Cache-Oblivious B-Trees

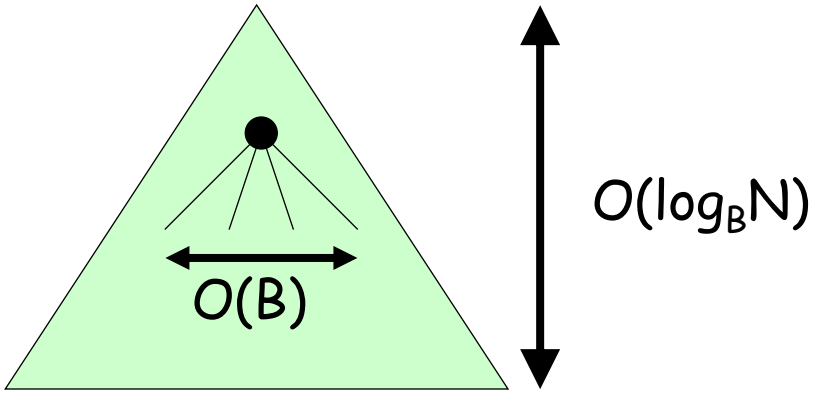
Michael A. Bender, Erik D. Demaine, Martin Farach-Colton.

Presented by: Itai Lahan

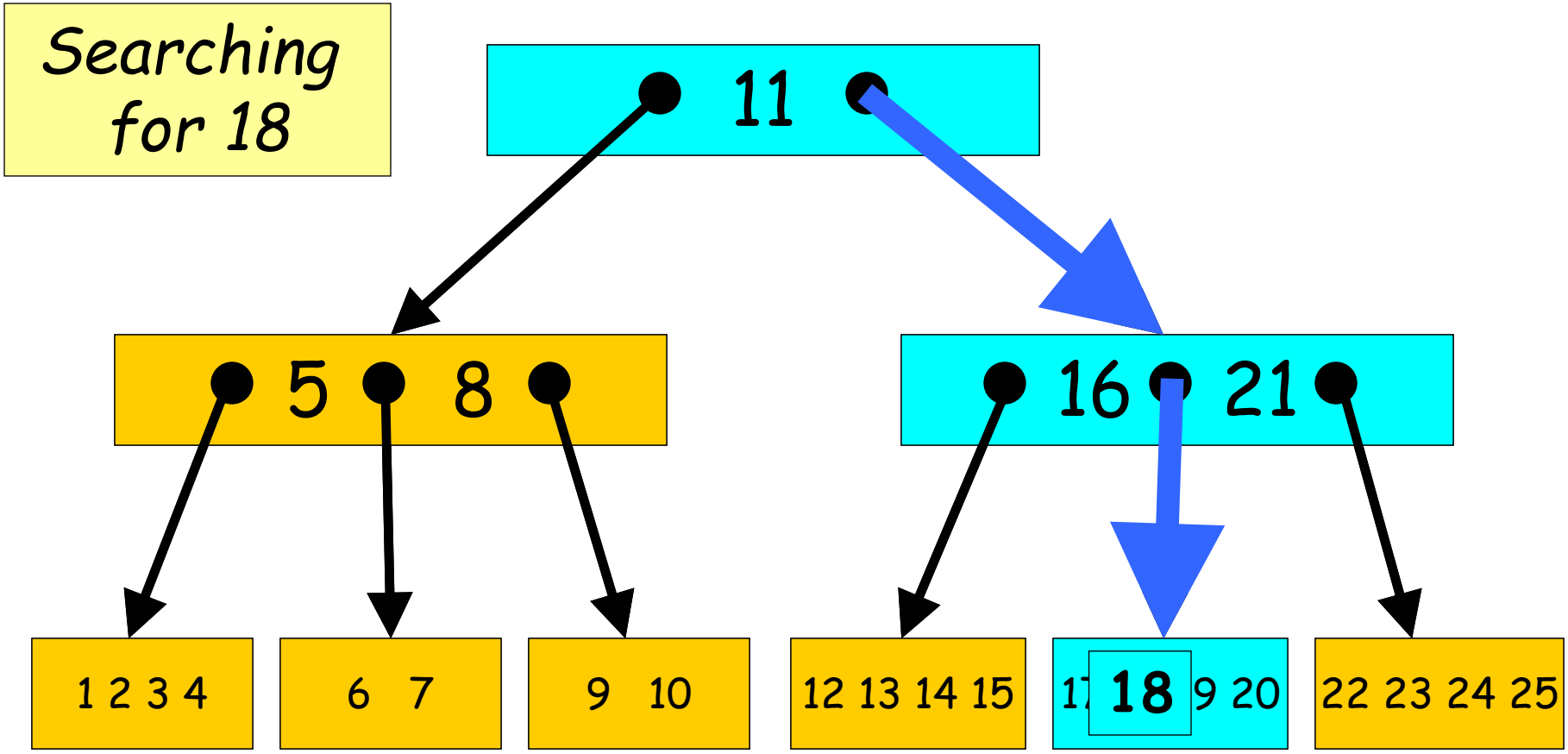
B-Trees – A reminder

[R. Bayer and E. M. McCreight. 1972]

- Balanced search tree
- All leaves have the same depth
- All nodes have degree at most B
  - The root has degree at least two
  - All internal nodes have degree at least B/2



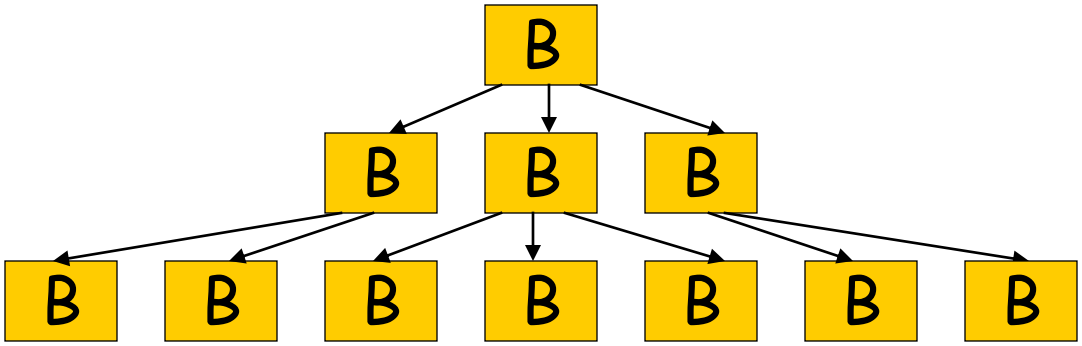
B-Trees - example



B-Trees

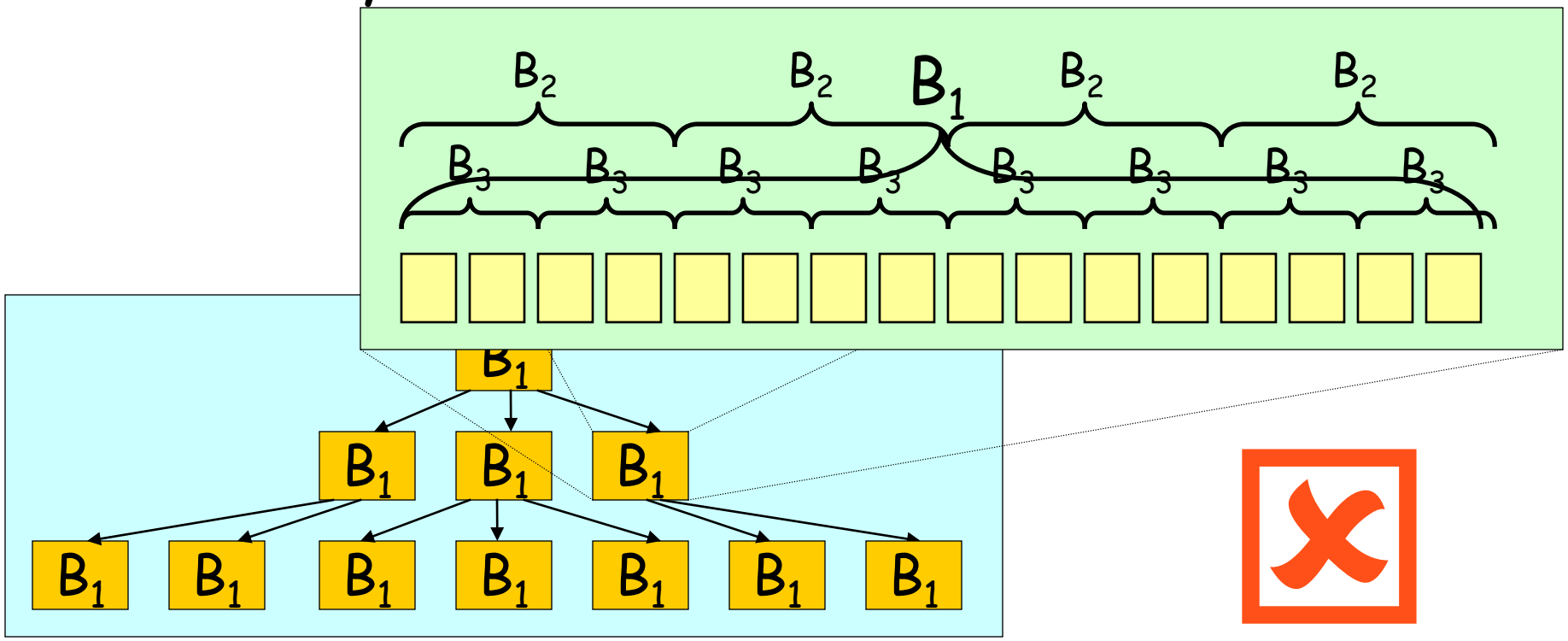
[R. Bayer and E. M. McCreight. 1972]

- Balanced search tree
- Fan-out is proportional to memory block size
- A single block read determines the next node
- Classical two-level memory model solution



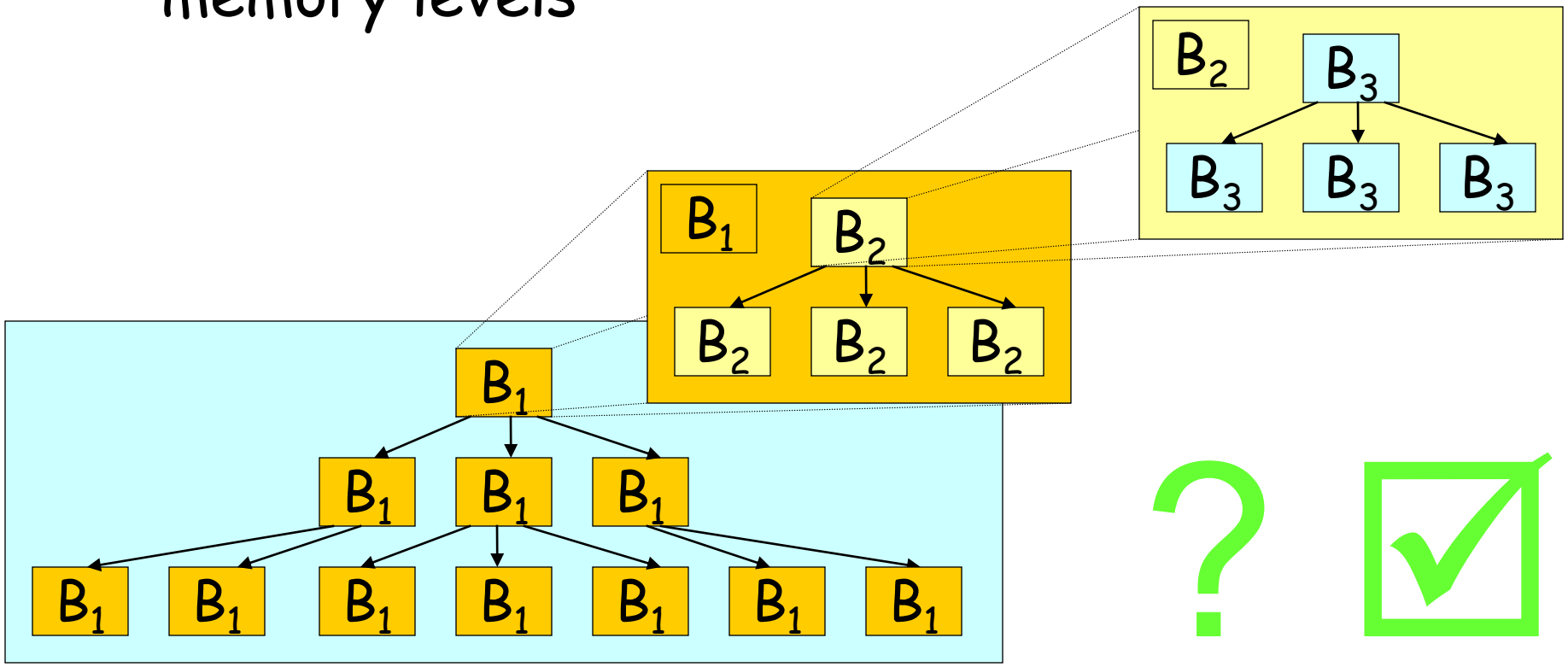
B-Trees in multilevel memory hierarchy ?

- $B_1 > B_2 > B_3 > \ldots > B_k$ block sizes between memory levels



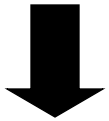
B-Trees in multilevel memory hierarchy ?

- $B_1 > B_2 > B_3 > \ldots > B_k$ block sizes between memory levels

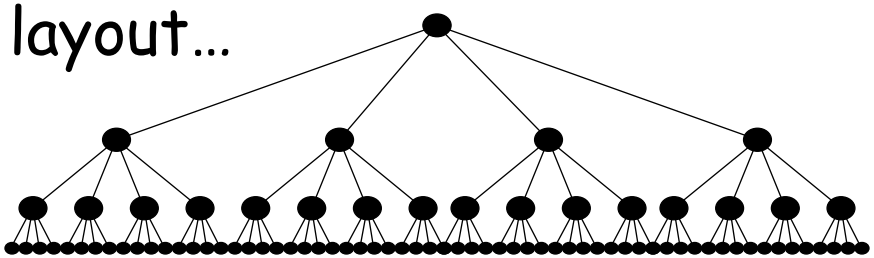


Cache Oblivious Algorithms

- No variable depends on hardware parameters
- Reason about a simple two level memory

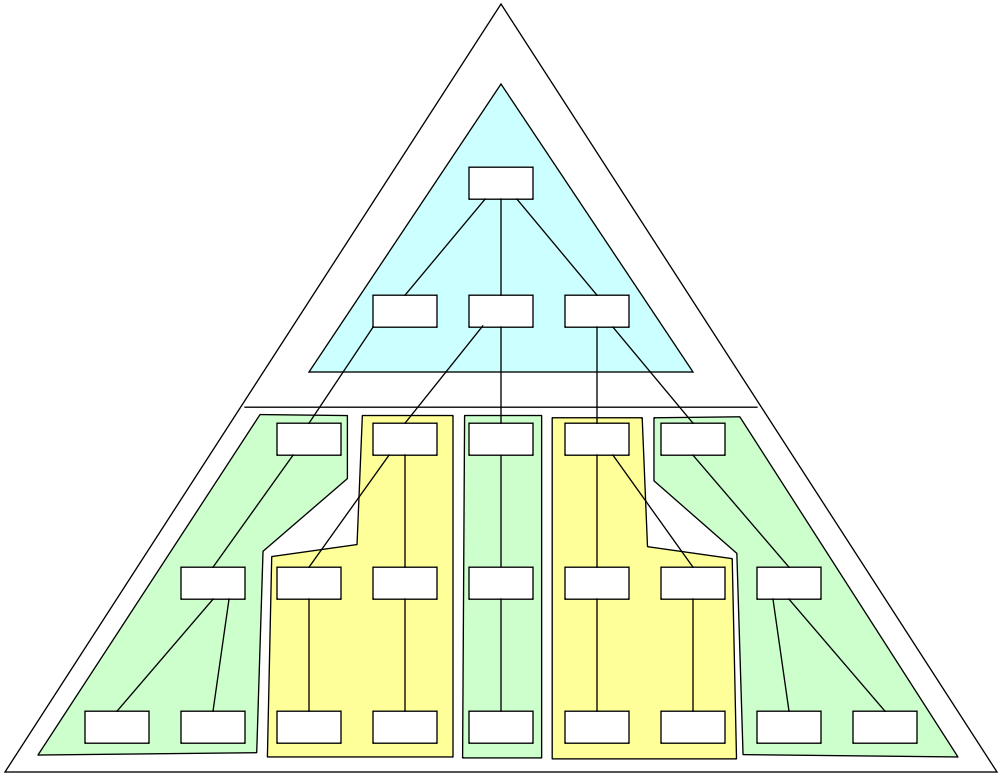


- Prove results about an unknown multilevel memory
- Cache oblivious B-Tree ?
  - How do we achieve $O(\log_B N)$ without knowing B ??
  - It's all about the memory layout…



