12 B+Trees

Content

- Maintaining a sequence set
- A simple prefix B+ Tree
- Simple Prefix B+ Tree Maintenance: Insertions and Deletions

Motivation

- Some applications require two views of a file:

Indexed view :	**Sequential view :**
Records are indexed by a key	Records can be sequentially accessed in order by key
Direct, indexed access	Sequential access (physically contiguous records)
Interactive, random access	Batch processing (Ex: co-sequential processing)

Example of Applications

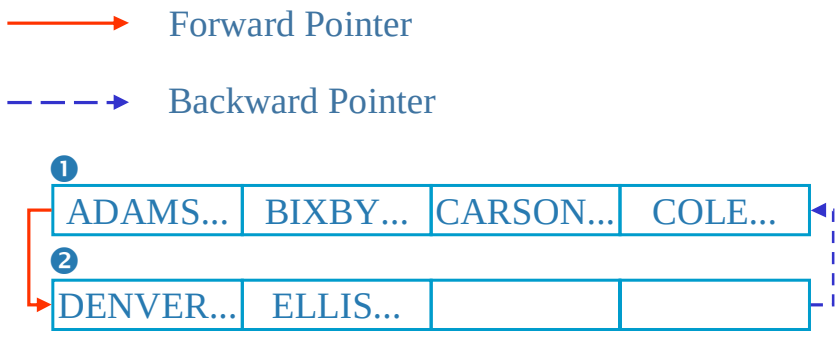
- Student record system in a university
  - Indexed view: access to individual records
  - Sequential view: batch processing when posting grades or when fees are paid
- Credit card system
  - Indexed view: interactive check of accounts
  - Sequential view: batch processing of payment slips
- We will look at the following two aspect of the problem:
  1. Maintaining a sequence set: keeping records in sequential order
  2. Adding an index set to the sequence set

Maintaining a Sequence Set

- Sorting and re-organizing after insertions and deletions is out of question
- We organize the sequence set in the following way
  - Records are grouped in **blocks**
  - Blocks should be **at least half full**
  - **Link fields** are used to point to the preceding block and the following block (similar to doubly linked list)
  - Changes (inserted/deletion) are localized into blocks by performing:
  1. **Block Splitting** when **insertion** causes overflow
  2. **Block Merging** or **Redistribution** when **deletion** causes underflow

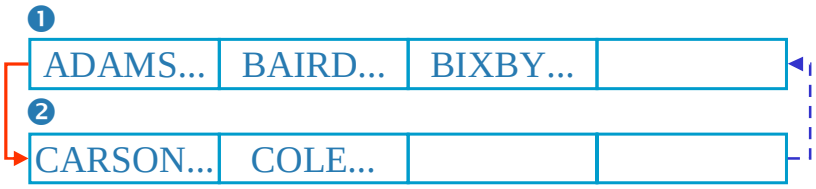
Example

- Block Size = 4
- Key = Last Name

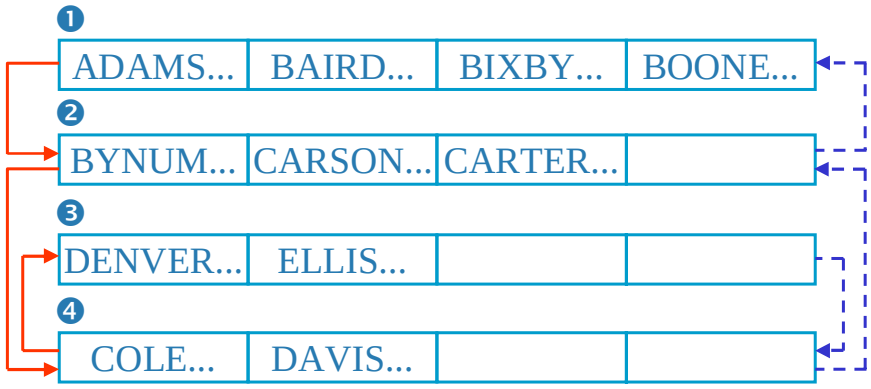


Insertion with Overflow

- Insert “BAIRD...”

