Introducing ggvis

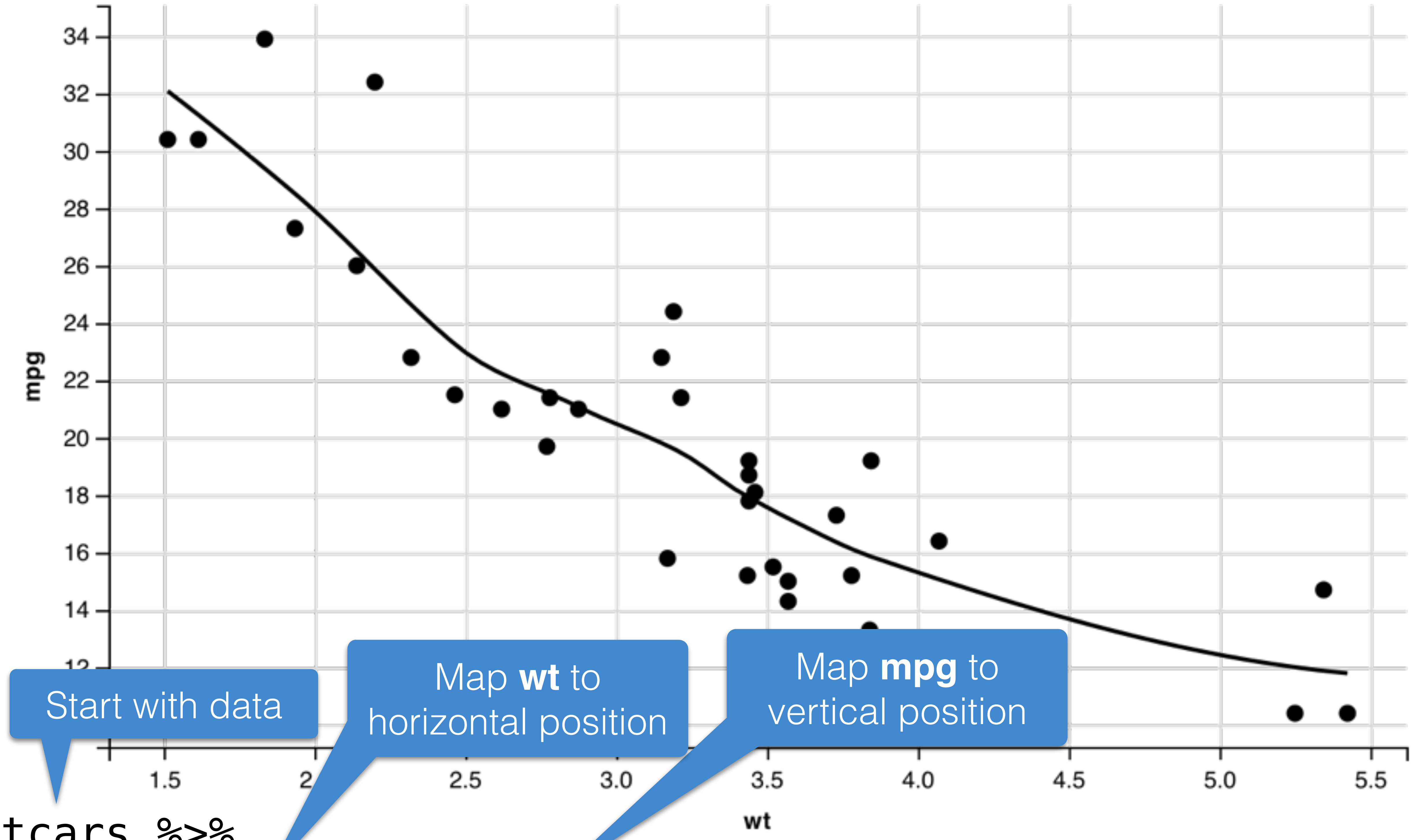
Winston Chang
Hadley Wickham
RStudio
June 2014

What is ggvis?

