



ТЕХНОСФЕРА

Индексация & булев поиск

Владимир Гулин

Этапы ранжирования поиска



План лекции:

1. Обход интернета
2. Базовая задача матчинга
3. Булев поиск
4. Обратный индекс
5. Поиск по обратному индексу, пересечение блоков

Алгоритм обхода

1. "Точка входа" - seed-урлы
2. Скачали
3. Распарсили, извлекли урлы, отправили урлы в очередь на обкачку
4. goto #2

Seed-урлы

КАТАЛОГ@mail.ru®

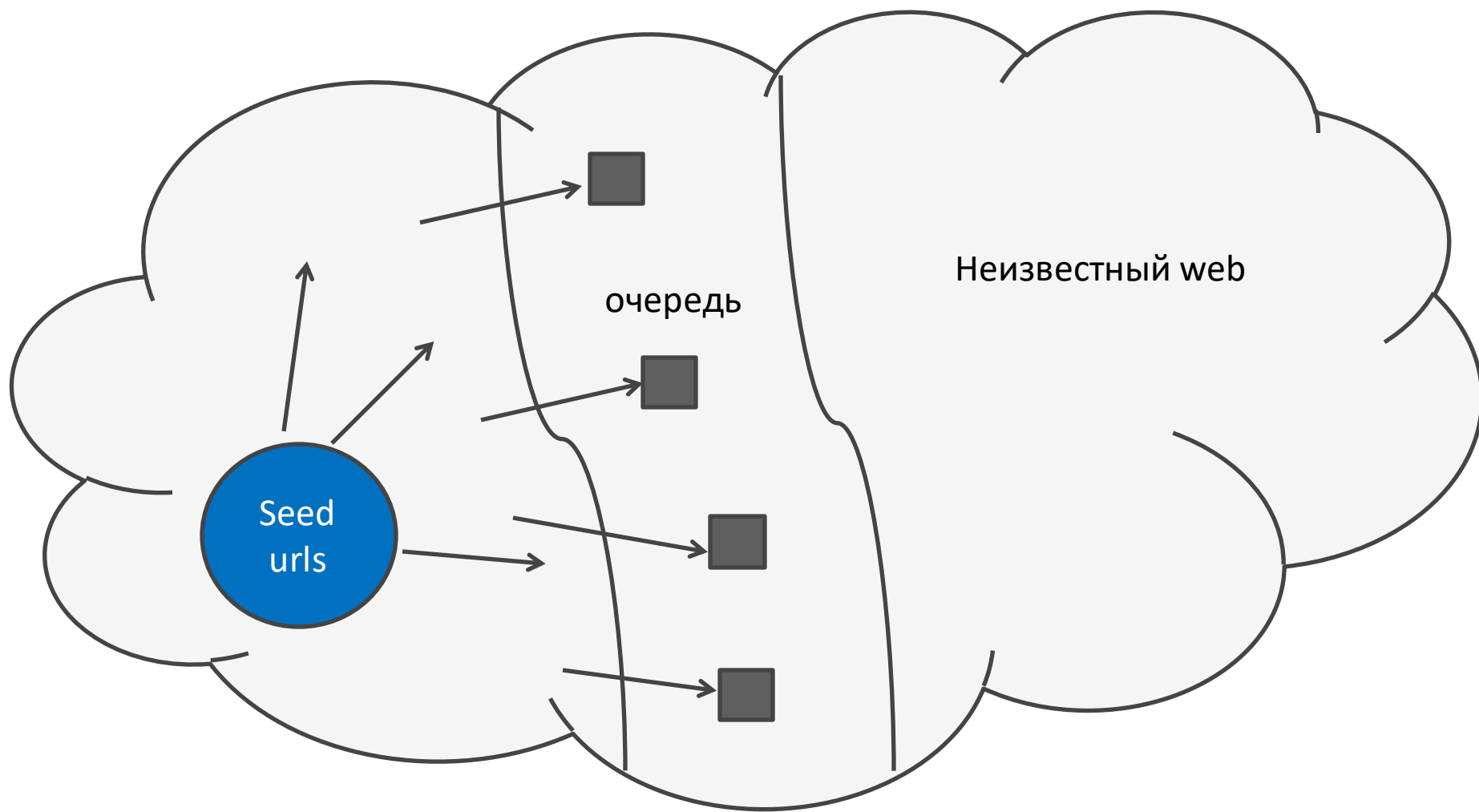
Яндекс
каталог

РЕЙТИНГ@mail.ru

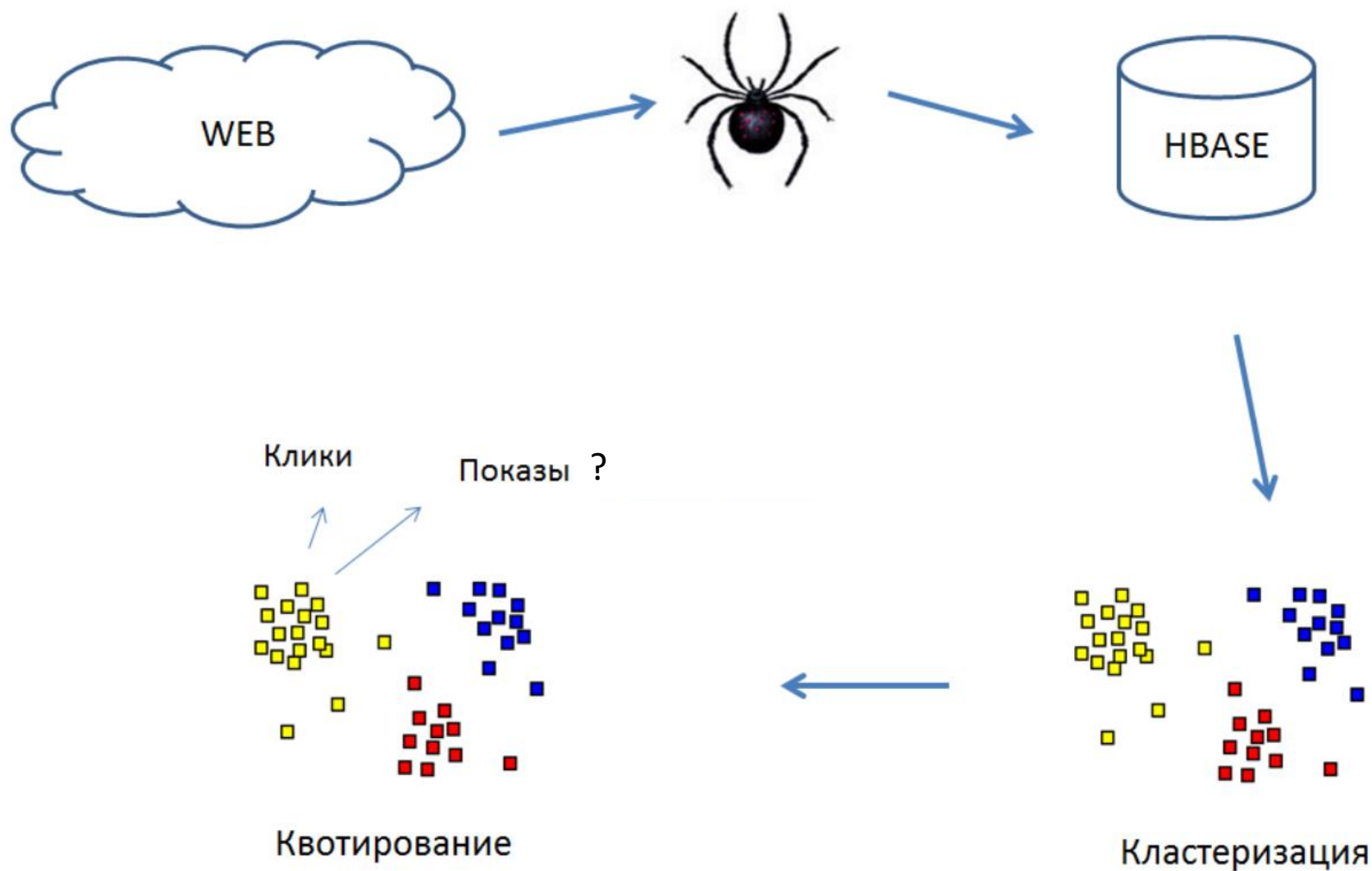


Википедия
Свободная энциклопедия

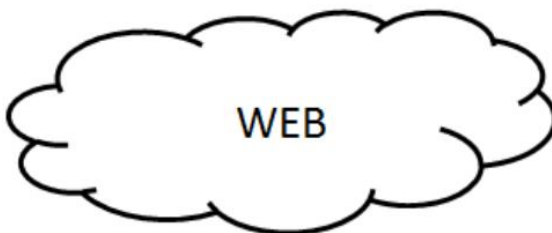
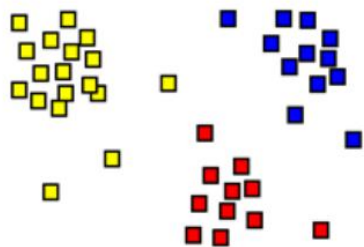
Выкачка



