Andra självbalanserande träd

Koffman & Wolfgang

- kapitel 9, avsnitt 4–6

2-3-träd

Ett 2-3-träd har två sorters noder: 2-noder och 3-noder

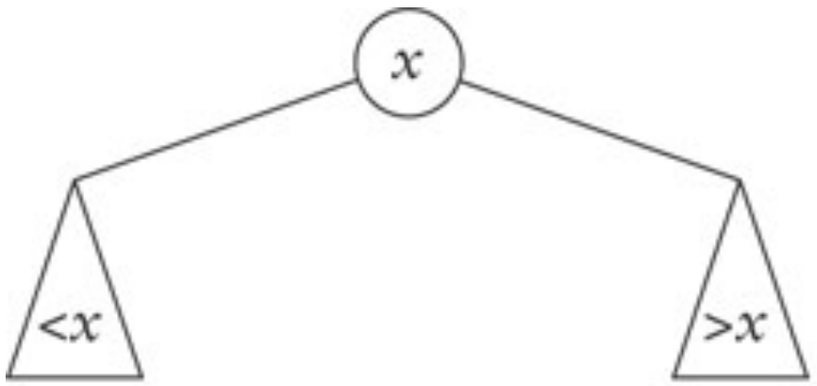
En 2-nod är en vanlig binär sökträdsnod:

- den innehåller ett datafält (d_1) och två barn (b_1, b_2)
- det första barnets (b_1) data är mindre än nodens (d_1)
- det andra barnets (b_2) data är större än nodens (d_1)

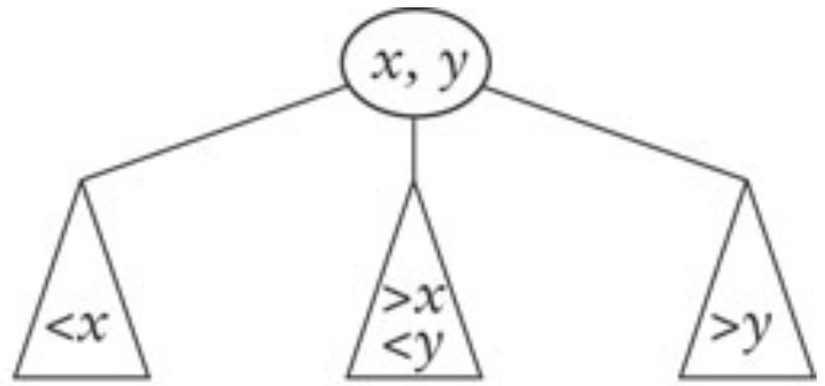
En 3-nod har två datafält (d_1, d_2) och tre barn (b_1, b_2, b_3):

- det första datafältet (d_1) är mindre än det andra (d_2)
- det första barnets (b_1) data är mindre än d_1
- det andra barnets (b_2) data ligger mellan d_1 och d_2
- det tredje barnets (b_3) data är större än d_2

2-3-träd

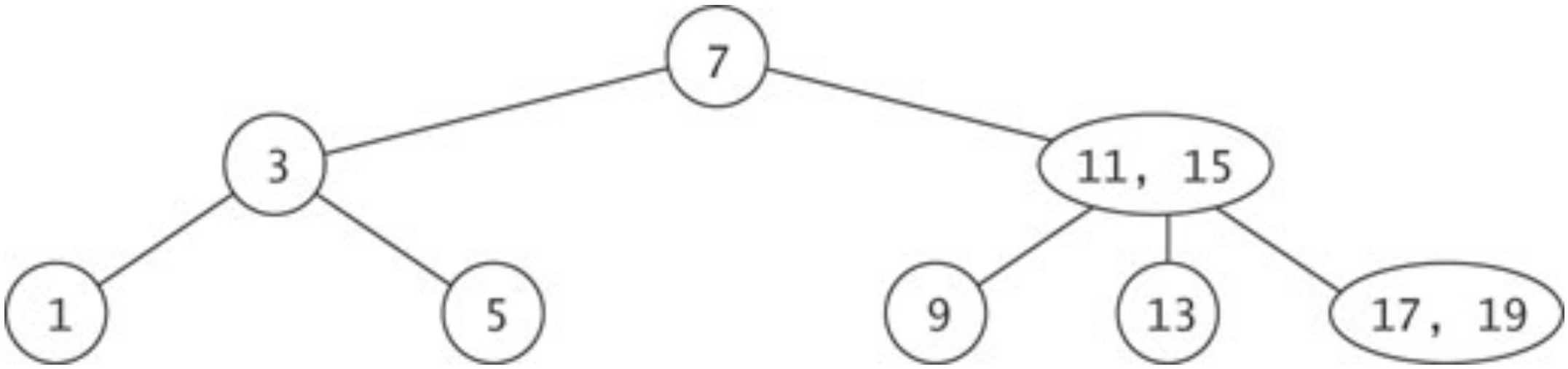


2-node



3-node

Ett exempel på ett 2-3-träd:



Ett 2-3-exempel

Nu ska vi bygga ett 2-3-träd för orden i

”The quick brown fox jumps over the lazy dog”

The...

