Disseminating Prediction Methods

Avoiding Computational Bottlenecks and Developing User-Friendly APIs

Byron C Jaeger

Wake Forest University School of Medicine

2022/08/08 (updated: 2022-08-07)

(Bio)statisticians create methods that can engage with contemporary data and make valid conclusions

Statistical software allows these methods to be shared with investigators

From proprietary to open-source

When SAS and SPSS were prominent, methods were shared by incorporating them into proprietary software.

As R and Python have become standard languages for data science, it has become more common for authors to write their own software for methods.

obliqueRSF

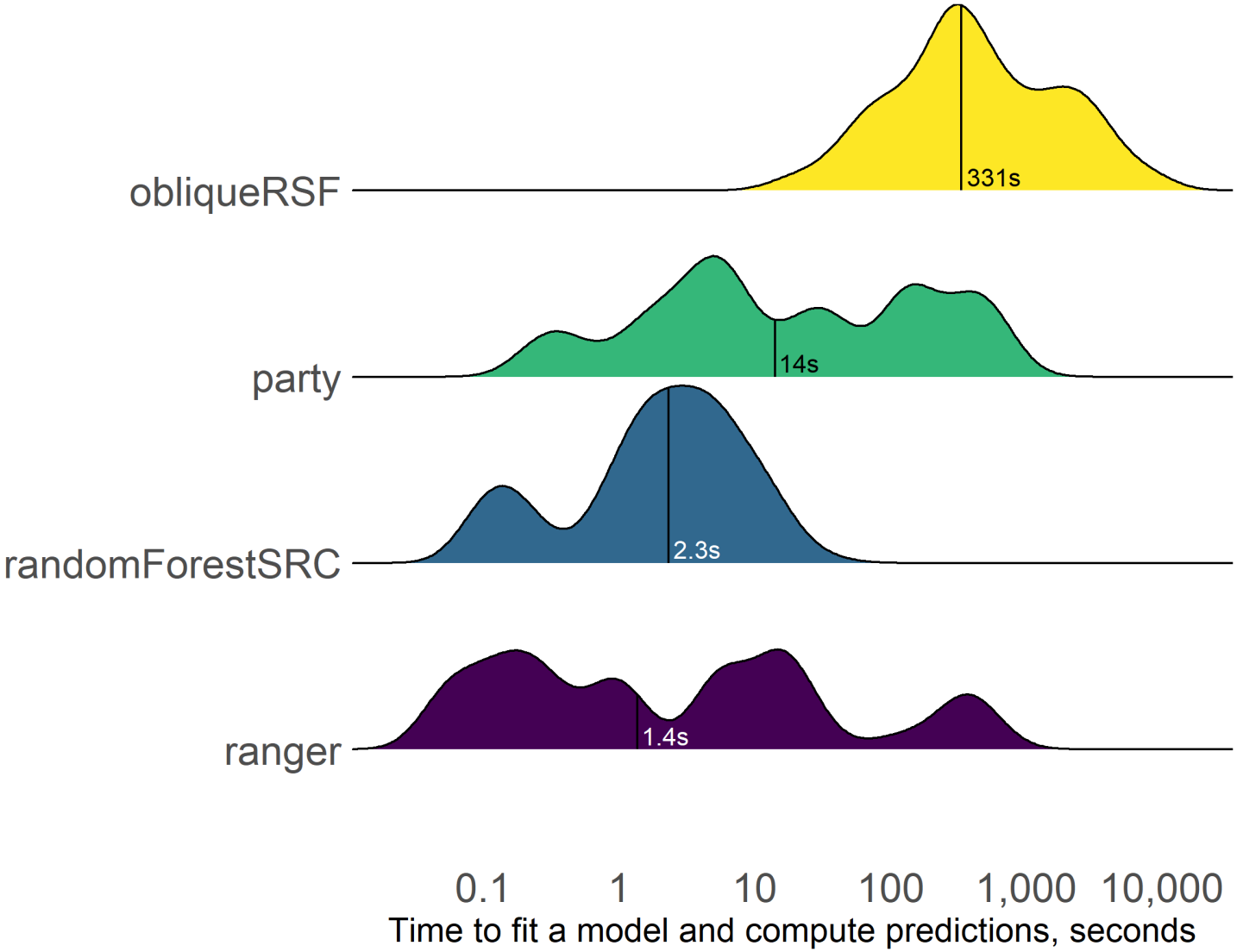
In 2019, I made the `obliqueRSF` R package

- oblique random survival forests

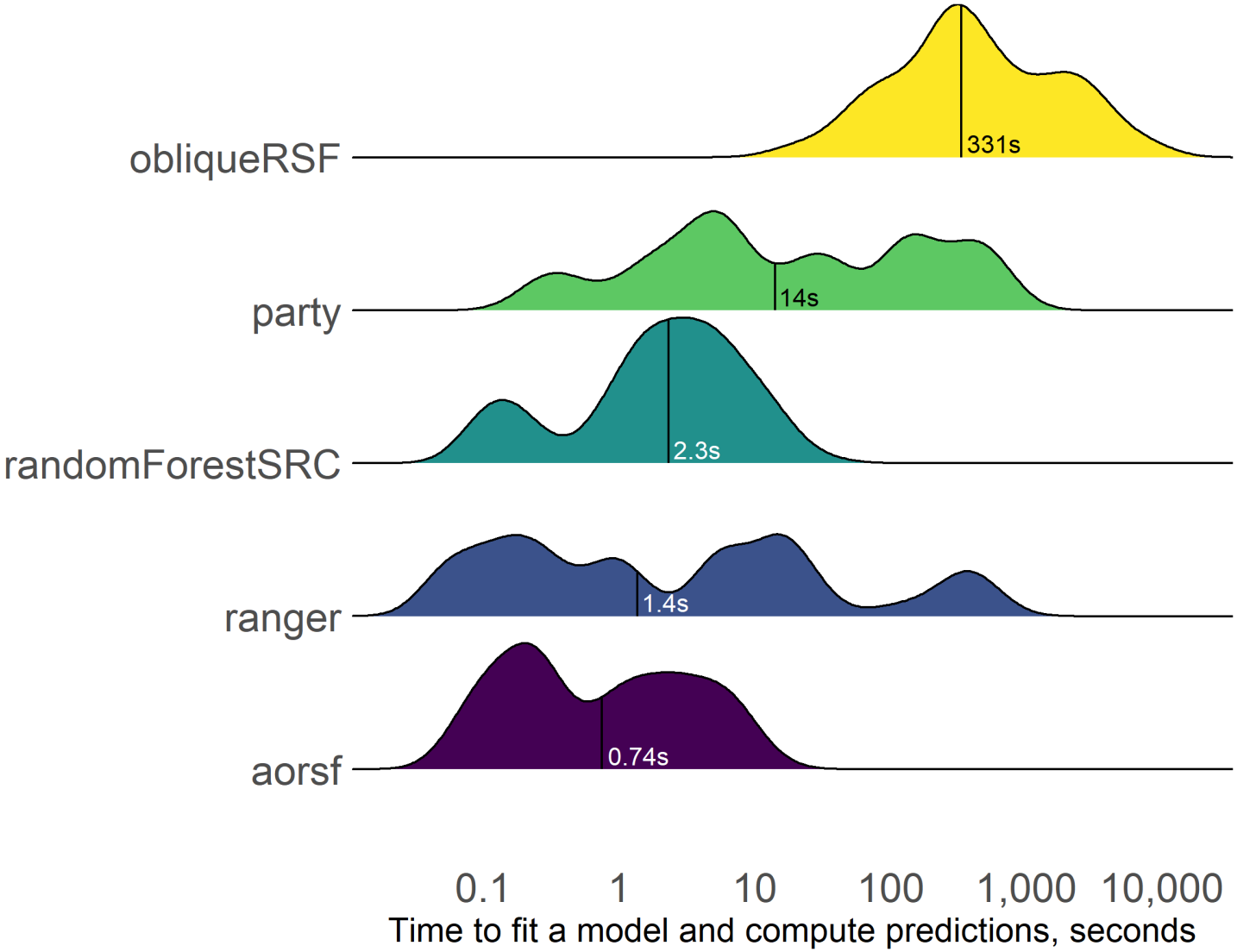
`obliqueRSF` had higher prediction accuracy versus:

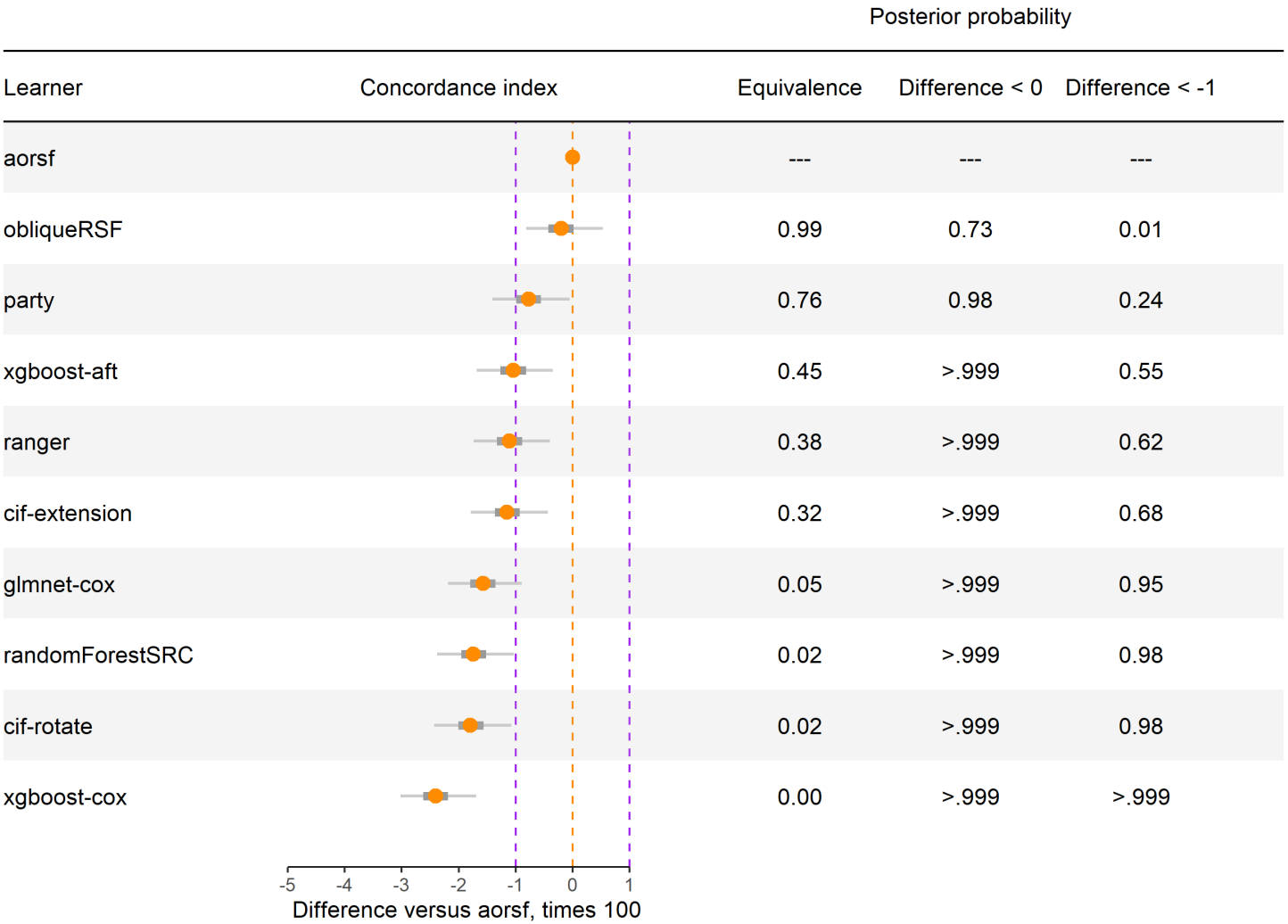
- `randomForestSRC`,
- `ranger`,
- `party`.

But was also hundreds of times slower.

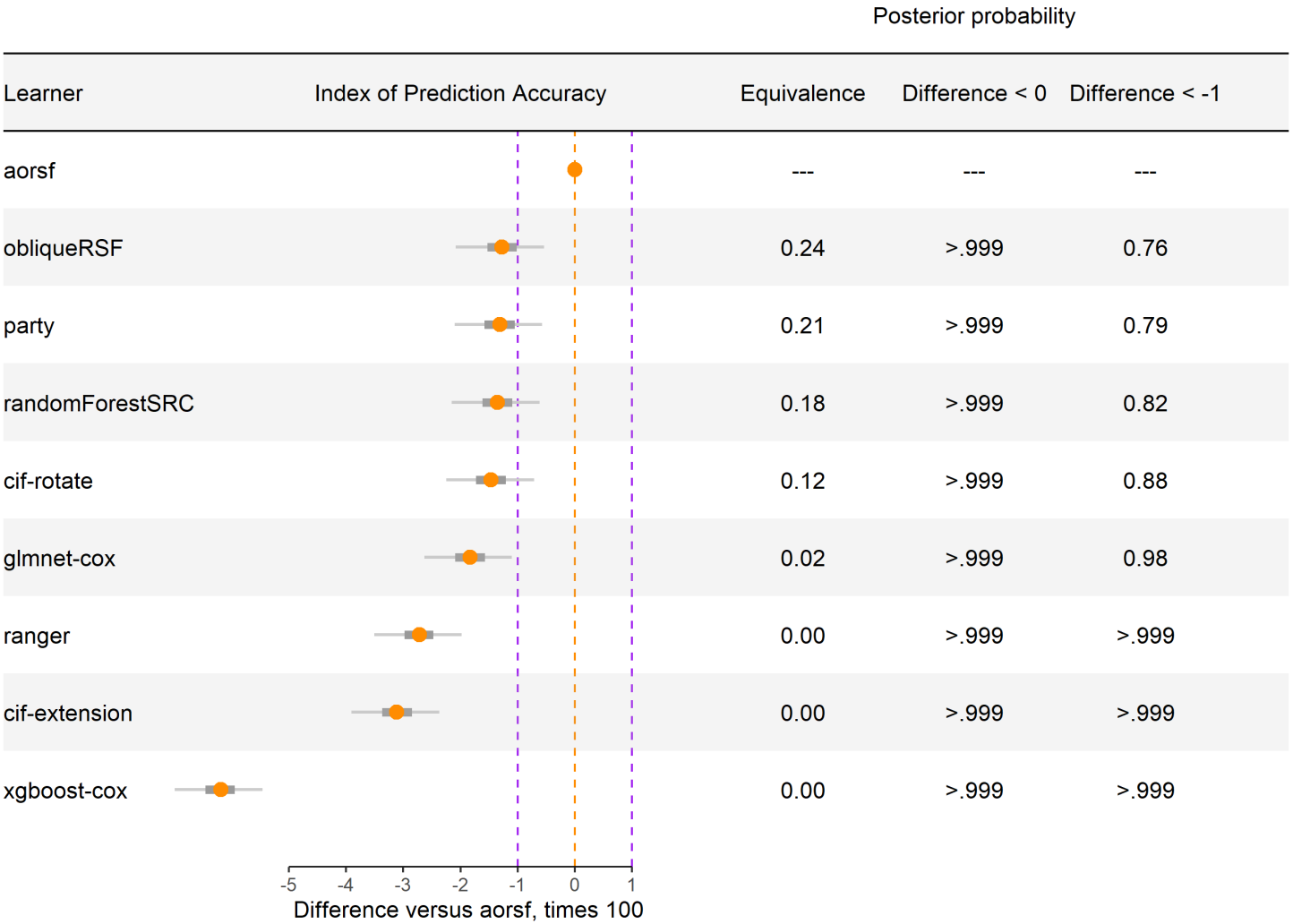

obliqueRSF
331s
party
14s
randomForestSRC
2.3s
ranger
1.4s
0.1
1
10
100
1,000
10,000
Time to fit a model and compute predictions, seconds


Tell me the truth...I'm...I'm ready to hear it.
NOBODY USES SLOW R PACKAGES
imgflip.com


obliqueRSF
331s
party
14s
randomForestSRC
2.3s
ranger
1.4s
aorsf
0.74s
0.1
1
10
100
1,000
10,000
Time to fit a model and compute predictions, seconds


Posterior probability
Learner
Concordance index
Equivalence
Difference < 0
Difference < -1
aorsf

---
obliqueRSF
0.99
0.73
0.01
party
0.76
0.98
0.24
xgboost-aft
0.45
>.999
0.55
ranger
0.38
>.999
0.62
cif-extension
0.32
>.999
0.68
glmnet-cox
0.05
>.999
0.95
randomForestSRC
0.02
>.999
0.98
cif-rotate
0.02
>.999
0.98
xgboost-cox
0.00
>.999
>.999
-5
-4
-3
-2
-1
0
1
Difference versus aorsf, times 100


Posterior probability
Learner
Index of Prediction Accuracy
Equivalence
Difference < 0
Difference < -1
aorsf

---
obliqueRSF
0.24
>.999
0.76
party
0.21
>.999
0.79
randomForestSRC
0.18
>.999
0.82
cif-rotate
0.12
>.999
0.88
glmnet-cox
0.02
>.999
0.98
ranger
0.00
>.999
>.999
cif-extension
0.00
>.999
>.999
xgboost-cox
0.00
>.999
>.999
-5
-4
-3
-2
-1
0
1
Difference versus aorsf, times 100

aorsf paper (https://arxiv.org/abs/2208.01129)

[Submitted on 1 Aug 2022]

Accelerated and interpretable oblique random survival forests

Byron C. Jaeger, Sawyer Welden, Kristin Lenoir, Jaime L. Speiser, Matthew W. Segar, Ambarish Pandey, Nicholas M. Pajewski

The oblique random survival forest (RSF) is an ensemble supervised learning method for right-censored outcomes. Trees in the oblique RSF are grown using linear combinations of predictors to create branches, whereas in the standard RSF, a single predictor is used. Oblique RSF ensembles often have higher prediction accuracy than standard RSF ensembles. However, assessing all possible linear combinations of predictors induces significant computational overhead that limits applications to large-scale data sets. In addition, few methods have been developed for interpretation of oblique RSF ensembles, and they remain more difficult to interpret compared to their axis-based counterparts. We introduce a method to increase computational efficiency of the oblique RSF and a method to estimate importance of individual predictor variables with the oblique RSF. Our strategy to reduce computational overhead makes use of Newton-Raphson scoring, a classical optimization technique that we apply to the Cox partial likelihood function


AORSF

Four guidelines for faster code

1. Always be benchmarking

Why "always"?

How to benchmark?

