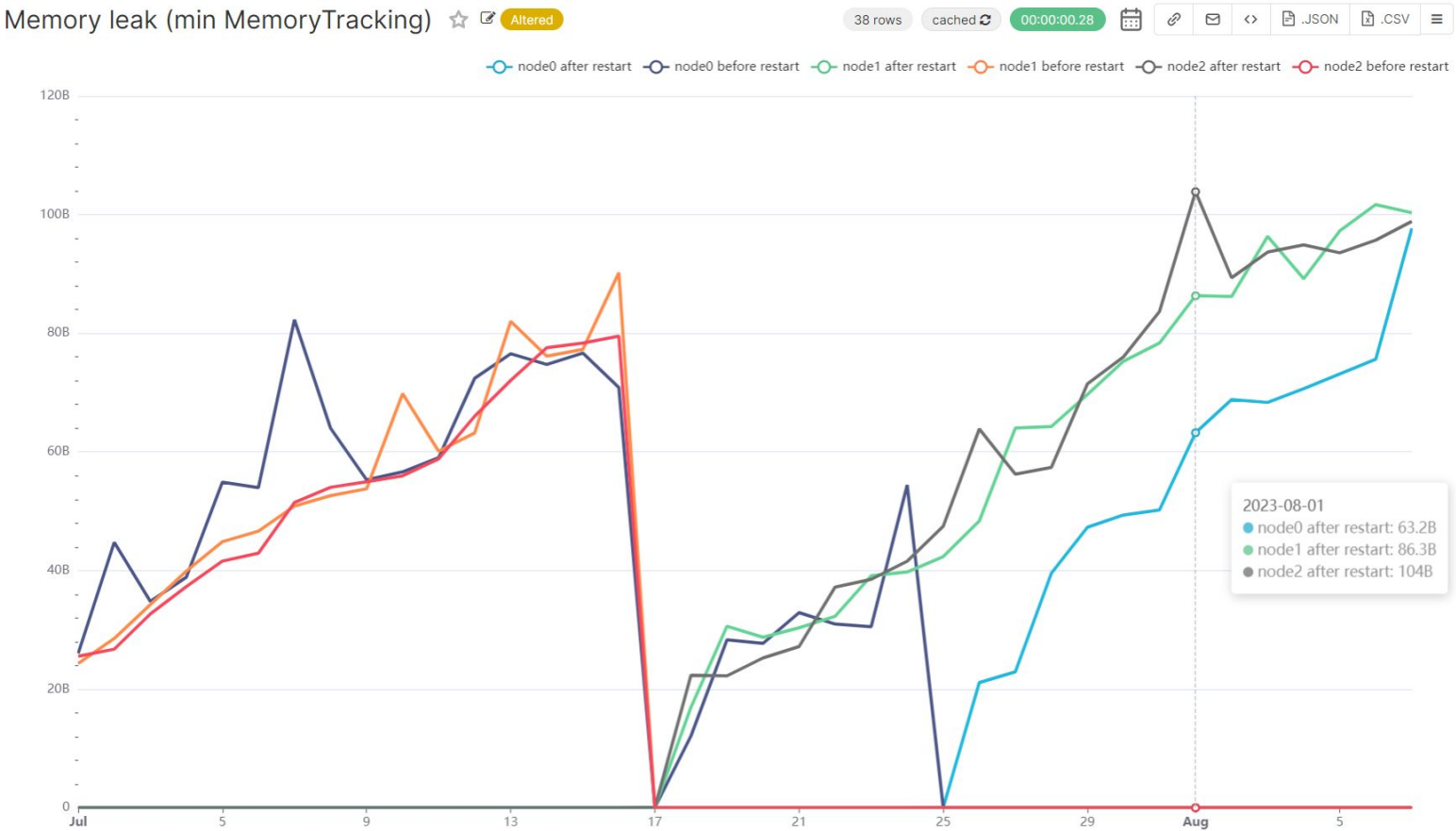
ClickHouse
memory usage introspection

Nikolai Kochetov

Memory Accounting is complicated

- Allocators cache memory internally
- Memory fragmentation
- Caches
- External libraries
- Direct memory mapping
- MADV_FREE pages
- System metrics are updated with delay
- cgroups and CAP_SYS_ADMIN
- Bugs

Memory leak?


Memory leak (min MemoryTracking)
Altered
38 rows
cached
00:00:00.28
.JSON
.CSV
node0 after restart
node0 before restart
node1 after restart
node1 before restart
node2 after restart
node2 before restart
120B
100B
80B
60B
40B
20B
0
Jul
5
9
13
17
21
25
29
Aug
5
2023-08-01
node0 after restart: 63.2B
node1 after restart: 86.3B
node2 after restart: 104B