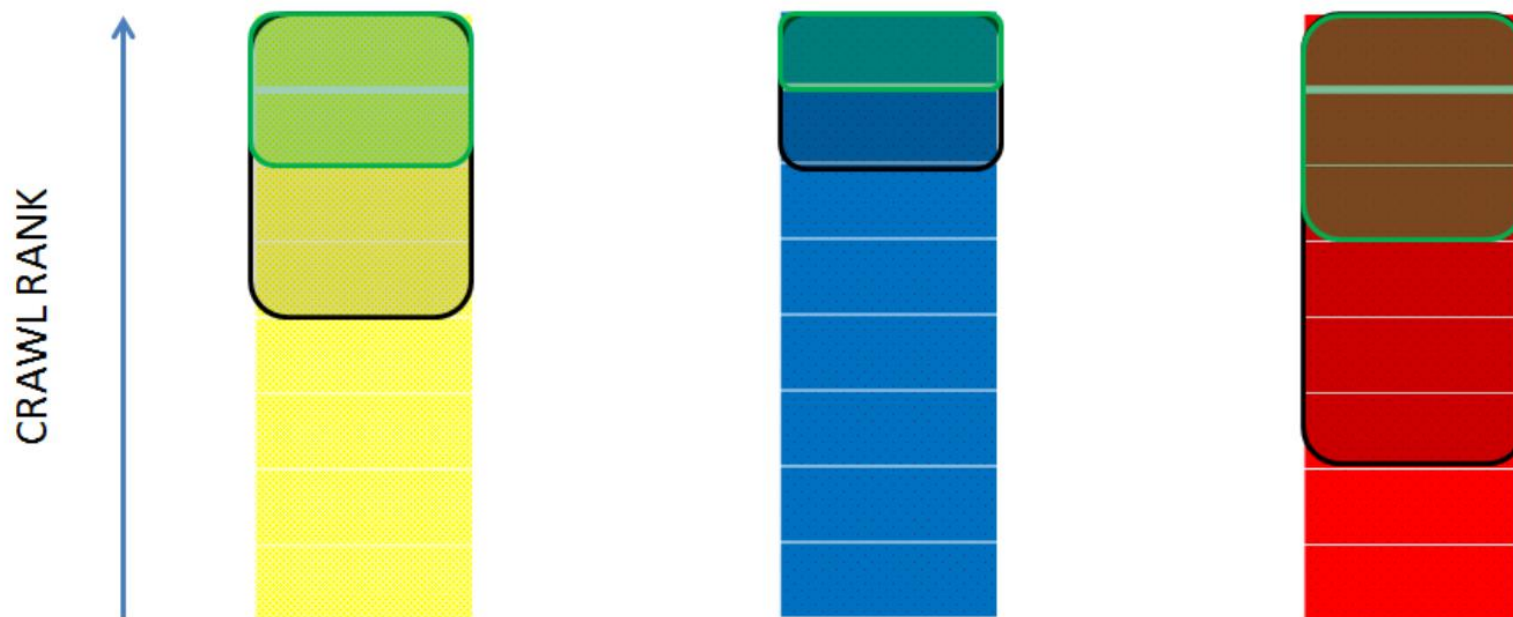
Принцип попадания урлов в индекс



Приоритезация



Квоты под индекс



Разметчик

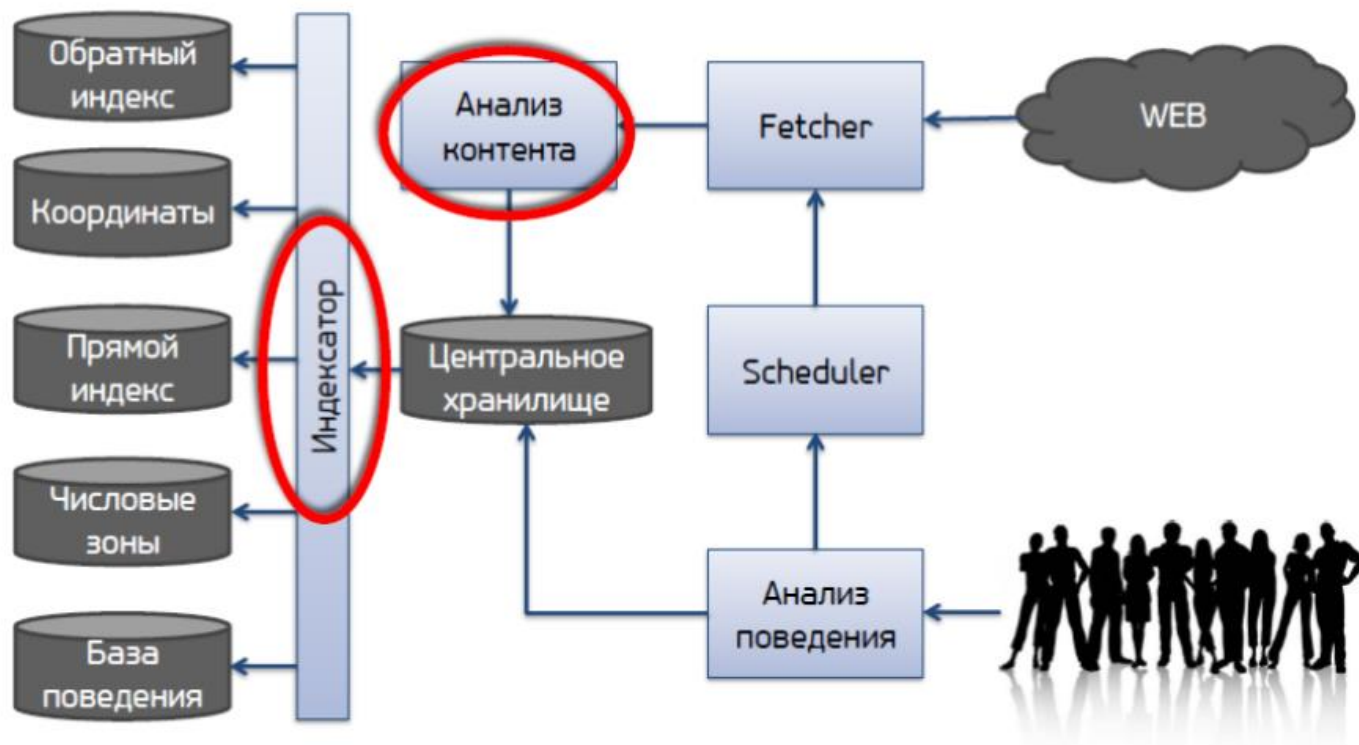
Цель

Хотим максимизировать число "хороших" документов в индексе

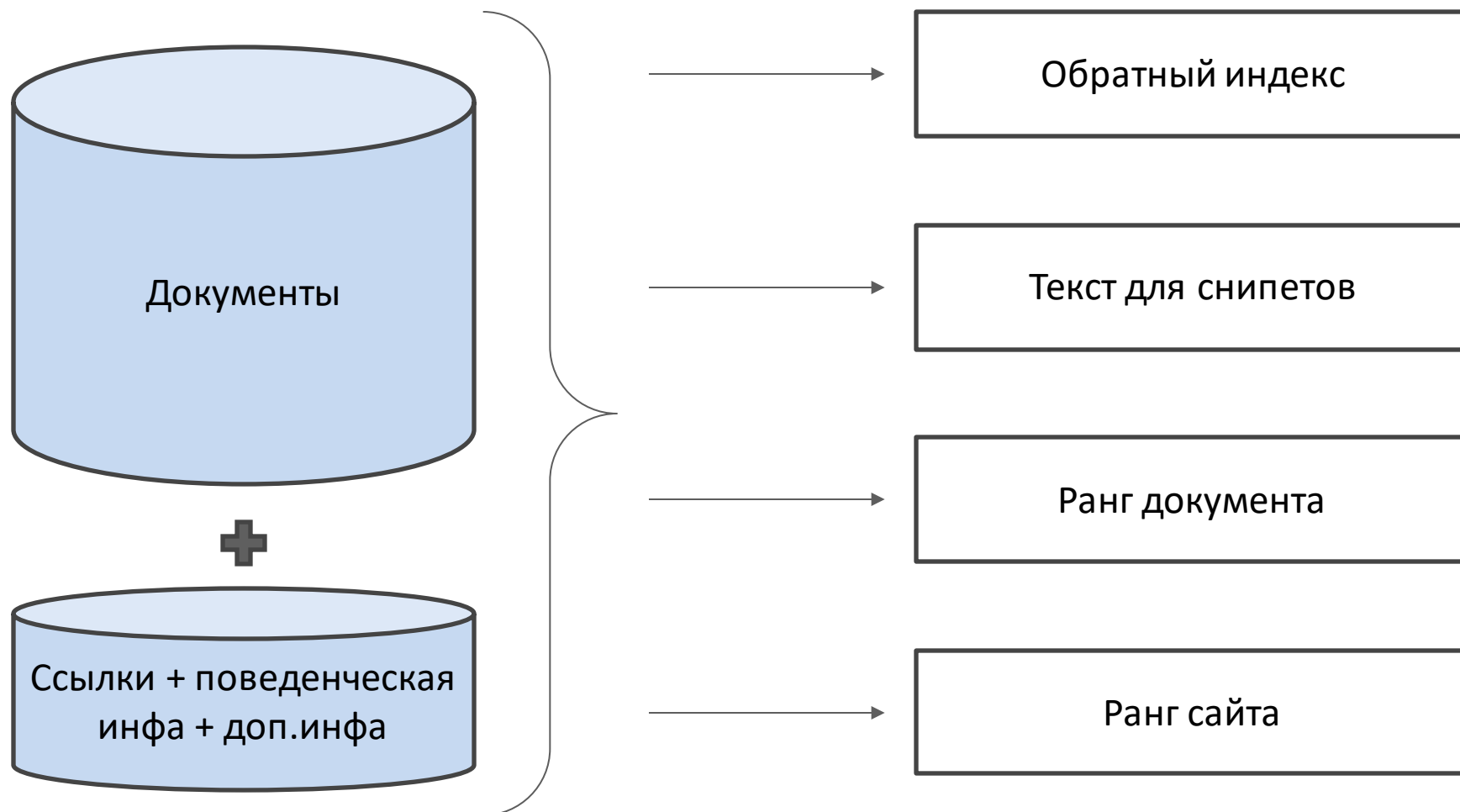
Click Prediction

$$p(click|url) = \sum_{q \in Q} p(q)p(click|q, url)$$

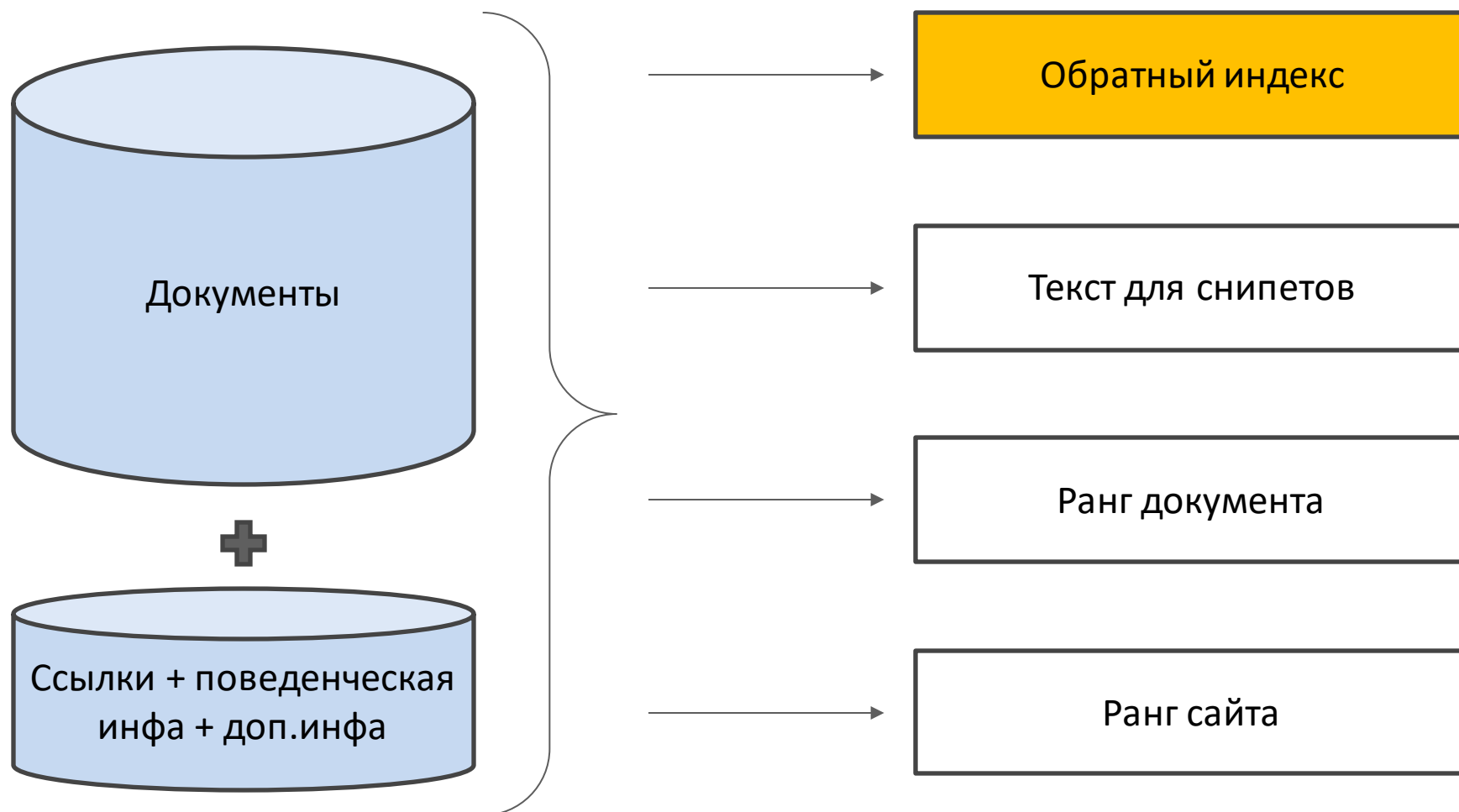
Этапы предобработки документа



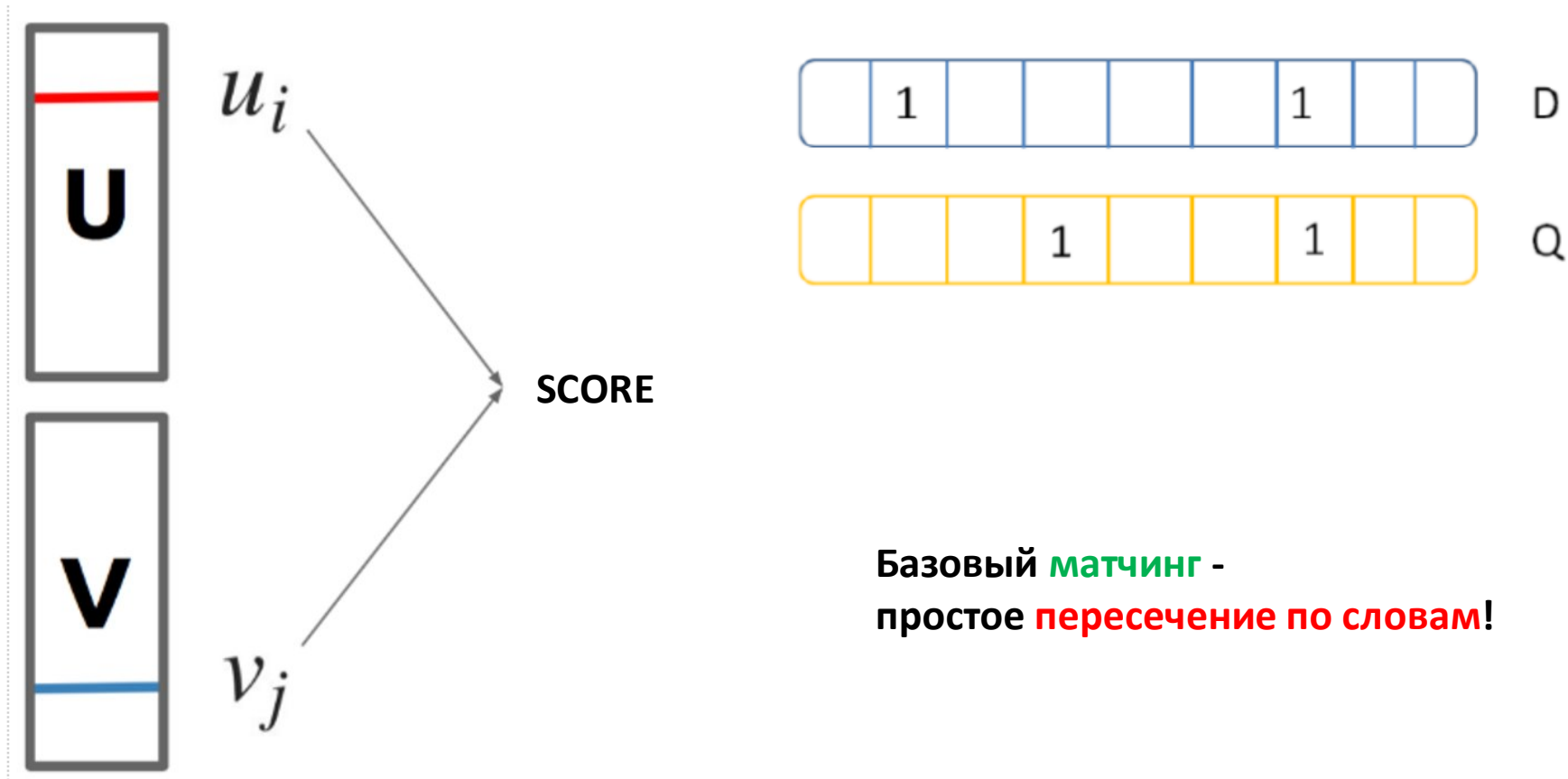
Что мы получаем из документов



Что мы получаем из документов



Базовая задача матчинга



Базовый **матчинг** -
простое **пересечение по словам!**

От документа к индексу

1. Документ \rightarrow текст

От документа к индексу

1. Документ → текст
2. Текст → слова

От документа к индексу

1. Документ → текст
2. Текст → слова
3. Слова → леммы (зачем?)

От документа к индексу

1. Документ → текст
2. Текст → слова
3. Слова → леммы (зачем?)

Лемма – лингв. нормализованная, основная форма слова, вместе с информацией о построении других форм

От документа к индексу

1. Документ → текст
2. Текст → слова
3. Слова → леммы (зачем?)
4. Лемма → список документов, в которых встречается:
«posting list», «инвертированный список» и т.д.

От слов к действию числам

Документ <-> URL

От слов к действию числам

Документ <-> URL <-> docID

От слов к действию числам

Документ \leftrightarrow URL \leftrightarrow docID

Слово \leftrightarrow termID (например, hash)

Обратный индекс



Чем дополнить обратный индекс?

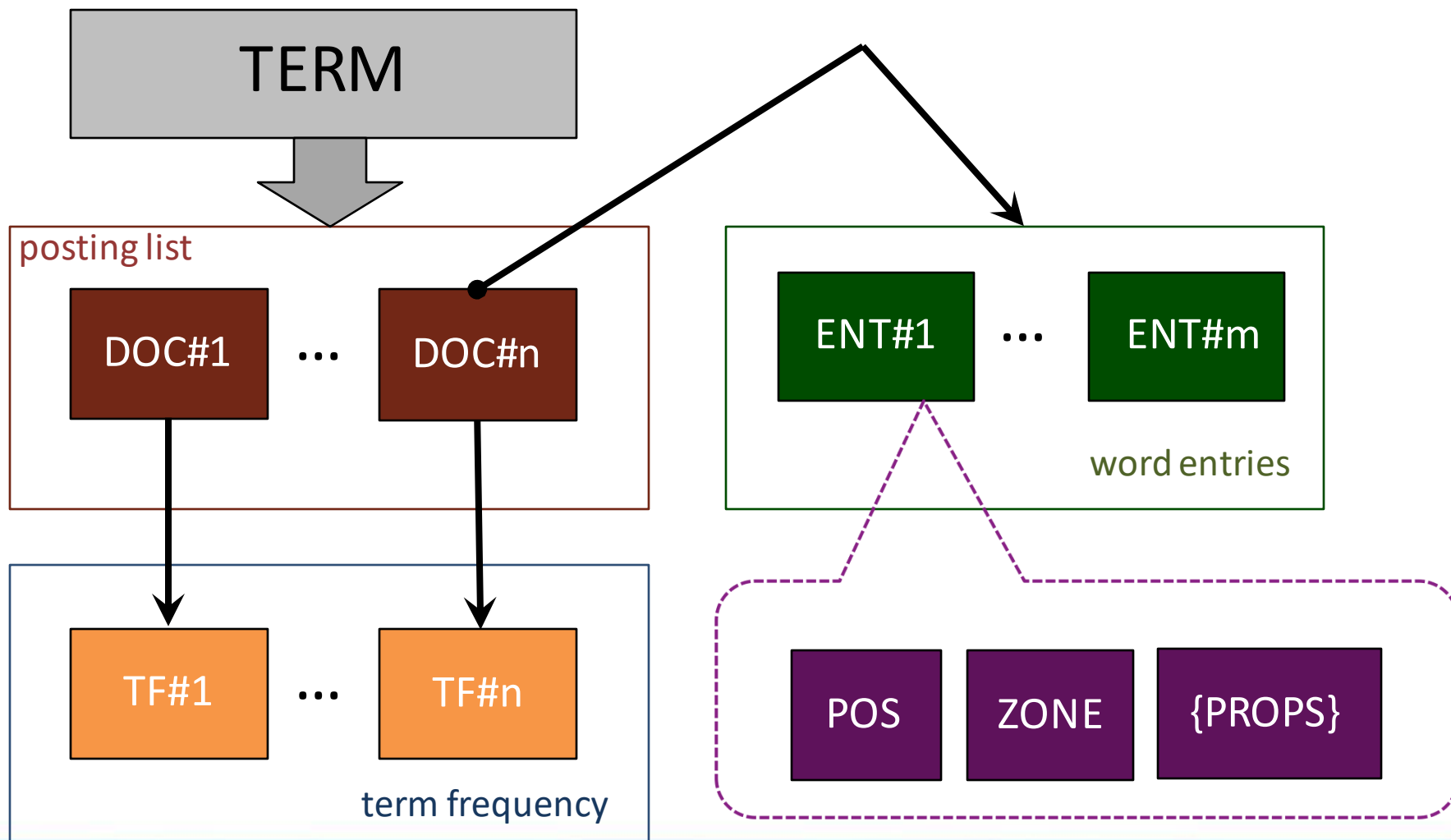


Чем дополнить обратный индекс?

1. Ранг термина
2. Координаты

Нужные данные для поиска и ранжирования
Мы ограничены размером и скоростью

Состав обратного индекса



Быстрый и компактный

1. Быстрый:

1. Больше нагрузка – все запросы
2. Пользователь не будет ждать!

2. Компактный:

1. Завязано на скорость – можем хранить в RAM

+

Гибкий:

- Хранить разные данные (зонные индексы)
- Масштабируемый / разделяемый

Физические ограничения

1. RAM быстрее HDD на 2-3 порядка
2. SSD? Быстро, но пока недостаточно надежно для ВНС
3. Гибриды – дорого и малый объем
4. RAM ограничена по объему

Скорость меряется в IOpS – Input/Output per Second



Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Память: как правильно с ней работать?

1. Считать блок:

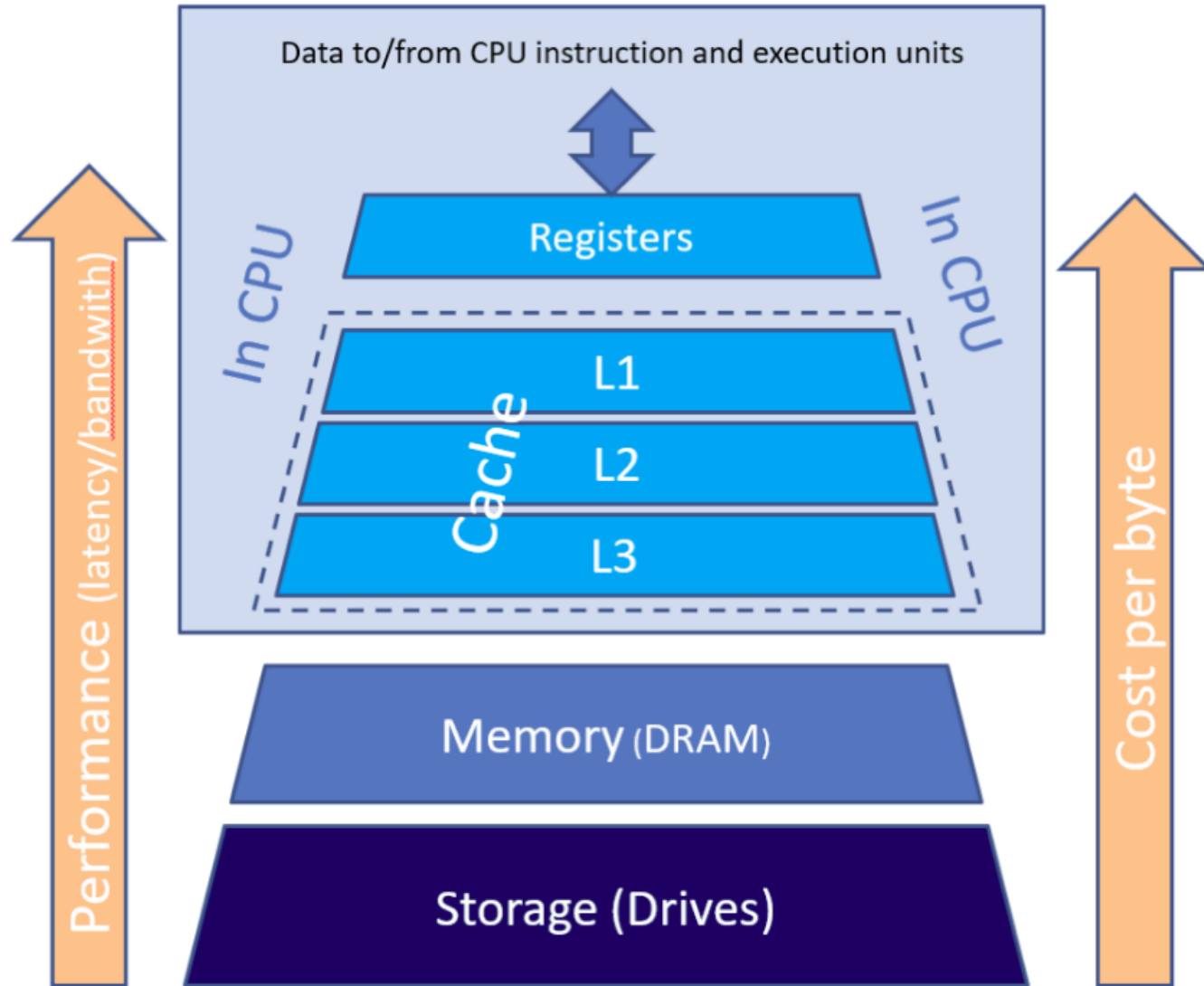
- Спозиционироваться
- Считать

2. Что быстрее?