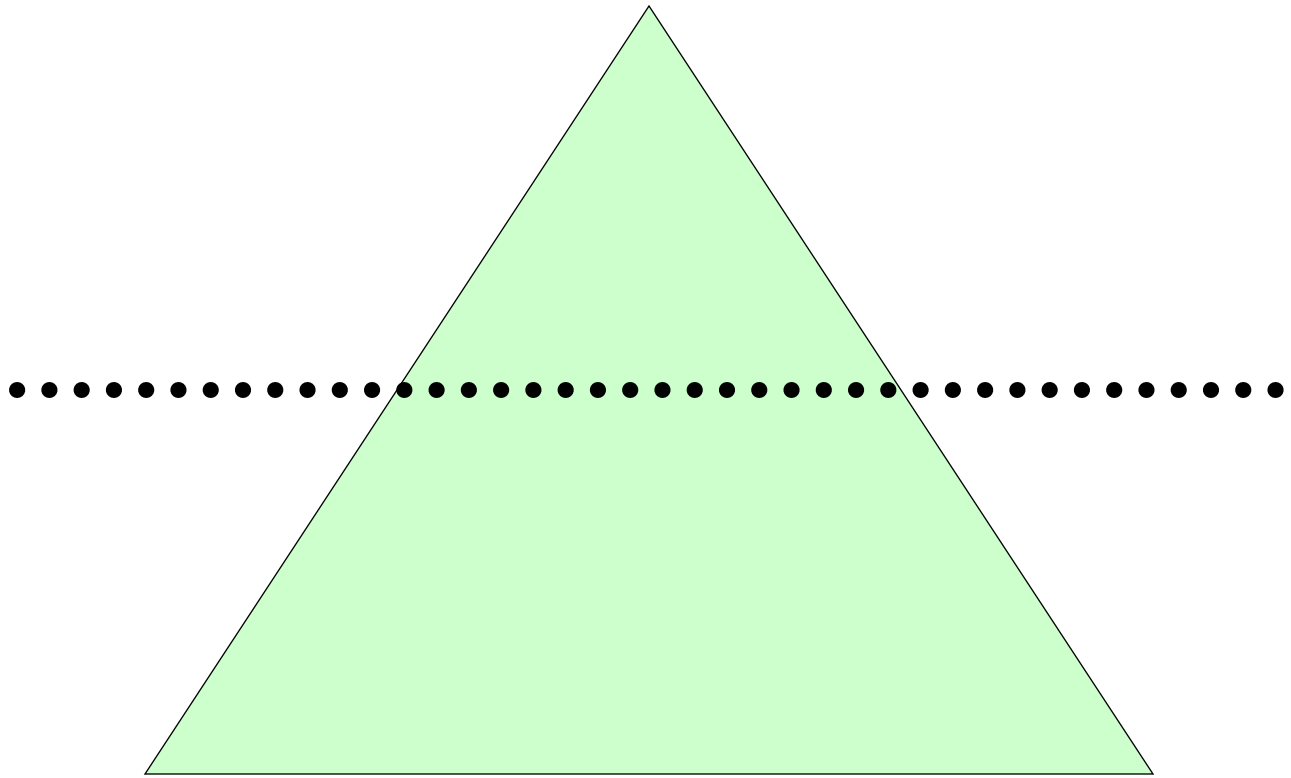
Van Emde Boas Layout

[P. van Emde Boas, R. Kaas, and E. Zijlstra. 1977]



Van Emde Boas Layout

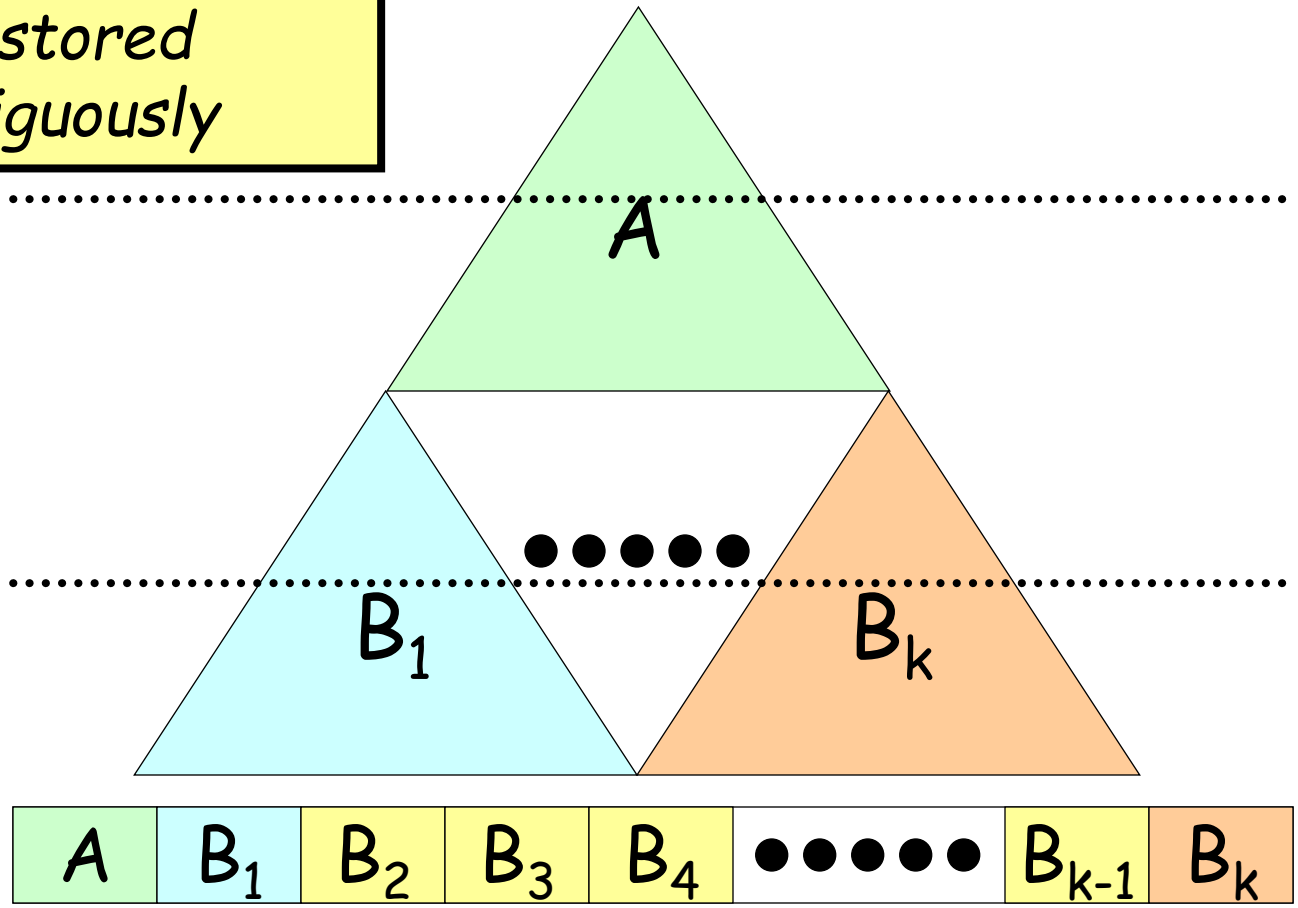
[P. van Emde Boas, R. Kaas, and E. Zijlstra. 1977]



Van Emde Boas Layout

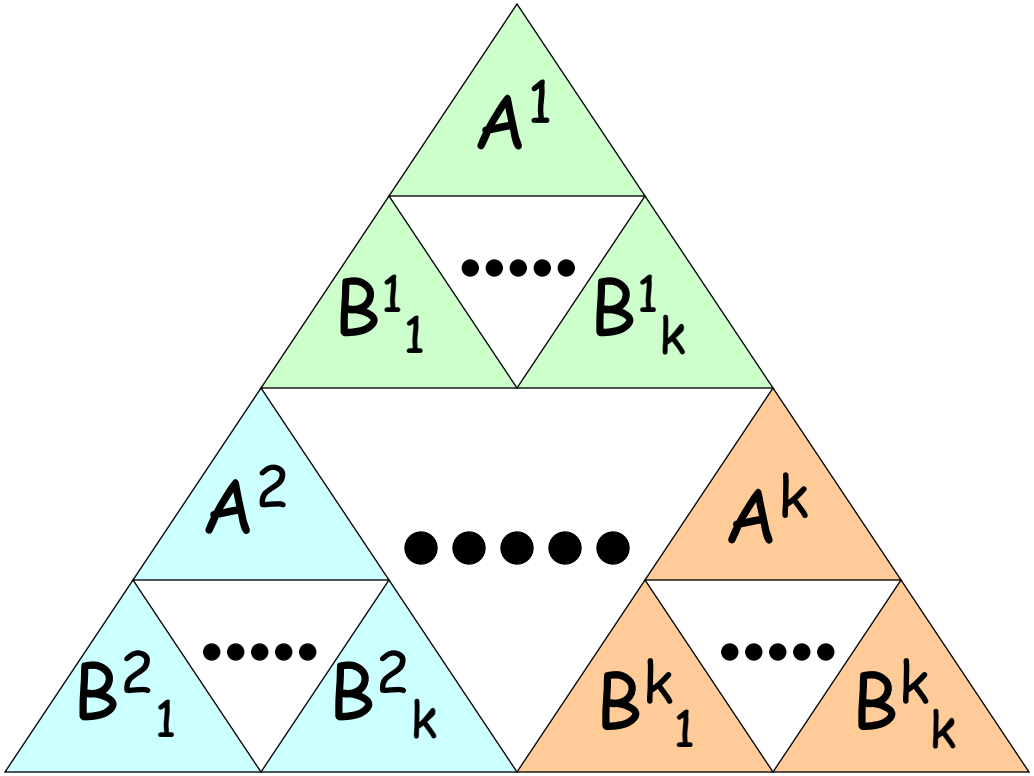
Recursive structures are stored contiguously

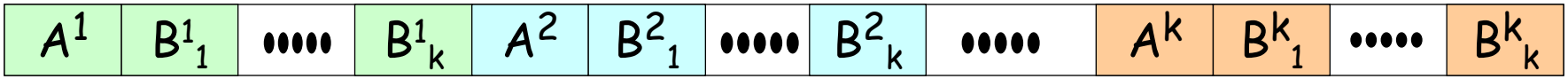
[P. van Emde Boas, R. Kaas, and E. Zijlstra. 1977]



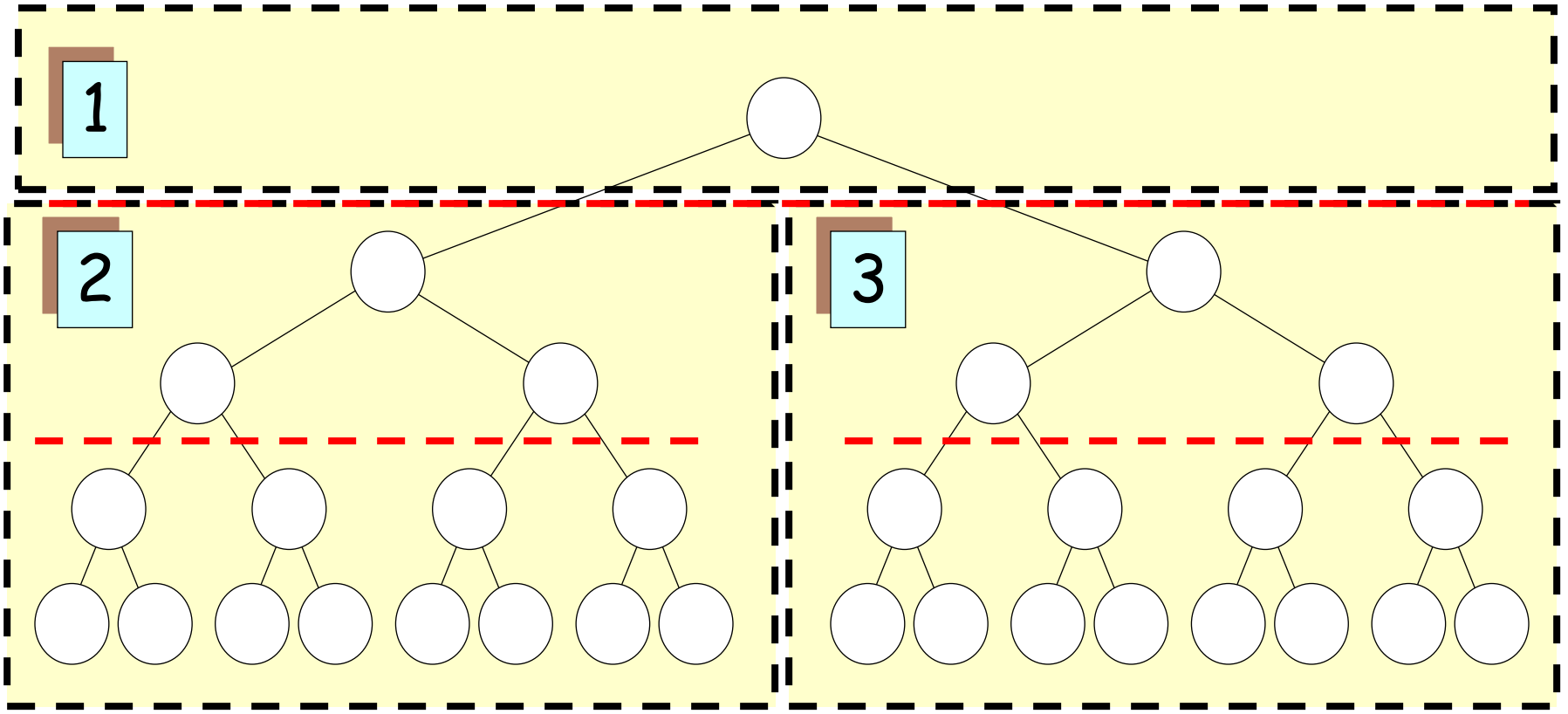
Van Emde Boas Layout

[P. van Emde Boas, R. Kaas, and E. Zijlstra. 1977]

