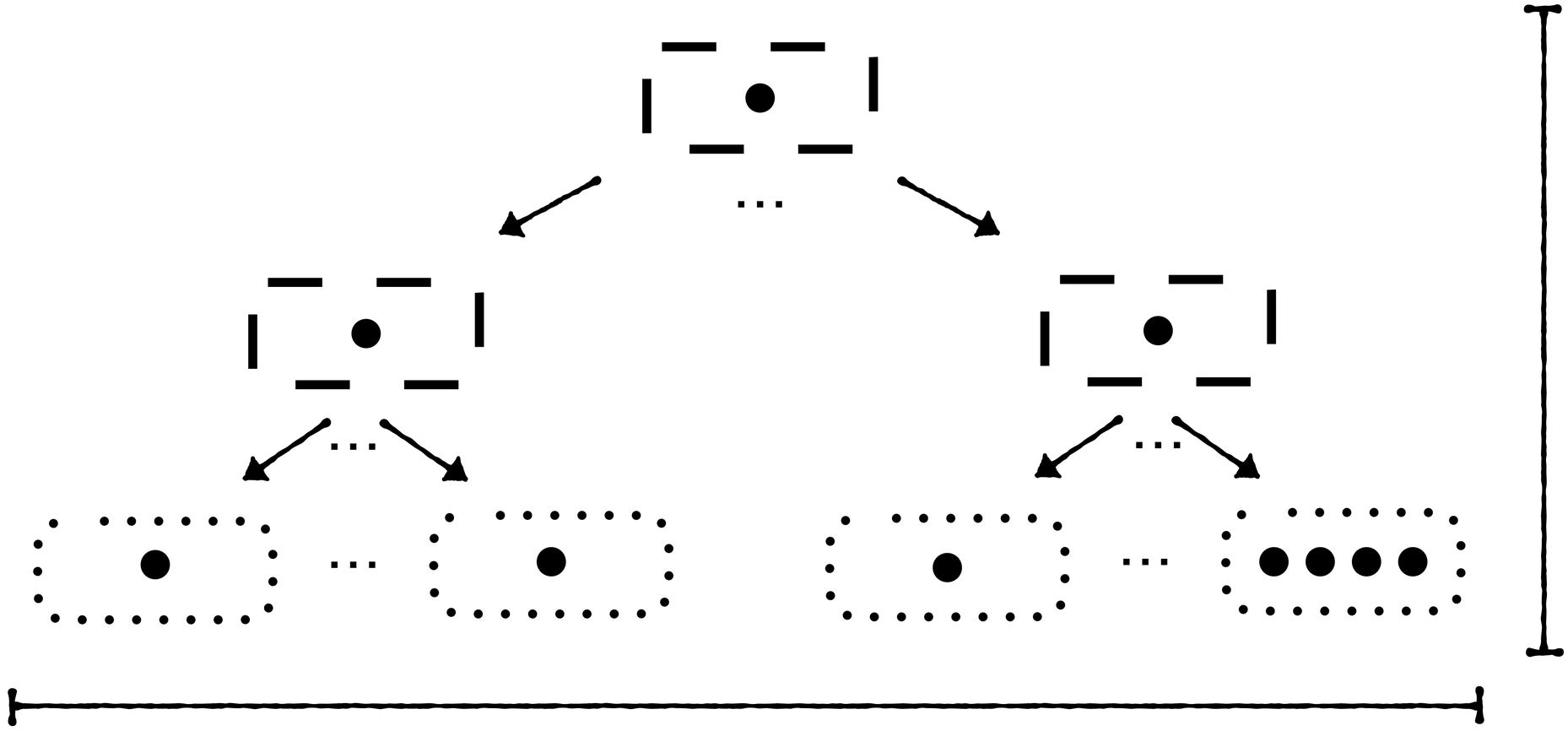
No as this would lead to an unbalanced tree.



Depth - Search time?

$O(\log_2(N))$

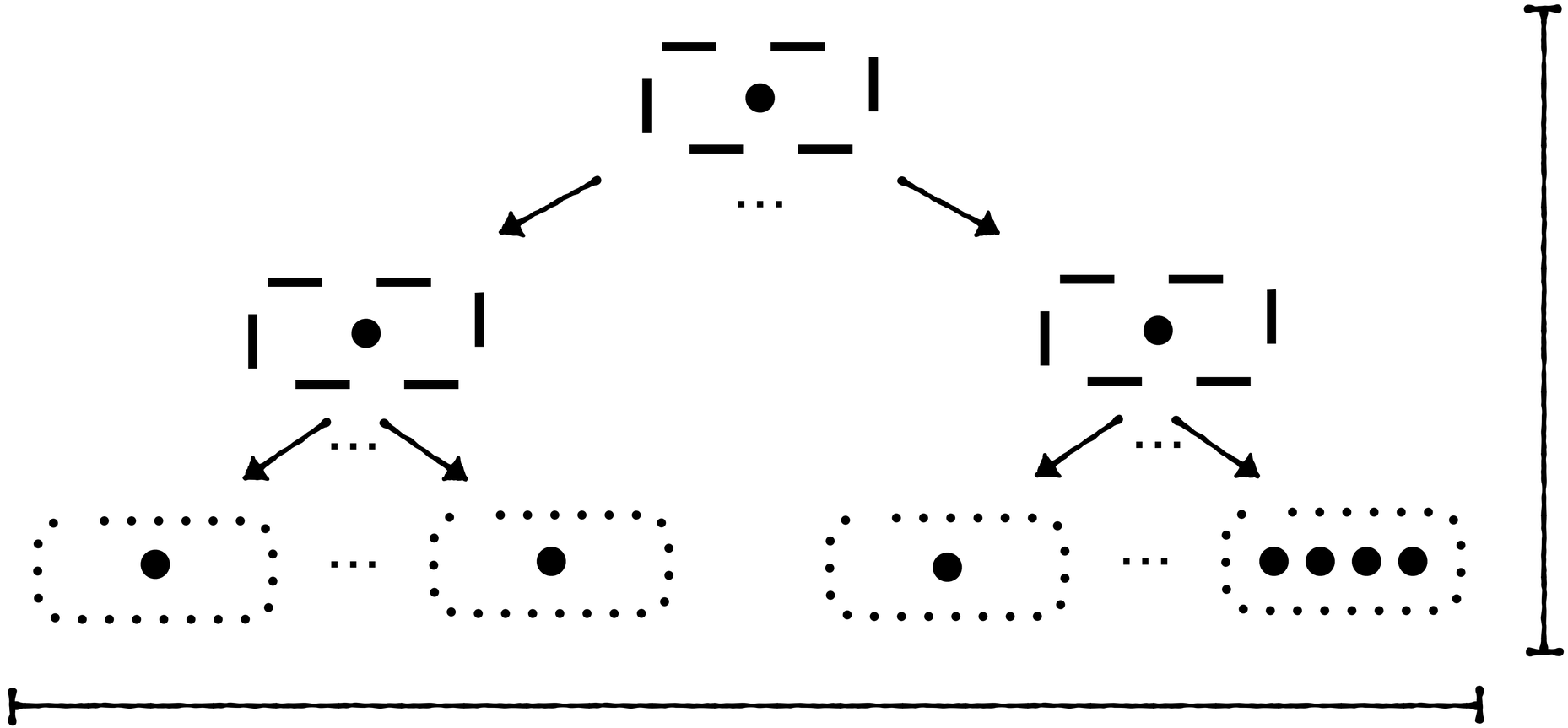
As opposed to $O(\log_B(N))$



Depth -
Search time?

$O(\log_B(N))$

Space-amplification?



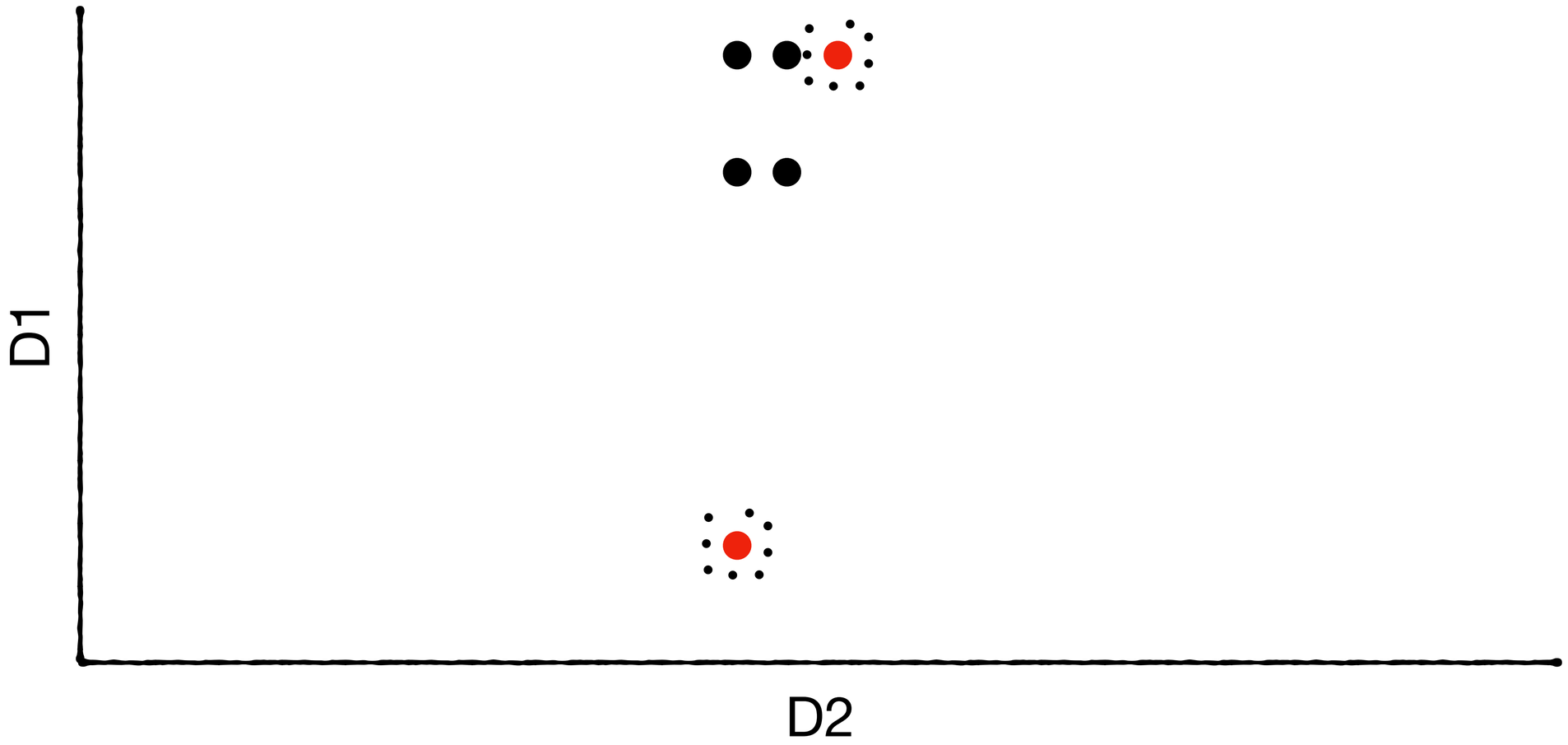
Depth -
Search time?

$O(\log_2(N))$

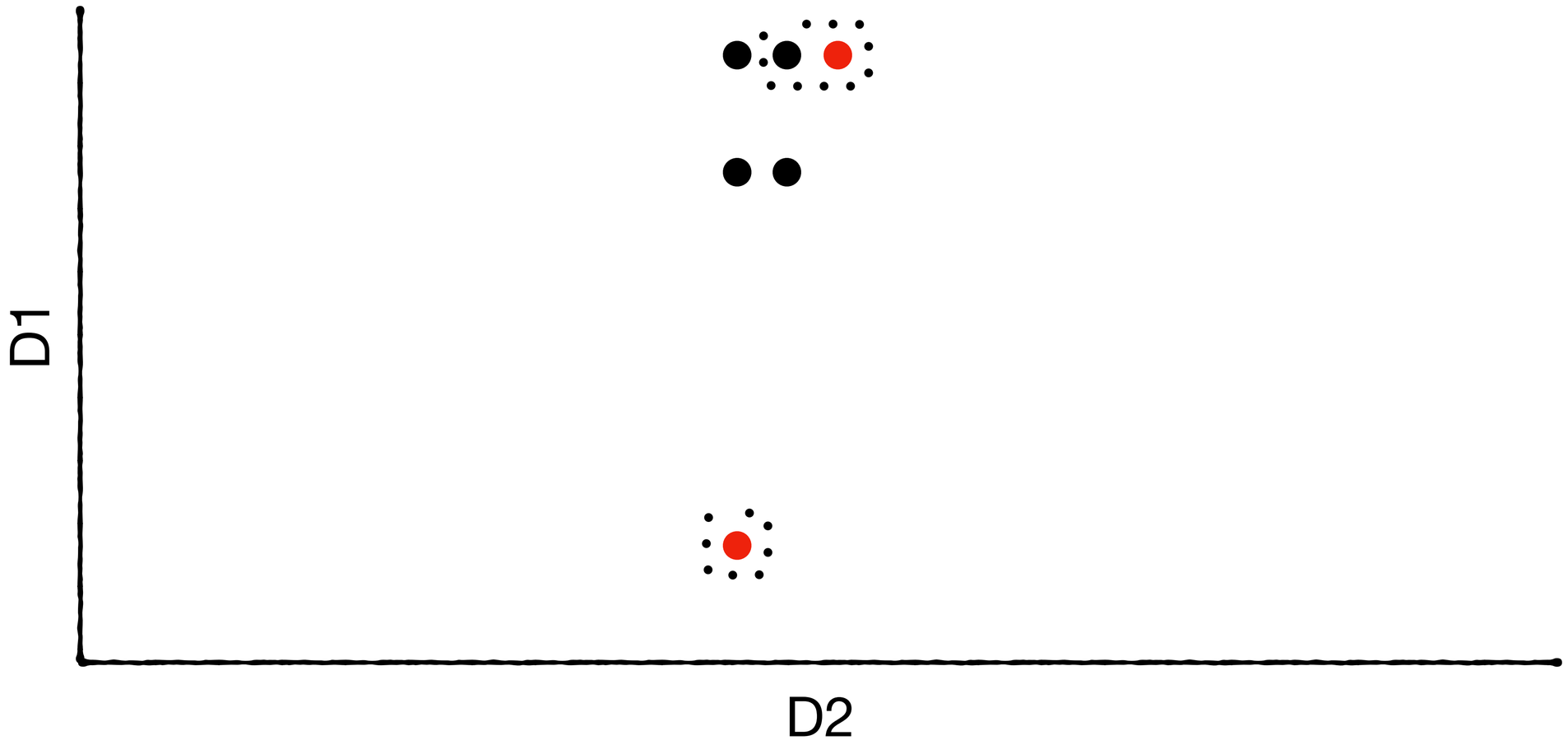
Space-amplification?

< B as opposed to < 2

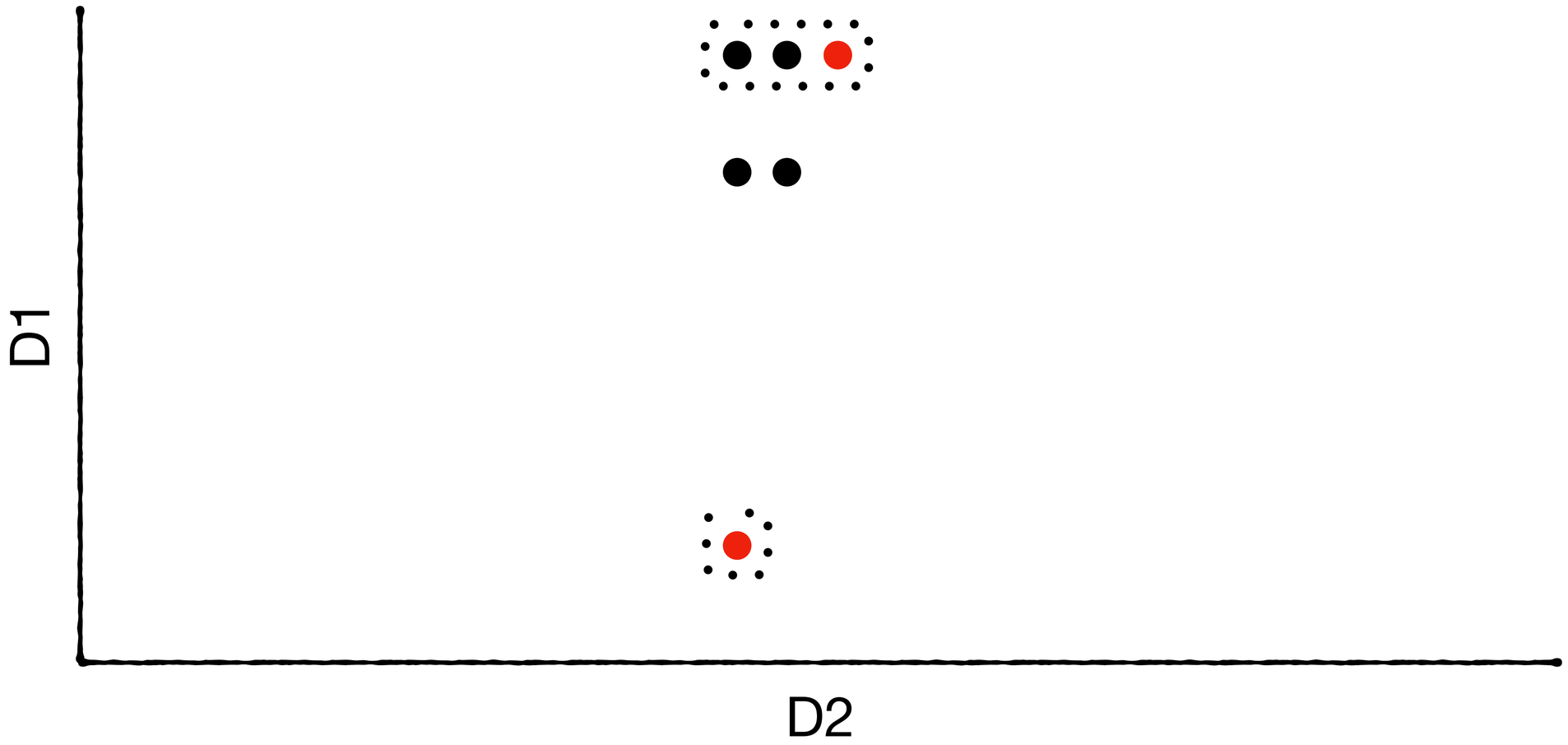
Instead, perform quadratic split subject to each node ultimately being half full



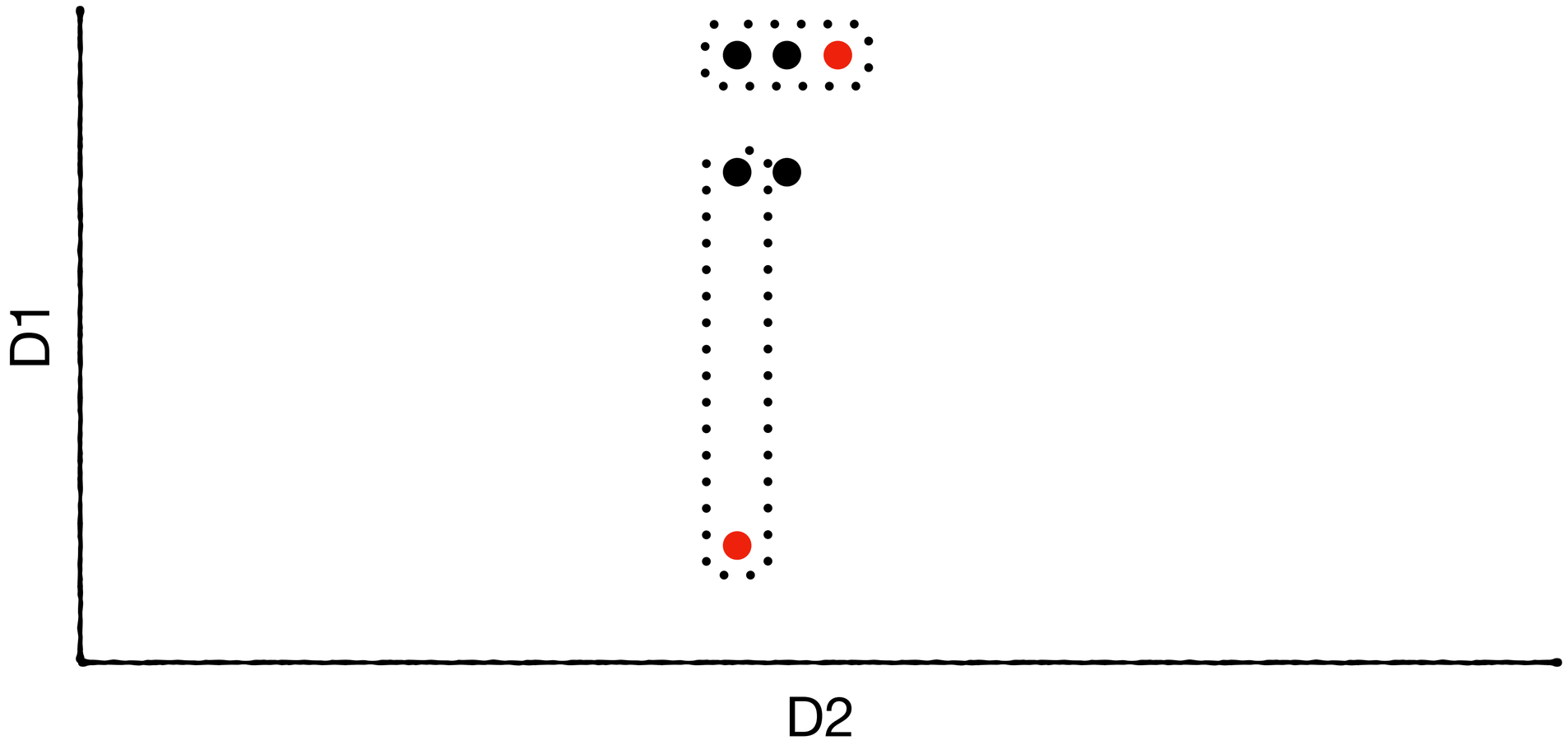
Instead, perform quadratic split subject to each node ultimately being half full



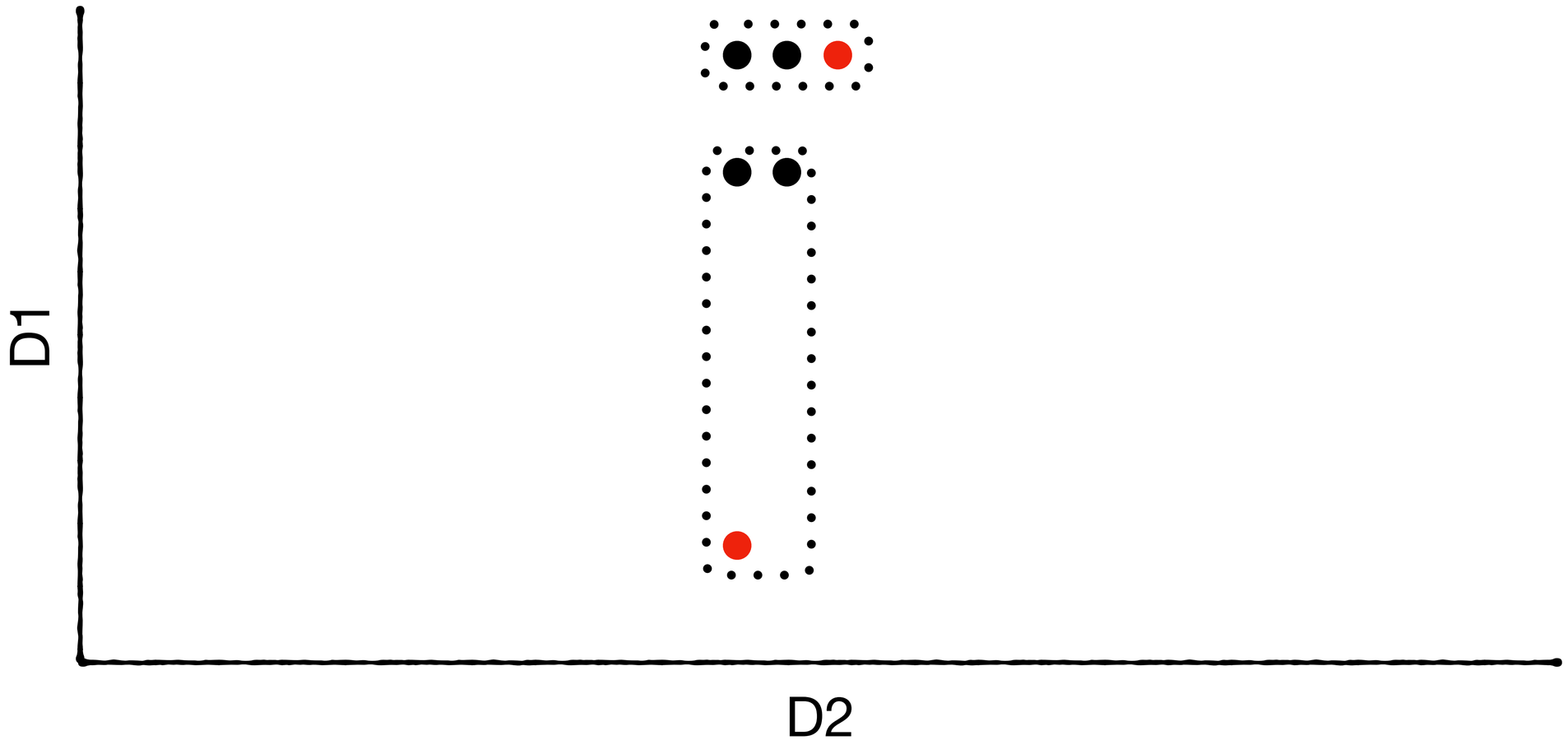
Instead, perform quadratic split subject to each node ultimately being half full



Instead, perform quadratic split subject to each node ultimately being half full



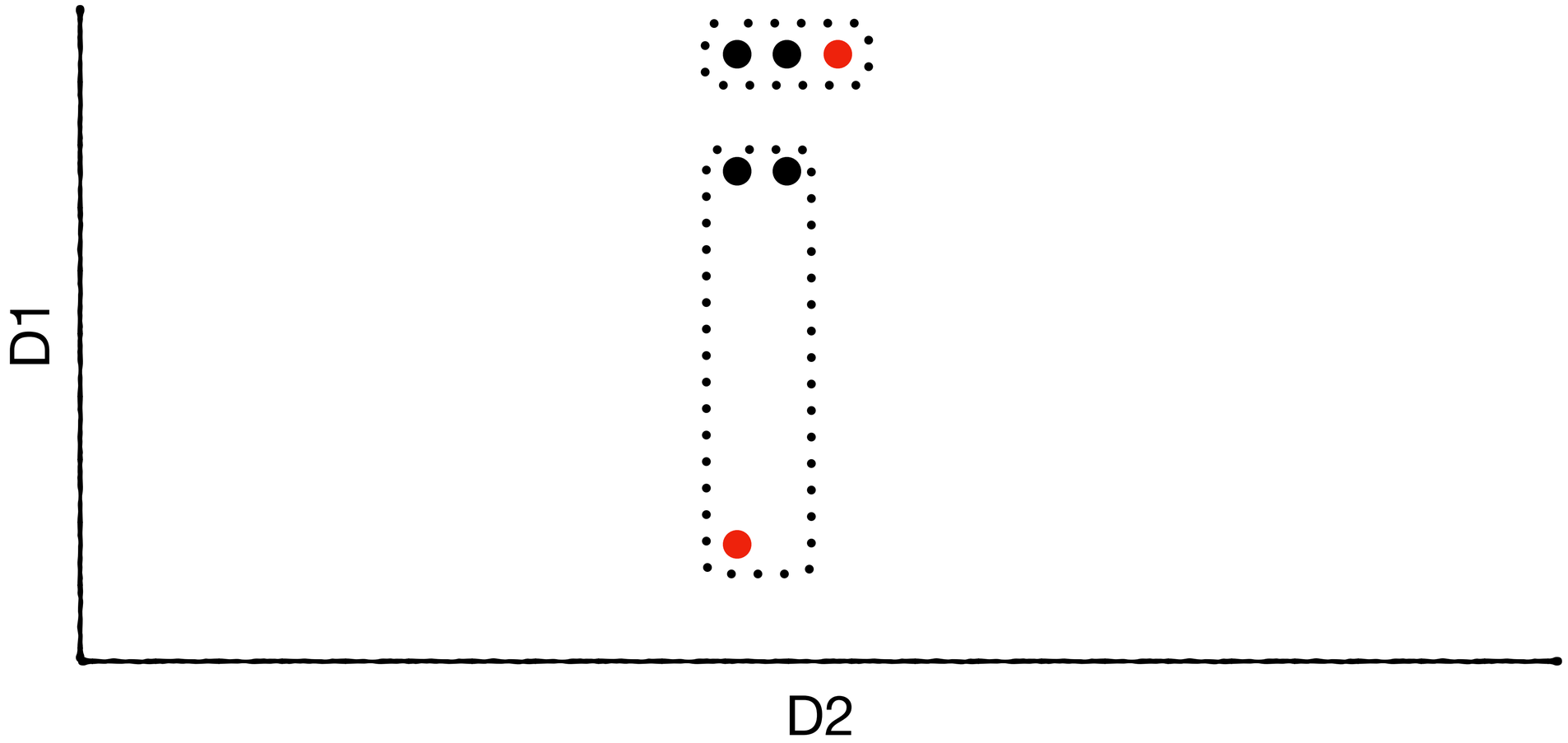
Instead, perform quadratic split subject to each node ultimately being half full



Goals: (1) minimize coverage

(2) minimize intersections

(3) maintain balance

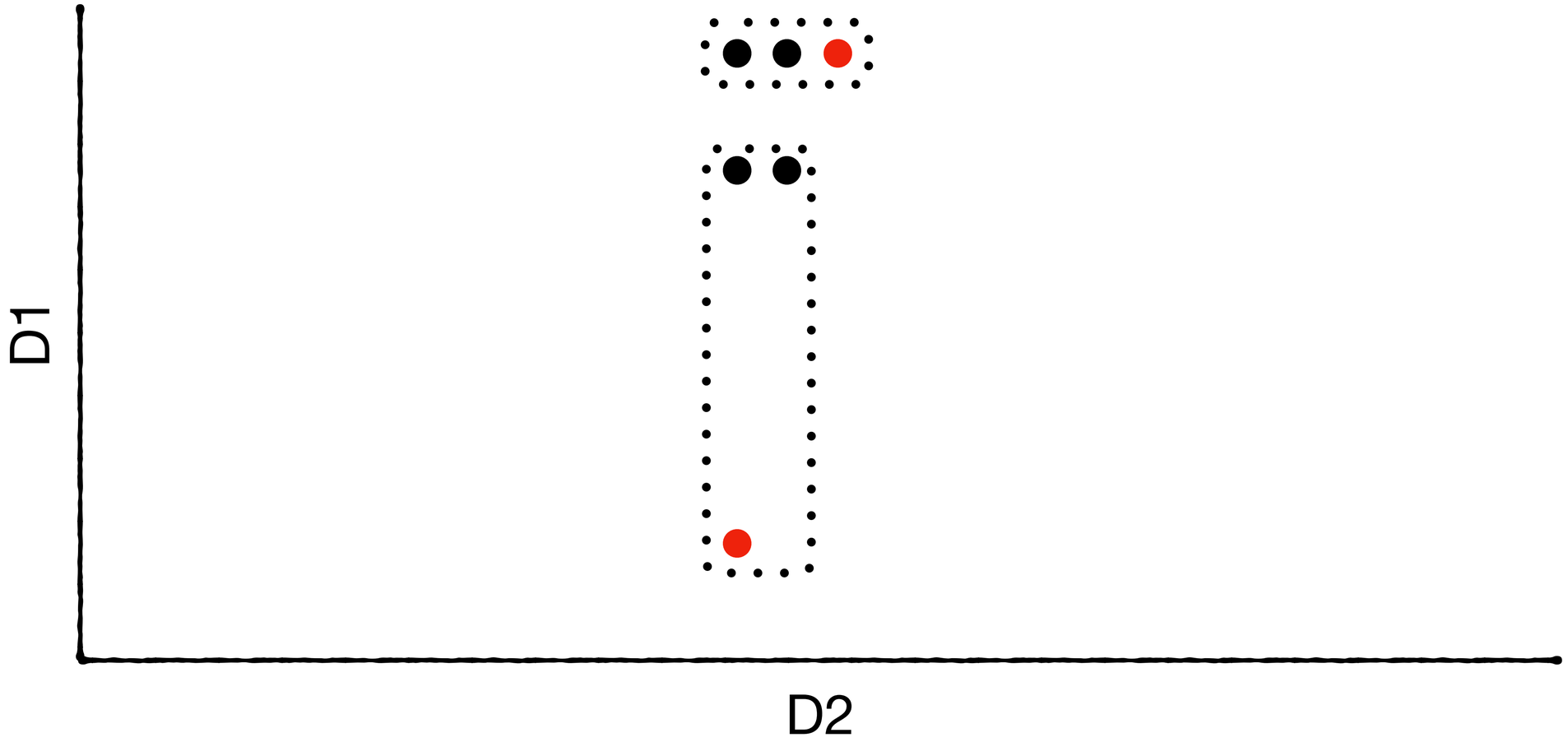


These sometimes conflict :)

Goals: (1) minimize coverage

(2) minimize intersections

(3) maintain balance



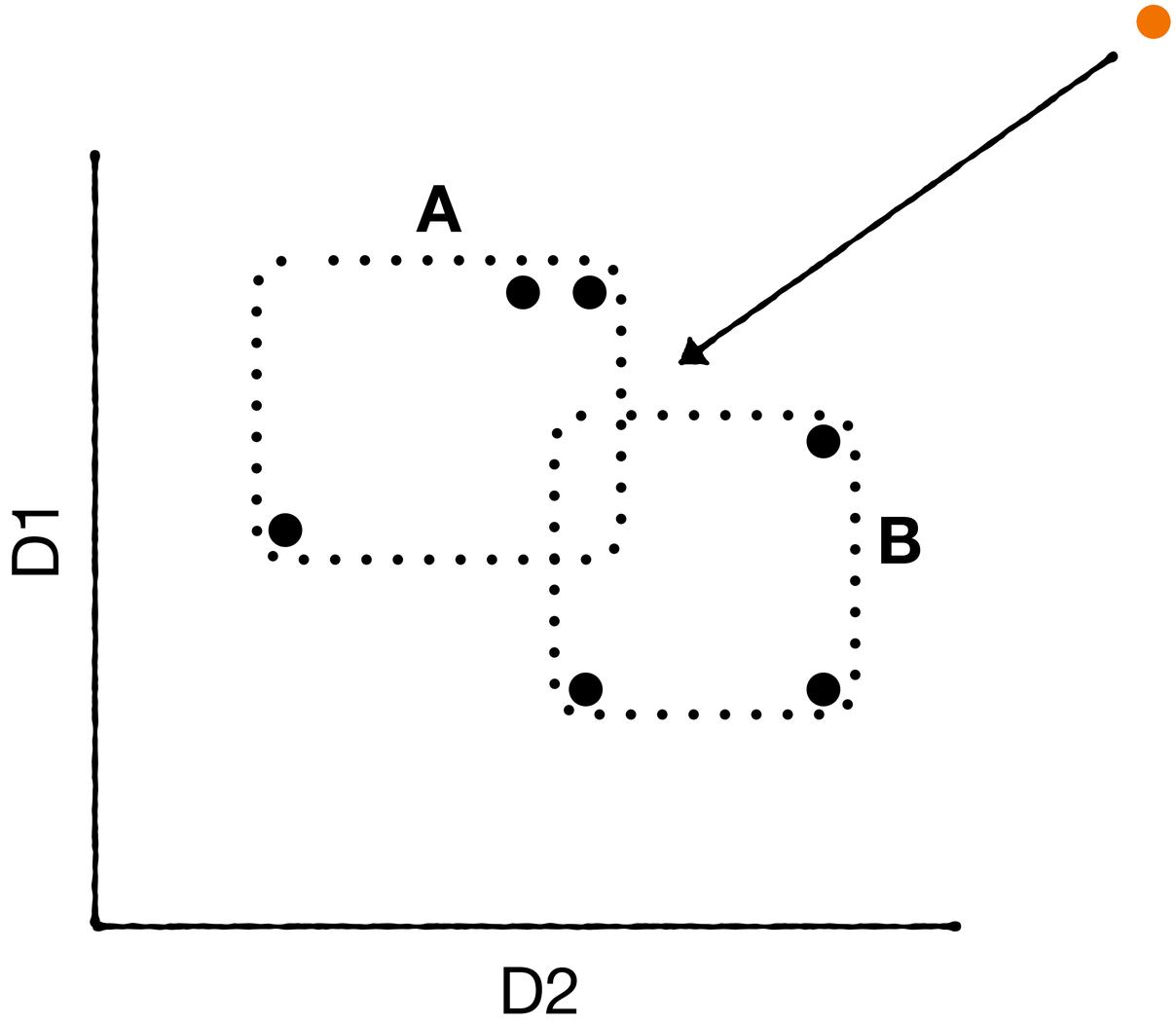
KD-Trees	R-Trees	R+Tree	R*-Tree	UB-Tree
1975	1984	1987	1990	2000

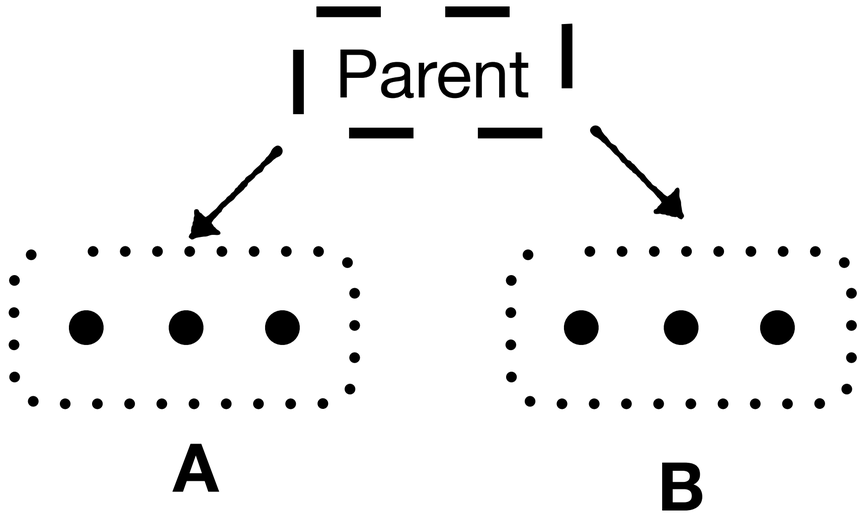
The R+Tree: A dynamic index for multi-dimensional objects