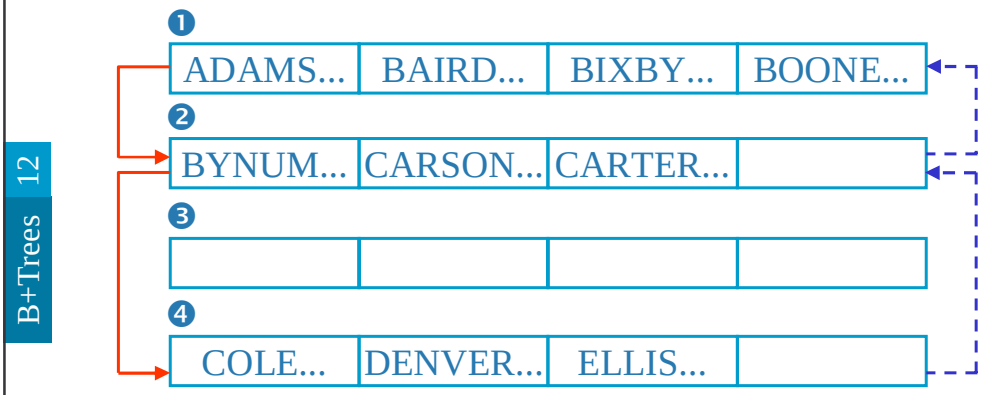


Deletion with Merging



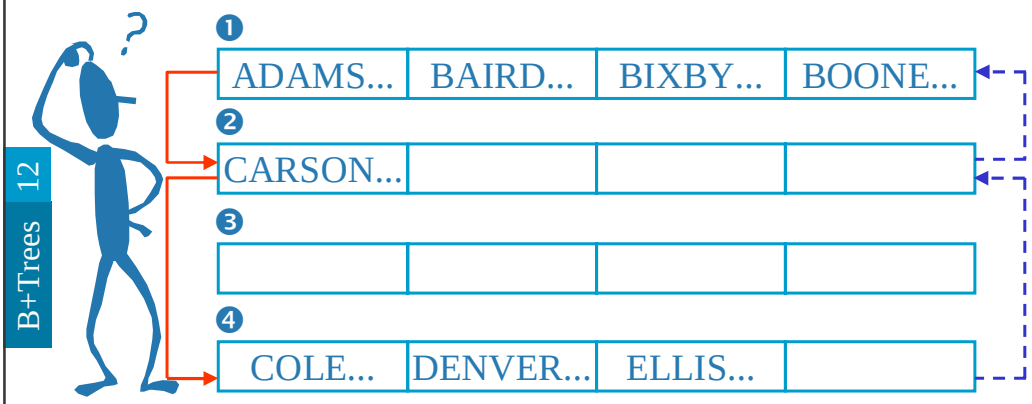
- Delete “DAVIS...”

Deletion with Merging (Con't)



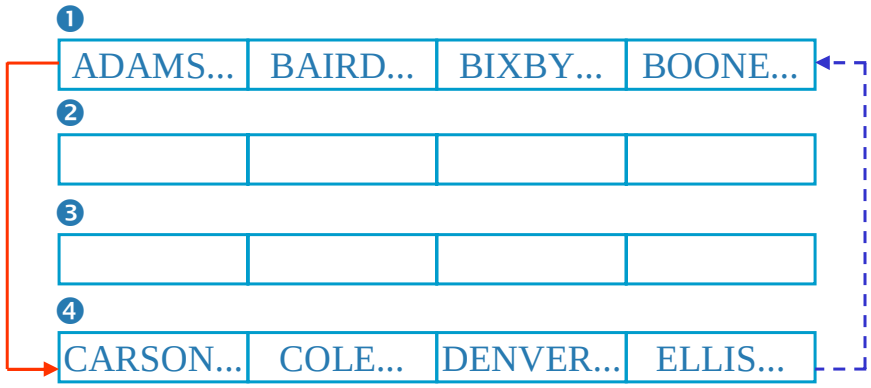
- Block ❸ is available for re-use

Delete "BYNUM", then Delete "CARTER"