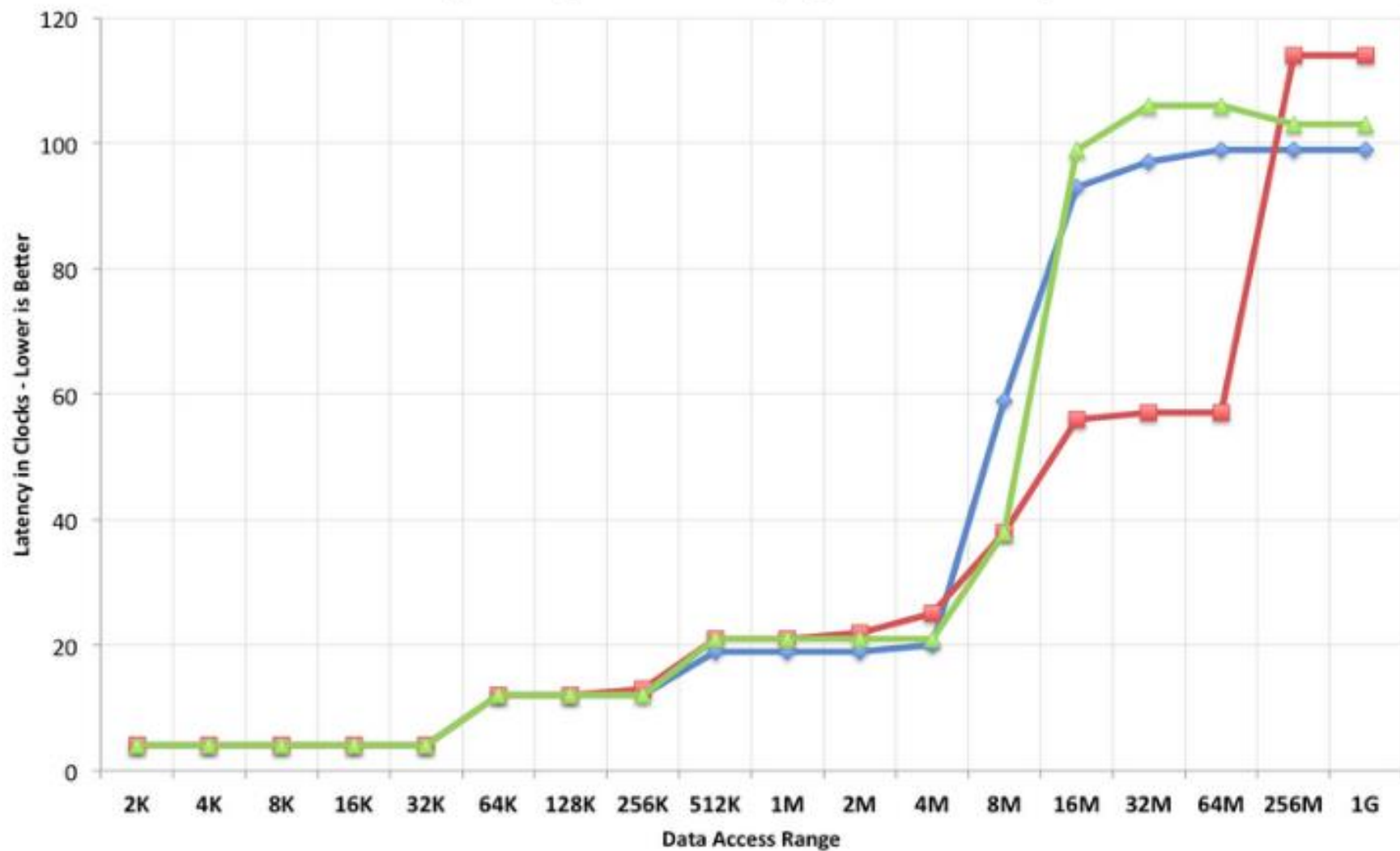
1. Считать 1GB, записанный непрерывно
2. Считать 1024 блока по 1MB
3. Считать 1024×1024 блока по 1KB

Память: как правильно с ней работать?

1. Меньше позиционируемся – больше читаем
2. Меньший объем данных – меньше читать

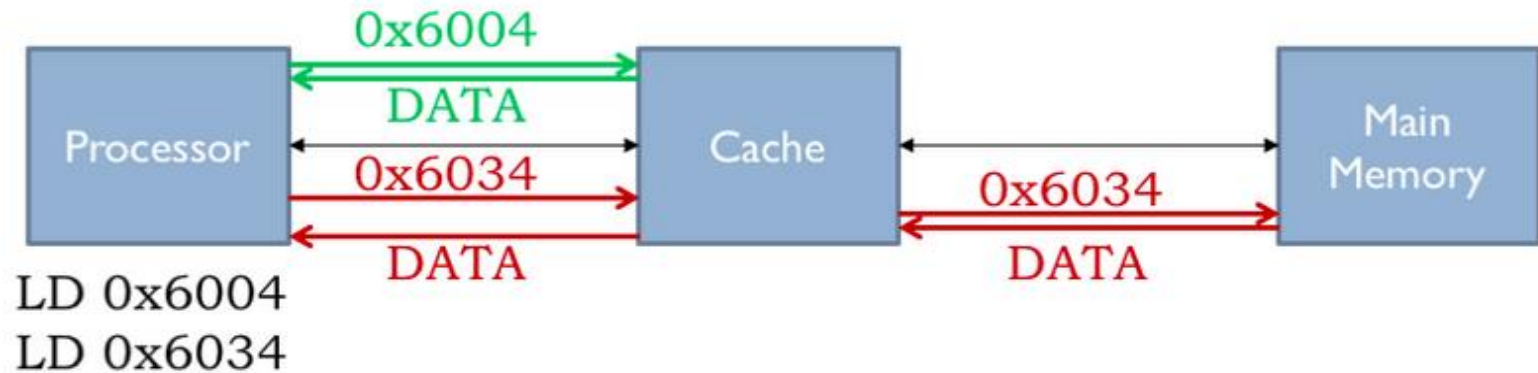


The traditional memory hierarchy pyramid.





Cache Access



- Processor sends address to cache
- Two options:
 - **Cache hit**: Data for this address in cache, returned quickly
 - **Cache miss**: Data not in cache
 - Fetch data from memory, send it back to processor
 - Retain this data in the cache (replacing some other data)
 - Processor must deal with variable memory access time

Размер индекса

1. 10кМ документов
2. 1 документ \sim 70Kb
3. 1 лемма \sim 8b

Проблема:

очень большой словарь

Разделяемый индекс

Как поделить большой индекс между несколькими серверами?



Разделяемый индекс

1. Сервер \leftrightarrow терм
2. Сервер \leftrightarrow документ

Бинарный поиск

$A \& B$


$A \parallel B$

$\neg B$


Пересечение блоков


 x Найти

 [Интернет](#)

 Соцсети beta

 Картинки

 Видео

 Новости

 Ответы

Пересечение блоков

 × Найти[Интернет](#)[Соцсети](#) beta[Картинки](#)[Видео](#)[Новости](#)[Ответы](#)

игра

94

7

12

55

57

43

керлинг

94

1

7


Пересечение блоков




×


Найти


 Интернет

 Соцсети beta

 Картинки

 Видео

 Новости

 Ответы

игра	94	7	12	55	57	43
керлинг	94	1	7			

Пересечение блоков

 × Найти[Интернет](#)[Соцсети](#) beta[Картинки](#)[Видео](#)[Новости](#)[Ответы](#)

игра

7

12

43

55

57

94

керлинг

1

7

94

Пройти по всему списку – долго

Пересечение блоков

 × Найти[Интернет](#)[Соцсети](#) beta[Картинки](#)[Видео](#)[Новости](#)[Ответы](#)

игра

7

12

43

55

57

94

керлинг

1

7

94

Сложность $O(N + M)$

Пересечение блоков

 × Найти[Интернет](#)[Соцсети](#) beta[Картинки](#)[Видео](#)[Новости](#)[Ответы](#)

игра

7

12

43

55

57

94

керлинг

1

7

94

Сложность при бин. поиске $O(M \log N + M)$

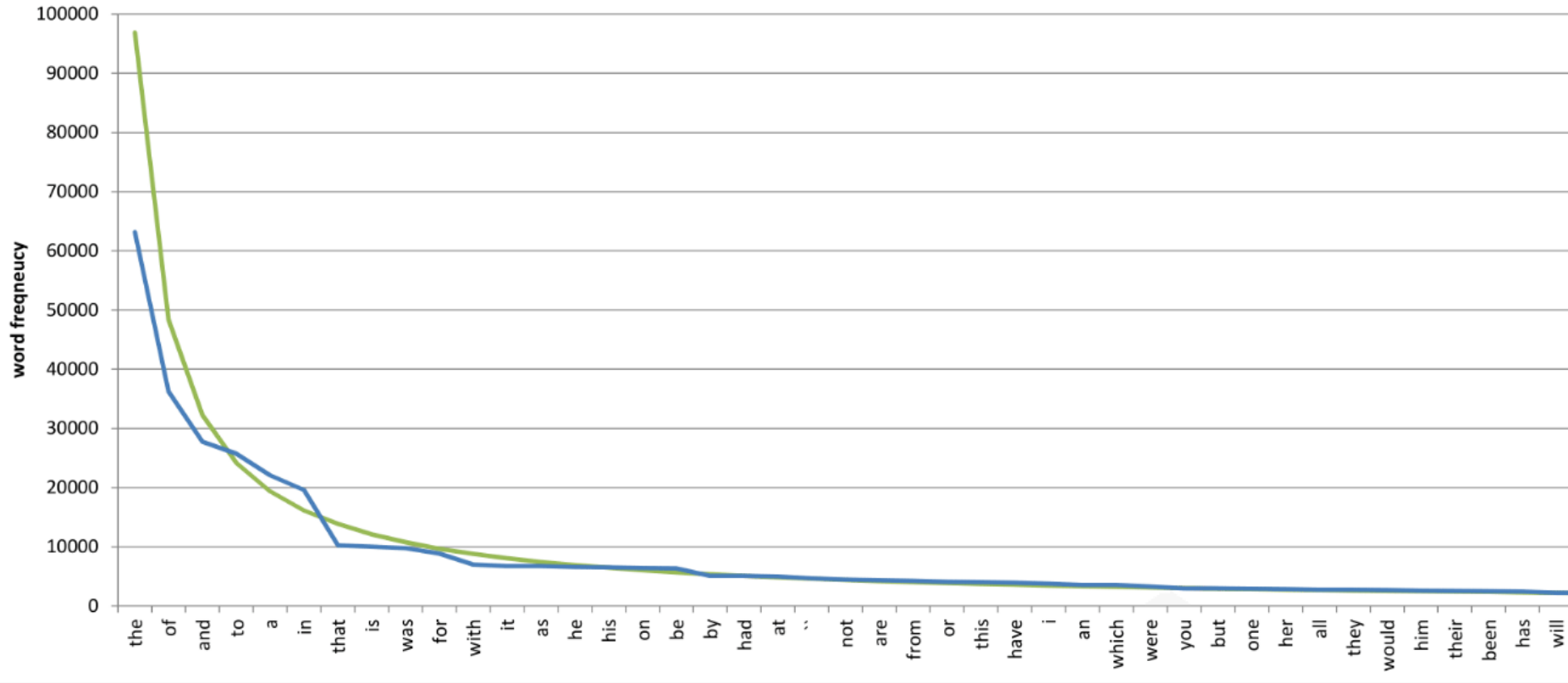
Пересечение блоков

 × Найти[Интернет](#)[Соцсети](#) beta[Картинки](#)[Видео](#)[Новости](#)[Ответы](#)

игра	7	12	43	55	57	94
керлинг	1	7	94			

На практике $O(M \log N)$, при $M \ll N$

Закон ципфа



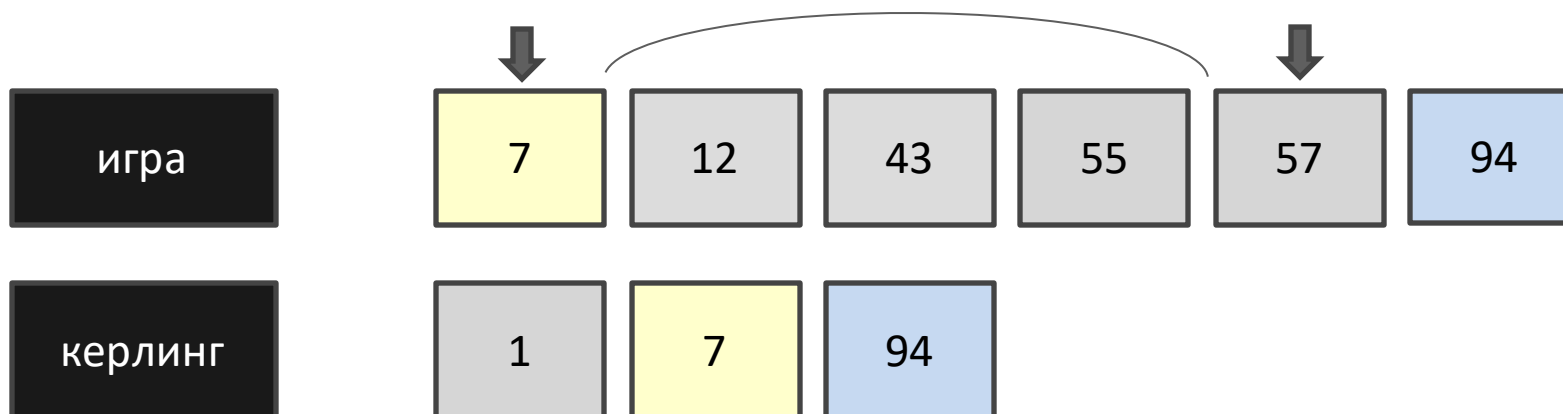
На практике $O(M \log N)$, при $M \ll N$

Пересечение блоков



× Найти

[Интернет](#) [Соцсети beta](#) [Картинки](#) [Видео](#) [Новости](#) [Ответы](#)



Jump Tables!