A package for interactive data visualization - a synthesis of ideas:

- Grammar of graphics (ggplot2)
- Reactivity (Shiny)
- Data pipeline (dplyr)
- Of the web (vega.js)

Grammar of graphics



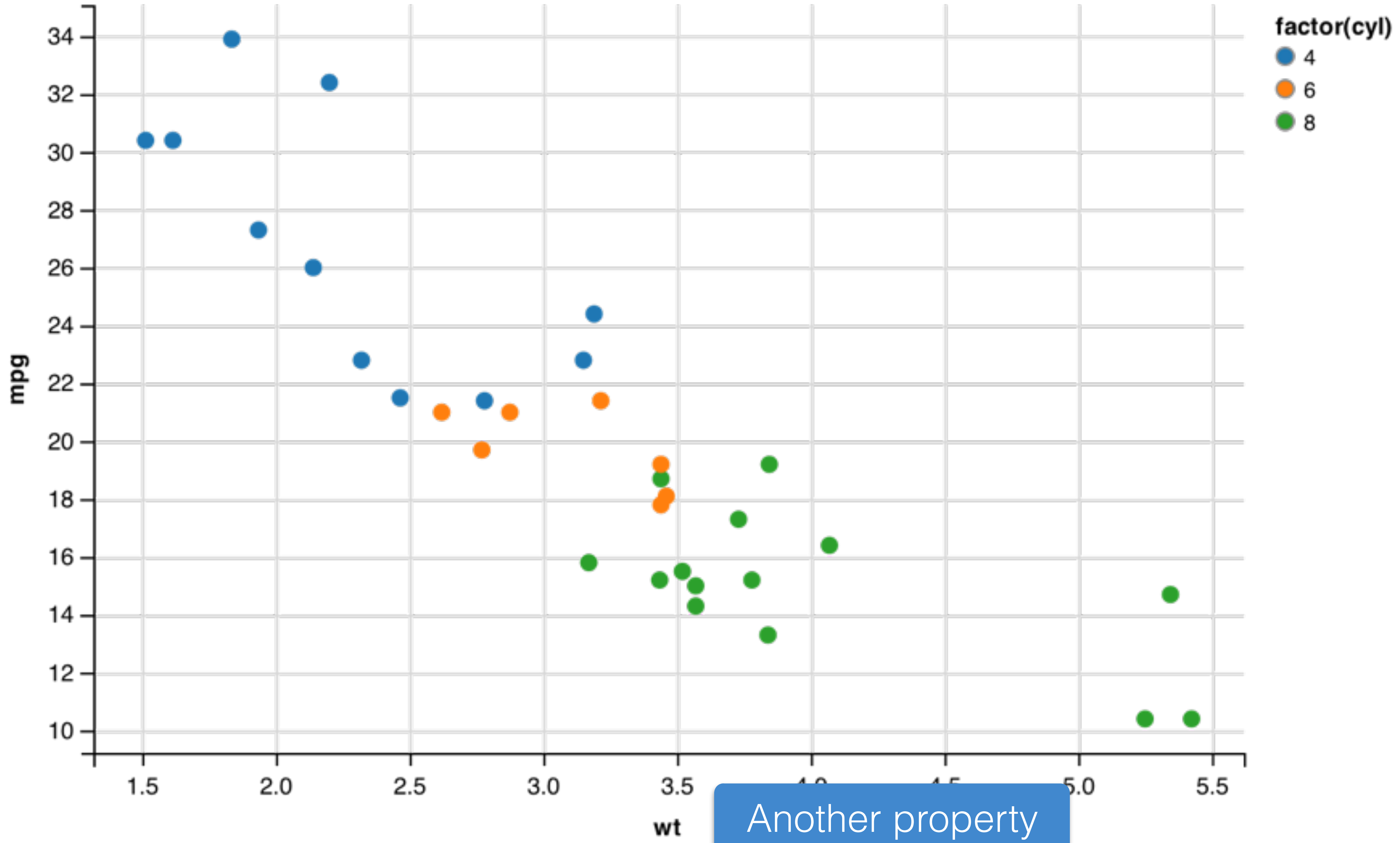
```
mtcars %>%
  ggvis(x = ~wt, y = ~mpg) %>%
  layer_points() %>%
  layer_smooths()
```

```
mtcars %>%
  ggvis(x = ~wt, y = ~mpg) %>%
  layer_points(x = ~wt, y = ~mpg) %>%
  layer_smooths(x = ~wt, y = ~mpg)
```

x and y are defaults

```
mtcars %>%
  ggvis(~wt, ~mpg) %>%
  layer_points() %>%
  layer_smooths()
```

