History of ClickHouse at Ahrefs

Yasunari Watanabe

Agenda

- **About us**
- Road to ClickHouse adoption
- Upstream patch: In-memory marks compression

ahrefs

About me

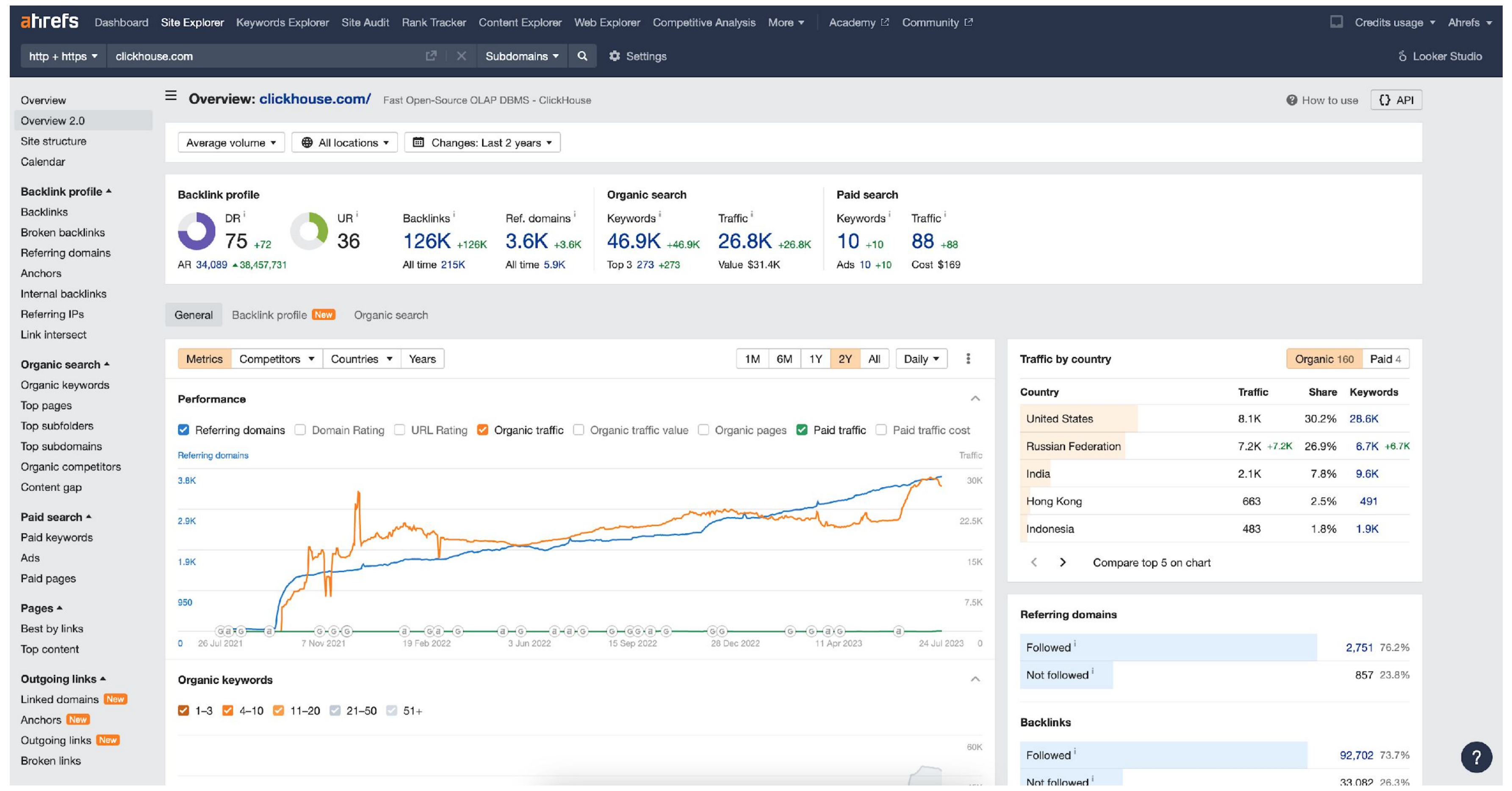
- Joined Ahrefs ~2 years ago
- Backend developer
- Working on our web crawler

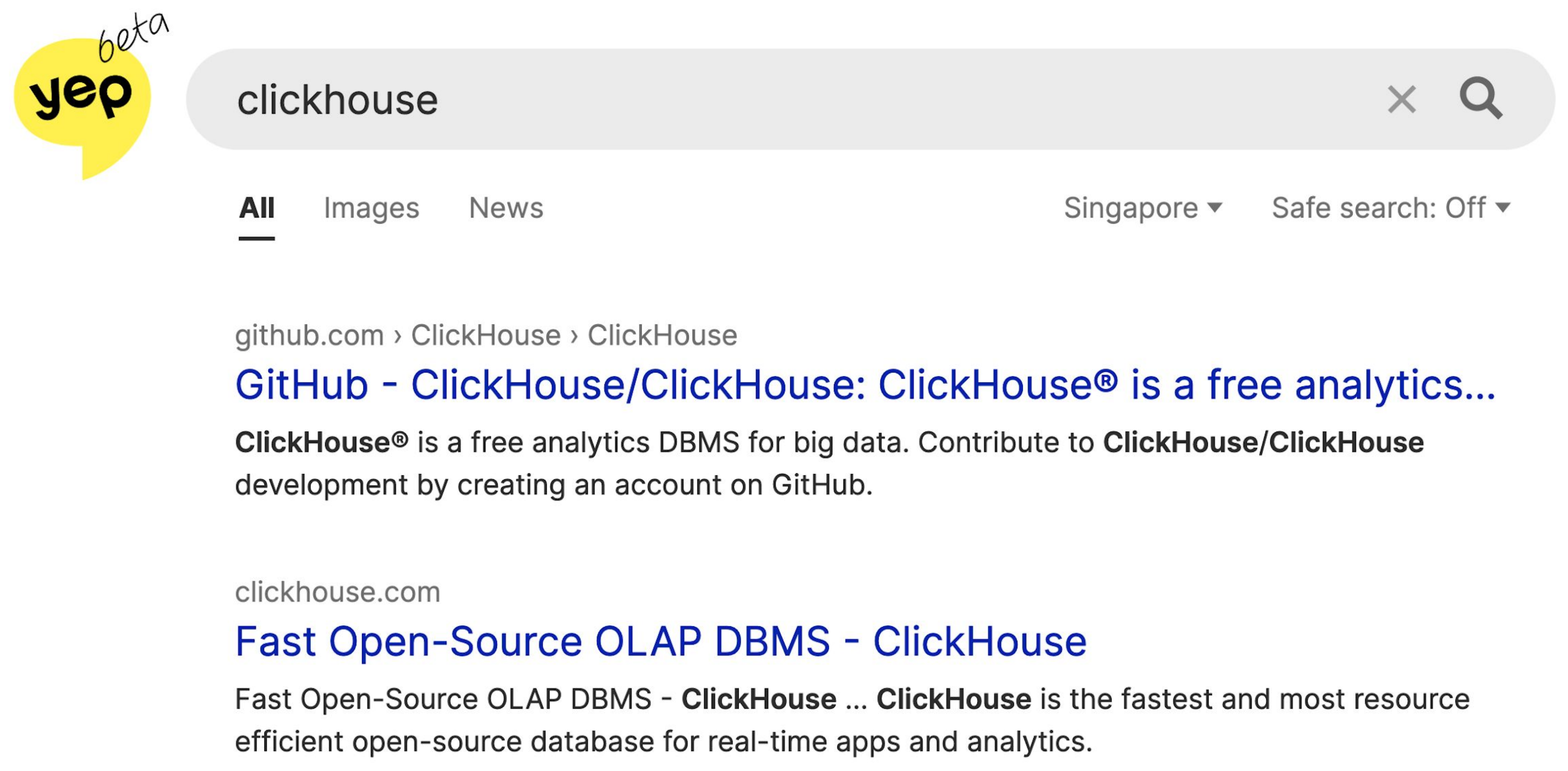
About Ahrefs: Who we are

- Founded in 2010, HQ in Singapore
- ~120 people, two thirds remote
- 50% engineers/data scientists
  - 19 backend developers,
    including 1 ClickHouse developer
  - 6 DevOps engineers