The

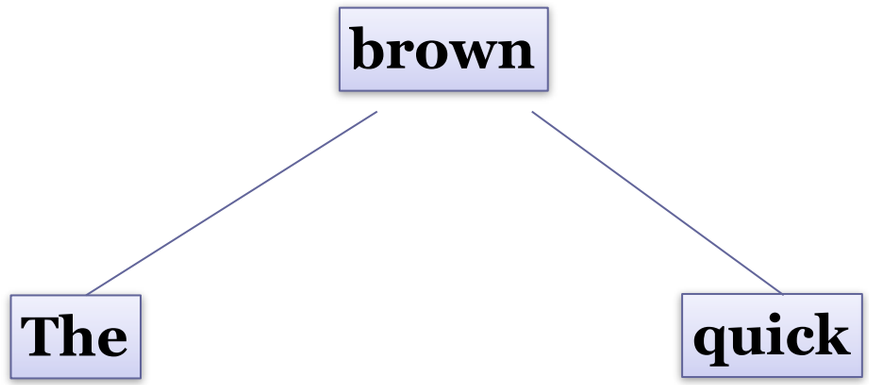
The quick…

The, quick

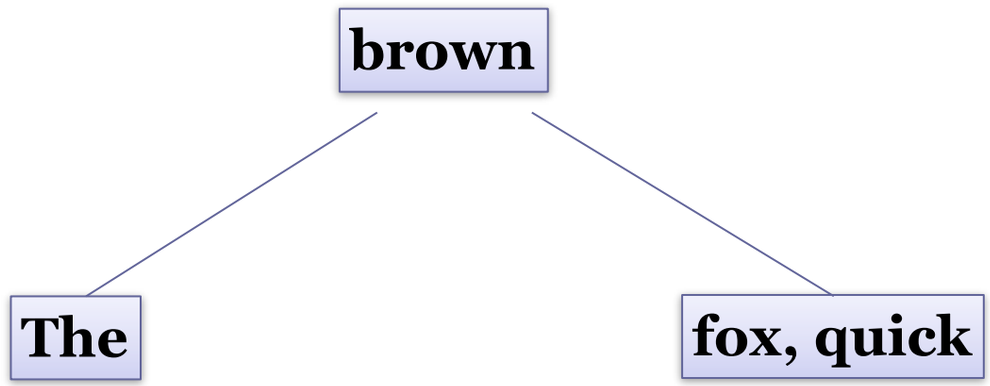
The quick brown…

The, brown, quick

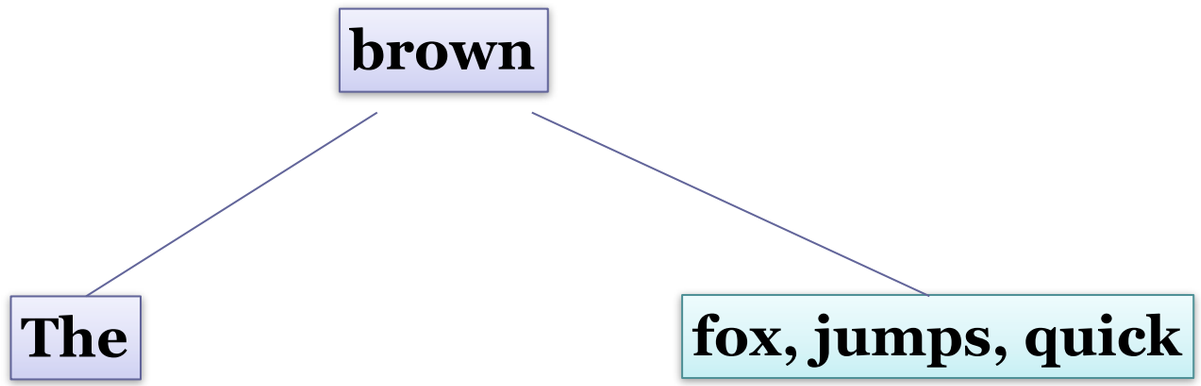
The quick brown…



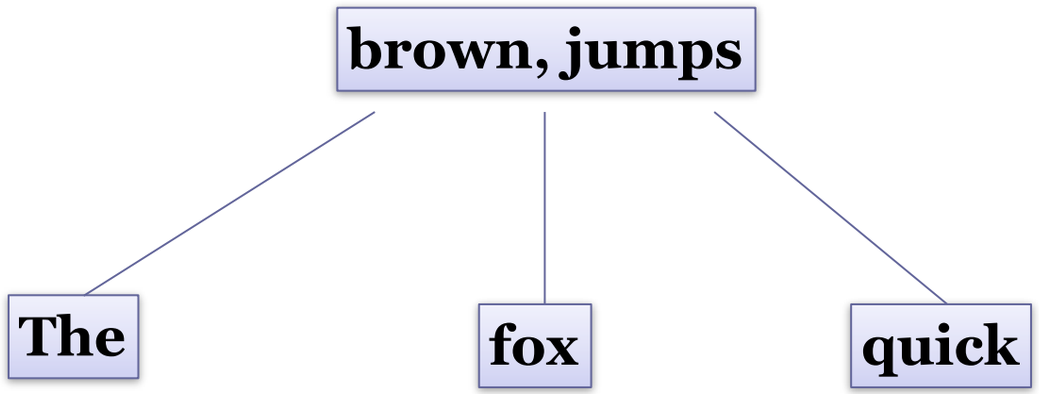
The quick brown fox…



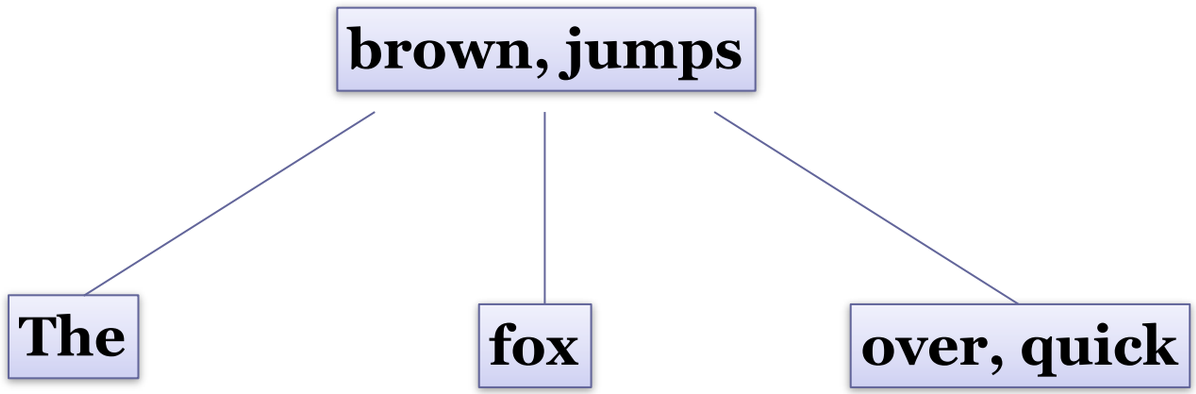
The quick brown fox jumps…



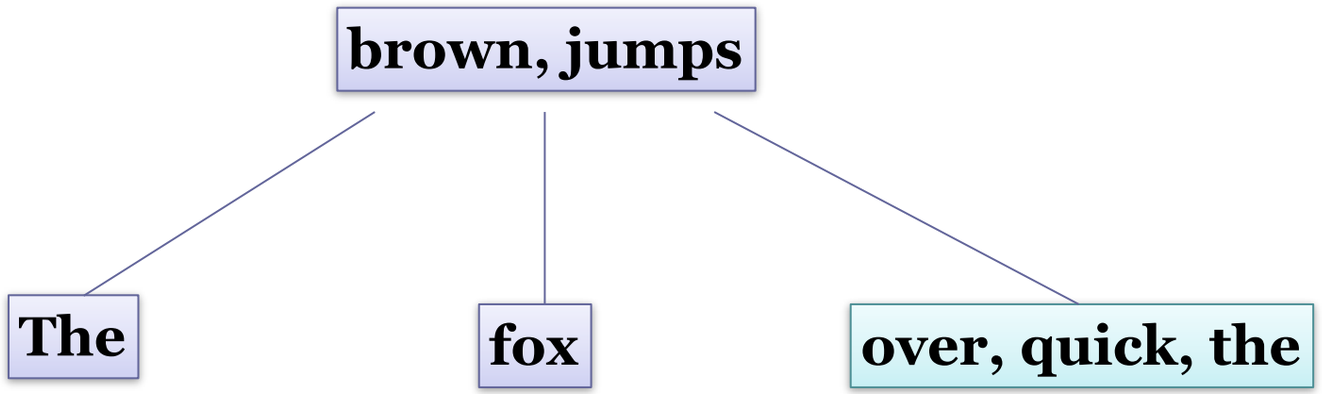
The quick brown fox jumps…



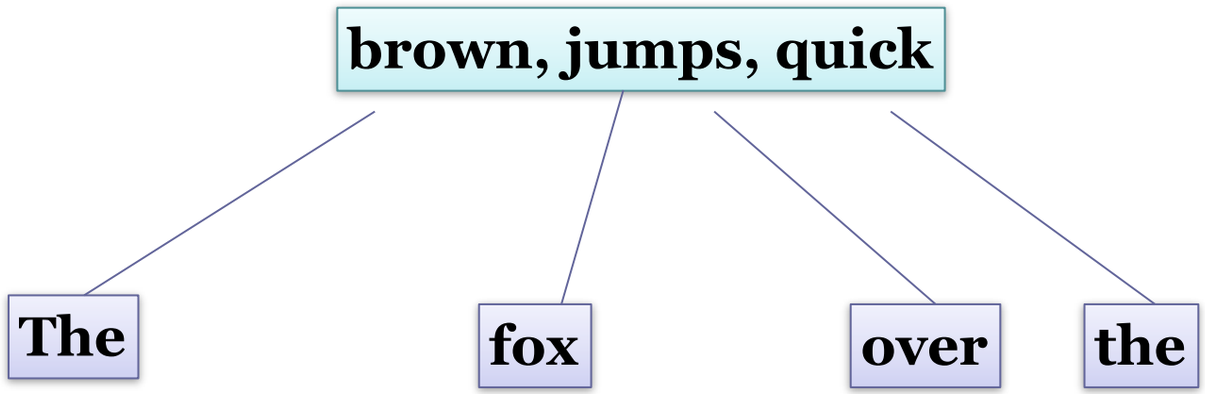
The quick brown fox jumps over…



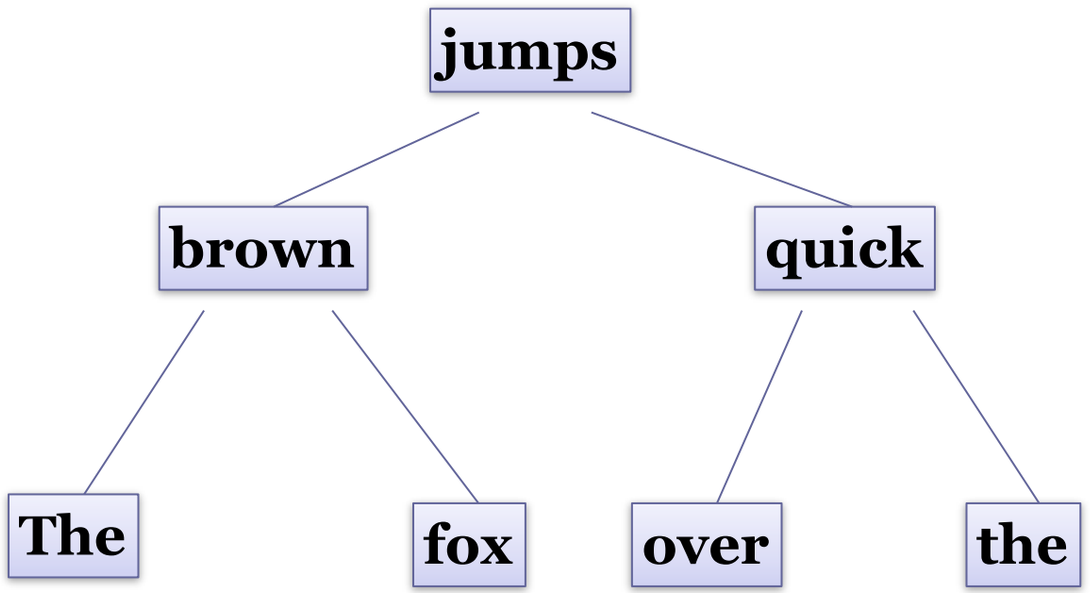
The quick brown fox jumps over the…



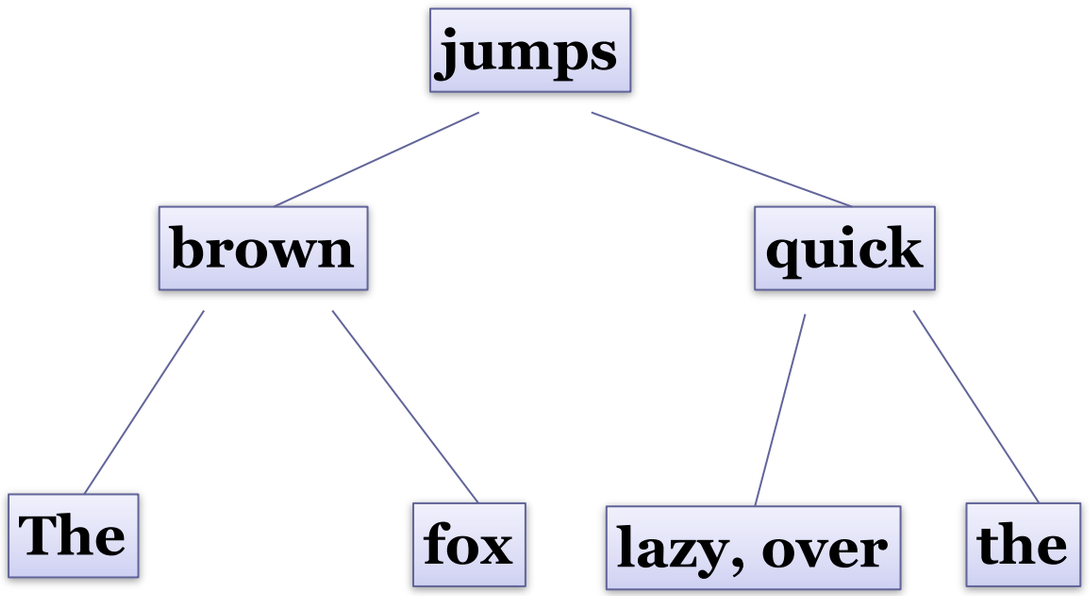
The quick brown fox jumps over the…



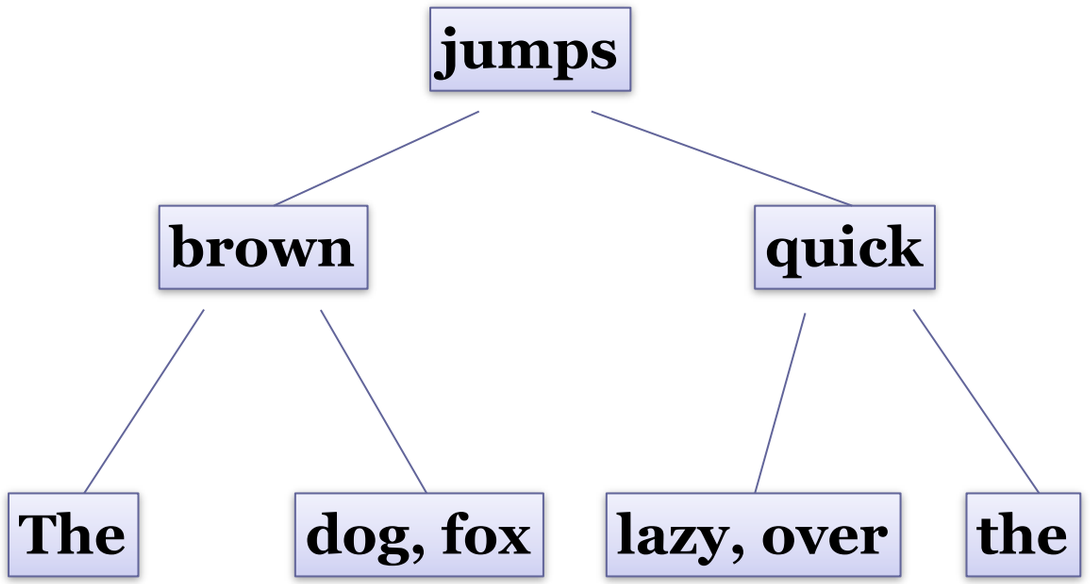
The quick brown fox jumps over the…



quick brown fox jumps over the lazy…



ick brown fox jumps over the lazy dog!