Step 1. Define a task: Count events per group (status of 1 $\Rightarrow$ event, 3 groups)

```
# status = c(1, 0, 1, ...)
status = sample(x = c(0L, 1L), size = 1e5, replace = TRUE)

# group = c(2, 0, 1, ...)
group = sample(x = c(0L, 1L, 2L), size = 1e5, replace = TRUE)
```

Step 2. Define the reference competitors

```
# competitors: table() and tapply()
table(status, group)[2, ]
```

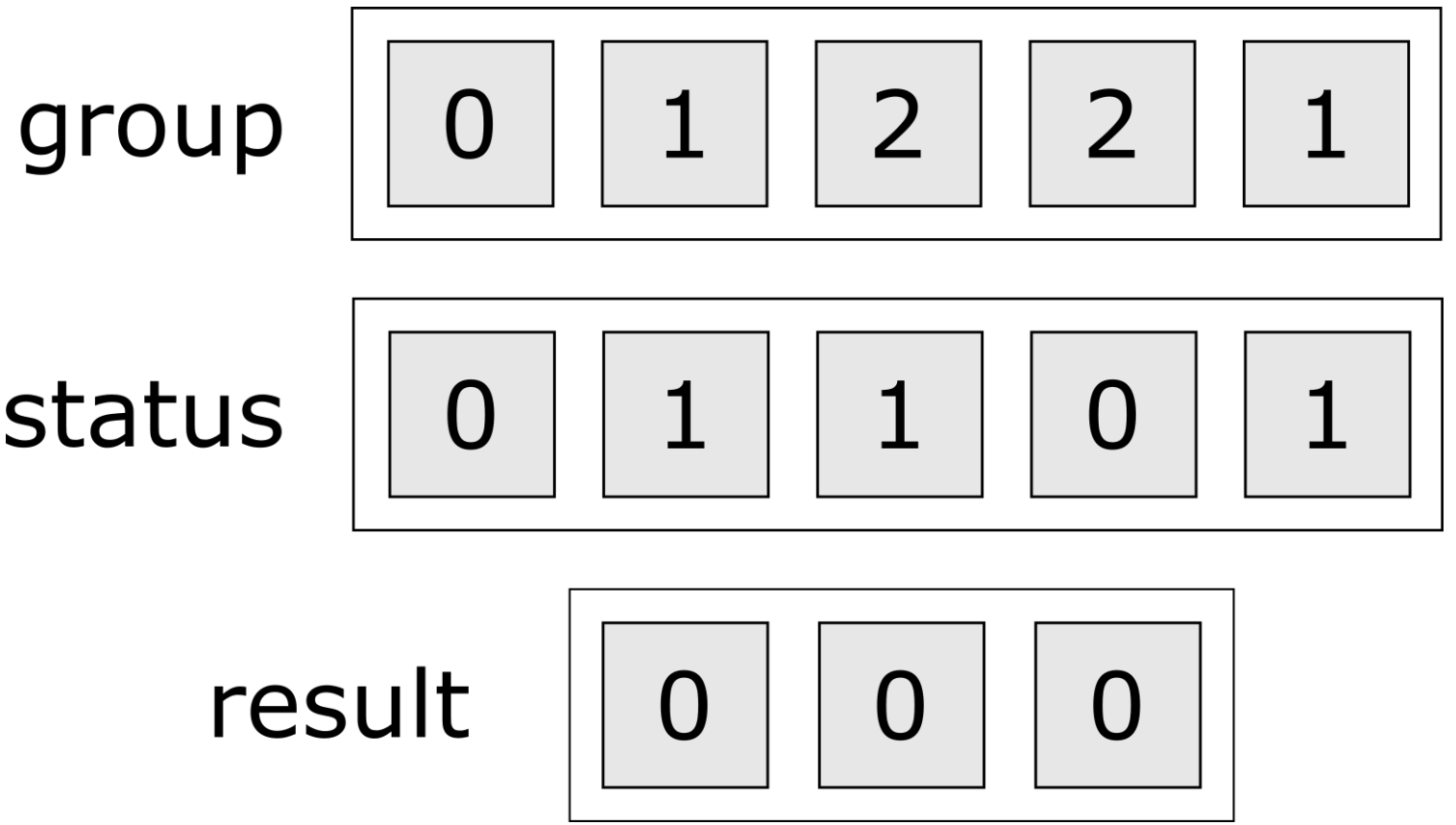
```
##     0     1     2
## 16780 16570 16497
```

```
tapply(status, group, FUN = sum)
```

```
##     0     1     2
## 16780 16570 16497
```

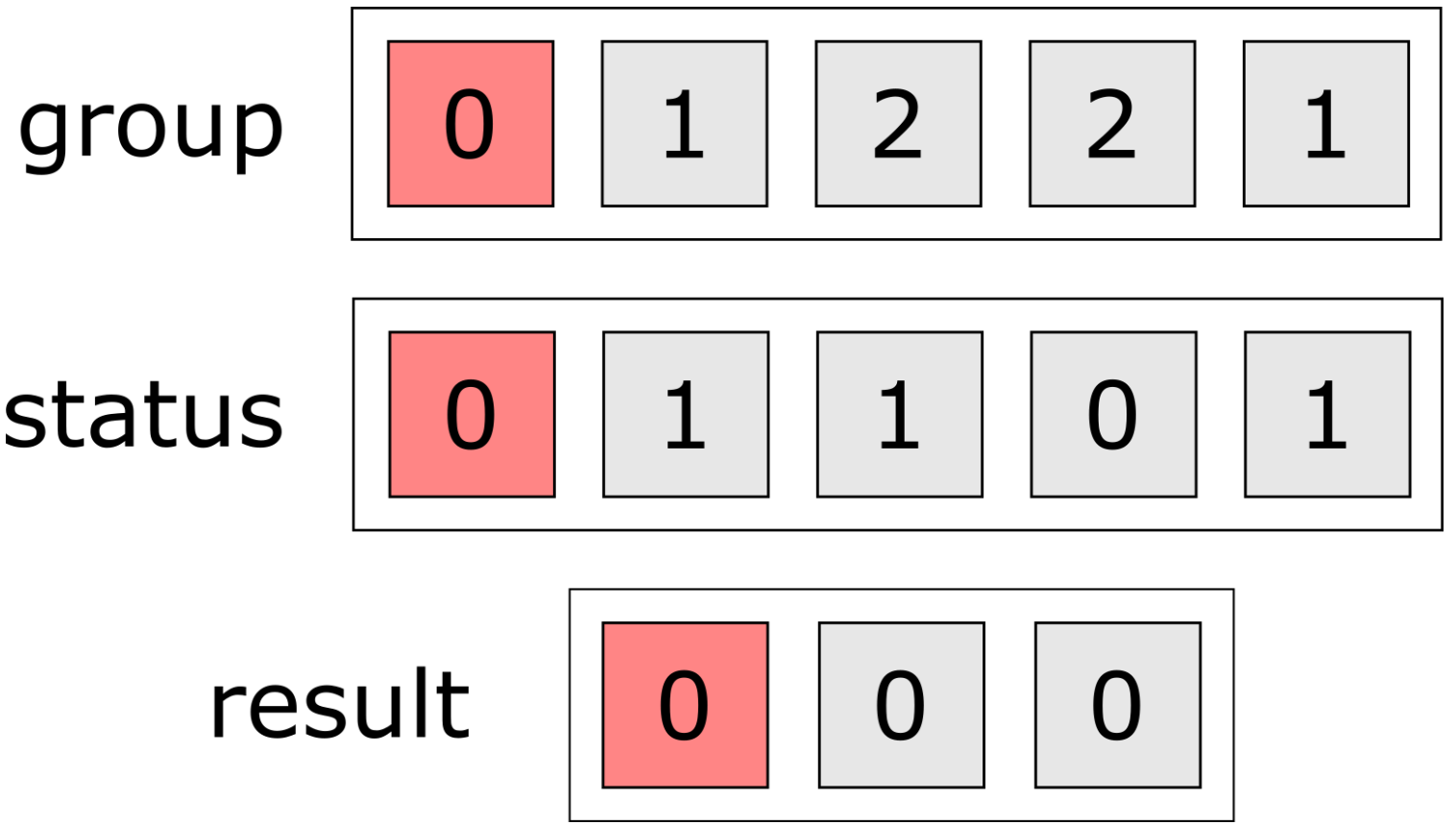
How to benchmark?

Step 3 (optional). Enter your own competitor(s)!



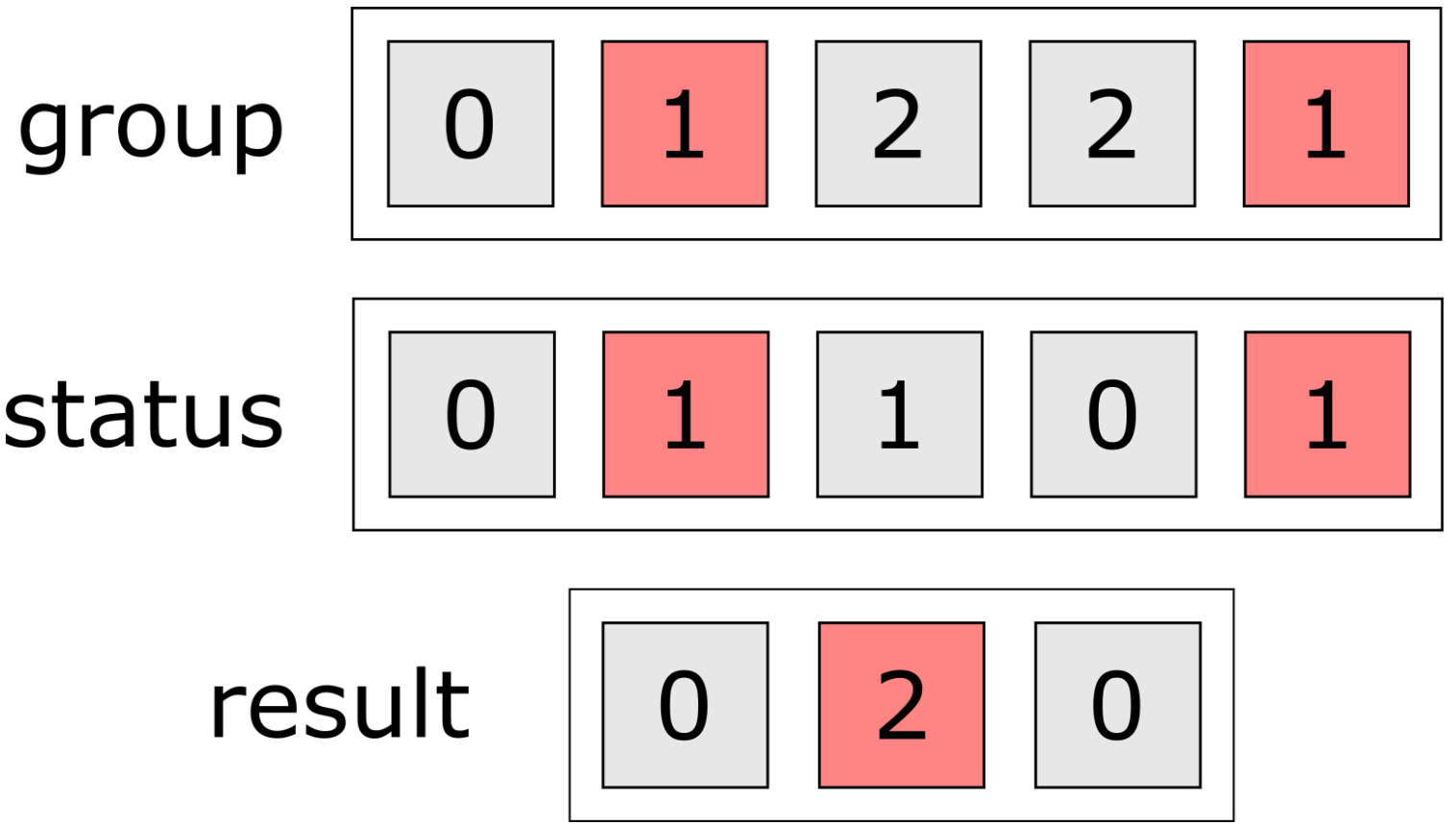
How to benchmark?