About Ahrefs: What we do

- Process big data about the Web and transform into relevant indicators
  - SEO metrics
  - Our new search engine, **Yep**

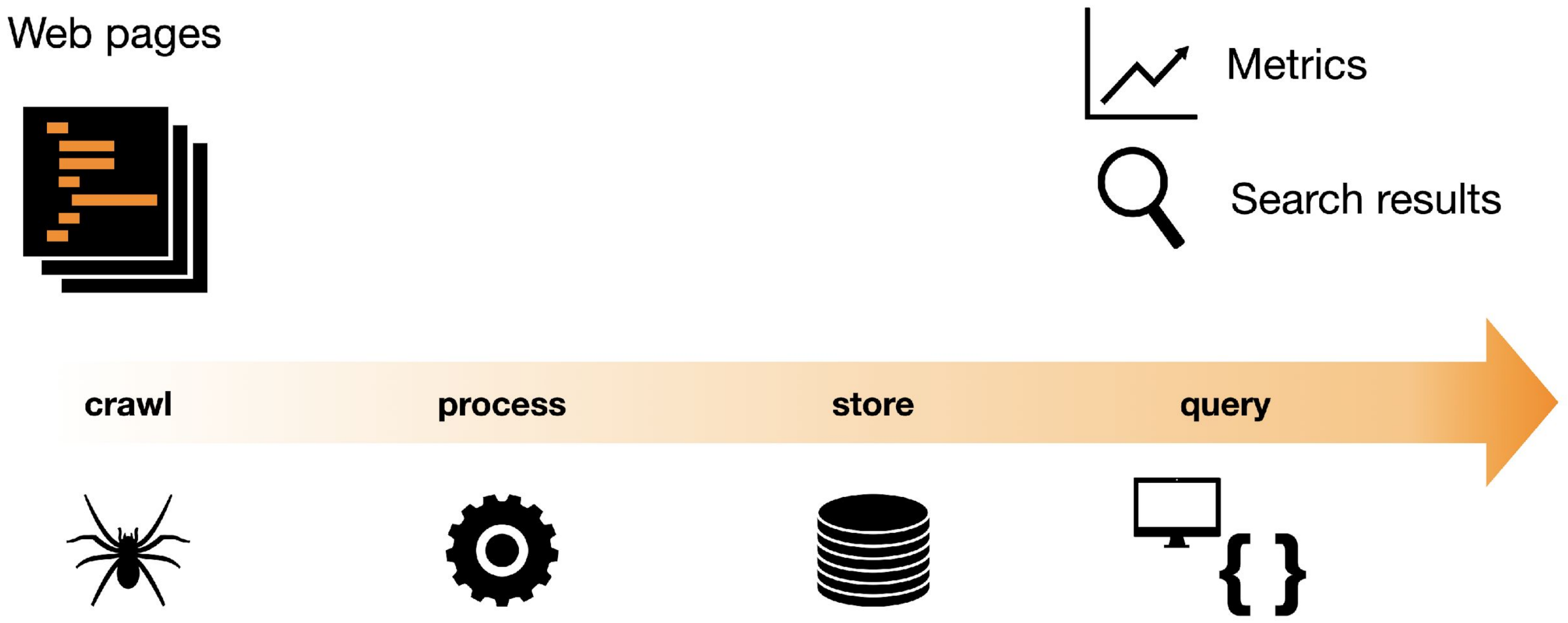




github.com › ClickHouse › ClickHouse
GitHub - ClickHouse/ClickHouse: ClickHouse® is a free analytics...
ClickHouse® is a free analytics DBMS for big data. Contribute to **ClickHouse/ClickHouse** development by creating an account on GitHub.

clickhouse.com
Fast Open-Source OLAP DBMS - ClickHouse
Fast Open-Source OLAP DBMS - **ClickHouse** ... **ClickHouse** is the fastest and most resource efficient open-source database for real-time apps and analytics.

