

# The other side of Go

## Programming Pictures

Anthony Starks



Go is great in the back end



But sometimes it's about the picture

API Design

Client Program Design

Visual Design and Relationships

Why Go?



Element

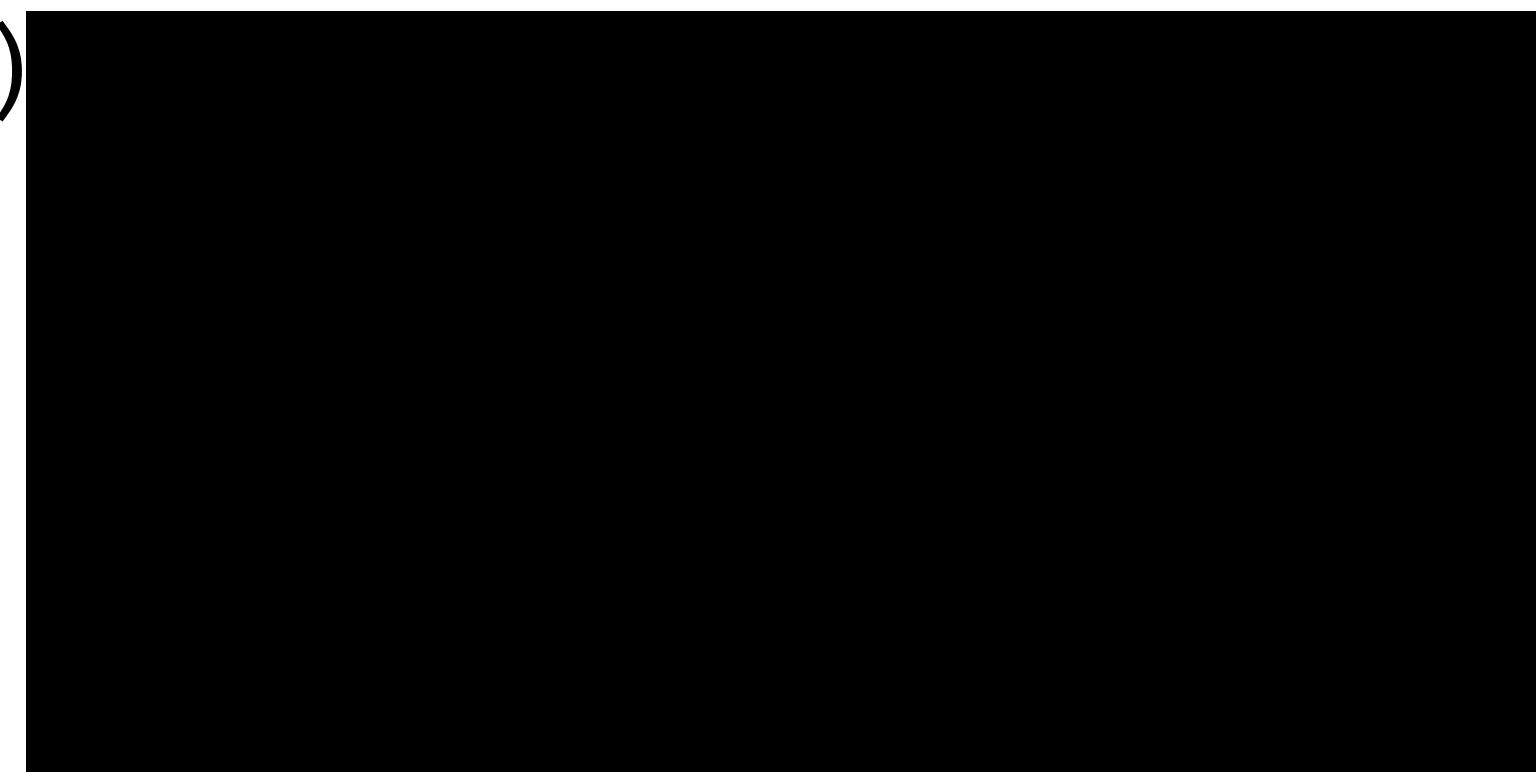
Rect

Arguments

(100,200,250,125)

```
<rect x="100" y="200" width="250" height="125"/>
```

(100, 200)



125

250

Element

Rect

Arguments

(100,200,250,125,

CSS Style

"fill:gray;stroke:blue")

```
<rect x="100" y="200" width="250" height="125"  
style="fill:gray;stroke:blue"/>
```

(100, 200)



125

250

Element

Rect

Arguments

(100,200,250,125,  
`id="box"`, `fill="gray"`, `stroke="blue"`)

Attributes

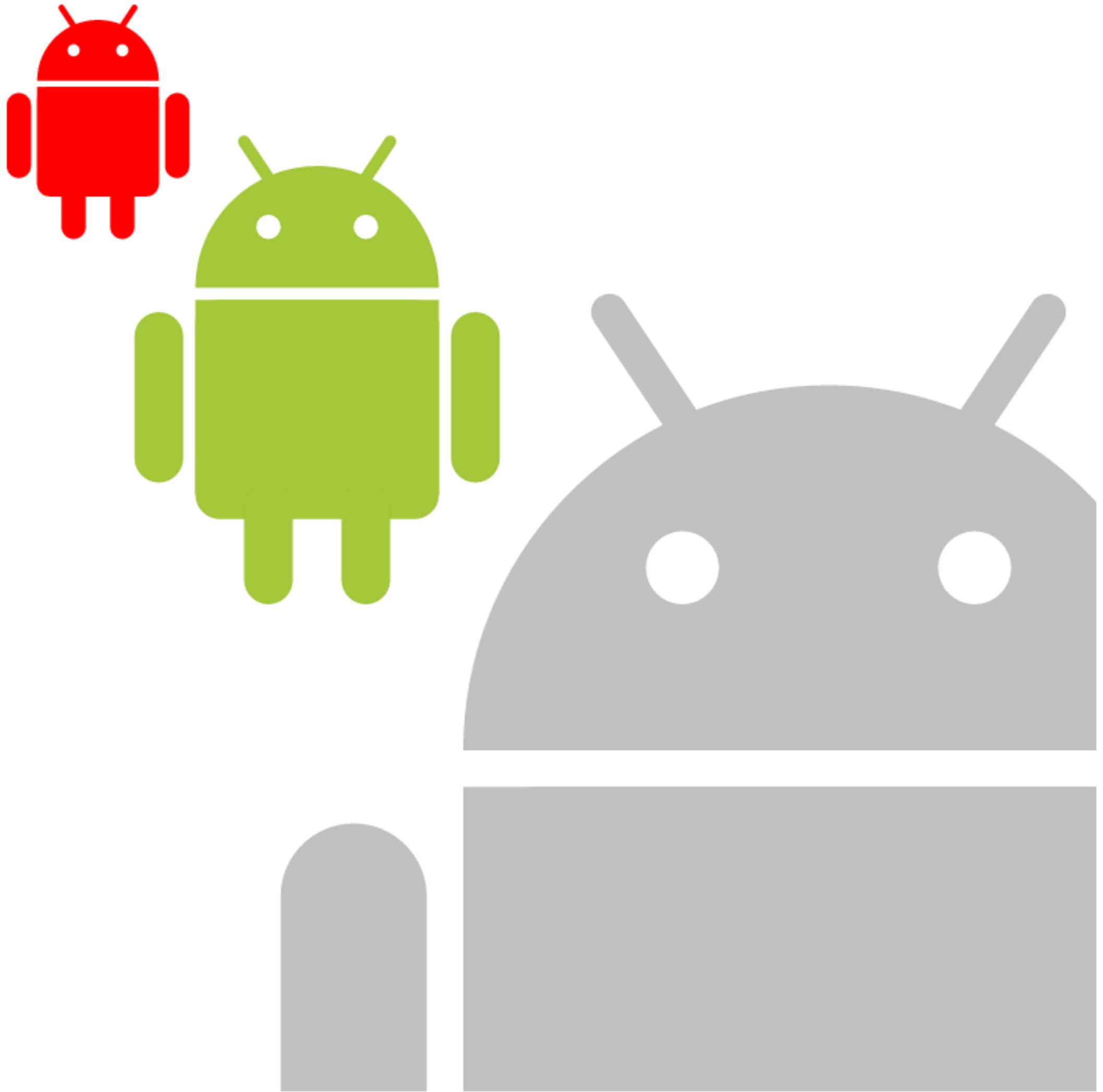
```
<rect x="100" y="200" width="250" height="125"  
id="box" fill="gray" stroke="blue"/>
```

(100, 200)



125

250



Scale



Roundrect



fill:rgb(164,198,57)

Line

Circle

Arc

Line

Rect

```
package main

import (
    "os"
    "github.com/ajstarks/svg"
)

func main() {
    width := 960
    height := 540
    canvas := svg.New(os.Stdout)
    canvas.Start(width, height)
    canvas.Rect(0, 0, width, height, "fill:black")
    canvas.Circle(width/2, height, width/2, "fill:rgb(44,77,232)")
    canvas.Text(width/2, height/2, "hello, world",
        "fill:white;font-size:60pt;font-family:serif;text-anchor:middle")
    canvas.End()
}
```



```

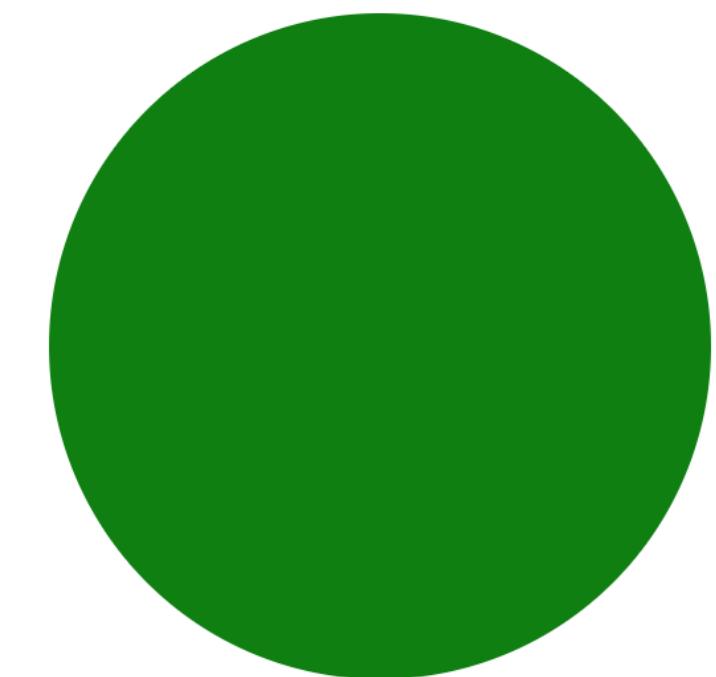
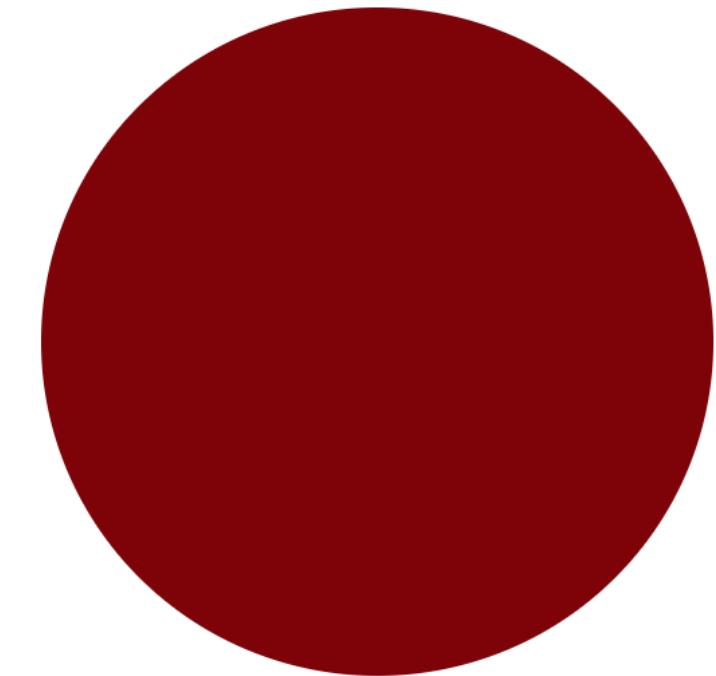
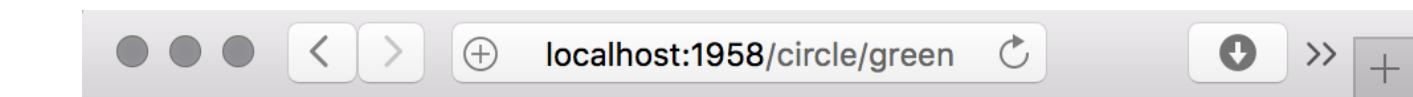
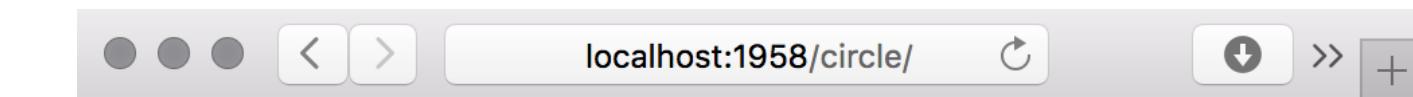
const defaultstyle = "fill:rgb(127,0,0)"

func main() {
    http.Handle("/circle/", http.HandlerFunc(circle))
    err := http.ListenAndServe("localhost:1958", nil)
    if err != nil {
        log.Println("ListenAndServe:", err)
    }
}

func circle(w http.ResponseWriter, req *http.Request) {
    w.Header().Set("Content-Type", "image/svg+xml")
    s := svg.New(w)
    s.Start(500, 500)
    s.Title("Circle")
    s.Circle(250, 250, 125, shapestyle(req.URL.Path))
    s.End()
}

func shapestyle(path string) string {
    i := strings.LastIndex(path, "/") + 1
    if i > 0 && len(path[i:]) > 0 {
        return "fill:" + path[i:]
    }
    return defaultstyle
}

