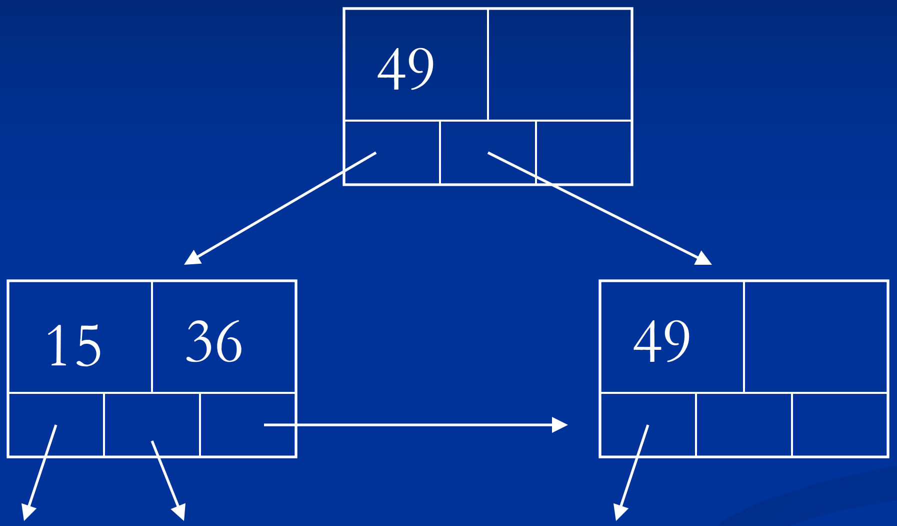
CPS216: Data-intensive Computing Systems

Operators for Data Access (contd.)