Step 3 (optional). Enter your own competitor(s)!



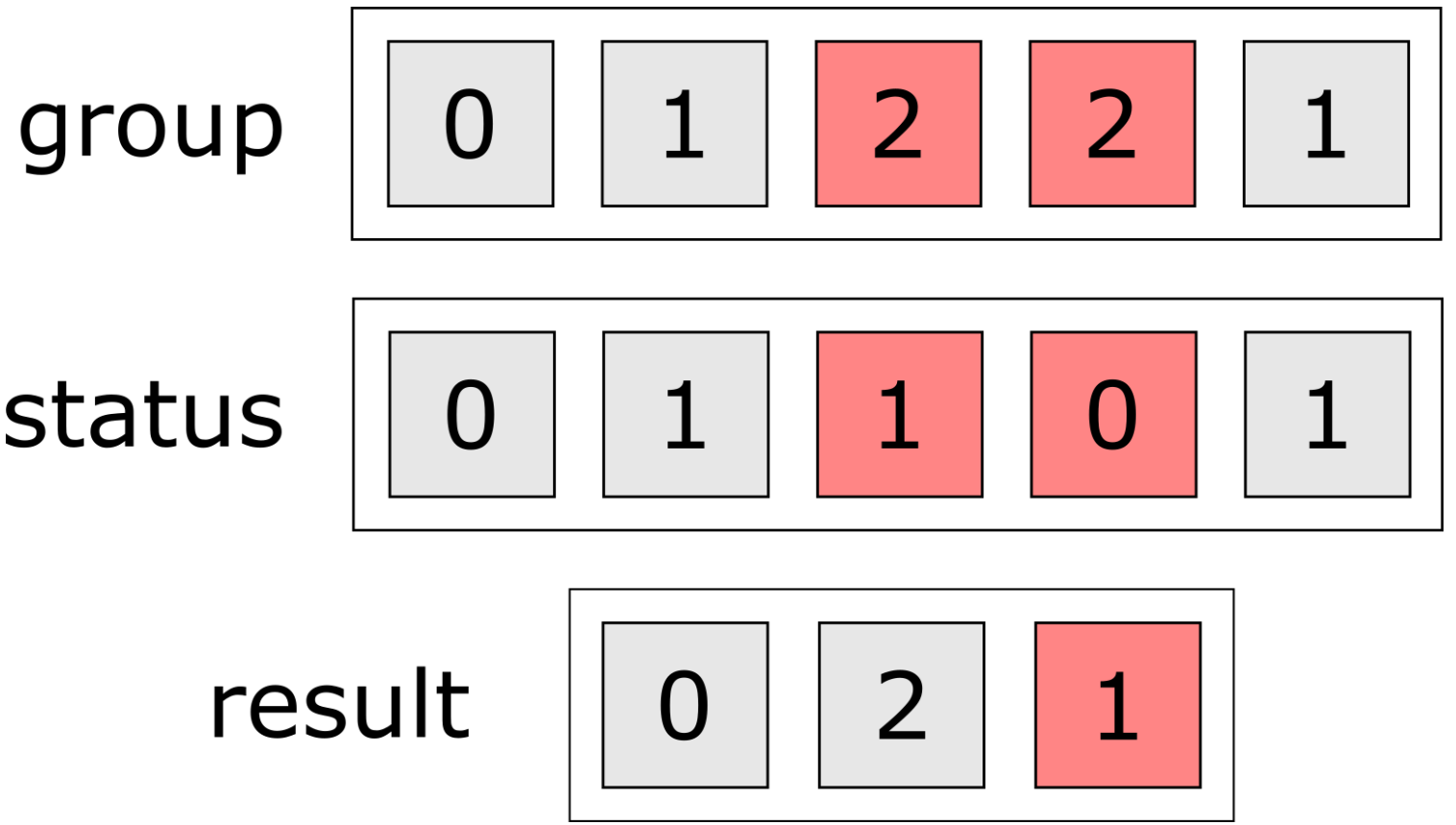
How to benchmark?

Step 3 (optional). Enter your own competitor(s)!



How to benchmark?

Step 3 (optional). Enter your own competitor(s)!



How to benchmark?

Step 3 (optional). Enter your own competitor(s)!

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector rcpp_count_dbl(NumericVector status,
                             NumericVector group,
                             int n_groups) {

 NumericVector out(n_groups);

 for( int i = 0; i < n_groups; i++ ){
  for( int j = 0; j < group.length(); j++ ){
   if(group[j] == i) out[i] += status[j];
  }
 }

 return(out);
}
```

How to benchmark?

Step 4. Off to the races:

```
library(microbenchmark)
microbenchmark(table = table(status, group)[2, ],
               tapply = tapply(status, group, FUN = sum),
               rcpp_count_dbl = rcpp_count_dbl(status, group, 3))
```

Benchmark demonstration: counting events in groups

table(), tapply(), and rcpp_count_dbl()

	Time, milliseconds					
Function	Minimum	25th %	Mean	Median	75th %	Maximum
table	5.5	6.0	7.8	*6.1*	6.3	73
tapply	2.1	2.5	3.2	*2.5*	2.6	81
rcpp_count_dbl	1.6	1.8	2.0	*1.8*	1.8	18

2. Trace your data

Trace your data

Definition be notified when data are copied or cast to a different type

(Copying and casting require additional memory, slowing down your code.)

How to trace?

Put the object you want to trace into `tracemem()`:

```
tracemem(status)
tracemem(group)
```

R will now notify you if `status` or `group` are copied or cast to a different type.

Here is a problem

`rcpp_count_dbl` casts both `status` and `group` from integer to double

```
rcpp_count_dbl(status, group, n_groups = 3)
```

```
## tracemem[0x00007ff4fd830010 -> 0x00007ff4fd660010]: .Call rcpp_count_dbl e
## tracemem[0x00007ff4fd910010 -> 0x00007ff4fd3a0010]: .Call rcpp_count_dbl e

## [1] 16780 16570 16497
```

Here is a problem

Our first Rcpp function expected numeric vectors. We gave it integers!

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector rcpp_count_dbl(NumericVector status,
                             NumericVector group,
                             int n_groups) {

NumericVector out(n_groups);

 for( int i = 0; i < n_groups; i++ ){
  for( int j = 0; j < group.length(); j++ ){
   if(group[j] == i) out[i] += status[j];
  }
 }

 return(out);
}
```

Let's fix that

use **integerVector** instead of **numericVector**

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
IntegerVector rcpp_count_int(IntegerVector status,
                             IntegerVector group,
                             int n_groups) {

IntegerVector out(n_groups);

 for( int i = 0; i < n_groups; i++ ){
  for( int j = 0; j < group.length(); j++ ){
   if(group[j] == i) out[i] += status[j];
  }
 }

 return(out);
}
```

Better

`tracemem()` has been pacified!