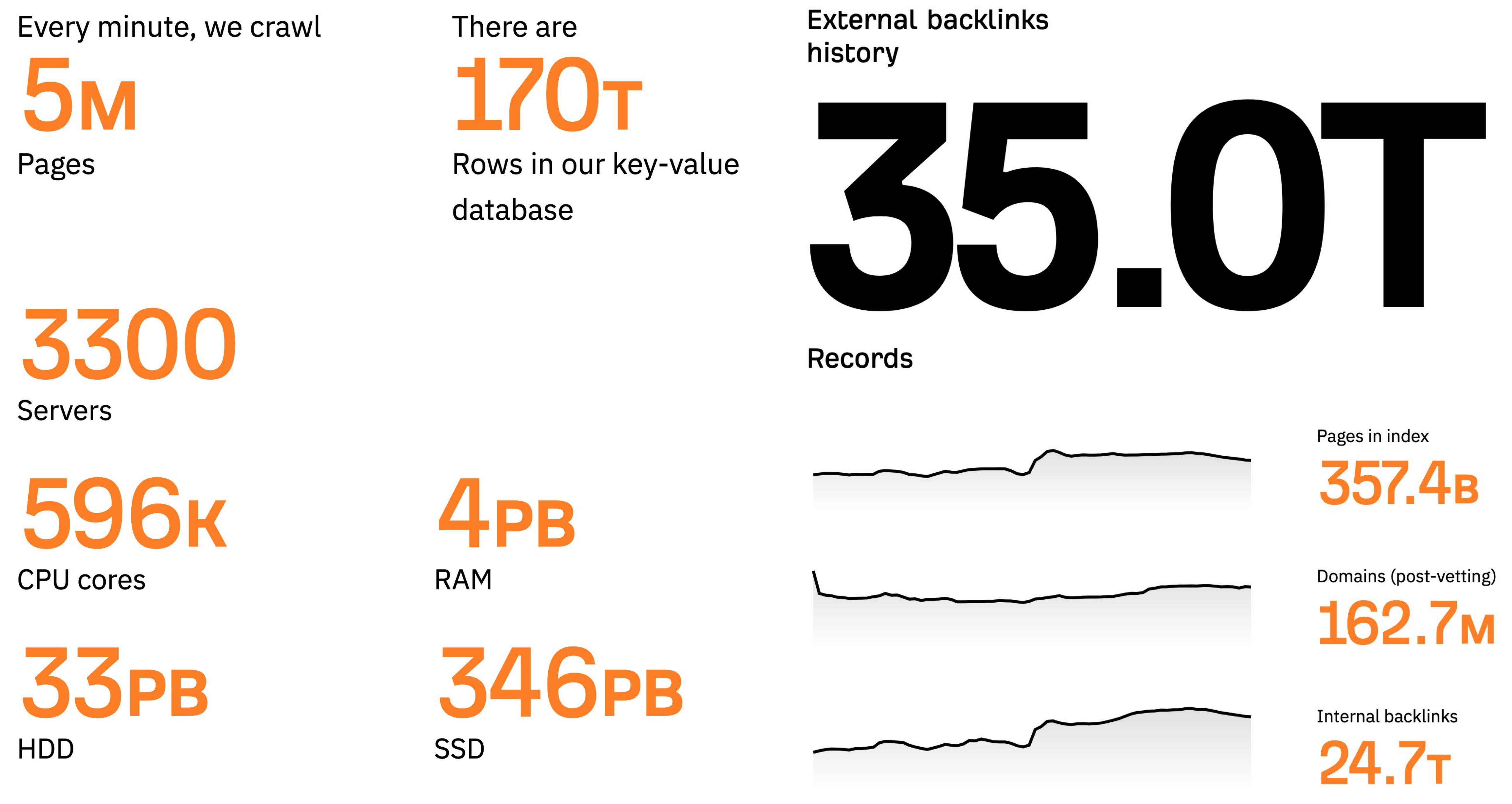
About Ahrefs: Data flow



Quick facts

- Most active web crawler in SEO industry
- World's largest index of live backlinks
- Running on our own hardware

ahrefs.com/big-data



Every minute, we crawl
5M
Pages

There are
170T
Rows in our key-value database

External backlinks history
35.0T
Records

3300
Servers

596K
CPU cores

4PB
RAM

33PB
HDD

346PB
SSD

Pages in index
357.4B

Domains (post-vetting)
162.7M

Internal backlinks
24.7T

Agenda

- About us
- **Road to ClickHouse adoption**
- Upstream patch: In-memory marks compression

Early days

- Tried all available solutions (Cassandra, Hypertable); none fast enough
- Developed custom solution optimized for crawling the web with limited resources (tens of servers, circa 2010)
- QFS to handle unevenness among shards
- Elasticsearch for various non-crawler use cases

Worked well, but…

- Limitations of custom storage
  - Lack of versatile querying API
  - Lack of bells and whistles
- Evolving feature requirements, size of the Web, our infra…

Enter ClickHouse

- Started experimenting with ClickHouse in 2019
- Similarities in design, but ClickHouse has dedicated time + resources to features and optimizations
  - SQL interface
  - Many input/output formats
- Notable difference: column-based

Migration to ClickHouse

- Custom storage
  - Migrated part of the web crawler a few years ago
  - Rest of migration ongoing
  - Challenge: need to keep everything running during migration
- Elasticsearch
  - Most indices rebuilt in ClickHouse

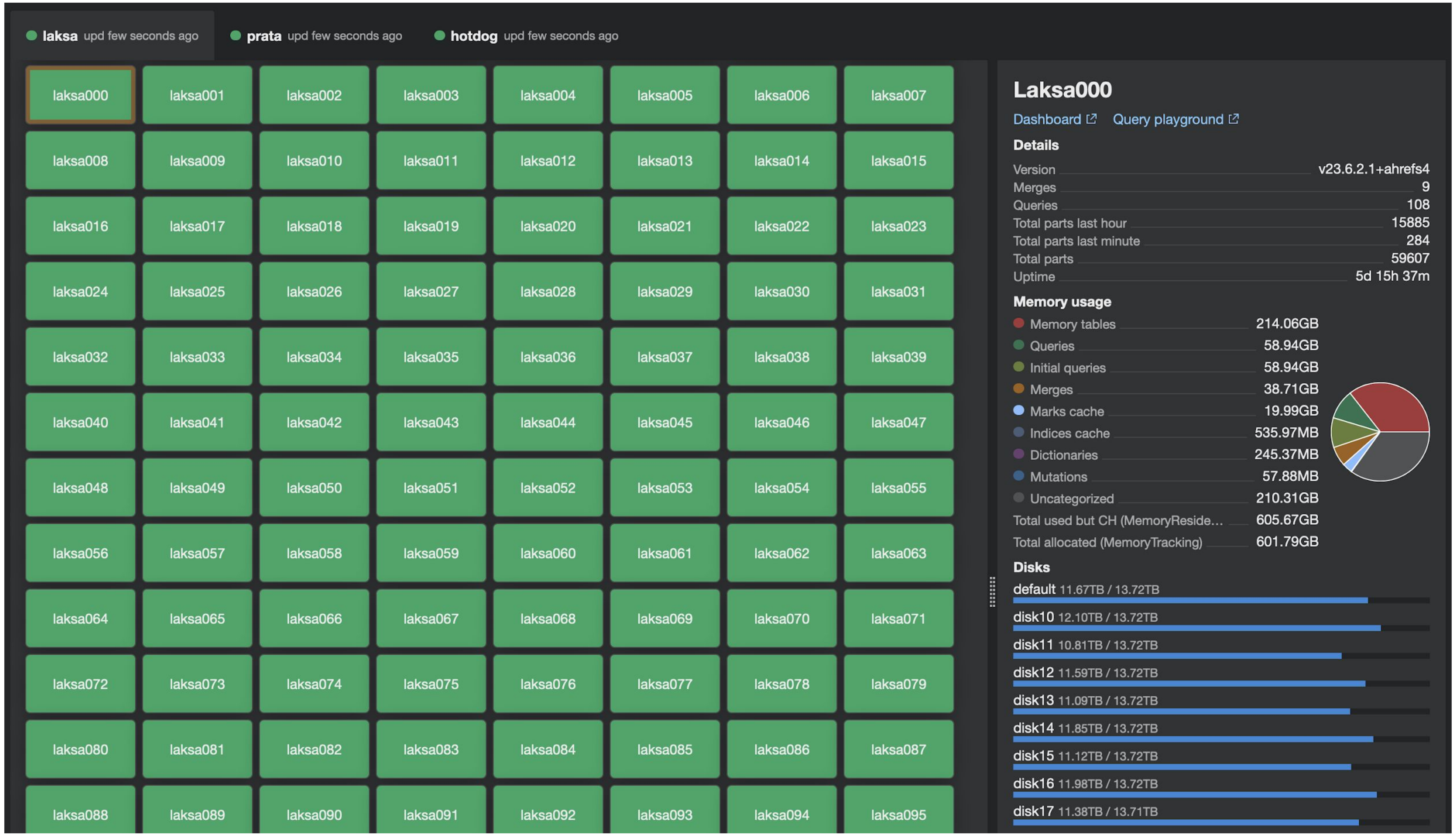
ClickHouse deployment

- Multiple clusters each with hundreds of hosts
- Geo-replicated main cluster, designated read/write replicas
- Large tables with trillions of rows, tens of columns
- More capacity to come

ClickHouse in our workflow

- Techniques
    - Optimizing insert: custom code for partitioned buffers
    - Fetch/attach to move parts between servers efficiently
- Heavy tooling on top of SQL interface
    - Metaprogramming for tight integration with application code (in OCaml)