Shivnath Babu

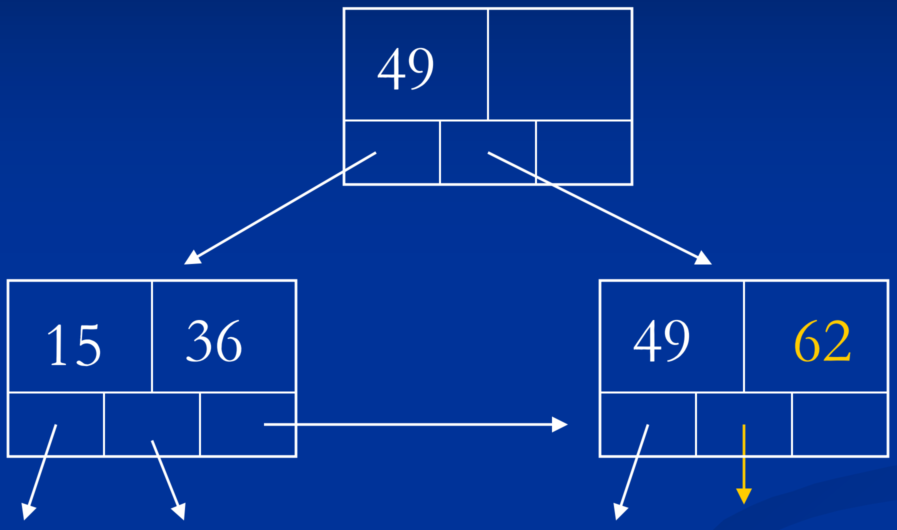
Insertion in a B-Tree



n = 2

Insert: 62

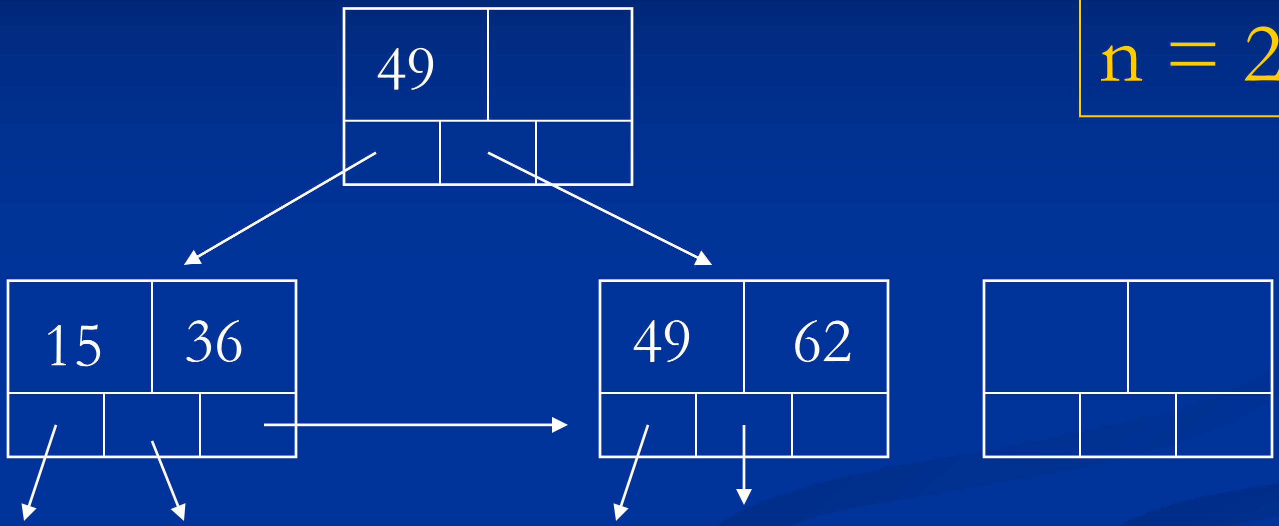
Insertion in a B-Tree



n = 2

Insert: 62

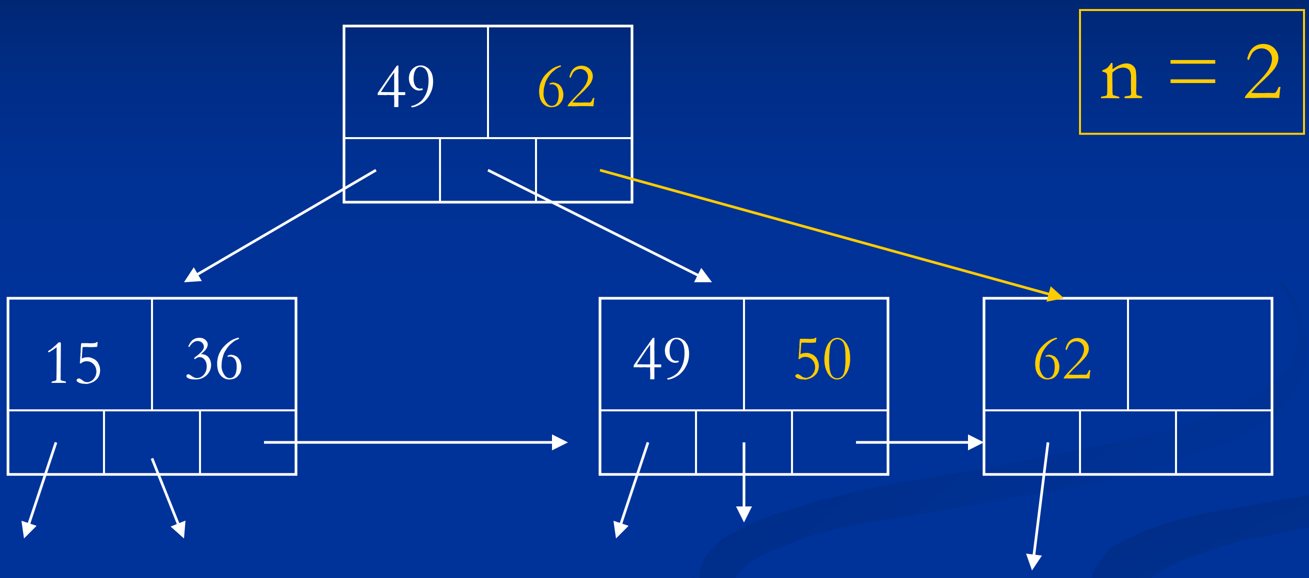
Insertion in a B-Tree



n = 2

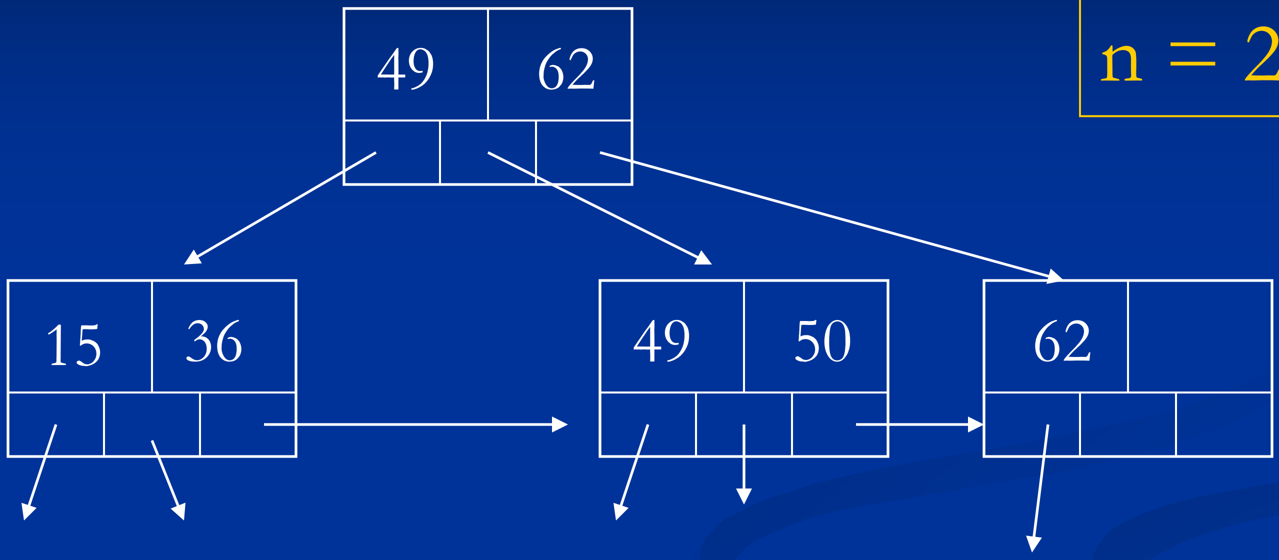
Insert: 50

Insertion in a B-Tree



Insert: 50

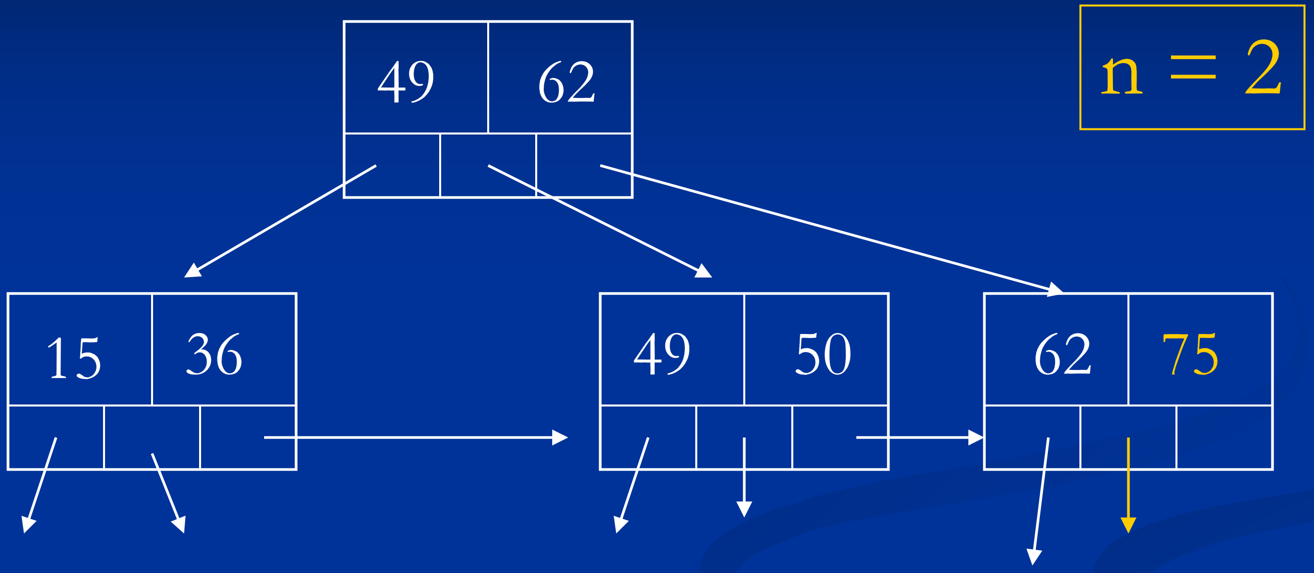
Insertion in a B-Tree



n = 2

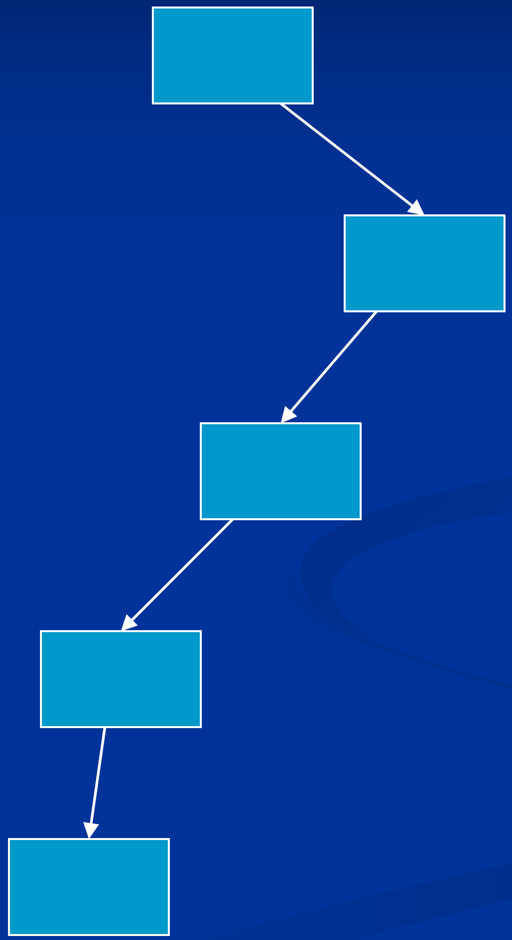
Insert: 75

Insertion in a B-Tree

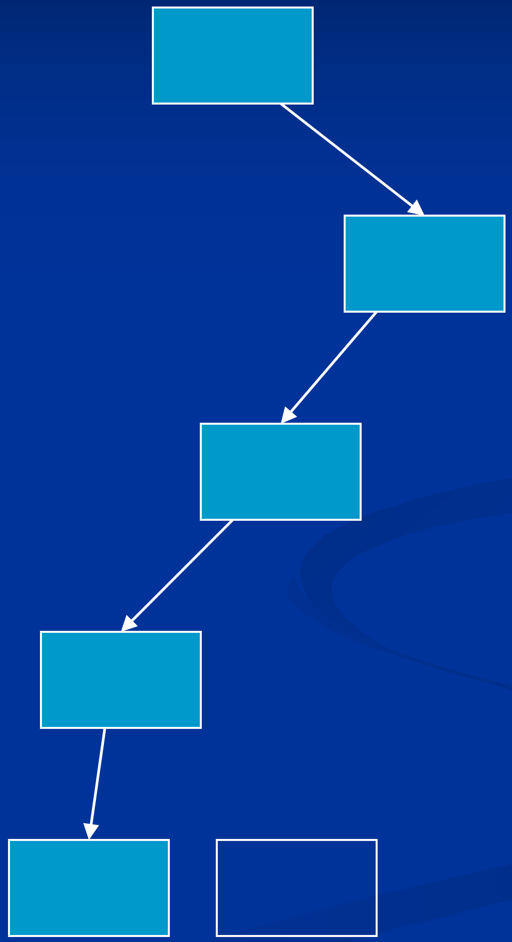


Insert: 75

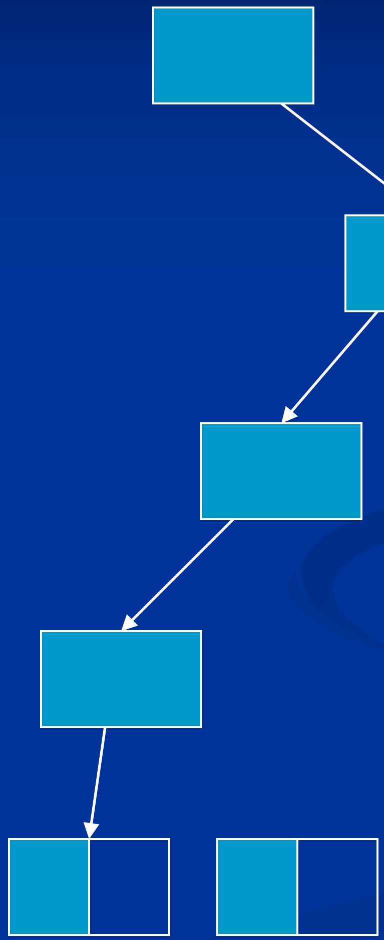
Insertion



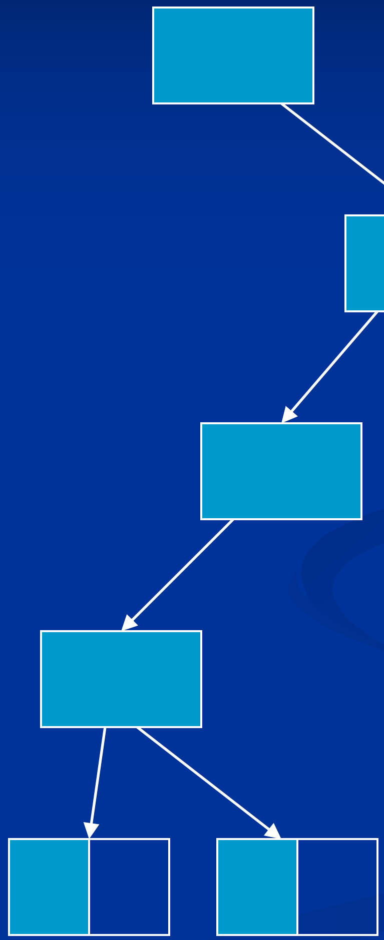
Insertion



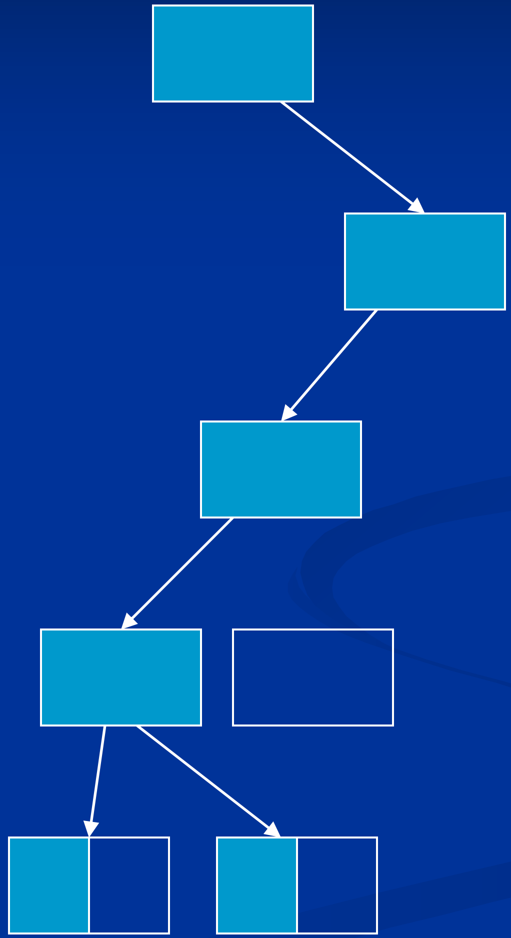
Insertion



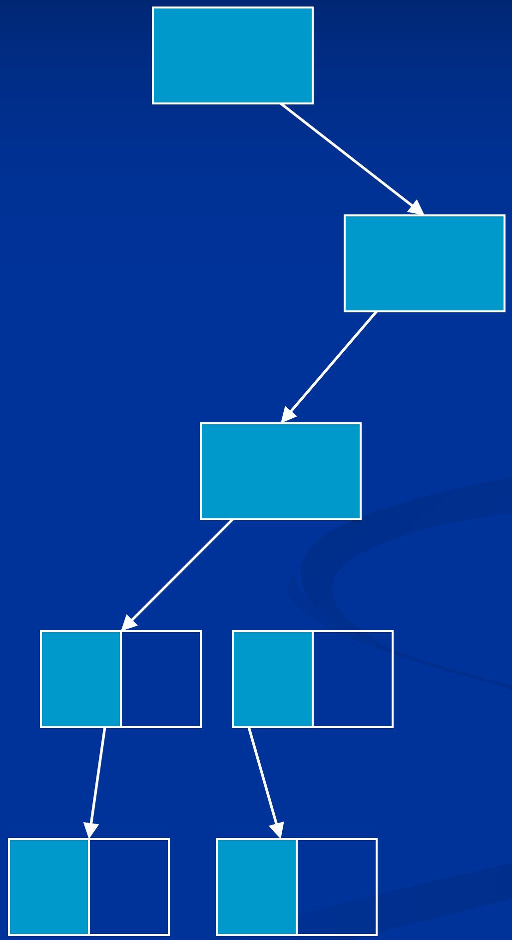
Insertion



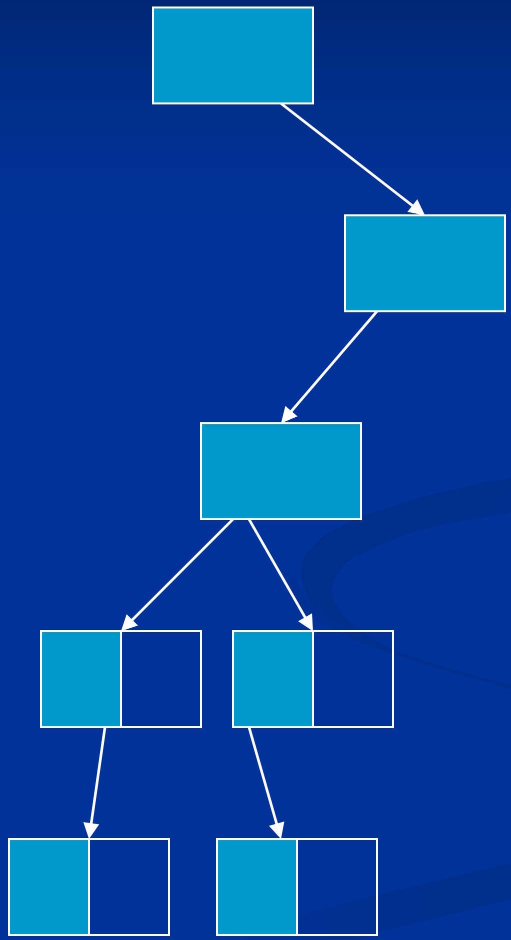
Insertion



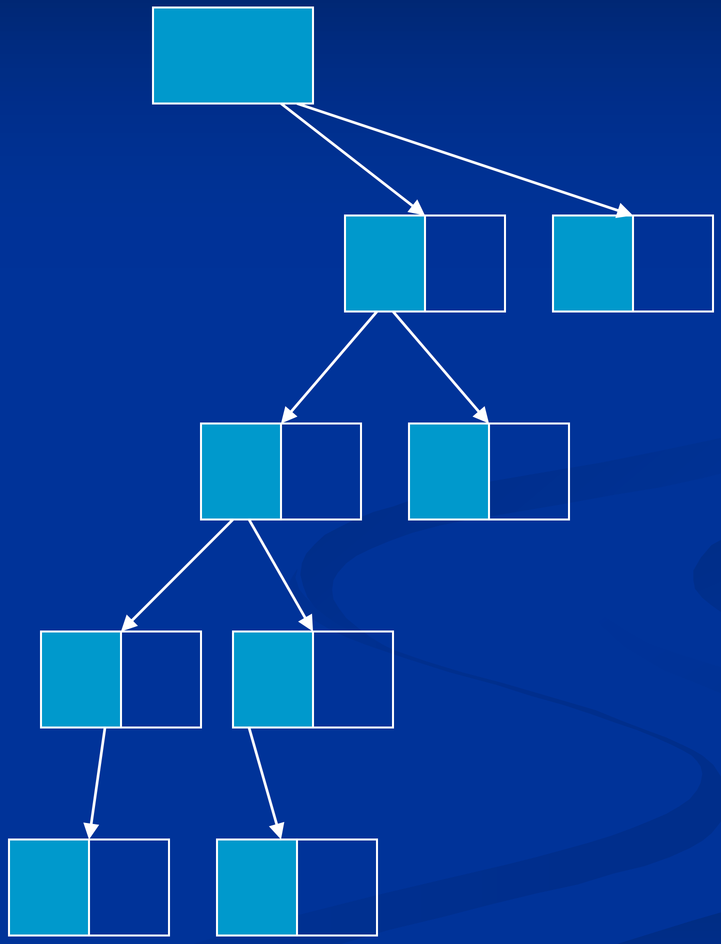
Insertion



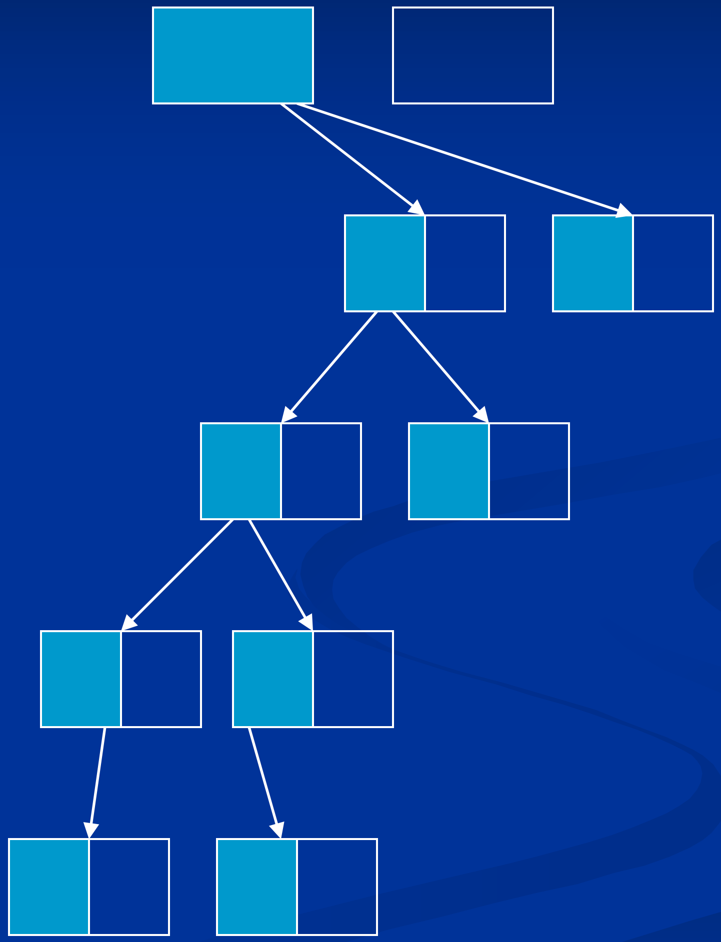
Insertion



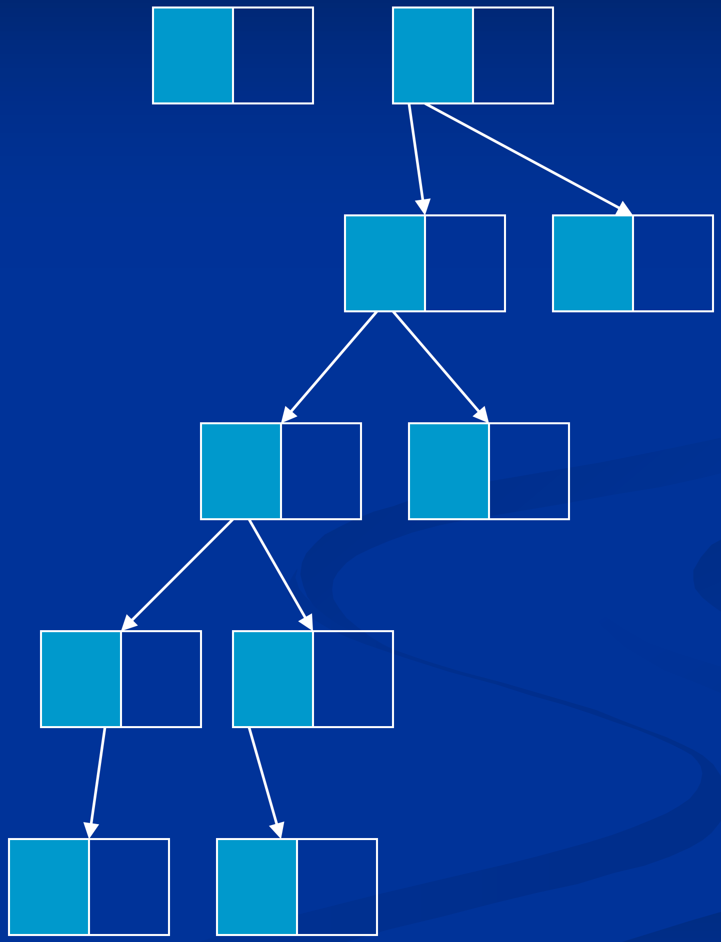
Insertion



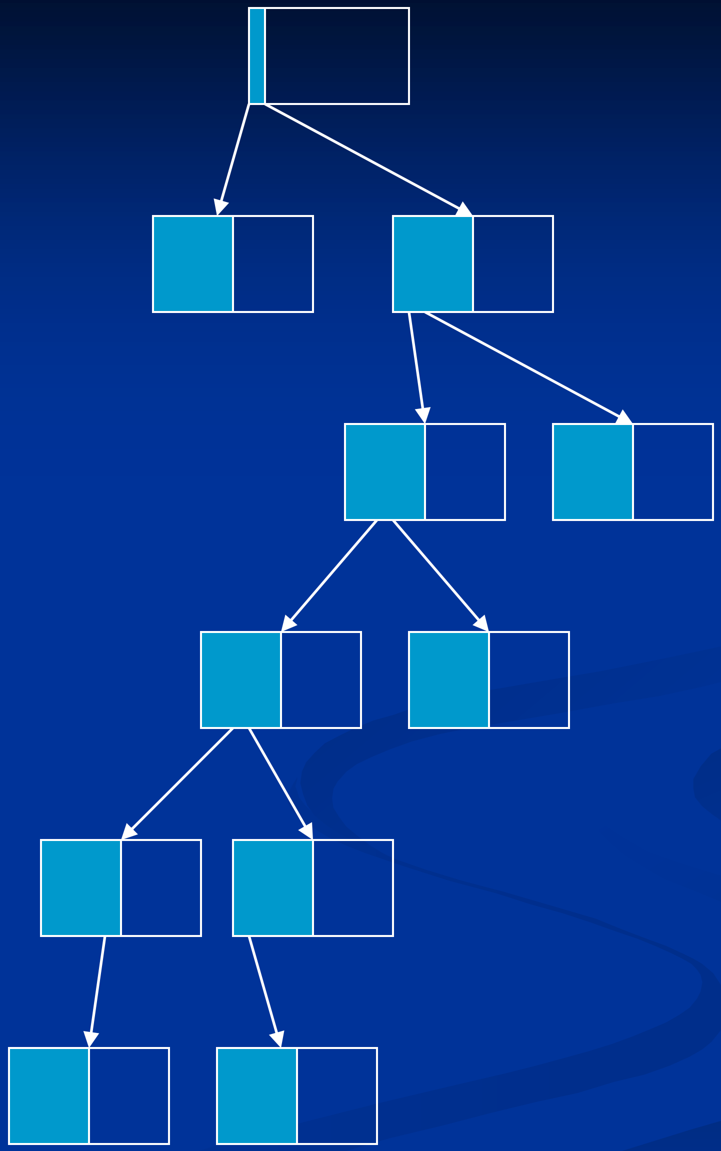
Insertion



Insertion



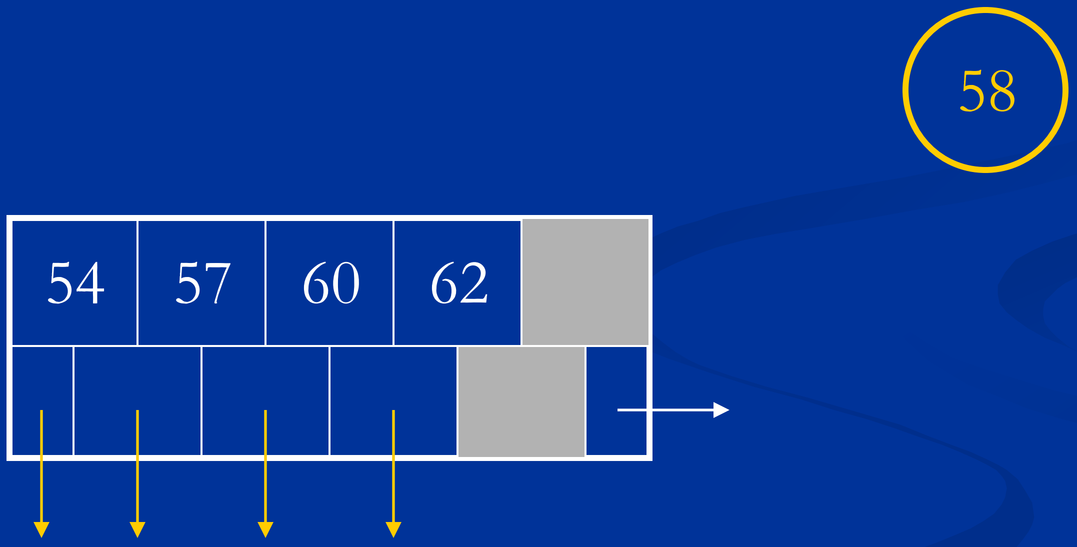
Insertion



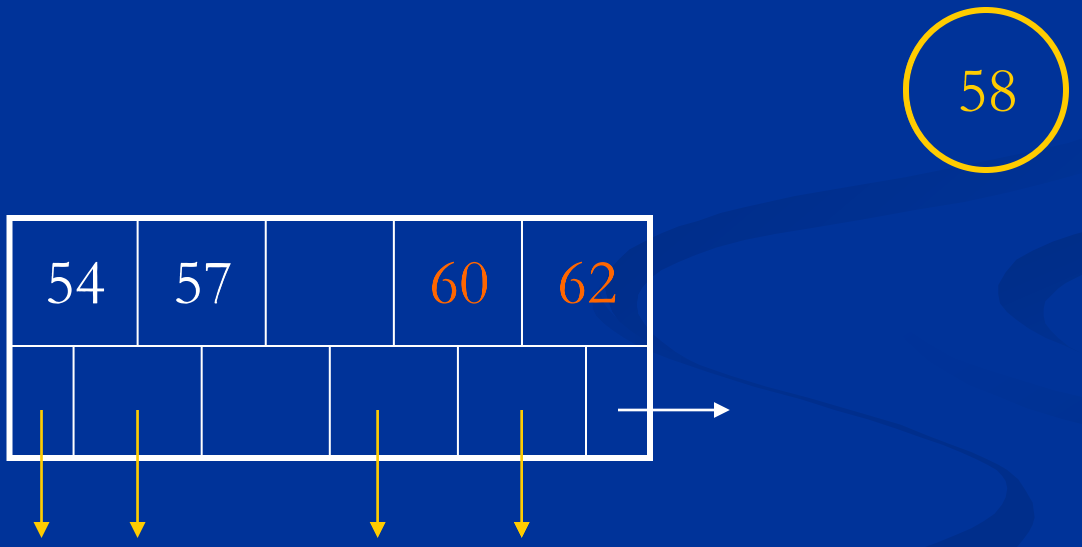
Insertion: Primitives

- Inserting into a leaf node
- Splitting a leaf node
- Splitting an internal node
- Splitting root node

