Scaling Up Data Analysis in R with Arrow

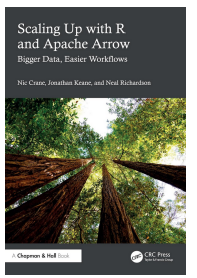
Nic Crane

Today

- The challenges of working with larger data in R
- What Arrow and Parquet are and how they help
- Analysing data using {arrow} (with demos)
- Real-world examples
- The wider Arrow ecosystem

About Me

- One of the Apache Arrow R package maintainers
- Part of Arctos Alliance
- Co-author of *Scaling Up with R and Arrow* (with Jonathan Keane and Neal Richardson)
- R consultant and trainer (niccrane.com)



Working with ~~Big~~ Larger-than-Memory Data in R

Big Data Naivety

- Data stored in Excel or CSV files
- Databases existed but weren’t part of day-to-day work
- Data was small enough that size wasn’t a problem

Consulting Learnings

- Exposed to larger, more complex datasets
- Learned dplyr, some data.table
- Larger data typically stored in databases

The Problem with Workarounds

- Query with SQL to get subsets to analyse
- Workarounds: stored procedures, APIs, batch processing, parallel execution
- These solutions work... but come at a cost.

Proprietary Software

- Data shipped in proprietary formats
- Expensive license fees
- No additional benefit over open source

Fractured Pipelines

- Analyse in one program
- Export to another for write-up
- Manual, time-consuming, error-prone
- Changes upstream = redo everything downstream

Expensive or Complex Infrastructure

- APIs, SQL queries, stored procedures
- Spark on clusters
- Setup time, maintenance, access configuration
- Cross-team dependencies, red tape
- New skillsets to learn

Scaling Up Ordinary Data Workflows

- We just want to work with the same data, at larger scale

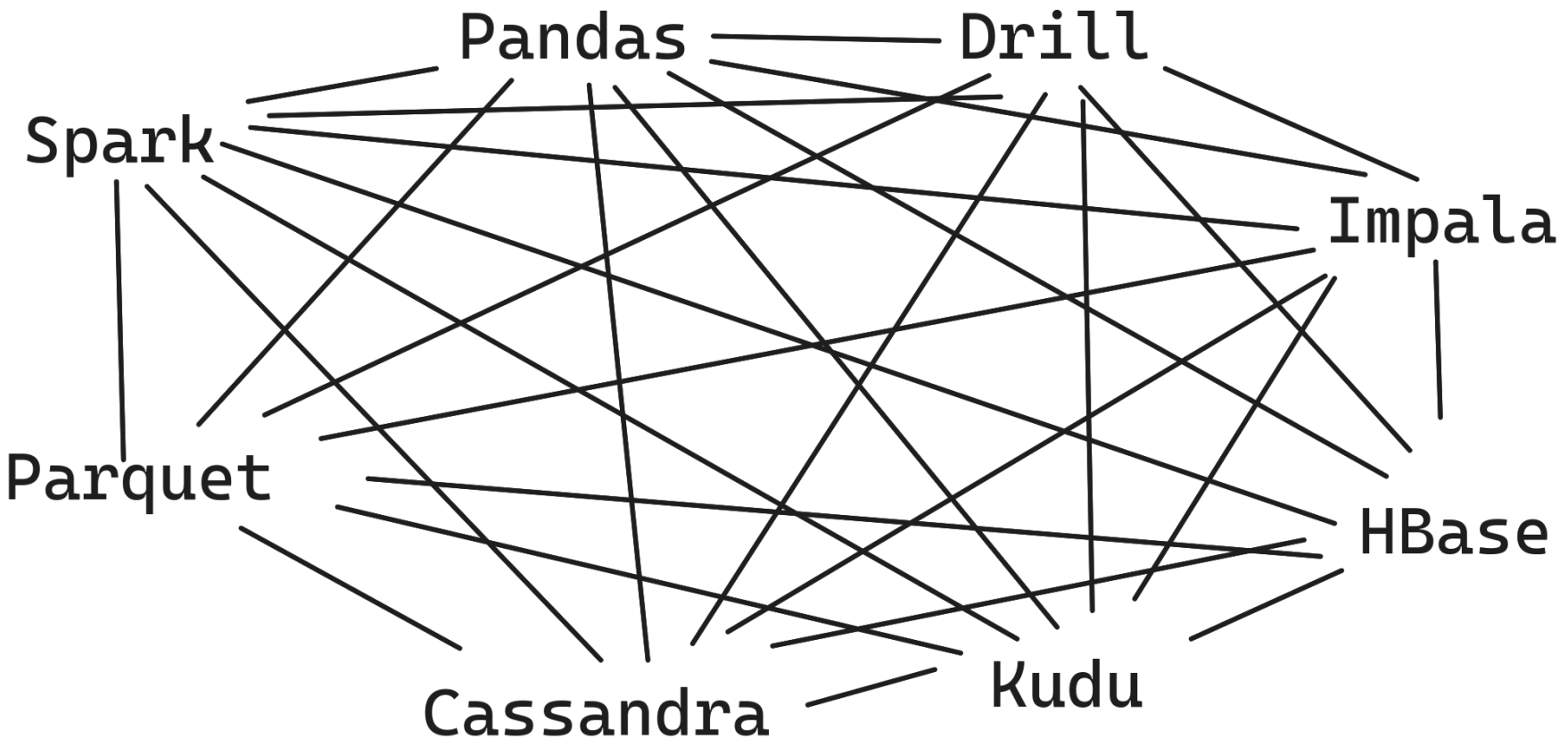
What If...

- Keep using the same dplyr code
- Work with larger than memory data
- No overcomplicated infrastructure
- No expensive proprietary software
- Keep your whole pipeline in R

You can do this with {arrow}

What is Arrow?

The Origin Story

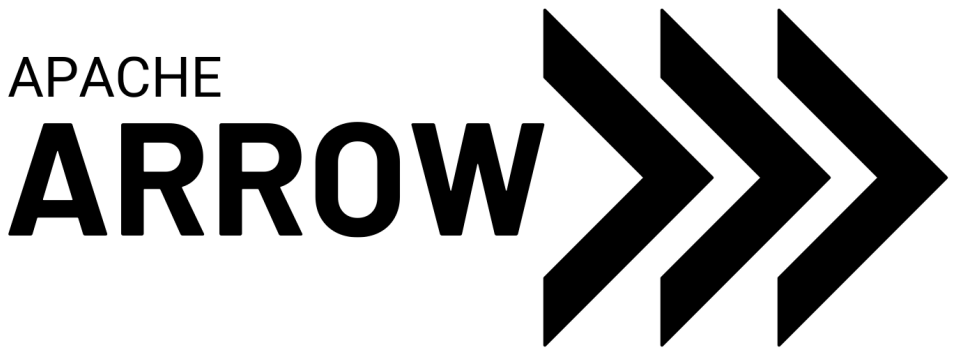


- Mid-2010s: Wes McKinney (creator of pandas) asked:
- “Why isn’t there a standardized, efficient in-memory format for tabular data?”