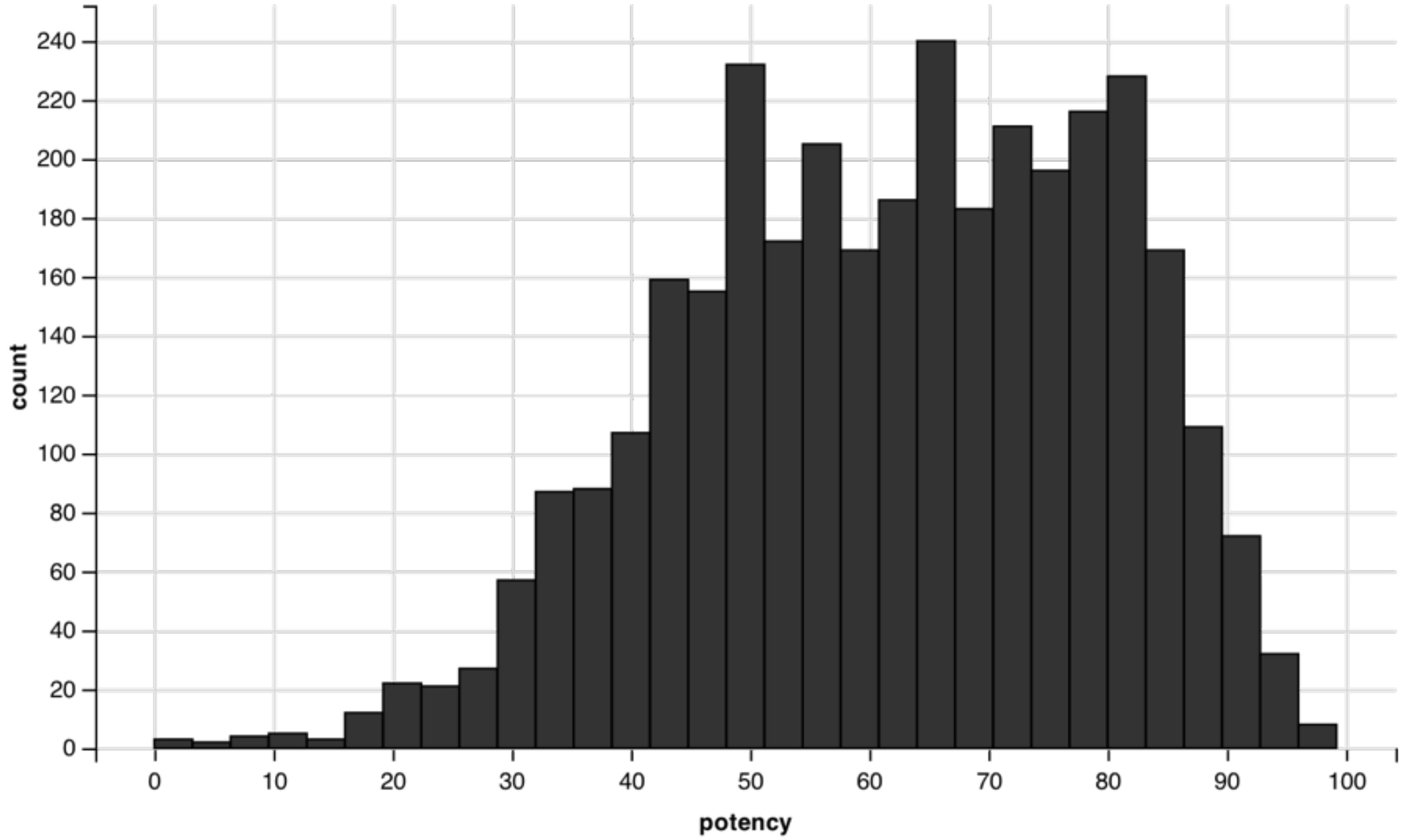
layers inherit property mappings



Another property mapping: fill color

```
mtcars %>%
  ggvis(x = ~wt, y = ~mpg, fill = ~factor(cyl)) %>%
  layer_points()
```

Use factor() to treat cyl as categorical



```
cocaine %>% ggvis(x = ~potency) %>% layer_histograms()
```

```
cocaine %>% ggvis(~potency)
```