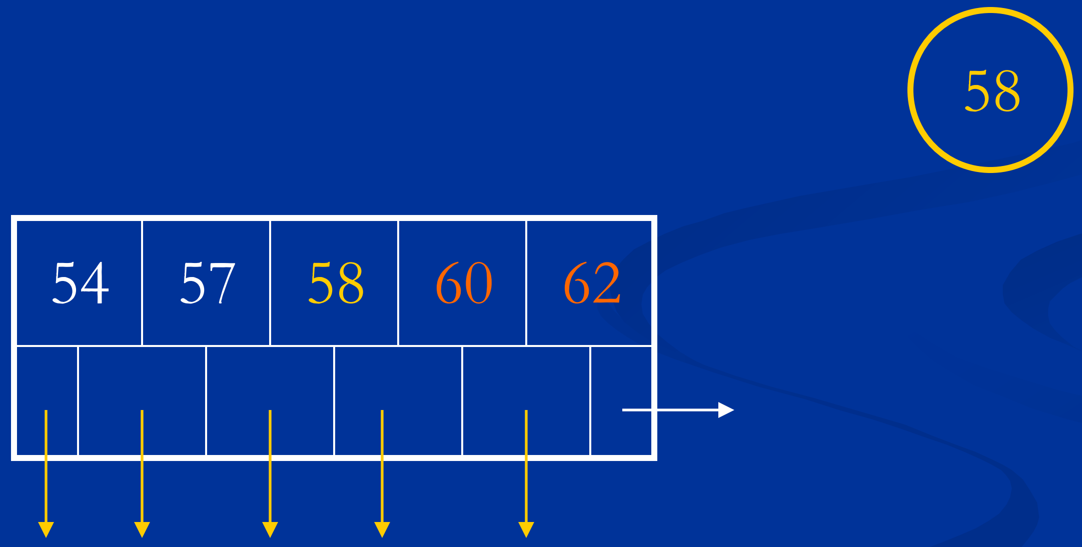
Inserting into a Leaf Node



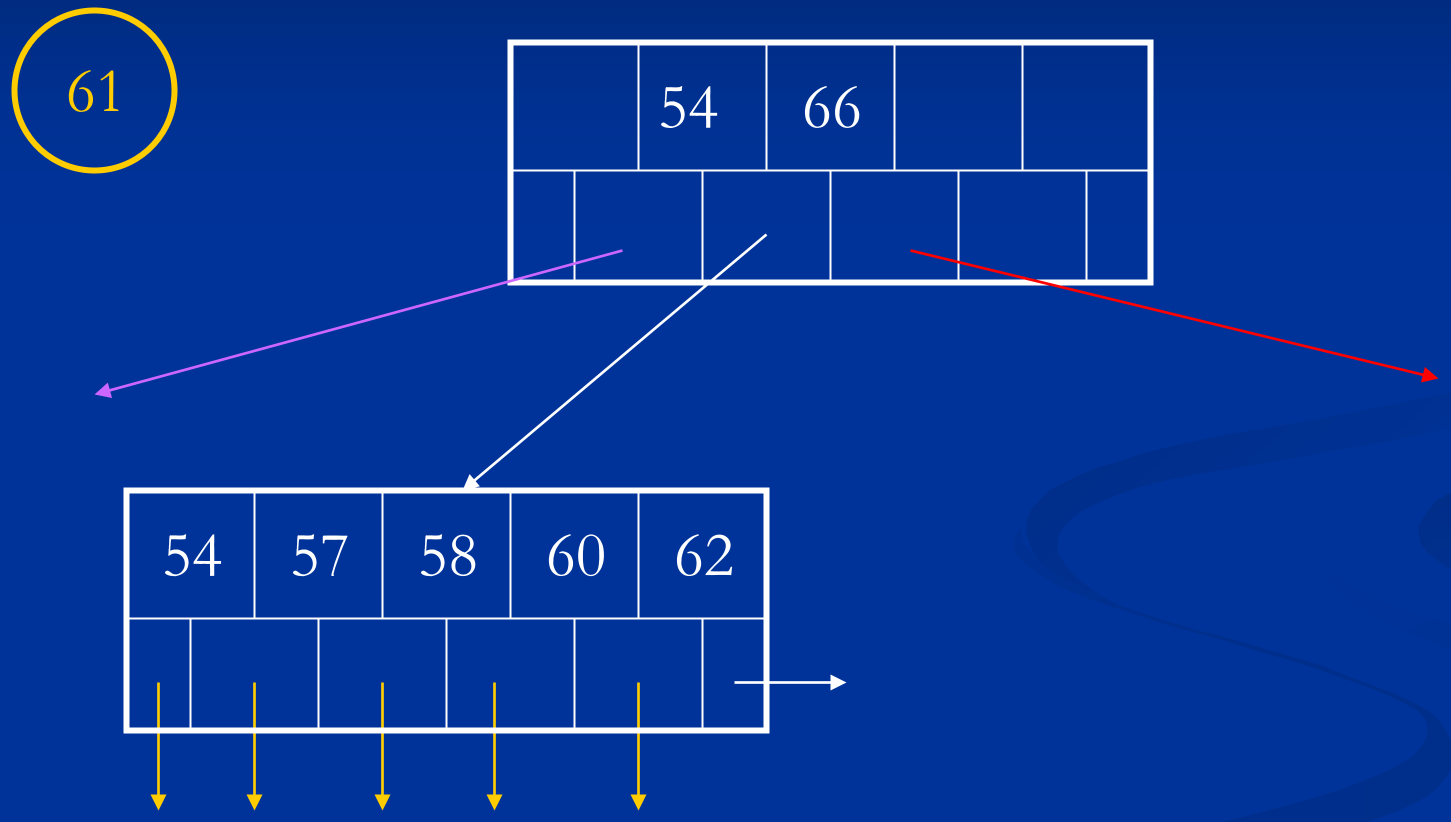
Inserting into a Leaf Node



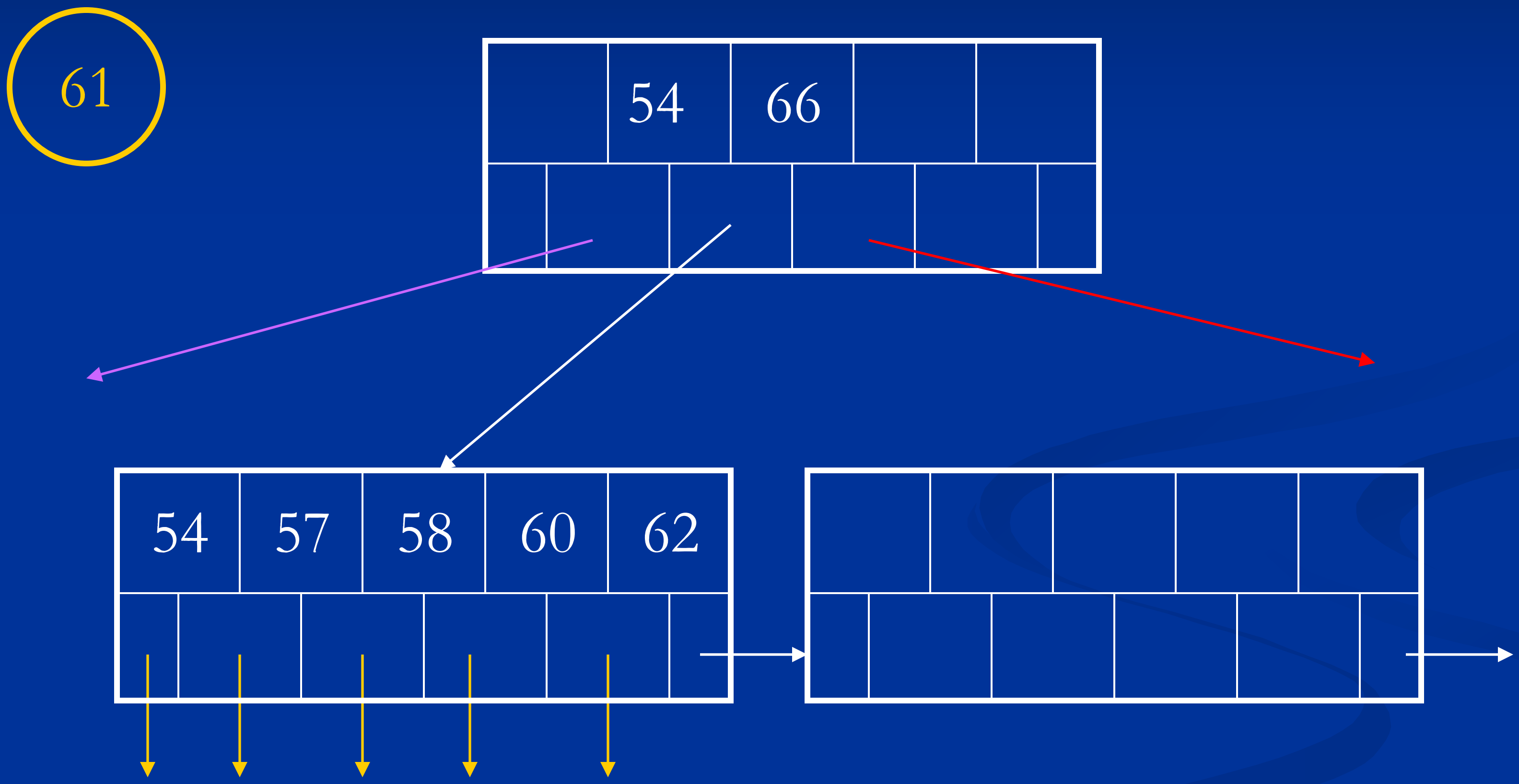
Inserting into a Leaf Node



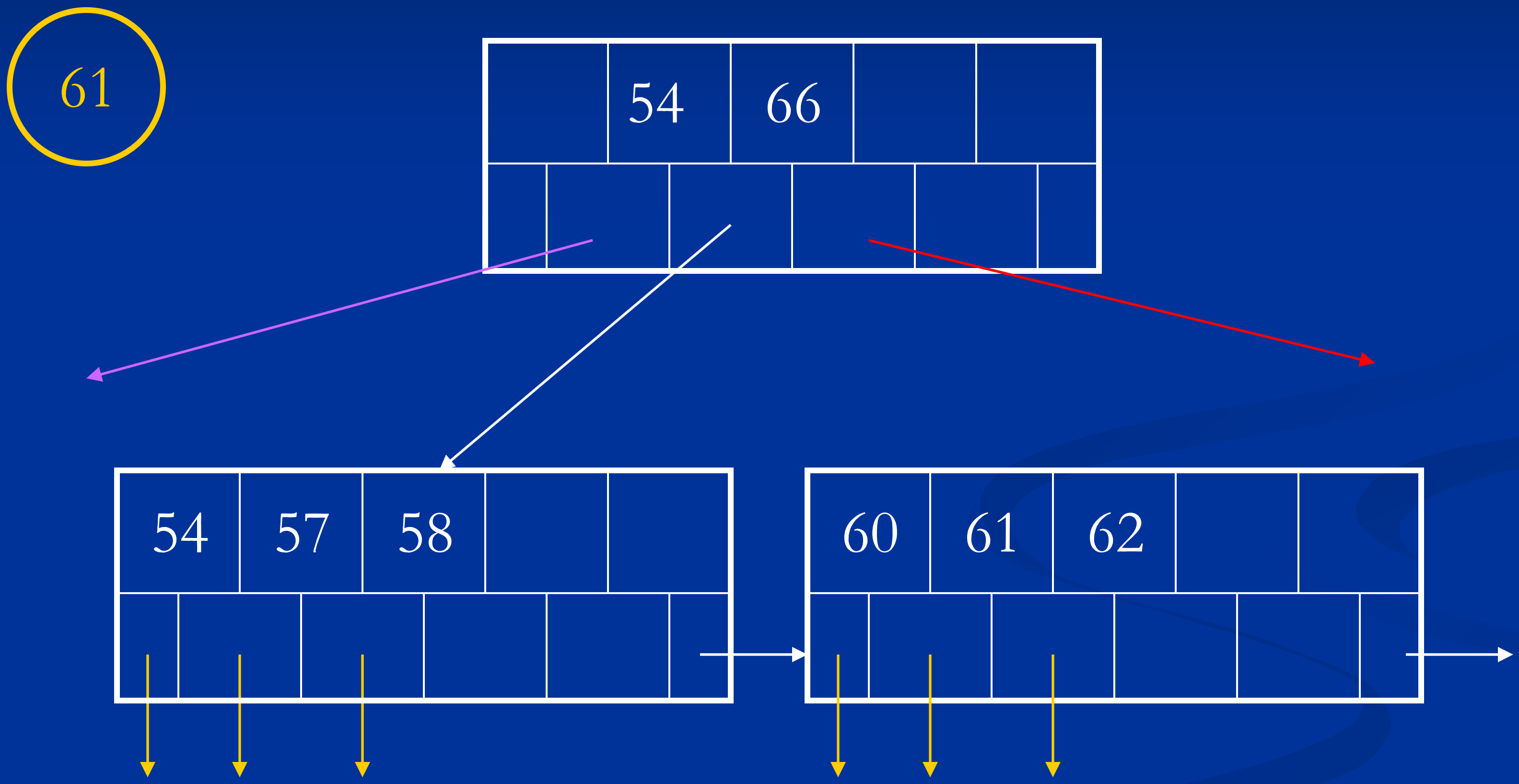
Splitting a Leaf Node



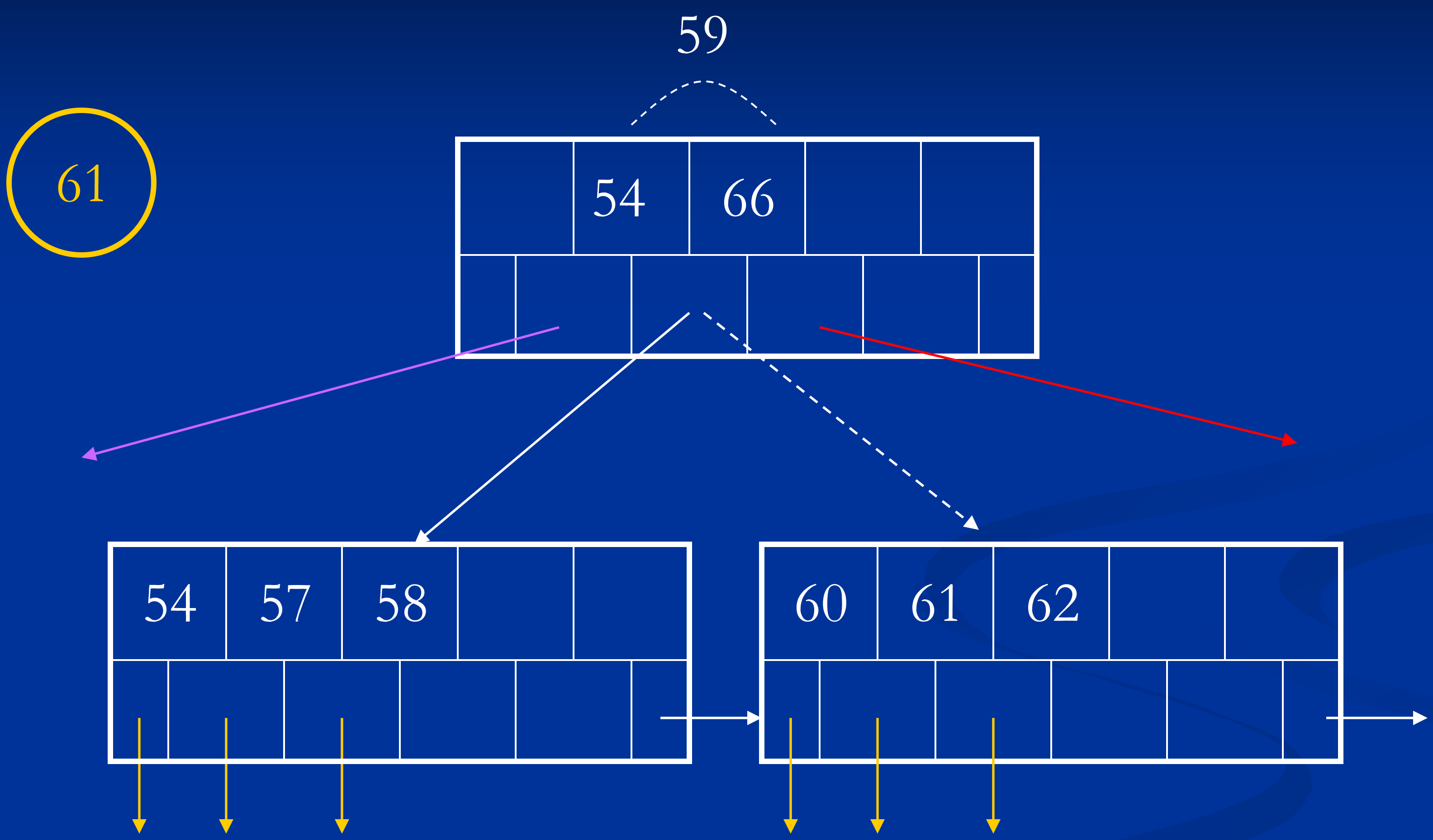
Splitting a Leaf Node



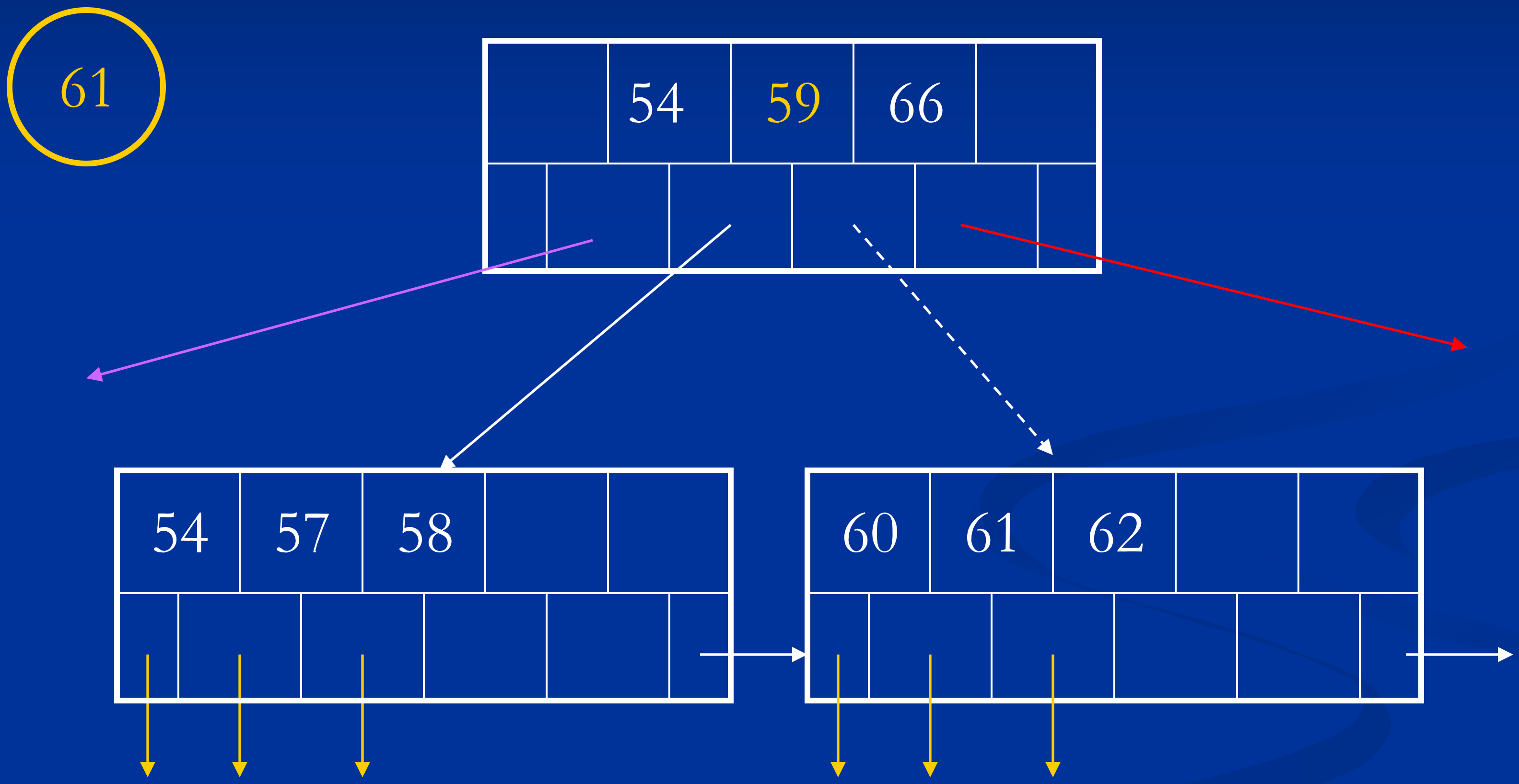
Splitting a Leaf Node



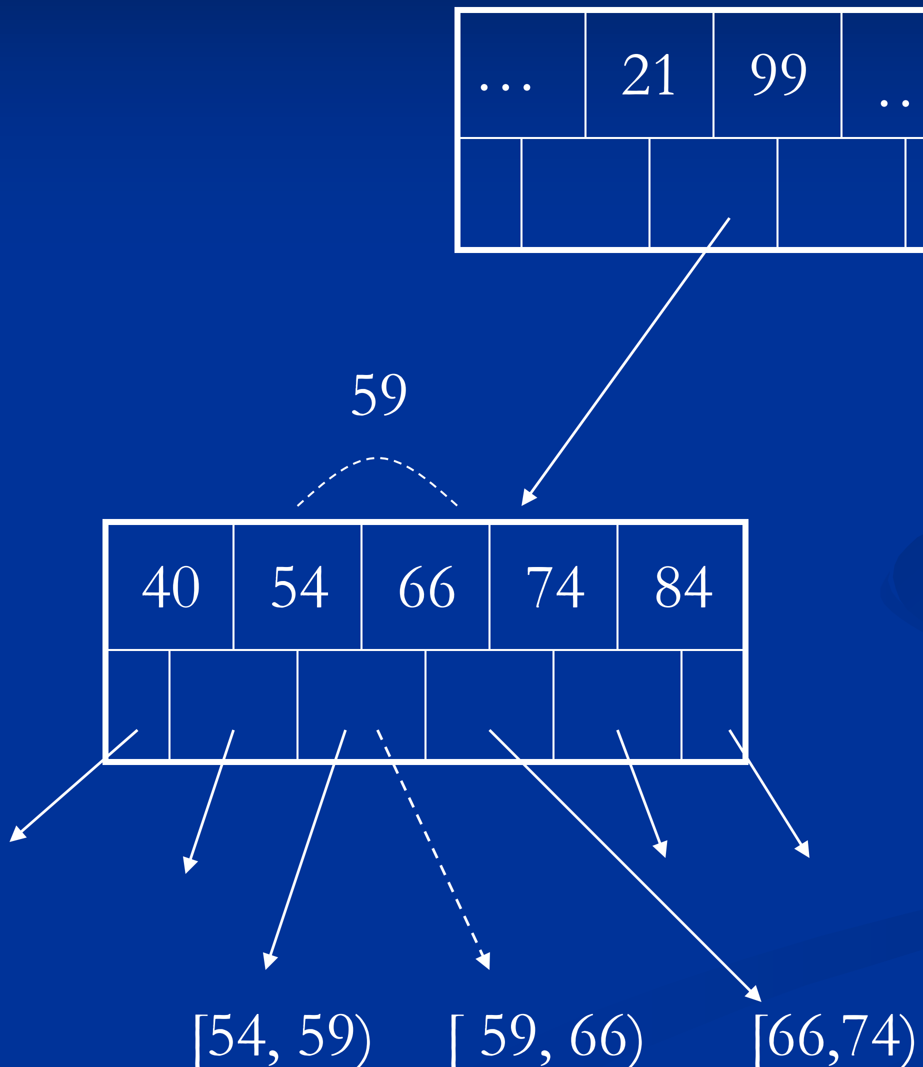