Tools: Cluster monitoring



● **laksa** upd few seconds ago ● **prata** upd few seconds ago ● **hotdog** upd few seconds ago

laksa000	laksa001	laksa002	laksa003	laksa004	laksa005	laksa006	laksa007
laksa008	laksa009	laksa010	laksa011	laksa012	laksa013	laksa014	laksa015
laksa016	laksa017	laksa018	laksa019	laksa020	laksa021	laksa022	laksa023
laksa024	laksa025	laksa026	laksa027	laksa028	laksa029	laksa030	laksa031
laksa032	laksa033	laksa034	laksa035	laksa036	laksa037	laksa038	laksa039
laksa040	laksa041	laksa042	laksa043	laksa044	laksa045	laksa046	laksa047
laksa048	laksa049	laksa050	laksa051	laksa052	laksa053	laksa054	laksa055
laksa056	laksa057	laksa058	laksa059	laksa060	laksa061	laksa062	laksa063
laksa064	laksa065	laksa066	laksa067	laksa068	laksa069	laksa070	laksa071
laksa072	laksa073	laksa074	laksa075	laksa076	laksa077	laksa078	laksa079
laksa080	laksa081	laksa082	laksa083	laksa084	laksa085	laksa086	laksa087
laksa088	laksa089	laksa090	laksa091	laksa092	laksa093	laksa094	laksa095

Laksa000

Dashboard Query playground

Details

Version	v23.6.2.1+ahrefs4
Merges	9
Queries	108
Total parts last hour	15885
Total parts last minute	284
Total parts	59607
Uptime	5d 15h 37m

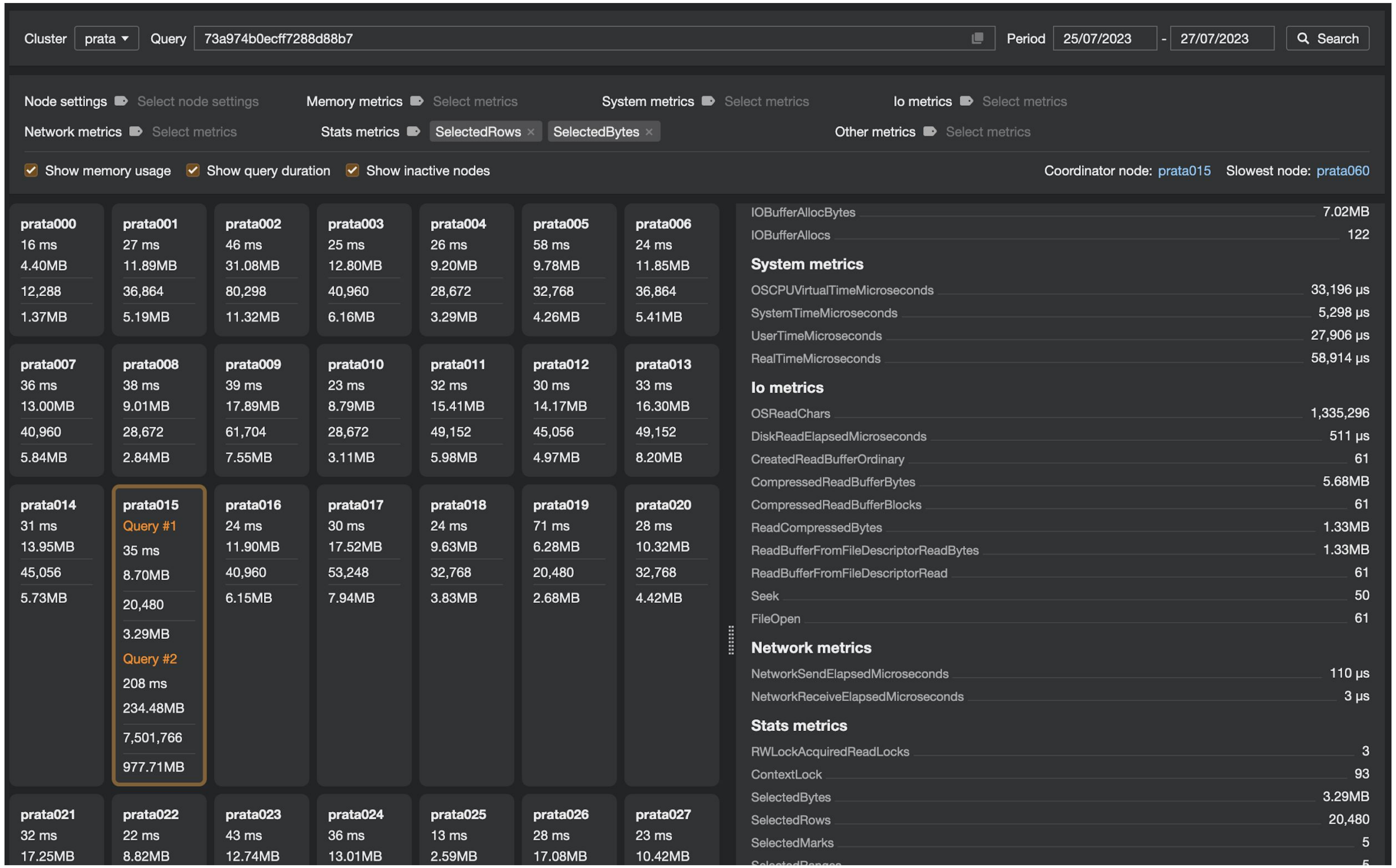
Memory usage

● Memory tables	214.06GB
● Queries	58.94GB
● Initial queries	58.94GB
● Merges	38.71GB
● Marks cache	19.99GB
● Indices cache	535.97MB
● Dictionaries	245.37MB
● Mutations	57.88MB
● Uncategorized	210.31GB
Total used but CH (MemoryReside...	605.67GB
Total allocated (MemoryTracking)	601.79GB

Disks

default	11.67TB / 13.72TB
disk10	12.10TB / 13.72TB
disk11	10.81TB / 13.72TB
disk12	11.59TB / 13.72TB
disk13	11.09TB / 13.72TB
disk14	11.85TB / 13.72TB
disk15	11.12TB / 13.72TB
disk16	11.98TB / 13.72TB
disk17	11.38TB / 13.71TB

Tools: Query analyzer



Cluster prata Query 73a974b0ecff7288d88b7 Period 25/07/2023 - 27/07/2023 Search

Node settings Select node settings · Memory metrics Select metrics · System metrics Select metrics · Io metrics Select metrics

Network metrics Select metrics · Stats metrics SelectedRows × SelectedBytes × · Other metrics Select metrics

✓ Show memory usage ✓ Show query duration ✓ Show inactive nodes

Coordinator node: prata015 Slowest node: prata060

prata000 16 ms 4.40MB 12,288 1.37MB	prata001 27 ms 11.89MB 36,864 5.19MB	prata002 46 ms 31.08MB 80,298 11.32MB	prata003 25 ms 12.80MB 40,960 6.16MB	prata004 26 ms 9.20MB 28,672 3.29MB	prata005 58 ms 9.78MB 32,768 4.26MB	prata006 24 ms 11.85MB 36,864 5.41MB
prata007 36 ms 13.00MB 40,960 5.84MB	prata008 38 ms 9.01MB 28,672 2.84MB	prata009 39 ms 17.89MB 61,704 7.55MB	prata010 23 ms 8.79MB 28,672 3.11MB	prata011 32 ms 15.41MB 49,152 5.98MB	prata012 30 ms 14.17MB 45,056 4.97MB	prata013 33 ms 16.30MB 49,152 8.20MB
prata014 31 ms 13.95MB 45,056 5.73MB	prata015 Query #1 35 ms 8.70MB 20,480 3.29MB Query #2 208 ms 234.48MB 7,501,766 977.71MB	prata016 24 ms 11.90MB 40,960 6.15MB	prata017 30 ms 17.52MB 53,248 7.94MB	prata018 24 ms 9.63MB 32,768 3.83MB	prata019 71 ms 6.28MB 20,480 2.68MB	prata020 28 ms 10.32MB 32,768 4.42MB
prata021 32 ms 17.25MB	prata022 22 ms 8.82MB	prata023 43 ms 12.74MB	prata024 36 ms 13.01MB	prata025 13 ms 2.59MB	prata026 28 ms 17.08MB	prata027 23 ms 10.42MB