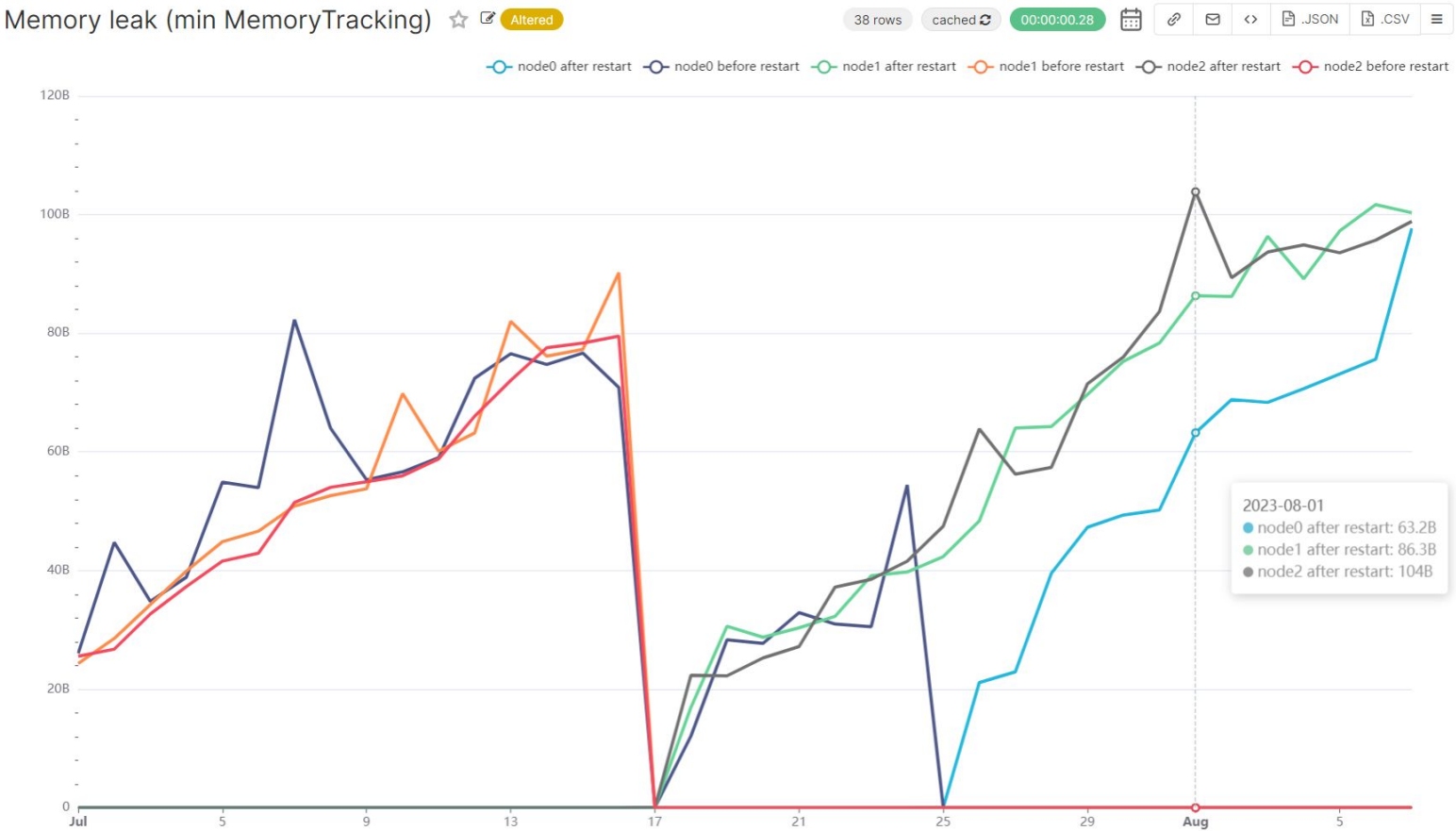
System tables

Database **system** contains ~100 tables with all kind of introspection

Documentation https://clickhouse.com/docs/en/operations/system-tables

```
SELECT
    formatReadableSize(memory_usage),
    formatReadableSize(ProfileEvents['ReadCompressedBytes']) AS ReadCompressedBytes
FROM system.query_log
WHERE (query_id = '6eb428d1-c9b2-4c4b-89fa-069383999edb') AND (type = 'QueryFinish')

Query id: 0923359e-2261-45b9-b46a-cf33a41e4d2a

┌─formatReadableSize(memory_usage)─┬─ReadCompressedBytes─┐
│ 51.67 GiB                        │ 124.45 MiB          │
└──────────────────────────────────┴─────────────────────┘
```


Mark cache


Memory leak (min MemoryTracking)
Altered
38 rows
cached
00:00:00.28
.JSON
.CSV
node0 after restart
node0 before restart
node1 after restart
node1 before restart
node2 after restart
node2 before restart
120B
100B
80B
60B
40B
20B
0
Jul
5
9
13
17
21
25
29
Aug
5
2023-08-01
node0 after restart: 63.2B
node1 after restart: 86.3B
node2 after restart: 104B

System tables

```
SELECT metric, formatReadableSize(value)
FROM system.metrics
WHERE metric LIKE '%MemoryTracking%'
```

```
┌─metric──────────────────────────┬─formatReadableSize(value)─┐
│ MemoryTracking                  │ 51.01 GiB                 │
│ MergesMutationsMemoryTracking   │ 3.46 GiB                  │
└─────────────────────────────────┴───────────────────────────┘
```

```
SELECT metric, value, formatReadableSize(value)
FROM system.asynchronous_metrics
WHERE metric LIKE 'MarkCache%'
```

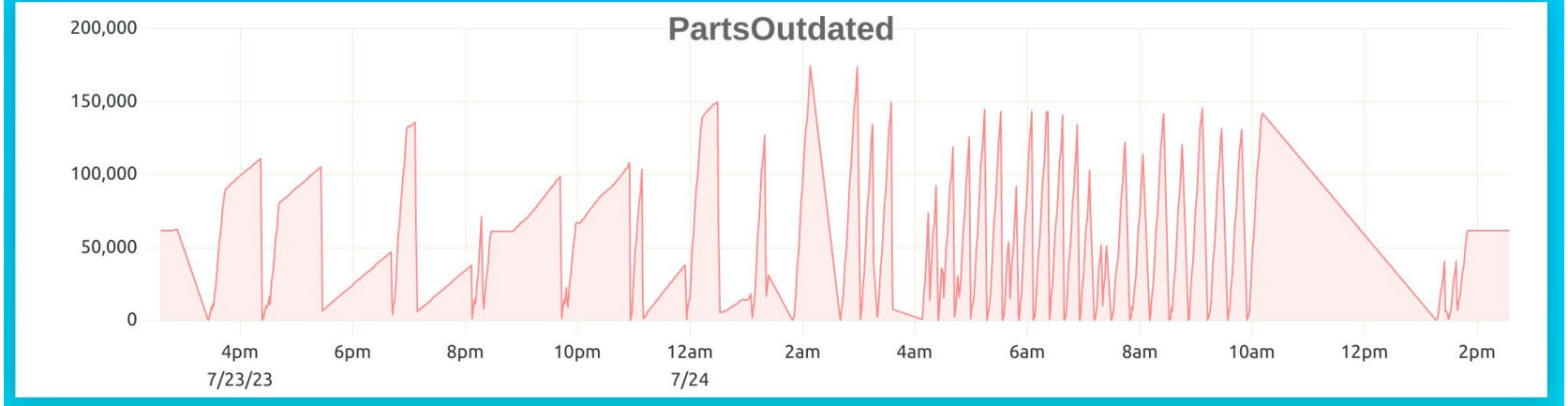
```
┌─metric─────────┬──────value─┬─formatReadableSize(value)─┐
│ MarkCacheFiles │      95703 │ 93.46 KiB                 │
│ MarkCacheBytes │ 10737330976 │ 10.00 GiB                 │
└────────────────┴────────────┴───────────────────────────┘
```

