How session detection is implemented in ClickHouse

...and why there was no 90% codebase rewrite...

...and some other stuff.

Agenda

Part I: The story

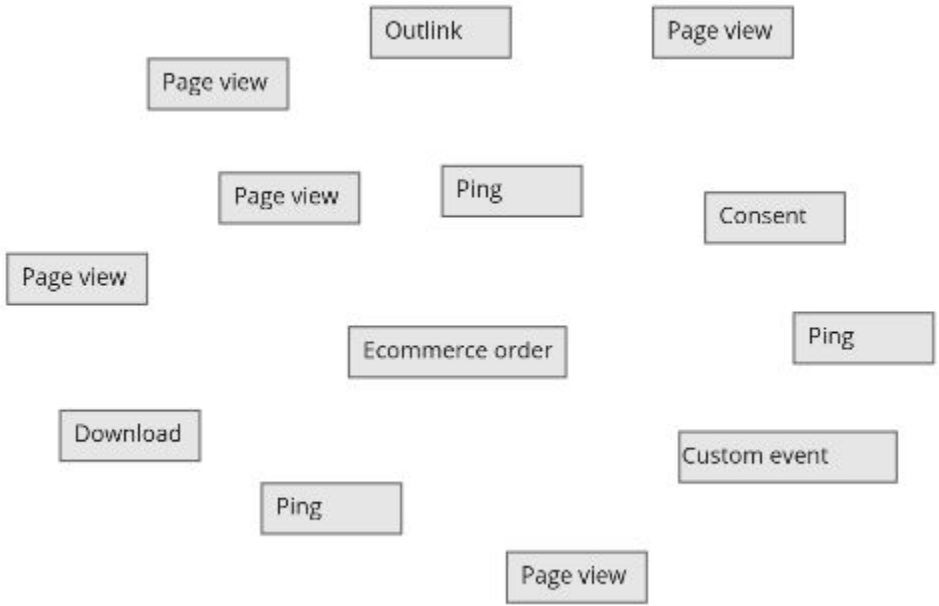
Part II: Detailed comparison

Part I

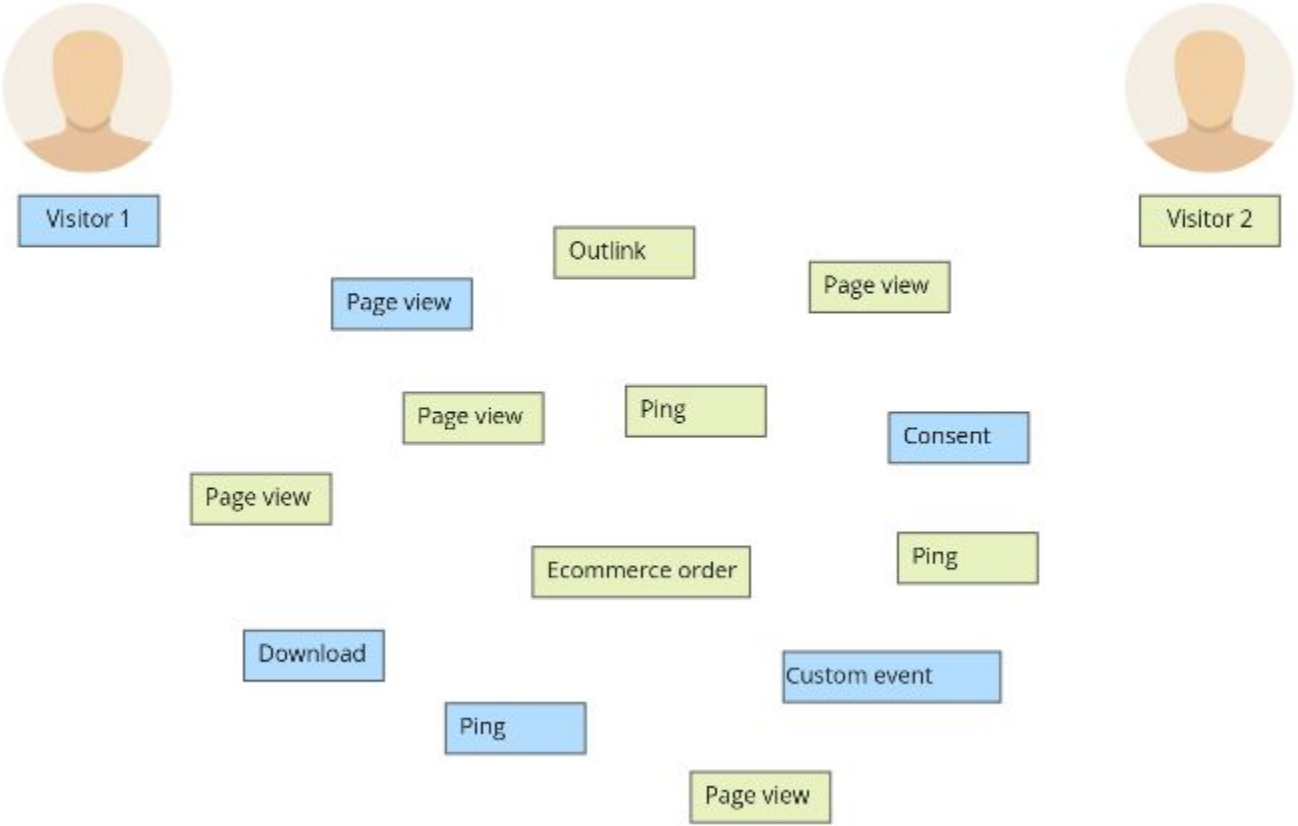
Background

- Trucker team currently creates a new tracker for Piwik PRO Analytics Suite
- The goal of the tracker is to receive tracking requests and preprocess them
- One of parts of preprocessing is session detection

What is session detection?

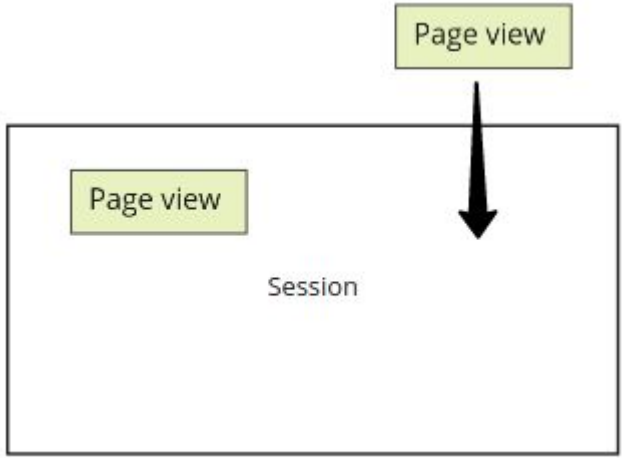


What is session detection?



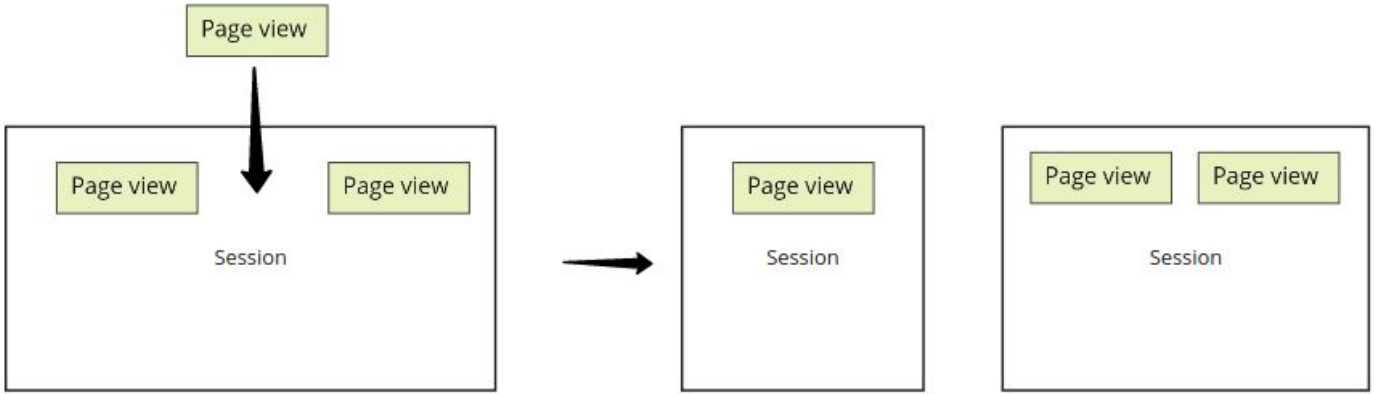
Session operations

Add event to session



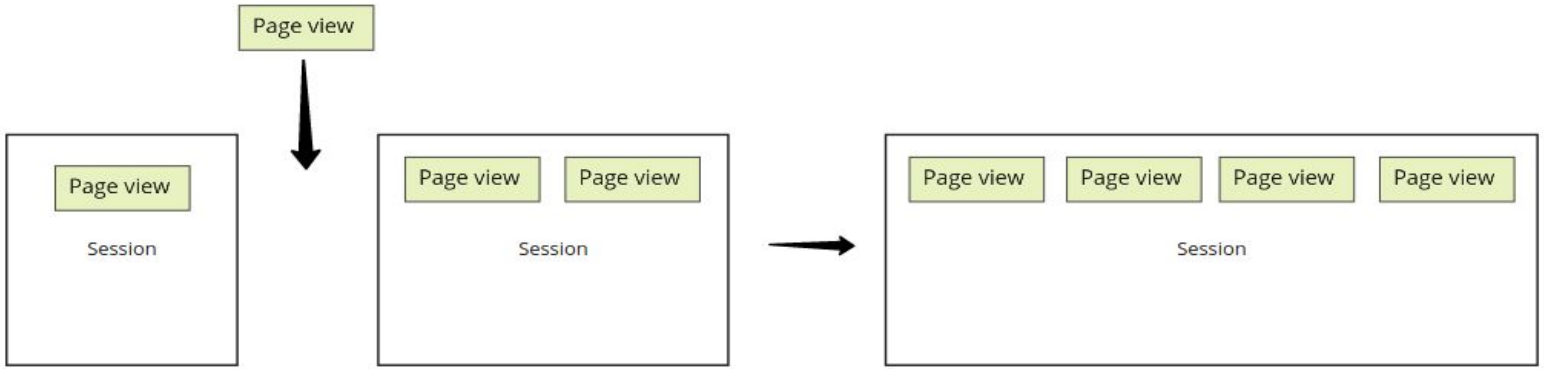
Session operations

Split session



Session operations

Merge session

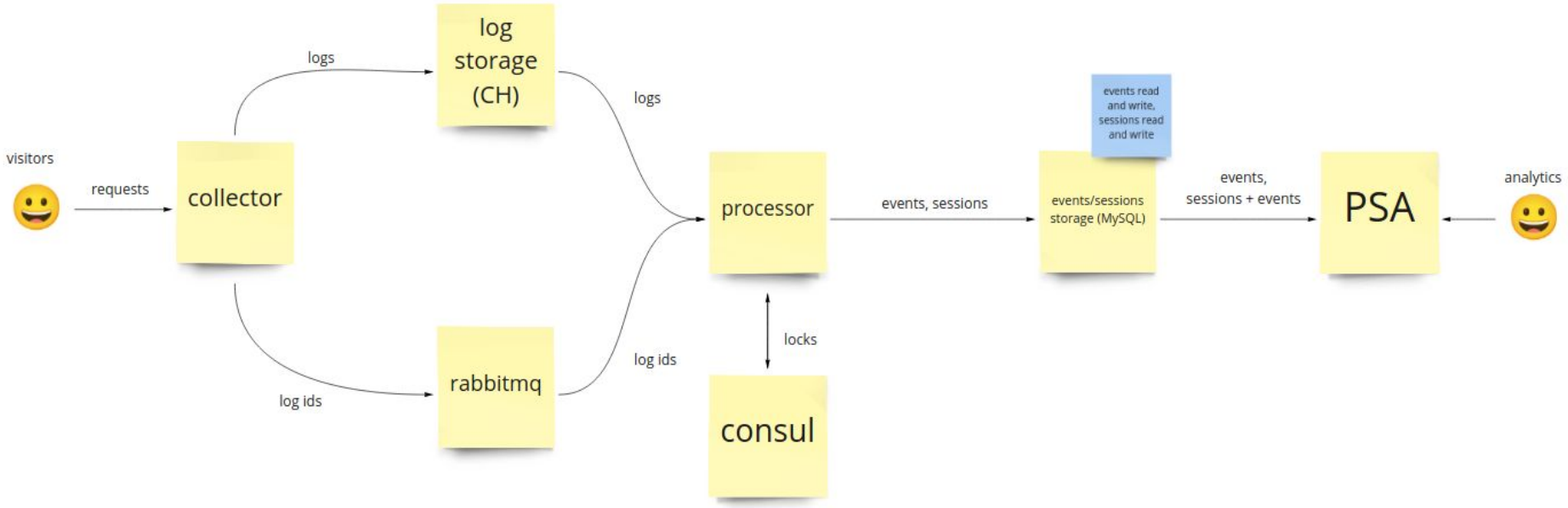


Approach #1

- We wanted get rid of MySQL
- ClickHouse was a candidate
- We chose MySQL

Approach #1

- Use MySQL
- Update session each time an event is processed



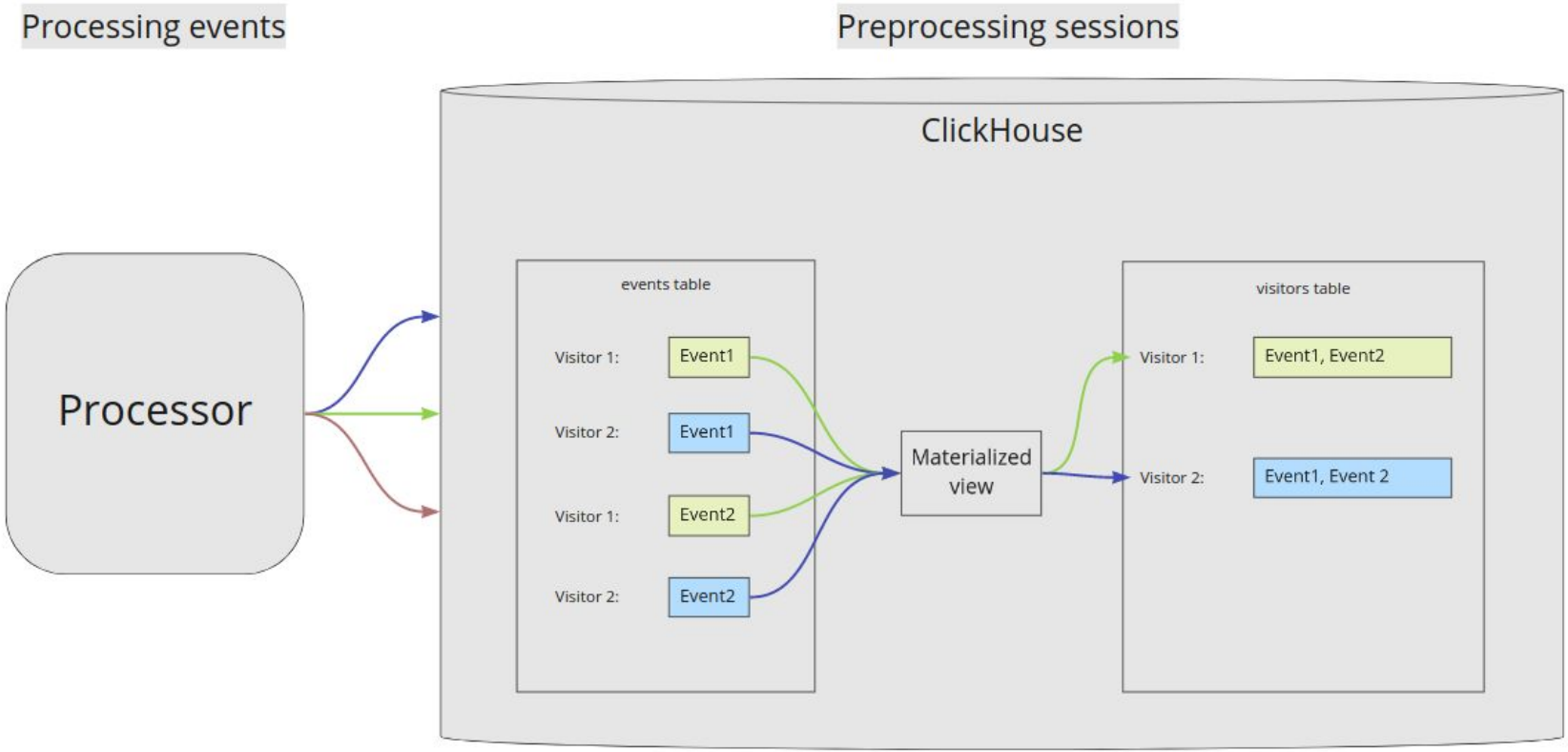
Approach #1

We made performance tests and it was far below expectations...

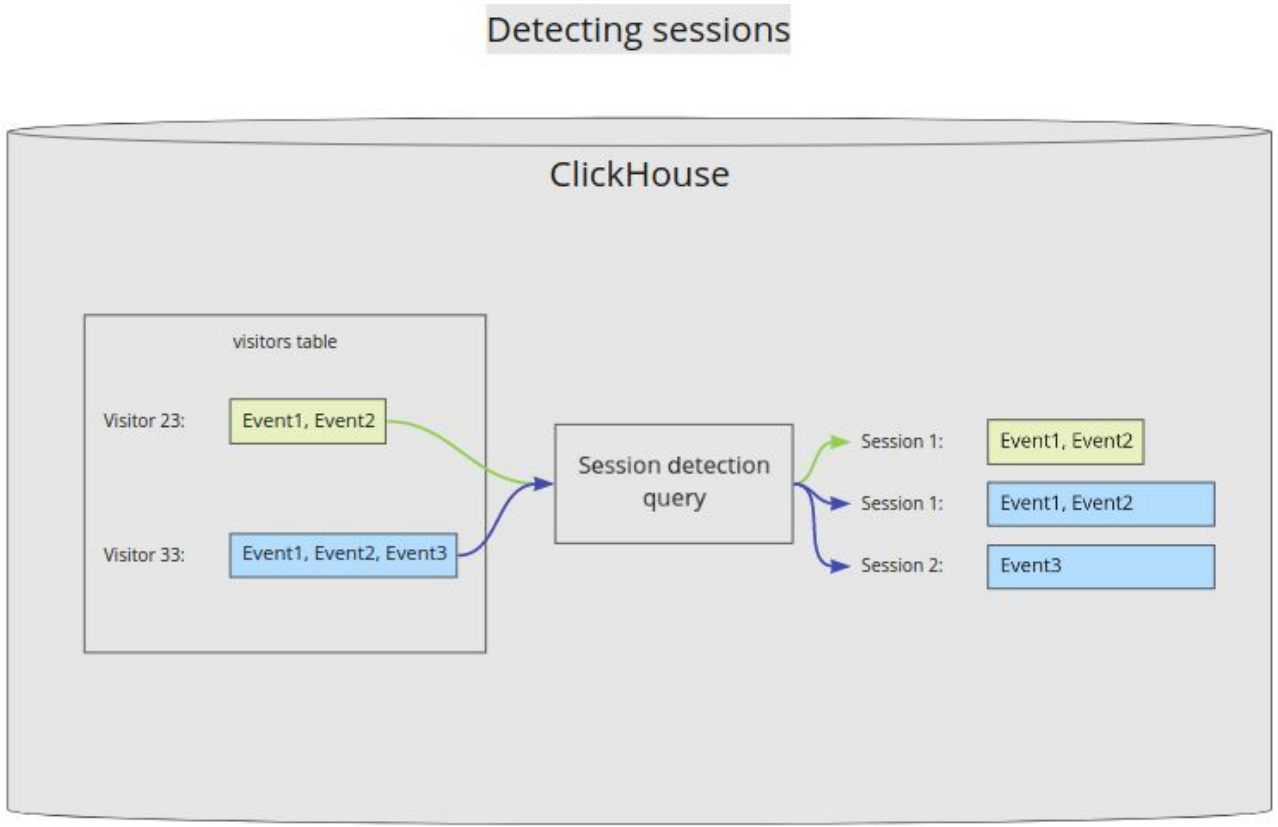
Attempt #2

- Use ClickHouse
- Delay session detection until events are gathered or do not detect them at all
- Optimize disk usage

Attempt #2

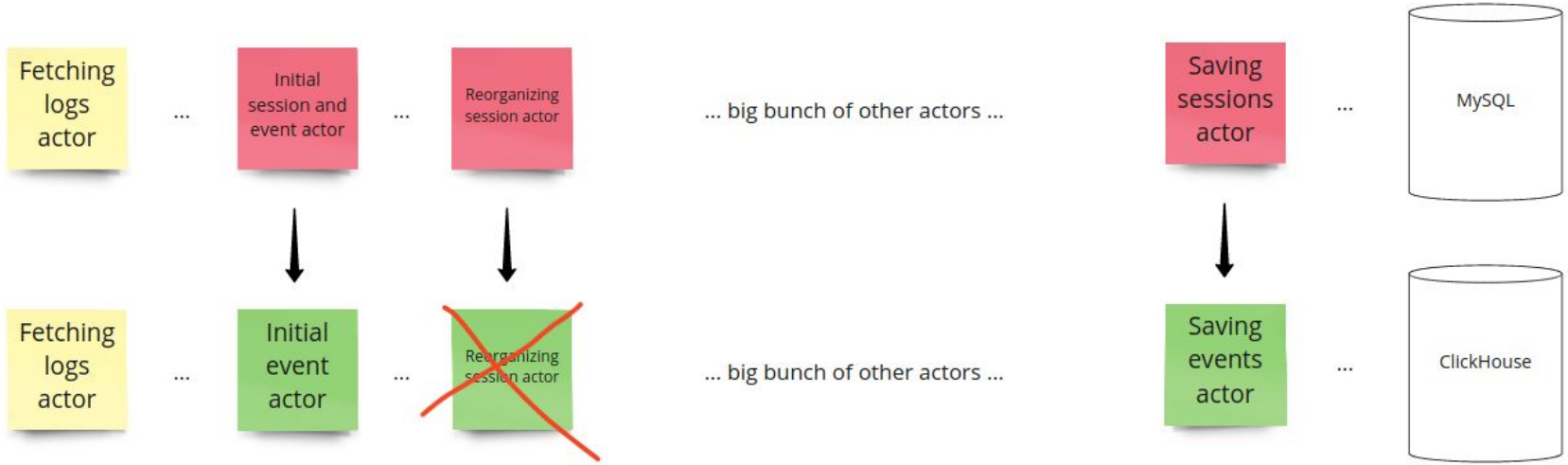

Processing events
Preprocessing sessions
ClickHouse
Processor
events table
Visitor 1:
Event1
Visitor 2:
Event1
Visitor 1:
Event2
Visitor 2:
Event2
Materialized view
visitors table
Visitor 1:
Event1, Event2
Visitor 2:
Event1, Event 2

Attempt #2


Detecting sessions
ClickHouse
visitors table
Visitor 23:
Event1, Event2
Visitor 33:
Event1, Event2, Event3
Session detection query
Session 1:
Event1, Event2
Session 1:
Event1, Event2
Session 2:
Event3

Why only a fraction of code was rewritten?

Actor model



Why only a fraction of code was rewritten?

Rust

- Type system
- Designed for managing big codebases
- Expressive

Part II

Session detection hardware requirements

- Modest RAM usage
- Moderately CPU-intensive
- Bottlenecks:
  - Storage
  - Network

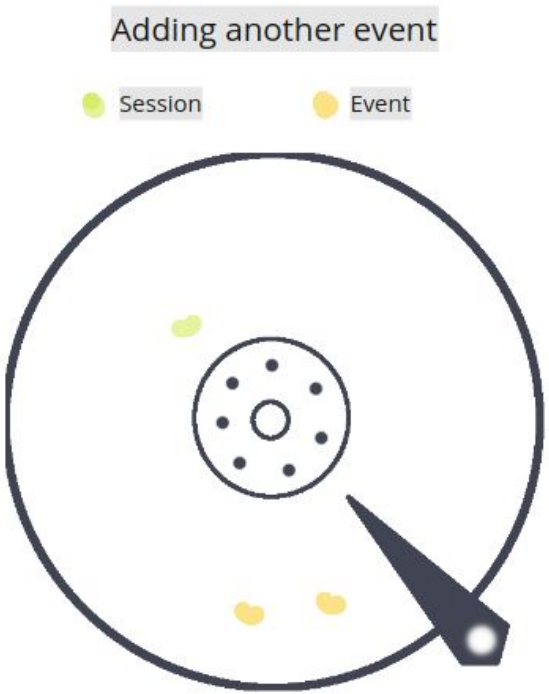
Why the attempt #1 was so slow?

- Detecting sessions when it is not needed
- No compression - more bytes to write and read
- Random reads and writes instead of sequential - delays caused by HDD head

How attempt #2 addresses attempt’s #1 pain points

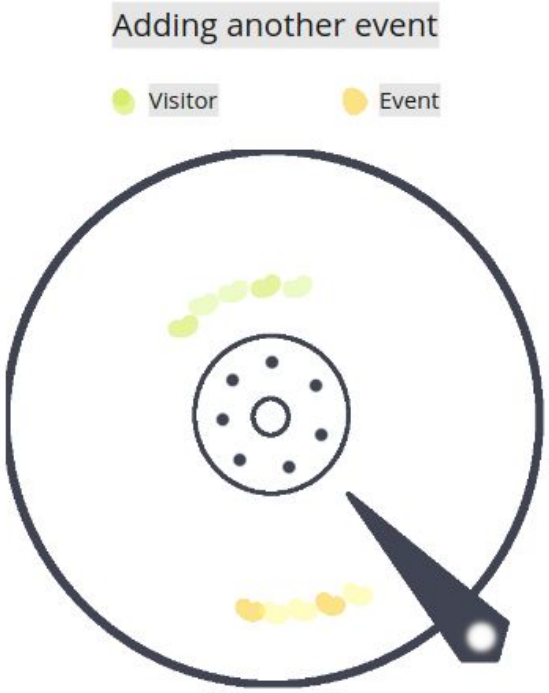
Attempt #1	Attempt #2
Detecting sessions when it is not needed	Detect only when asked for, but be prepared
No compression	Compress as much as possible
Random reads and writes	Only sequential reads and writes

Attempt #1 vs Attempt #2 on HDD



1 select + 1 insert

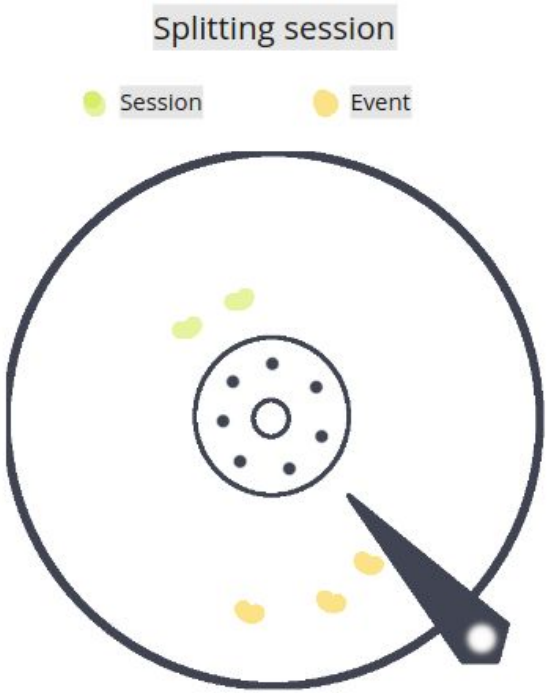
random random



2 inserts

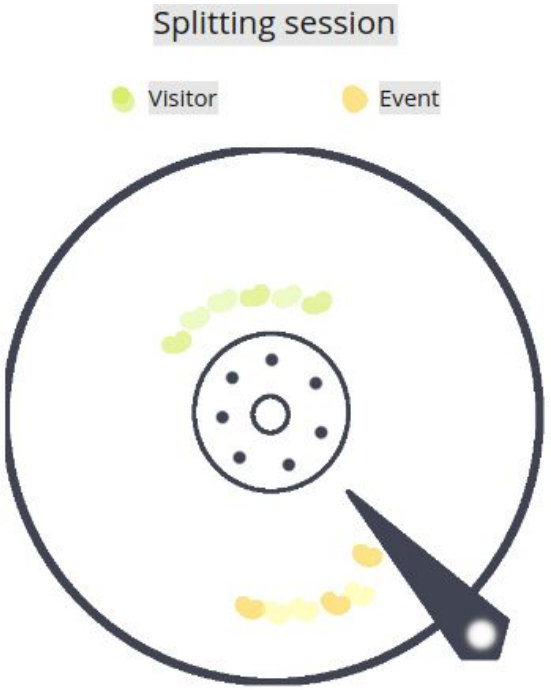
sequential

Attempt #1 vs Attempt #2 on HDD



1 select + 2 inserts (+ 1 update)

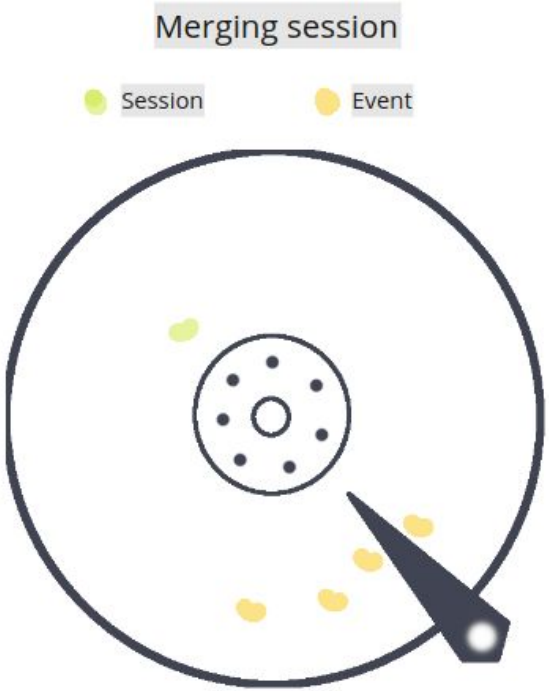
random sequential random



2 inserts

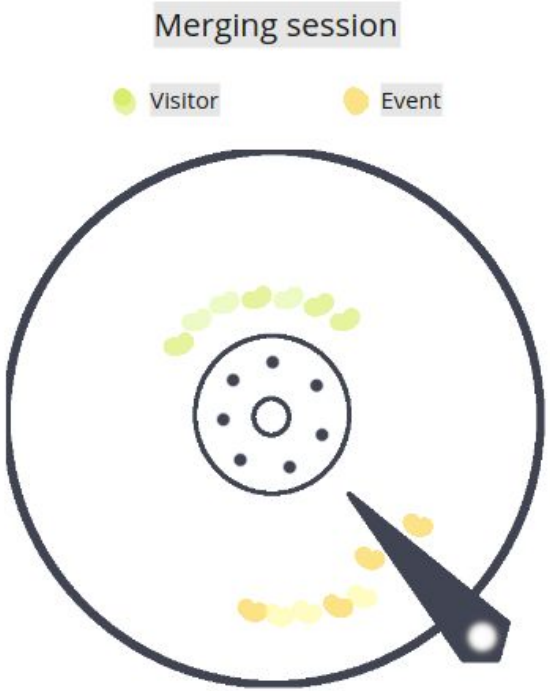
sequential

Attempt #1 vs Attempt #2 on HDD



1 select + 1 insert + 1 delete + 3 updates

random random random random



2 inserts

sequential

Attempt #1 vs Attempt #2 on...

...SSD?

Attempt #2 - what’s next?

- Still potential to optimize
- New promising features in ClickHouse: projections

Thank you!