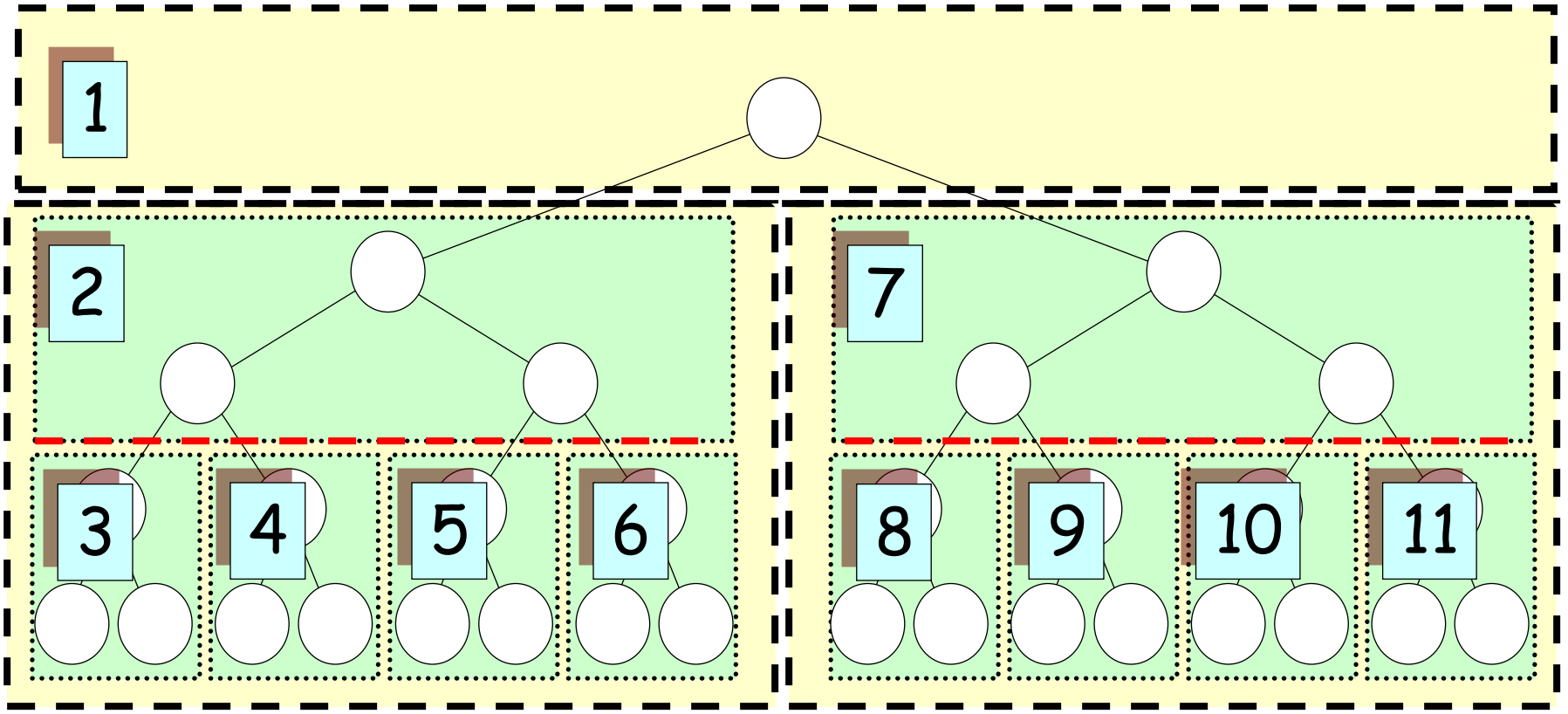




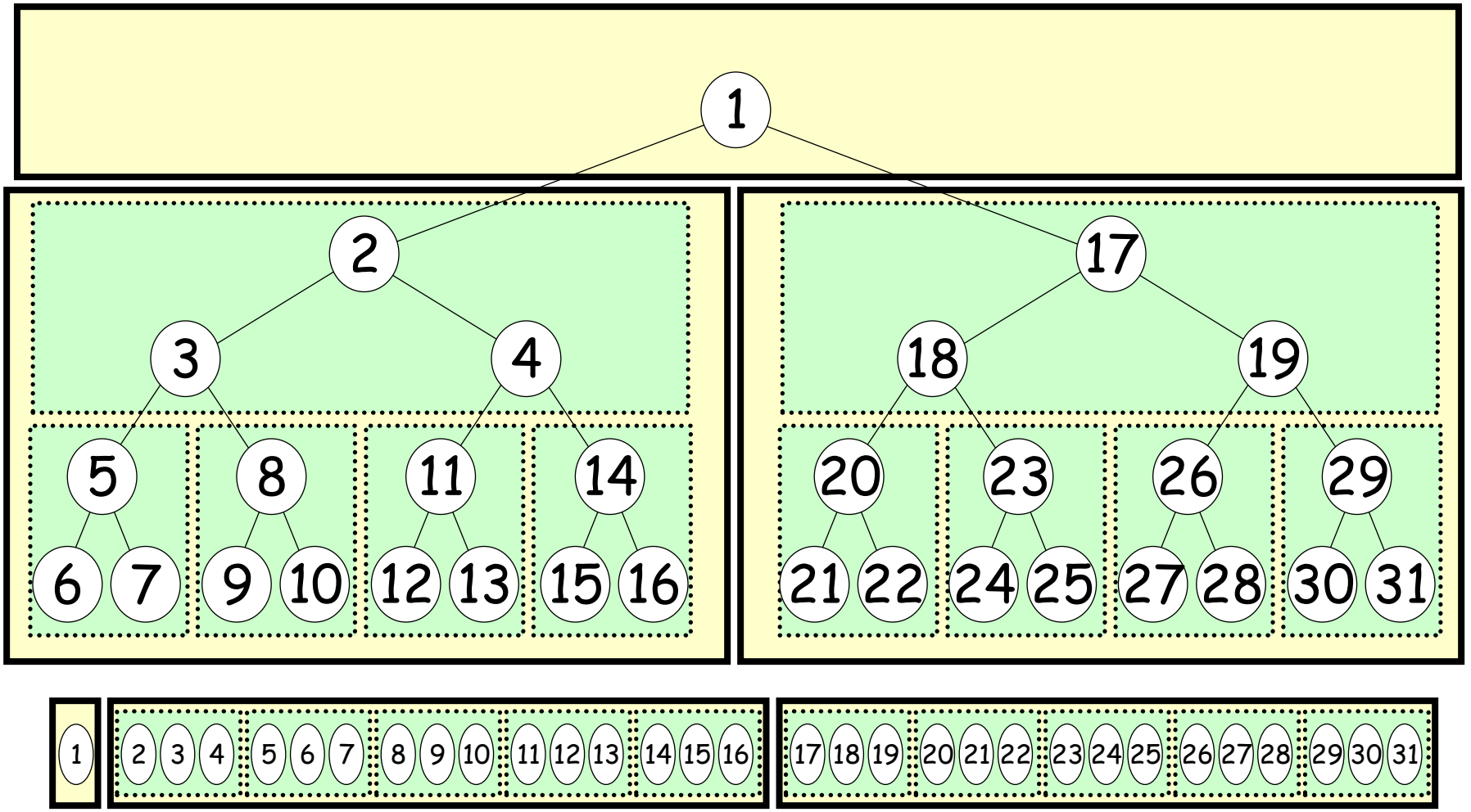
Van Emde Boas Layout



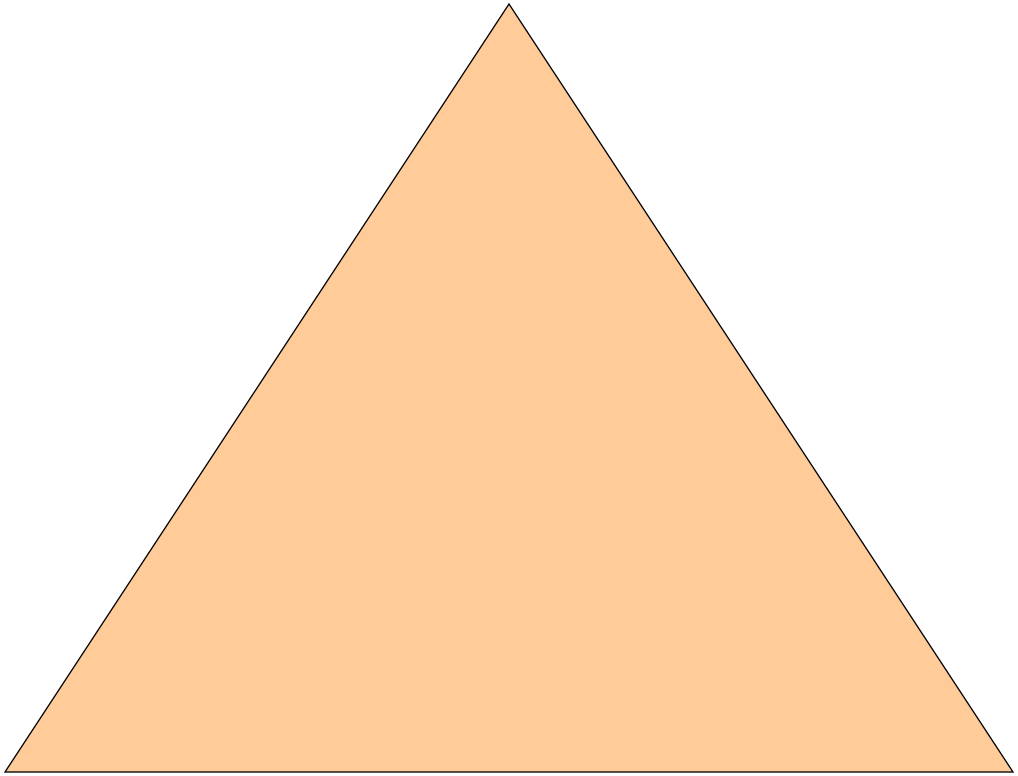
Van Emde Boas Layout



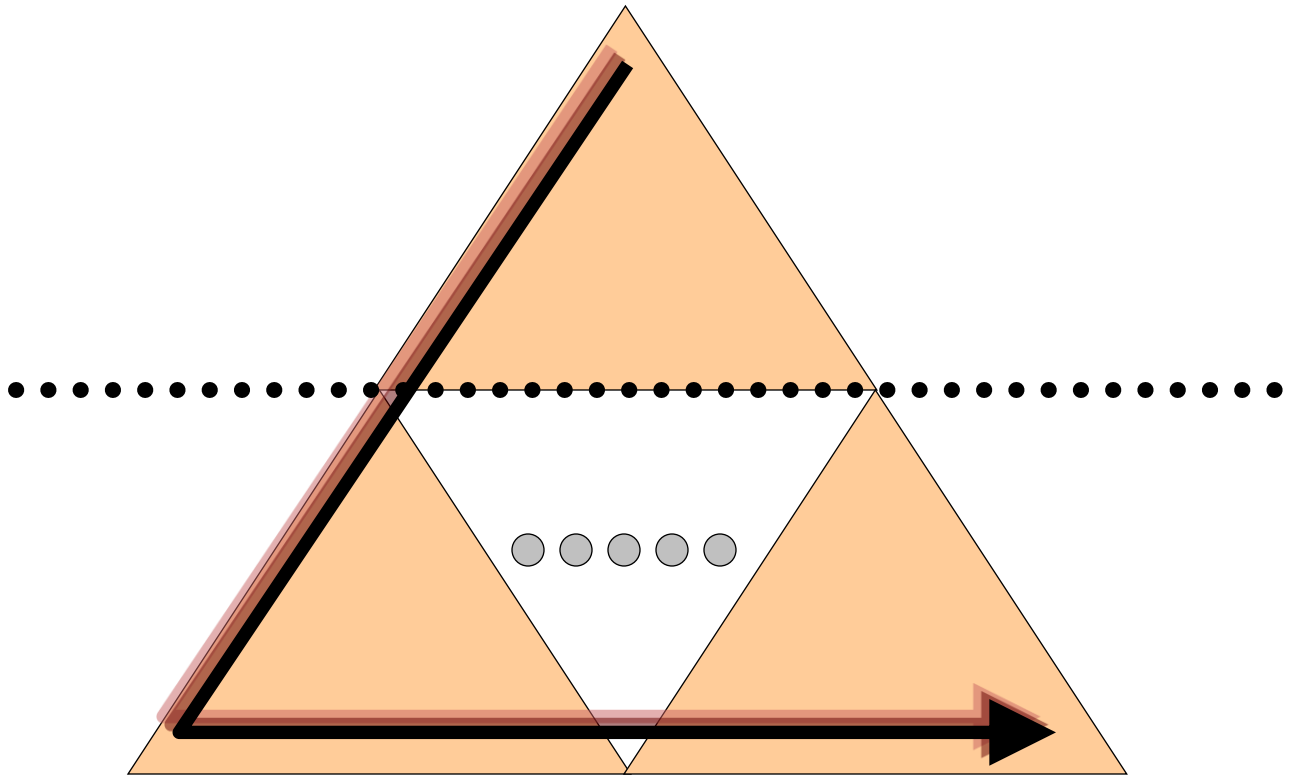
Van Emde Boas Layout



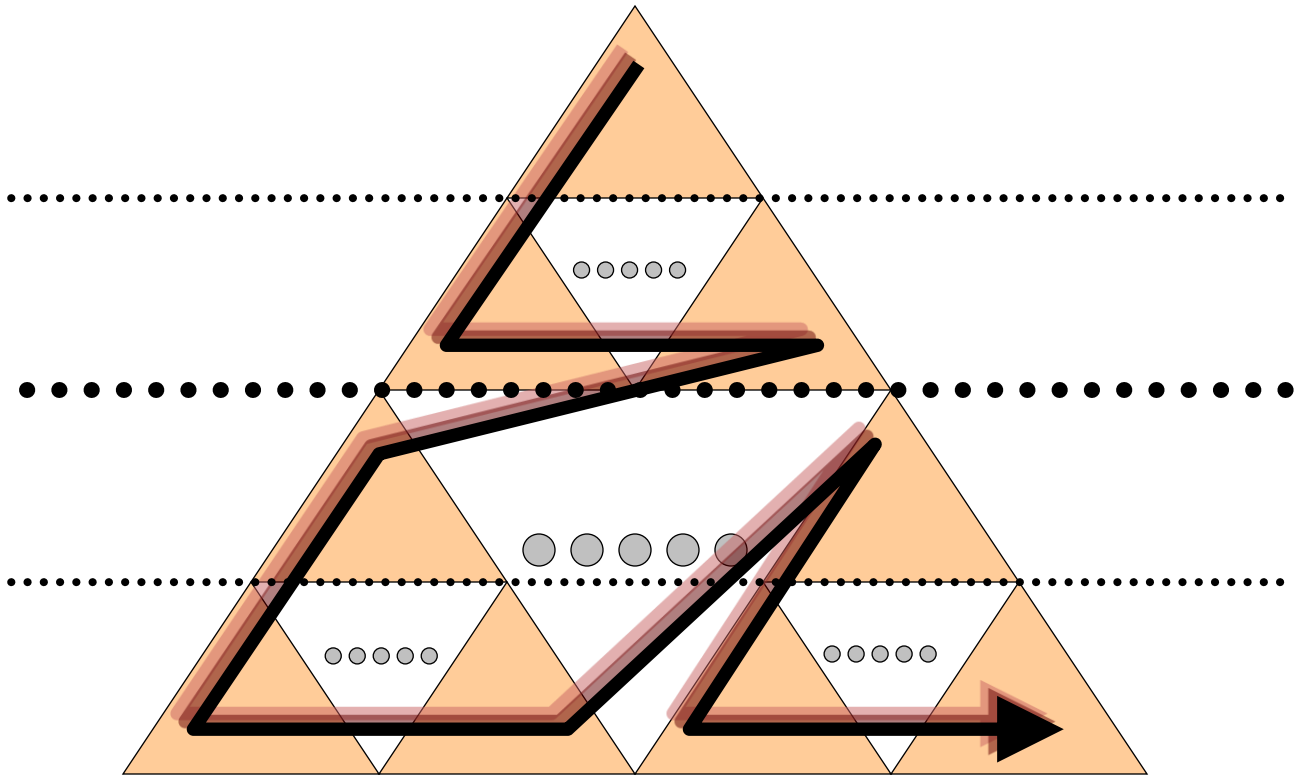
Van Emde Boas Layout



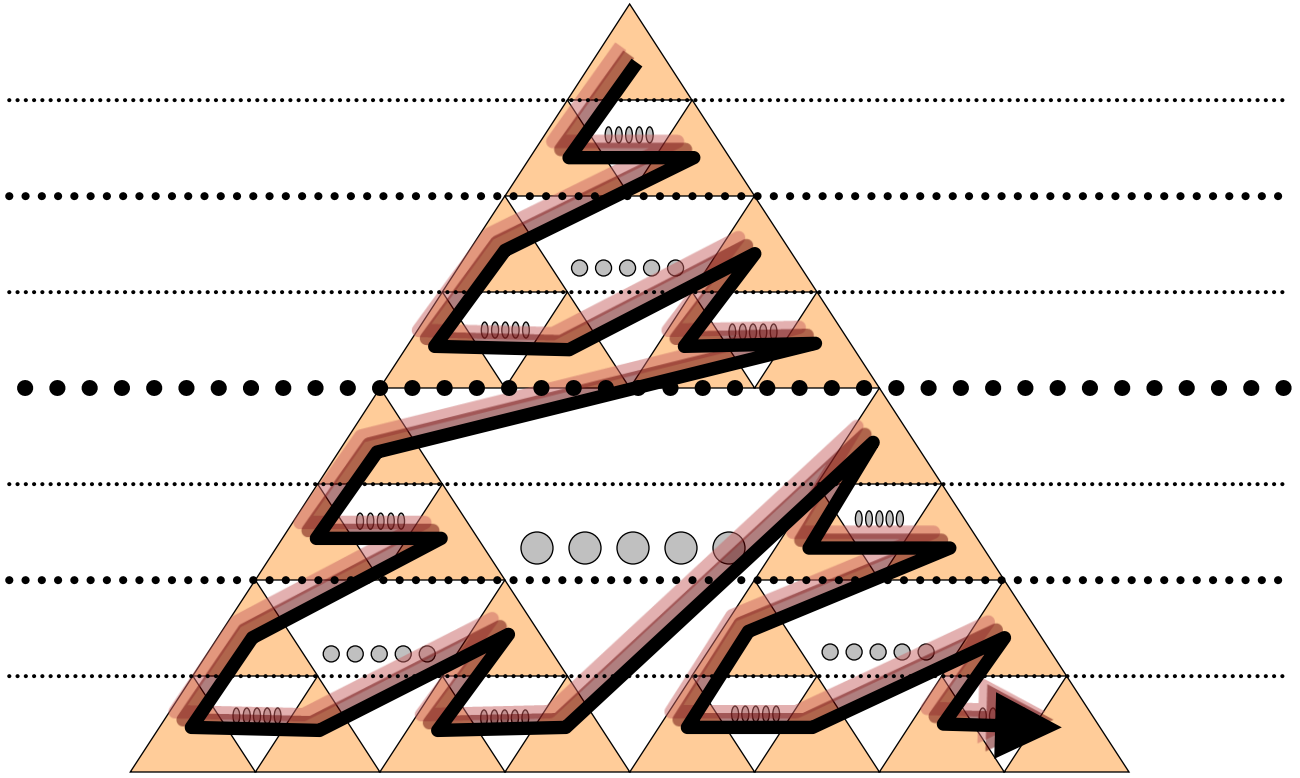