Analys av 2-3-träd

2-3-träd använder inte rotationer

- i motsats till AVL- och rödsvarta träd
- istället ”bubblar” noderna uppåt genom trädet

Komplexiteten:

- antal noder n som får plats i ett 2-3-träd med höjd h är mellan 2^h (om alla noder är 2-noder) och 3^h (om alla noder är 3-noder)
- alltså, höjden h ligger mellan $\log_3 n$ och $\log_2 n$
- söktiden är O(log n)

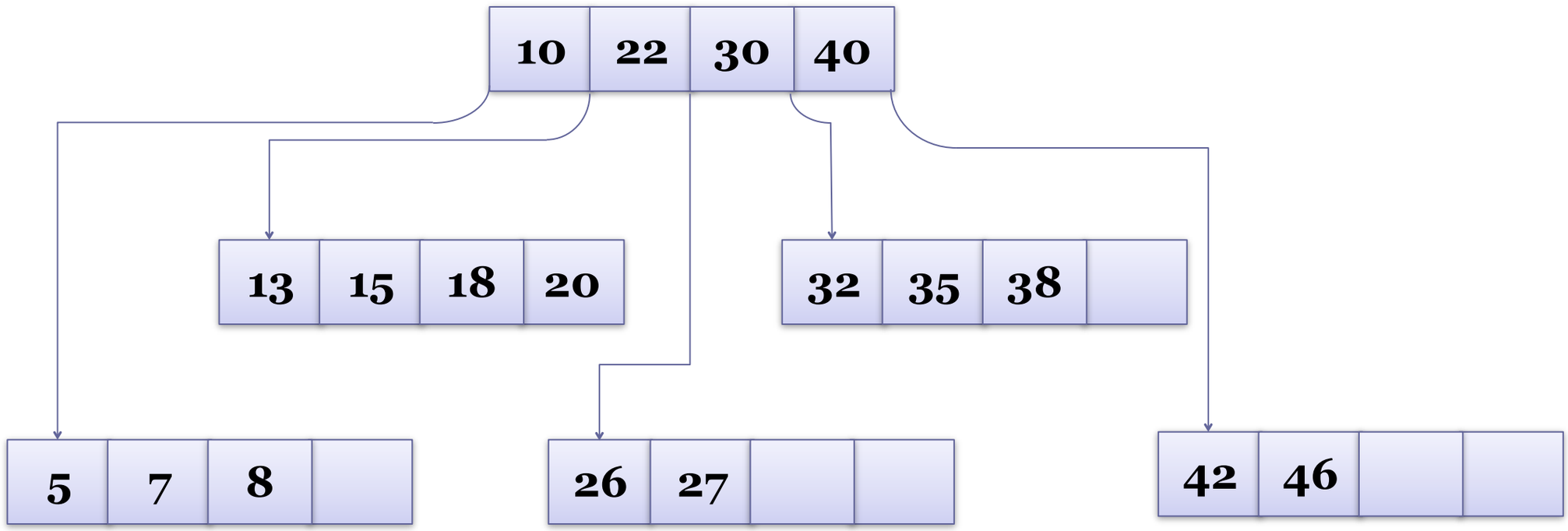
B-träd

B-träd är en generalisering av 2-3-träd:

- i ett B-träd av ordning *k*, kan varje nod ha upp till *k* barn

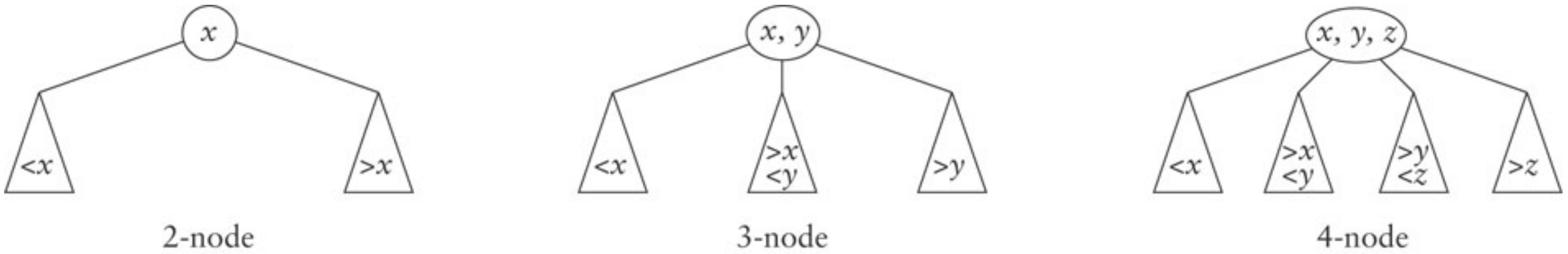
B-träd är designade för att lagra index till stora databaser:

- hårddiskars diskutrymme är uppdelade i block
- B-trädets ordning är precis sådan att en nod får plats i ett block
- (detta för att minimera antalet läsningar från hårddisken)