```
rcpp_count_int(status, group, n_groups = 3)
```

```
## [1] 16780 16570 16497
```

Faster

Benchmark demonstration: counting events in groups

table(), tapply(), and Rcpp functions

	Time, milliseconds					
Function	Minimum	25th %	Mean	Median	75th %	Maximum
table	5.5	6.0	7.8	*6.1*	6.3	73
tapply	2.1	2.5	3.2	*2.5*	2.6	81
rcpp_count_dbl	1.6	1.8	2.0	*1.8*	1.8	18
rcpp_count_int	1.4	1.4	1.4	*1.4*	1.4	2.4

3. Count your operations

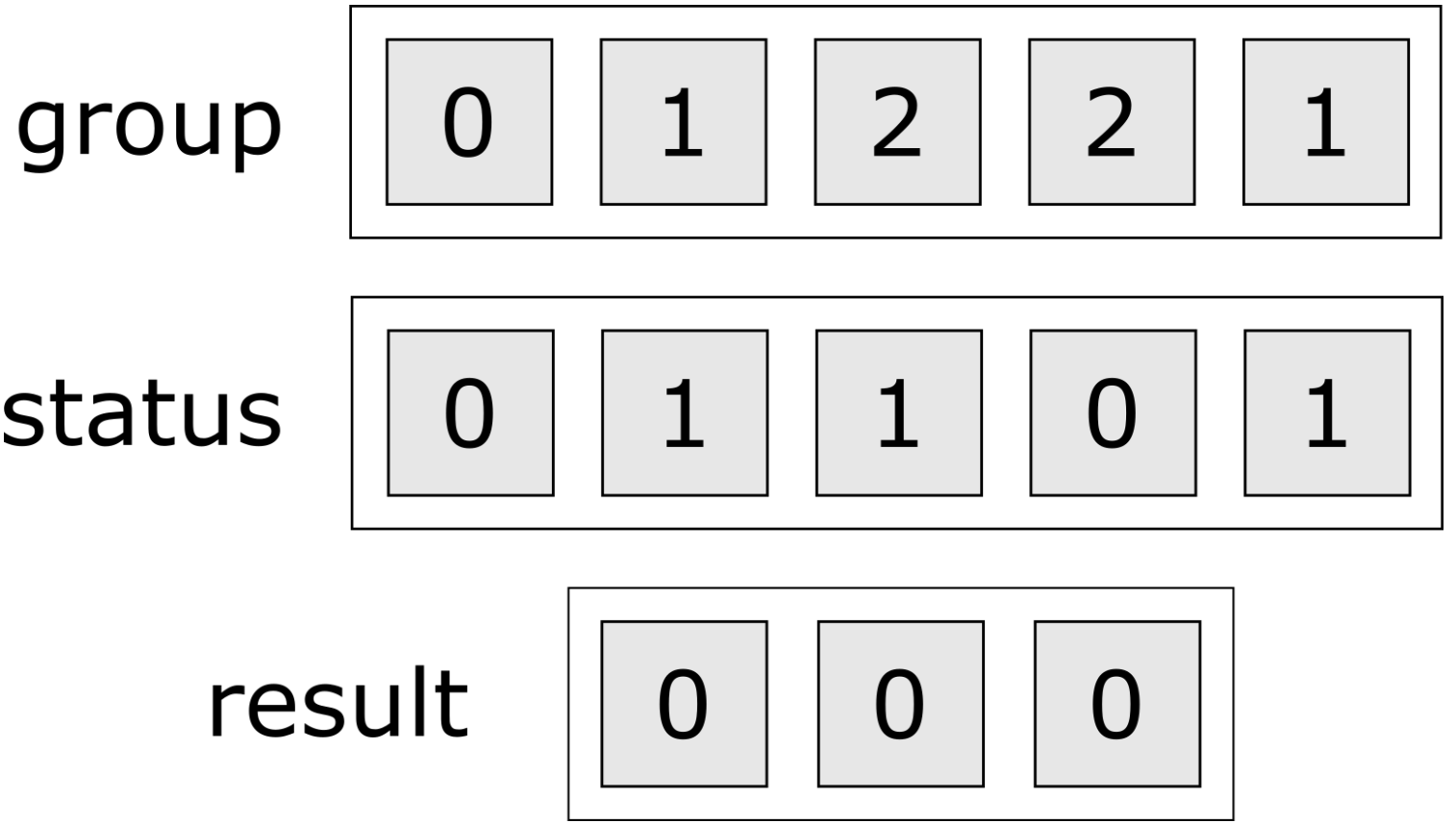
(Number of operations $\approx$ speed of your code)

How to count

You don't have to be exact - think big picture

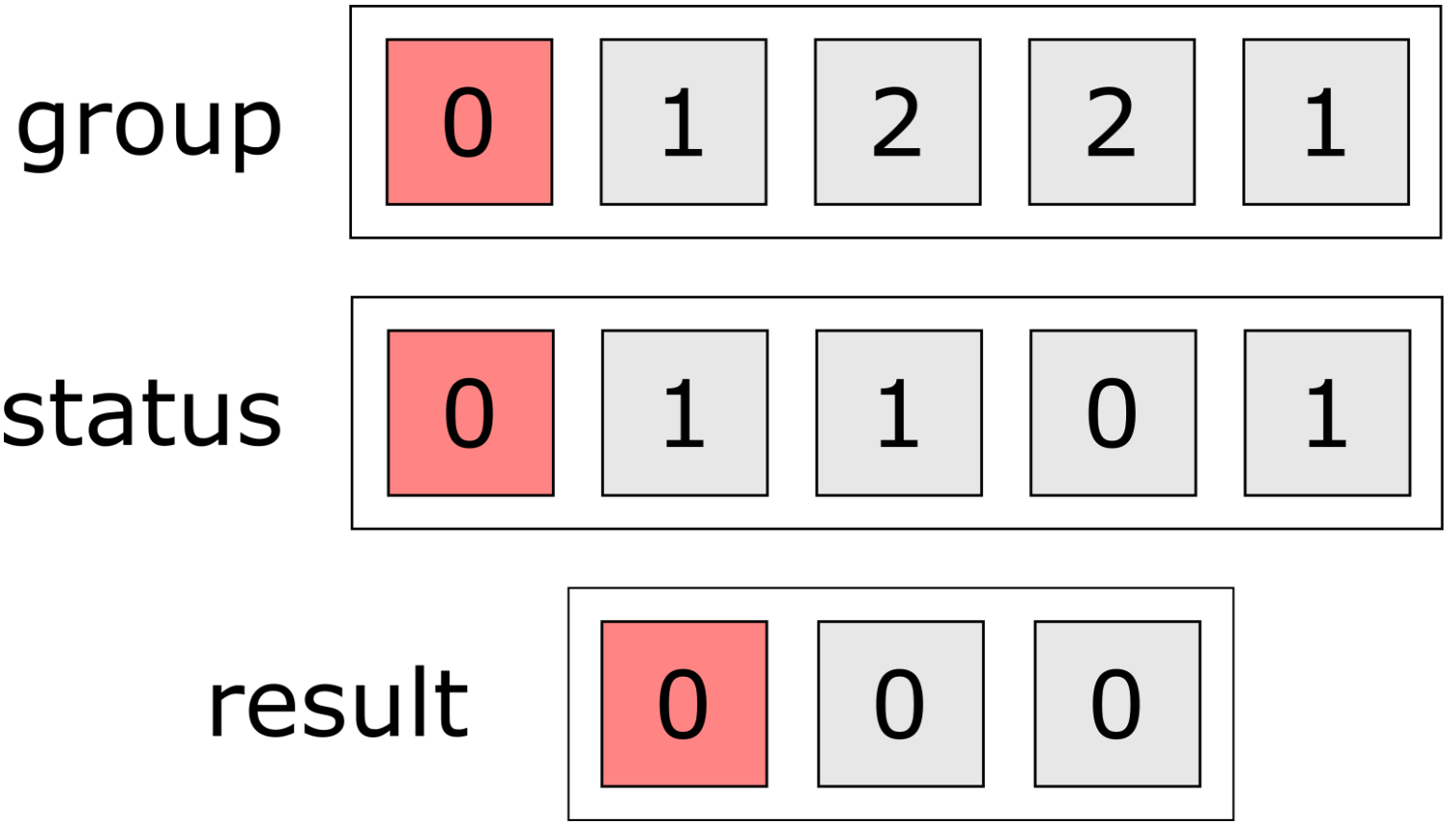
How to count

In our C++ function, we use n operations for each unique value in `group`,



