Futures

PARALLEL PROGRAMMING IN R

Nabeel Imam
Data Scientist

datacamp

The demanding boss

Immediate execution

```
report <- "<important report text>"
task <- {
  print("Hi, here is the report.") # Email text
  report
}
```

```
"Hi, please find attached the report."
```

```
task
```

```
"<important report text>"
```

Delayed execution

```
library(future)


task_future <- future({
  print("Hi, here is the report.")
  report
})
```

Delayed execution

```
print(task_future)
```

```
Expression: {
print("Hi, here is the report.")
report
}
Environment: R_GlobalEnv
Resolved: TRUE
Value: 136 bytes of class "character"
...
```

- The code or "how-to" of the task
- The environment or the context of the task
- Is it done? Seems so!
- Description of the outcome

Delayed execution

```
value(task_future)
```

```
[1] "Hi, here is the report."
[1] "<important report text>"
```

A team of workers

[1] Designed by Freepik

A team of workers

```
task1 <- future({
  print("Hi, here is the report.")
  report
})

task2 <- future({
  run_analysis()
  print("Analysis done")
})

task3 <- future({
  book_meeting_rooms()
  print("Meeting rooms booked!")
})
```

```
value(task1) # 10:00 AM Office Administrator
```

```
[1] "Hi, here is the report."
[1] "<important report text>"
```

```
value(task3) # 11:00 AM Office Administrator
```

```
"Meeting rooms booked!"
```

```
value(task2) # 03:00 PM Analyst
```

```
"Analysis done"
```

Moving average with futures

```
print(amazon_prices)
```

```
 [1] 312.58 307.01 302.24
 [4] 297.50 300.32 301.48
 [7] 297.56 297.48 291.93
[10] 294.00 286.28 292.59
...
```

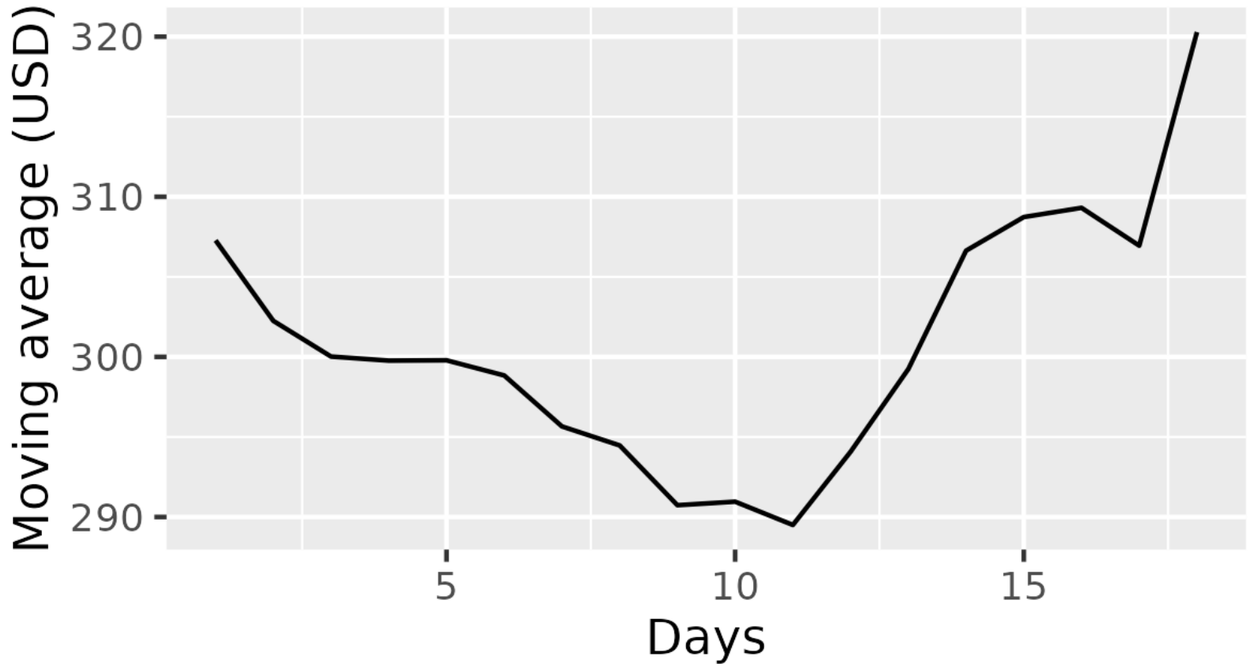
```
ma <- future({moving_average(amazon_prices)})
```

```

