Boosting R Code Performance

Kun Ren

About Me

- Hedge fund quant researcher in Shanghai, China
- Low frequency and high frequency trading of equity and futures
- Financial data analysis, predictive statistical modeling, etc.
- Creator of several R packages: formattable, rlist, and pipeR
- Author of *Learning R Programming*

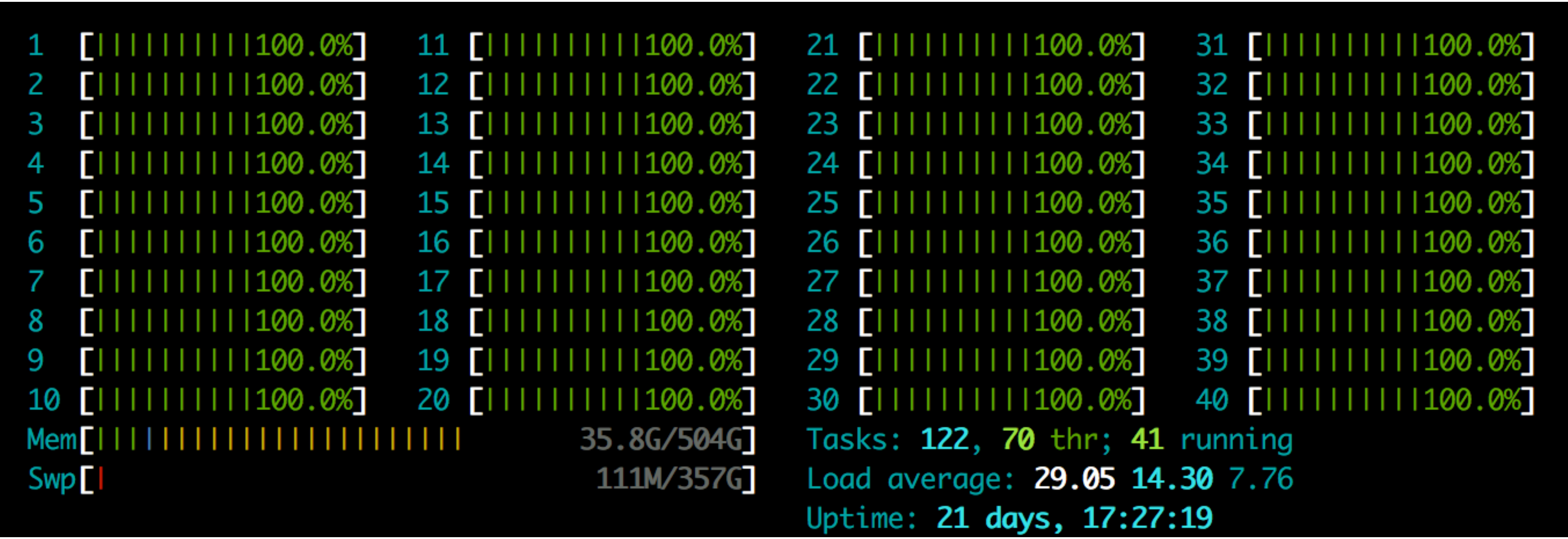
Performance of R

- R is designed for easiness of use rather than performance
- R is interpreted, dynamic scripting language
- R can be much slower compared with C++ but fast enough for most of its purposes

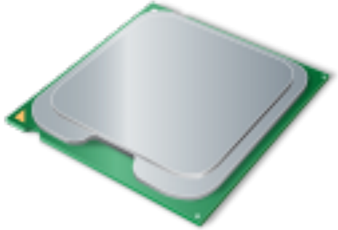
When performance is important?

- Scenario 1: When data is of billions of rows
- Scenario 2: When an operation needs to be repeated many times
- Scenario 3: When the computing time is limited