```



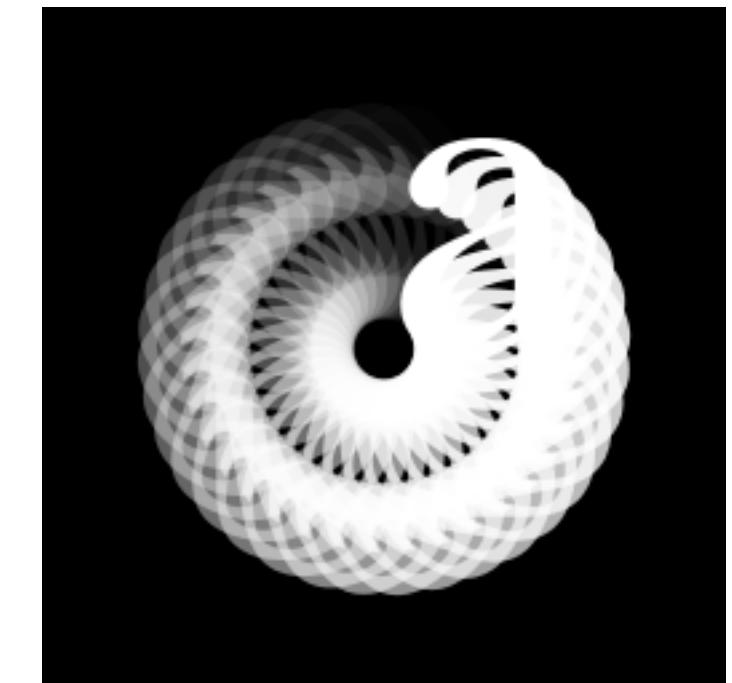
<http://ajstarks.org:1958/{thing}/>



clock



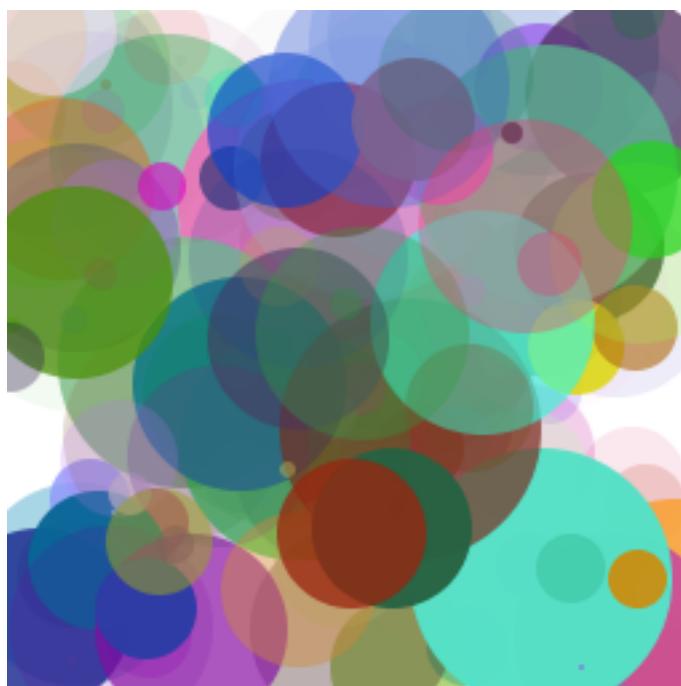
funnel



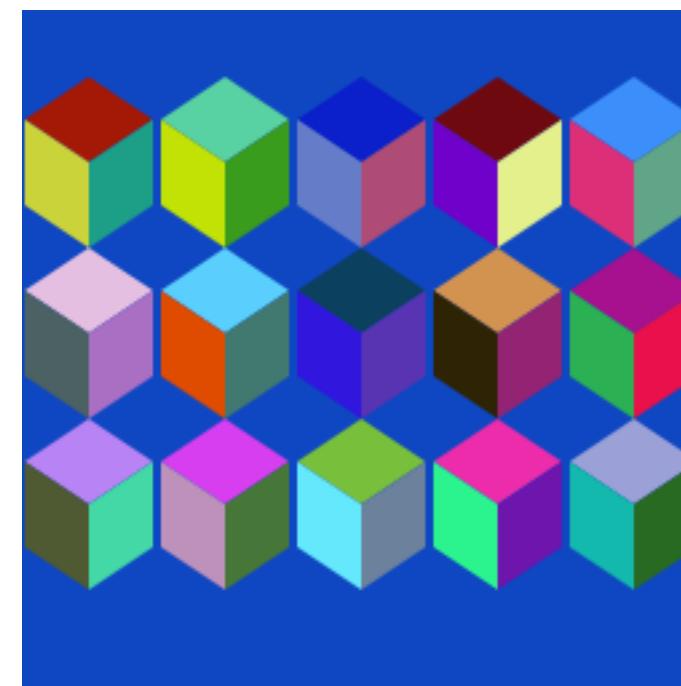
rotext



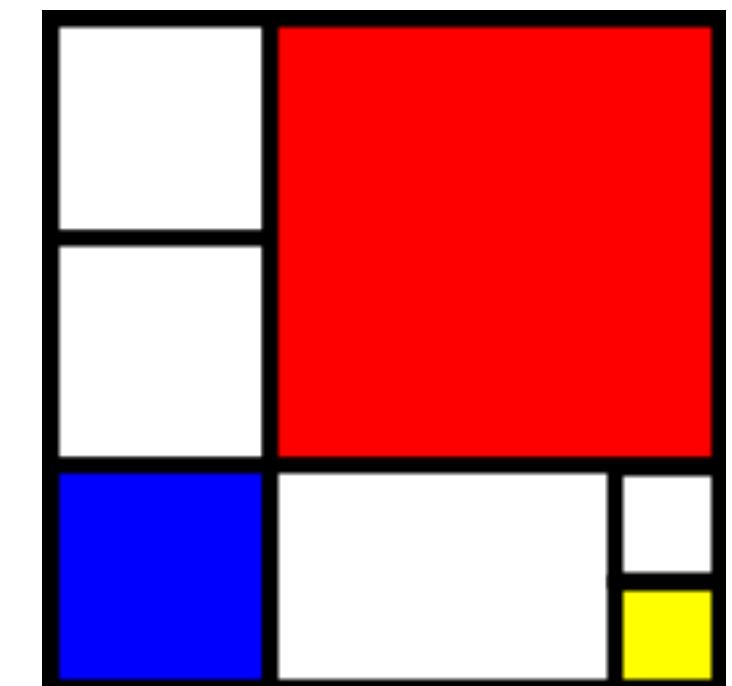
flower



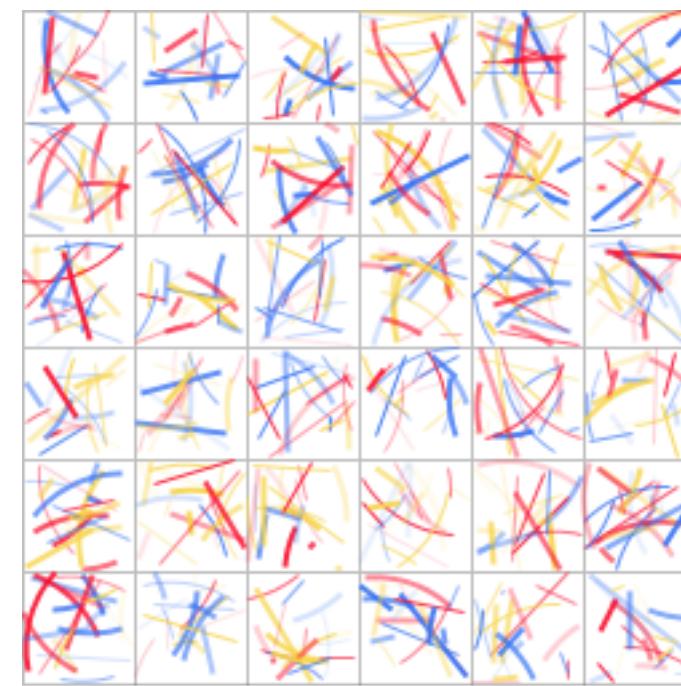
rshape



cube



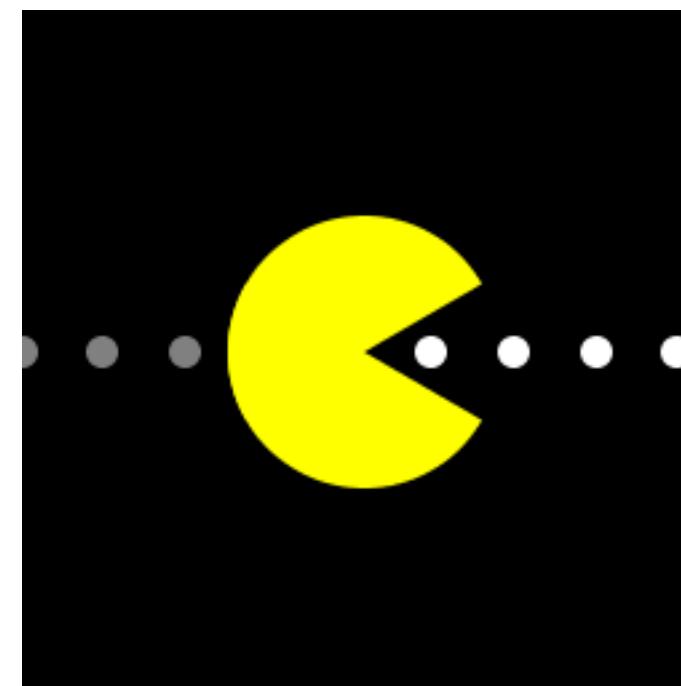
mondrian



lewitt



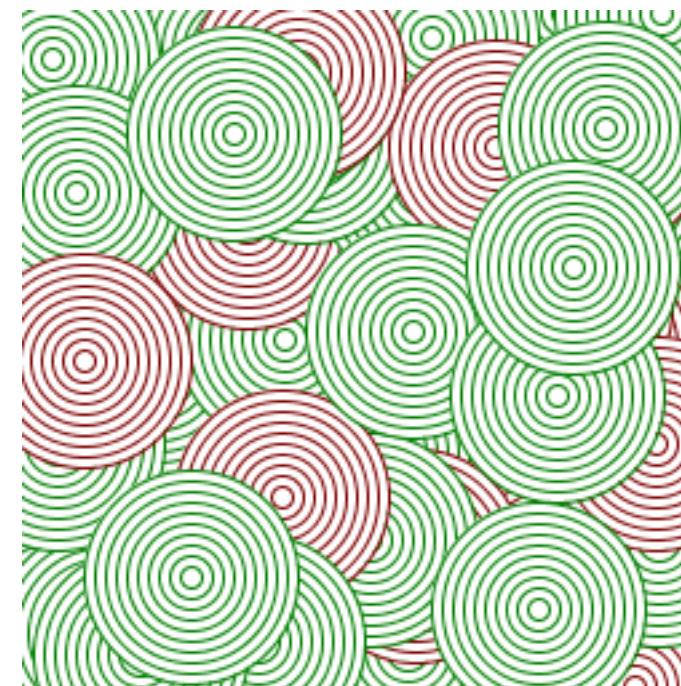
face



pacman

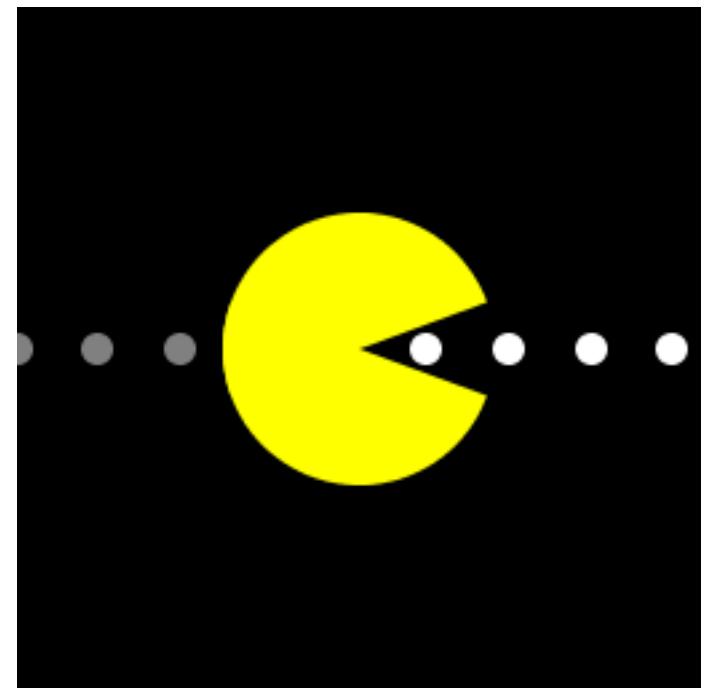


tux

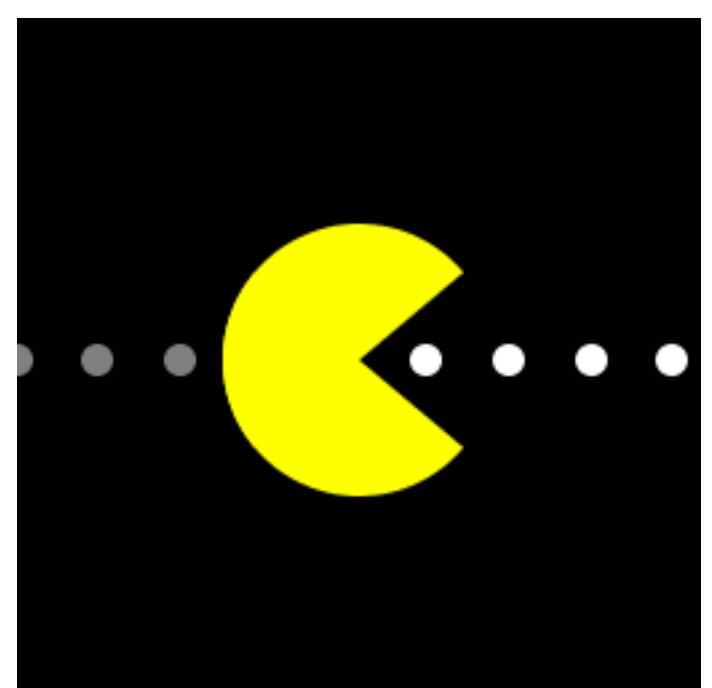


concentric

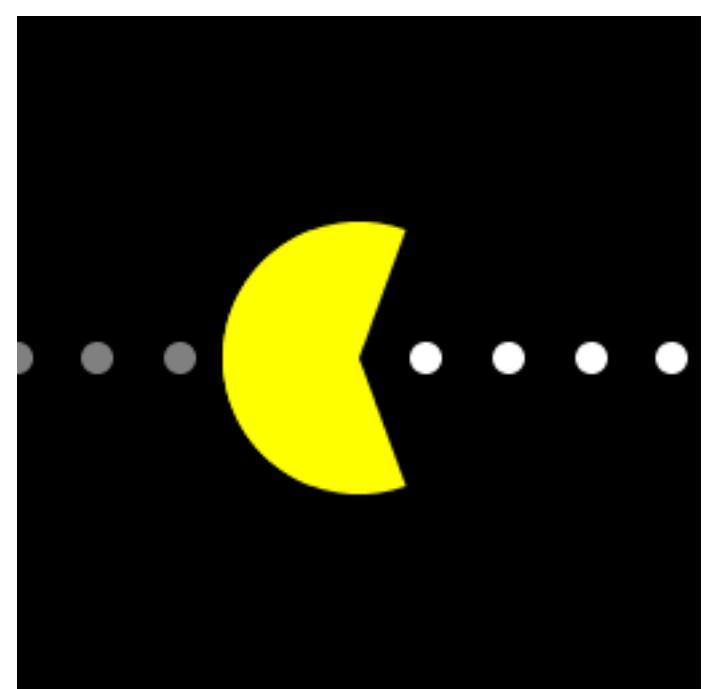
/pacman/?angle=20



/pacman/?angle=40



/pacman/?angle=70

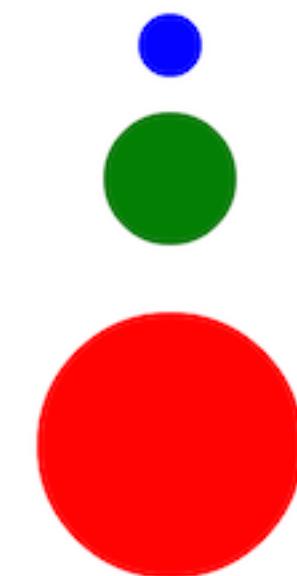


# Read/Parse/Draw Pattern

# Data



# Picture



Little:This is small/blue

Med:This is medium/green

Big:This is large/red

```
<thing top="100" left="100" sep="100">
  <item width="50" height="50" name="Little" color="blue">This is small</item>
  <item width="75" height="100" name="Med"   color="green">This is medium</item>
  <item width="100" height="200" name="Big"   color="red">This is large</item>
</thing>
```

## Imports

```
package main
import (
    "encoding/xml"
    "flag"
    "fmt"
    "io"
    "os"
    "github.com/ajstarks/svggo"
)
```

## Read the Input

```
func dothing(location string) {
    f, err := os.Open(location)
    if err != nil {
        fmt.Fprintf(os.Stderr, "%v\n", err)
        return
    }
    defer f.Close()
    readthing(f)
}
```

## Data Structures

```
type Thing struct {
    Top int `xml:"top,attr"`
    Left int `xml:"left,attr"`
    Sep int `xml:"sep,attr"`
    Item []item `xml:"item"`
}

type item struct {
    Width int `xml:"width,attr"`
    Height int `xml:"height,attr"`
    Name string `xml:"name,attr"`
    Color string `xml:"color,attr"`
    Text string `xml:",chardata"`
}
```

## Parse and Load

```
func readthing(r io.Reader) {
    var t Thing
    err := xml.NewDecoder(r).Decode(&t)
    if err != nil {
        fmt.Fprintf(os.Stderr, "%v\n", err)
        return
    }
    drawthing(t)
}
```

## Flags and Main

```
var (
    width = flag.Int("w", 1024, "width")
    height = flag.Int("h", 768, "height")
    canvas = svg.New(os.Stdout)
)

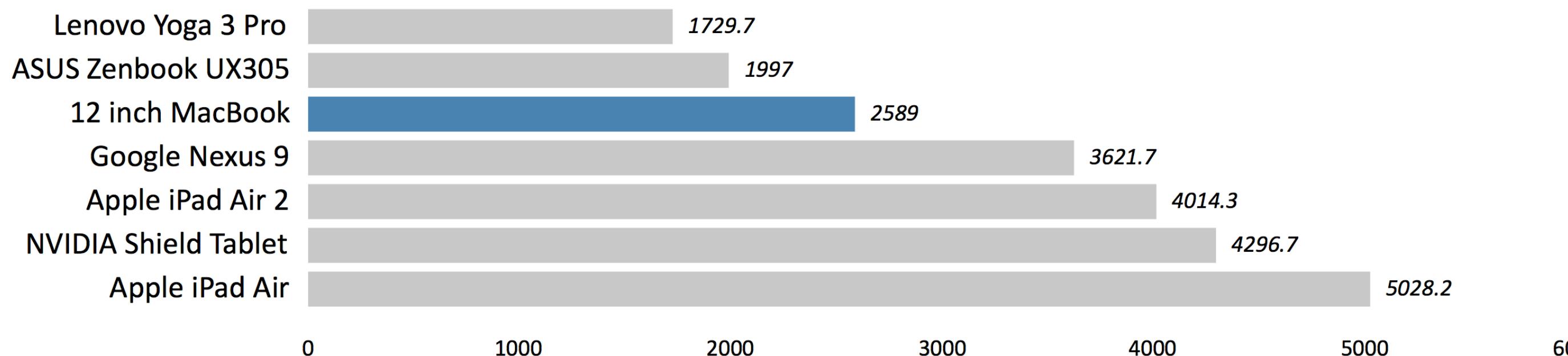
func main() {
    flag.Parse()
    for _, f := range flag.Args() {
        canvas.Start(*width, *height)
        dothing(f)
        canvas.End()
    }
}
```

## Draw

```
func drawthing(t Thing) {
    x := t.Left
    y := t.Top
    thingfmt := "font-size:%dpx;fill:%s"
    tfmt := "%s:%s/%s"
    for _, v := range t.Item {
        s := fmt.Sprintf(thingfmt,
                         v.Width/2, v.Color)
        canvas.Circle(x, y, v.Height/4,
                      "fill:"+v.Color)
        canvas.Text(x+t.Sep, y,
                    fmt.Sprintf(tfmt,
                               v.Name, v.Text, v.Color, s))
        y += v.Height
    }
}
```

```
<barchart title="2015 MacBook Benchmarks">
  <note>Source: AnandTech, April 14, 2015</note>
  <bdata scale="0,6000,1000"
    title="Mozilla Kraken 1.1 (native browser, milliseconds)">
    <bitem value="1729.7" name="Lenovo Yoga 3 Pro"/>
    <bitem value="1997.0" name="ASUS Zenbook UX305"/>
    <bitem color="steelblue" value="2589.0" name="12 inch MacBook"/>
    <bitem value="3621.7" name="Google Nexus 9"/>
    <bitem value="4014.3" name="Apple iPad Air 2"/>
    <bitem value="4296.7" name="NVIDIA Shield Tablet"/>
    <bitem value="5028.2" name="Apple iPad Air"/>
  </bdata>
</barchart>
```

*Mozilla Kraken 1.1 (native browser, milliseconds)*



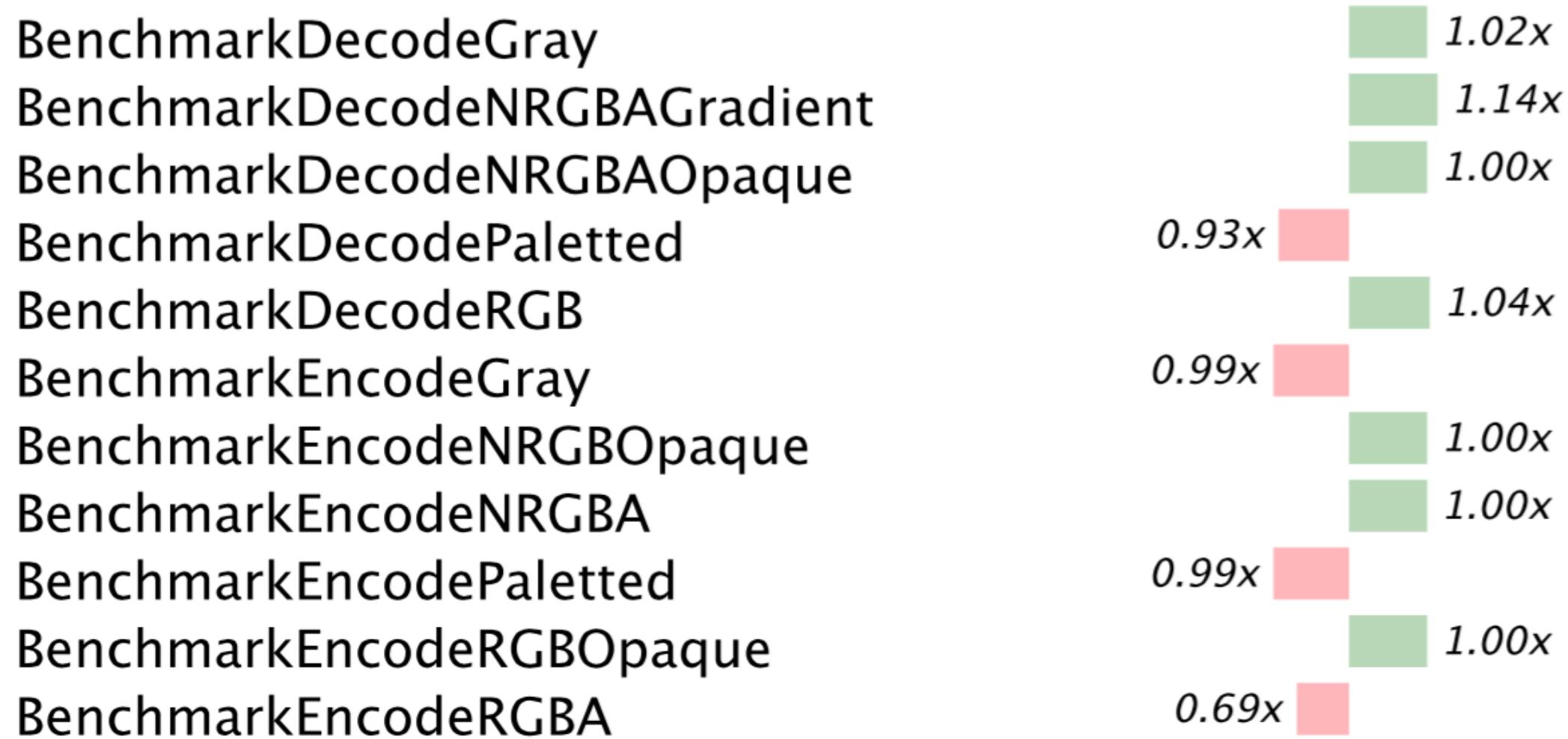
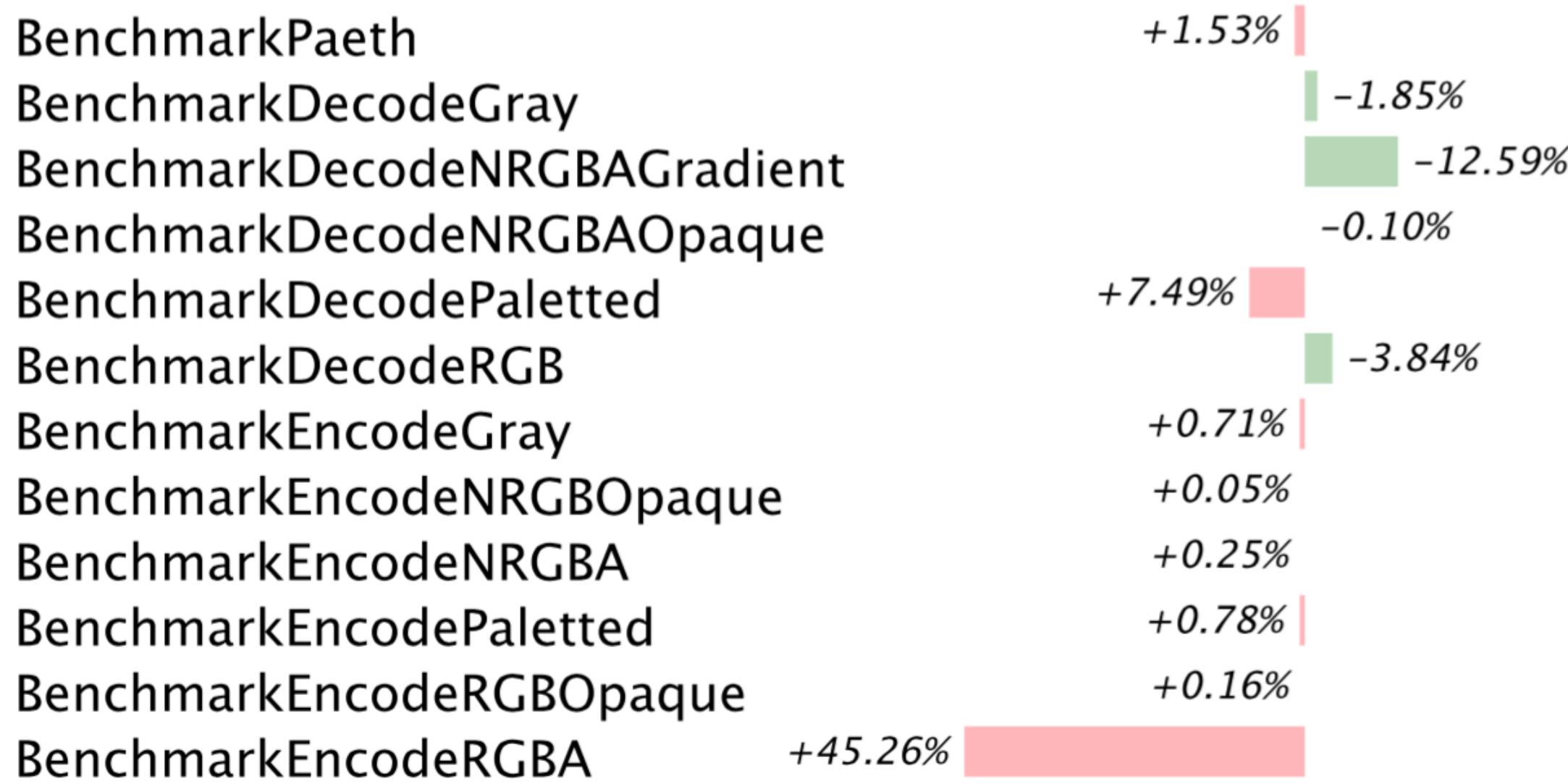
# 2015 MacBook Benchmarks

*Source: AnandTech, April 14, 2015*

# SVGo Clients

# benchpng.txt

*image/png package: Go 1.3 vs Go 1.4*

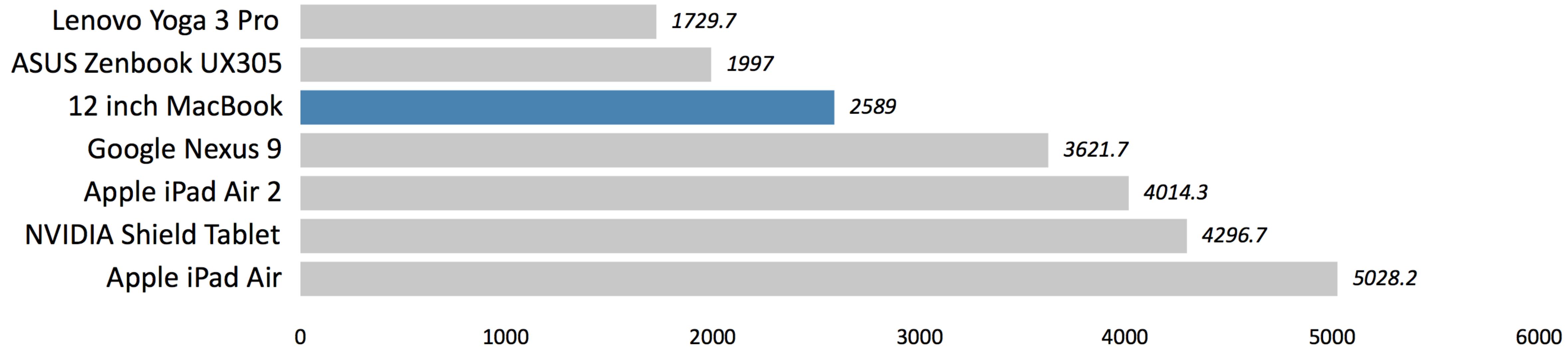




## Sample Bullet Graph

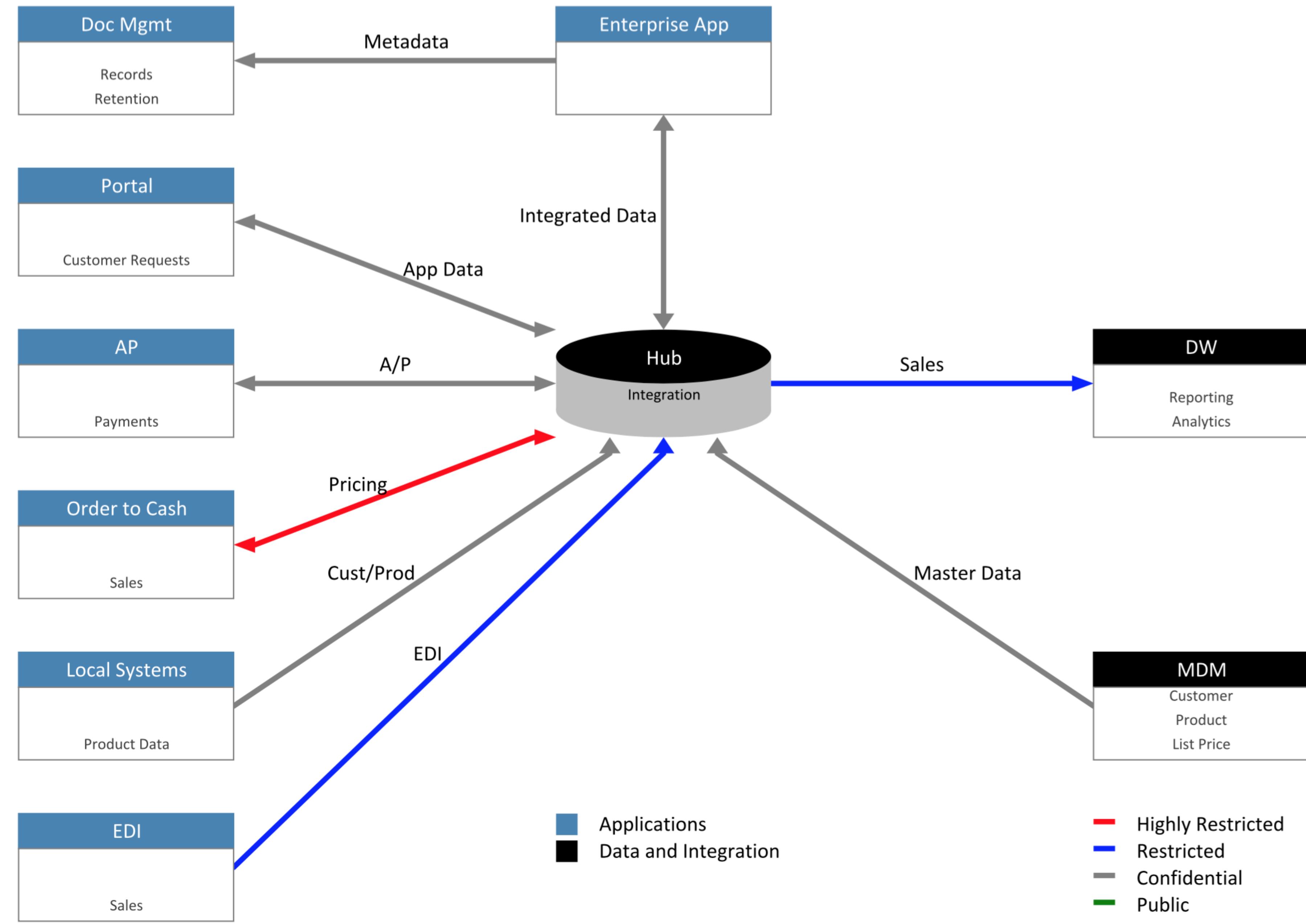
The bullet graph features a single, primary measure (for example, current year-to-date revenue) compares that measure to one or more other measures to enrich its meaning, for example, compared to a target), and displays it in the context of qualitative ranges of performance, such as poor, satisfactory, and good.