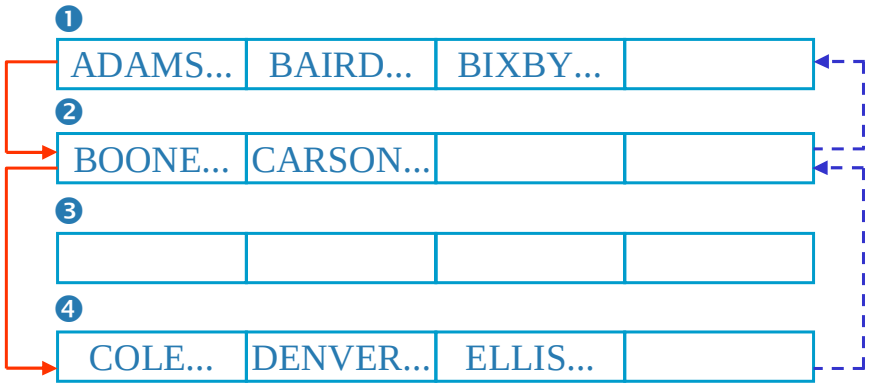


Solution # 1

- We can merge Block ❷ and ❹



Solution # 2: Deletion with Redistribution



Advantages and Disadvantages of Sequence Set

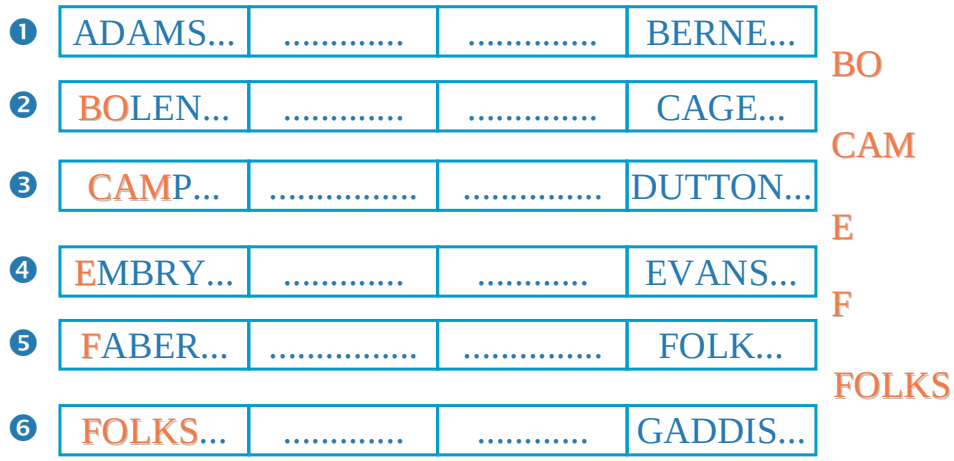
- Advantages
  - No need to re-organize the whole file after insertions/deletions
- Disadvantages
  - File takes more space than unblocked files (since blocks may be half full)
  - The order of the records is not necessarily **physically** sequential (we only guarantee physical sequentiality within a block)

Choosing Block Size

- Main memory constraints (must hold at least 2 blocks)
- Avoid seeking within a block (e.g., in sector formatted disks choose block size equal to cluster size).