T Sellis, N Roussopoulos, C Faloutsos

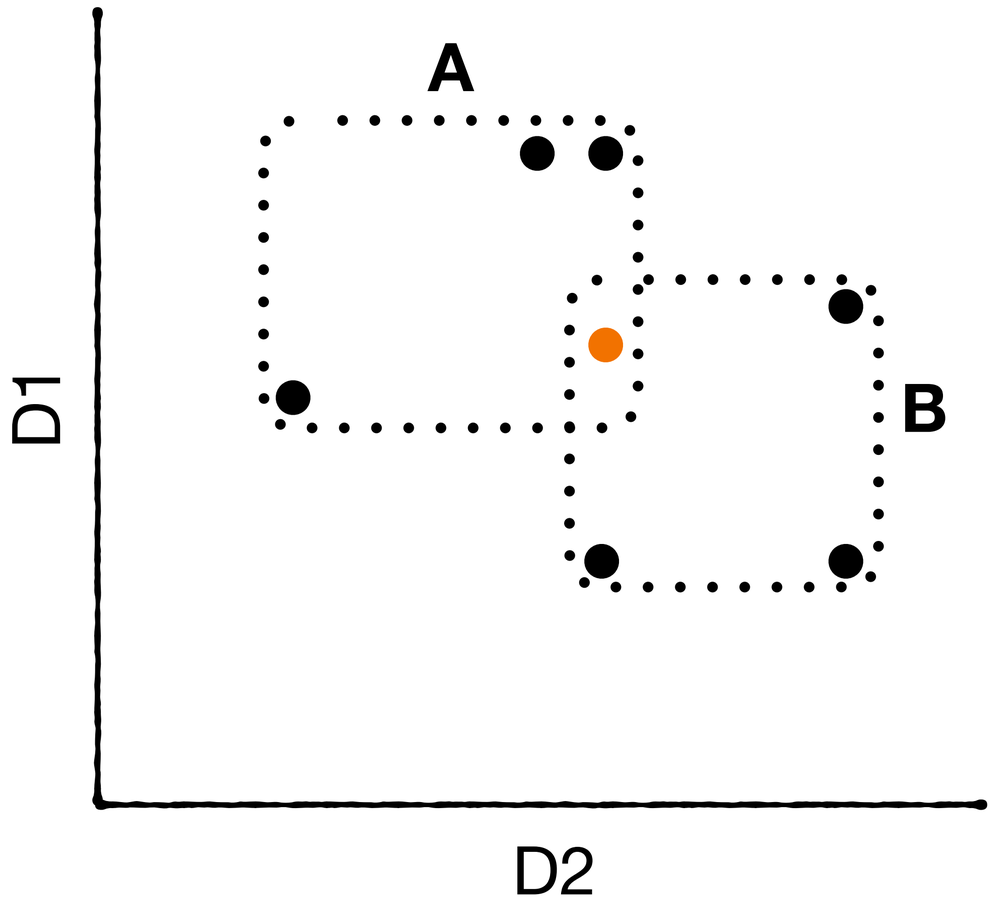
Published in **1987** (may be as technical report)

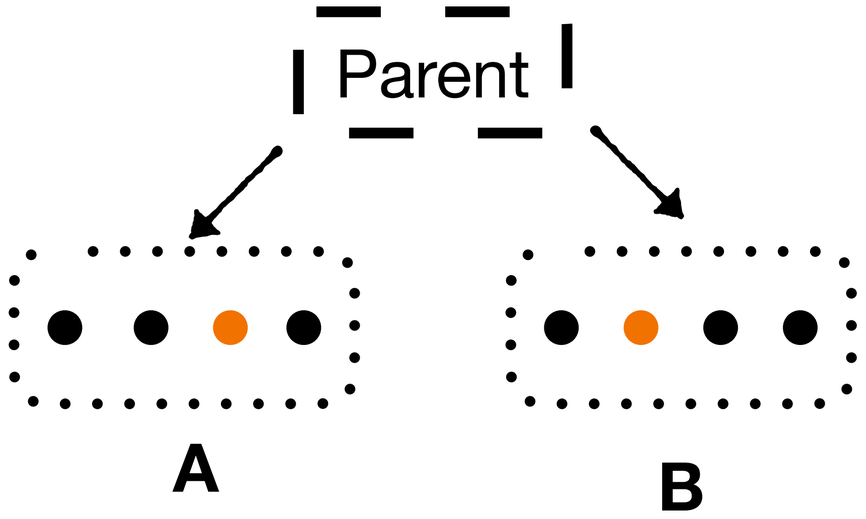
Insert entry to all intersecting nodes

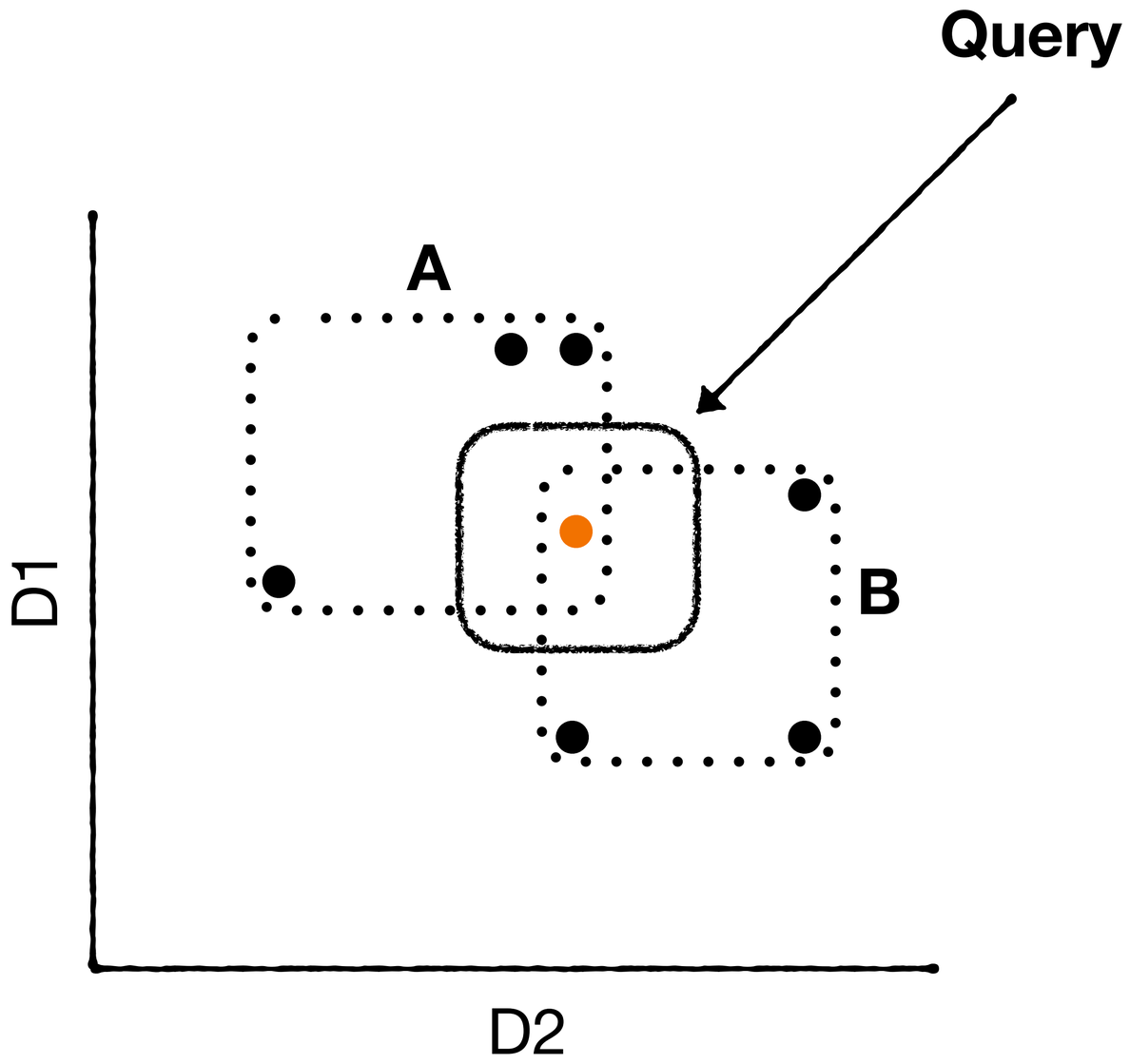


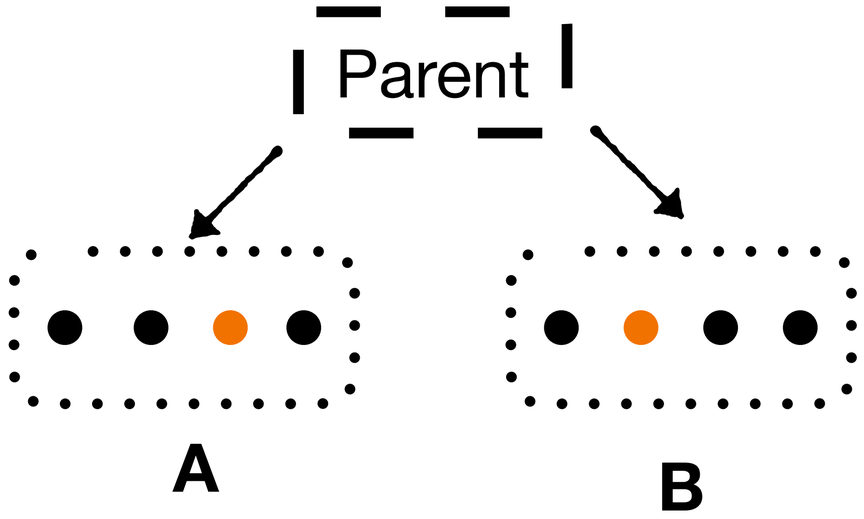


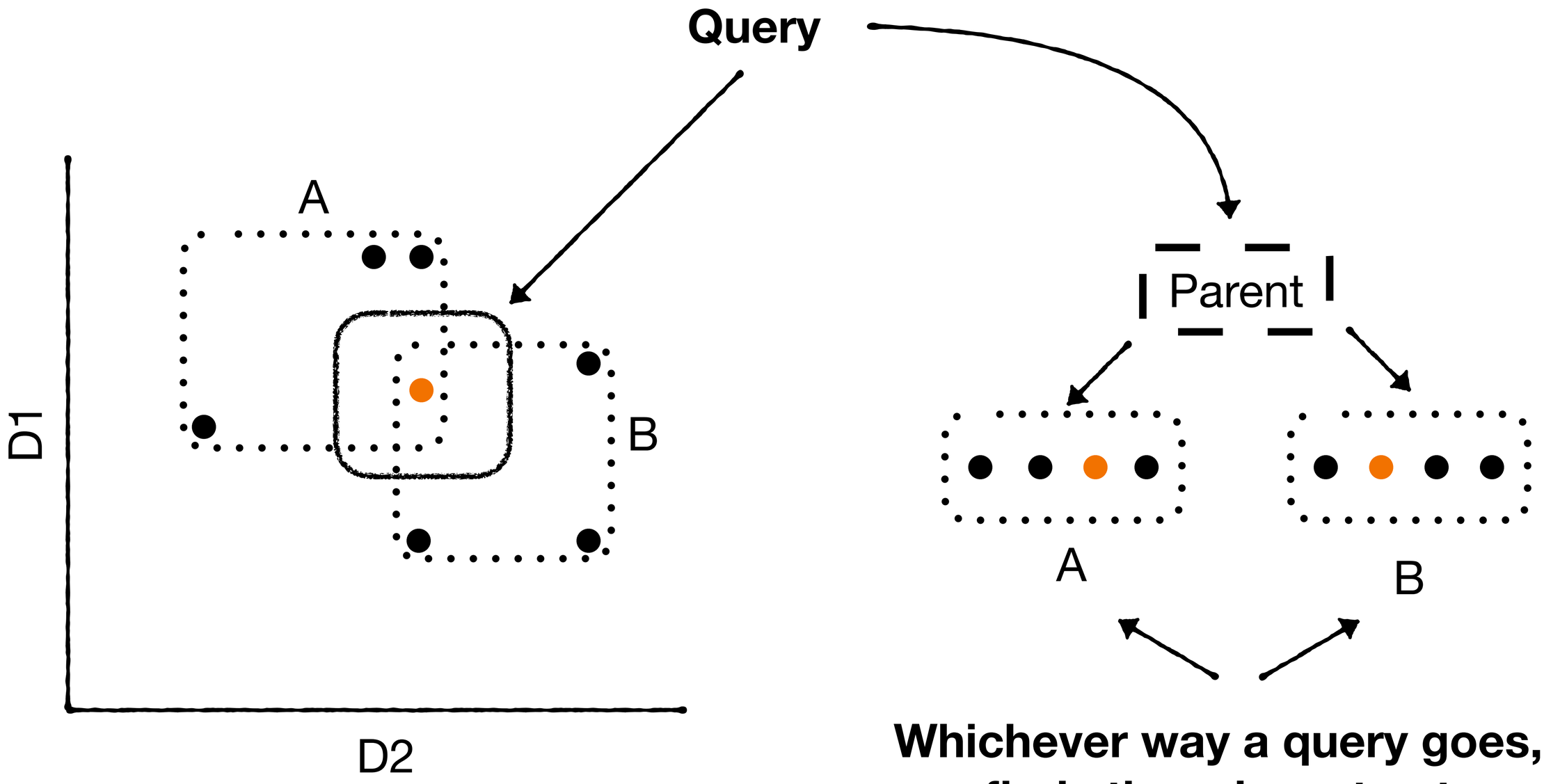
Insert entry to all intersecting nodes





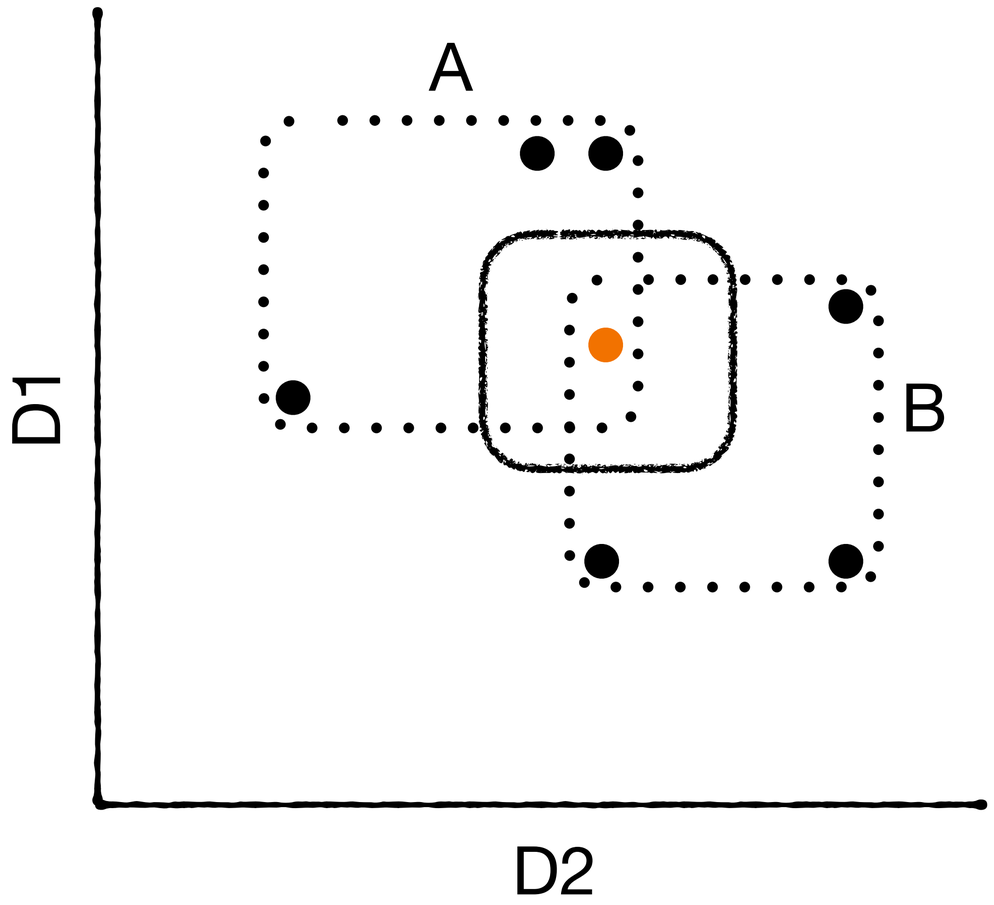

Query
A
B
D1
D2

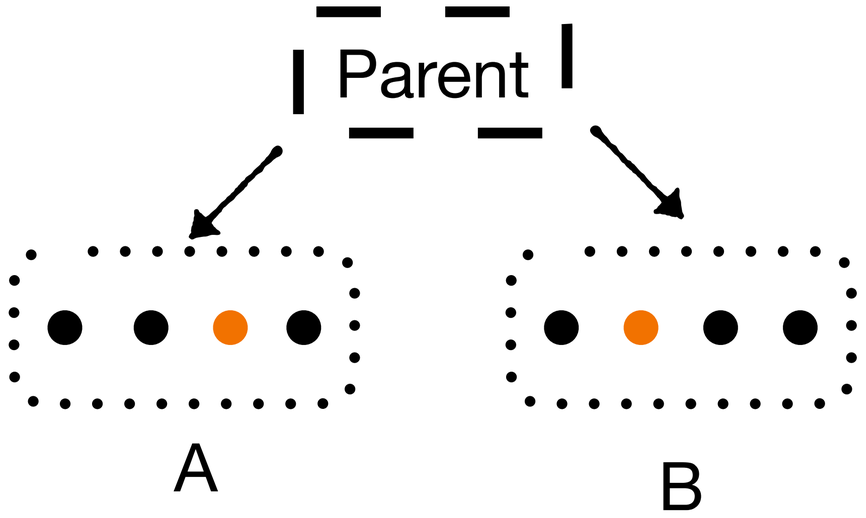

Parent
A
B



Whichever way a query goes, it finds the relevant entry

But duplication costs space





KD-Trees	R-Trees	R+Tree	R*-Tree	UB-Tree
1975	1984	1987	1990	2000

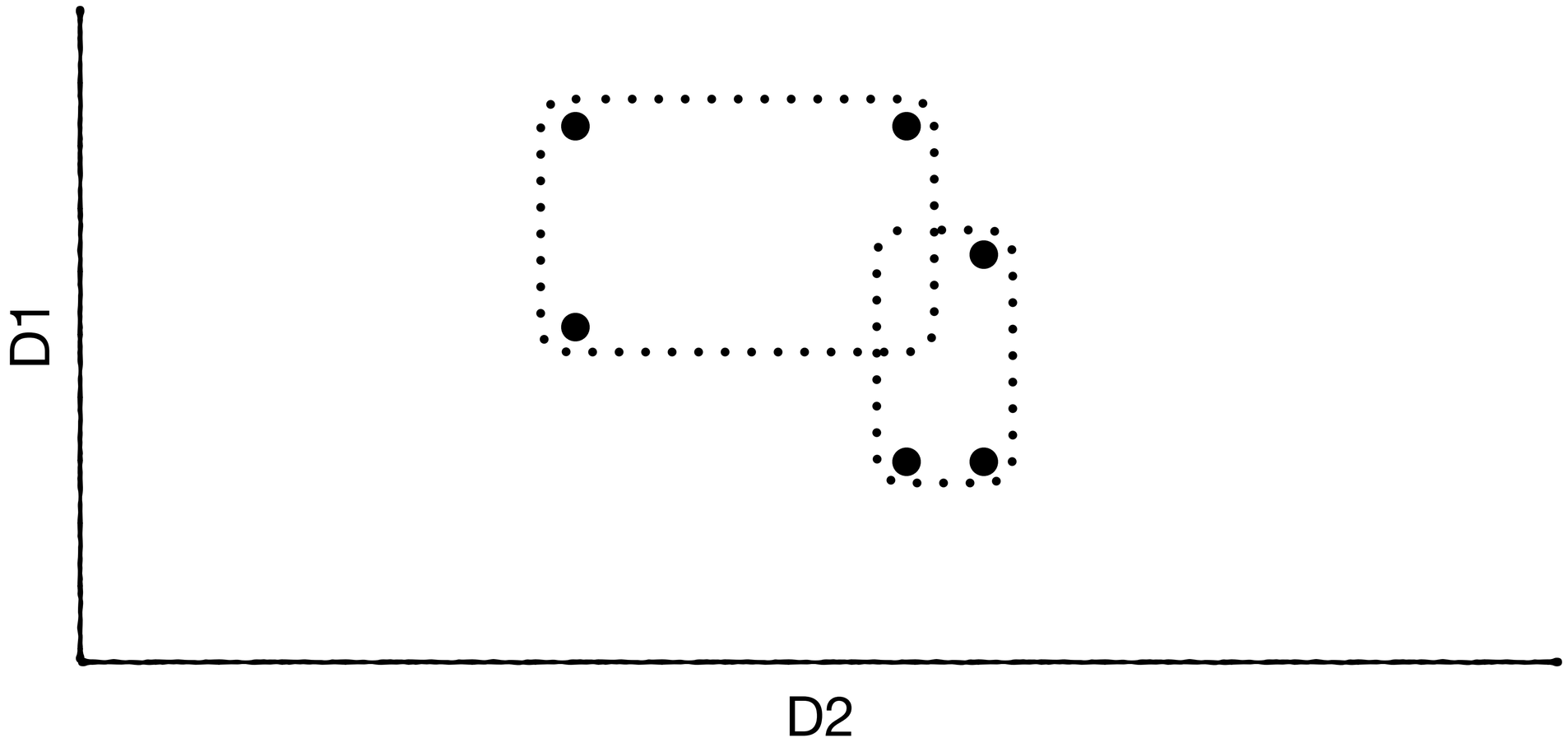
The R*-Tree: an efficient and robust access method for points and rectangles

Beckmann, N.; Kriegel, H. P.; Schneider, R.; Seeger

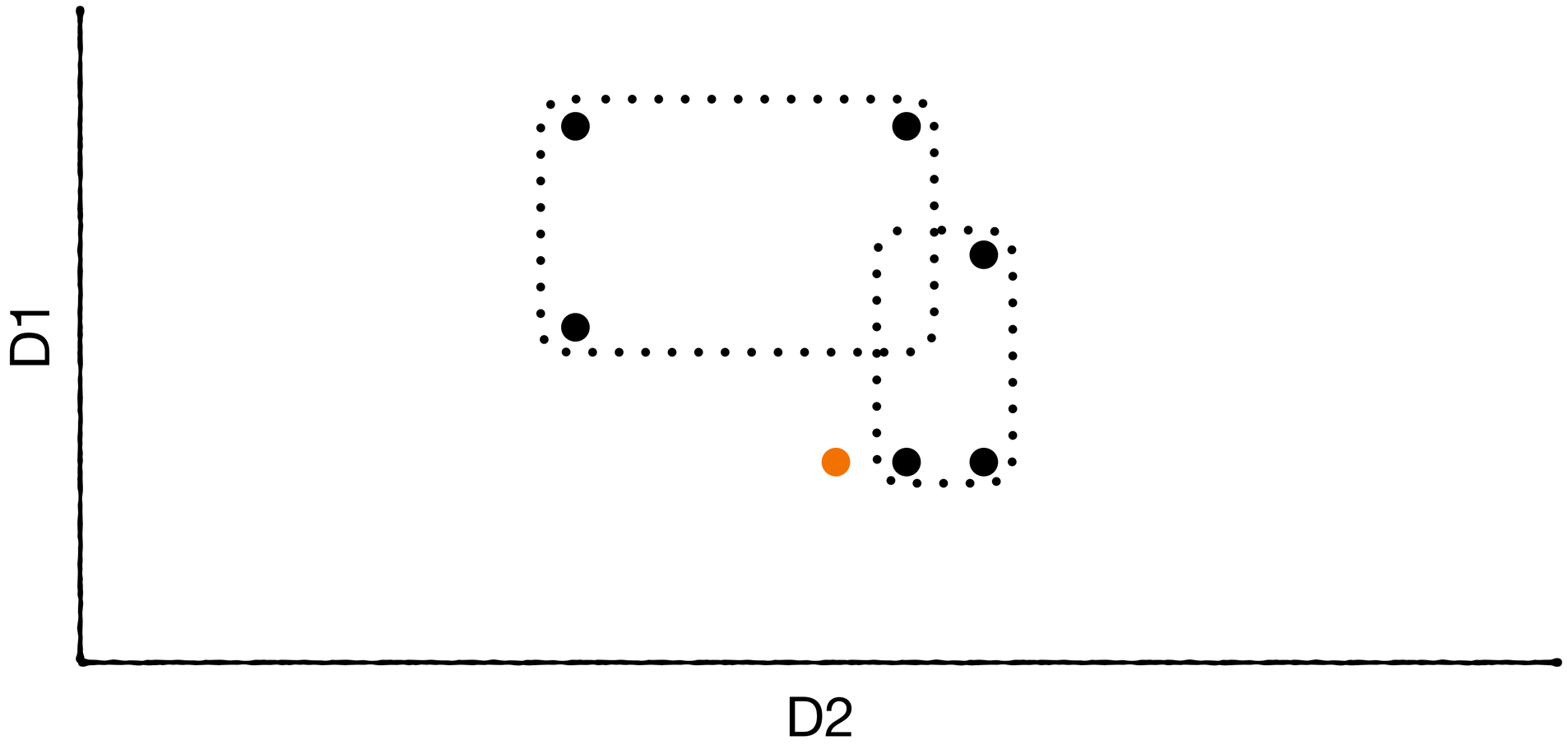
SIGMOD 1990

Insertion order affects overlap among rectangles

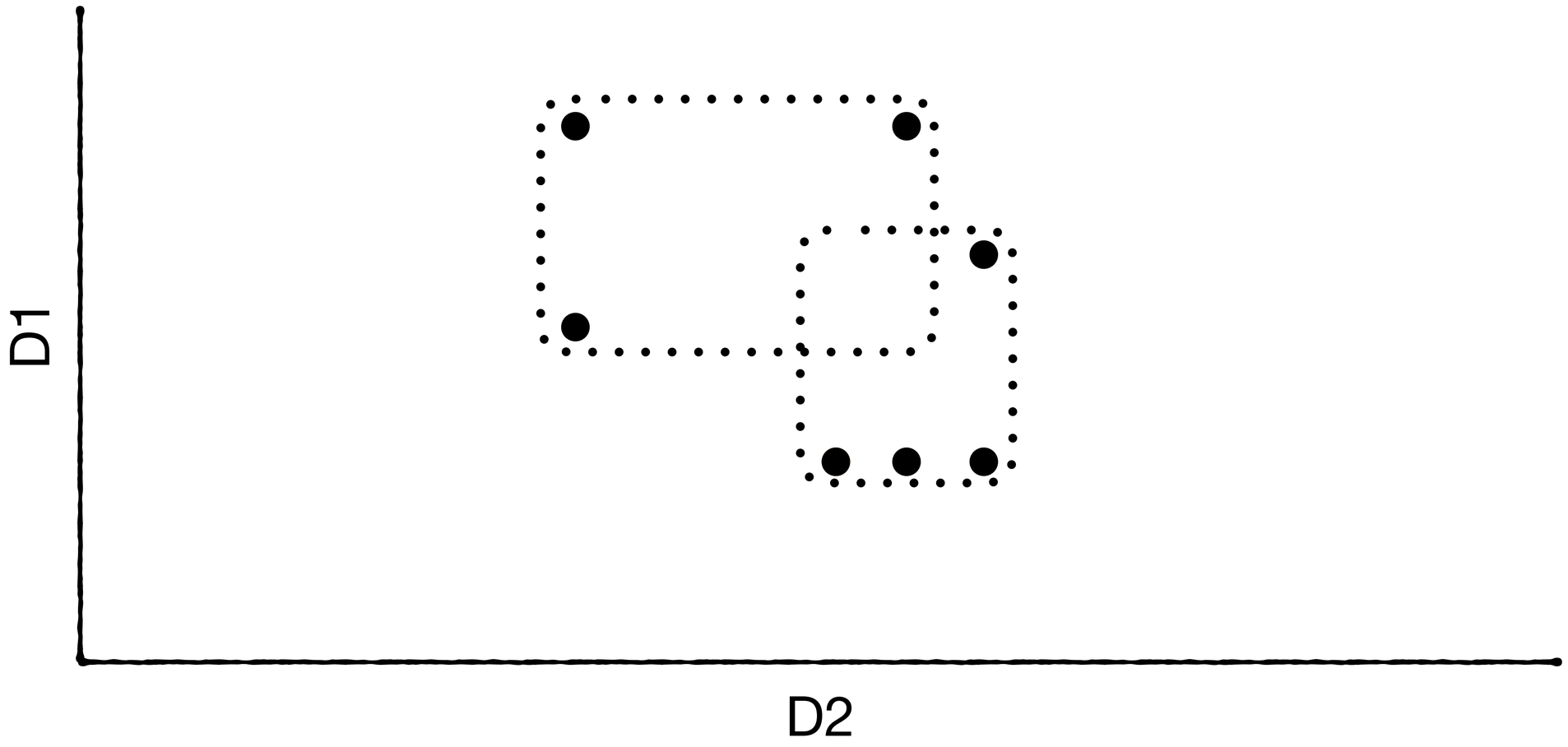
Insertion order affects overlap among rectangles



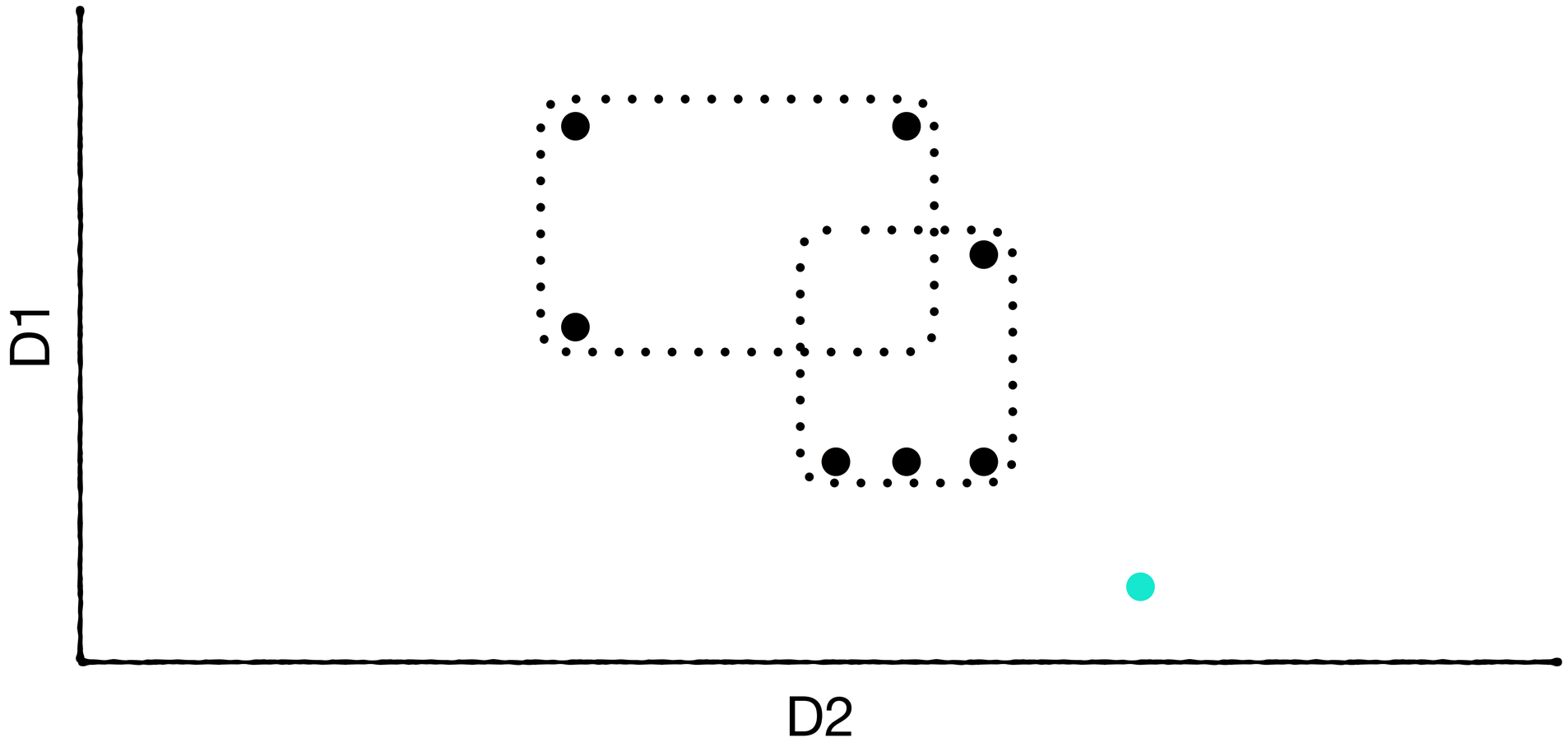
Insertion order affects overlap among rectangles



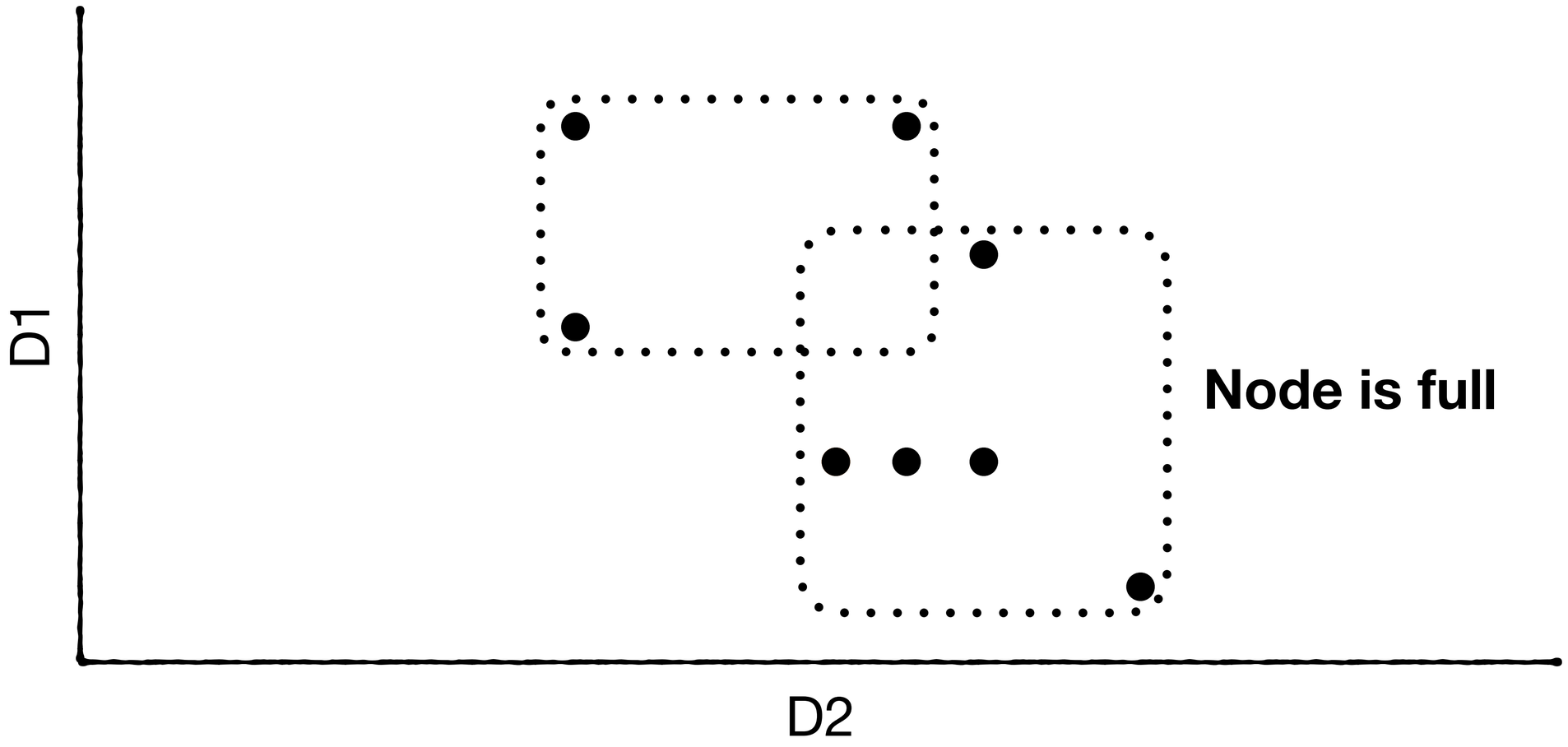
Insertion order affects overlap among rectangles



