Storytelling with data

10 | Technologies and tools of interactive data-driven, visual design

Scott Spencer | Columbia University

course overview | main course deliverables

open-source technology stack for interactive, data-driven graphics

interactive technology stack, components and relationships — ***click a technology below to learn more***

browser

event listeners

DOM

html
css
grid
svg
canvas

js

scrollama.js
d3.js
plotly.js
p5.js
react.js
jQuery.js
DataTables.js
three.js
...

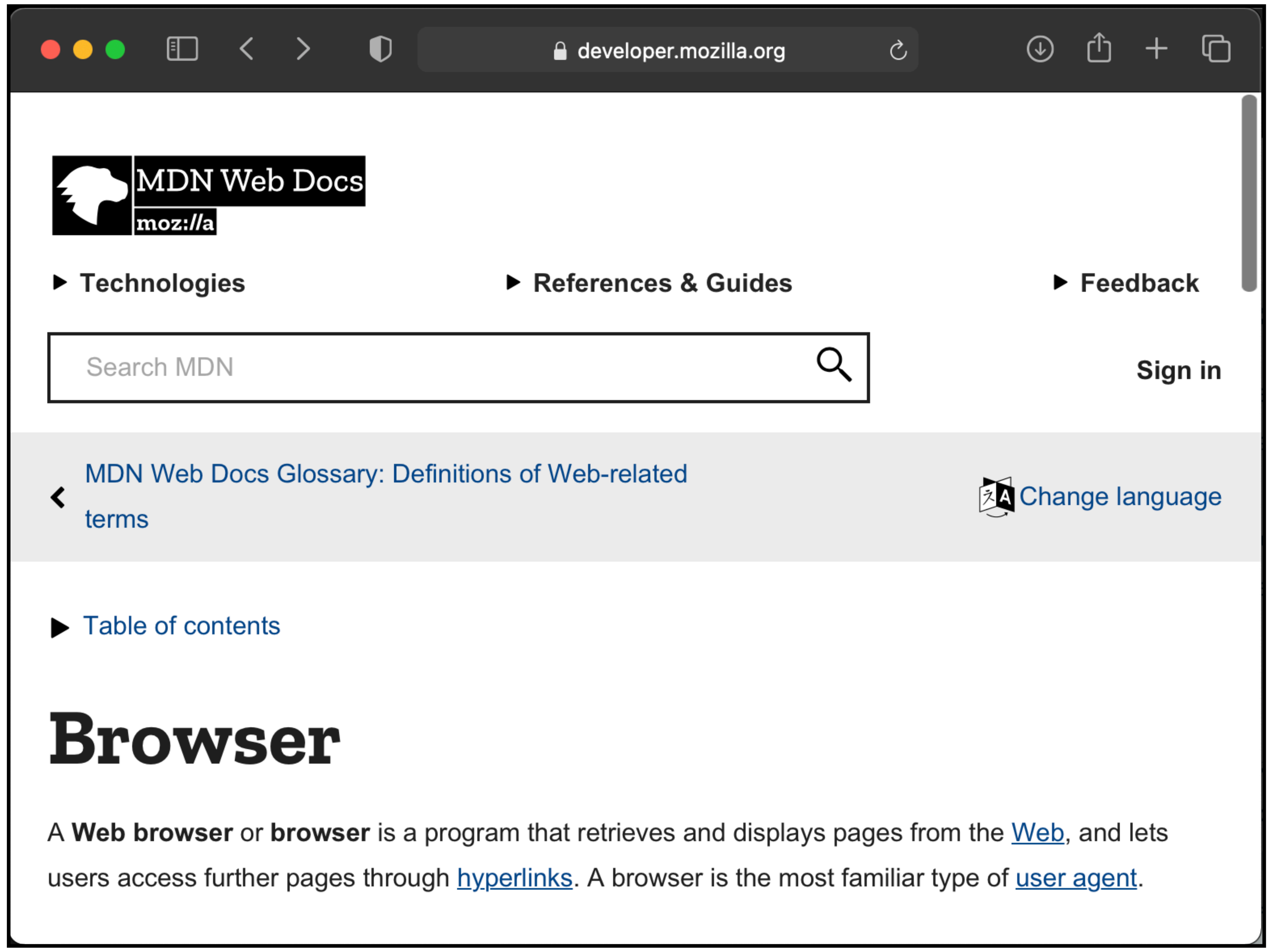
R

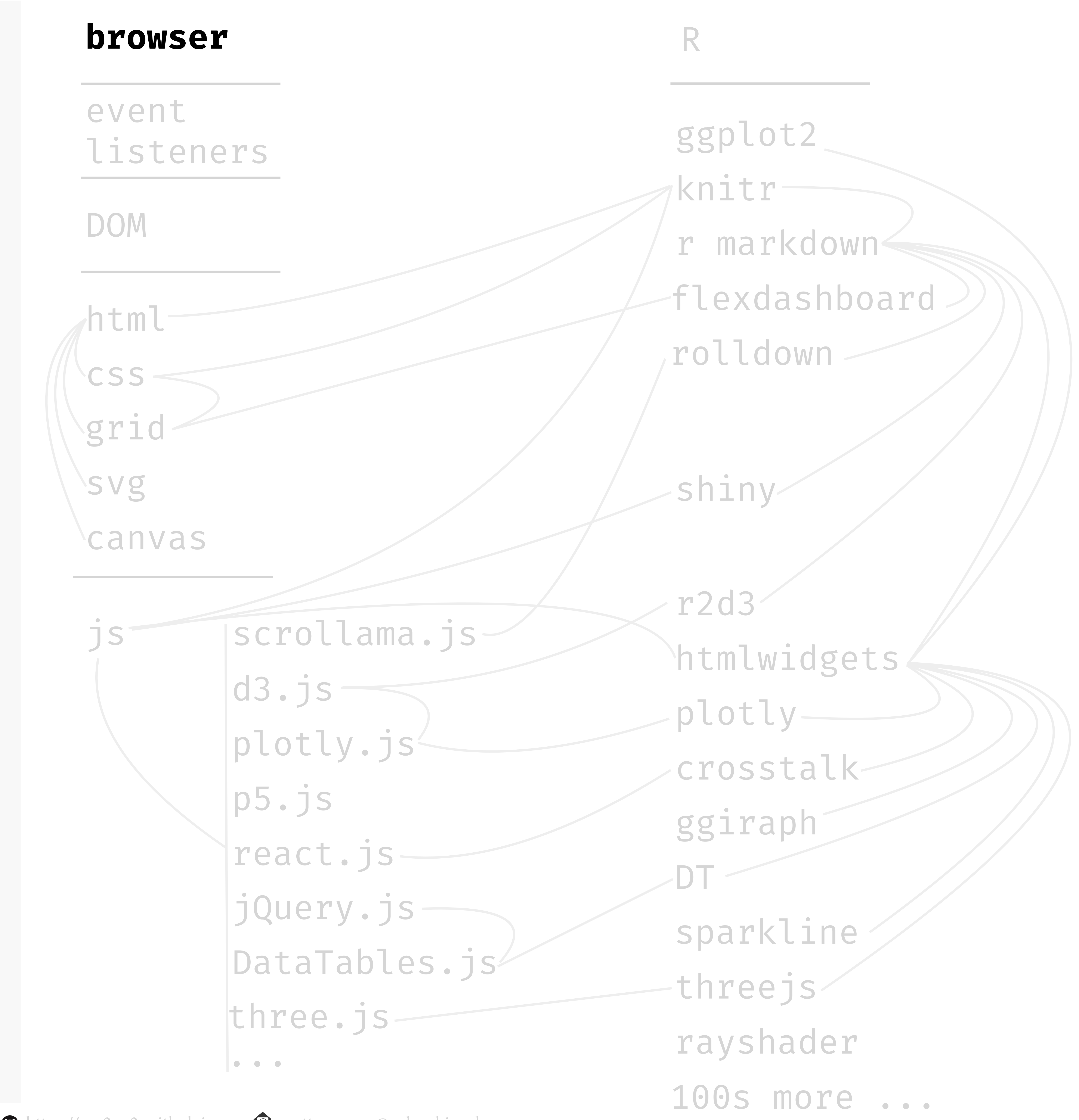
ggplot2
knitr
r markdown
flexdashboard
rolldown

shiny

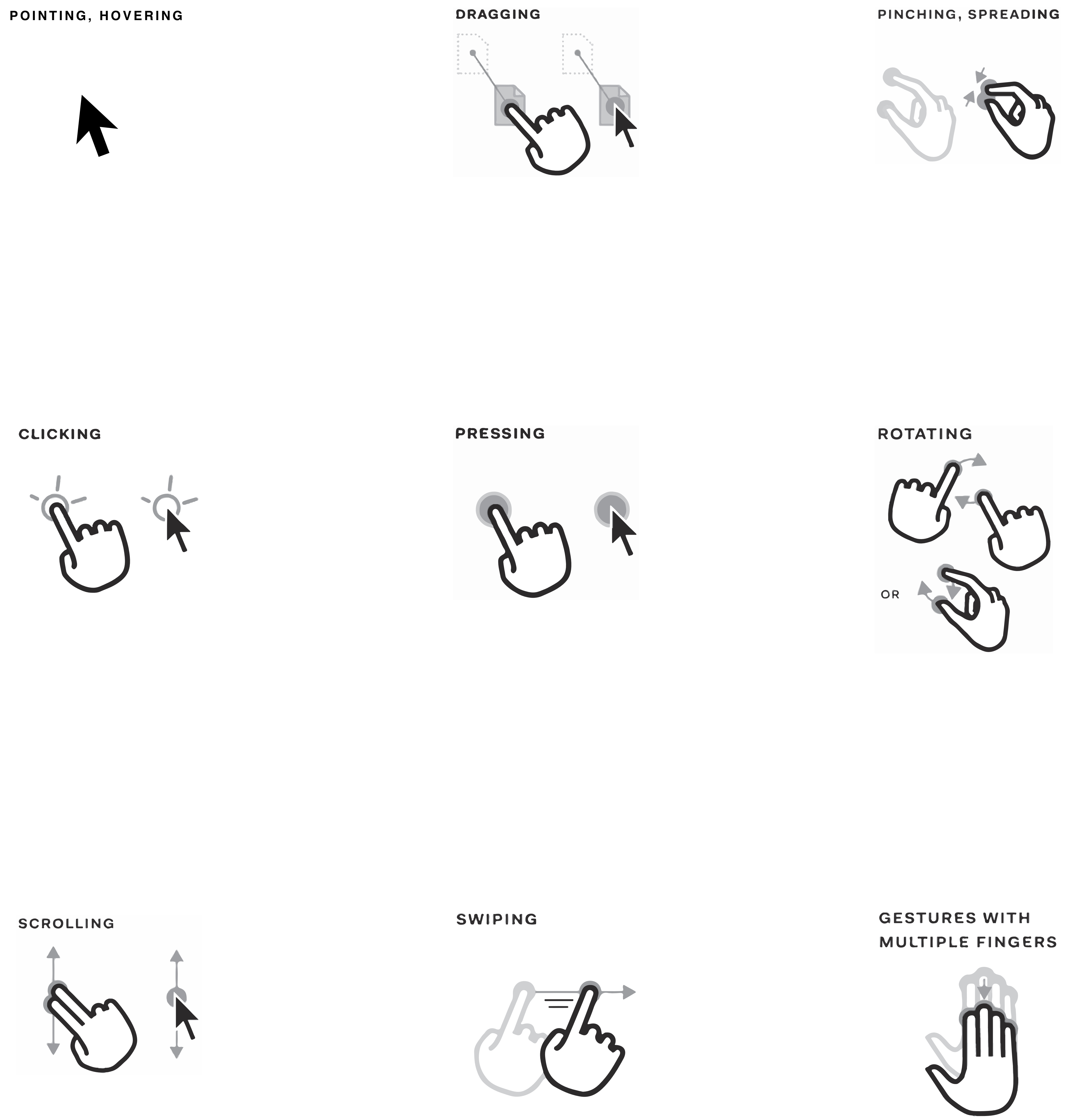
r2d3
htmlwidgets
plotly
crosstalk
ggiraph
DT
sparkline
threejs
rayshader
100s more ...

interactive technology stack, browsers parse various code to render content and respond to actions





interactive technology stack, **actions trigger events, for which page elements can be bound to listen**

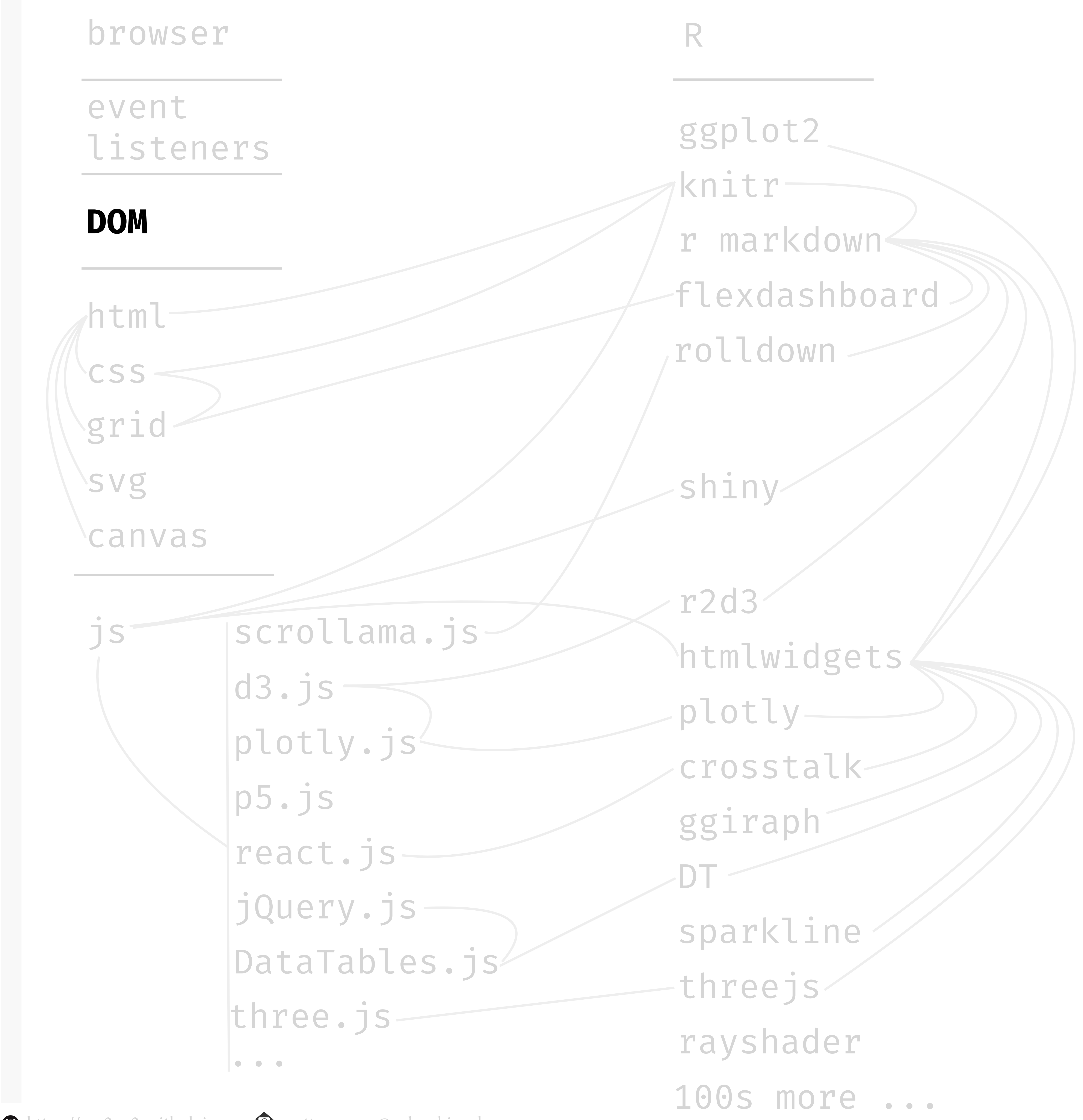


 a web page includes several languages, each has a purpose

web page structure

(Interactive) web **pages** all begin and end with <html> and </html> respectively. contain a **head** and **body**. Content between <body> and </body> is shown inside the main browser window Before the <body> element you will often see a <head> element. This contains information *about* the page, rather than information that is shown within the main part of the browser window. You will usually find a <title> element and <script> (not shown below) element(s) inside the <head> element. Notice how tag enclosures create a *tree-like structure* we can traverse — that's the Document Object Model, or **DOM**.

```
<html>
  <head>
    <title>This is the Title of the Page</title>
  </head>
  <body>
    <h1>This is in the Body of the Page</h1>
      <p>Anything within the body of a web page is
         displayed in the main browser window.</p>
  </body>
</html>
```



`html` elements

Added to the content of a **page** to describe its structure. An element consists of an *opening* and *closing* **tag** and its **content**. Opening tags can carry **attributes**.

The `<p></p>` below instructs the **browser** to structure the content as a paragraph. There are **many** pre-defined **tag types** and attributes, and we can define our own.