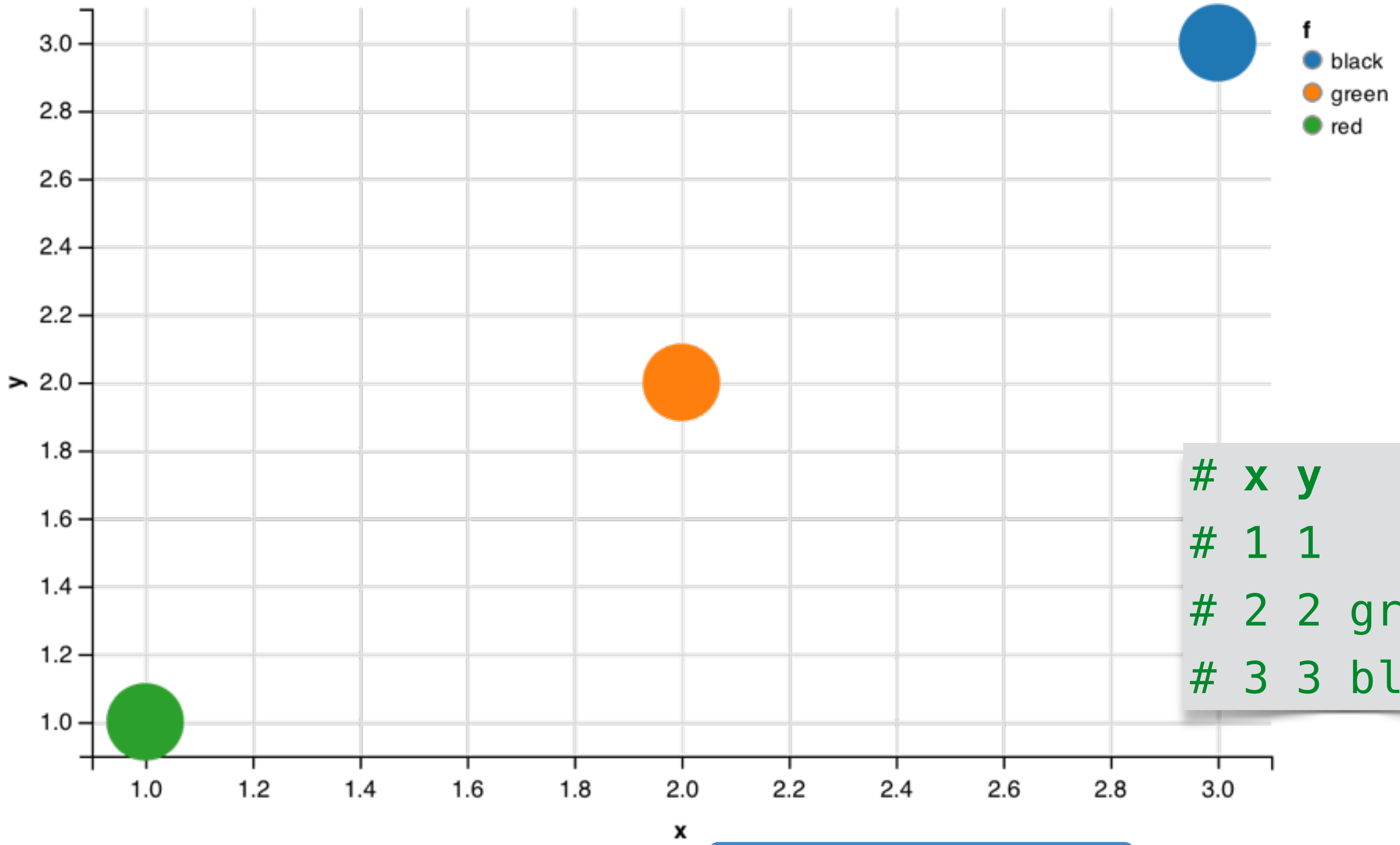
When no layer specified, ggvis will guess

Scaled and unscaled values

```
dat <- data.frame(x = 1:3, y = 1:3,
                  f = c("red", "green", "black"))
# x y     f
# 1 1   red
# 2 2 green
# 3 3 black

dat %>%
  ggvis(x = ~x, y = ~y, fill = ~f) %>%
  layer_points()

dat %>%
  ggvis(x = ~x, y = ~y, fill := ~f) %>%
  layer_points()
```