Understanding performance in general



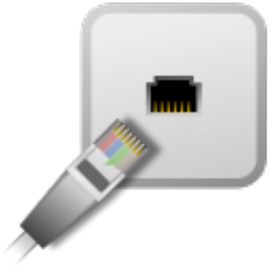
Limited resources



CPU-bound

Memory-bound

I/O-bound

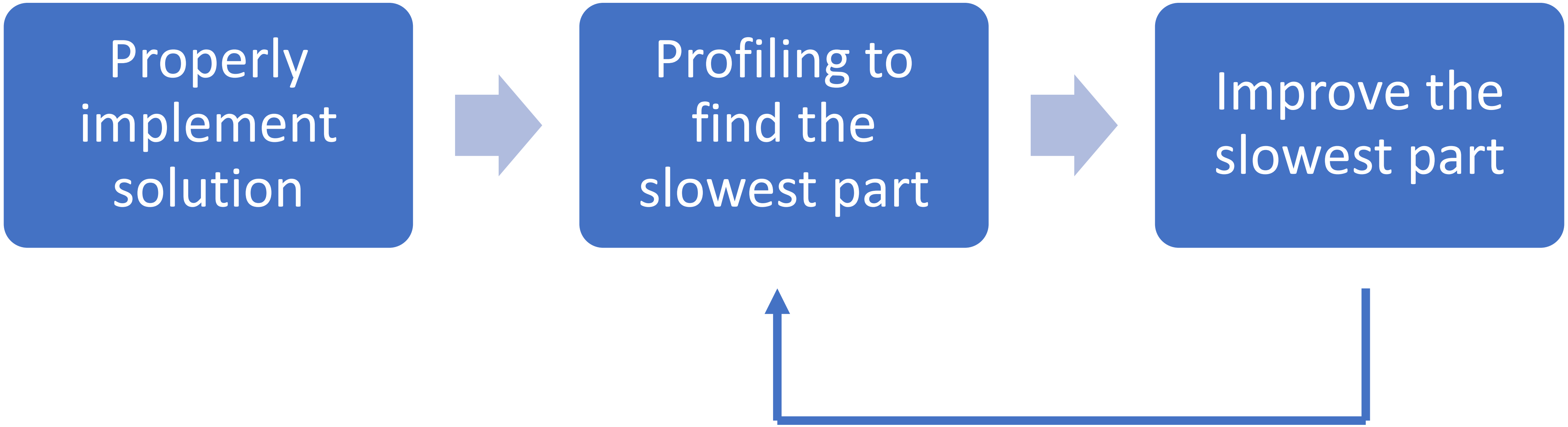


Network-bound

Why can R be slow?

- R as an interpreted, dynamic scripting language
- Overhead of function calling
- Inefficient use of data structures
- Frequent copy of data: growing vectors, converting between data frame and matrix, etc

A road map to speed up R code



A rolling regression problem

date	ret	mret
1	-0.560475647	-0.4941738794
2	-0.230177489	1.1275934662
3	1.558708314	-1.1469495486
4	0.070508391	1.4810185971
5	0.129287735	0.9161912125
6	1.715064987	0.3351310171
7	0.460916206	0.5746753285
8	-1.265061235	0.2036196619
9	-0.686852852	-0.4470411920
10	-0.445661970	-0.3435259276
11	1.224081797	-0.6038357965
12	0.359813827	1.2386725185
13	0.400771451	0.5994920533
14	0.110682716	-0.1087337576
15	-0.555841135	-1.1388569648

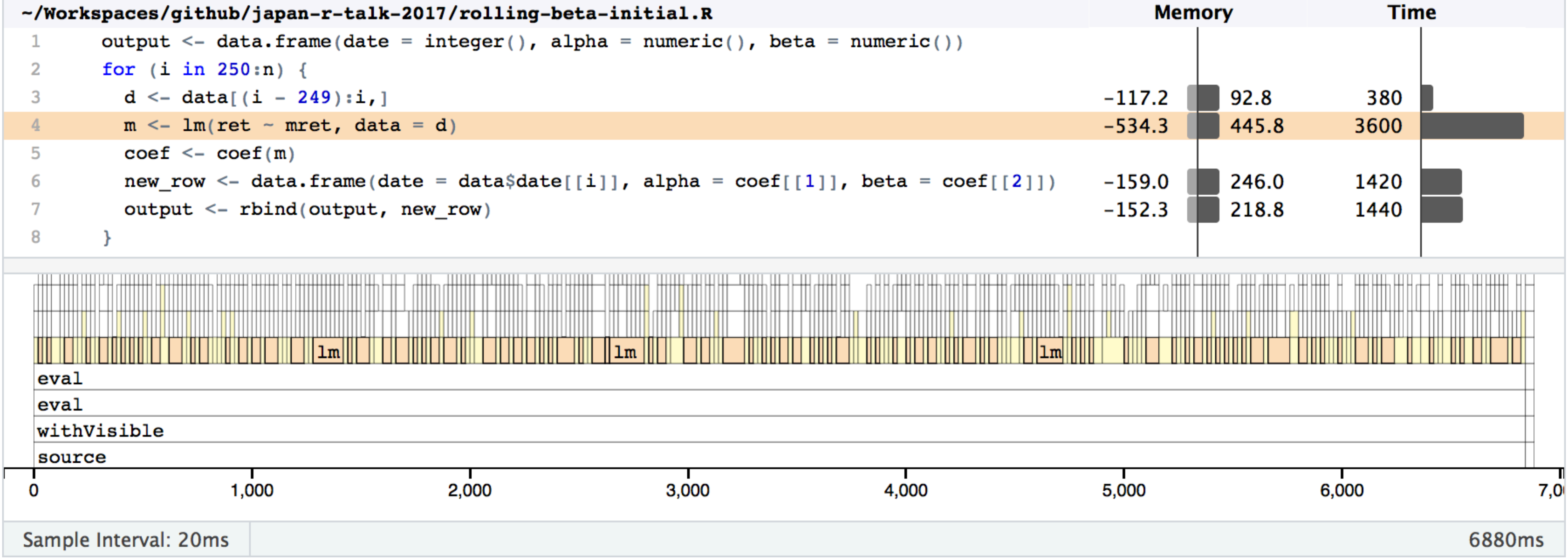
```r
set.seed(123)
n <- 250 * 20
data <- data.frame(date = 1:n, ret = rnorm(n), mret = rnorm(n))
```

- For each date, estimate the linear coefficients for the recent 250 days
- $r = \alpha + \beta r_m + \epsilon$
- `lm(ret ~ mret)`

An initial solution

```r
output <- data.frame(date = integer(), alpha = numeric(), beta = numeric())
for (i in 250:n) {
  d <- data[(i - 249):i,]
  m <- lm(ret ~ mret, data = d)
  coef <- coef(m)
  new_row <- data.frame(date = data$date[[i]], alpha = coef[[1]], beta = coef[[2]])
  output <- rbind(output, new_row)
}
```

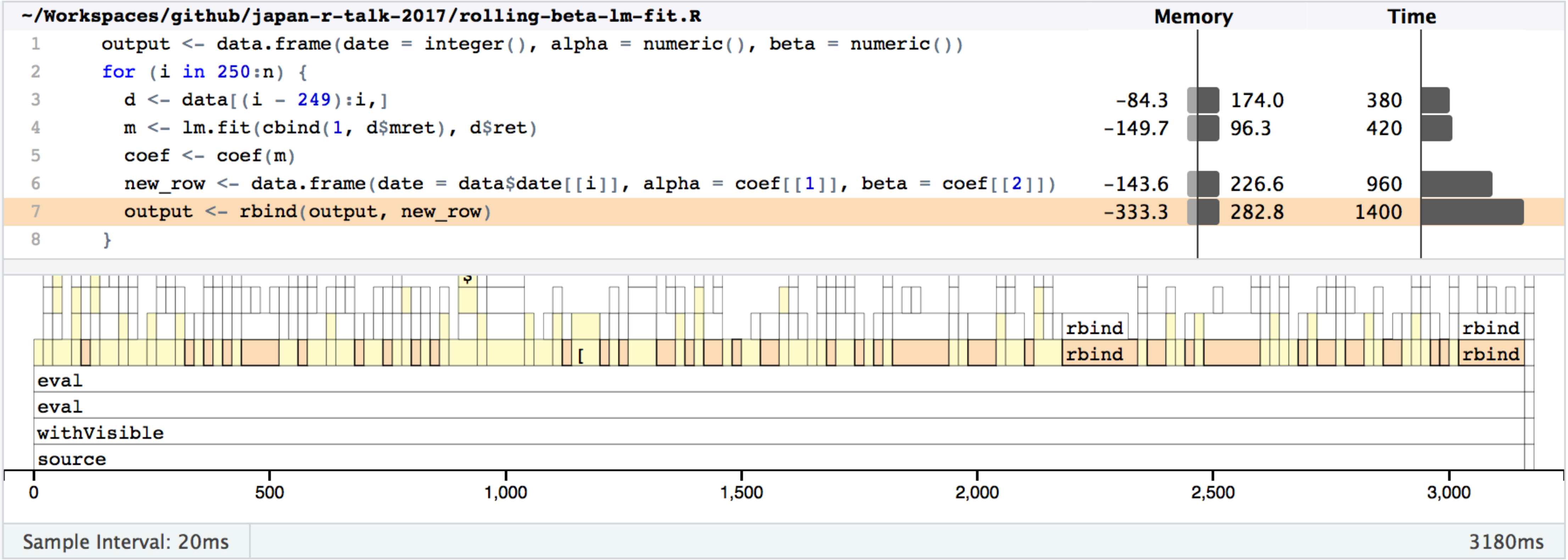
Profiling initial solution (6880ms)



Replace `lm` with `lm.fit`

```r
output <- data.frame(date = integer(), alpha = numeric(), beta = numeric())
for (i in 250:n) {
  d <- data[(i - 249):i,]
  m <- lm.fit(cbind(1, d$mret), d$ret)
  coef <- coef(m)
  new_row <- data.frame(date = data$date[[i]], alpha = coef[[1]], beta = coef[[2]])
  output <- rbind(output, new_row)
}
```

Replace `lm` with `lm.fit` (3180ms)



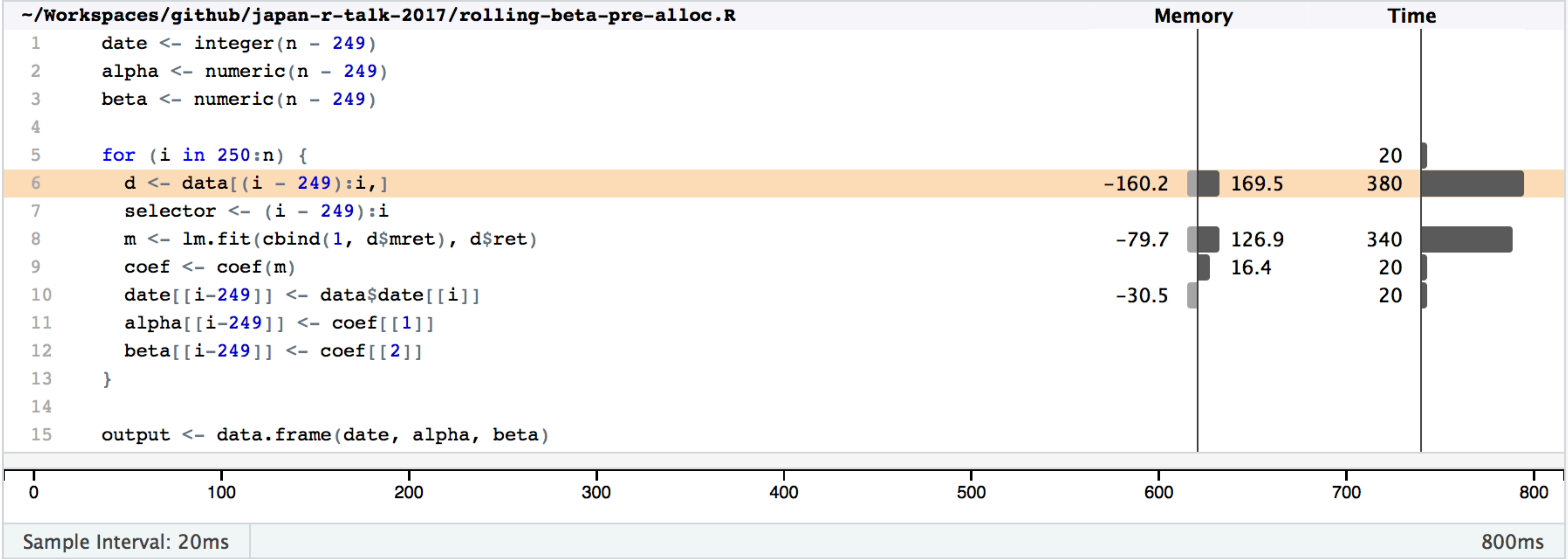
Pre-allocate vectors

```r
date <- integer(n - 249)
alpha <- numeric(n - 249)
beta <- numeric(n - 249)

for (i in 250:n) {
  d <- data[(i - 249):i,]
  selector <- (i - 249):i
  m <- lm.fit(cbind(1, d$mret), d$ret)
  coef <- coef(m)
  date[[i-249]] <- data$date[[i]]
  alpha[[i-249]] <- coef[[1]]
  beta[[i-249]] <- coef[[2]]
}

output <- data.frame(date, alpha, beta)
```

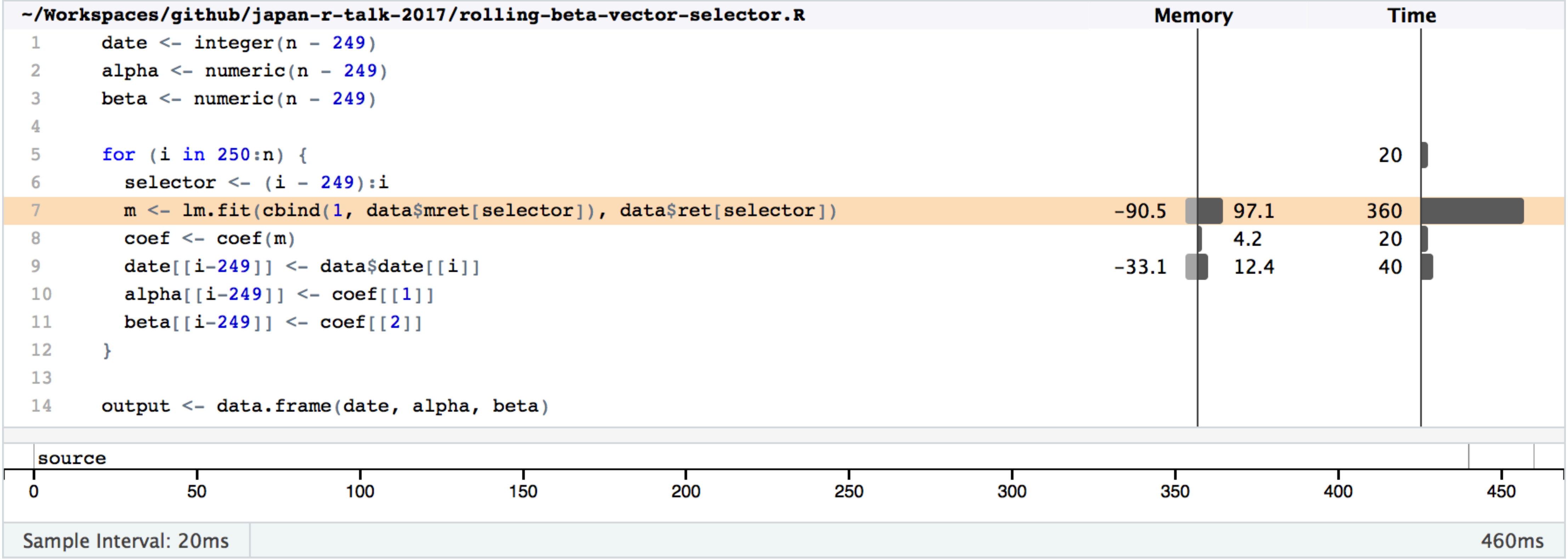
Pre-allocation of vectors

Avoid copying vectors

Pre-allocate vectors (800ms)



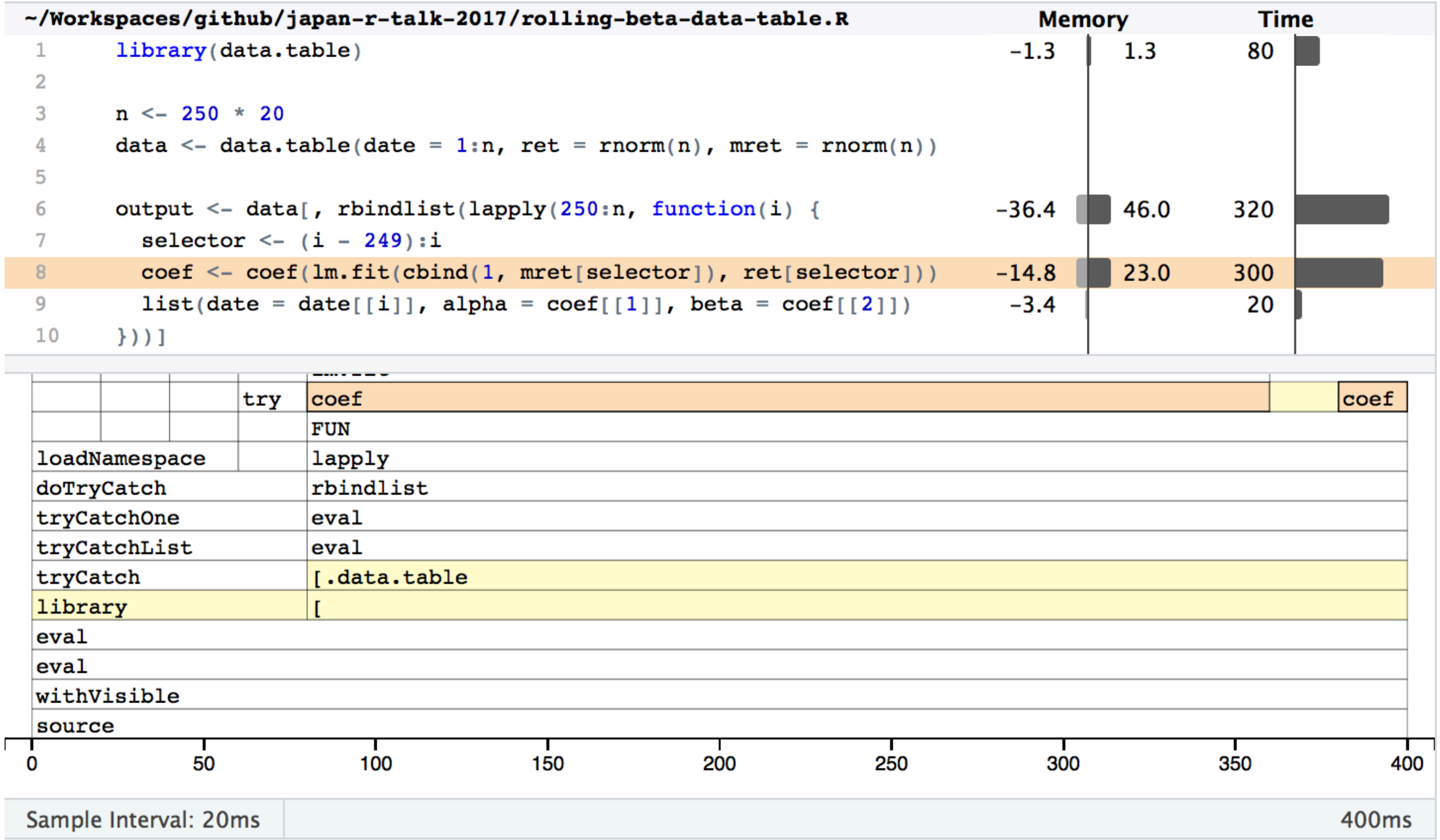
Avoid row selection from `data.frame` (440ms)



Rewrite with `data.table`

```r
library(data.table)

n <- 250 * 20
data <- data.table(date = 1:n, ret = rnorm(n), mret = rnorm(n))

output <- data[, rbindlist(lapply(250:n, function(i) {
  selector <- (i - 249):i
  coef <- coef(lm.fit(cbind(1, mret[selector]), ret[selector]))
  list(date = date[[i]], alpha = coef[[1]], beta = coef[[2]])
}))]
```

Rewrite with `data.table` (320ms)



Multi-symbol rolling regression

symbol	date	ret	mret
1	1	-0.560475647	-0.079625050
1	2	-0.230177489	0.045941483
1	3	1.558708314	0.126138891
1	4	0.070508391	-0.001914877
1	5	0.129287735	-0.058701079
1	6	1.715064987	-0.014730471
1	7	0.460916206	0.014168433
1	8	-1.265061235	0.028467197
1	9	-0.686852852	0.083210147
1	10	-0.445661970	-0.190119626
1	11	1.224081797	0.033166720
1	12	0.359813827	0.043090255
1	13	0.400771451	0.092460001
1	14	0.110682716	-0.112767268
1	15	-0.555841135	-0.217610923
1	16	1.786913137	0.096349661
1	17	0.497850478	-0.052769189

```
library(data.table)
set.seed(123)
dt <- expand.grid(symbol = 1:100, date = 1:(250 * 20))
setDT(dt, key = c("symbol", "date"))
dt[, ret := rnorm(.N)]
dt[, mret := mean(ret), by = date]
```

- 100 symbols, 20 years of history
- `setDT` to transform `dt` as `data.table`
- `(symbol, date)` as key to ensure order
- For each symbol and each date, estimate the linear coefficients for the recent 250 days of that symbol

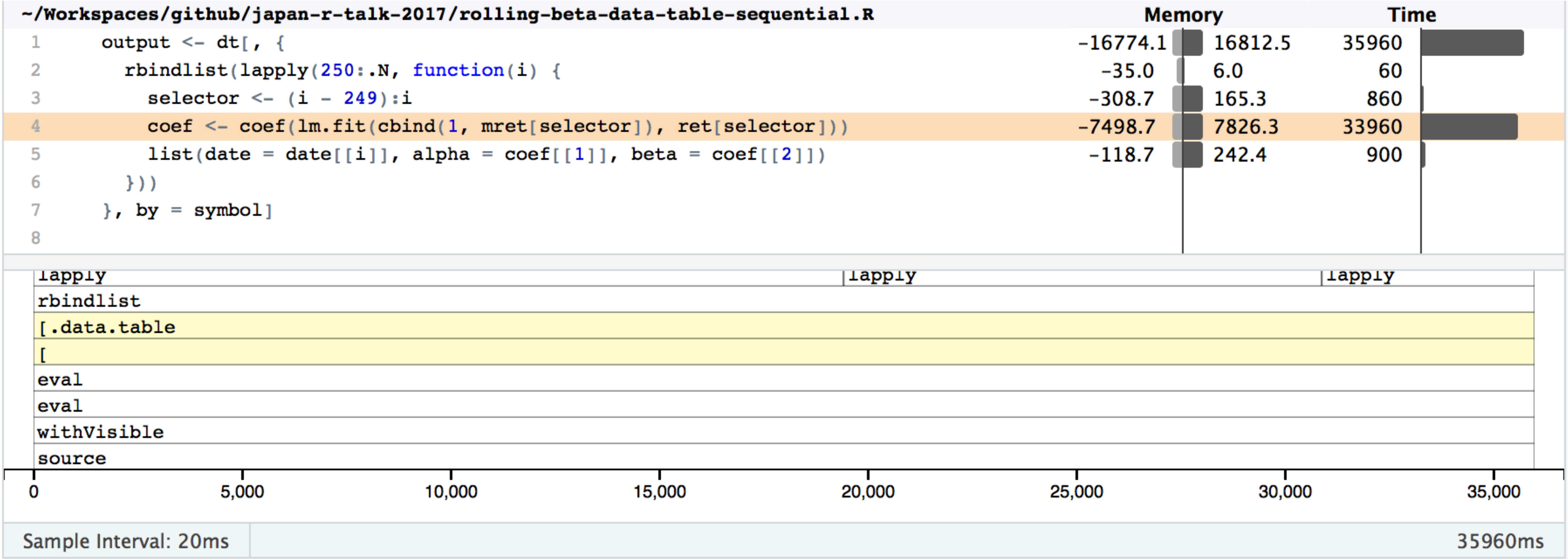
The `data.table` solution

```r
output <- dt[, {
  rbindlist(lapply(250:.N, function(i) {
    selector <- (i - 249):i
    coef <- coef(lm.fit(cbind(1, mret[selector]), ret[selector]))
    list(date = date[[i]], alpha = coef[[1]], beta = coef[[2]])
  }))
}, by = symbol]
```

Rolling by each symbol
Combine with `rbindlist`

Use by for group operation

The `data.table` solution (35960ms)



`mclapply` + `data.table` solution (8640ms)

```r
library(parallel)
symbols <- dt[, sort(unique(symbol))]
output <- mclapply(symbols, function(sym) {
  dt[symbol == sym, {
    rbindlist(lapply(250:.N, function(i) {
      selector <- (i - 249):i
      coef <- coef(lm.fit(cbind(1, mret[selector]), ret[selector]))
      list(date = date[[i]], alpha = coef[[1]], beta = coef[[2]])
    }))
  }, by = symbol]
}, mc.cores = 8)
output <- rbindlist(output)
setkey(output, symbol, date)
```

Automatically use binary search on symbol

```
  user  system elapsed
52.637   2.097   8.640
```

Solutions to R performance issues

- Built-in or lower-level functions (`lm -> lm.fit`)
- Vectorization (`cumsum, cumprod, ...`)
- Matrix algebra (`%*%, solve, crossprod, ...`)
- High performance packages (data.table, fst, RcppRoll, ...)
- Rcpp and Rcpp template packages (RcppArmadillo, RcppEigen, etc.)

Package for performance: data.table

- Algorithm: Fast merge based on binary search
- OpenMP-powered multi-threaded subsetting
- Control copy behavior: In-place modification (`:=`)
- High performance helper functions
  - `setnames, setcolorder, setkey, setindex, ...`
  - `rbindlist, dcast, melt, frank, ...`
  - `%between%, %chin%, ...`
  - `fread, fwrite`
- The data.table cheatsheet by DataCamp

Package for performance: fst

https://github.com/fstpackage/fst: Lightning Fast Serialization of Data Frames for R

Chinese Stock Market 1-minute data (2014-2017): 542M+ rows, 23 columns

Function	Compression	File size (GB)	Time (s)
saveRDS	Yes	22	3214
readRDS	Yes	22	341
saveRDS	No	75	166
readRDS	No	75	96
data.table::fwrite	No	73	57
data.table::fread	No	73	1134
fst::write_fst	No	75	42
fst::read_fst	No	75	37
fst::write_fst	100 (40 threads)	25	37
fst::read_fst	100 (40 threads)	25	11

Parallel computing

Multi-threading

- OpenBLAS
- Intel MKL
- OpenMP

Drop-in solutions

Multi-processing

- Fork-based (Linux, macOS)
- Socket-based (Windows)

Platform dependent

Multi-node

- Hadoop
- Spark
- etc.

Use parallel: fork vs. cluster

- Platform dependent:
  - Windows: socket-based master-slave cluster
  - Linux and macOS: mcparallel and mccollect
- Inter-process communication: serialization: slow and high memory peak
- Multi-threading techniques do not work well with fork-based parallelism.
- Use `RhpcBLASctl` to force number of threads in OpenMP and BLAS to avoid deadlock in fork process

Rcpp

- R data structures + C++ data structures
- Multi-threading by OpenMP
- Template packages: RcppArmadillo, RcppEigen, etc.
- Rcpp now used by 1000+ packages

data.table + Rcpp + RcppArmadillo (3700ms)

C++

```cpp
// [[Rcpp::depends(RcppArmadillo)]]
#include <RcppArmadillo.h>
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
List roll_lm(const IntegerVector& date, const arma::mat& x, const arma::colvec& y, int n) {
  int size = date.size();
  IntegerVector _date(size - n + 1);
  NumericVector alpha(size - n + 1), beta(size - n + 1);
  for (int i = n - 1; i < size; i++) {
    const arma::mat& xi = x.rows(i - n + 1, i);
    const arma::colvec& yi = y.subvec(i - n + 1, i);
    const arma::colvec& coef = solve(xi, yi);
    _date[i - n + 1] = date[i];
    alpha[i - n + 1] = coef[0];
    beta[i - n + 1] = coef[1];
  }
  return List::create(_["date"] = _date, _["alpha"] = alpha, _["beta"] = beta);
}
```

R

```r
output <- dt[, roll_lm(date, cbind(1, mret), ret, 250), by = symbol]
```

```
 user  system elapsed
3.669   0.029   3.700
```

data.table + Rcpp + RcppArmadillo + OpenMP (840ms)

C++

```cpp
// [[Rcpp::depends(RcppArmadillo)]]
// [[Rcpp::plugins(openmp)]]
#include <omp.h>
#include <RcppArmadillo.h>
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
List par_roll_lm(const IntegerVector& date, const arma::mat& x, const arma::colvec& y, int n, int threads) {
  int size = y.size();
  IntegerVector _date(size - n + 1);
  NumericVector alpha(size - n + 1), beta(size - n + 1);
#pragma omp parallel for shared(date, x, y), num_threads(threads)
  for (int i = n - 1; i < size; i++) {
    const arma::mat& xi = x.rows(i - n + 1, i);
    const arma::colvec& yi = y.subvec(i - n + 1, i);
    const arma::colvec& coef = solve(xi, yi);
    INTEGER(_date)[i - n + 1] = date[i];
    REAL(alpha)[i - n + 1] = coef[0];
    REAL(beta)[i - n + 1] = coef[1];
  }
  return List::create(_["date"] = _date, _["alpha"] = alpha, _["beta"] = beta);
}
```

Use OpenMP header file and update compiler flags

OpenMP transforms the for loop into parallel

Use pointer to update R vectors

R

```r
output <- dt[, par_roll_lm(date, cbind(1, mret), ret, 250, threads = 8), by = symbol]
```

```
 user  system elapsed
6.375   0.065   0.840
```

Performance summary

Solution	Time	Performance
Initial solution (plain for loop)	6880ms	1x
Replace lm with lm.fit	3180ms	2x
Pre-allocate vectors	800ms	9x
Avoid row selection from data.frame	440ms	16x
Rewrite with data.table	320ms	21x
Increase to 100 symbols	35960ms	21x
mclapply + data.table	8640ms	87x (mc.core = 8)
data.table + Rcpp + RcppArmadillo	3700ms	204x
data.table + Rcpp + RcppArmadillo + OpenMP	840ms	900x (threads = 8)

- Kun Ren <renkun@outlook.com>
- GitHub: renkun-ken
- Twitter: @renkun_ken
- Slides and code: https://github.com/renkun-ken/japan-r-talk-2017