ТЕХНОСФЕРА

Индексация
часть 2

Сергукова Юлия,
программист отдела инфраструктуры проекта
Поиск@Mail.Ru

В прошлой серии:

1. **Обратный индекс**
2. **Поиск по обратному индексу, пересечение блоков**
3. **Сжатие индекса**

Темы занятия:

1. **Сжатие индекса: Simple9**
2. **Бинарные данные в Python**
3. **Словарь**
4. **Дерево запроса (Q-Tree)**
5. **Работа с индексом**

Сжатие

1. VarByte – быстрый, но много места (гранулярность – 1 байт)
2. Fibbonaci – компактный, но медленный
3. Elias Gamma – избыточный (много места занимают 0)

Simple9

Машинное слово: 4байта = 32бита

32бита = 4бита (code) + 28бит (payload)

Simple9

Payload (28бит):

- 1 x 28бит
- 2 x 14бит
- 3 x 9бит (+1бит теряем)
- 4 x 7бит
- 5 x 5бит (+3бита теряем)
- 7 x 4бит  Где 6x4бита?
- …
- 28 x 1бит

Simple9

Плюсы:

- Очень быстрый (2*10^9 чисел / сек)
- Компактный

Минус:

- Избыточен для 1 числа

Что использовать?

- Постинг-листы – Simple9
- Вхождения в документ (координаты) – VarByte

Бинарные данные в Python

10100110 – 8бит (1байт)

Строка «10100110» - 8байт

Большие объемы нужно хранить в бинарном виде

Бинарные данные в Python

```
import struct
```

- struct.pack(fmt, v1, v2, …) – сериализация
- struct.unpack(fmt, bstr) – десериализация
- struct.calcsize(fmt)

Бинарные данные в Python

import array

В Python массив – не массив (а вектор с указателями на объекты)

array даёт более «честный» массив

Бинарные данные в Python

- array.array(‘c’) # медленно
- cStringIO.cStringIO()
- bytearray

Словарь

Терм <-> termID

Словарь очень большой (опечатки, несловарные слова и т.д.)

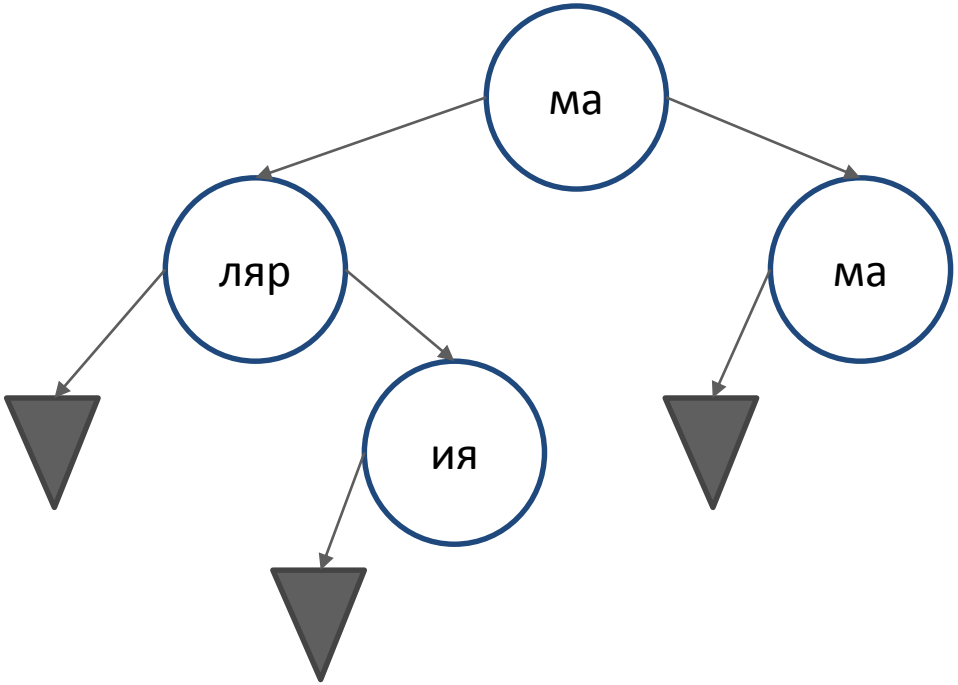
Словарь

Как хранить?

Словарь

Как хранить?

- хэш-таблицы (быстро)
- префиксные деревья (компактно и упорядоченно)



Словарь

Деревья:

- бинарные
- B-деревья

Нечеткий поиск

Метасимволы:

- ко*ар («комар» и «кошмар»)

Нечеткий поиск

- Префиксное дерево: A*
- Постфиксное дерево: *A
- ???: A*B

Нечеткий поиск

- Префиксное дерево: A*
- Постфиксное дерево: *A
- Конъюнкция результатов от префиксного и постфиксного: A*B

Нечеткий поиск

Как искать A*B*C?

Нечеткий поиск

N-граммы

Пусть N=3 → нарезаем слово + маркеры границы по 3 символа

кошка → $кошка$ → $ко + кош + ошк + шка + ка$

Нечеткий поиск

N-граммы

Пусть N=3 → нарезаем слово + маркеры границы по 3 символа

кошка → $кошка$ → $ко + кош + ошк + шка + ка$

ко*ка → $ко + ка$

Нечеткий поиск

Почему большой поиск не использует такой нечеткий поиск?

Стоп-слова

- Слова с наивысшей частотой
- Не имеют собственного смысла (нужен контекст)

Примеры:

- «и», «или», «где»
- специфичные: «читать» при поиске по книгам

Стоп-слова

Что делать?

Стоп-слова

Что делать?

- выкидывать – не найдем «что?где?когда?» и проблемы с цитатами
- оставлять – сильно увеличит словарь

Стоп-слова

Что делать?

- инверсный индекс – список документов, где НЕТ этого слова
- учёт контекста

Стоп-слова

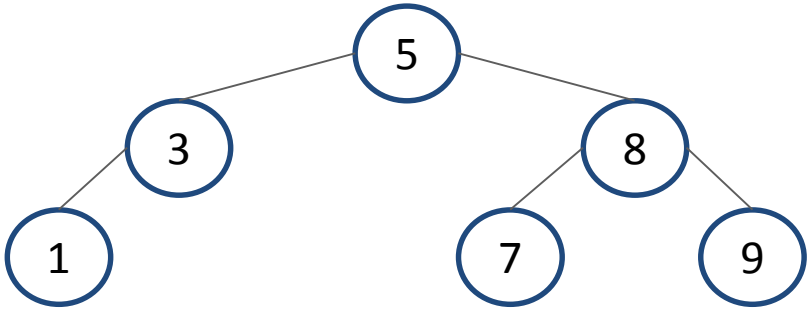
Учёт контекста

- «война и мир»:
  - война
  - йна_и
  - и_мир
  - мир

Словари в Python

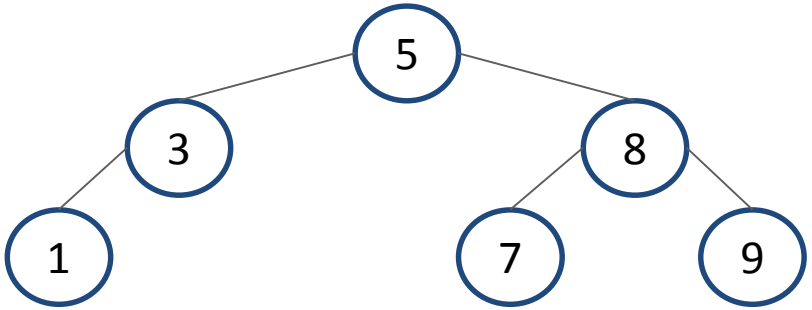
- dict()
- сериализовать структуру на диск

Сериализация дерева



5 3 8 1 -1 7 9

Сериализация дерева



5 (1, 3) 3 (2, -1) 1 (-1) 8 (4, 5) 7 (-1) 9 (-1)
0 1 2 3 4 5

Сериализация хэш-таблицы

Хэш-таблица – набор «корзин»

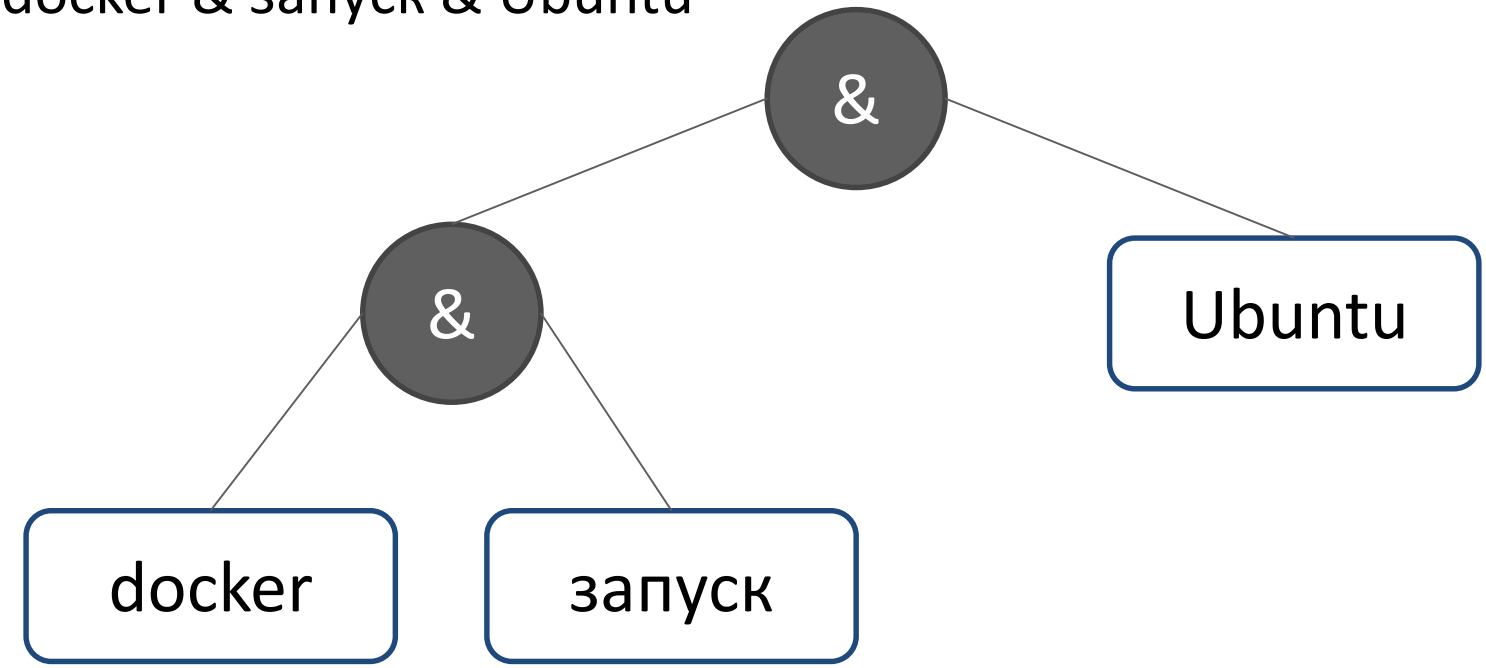
3	2	1	3	15	18	4	5	14	17

- количество корзин (3)
- размеры корзин (2, 1 и 3)
- содержимое корзин

Исполнение запроса

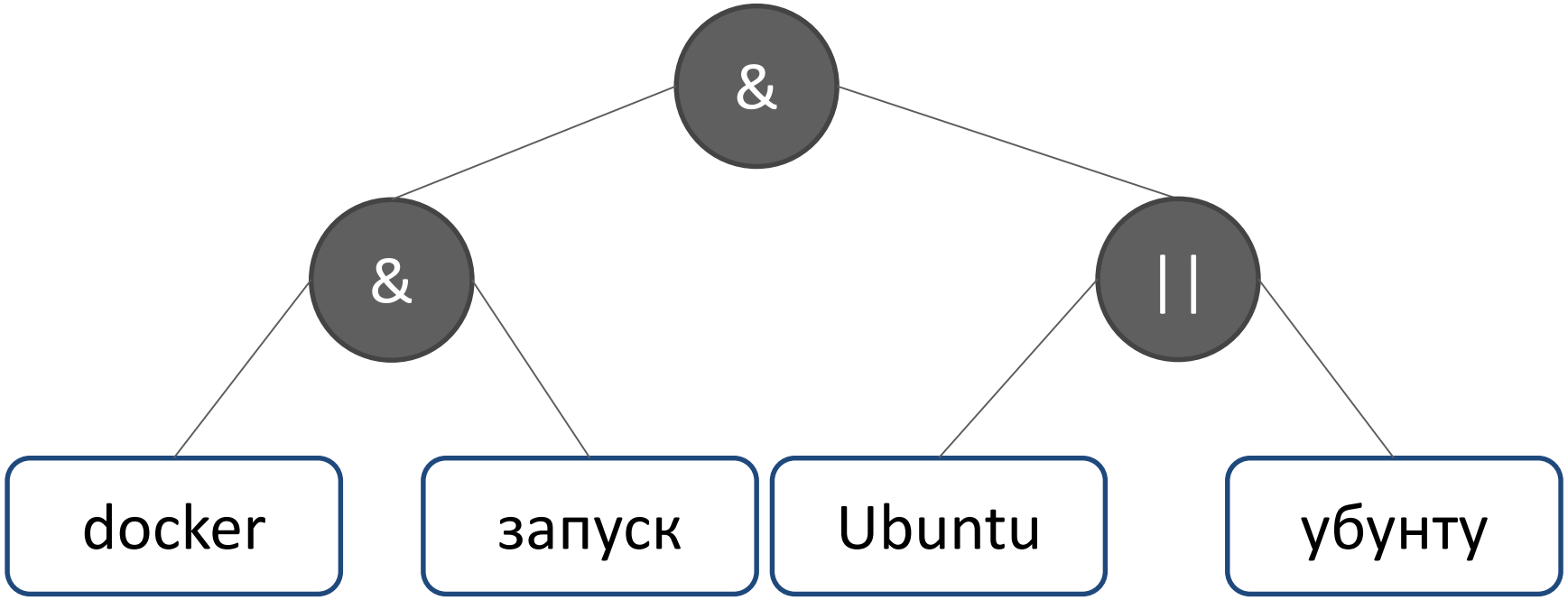
Запрос: docker запуск Ubuntu

docker & запуск & Ubuntu



Исполнение запроса

Добавим синонимы:
docker & запуск & (Ubuntu || убунту)



Исполнение запроса

Подход «в лоб»:

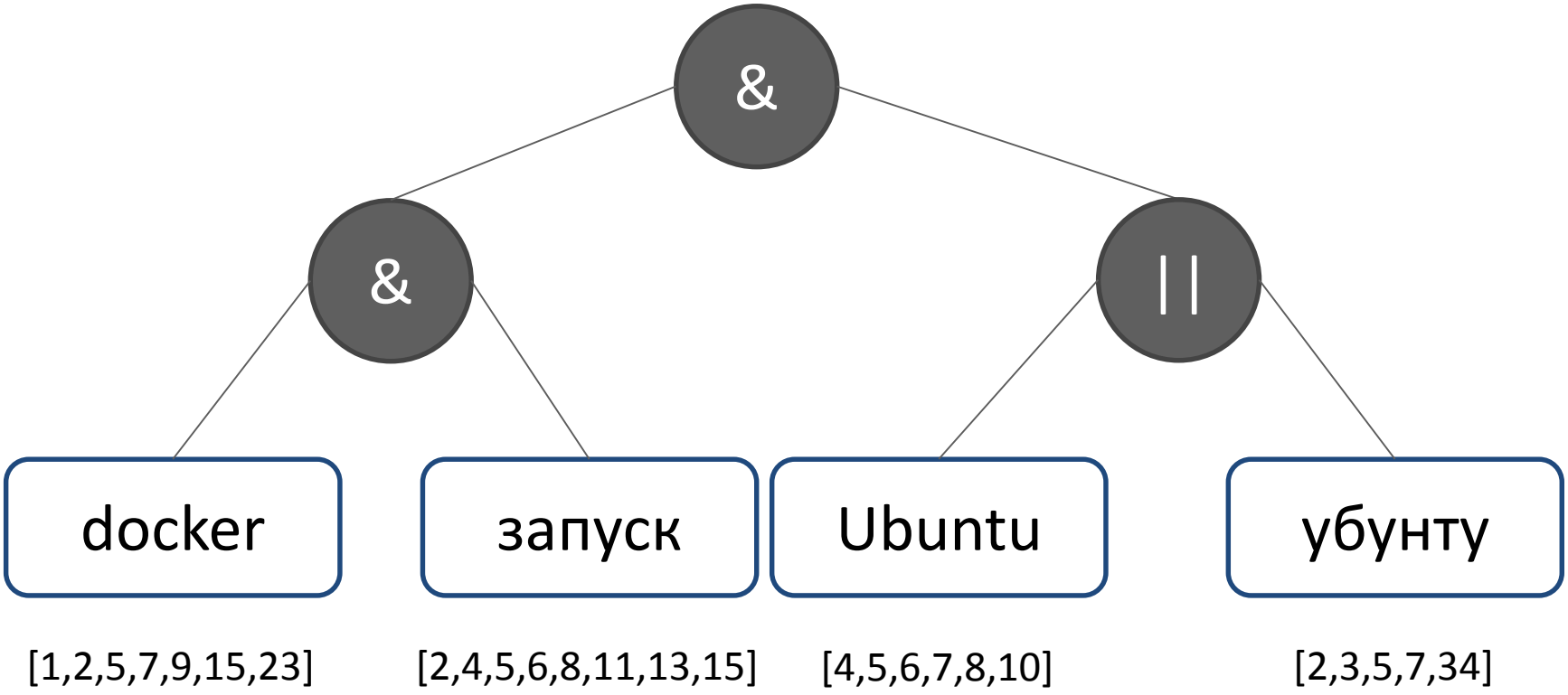
- находим множества документов для всех листьев (термы)
- идём вверх по дереву до корня

Исполнение запроса

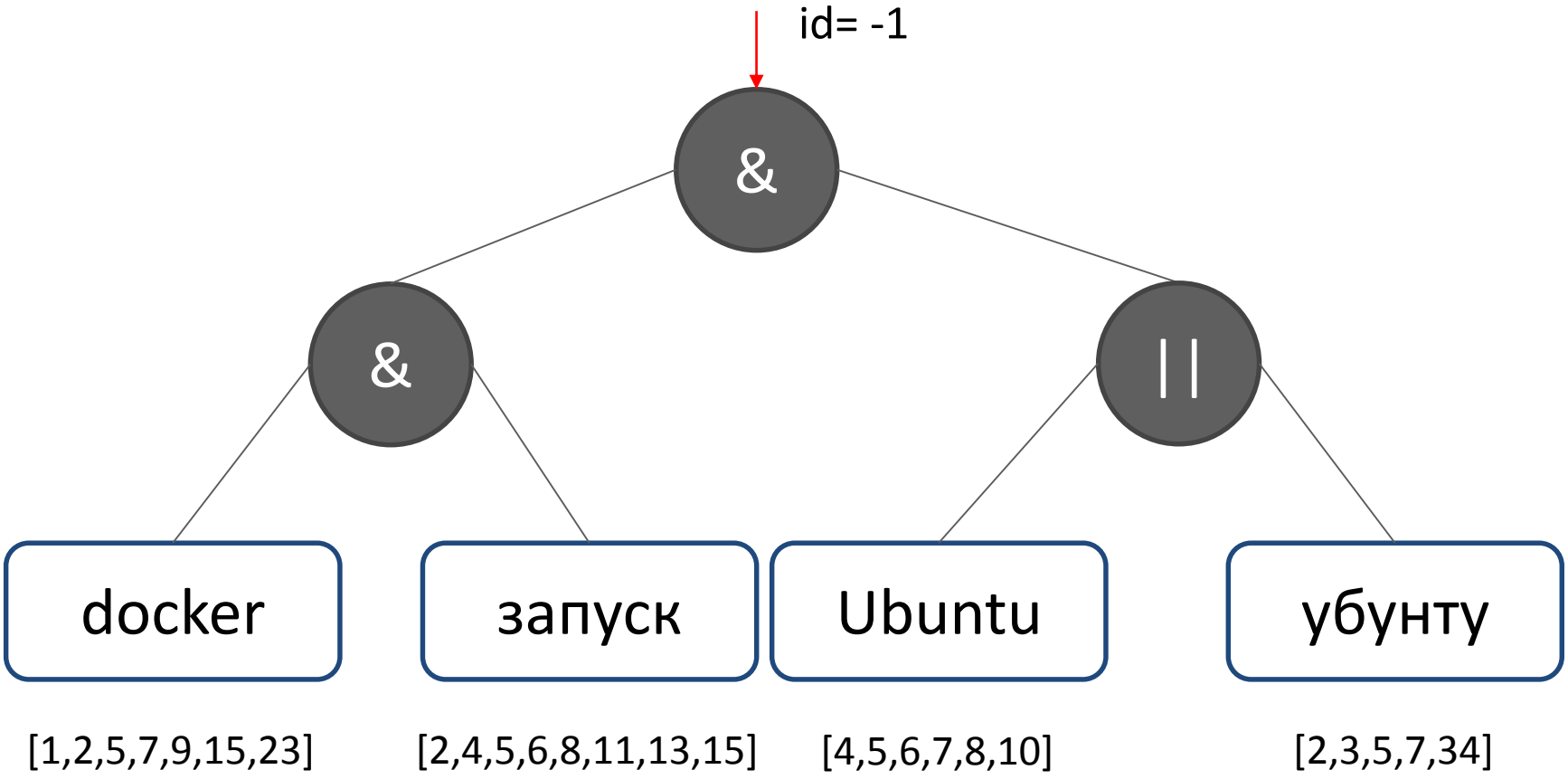
Подход «потоковый»:

- исполняем дерево пошагово
  - стартовый docID = -1
  - current_docID += 1
  - исполняем узлы: текущий docID
    - дизъюнкция: минимум
    - конъюнкция: не менее максимума

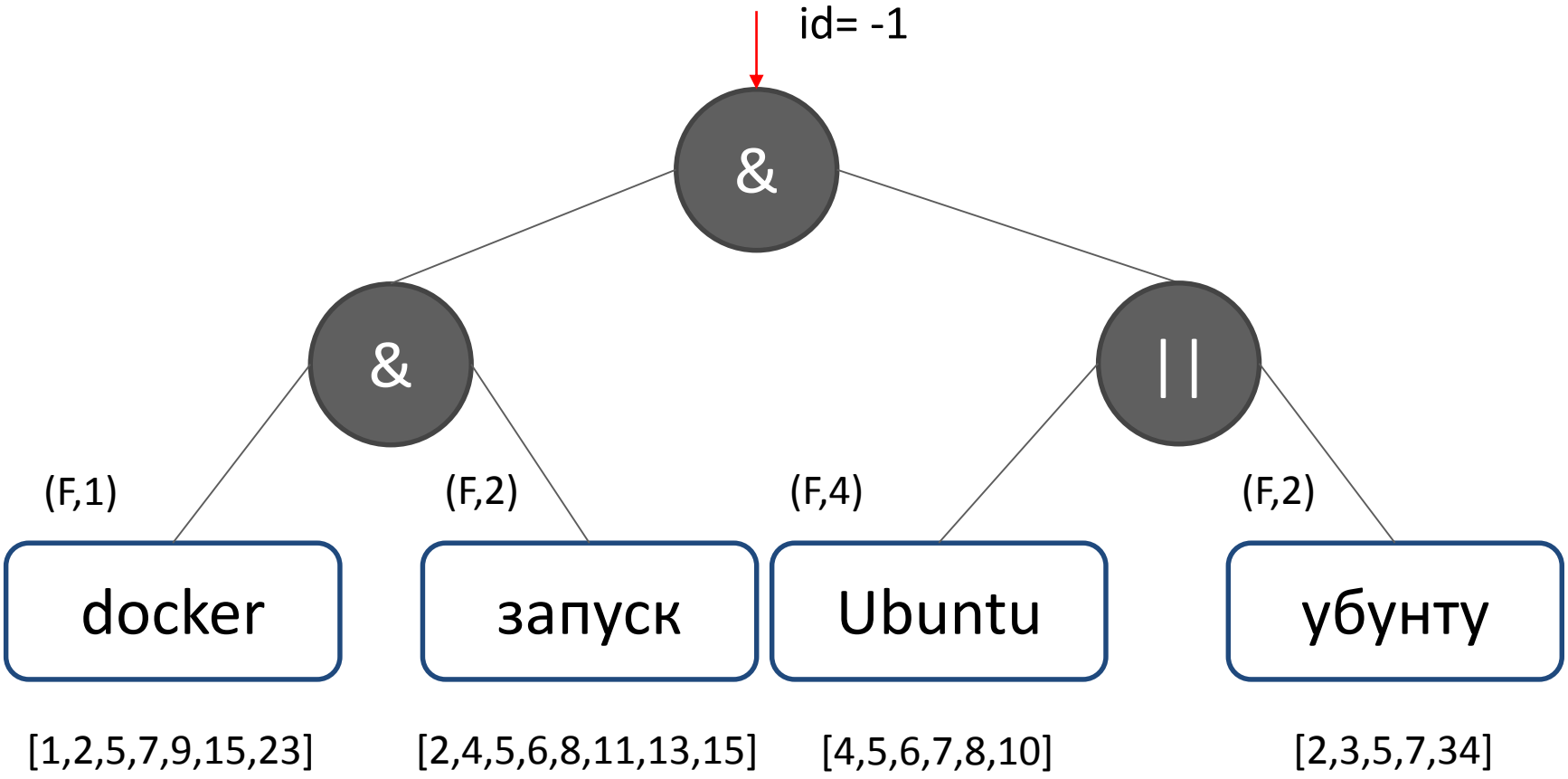
Исполнение запроса



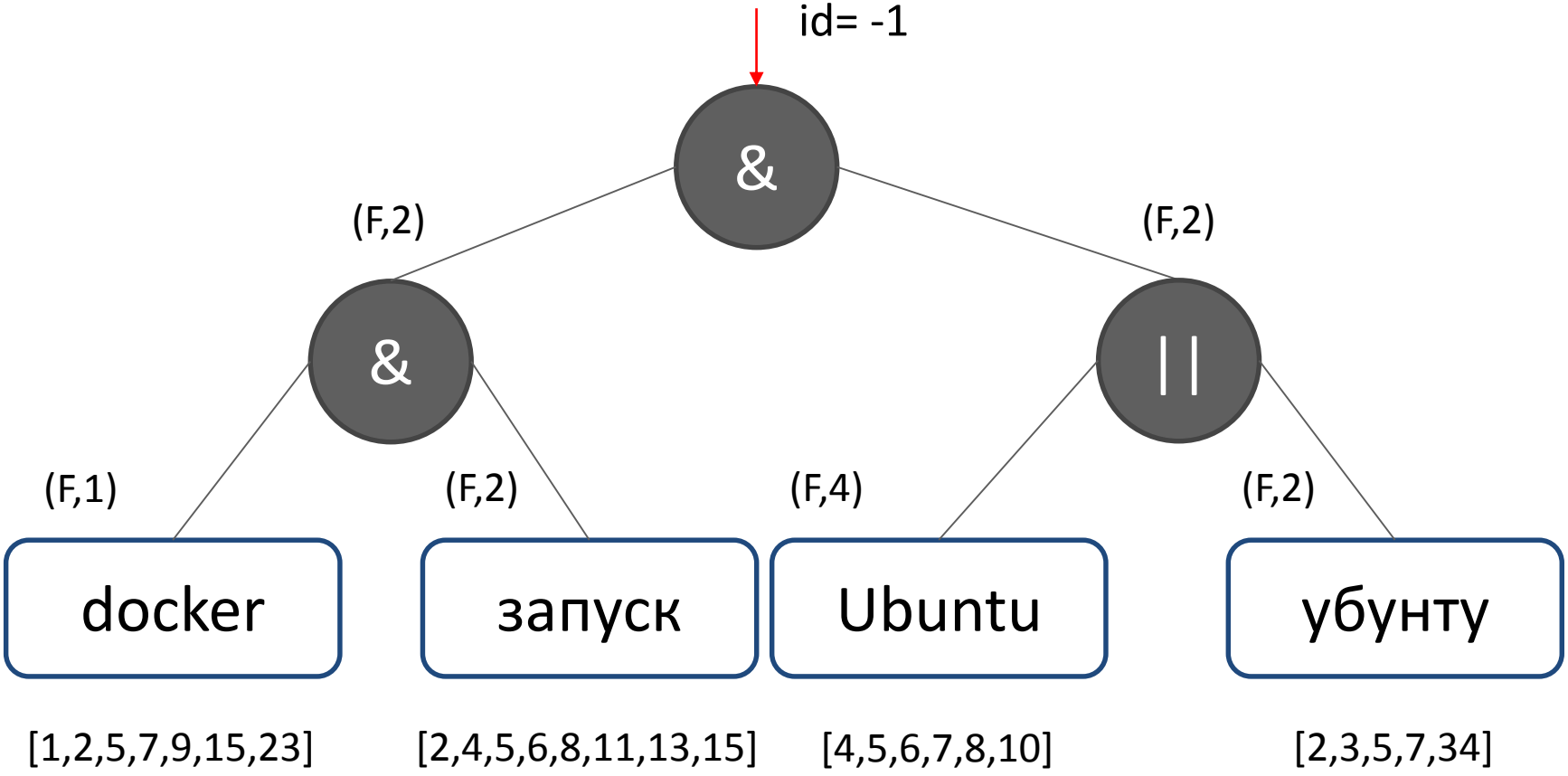
Исполнение запроса



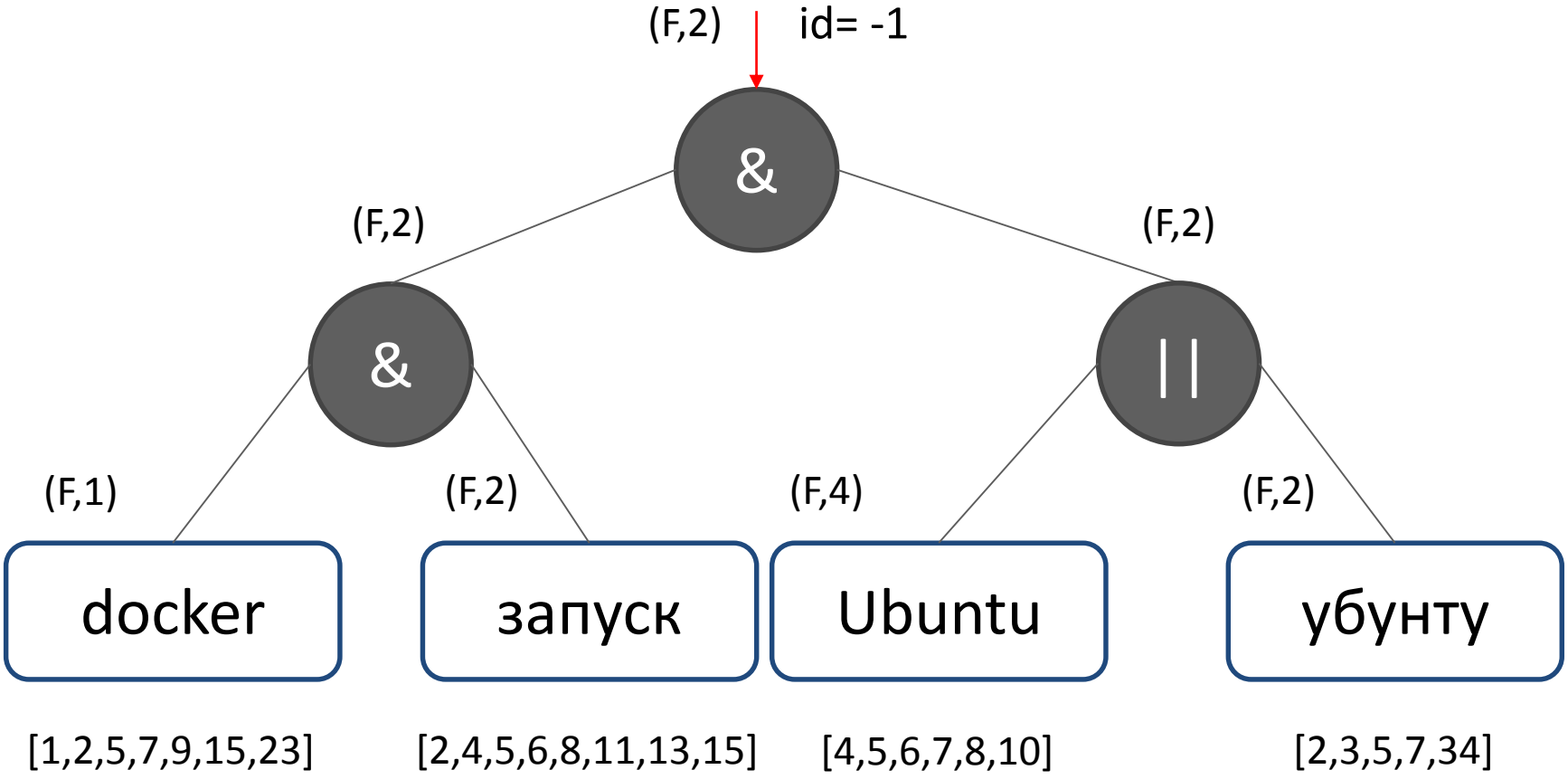
Исполнение запроса



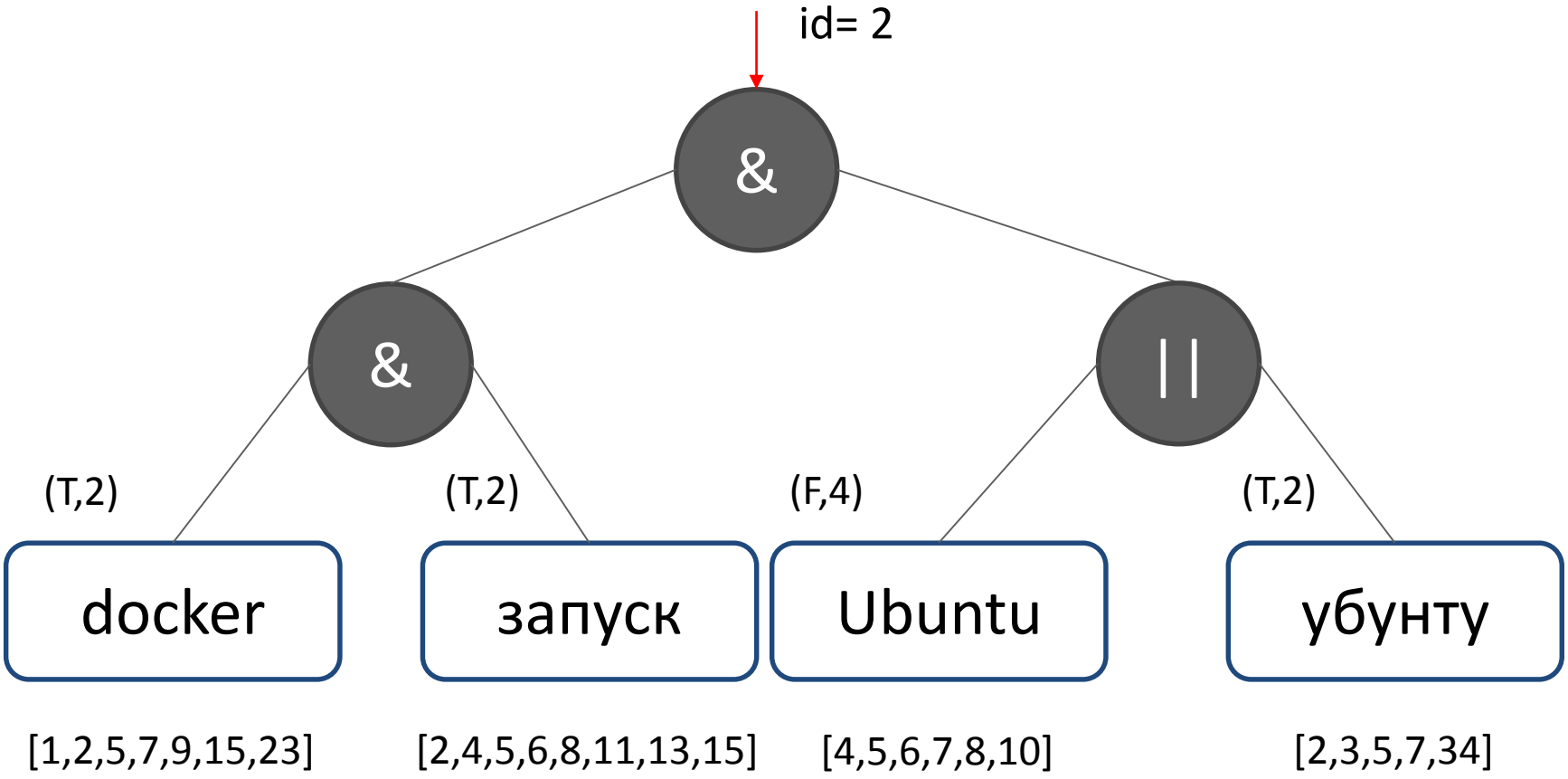
Исполнение запроса



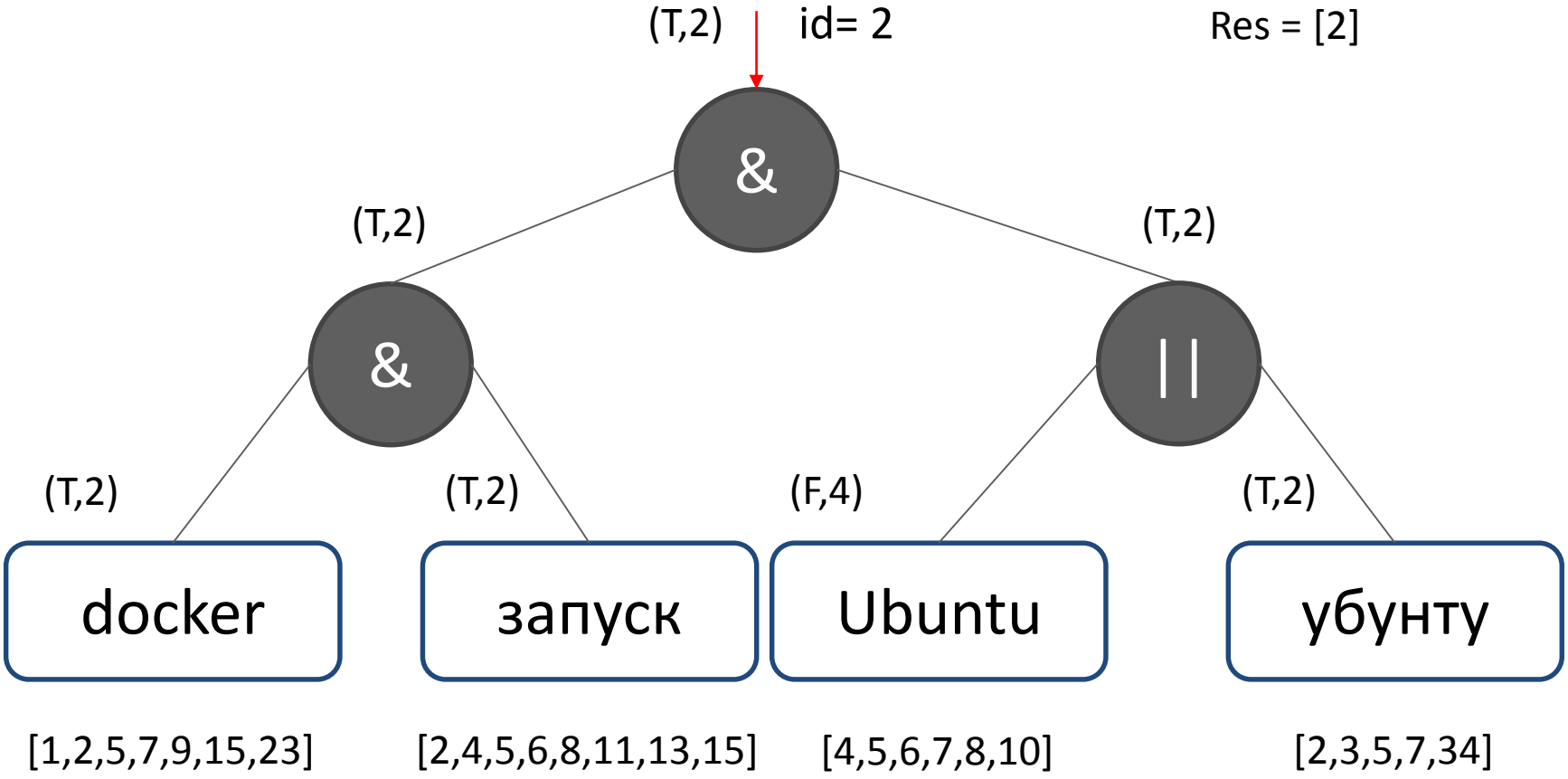
Исполнение запроса



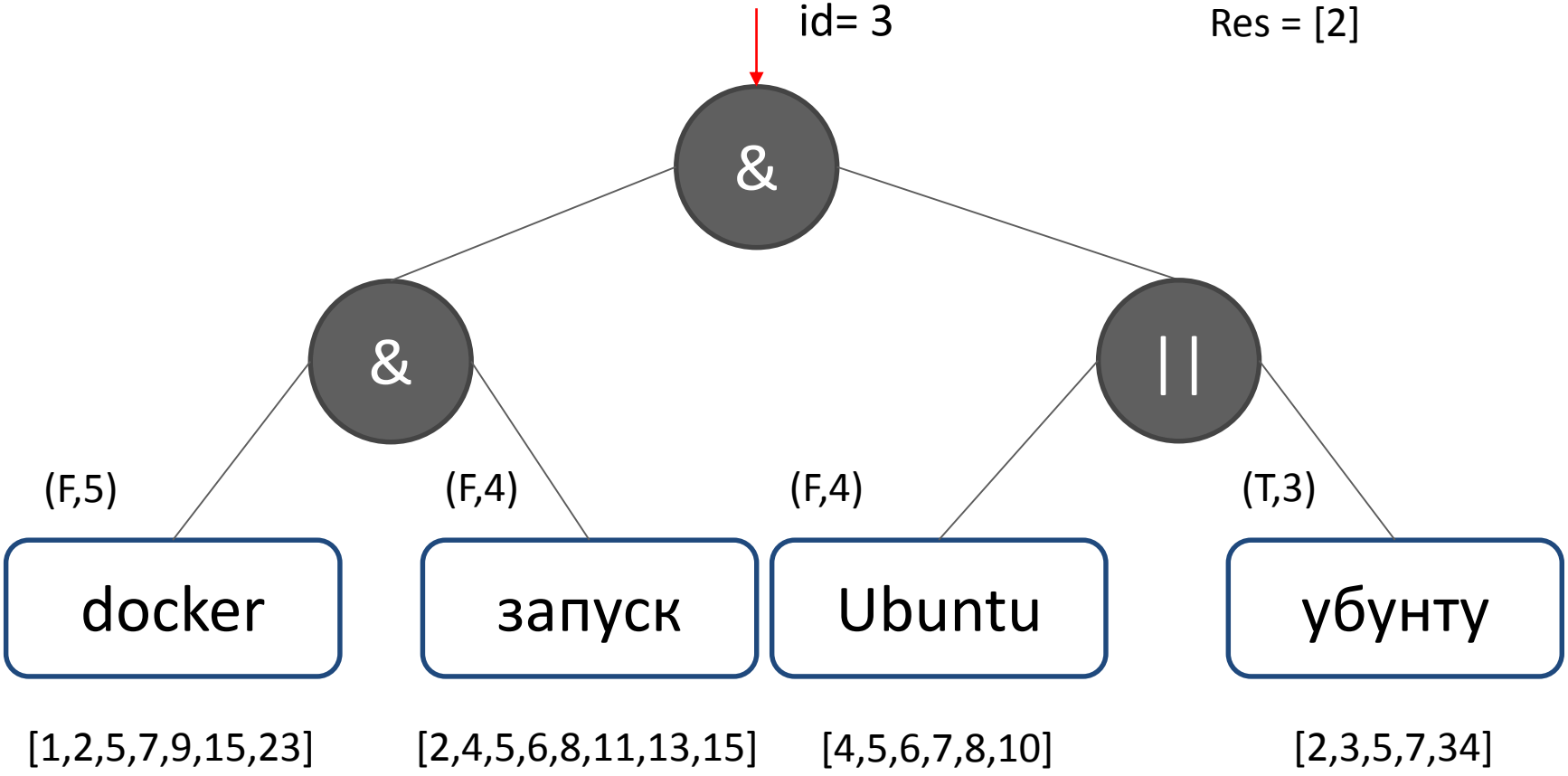
Исполнение запроса



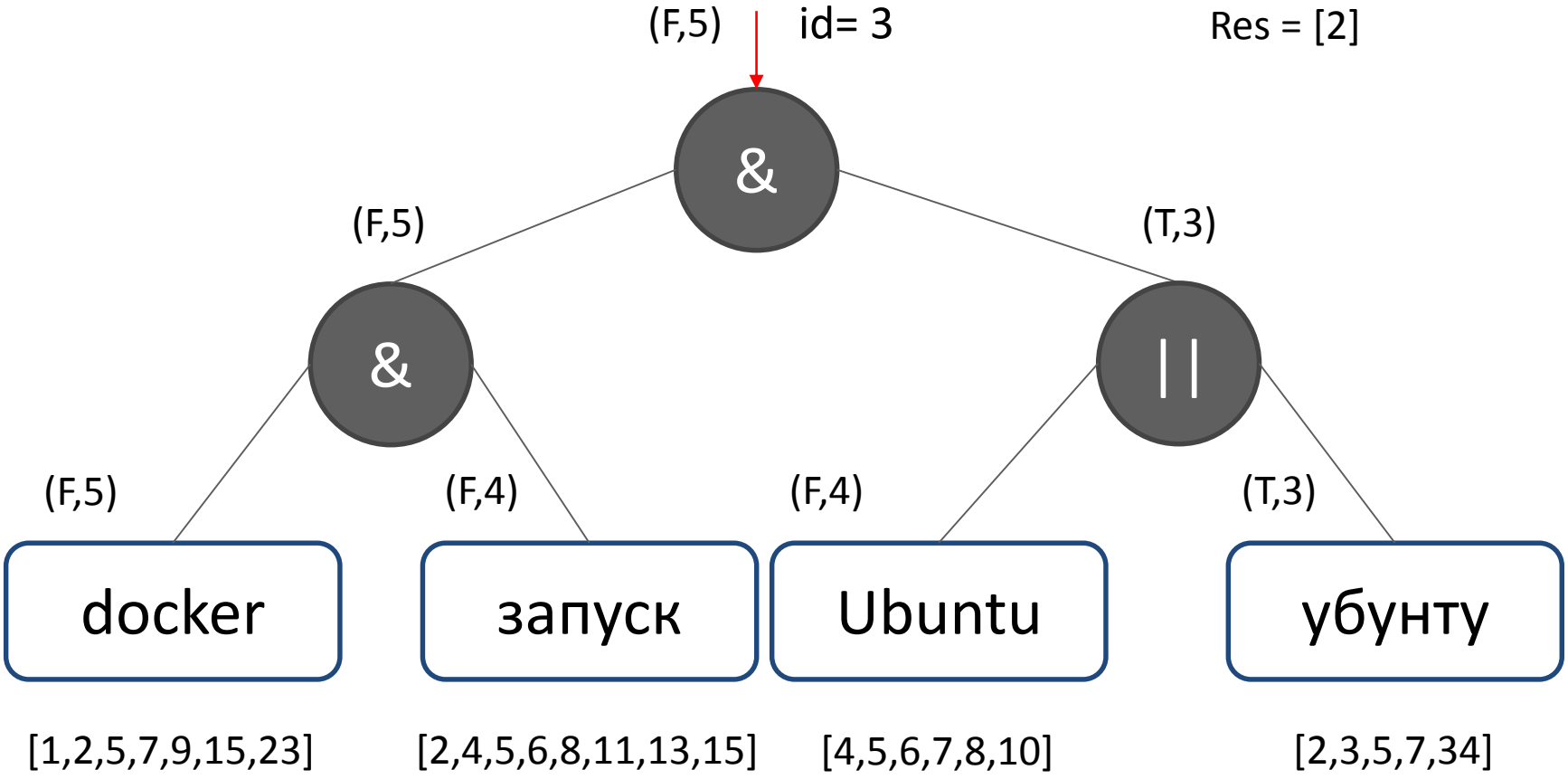
Исполнение запроса



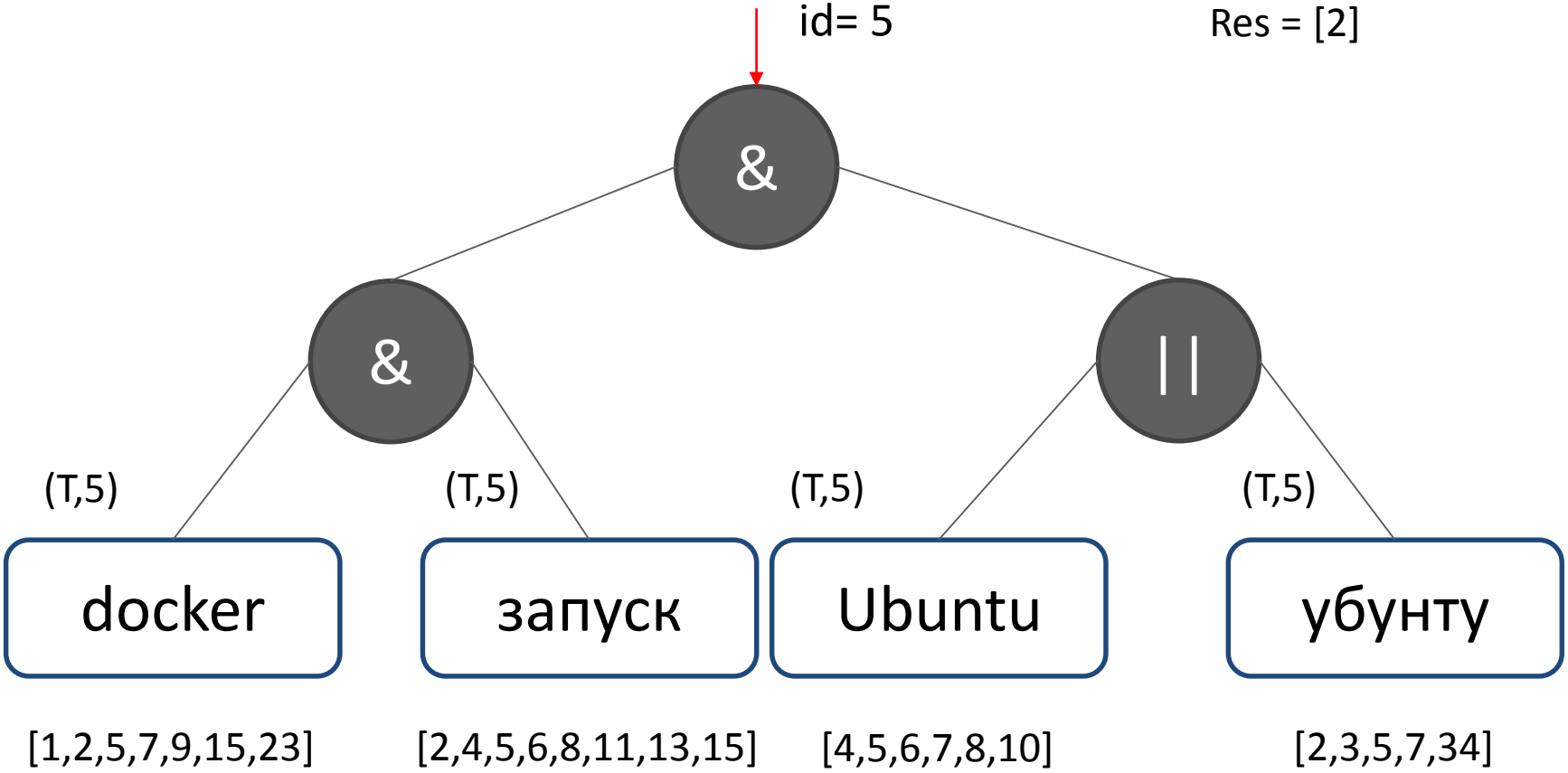
Исполнение запроса



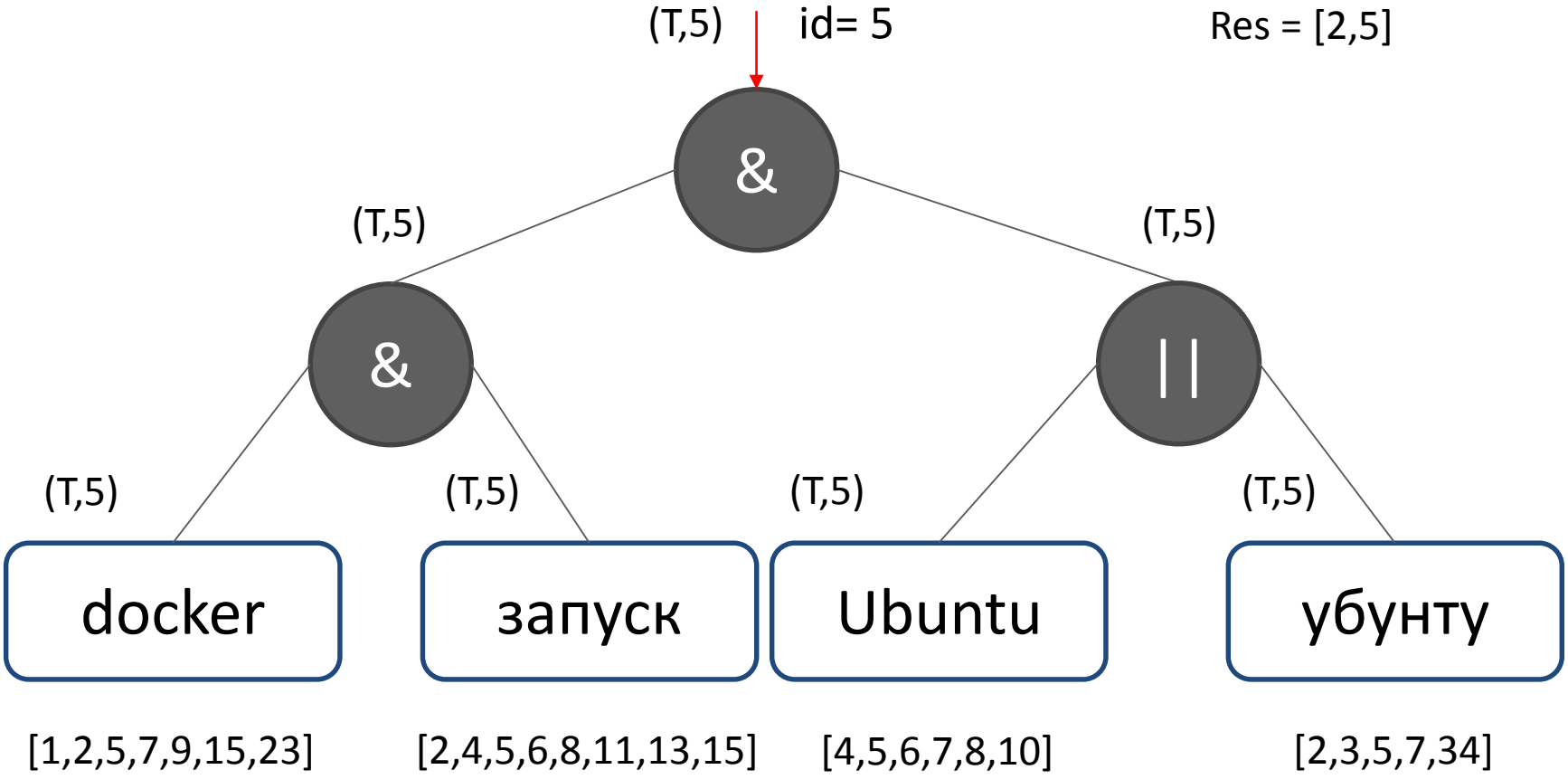
Исполнение запроса



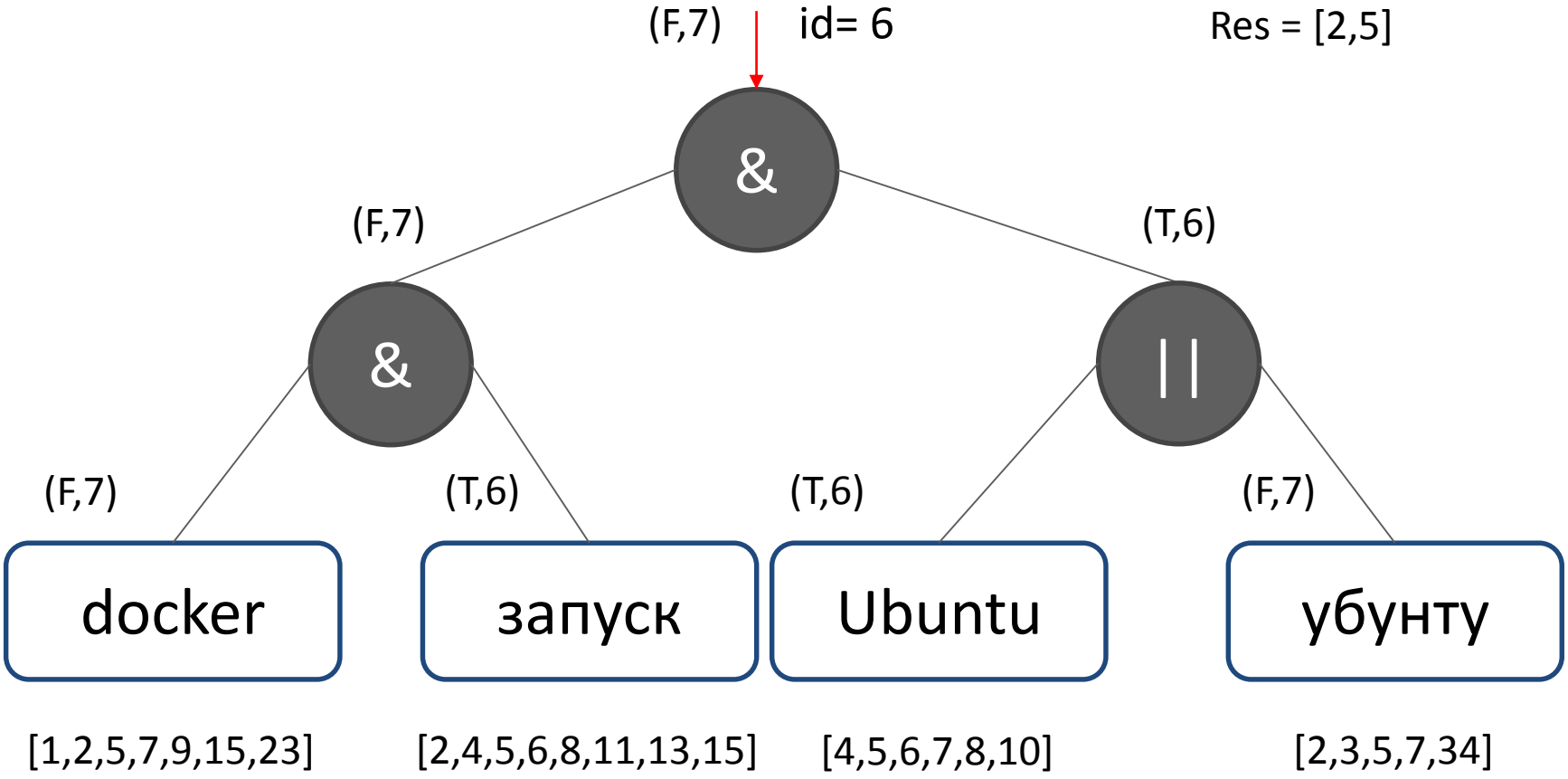
Исполнение запроса



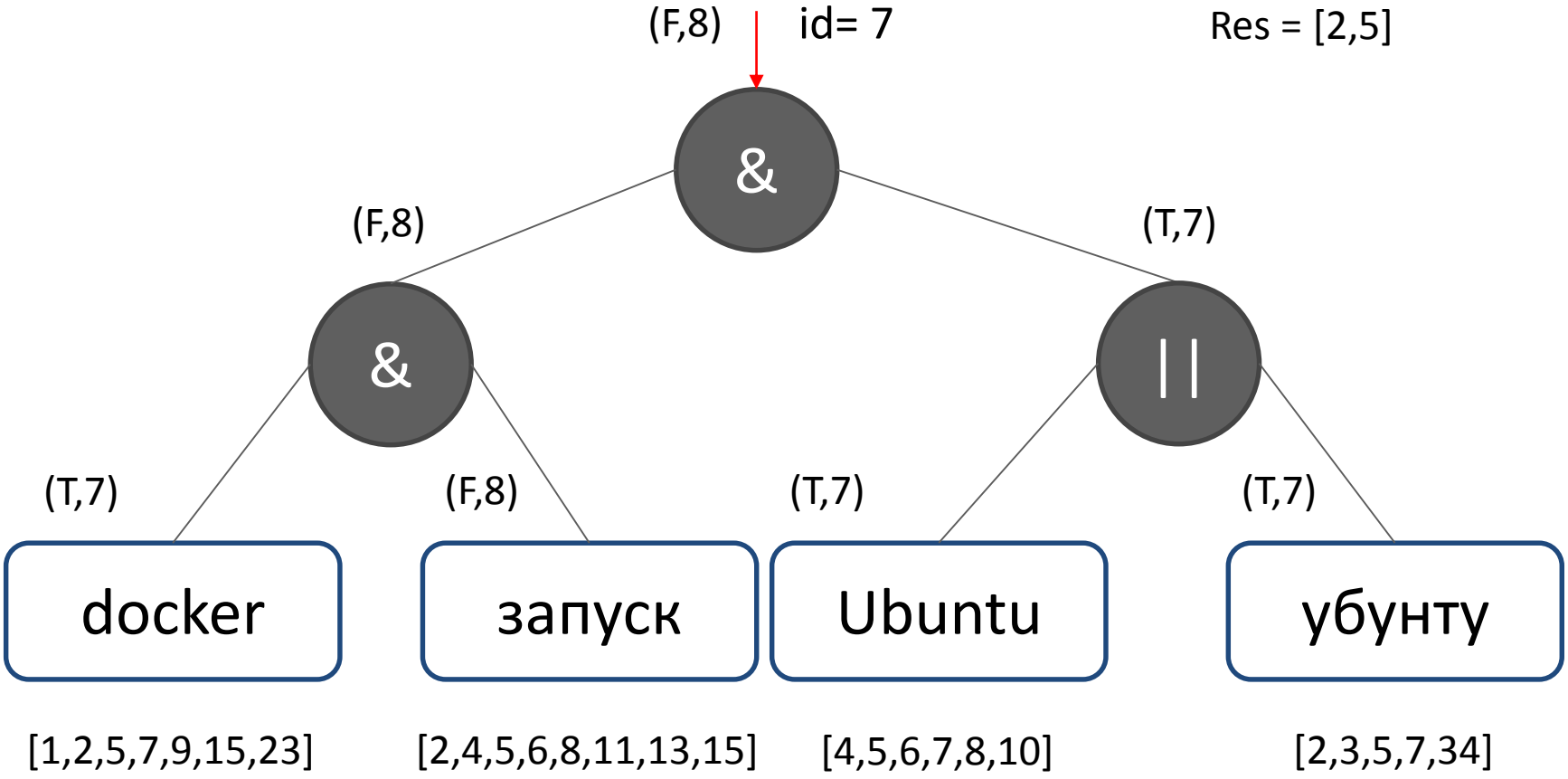
Исполнение запроса



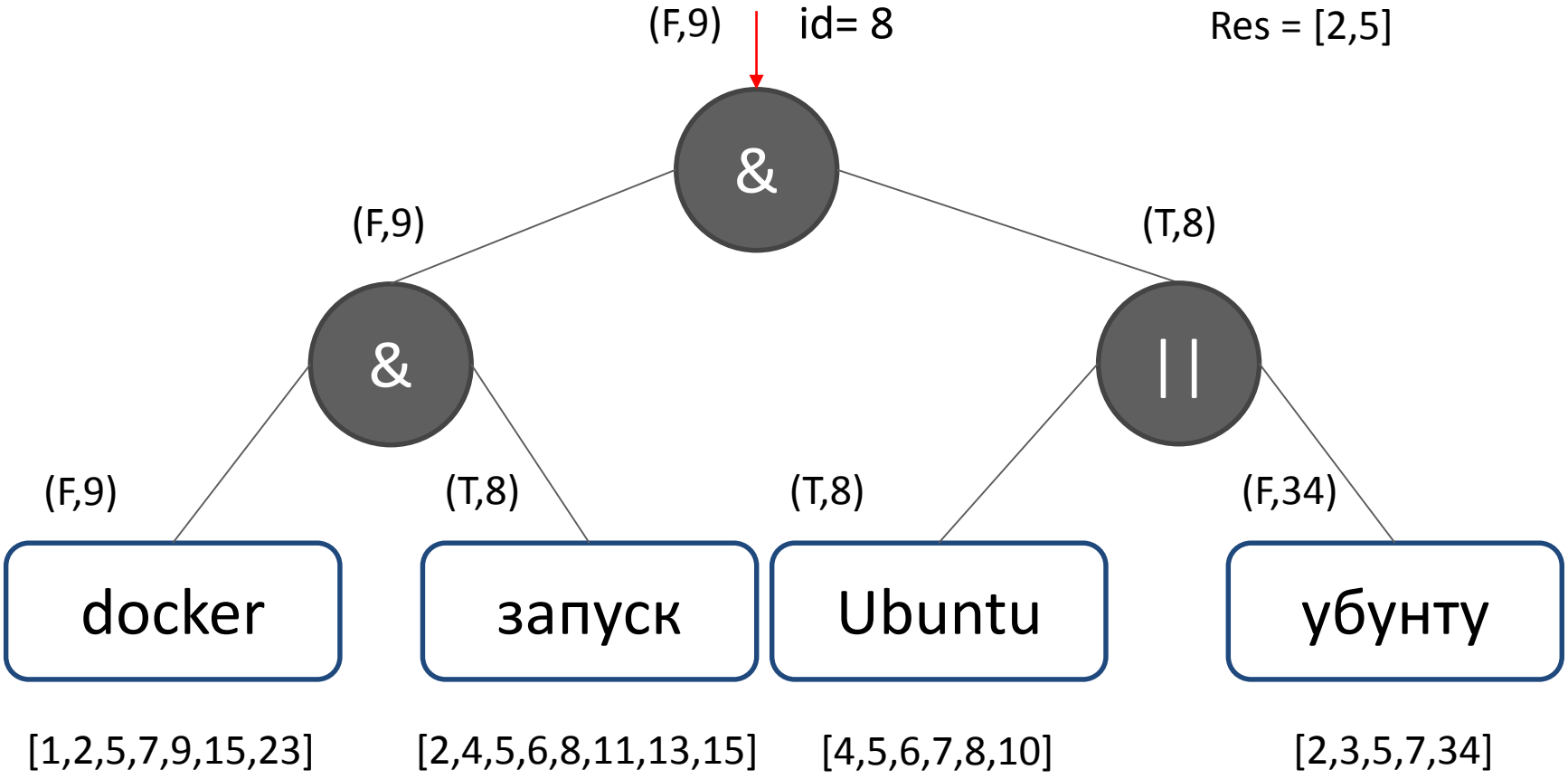
Исполнение запроса



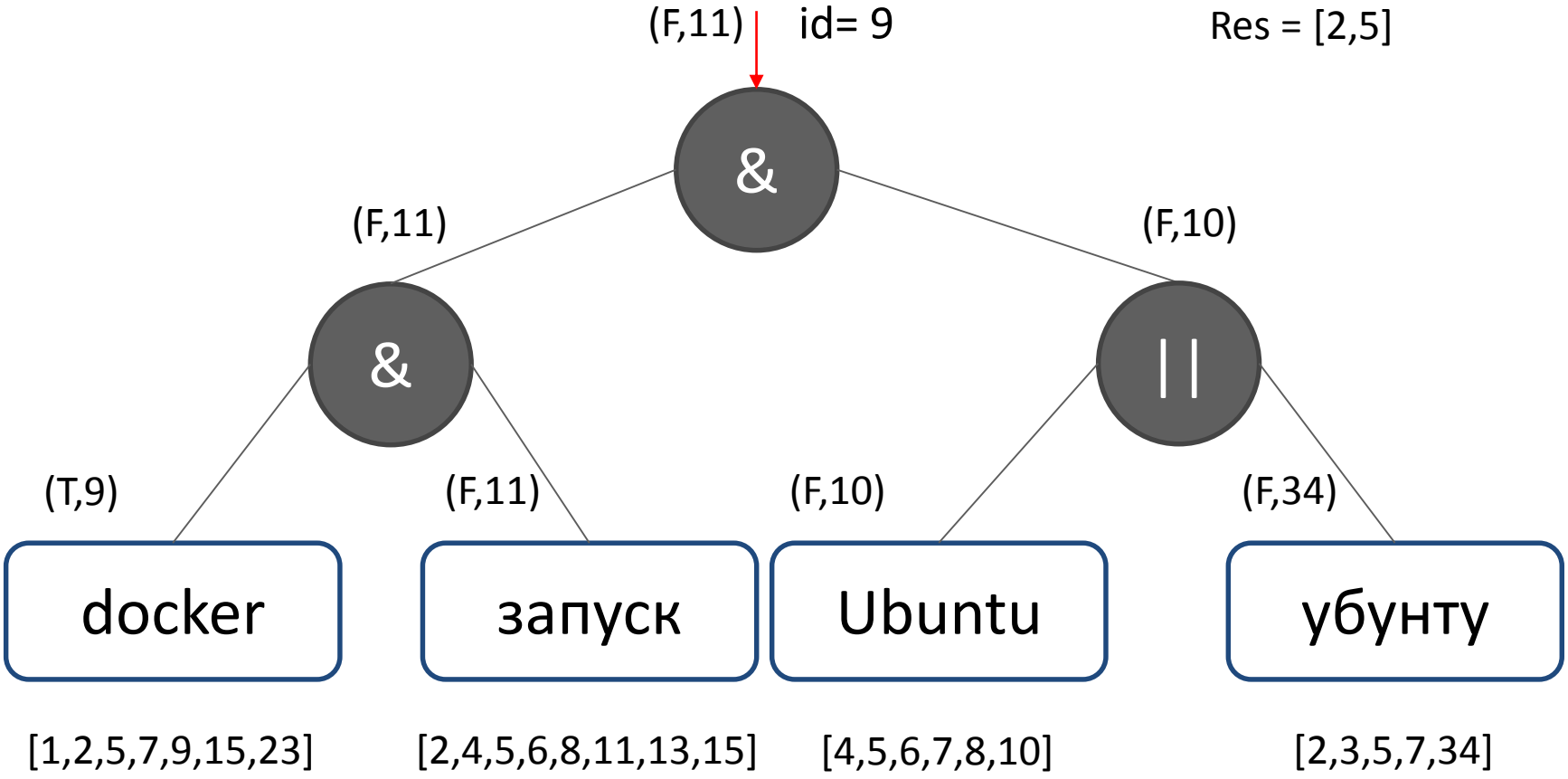
Исполнение запроса



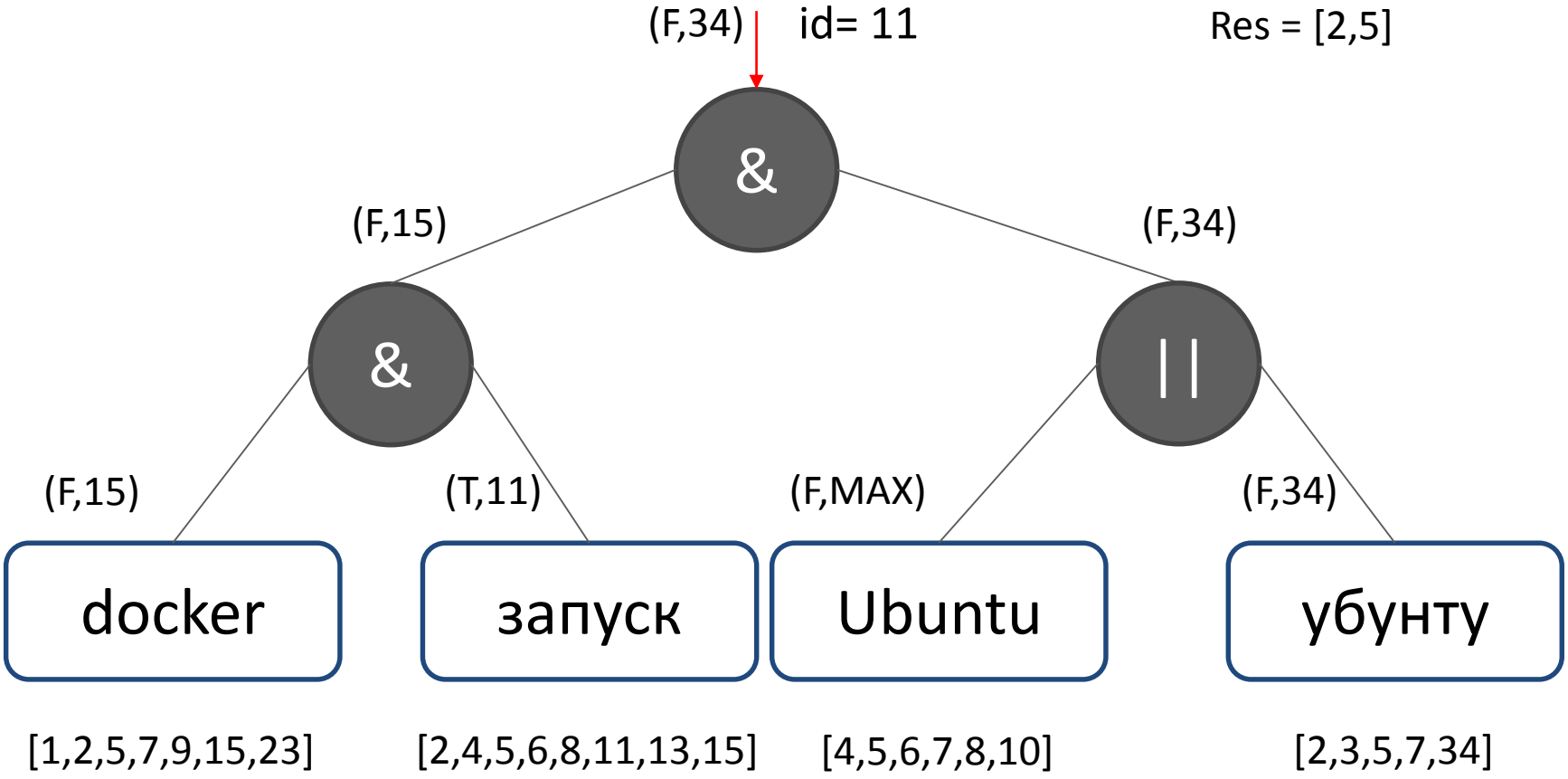
Исполнение запроса



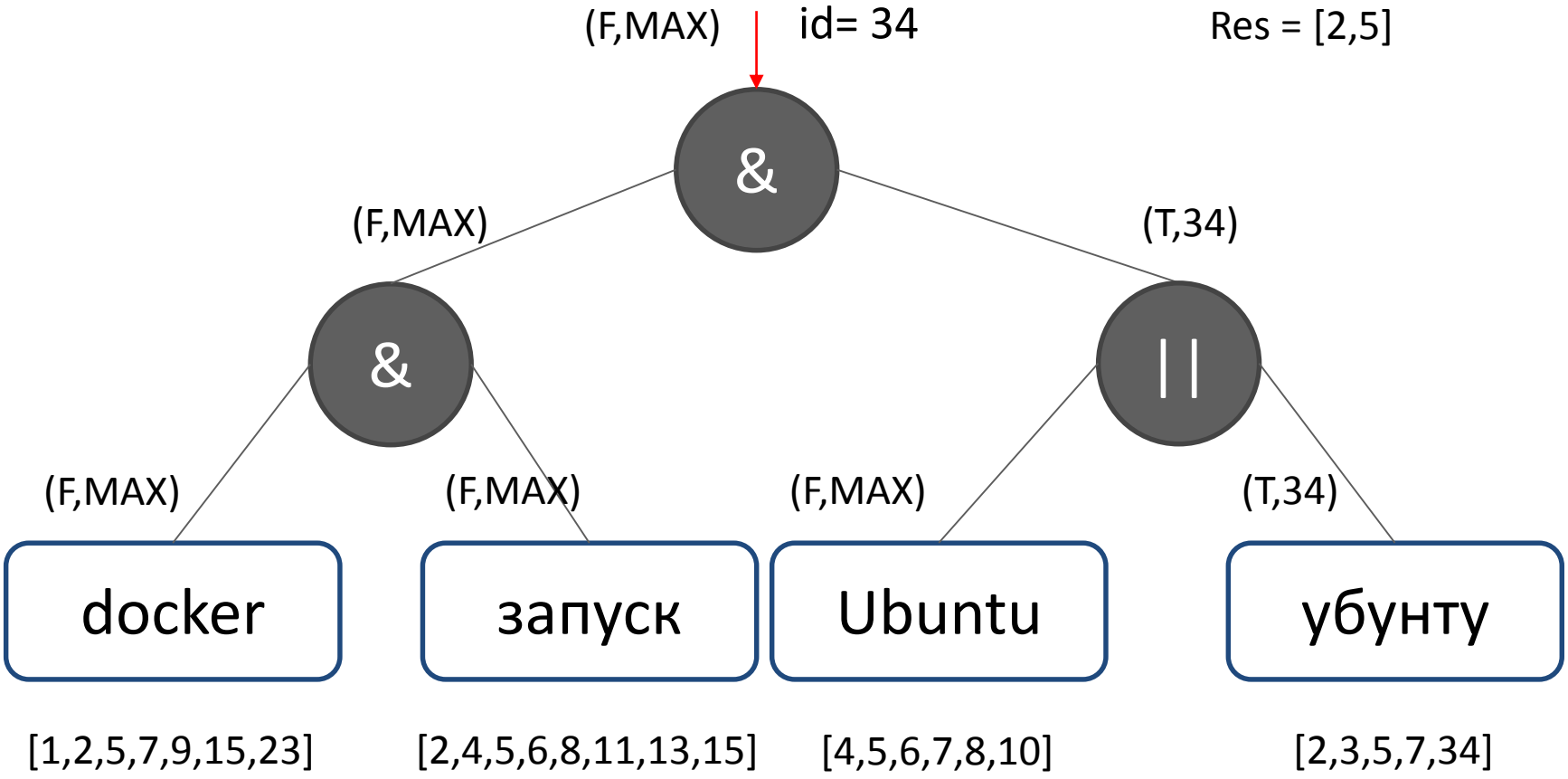
Исполнение запроса



Исполнение запроса



Исполнение запроса



Разбор запроса

docker & запуск & (Ubuntu || убунту)

1. операции и термы
2. определяем приоритет и порядок операций
3. дерево строится от меньшего приоритета (корень, исполняется последним)

Общий workflow индексации

1. индексация входных данных (index.sh)

Наиболее затратна по времени (много данных + можем себе позволить время)
Можем индексировать по частям
Содержит ссылки на блоки
сохраняем соответствие url <-> docID (в выдаче нужны урлы)

Общий workflow индексации

1. индексация входных данных (index.sh)
2. оптимизация индекса

Общий workflow индексации

1. индексация входных данных (index.sh)
2. оптимизация индекса
3. построение словаря (make_dict.sh)

Общий workflow индексации

1. индексация входных данных (index.sh)
2. оптимизация индекса
3. построение словаря (make_dict.sh)
4. поиск (search.sh):
    1. строим Q-Tree
    2. ставим в соответствие блоки
    3. ищем конкретные docID → преобразуем в URL
    4. …
    5. PROFIT!