Row-oriented workflows in R + tidyverse

Jennifer Bryan
RStudio, University of British Columbia
@JennyBryan @jennybc

rstd.io/row-work

GitHub repo has all code.
Link to slides on SpeakerDeck.

Get the .R files to play along.
Or follow via rendered .md.

I assume you know or want to know:

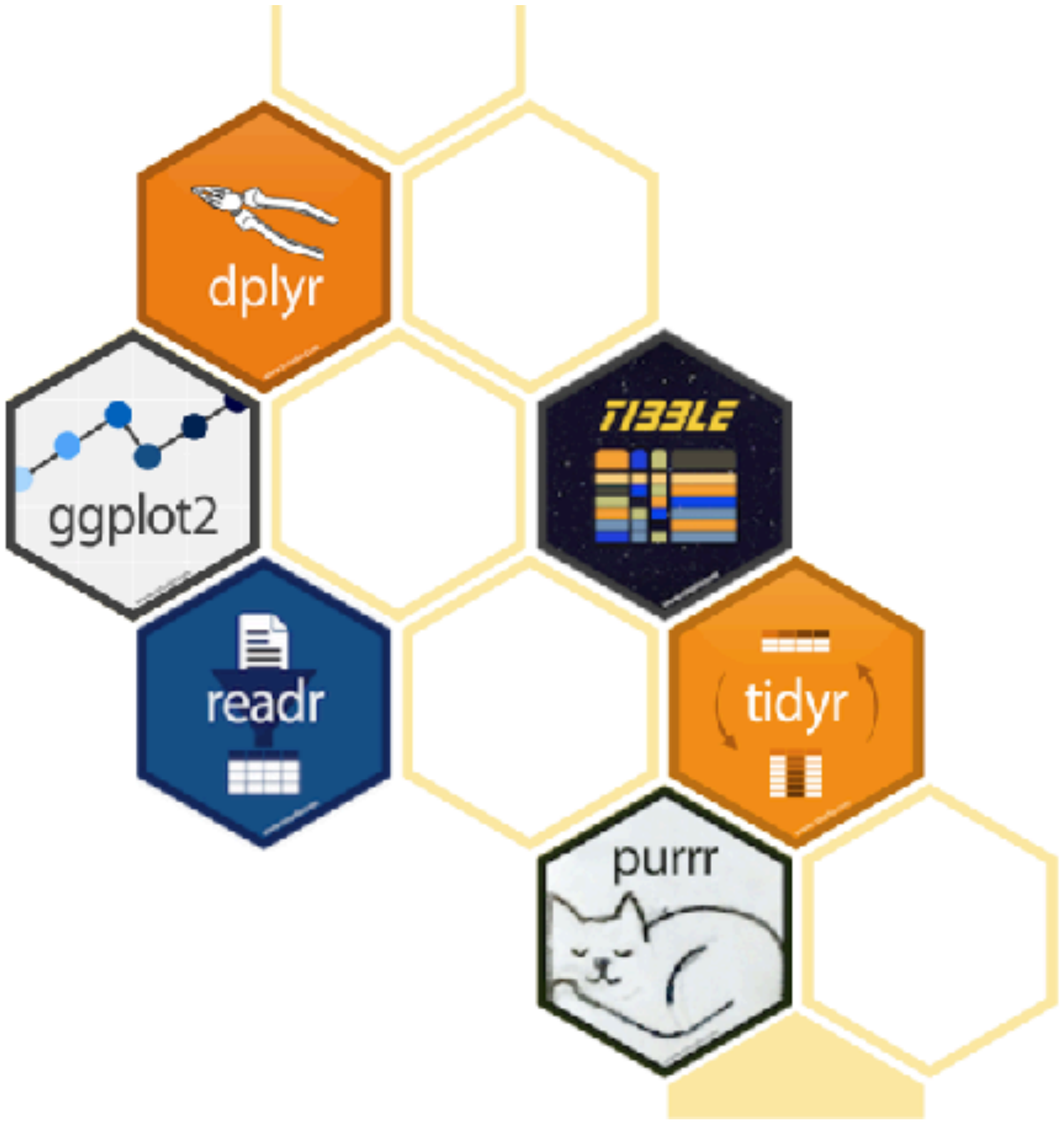
the tidyverse packages

the pipe operator, %>%

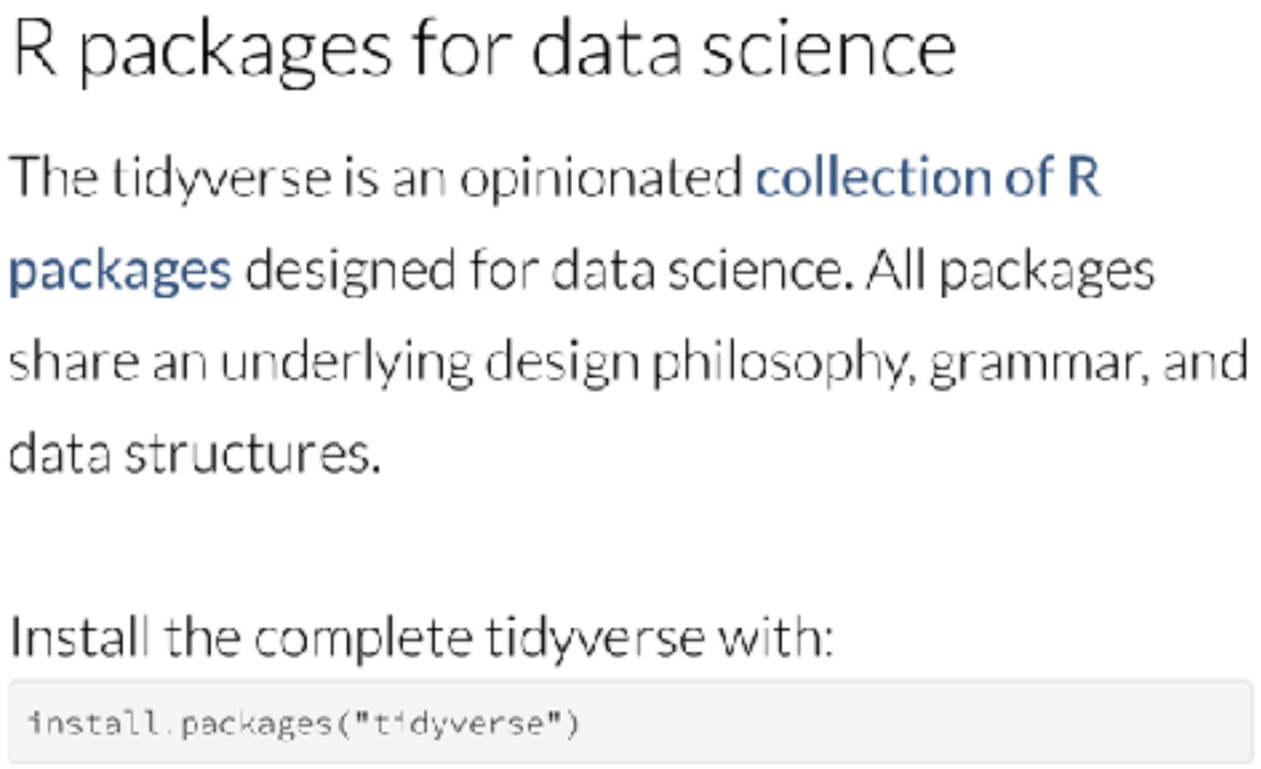
list = core data structure

"apply" or "map" functions,

  e.g. base::lapply() and purrr::map()