IOBufferAllocBytes	7.02MB
IOBufferAllocs	122
System metrics	
OSCPUVirtualTimeMicroseconds	33,196 μs
SystemTimeMicroseconds	5,298 μs
UserTimeMicroseconds	27,906 μs
RealTimeMicroseconds	58,914 μs
Io metrics	
OSReadChars	1,335,296
DiskReadElapsedMicroseconds	511 μs
CreatedReadBufferOrdinary	61
CompressedReadBufferBytes	5.68MB
CompressedReadBufferBlocks	61
ReadCompressedBytes	1.33MB
ReadBufferFromFileDescriptorReadBytes	1.33MB
ReadBufferFromFileDescriptorRead	61
Seek	50
FileOpen	61
Network metrics	
NetworkSendElapsedMicroseconds	110 μs
NetworkReceiveElapsedMicroseconds	3 μs
Stats metrics	
RWLockAcquiredReadLocks	3
ContextLock	93
SelectedBytes	3.29MB
SelectedRows	20,480
SelectedMarks	5

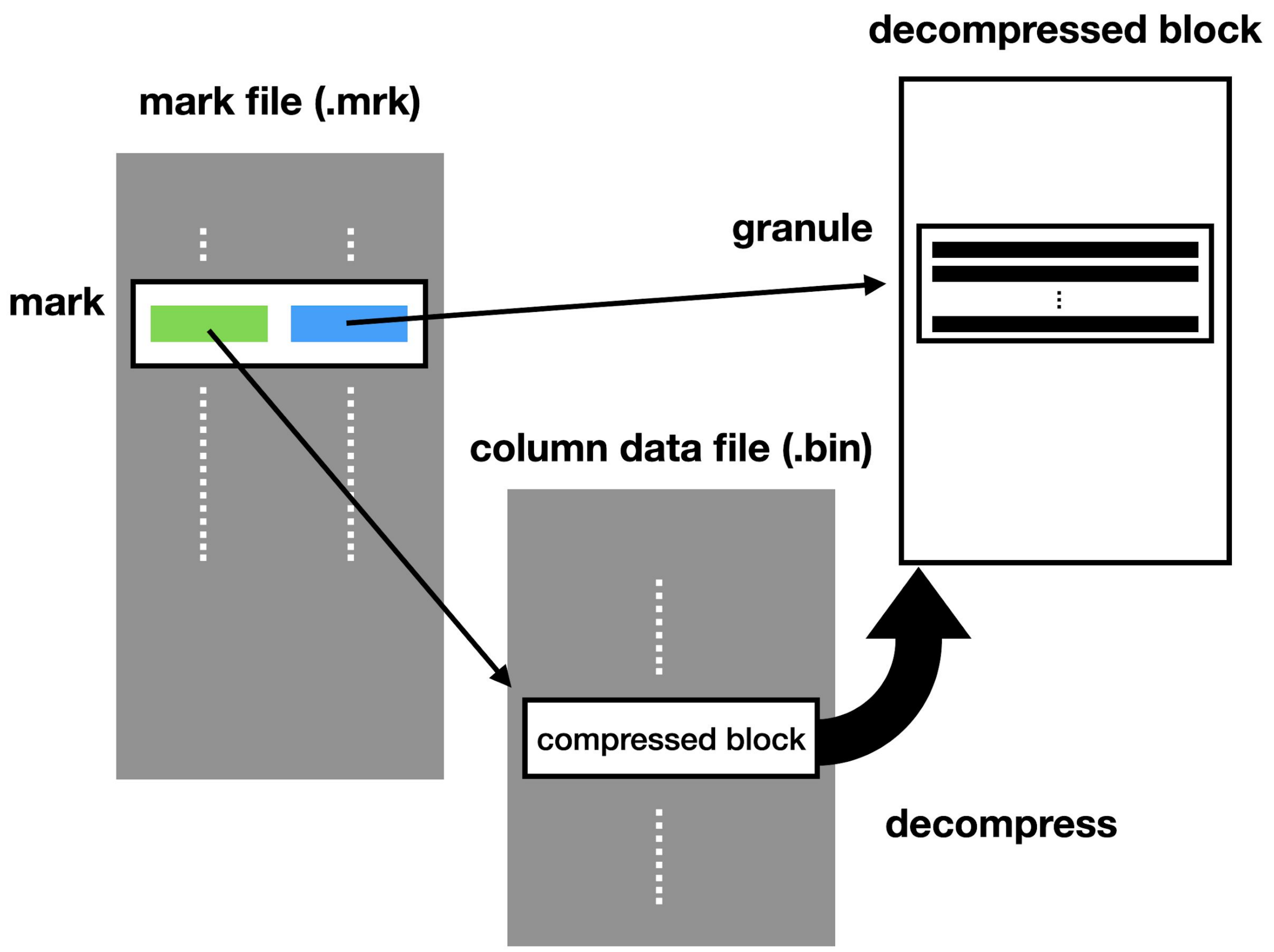
Agenda

- About us
- Road to ClickHouse adoption
- **Upstream patch: In-memory marks compression**

