PRINCIPLES OF DATA ORGANISATION

Hierarchical Indexing – Advanced

MOTIVATION

- Key, pointer pairs ~ index
- B-tree
    - Balanced tree
    - Node = page/block
    - Redundant/non-redundant

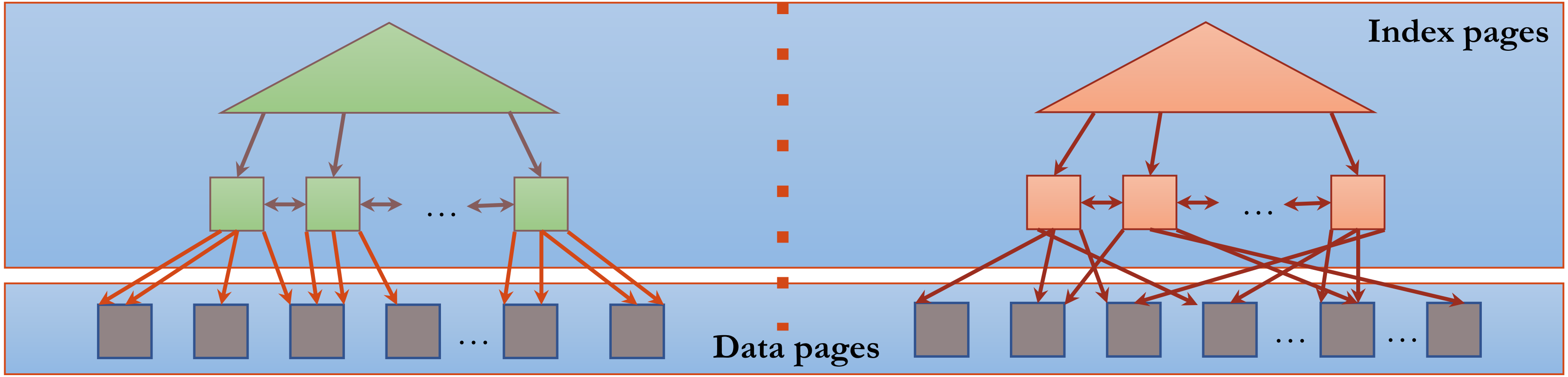
CLUSTERED VS. NONCLUSTERED INDEXES

Clustered index

- Corresponds to the idea of index-sequential file organization
- Logical order of the key values determines the physical order of the corresponding data records
- Only one
- Fast range queries

Nonclustered index

- Order of data in the index and the primary file is not related
- Multiple nonclustered indexes can exist



SPARSE VS. DENSE INDEXES

Sparse index

- Entry for each page/block
- Clustered index – data in a page/block the data is sorted