```

```
moving_average <- function (prices) {

  N <- length(prices) - 2
  moving_avgs <- rep(NA, N)

  for (i in 1:N) {
    moving_avgs[i] <- mean(prices[i: (i + 2)])
  }

  return(moving_avgs)
}
```

Moving average with futures

```
ggplot() +
  geom_line(aes(x = 1:(length(amazon_prices) - 2),
                y = value(ma))) + # Value of futures is required here
  labs(y = "Moving price average (USD)", x = "Days")
```



Parallel futures

```
print(amazon_list)
```

```
$`2015-01`
 [1] 312.58 307.01 302.24
 [4] 297.50 300.32 301.48
 ...
$`2015-02`
 [1] 350.05 360.29 358.38
 [4] 366.00 374.87 371.00
 ...
$`2015-03`
 [1] 380.85 383.95 385.71
 [4] 385.61 385.52 378.40
 ...
...
```

```
moving_average <- function (prices) {

  N <- length(prices) - 2
  moving_avgs <- rep(NA, N)

  for (i in 1:N) {
    moving_avgs[i] <- mean(prices[i: (i + 2)])
  }

  return(moving_avgs)
}
```

Parallel futures

```
plan(multisession, workers = 4)

ma <- lapply(amazon_list,
             function (x) {
               future(moving_average(x))
               })
```

- Plan to employ four worker cores in a multisession
- Create one future task for every element of `amazon_list`
  - Futures created under multisession plan will be executed in parallel

Parallel futures

```
value(ma)


plan(sequential)
```

```
$`2015-01`
 [1] 307.2767 302.2500 300.0200 299.7667 299.7867 298.8400 295.6567 ...
$`2015-02`
 [1] 356.2400 361.5567 366.4167 370.6233 372.3533 371.1400 372.5067 ...
$`2015-03`
 [1] 383.5033 385.0900 385.6133 383.1767 380.4567 375.4867 372.2933 ...
...
```

Why future?

Sequential

```
plan(sequential)

ma <- lapply(amazon_list,
             function (x) {
               future(moving_average(x))
               })

value(ls)
```

Parallel

```
plan(multisession, workers = 4)

ls <- lapply(amazon_list,
             function (x) {
               future(moving_average(x))
               })

value(ls)

plan(sequential)
```

Let's practice!

PARALLEL PROGRAMMING IN R

Mapping the future

PARALLEL PROGRAMMING IN R

Nabeel Imam
Data Scientist

datacamp

We know our futures

```
report <- "<important report text>"


task_future <- future({
  print("Hi, please find attached the report.")
  report
})
```

Mapping with purrr

```
library(purrr)

numbers <- 1:1000000
map(numbers, sqrt)
```

```
[[1]]
[1] 1

[[2]]
[1] 1.414214

[[3]]
[1] 1.732051

[[4]]
[1] 2
...
```

Incarnations of map

```
map_dbl(numbers, sqrt)
```

```
 [1]  1.000000  1.414214  1.732051
 [4]  2.000000  2.236068  2.449490
 [7]  2.645751  2.828427  3.000000
[10]  3.162278  3.316625  3.464102
[13]  3.605551  3.741657  3.872983
  ...
```

- **map + "_dbl" suffix**
  - For double or decimal number output
- Results returned as a vector

Incarnations of map

```
map_chr(numbers, sqrt)
```

```
 [1] "1.000000"  "1.414214"  "1.732051"
 [4] "2.000000"  "2.236068"  "2.449490"
 [7] "2.645751"  "2.828427"  "3.000000"
[10] "3.162278"  "3.316625"  "3.464102"
[13] "3.605551"  "3.741657"  "3.872983"
  ...
```

- **map + "_chr" suffix**
  - For character or string output

- Vector of character strings

Incarnations of map

Function

- `map()`
- `map_chr()`
- `map_dbl()`
- `map_int()`
- `map_lgl()`

Output format

- List of vectors, data frames, etc.
- Character or string (one value)
- Double or decimal numbers (one value)
- Integers or whole numbers (one value)
- Logical or boolean (one value)