Background

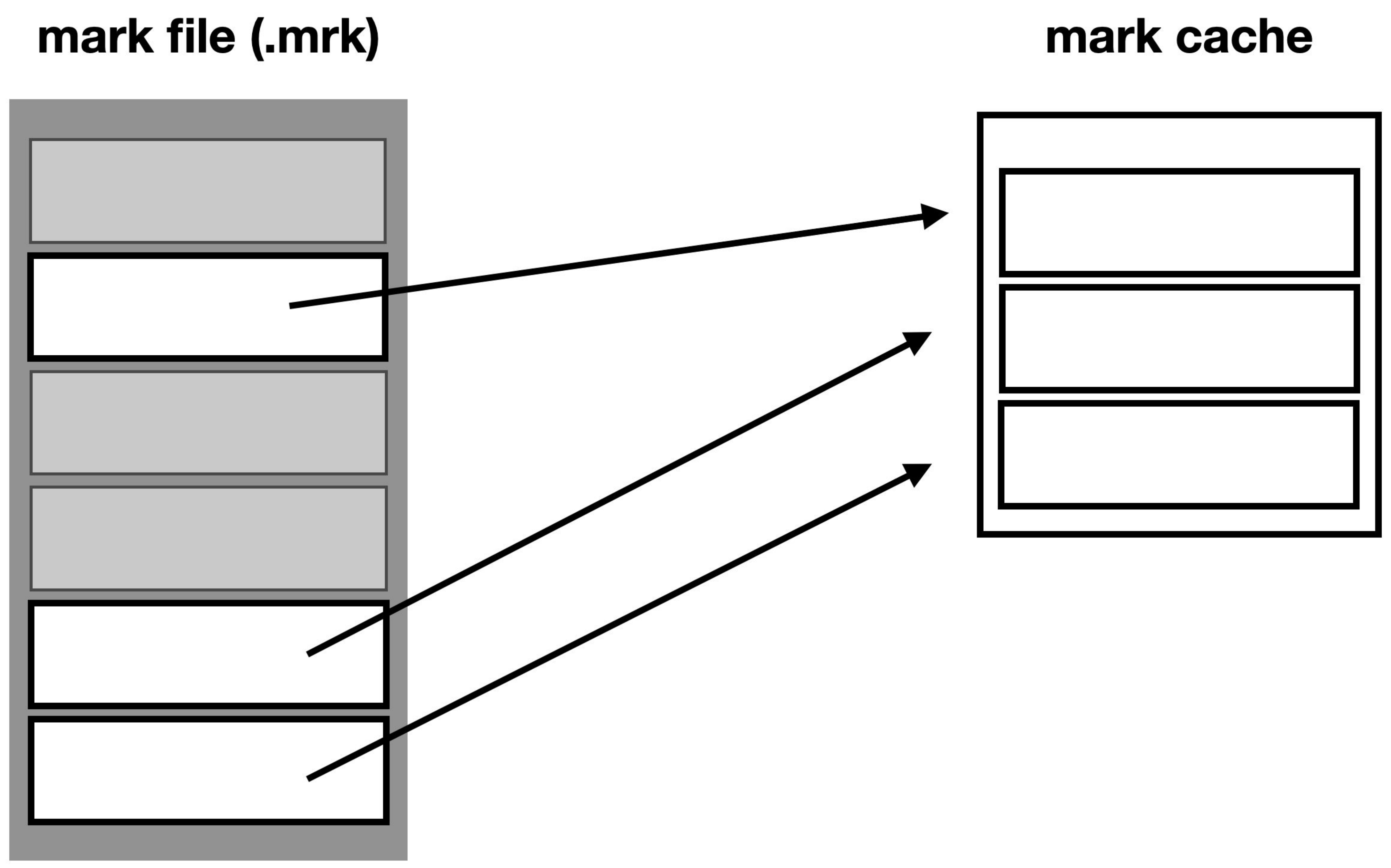
- ClickHouse uses offset pairs ("marks") to locate groups of rows in compressed data files

- Marks are stored in "mark files" and cached in memory



Problem

- Large queries read rows from many marks
  → poor cache performance
- At Ahrefs, want all marks to fit in memory for max performance
- Need more memory-efficient marks representation

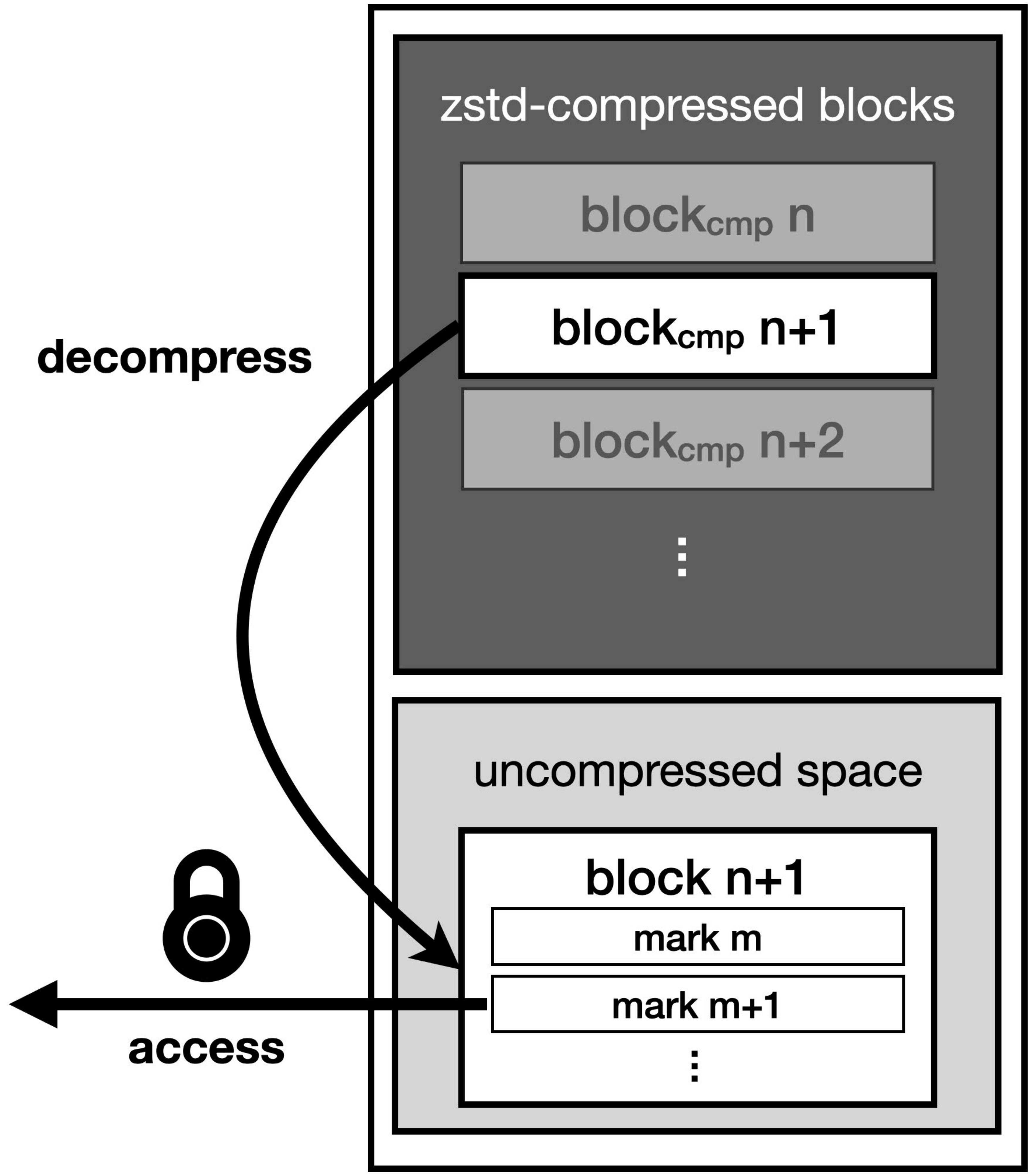


Insight

- The marks are suitable for compression
  - First offset is monotonically increasing
  - Second offset is almost always (but sometimes not) zero
  - But both stored as 64-bit integers (wasteful)