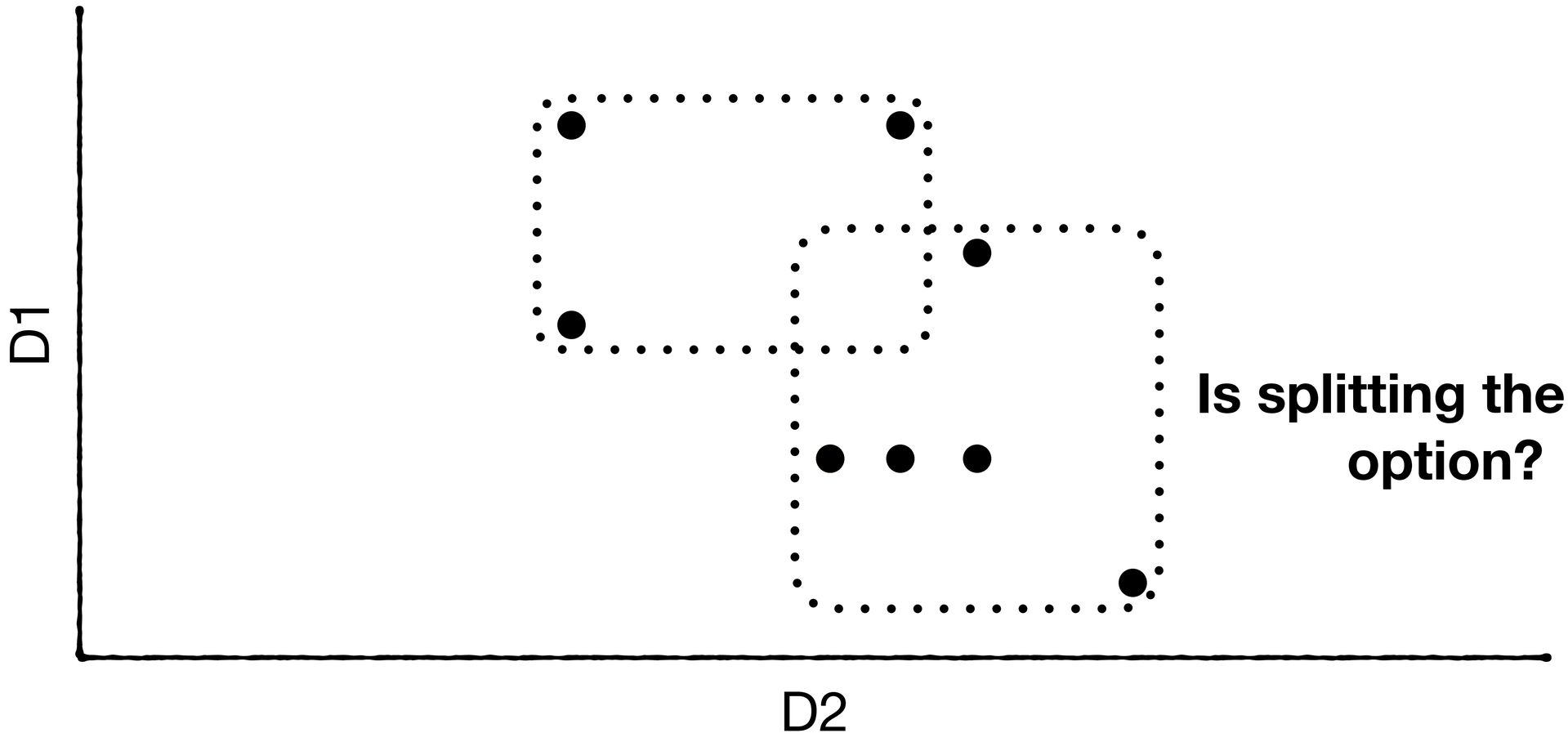
Insertion order affects overlap among rectangles



Insertion order affects overlap among rectangles



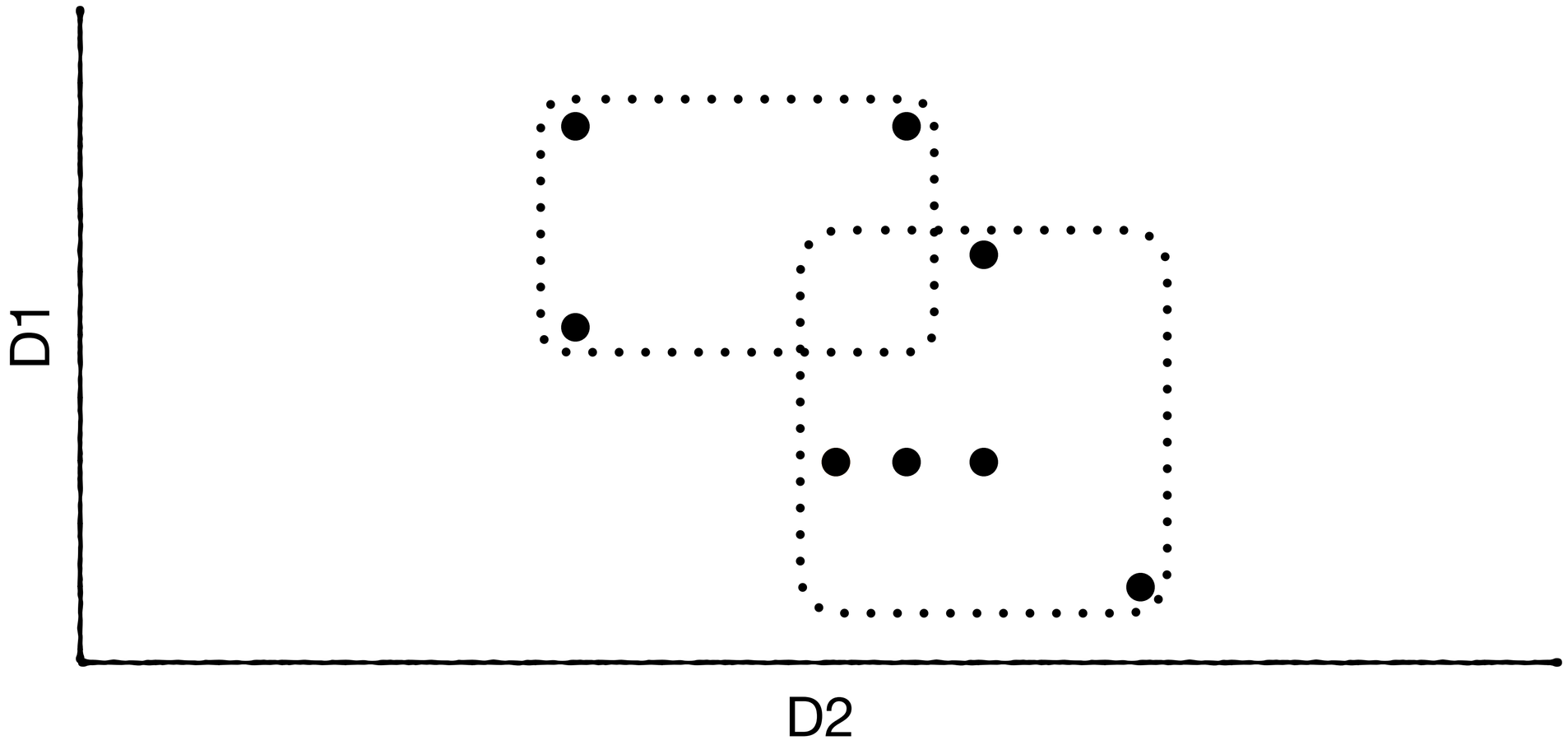
Insertion order affects overlap among rectangles



Is splitting the best option?

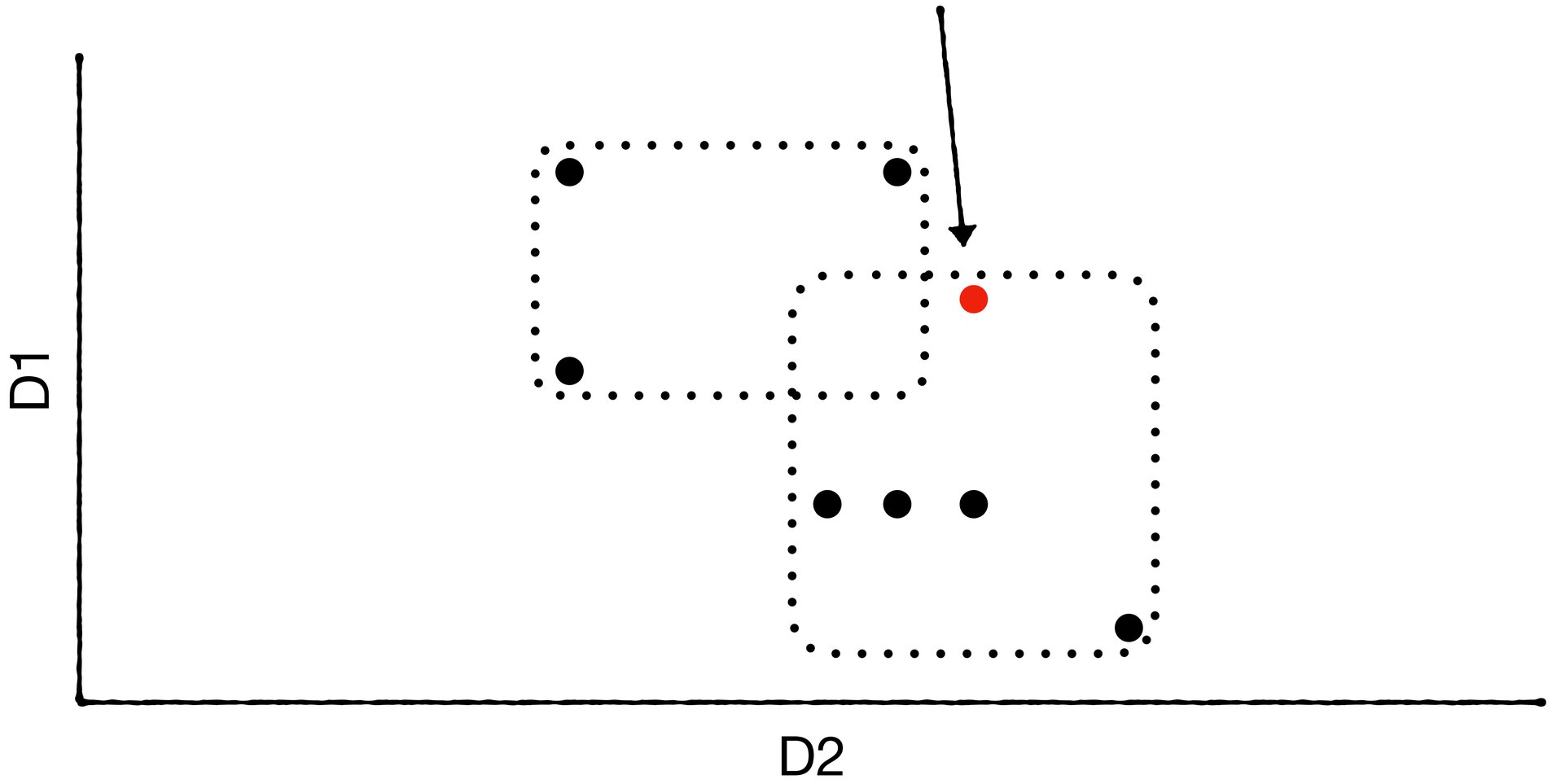
Suppose instead we reinsert some anomalous dot in filled up node

(Which?)

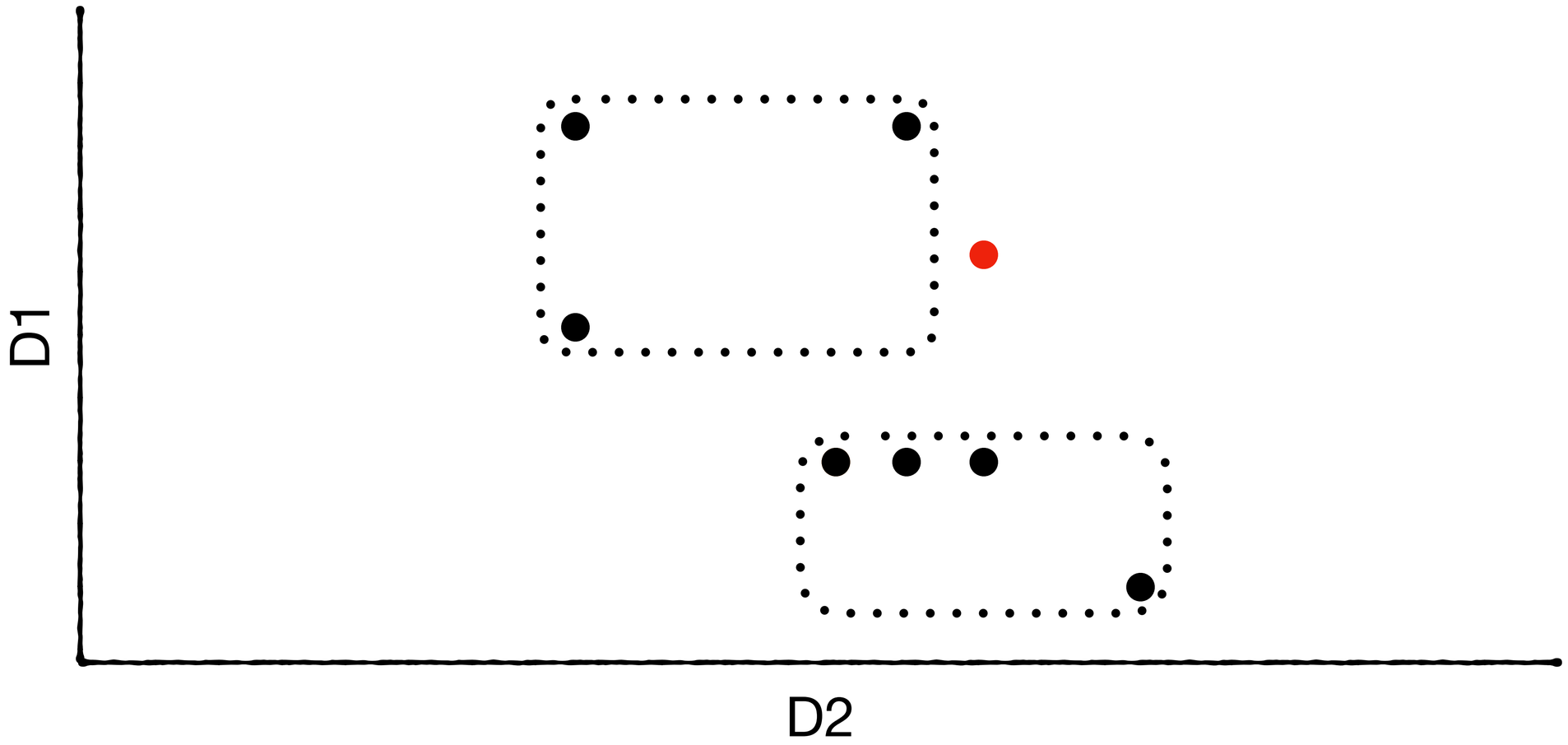


Suppose instead we reinsert some anomalous dot in filled up node

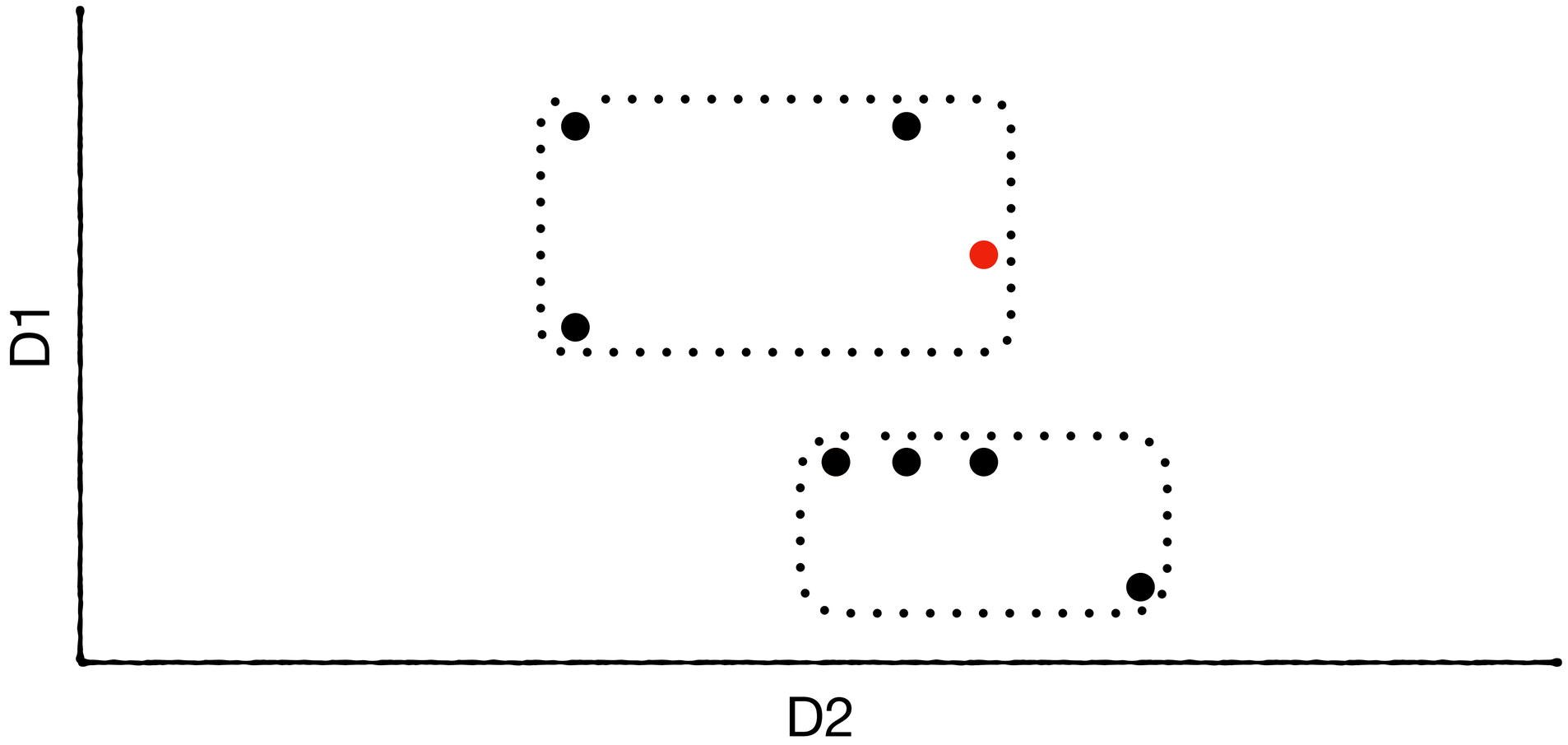
(Which?)



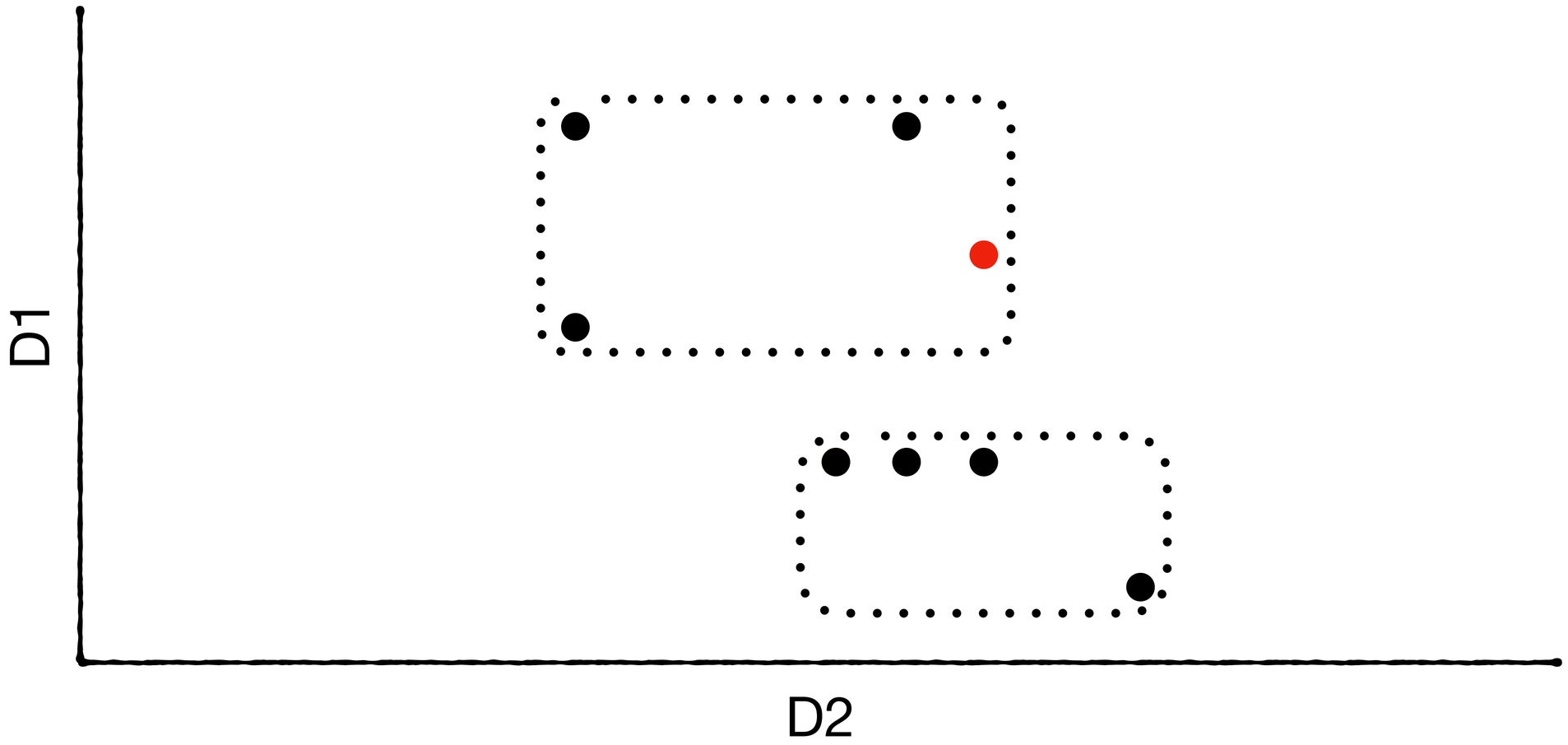
Suppose instead we reinsert some anomalous dot in filled up node



reinsertion places entry in other rectangle, eliminating overlap

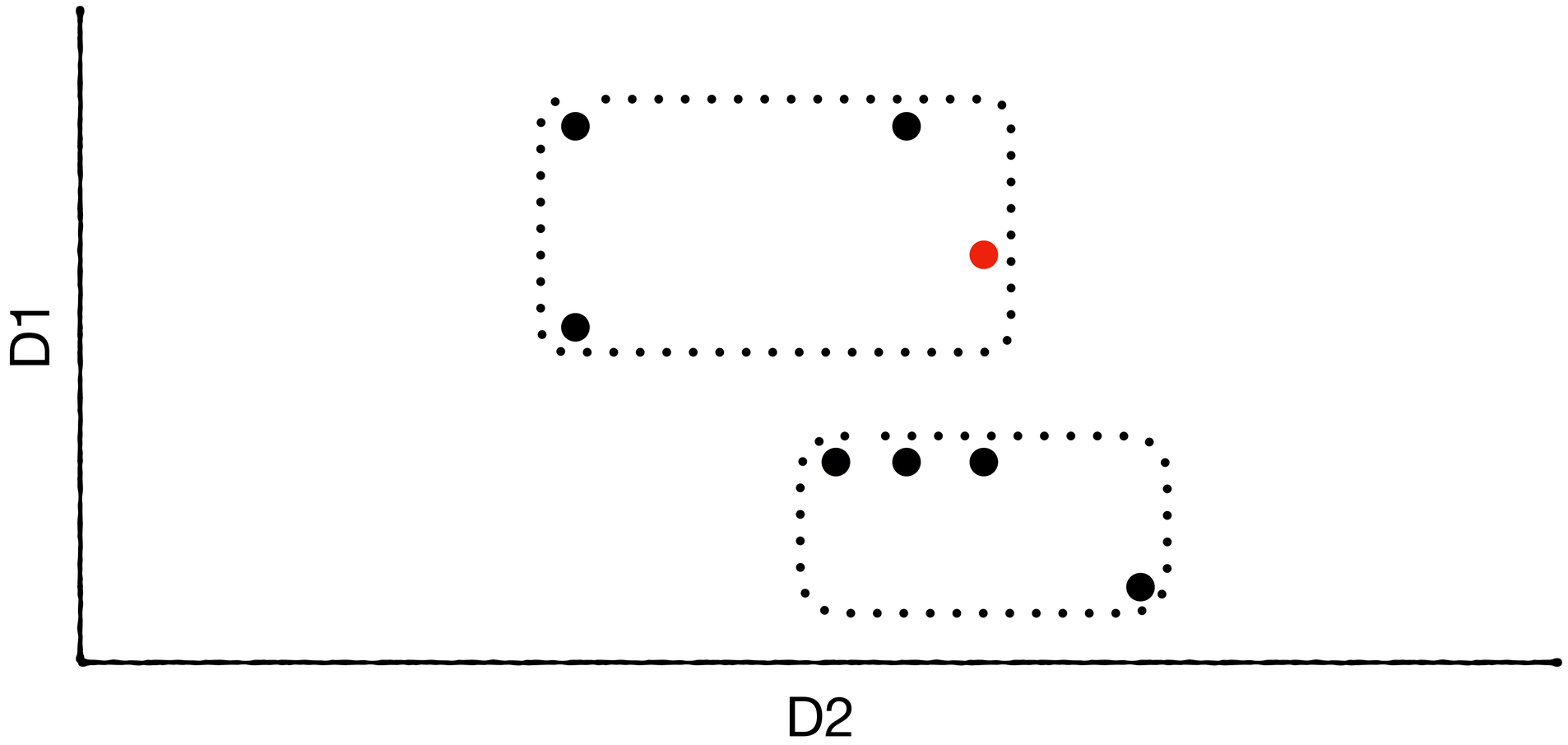


R*-Tree reinserts entries from full nodes to find better rectangle for them



R*-Tree reinserts entries from full nodes to find better rectangle for them

Detailed algorithm out of scope

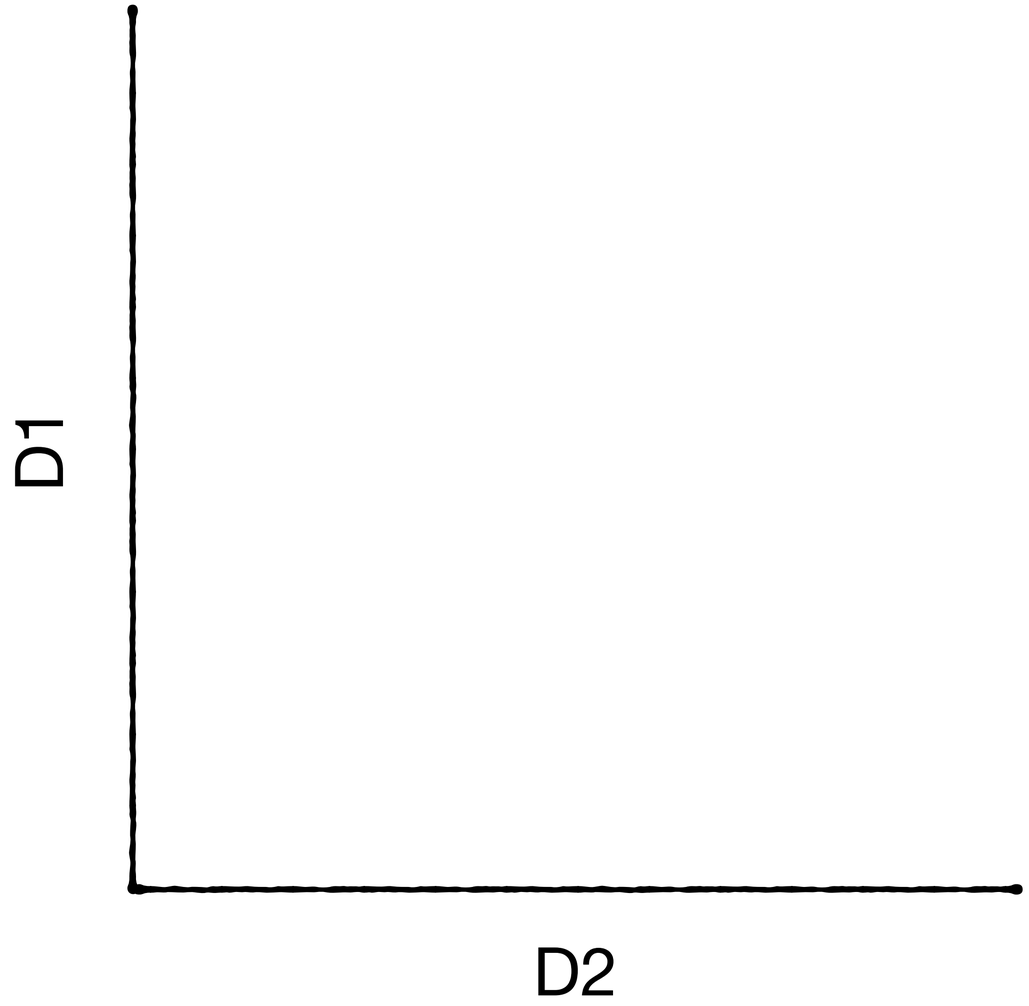


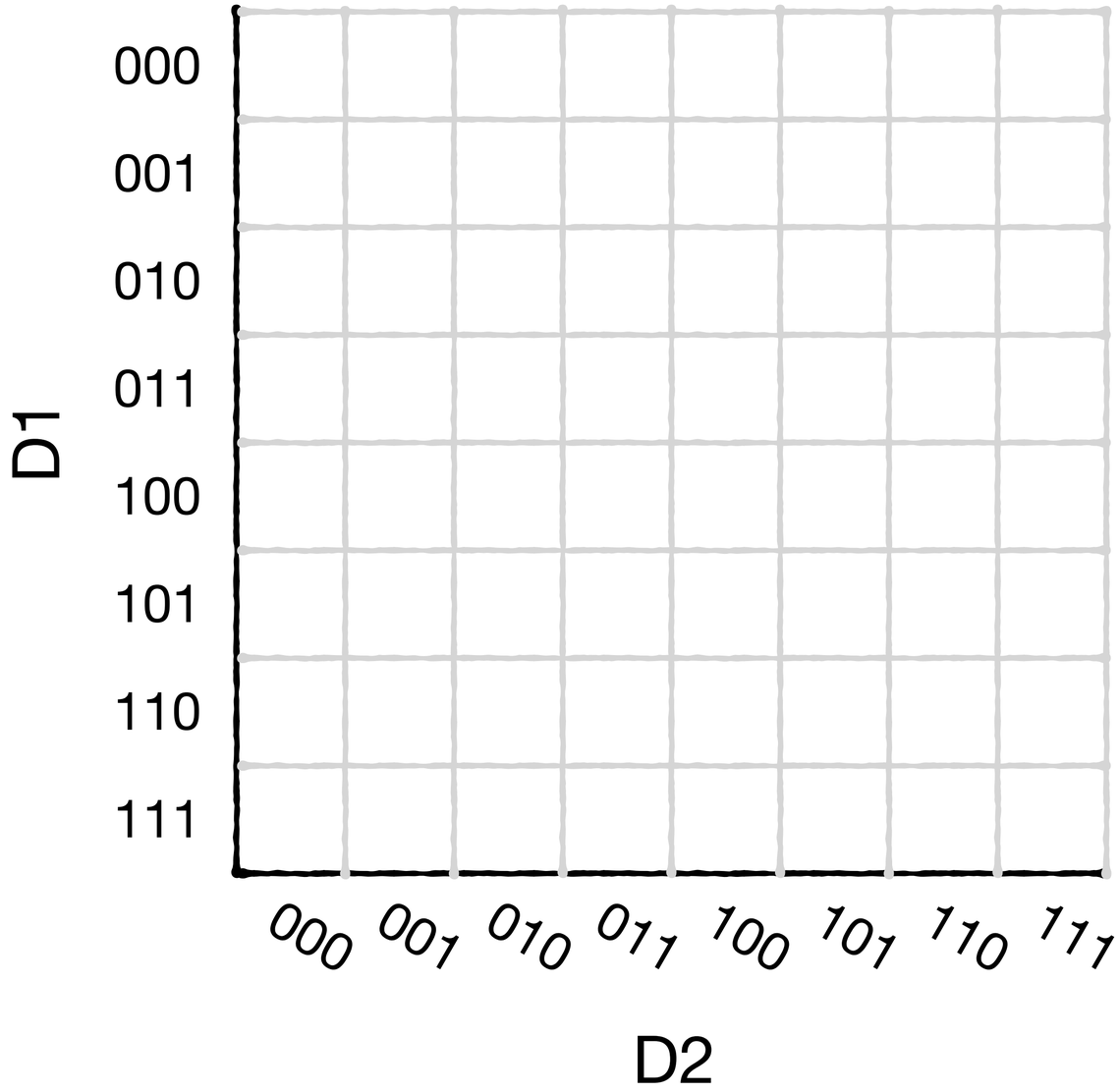
KD-Trees	R-Trees	R+Tree	R*-Tree	UB-Tree
1975	1984	1987	1990	2000

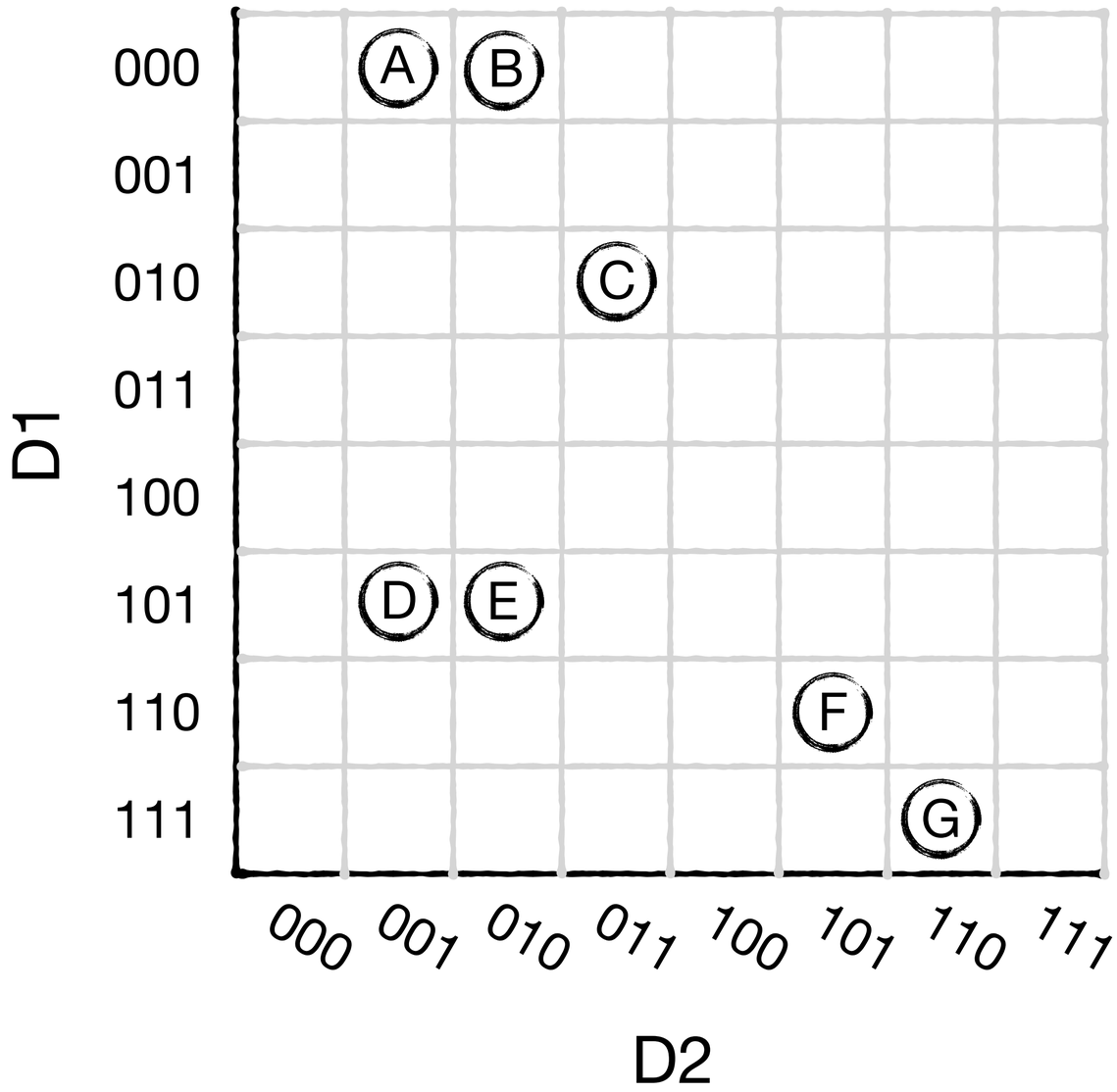
Integrating the UB-Tree into a Database System Kernel

Frank Ramsak, Volker Markl, Robert Fenk, Martin Zirkel, Klaus Elhardt, Rudolf Bayer

