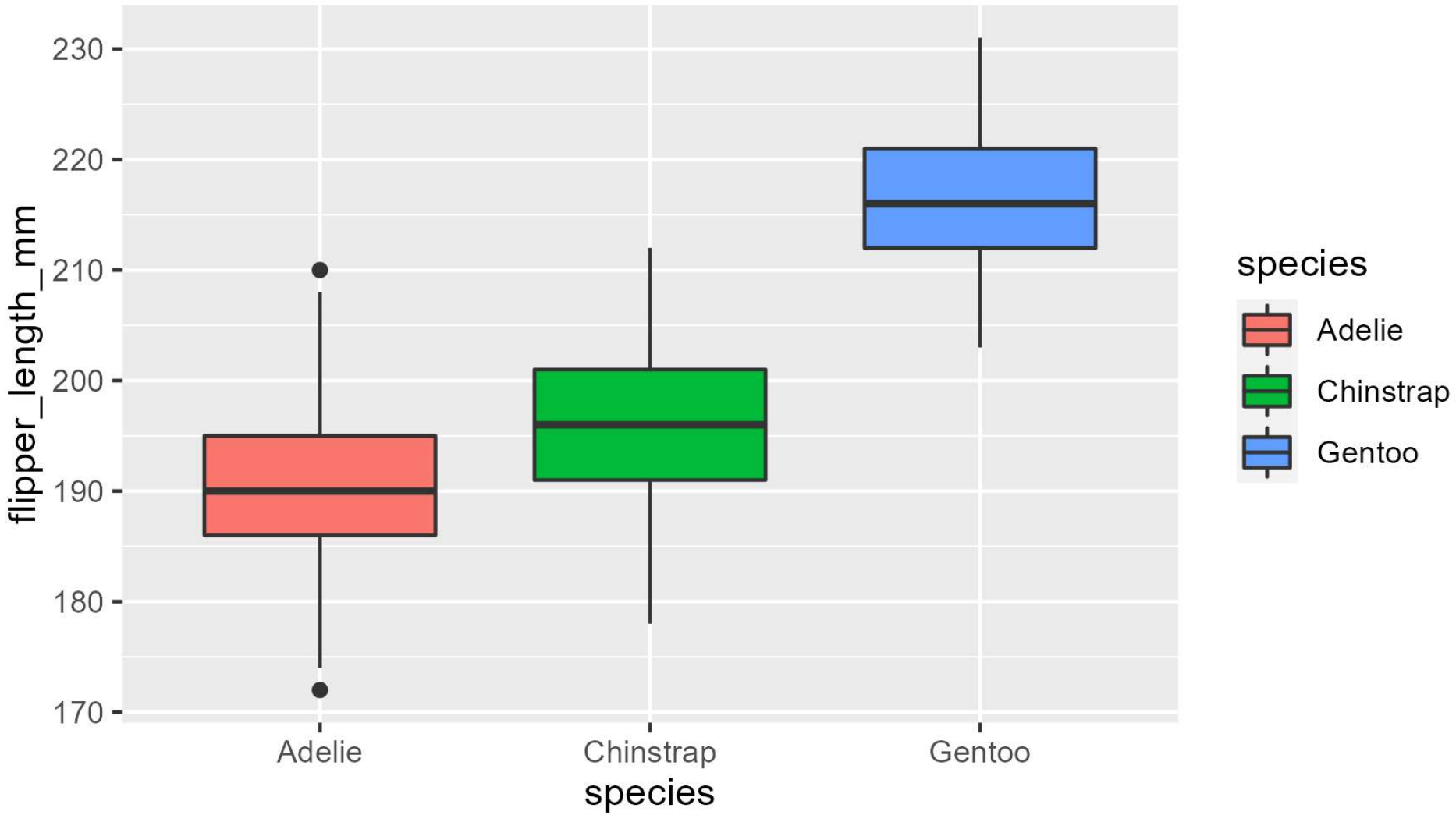
Stepping into {ggplot2} internals with {ggtrace}

June Choe

University of Pennsylvania

@yjunechoe

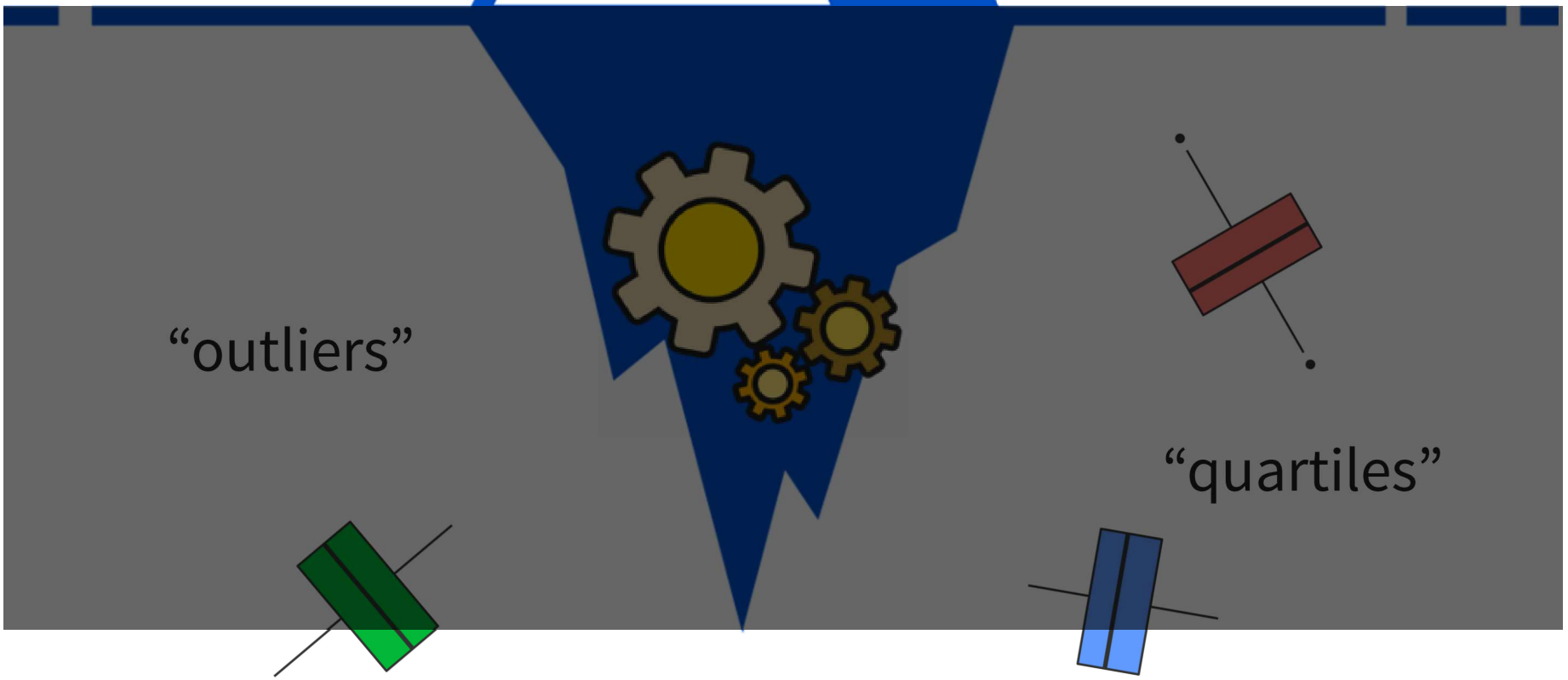
Penguin flipper lengths



User-facing code vs. Internals

```
ggplot(palmerpenguins::penguins,
       aes(species, flipper_length_mm)) +
  geom_boxplot(aes(fill = species), width = .7)
```