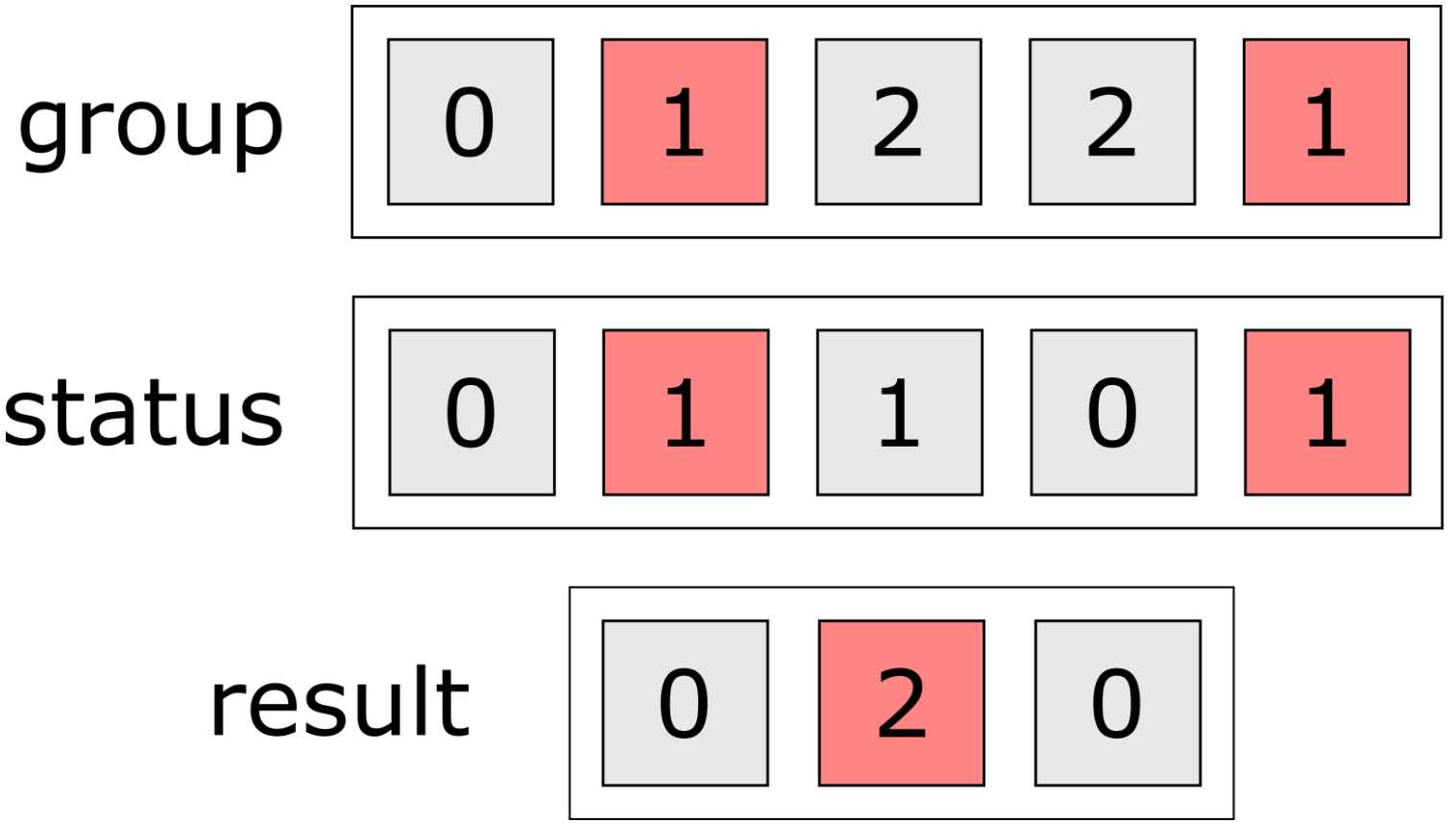
How to count

In our C++ function, we use n operations for each unique value in `group`,



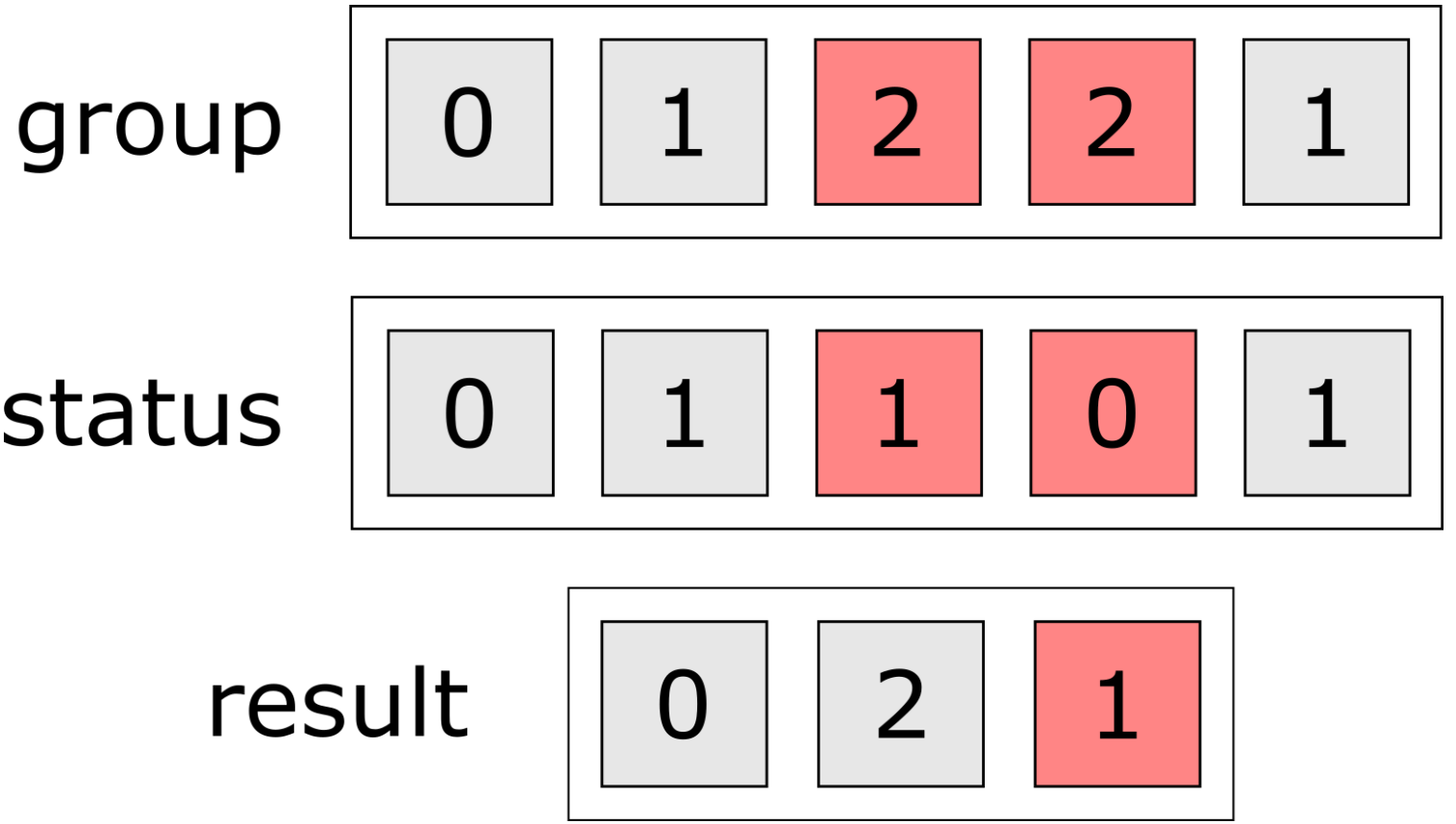
How to count

In our C++ function, we use n operations for each unique value in `group`,



How to count

In our C++ function, we use n operations for each unique value in `group`,

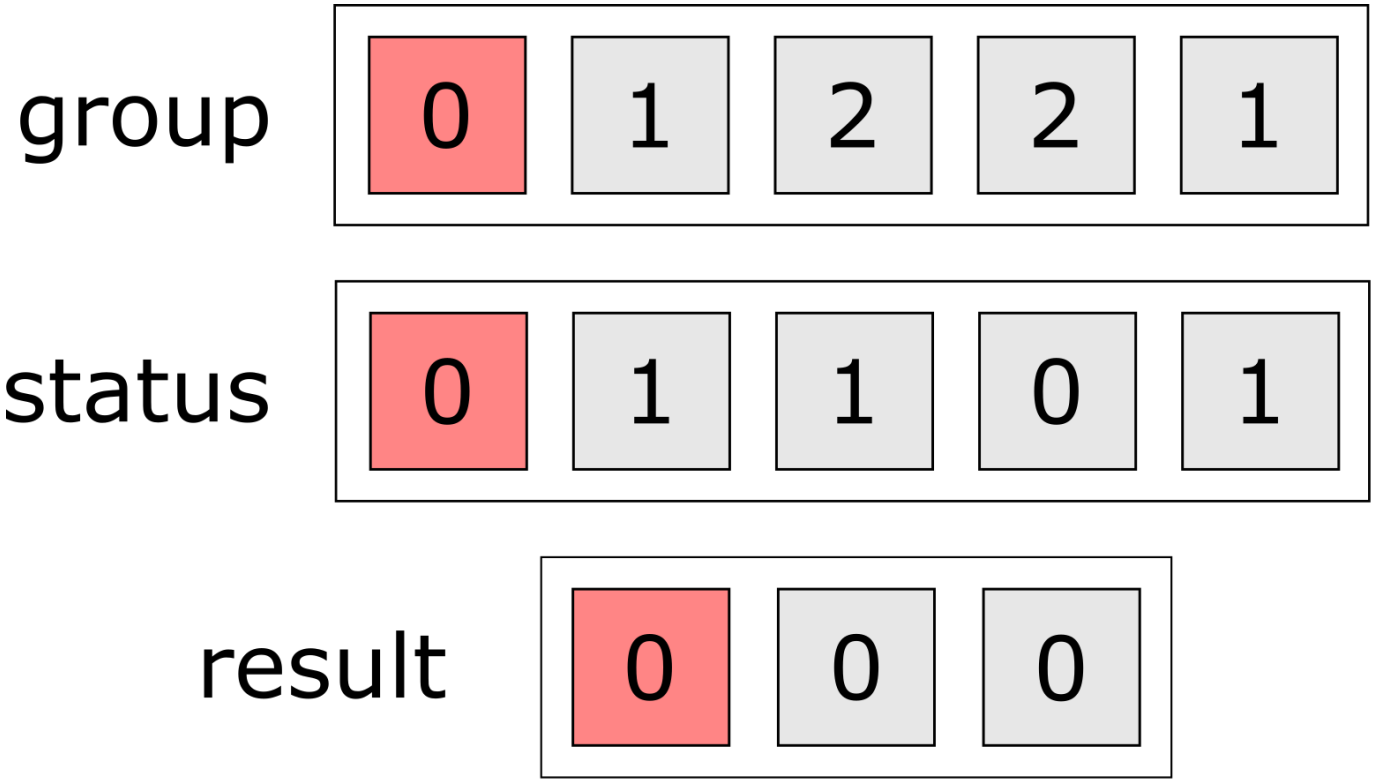


How to count

As $n, g \to \infty$, we use $\mathcal{O}(n \cdot g)$ operations, where g = number of groups

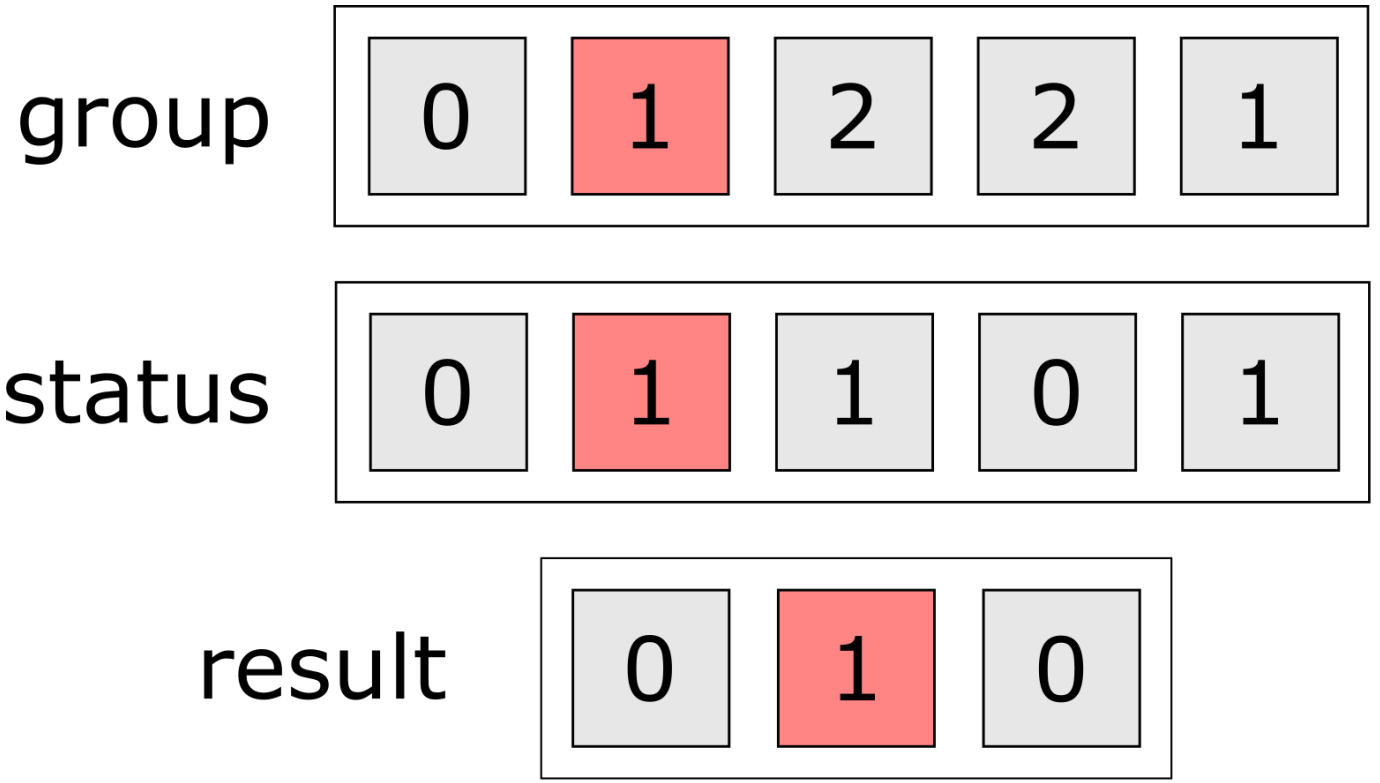
Can we reduce the operation cost?

1 loop instead of 2



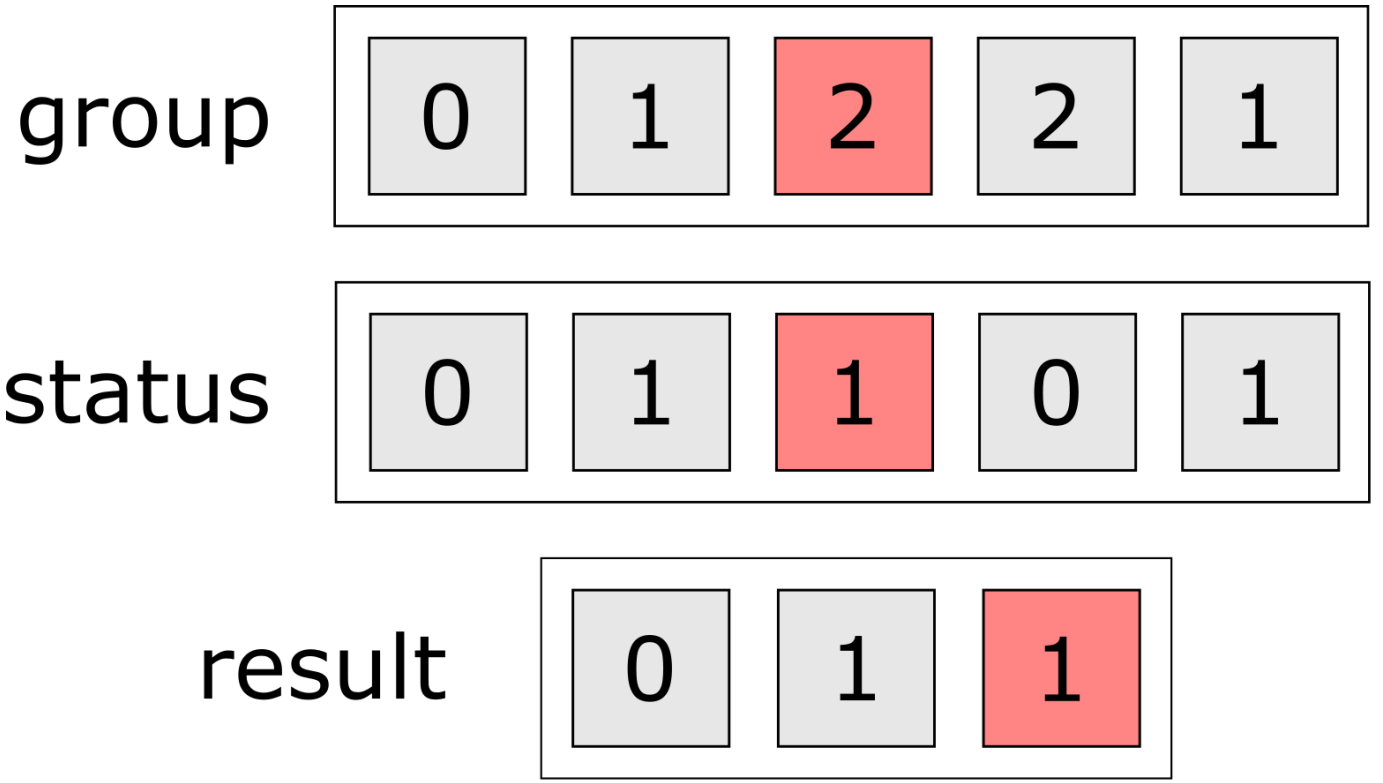

