



浙江财经大学  
Zhejiang University of Finance & Economics

数据科学学院  
SCHOOL OF DATA SCIENCES

# 内存不足情况下前沿数据分析 框架性能比较

## ——以Arrow、DuckDB和Polars为例

黄天元

浙江财经大学数据科学学院  
数据科学与大数据技术系

第18届中国R会议（广州） 2025.7



中国R会议  
The China-R Conference



# 目录

- 内存不足的实践背景
- 前沿大数据分析框架
- R语言中的相关工具
- 三种框架的性能比较
- 总结与展望





# 目录

- 内存不足的实践背景
- 前沿大数据分析框架
- R语言中的相关工具
- 三种框架的性能比较
- 总结与展望





# 与数据操作的故事：硕士-博士-博后

2013



2017



复旦大学生命科学学院

School of Life Sciences Fudan University



Scopus®



2021

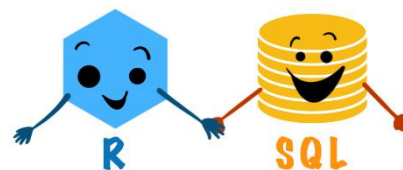


中国科学院文献情报中心

NATIONAL SCIENCE LIBRARY, CHINESE ACADEMY OF SCIENCES



Clarivate  
Web of Science™



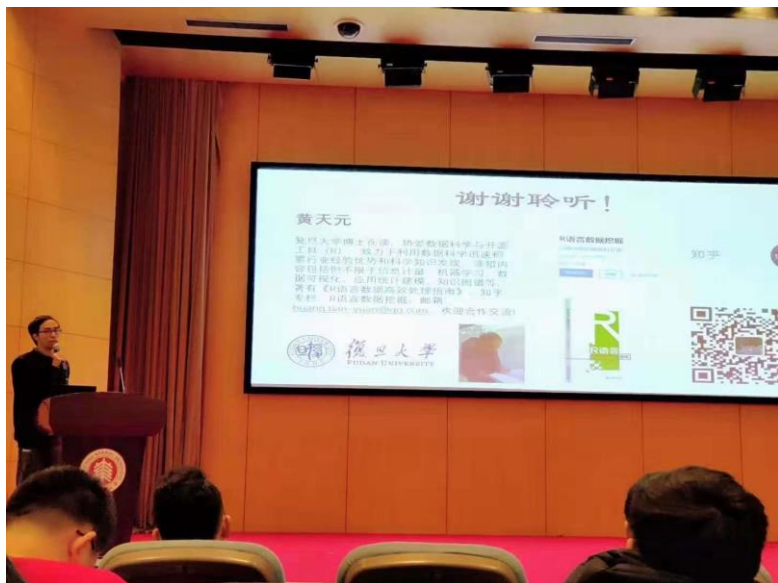
浙江财经大学  
Zhejiang University of Finance & Economic



# 与数据操作的故事

第12届中国R会议（上海）  
华东师范大学  
2019.12  
R语言数据操纵之美

第15届中国R会议（北京）  
中国人民大学  
2022.11  
R语言高效数据操作工具：tidyfst



R语言高效数据操作工具：tidyfst



黄天元  
2022年11月

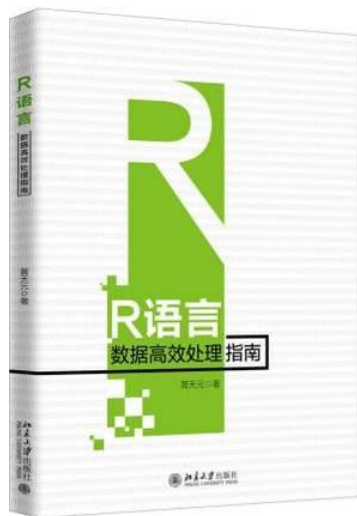
第15届中国R会议（北京）



浙江财经大学  
Zhejiang University of Finance & Economic



# 与数据操作的故事



## tidyfst: Tidy Verbs for Fast Data Manipulation

CRAN 1.8.2 devel version 2.0.0 lifecycle stable downloads 74K  
downloads 2105/month downloads 1173/week downloads 188/day  
DOI 10.5281/zenodo.7264887 JOSS 10.21105/joss.02388



## Using qs

R-CMD-check passing CRAN 0.27.3 downloads 13K/month downloads 656K

Quick serialization of R objects



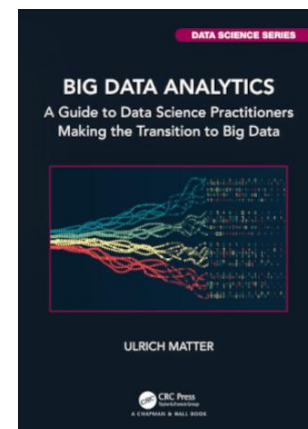
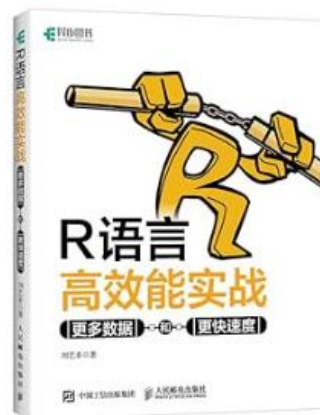
## disk.frame

FOR >> RAM DATA



## r-dbi/DBI

A database interface (DBI) definition for communication between R and RDBMSs







# 与数据操作的故事

实战大数据：基于R语言

前言

- 1 大数据基本概念
  - 2 R语言编程入门
  - 3 数据处理效能的衡量
  - 4 快速读写：大数据的导入与导出
  - 5 快速整理：基于data.table的数据处理工具
  - 6 快速绘图：大数据可视化工具
  - 7 快速建模：高性能机器学习工具
  - 8 化整为零：对文件进行批处理
  - 9 跨语言召唤术：在R中调用其他编程工具
  - 10 时间换空间：大数据流式计算
  - 11 空间换时间：大数据并行计算
  - 12 从内存到外存：用数据库管理数据
  - 13 从本地到集群：大数据分布式计算
- 参考资料

实战大数据：基于R语言

AUTHOR  
黄天元

PUBLISHED  
October 23, 2024

## 前言

本教程主要介绍如何使用R语言来进行高性能计算，从而应对大数据时代给我们带来的各种挑战。这本书面向的是已经具有一定R语言基础的读者，在面对海量观测构成的数据集时，如何从容地像往常一样对数据进行丰富的分析与建模。为了对R基础较为薄弱的读者也友好，本书不会使用过分深刻晦涩的材料，力求深入浅出。书会结合当前最先进的R语言工具包（包括但不限于**data.table**、**duckdb**、**arrow**、**sparklyr**、**Rcpp**、**future**），系统地介绍以下几个部分的内容：

- 大数据基本概念
- R语言编程入门
- 数据处理效能的衡量
- 快速读写：大数据的导入与导出
- 快速整理：基于data.table的数据处理工具
- 快速绘图：大数据可视化工具
- 快速建模：高性能机器学习工具
- 化整为零：对文件进行批处理
- 跨语言召唤术：在R中调用其他编程工具
- 时间换空间：大数据流式计算
- 空间换时间：大数据并行计算
- 从内存到外存：用数据库管理数据
- 从本地到集群：大数据分布式计算



<https://hope-data-science.github.io/R4BD/>



浙江财经大学  
Zhejiang University of Finance & Economics



# 内存不足的实践背景

在大数据处理和分析的过程中，随着**数据规模的不**  
**断增长**，尤其是当数据量达到数十亿甚至数百亿行时，  
**传统的内存计算方式面临着显著的挑战**。单纯依赖内  
存存储和处理数据已变得不再现实，**内存的限制往往**  
**成为性能瓶颈**，导致内存不足、计算效率低下等问题。  
因此，如何在**内存受限的环境下高效处理大规模数据**，  
已成为当前数据计算领域的重要课题。