Van Emde Boas Layout



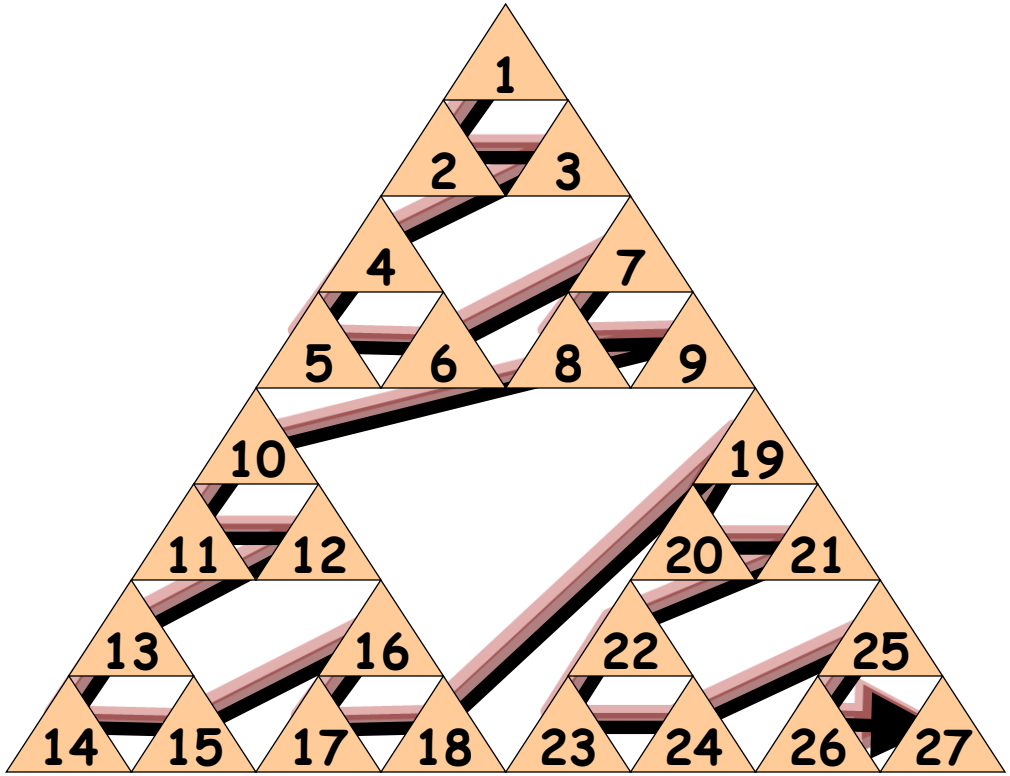
Van Emde Boas Layout



Van Emde Boas Layout



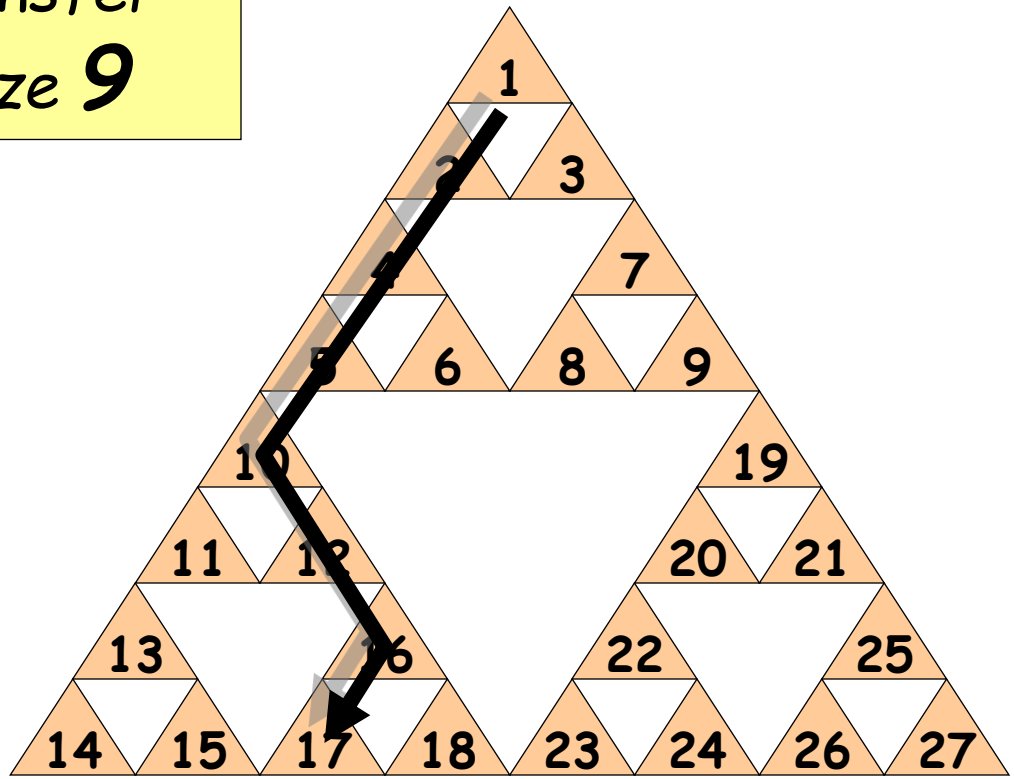
Van Emde Boas Layout



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

Van Emde Boas Layout

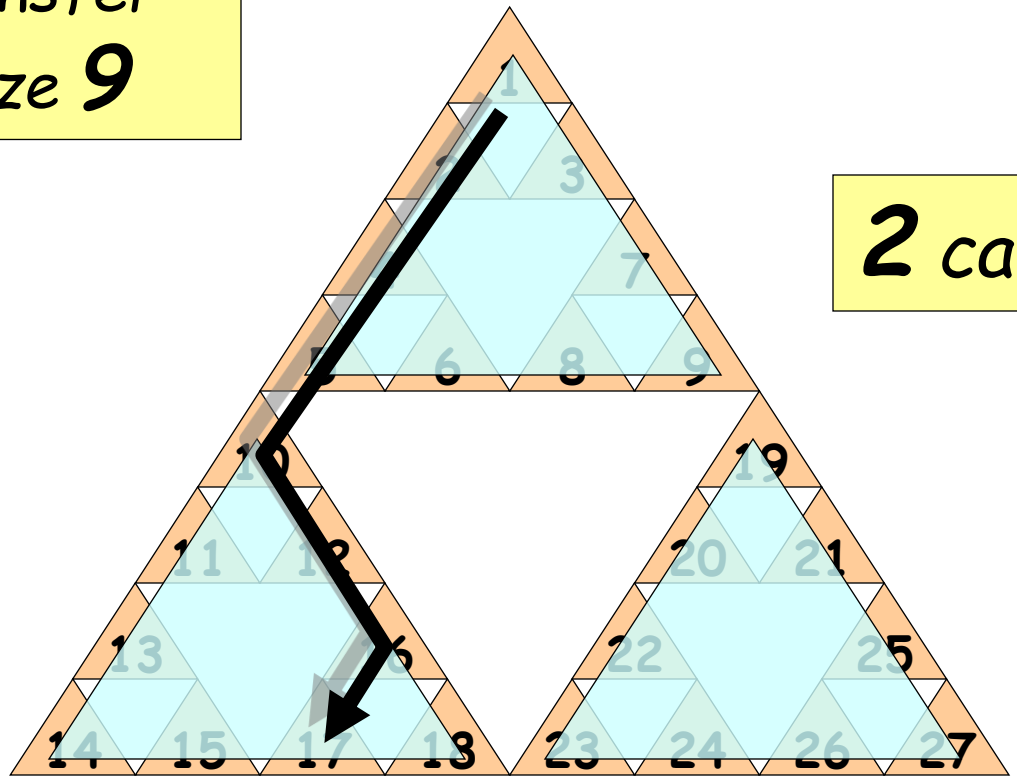
Searching with memory transfer block of size **9**



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

Van Emde Boas Layout

Searching with memory transfer block of size **9**

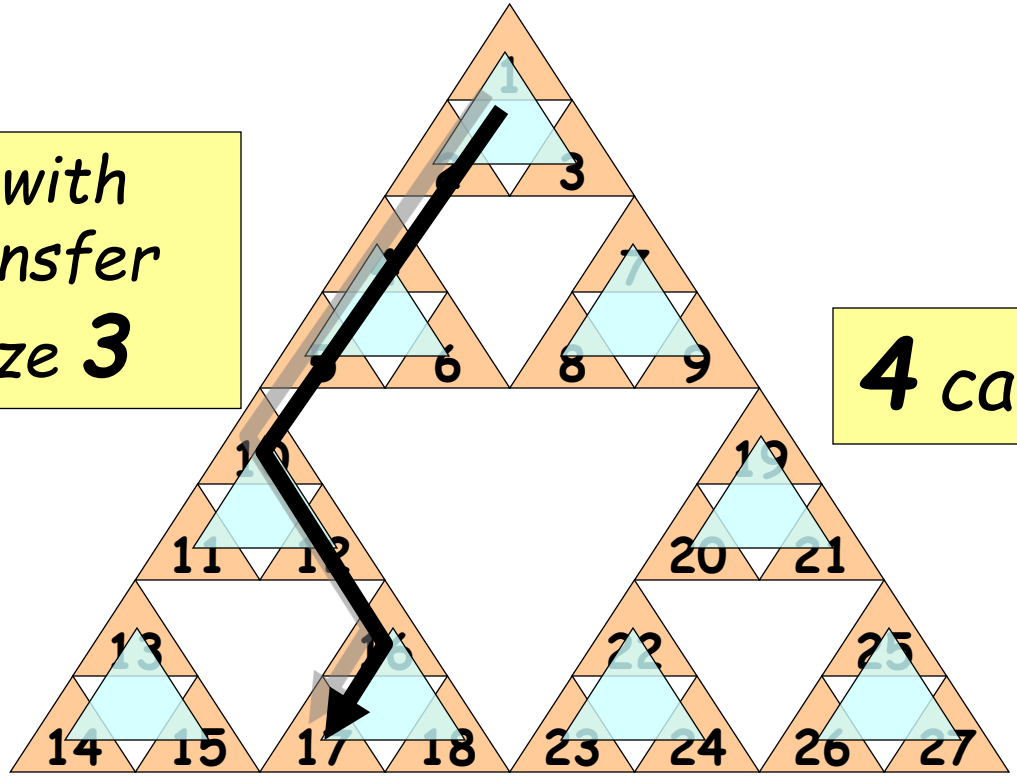


2 *cache misses*

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

Van Emde Boas Layout

Searching with memory transfer block of size ***3***



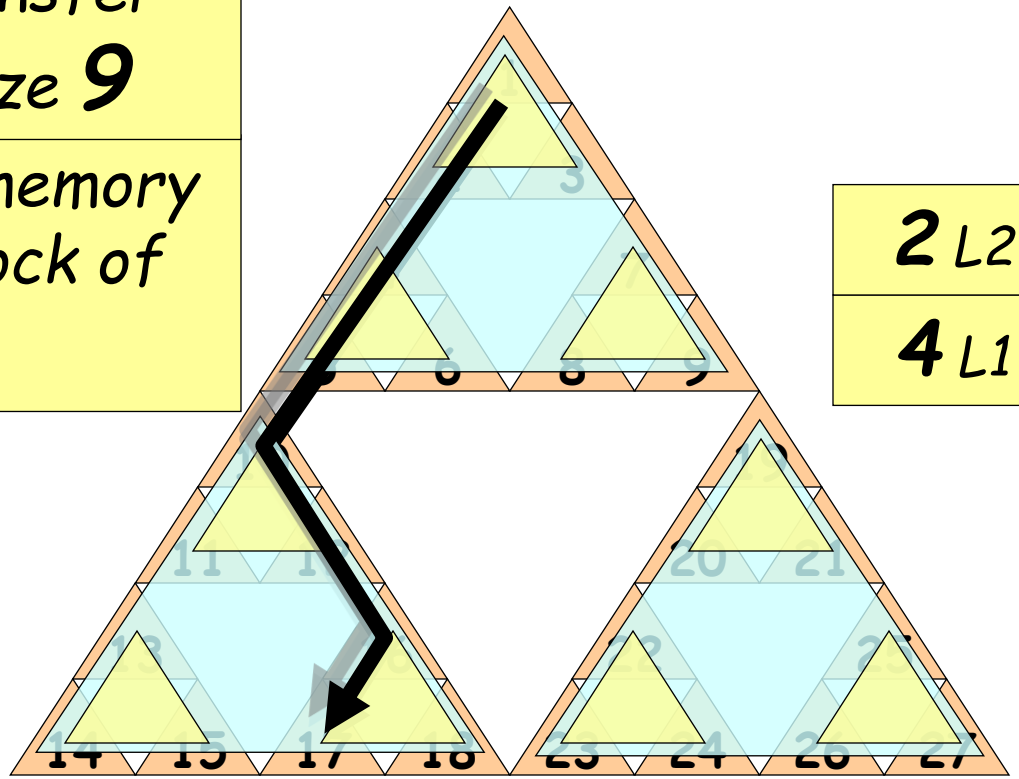
4 *cache misses*

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

Van Emde Boas Layout

Searching with L2 memory transfer block of size **9**

and with L1 memory transfer block of size **3**



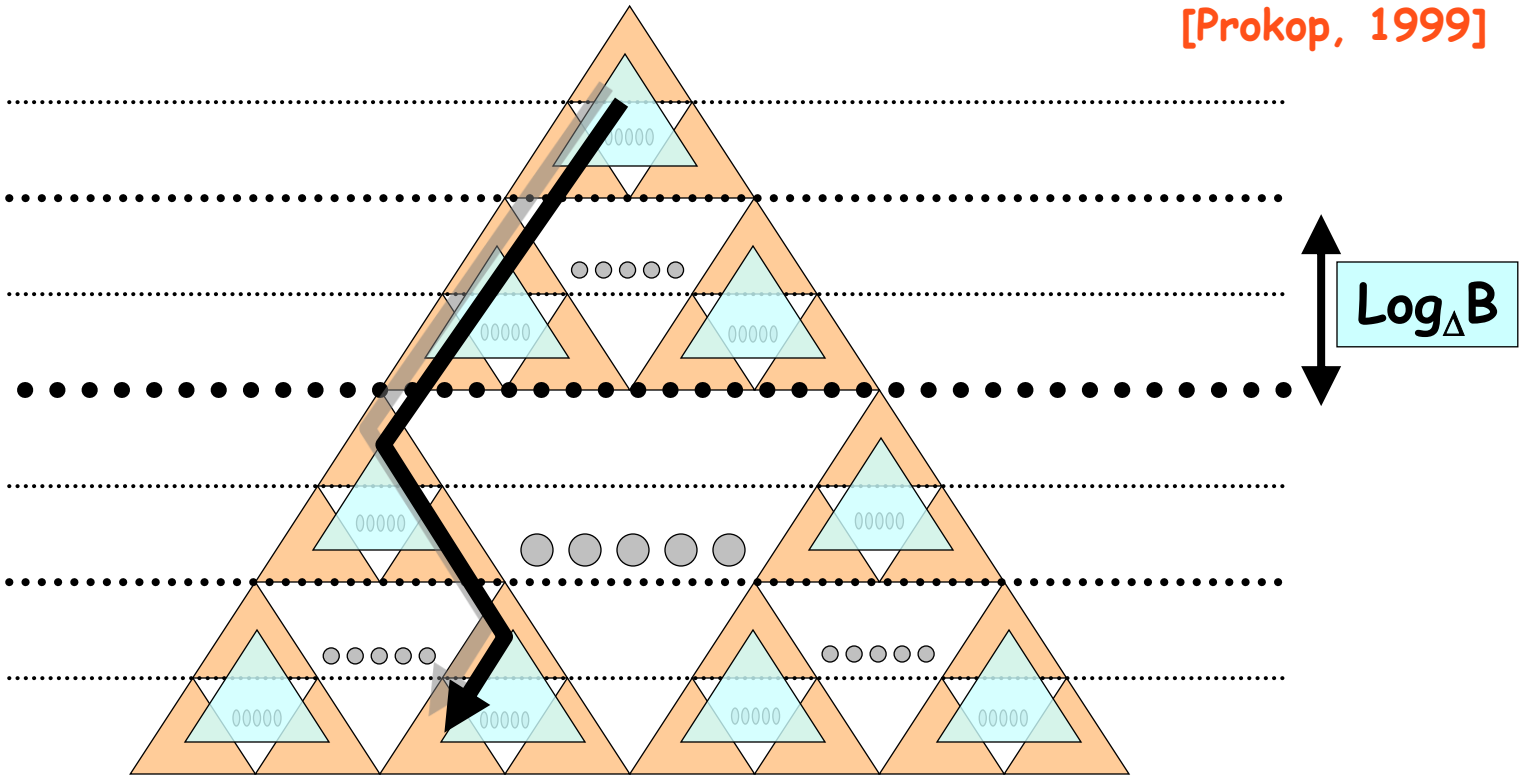
2 *L2 cache misses*

4 *L1 cache misses*

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

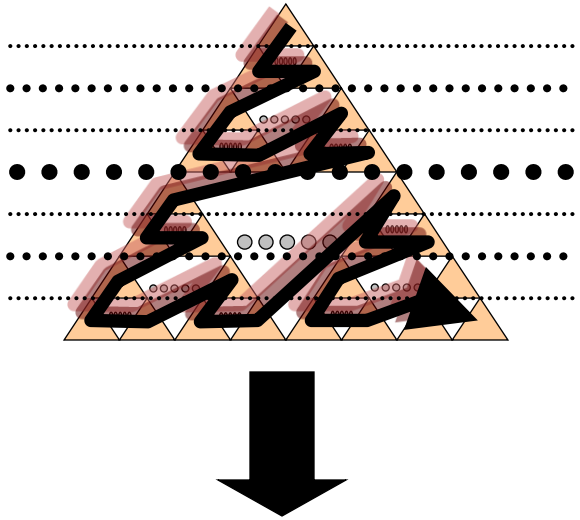
Static Cache Oblivious Tree

[Prokop, 1999]



#IOs per search = $O\left(\frac{\log_\Delta N}{\log_\Delta B}\right) = O(\log_B N)$

From Static to Dynamic

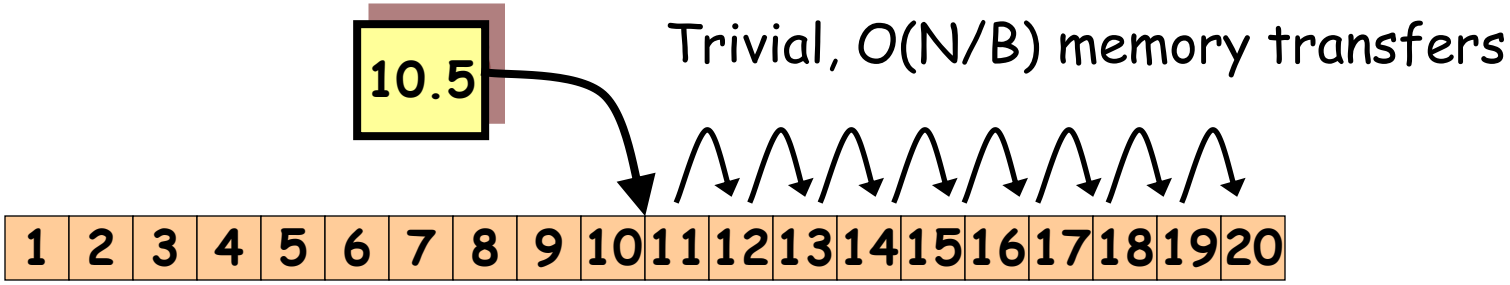


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

- Static is fine, but… A Dynamic B-Tree should allow the addition and removal of nodes
- How do we add an element efficiently to an ordered array ?

Dynamic B-Tree – restating the problem

- How do we add an element efficiently to an ordered array ?



Can we do better ?

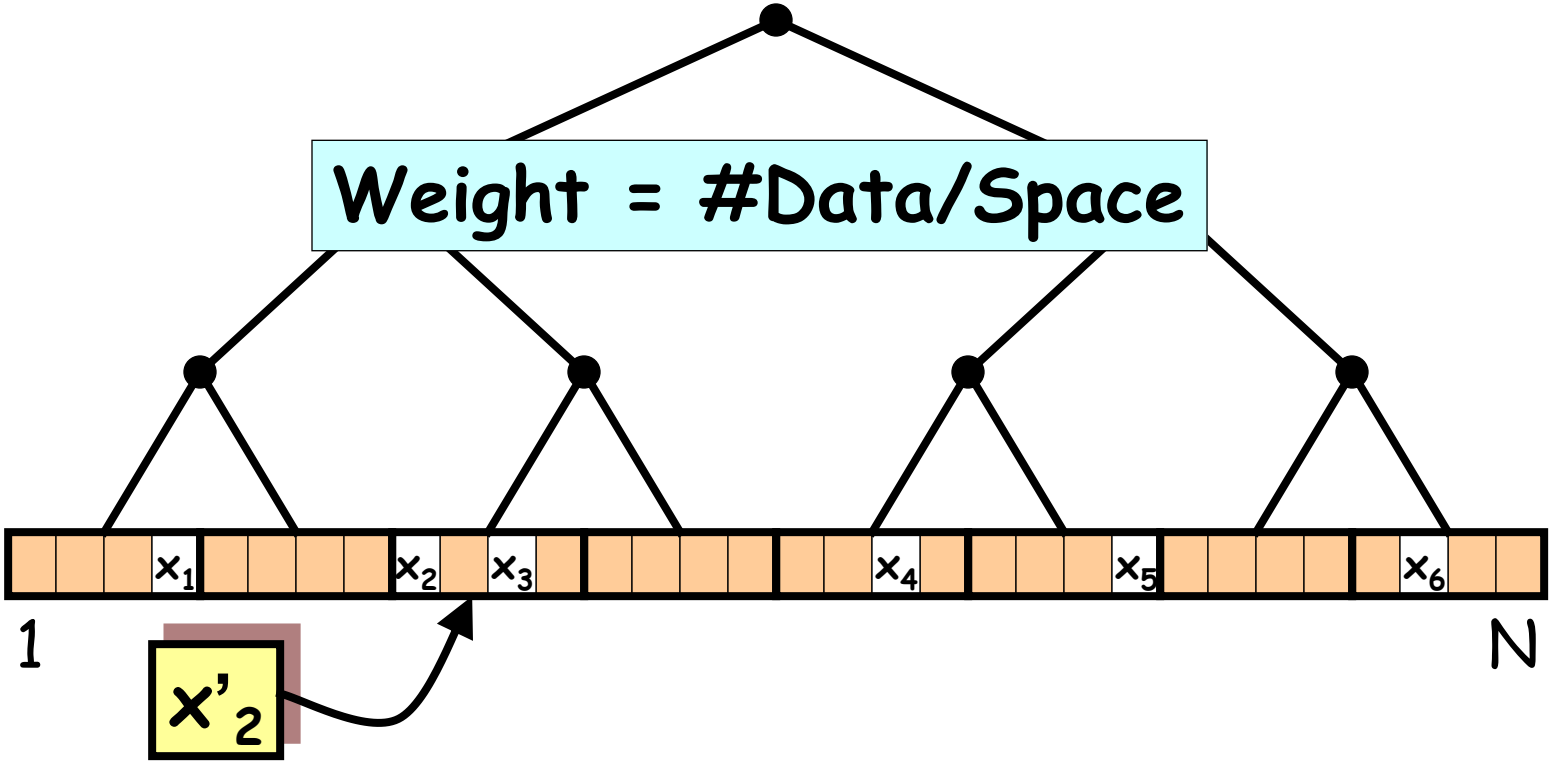
Dynamic B-Tree – restating the problem

- General idea – Array with enough extra space between nodes for future insertions

- Contradictory goals – nodes should be packed densely for locality of reference
- Guideline - Distribute elements evenly in the array
  - Store any set of k contiguous elements xi1,…, xik in a contiguous subarray of size O(k)

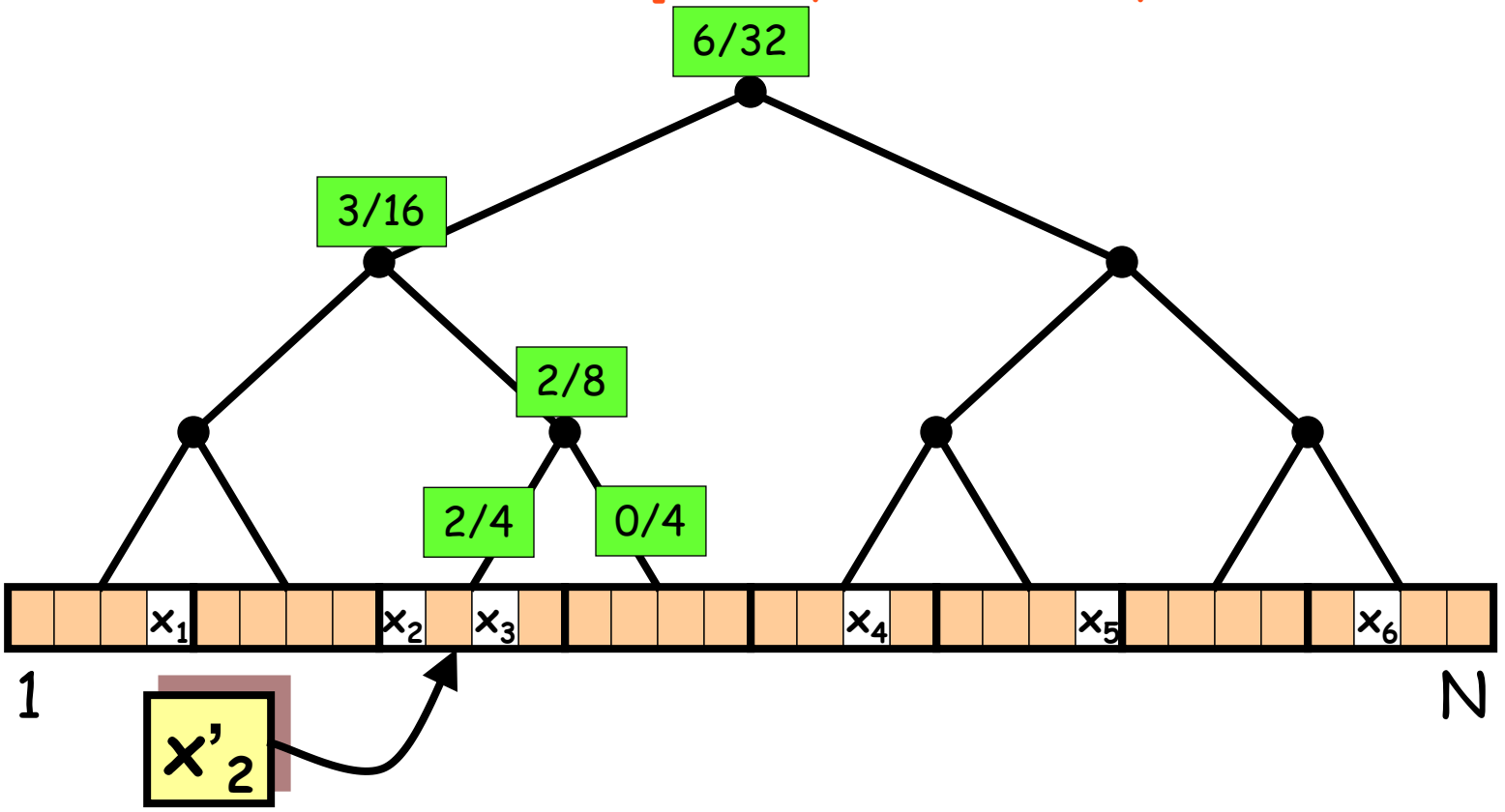
Packed Memory Management

[A. Itai, A. G. Konheim, and M. Rodeh 1981]



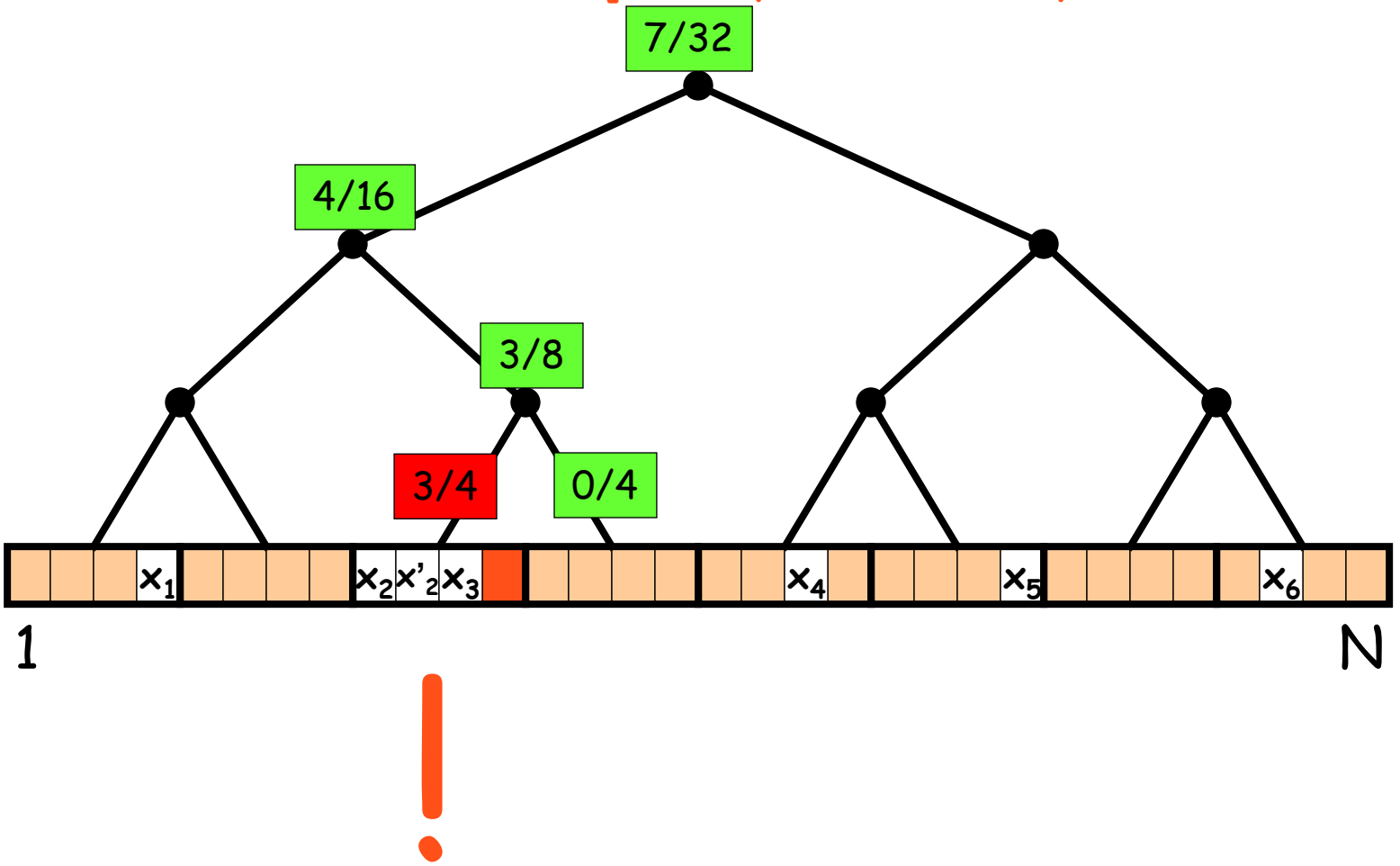
Packed Memory Management

[A. Itai, A. G. Konheim, and M. Rodeh 1981]



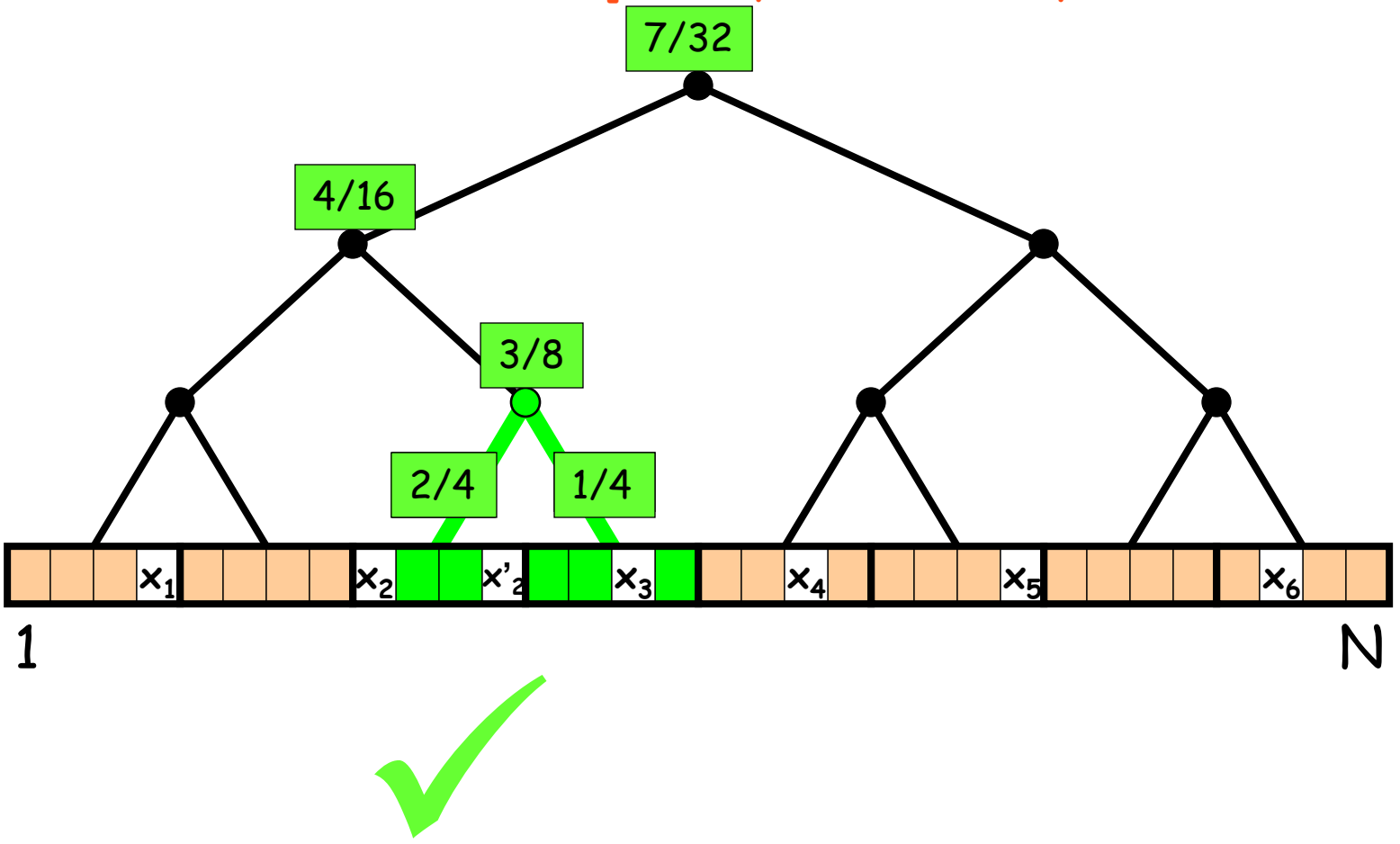
Packed Memory Management

[A. Itai, A. G. Konheim, and M. Rodeh 1981]



Packed Memory Management

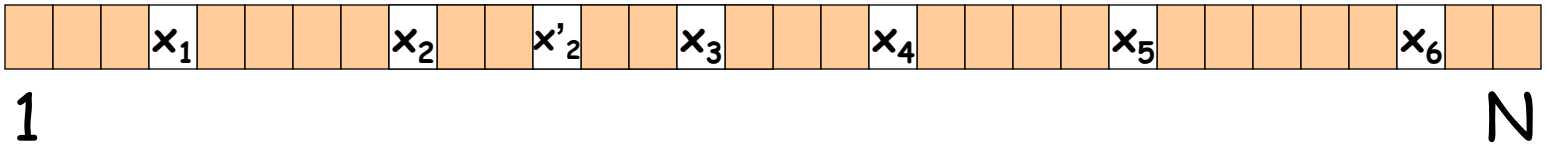
[A. Itai, A. G. Konheim, and M. Rodeh 1981]



Packed Memory Management

[A. Itai, A. G. Konheim, and M. Rodeh 1981]

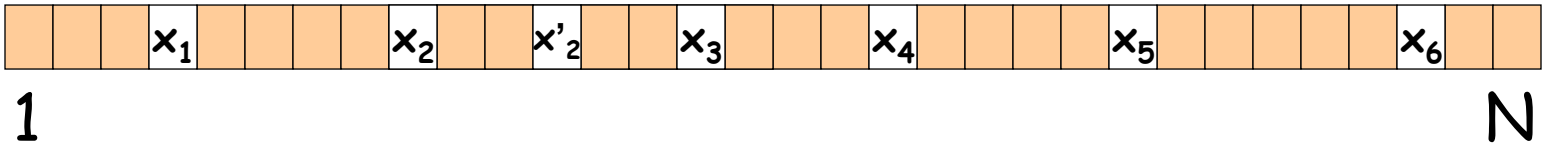
- Size of the interval to rebalance ?
- Thresholds for interval over/underflows ?



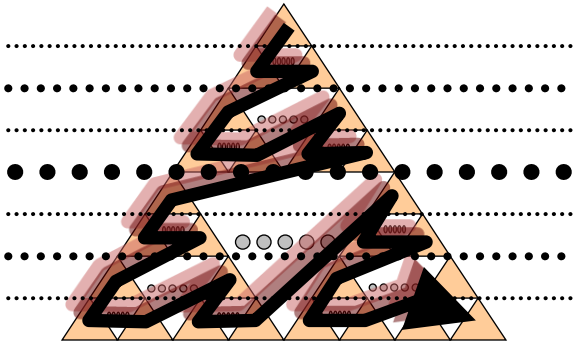
Packed Memory Management

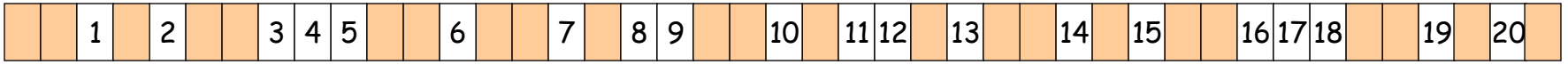
[A. Itai, A. G. Konheim, and M. Rodeh 1981]

- Scanning a k consecutive elements
  - k Elements are stored in O(k) memory cells
  - O(1+k/B) memory transfers
- Inserting/deleting an element
  - Moves $O(\log^2 N)$ amortized consecutive cells
  - $O(1 + \log^2 N / B)$ amortized memory transfers

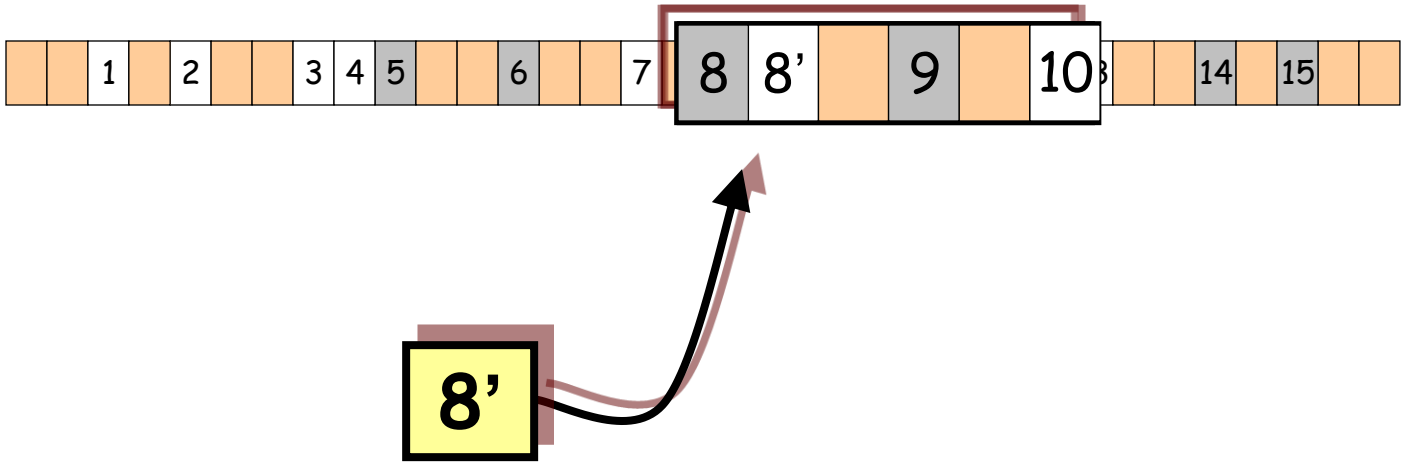


Putting it all together… ?

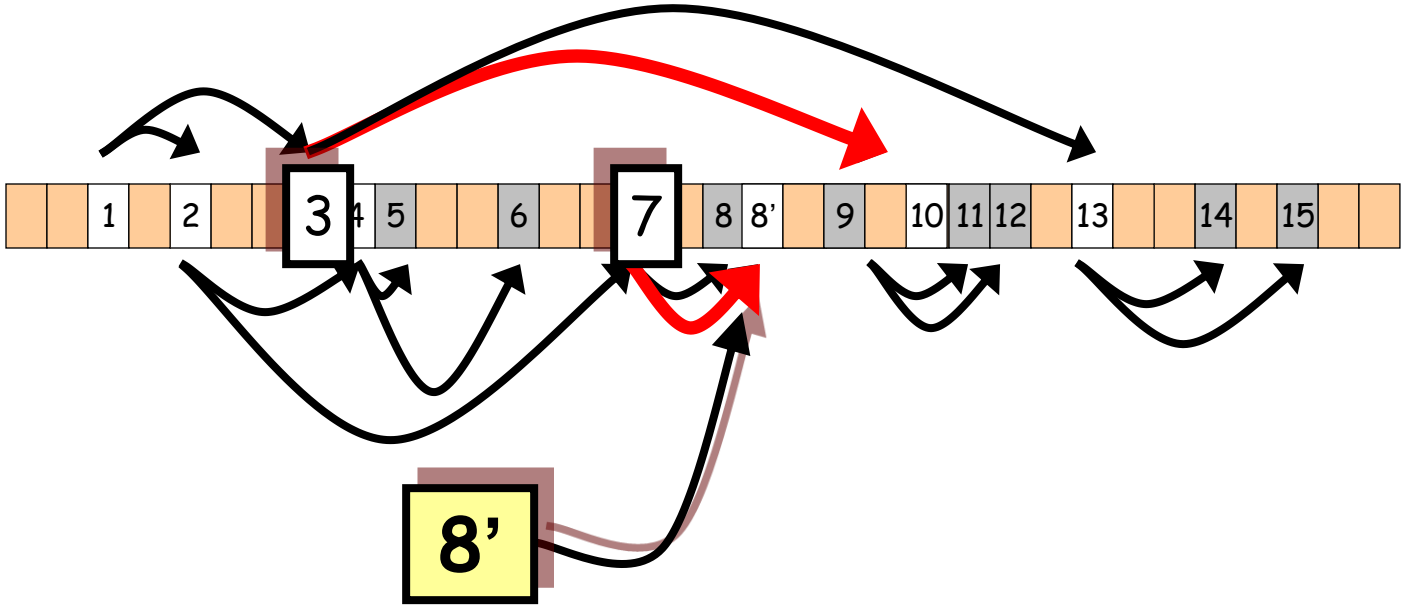




Putting it all together… ?

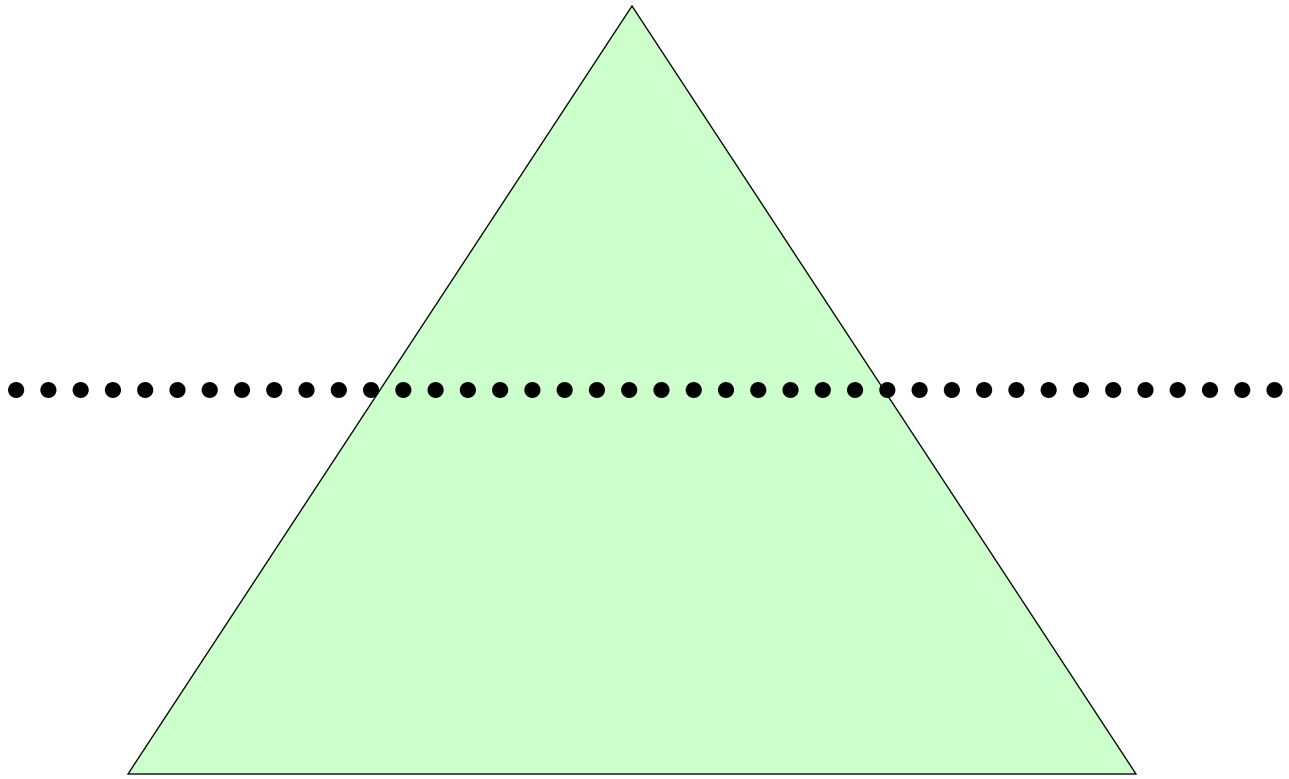


It's still a tree !

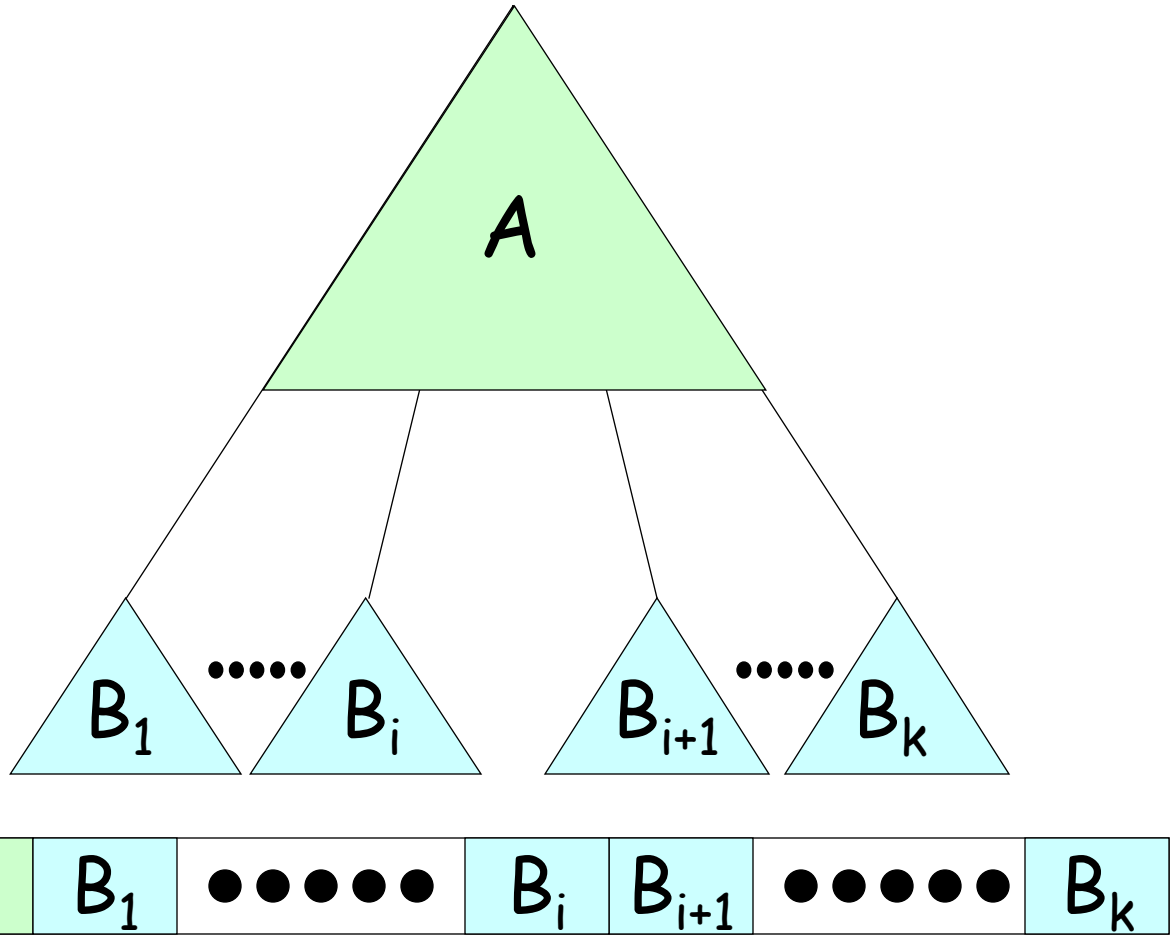


- Insertion moves $O(\log^2 N)$ amortized nodes
- Requires $O(\log^2 N)$ back pointers updates
- And what about the van Emde layout ?

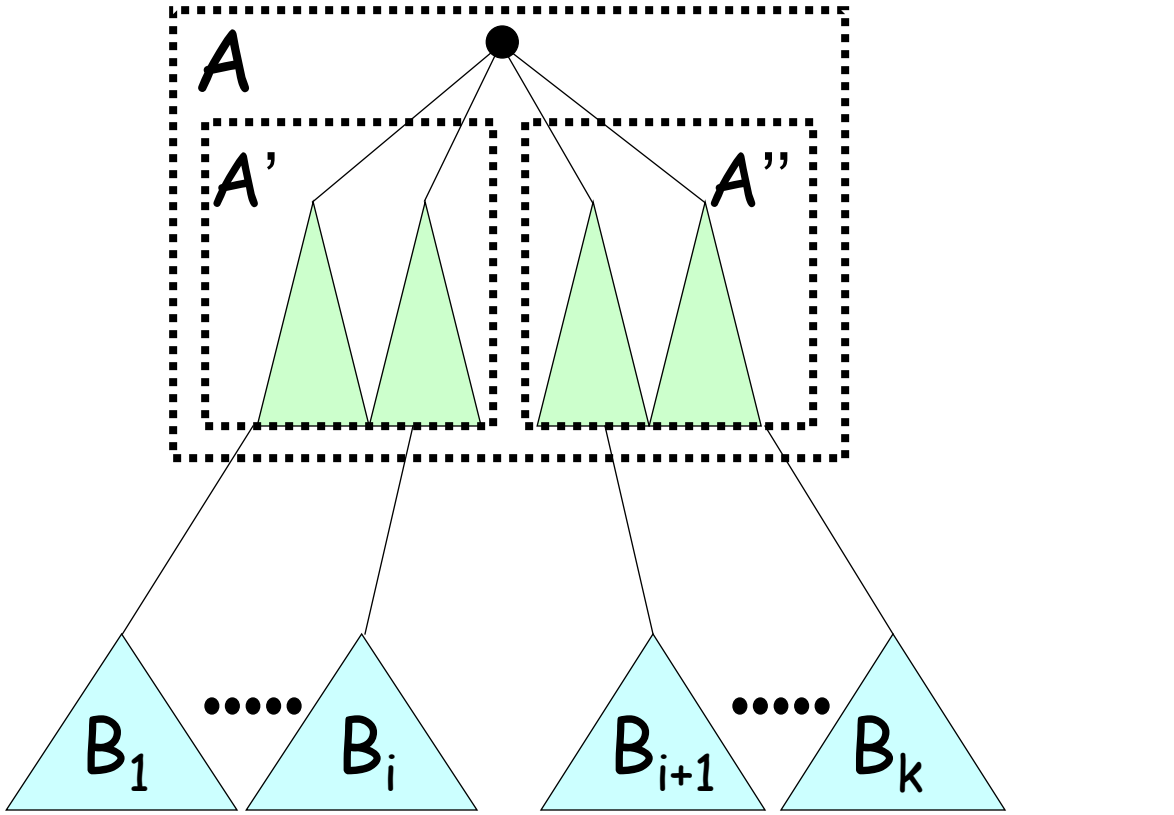
Splits and Merges

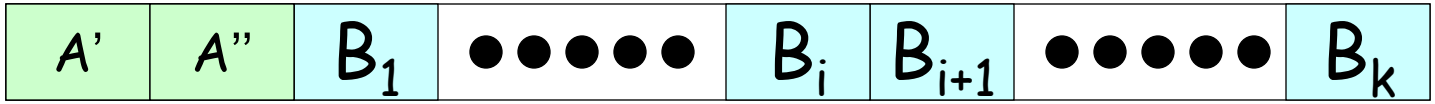


Splits and Merges

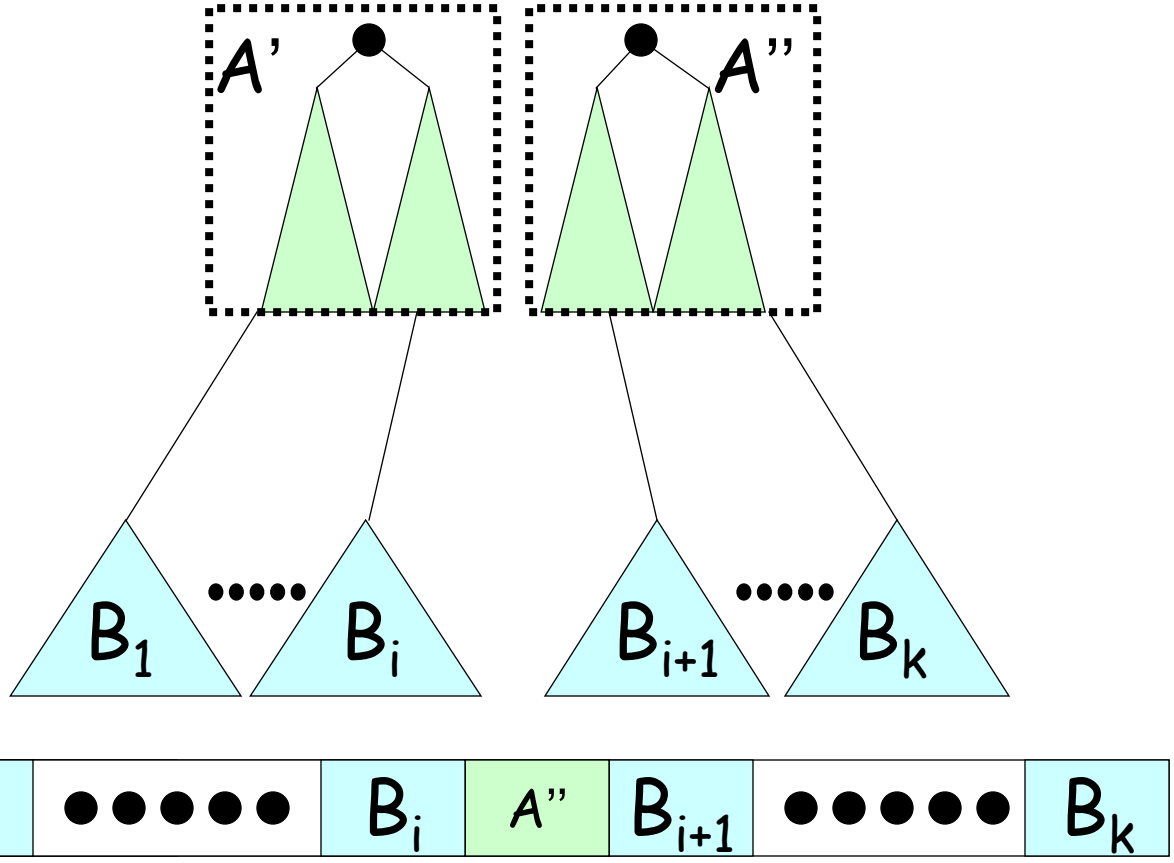


Splits and Merges

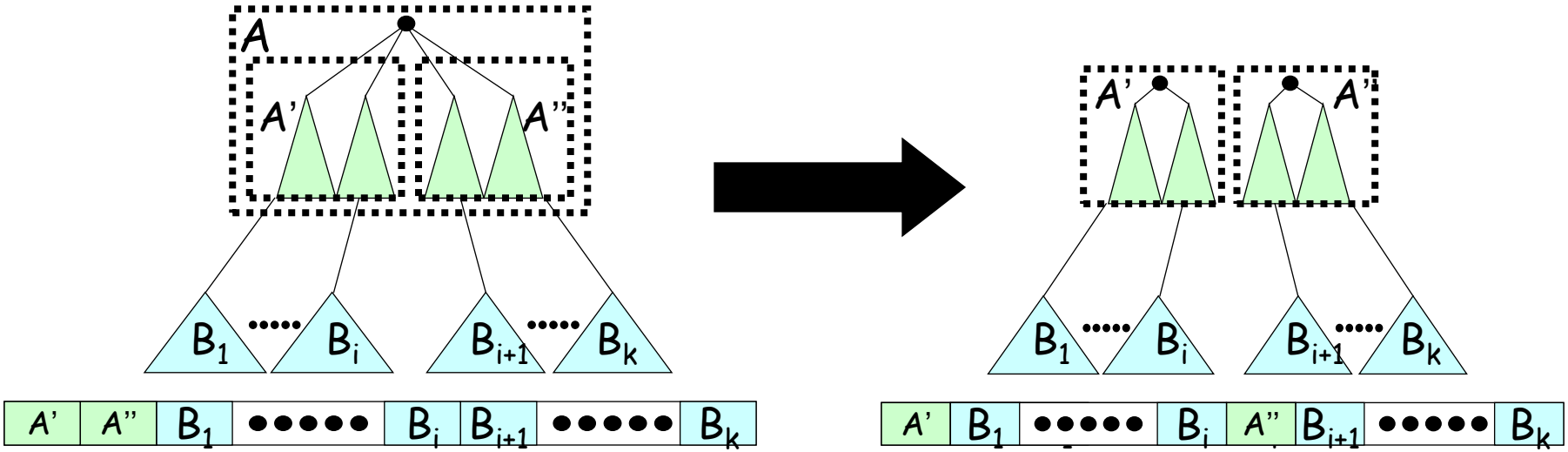




Splits and Merges



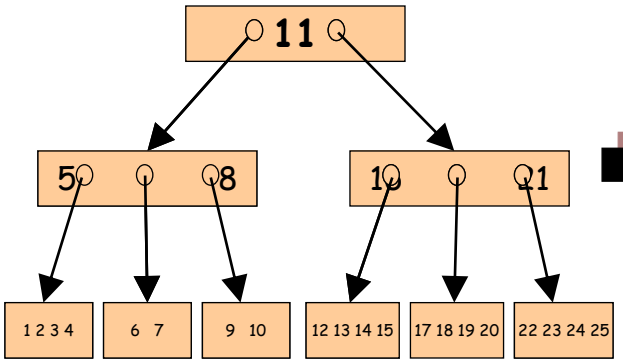
Splits and Merges



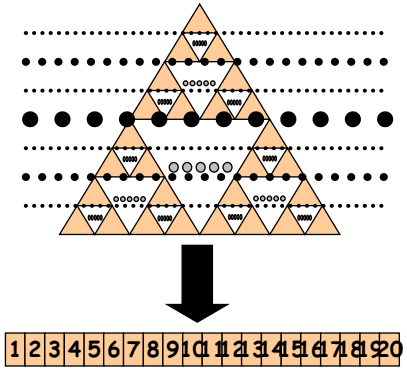
- Each update requires O(1 + logN/B) amortized memory transfers

Putting it all together… again

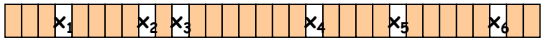
Weight-balanced B-tree



van Emde Boas layout



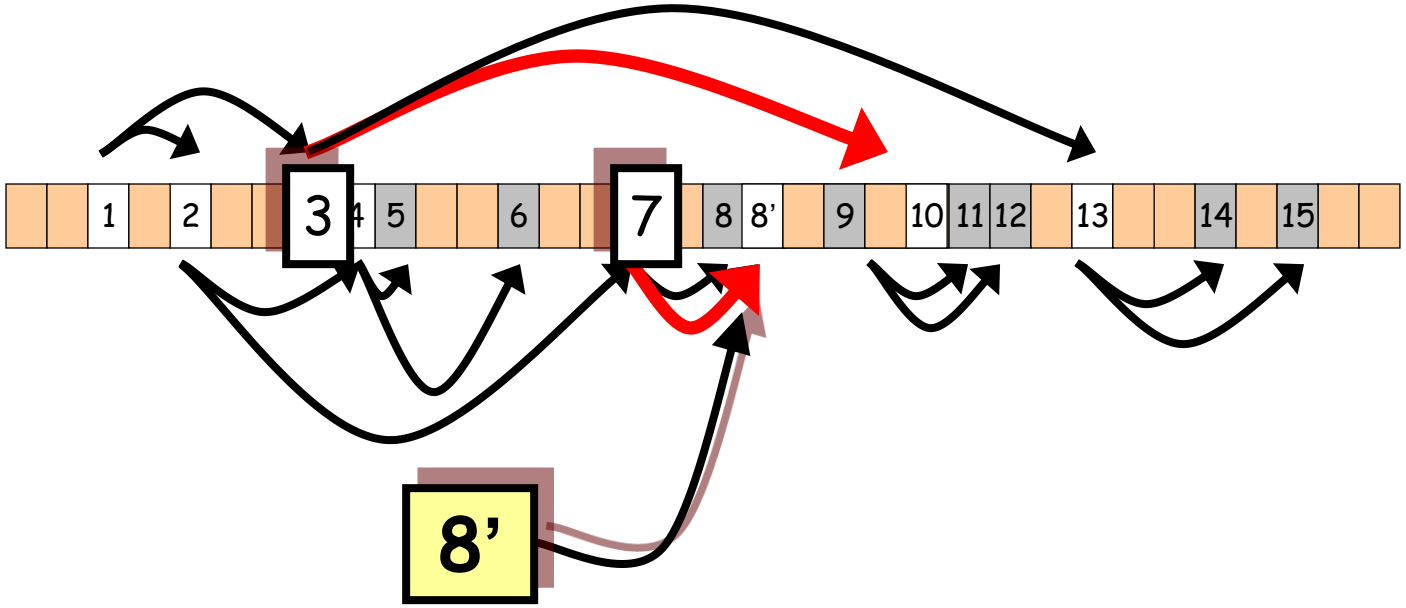
Packed Memory Maintenance



- Search $O(\log_B N)$
- Scan $O(1 + k/B)$

- Update $O(\log_B N + \log^2 N)$
  - Search for key $O(\log_B N)$
  - Create/delete leaf $O(1)$
  - Rearange packed array $O(1+\log^2 N/B)$
  - Update back pointers $O(1+\log^2 N)$
  - Split/Merge nodes $O(1+\log N/B)$

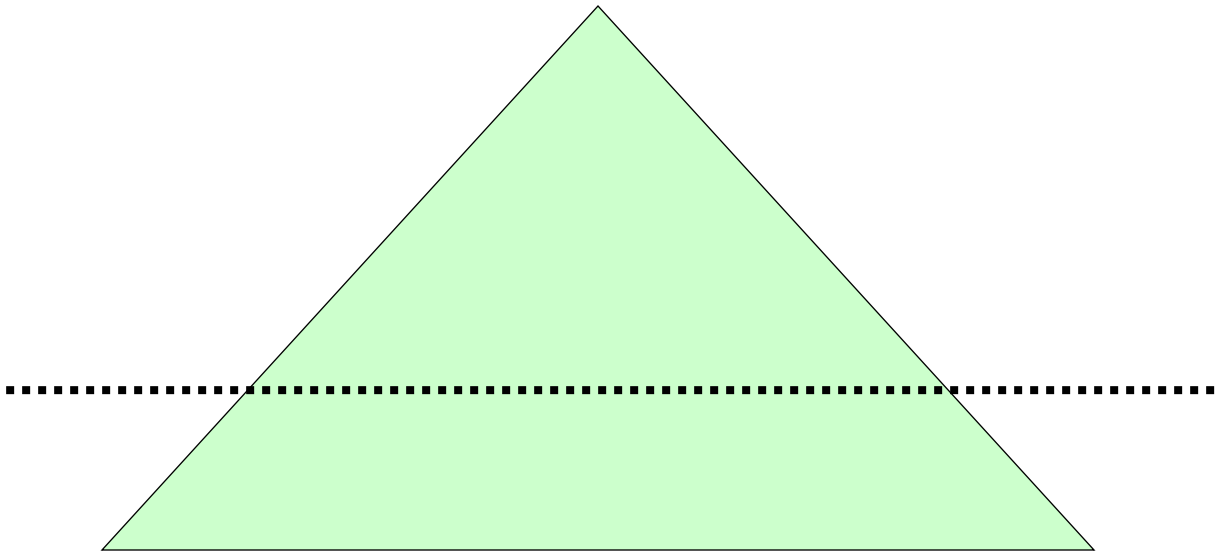
Saving on back pointer updates ?



- How can we reduce back pointer updates ?

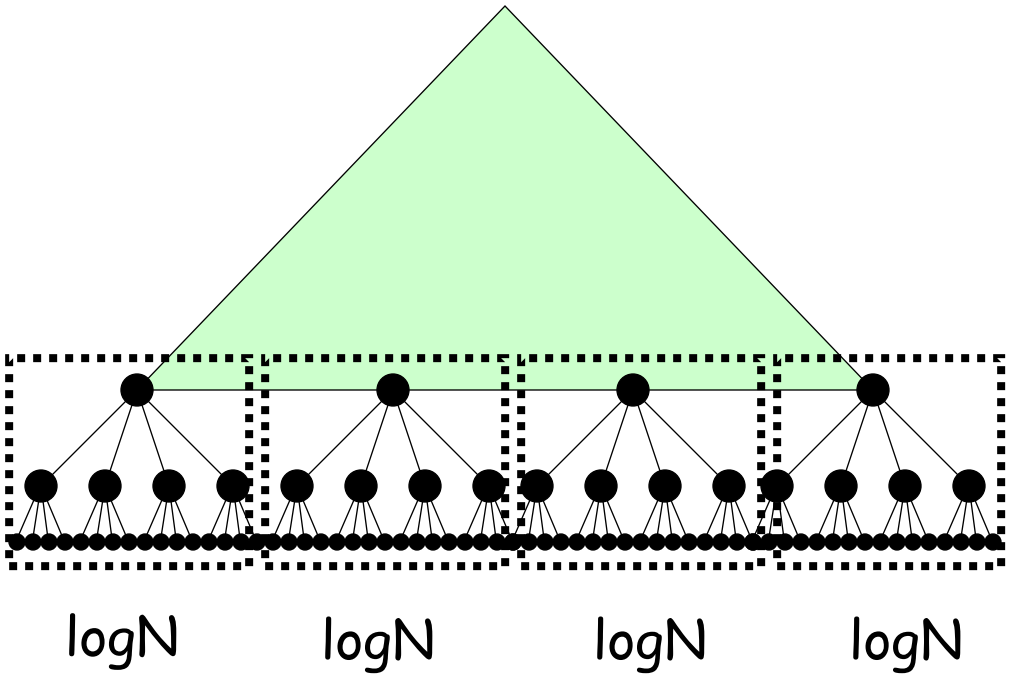
Saving on back pointer updates ?

- Idea – pack groups of leaves together beneath a single parent



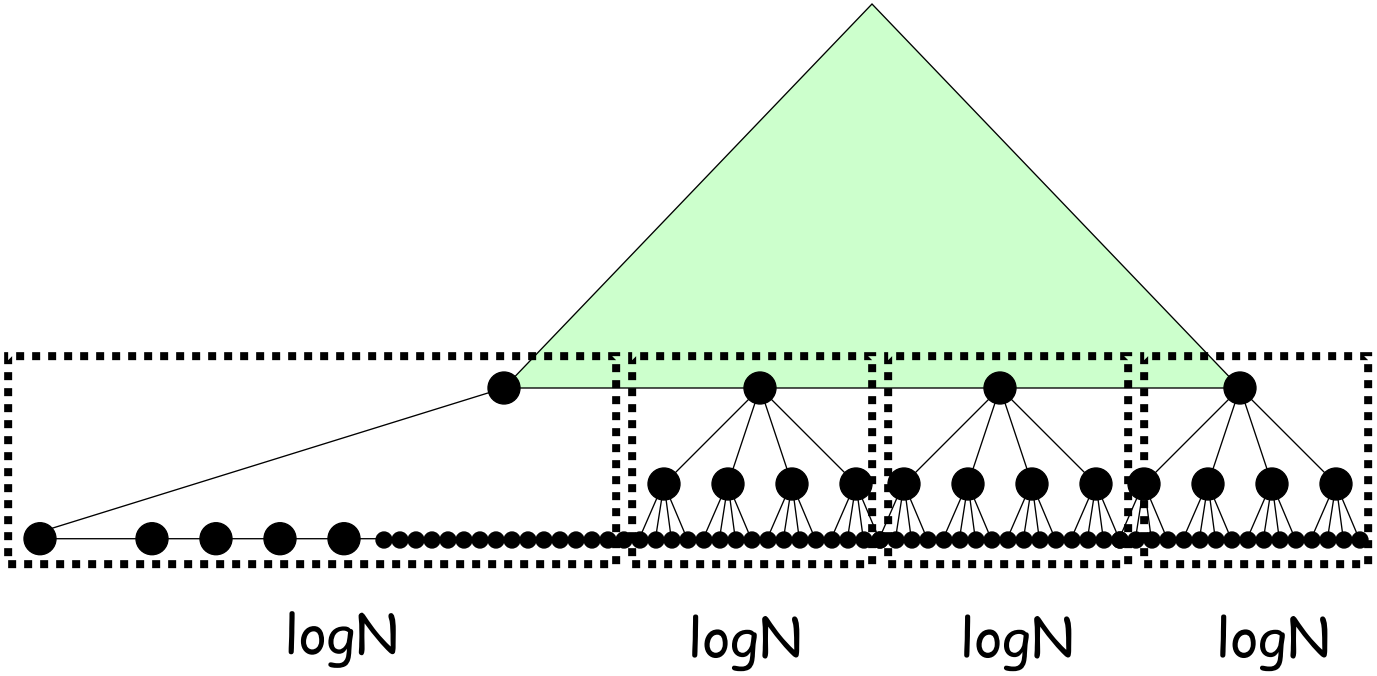
Saving on back pointer updates ?

- Idea – pack groups of leaves together beneath a single parent



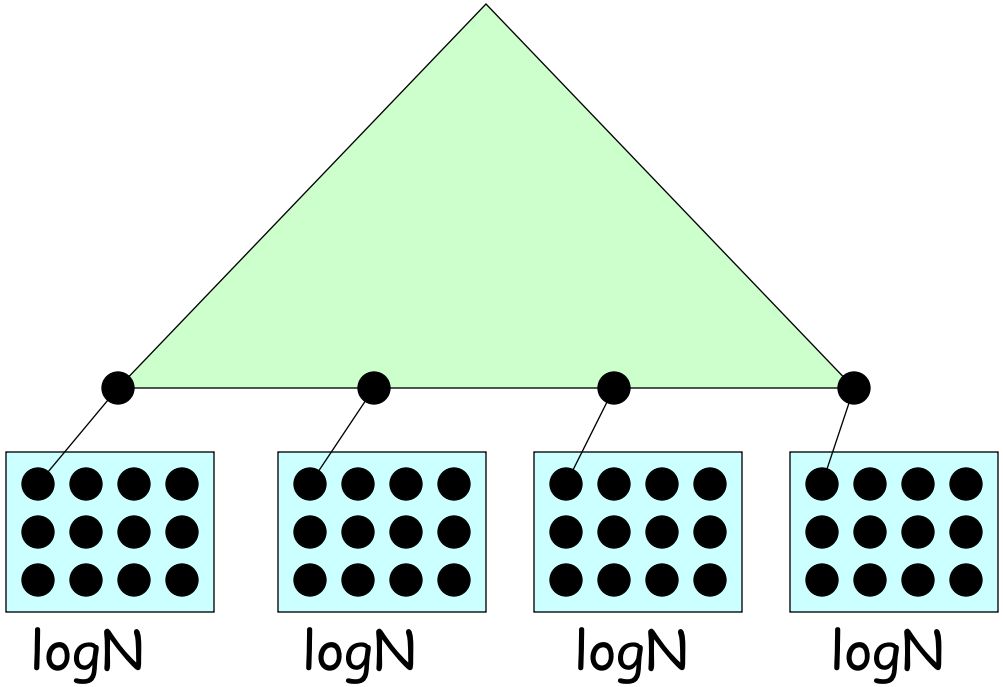
Indirection

- Idea – pack groups of leaves together beneath a single parent



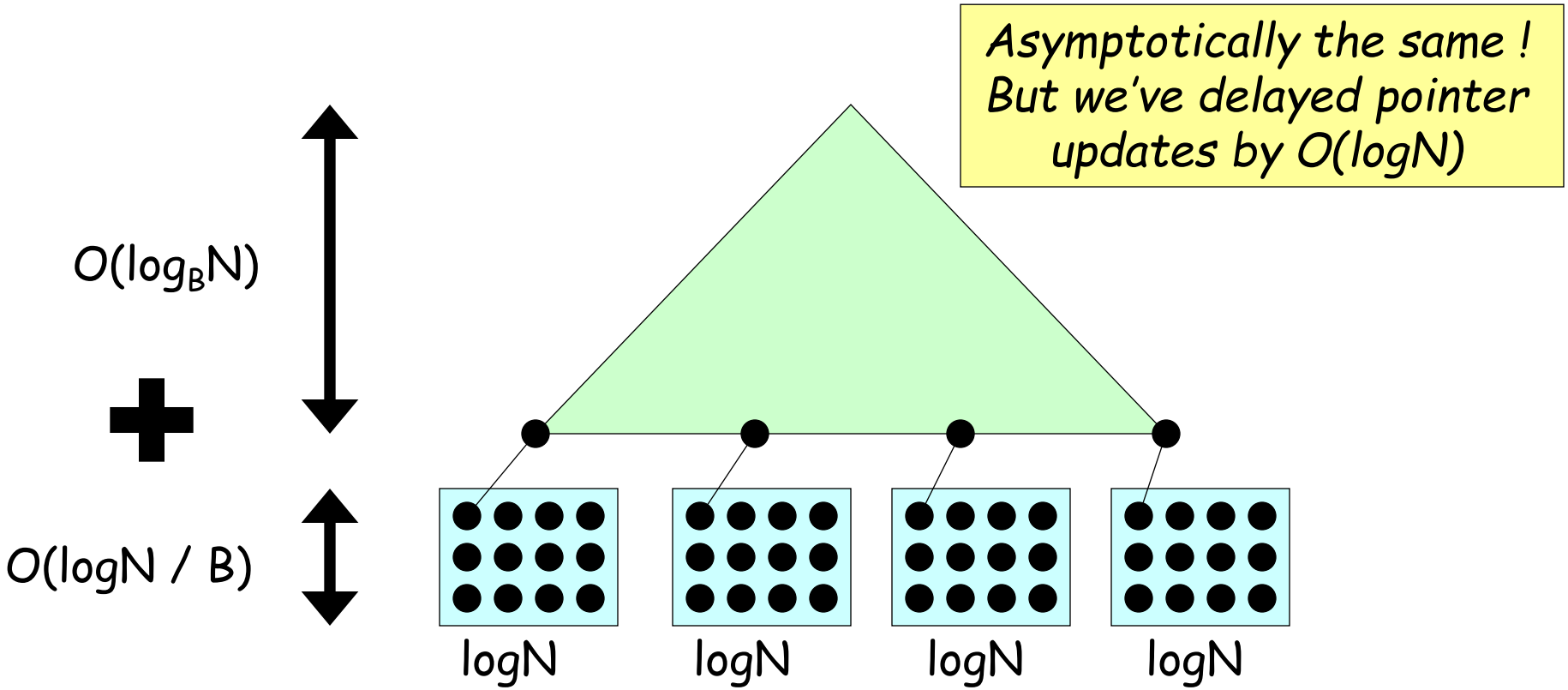
Indirection

- Idea – pack groups of leaves together beneath a single parent



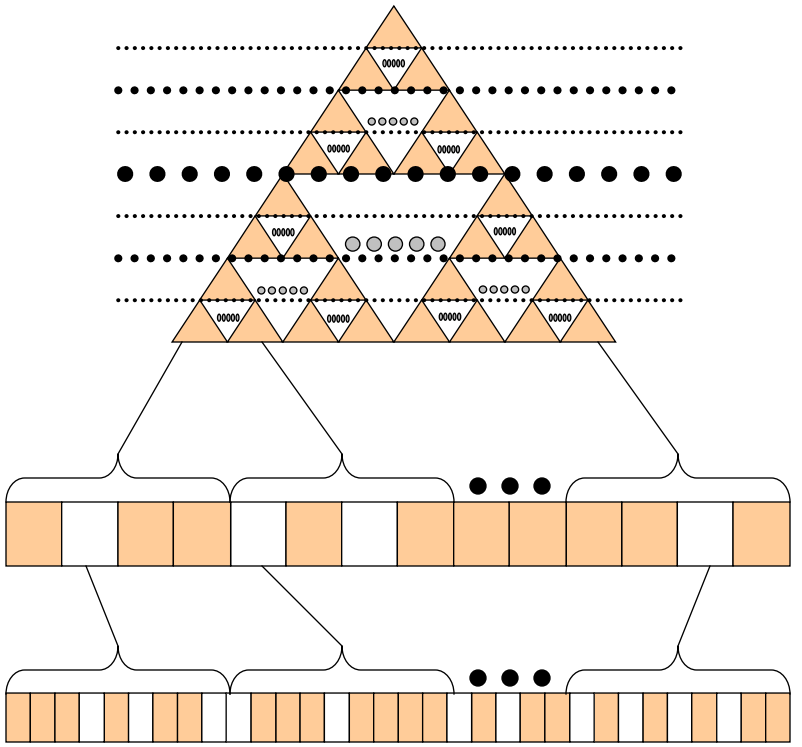
Indirection

- Searching for an element ?



Final build – two levels of indirection

- Search $O(\log_B N)$
- Scan $O(1 + k/B)$
- Updates
  - $O(\log_B N + \log^2 N / B)$



L_0 contains all the values

The end…

Presented by: Itai Lahan