Пересечение блоков





×


Найти

 Интернет

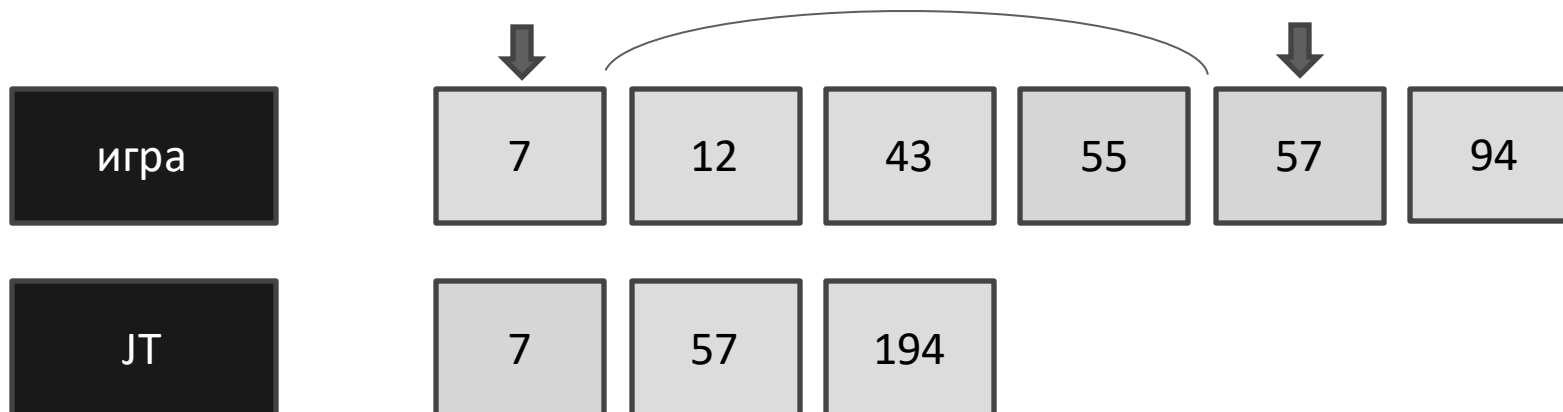
 Соцсети beta

 Картинки

 Видео

 Новости

 Ответы



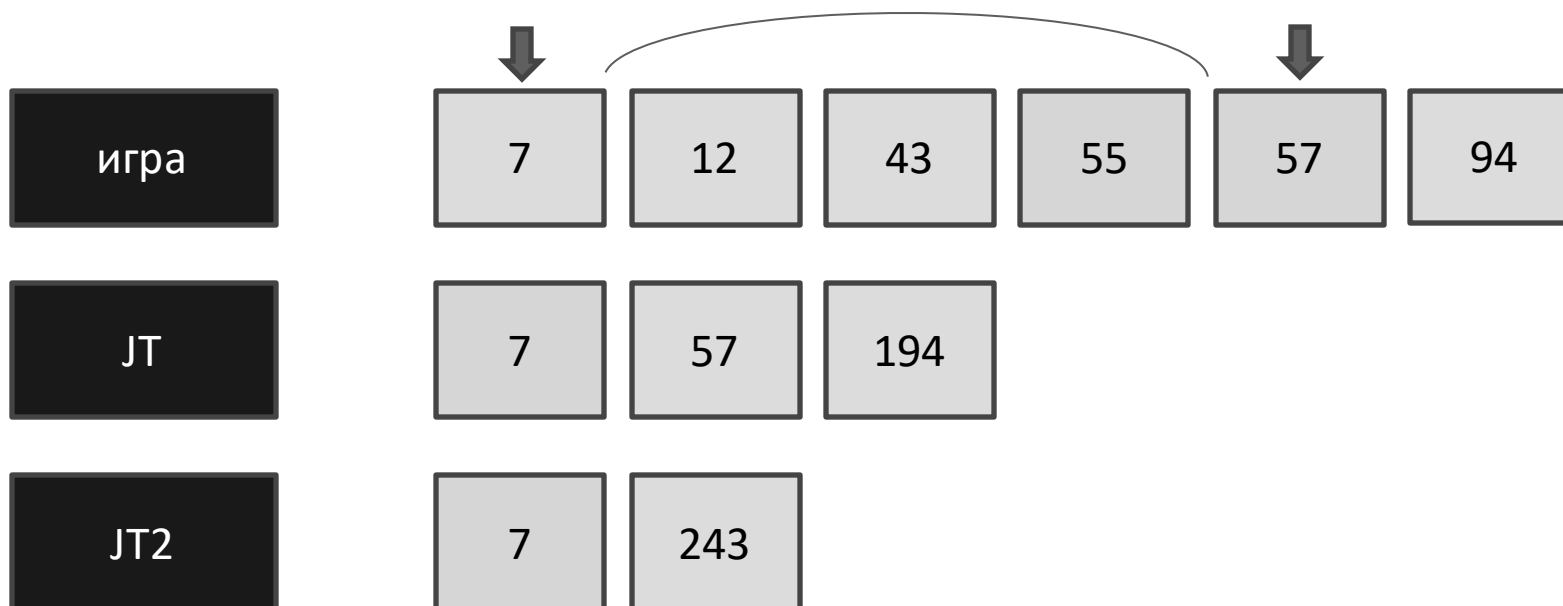
Jump Tables!

Пересечение блоков



× Найти

[Интернет](#) [Соцсети beta](#) [Картинки](#) [Видео](#) [Новости](#) [Ответы](#)



Исполнение запросов:

1. Словарь
2. Дерево запроса (Q-Tree)
3. Работа с индексом

Словарь

Терм <-> termID

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

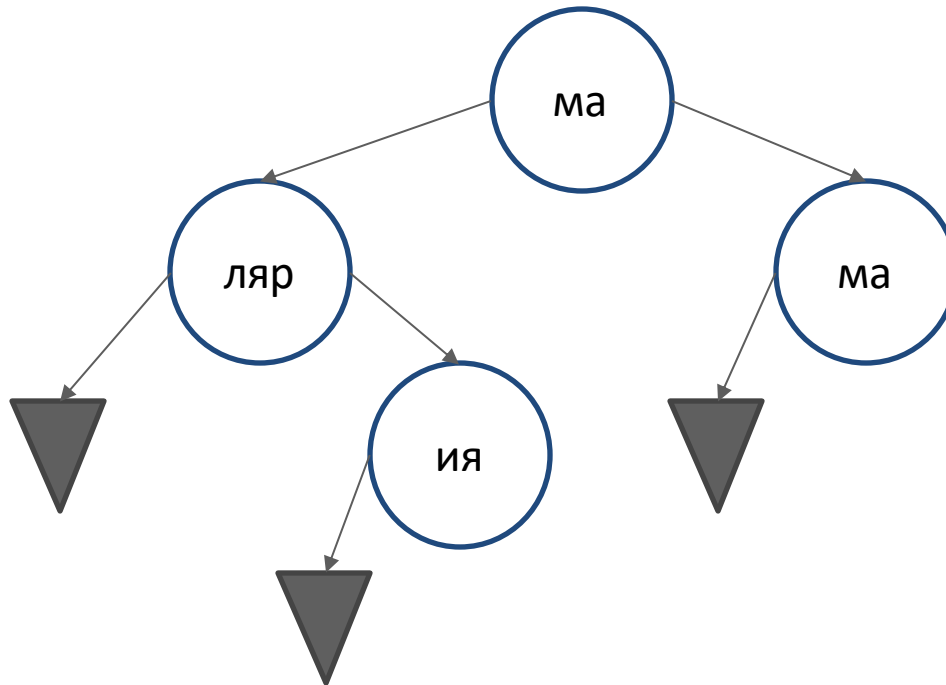
Словарь

Как хранить?

Словарь

Как хранить?

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



Словарь

Деревья:

- бинарные
- В-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 $N=3 \rightarrow$ нарезаем слово + маркеры границы по 3 символа

кошка \rightarrow \$кошка\$ \rightarrow \$ко + кош + ошк + шка + ка\$

Нечеткий поиск

N-граммы

Пусть $N=3 \rightarrow$ нарезаем слово + маркеры границы по 3 символа

кошка \rightarrow \$кошка\$ \rightarrow \$ко + кош + ошк + шка + ка\$

ко*ка \rightarrow \$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?

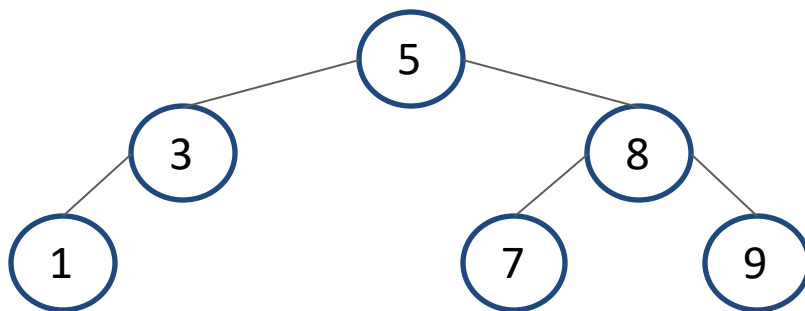
- инверсный индекс – список документов, где **НЕТ** этого слова
- учёт контекста

Стоп-слова

Учёт контекста

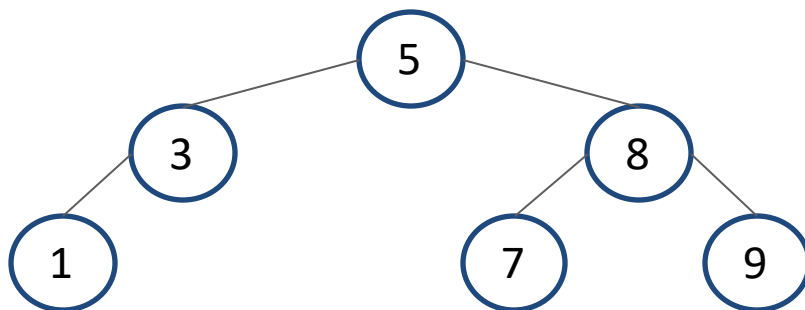
- «война и мир»:
 - война
 - йна_и
 - и_мир
 - мир

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»

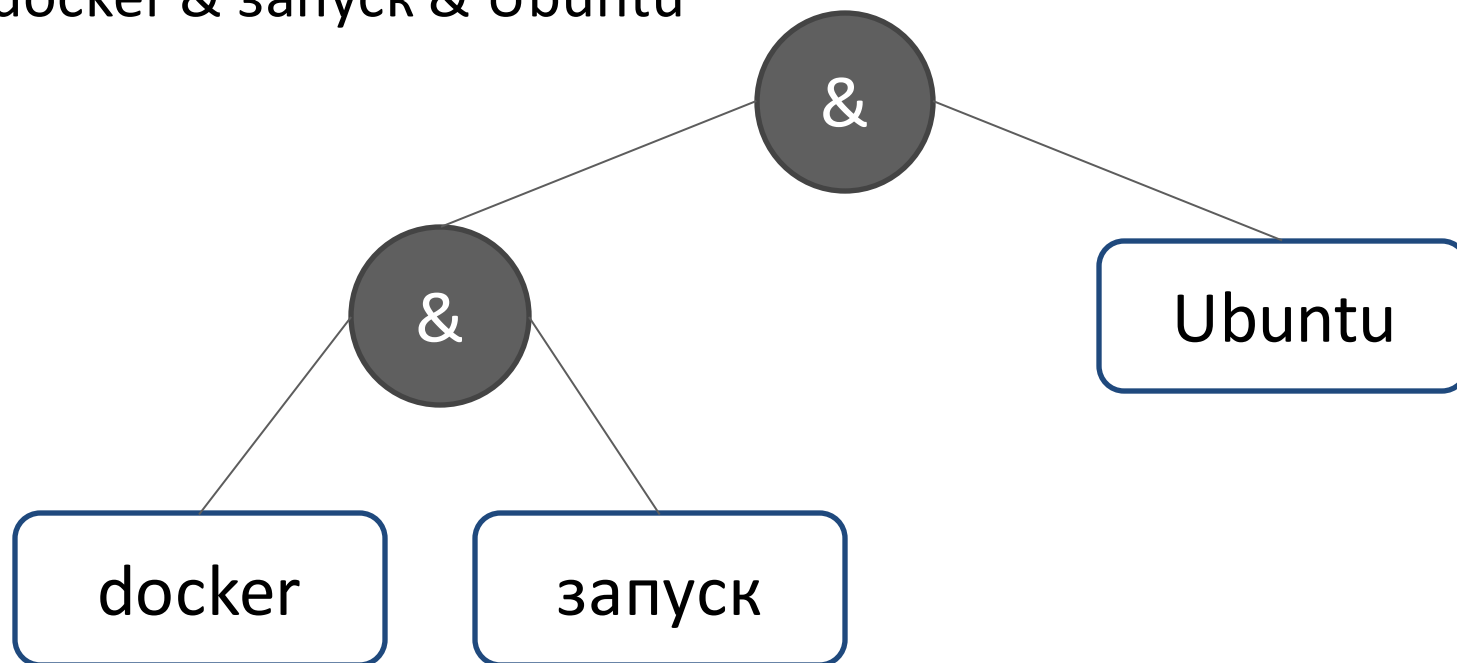
3	2	1	3	15	18	4	5	14	17
---	---	---	---	----	----	---	---	----	----

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

Исполнение запроса

Запрос: docker запуск Ubuntu

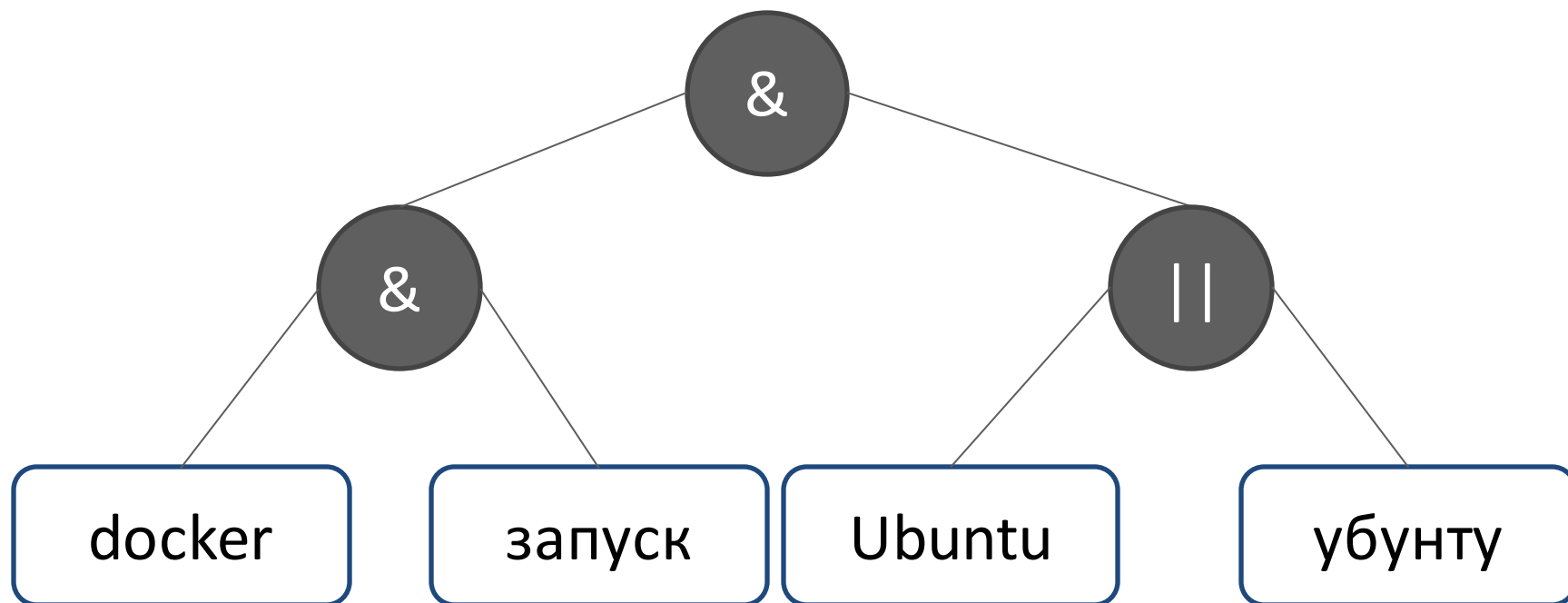
docker & запуск & Ubuntu



Исполнение запроса

Добавим синонимы:

docker & запуск & (Ubuntu || убунту)



Исполнение запроса

Подход «в лоб»:

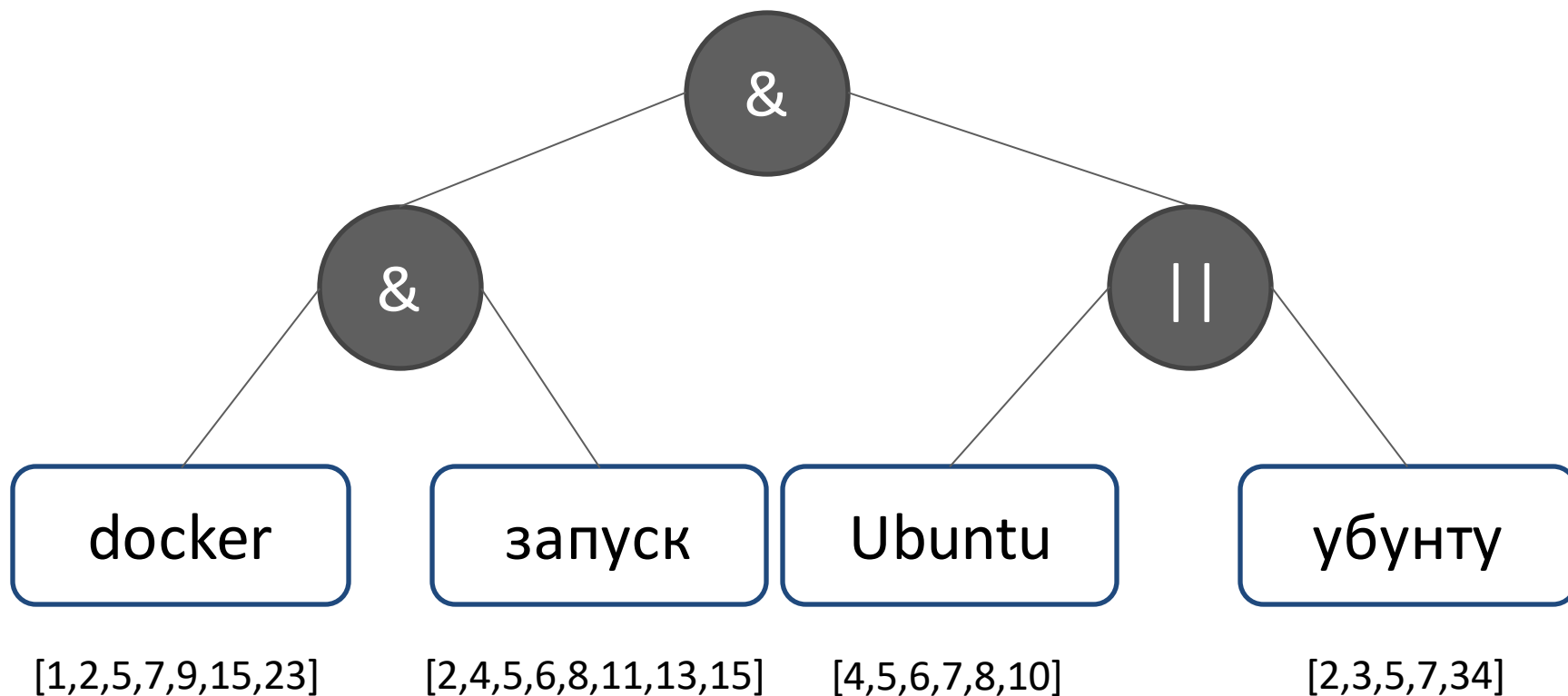
- находим множества документов для всех листьев (термы)
- идём вверх по дереву до корня