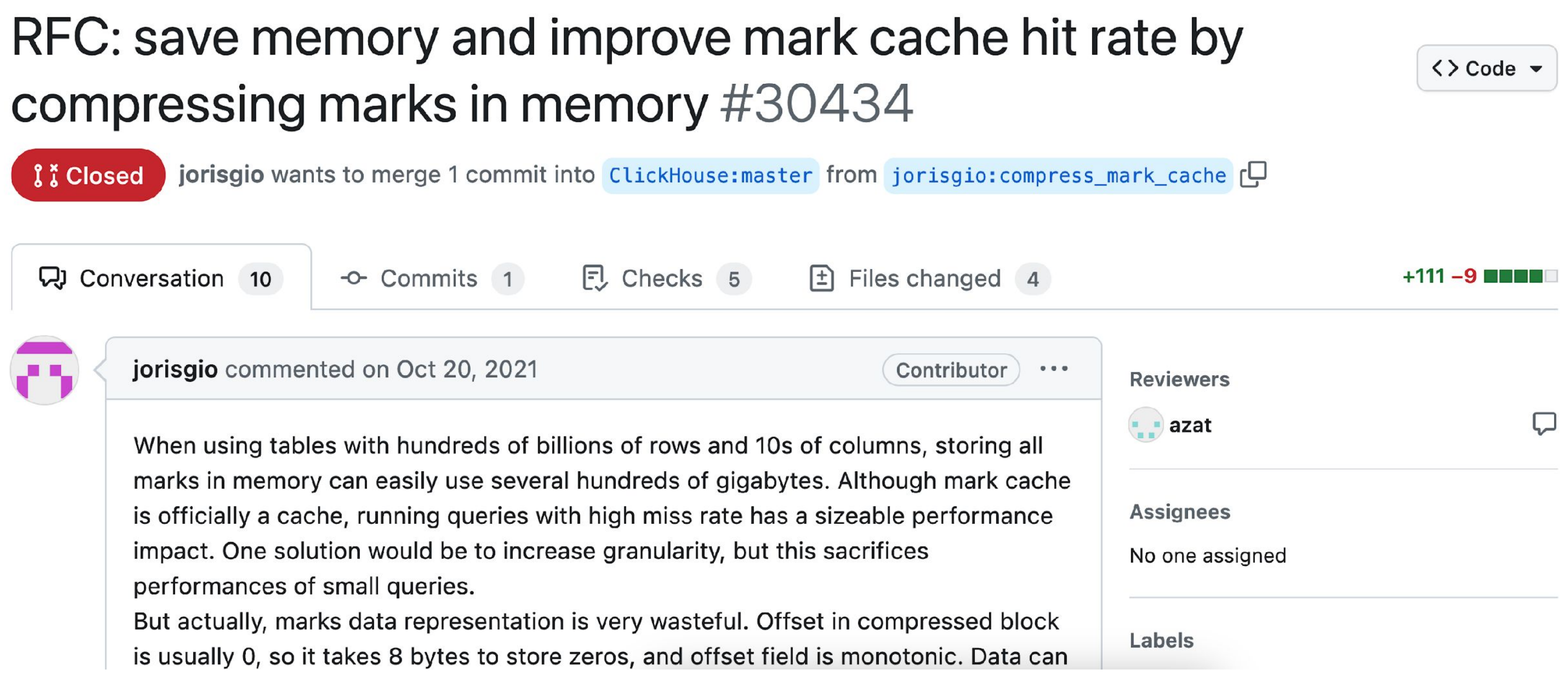
Early implementation at Ahrefs

- Split marks into blocks, store blocks compressed with zstd in memory
- Allocate space for one uncompressed block
- Guard access with mutex
    - Downside: contention if many threads
    - In practice, okay



Early implementation at Ahrefs

Oct 2021



RFC: save memory and improve mark cache hit rate by compressing marks in memory #30434

Closed jorisgio wants to merge 1 commit into `ClickHouse:master` from `jorisgio:compress_mark_cache`

jorisgio commented on Oct 20, 2021 — Contributor

When using tables with hundreds of billions of rows and 10s of columns, storing all marks in memory can easily use several hundreds of gigabytes. Although mark cache is officially a cache, running queries with high miss rate has a sizeable performance impact. One solution would be to increase granularity, but this sacrifices performances of small queries.
But actually, marks data representation is very wasteful. Offset in compressed block is usually 0, so it takes 8 bytes to store zeros, and offset field is monotonic. Data can

