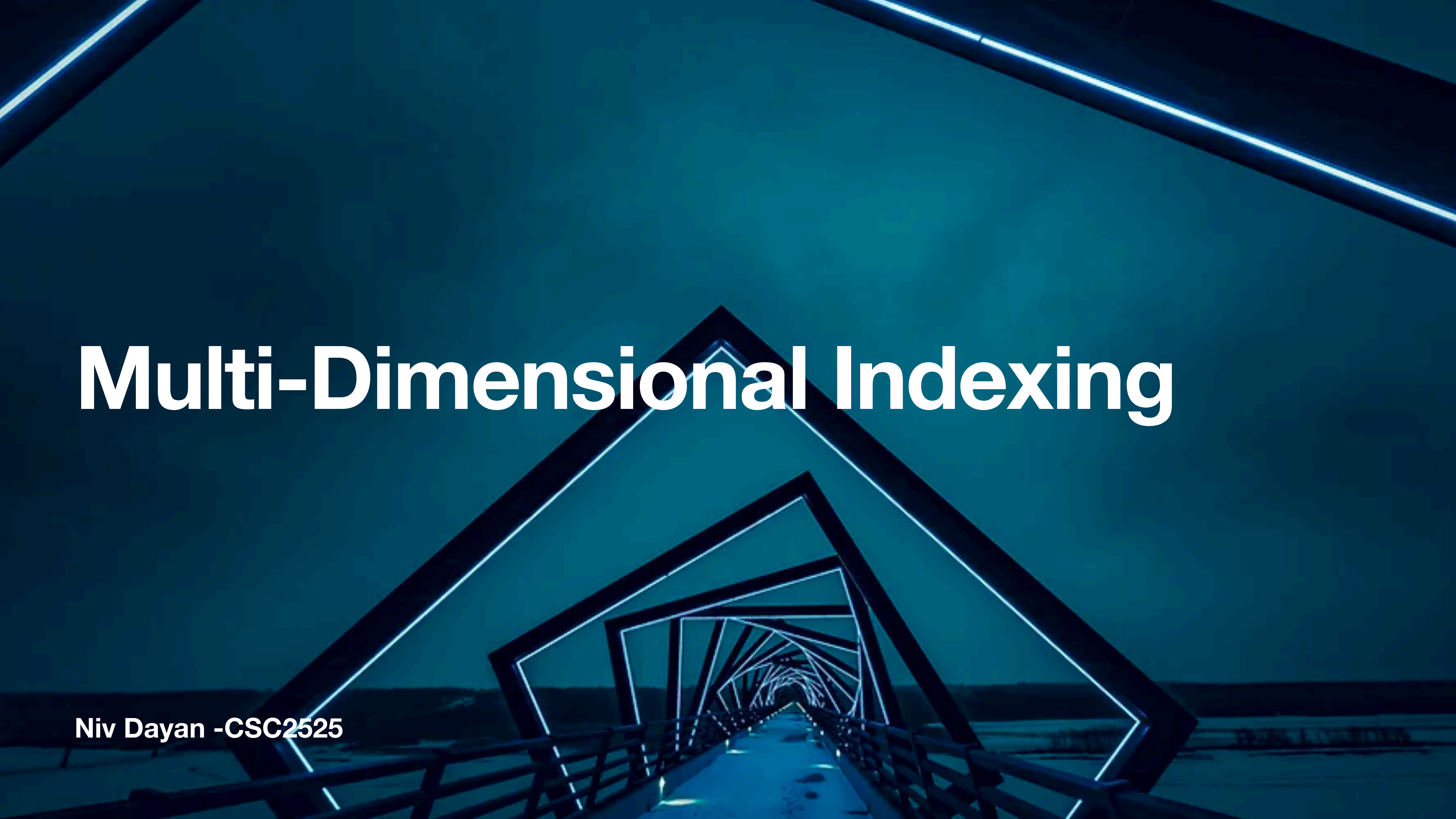
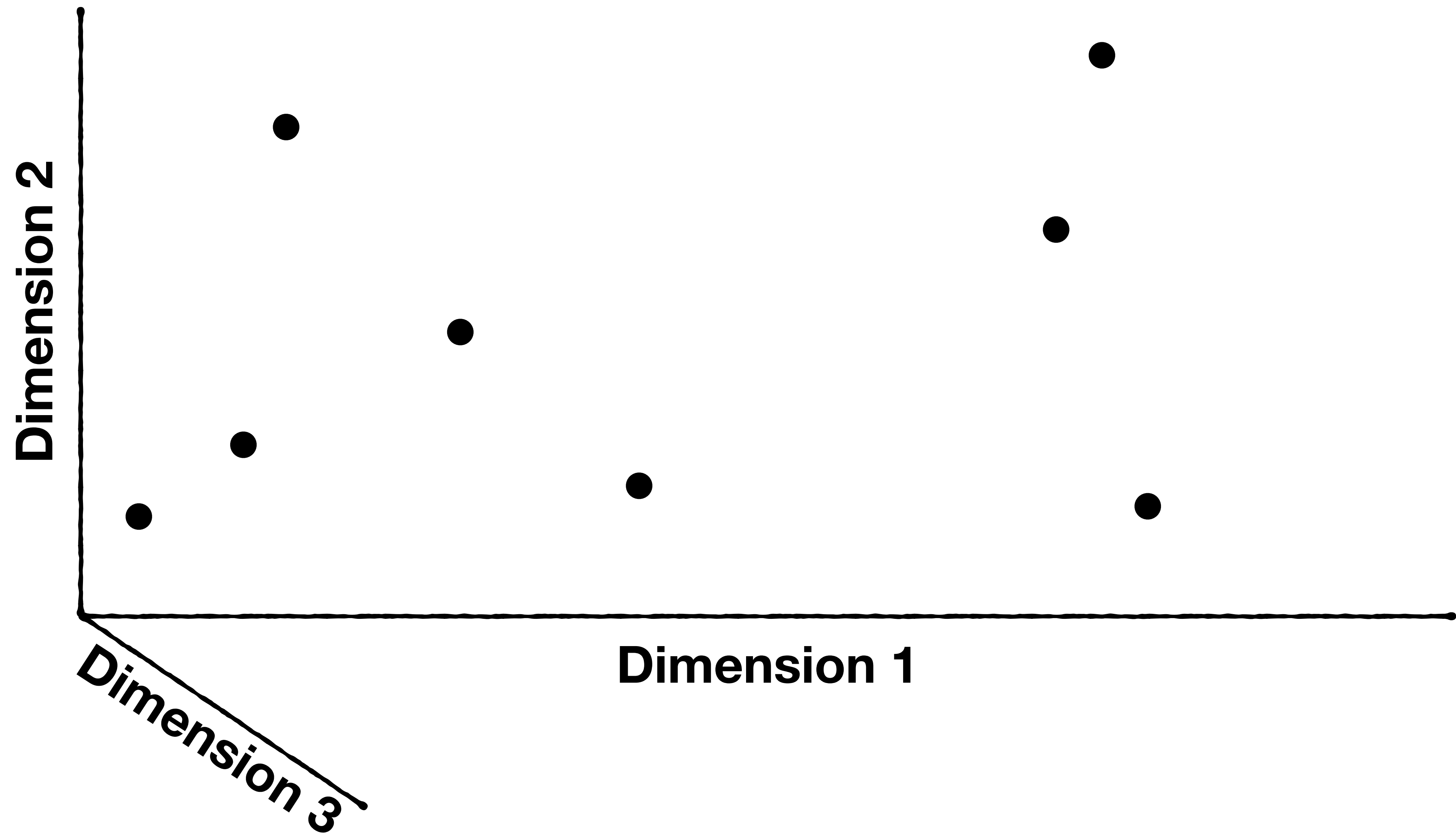


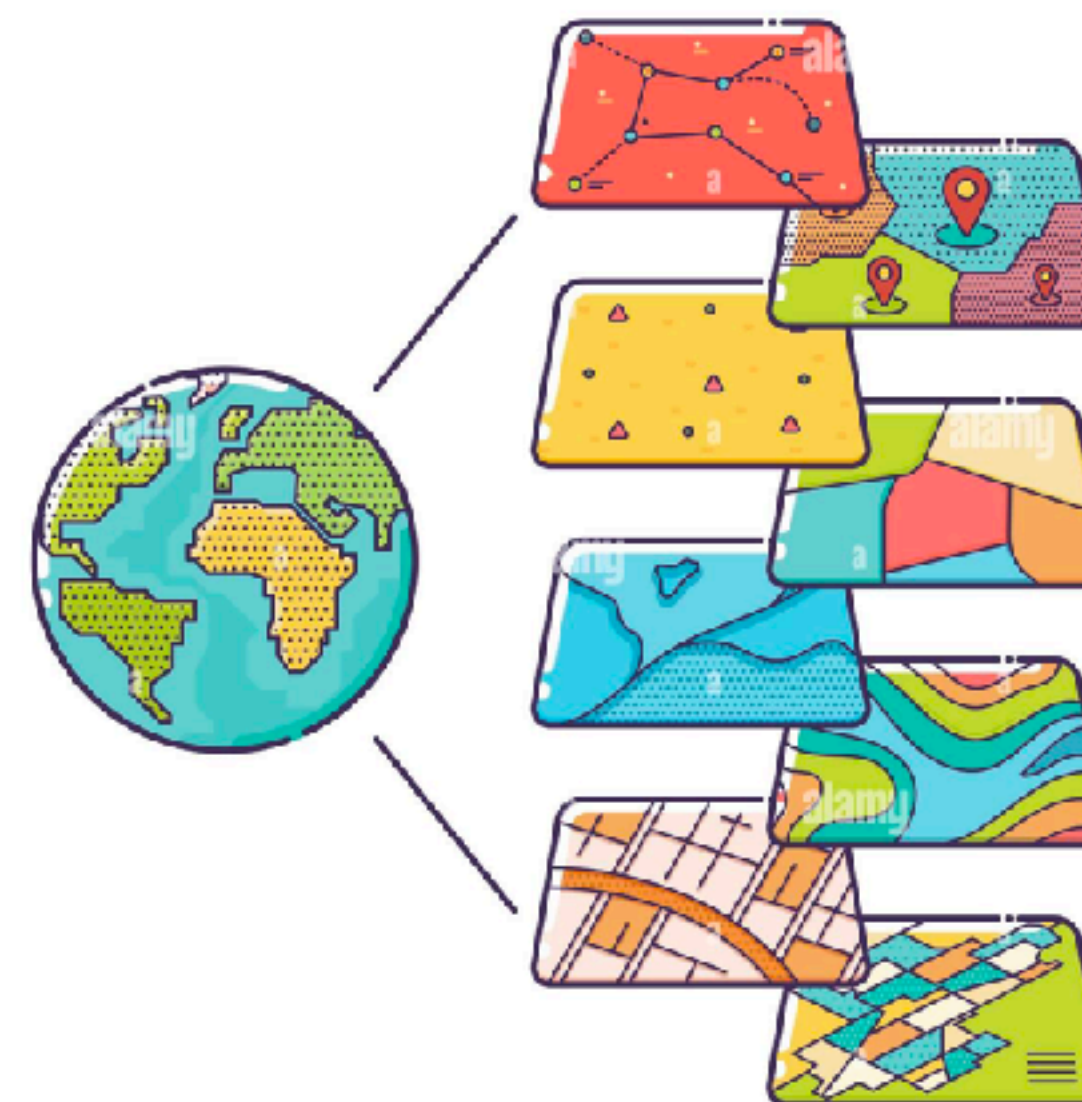
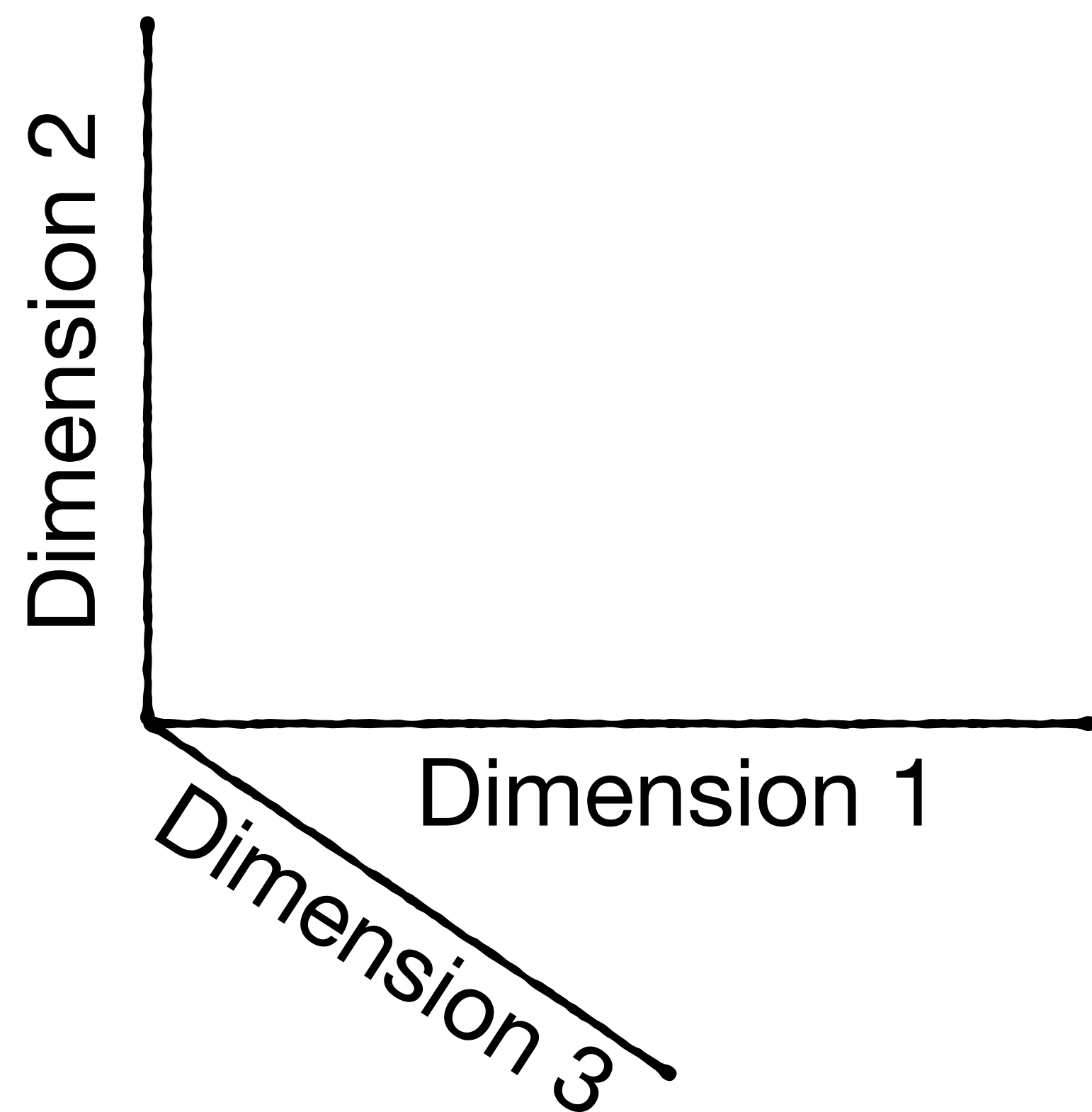
Multi-Dimensional Indexing

The background of the slide is a blue-toned photograph of a long, covered walkway or bridge. The structure is composed of a series of repeating triangular roof frames, creating a strong perspective effect that draws the eye towards the horizon. The walkway has railings on both sides and is illuminated by a cool blue light. The sky is a deep, uniform blue.

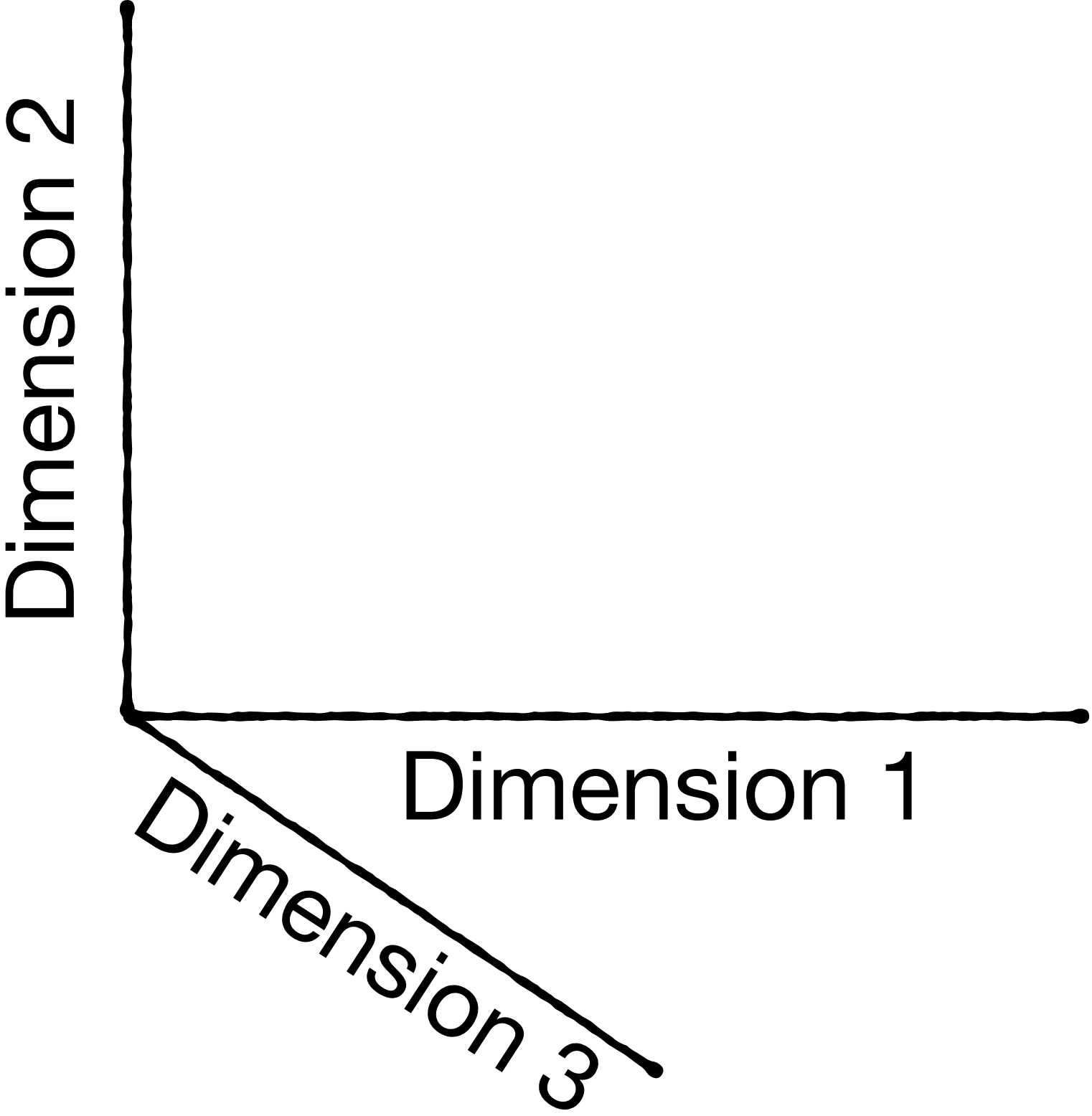
Niv Dayan -CSC2525



Spatial Data

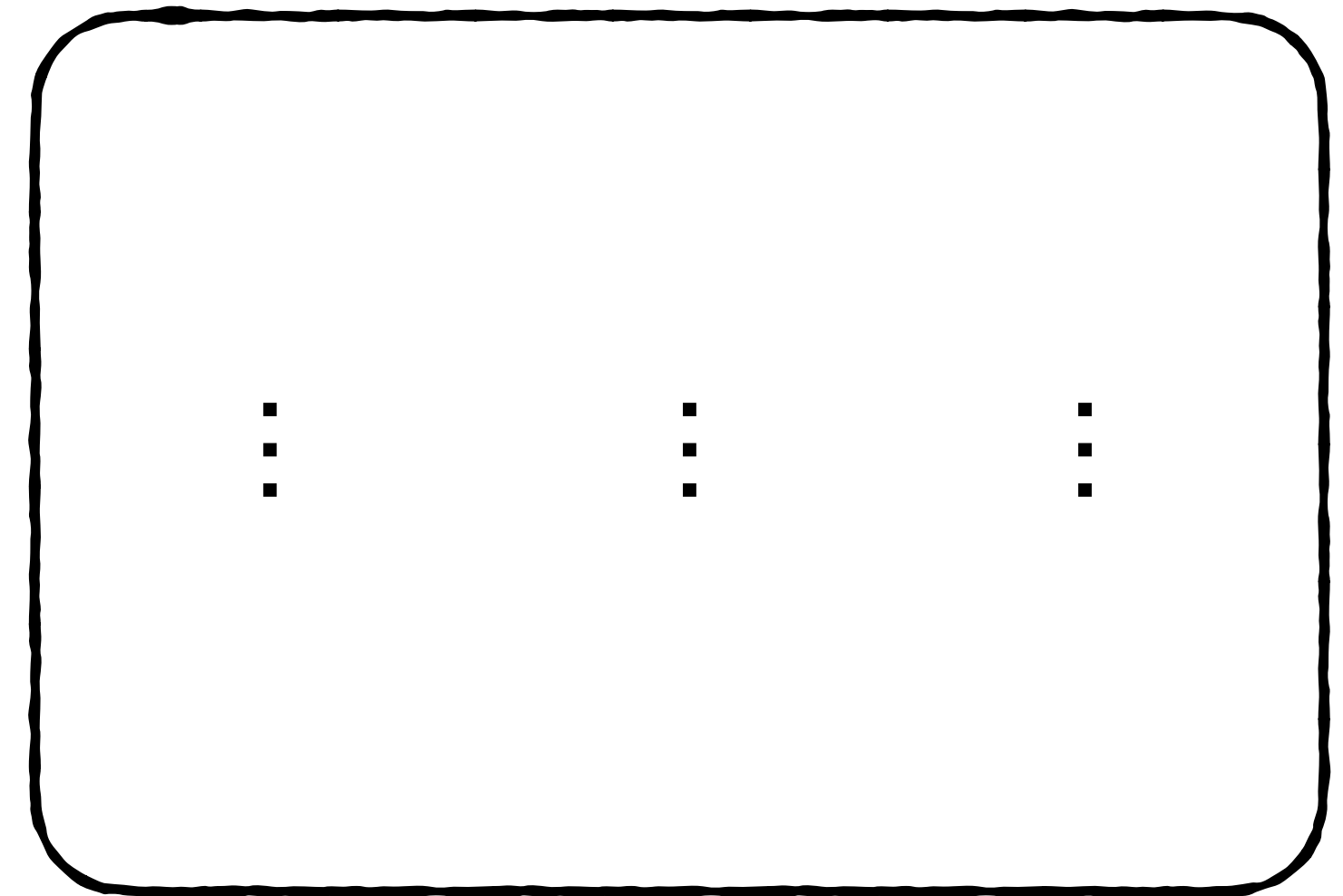
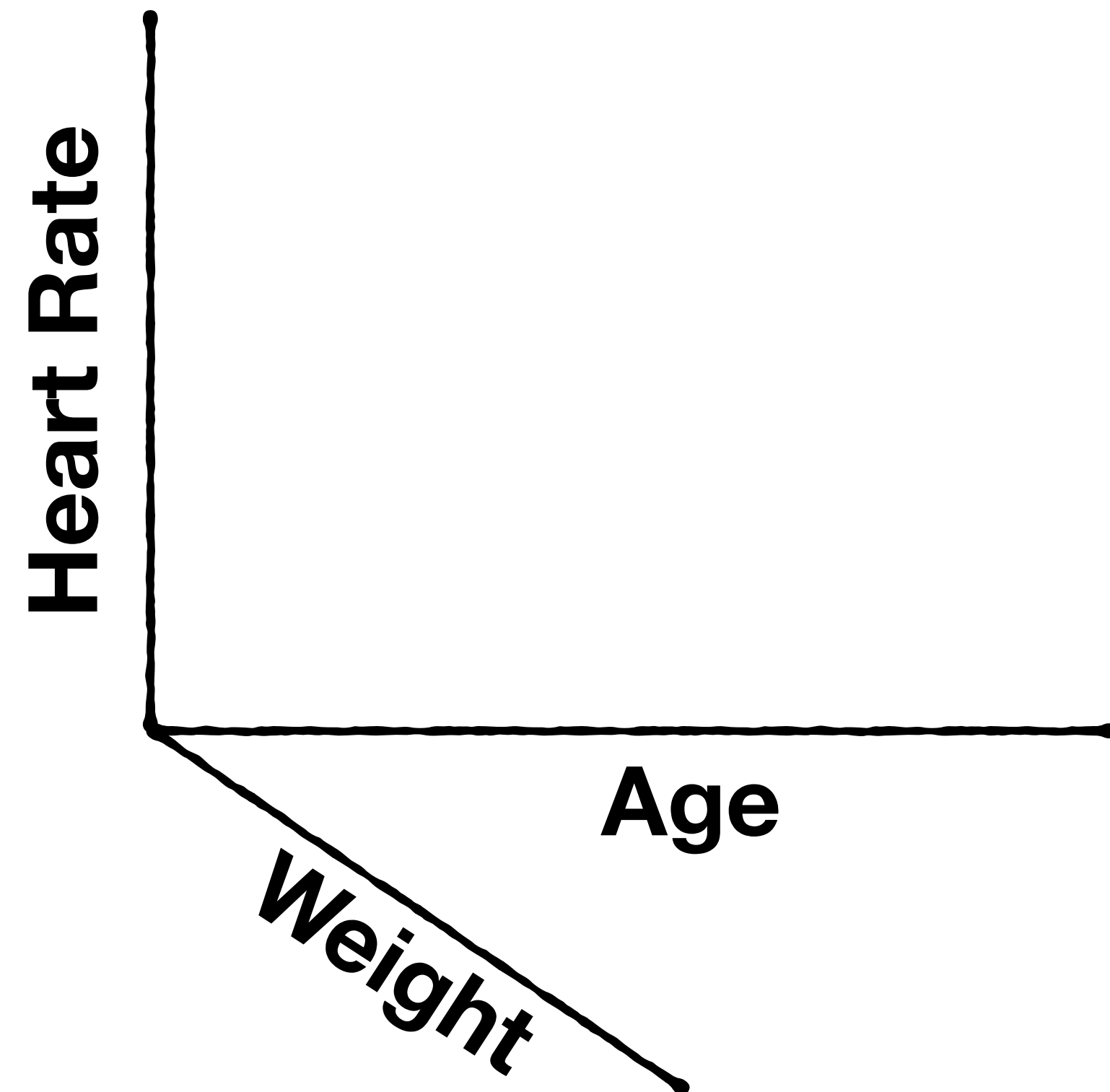


Tables in relational DB

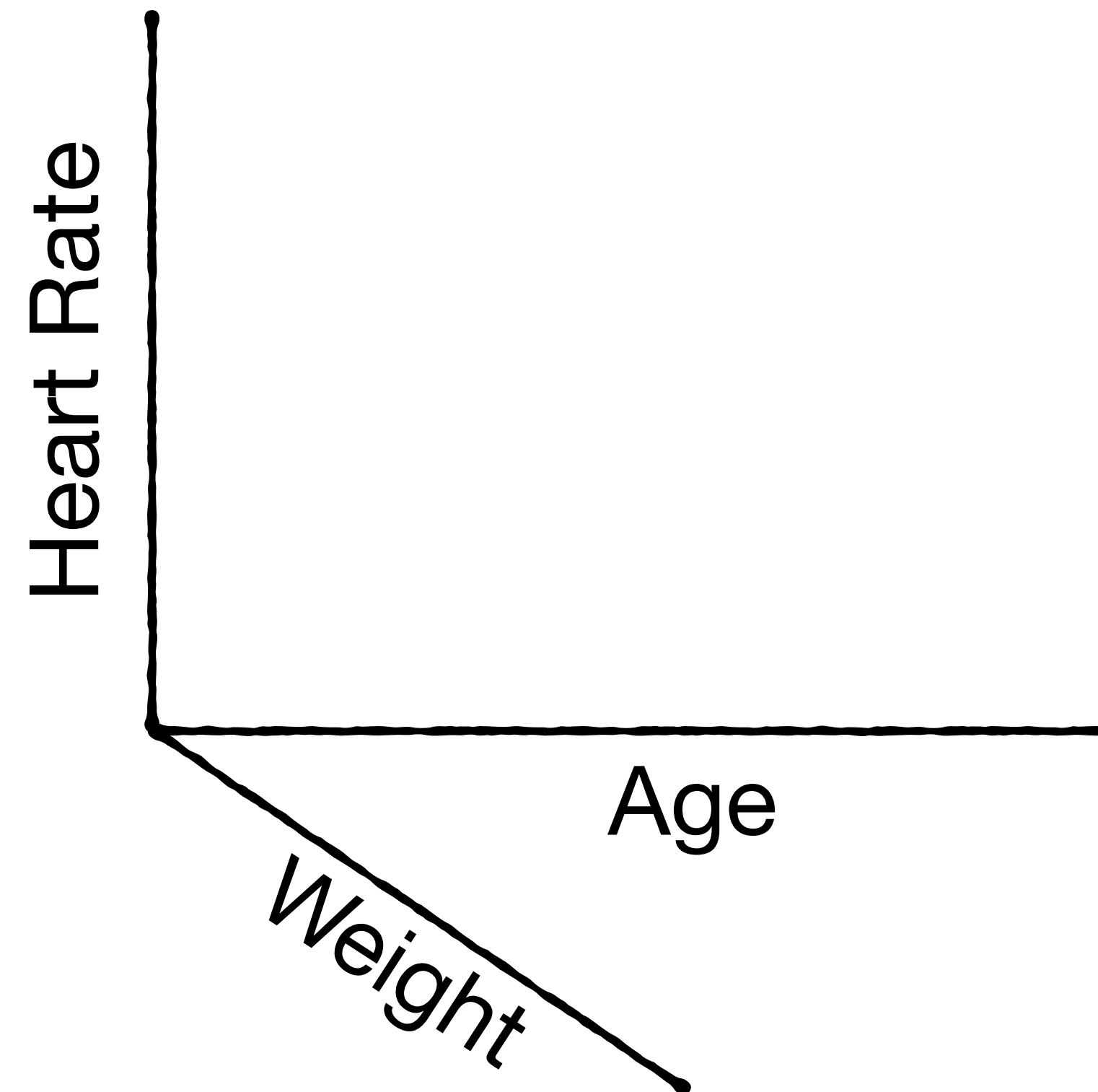


Heart Rate	Age	Weight
⋮	⋮	⋮

Tables in relational DB



How to index multi-dimensional data?



KD-Trees
1975

R-Trees
1984

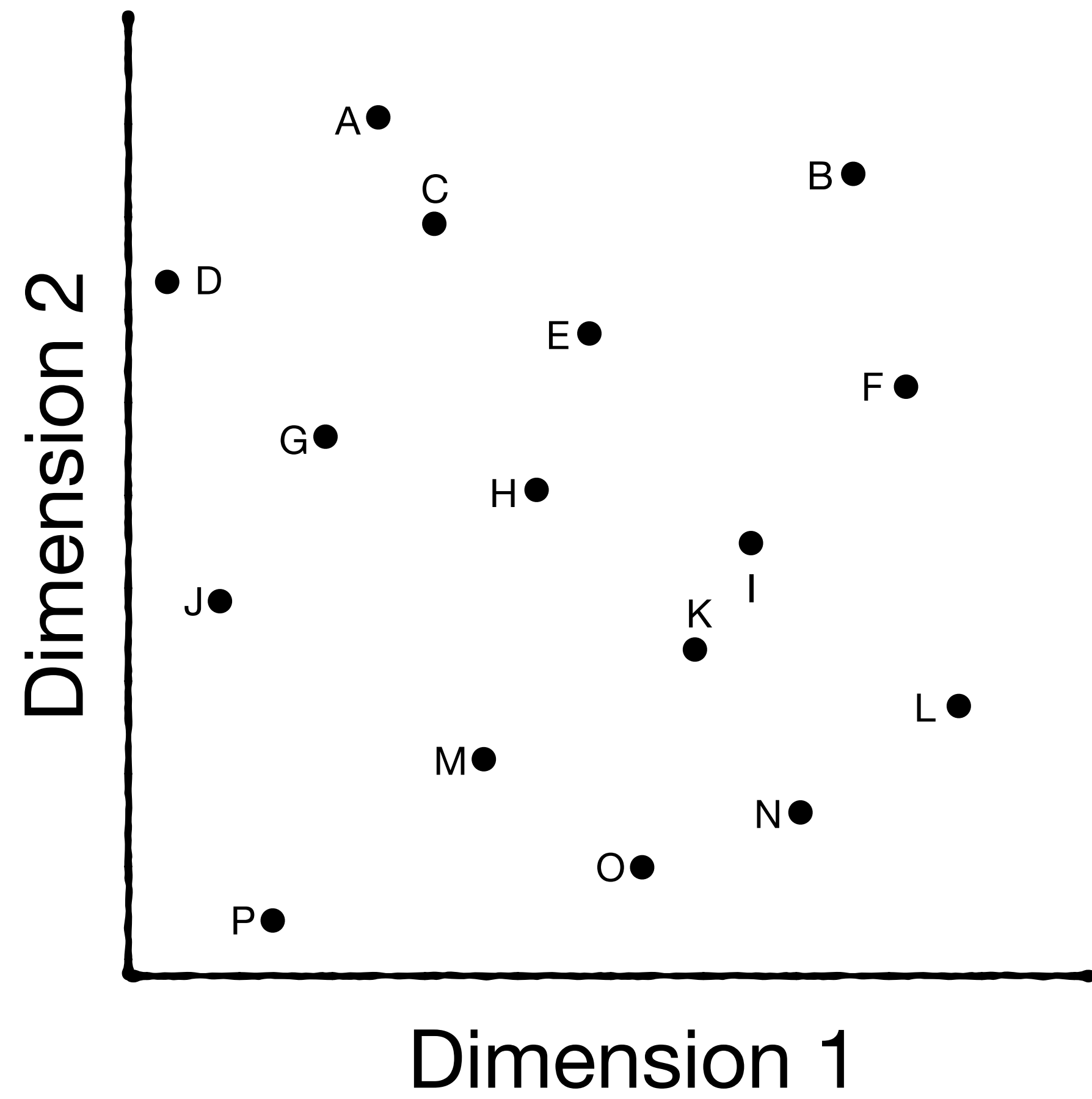
R+tree
1987

R*-tree
1990

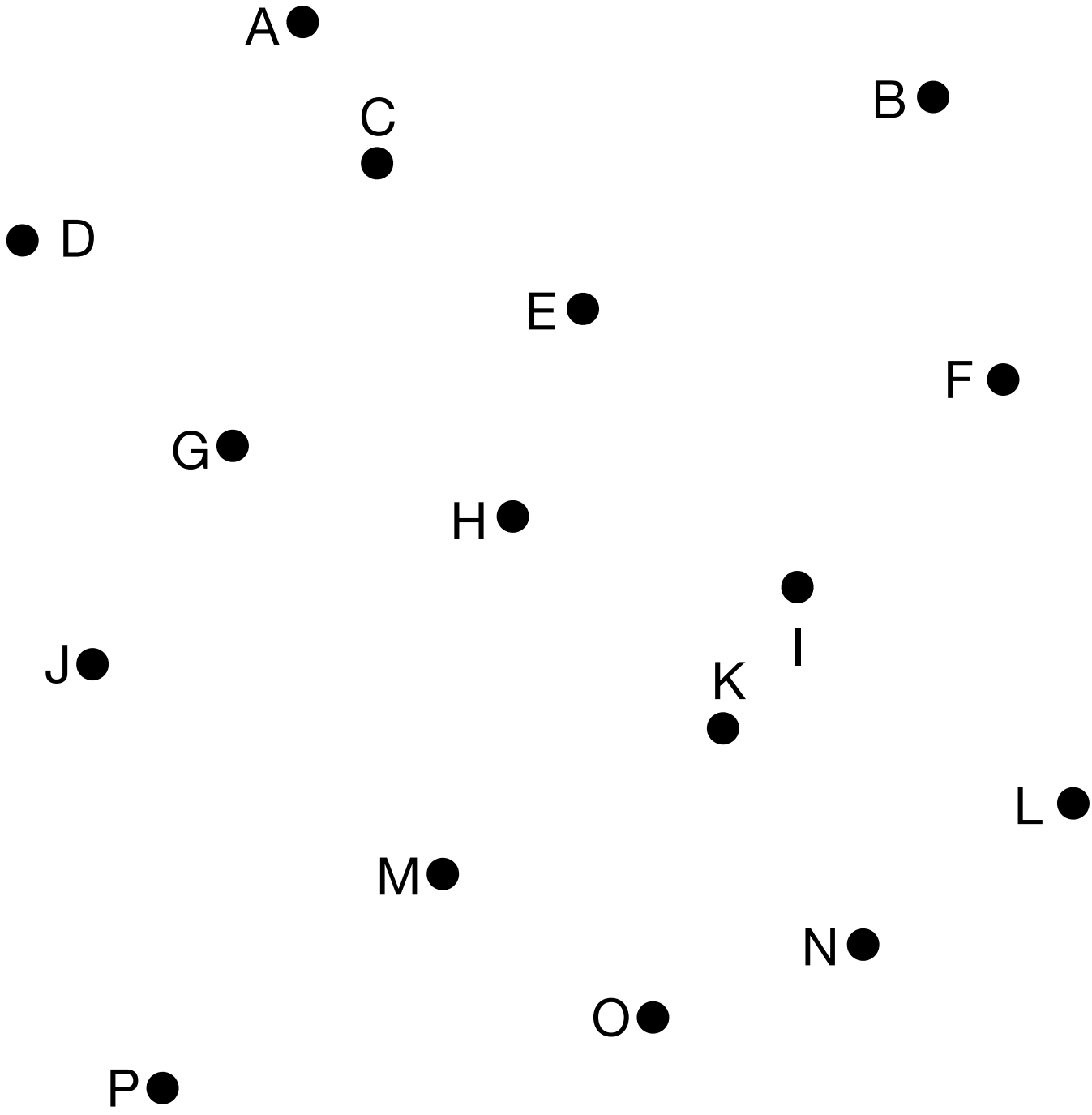
UB-tree
2000

KD-Trees - 1975

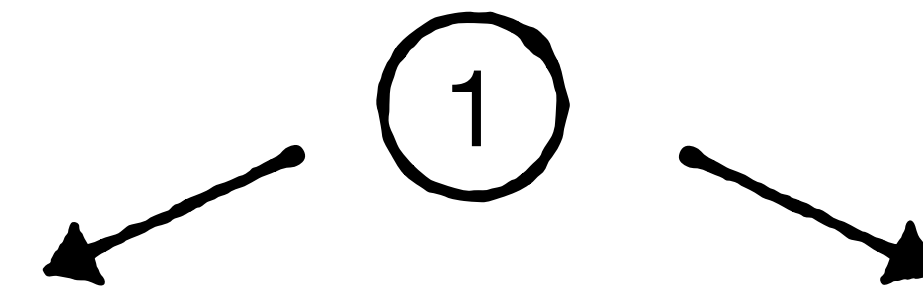
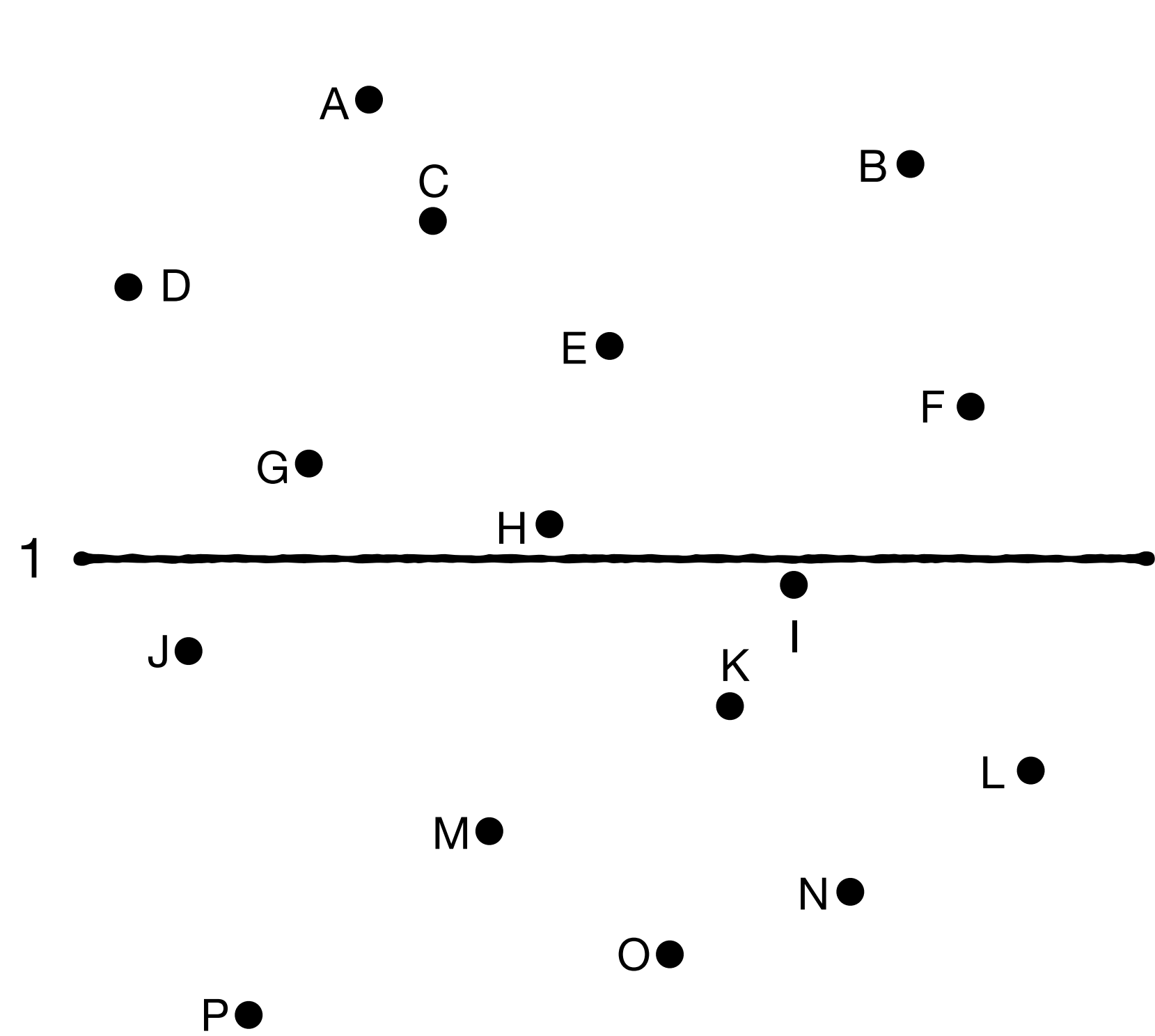
An algorithm for finding best matches in logarithmic expected time. ACM TOMS.
Jerome Harold Friedman, Jon Louis Bentley, and Raphael Ari Finkel.



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

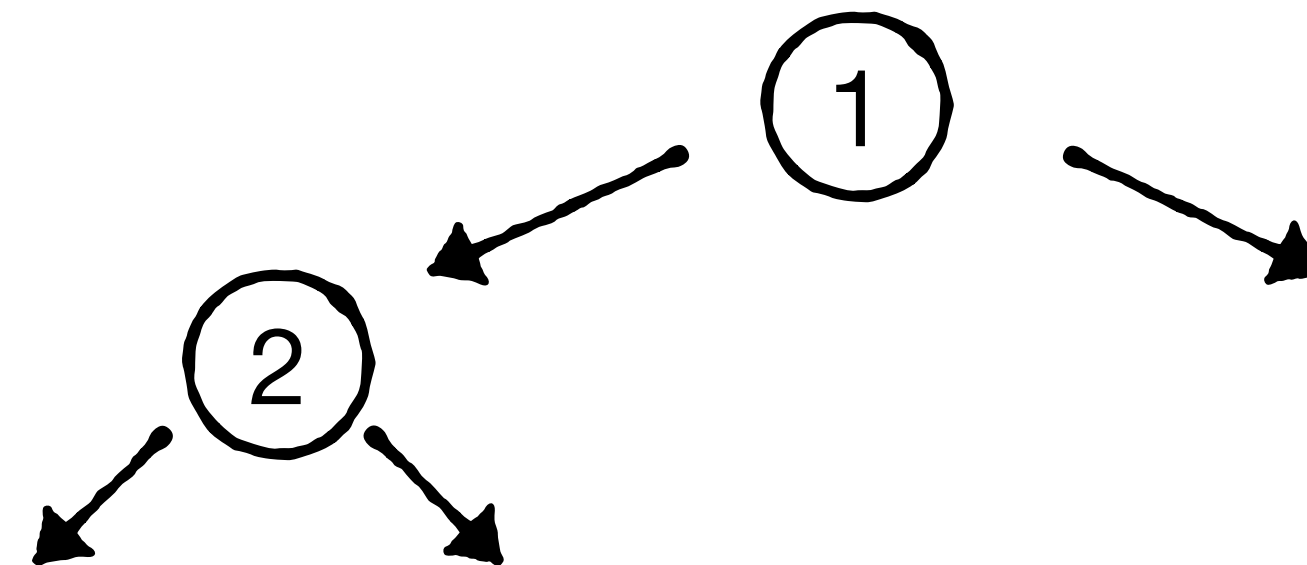
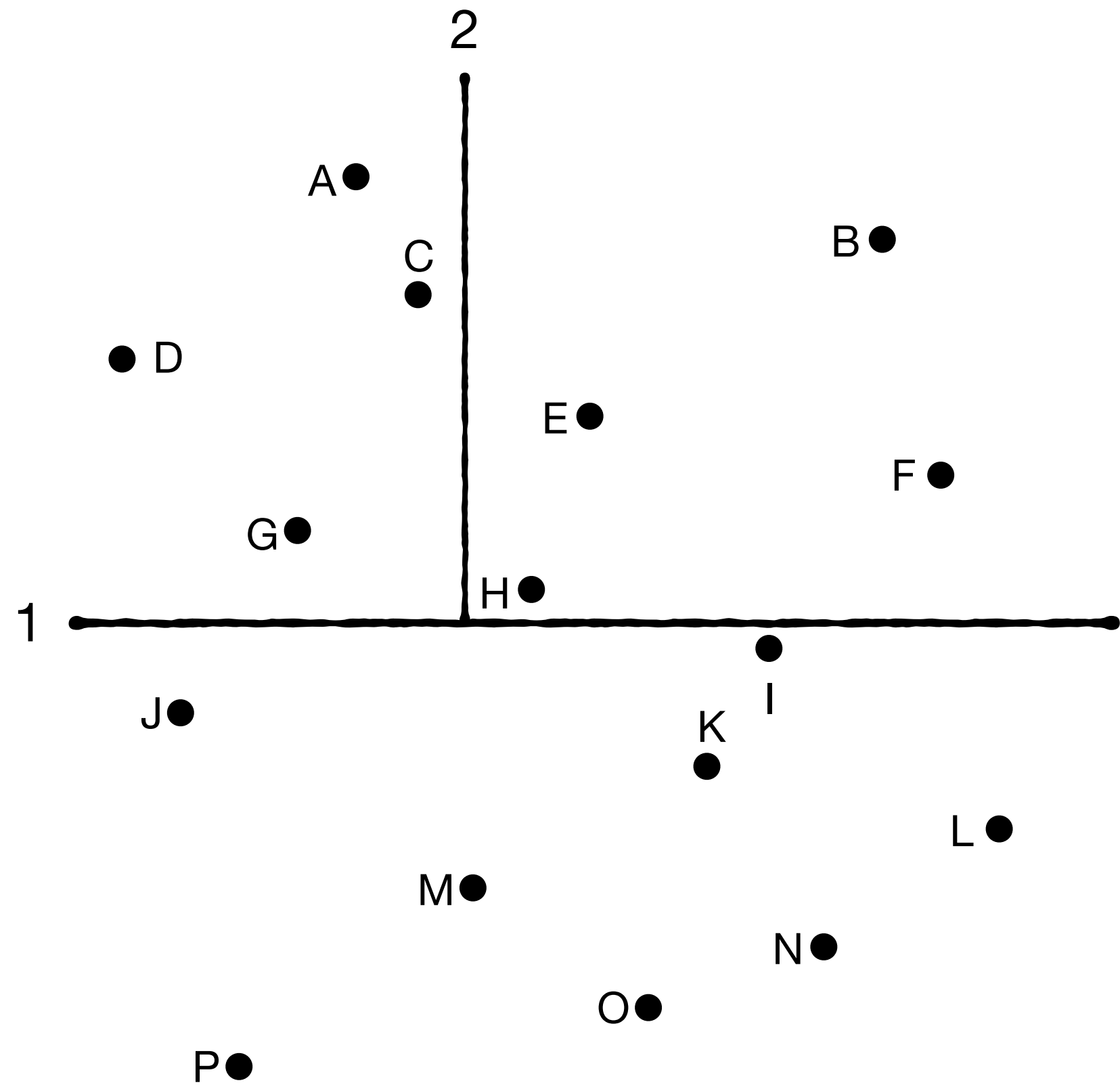


Partition based on median in one dimension - add node to tree



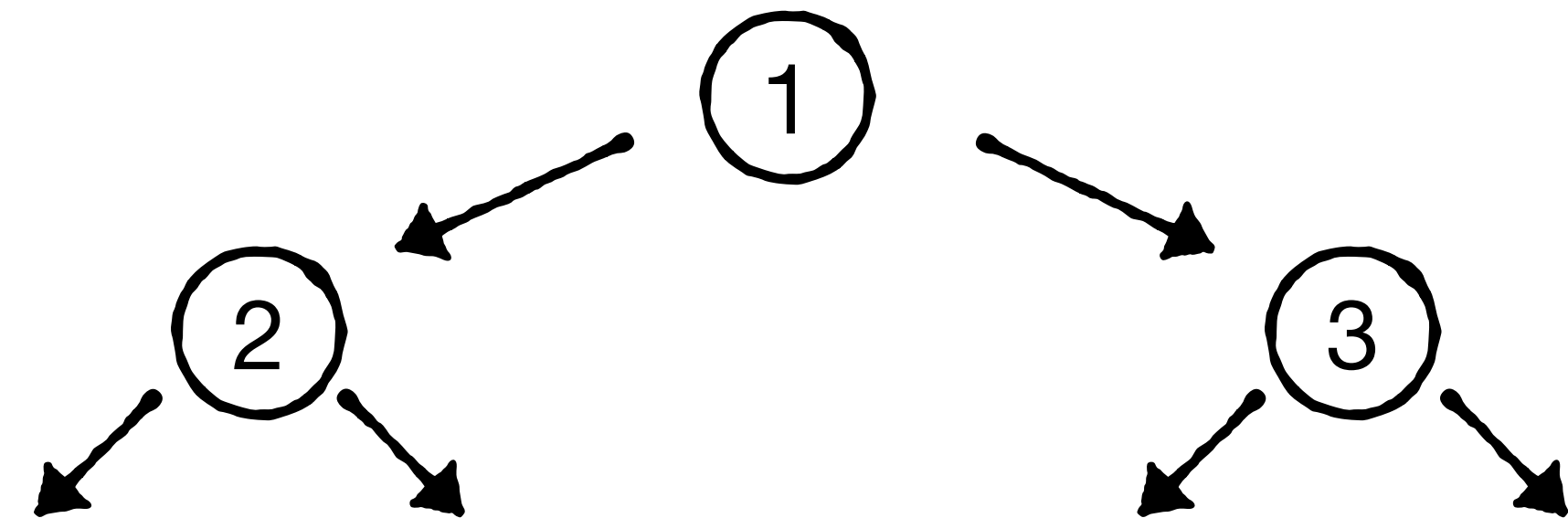
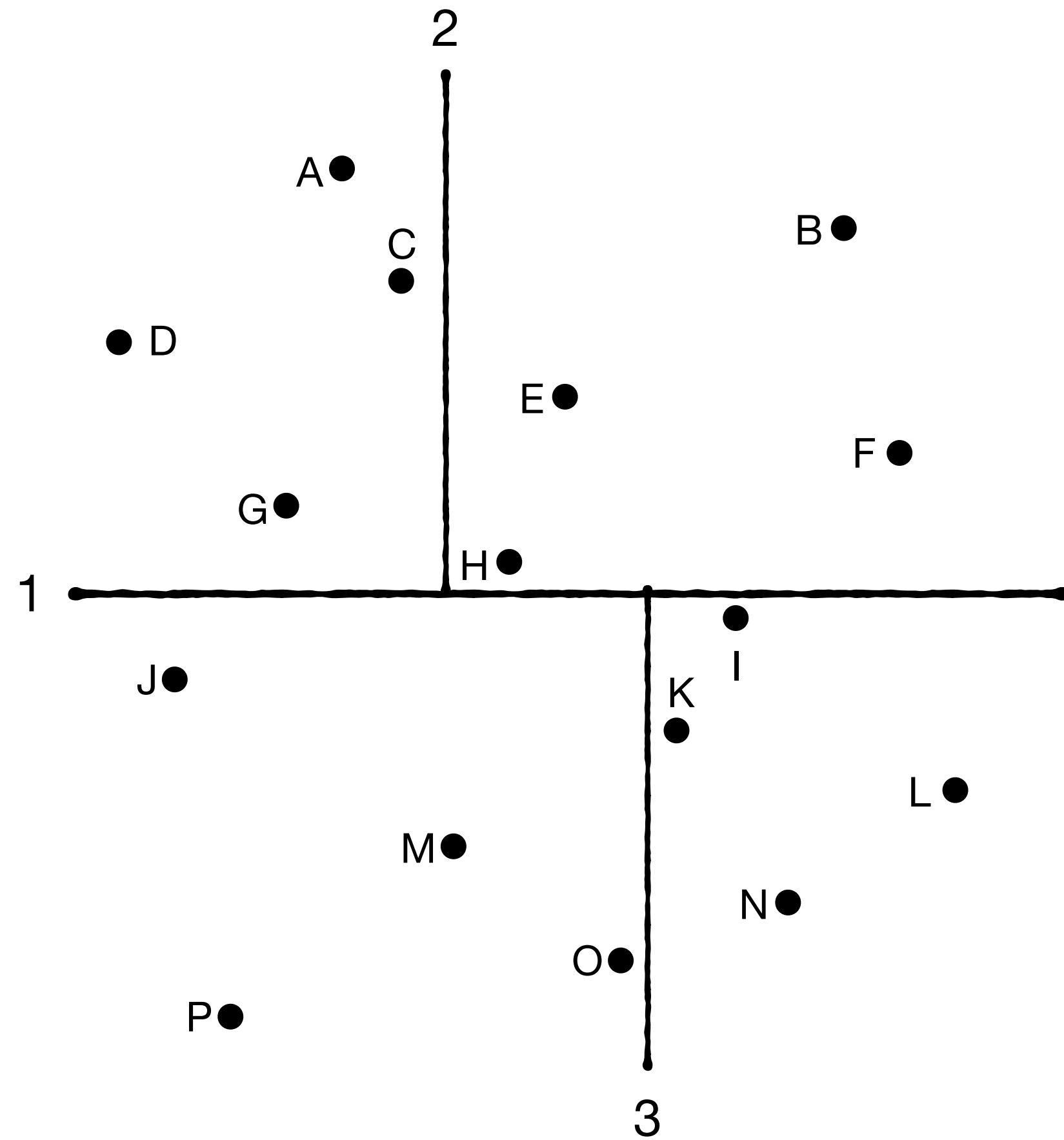
Partition based on median in one dimension - add node to tree

Recurse and alternate the dimensions



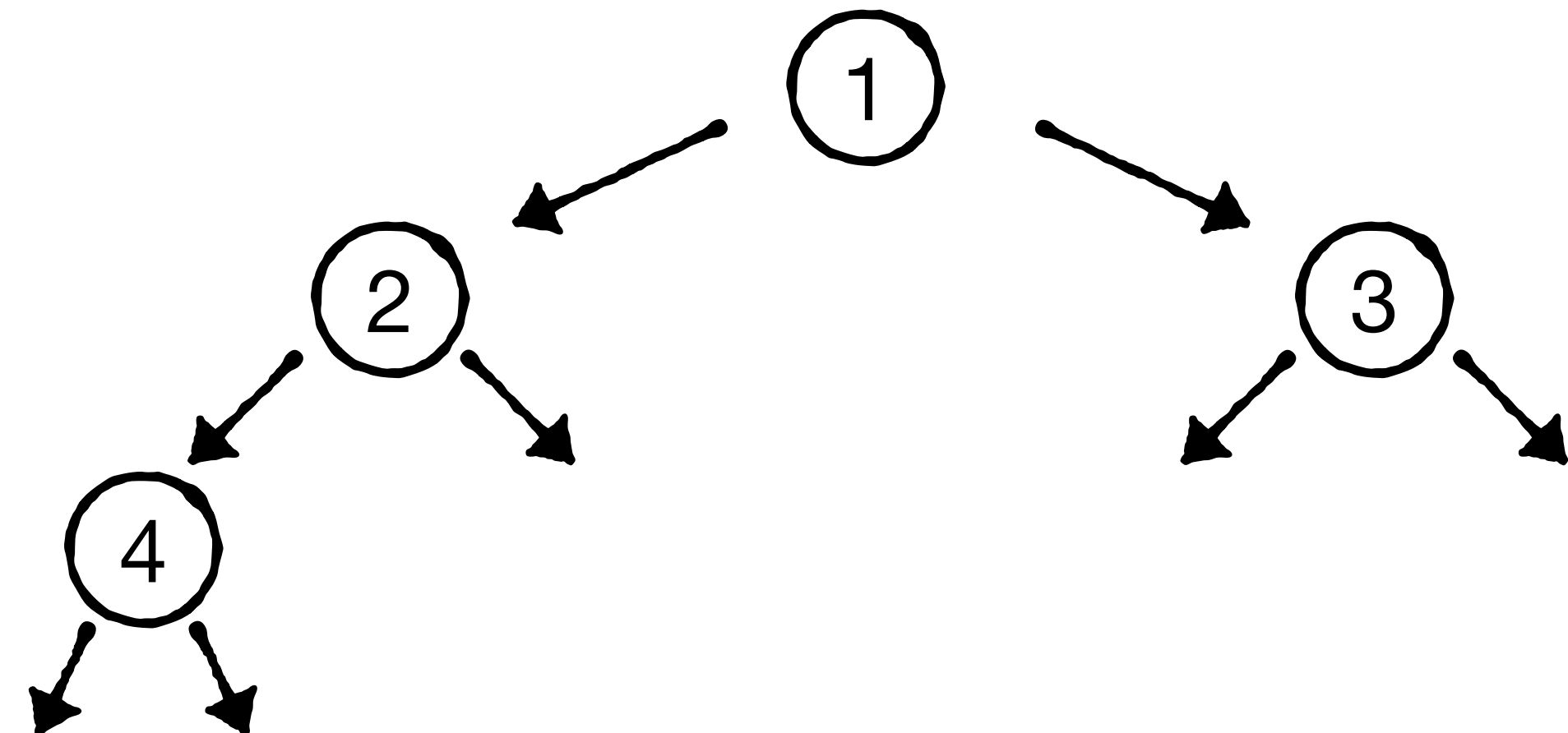
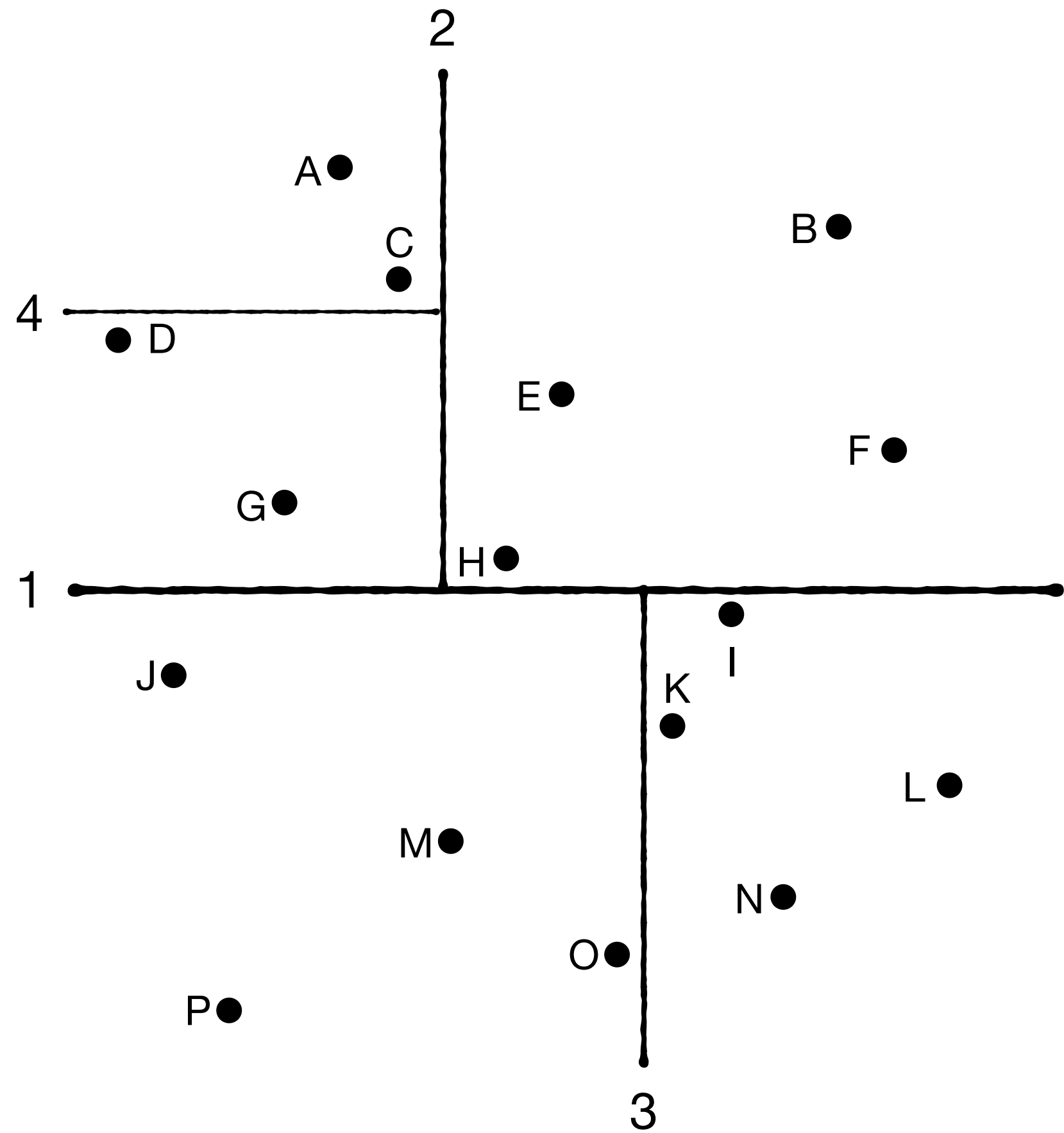
Partition based on median in one dimension - add node to tree

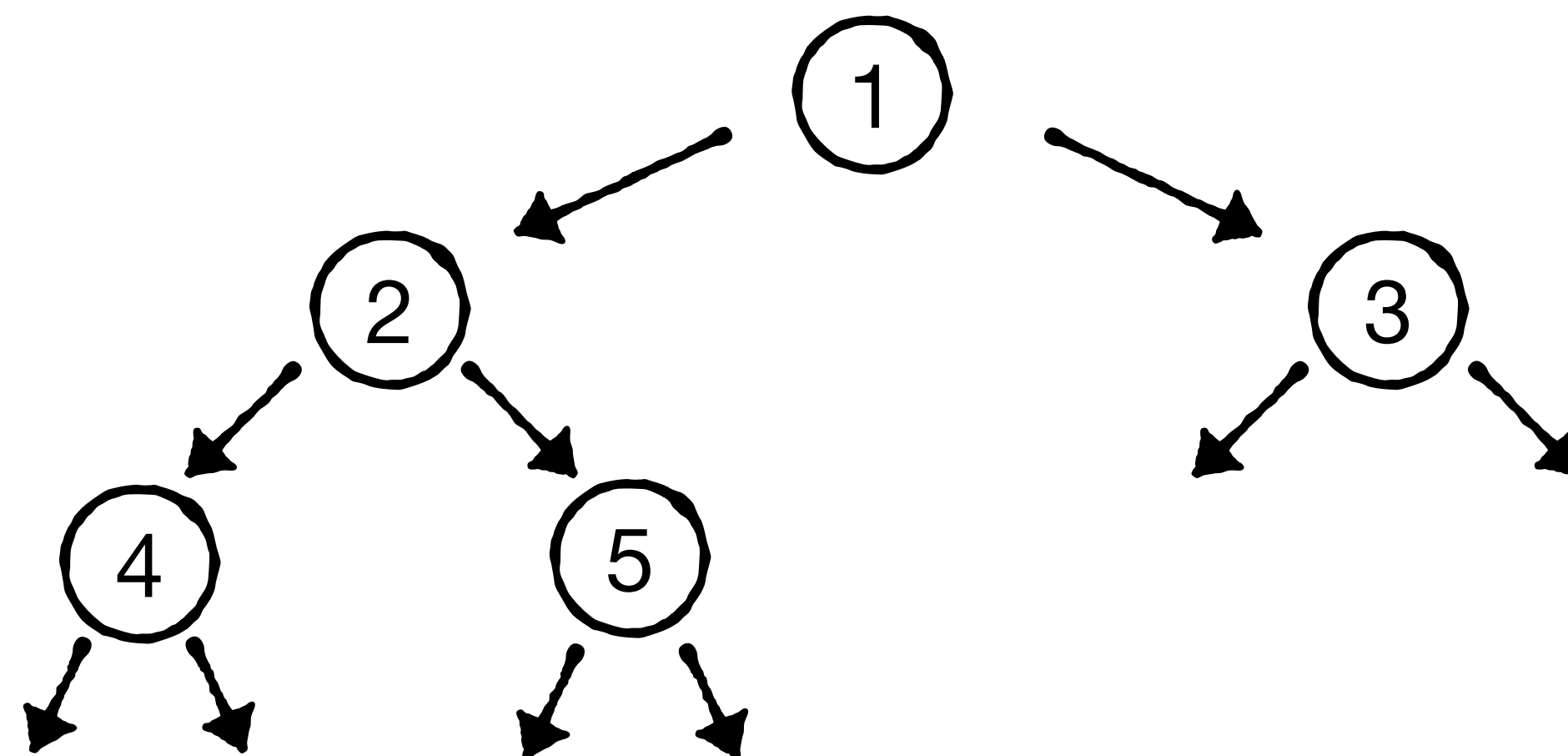
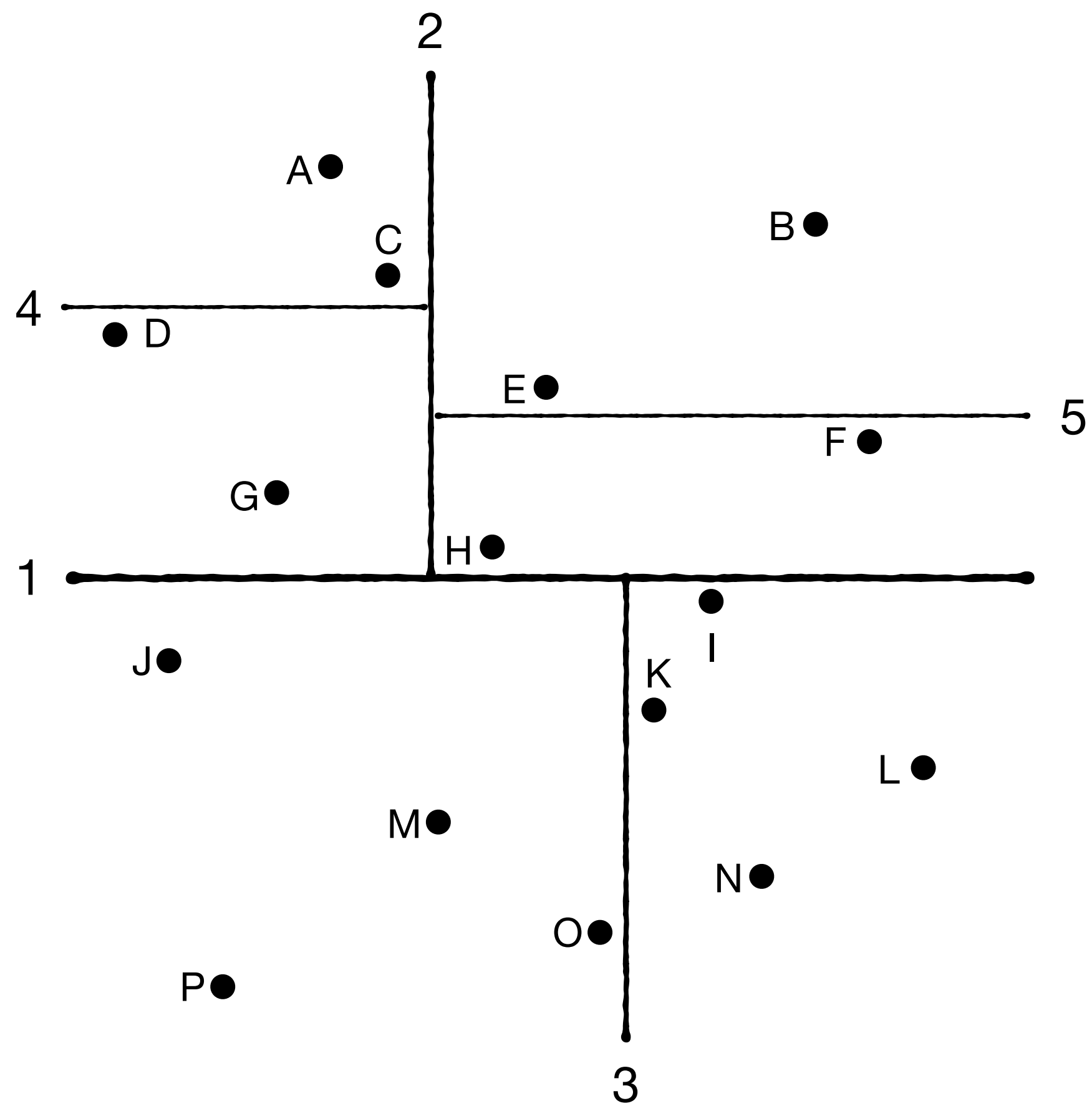
Recurse and alternate the dimensions

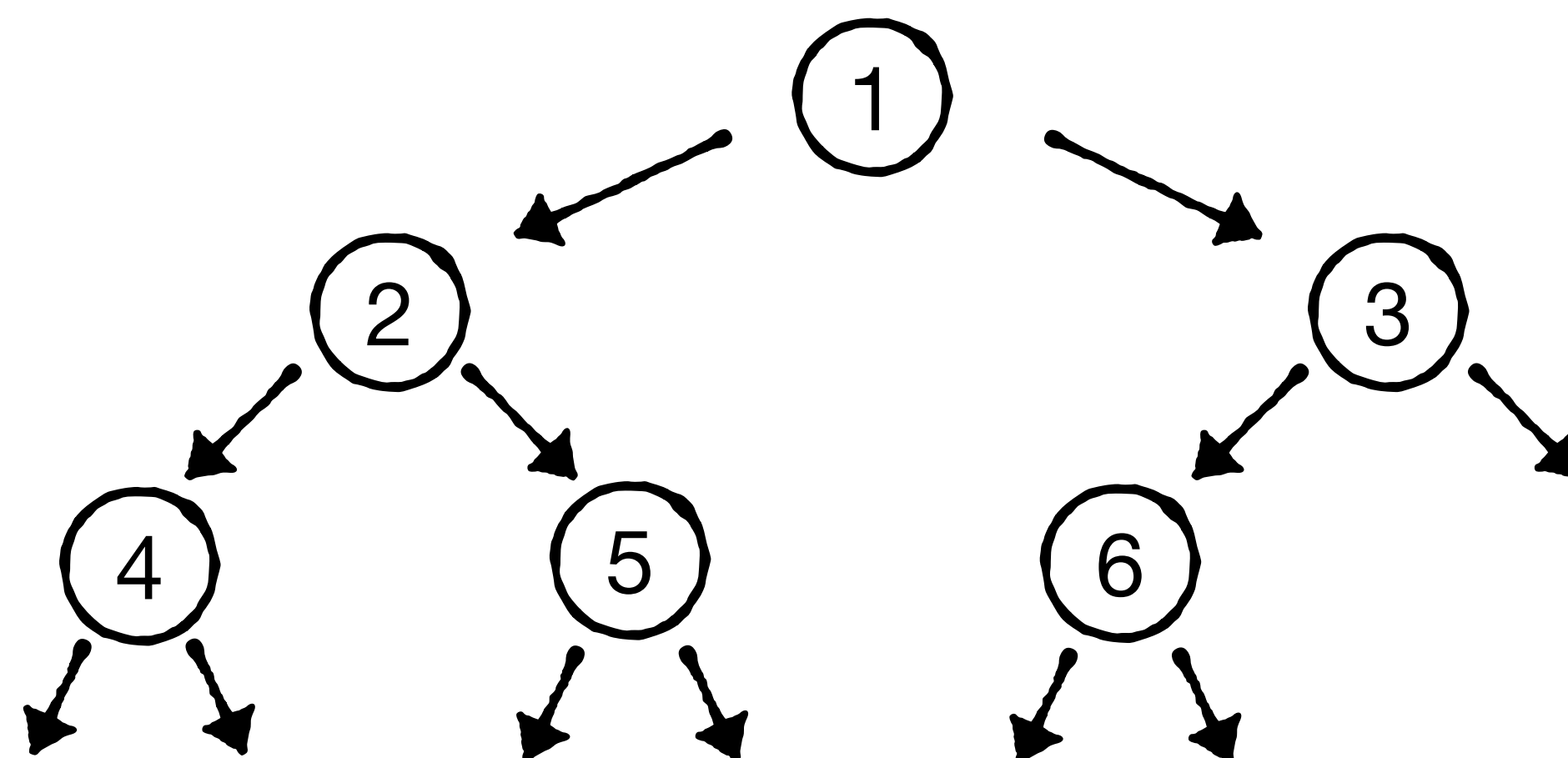
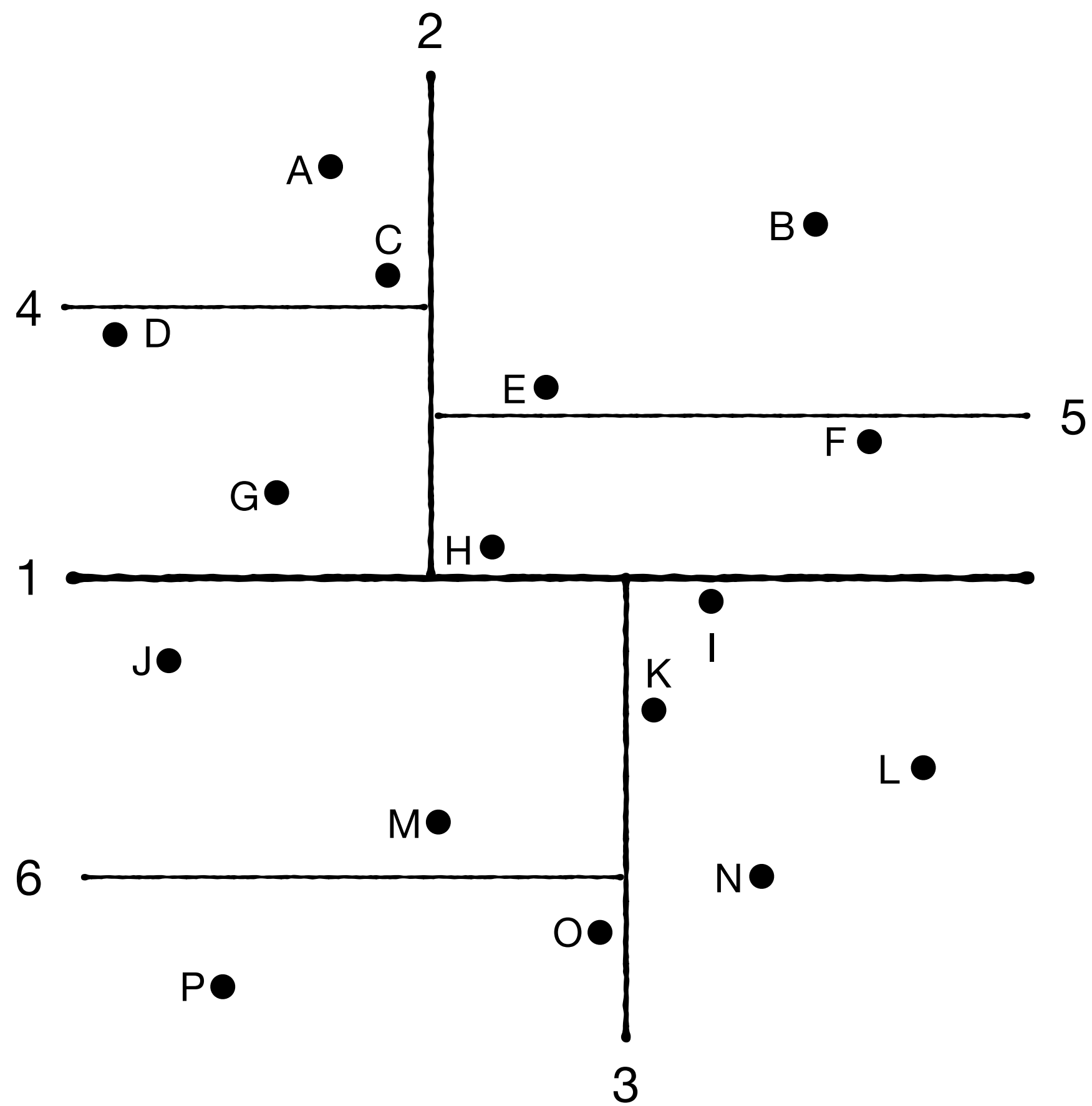


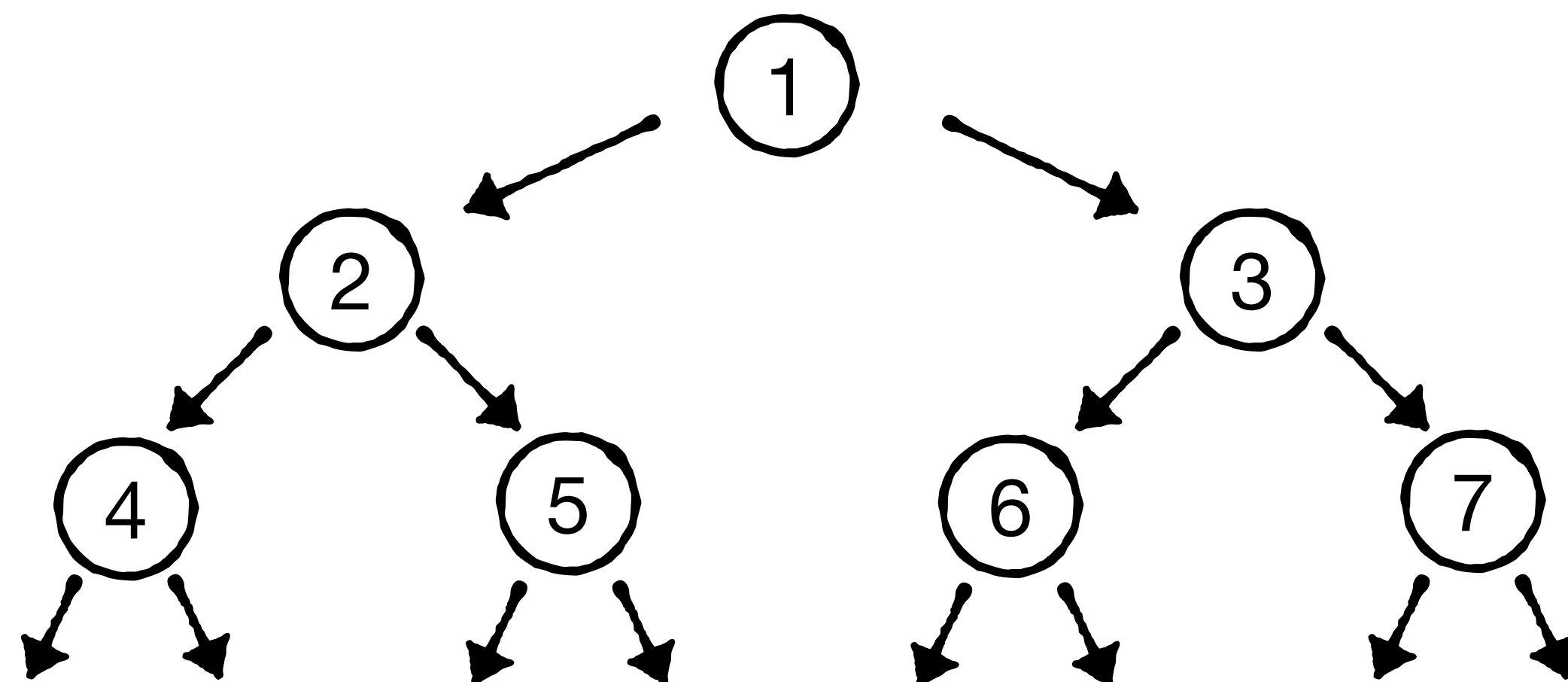
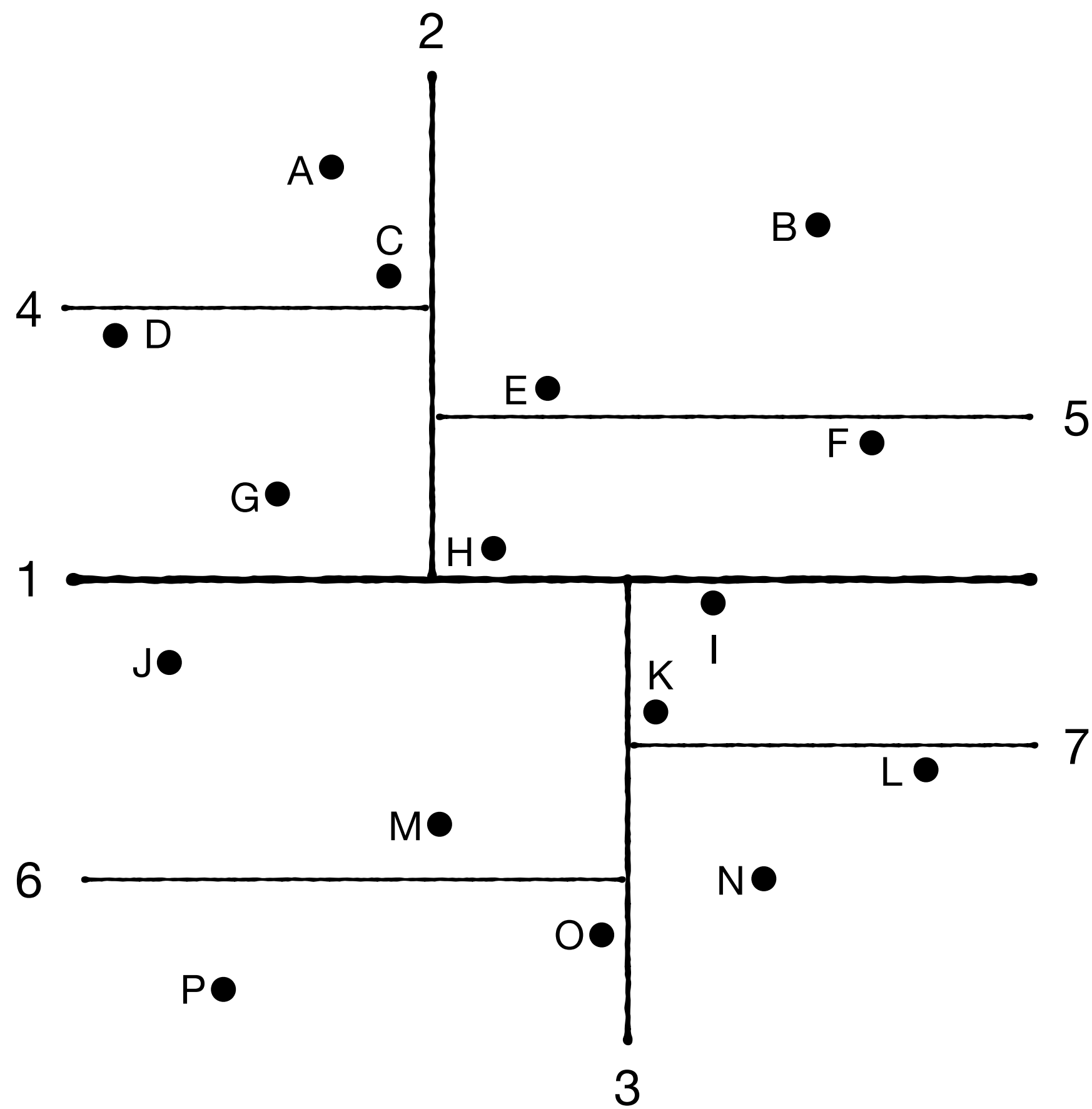
Partition based on median in one dimension - add node to tree

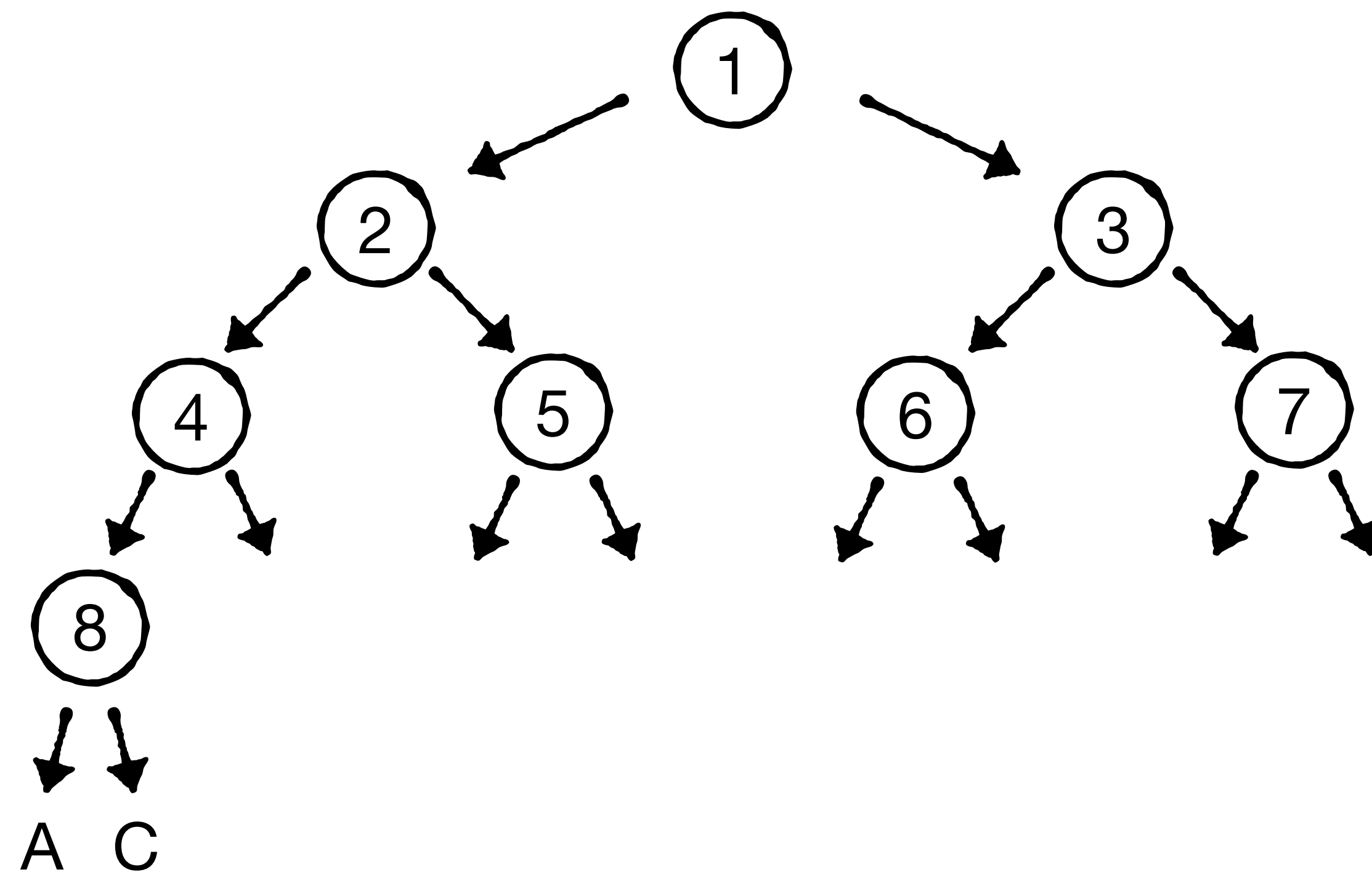
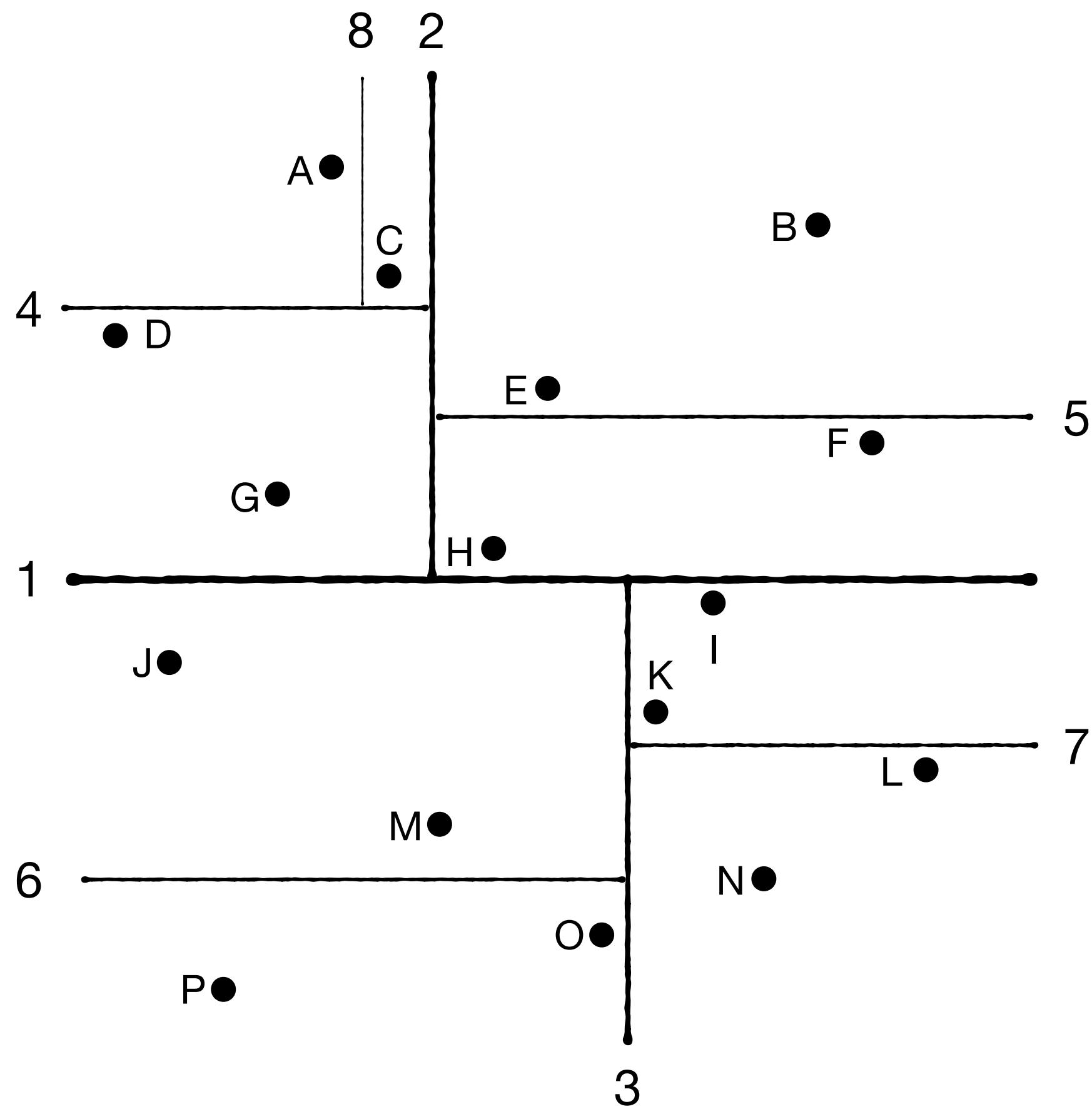
Recurse and alternate the dimensions

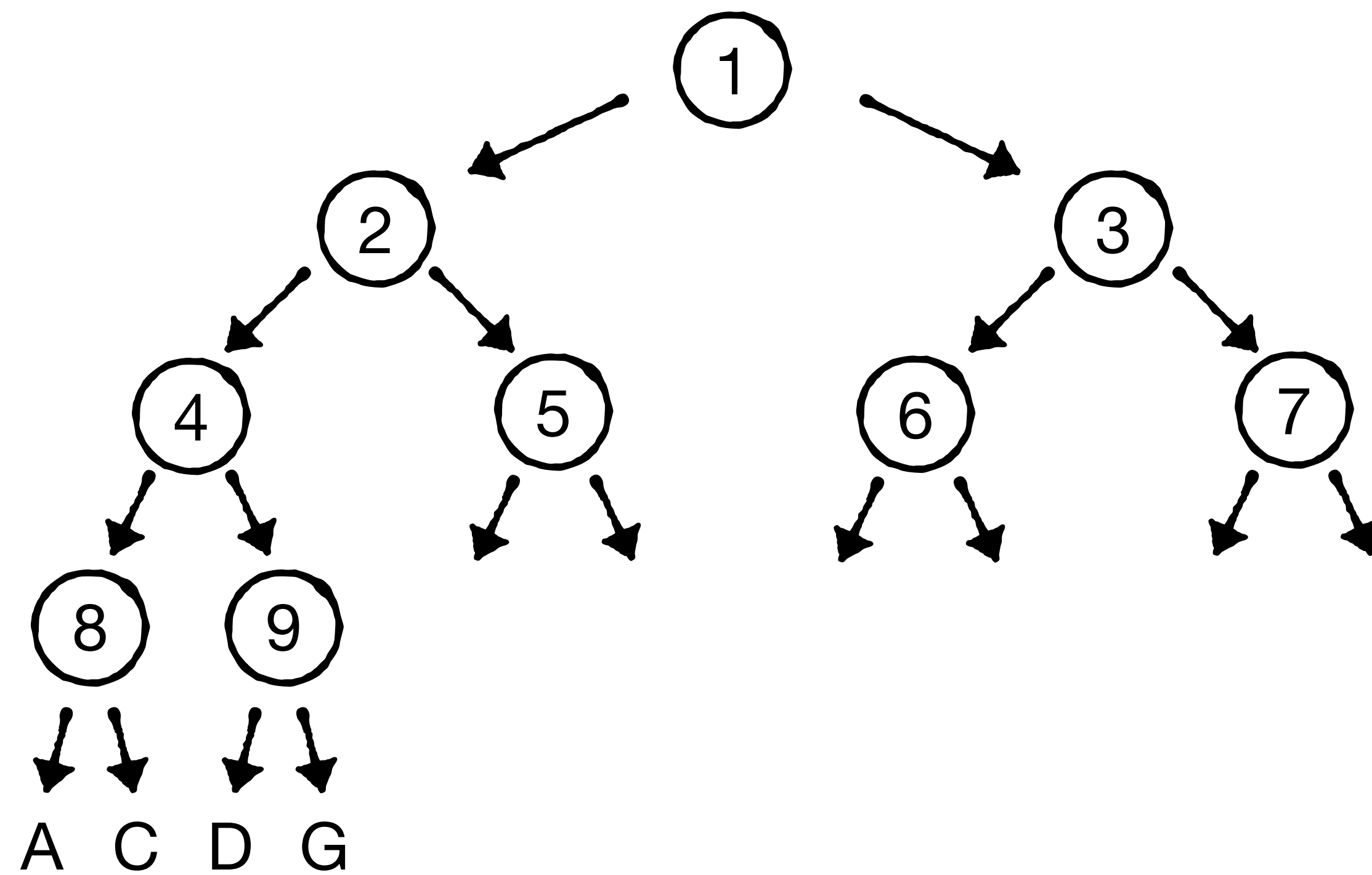
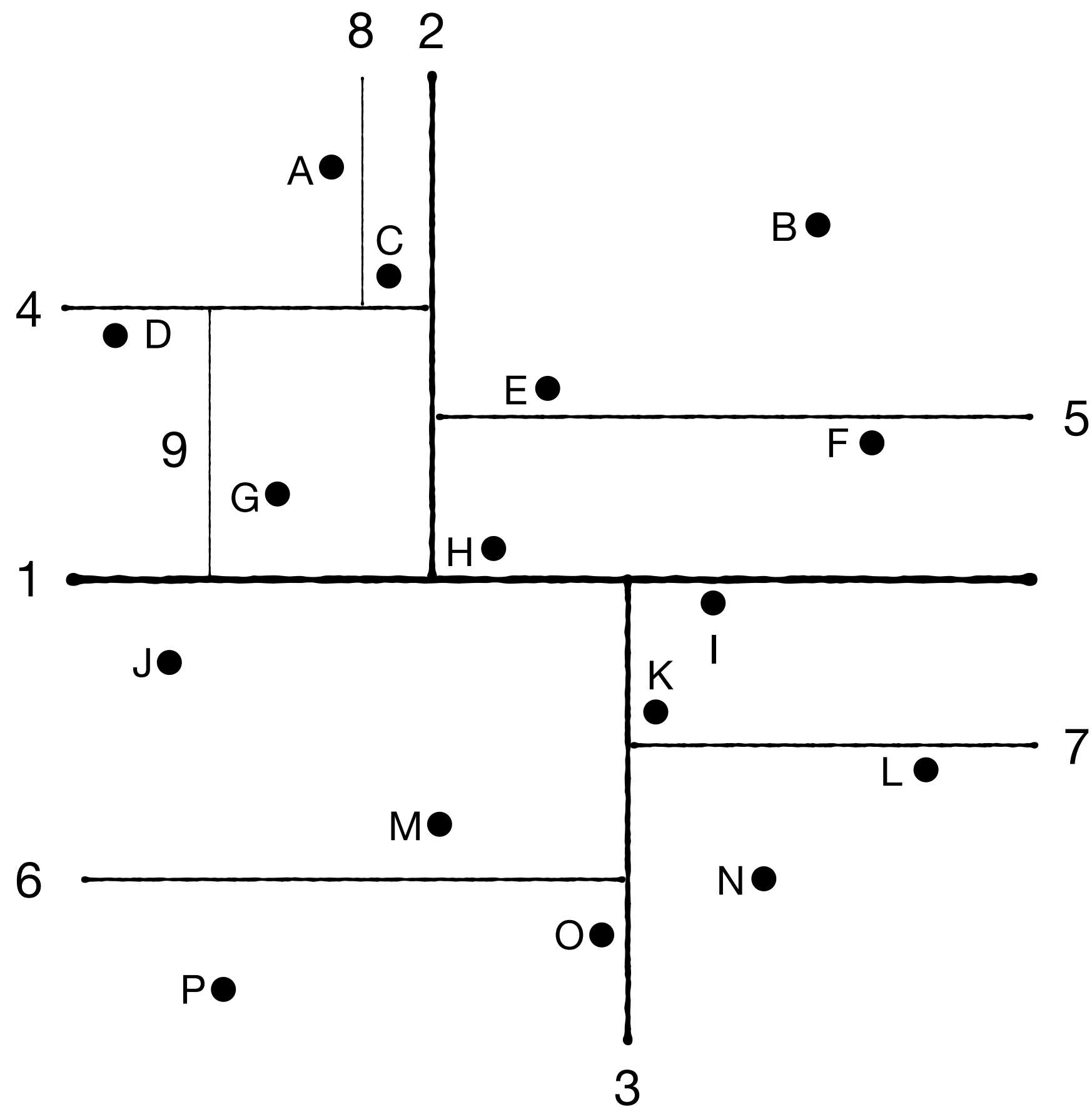


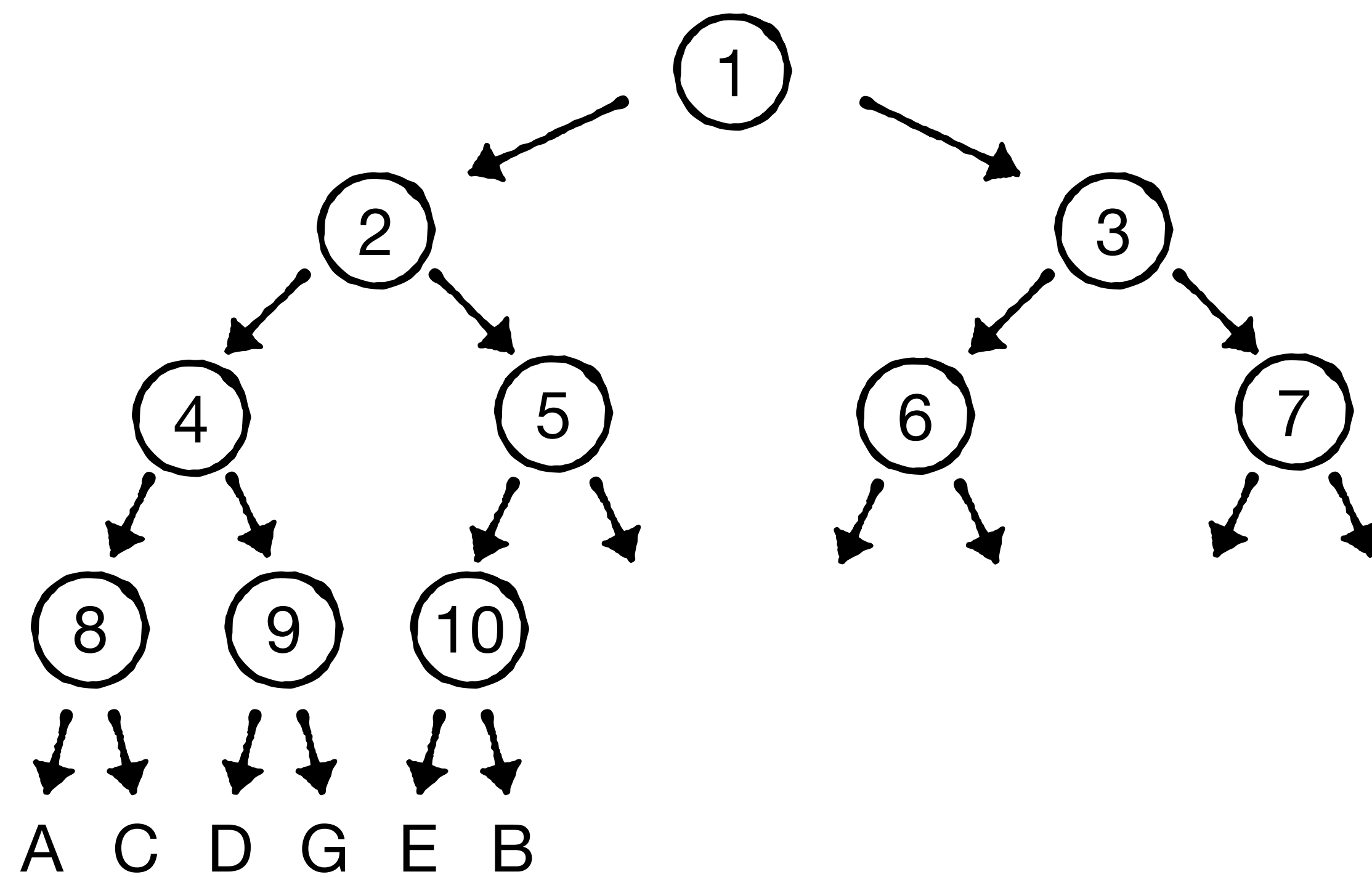
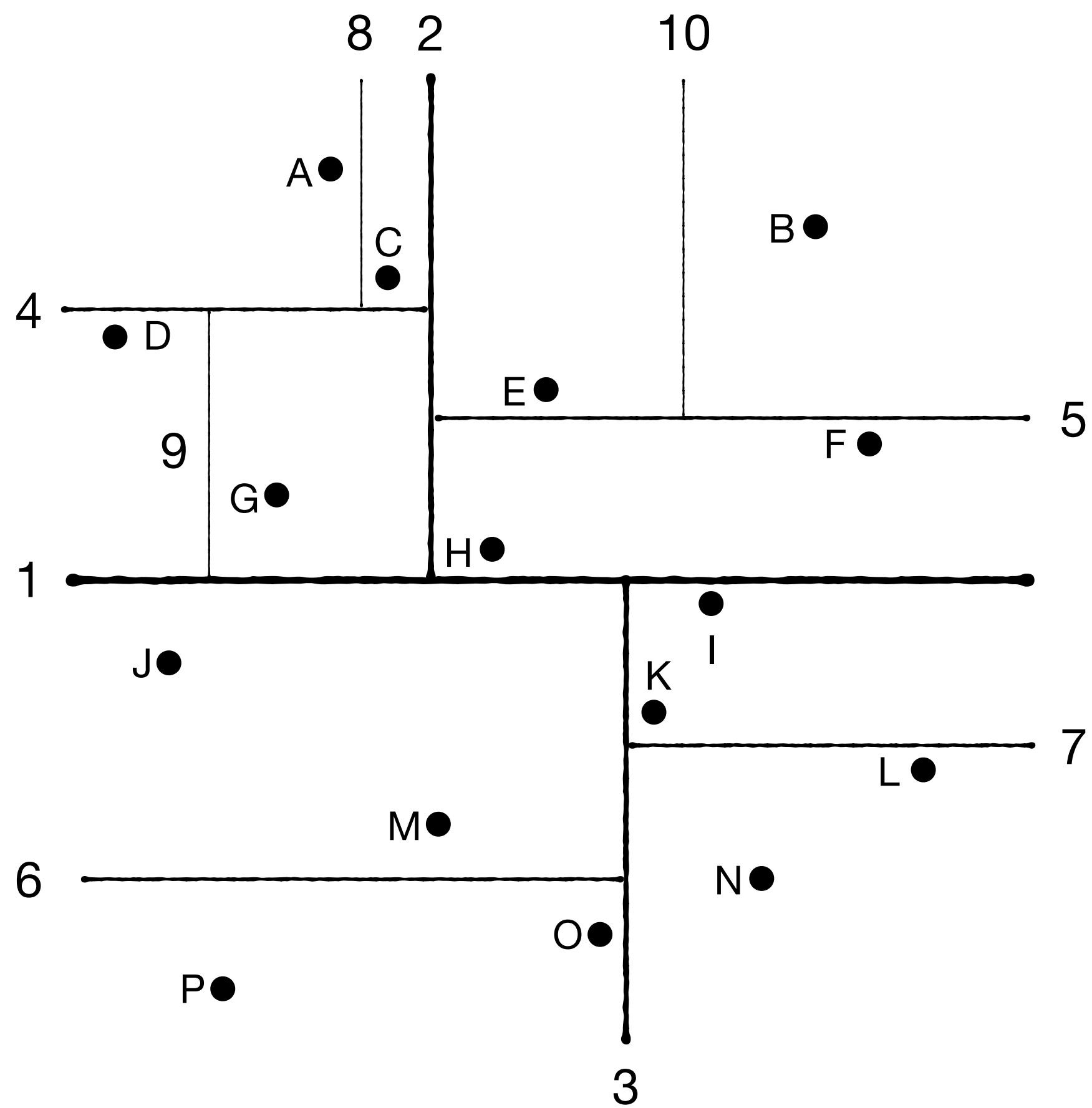