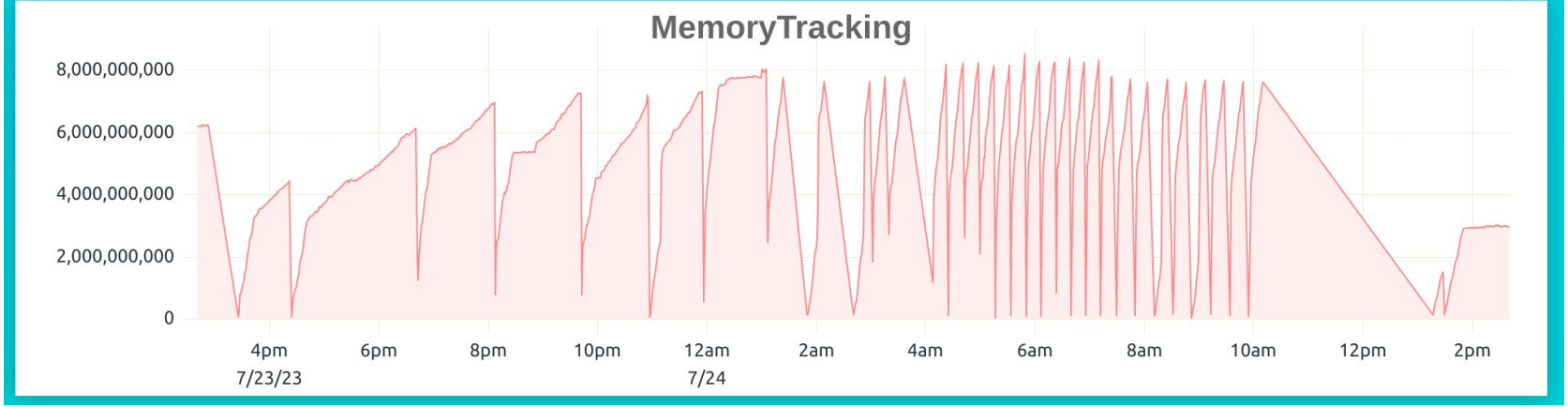

Misconfiguration and Misusage

- Invalid partitioning
- `parts_to_throw_insert, max_parts_in_total, max_partitions_per_insert_block`
- `max_server_memory_usage, max_server_memory_usage_to_ram_ratio`
- High insertion rate (except async_insert)
- Heavy background operations
- Frequent mutations, OPTIMIZE FINAL