- Note: Clustered index can be sparse or dense

Dense index

- Entry for every data record
- Nonclustered (non-primary) index must be dense

CLUSTERED VS NONCLUSTERED INDEXES

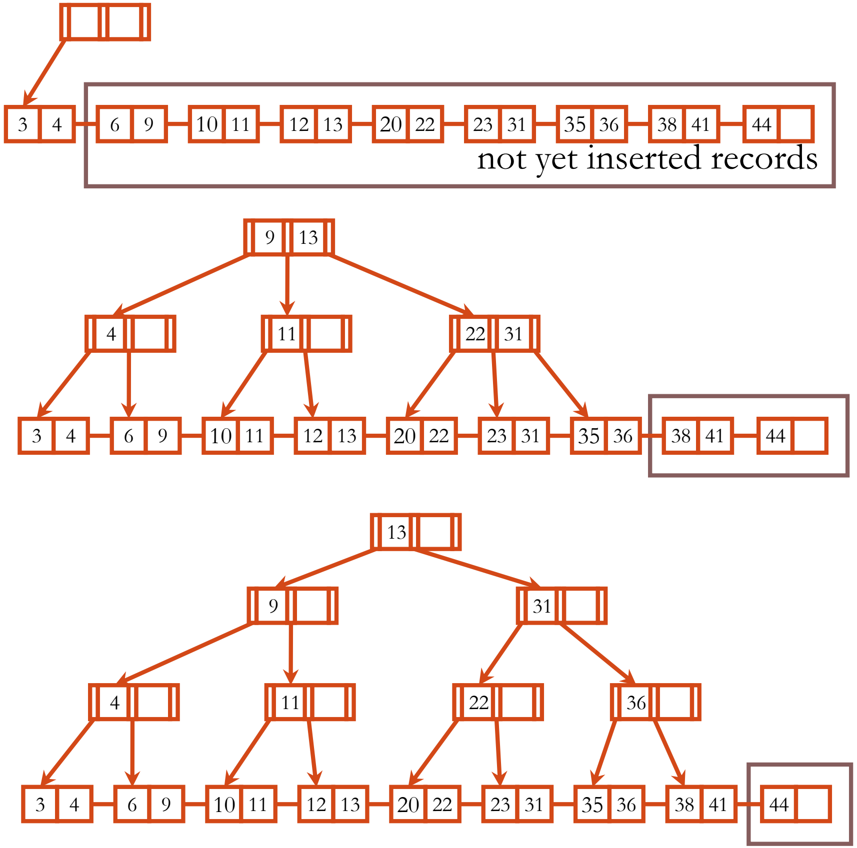
```
CREATE TABLE Product(
	id INT PRIMARY KEY NONCLUSTERED,
	code NVARCHAR(5),
	name NVARCHAR(50),
	type INT);

CREATE NONCLUSTERED INDEX ixProductCode ON Product(code);

CREATE CLUSTERED INDEX ixProductName ON Product(name);
```

Forces ordering

B-TREE : BULK LOADING

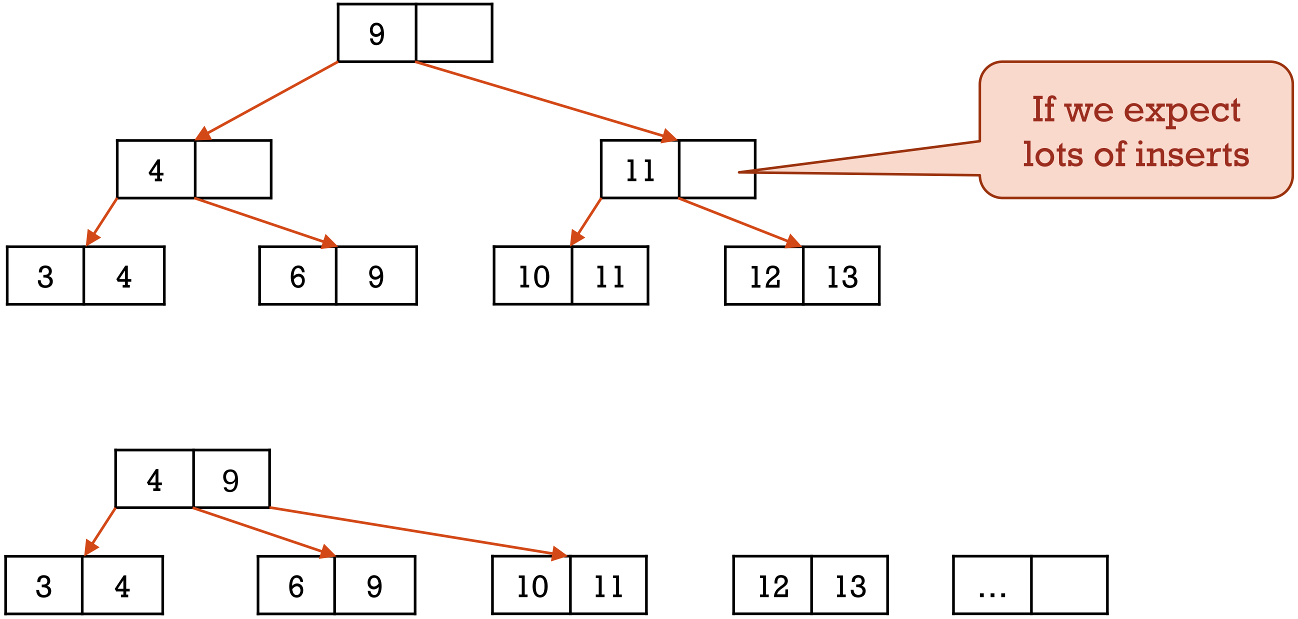
When indexing a large collection, inserting records one by one can be tedious