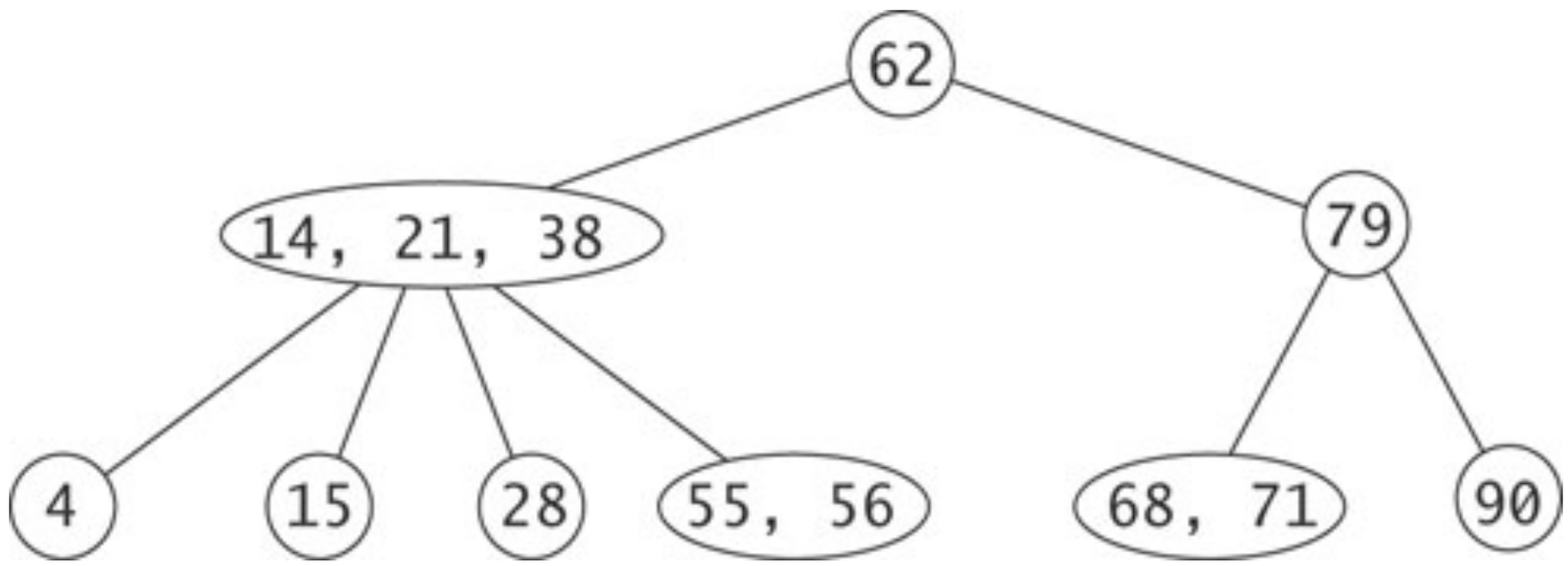


2-3-4-träd

2-3-4-träd är B-träd av ordning 4



Ett exempel på ett 2-3-4-träd:



2-3-4-träd ≡ rödsvarta träd

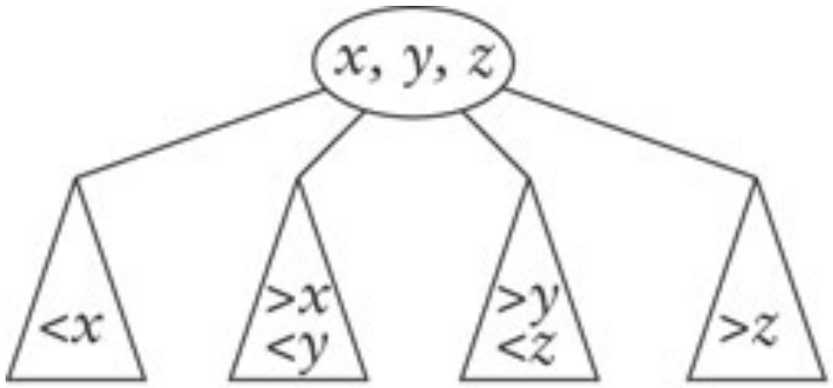
Ett rödsvart träd är ekvivalent med ett 2-3-4-träd:

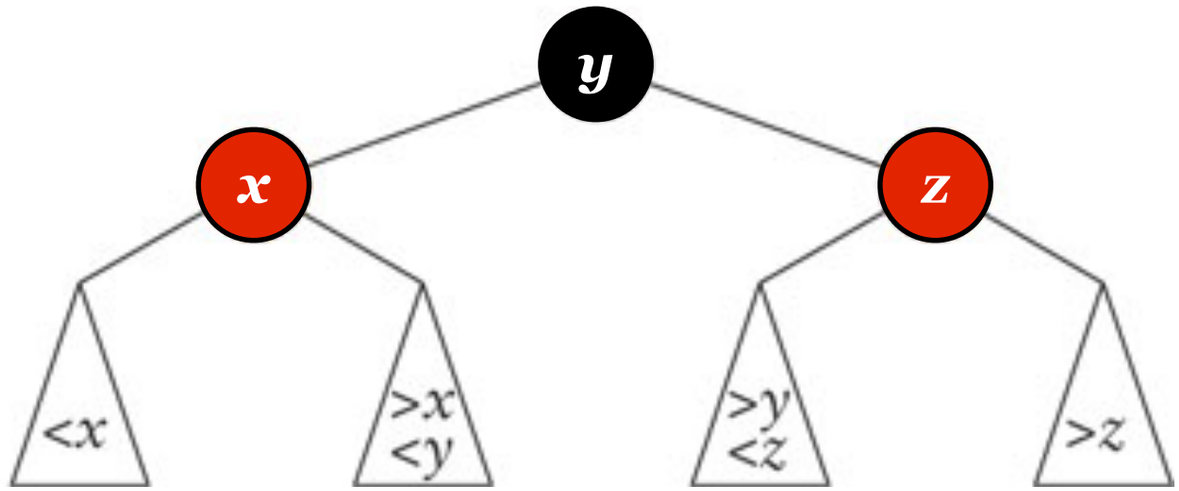
- en 2-nod är en svart nod

2-3-4-träd ≡ rödsvarta träd

Ett rödsvart träd är ekvivalent med ett 2-3-4-träd:

- en 4-nod är en svart nod med två röda barn

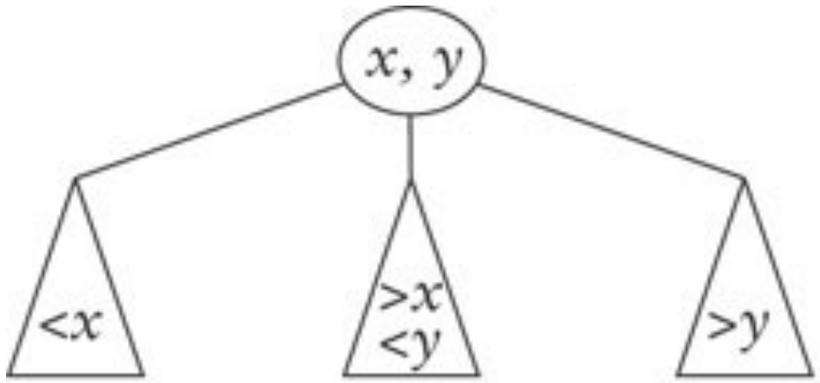


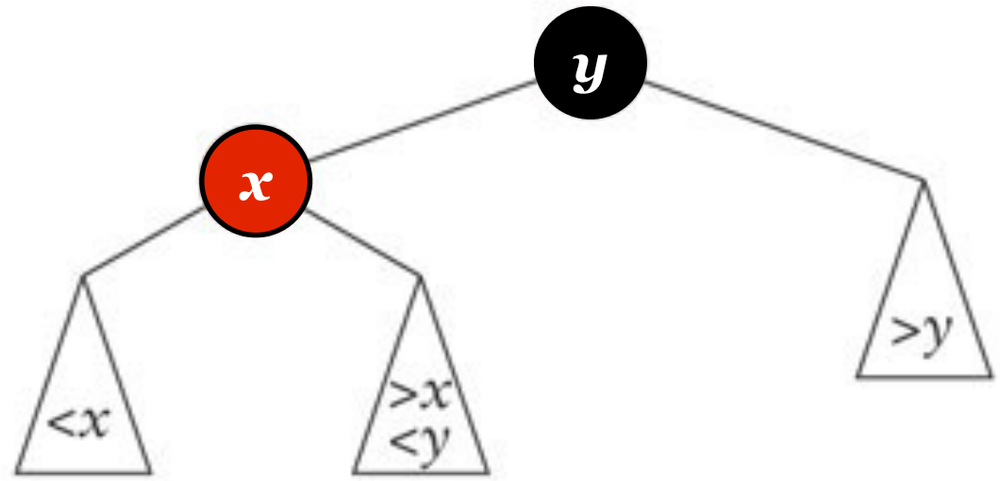


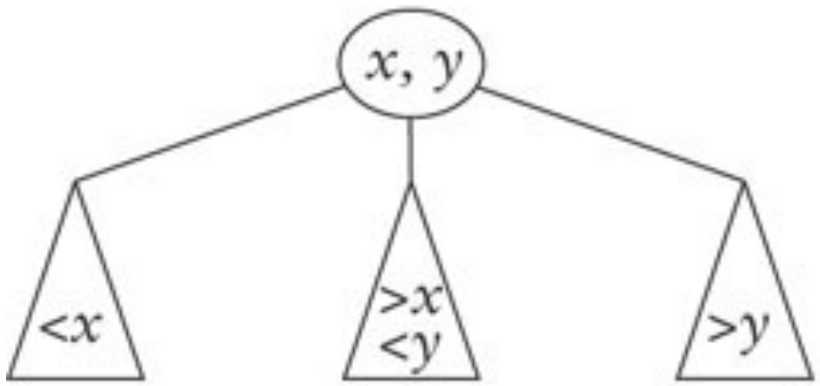
2-3-4-träd ≡ rödsvarta träd

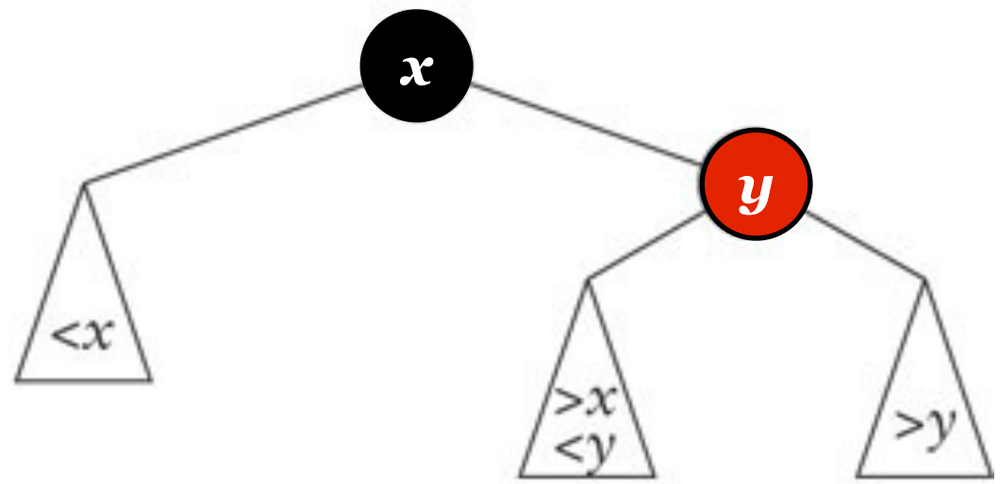
Ett rödsvart träd är ekvivalent med ett 2-3-4-träd:

- en 3-nod är en svart nod med ett rött barn









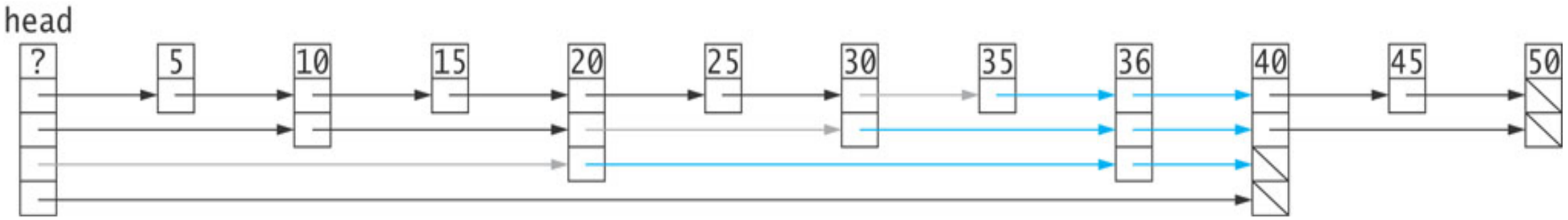
Skiplistor

En skiplista är en en lista med listor

- den första listan innehåller alla noder
- den andra listan innehåller (ungefär) hälften
- den tredje listan innehåller (ungefär) en fjärdedel

Varje nod i en skiplista har en nivå

- skiplistans nivå är den högsta nivån i listan
- nivån bestäms slumpmässigt så att det är 50% chans för nivå 1, 25% för nivå 2, 12,5% för nivå 3, etc.



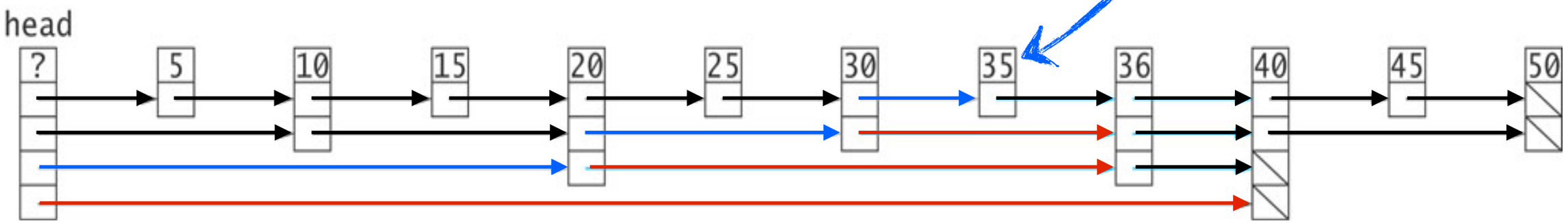
Att leta i en skiplista

1. **set** node **to** head
2. **set** m **to the highest level of** node
2. **while** m > 0**:**
3. **from** node**, find the largest level-**m **node** node‘ **such that** node’.data ≤ target
4. **set** node **to** node’
5. **if** node.data == target**, exit loop**
6. **decrease** m
7. **if** m > 0**, return** node
8. **otherwise, the** target **is not in the list**

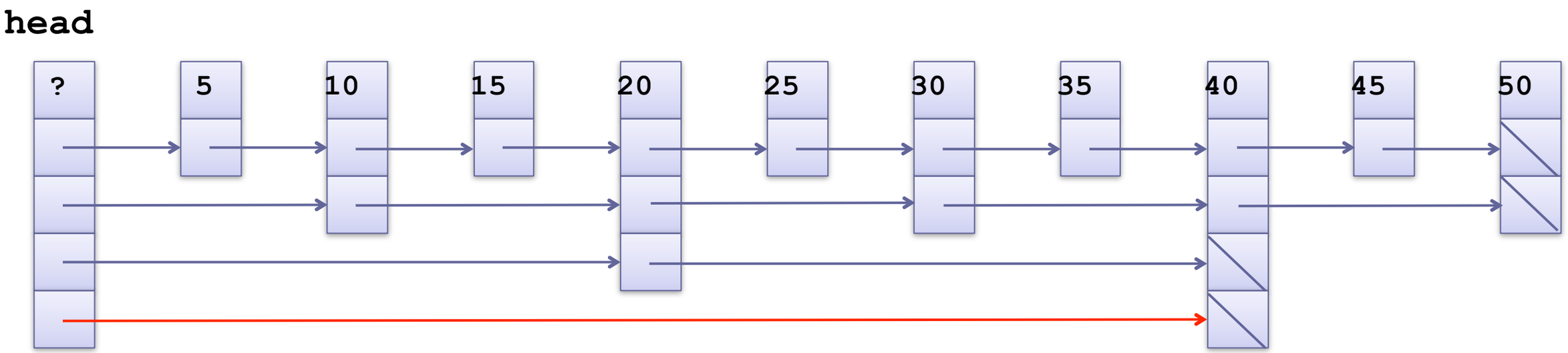
att leta efter **35**:

de röda pilarna måste vi följa för att se att vi har gått för långt

de blå pilarna leder oss sedan rätt



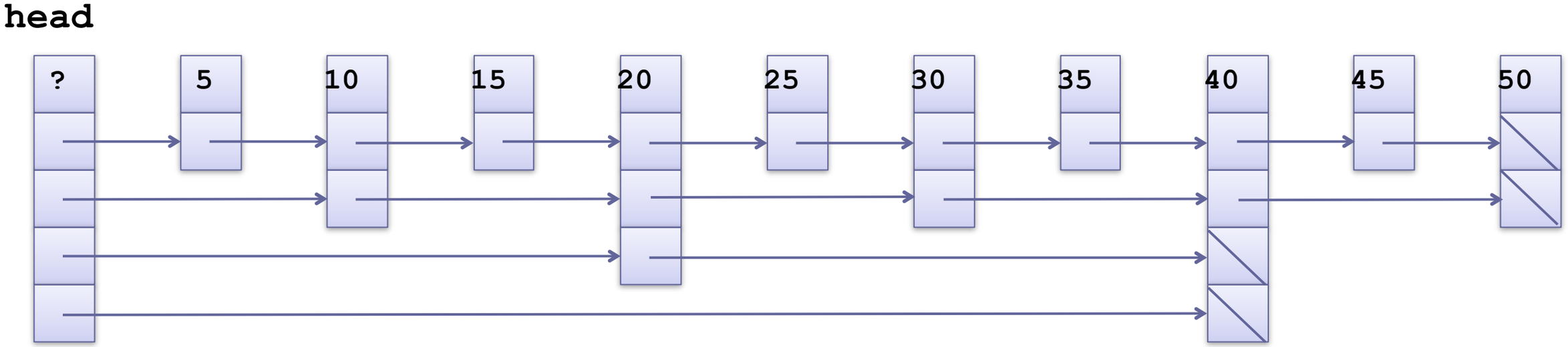
Att leta i en skiplista, exempel



A search always begins in the highest level list (the list with the fewest elements)