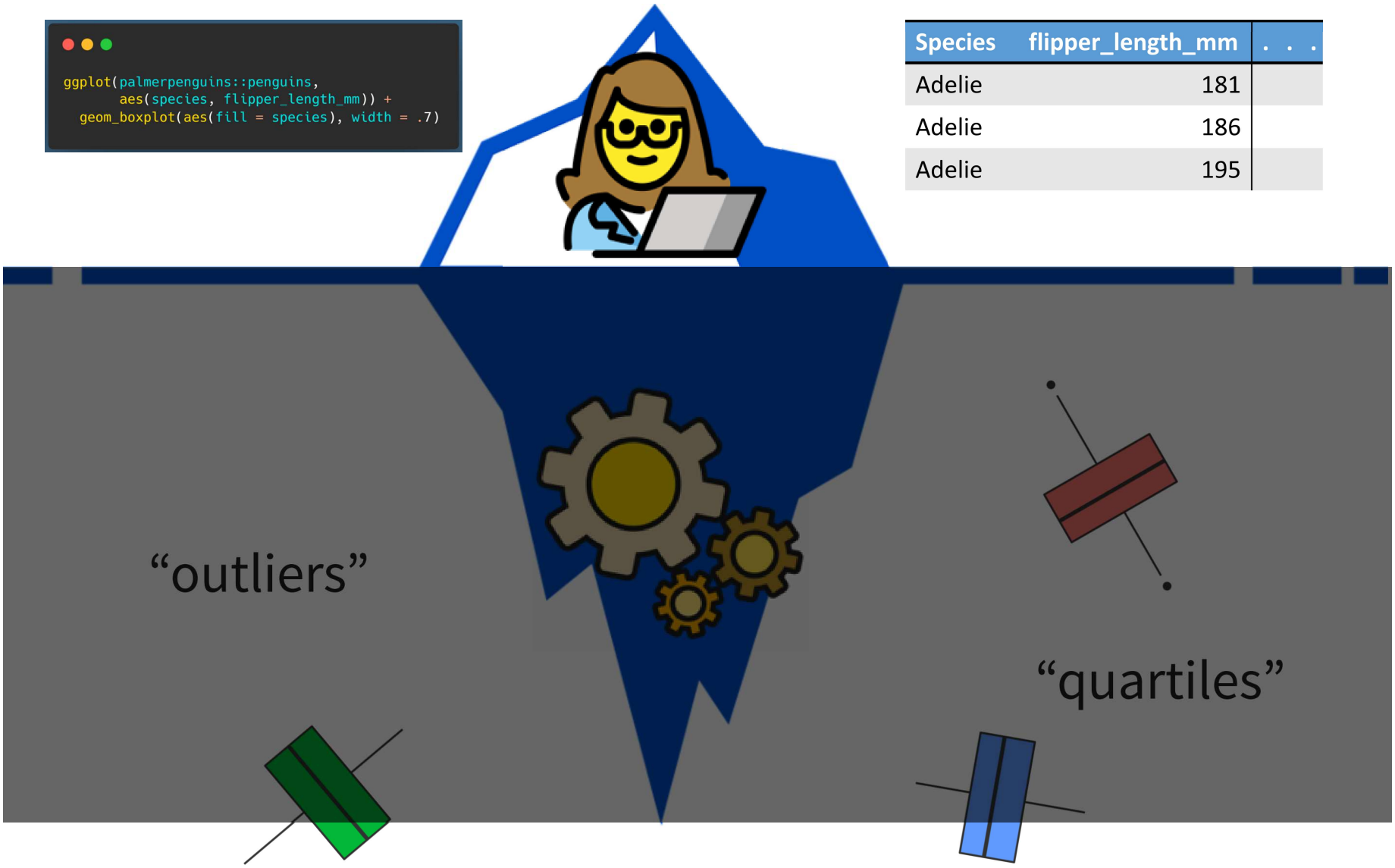
Species	flipper_length_mm	. . .
Adelie	181	
Adelie	186	
Adelie	195	



User-facing code vs. Internals

```
ggplot(palmerpenguins::penguins,
       aes(species, flipper_length_mm)) +
  geom_boxplot(aes(fill = species), width = .7)
```

Species	flipper_length_mm	. . .
Adelie	181	
Adelie	186	
Adelie	195	



{ggtrace}

Let’s interact with ggplot internals!

Outline

1. Introduction to ggplot internals

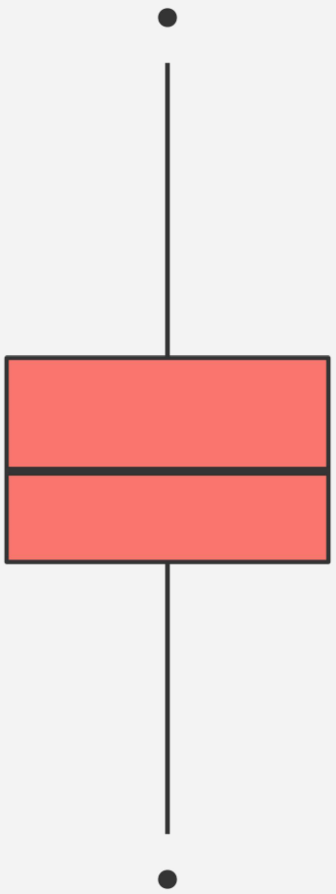
- Just knowing `{dplyr}` can get you very far!

2. A showcase of `{ggtrace}` features

- Gives users finer control over the plots they make
- Helps developers debug custom internal functions

{ggplot2} internals

Thinking like ggplot internals



outliers

whisker

3rd quartile

median

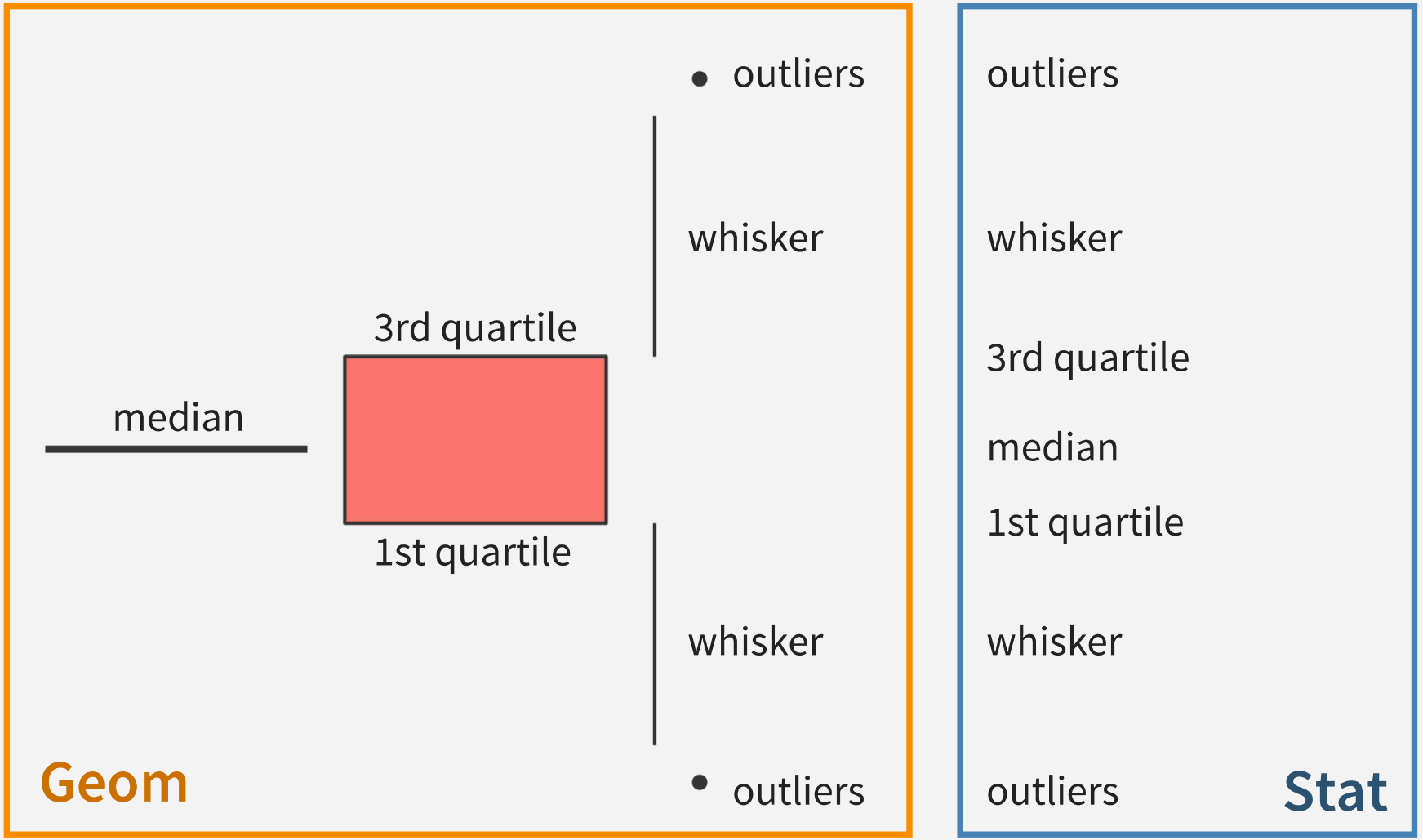
1st quartile

whisker

outliers

Stat

Thinking like ggplot internals



{ggplot2} Stat and Geom methods

Statistical transformation (**compute**)

```
Stat$compute_layer()
    Stat$compute_panel()
        Stat$compute_group()
```

Geometric transformation (**draw**)

```
Geom$draw_layer()
    Geom$draw_panel()
        Geom$draw_group()
```

{ggplot2} Stat and Geom methods

Statistical transformation (**compute**)

```
Stat$compute_layer()
    Stat$compute_panel()
        Stat$compute_group()
```

Geometric transformation (**draw**)

```
Geom$draw_layer()
    Geom$draw_panel()
        Geom$draw_group()
```

{ggplot2} Stat and Geom methods

Statistical transformation (**compute**)

```
Stat$compute_layer()
    Stat$compute_panel()
        Stat$compute_group()
```