### *Mozilla Kraken 1.1 (native browser, milliseconds)*



## 2015 MacBook Benchmarks

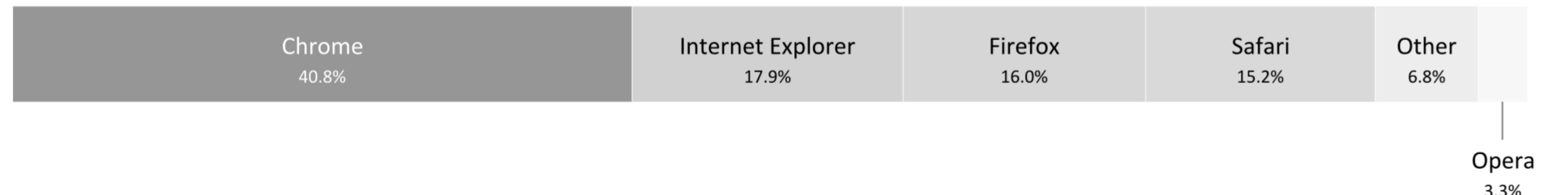
*Source: AnandTech, April 14, 2015*



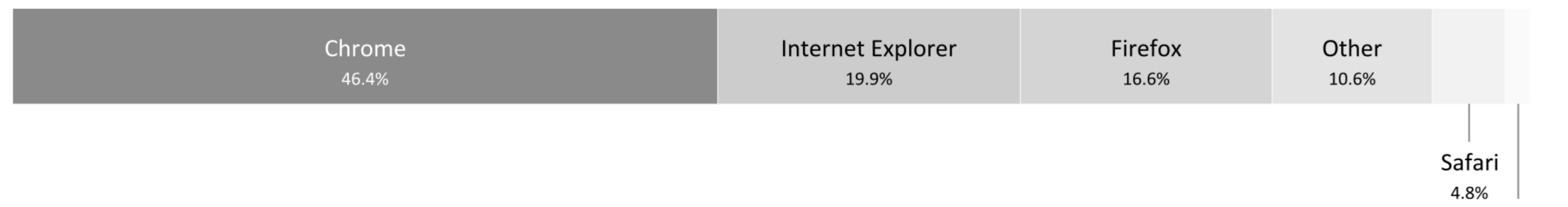
## StatCounter



## W3C Counter



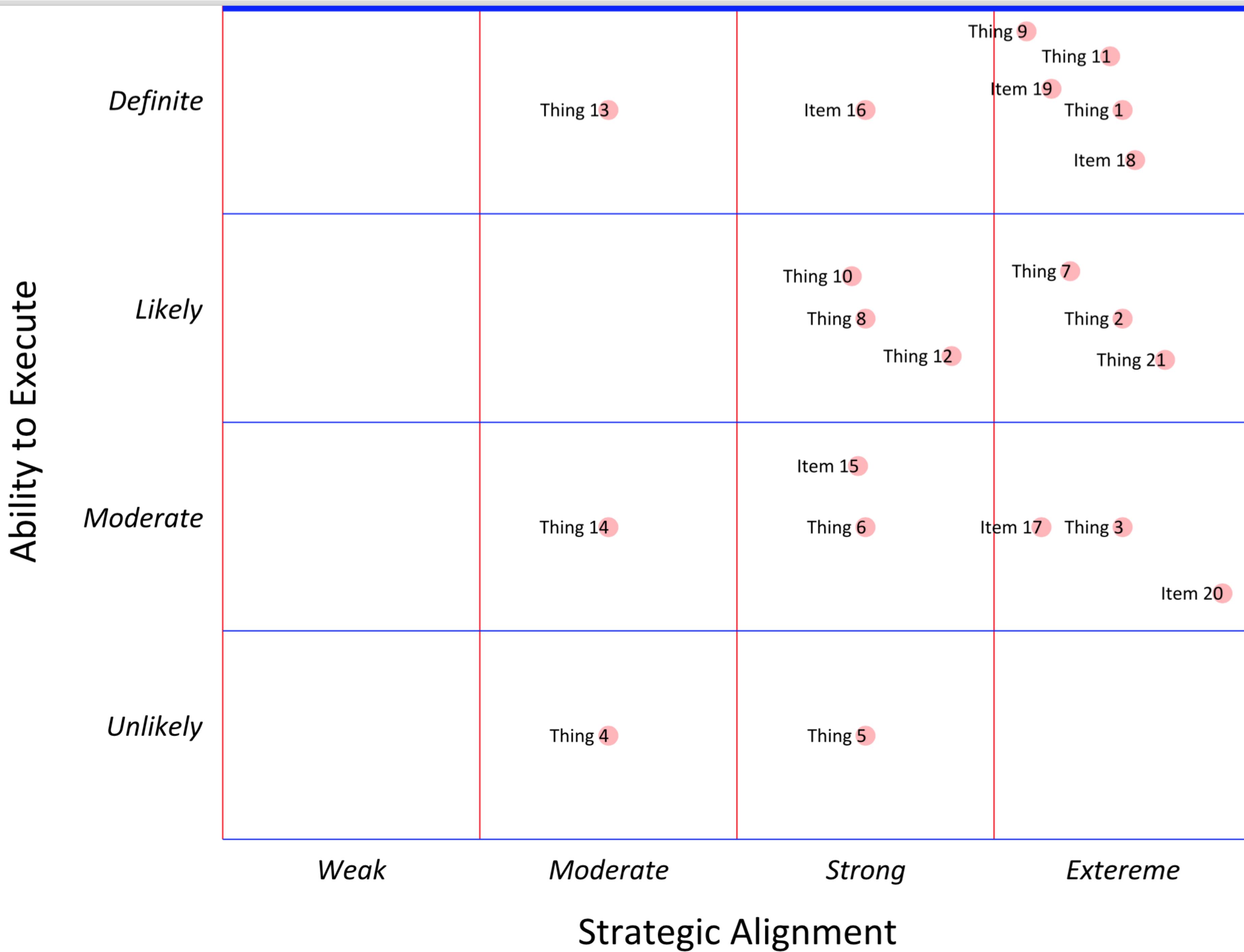
## Wikimedia



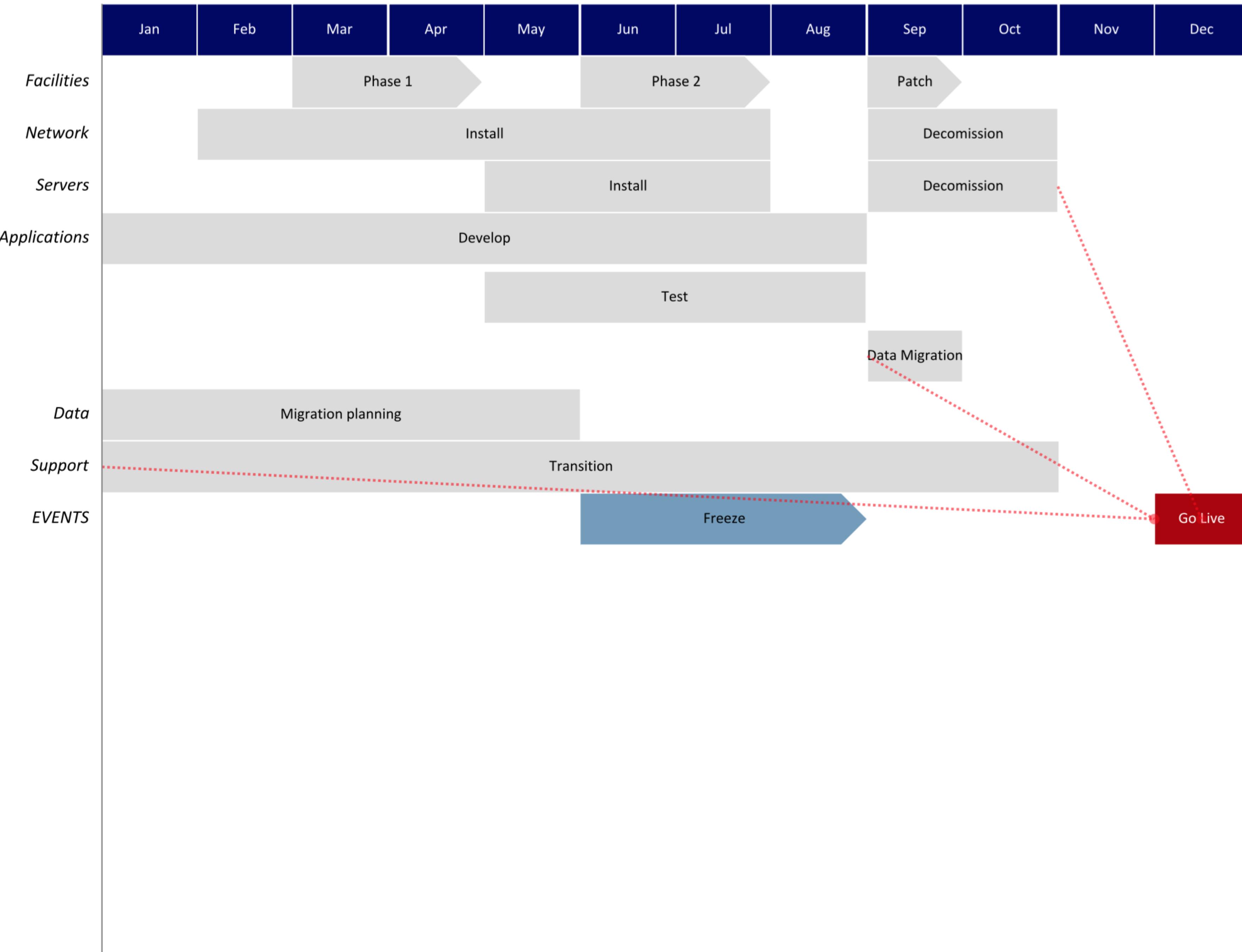
## Net Applications

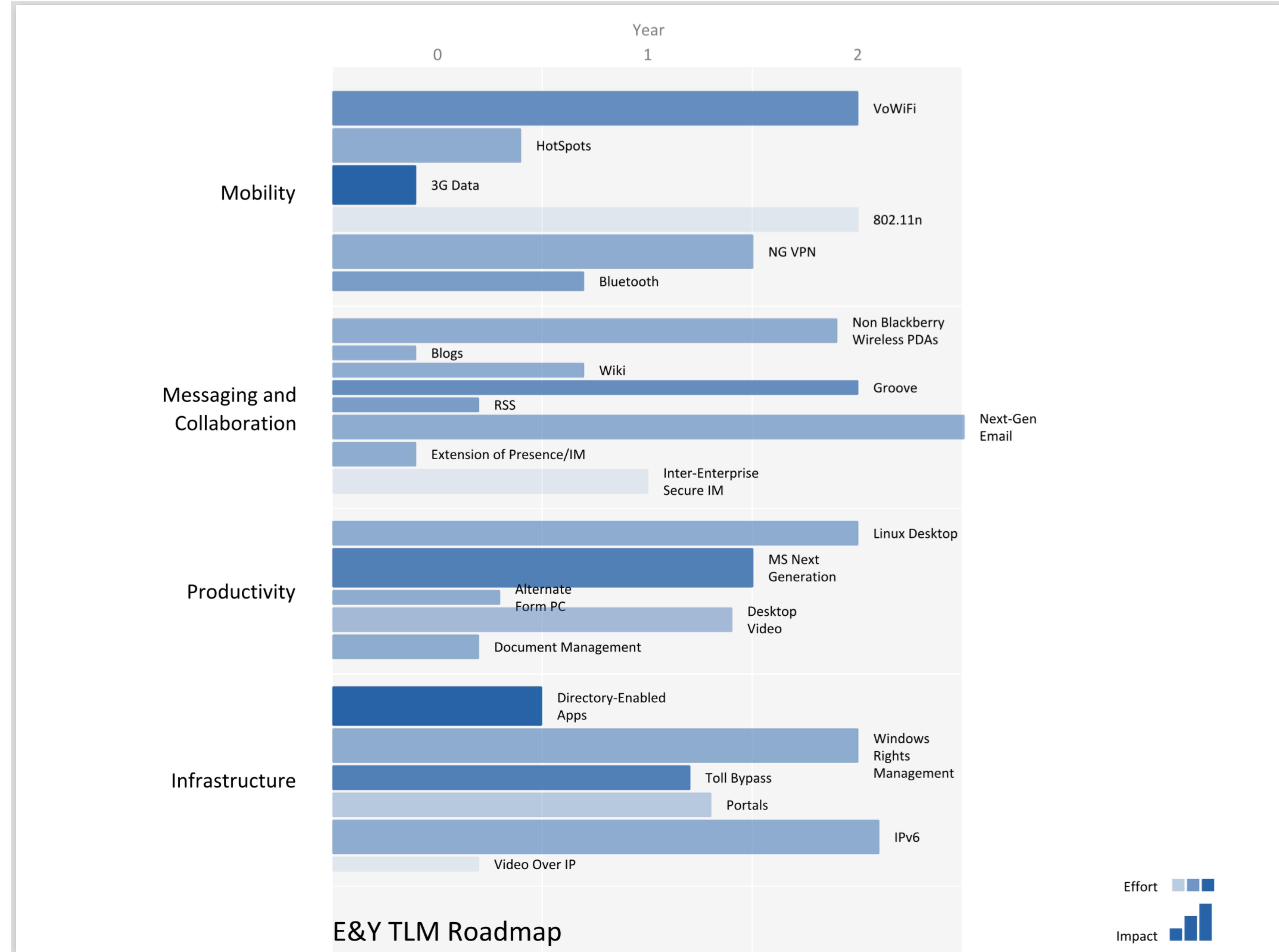


## Browser Market Share



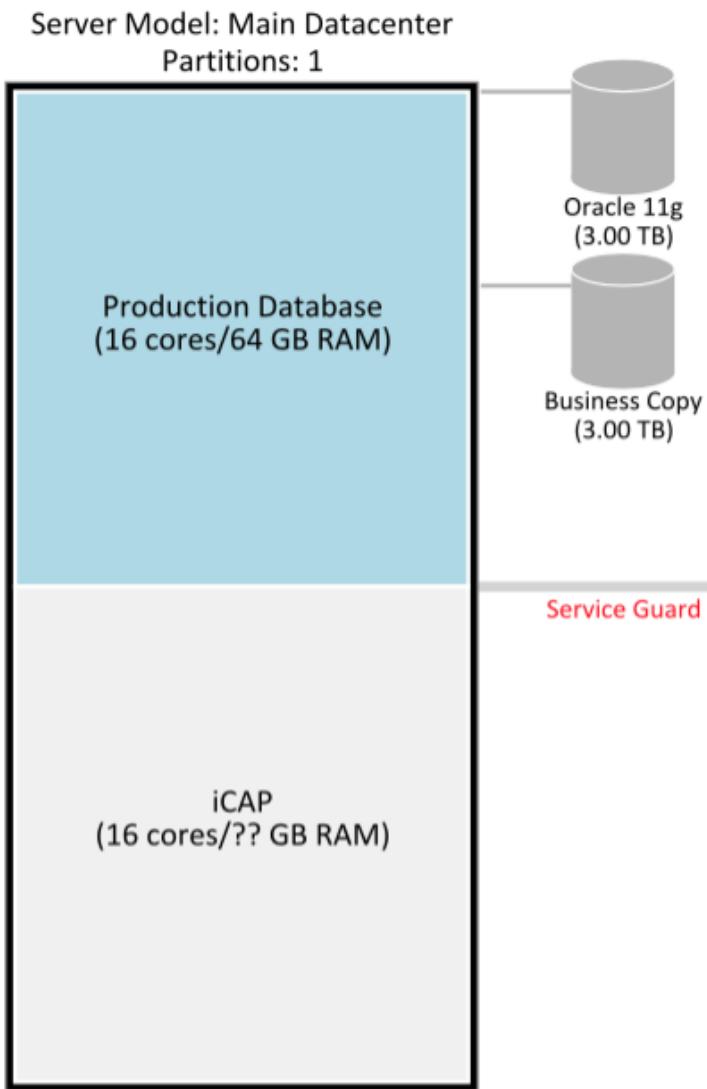
# PLANNING TO PLAN



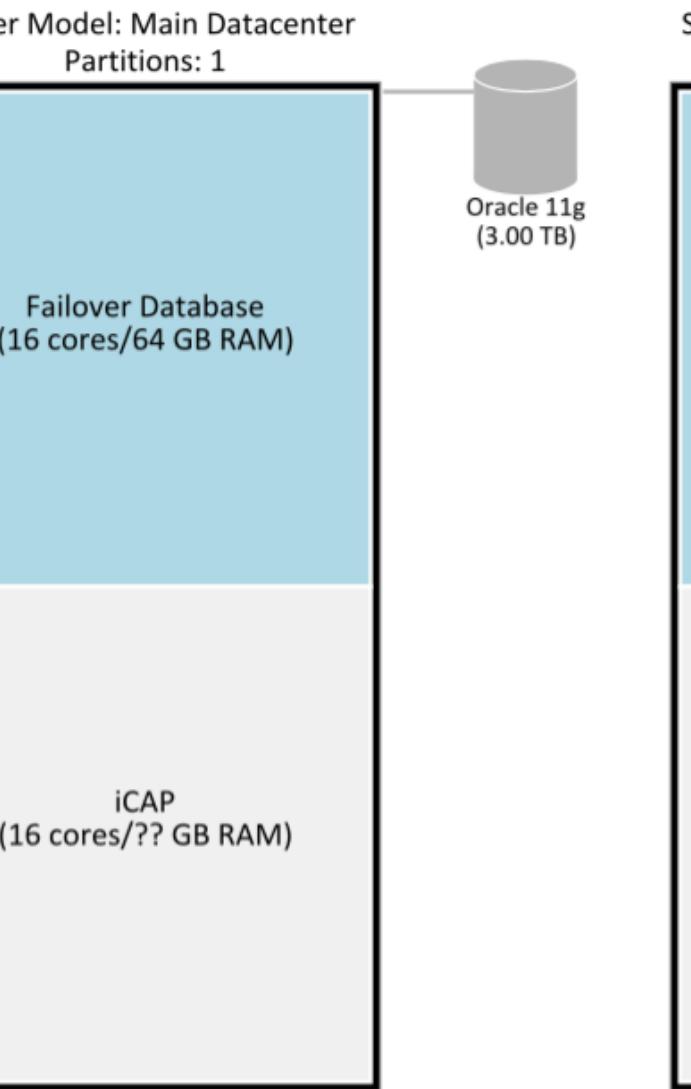




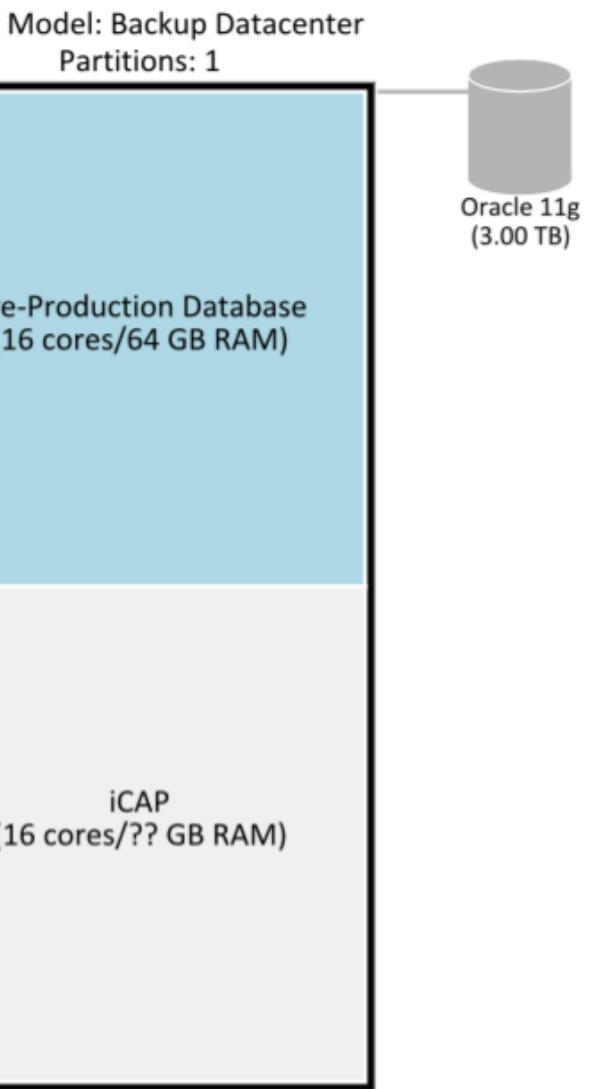
## PRODUCTION



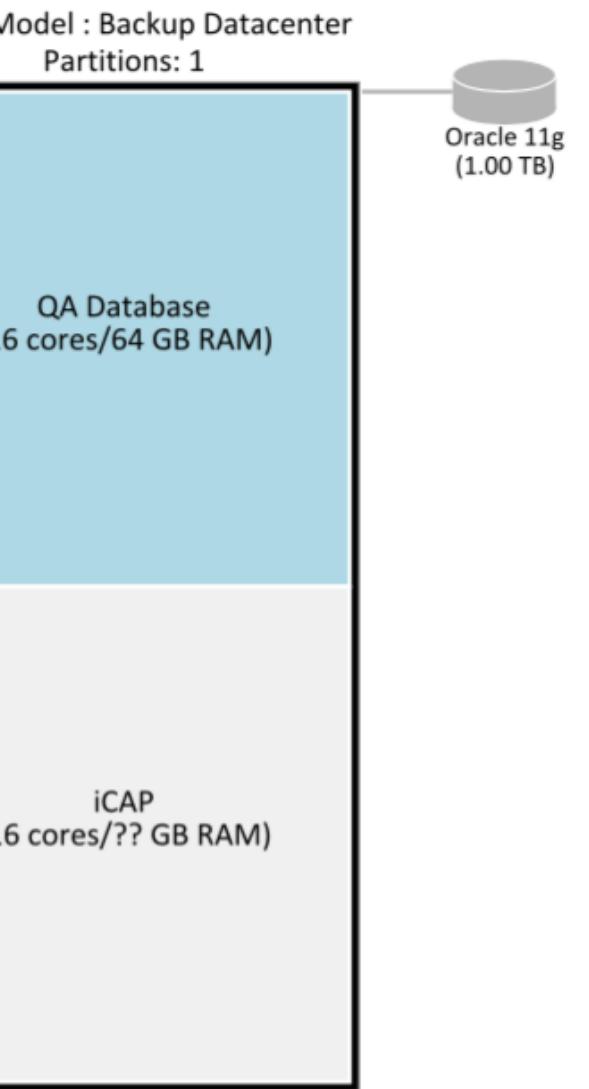
## FAILOVER



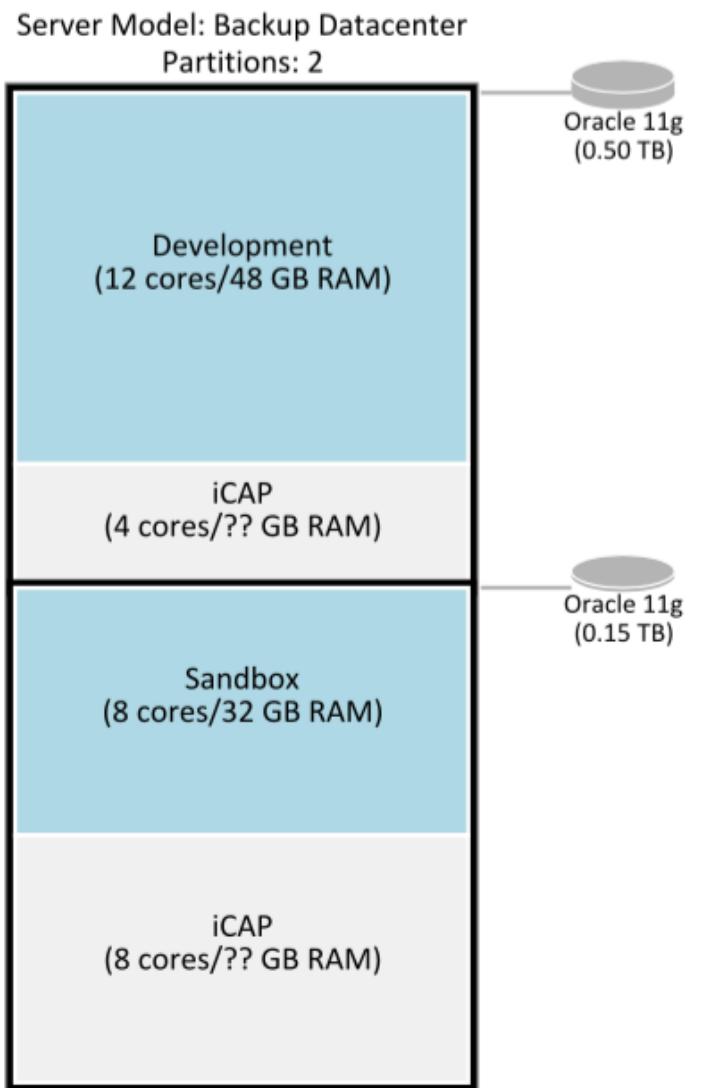
## PRE-PROD



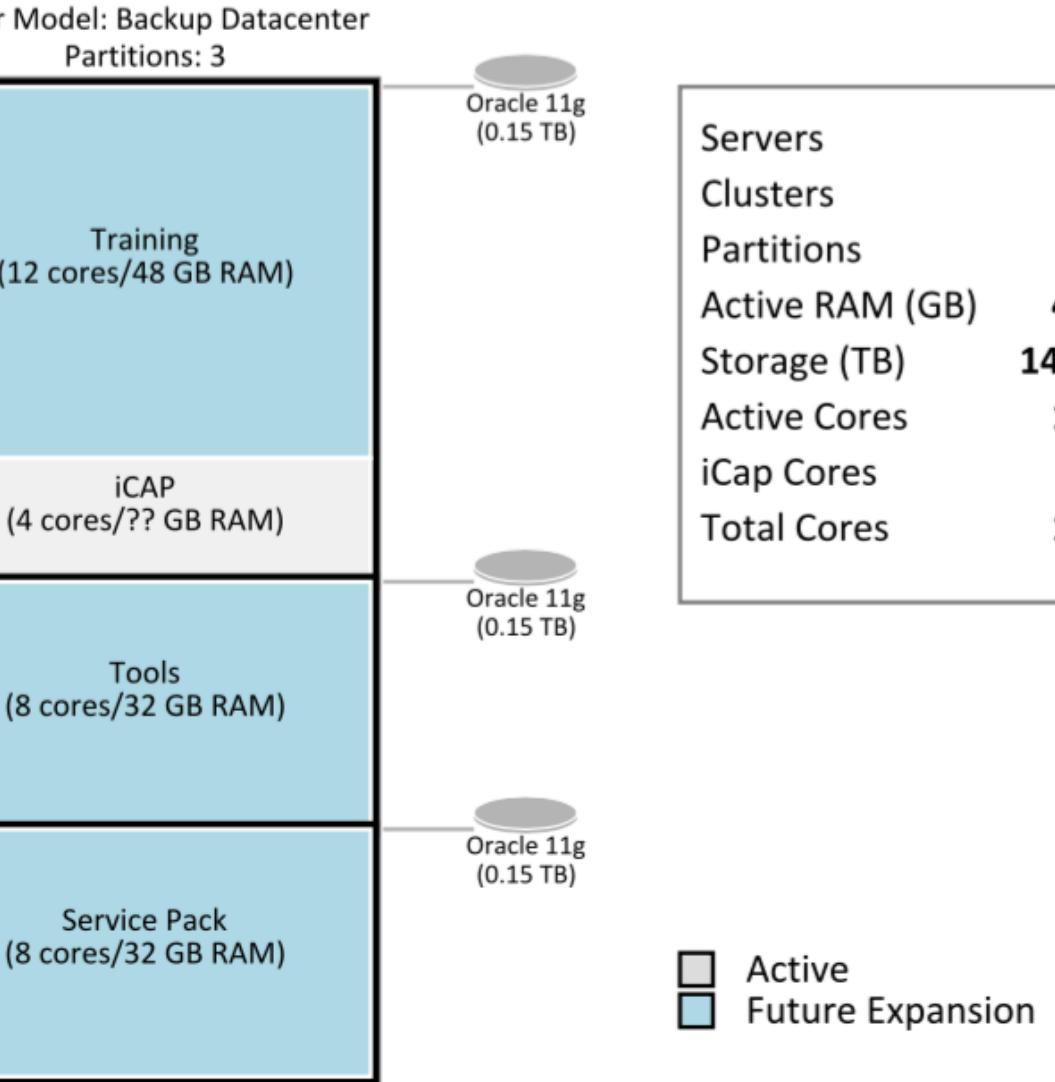
## QA



## DEV



## TRAINING/TOOLS

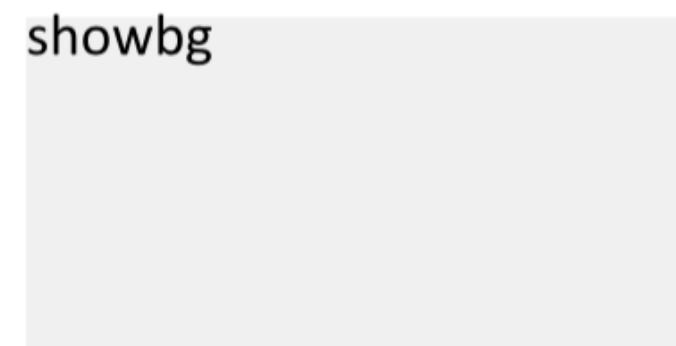


Servers	<b>6</b>
Clusters	<b>1</b>
Partitions	<b>9</b>
Active RAM (GB)	<b>448</b>
Storage (TB)	<b>14.10</b>
Active Cores	<b>112</b>
iCap Cores	<b>80</b>
Total Cores	<b>192</b>