Исполнение запроса

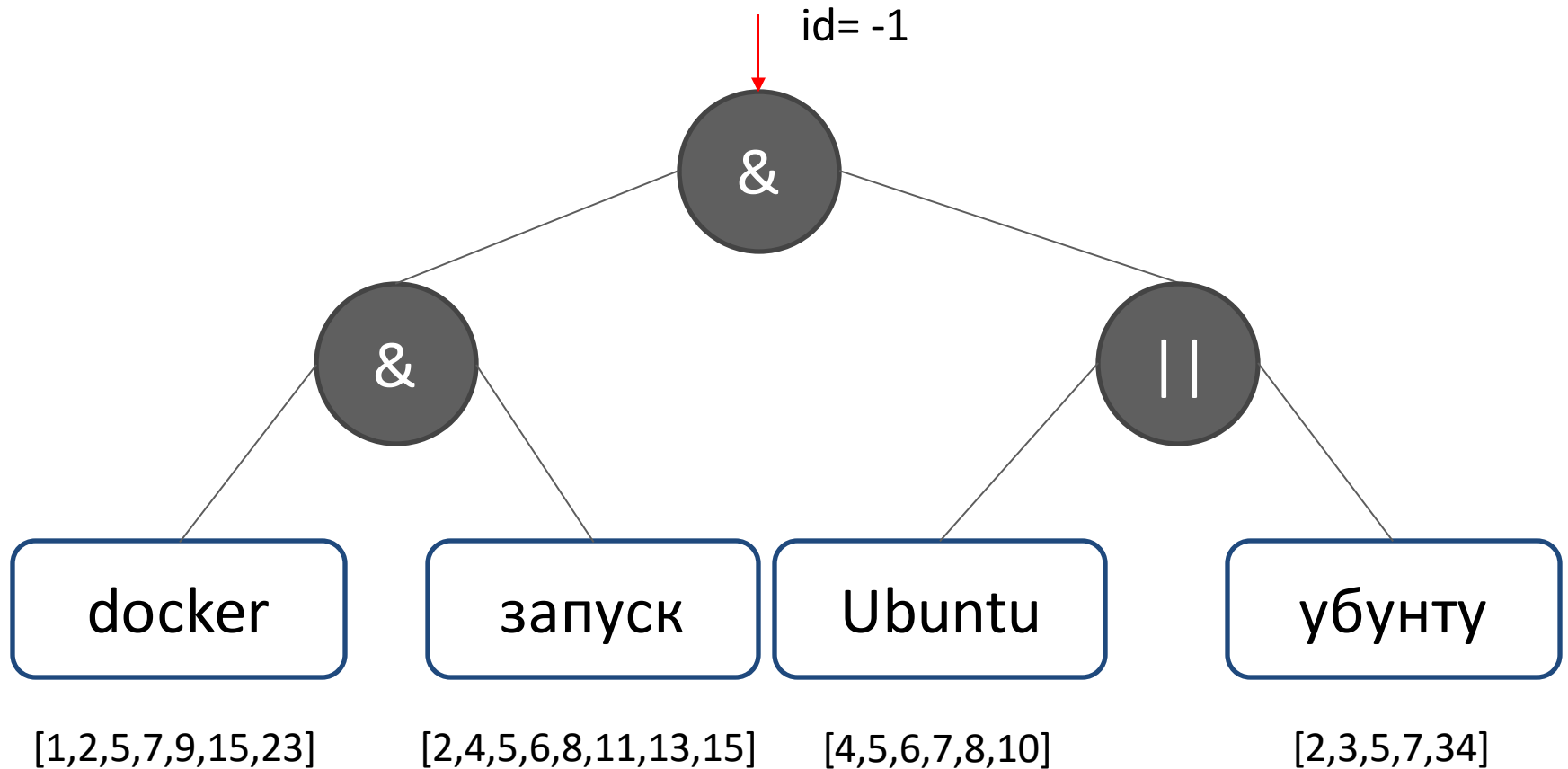
Подход «поточковый»:

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

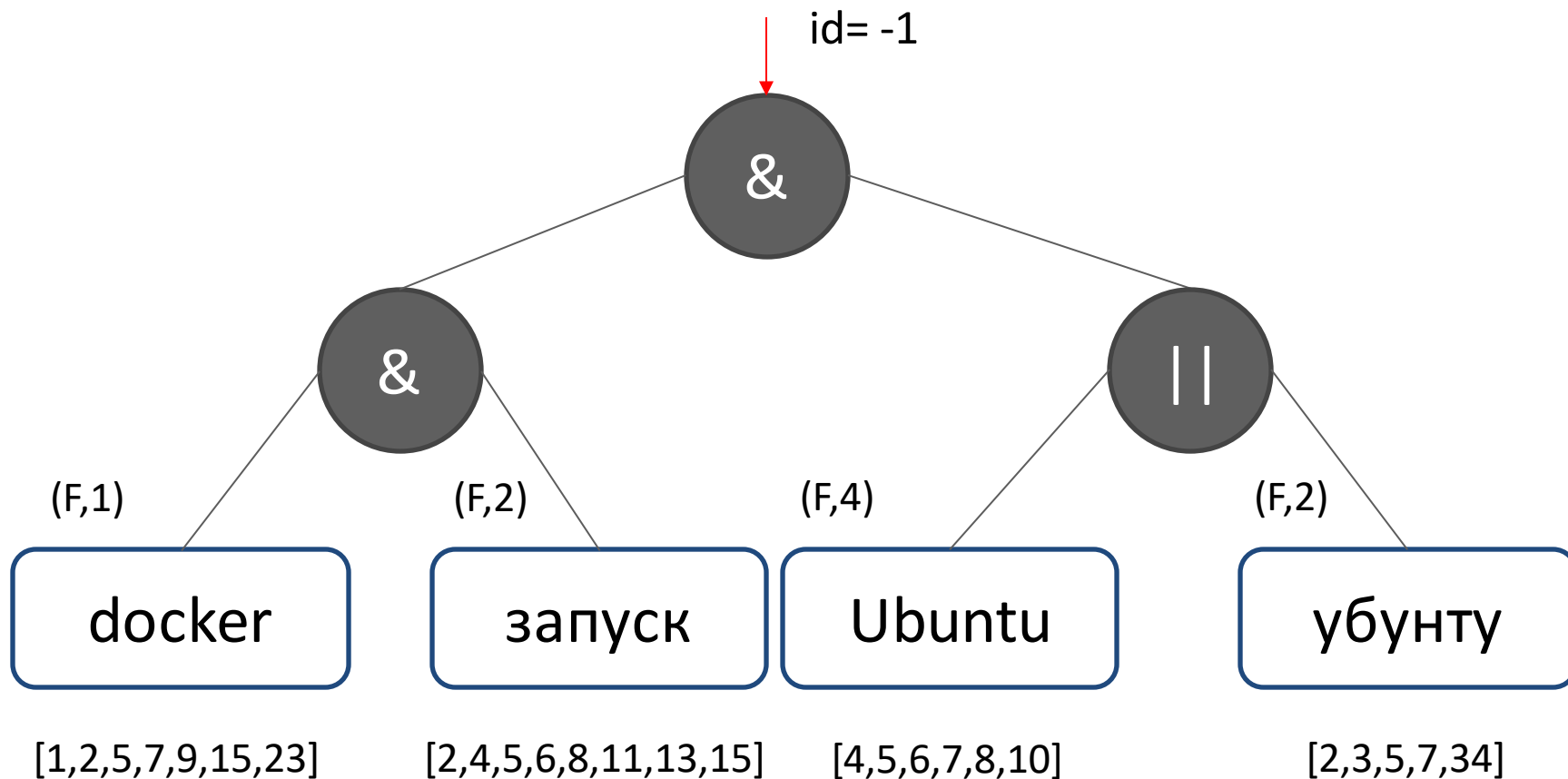
Исполнение запроса



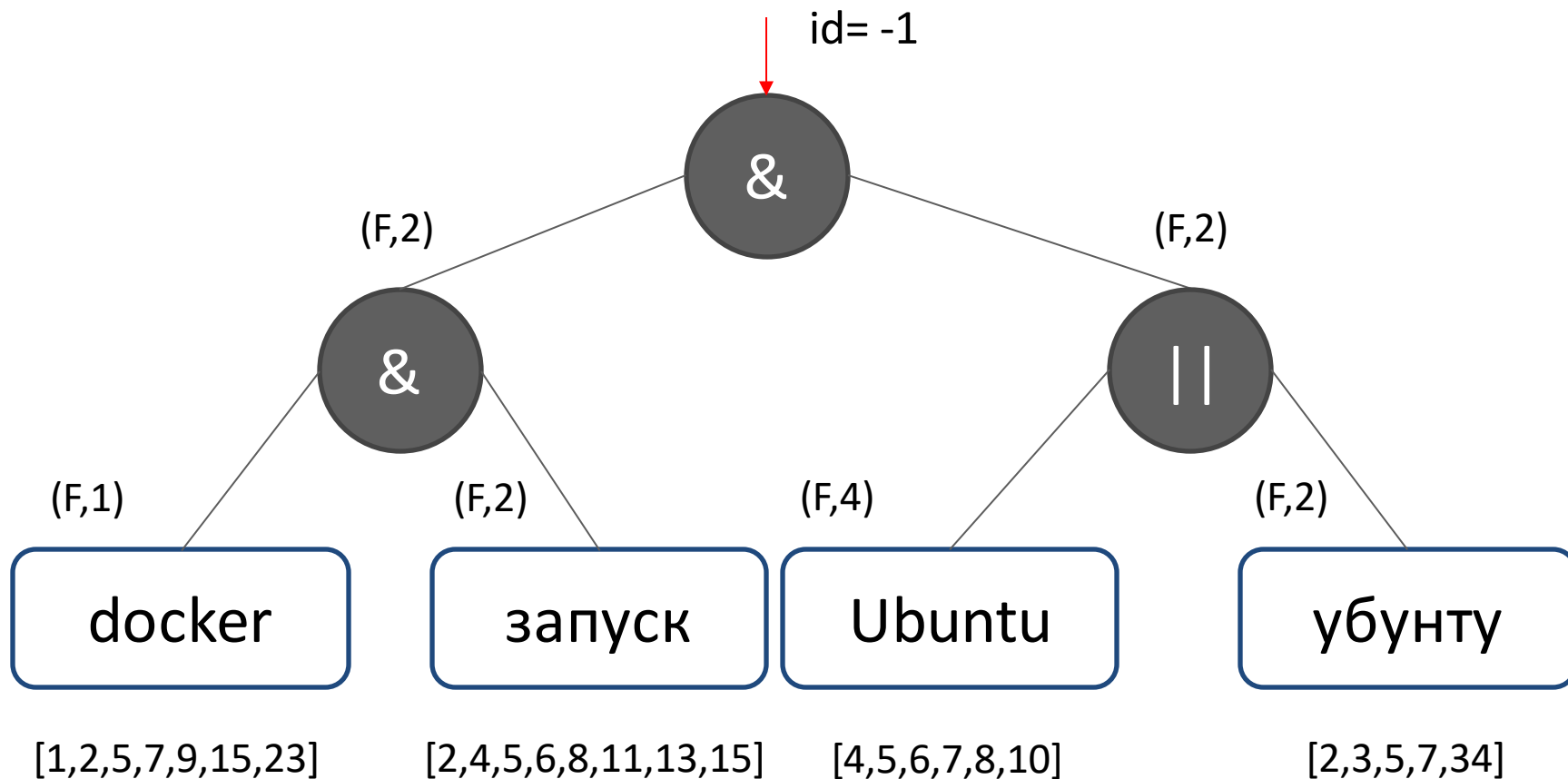
Исполнение запроса



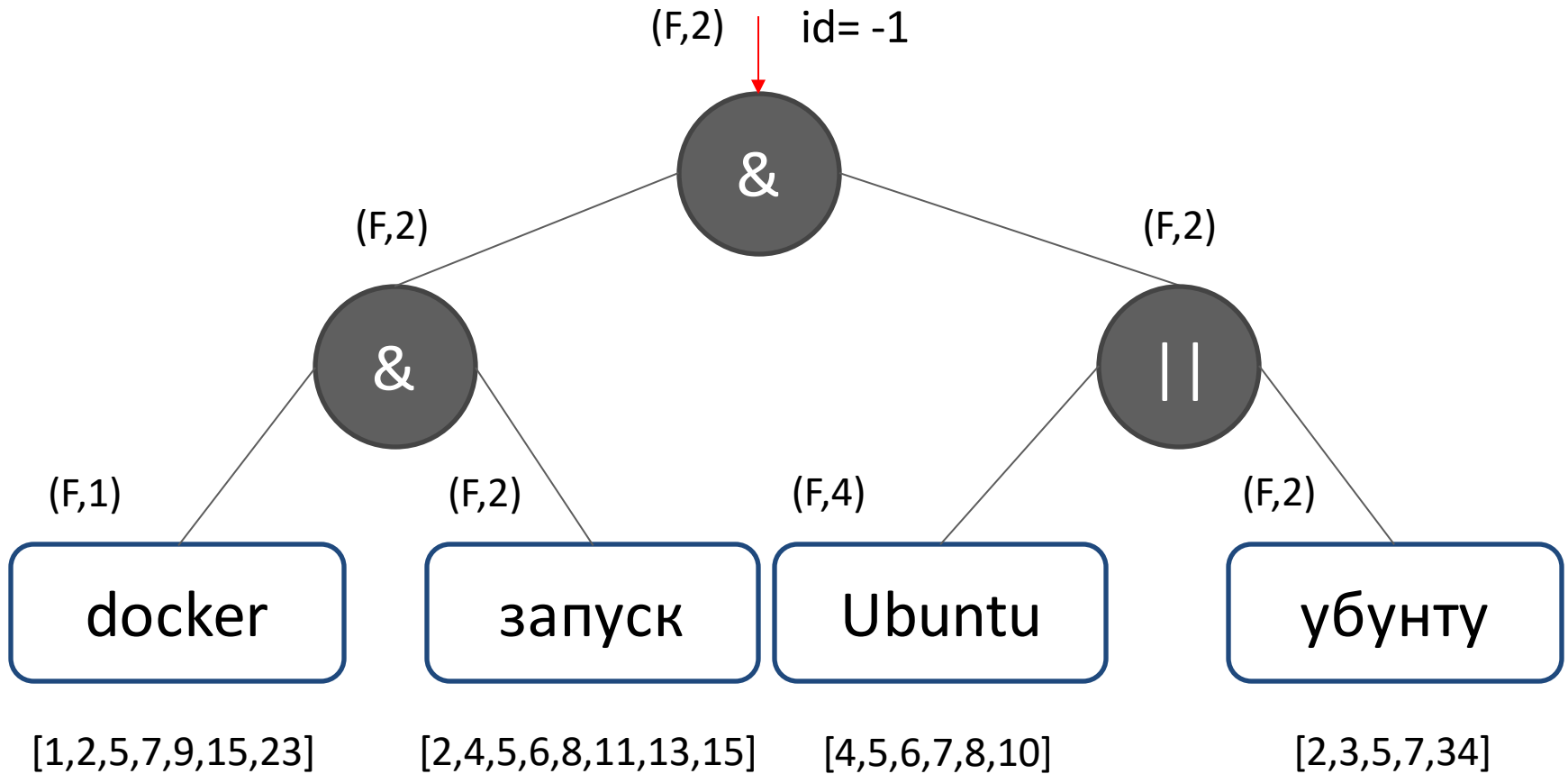
Исполнение запроса



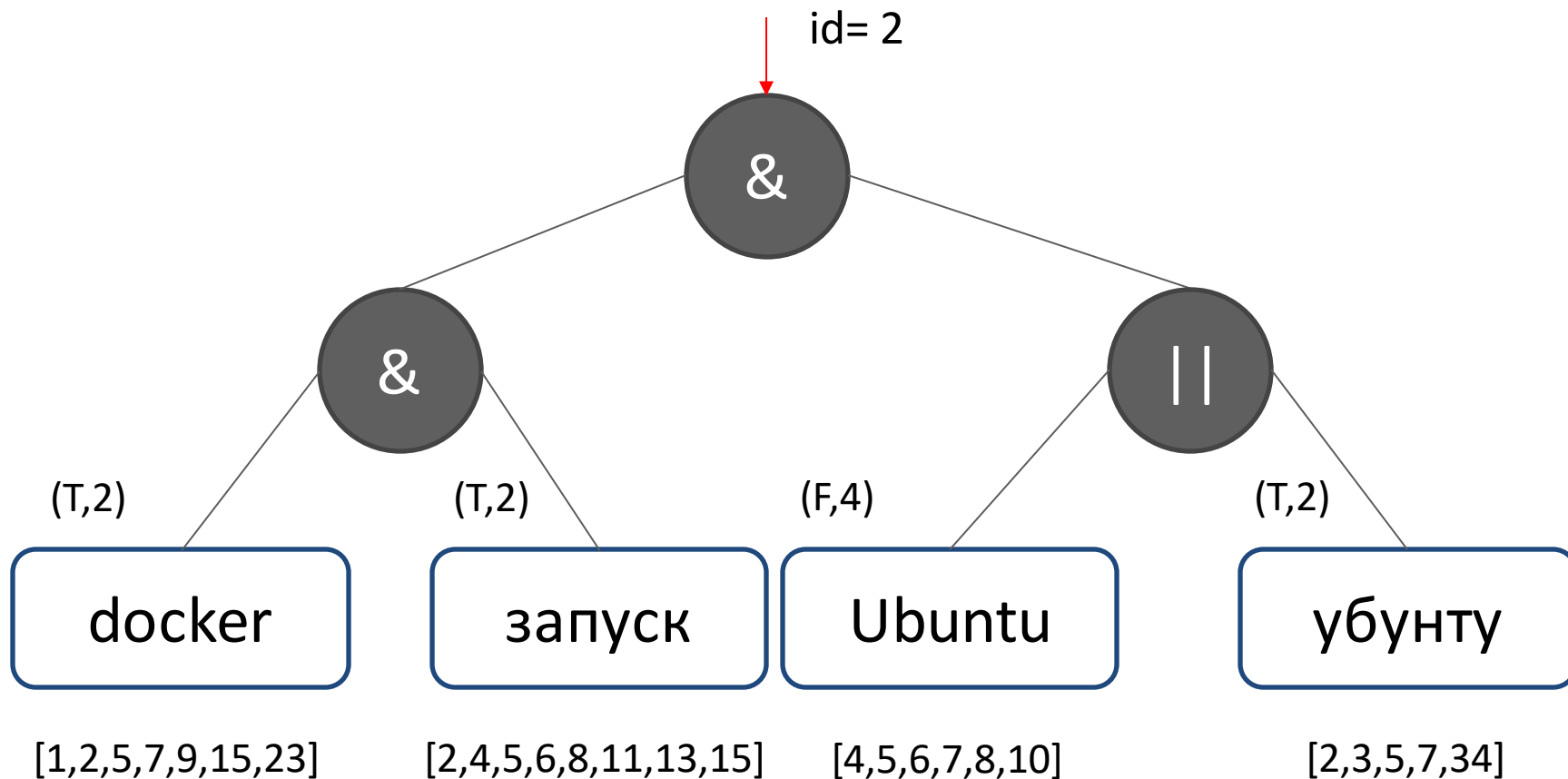
Исполнение запроса



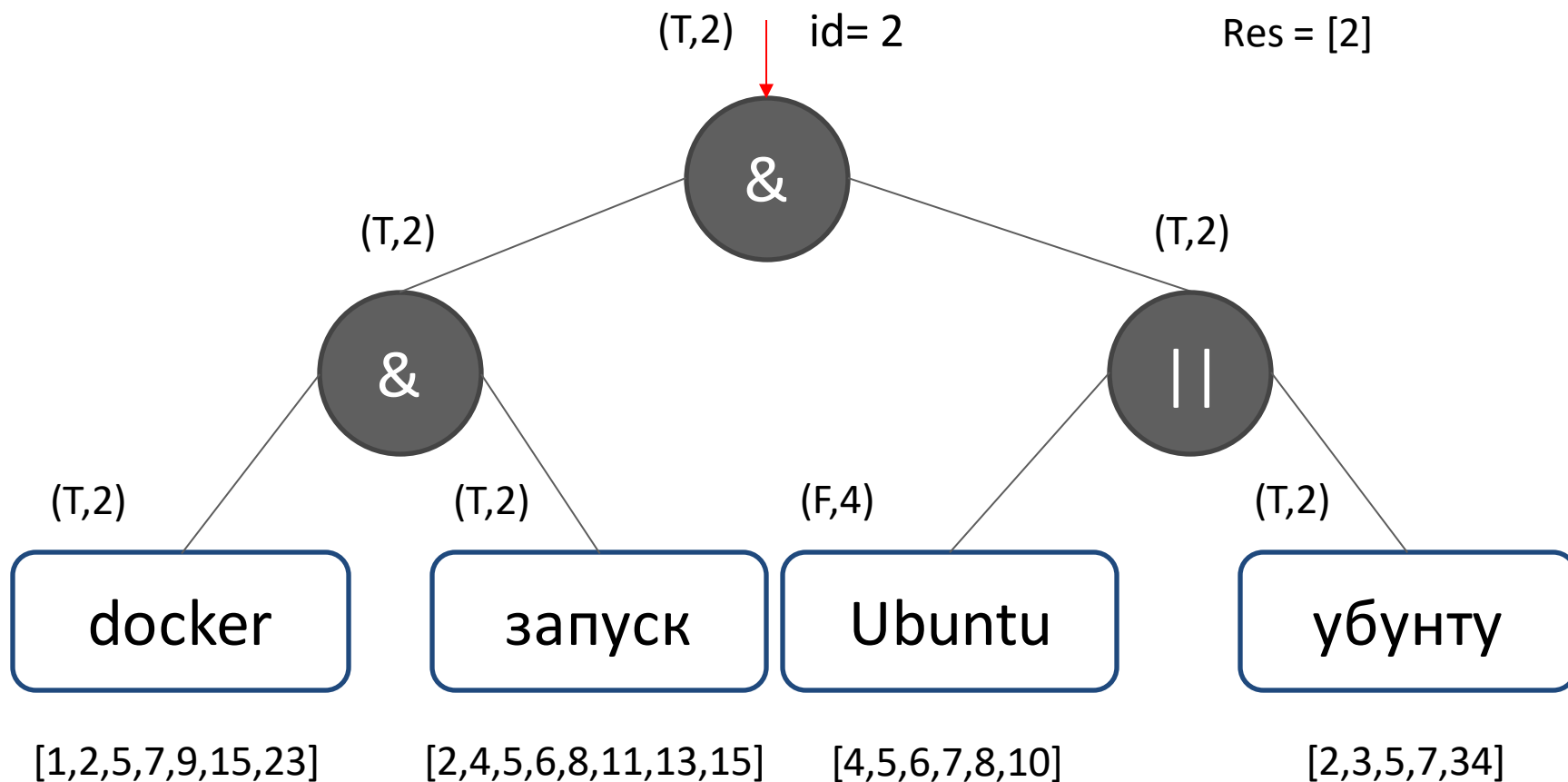
Исполнение запроса



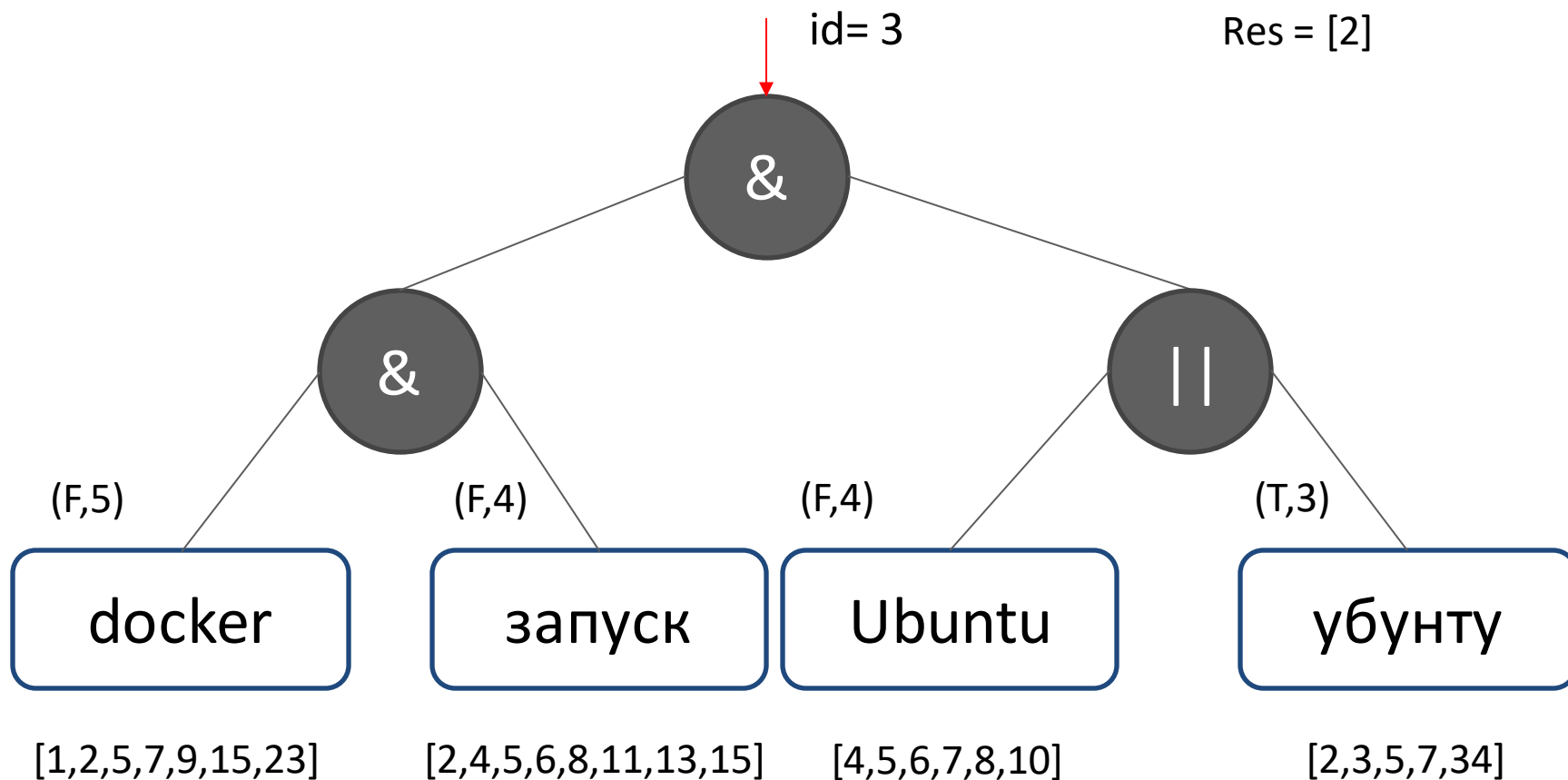
Исполнение запроса



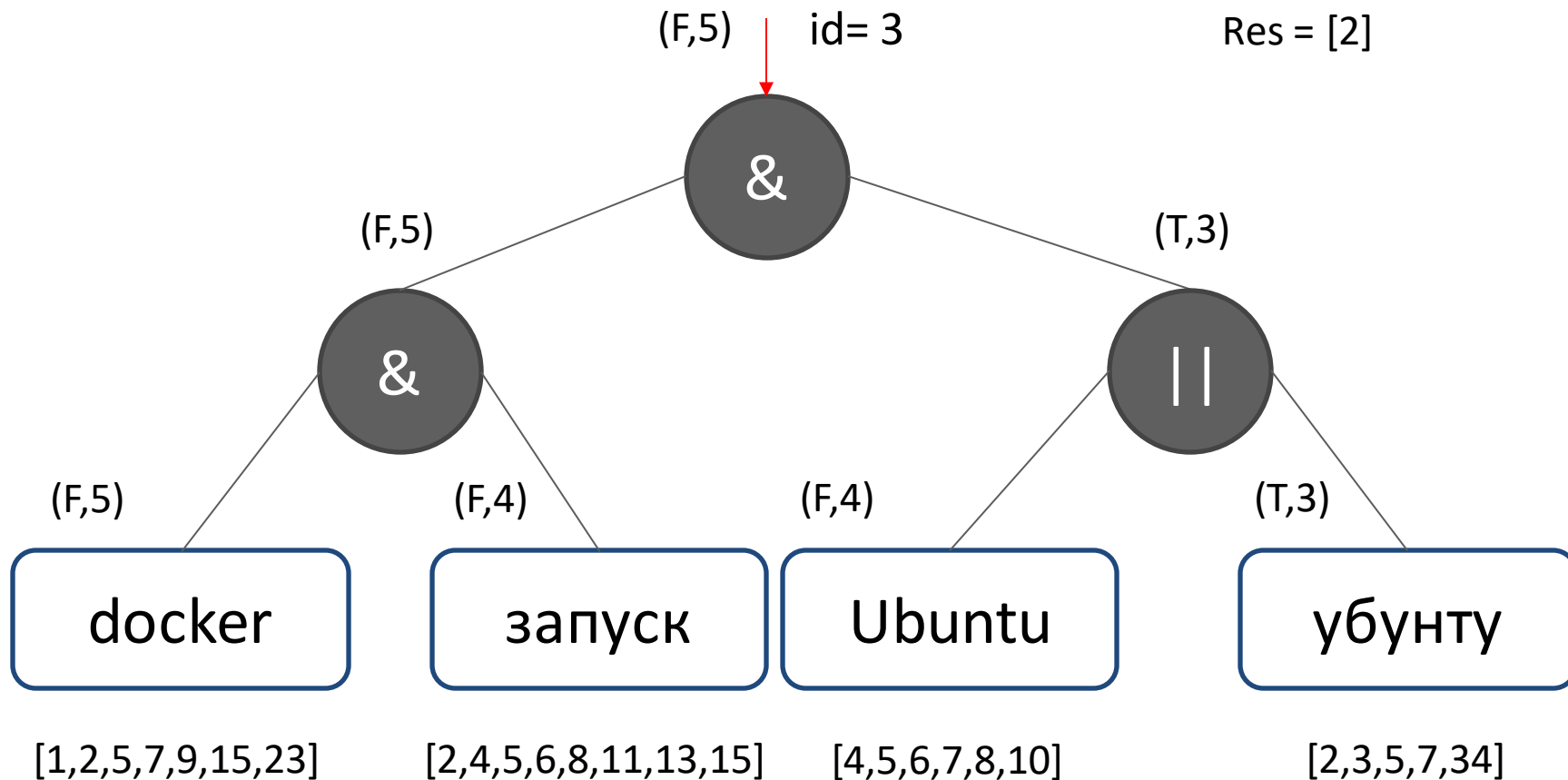
Исполнение запроса



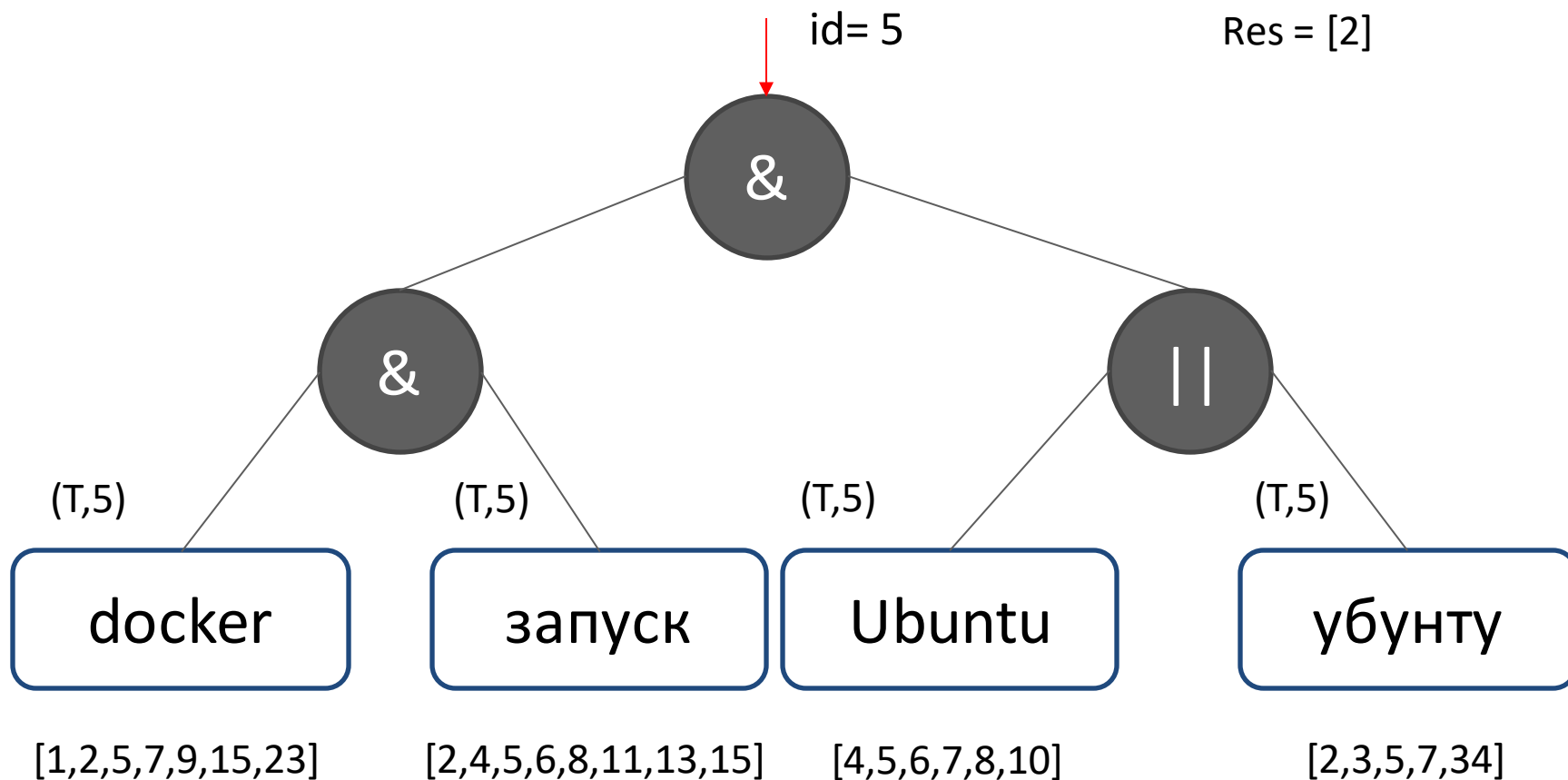