Splitting a Leaf Node



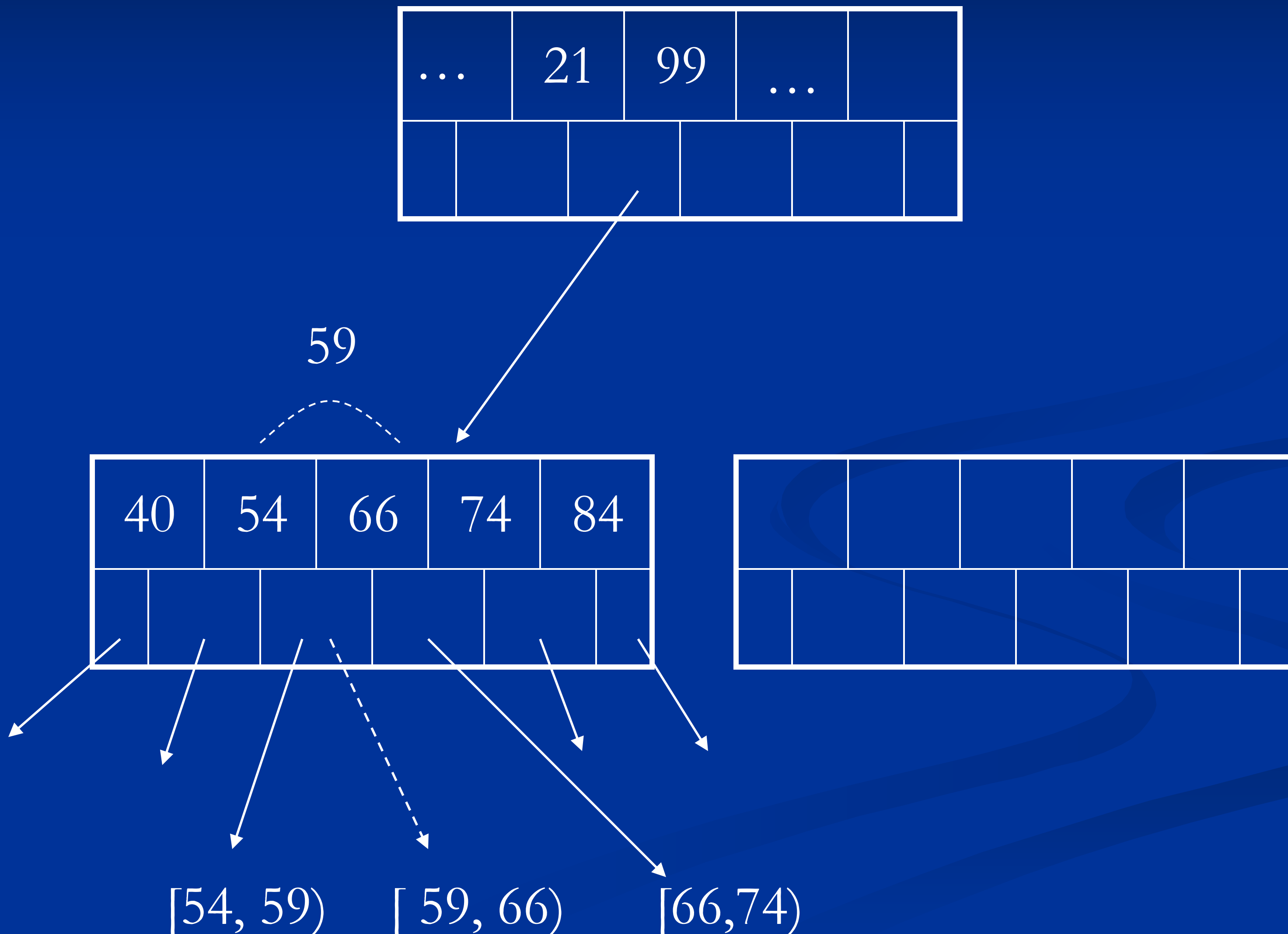
Splitting a Leaf Node



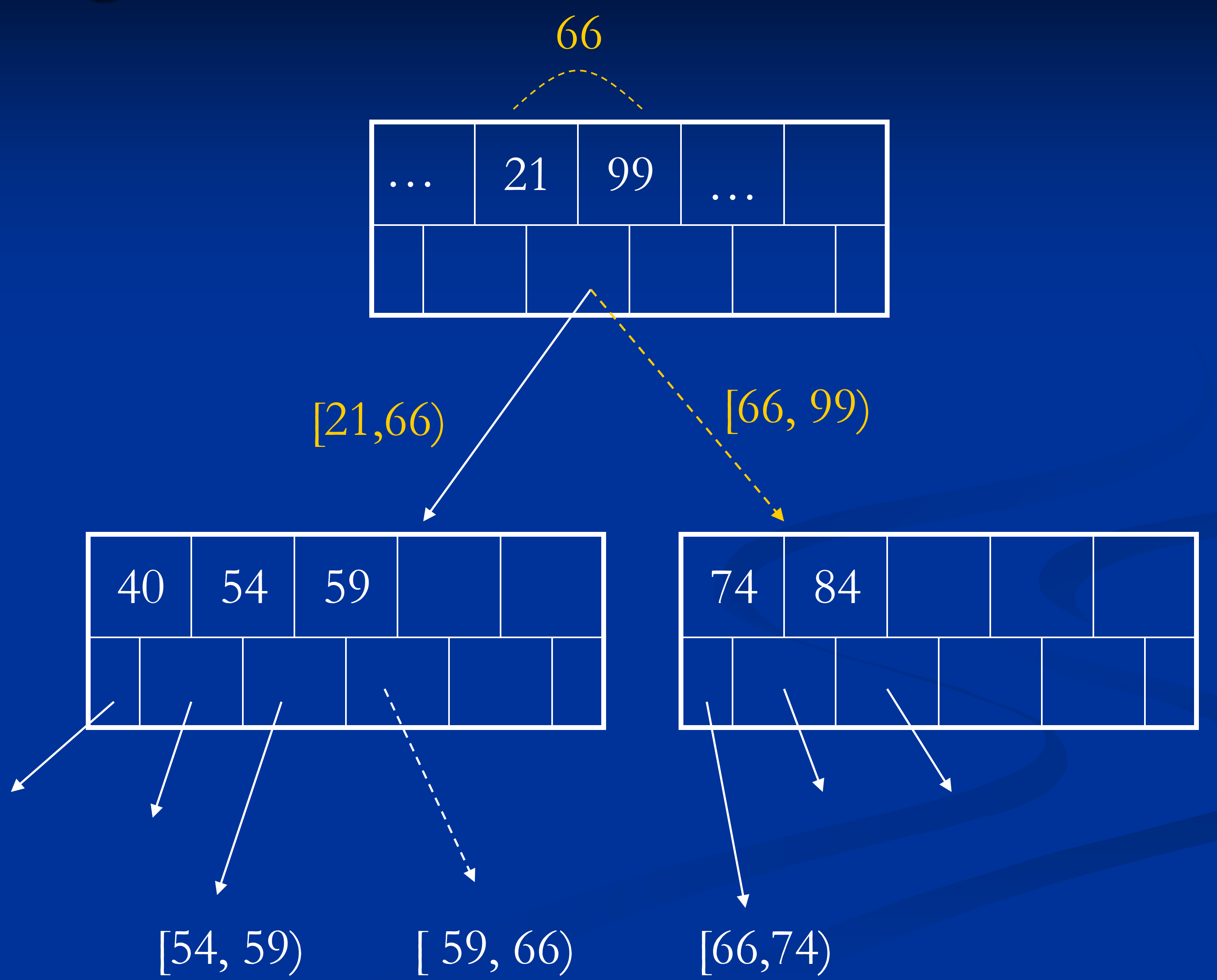
Splitting an Internal Node



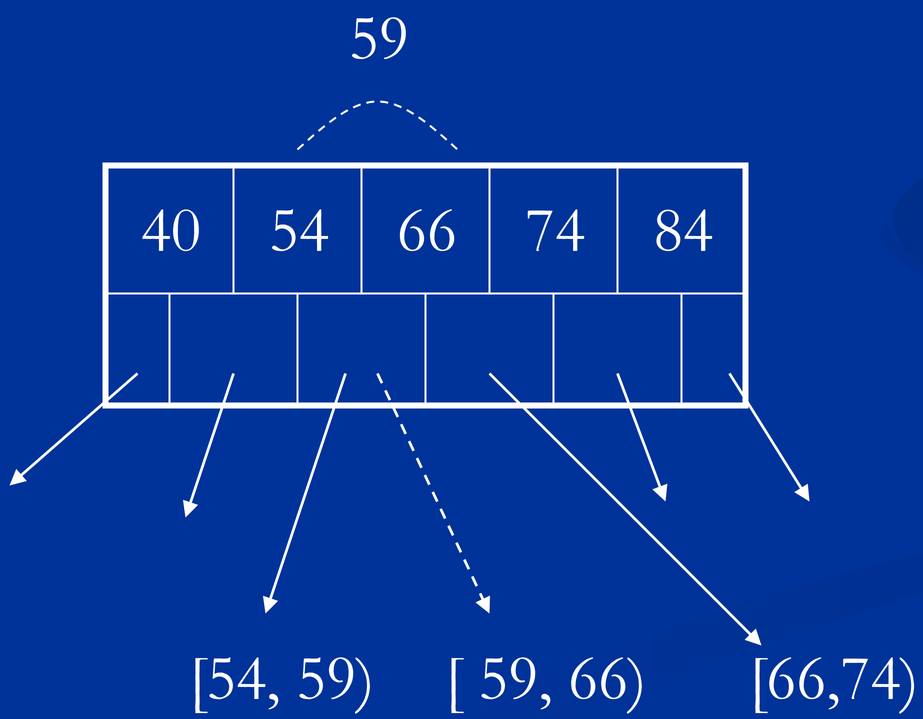
Splitting an Internal Node



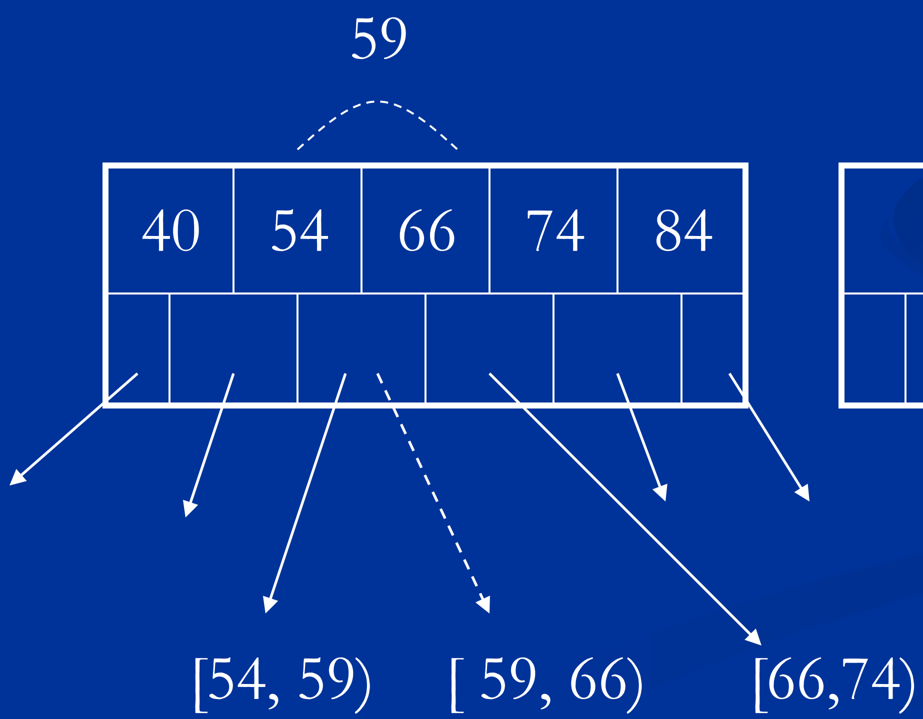
Splitting an Internal Node



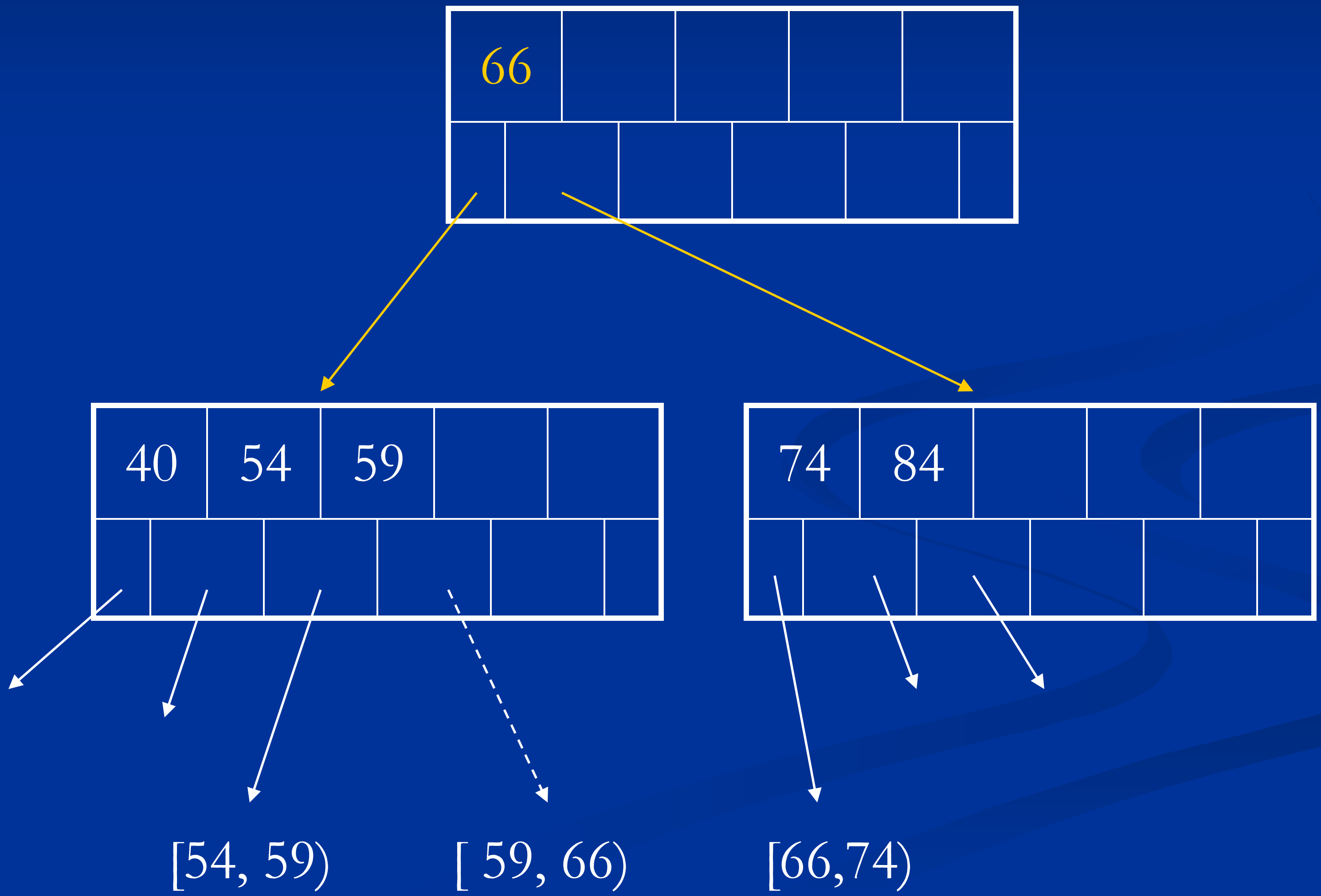
Splitting the Root



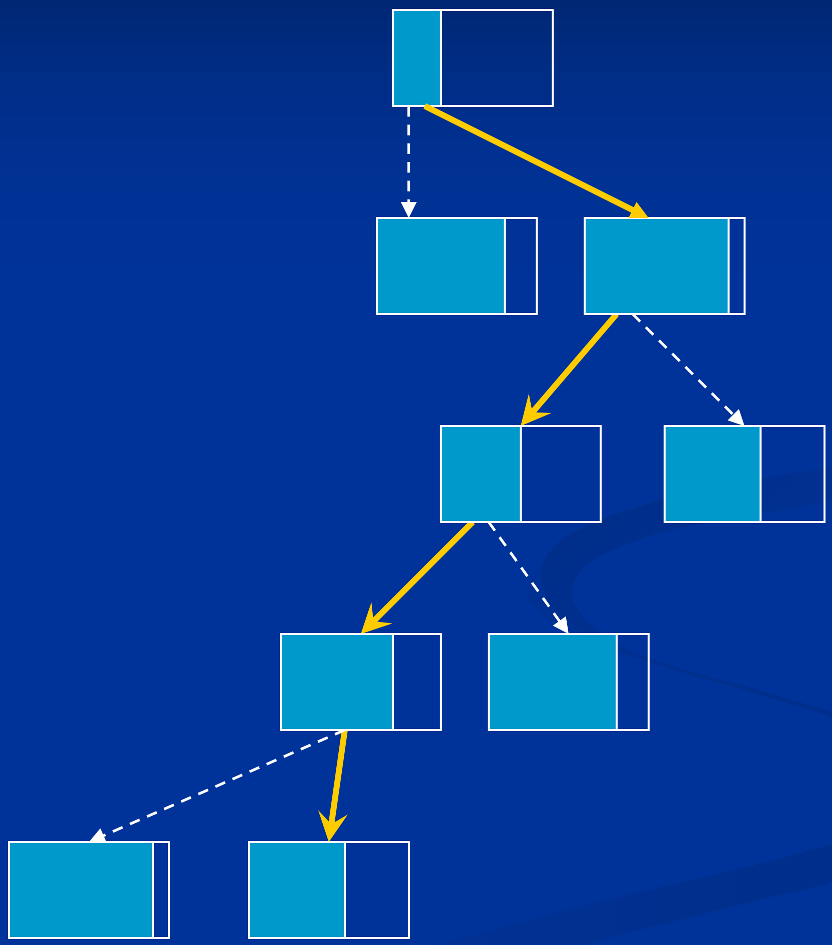