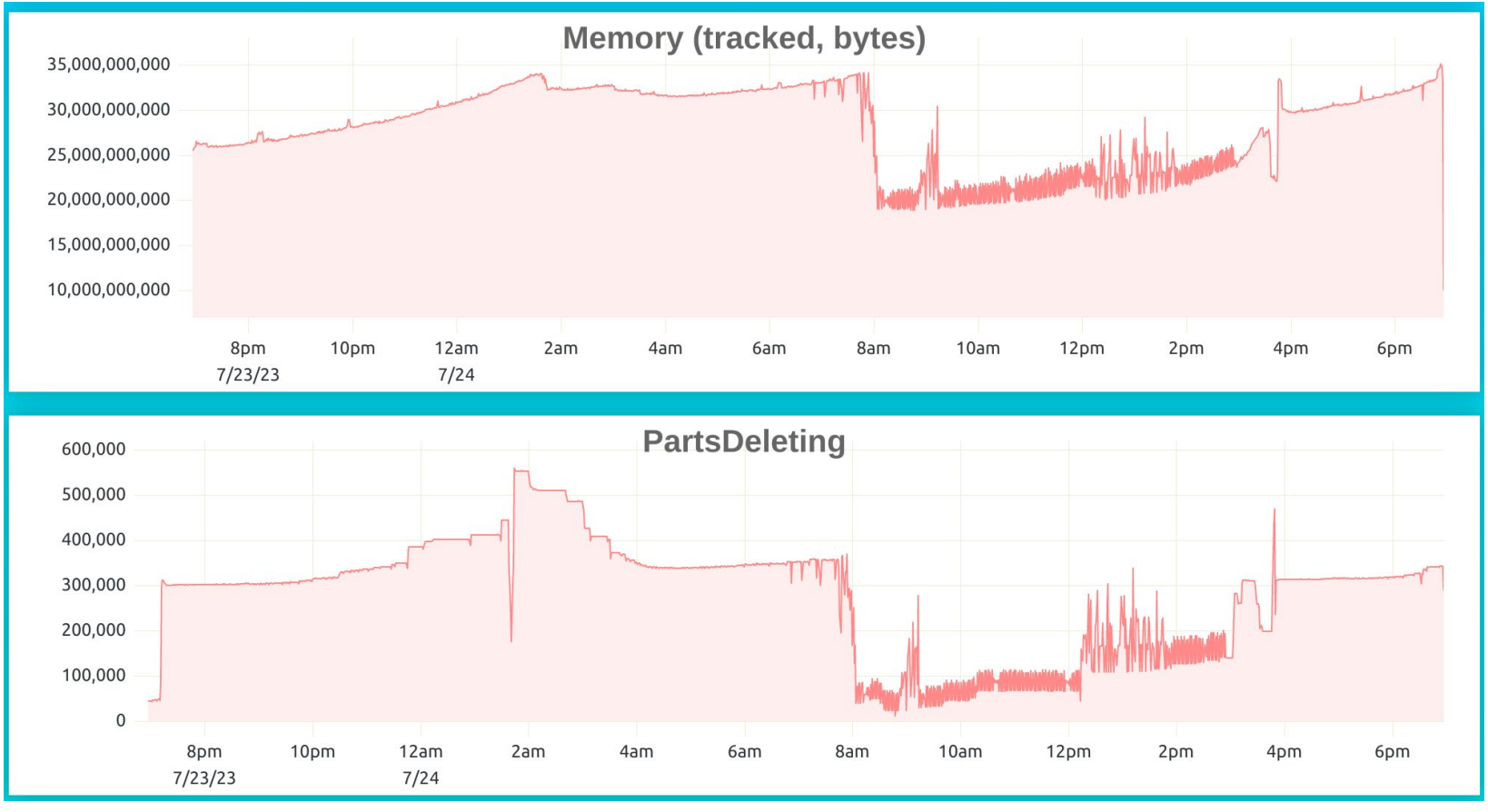
http://localhost:8123/dashboard

Many Outdated Parts





Many Deleting Parts



Understanding of Memory Consumption

- Poor usability (currently)
- Metrics and finding metrics correlation helps a lot
- Current activity

  `system.processes, system.user_processes, system.merges, system.metrics, system.asynchronous_metrics`

- Historical activity

  `system.metric_log, system.asynchronous_metric_log, system.query_log, system.text_log`

Asynchronous Insert Memory Drift

Run a query with asynchronous insert 100 times with N = 1 million rows

```
INSERT INTO async_table SETTINGS async_insert=1, wait_for_async_insert=1 VALUES (1), …, (N)
```

User Memory usage (before)

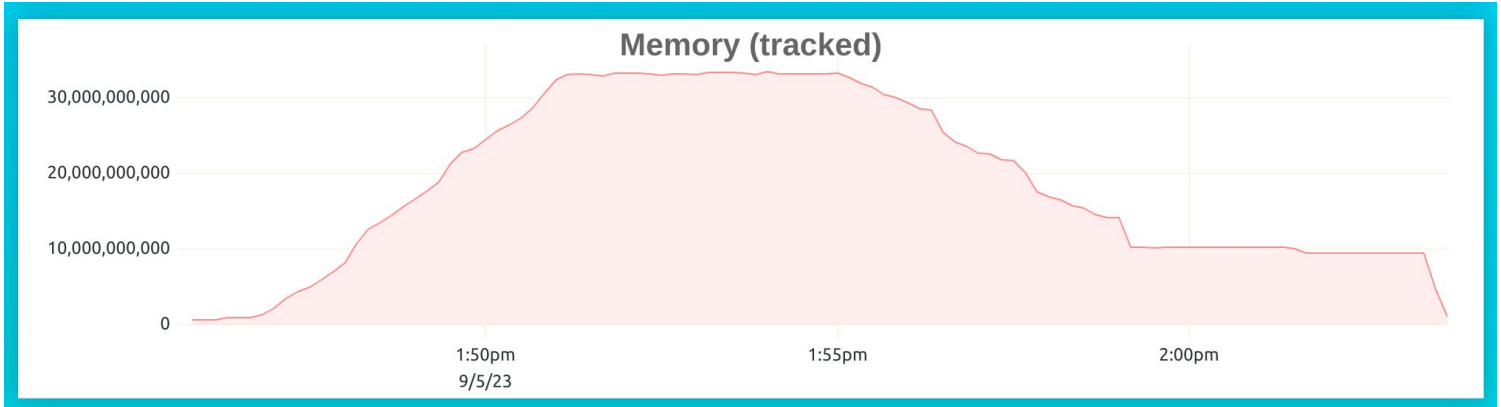
```
SELECT formatReadableSize(memory_usage)
FROM system.user_processes
WHERE user = 'A'
```

```
┌─formatReadableSize(memory_usage)─┐
│ 32.70 KiB                        │
└──────────────────────────────────┘
```

User Memory usage (after)

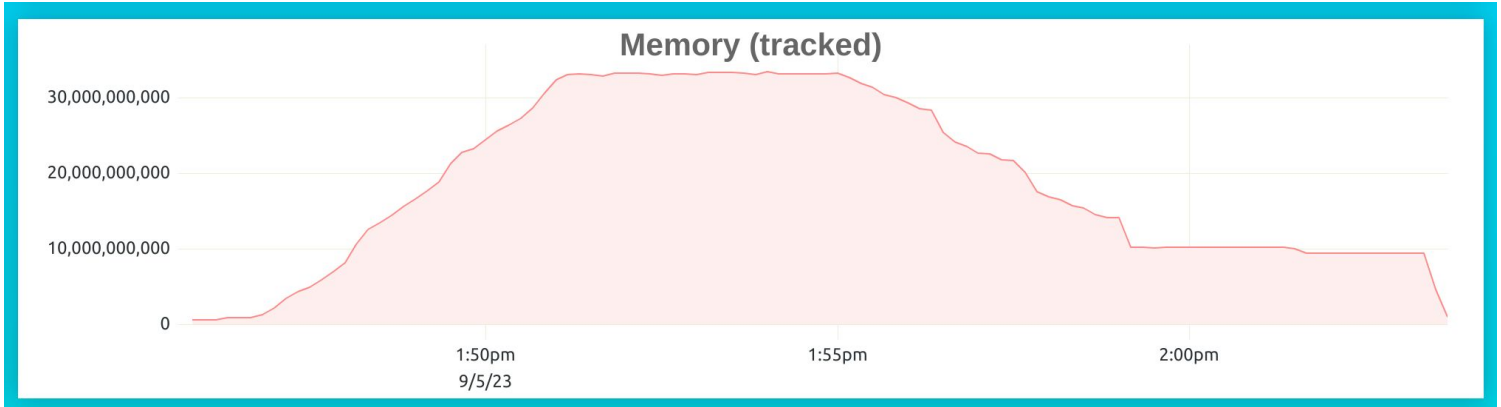
```
SELECT formatReadableSize(memory_usage)
FROM system.user_processes
WHERE user = 'A'
```

```
┌─formatReadableSize(memory_usage)─┐
│ 12.21 GiB                        │
└──────────────────────────────────┘
```



