Больше индексов,
хороших и разных

Вступление

О себе

Почему такая тема и зачем все это?

Демонстрационный пример

О себе

Егор Рогов

Отдел образовательных программ Postgres Professional

- учебные курсы
- перевод документации и глоссарий
- книги и демонстрационная база данных

edu@postgrespro.ru

Зачем что-то, кроме btree?

B-дерево: только сортируемые данные

- числа, строки, даты

Современный мир шире:

- документы, XML, JSON
- геометрические типы, изображения, звуки…

Расширяемость PostgreSQL позволяет добавлять новые типы данных и новые методы доступа для них

Что будет на этом МК

Точки

- SP-GiST и GiST

Полнотекстовый поиск

- GiST и GIN

JSON

- GIN

Демо-пример (оч. сокр.)

Твиттер: твиты по запросу «database»

API для поиска твитов, документ в формате JSON


```
{   "id": 951552165704855552,
    "lang": "en",
    "text": "#tbt 1985 - Before we had a digital database,
             students at used the card catalog to find…
             https://t.co/9GmyiLOEiP",
    "user": {
        "id": 507704387, "name": "MephamHS"
    },
    "entities": {
        "hashtags": [ { "text": "tbt", "indices": [0, 4] } ]
    },
    "created_at": "Thu Jan 11 20:31:29 +0000 2018",
    "coordinates": {
        "type": "Point", "coordinates": [-73.5422, 40.6748]
    }
}
```

Терминология

Метод доступа

Тип данных

Класс операторов

Индекс

Метод доступа

Каркас:

- алгоритмы для работы с поисковой структурой данных
- низкоуровневые детали: эффективная одновременная работа, страничная организация, журнализация и т. п.

Методы доступа могут добавляться как расширения (9.6)

По умолчанию используется метод доступа btree

Примеры:

- hash, btree, gist, spgist, gin, brin

Тип данных

Допустимые значения, операторы

Типы данных могут добавляться

Примеры:

- integer, real, numeric, boolean, date, timestamp
- point, box, circle
- xml, jsonb
- tsvector, tsquery

Класс операторов

Набор операторов (и вспомогательных функций), который используется *методом доступа* для работы с конкретным *типом данных*

- для одного метода доступа можно определить несколько классов операторов для одного типа данных (один из них — по умолчанию)
- вспомогательные функции реализуют API метода доступа

Примеры:

- int2_ops, int4_ops, int8_ops (*семейство* integer_ops)
- point_ops, box_ops

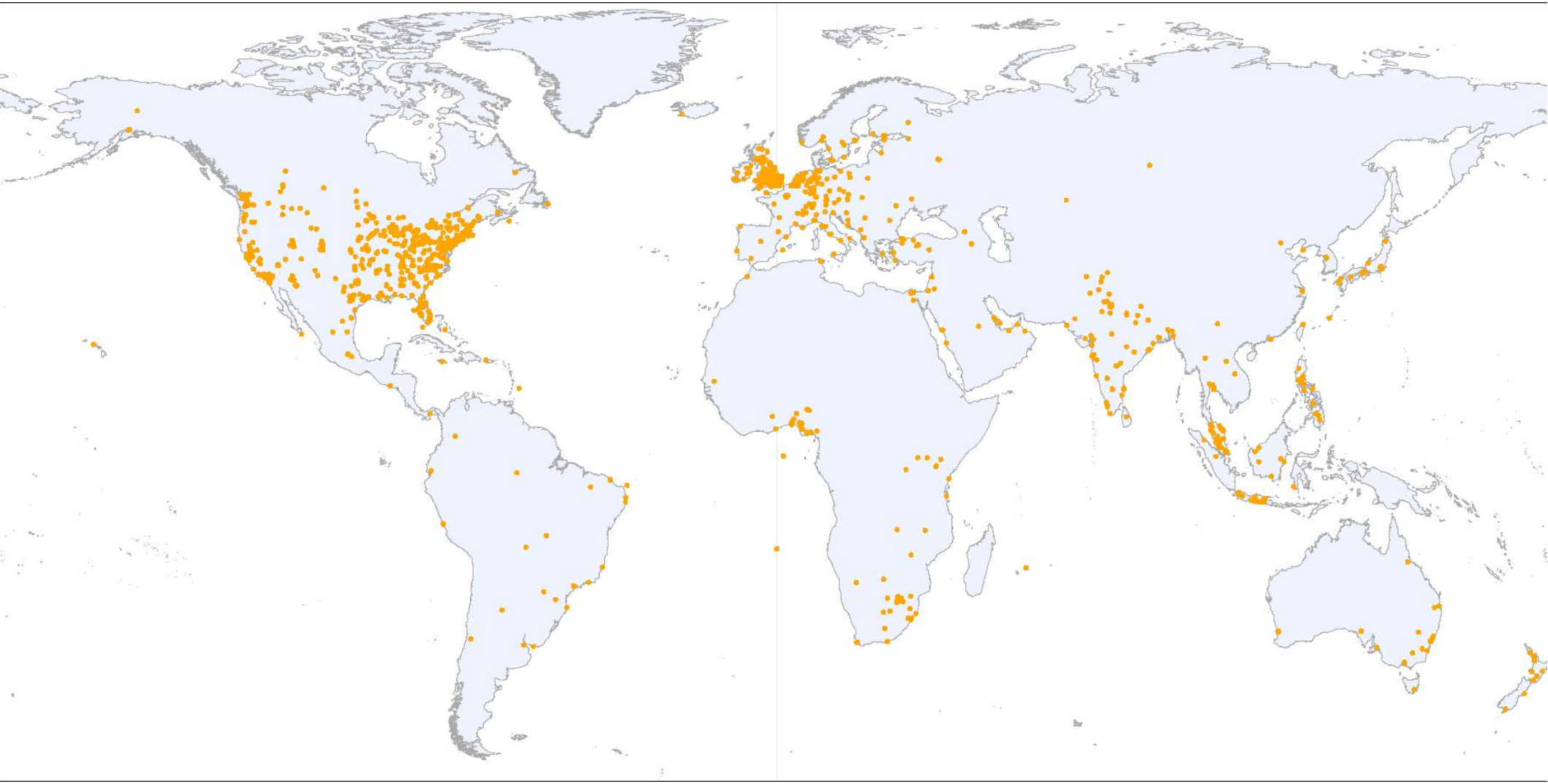
Индекс

Объект базы данных

- создан на конкретной таблице на основе метода доступа
- содержит один или несколько столбцов (или выражений)
- для каждого столбца используется свой класс операторов

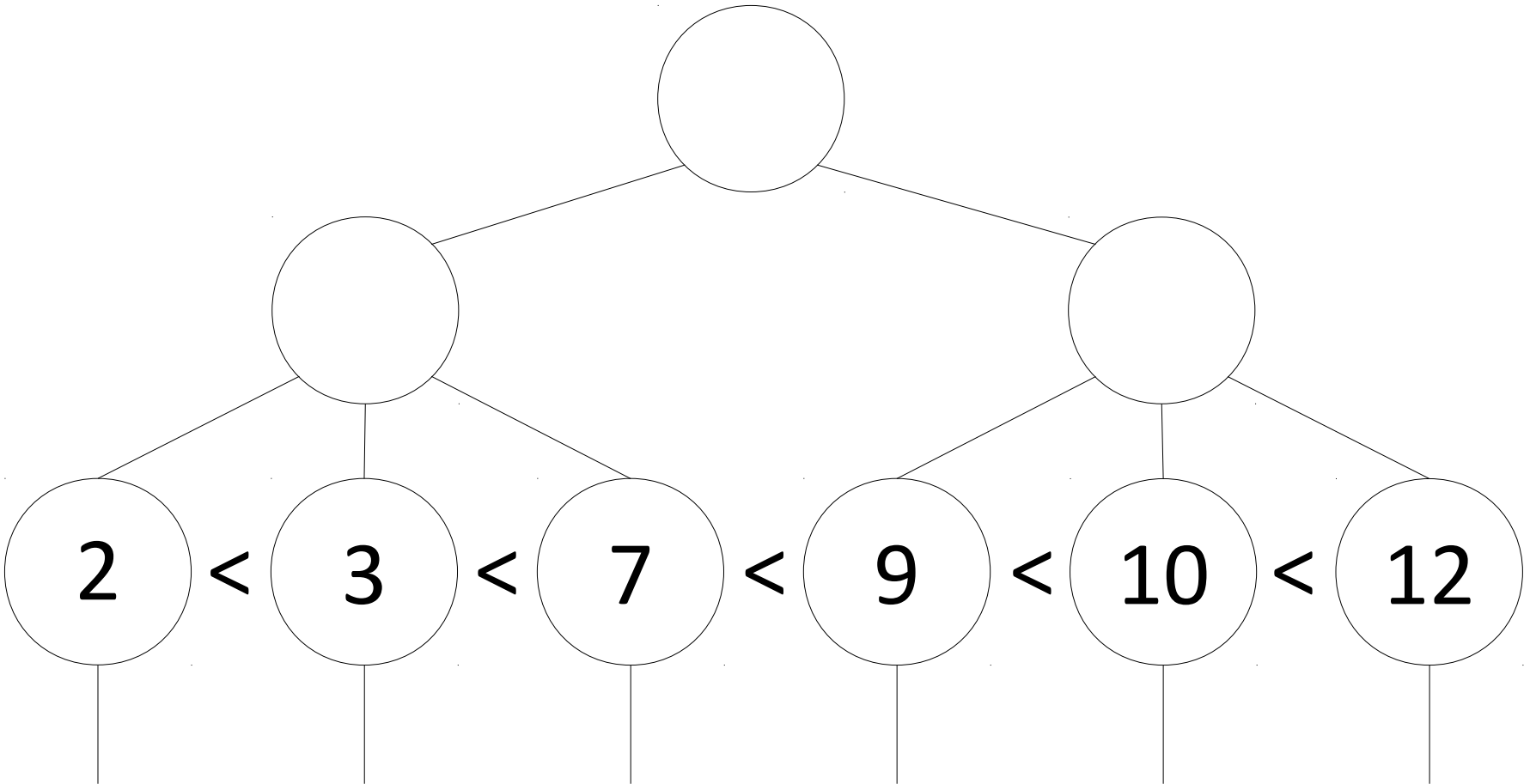
GiST и SP-GiST

География твитов



B-дерево

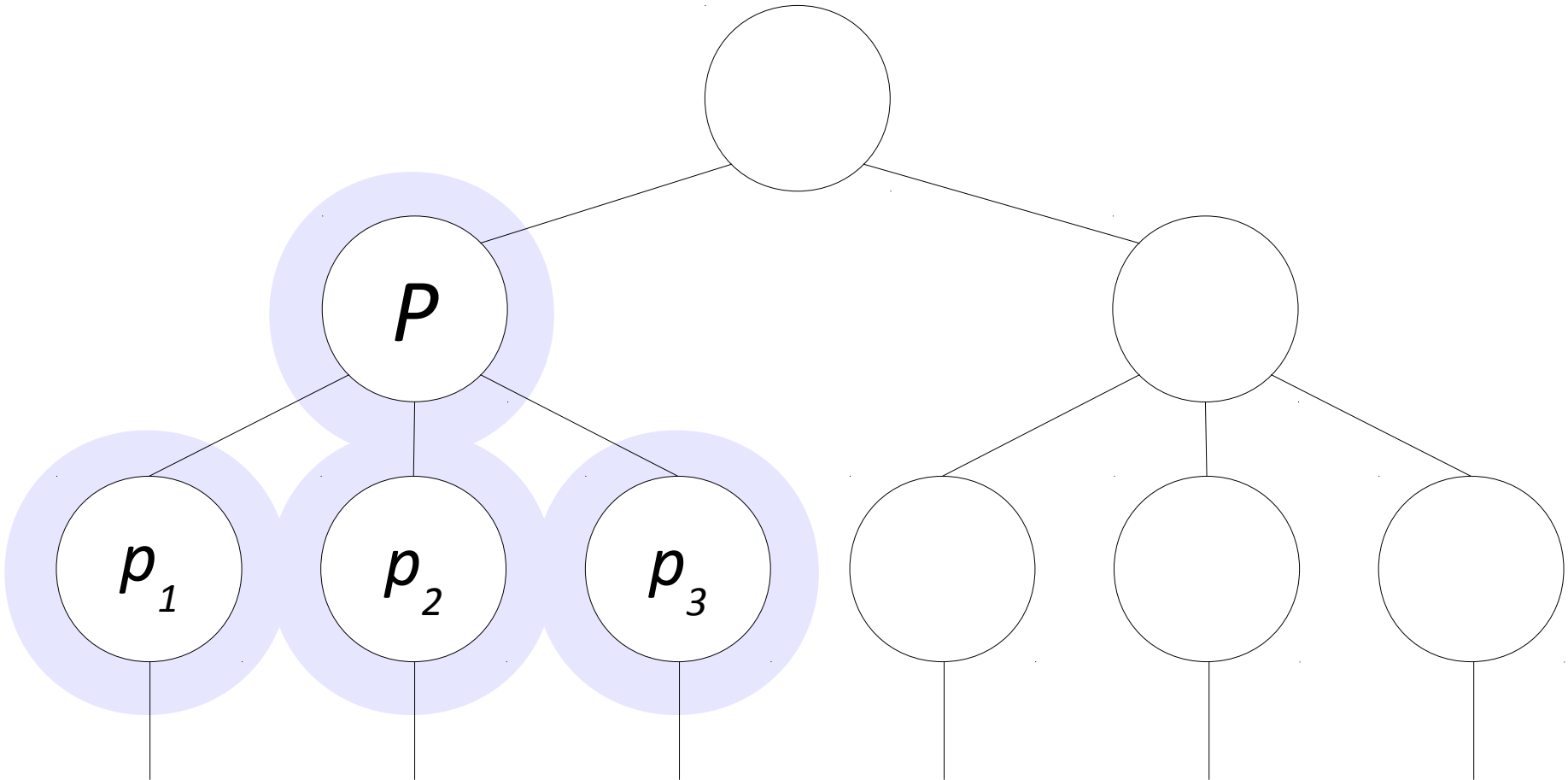
Сбалансированное дерево, значения упорядочены