= means scaled value

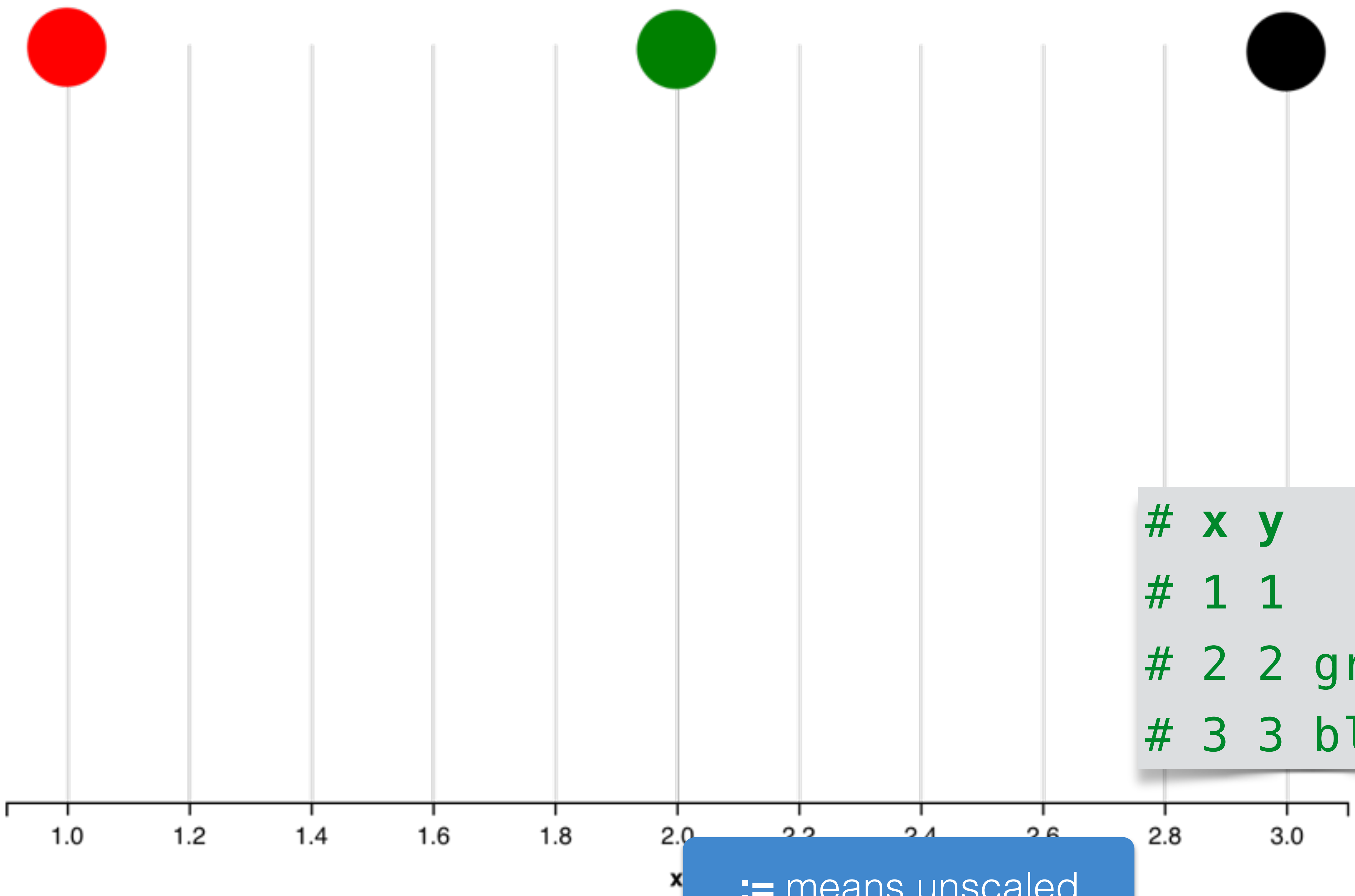
:= means unscaled (raw) value



```
# x y     f
# 1 1   red
# 2 2 green
# 3 3 black
```

= means scaled value

```
dat %>%
  ggvis(x = ~x, y = ~y, fill = ~f) %>%
  layer_points()
```



```
# x y     f
# 1 1   red
# 2 2 green
# 3 3 black
```