Hardware Had Changed

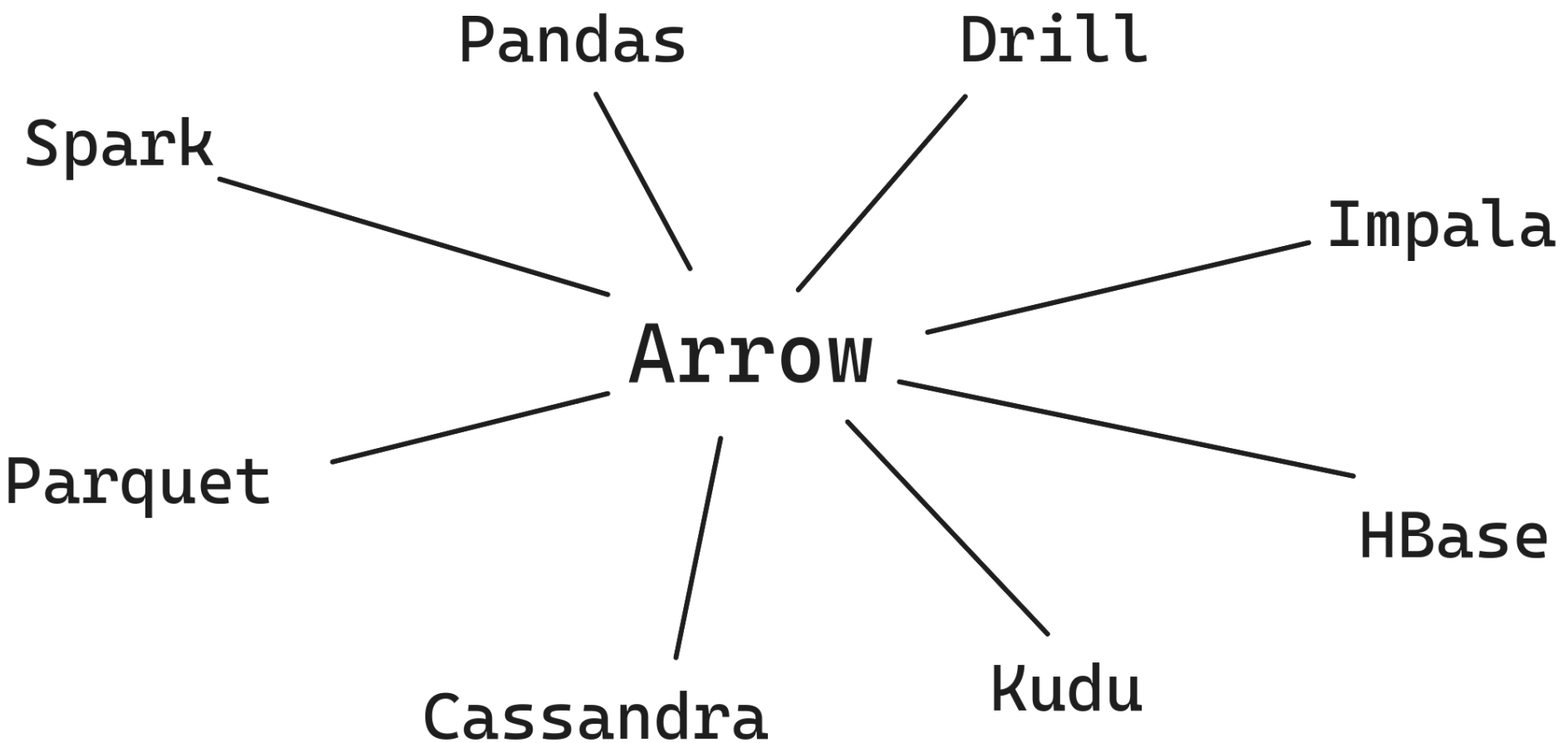
- Solid state drives got much faster
- Machines had multiple cores
- What used to require clusters could now run on laptops

Arrow: The Specification



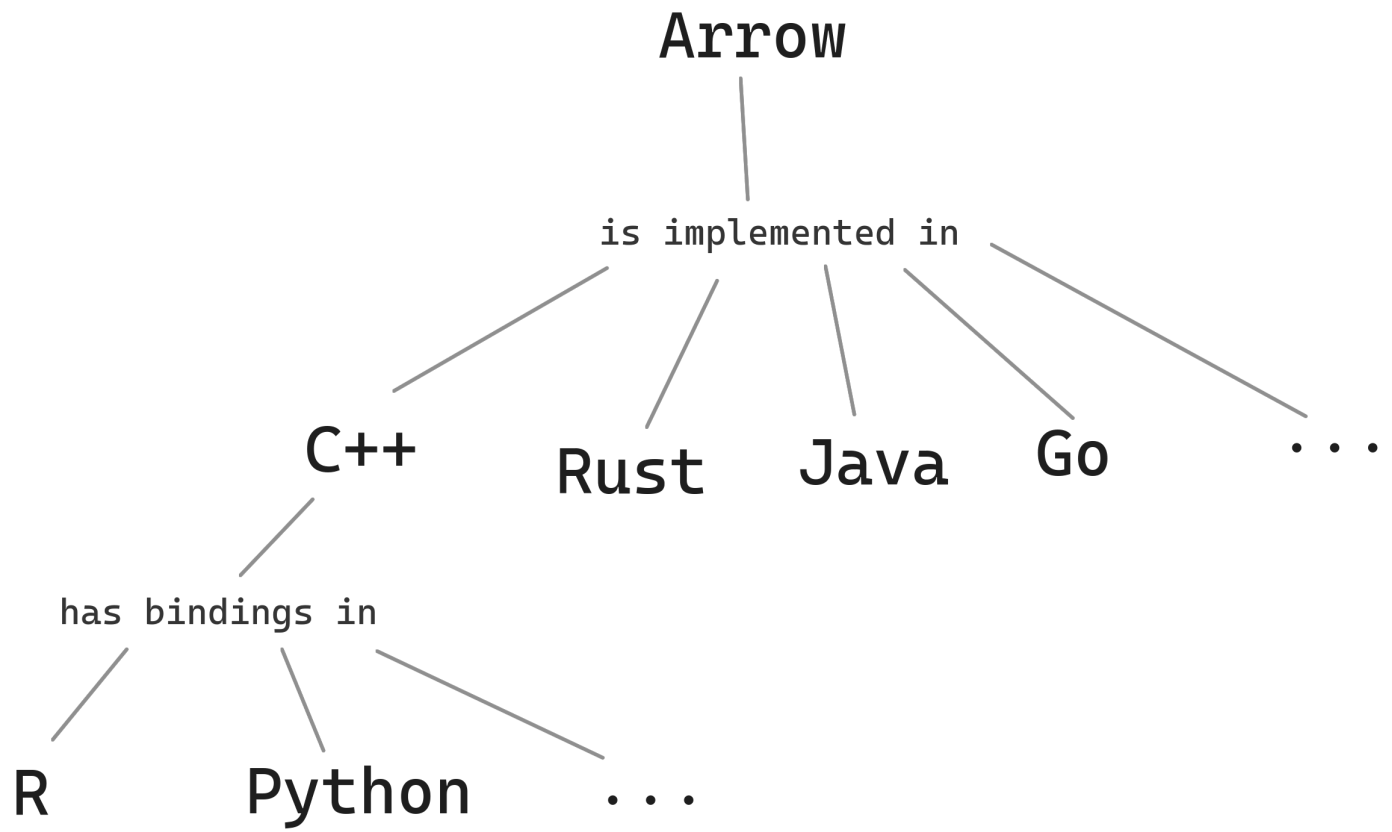
- Created in 2016
- A standard for representing tabular data in memory
- Designed for analytical workloads
- Works across programming languages
- Columnar: stores data column by column

Why a Standard Matters



- Different libraries can share data without conversion
- No need to copy data between systems