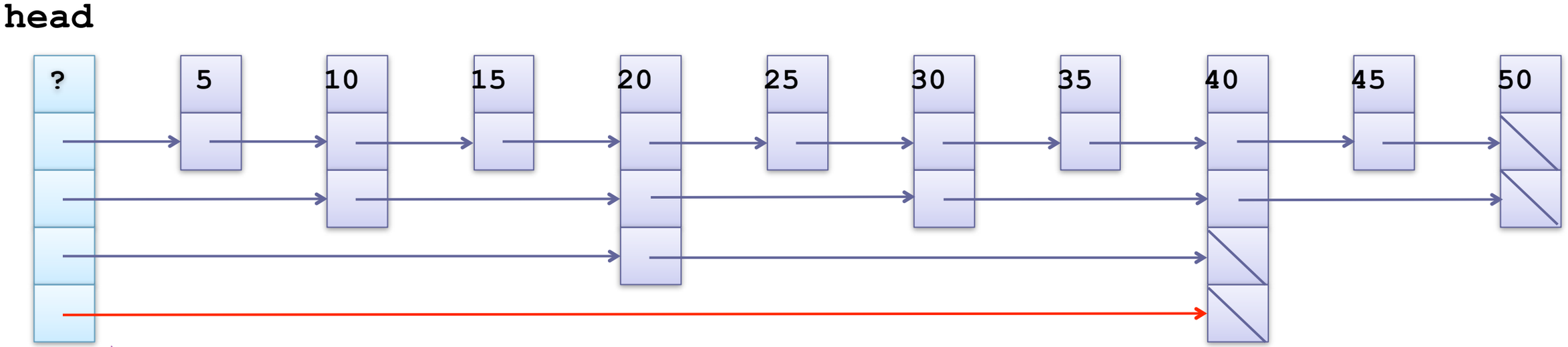
Att leta i en skiplista, exempel

Search for 35



Att leta i en skiplista, exempel

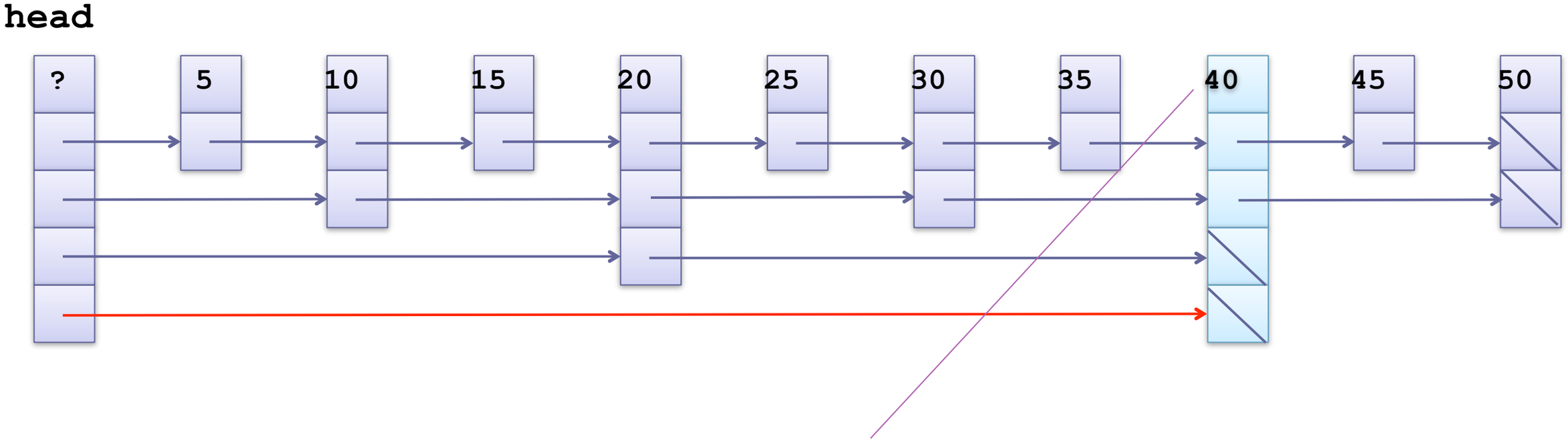
Search for 35



Start with the highest list, in this case, level 4

Att leta i en skiplista, exempel

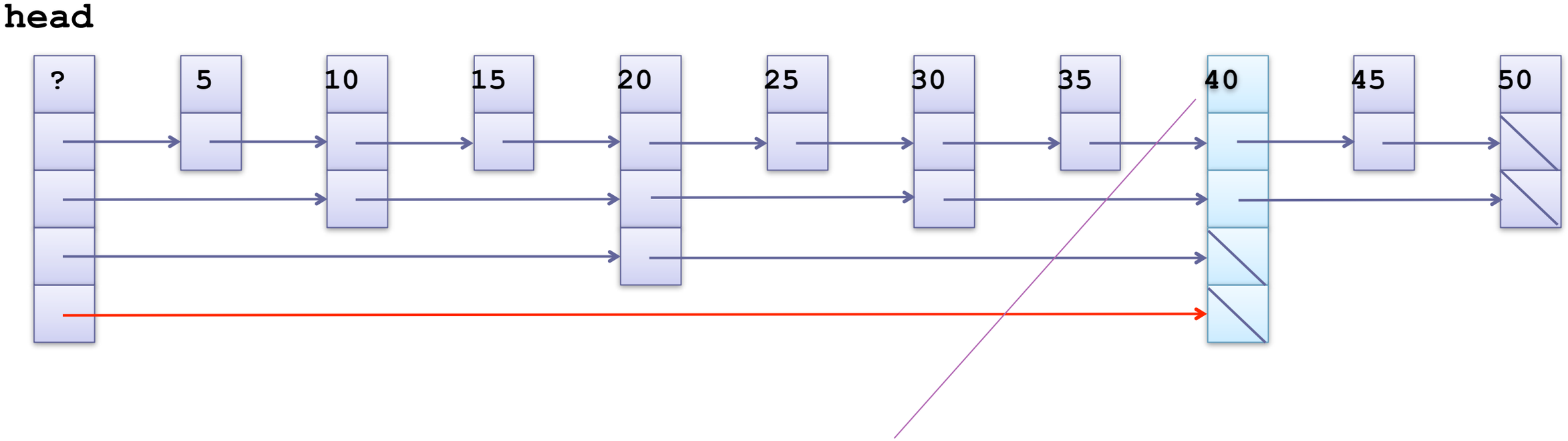
Search for 35



The value of this node is 40 (we have also reached the end of this list)

Att leta i en skiplista, exempel

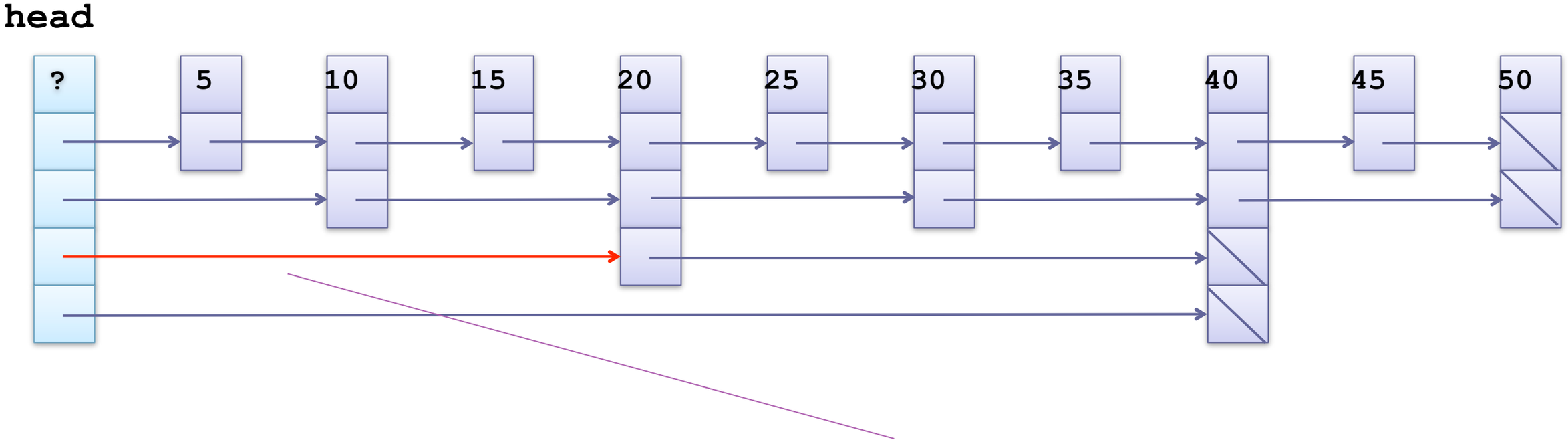
Search for 35



Since 35 < 40, we move back to the predecessor node and search the next lower level

Att leta i en skiplista, exempel

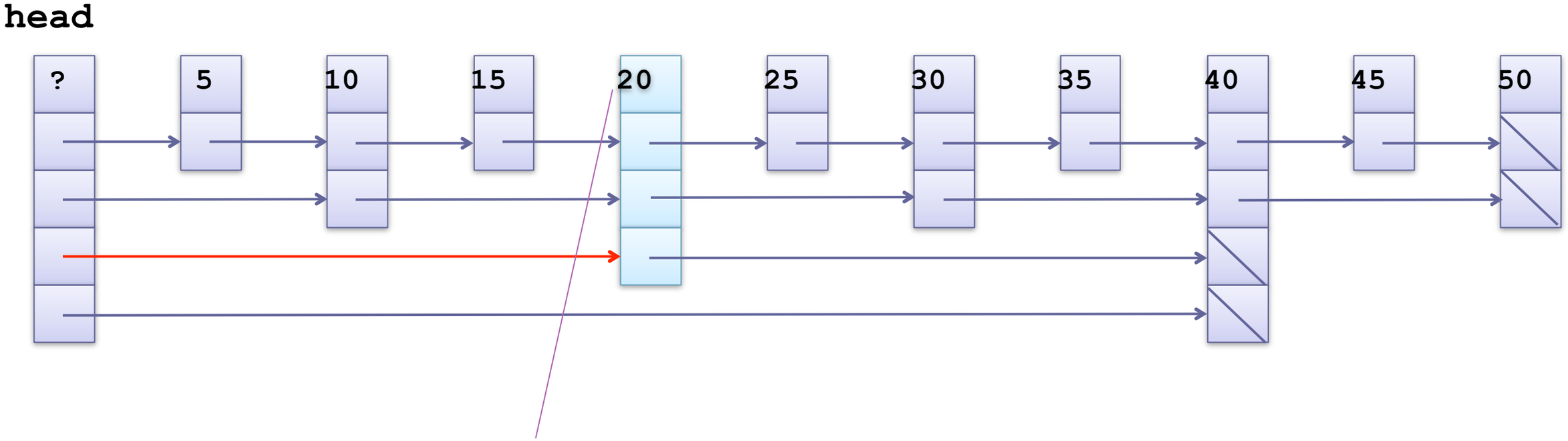
Search for 35



Since 35 < 40, we move back to the predecessor node and search the next lower level

Att leta i en skiplista, exempel

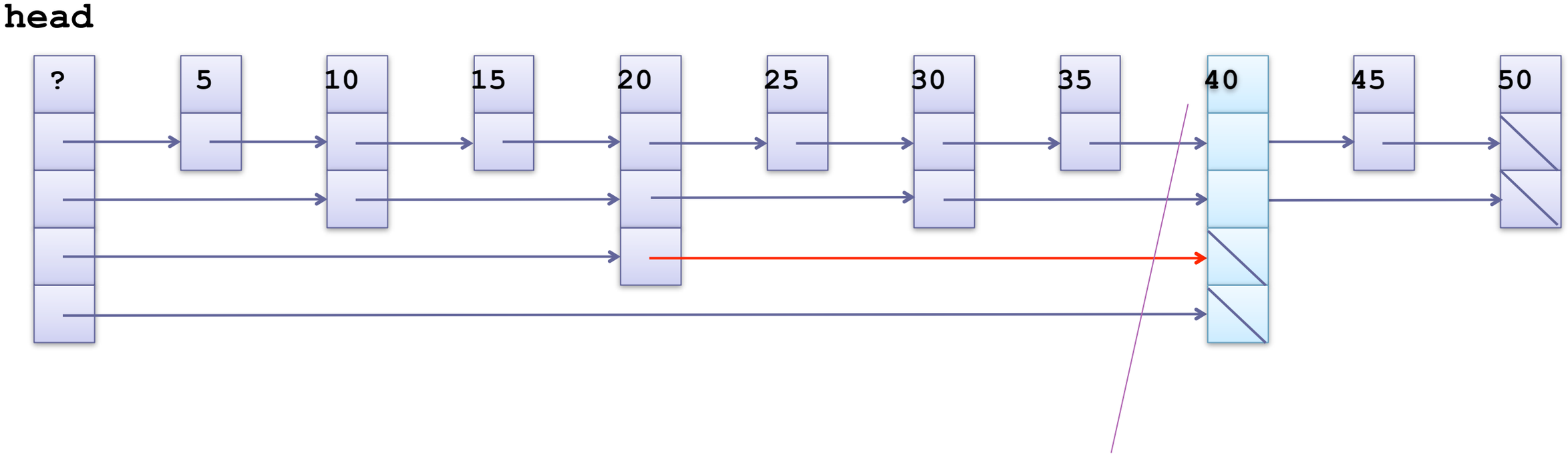
Search for 35



The value of the next node is 20. 35 > 20, so we move to the next node on this level

