# Why do we may need double dispatch?

In most cases, when writing R scripts or even creating R packages, it is enough to use standard functions or S3 methods. However, there is one important field that forces us to consider **double dispatch** question: **arithmetic operators**.

Suppose we’d like to create a class, which fits the problem we’re currently working on. Let’s name such class **beer**.

<span class="n">beer</span><span class="w"> </span><span class="o">

<-</span><span class="w"> </span><span class="k">function</span><span class="p">(</span><span class="n">type</span><span class="p">)

{</span><span class="w">

</span><span class="n">structure</span><span class="p">(</span><span class="nf">list</span><span class="p">(</span><span class="n">type</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="n">type</span><span class="p">),</span><span class="n">class</span><span class="w">

</span><span class="o">=</span><span class="w"> </span><span class="s2">"beer"</span><span class="p">)</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="n">opener</span><span class="w"> </span><span class="o"><-</span><span class="w"> </span><span class="k">function</span><span class="p">(){</span><span class="w">

</span><span class="n">structure</span><span class="p">(</span><span class="nf">list</span><span class="p">(),</span><span class="w">

</span><span class="n">class</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="s2">"opener"

</span><span class="p">)</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="n">pilsner</span><span class="w"> </span><span class="o"><-</span><span class="w"> </span><span class="n">beer</span><span class="p">(</span><span class="s2">"pilnser"

</span><span class="p">)</span><span class="w">

</span><span class="n">my\_opener</span><span class="w"> </span><span class="o"><-</span><span class="w"> </span><span class="n">opener</span><span class="p">()</span><span class="w">

</span>

Then, we create an operator which defines some non-standard behaviour.

if we add an opener to the beer, we get an **opened\_beer**. adding a **numeric** *x*, we get a case of beers (which even contain a negative number of bees, i.e. our owe…)

if second argument is different than a or **opener** or **numeric**, we get… untouched beer

Let’s demonstrate, how does it work:

<span class="n">`+.beer`</span><span class="w"> </span><span class="o">

<-</span><span class="w"> </span><span class="k">function</span><span class="p">(</span><span class="n">a</span><span class="p">,</span><span class="w"> </span><span class="n">b</span><span class="p">)

{</span><span class="w">

</span><span class="k">if</span><span class="w"> </span><span class="p">(</span><span class="n">inherits</span><span class="p"> (</span><span class="n">b</span><span class="p">,</span><span class="w"> </span><span class="s2">"opener"</span><span class="p">))

</span><span class="w"> </span><span class="p">{</span><span class="w">

</span><span class="nf">return</span><span class="p"> (</span><span class="n">structure</span><span class="p">(</span><span class="nf">list</span><span class="p">(</span><span class="w">

</span><span class="n">name</span><span class="w">

</span><span class="o">=</span><span class="w"> </span><span class="n">paste</span><span class="p">(</span><span class="s2">"opened"

</span><span class="p">,</span><span class="w"> </span><span class="n">a</span><span class="o">$</span><span class="n">name</span><span class="p">)</span><span class="w">

</span><span class="p">),</span><span class="w"> </span><span class="n">class</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="s2">"opened\_beer"</ span><span class="p">))</span><span class="w">

</span><span class="p">}</span><span class="w"> </span><span class="k">else</span><span class="w"> </span><span class="k">if</span><span class="w"> </span><span class="p"> (</span><span class="n">inherits</span><span class="p">(</span><span class="n">b</span><span class="p">,</span><span class="w"> </span><span class="s2">"numeric"</span><span class="p">))</span><span class="w">

</span><span class="p">{</span><span class="w">

</span><span class="n">print</span><span class="p">(</span><span class="s2">"It's magic! You've got a case of beers!"</span><span class="p">)</span><span class="w">

</span><span class="nf">return</span><span class="p">(</span><span class="n">structure</span><span class="p">(</span><span class="nf">list</span><span class="p">(</span><span class="w">

</span><span class="n">n\_beers</span><span class="w">

</span><span class="o">=</span><span class="w"> </span><span class="m">1</span><span class="w"> </span><span class="o">+</span><span class="w"> </span><span class="n">b</span><span class="w">

</span><span class="p">),</span><span class="w"> </span><span class="n">class</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="s2">"case\_of\_beers"</ span><span class="p">))</span><span class="w">

</span><span class="p">}</span><span class="w"> </span><span class="k">else</span><span class="w"> </span><span class="p">

{</span><span class="w">

</span><span class="nf">return</span><span class="p">(</span><span class="n">a</span><span class="p">)</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="p">}</span><span class="w">

</span>

<span class="n">pilsner</span><span class="w"> </span><span class="o">+

</span><span class="w"> </span><span class="n">my\_opener</span><span class="w">

</span>

## $name

## [1] "opened " ##

## attr(,"class") ## [1] "opened\_beer"

<span class="n">pilsner</span><span class="w"> </span><span class="o">+

</span><span class="w"> </span><span class="m">-0.1</span><span class="w">

</span>

## [1] "It's magic! You've got a case of beers!"

## $n\_beers ## [1] 0.9 ##

## attr(,"class")

## [1] "case\_of\_beers"

Don’t you think, that such operations should be **commutative**?

<span class="n">my\_opener</span><span class="w"> </span><span class="o">+</span><span class="w"> </span><span class="n">pilsner</span><span class="w">

</span>

## list()

## attr(,"class")

## [1] "opener"

What did happen here? This is an example of the way the R interpreter handles arithmetic operator.

Briefly speaking, in this particular case R engine matched method to the second argument (not to the first one), because there is no +.opener S3 method. What about such trick:

<span class="n">`+.opener`</span><span class="w"> </span><span class="o"><-</span><span class="w"> </span><span class="k">function</span><span class="p">(</span><span class="n">a</span><span class="p">,</span><span class="w"> </span><span class="n">b</span><span class="p">)</span><span class="w"> </span><span class="n">b</span><span class="w"> </span><span class="o">+</span><span class="w"> </span><span class="n">a</span><span class="w">

</span>

After that, the result is different:

<span class="n">my\_opener</span><span class="w"> </span><span class="o">+</span><span class="w"> </span><span class="n">pilsner</span><span class="w">

</span>

## Warning: Incompatible methods ("+.opener", "+.beer") for "+"

## Error in my\_opener + pilsner: non-numeric argument to binary operator

We crashed our function call. When both objects have the + method defined and these methods are not the same, R is trying to resolve the conflict by applying an internal +. It obviously cannot work. This

case could be easily solved using more ‘ifs’ in the +.beer beer function body. But let’s face a different situation.

<span class="m">-0.1</span><span class="w"> </span><span class="o">+

</span><span class="w"> </span><span class="n">pilsner</span><span class="w">

</span>

## [1] -0.1

What a mess! Simple S3 methods are definitely not the best solution when we need the double dispatch.

# S4 class: a classic approach

To civilize such code, we can use classic R approach, S4 methods. We’ll start from S4 classes declaration.

<span class="n">.S4\_beer</span><span class="w"> </span><span class="o"><-</span><span class="w"> </span><span class="n">setClass</span><span class="p">(</span><span class="s2">"S4\_beer"</span><span class="p">,</span><span class="w">

</span><span class="n">representation</span><span class="p"> (</span><span class="n">type</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="s2">"character"

</span><span class="p">))</span><span class="w">

</span><span class="n">.S4\_opened\_beer</span><span class="w">

</span><span class="o"><-</span><span class="w"> </span><span class="n">setClass</span><span class="p">(</span><span class="s2">"S4\_opened\_beer"</span><span class="p">,</span><span class="w"> </span><span class="n">representation</span><span class="p"> (</span><span class="n">type</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="s2">"character"

</span><span class="p">))</span><span class="w">

</span><span class="n">.S4\_opener</span><span class="w">

</span><span class="o"><-</span><span class="w"> </span><span class="n">setClass</span><span class="p">(</span><span class="s2">"S4\_opener"</span><span class="p">,</span><span class="w">

</span><span class="n">representation</span><span class="p"> (</span><span class="n">ID</span><span class="w"> </span><span

class="o">=</span><span class="w"> </span><span class="s2">"numeric"

</span><span class="p">))</span><span class="w">

</span><span class="n">.S4\_case\_of\_beers</span><span class="w">

</span><span class="o"><-</span><span class="w"> </span><span class="n">setClass</span><span class="p">(</span><span class="s2">"S4\_case\_of\_beers"</span><span class="p">,</span><span class="w"> </span><span class="n">representation</span><span class="p"> (</span><span class="n">n\_beers</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="s2">"numeric"

</span><span class="p">))</span><span class="w">

</span>

Then, we can two otptions, how to handle + operators. I didn’t mention about it in the previous example, but both S3 and S4 operators are grouped as so-called **group generic functions** (learn more:

**S3**, **S4**).

We can set a S4 method for a single operator and that looks as follows:

<span class="n">setMethod</span><span class="p">(</span><span class="s2">"+"</span><span class="p">,</span><span class="w">

</span><span class="nf">c</span><span class="p">(</span><span class="n">e1</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="s2">"S4\_beer"</span><span class="p">,</span><span class="w"> </span><span class="n">e2</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="s2">"S4\_opener"</span><span class="p">),</span><span class="w">

</span><span class="k">function</span><span class="p"> (</span><span class="n">e1</span><span class="p">,</span><span class="w"> </span><span class="n">e2</span><span class="p">)

{</span><span class="w">

</span><span class="k">if</span><span class="w"> </span><span class="p">(</span><span class="n">inherits</span><span class="p"> (</span><span class="n">e2</span><span class="p">,</span><span class="w"> </span><span class="s2">"S4\_opener"</span><span class="p">))

</span><span class="w"> </span><span class="p">{</span><span class="w">

</span><span class="nf">return</span><span class="p"> (</span><span class="n">.S4\_opened\_beer</span><span class="p"> (</span><span class="n">type</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="n">paste</span><span class="p">(</span><span class="s2">"opened"

</span><span class="p">,</span><span class="w"> </span><span class="n">e1</span><span class="o">@</span><span class="n">type</span><span class="p">)))</span><span class="w">

</span><span class="p">}</span><span class="w"> </span><span class="k">else</span><span class="w"> </span><span class="k">if</span><span class="w"> </span><span class="p"> (</span><span class="n">inherits</span><span class="p">(</span><span class="n">e2</span><span class="p">,</span><span class="w">

</span><span class="s2">"numeric"</span><span class="p">))</span><span

class="w"> </span><span class="p">{</span><span class="w">

</span><span class="n">print</span><span class="p">(</span><span class="s2">"It's magic! You've got a case of beers!"</span><span class="p">)</span><span class="w">

</span><span class="nf">return</span><span class="p">(</span><span class="n">.S4\_case\_of\_beers</span><span class="p">(</span><span class="n">n\_beers</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="m">1</span><span class="w">

</span><span class="o">+</span><span class="w"> </span><span class="n">e2</span><span class="p">))</span><span class="w">

</span><span class="p">}</span><span class="w"> </span><span class="k">else</span><span class="w"> </span><span class="p">

{</span><span class="w">

</span><span class="nf">return</span><span class="p">(</span><span class="n">e1</span><span class="p">)</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="p">})</span><span class="w">

</span><span class="n">setMethod</span><span class="p">(</span><span class="s2">"+"</span><span class="p">,</span><span class="w">

</span><span class="nf">c</span><span class="p">(</span><span class="n">e1</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="s2">"S4\_opener"</span><span class="p">,</span><span class="w"> </span><span class="n">e2</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="s2">"S4\_beer"</span><span class="p">),</span><span class="w">

</span><span class="k">function</span><span class="p"> (</span><span class="n">e1</span><span class="p">,</span><span class="w"> </span><span class="n">e2</span><span class="p">)

</span><span class="w"> </span><span class="n">e2</span><span class="w"> </span><span class="o">+</span><span class="w"> </span><span class="n">e1</span><span class="p">)</span><span class="w">

</span>

Alternatively, we can define a method for Arith geneneric and check, what method is exactly called at the moment. I decided to use the second approach, because it’s more similar to the way the double dispatch is implemented in the **vctrs** library.

<span class="n">.S4\_fun</span><span class="w"> </span><span class="o">

<-</span><span class="w"> </span><span class="k">function</span><span class="p">(</span><span class="n">e1</span><span class="p">,

</span><span class="w"> </span><span class="n">e2</span><span class="p">){</span><span class="w">

</span><span class="k">if</span><span class="w"> </span><span class="p">(</span><span class="n">inherits</span><span class="p"> (</span><span class="n">e2</span><span class="p">,</span><span class="w"> </span><span class="s2">"S4\_opener"</span><span class="p">))

</span><span class="w"> </span><span class="p">{</span><span class="w">

</span><span class="nf">return</span><span class="p"> (</span><span class="n">.S4\_opened\_beer</span><span class="p">

(</span><span class="n">type</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="n">paste</span><span class="p">(</span><span class="s2">"opened"

</span><span class="p">,</span><span class="w"> </span><span class="n">e1</span><span class="o">@</span><span class="n">type</span><span class="p">)))</span><span class="w">

</span><span class="p">}</span><span class="w"> </span><span class="k">else</span><span class="w"> </span><span class="k">if</span><span class="w"> </span><span class="p"> (</span><span class="n">inherits</span><span class="p">(</span><span class="n">e2</span><span class="p">,</span><span class="w">

</span><span class="s2">"numeric"</span><span class="p">))</span><span class="w"> </span><span class="p">{</span><span class="w">

</span><span class="n">print</span><span class="p">(</span><span class="s2">"It's magic! You've got a case of beers!"</span><span class="p">)</span><span class="w">

</span><span class="nf">return</span><span class="p">(</span><span class="n">.S4\_case\_of\_beers</span><span class="p">(</span><span class="n">n\_beers</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="m">1</span><span class="w">

</span><span class="o">+</span><span class="w"> </span><span class="n">e2</span><span class="p">))</span><span class="w">

</span><span class="p">}</span><span class="w"> </span><span class="k">else</span><span class="w"> </span><span class="p">

{</span><span class="w">

</span><span class="nf">return</span><span class="p">(</span><span class="n">e1</span><span class="p">)</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="n">setMethod</span><span class="p">(</span><span class="s2">"Arith"</span><span class="p">,</span><span class="w">

</span><span class="nf">c</span><span class="p">(</span><span class="n">e1</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="s2">"S4\_beer"</span><span class="p">,</span><span class="w"> </span><span class="n">e2</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="s2">"S4\_opener"</span><span class="p">),</span><span class="w">

</span><span class="k">function</span><span class="p"> (</span><span class="n">e1</span><span class="p">,</span><span class="w"> </span><span class="n">e2</span><span class="p">)

</span><span class="w">

</span><span class="p">{</span><span class="w">

</span><span class="n">op</span><span class="w">

</span><span class="o">=</span><span class="w"> </span><span class="n">.Generic</span><span class="p">[[</span><span class="m">1</span><span class="p">]]</span><span class="w">

</span><span class="nf">switch</span><span class="p"> (</span><span class="n">op</span><span class="p">,</span><span class="w">

</span><span class="n">`+`</span><span class="w">

</span><span class="o">=</span><span class="w"> </span><span class="n">.S4\_fun</span><span class="p">(</span><span class="n">e1</span><span class="p">,</span><span class="w">

</span><span class="n">e2</span><span class="p">),</span><span class="w">

</span><span class="n">stop</span><span class="p"> (</span><span class="s2">"undefined operation"</span><span class="p">)

</span><span class="w">

</span><span class="p">)</span><span class="w">

</span><span class="p">})</span><span class="w">

</span><span class="n">setMethod</span><span class="p">(</span><span class="s2">"Arith"</span><span class="p">,</span><span class="w">

</span><span class="nf">c</span><span class="p">(</span><span class="n">e1</span><span class="o">=</span><span class="s2">"S4\_opener"

</span><span class="p">,</span><span class="w"> </span><span class="n">e2</span><span class="o">=</span><span class="s2">"S4\_beer"

</span><span class="p">),</span><span class="w">

</span><span class="k">function</span><span class="p"> (</span><span class="n">e1</span><span class="p">,</span><span class="w"> </span><span class="n">e2</span><span class="p">)

</span><span class="w">

</span><span class="p">{</span><span class="w">

</span><span class="n">op</span><span class="w">

</span><span class="o">=</span><span class="w"> </span><span class="n">.Generic</span><span class="p">[[</span><span class="m">1</span><span class="p">]]</span><span class="w">

</span><span class="nf">switch</span><span class="p"> (</span><span class="n">op</span><span class="p">,</span><span class="w">

</span><span class="n">`+`</span><span class="w">

</span><span class="o">=</span><span class="w"> </span><span class="n">e2</span><span class="w"> </span><span class="o">+

</span><span class="w"> </span><span class="n">e1</span><span class="p">,</span><span class="w">

</span><span class="n">stop</span><span class="p"> (</span><span class="s2">"undefined operation"</span><span class="p">)

</span><span class="w">

</span><span class="p">)</span><span class="w">

</span><span class="p">})</span><span class="w">

</span>

Let’s create our class instances and do a piece of math.

<span class="n">S4\_pilsner</span><span class="w"> </span><span class="o"><-</span><span class="w"> </span><span class="n">.S4\_beer</span><span class="p">(</span><span class="n">type</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="s2">"Pilsner"</span><span class="p">)</span><span class="w">

</span><span class="n">S4\_opener</span><span class="w"> </span><span class="o"><-</span><span class="w"> </span><span

class="n">.S4\_opener</span><span class="p">(</span><span class="n">ID</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="m">1</span><span class="p">)</span><span class="w">

</span>

<span class="n">S4\_pilsner</span><span class="w"> </span><span class="o">+</span><span class="w"> </span><span class="n">S4\_opener</span><span class="w">

</span>

## An object of class "S4\_opened\_beer" ## Slot "type":

## [1] "opened Pilsner"

<span class="n">S4\_opener</span><span class="w"> </span><span class="o">+</span><span class="w"> </span><span class="n">S4\_pilsner</span><span class="w">

</span>

## An object of class "S4\_opened\_beer" ## Slot "type":

## [1] "opened Pilsner"

Declared methods are clear, and, the most important: they work correctly.

# vctrs library: a tidyverse approach

**vctrs** is an interesting library,

thought as a remedy for a couple of R disadvantages. It delivers, among others, a custom double-dispatch system based on well-known S3 mechanism.

At the first step we declare class ‘constructors’.

<span class="n">library</span><span class="p">(</span><span class="n">vctrs</span><span class="p">)</span><span class="w">

</span><span class="n">.vec\_beer</span><span class="w"> </span><span class="o"><-</span><span class="w"> </span><span class="k">function</span><span class="p">(</span><span class="n">type</span><span class="p">){</span><span class="w">

</span><span class="n">new\_vctr</span><span class="p">(</span><span class="n">.data</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="nf">list</span><span class="p">(</span><span class="n">type</span><span class="w">

</span><span class="o">=</span><span class="w"> </span><span class="n">type</span><span class="p">),</span><span class="w">

</span><span class="n">class</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="s2">"vec\_beer"

</span><span class="p">)</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="n">.vec\_opened\_beer</span><span class="w">

</span><span class="o"><-</span><span class="w"> </span><span class="k">function</span><span class="p">(</span><span class="n">type</span><span class="p">){</span><span class="w">

</span><span class="n">new\_vctr</span><span class="p">(</span><span class="n">.data</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="nf">list</span><span class="p">(</span><span class="n">type</span><span class="w">

</span><span class="o">=</span><span class="w"> </span><span class="n">type</span><span class="p">),</span><span class="w">

</span><span class="n">class</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="s2">"vec\_opened\_beer"</span><span class="p">)</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="n">.vec\_case\_of\_beers</span><span class="w">

</span><span class="o"><-</span><span class="w"> </span><span class="k">function</span><span class="p">(</span><span class="n">n\_beers</span><span class="p">){</span><span class="w">

</span><span class="n">new\_vctr</span><span class="p">(</span><span class="n">.data</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="nf">list</span><span class="p">(</span><span class="n">n\_beers</span><span class="w">

</span><span class="o">=</span><span class="w"> </span><span class="n">n\_beers</span><span class="p">),</span><span class="w">

</span><span class="n">class</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="s2">"vec\_case\_of\_beers"</span><span class="p">)</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="n">.vec\_opener</span><span class="w"> </span><span class="o"><-</span><span class="w"> </span><span class="k">function</span><span class="p">(){</span><span class="w">

</span><span class="n">new\_vctr</span><span class="p">(</span><span class="n">.data</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="nf">list</span><span class="p">(),</span><span class="w"> </span><span class="n">class</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="s2">"vec\_opener"</span>

<span class="p">)</span><span class="w">

</span><span class="p">}</span><span class="w">

</span>

Then, we create class instances.

<span class="n">vec\_pilsner</span><span class="w"> </span><span class="o"><-</span><span class="w"> </span><span class="n">.vec\_beer</span><span class="p">(</span><span class="s2">"pilnser"</span><span class="p">)</span><span class="w">

</span><span class="n">vec\_opener</span><span class="w"> </span><span

class="o"><-</span><span class="w"> </span><span class="n">.vec\_opener</span><span class="p">()</span><span class="w">

</span><span class="n">print</span><span class="p">(</span><span class="nf">class</span><span class="p">(</span><span class="n">vec\_pilsner</span><span class="p">))</span><span class="w">

</span>

## [1] "vec\_beer" "vctrs\_vctr"

<span class="n">print</span><span class="p">(</span><span class="nf">class</span><span class="p">(</span><span class="n">vec\_opener</span><span class="p">))</span><span class="w">

</span>

## [1] "vec\_opener" "vctrs\_vctr"

At the end, we write a double-dispatched methods **in vctrs style**. As you can see,

<span class="n">.fun</span><span class="w"> </span><span class="o">

<-</span><span class="w"> </span><span class="k">function</span><span class="p">(</span><span class="n">a</span><span class="p">,</span><span class="w"> </span><span class="n">b</span><span class="p">)

{</span><span class="w">

</span><span class="k">if</span><span class="w"> </span><span class="p">(</span><span class="n">inherits</span><span class="p"> (</span><span class="n">b</span><span class="p">,</span><span class="w"> </span><span class="s2">"vec\_opener"</span><span class="p">))</span><span class="w"> </span><span class="p">

{</span><span class="w">

</span><span class="nf">return</span><span class="p"> (</span><span class="n">.vec\_opened\_beer</span><span class="p"> (</span><span class="n">type</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="n">paste</span><span class="p">(</span><span class="s2">"opened"

</span><span class="p">,</span><span class="w"> </span><span class="n">a</span><span class="o">$</span><span class="n">type</span><span class="p">)))</span><span class="w">

</span><span class="p">}</span><span class="w"> </span><span class="k">else</span><span class="w"> </span><span class="k">if</span><span class="w"> </span><span class="p"> (</span><span class="n">inherits</span><span class="p">(</span><span class="n">b</span><span class="p">,</span><span class="w"> </span><span class="s2">"numeric"</span><span class="p">))</span><span class="w">

</span><span class="p">{</span><span class="w">

</span><span class="n">print</span><span class="p">(</span><span class="s2">"It's magic! You've got a case of beers!"</span><span class="p">)</span><span class="w">

</span><span class="nf">return</span><span class="p">(</span><span class="n">.vec\_case\_of\_beers</span><span class="p">(</span><span class="n">n\_beers</span><span class="w"> </span><span class="o">=

</span><span class="w"> </span><span class="m">1</span><span class="w">

</span><span class="o">+</span><span class="w"> </span><span

class="n">b</span><span class="p">))</span><span class="w">

</span><span class="p">}</span><span class="w"> </span><span class="k">else</span><span class="w"> </span><span class="p">

{</span><span class="w">

</span><span class="nf">return</span><span class="p">(</span><span class="n">a</span><span class="p">)</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="n">vec\_arith.vec\_beer</span><span class="w">

</span><span class="o"><-</span><span class="w"> </span><span class="k">function</span><span class="p">(</span><span class="n">op</span><span class="p">,</span><span class="w">

</span><span class="n">x</span><span class="p">,</span><span class="w">

</span><span class="n">y</span><span class="p">,</span><span class="w">

</span><span class="n">...</span><span class="p">)</span><span class="w"> </span><span class="p">{</span><span class="w">

</span><span class="nf">UseMethod</span><span class="p">(</span><span class="s2">"vec\_arith.vec\_beer"</span><span class="p">,</span><span class="w"> </span><span class="n">y</span><span class="p">)</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="n">vec\_arith.vec\_opener</span><span class="w">

</span><span class="o"><-</span><span class="w"> </span><span class="k">function</span><span class="p">(</span><span class="n">op</span><span class="p">,</span><span class="w">

</span><span class="n">x</span><span class="p">,</span><span class="w">

</span><span class="n">y</span><span class="p">,</span><span class="w">

</span><span class="n">...</span><span class="p">)</span><span class="w"> </span><span class="p">{</span><span class="w">

</span><span class="nf">UseMethod</span><span class="p">(</span><span class="s2">"vec\_arith.vec\_opener"</span><span class="p">,</span><span class="w"> </span><span class="n">y</span><span class="p">)</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="n">vec\_arith.vec\_beer.vec\_opener</span><span class="w"> </span><span class="o"><-</span><span class="w">

</span><span class="k">function</span><span class="p">(</span><span class="n">op</span><span class="p">,</span><span class="w">

</span><span class="n">x</span><span class="p">,</span><span class="w">

</span><span class="n">y</span><span class="p">,</span><span class="w">

</span><span class="n">...</span><span class="p">){</span><span class="w">

</span><span class="nf">switch</span><span class="p">(</span><span class="n">op</span><span class="p">,</span><span class="w">

</span><span class="n">`+`</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="n">.fun</span><span class="p">(</span><span class="n">x</span><span class="p">,</span><span class="w"> </span><span class="n">y</span><span class="p">),</span><span class="w">

</span><span class="n">stop\_incompatible\_op</span><span class="p">(</span><span class="n">op</span><span class="p">,

</span><span class="w"> </span><span class="n">x</span><span class="p">,</span><span class="w"> </span><span class="n">y</span><span class="p">)</span><span class="w">

</span><span class="p">)</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="n">vec\_arith.vec\_opener.vec\_beer</span><span class="w"> </span><span class="o"><-</span><span class="w">

</span><span class="k">function</span><span class="p">(</span><span class="n">op</span><span class="p">,</span><span class="w">

</span><span class="n">x</span><span class="p">,</span><span class="w">

</span><span class="n">y</span><span class="p">,</span><span class="w">

</span><span class="n">...</span><span class="p">){</span><span class="w">

</span><span class="n">y</span><span class="w"> </span><span class="o">+</span><span class="w"> </span><span class="n">x</span><span class="w">

</span><span class="p">}</span><span class="w">

</span><span class="n">vec\_pilsner</span><span class="w"> </span><span class="o">+</span><span class="w"> </span><span class="n">vec\_opener</span><span class="w">

</span>

## <vec\_opened\_beer[1]> ## type

## opened pilnser

<span class="n">vec\_opener</span><span class="w"> </span><span class="o">+</span><span class="w"> </span><span class="n">vec\_pilsner</span><span class="w">

</span>

## <vec\_opened\_beer[1]> ## type

## opened pilnser

It works properly, too.

# Benchmark

I’ve created all the classes and methods above not only to demonstate, how to implement double dispatch in R. My main goal is to benchmark both approaches and check, which one has smaller overhead. The hardware I used for the test looks as follows:

## $vendor\_id

## [1] "GenuineIntel" ##

## $model\_name

## [1] "Intel(R) Core(TM) i3 CPU M 350 @ 2.27GHz"

##

## $no\_of\_cores ## [1] 4

## 8.19 GB

<span class="n">sessionInfo</span><span class="p">()</span><span class="w">

</span>

## R version 3.6.1 (2019-07-05)

## Platform: x86\_64-pc-linux-gnu (64-bit) ## Running under: Ubuntu 18.04.2 LTS

##

## Matrix products: default

## BLAS: /usr/local/lib/R/lib/libRblas.so ## LAPACK: /usr/local/lib/R/lib/libRlapack.so ##

## locale:

## [1] LC\_CTYPE=pl\_PL.UTF-8 LC\_NUMERIC=C

## [3] LC\_TIME=pl\_PL.UTF-8 LC\_COLLATE=pl\_PL.UTF-8 ## [5] LC\_MONETARY=pl\_PL.UTF-8 LC\_MESSAGES=en\_US.utf8 ## [7] LC\_PAPER=pl\_PL.UTF-8 LC\_NAME=C

## [9] LC\_ADDRESS=C LC\_TELEPHONE=C

## [11] LC\_MEASUREMENT=pl\_PL.UTF-8 LC\_IDENTIFICATION=C ##

## attached base packages:

## [1] stats graphics grDevices utils datasets methods base ##

## other attached packages:

## [1] vctrs\_0.2.3 ##

## loaded via a namespace (and not attached):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ## | [1] | Rcpp\_1.0.3 | benchmarkmeData\_1.0.3 | knitr\_1.23 |
| ## | [4] | magrittr\_1.5 | tidyselect\_0.2.5 | doParallel\_1.0.15 |
| ## | [7] | lattice\_0.20-38 | R6\_2.4.0 | rlang\_0.4.2 |
| ## | [10] | foreach\_1.4.7 | httr\_1.4.1 | stringr\_1.4.0 |
| ## | [13] | dplyr\_0.8.3 | tools\_3.6.1 | parallel\_3.6.1 |
| ## | [16] | grid\_3.6.1 | xfun\_0.9 | htmltools\_0.3.6 |
| ## | [19] | iterators\_1.0.12 | yaml\_2.2.0 | digest\_0.6.25 |
| ## | [22] | assertthat\_0.2.1 | tibble\_2.1.3 | benchmarkme\_1.0.3 |
| ## | [25] | crayon\_1.3.4 | Matrix\_1.2-17 | purrr\_0.3.3 |
| ## | [28] | codetools\_0.2-16 | glue\_1.3.1 | evaluate\_0.14 |
| ## | [31] | rmarkdown\_1.14 | stringi\_1.4.3 | pillar\_1.4.2 |
| ## | [34] | compiler\_3.6.1 | pkgconfig\_2.0.2 |  |

It’s my good old notebook, which is not a beast.

<span class="n">library</span><span class="p">(</span><span class="n">microbenchmark</span><span class="p">)</span><span class="w">

</span><span class="n">library</span><span class="p">(</span><span class="n">ggplot2</span><span class="p">)</span><span class="w">

</span>

## Beer + opener

<span class="n">bm1</span><span class="w"> </span><span class="o">

<-</span><span class="w"> </span><span class="n">microbenchmark</ span><span class="p">(</span><span class="w">

</span><span class="n">s4</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="n">S4\_pilsner</span><span class="w"> </span><span class="o">+

</span><span class="w"> </span><span class="n">S4\_opener</span><span class="p">,</span><span class="w">

</span><span class="n">s3\_vec</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="n">vec\_pilsner</span><span class="w"> </span><span class="o">+

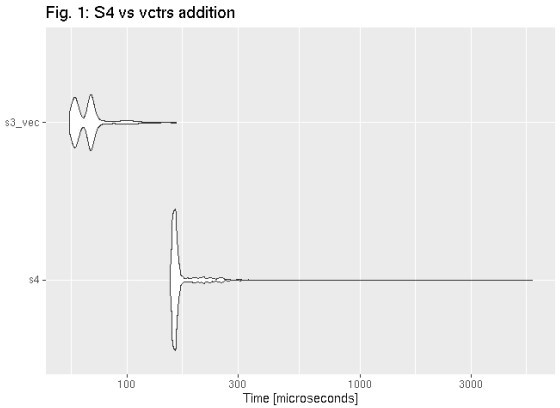
</span><span class="w"> </span><span class="n">vec\_opener</span><span class="p">,</span><span class="w">

</span><span class="n">times</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="m">1000</span><span class="w">

</span><span class="p">)</span><span class="w">

</span>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ##  ## | Unit: microseconds  expr min lq | mean | median | uq | max | neval |
| ## | s4 153.292 158.2120 | 178.40541 | 161.4225 | 165.6375 | 5506.681 | 1000 |
| ## | s3\_vec 56.686 60.1265 | 69.52364 | 68.9240 | 70.8830 | 163.278 | 1000 |



## Opener + beer

<span class="n">bm2</span><span class="w"> </span><span class="o">

<-</span><span class="w"> </span><span class="n">microbenchmark</ span><span class="p">(</span><span class="w">

</span><span class="n">s4</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="n">S4\_opener</span><span class="w"> </span><span class="o">+

</span><span class="w"> </span><span class="n">S4\_pilsner</span><span class="p">,</span><span class="w">

</span><span class="n">s3\_vec</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="n">vec\_opener</span><span class="w"> </span><span class="o">+

</span><span class="w"> </span><span class="n">vec\_pilsner</span><span class="p">,</span><span class="w">

</span><span class="n">times</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="m">1000</span><span class="w">

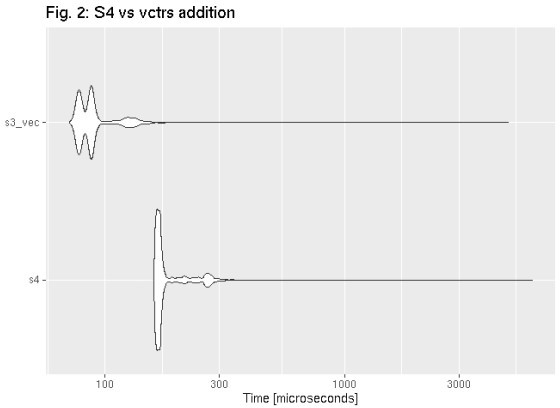
</span><span class="p">)</span><span class="w">

</span>

## Unit: microseconds

## expr min lq mean median uq max neval ## s4 159.512 164.6735 191.74781 168.9655 176.3165 6068.477 1000

## s3\_vec 71.110 78.5835 96.22535 86.6720 89.4015 4796.377 1000



## Bonus: opener + beer vs addtion of numerics

<span class="n">bm3</span><span class="w"> </span><span class="o">

<-</span><span class="w"> </span><span class="n">microbenchmark</ span><span class="p">(</span><span class="w">

</span><span class="n">simple\_R</span><span class="w"> </span><span

class="o">=</span><span class="w"> </span><span class="m">1</span><span class="w"> </span><span class="o">+</span><span class="w"> </span><span class="m">2</span><span class="p">,</span><span class="w">

</span><span class="n">s4</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="n">S4\_opener</span><span class="w"> </span><span class="o">+

</span><span class="w"> </span><span class="n">S4\_pilsner</span><span class="p">,</span><span class="w">

</span><span class="n">s3\_vec</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="n">vec\_opener</span><span class="w"> </span><span class="o">+

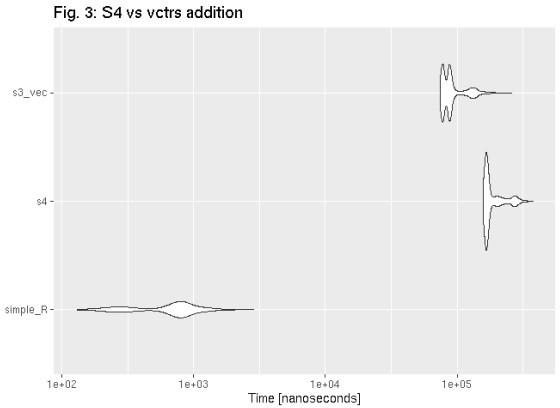
</span><span class="w"> </span><span class="n">vec\_pilsner</span><span class="p">,</span><span class="w">

</span><span class="n">times</span><span class="w"> </span><span class="o">=</span><span class="w"> </span><span class="m">1000</span><span class="w">

</span><span class="p">)</span><span class="w">

</span>

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ##  ## | Unit: nanoseconds  expr min | lq | mean | median | uq | max | neval |
| ## | simple\_R 130 | 344.0 | 697.49 | 744.5 | 857 | 2862 | 1000 |
| ## | s4 158769 | 164522.5 | 189297.35 | 169270.5 | 198120 | 375648 | 1000 |
| ## | s3\_vec 74775 | 78395.5 | 94786.28 | 87192.5 | 94085 | 258129 | 1000 |



# Conclusions

It seems that **vctrs-based** performs better than traditional **S4 methods**. Obviously, I checked only one operation and probably some

edge cases may exists. However, I think that it shows us some direction, what execution time we can expect.