- Sort the data based on the search key in the pages
- Insert pointer to the leftmost page into a new root
- Move over the data file and insert the respective keys into the rightmost index page above the leaf level. If the rightmost page overflows, split.



B-TREE : BULK LOADING

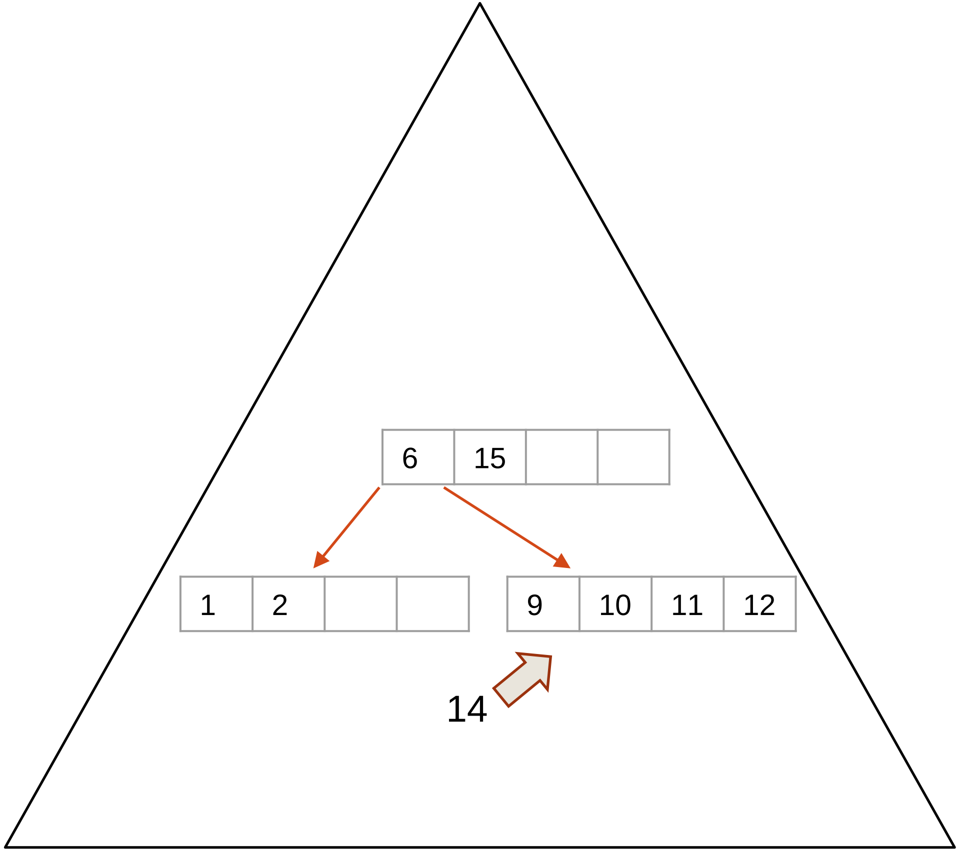
- Sorted: 3, 4, 6, 9, 10, 11, 12, 13…

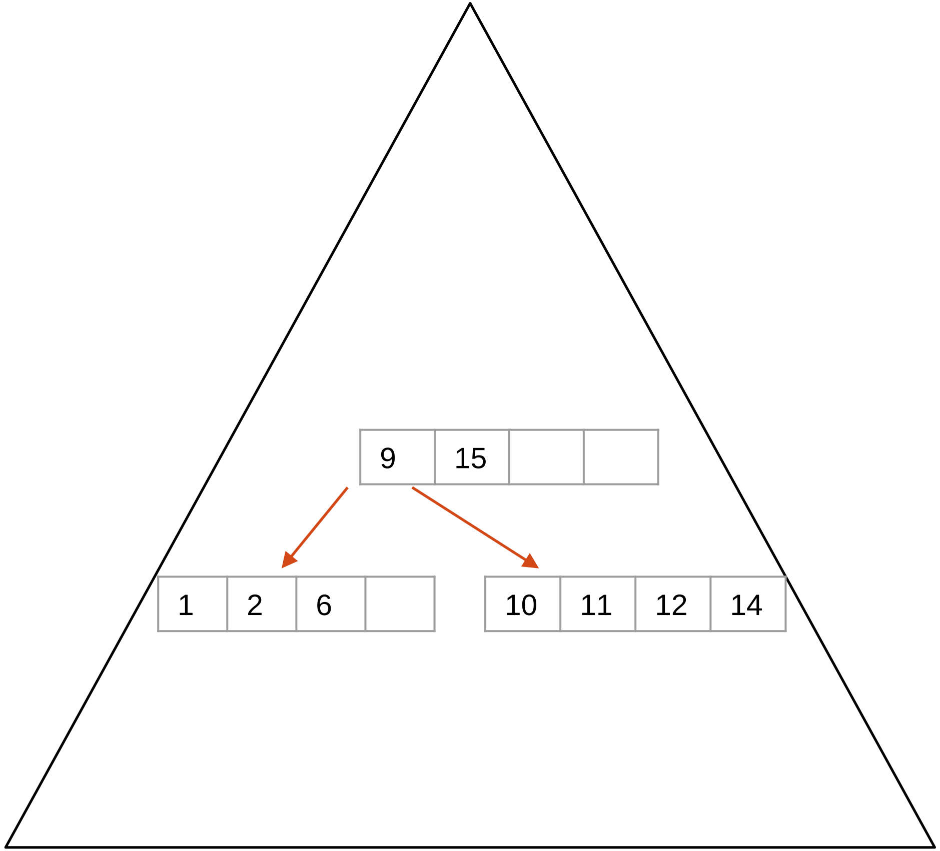


B-TREE : PAGE BALANCING

- Modification of B-tree where an overflow does not have to lead to a page split
- When a page overflows
  - Sibling pages are checked
  - The content of the overflowed page is joined into set X with the left or right neighbors
  - The record to be inserted is added into X and the content is equally distributed into the two nodes
  - The changes are projected into the parent node where the keys have to be modified (but no new key is inserted → no split cascade)
- For high m this change leads to about 75% utilization in the worst case