# 内存不足的实践背景

satijalab/seurat

#2997 **Error: cannot  
allocate vector of size  
790.8 Gb**



 5 comments



**mamtagiri** opened on May 12, 2020





## 内存不足的实践背景

为了解决这一问题，越来越多的现代数据处理框架应运而生，这些框架通过**优化内存使用**、**支持磁盘存储计算**等方式，提高了数据处理的灵活性和效率。**Arrow**、**DuckDB**和**Polars**等框架是当前最受关注的解决方案，它们通过**列式存储**、**智能查询优化**和**内存外计算**等技术，能够有效降低内存压力，实现高效的数据处理。



DuckDB





# 目录

- 内存不足的实践背景
- 前沿大数据分析框架
- R语言中的相关工具
- 特定场景的性能比较
- 总结与展望





# Arrow简介

**Apache Arrow**是一个跨平台的开源库，旨在提供高速的列式存储和内存共享格式，特别适合大规模数据处理。允许不同的计算框架（如**Pandas**、**R**、**Spark**、**Dask**等）共享内存中的数据，而不需要序列化/反序列化。

特点：

- **列式**内存格式：Arrow使用列式数据格式，适合高速的数据分析和查询。
- **跨语言**支持：支持多种编程语言，包括**Python**、**R**、**Java**、**C++**等，提供高效的数据交换机制。
- 共享内存格式：Arrow可以让不同的计算框架共享内存中的数据，而无需额外的内存拷贝或序列化，对于**并行计算和分布式**计算系统非常有利。
- 高效数据**读**取和**写**入：有效减少I/O成本。





# Arrow简介

APACHE  
ARROW



The universal columnar format and multi-language toolbox for fast data interchange and in-memory analytics

## What is Arrow?

### Format

Apache Arrow defines a language-independent columnar memory format for flat and nested data, organized for efficient analytic operations on modern hardware like CPUs and GPUs. The Arrow memory format also supports zero-copy reads for lightning-fast data access without serialization overhead.

Learn more about the design or read the specification.

### Libraries

Arrow's libraries implement the format and provide building blocks for a range of use cases, including high performance analytics. Many popular projects use Arrow to ship columnar data efficiently or as the basis for analytic engines.

Libraries are available for C, C++, C#, Go, Java, JavaScript, Julia, MATLAB, Python, R, Ruby, Rust, and Swift. See how to install and get started.

### Ecosystem

Apache Arrow is software created by and for the developer community. We are dedicated to open, kind communication and consensus decisionmaking. Our committers come from a range of organizations and backgrounds, and we welcome all to participate with us.

Learn more about how you can ask questions and get involved in the Arrow project.





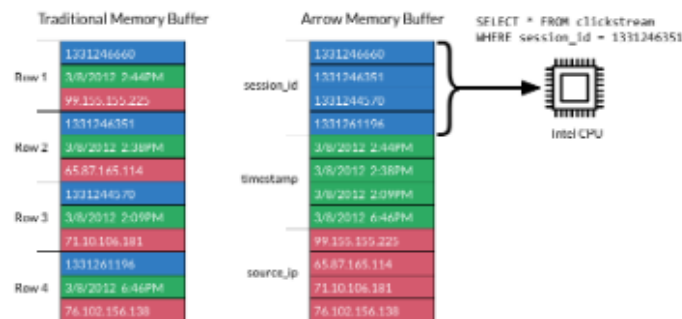


# Arrow简介：列处理

## Columnar is Fast

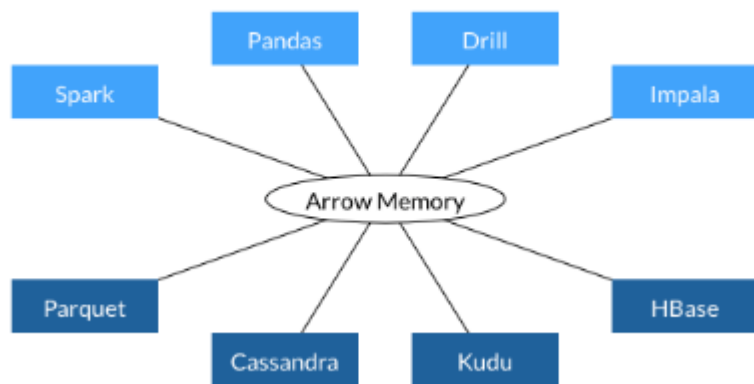
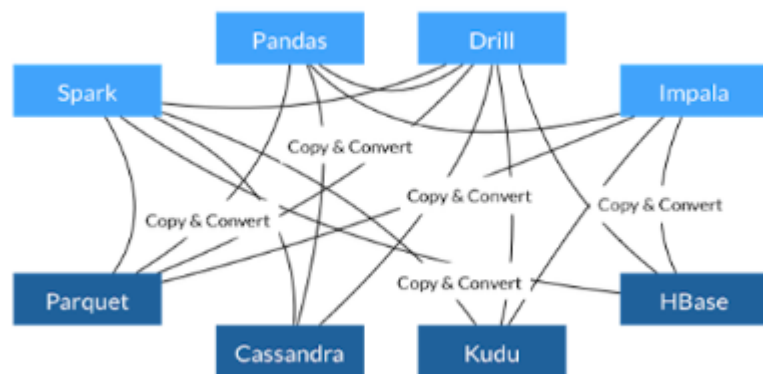
The Apache Arrow format allows computational routines and execution engines to maximize their efficiency when scanning and iterating large chunks of data. In particular, the contiguous columnar layout enables vectorization using the latest SIMD (Single Instruction, Multiple Data) operations included in modern processors.

	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138





# Arrow简介：标准化存储



## Standardization Saves

Without a standard columnar data format, every database and language has to implement its own internal data format. This generates a lot of waste. Moving data from one system to another involves costly serialization and deserialization. In addition, common algorithms must often be rewritten for each data format.

Arrow's in-memory columnar data format is an out-of-the-box solution to these problems. Systems that use or support Arrow can transfer data between them at little-to-no cost. Moreover, they don't need to implement custom connectors for every other system. On top of these savings, a standardized memory format facilitates reuse of libraries of algorithms, even across languages.





# Arrow简介：多语言支持

## Arrow Libraries

The Arrow project contains libraries that enable you to work with data in the Arrow columnar format in many languages. The C++, C#, Go, Java, JavaScript, Julia, Rust, and Swift libraries contain distinct implementations of the Arrow format. These libraries are **integration-tested** against each other to ensure their fidelity to the format. In addition, Arrow libraries for C (GLib), MATLAB, Python, R, and Ruby are built on top of the C++ library.

These official libraries enable third-party projects to work with Arrow data without having to implement the Arrow columnar format themselves. They also contain many software components that assist with systems problems related to getting data in and out of remote storage systems and moving Arrow-formatted data over network interfaces, among other **use cases**.





# Polars简介

**Polars**是一个快速且高效的**DataFrame**库。核心设计基于**Rust**编程语言，在多核处理器上具备优秀的性能表现。提供与**Pandas**类似的接口，但性能通常优于**Pandas**，特别是在大规模数据集上。

特点：

- 多线程**并行**处理：Polars对多核并行进行优化，能够显著提高数据处理速度。
- **惰性评估**（Lazy Evaluation）：支持惰性计算，可以构建一系列的计算操作，只有在实际需要计算结果时才执行，从而减少不必要的计算。
- **列式存储**：类似于Arrow，Polars采用列式存储格式，在内存外操作时能够提高性能。
- 简洁的**API**：Polars提供了类似Pandas的API，可快速上手。





## Polars简介

# DataFrames for the new era

175M+

Downloads to date

33k+

Github stars

Polars is an open-source library for data manipulation, known for being one of the fastest data processing solutions on a single machine. It features a well-structured, typed API that is both expressive and easy to use.