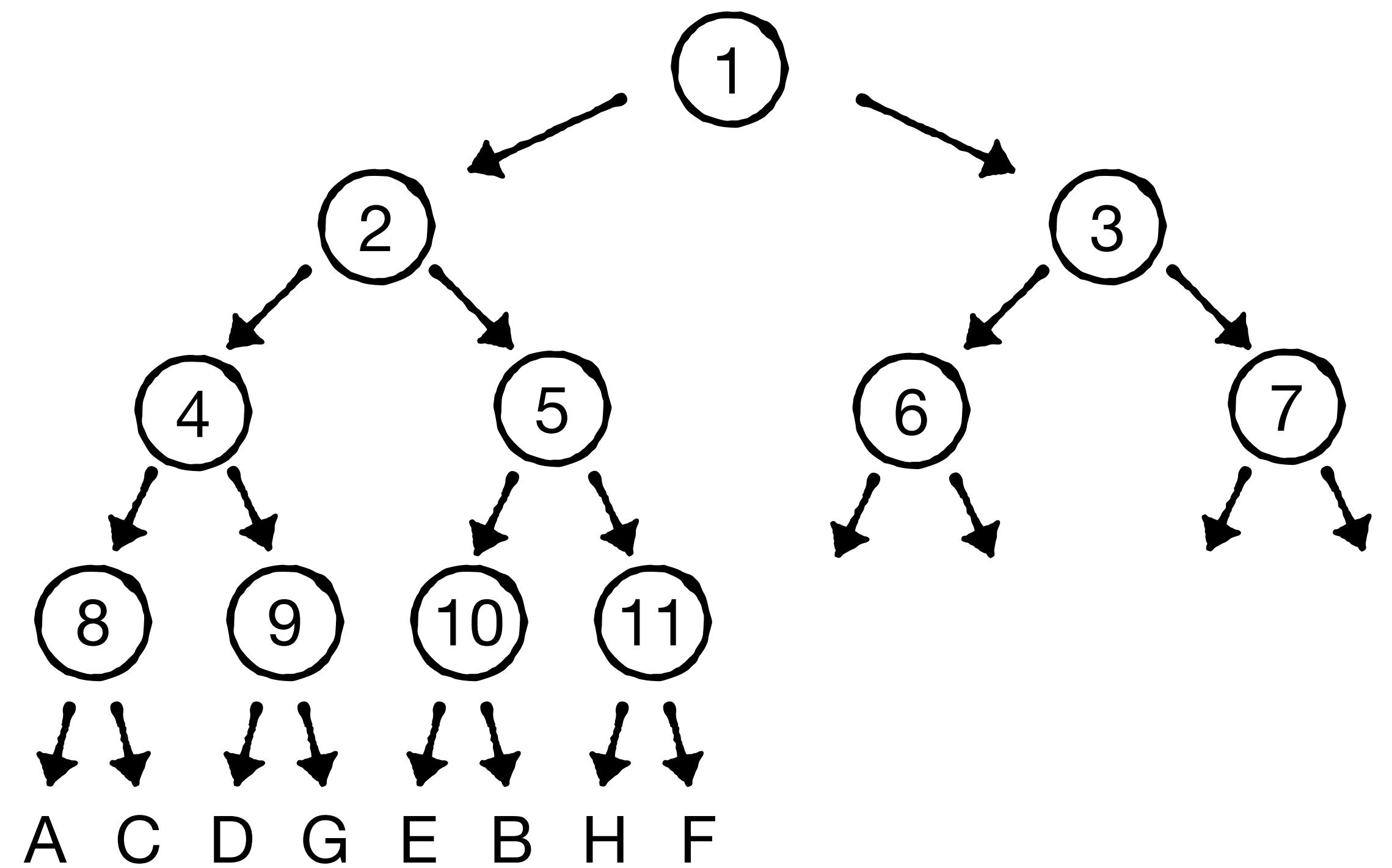
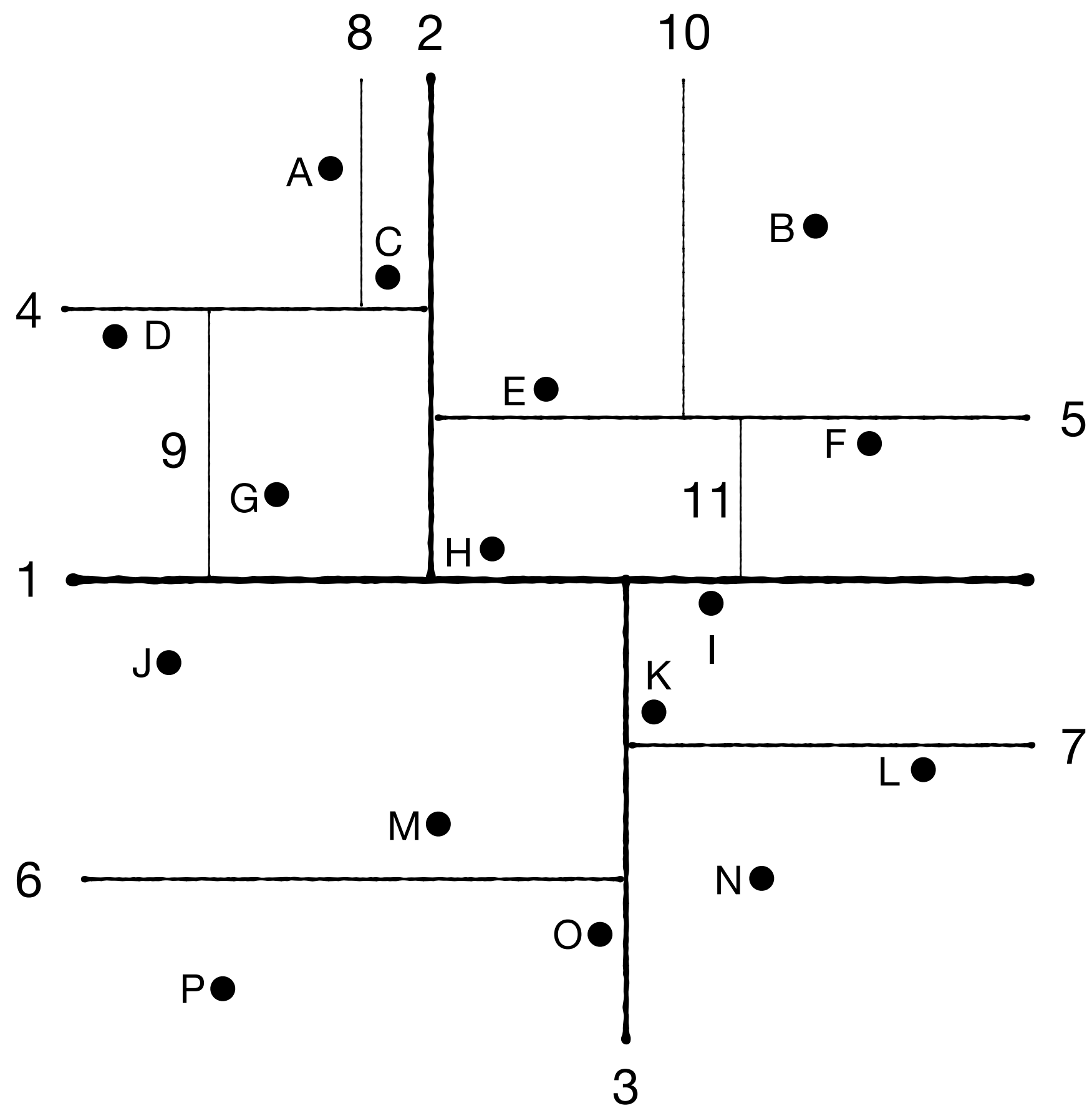


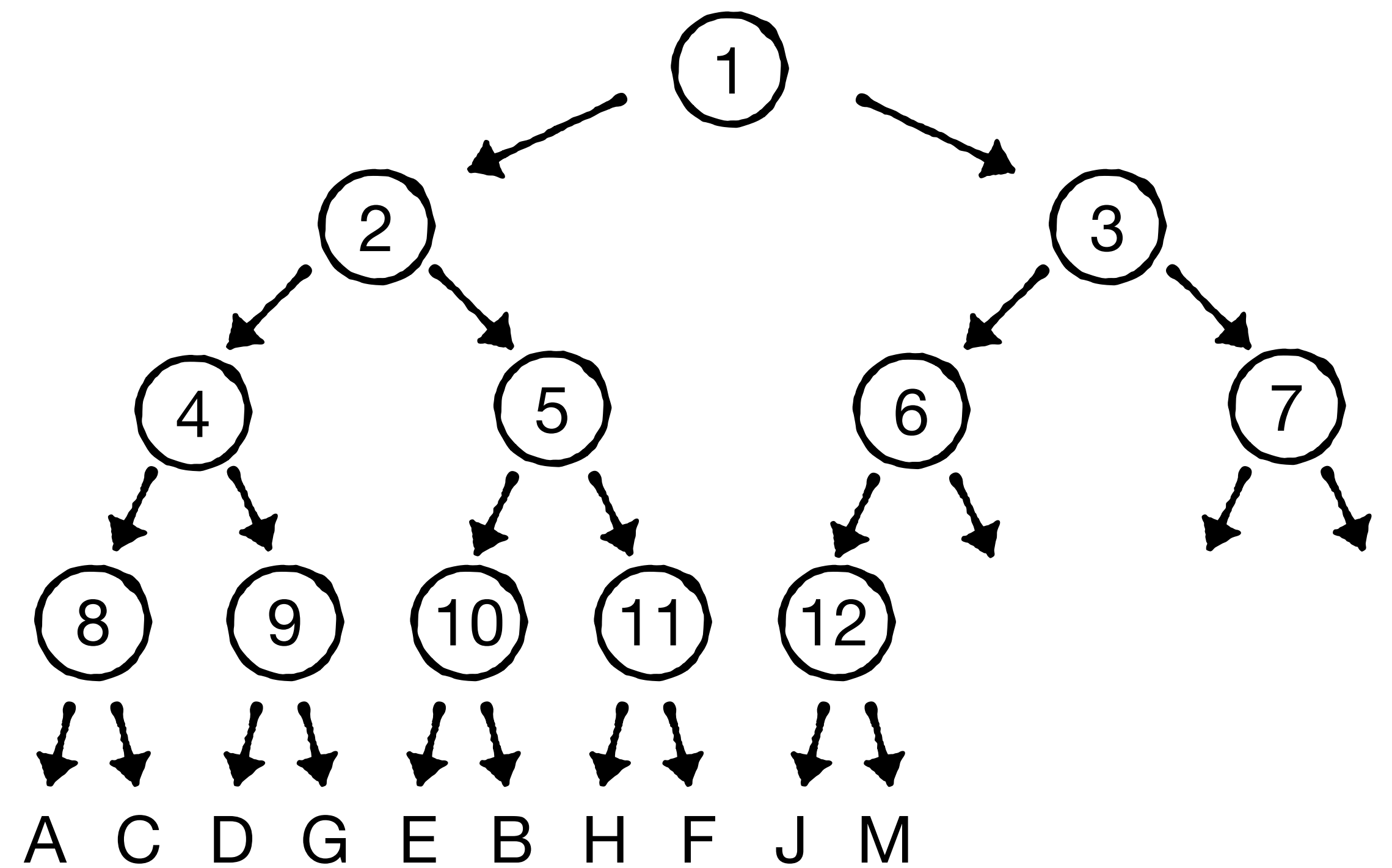
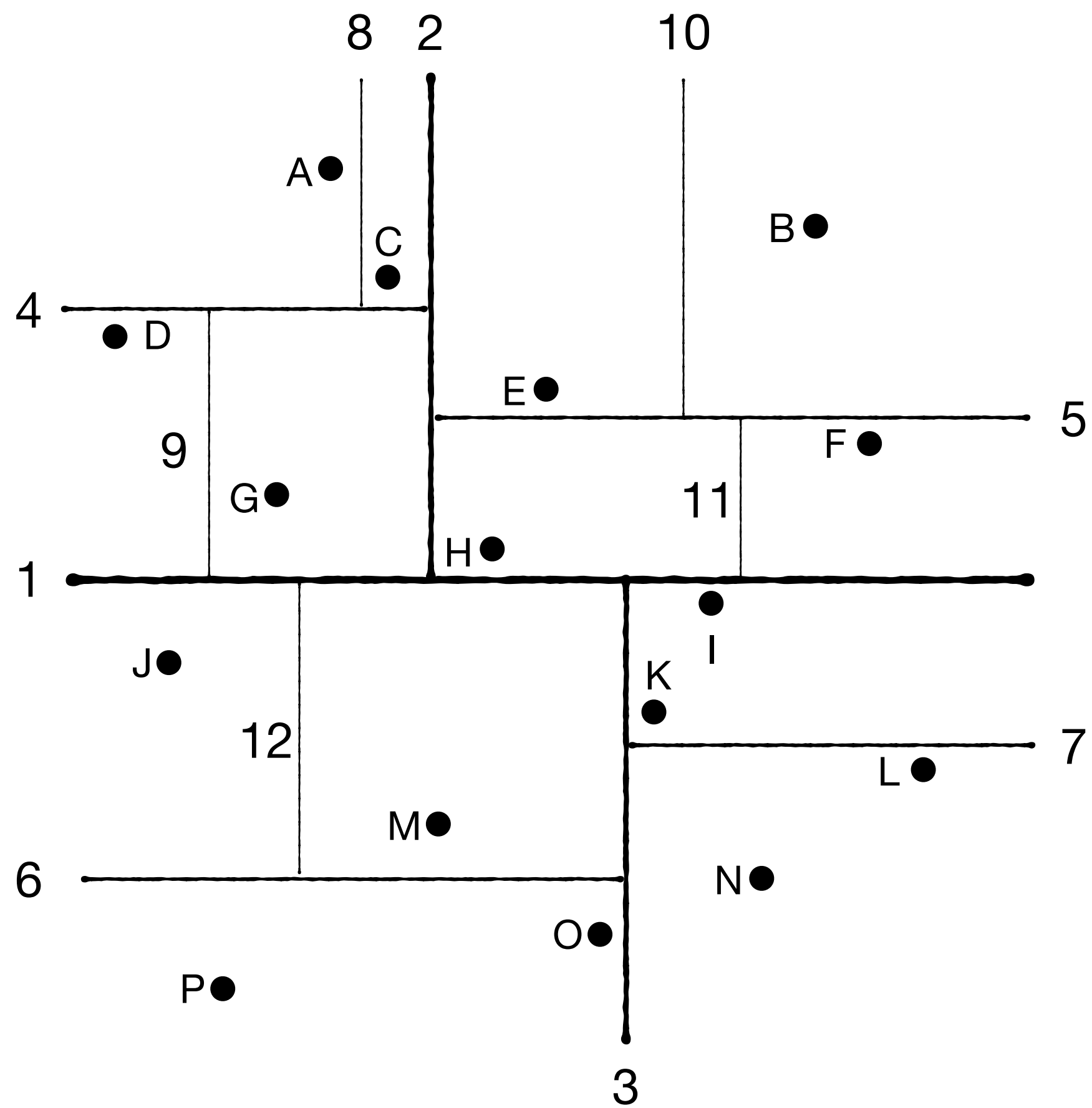


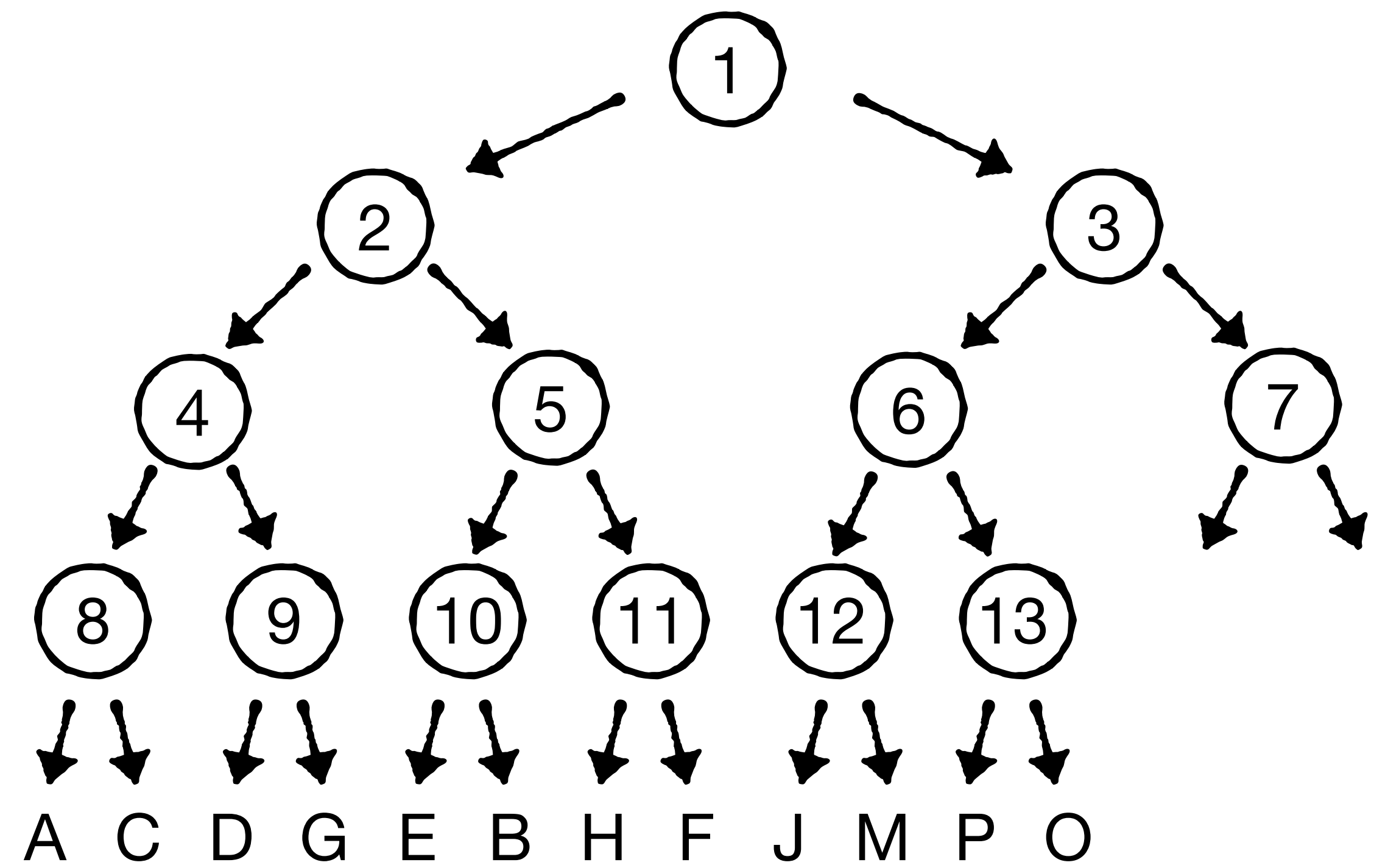
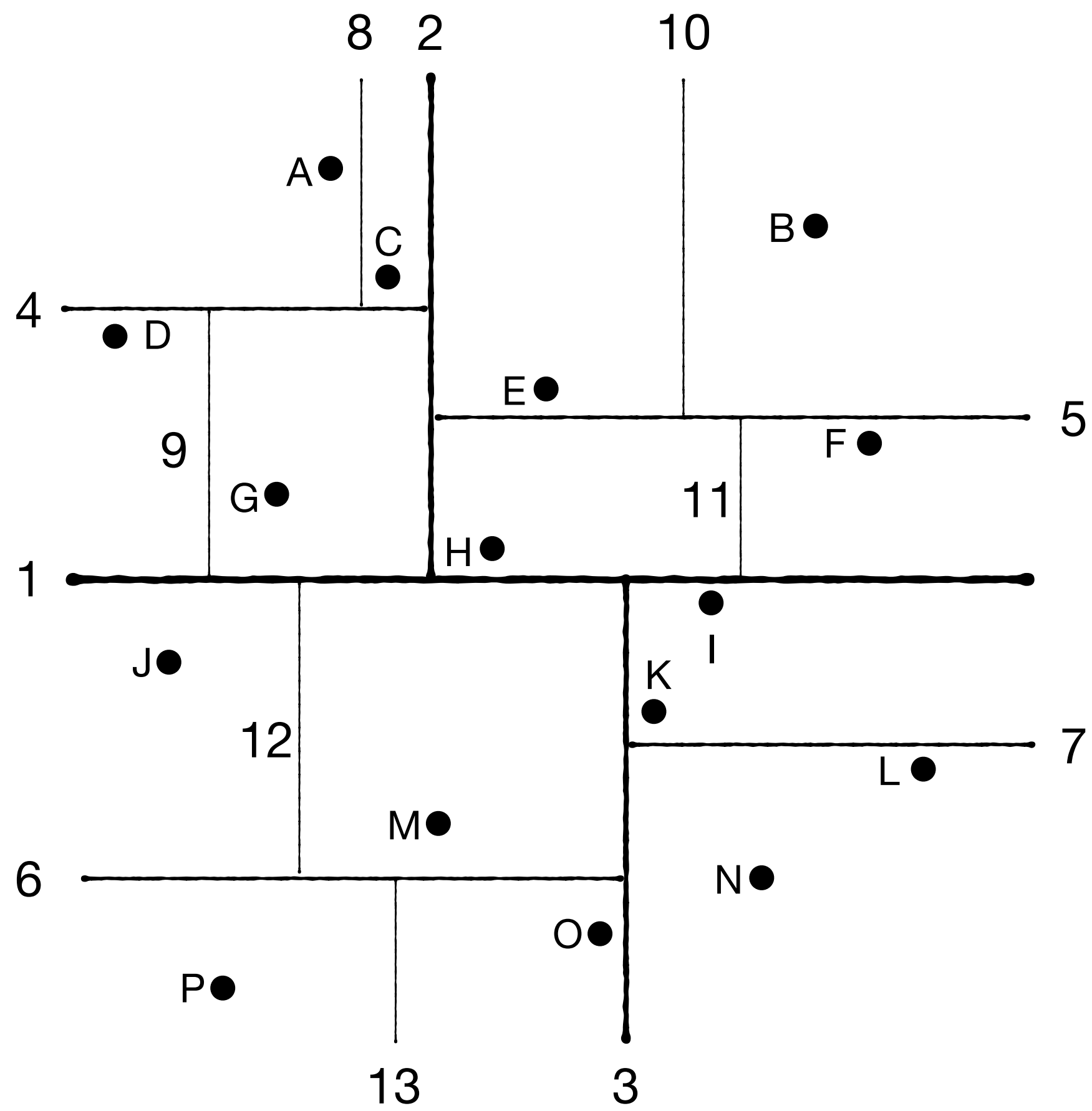


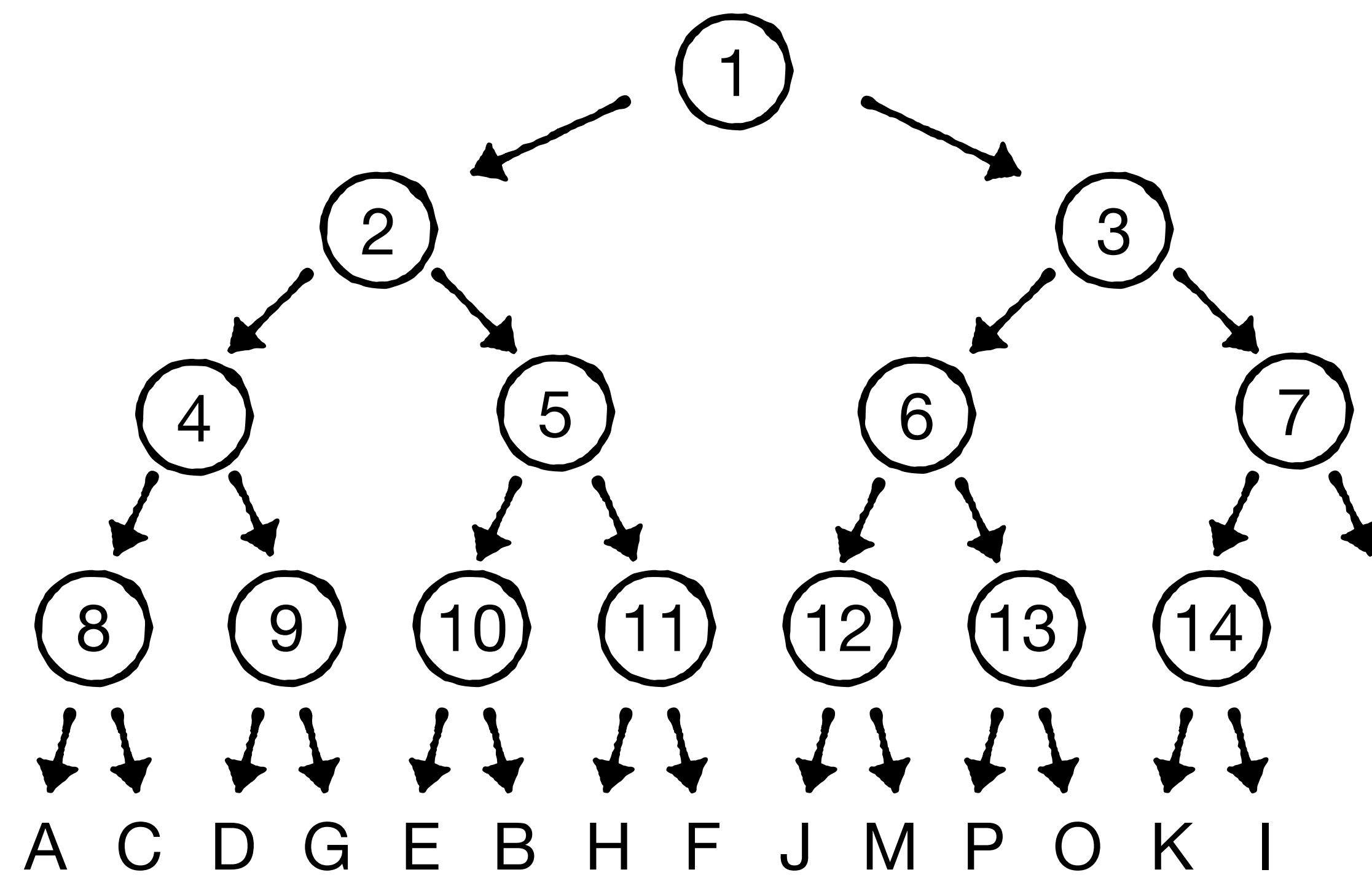
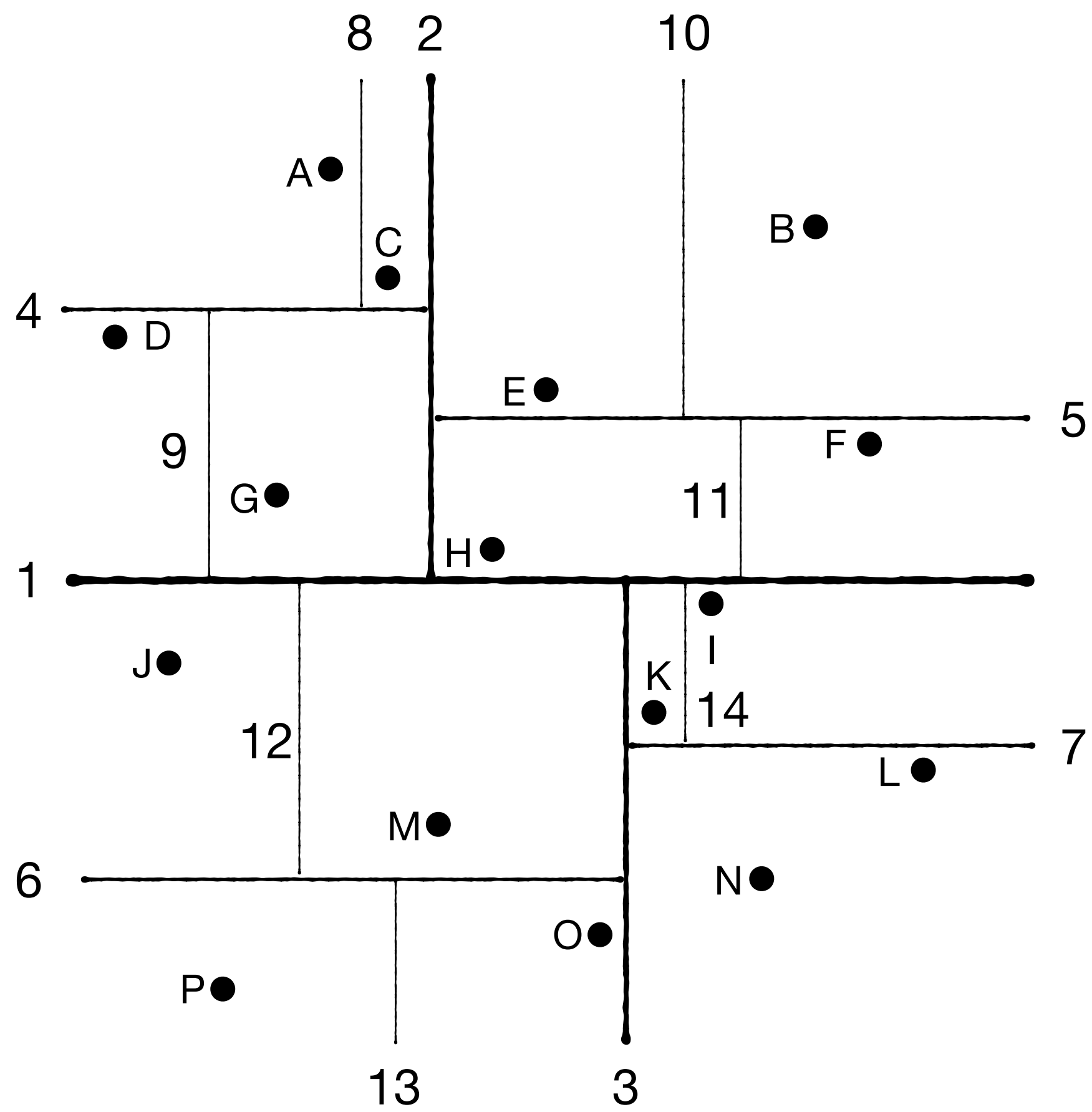


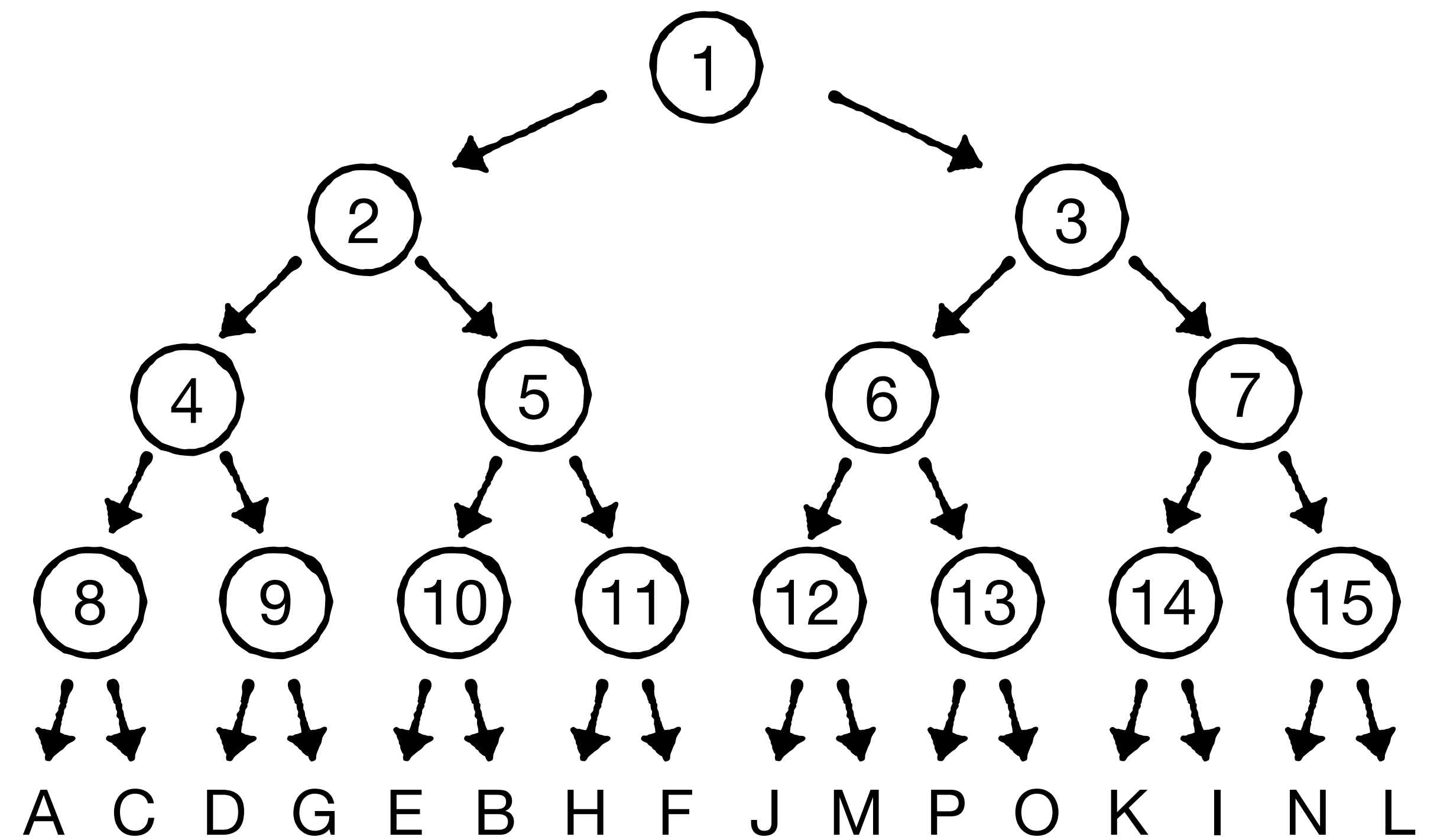
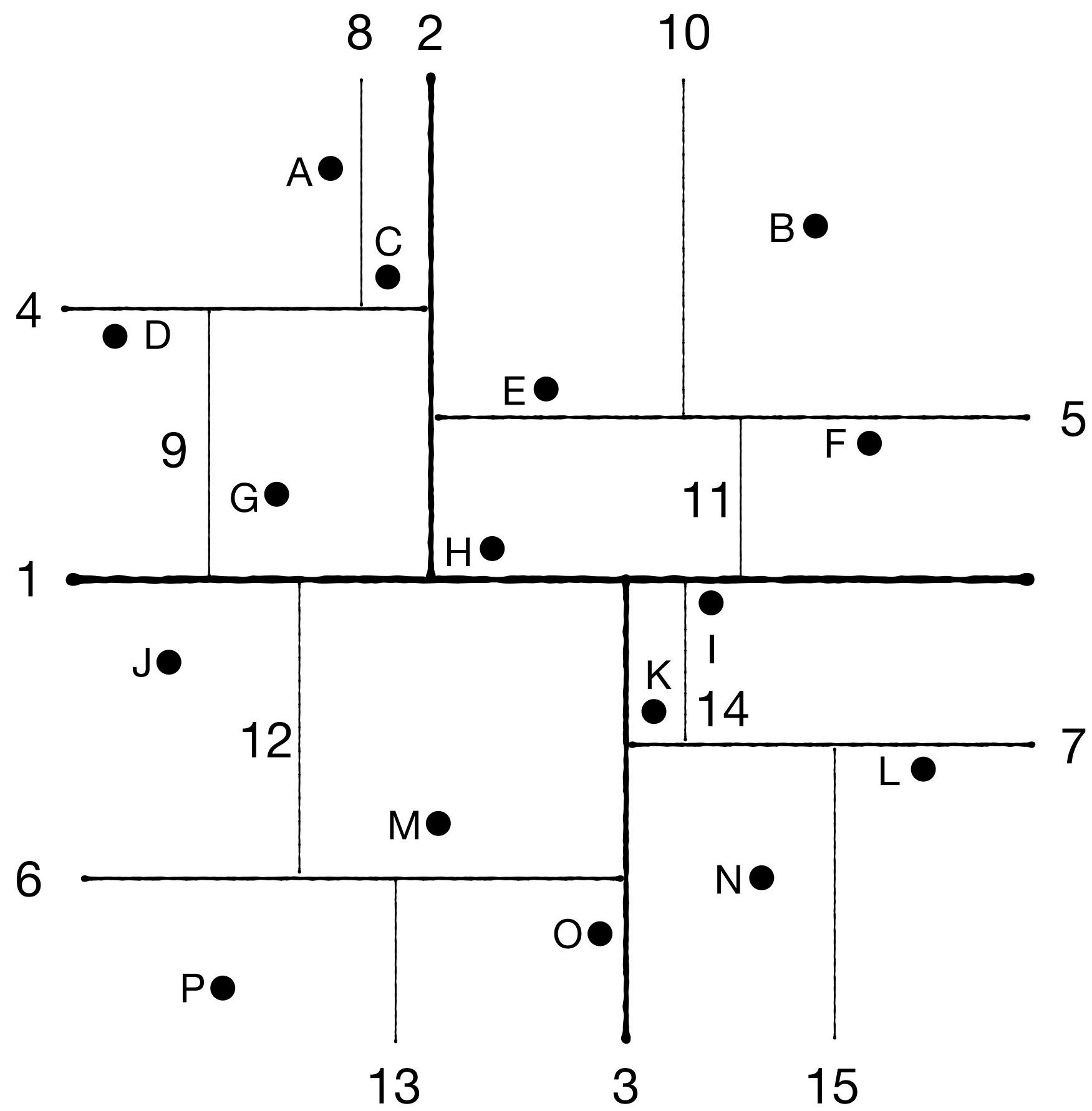




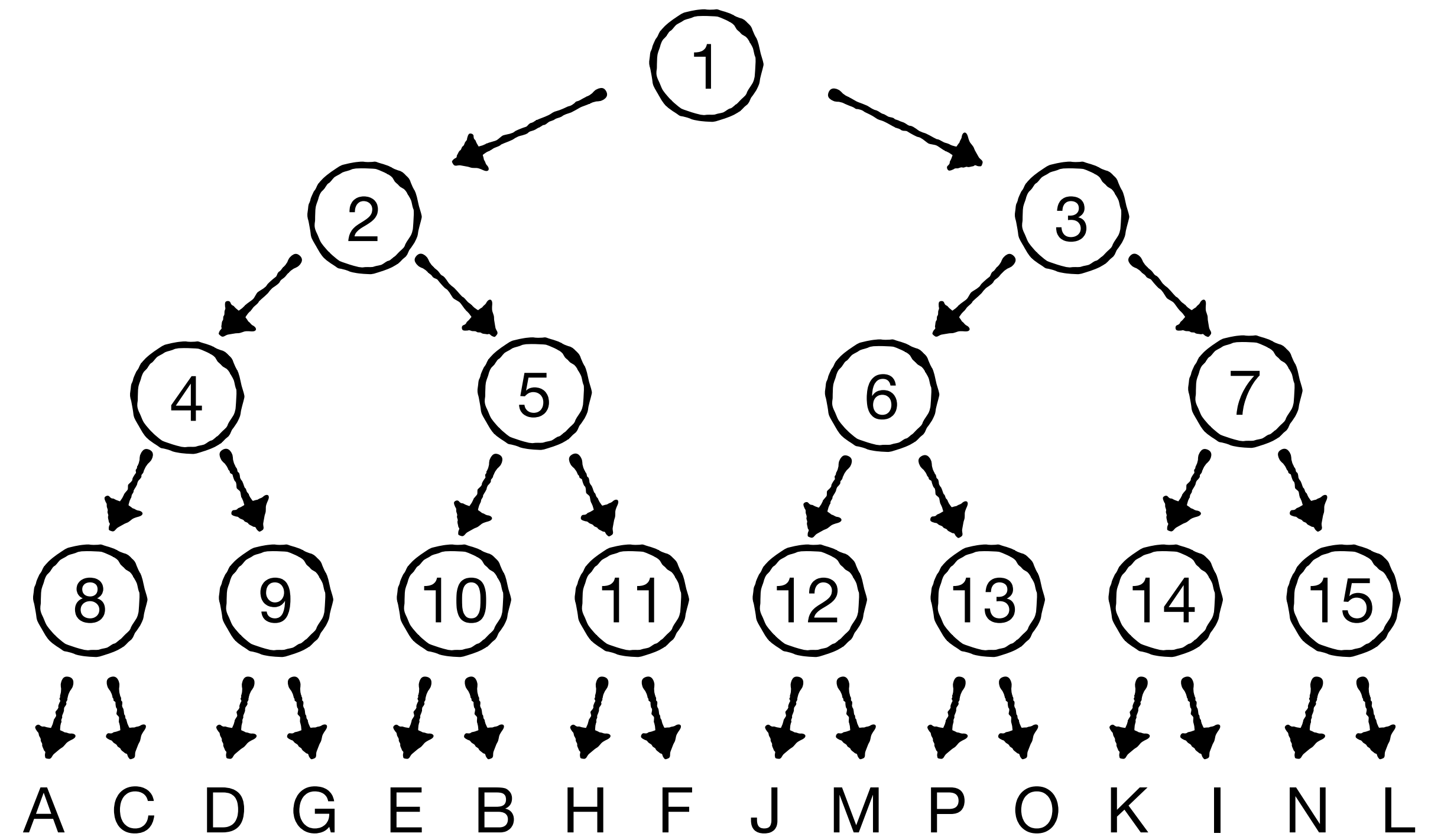
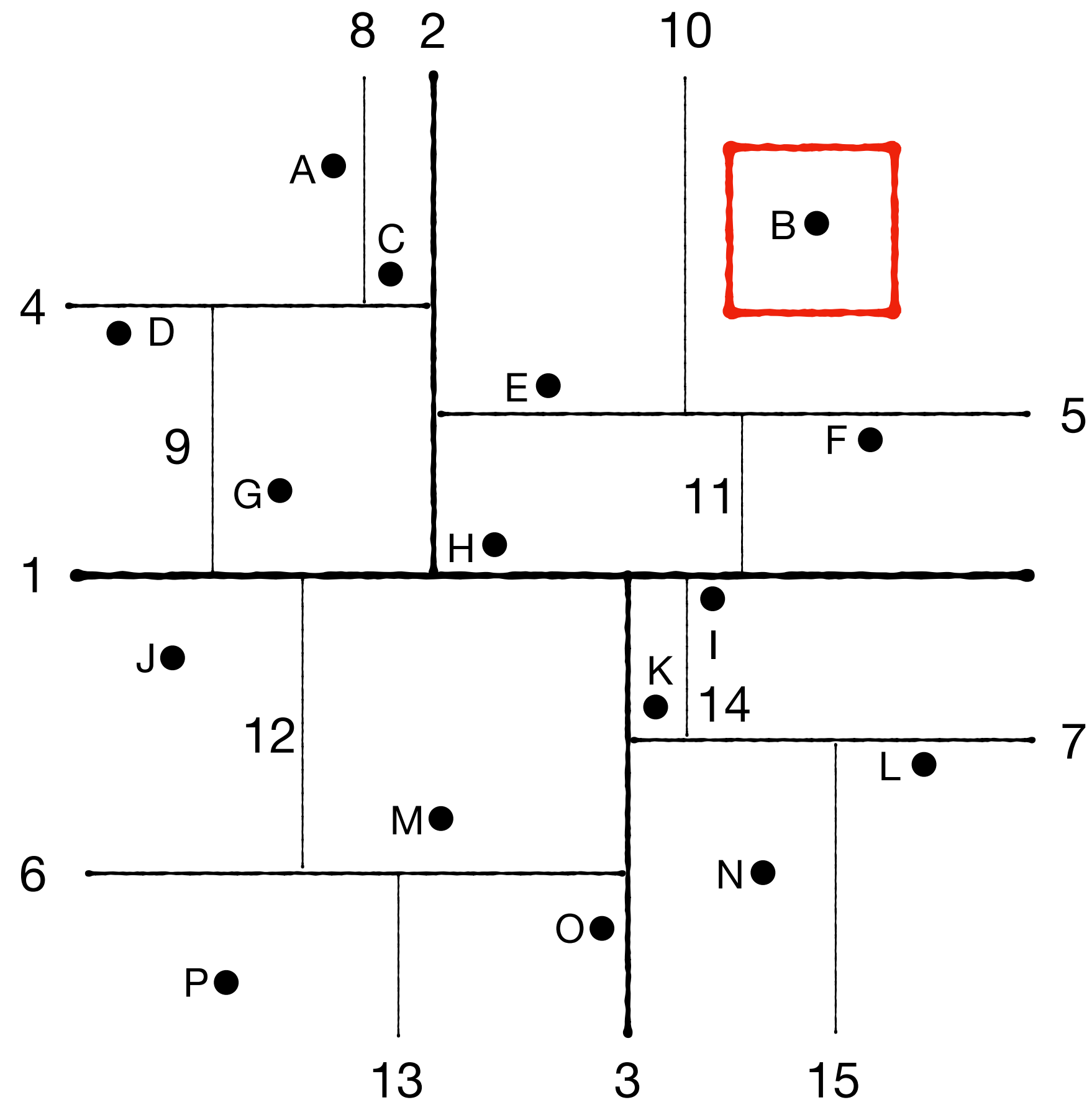




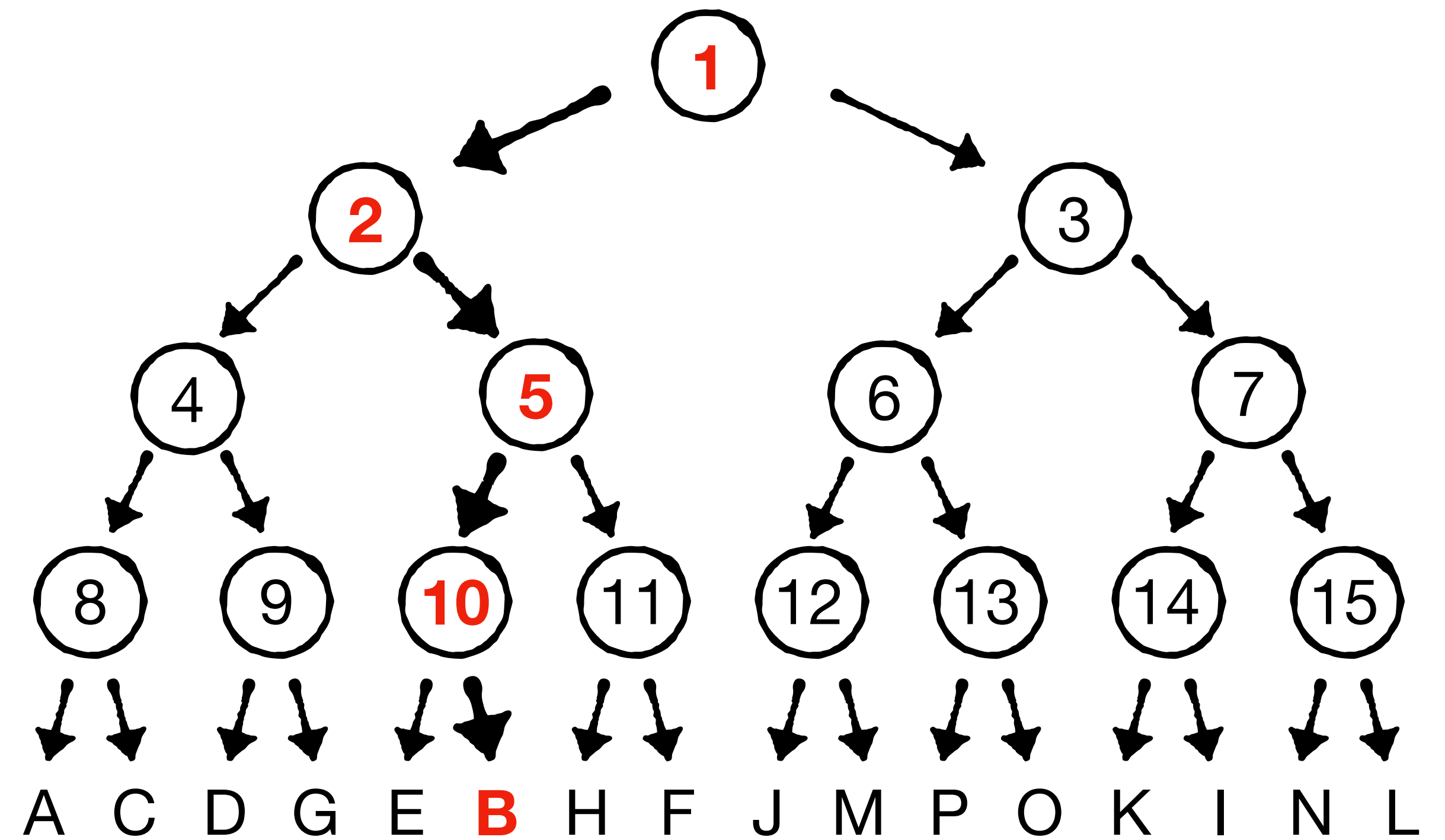
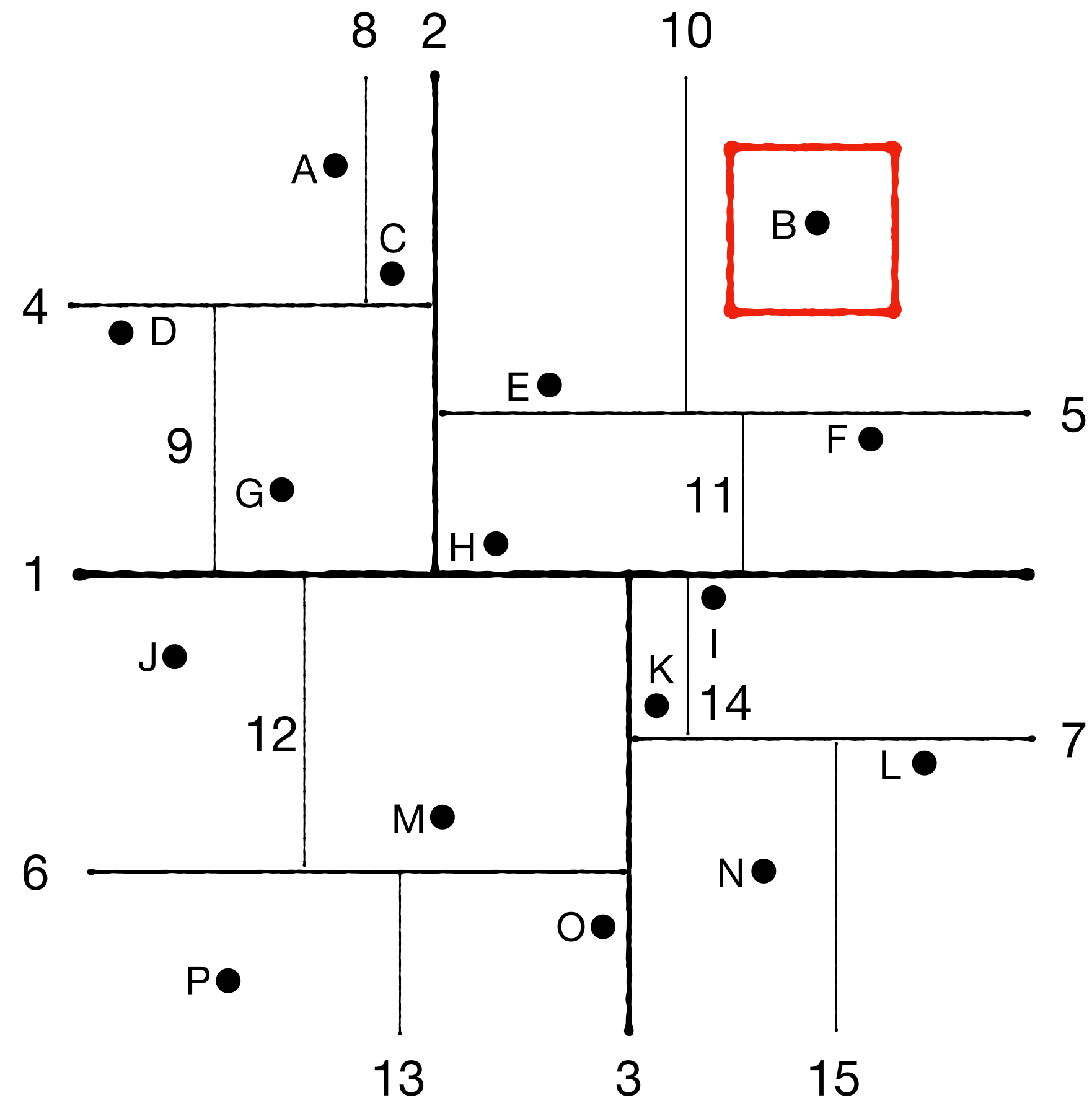




How to process a range query?

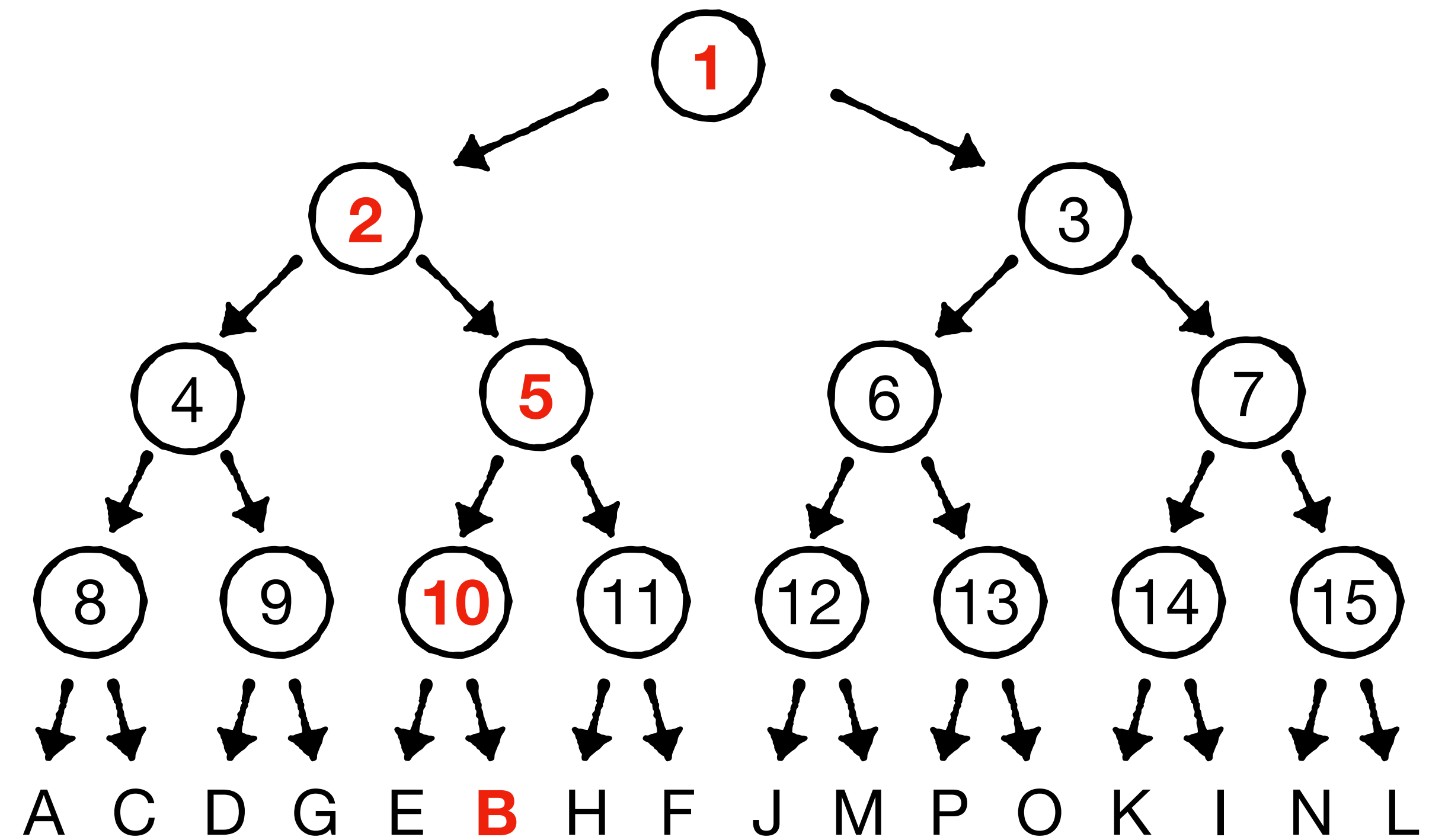
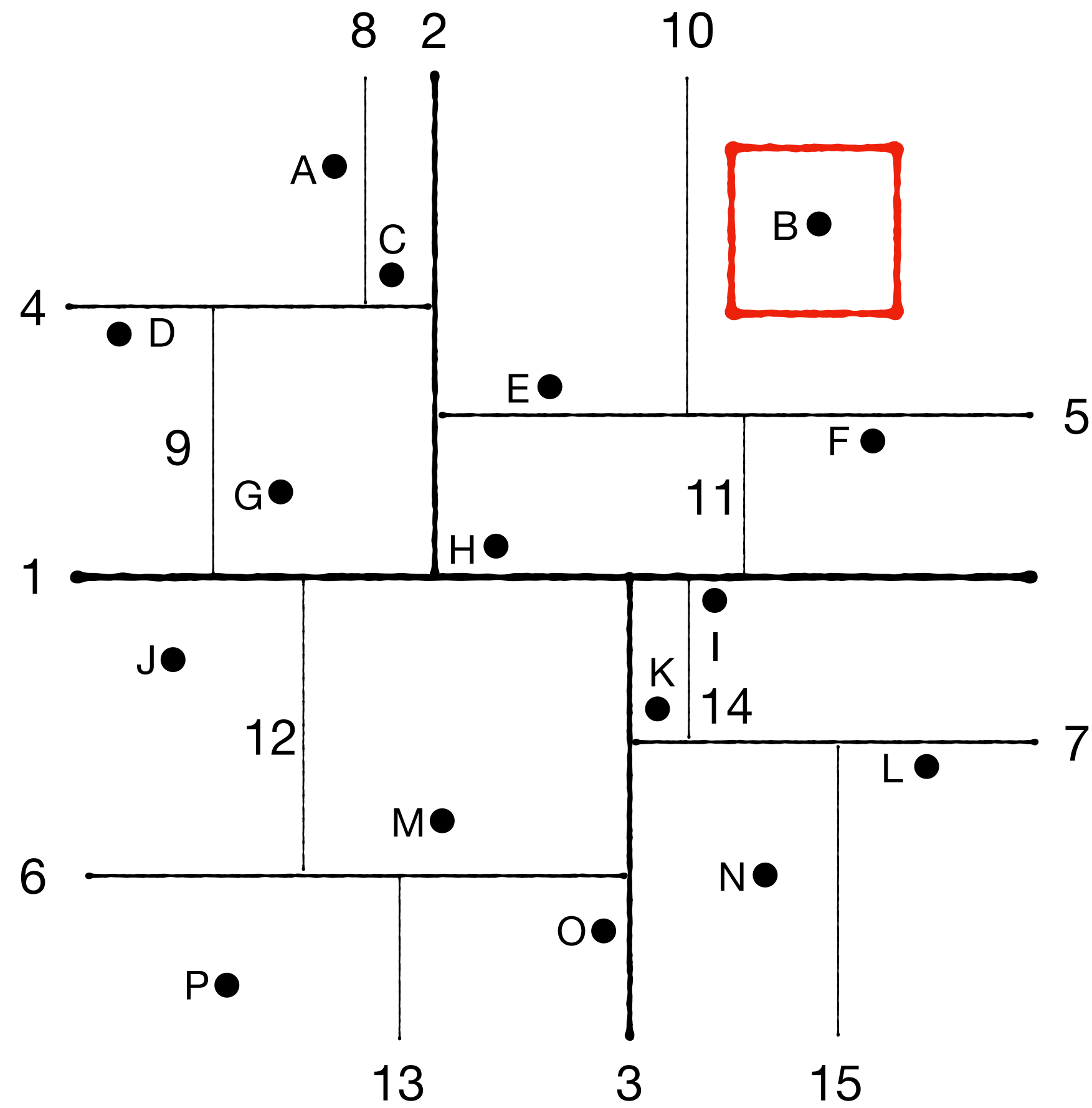


How to process a range query?

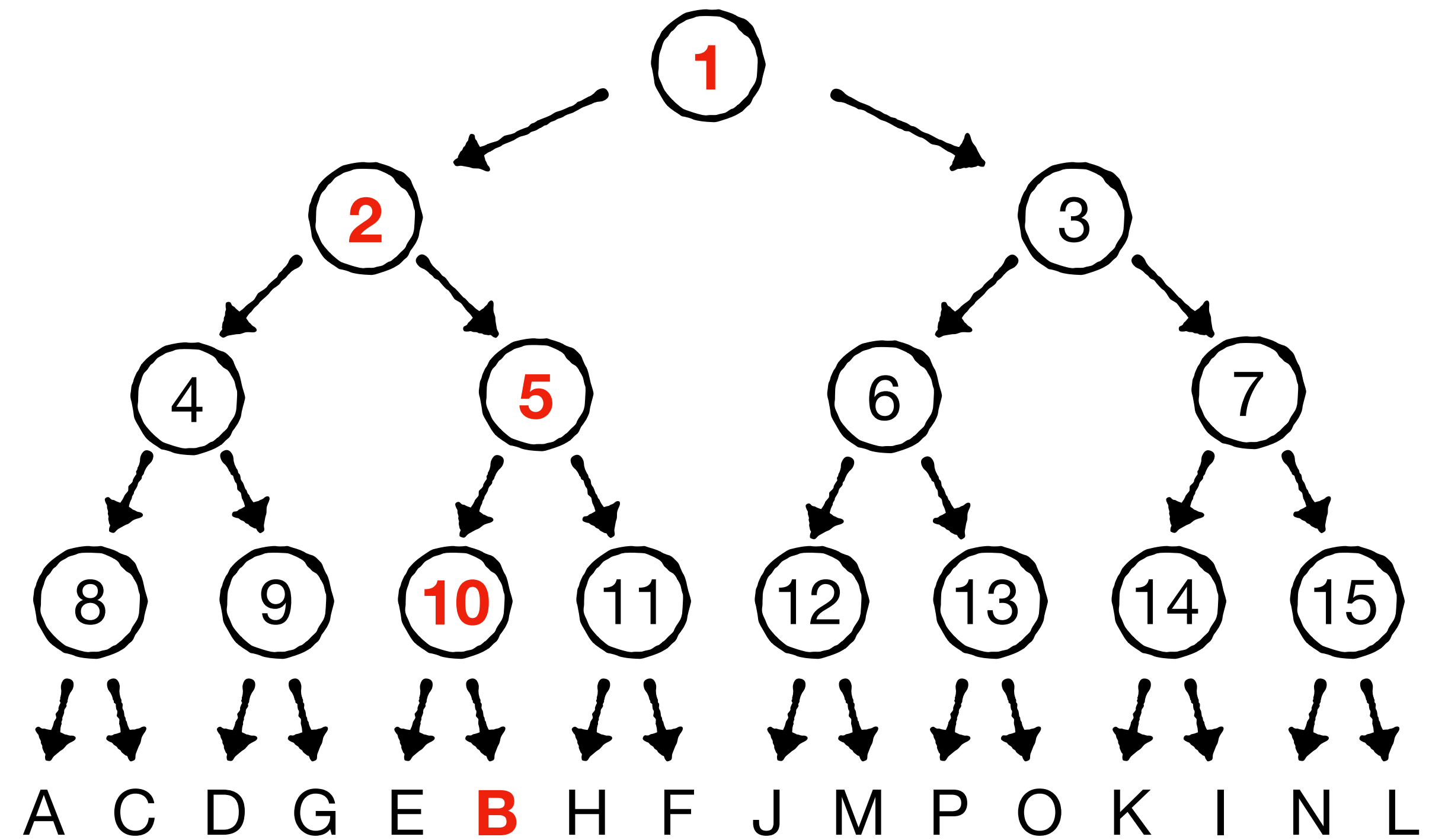
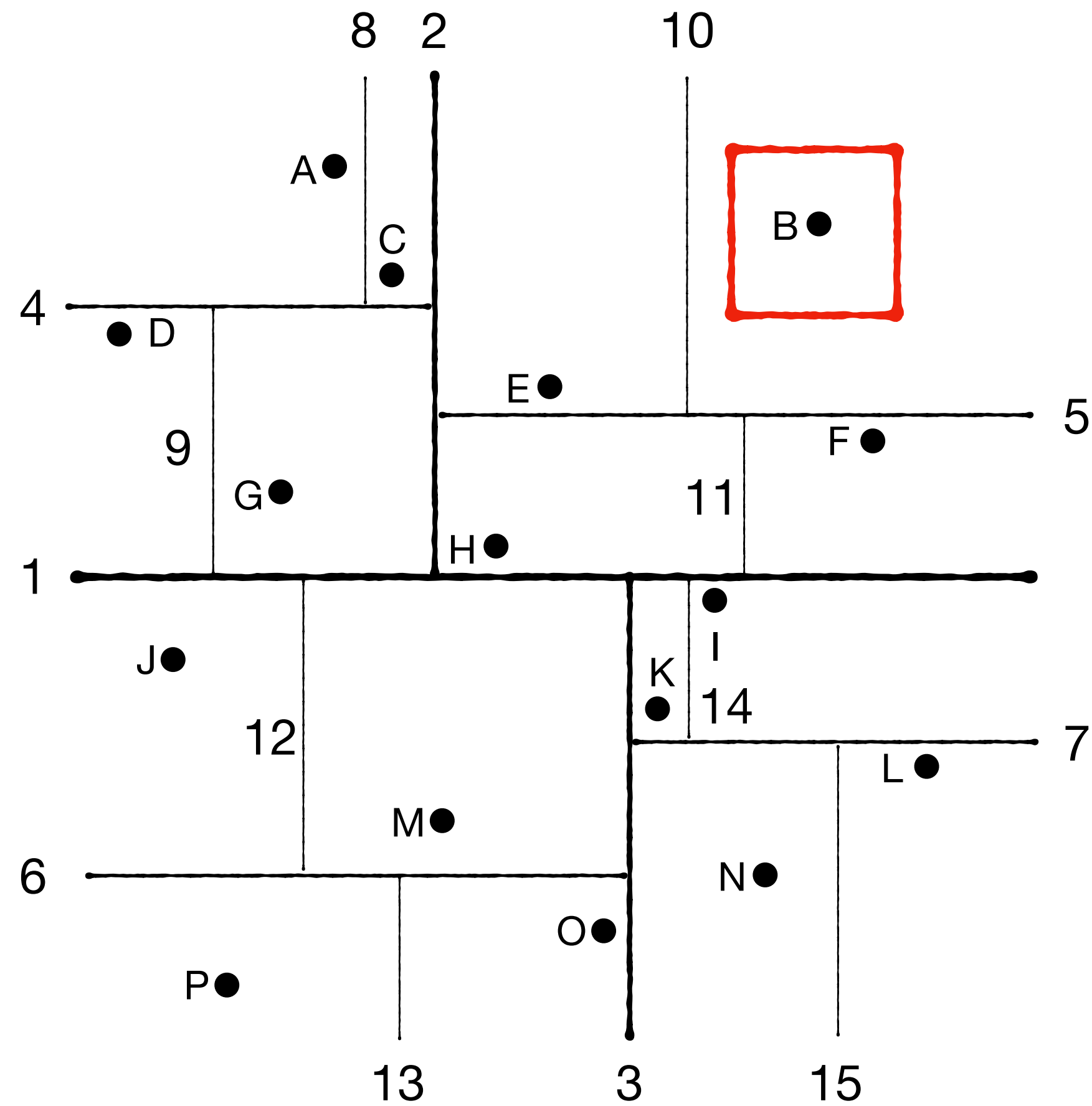


Axis Parallel Range query

Cost?

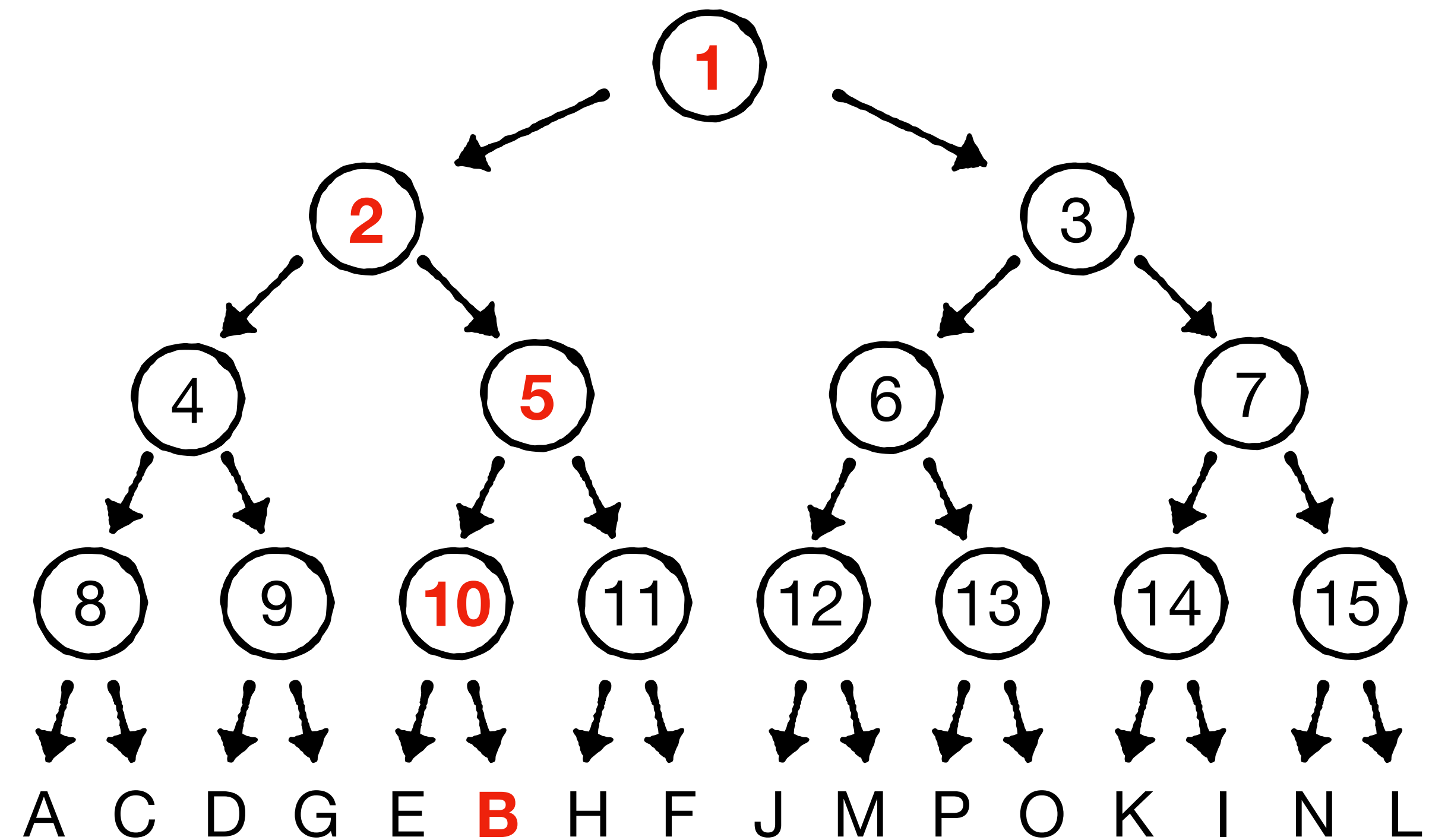
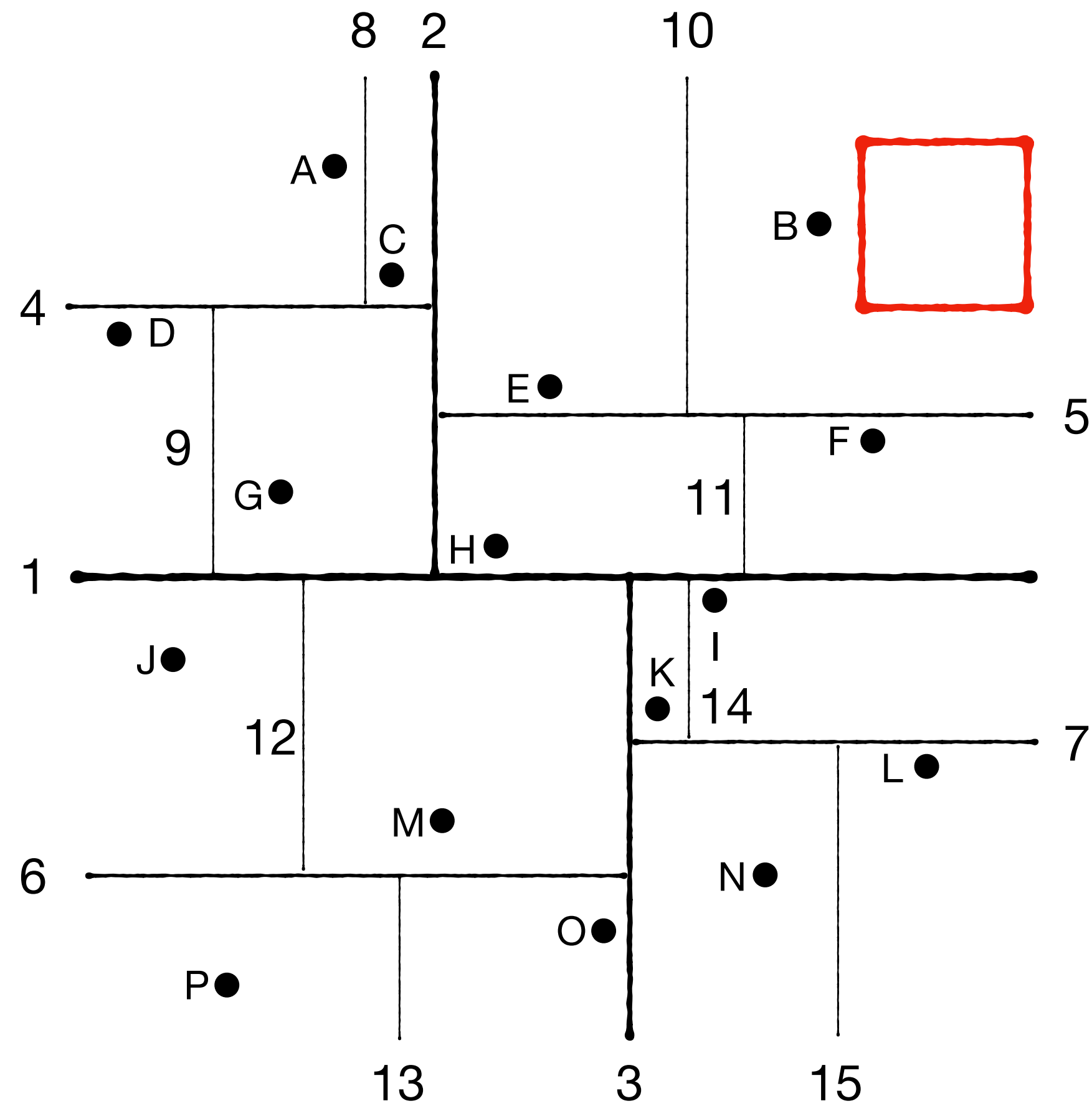


Cost? $O(\log N)$ If query is contained in one region

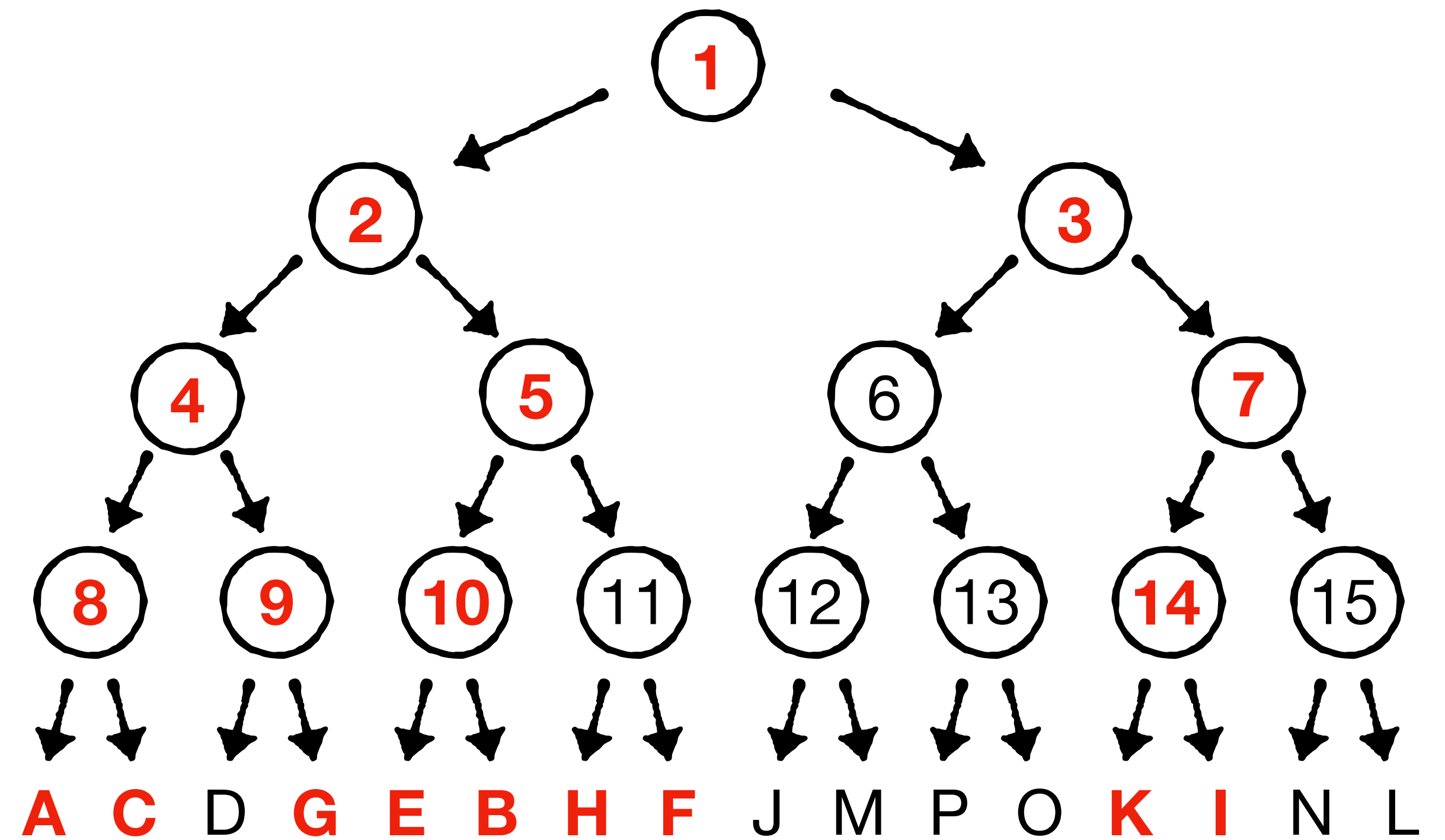
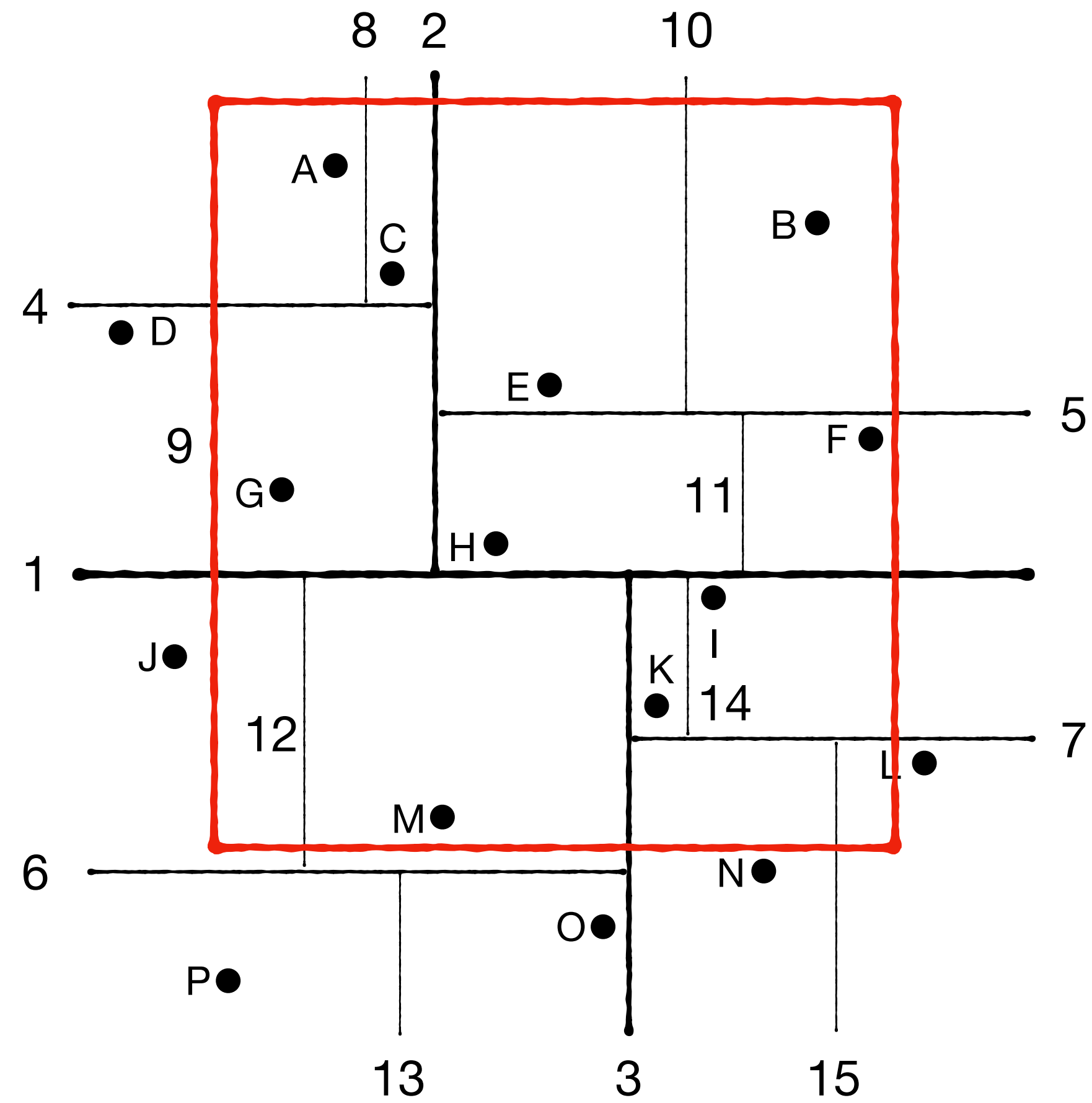


Cost? $O(\log N)$ If query is contained in one region

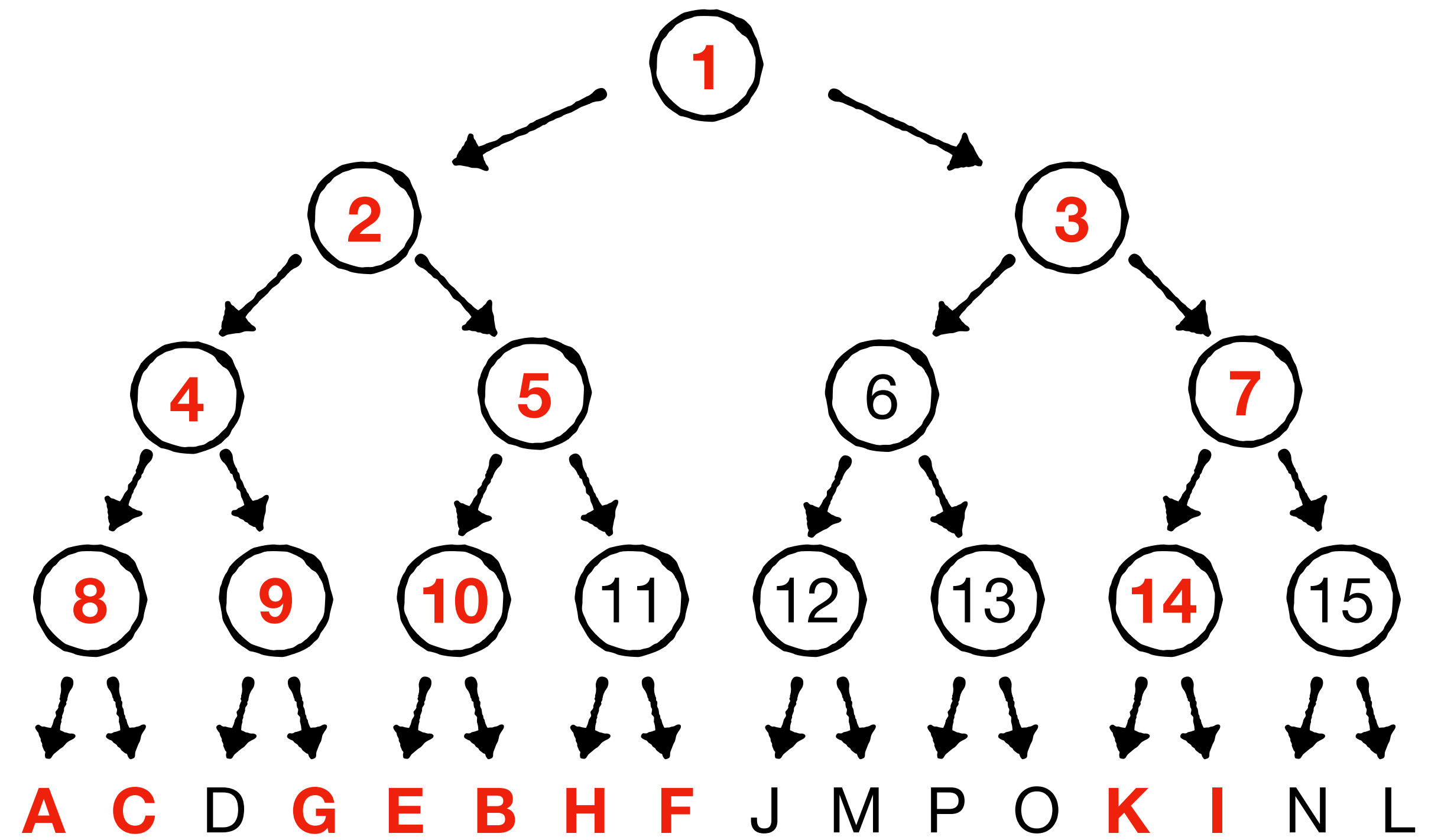
Also applies for empty range query



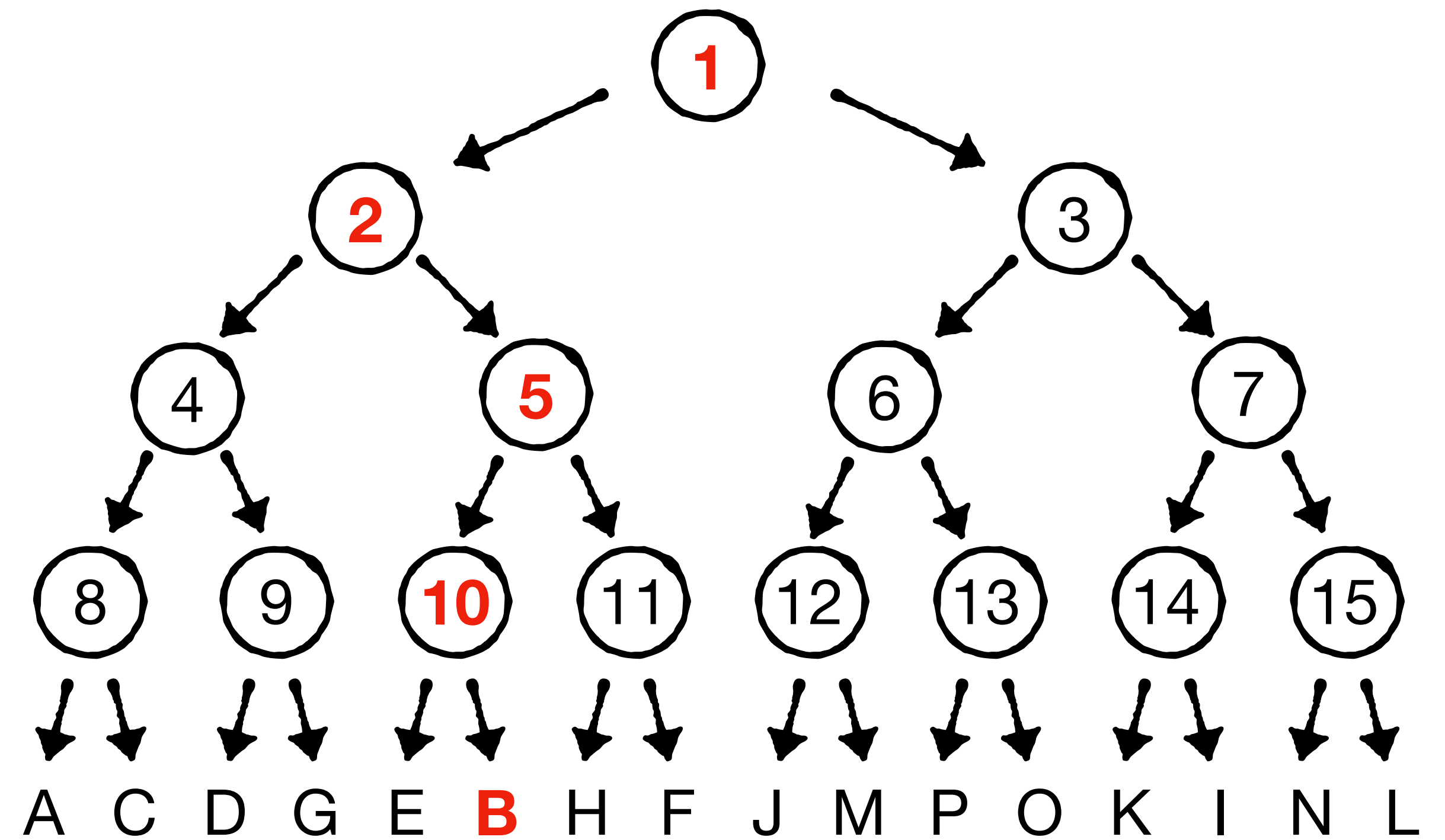
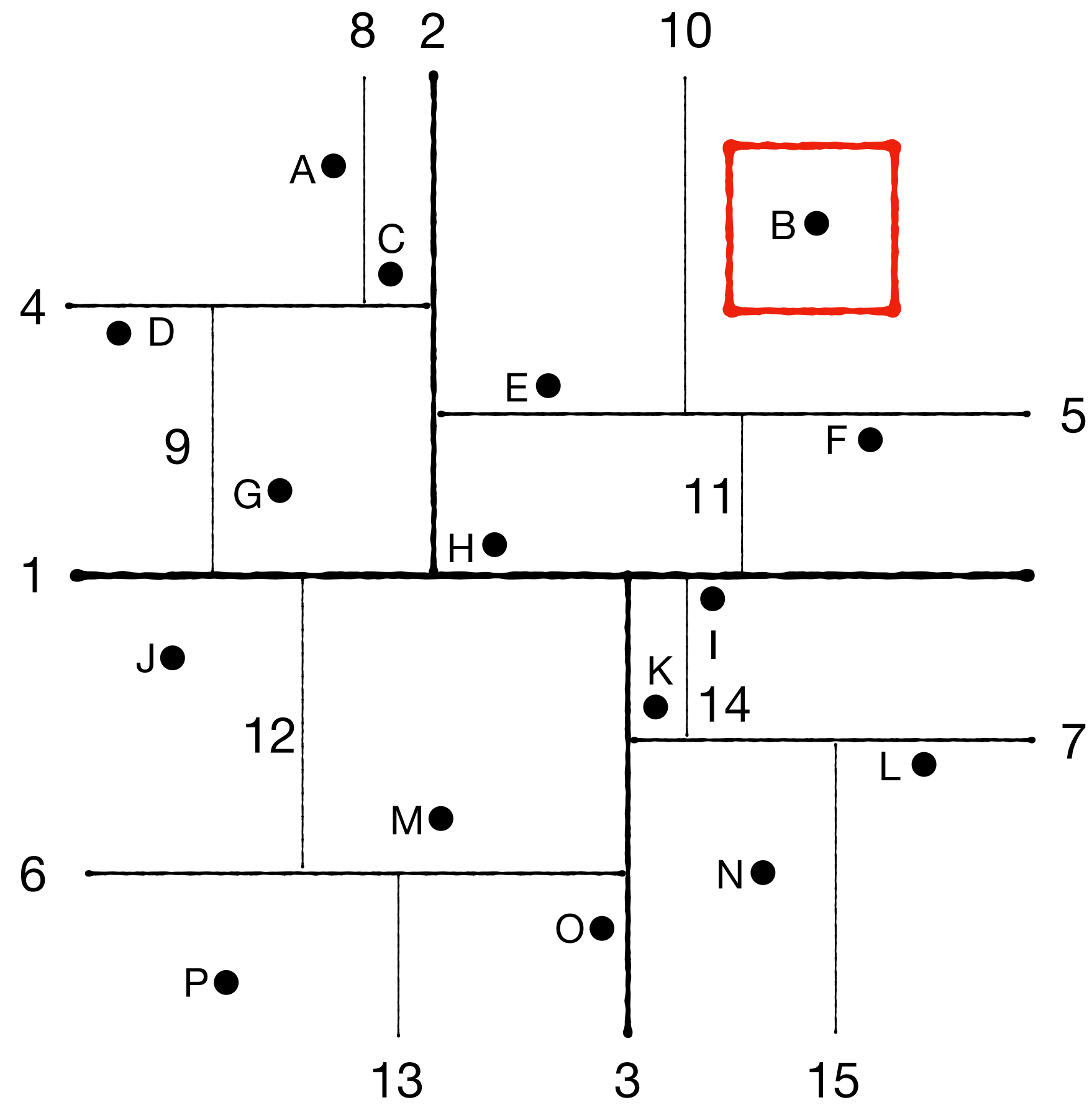
How about a large query?



O(N)

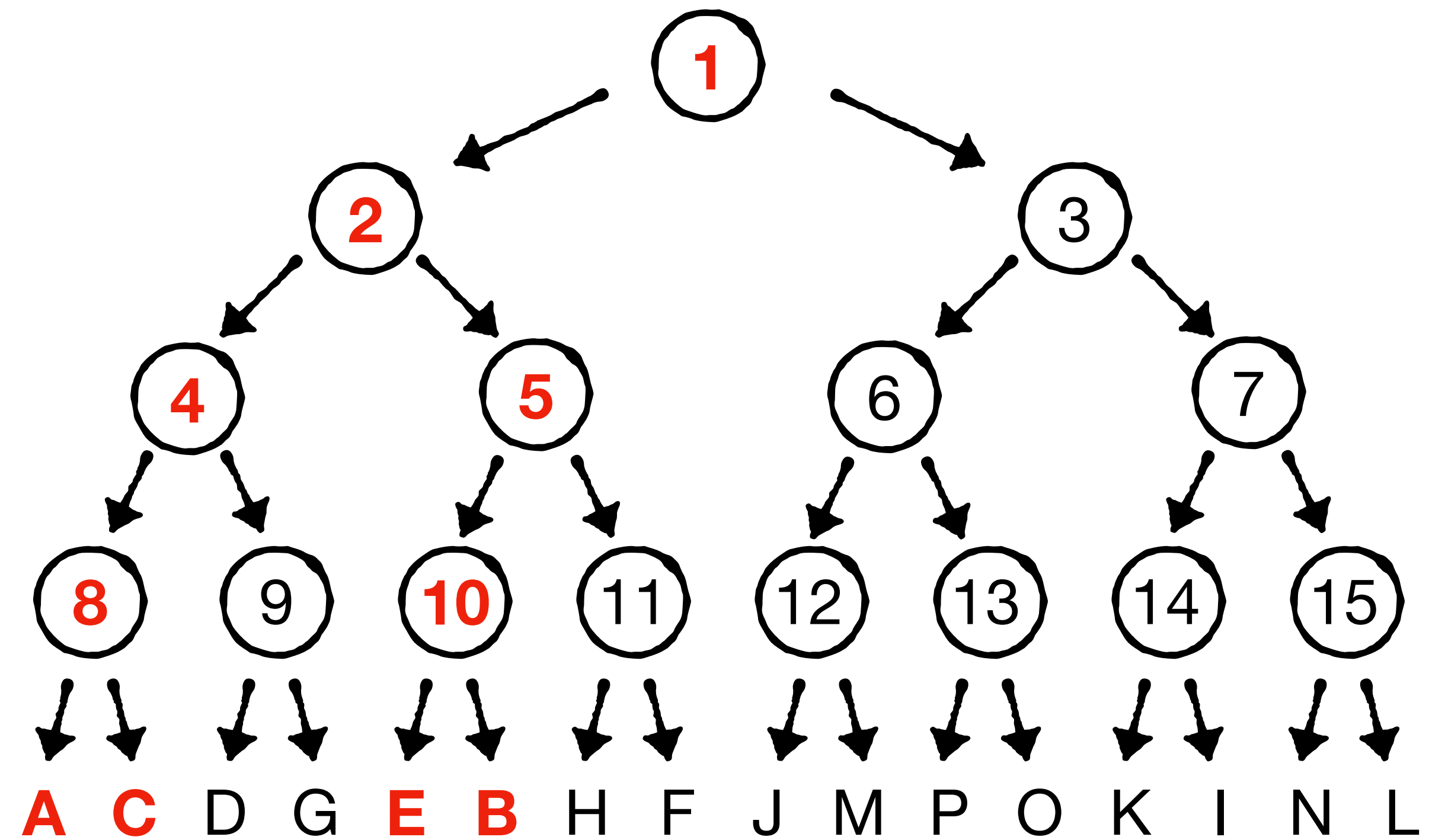
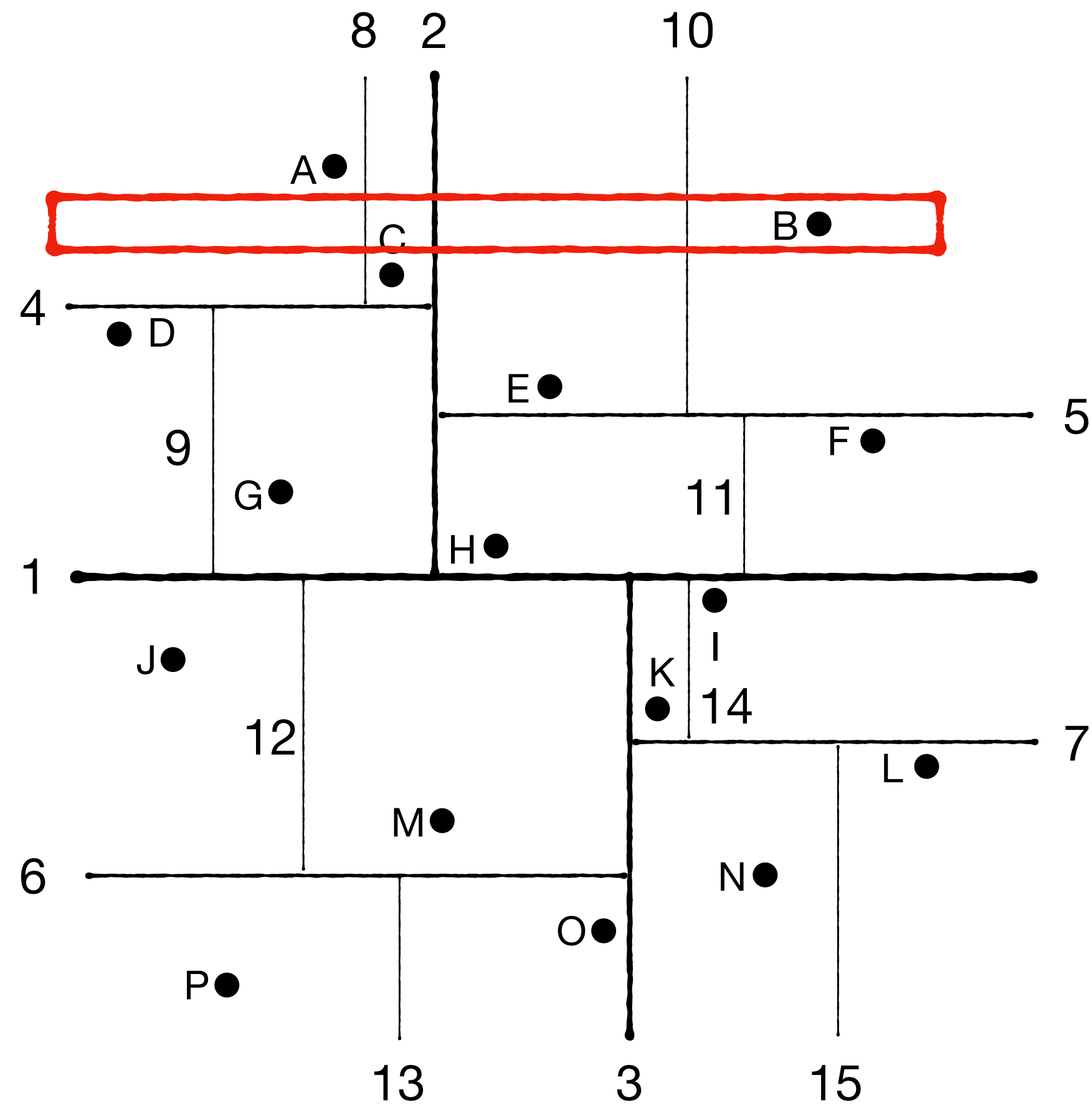


Worst-case for short query? (i.e., one qualifying key)



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

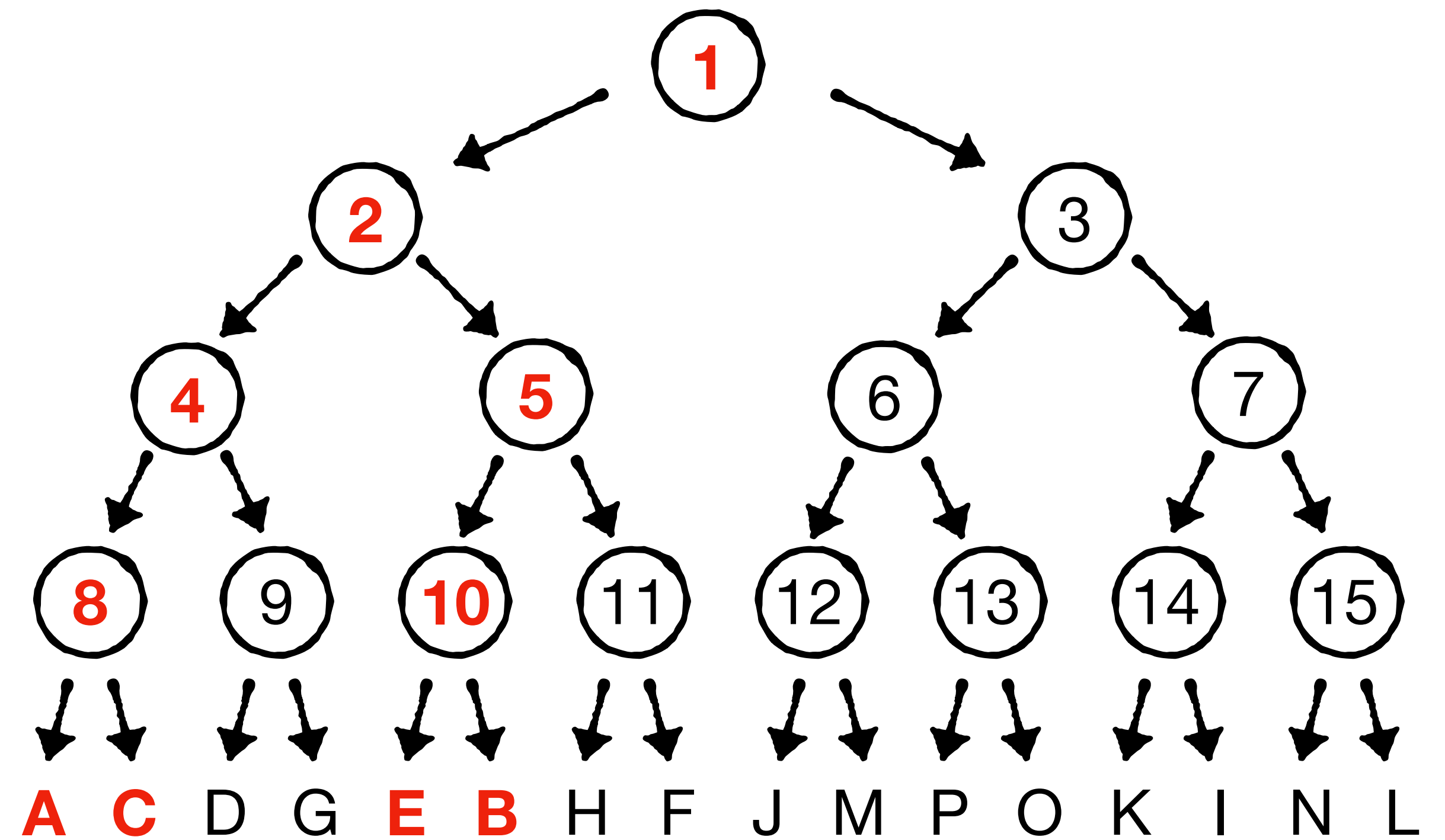
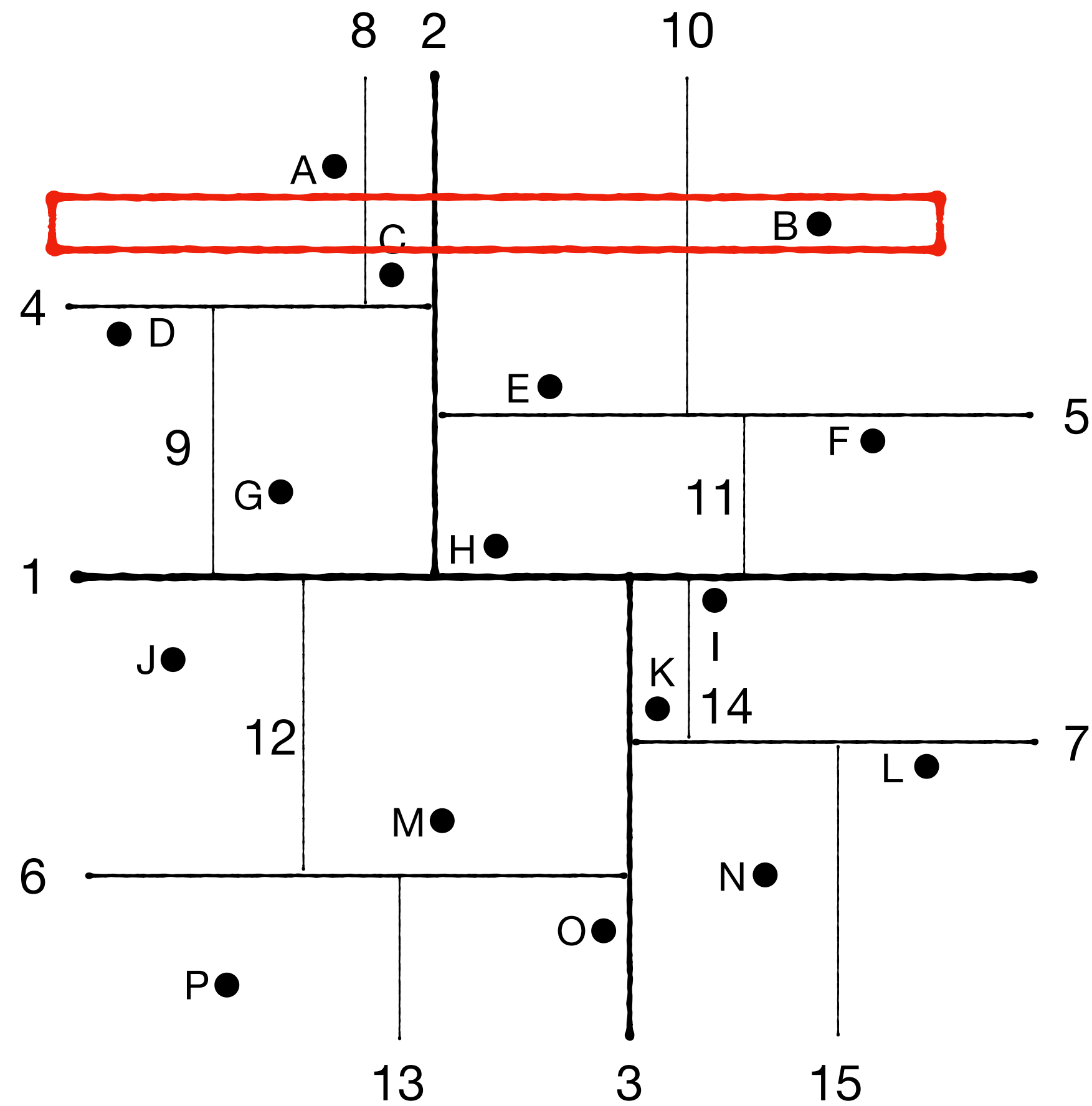
Maximize intersections across regions



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

Maximize intersections across regions

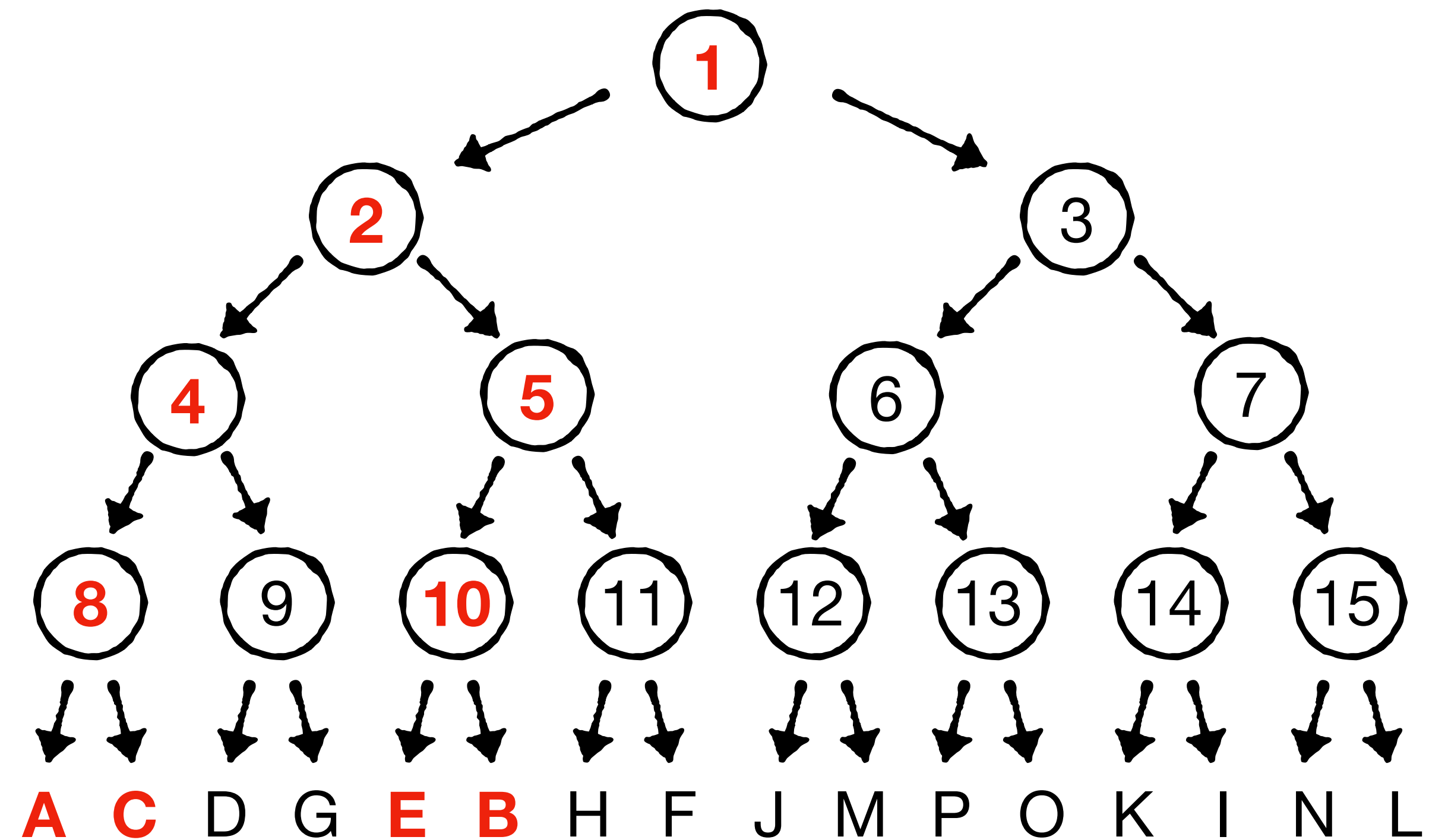
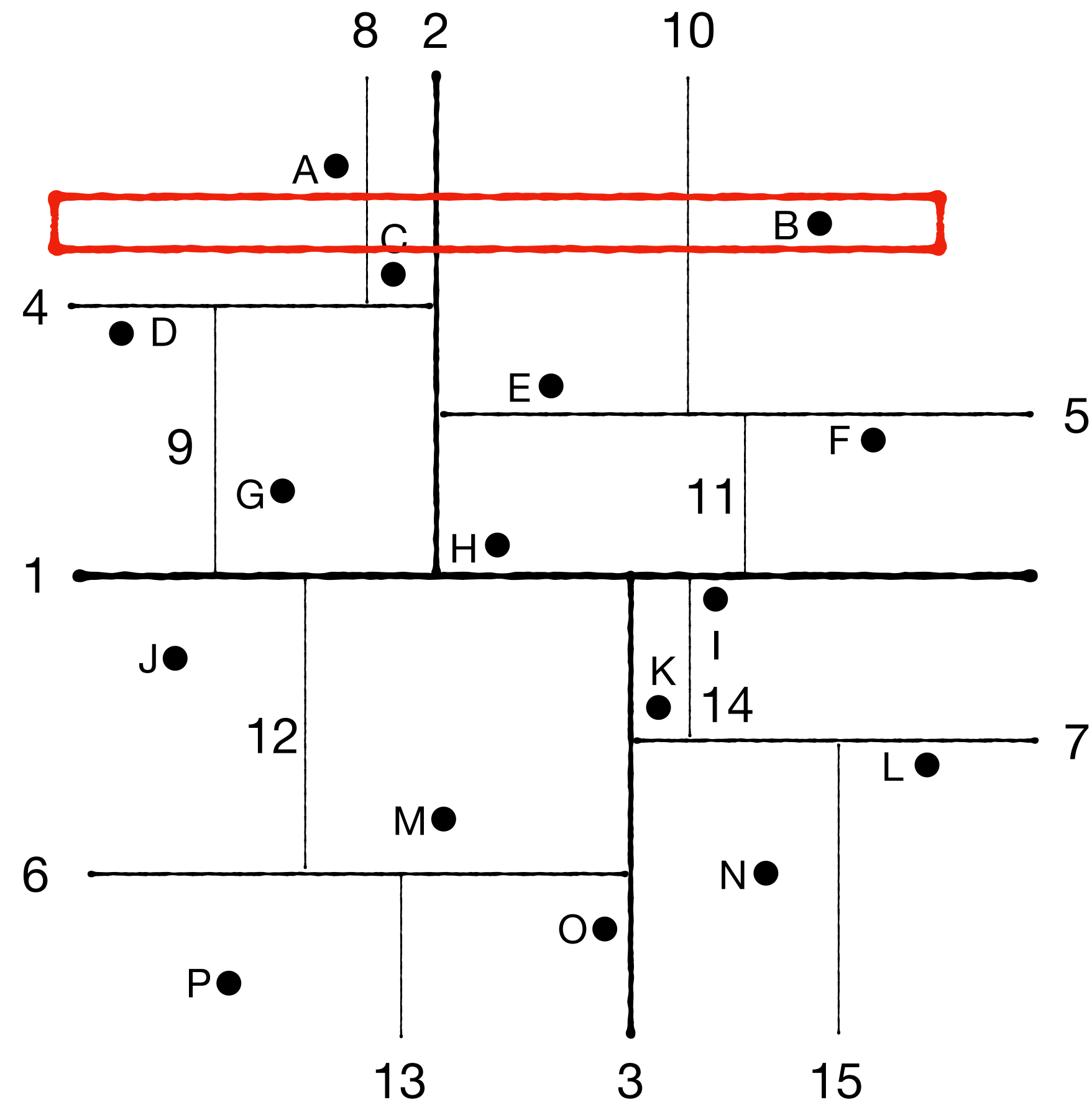
Cost?



