YTsaurus

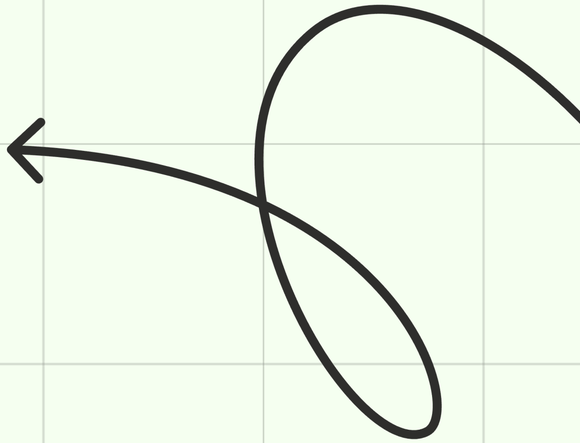
Планирование миллиардов задач каждый день

Колесниченко Игнат,
руководитель службы, YTsaurus

6 сентября 2023

Кто такой YTsaurus?

YTsaurus — BigData экосистема



1

Решает задачи хранения и обработки данных

2

Разработана в Яндексе и является ключевым компонентом инфраструктуры

3

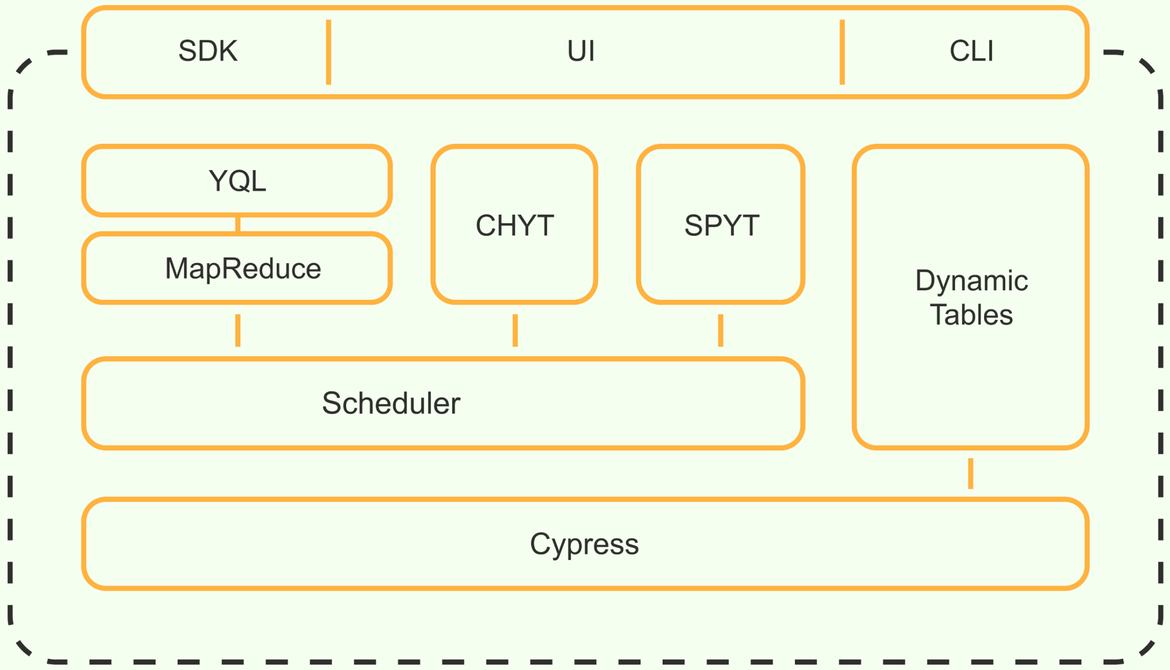
В марте 2023 система была выложена в OpenSource

Возможности YTsaurus

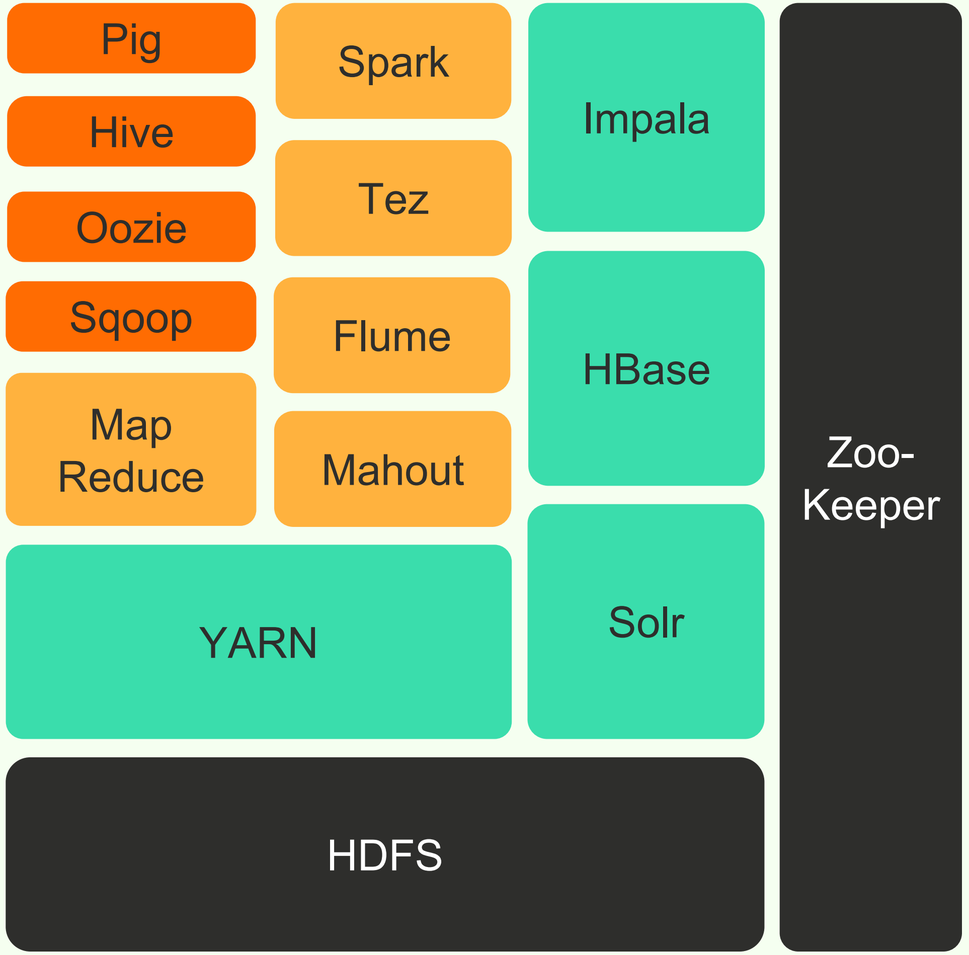
Кипарис: пространство имён и сервис координации	ZK HDFS NameNode Hive MetaStore
Статические таблицы/файлы: иммутабельные данные	HDFS
Динамические таблицы: масштабируемые KV-таблицы с транзакциями	HBase Spanner
Планировщик: multi-tenant менеджер ресурсов	YaRN Mesos
Обработка данных: • YTsaurus Map-Reduce • ClickHouse over YT • Spark over YT • YQL	Hadoop Map-Reduce Presto / Trino / Impala Spark HiveQL

Архитектура

YTsaurus

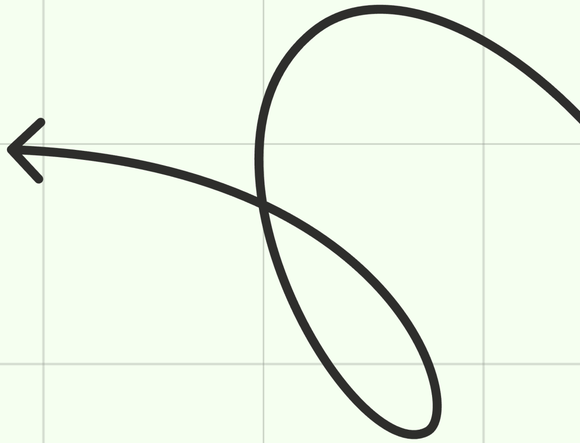


Hadoop



Какие задачи решает планировщик?

Обязанности планировщика



1

Реализация пользовательского API для **запуска операций**

2

Общение с нодами кластера и планирование отдельных **джобов**

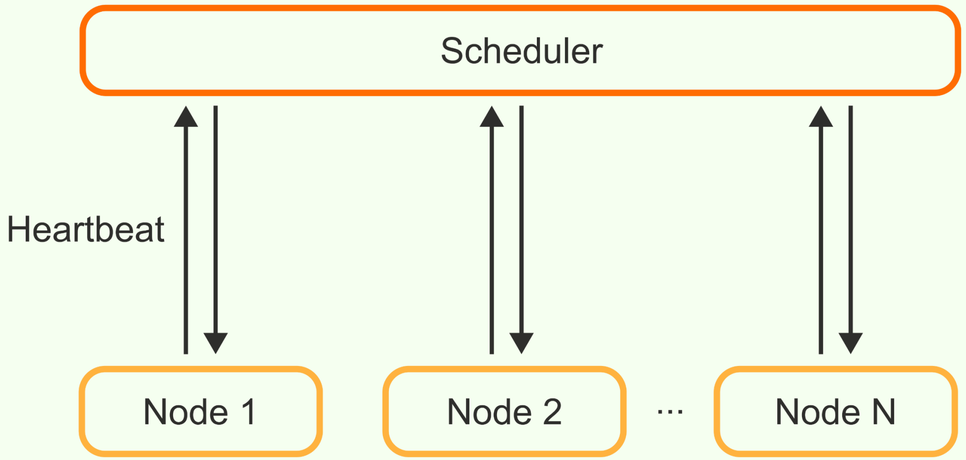
3

Обеспечение **честного** распределения ресурсов кластера между пользователями на основе **дерева пулов**

Запуск операции

```
{
    "input_tables": [...],
    "output_tables": [...],
    "mapper": {
        "command": "./my_wonderful_binary",
        "files": ["//home/ignat/my_wonderful_binary", ...],
        "cpu_limit": 10,
        "memory_limit": 20’000'000'000,
        ...
    }
    "pool": "dev",
    ...
}
```

Общение с нодами кластера



Heartbeat request

- Бегущие джобы
- Свободные ресурсы

Heartbeat response

- Новые джобы
- Прерывание некоторых бегущих джобов

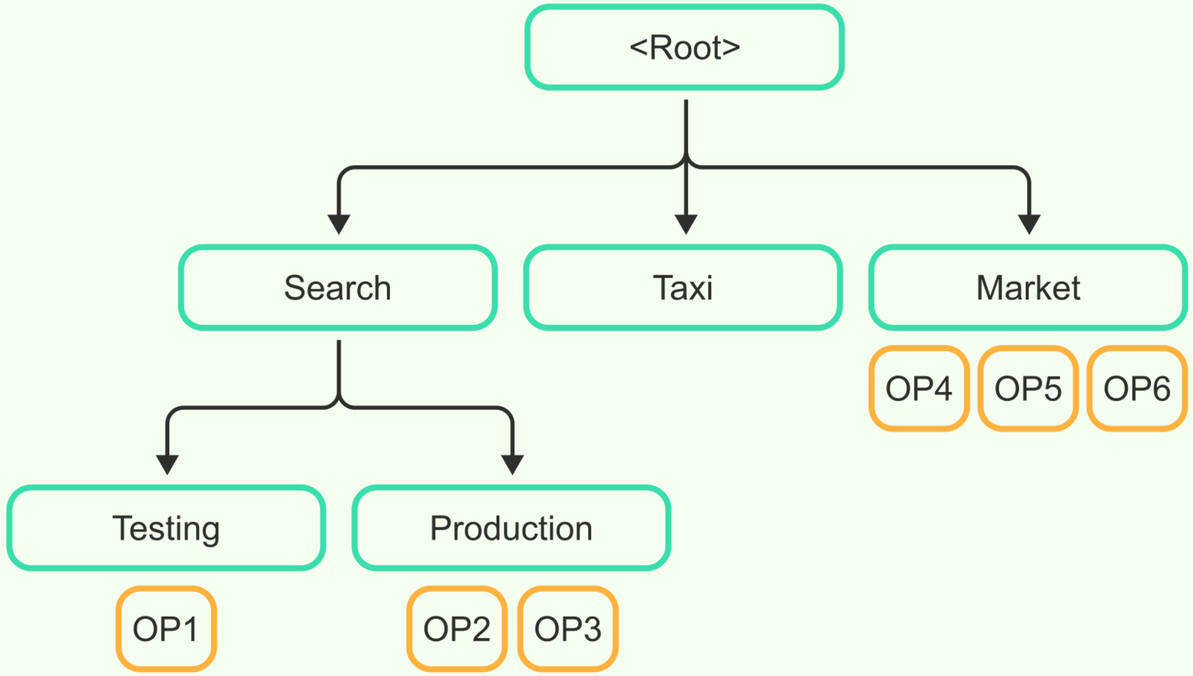
Дерево вычислительных пулов

Настройки пула

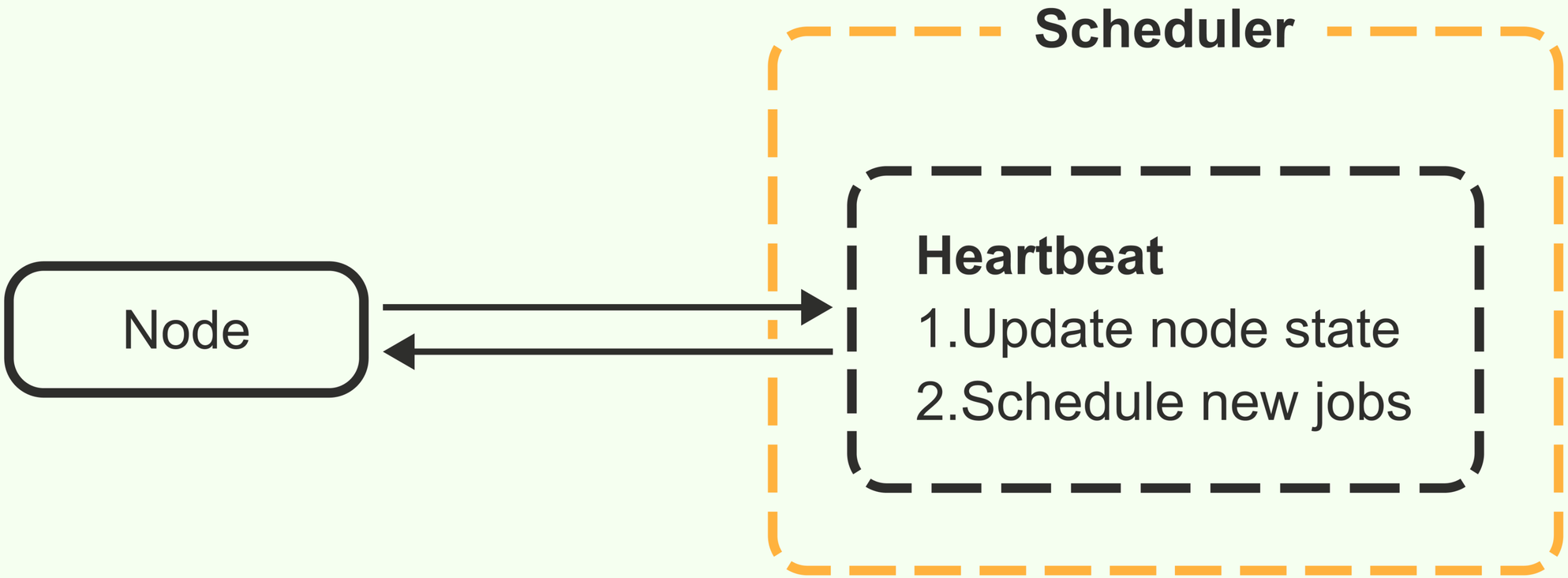
- Гарантии (CPU, RAM)
- Лимиты (число джобов)
- …

Атрибуты узлов дерева

- Usage (CPU, RAM)
- Demand (CPU, RAM)
- **Fair Share (CPU, RAM)**



Как происходит планирование



Нагрузка на планировщик

20’000

RPS с хартбитами
от нод кластера

20’000

Одновременно бегущих
операций

1M+

CPU под управлением
кластера

5B+

Попыток запланировать
джоб за одни сутки

Как развивался планировщик

2013 год: первый production кластер YTsaurus

500

Нод в кластере

~100

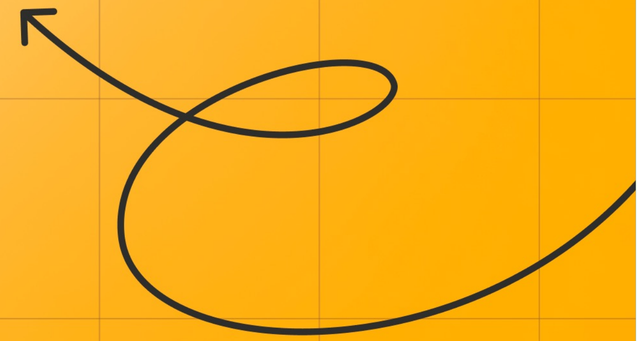
Одновременно
бегущих операций

~20 тыс.

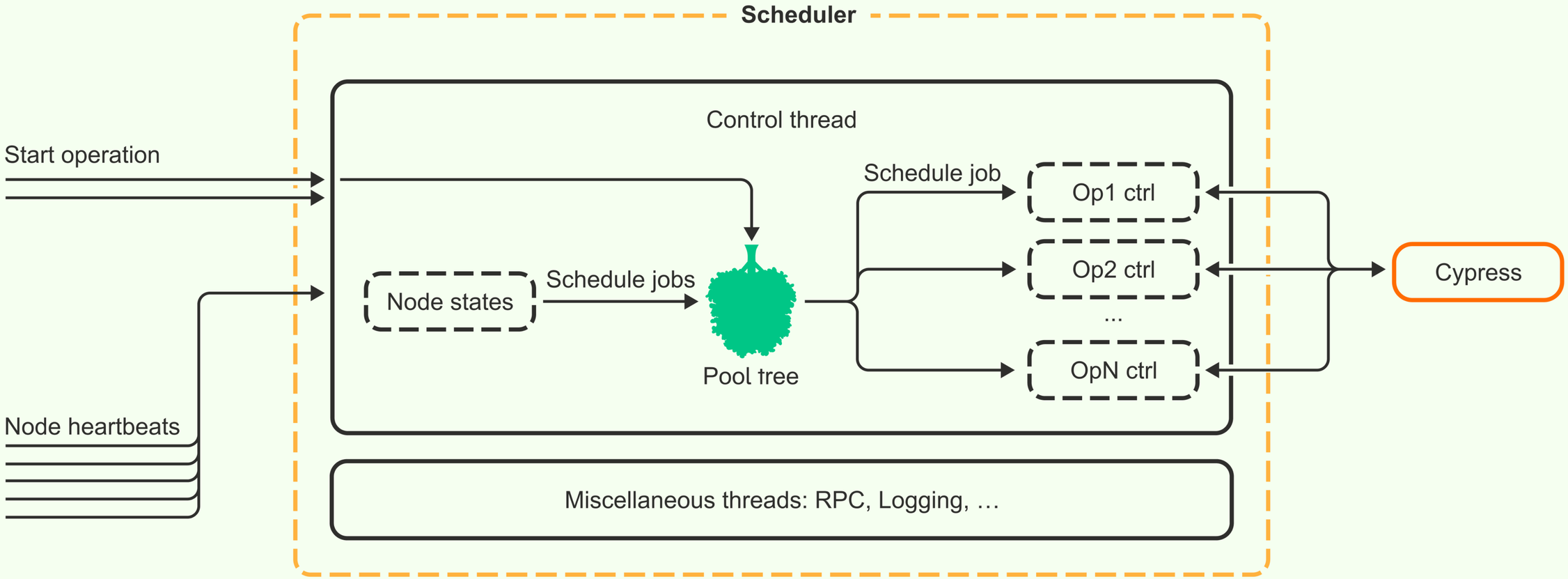
CPU

Несколько десятков

Вычислительных пулов



2013 год: планировщик изнутри



2013 год: планировщик изнутри

Node states:
hash-таблица
с состоянием нод

- Ресурсы
- Джобы
- Метаинформация

Pool tree:
дерево с насчитанной
атрибутикой

- Старт (окончание) операции добавляет (удаляет) лист в дерево
- Старт/окончание джобы изменяет атрибуты у листа и всех его предков

Operation controller:
отвечает за исполнение конкретной операции, формирует спецификации отдельных джобов

2015 год: распараллеливание контроллеров

1000+

Нод в кластере

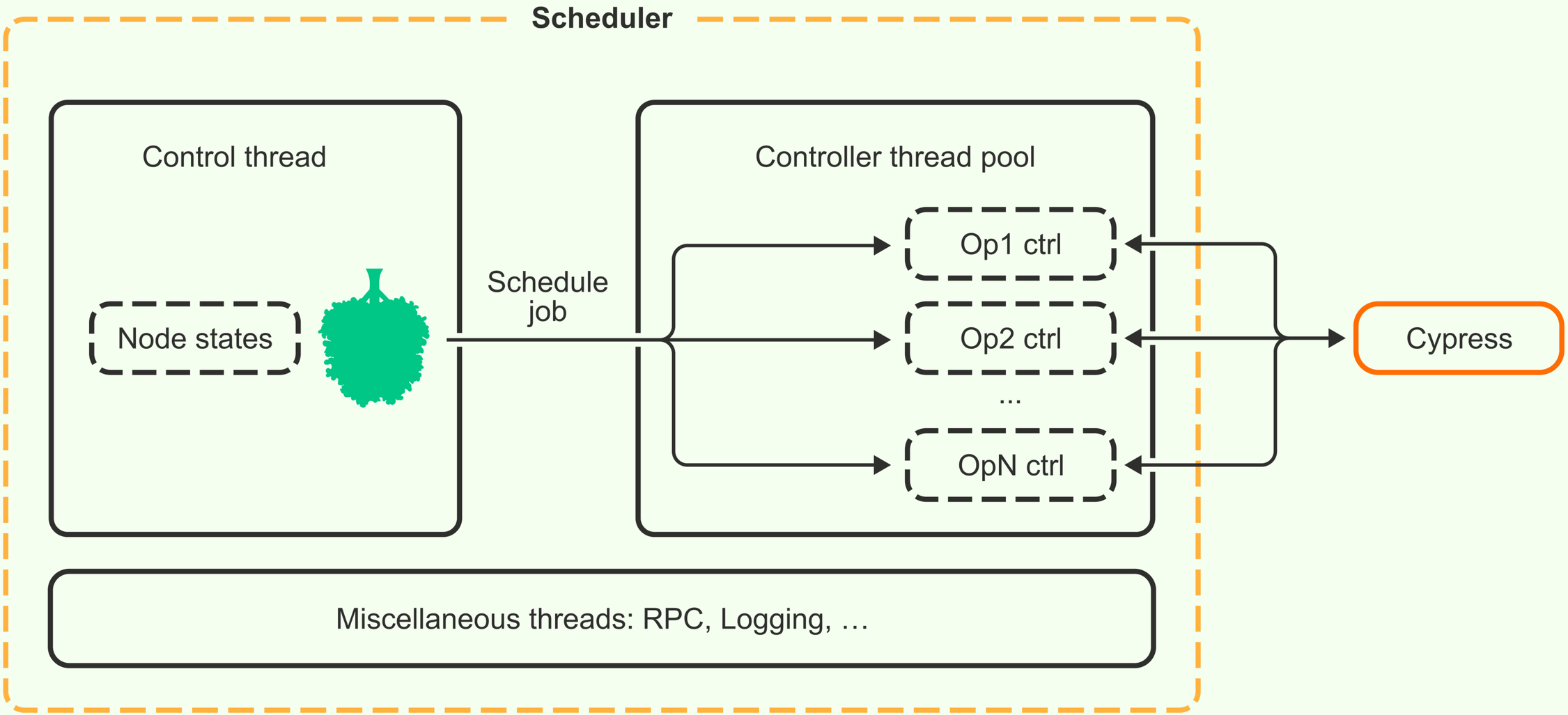
150-200

Одновременно
бегущих операций

50+

Вычислительных
пулов

2015 год: распараллеливание контроллеров



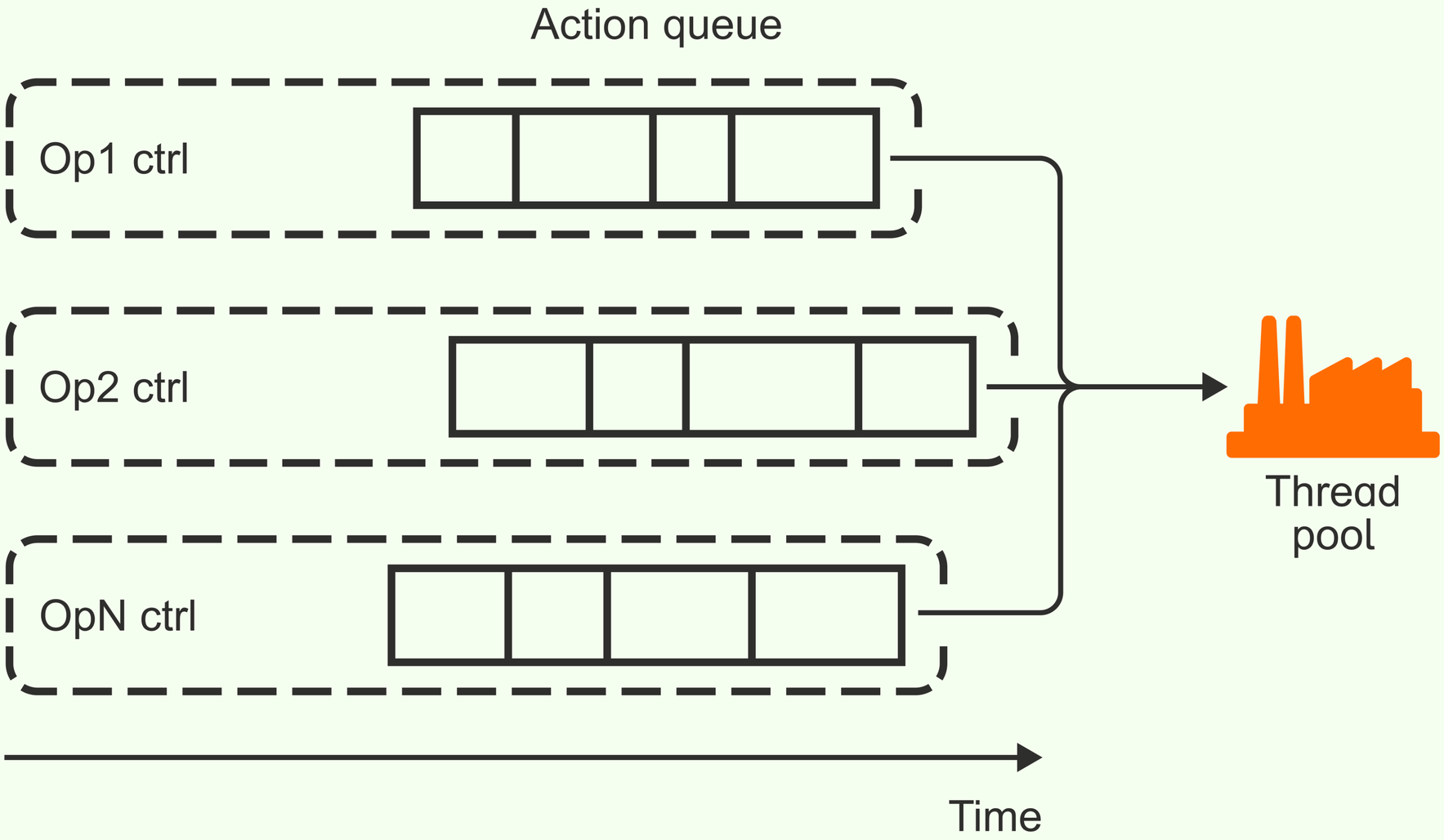
2015 год: распараллеливание контроллеров

Operation controller имеет нетривиальное состояние и реализует сложную логику

Возможные подходы

1. Заводить свой тред для каждой операции
   - Имеет пределы по масштабированию
   - Обычно операция потребляет очень мало CPU
2. Сделать работу с состоянием потокобезопасной
   - Очень трудоемко, сильно усложнит разработку контроллеров

2015 год: распараллеливание контроллеров



2016 год: распараллеливание хартбитов

2000

Нод в кластере

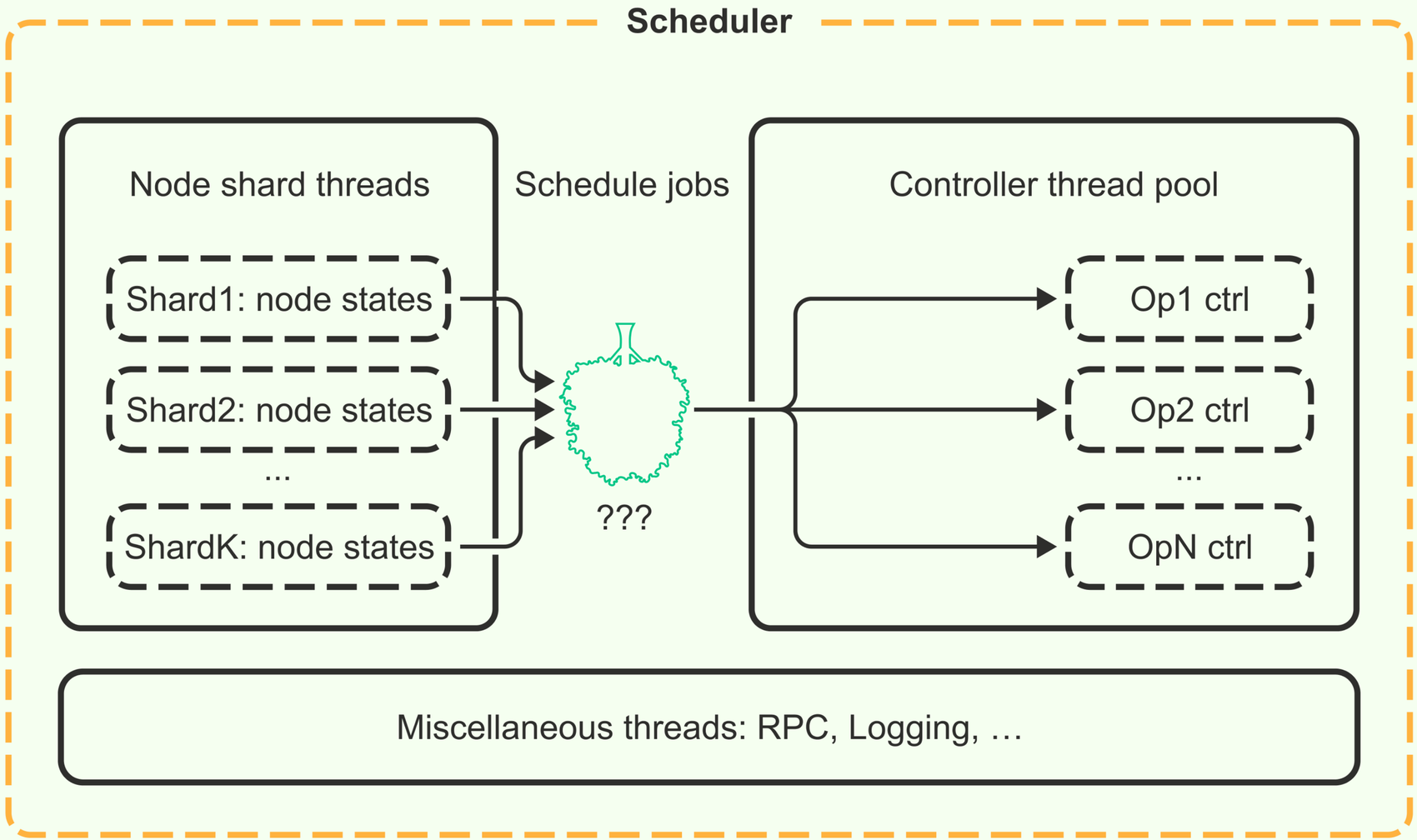
500

Одновременно
бегущих операций

100

Вычислительных
пулов

2016 год: распараллеливание хартбитов



2016 год: распараллеливание хартбитов

Проблема

Параллельные операции над деревом — это сложно и неэффективно

Решение

1. Не менять структуру дерева из разных потоков
2. Минимизировать shared state в узлах

2016 год: распараллеливание хартбитов

Trunk-версия дерева — изменяется только из контрольного потока

- Старт/завершение операций
- Изменение структуры пулов

Snapshot дерева — создается из trunk-версии

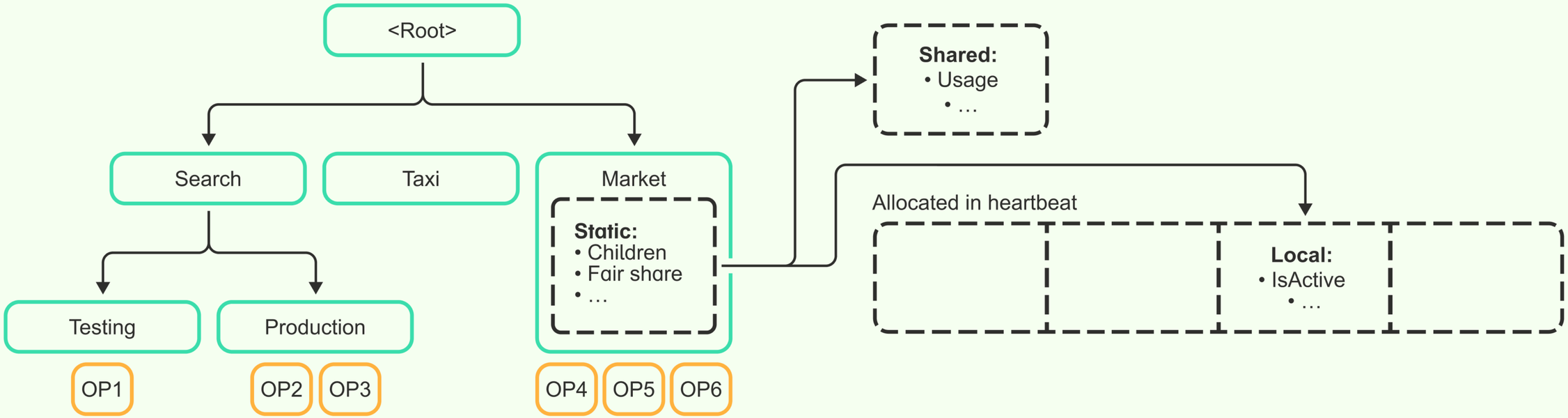
- Имеет фиксированную структуру
- Содержит рассчитанные атрибуты на узлах дерева (fair share)

В рамках хартбита берется последний снепшот, и он используется для планирования на протяжении всего хартбита

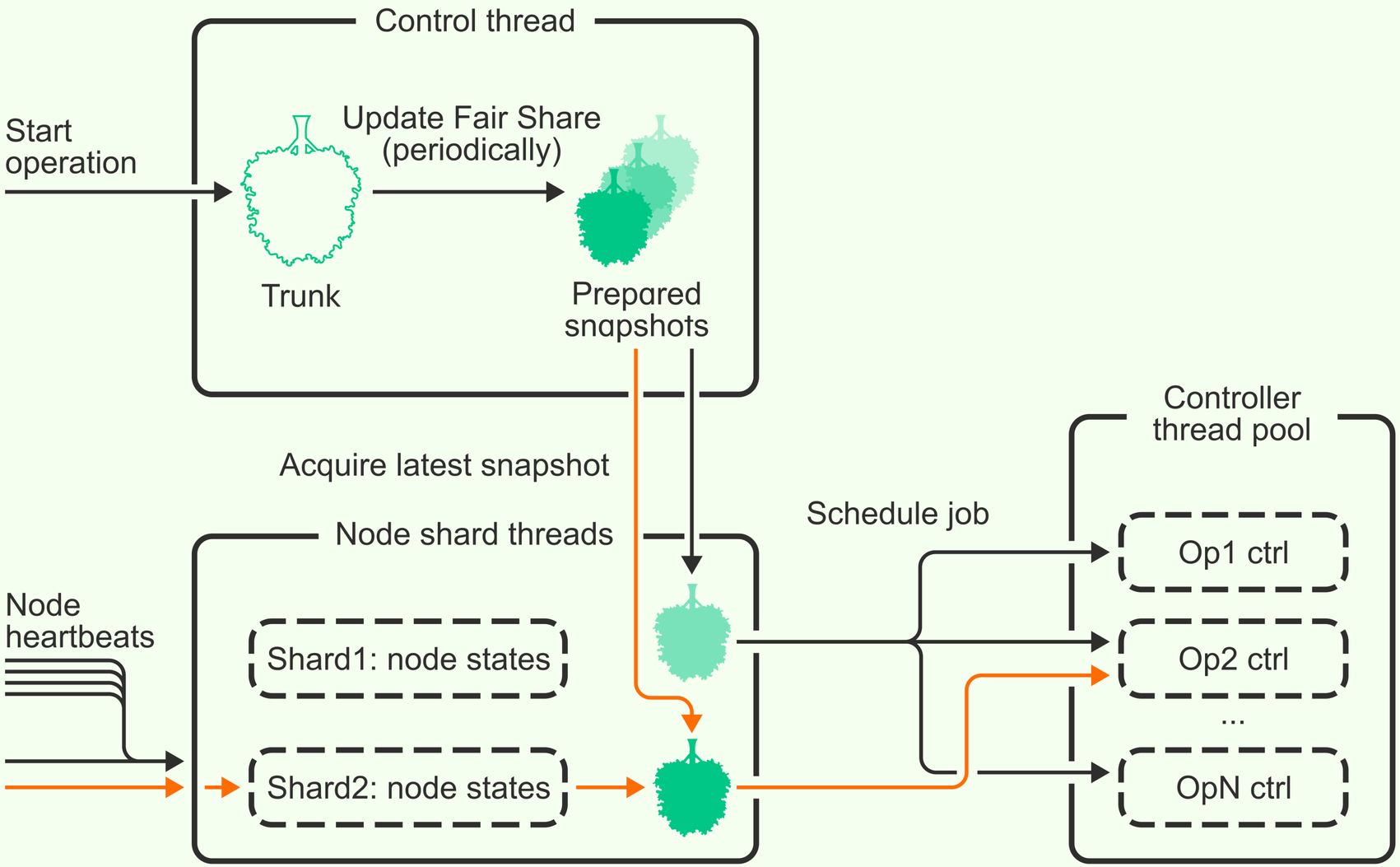
2016 год: распараллеливание хартбитов

Узел дерева состоит из трех частей

- **Static:** атрибуты, которые насчитываются при построении снепшота
- **Shared:** атрибуты, требующие realtime обновлений (например, usage)
- **Local:** атрибуты, необходимые только в рамках хартбита



2016 год: распараллеливание хартбитов



2017-2018: шардирование контроллеров операций

5000

Нод в кластере

1000+

Одновременно бегущих операций

300+

Вычислительных пулов

2017-2018 год: шардирование

Таксономия большой операции

- 1M+ чанков на входе shuffle-стадия
- 10000+ бегущих джобов
- Memory footprint 10GB+

Проблема: планировщик потребляет много RAM (200GB+)

- Нет возможности растить число операций
- Процесс «залипает» на 5-10 секунд при fork-ах*

* Форки используются для того, чтобы периодически писать снепшоты состояний операций

2017-2018 год: шардирование

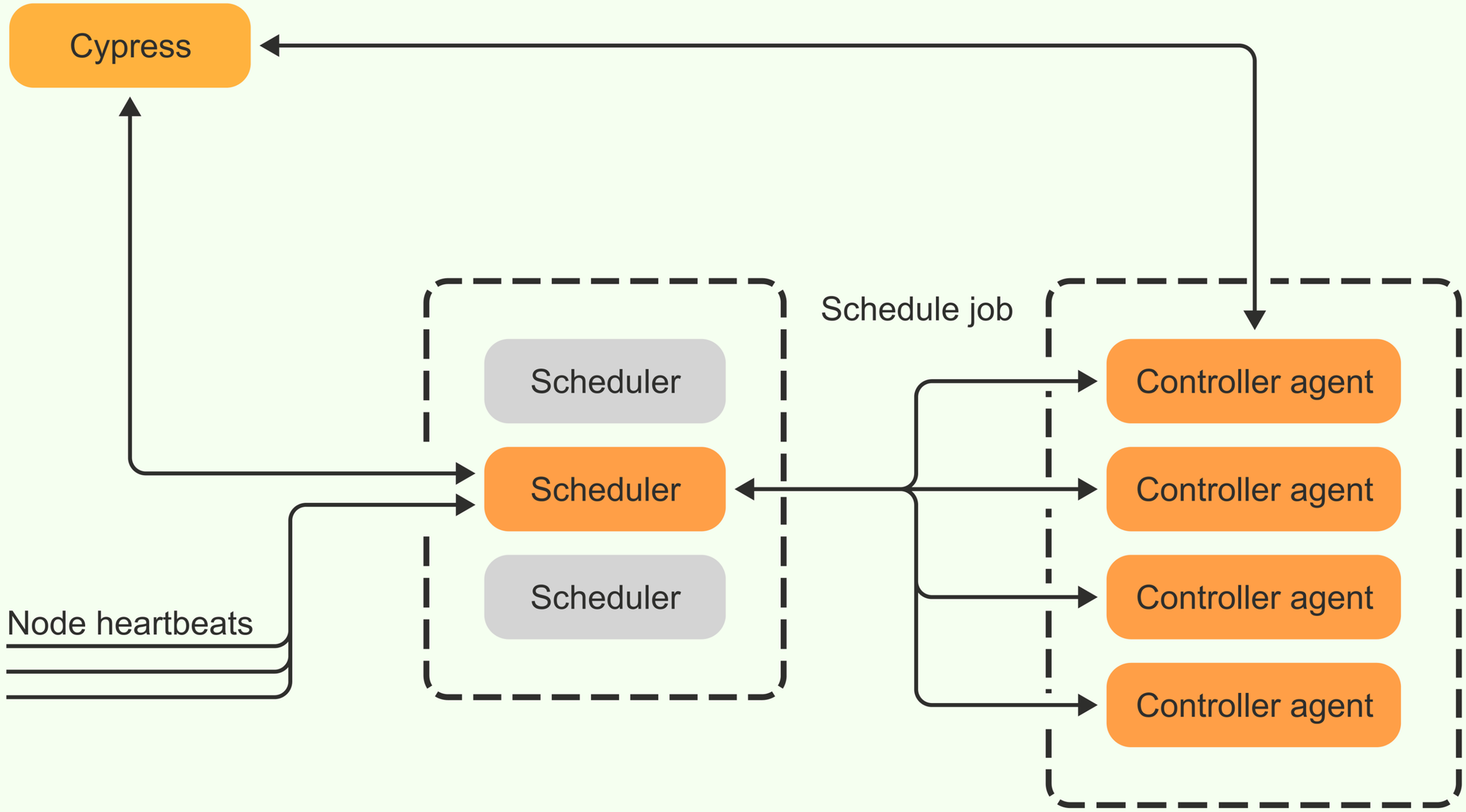
Решение

Выселить контроллеры операций
на отдельные хосты

Сложности

- Явно выделить протокол взаимодействия шедулера с операциями и разделить стейт
- Решить проблемы распределённости состояния

2017-2018 год: шардирование



2019 год: алгоритмическая оптимизация снепшота дерева

7000

Нод в кластере

2000+

Одновременно бегущих операций

500+

Вычислительных пулов

2019 год: оптимизация снепшота дерева

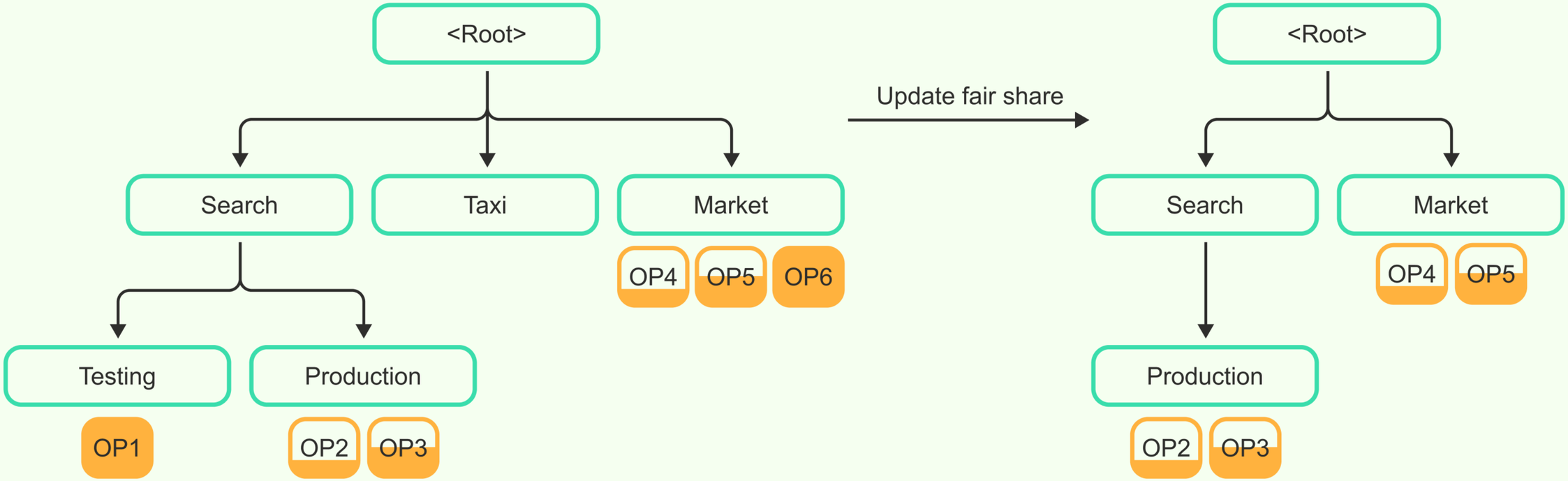
Проблема

Каждый хартбит инициализирует local-состояние для всех узлов дерева, при этом после этого может спланировать всего один джоб

Решение

Исключить из снепшота заведомо лишние узлы

2019 год: оптимизация снепшота дерева



2020-2021 года: локальные оптимизации

10000

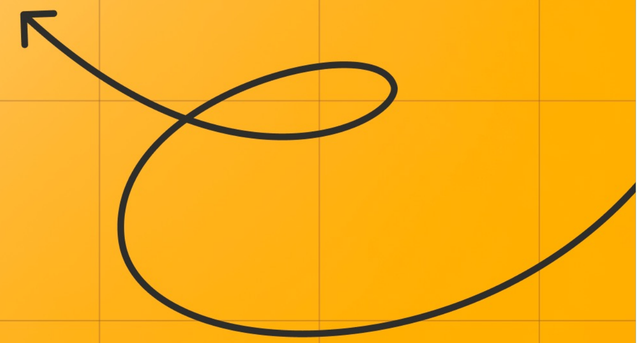
Нод в кластере

5000+

Одновременно бегущих операций

1000+

Вычислительных пулов



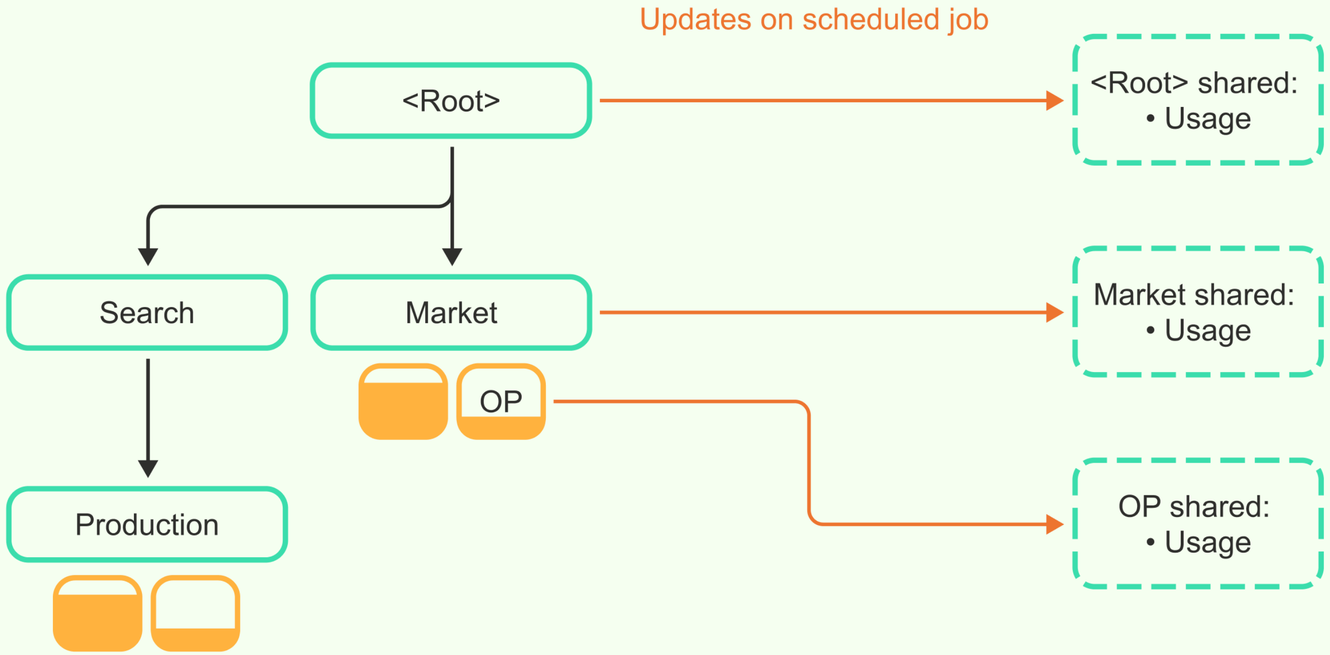
2020-2021 года: локальные оптимизации

Проблема:

Работает 16 Node Shard тредов, в среднем они загружены на 70-80%, увеличение числа тредов до 32 не дает эффекта

Проблема в высоком contention:

Любой schedule job трогает shared часть корня



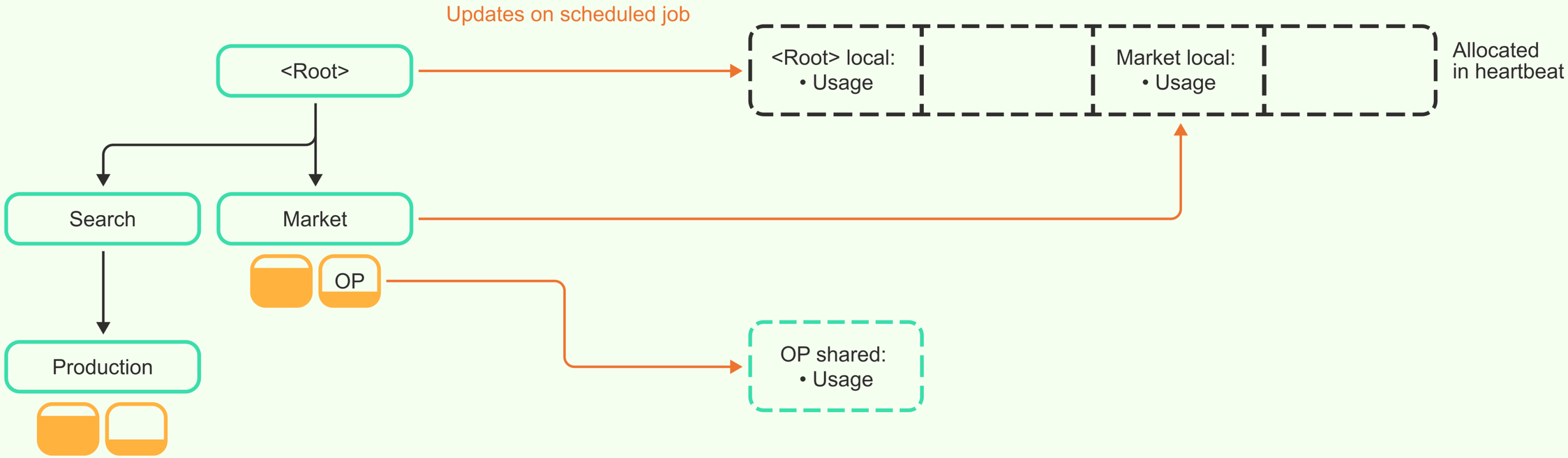
2020-2021 года: локальные оптимизации

Наблюдения:

- Абсолютно точный usage нужен в операциях и пулах с лимитами
- Эмпирически проверено, что отставание usage в пулах не влияет на качество

Решение:

- Поддерживать shared состояние только для операций и пулов с лимитами
- Для остальных узлов поддерживать в local состоянии



2022 год — сейчас: локальные оптимизации

15000+

Нод в кластере

20000

Одновременно бегущих операций

2000

Вычислительных пулов

2022 год — сейчас

Проблема: алгоритмическая сложность работы планировщика описывается формулой

Heartbeat RPS × Tree size

- Heartbeat RPS растет с увеличением количества ресурсов в кластере
- Tree size растет с увеличением числа пользователей

2022 год — сейчас

Есть три направления по преодолению роста

1

Ограничение роста RPS: в больших операциях джобы имеют примерно одинаковую геометрию, можно переиспользовать уже выделенные аллокации

2

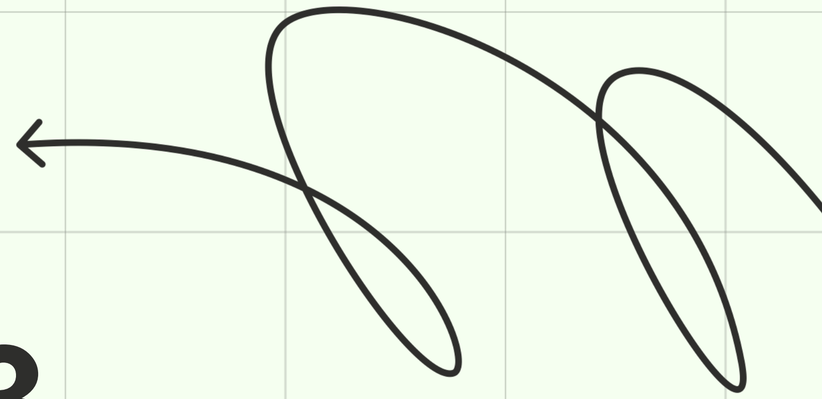
Уменьшение размера снепшота: можно выделить поддерево ограниченного размера, в которое с высокой вероятностью будут планироваться джобы

3

Шардирование планировщика: превратить Node Shard треды в отдельные компоненты, но нужно жертвовать синхронностью shared состояния и качеством планирования (или найти иную размерность для шардирования)

Чему мы научились?

Выводы или вместо заключения



1

Разделение алгоритма на независимые части и выделение простых API — путь к безболезненному распараллеливанию

2

Есть трейд-офф между богатыми возможностями системы и возможностью масштабироваться

3

Алгоритмические оптимизации лучше низкоуровневых: выгодней не делать работу, чем ускорять её выполнение

Дашборд


Scheduler Control CPU
100
80
60
40
20
0
22.08.23
24.08.23
26.08.23
28.08.23
Controller Agent Control CPU
40
30
20
10
0
22.08.23
24.08.23
26.08.23
28.08.23
Scheduler Node Shards CPU
4K
3K
2K
1K
0
22.08.23
24.08.23
26.08.23
28.08.23
Controller Agent Controllers CPU
800
600
400
200
0
22.08.23
24.08.23
26.08.23
28.08.23

YTsaurus

Вопросы?

Колесниченко Игнат
Руководитель службы, YTsaurus