Geometric transformation (**draw**)

```
Geom$draw_layer()
    Geom$draw_panel()
        Geom$draw_group()
```

{ggplot2} Stat and Geom methods

Statistical transformation (**compute**)

```
Stat$compute_layer()
    Stat$compute_panel()
        Stat$compute_group()
```

Geometric transformation (**draw**)

```
Geom$draw_layer()
    Geom$draw_panel()
        Geom$draw_group()
```

{ggplot2} Stat and Geom methods

Statistical transformation (**compute**)

```
Stat$compute_layer()
    Stat$compute_panel()
        Stat$compute_group()
```

Geometric transformation (**draw**)

```
Geom$draw_layer()
    Geom$draw_panel()
        Geom$draw_group()
```

The choice of specific `Stat` and `Geom` ggprotos are layer-specific

{dplyr} all the way down

```
ggplot_output <- data_input %>%

  ... # aesthetic mappings, x-y scale transformations, etc.

  group_by(layer) %>%
    group_by(panel) %>%
      group_by(group) %>%
        summarize( ... ) %>%

  ... # constructing plot layout with {grid} and {gtable}, etc.
```

{dplyr} all the way down

```
ggplot_output <- data_input %>%

  ... # aesthetic mappings, x-y scale transformations, etc.

  group_by(layer) %>%
    mutate( ... ) %>%
    group_by(panel) %>%
      mutate( ... ) %>%
      group_by(group) %>%
        mutate( ... ) %>%
        summarize( ... ) %>%
        mutate( ... ) %>%
      mutate( ... ) %>%
    mutate( ... ) %>%

  ... # constructing plot layout with {grid} and {gtable}, etc.
```

Motivating idea

```
ggplot_output <- data_input %>%

  ... # aesthetic mappings, x-y scale transformations, etc.

  group_by(layer) %>%
    mutate( ... ) %>%
    group_by(panel) %>%
      mutate( ... ) %>%
      # <-- What does the data for 2nd layer, 3rd panel look like at this point?
      group_by(group) %>%
        mutate( ... ) %>%
        summarize( ... ) %>%
        mutate( ... ) %>%
      mutate( ... ) %>%
    mutate( ... ) %>%

  ... # constructing plot layout with {grid} and {gtable}, etc.
```

Motivating idea

```
ggplot_output <- data_input %>%

  ... # aesthetic mappings, x-y scale transformations, etc.

  group_by(layer) %>%
    mutate( ... ) %>%
    group_by(panel) %>%
      mutate( ... ) %>%
      group_by(group) %>%
        mutate( ... ) %>%
        summarize( ... ) %>%
        my_fun() %>% # <-- What consequence does this have on the pipeline?
        mutate( ... ) %>%
      mutate( ... ) %>%
    mutate( ... ) %>%

  ... # constructing plot layout with {grid} and {gtable}, etc.
```

Inaccessibility of ggproto methods

They are essentially functions, but look weird:

```
class(StatBoxplot$compute_group)
```

```
[1] "ggproto_method"
```

```
typeof(StatBoxplot$compute_group)
```

```
[1] "closure"
```

You can't do any useful things with them interactively:

```
formals(StatBoxplot$compute_group)
```

```
$...
```

```
body(StatBoxplot$compute_group)
```

```
f(...)
```

Familiar debugging tools (ex: `debug()`) fail out of the box

Introducing {ggtrace}

Introducing `{ggtrace}`

Toolkit to **Inspect**, **Capture**, and **Highjack** the internals

Workflow functions `ggtrace_{action}_{value}()`:

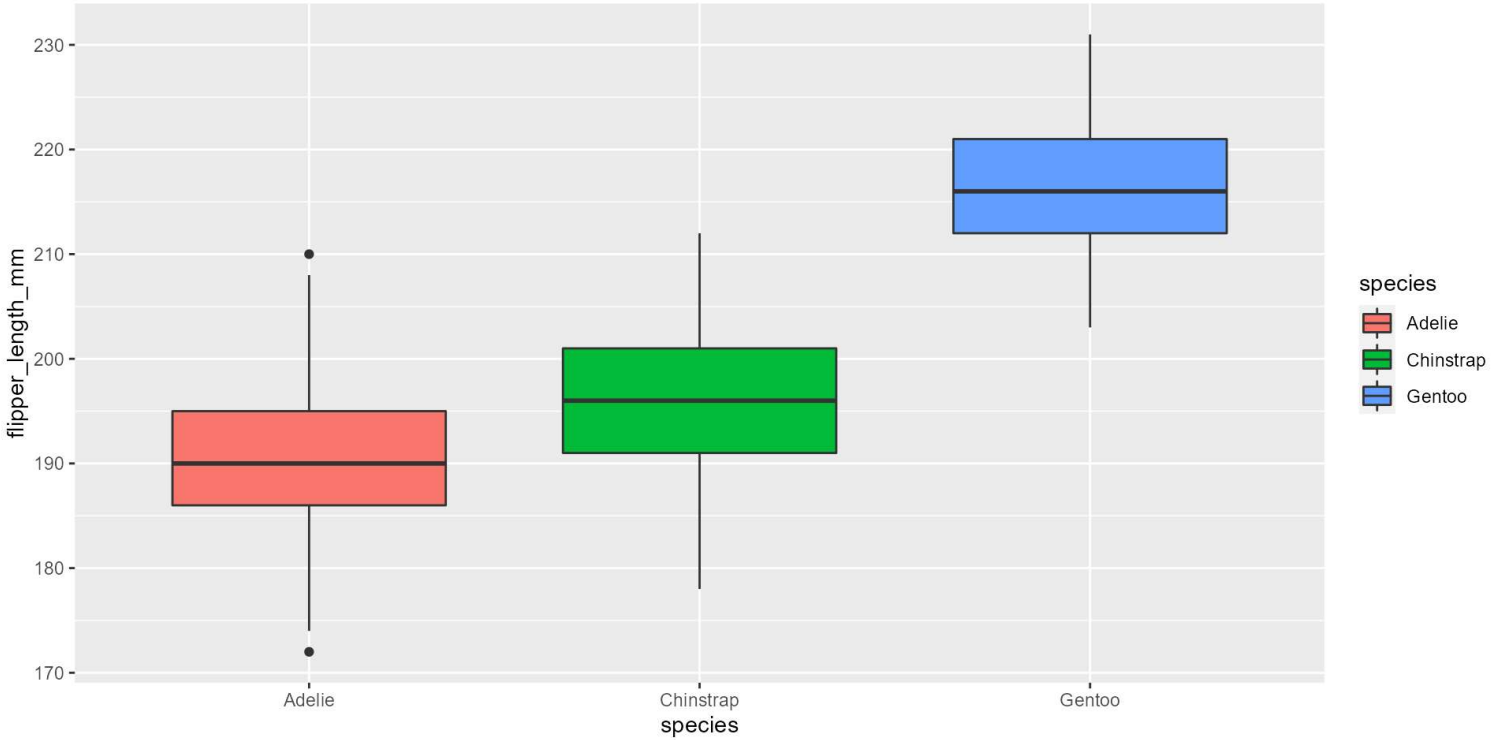
- `x`: The ggplot object
- `method`: The ggproto method
- `cond`: When to interact with the method[1]

"While rendering `x`, interact with `method` when `cond` is met"

1. If `cond` is integer *N*, targets the *N*th time `method` is called

Our boxplot

```
p <- palmerpenguins::penguins %>%
  filter(!is.na(flipper_length_mm)) %>%
  ggplot(aes(species, flipper_length_mm)) +
  geom_boxplot(aes(fill = species), width = .7)
p
```



`Inspect` workflow

Inspect `compute_layer()`

```
compute_layer_input <- ggtrace_inspect_args(x = p, method = Stat$compute_layer)
names(compute_layer_input)
```

```
[1] "self"   "data"   "params" "layout"
```

```
compute_layer_input$params
```

```
$na.rm
[1] FALSE

$orientation
[1] NA

$width
[1] 0.7

$flipped_aes
[1] FALSE
```

Inspect `compute_layer()`

```
compute_layer_input$data %>% tibble() %>% print(n = 3)
```

```
# A tibble: 342 × 5
  fill   x              y PANEL group
  <fct>  <mppd_dsc> <dbl> <fct> <int>
1 Adelie 1            181 1         1
2 Adelie 1            186 1         1
3 Adelie 1            195 1         1
# … with 339 more rows
```

```
compute_layer_input$data %>% count(PANEL, group)
```

```
  PANEL group   n
1     1     1 151
2     1     2  68
3     1     3 123
```

```
ggtrace_inspect_return(x = p, method = Stat$compute_layer)
```

```
  ymin lower middle upper ymax outliers notchupper notchlower x width relvarwidth
flipped_aes      fill PANEL group
1  174   186    190   195  208 172, 210   191.1572   188.8428 1   0.7   12.288206
FALSE    Adelie     1     1
2  178   191    196   201  212           197.9160   194.0840 2   0.7    8.246211
FALSE Chinstrap     1     2
```