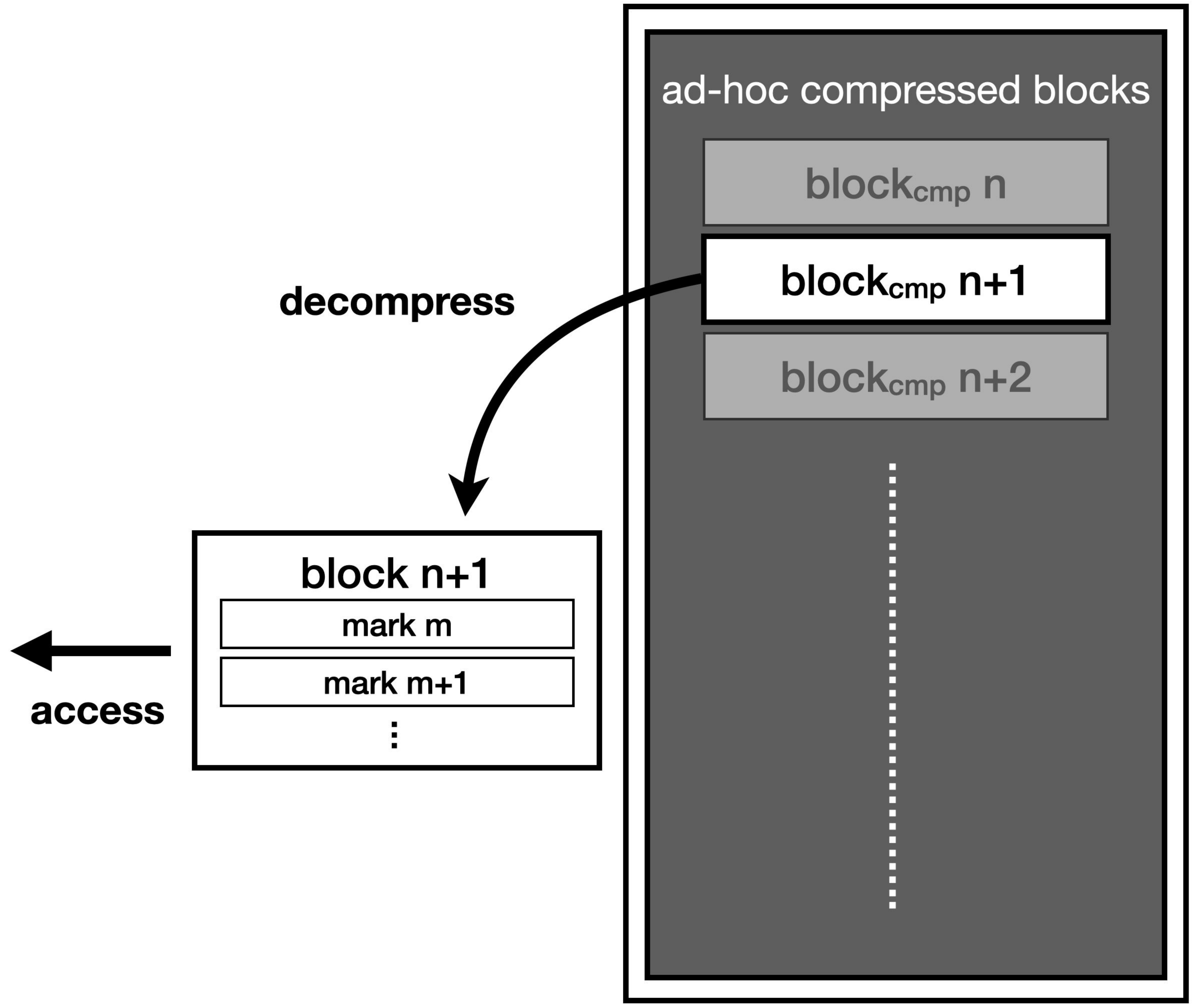
Reviewers: azat

Assignees: No one assigned

Labels

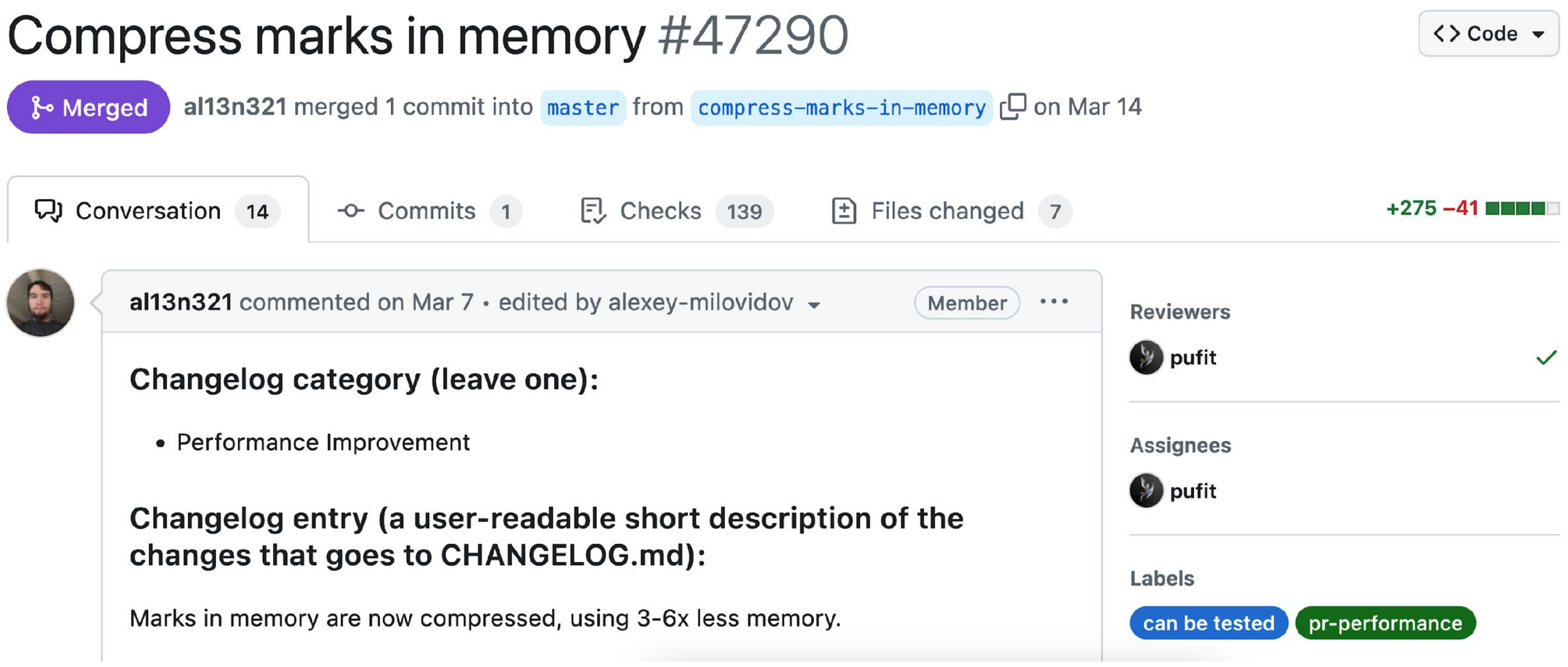
Eventual upstream solution

- Also split marks into blocks, but use ad-hoc compression
  - Store various deltas, bitpacked
- All in-memory marks compressed → no mutex
  - Downside: decompression for each mark access
- Result: 3-6x less memory than without compression



Eventual upstream solution

March 2023



Compress marks in memory #47290

Merged al13n321 merged 1 commit into `master` from `compress-marks-in-memory` on Mar 14

al13n321 commented on Mar 7 • edited by alexey-milovidov Member

Changelog category (leave one):

- Performance Improvement

Changelog entry (a user-readable short description of the changes that goes to CHANGELOG.md):

Marks in memory are now compressed, using 3-6x less memory.

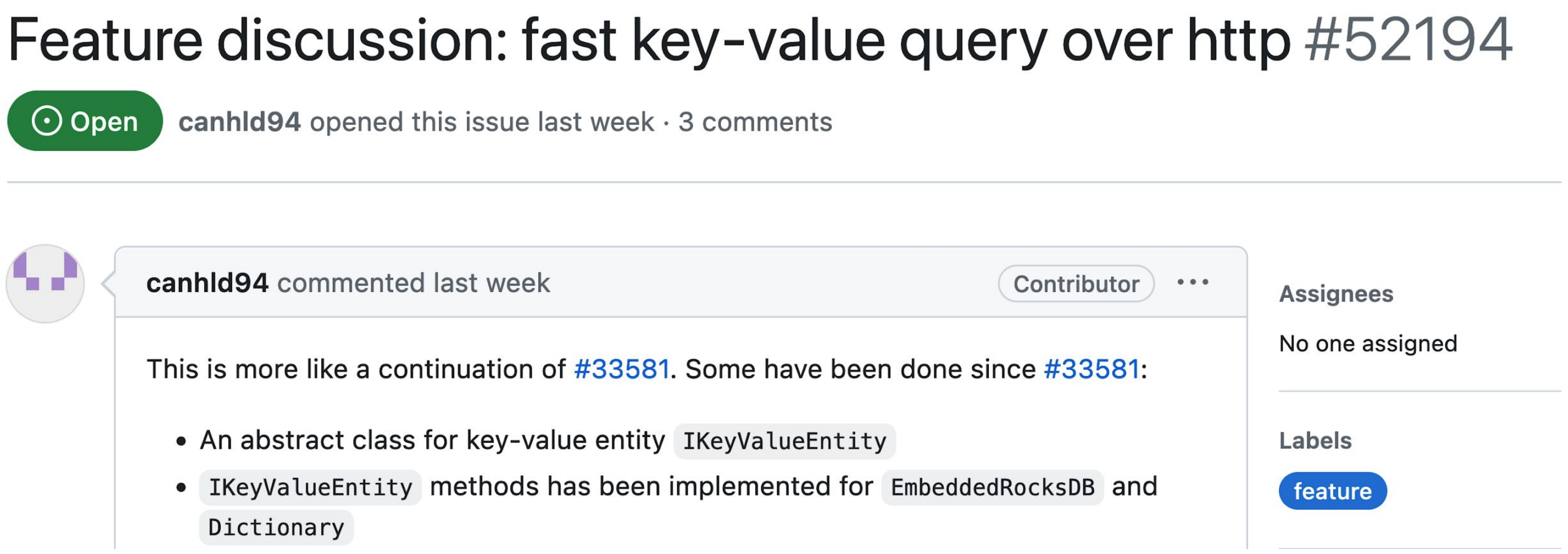
Reviewers
pufit ✓

Assignees
pufit

Labels
can be tested pr-performance

ahrefs

New: Key-value interface



Feature discussion: fast key-value query over http #52194

Open canhld94 opened this issue last week · 3 comments

canhld94 commented last week Contributor

This is more like a continuation of #33581. Some have been done since #33581:

- An abstract class for key-value entity `IKeyValueEntity`
- `IKeyValueEntity` methods has been implemented for `EmbeddedRocksDB` and `Dictionary`

Assignees

No one assigned

Labels

feature

ClickHouse at Ahrefs

- Great success overall
- Performance that meets our usage demands
- Active feature development and bug fixes, regular monthly releases

We’re hiring!

ahrefs.com/jobs

- D/C++ developers
- OCaml developers
- Data scientists