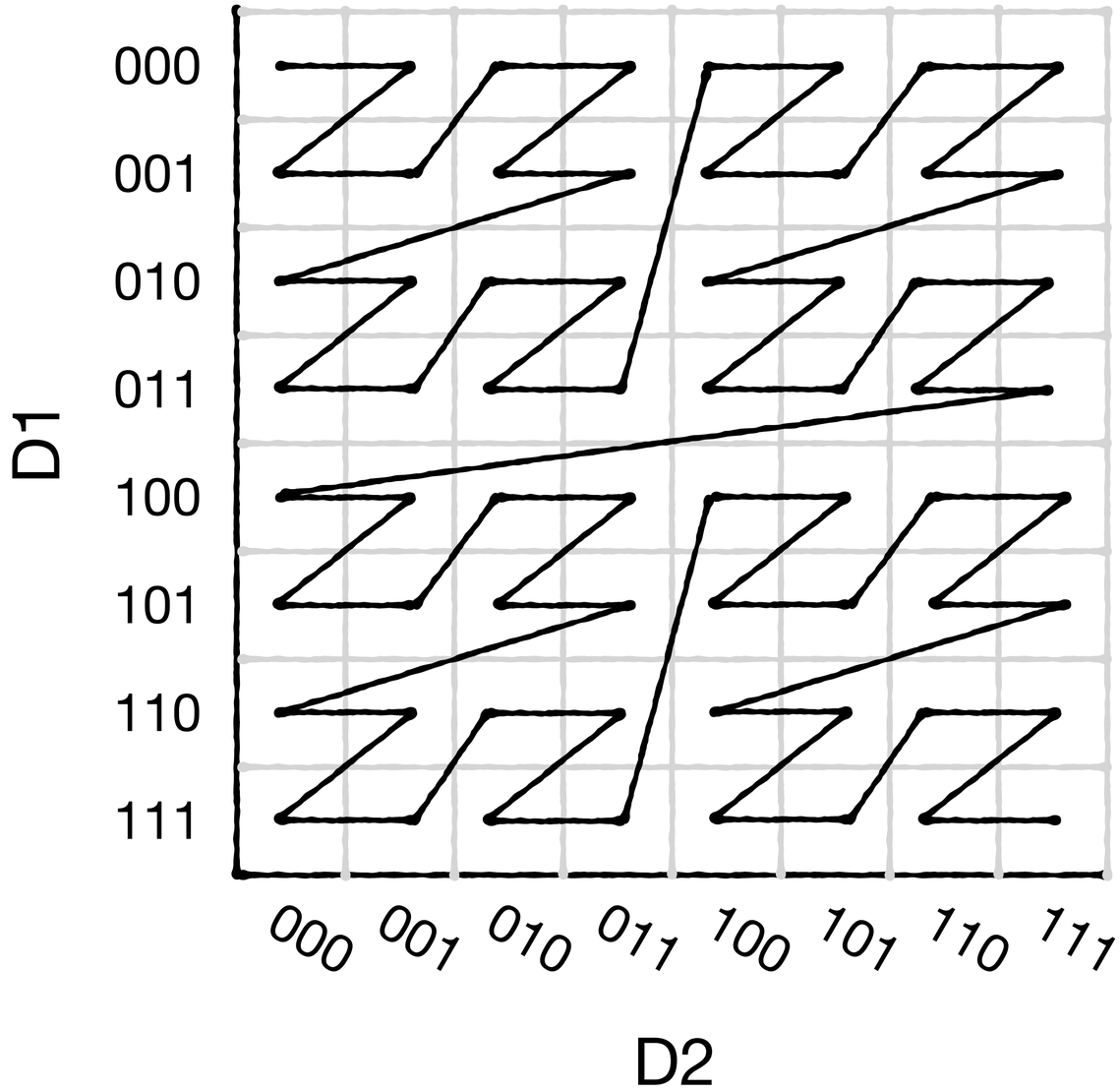
VLDB 2000


D1
D2

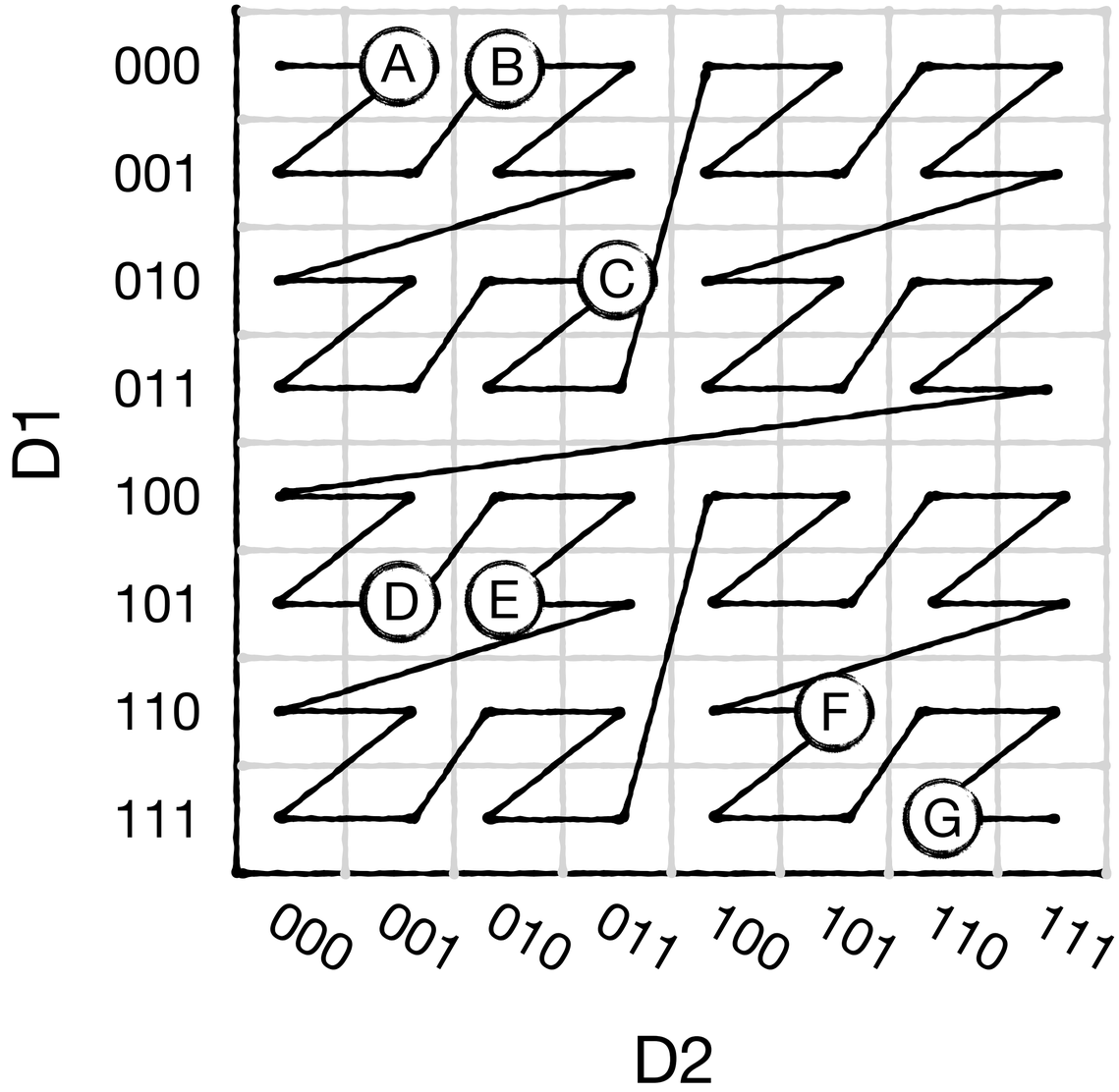

000
001
010
011
100
101
110
111
D1
000
001
010
011
100
101
110
111
D2

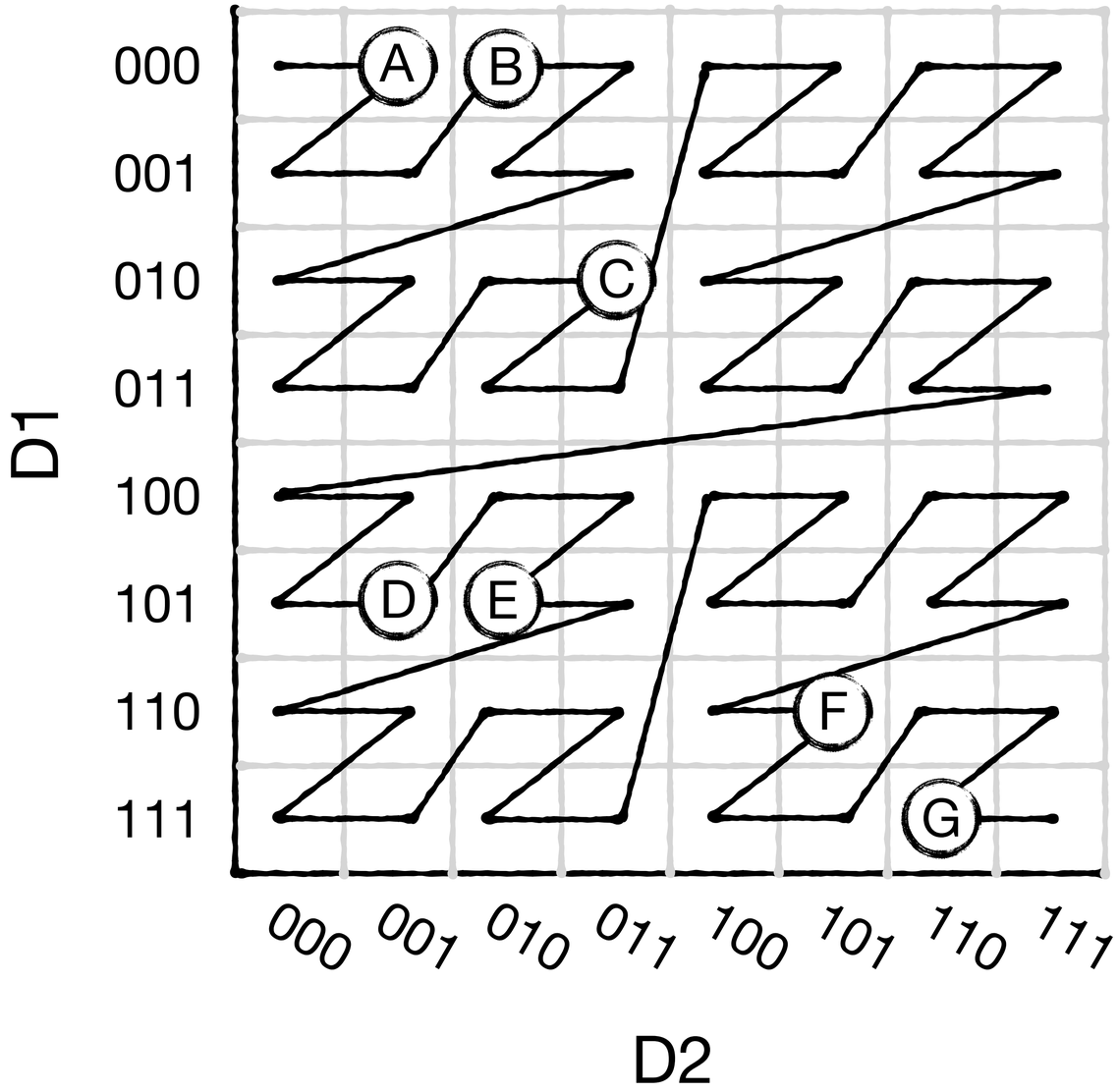

D1
000
001
010
011
100
101
110
111
A
B
C
D
E
F
G
000
001
010
011
100
101
110
111
D2

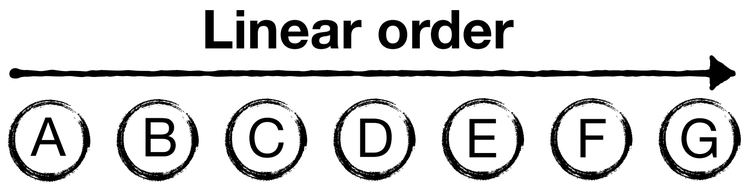
Z-Order Curves

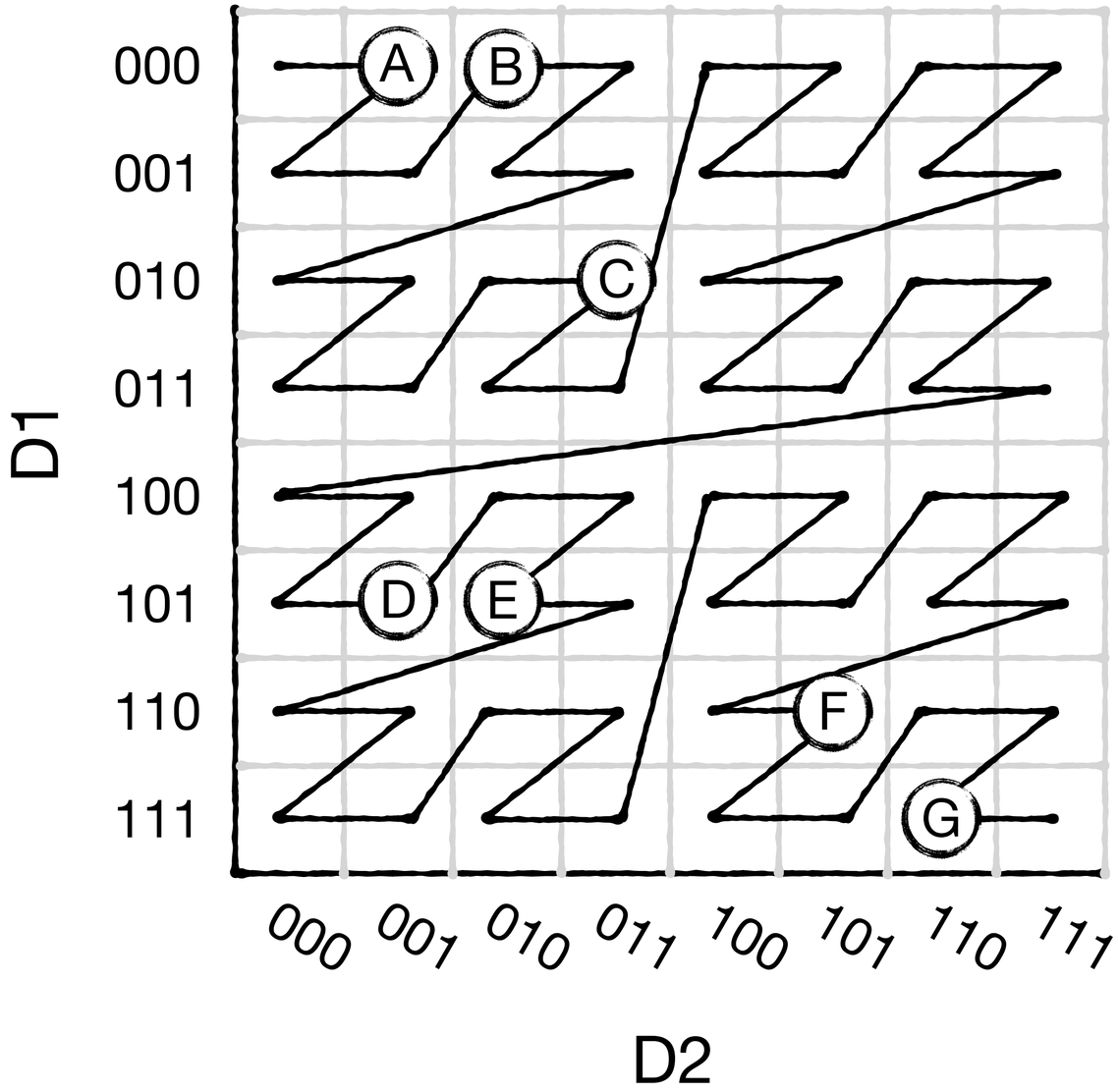


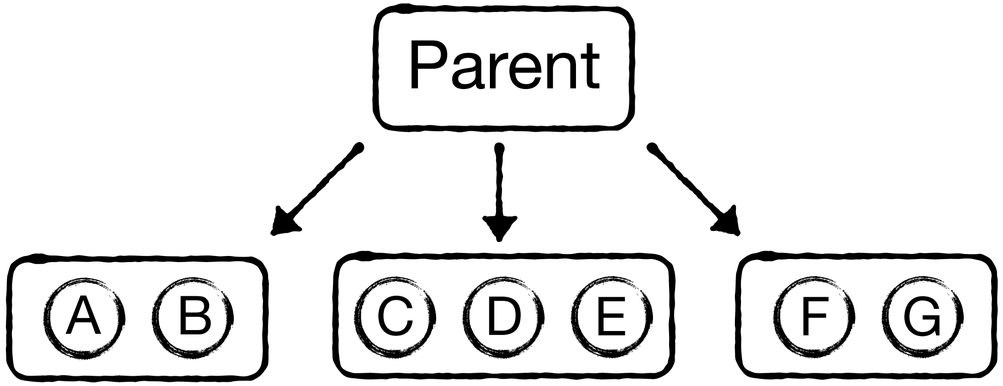
Z-Order Curves




000
001
010
011
100
101
110
111
A
B
C
D
E
F
G
D1
000 001 010 011 100 101 110 111
D2

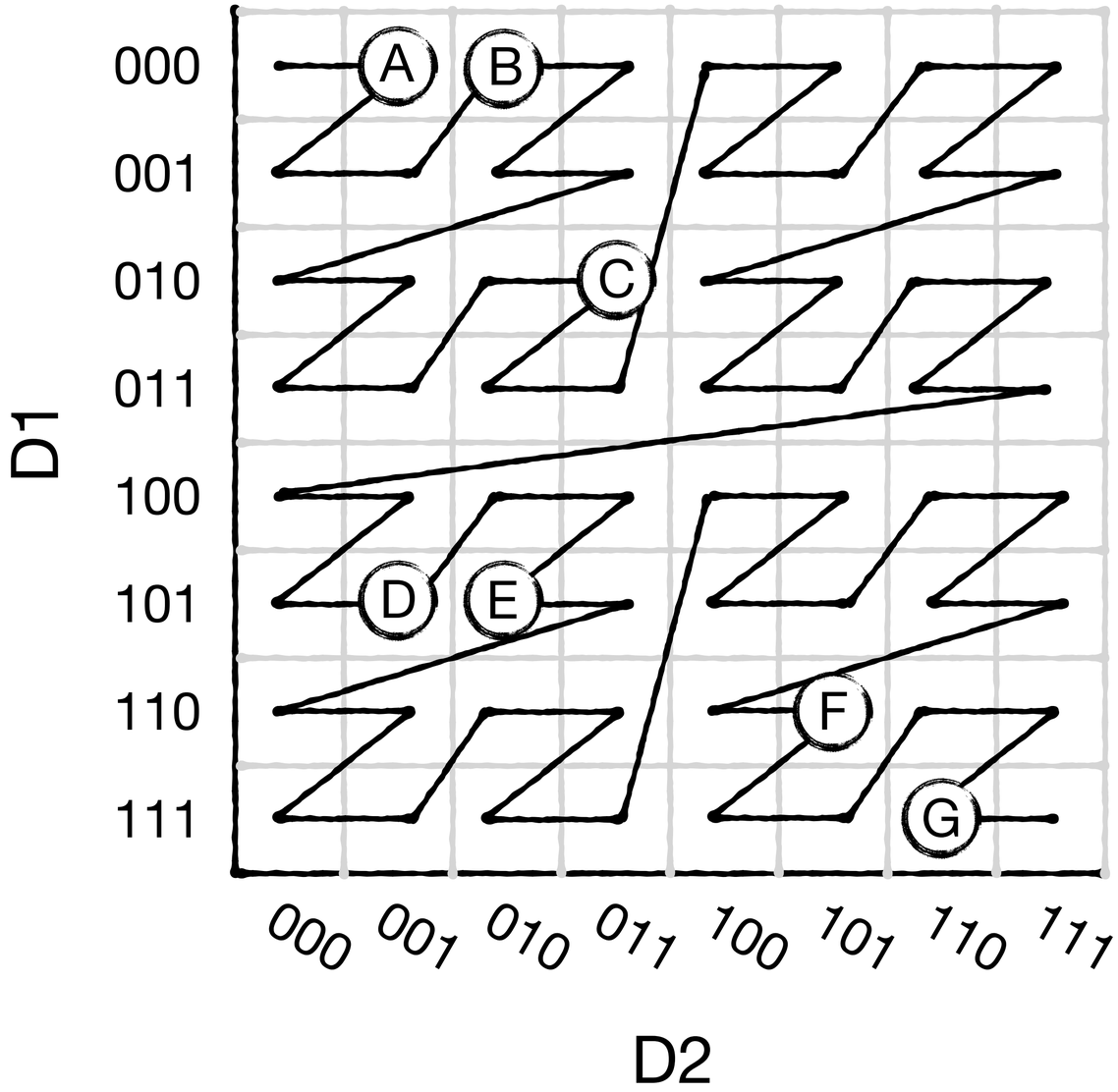

Linear order
A
B
C
D
E
F
G





Construct a B-tree

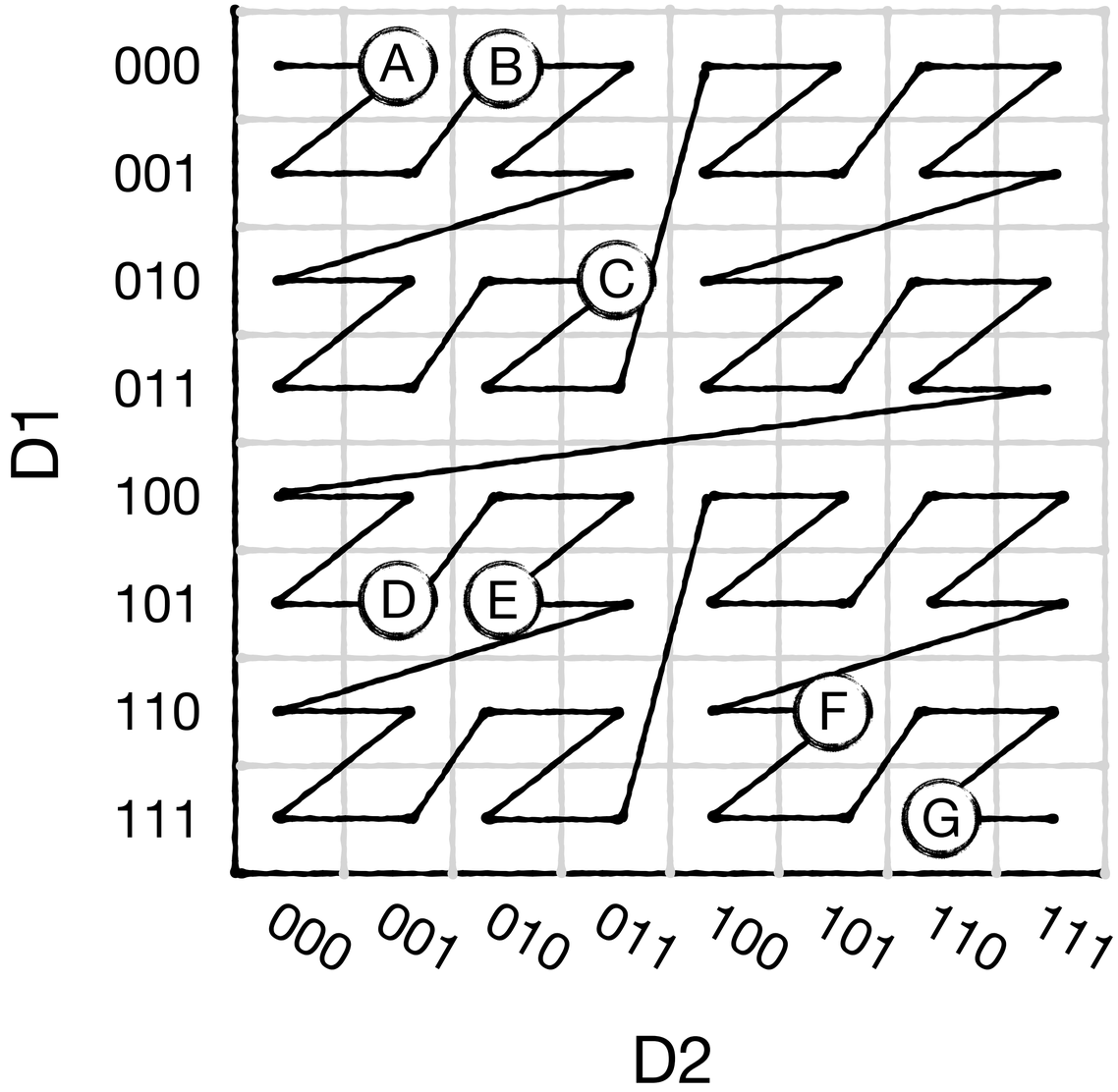
How to compare keys in the Z-Order?



A <? B

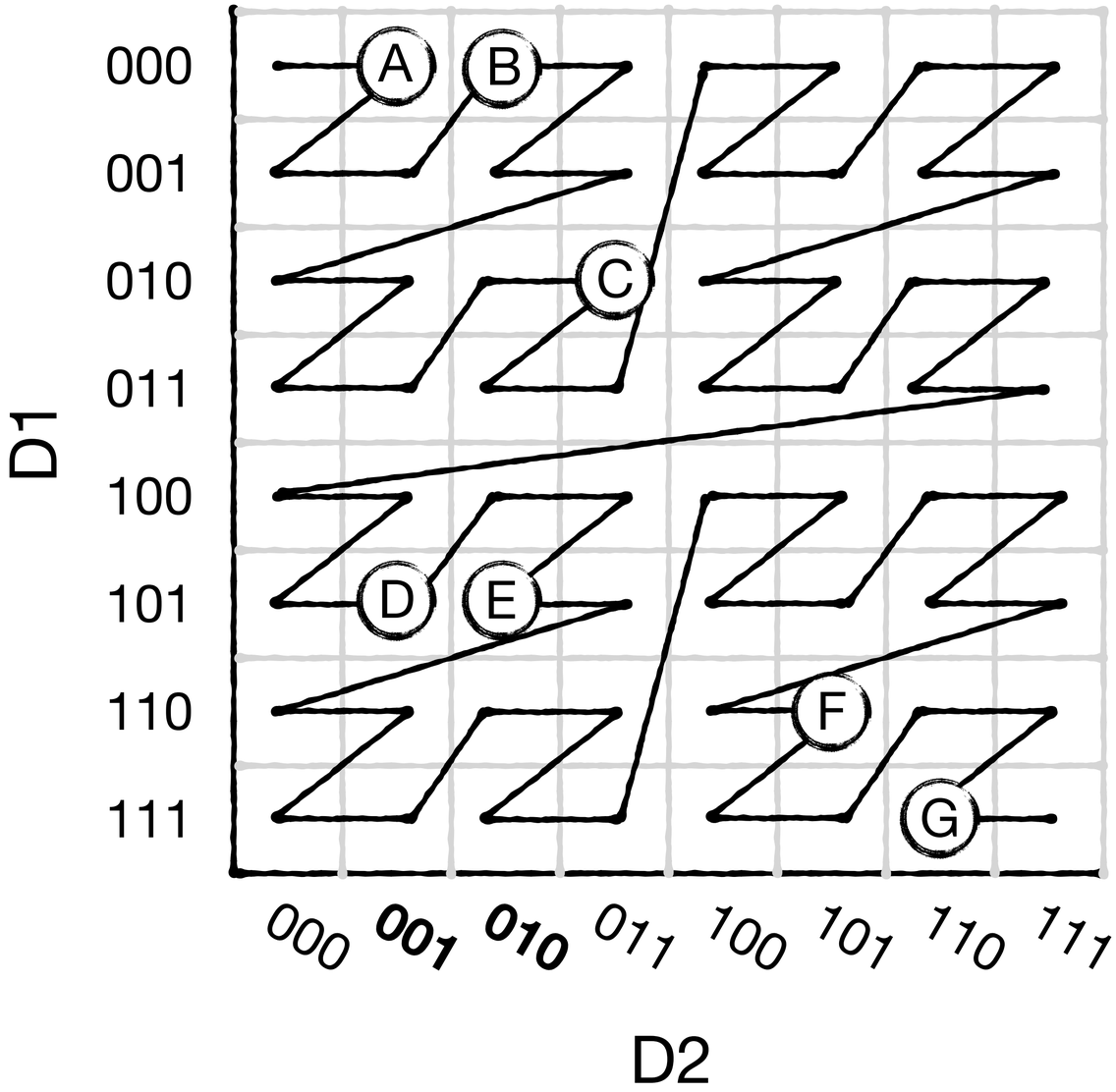
How to compare keys in the Z-Order?

Interleaving dimension bits



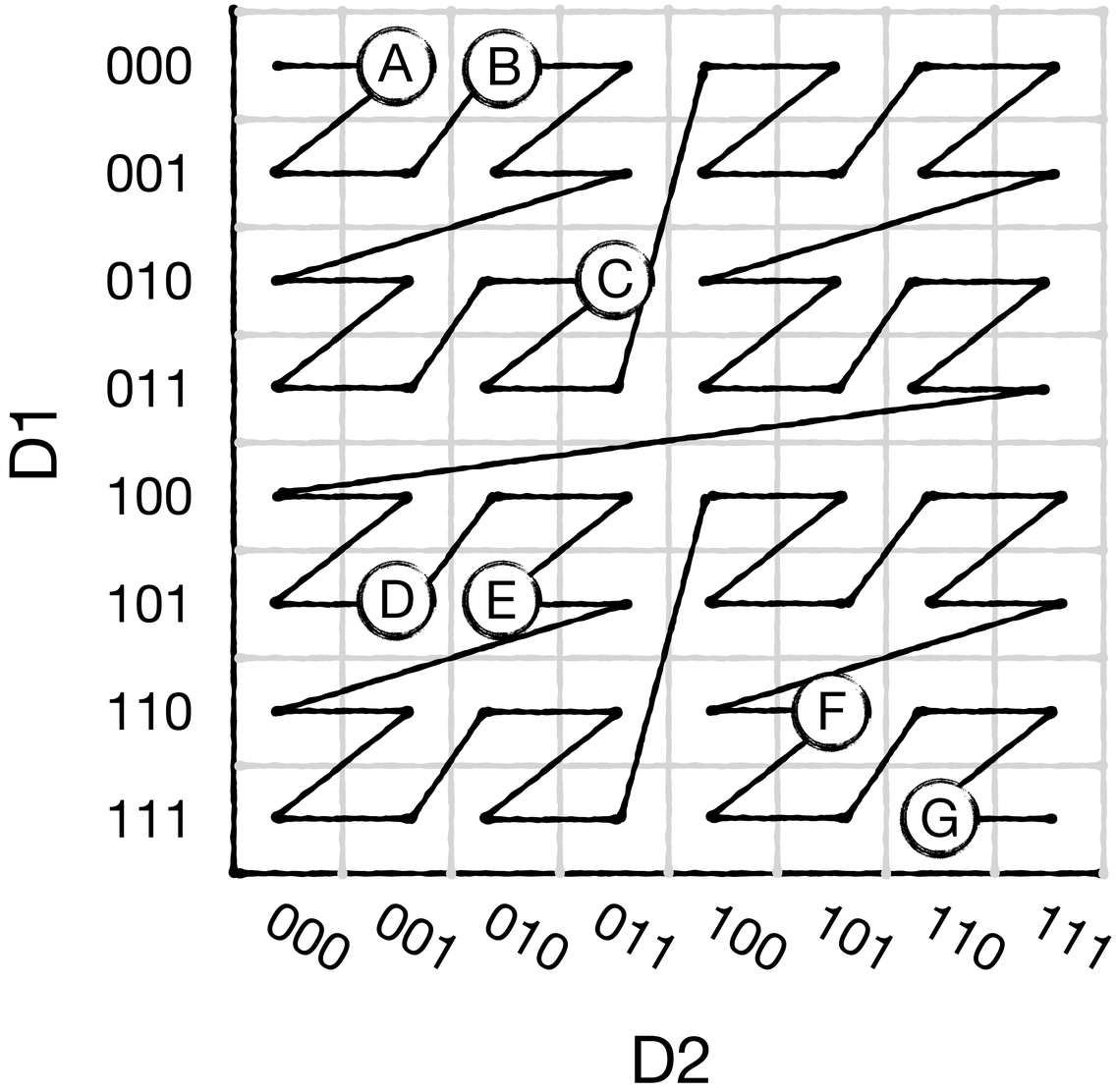
A <? B

Interleaving dimension bits



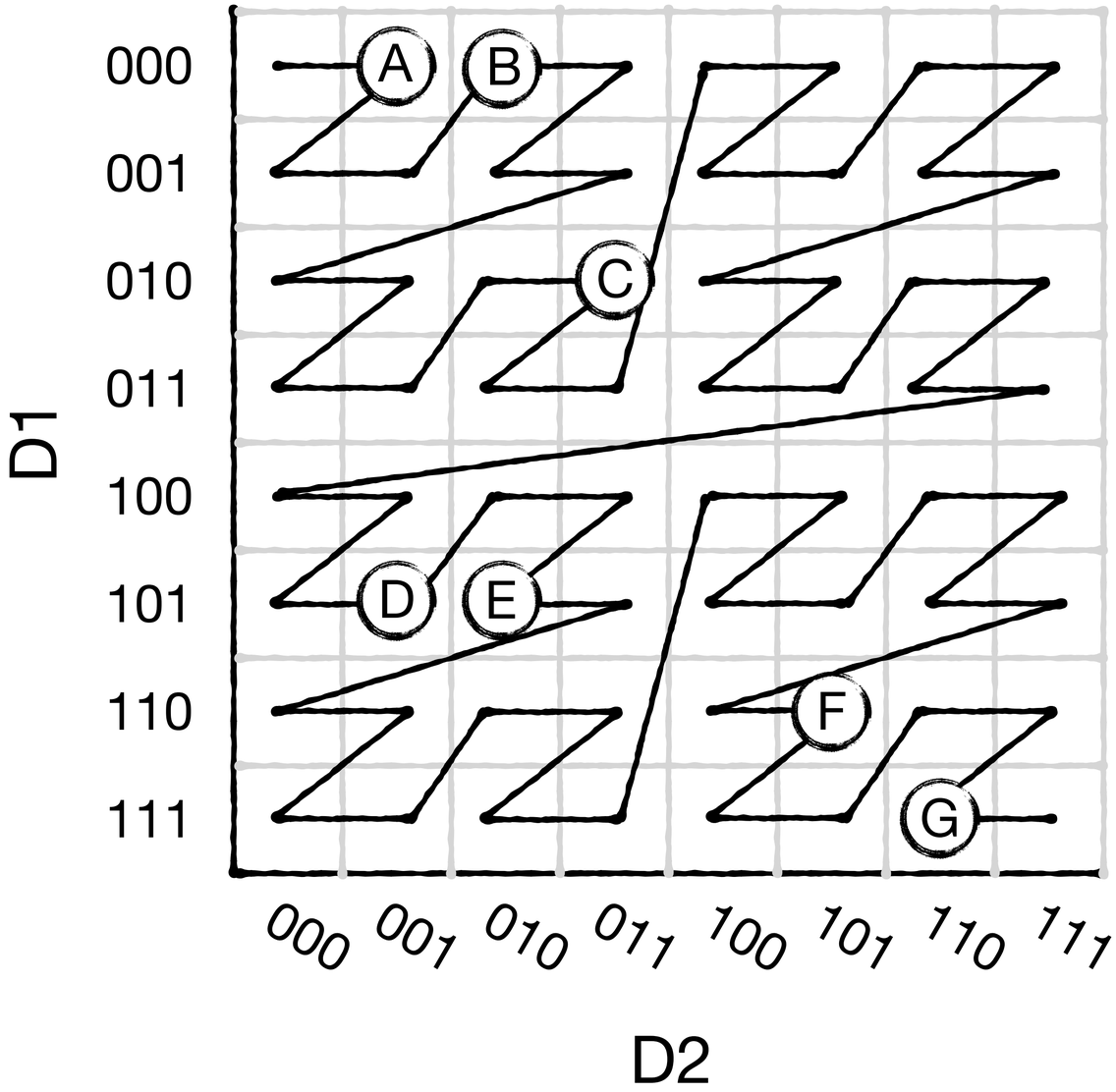
D1	**000**		**000**
D2	**001**		**010**
	Ⓐ	**<?**	Ⓑ