Stat split-apply-combine design

```
ggtrace_inspect_n(p, Stat$compute_layer)
```

```
[1] 1
```

```
ggtrace_inspect_n(p, Stat$compute_panel)
```

```
[1] 1
```

```
ggtrace_inspect_n(p, StatBoxplot$compute_group)
```

```
[1] 3
```

```
ggtrace::get_method_inheritance( geom_boxplot()$stat )
```

```
$Stat
[1] "aesthetics" "compute_layer" "compute_panel"
"default_aes""finish_layer""optional_aes""parameters" "retransform"
$StatBoxplot
[1] "compute_group""extra_params" "non_missing_aes" "required_aes"
"setup_data""setup_params"
```

Stat split-apply-combine inputs

```
ggtrace_inspect_args(p, Stat$compute_layer)$data %>% head(2)
```

```
     fill x   y PANEL group
1 Adelie 1 181     1     1
2 Adelie 1 186     1     1
```

```
ggtrace_inspect_args(p, Stat$compute_panel)$data # same as `compute_layer()`
```

```
ggtrace_inspect_args(p, StatBoxplot$compute_group, 1)$data %>% head(2
```

```
     fill x   y PANEL group
1 Adelie 1 181     1     1
2 Adelie 1 186     1     1
```

```
ggtrace_inspect_args(p, StatBoxplot$compute_group, 2)$data %>% head(2
```

```
          fill x   y PANEL group
275 Chinstrap 2 192     1     2
276 Chinstrap 2 196     1     2
```

```
ggtrace_inspect_args(p, StatBoxplot$compute_group, 3)$data %>% head(2
```

```
       fill x   y PANEL group
152 Gentoo 3 211     1     3
153 Gentoo 3 230     1     3
```

Stat split-apply-combine outputs

```
ggtrace_inspect_return(p, StatBoxplot$compute_group, 1)
```

```
  ymin lower middle upper ymax outliers notchupper notchlower x width
relvarwidth flipped_aes
1  174   186    190   195  208 172, 210   191.1572   188.8428 1   0.7
12.28821       FALSE
```

```
ggtrace_inspect_return(p, StatBoxplot$compute_group, 2)
```

```
  ymin lower middle upper ymax outliers notchupper notchlower x width
relvarwidth flipped_aes
1  178   191    196   201  212             197.916    194.084 2   0.7
8.246211       FALSE
```

```
ggtrace_inspect_return(p, StatBoxplot$compute_group, 3)
```

```
  ymin lower middle upper ymax outliers notchupper notchlower x width
relvarwidth flipped_aes
1  203   212    216   221  231           217.2822   214.7178 3   0.7
11.09054       FALSE
```

```
ggtrace_inspect_return(p, Stat$compute_panel)
```

```
  ymin lower middle upper ymax outliers notchupper notchlower x width
relvarwidth flipped_aes      fill PANEL group
1  174   186    190   195  208 172, 210   191.1572   188.8428 1   0.7
12.288206      FALSE    Adelie     1     1
```

`Capture` workflow

Capture `compute_group()`

```
compute_group1_fn <- ggtrace_capture_fn(
  x = p, method = StatBoxplot$compute_group, cond = 1
)
```

```
compute_group1_fn()
```

```
  ymin lower middle upper ymax outliers notchupper notchlower x width relvarwidth
flipped_aes
1  174   186    190   195  208 172, 210   191.1572   188.8428 1   0.7    12.28821
FALSE
```


```
compute_group1_fn(coef = 0.8)$outliers
```

```
[[1]]
 [1] 174 172 178 178 205 208 178 203 176 210
```

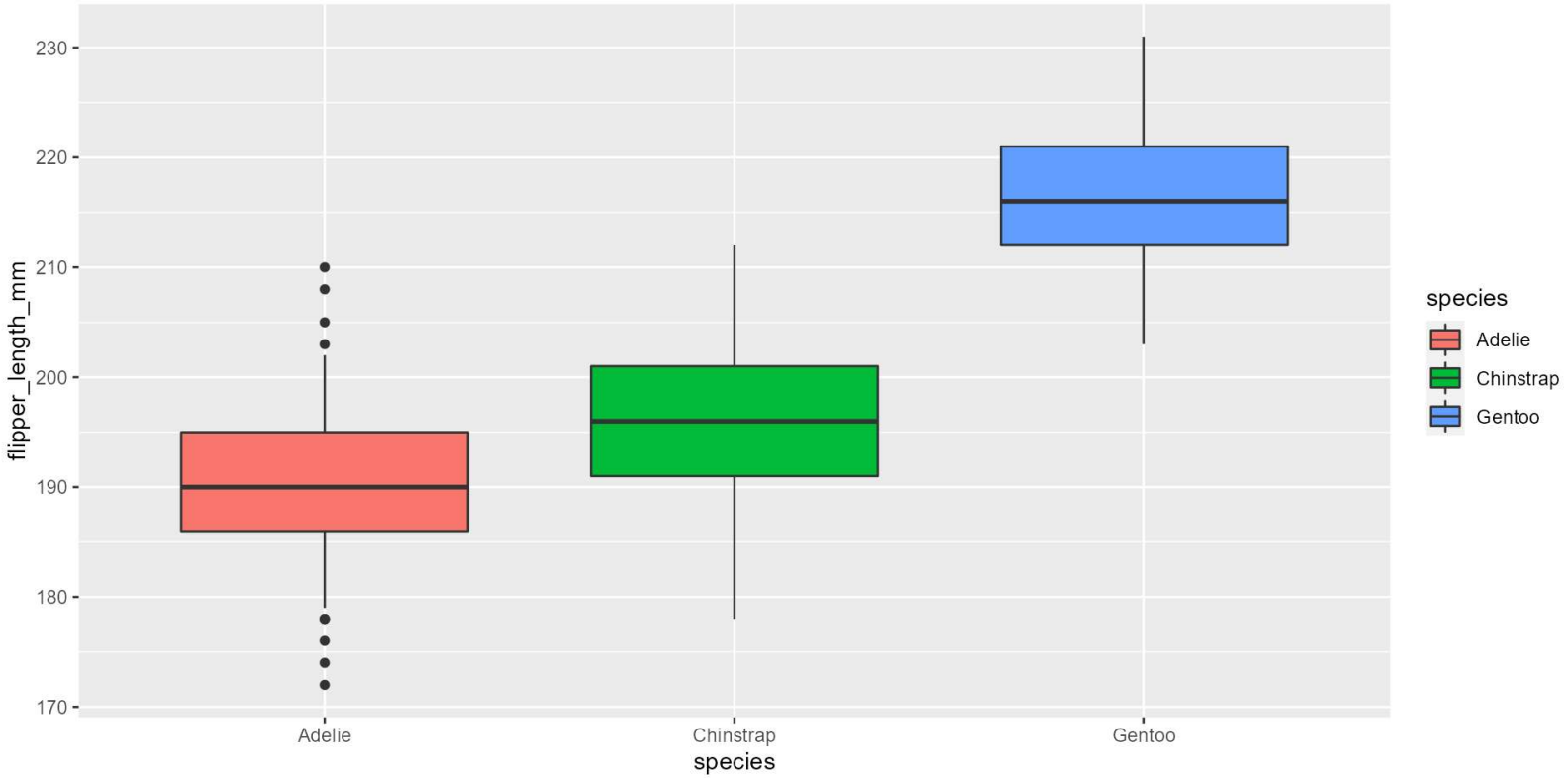
```
# v3.3.0 `geom_boxplot(orientation = "y")`
compute_group1_fn(flipped_aes = TRUE) %>% names()
```

```
 [1] "xmin"         "xlower"       "xmiddle"
 [4] "xupper"       "xmax"         "outliers"
 [7] "notchupper"   "notchlower"   "y"
[10] "width"        "relvarwidth"  "flipped_aes"
```