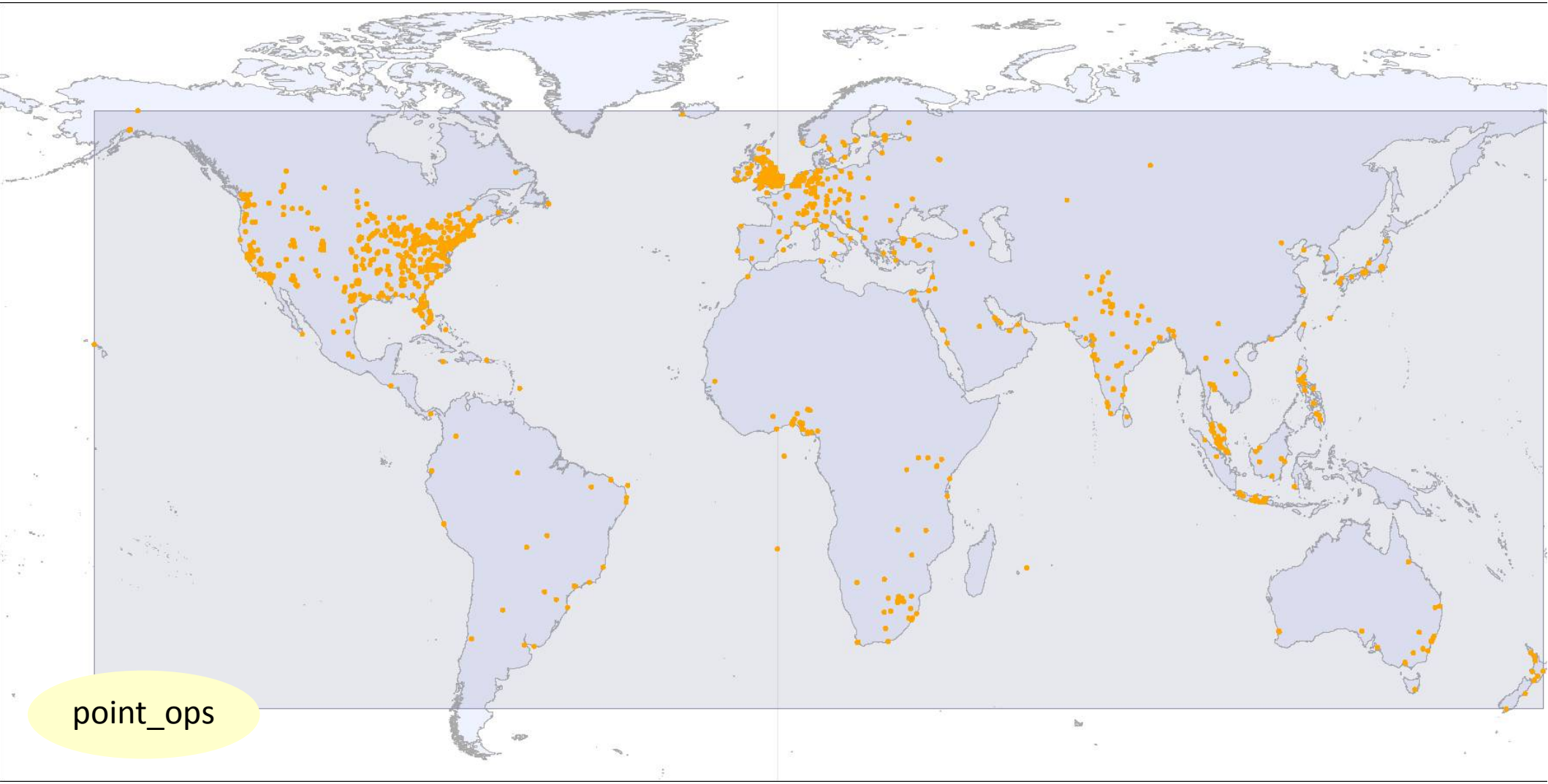
GiST

Сбалансированное дерево, общий предикат для поддерева

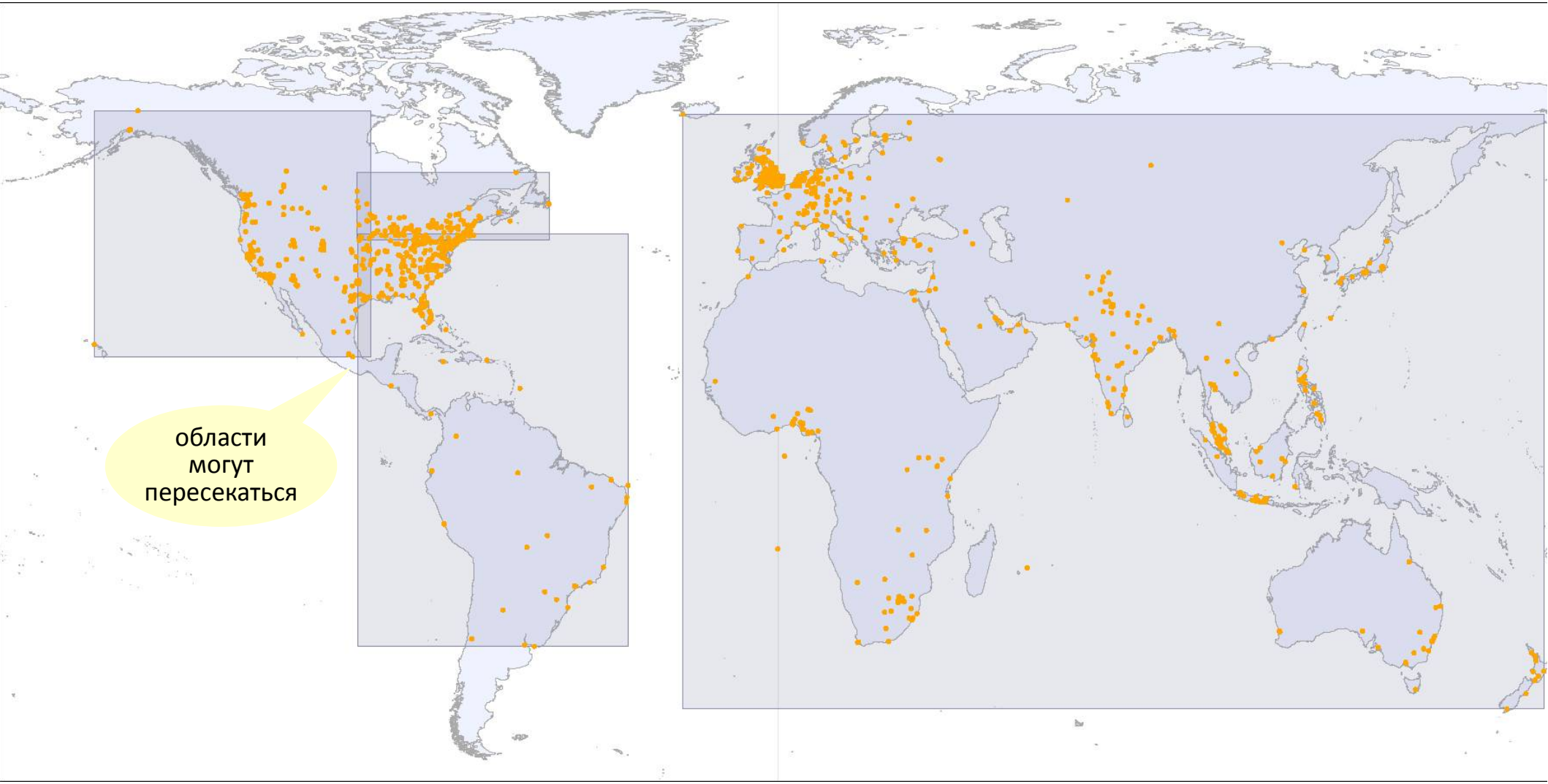
$P(p_1) = P(p_2) = P(p_3) = \text{true}$