Interleaving dimension bits



D1	**000**		**000**
D2	**001**		**010**
Z-Key			
	Ⓐ	**<?**	Ⓑ

Interleaving dimension bits



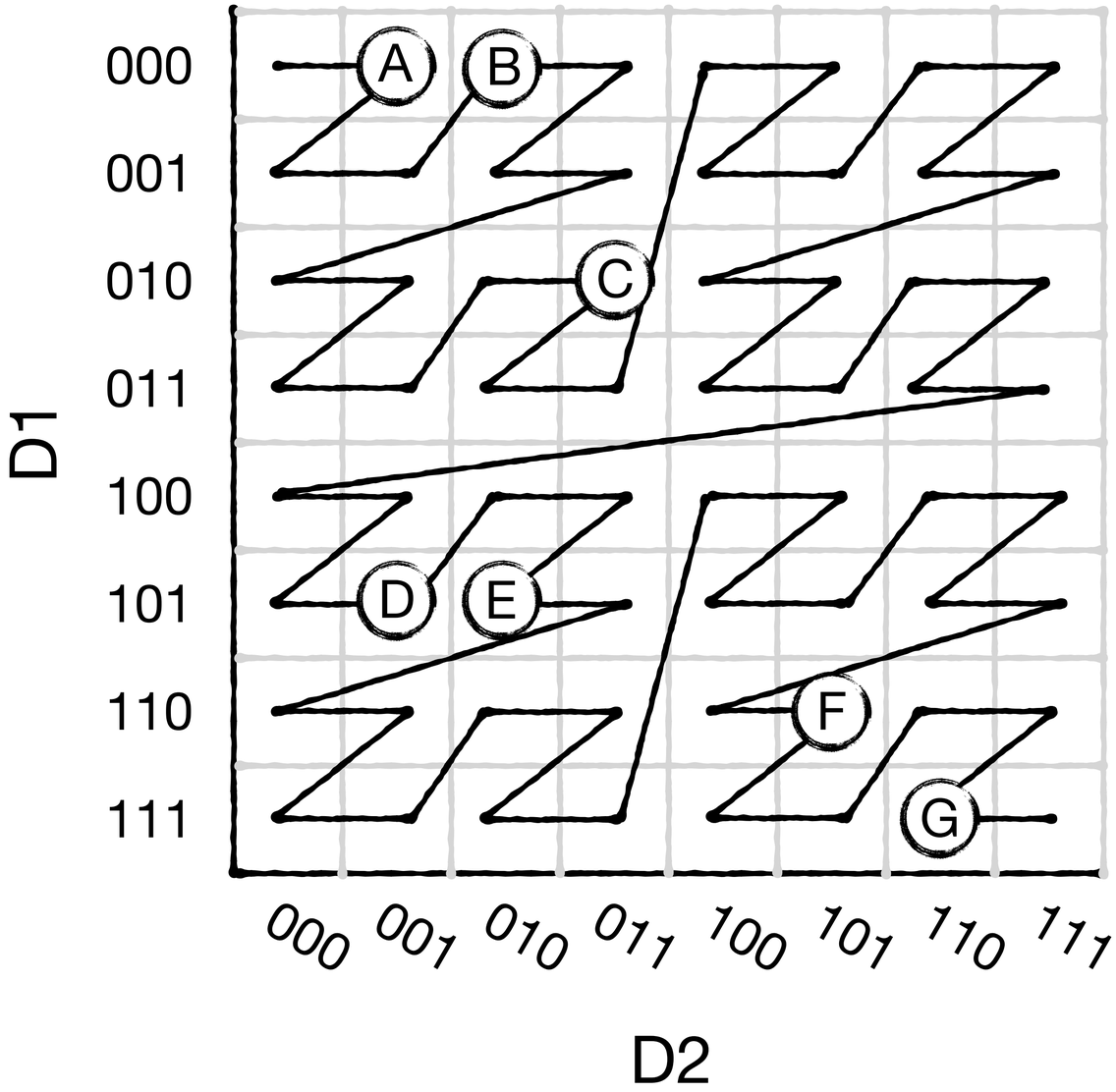
D1 **000** **000**

D2 **001** **010**

Z-Key **00**

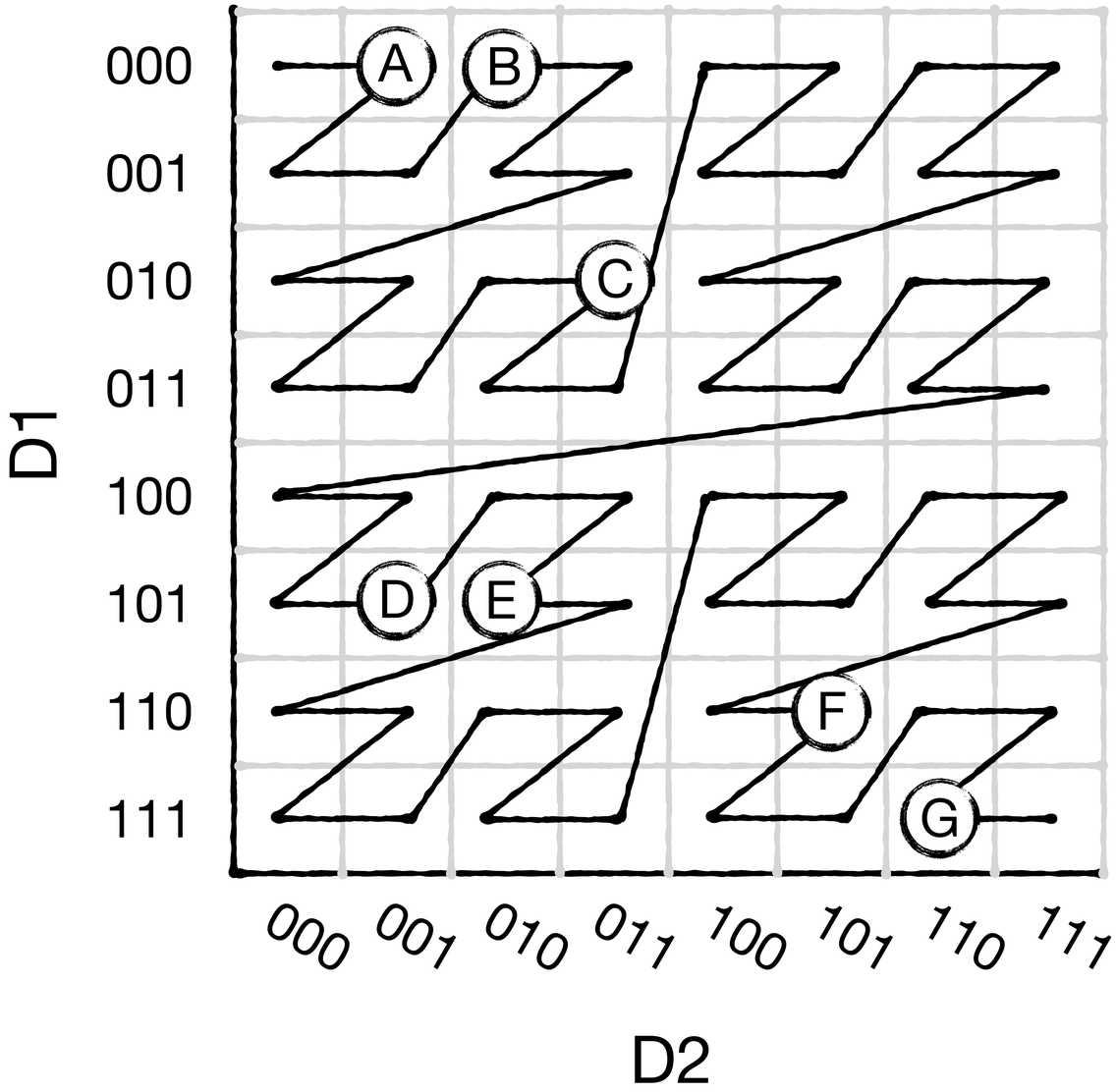
A **<?** B

Interleaving dimension bits



D1	**000**		**000**
D2	**001**		**010**
Z-Key	**00 00**		
	A	**<?**	B

Interleaving dimension bits



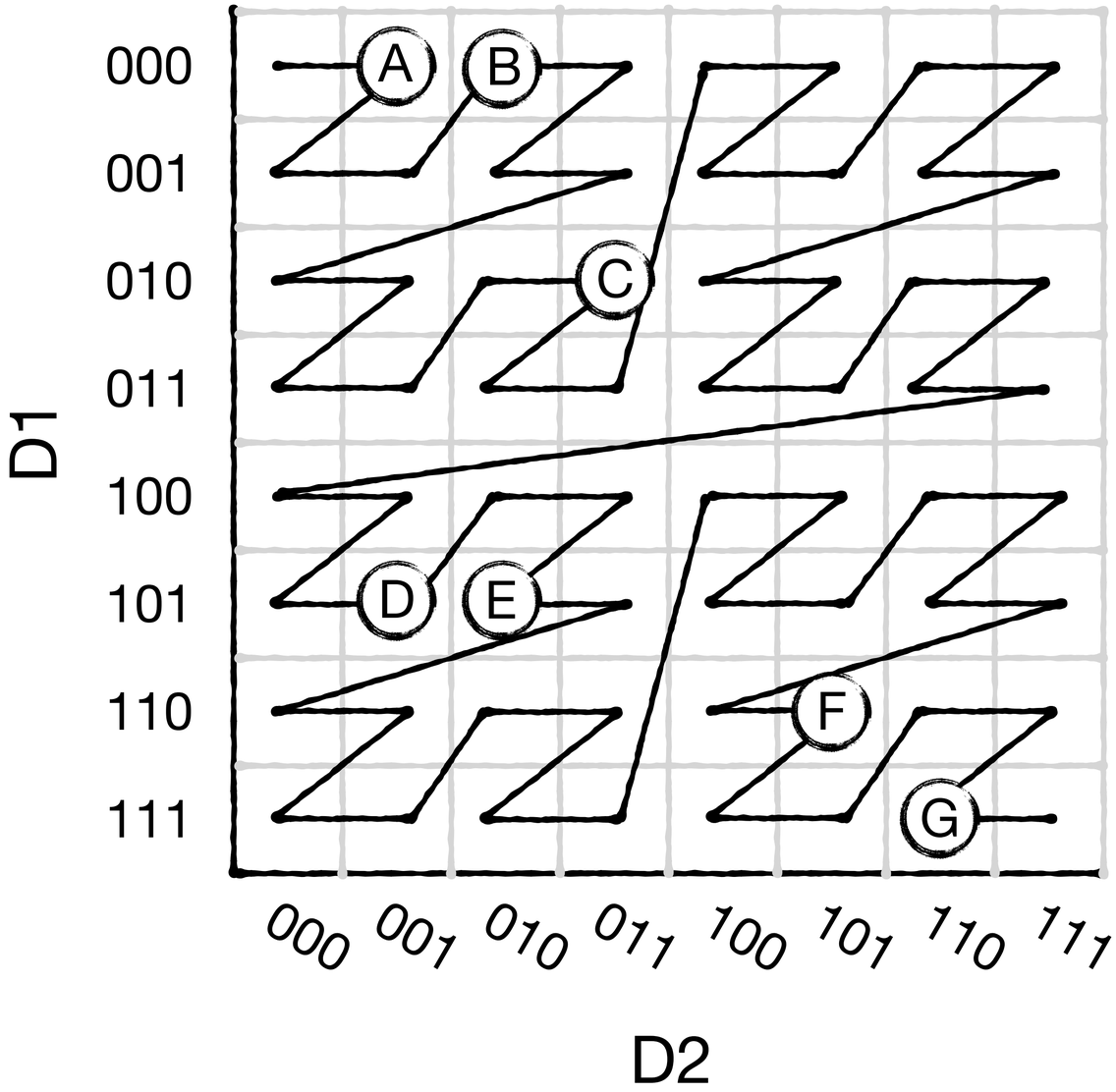
D1 000 **000**

D2 001 **010**

Z-Key 00 00 01

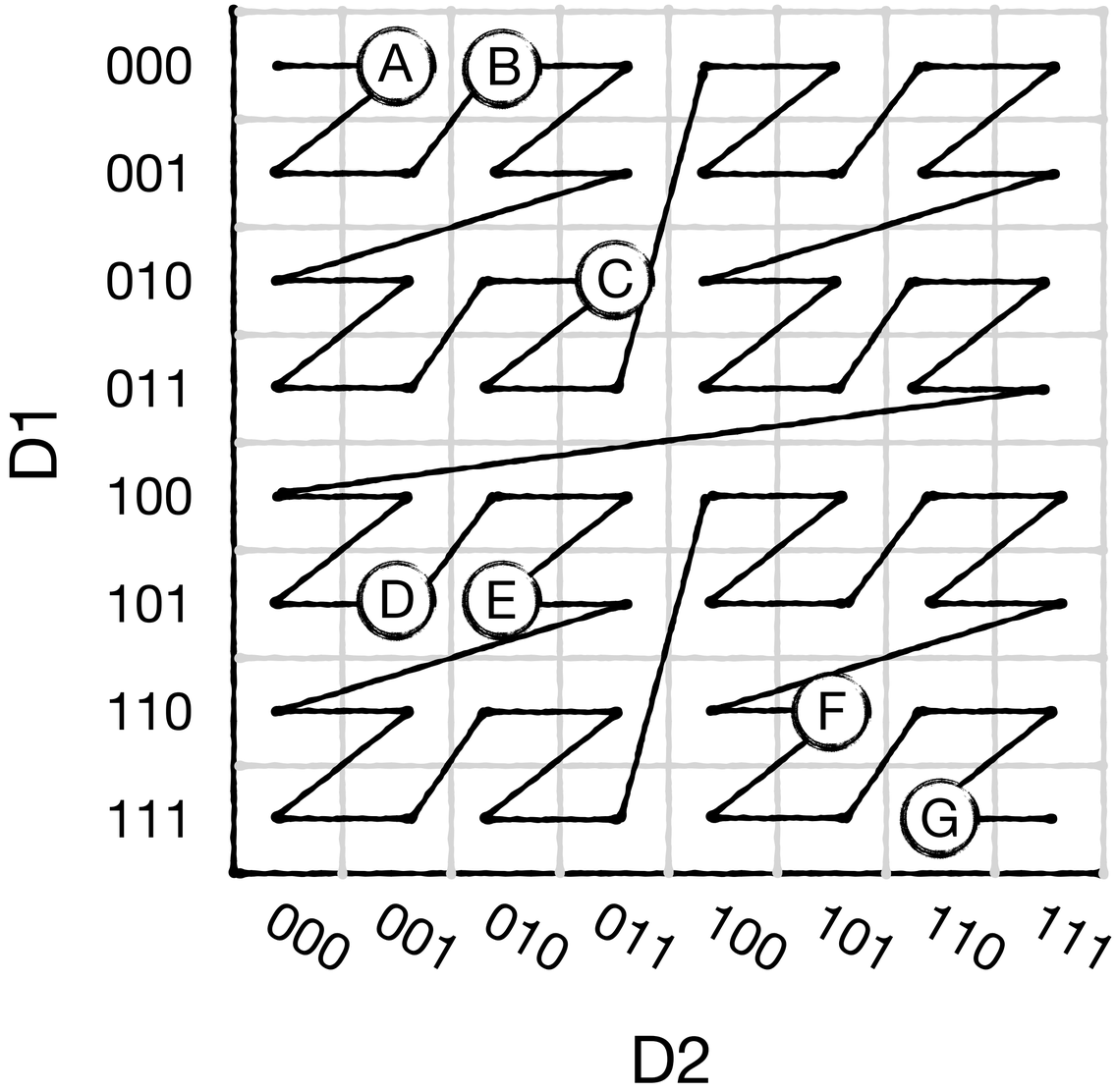
A **<?** B

Interleaving dimension bits



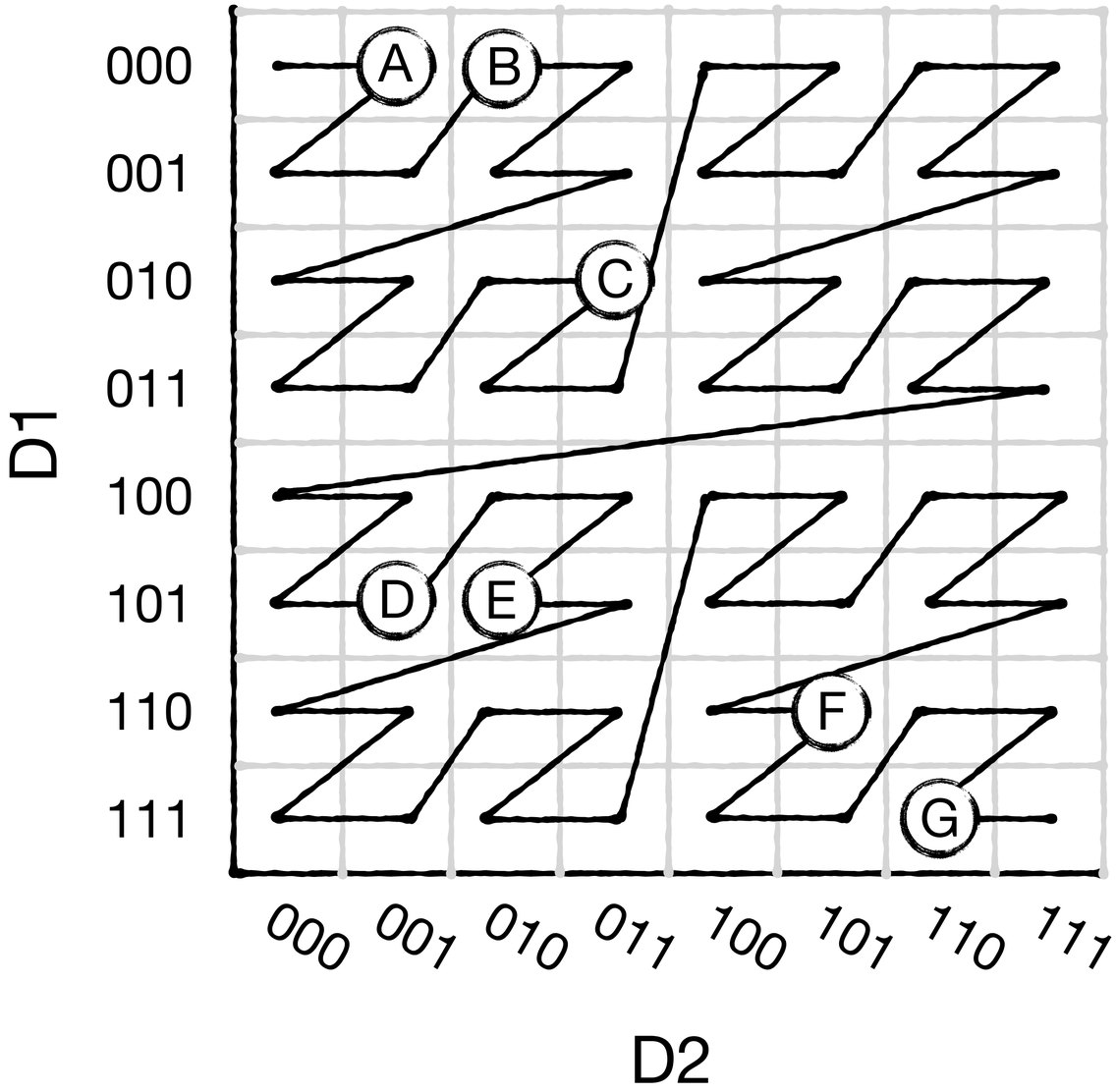
D1	**0**	**0**	**0**		**000**
D2	**0**	**0**	**1**		**010**
Z-Key	**00**	**00**	**01**	**00**	
	A			**<?**	B

Interleaving dimension bits

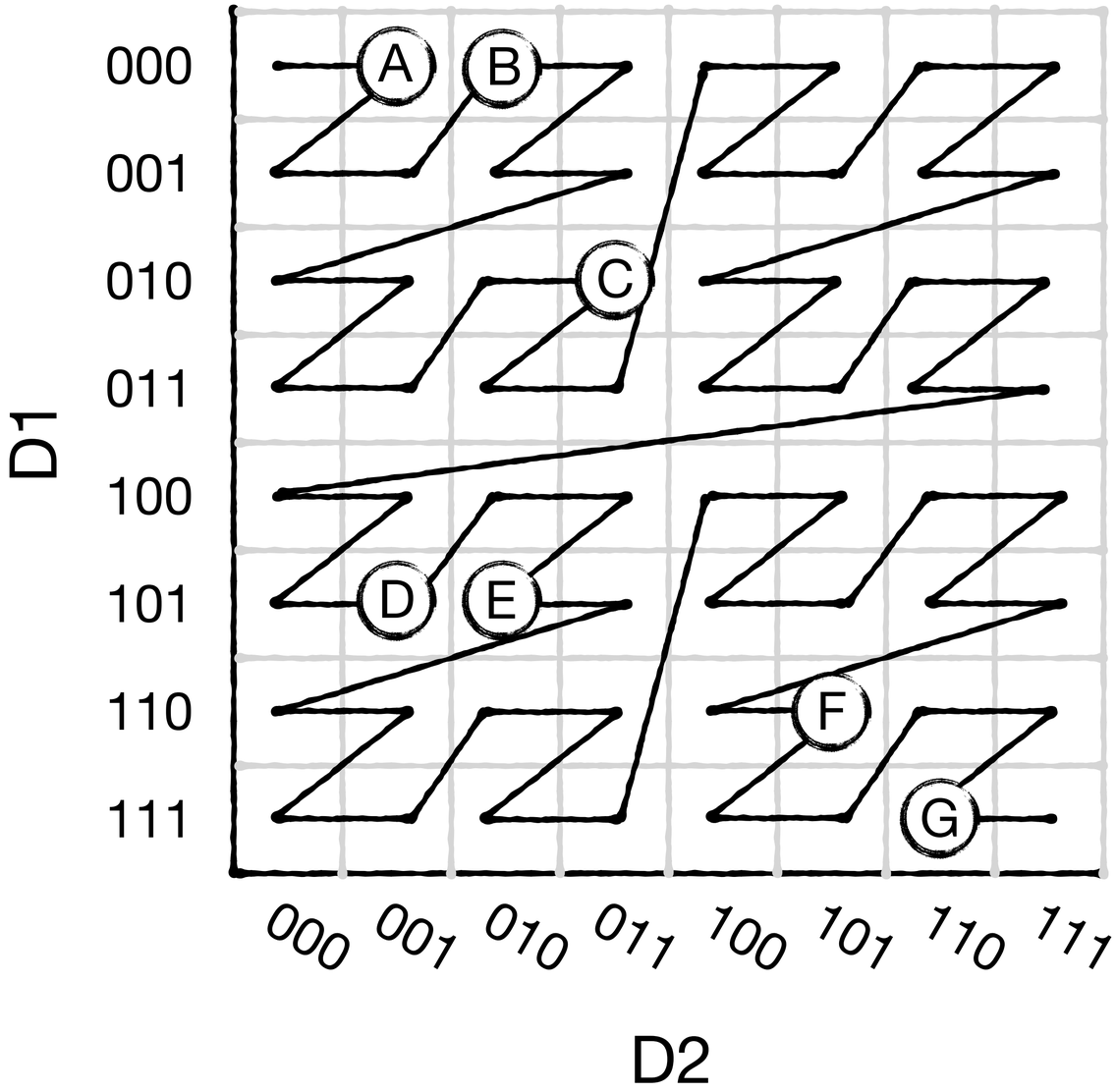


D1	**0**	**0**	**0**		**0**	**0**	**0**
D2	**0**	**0**	**1**		**0**	**1**	**0**
Z-Key	**00**	**00**	**01**		**00**	**01**	
	(A)			**<?**	(B)		

Interleaving dimension bits

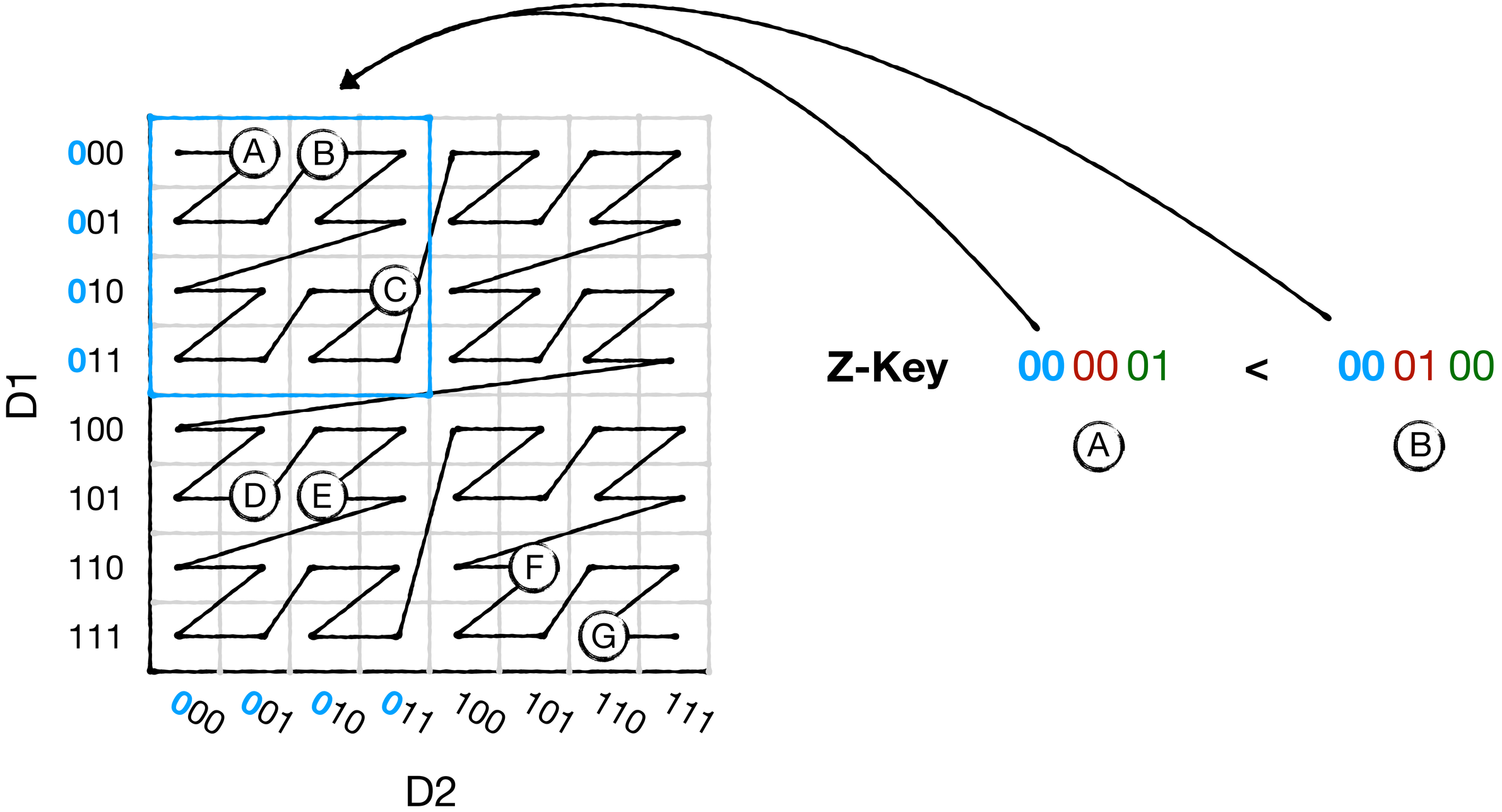


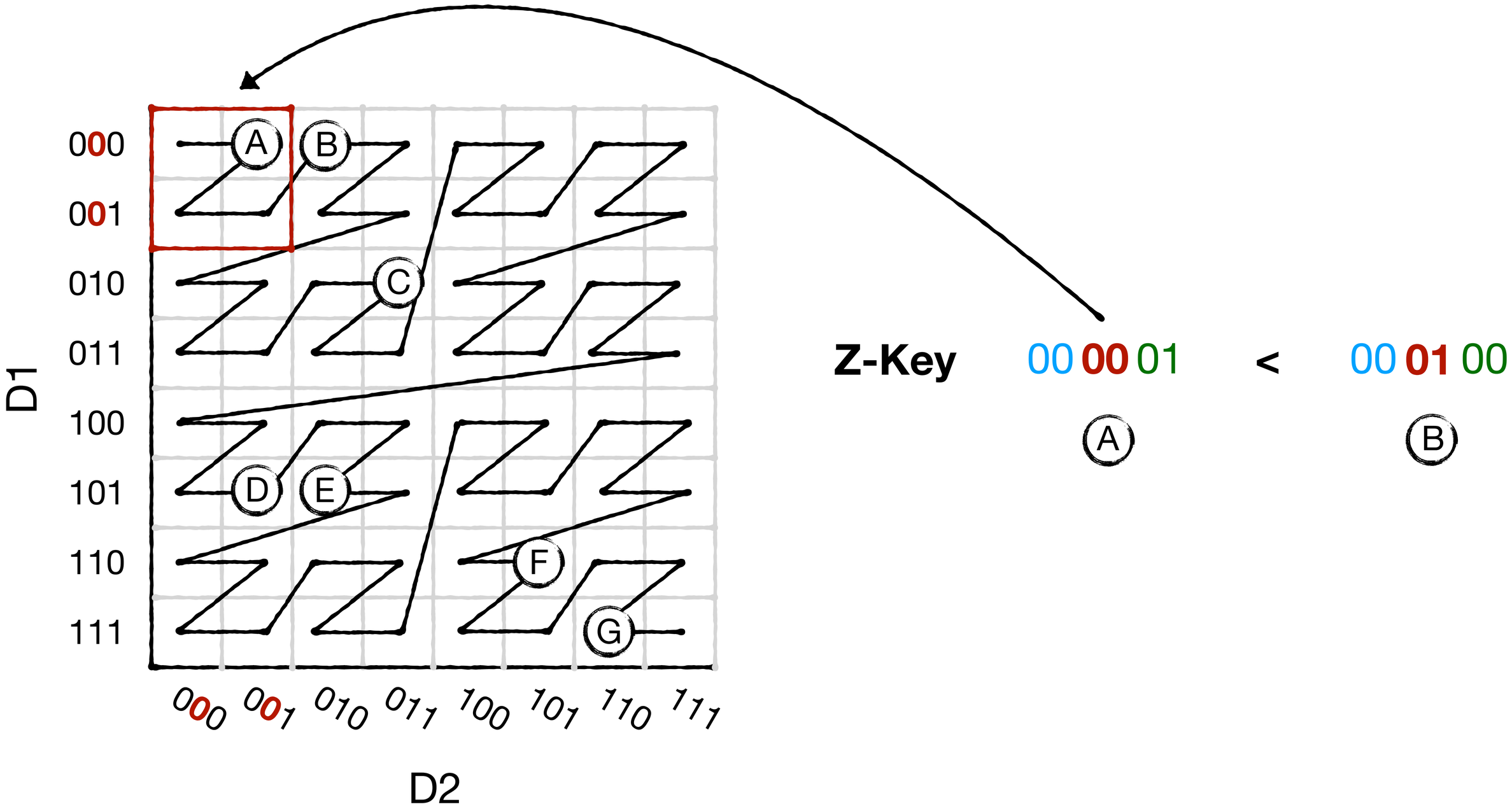
D1	**0**	**0**	**0**			**0**	**0**	**0**
D2	**0**	**0**	**1**			**0**	**1**	**0**
Z-Key	**00**	**00**	**01**			**00**	**01**	**00**
	A			**<?**		B		