`Highjack` workflow

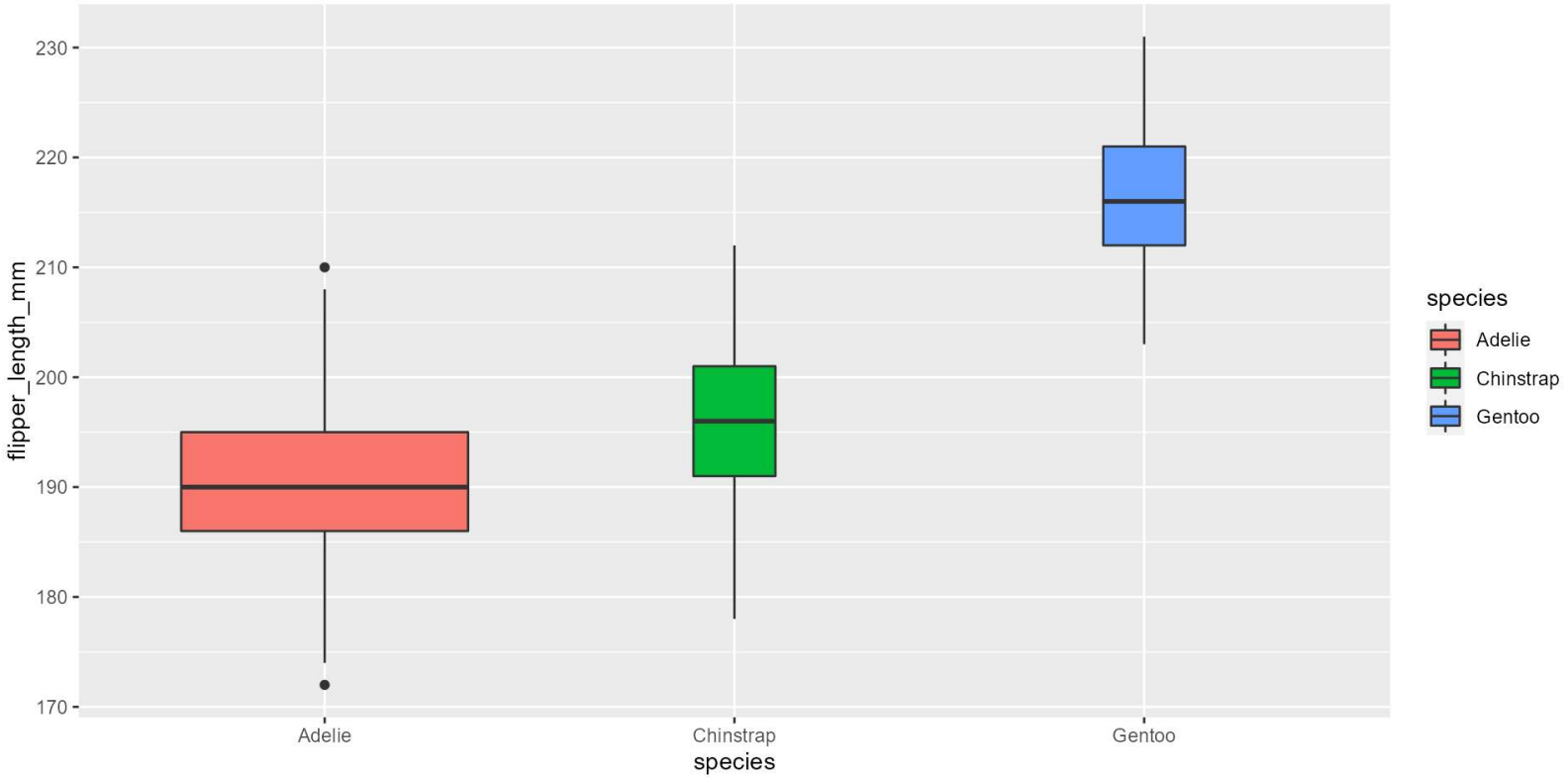
Highjack args

```
ggtrace_highjack_args(
  x = p, method = StatBoxplot$compute_group, cond = 1,
  values = list(coef = 0.8) # list of new argument values
)
```



Highjack args (multi-cond)

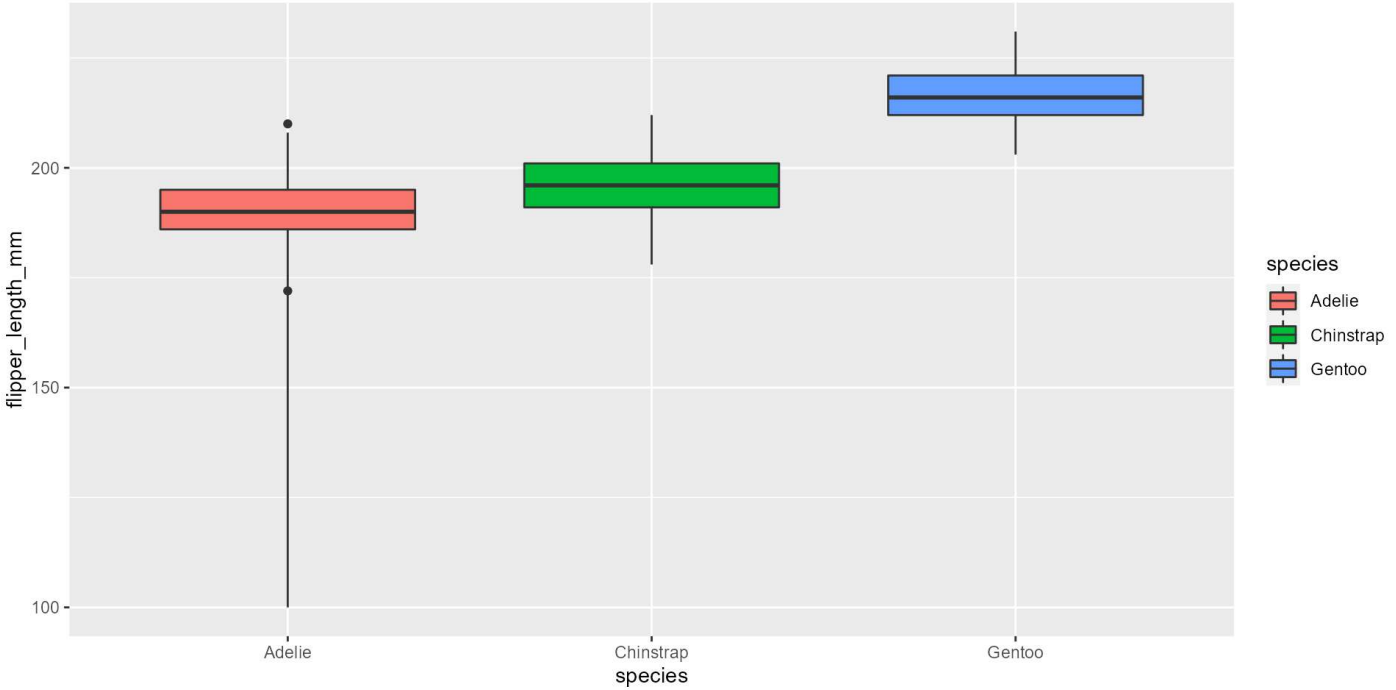
```
ggtrace_highjack_args(
  p, StatBoxplot$compute_group, cond = c(2, 3),
  values = list(width = .2)
)
```



Highjack return

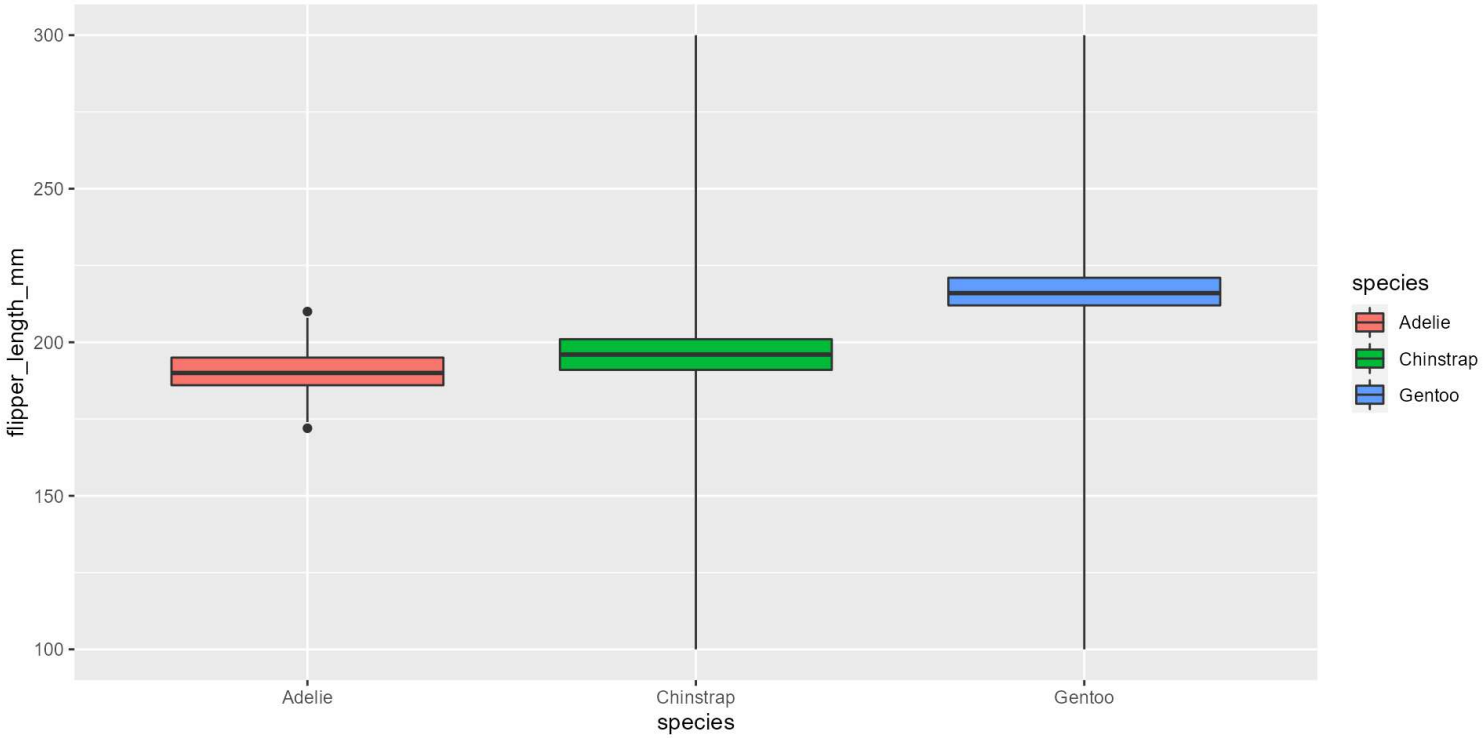
```
modified_compute_group1 <- ggtrace_inspect_return(p, StatBoxplot$compute_group, 1) %>%
  mutate(ymin = 100)
```

```
ggtrace_highjack_return(p, StatBoxplot$compute_group, 1,
  value = modified_compute_group1
)
```