tidyverse.org



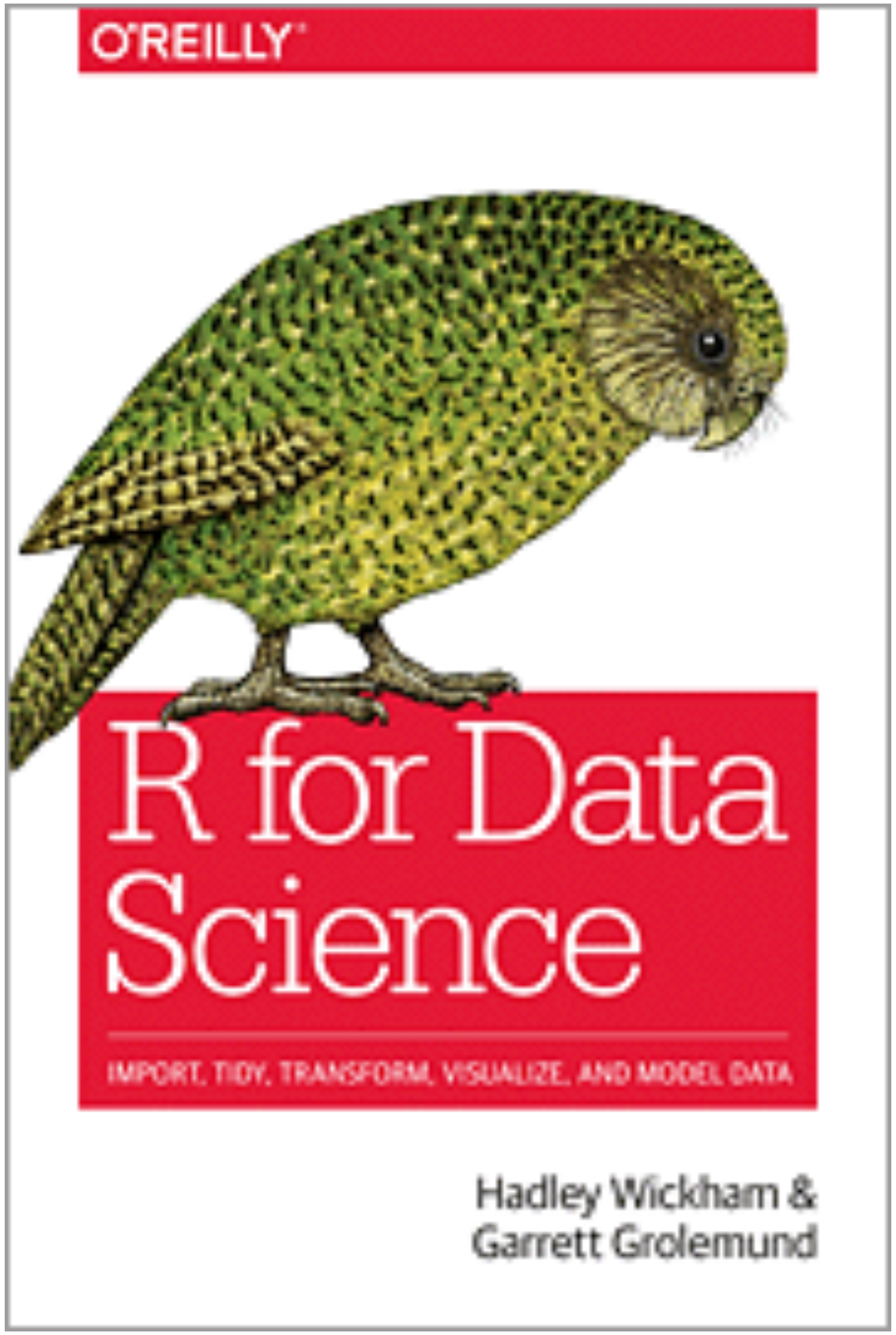
R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