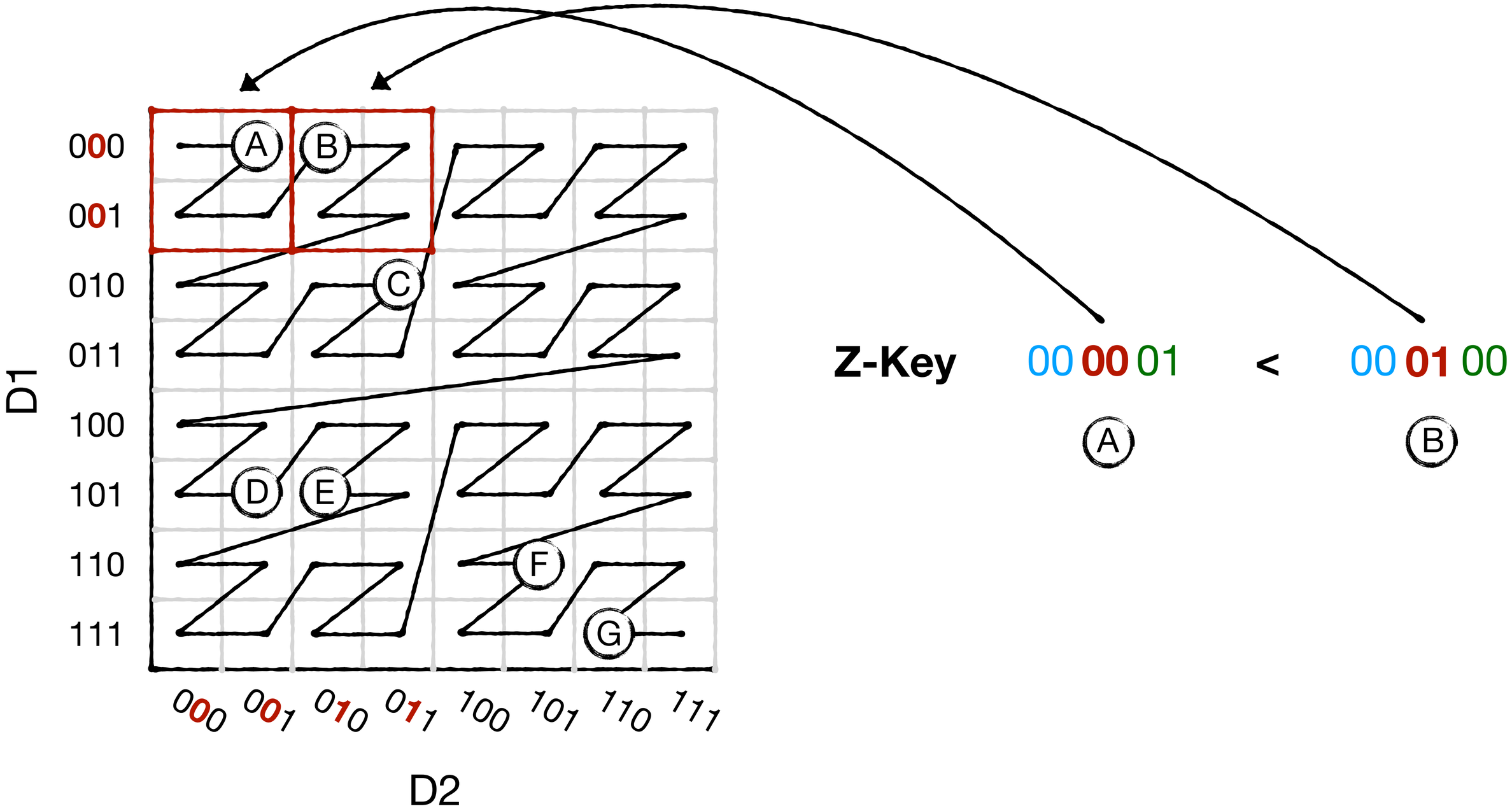


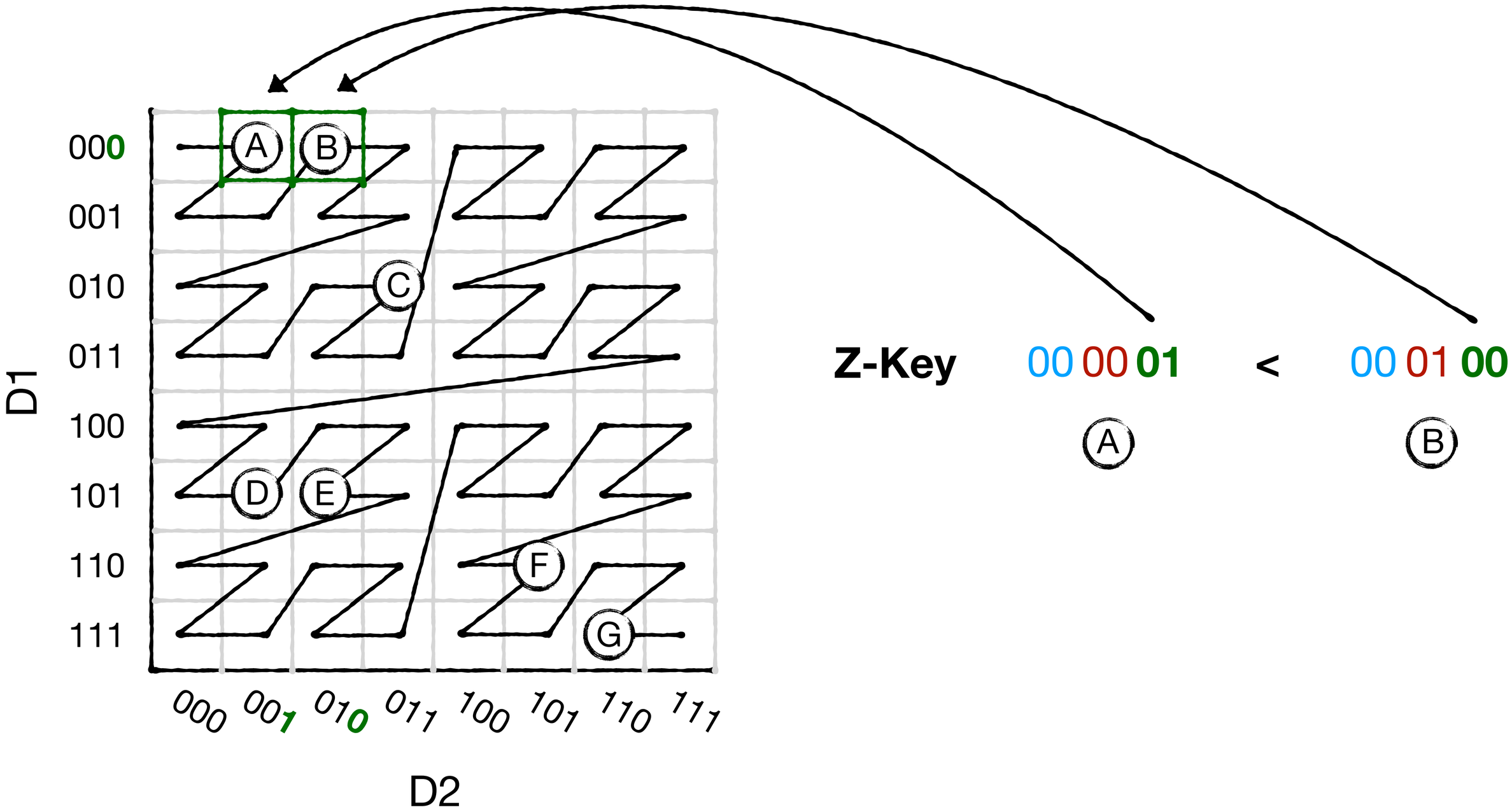
Z-Key 00 00 01 < 00 01 00

A B

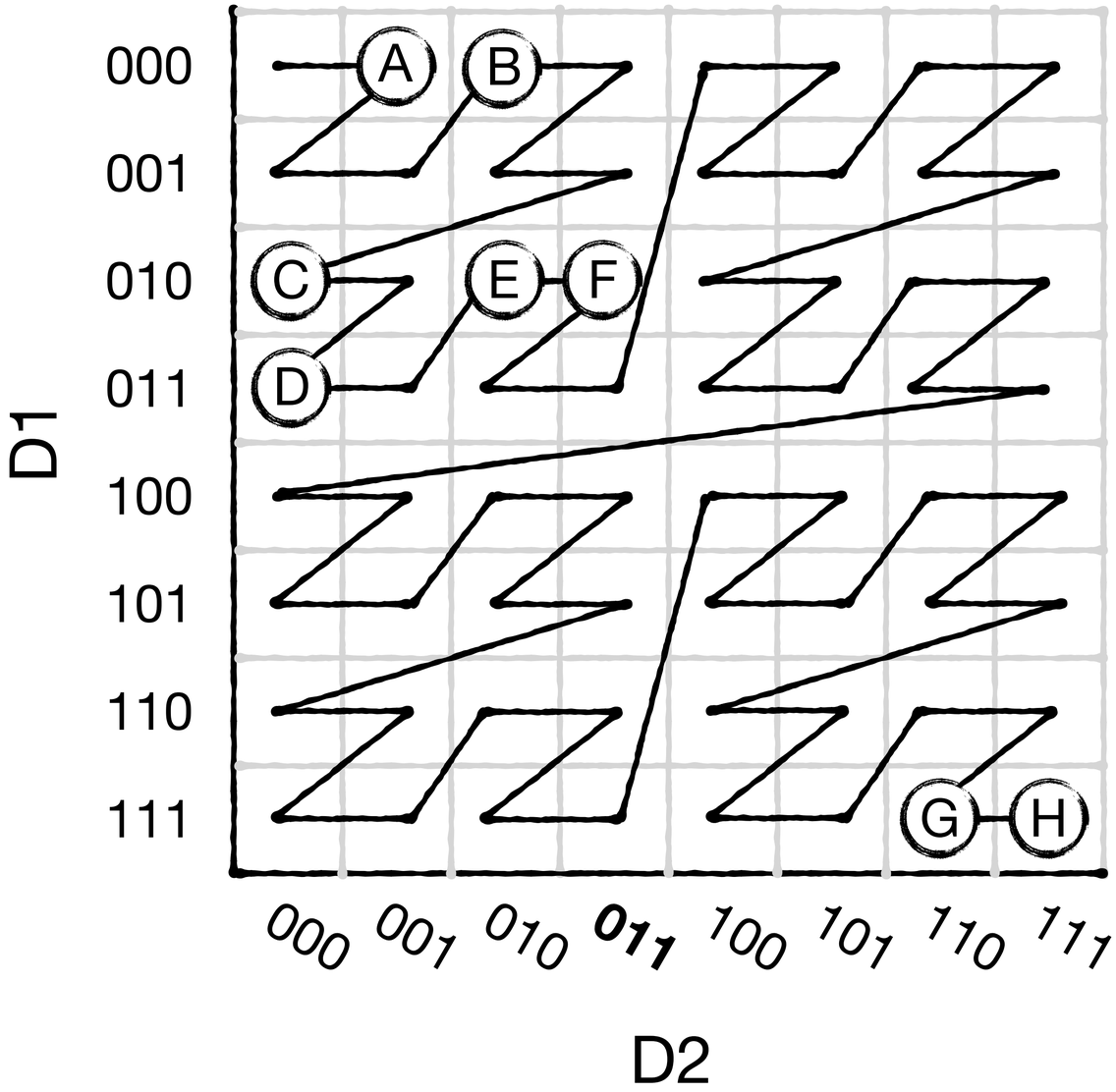

000
001
010
011
100
101
110
111
D1
A
B
C
D
E
F
G
000 001 010 011 100 101 110 111
D2
Z-Key
00 00 01
<
00 01 00
A
B


D1
000
001
010
011
100
101
110
111
A
B
C
D
E
F
G
000 001 010 011 100 101 110 111
D2
Z-Key
00 00 01
<
00 01 00
A
B

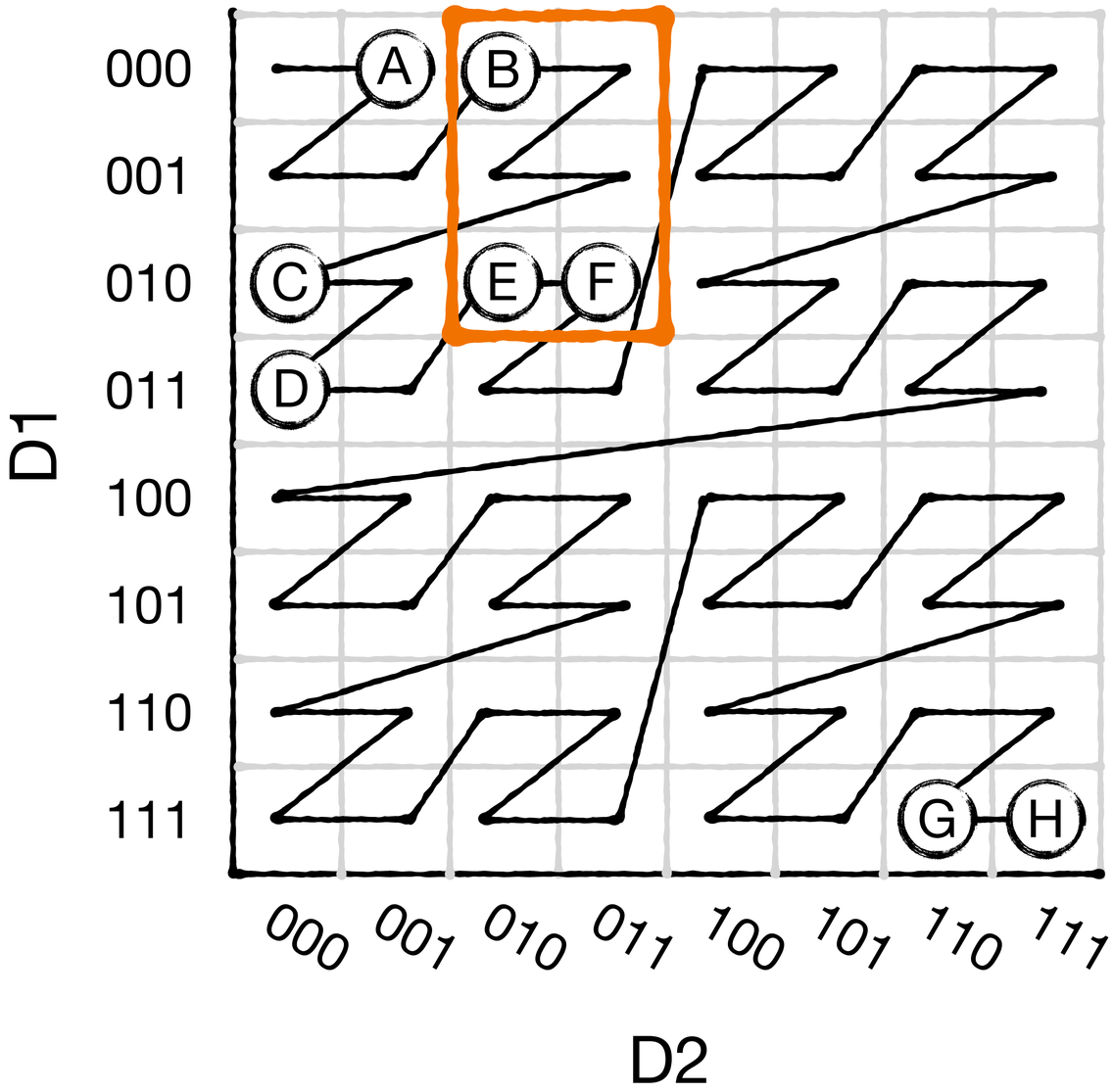

000
001
010
011
100
101
110
111
D1
A
B
C
D
E
F
G
000 001 010 011 100 101 110 111
D2
Z-Key
00 00 01
<
00 01 00
A
B


000
001
010
011
100
101
110
111
D1
A
B
C
D
E
F
G
000 001 010 011 100 101 110 111
D2
Z-Key
00 00 01
<
00 01 00
A
B

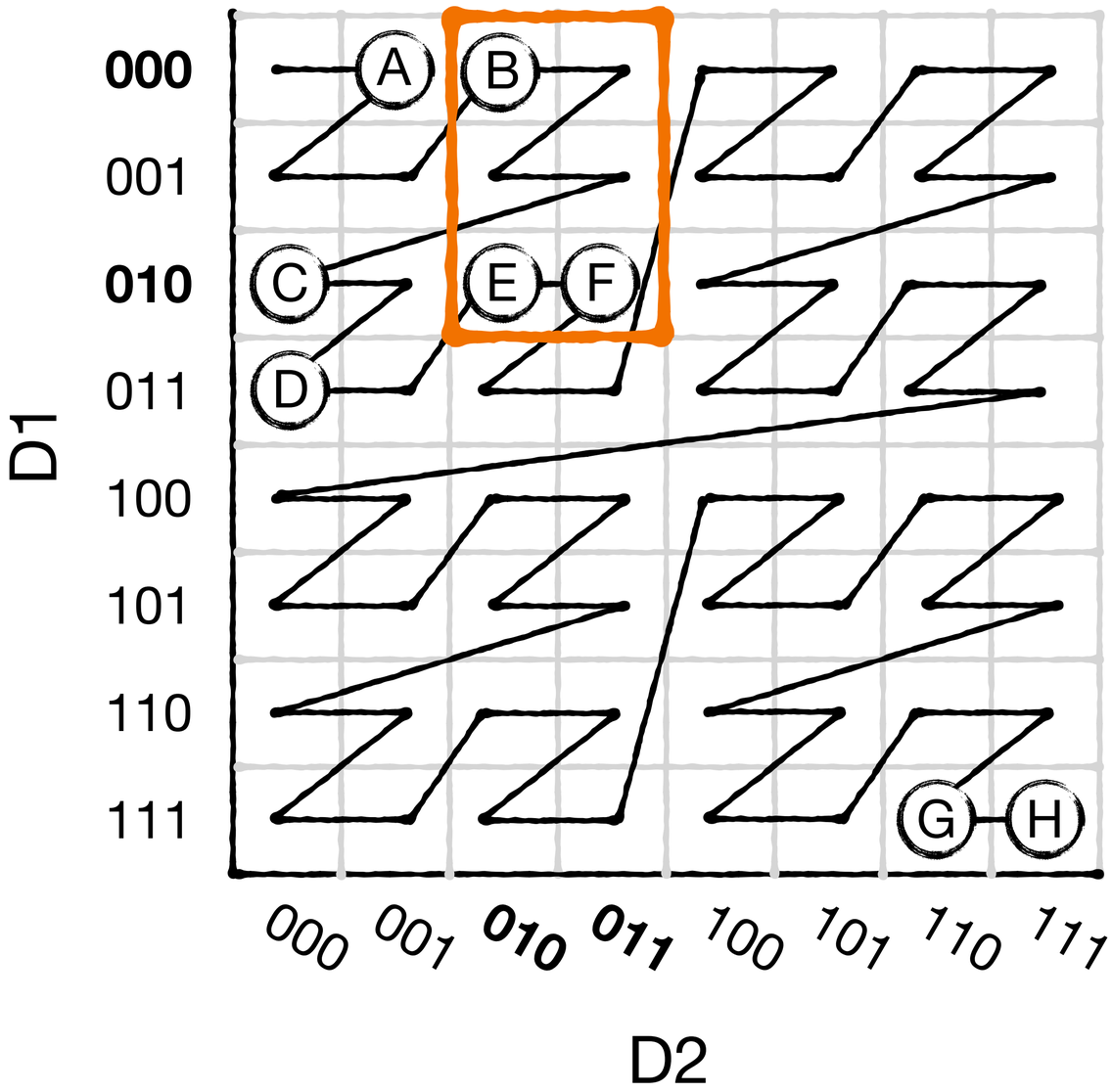
Query Handling



Query Handling



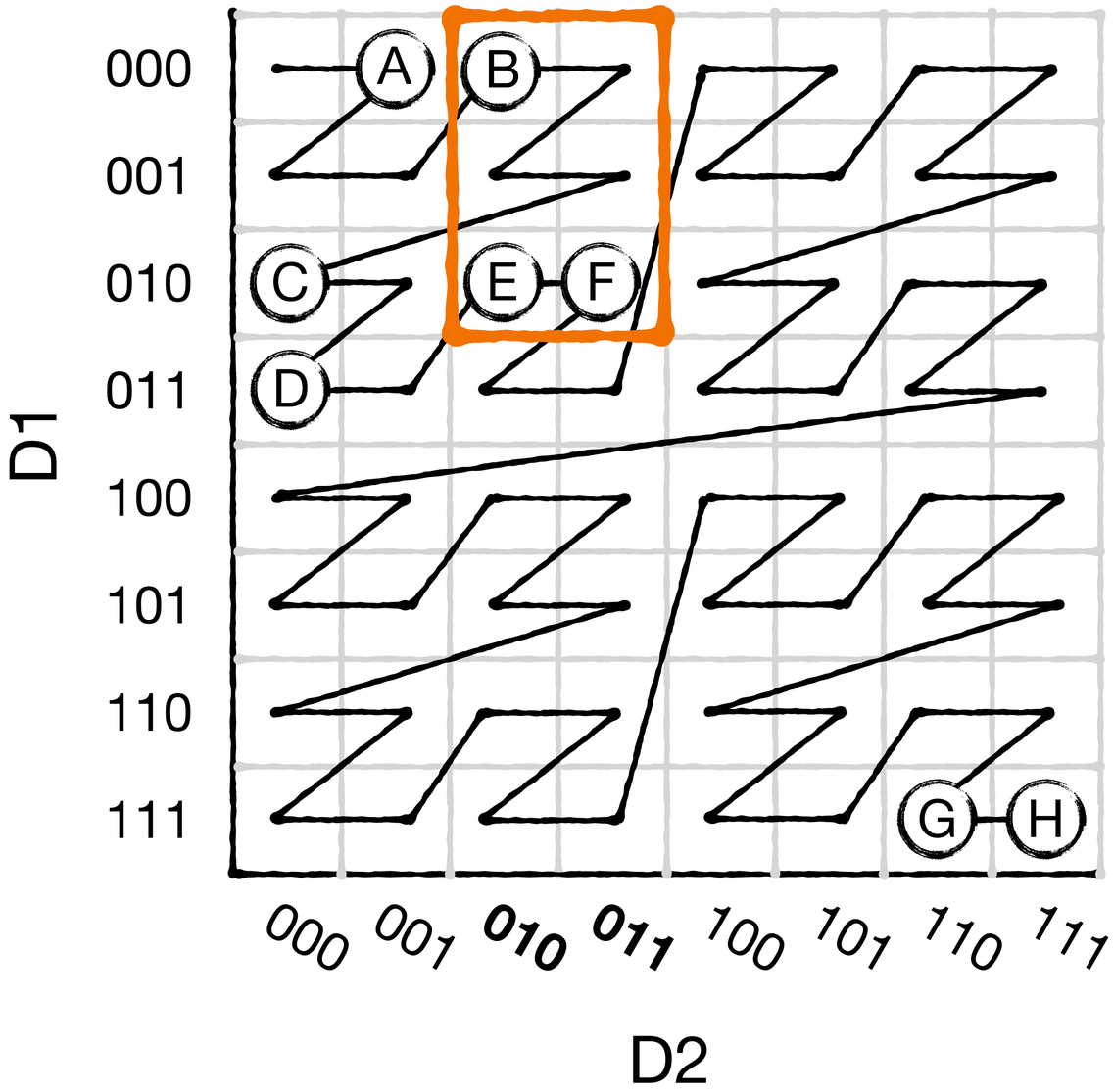
Query is a rectangle defined by start and end Z-key

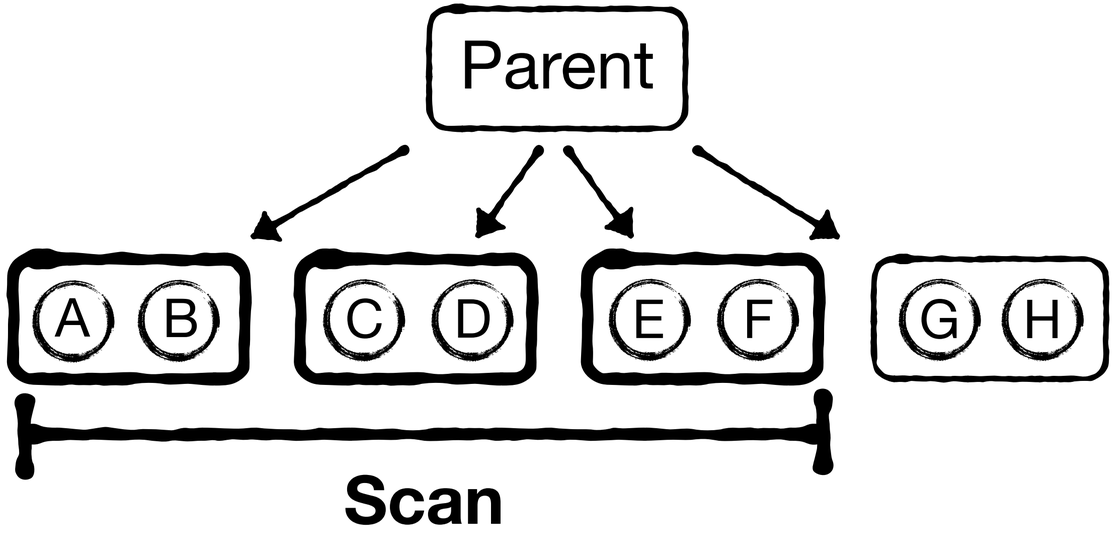


Example

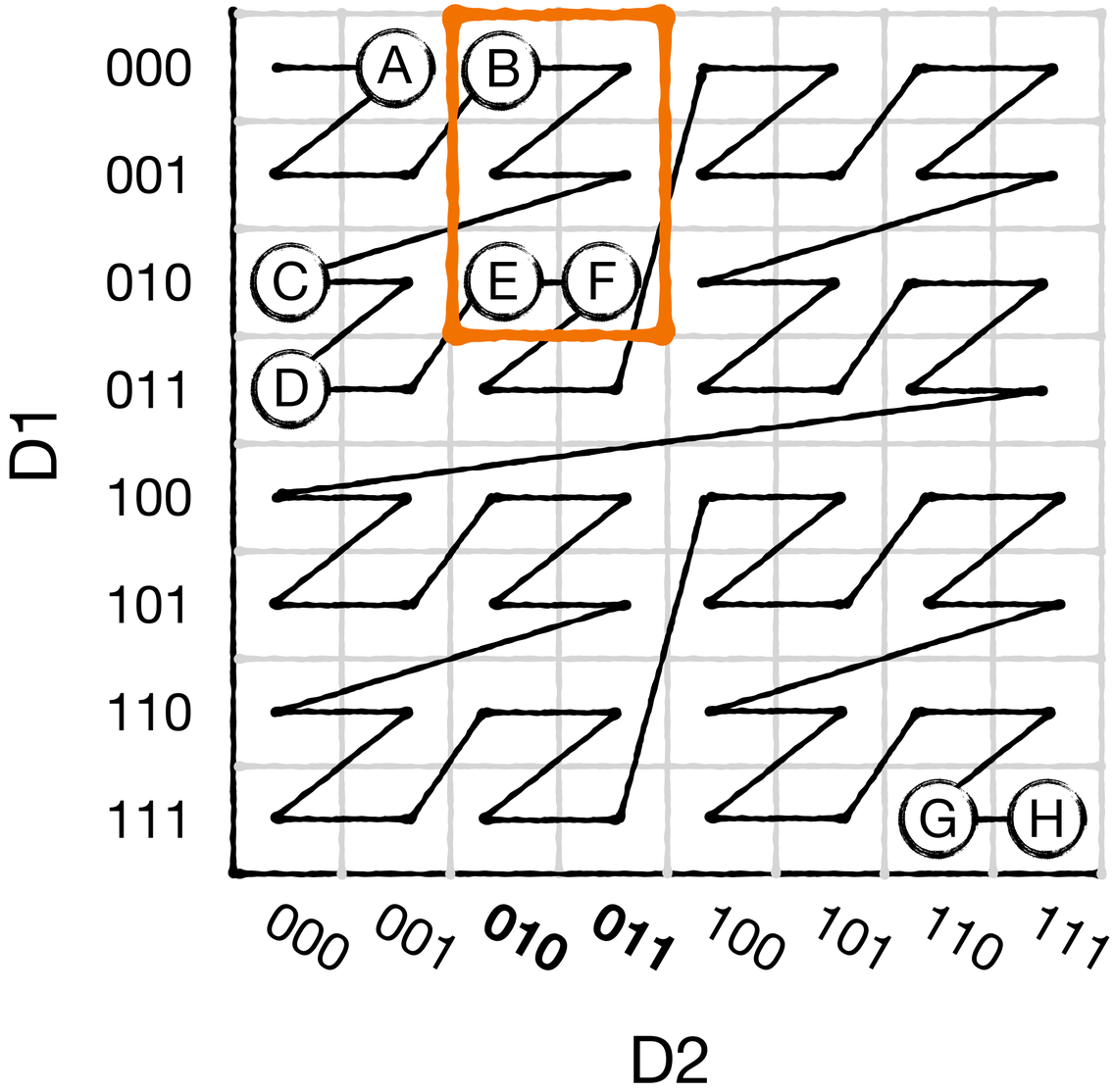
start: 00 01 00

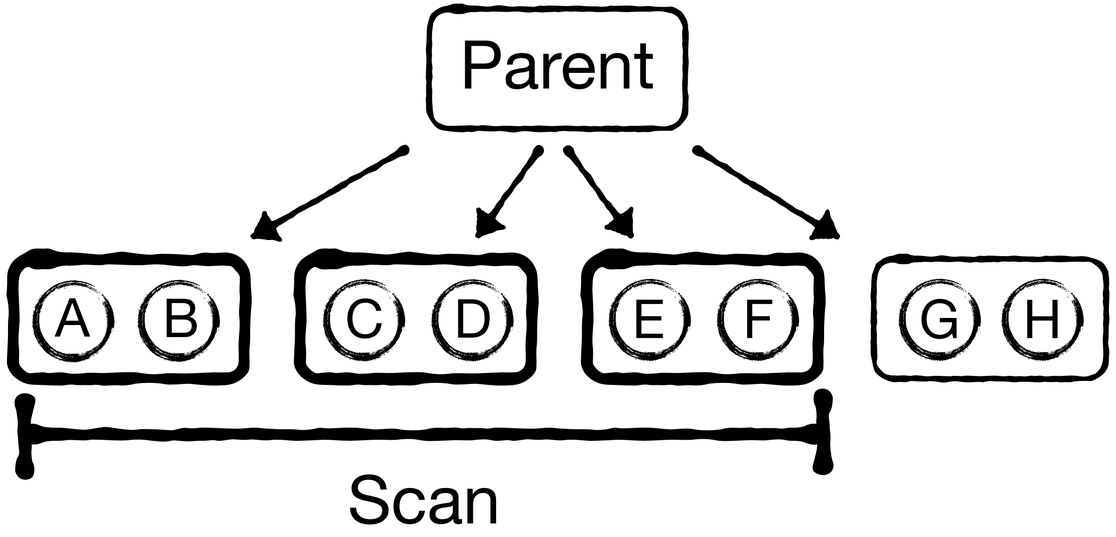
end: 00 11 01


000
001
010
011
100
101
110
111
D1
A
B
C
D
E
F
G
H
000 001 010 011 100 101 110 111
D2

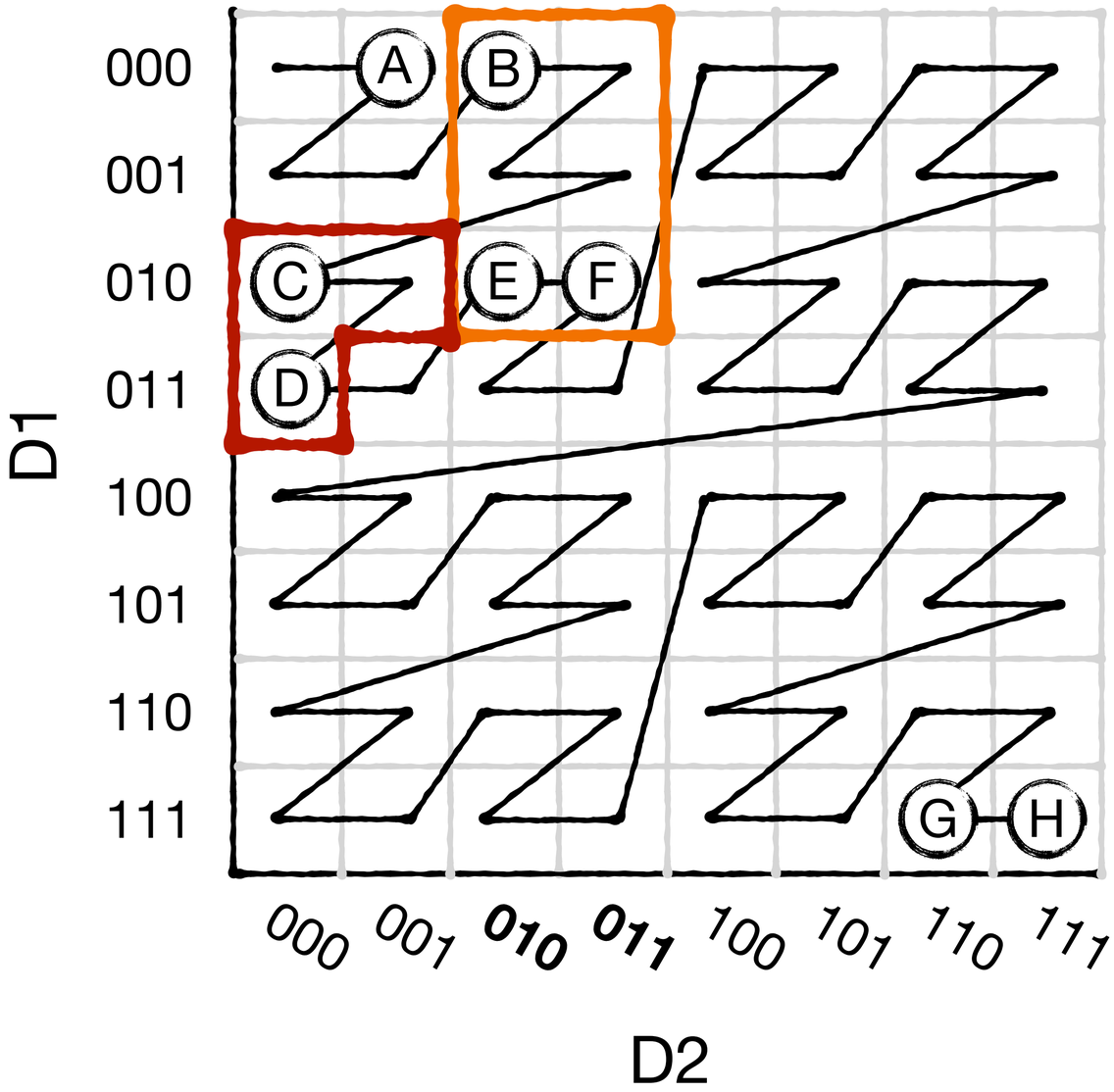

Parent
A B
C D
E F
G H
Scan

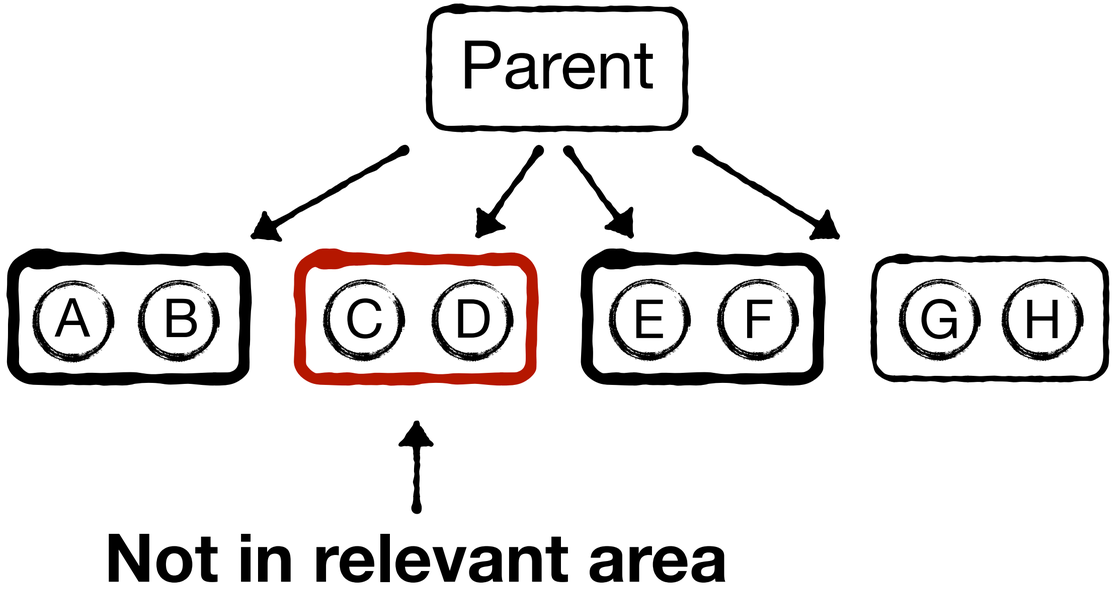
Issues?


000
001
010
011
100
101
110
111
A
B
C
D
E
F
G
H
D1
000 001 010 011 100 101 110 111
D2

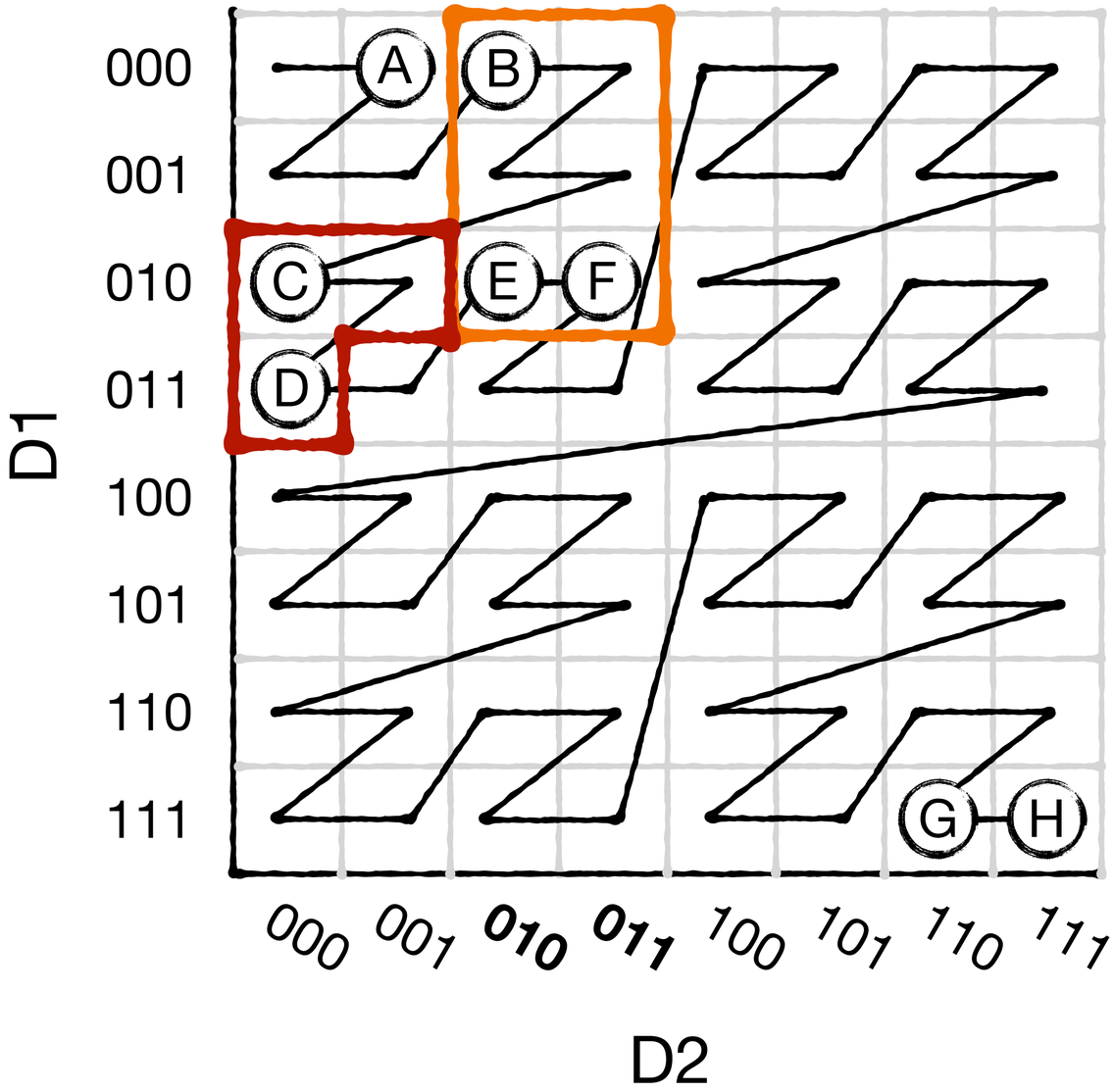

Parent
A B
C D
E F
G H
Scan

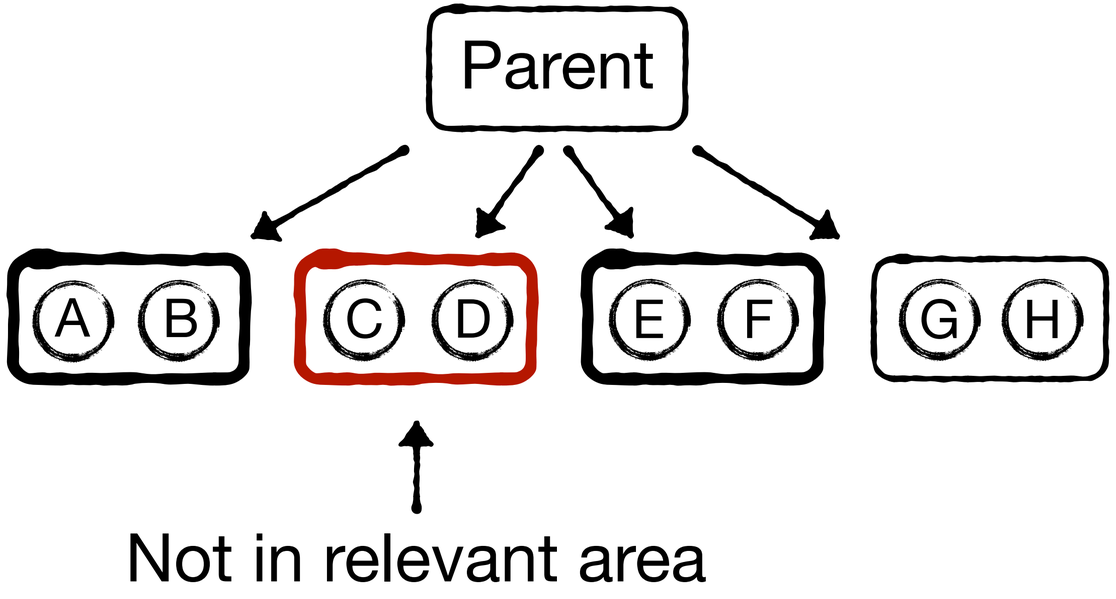
Issues?


000
001
010
011
100
101
110
111
A
B
C
D
E
F
G
H
D1
000 001 010 011 100 101 110 111
D2

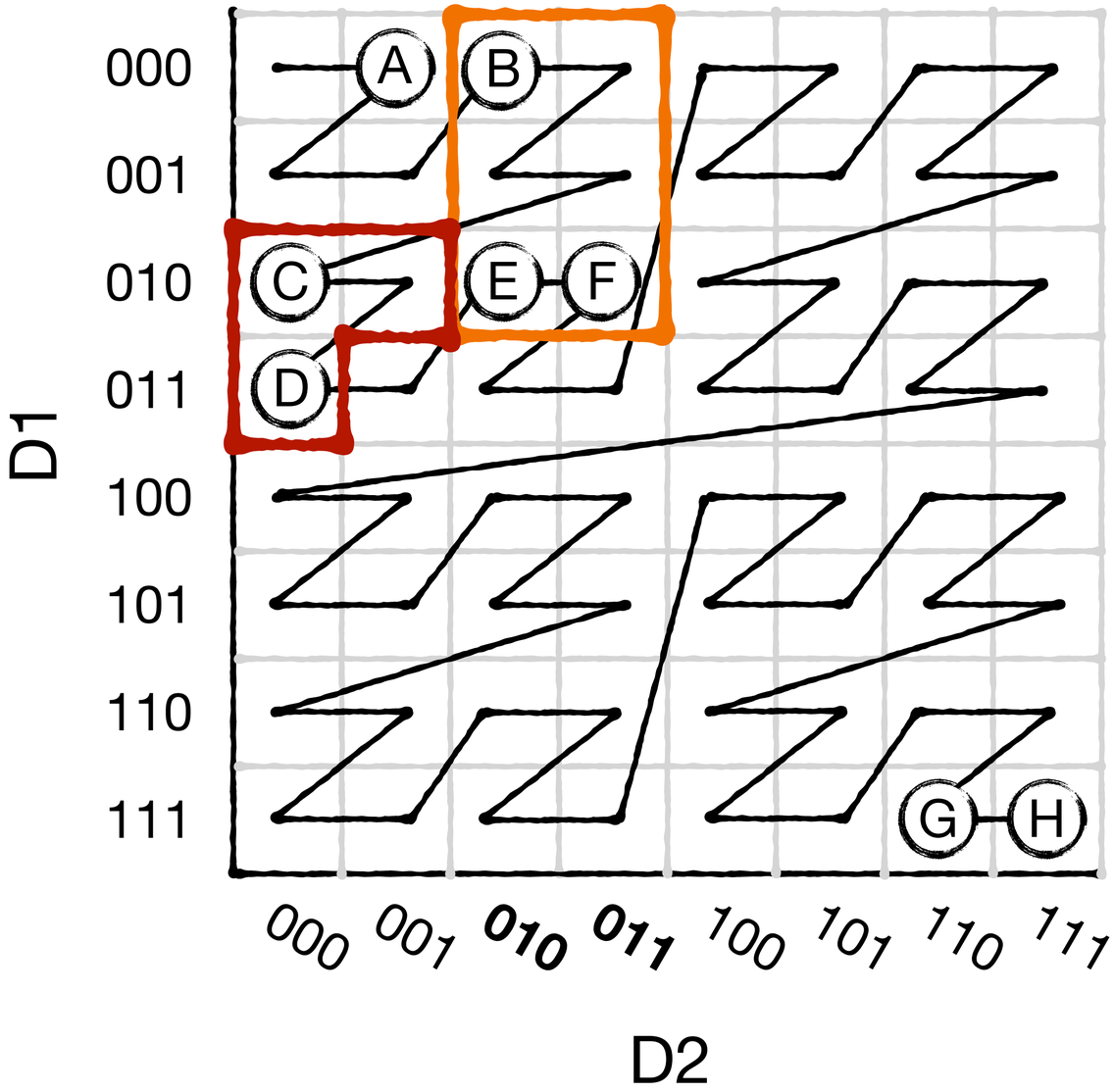

Parent
A B
C D
E F
G H
Not in relevant area

How to skip such nodes to speed up search?