Type specification

```
microbenchmark(
  "map" = map(numbers, sqrt),
  "map_dbl" = map_dbl(numbers, sqrt),
  "map_chr" = map_chr(numbers, sqrt),
  times = 10
)
```

```
Unit: milliseconds
     expr     min    mean     max neval
1     map  794.20 1254.46 1904.02    10
2 map_dbl  889.33 1067.41 1496.56    10
3 map_chr 1735.96 1934.97 2269.59    10
```

future + purrr = furrr

purrr

Sequential

```
library(purrr)
map_dbl(1:1000000, sqrt)
```

furrr

Sequential

```
library(furrr)
future_map_dbl(1:1000000, sqrt)
```

furrr in parallel

```
n_cores <- detectCore() - 2

plan(multisession, workers = n_cores)

future_map_dbl(1:10000000, sqrt) # Future enabled map_dbl()

plan(sequential)
```

```
   [1]  1.000000  1.414214  1.732051  2.000000  2.236068  2.449490  2.645751
   [8]  2.828427  3.000000  3.162278  3.316625  3.464102  3.605551  3.741657
  [15]  3.872983  4.000000  4.123106  4.242641  4.358899  4.472136  4.582576
```

future_map and family

purrr function

- `map()`
- `map_chr()`
- `map_dbl()`
- `map_int()`
- `map_lgl()`

future-enabled furrr function

- `future_map()`
- `future_map_chr()`
- `future_map_dbl()`
- `future_map_int()`
- `future_map_lgl()`

The advantage of furrr

- Given a series of inputs, `input_list` and a function to map, `calculate()`:

```
future_list <- lapply(input_list, function (x) future(calculate(x)))
result_list <- value(future_list)

# Further processing of result_list to make a numeric vector
```

```
library(furrr)

# Using future_map_dbl() variant to get a numeric vector
result_list <- future_map_dbl(input_list, calculate)
```

Let's practice!

PARALLEL PROGRAMMING IN R

Advanced operations with furrr

PARALLEL PROGRAMMING IN R

Nabeel Imam
Data Scientist

datacamp

A bootstrap example

```
boot_dist <- function (variable, B) {

  est <- rep(0, B)

  for (i in 1:B) {
    b <- sample(variable, replace = T)
    est[i] <- mean(b)
  }

  return(est)
}

n_samples <- 10000
```

```
print(age_list)
```

```
$AK
  [1] 43 34 26 19 22 36 17 24 33 25 18 44 21
 [14] 32 31 27 27 21 27 30 20 37 31 21 29 32
  ...
$AL
   [1] 33 25 24 22 27 21 34 21 45 36 31 29 25
  [14] 23 20 27 28 30 33 16 29 27 24 26 20 15
  ...
$AR
  [1] 28 27 20 27 26 21 20 30 22 17 22 30 15
 [14] 27 31 30 30 19 25 25 22 26 23 18 26 20
  ...
...
```

The global problem

```
plan(multisession, workers = 4)


future_map(age_list, boot_dist)
```

```
Error in ...furrr_fn(...) : argument "B" is missing, with no default
```

- `boot_dist` requires a value for the argument `B`
- `multisession` plans do not share workspace

furrr_options

```
# Create configuration
config <- furrr_options(globals = "n_samples")

# Plan multisession of four
plan(multisession, workers = 4)

# Pass inputs and functions to future_map,
future_map(age_list, boot_dist,
# specify value for second argument of boot_dist,
           B = n_samples,
# and supply configuration to .options
           .options = config)

plan(sequential)
```

```
$AK
   [1] 27.22053 27.29658 26.03422 26.85551
   [5] 27.13688 26.77947 27.23574 26.67300
   ...
$AL
   [1] 25.16612 25.25577 25.05735 25.18062
   [5] 24.89914 24.82729 25.29071 25.00725
   ...
$AR
   [1] 24.50978 24.74450 24.38631 24.48533
   [5] 24.91565 24.50000 24.48533 24.82152
   ...
...
```

Filtering the births dataset

```
print(ls_births)
```

```
$AK
  state month plurality weight_gain_pounds mother_age
     AK     6         1                 30         15
     AK     3         1                 38         15
    ...
$AL
  state month plurality weight_gain_pounds mother_age
     AL     1         1                 22         15
     AL     6         1                 53         16
    ...
...
```

future_map with packages

```
filter_df <- function (df, min_value) {
  df %>% dplyr::filter(mother_age >= min_value)
}
cutoff <- 20

config <- furrr_options(globals = "cutoff",  # Global variables
                        packages = "dplyr")  # Packages

plan(multisession, workers = 4)

ls_filtered <- future_map(ls_births, filter_df,
                          min_value = cutoff,
                          .options = config)

plan(sequential)
```

future_map with packages

```
print(ls_filtered)
```

```
$AK
  state month plurality weight_gain_pounds mother_age
     AK     4         1                99         20
     AK     6         1                22         21
    ...
$AL
  state month plurality weight_gain_pounds mother_age
     AL     8         1                19         20
     AL     4         1                50         22
    ...
...
```

Multiple arguments to loop over

```
print(ls_weights)
```

```
$AK
  [1] 99 22  2 60 20 18 43 10 37 29 30
 [12] 15 26 50 11 22 17 20 40 30 99 24
 ...
$AL
   [1] 19 50 14 32 40 40 41 20 59 23 99
  [12] 36 31 50 34 39 15 73 99 99 38 40
  ...
...
```

```
print(ls_plur)
```

```
$AK
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [18] 1 1 1 1 1 1 1 1 1 2 1 1 1 3 1 1 1
 ...
$AL
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1
 [18] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  ...
...
```

Multiple arguments to loop over

future_pmap()

```
plan(multisession, workers = 4)

calculate <- function (weights, babies) {
  weights/babies
}

future_pmap(list(ls_weights, ls_plur), # Combine arguments into one list
            calculate)                 # Function to apply

plan(sequential)
```

Multiple arguments to loop over

```
$AK
  [1] 99.0 22.0  2.0 60.0 20.0 18.0 43.0 10.0 37.0 29.0 30.0
 [12] 15.0 26.0 50.0 11.0 22.0 17.0 20.0 40.0 30.0 99.0 24.0
 ...
$AL
   [1] 19.00 50.00 14.00 32.00 40.00 40.00 41.00 20.00 59.00
  [10] 23.00 99.00 36.00 31.00 25.00 34.00 39.00 15.00 73.00
  ...
...
```

A note about argument order

future_map

1. Single input to loop over
   - One list or vector
2. Function to be applied
3. Other arguments

future_pmap

1. Multiple inputs to loop over
   - Lists or vectors combined into one list
2. Function to be applied
3. Other arguments

Let's practice!

PARALLEL PROGRAMMING IN R

Do more with future_map()

PARALLEL PROGRAMMING IN R

Nabeel Imam
Data Scientist

datacamp

The data and the story

```
head(data)
```

```
  state  year month plurality
1 AK     1995     1          1
2 AK     1995     1          1
3 AK     1995     1          1
4 AK     1995     1          1
5 AK     1995     1          1
6 AK     1995     1          1
```

Generating a new character column

```
twins <- function (x) {
    ifelse(x == 2,"Twins", "Not twins")
}
```