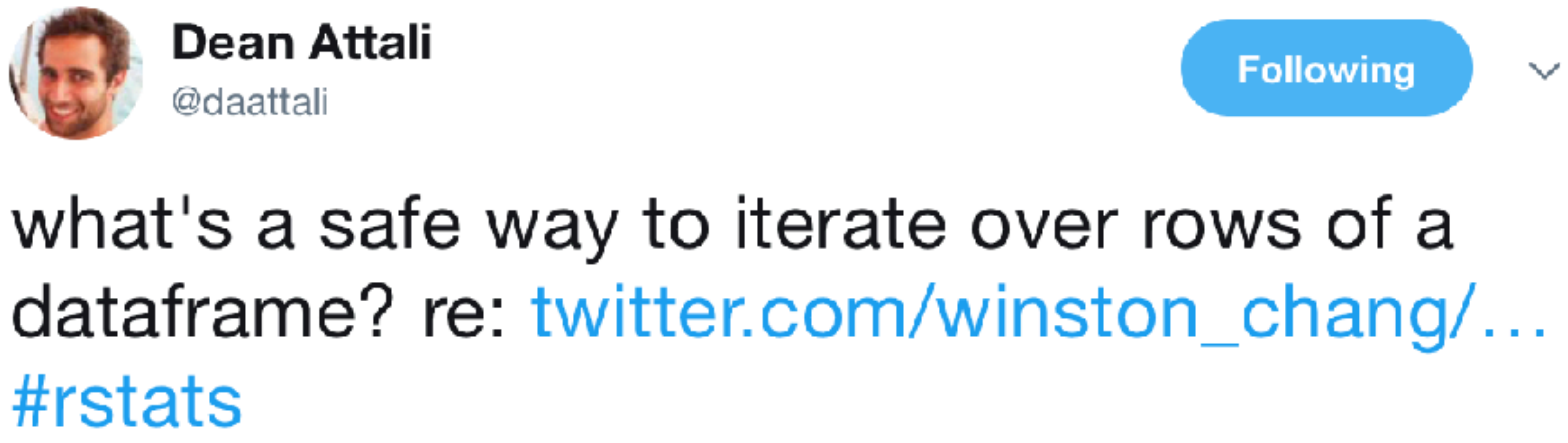
```
install.packages("tidyverse")
```

download materials: **rstd.io/row-work**



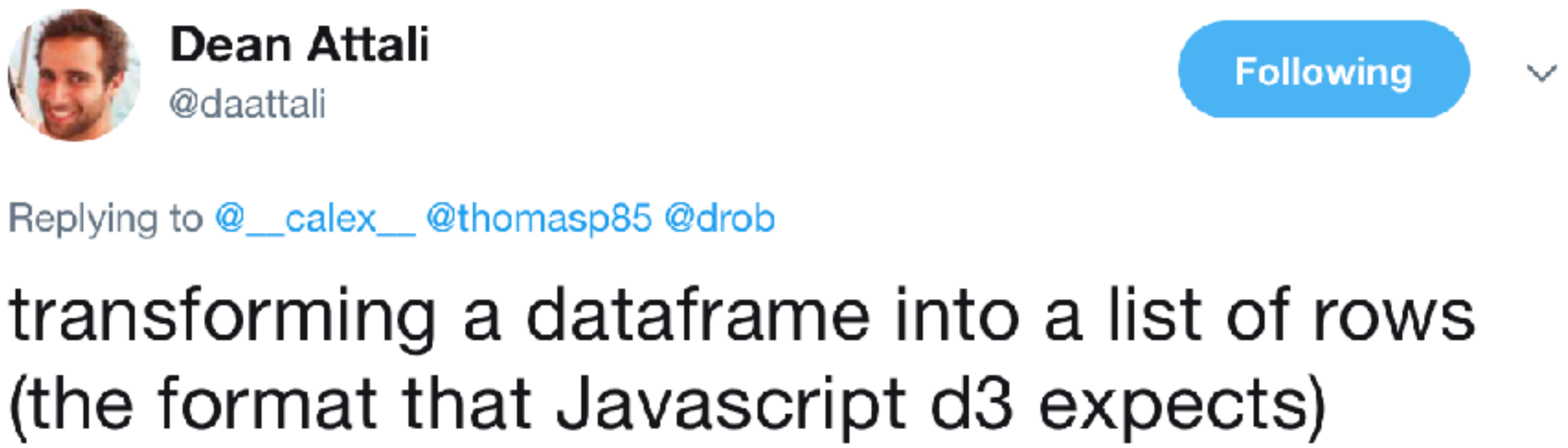
r4ds.had.co.nz

download materials: **rstd.io/row-work**


Dean Attali
@daattali

what's a safe way to iterate over rows of a dataframe? re: twitter.com/winston_chang/… #rstats

https://twitter.com/daattali/status/761058049859518464


Dean Attali
@daattali

Replying to @__calex__ @thomasp85 @drob

transforming a dataframe into a list of rows (the format that Javascript d3 expects)