Highjack return - `returnValue()`

```
ggtrace_highjack_return(p, StatBoxplot$compute_group,
  cond = c(2, 3),
  value = quote({
    returnValue() %>% mutate(ymin = 100, ymax = 300)
}))
```



Inspect `draw_group()`

```
ggtrace_inspect_args(p, GeomBoxplot$draw_group, 1)$data
```

```
     fill ymin lower middle upper ymax outliers notchupper notchlower x flipped_aes
PANEL group
1 #F8766D  174   186    190   195  208 172, 210   191.1572   188.8428 1       FALSE
1     1
  ymin_final ymax_final xmin xmax xid newx new_width weight colour size alpha shape
linetype
1        172        210 0.65 1.35   1    1       0.7      1 grey20  0.5    NA    19
solid
```

```
draw_group1_output <- ggtrace_inspect_return(p, GeomBoxplot$draw_group, 1)
```

```
class(draw_group1_output)
```

```
[1] "gTree" "grob"  "gDesc"
```

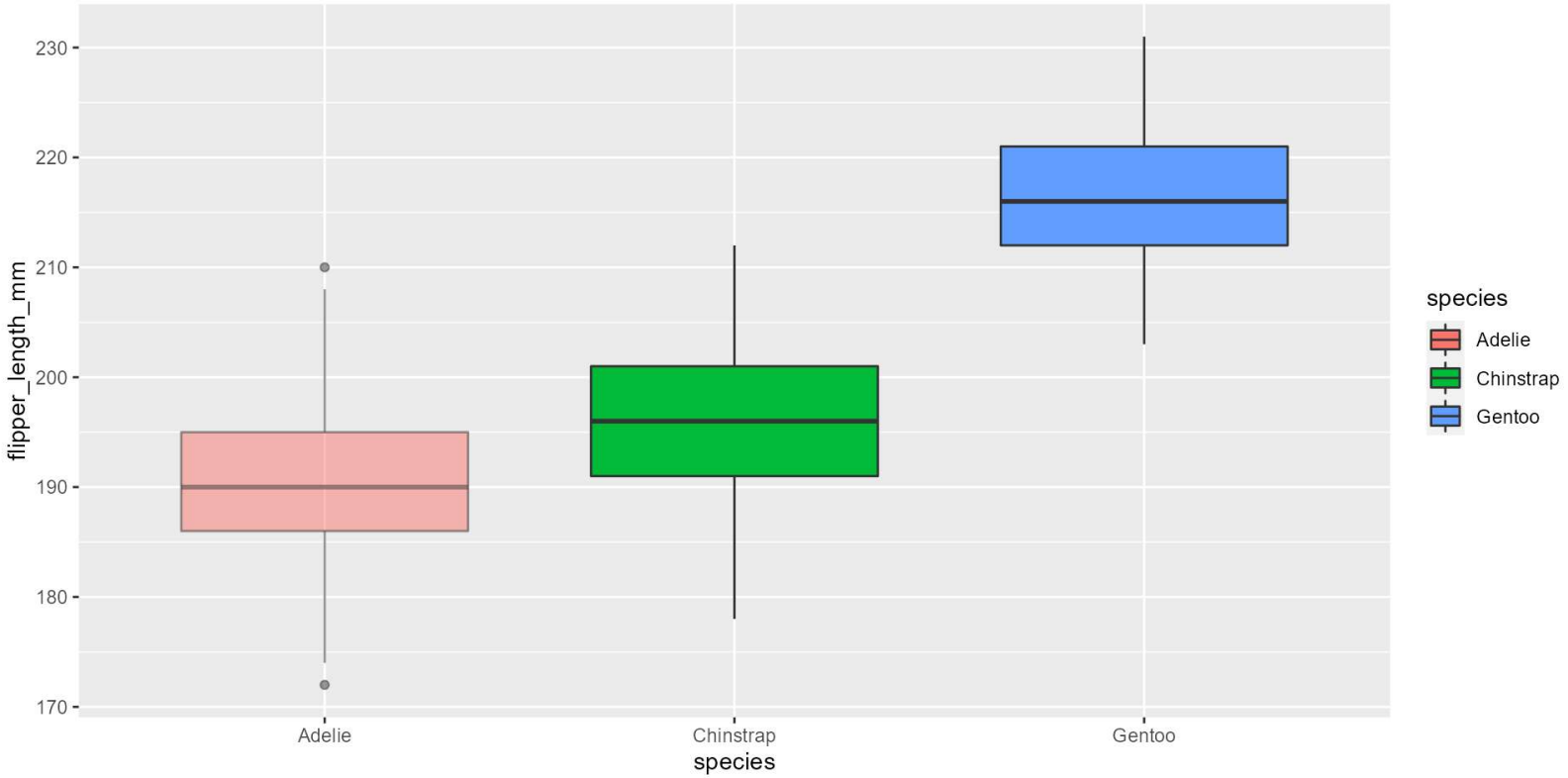
```
typeof(draw_group1_output)
```

```
[1] "list"
```

What can we do with these "grob"s?

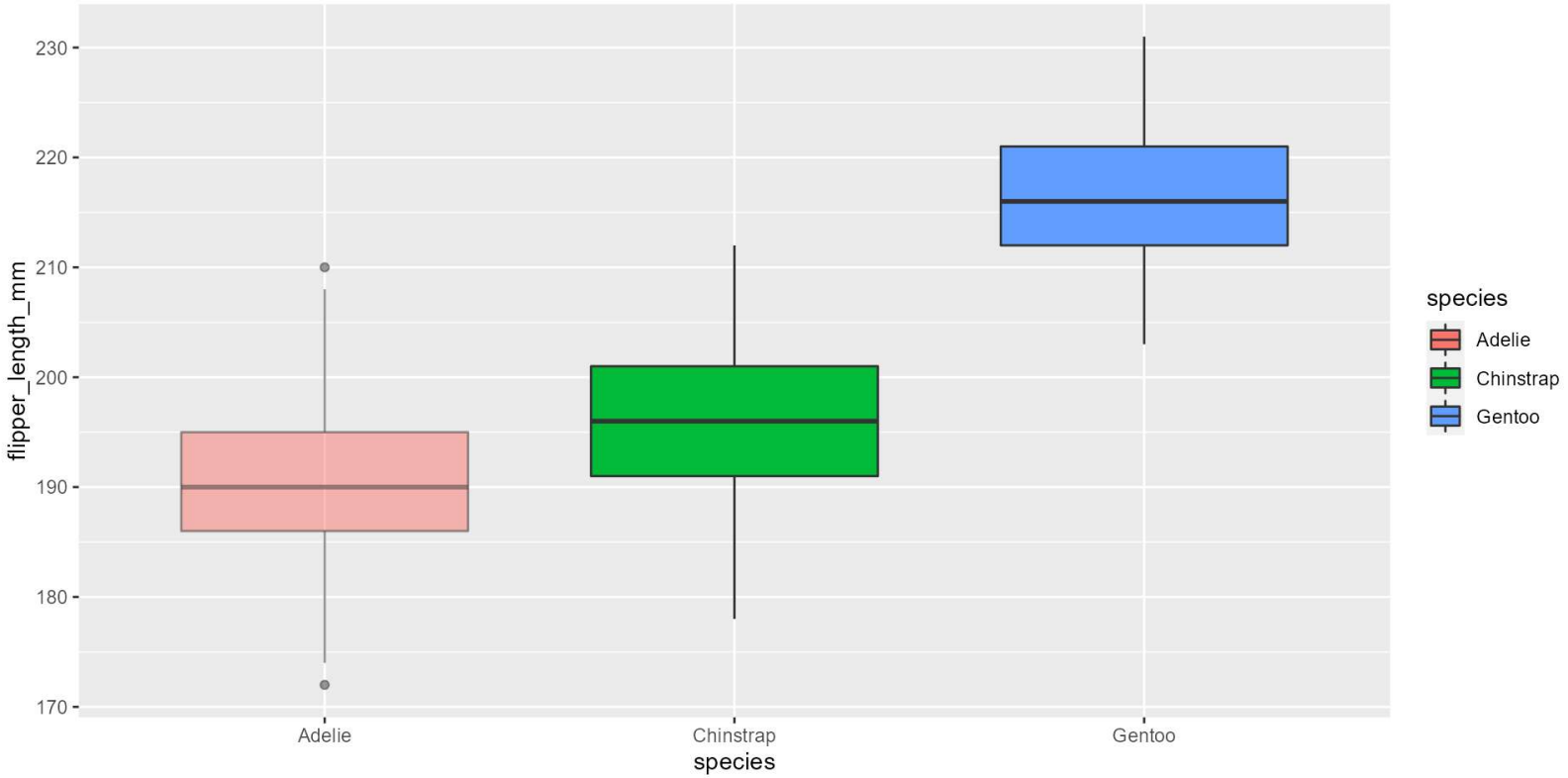
Highjack `draw_group()`

```
modified_group1_output <- grid::editGrob(draw_group1_output, gp = grid::gpar(alpha = .5))
ggtrace_highjack_return(p, GeomBoxplot$draw_group, 1,
  value = modified_group1_output
)
```