:= means unscaled (raw) value

```
dat %>%
  ggvis(x = ~x, y := ~y, fill := ~f) %>%
  layer_points()
```

Capturing expressions with ~

```
mtcars %>%
  ggvis(x = ~wt, y = ~mpg, fill := "red") %>%
  layer_points()
```

```
~ means capture the expression for later evaluation
in the context of the data.
No ~ means evaluate the expression now.
```

- Scaled vs. unscaled
  - **:=** means use raw, unscaled value.
  - **=** means use scaled value.
- Capturing expressions
  - **~** means capture this expression for later evaluation, in the context of the data.
  - No **~** means evaluate this expression now.
- Most commonly use:
  - scaled expressions (x = ~wt, fill = ~factor(cyl))
  - unscaled literals (fill := "red")

Data pipeline

Functional interface

Each ggvis function takes a visualization object as an input and returns a modified visualization as an output:

```
p <- ggvis(mtcars, x = ~wt, y = ~mpg)
p <- layer_points(p)
p <- layer_smooths(p)
p
```

Create a ggvis object with mtcars data.

Layer on points

Layer on smoothing lines

Print

The **%>%** operator

From **magrittr** package. Used extensively in **dplyr**.

%>% is a piping operator, pronounced “then”.

It takes the output of the left side, and uses it as the first argument of the function on the right side.

```
subset(mtcars, cyl == 6, c(mpg, wt))
mtcars %>% subset(cyl == 6, c(mpg, wt))

summary(subset(mtcars, cyl == 6, c(mpg, wt)), digits=2)
mtcars %>% subset(cyl == 6, c(mpg, wt)) %>% summary(digits=2)
```

```r
# Three equivalent forms

layer_smooths(layer_points(ggvis(mtcars, ~wt, ~mpg)),
  span = 0.5)


p <- ggvis(mtcars, ~wt, ~mpg)
p <- layer_points(p)
p <- layer_smooths(p, span = 0.5)
p


mtcars %>%
  ggvis(x = ~wt, y = ~mpg) %>%
  layer_points() %>%
  layer_smooths(span = 0.5)
```

Some layers perform a computation on the data

```
mtcars %>% ggvis(~wt, ~mpg) %>%
 layer_smooths()

# Roughly equivalent to:
mtcars %>% ggvis(~wt, ~mpg) %>%
  compute_smooth(mpg ~ wt) %>%
  layer_lines()
```

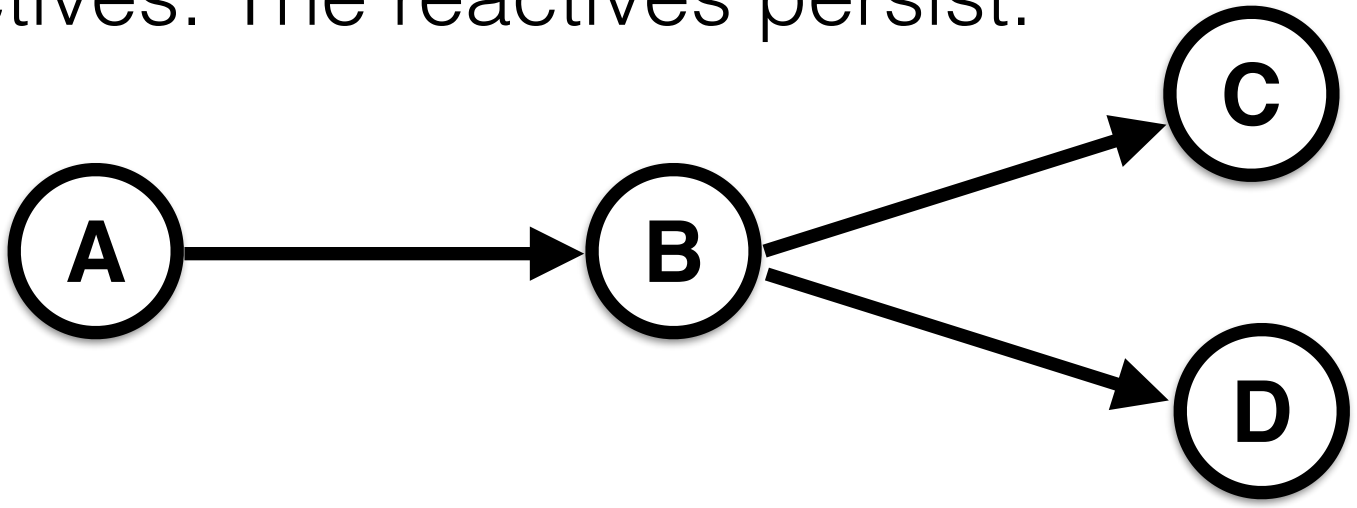
```
# Compute functions can operate directly on data,
# without ggvis().
mtcars %>% compute_smooth(mpg ~ wt)
      pred_    resp_
1  1.513000 32.08897
2  1.562506 31.68786
3  1.612013 31.28163
4  1.661519 30.87037
5  1.711025 30.45419
6  1.760532 30.03318
7  1.810038 29.60745
8  1.859544 29.17711
9  1.909051 28.74224
10 1.958557 28.30017
   ...

# Or for a linear model:
mtcars %>% compute_smooth(mpg ~ wt, method = "lm")
```