# Polars简介：核心优势



01

## Fast

Polars is written from the ground up with performance in mind. Its multi-threaded query engine is written in Rust and designed for effective parallelism. Its vectorized and columnar processing enables cache-coherent algorithms and high performance on modern processors.

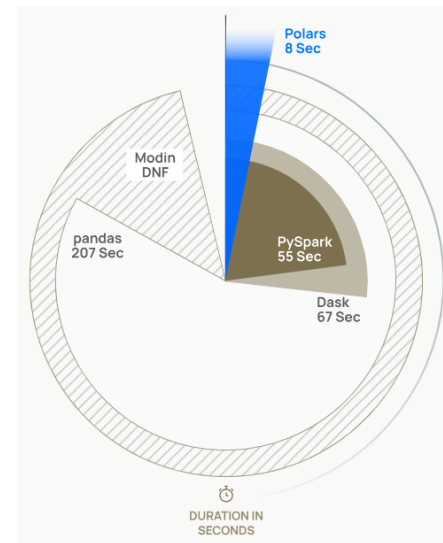


02

## Easy to use

You will feel right at home with Polars if you are familiar with data wrangling. Its expressions are intuitive and empower you to write code which is readable and performant at the same time.

Built by developers for developers to achieve up to **50x performance**



03

















## Open source

Polars is and always will be open source. Driven by an active community of developers, everyone is encouraged to add new features and contribute. Polars is free to use under the MIT license.





# Polars简介：文件支持广泛

				<p>SUPPORT</p> <h2>Support for all common data formats</h2> <p>Polars supports reading and writing to all common data formats. This allows you to easily integrate Polars with your existing data stack.</p> <ul style="list-style-type: none"><li>✓ Text: CSV &amp; JSON</li><li>✓ Binary: Parquet, Delta Lake, AVRO &amp; Excel</li><li>✓ IPC: Feather, Arrow</li><li>✓ Databases: MySQL, Postgres, SQL Server, Sqlite, Redshift &amp; Oracle</li><li>✓ Cloud storage: S3, Azure Blob &amp; Azure File</li></ul>
				
				
				





# Polars简介：其他特性

## How Polars will make your life easier

### 01 Easy to use

Write your queries the way they were intended. Polars will determine the most efficient way to execute them using its query optimizer.

### 02 Embarrassingly parallel

Complete your queries faster! Polars fully utilizes the power of your machine by dividing the workload among the available CPU cores without any additional configuration or serialization overhead.

### 03 Apache Arrow

Polars utilizes the Apache Arrow memory model allowing you to easily integrate with existing tools in the data landscape. It supports zero-copy data sharing for efficient collaboration.

### 04 Close to the metal

Polars is written from the ground up, designed close to the machine and without external dependencies. This allows for full control of the ecosystem (API, memory & execution).

### 05 Written in Rust

The core of Polars is written in Rust, one of the fastest growing programming languages in the world. Rust allows for high performance with fine-grained control over memory.

### 06 Out of core

Want to process large data sets that are bigger than your memory? Our streaming API allows you to process your results efficiently, eliminating the need to keep all data in memory.





# DuckDB简介

**DuckDB**是一个轻量级的、嵌入式的列式数据库，设计目标是高效地执行SQL查询。它被设计为支持高速的OLAP（在线分析处理）查询，可以直接操作本地文件系统中的数据，而不需要将数据完全加载到内存中。

特点：

- **嵌入式**数据库：与传统数据库系统不同，**DuckDB**是一个嵌入式数据库，可以轻松集成到应用程序中，不需要复杂的数据库管理系统。
- **列式**存储：数据以列而非行的形式存储，适合执行大规模数据分析操作。
- 高效的**磁盘操作**：即使不将数据加载到内存，**DuckDB**仍能高效地进行查询，适合处理超过内存大小的大型数据集。
- **SQL**支持：可以使用标准的SQL语法来进行数据查询和操作，支持查询、聚合、连接等。





# DuckDB简介



# DuckDB

🔄 Main passing chat 642 online 📦 release v1.3.1

## DuckDB

DuckDB is a high-performance analytical database system. It is designed to be fast, reliable, portable, and easy to use. DuckDB provides a rich SQL dialect, with support far beyond basic SQL. DuckDB supports arbitrary and nested correlated subqueries, window functions, collations, complex types (arrays, structs, maps), and [several extensions designed to make SQL easier to use](#).

DuckDB is available as a [standalone CLI application](#) and has clients for [Python](#), [R](#), [Java](#), [Wasm](#), etc., with deep integrations with packages such as [pandas](#) and [dplyr](#).







# DuckDB简介：特点



## Simple

DuckDB is easy to [install](#) and deploy. It has zero external dependencies and runs in-process in its host application or as a single binary.

[Read more →](#)



## Portable

DuckDB runs on Linux, macOS, Windows, Android, iOS and all popular hardware architectures. It has idiomatic [client APIs](#) for major programming languages.

[Read more →](#)



## Feature-rich

DuckDB offers a [rich SQL dialect](#). It can read and write file formats such as CSV, Parquet, and JSON, to and from the local file system and remote endpoints such as S3 buckets.

[Read more →](#)



## Fast

DuckDB runs analytical queries at blazing speed thanks to its columnar engine, which supports parallel execution and can process larger-than-memory workloads.

[Read more →](#)



## Extensible

DuckDB is extensible by third-party features such as new data types, functions, file formats and new SQL syntax. User contributions are available as community extensions.

[Read more →](#)



## Free

DuckDB and its core extensions are open-source under the permissive MIT License. The intellectual property of the project is held by the [DuckDB Foundation](#).

[Read more →](#)





# 三种分析框架对比

特性	DuckDB	Arrow	Polars
架构类型	嵌入式数据库	数据交换格式、内存共享	高性能数据框架
存储格式	列式存储	列式存储 (Apache Arrow格式)	列式存储
内存外操作	支持 磁盘上进行操作	支持 Parquet等格式数据存储	支持大数据集和 内存外计算
并行计算	不强制支持	不直接支持并行计算	强力支持多核并行计算
SQL支持	完全支持SQL查询	无SQL支持	无SQL支持
适用场景	数据库操作， 复杂的查询和分析	数据交换， 跨平台高效数据共享	大规模数据分析 支持内存外处理
性能	非常高效， 适合复杂查询和磁盘操作	高效的数据交换 和内存共享，适合跨语言	高效，特别适合 大规模数据集和并行计算





# 目录

- 内存不足的实践背景
- 前沿大数据分析框架
- **R语言中的相关工具**
- 三种框架的性能比较
- 总结与展望





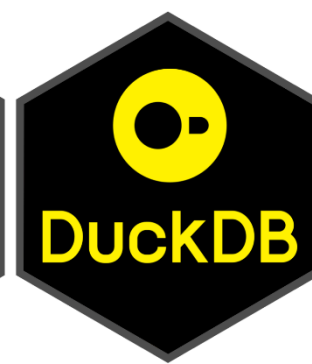
# 相关R语言包

## 核心工具包:

- **Arrow: arrow**
- **DuckDB: duckdb**
- **Polars: polars; tidypolars**

## 辅助工具包:

- **数据操作: dplyr**
- **SQL转译: dbplyr**
- **数据库连接: DBI**





# 文件存储格式建议

在使用 **DuckDB**、**Arrow** 和 **Polars** 等框架时，若追求高效的列式读写和压缩性能，应优先选用 **Parquet**；若需要频繁执行复杂 **SQL** 查询、表间联结或保留索引与统计信息，则可将数据持久存为 **DuckDB** 单文件数据库；若侧重跨语言或内存管道高速交换，适合在运行时使用 **Arrow** 格式；仅在小规模数据需手工查看或兼容性最重要时，才考虑使用 **CSV**。