Highjack `draw_group()`

```
ggtrace_highjack_return(p, GeomBoxplot$draw_group, 1,
  value = quote({
    editGrob(returnValue(), gp = gpar(alpha = .5))
}))
```



gTrees - a collection of grobs

```
draw_group1_output
```

```
gTree[geom_boxplot.gTree.2112]
```

```
class( draw_group1_output )
```

```
[1] "gTree" "grob"  "gDesc"
```

```
grid::grid.ls(draw_group1_output)
```

```
geom_boxplot.gTree.2112
  geom_point.points.2104
  GRID.segments.2106
  geom_crossbar.gTree.2111
    geom_polygon.polygon.2109
    GRID.segments.2110
```

```
draw_group1_output$children
```

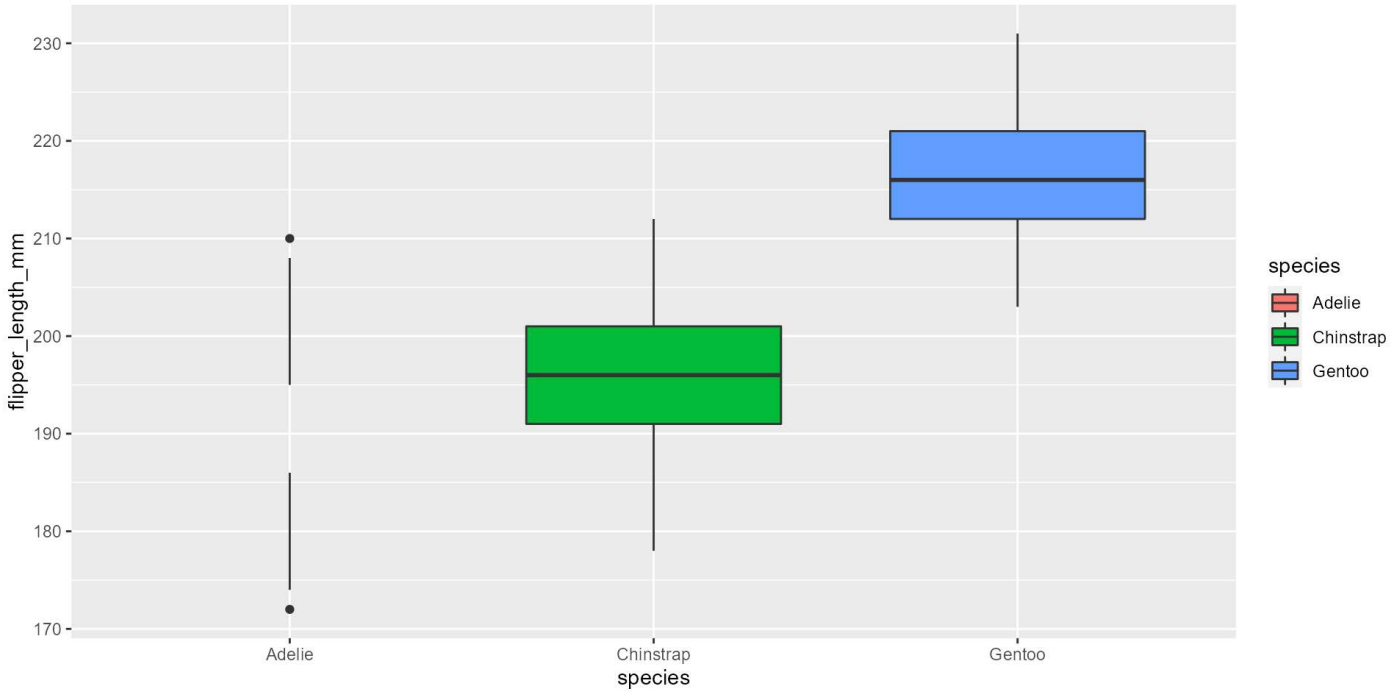
```
(points[geom_point.points.2104], segments[GRID.segments.2106],
gTree[geom_crossbar.gTree.2111])
```

```
draw_group1_output$children[[3]]$children
```

```
(polygon[geom_polygon.polygon.2109], segments[GRID.segments.2110])
```

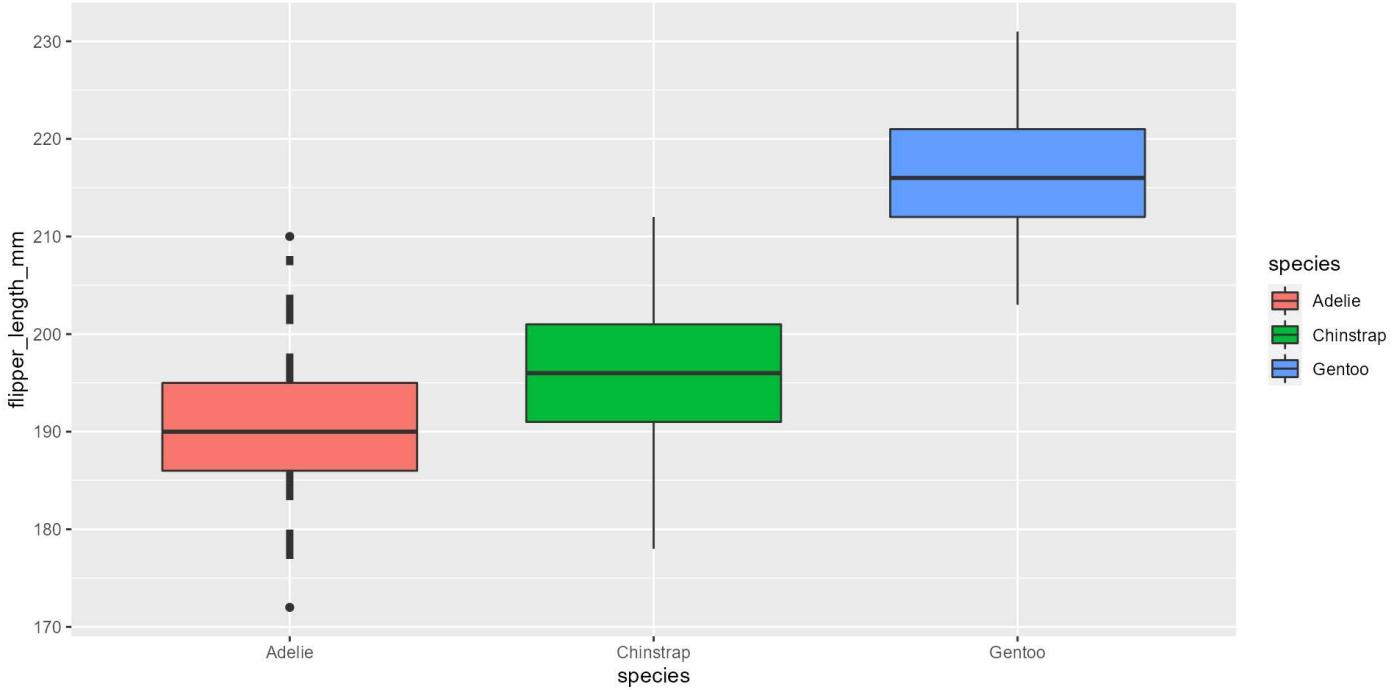
Highjack `draw_group()`

```
ggtrace_highjack_return(p, GeomBoxplot$draw_group, 1,
  value = quote({
    out <- returnValue()
    out$children[3] <- NULL
    out
}))
```