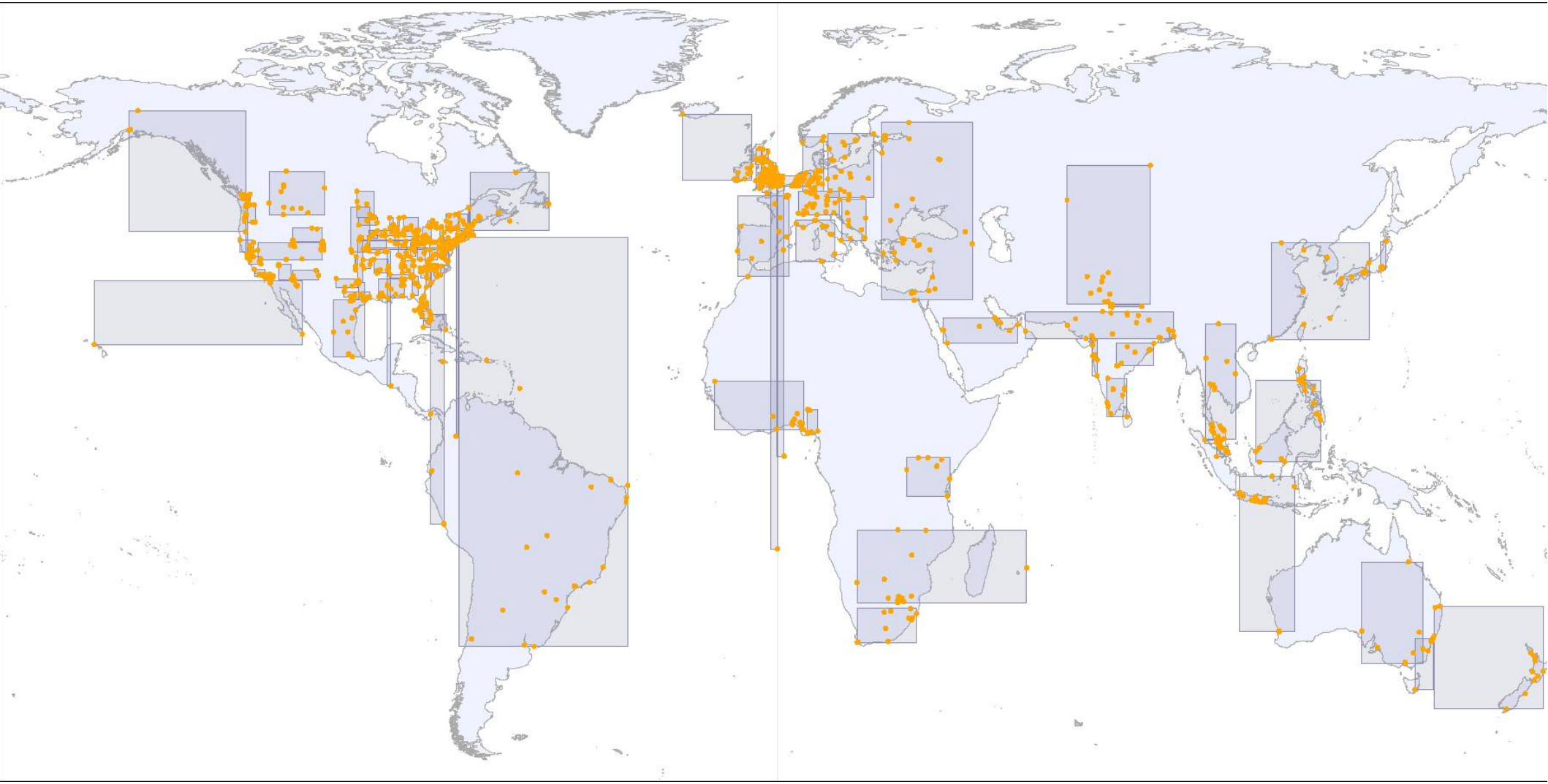
R-дерево, 1 уровень



R-дерево, 2 уровень



R-дерево, 3 уровень

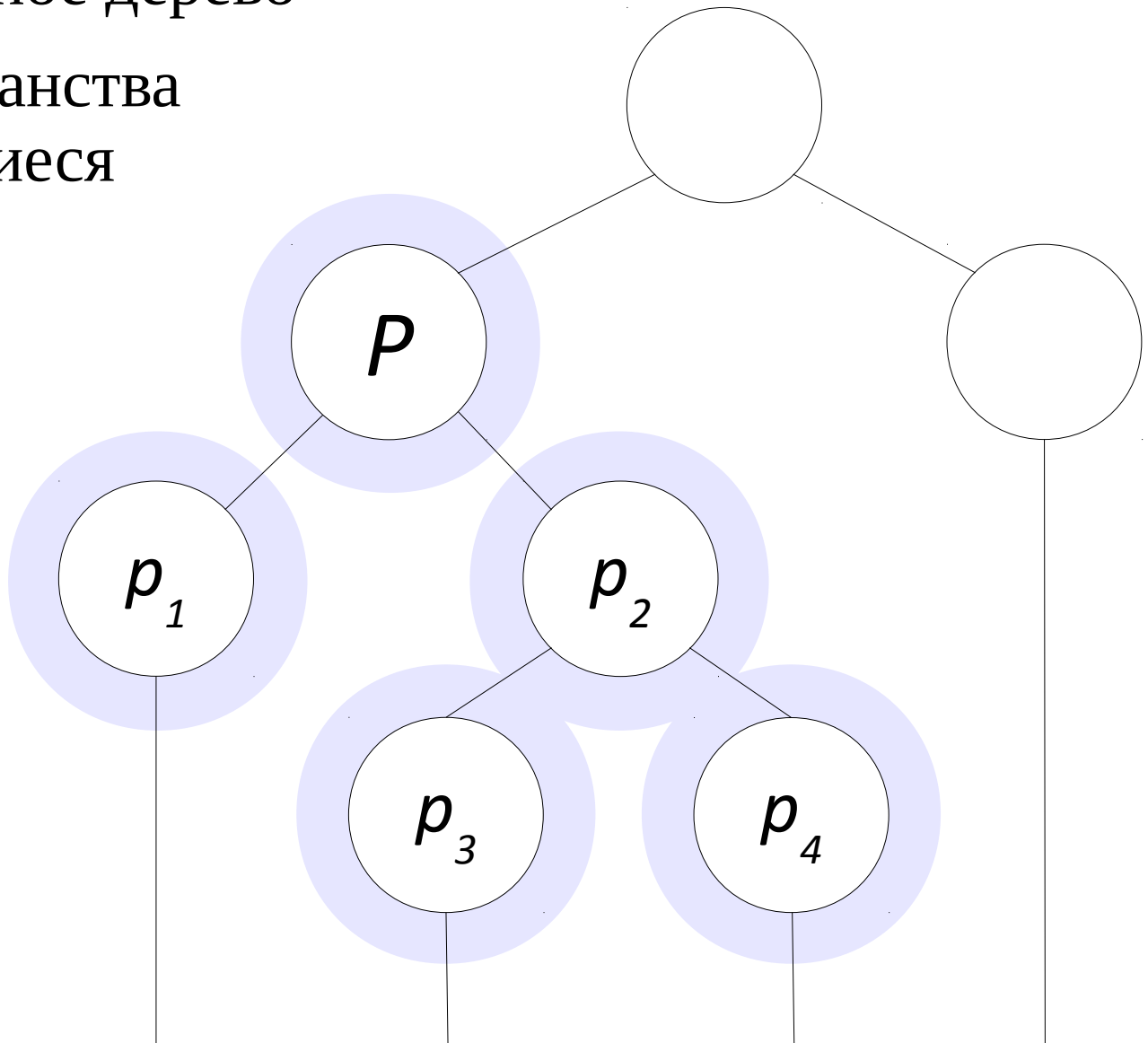


SP-GiST

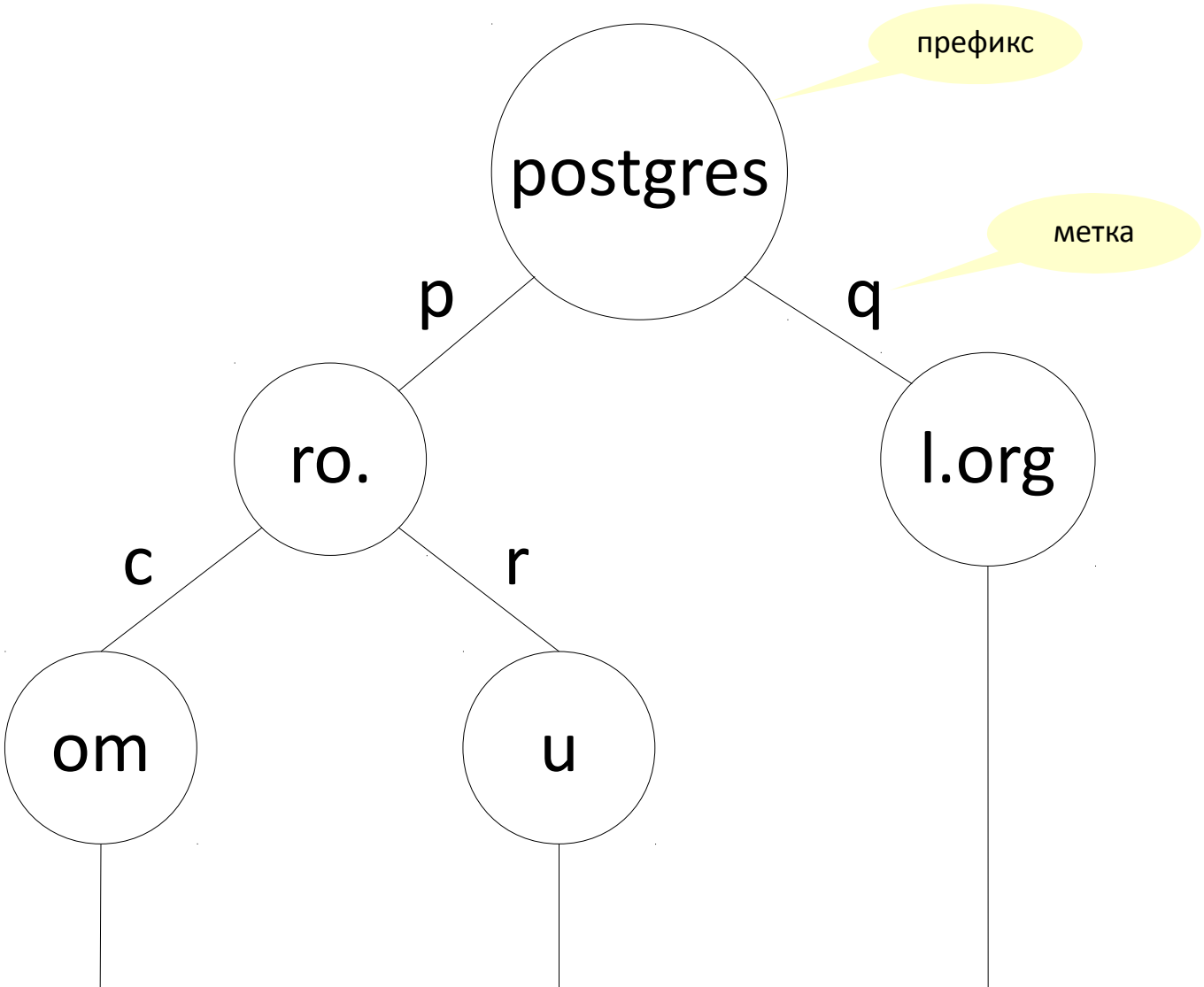
Несбалансированное дерево

Разбиение пространства на непересекающиеся области

Общий префикс