Splitting the Root



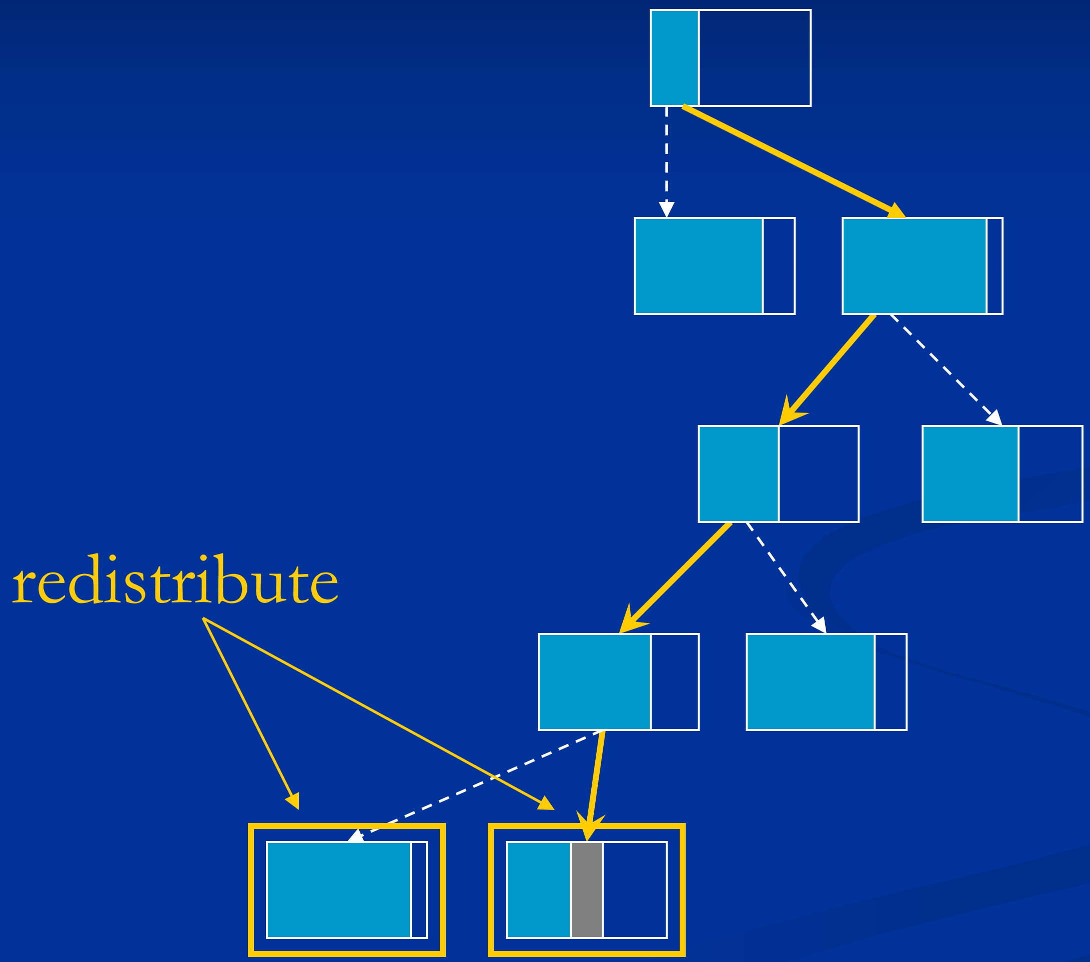
Splitting the Root



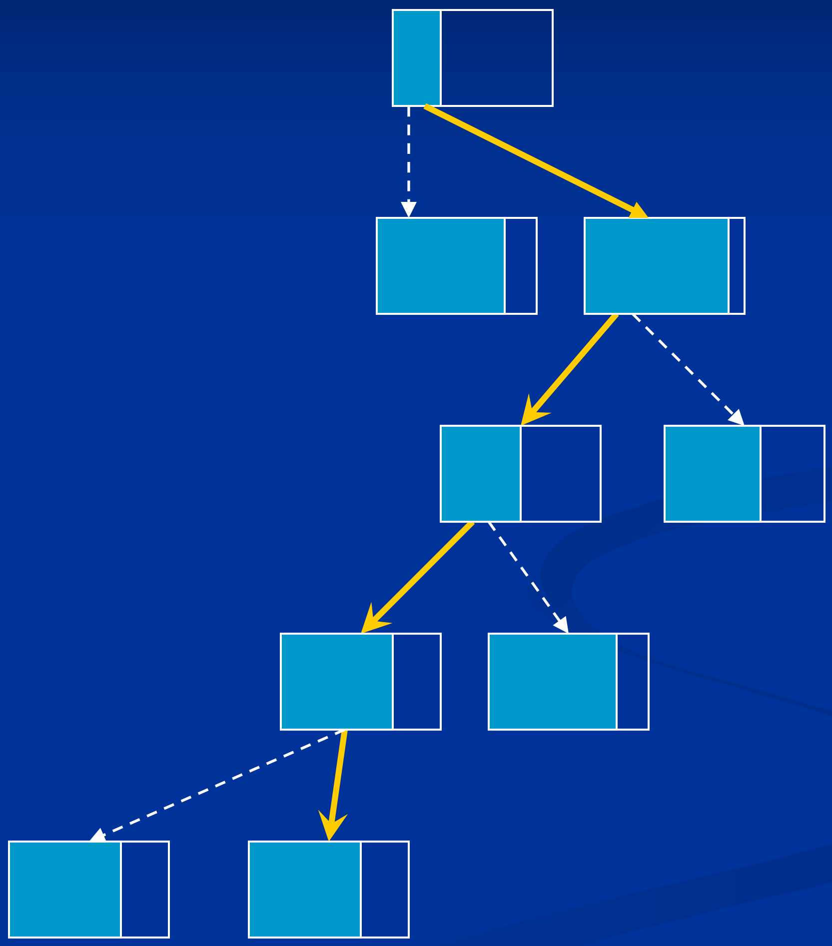
Deletion



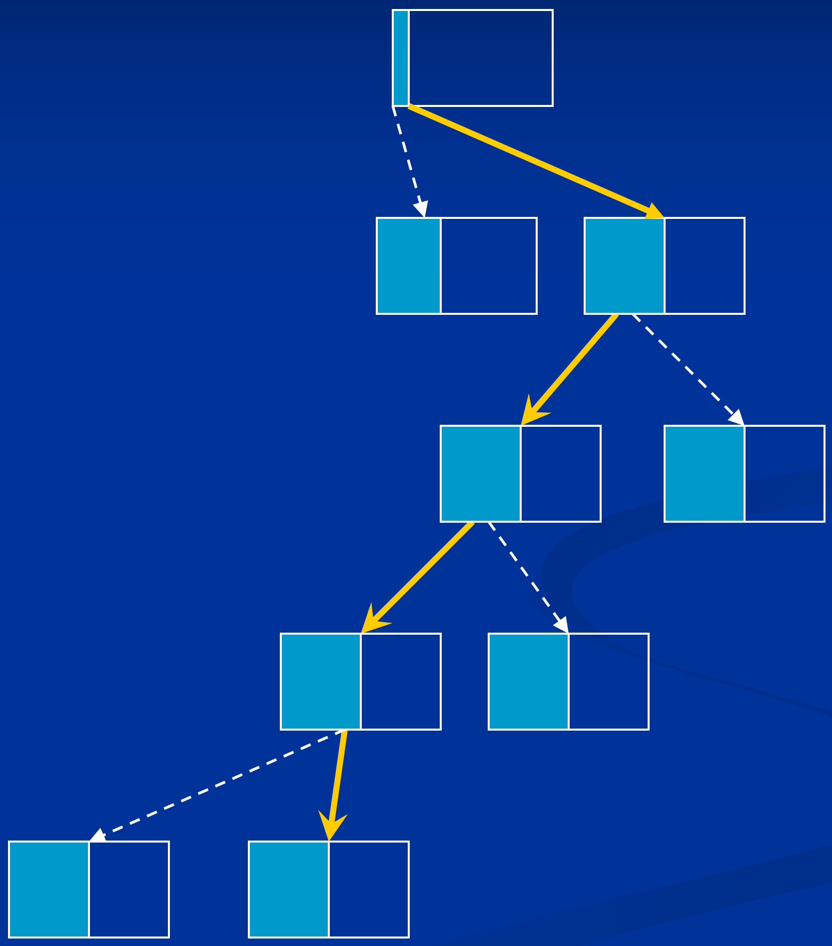
Deletion



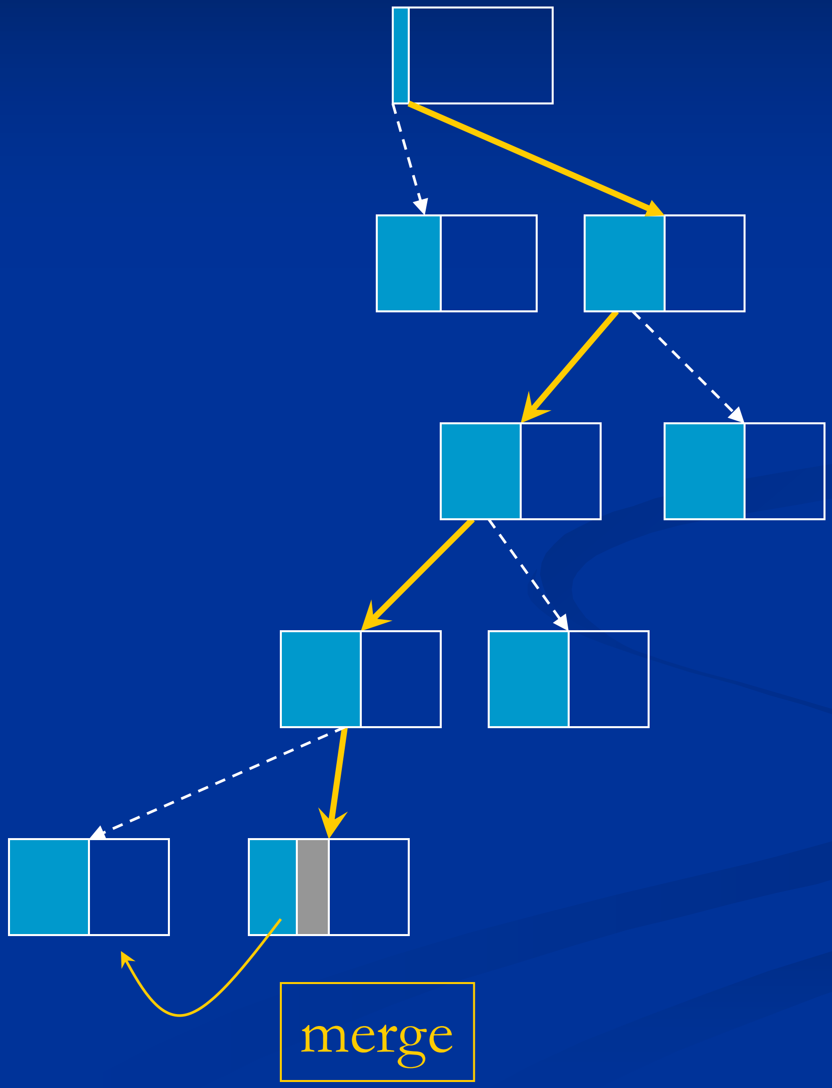
Deletion



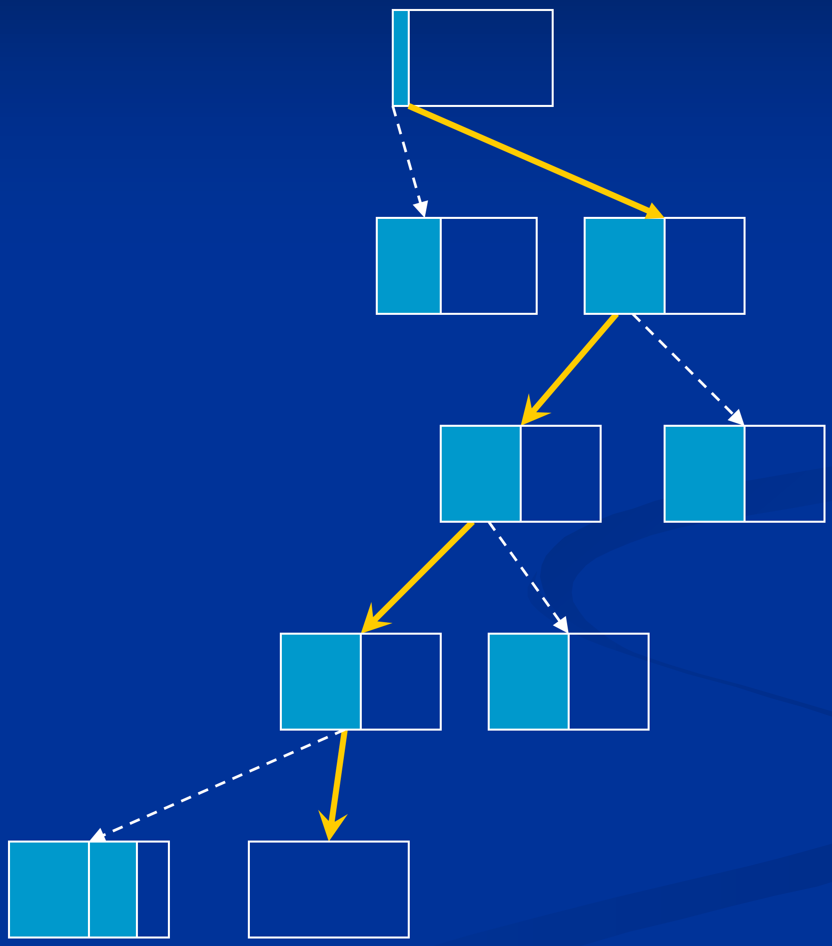
Deletion - II



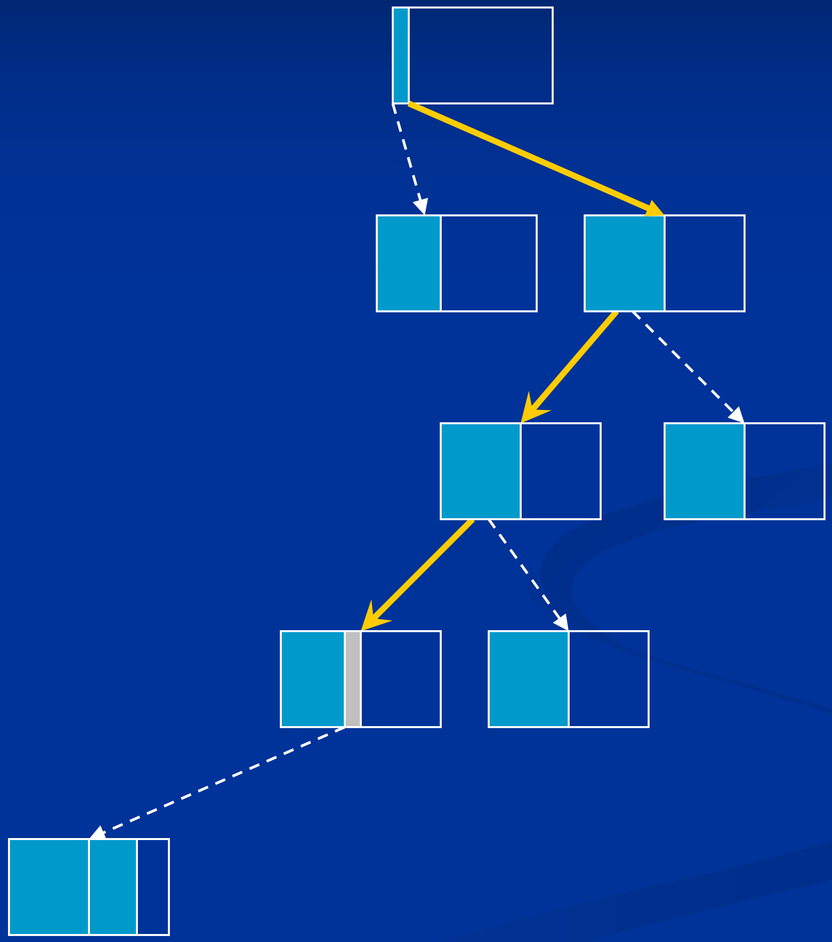
Deletion - II



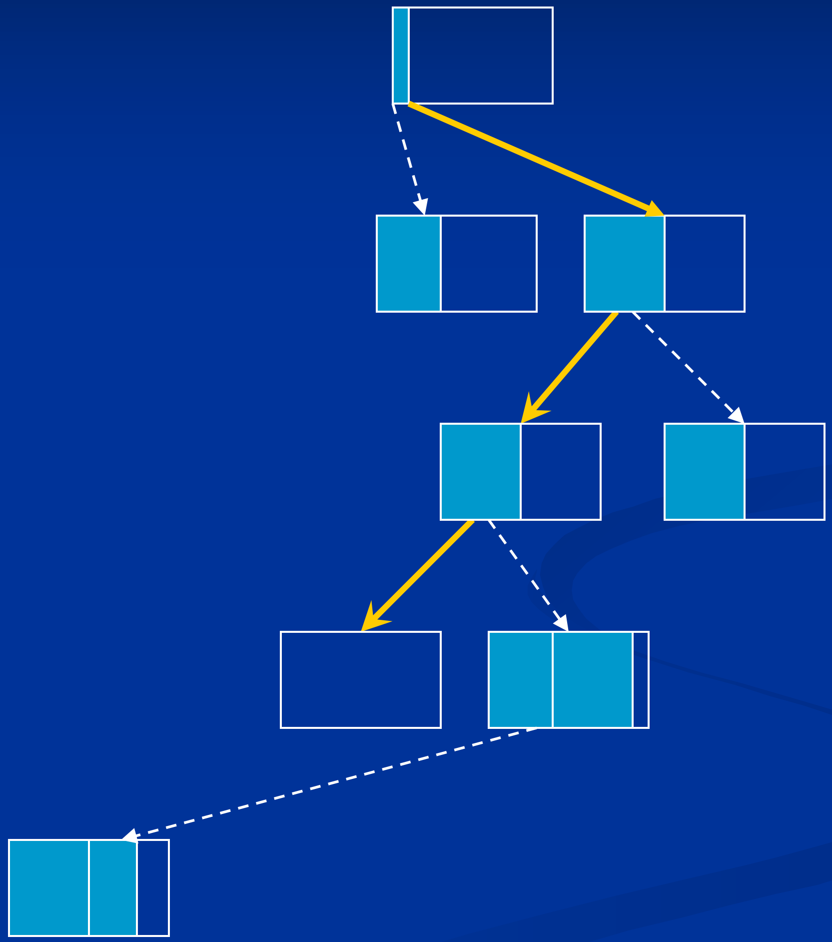
Deletion - II



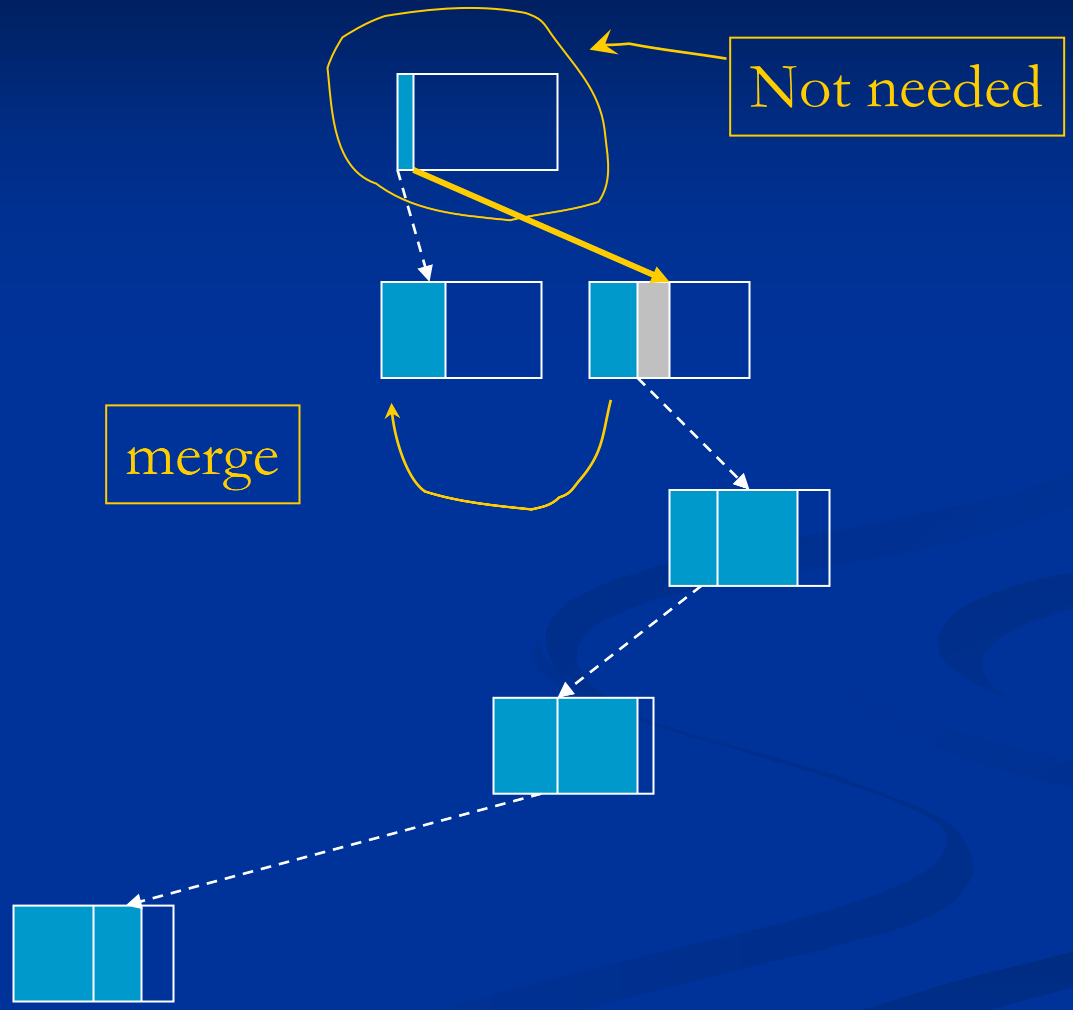