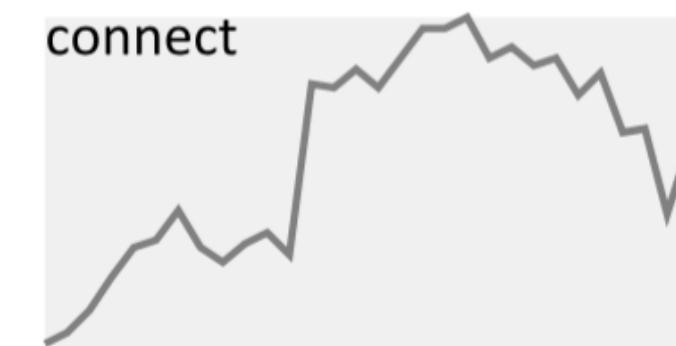
□ Active  
■ Future Expansion

# Server Layout

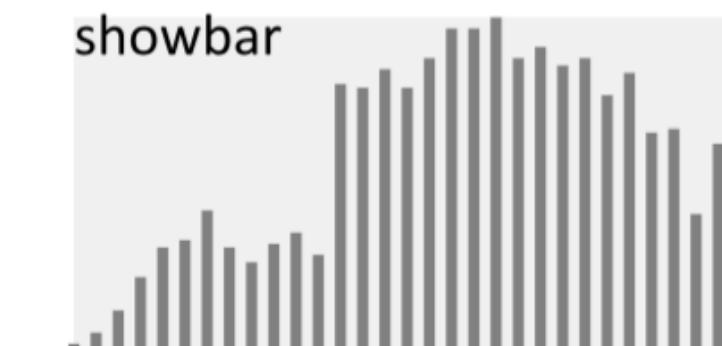
showbg



connect



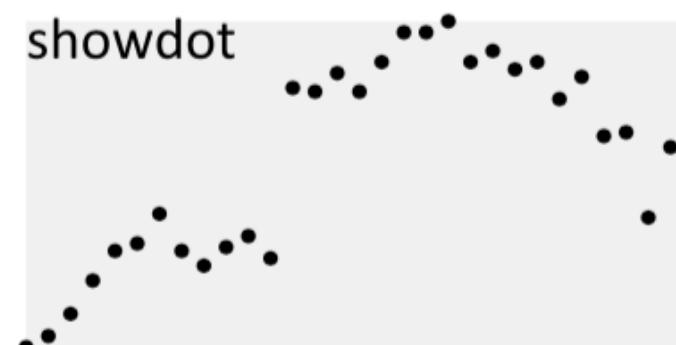
showbar



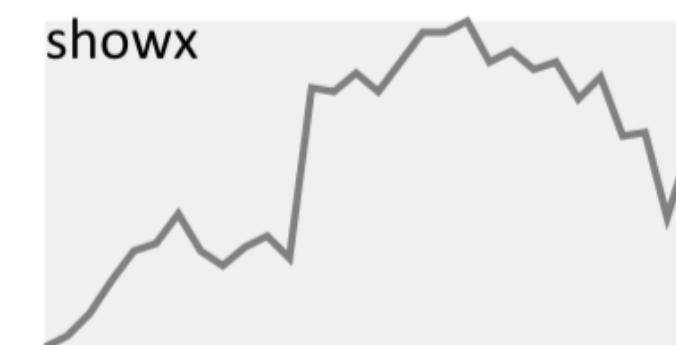
area



showdot



showx



623.0  
589.8  
556.5  
523.2  
490.0

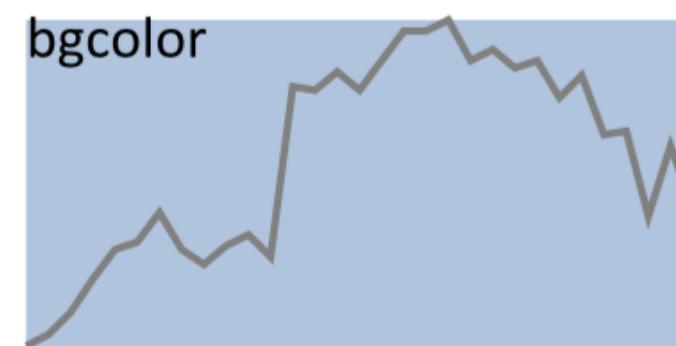
showy



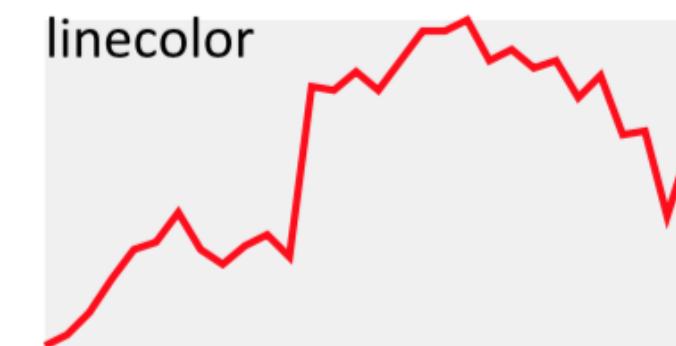
test.d



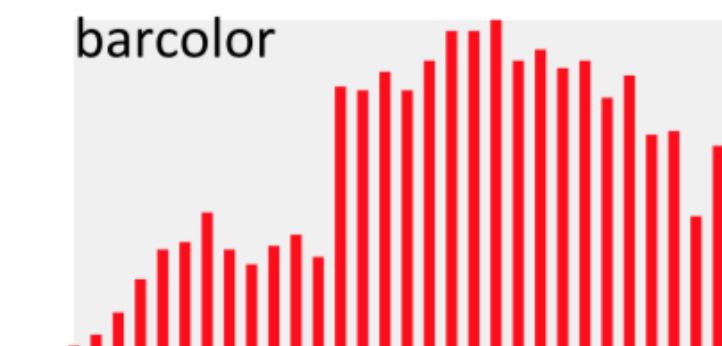
bgcolor



linecolor



barcolor



hcolor



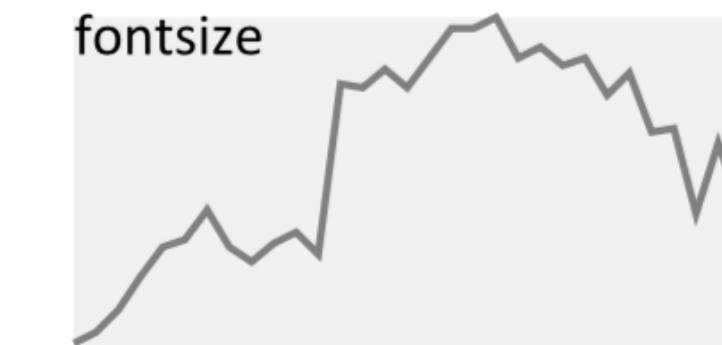
dotcolor



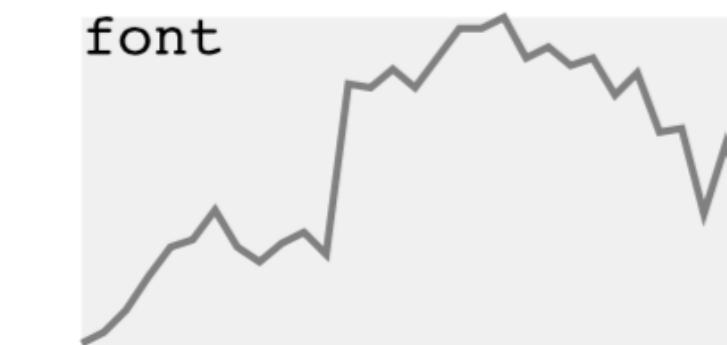
labelcolor



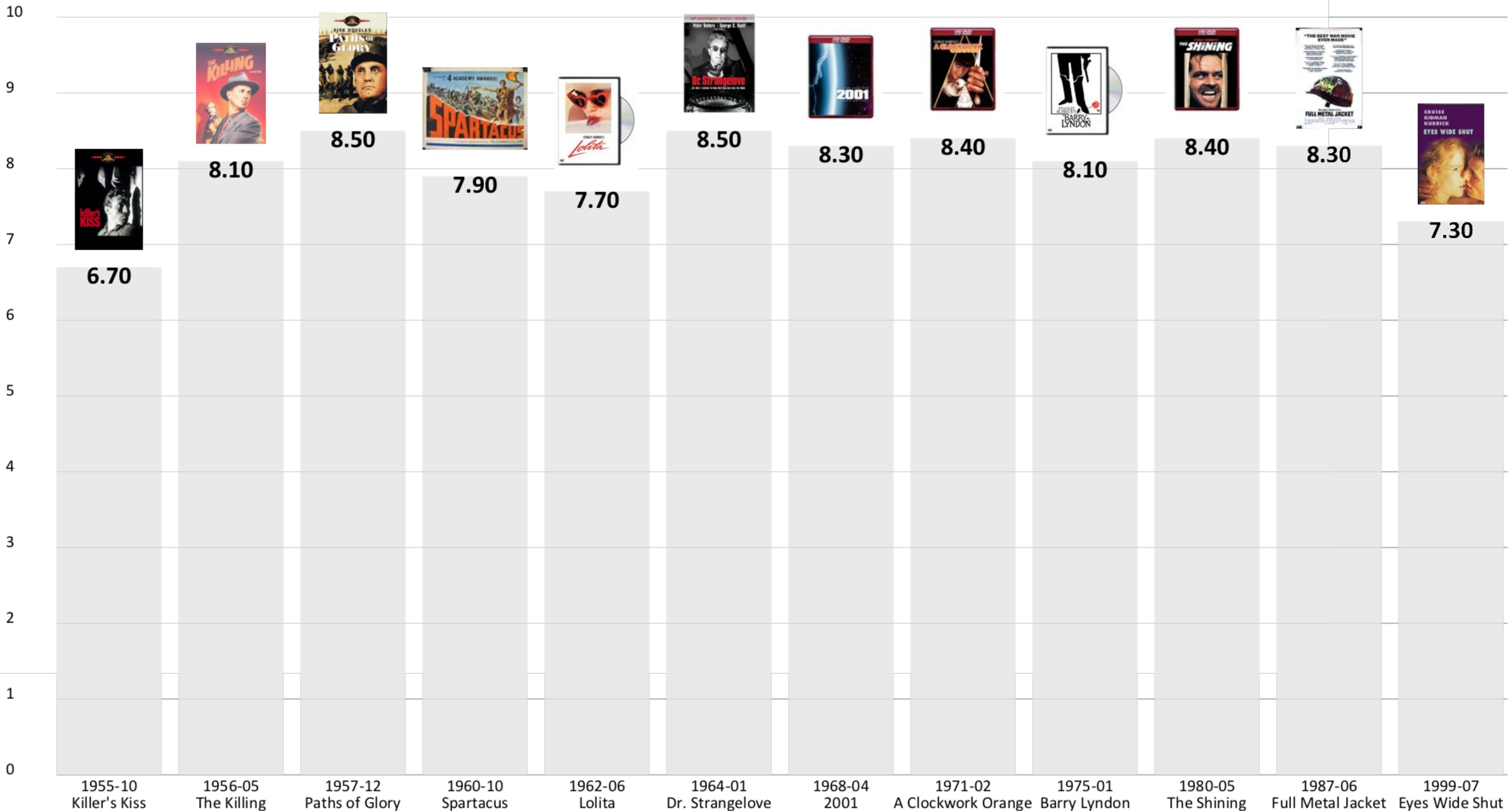
fontsize



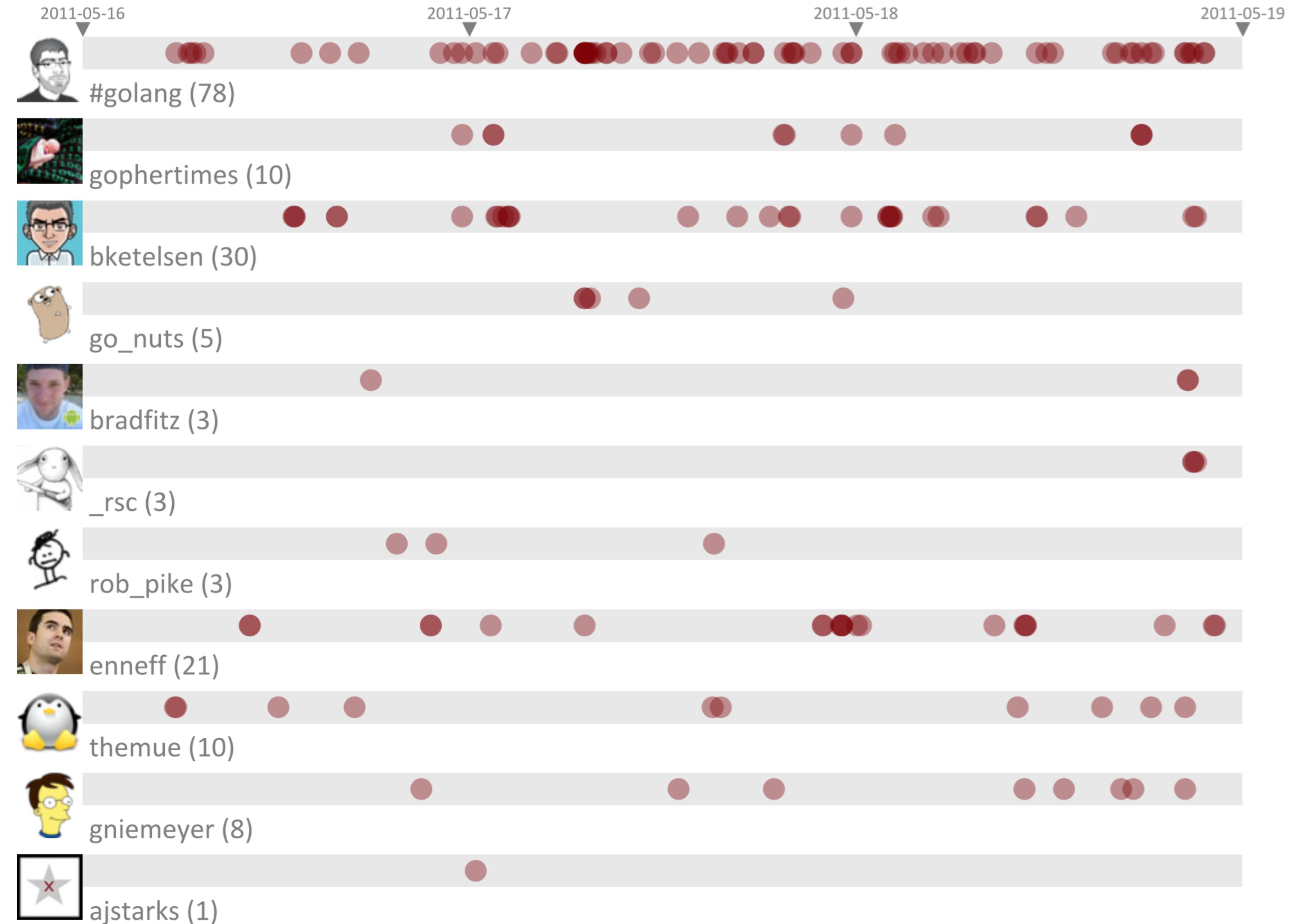
font



# IMDB Ratings of Kubrick Films



# Twitter Update Frequency



# JONES

- LAYER TENNIS SEASON 4 WEEK 6 MATCH COMMENTARY BY MIKE MONTEIRO -

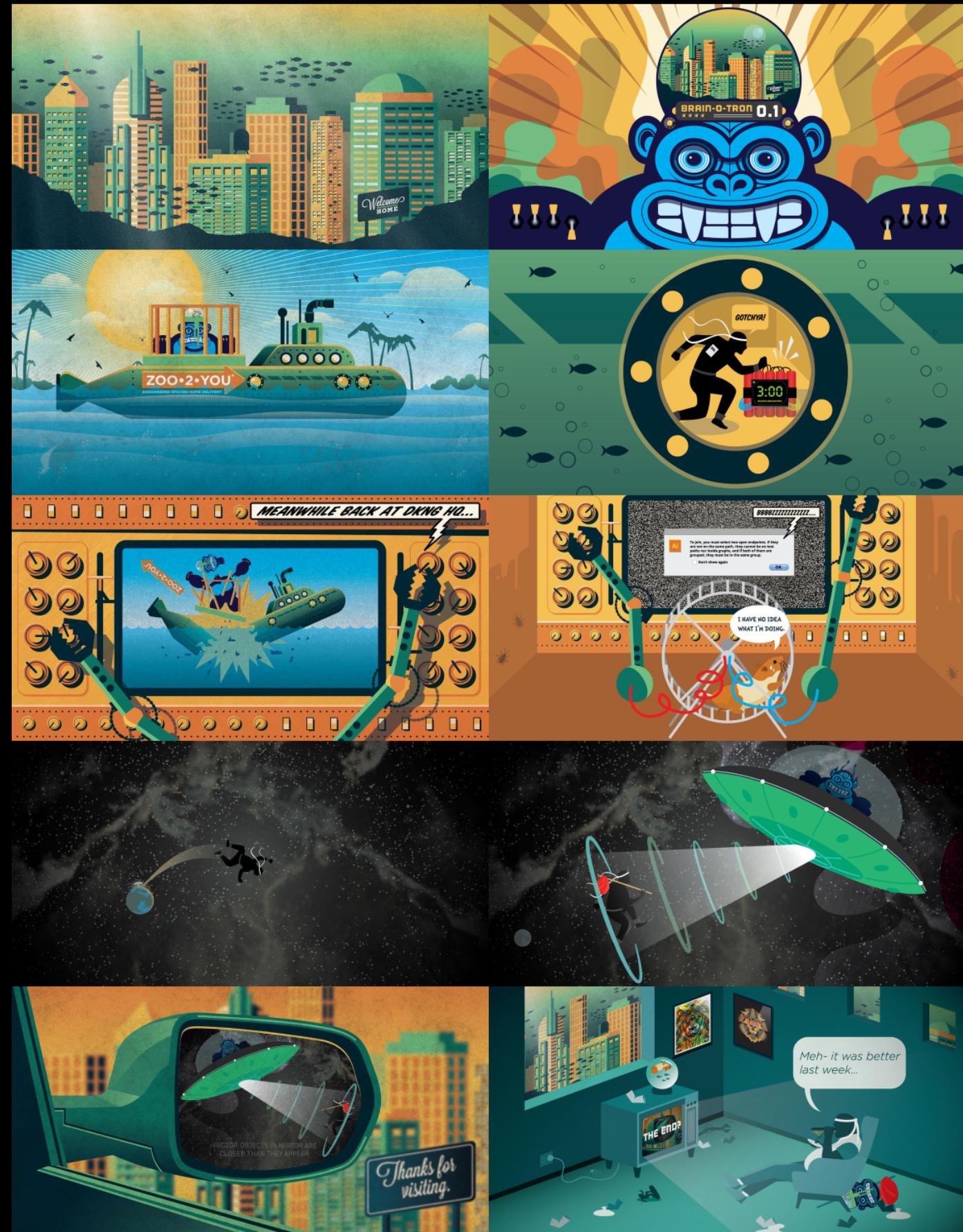
# MONTIEL



# DKNG

- LAYER TENNIS SEASON 4 WEEK 7 MATCH COMMENTARY BY JOHN GRUBER -

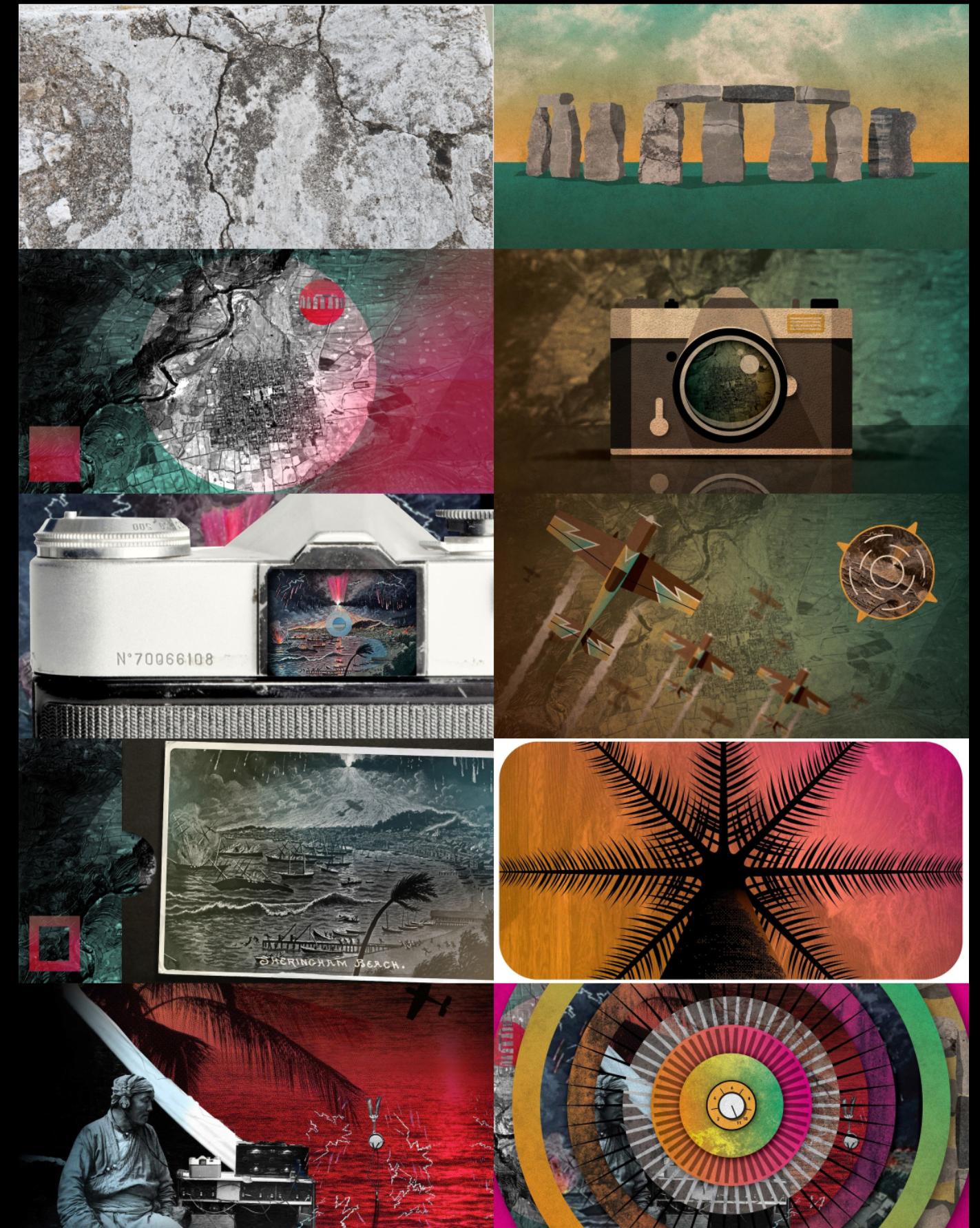
# DDL



# ANDERSON

- LAYER TENNIS SEASON 4 PLAYOFF QUARTERFINAL COMMENTARY BY BRYAN BEDELL -