场景	推荐格式
多次查询、复杂 SQL、join-heavy 数据分析	DuckDB 数据库 (.duckdb)
数据湖 / 分布式查询 / OLAP 扫描	Parquet
多语言管道交互 (Python $\leftrightarrow$ Rust 等)	Arrow buffer (.arrow)
小数据量、手工查看或导出 Excel	CSV

在内存不足场景下，尽管以上提到格式都可以使用，但是主要推荐使用**Parquet**和**DuckDB数据库**文件存储。其中，**Parquet**通用性最强。

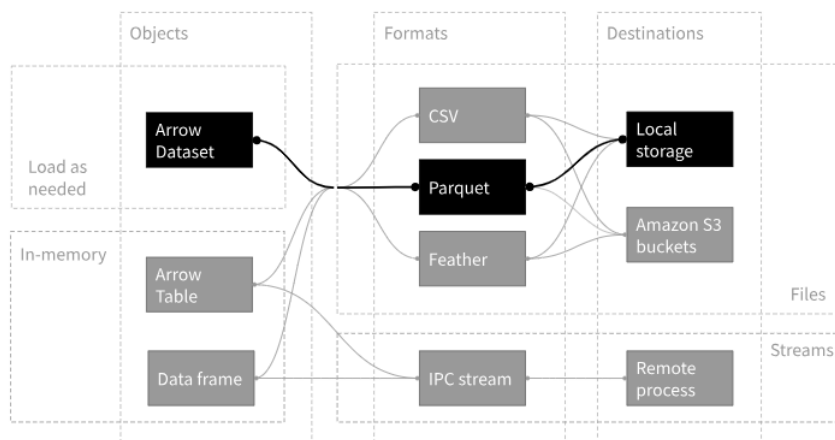






# 内存不足场景下的数据I/O：Arrow

Apache Arrow 通过**惰性加载**（计算时只加载必要的**数据**）与**分批次处理**（**试验阶段**）机制，仅将当前计算所需的数据块载入内存，而非全量数据，从而实现高效处理远超单机内存限制的大数据集。



```
nyc_taxi <- open_dataset("~/Datasets/nyc-taxi/")  
nyc_taxi
```

```
FileSystemDataset with 158 Parquet files  
vendor_name: string  
pickup_datetime: timestamp[ms]  
dropoff_datetime: timestamp[ms]  
passenger_count: int64  
...
```

Arrow 的 `open_dataset` 并非“完全不需内存”，而是通过分块惰性加载，将内存占用限制在可控范围内，从而高效处理大规模数据。用户仍需注意单批次数据的大小，但无需手动实现分块逻辑。





# 内存不足场景下的数据I/O: Polars

**Polars** 利用懒查询计划在**流式**（**streaming**）模式下将数据拆分为可控的**小批次处理**，对每批数据顺序读取、计算并输出，从而无需将整个数据集一次性加载进内存，实现真正的内存外（**out-of-core**）操作。

## Import data from Parquet file(s)

Source: [R/read\\_scan.R](#)

`read_parquet_polars()` imports the data as a Polars DataFrame.

`scan_parquet_polars()` imports the data as a Polars **LazyFrame**.

## Stream output to a parquet file

Source: [R/sink.R](#)

This function allows to stream a LazyFrame that is larger than RAM directly to a `.parquet` file **without collecting it in the R session**, thus preventing crashes because of too small memory.

## Usage

`sink_parquet(`





# 内存不足场景下的数据I/O: Polars

Polars 的 Lazy API 本身只是构建了一个查询计划，真正的计算分两种模式：

## ➤ 默认 `.collect()`

不传 `streaming` 参数 或 `streaming=False` 时，Polars 会先把所有满足条件的数据一次性读到内存，然后在内存里完成所有算子（过滤、聚合、连接等）再返回一个完整的 `DataFrame`。这是**纯内存中计算**。

## ➤ 流式 `.collect(streaming=True)`

启用 `streaming` 模式后，Polars 会把数据分成一个个小batch（`RecordBatch`）从磁盘读取进来，依次对每个batch应用算子，然后直接把结果输出或写出，不会把全量数据一次性载入内存。每个 batch 的中间结果放在内存里，但不会累积整个表，所以整体上是内存外（`out-of-core`）计算，只要单个batch能装进内存就能处理任意大的数据集。





# 内存不足场景下的数据I/O: DuckDB

**DuckDB** 在运行时监测内存使用，当中间算子（如聚合或排序）生成的数据**超出内存限制时**，就会将这些溢出分区或临时中间结果**分批写入磁盘临时文件（spilling）**，随后**在磁盘上执行外部聚合和外部排序**，再将结果合并，从而实现超内存情况下的平滑执行。

## No Memory? No Problem. External Aggregation in DuckDB



Laurens Kuiper

2024-03-29 · 17 min

TL;DR: Since the 0.9.0 release, DuckDB's fully parallel aggregate hash table can efficiently aggregate over many more groups than fit in memory.

<https://duckdb.org/2024/03/29/external-aggregation.html>



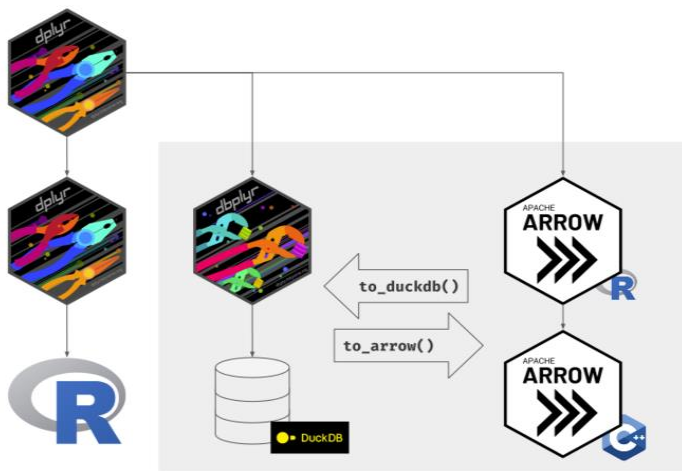


# 数据操作层

支持众多基本函数，包括**dplyr**、**tidyr**等。

同时，由于共享列式存储的特性，三种框架的数据工作流可以无缝衔接（比如，**parquet**文件格式除了能够被**Arrow**和**Polars**读取外，也能够被**DuckDB**直接处理）。

## DuckDB 数据库



```
library(polars)
library(tidypolars)
library(dplyr, warn.conflicts = FALSE)
library(tidyr, warn.conflicts = FALSE)
```

```
who_df <- tidyr::who
who_pl <- as_polars_df(tidyr::who)
```

```
who_df |>
  filter(year > 1990) |>
  drop_na(newrel_f3544) |>
  select(iso3, year, matches("^newrel(.*)_f")) |>
  arrange(iso3, year) |>
  rename_with(.fn = toupper) |>
  head()
```

已经掌握**tidyverse**的用户可以轻易地把知识经验进行移植！





# 目录

- 内存不足的实践背景
- 前沿大数据分析框架
- R语言中的相关工具
- 三种框架的性能比较
- 总结与展望





# 线上官方比较：16核心+32G

Small (16 cores / 32GB memory)

X-Large (128 cores / 256GB memory)

Task

groupby join

0.5 GB 5 GB 50 GB

basic questions

Input table: 100000000 rows x 9 columns ( 5 GB )

DuckDB	1.3.0	2025-06-13	5s
Datafusion	47.0.0	2025-06-02	6s
Polars	1.30.0	2025-06-02	7s
ClickHouse	25.5.2.47	2025-06-16	7s
data.table	1.16.99	2025-01-22	10s
R-arrow	20.0.0.2	2025-06-02	11s
collapse	2.1.2	2025-06-02	11s
dplyr	1.1.4	2025-01-22	92s

Small (16 cores / 32GB memory)

X-Large (128 cores / 256GB memory)

Task

groupby join

0.5 GB 5 GB 50 GB

basic questions

Input table: 100000000 rows x 7 columns ( 5 GB )

Polars	1.30.0	2025-06-02	10s
DuckDB	1.3.0	2025-06-13	38s
data.table	1.16.99	2025-01-22	61s
collapse	2.1.2	2025-06-02	70s
ClickHouse	25.5.2.47	2025-06-16	122s
spark	4.0.0	2025-06-02 undefined exception	
R-arrow	20.0.0.2	2025-06-02 out of memory	
Datafusion	43.1.0	2025-01-23 undefined exception	
Modin		see README	pending

Small (16 cores / 32GB memory)

X-Large (128 cores / 256GB memory)

Task

groupby join

0.5 GB 5 GB 50 GB

basic questions

Input table: 1000000000 rows x 9 columns ( 50 GB )

ClickHouse	25.5.2.47	2025-06-16	84s
DuckDB*	1.3.0	2025-06-15	326s
spark	3.5.4	2025-01-23	Not Tested
Polars	1.20.0	2025-01-23	Not Tested
Datafusion	43.1.0	2025-01-23	Not Tested
collapse	2.0.19	2025-01-23	Not Tested
Modin		see README	pending

Small (16 cores / 32GB memory)

X-Large (128 cores / 256GB memory)

Task

groupby join

0.5 GB 5 GB 50 GB

basic questions

Input table: 1000000000 rows x 7 columns ( 55 GB )

DuckDB*	1.3.0	2025-06-16	1485s
spark	3.5.4	2025-01-23	Not tested
Polars	1.20.0	2025-01-23	out of memory
Datafusion	43.1.0	2025-01-23	Not tested
ClickHouse	25.5.2.47	2025-06-16	Out of Memory
collapse	2.0.19	2025-01-23	Not tested
Modin		see README	pending

<https://duckdblabs.github.io/db-benchmark/>



浙江财经大学  
Zhejiang University of Finance & Economic





# 线上官方比较：128核+256G

Small (16 cores / 32GB memory)

X-Large (128 cores / 256GB memory)

Task

groupby join

0.5 GB 5 GB 50 GB

basic questions

Input table: 100000000 rows x 9 columns ( 5 GB )

DuckDB	1.3.0	2025-06-05	2s
Datafusion	47.0.0	2025-06-05	2s
ClickHouse	25.5.2.47	2025-06-06	2s
Polars	1.30.0	2025-06-05	4s
data.table	1.16.99	2025-02-10	9s
collapse	2.0.19	2025-02-10	11s
R-arrow	20.0.0.2	2025-06-05	26s
spark	4.0.0	2025-06-05	26s
pandas	2.2.3	2025-02-10	42s
(py)datatable	1.2.0a0	2024-09-13	111s

Small (16 cores / 32GB memory)

X-Large (128 cores / 256GB memory)

Task

groupby join

0.5 GB 5 GB 50 GB

basic questions

Input table: 1000000000 rows x 9 columns ( 50 GB )

Datafusion	47.0.0	2025-06-06	26s
ClickHouse	25.5.2.47	2025-06-06	29s
DuckDB	1.3.0	2025-06-05	31s
data.table	1.16.99	2025-02-10	84s
Polars	1.30.0	2025-06-05	175s
collapse	2.1.2	2025-06-18	214s
R-arrow	20.0.0.2	2025-05-29	391s
spark*	4.0.0	2025-06-05	520s
pandas	2.2.2	2024-09-13	822s
(py)datatable	1.2.0a0	2024-09-13	918s
dplyr	1.1.4	2024-09-13	1194s

Small (16 cores / 32GB memory)

X-Large (128 cores / 256GB memory)

Task

groupby join

0.5 GB 5 GB 50 GB

basic questions

Input table: 100000000 rows x 7 columns ( 5 GB )

Polars	1.30.0	2025-06-05	8s
DuckDB	1.3.0	2025-06-05	9s
Datafusion	47.0.0	2025-06-05	13s
ClickHouse	25.5.2.47	2025-06-06	13s
R-arrow	20.0.0.2	2025-06-05	30s
data.table	1.16.99	2025-02-10	58s
collapse	2.0.19	2025-02-10	68s
dask	2025.1.0	2025-02-10	159s
spark	4.0.0	2025-06-05	207s
pandas	2.2.3	2025-02-10	260s

Small (16 cores / 32GB memory)

X-Large (128 cores / 256GB memory)

Task

groupby join

0.5 GB 5 GB 50 GB

basic questions

Input table: 1000000000 rows x 7 columns ( 55 GB )

DuckDB*	1.3.0	2025-06-05	98s
ClickHouse	25.5.2.47	2025-06-06	521s
InMemData.jl	0.7.21	2024-09-30	CSV import Segfault
DataFrames.jl	1.6.1	2024-09-30	CSV import Segfault
data.table	1.16.99	2025-02-10	timeout
dplyr	1.1.4	2024-09-17	out of memory
pandas	2.2.2	2024-09-17	out of memory
(py)datatable	1.2.0a0	2024-09-17	out of memory
spark	3.5.2	2024-09-17	Not tested
dask	2024.9.0	2024-09-17	out of memory
Polars	1.30.0	2025-06-05	out of memory
R-arrow	20.0.0.2	2025-05-29	out of memory





# 线下个人试验：实验设计

## 2.2.2 实验变量

- 自变量：测试用例、内存限制量
- 因变量：运行时间、内存使用峰值

## 2.2.3 实验流程

整体的实验流程可以用如下伪代码表示：

---

### Algorithm 1 实验流程

---

```
1: 最大内存限制  $\leftarrow$  list
2: 被测试包  $\leftarrow$  list
3: 测试用例  $\leftarrow$  list
4: for  $(io, mem, pack, case) \in$  最大内存限制  $\times$  被测试包  $\times$  测试用例 do
5:     限制最大内存
6:     设置测试语
7:     运行测试用例
8:     统计并保存数据
9: end for
```

---





# 线下个人试验：实验环境

## System Details Report

Date generated: 2024-12-18 15:15:50

## Hardware Information

Hardware Model	ASUSTeK COMPUTER INC. ASUS TUF Gaming F15 FX507ZM_FX507ZM
Memory	40.0 GiB
Processor	12th Gen Intel® Core™ i7-12700H × 20
Graphics	Intel® Graphics (ADL GT2)
Graphics 1	NVIDIA GeForce RTX™ 3060 Laptop GPU
Disk Capacity	1.5 TB

## Software Information

Firmware Version	FX507ZM.316
OS Name	Ubuntu 24.10
OS Build	(null)
OS Type	64-bit
GNOME Version	47
Windowing System	Wayland
Kernel Version	Linux 6.11.0-13-generic





# 线下个人试验：测试例子

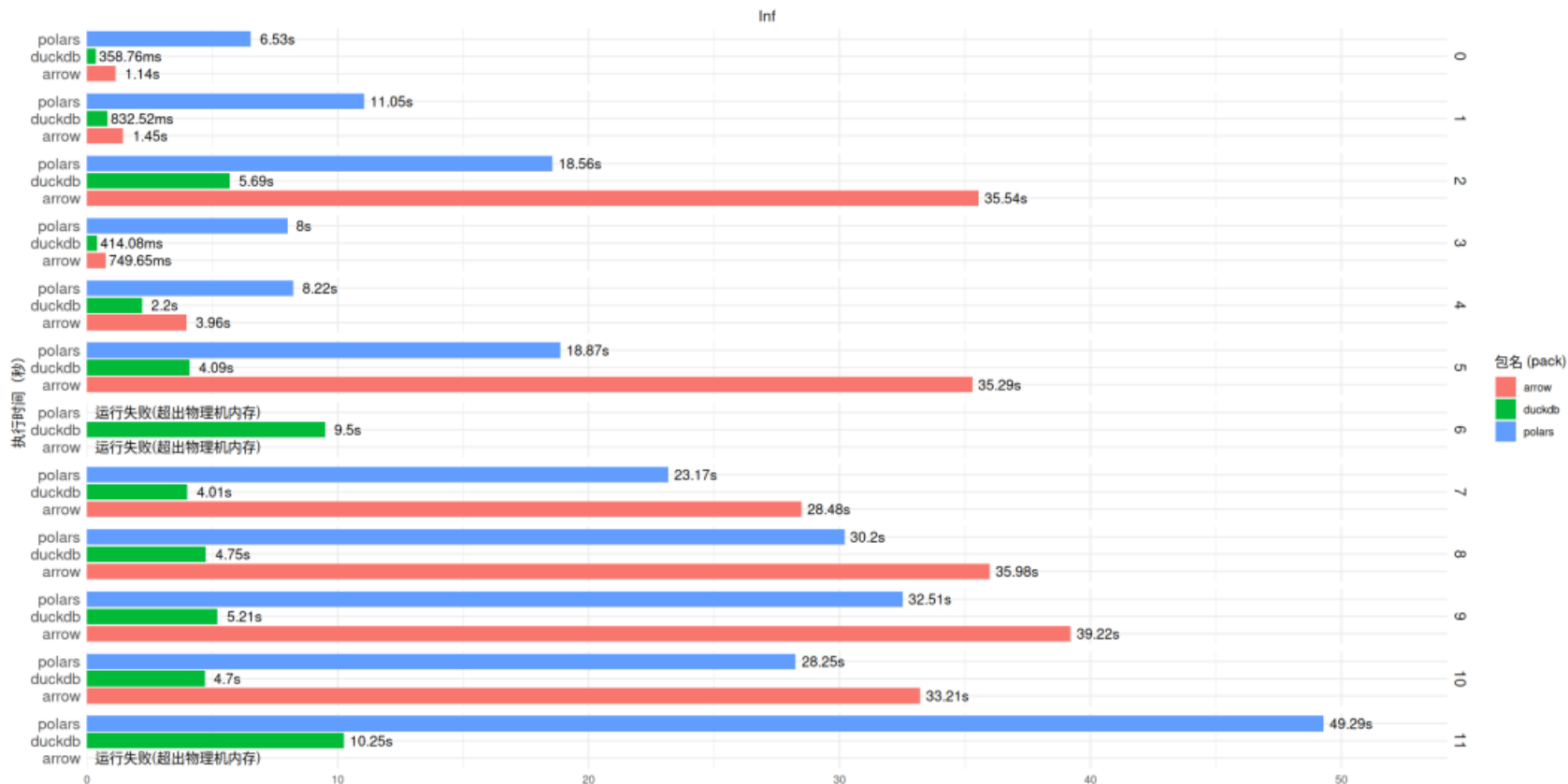
GROUP 情景测试		JOIN 情景测试	
测试 ID	测试情景	测试 ID	测试情景
0	sum v1 by id1	7	small inner on int
1	sum v1 by id1:id2	8	medium inner on int
2	sum v1 mean v3 by id3	9	medium outer on int
3	mean v1:v3 by id4	10	medium inner on factor
4	sum v1:v3 by id6	11	big inner on int
5	median v3 sd v3 by id4 id5		
6	max v1 - min v2 by id3		





# 线下个人试验：性能比较（不限内存）

各包在不限内存时所有测试用例下的执行时间





# 线下个人试验：性能比较（不限内存）

各包在不限内存时所有测试用例下的内存占用峰值

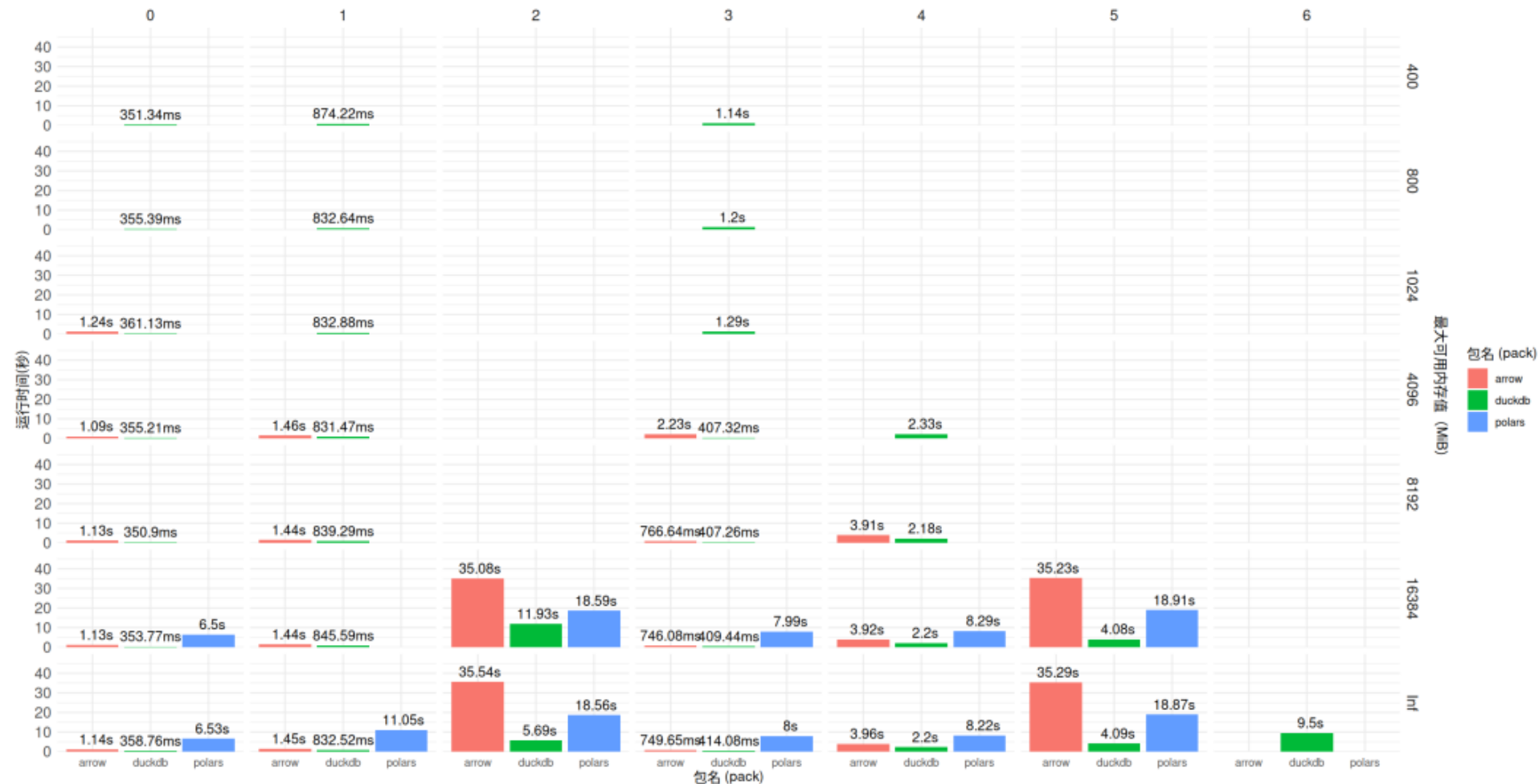
Inf





# 线下个人试验：性能比较（限制内存）

各包的GROUPBY测试用例在内存限制下的运行时间

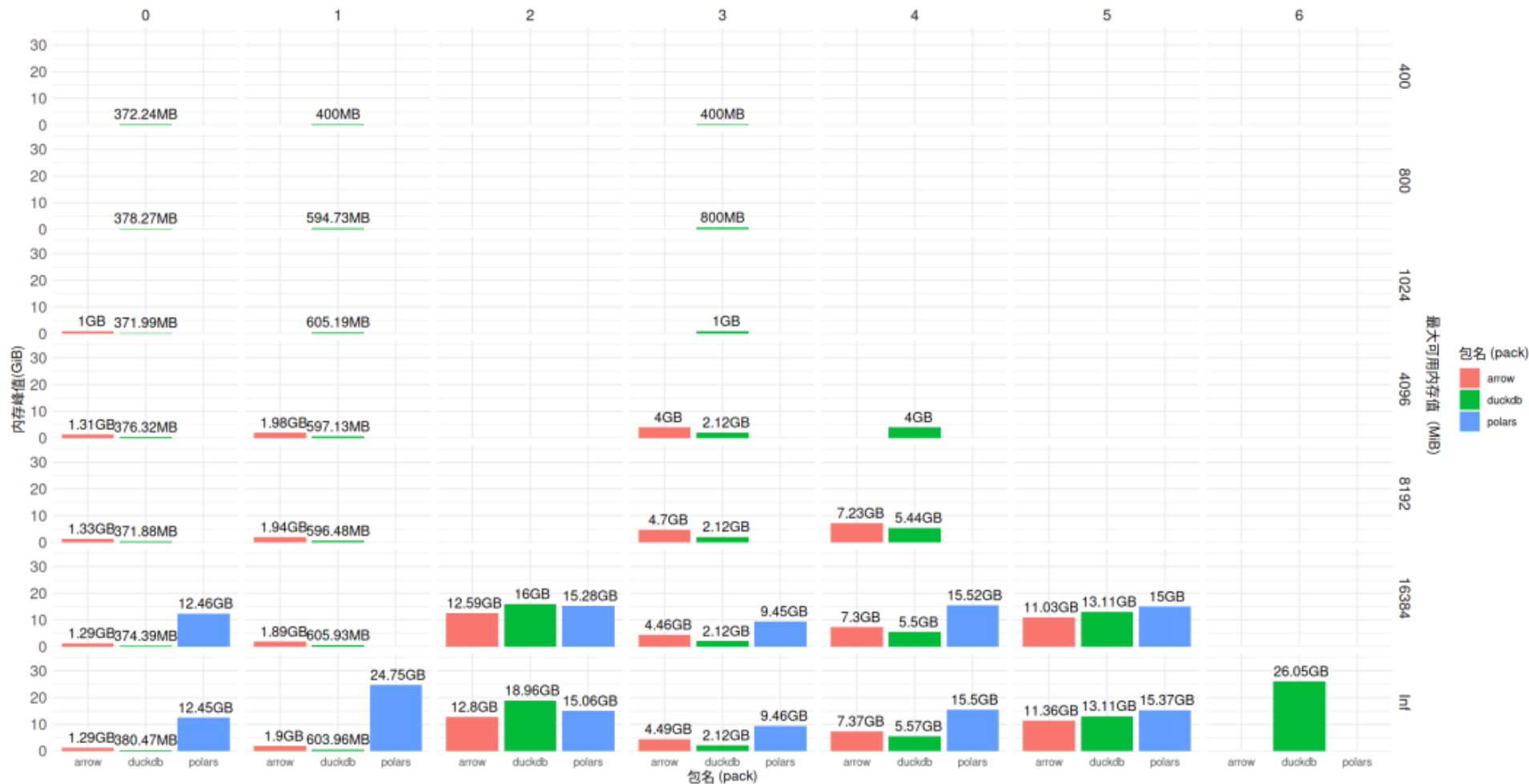






# 线下个人试验：性能比较（限制内存）

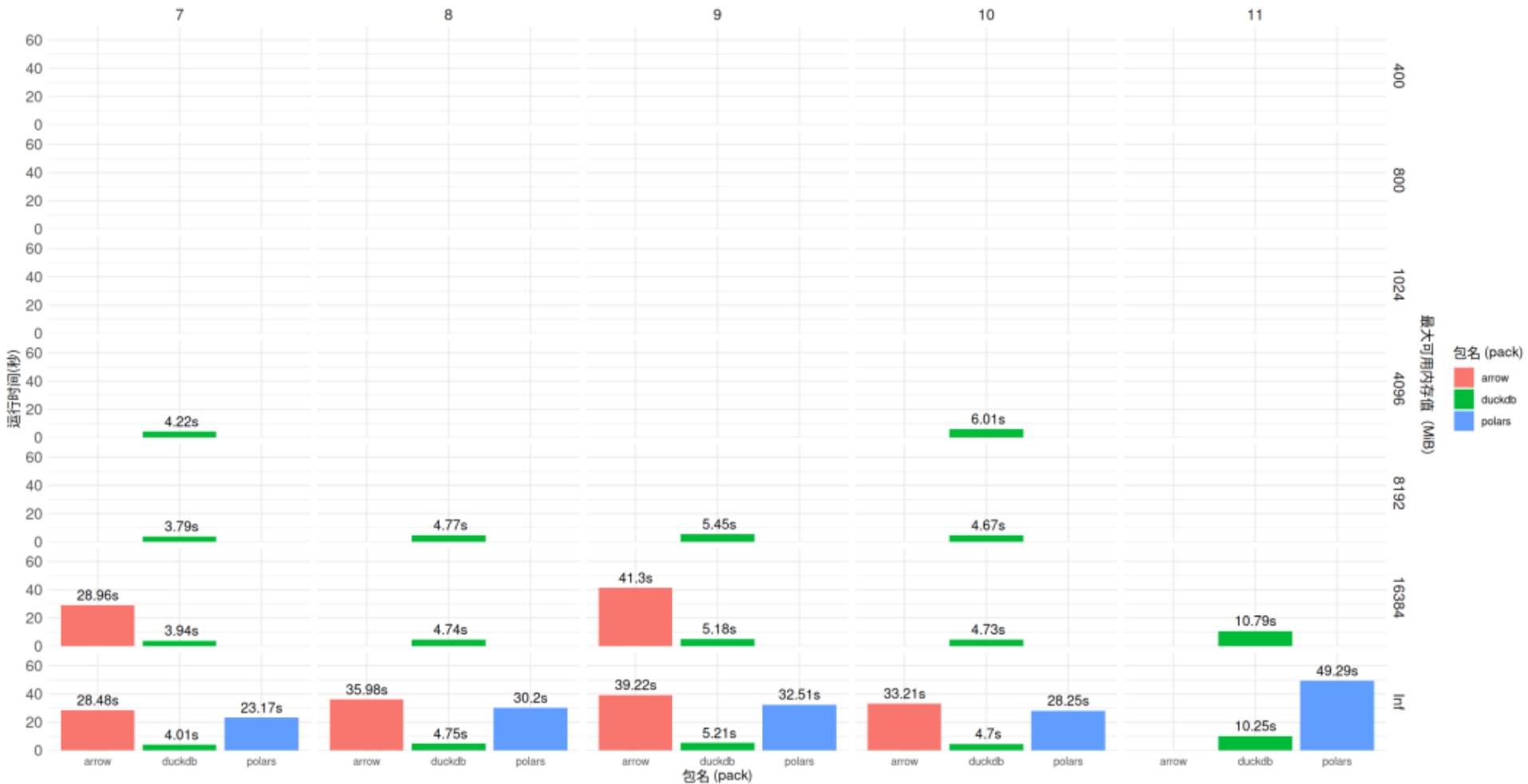
各包的GROUPBY测试用例在内存限制下的内存峰值





# 线下个人试验：性能比较（限制内存）

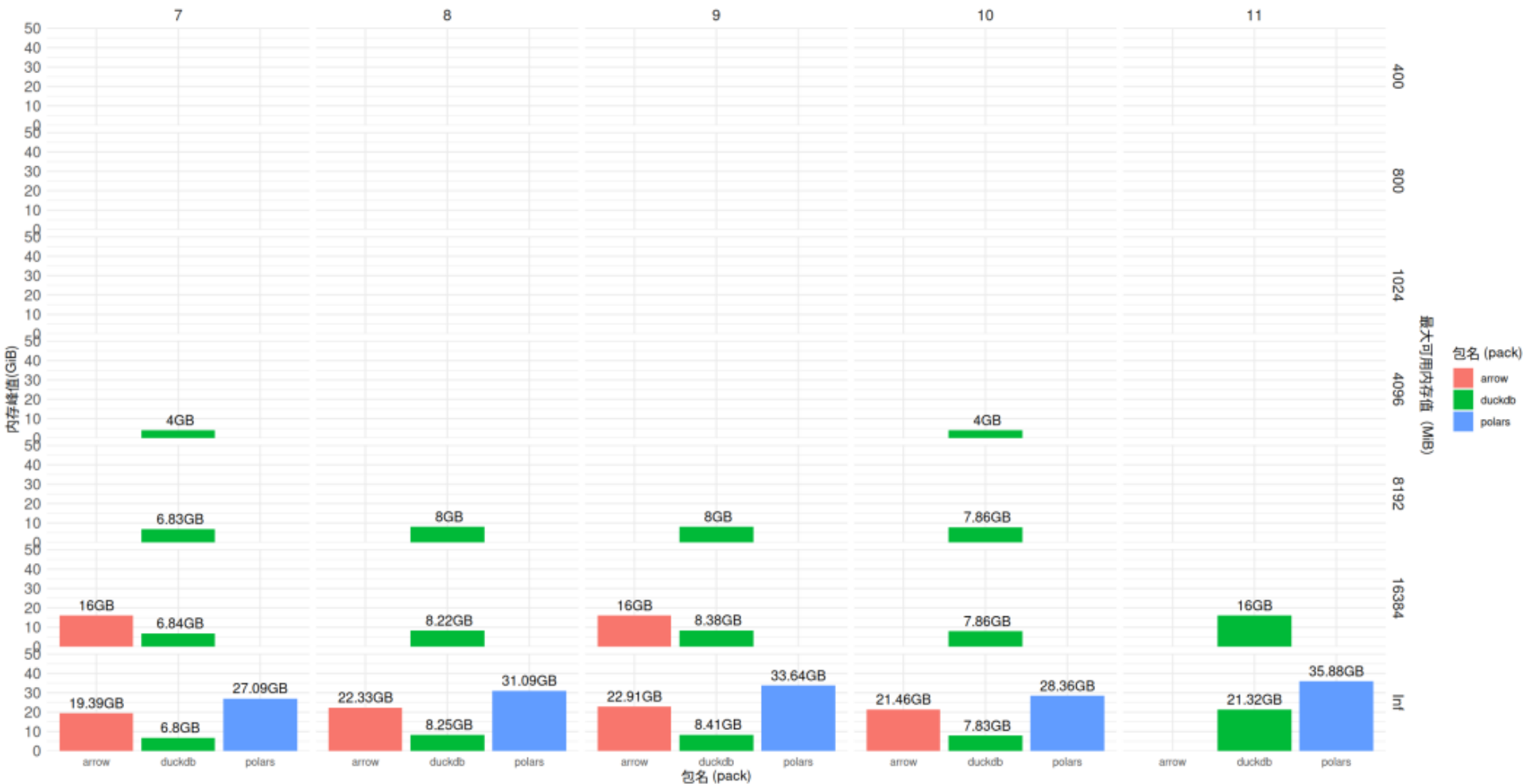
各包的JOIN测试用例在内存限制下的运行时间





# 线下个人试验：性能比较（限制内存）

各包的JOIN测试用例在内存限制下的内存峰值





## 框架比较的结论

总体来说，表现大体为：**DuckDB > Polars > Arrow**。内存充足的时候，**Polars**对于连接（**JOIN**）操作而言比较快，但是总体来说**DuckDB**对于内存不足计算的适配性更强，能够处理超过自身内存容量的数据（内存为32G的时候也可以处理50G的数据）。





# 目录

- 内存不足的实践背景
- 前沿大数据分析框架
- R语言中的相关工具
- 三种框架的性能比较
- 总结与展望





## 总结：语言支持

- **Arrow:** 多语言绑定（Python/R/Java等），生态最广，但需配合计算引擎使用（**列式存储的数据格式标准**）。
- **Polars:** 主推Python/R API，DataFrame语法近似pandas。
- **DuckDB:** 支持SQL+Python/R/Java等API，适合SQL重度用户（**数据库操作**）。





## 使用建议

- 1、使用**DuckDB**进行数据存储（数据库）
- 2、能够使用**SQL**的情况下，在**DuckDB**中调取数据，然后保存为**Parquet**格式，以便于跨平台交流协作
- 3、对于保存在本地的**Parquet**文件，使用**Polars**进行数据操作。对于以上框架不支持的操作，将数据载入内存中处理，再导出结果（足够小时，可以考虑使用**csv**进行交流）。







# 欢迎合作与交流！

## 黄天元



浙江财经大学数据科学学院数据科学与大数据技术系讲师，复旦大学理学博士。热爱数据科学与开源工具，致力于利用数据科学迅速积累行业经验优势和科学知识发现，著有《机器学习全解（R语言版）》、《文本数据挖掘——基于R语言》、《R语言高效数据处理指南》。知乎专栏：R语言数据挖掘，邮箱：[huangtianyuan@zufe.edu.cn](mailto:huangtianyuan@zufe.edu.cn)。

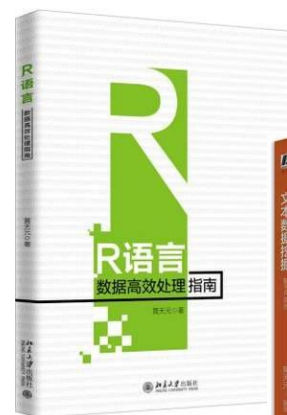
### R语言数据挖掘

### 知乎

兴趣使然的数据科学家



HopeR · 328 篇内容 · 4287 赞同 · 2117 订阅



浙江财经大学  
Zhejiang University of Finance & Economic



# 谢谢!



浙江财经大学数据科学学院  
数据科学与大数据技术系  
黄天元

[huangtianyuan@zufe.edu.cn](mailto:huangtianyuan@zufe.edu.cn)