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

Maximize intersections across regions

Cost?

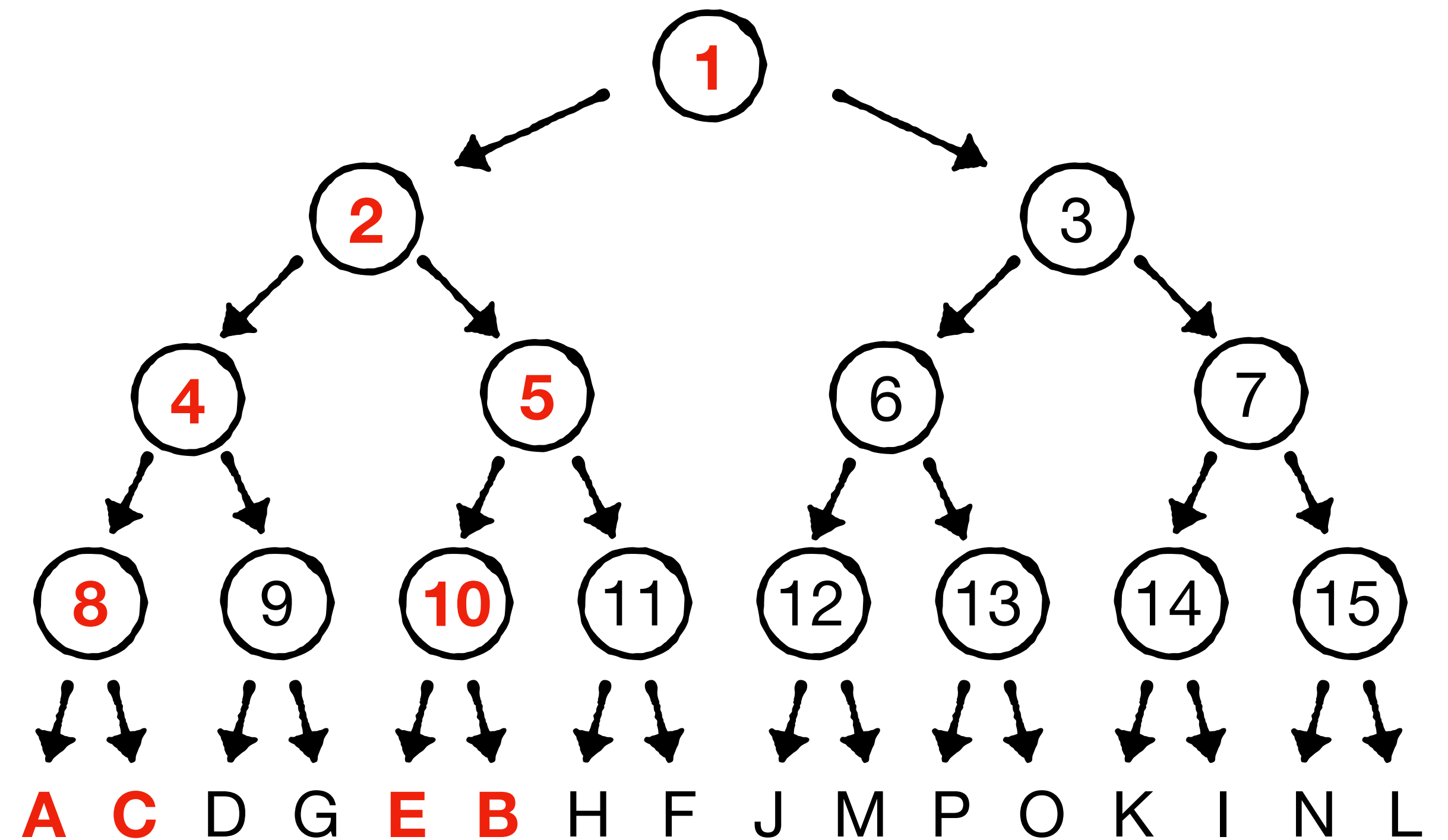
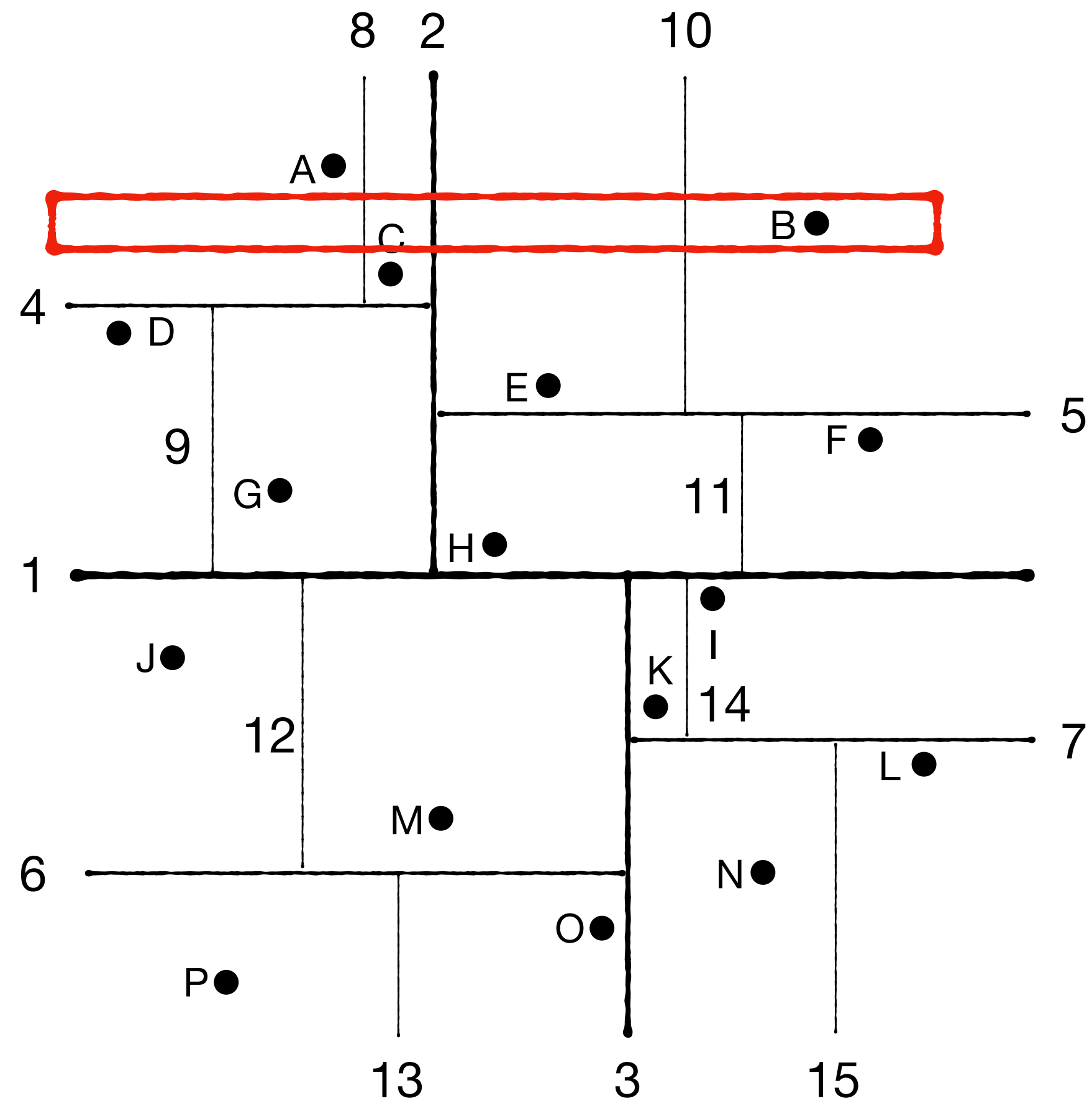


Max regions along one dimension?

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

Maximize intersections across regions

Cost?

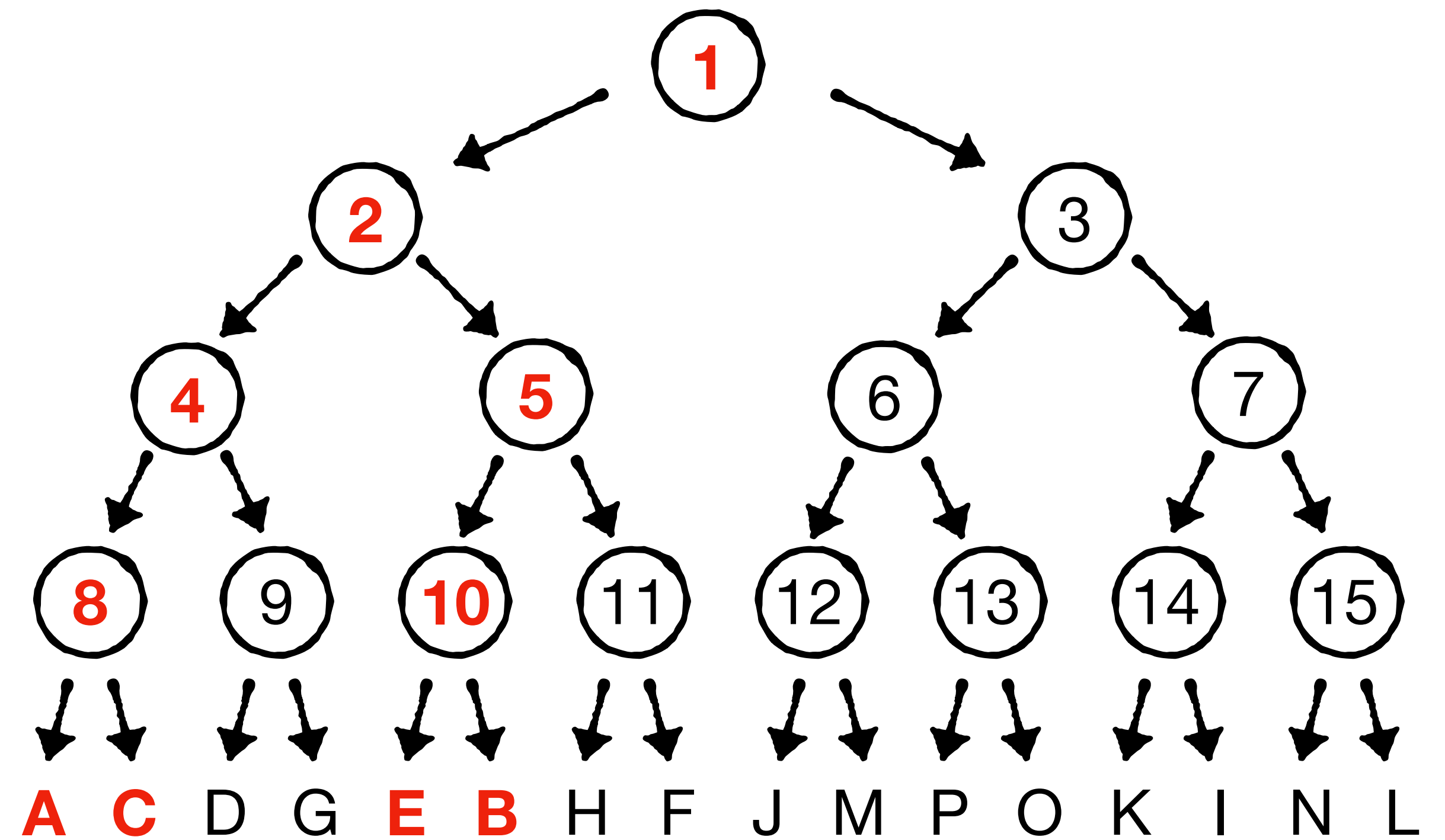
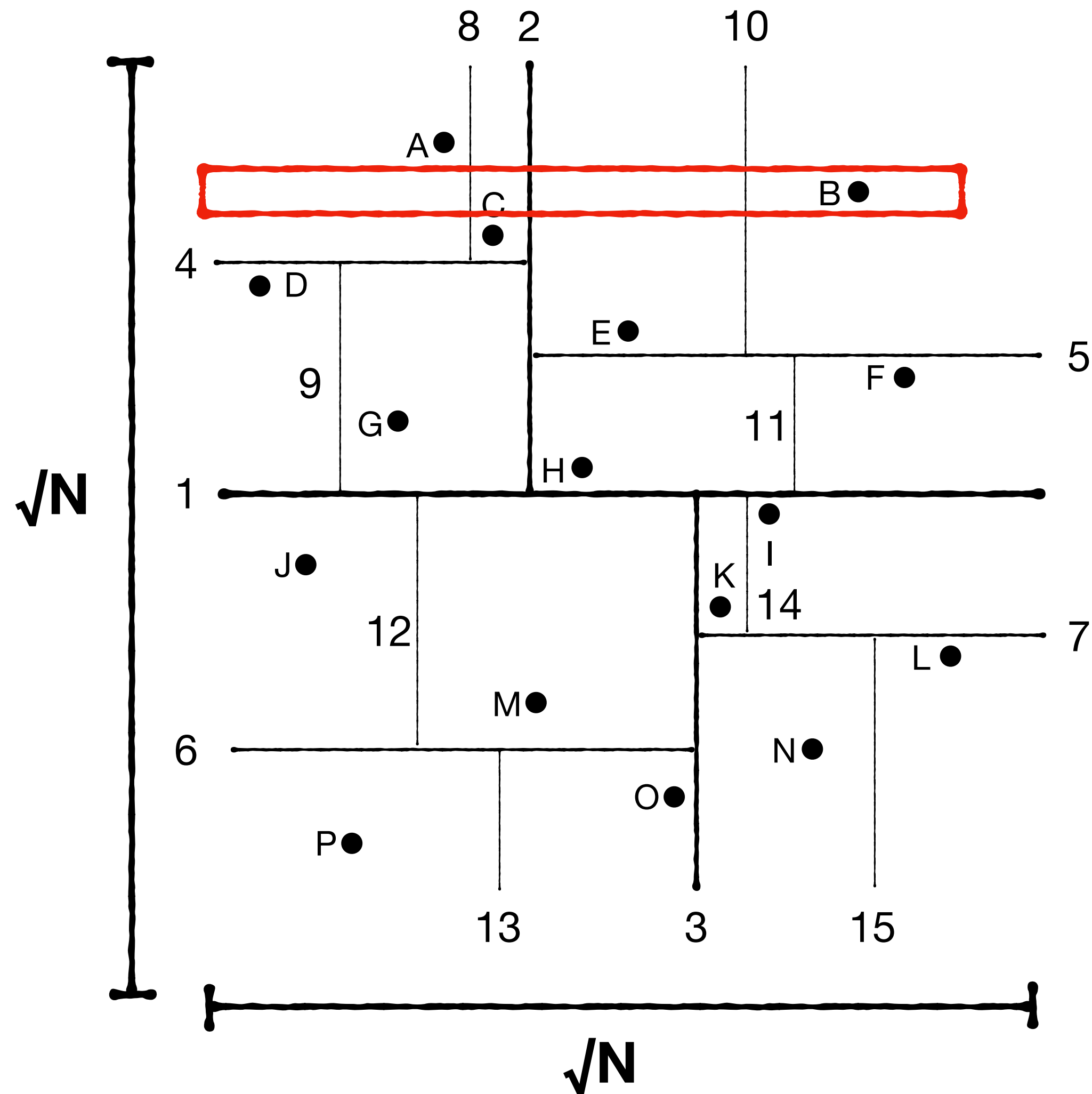


Max regions along one dimension? $\sqrt{N}$

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

Maximize intersections across regions

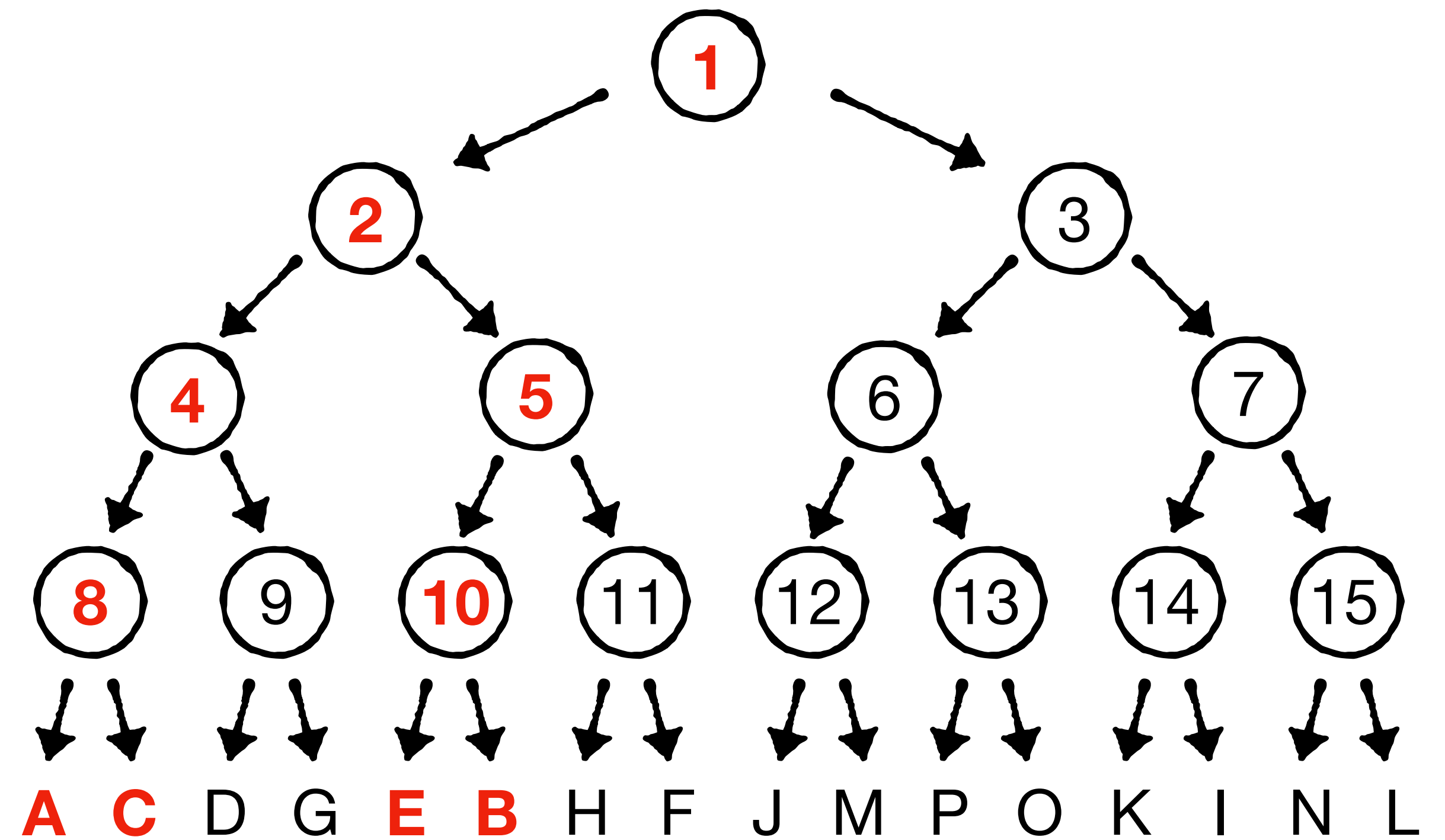
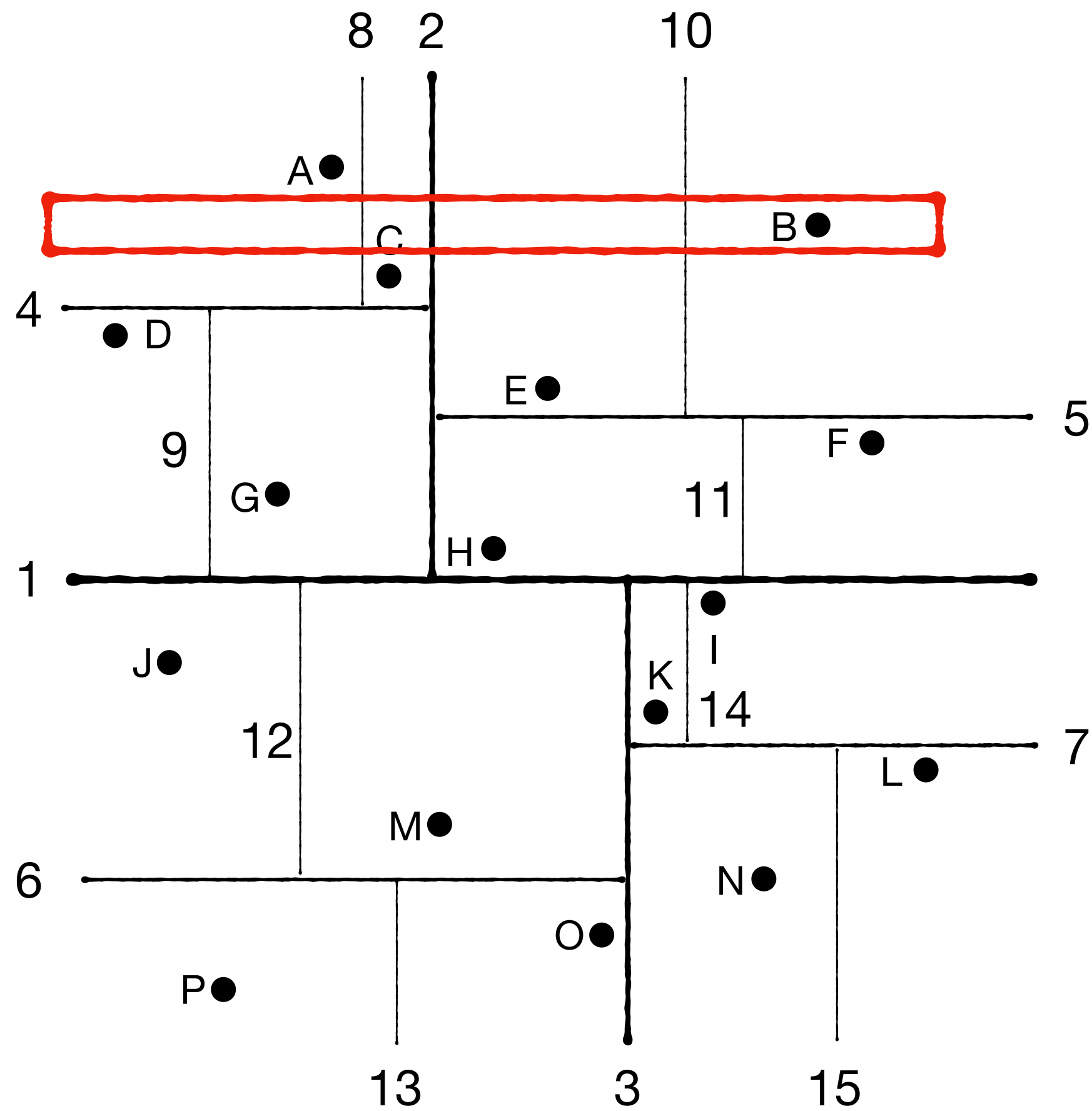
Cost?



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

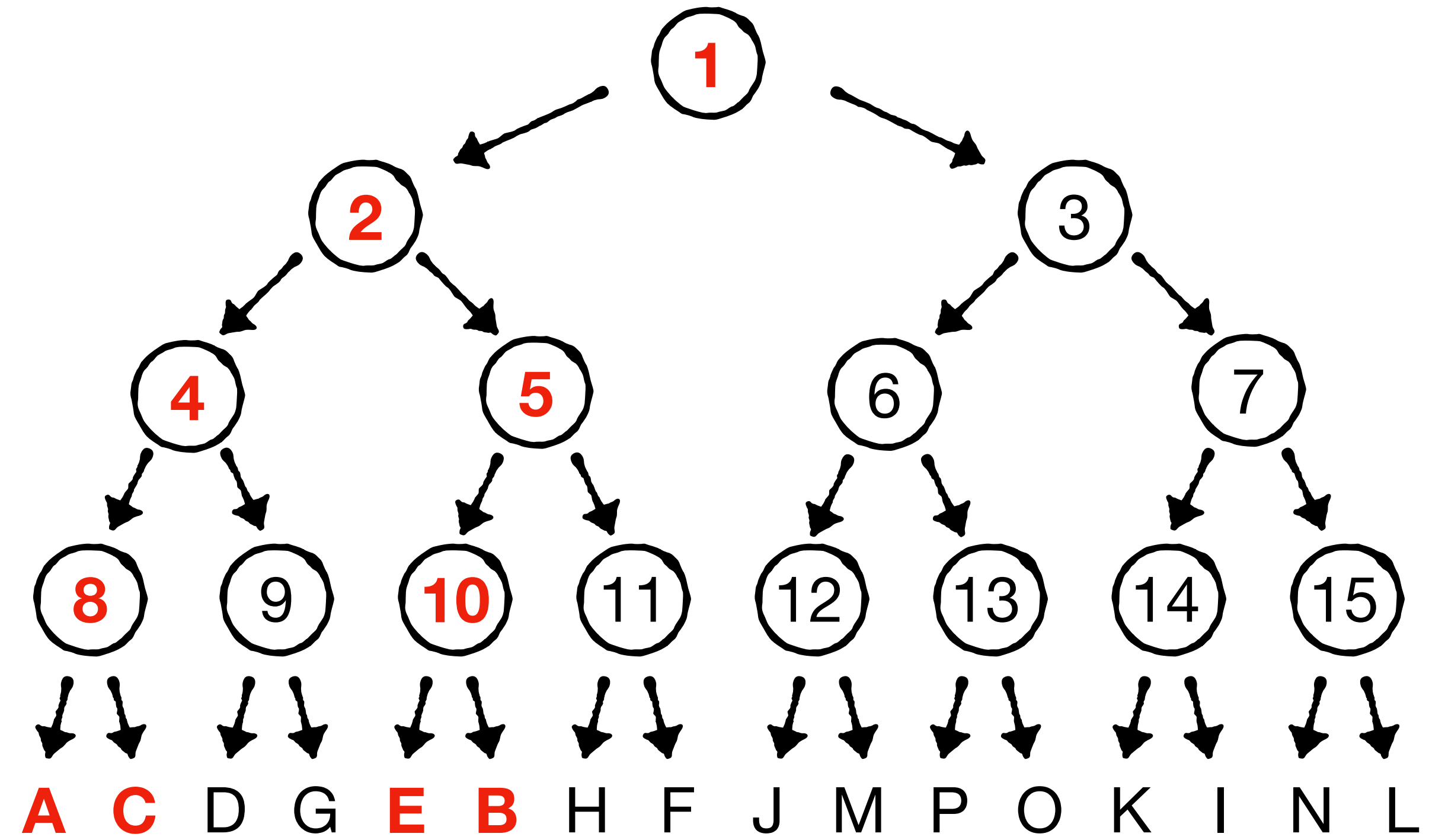
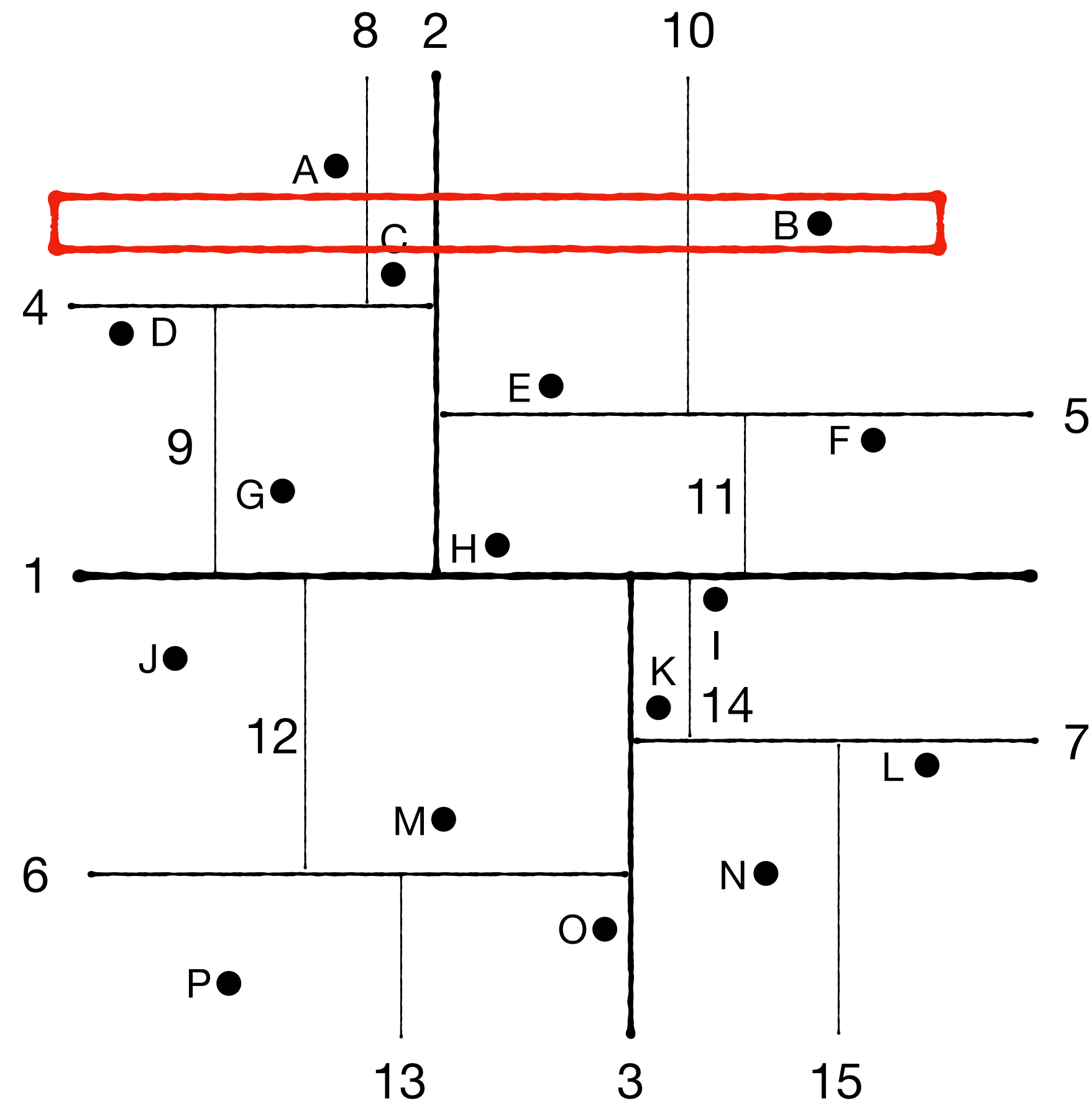
Maximize intersections across regions

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



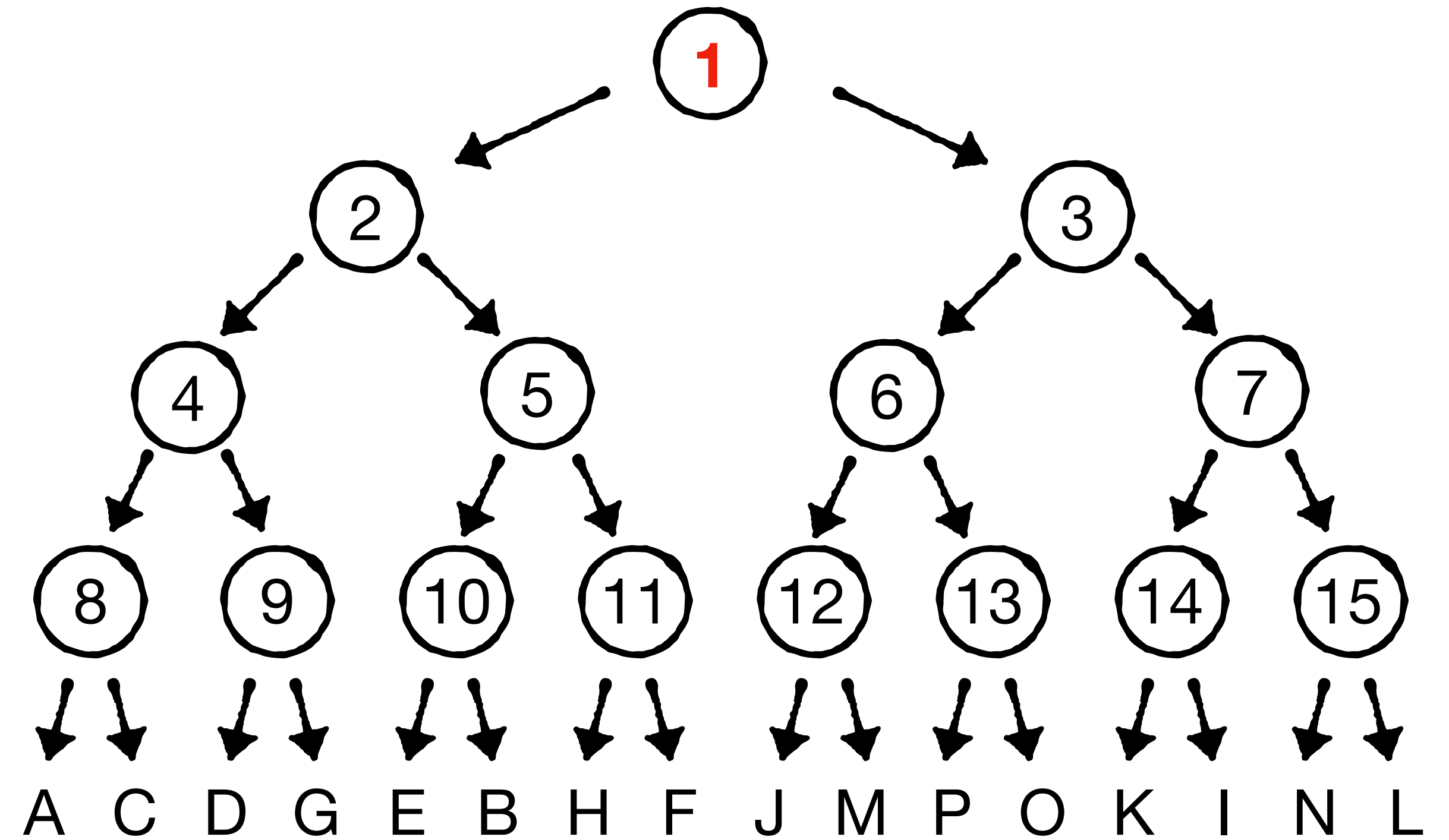
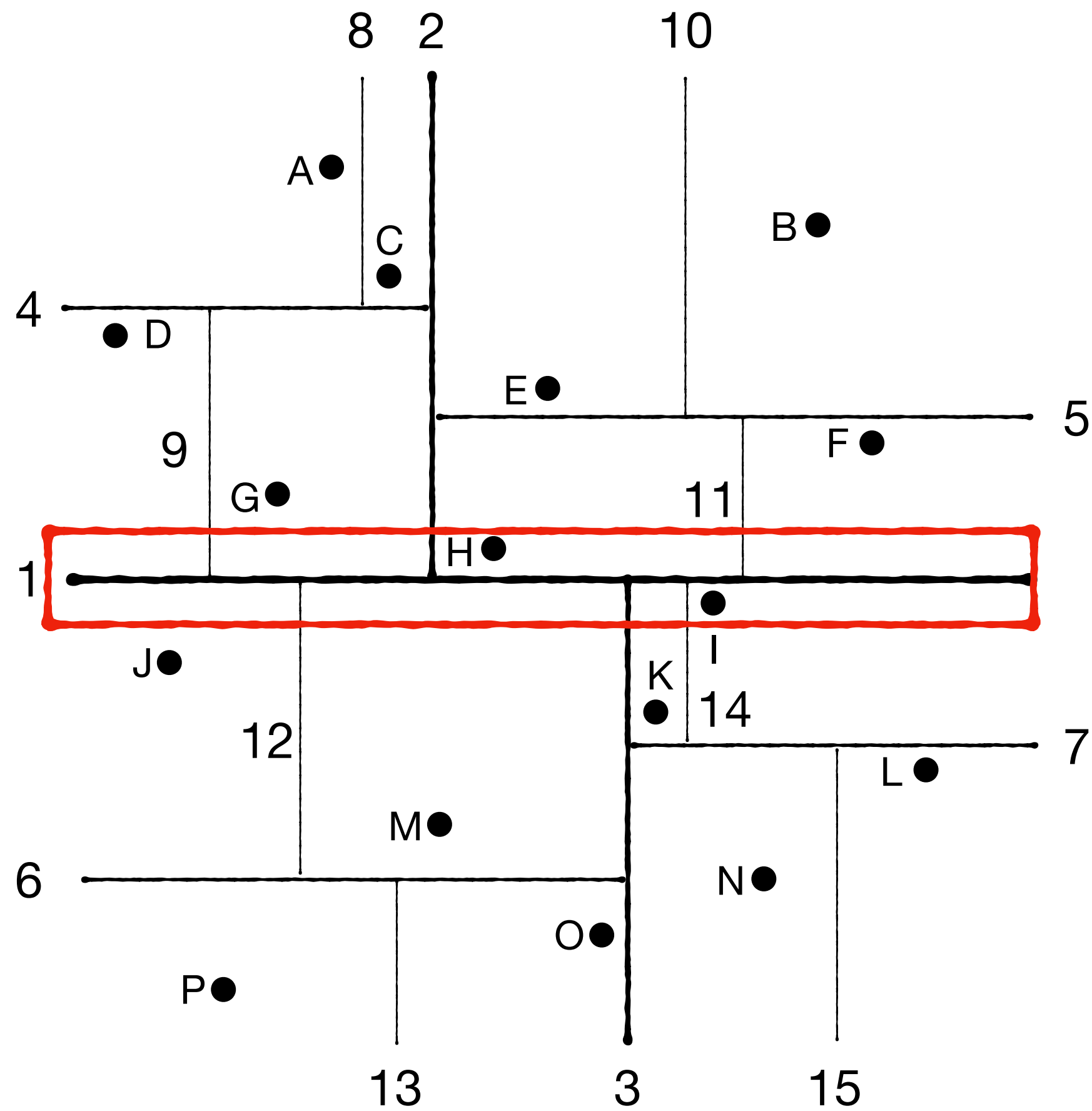
Cost? $O(\sqrt{N})$

Can we make this even worse?



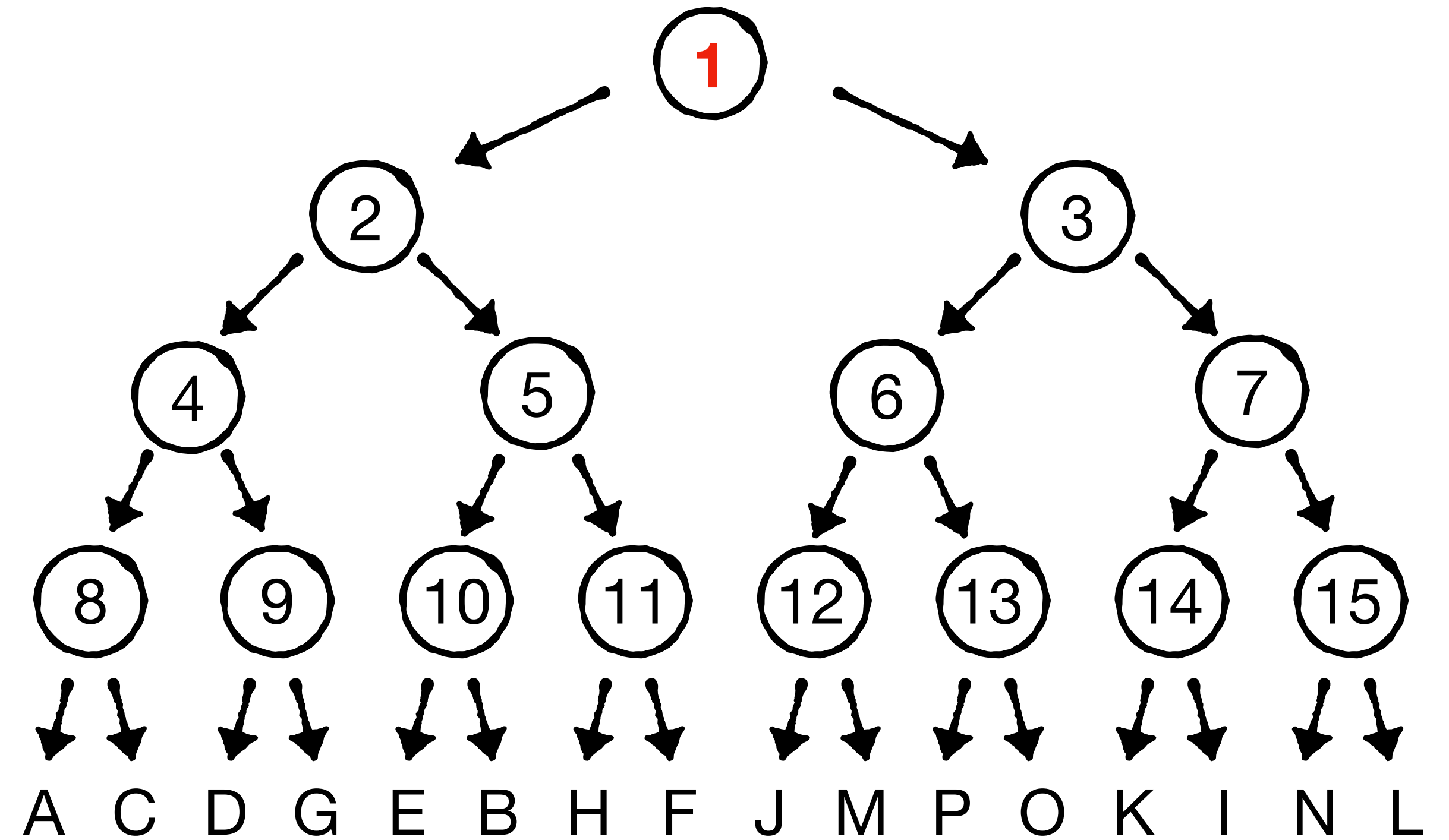
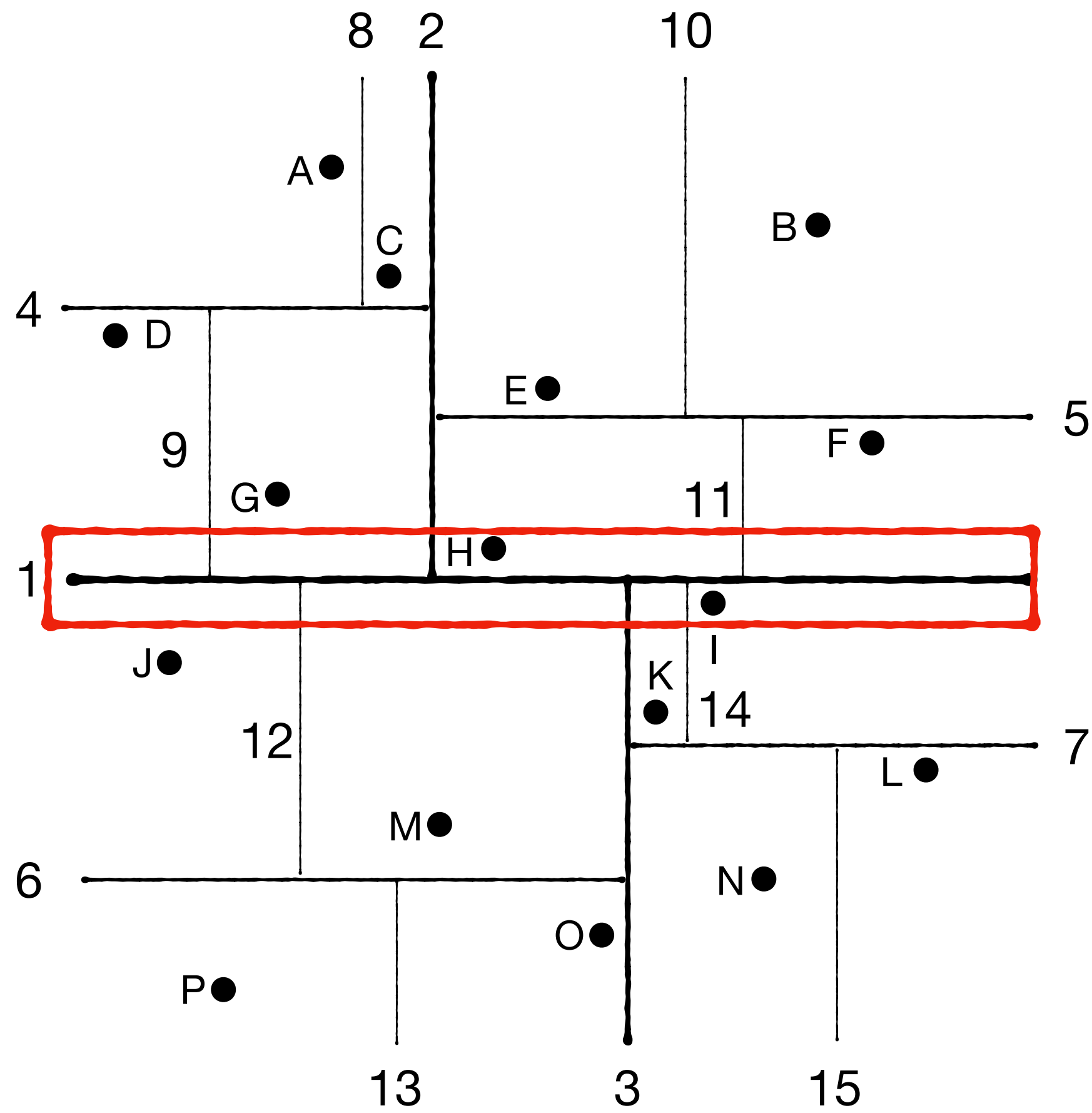
Cost? $O(\sqrt{N})$

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



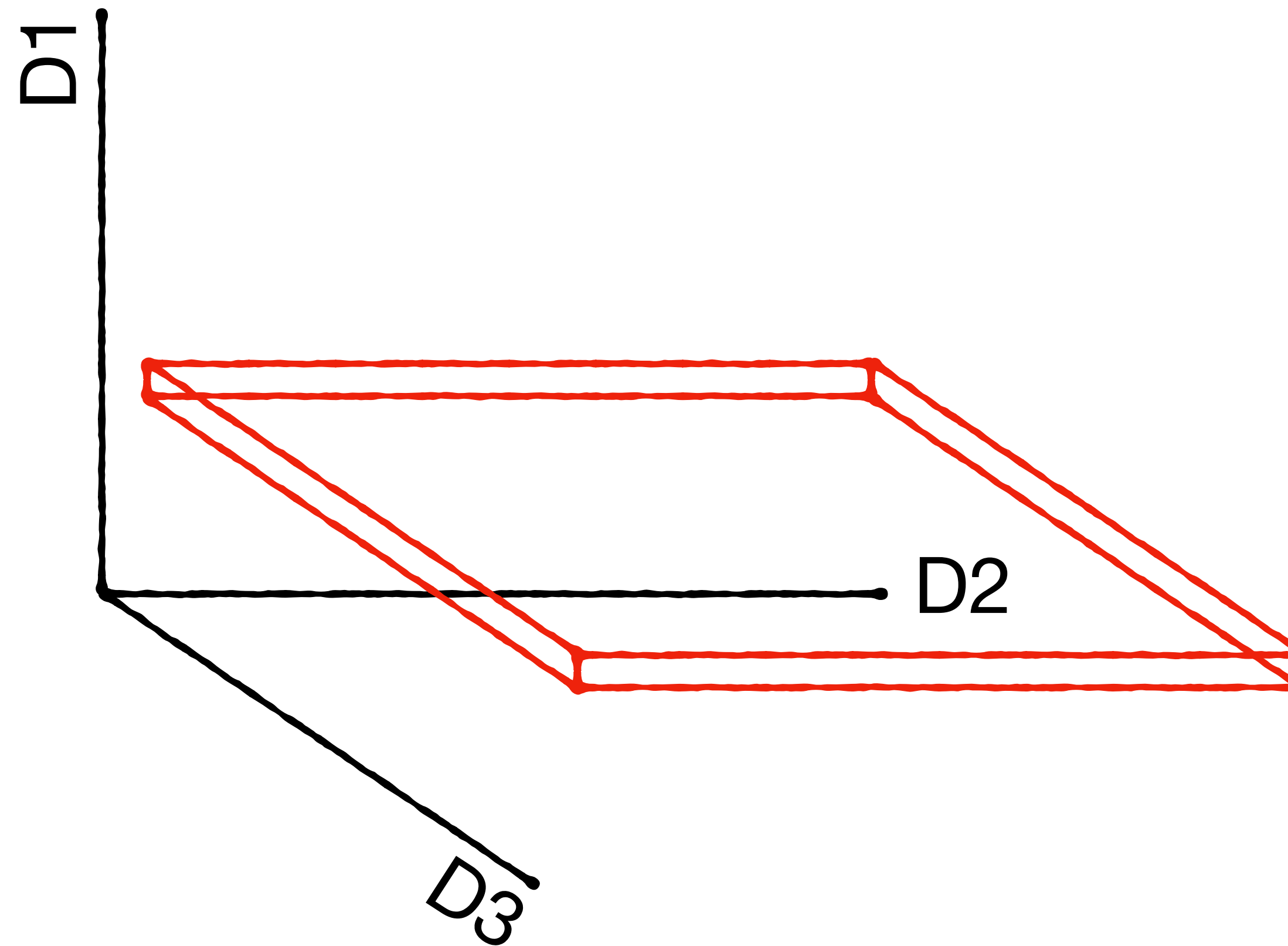
Cost? $O(2 \cdot \sqrt{N})$

Can we make this even worse? Intersect across dimensions



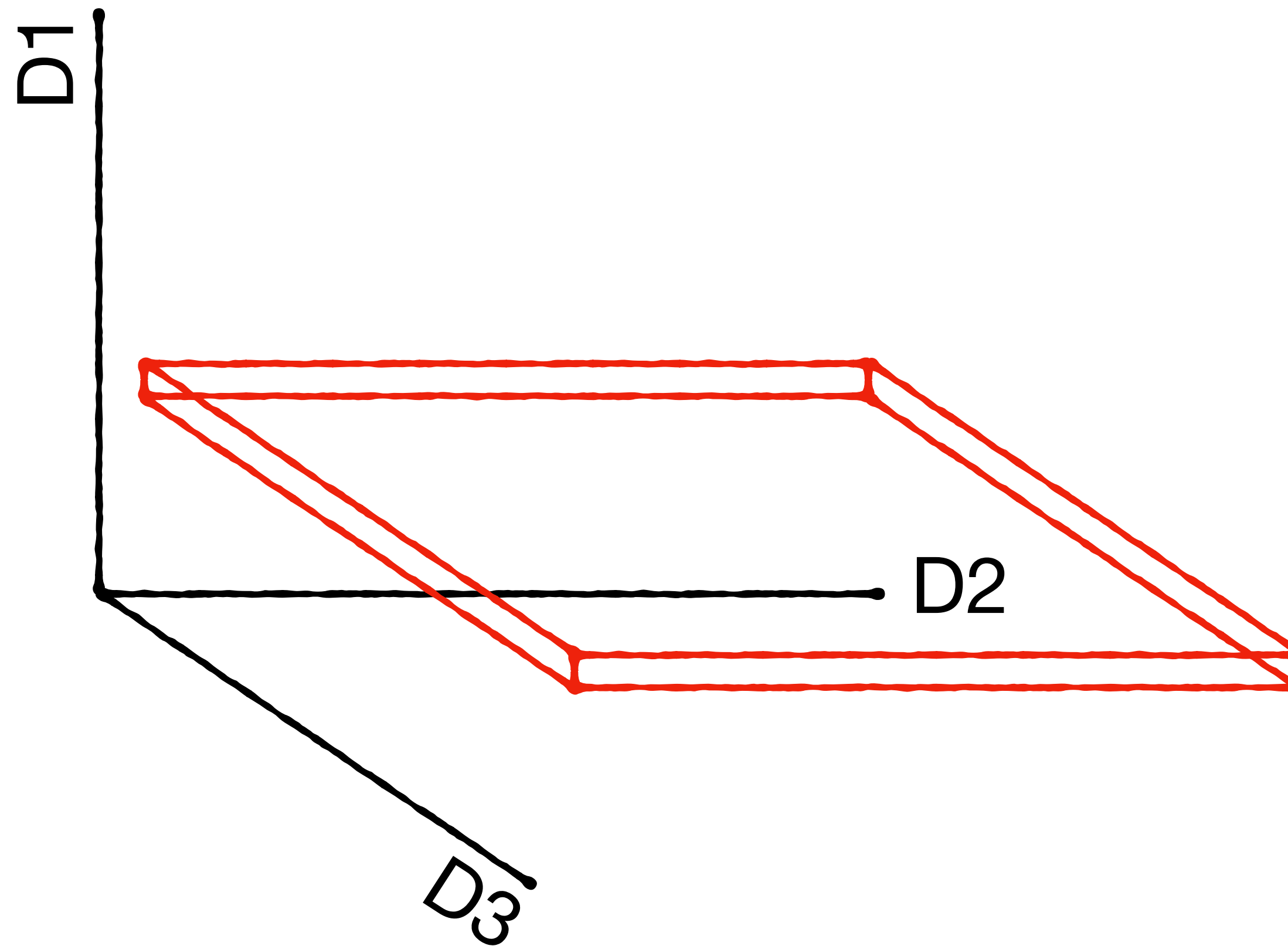
What if we increase dimensionality?

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



What if we increase dimensionality?

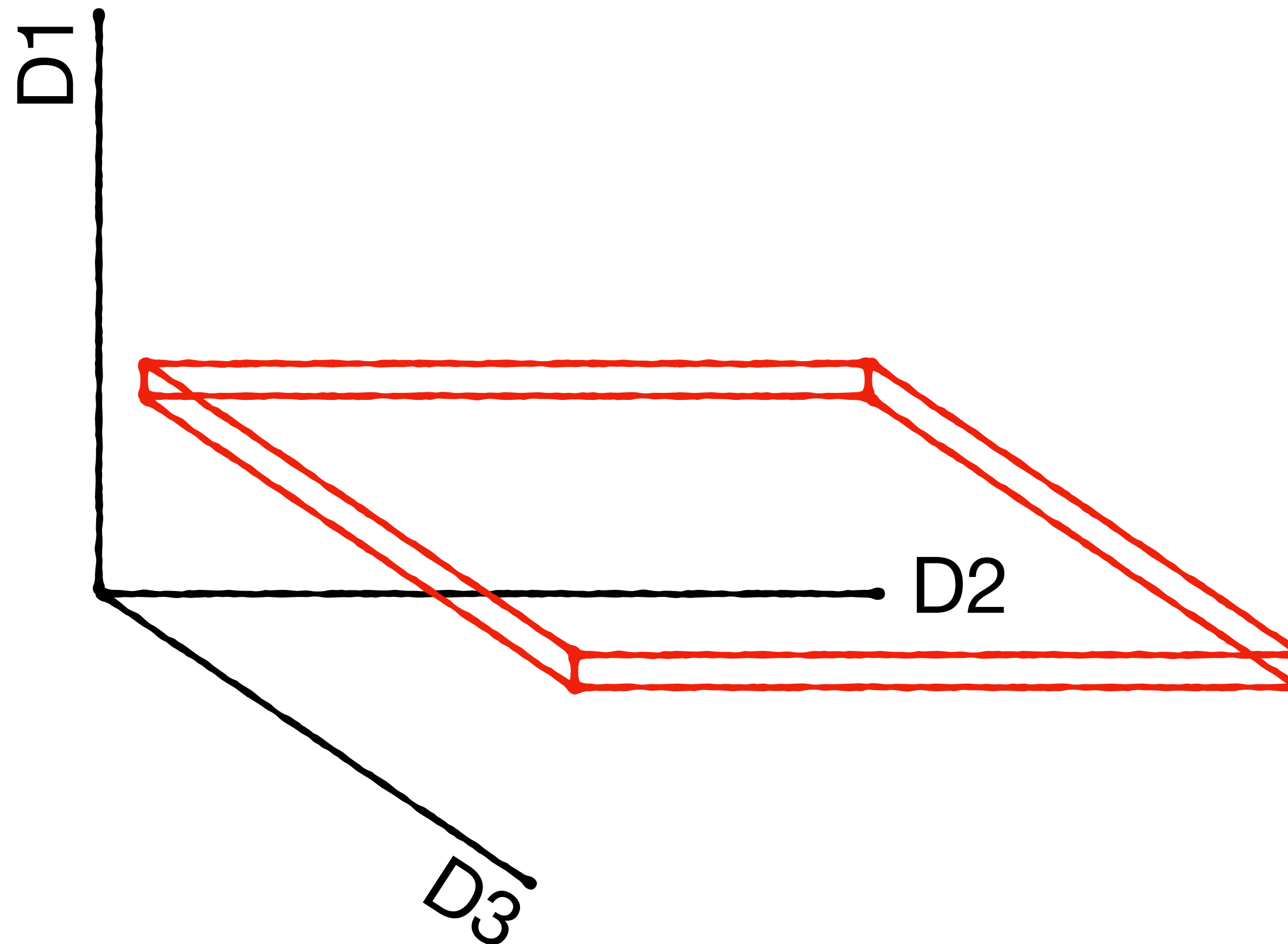
Cost? $O(D \cdot N^{(D-1)/D})$



What if we increase dimensionality?

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

Seems to only occur with low selectivity over most dimensions

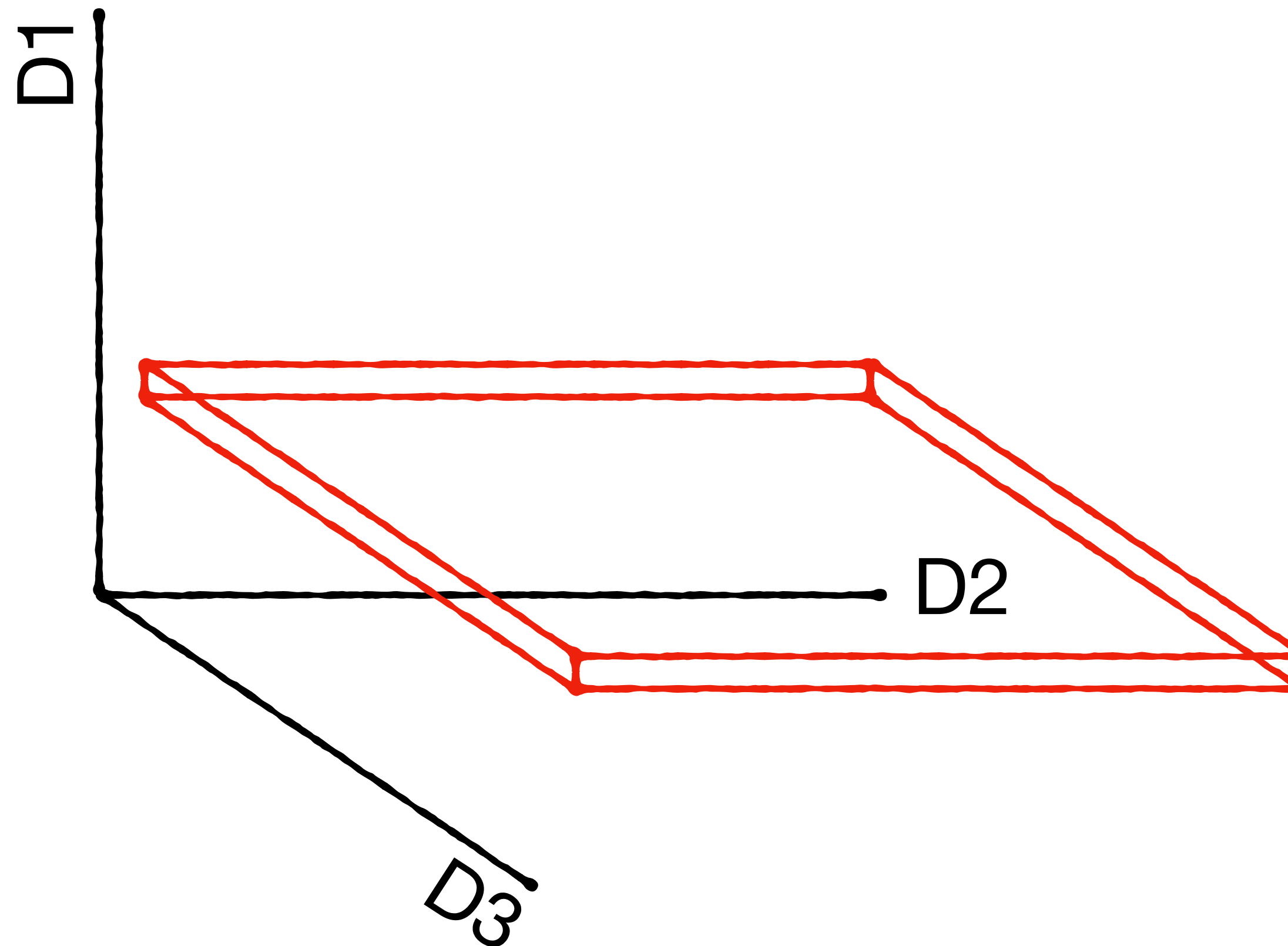


What if we increase dimensionality?

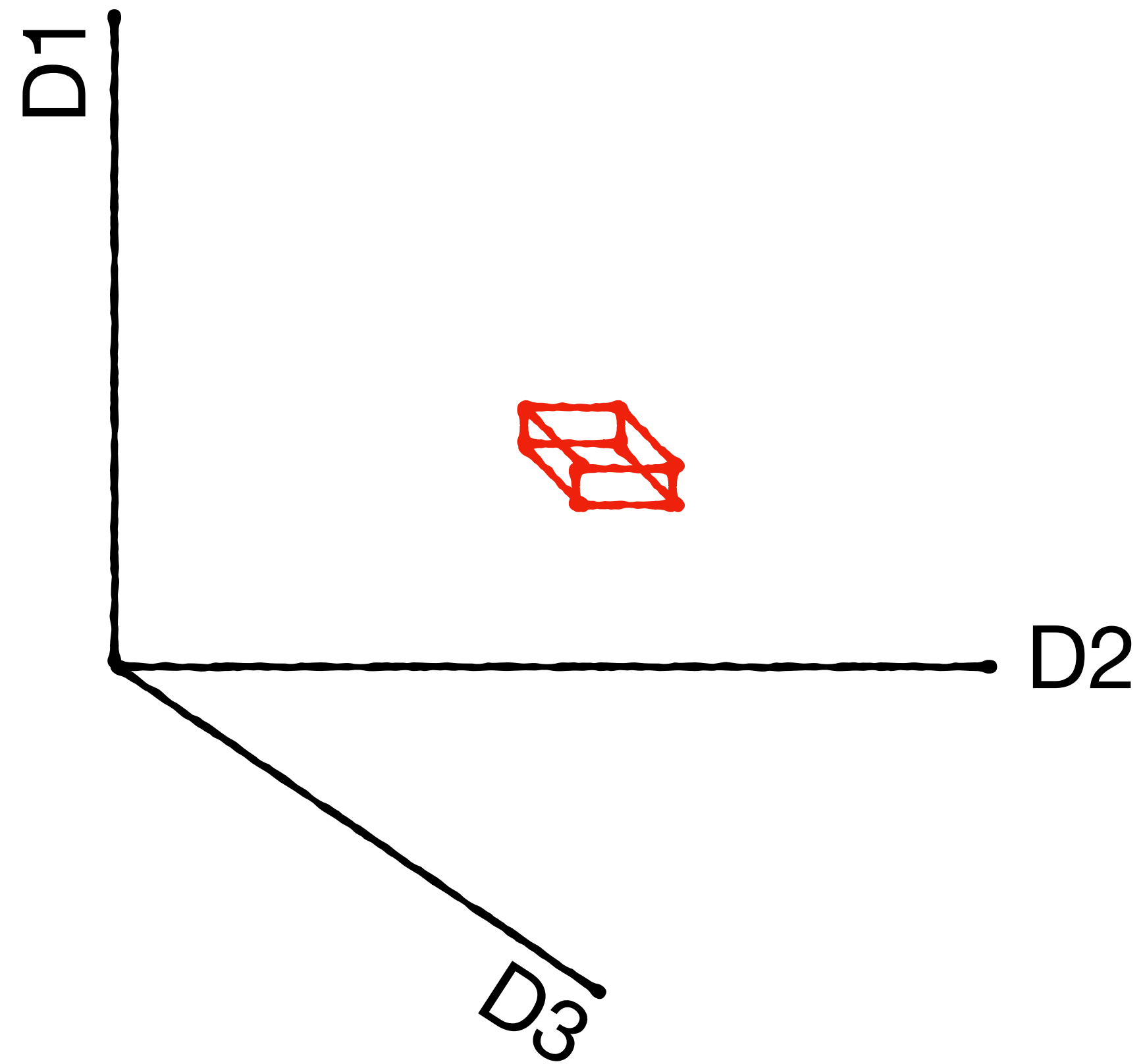
Cost? $O(D \cdot N^{(D-1)/D})$

Seems to only occur with low selectivity over most dimensions

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

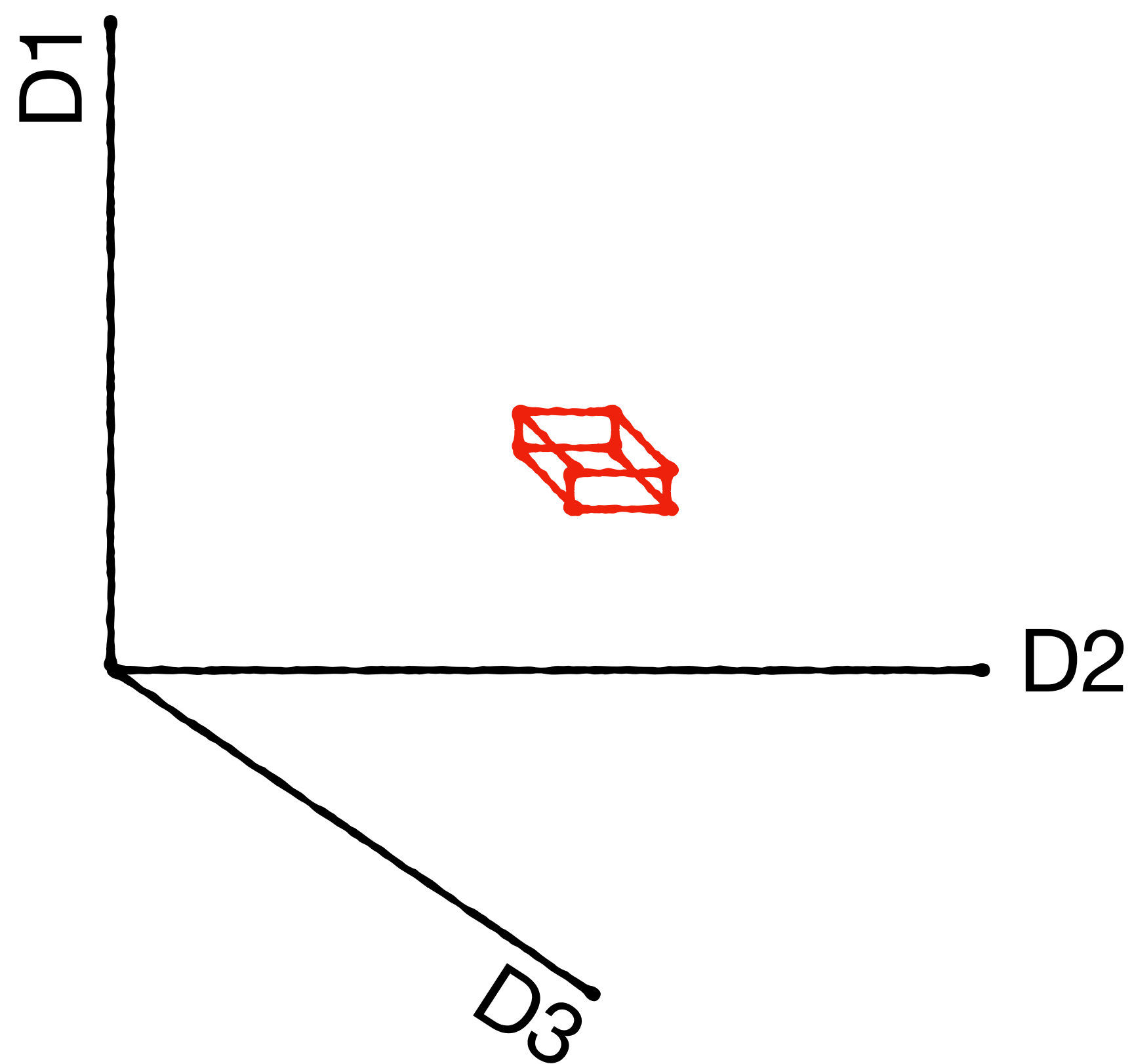


Selective queries across all dimensions are likely not so bad

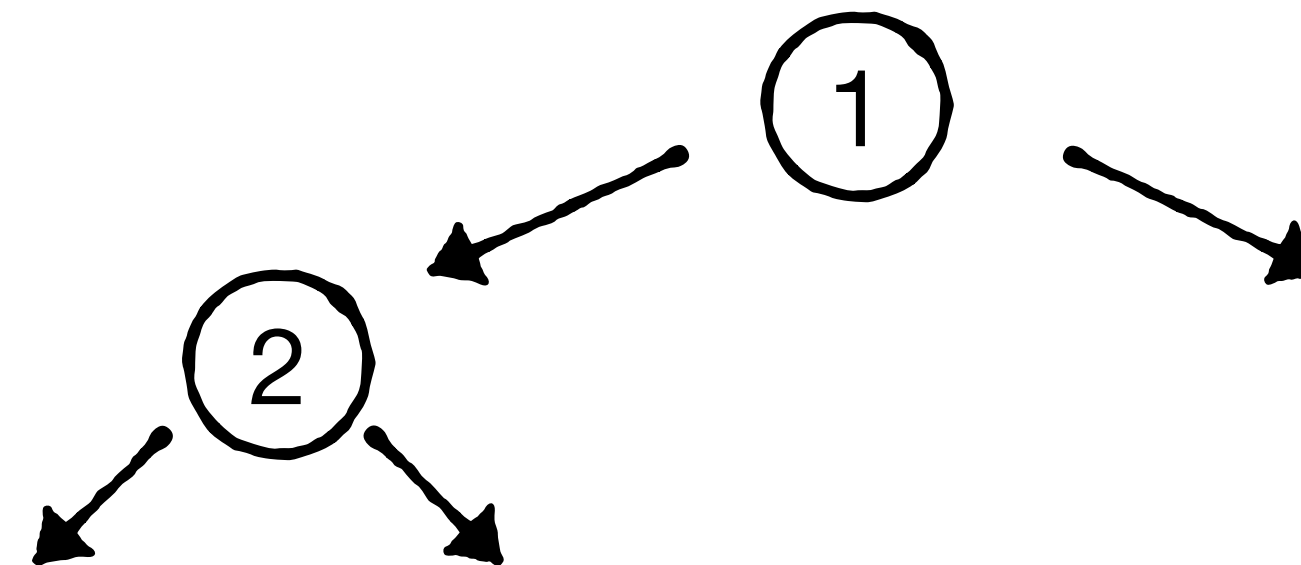
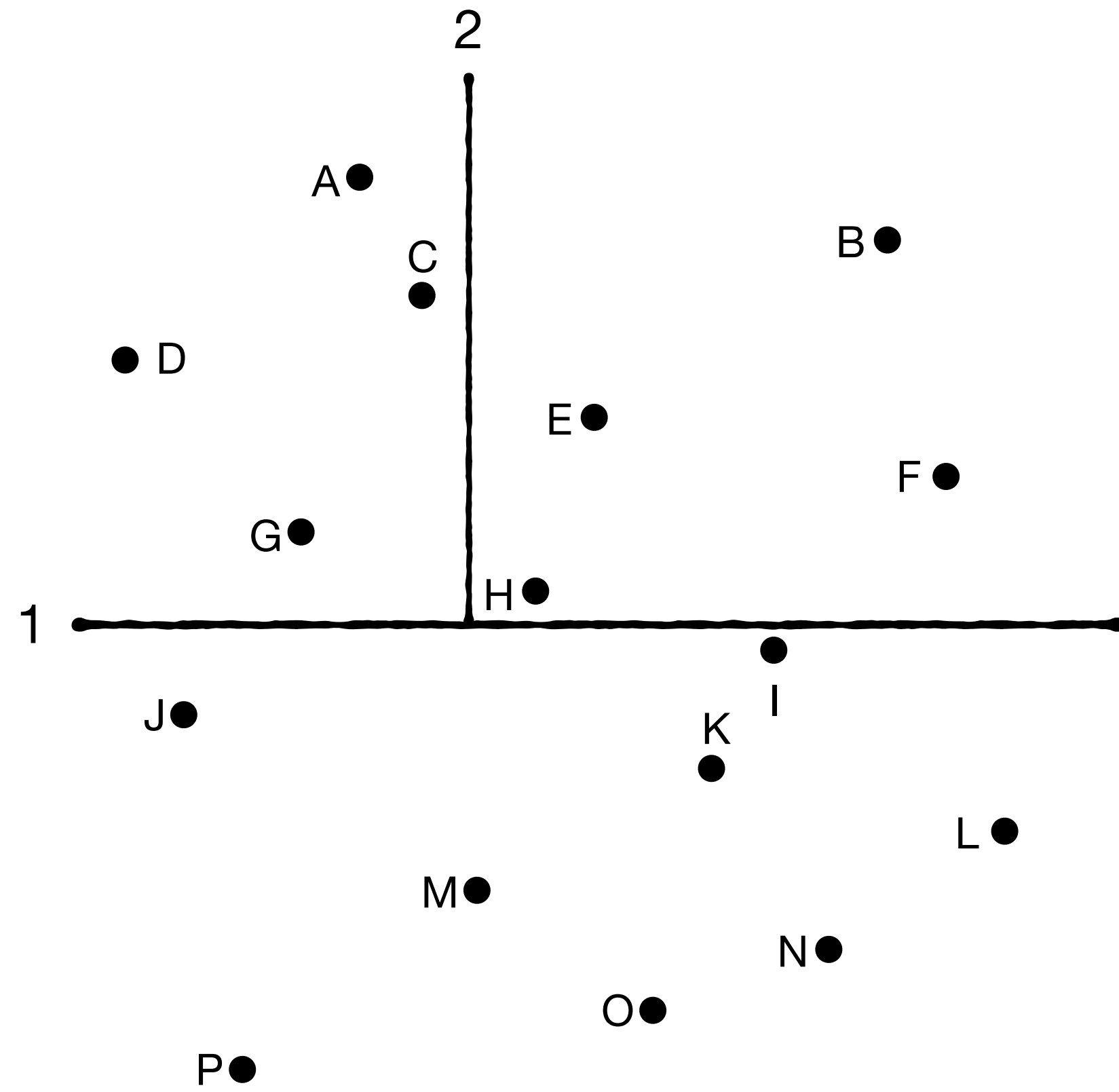


Selective queries across all dimensions are likely not so bad

My guess: $O(D \cdot \log(N))$

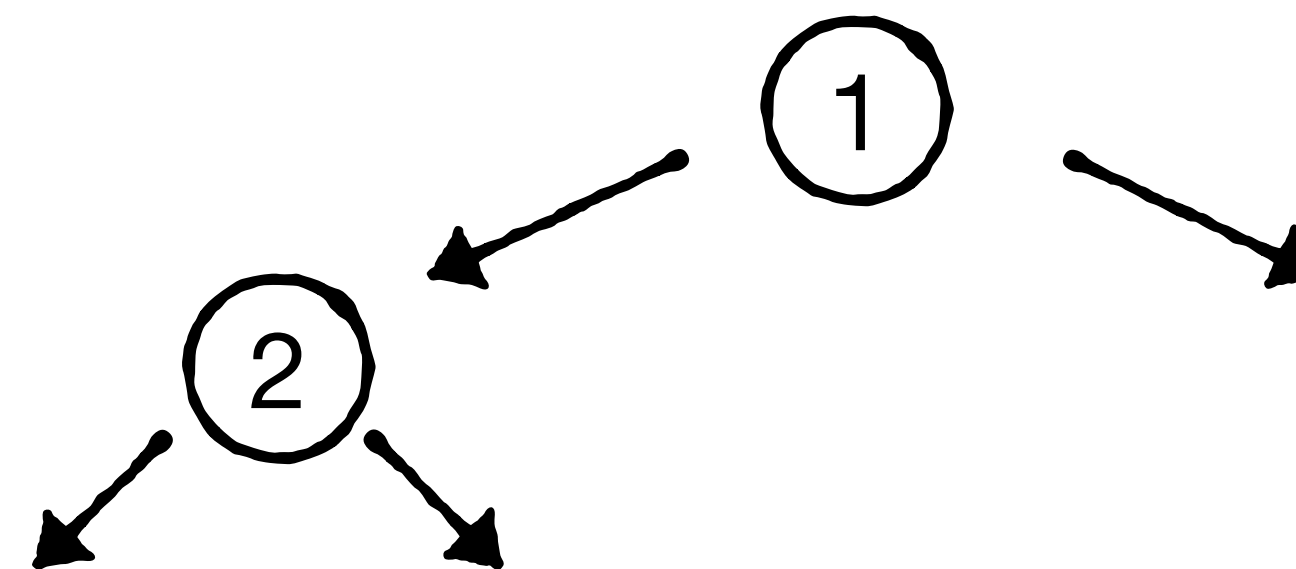
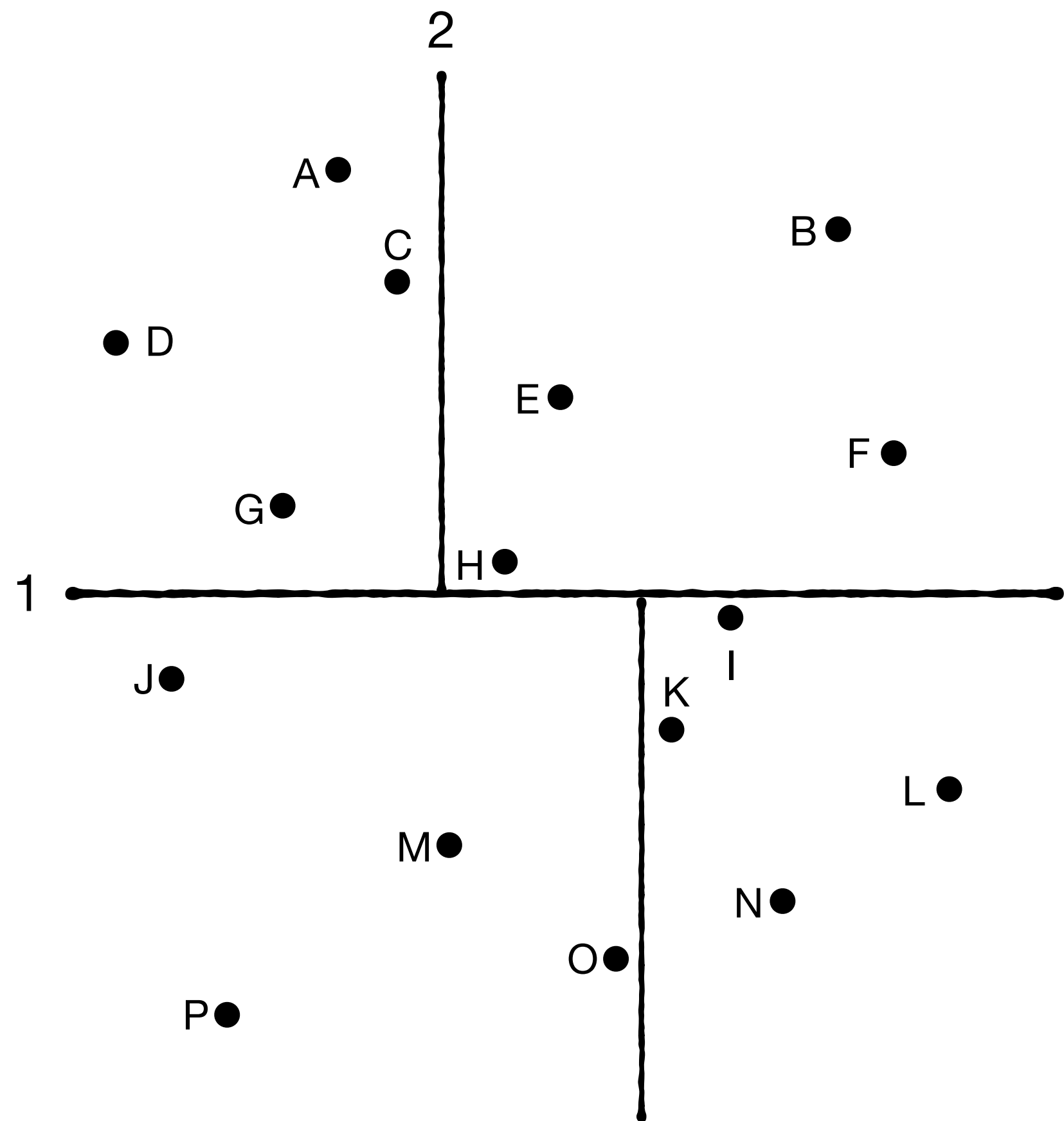


Efficient constructions



Efficient constructions

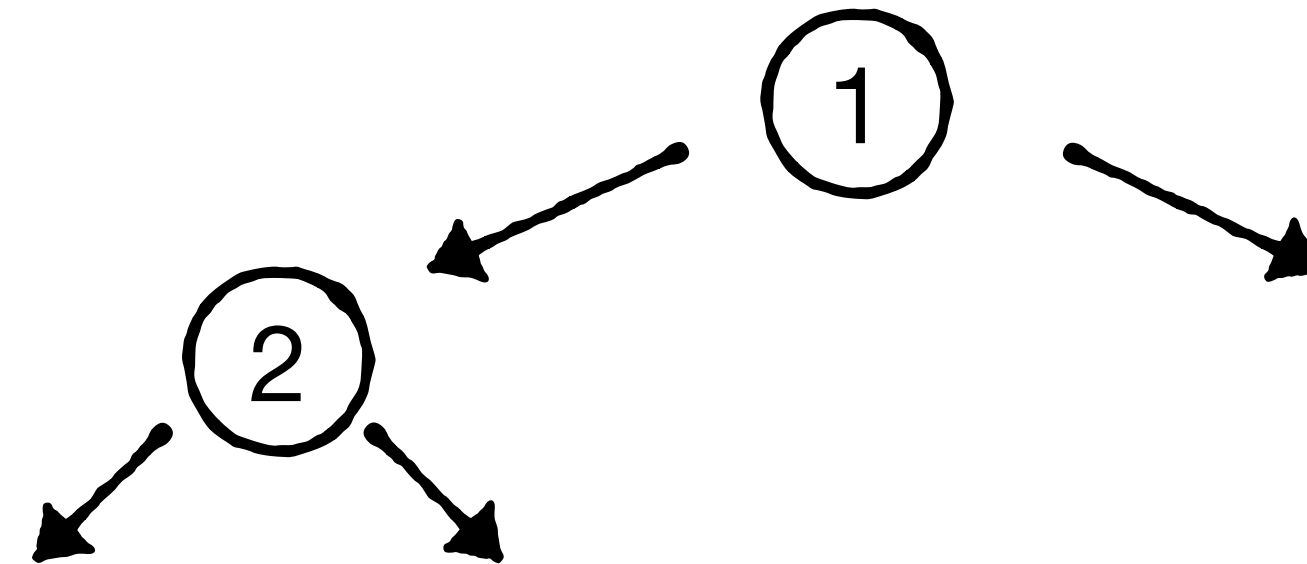
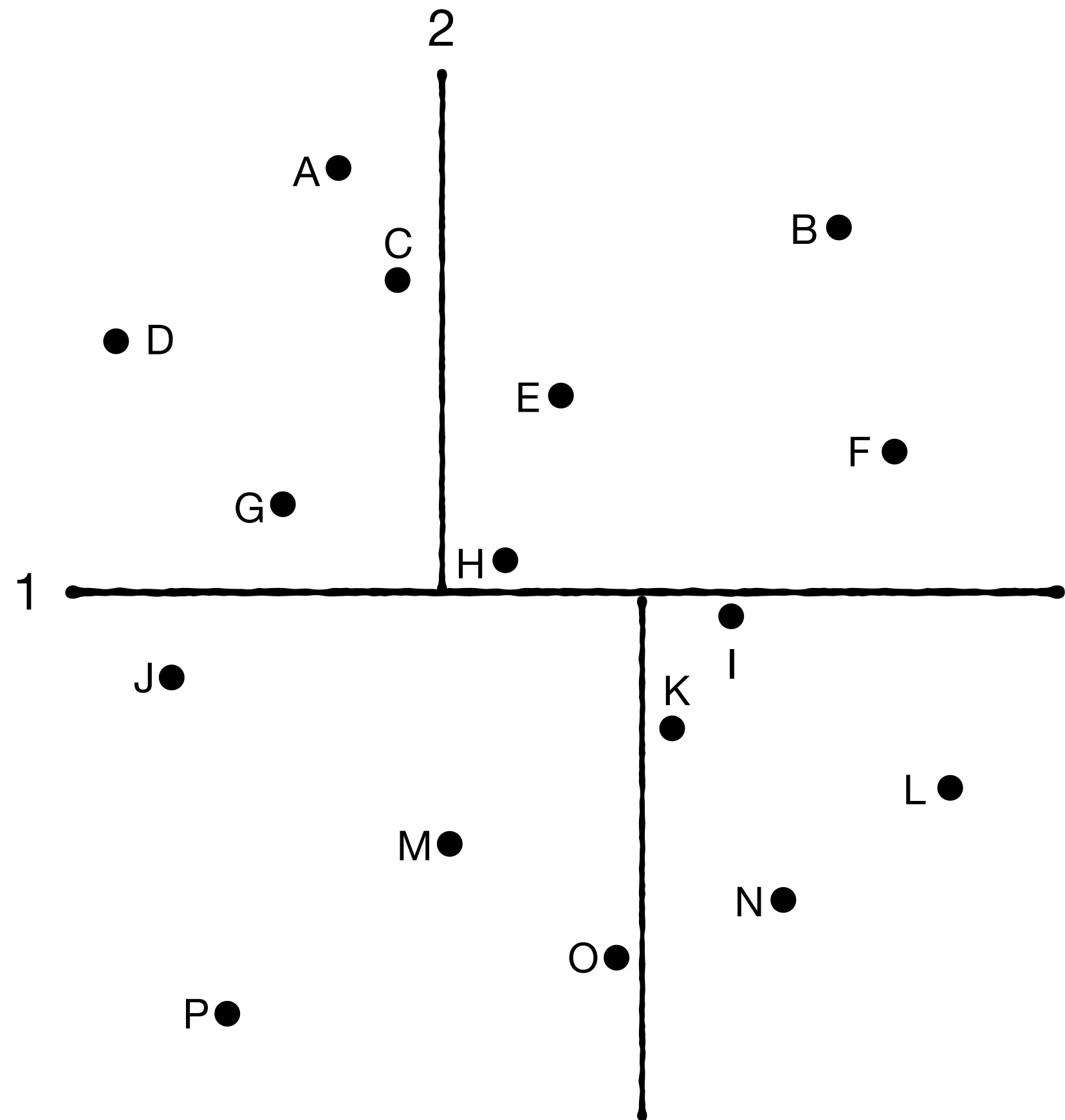
How to find median in each iteration?



Efficient constructions

How to find median in each iteration?

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

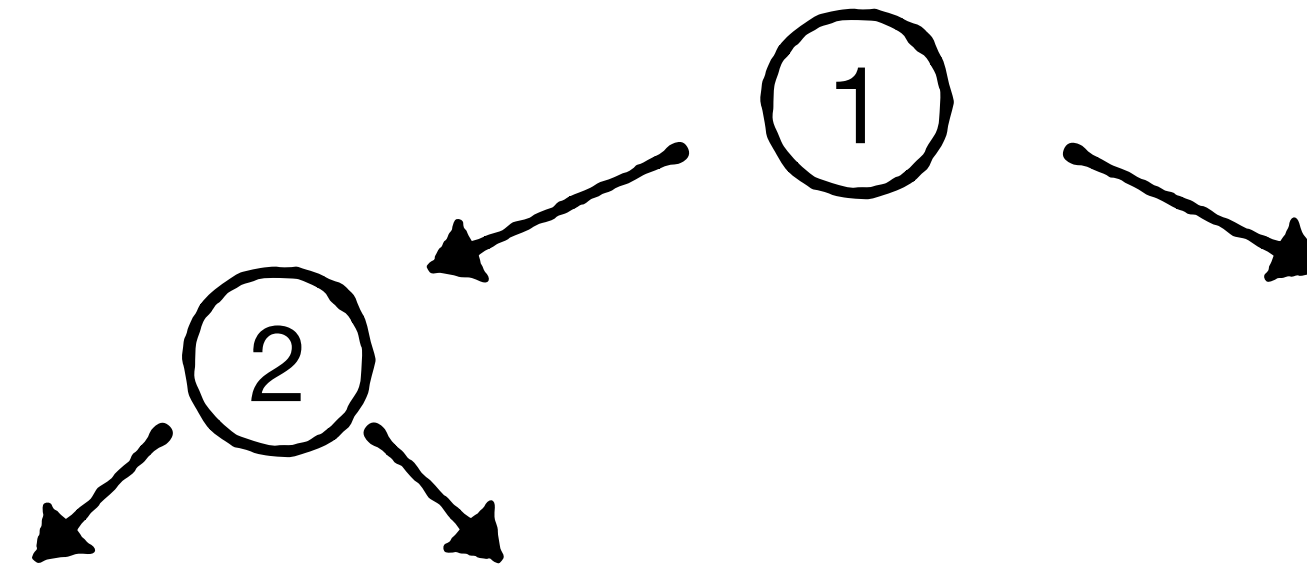
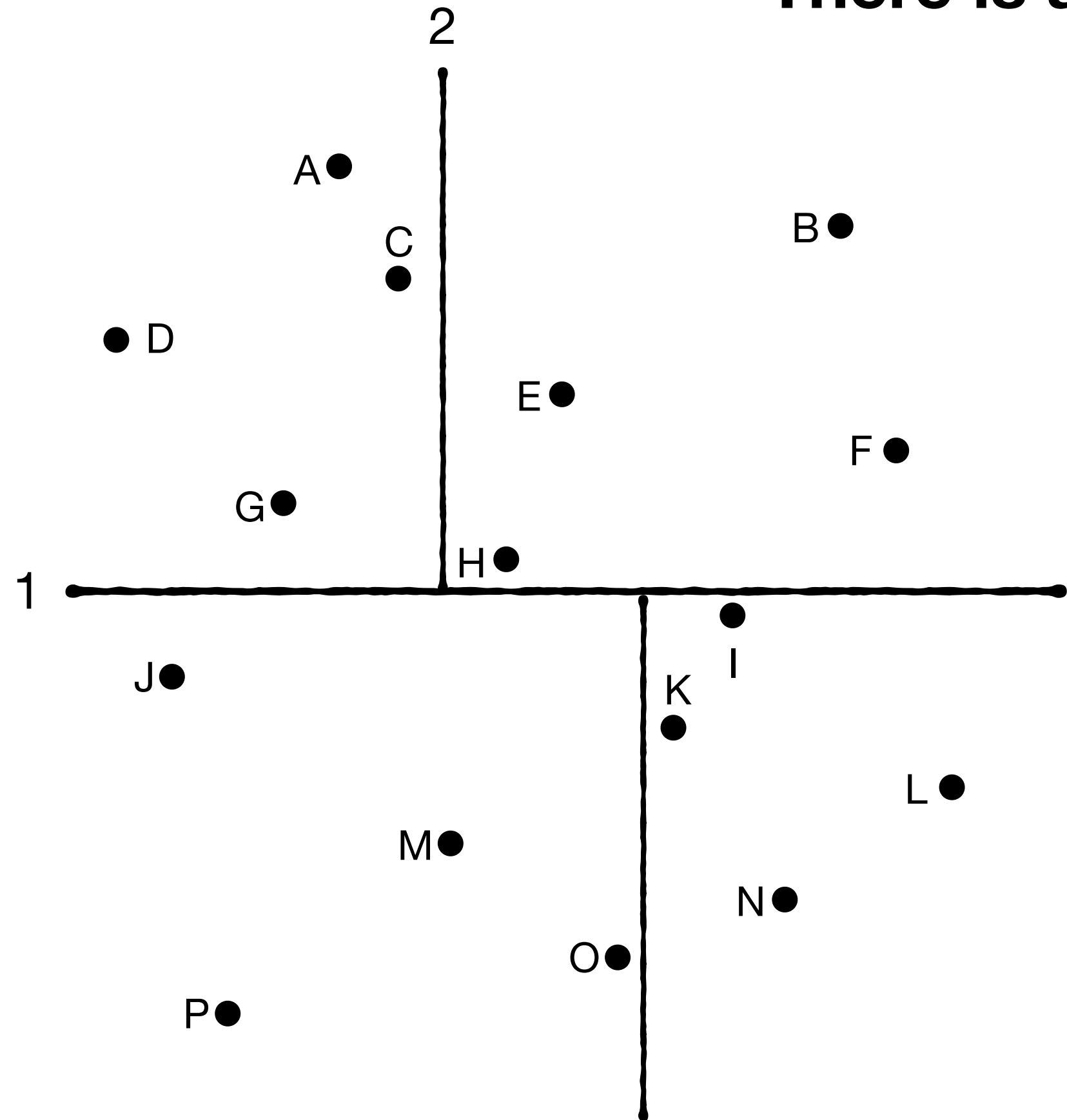


Efficient constructions

How to find median in each iteration?

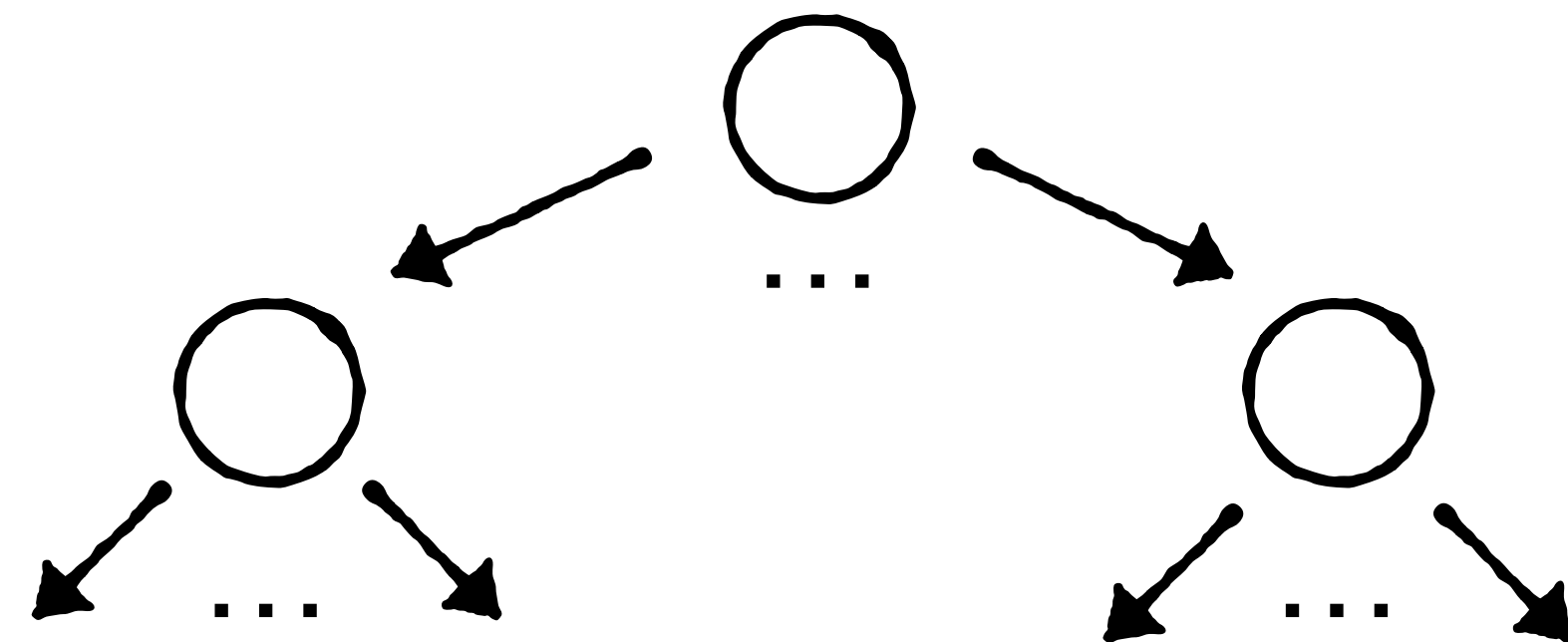
Sort based on each dimension: $O(D \cdot N \cdot \log_2(N))$

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



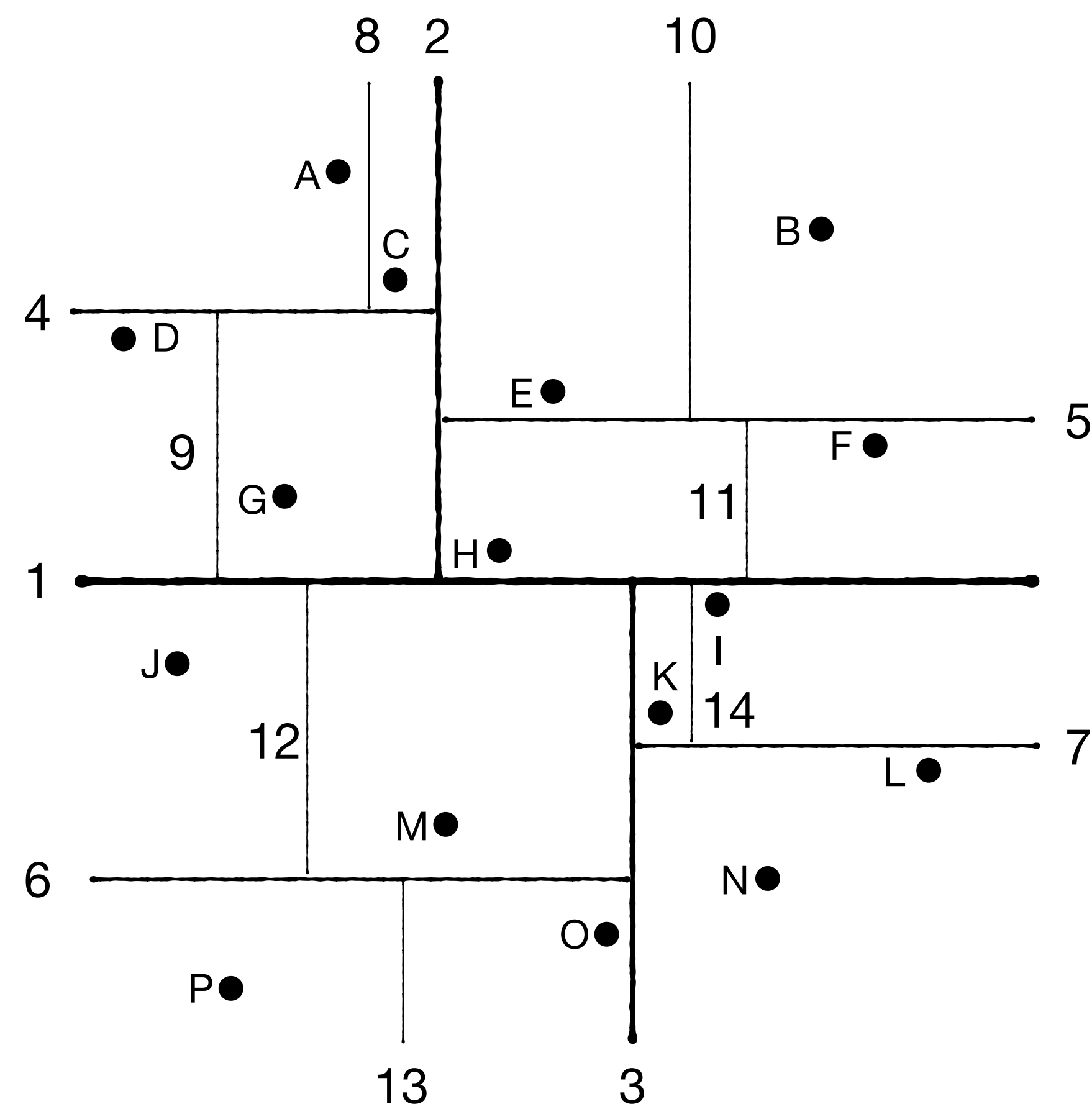
Adaptation for Storage

The KDB-tree: a search structure for large multidimensional dynamic indexes
John Robinson. SIGMOD 1981.



Is partitioning the space the best approach?

1



KD-Trees
1975

R-Trees
1984

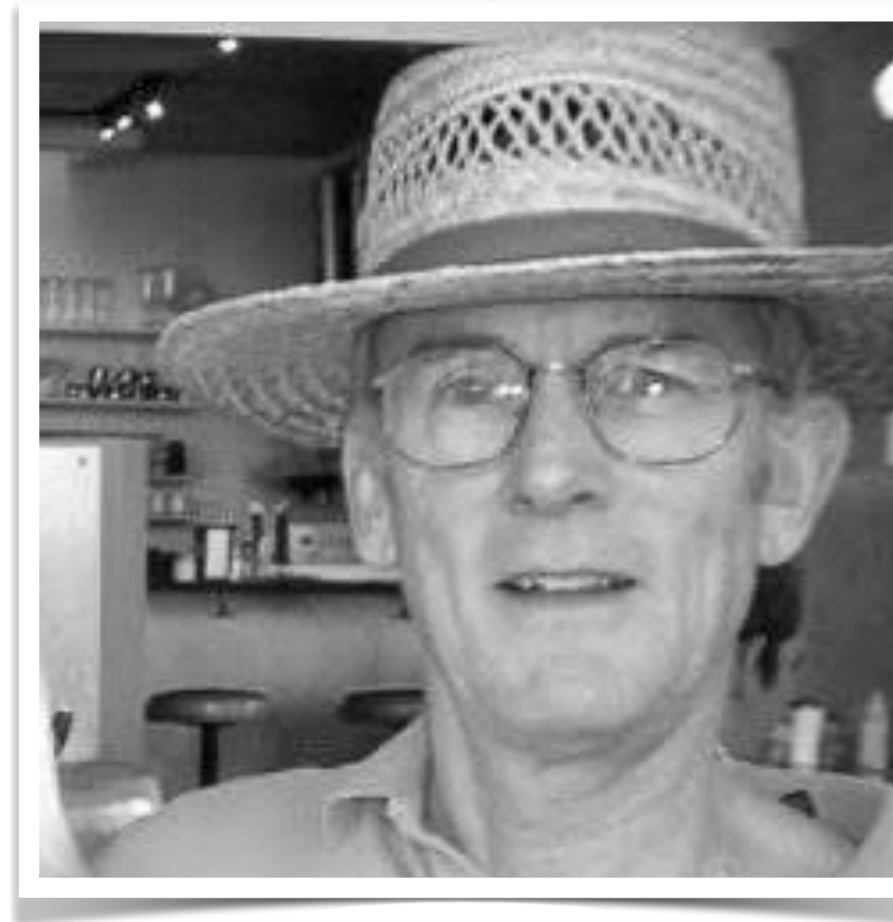
R+tree
1987

R*-tree
1990

UB-tree
2000

R-Trees: A Dynamic Index Structure for Spatial Searching

ACM SIGMOD Record Journal, 1984



Antonin Gutman

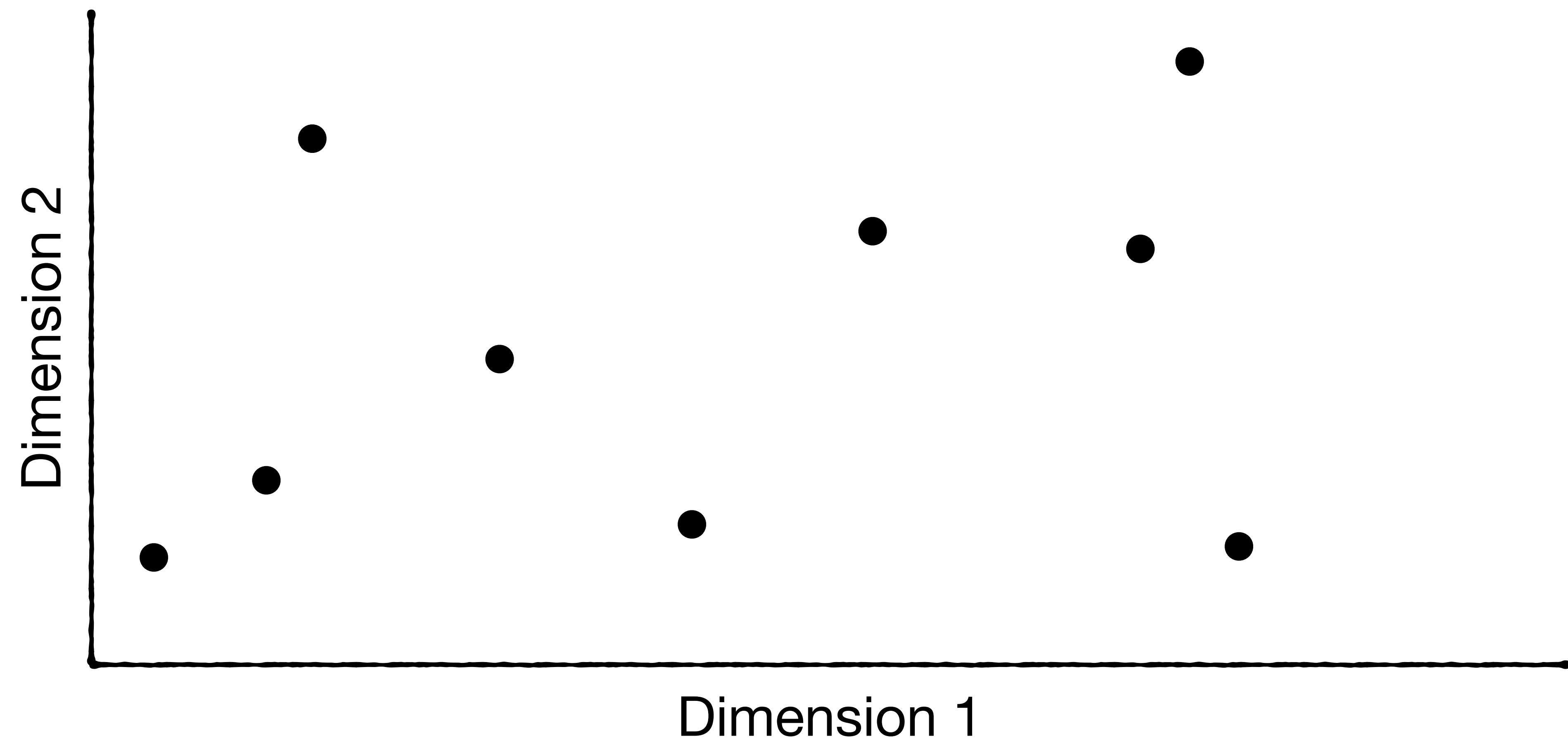
Rectangle-Trees: A Dynamic Index Structure for Spatial Searching

ACM SIGMOD Record Journal, 1984

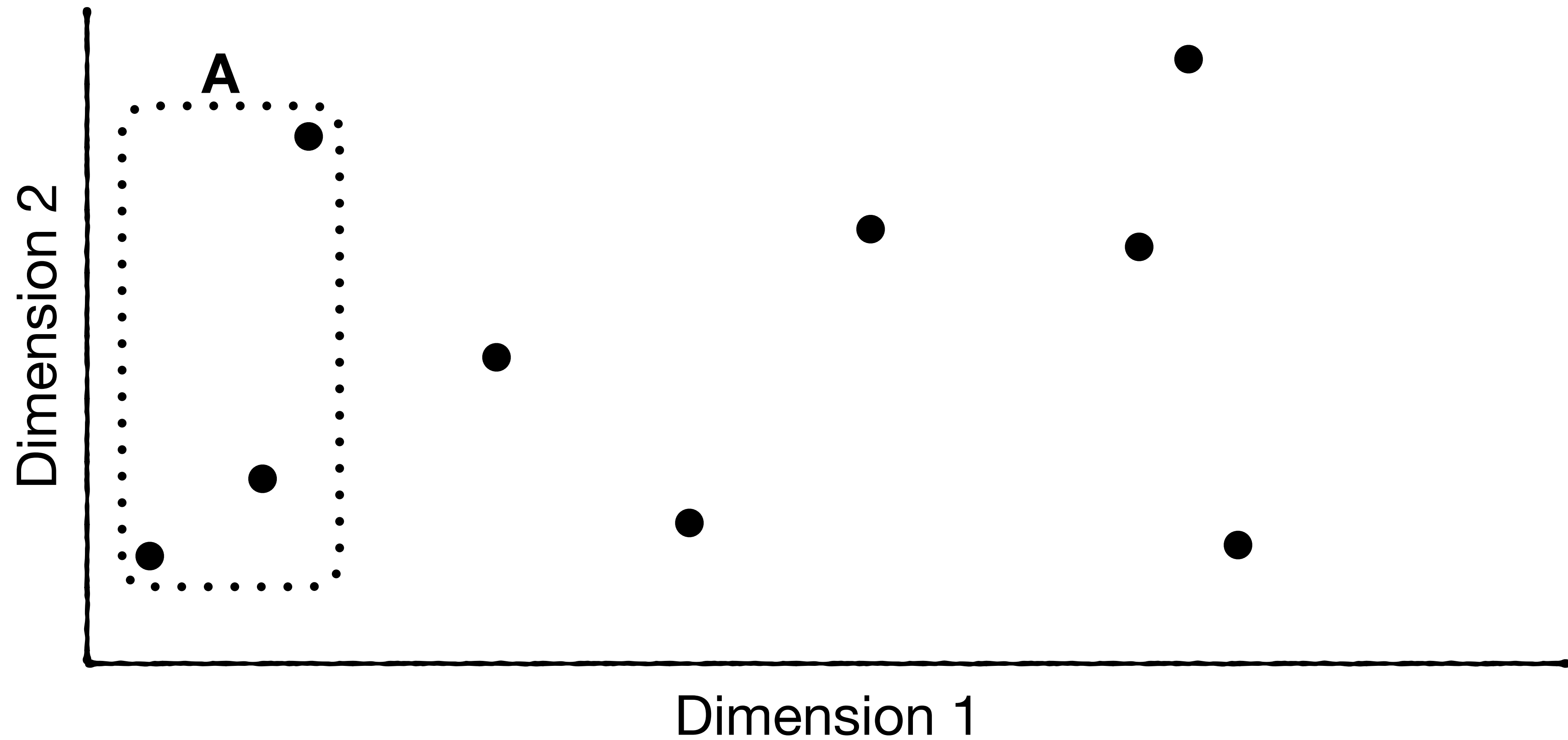


Antonin Gutman

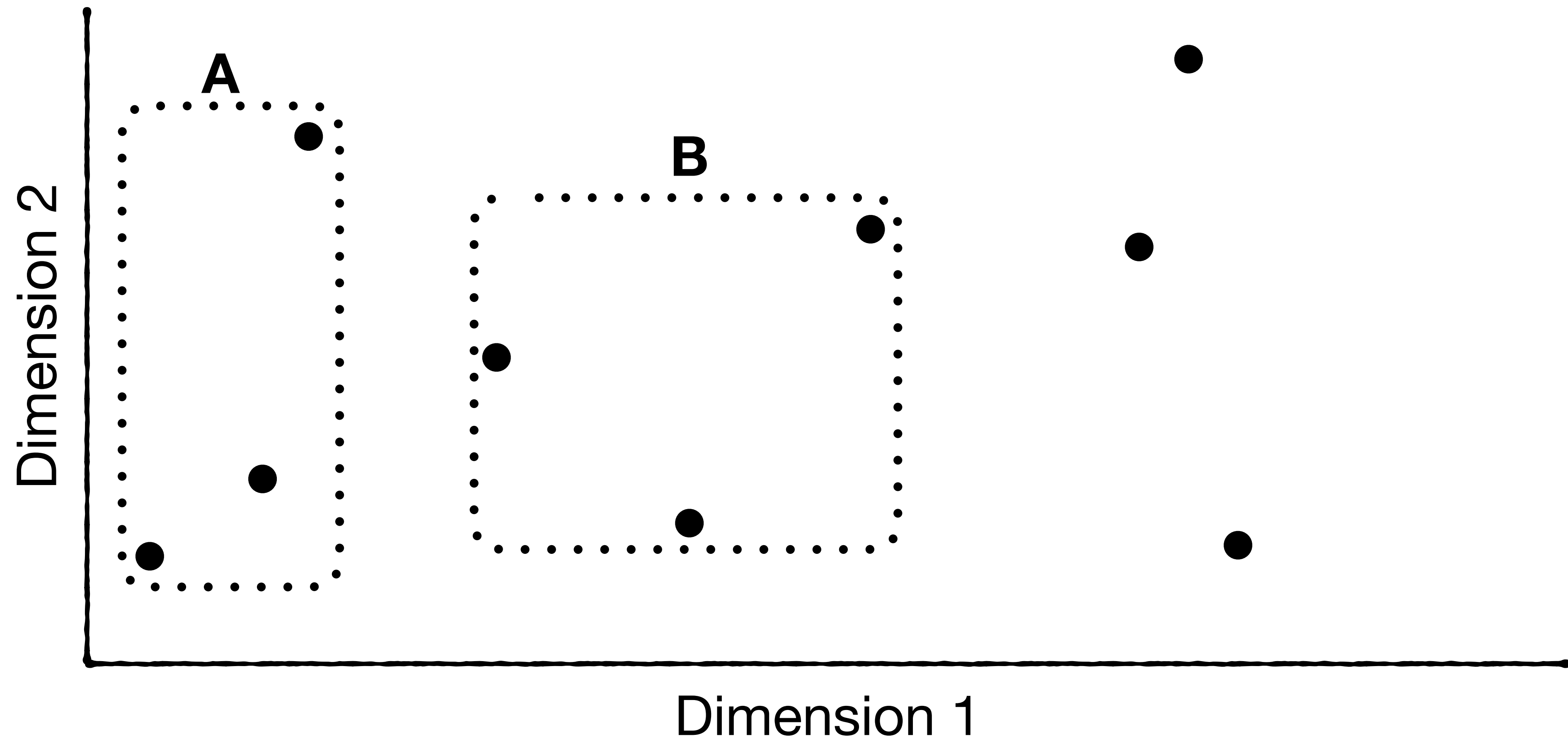
A hierarchy of minimum bounding rectangles



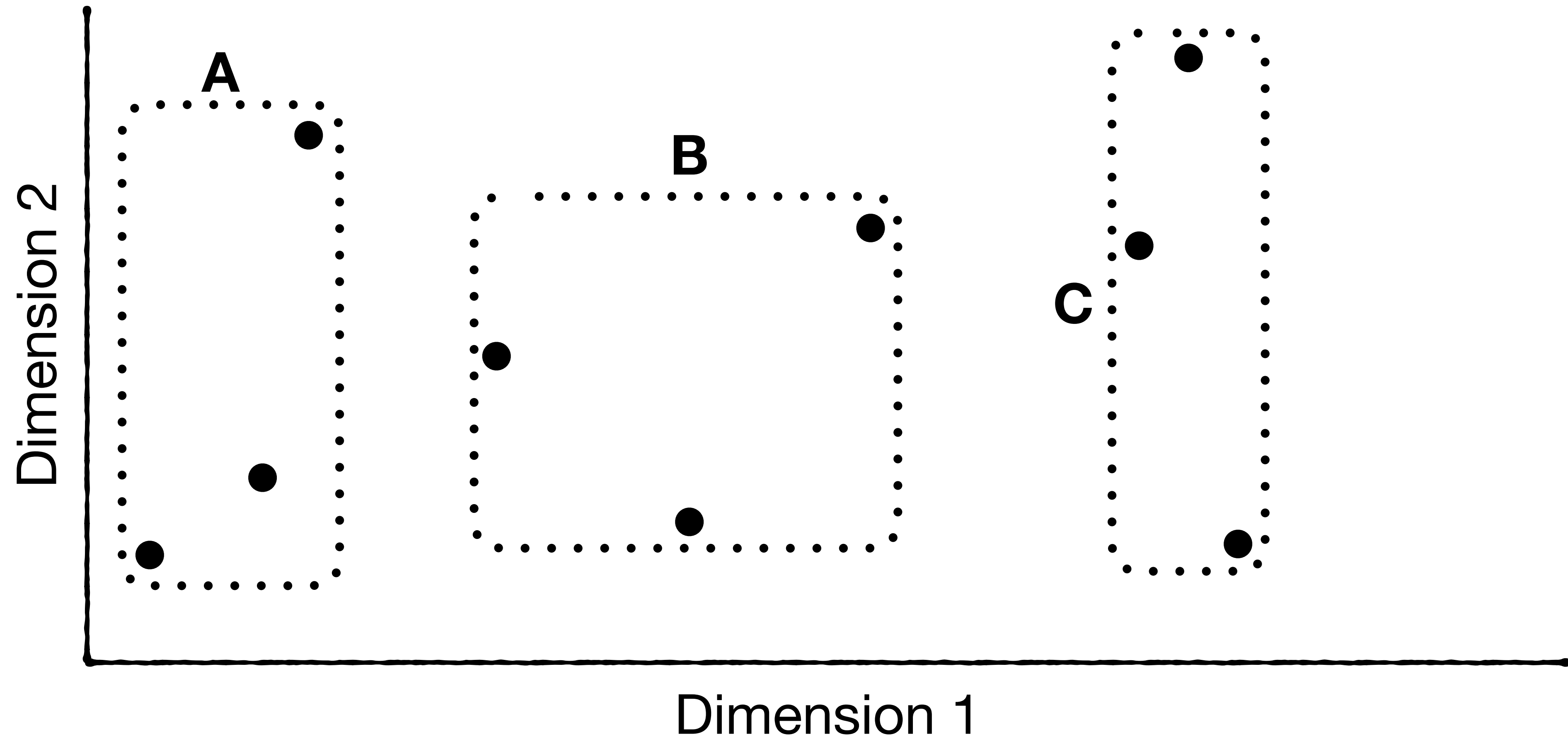
A hierarchy of minimum bounding rectangles



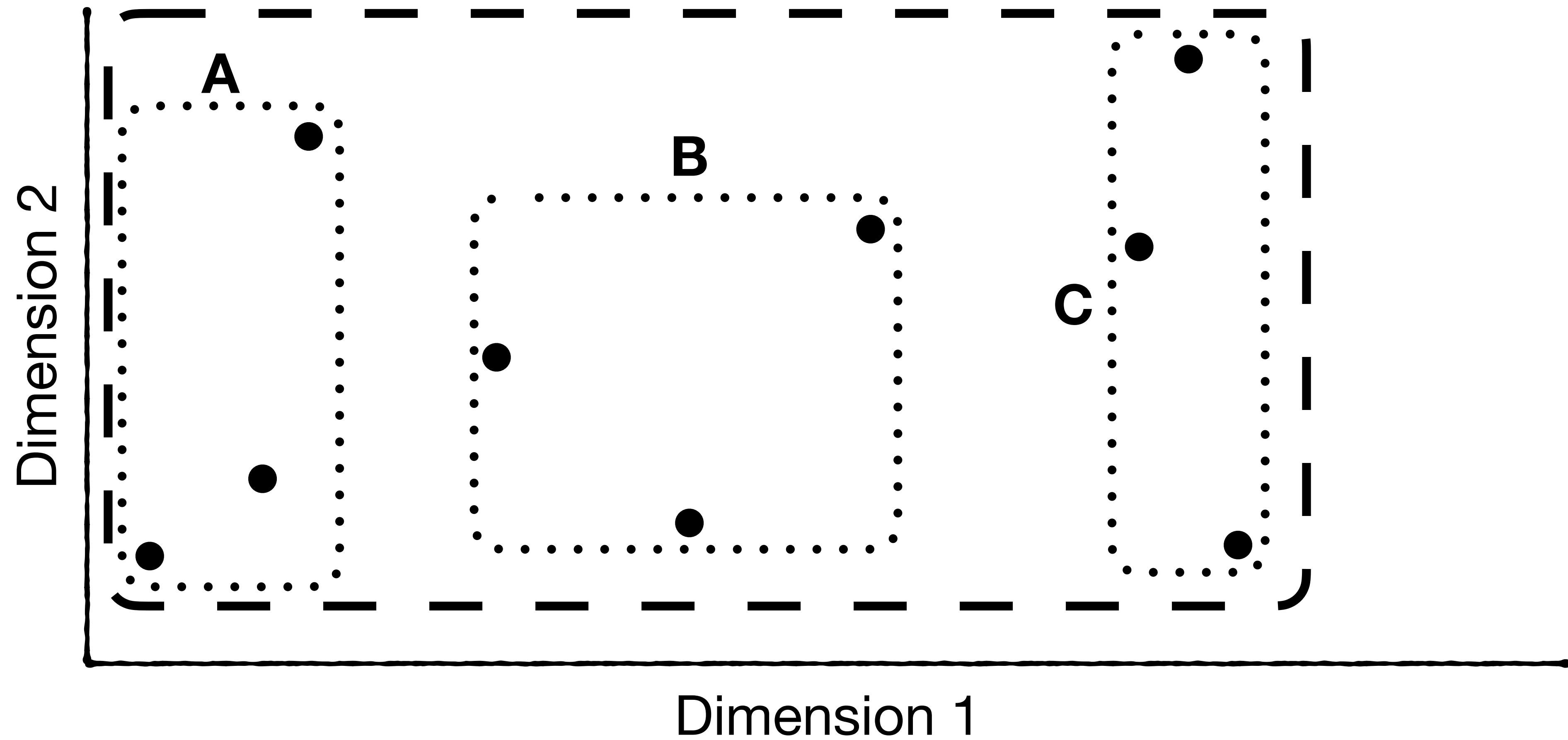
A hierarchy of minimum bounding rectangles



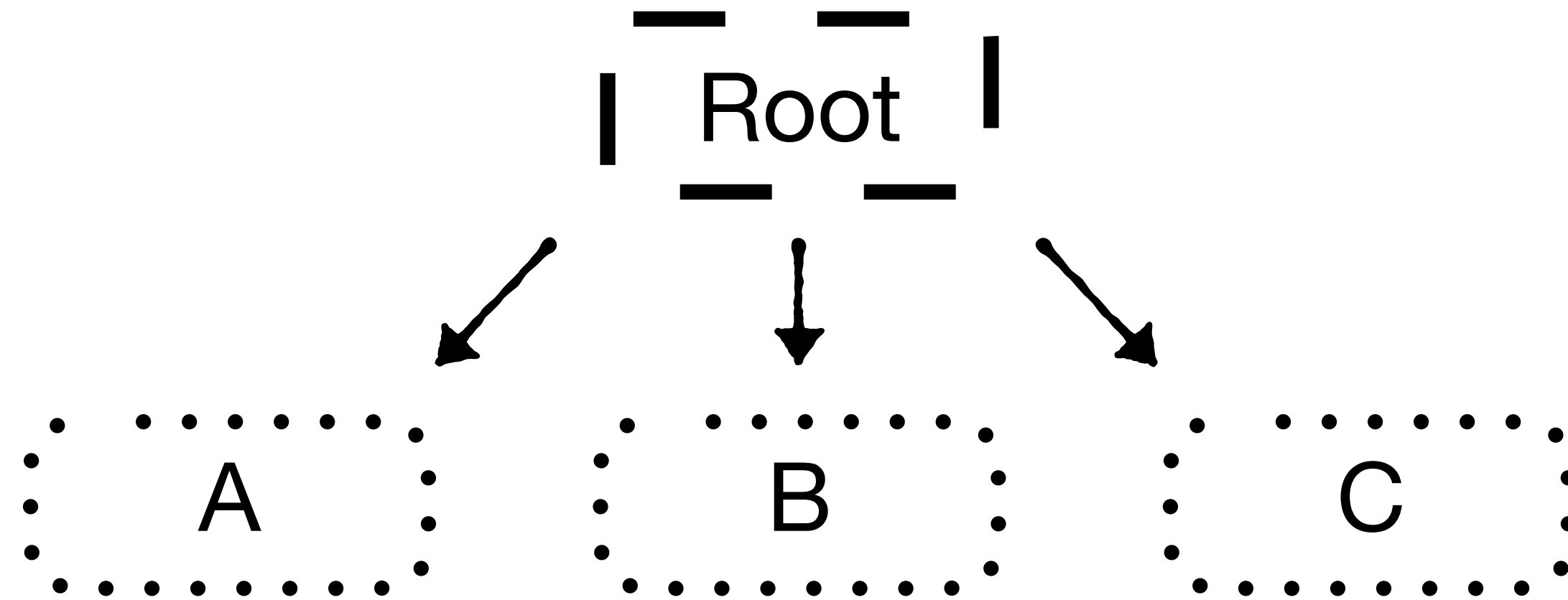
A hierarchy of minimum bounding rectangles



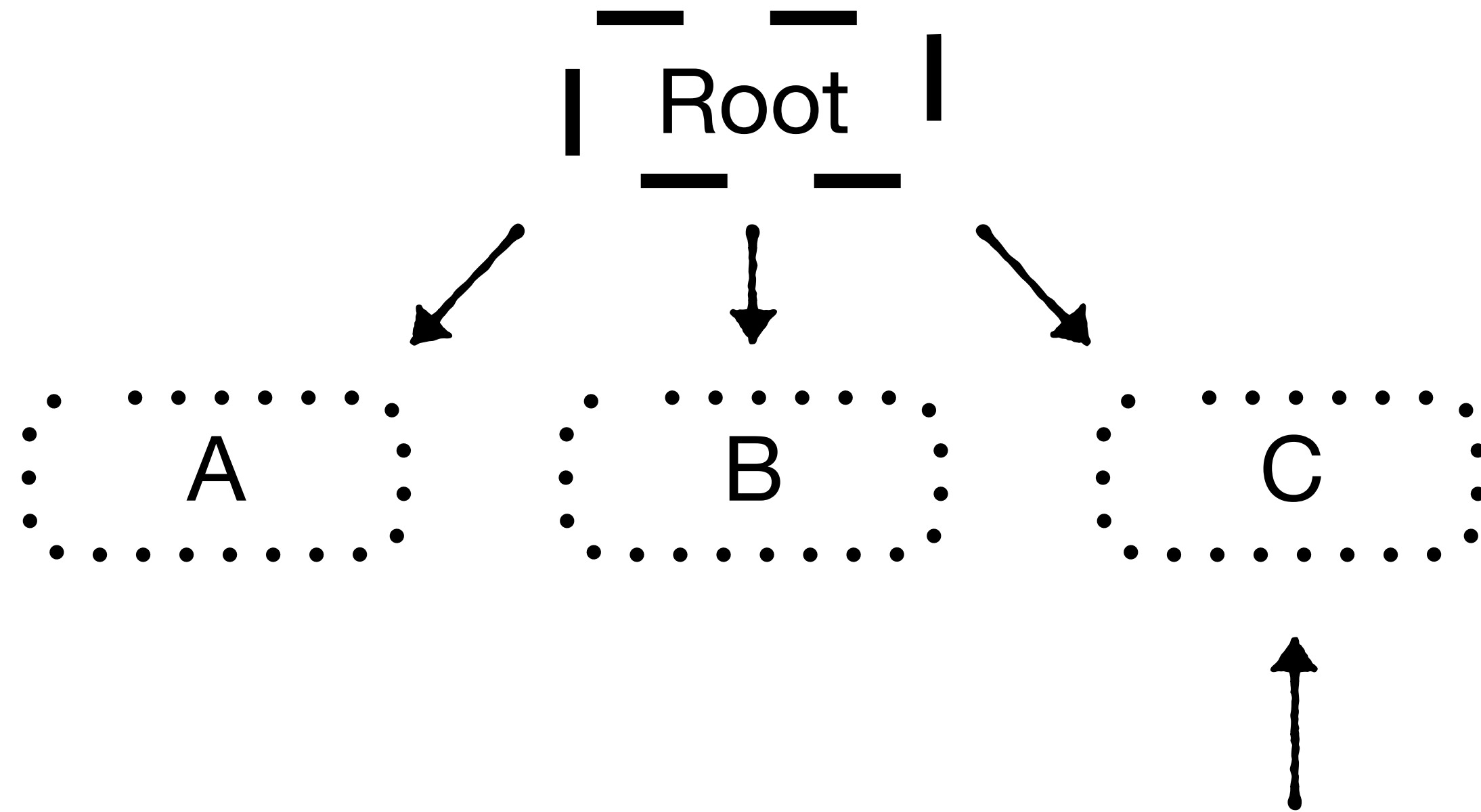
A hierarchy of minimum bounding rectangles



Physically Structured as a B-Tree

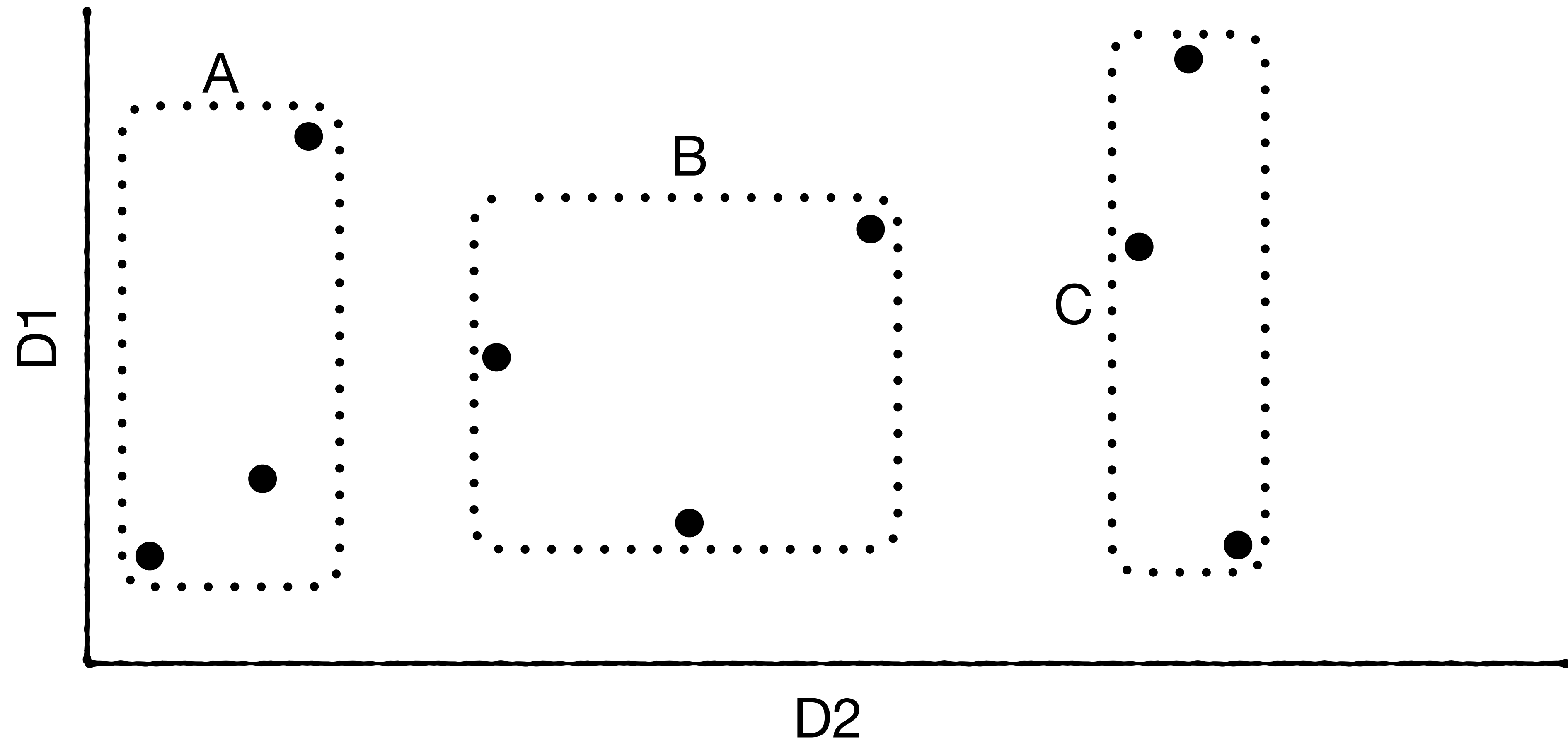


Physically Structured as a B-Tree

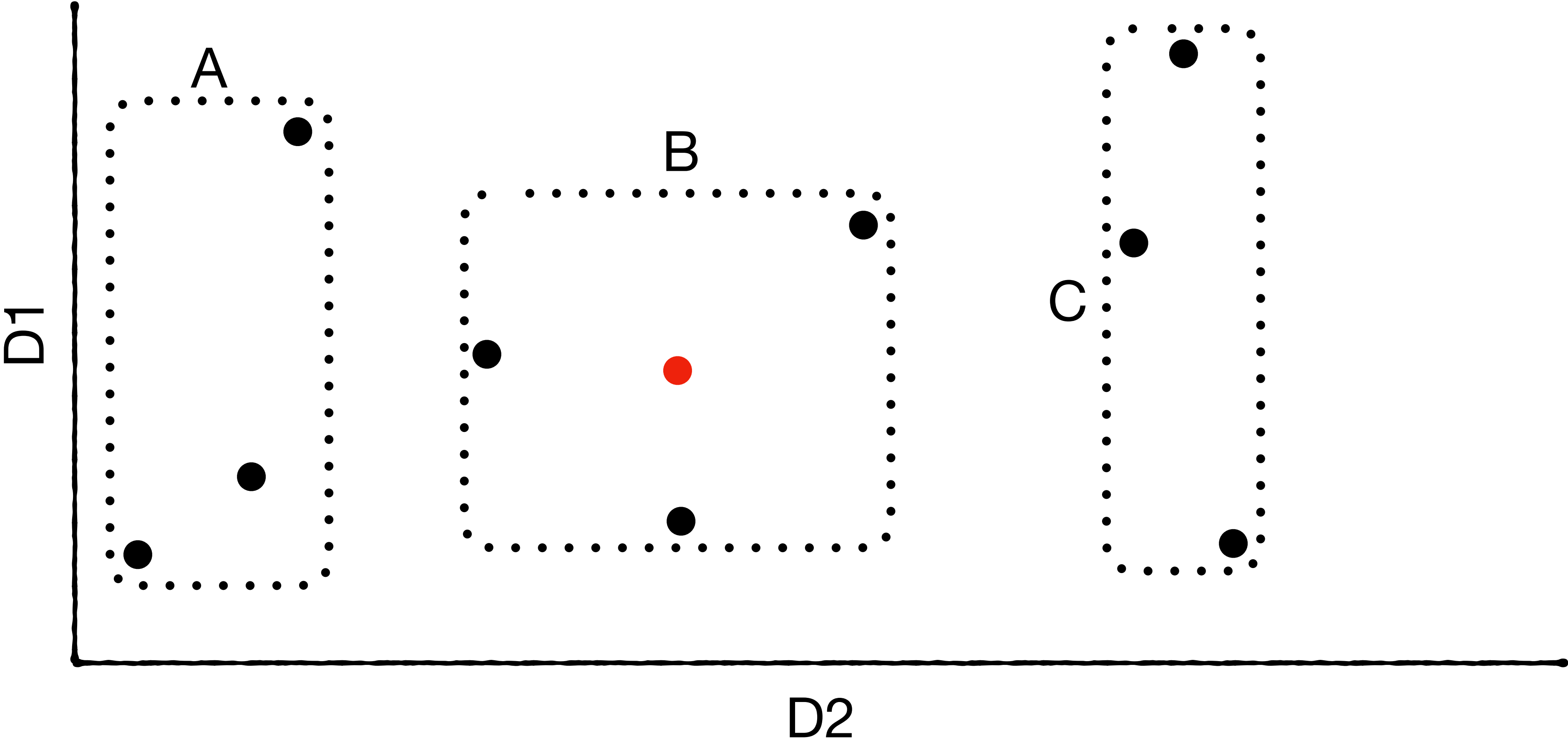


B entries per node

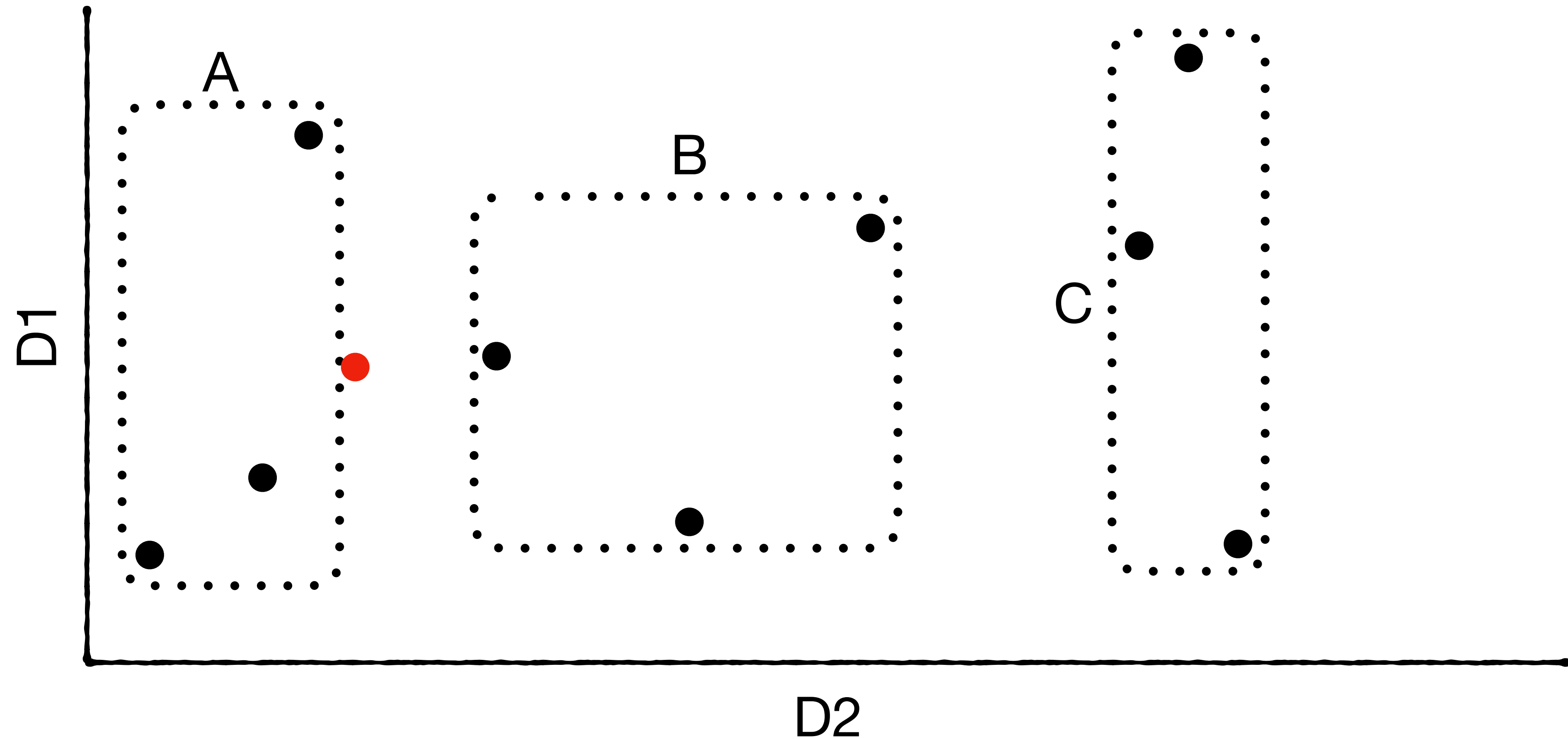
How to handle inserts? ●



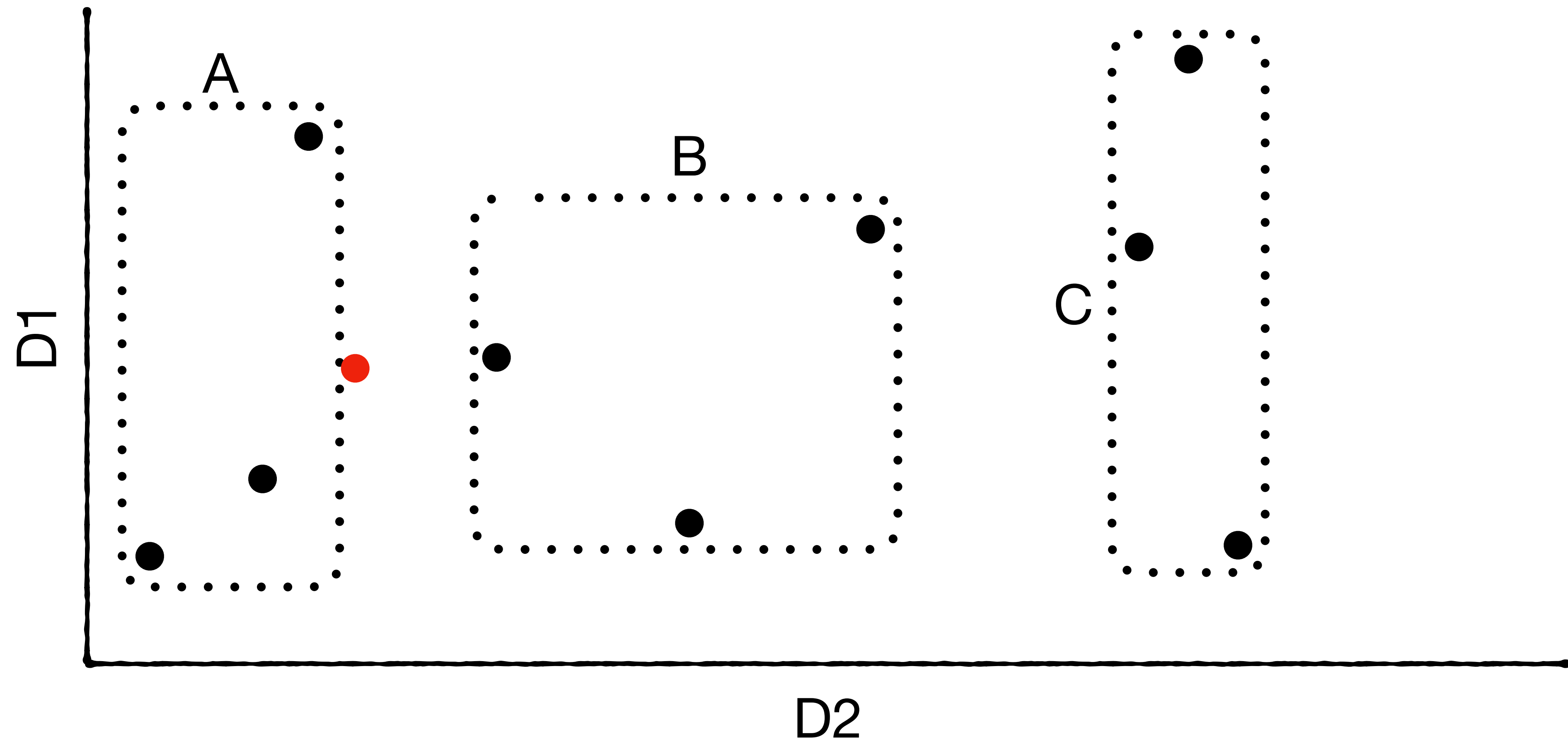
If point is in existing rectangle, add it to that rectangle



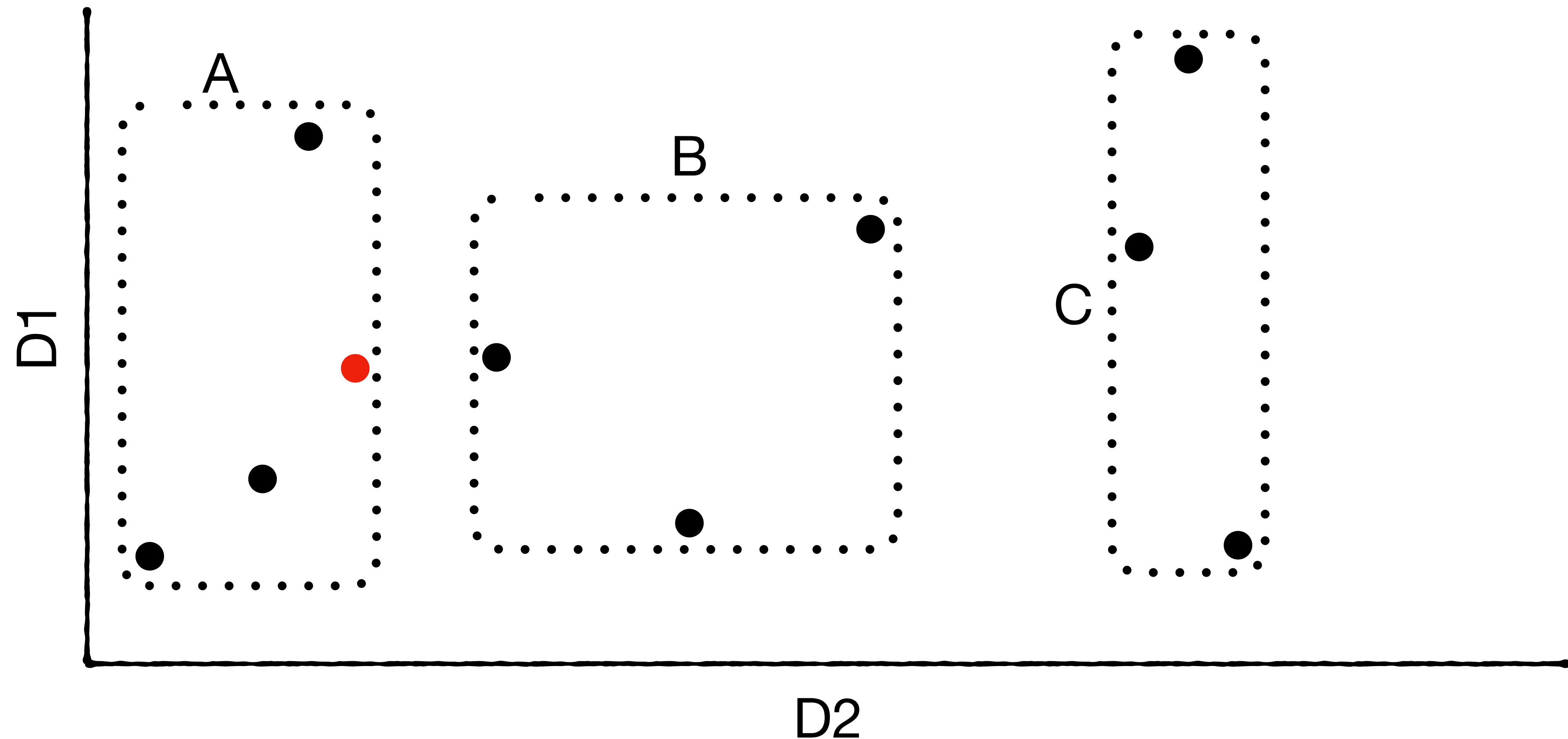
If point is in existing rectangle, add it to that rectangle
Otherwise?



If point is in existing rectangle, add it to that rectangle
Otherwise, add to rectangle we need to expand the least

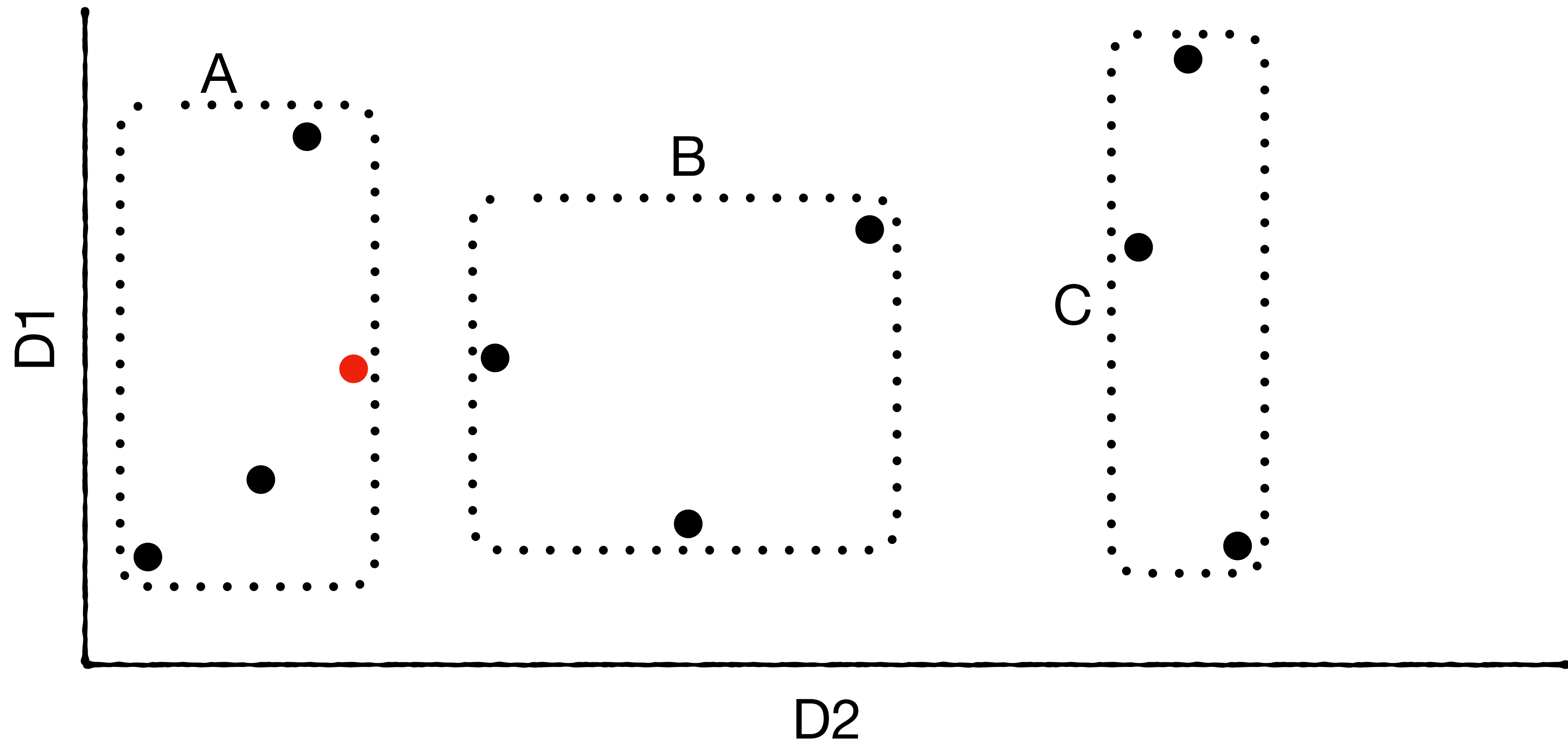


If point is in existing rectangle, add it to that rectangle
Otherwise, add to rectangle we need to expand the least

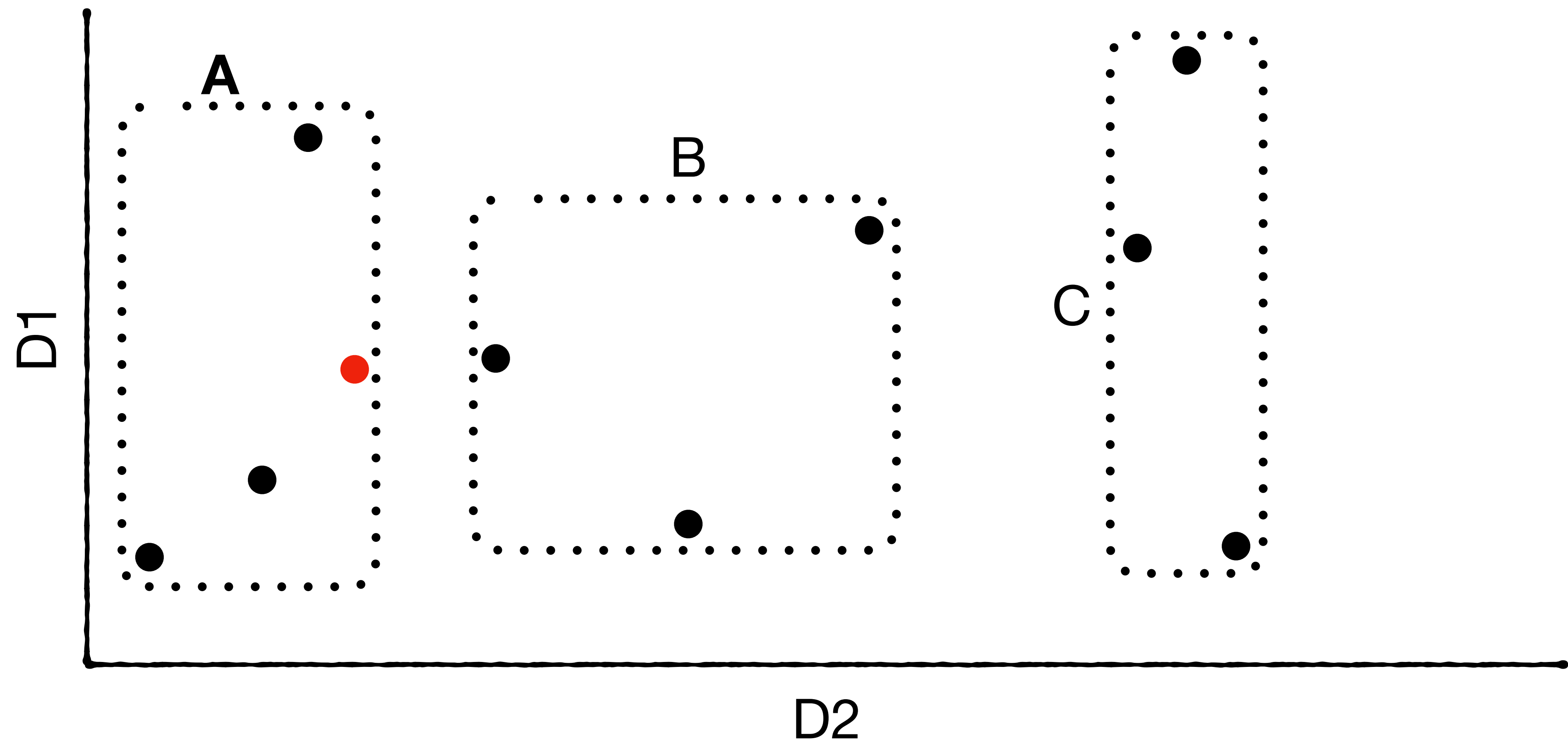


If point is in existing rectangle, add it to that rectangle
Otherwise, add to rectangle we need to expand the least

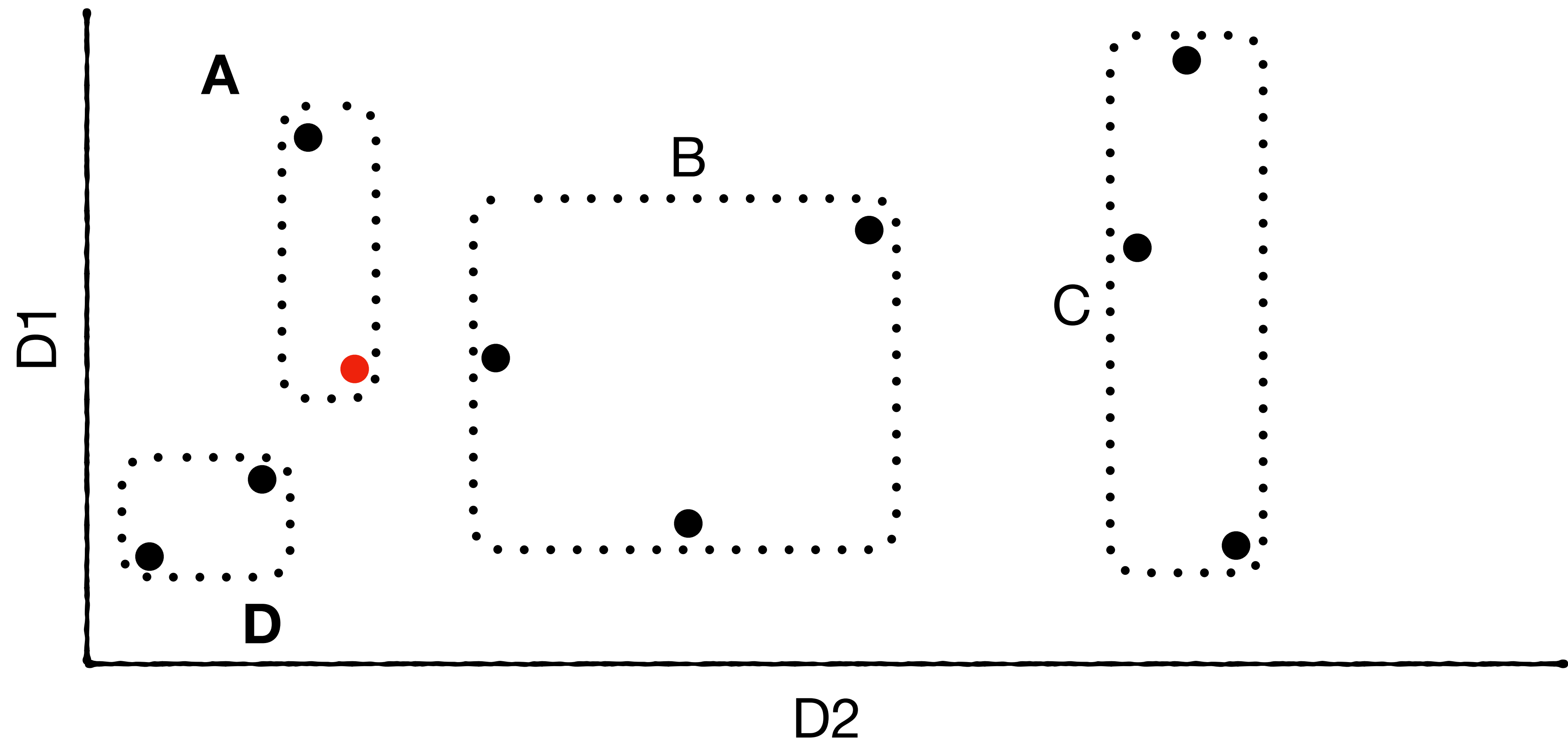
Why? To minimize coverage



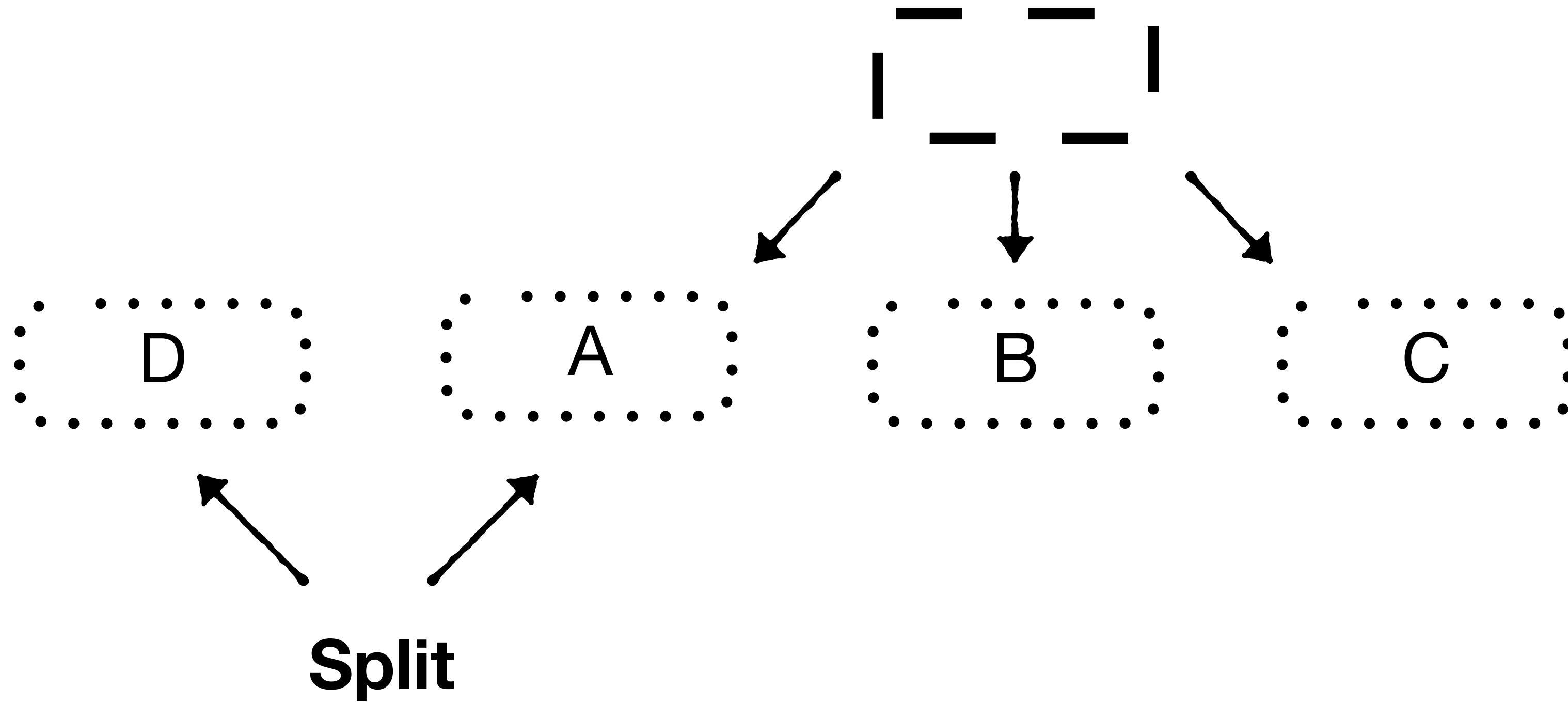
If a rectangle now contains B entries, split it (e.g., B=4)



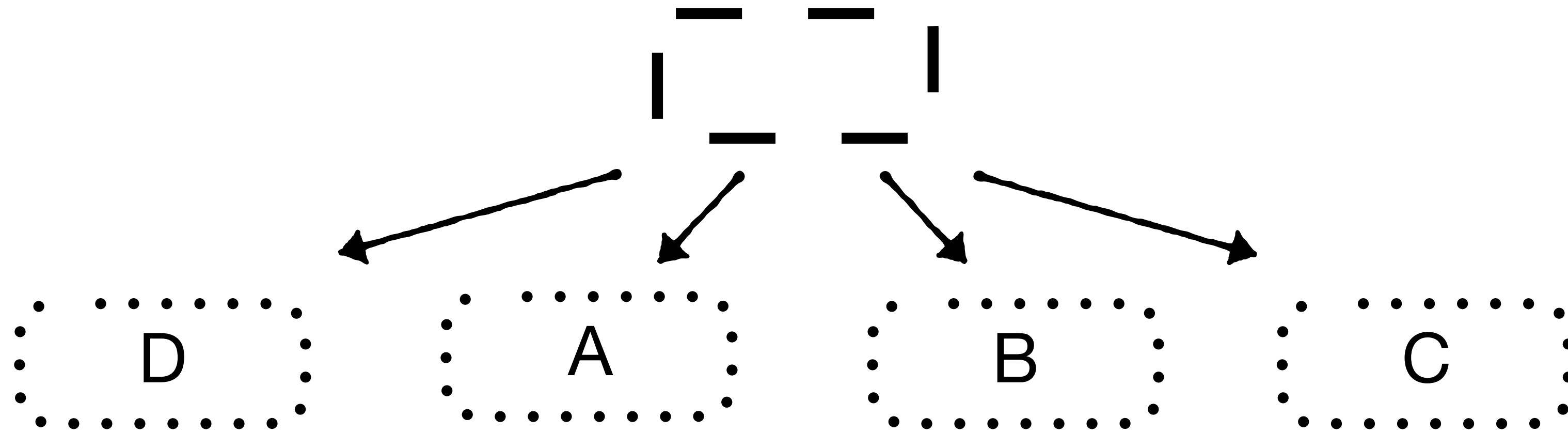
If a rectangle now contains B entries, split it (e.g., B=4)



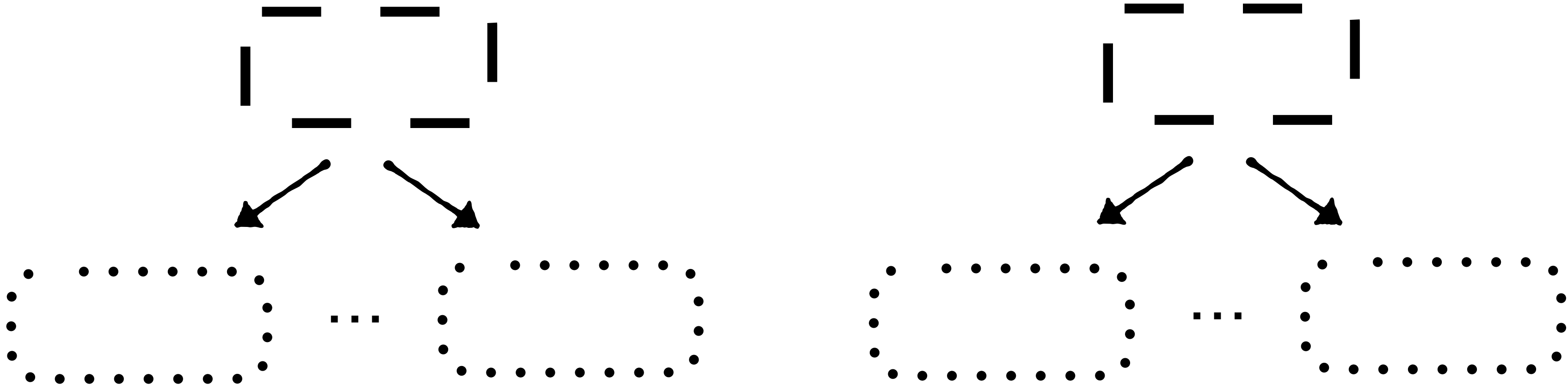
Physically Structured as a B-Tree



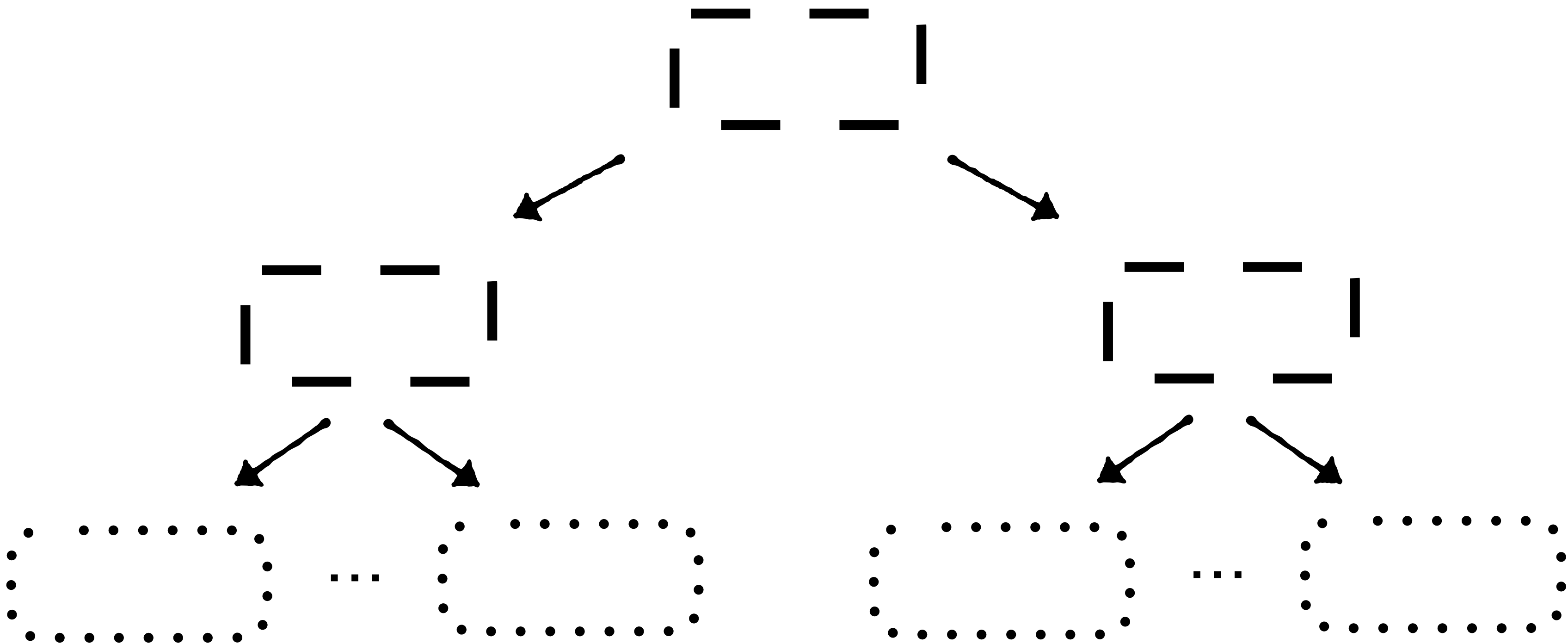
Add pointer in parent



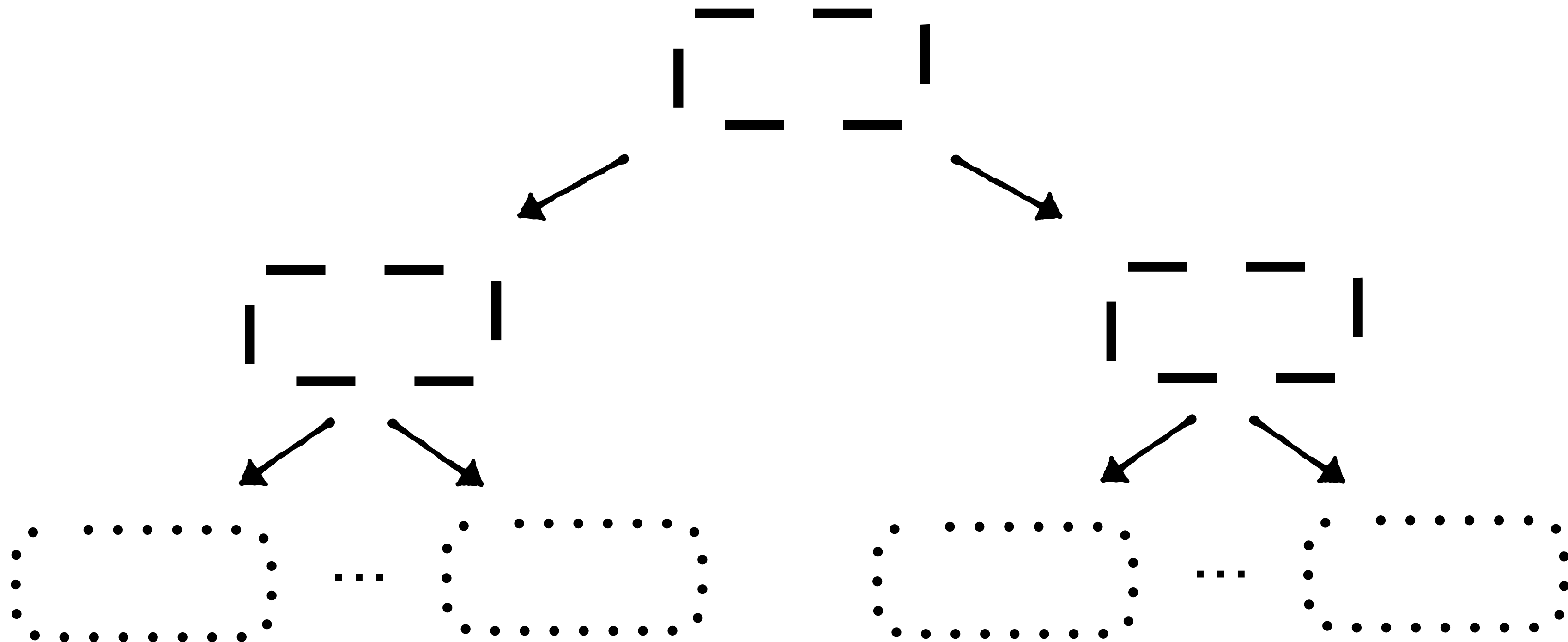
Parent may split as well



Add new parent, causing depth to grow

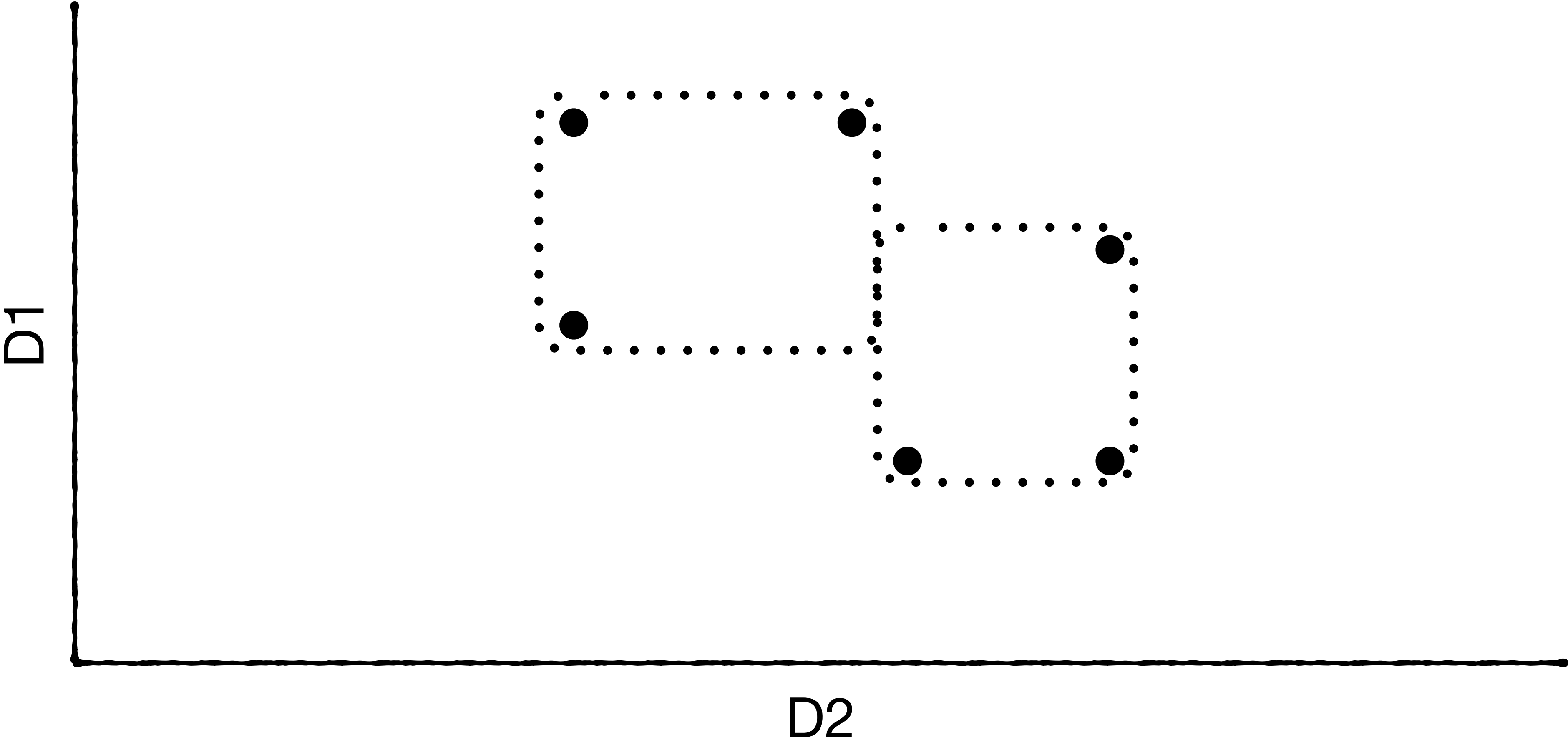


Add new parent, causing depth to grow

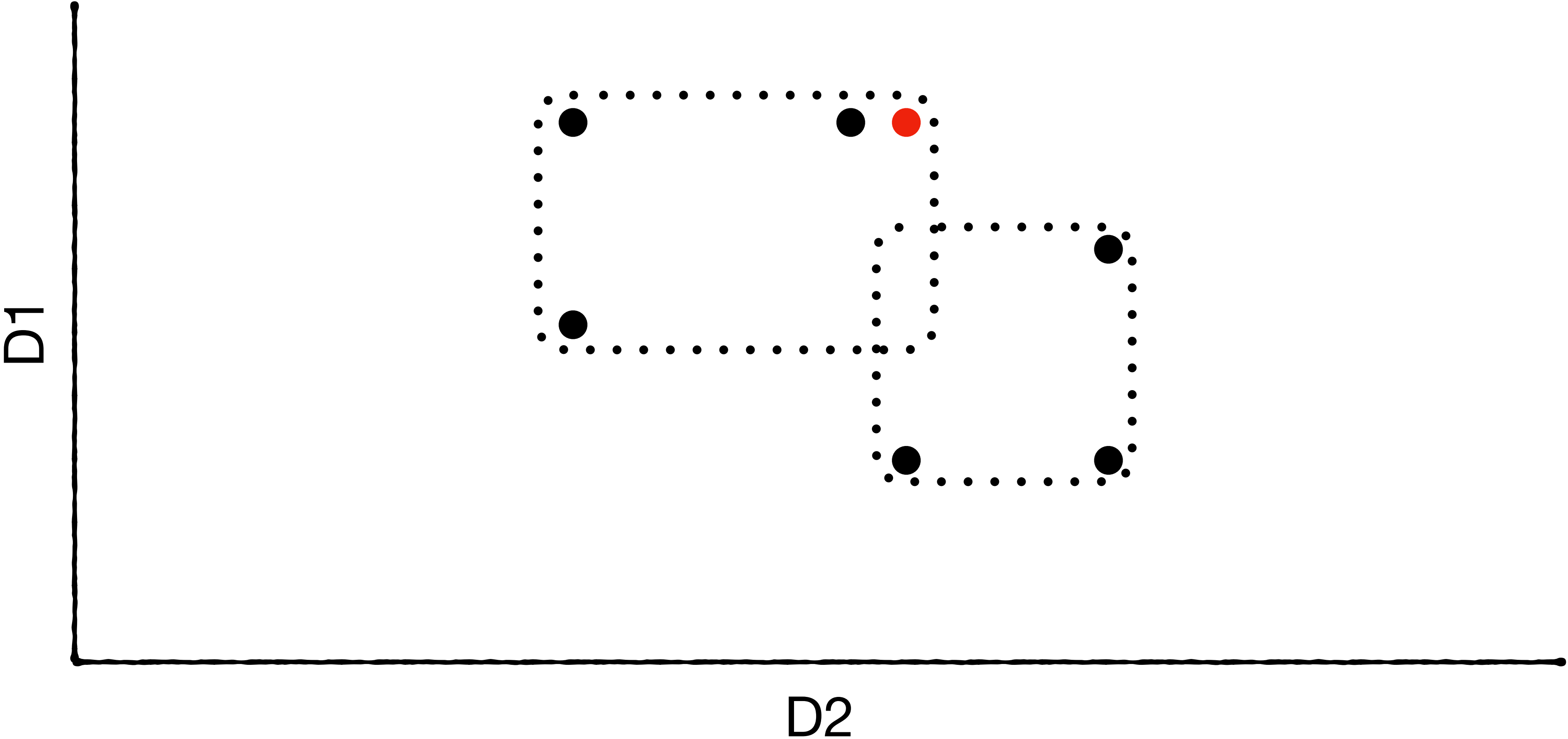


The tree stays perfectly balanced

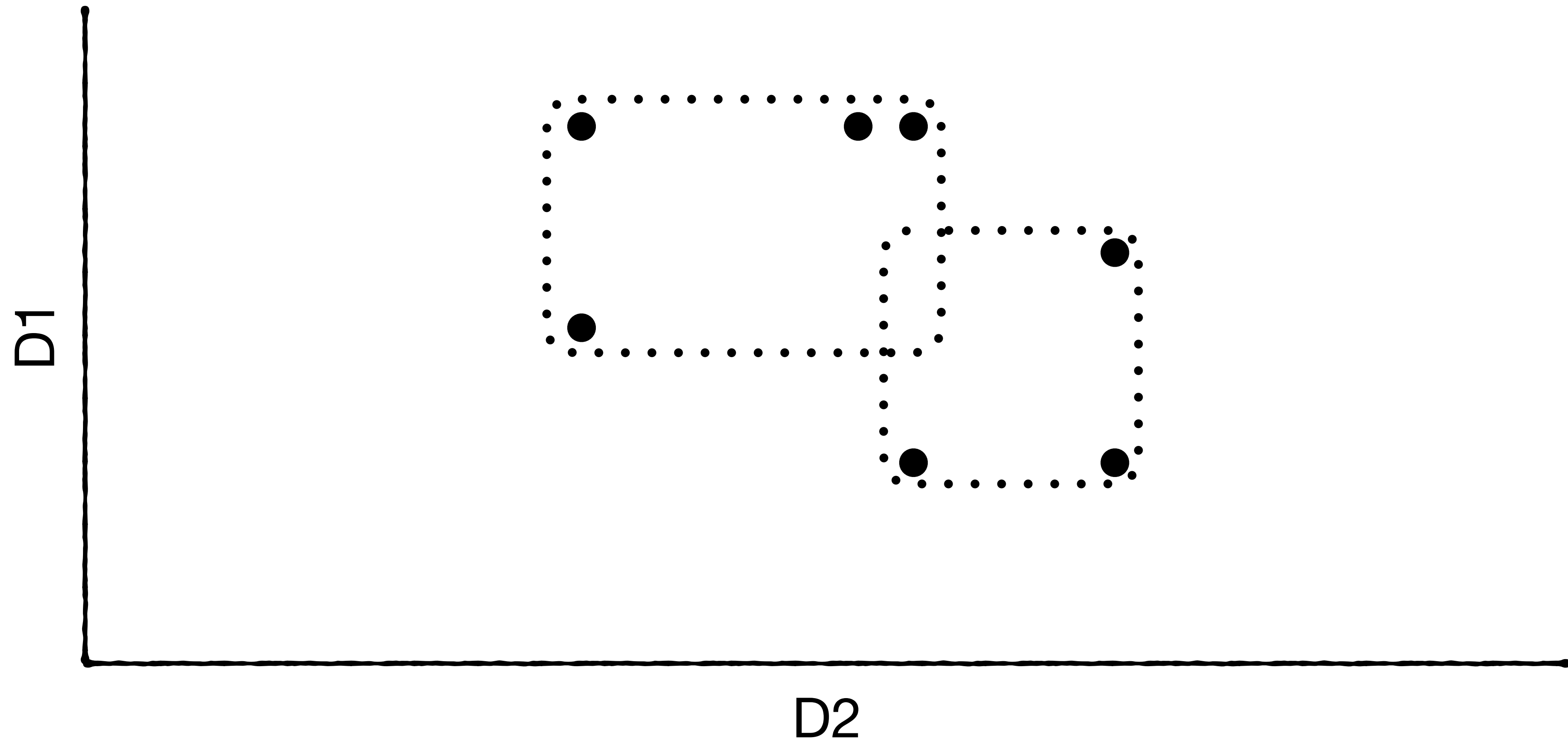
Insertions may cause overlap across rectangles ●



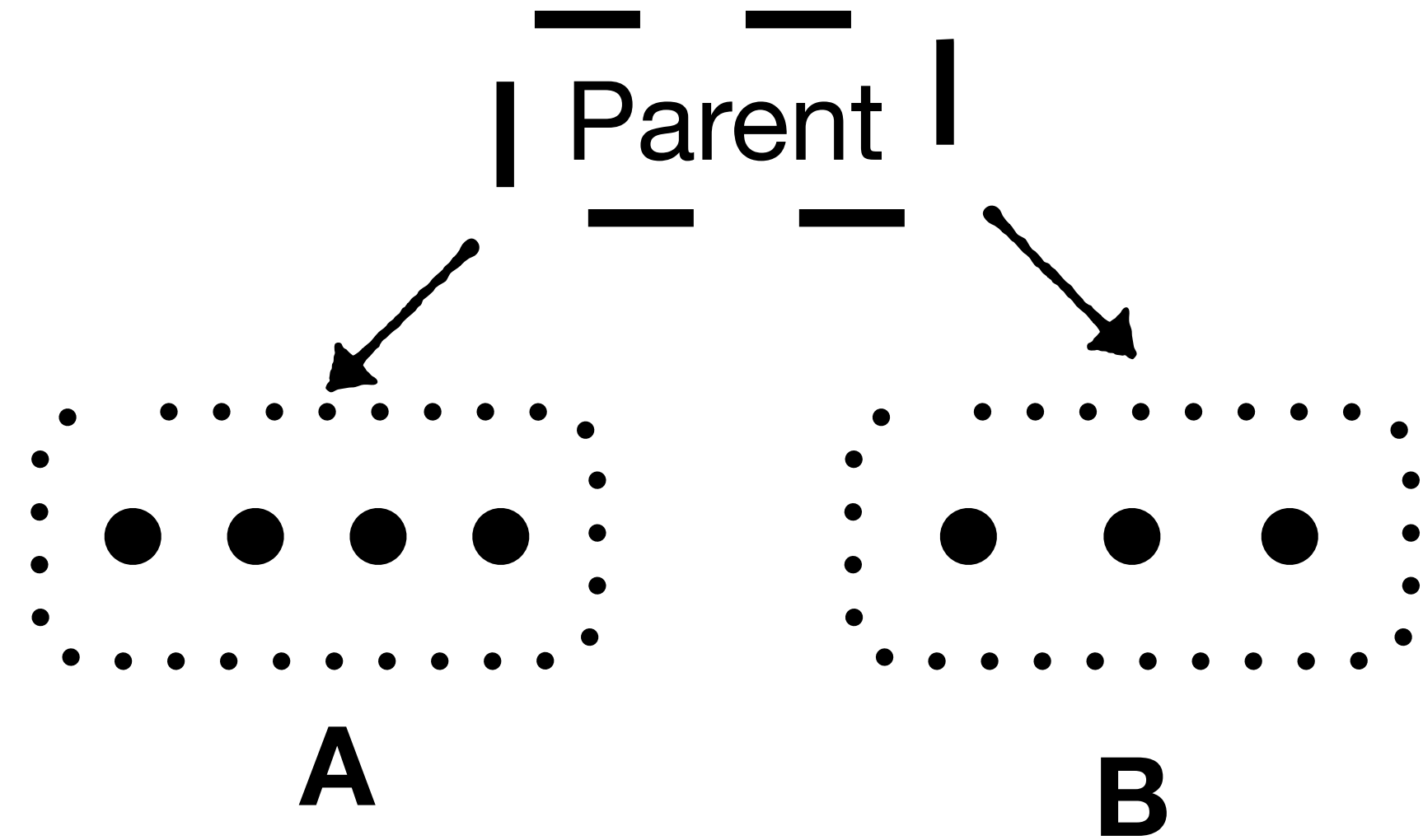
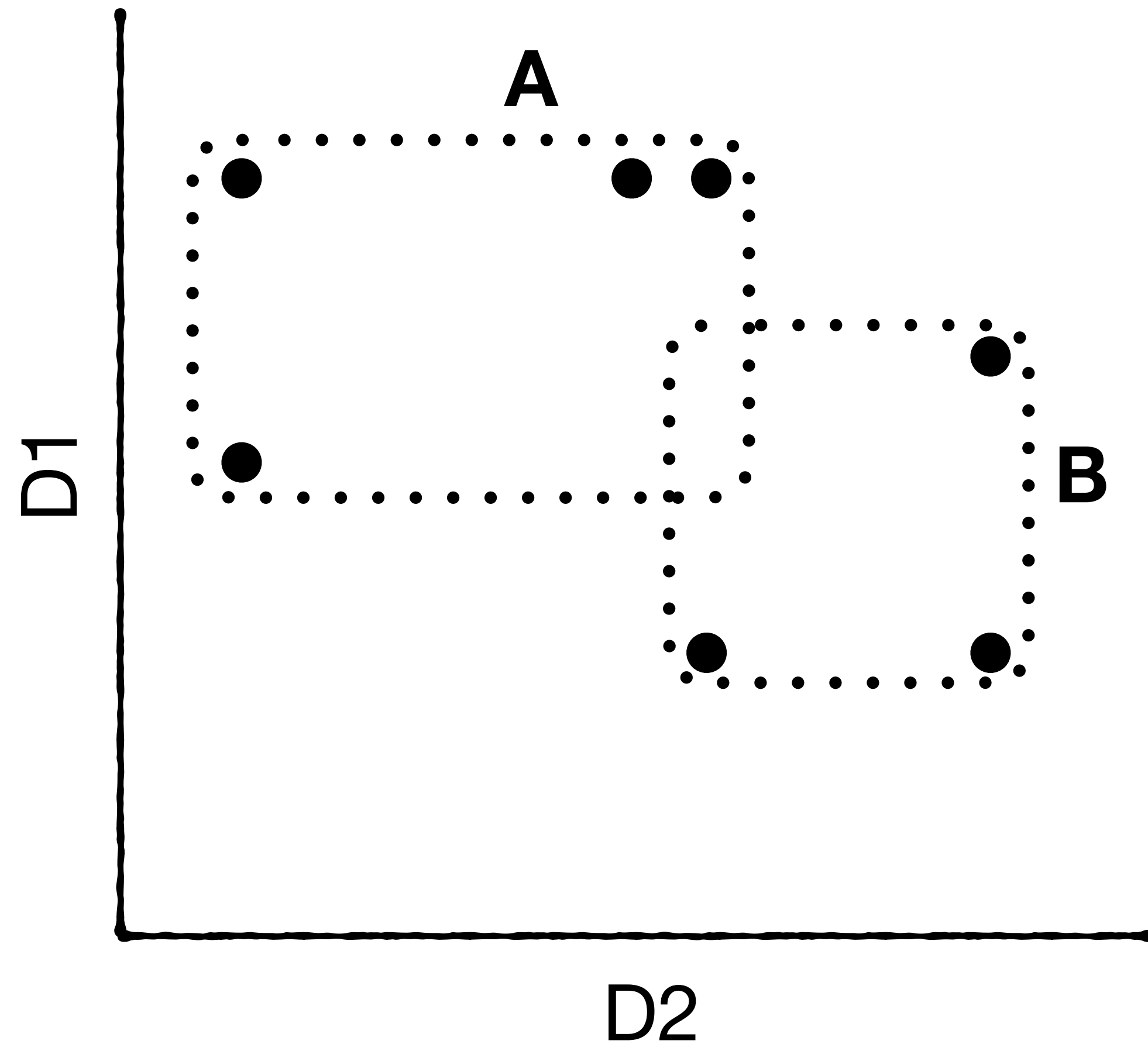
Insertions may cause overlap across rectangles



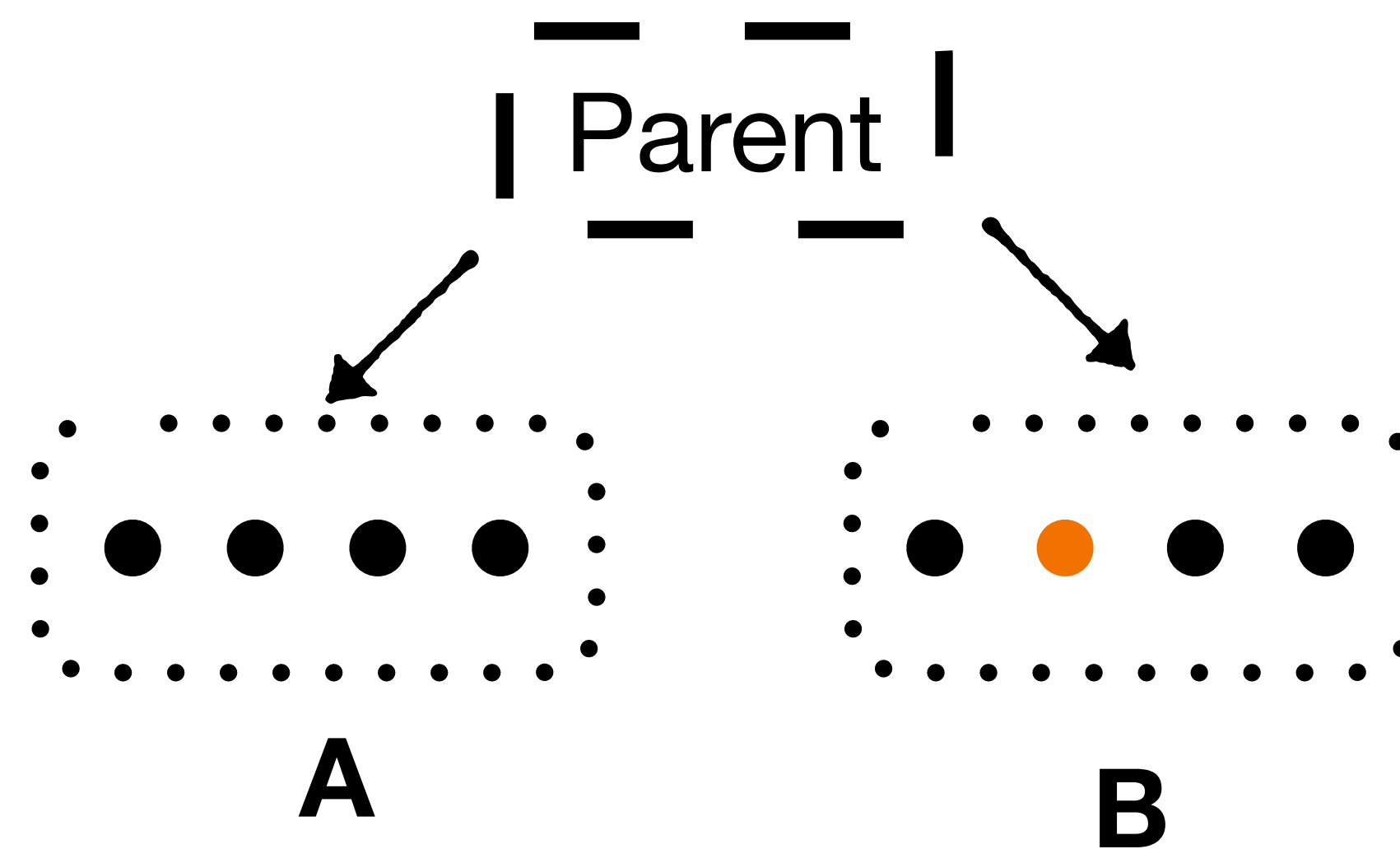
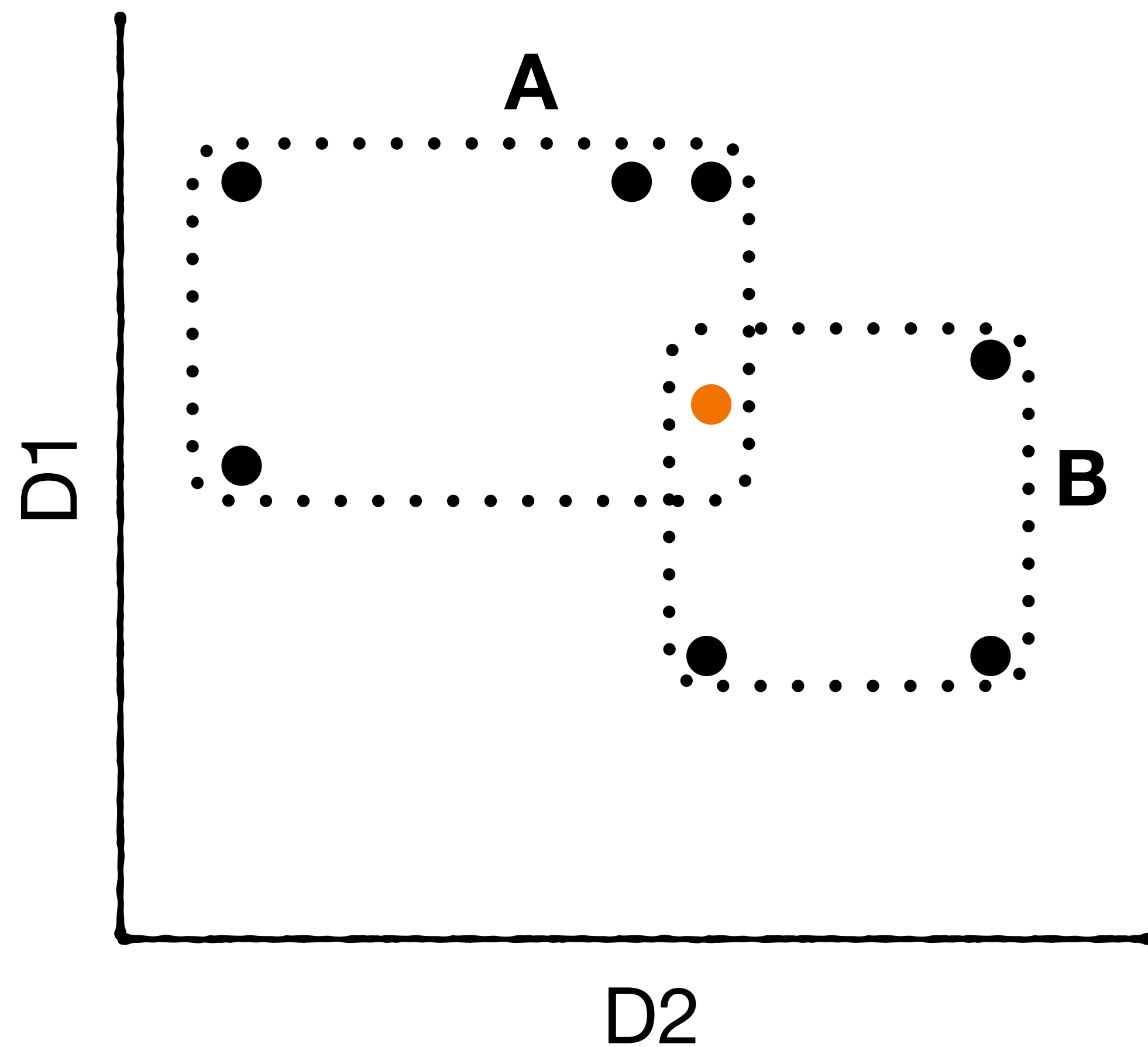
An entry may be inserted into an intersection ●

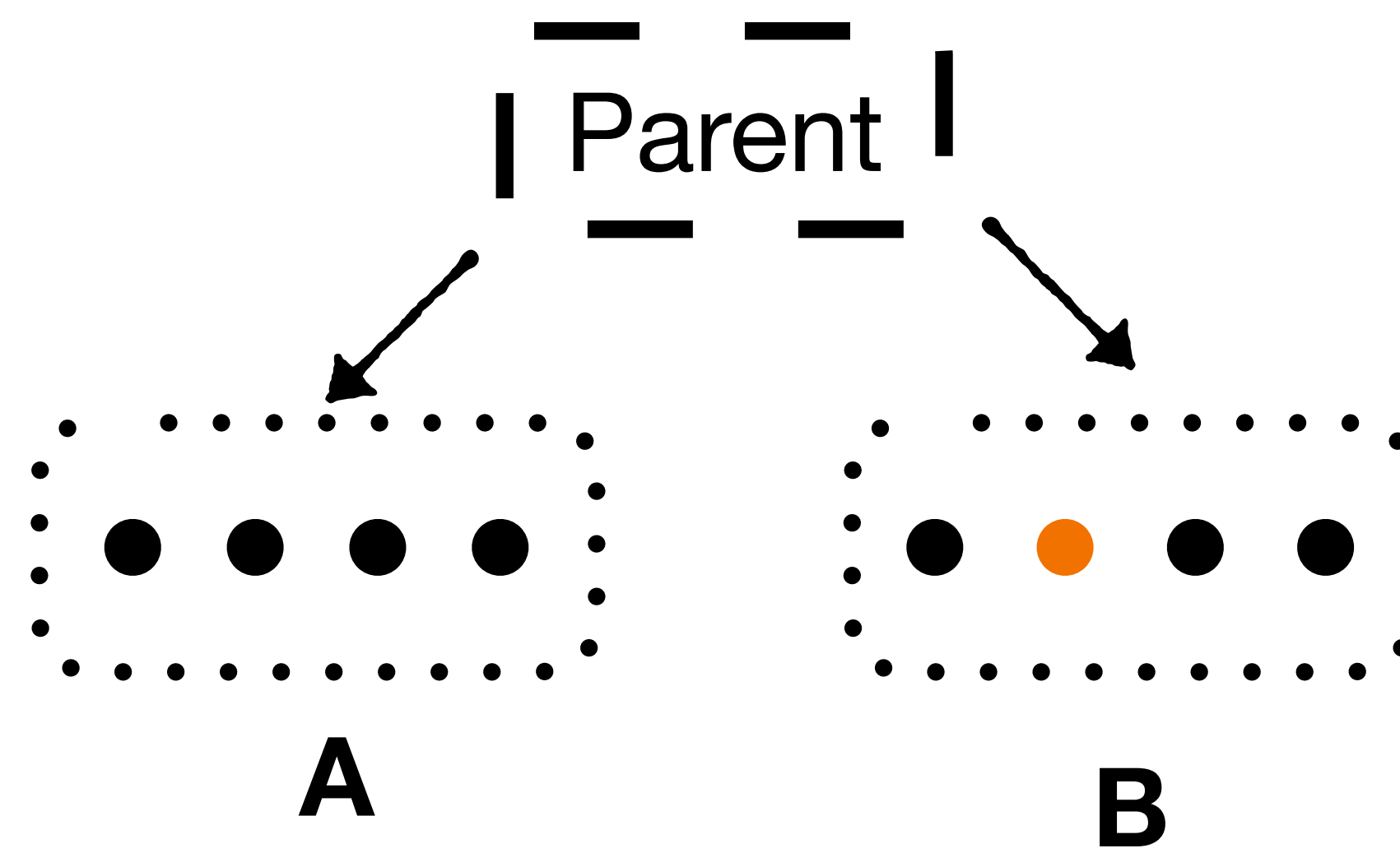
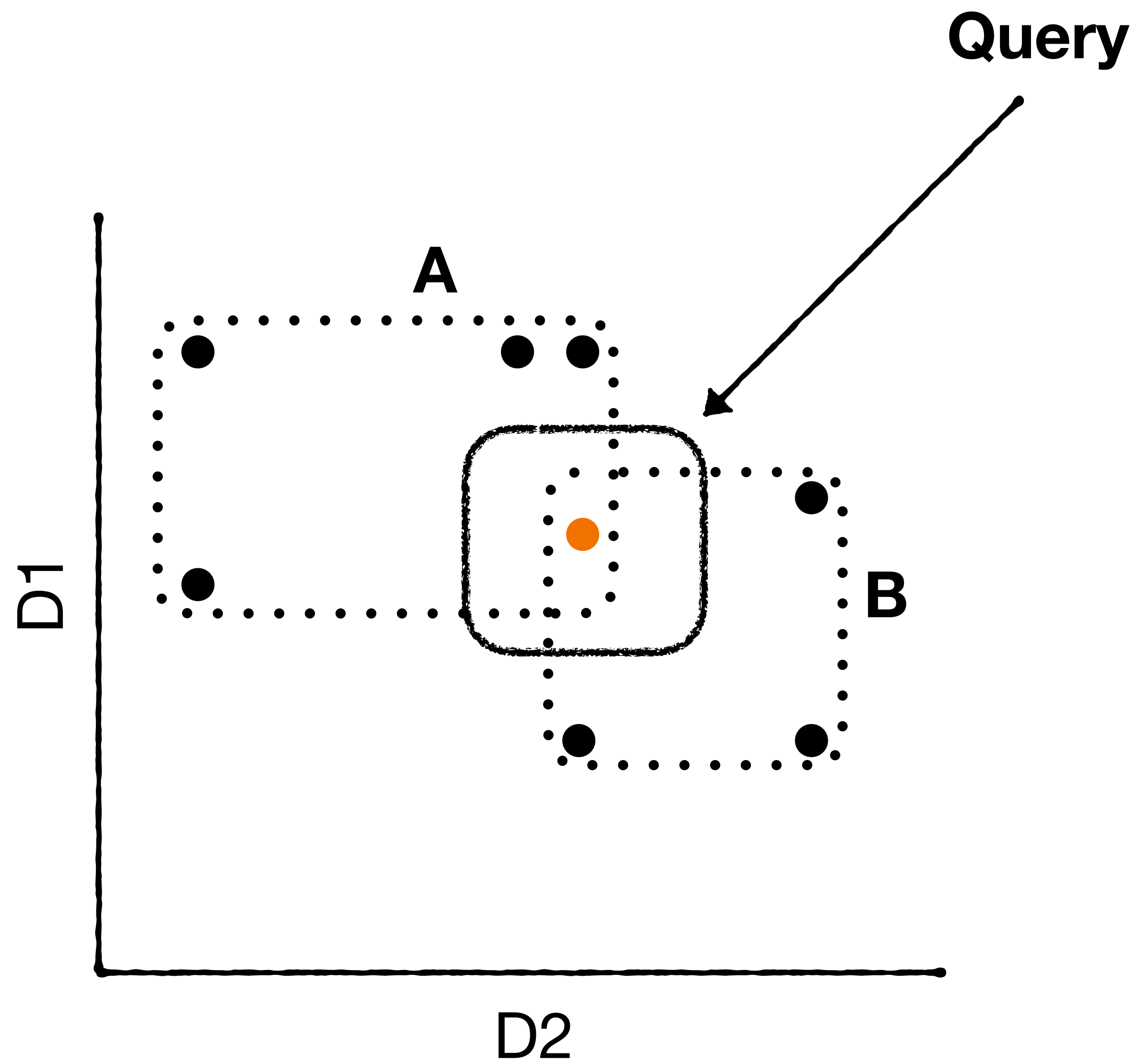


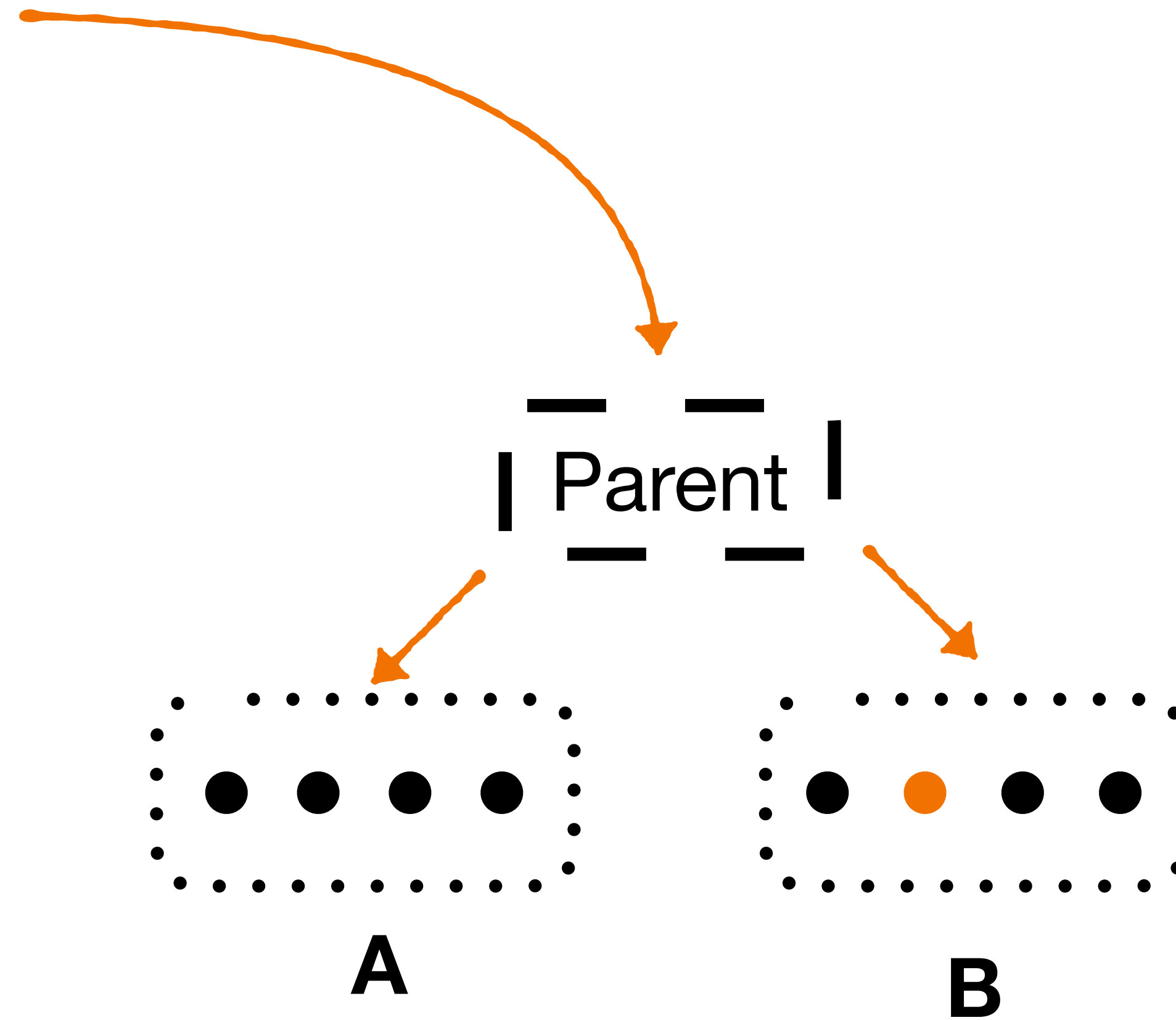
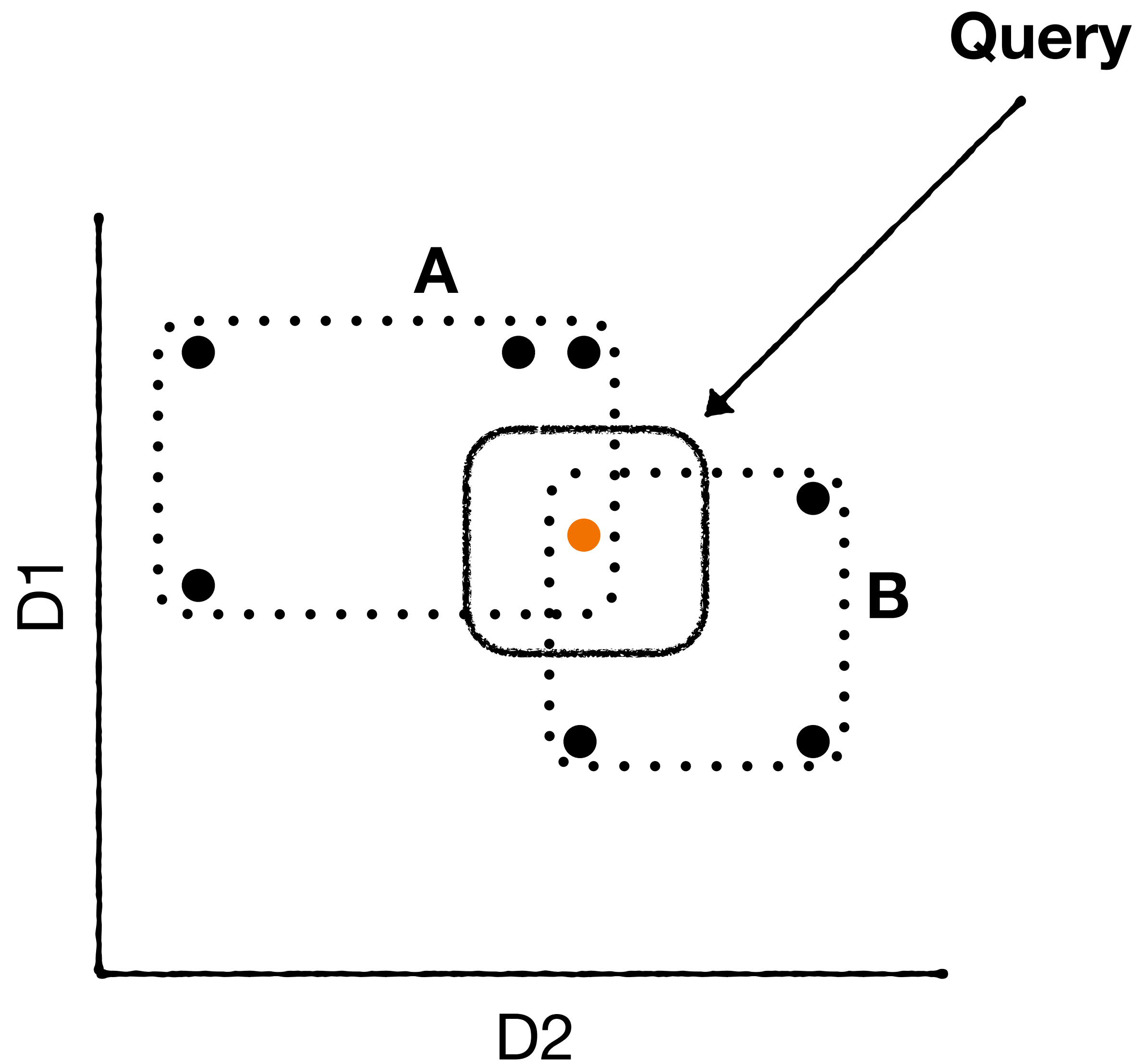
An entry may be inserted into an intersection ●