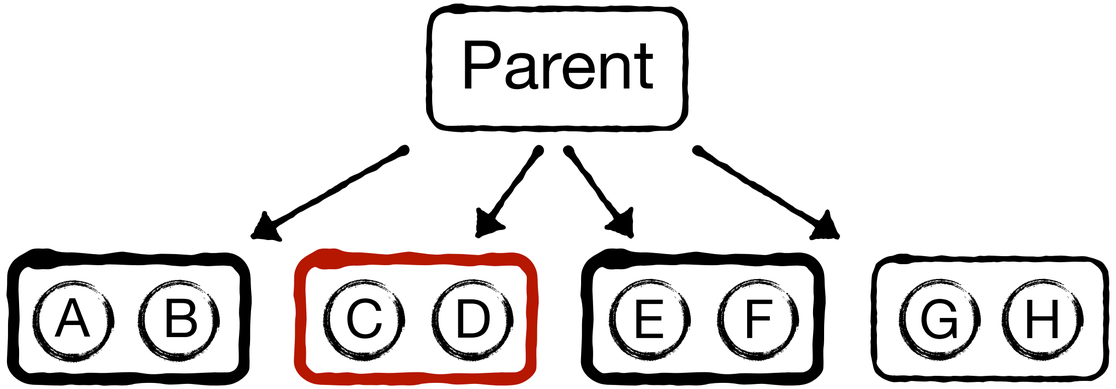




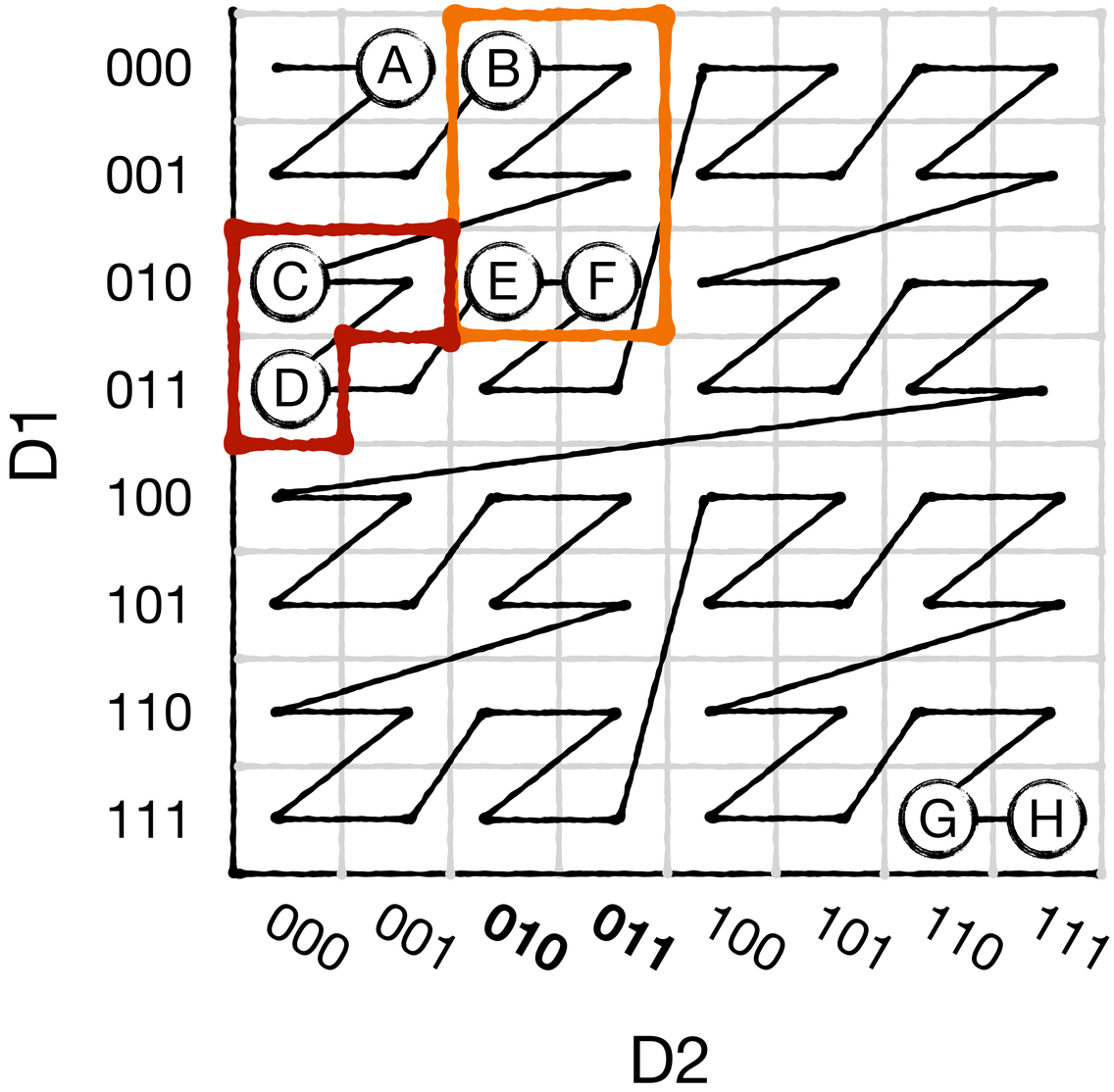
How to skip such nodes to speed up search?



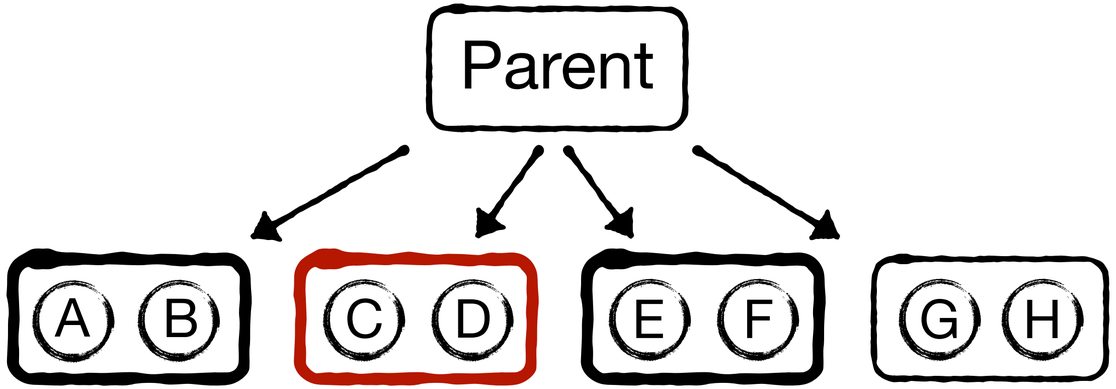
	Query	Node
start:	**00 01 00**	**00 10 00**
end:	**00 11 01**	**00 10 10**



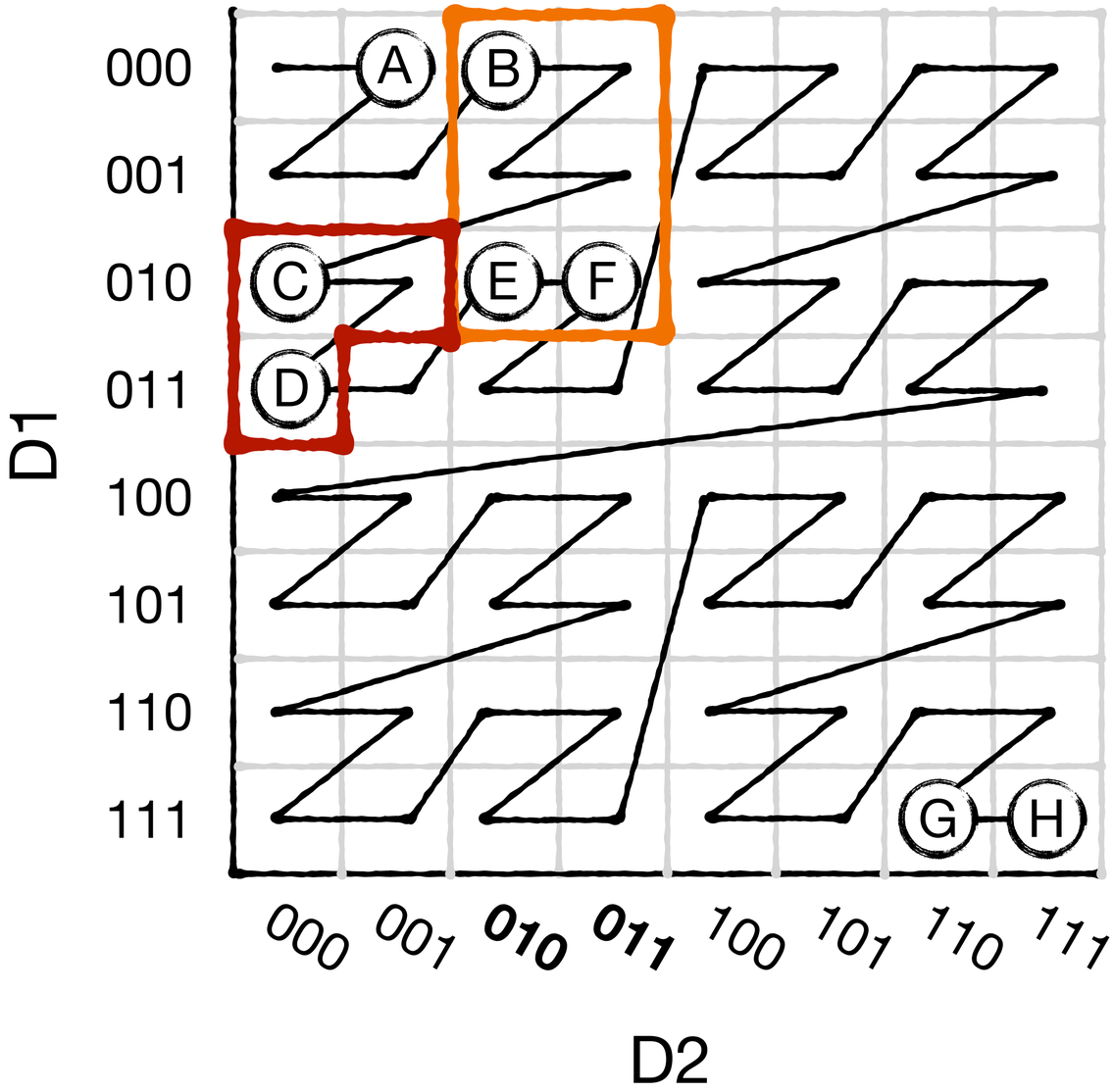
How to skip such nodes to speed up search?

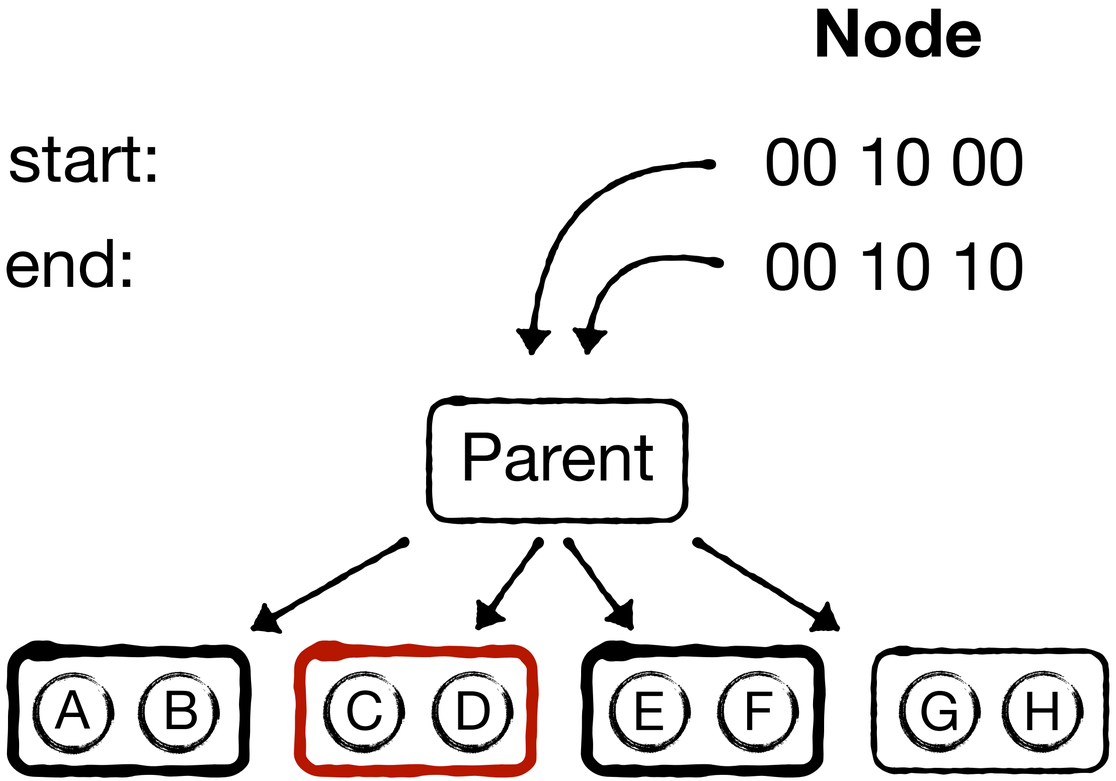


	Query	Node
start:	00 **01** 00	00 **10** 00
end:	00 **11** 01	00 **10** 10

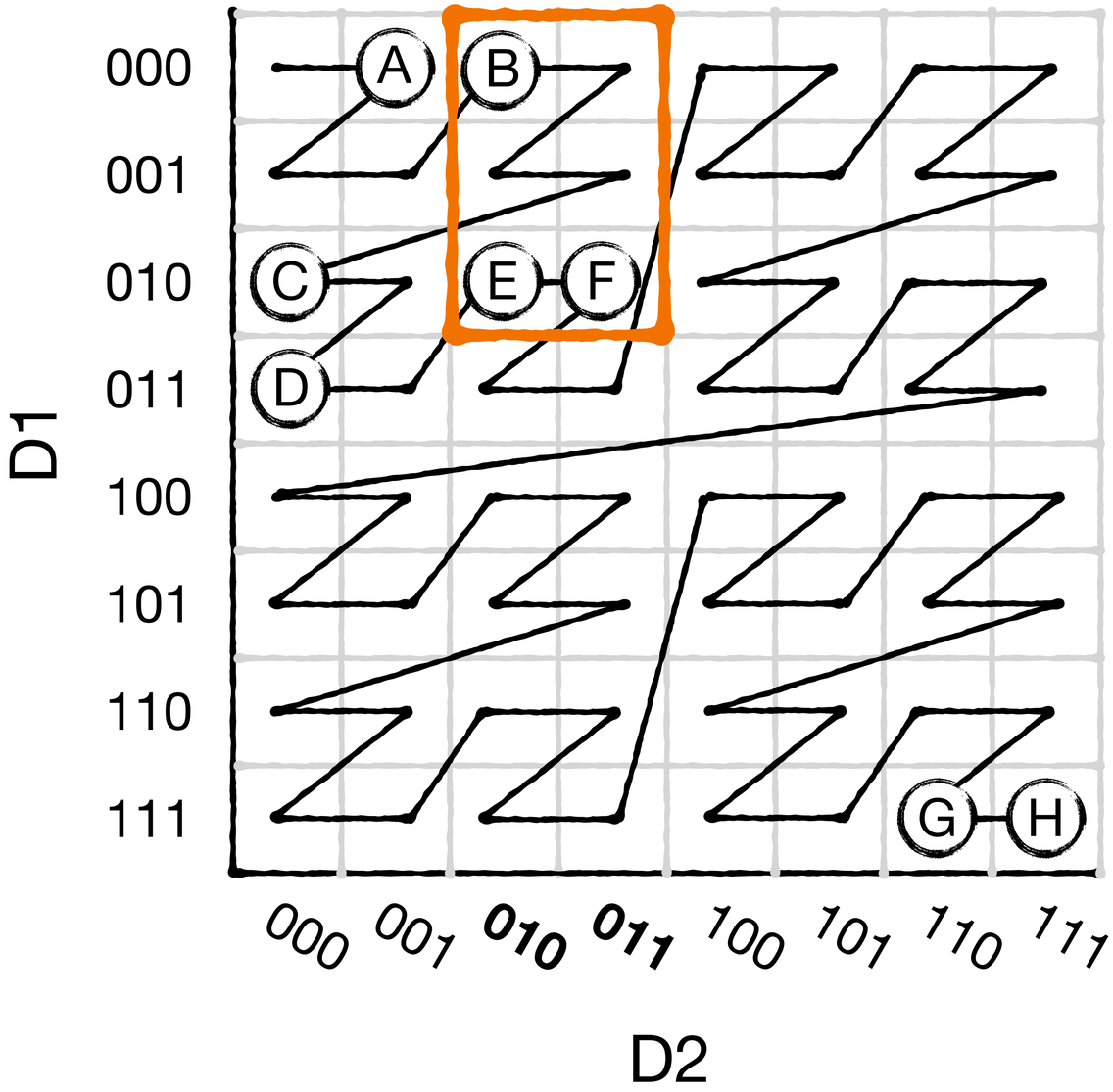


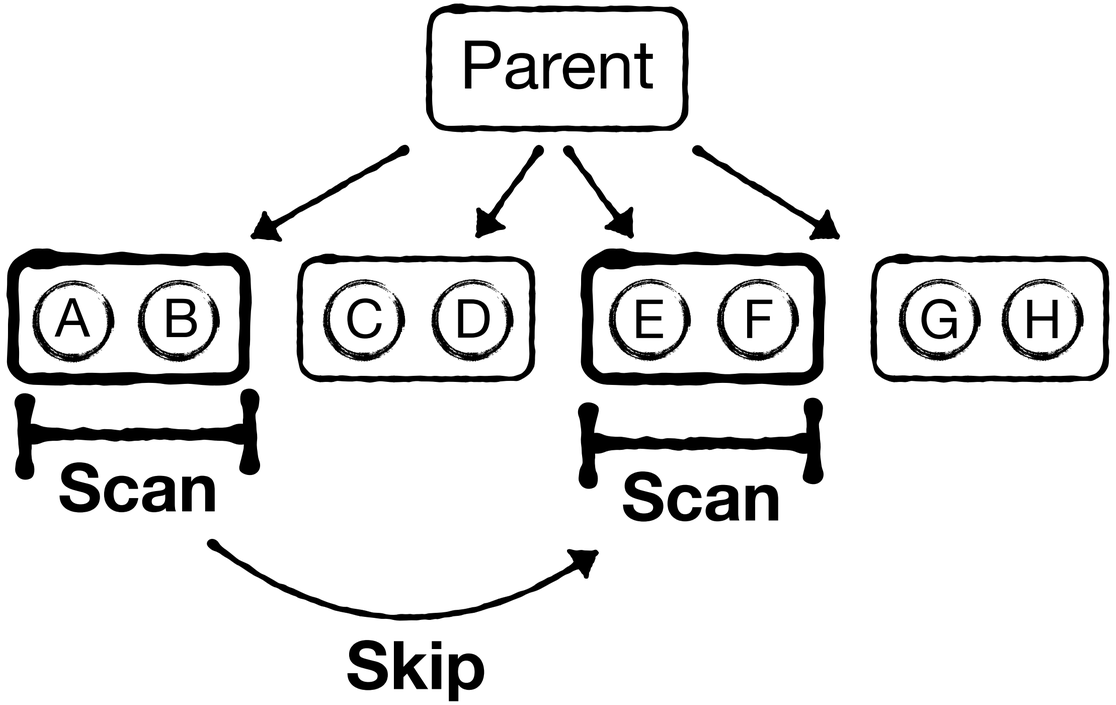
If parent contains start & end of each node, we can infer what to skip



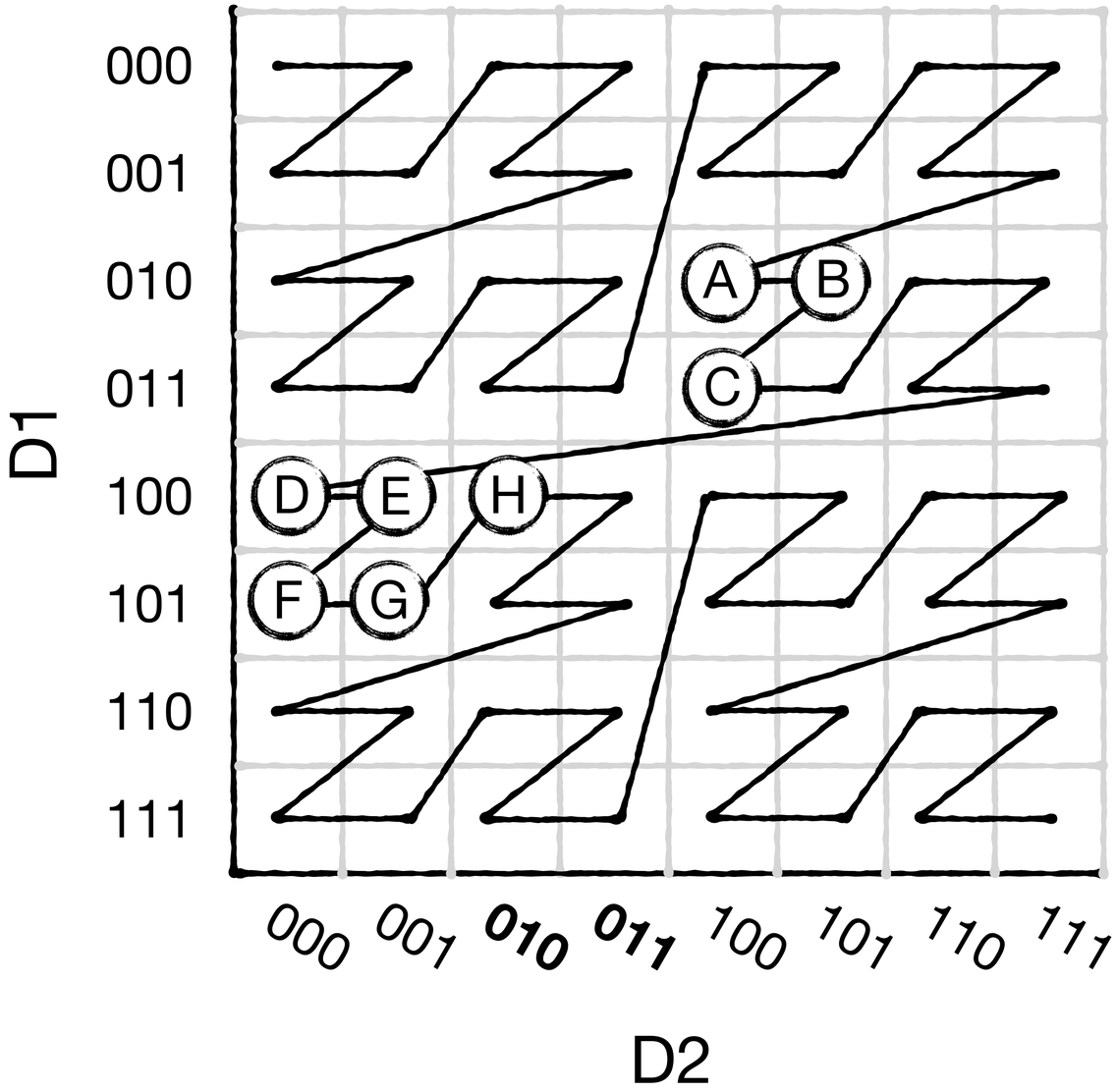


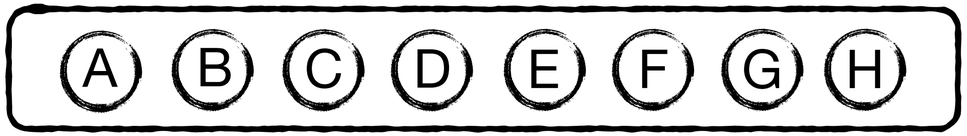
If parent contains start & end of each node, we can infer what to skip



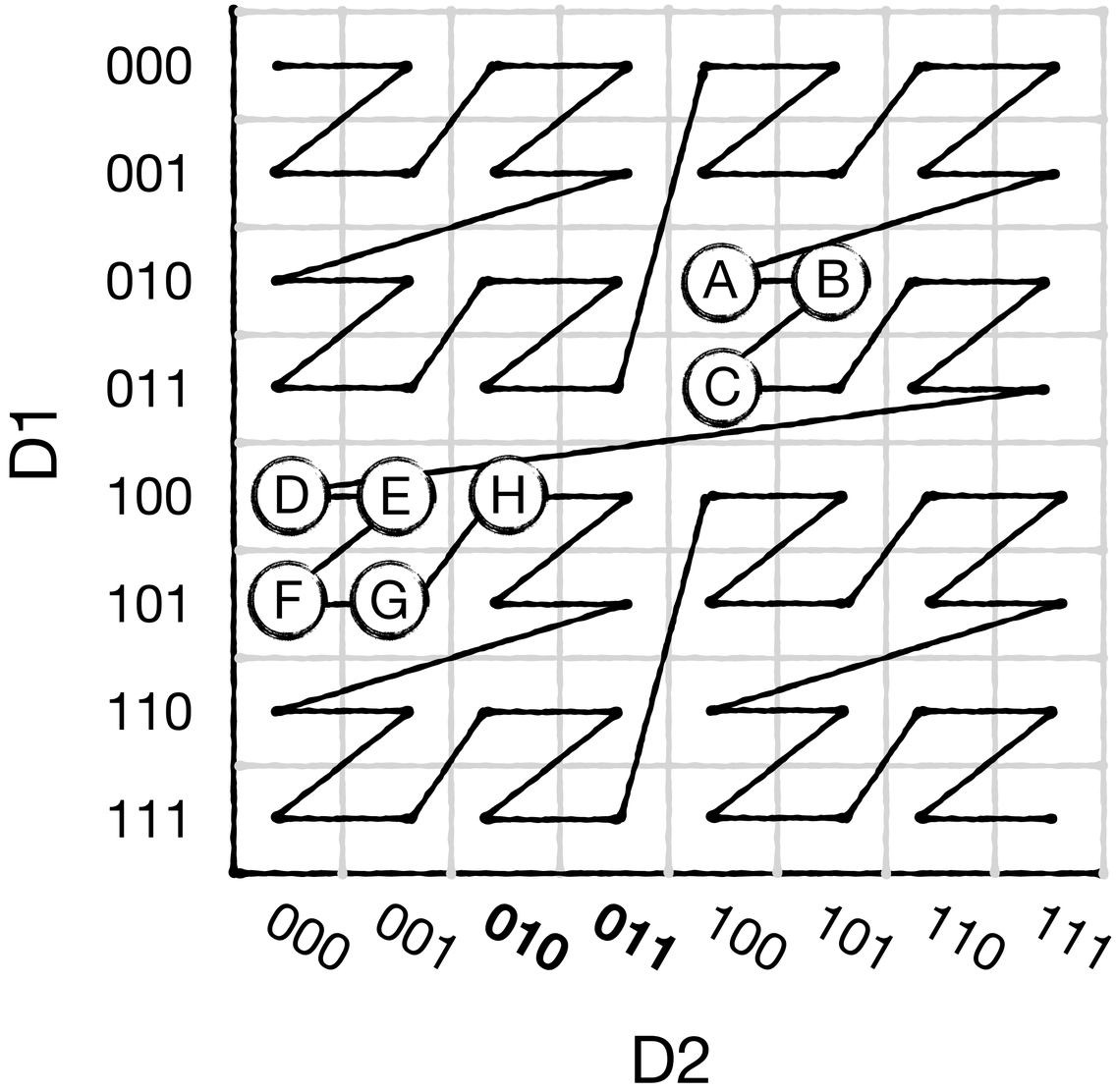


Consider the following node, which is full and about to split

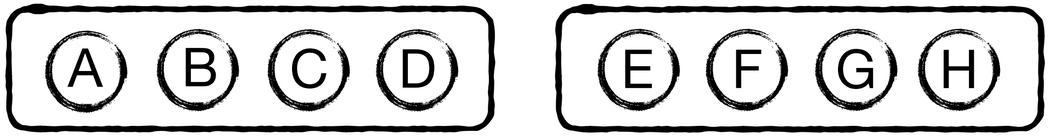




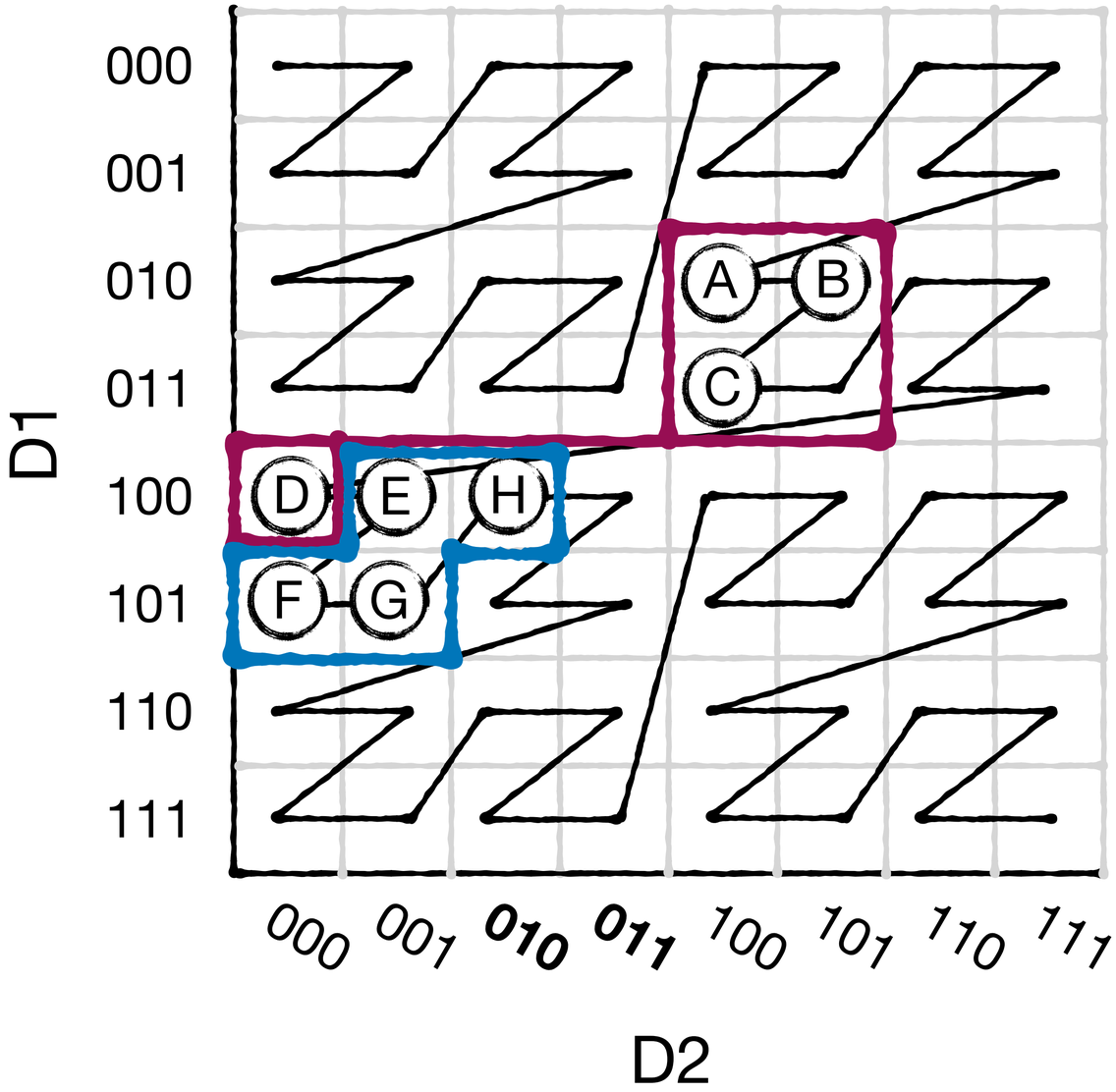
Consider the following node, which is full and about to split



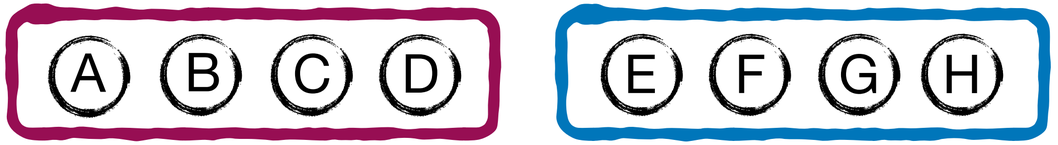
What's wrong with a fifty-fifty split?



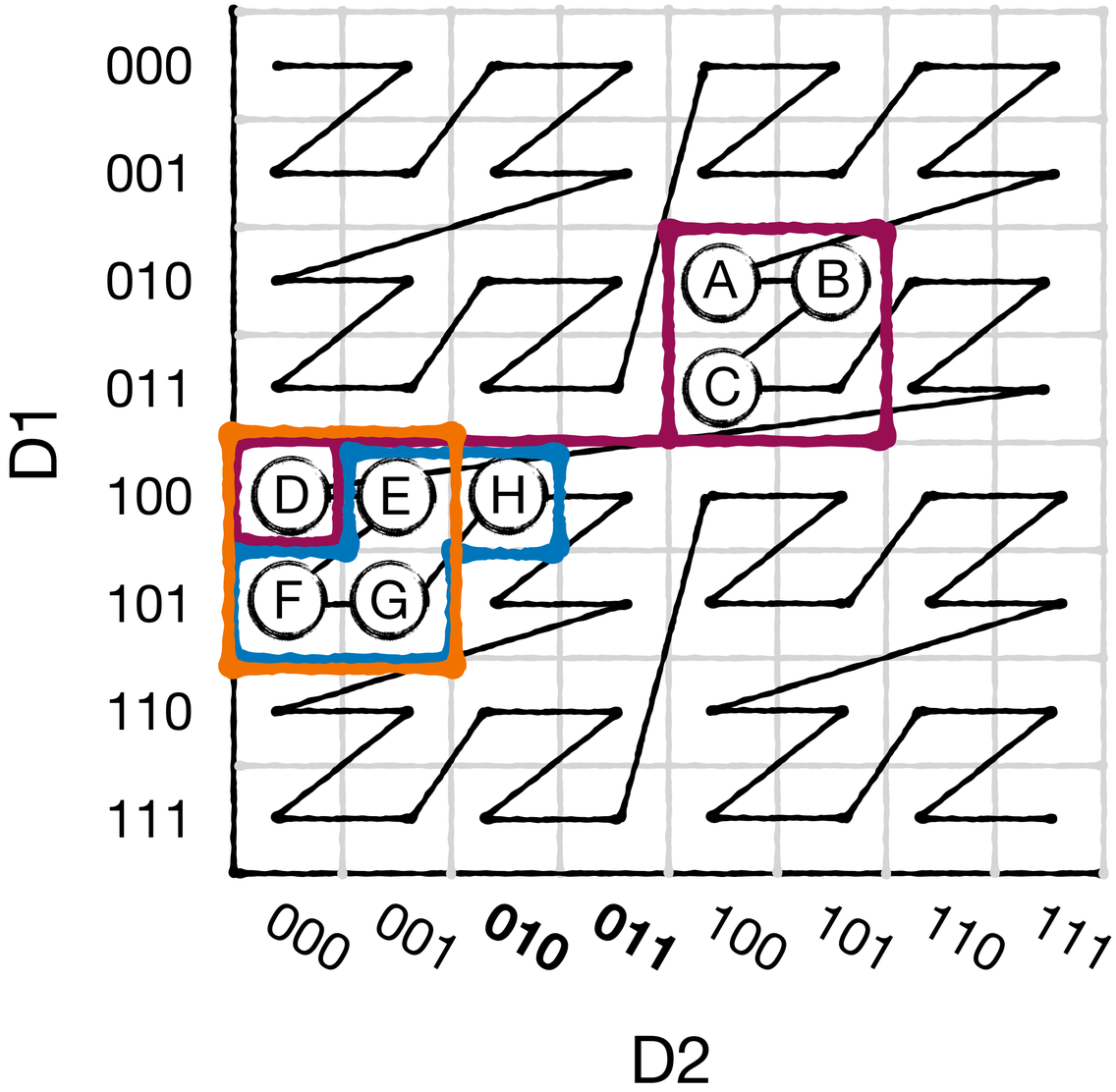
Consider the following node, which is full and about to split



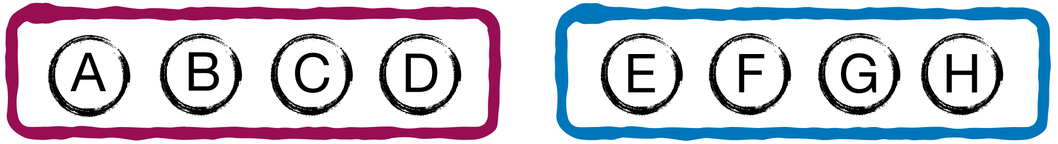
What's wrong with a fifty-fifty split?



Consider the following node, which is full and about to split

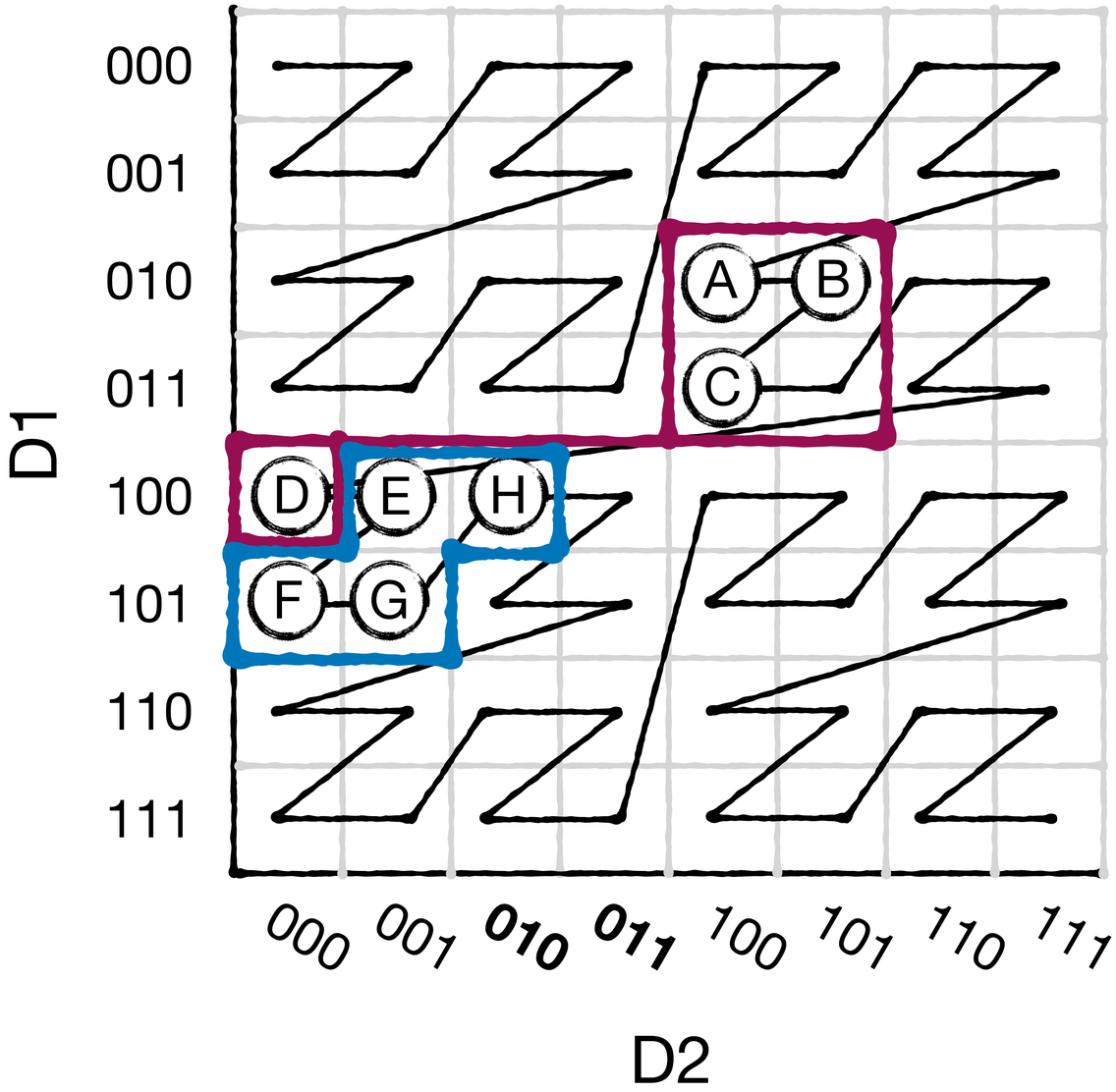


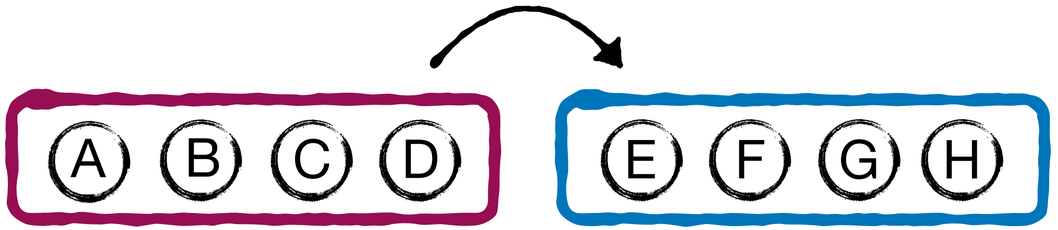
What's wrong with a fifty-fifty split?