Asynchronous Insert Memory Drift

- Found (mainly) by checking hypothesis and reading the code
- Fixed #46622
- Better introspection is required



Tracking Allocations

```
SELECT
    query_id,
    trace,
    size,
    ptr
FROM system.trace_log
WHERE (trace_type = 'MemorySample') AND (query_id != '') AND (size > 1000000)
LIMIT 1

Query id: 9a555d30-2f8e-49a9-8349-94875ebfc2a8

Row 1:
──────
query_id: test_async_insert_2
trace:
[177950546,177832181,177581809,292324046,292355899,292360407,292360640,271187286,271208959,284578095,2845923
13,285024728,302263591,302264845,302681767,302672578,139962546288137,139962545389875]
size:     1572864
ptr:      139955000020096
```


Allocation StackTrace

```
SELECT arrayStringConcat(arrayMap(x -> demangle(addressToSymbol(x)), trace), '\n')
SETTINGS allow_introspection_functions = 1
```

```
AllocationTrace::onAllocImpl(void*, unsigned long) const
operator new(unsigned long)
DB::Tokens::Tokens(char const*, char const*, unsigned long, bool)
DB::tryParseQuery(...)
DB::parseQueryAndMovePosition(...)
DB::executeQueryImpl(...)
DB::executeQuery(...)
DB::HTTPHandler::processQuery(...)
DB::HTTPHandler::handleRequest(DB::HTTPServerRequest&, DB::HTTPServerResponse&)
DB::HTTPServerConnection::run()
Poco::Net::TCPServerConnection::start()
Poco::Net::TCPServerDispatcher::run()
Poco::PooledThread::run()
Poco::ThreadImpl::runnableEntry(void*)
```


FlameGraphs

Query example (from the benchmark)

```
SET memory_profiler_sample_probability=1, max_untracked_memory=1;

SELECT SearchPhrase, COUNTDistinct(UserID) AS u
FROM hits WHERE SearchPhrase != ''
GROUP BY SearchPhrase ORDER BY u DESC LIMIT 10
```

Use `flamegraph.pl` from https://github.com/brendangregg/FlameGraph and `flameGraph` function

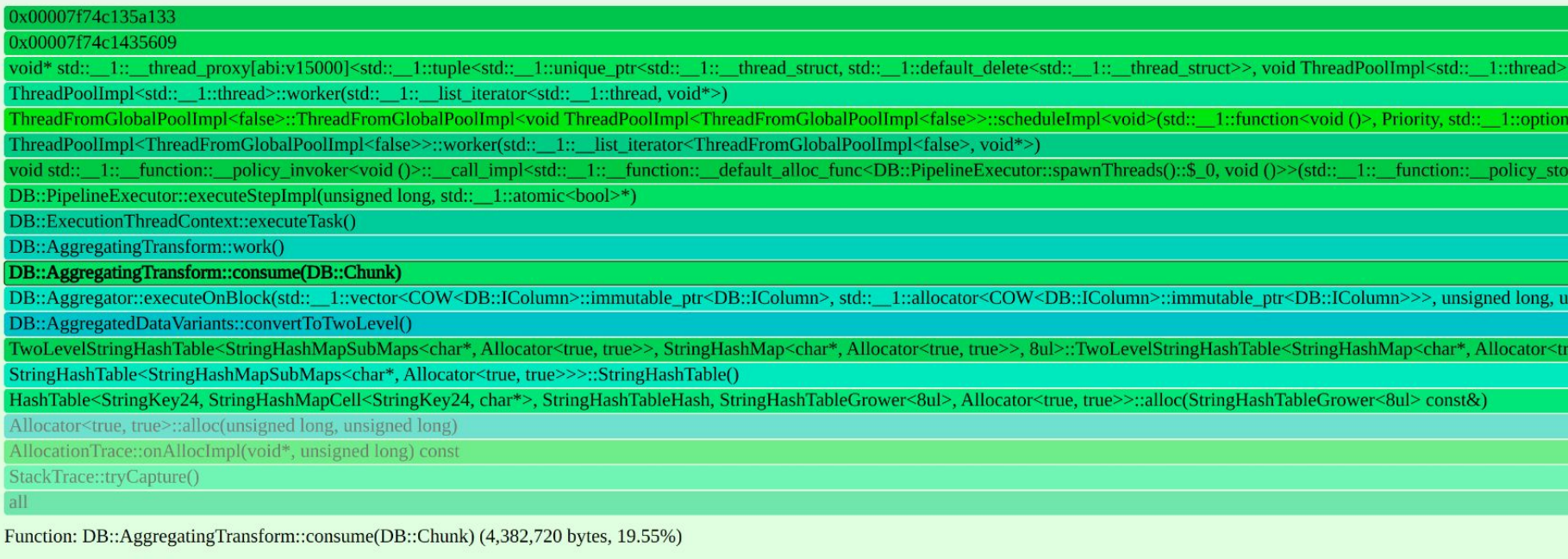
```
SELECT arrayJoin(flameGraph(trace, size)) FROM system.trace_log WHERE trace_type = 'MemorySample' AND query_id = '..'

output | ./FlameGraph/flamegraph.pl --countname=bytes --color=mem > flame_mem.svg
```