https://twitter.com/daattali/status/761233607822221312

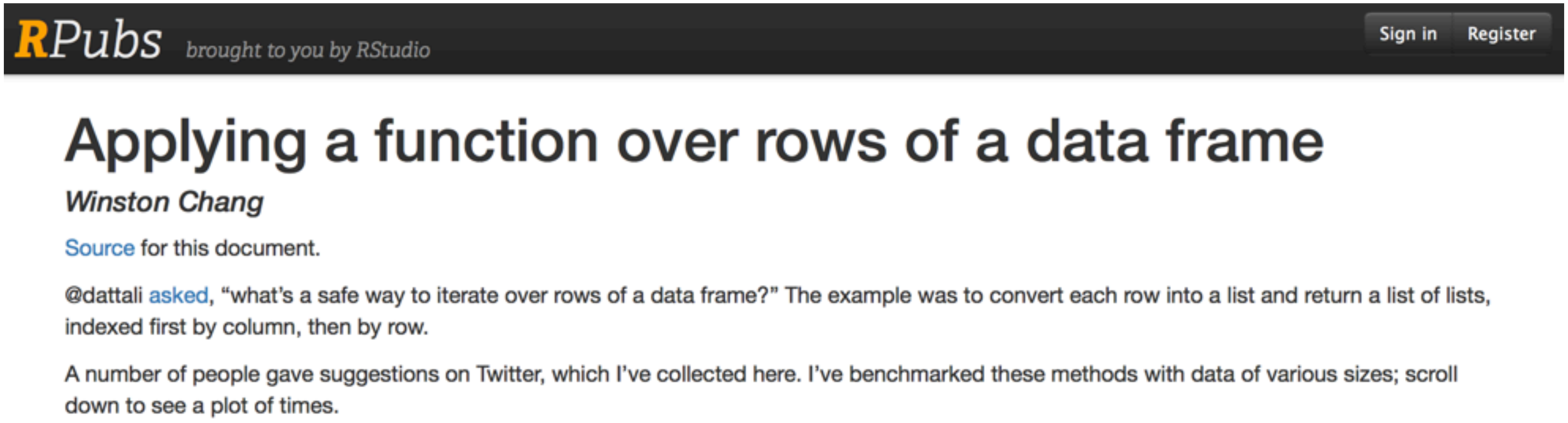
download materials: **rstd.io/row-work**

How to do this?

```
> i_have
# A tibble: 2 x 2
      x y
  <dbl> <chr>
1    1. one
2    2. two
```

```
> str(i_want)
List of 2
 $ :List of 2
  ..$ x: num 1
  ..$ y: chr "one"
 $ :List of 2
  ..$ x: num 2
  ..$ y: chr "two"
```

Winston compiled, I updated.



Applying a function over rows of a data frame

Winston Chang

Source for this document.

@dattali asked, "what's a safe way to iterate over rows of a data frame?" The example was to convert each row into a list and return a list of lists, indexed first by column, then by row.

A number of people gave suggestions on Twitter, which I've collected here. I've benchmarked these methods with data of various sizes; scroll down to see a plot of times.

https://rpubs.com/wch/200398

for loop

```
df <- SOME DATA FRAME
out <- vector(mode = "list", length = nrow(df))
for (i in seq_along(out)) {
  out[[i]] <- as.list(df[i, , drop = FALSE])
}
out
```

split by row then lapply

```
df <- SOME DATA FRAME
df <- split(df, seq_len(nrow(df)))
lapply(df, function(row) as.list(row))
```

lapply over row numbers

```
df <- SOME DATA FRAME
lapply(
  seq_len(nrow(df)),
  function(i) as.list(df[i, , drop = FALSE])
)
```

purrr::pmap()

```
df <- SOME DATA FRAME
pmap(df, list)
```

purrr::transpose()*

```
df <- SOME DATA FRAME
transpose(df)
```

* Happens to be exactly what's needed in this specific example.

Why so many ways to do **THING** for each row?

Because there is no way.

Why so many ways to do **THING** for each row?

Columns are very special in R.
This is fantastic for data analysis.
Tradeoff: row-oriented work is harder.

How to choose?

Speed and ease of:

- **Writing** the code
- **Reading** the code
- **Executing** the code

Of course someone has to write **loops**

It doesn't have to be you

Pro tip #1

Use vectorized functions.

Let other people write loop-y code for you.

paste() example

ex03_row-wise-iteration-are-you-sure.R

Pro tip #2

Use purrr::map()* and friends.

Let other people write loop-y code for you.