# DKNG



CONTINO  
CASSARO



PUTNAM  
STOCKS



WHITE  
TAYLOR



HALL  
WARREN



STRAWBERRY LUNA  
DOUBLENAUT



JONES  
MONTIEL



DKNG  
DDL



SHAWNA X  
STEVENS



TAYLOR  
JONES



PUTNAM  
RAJKUMAR



REYES  
WHITE



ANDERSON  
DKNG



TAYLOR  
WHITE



RAJKUMAR  
ANDERSON



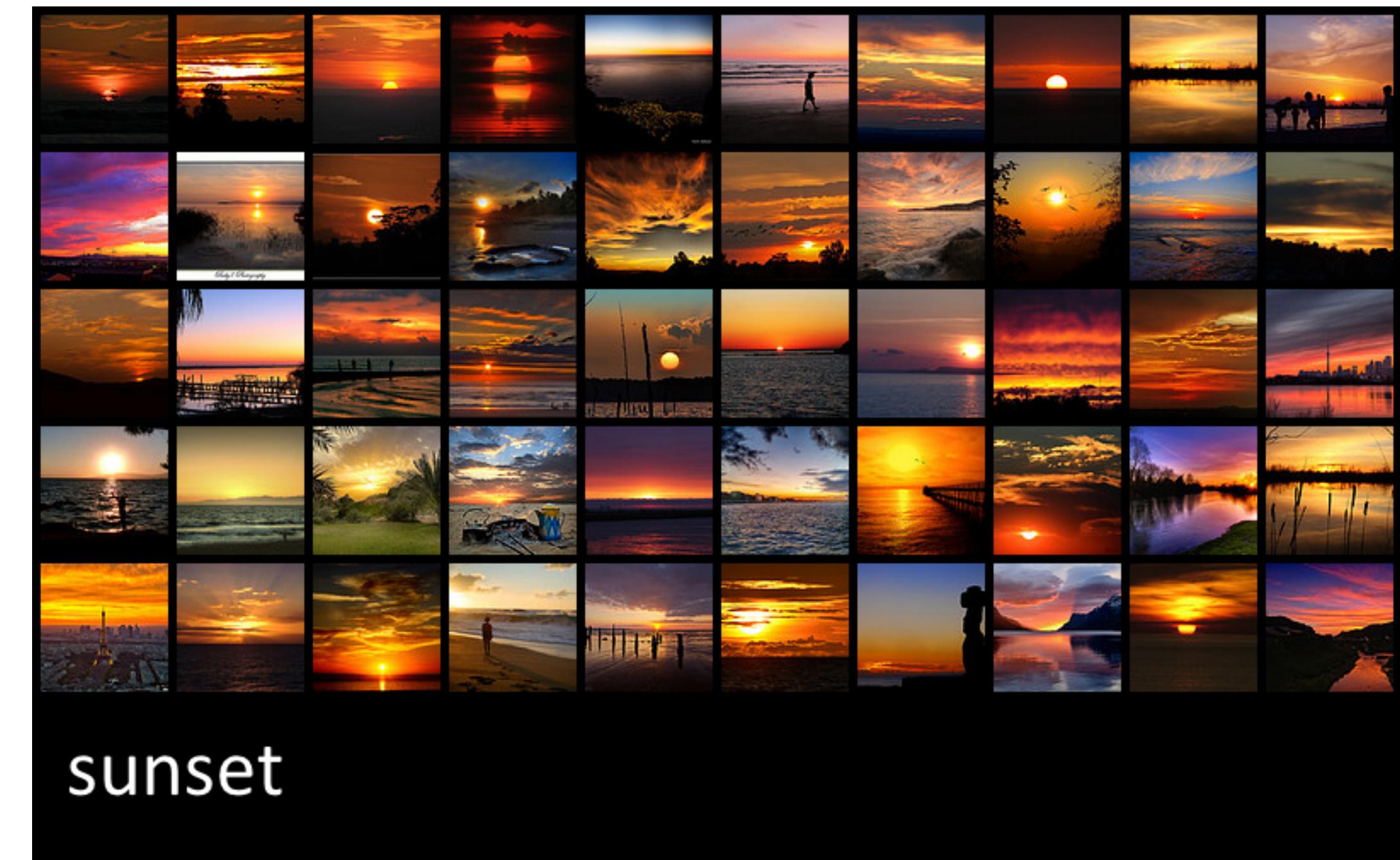
ANDERSON  
WHITE

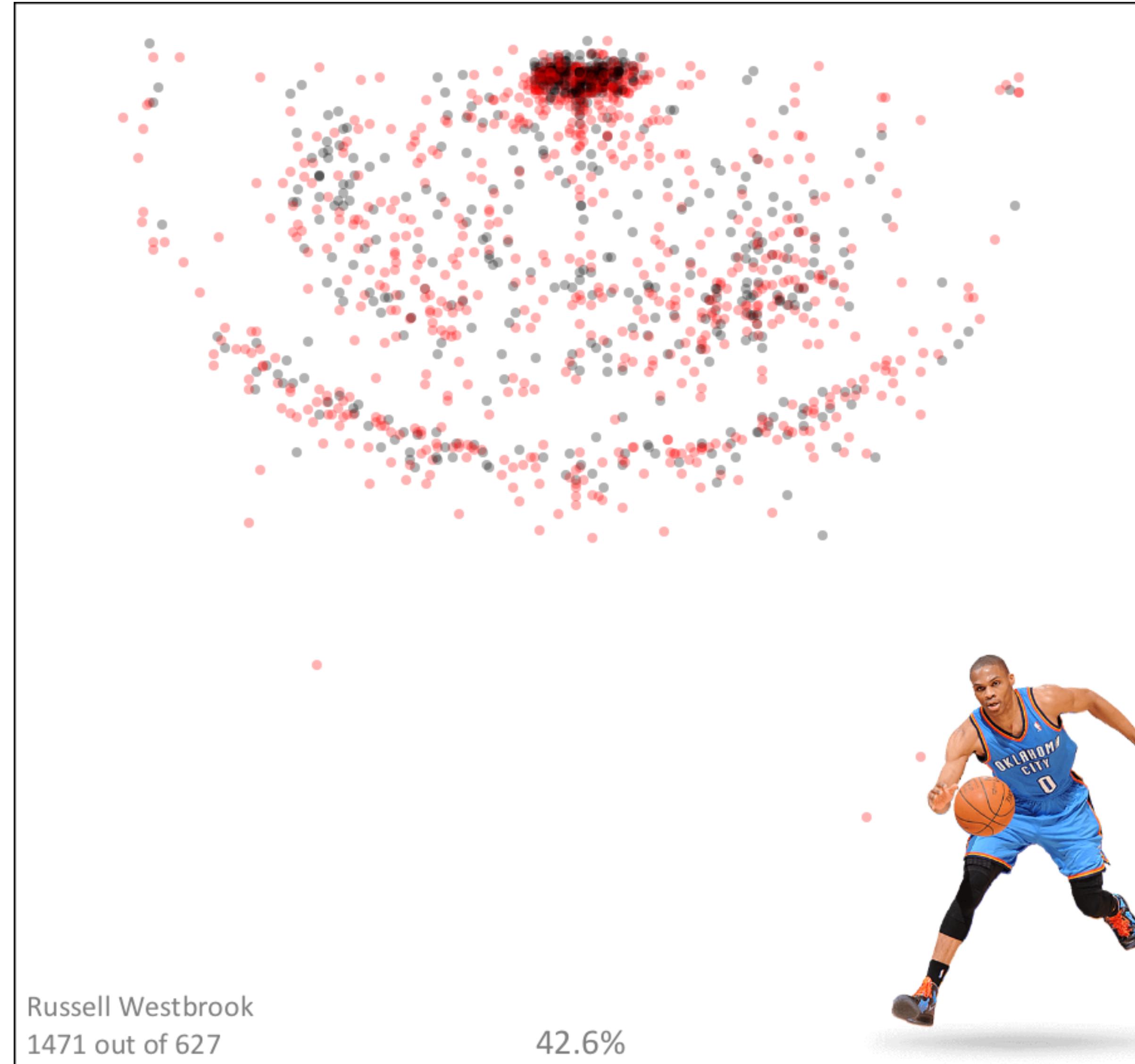


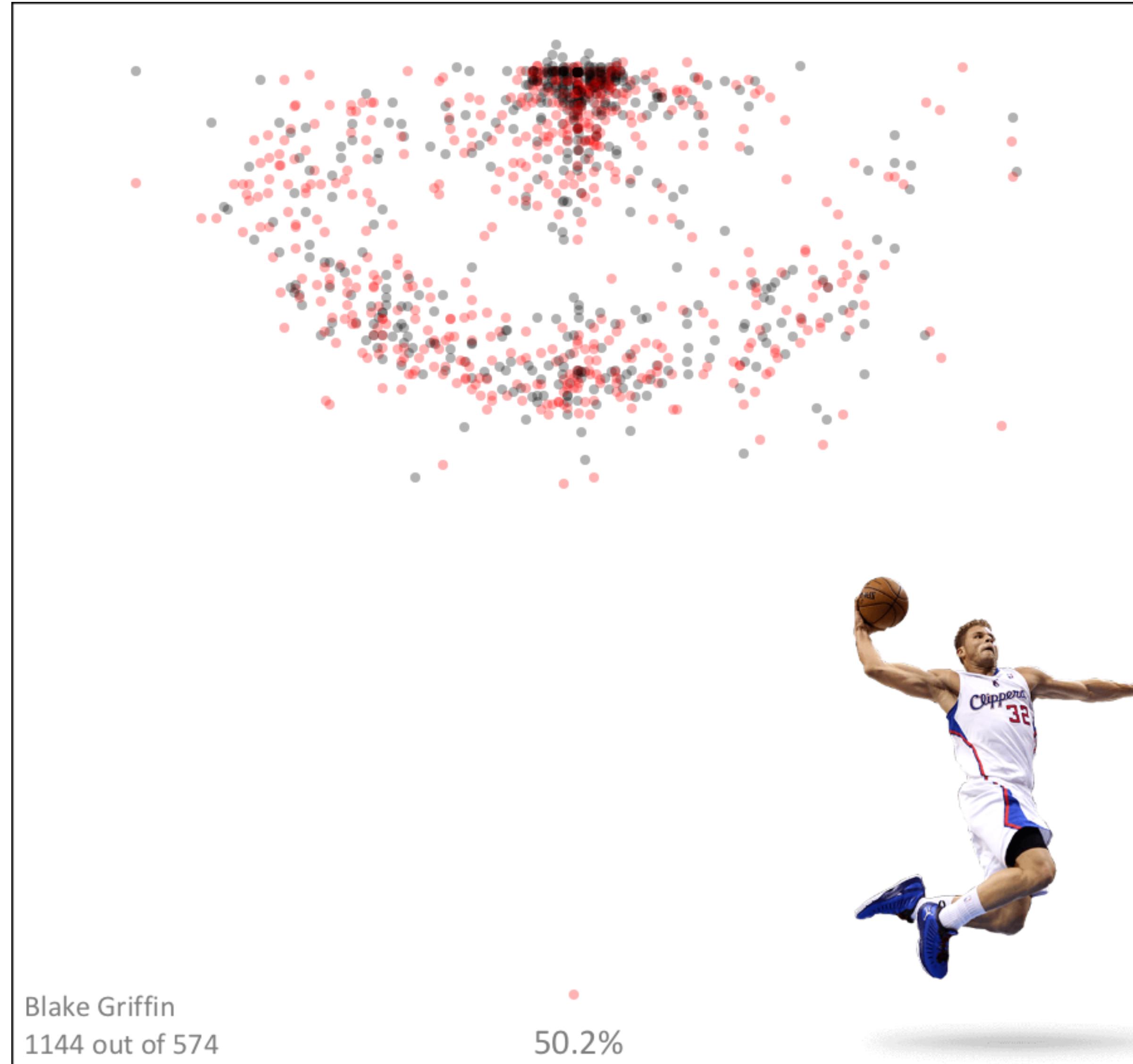
# f50 sunset

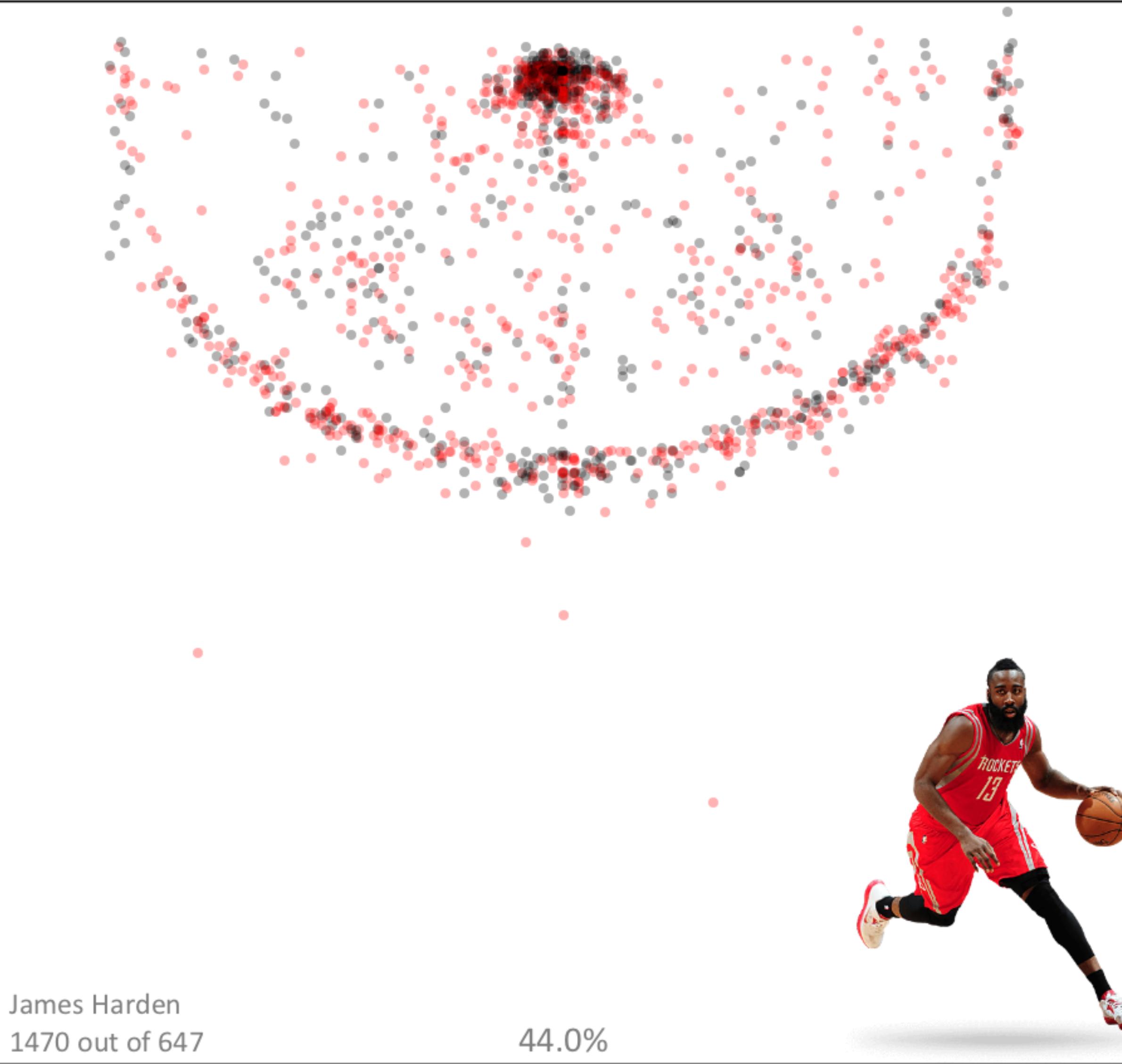
[https://api.flickr.com/services/rest/?method=flickr.photos.search&api\\_key=...&text=sunset&per\\_page=50&sort=interestingness-desc](https://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=...&text=sunset&per_page=50&sort=interestingness-desc)

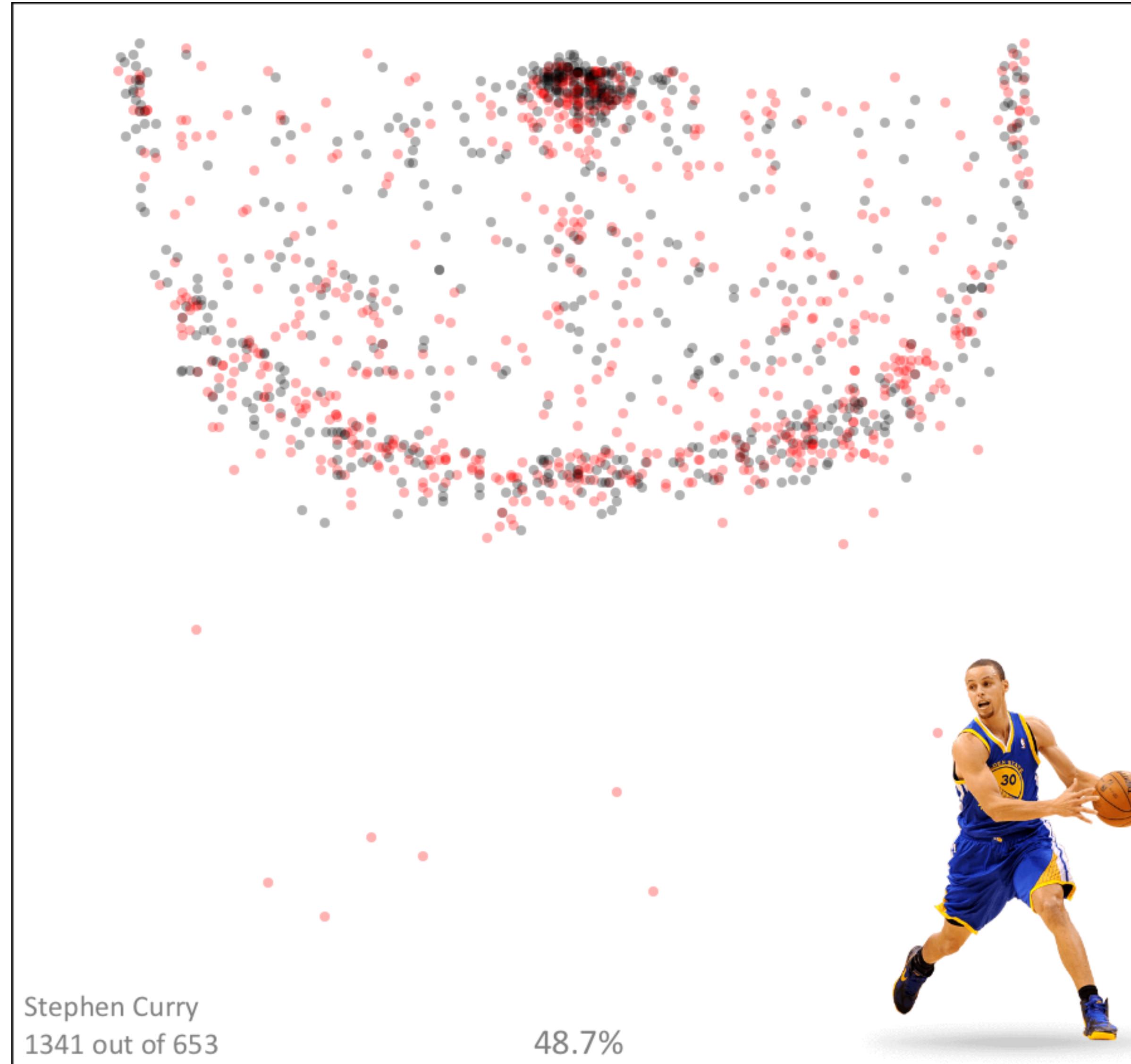
```
<?xml version="1.0" encoding="utf-8" ?>
<rsp stat="ok">
  <photos page="1" pages="105615" perpage="50" total="5280747">
    <photo id="4671838925" ... secret="b070f3363e" server="4068" farm="5" ... />
    <photo id="3590142202" ... secret="c46752e4d8" server="2441" farm="3" .../>
    ...
  </photos>
</rsp>
```

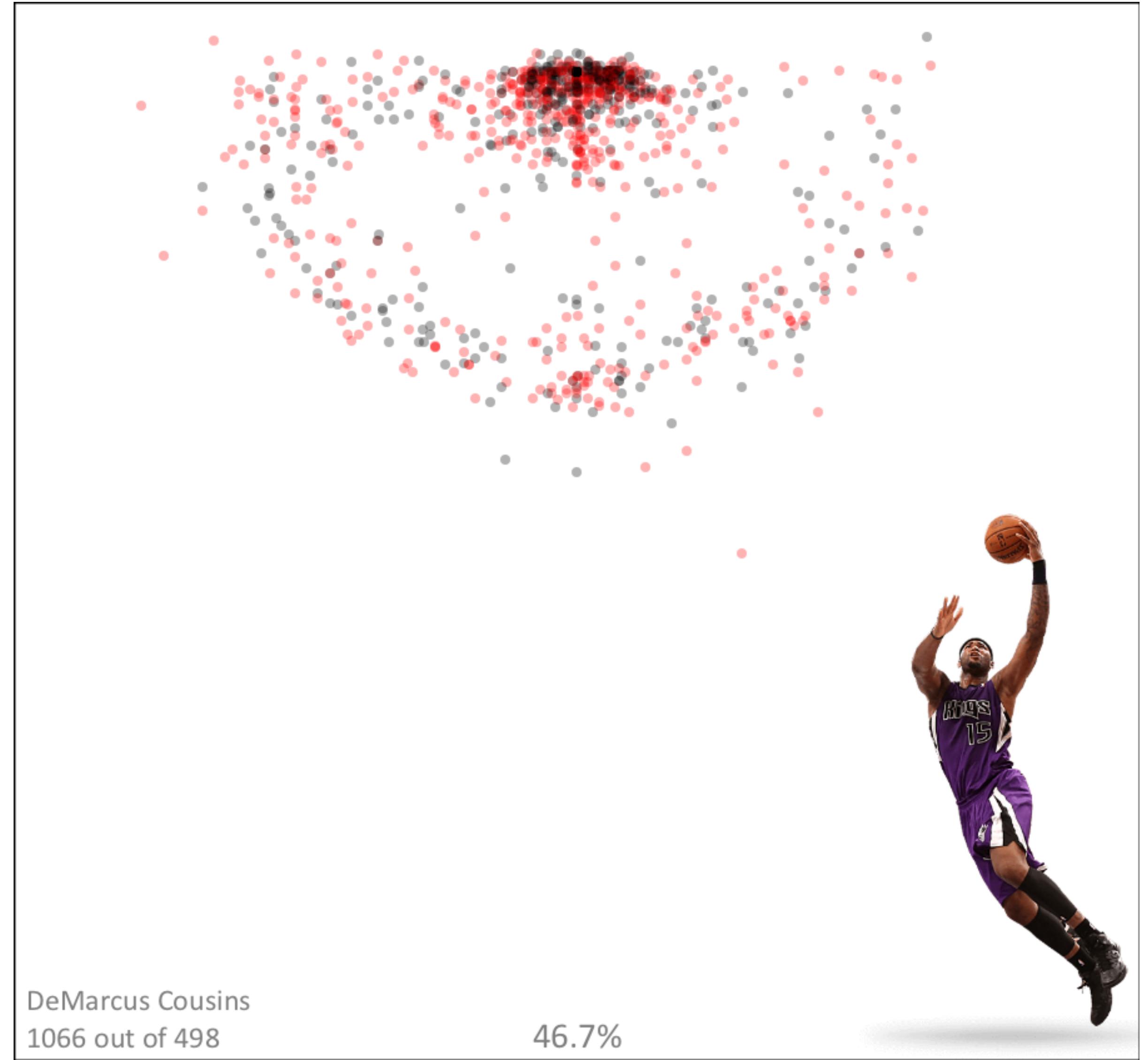


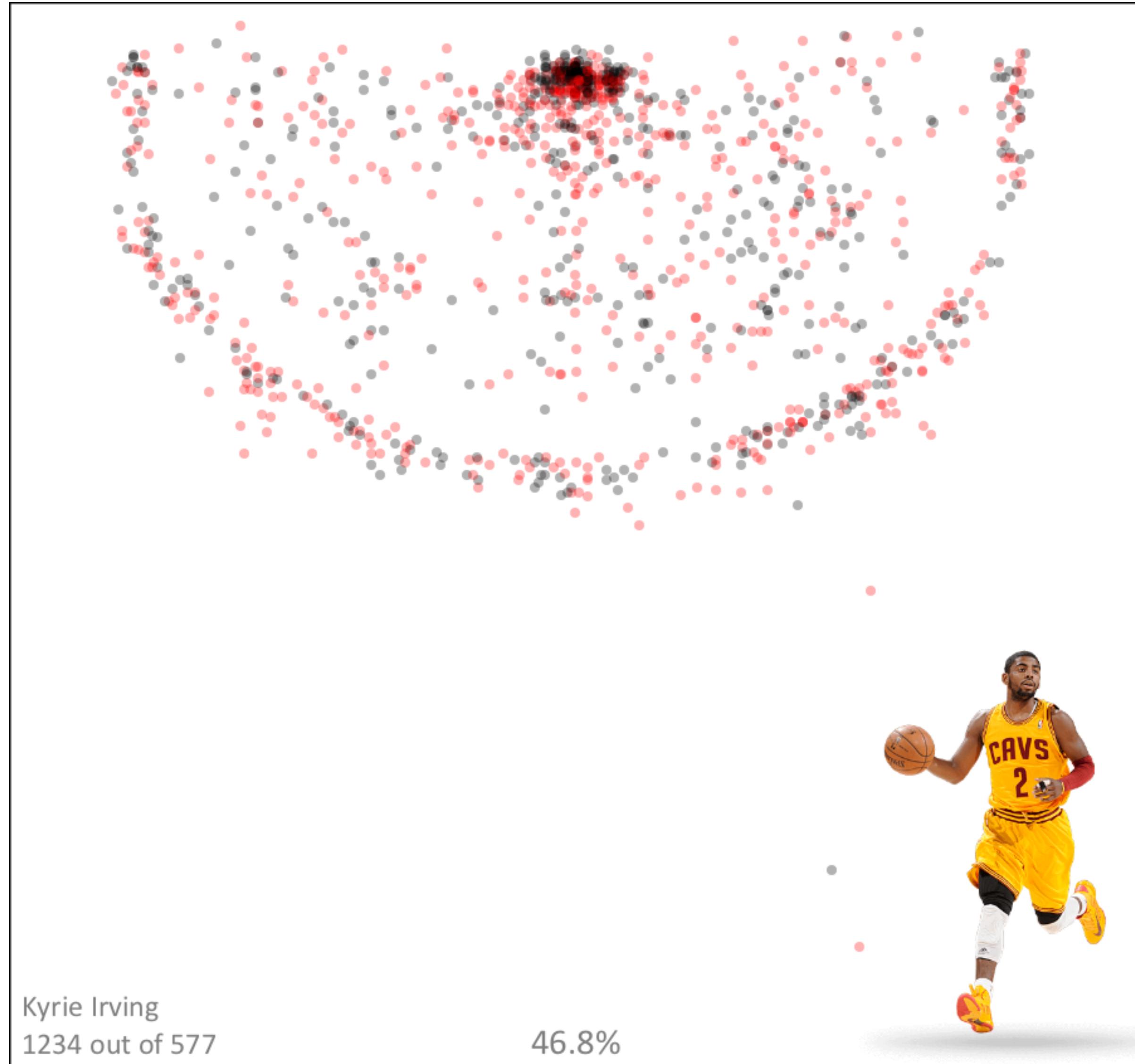


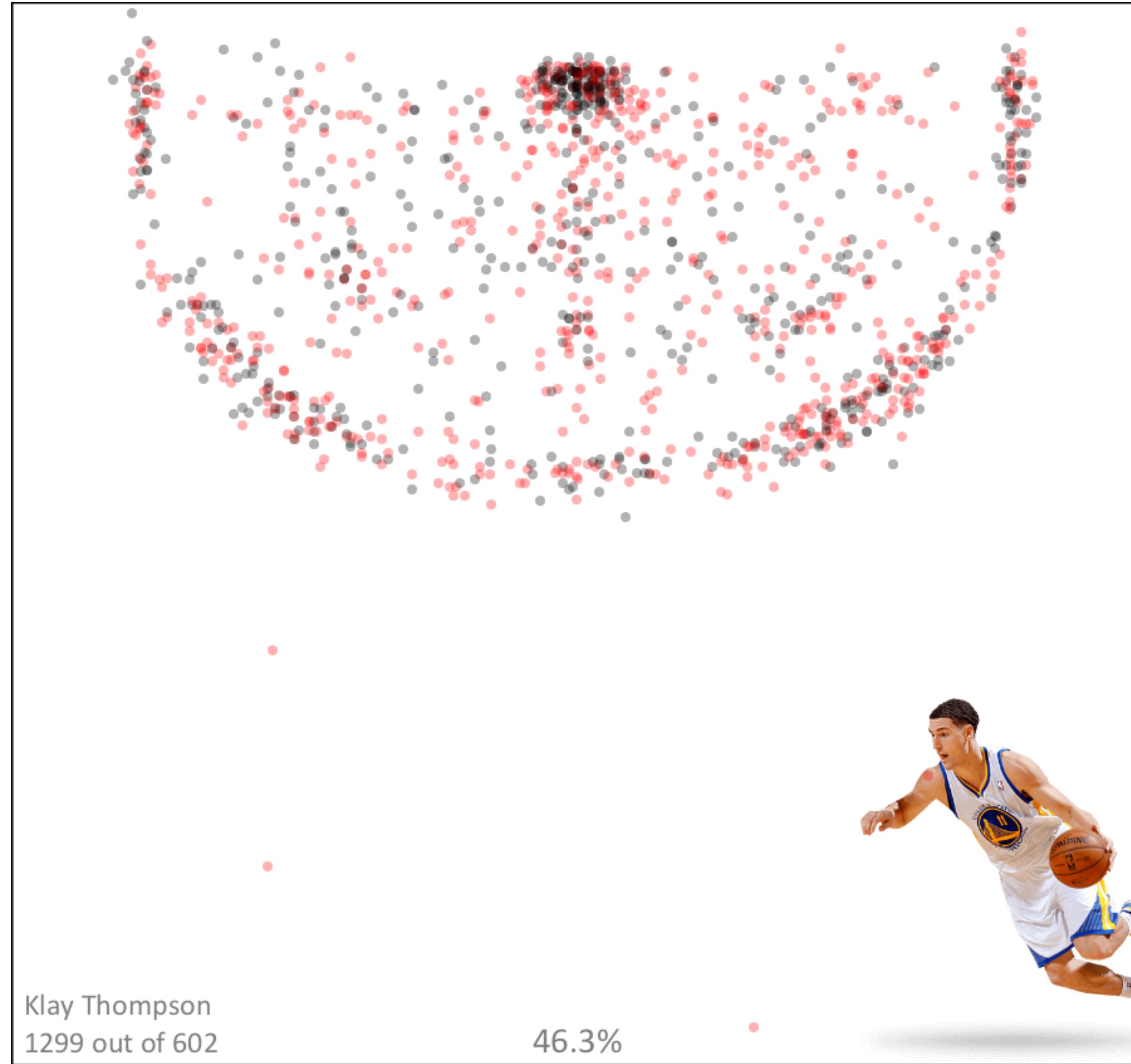


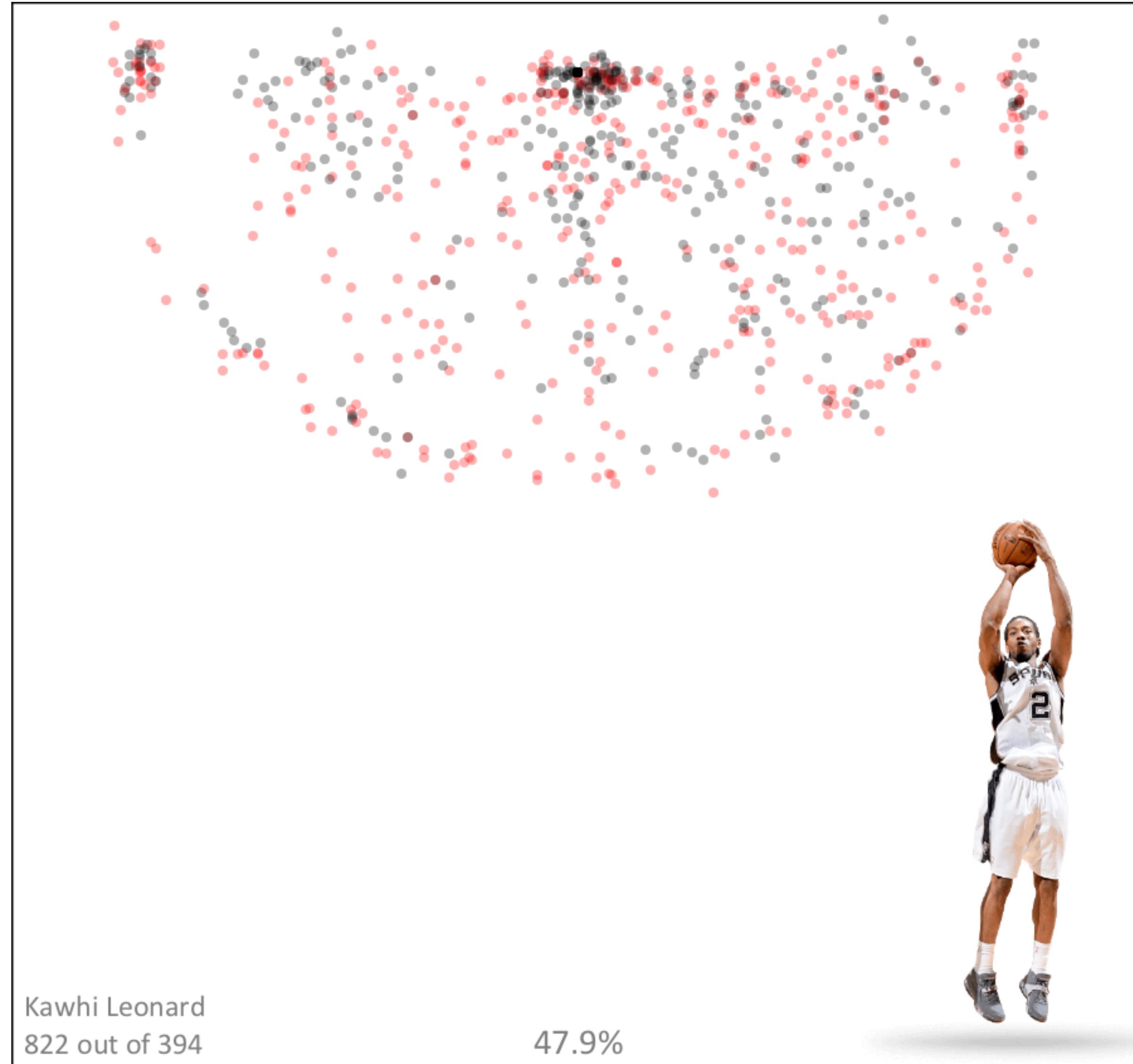


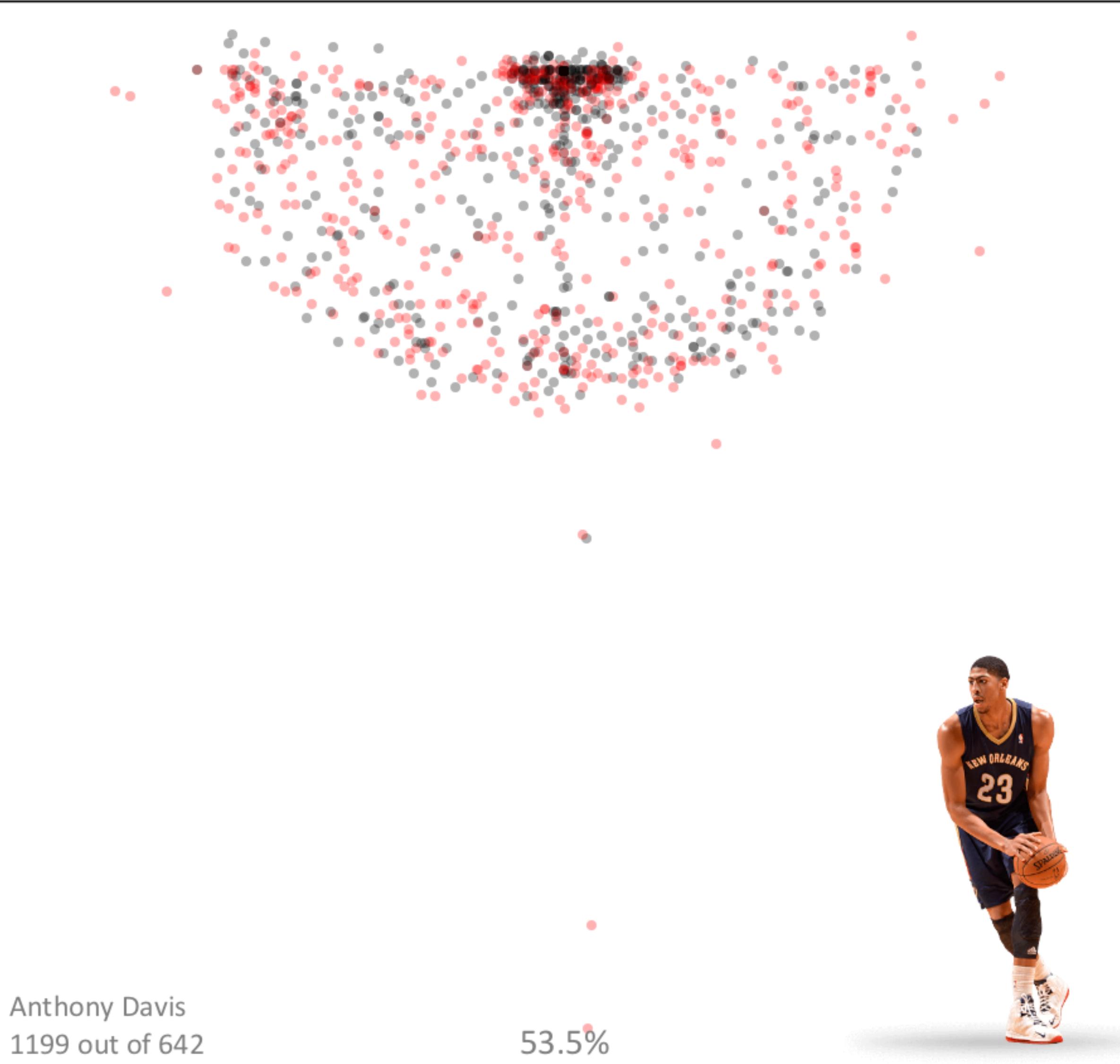


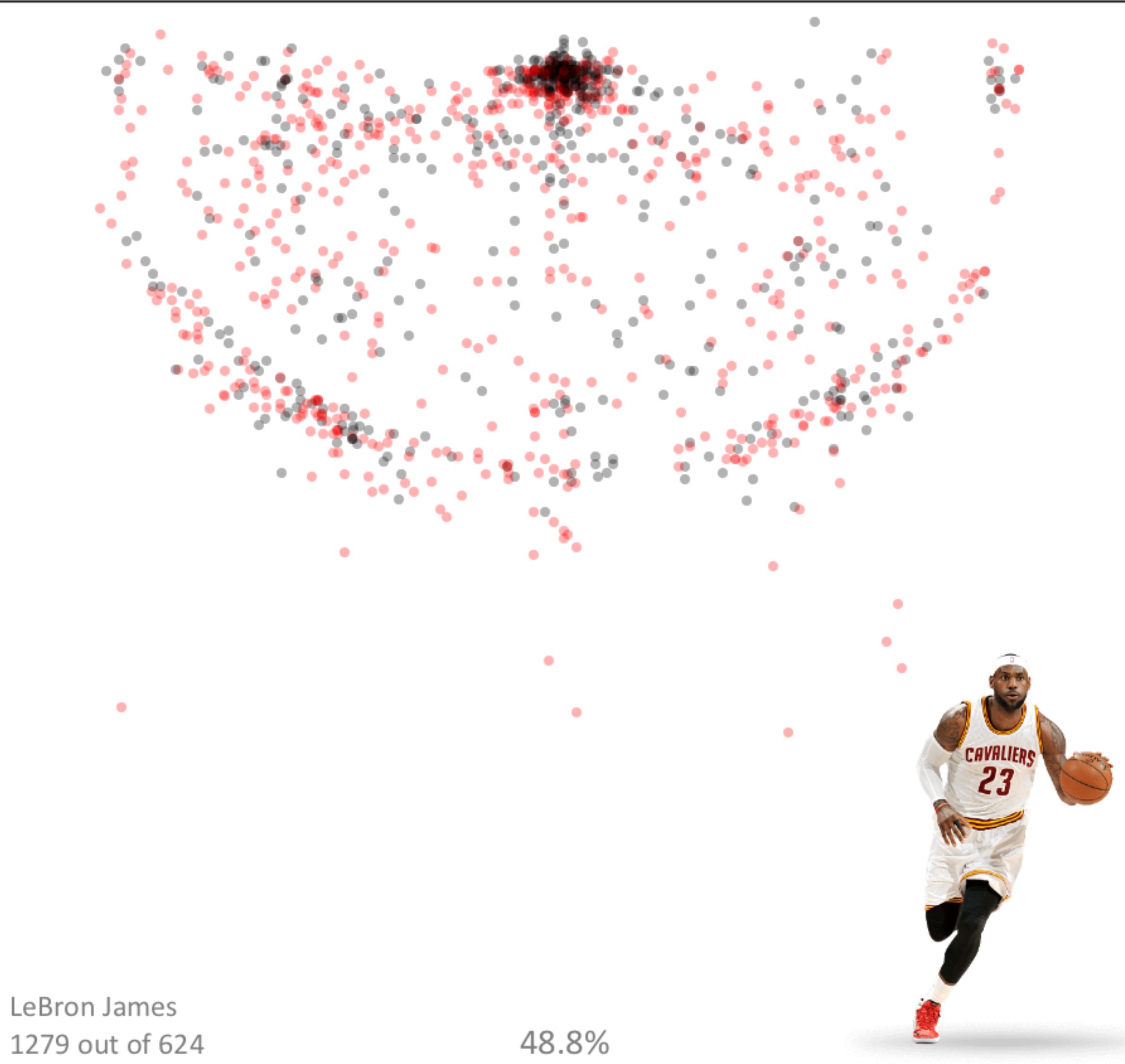


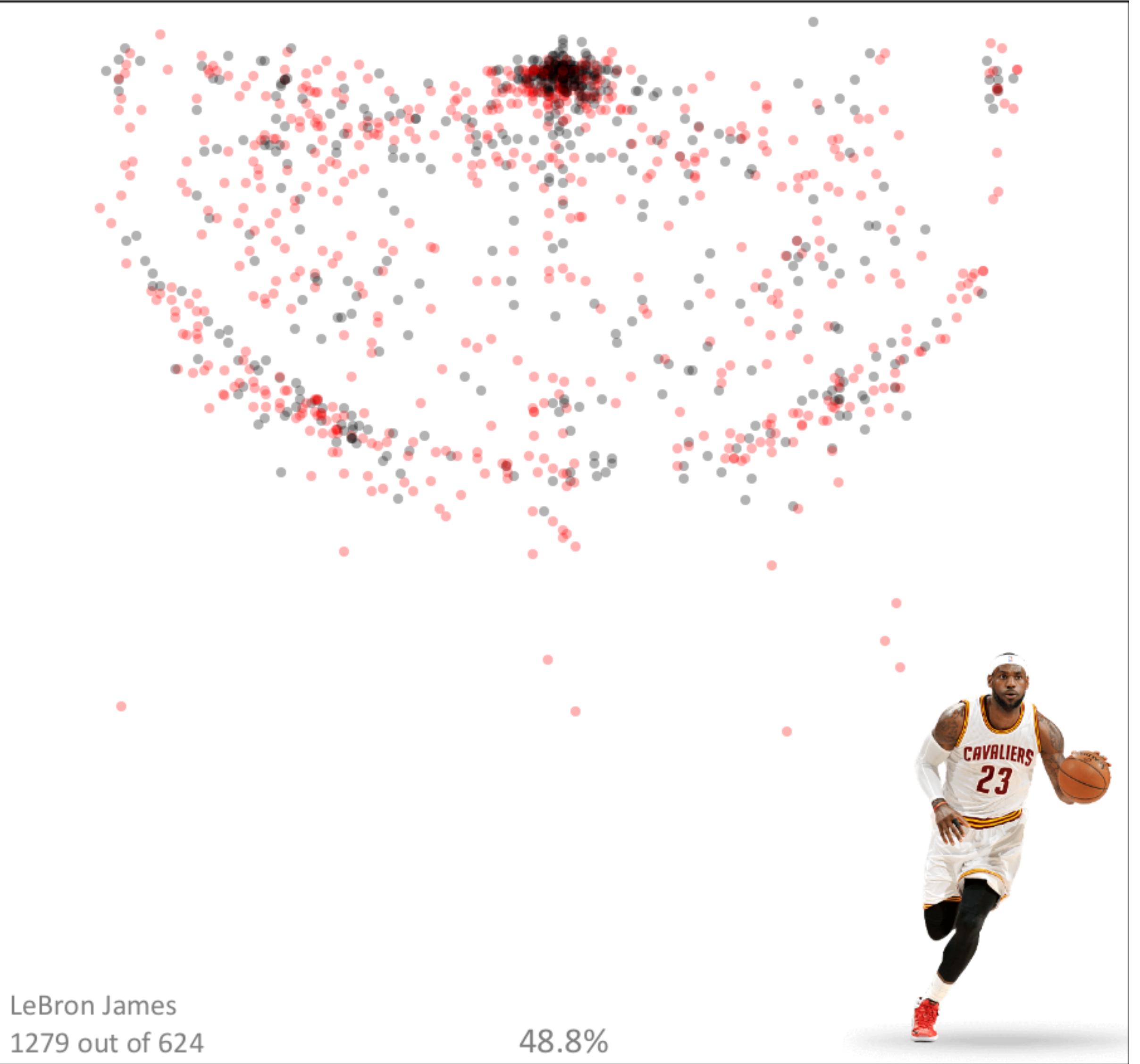








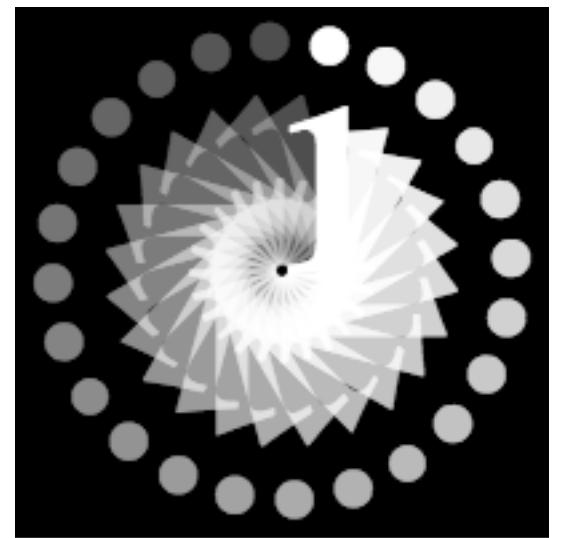




```
func main() {
```

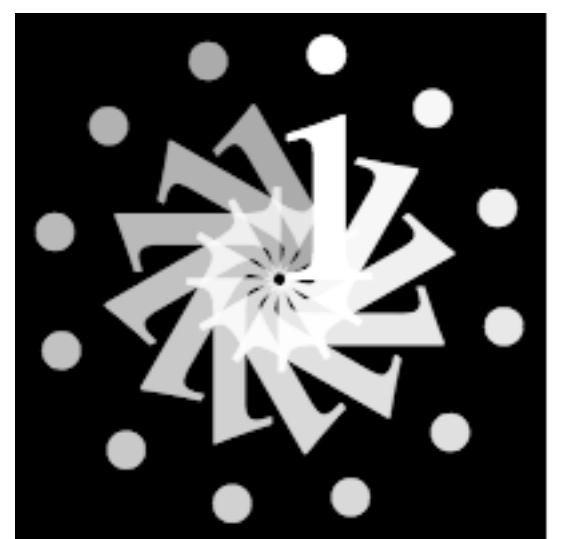
```
    width := 200  
    height := 200  
    a := 1.0  
    ai := 0.03  
    ti := 15.0
```

ti = 15



```
    canvas := svg.New(os.Stdout)  
    canvas.Start(width, height)  
    canvas.Rect(0, 0, width, height)  
    canvas.Gstyle("font-family:serif;font-size:100pt")
```

ti = 30

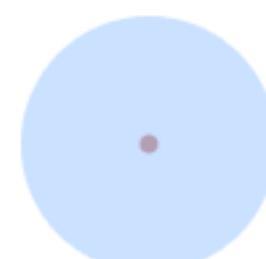


```
    for t := 0.0; t <= 360.0; t += ti {  
        canvas.TranslateRotate(width/2, height/2, t)  
        canvas.Text(0, 0, "i", canvas.RGBA(255, 255, 255, a))  
        canvas.Gend()  
        a -= ai  
    }  
    canvas.Gend()  
    canvas.End()
```

ti = 45



```
}
```



Circle



Ellipse



Rectangle



Rounded Rectangle



Line



Polyline



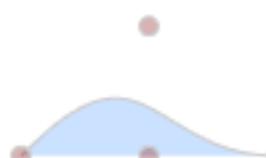
Polygon



Arc



Quadratic Bezier

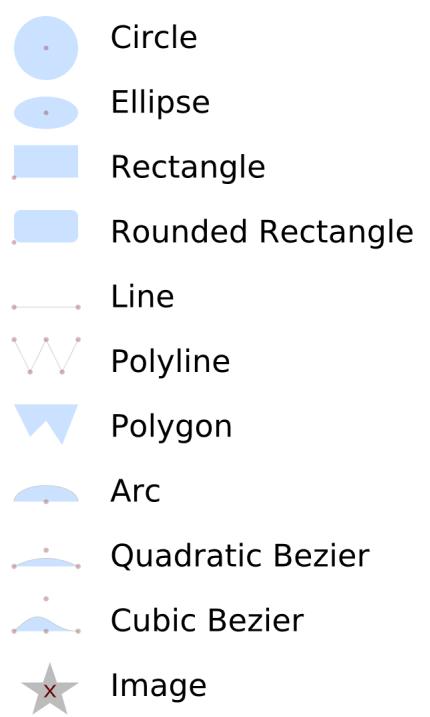


Cubic Bezier

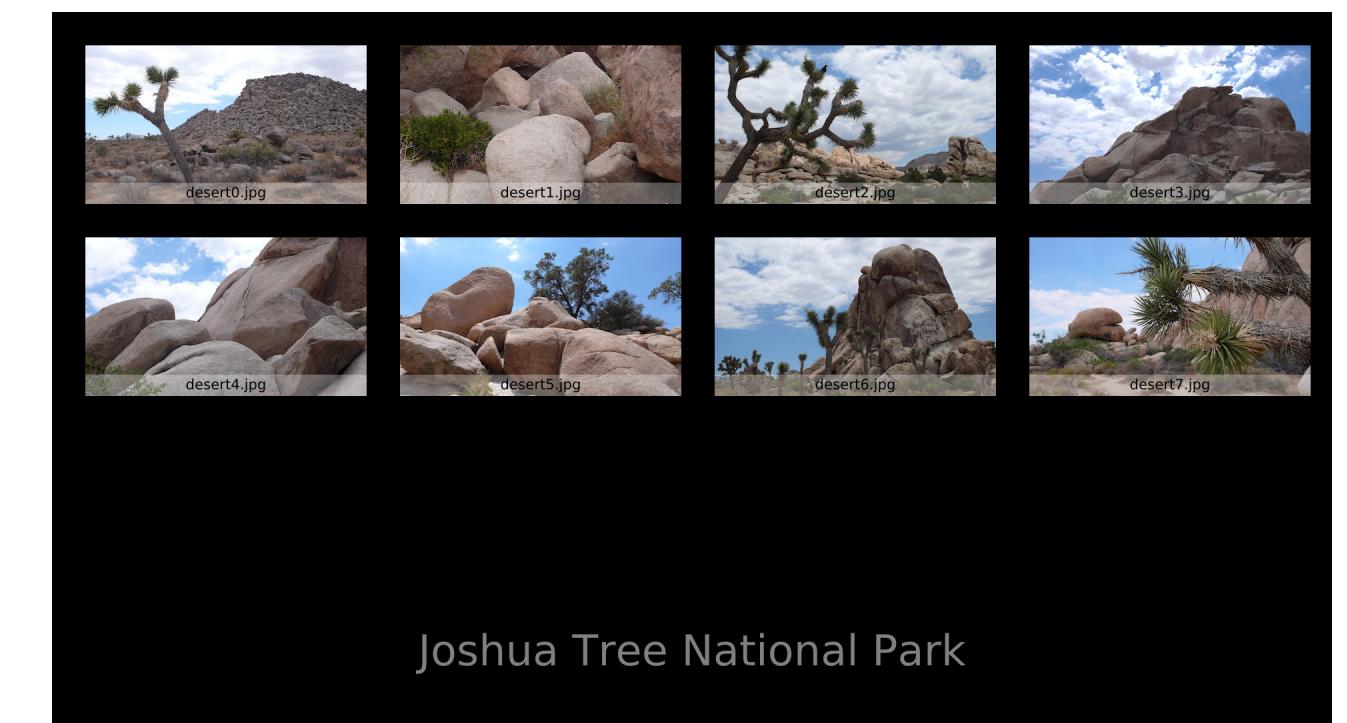
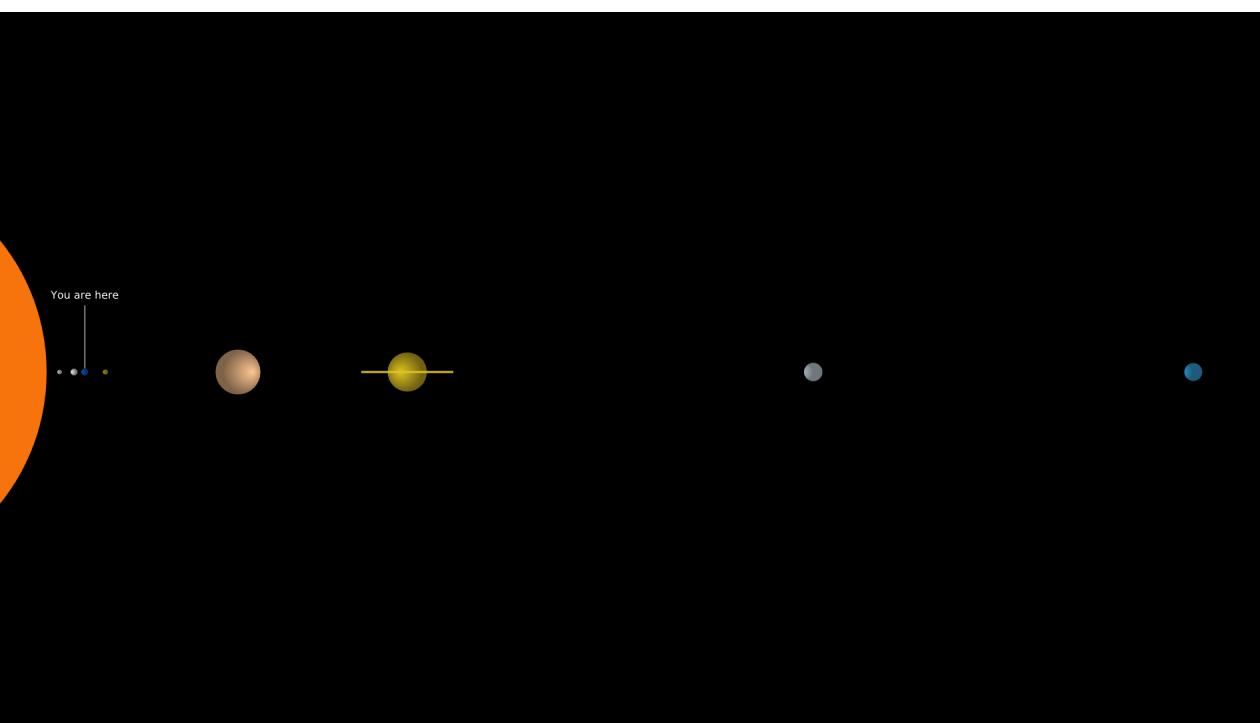
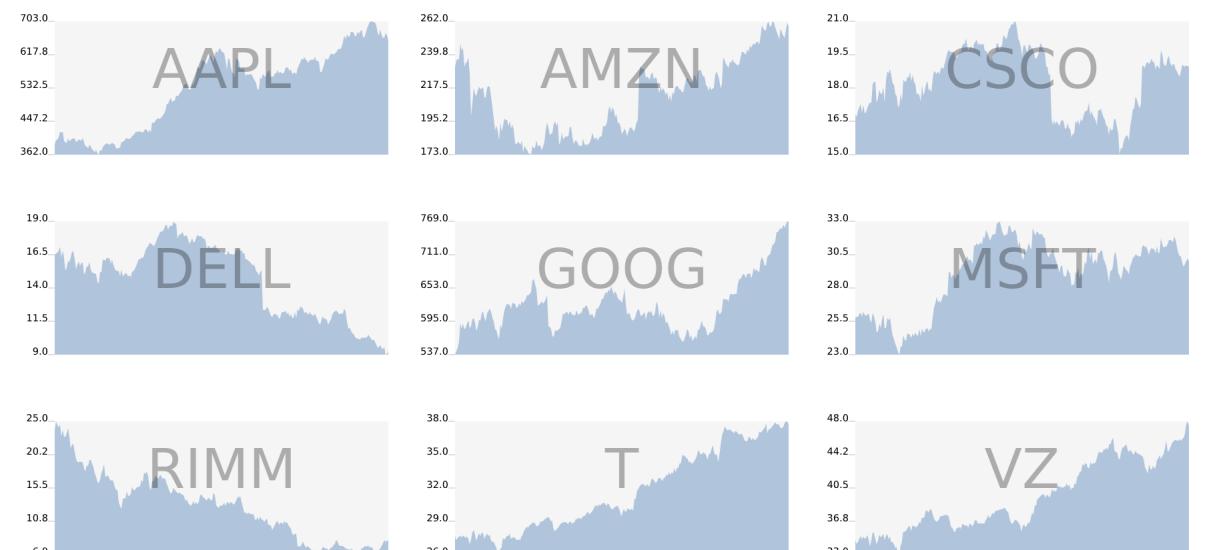
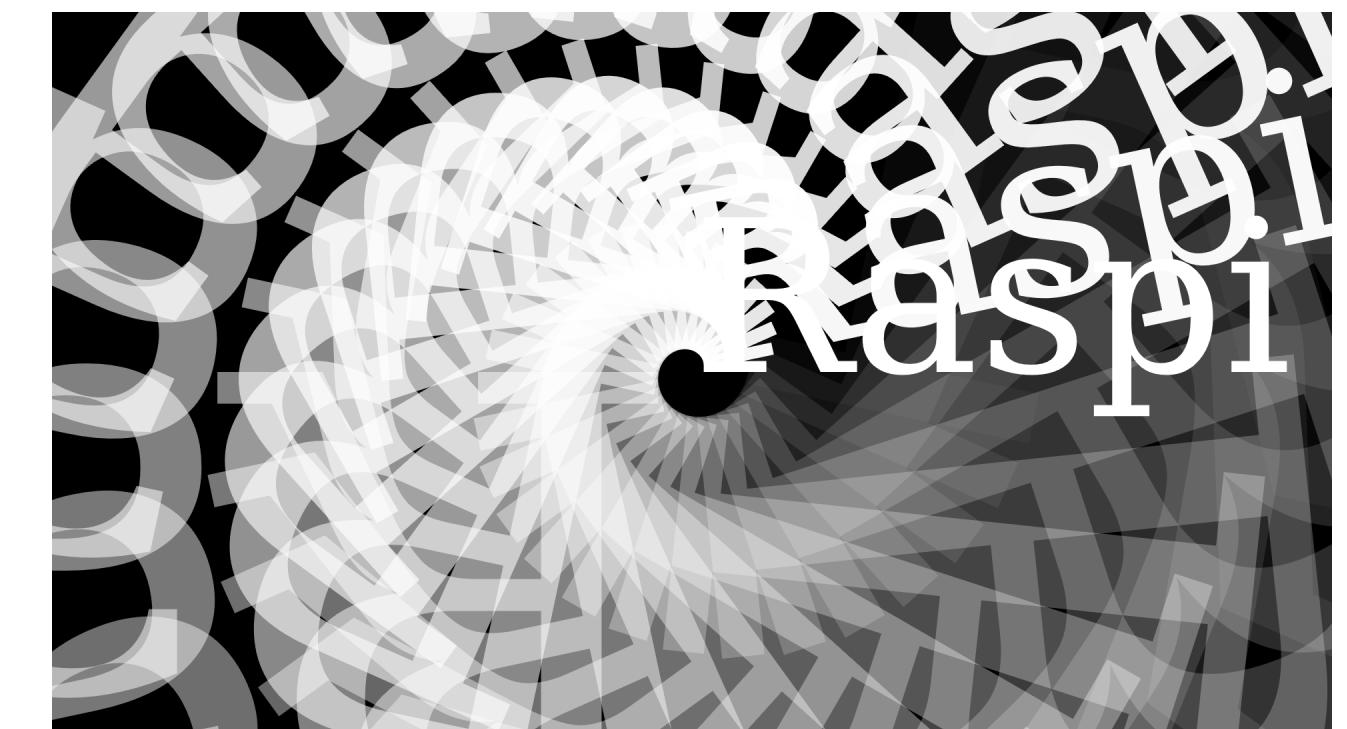
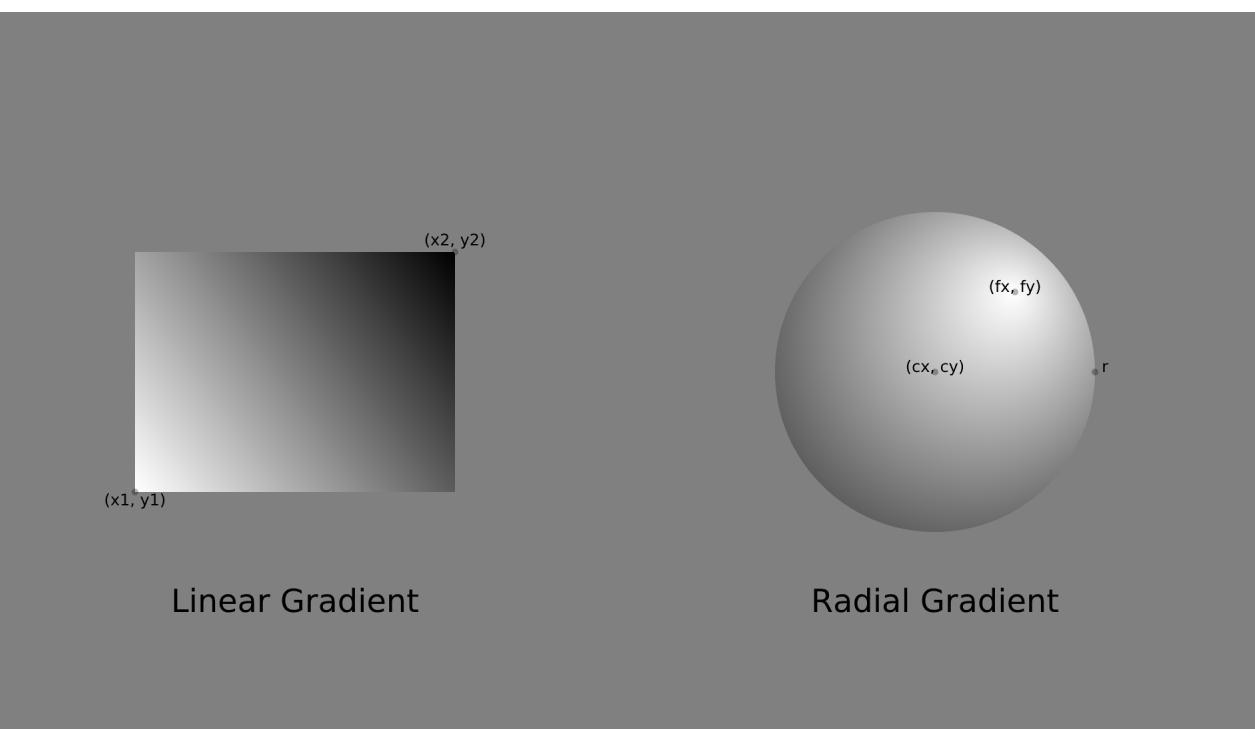


Image

# OpenVG on the Raspberry Pi



## OpenVG on the Raspberry Pi



## Why I use Go



Anthony Starks

@ajstarks  
ajstarks@gmail.com



```
package main
import "fmt"

func main() {
    fmt.Println("hello, world")
}
```

This is the traditional hello, world program.  
It's six lines long, and imports the fmt package,  
leading some to be surprised at the size of the resulting binary.

```

package main

import (
    "bufio"
    "github.com/ajstarks/openvg"
    "os"
)

func main() {
    width, height := openvg.Init()

    w2 := openvg.VGfloat(width / 2)
    h2 := openvg.VGfloat(height / 2)
    w := openvg.VGfloat(width)

    openvg.Start(width, height)                      // Start the picture
    openvg.BackgroundColor("black")                   // Black background
    openvg.FillRGB(44, 100, 232, 1)                 // Big blue marble