Deletion - II



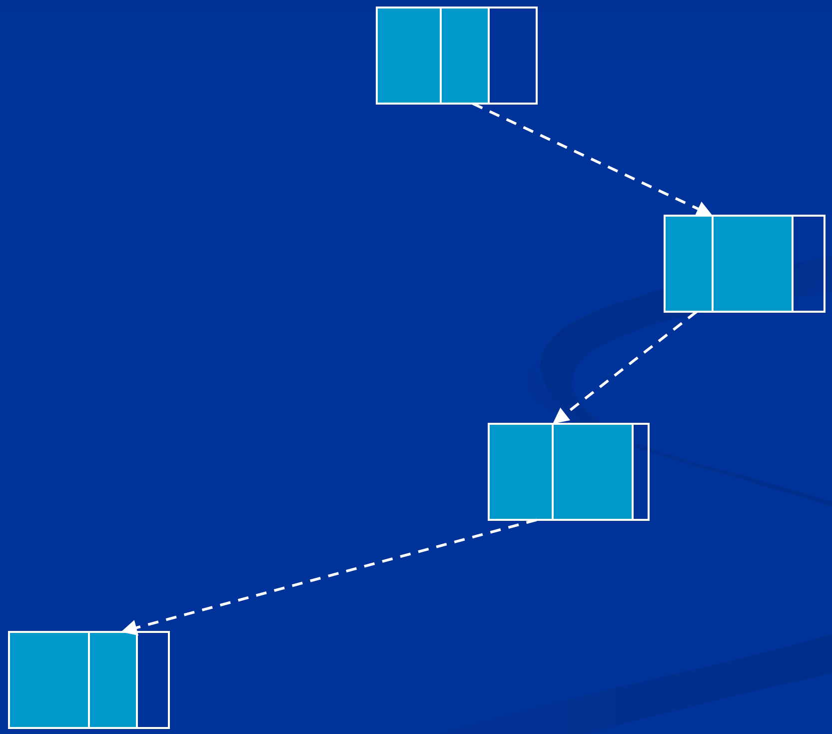
Deletion - II



Deletion - II



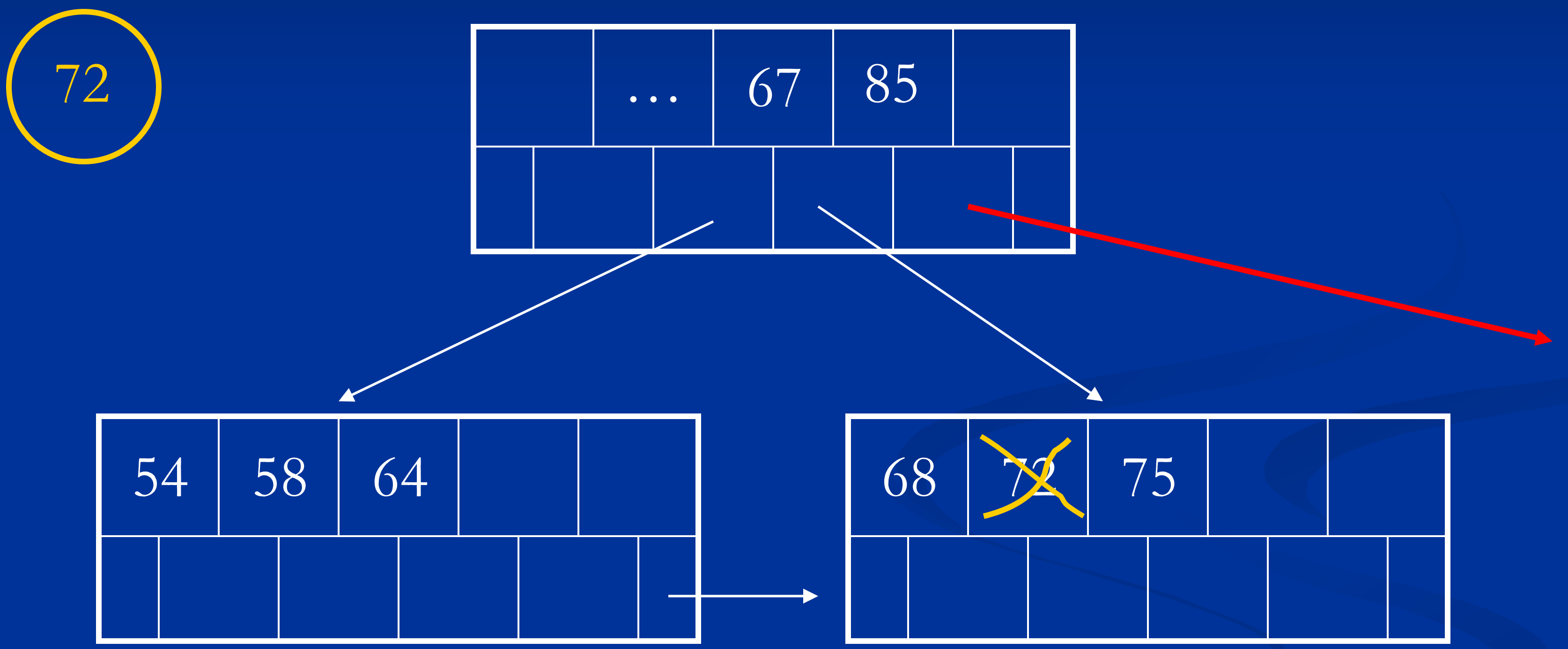
Deletion - II



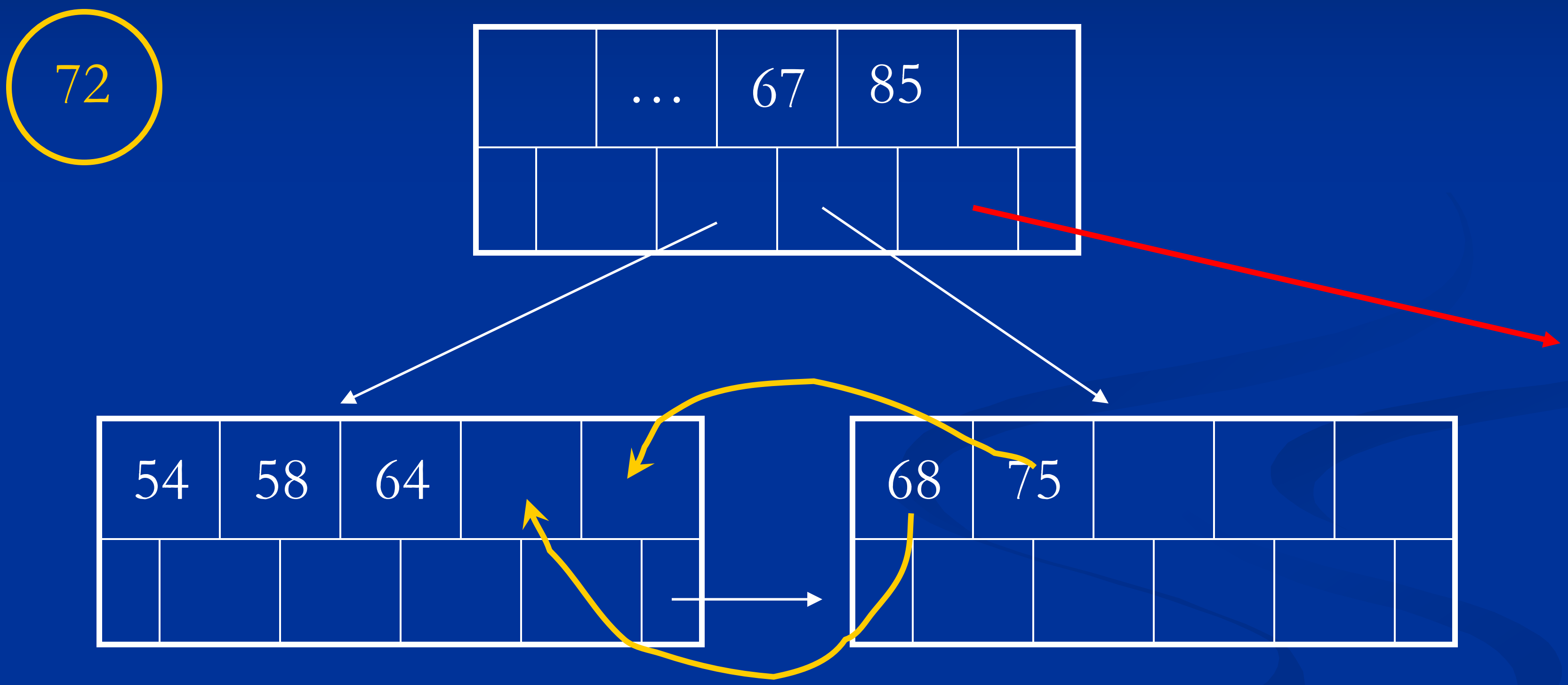
Deletion: Primitives

- Delete key from a leaf
- Redistribute keys between sibling leaves
- ➡ Merge a leaf into its sibling
- Redistribute keys between two sibling internal nodes
- ➡ Merge an internal node into its sibling