Reactivity and Interactivity

Reactives from Shiny

- In “regular” programming, function calls happen once. The function takes in a value and returns a value.

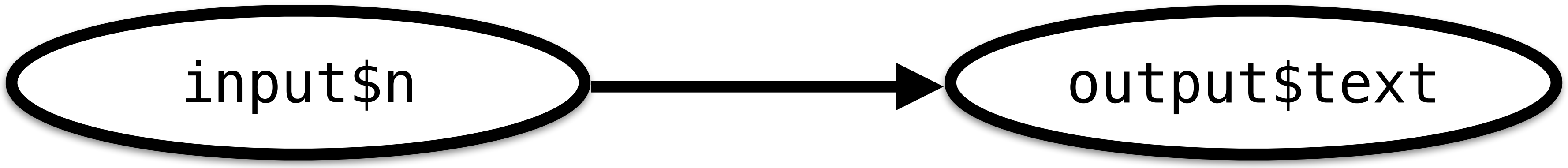
- In functional reactive programming, a reactive can use a value from another reactive; this creates a dependency graph of reactives. The reactives persist.



- When the value of an ancestor node changes, it triggers recomputation of all its descendants.

```
# A very simple Shiny application demonstrating basic
# reactivity.
library(shiny)

runApp(list(
  ui = basicPage(
    sliderInput('n', 'Number', 1, 100, value = 50),
    textOutput('text')
  ),
  server = function(input, output) {
    output$text <- renderText({
      paste("The value of n is ", input$n)
    })
  }
))
```

Reactive



Reactive computation parameters

```
faithful %>%
  ggvis(x = ~waiting) %>%
  layer_histograms(binwidth =
    input_slider(min=1, max=20, value=11))
```

Reactive properties

```
mtcars %>%
  ggvis(x = ~wt, y = ~mpg) %>%
  layer_points(
    size := input_slider(10, 400, value=50, label="size"),
    fill := input_select(c("red", “blue"), label="color")
  )
```

Reactive data sources

```
dat <- data.frame(time = 1:10, value = runif(10))

# Create a reactive that returns a data frame, adding a new
# row every 2 seconds
ddat <- reactive({
  invalidateLater(2000, NULL)
  dat$time  <<- c(dat$time[-1], dat$time[length(dat$time)] + 1)
  dat$value <<- c(dat$value[-1], runif(1))
  dat
})

ddat %>% ggvis(x = ~time, y = ~value, key := ~time) %>%
  layer_points() %>%
  layer_paths()
```

Direct interaction

```r
# This function receives information about the hovered
# point and returns a string to display
all_values <- function(x) {
  if(is.null(x)) return(NULL)
  paste0(names(x), ": ", format(x), collapse = "<br />")
}

mtcars %>% ggvis(x = ~wt, y = ~mpg) %>%
  layer_points() %>%
  add_tooltip(all_values, "hover")
```