* Like base::lapply(), but anchors a large, coherent family of map functions.

```
purrr::
map(.x, .f, ...)
map(.x, .f, ...)
```

for every element of **.x**

apply **.f**

.x = minis

```
map(minis, antennate)
```

map(.x, .f, ...)

```
.x <- SOME VECTOR OR LIST
out <- vector(mode = "list", length = length(.x))
for (i in seq_along(out)) {
  out[[i]] <- .f(.x[[i]])
}
out
```

```
map(.x, .f, ...)
```

purrr::map() implements a for loop!

But with less code clutter.

purrr::map() example
ex04_map-example.R

No, I really do need to do **THING** for each row.

How to do this?

```
> i_have
# A tibble: 2 x 2
      x y
  <dbl> <chr>
1    1. one
2    2. two
```

```
> str(i_want)
List of 2
 $ :List of 2
  ..$ x: num 1
  ..$ y: chr "one"
 $ :List of 2
  ..$ x: num 2
  ..$ y: chr "two"
```

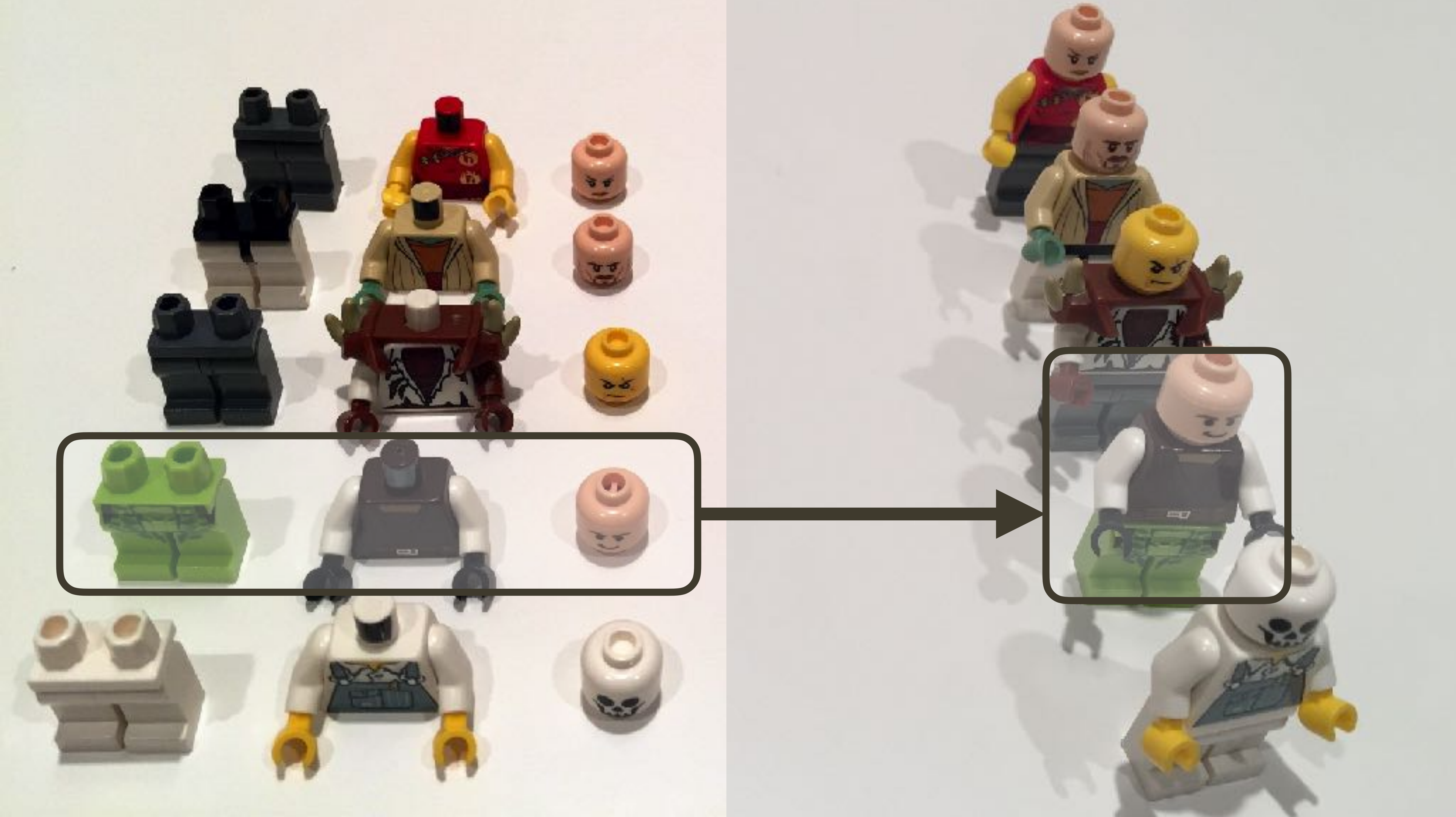
```
pmap(.l, .f, ...)
```

for every tuple in `.l`

apply `.f`

`pmap(.l, embody)`

pmap(.l, embody)



`pmap(.l, .f, ...)`

```
.l <- LIST OF LENGTH-N VECTORS
out <- vector(mode = "list", length = N)
for (i in seq_along(out)) {
  out[[i]] <- .f(.l[[1]][[i]], .l[[2]][[i]], ...)
}
out
```

pmap(.l, .f, ...)

A data frame works!

```
.l <- LIST OF LENGTH-N VECTORS
out <- vector(mode = "list", length = N)
for (i in seq_along(out)) {
  out[[i]] <- .f(.l[[1]][[i]], .l[[2]][[i]], ...)
}
out
```

row i

download materials: **rstd.io/row-work**

pmap(.l, .f, ...)