    openvg.Circle(w2, 0, w)                          // The "world"
    openvg.FillColor("white")                        // White text
    openvg.TextMid(w2, h2, "hello, world", "serif", width/10) // Greetings
    openvg.End()                                    // End the picture
    bufio.NewReader(os.Stdin).ReadBytes('\n') // Pause until [RETURN]
    openvg.Finish()                                // Graphics cleanup
}

```



# OpenVG Functions

**Circle** (x, y, r VGfloat)

**Ellipse** (x, y, w, h VGfloat)

**Rect** (x, y, w, h VGfloat)

**Roundrect** (x, y, w, h, rw, rh VGfloat)

**Line** (x1, y1, x2, y2 VGfloat)

**Polyline** (x, y [])VGfloat)

**Polygon** (x, y [])VGfloat)

**Arc** (x, y, w, h, sa, aext VGfloat)

**Qbezier** (sx, sy, cx, cy, ex, ey VGfloat)

**Cbezier** (sx, sy, cx, cy, px, py, ex, ey VGfloat)

**Image** (x, y VGfloat, w, h int, s string)

**Text** (x, y VGfloat, s, font string, size int)

**TextMid** (x, y VGfloat, s, font string, size int)

**TextEnd** (x, y VGfloat, s, font string, size int)

# Deck



a Go package for presentations

```
Start the deck      <deck>

Set the canvas size    <canvas width="1024" height="768" />

Begin a slide        <slide bg="white" fg="black">

Place an image       <image xp="70" yp="60" width="640" height="480" name="follow.png" sp="1" caption="Dreams"/>

Draw some text        <text xp="20" yp="80" sp="4" link="http://goo.gl/Wm05Ex">Deck elements</text>

Make a bullet list    <list xp="20" yp="70" sp="3" type="bullet">

                        <li>text, list, image</li>
                        <li>line, rect, ellipse</li>
                        <li>arc, curve, polygon</li>

End the list          </list>

Draw a line            <line xp1="20" yp1="10" xp2="30" yp2="10"/>

Draw a rectangle       <rect xp="35" yp="10" wp="4" hr="75" color="rgb(127,0,0)"/>

Draw an ellipse        <ellipse xp="45" yp="10" wp="4" hr="75" color="rgb(0,127,0)"/>

Draw an arc             <arc xp="55" yp="10" wp="4" hp="3" a1="0" a2="180" color="rgb(0,0,127)"/>

Draw a quadratic bezier <curve xp1="60" yp1="10" xp2="75" yp2="20" xp3="70" yp3="10" />

Draw a polygon          <polygon xc=75 75 80" yc="8 12 10" color="rgb(0,0,127)"/>

End the slide          </slide>

End of the deck        </deck>
```