Labeling the twins

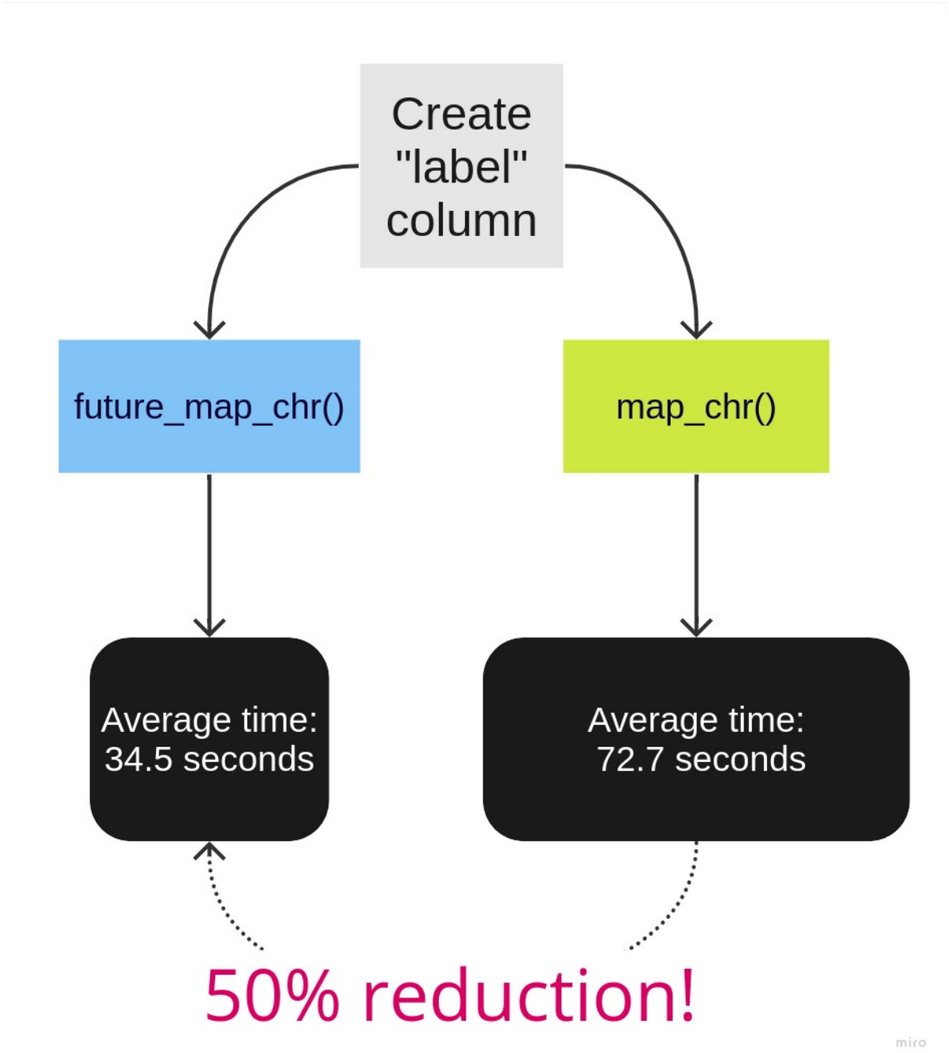
```
plan(multisession, workers = 4)

data %>%
 mutate(label = future_map_chr(plurality, twins))

plan(sequential)
```

```
   state  year month plurality label
 1 AK     1995     1          1 Not twins
 2 AK     1995     1          1 Not twins
 3 AK     1995     1          1 Not twins
 4 AK     1995     1          1 Not twins
 5 AK     1995     1          1 Not twins
 6 AK     1995     1          1 Not twins
 7 AK     1995     1          1 Not twins
 8 AK     1995     1          1 Not twins
 9 AK     1995     1          1 Not twins
10 AK     1995     1          1 Not twins
...
```

Labeling the twins

```
microbenchmark(
  "map_chr" = {
    data %>%
 mutate(label = map_chr(plurality, twins))
  },
  "future_map_chr" = {
    data %>%
 mutate(label = future_map_chr(plurality, twins))
  }
)
```

```
Unit: seconds
            expr      mean    median neval
1        map_chr 72.69067 73.50705    10
2 future_map_chr 34.52351 34.56357    10
```

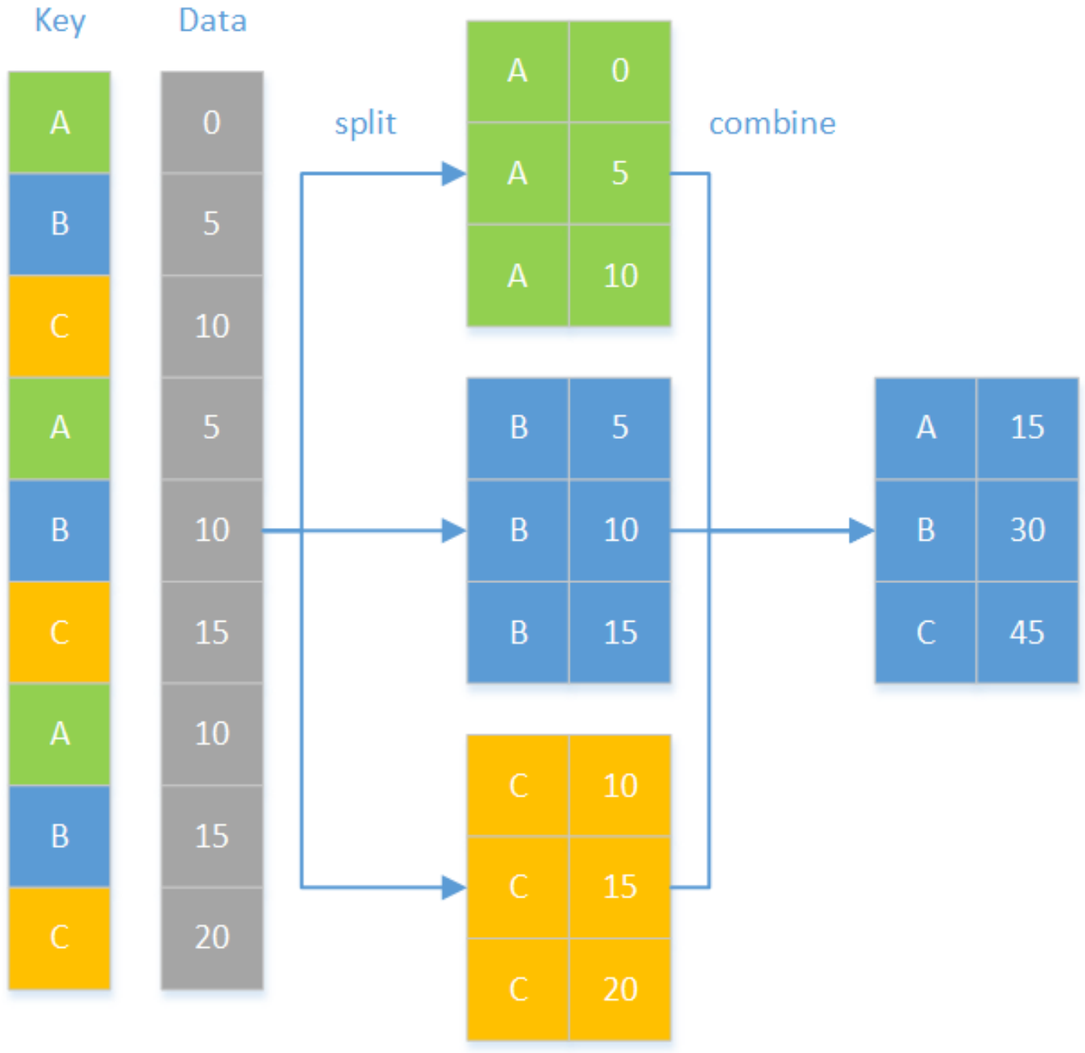


Operations by groups

The proportion of twins

```
birth_prop <- function (df) {

  N <- sum(df$plurality == 2)  # Total number of twin births
  prop <- N/nrow(df)           # Proportion out of all births
  names(prop) <- "proportion"  # Provide a name for value

  return(prop)
}
```

Operations by groups



Operations by groups

```
plan(multisession, workers = 6)

data %>%
  # Split by year, pipe to future_map_dfr()
  split(data$state) %>%
  # Supply only function here
  future_map_dfr(birth_prop,
  # Name of grouping column to .id argument
                 .id = "state")
plan(sequential)
```

Results row-binded into one data frame

```
   year  proportion
 1 AK         0.0114
 2 AL         0.0264
 3 AR         0.0196
 4 AZ         0.0218
 5 CA         0.0197
 6 CO         0.0217
 7 CT         0.0225
 8 DC         0.0283
 9 DE         0.0268
10 FL         0.0212
...
```

Using global variables

```
# Second argument to specify plurality value
birth_prop <- function (df, plur_value) {

  N <- sum(df$plurality == plur_value) # Total number of births of given plurality
  prop <- N/nrow(df)
  names(prop) <- "proportion"

  return(prop)
}

new_plur <- 3 # A global single-valued variable
```

Using global variables

```
config <- furrr_options(globals = "new_plur")

plan(multisession, workers = 4)

data %>%
  split(data$state) %>%
  future_map_dfr(birth_prop,
                 plur_value = new_plur,
                 .options = config,
                 .id = "state")

plan(sequential)
```

```
   state proportion
 1 AK      0
 2 AL      0.000659
 3 AR      0
 4 AZ      0.000605
 5 CA      0.000673
 6 CO      0.000776
 7 CT      0.000867
 8 DC      0.00189
 9 DE      0
10 FL      0.00106
...
```

Column-bind to a data frame

```
data %>%
  split(data$state) %>%
  future_map_dfc(birth_prop, # The _dfc variant
                 plur_value = new_plur,
                 .options = config)
```

```
     AK        AL    AR        AZ        CA        CO        CT       DC
1     0 0.000659     0 0.000605 0.000673 0.000776 0.000867 0.00189 ...
```

Let's practice!

PARALLEL PROGRAMMING IN R