

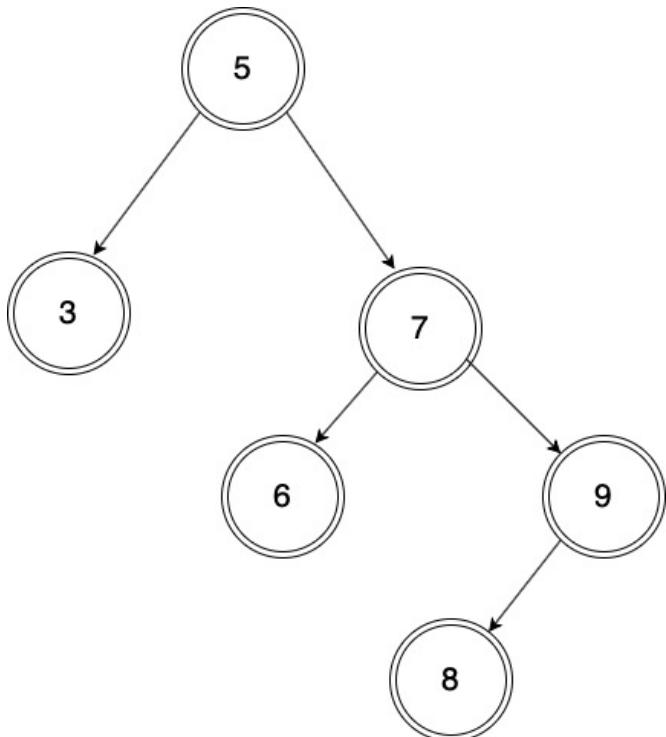
В+ Древья

# Поисковые деревья

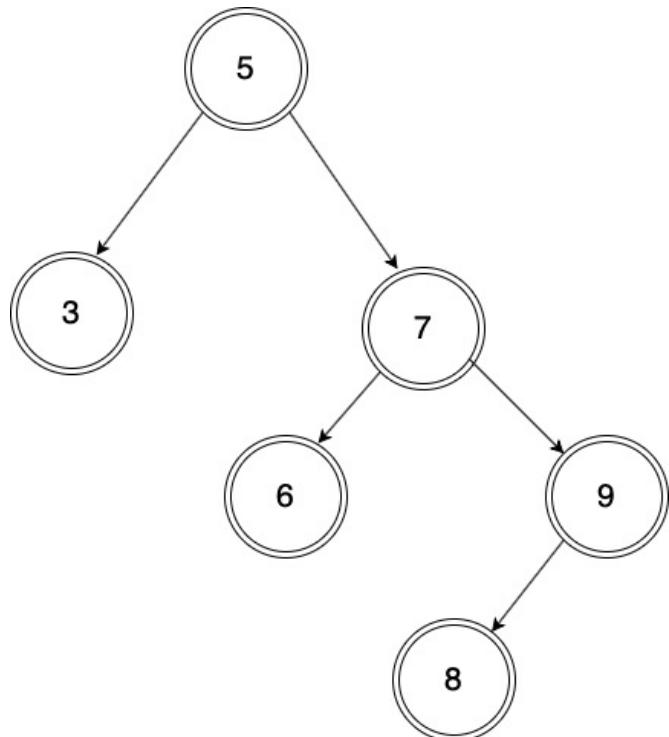
Бинарное поисковое дерево – это сортированная структура данных в памяти, используемая для эффективного поиска вида «ключ-значение».

Бинарное дерево поиска состоит из нескольких вершин. Каждая вершина дерева представляется ключом и двумя указателями на дочерние узлы, и указатель на родительский узел.

# Поисковые деревья

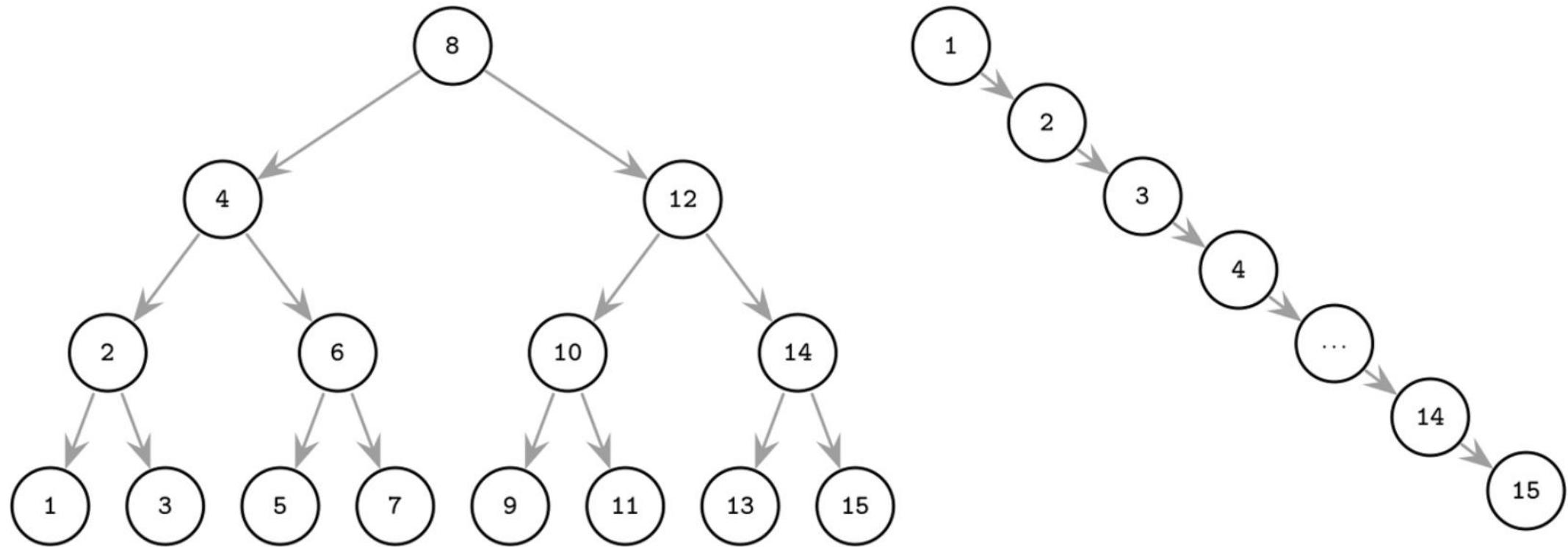


# Поисковые деревья



Ключ в любой вершине всегда больше любого ключа в левом поддереве и меньше любого ключа в правом поддереве

# Балансировка деревьев



# Сбалансированные деревья

Дерево называется сбалансированным (АВЛ-дерево), если разница в высоте двух поддеревьев любой вершины не более одного.

Высота дерева  $\log_2 n$ , поэтому время поиска в худшем сокращается до  $O(\log_2 n)$ , а время поиска по обычному дереву может быть  $O(n)$ .

Алгоритм вставки усложняется, так как требуется балансировка каждый раз.

# Ветвистость дерева

– это свойство каждого узла дерева ссылаться на большое число узлов-потомков.

Для деревьев с низким ветвлением высота деревьев является логарифмом с базой небольшого значения, что приводит к большему числу операций работы с диском.

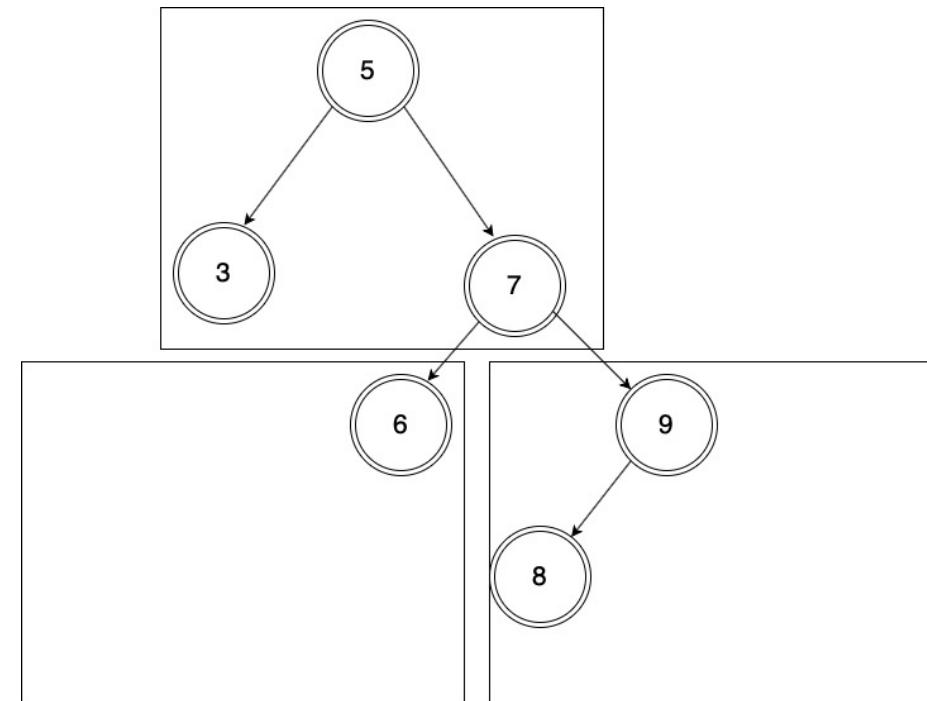
Версия деревьев для хранения на диске должны обладать высокой степенью ветвлении и небольшой высотой.

# Структуры для работы с диском

Минимальным элементом чтения с диска является блок (страница), поэтому структуры данных должны быть оптимизированы для поблочного чтения.

Пример: Страницное дерево поиска

С точки зрения физической организации дерево представляется как мультисписочная структура страниц внешней памяти, т.е. каждому узлу дерева соответствует блок внешней памяти (страница).



# Семейство структур данных В-деревьев

Концептуально В-деревья основаны на бинарных сбалансированных деревьях поиска с идеей на **увеличение степени ветвления (бинарность совсем не обязательна!)** и уменьшения значения высоты

- B Tree (1971)
- B+ Tree (1973)
- B\* Tree (1977)
- B-link Tree (1981)

# В-дерево

- В-дерево – структура данных в виде **сбалансированного** дерева поиска. Сбалансированность означает, что длина пути от корня дерева к любому его листу одна и та же.
- Обобщения бинарного дерева поиска с условием, что вершина содержит более, чем 2 ребенка
- Недостаток В-деревьев в том, что их трудно балансировать

# B+ дерево

- B+ дерево – модификация B-дерева, для которого данные остаются сортированными и время поиска, забор, вставки или удаления -  $O(\log n)$
- Преимущество B+дерева в том, что его достаточно просто балансировать
- Достигается это следующими свойствами:

# Свойства B+ дерева

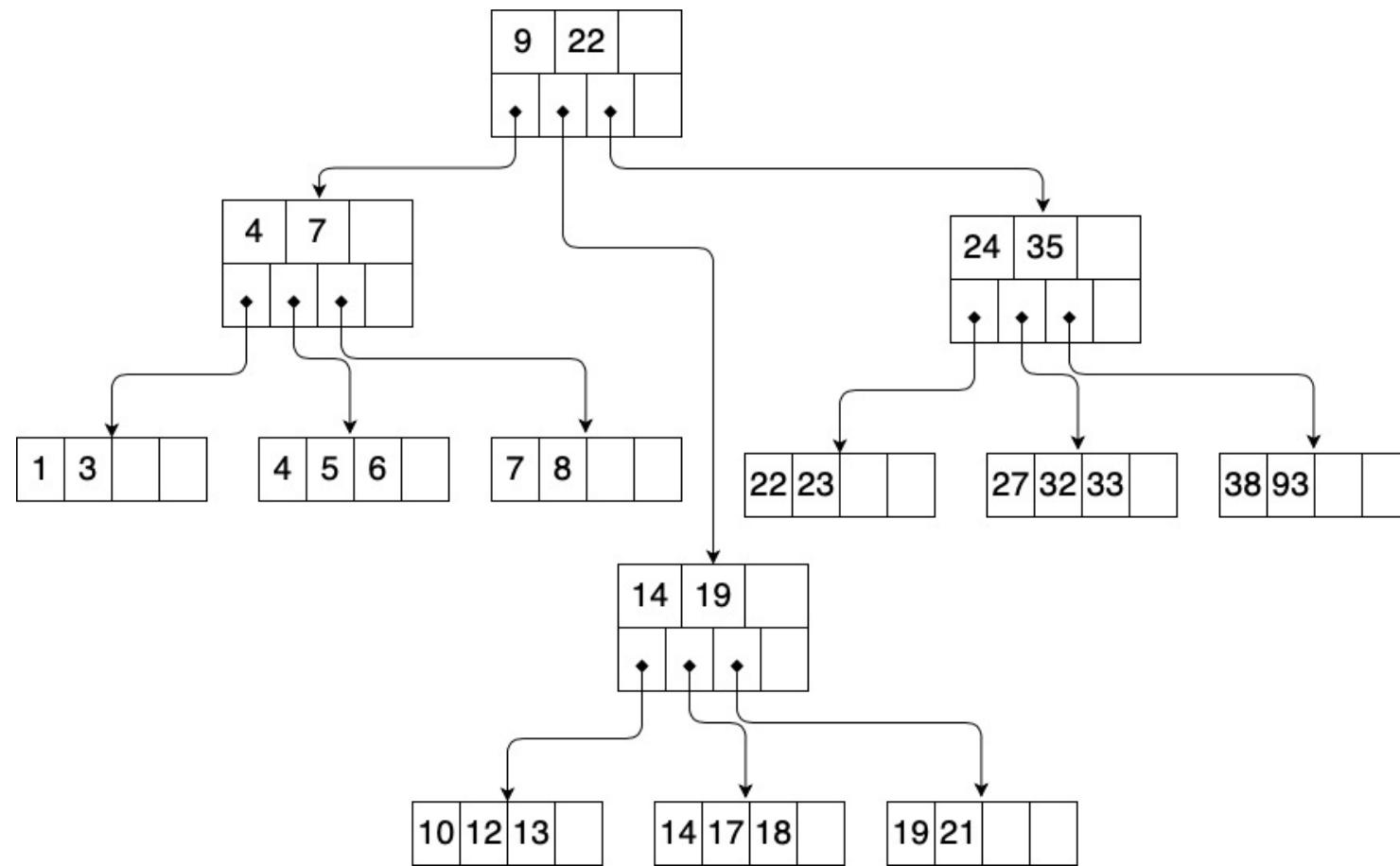
B+ дерево это дерево поиска с M элементами разветвления со следующими свойствами:

- идеально сбалансированное дерево (все листья на одной высоте)
- Каждая некорневая вершина заполнена хотя бы наполовину  $\frac{M}{2} - 1 \leq$  количество значений в вершине  $\leq M - 1$
- Каждая внутренняя вершина с k ключами имеет k+1 непустого ребенка
- Корень может содержать менее  $\frac{M}{2} - 1$  указателей

# Концептуальное описание структуры



# Альтернативное изображение B+ деревьев



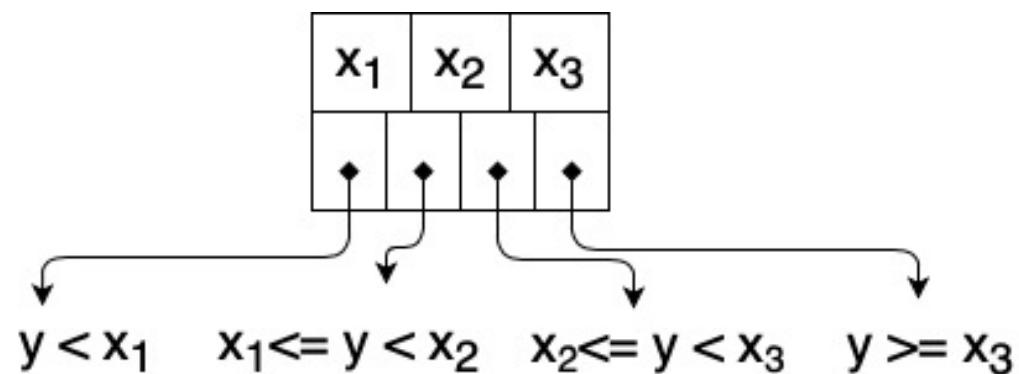
# B+ деревья

Типовая структура листовой страницы B+-дерева выглядит как упорядоченная последовательность ключей и списков идентификаторов строк.