B-TREE : PAGE BALANCING



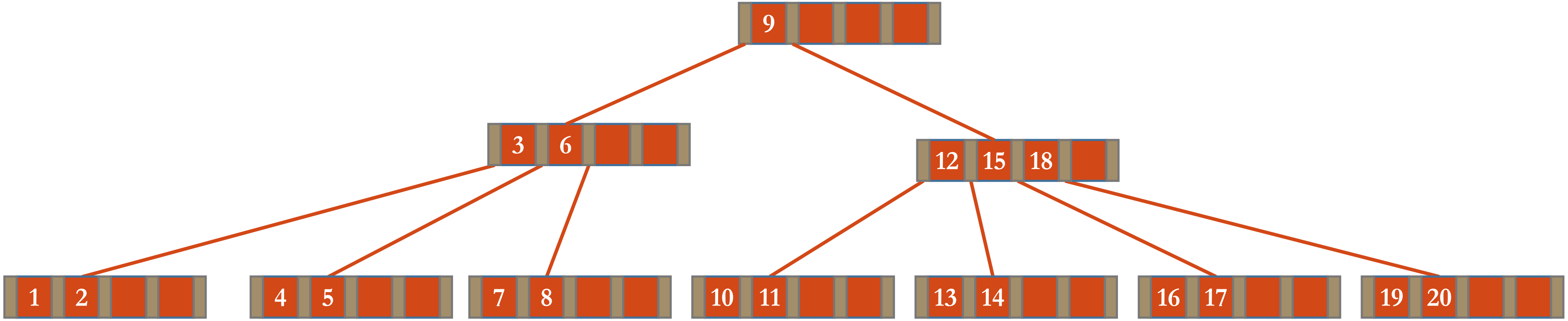


B-TREE : DEFERRED SPLITTING

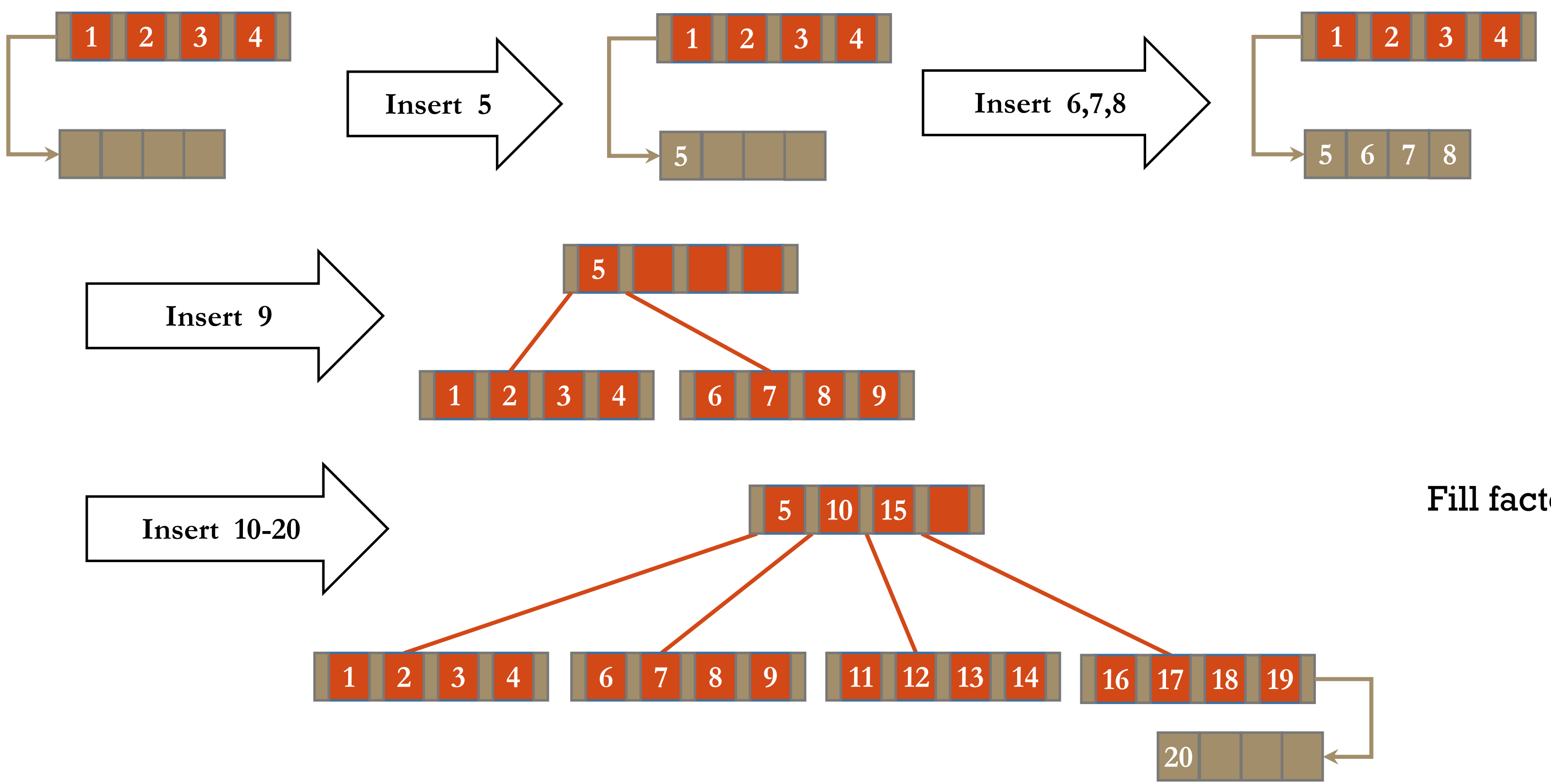
- Certain sequences of inserts can lead to only 50% utilization
- Let us keep an overflow page for each node
- When a page overflows, the overflown record is inserted into the respective overflow page
  - Insert is faster
  - Better utilization
  - Search is slower
    - We need to searh the overflow area
- When both the original and the overflow page are full, the original page is split and the overflow page is emptied