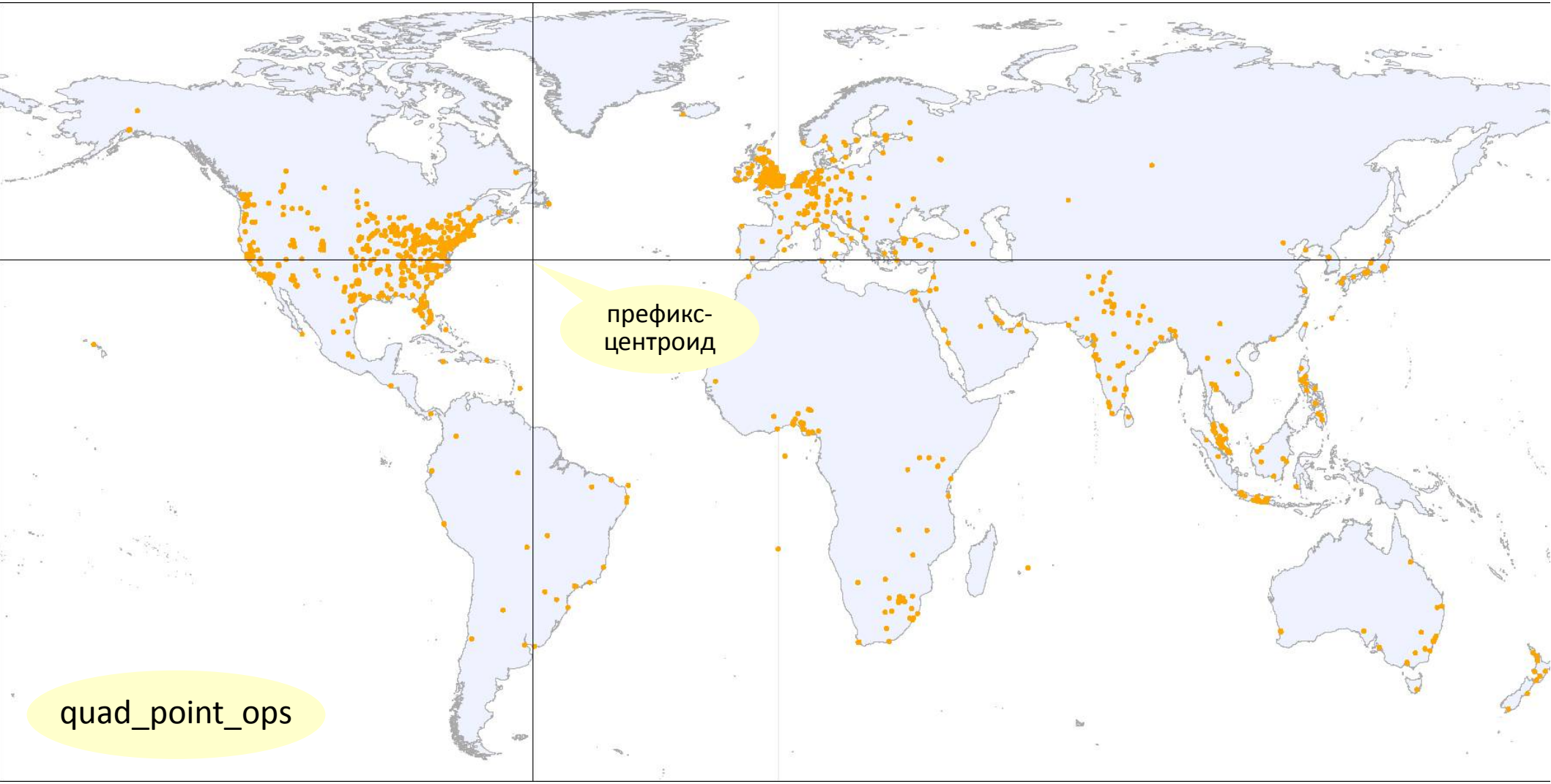


Префиксное дерево

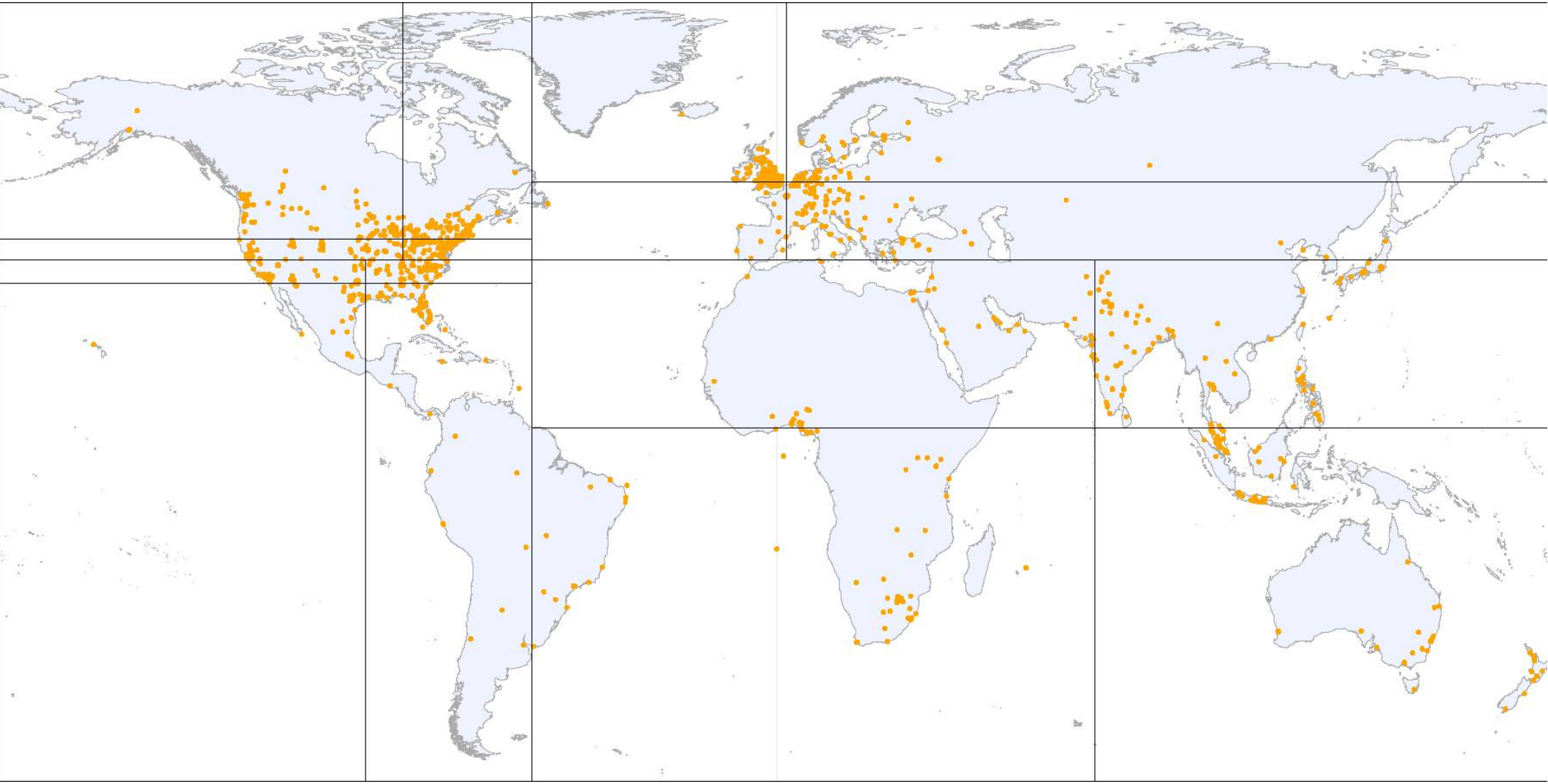


text_ops

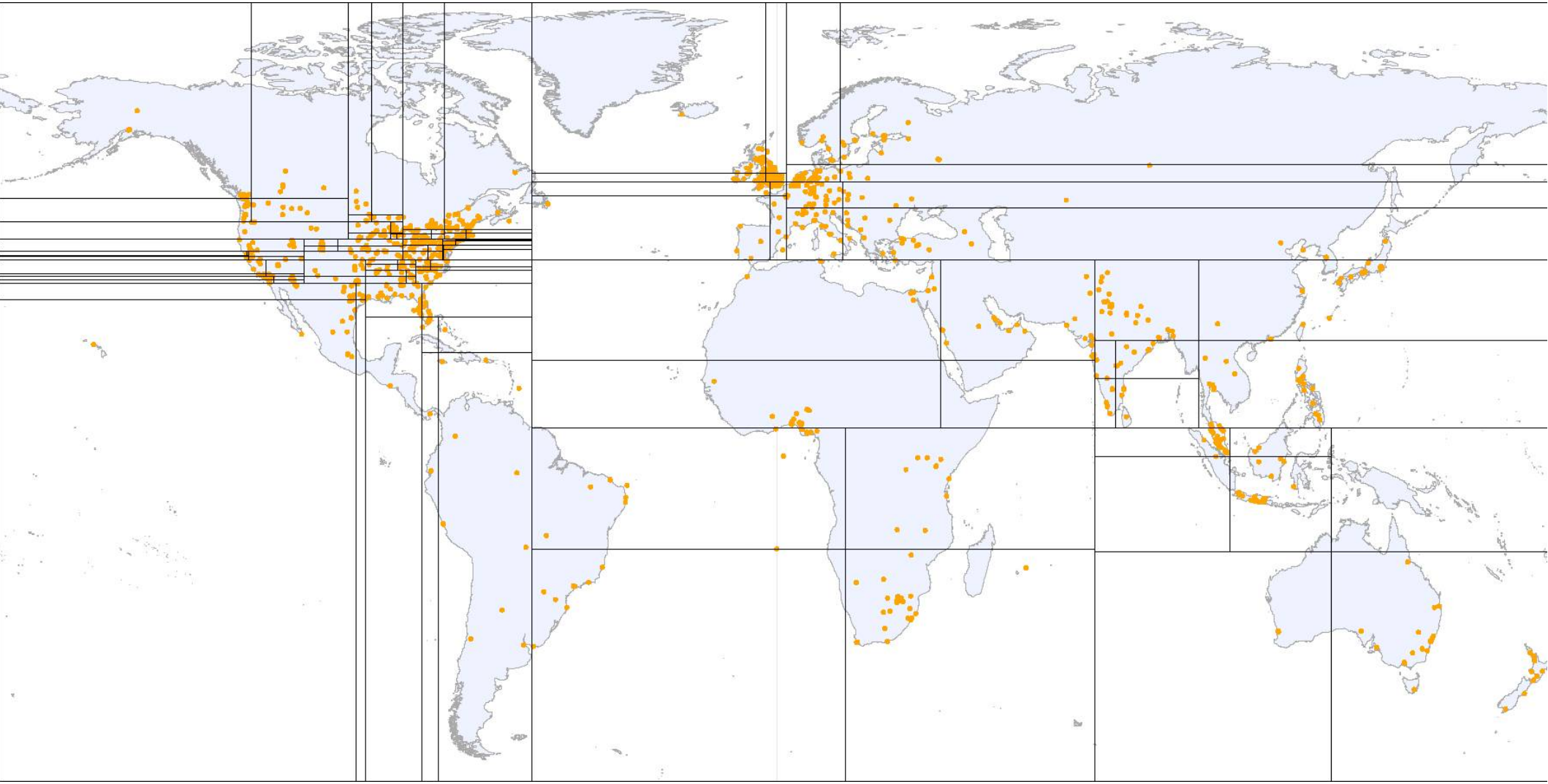
Дерево квадрантов, 1 ур.



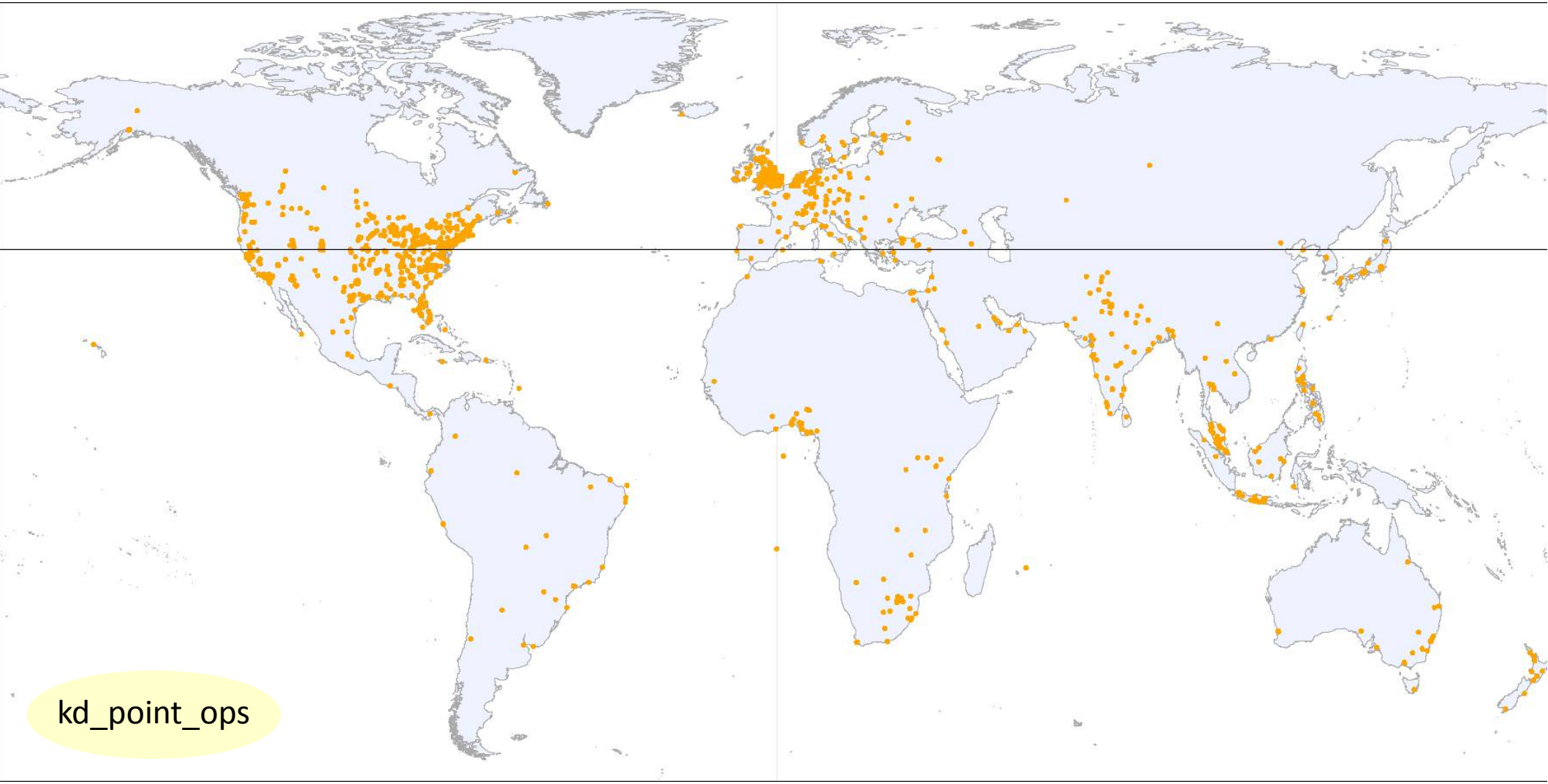
Дерево квадрантов, 2 ур.



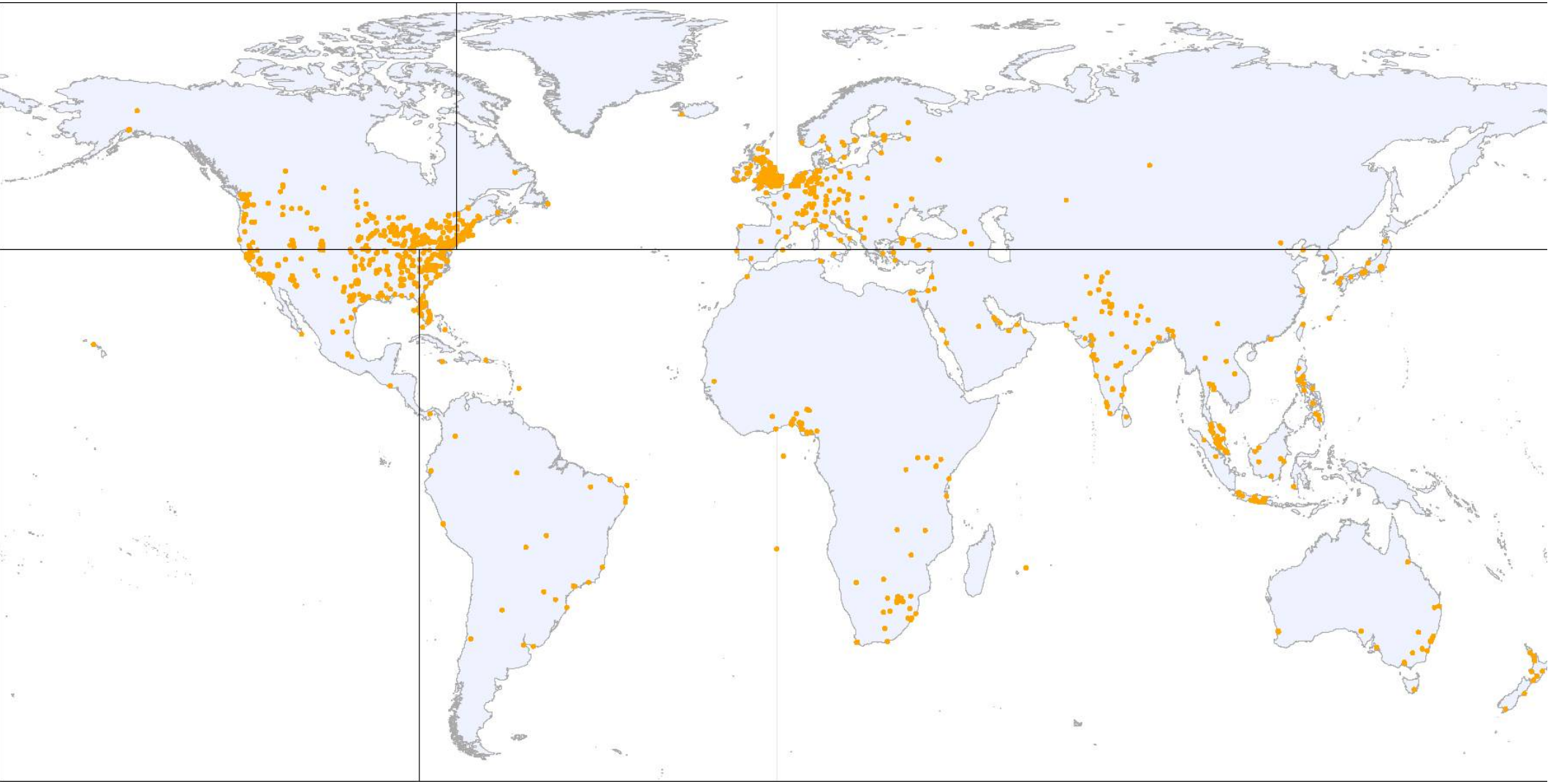
Дерево квадрантов, 3 ур.