Att leta i en skiplista, exempel

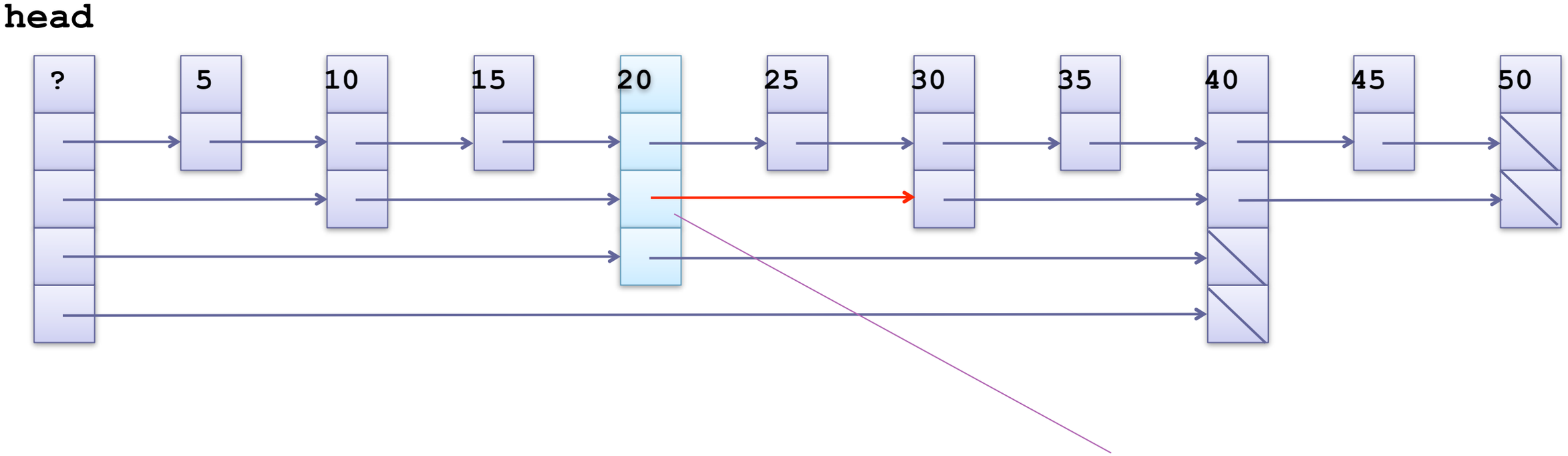
Search for 35



35 < 40 so we move to the predecessor and search the next lower level

Att leta i en skiplista, exempel

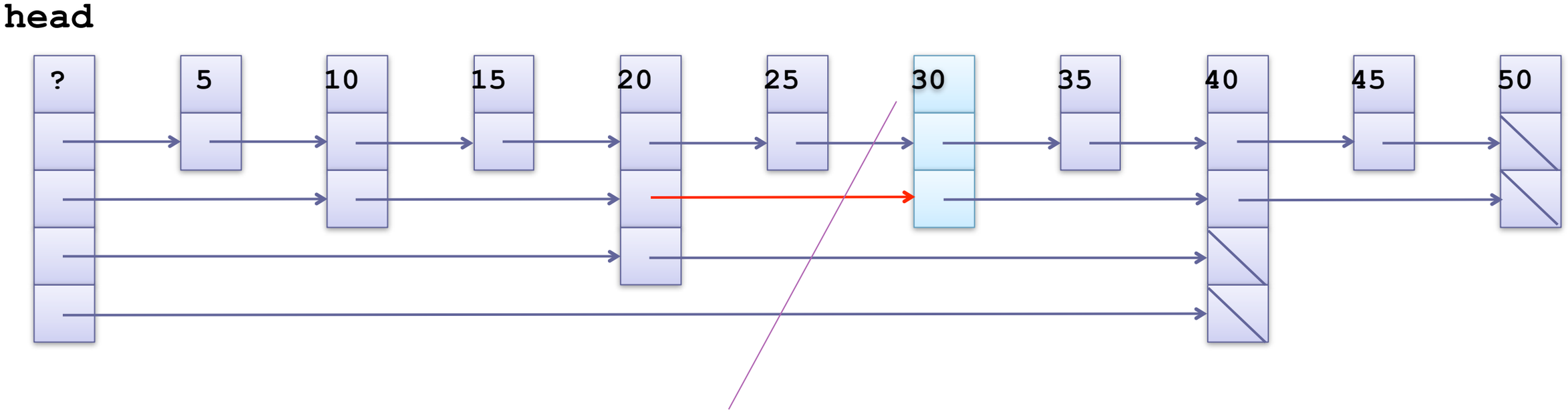
Search for 35



35 < 40 so we move to the predecessor and search the next lower level

Att leta i en skiplista, exempel

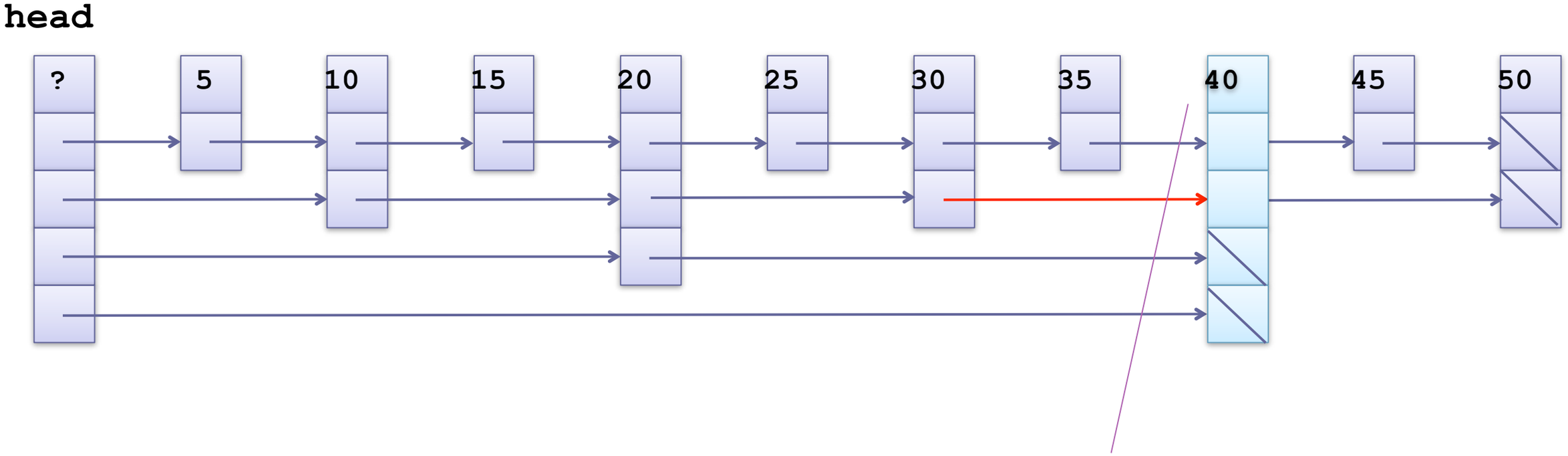
Search for 35



35 > 30, so we move to the next node

Att leta i en skiplista, exempel

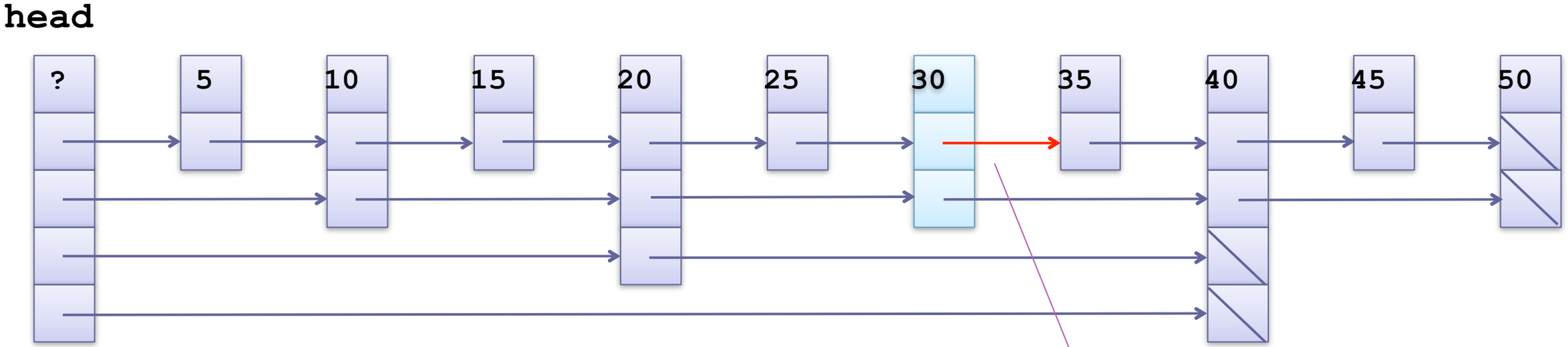
Search for 35



35 < 40, so we stop and move to the predecessor and search the next lower level

Att leta i en skiplista, exempel

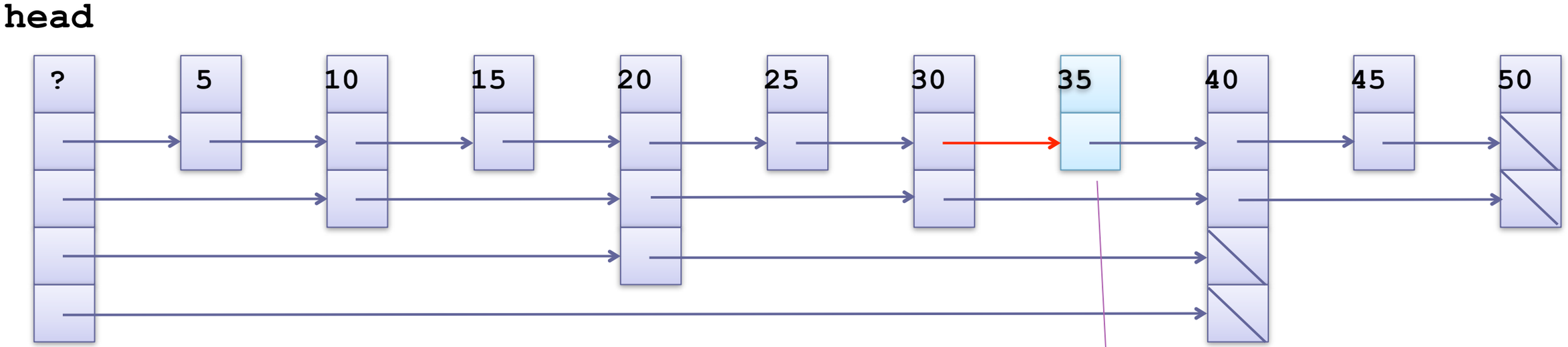
Search for 35



35 < 40, so we stop and move to the predecessor and search the next lower level

Att leta i en skiplista, exempel

Search for 35



35 is found!

Att lägga till i en skiplista

Om sökningen inte hittar målet, så hittar den iallafall dess föregångare i nivå-1-listan:

- det är efter den vi ska stoppa in det nya noden

Vi måste också veta vilken nivå den nya noden ska ha:

- nivån väljs slumpmässigt enligt en logaritmisk distribution: 50% chans för nivå 1, 25% för nivå 2, 12,5% för nivå 3, etc.

När vi sökte efter målet så gick vi igenom föregångaren i nivå-2-listan, och i nivå-3-listan, etc.

- då är det bara att peka om alla dessa föregångare till den nya noden

Effektivitet för en skiplista

Den första listan har ett konstant antal element
(k, som kanske är ca 2–3)

- den andra listan har dubbelt så många element, men vi har ju avgränsat sökrymden till halva antalet noder: alltså $k \cdot 2/2 = k$
- den tredje listan har i sin tur dubbel så många element, men nu har vi ju halverat sökrymden en gång till: alltså $k \cdot 4/4 = k$
- och så vidare…
- varje listsökning blir alltså konstant, O(1)

Den totala komplexiteten beror då på antalet listor,
dvs antalet nivåer:

- antalet nivåer är logaritmen av antalet noder n
- alltså är sökkomplexiteten logaritmisk, O(log n)
- (om slumptalsgenereringen är korrekt, förstås)