OPENING TAG

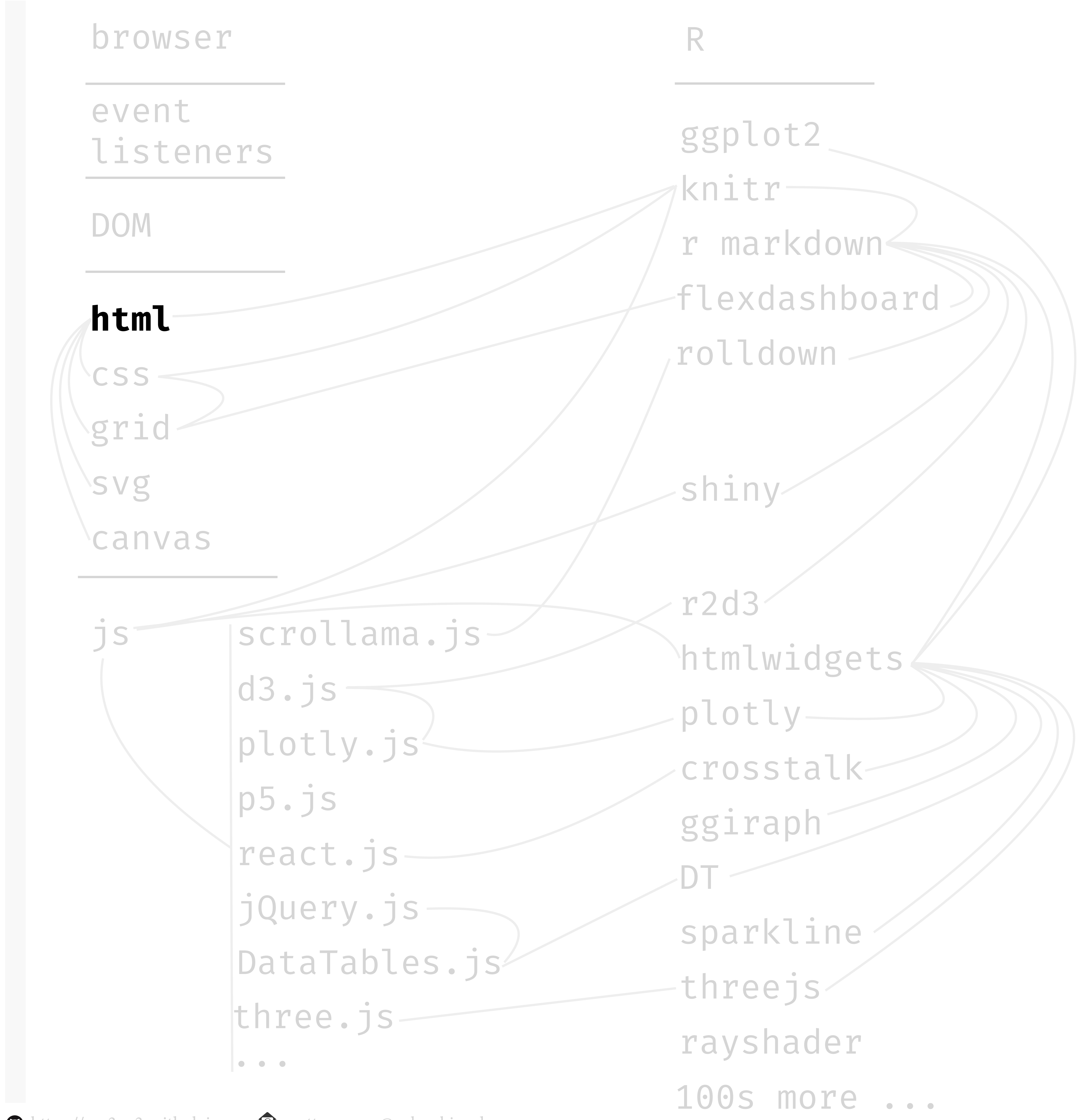
CLOSING TAG

```
<p class="cycling_team">Education First</p>
```

ATTRIBUTE NAME

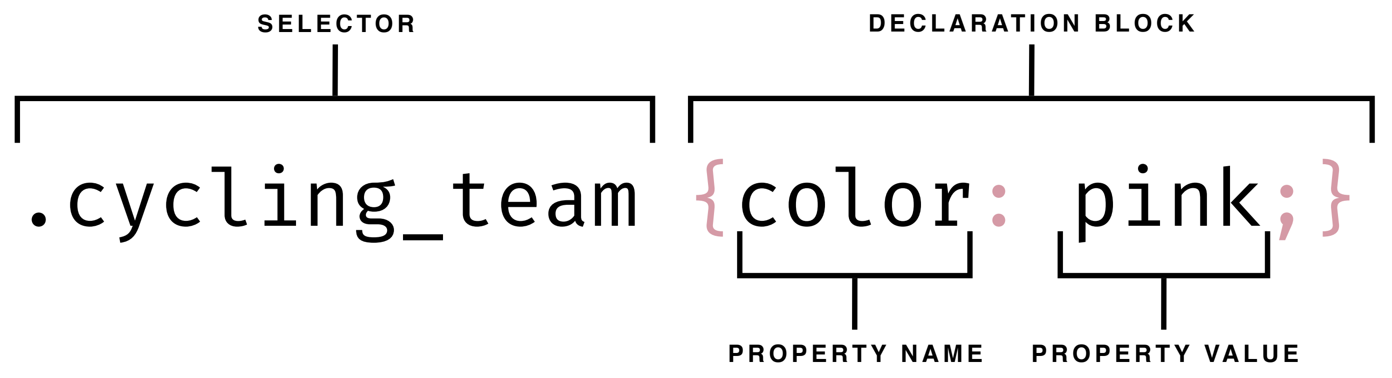
ATTRIBUTE VALUE

CONTENT

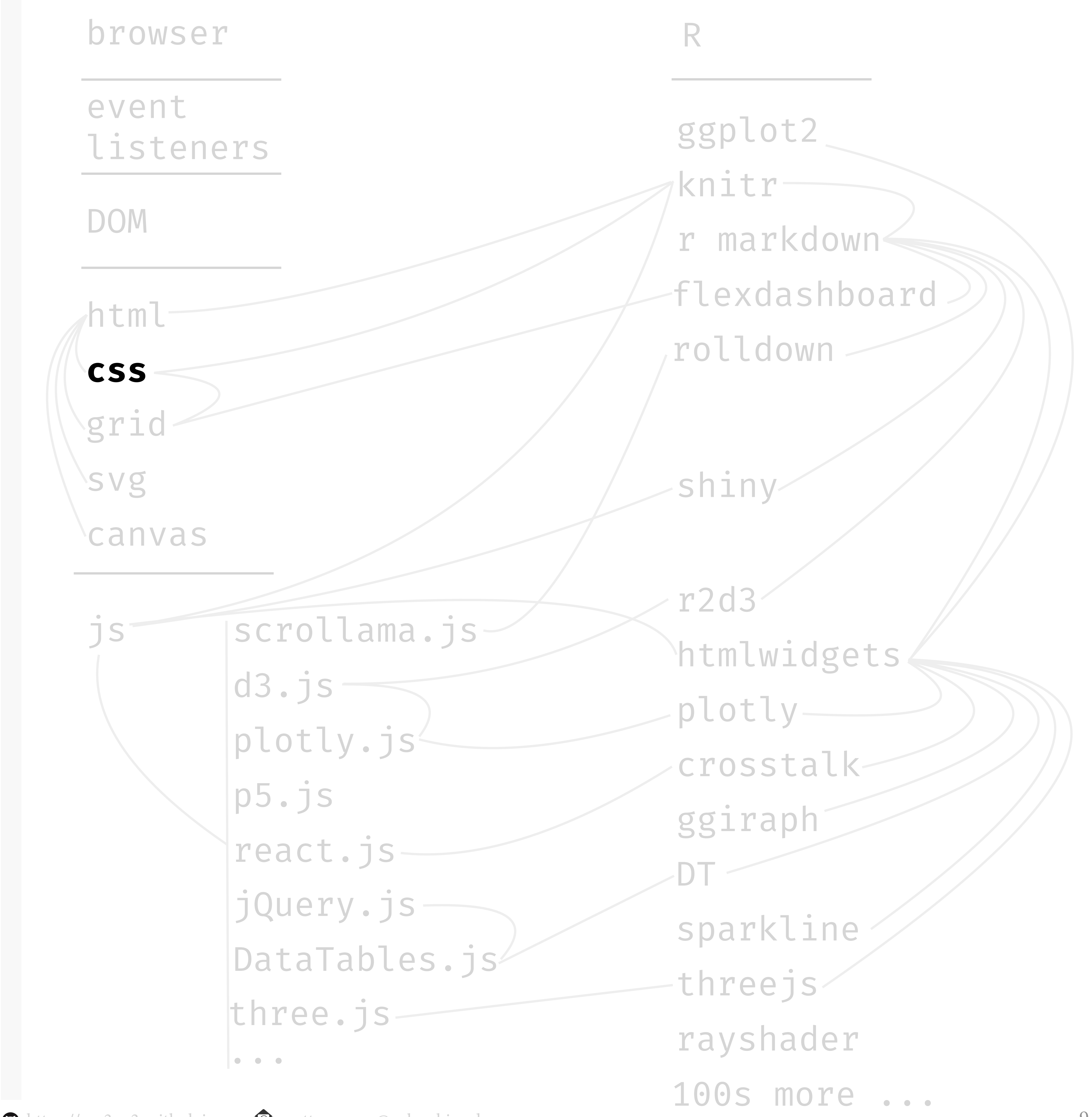


css rules

Indicates how the contents of one or more elements should be displayed in the browser. Each rule has a selector and a declaration block. The **selector** indicates to which element(s) the rule applies. Each **declaration** block specifies one or more properties and corresponding values.Below, applying the **class** .cycling_team to a tag as an attribute, it will **color** the text a **pink** hue. CSS rules are specified within <style> elements.




```
<style>
  .cycling_team {
    color: pink;
  }
</style>
```

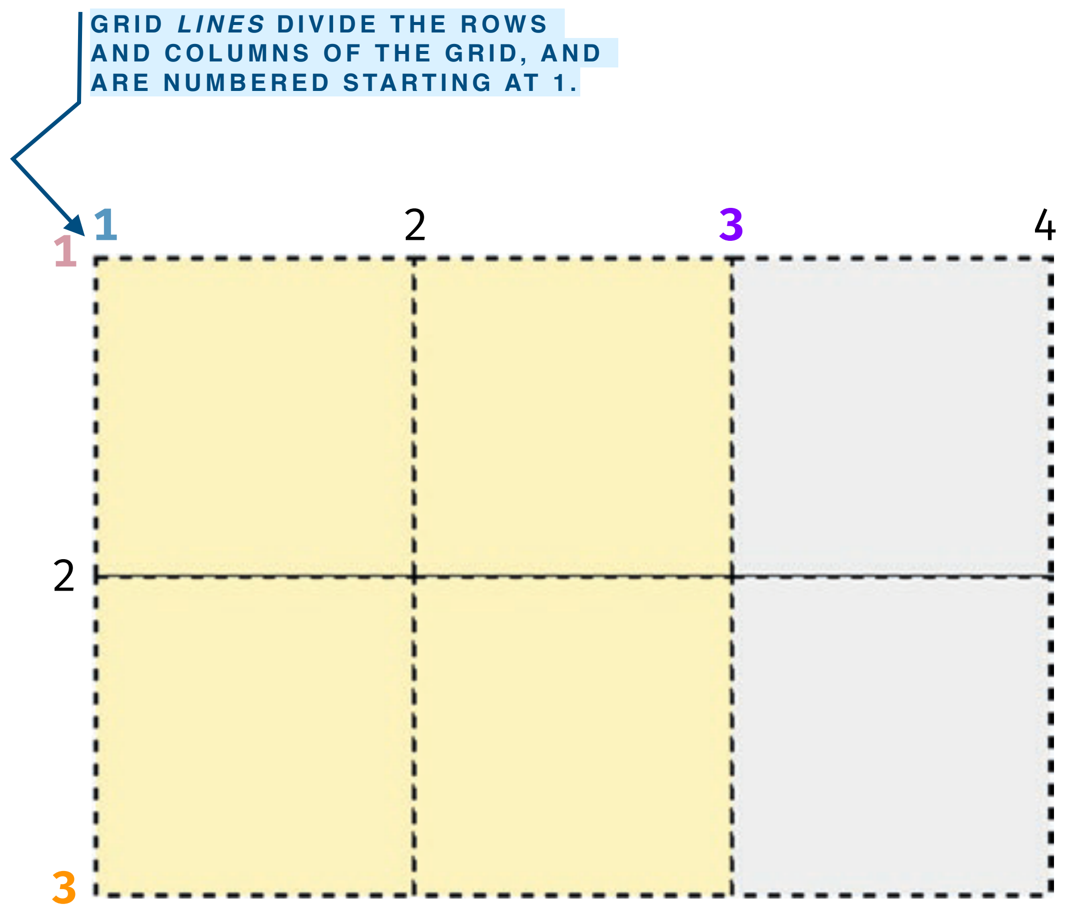


`css` grid

We've discussed and practiced using **grids** earlier in the semester to help us organize text and data graphics for memos, proposals, and information graphics. The `html` language includes grids we can specify using **tags**. Below, we define a **class** `.gridlayout` and in that specify `{display: grid;}` and related properties. Then, we use our `class` attributes in **divider** tags `<div></div>` to format the content. The example below displays a 2 x 3 grid of **cells**, each with a size specified and placed in row major order.

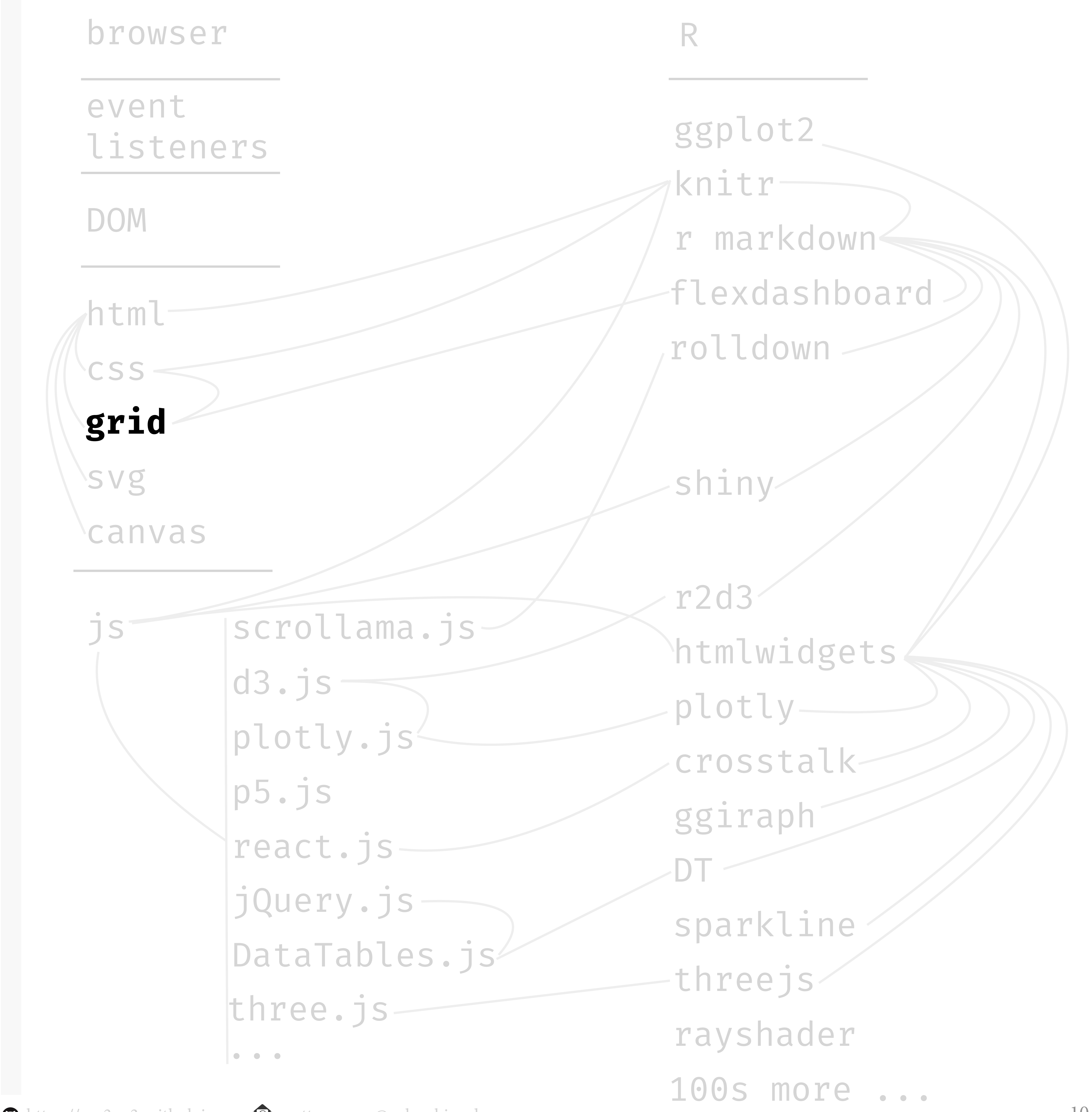
WE PLACE OUR CLASSES FOR THE GRID BETWEEN THE TWO CSS <style> TAGS.

```
<style>
  .gridlayout {
    display: grid;
    grid-template-columns: 1fr 1fr;
    grid-template-rows: 5rem 5rem;
    gap: 5px;
  }
  .item {
    background: lightgray;
    text-align: center;
  }
  .area {
    grid-column: 1 / 3;
    grid-row: 1 / 3;
    background: lightyellow;
    text-align: center;
  }
</style>
```



TO ADD CONTENT, WE PLACE OUR CONTENT BETWEEN <div> TAGS, AND FORMAT USING OUR CLASSES WE DEFINED.

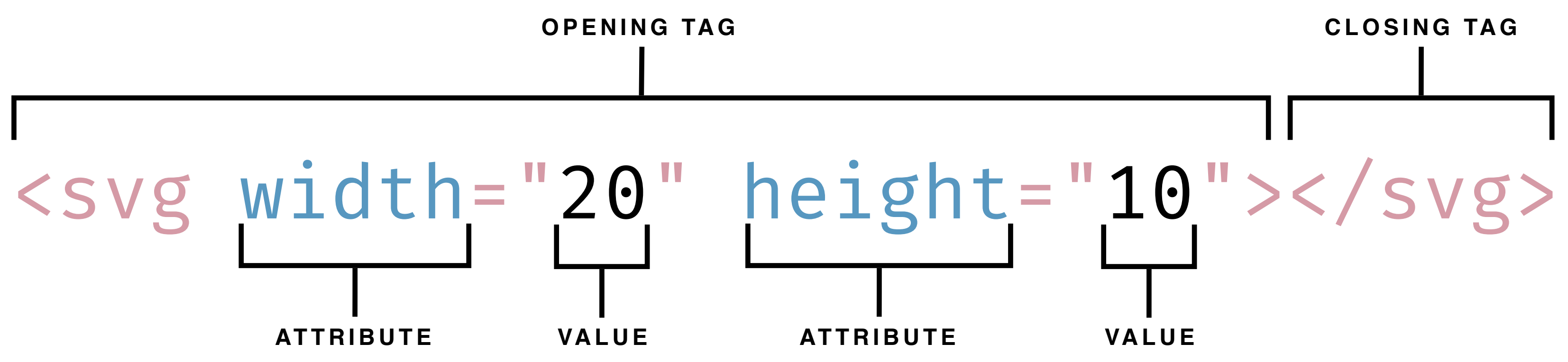
```
<div class="gridlayout">
  <div class="area"></div>
  <div class="item"></div>
  <div class="item"></div>
</div>
```