```
result[group[i]] += status[i];
```

1 loop instead of 2



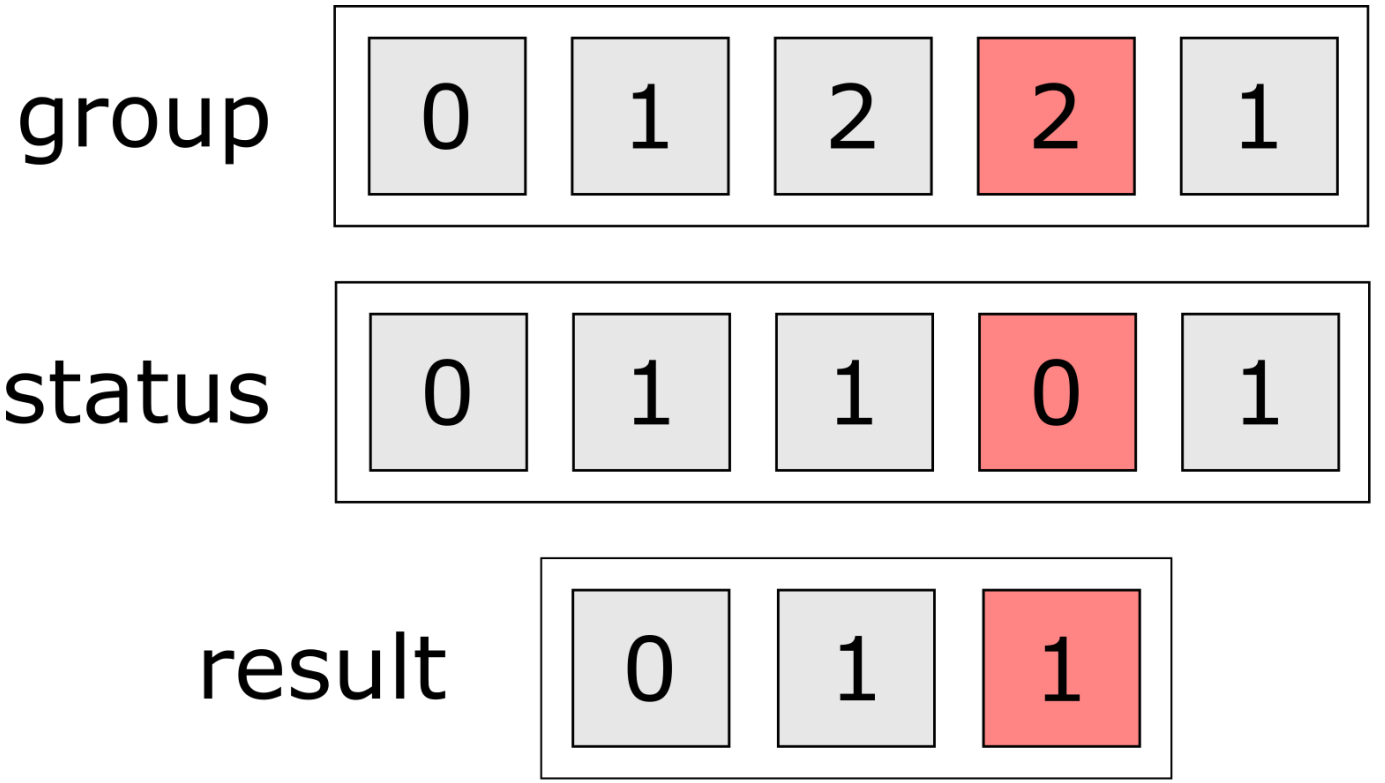