B-TREE : DEFERRED SPLITTING EXAMPLE

Example: Increasing sequence of numbers 1, 2, 3, 4, …, 20 (e.g. typically primary key) → fill factor is 0.5 so the worst possible



B-TREE : DEFERRED SPLITTING EXAMPLE



Fill factor 0.83

VARIABLE LENGTH RECORDS

- Often we want to index not only numbers but also strings
  → variable length-records (VLR) → different m for different nodes
  - Note: In existing DB systems, indexable string data types have upper limit on the number of characters (NVARCHAR(n)) → not exactly VLR
- When splitting a page with VLR, rather length of the records is taken into account than the number of records
  - Result: the distribution is driven by the resulting length
- Can lead to violation of the condition regarding the minimum number of records in a B-tree
- Longer records tend to get closer to the root, causing lower arity close to the root
- When merging, a short record can be replaced by a longer one causing height increase

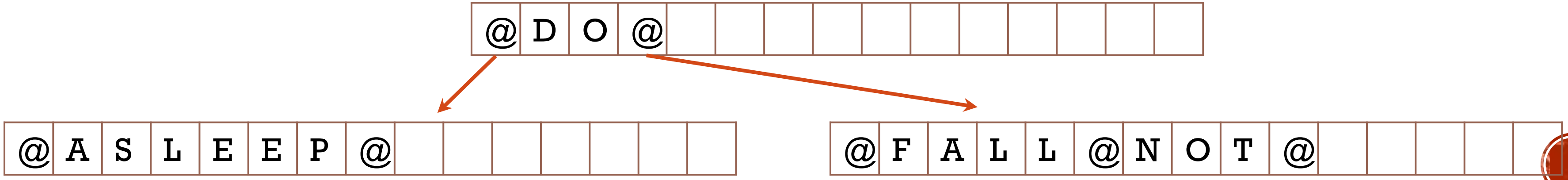
VARIABLE LENGTH RECORDS : EXAMPLE

Representing the sentence: "DO NOT FALL ASLEEP"

- node size is 15
- pointers represented by @
  - For the sake of simplicity let us consider size of a pointer to be identical to the size of a character