Using ggvis with Shiny

Shiny applications

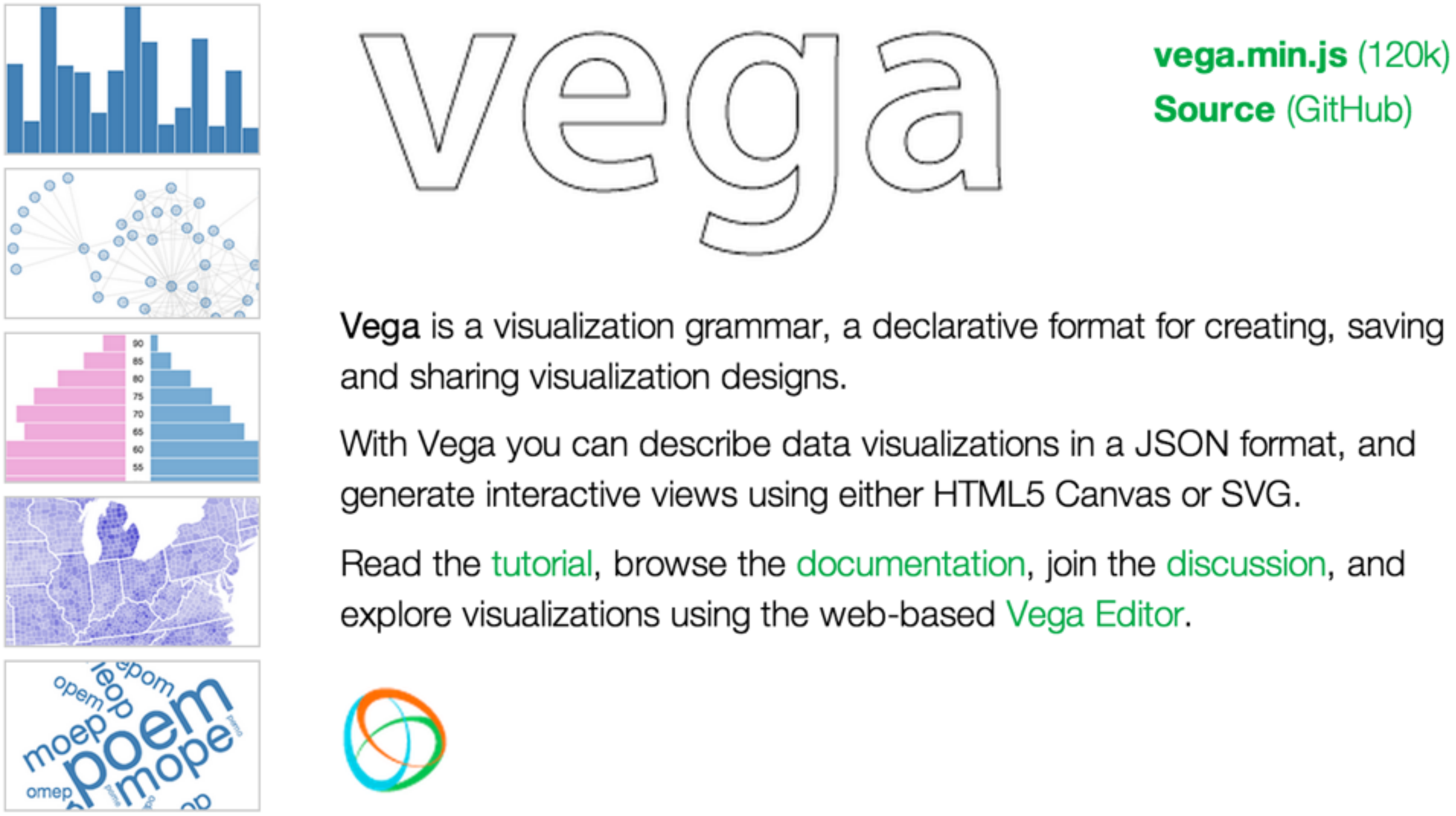
- Movies app

- brush-summary app

Interactive (Shiny) docs

- R Markdown documents + Shiny
- More information: http://bit.ly/TkiPhR

Made of the web



vega

vega.min.js (120k)

Source (GitHub)

Vega is a visualization grammar, a declarative format for creating, saving and sharing visualization designs.

With Vega you can describe data visualizations in a JSON format, and generate interactive views using either HTML5 Canvas or SVG.

Read the tutorial, browse the documentation, join the discussion, and explore visualizations using the web-based Vega Editor.

http://trifacta.github.io/vega/

The future

- Zooming and panning
- Subvisualizations (Faceting)
- ggplot2 feature parity
- Performance improvements
- Rendering without a web browser

More information

- http://ggvis.rstudio.com/
- Examples at: https://github.com/rstudio/ggvis/tree/master/demo