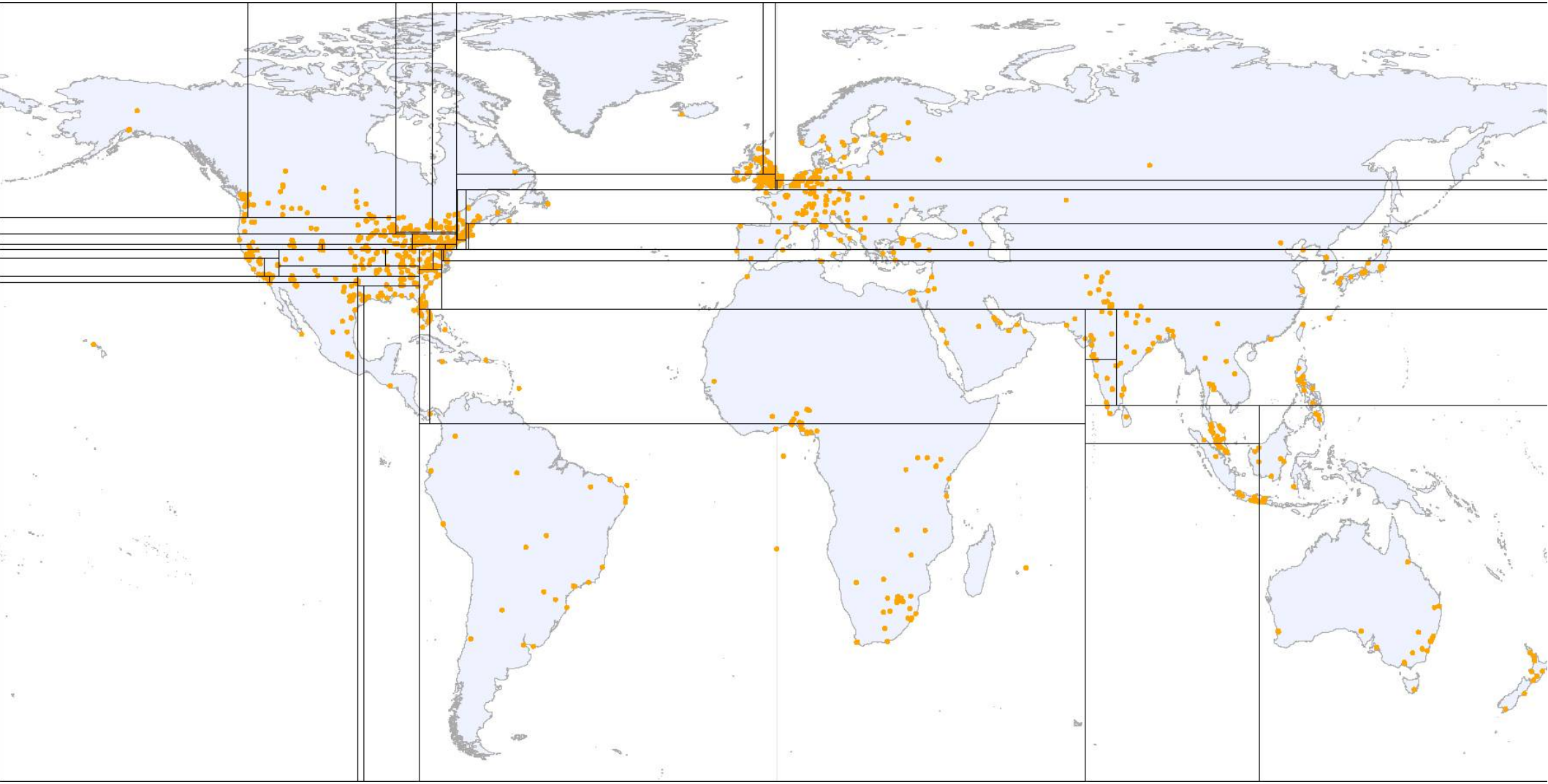
k-D-дерево, 1 уровень



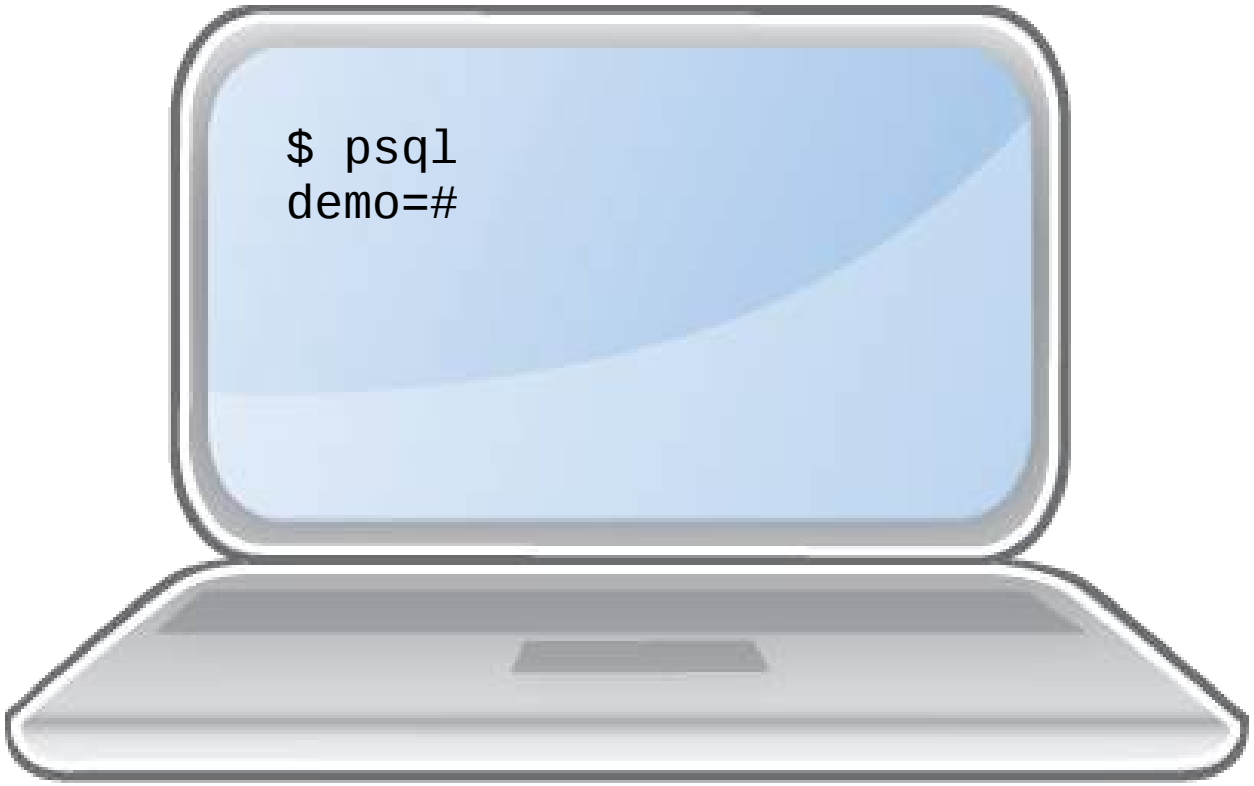
k-D-дерево, 2 уровень



k-D-дерево, 3 уровень



Первая демонстрация



GiST и GIN

Начнем с массивов

Составной тип, содержащий элементы другого типа

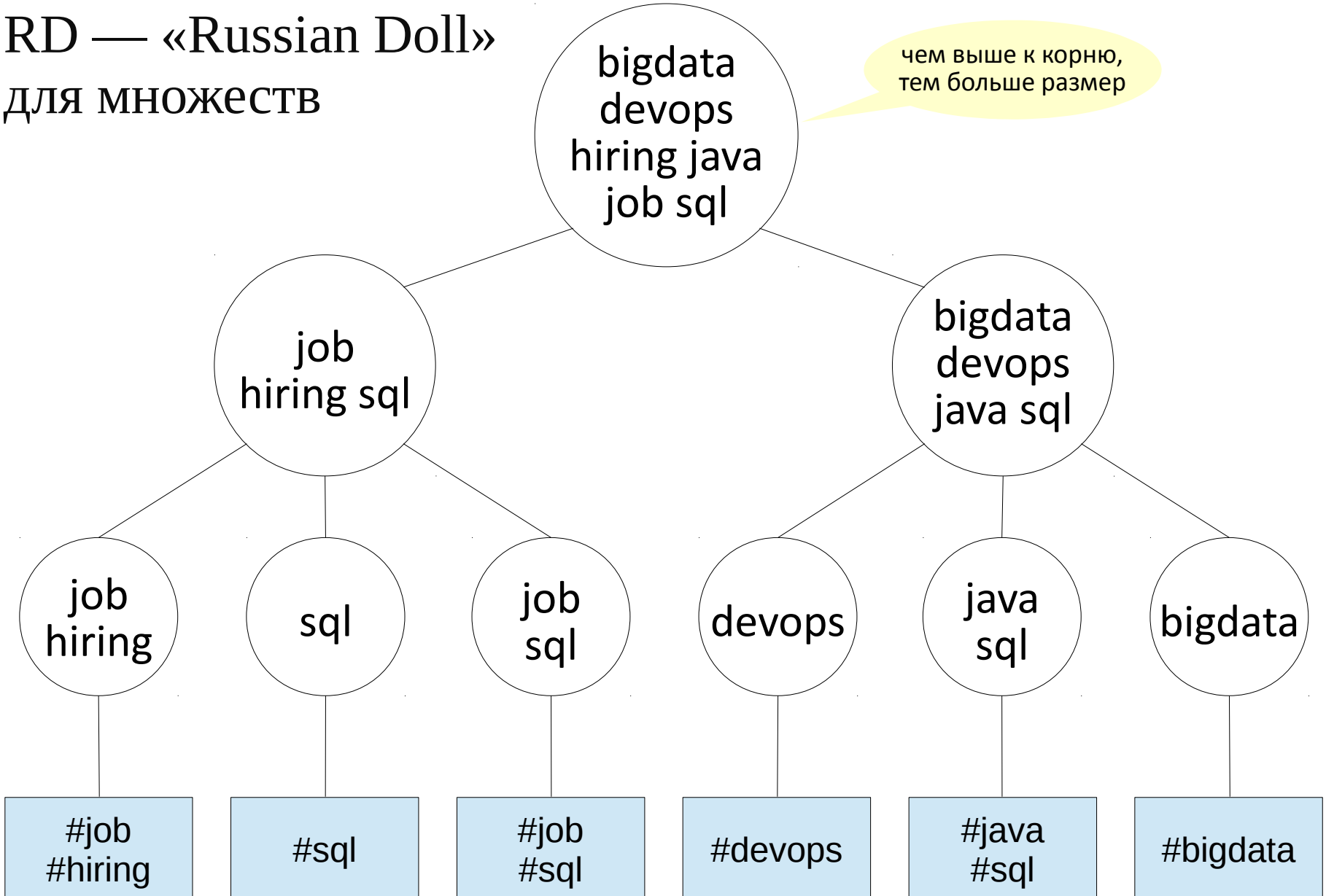
Операции:

- вхождение одного массива в другой
- пересечение с массивом

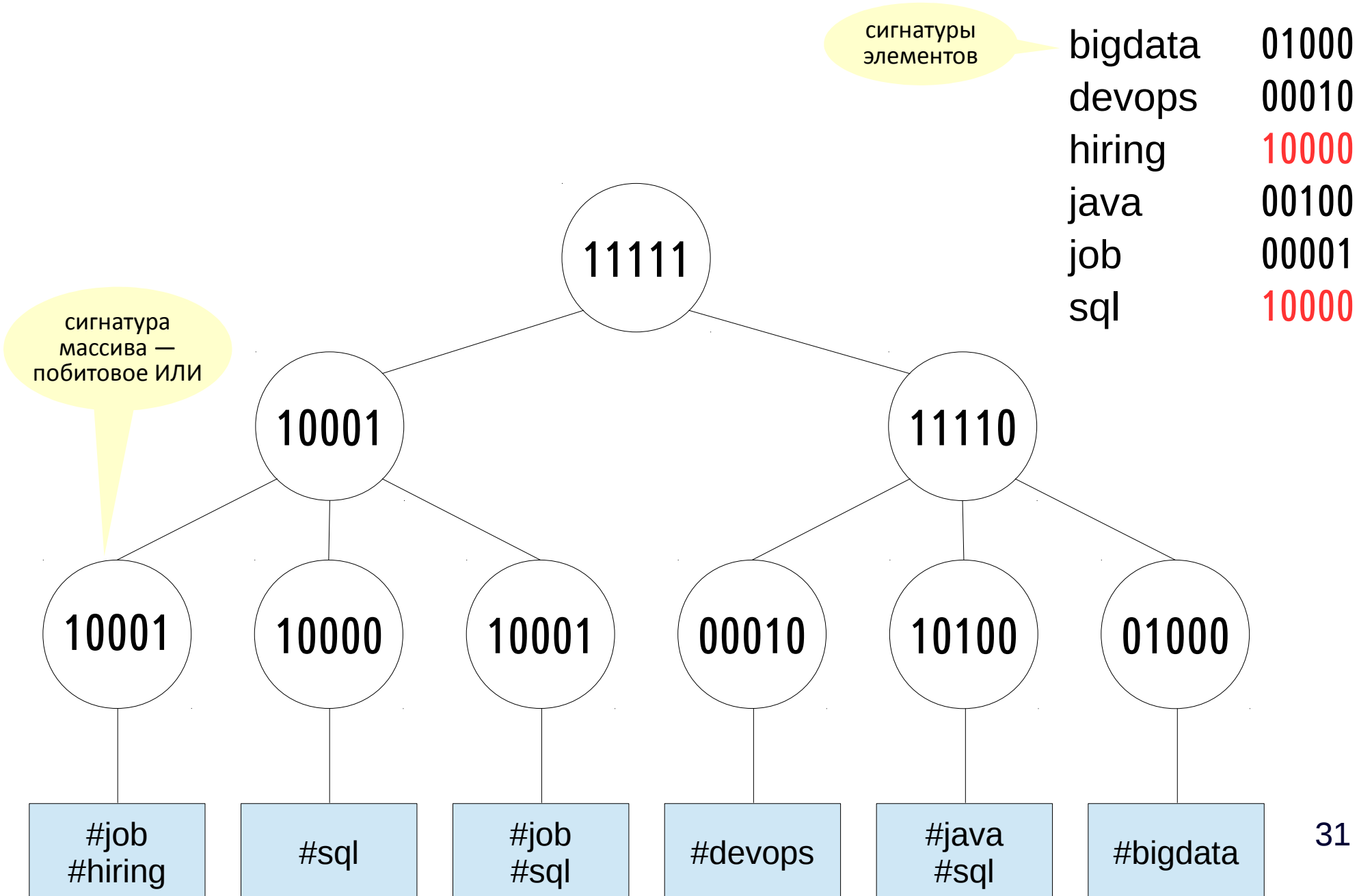
Надо индексировать массив, учитывая его составляющие