Arrow Implementations

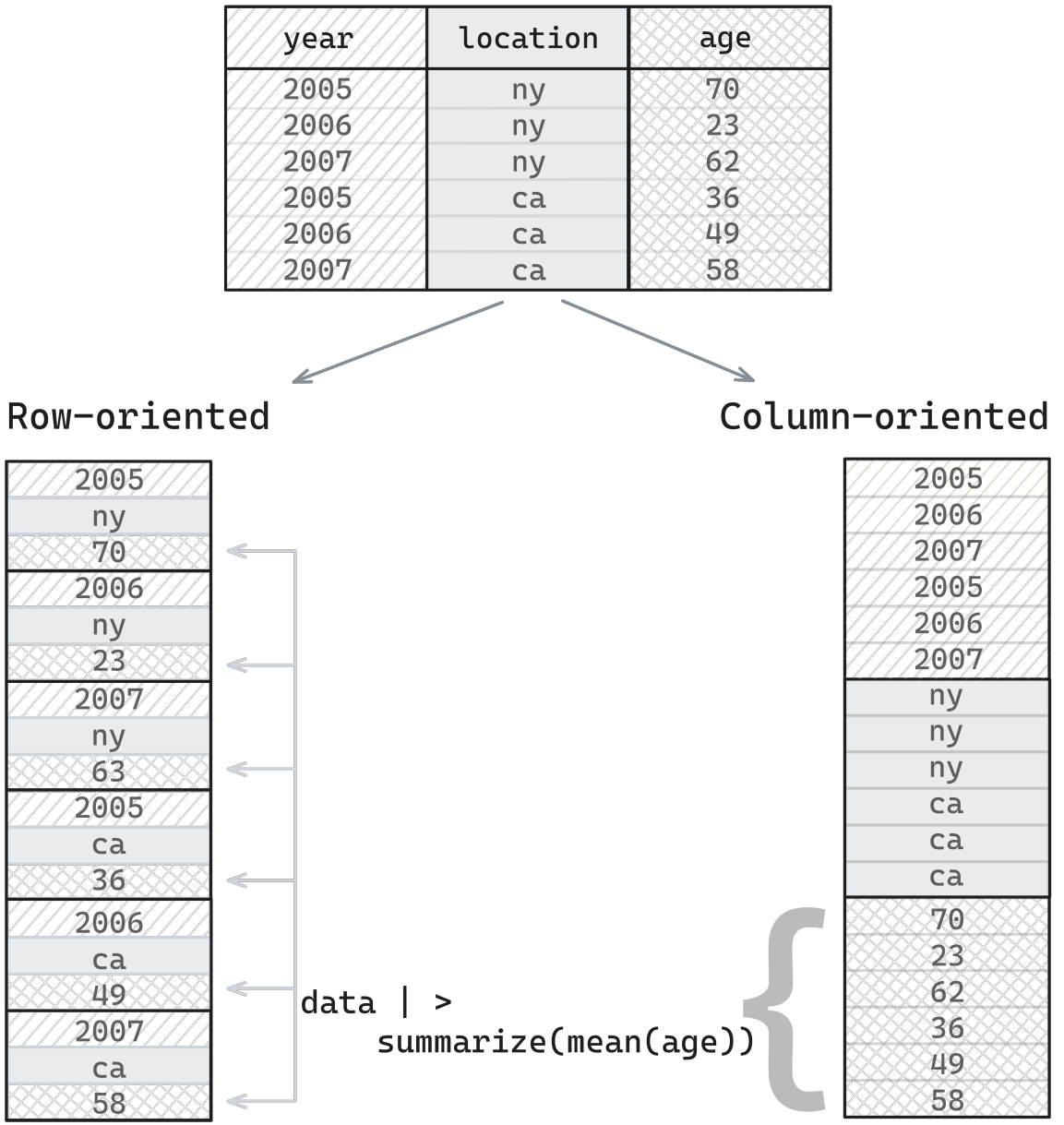


- Arrow = the specification AND the implementations
- Key implementation for us: Arrow C++ library
- The arrow R package is built on Arrow C++

The Problem with CSVs

- Row-oriented: read everything even for a few columns
- No data type information stored
- Software has to guess types (and sometimes gets it wrong)

Columnar Format



Parquet: The Solution

- Columnar: only read the columns you need
- Smart compression: repeated values stored efficiently
- Smaller files, faster reads
- Schema stored with data: no guessing, no information loss

The {arrow} R Package

Demo 1: Fast Aggregation

What is the mean age of respondents in Washington and how does that change over the years?

Demo 1: Code

```
library(arrow)
library(tictoc)

pums_person <- open_dataset("./data/person")
nrow(pums_person)

tic()
pums_person |>
  filter(location == "wa") |>
  group_by(year) |>
  summarise(mean_age = sum(AGEP * PWGTP) / sum(PWGTP)) |>
  collect()
toc()
```

The dplyr API

- Use dplyr code to work with Arrow data
- If you know dplyr, you know how to use arrow
- Same verbs: filter, mutate, group_by, summarise
- 37 dplyr verbs, 220+ R functions
- Includes stringr and lubridate bindings

Demo 2: dplyr on Arrow with stringr

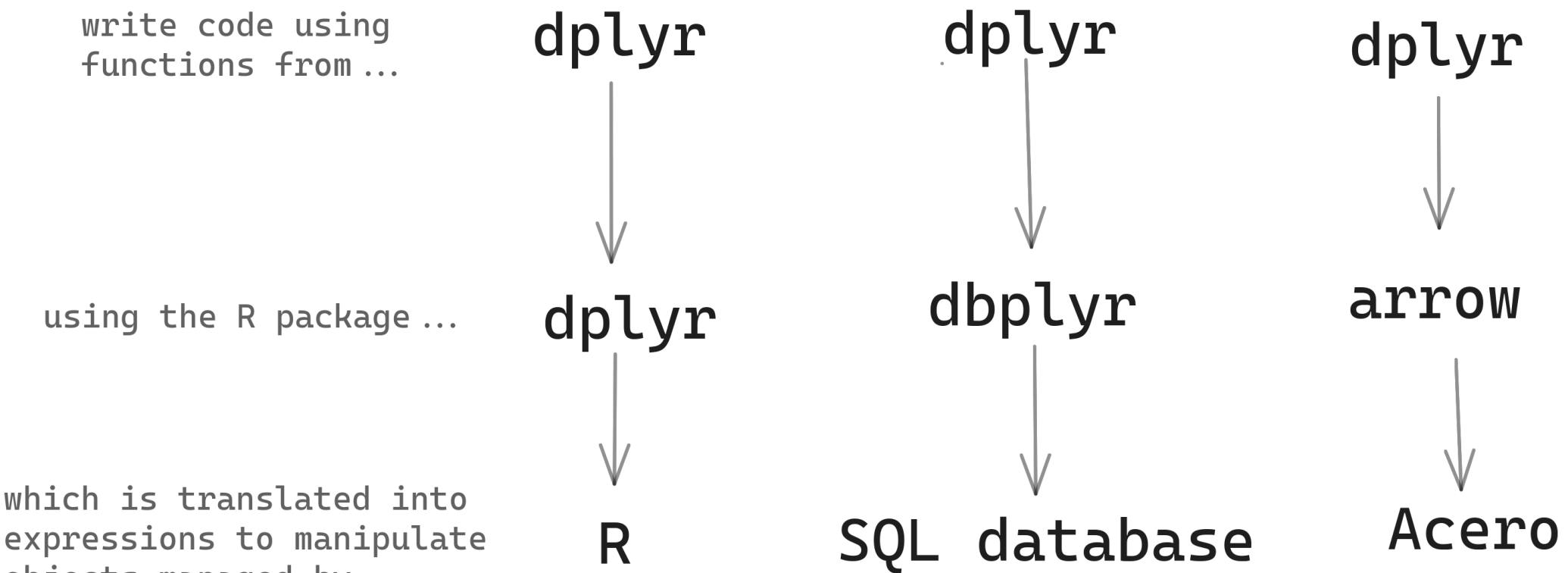
Which states have the highest proportion of adults with higher education?

Demo 2: Code

```
pums_person |>
  filter(AGEP > 18) |>
  mutate(
    higher_ed = if_else(
      str_detect(SCHL, "(Bach|Mast|Prof|Doct|college|degree)"), 1L, 0L
    )
  ) |>
  group_by(location) |>
  summarise(prop_higher_ed = sum(higher_ed * PWGTP) / sum(PWGTP)) |>
  arrange(desc(prop_higher_ed)) |>
  head(3) |>
  collect()

#> # A tibble: 3 × 2
#>   location prop_higher_ed
#>   <chr>             <dbl>
#> 1 dc                0.709
#> 2 co                0.679
#> 3 ut                0.670
```

How Does This Work?