```
result[group[i]] += status[i];
```

1 loop instead of 2



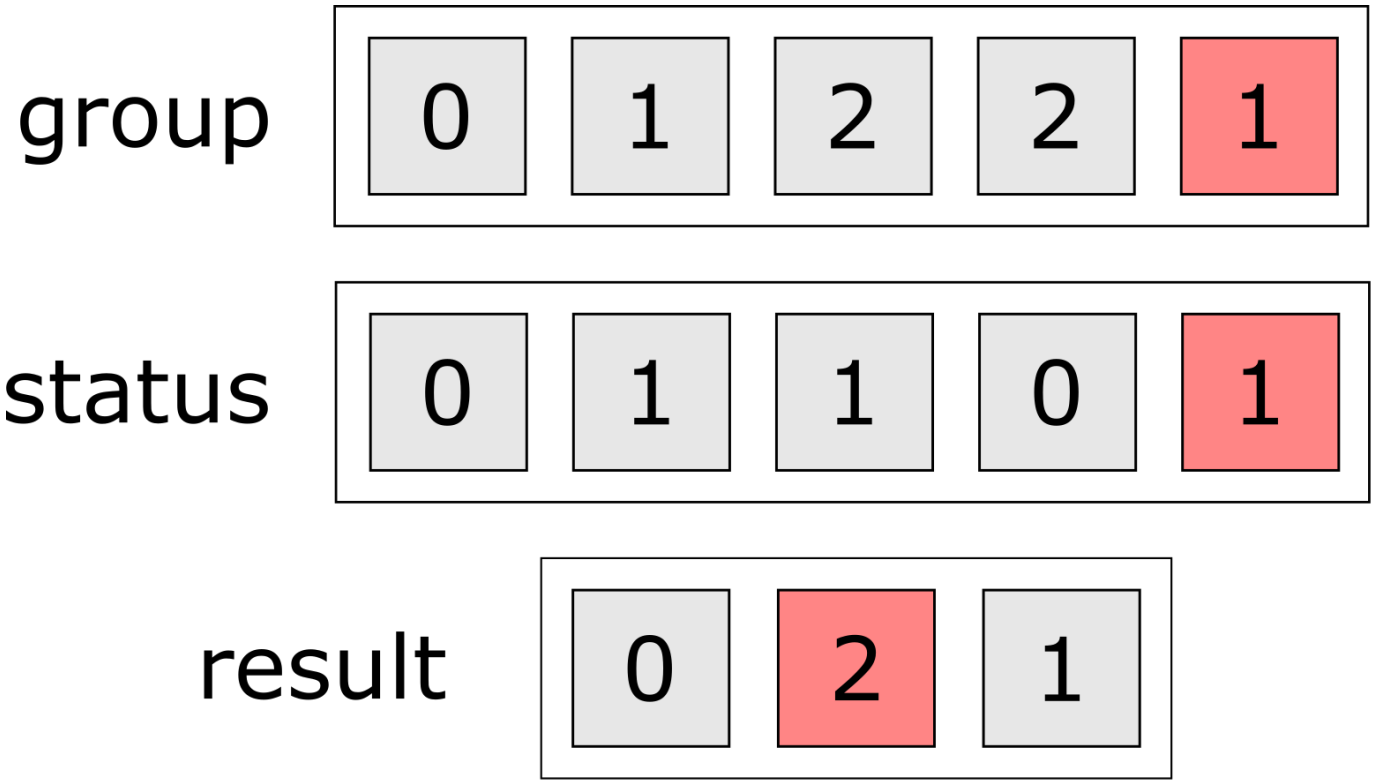

```
result[group[i]] += status[i];
```

1 loop instead of 2




```
result[group[i]] += status[i];
```

1 loop instead of 2




```
result[group[i]] += status[i];
```

1 loop instead of 2

code adapted from `rcpp_count_int`

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
IntegerVector rcpp_count_1loop_int(IntegerVector status,
                                   IntegerVector group,
                                   int n_groups) {

 IntegerVector out(n_groups);
 IntegerVector::iterator i;
 int j = 0;

 for(i = group.begin() ; i != group.end(); ++i, ++j){
  out[*i] += status[j];
 }

 return(out);
}
```

Much faster!

Benchmark demonstration: counting events in groups

table(), tapply(), and Rcpp functions

	Time, milliseconds					
Function	Minimum	25th %	Mean	Median	75th %	Maximum
table	5.5	6.0	7.8	*6.1*	6.3	73
tapply	2.1	2.5	3.2	*2.5*	2.6	81
rcpp_count_dbl	1.6	1.8	2.0	*1.8*	1.8	18
rcpp_count_int	1.4	1.4	1.4	*1.4*	1.4	2.4
rcpp_count_1loop_int	0.18	0.19	0.19	*0.19*	0.19	1.1

4. Ride the Armadillo


aorsf
RcppArmadillo

Armadillo (http://arma.sourceforge.net/)

Armadillo
C++ library for linear algebra & scientific computing

About | Documentation | Questions | Speed | Contact | Download

- Armadillo is a high quality linear algebra library (matrix maths) for the C++ language, aiming towards a good balance between speed and ease of use
- Provides high-level syntax and functionality deliberately similar to Matlab
- Useful for algorithm development directly in C++, or quick conversion of research code into production environments
- Provides efficient classes for vectors, matrices and cubes; dense and sparse matrices are supported
- Integer, floating point and complex numbers are supported
- A sophisticated expression evaluator (based on template meta-programming) automatically combines several operations to increase speed and efficiency
- Dynamic evaluation automatically chooses optimal code paths based on detected matrix structures

Armadillo

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

// [[Rcpp::export]]
arma::ivec arma_count_1loop_int(arma::ivec& status,
                                arma::ivec& group,
                                arma::uword n_groups) {

 ivec out(n_groups);
 ivec::iterator i;
 uword j = 0;

 for(i = group.begin() ; i != group.end(); ++i, ++j){
  out[*i] += status[j];
 }

 return(out);

}
```

Better! (almost perfect)

Benchmark demonstration: counting events in groups

table(), tapply(), and Rcpp functions

Function	Time, milliseconds					
	Minimum	25th %	Mean	Median	75th %	Maximum
table	5.5	6.0	7.8	*6.1*	6.3	73
tapply	2.1	2.5	3.2	*2.5*	2.6	81
rcpp_count_dbl	1.6	1.8	2.0	*1.8*	1.8	18
rcpp_count_int	1.4	1.4	1.4	*1.4*	1.4	2.4
rcpp_count_1loop_int	0.18	0.19	0.19	*0.19*	0.19	1.1
arma_count_1loop_int	0.07	0.07	0.08	*0.07*	0.08	1.0
sum	0.05	0.05	0.05	*0.05*	0.05	0.13

Creating a Friendly API

Names

be consistent: use *one* convention throughout your package

- `snake_case`
- `camelCase`
- `SCREAMING_SNAKE`
- `ConFuse.EVERY_one` (don't do it)

Name functions with a **Noun**, then a **verb**, then **details** (helps auto-completion & creates function families)

- `orsf_vi_negate()`, `orsf_vi_anova()`, `orsf_vi_permute()`
- `orsf_pd_ice()`, `orsf_pd_summary()`
- `orsf_control_cph()`, `orsf_control_net()`

Exception: the modeling function is just the model name:

- `lm()`
- `glm()`
- `orsf()`

Check arguments

Vet inputs and write error messages that spark joy

```
library(aorsf)

pbc_orsf$date_var <- lubridate::today()
pbc_orsf$char_var <- "I'm a character"

fit <- orsf(
 data = pbc_orsf,
 formula = Surv(time, status) ~ date_var + char_var
)
```

```
## Error: some variables have unsupported type:
##  <date_var> has type <Date>
##  <char_var> has type <character>
## supported types are numeric, integer, units, factor, and ordered
```

Generic methods

Include a generic `print` function (so your users won't hate you)

```
print(fit)
```

```
## ---------- Oblique random survival forest
##
##           N observations: 276
##                 N events: 111
##                  N trees: 500
##       N predictors total: 17
##    N predictors per node: 5
##  Average leaves per tree: 24
## Min observations in leaf: 5
##       Min events in leaf: 1
##           OOB stat value: 0.84
##            OOB stat type: Harrell's C-statistic
##
## ----------------------------------------
```

Generic methods

Include a generic `print` function (so your users won't hate you)

```
print(unclass(orsf_fit))
```

```
## $forest
## $forest[[1]]
## $forest[[1]]$leaf_nodes
##       [,1]      [,2]
##  [1,]   41 0.6000000
##  [2,]  131 0.4000000
##  [3,]  140 0.2000000
##  [4,] 1165 0.0000000
##  [5,]  334 0.0000000
##  [6,]  388 0.9230769
##  [7,] 2055 0.9615385
##  [8,] 2090 0.9423077
##  [9,] 1360 0.2500000
## [10,]  552 0.9285714
## [11,]  790 0.7142857
## [12,]  890 0.6428571
## [13,]  930 0.5625000
## [14,] 1427 0.1406250
```

Your guide to a friendly modeling package

click here to view this book from `tidymodels` online

Conventions for R Modeling Packages

draft version compiled on 2019-07-03

Chapter 1 Introduction

The S language has had unofficial conventions for modeling function, such as:

- using formulas to specify model terms, variable roles, and some statistical calculations

Thank you!

Incredible team members:

- `aorsf`: Sawyer Welden, Kristin Lenoir, Jaime L. Speiser, Matthew W. Segar, Ambarish Pandey, and Nicholas M. Pajewski
- today's talk: Jaime Lynn Speiser, Joseph Rigdon, Heather Marie Shappell, Nathaniel Sean O'Connell, and Michael Kattan.

Research reported in this presentation was supported[1] by

- Center for Biomedical Informatics, Wake Forest University School of Medicine.
- National Center for Advancing Translational Sciences (NCATS), National Institutes of Health, through Grant Award Number UL1TR001420.

[1] The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH.