

# ClickHouseのアーキテクチャと活用事例

2025

||||· ClickHouse

# アジェンダ

**01**

ClickHouseとは

**02**

アーキテクチャ

**03**

活用事例



# 01. ClickHouse とは

# ClickHouse とは？

## オープンソース

2009年から開発開始  
2016年にオープンソース化  
36,000+ GitHubスター  
1,300+ コントリビューター  
500+ リリース

## 列指向

集計に最適  
カラムごとのファイル管理  
ソートとインデックス  
バックグラウンドマージ

## 分散

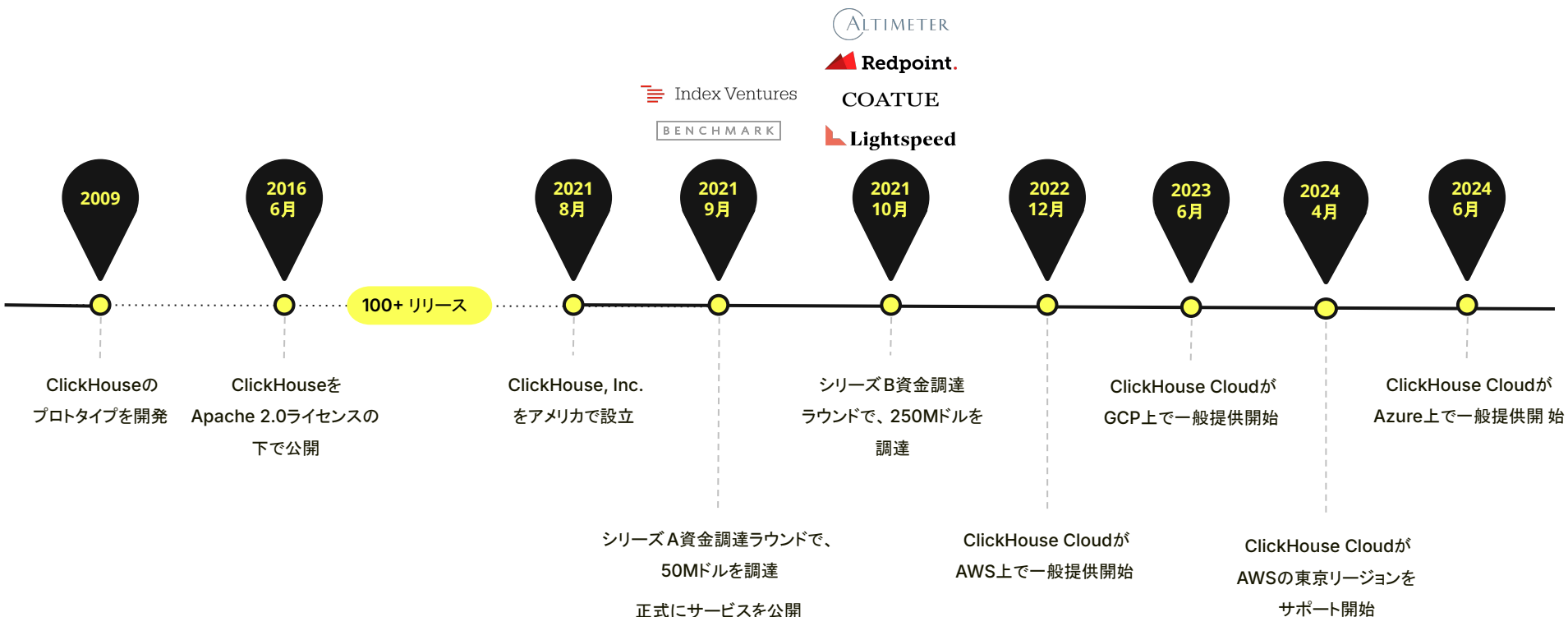
レプリケーション  
シャーディング  
マルチマスター  
クロスリージョン

## OLAPデータベース

分析ユースケース  
集計処理  
データの可視化  
ほぼイミュータブルなデータ

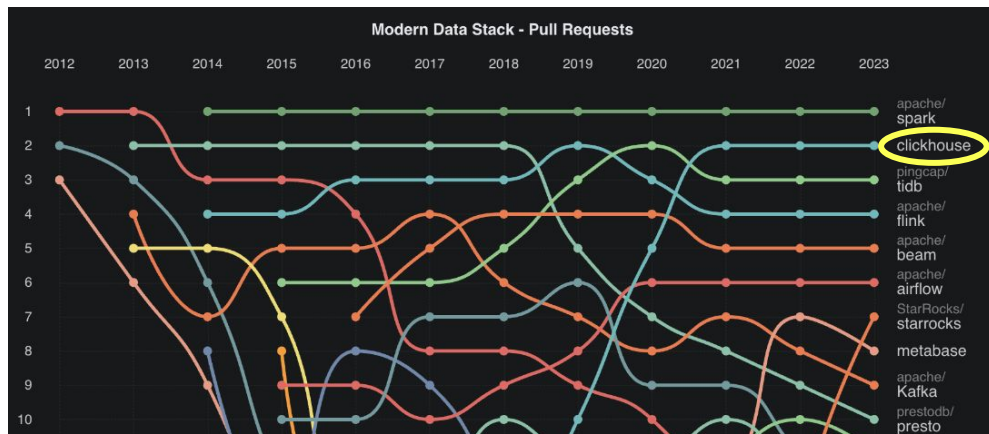


# ClickHouse の歴史



# ClickHouse オープンソース

- ✓ 36k以上のGitHubスター
- ✓ 6.4k以上のフォーク
- ✓ 1.3k以上のコントリビュータ
- ✓ 100k以上のコミット
- ✓ 114k のアクティブなコミュニティメンバー



## ClickHouse Cloud



- ✓ 高速、スケーラブル、そして信頼性が高い
- ✓ 柔軟で機能が豊富、かつ使いやすい
- ✓ 毎日数十億のクエリを処理

|||||

NETFLIX

vimeo

Spotify

Microsoft

Deutsche Bank

NGINX

CLOUDFLARE

CISCO

COMCAST

Walmart

ROKT

GitLab

ebay

IBM

Contentsquare

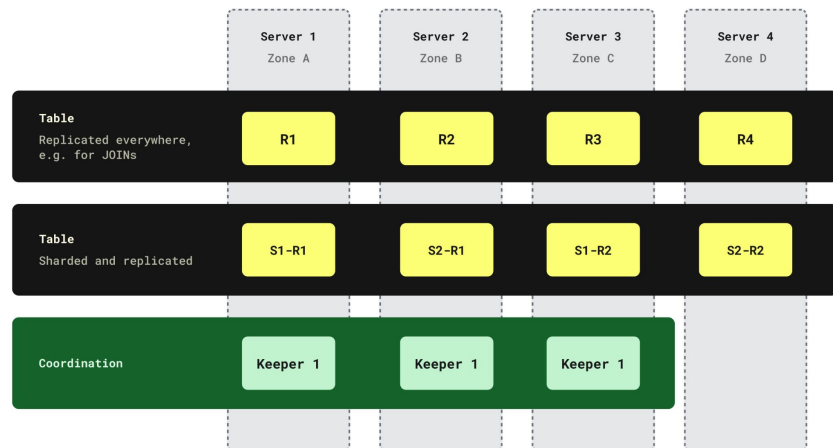


# ClickHouse

## セルフマネージド

- ✓ オープンソース
- ✓ 柔軟なアーキテクチャ
- ✓ 効率的で堅牢
- ✓ サポート契約が利用可能

セルフマネージドにおけるアーキテクチャのサンプル

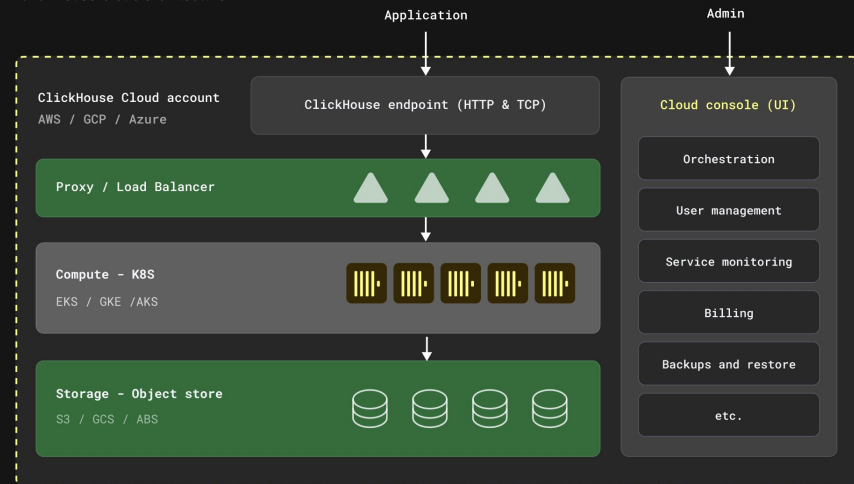


# ClickHouse

## Cloud

- ✓ 使いやすい
  - ✓ 機能が豊富
  - ✓ 高速
  - ✓ スケーラブル
  - ✓ 信頼性が高い
  - ✓ PAYG
- マネージド型サービス  
クラウドファースト機能とツールを提供  
自動的にパフォーマンスと効率を最適化  
シームレスなスケーリング  
高い信頼性を保証  
利用量と容量に応じた料金設定

ClickHouse Cloud architecture



## 02. アーキテクチャ



**ANSI-SQL の DDL やクエリは、  
ClickHouse で動作しますか？**



**ANSI-SQL の DDL やクエリは、  
ClickHouse で動作しますか？**

**はい！ ただし...**



# より高速な分析を実現するためのポイント

- ほぼ ANSI SQL と同様の構文
  - SQLシンタックスはほぼ一緒
  - 追加でデータ型や特殊関数をサポート
- 列指向ストレージの活用
  - 必要なカラムだけを SELECT して I/O を最小化
  - 大量データを扱う分析クエリで真価を発揮
- 従来のRDBMS(OLTP) とは設計思想が異なる
  - 既存のクエリにおいても、列指向の恩恵で分析クエリは高速
  - さらに高速化するにはテーブル設計やクエリの最適化が重要
  - トランザクションについては、部分的にサポート(完全にサポートしていない)



# 行指向と列指向の違い

# 行指向と列指向の理解

一般的な  
行指向のテーブル

id	date	user	product	price
1	2025-01-01	やまだ	リンゴ	200
2	2025-01-01	さとう	バナナ	80
3	2025-01-02	すずき	オレンジ	120
4	2025-01-02	たかはし	メロン	450

# 行指向と列指向の理解

一般的な  
行指向のテーブル

id	date	user	product	price
1	2025-01-01	やまだ	リンゴ	200
2	2025-01-01	さとう	バナナ	80
3	2025-01-02	すずき	オレンジ	120
4	2025-01-02	たかはし	メロン	450

一般的な  
RDBMS クエリ

```
SELECT price FROM sales WHERE user=すずき
```



# 行指向と列指向の理解

列指向の  
テーブルの場合

id	date	user	product	price
1	2025-01-01	やまだ	リンゴ	200
2	2025-01-01	さとう	バナナ	80
3	2025-01-02	すずき	オレンジ	120
4	2025-01-02	たかはし	メロン	450

# 行指向と列指向の理解

列指向の  
テーブルの場合

id	date	user	product	price
1	2025-01-01	やまだ	リンゴ	200
2	2025-01-01	さとう	バナナ	80
3	2025-01-02	すずき	オレンジ	120
4	2025-01-02	たかはし	メロン	450

一般的な  
分析クエリ

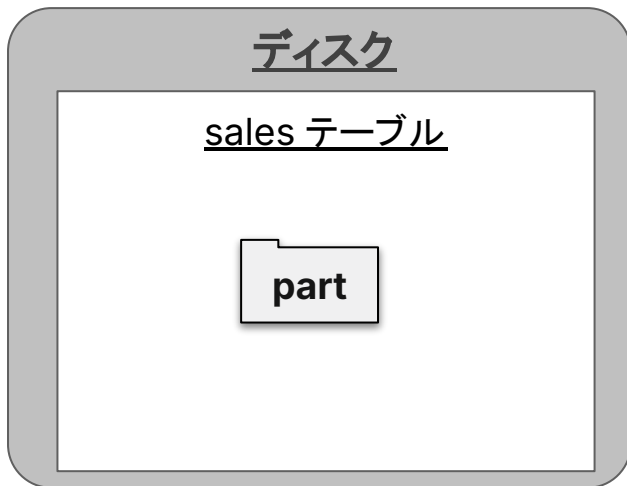
```
SELECT avg(price) FROM sale
```



# テーブルへの書き込みフロー

ID	date	user	product	price
1	2025-01-01	やまだ	リンゴ	200
2	2025-01-01	さとう	バナナ	80
3	2025-01-02	すずき	オレンジ	120
4	2025-01-02	たかはし	メロン	450

インサート



書き込み



## ① Sort Keyで並べ替える (date, user)

ID	date	user	product	price
2	2025-01-01	さとう	バナナ	80
1	2025-01-01	やまだ	リンゴ	200
3	2025-01-02	すずき	オレンジ	120
4	2025-01-02	たかはし	メロン	450



## ② 列単位に分割する

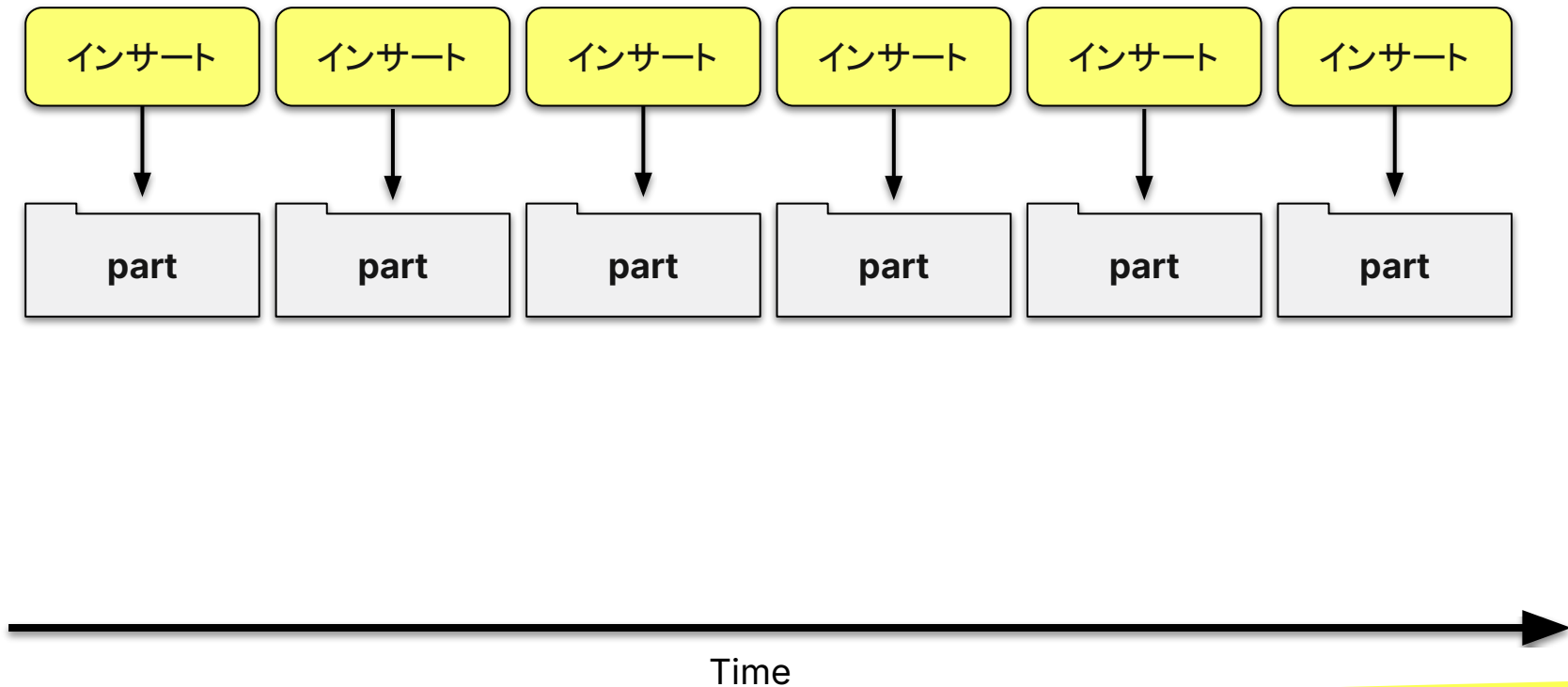
ID	date	user	product	price
2	2025-01-01	さとう	バナナ	80
1	2025-01-01	やまだ	リンゴ	200
3	2025-01-02	すずき	オレンジ	120
4	2025-01-02	たかはし	メロン	450



## ③ 列ごとに圧縮する

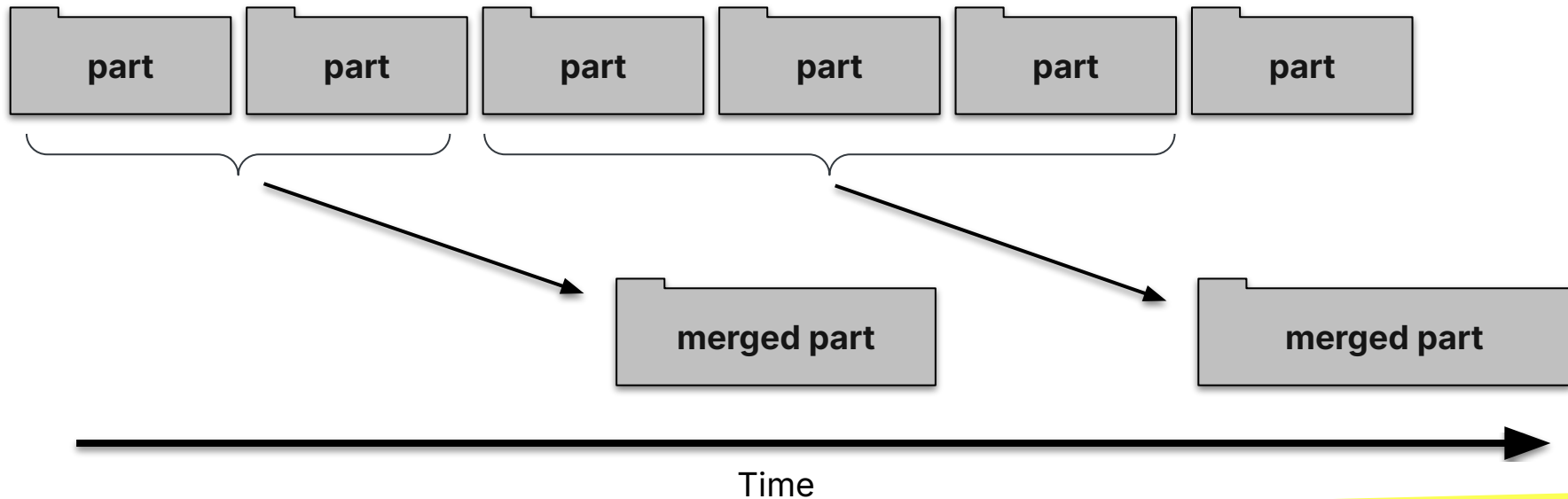
ID	date	user	product	price
----	------	------	---------	-------

# INSERTごとにPARTが作成される

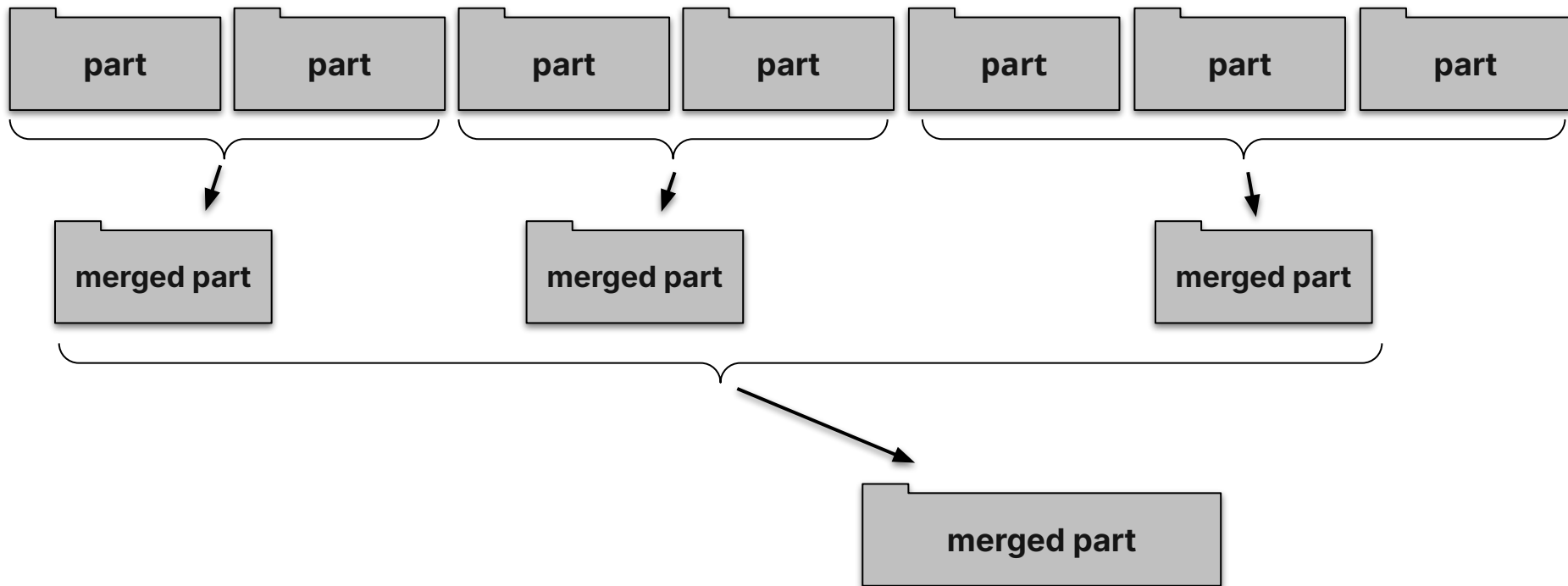


# 時間の経過とともに、パーツがバックグラウンドでマージ

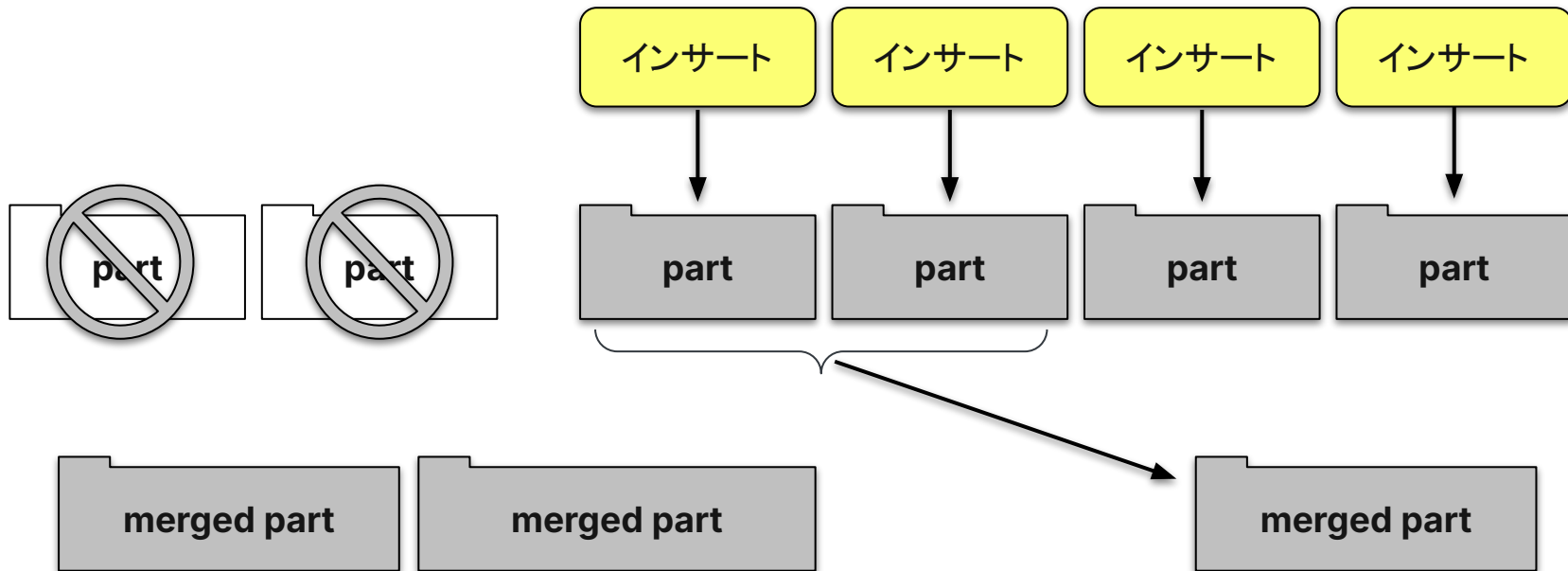
- これが *MergeTree* と呼ばれる理由です



マージされた **part** は引き続きマージされます



マージ後、未使用の部分は最終的に削除される



# PRIMARY KEY による 高速なデータ READ

# PRIMARY KEY のアーキテクチャ

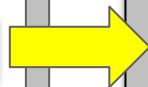
## ディスク

### sales テーブル

Part Part Part

⋮

Part Part Part



date	user	product	price
2025-01-01	あべ	リンゴ	100
2025-01-01	あべ	バナナ	80
2025-01-01	さとう	オレンジ	120
⋮	⋮	⋮	⋮
2025-01-02	すずき	オレンジ	120
2025-01-02	すずき	メロン	450
⋮	⋮	⋮	⋮
2025-01-04	たぐち	モモ	250
⋮	⋮	⋮	⋮
2025-01-09	いしい	ブドウ	1500
2025-01-10	まつもと	スイカ	2000



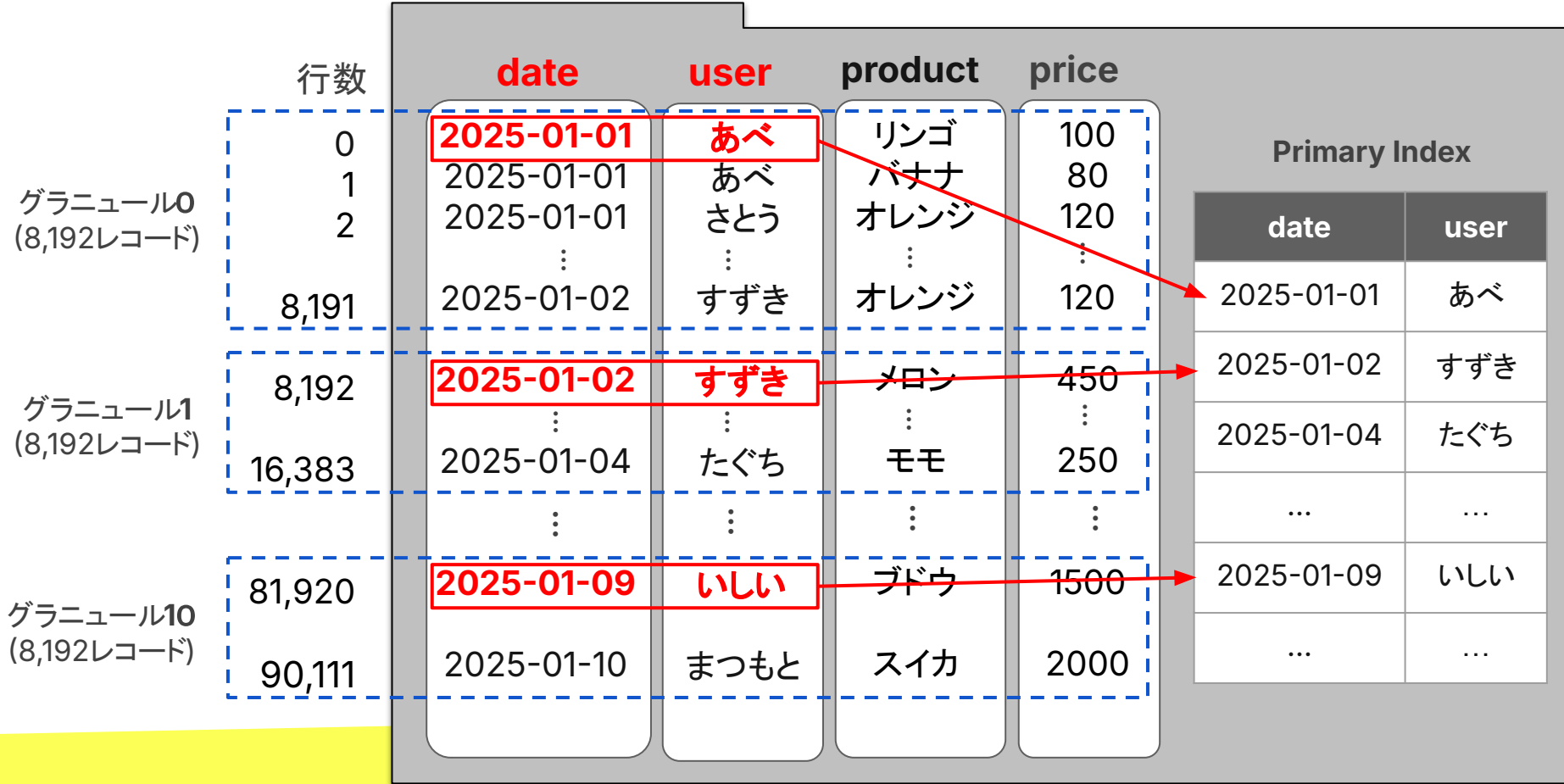
# PRIMARY KEY のアーキテクチャ

	行数	date	user	product	price
グラニューロ0 (8,192レコード)	0	2025-01-01	あべ	リンゴ	100
	1	2025-01-01	あべ	バナナ	80
	2	2025-01-01	さとう	オレンジ	120
		⋮	⋮	⋮	⋮
	8,191	2025-01-02	すずき	オレンジ	120
グラニューロ1 (8,192レコード)	8,192	2025-01-02	すずき	メロン	450
		⋮	⋮	⋮	⋮
	16,383	2025-01-04	たぐち	モモ	250
		⋮	⋮	⋮	⋮
グラニューロ10 (8,192レコード)	81,920	2025-01-09	いしい	ブドウ	1500
	90,111	2025-01-10	まつもと	スイカ	2000

# PRIMARY KEY のアーキテクチャ

	行数	date	user	product	price
グラニューロ0 (8,192レコード)	0	2025-01-01	あべ	リンゴ	100
	1	2025-01-01	あべ	バナナ	80
	2	2025-01-01	さとう	オレンジ	120
		⋮	⋮	⋮	⋮
	8,191	2025-01-02	すずき	オレンジ	120
グラニューロ1 (8,192レコード)	8,192	2025-01-02	すずき	メロン	450
		⋮	⋮	⋮	⋮
	16,383	2025-01-04	たぐち	モモ	250
		⋮	⋮	⋮	⋮
グラニューロ10 (8,192レコード)	81,920	2025-01-09	いしい	ブドウ	1500
	90,111	2025-01-10	まつもと	スイカ	2000

# PRIMARY KEY のアーキテクチャ



# 各グラニューールはスレッドによって処理

- Primry Keyを利用して、処理対象となるグラニューールを見つけ出し、各スレッドで処理
  - 各グラニューールは並行処理されます
  - 処理が速いスレッドが、遅れているスレッドのタスクを引き受けて処理する仕組みがあり、その結果、全体の負荷を均等にして処理を高速化される
- 並行処理が ClickHouse の高速性を支える理由の1つとなっています



<https://www.youtube.com/watch?v=7QXKBKDOKJE>




## 03. 活用事例

# ユースケース




## ログ、イベント、トレース

ログ、イベント、トレースの確実な監視を実現。異常検知や不正検知、ネットワーク・インフラの問題など、様々な課題を検出可能。

 Darwinium



 BENOCS

 resmo

 zomato

 runreveal



## リアルタイム分析

大規模データのリアルタイム分析・集計が可能なインタラクティブなアプリケーションとダッシュボードを実現。社内の複雑な分析処理も、分や時間単位ではなく、ミリ秒単位での実行を実現。


 vimeo

 CLOUDFLARE

 Microsoft



 Contentsquare


 highlight.io




## ビジネスインテリジェンス

データを自在に分析し、分析レポートや社内アプリケーションの構築に活用。ユーザー行動分析、広告・メディア効果測定、市場動向分析など、幅広い用途に対応。

 ROKT

Deutsche Bank 

 QuickCheck

 NANO CORP.

 TrillaBit

 HiFi




## 機械学習と生成 AI

高速かつ効率的なベクトル検索を実現。様々なプロバイダーの生成 AI モデルをすぐに利用可能。ペタバイト規模のモデルトレーニングも、超高速な集計処理で実現。

 denic

 DeepL

 ADMIXER

 ensemble





Lyftでは、**毎日**で数千万行のデータをClickHouseに取り込み、数百万回のクエリを実行しています。その処理量は増加傾向にあります。

月単位では、25TB以上のデータ読み書きを実現しています。

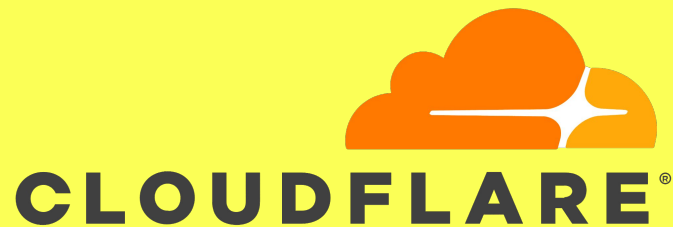
リアルタイム分析用データストア





ClickHouseを活用することで、**何兆件規模のインターネットリクエストログ** を効率的かつ確実に分析し、悪意のあるトラフィックの検出と、顧客への詳細な分析データの提供を実現しています。

オブザーバビリティプラットフォームの  
バックエンド





ありがとう ございます  
**Thank you**

Keep in touch!



[clickhouse.com/slack](https://clickhouse.com/slack)



[#clickhouseDB](#) [@clickhouseinc](#)



[clickhouse](#)