GiST (RD-дерево)

RD — «Russian Doll» для множеств



GiST (сигнатурное дерево)

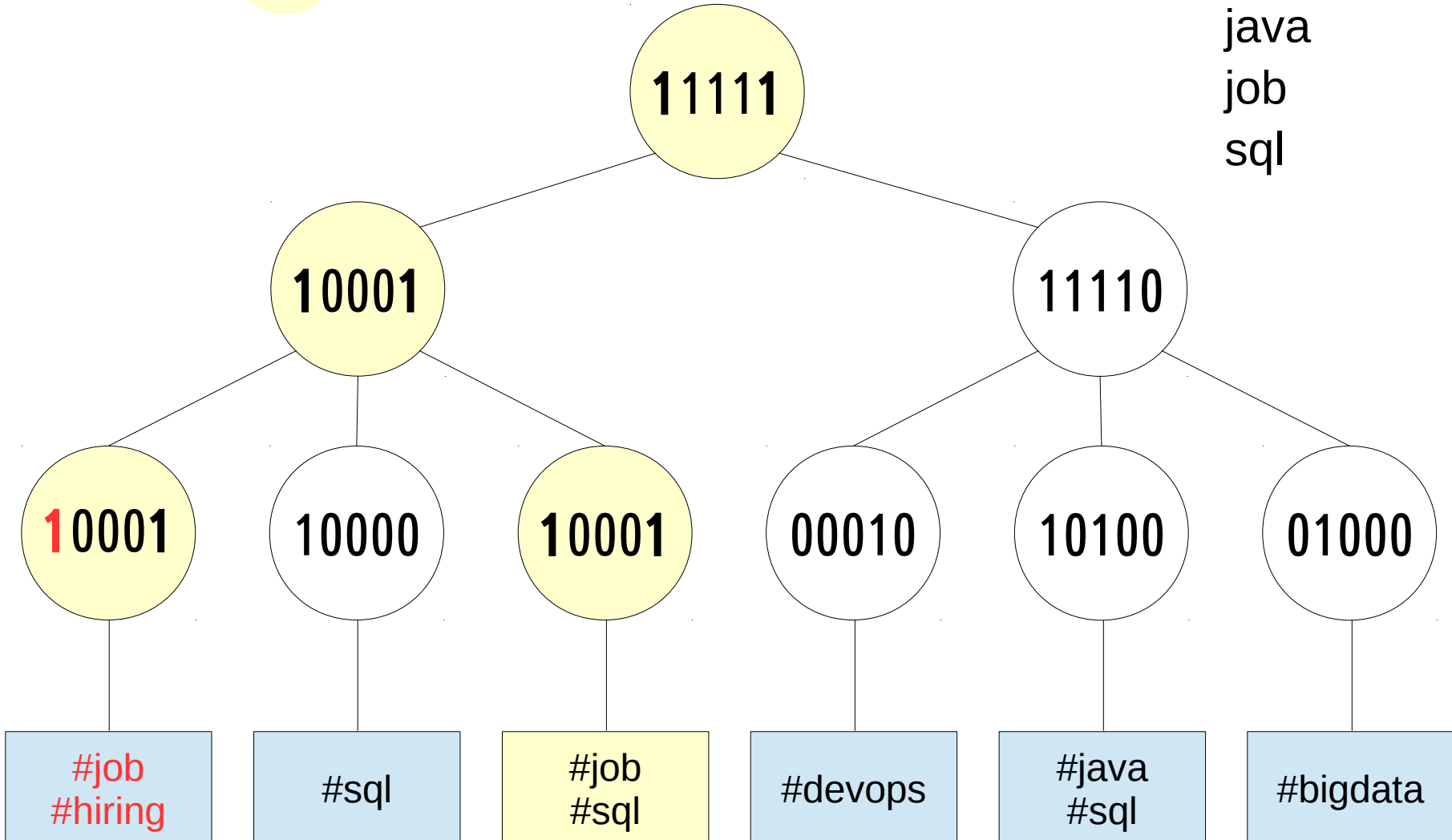


Выполнение запроса

```
hashtags @> '{job,sql}'
```

⊇

bigdata	01000
devops	00010
hiring	10000
java	00100
job	00001
sql	10000



GiST (сигнатурное дерево)

Преимущества

- фиксированный размер битовой строки
- универсальность: можно индексировать что угодно, если построить хорошую сигнатуру

Недостатки

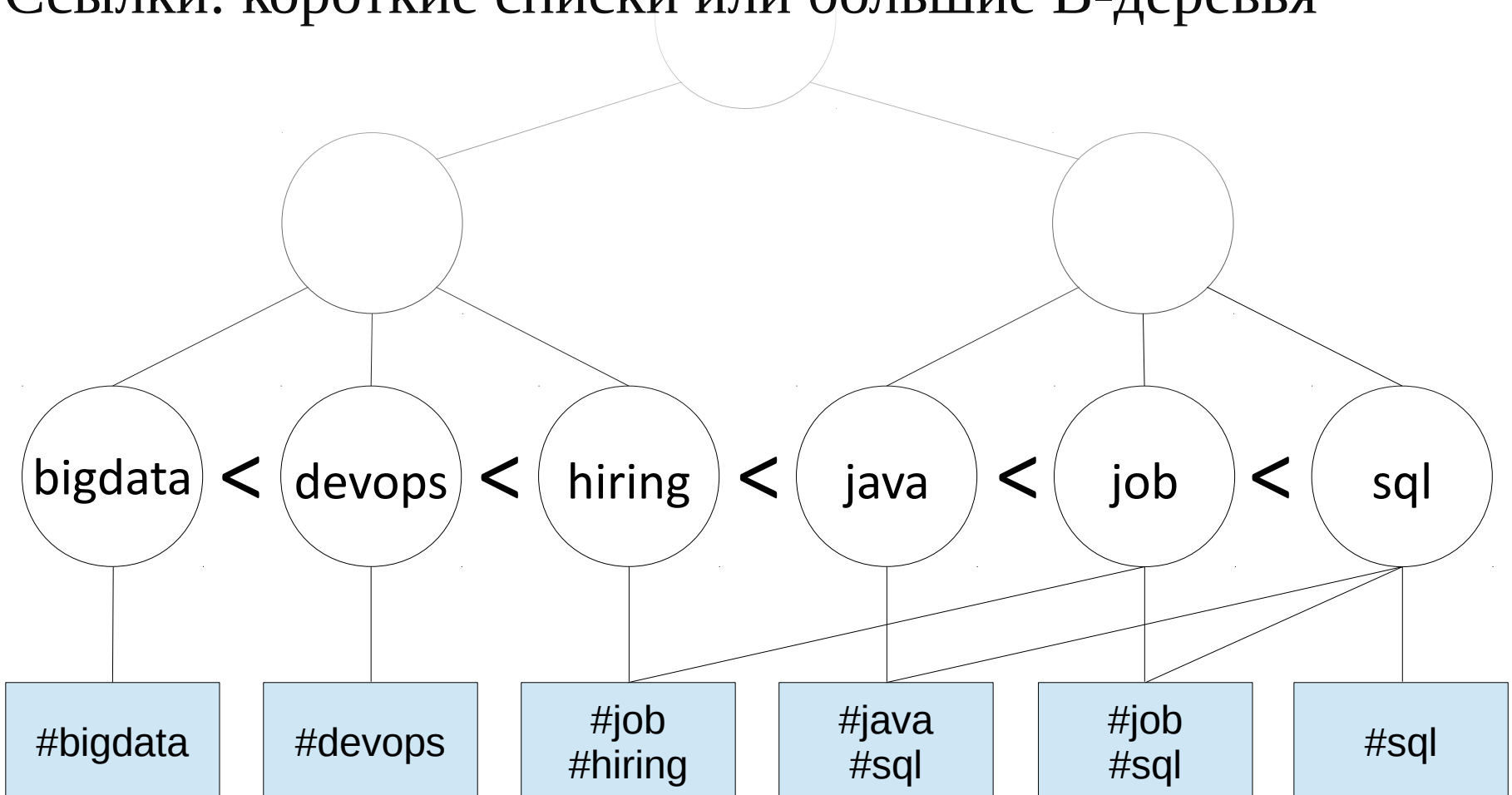
- неточный поиск, результат надо перепроверять *почти всегда*
- как следствие — уменьшение эффективности поиска

GIN

B-дерево элементов

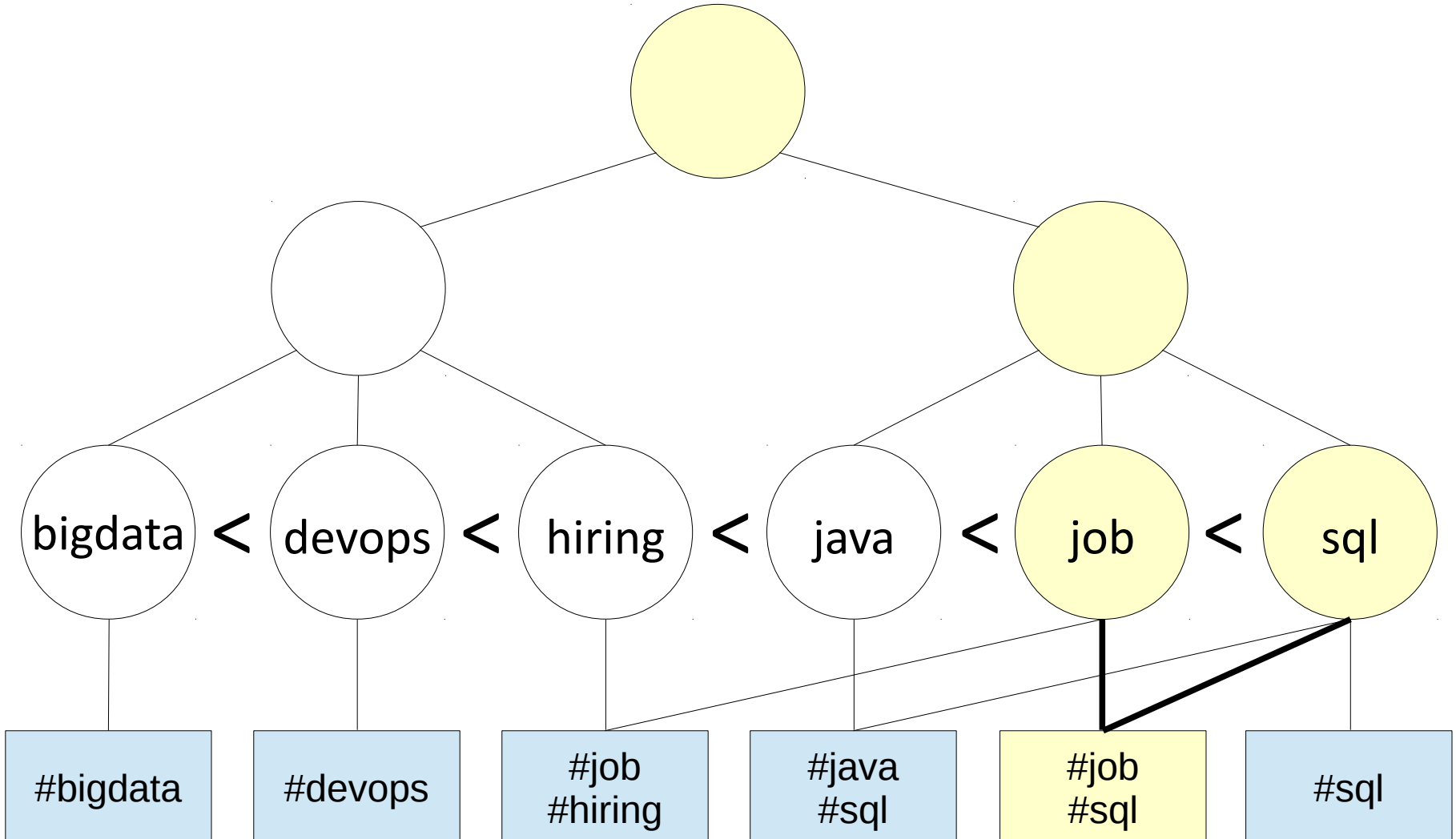
листовые узлы ссылаются на все строки, в которых встречается элемент

Ссылки: короткие списки или большие B-деревья



Выполнение запроса

```
hashtags @> '{job,sql}'
```



GIN

Преимущества

- относительно точный и быстрый поиск
- компактное представление (элементы не дублируются)

Недостатки

- дорогое обновление: изменение одного массива требует изменения всех его элементов в индексе

Сложности обновления

GiST

- предикаты во внутренних узлах только расширяются
- сжатие предиката — только во время разделения страницы, либо при полной перестройке индекса

GIN

- само по себе медленно (но есть возможность пакетного обновления)
- элементы никогда не удаляются из индекса

Полнотекстовый поиск

Массив → набор лексем (тип tsvector)

Предварительно надо превратить документ в набор лексем

- анализатор: разобрать текст, выделить слова
- словари: нормализовать слова — удалить стоп-слова, привести словоформы к одному виду, учесть синонимы и т. п.