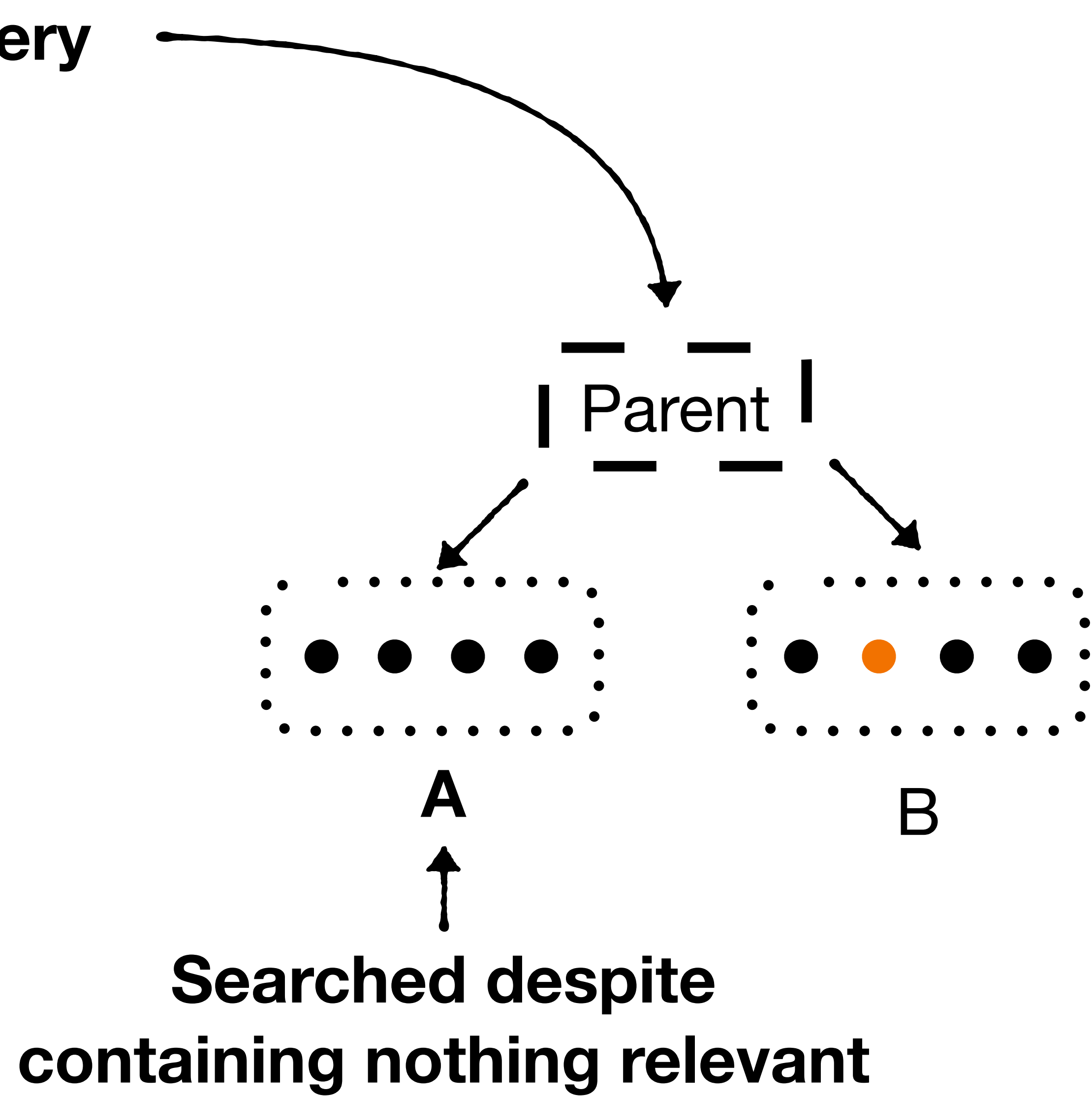
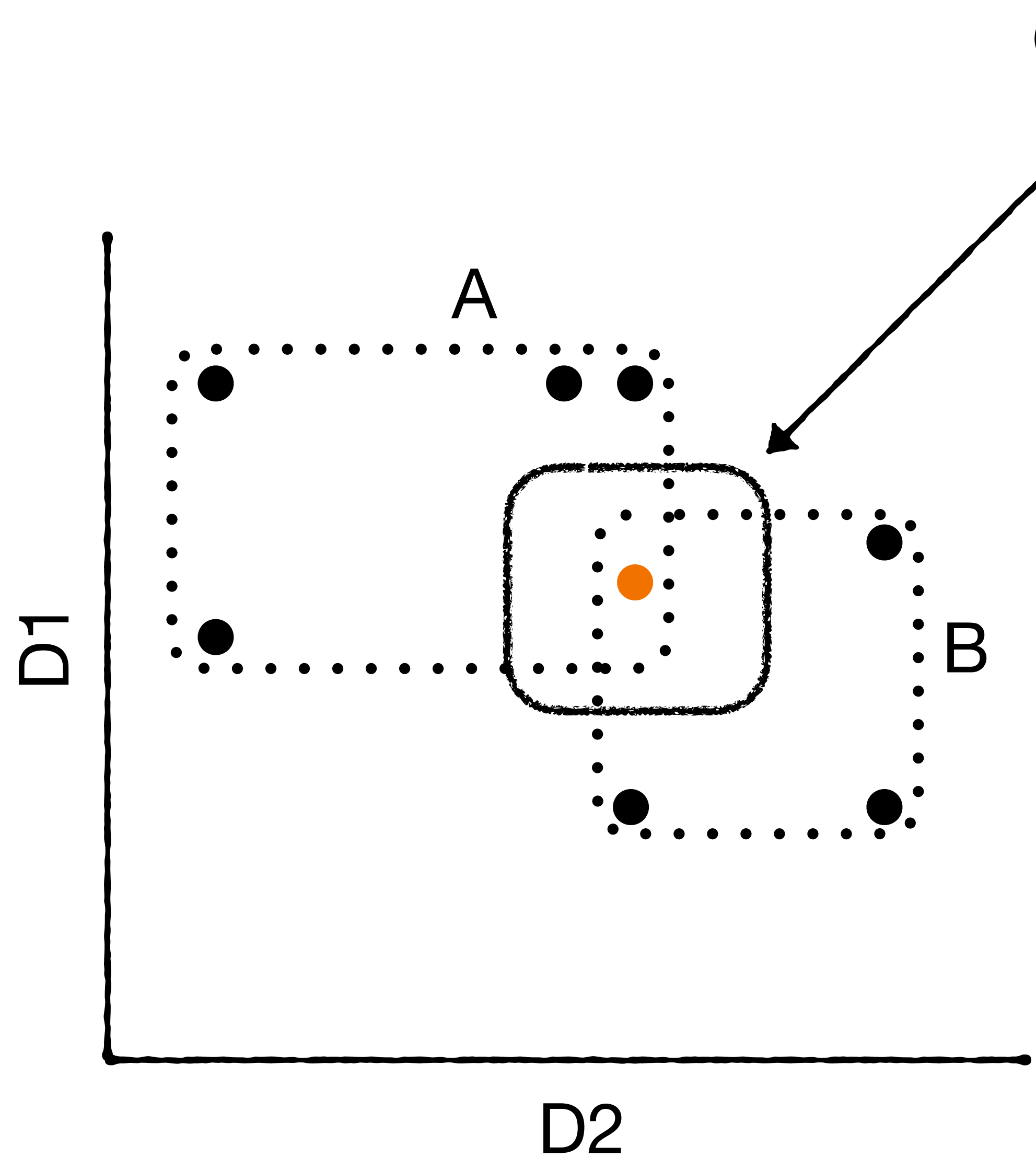


An entry may be inserted into an intersection



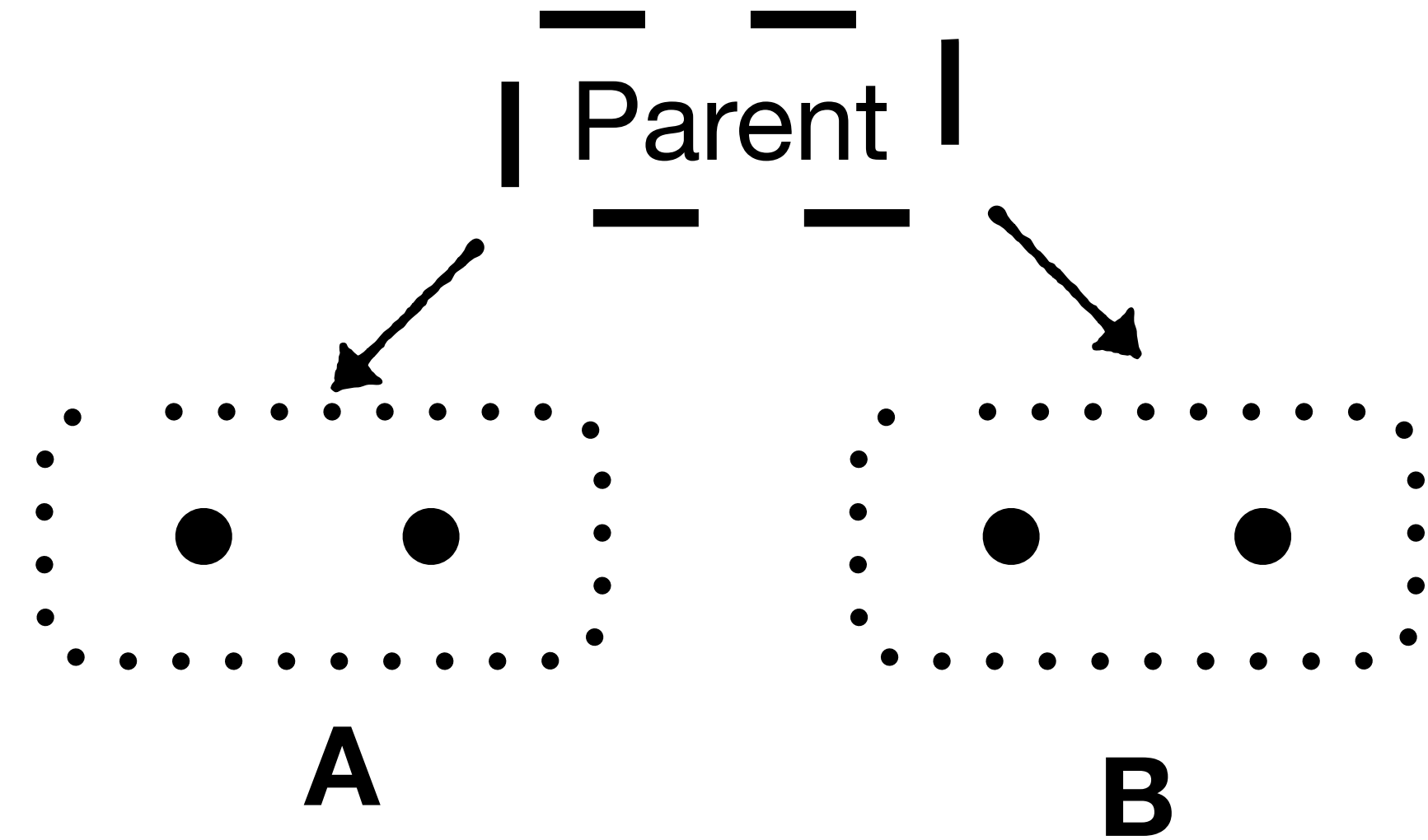
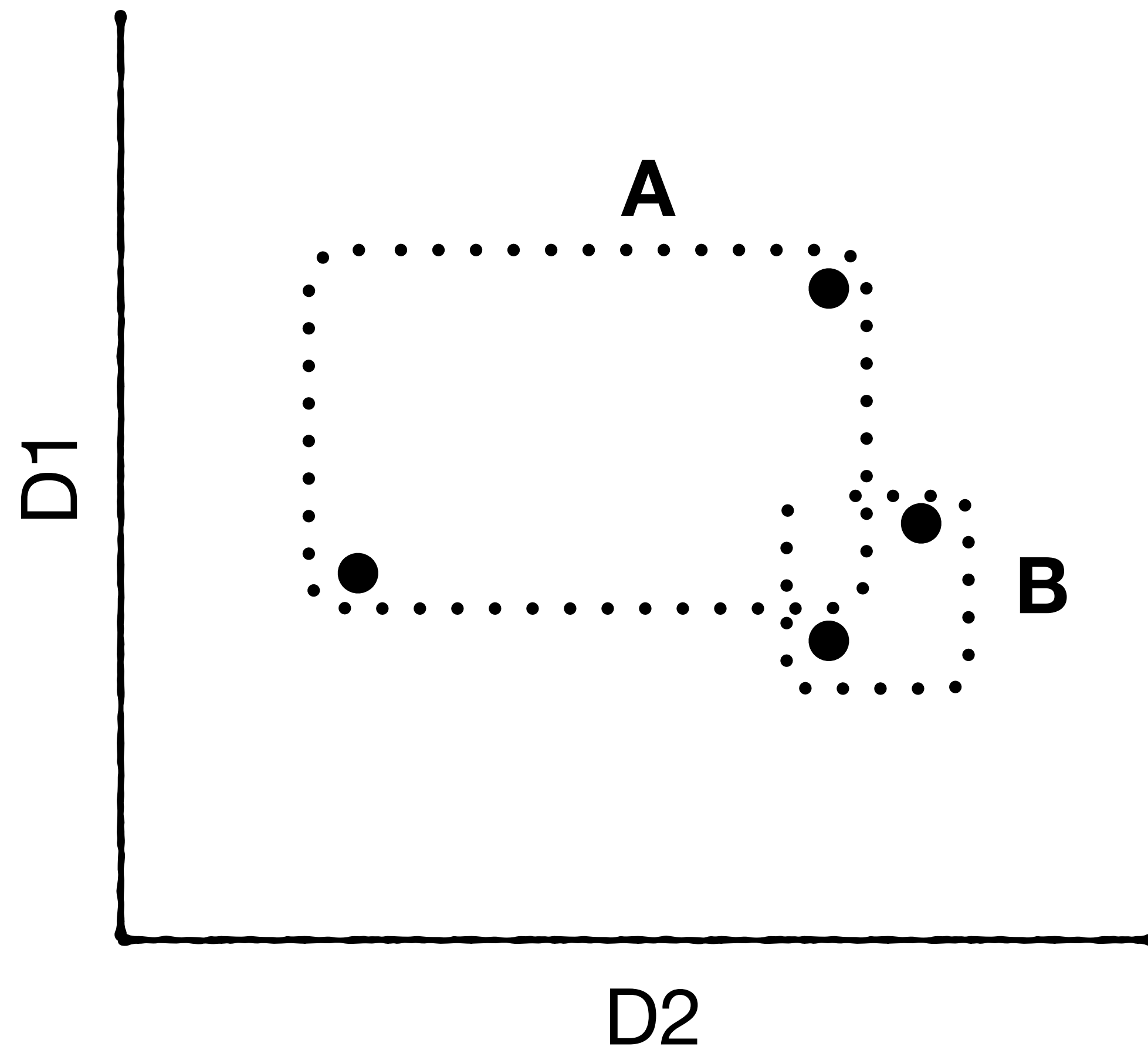




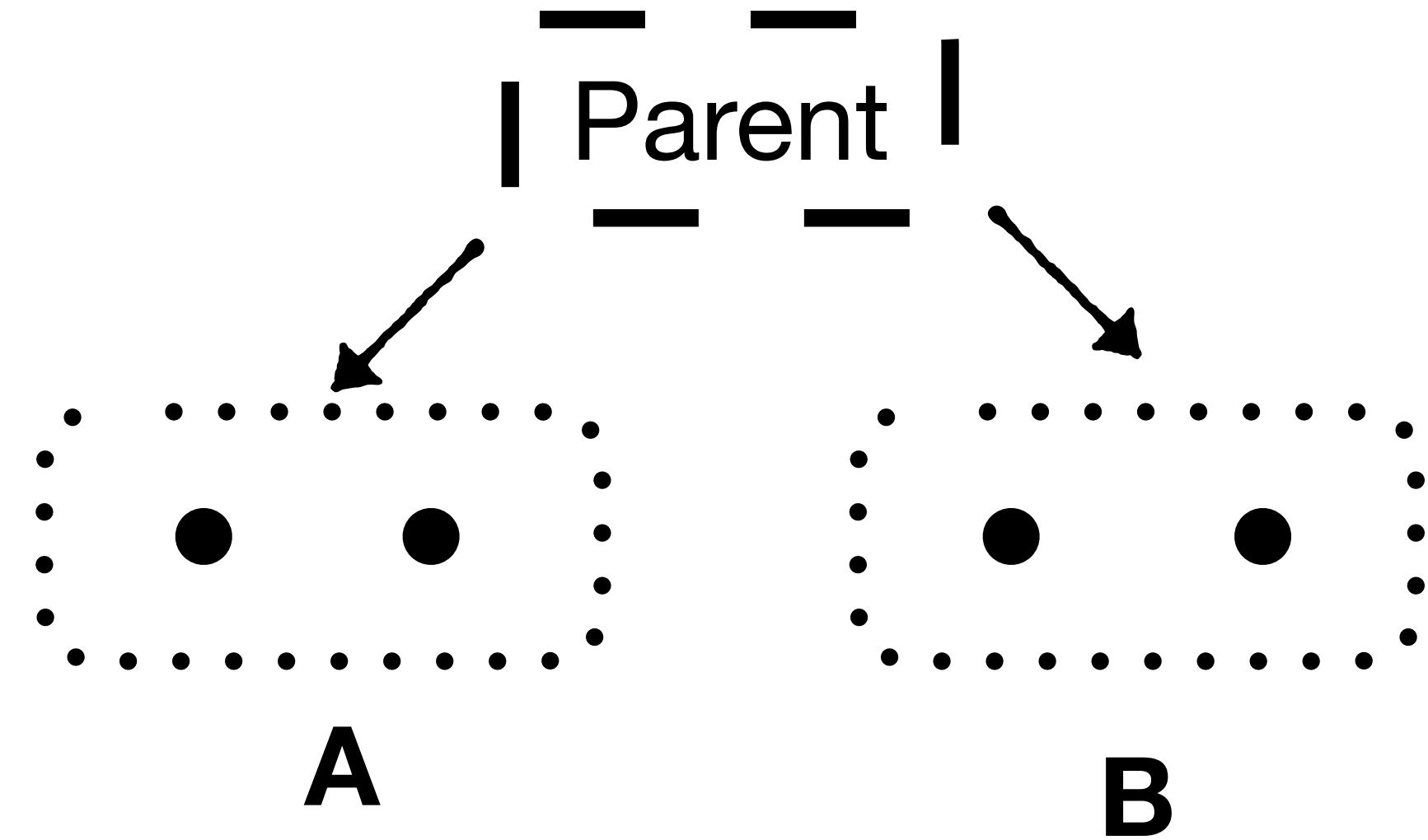
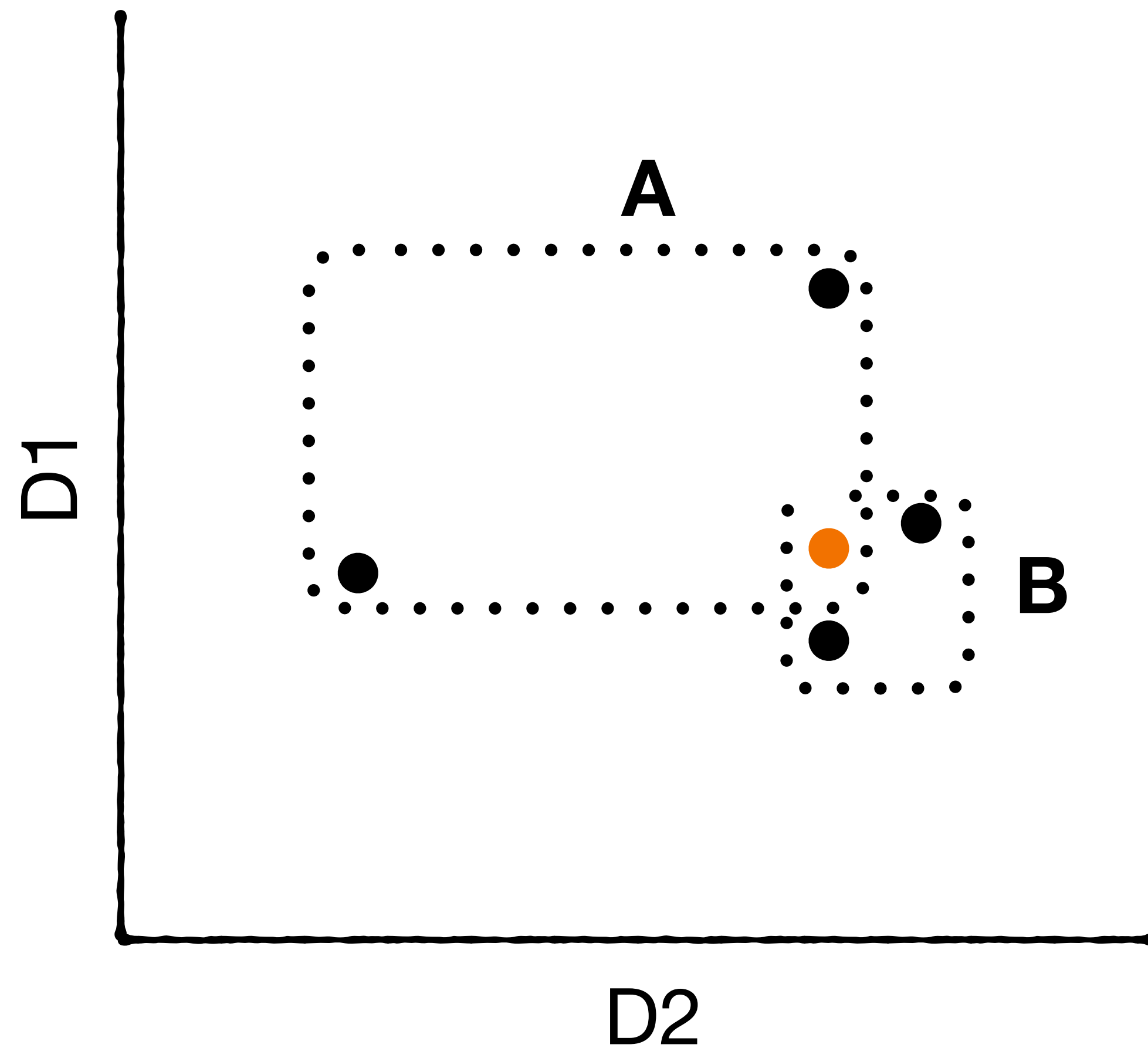


Dual goal: maintain R-tree with least coverage & intersections :)

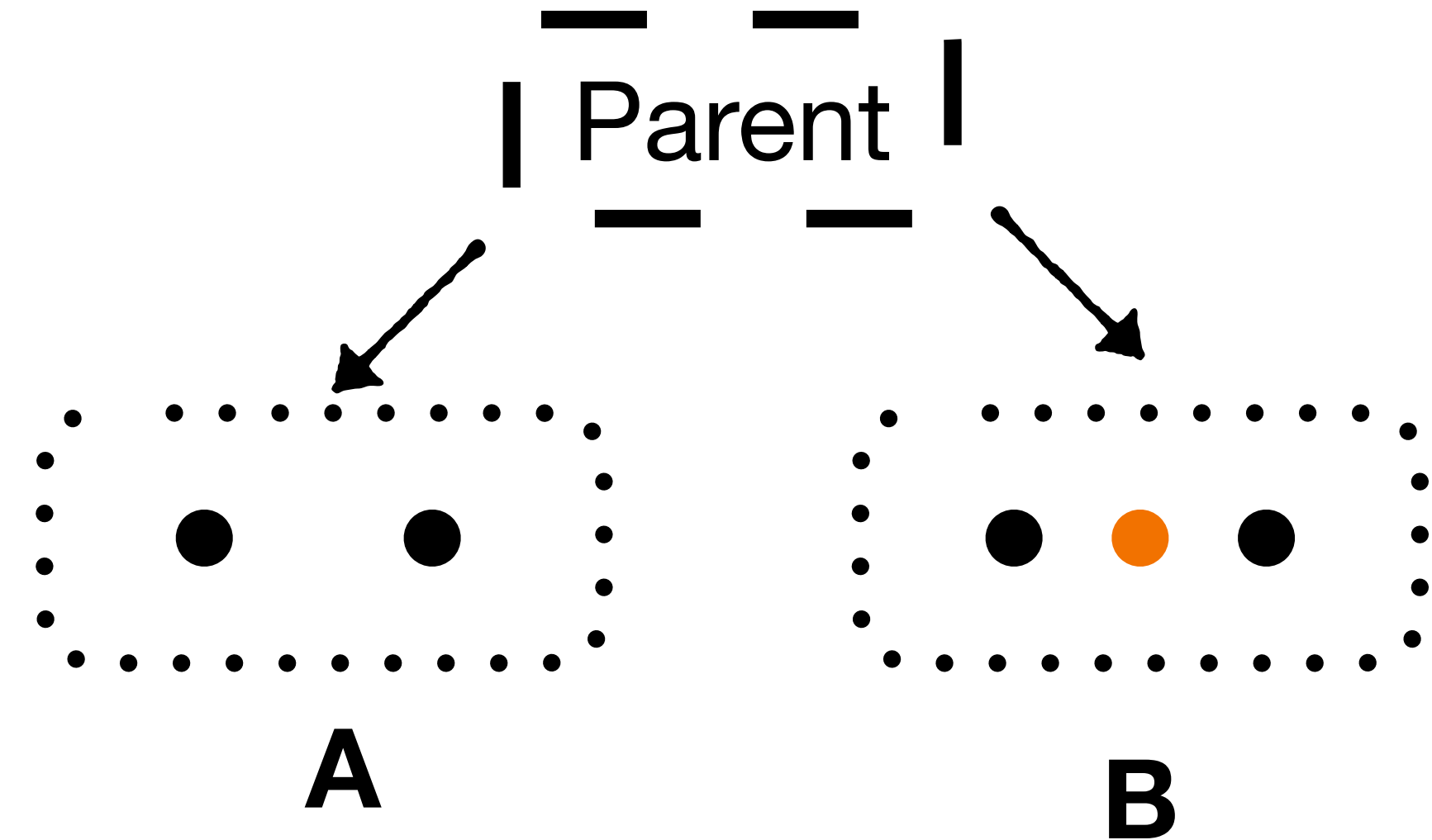
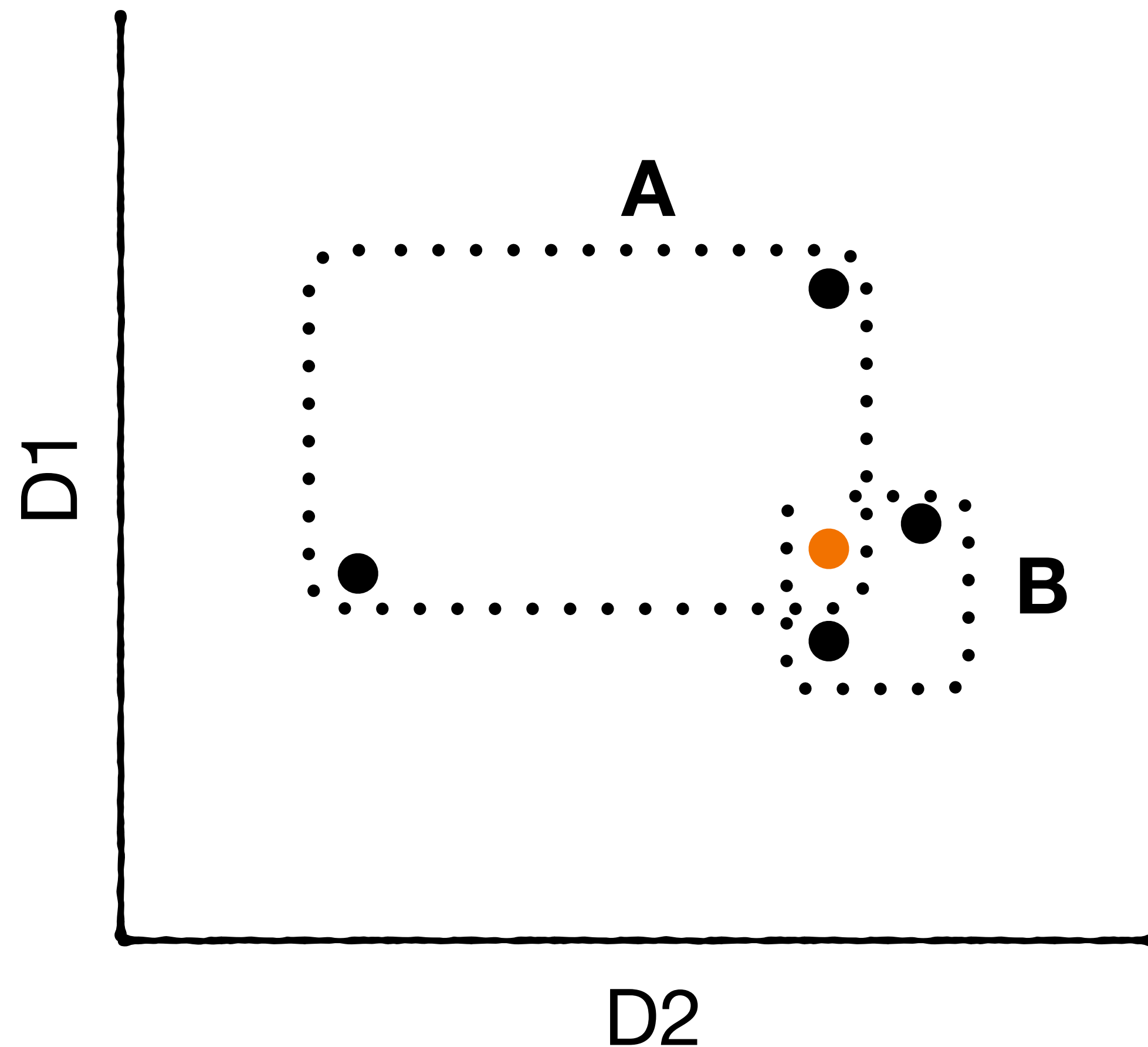
Suppose we insert entry into interdiction



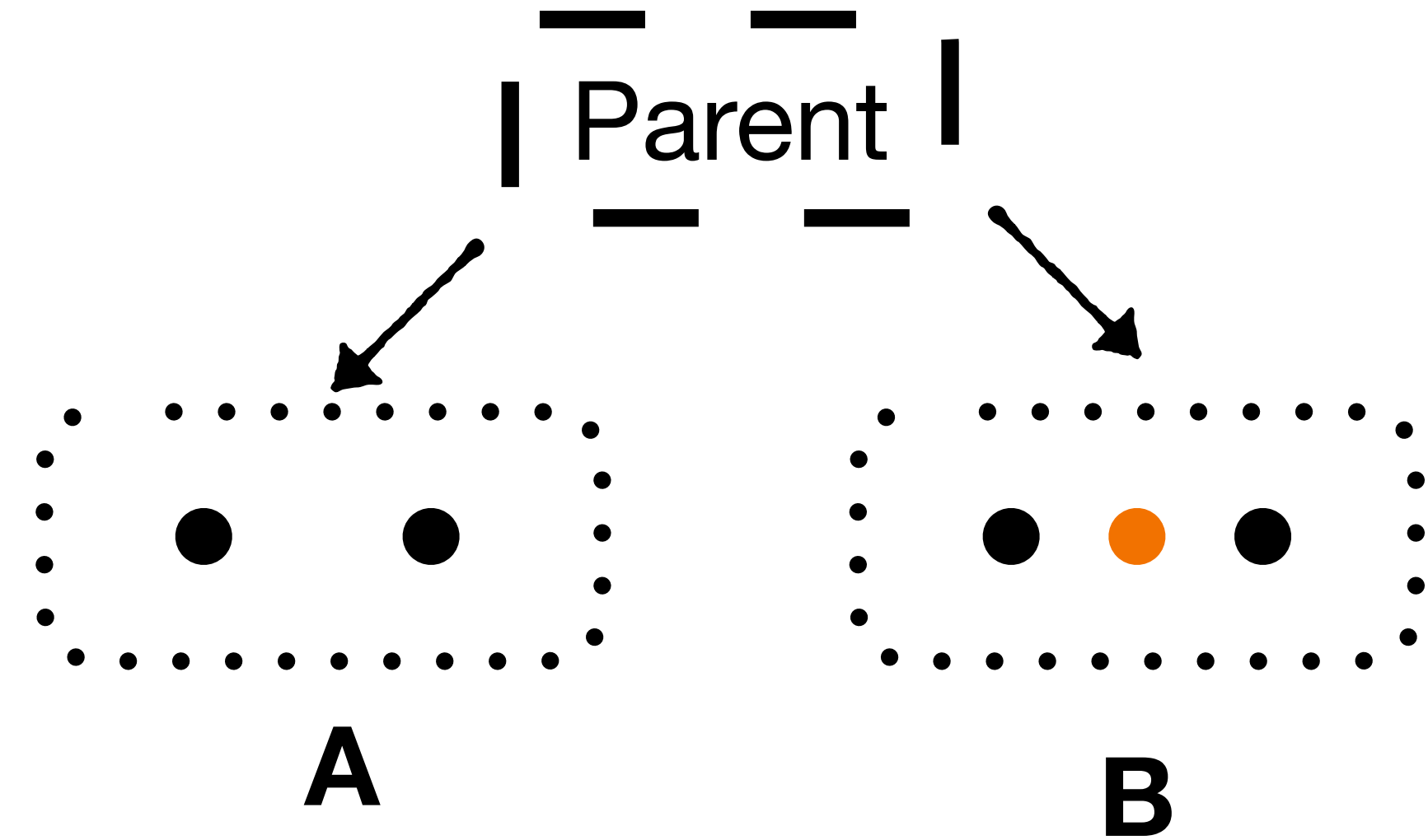
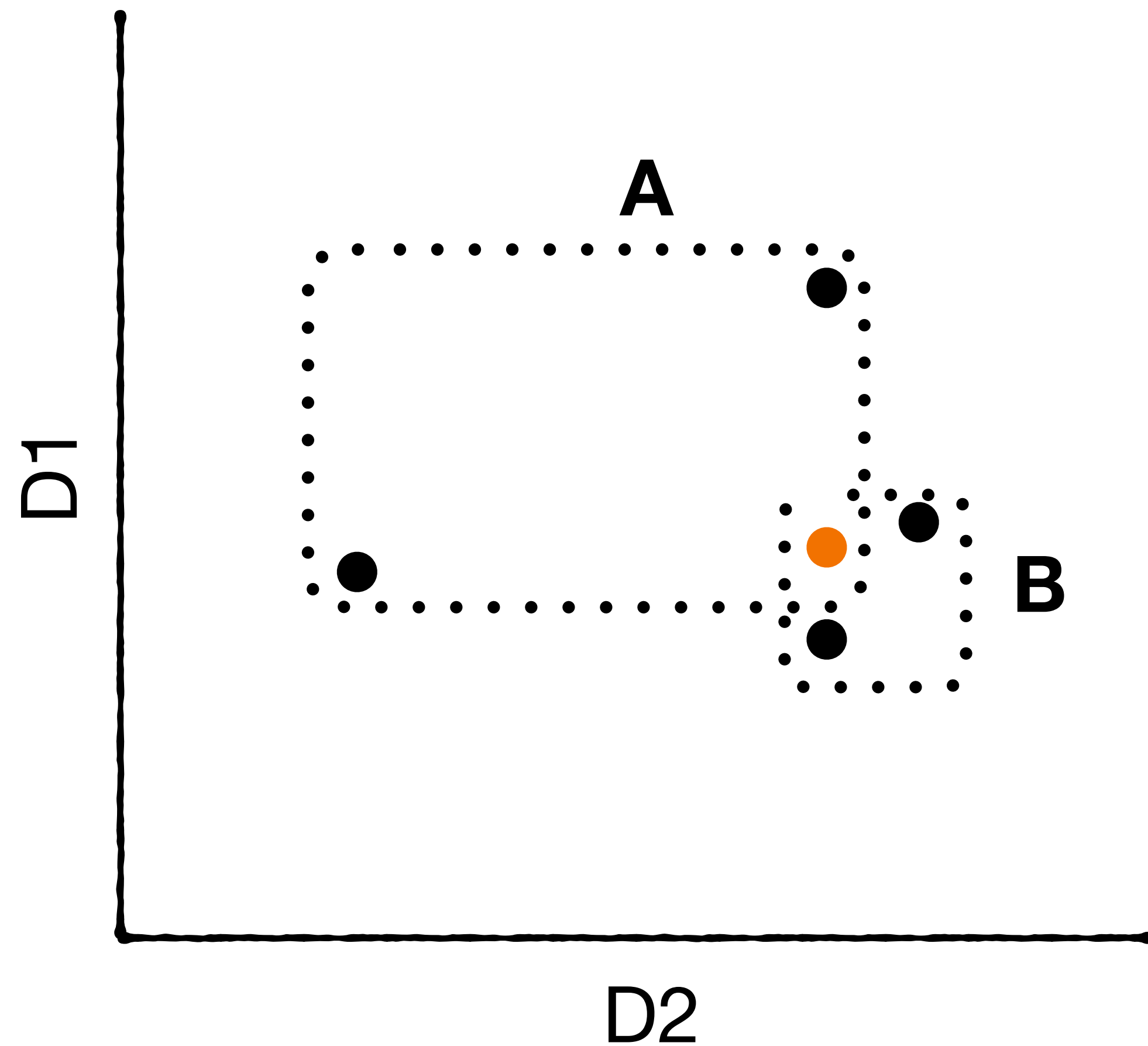
Suppose we insert entry into interdiction
To which node should we add it?



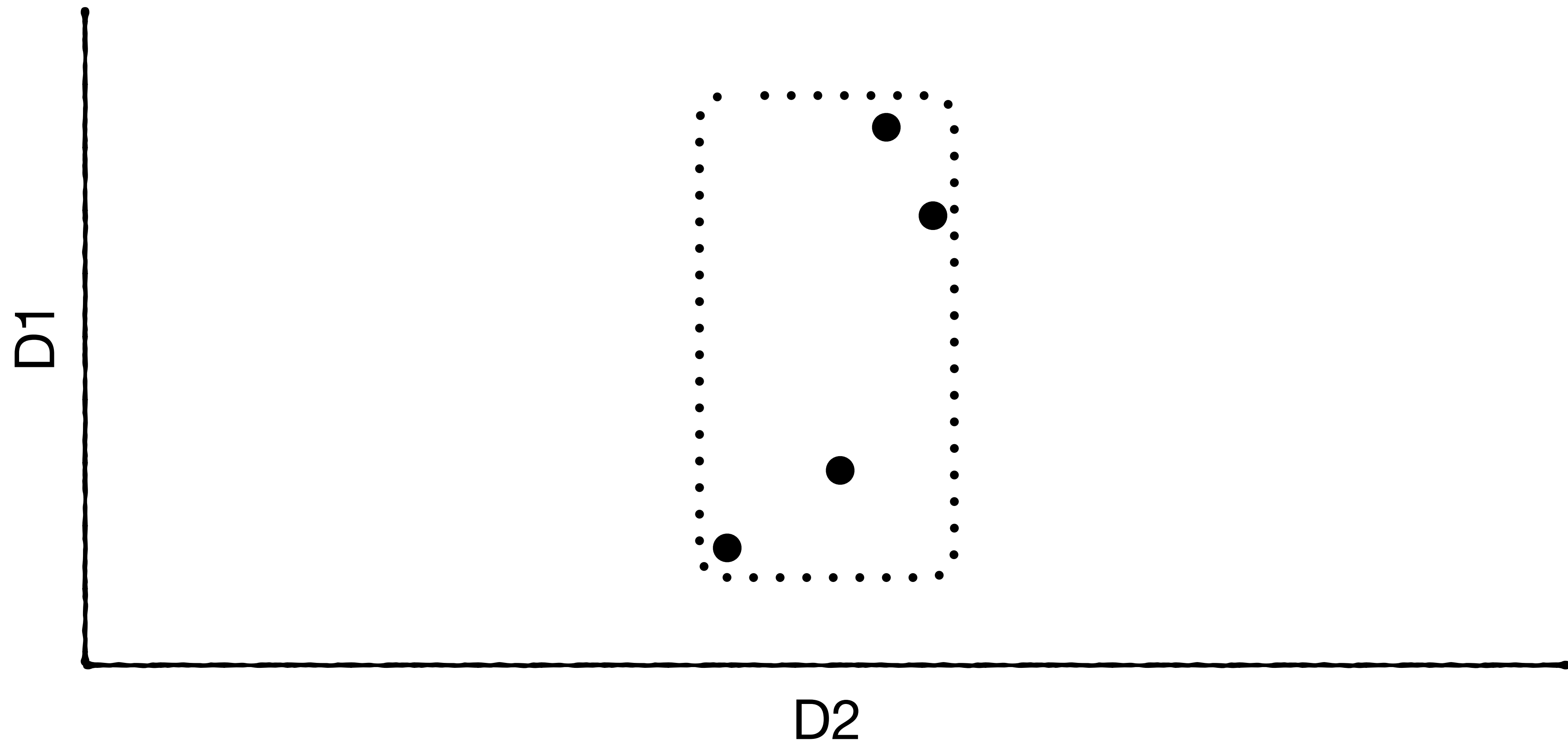
Suppose we insert entry into interdiction
To which node should we add it?
smaller intersecting rectangle



smaller intersecting rectangle
keeps points densely packed

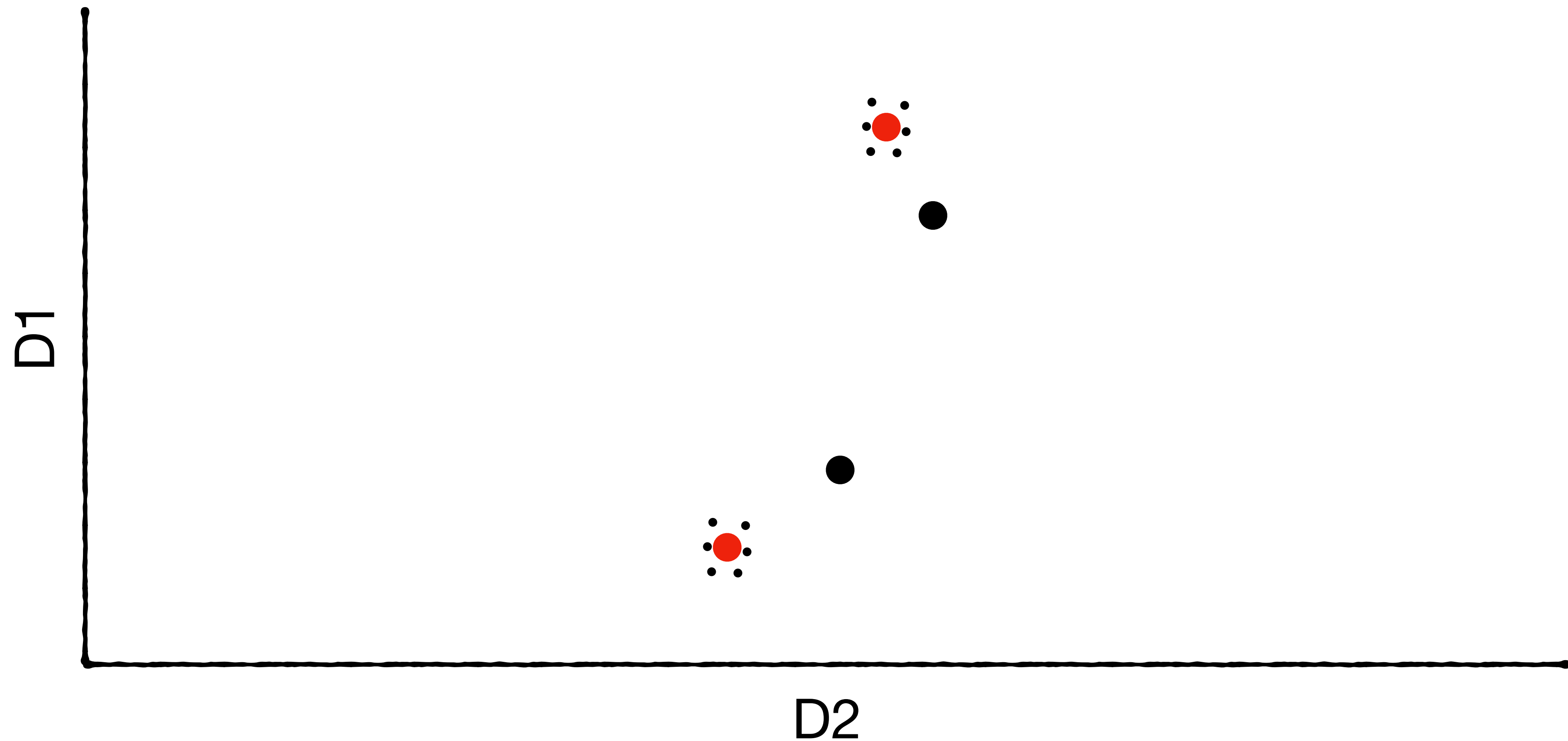


How to split a node?



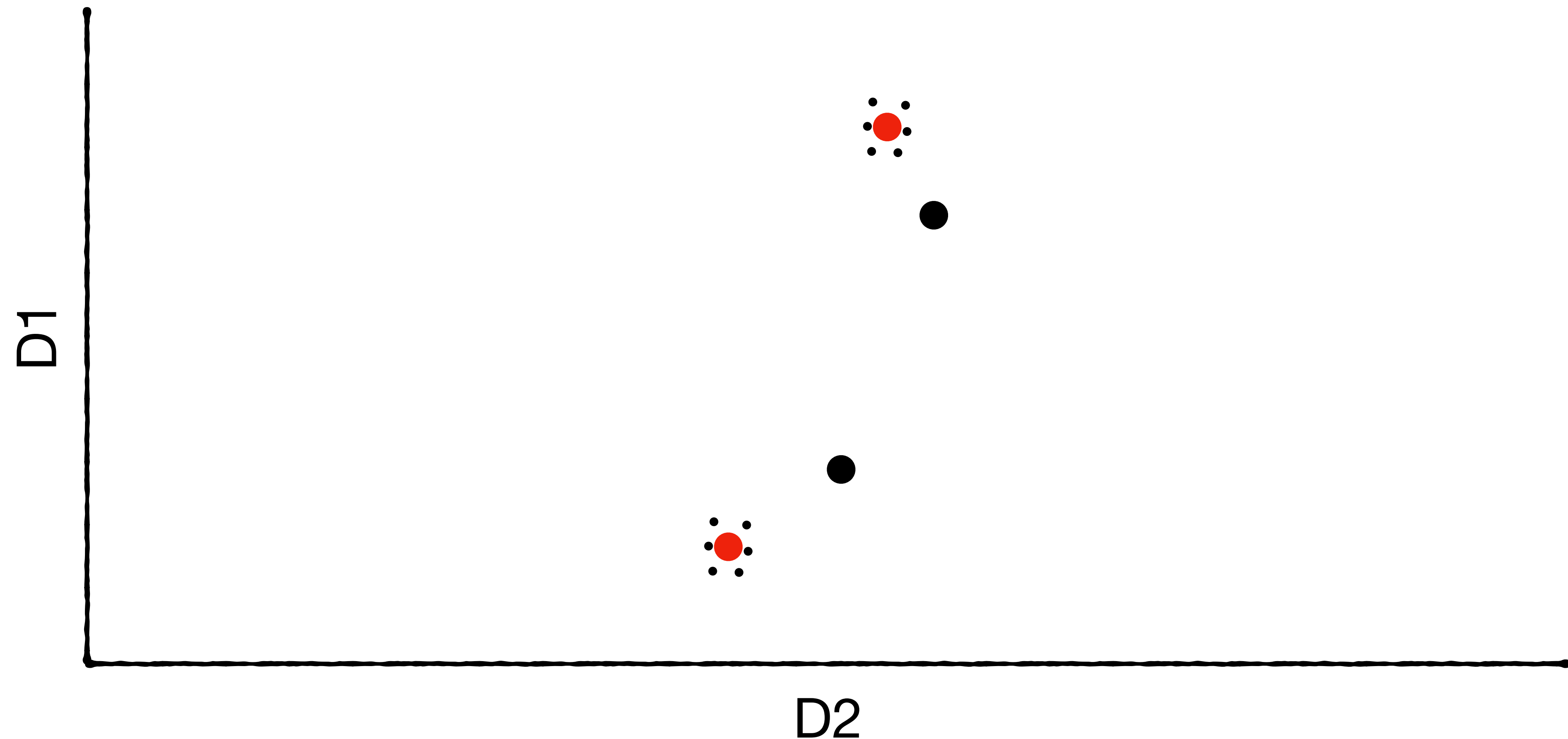
How to split a node?

Pick farthest apart pair of entries as new rectangles



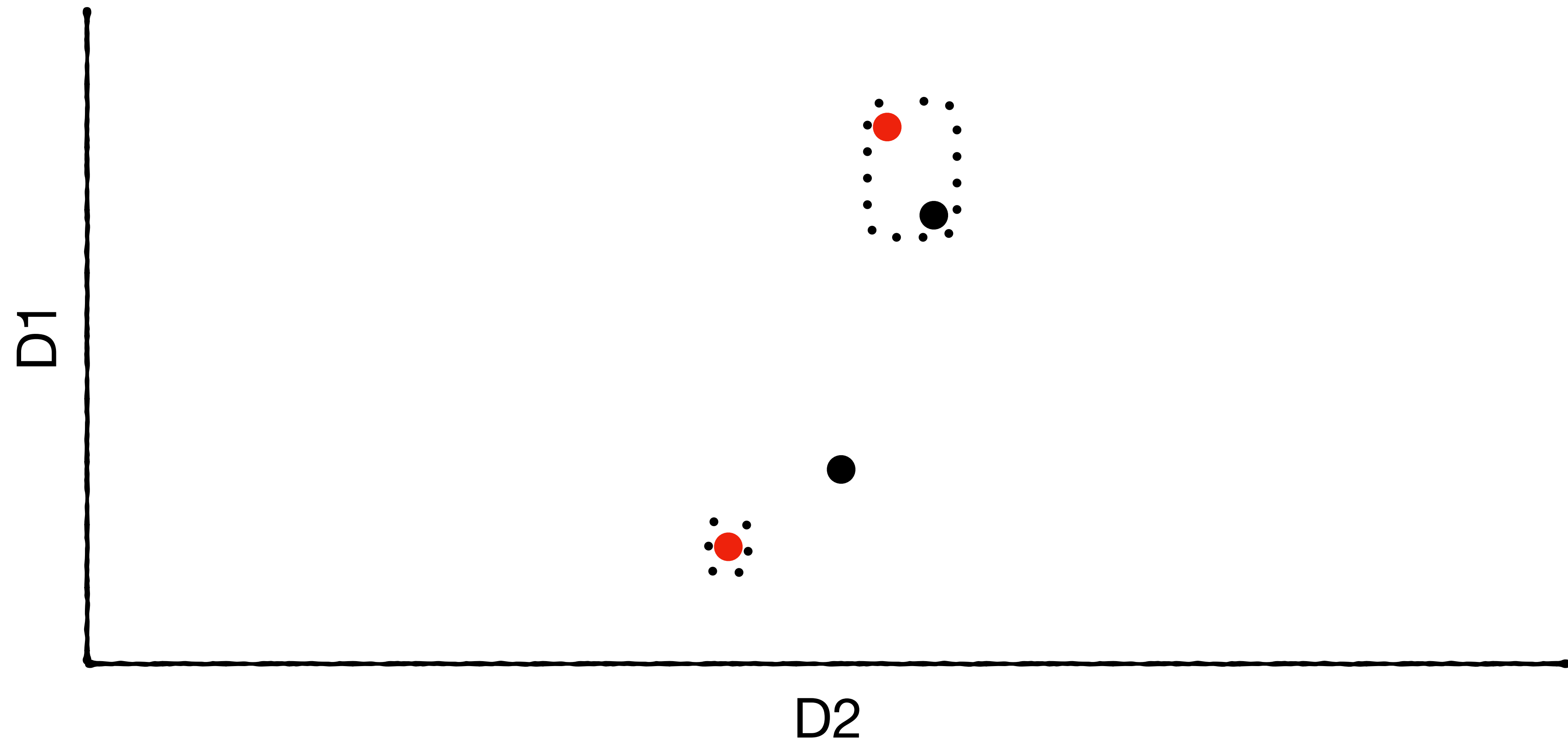
Pick farthest apart pair of entries as new rectangles

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



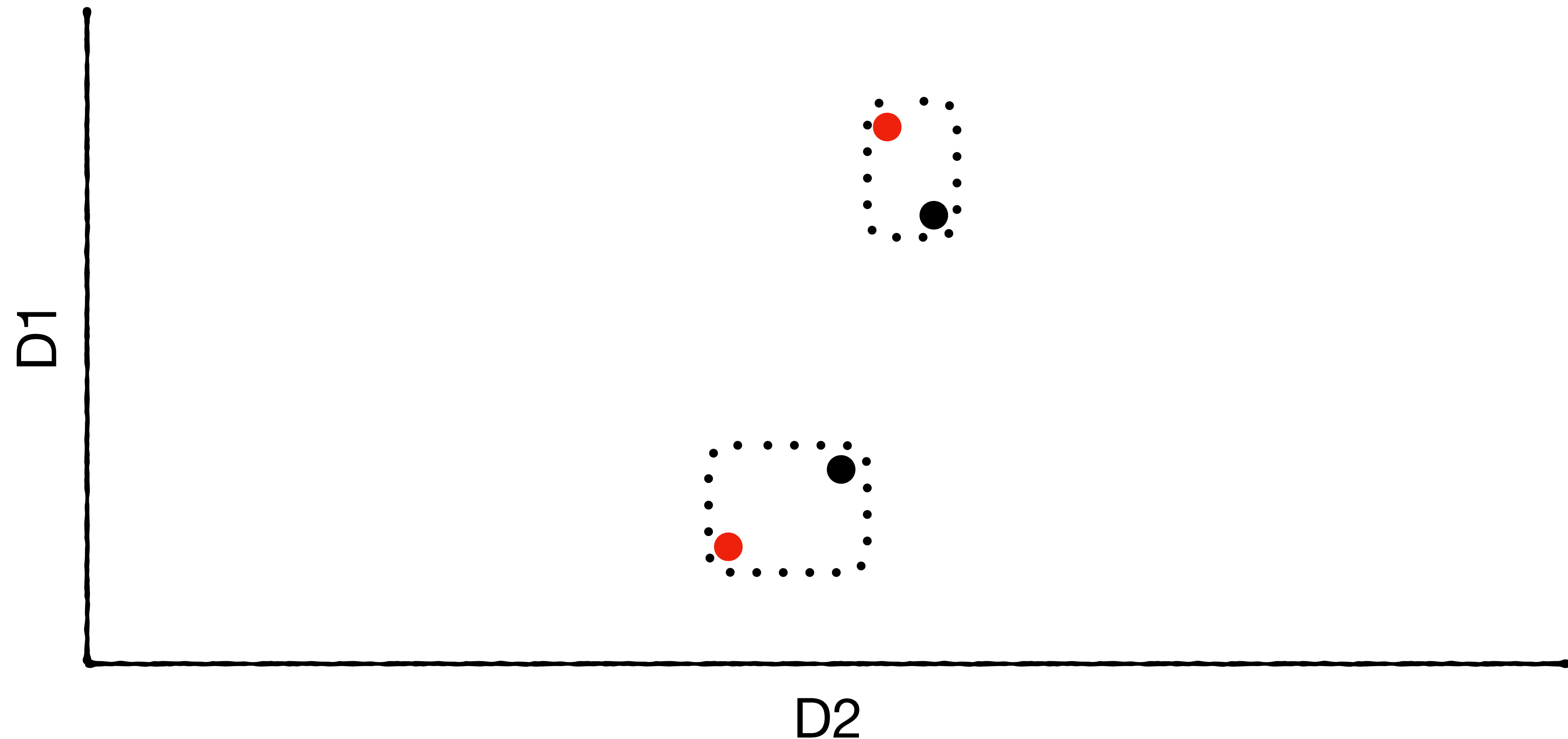
Pick farthest apart pair of entries as new rectangles

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



Pick farthest apart pair of entries as new rectangles

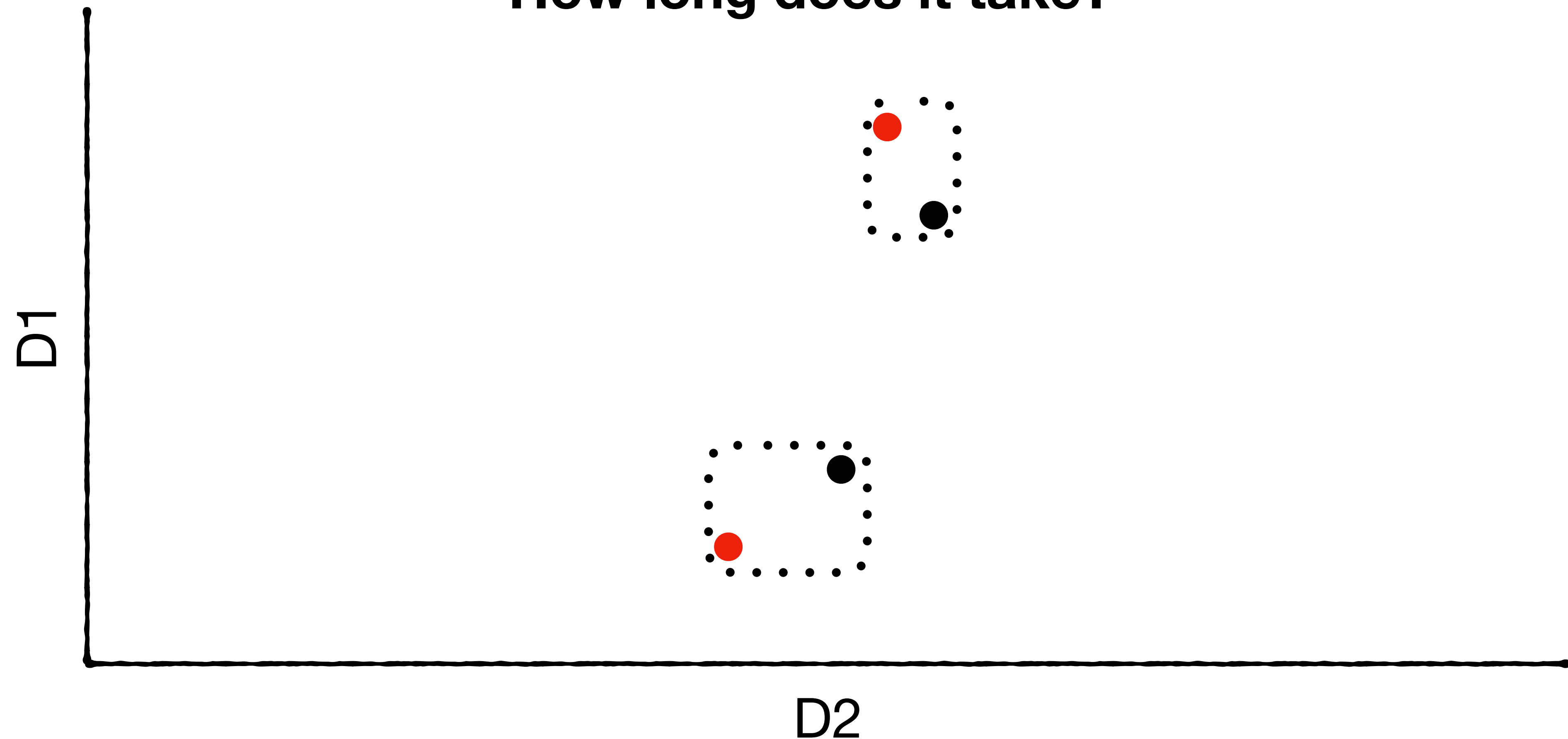
Identify next point that would result in minimum expansion for one of the rectangles



Pick farthest apart pair of entries as new rectangles

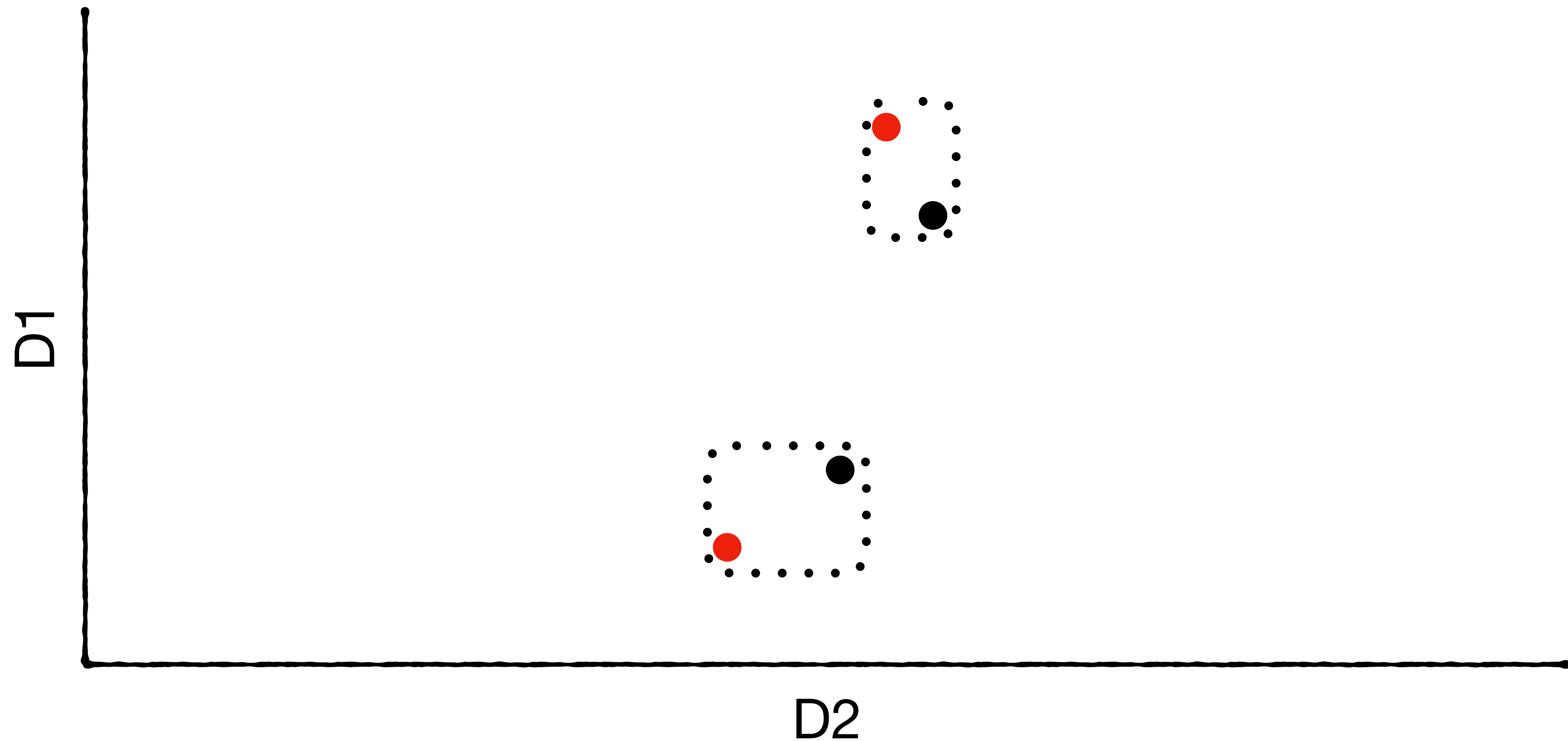
Identify next point that would result in minimum
expansion for one of the rectangles

How long does it take?



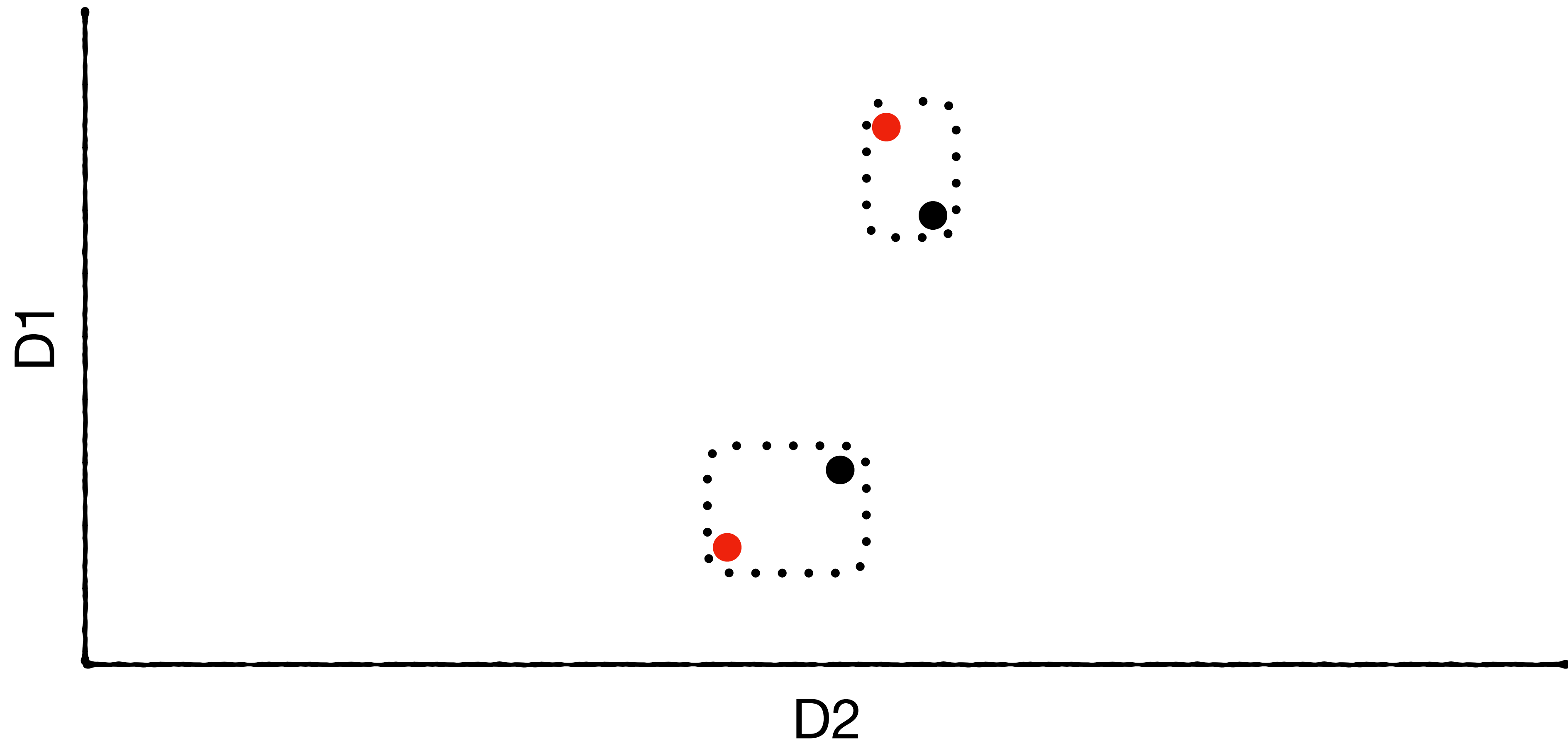
How long does it take?

$O(B^2)$ CPU cycles

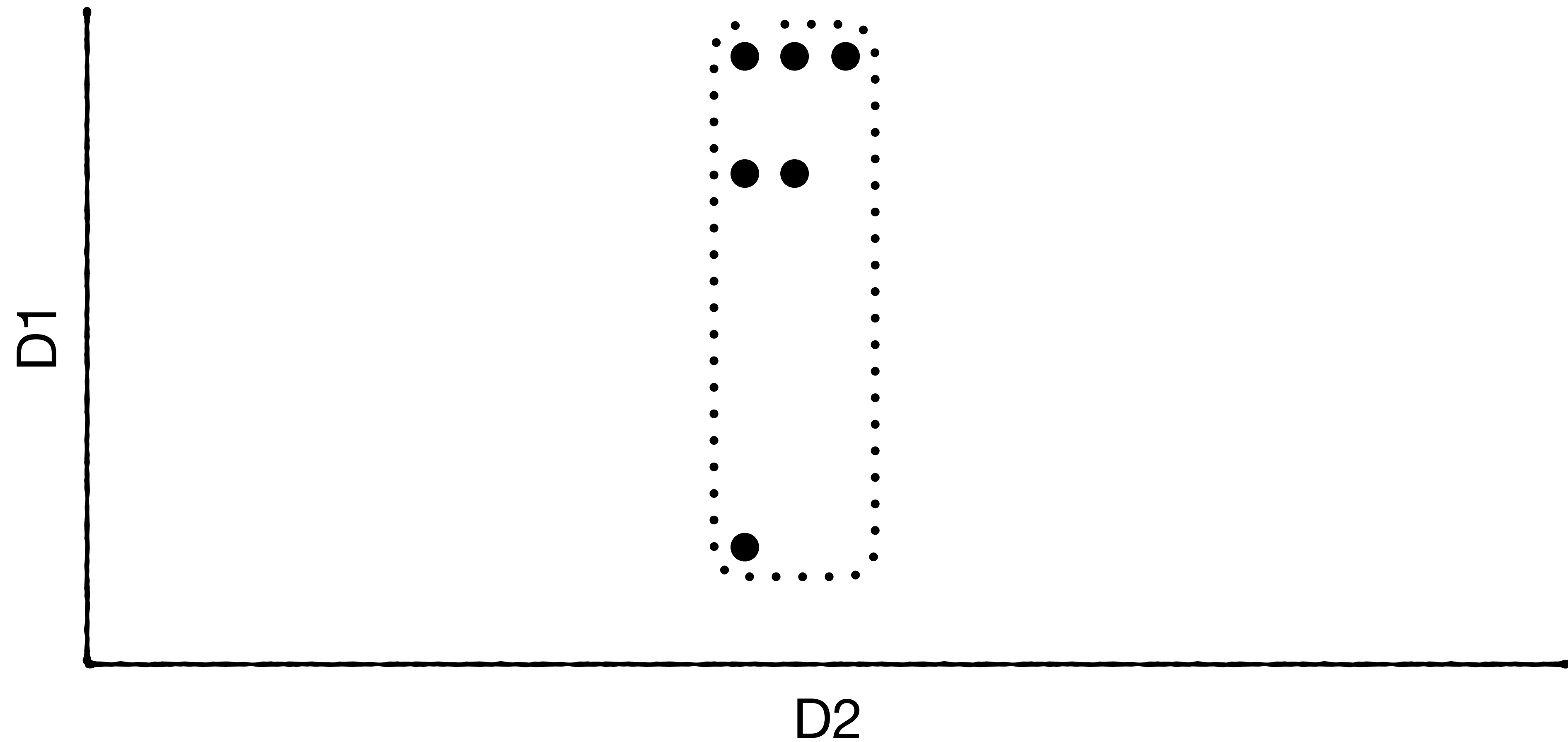


How long does it take?

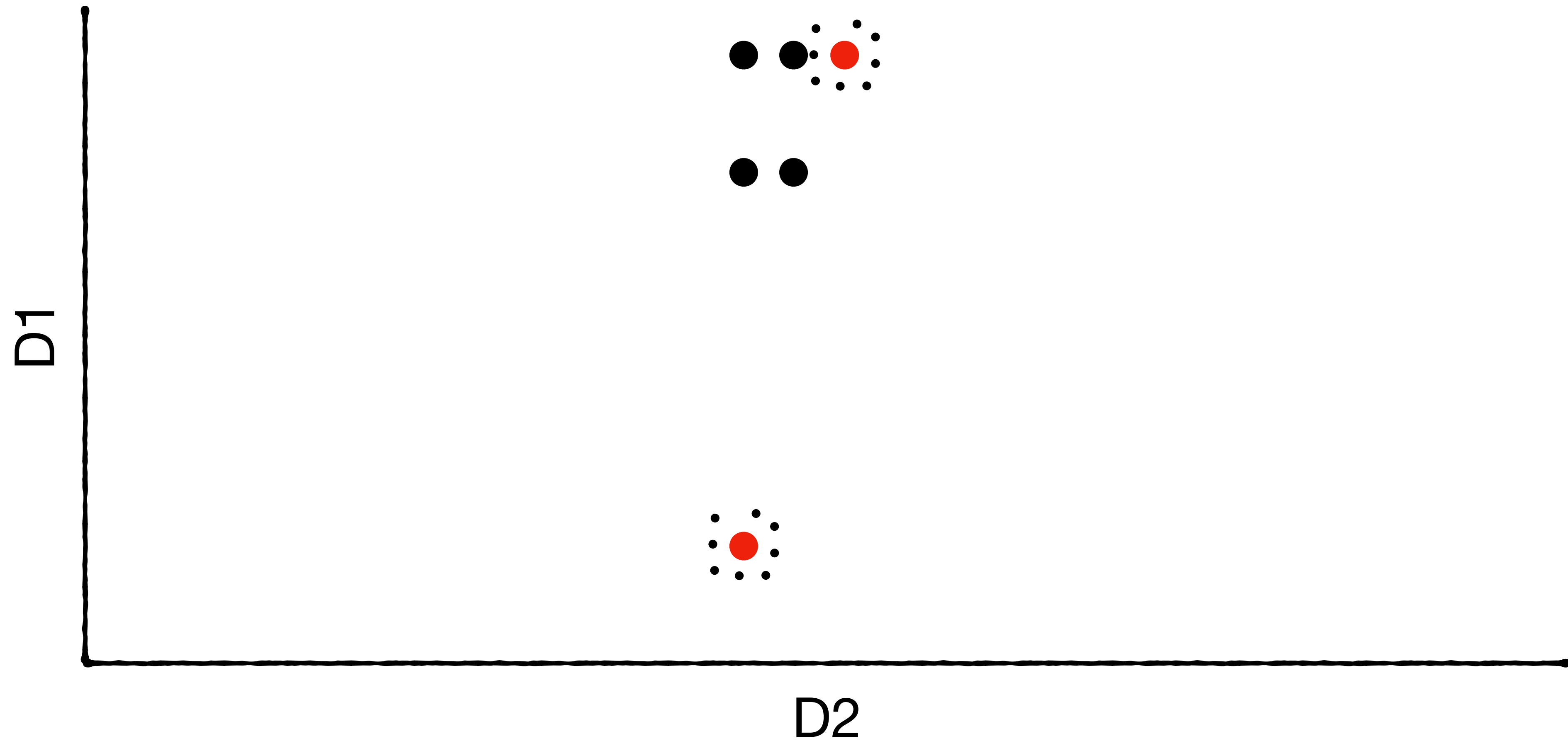
$O(B^2)$ CPU cycles - **referred to as Quadratic Split**



Consider a more nuanced split example

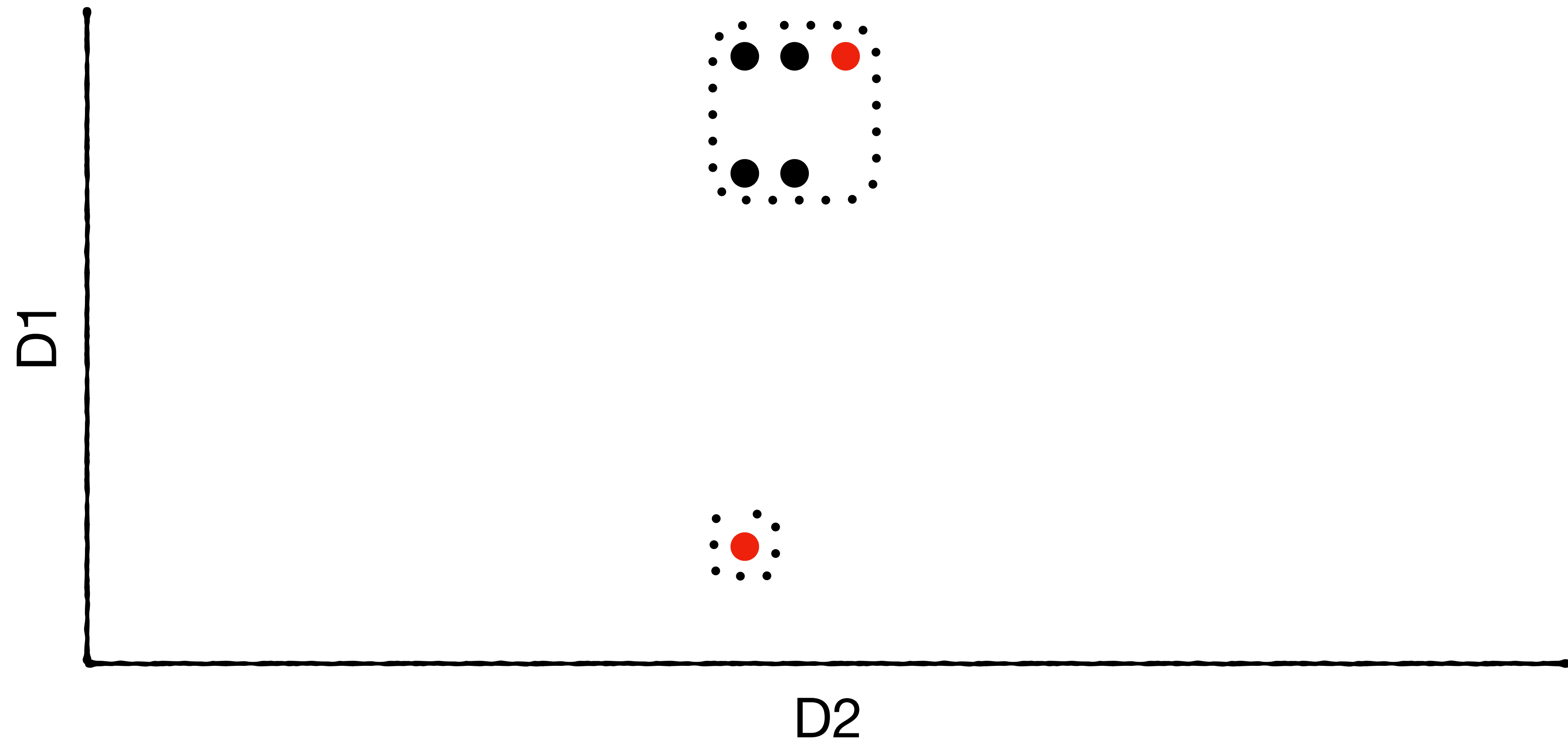


Find farthest apart points



Find farthest apart points

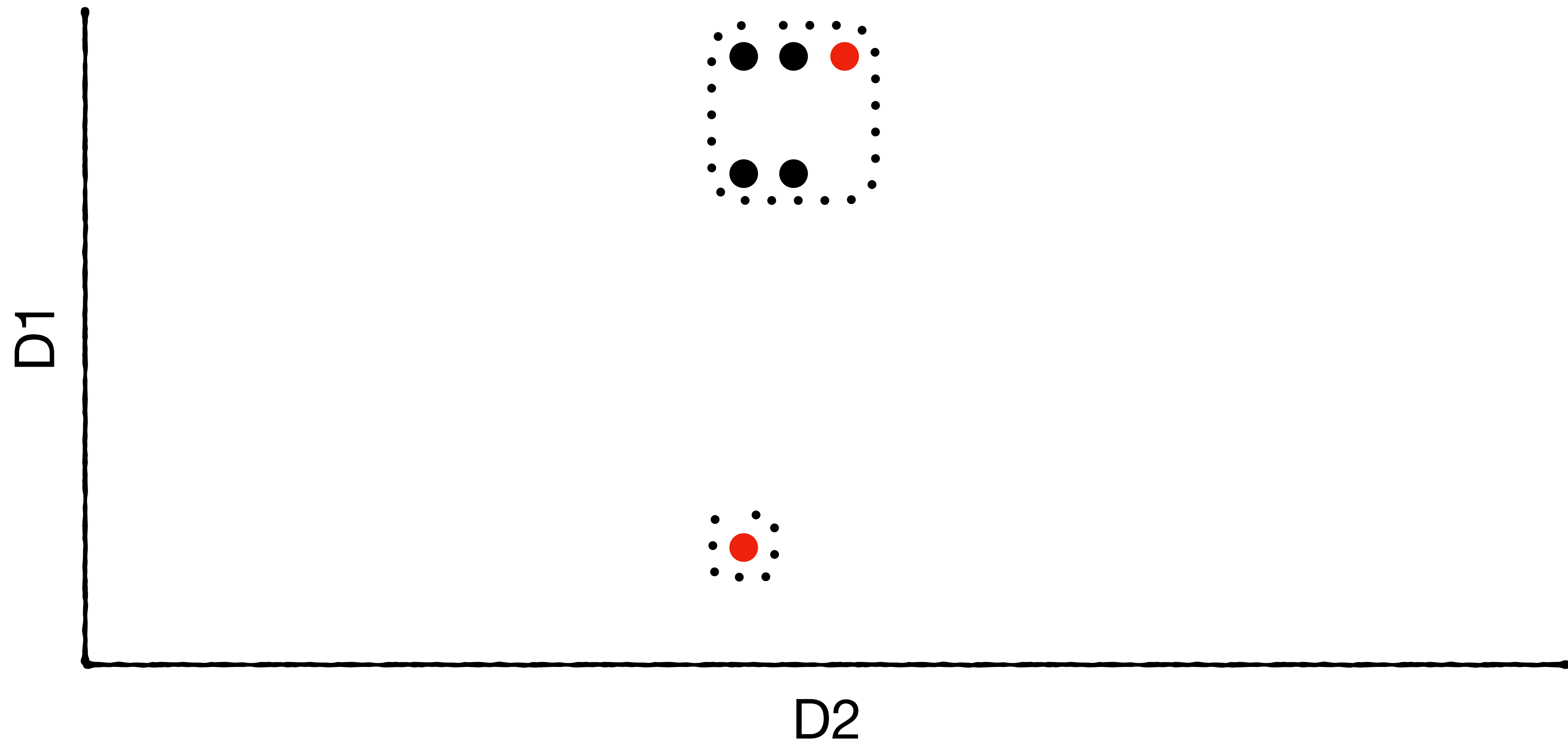
If we apply simple quadratic split, all points would go in one node



Find farthest apart points

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

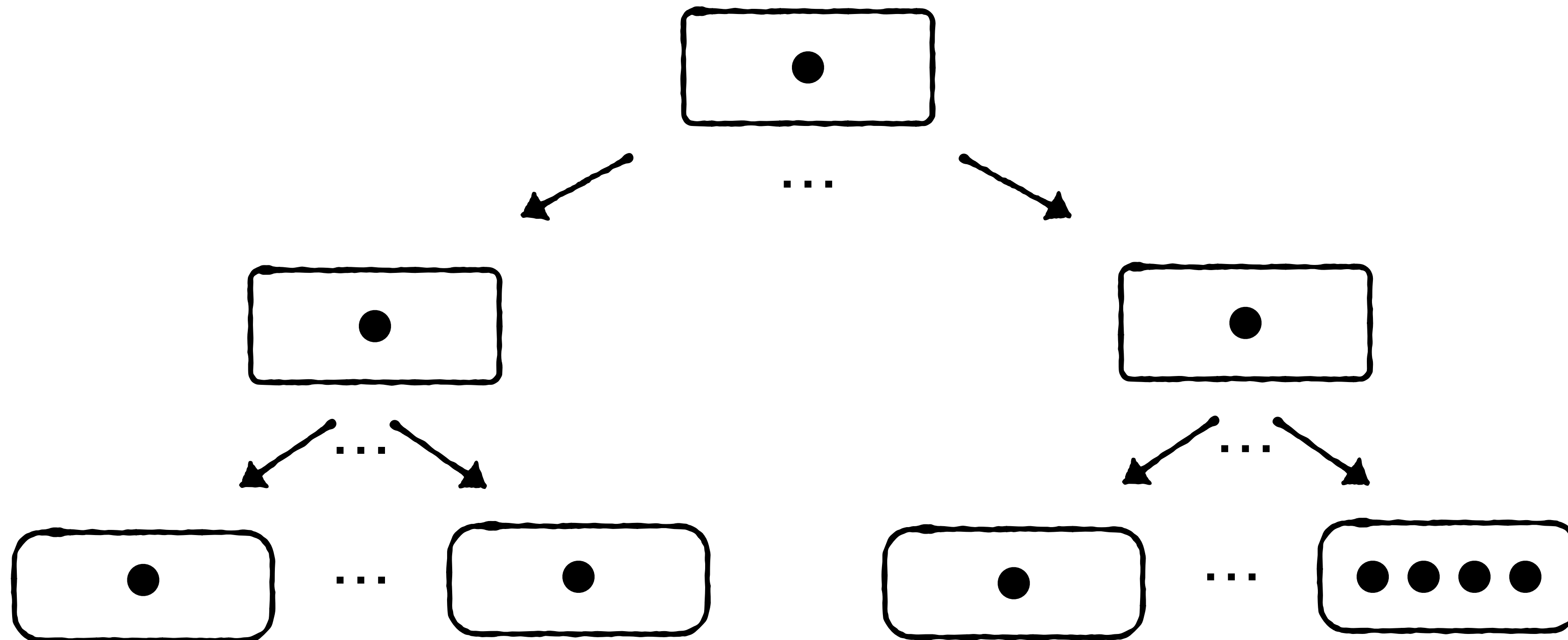
Why is this bad?



Find farthest apart points

Should we insert remaining points to closest node?

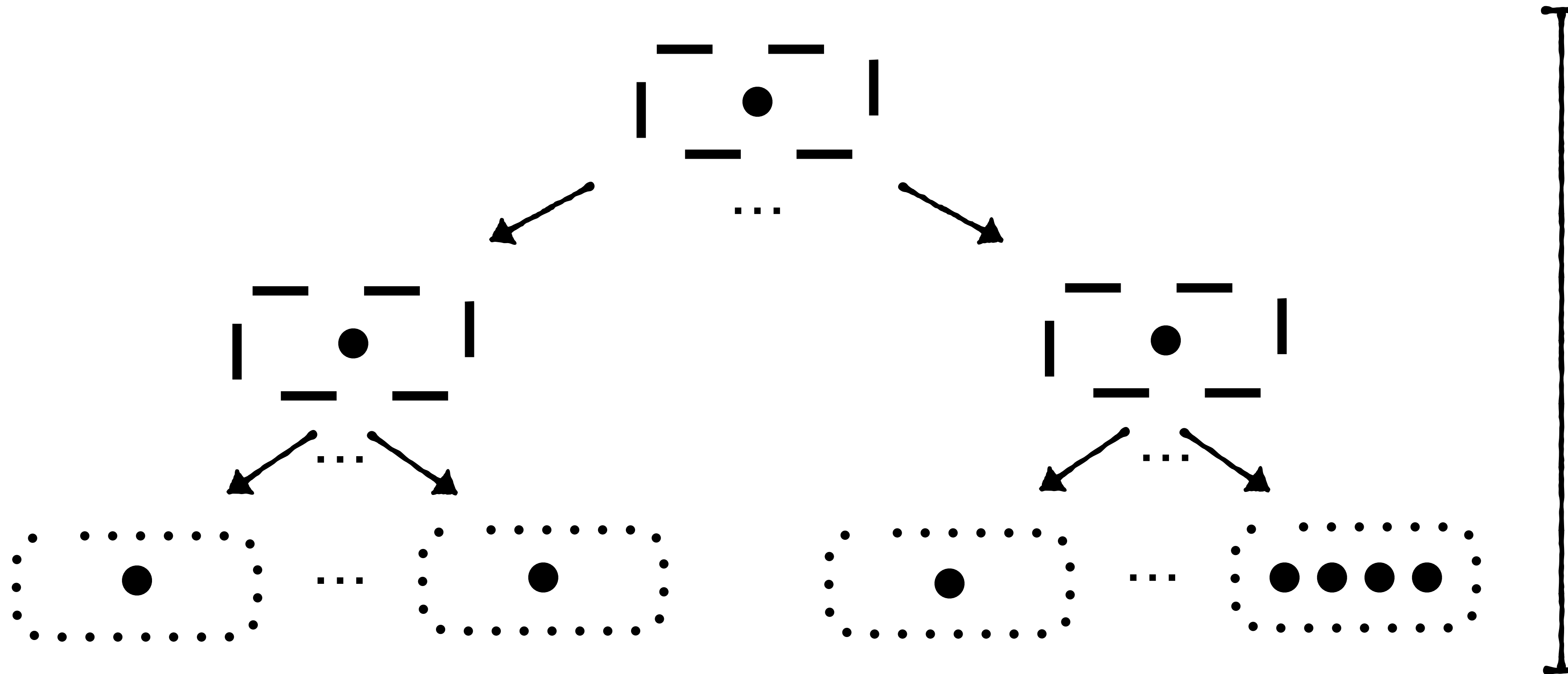
No as this would lead to an unbalanced tree.



Find farthest apart points

Should we insert remaining points to closest node?

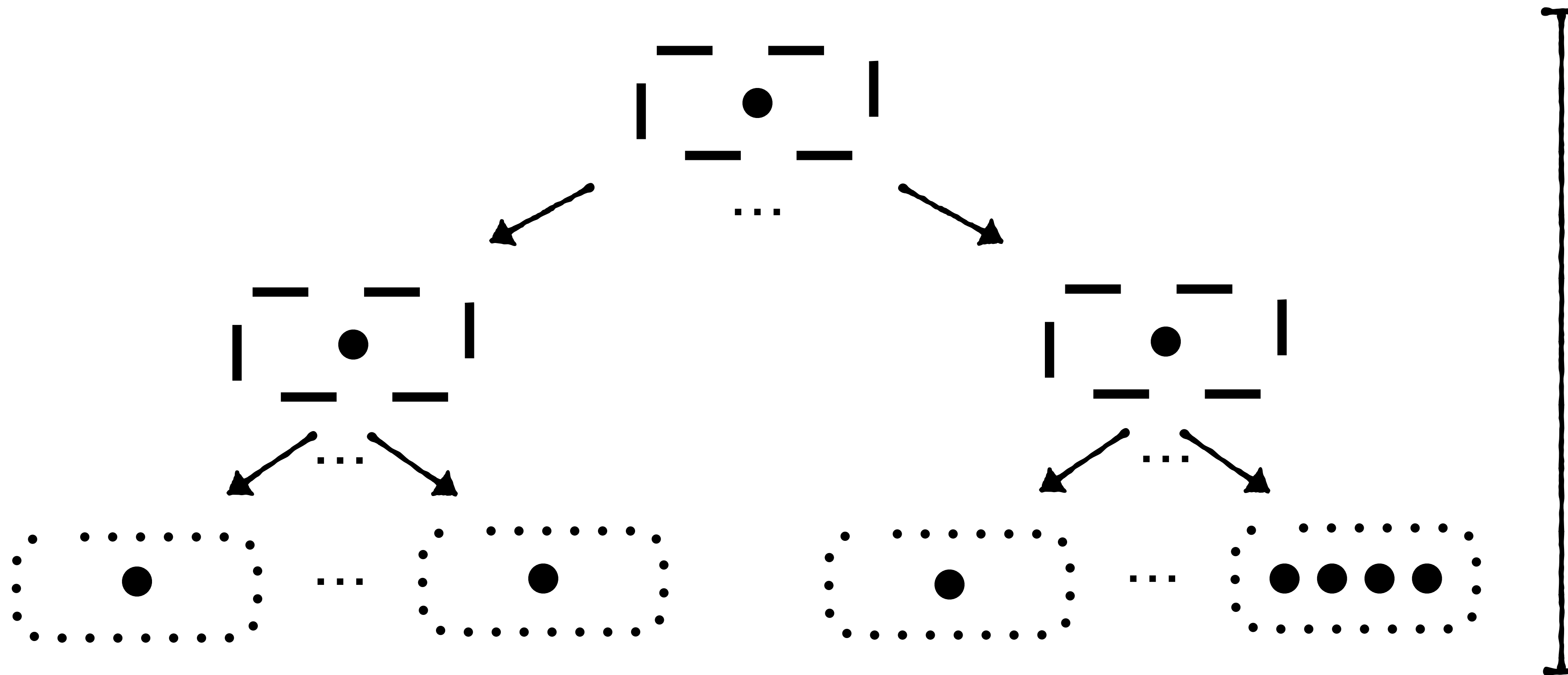
No as this would lead to an unbalanced tree.



Find farthest apart points

Should we insert remaining points to closest node?

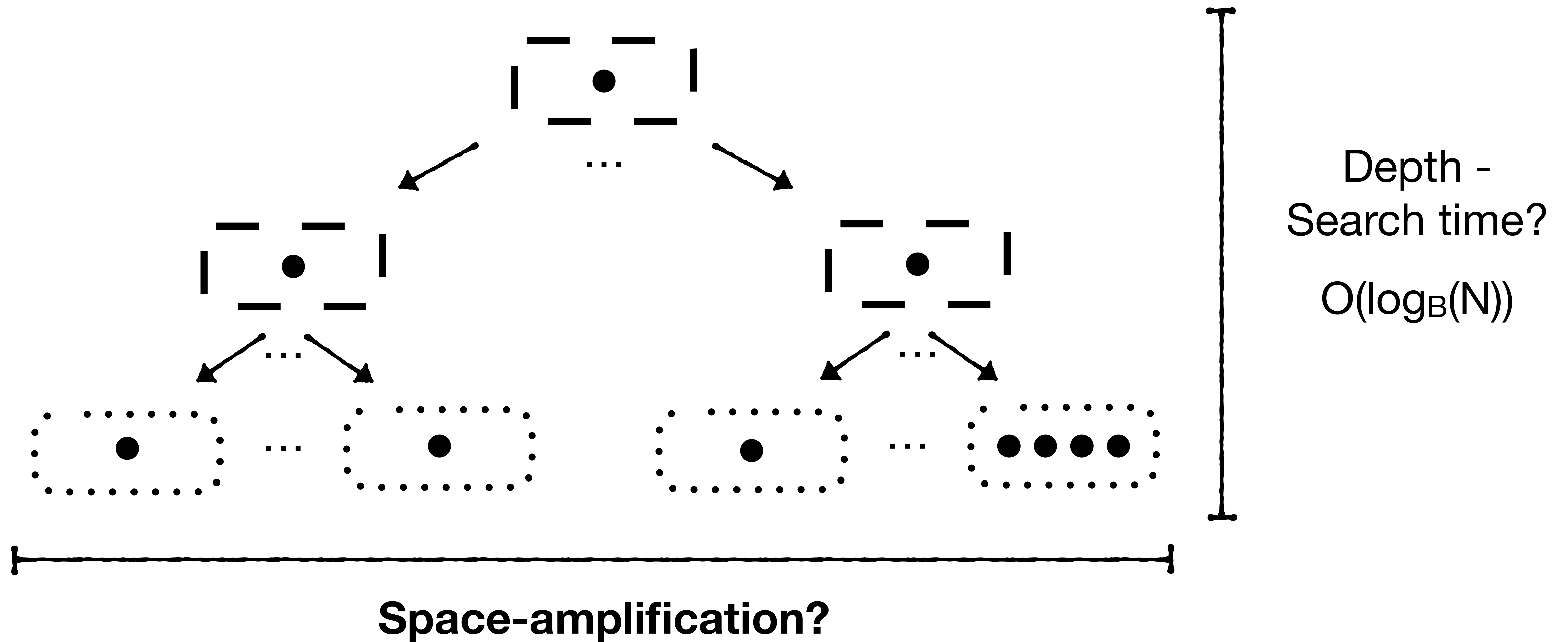
No as this would lead to an unbalanced tree.