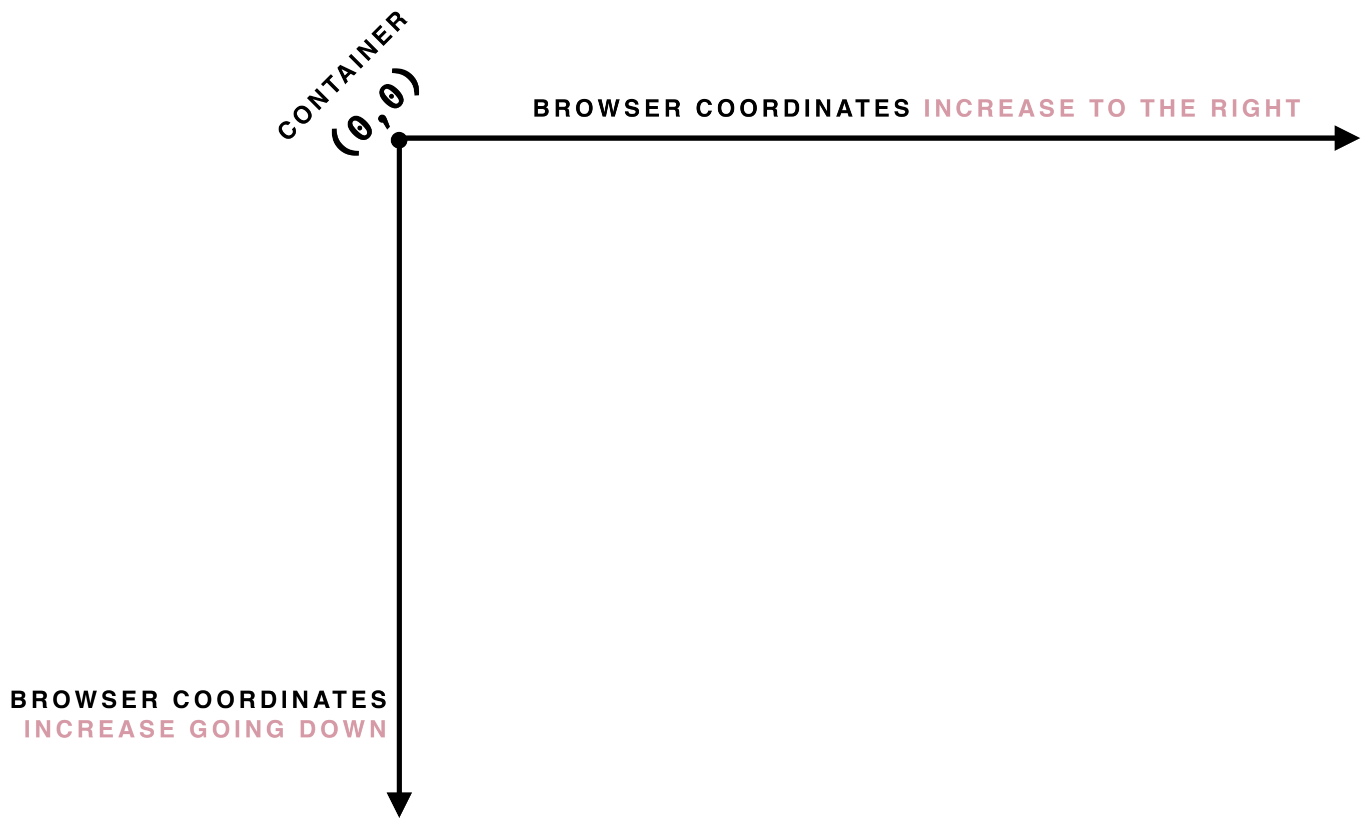


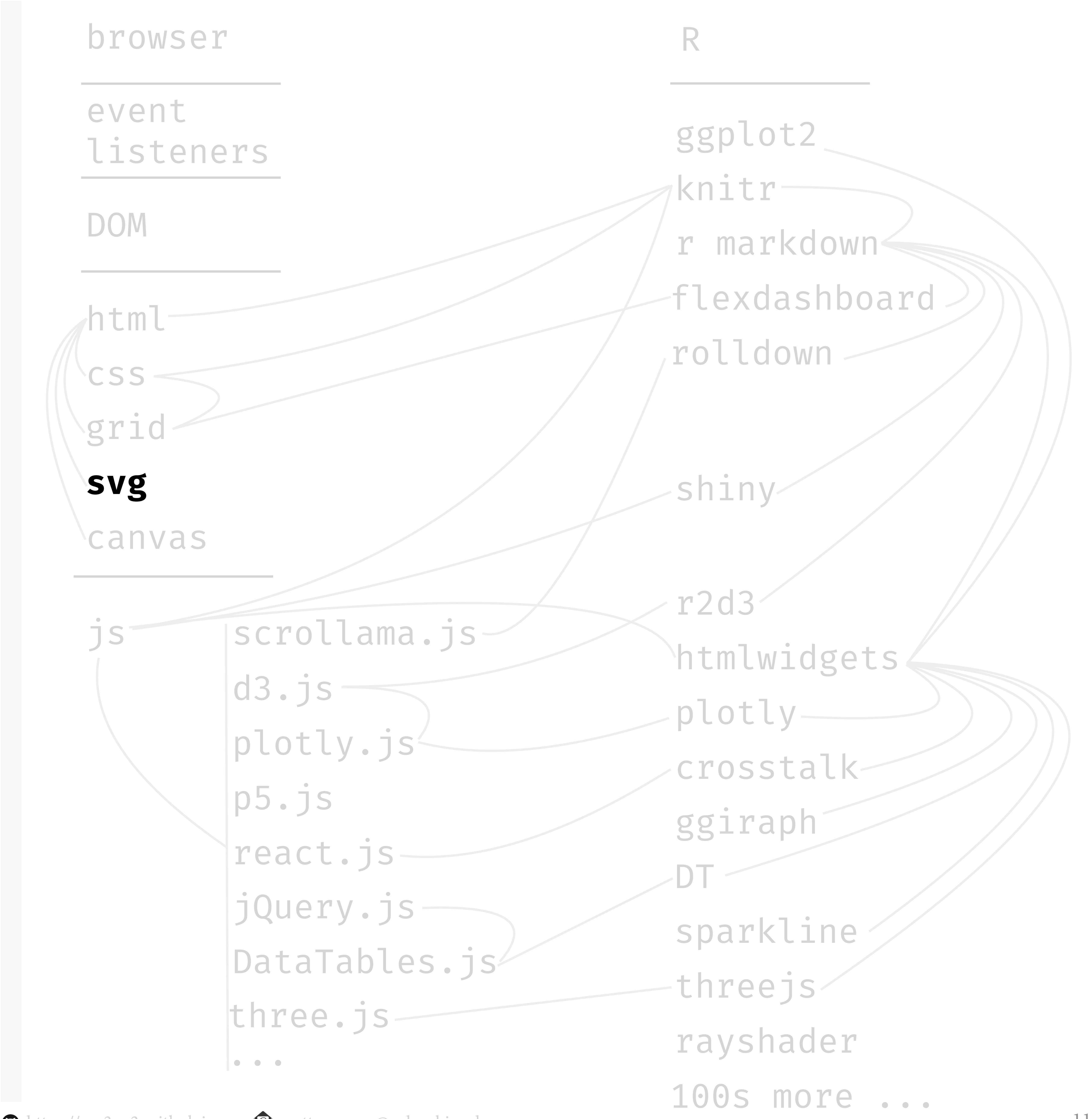
svg

Scalable vector graphics — **svg** — are human-readable descriptions of **shapes** or **paths** that the browser can display. As we've discussed, *enlarging* **vector** graphics, unlike raster-based graphics, will not reduce **resolution**. Together these paths and shapes comprise a graphic.

We put them in the `html` document body between **svg** `<svg>` and `</svg>` tags. Shapes I commonly use include the circle `<circle>`, rectangle `<rect>`, text `<text>`, path `<path>`, and group `<g>`. We can edit vector graphic shapes using software like Adobe Illustrator or Inkscape, too.





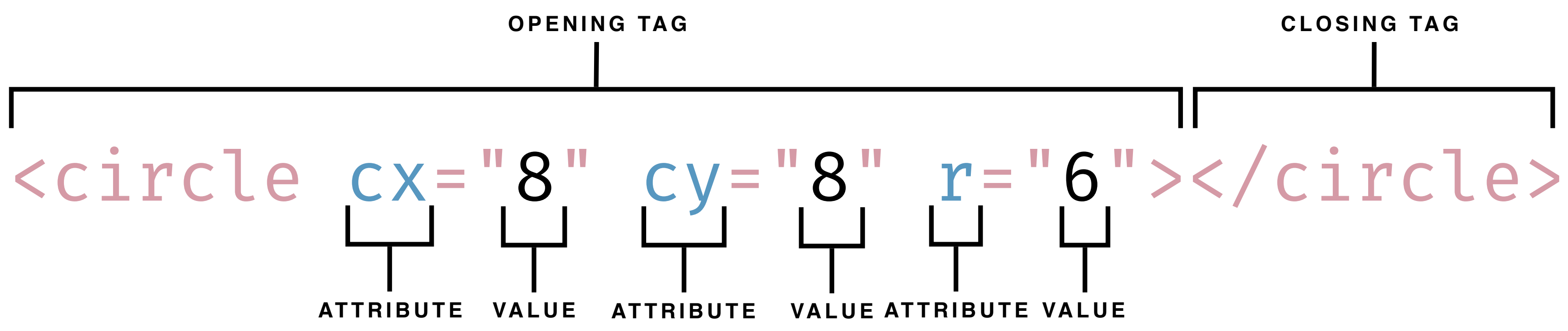


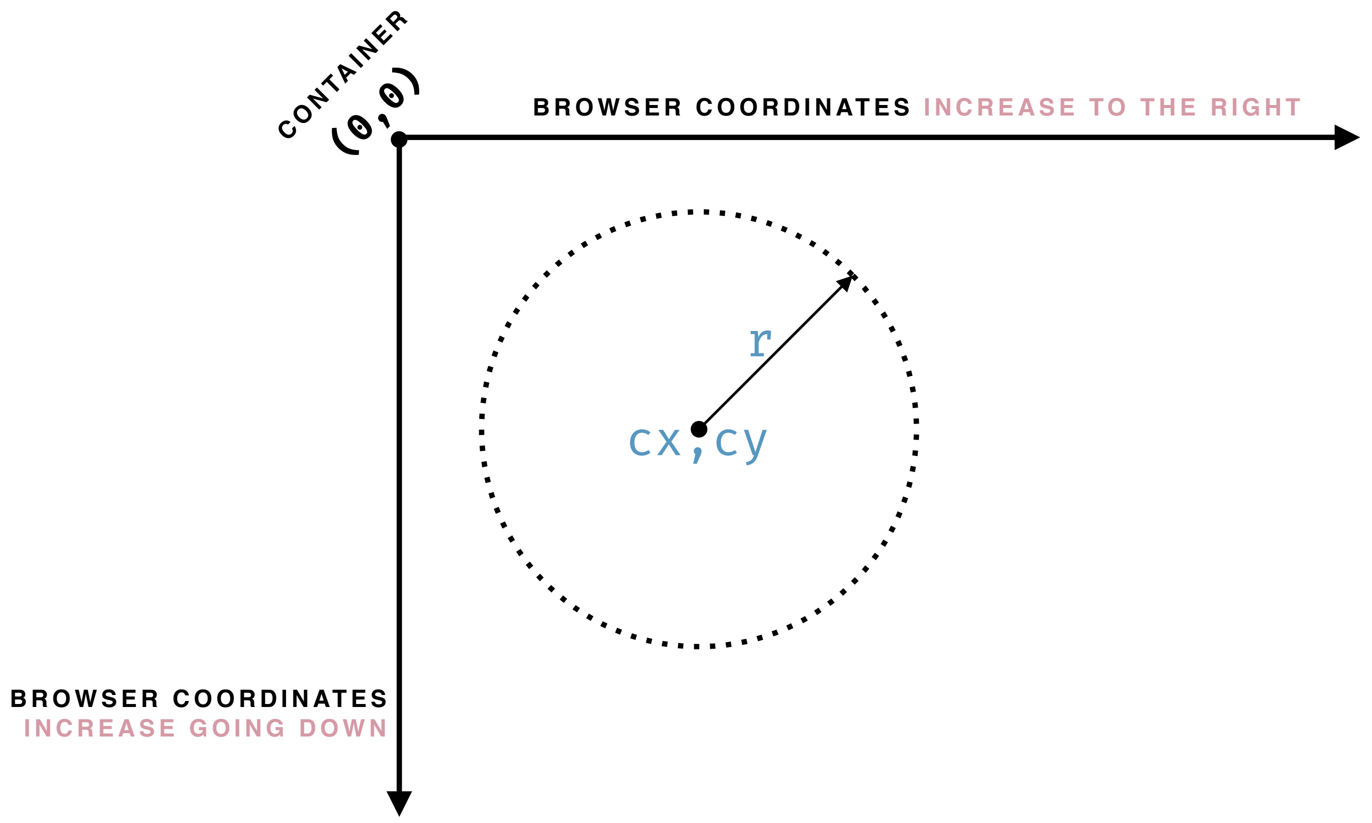
interactive technology stack, **draw shapes within `svg` tags, a content layer**

svg

Scalable vector graphics — **svg** — are human-readable descriptions of **shapes** or **paths** that the browser can display. As we've discussed, *enlarging* **vector** graphics, unlike raster-based graphics, will not reduce **resolution**. Together these paths and shapes comprise a graphic.

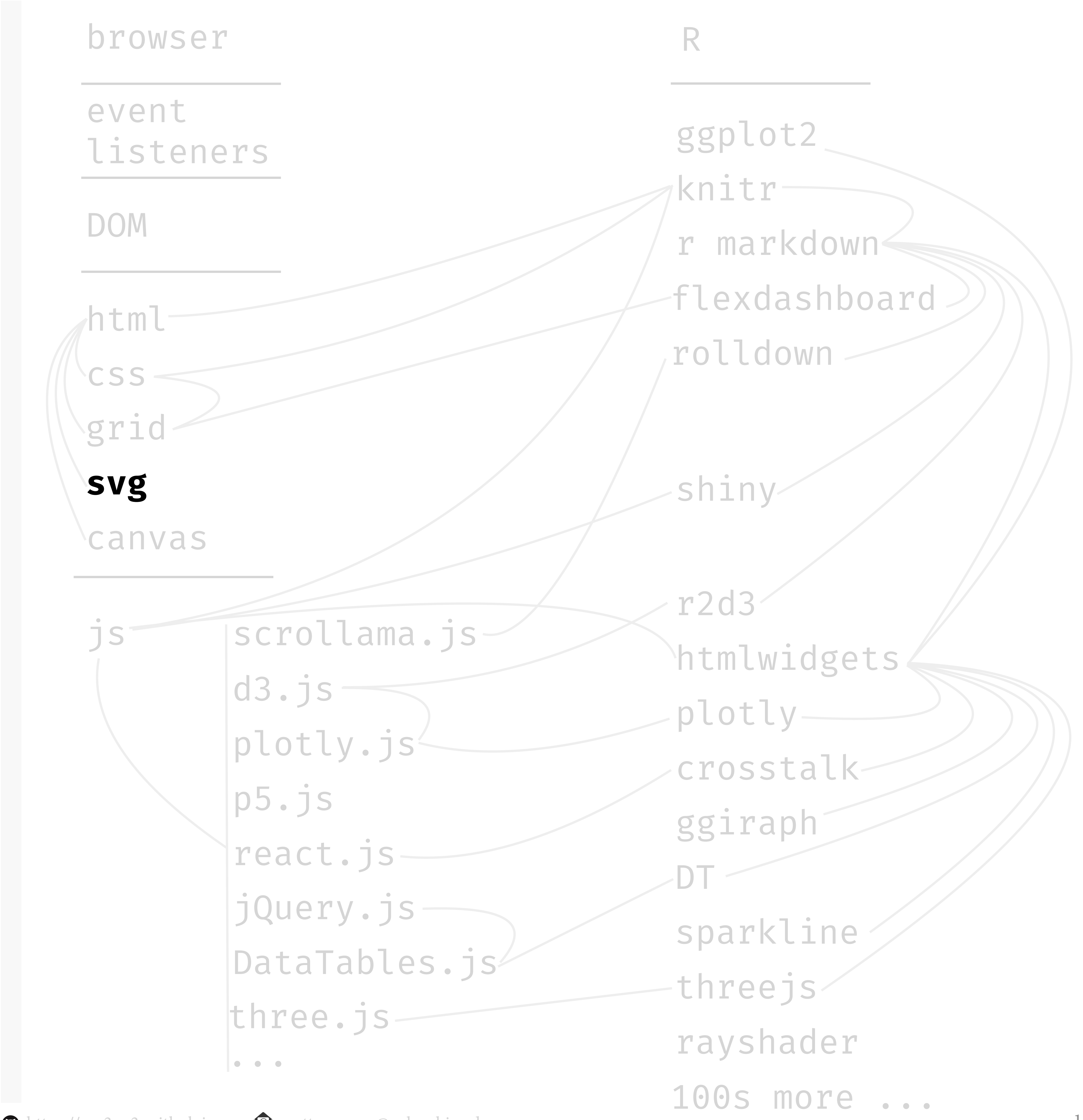
We put them in the `html` document body between svg `<svg>` and `</svg>` tags. Shapes I commonly use include the **circle** `<circle>`, rectangle `<rect>`, text `<text>`, path `<path>`, and group `<g>`. We can edit vector graphic shapes using software like Adobe Illustrator or Inkscape, too.





OTHER COMMON SHAPE ATTRIBUTES

```
stroke
stroke-width
stroke-opacity

fill
fill-color
fill-opacity
```

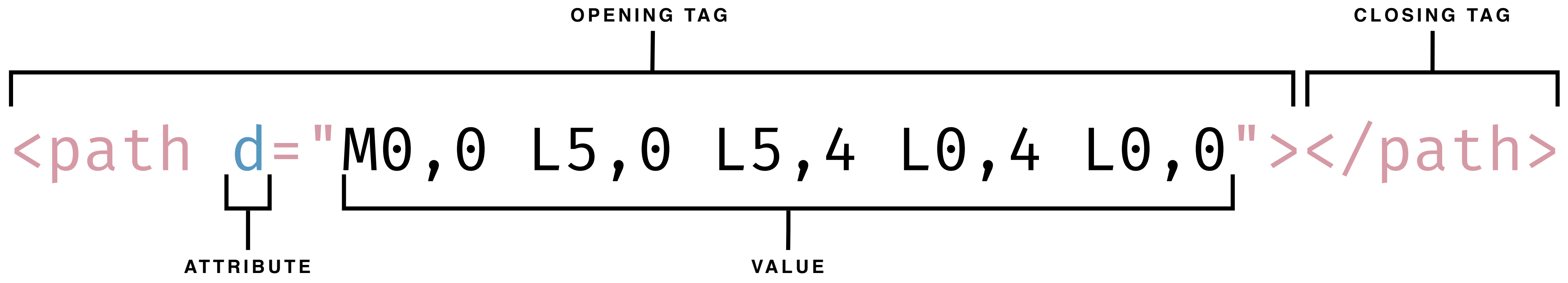


 draw shapes within `svg` tags, a content layer

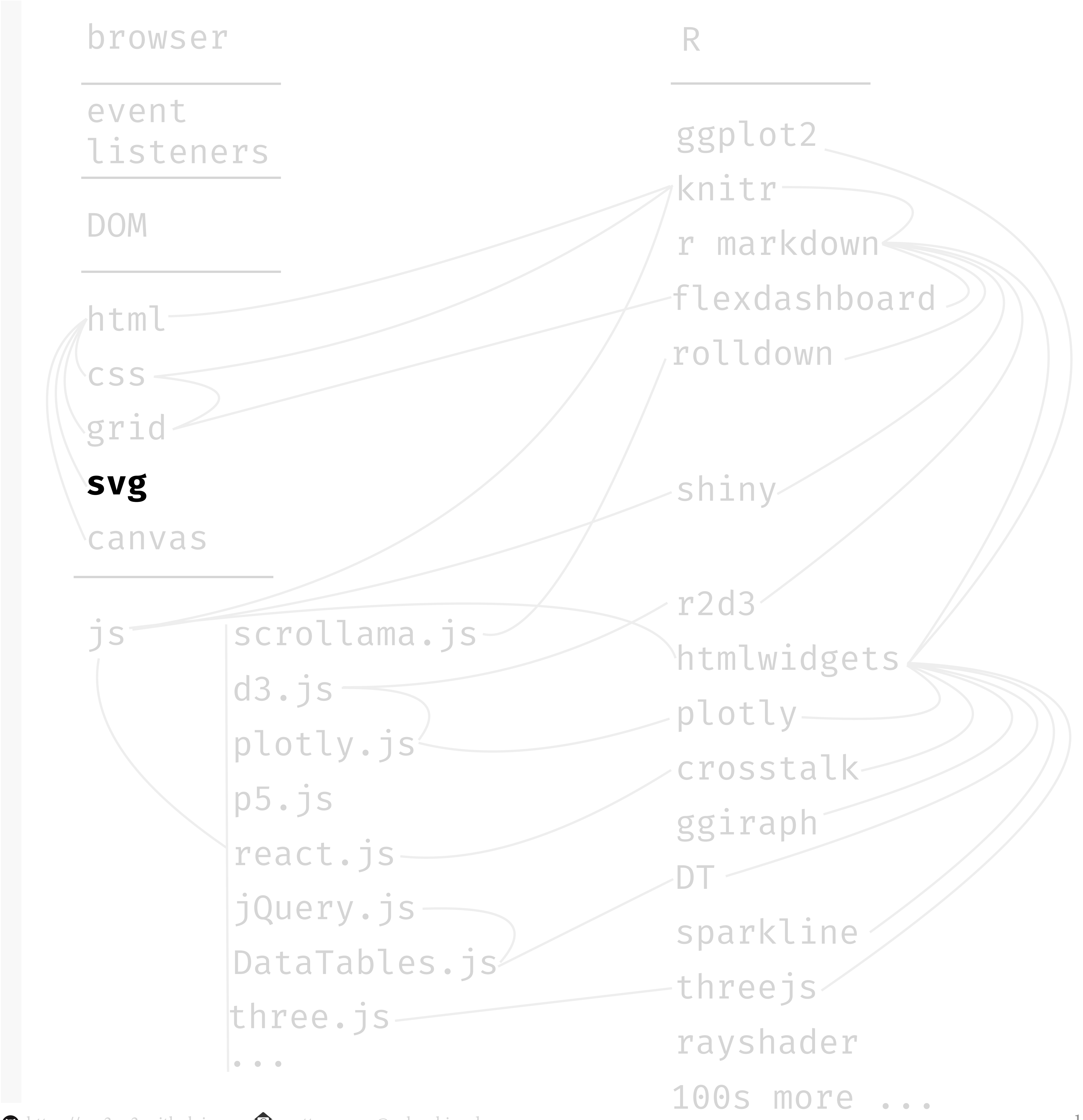
svg

Scalable vector graphics — **svg** — are human-readable descriptions of **shapes** or **paths** that the browser can display. As we've discussed, *enlarging* **vector** graphics, unlike raster-based graphics, will not reduce **resolution**. Together these paths and shapes comprise a graphic.

We put them in the `html` document body between svg `<svg>` and `</svg>` tags. Shapes I commonly use include the circle `<circle>`, rectangle `<rect>`, text `<text>`, **path** `<path>`, and group `<g>`. We can edit vector graphic shapes using software like Adobe Illustrator or Inkscape, too.



COMMAND	SYNTAX	MEANING
MOVE TO	Mx,y	location coordinate x, y where the drawing starts.
LINE TO	Lx,y	draw straight path from previous coordinate x, y to this coordinate x, y.
CURVE TO	Cx,y x,y x,y	draw curve path from previous coordinate x, y using two control points x, y and x,y to this coordinate x, y.



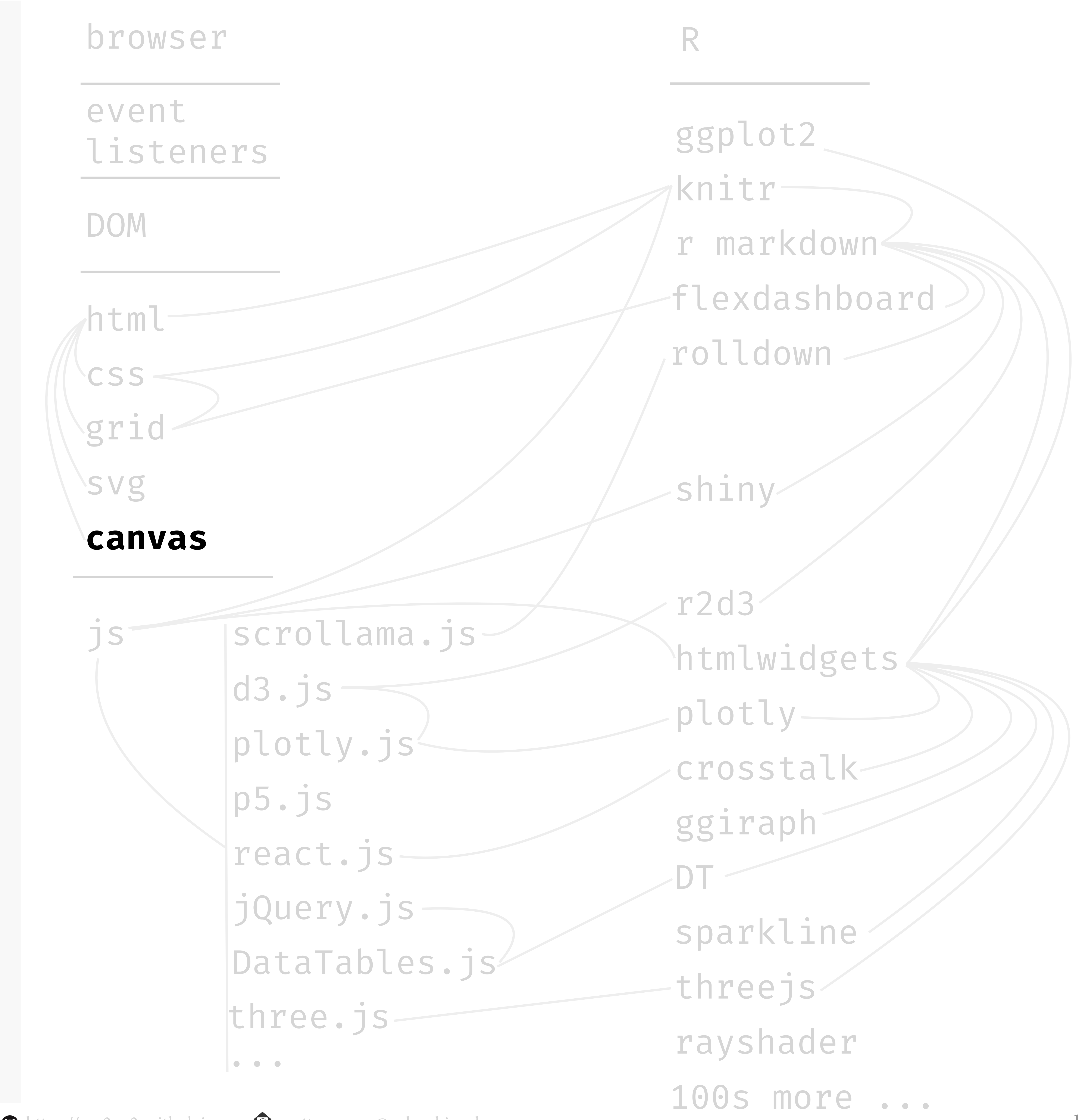
canvas

When **performance** drawing `svg` shapes becomes an issue—which may occur on slower computers with 1,000 to 10,000 shapes, more with today's computers—we gain performance by switching to **raster** graphics. For raster graphics, we draw **pixels** on **canvas**, which we specify within `html` using the `<canvas></canvas>` tag. From pixels, we cannot select shapes or paths like we can with `svg` graphics, and **resolution drops** upon **zooming** into the canvas. To edit rasters, we're better off using something like Photoshop.

OPENING TAG CLOSING TAG

```
<canvas id="c1" width="20" height="10"></canvas>
```

ATTRIBUTE VALUE ATTRIBUTE VALUE ATTRIBUTE VALUE



interactive technology stack, respond to events by changing content or style with `js`, a behavior layer

JavaScript

We can ***bind* elements** to **events** that, upon happening, ***trigger javascript code***, which in turn can modify content: html elements and attributes, svg or canvas, or css styles. Really it **can modify anything in the DOM**. As with R packages that abstract and ease our application of specialized functionality, easing the burden of writing code, many **javascript libraries** are available to do the same. Those listed to the right are particularly important for interactive data visualization, but many more not listed are also available.

```
element.onevent = functionName;
```

ELEMENT — DOM element node to target

EVENT — Event bound to node(s) preceded by word "on".

CODE — Name of function to call with no parenthesis following it.

```
function functionName() {
  // code to do something
}
```

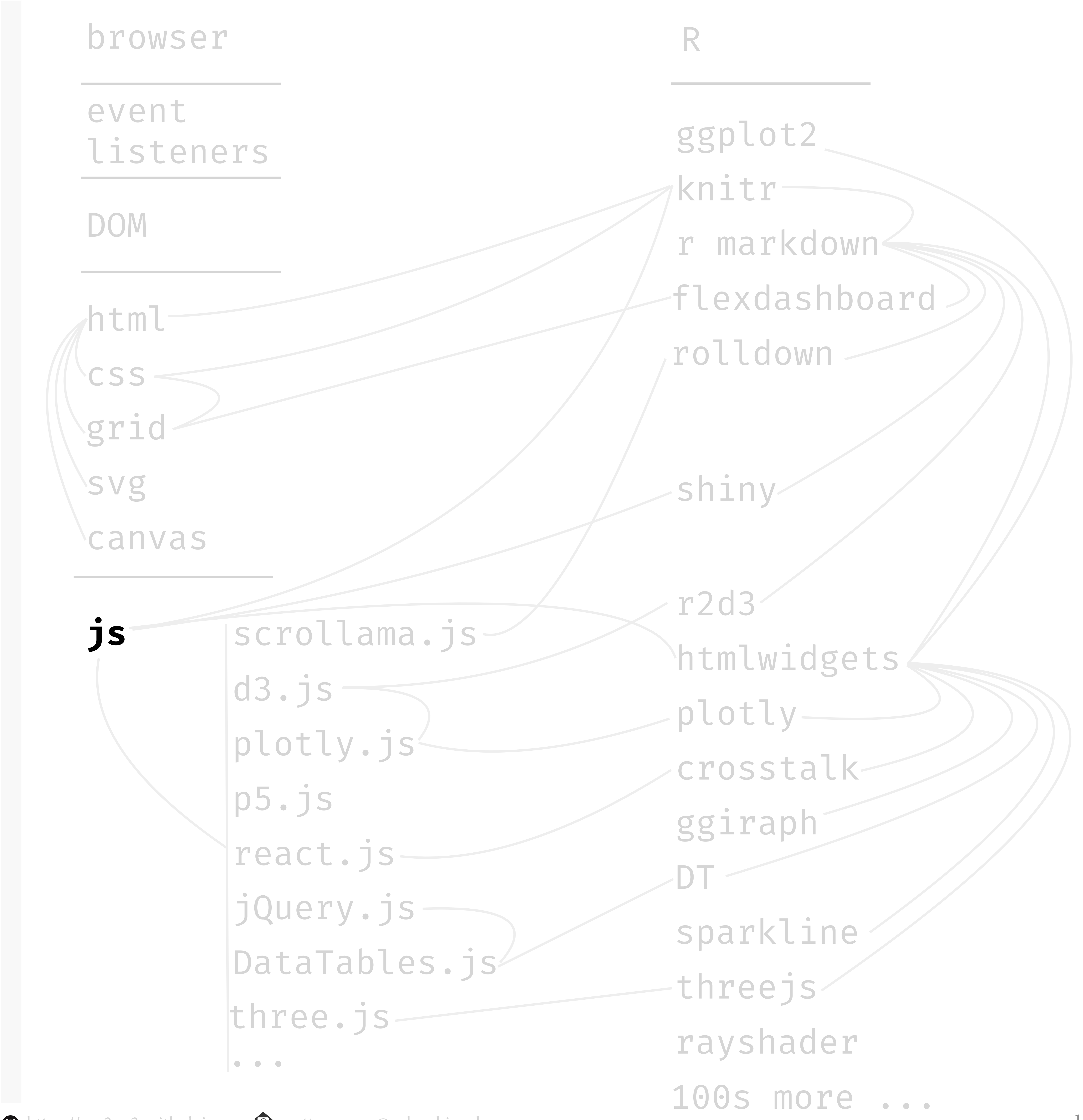
THE CODE STARTS BY DEFINING THE NAMED FUNCTION.

A REFERENCE TO THE DOM ELEMENT IS OFTEN STORED IN A VARIABLE.

```
var el = document.getElementById('element');
el.on_______ = functionName;
```

THE EVENT NAME IS PRECEDED BY THE WORD "ON".

THE FUNCTION IS CALLED BY THE EVENT HANDLER ON THE LAST LINE, BUT THE PARENTHESIS ARE OMITTED.

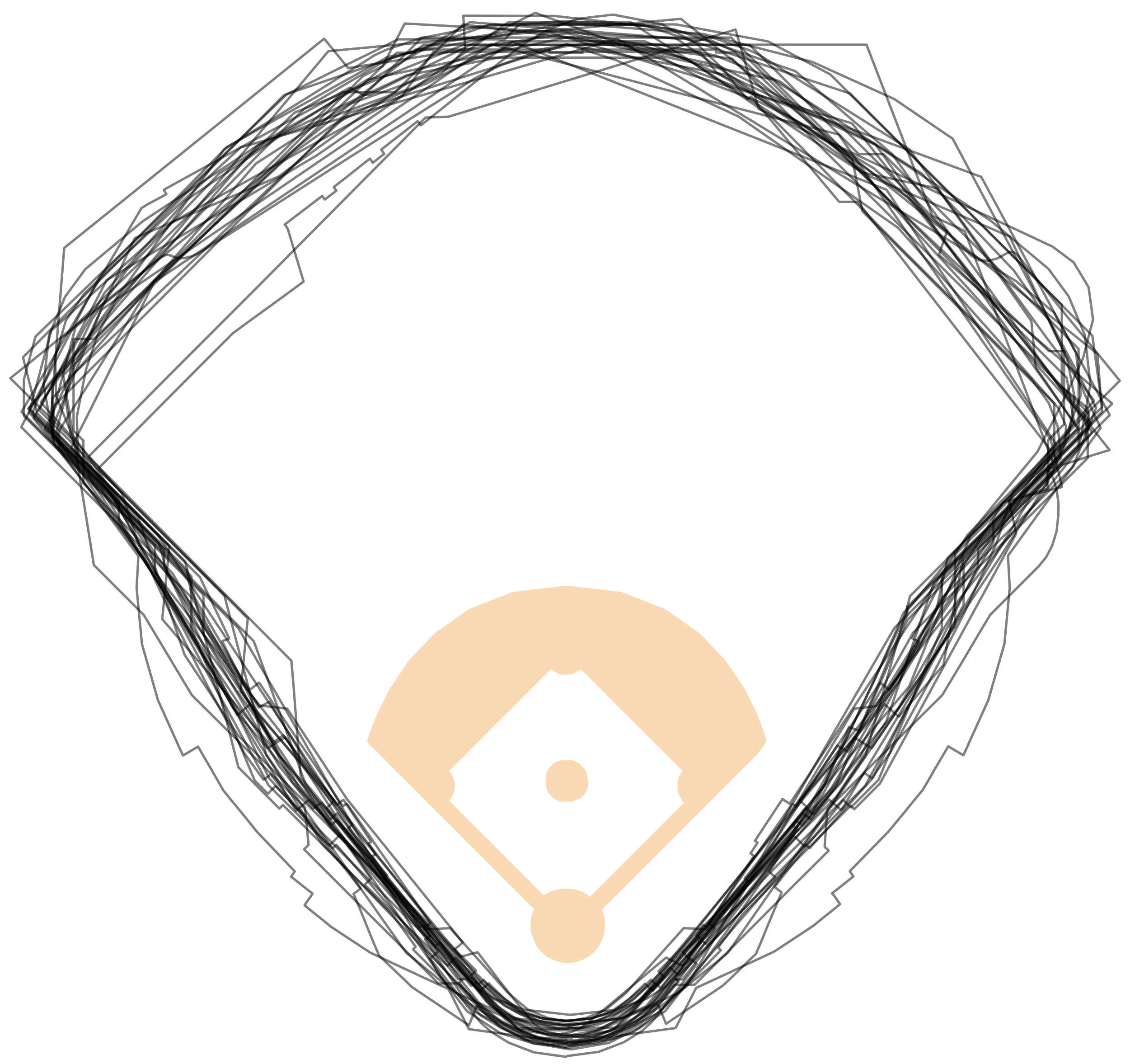


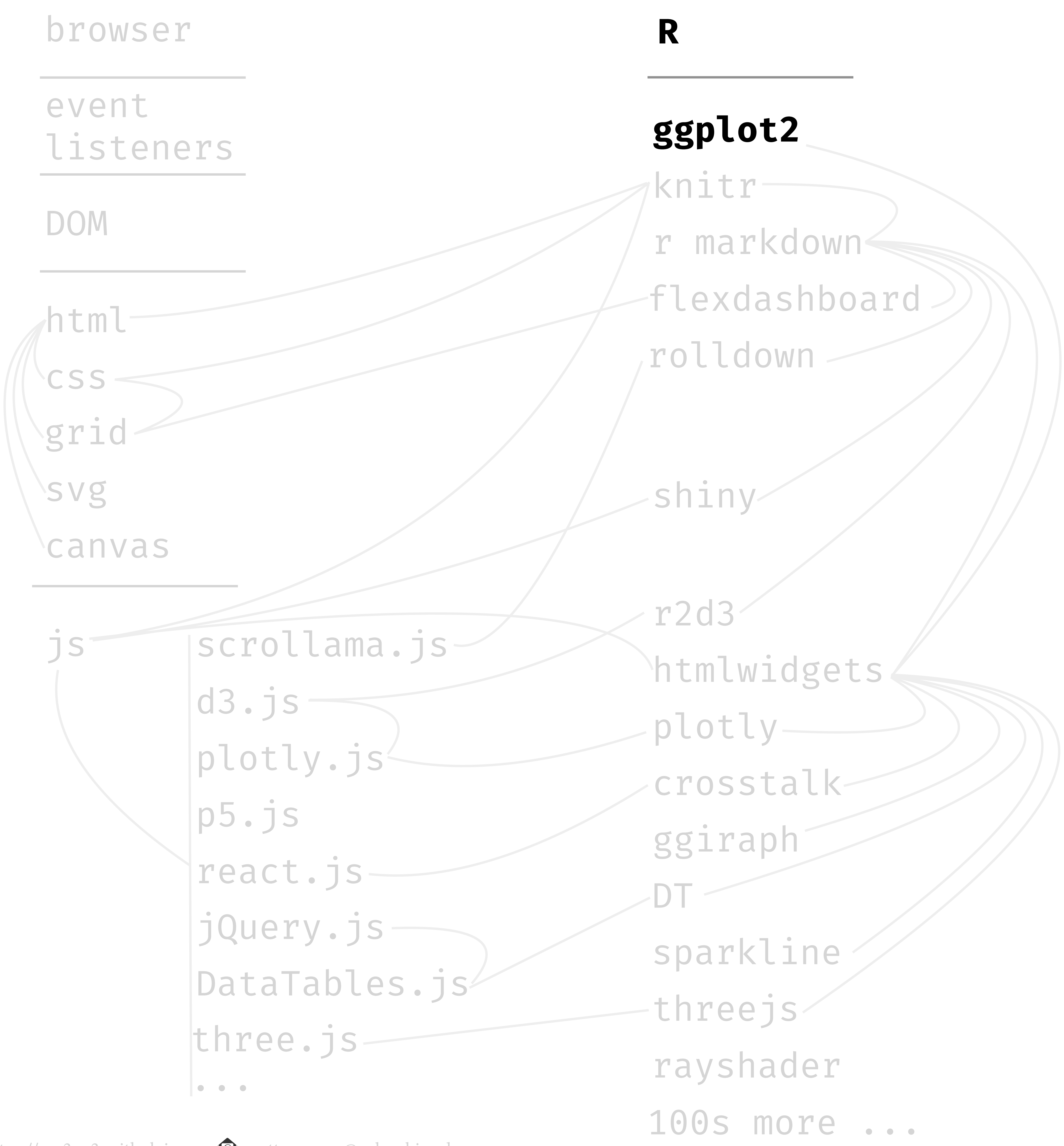
content creation for this
interactive technology stack

`ggplot2`

The **grammar of graphics** — implemented in `R` as `ggplot2` — is among the most powerful coding libraries for creating static graphics. We've already seen how to use a complementary package with `ggplot2` to add animation: `gganimate`, a grammar of animated graphics. With similar complementary packages, we can specify **interactivity**. Let's see a static version of a class example, the 30 baseball outfields, then make it interactive using `ggiraph`.

30 baseball outfields — *static* version





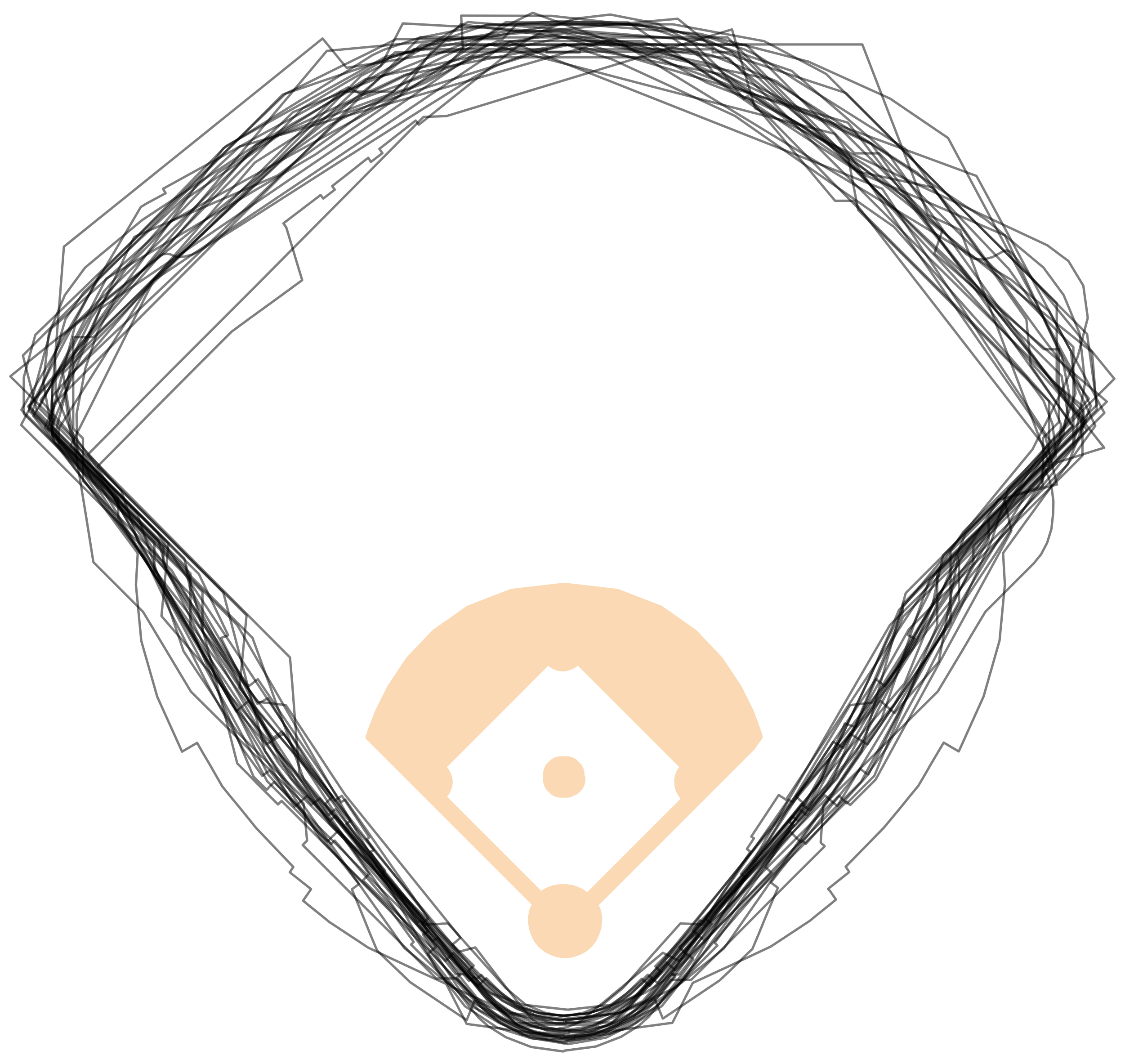
tools for interactive content, several R packages transform ggplot2 into interactive graphics

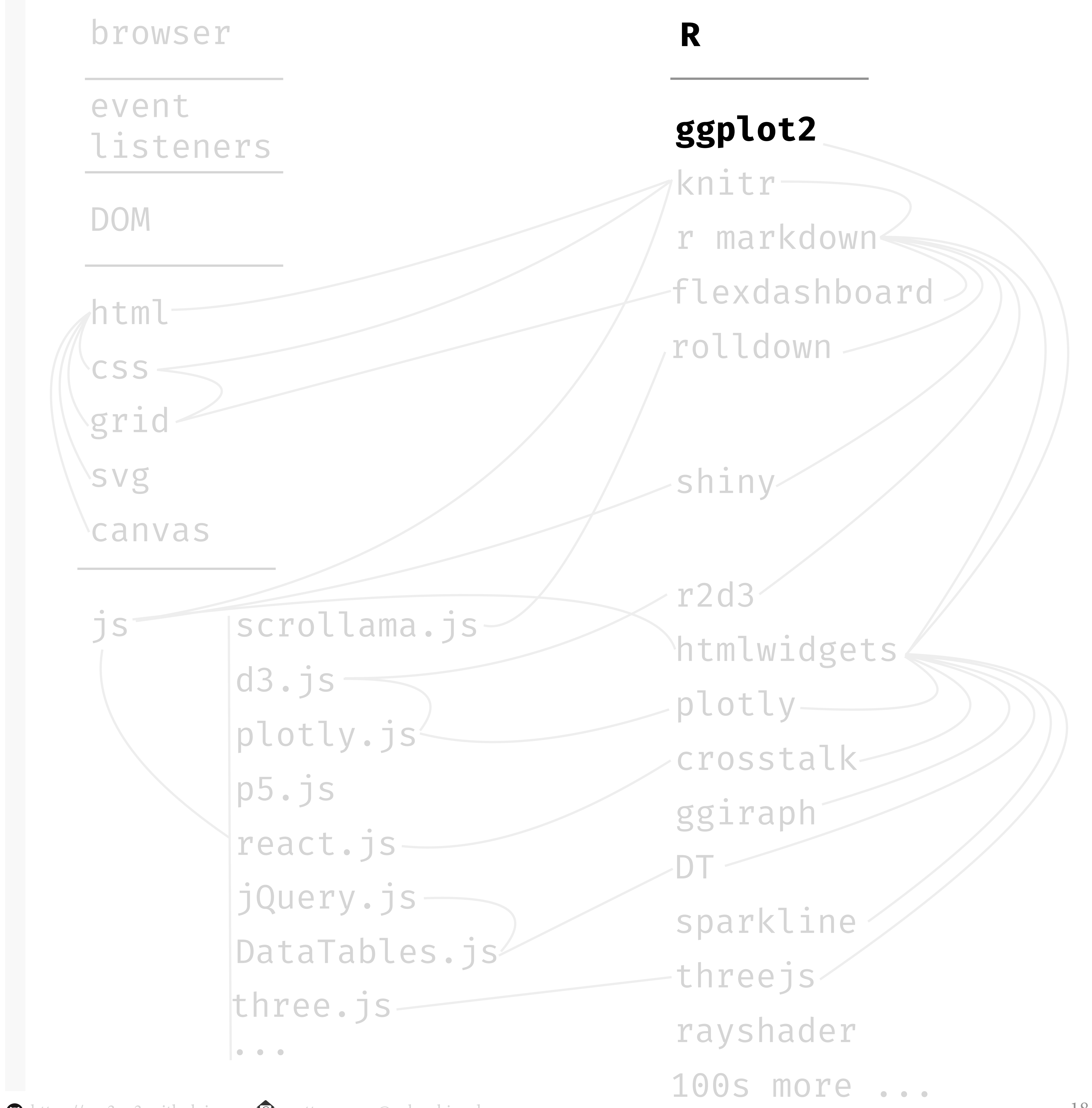
ggplot2

The **grammar of graphics** — implemented in R as ggplot2 — is among the most powerful coding libraries for creating static graphics. We've already seen how to use a complementary package with ggplot2 to add animation: gganimate, a grammar of animated graphics. With similar complementary packages, we can specify **interactivity**. Let's see a static version of a class example, the 30 baseball outfields, then make it interactive using ggiraph.

```
gg_boundaries <-
  ggplot() +
  theme_void() +
  coord_equal() +
  geom_path(
    data = subset(
      fields,
      is_infield == FALSE),
    mapping = aes(
      x = xsh,
      y = ysh,
      group = id),
    color = '#000000',
    alpha = 0.5) +

  geom_polygon(
    data = subset(
      fields,
      is_infield == TRUE),
    mapping = aes(
      x = xsh,
      y = ysh,
      group = id),
    fill = '#FAD9B4',
    color = '#FAD9B4')
```

30 baseball outfields — *static* version





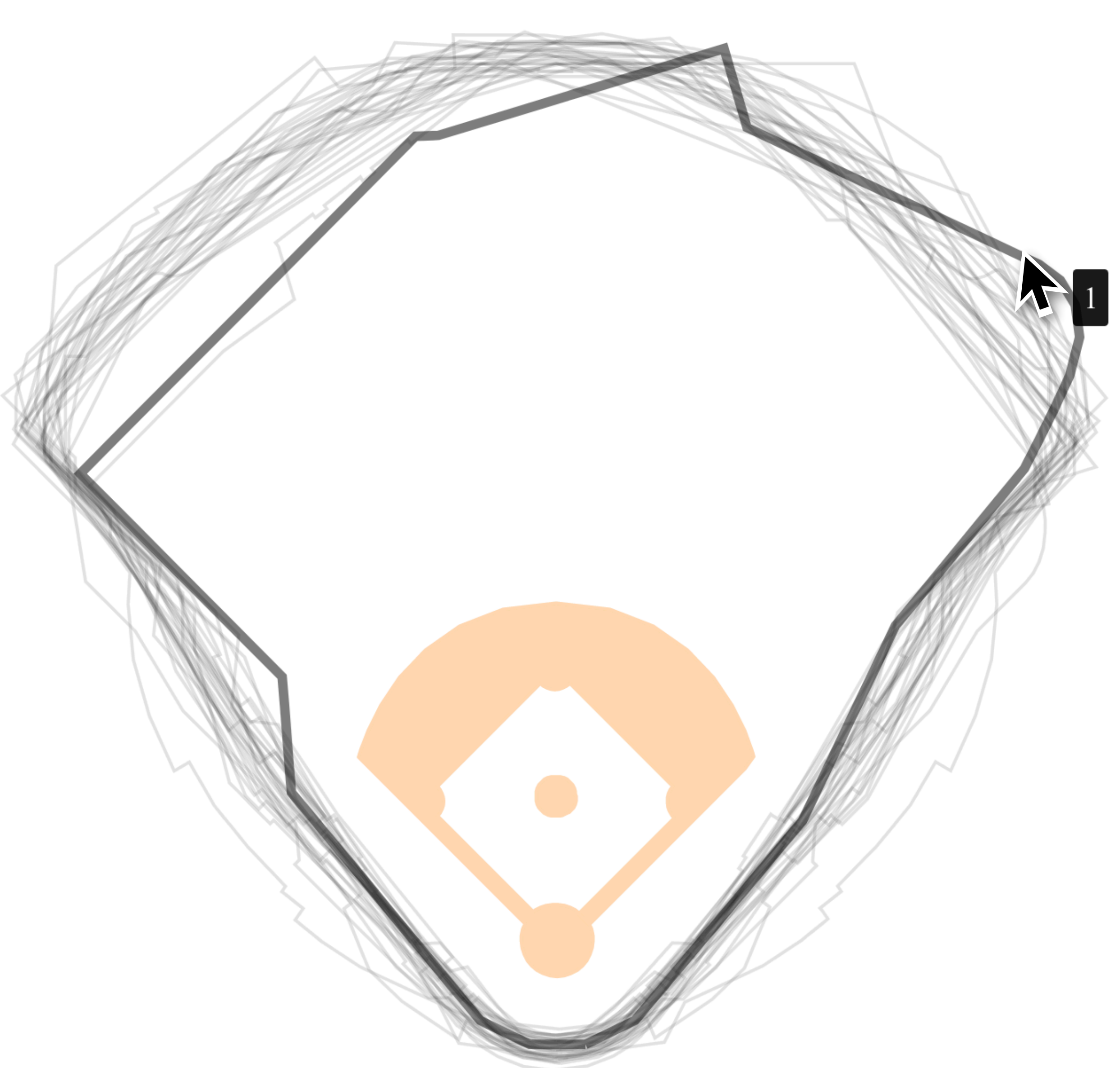
tools for interactive content, several R packages transform ggplot2 into interactive graphics

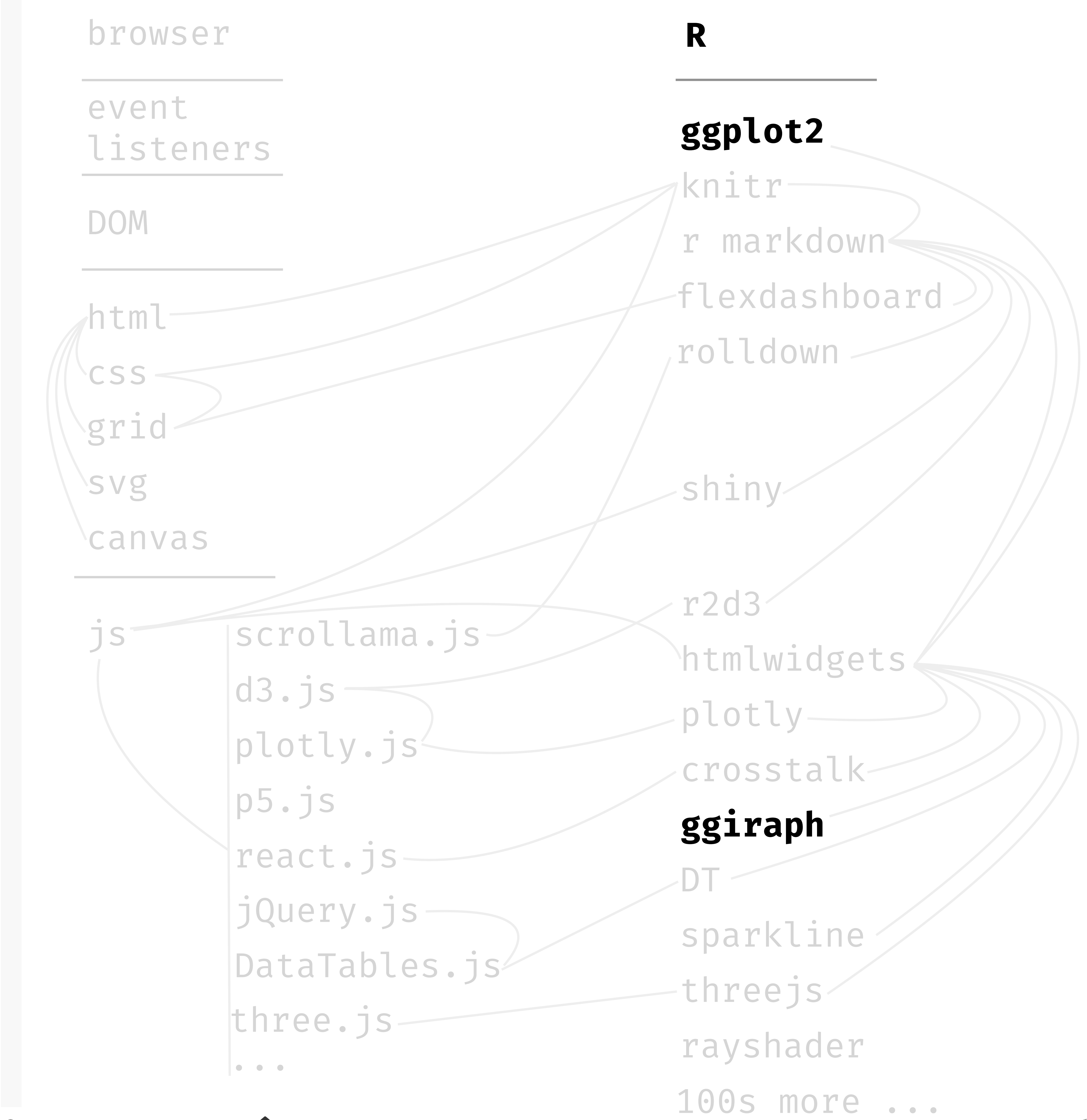
ggplot2 + ggiraph

The **grammar of graphics** — implemented in R as ggplot2 — is among the most powerful coding libraries for creating static graphics. We've already seen how to use a complementary package with ggplot2 to add animation: gganimate, a grammar of animated graphics. With similar complementary packages, we can specify **interactivity**. Let's see a static version of a class example, the 30 baseball outfields, then make it interactive using ggiraph.

```
gg_boundaries <-
  ggplot() +
  theme_void() +
  coord_equal() +
  geom_path_interactive(
    data = subset(
      fields,
      is_infield == FALSE),
    mapping = aes(
      x = xsh,
      y = ysh,
      group = id,
      tooltip = id,
      data_id = id
      ),
    color = '#000000',
    alpha = 0.5) +

  geom_polygon(
    data = subset(
      fields,
      is_infield == TRUE),
    mapping = aes(
      x = xsh,
      y = ysh,
      group = id),
    fill = '#FAD9B4',
    color = '#FAD9B4')

girafe(
  code = print(gg_boundaries),
  options = list(
    opts_hover(
      css = 'stroke-width:3;'),
    opts_hover_inv(
      css = 'stroke-opacity:0.1;')
    )
  )
```

30 baseball outfields — an *interactive* version





tools for interactive content, **several R packages transform ggplot2 into interactive graphics**

ggplot2 + ggiraph

The **grammar of graphics** — implemented in R as ggplot2 — is among the most powerful coding libraries for creating static graphics. We've already seen how to use a complementary package with ggplot2 to add animation: gganimate, a grammar of animated graphics. With similar complementary packages, we can specify **interactivity**. Let's see a static version of a class example, the 30 baseball outfields, then make it interactive using ggiraph.

```
gg_boundaries <-
  ggplot() +
  theme_void() +
  coord_equal() +
  geom_path_interactive(
    data = subset(
      fields,
      is_infield == FALSE),
    mapping = aes(
      x = xsh,
      y = ysh,
      group = id,
      tooltip = id,
      data_id = id
    ),
    color = '#000000',
    alpha = 0.5) +

  geom_polygon(
    data = subset(
      fields,
      is_infield == TRUE),
    mapping = aes(
      x = xsh,
      y = ysh,
      group = id),
    fill = '#FAD9B4',
    color = '#FAD9B4')

girafe(
  code = print(gg_boundaries),
  options = list(
    opts_hover(
      css = 'stroke-width:3;'),
    opts_hover_inv(
      css = 'stroke-opacity:0.1;')
    )
  )
```

NOTICE THE ONE-TO-ONE MATCHUP WITH GGPLOT2 GEOMETRIES

DETAILS-ON-DEMAND: ADD INFORMATION TO SHOW IN THE TOOLTIP, SHOWN WHEN HOVERING

USED TO BIND ELEMENTS TO EVENT LISTENERS

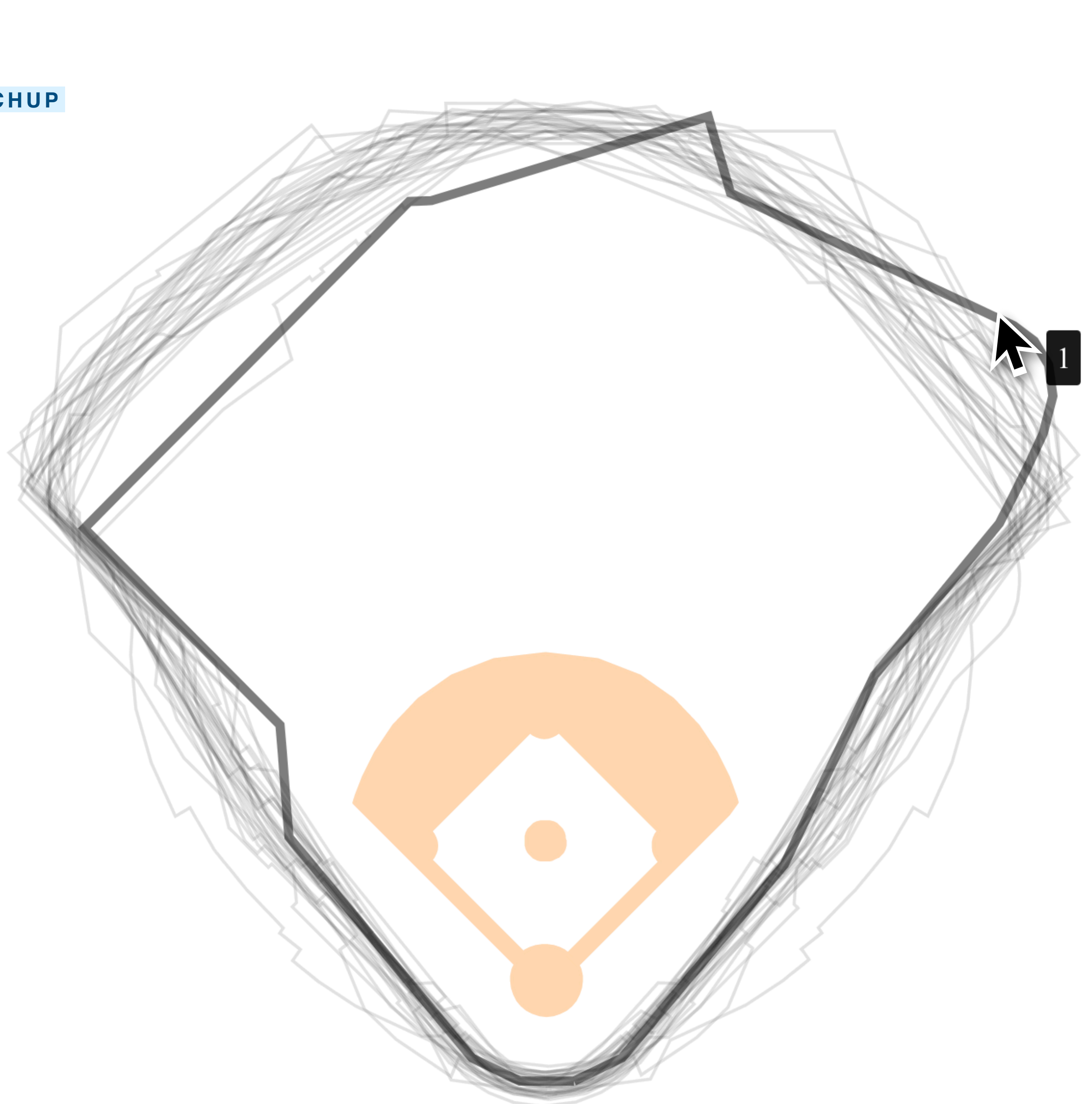
CHANGE PROPERTIES OF THINGS OF SHAPE REACTING TO HOVER

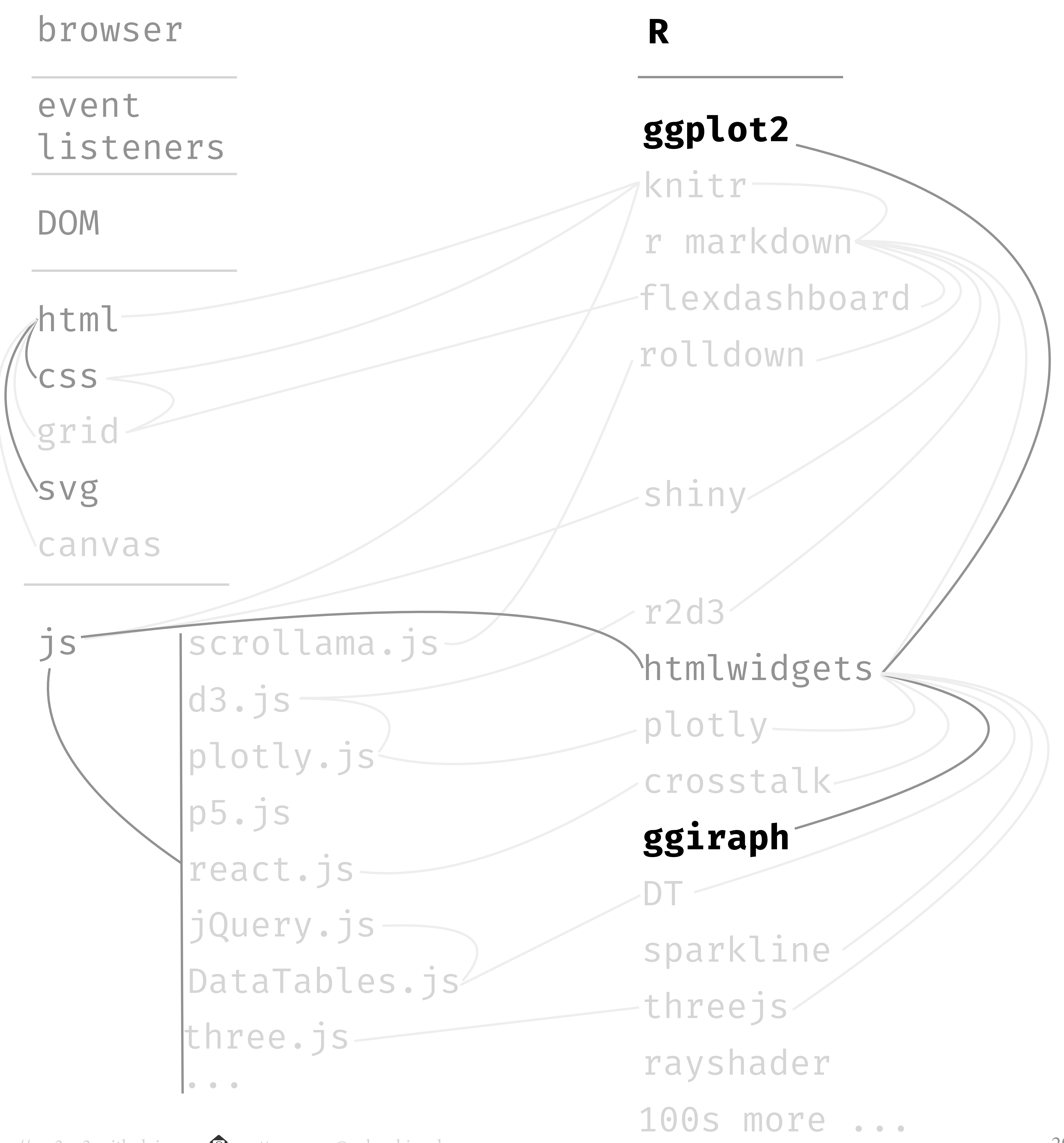
SET CSS PROPERTIES OF SHAPE; HERE STROKE-WIDTH

CHANGE PROPERTIES OF THINGS ELEMENT *NOT* REACTING TO HOVER

SET CSS PROPERTIES OF SHAPE; HERE STROKE-OPACITY

30 baseball outfields — an *interactive* version





 several R packages transform ggplot2 into interactive graphics

ggplot2 + ggiraph

The **grammar of graphics** — implemented in R as ggplot2 — is among the most powerful coding libraries for creating static graphics. We've already seen how to use a complementary package with ggplot2 to add animation: gganimate, a grammar of animated graphics. With similar complementary packages, we can specify **interactivity**. Let's see a static version of a class example, the 30 baseball outfields, then make it interactive using ggiraph.

```
gg_fences <-
    ggplot() +
    theme_void() +
    theme( axis.text.x = element_text() ) +
    coord_equal() +
    scale_x_continuous(
      breaks = c(100, 300, 500),
      labels = c("Left Field",
                 "Center Field",
                 "Right Field")) +
    geom_path_interactive(
      data = fences,
      mapping = aes(
        x = xsh,
        y = -ysh,
        group = id,
        tooltip = id,
        data_id = id),
      color = 'black',
      alpha = 0.5)

girafe(
  code = print(gg_boundaries / gg_fences),
  options = list(
    opts_hover(
      css = 'stroke-width:3;'),
    opts_hover_inv(
      css = 'stroke-opacity:0.1;')
    )
  )
```

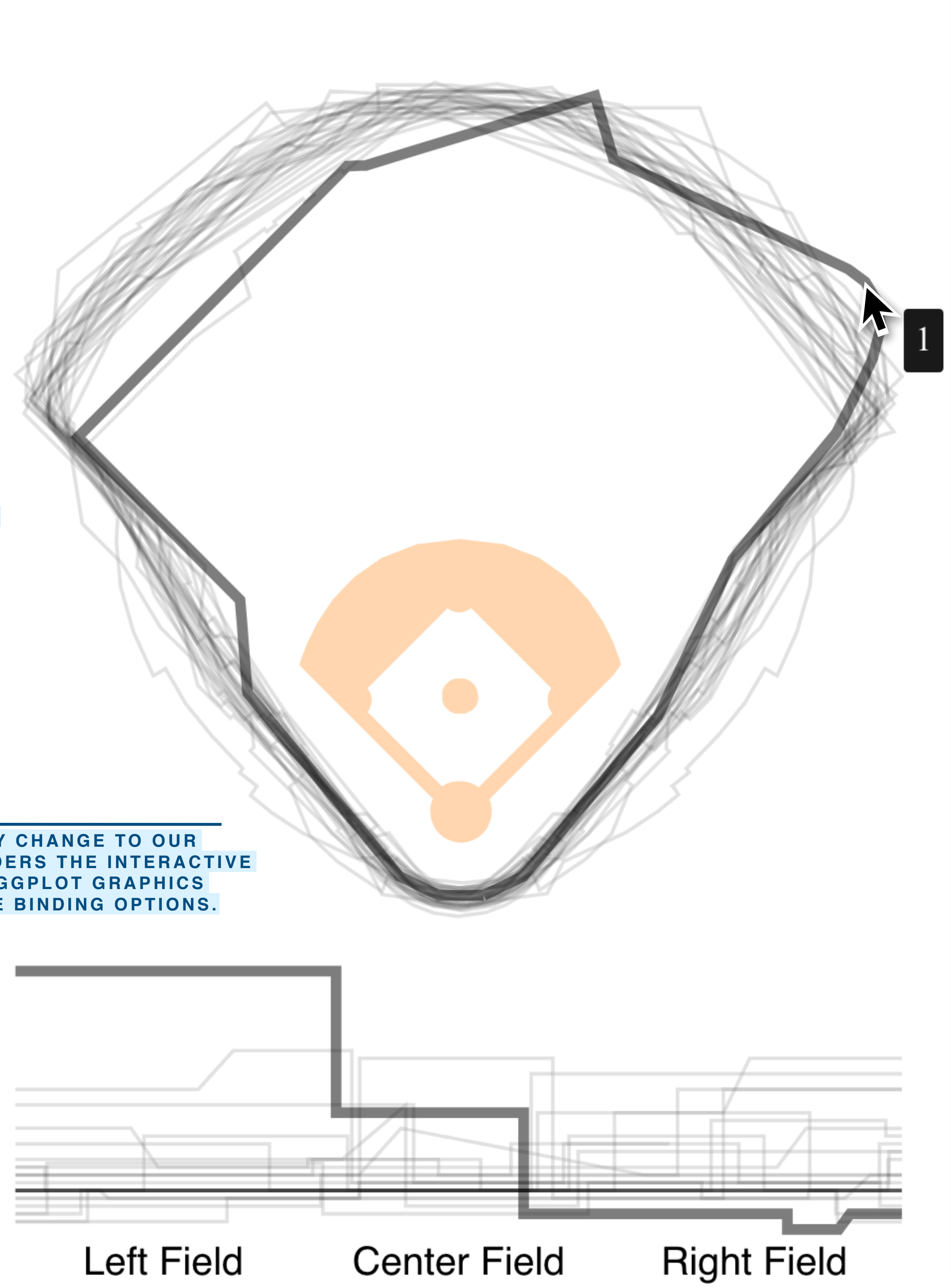
NOTICE THE SAME DATA ID USED HERE AS FOR THE OUTFIELD BOUNDARIES

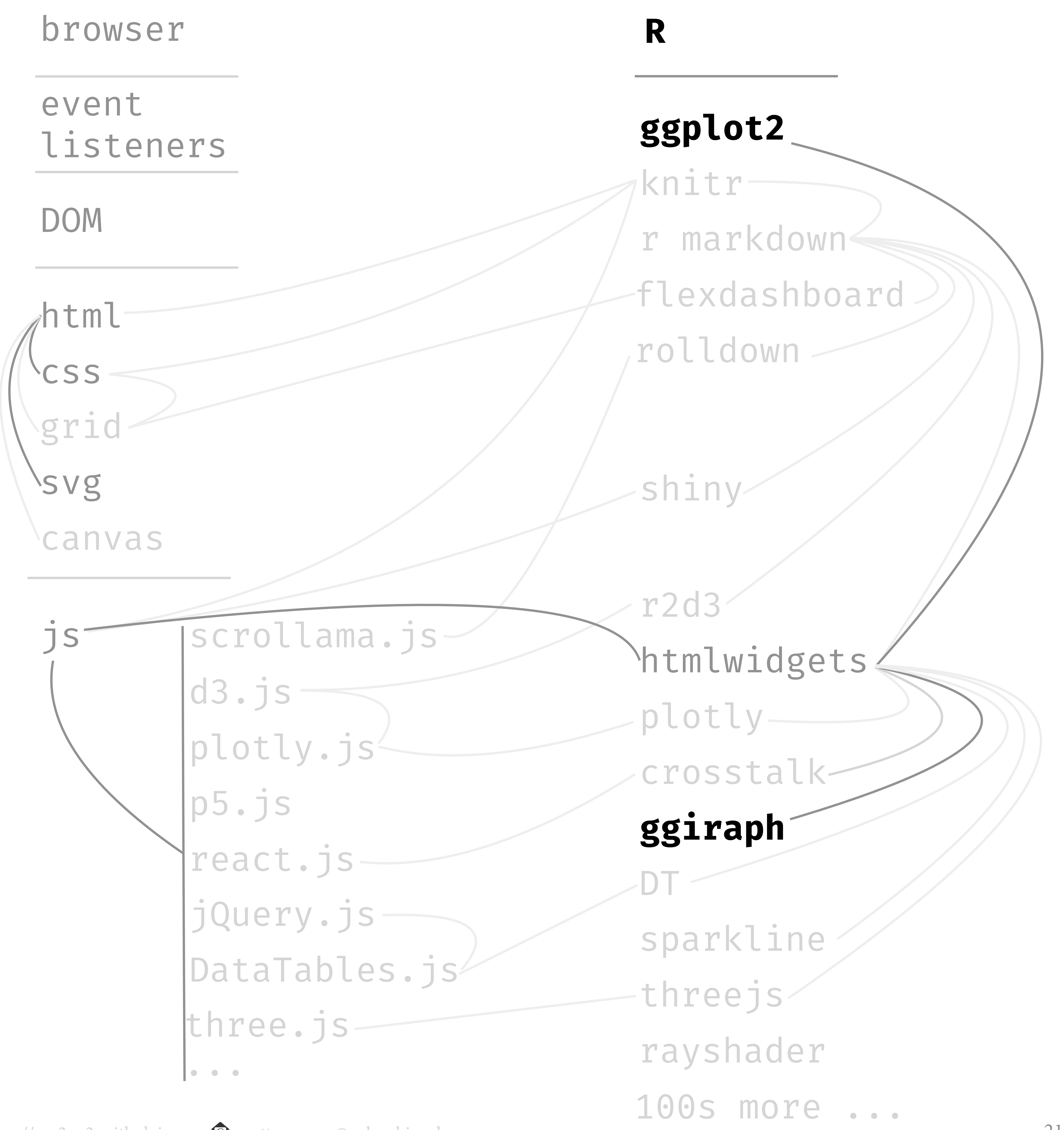
THIS IS THE ONLY CHANGE TO OUR CODE THAT RENDERS THE INTERACTIVE GRAPHIC. BOTH GGPLOT GRAPHICS SHARE THE SAME BINDING OPTIONS.

IN RSTUDIO, WE CAN SAVE THE INTERACTIVE GRAPHIC AS A STAND-ALONE WEB PAGE.

OR WE CAN INCLUDE THE CODE IN AN R MARKDOWN CHUNK OR FLEXDASHBOARD AND KNIT TO HTML.

30 baseball outfields — an *interactive* version

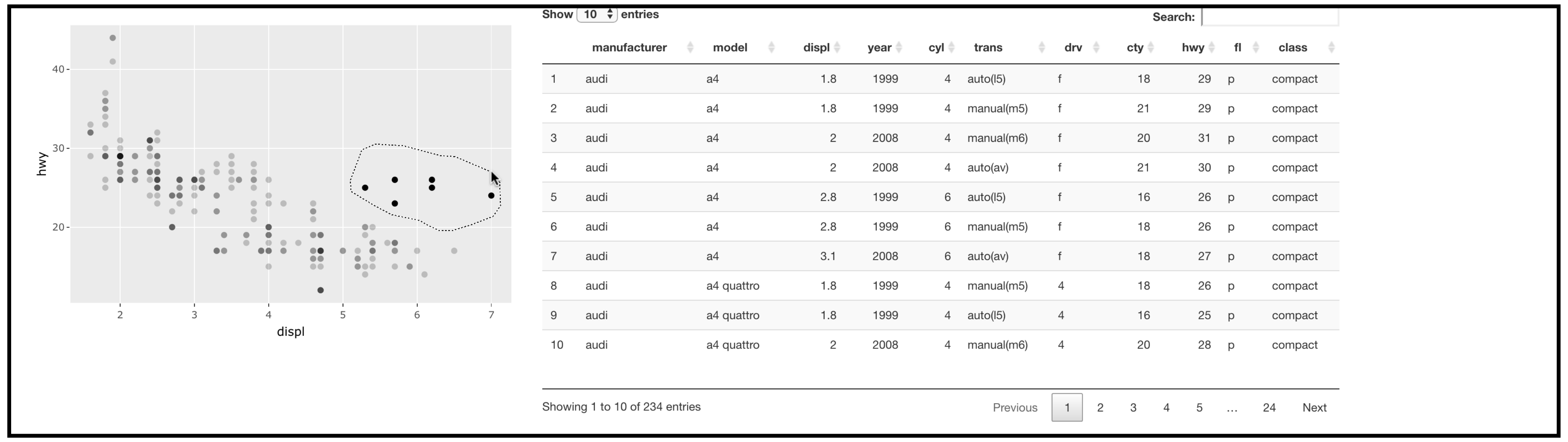




tools for interactive content, `plotly` **is a charting library that can bind with other** `htmlwidgets`

`ggplot2 + plotly + crosstalk + DT`

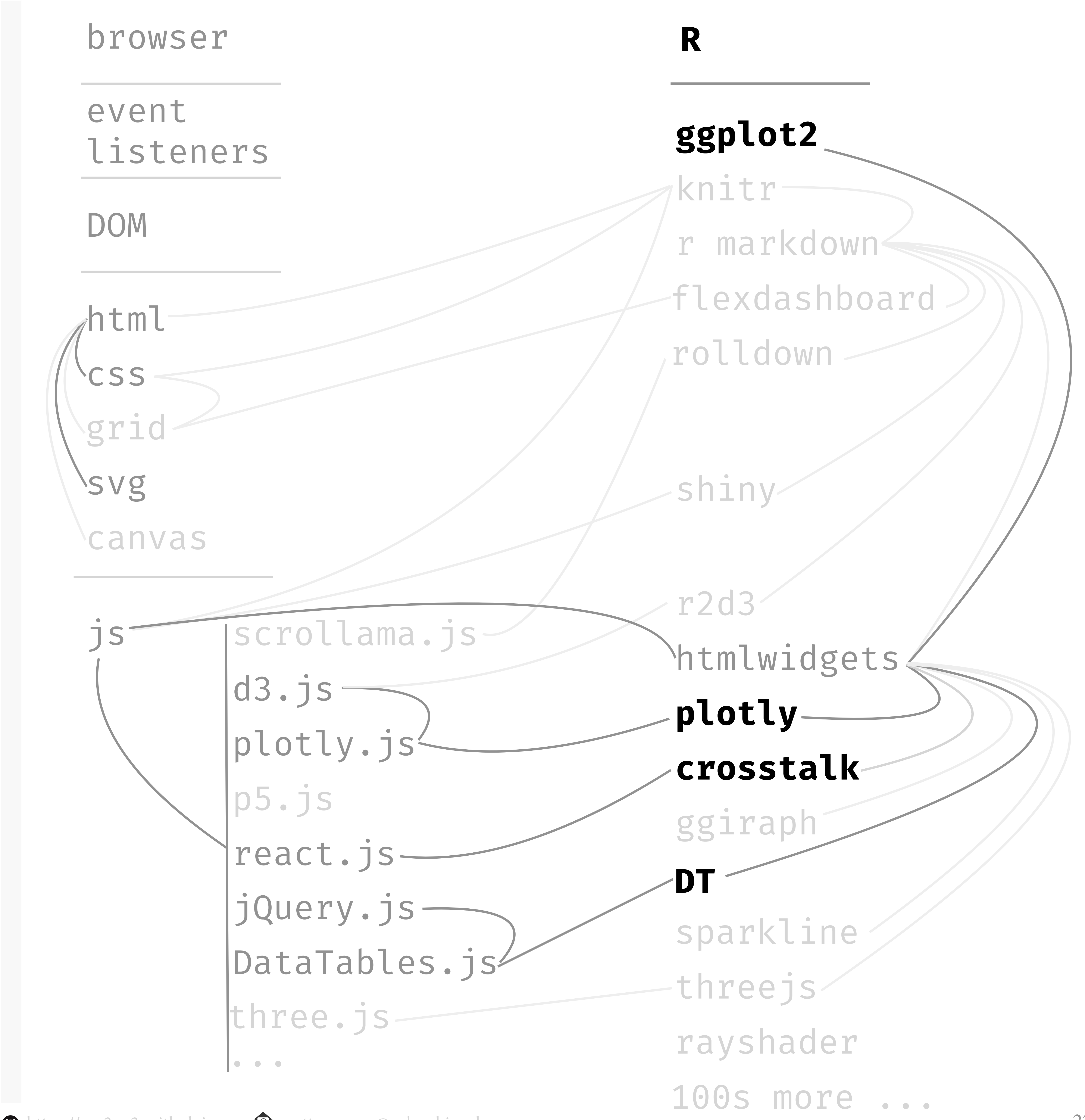
`plotly` is an R package for creating interactive graphics, and interfaces with the same-named javascript library, `plotly.js`, which in turn is based on `d3.js`. R's `plotly` has several helpful features. The first of these are it, like, `ggiraph`, allows **easy integration** with `ggplot2`. The first function, perhaps, to learn, is `ggplotly` which takes as a parameter a `ggplot` object and makes it interactive. And combined with another package, `crosstalk`, it a `plotly` graphic **can link or bind with other `htmlwidgets`**. Here's an example:



Show 10 entries Search:

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
1	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
2	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
3	audi	a4	2	2008	4	manual(m6)	f	20	31	p	compact
4	audi	a4	2	2008	4	auto(av)	f	21	30	p	compact
5	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact
6	audi	a4	2.8	1999	6	manual(m5)	f	18	26	p	compact
7	audi	a4	3.1	2008	6	auto(av)	f	18	27	p	compact
8	audi	a4 quattro	1.8	1999	4	manual(m5)	4	18	26	p	compact
9	audi	a4 quattro	1.8	1999	4	auto(l5)	4	16	25	p	compact
10	audi	a4 quattro	2	2008	4	manual(m6)	4	20	28	p	compact

Showing 1 to 10 of 234 entries Previous 1 2 3 4 5 … 24 Next



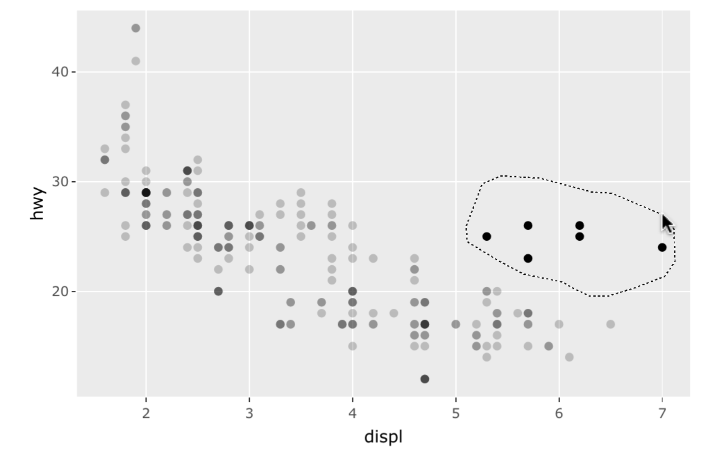
tools for interactive content, `plotly` is a charting library that can bind with other `htmlwidgets`

ggplot2 + plotly + crosstalk + DT

`plotly` is an R package for creating interactive graphics, and interfaces with the same-named javascript library, `plotly.js`, which in turn is based on `d3.js`. R's `plotly` has several helpful features. The first of these are it, like, ggiraph, allows **easy integration** with `ggplot2`. The first function, perhaps, to learn, is `ggplotly` which takes as a parameter a `ggplot` object and makes it interactive. And combined with another package, `crosstalk`, it a `plotly` graphic **can link or bind with other `htmlwidgets`**. Here's an example:

```r
library(ggplot2)
library(plotly)      # FUNCTIONS FOR INTERACTIVE CHARTS
library(crosstalk)   # FUNCTIONS ENABLING HTMLWIDGETS TO SHARE INTERACTIVITY
library(DT)          # FUNCTIONS FOR INTERACTIVE TABLES

m <- highlight_key(mpg)
```

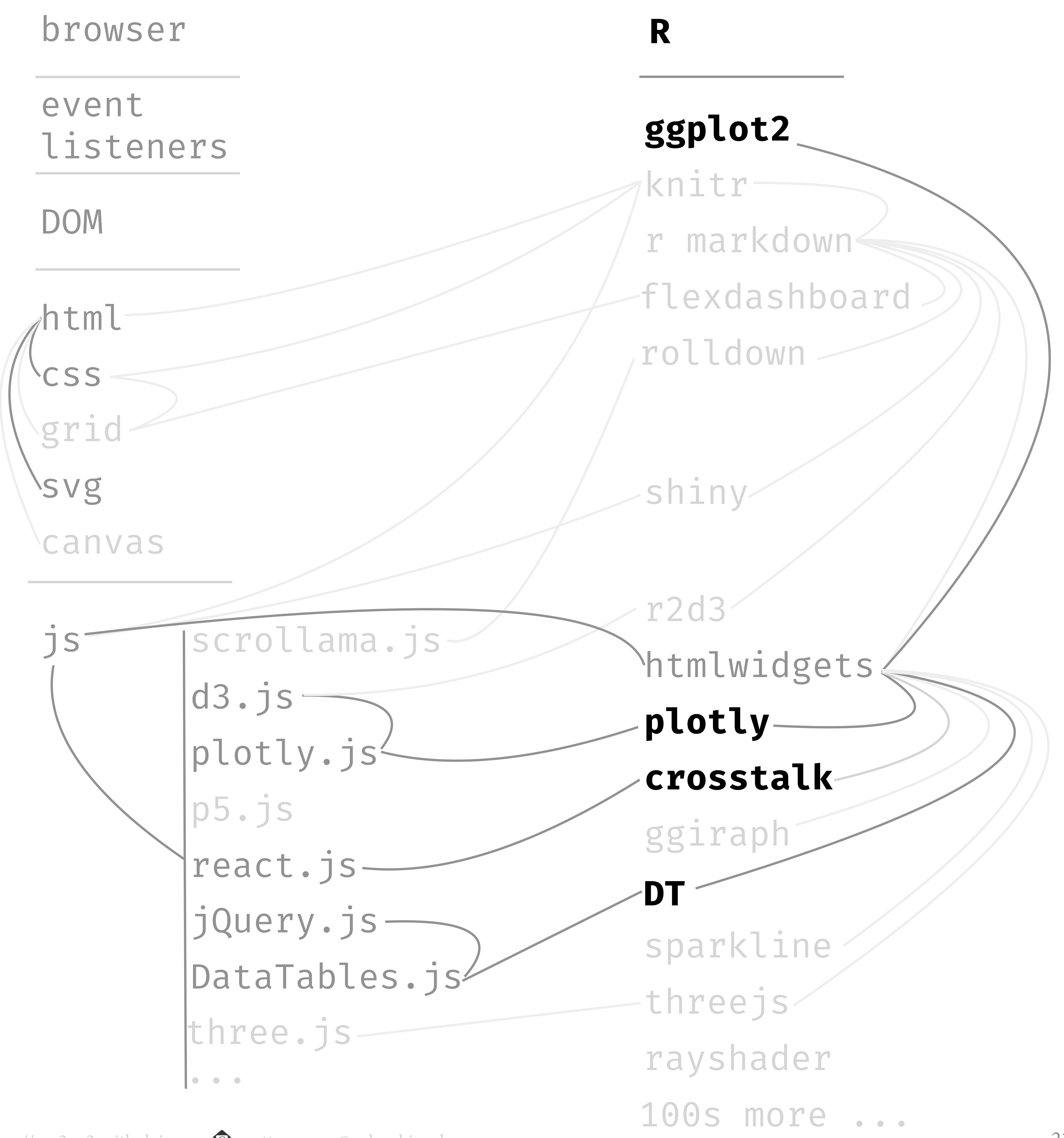
```r
p <- ggplot(          # HERE'S OUR GGPLOT OBJECT
  data = m,
  mapping = aes(
   x = displ,
   y = hwy)) +
  geom_point()

gg <- highlight(      # NOW WE MAKE THE GGPLOT OBJECT INTERACTIVE
  p = ggplotly(p),
  on = "plotly_selected")

bscols(gg, datatable(m))   # FINALLY WE LINK BOTH INTERACTIVE WIDGETS.
```



Show 10 entries Search:

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
1	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
2	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
3	audi	a4	2	2008	4	manual(m6)	f	20	31	p	compact
4	audi	a4	2	2008	4	auto(av)	f	21	30	p	compact
5	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact
6	audi	a4	2.8	1999	6	manual(m5)	f	18	26	p	compact
7	audi	a4	3.1	2008	6	auto(av)	f	18	27	p	compact
8	audi	a4 quattro	1.8	1999	4	manual(m5)	4	18	26	p	compact
9	audi	a4 quattro	1.8	1999	4	auto(l5)	4	16	25	p	compact
10	audi	a4 quattro	2	2008	4	manual(m6)	4	20	28	p	compact

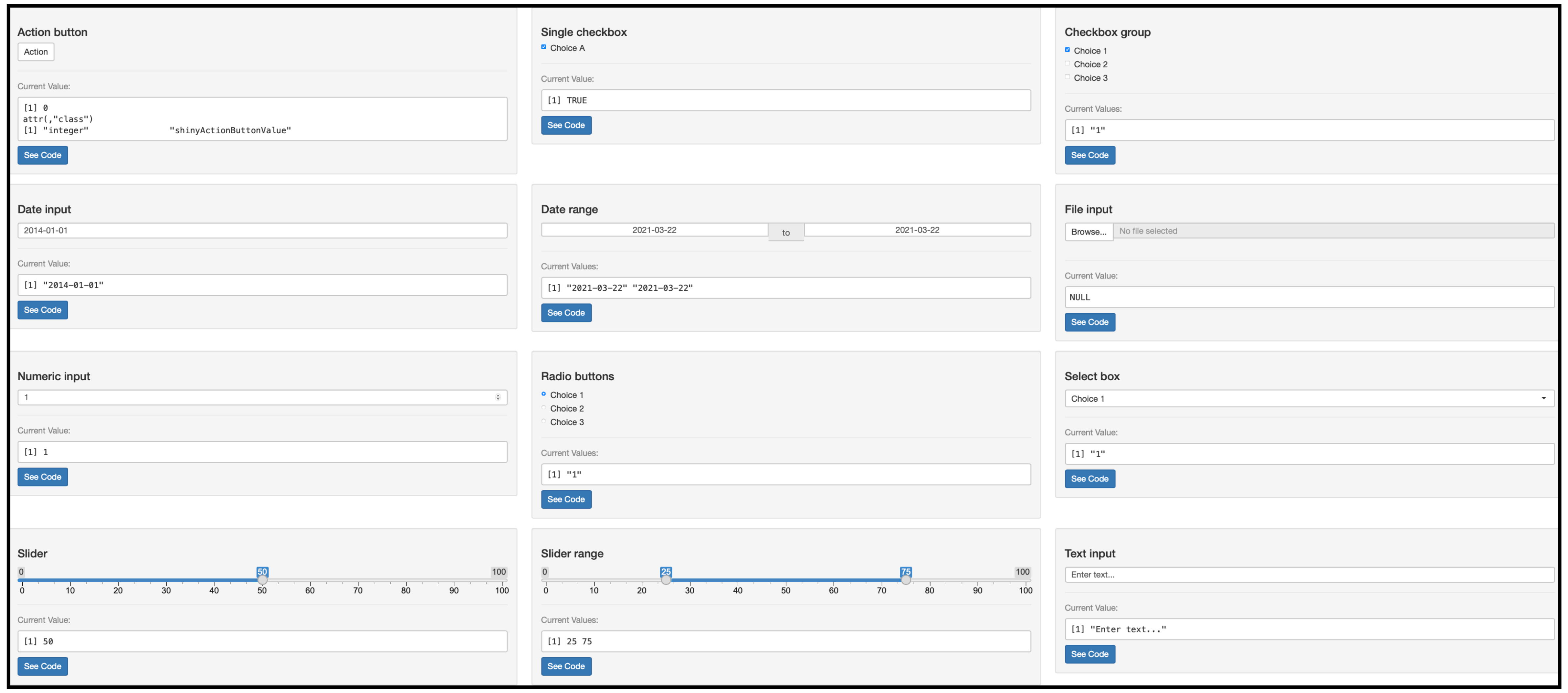
Showing 1 to 10 of 234 entries Previous 1 2 3 4 5 … 24 Next

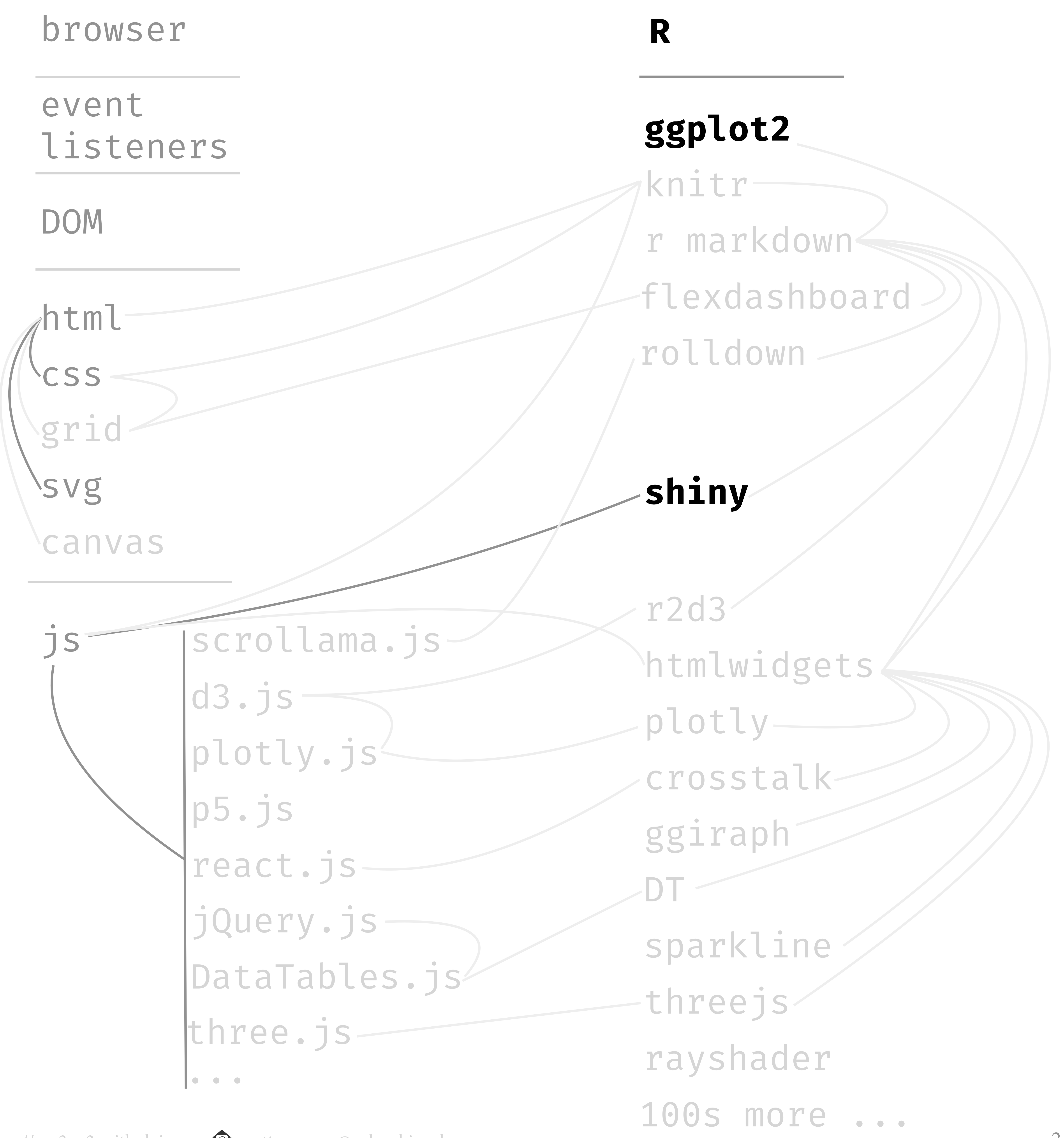


 web application tools are more complex but allow more sophisticated interactions

ggplot2 + shiny + ...

Shiny is for developing **web applications**. This means it runs on a **web server** to enable user interface widgets on a **webpage**. Further, it **requires linking to an active R session**. Thus, unlike the previous software, we cannot share a single, standalone html file. The closest we get is to share an r markdown file with shiny code that someone can open in RStudio and click "run" to start a server. Below are examples of various widgets we can use to create these interactive, web applications.

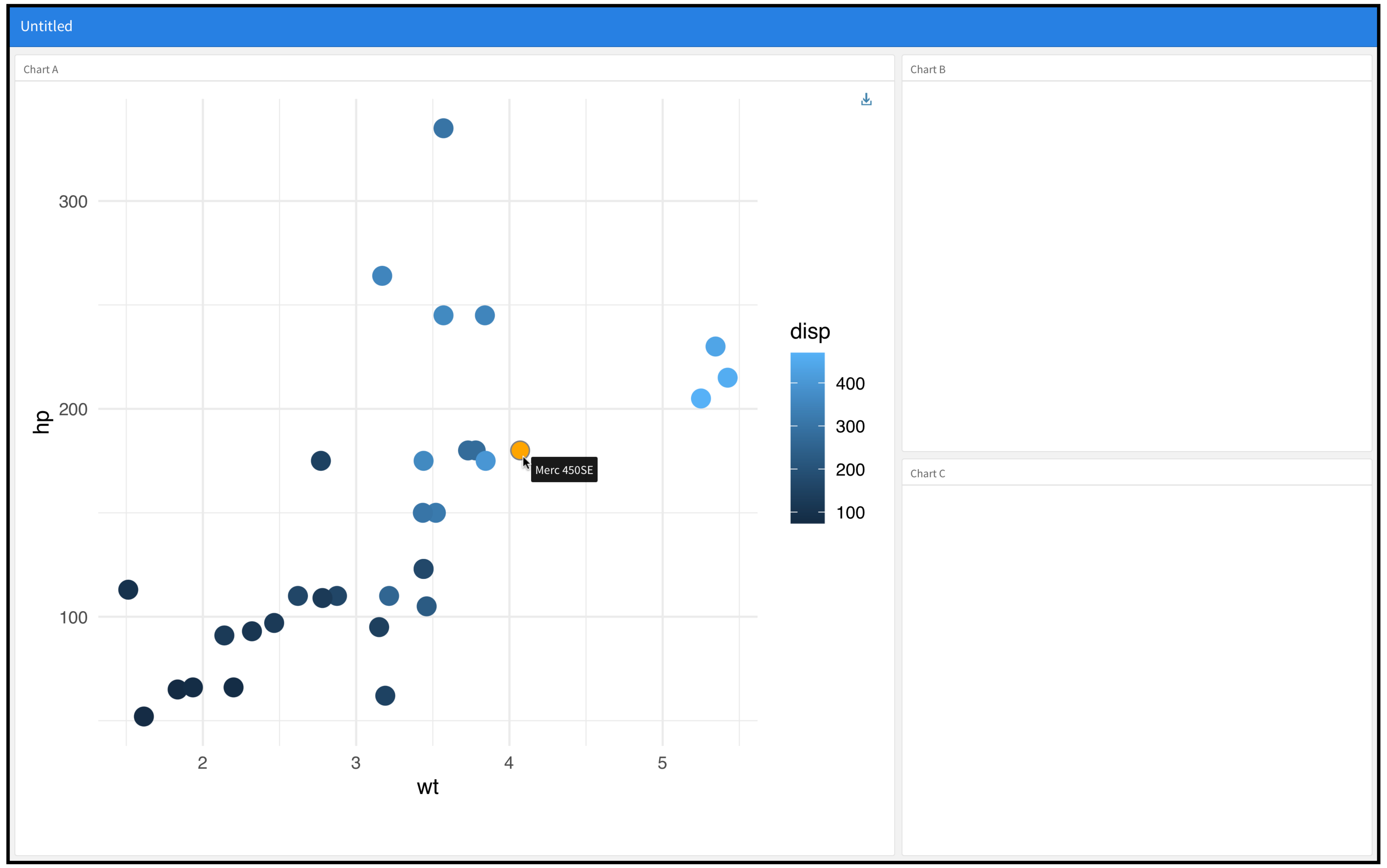


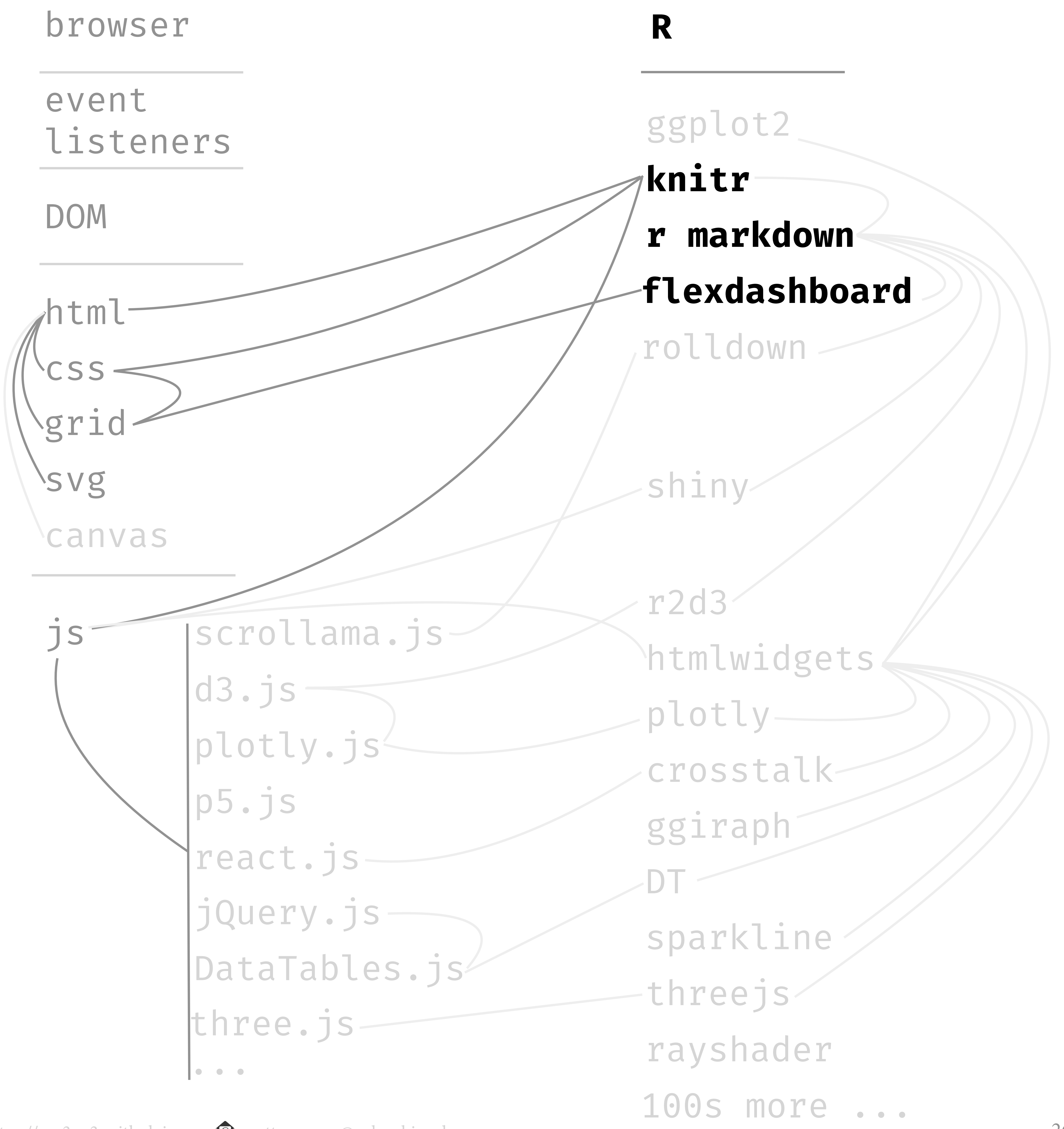


organizing interactive graphics
with web technologies — (for dashboards)

`knitr + rmarkdown + flexdashboard`

We can **organize** various widgets and enable their communication through web technologies, all placed inside an `html` file. Perhaps my favorite way to bring these technologies together is using **`r markdown` templates** like **flexdashboard** that **`knitr`** and **`RStudio`** uses to weave together text, image, code and results. Along with markdown templates, we can roll our own with **`css grid`**, adding code chunks between `<div class="">` and `</div>` where we define our own `css` classes. Here's a screenshot of an example below:

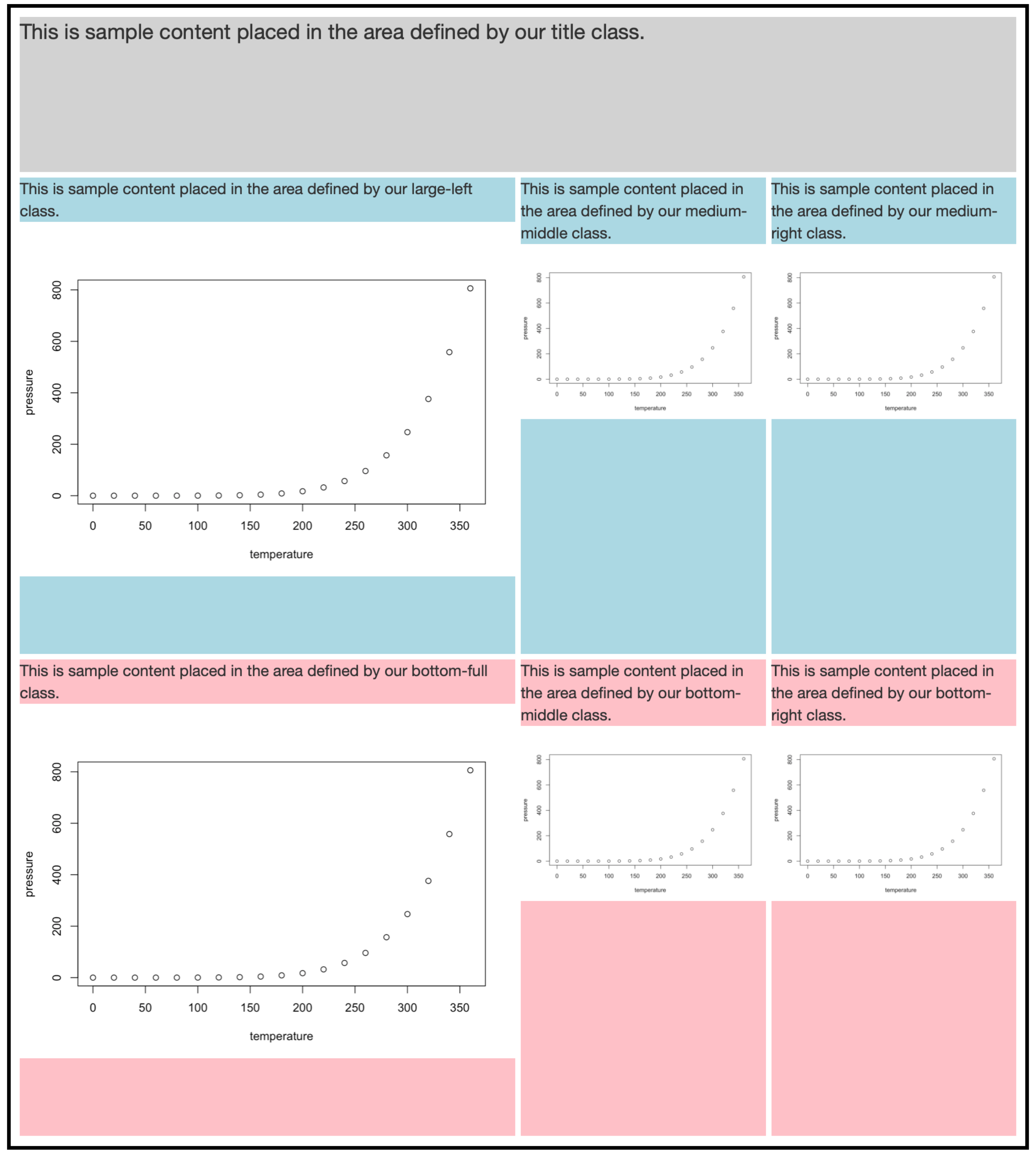


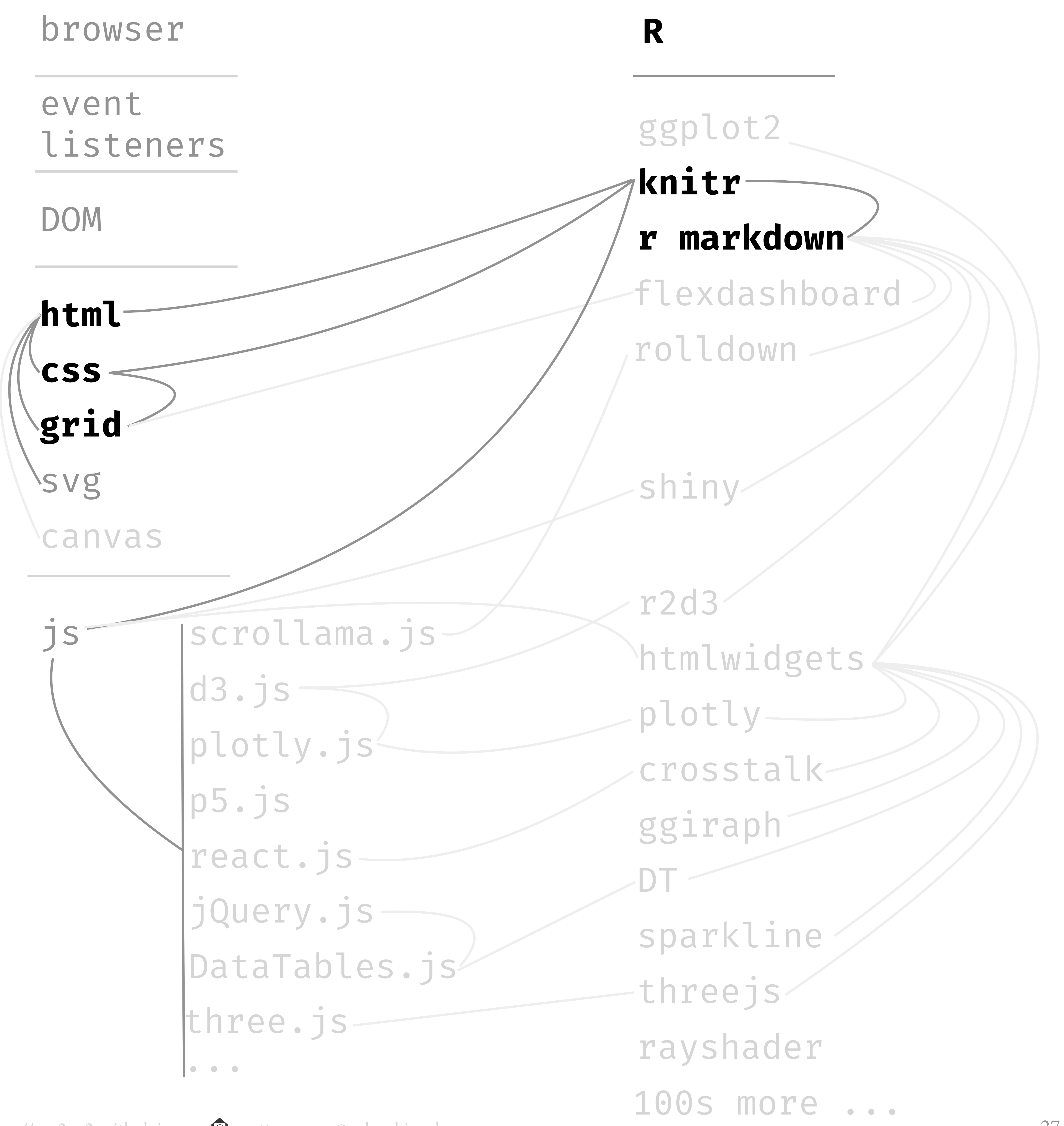


knitr + rmarkdown + css grid + html

We can **organize** various widgets and enable their communication through web technologies, all placed inside an `html` file. Perhaps my favorite way to bring these technologies together is using **`r markdown` templates** like **flexdashboard** that **`knitr`** and **`RStudio`** uses to weave together text, image, code and results. Along with markdown templates, we can roll our own with **`css grid`**, adding code chunks between `<div class="">` and `</div>` where we define our own `css` classes. Here's a screenshot of an example below:

```
<style>

.main-container {
  min-width: 950px;
  max-width: 950px;

}

.gridlayout {
  display: grid;
  position: relative;
  margin: 10px;
  gap: 5px;
  grid-template-columns:
    repeat(8, 1fr);
  grid-template-rows:
    repeat(8, 140px);
}

.title {
  background: lightgray;
  grid-column: 1 / 9;
  grid-row: 1 / 2;
  font-size: 14pt;
}

.large-left {
  background: lightblue;
  grid-column: 1 / 5;
  grid-row: 2 / 5;
}

.medium-middle {
  background: lightblue;
  grid-column: 5 / 7;
  grid-row: 2 / 5;
```





resources

References

Spencer, Scott. "Interaction: technologies and tools of interactive data-driven, visual design," Sec. 3.2 In *Data in Wonderland*. 2021. https://ssp3nc3r.github.io/data_in_wonderland.

Attardi, Joe. *Modern CSS: Master the Key Concepts of CSS for Modern Web Development*, 2020. https://doi.org/10.1007/978-1-4842-6294-8.

Bellamy-Royds, Amelia, Kurt Cagle, and Dudley Storey. *Using SVG with CSS3 and HTML5 : Vector Graphics for Web Design*. O'Reilly, 2018.

Duckett, Jon. *HTML & CSS. Design and Build Websites*. Wiley, 2011.

Duckett, Jon, Gilles Ruppert, and Jack Moore. *JavaScript & JQuery: Interactive Front-End Web Development*. Indianapolis, IN: Wiley, 2014.

Fay, Colin, Vincent Guyader, Sebastien Rochette, and Girard Cervan. *Engineering Production-Grade Shiny Apps*. First edition. R Series. Boca Raton: CRC Press, 2021. https://engineering-shiny.org.

Gohel, David, and Panagiotis Skintzos. "*Ggiraph: Make 'ggplot2' Graphics Interactive*." Manual, 2021. https://davidgohel.github.io/ggiraph.

Janert, Philipp K. *D3 for the Impatient: Interactive Graphics for Programmers and Scientists*. First edition. Sebastopol, CA: O'Reilly Media, Inc, 2019.

Meeks, Elijah. *D3.Js in Action*. Second. Manning, 2018.

Murray, Scott. *Interactive Data Visualization for the Web*. Second. *An Introduction to Designing with D3*. O'Reilly, 2017.

Reas, Casey, and Ben Fry. *Processing A Programming Handbook for Visual Designers and Artists*. Second. The MIT Press, 2014.

Sievert, Carson. *Interactive Web-Based Data Visualization with R, Plotly, and Shiny*. Boca Raton, FL: CRC Press, Taylor and Francis Group, 2020. https://plotly-r.com.

Vaidyanathan, Ramnath, Yihui Xie, JJ Allaire, Joe Cheng, Carson Sievert, and Kenton Russell. "*Htmlwidgets: HTML Widgets for r.*" Manual, 2020. https://CRAN.R-project.org/package=htmlwidgets; main introduction: http://www.htmlwidgets.org.

Wickham, Hadley. "*Create Elegant Data Visualisations Using the Grammar of Graphics • Ggplot2*." Accessed February 26, 2021. https://ggplot2.tidyverse.org/.