Demo 3: Custom Functions

Write your own R functions - they work if the internals have Arrow bindings

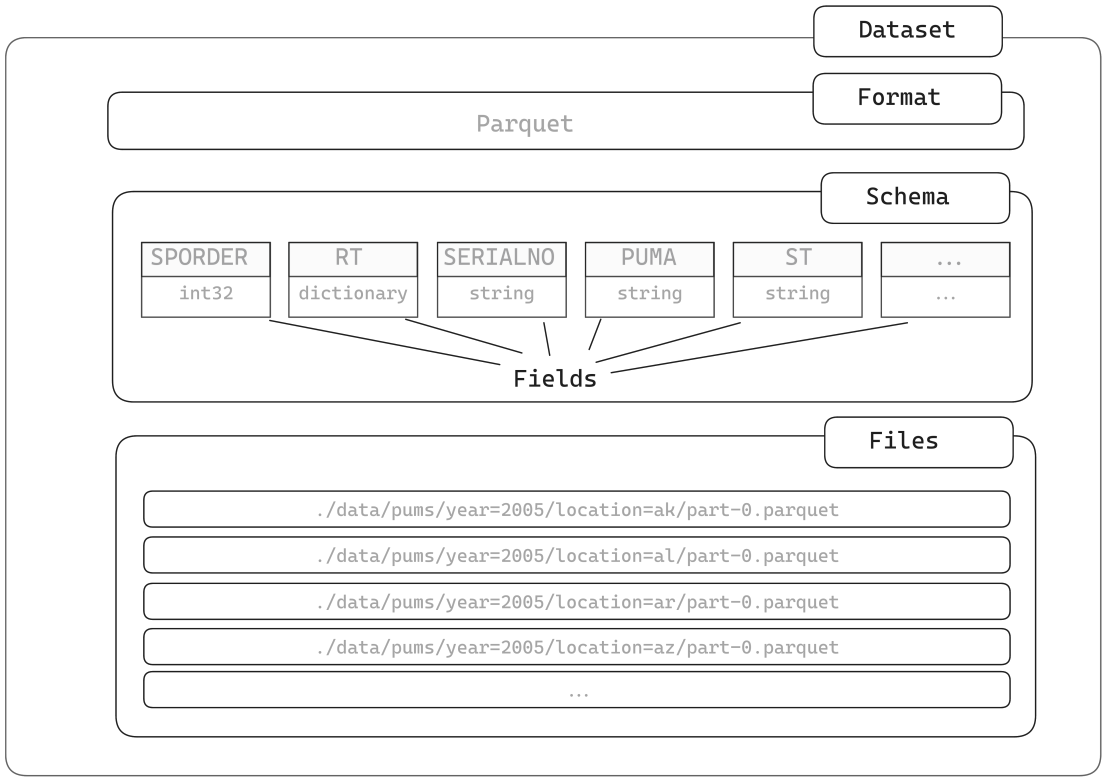
Demo 3: Code

```
time_hours_rounded <- function(mins) {
  round(mins / 60, 2)
}

pums_person |>
  filter(location == "wa") |>
  select(JWMNP) |>
  mutate(commute_hours = time_hours_rounded(JWMNP)) |>
  head() |>
  collect()

#> # A tibble: 6 × 2
#>   JWMNP commute_hours
#>   <int>         <dbl>
#> 1    NA         NA
#> 2    10          0.17
#> 3    10          0.17
#> 4    10          0.17
#> 5    37          0.62
```

Datasets and Lazy Evaluation



- Arrow dataset = similar to a database connection
- Point at files, nothing loaded into memory yet
- dplyr pipeline constructs query but doesn't run it

`collect()` executes and pulls results into R

Demo 4: CSV vs Parquet

Same query, same data (Washington state, all years)

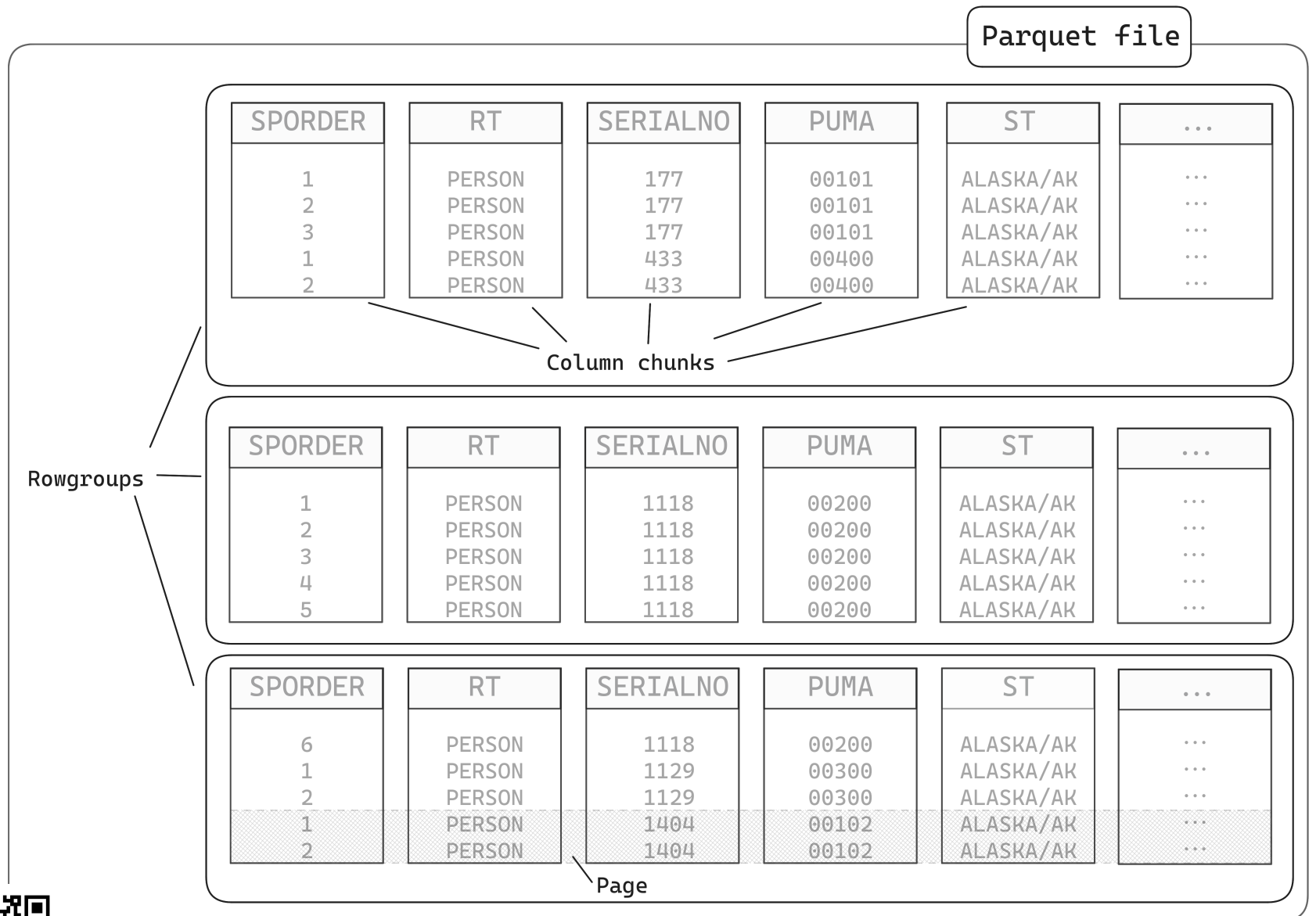
Demo 4: Code

```
# CSV: 1.93 GB
tic()
open_dataset("data/wa_all_years.csv", format = "csv") |>
  group_by(year) |>
  summarise(mean_age = mean(AGEP, na.rm = TRUE)) |>
  collect()
toc()

# Parquet: 186MB
tic()
open_dataset("data/wa_all_years.parquet", format = "parquet") |>
  group_by(year) |>
  summarise(mean_age = mean(AGEP, na.rm = TRUE)) |>
  collect()
toc()
```

10x smaller, 20x faster

How Does This Work?