Исполнение запроса



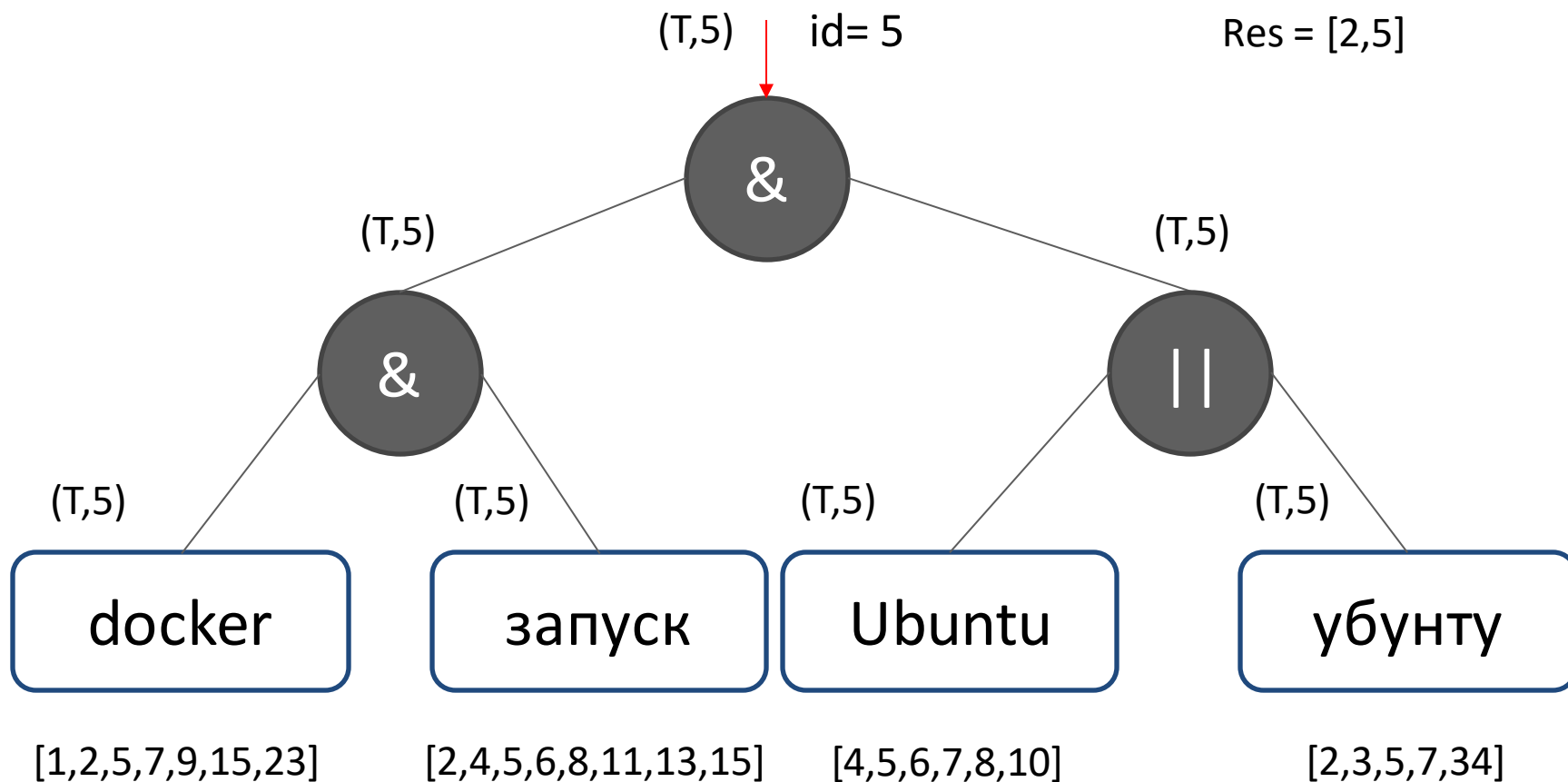
Исполнение запроса



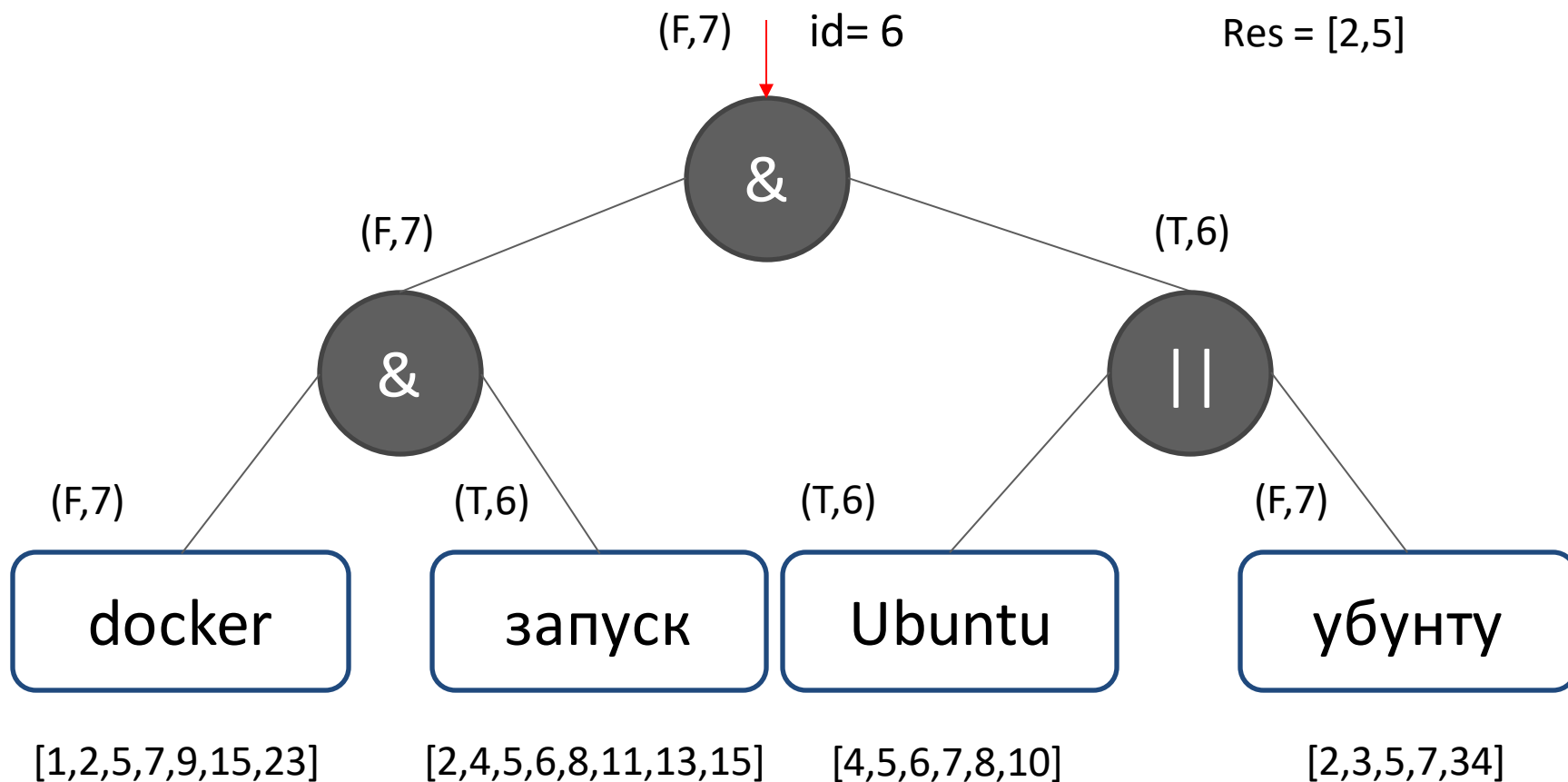
Исполнение запроса



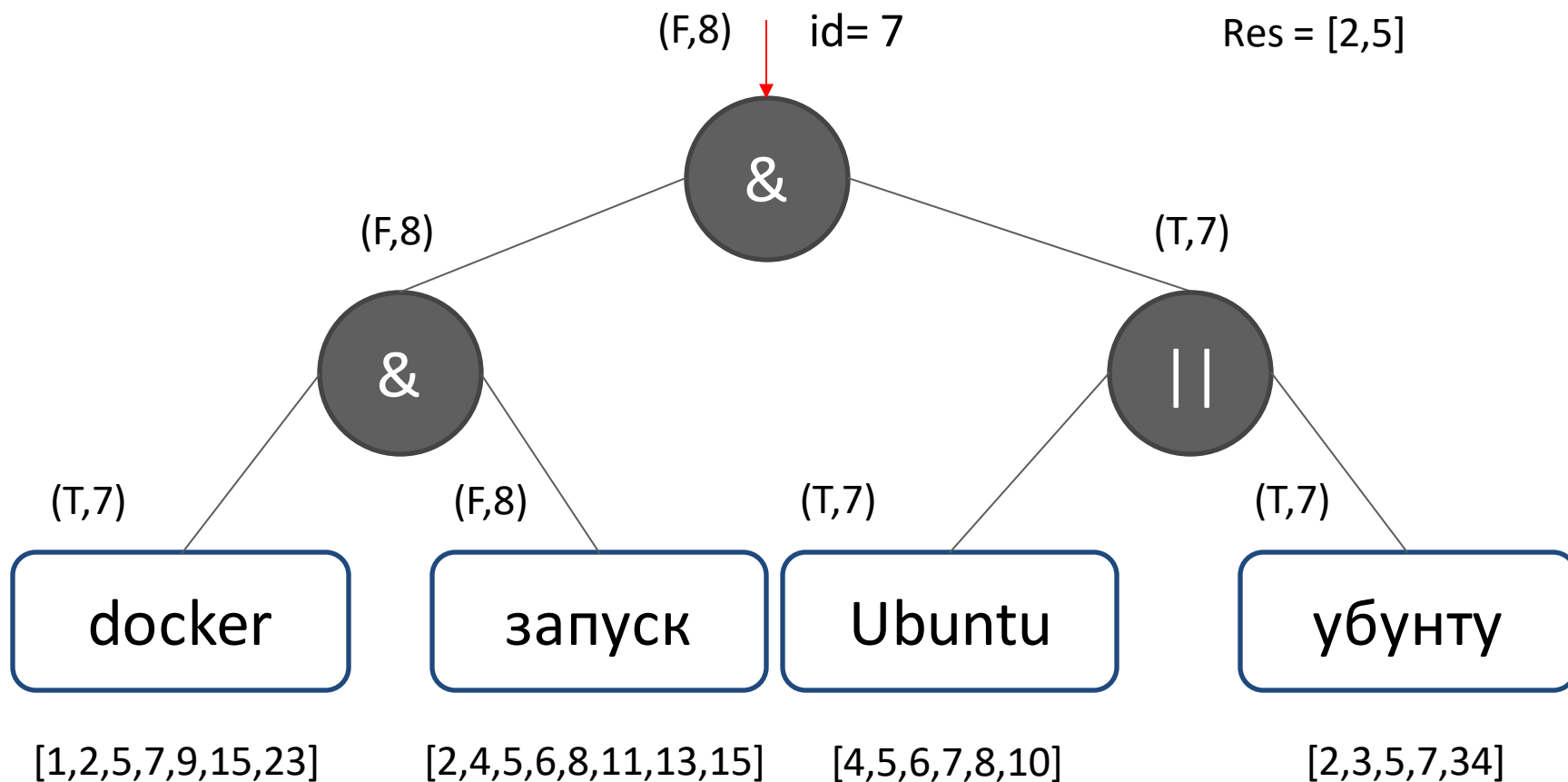
Исполнение запроса



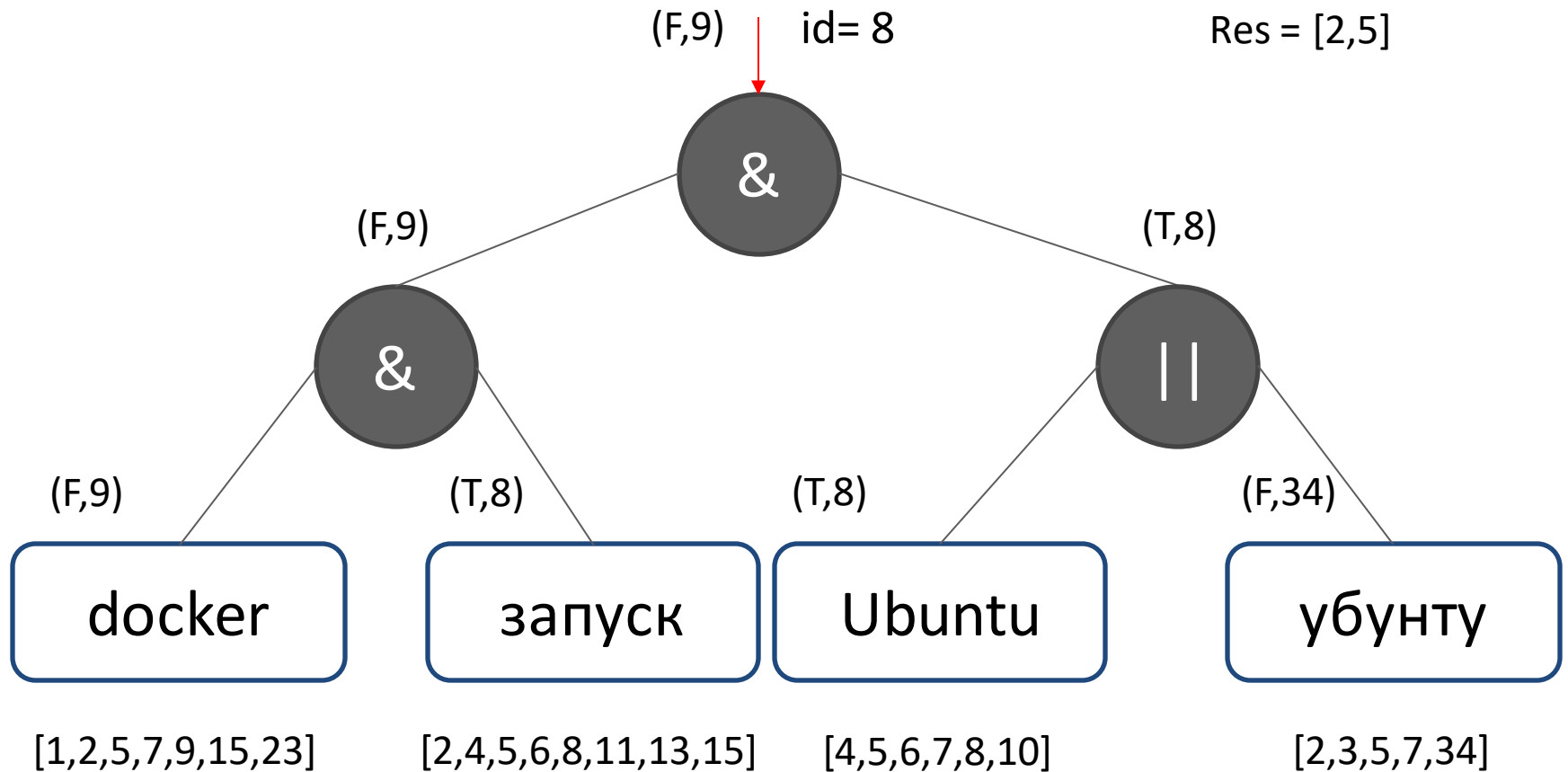
Исполнение запроса



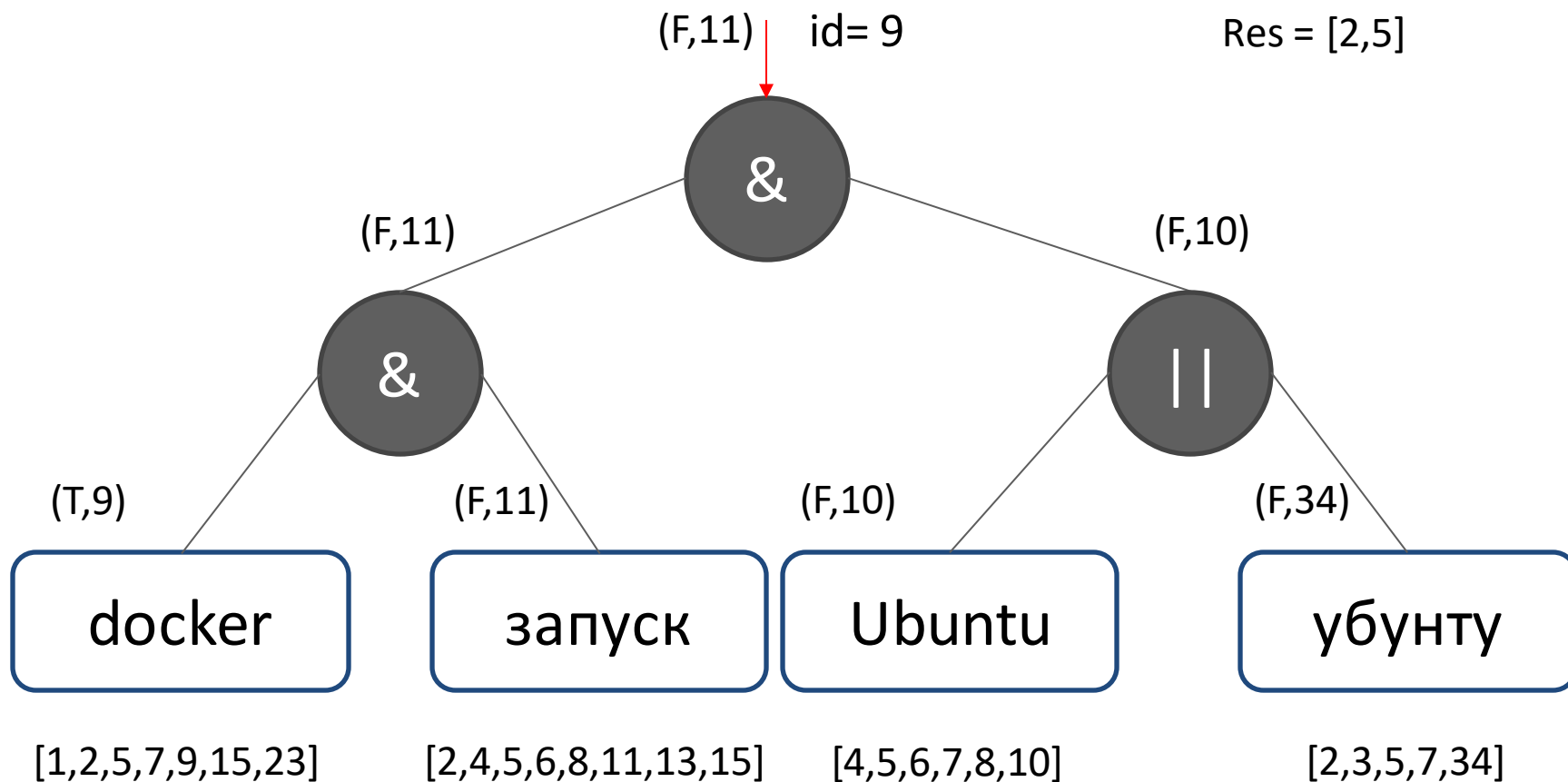
Исполнение запроса



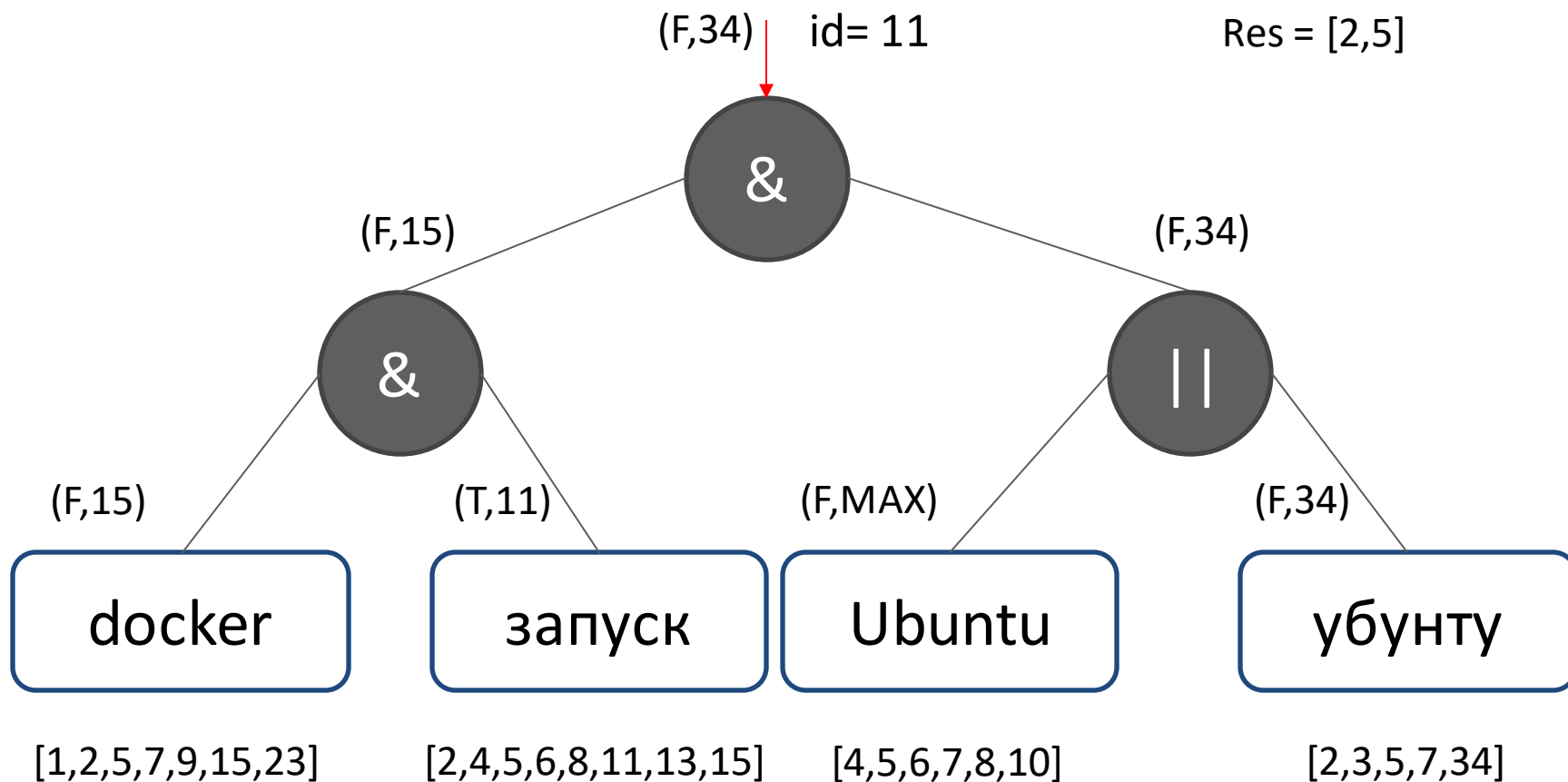
Исполнение запроса



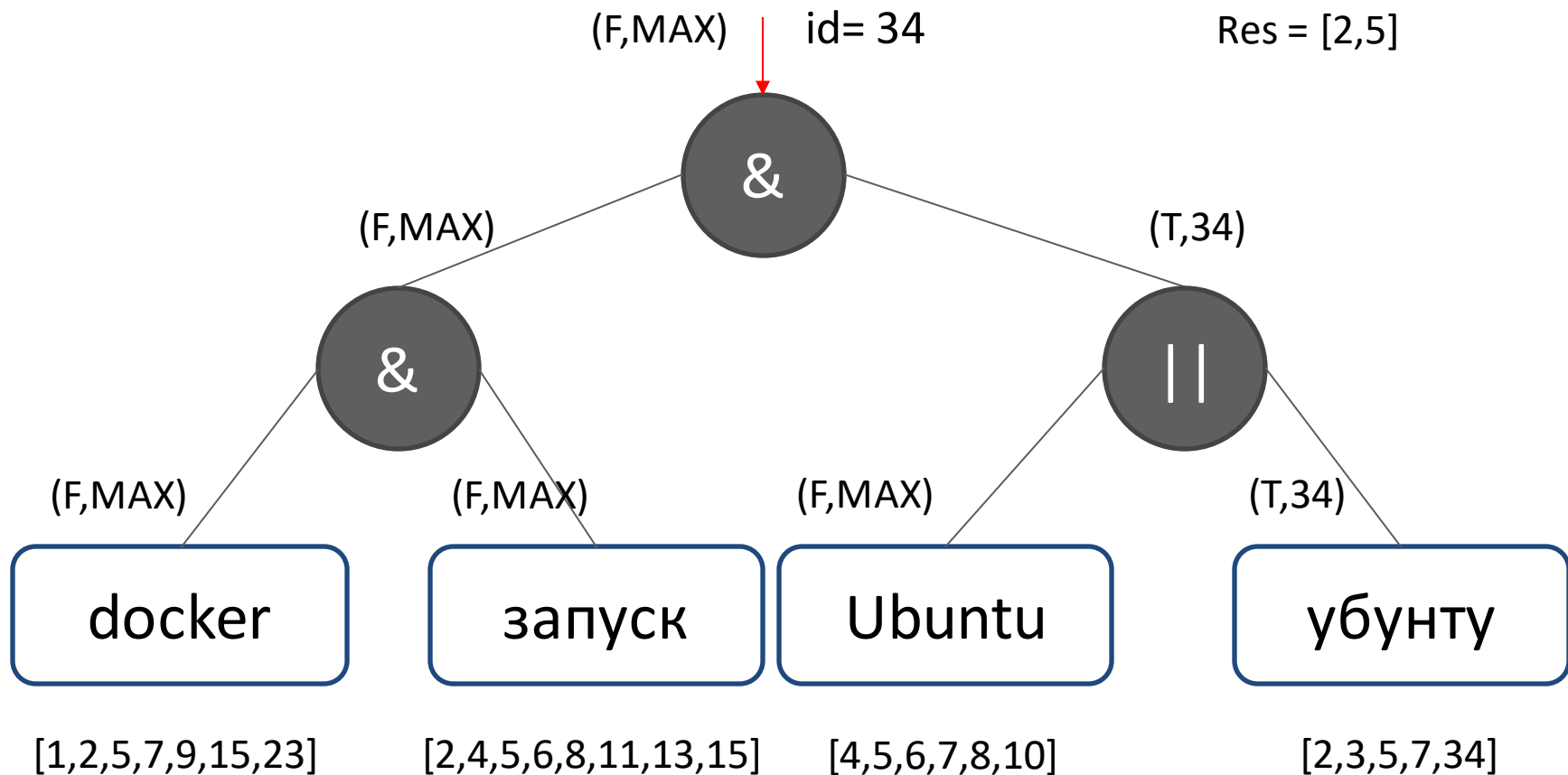
Исполнение запроса



Исполнение запроса