Операция @@

- сопоставление tsvector поисковому запросу (тип tsquery)

Варианты индексирования

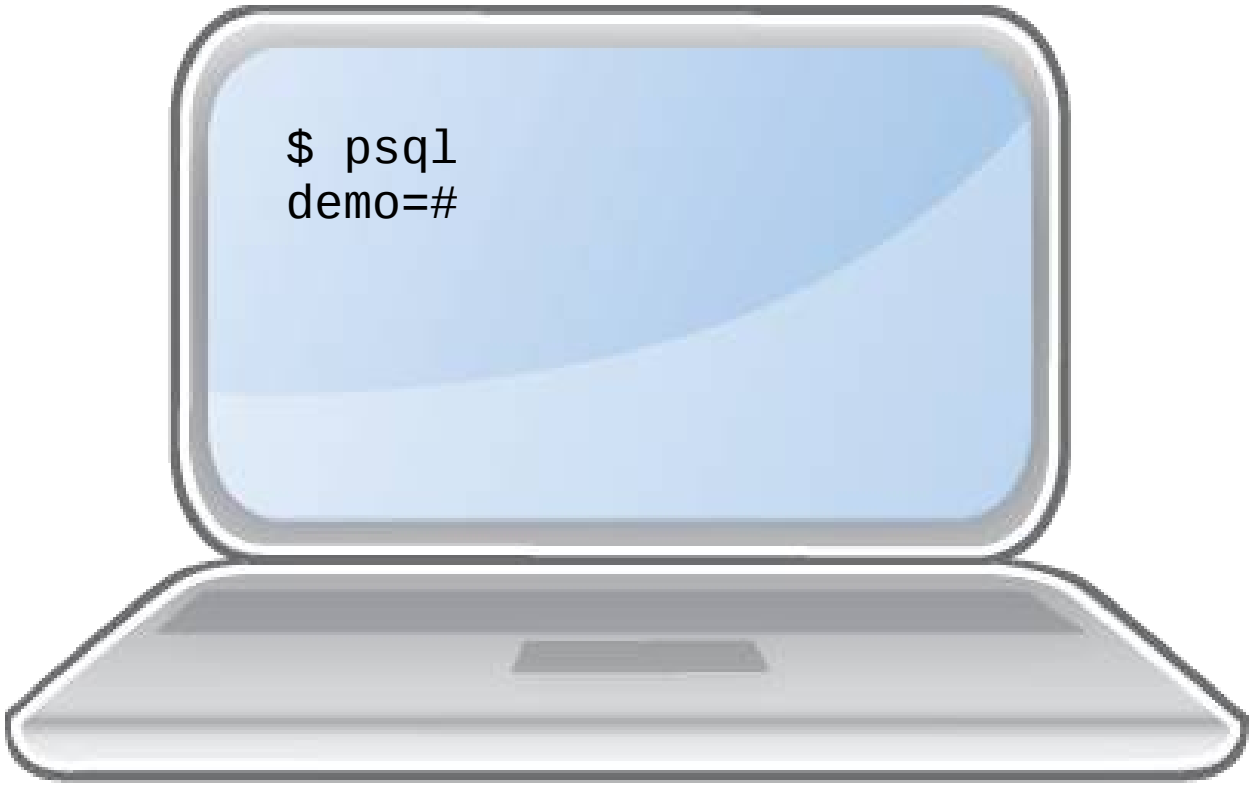
Индекс по отдельному полю типа tsvector

- поле необходимо обновлять триггером

Индекс по выражению to_tsvector(*документ*)

- не требуется дополнительное место для хранения tsvector
- снижение эффективности при необходимости перепроверки по tsvector

Вторая демонстрация



GIN и... GIN

Типы json и jsonb

Два типа данных: json...

- текстовое представление JSON
- каждый раз требуется разбор, минимум доступных операций

...и jsonb

- эффективное внутреннее представление
- операции с индексной поддержкой
- а также jsquery и в недалеком будущем SQL/JSON

Операции без индекса

Получение части документа (как JSON или как текст)

```
tweet ->  'lang'
tweet ->> 'lang'
```

```
    {  "lang": "en",
       "user": {
           "id": 507704387, "name": "MephamHS"
       }
    }
```

```
tweet #>  '{user,name}'
tweet #>> '{user,name}'
```

```
    {  "lang": "en",
       "user": {
           "id": 507704387, "name": "MephamHS"
       }
    }
```

Индексируемые операции

Наличие ключа (или элемента массива) на *верхнем* уровне

```
tweet ? 'lang'
```

```
{  "lang": "en",
   "user": {
       "id": 507704387, "name": "MephamHS"
   }
}
```

```
(tweet -> 'user') ? 'name'
```

нужен индекс по выражению

```
{  "lang": "en",
   "user": {
       "id": 507704387, "name": "MephamHS"
   }
}
```

Индексируемые операции

Вхождение одного документа в другой

```
tweet @> '{"lang": "en"}'::jsonb
```

```
{  "lang": "en",
   "user": {
       "id": 507704387, "name": "MephamHS"
   }
}
```

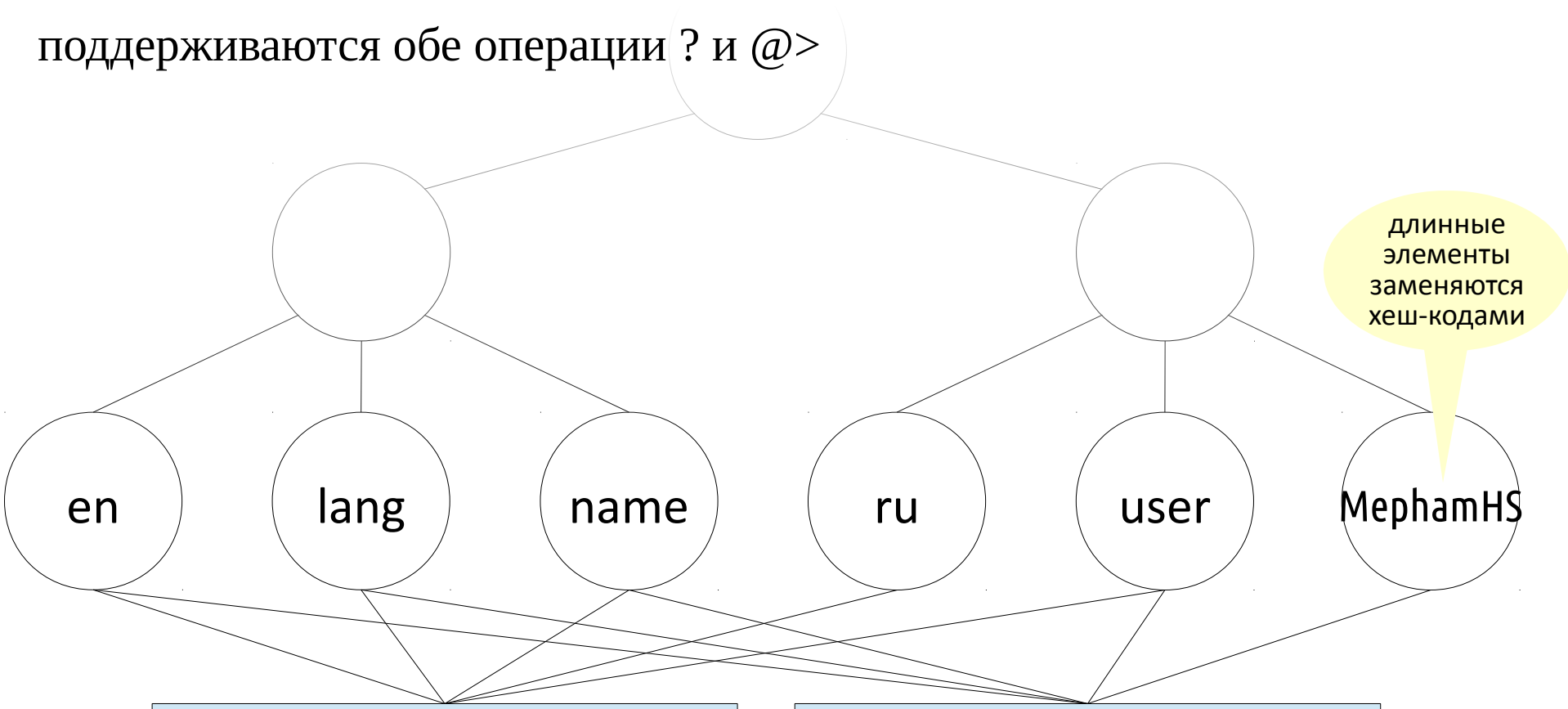
```
tweet @> '{"user": {"name": "MemphamHS"}}'::jsonb
```

```
{  "lang": "en",
   "user": {
       "id": 507704387, "name": "MephamHS"
   }
}
```

Класс jsonb_ops

Индексируются все ключи, значения и элементы массивов

поддерживаются обе операции ? и @>

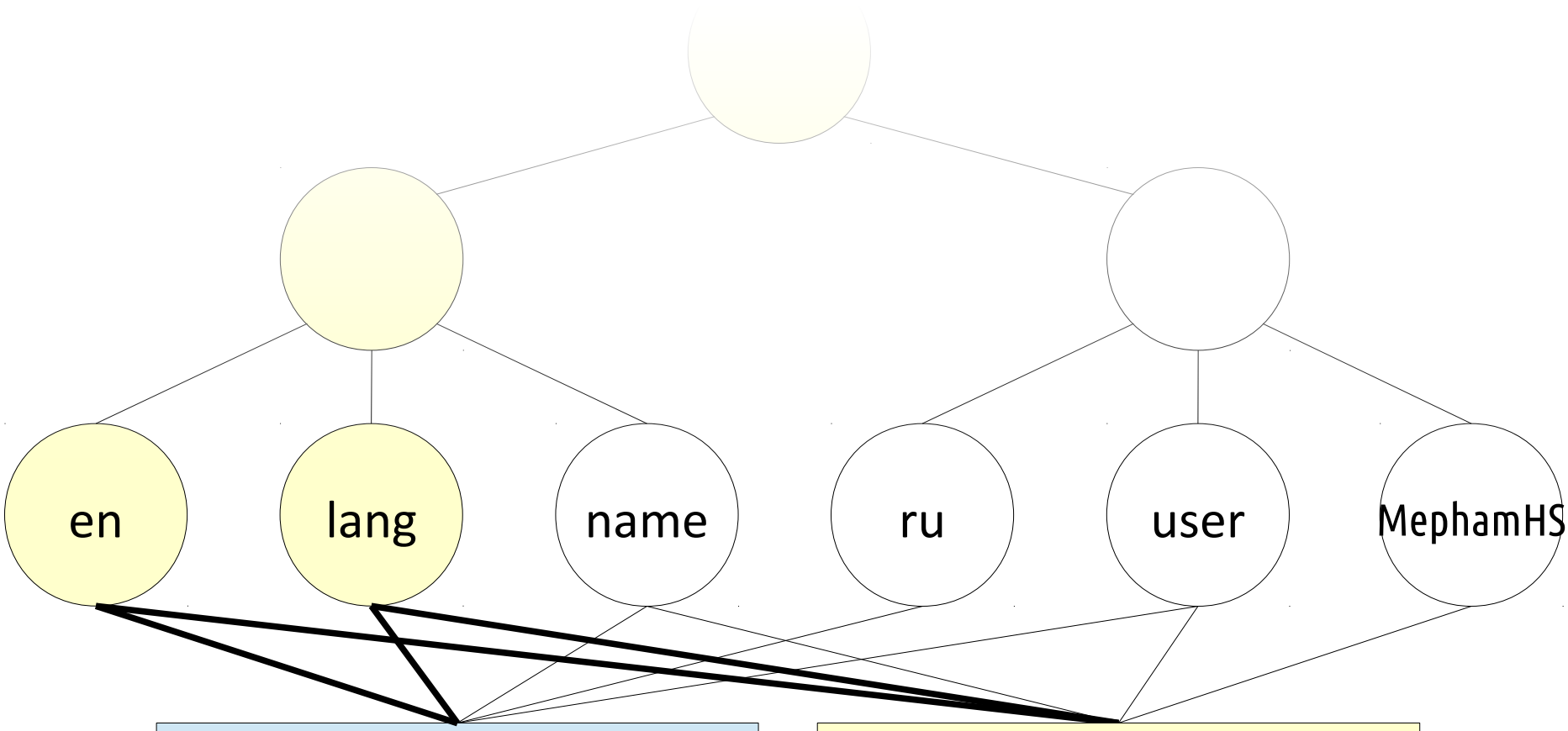



```
{  "lang": "ru",
   "user": {
       "name": "en"
   }
}
```

```
{  "lang": "en",
   "user": {
       "name": "MephamHS"
   }
}
```

Выполнение запроса

```
tweet @> '{"lang": "en"}'::jsonb
```



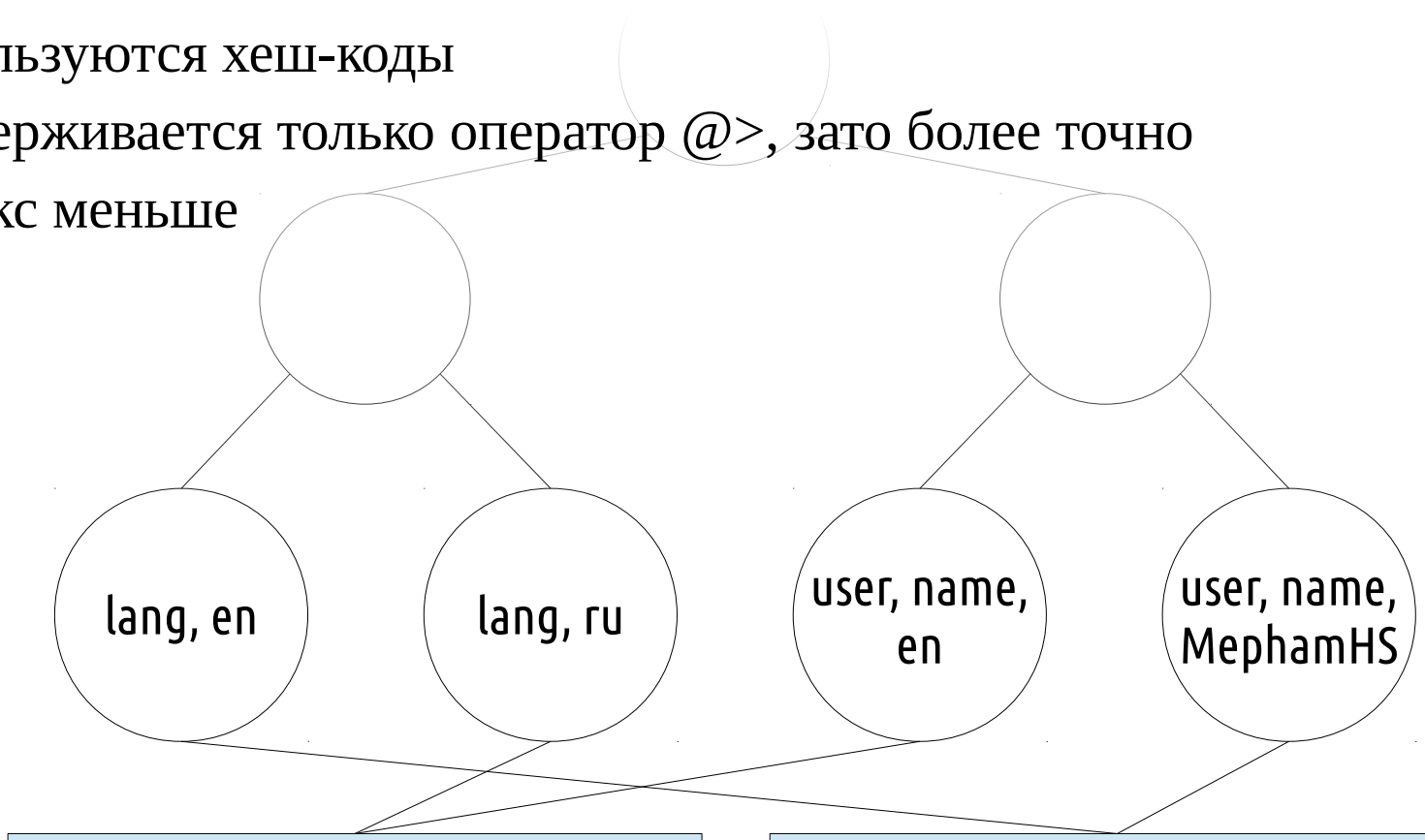

```
{   "lang": "ru",
    "user": {
        "name": "en"
    }
}
```

```
{   "lang": "en",
    "user": {
        "name": "MephamHS"
    }
}
```

Класс jsonb_path_ops

Индексируются значения и ключи, приводящие к ним

- используются хеш-коды
- поддерживается только оператор @>, зато более точно
- индекс меньше

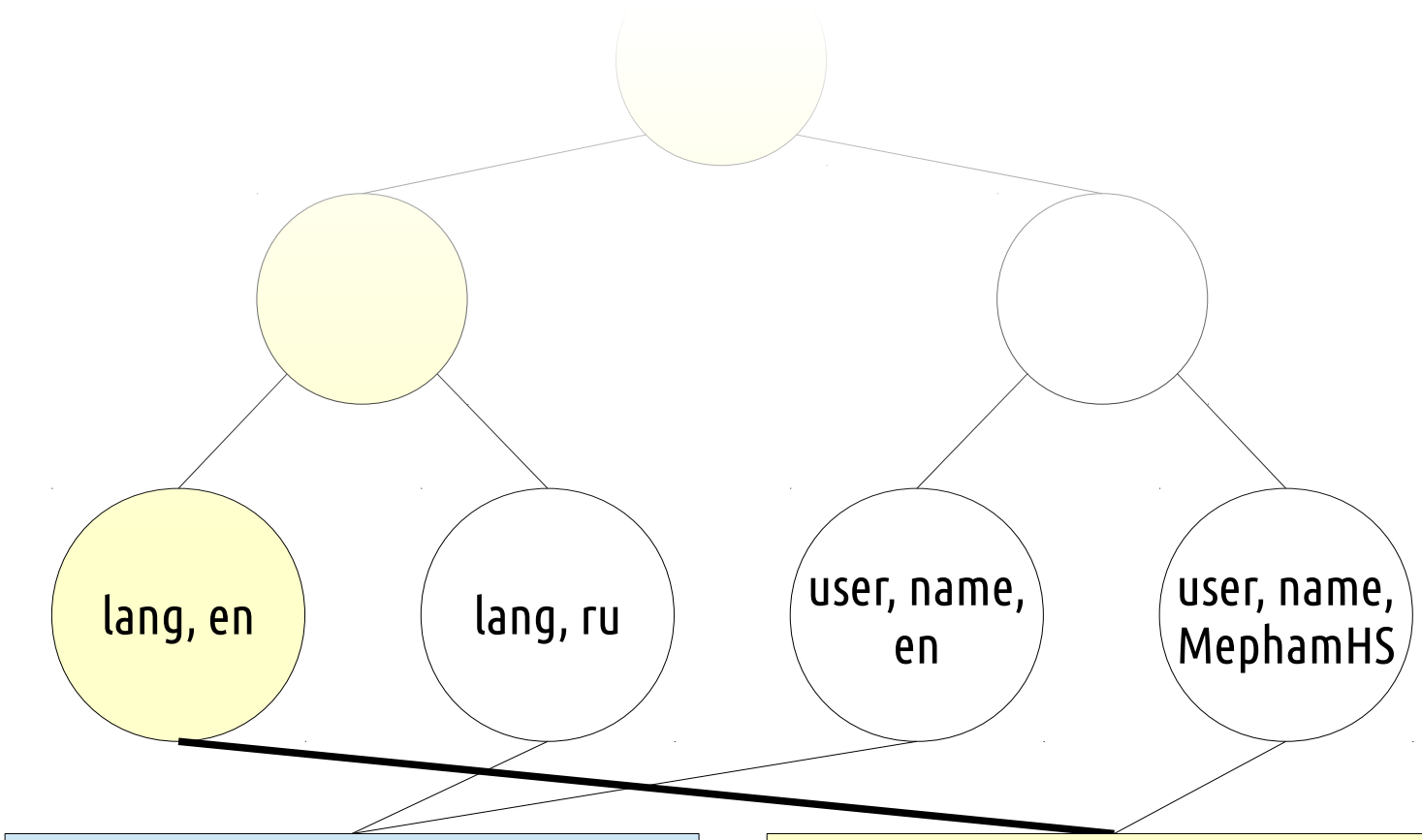



```
{  "lang": "ru",
   "user": {
       "name": "en"
   }
}
```

```
{  "lang": "en",
   "user": {
       "name": "MephamHS"
   }
}
```

Выполнение запроса

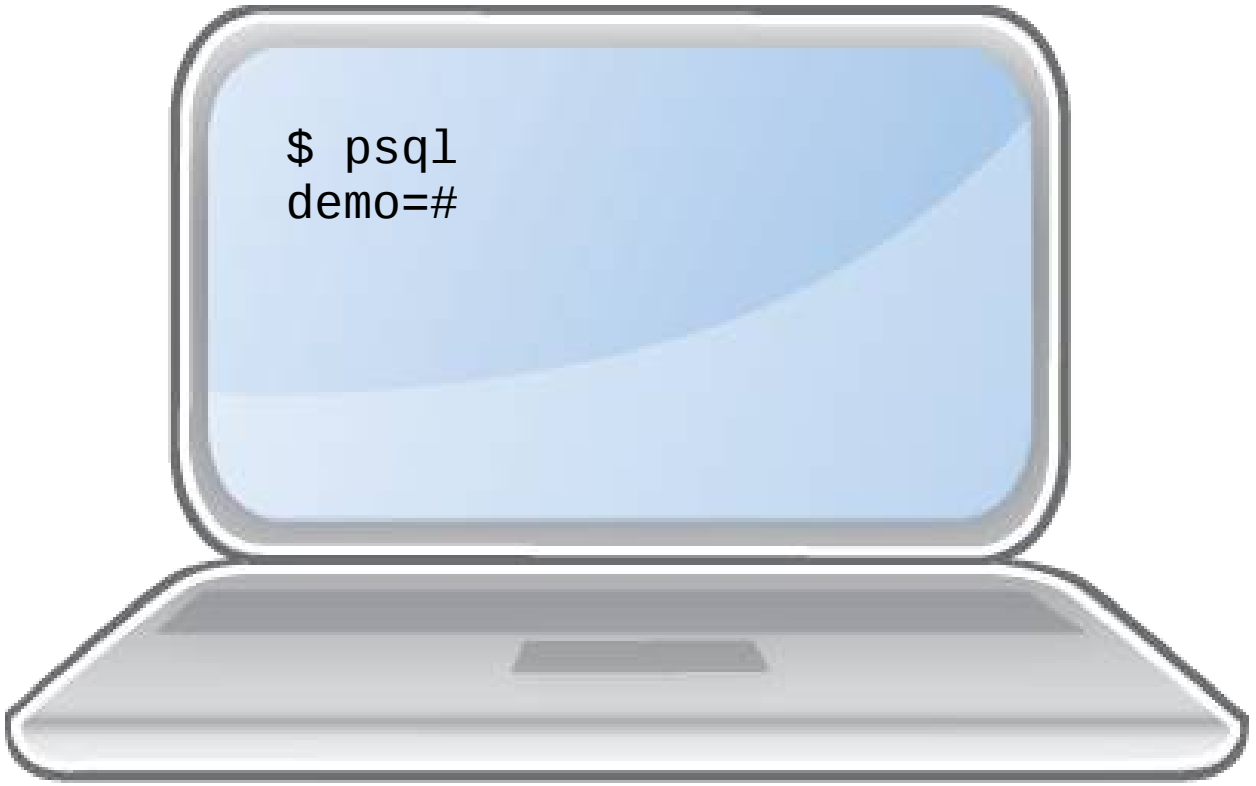
```
tweet @> '{"lang": "en"}'::jsonb
```




```
{   "lang": "ru",
    "user": {
        "name": "en"
    }
}
```

```
{   "lang": "en",
    "user": {
        "name": "MephamHS"
    }
}
```

Третья демонстрация



Заключение

Итоги

Что осталось за кадром

Вопросы и, возможно, ответы

Геометрические типы

GiST

- сбалансированное дерево
- поиск ближайших соседей

SP-GiST

- несбалансированное дерево

Разные алгоритмы: надо смотреть на реальных данных

См. также

- расширение PostGIS

Полнотекстовый поиск

GiST

неточный, относительно медленный поиск

GIN

значительно более быстрый поиск

возможно медленное обновление

См. также

расширение rum — фразовый поиск и ранжирование

расширение pg_trgm

расширения gist_btree и gin_btree

JSON

GIN (jsonb_ops)

- поддержка ? и @>

GIN (jsonb_path_ops)

- поддержка @>
- выше точность, меньше размер

См. также

- расширение jsquery
- стандарт SQL/JSON

Что осталось за кадром

Нельзя объять необъятное

- применение рассмотренных методов к другим типам данных
- другие методы доступа (hash, btree, brin, bloom, rum…)
- страничная организация
- алгоритмы вставки и обновления
- точные алгоритмы поиска
- детали реализации
- API методов доступа
- и так далее

Ссылки для изучения

Цикл статей про индексы

https://habrahabr.ru/company/postgrespro/blog/326096/

Документация

GiST	https://postgrespro.ru/docs/postgresql/10/gist
SP-GiST	https://postgrespro.ru/docs/postgresql/10/spgist
GIN	https://postgrespro.ru/docs/postgresql/10/gin
Полнотектовый поиск	https://postgrespro.ru/docs/postgresql/10/textsearch
JSON	https://postgrespro.ru/docs/postgresql/10/datatype-json

Статьи

Hellerstein, Naughton, Pfeffer, «Generalized Search Trees for Database Systems»
VLDB '95 Proceedings of the 21th International Conference on Very Large Data Bases, 562–573

Hellerstein, Pfeffer, «The RD-Tree: An Index Structure for Sets», 1994

Aref, Ilyas, «SP-GiST: An Extensible Database Index for Supporting Space Partitioning Trees»
Journal of Intelligent Information Systems, 17:2/3, 215–240, 2001

Спасибо за внимание

Хорошая мысля приходит опосля?

— edu@postgrespro.ru