Examples are in this PR

All Query Allocations

```
SELECT arrayJoin(flameGraph(trace, size)) FROM system.trace_log WHERE trace_type = 'MemorySample' AND query_id = '..'
```

Some memory is used to build hash table for aggregation



FlameGraphs

Example: enable uncompressed cache

```
SET memory_profiler_sample_probability=1, max_untracked_memory=1, use_uncompressed_cache=1,
merge_tree_max_rows_to_use_cache=100000000000, merge_tree_max_bytes_to_use_cache=1000000000000;

SELECT SearchPhrase, COUNTDistinct(UserID) AS u
FROM hits WHERE SearchPhrase != ''
GROUP BY SearchPhrase ORDER BY u DESC LIMIT 10;
```

Now, query allocated and not released 100M

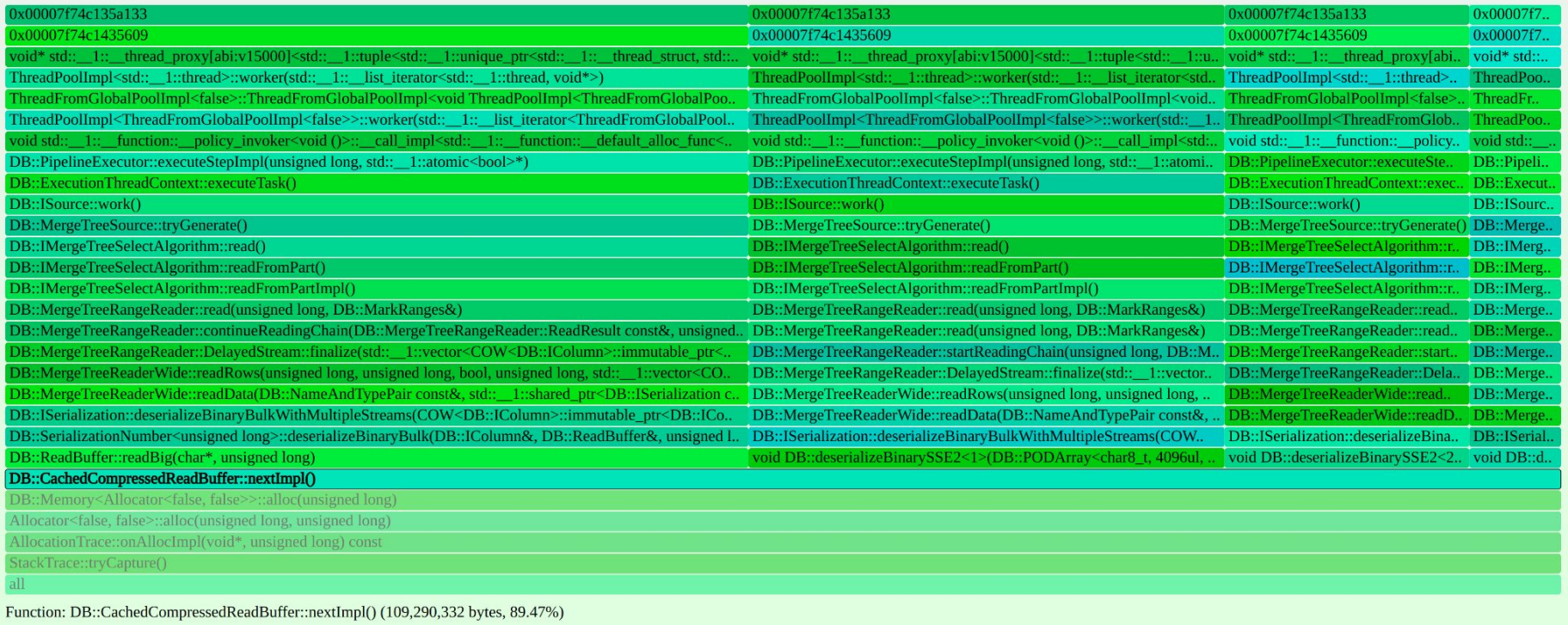
```
SELECT formatReadableSize(sum(size)) FROM system.trace_log WHERE (query_id = '..') AND (trace_type = 'MemorySample')

┌─formatReadableSize(sum(size))─┐
│ 108.90 MiB                    │
└───────────────────────────────┘
```

Find only those allocations which were not released by query

```
SELECT arrayJoin(flameGraph(trace, size, ptr)) FROM system.trace_log WHERE trace_type = 'MemorySample' AND query_id = '...'
```