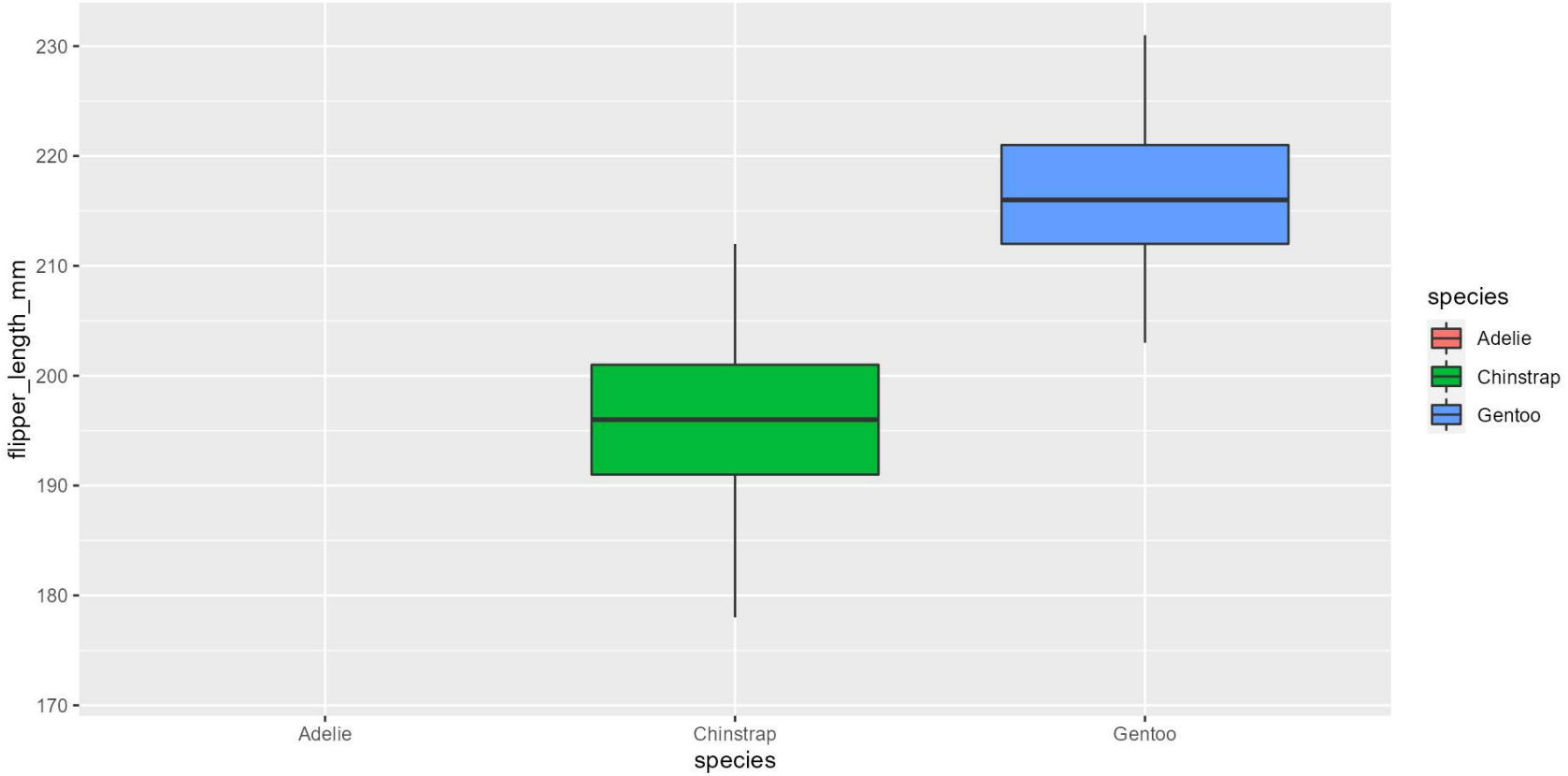
Highjack `draw_group()`

```
ggtrace_highjack_return(p, GeomBoxplot$draw_group, 1,
  value = quote({
    out <- returnValue()
    out$children[[2]] <- editGrob(out$children[[2]], gp = gpar(lty = 2, lwd = 5))
    out
}))
```



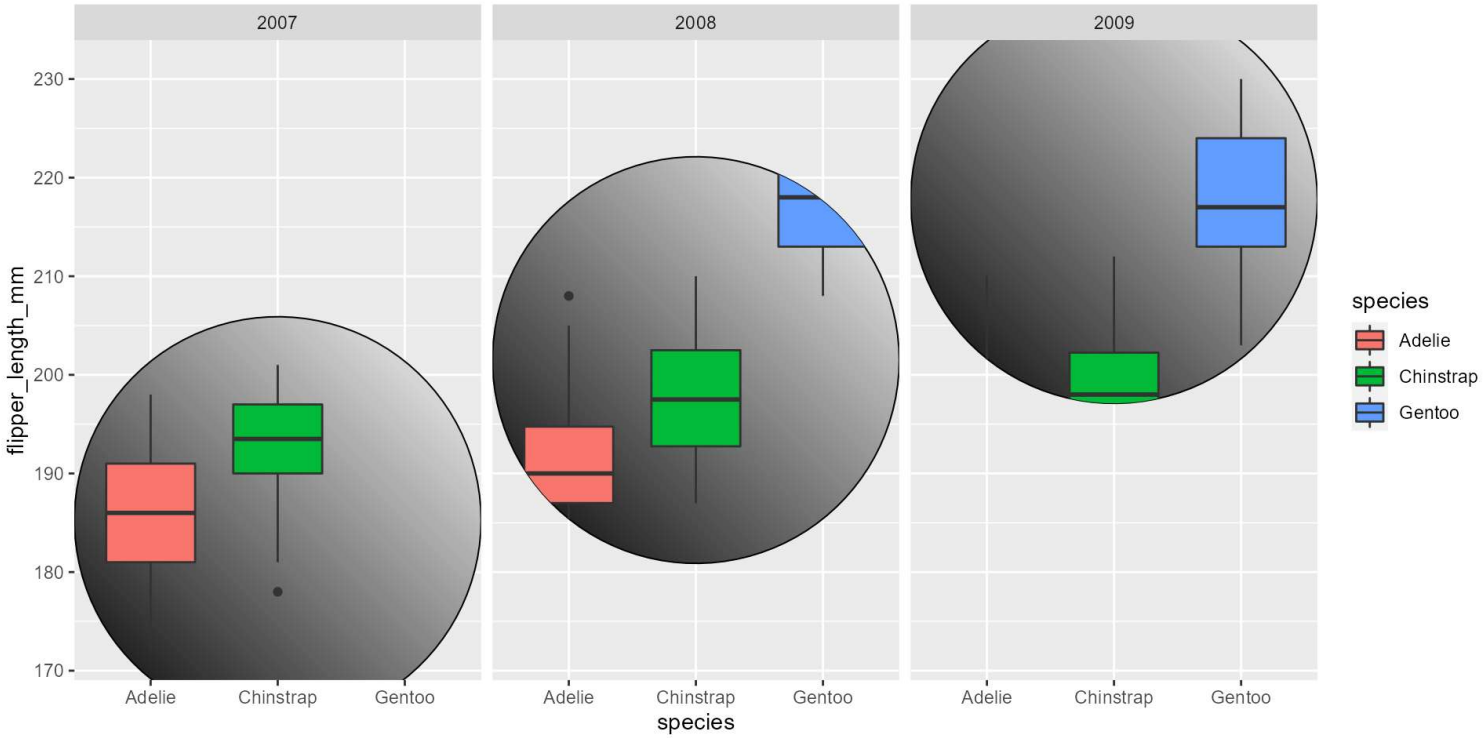
Highjack `draw_group()` - hide

```
ggtrace_highjack_return(p, GeomBoxplot$draw_group, 1,
  value = ggplot2::zeroGrob()
)
```



For {grid} power users

```
ggtrace_highjack_return(p + facet_wrap(~ year), Geom$draw_panel, cond = TRUE,
  value = quote({
    y_pos <- .25 * ._counter_ #<- internal counter tracking nth time the method is calle
    grobTree( circleGrob(y = y_pos, gp = gpar(fill = linearGradient())), # R >= 4.1
              editGrob(returnValue(), vp = viewport(clip = circleGrob(y = y_pos))) )  })
```



… and much more!

Workflow functions also work for:

- Other ggprotos, unexported functions, S3/S4 methods
- Functions and methods from extension packages

Beyond workflow functions, `{ggtrace}` also provides:

- **Interactive debugging** with `ggedit()` and `ggdebug()`
- Low-level control with `ggtrace()` and `with_ggtrace()`

https://yjunechoe.github.io/ggtrace/