Adding an Index Set to the Sequential Set

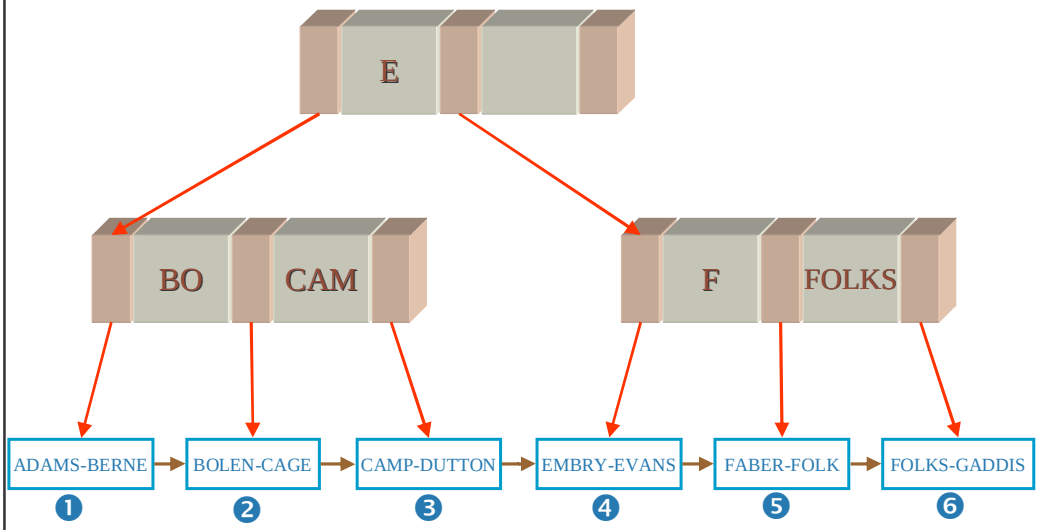
- Index will contain SEPERATORS instead of KEYS



The Simple Prefix B+ Tree

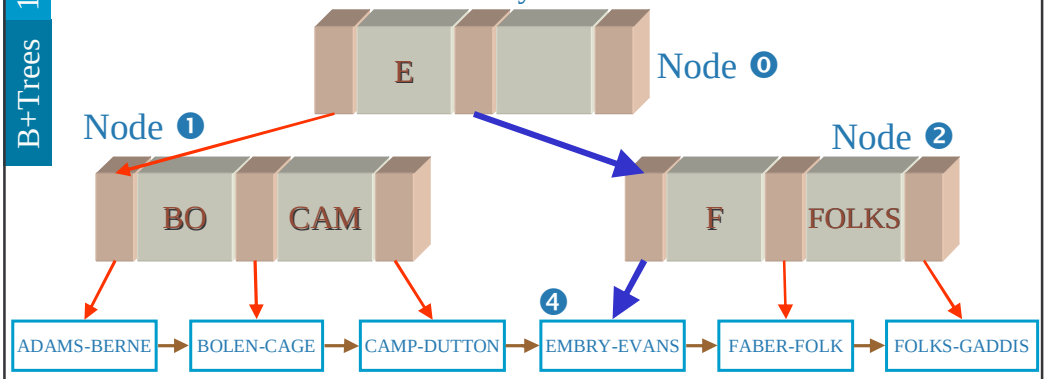
- The **simple prefix B+ tree** consists of
  - **Sequence Set**
  - **Index Set**: similar to a B-tree index, but storing the shortest separators for the sequence set.

Example: Order of the index set is 3



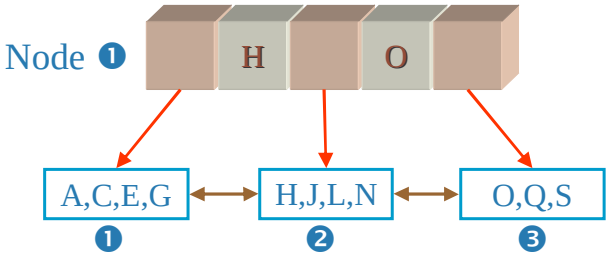
Search in a Simple Prefix B+ Tree

- Search for “EMBRY”
  - Retrieve Node ⓪ (Root)
  - Since “EMBRY”> “E”, so go right, and retrieve Node ❷.
  - Since “EMBRY”< “F”, so go left, and Block # ❹
  - Look for the record with key “EMBRY” in Block # ❹



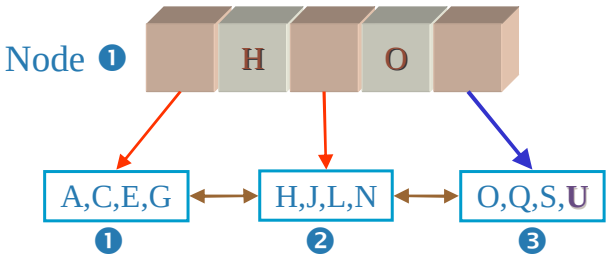
Simple Prefix B+ Tree Maintenance

- Example:
  - Sequence set has blocking factor 4
  - Index set is a B tree of order 3



Example (Cont'd)

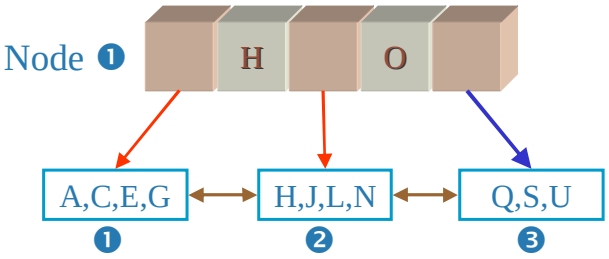
1. Changes which are local to single blocks in the sequence set
   - Insert "U"
     - Go to the root
     - Go to the right of "O"
     - Insert "U" to Block ❸



–There is no change in the index set

Example (Cont'd)

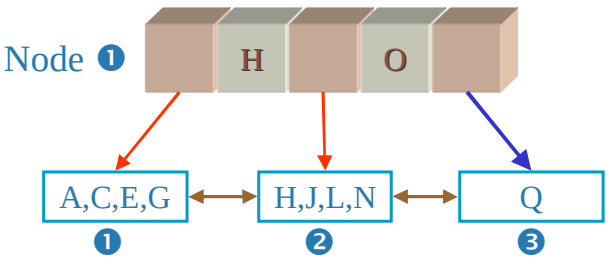
- Delete "O"
  - Go to the root
  - Go to the right of "O"
  - Delete "O" from Block ❸



–There is no change in the index set: "O" is still a perfect separator for Blocks ❷ and ❸

Example (Cont'd)

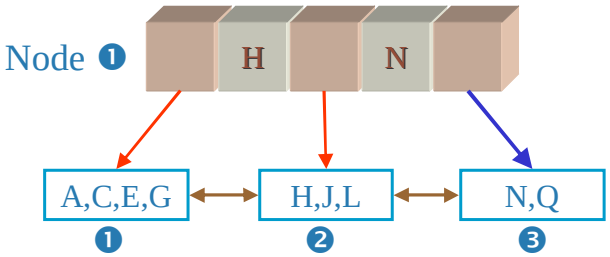
2. Changes involving multiple blocks in the sequence set
   - Delete "S" and "U"



–Now Block ❸ becomes less than ½ full (UNDERFLOW)

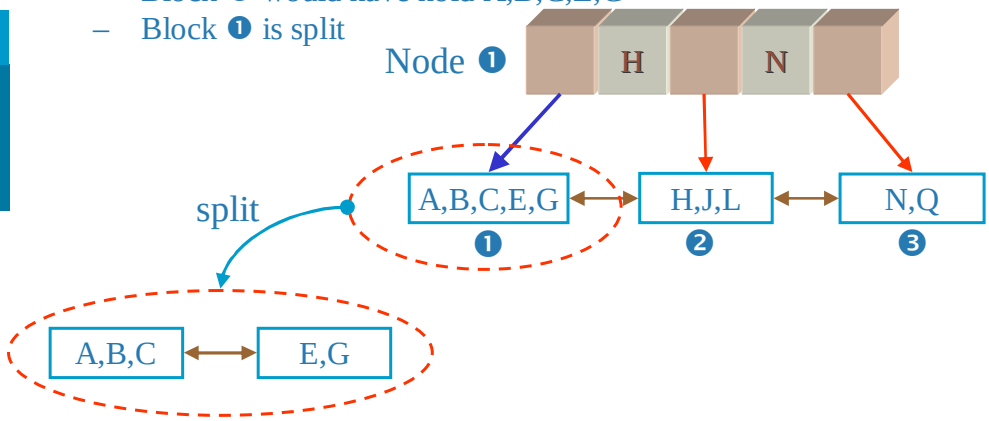
Example (Cont'd)

- Since Block ❷ is full, the only position is re-distribution bringing a key from Block ❷ to Block ❸
- We must update the separator "O" to "N"



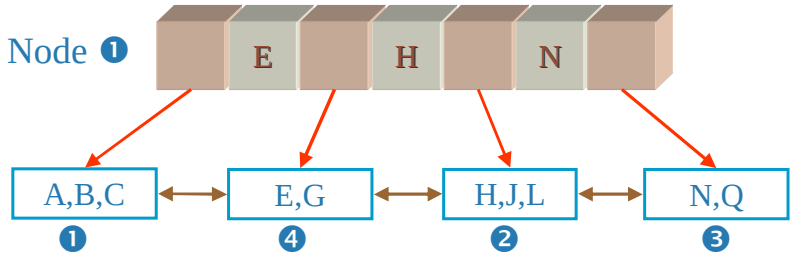
Example (Cont'd)

- Insert "B"
  - Go to the root
  - Go to the left of "H" to Block ❶
  - Block ❶ would have hold A,B,C,E,G
  - Block ❶ is split

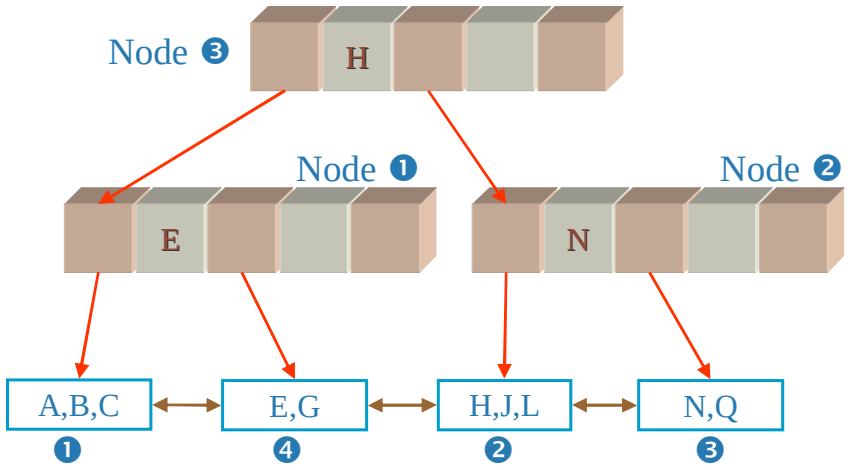


Example (Cont'd)

- So this causes Node ❶ to split

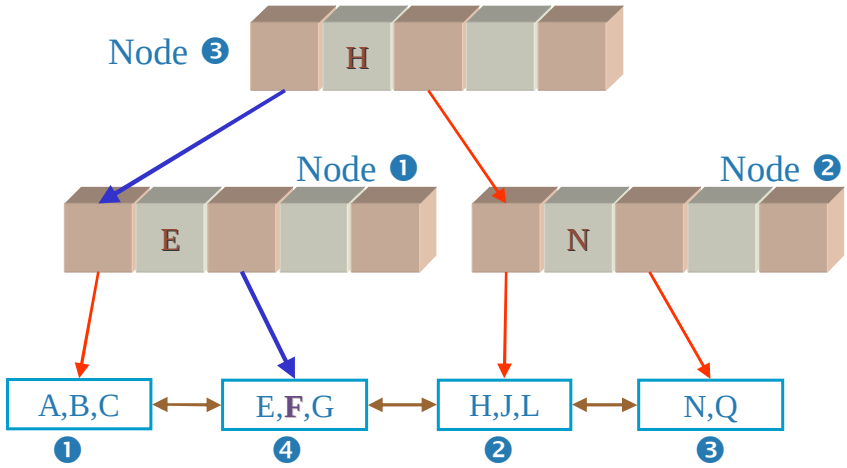


Example (Cont'd)