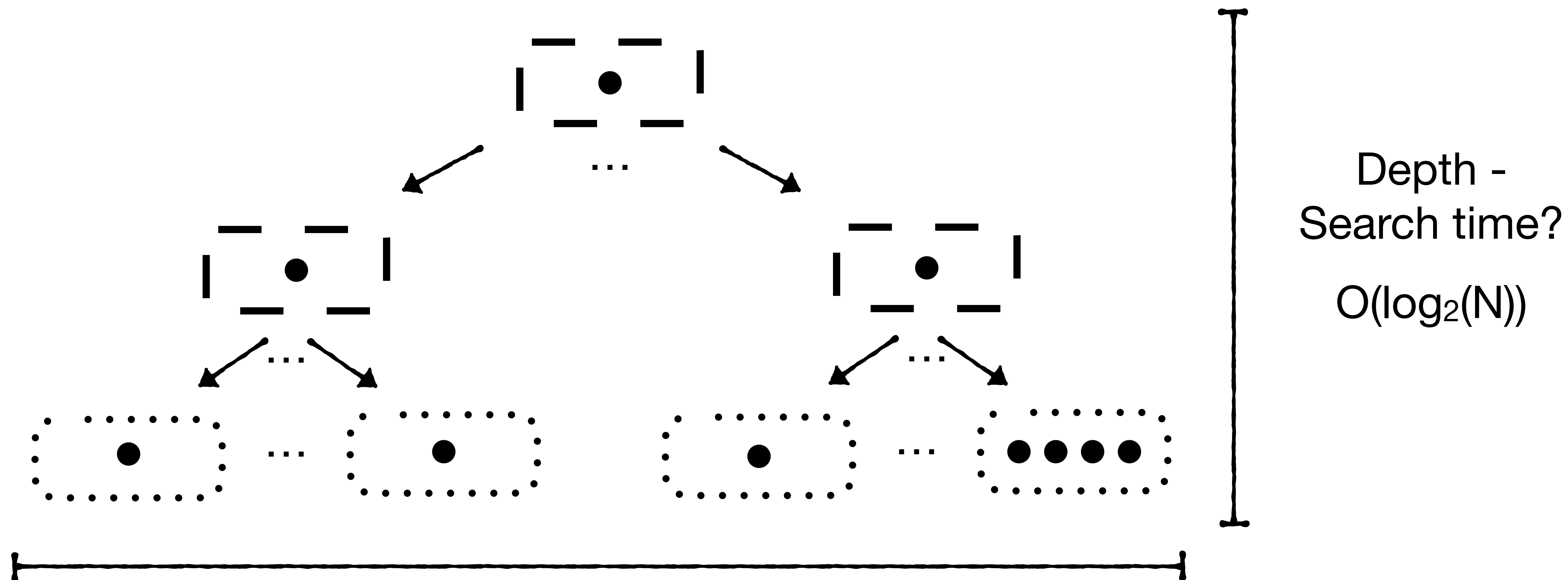


**Depth -
Search time?**

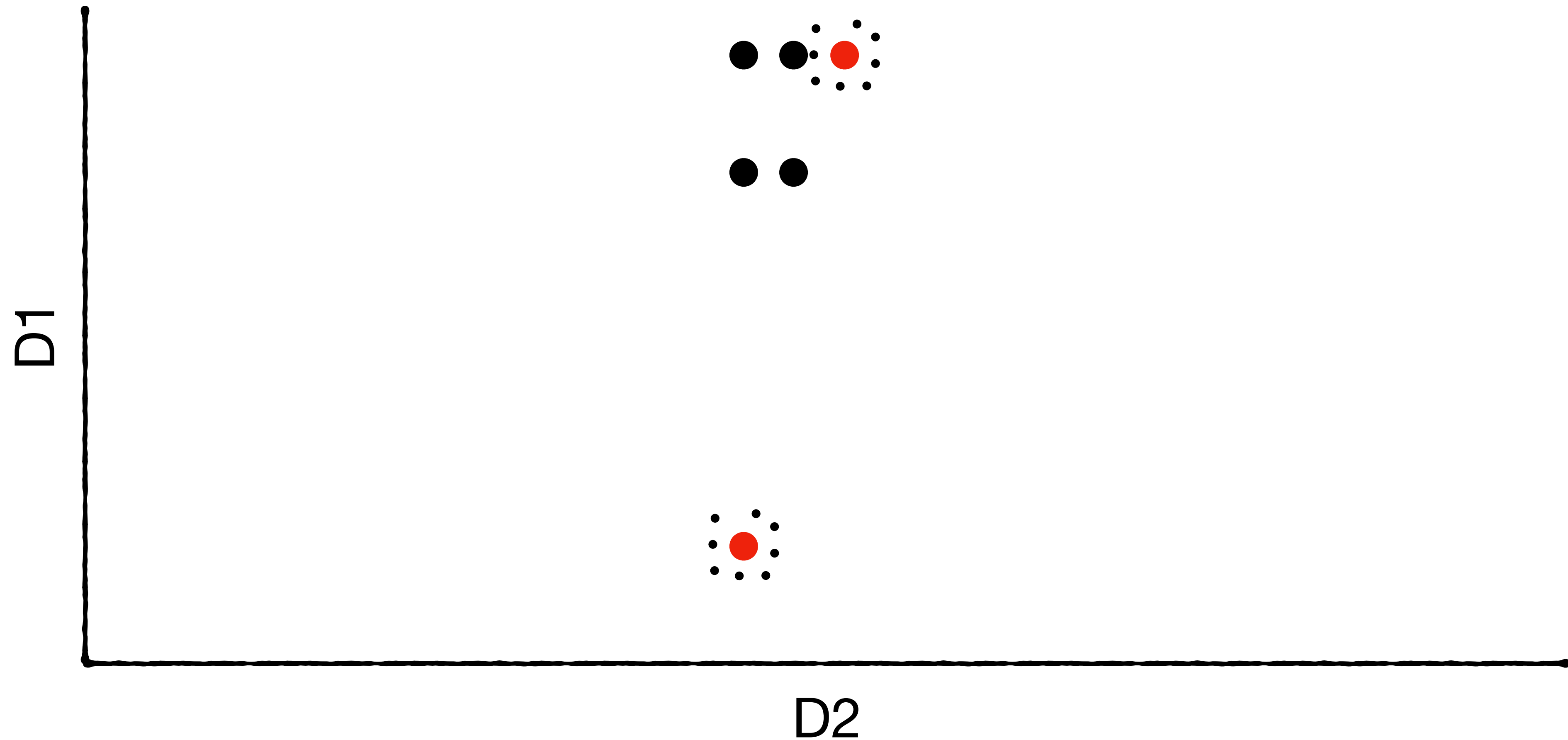
$O(\log_2(N))$

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

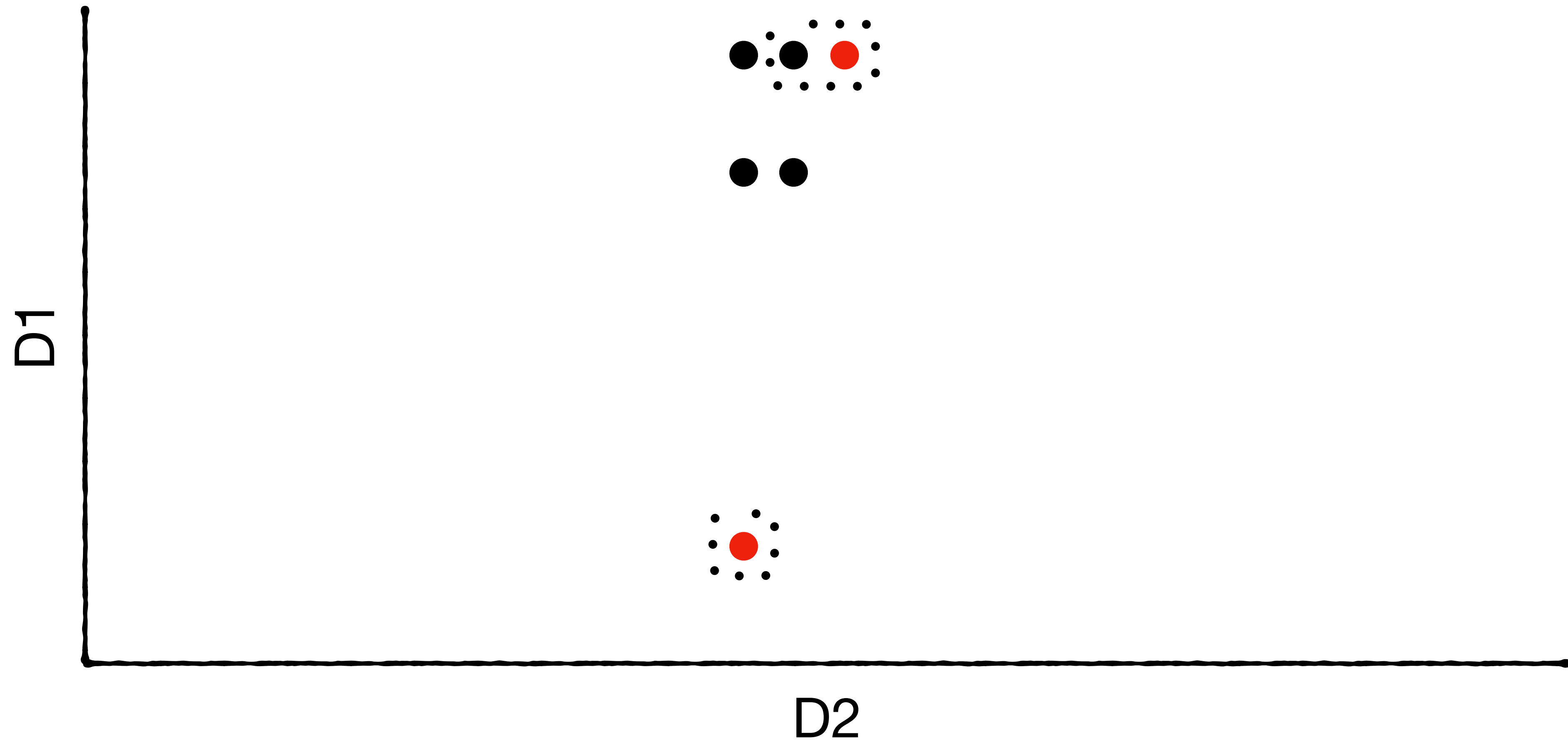




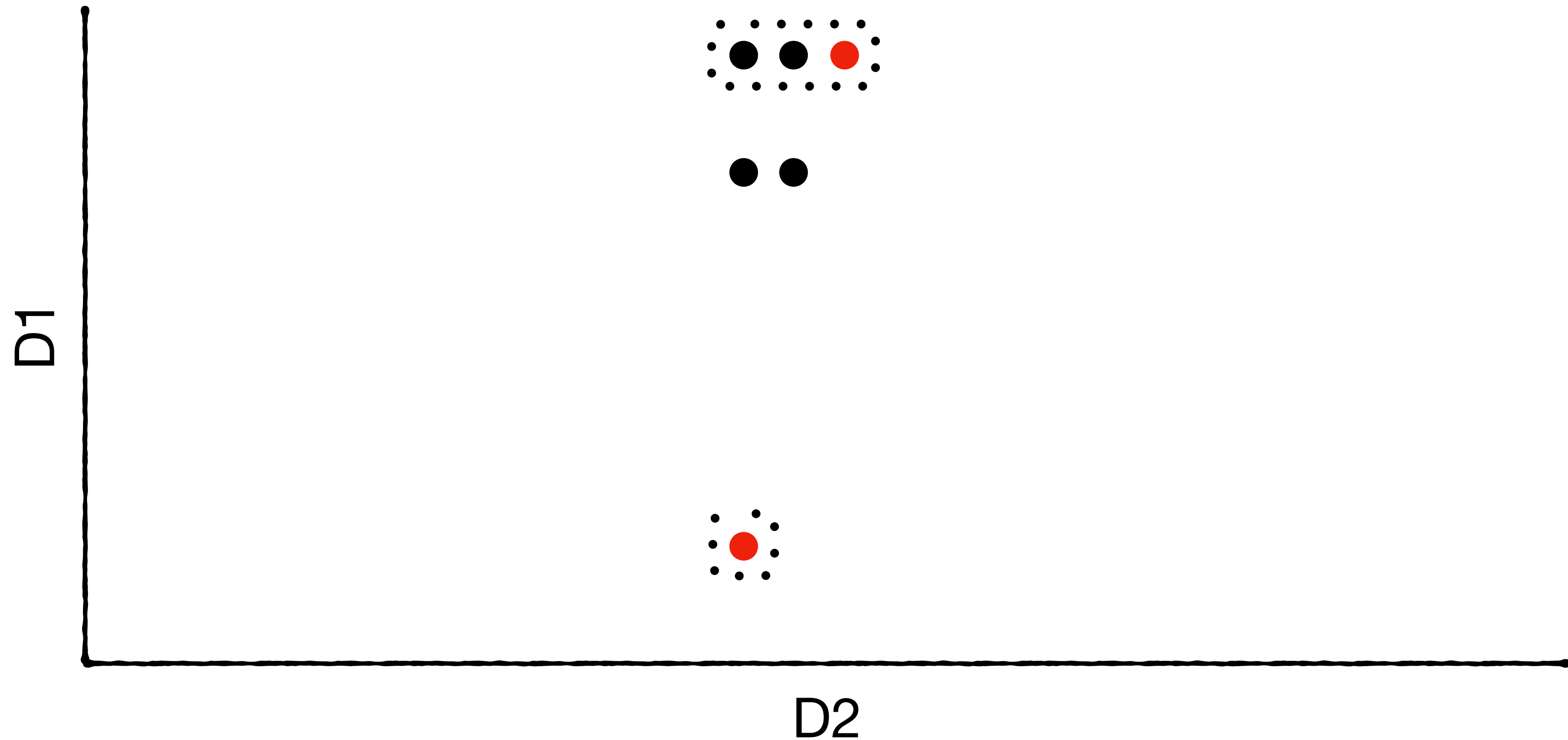
**Instead, perform quadratic split subject to each
node ultimately being half full**



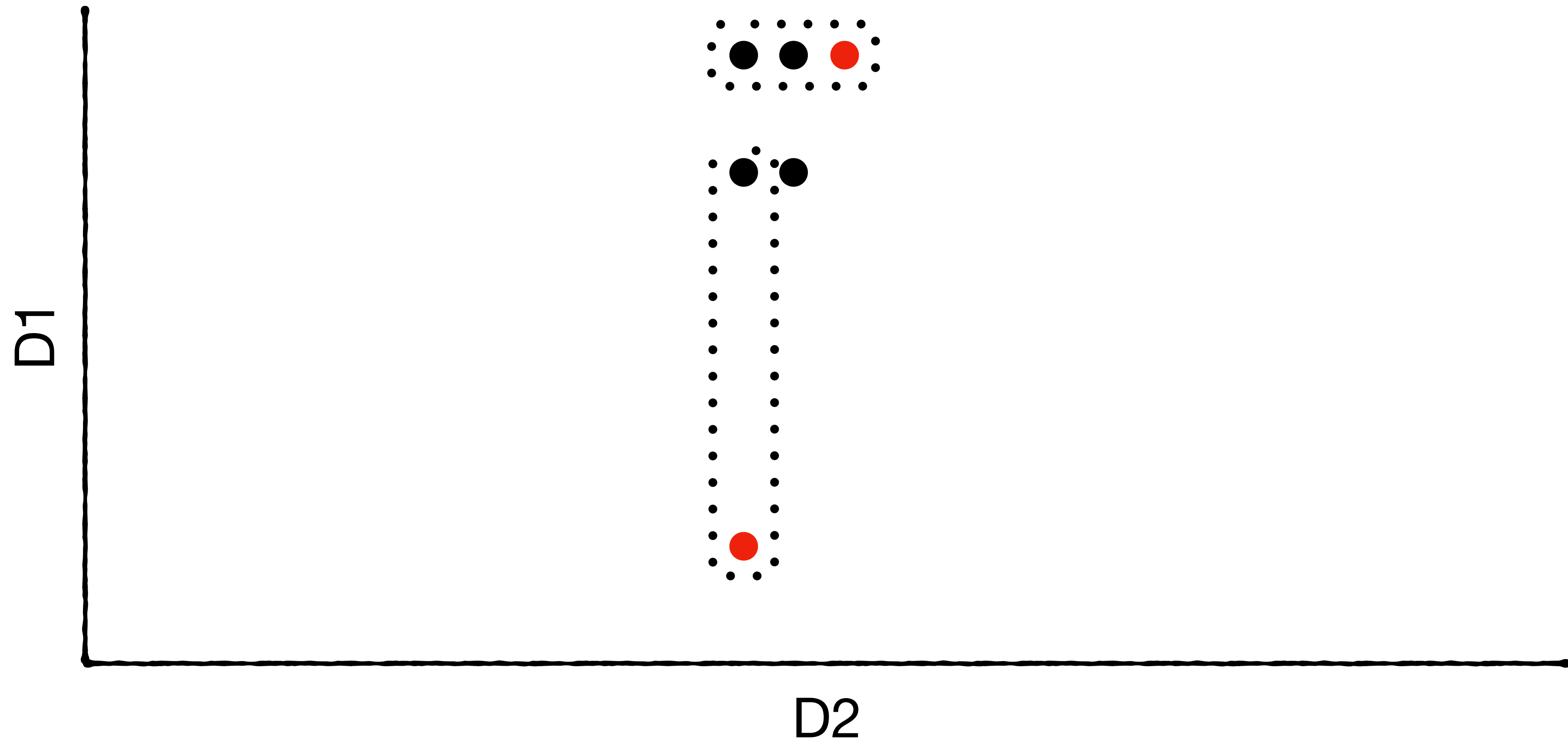
**Instead, perform quadratic split subject to each
node ultimately being half full**



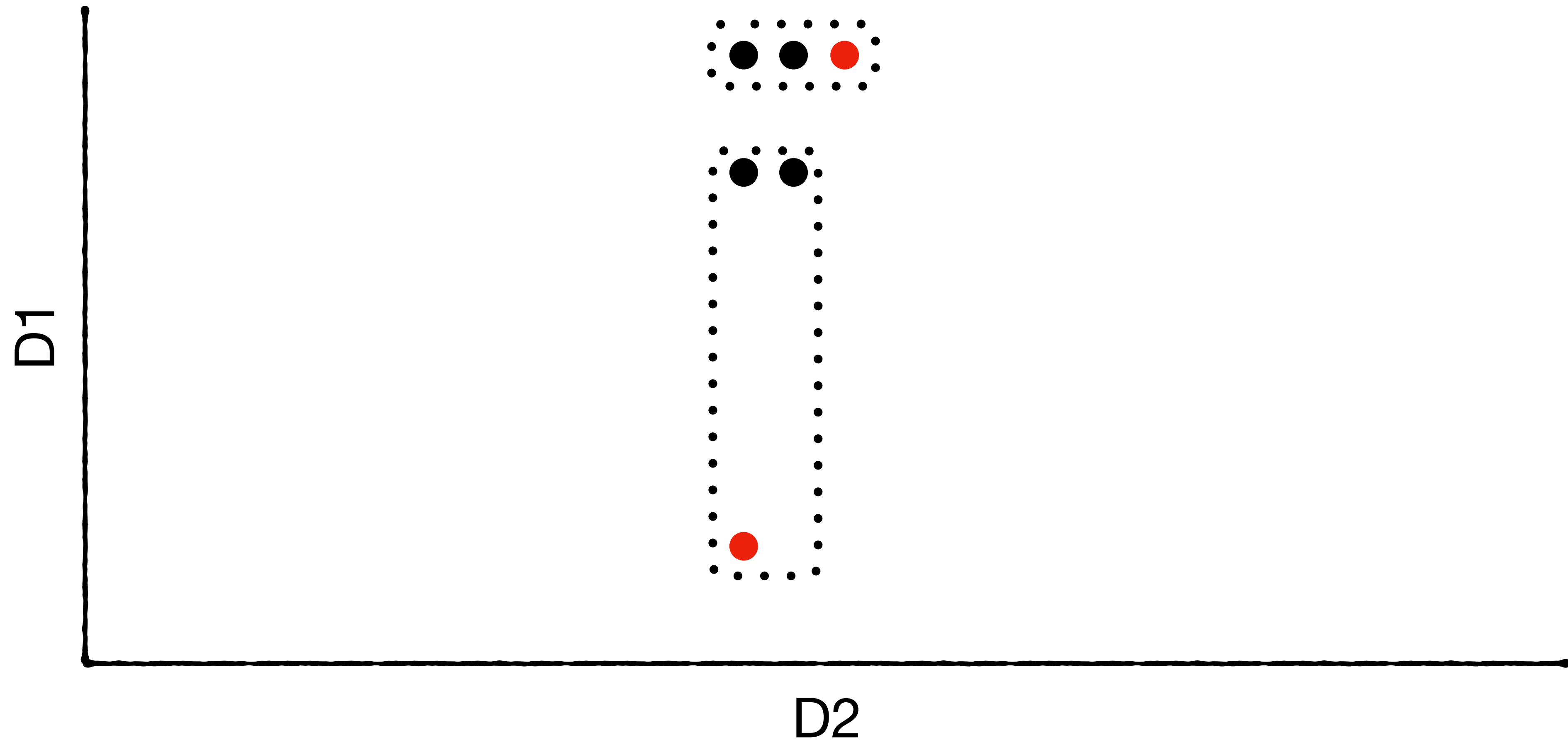
**Instead, perform quadratic split subject to each
node ultimately being half full**



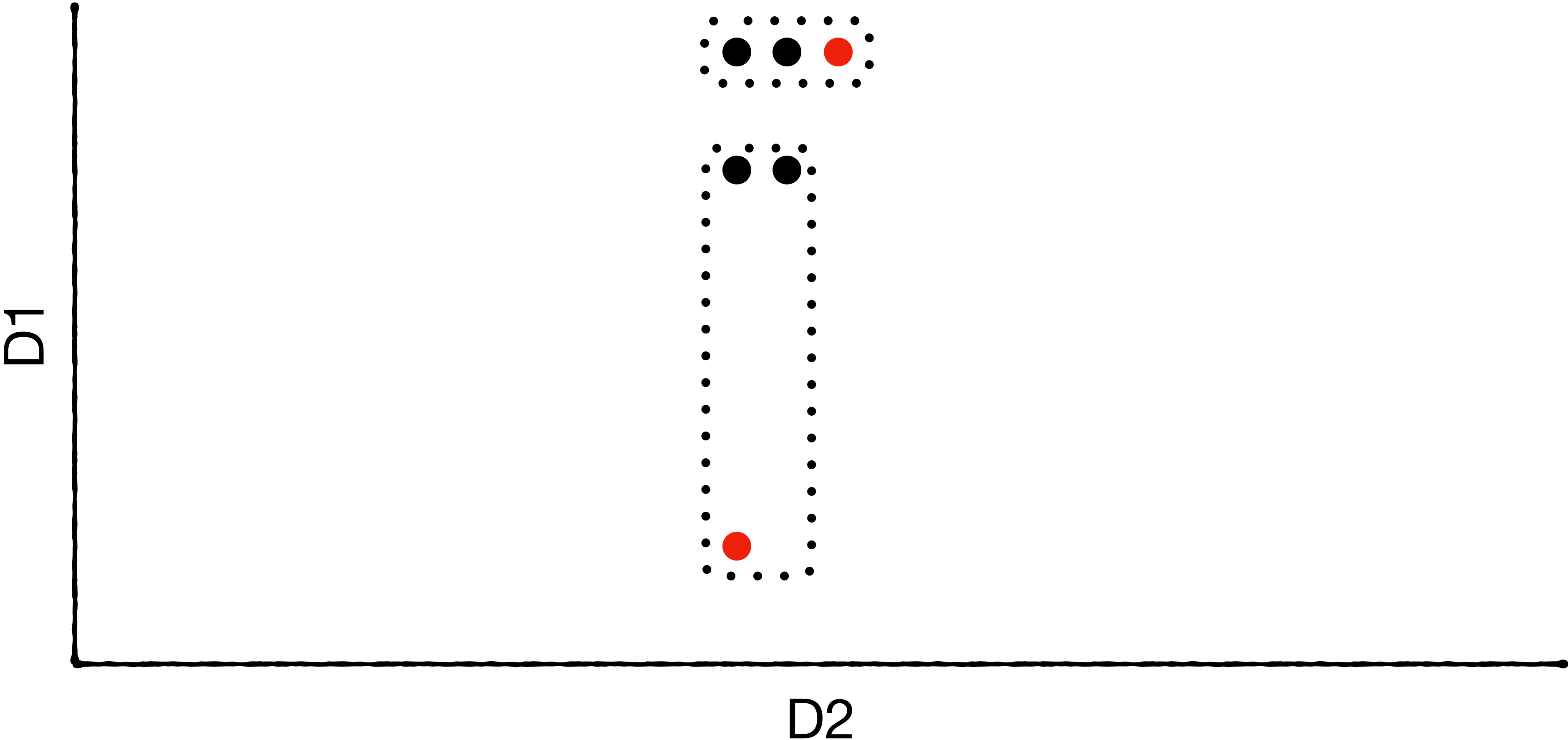
**Instead, perform quadratic split subject to each
node ultimately being half full**



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



- Goals:**
- (1) minimize coverage**
 - (2) minimize intersections**
 - (3) maintain balance**

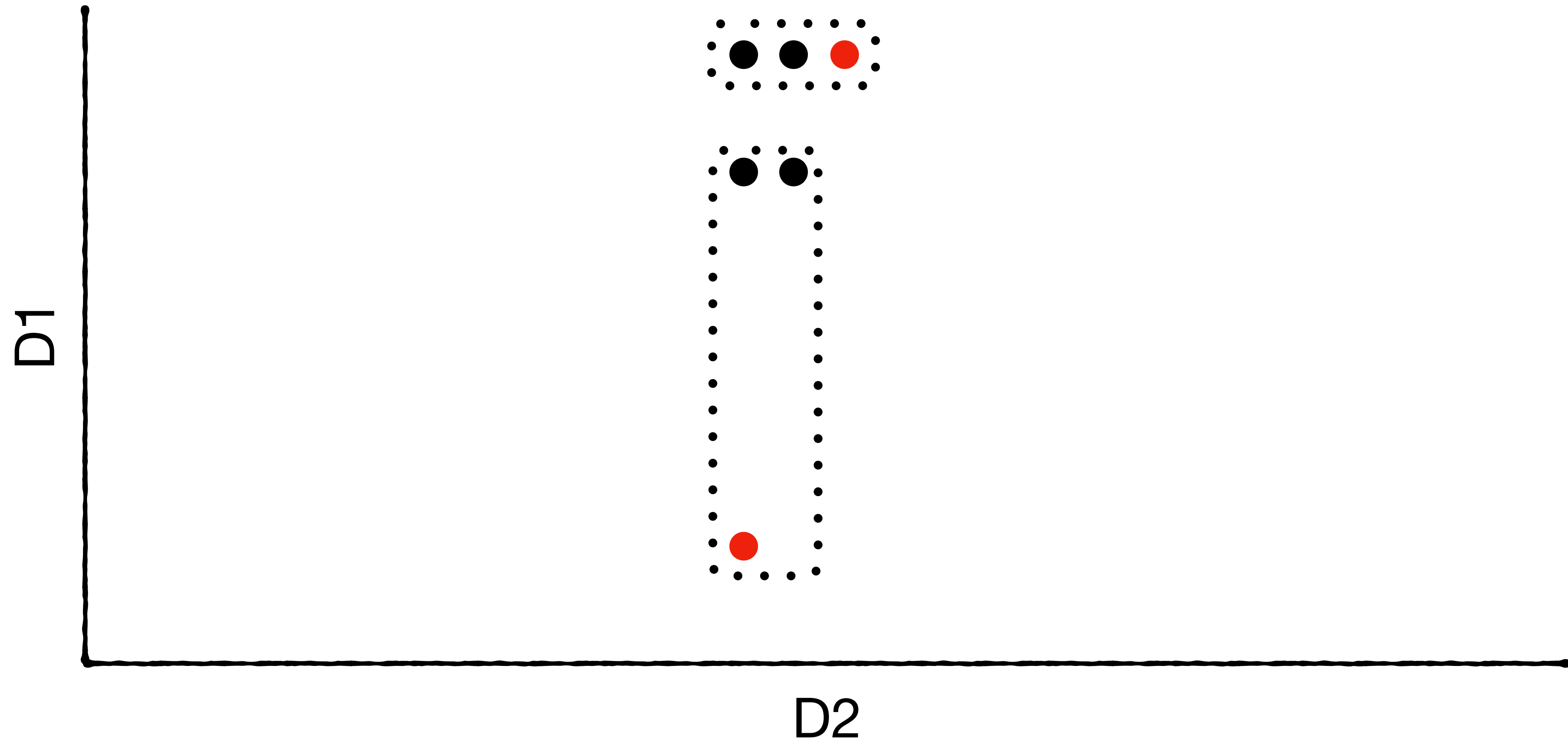


These sometimes conflict :)

Goals: (1) minimize coverage

(2) minimize intersections

(3) maintain balance



KD-Trees
1975

R-Trees
1984

R+Tree
1987

R*-Tree
1990

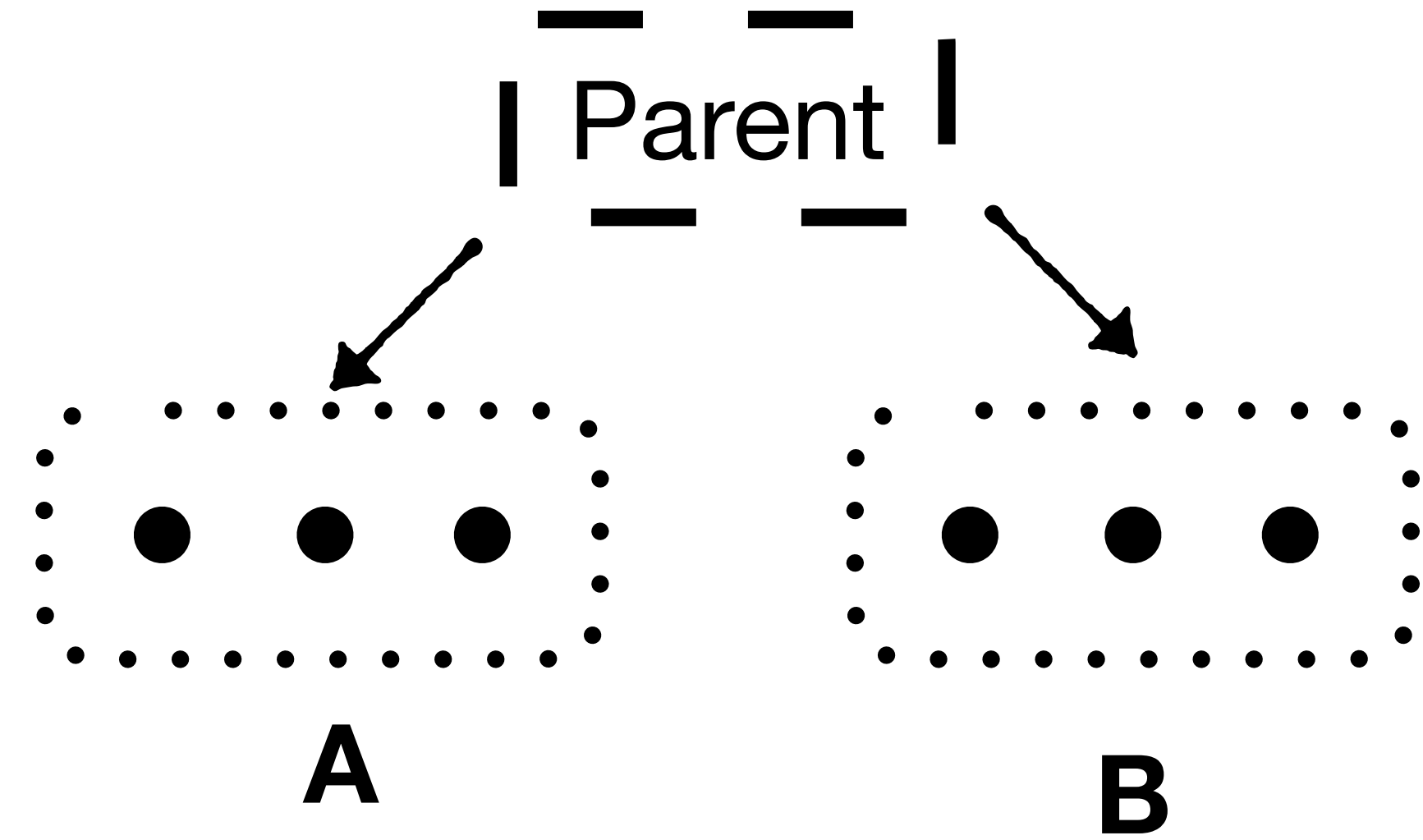
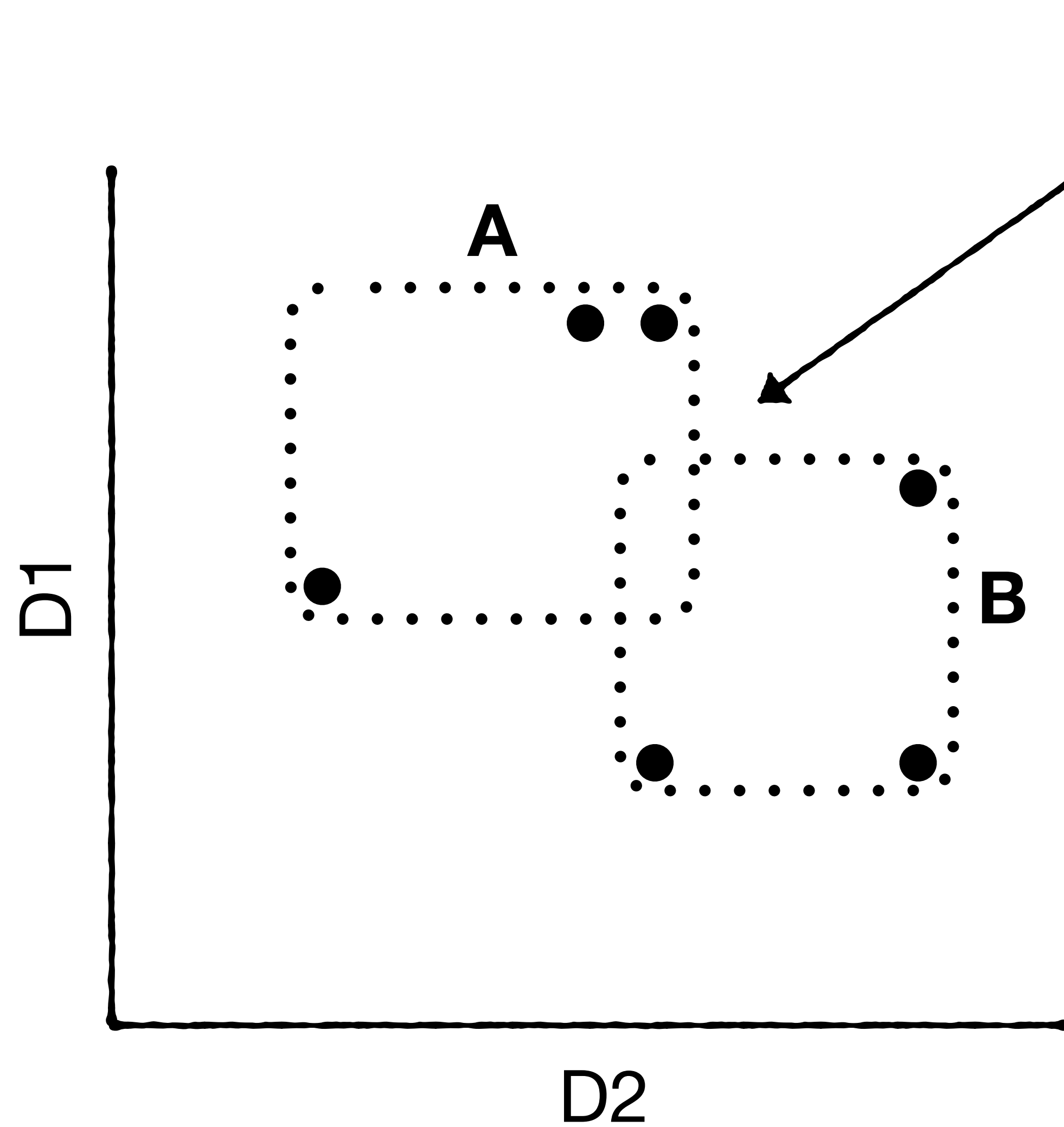
UB-Tree
2000

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

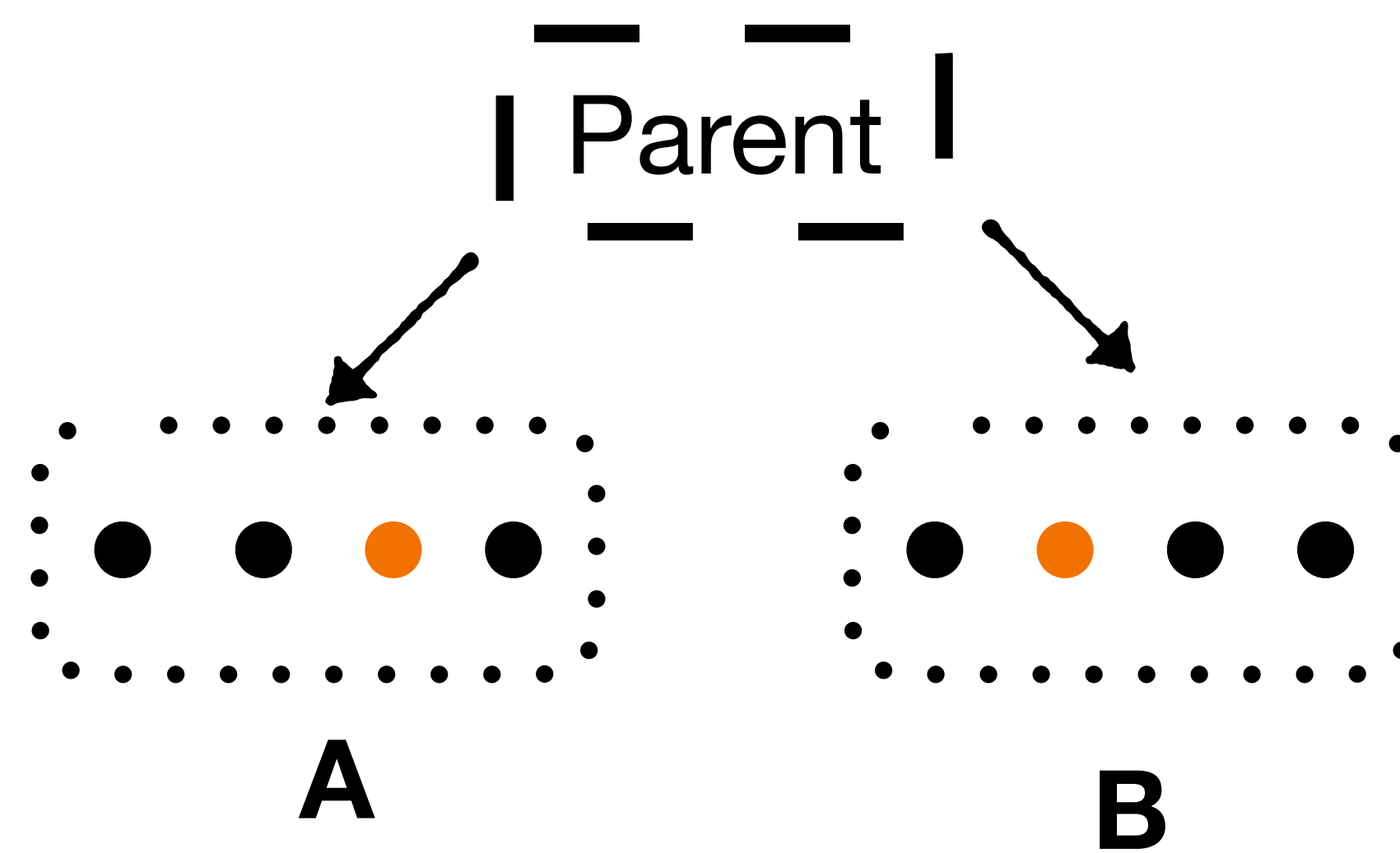
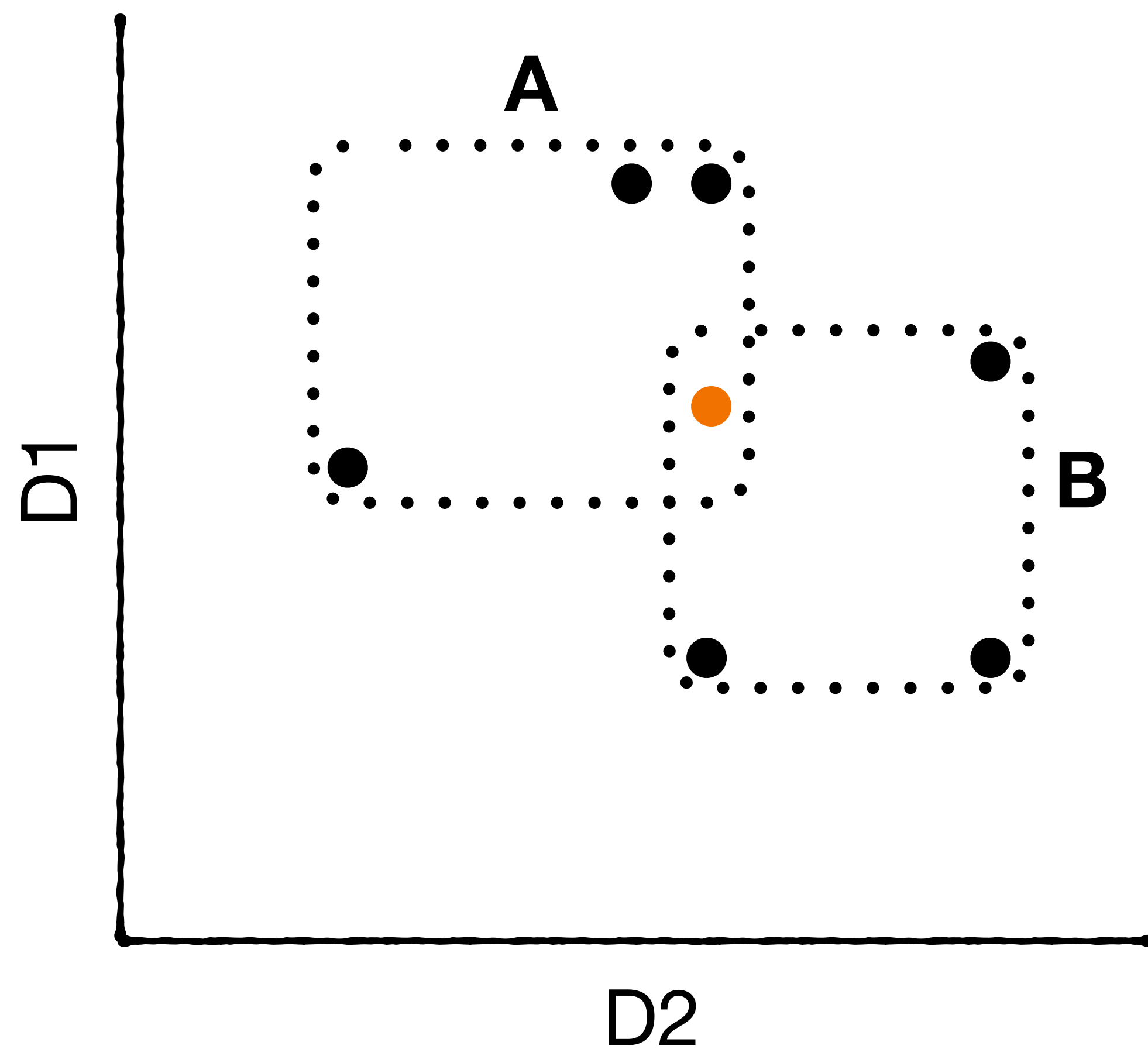
T Sellis, N Roussopoulos, C Faloutsos

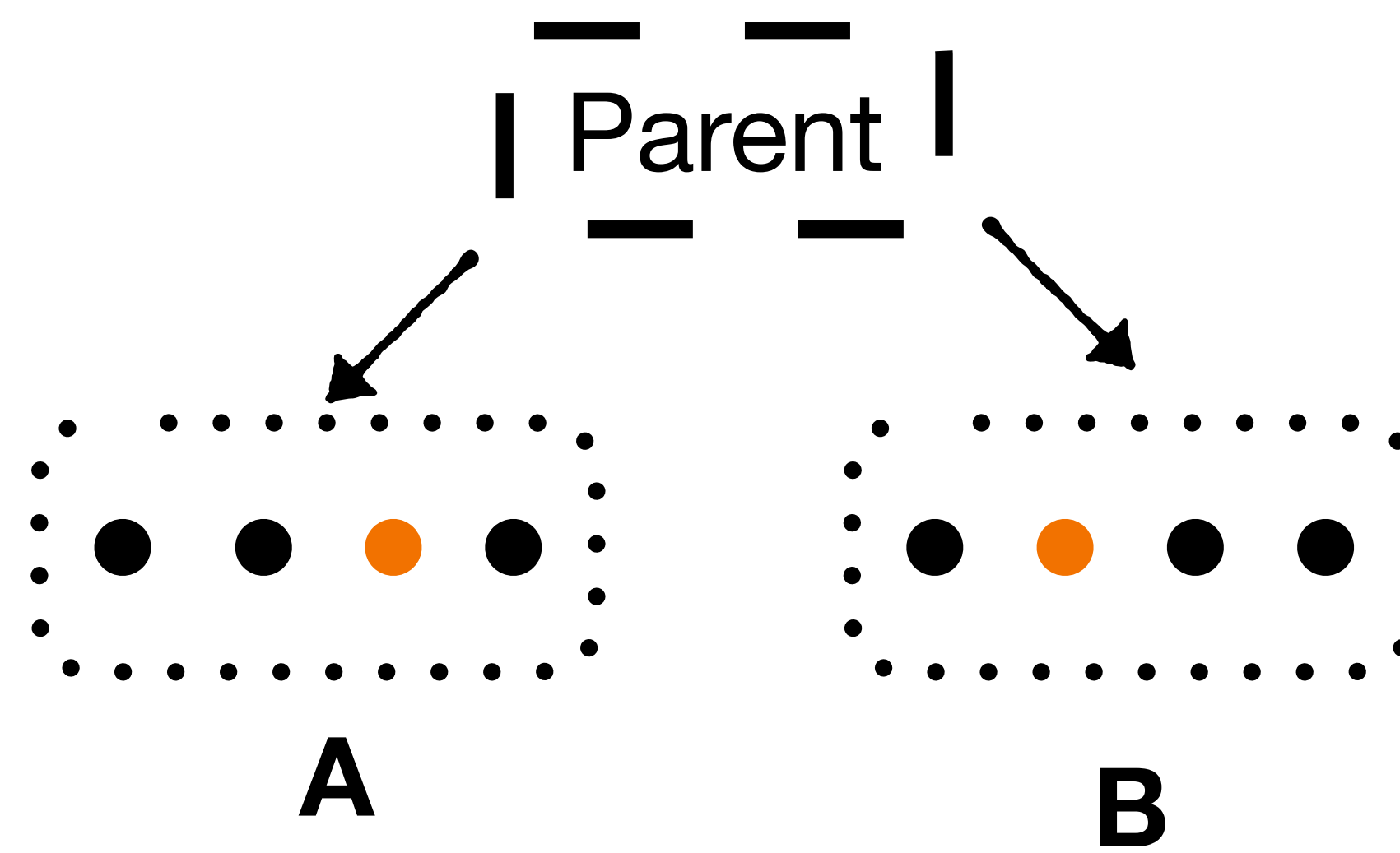
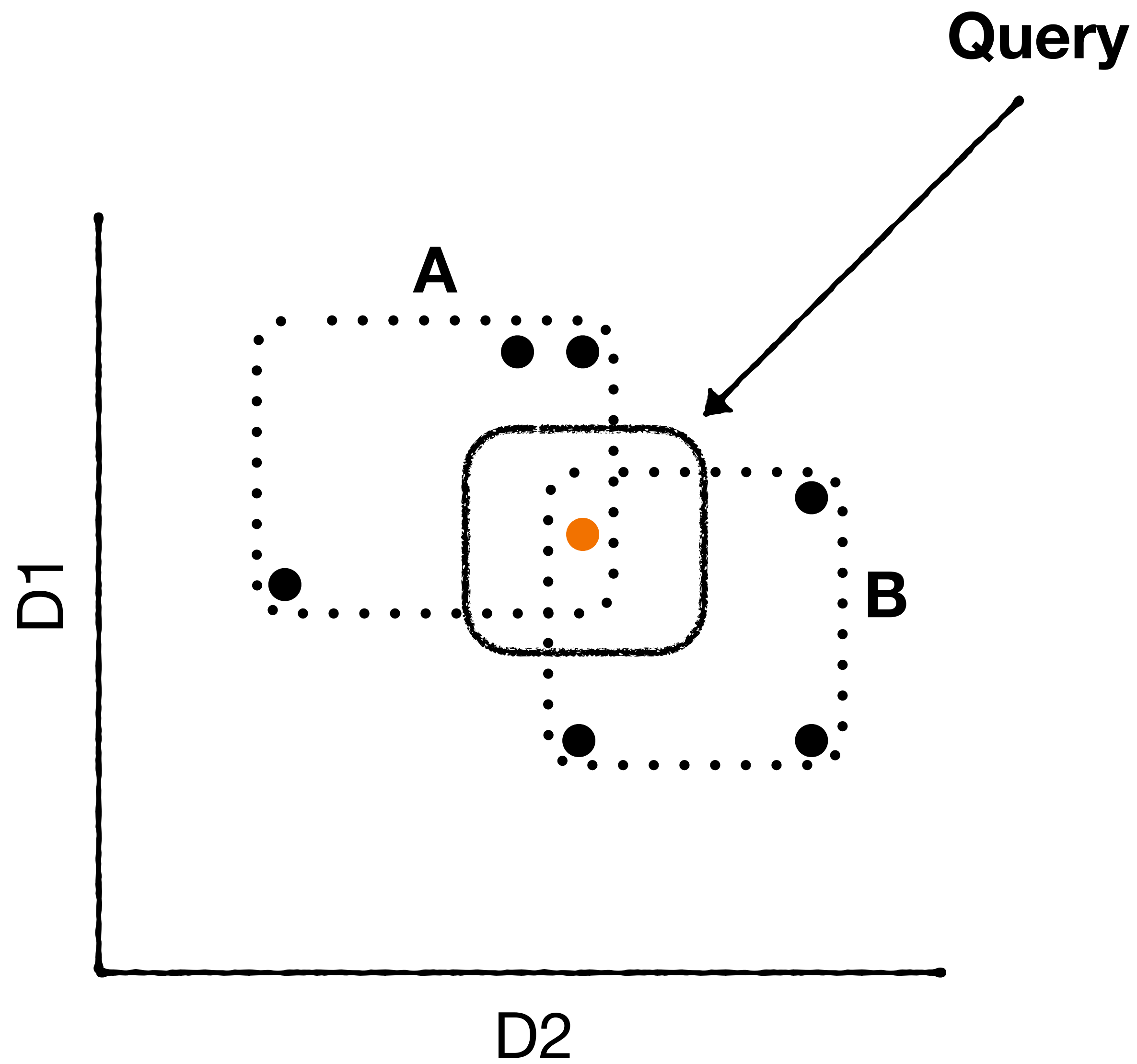
Published in **1987** (may be as technical report)

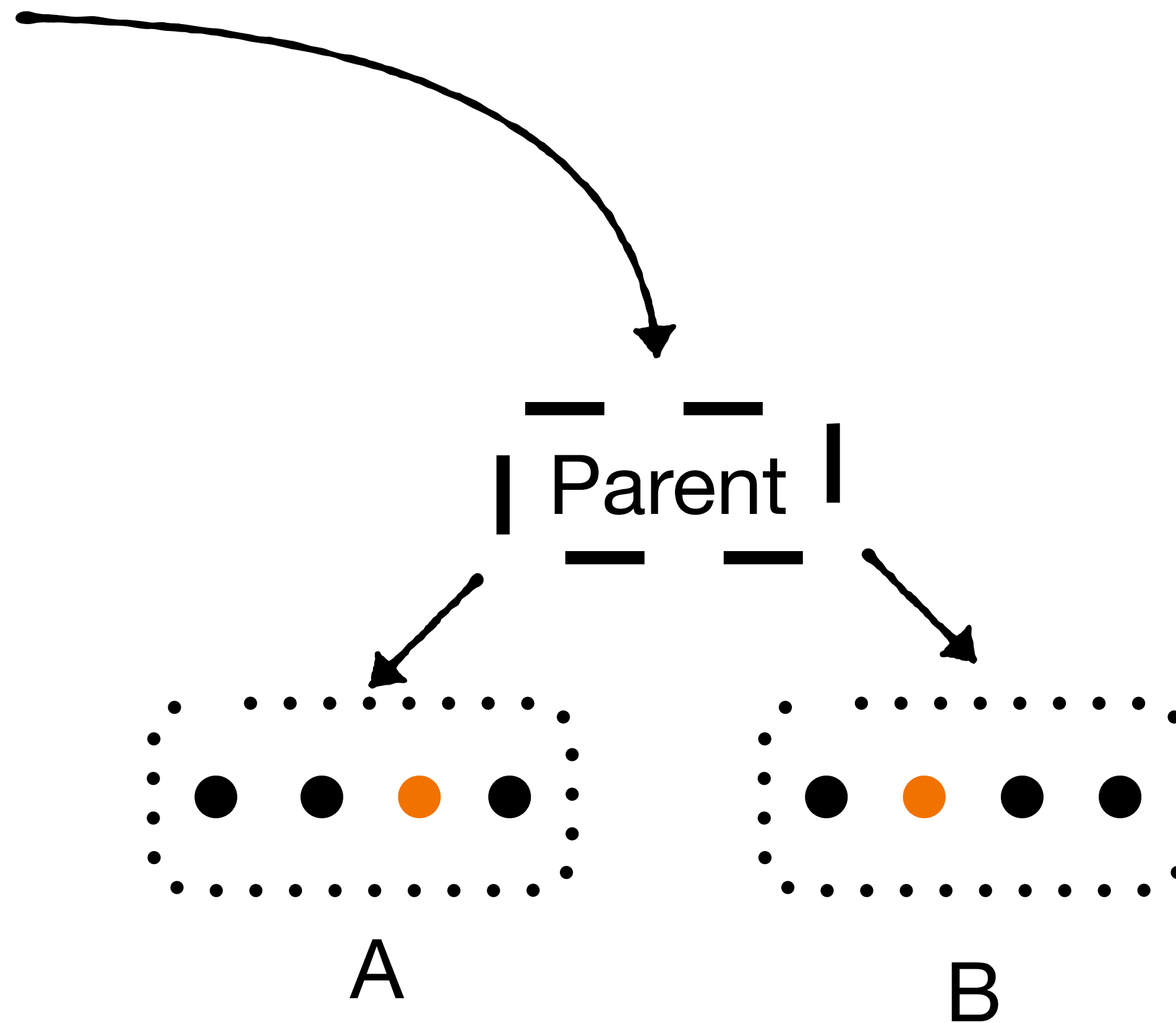
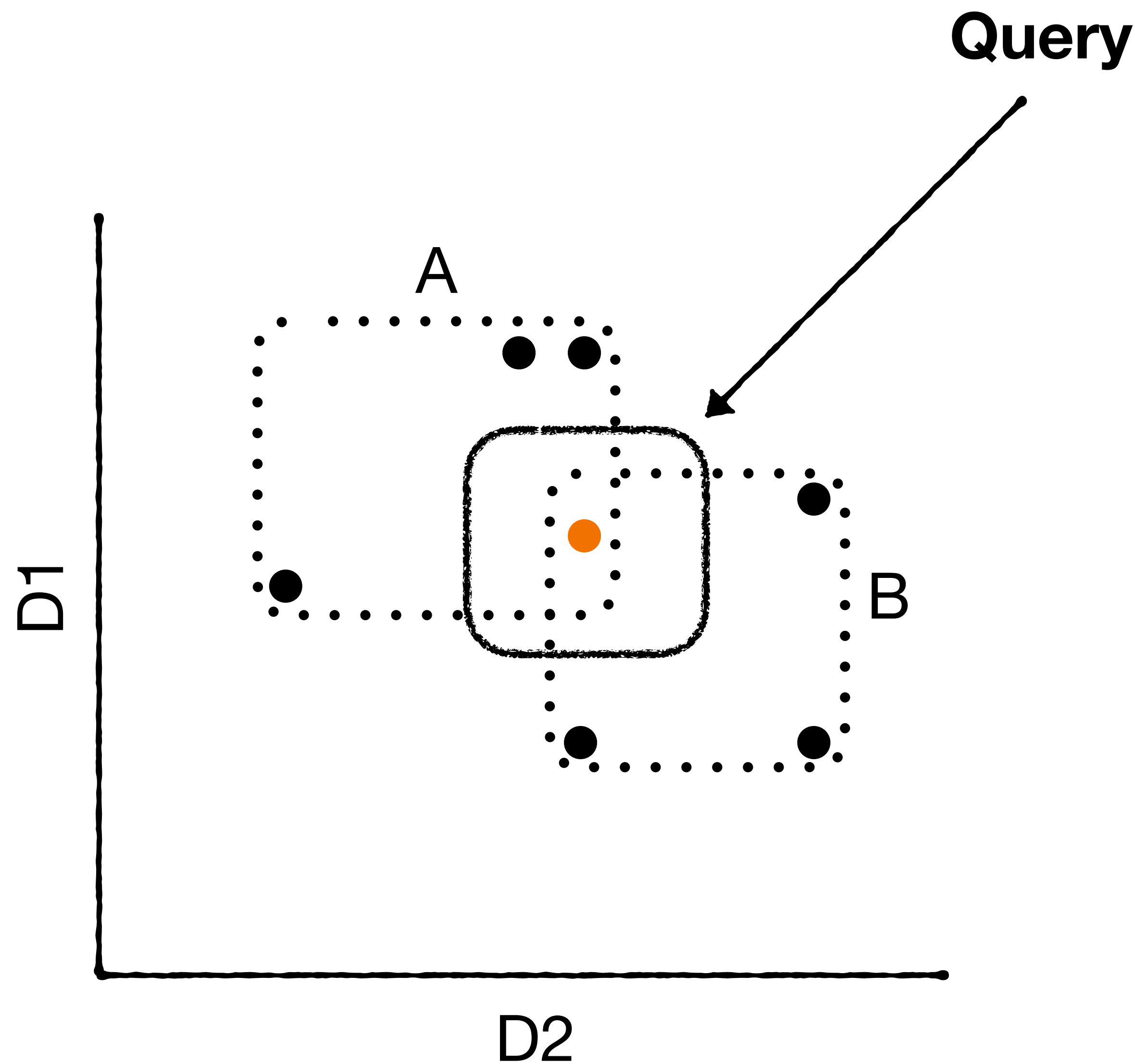
Insert entry to all intersecting nodes



Insert entry to all intersecting nodes

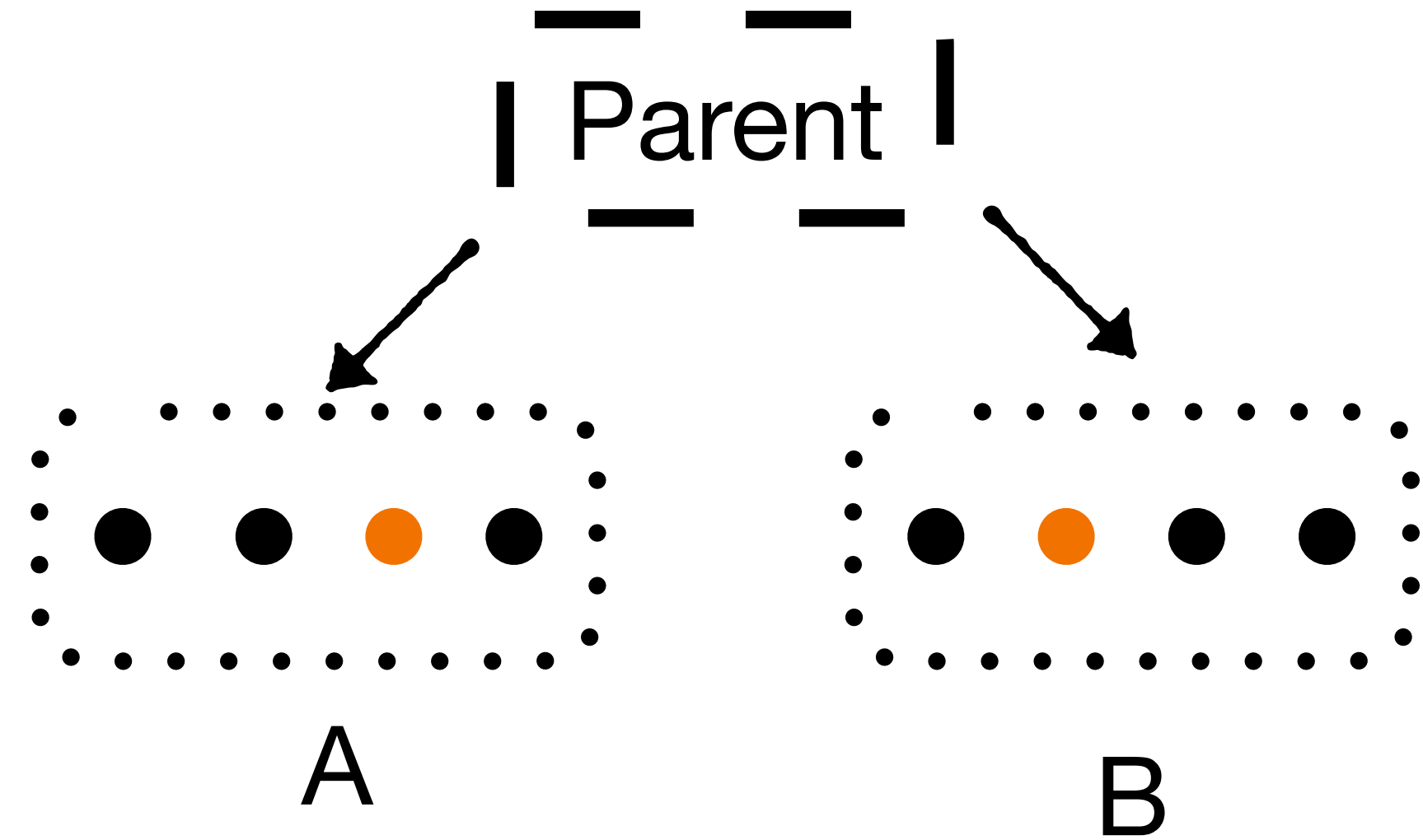
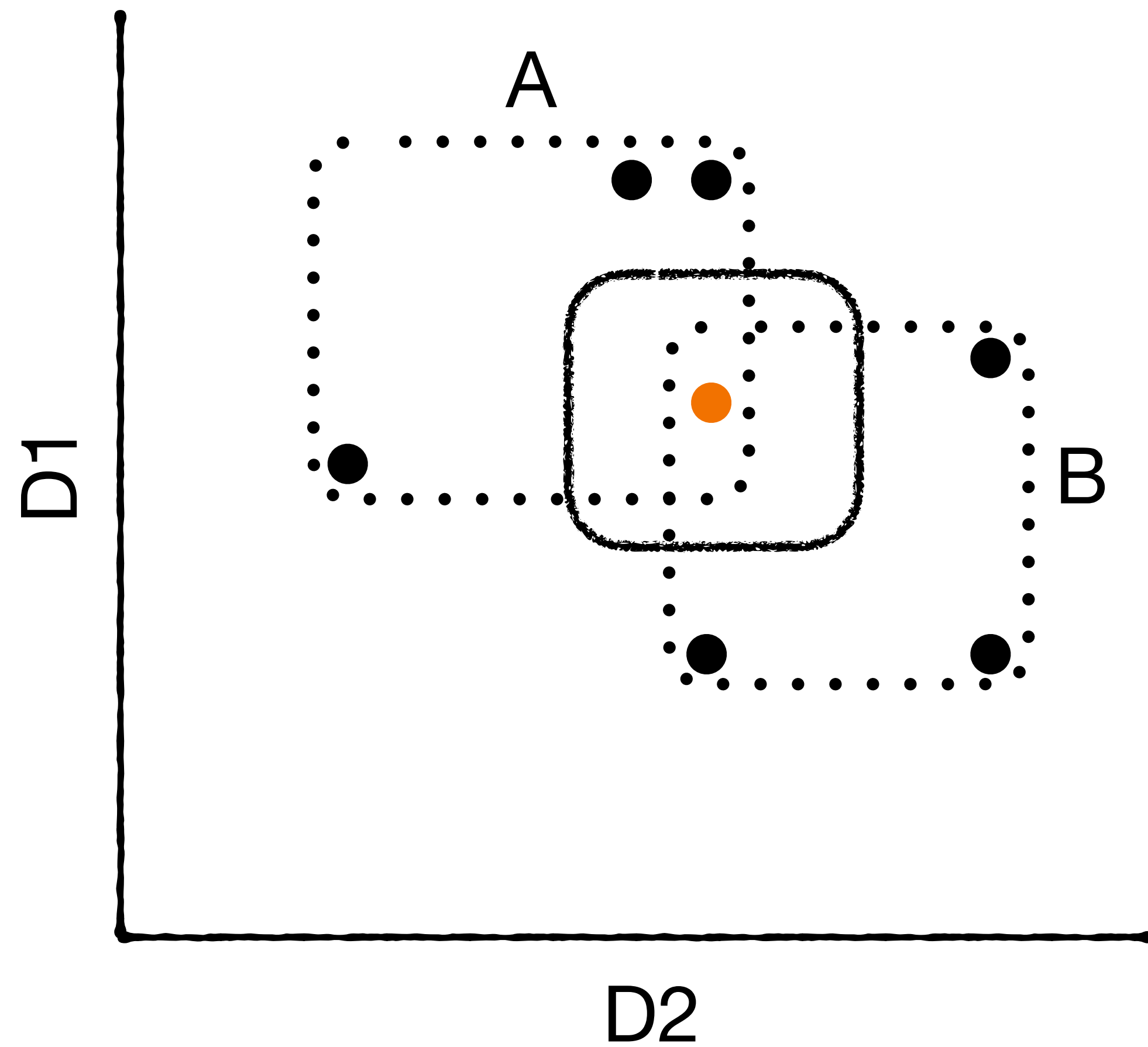






Whichever way a query goes, it finds the relevant entry

But duplication costs space



KD-Trees
1975

R-Trees
1984

R+Tree
1987

R*-Tree
1990

UB-Tree
2000

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

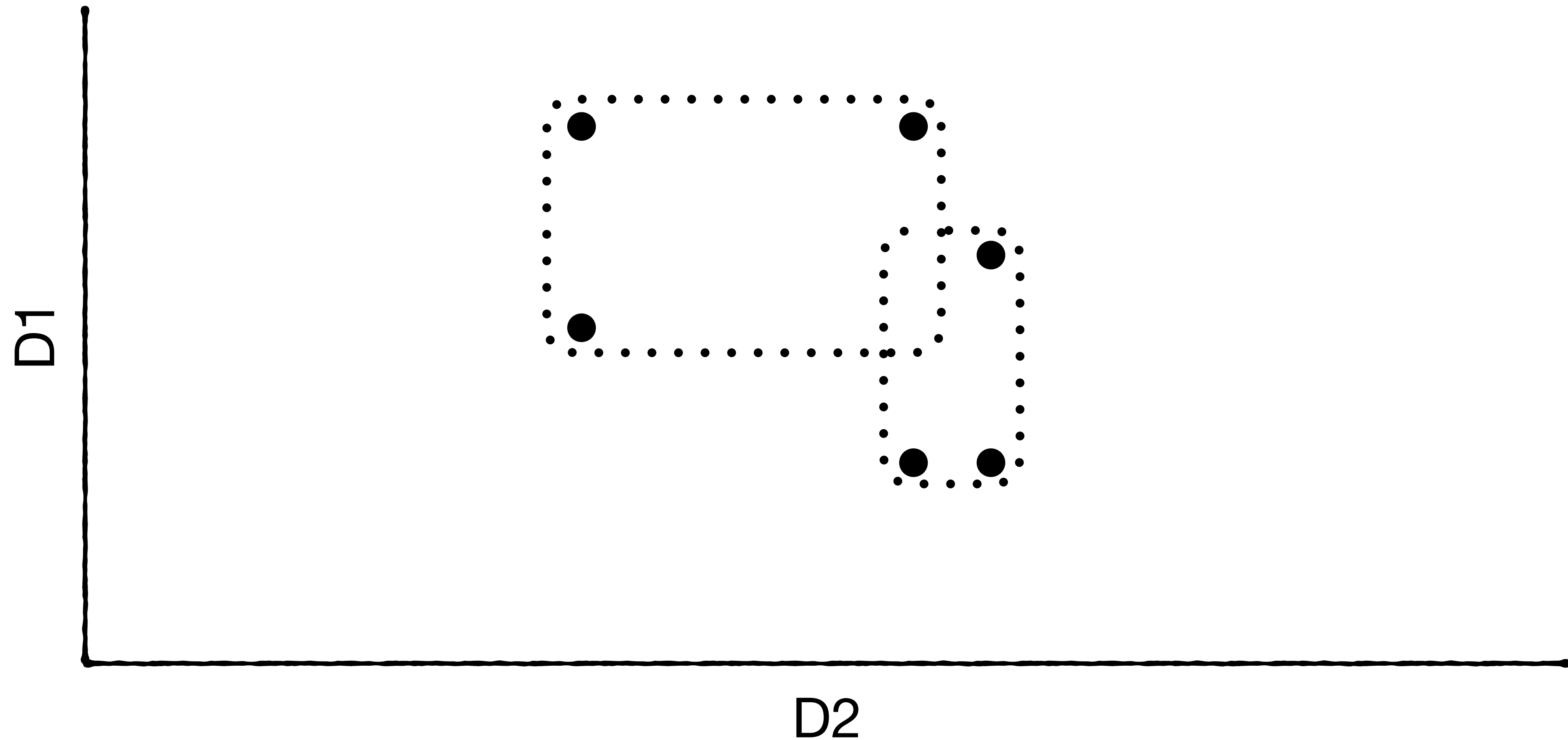
Beckmann, N.; Kriegel, H. P.; Schneider, R.; Seeger

SIGMOD 1990

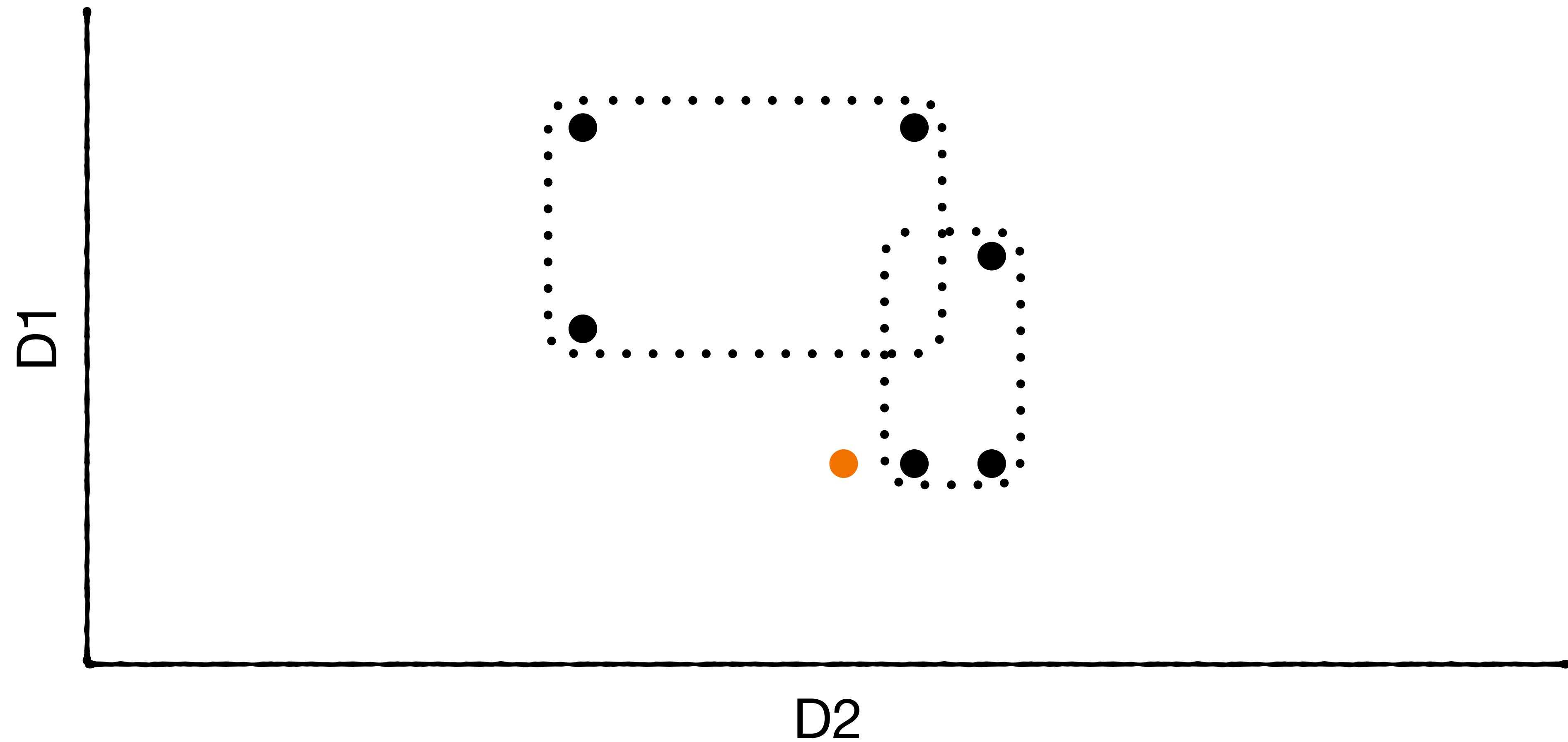
Insertion order affects overlap among rectangles



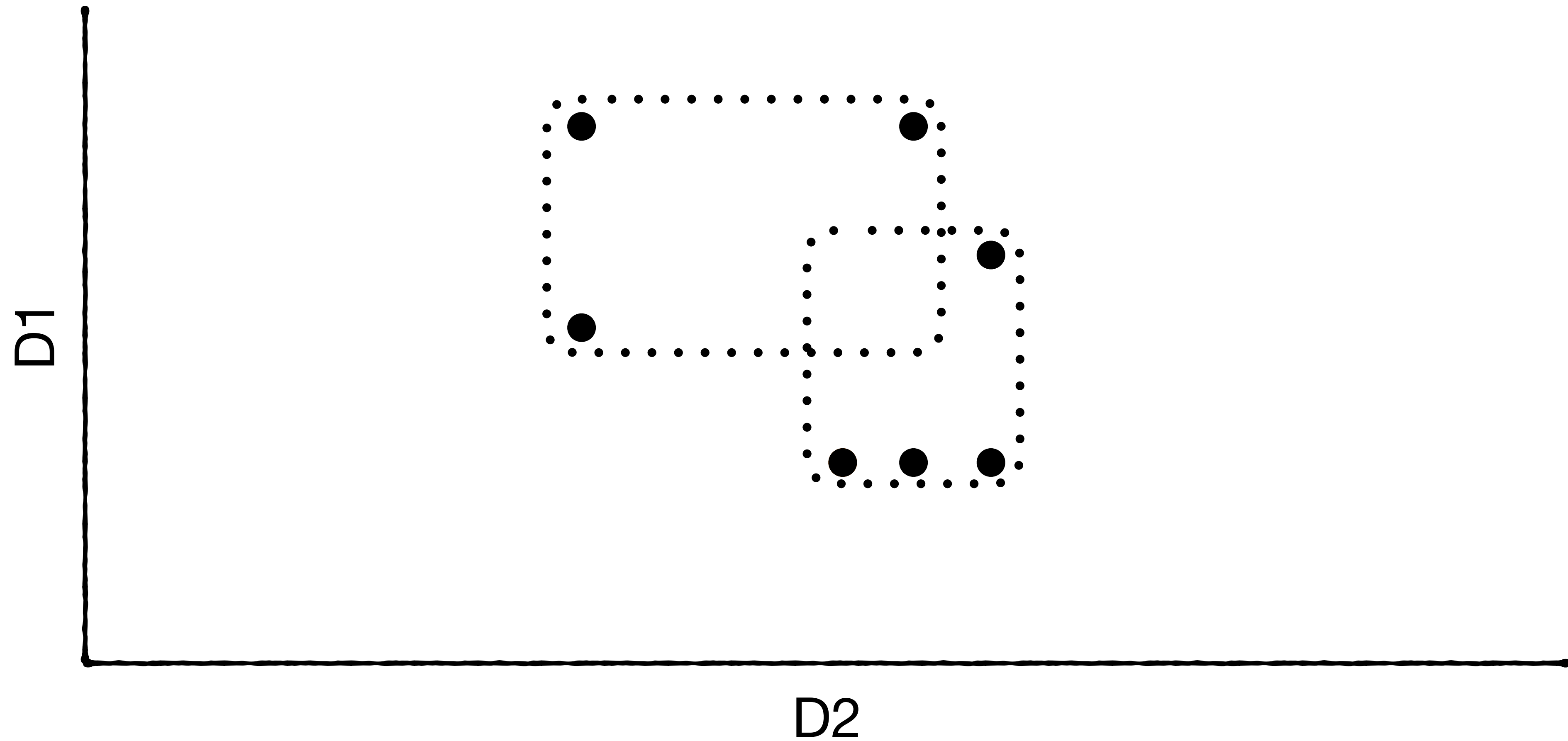
Insertion order affects overlap among rectangles



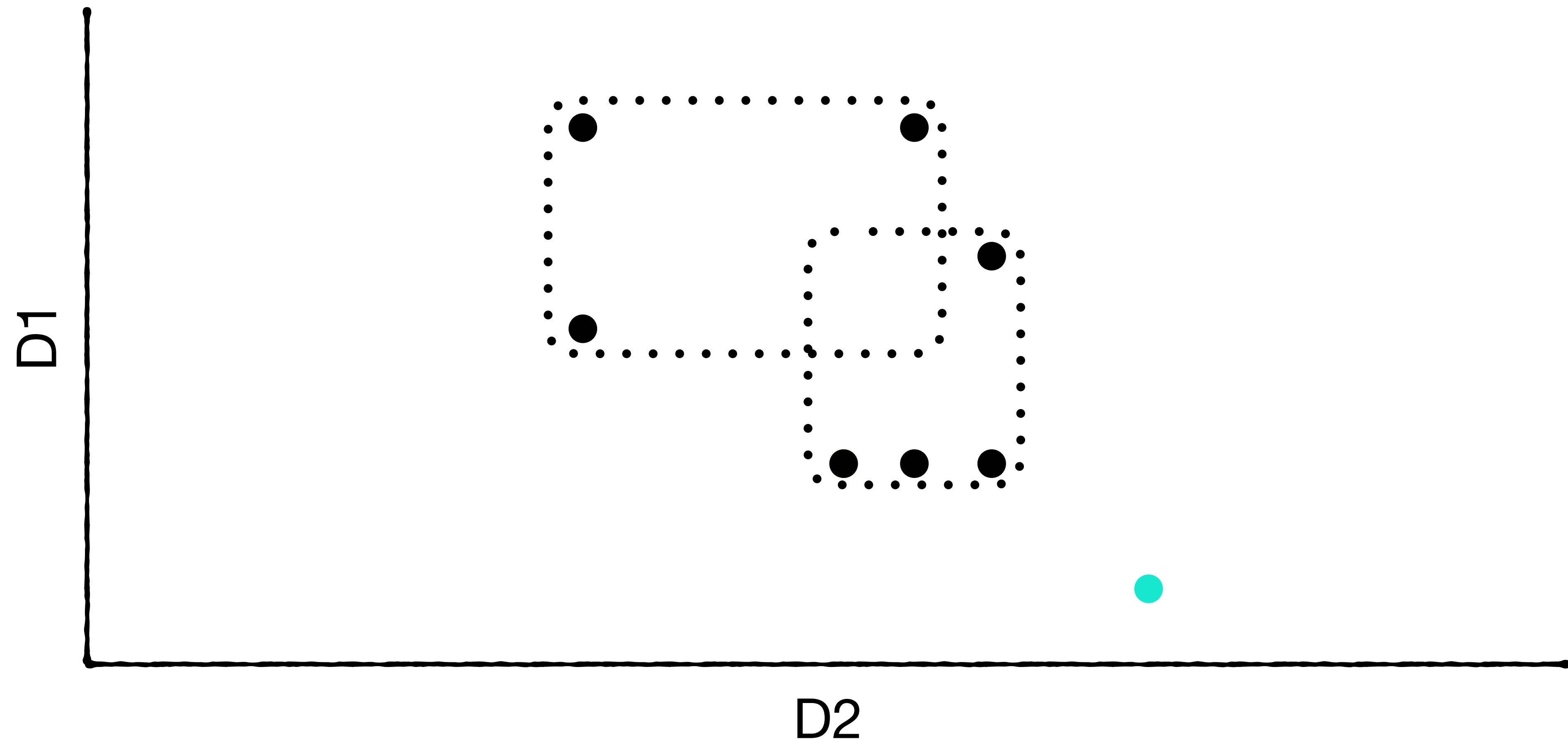
Insertion order affects overlap among rectangles



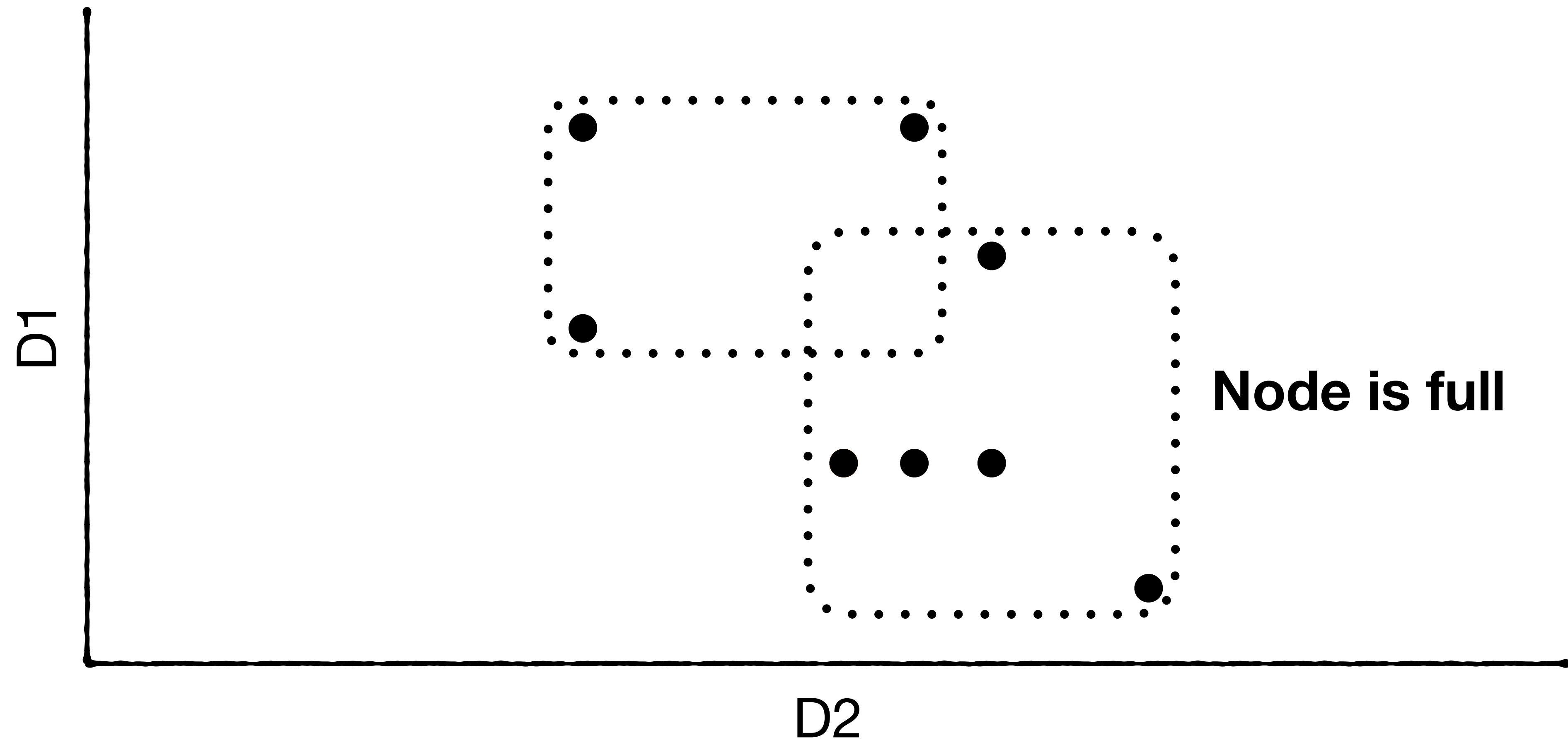
Insertion order affects overlap among rectangles



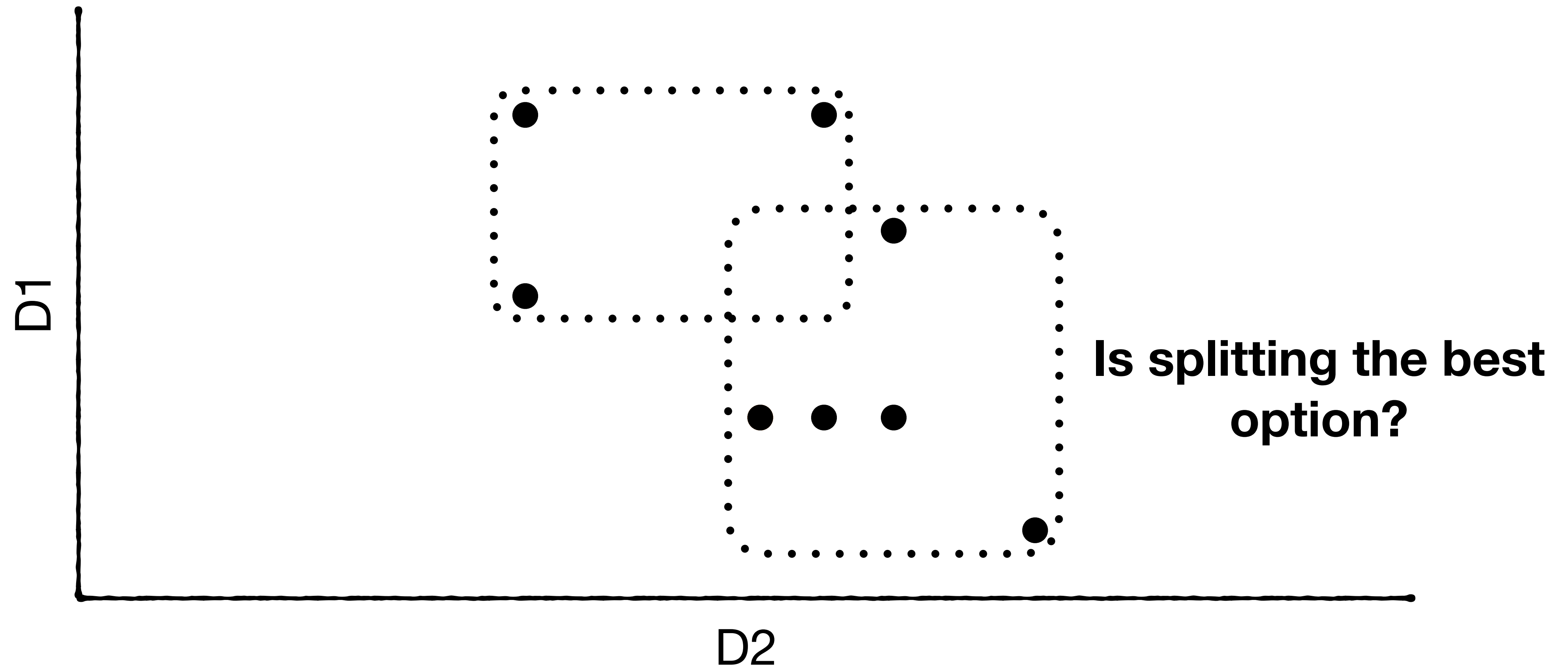
Insertion order affects overlap among rectangles



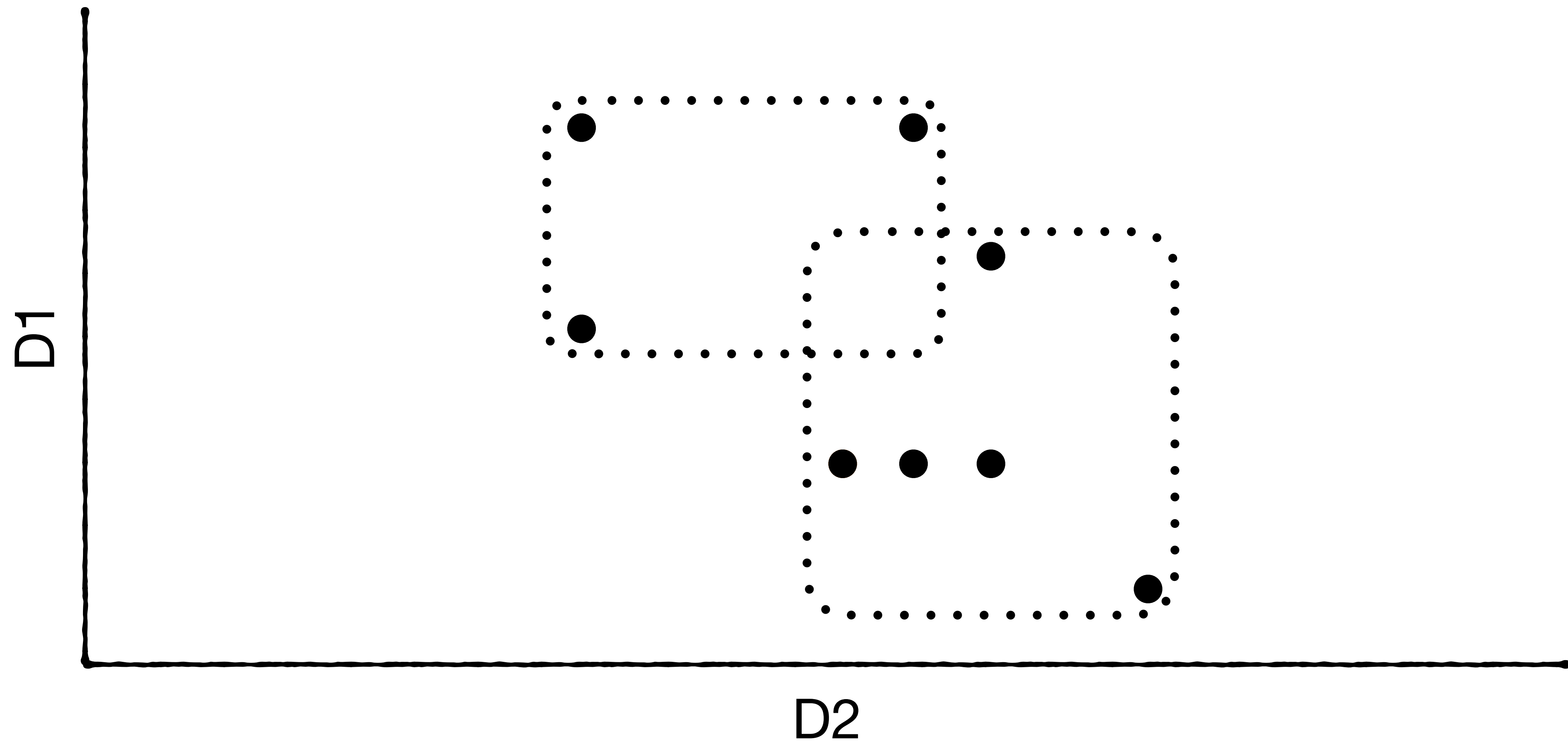
Insertion order affects overlap among rectangles



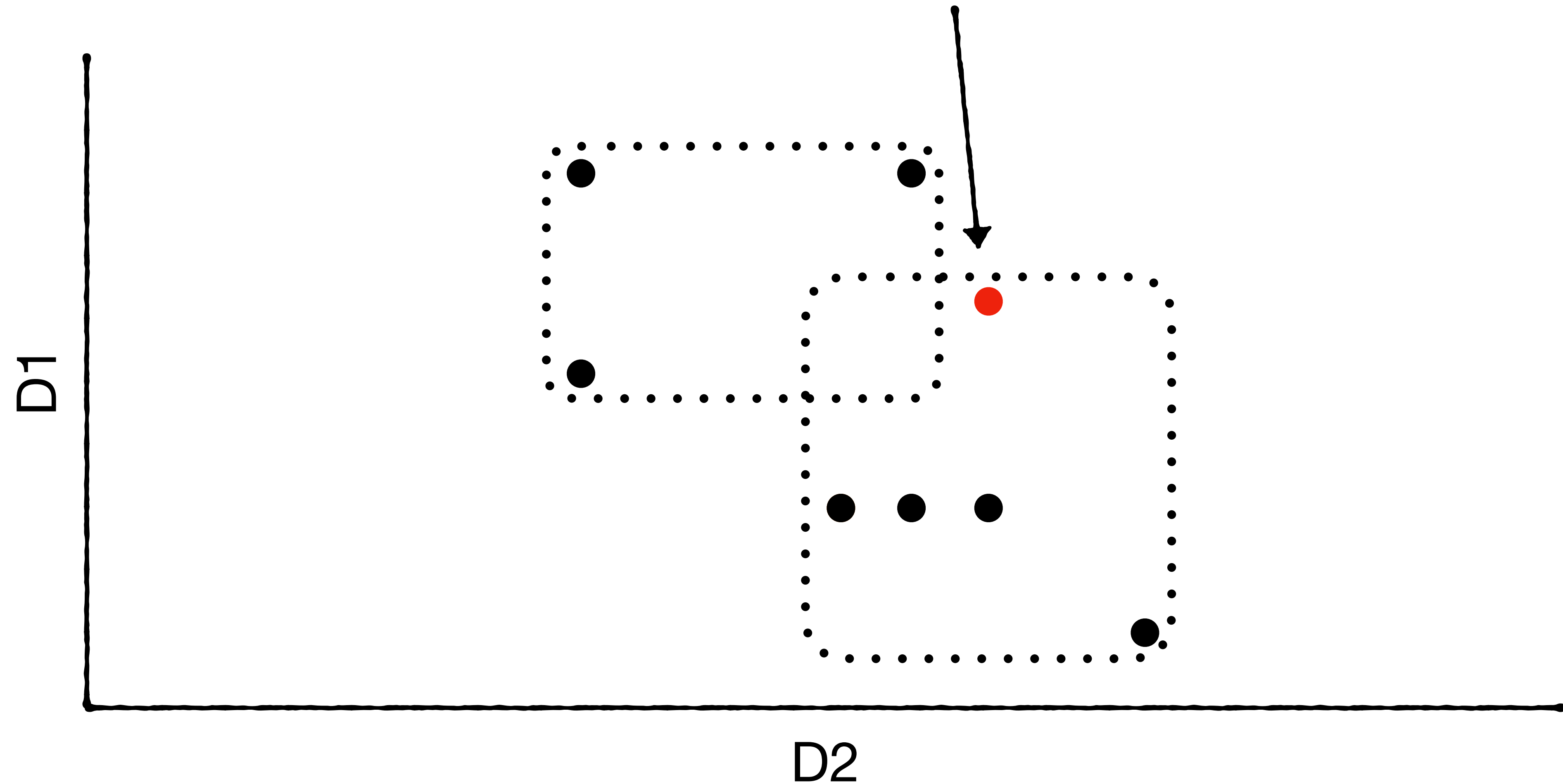
Insertion order affects overlap among rectangles



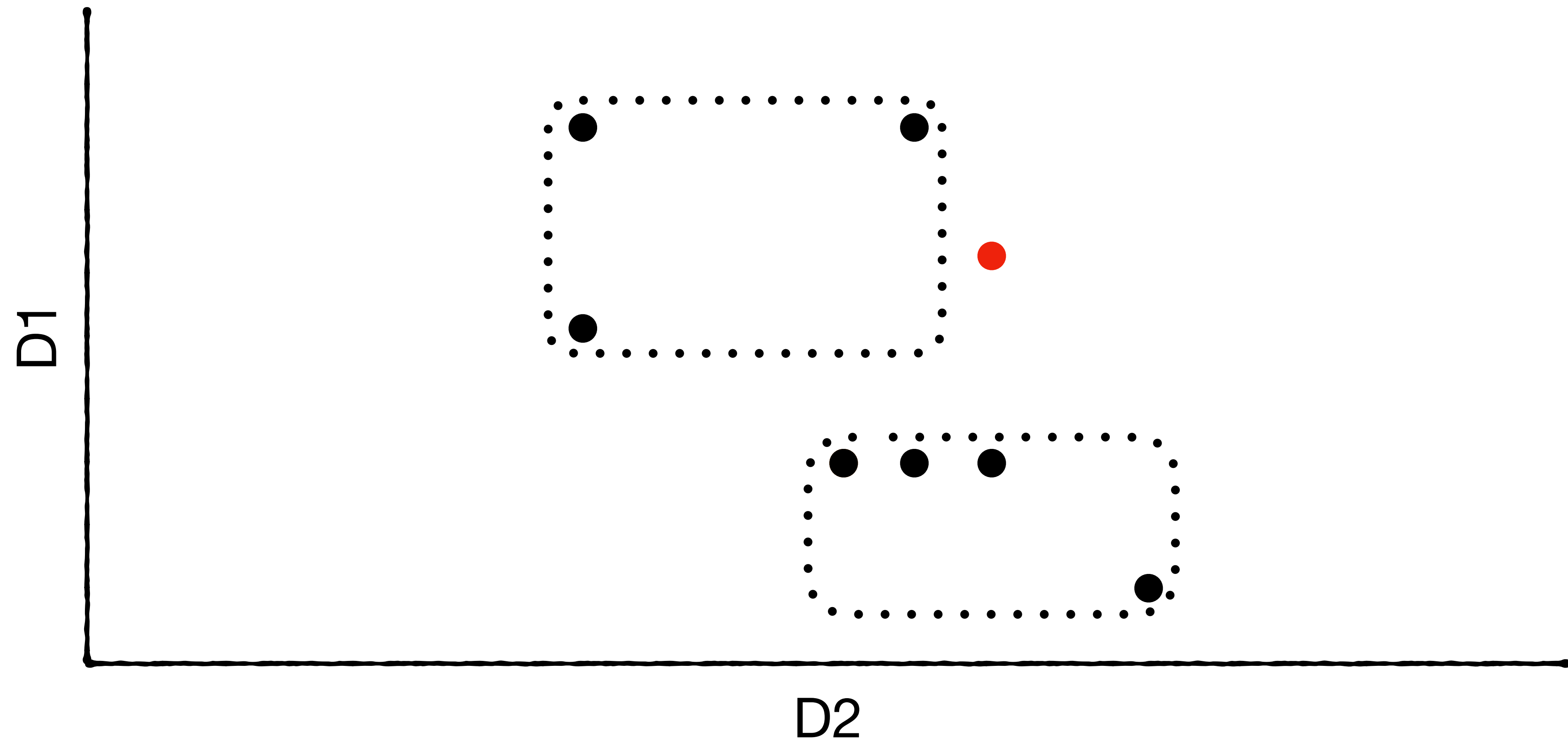
Suppose instead we reinsert some anomalous dot in filled up node
(Which?)



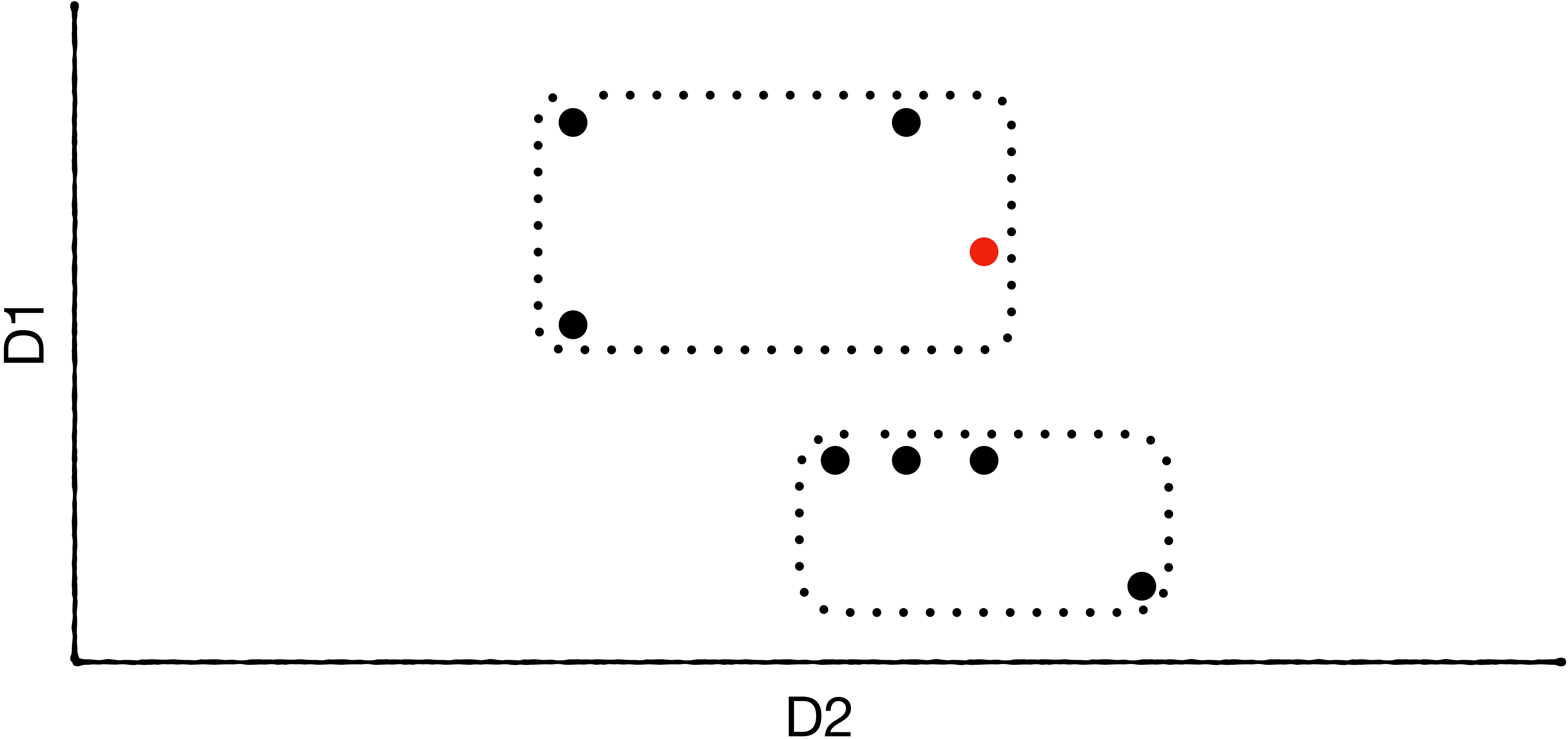
Suppose instead we reinsert some anomalous dot in filled up node
(Which?)



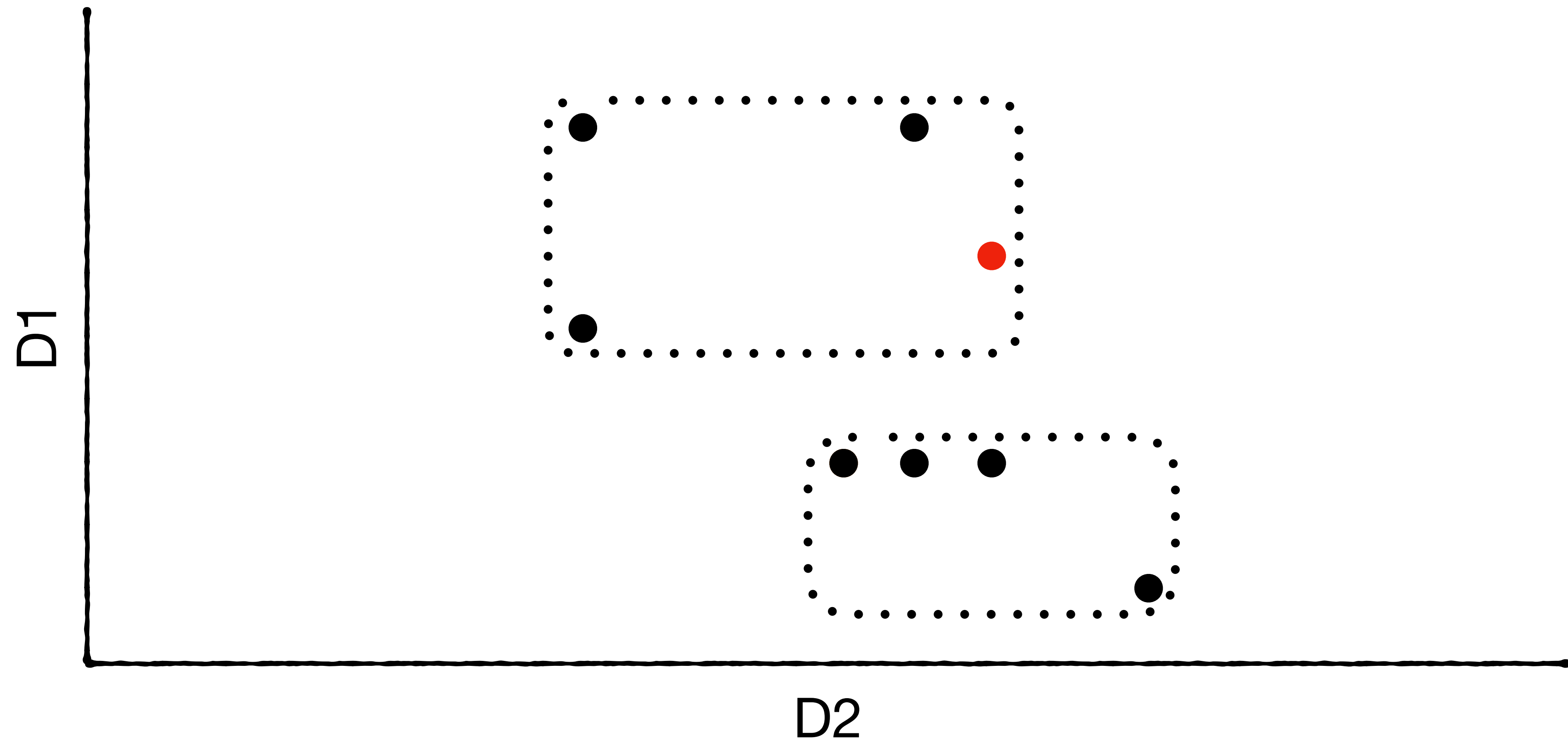
Suppose instead we reinsert some anomalous dot in filled up node



reinsertion places entry in other rectangle, eliminating overlap

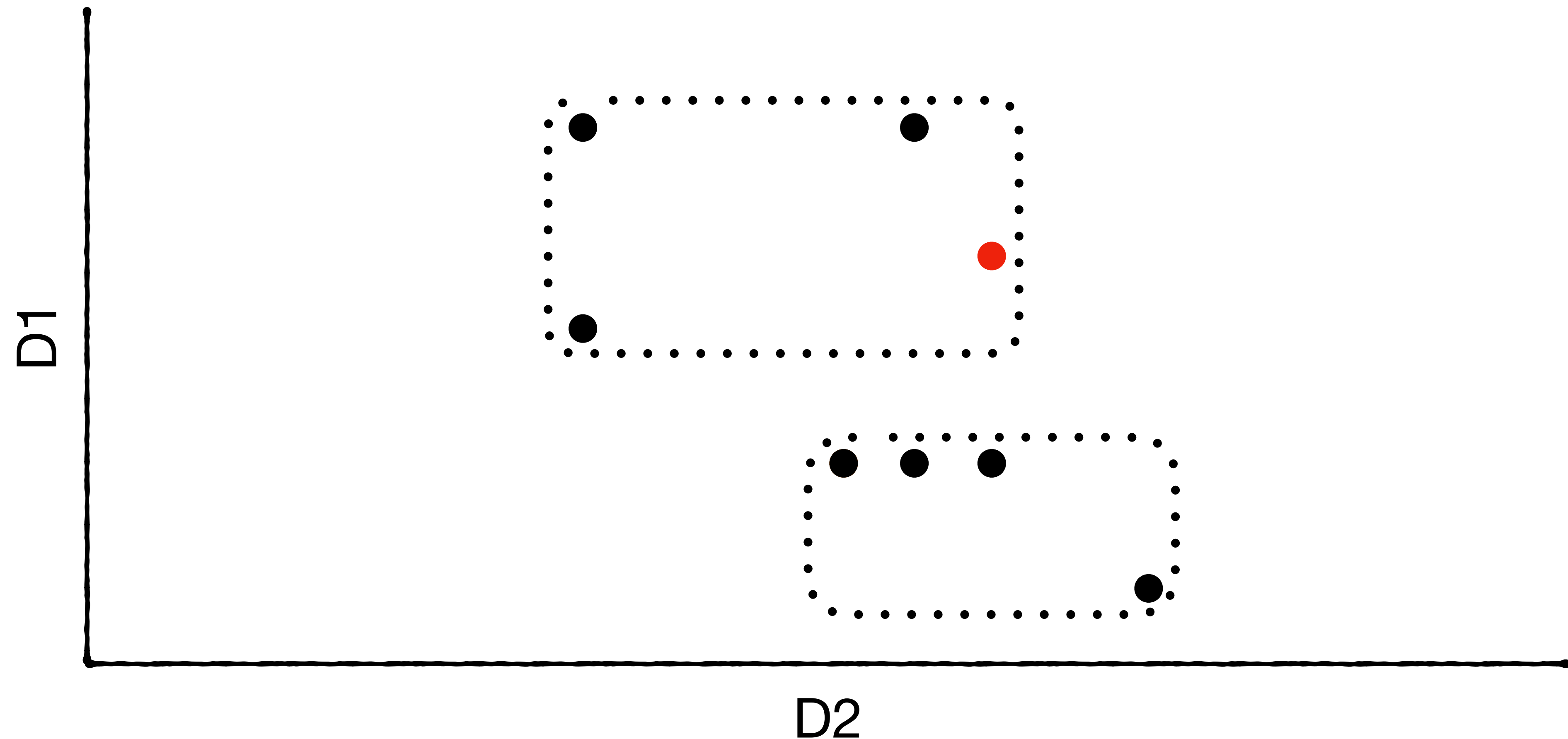


R*-Tree reinserts entries from full nodes to find better rectangle for them



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

Detailed algorithm out of scope



KD-Trees
1975

R-Trees
1984

R+Tree
1987

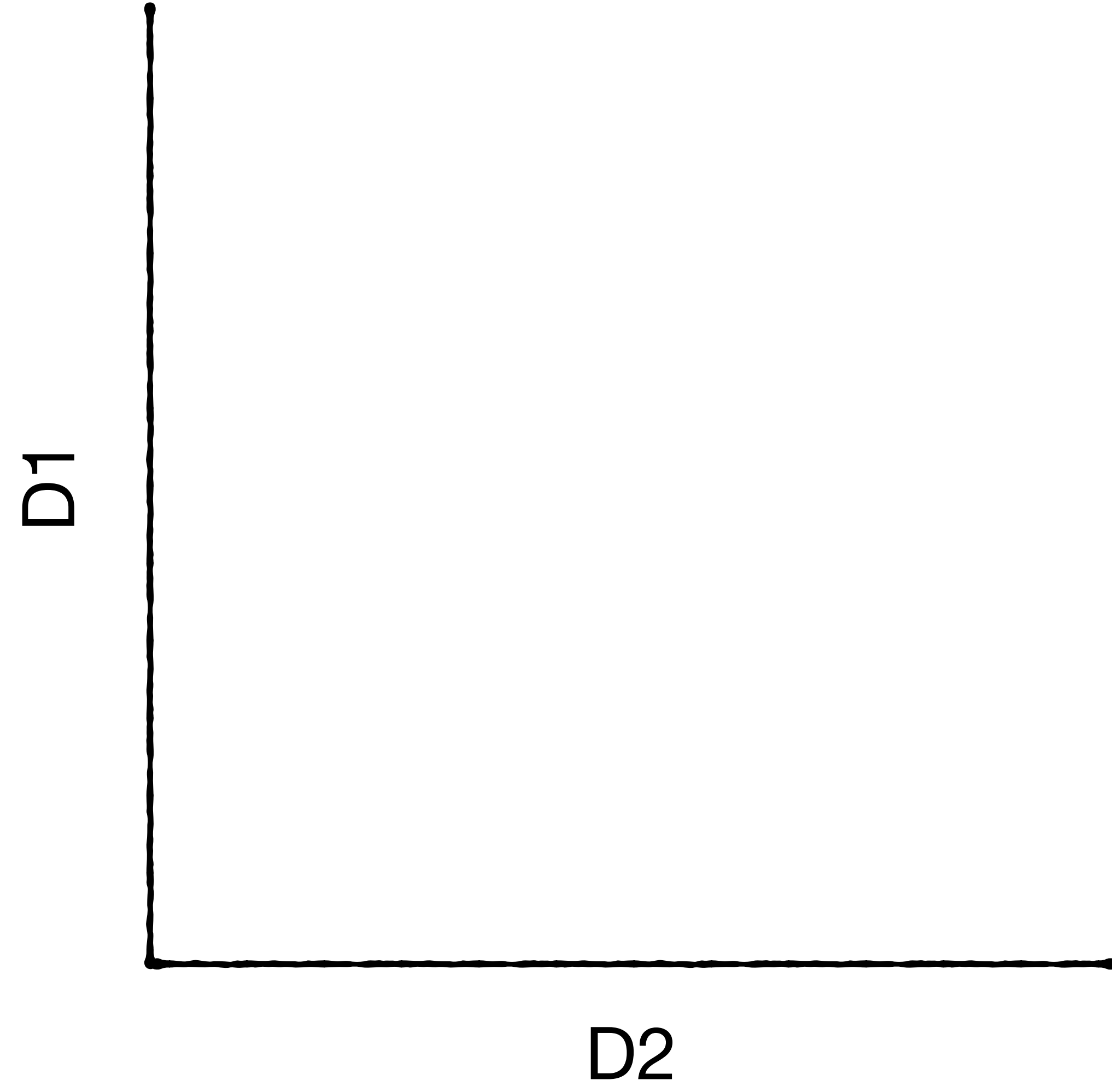
R*-Tree
1990

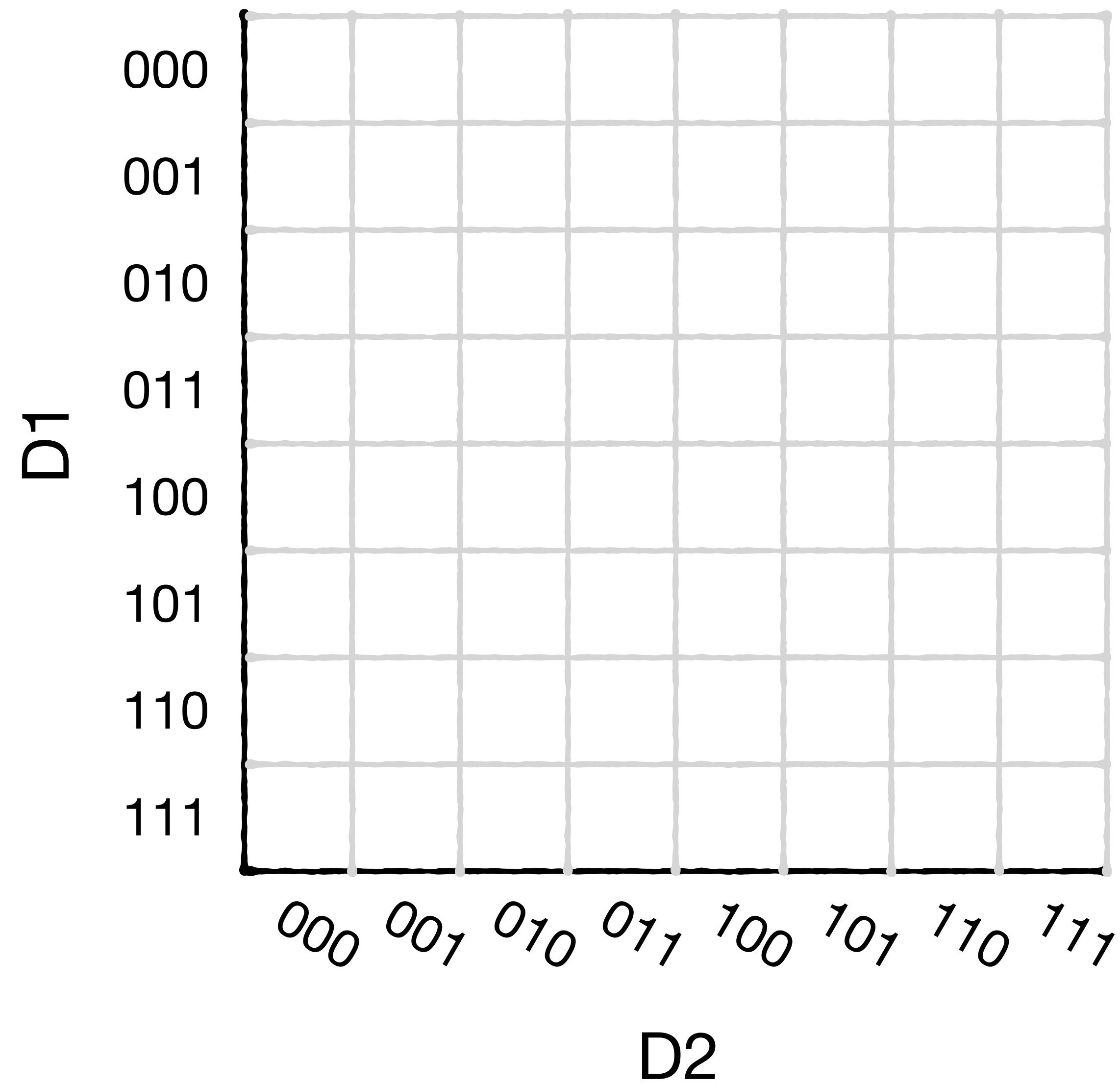
UB-Tree
2000

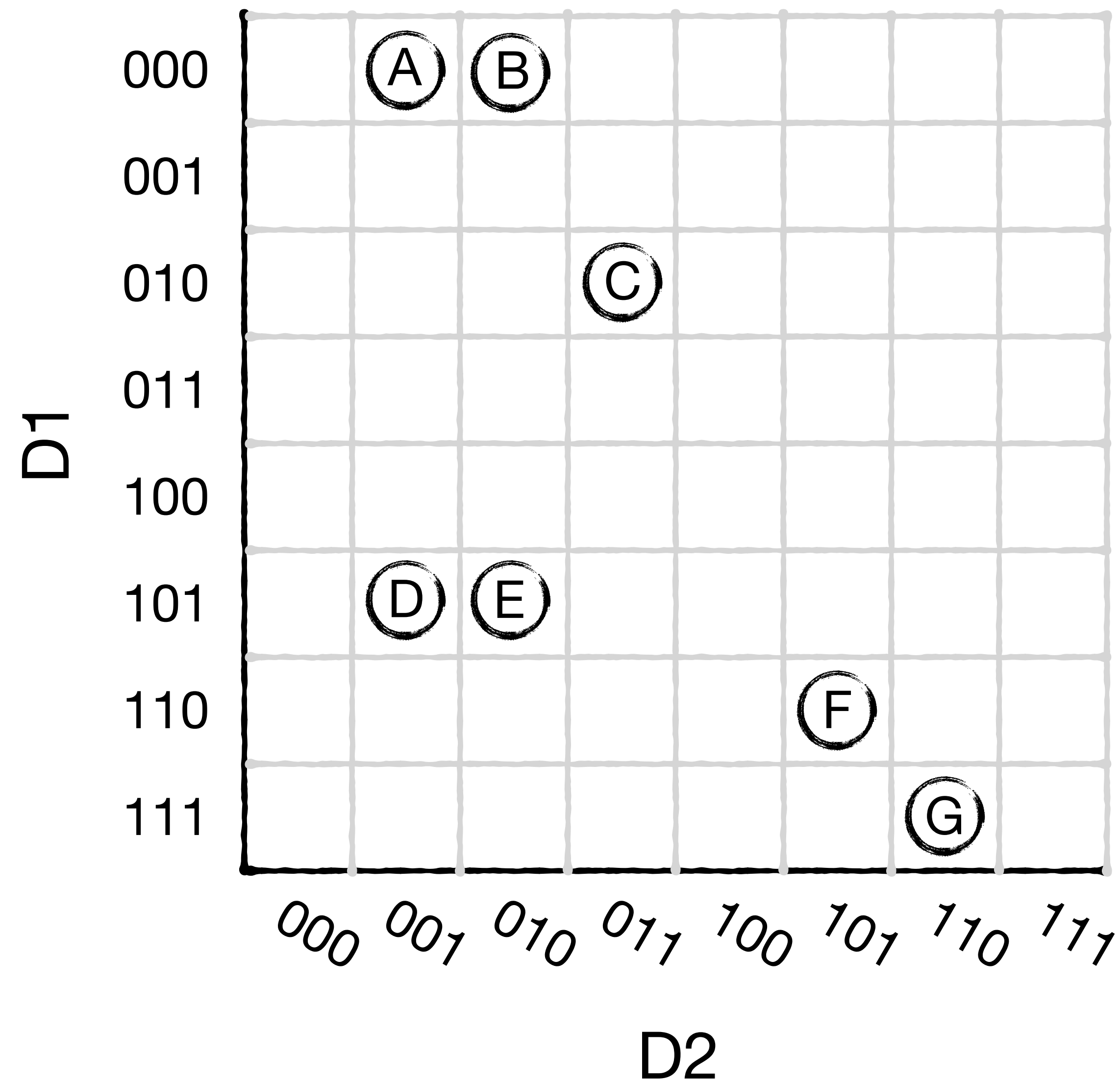
Integrating the UB-Tree into a Database System Kernel