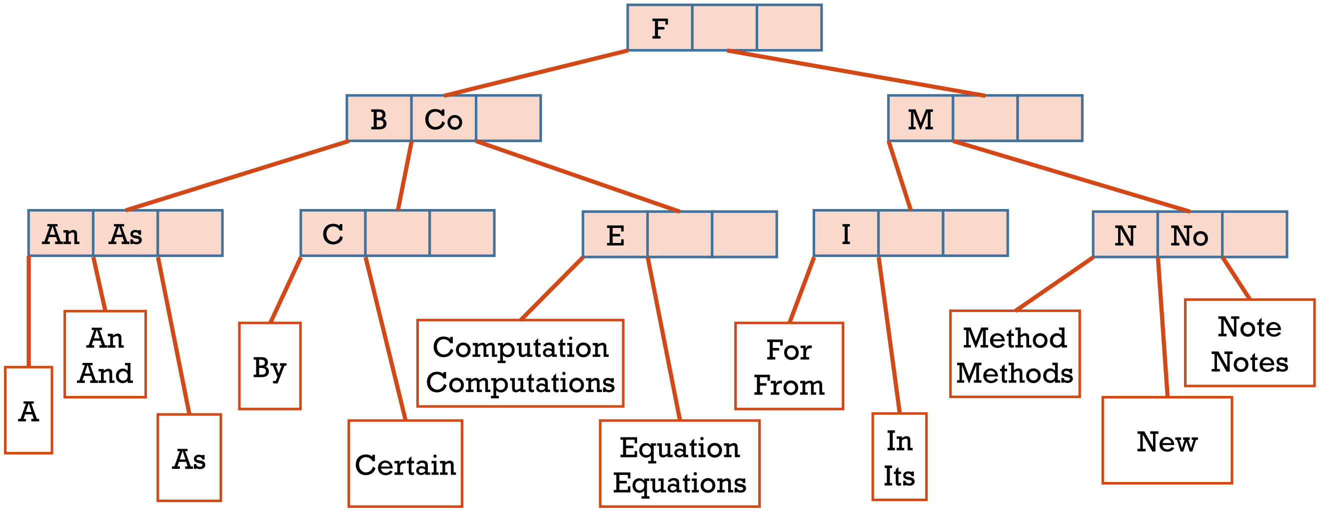
@	D	O	@	F	A	L	L	@	N	O	T	@		

- Inserting "ASLEEP" causes overflow → splitting
- Sequence to be split: @ASLEEP@D**O**@FALL@NOT@
  - → **O** is the middle character



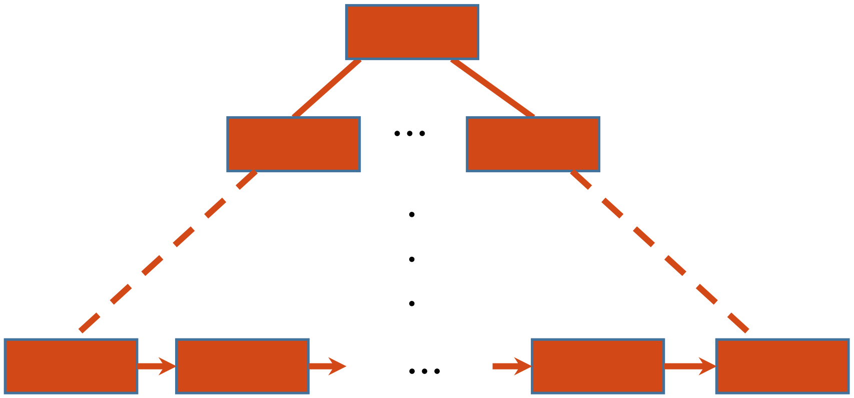
VARIABLE LENGTH RECORDS : PREFIX (B-)TREE

- Modification of redundant B-tree
- Inner node keys do not have to be subsets of the keys in the leaf level, they only need to separate
- Smaller keys lead to higher node capacity → lower trees → faster access
- Suitable choice of separators are prefixes of the keys



B+TREE

- Redundant B-tree
- The leaf level is chained by pointers
  - The leaf nodes do not have to be physically next to each other
- Faster range queries
- Preferred in existing database management system
- Sometimes the inner levels chained as well
  - e.g., Microsoft SQL Server