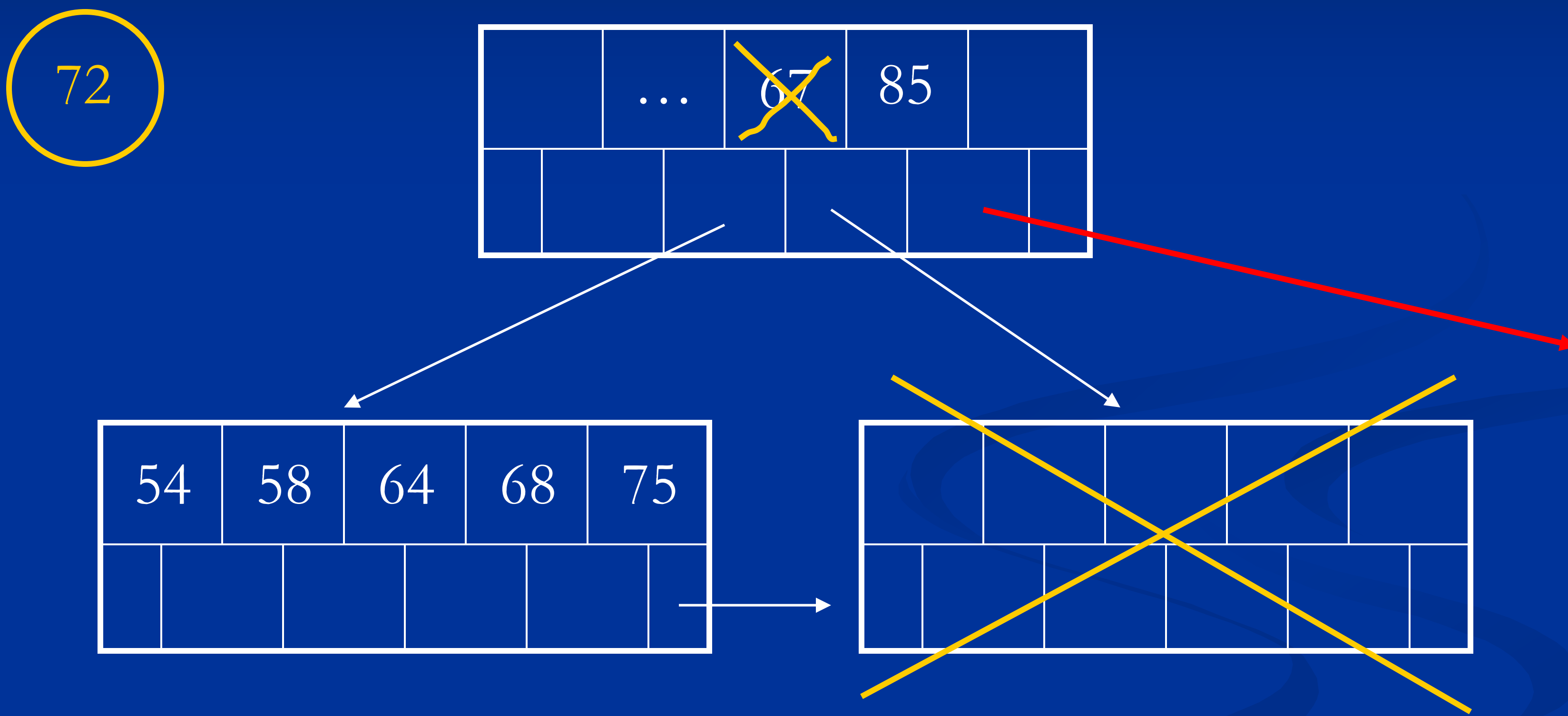
Merge Leaf into Sibling



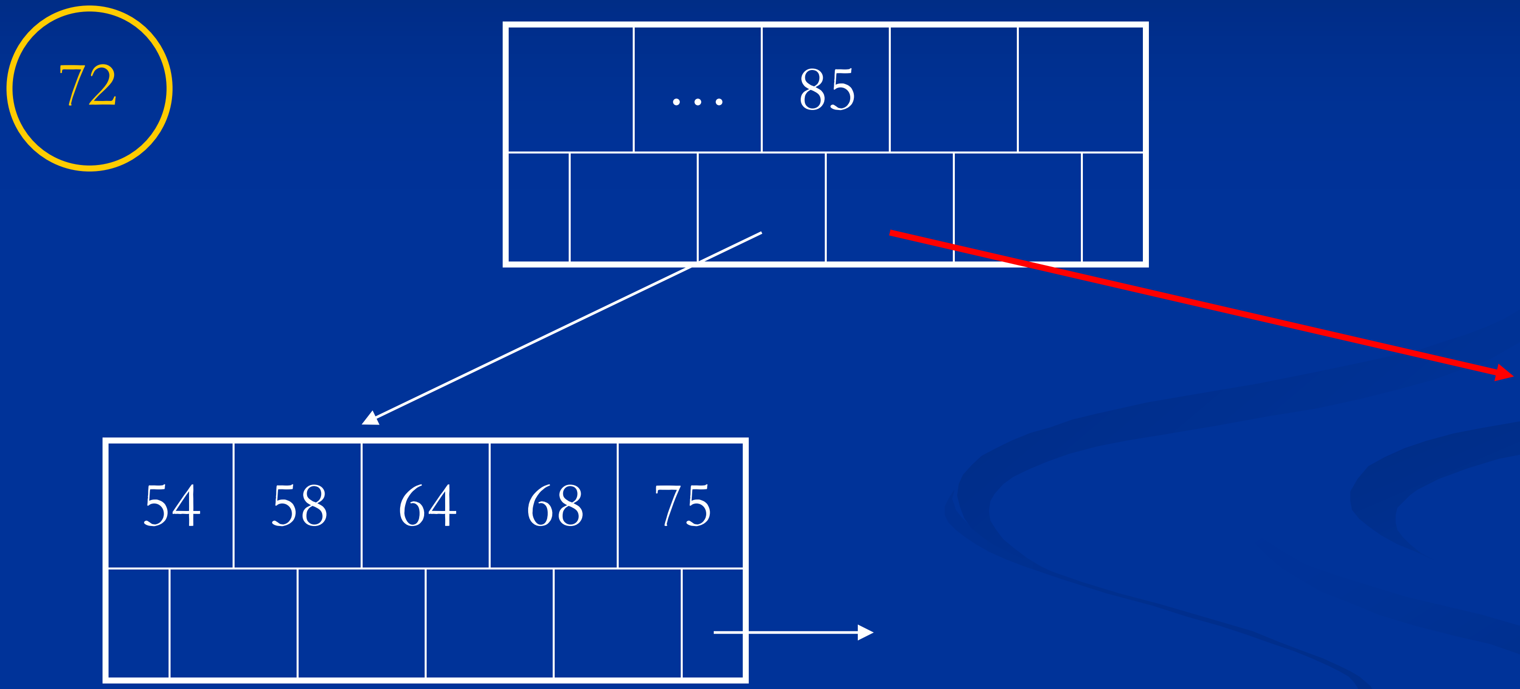
Merge Leaf into Sibling



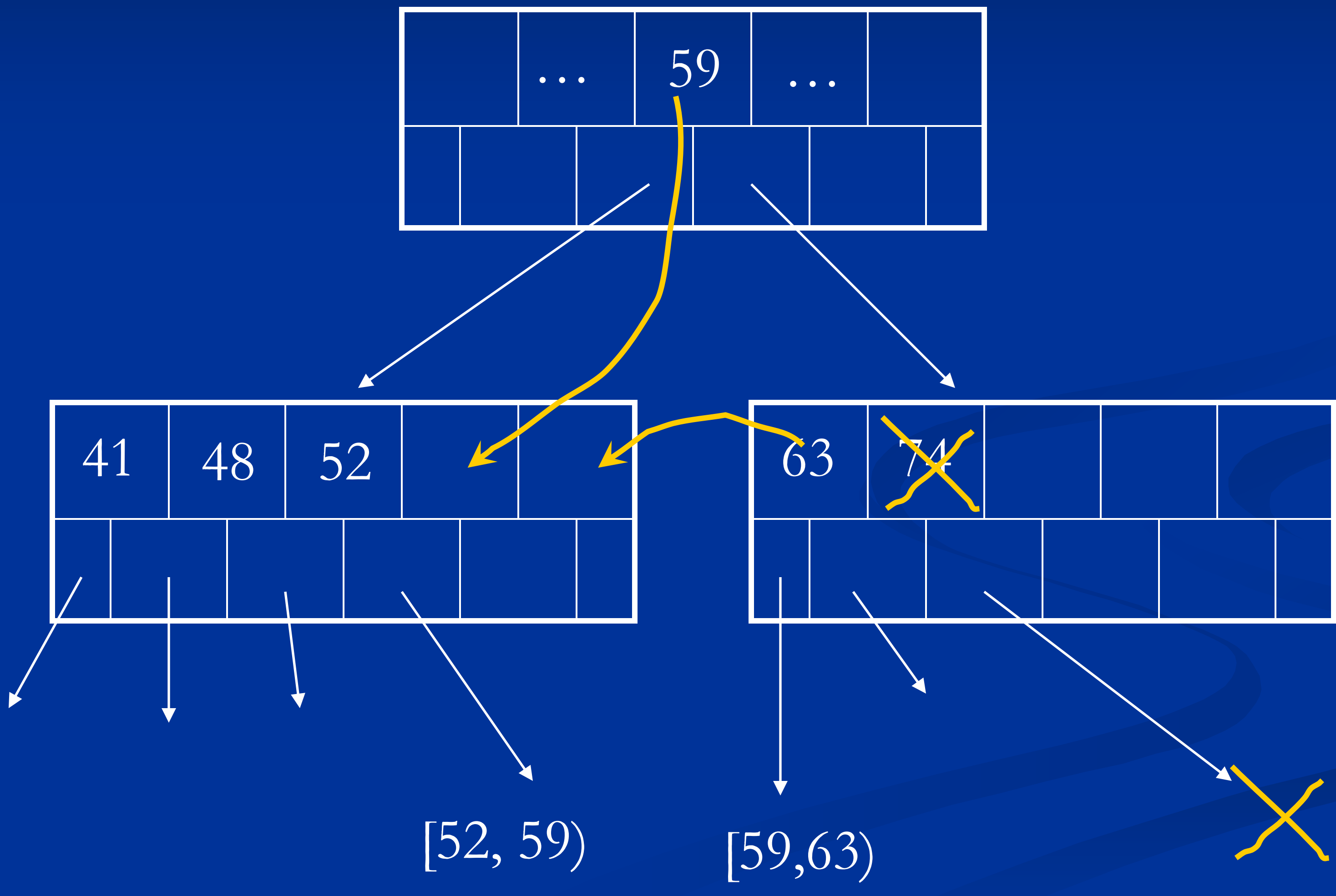
Merge Leaf into Sibling



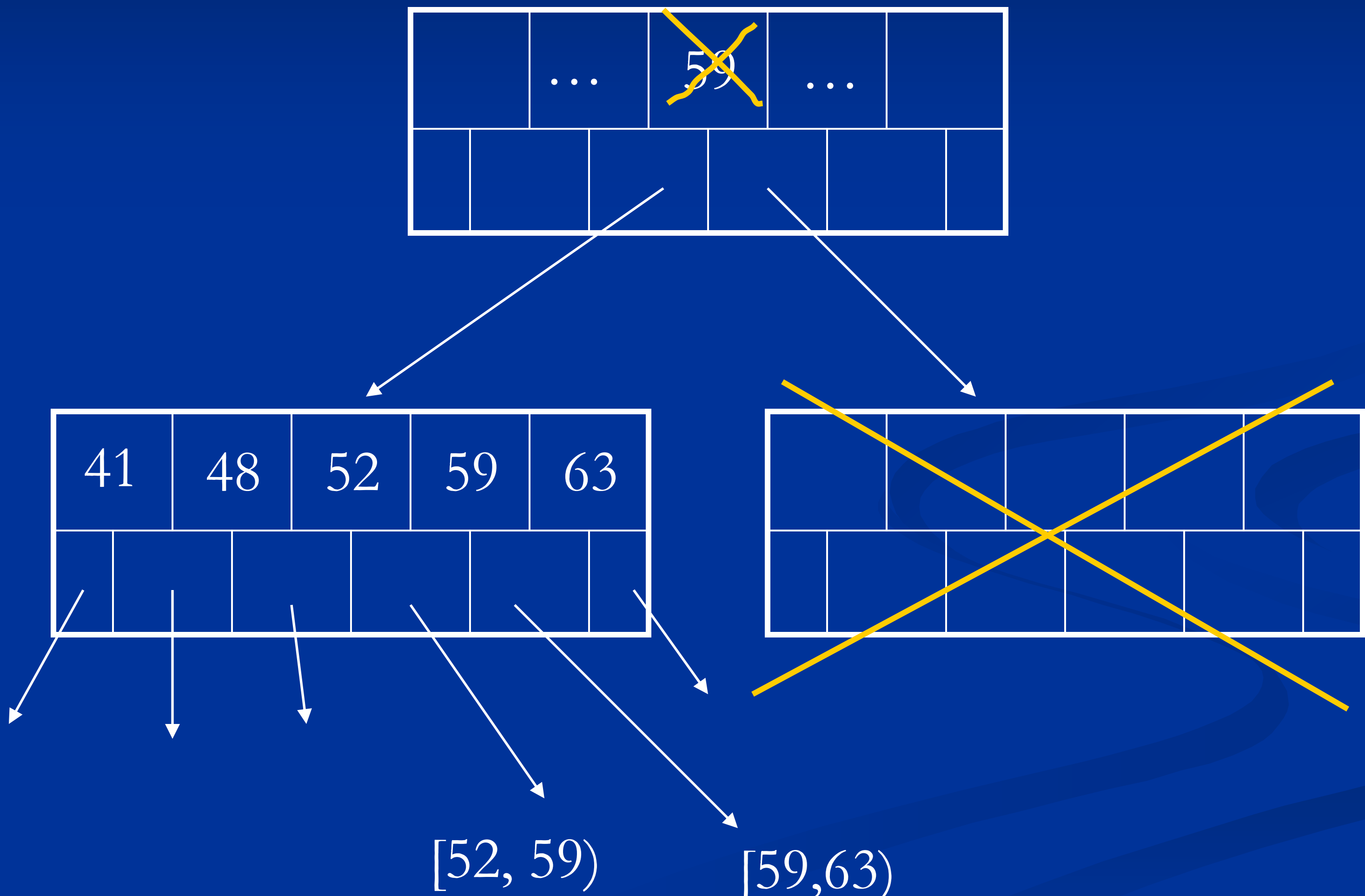
Merge Leaf into Sibling



Merge Internal Node into Sibling



Merge Internal Node into Sibling



B-Tree Roadmap

- B-Tree
  - Recap
  - Insertion (recap)
  - Deletion
  - Construction
  - Efficiency
- B-Tree variants
- Hash-based Indexes

Question

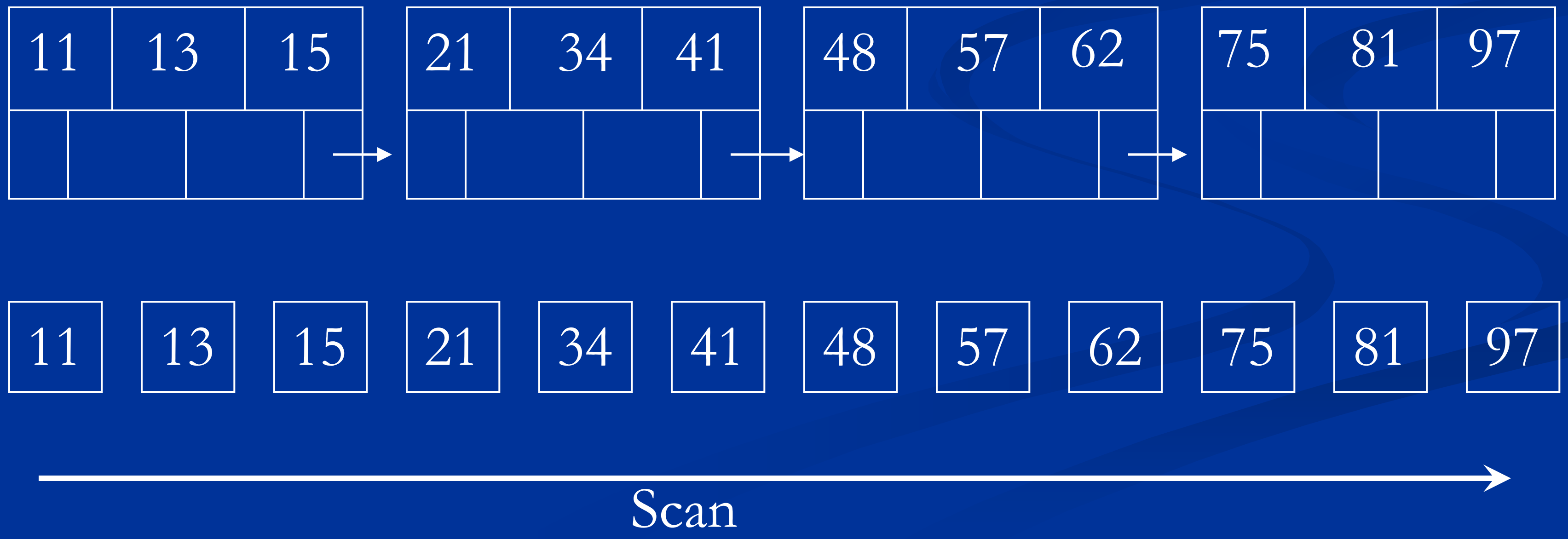
How does insertion-based construction perform?

B-Tree Construction

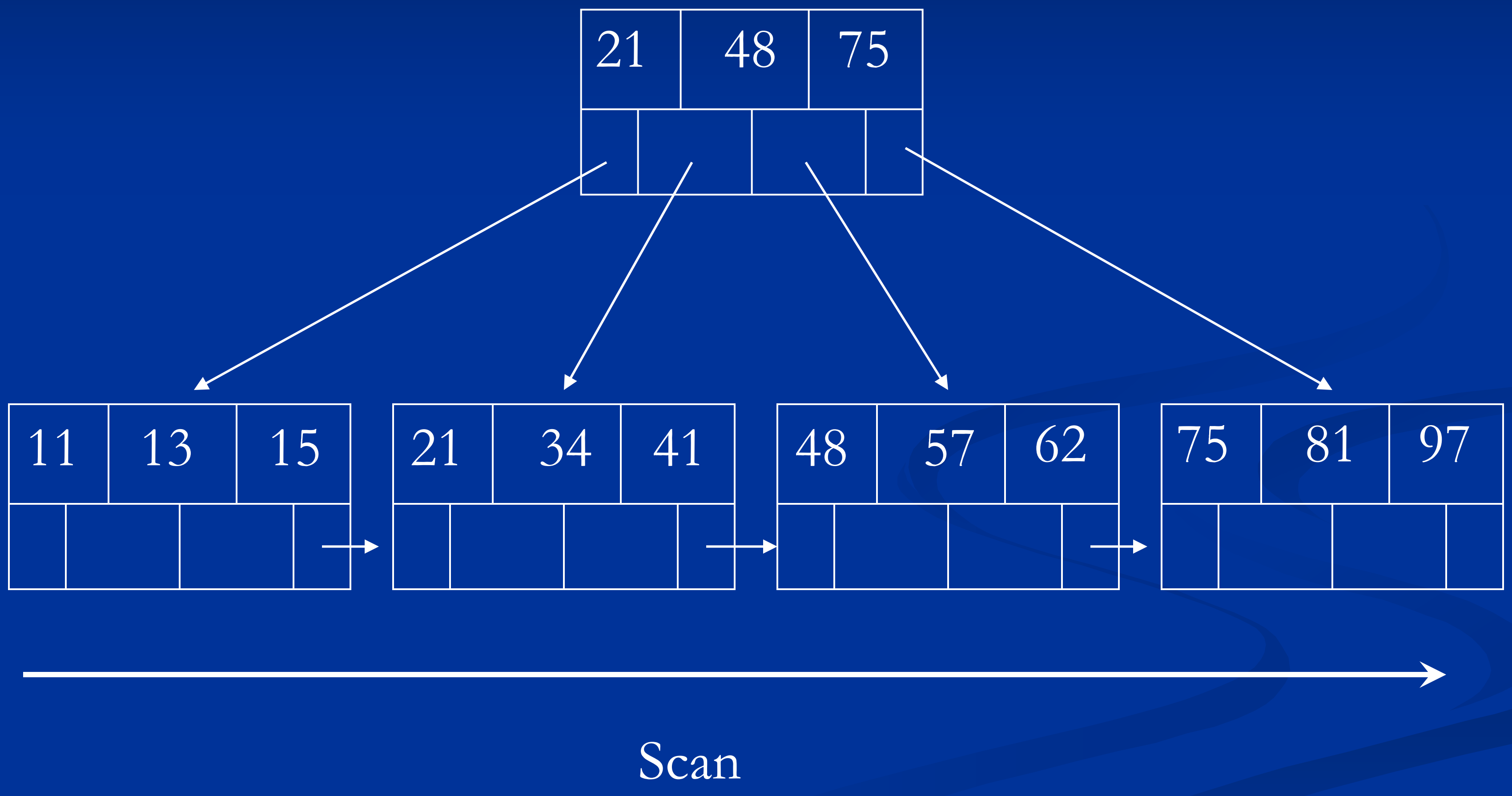
Sort

48	57	41	15	75	21	62	34	81	11	97	13

B-Tree Construction



B-Tree Construction



B-Tree Construction

Why is sort-based construction better than insertion-based one?