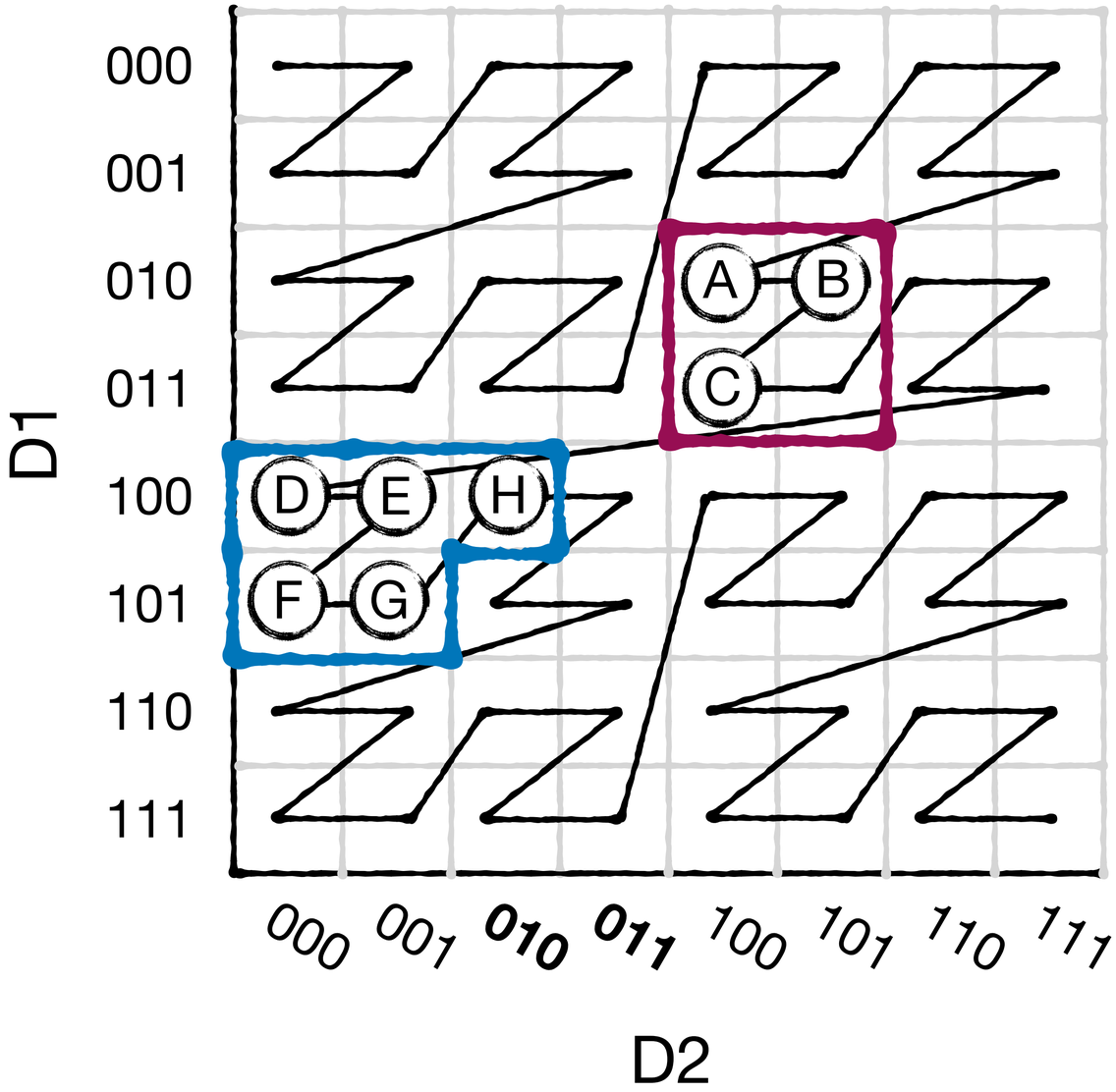
Queries are more likely to intersect with irregular node shapes

Insight: Picking some other close splitting point can greatly alleviate the issue

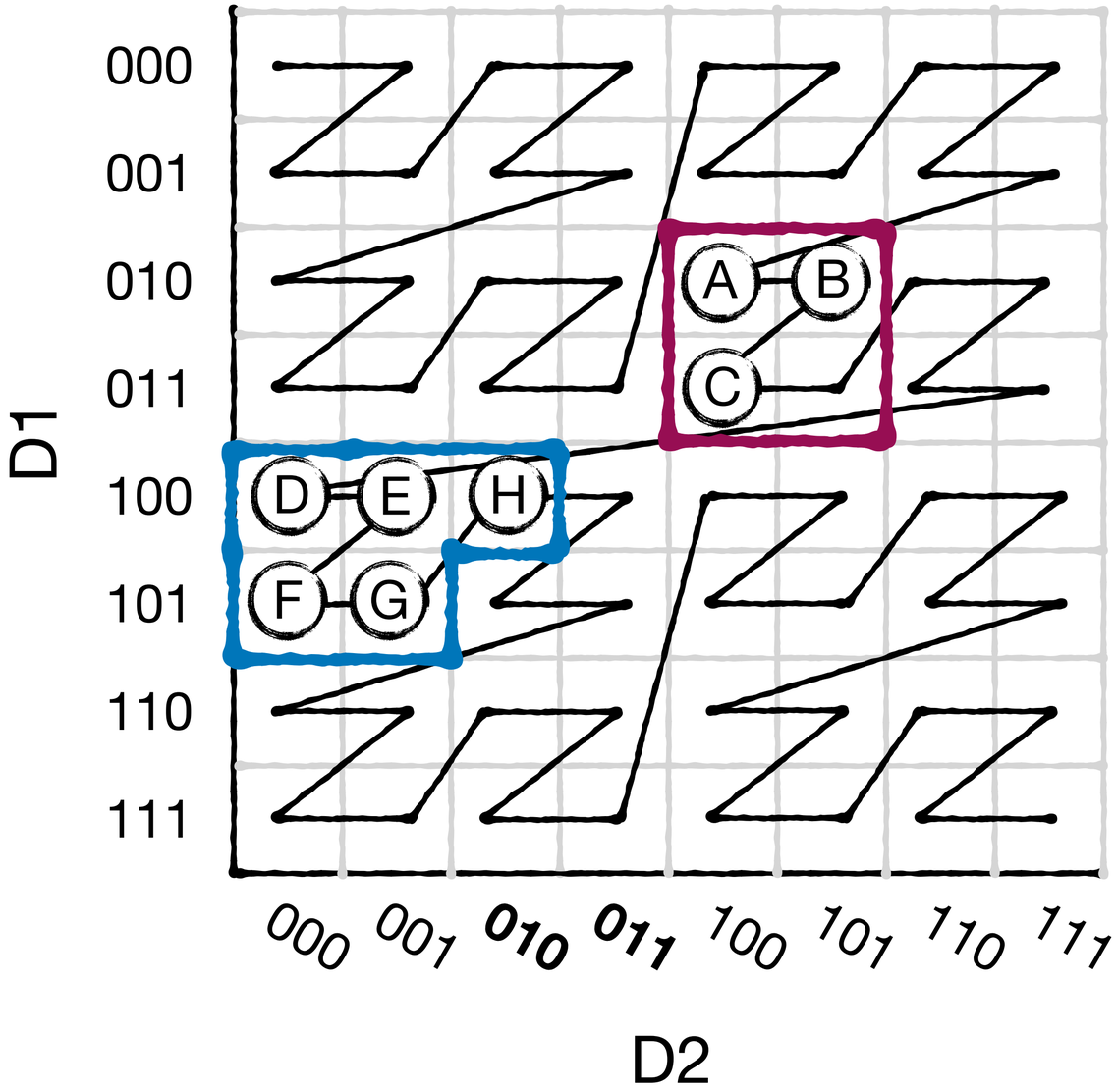




Insight: Picking some other close splitting point can greatly alleviate the issue

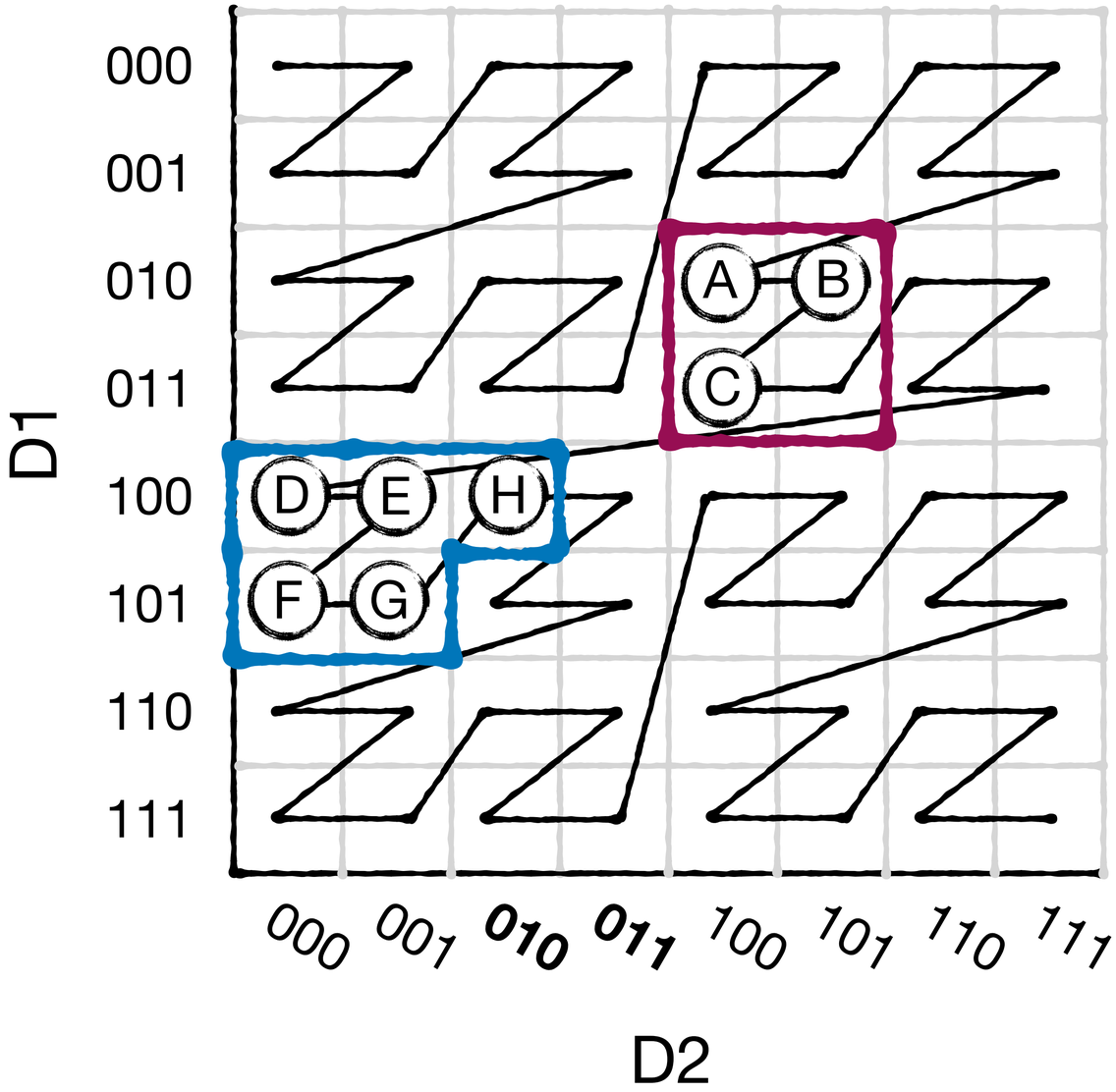


Insight: Picking some other close splitting point can greatly alleviate the issue



What's the trade-off?

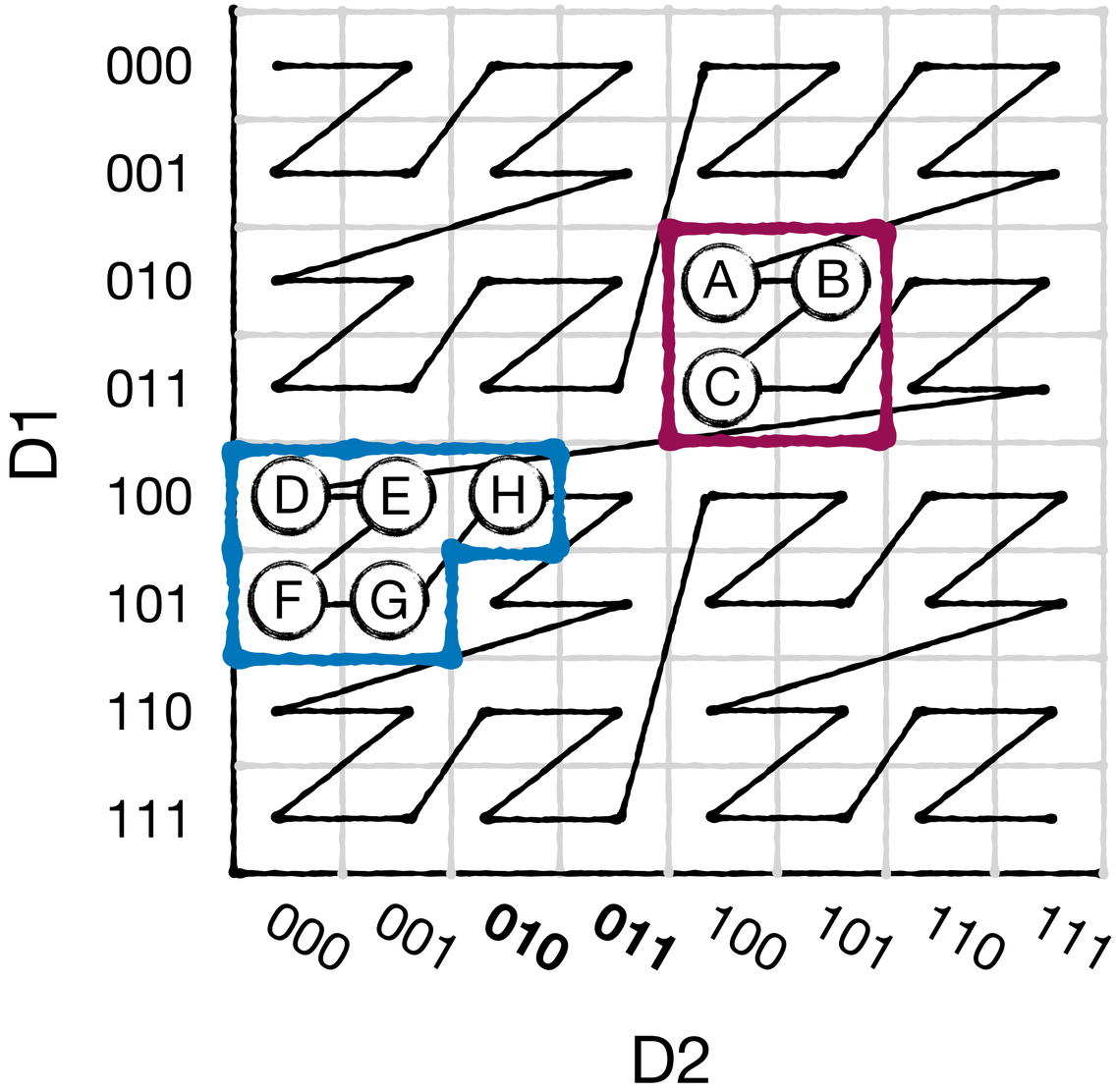
Insight: Picking some other close splitting point can greatly alleviate the issue



What's the trade-off?

Some nodes are less utilized

Insight: Picking some other close splitting point can greatly alleviate the issue

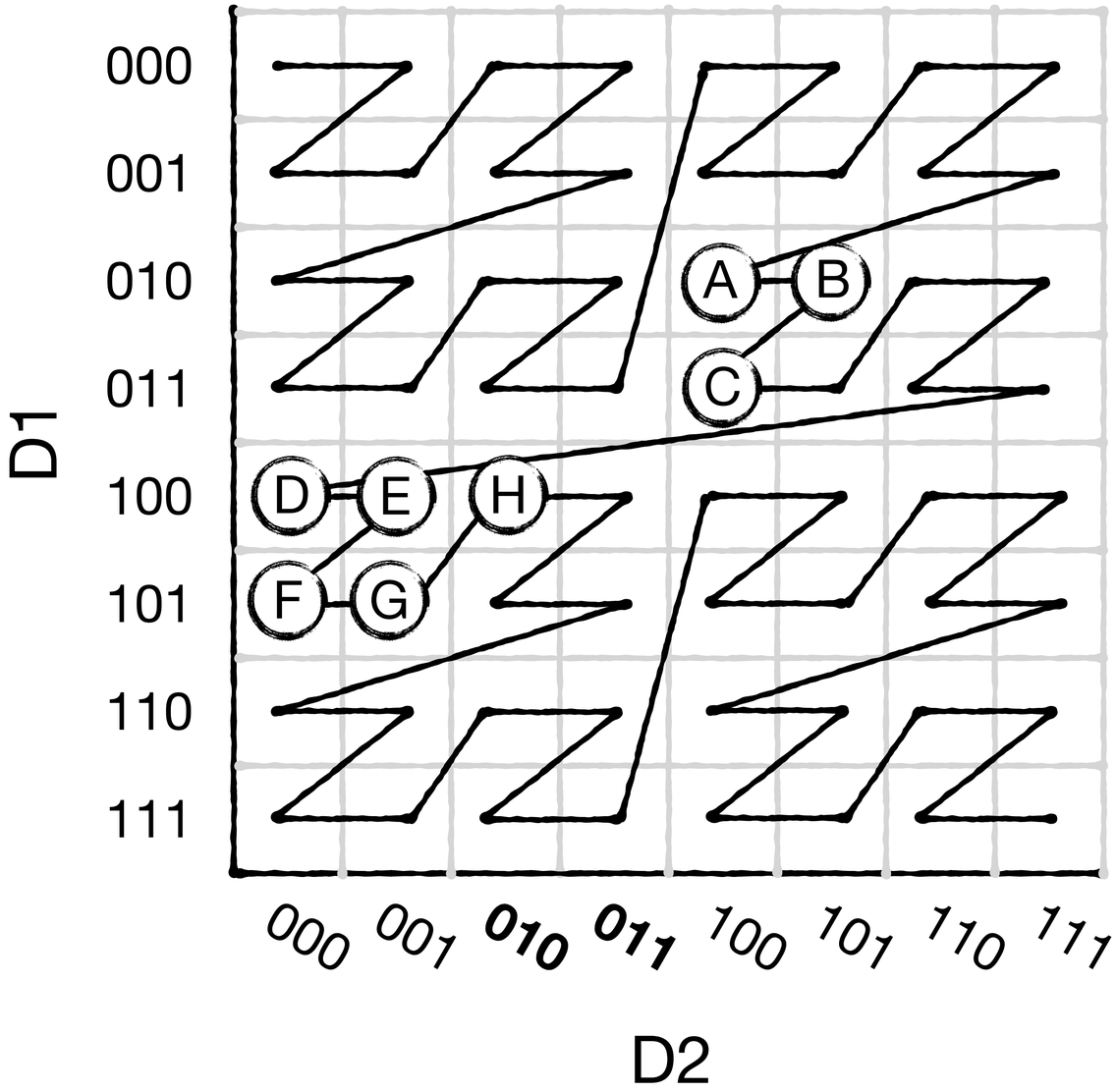


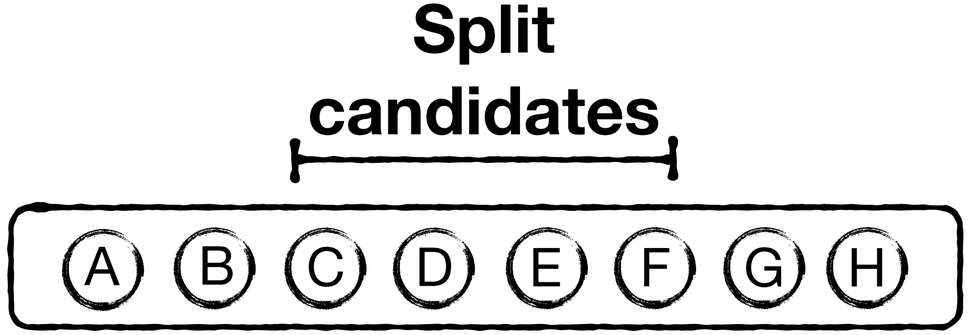
What's the trade-off?

Some nodes are less utilized

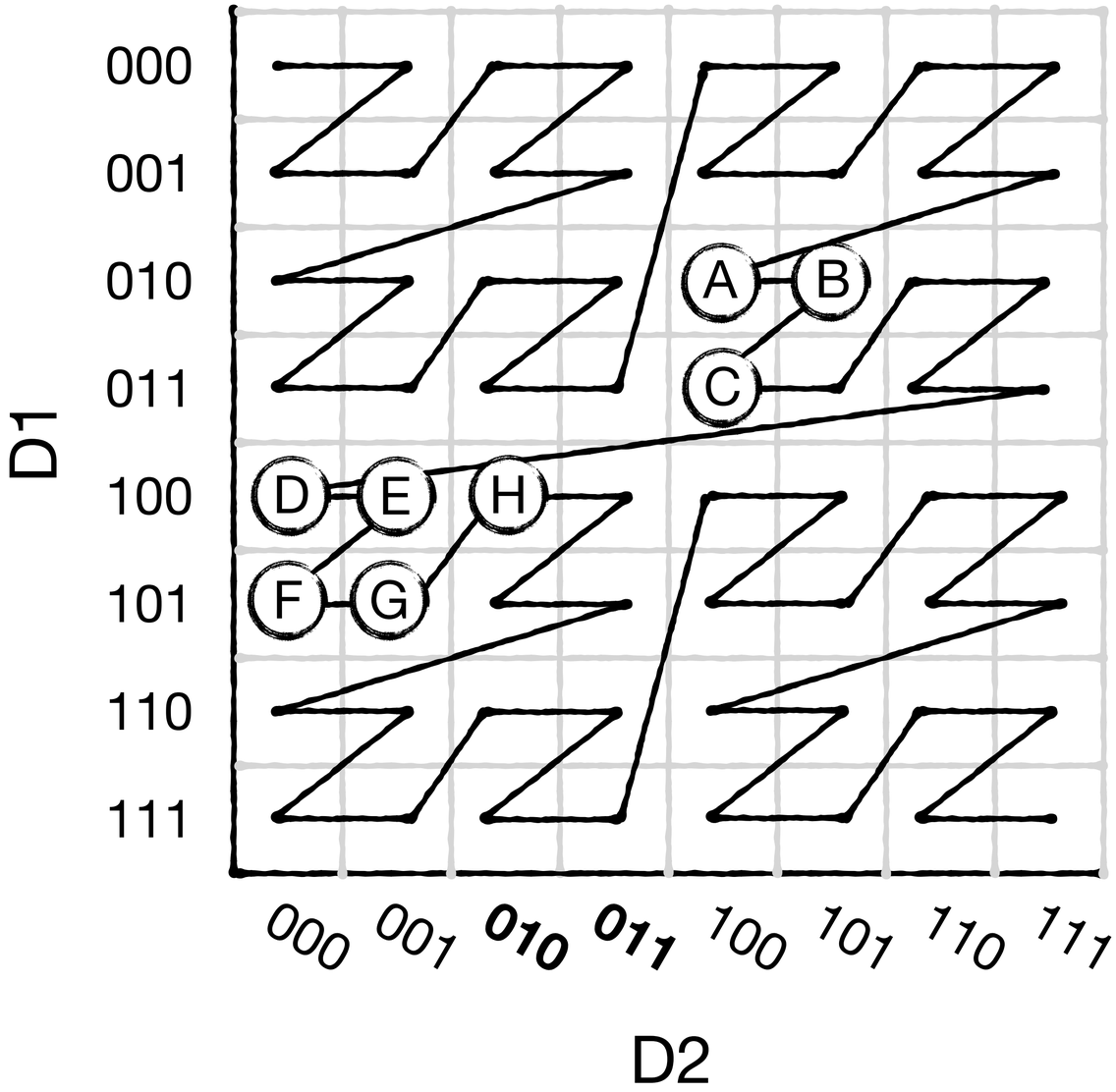
Can lead to deeper tree with more space wastage

Balance by picking best split from a middle nodes

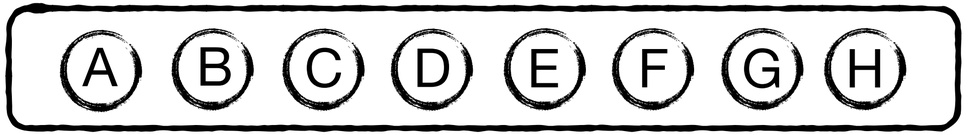




Balance by picking best split from a middle nodes



Split candidates



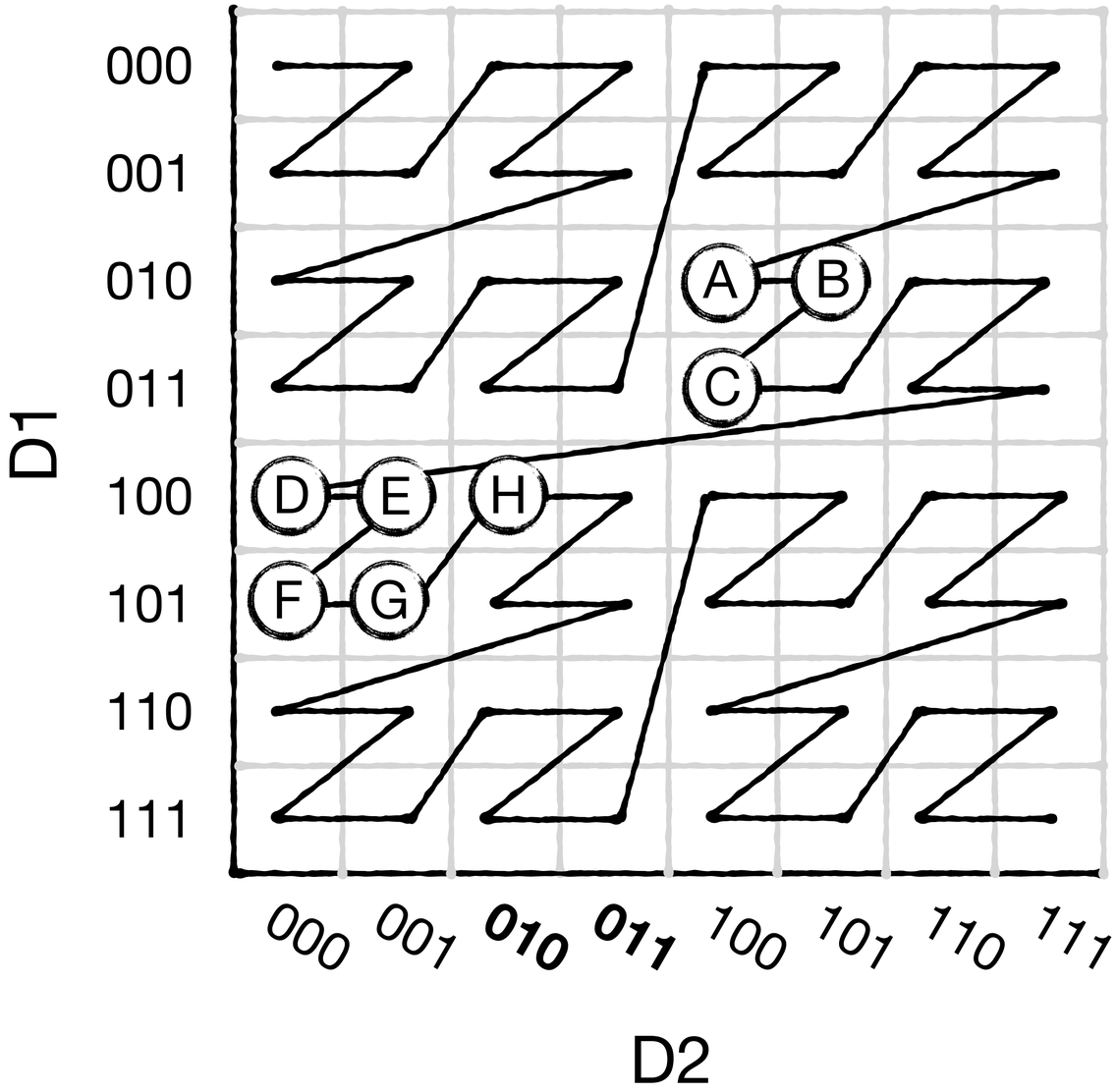
C 01 00 10

D 10 00 00

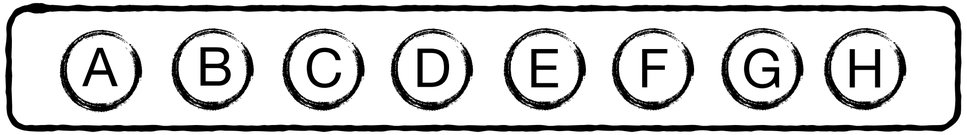
E 10 00 01

F 10 00 10

Balance by picking best split from a middle nodes



Split candidates

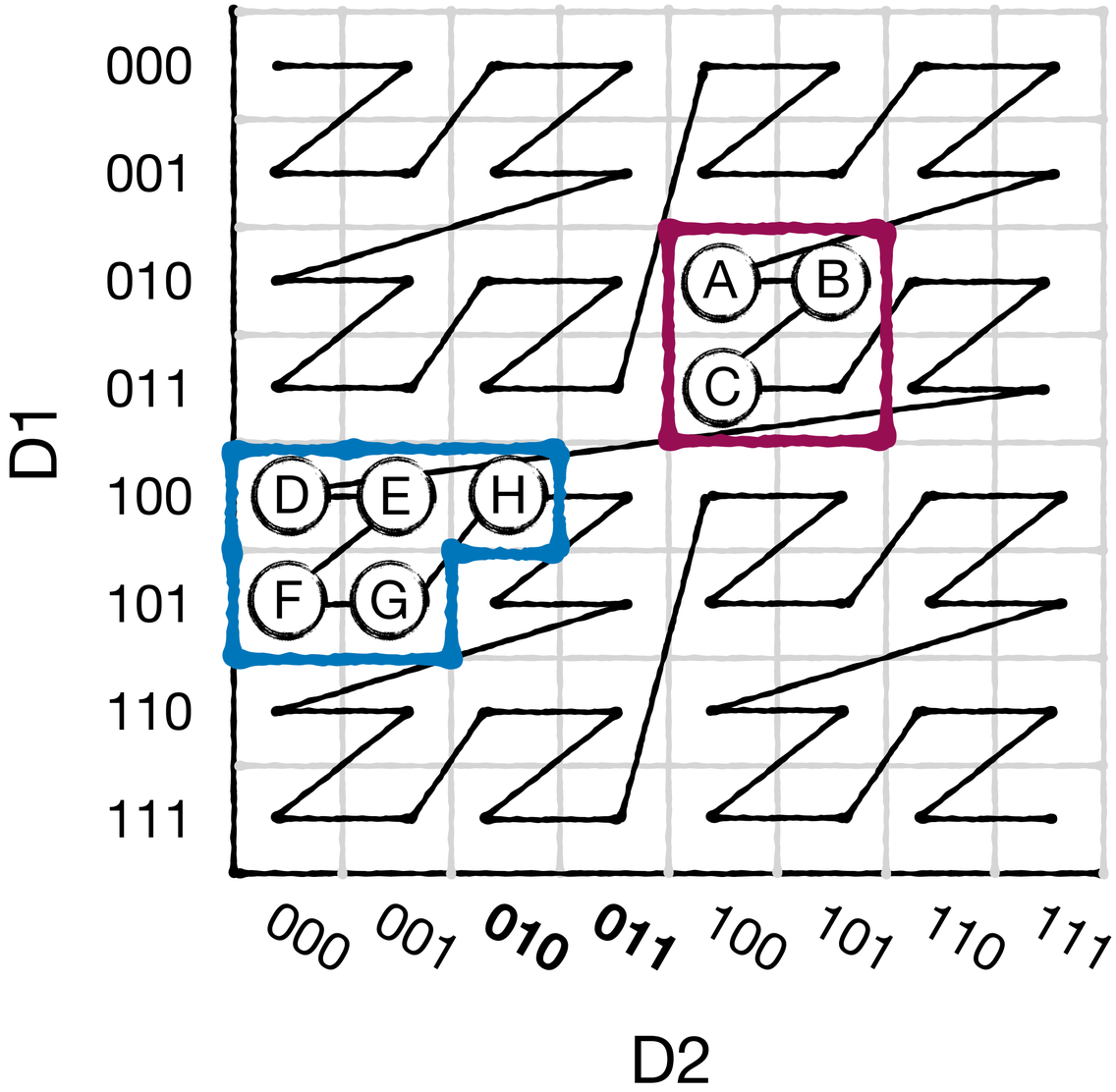


C 01 00 10

D **10 00 00** ← **Most trailing zeros, best splitter**

E 10 00 01

F 10 00 10

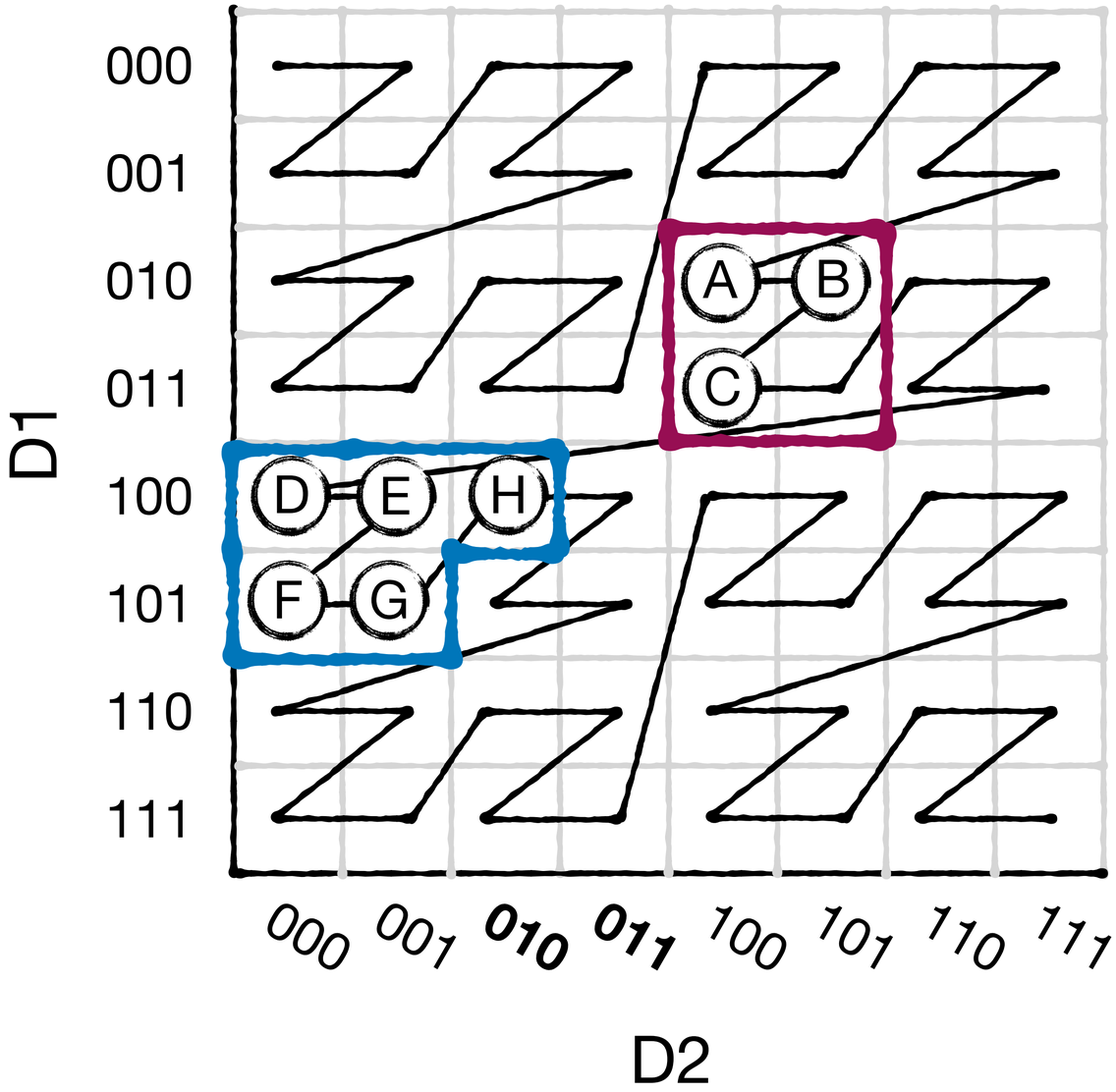


C 01 00 10

D **10 00 00** ← **Most trailing zeros, best splitter**

E 10 00 01

F 10 00 10



Balances goals of having rectangular nodes against having shallow and space-efficient tree

Thank you