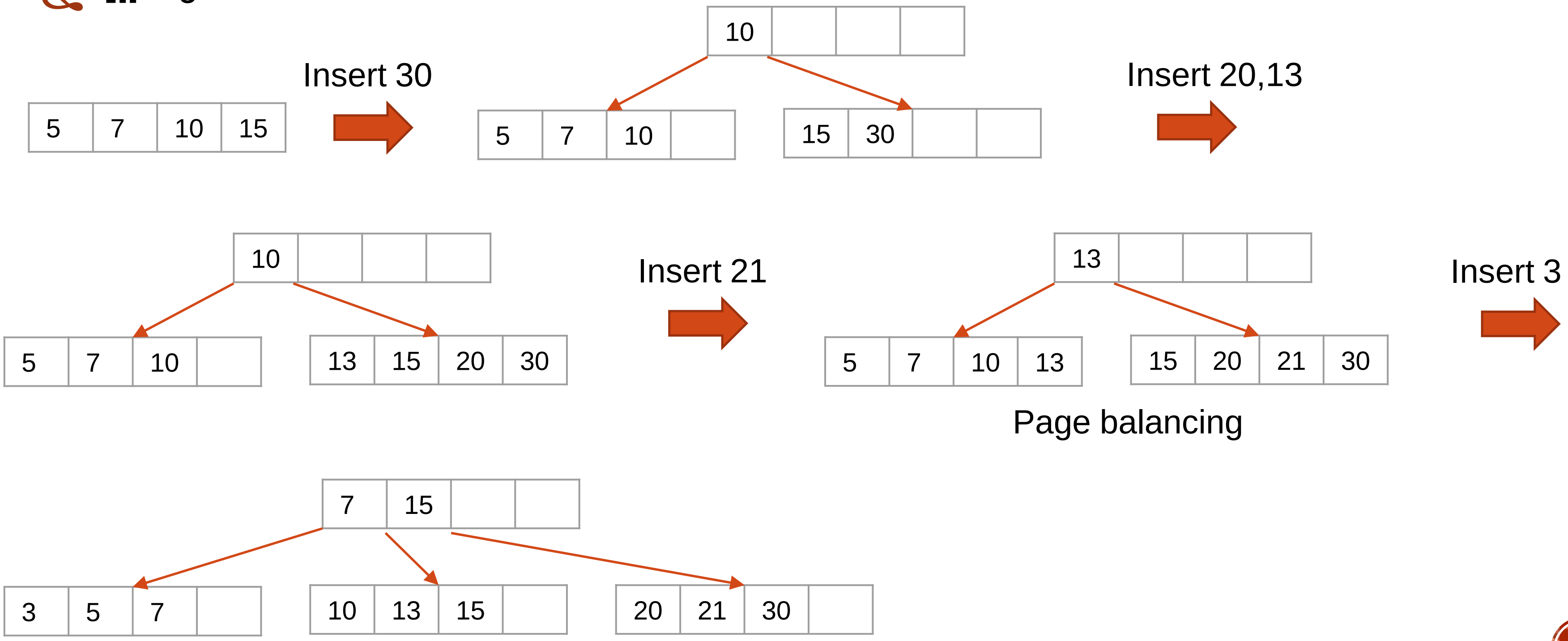


B*TREE

- Generalization of page balancing
  - The root node has at least 2 children
  - Every node different from the root has at least $\lceil(\mathbf{2}\boldsymbol{m}-\mathbf{1})/\mathbf{3}\rceil$ children
    - 2/3 utilization (in B-tree we have 50%)
- Idea: 2 full pages are split into 3 pages (one new page)
- Algorithm:
  - If a node is full but none of its neighbors is full, page balancing takes place
  - If the insert occurs in a full page which has full left or right neighbor
    - Their content is joined into a set X together with the new record
    - A new page P is allocated
    - The records from X are equally distributed into the 3 pages (the 2 existing and P)
    - A new key is added into the parent node and the keys are adjusted
- The delete operation is handled similarly
  - Idea: We use page balancing or we take 3 nodes and merge into 2

EXAMPLE: B*TREE

- m = 5



INDEXES IN EXISTING LEADING DATABASE SYSTEMS

	Oracle 11g	**MSQL Server 2016**	**PostgreSQL 9.2**	**MySQL 5.5**
Standard index	B+tree	B+tree	B+tree	B+tree
Bitmap index	Yes	No	No	No
Hash index	Yes (clustering)	Yes (clustering)	Yes	Yes
Spatial index	R-tree	B+tree Hilbert curve	R-tree	R-tree