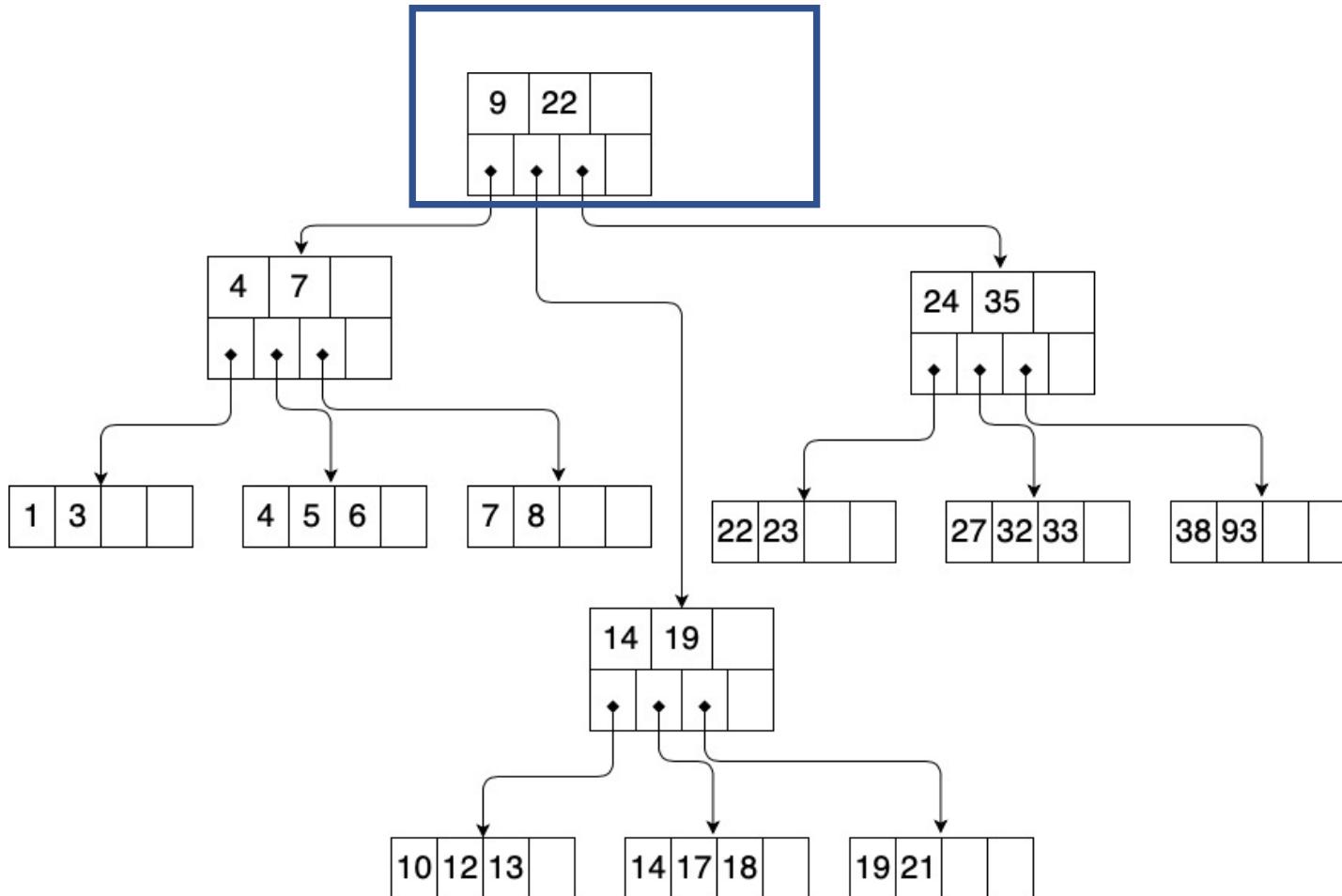
Будем для простоты предполагать, что одна вершина = одна страница. С точки зрения оптимизации могут быть более сложные решения.

# B+ деревья

Поиск по каждой не листовой вершине итерационно выполняет ряд сравнений



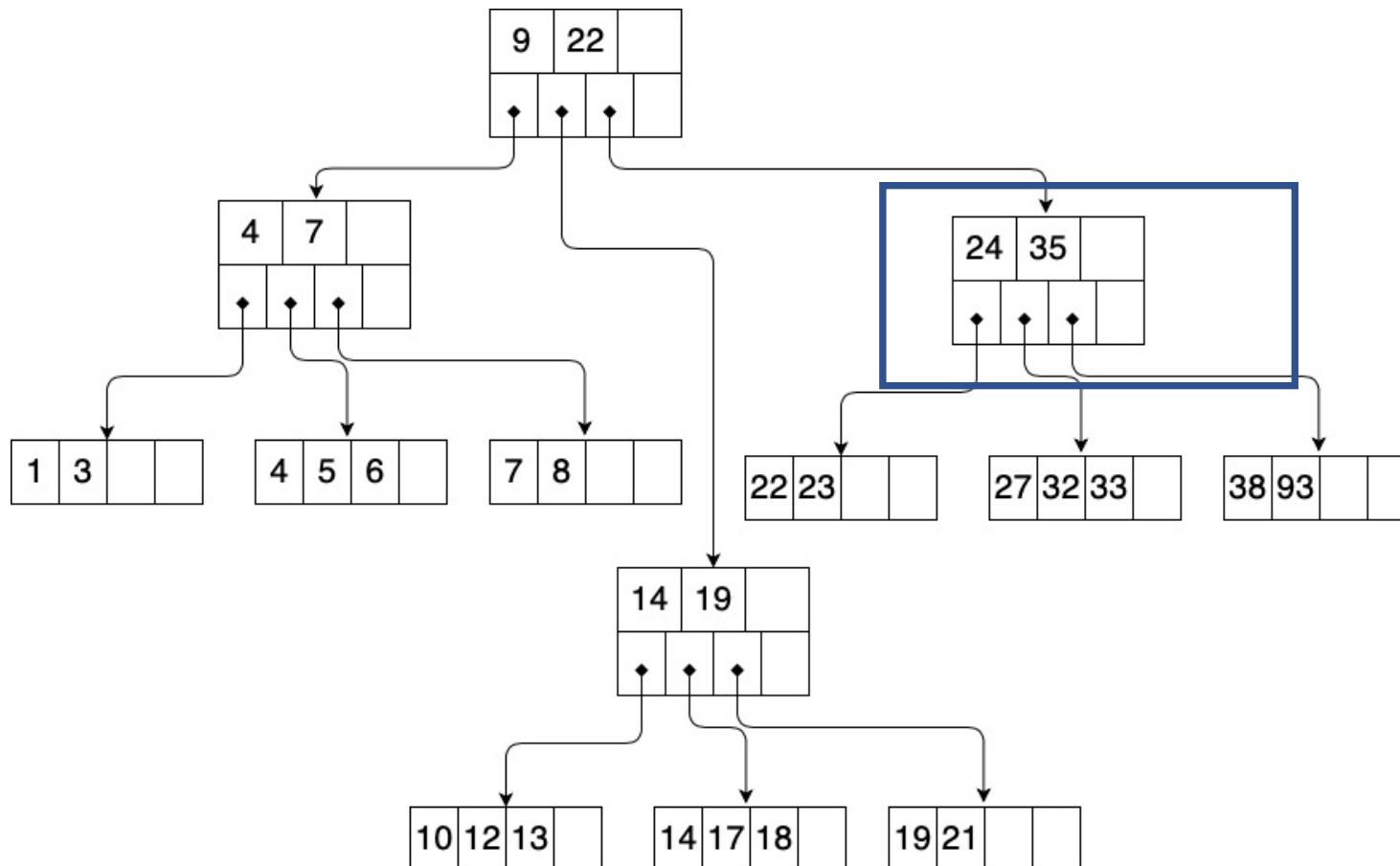
# Поиск значения



Поиск элемента 33

1. Считываем корень
2. Сравниваем 9 и 33
3. Сравниваем 21 и 33
4. Переходим по 3 ссылке

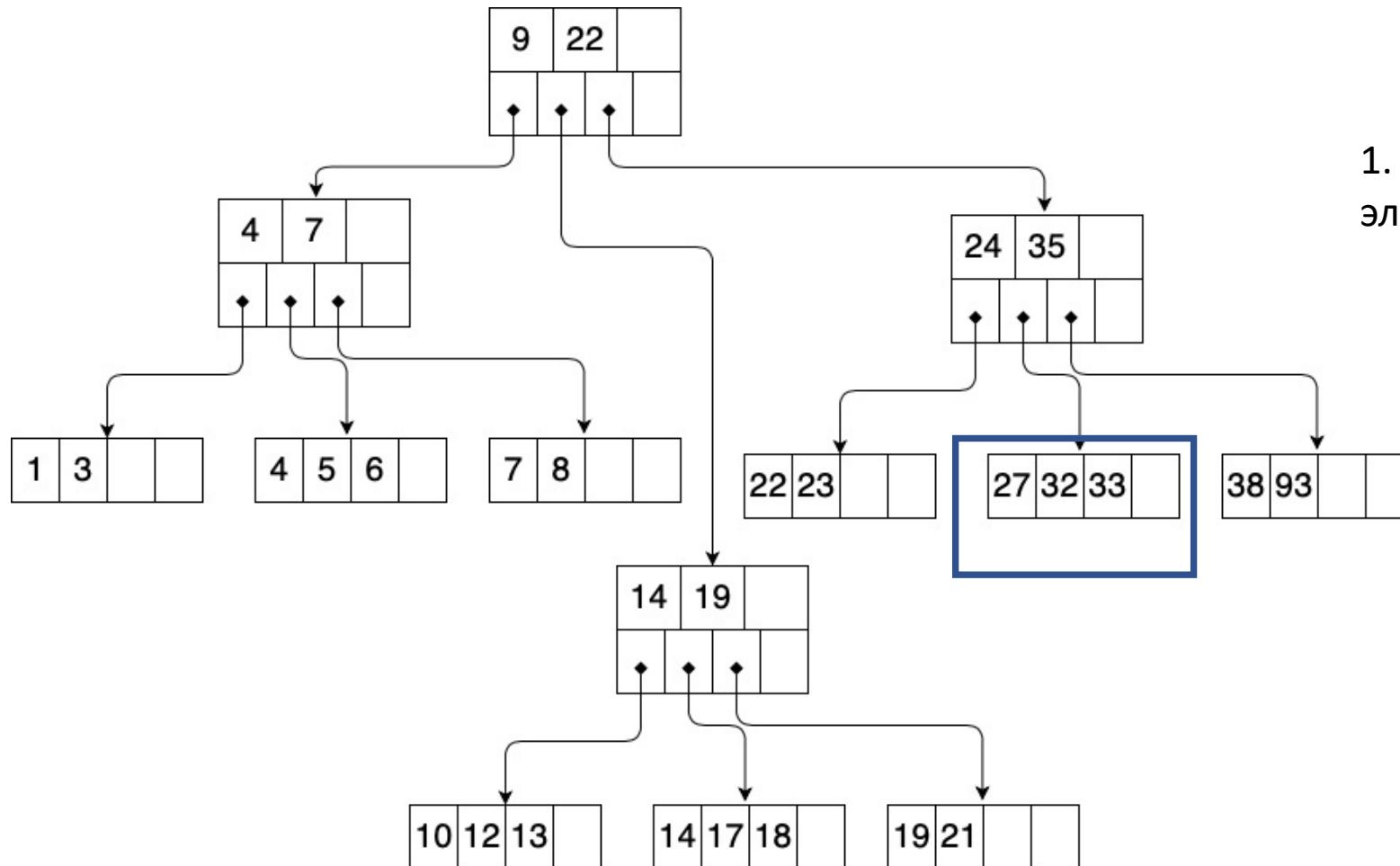
# Поиск значения



Поиск элемента 33

1. Считываем новую вершину
2. Сравниваем 24 и 33
3. Сравниваем 35 и 33
4. Переходим по 2 ссылке

# Поиск значения



Поиск элемента 33

1. Осуществляем поиск элемента 33

# Оценка сложности

Пусть есть  $N$  элементов на вершину и всего  $M$  элементов в  $B+$ -дереве. Тогда количество чтений с диска -  $\log_N M$ , а количество сравнений -  $\log_2 M$

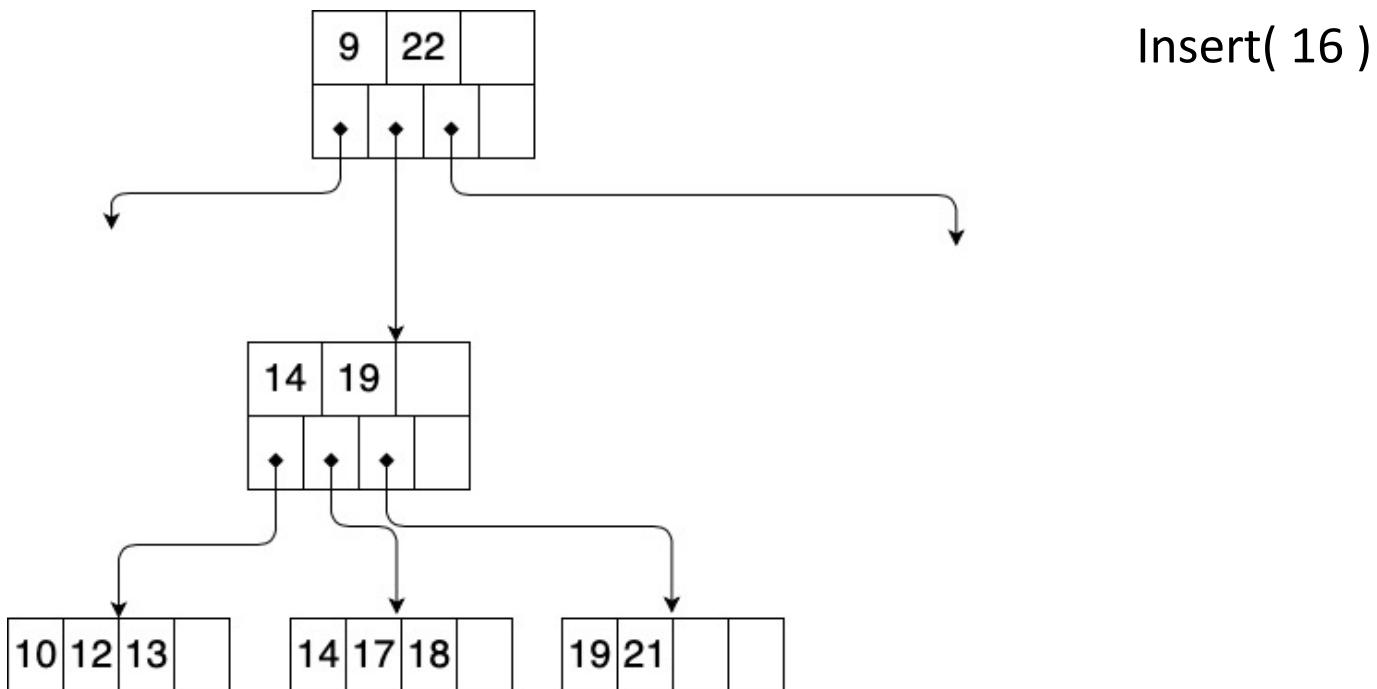
Для асимптотической сложности базу иногда опускают и пишут  $O(\log M)$

# B+ дерево. Вставка

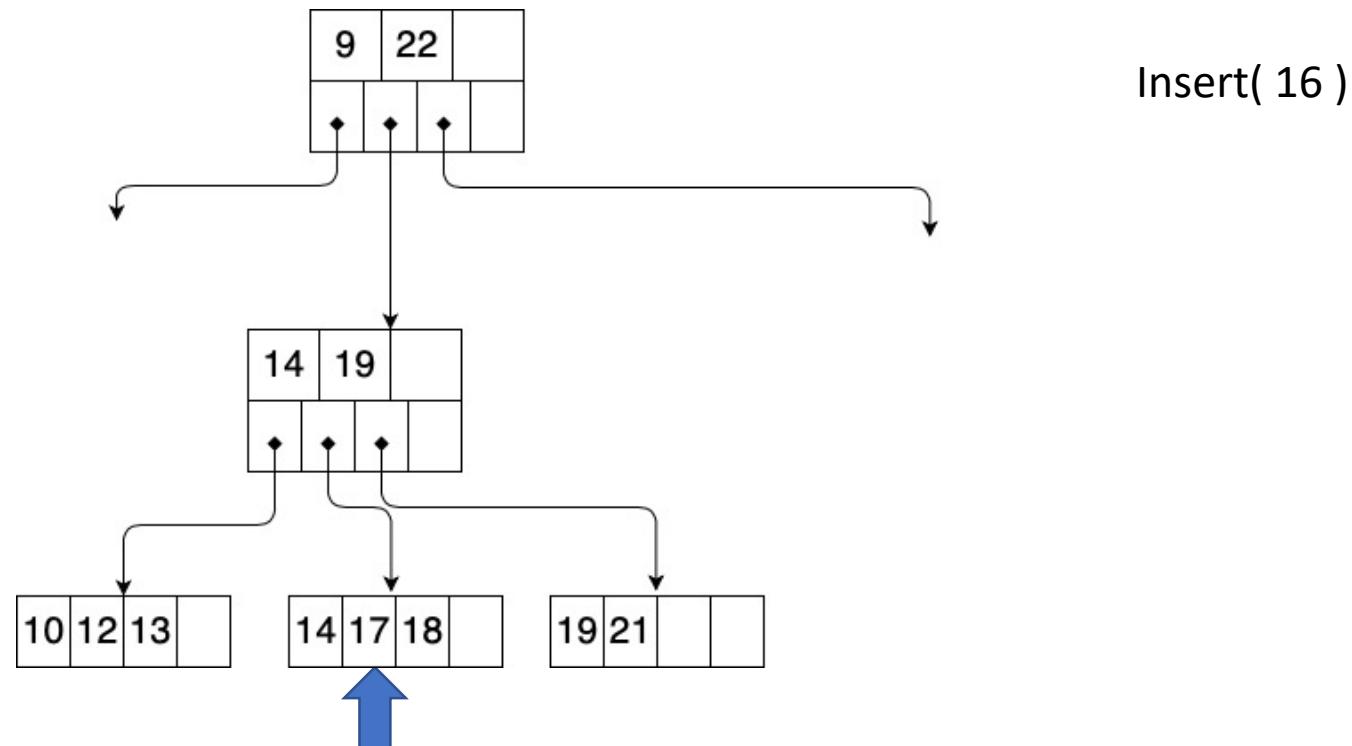
Для вставки в B+ дерево, сначала осуществляется поиск местоположения точки вставки (поиск по ключу).

- Если в B+-дереве не содержится ключ с заданным значением, то будет получен номер страницы, где нужно такой ключ содержать, и соответствующие координаты внутри страницы.
- Если такой элемент уже существует, то для уникального индекса выдается ошибка.
- Если лист заполнен не полностью, то происходит вставка в лист и алгоритм окончен.
- Если лист заполнен, то выполняется алгоритм разделения листа.

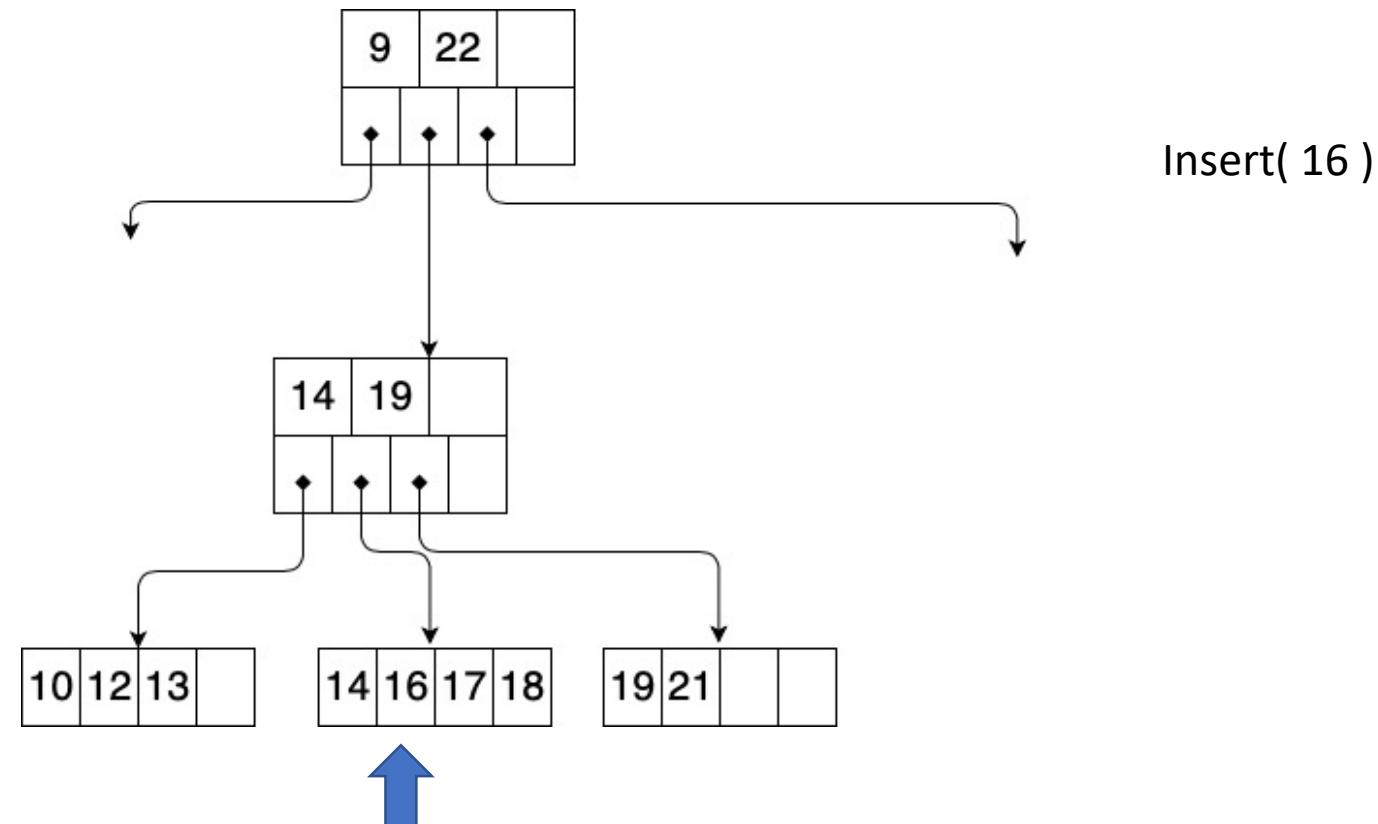
# B+ Дерево. Вставка



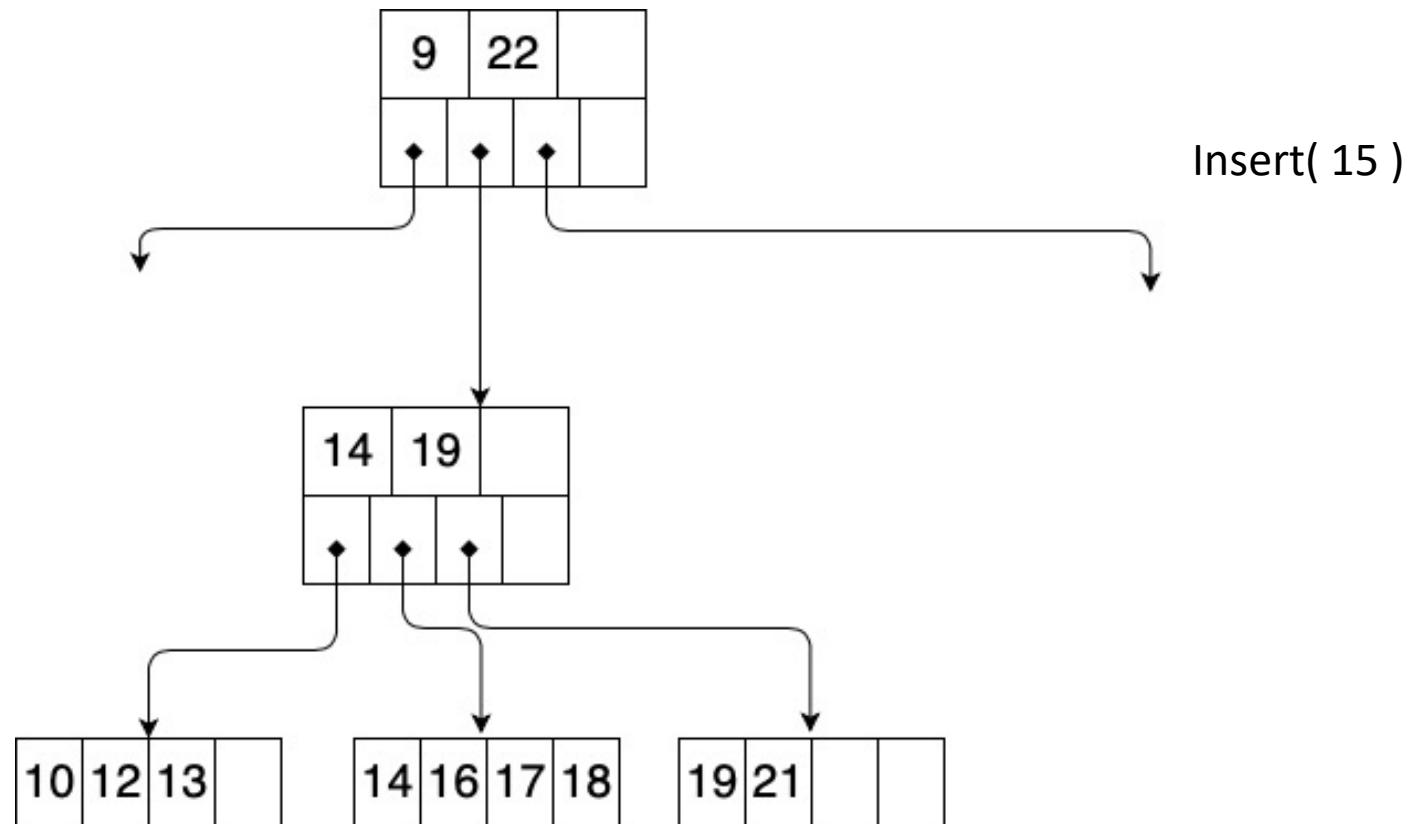
# В+ Дерево. Вставка



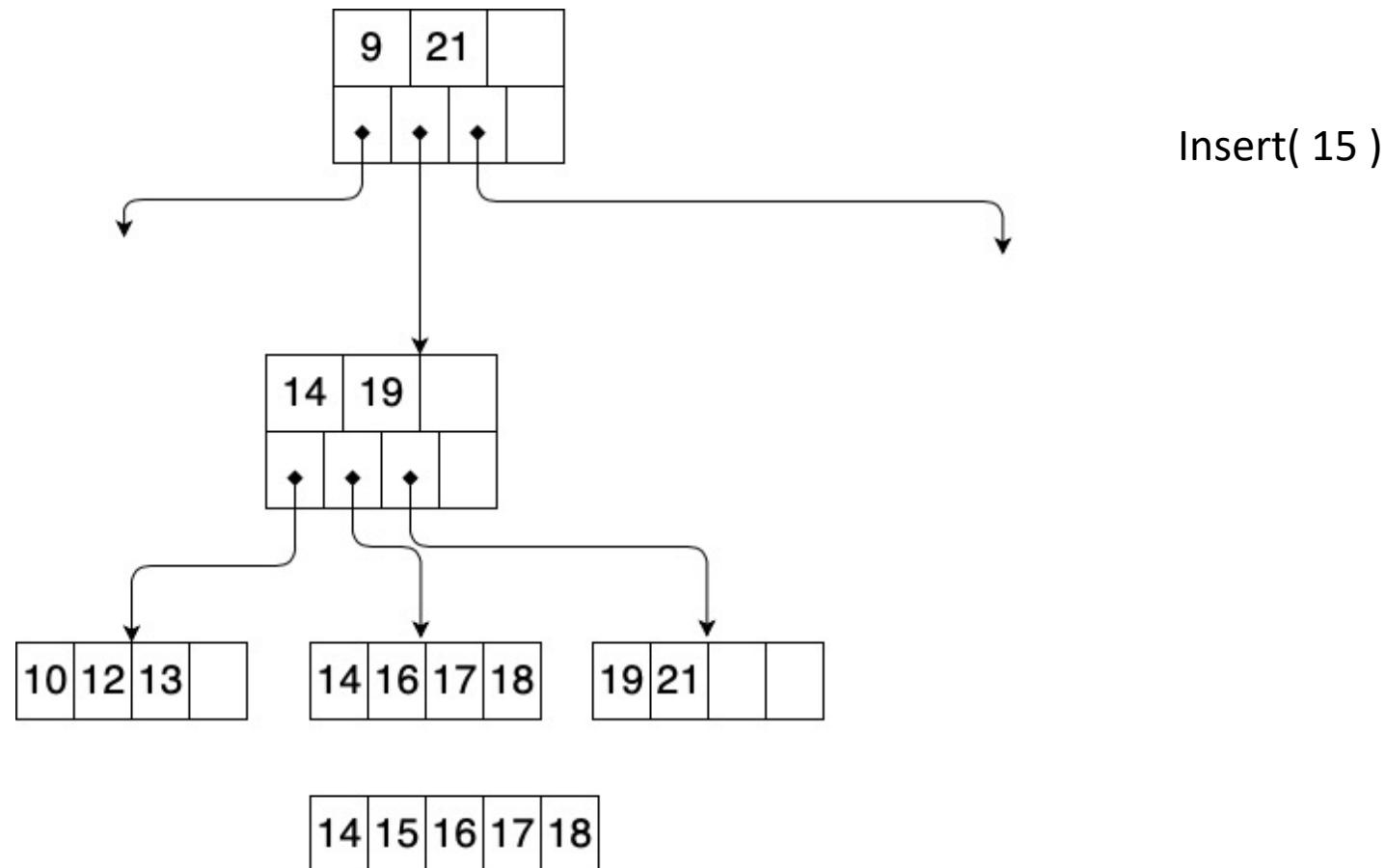
# B+ Дерево. Вставка



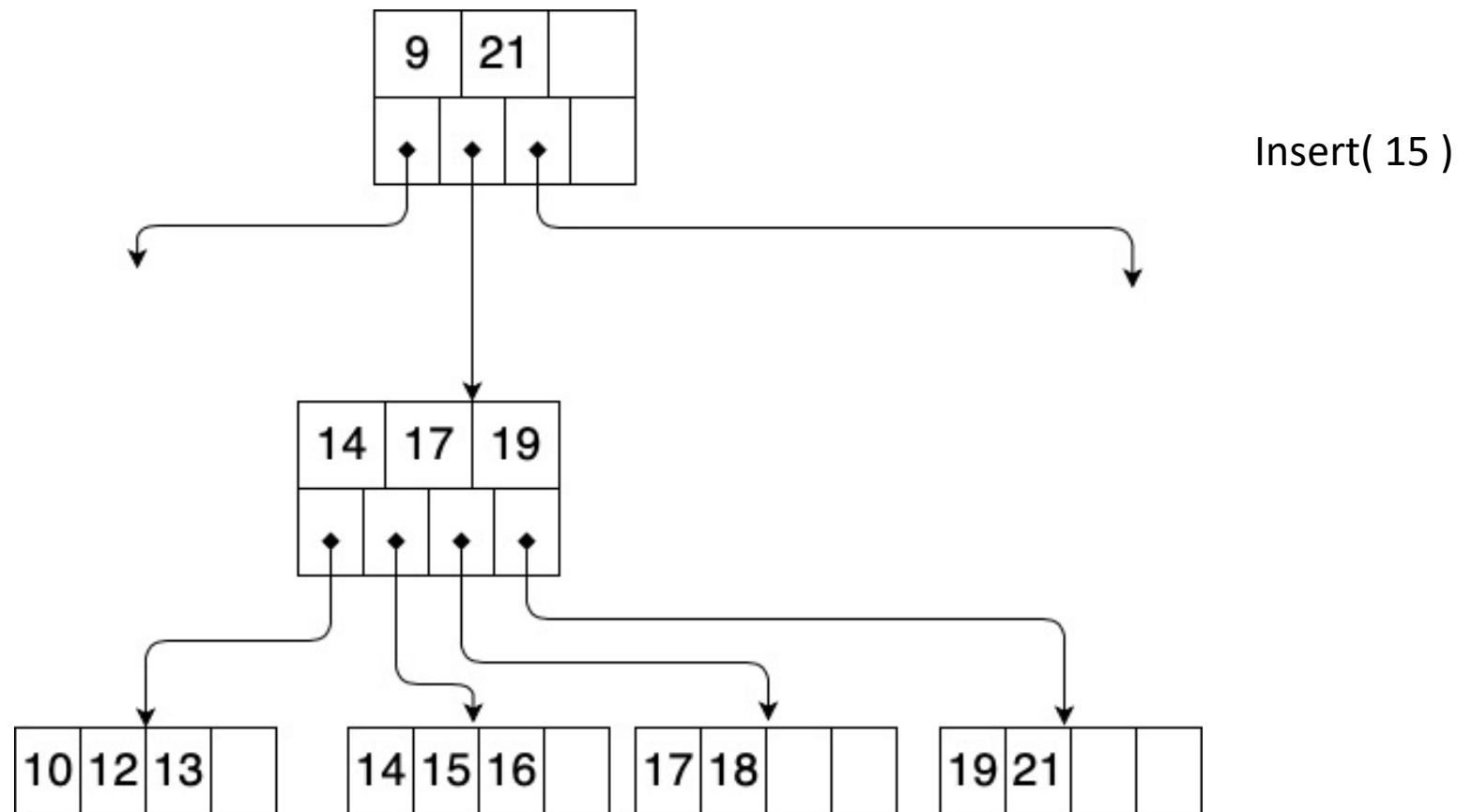
# B+ Дерево. Вставка



# B+ Дерево. Вставка



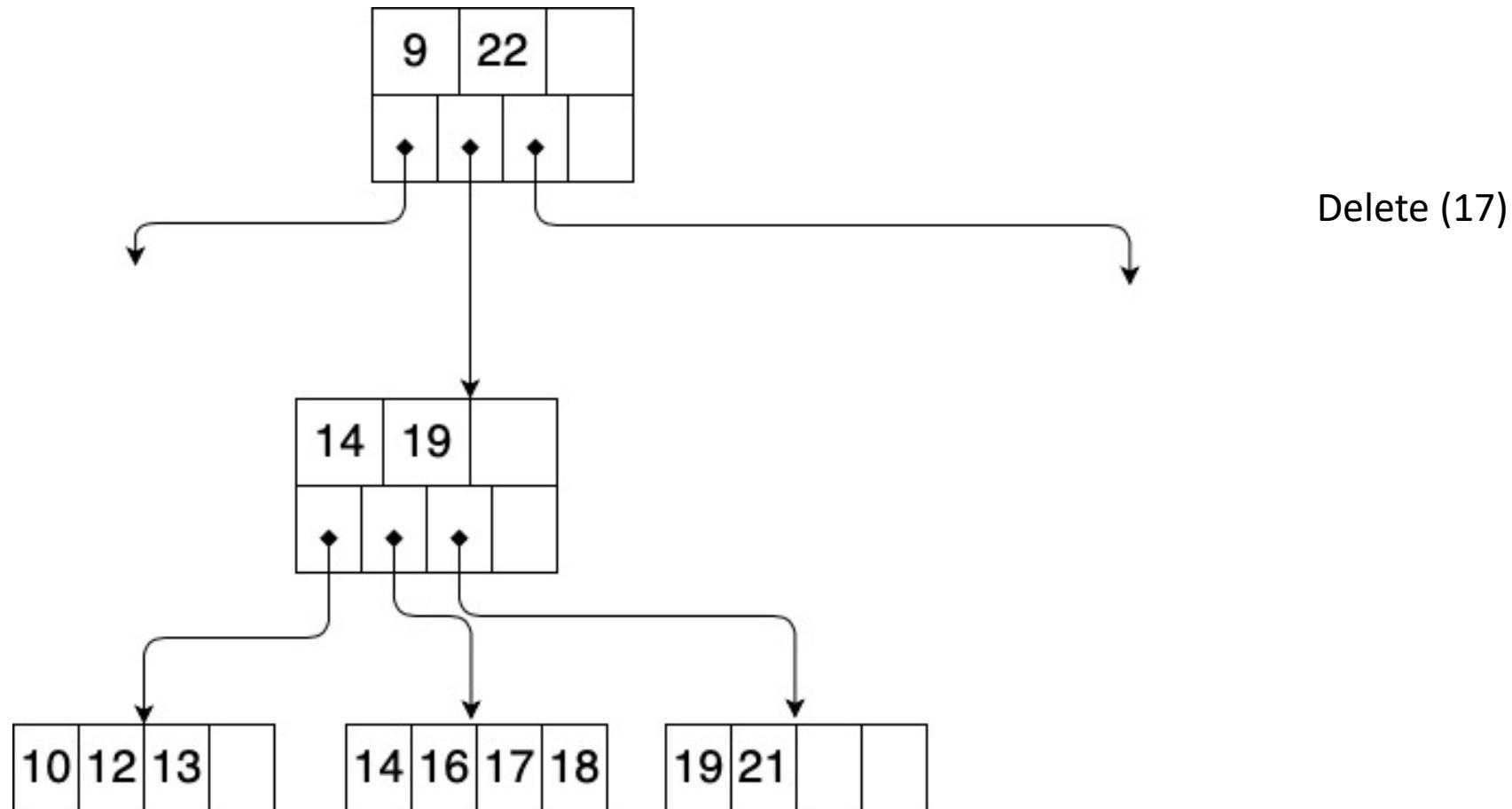
# B+ Дерево. Вставка



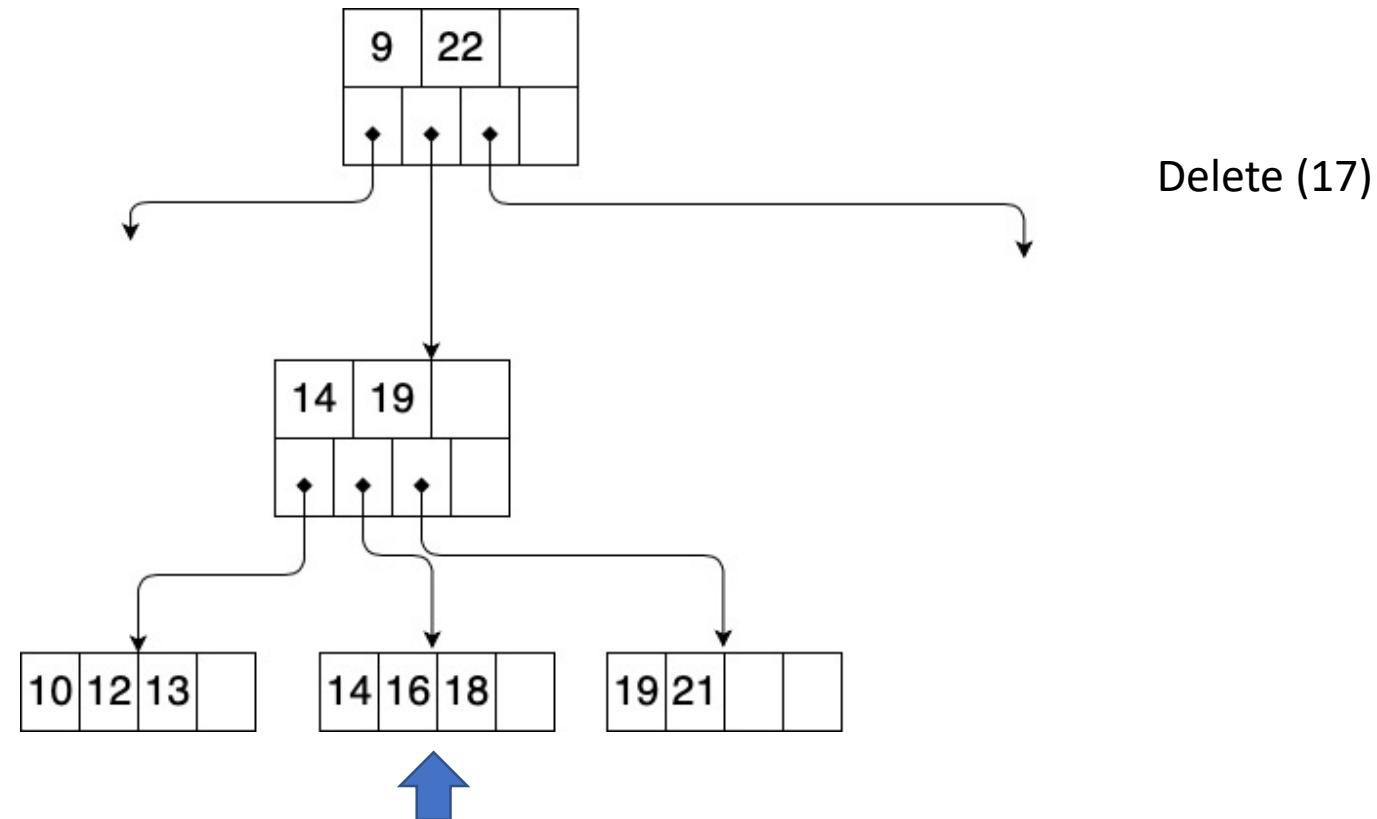
# B+ Дерево. Удаление

- Осуществить поиск элемента по ключу.
- Удалить запись.
- Если вершина осталась наполовину заполненной, алгоритм окончен.
- Иначе, попытаться перераспределить элементы с соседней вершиной
  - В случае неудачи осуществить слияние с правым или левым соседним блоком.

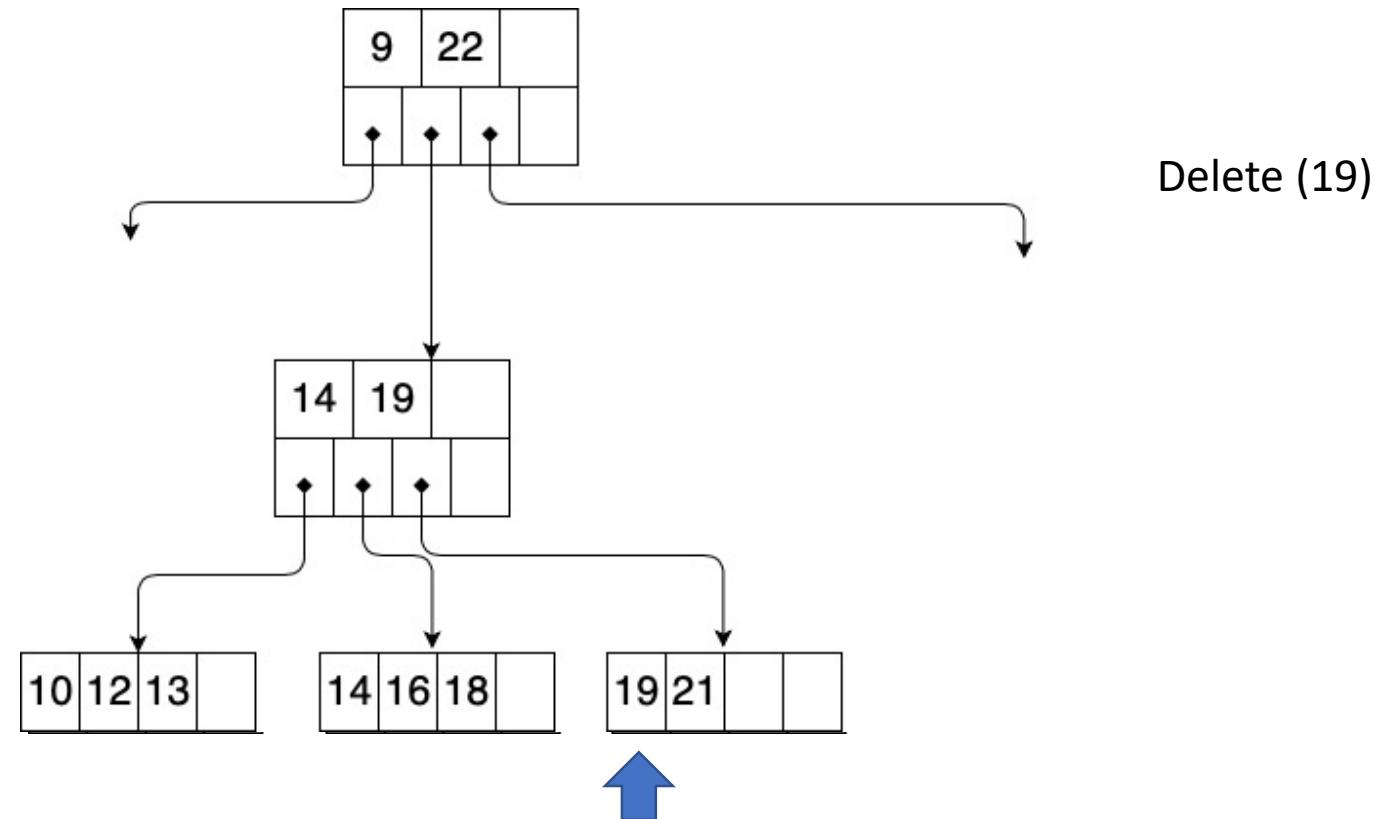
# В+ Дерево. Удаление



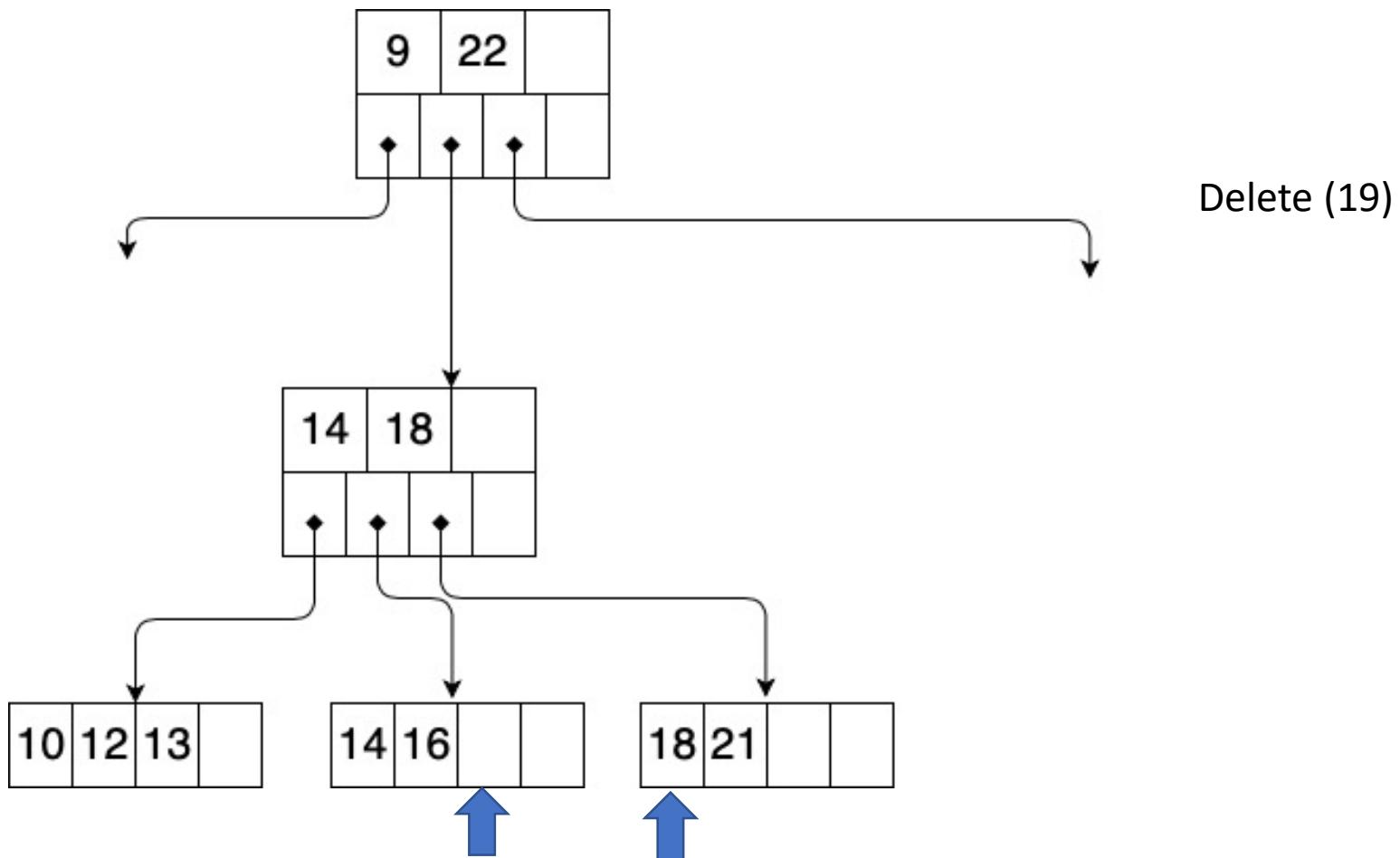
# В+ Дерево. Удаление



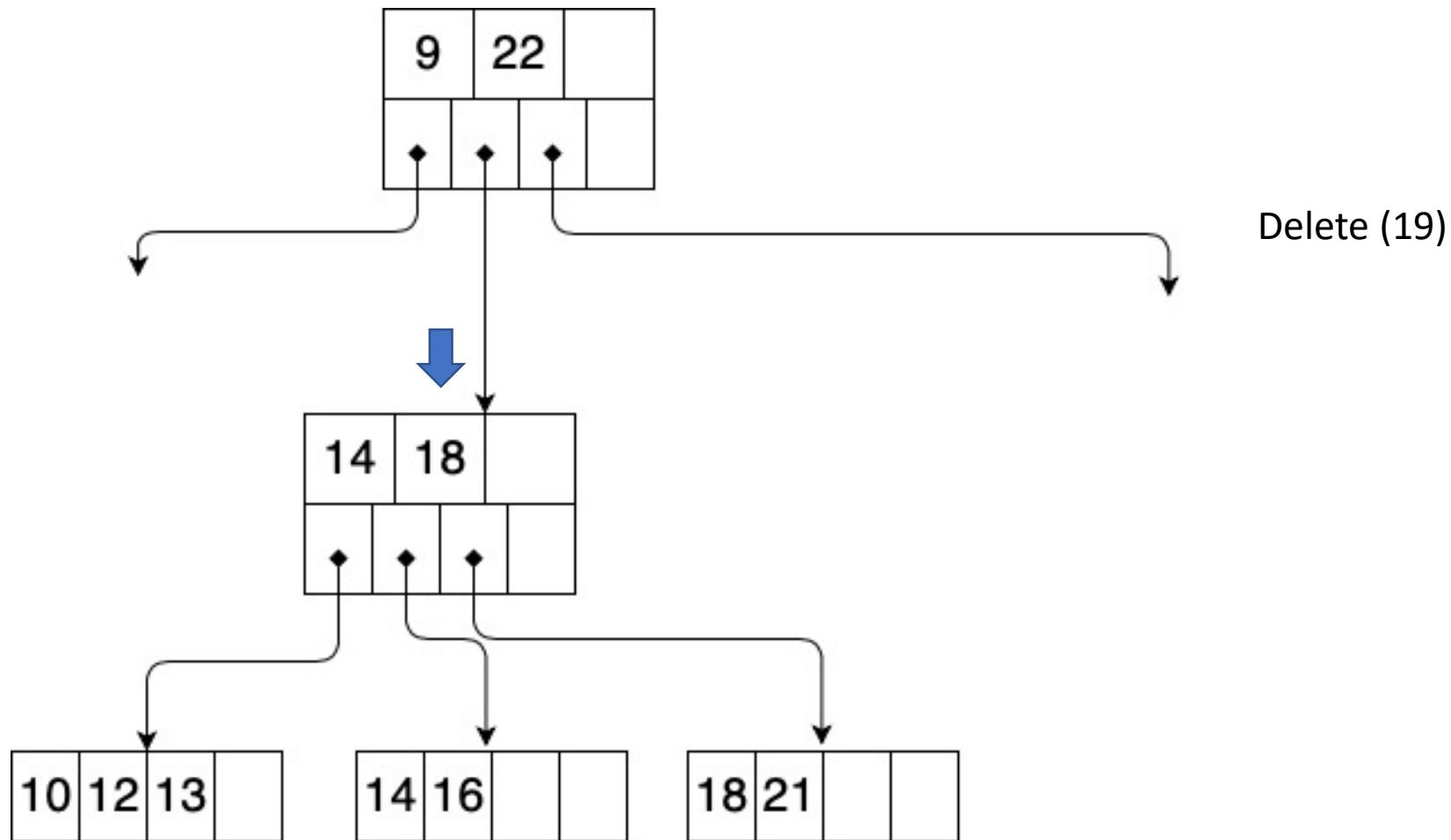
# В+ Дерево. Удаление



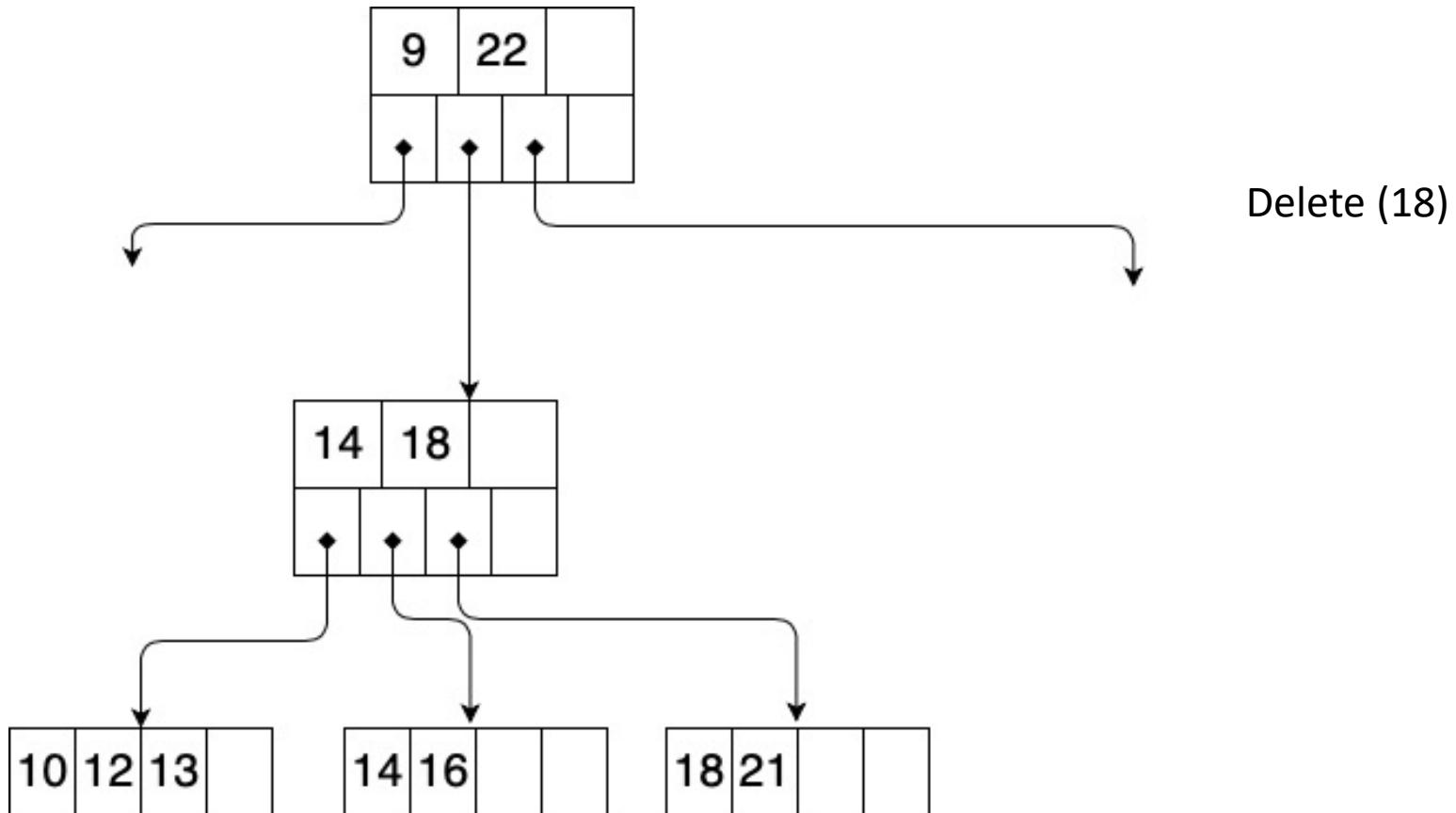
# В+ Дерево. Удаление



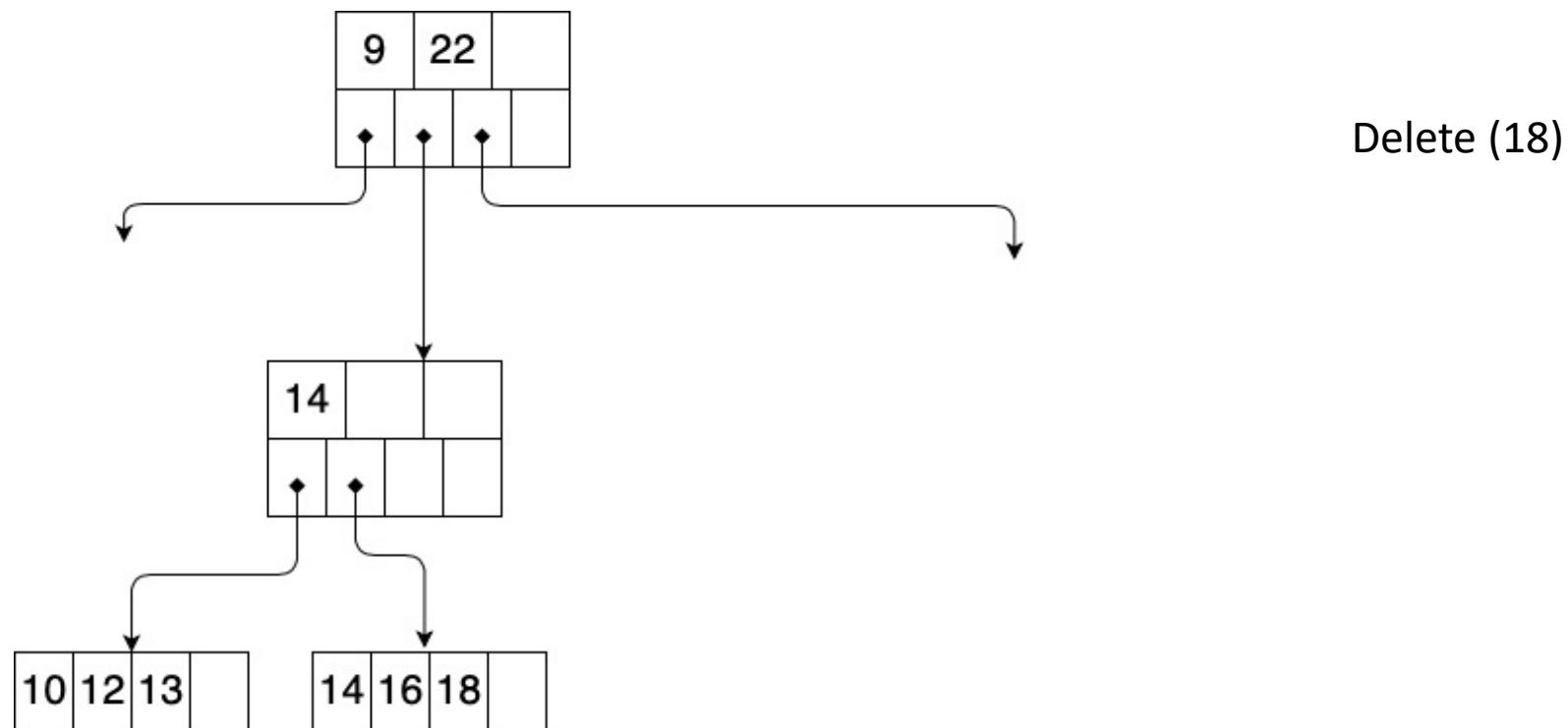
# В+ Дерево. Удаление



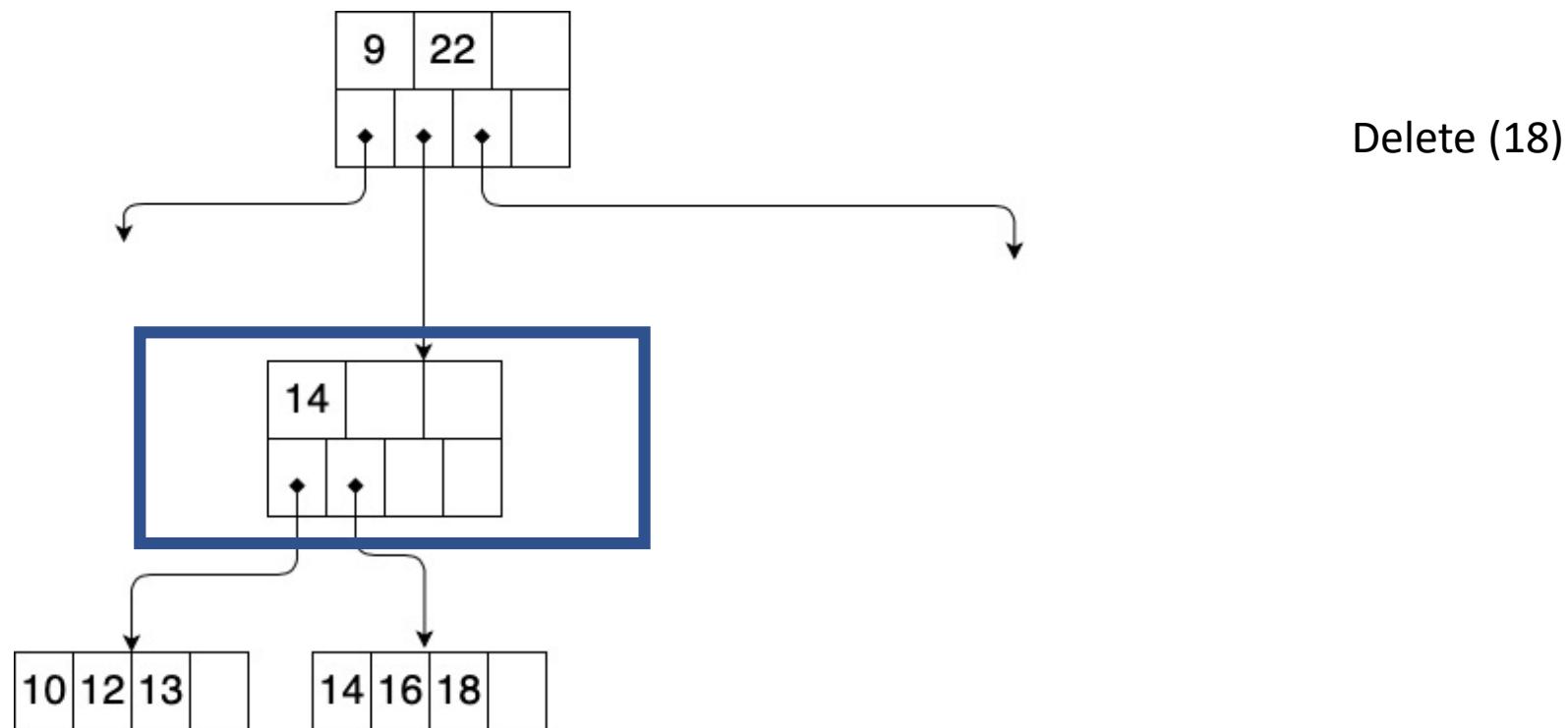
# В+ Дерево. Удаление



# В+ Дерево. Удаление



# В+ Дерево. Удаление



# Приёмы повышения эффективности

- можно делать упреждающие слияния, т.е. слияния, когда суммарный размер данных в текущей странице и соседней близок к половине
- аналогично с упреждающими расщеплениями: производить несколько раньше, когда остаётся мало свободного места
- перераспределение между соседними блоками
- возможны также такие приёмы как слияние «три в два» и расщепление «два в три».

Обратной стороной этих методов является менее экономное использование дисковой памяти.

# В-деревья. Отложенные слияния

Иногда слияния не самое эффективное решение.

Поэтому иногда выполняется отложение моментального слияния.

Для этого могут быть использованы страницы переполнения.

# Массовая вставка

- При вставке большого числа строк, поэлементная вставка может быть крайне затратной.
- Осуществляется построение маленького В+-дерева для вставляемых элементов.
- После этого осуществляется слияние двух деревьев.

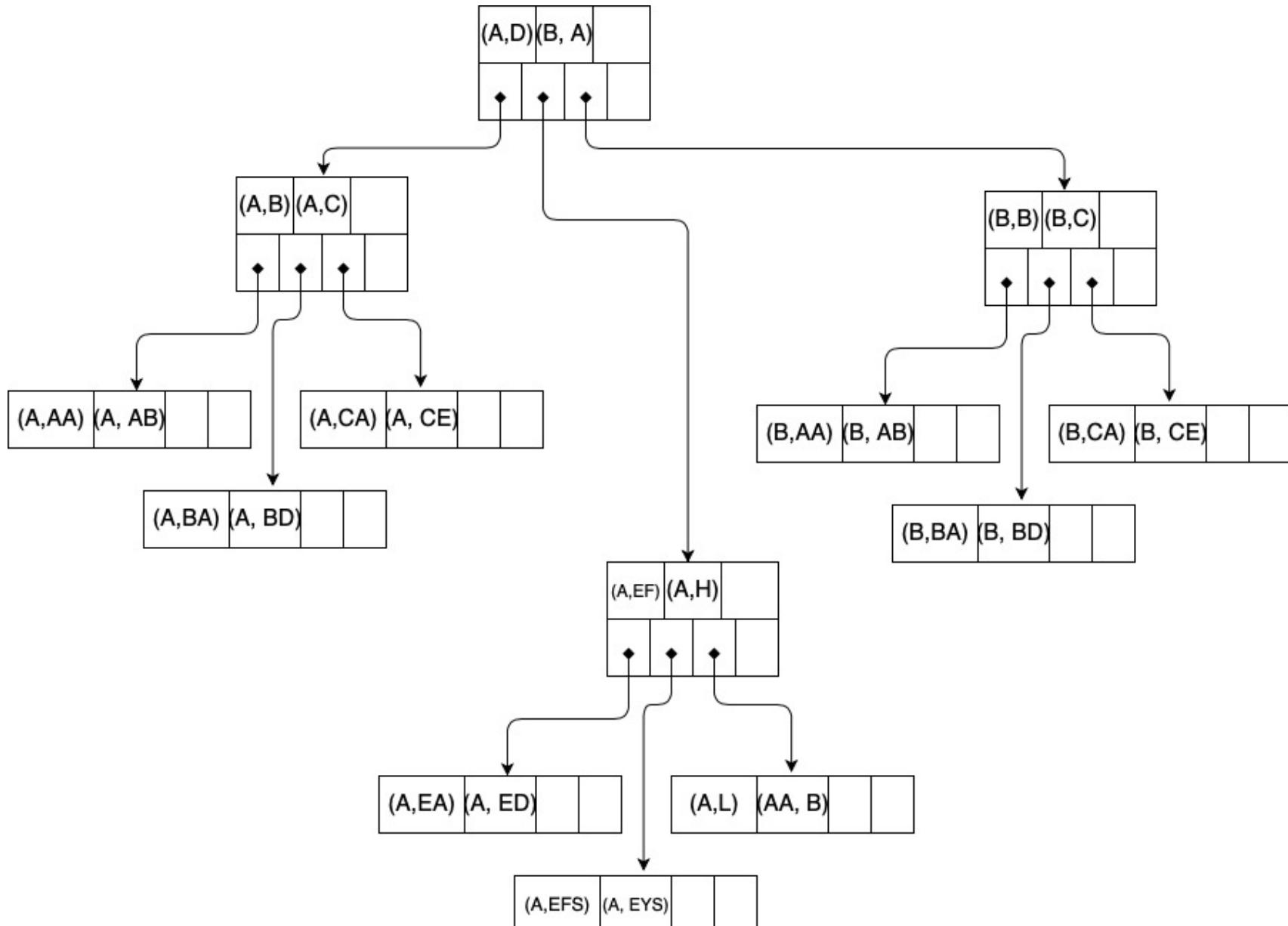
# Условие выборки для составного ключа

СУБД используют могут использовать индексы для составного ключа

Пример: Составной ключ по полям (A, B). A, B - varchar

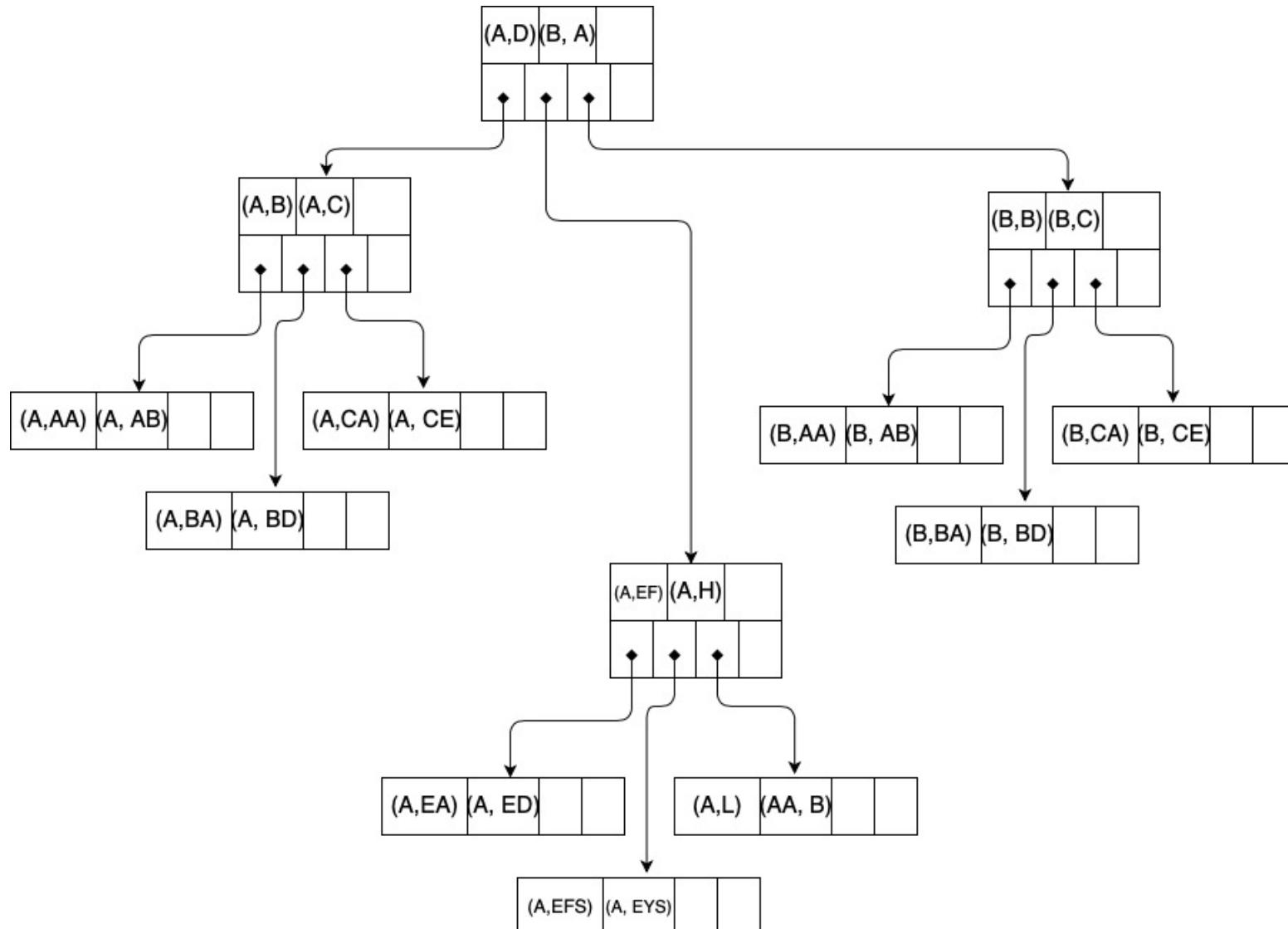
- Select \* from t where A = <x> and B = <y>

# B+ Дерево. Составной ключ



Пусть задан  
составной  
индекс по полям  
(X varchar, Y varchar)

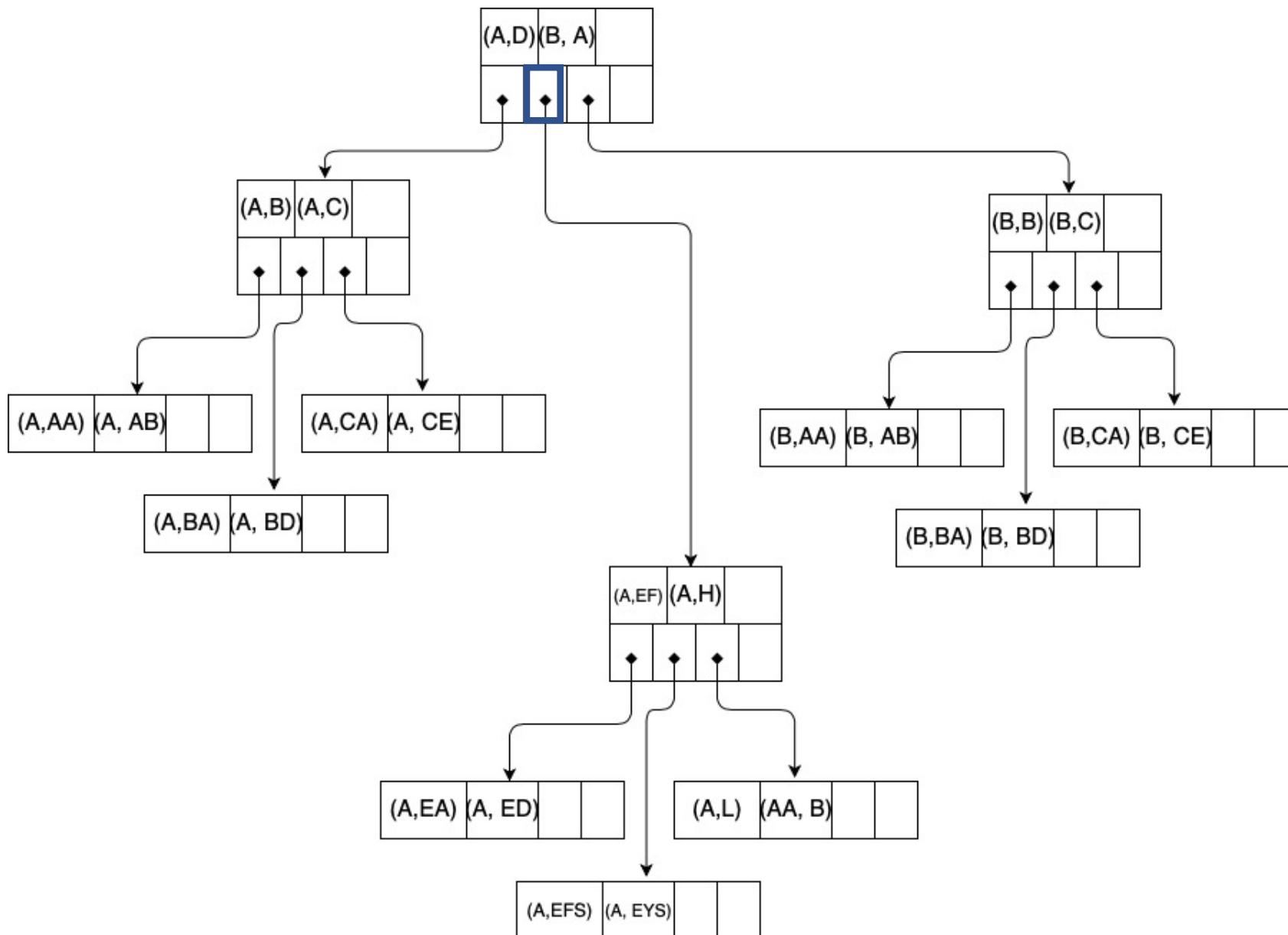
# B+ Дерево. Составной ключ



```
SELECT *  
FROM T  
WHERE X = 'AA'  
AND Y = 'B'
```

# B+ Дерево. Составной ключ

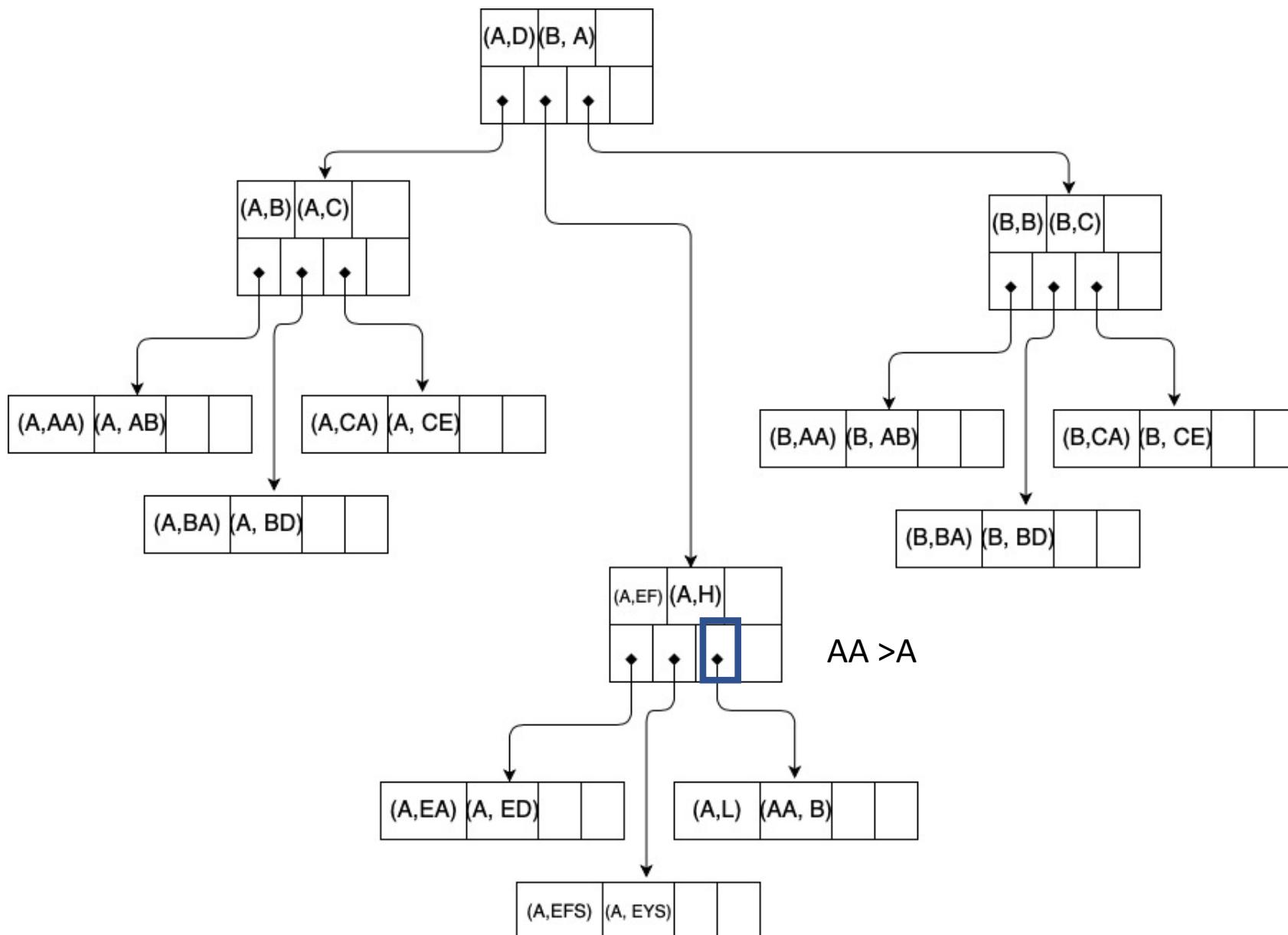
AA > A && AA < B



```
SELECT *  
FROM T  
WHERE X = 'AA'  
AND Y = 'B'
```

# B+ Дерево. Составной ключ

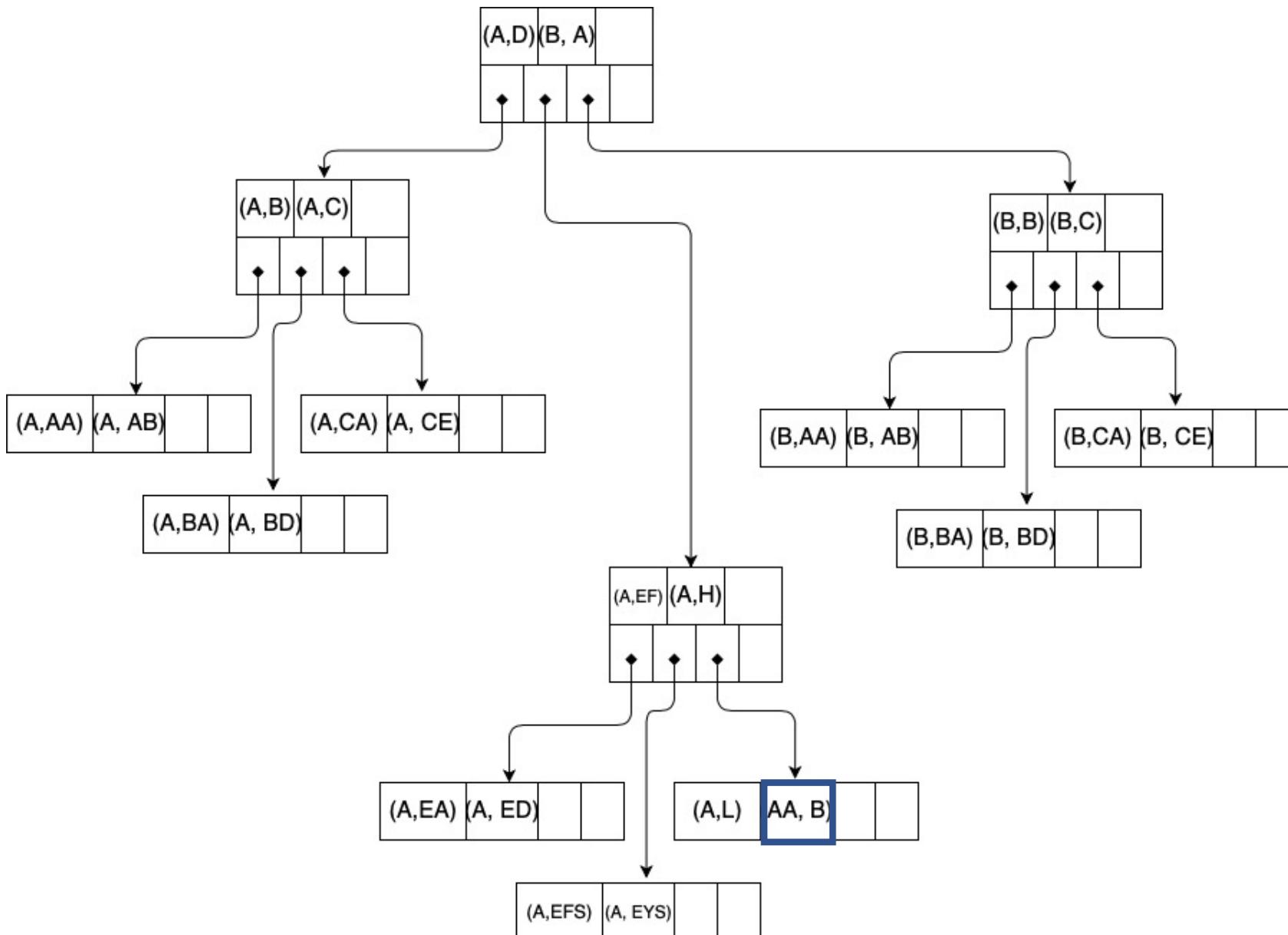
AA > A && AA < B



```
SELECT *  
FROM T  
WHERE X = 'AA'  
AND Y = 'B'
```

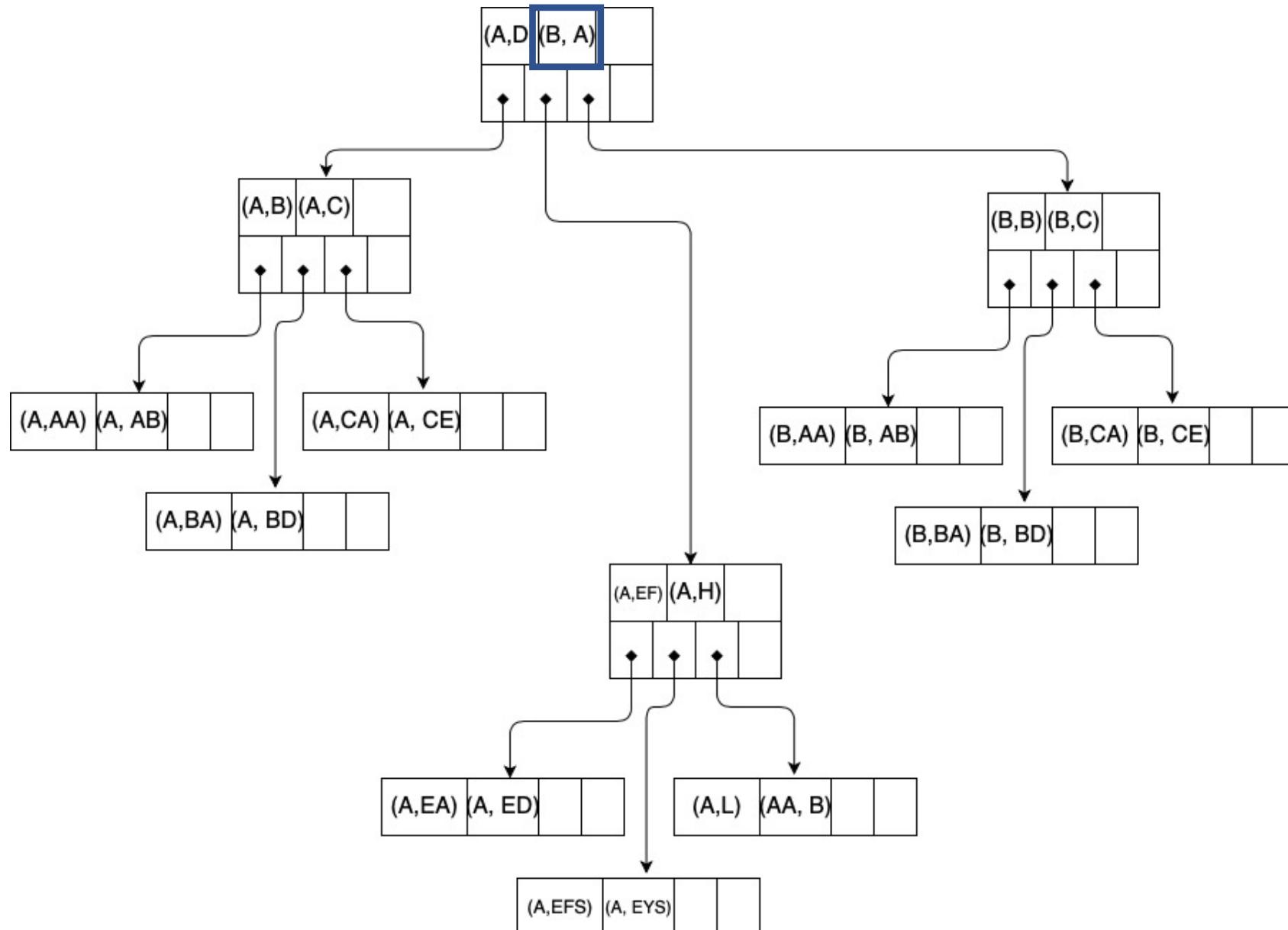
# B+ Дерево. Составной ключ

AA > A && AA < B



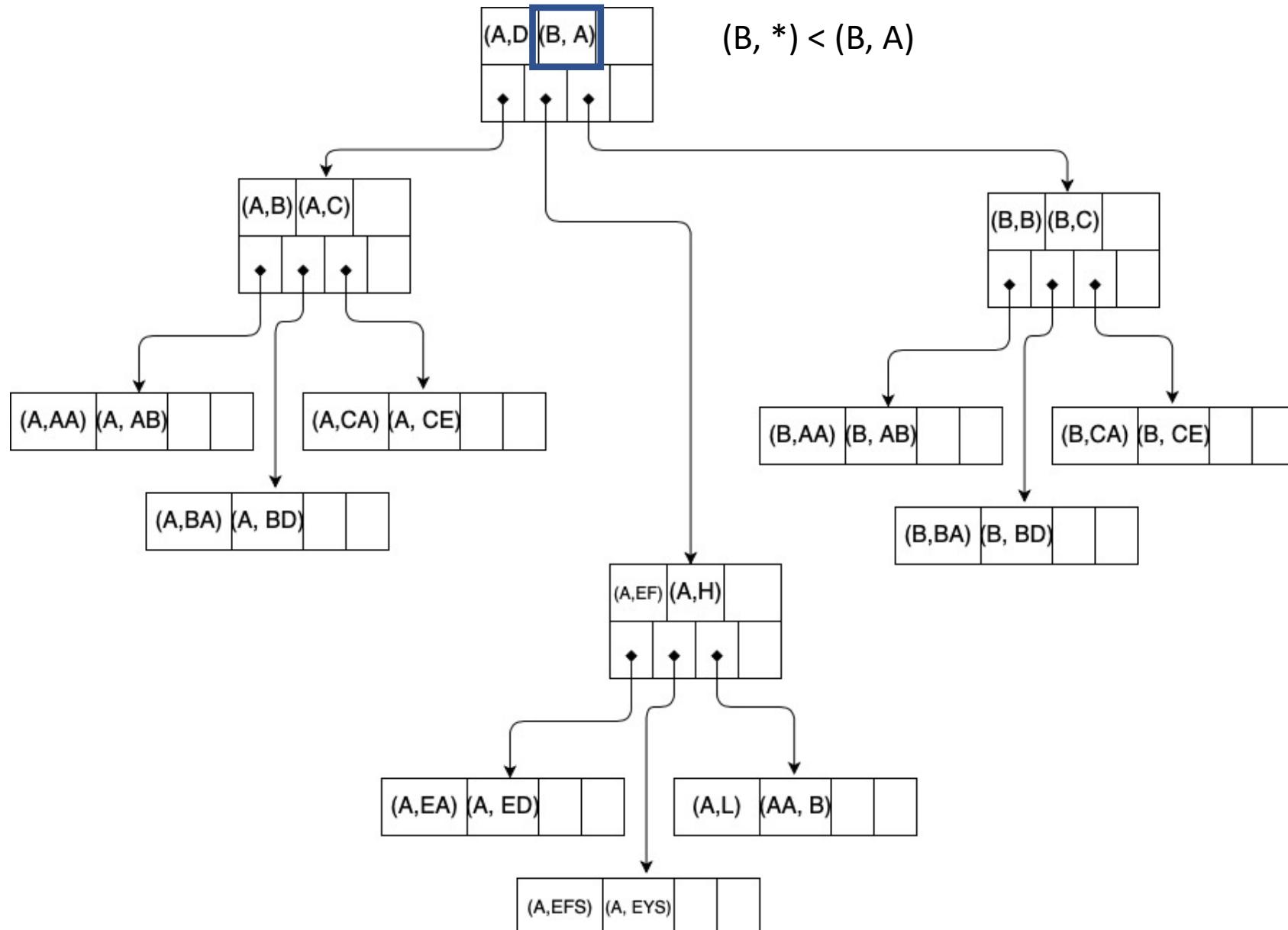
```
SELECT *  
FROM T  
WHERE X = 'AA'  
AND Y = 'B'
```

# B+ Дерево. Составной ключ



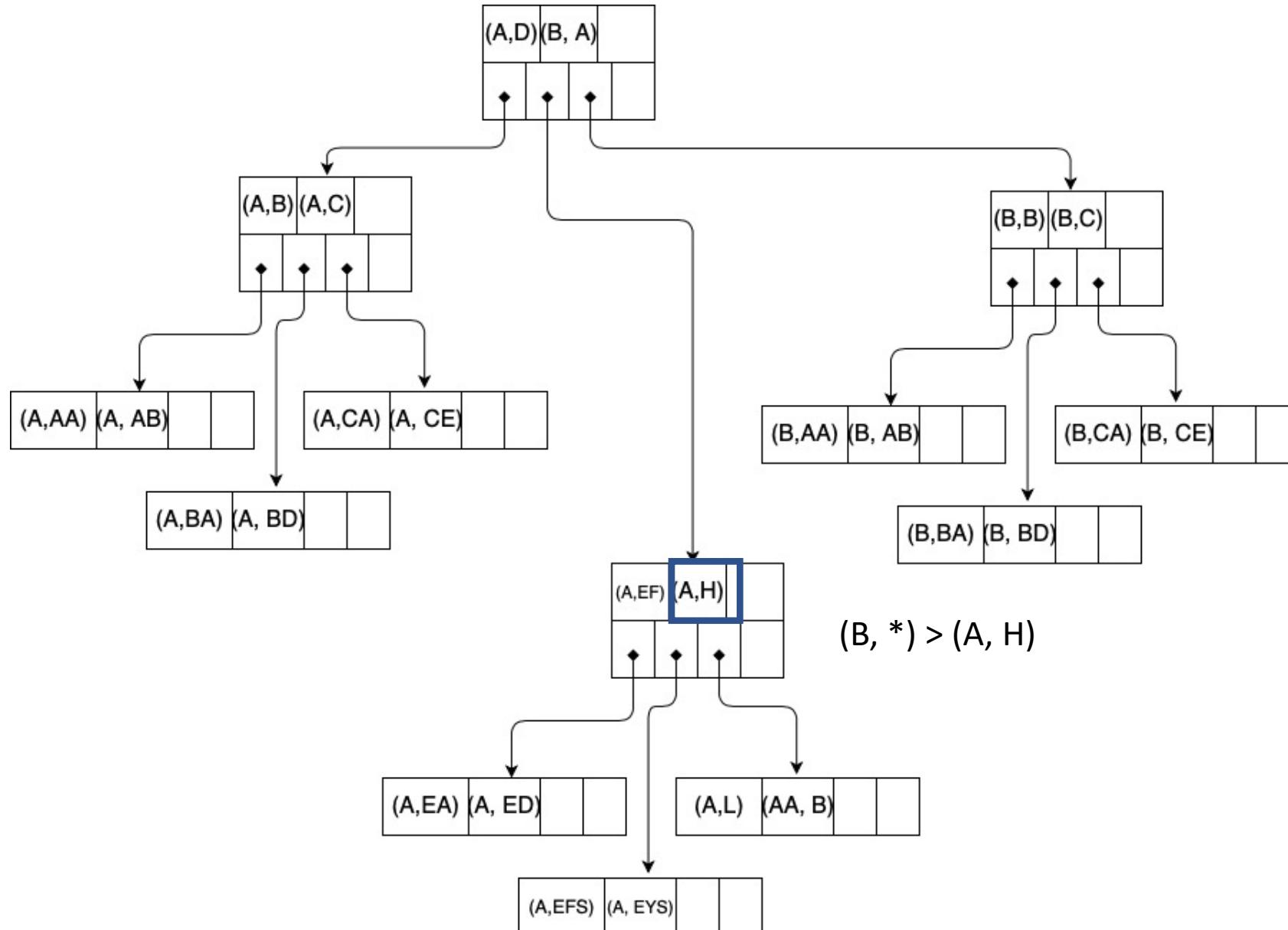
```
SELECT *  
FROM T  
WHERE X = 'B'
```

# B+ Дерево. Составной ключ



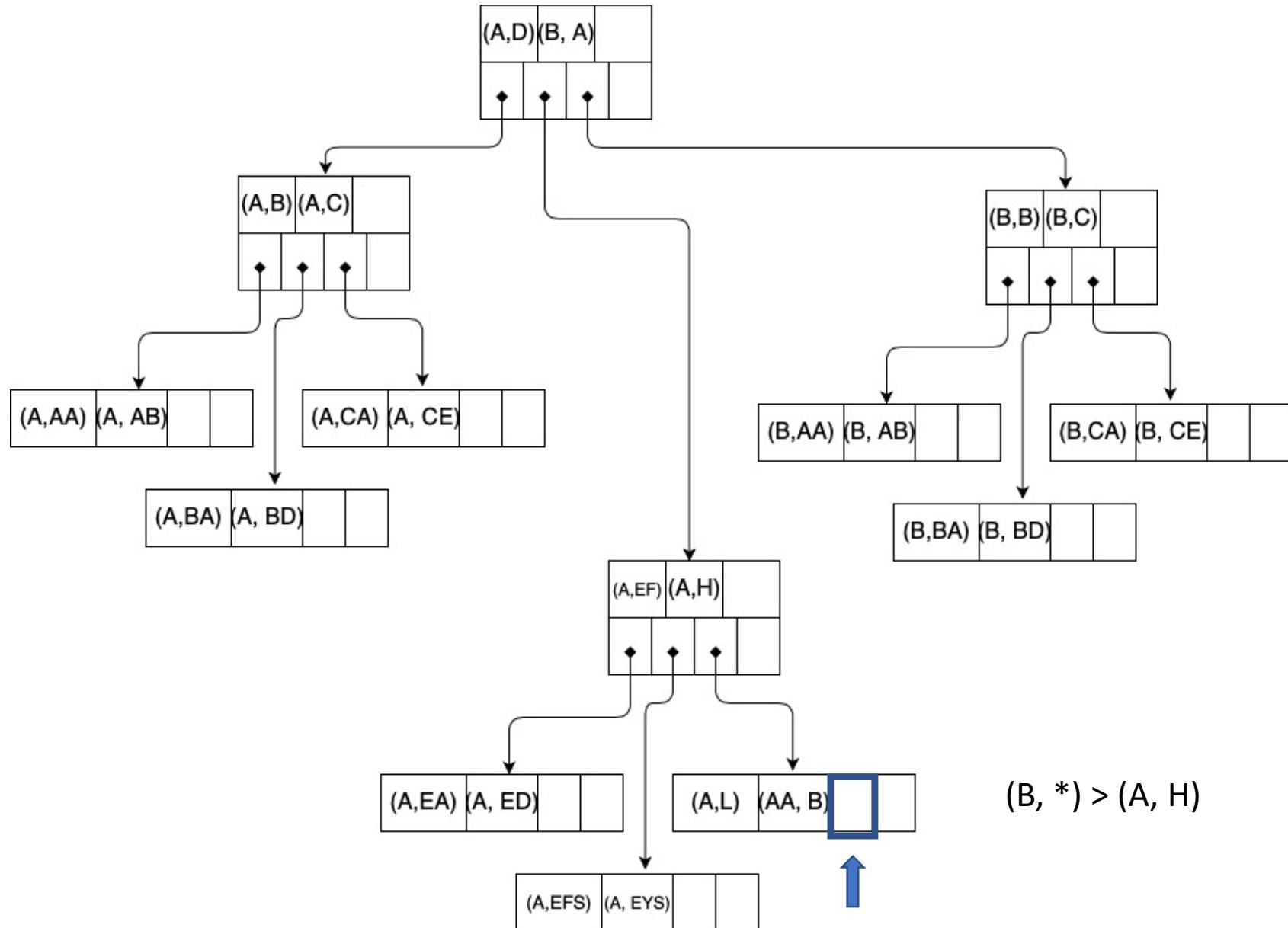
SELECT \*  
FROM T  
WHERE X = 'B'

# B+ Дерево. Составной ключ



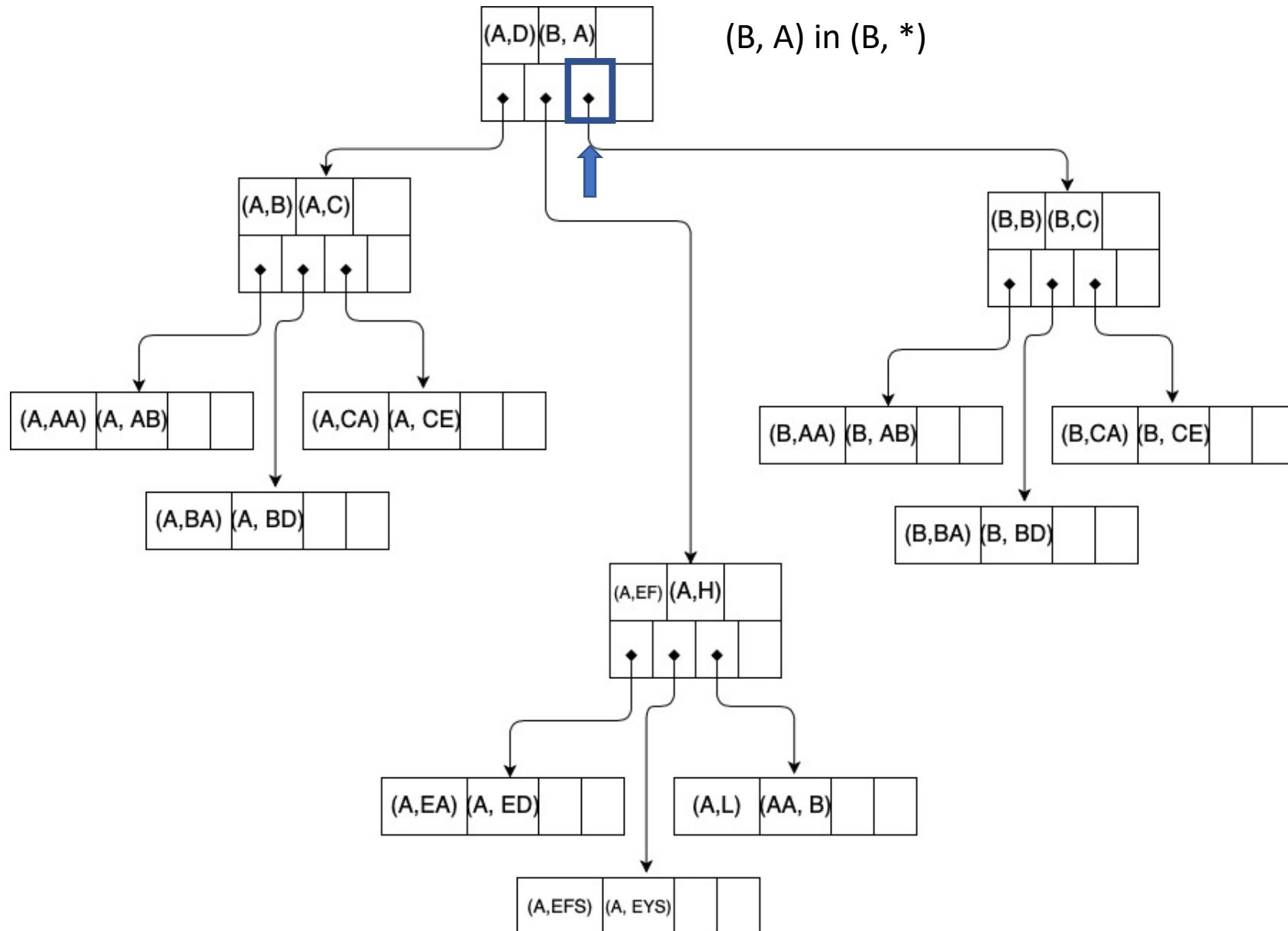
```
SELECT *  
FROM T  
WHERE X = 'B'
```

# B+ Дерево. Составной ключ



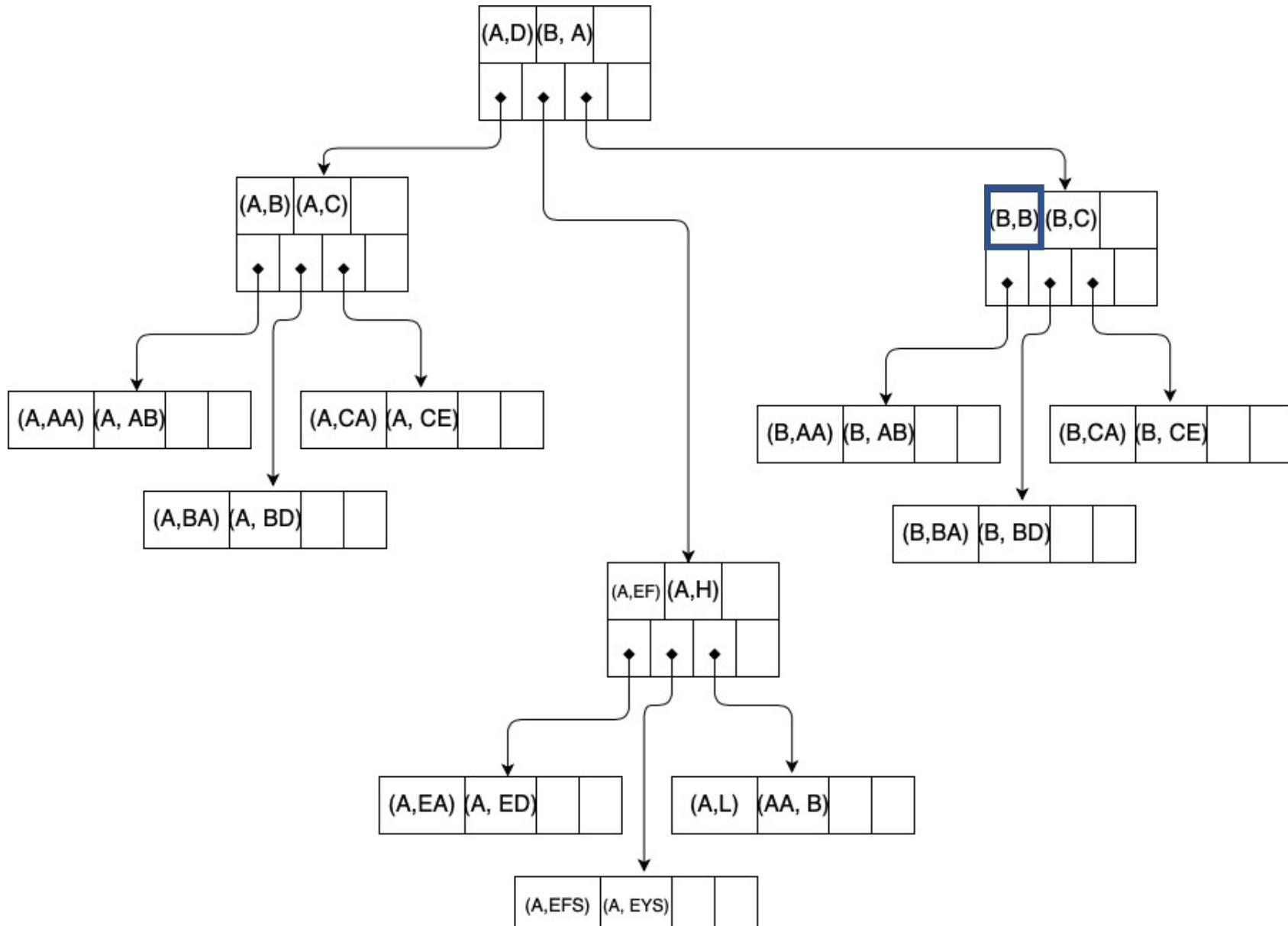
SELECT \*  
FROM T  
WHERE X = 'B'

# B+ Дерево. Составной ключ



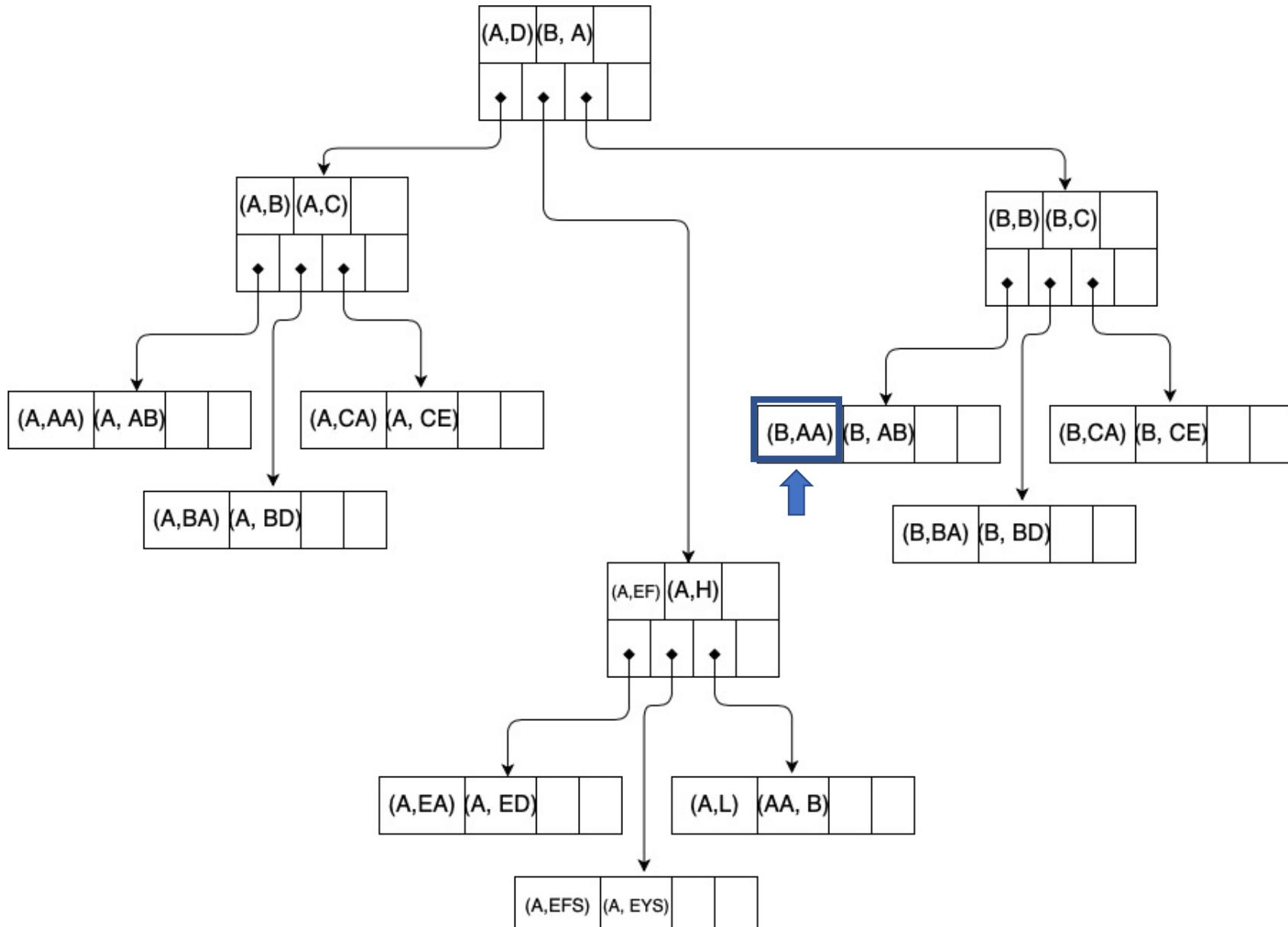
```
SELECT *
FROM T
WHERE X = 'B'
```

# B+ Дерево. Составной ключ



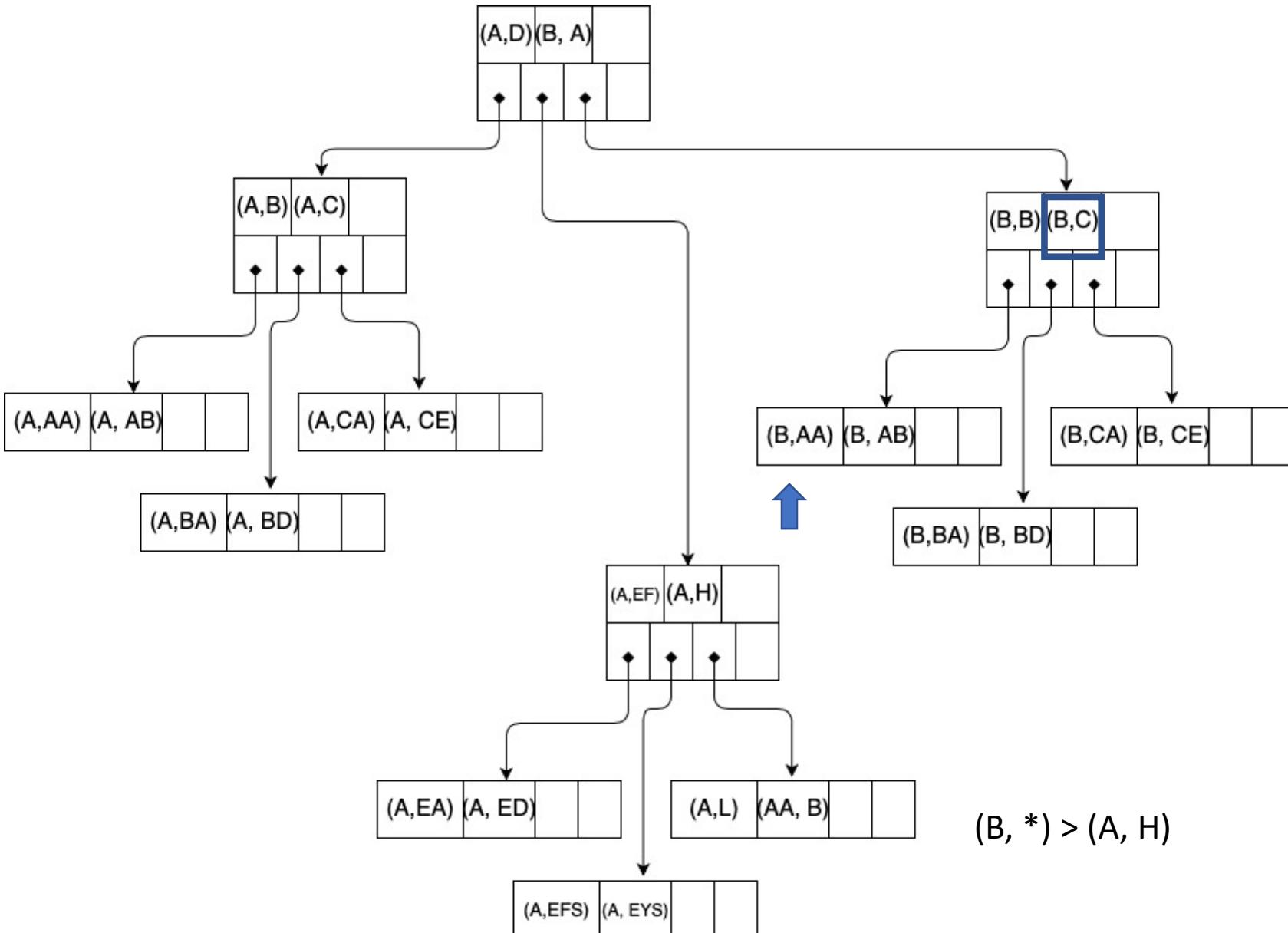
```
SELECT *  
FROM T  
WHERE X = 'B'
```

# B+ Дерево. Составной ключ

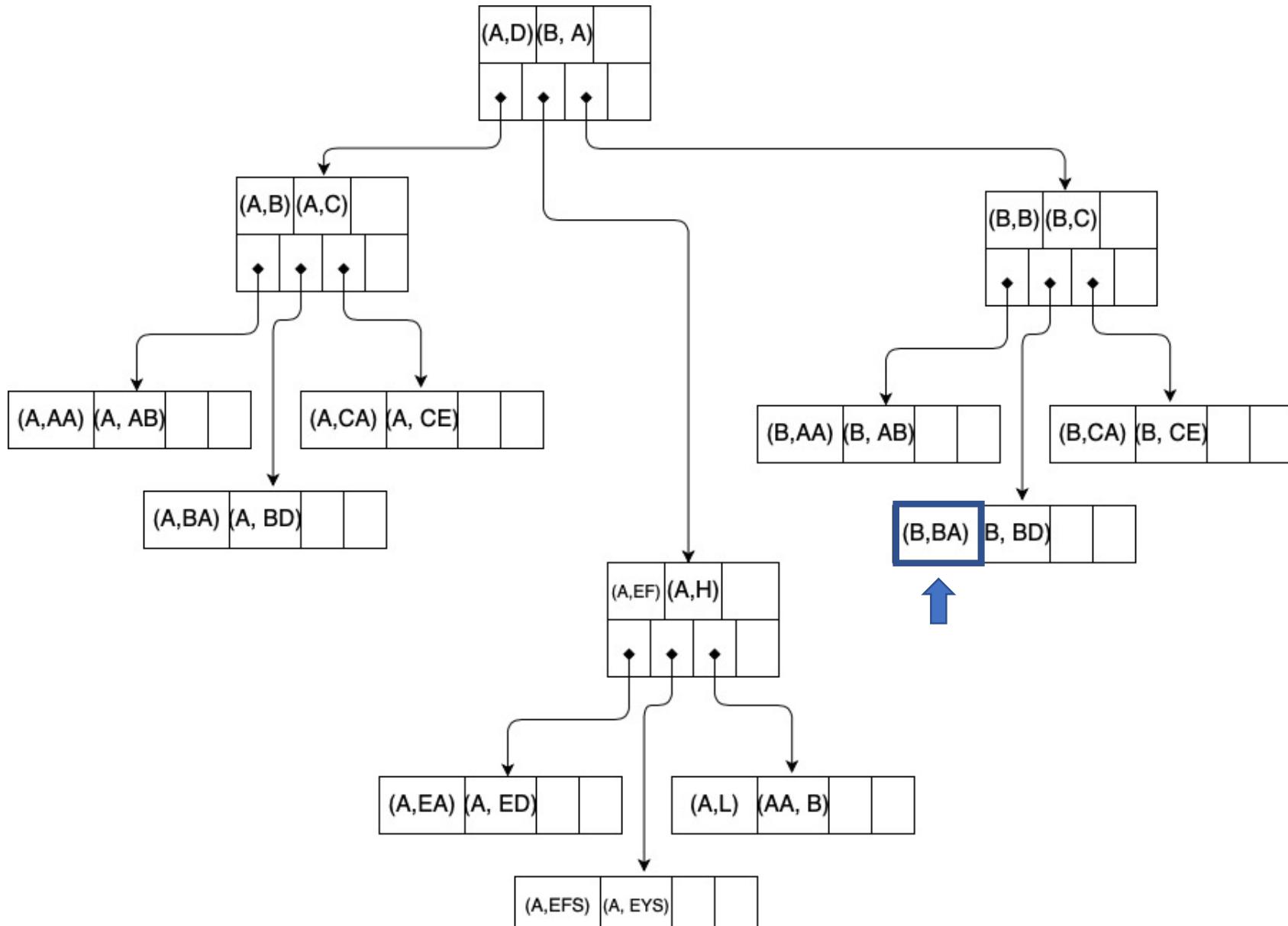


SELECT \*  
FROM T  
WHERE X = 'B'

# B+ Дерево. Составной ключ

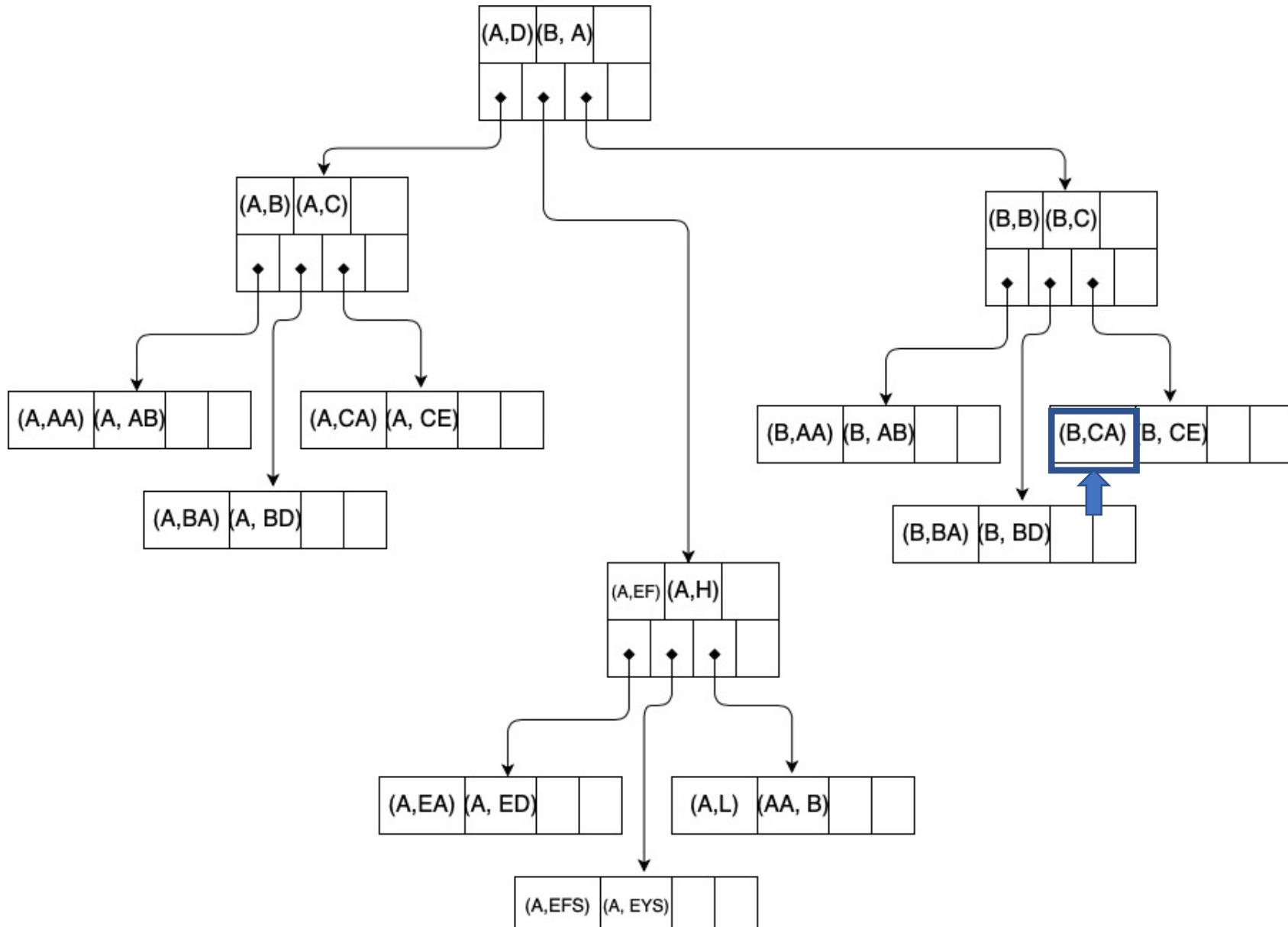


# B+ Дерево. Составной ключ



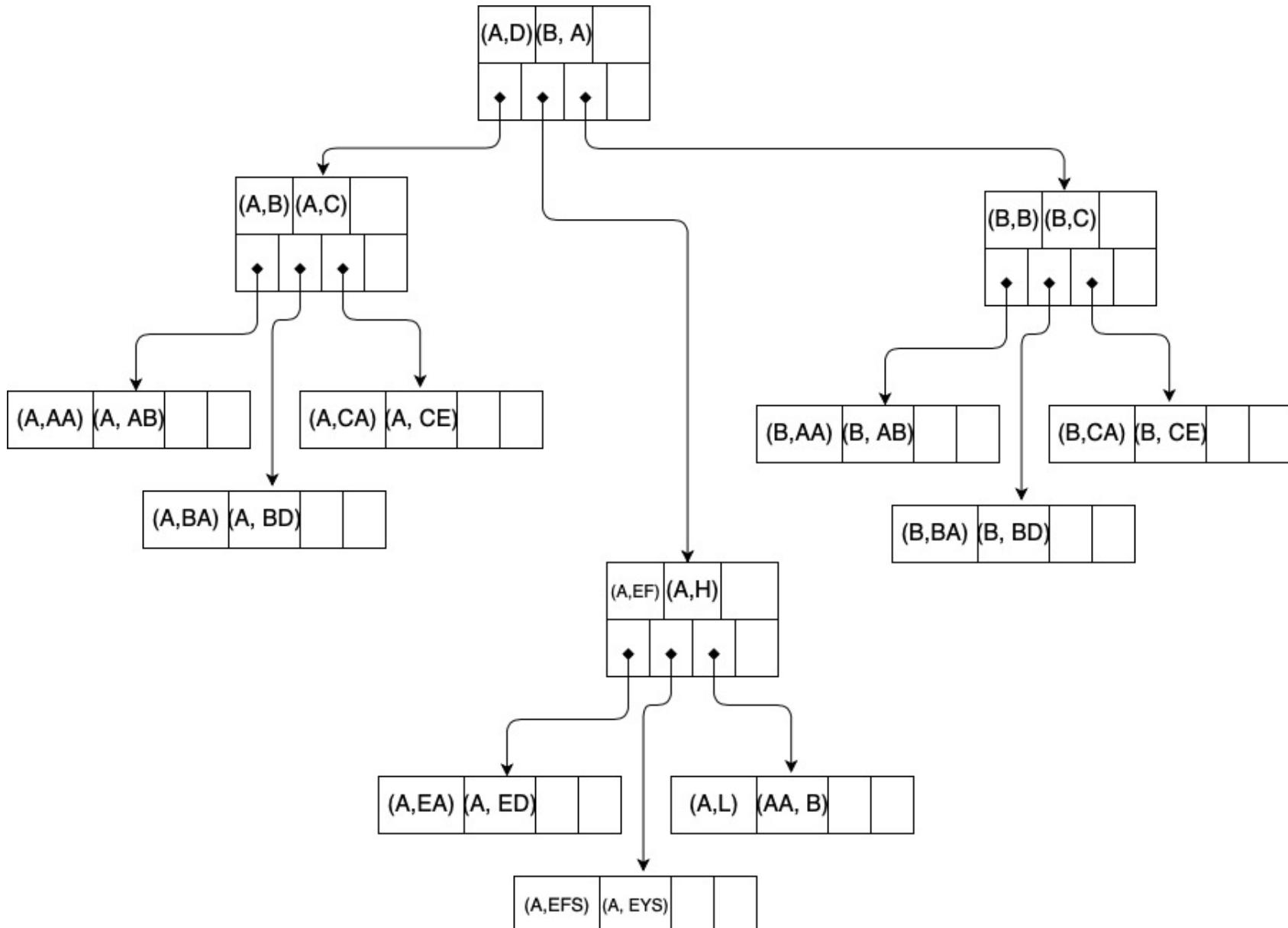
```
SELECT *
FROM T
WHERE X = 'B'
```

# B+ Дерево. Составной ключ



```
SELECT *
FROM T
WHERE X = 'B'
```

# B+ Дерево. Составной ключ



SELECT \*  
FROM T  
WHERE Y = 'B'

???

# B-деревья. Размер вершин

Чем медленнее диск, тем больше размер вершины

- HDD – 1 МВ
- SSD – 10 КВ
- Память – 512 В

В зависимости от типа операций размер также может варьироваться.

# Поиск внутри вершины

- Линейный
- Бинарный
- Апроксимация – применение вероятностных метрик для понимания, где будет элемент на странице.