Demo 5: Partitioned Datasets

Unpartitioned vs partitioned Parquet - querying the full dataset

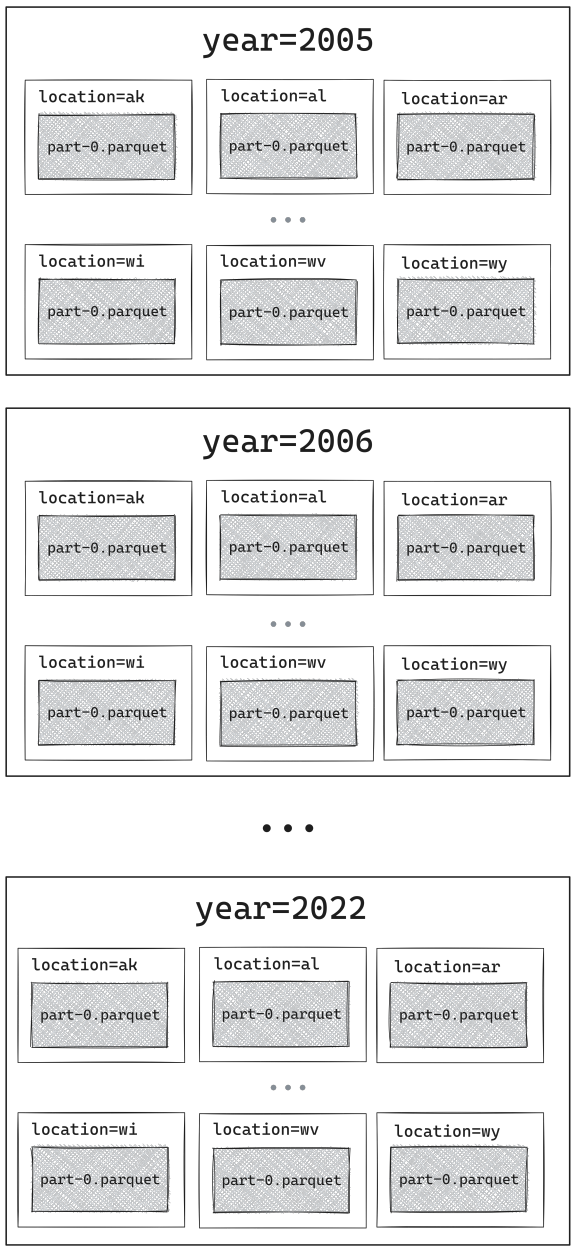
Demo 5: Code

```
# Single Parquet file with all data
tic()
open_dataset("data/one_big_file.parquet") |>
  filter(location == "wa") |>
  group_by(year) |>
  summarise(mean_age = mean(AGEP, na.rm = TRUE)) |>
  collect()
toc()

# Partitioned: across year/location folders
tic()
open_dataset("data/person") |>
  filter(location == "wa") |>
  group_by(year) |>
  summarize(mean_age = mean(AGEP, na.rm = TRUE)) |>
  collect()
toc()
```

6x faster - Arrow reads multiple files in parallel

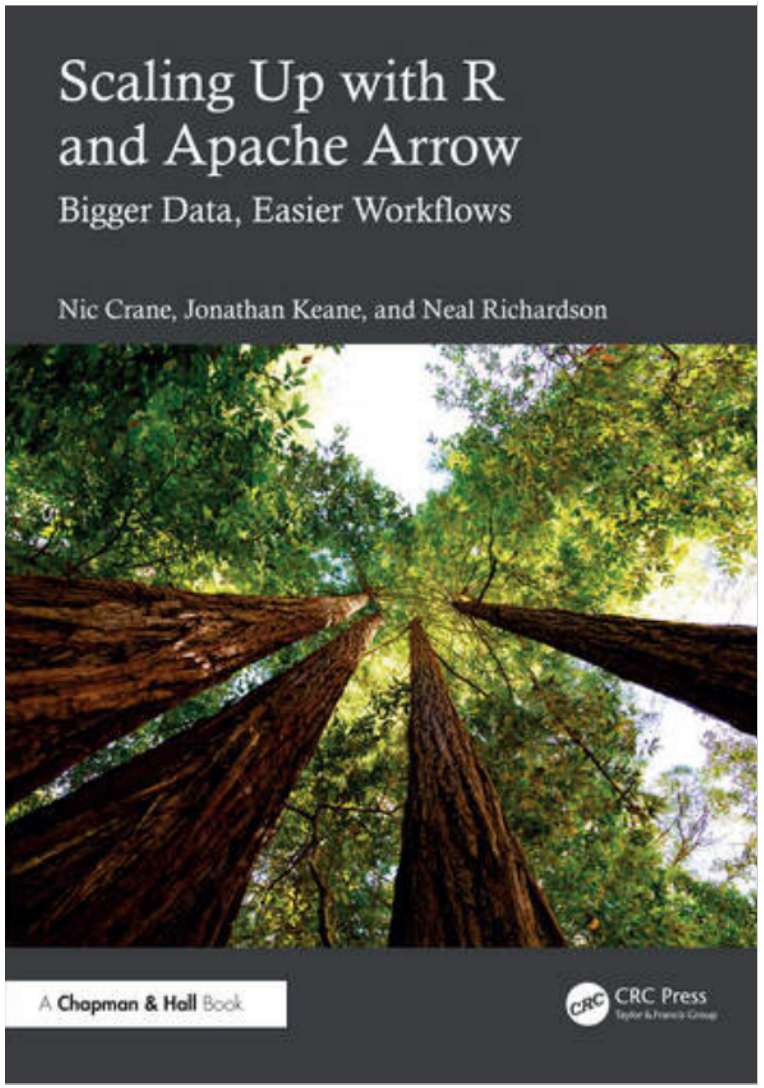
How Does This Work?



Converting to Parquet

```
1 open_dataset("pums_csv") |>
2   write_dataset("pums_parquet", partitioning = "location")
```

Learn More



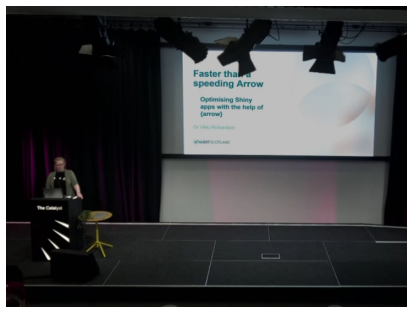
e book *Scaling Up with R and Arrow* is free online at

rowrbook.com

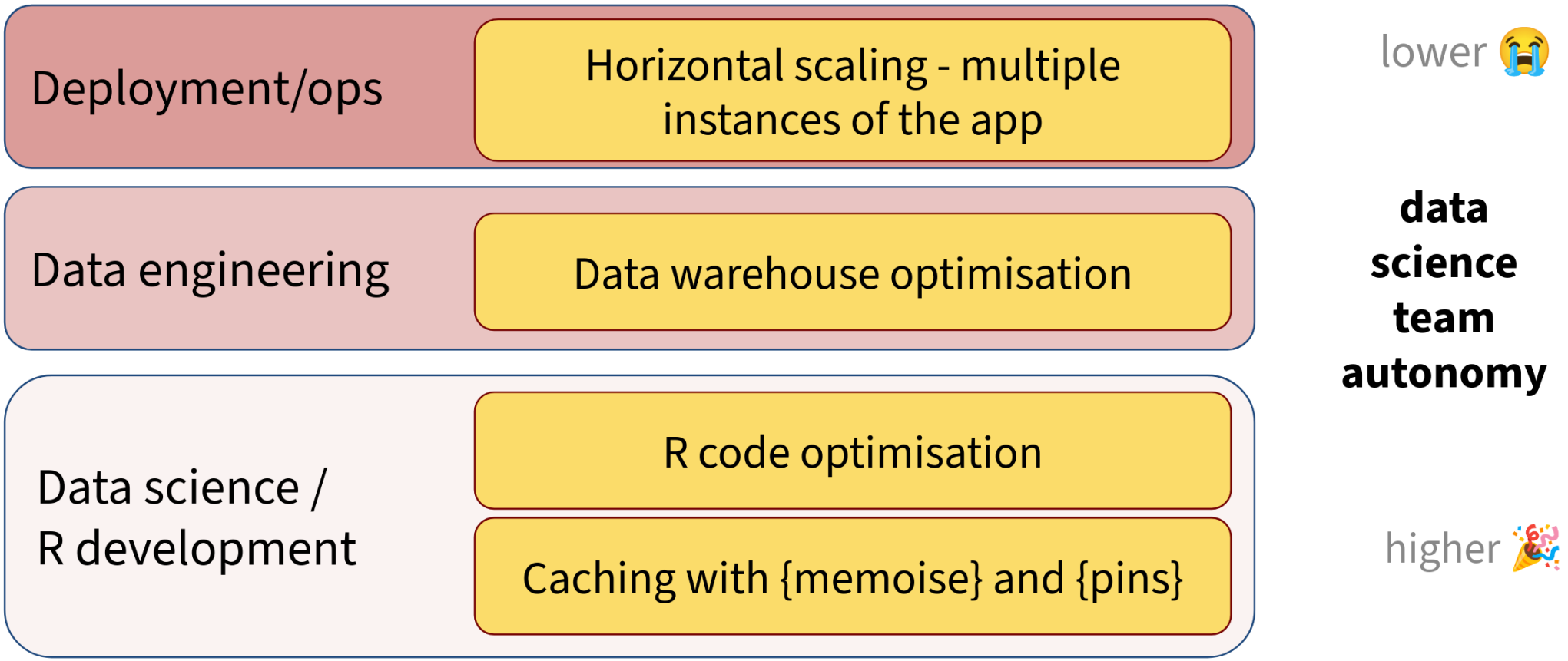
Real World Examples

Shiny App Speedup

- Vikki Richardson, Audit Scotland (Shiny in Production 2024)
- Financial ledger auditing app
- 6 million rows, 42 columns
- Original load: 4 minutes
- With memoise caching: 30 seconds
- With Arrow + pins: **2 seconds**

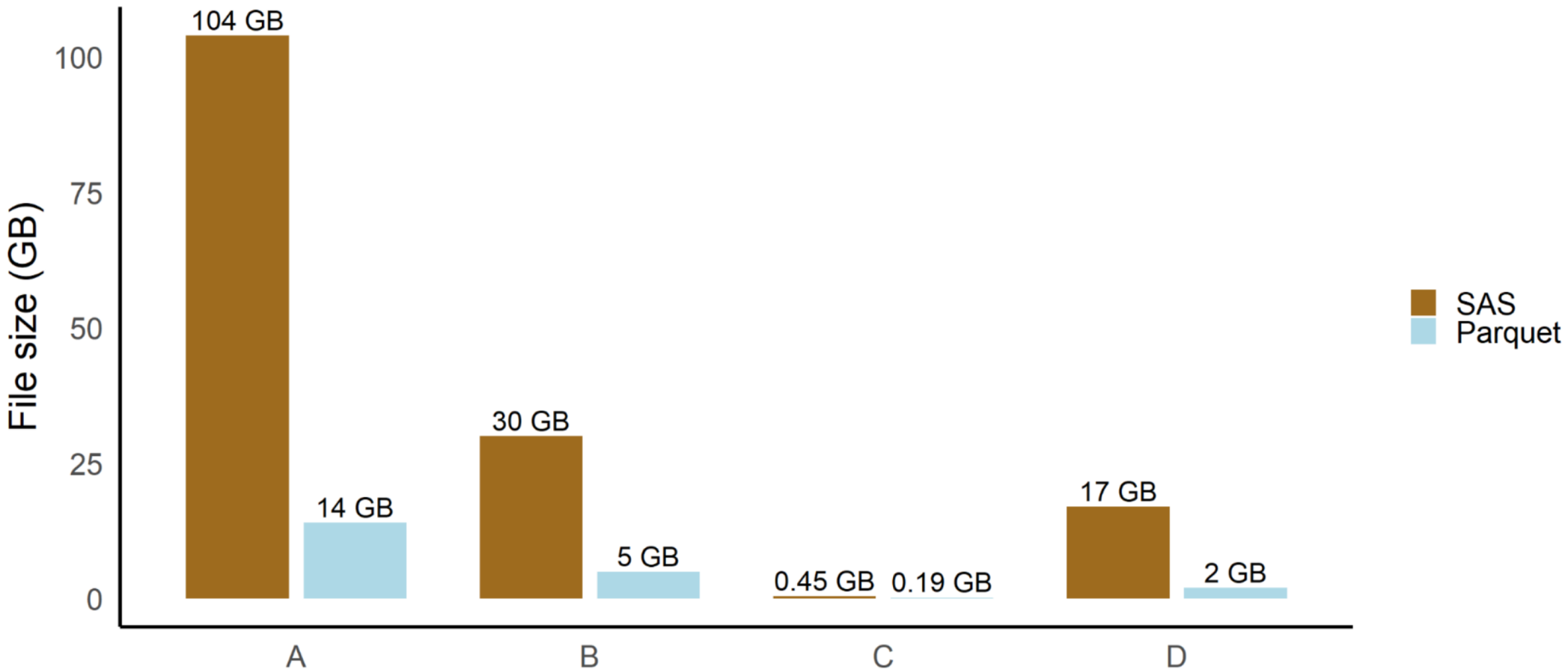


Why This Matters

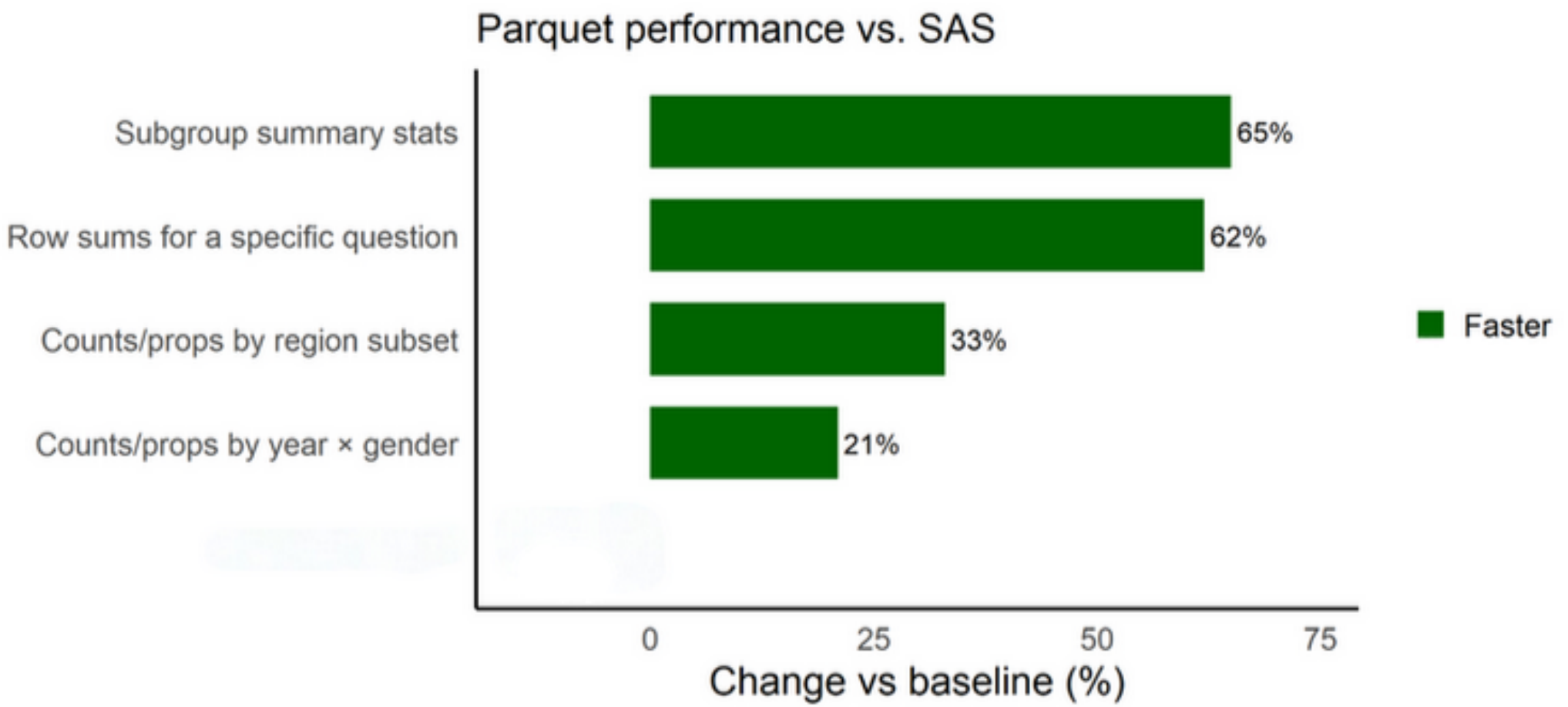


Pharma and Open Source

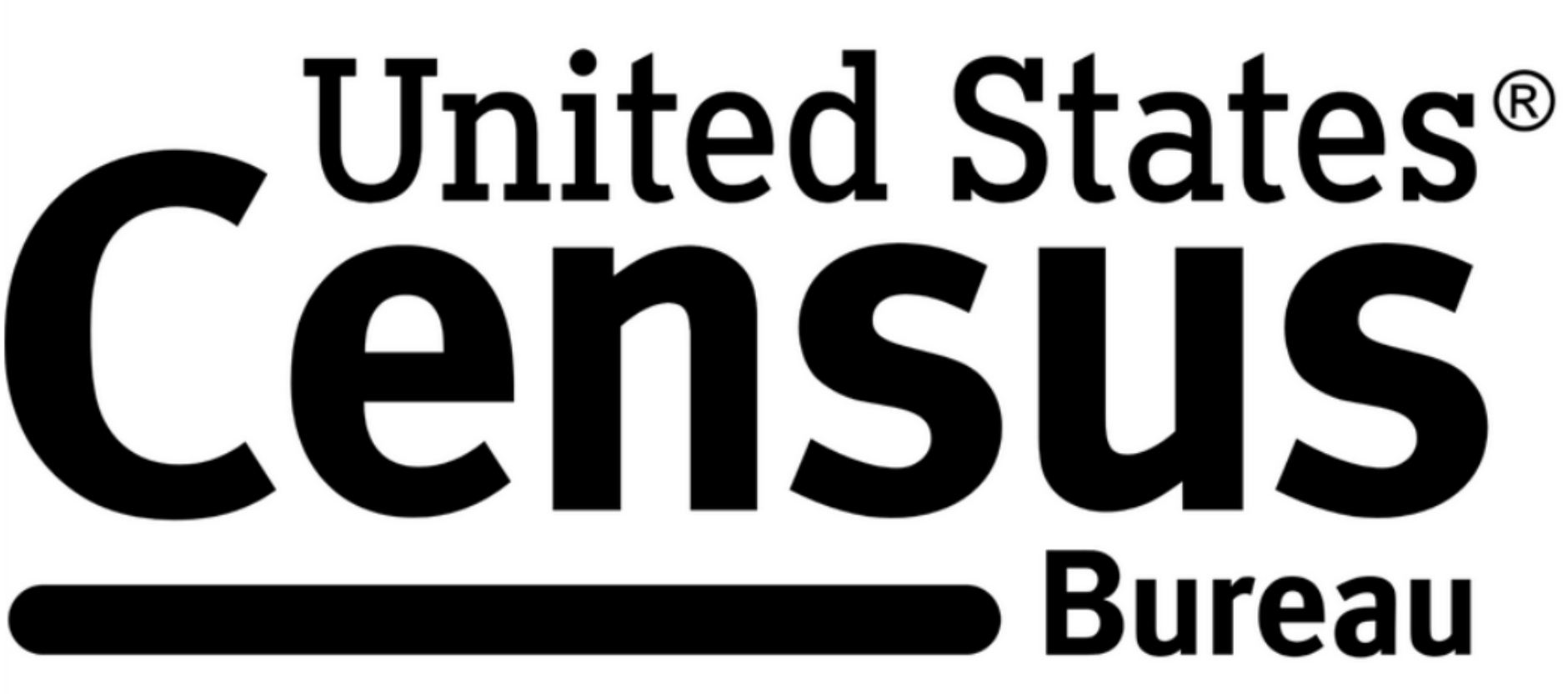
The Results


104 GB
14 GB
30 GB
5 GB
0.45 GB
0.19 GB
17 GB
2 GB
100
75
50
25
0
File size (GB)
A
B
C
D
SAS
Parquet

The Results

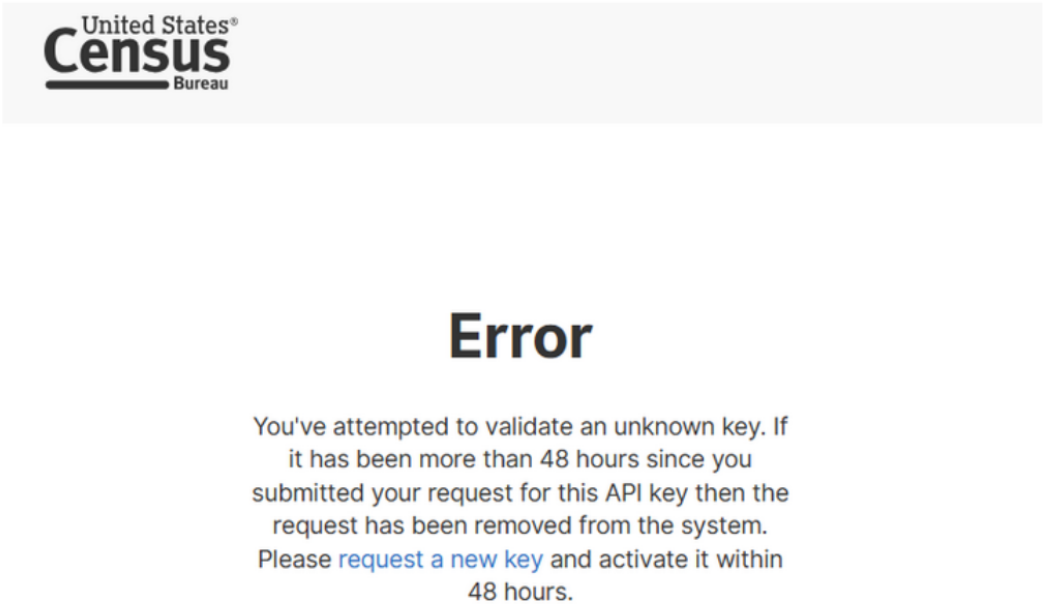

Parquet performance vs. SAS
Subgroup summary stats
Row sums for a specific question
Counts/props by region subset
Counts/props by year × gender
65%
62%
33%
21%
Faster
0
25
50
75
Change vs baseline (%)

Survey Data Example



- US Census PUMS dataset (from the Arrow book)
- 53 million rows, 311 columns, 45GB CSV

Government Shutdown!




```
Error in `get_decennial()`:
! Error : You have supplied an inval
id or inactive API key. To obtain a
valid API key, visit https://api.cen
sus.gov/data/key_signup.html. To act
ivate your key, be sure to click the
link provided to you in the email fr
om the Census Bureau that contained
your key.
Run `rlang::last_trace()` to see whe
re the error occurred.
```

- tidycensus uses Census API
- Government shutdown took down the API while writing a previous talk!

The Wider Arrow Ecosystem

Arrow as Infrastructure

- Arrow isn’t just a package - it’s a standard
- Many modern data tools are built on top of it
- Learning about Arrow means understanding how these tools work

Tools for Data Manipulation

Polars

- Uses Arrow as its internal format
- Zero-copy data sharing with other Arrow tools

DuckDB

- Uses Arrow for zero-copy data interchange
- Can query Arrow data directly without copying

Parquet Everywhere

- BigQuery, Snowflake, Databricks all read Parquet natively
- AWS Athena, Redshift Spectrum designed around Parquet
- Delta Lake uses Parquet as its underlying file format
- Your local Parquet files work with cloud infrastructure

ADBC: Faster Database Connections

```r
library(adbcdrivermanager)

# Use the driver manager to connect to a database
db <- adbc_database_init(adbcsqlite::adbcsqlite(), uri = ":memory:")
con <- adbc_connection_init(db)

# Write a table
mtcars |>
  write_adbc(con, "mtcars")

# Query it
con |>
  read_adbc("SELECT * from mtcars") |>
  tibble::as_tibble()

# Clean up
con |>
  execute_adbc("DROP TABLE mtcars")
adbc_connection_release(con)
```

- Traditional interfaces (ODBC/JDBC) designed for row-wise data
- Converting columnar data to rows is expensive
- ADBC speaks Arrow natively: 38x faster in benchmarks
- https://arrow.apache.org/adbc/current/r/index.html

GeoArrow and GeoParquet

- Spatial data joining the Arrow ecosystem
- GeoParquet: spatial data in Parquet files
- GeoArrow: spatial data representation in memory

geoarrow R package bridges sf and arrow

Other Packages in the R Arrow Ecosystem

- **nanoarrow**: lightweight Arrow support for package developers
- **nanoparquet**: dependency-free Parquet reader/writer

The arrow Package: Your Entry Point

- Arrow and Parquet are the foundation of modern data tooling
- The arrow R package is how you tap into all of this from R
- Learn Arrow once, apply it everywhere

Wrapping Up

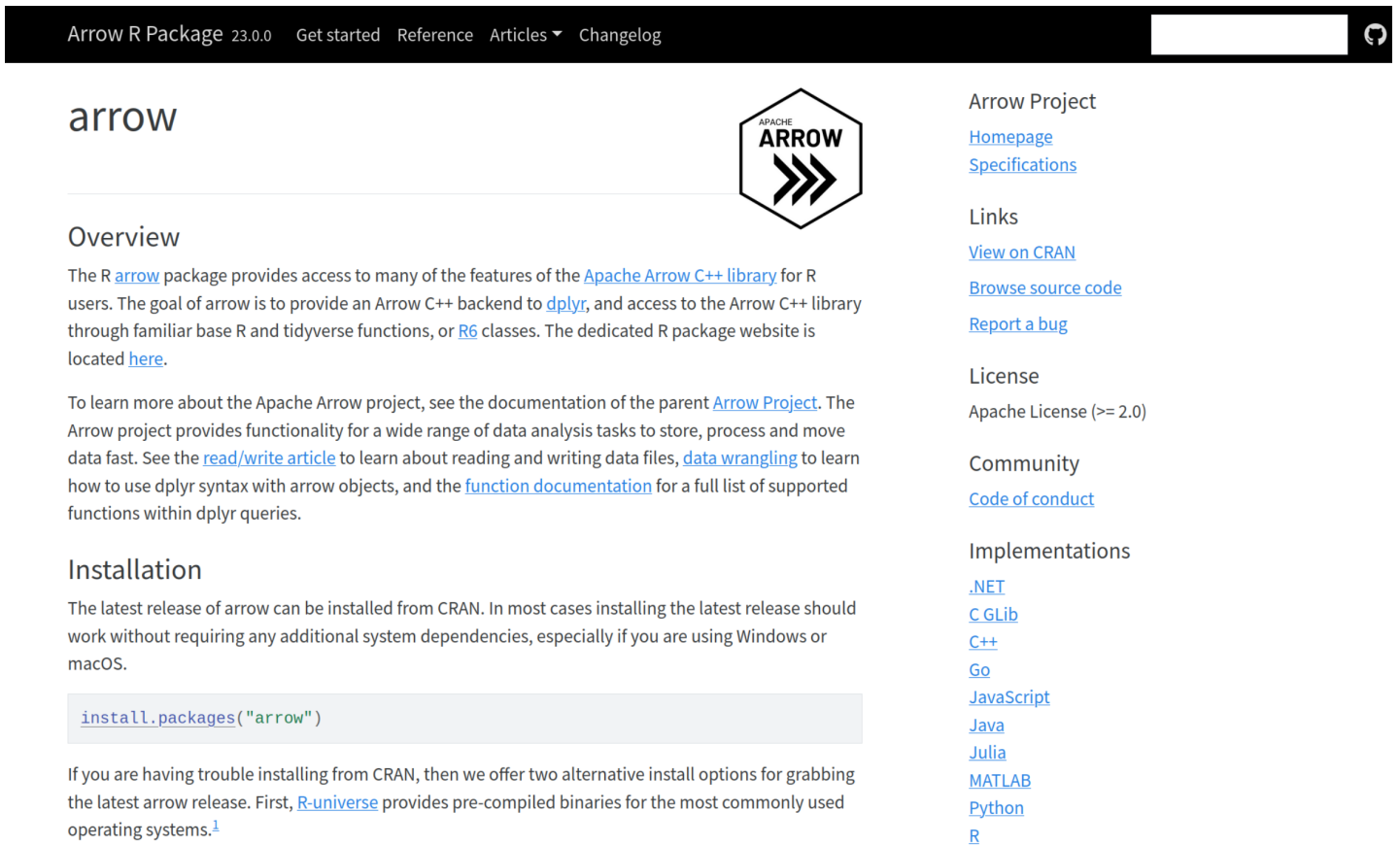
The Problems Arrow Solves

- Proprietary software lock-in
- Fractured pipelines
- Expensive infrastructure
- New skillsets just to work with bigger data

The Arrow Solution

- Work with larger-than-memory data
- Stay in R
- Use dplyr code you already know
- Keep your whole workflow in one place
- Open, free, works together

Resources



Arrow R Package 23.0.0 Get started Reference Articles Changelog

arrow

Overview

The R arrow package provides access to many of the features of the Apache Arrow C++ library for R users. The goal of arrow is to provide an Arrow C++ backend to dplyr, and access to the Arrow C++ library through familiar base R and tidyverse functions, or R6 classes. The dedicated R package website is located here.

To learn more about the Apache Arrow project, see the documentation of the parent Arrow Project. The Arrow project provides functionality for a wide range of data analysis tasks to store, process and move data fast. See the read/write article to learn about reading and writing data files, data wrangling to learn how to use dplyr syntax with arrow objects, and the function documentation for a full list of supported functions within dplyr queries.

Installation

The latest release of arrow can be installed from CRAN. In most cases installing the latest release should work without requiring any additional system dependencies, especially if you are using Windows or macOS.

```
install.packages("arrow")
```

If you are having trouble installing from CRAN, then we offer two alternative install options for grabbing the latest arrow release. First, R-universe provides pre-compiled binaries for the most commonly used operating systems.[1]

Arrow Project
Homepage
Specifications

Links
View on CRAN
Browse source code
Report a bug

License
Apache License (>= 2.0)

Community
Code of conduct

Implementations
.NET
C GLib
C++
Go
JavaScript
Java
Julia
MATLAB
Python
R

- Docs: **https://arrow.apache.org/docs/r/**
- Book: **arrowrbook.com** (free online)

Scan the QR code for links to workshops etc

Thank You

Questions?