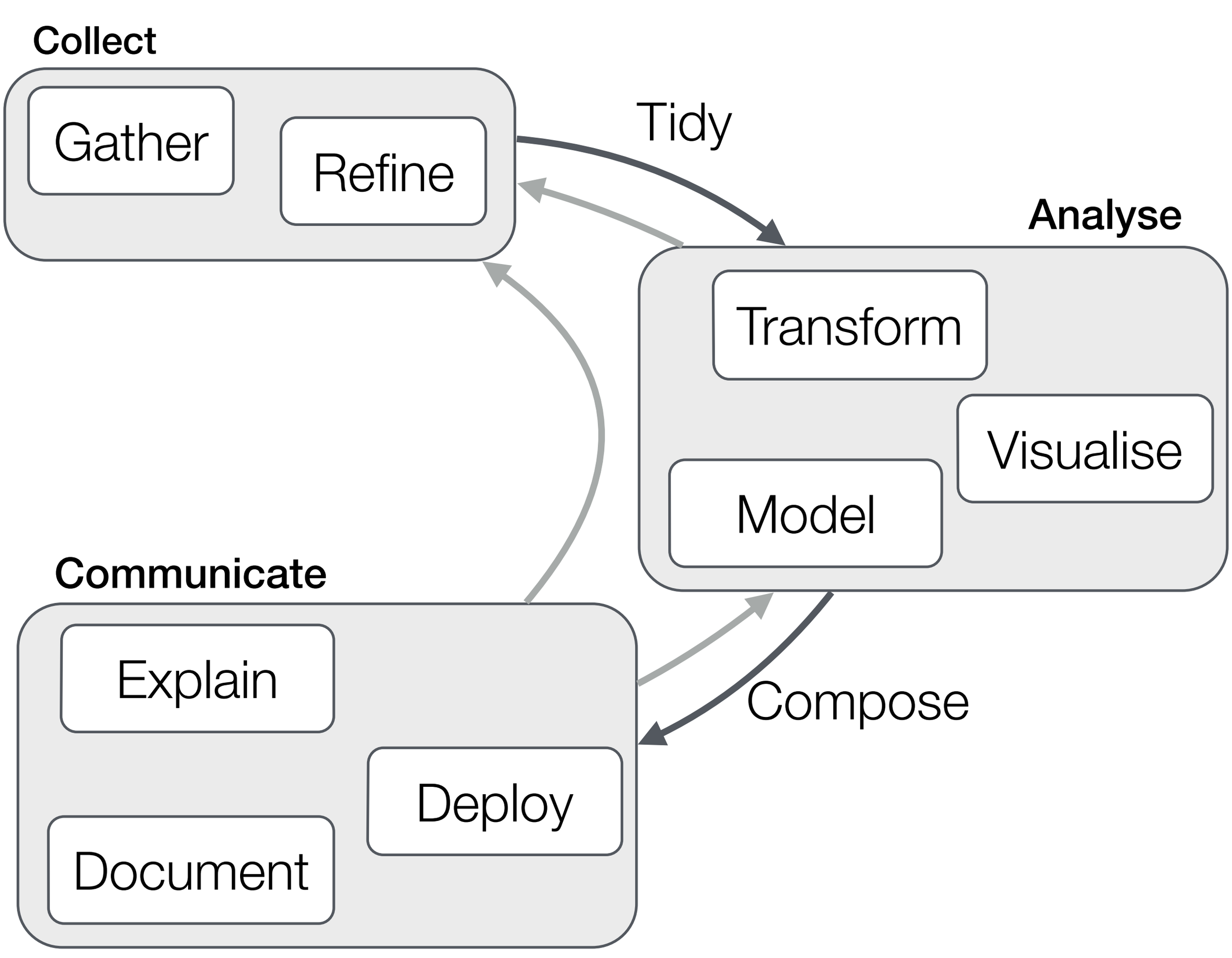
Interactive data visualization with ggvis

Winston Chang
RStudio


Collect
Gather
Refine
Tidy
Analyse
Transform
Visualise
Model
Communicate
Explain
Deploy
Document
Compose