Unreleased Query Allocations

Memory is hold by uncompressed cache



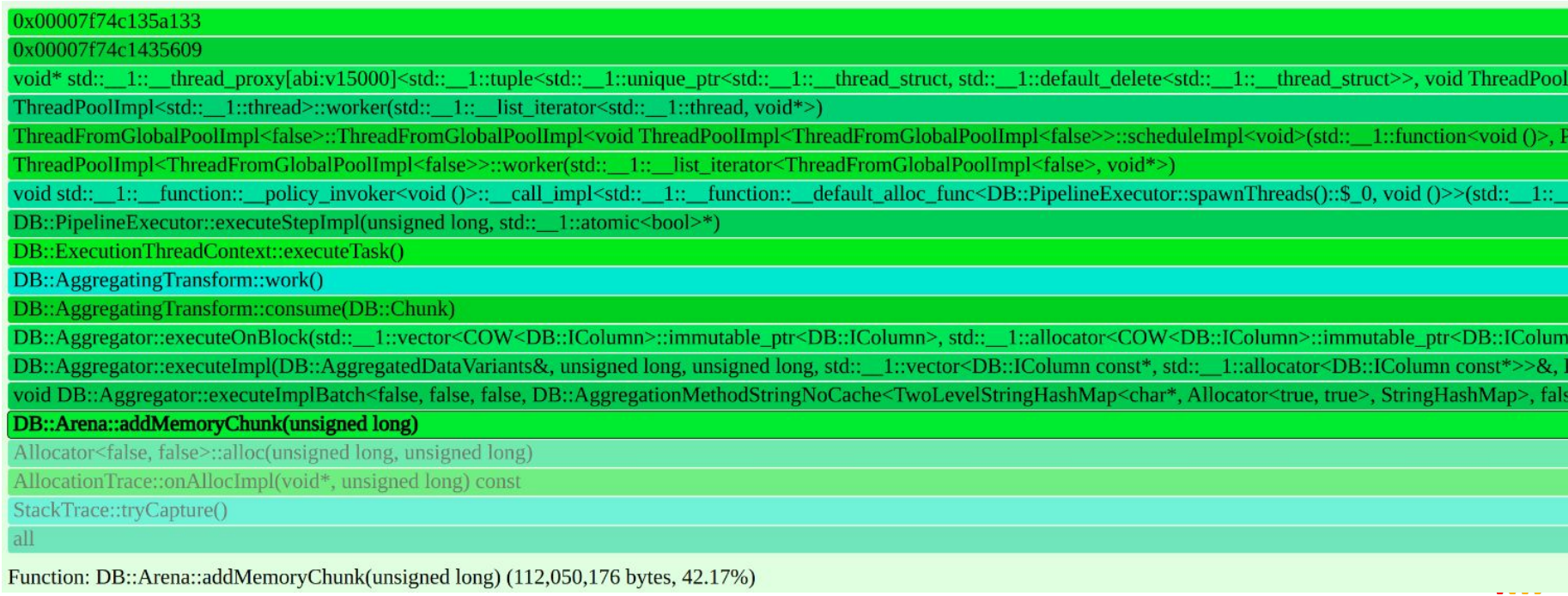
Peak of Memory Usage

```
SELECT
    toStartOfInterval(event_time_microseconds, toIntervalMillisecond(500)) AS t,
    max(sum) AS memory,
    formatReadableSize(memory)
FROM
(
    SELECT
        event_time_microseconds,
        sum(size) OVER (ORDER BY event_time_microseconds ASC) AS sum
    FROM system.trace_log
    WHERE (query_id = '..') AND (trace_type = 'MemorySample')
)
GROUP BY t
ORDER BY t ASC
```

```
┌──────────────────────t─┬─────memory─┬─formatReadableSize(memory)─┐
│ 2023-09-04 20:11:33.500 │      98272 │ 95.97 KiB                  │
│ 2023-09-04 20:11:34.000 │     246872 │ 241.09 KiB                 │
│ 2023-09-04 20:11:34.500 │     555200 │ 542.19 KiB                 │
│ 2023-09-04 20:11:35.000 │ 268265964 │ 255.84 MiB                 │
│ 2023-09-04 20:11:35.500 │ 284551358 │ 271.37 MiB                 │
│ 2023-09-04 20:11:36.000 │ 149204233 │ 142.29 MiB                 │
└─────────────────────────┴────────────┴────────────────────────────┘
```

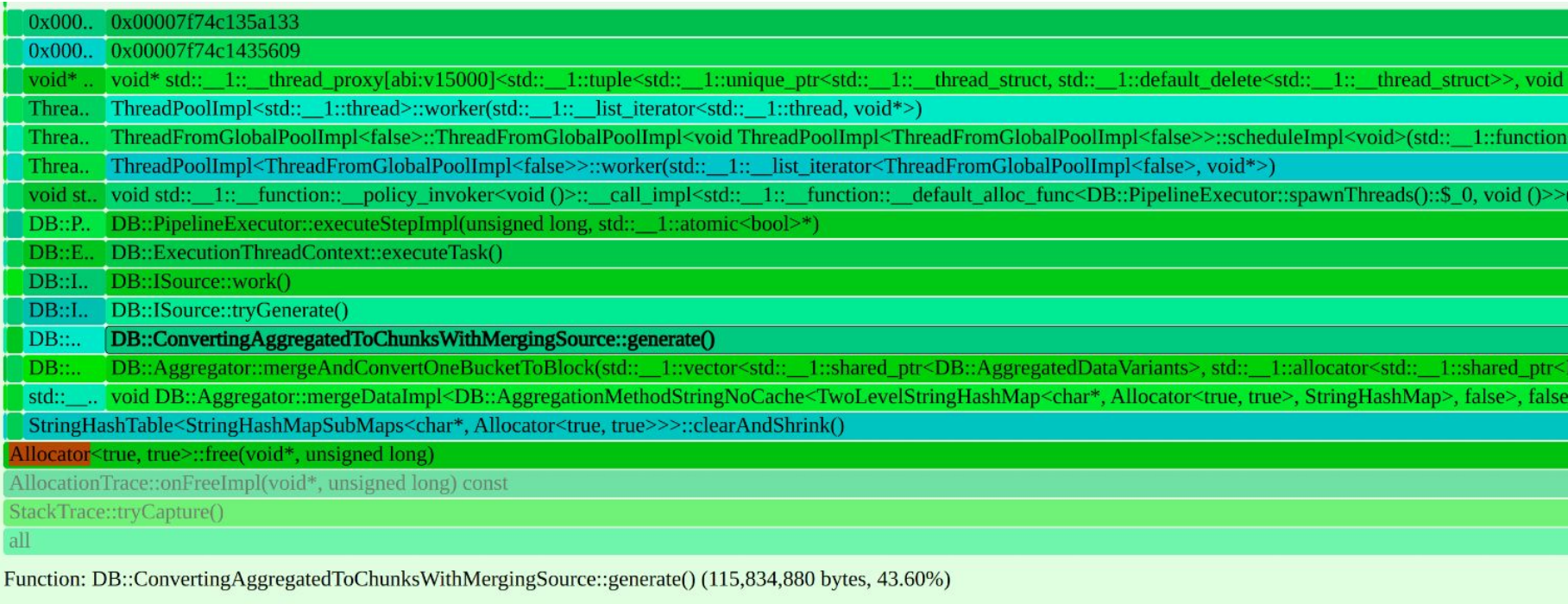

Query Allocations Snapshot

```
SELECT arrayJoin(flameGraph(trace, size,  ptr)) FROM system.trace_log
WHERE trace_type = 'MemorySample' AND query_id = '..' AND  event_time < 'timestamp'
```

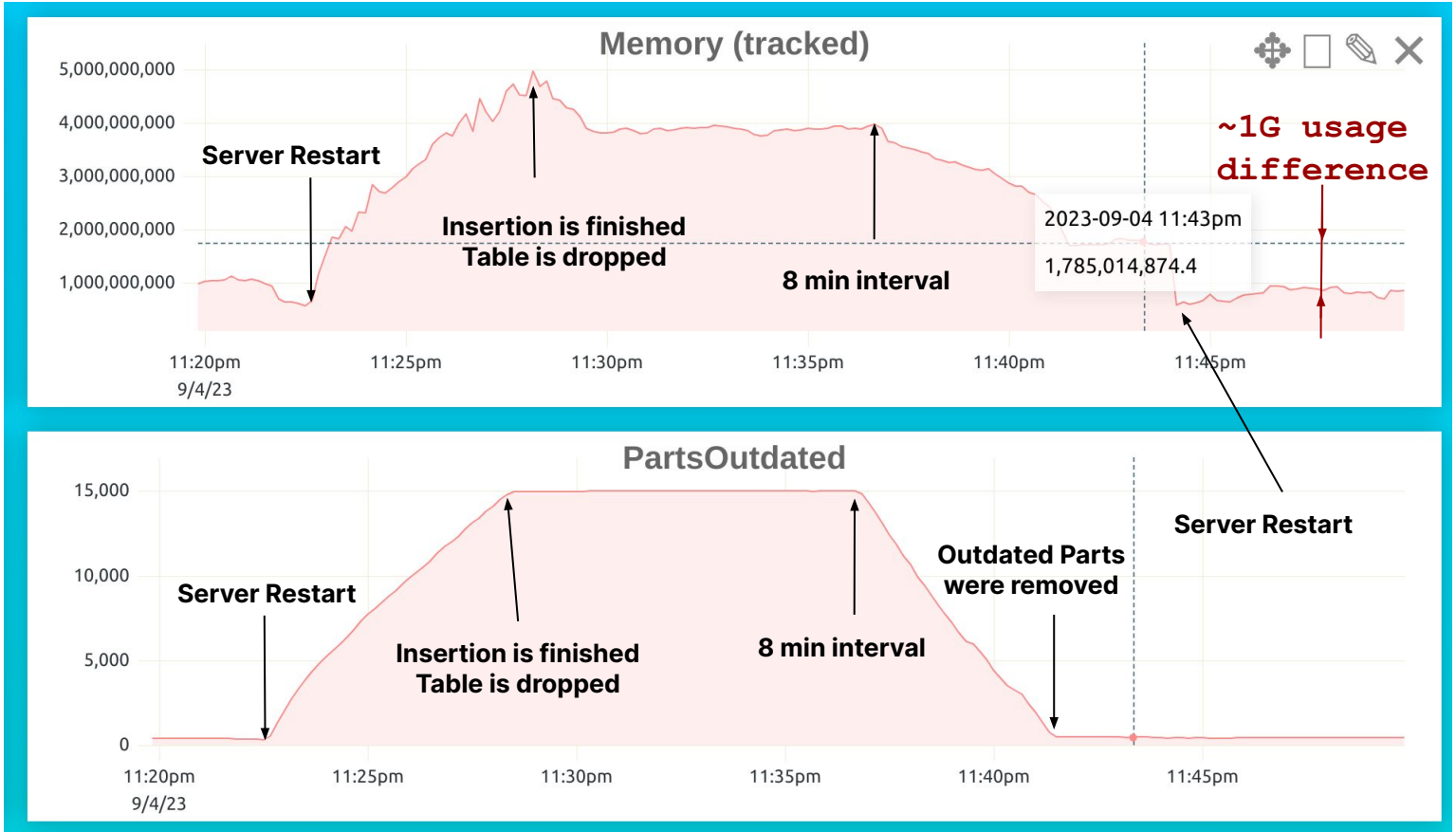


Query Deallocations Snapshot

```
SELECT arrayJoin(flameGraph(trace, size,  -ptr)) FROM system.trace_log
WHERE trace_type = 'MemorySample' AND query_id = '..' AND  event_time >= 'timestamp'
```



Insert + Drop test



Insert + Drop test

Hard to debug with memory profiler

- A lot of allocations
- system.trace_log traces itself
- High memory consumption was detected in narrow size class

More introspection features

- Settings `memory_profiler_sample_min/max_allocation_size` to limit tracked allocations size
- Table `system.jemalloc_bins` to show statistics from jemalloc arenas

Fix #51732

Summary

- Metrics correlation is useful
- System tables can show current and historical activity
- Recent new features
  - `system.user_processes`
  - `memory_profiler_sample_min/max_allocation_size`
  - `system.jemalloc_bins`
  - `flameGraph`

- With system.trace_log we can build a snapshot of query allocations at any point of time
- Flamegraphs are built with an external script
- Need to support flamegraphs it in /dashboard

Thank You!