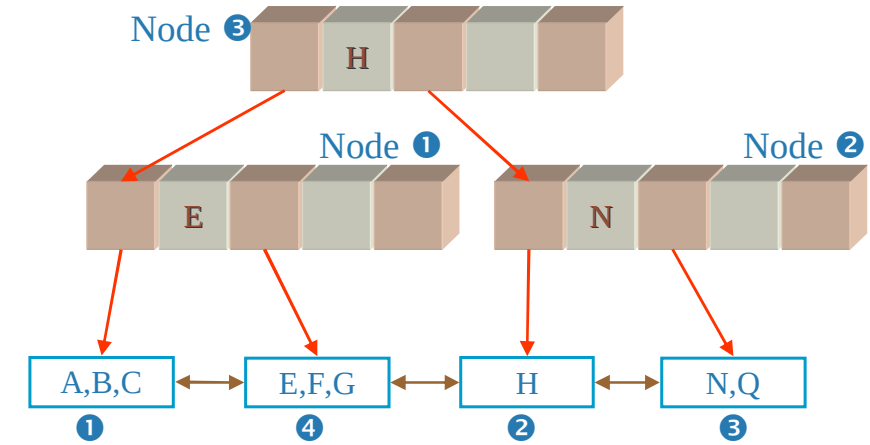
Example (Cont'd)

- Insert "F"



Example (Cont'd)

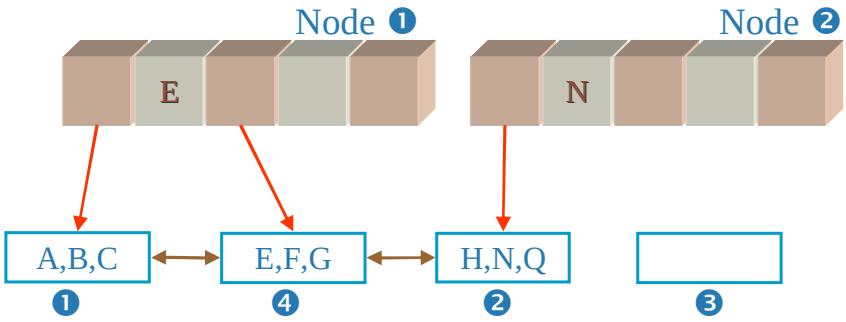
- Delete "J" and "L"



- This is an UNDERFLOW. You may think to redistribute Blocks ❷ and ❹: E,F,G,H becomes E,F and G,H.

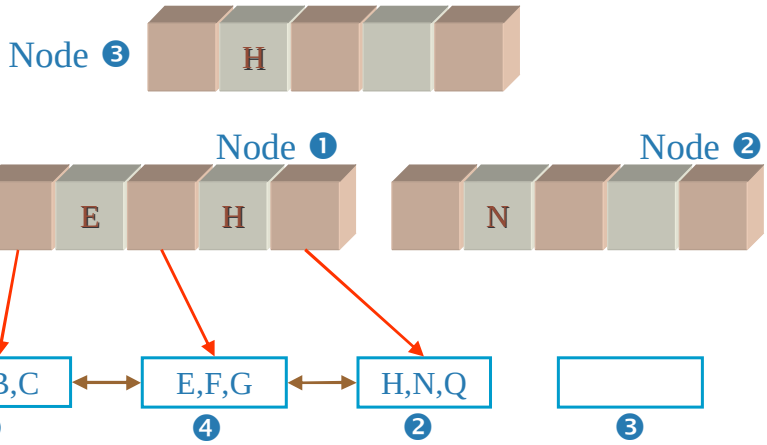
Example (Cont'd)

- Why this is not possible?
- Blocks ❷ and ❹ are not siblings! They are cousins.
- Merge Blocks ❷ and ❸
- Send Block ❸ to AVAIL LIST
- Remove the Link Between Node ❷ and Block ❸



Example (Cont'd)

- Send Node ❷ and ❸ to AVAIL LIST



Example (Cont'd)

- Blocks were reunited as a big happy family again ☺