Tidy

tidyr

Transform

dplyr

Visualise

ggvis

Model

Tidy

tidyr

Transform

dplyr

Visualise

ggvis

Model

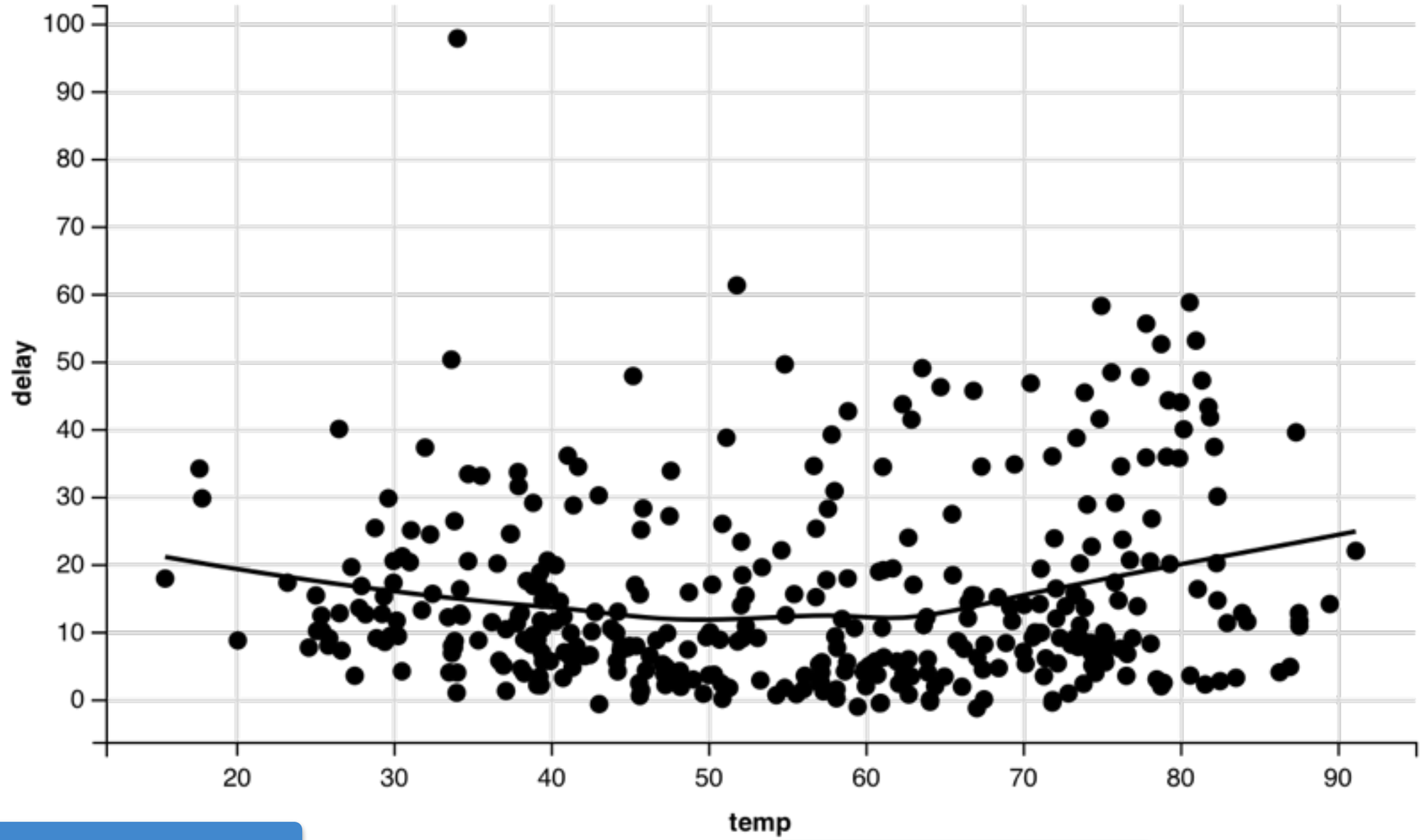
What is ggvis?

A package for interactive data visualization to make it easier to **explore data** and **communicate findings**.

A synthesis of ideas:

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

Grammar of graphics



Start with data

Map **temp** to horizontal position

Map **delay** to vertical position

```
both %>%
  ggvis(x = ~temp, y = ~delay) %>%
  layer_points() %>%
  layer_smooths()
```

Add a layer of points

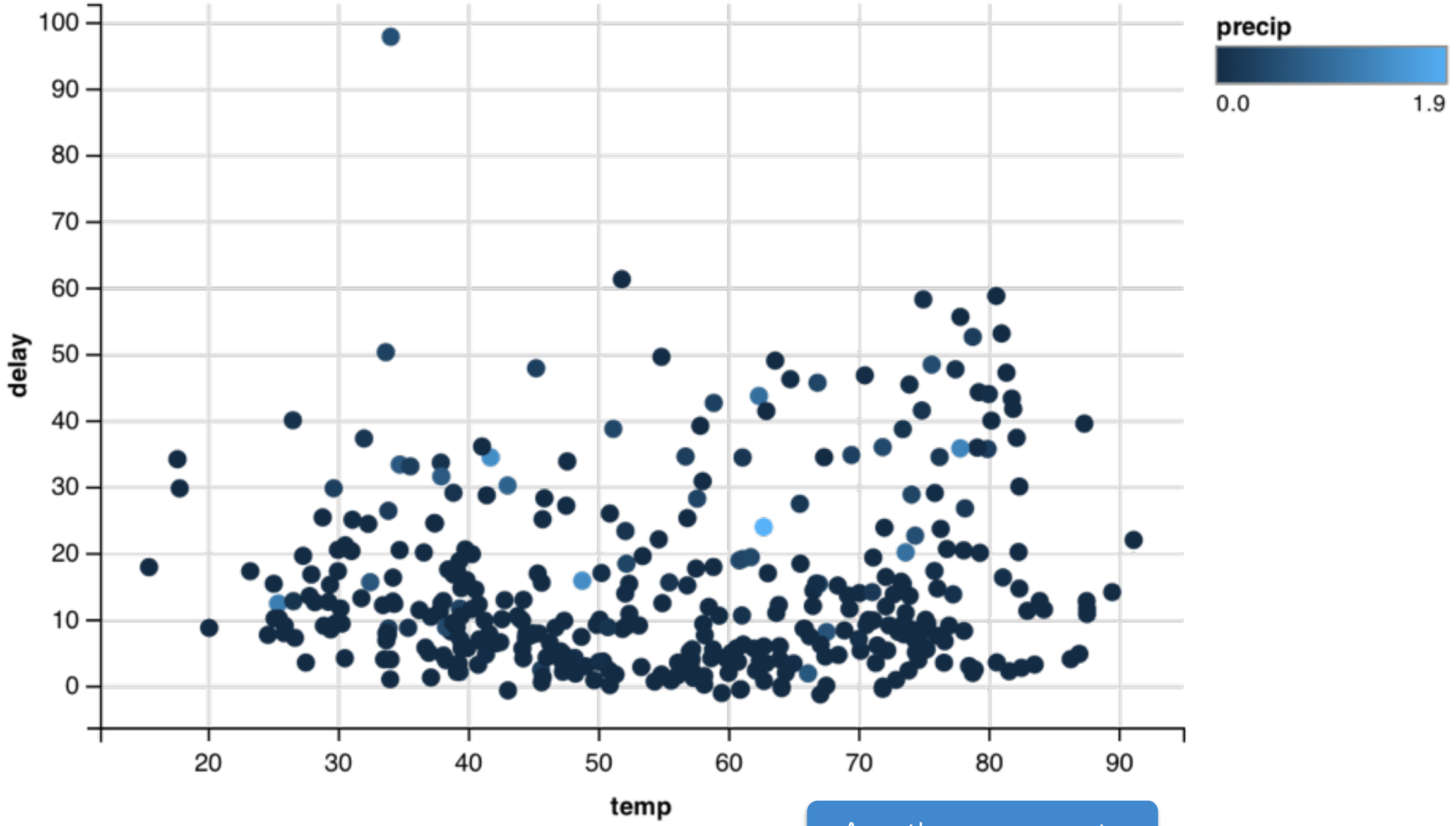
Add a layer of smoothing lines

```
both %>%
  ggvis(x = ~temp, y = ~delay) %>%
  layer_points(x = ~wt, y = ~mpg) %>%
  layer_smooths(x = ~wt, y = ~mpg)
```

```
both %>%
  ggvis(~temp, ~delay) %>%
  layer_points() %>%
  layer_smooths()
```

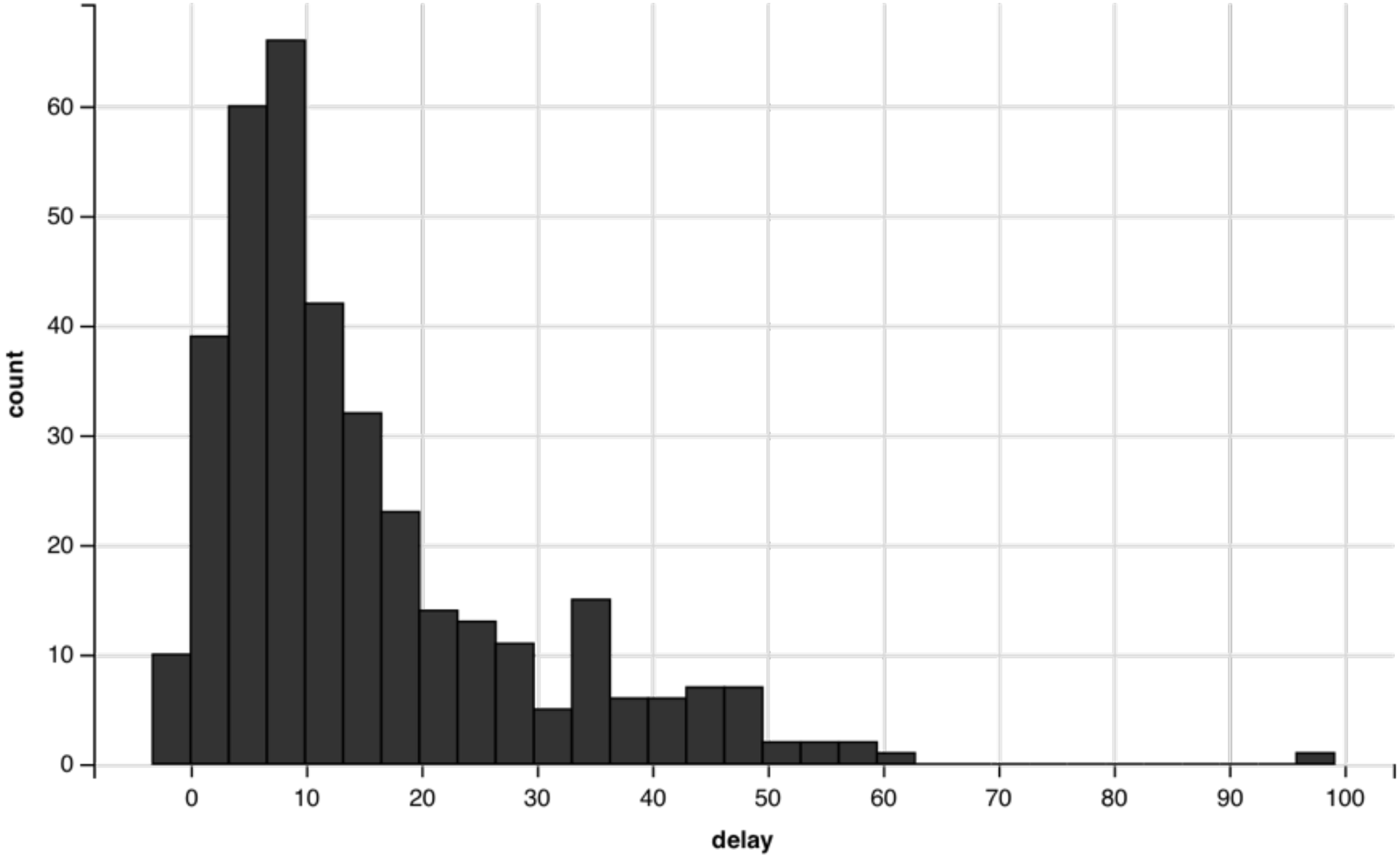
x and y are defaults

layers inherit property mappings



Another property mapping: fill color

```
both %>%
  ggvis(~temp, ~delay, fill = ~precip) %>%
  layer_points()
```




```
both %>% ggvis(~delay) %>% layer_histograms()
```

```
both %>% ggvis(~delay)
```

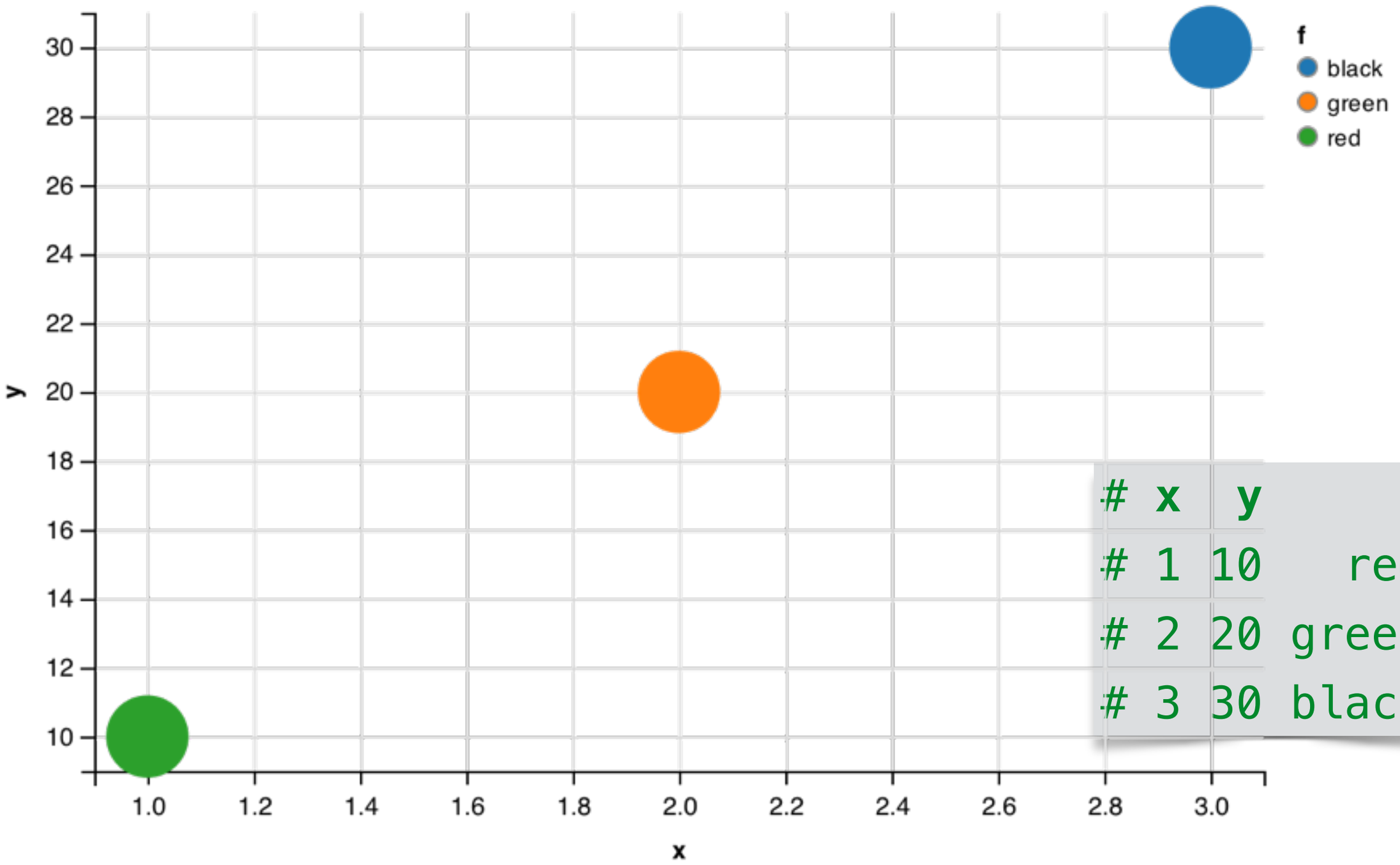
When no layer specified, ggvis will guess

Scaled and unscaled values

```
dat <- data.frame(x = c(1,2,3), y = c(10,20,30),
                  f = c("red", "green", "black"))
# x  y     f
# 1 10   red
# 2 20 green
# 3 30 black

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

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

= means scaled value

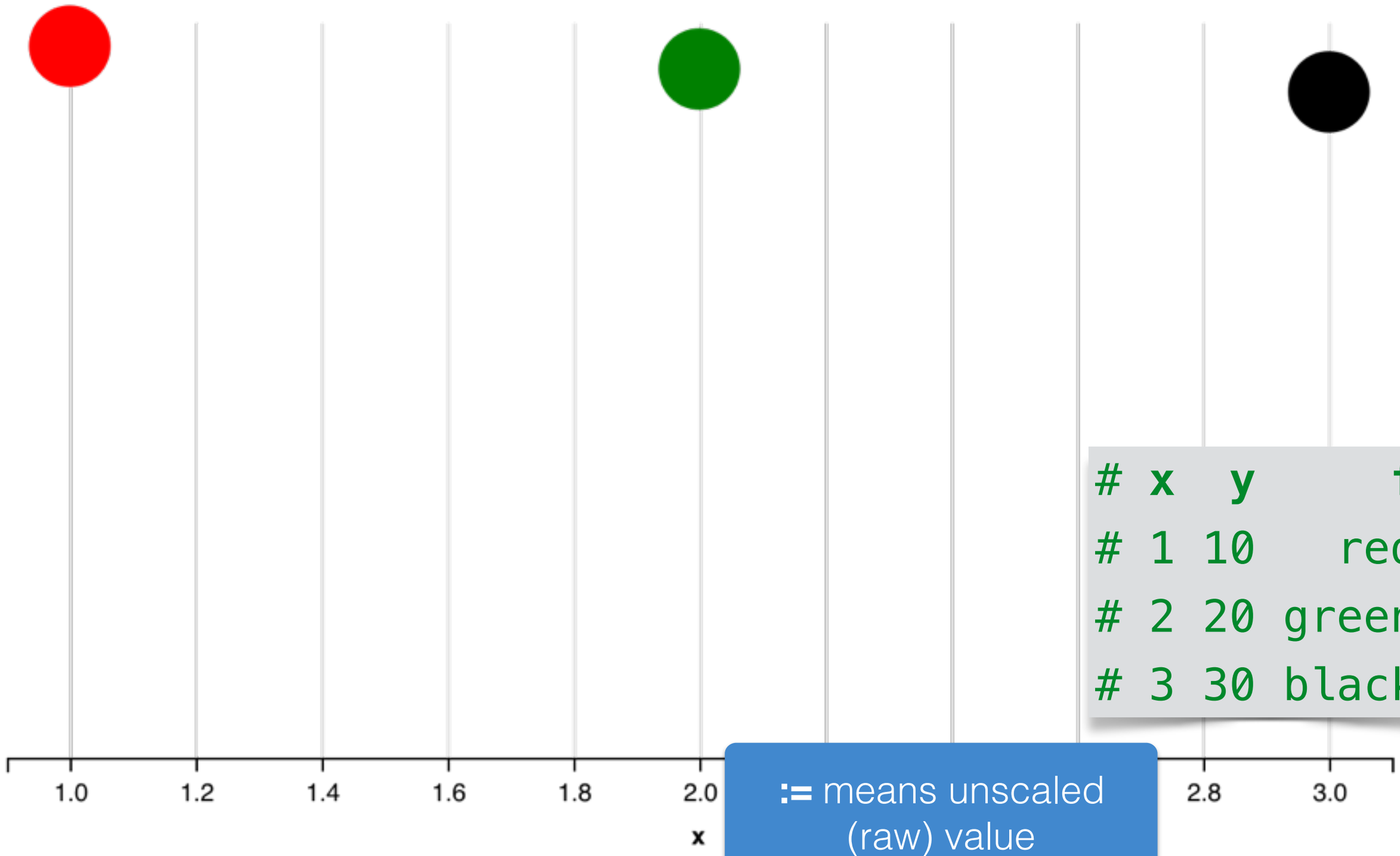
:= means unscaled (raw) value




```
#  x  y     f
#  1 10   red
#  2 20 green
#  3 30 black
```

= means scaled value

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




```
# x  y     f
# 1 10   red
# 2 20 green
# 3 30 black
```

:= means unscaled (raw) value

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

Capturing expressions with ~

```
both %>%
  ggvis(x = ~temp, y = ~delay, fill := "red") %>%
  layer_points()
```

~ means capture the expression for later evaluation in the context of the data.

No ~ means evaluate the expression now.

Data pipeline

Functional interface

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

```
p <- ggvis(both, x = ~temp, y = ~delay)
p <- layer_points(p)
p <- layer_smooths(p)
p
```

Create a ggvis object with 'both' data.

Layer on points

Layer on smoothing lines

Print

```
# Three equivalent forms
p <- ggvis(both, ~temp, ~delay)
p <- layer_points(p)
p <- layer_smooths(p, span = 0.5)
p
```

```
layer_smooths(layer_points(ggvis(both, ~temp, ~delay)),
  span = 0.5)
```

```
both %>%
  ggvis(x = ~temp, y = ~delay) %>%
  layer_points() %>%
  layer_smooths(span = 0.5)
```

Some layers perform a computation on the data

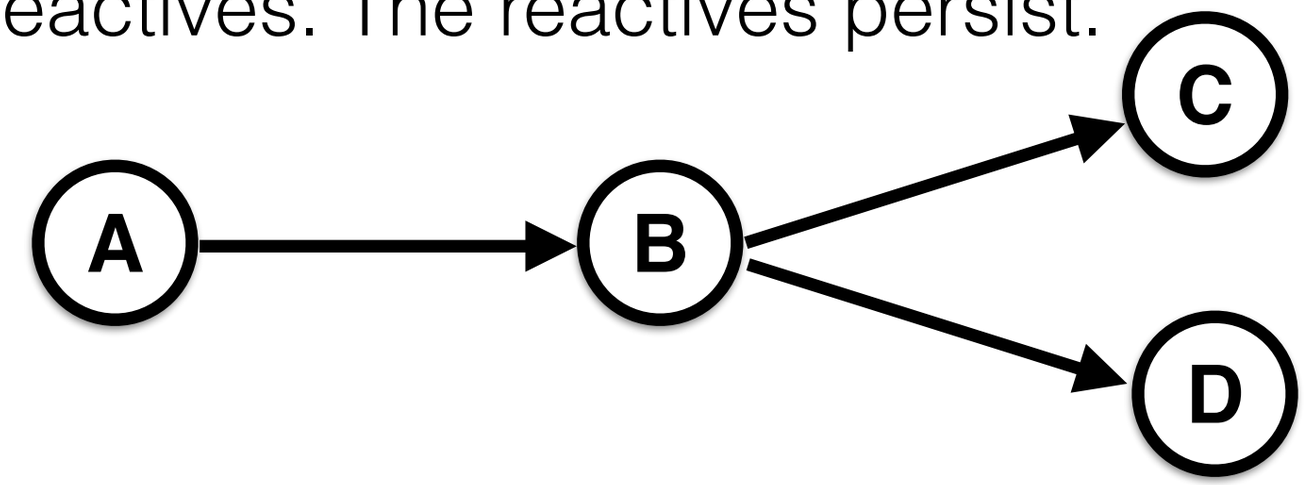
```
both %>% ggvis(~temp, ~delay) %>%
  layer_histograms()

# Roughly equivalent to:
both %>% ggvis(~temp, ~delay) %>%
  compute_bin() %>%
  layer_rects()
```

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.

Reactive computation parameters

```
both %>%
  ggvis(~delay) %>%
  layer_histograms(binwidth =
     input_slider(1, 10, value = 5))
```

Reactive properties

```
both %>%
  ggvis(~precip, ~delay) %>%
  layer_points(opacity := input_slider(0, 1))
```

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()
```

Using ggvis in interactive Shiny docs

The future

- Subvisualizations (faceting)
- Zooming and panning
- ggplot2 feature parity
- Performance improvements
- Scriptable file output (without browser)

More information at
http://ggvis.rstudio.com/

Mailing list:
https://groups.google.com/forum/#!forum/ggvis

THANKS FOR ATTENDING

Learn more about Shiny Server & RStudio

rstudio.com/products/shiny/shiny-server/

&

rstudio.com/products/rstudio/