```
.l <- LIST OF LENGTH-N VECTORS
out <- vector(mode = "list", length = N)
for (i in seq_along(out)) {
  out[[i]] <- .f(.l[[1]][[i]], .l[[2]][[i]], ...)
}
out
```

pmap() <u>is</u> a for loop!

it applies .f to each row

purrr::pmap() example
ex06_runif-via-pmap.R

How to choose?

Speed and ease of:

- **Writing** the code
- **Reading** the code
- **Executing** the code

map()

map_lgl(), map_int(), map_dbl(), map_chr()

map_if(), map_at()

map_dfr(), map_dfc()

map2()

map2_lgl(), map2_int(), map2_dbl(), map2_chr()

map2_dfr(), map2_dfc()

pmap()

pmap_lgl(), pmap_int(), pmap_dbl(), pmap_chr()

pmap_dfr(), pmap_dfc()

imap()

imap_lgl(), imap_chr(), imap_int(), imap_dbl()

imap_dfr(), imap_dfc()

map()

map_lgl(), map_int(), map_dbl(), map_chr()

map

map

map

map r()

map

pma

pma hr()

pma

imap

imap_lgl(), imap_chr(), imap_int(), imap_dbl()

imap_dfr(), imap_dfc()

purrr's map functions have

a common interface

learn it once,

use it everywhere

for loop

```
df <- SOME DATA FRAME
out <- vector(mode = "list", length = nrow(df))
for (i in seq_along(out)) {
  out[[i]] <- as.list(df[i, , drop = FALSE])
}
out
```

split by row then lapply

```
df <- SOME DATA FRAME
df <- split(df, seq_len(nrow(df)))
lapply(df, function(row) as.list(row))
```

lapply over row numbers

```
df <- SOME DATA FRAME
lapply(
  seq_len(nrow(df)),
  function(i) as.list(df[i, , drop = FALSE])
)
```

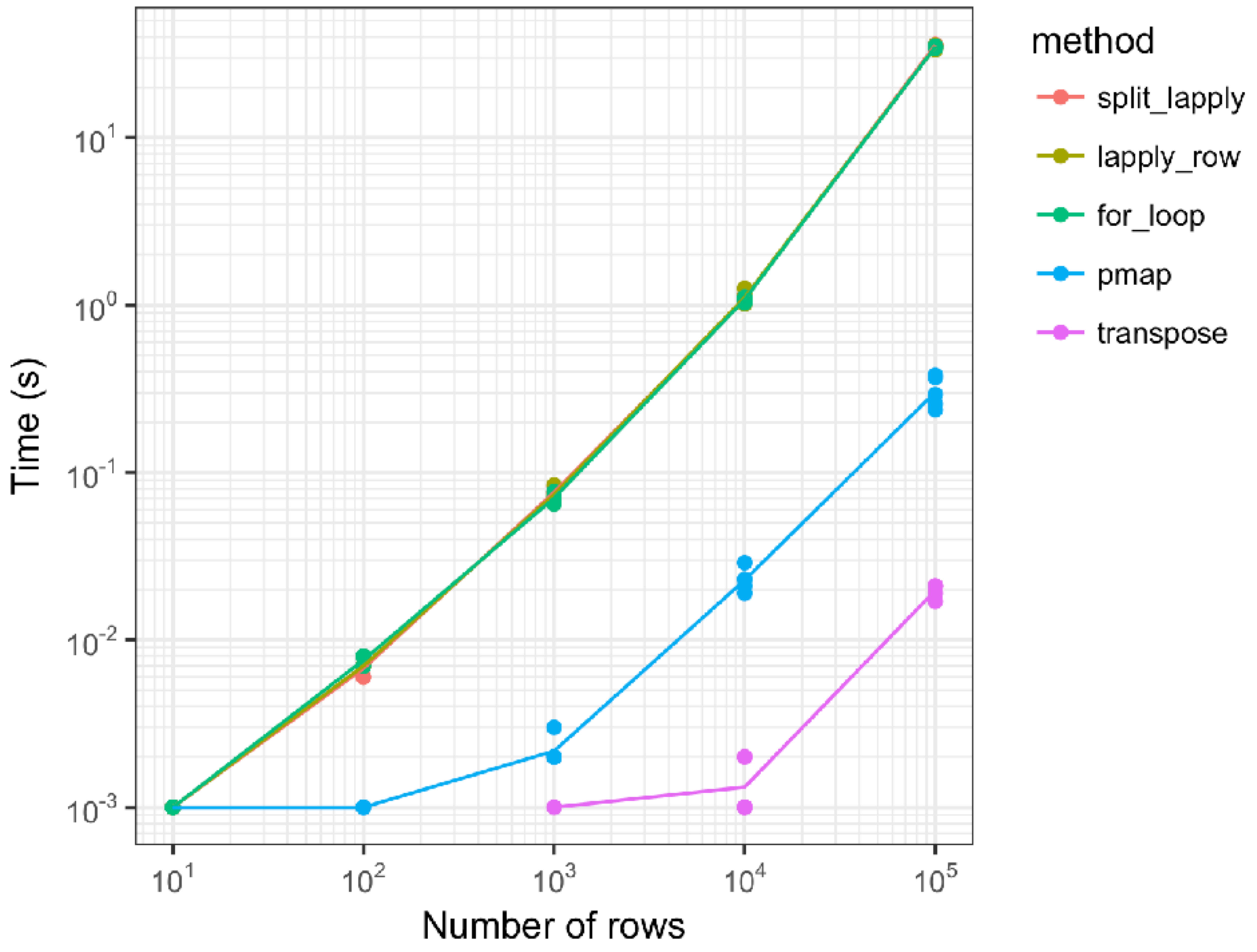
purrr::pmap()

```
df <- SOME DATA FRAME
pmap(df, list)
```

purrr::transpose()

```
df <- SOME DATA FRAME
transpose(df)
```

download materials: **rstd.io/row-work**


method
split_lapply
lapply_row
for_loop
pmap
transpose
Time (s)
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
Number of rows
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}

code for that study:

iterate-over-rows.R

for each row of df

```r
purrr::pmap(df, .f)
```

do this

What if I need to work on **groups** of rows?

Pro tip #3

Use dplyr::group_by() + summarize().

Let other people write loop-y code for you.

group_by() + summarize() example

ex07_group-by-summarise.R

No, I really must work on **groups** of rows.

Use **nesting**

to restate as

"do **THING** for each row"

Use **nesting**

to restate as

"do **THING** for each row"

DONE*

* See everything up 'til now in this talk.

dplyr::group_by() + tidyr::nest()
ex08_nesting-is-good.R

Tips for row-oriented workflows

embrace the **data frame**

esp. the **tibble** = tidyverse data frame

embrace **lists**

embrace lists as variables in a tibble

"**list-columns**", may come from nesting

embrace **purrr::map()** & friends

download materials: **rstd.io/row-work**