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

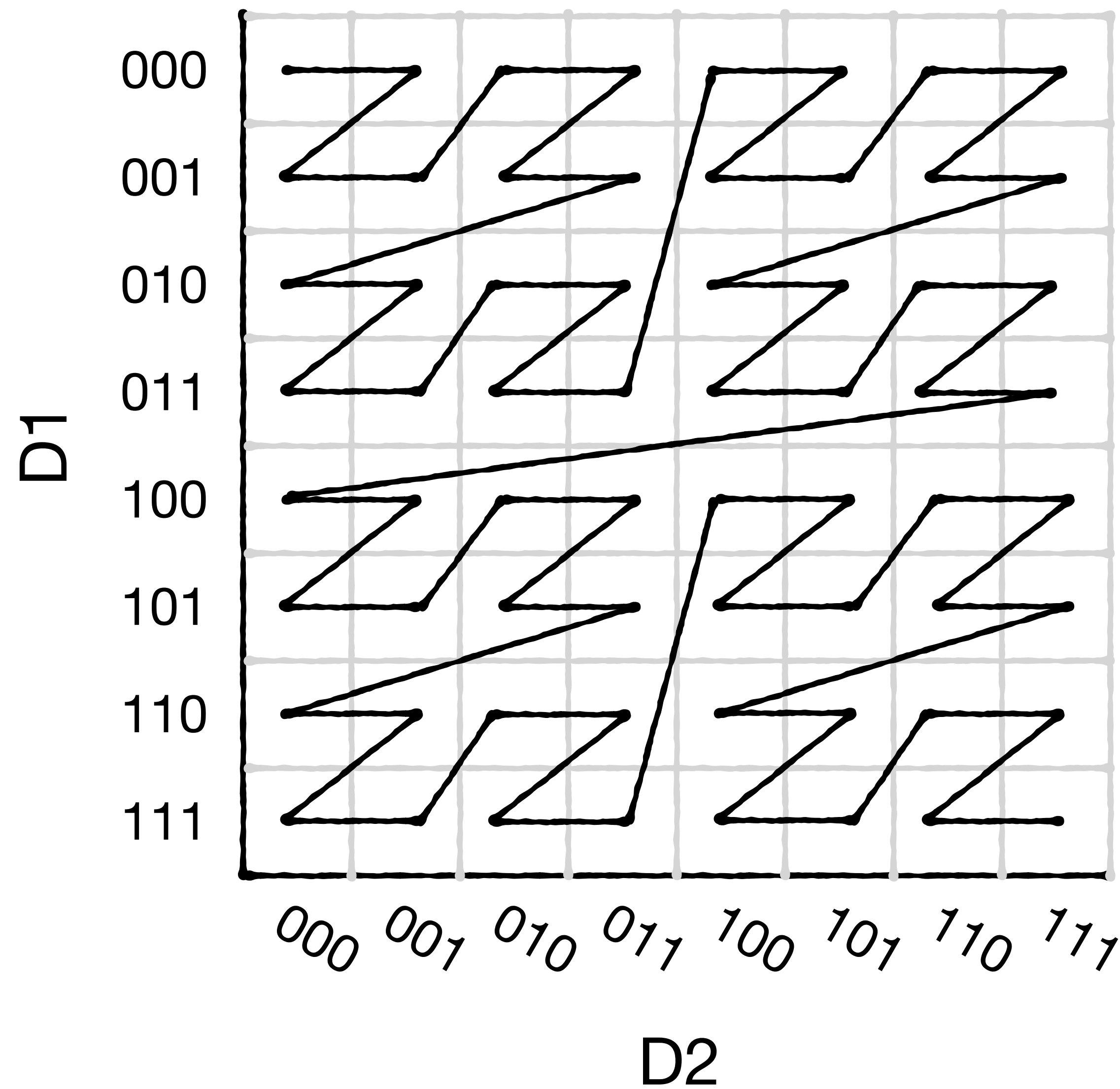
VLDB 2000



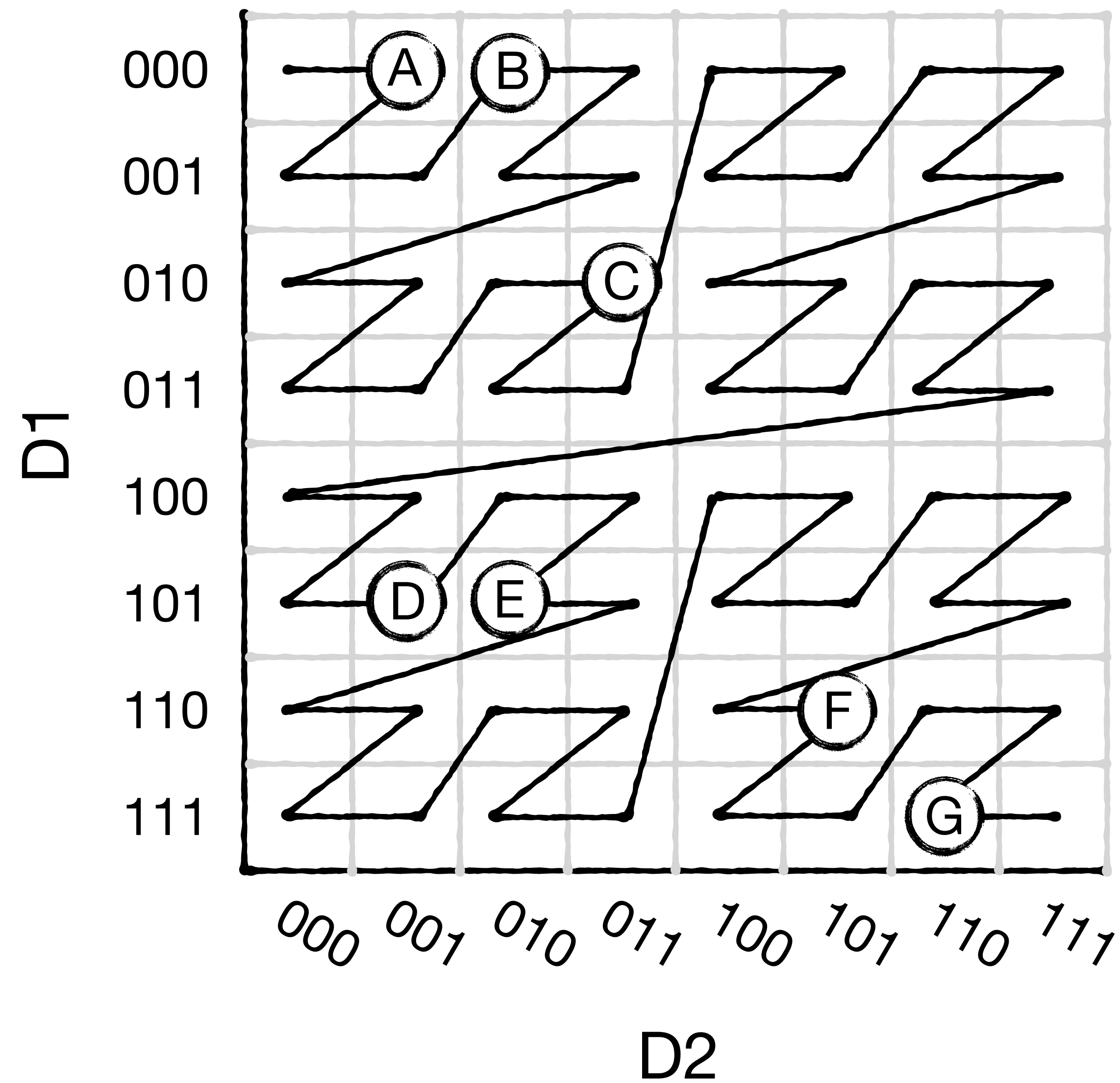


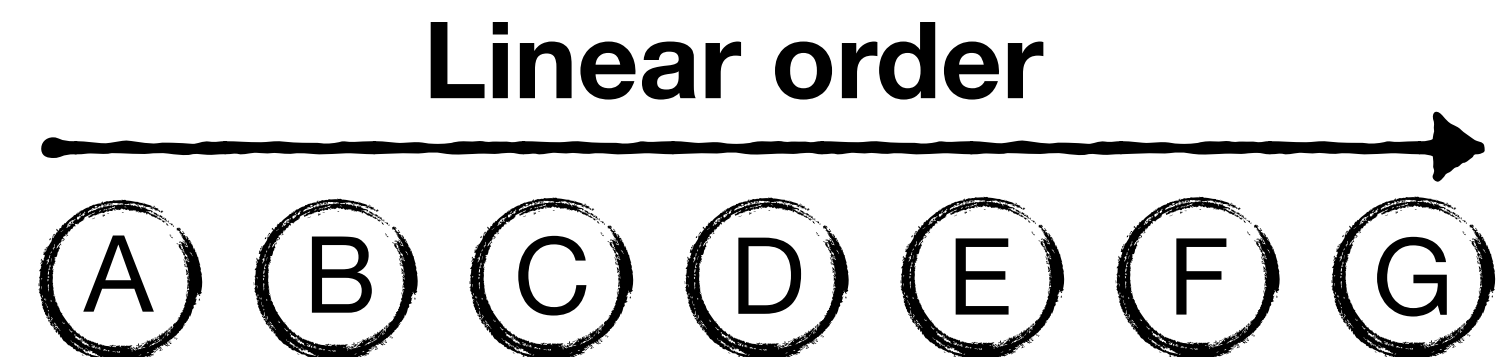
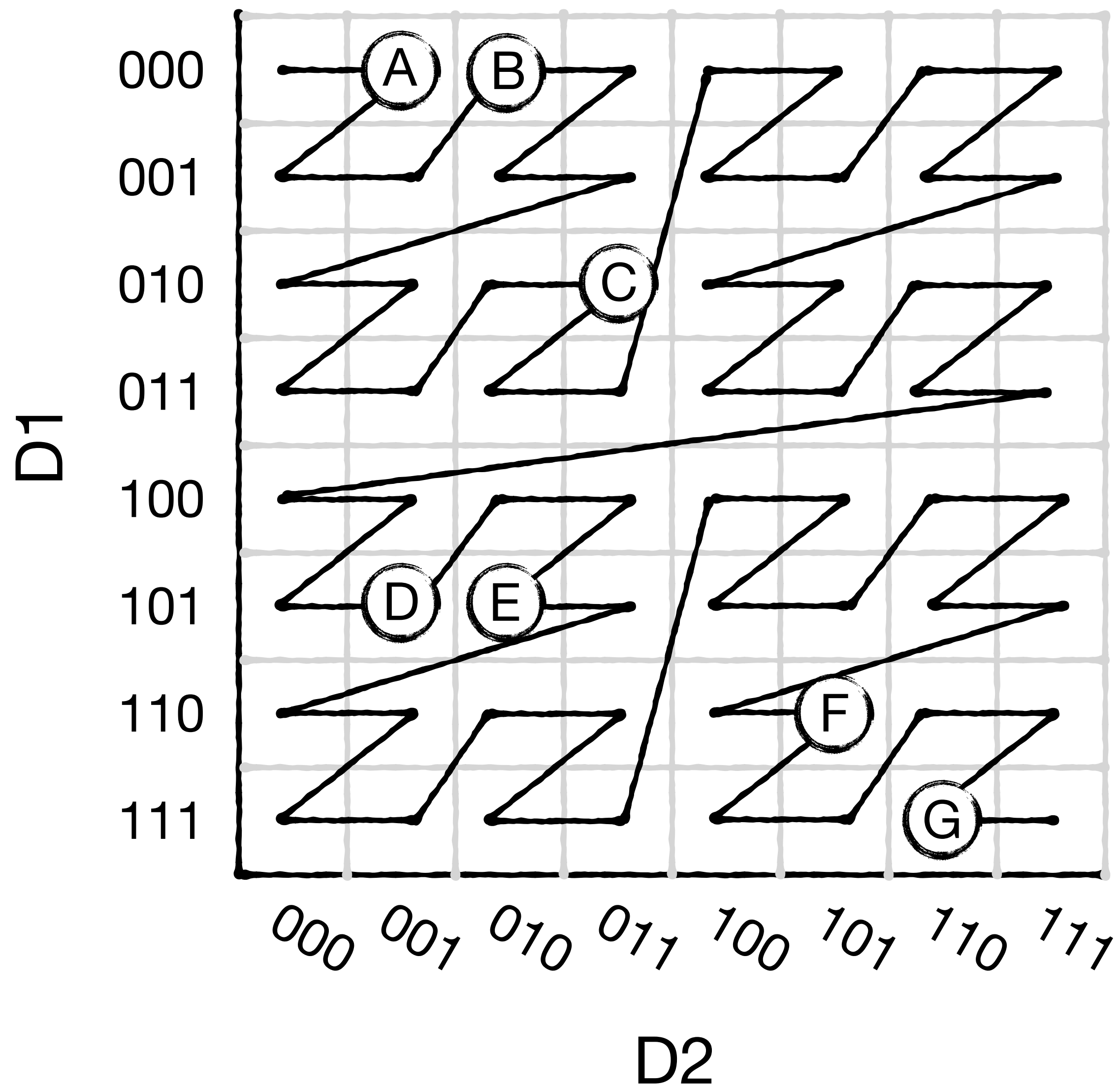


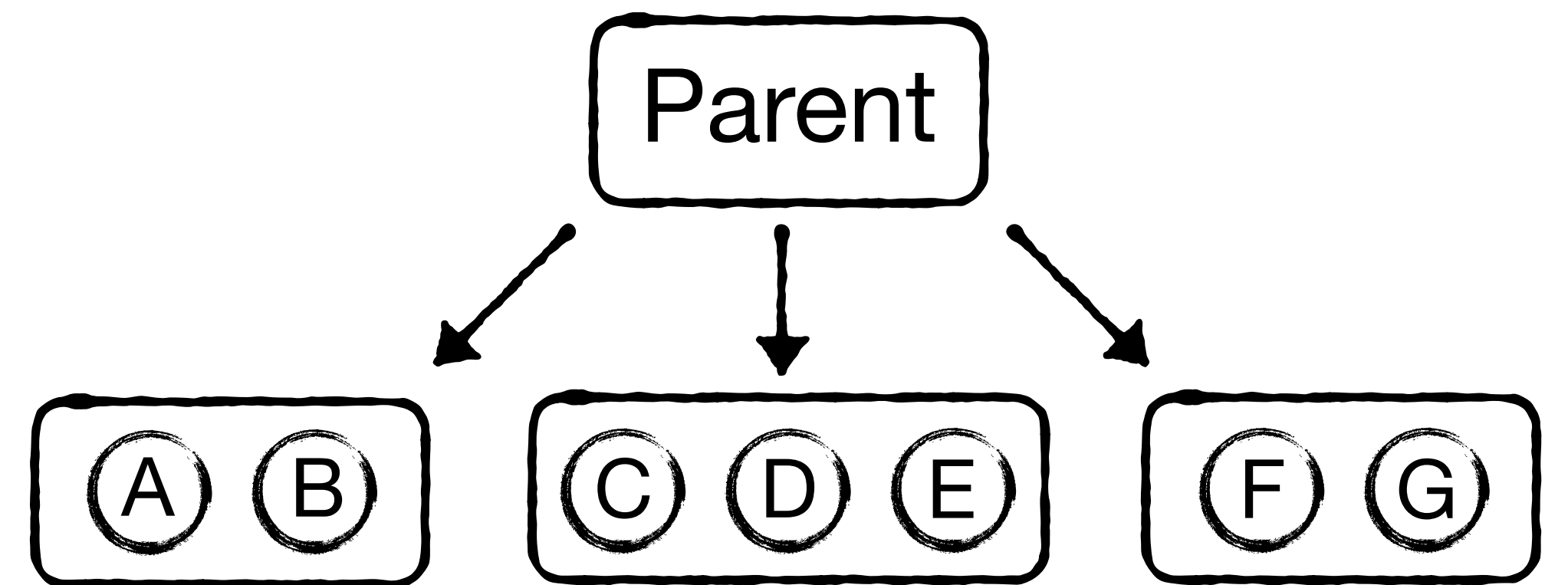
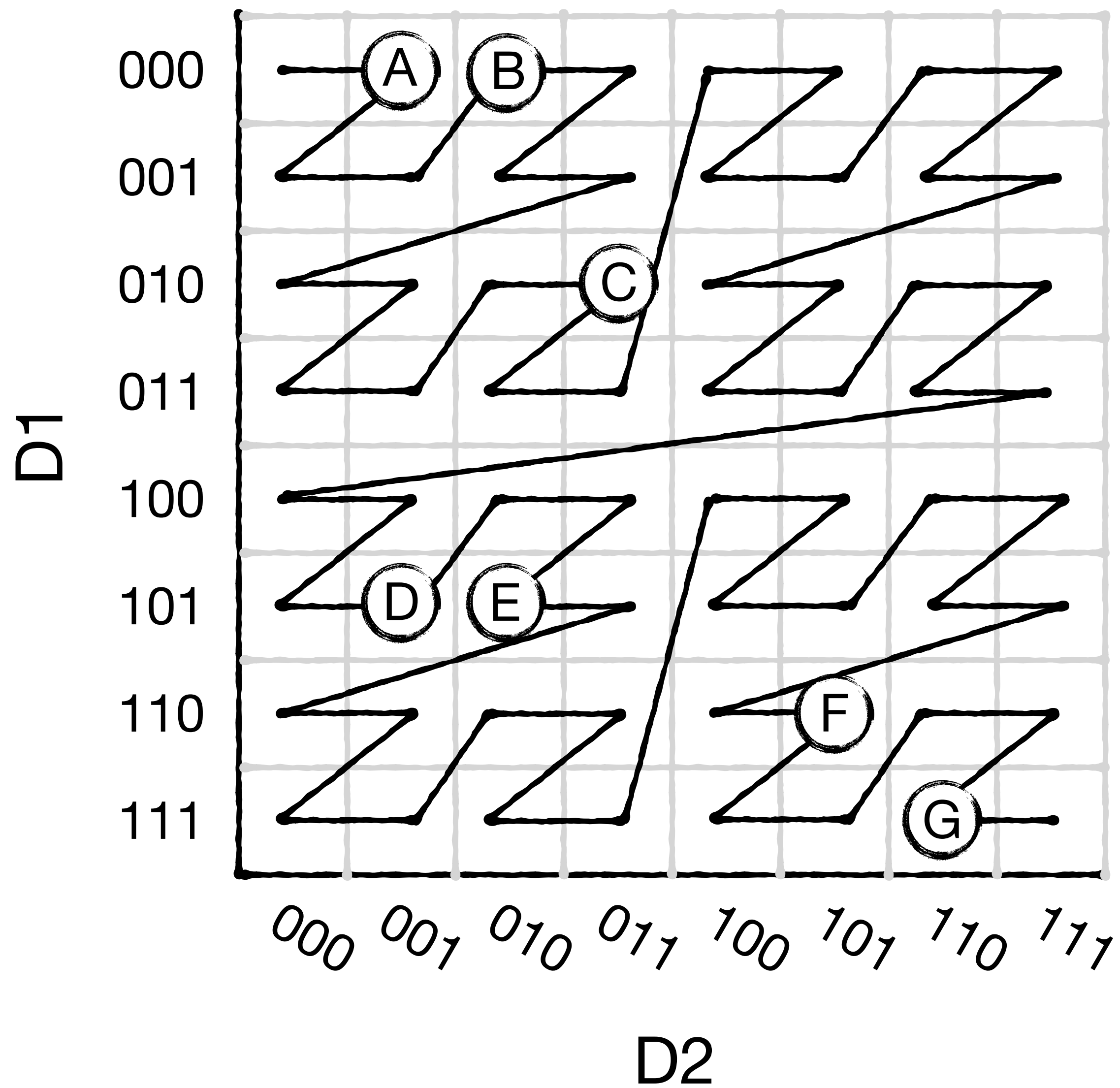
Z-Order Curves



Z-Order Curves

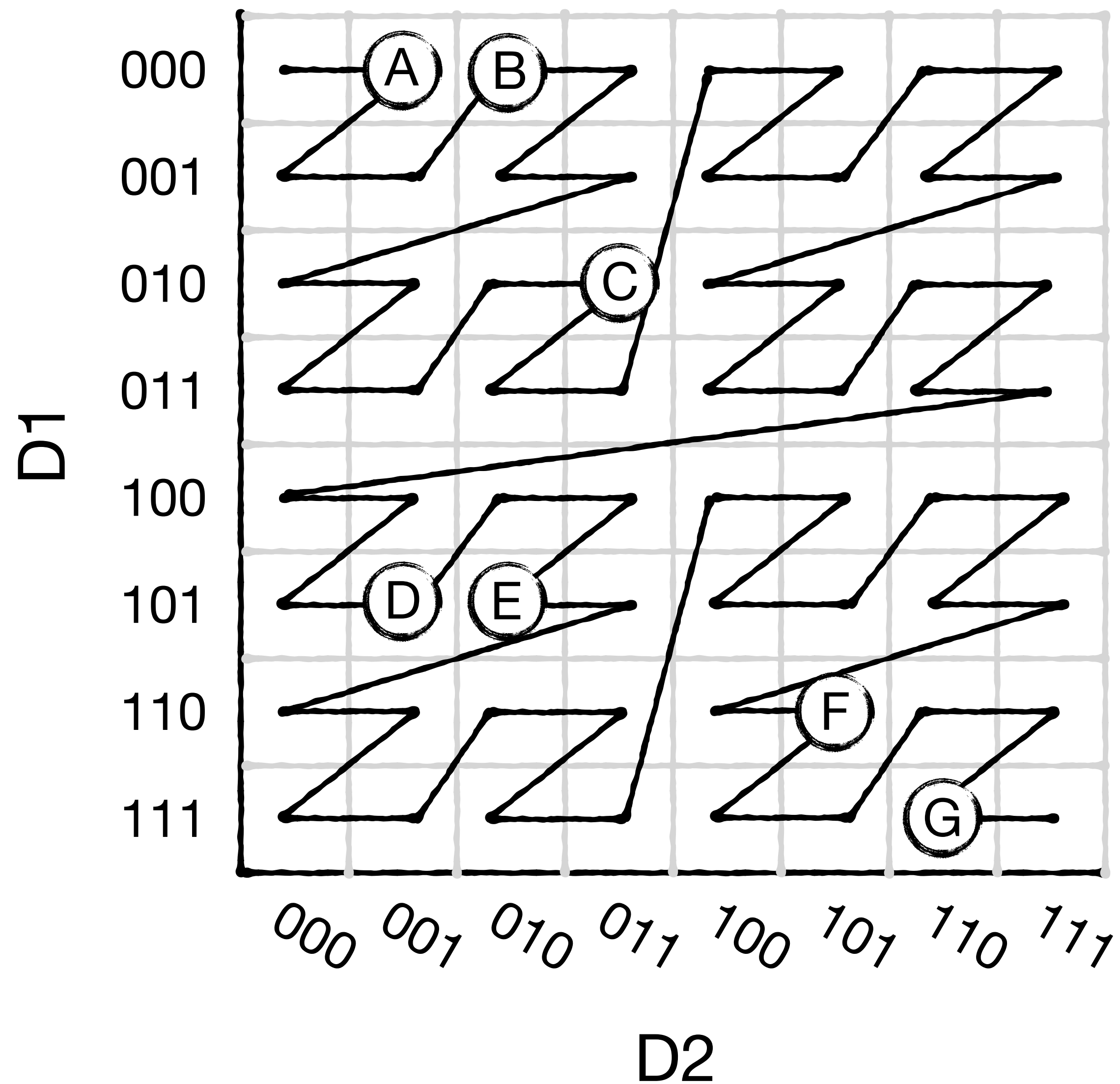






Construct a B-tree

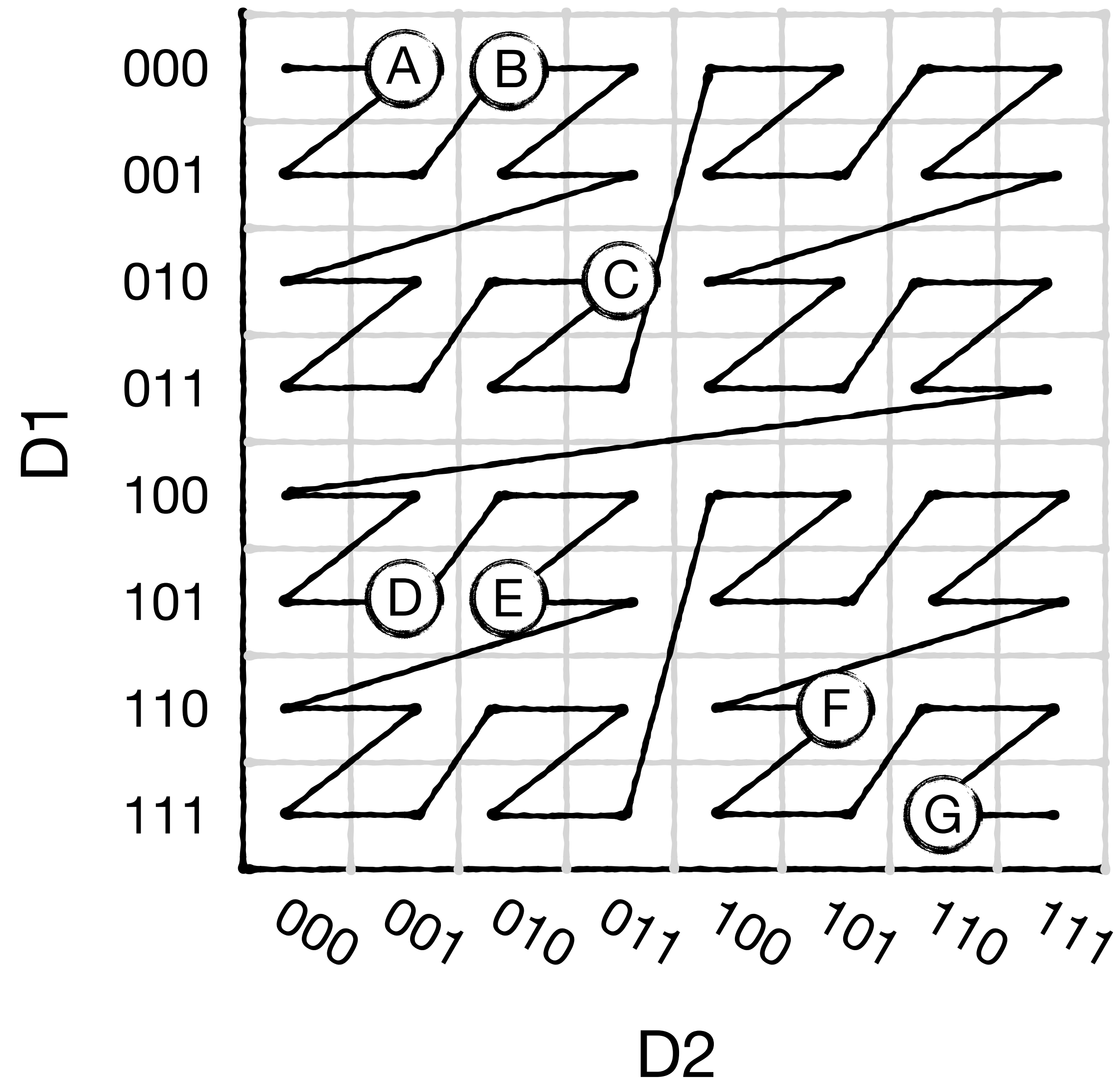
How to compare keys in the Z-Order?



Ⓐ <? Ⓑ

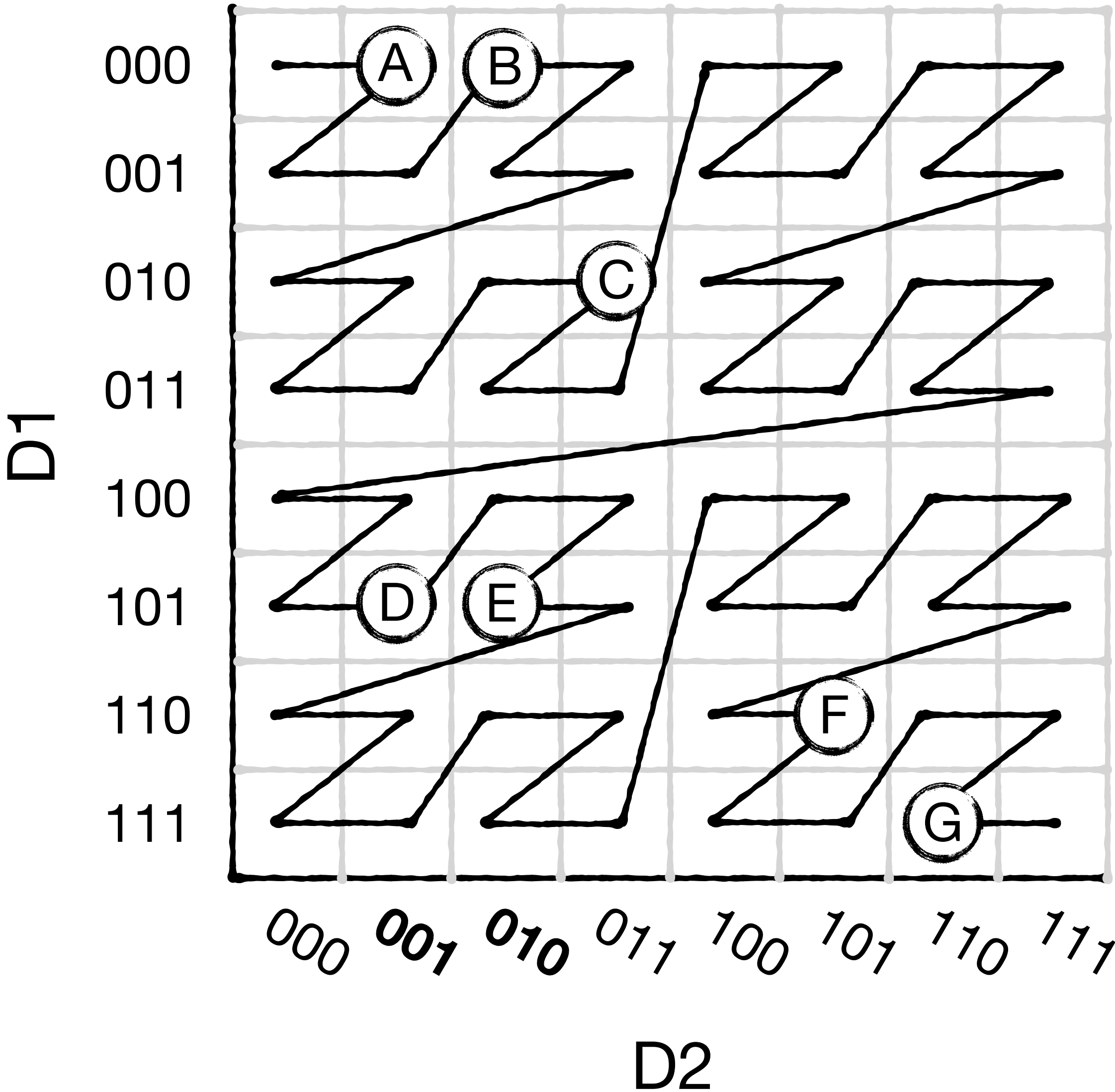
How to compare keys in the Z-Order?

Interleaving dimension bits



Ⓐ <? Ⓑ

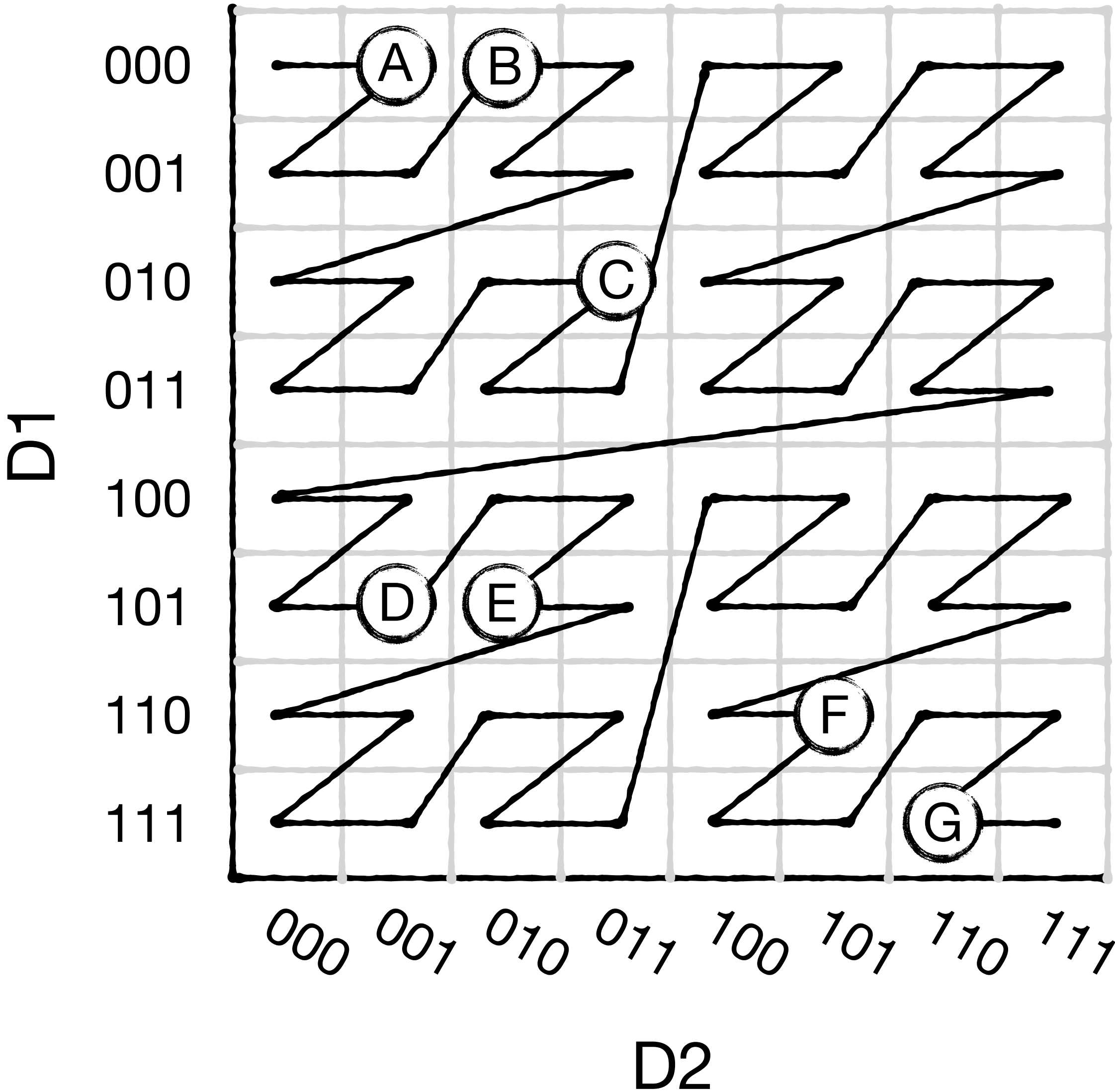
Interleaving dimension bits



D1
D2

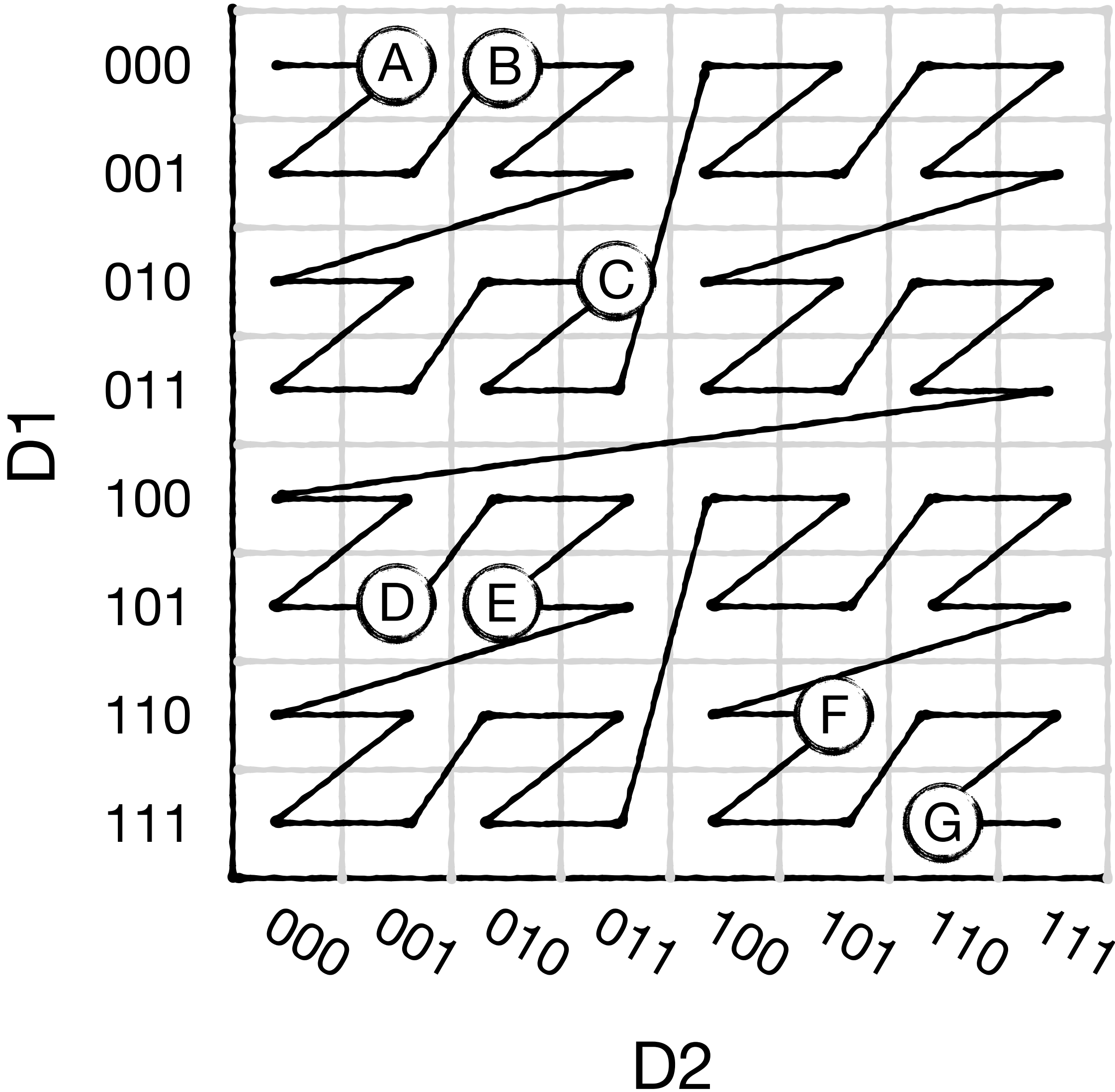
000		000
001		010
(A)	<?	(B)

Interleaving dimension bits



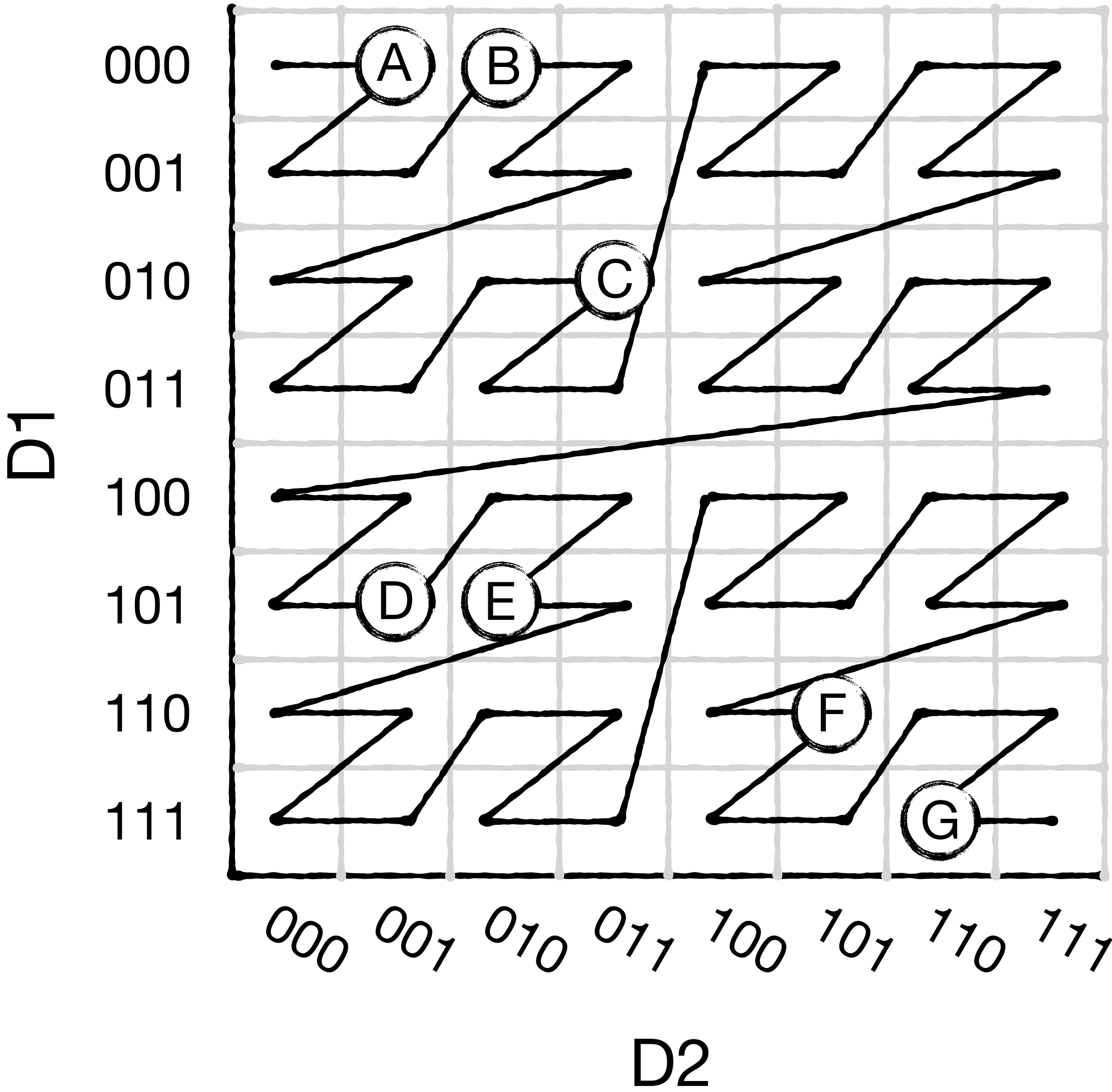
D1	000	000
D2	001	010
Z-Key		
	(A)	(B)
	<?	

Interleaving dimension bits



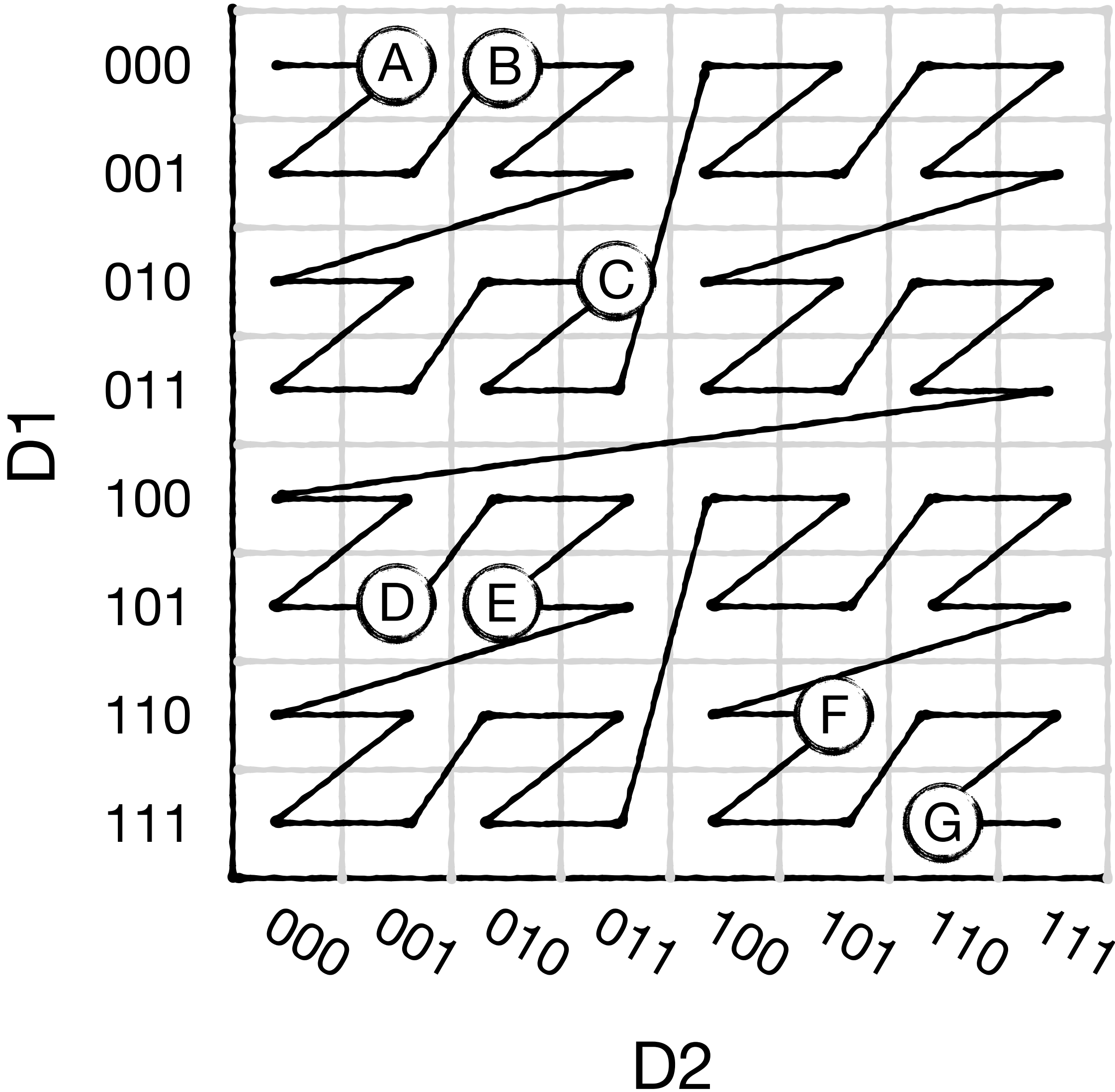
D1	000	000
D2	001	010
Z-Key	00	
	A	<? B

Interleaving dimension bits



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

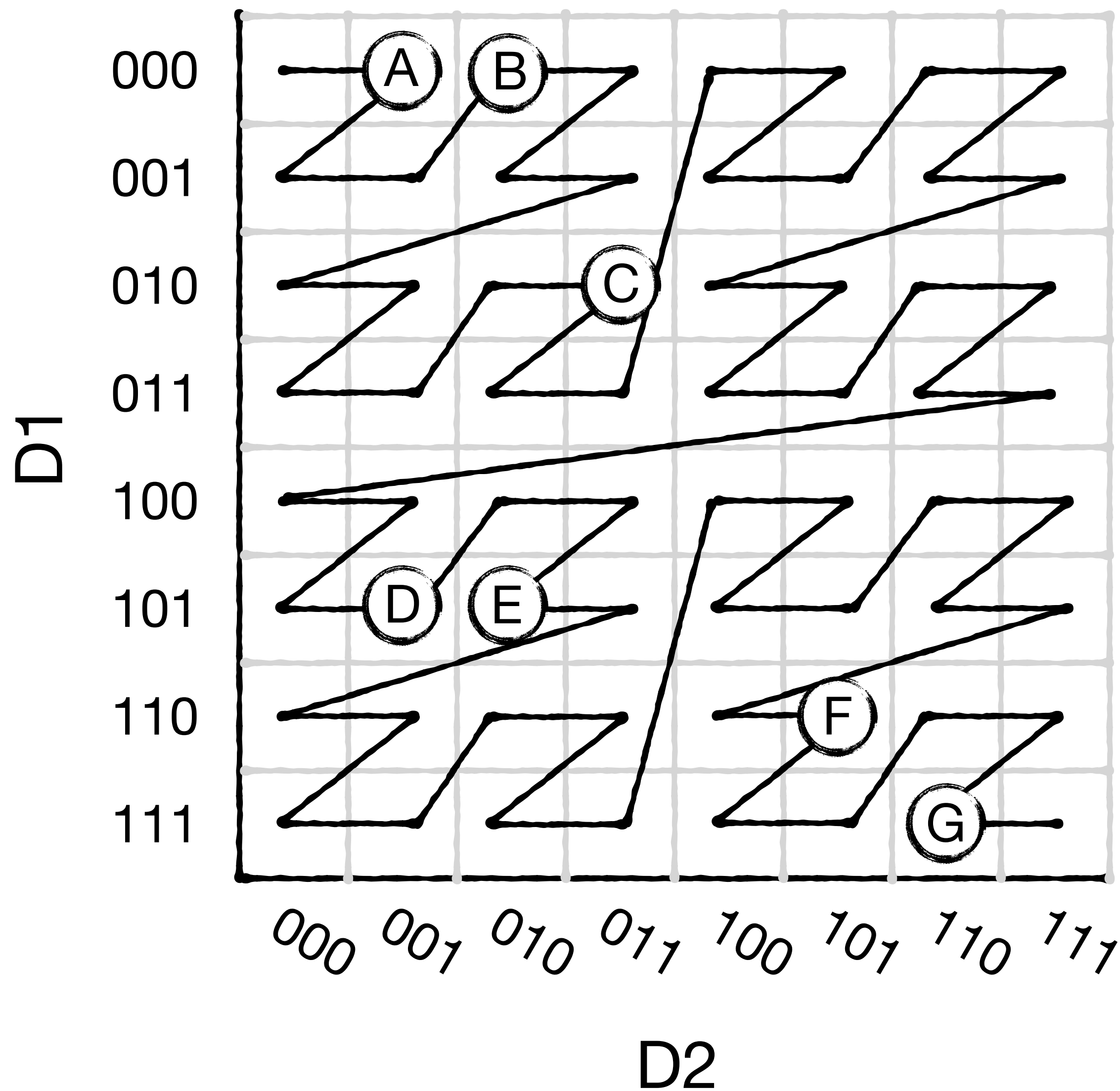
Interleaving dimension bits



D1 000
D2 001
Z-Key 00 00 01

(A) <? (B)

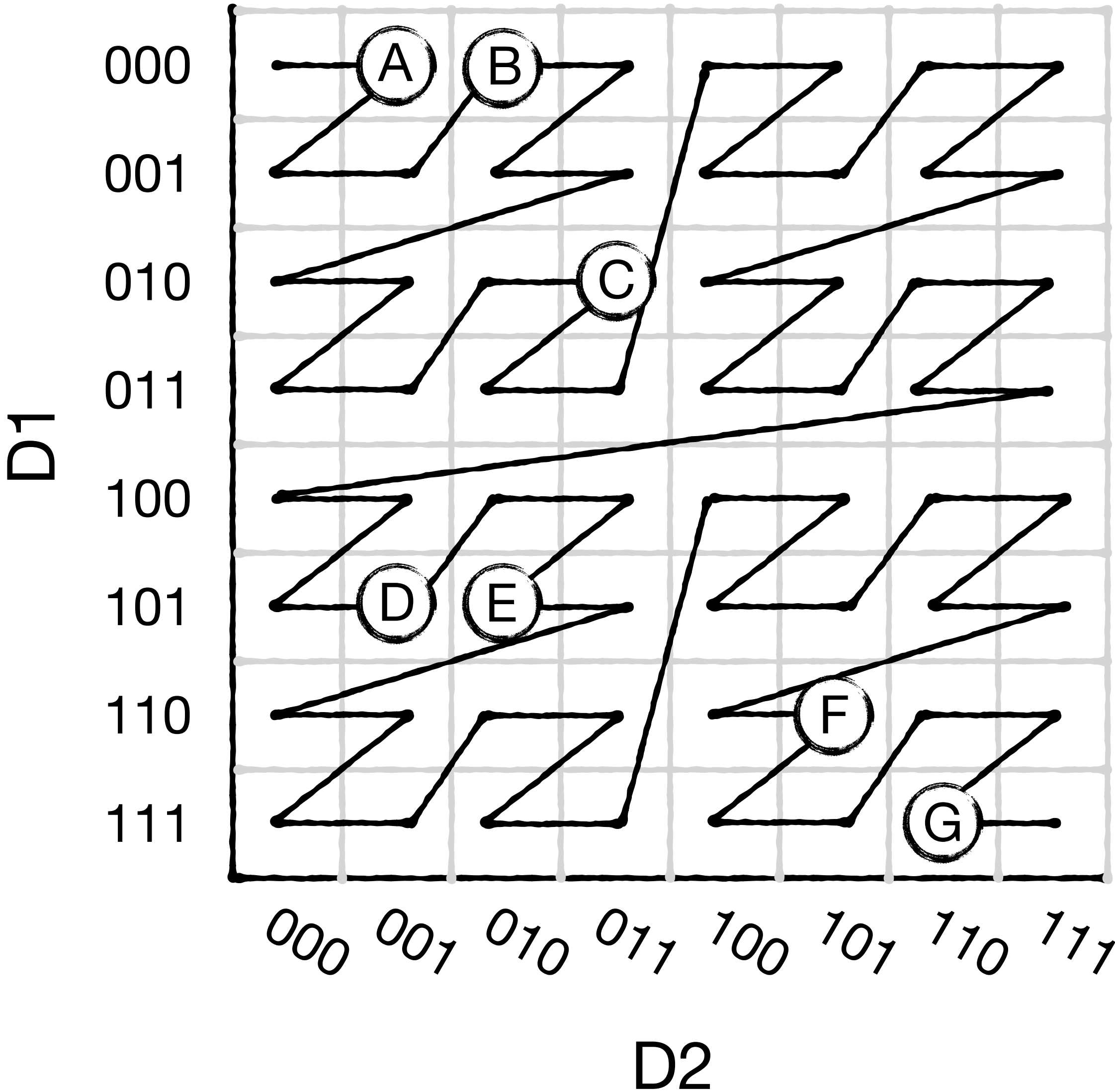
Interleaving dimension bits



D1 000
D2 001
Z-Key 00 00 01

(A) <? (B)

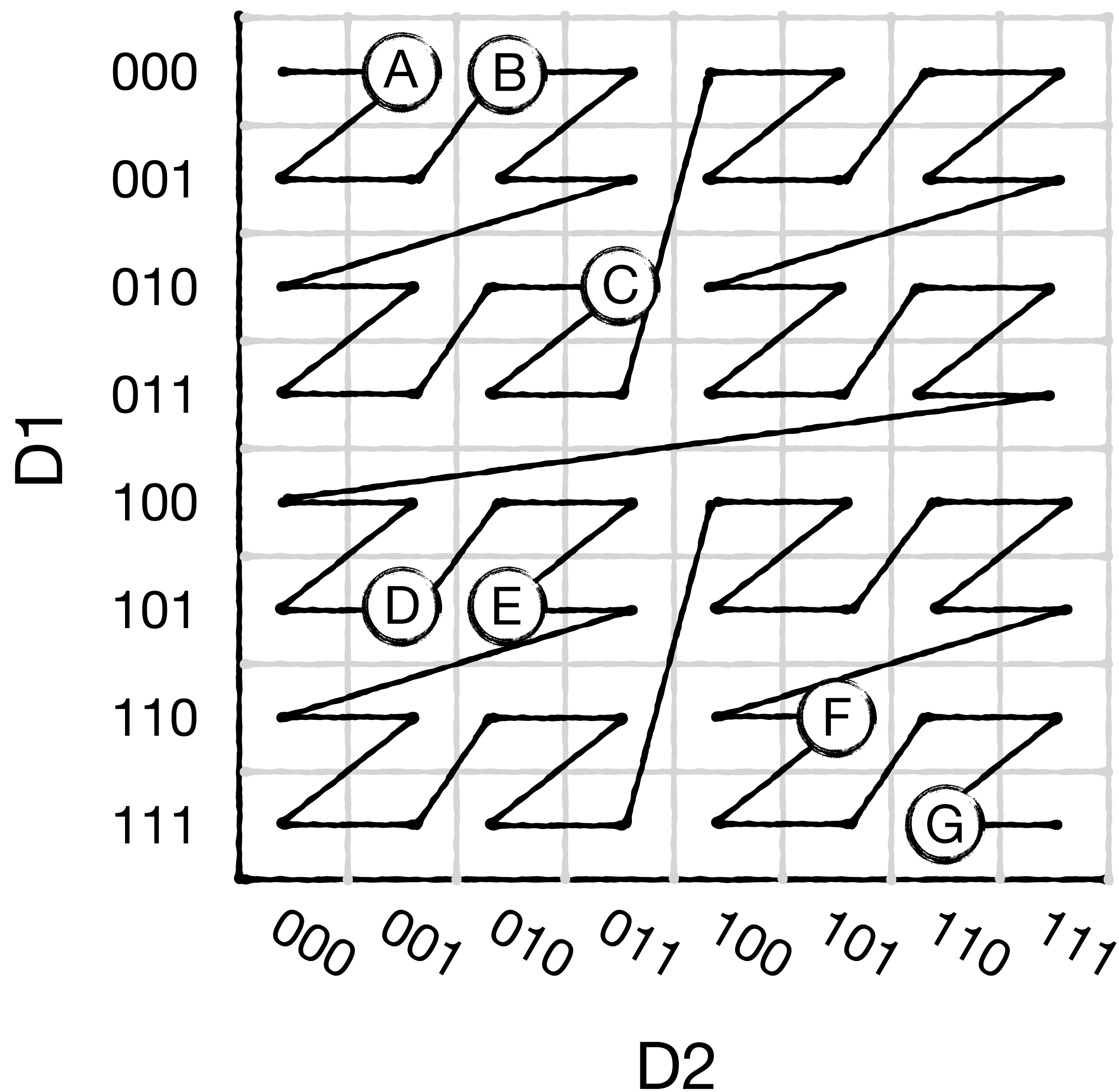
Interleaving dimension bits



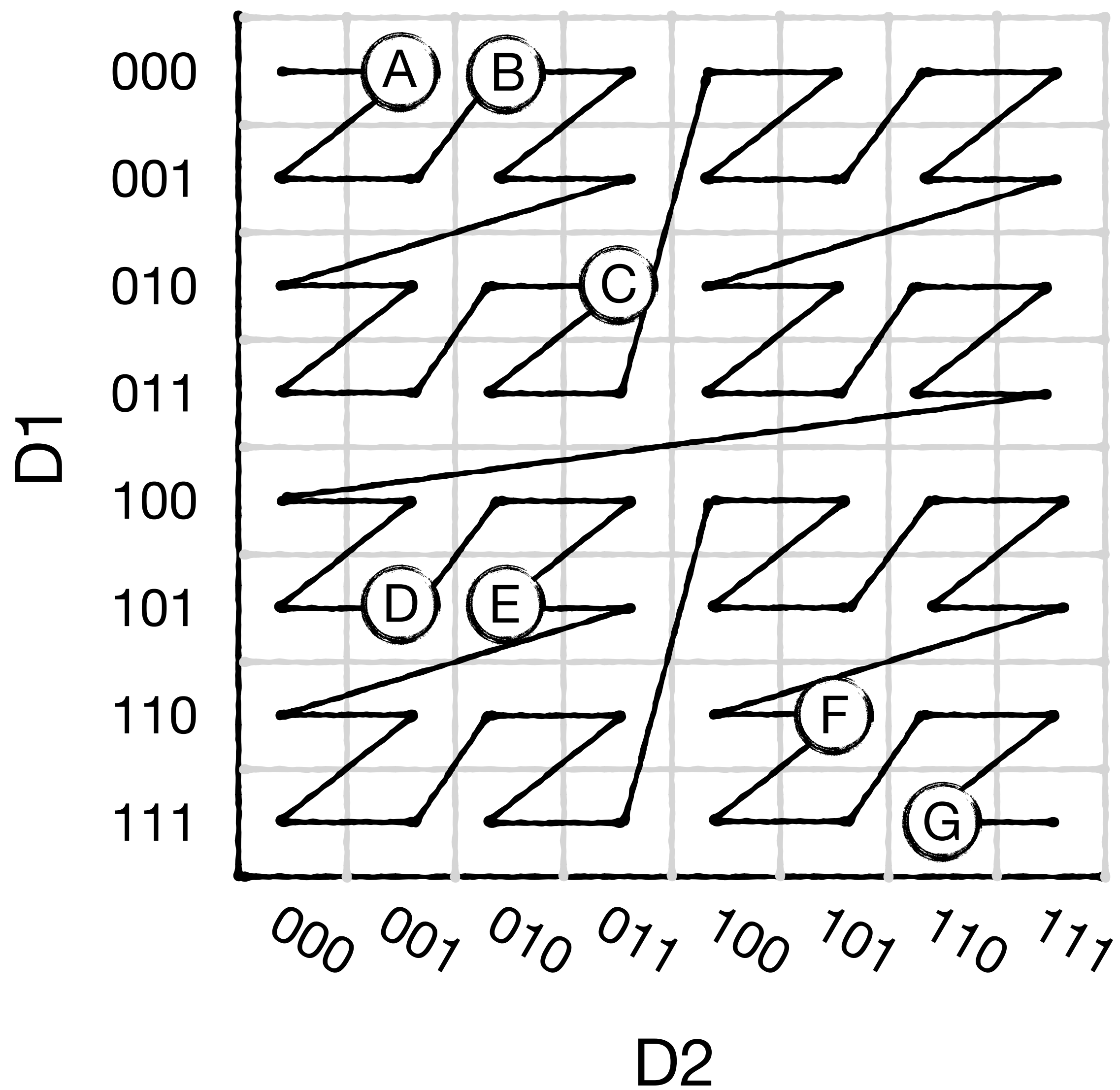
D1	000	000
D2	001	010
Z-Key	00 00 01	00 01
	A	B

<?

Interleaving dimension bits



| | | |
|-------|----------|----------|
| D1 | 000 | 000 |
| D2 | 001 | 010 |
| Z-Key | 00 00 01 | 00 01 00 |
| | (A) | (B) |
| | <? | |



Z-Key

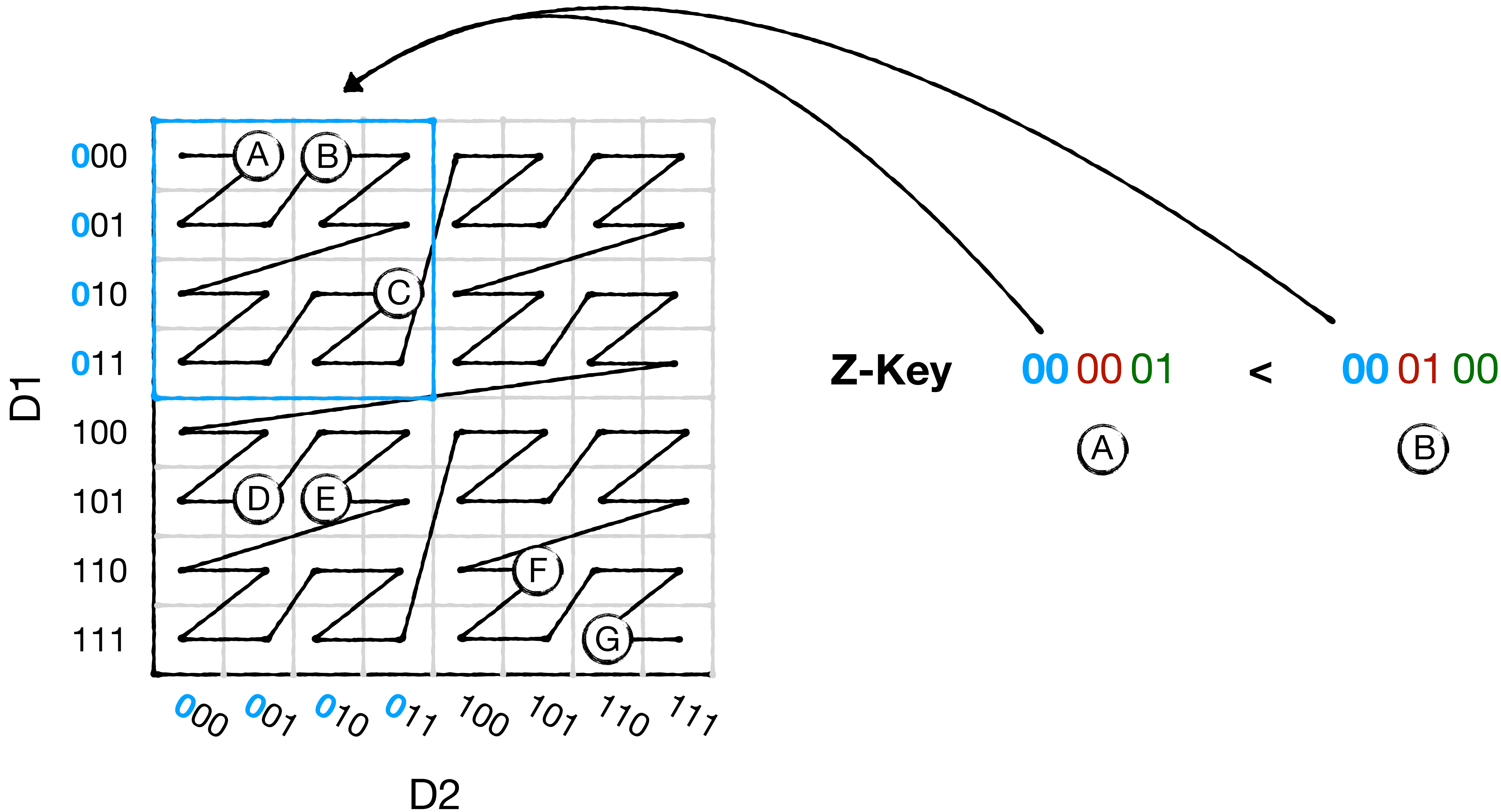
00 00 01

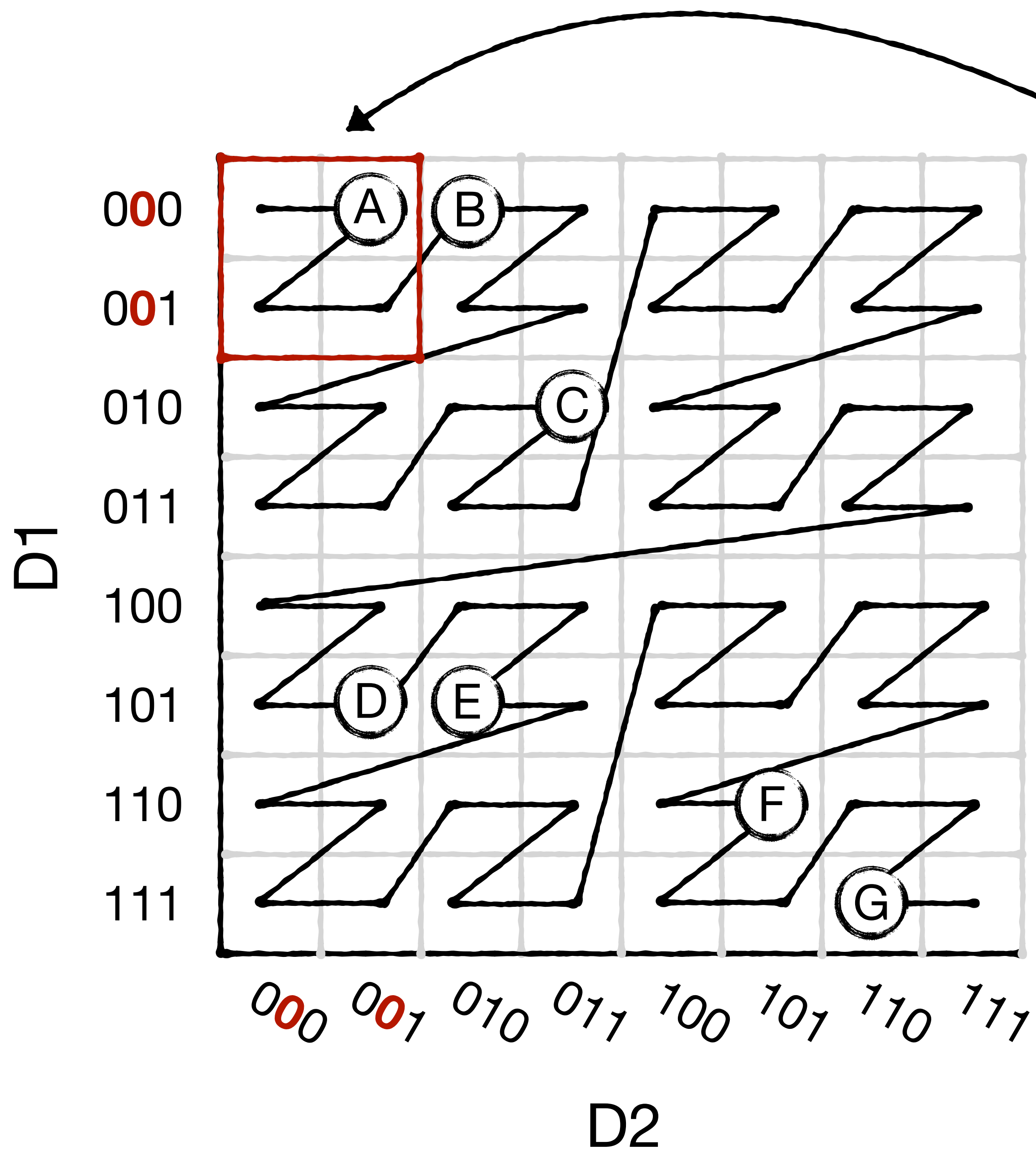
<

00 01 00

A

B





Z-Key

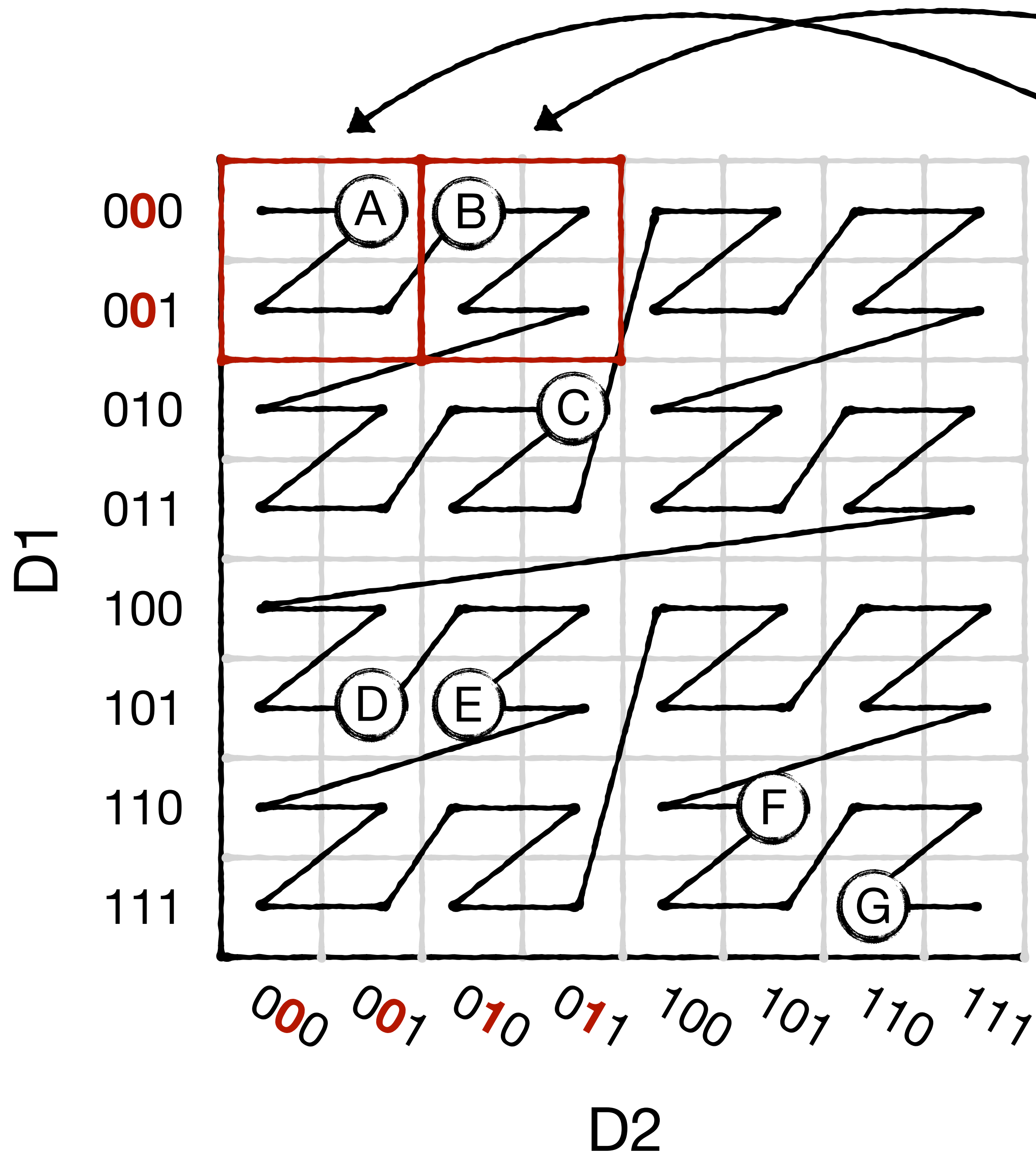
$00\ 00\ 01$

$<$

$00\ 01\ 00$

(A)

(B)



Z-Key

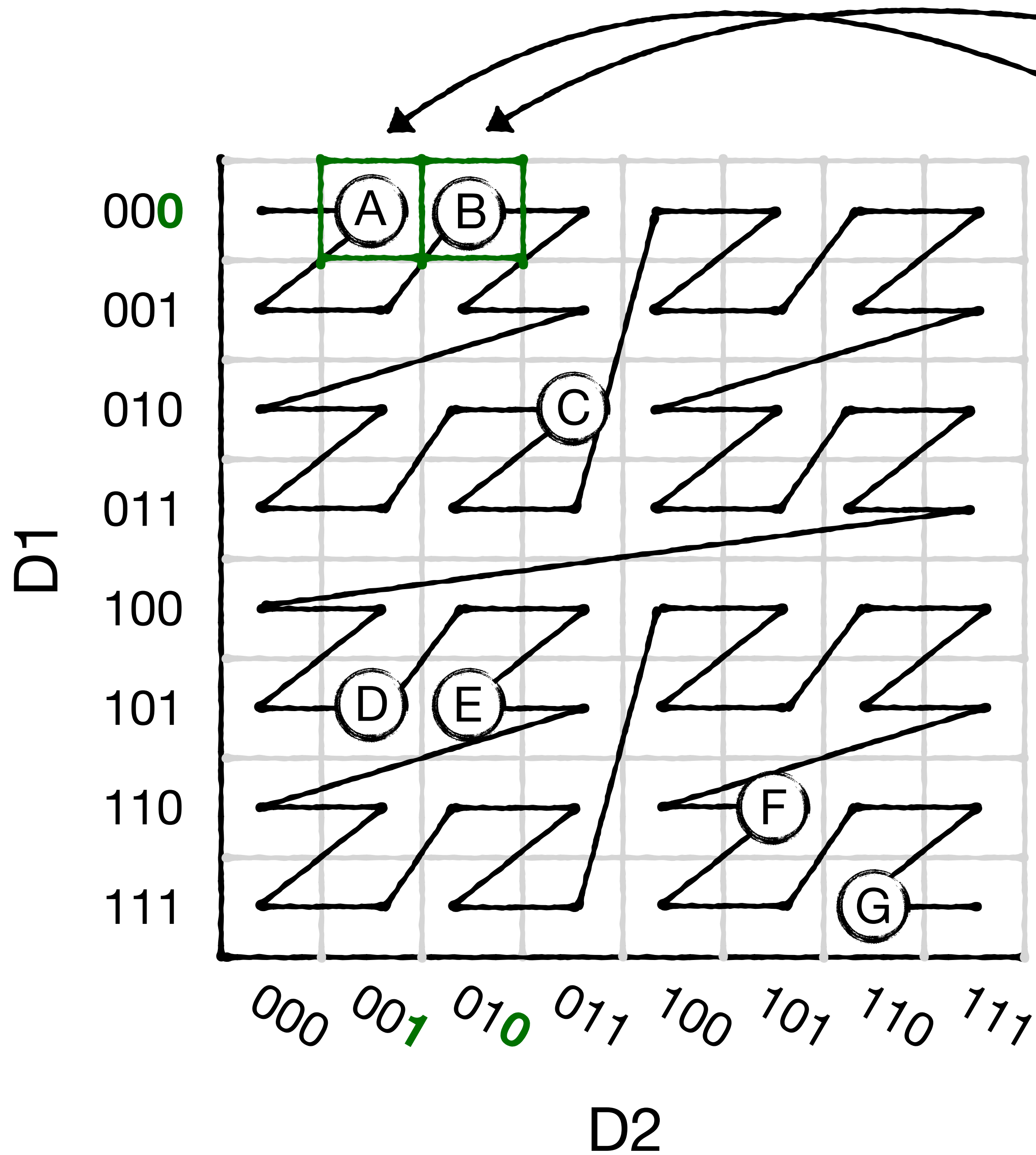
00 00 01

<

00 01 00

(A)

(B)



Z-Key

00 00 01

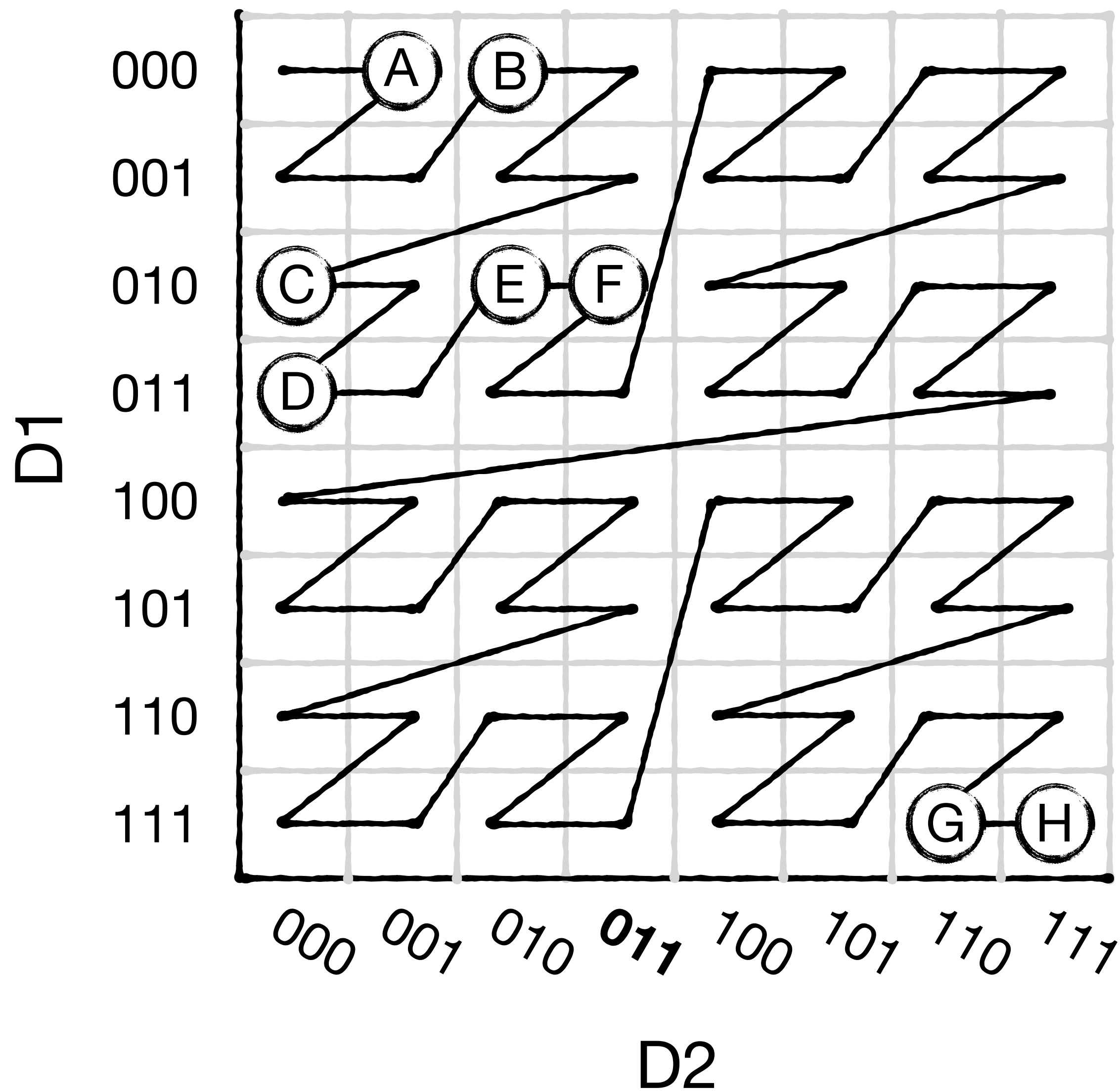
<

00 01 00

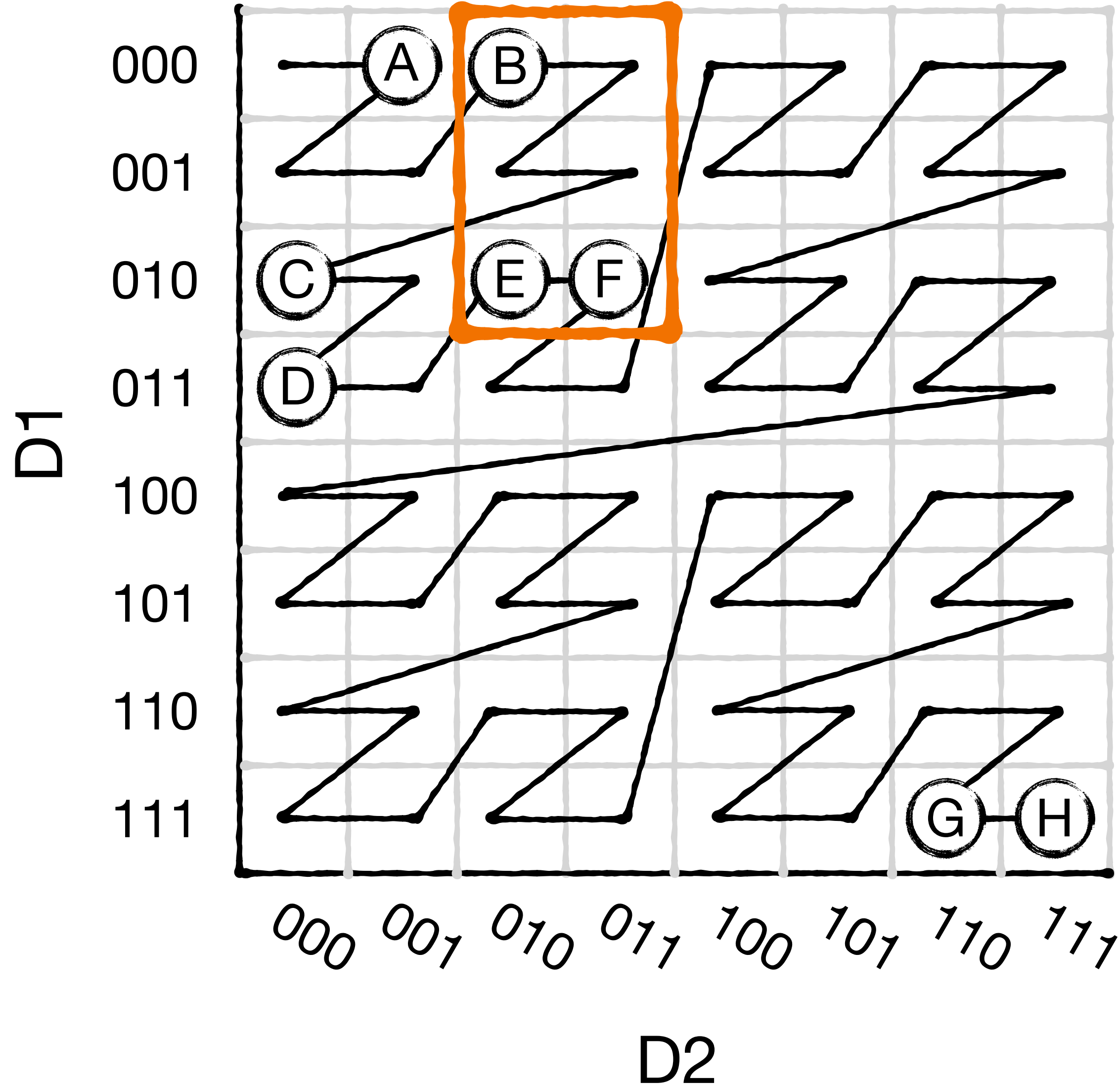
A

B

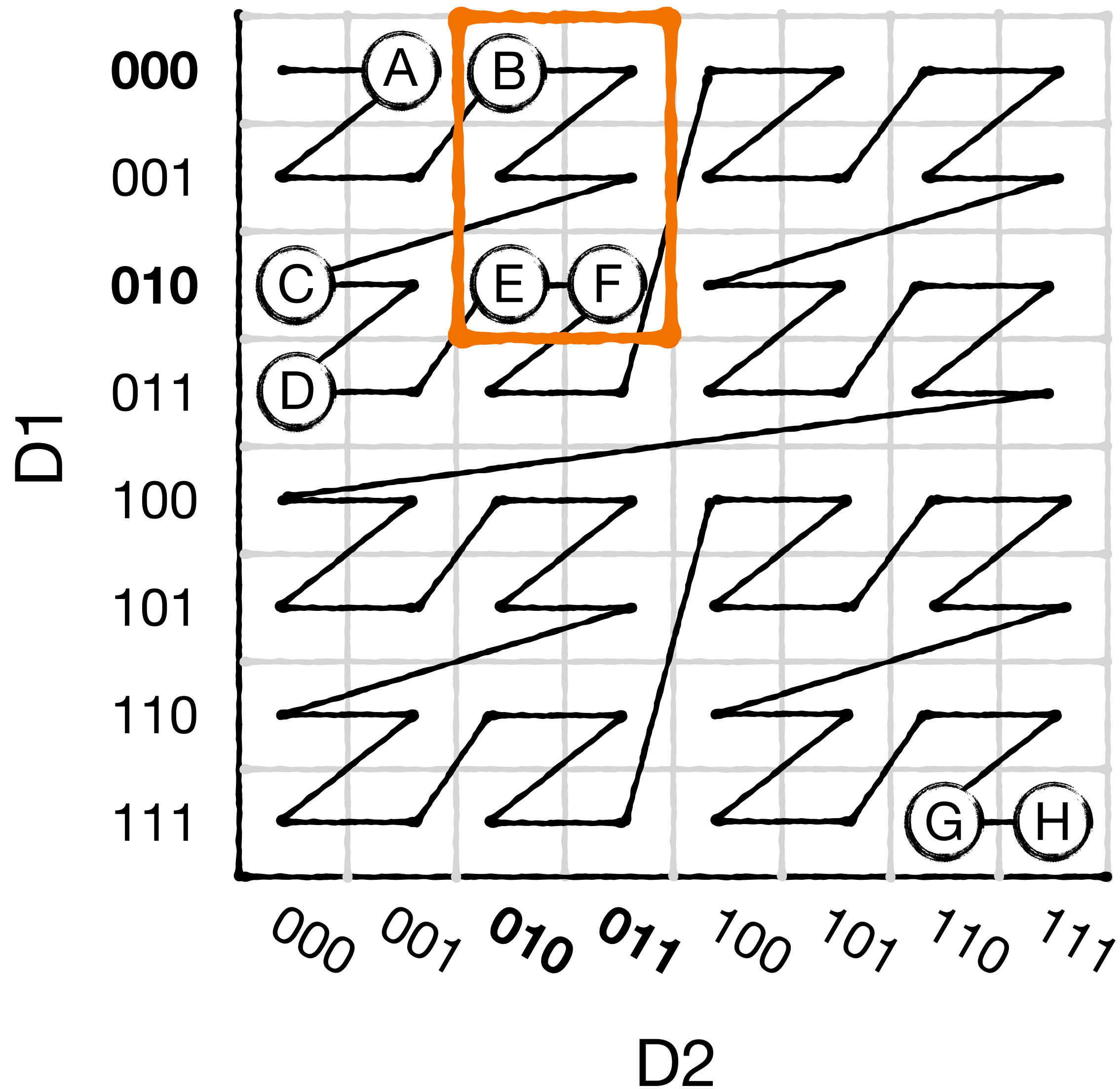
Query Handling



Query Handling



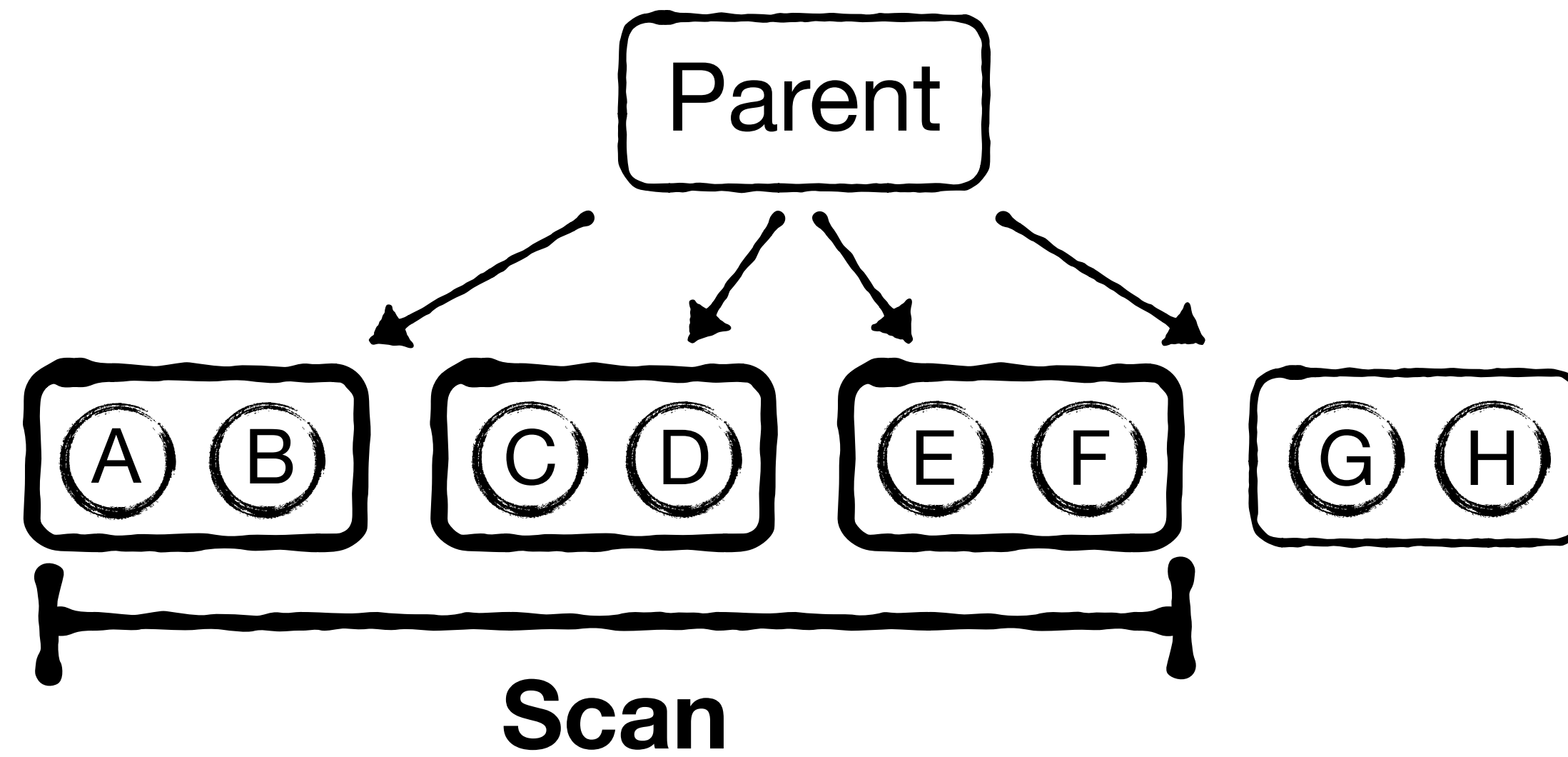
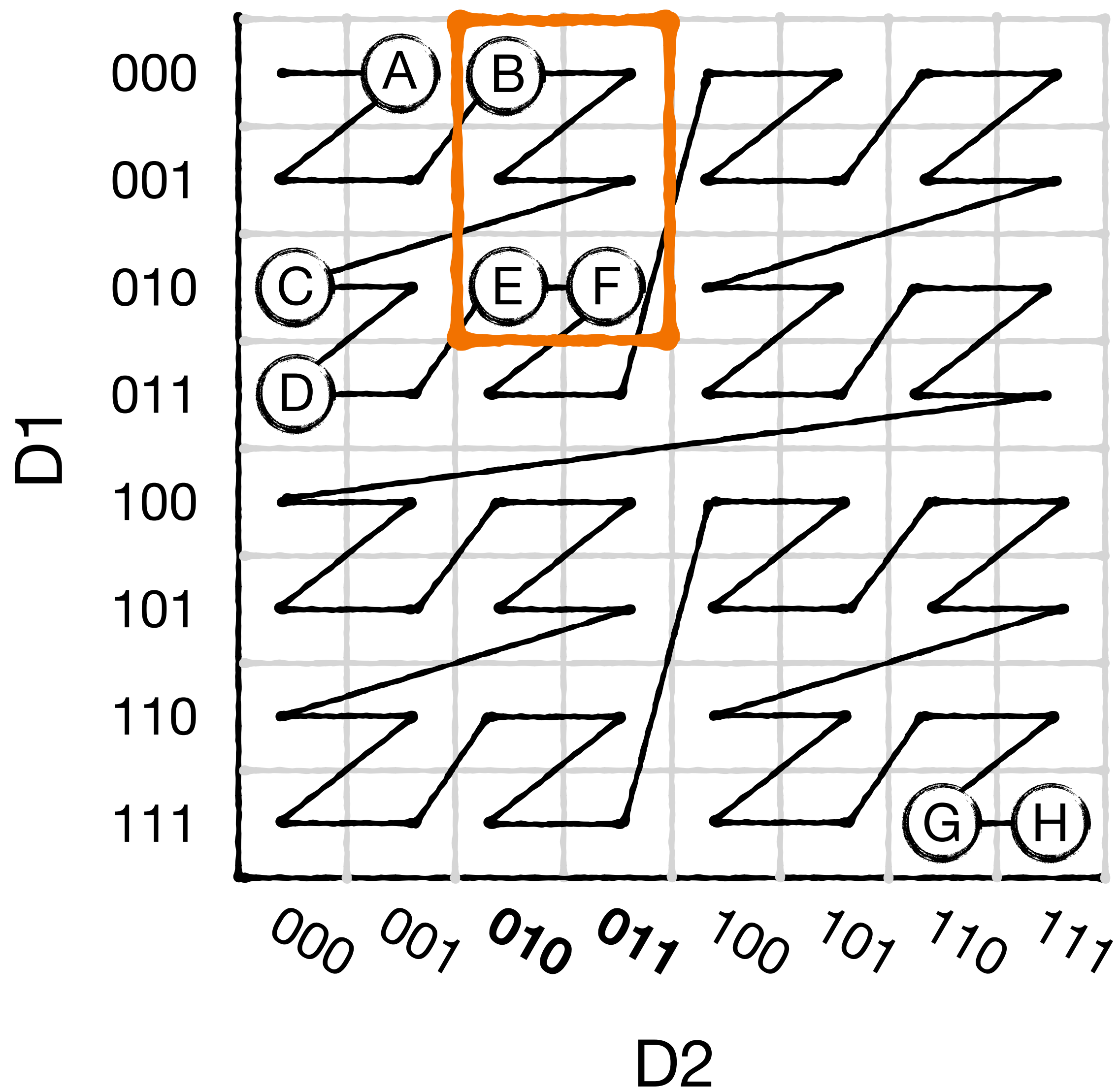
Query is a rectangle defined by start and end Z-key



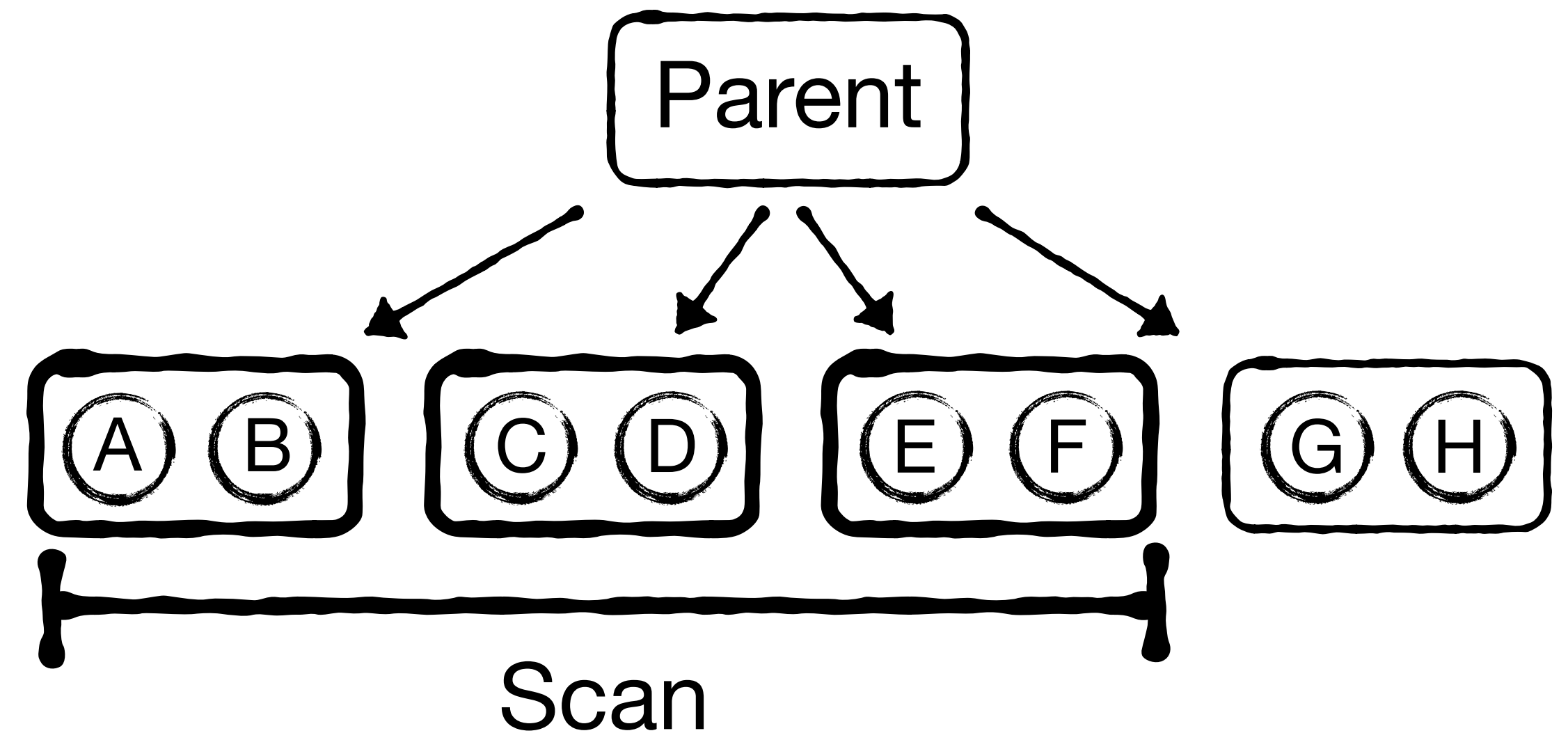
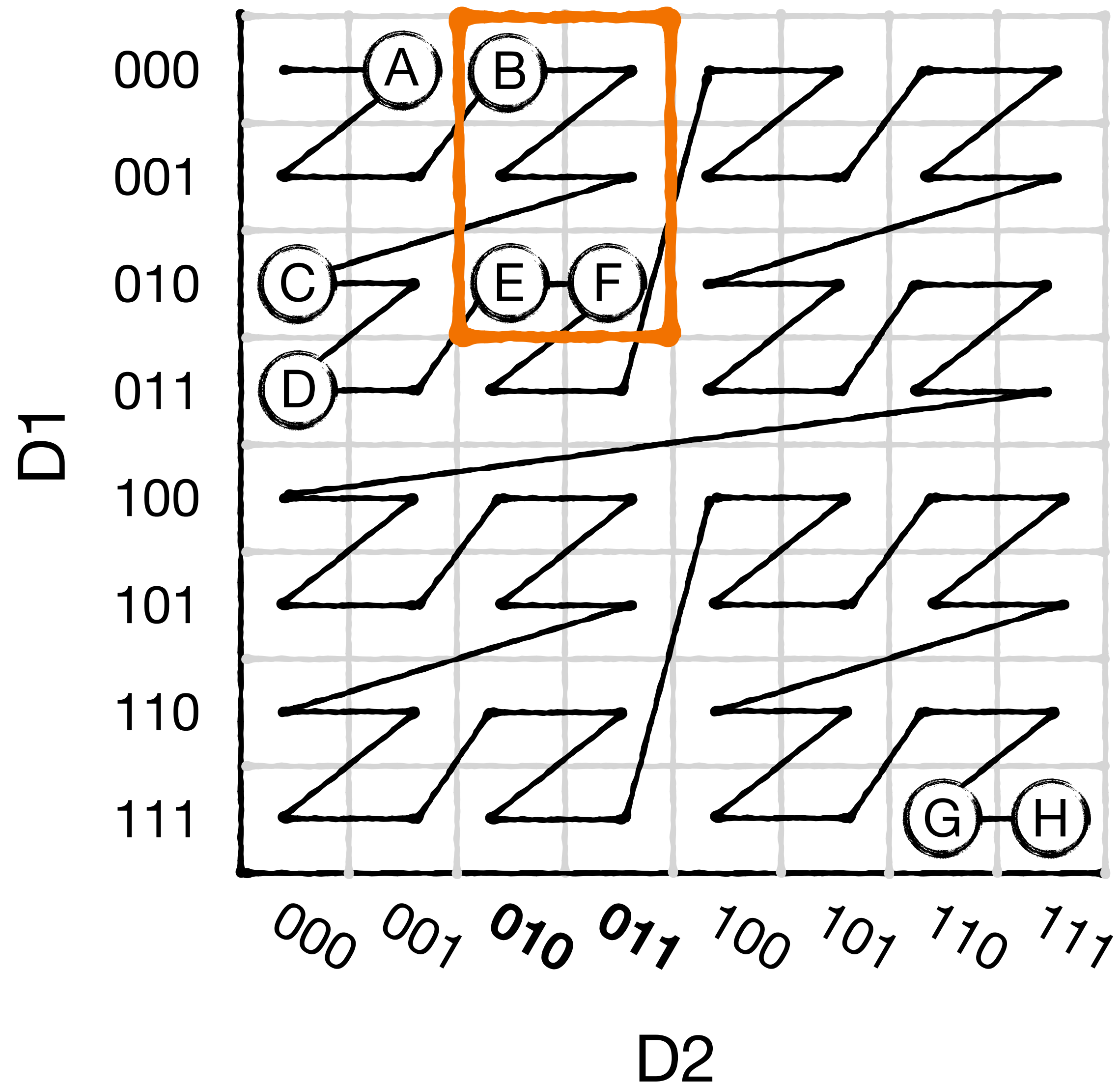
Example

start: 00 01 00

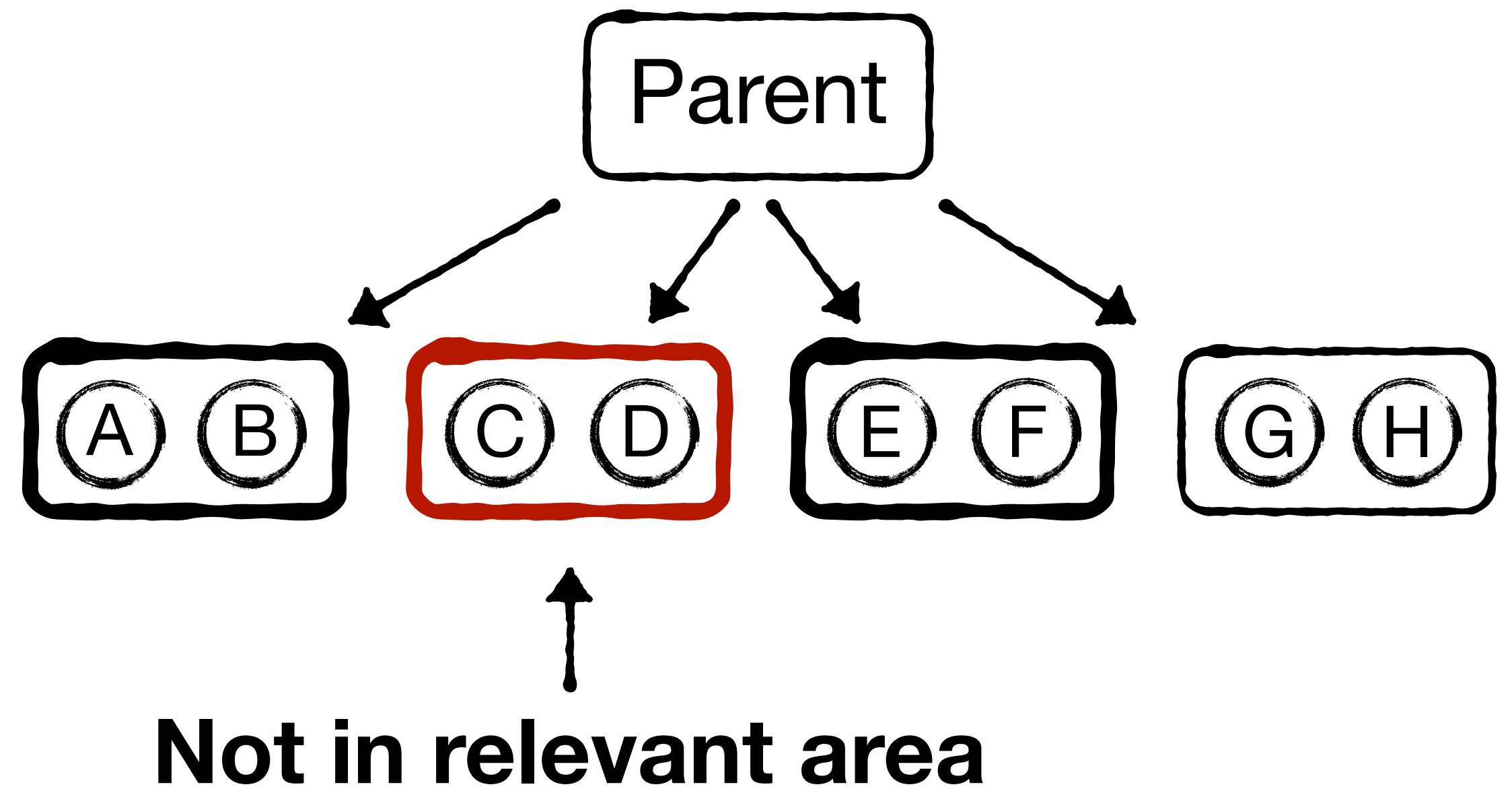
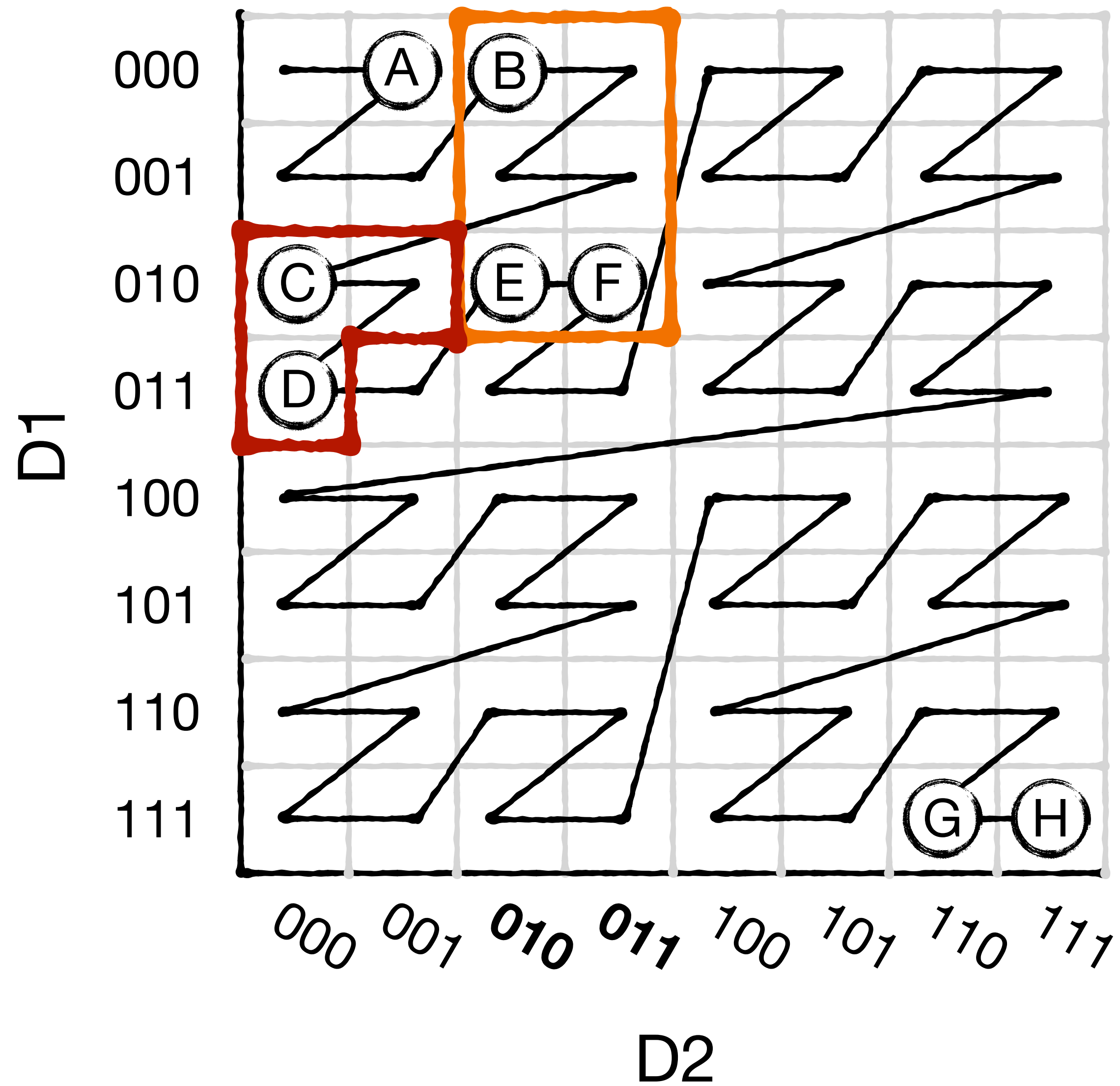
end: 00 11 01



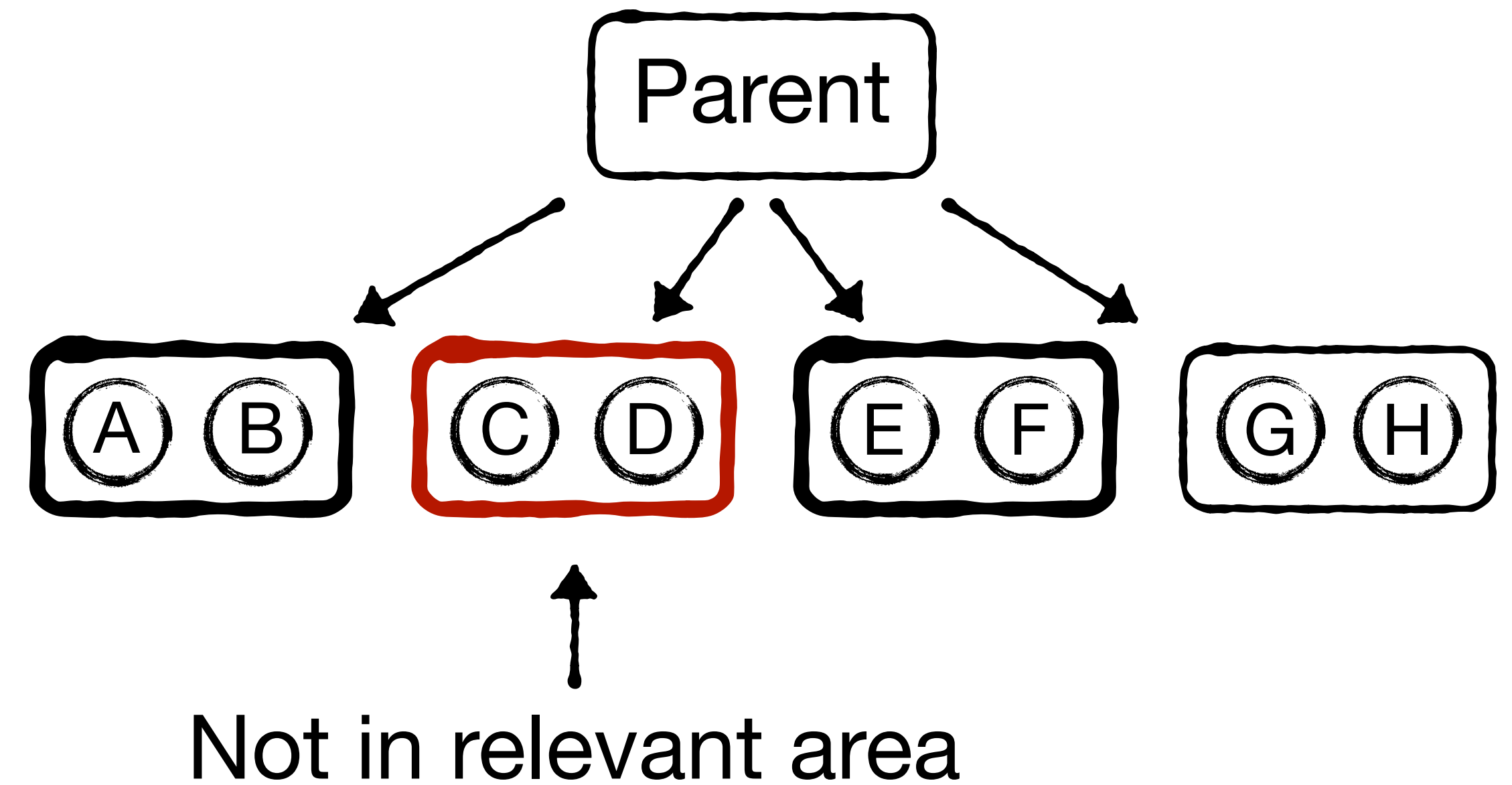
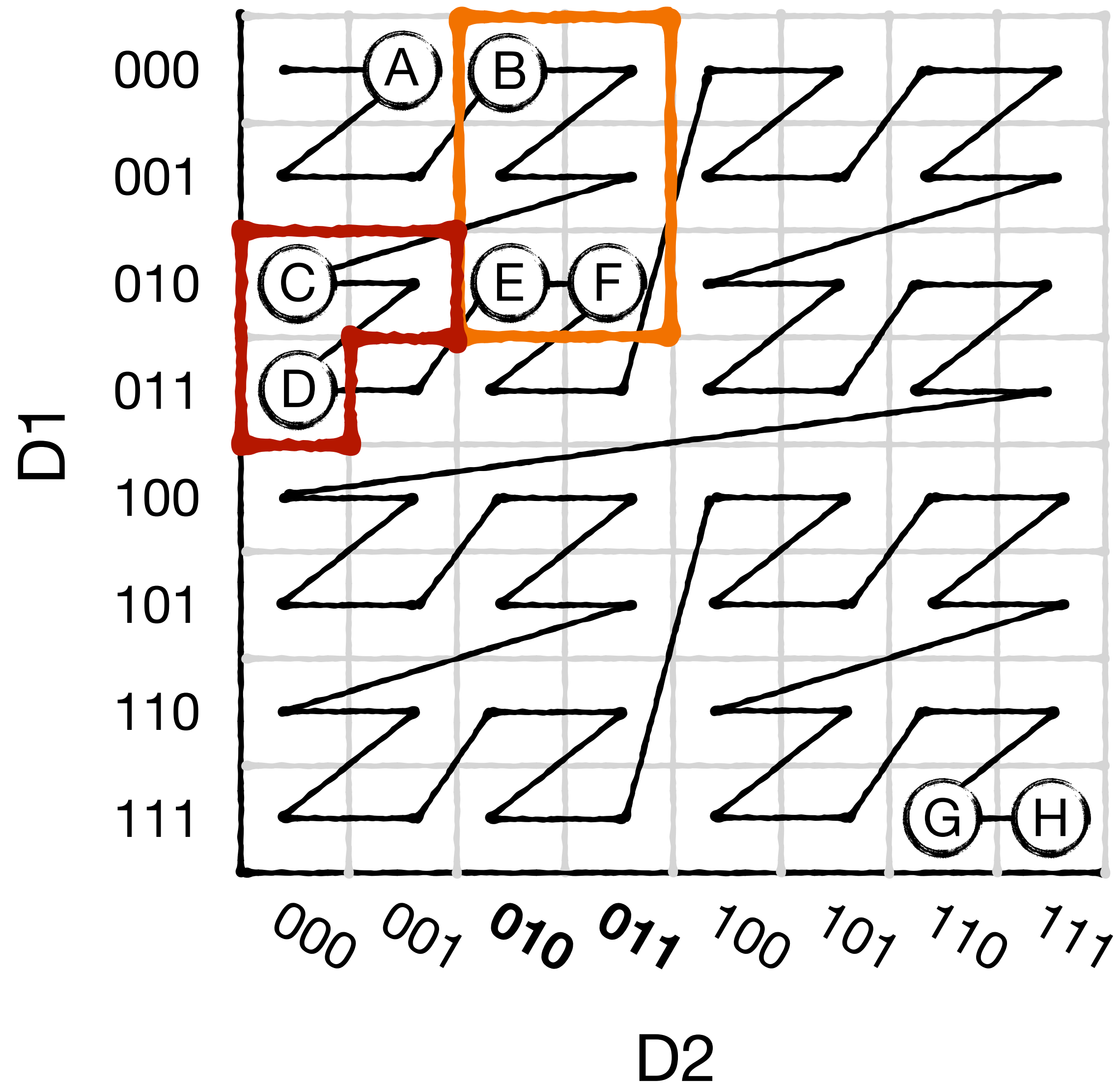
Issues?