Cost of B-Tree Operations

- Height of B-Tree: H
- Assume no duplicates
- Question: what is the random I/O cost of:
  - Insertion:
  - Deletion:
  - Equality search:
  - Range Search:

Height of B-Tree

- Number of keys: N
- B-Tree parameter: n

$$\text{Height} \approx \log_n N = \frac{\log N}{\log n}$$

In practice: 2-3 levels

Question: How do you pick parameter n?

1. Ignore inserts and deletes
2. Optimize for equality searches
3. Assume no duplicates

Roadmap

- B-Tree
- B-Tree variants
  - Sparse Index
  - Duplicate Keys
- Hash-based Indexes

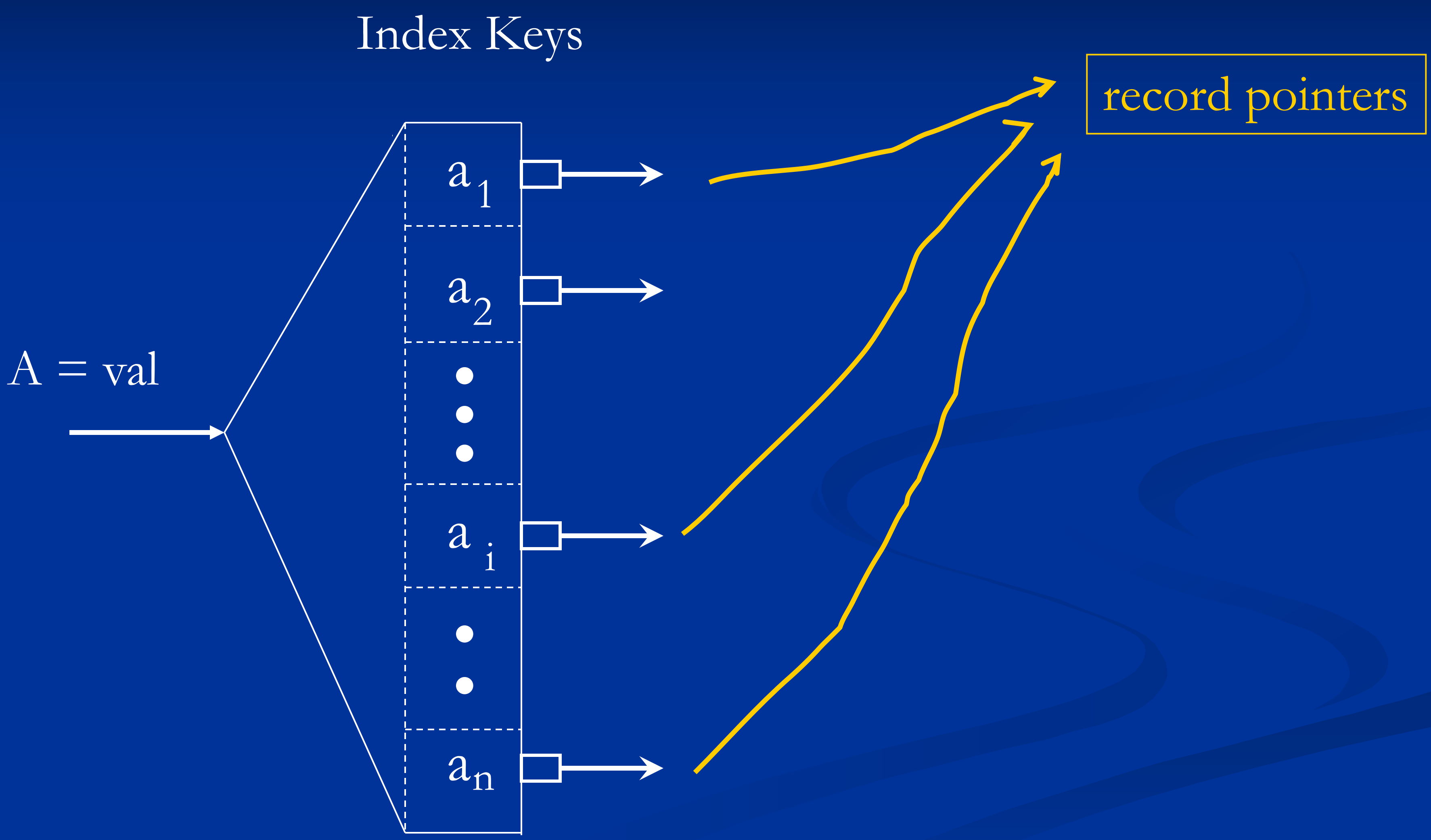
Roadmap

- B-Tree
- B-Tree variants
- Hash-based Indexes
  - Static Hash Table
  - Extensible Hash Table
  - Linear Hash Table

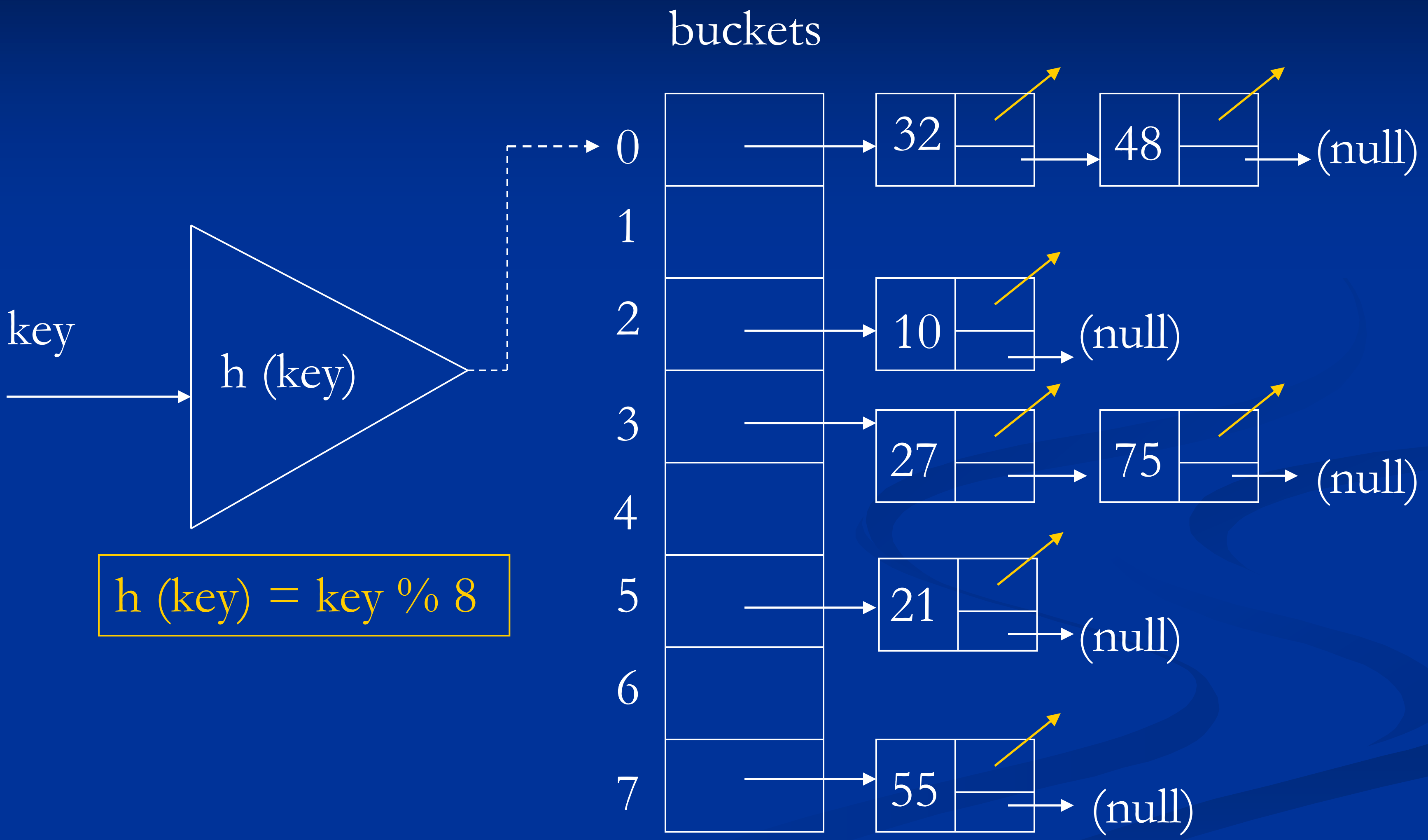
Hash-Based Indexes

- Adaptations of main memory hash tables
- Support equality searches
- No range searches

Indexing Problem (recap)



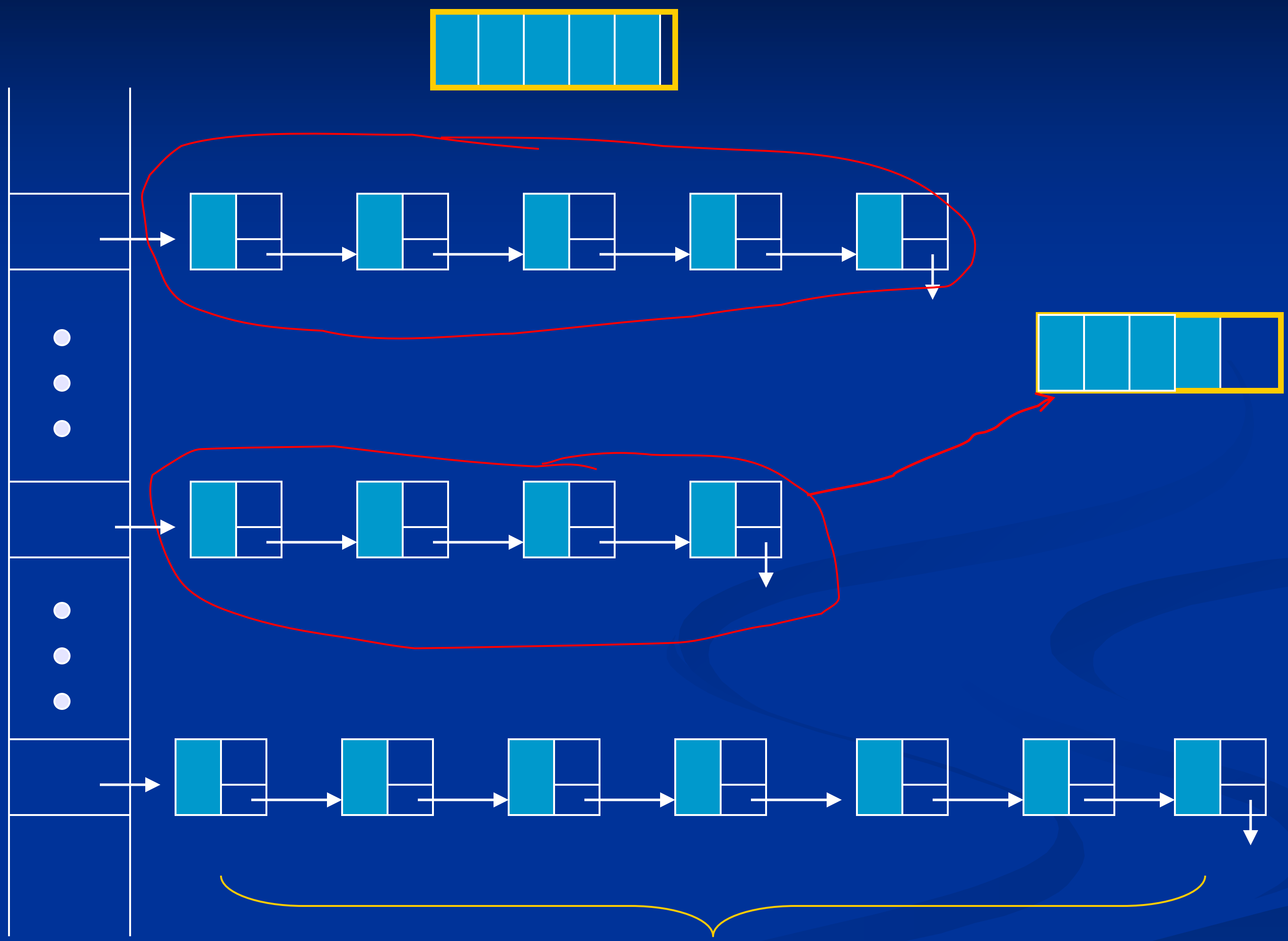
Main Memory Hash Table



Adapting to disk

- 1 Hash Bucket = 1 Block
  - All keys that hash to bucket stored in the block
  - Intuition: keys in a bucket usually accessed together
  - No need for linked lists of keys …

Adapting to Disk

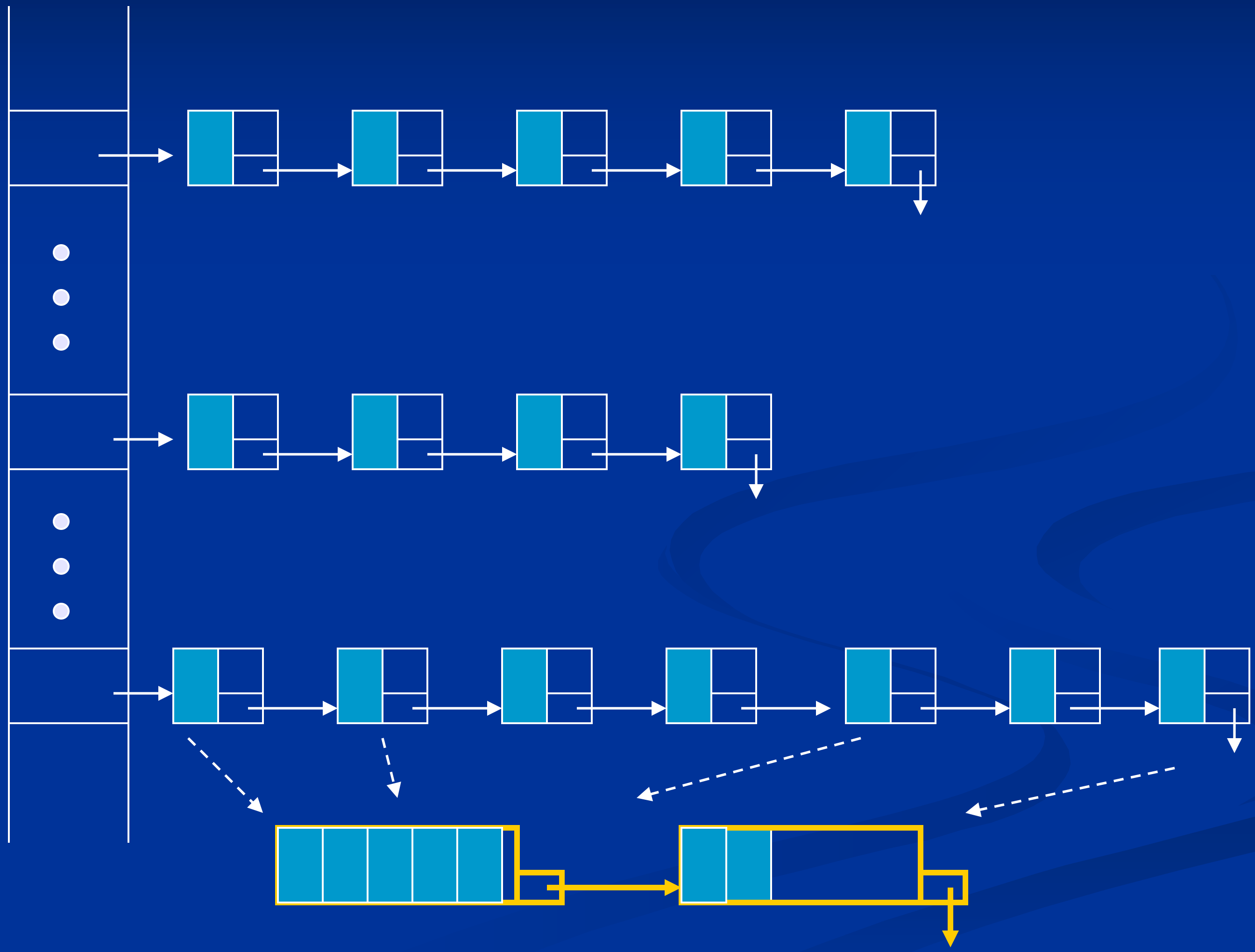


How do we handle this?

Adapting to disk

- 1 Hash Bucket = 1 Block
  - All keys that hash to bucket stored in the block
  - Intuition: keys in a bucket usually accessed together
  - No need for linked lists of keys …
  - … but need linked list of blocks (overflow blocks)

Adapting to Disk



Adapting to disk

- Bucket Id → Disk Address mapping
  - Contiguous blocks
  - Store mapping in main memory
    - Too large?
    - Dynamic → Linear and Extensible hash tables

Beware of claims that assume 1 I/O for hash tables and 3 I/Os for B-Tree!!