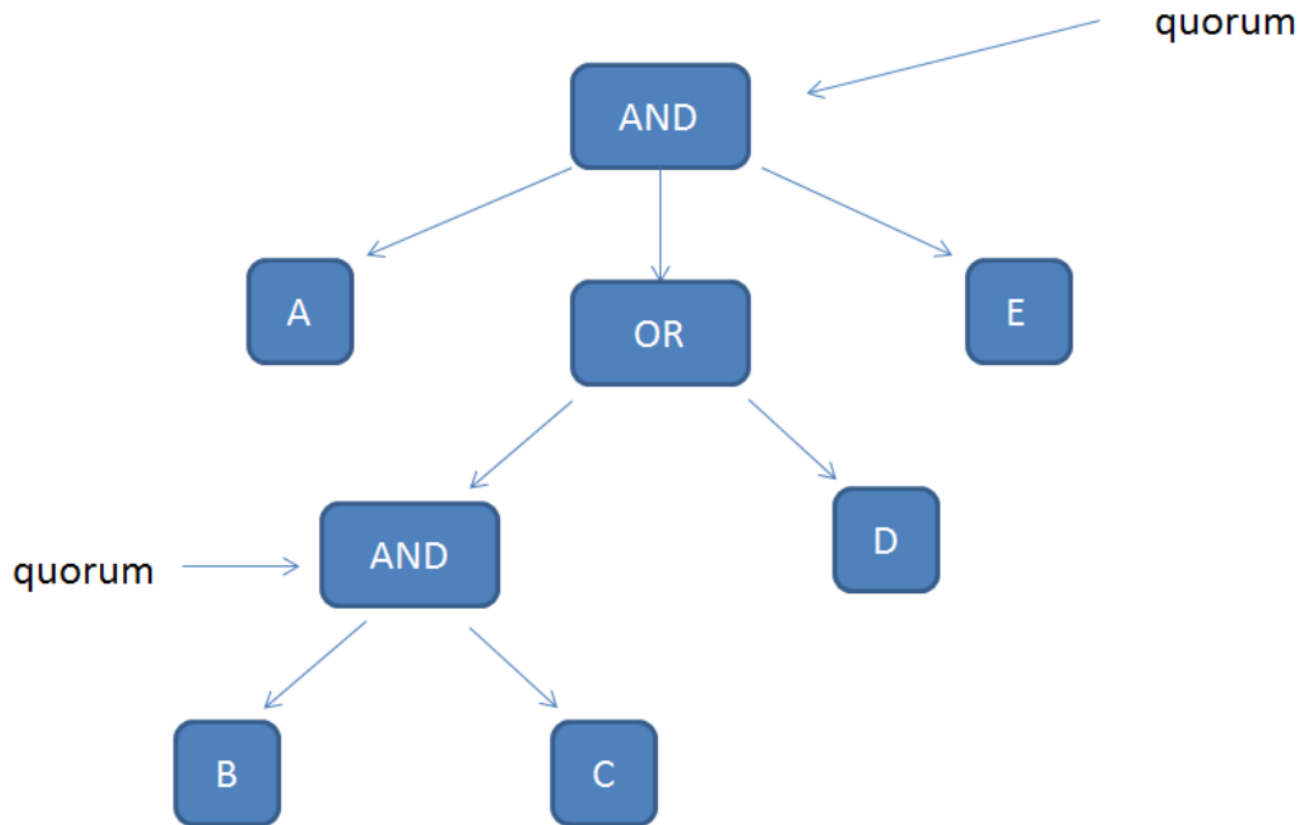
Исполнение запроса



Вопрос:

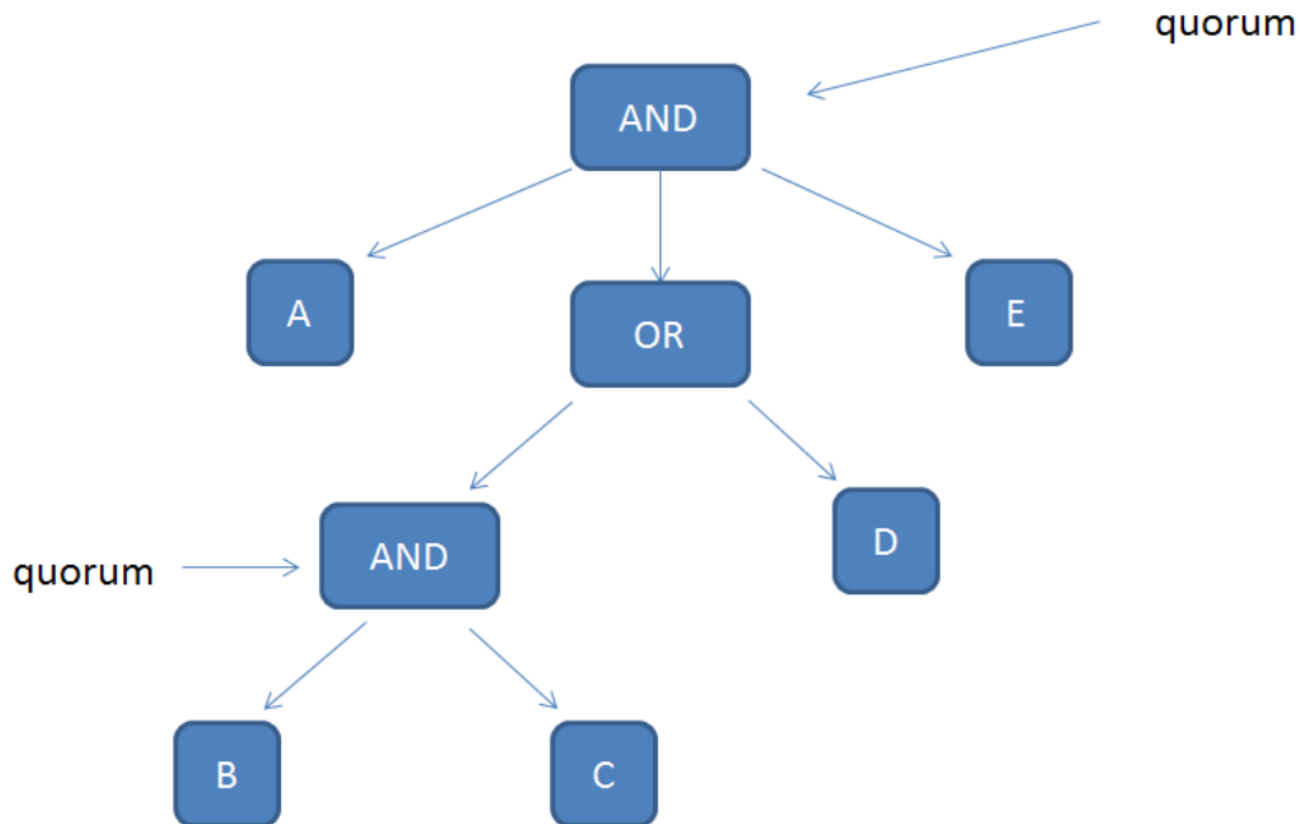
А что делать если нет документов со всеми словами из запроса?

Механизм кворума



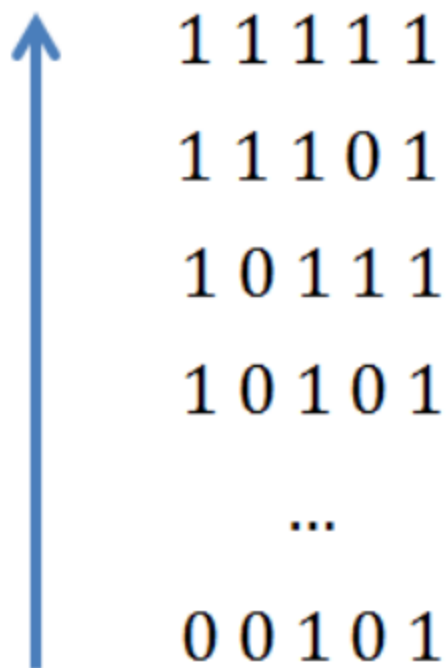
$$quorum = sumIDF * \left(1 - softness^{\frac{1}{\sqrt{n-1}}}\right), \quad softness = 0.05$$

Какие проблемы?



$$quorum = sumIDF * \left(1 - softness^{\frac{1}{\sqrt{n-1}}}\right), \quad softness = 0.05$$

Ранжирование вместо классификации



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!