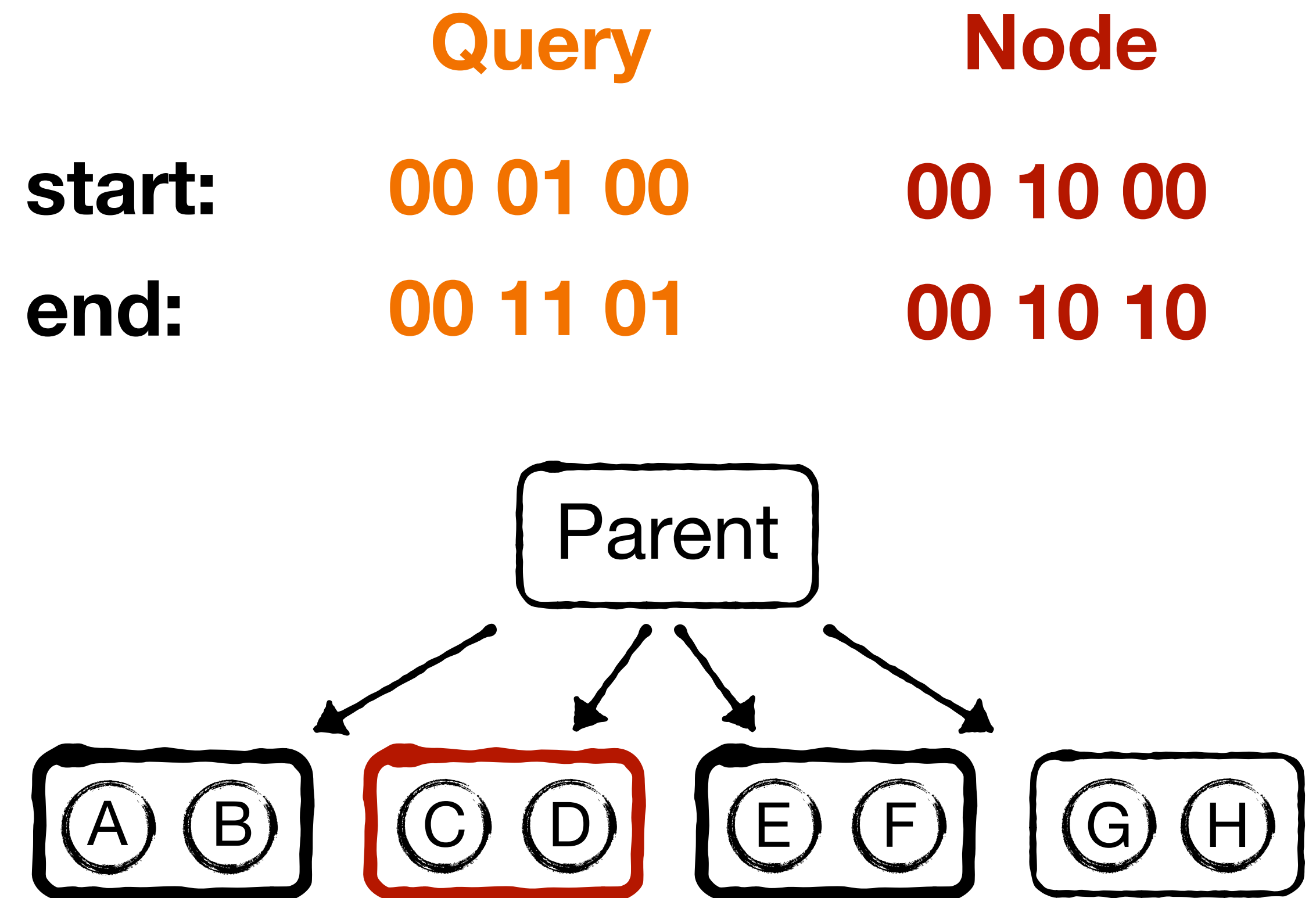
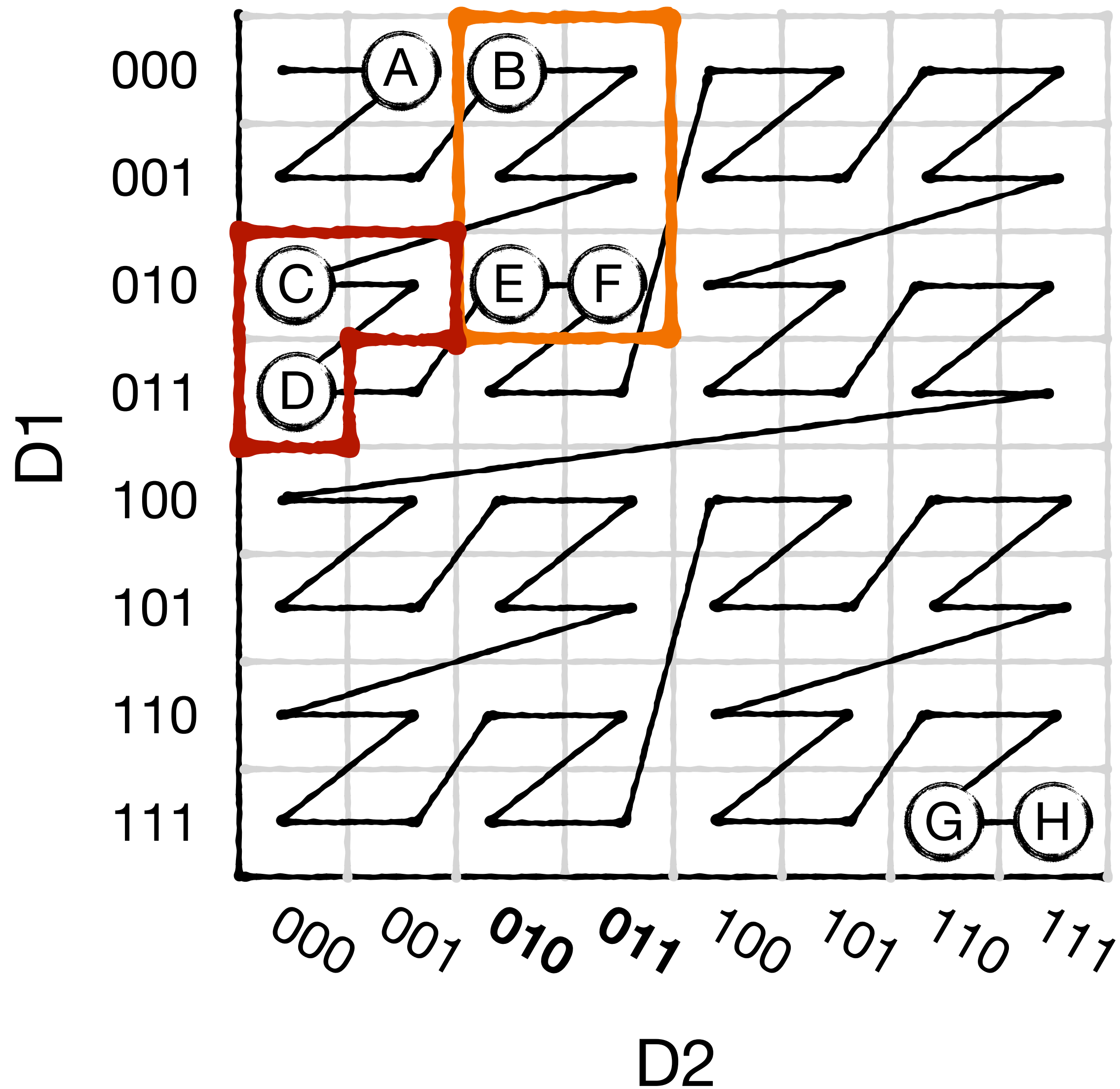
Issues?



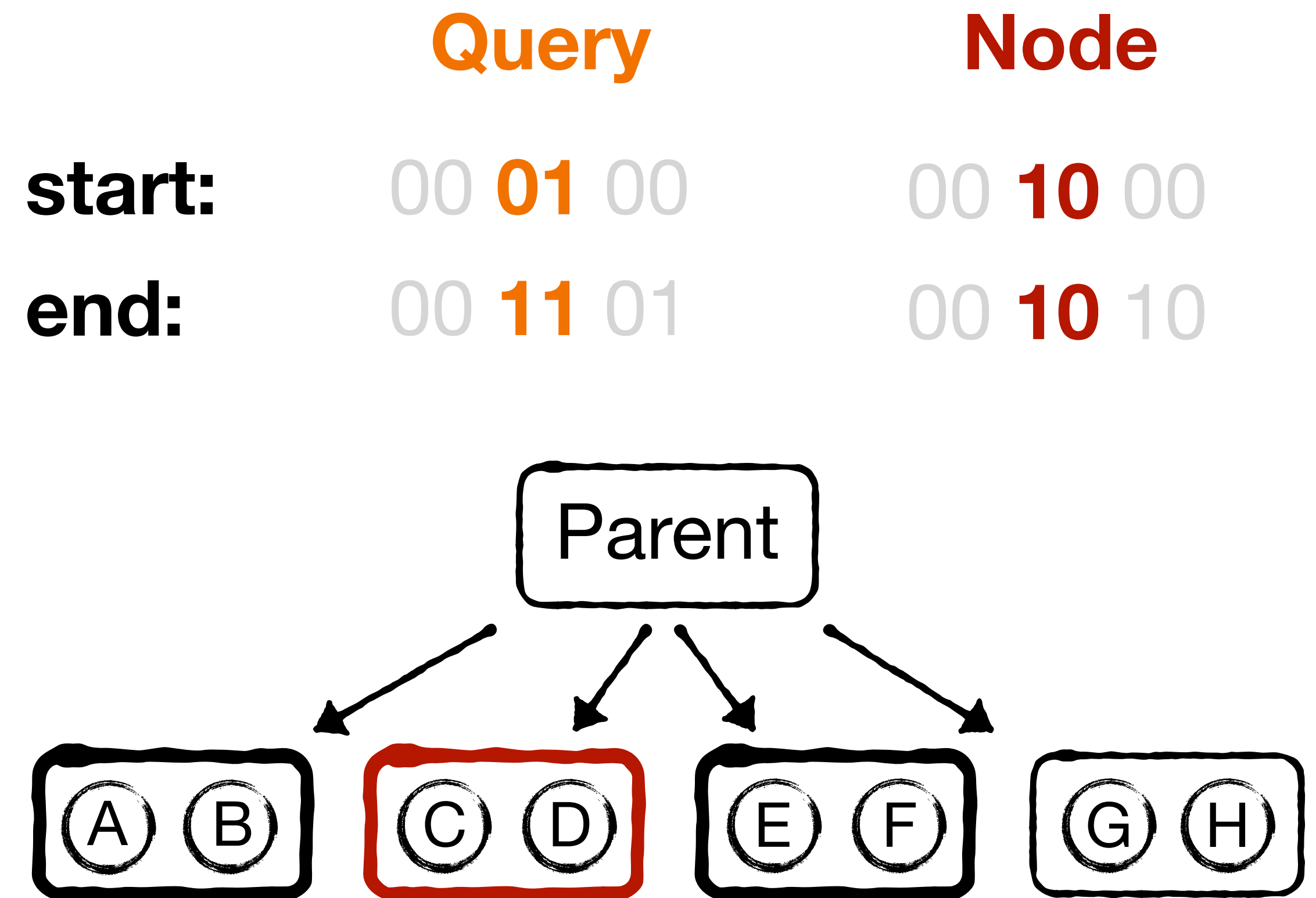
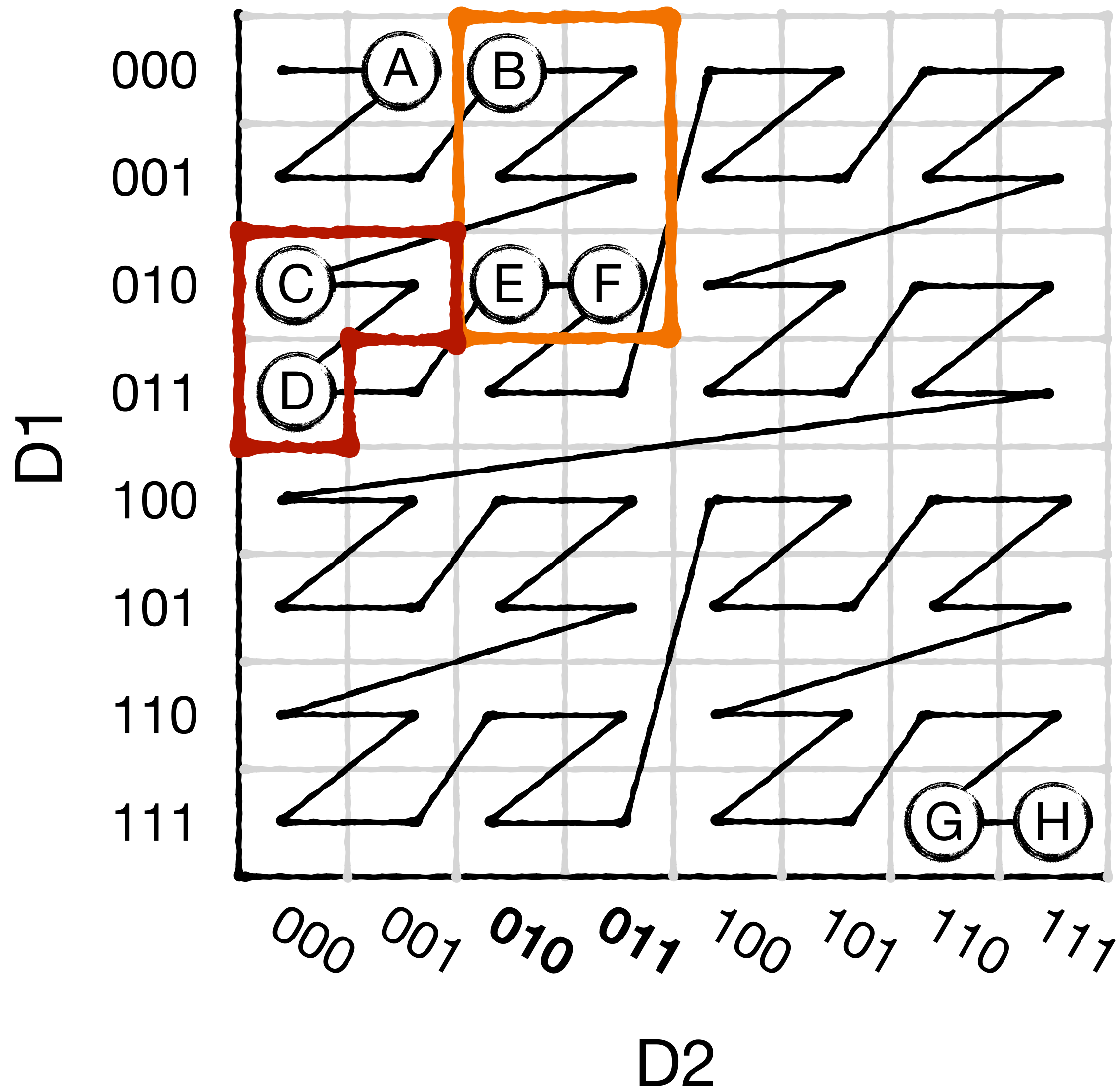
How to skip such nodes to speed up search?



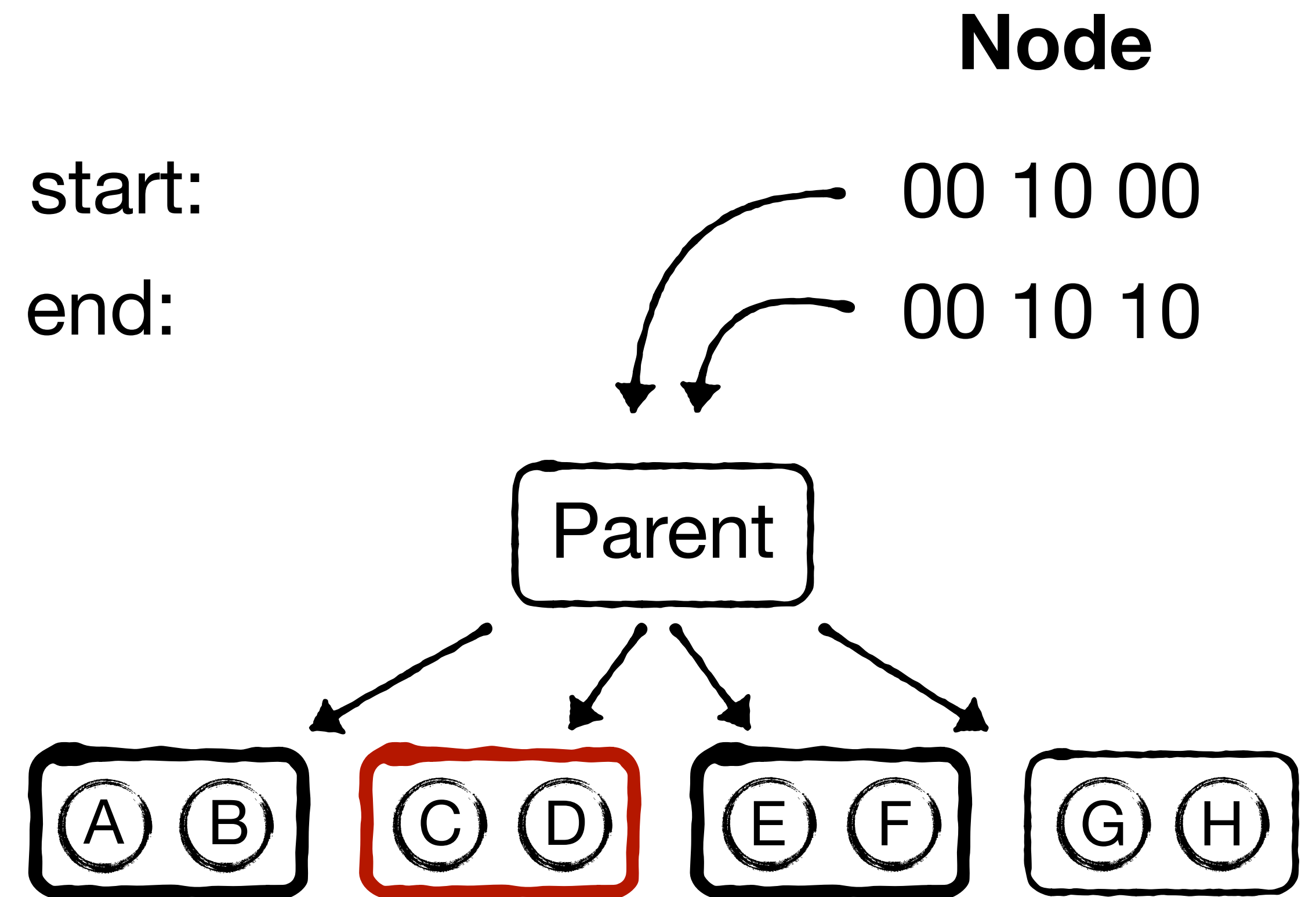
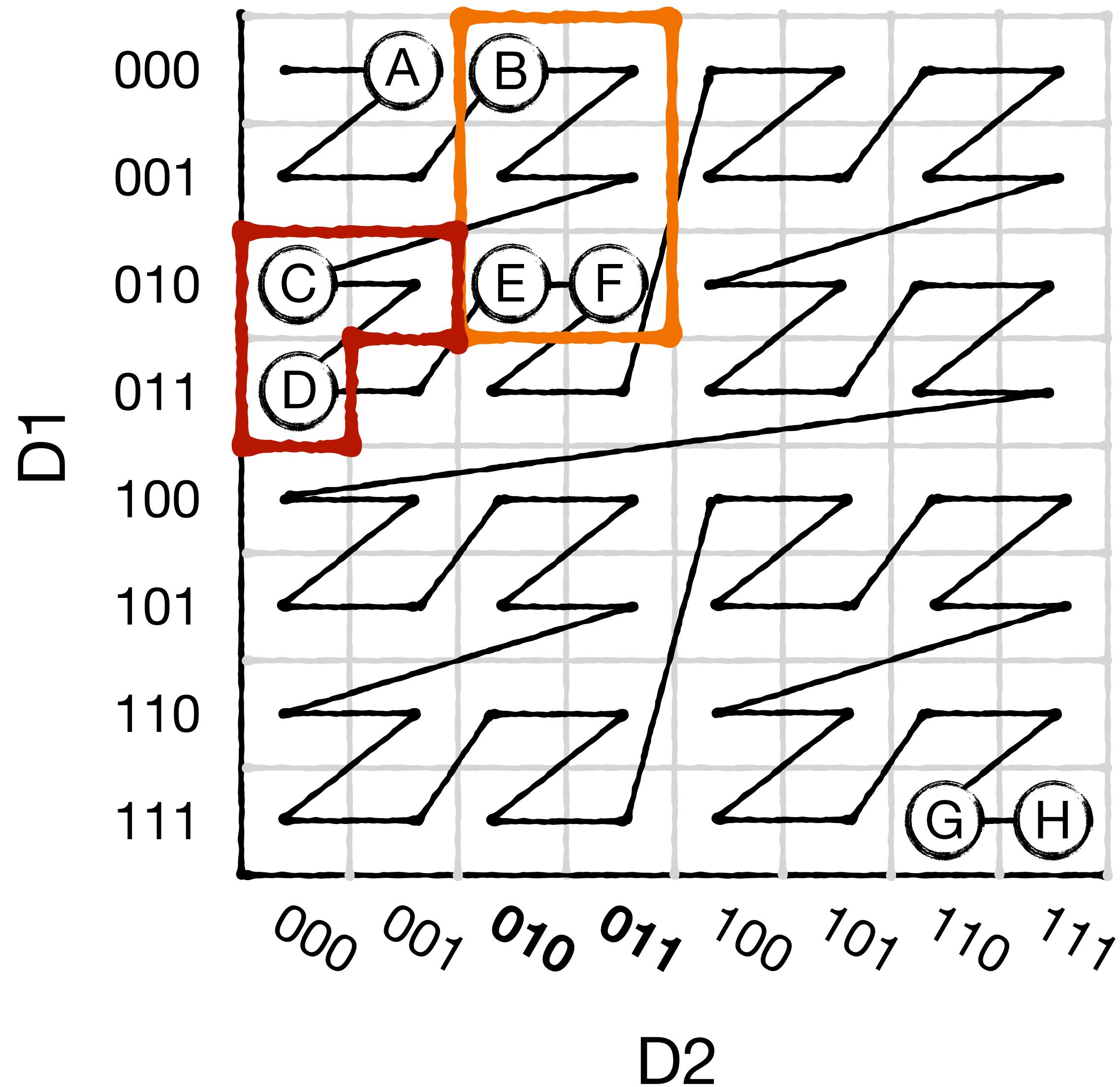
How to skip such nodes to speed up search?



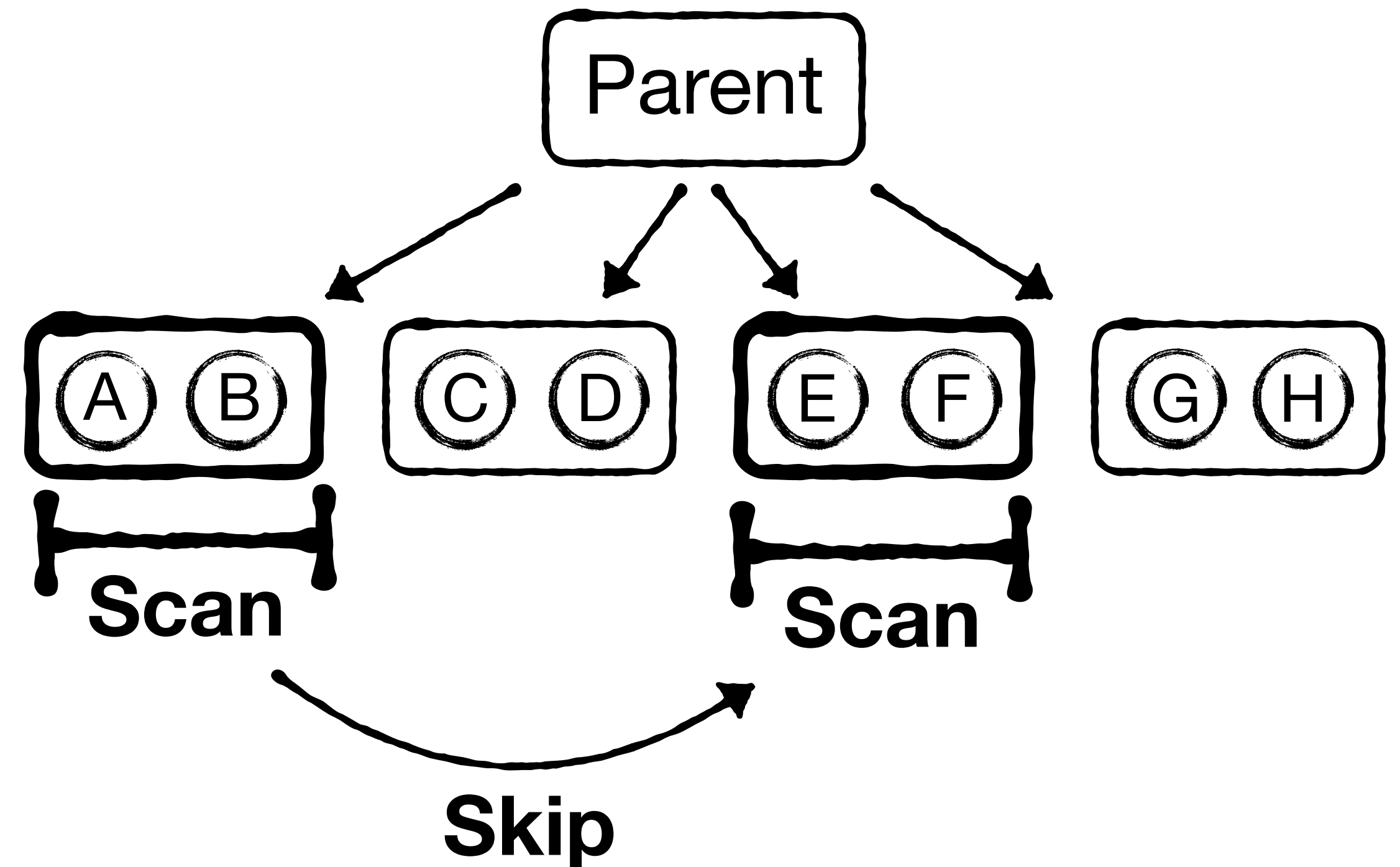
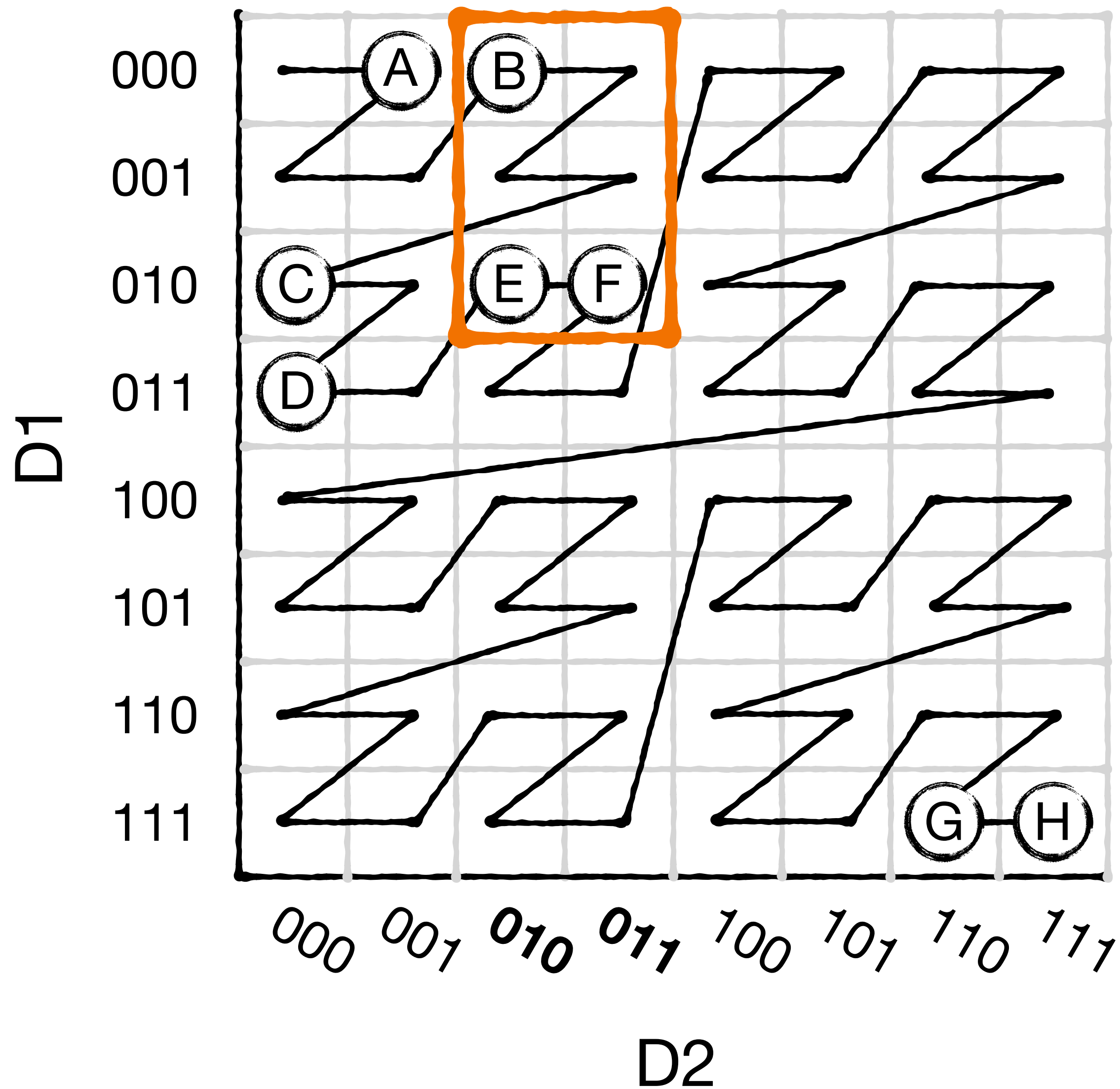
How to skip such nodes to speed up search?



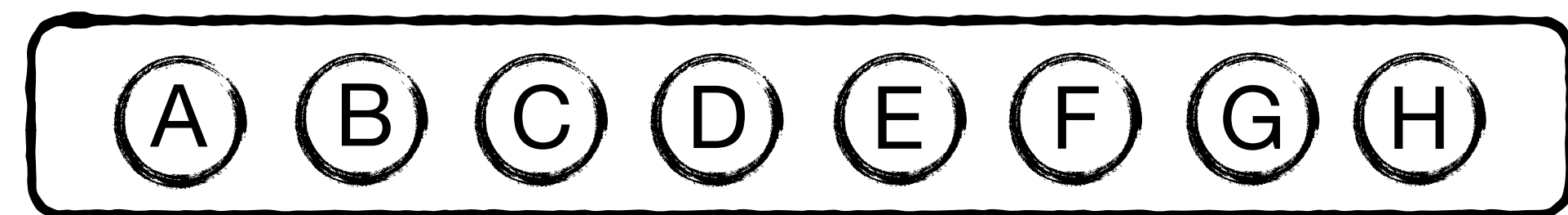
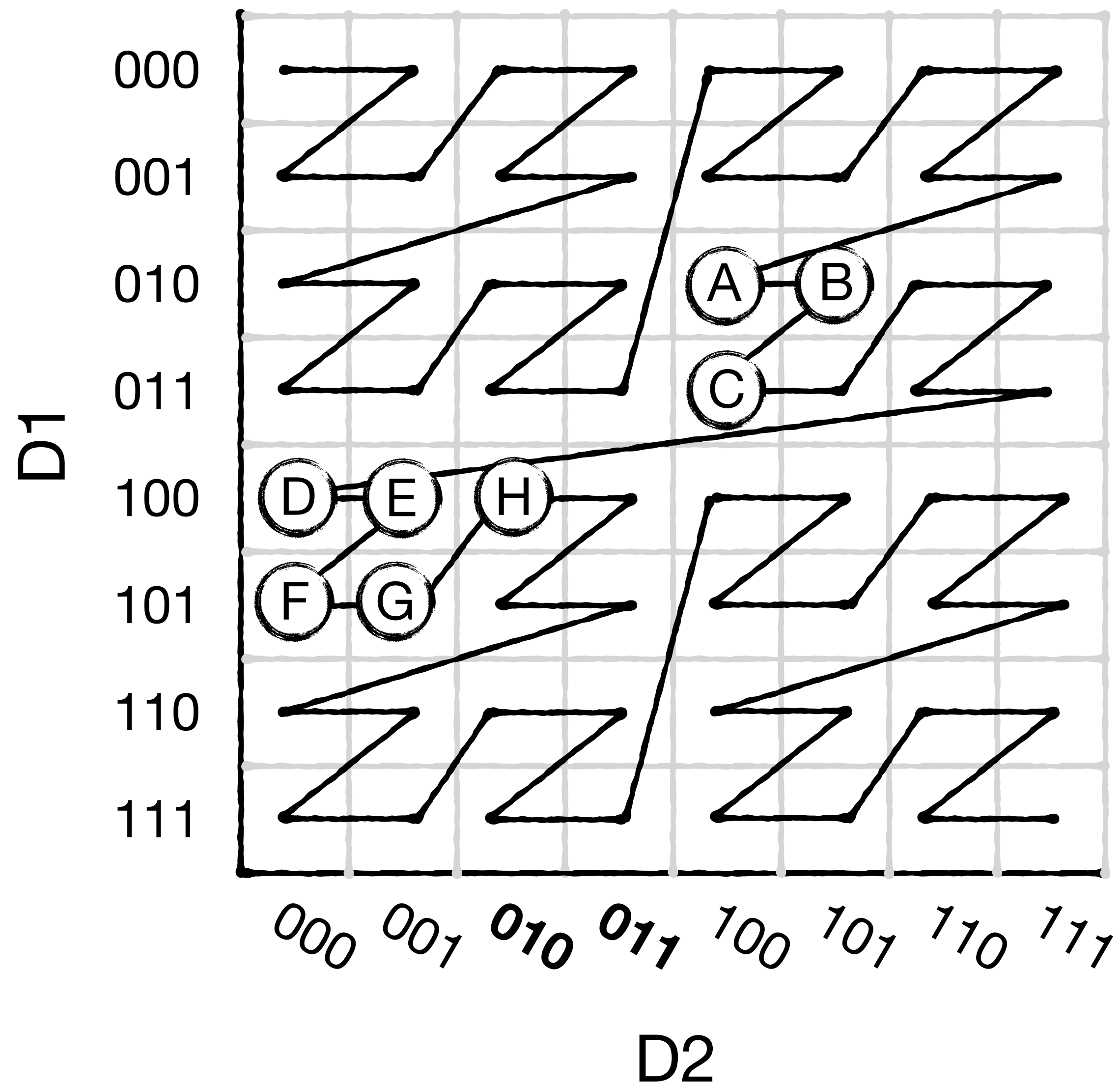
If parent contains start & end of each node, we can infer what to skip



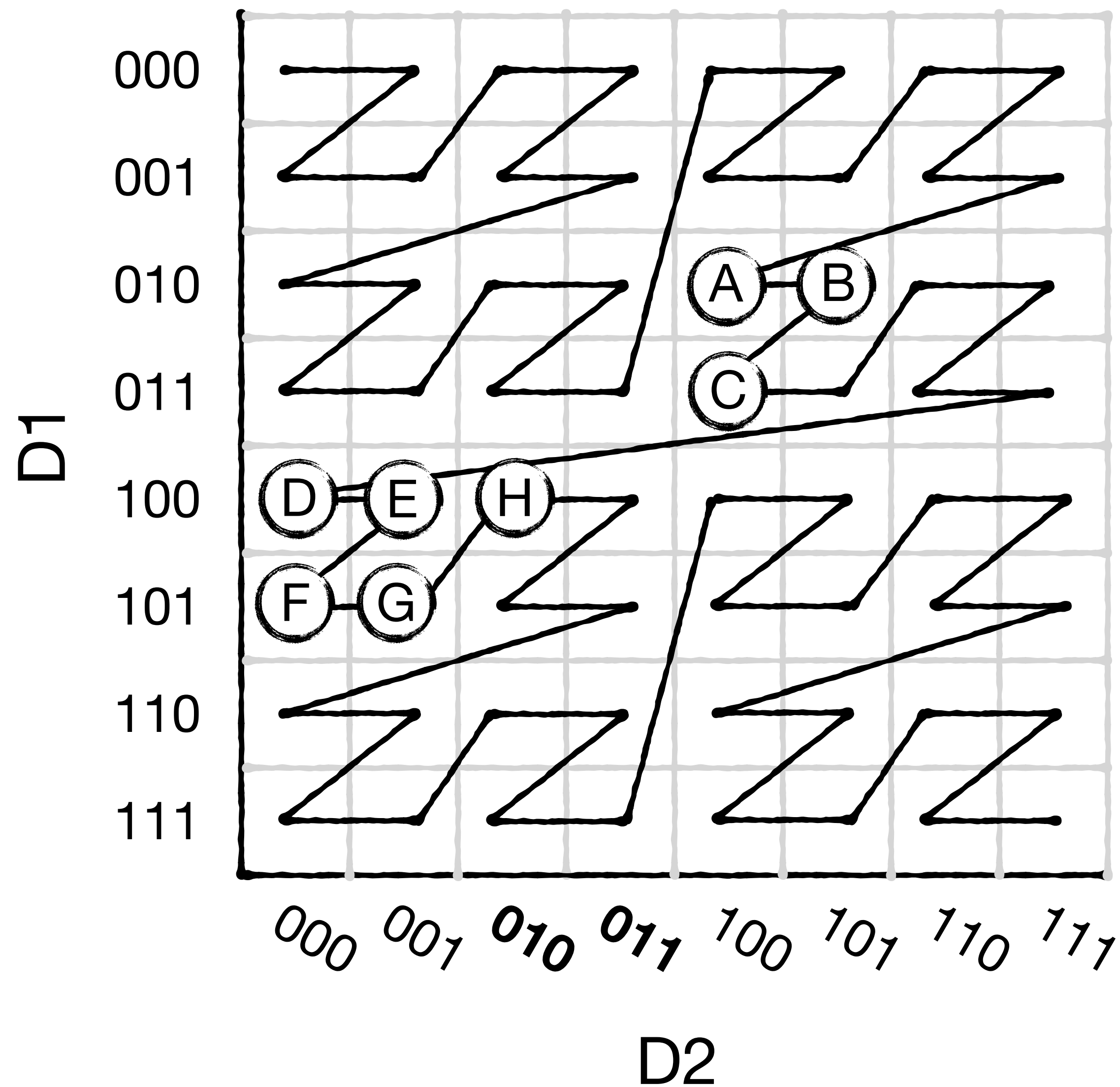
If parent contains start & end of each node, we can infer what to skip



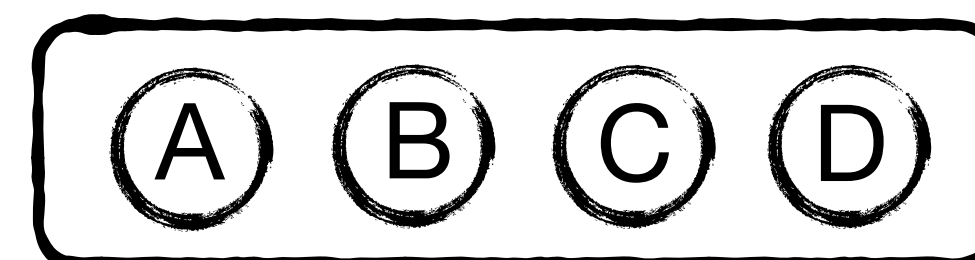
Consider the following node, which is full and about to split



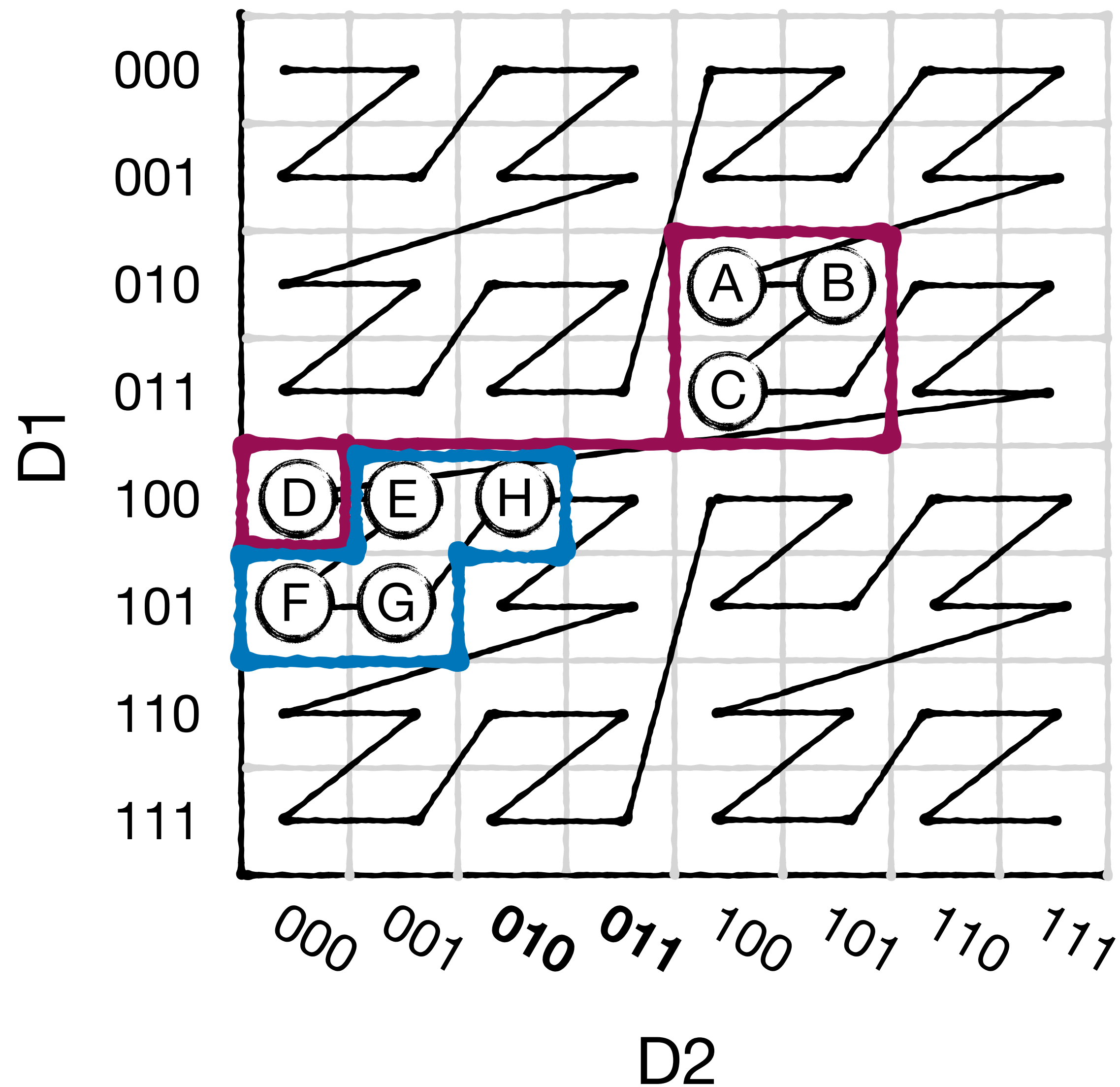
Consider the following node, which is full and about to split



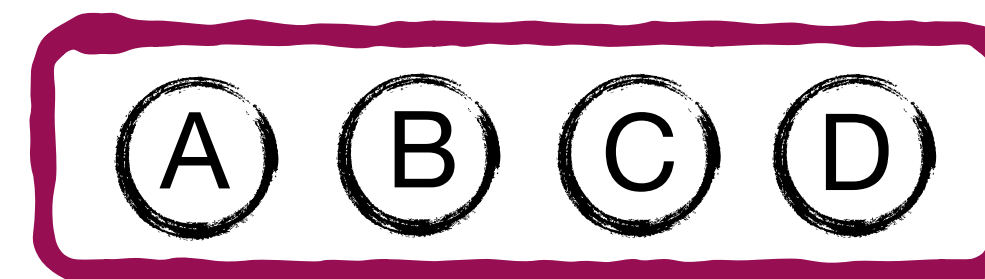
What's wrong with a fifty-fifty split?



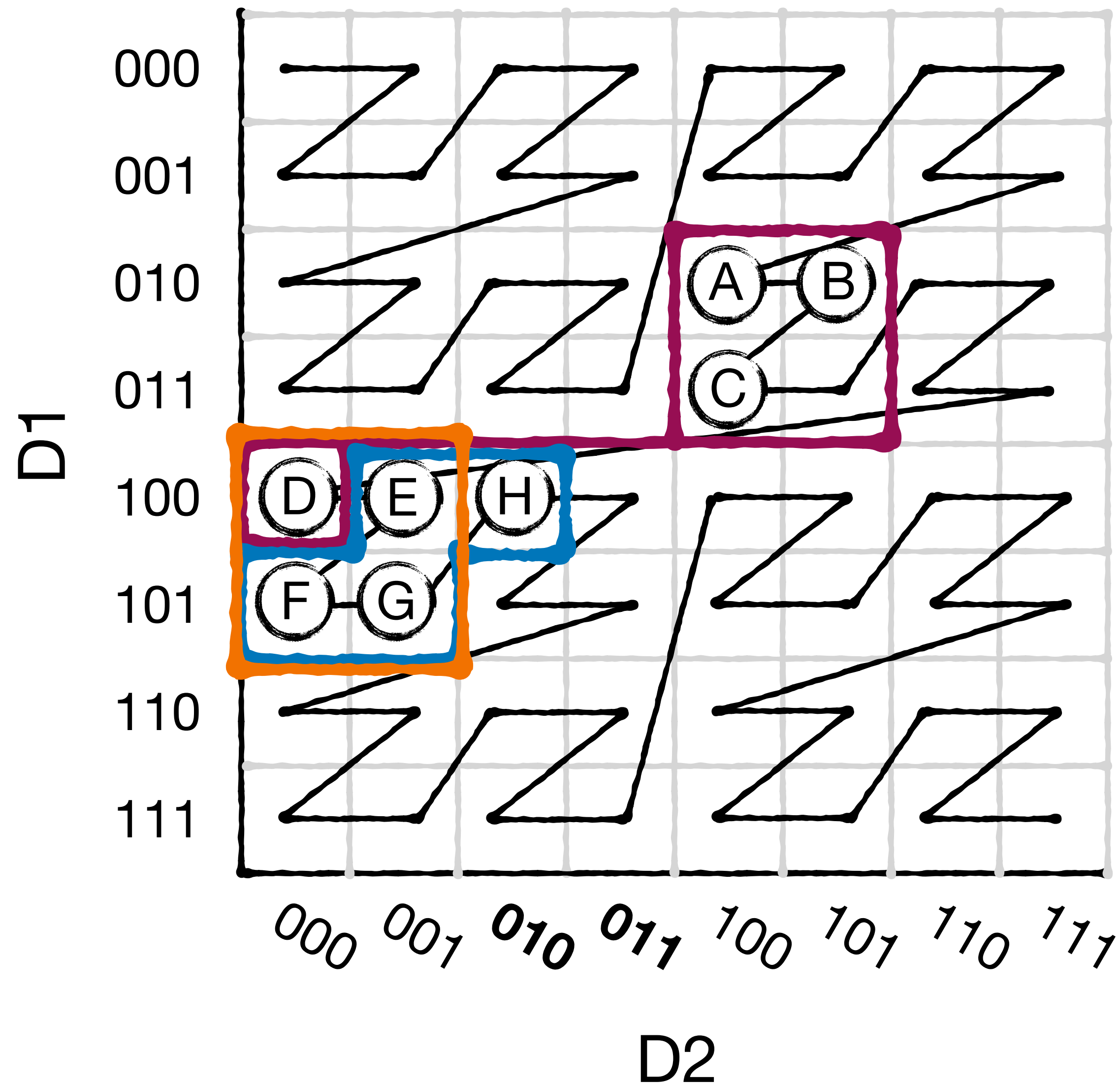
Consider the following node, which is full and about to split



What's wrong with a fifty-fifty split?



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

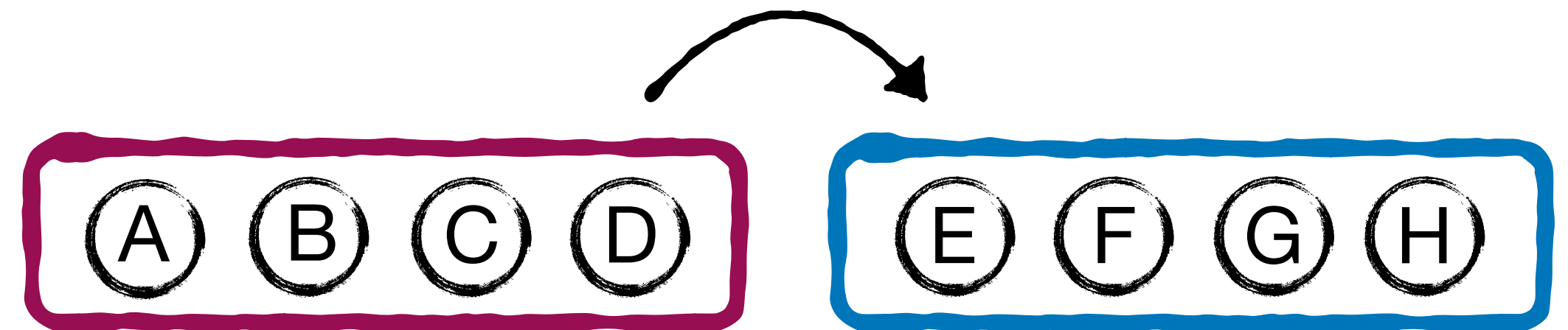
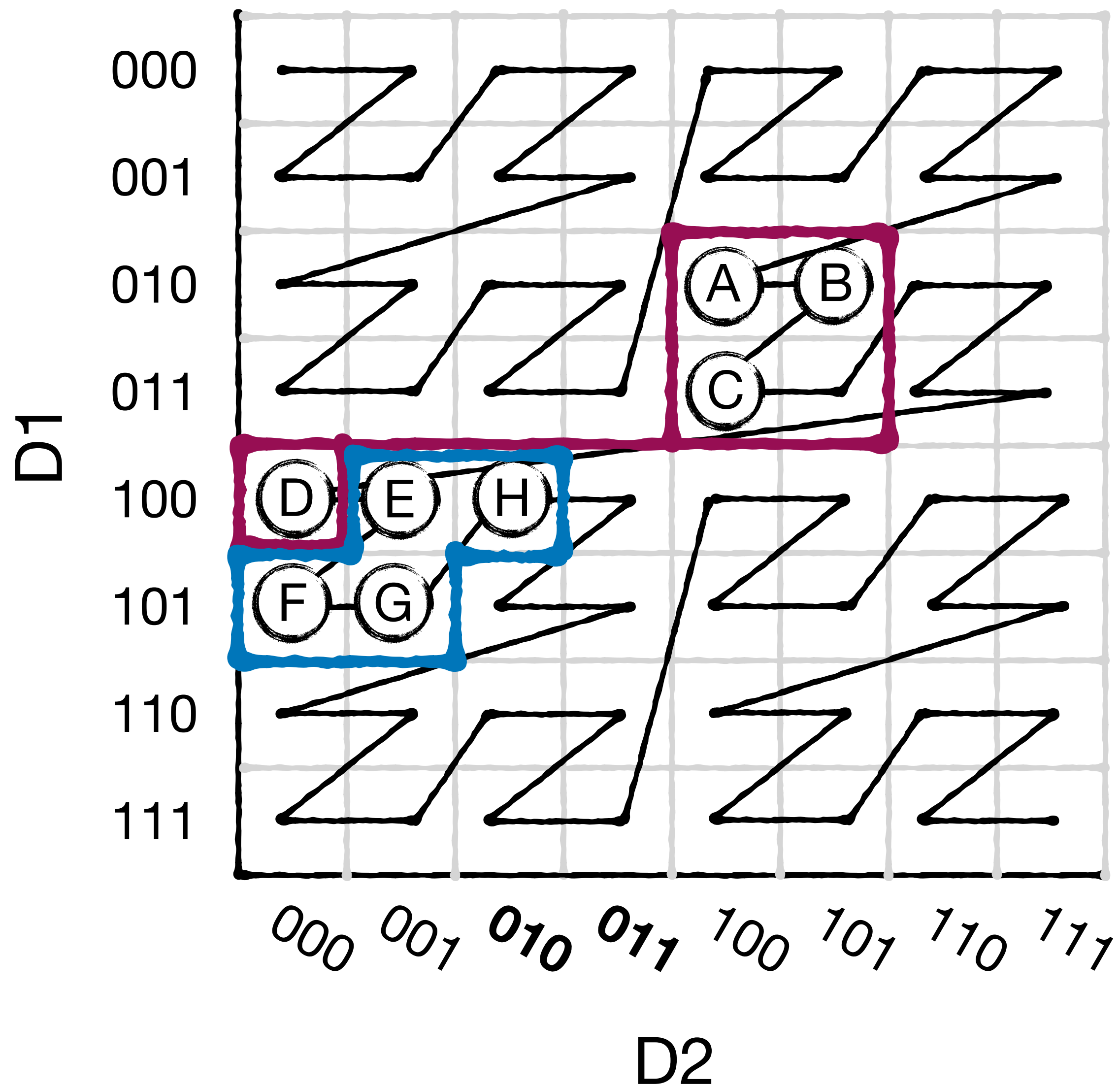


What's wrong with a fifty-fifty split?

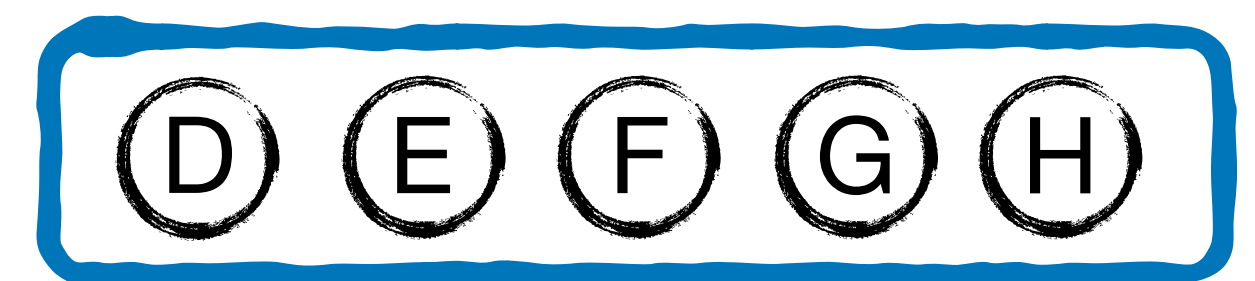
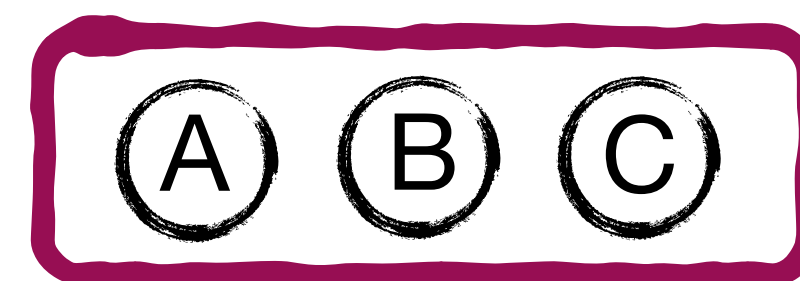
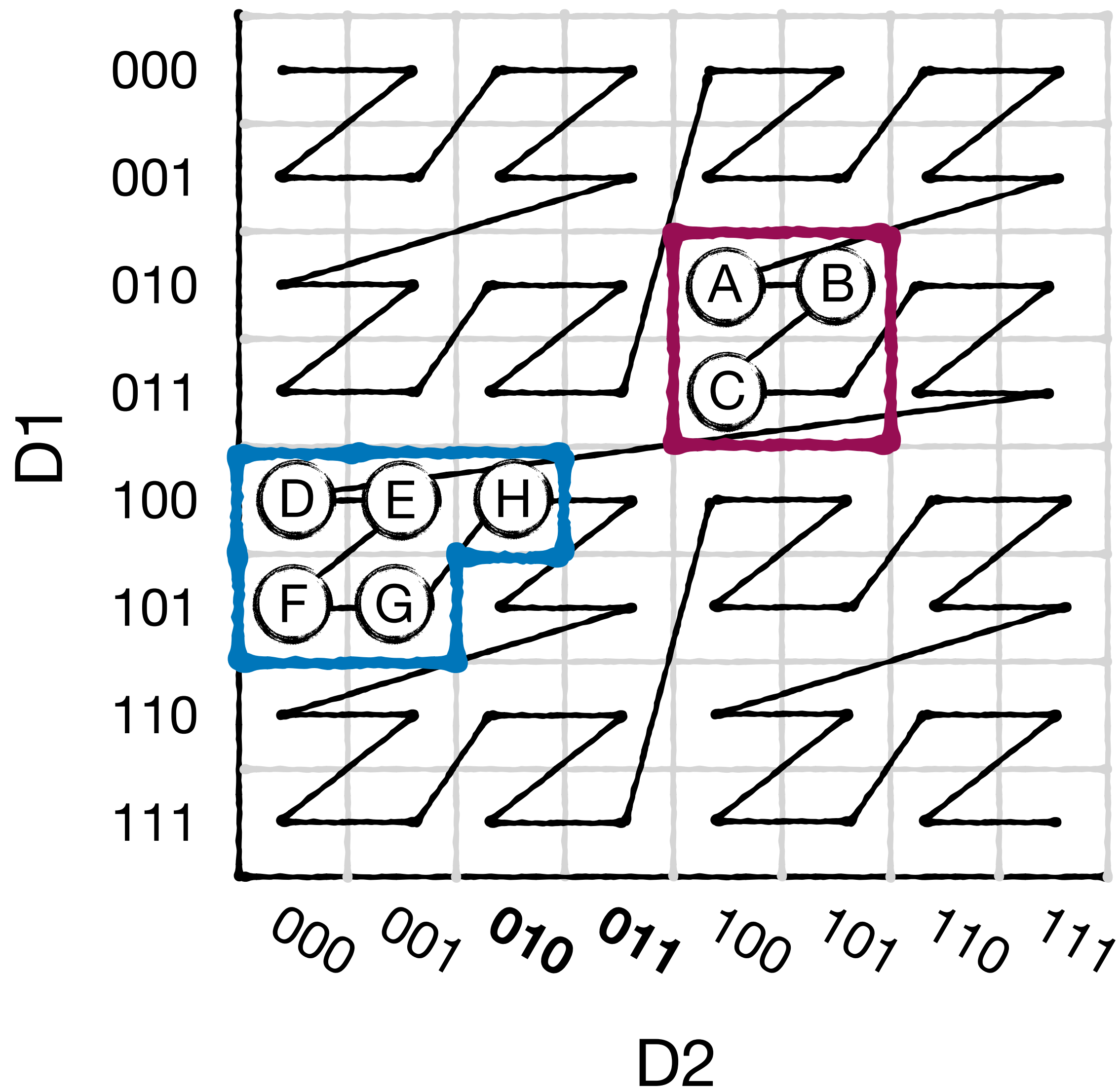


Queries are more likely to intersect with irregular node shapes

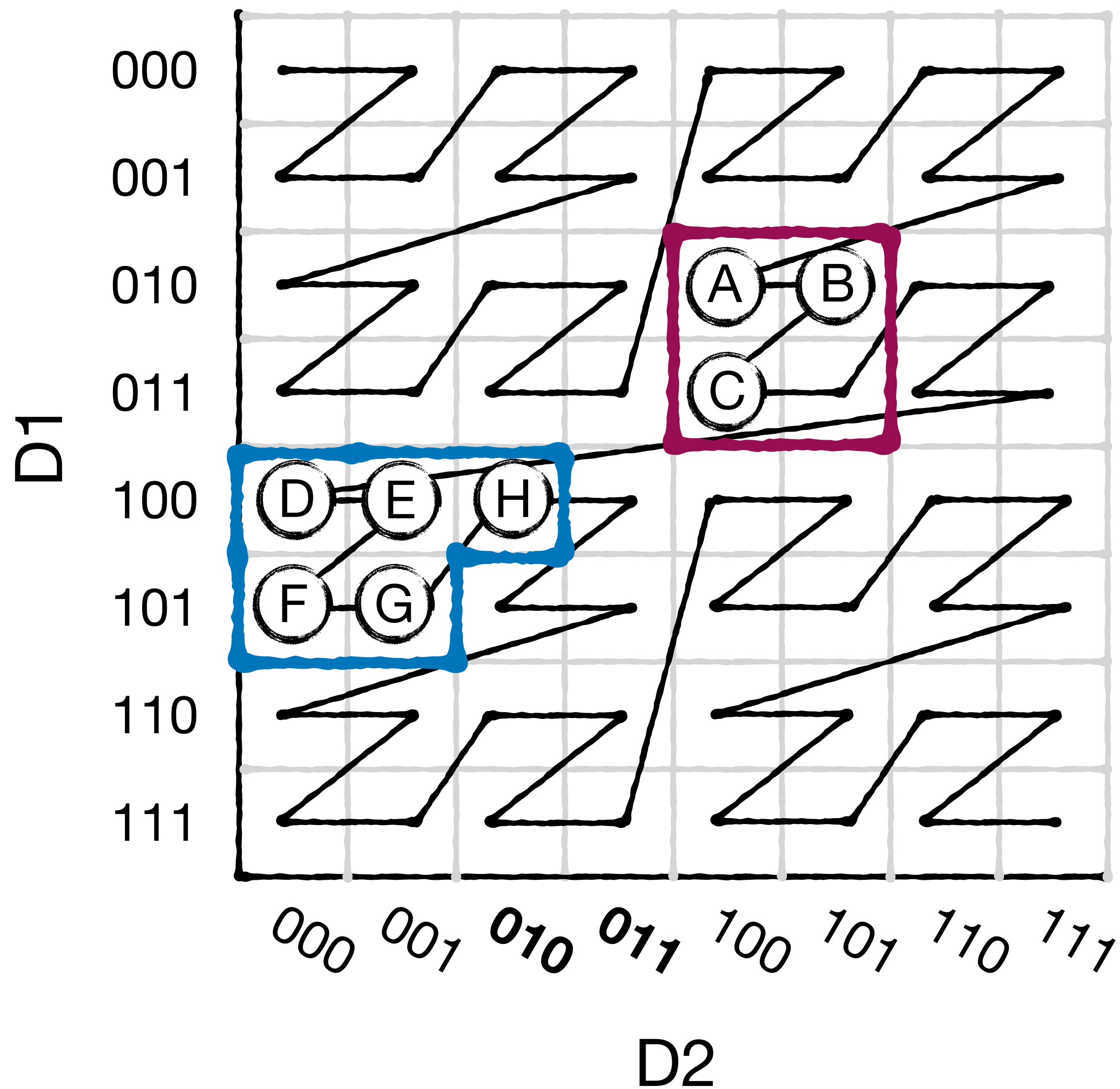
Insight: Picking some other close splitting point can greatly alleviate the issue



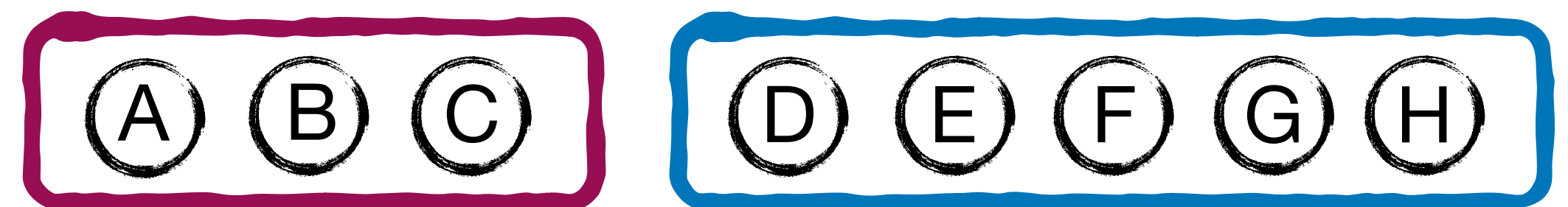
Insight: Picking some other close splitting point can greatly alleviate the issue



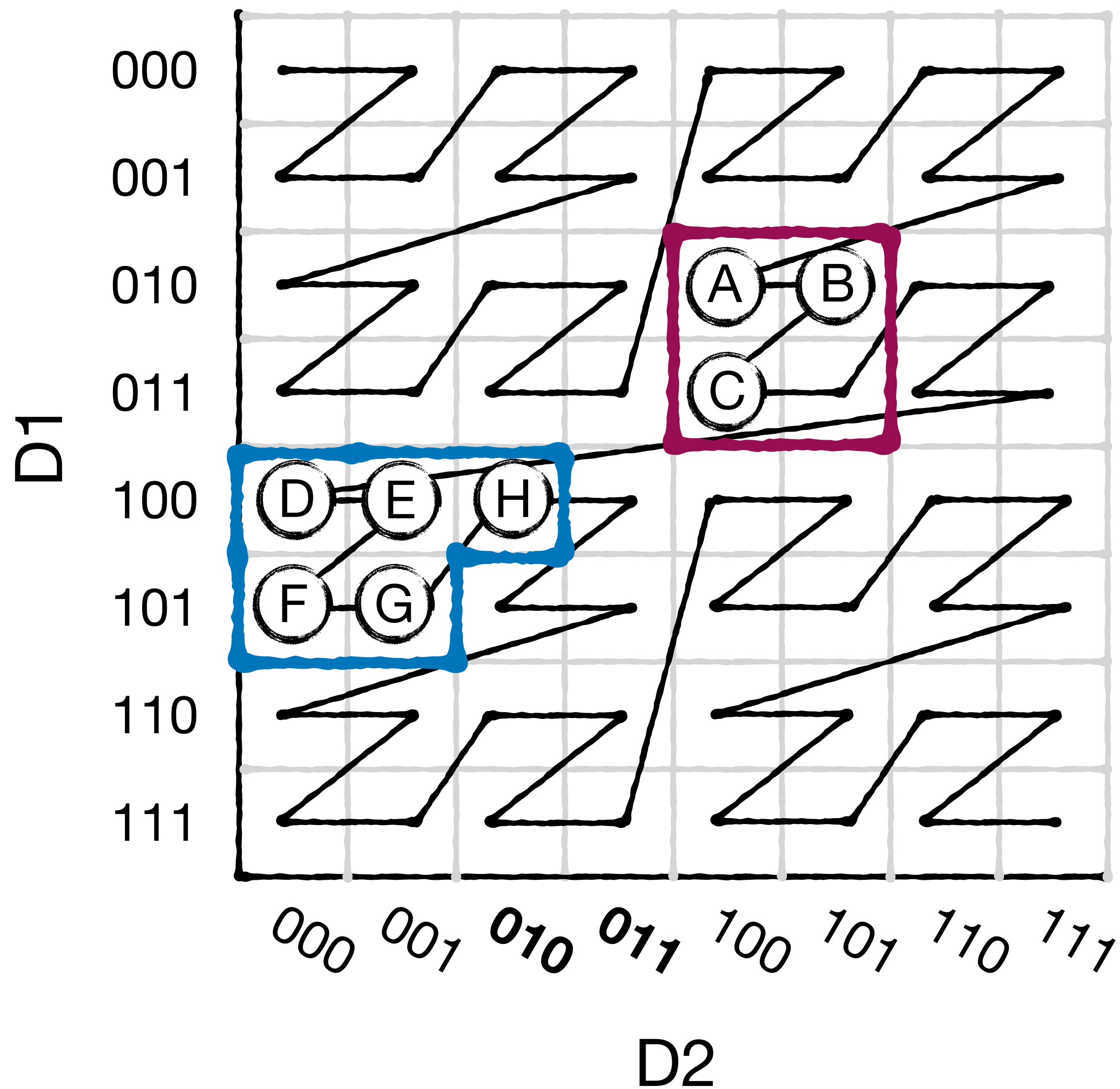
Insight: Picking some other close splitting point can greatly alleviate the issue



What's the trade-off?

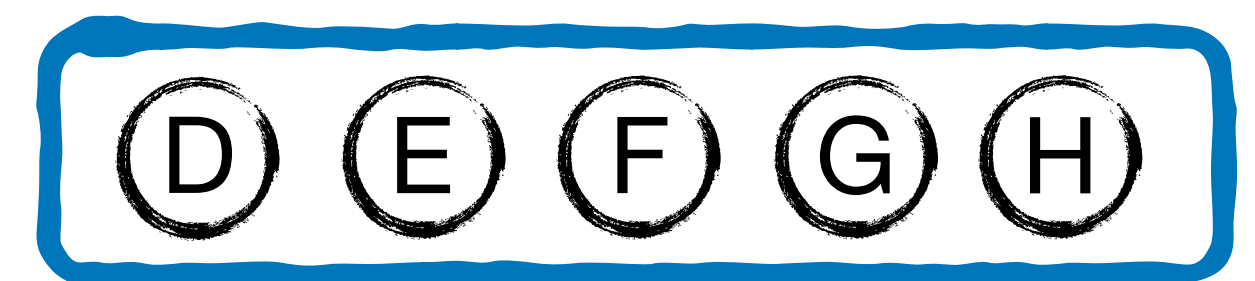
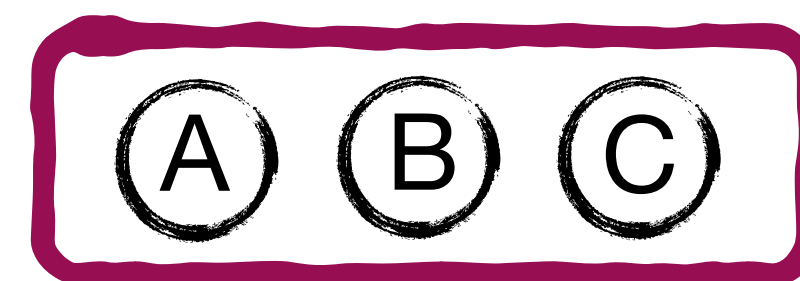


Insight: Picking some other close splitting point can greatly alleviate the issue

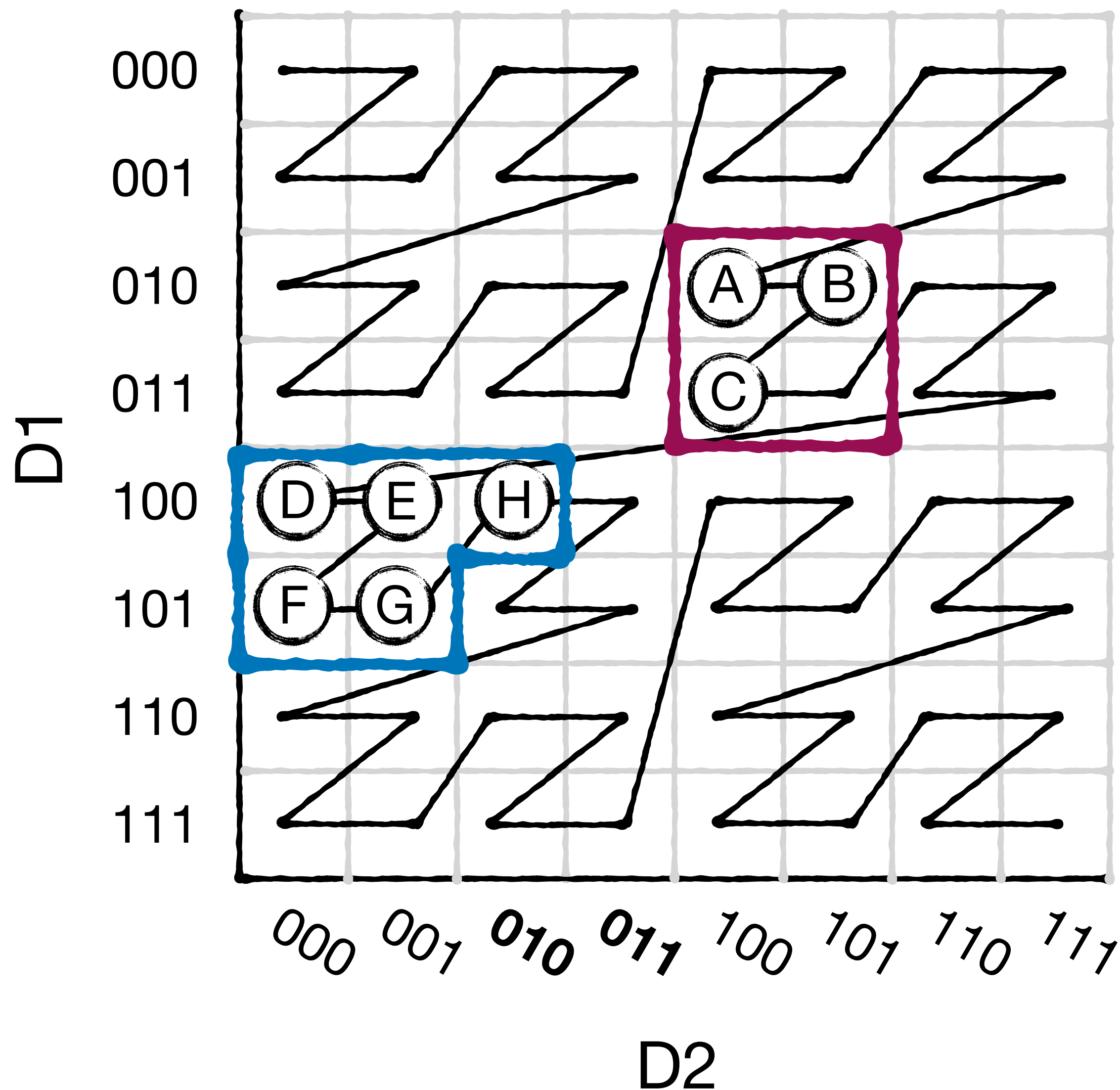


What's the trade-off?

Some nodes are less utilized

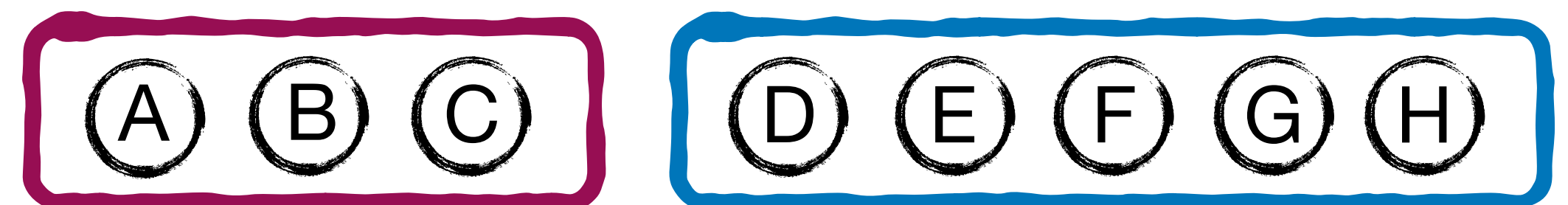


Insight: Picking some other close splitting point can greatly alleviate the issue



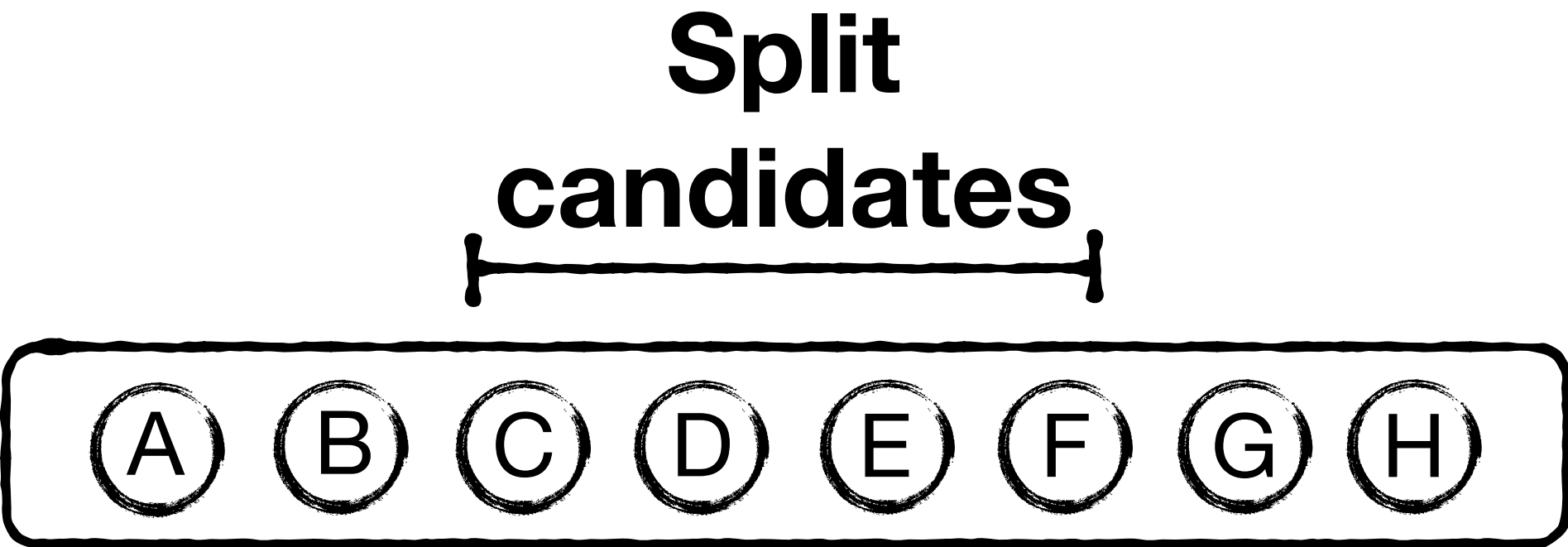
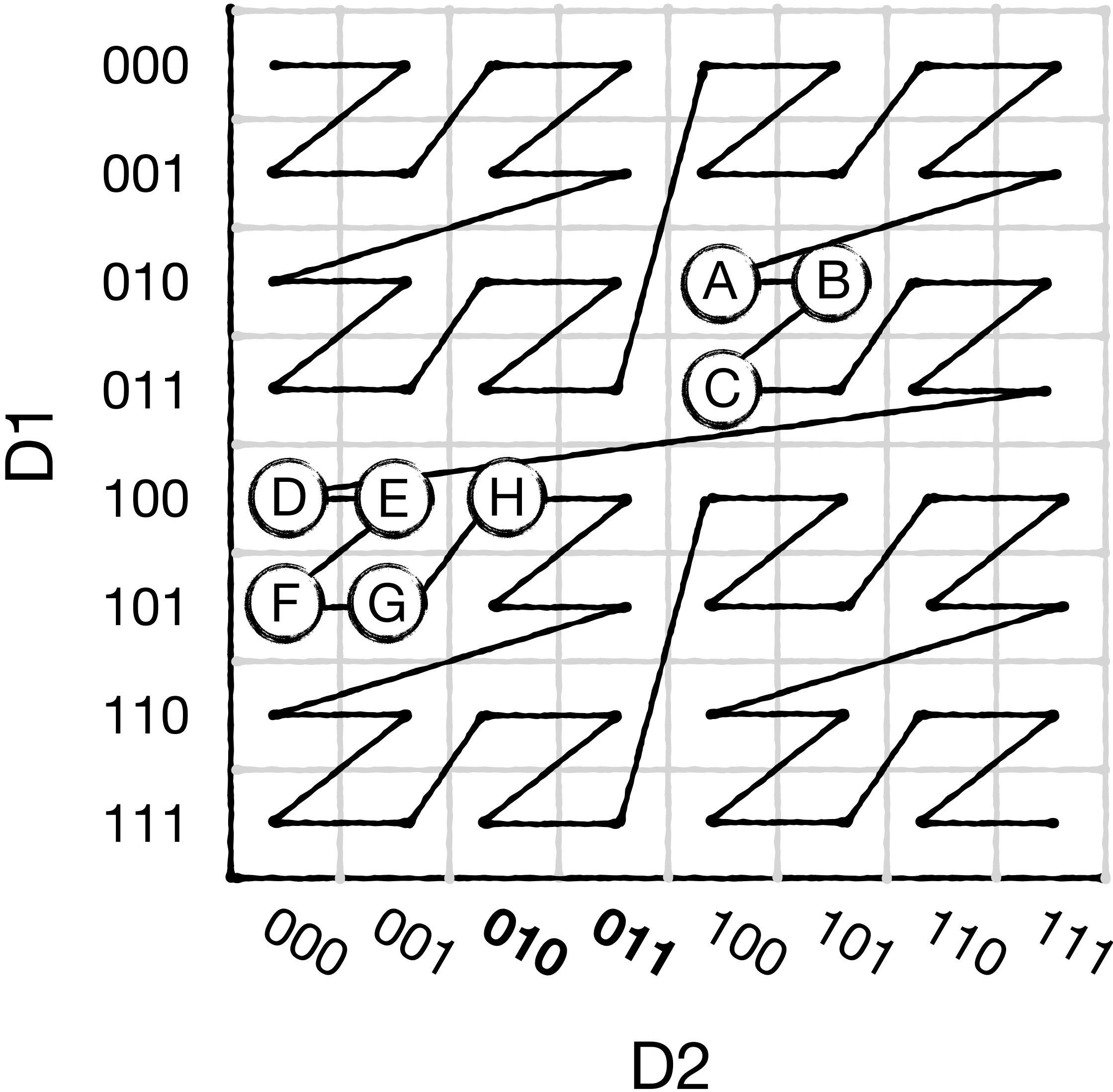
What's the trade-off?

Some nodes are less utilized

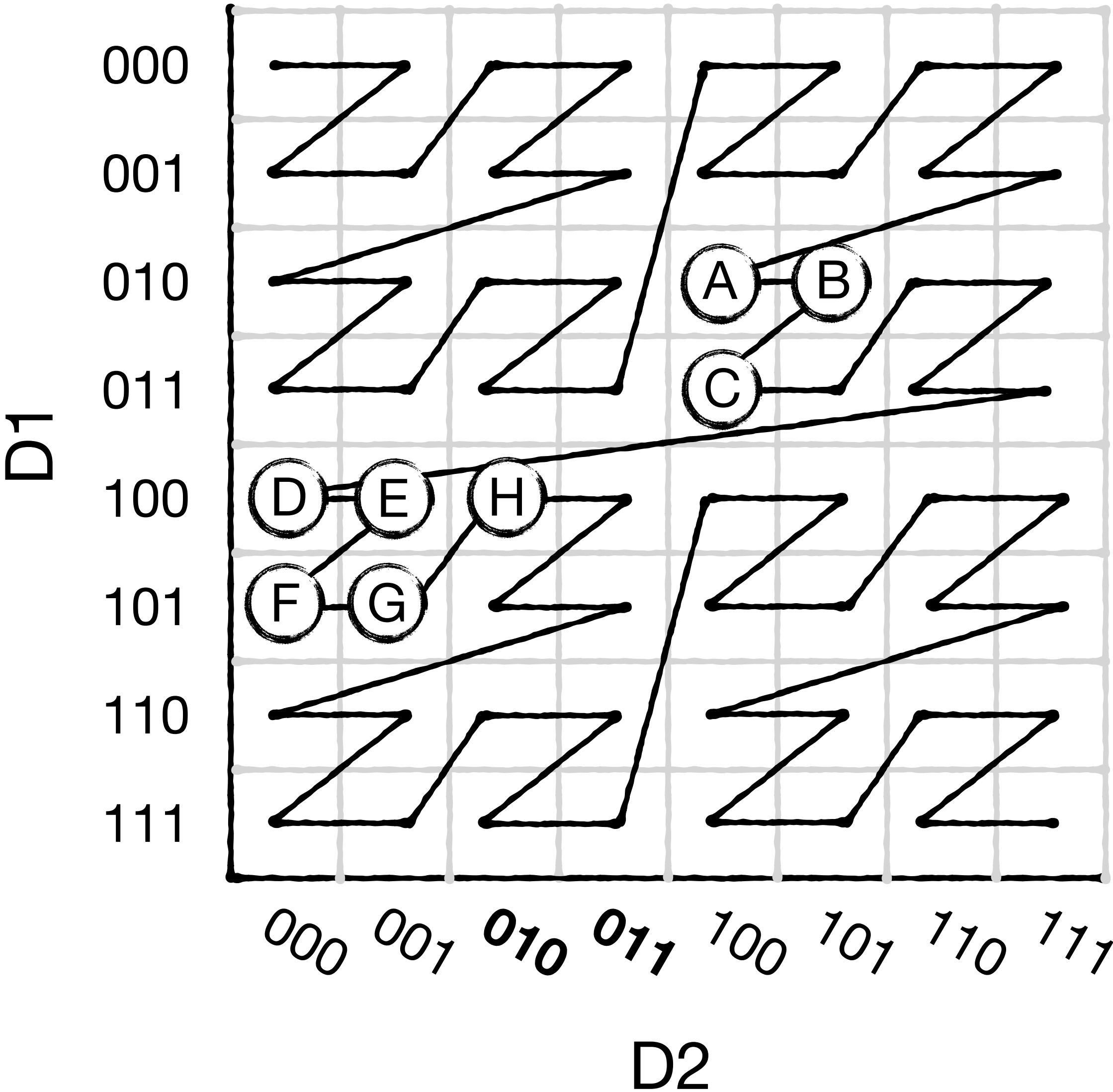


**Can lead to deeper tree with
more space wastage**

Balance by picking best split from a middle nodes



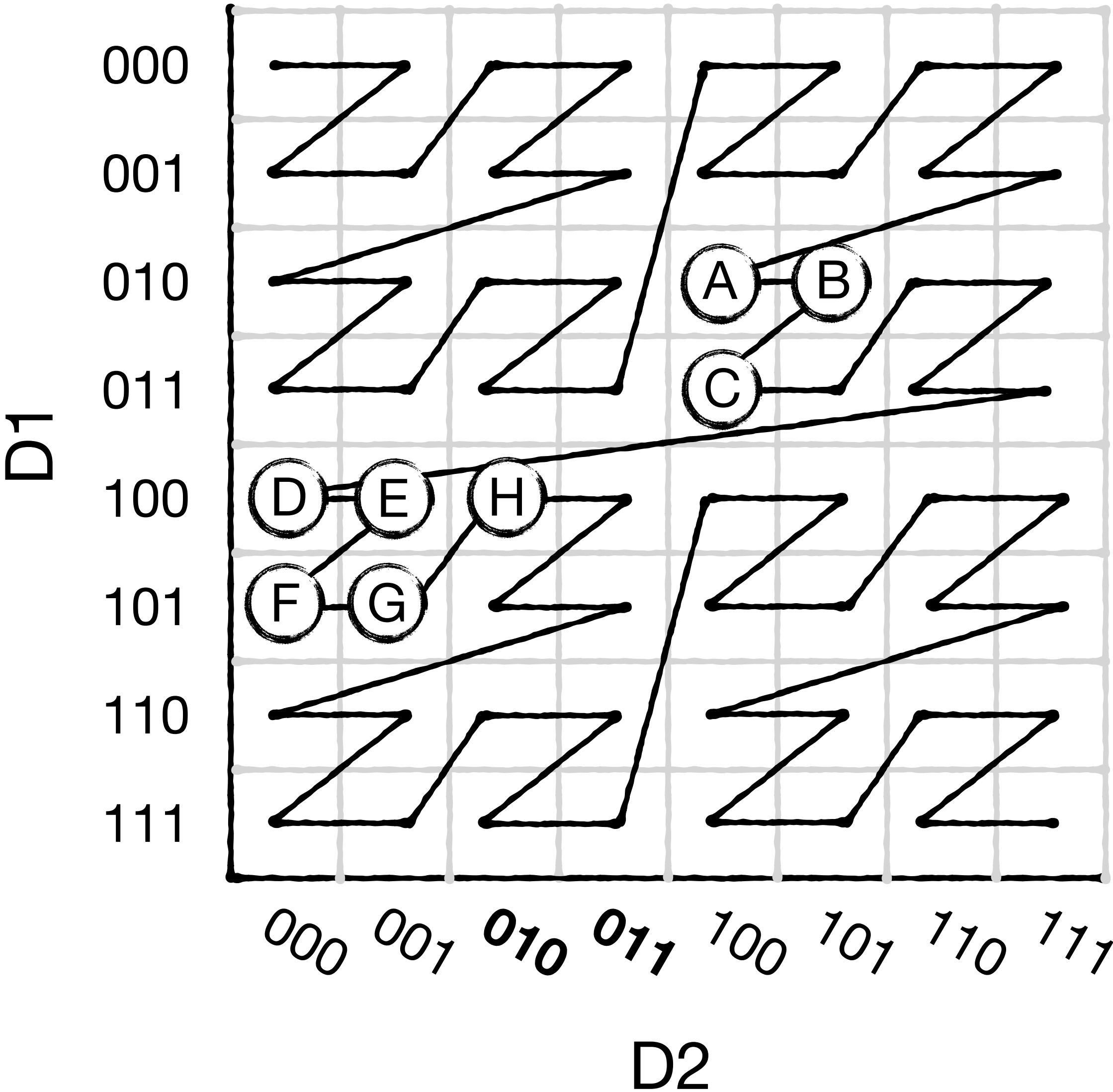
Balance by picking best split from a middle nodes



Split candidates

| | | | | | | | |
|---|----|----|----|---|---|---|---|
| A | B | C | D | E | F | G | H |
| C | 01 | 00 | 10 | | | | |
| D | 10 | 00 | 00 | | | | |
| E | 10 | 00 | 01 | | | | |
| F | 10 | 00 | 10 | | | | |

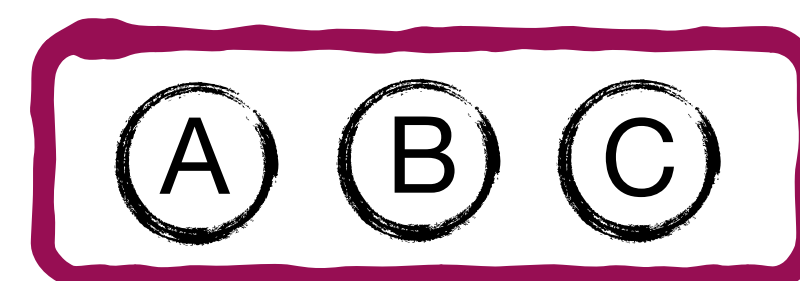
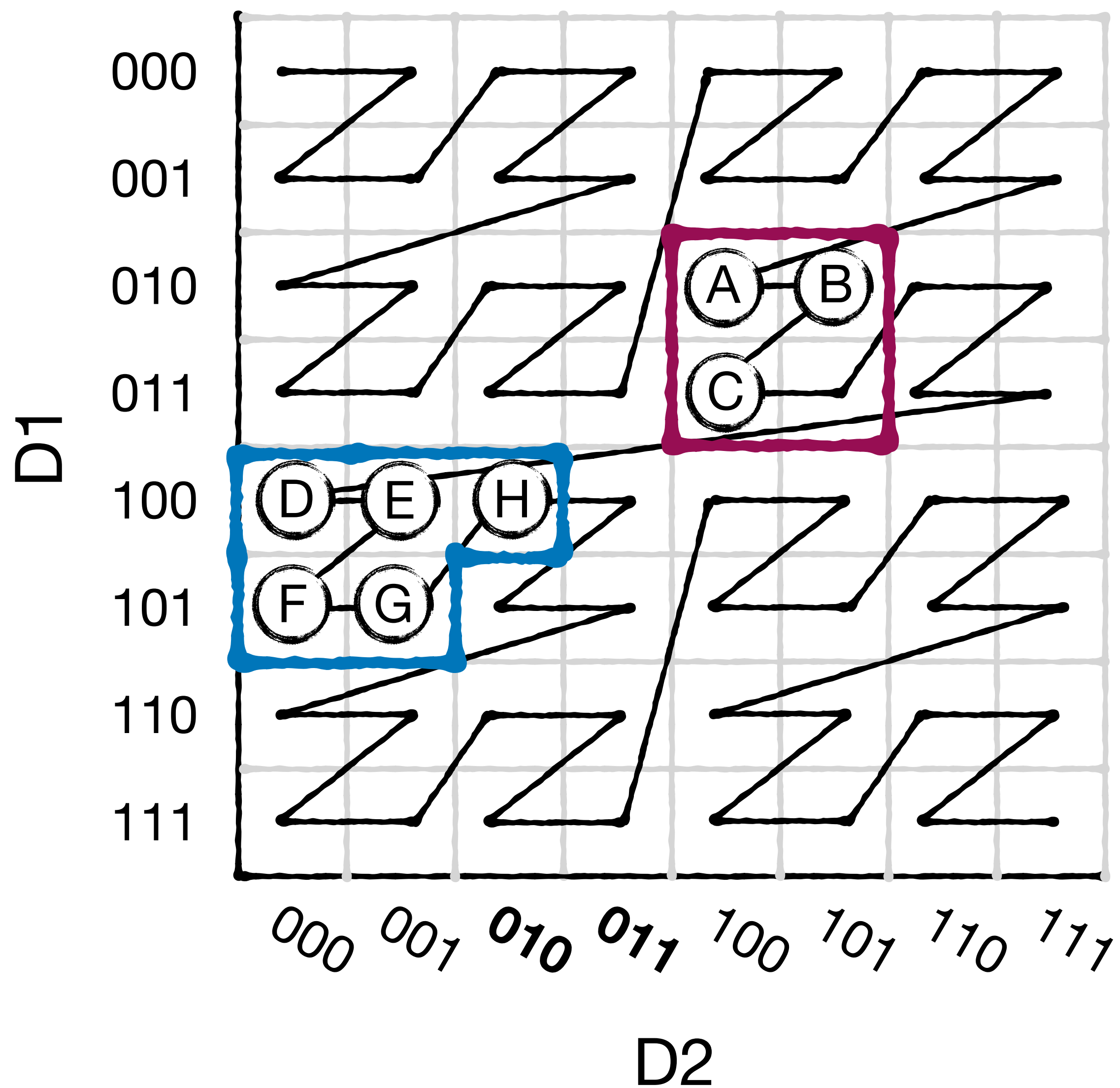
Balance by picking best split from a middle nodes



Split candidates

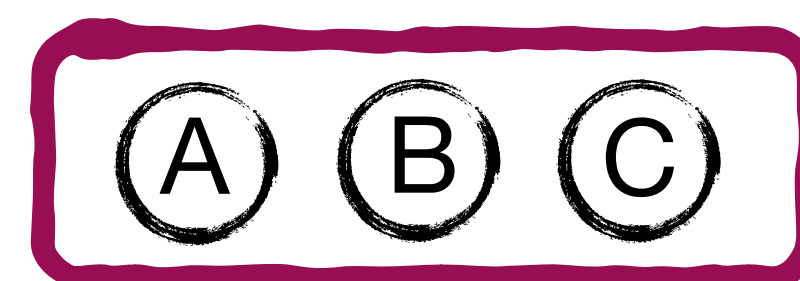
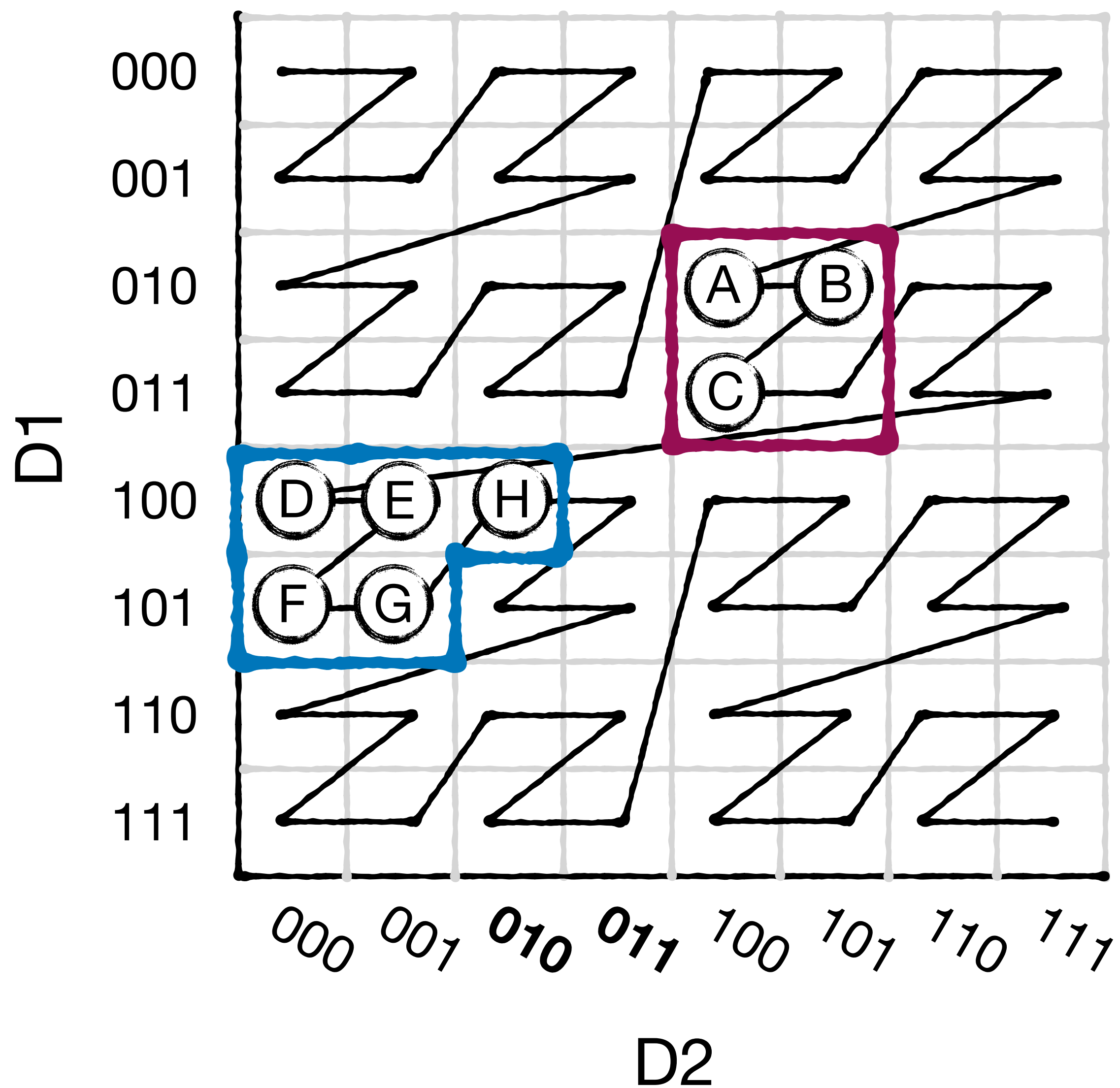
A B C D E F G H

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
 E 10 00 01
 F 10 00 10

← Most trailing
 zeros, best splitter



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

**And now, a student
presentation**