## Anatomy of a Deck

# Deck elements

- text, list, image
- line, rect, ellipse
- arc, curve, polygon



Dreams



**Percent Grid**

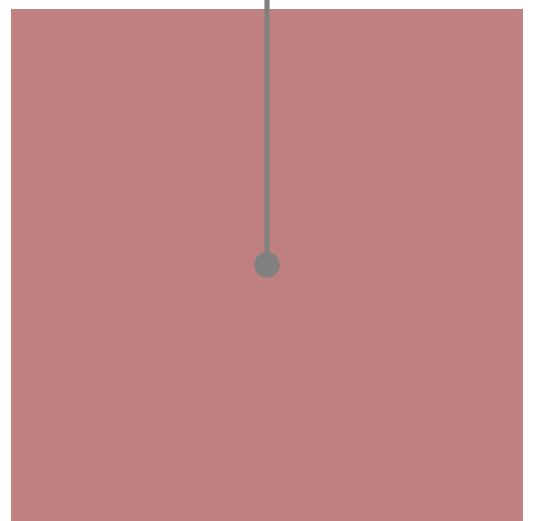
10%, 50%

Hello

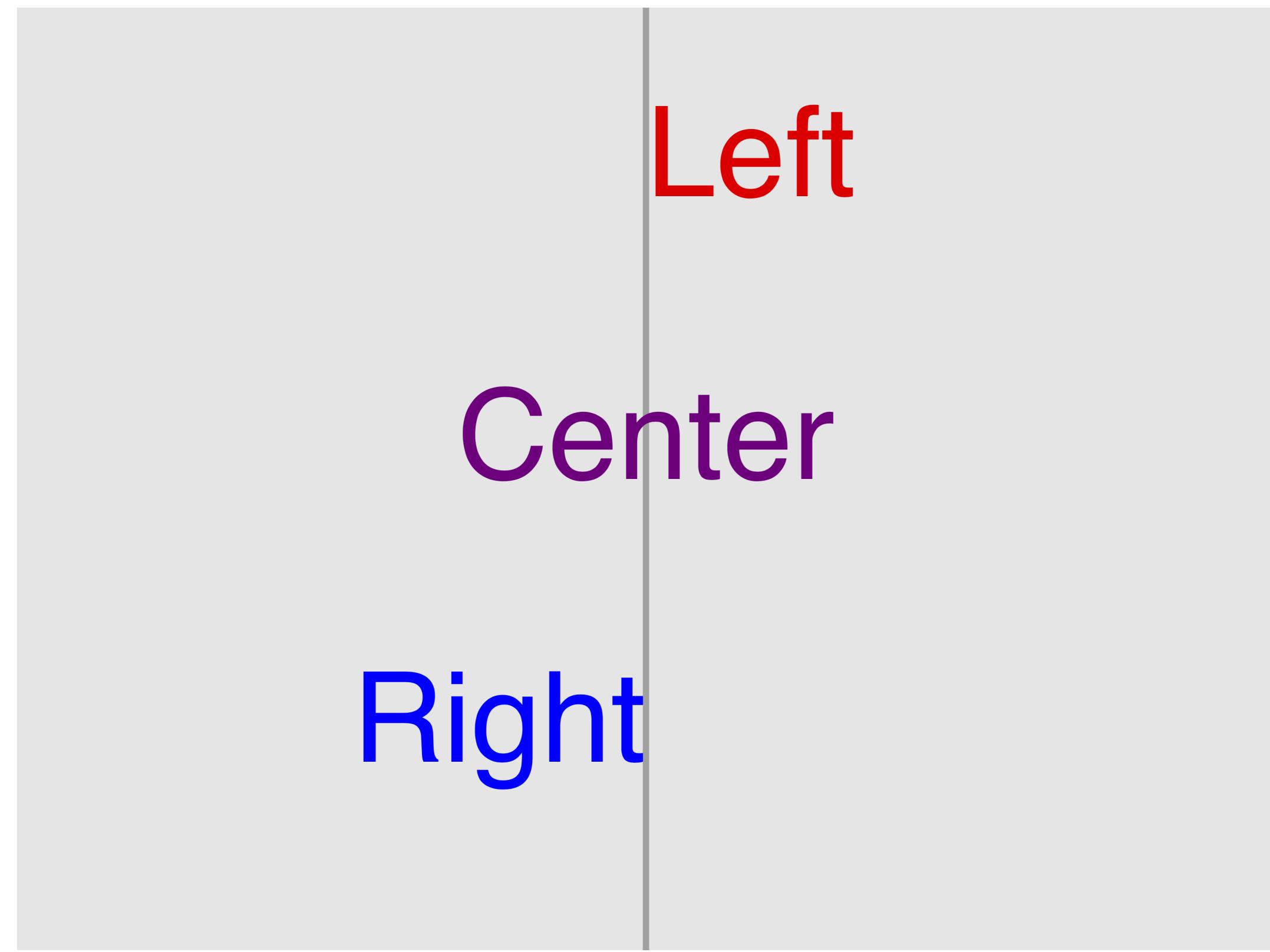
50%, 50%



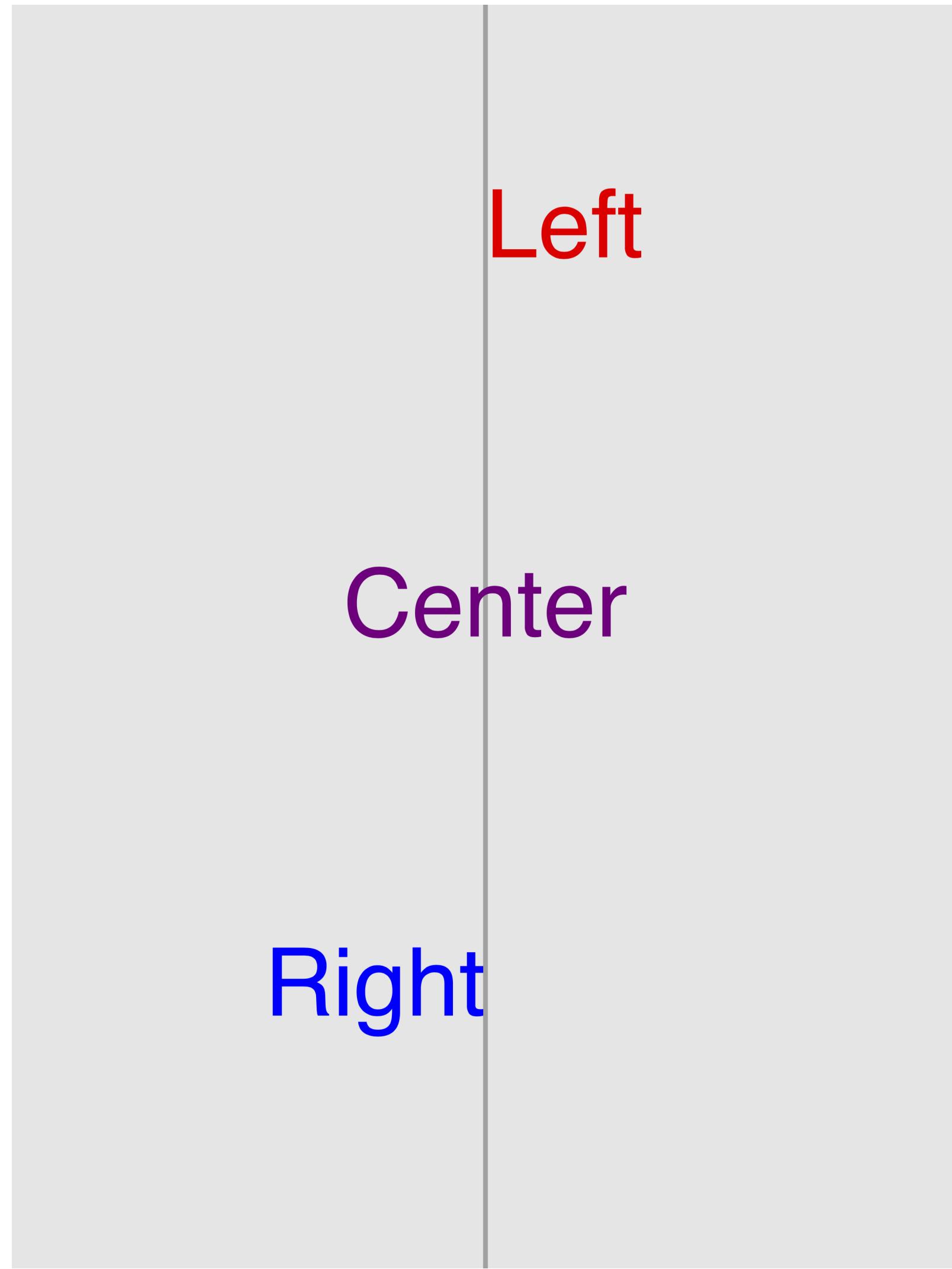
90%, 50%



Percentage-based layout



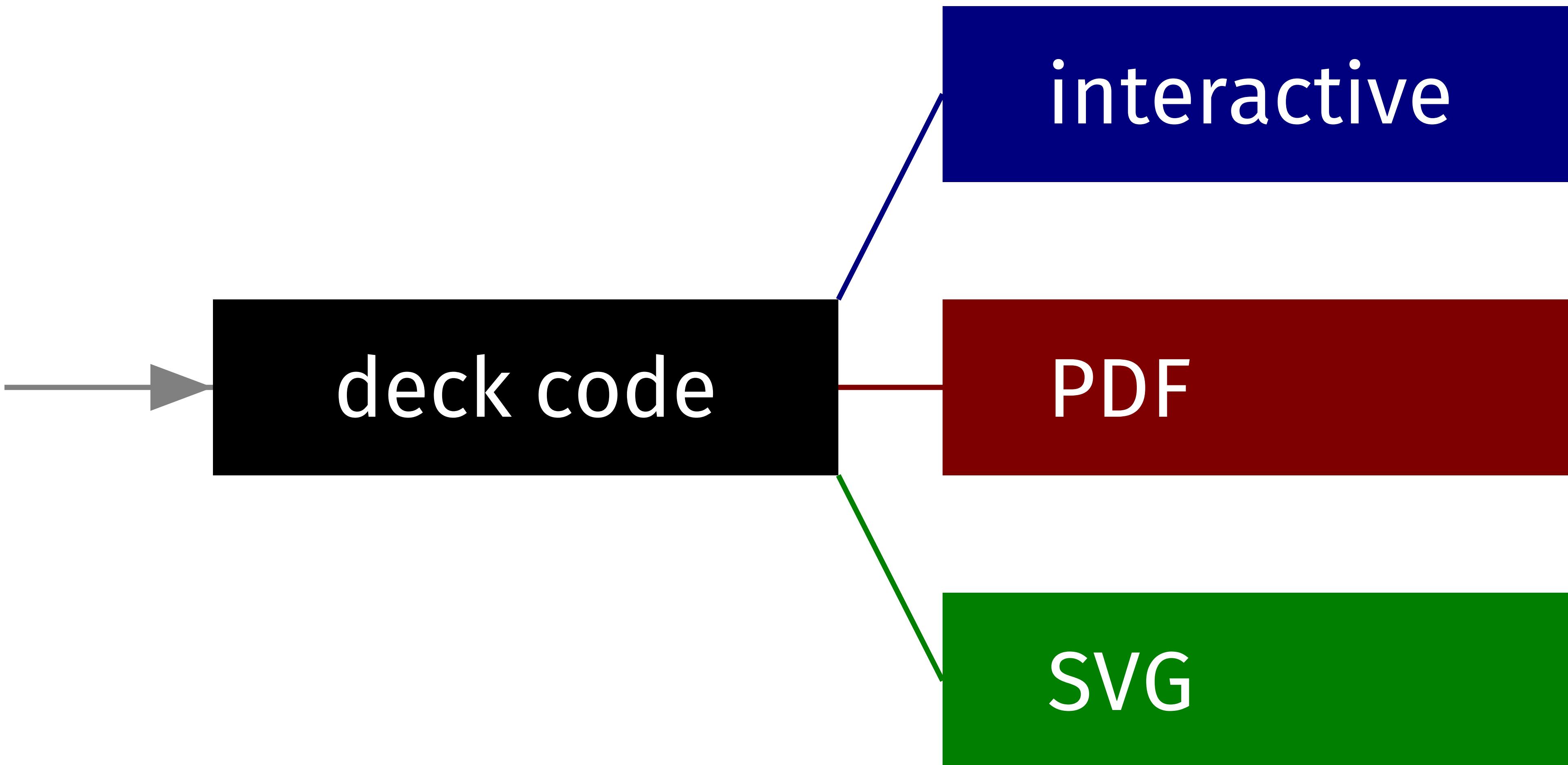
Landscape



Portrait

# Scaling the canvas

Process



# deck/generate text and list functions

<b>Text</b>	(x, y float64, s, font string, size float64, color string, opacity ...float64)
<b>TextBlock</b>	(x, y float64, s, font string, size, margin float64, color string, opacity ...float64)
<b>TextMid</b>	(x, y float64, s, font string, size float64, color string, opacity ...float64)
<b>TextEnd</b>	(x, y float64, s, font string, size float64, color string, opacity ...float64)
<b>Code</b>	(x, y float64, s string, size, margin float64, color string, opacity ...float64)
<b>List</b>	(x, y, size float64, items []string, ltype, font, color string)

# deck/generate graphic functions

Image	(x, y float64, w, h int, name string)
Arc	(x, y, w, h, size, a1, a2 float64, color string, opacity ...float64)
Circle	(x, y, w float64, color string, opacity ...float64)
Ellipse	(x, y, w, h float64, color string, opacity ...float64)
Square	(x, y, w float64, color string, opacity ...float64)
Rect	(x, y, w, h float64, color string, opacity ...float64)
Curve	(x1, y1, x2, y2, x3, y3, size float64, color string, opacity ...float64)
Line	(x1, y1, x2, y2, size float64, color string, opacity ...float64)
Polygon	(x, y []float64, color string, opacity ...float64)

```

package main

import (
    "github.com/ajstarks/deck/generate"
    "os"
)

func main() {
    deck := generate.NewSlides(os.Stdout, 1600, 900) // 16x9 deck to standard output
    deck.StartDeck() // start the deck

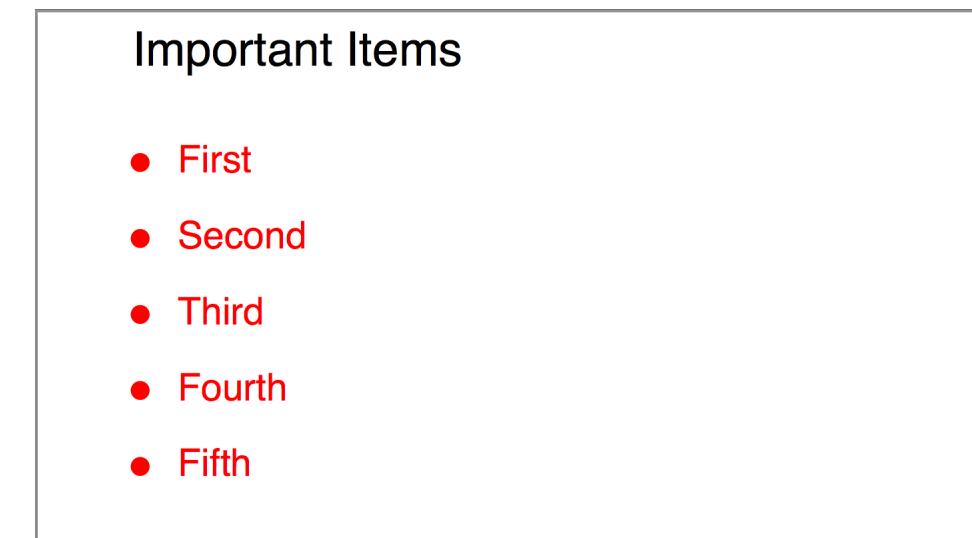
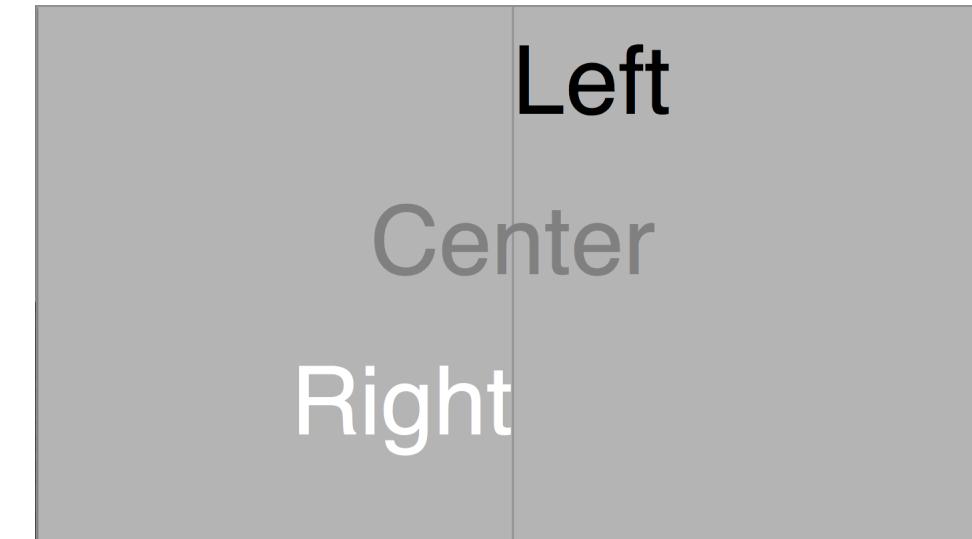
    // Text alignment
    deck.StartSlide("rgb(180,180,180)")
    deck.Text(50, 80, "Left", "sans", 10, "black")
    deck.TextMid(50, 50, "Center", "sans", 10, "gray")
    deck.TextEnd(50, 20, "Right", "sans", 10, "white")
    deck.Line(50, 100, 50, 0, 0.2, "black", 20)
    deck.EndSlide()

    // List
    items := []string{"First", "Second",
                      "Third", "Fourth", "Fifth"}
    deck.StartSlide()
    deck.Text(10, 90, "Important Items", "sans", 5, "")
    deck.List(10, 70, 4, items, "bullet", "sans", "red")
    deck.EndSlide()

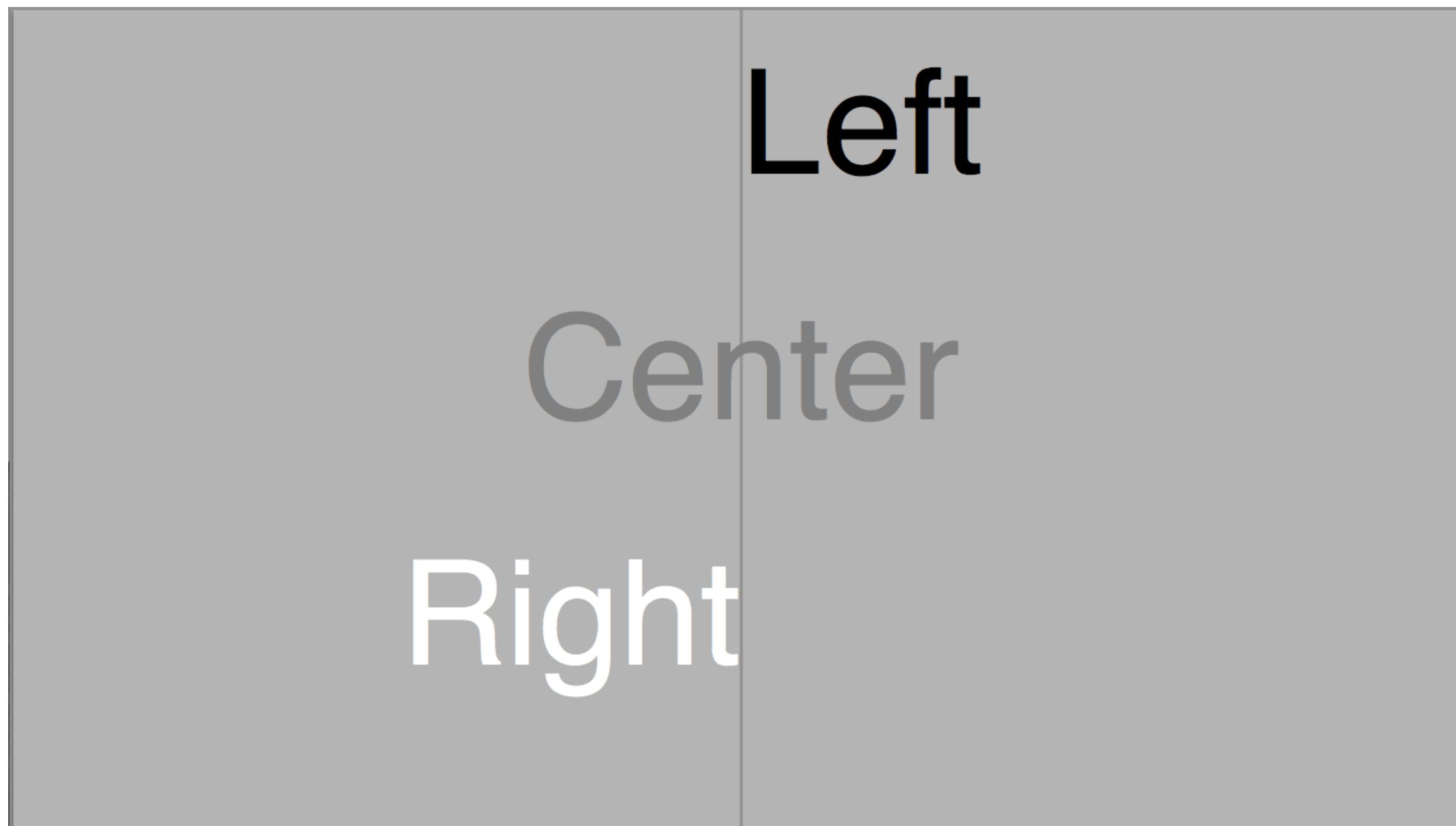
    // Picture with text annotation
    quote := "Yours is some tepid, off-brand, generic 'cola'. " +
        "What I'm making is \"Classic Coke\""
    person := "Heisenberg"
    deck.StartSlide("black", "white")
    deck.Image(50, 50, 1440, 900, "classic-coke.png")
    deck.TextBlock(10, 80, quote, "sans", 2.5, 30, "")
    deck.Text(65, 15, person, "sans", 1.2, "")
    deck.EndSlide()

    deck.EndDeck() // end the deck
}

```



```
// Text alignment
deck.StartSlide("rgb(180,180,180)")
deck.Text(50, 80, "Left", "sans", 10, "black")
deck.TextMid(50, 50, "Center", "sans", 10, "gray")
deck.TextEnd(50, 20, "Right", "sans", 10, "white")
deck.Line(50, 100, 50, 0, 0.2, "black", 20)
deck.EndSlide()
```



```
// List
items := []string{"First", "Second", "Third",
                    "Fourth", "Fifth"}
deck.StartSlide()
deck.Text(10, 90, "Important Items", "sans", 5, "")
deck.List(10, 70, 4, items, "bullet", "sans", "red")
deck.EndSlide()
```

## Important Items

- First
- Second
- Third
- Fourth
- Fifth

```
// Picture with text annotation
quote := "Yours is some tepid, off-brand, generic 'cola'. " +
        "What I'm making is \"Classic Coke\""
person := "Heisenberg"
deck.StartSlide("black", "white")
deck.Image(50, 50, 1440, 900, "classic-coke.png")
deck.TextBlock(10, 80, quote, "sans", 2.5, 30, "")
deck.Text(65, 15, person, "sans", 1.2, "")
deck.EndSlide()
```



# A View of User Experience: Designing for People

Anthony Starks / ajstarks@gmail.com / @ajstarks

Design



What works good is better than what looks good, because what works good lasts.

Ray Eames

## # Designing for People

title A View of User Experience: Designing for People Anthony Starks / ajstarks@gmail.com / @ajstarks

section Design gray white

caption eames.png Ray Eames What works good is better than what looks good, because what works good lasts.

capgen slides.txt | pdfdeck ... > slides.pdf

# Design Examples

Top

Left

Right

Bottom

10%

30%

70%

10%

# Header (top 10%)

Summary  
(30%)

Detail  
(70%)

Footer (bottom 10%)

# My Story

## Section

One

Two

Three

Four

Five

Six

# Document Links

Add web and mailto links with the link attribute of the text element.

Once rendered as a PDF, clicking on the link opens the default browser or email client.

The image contains two screenshots. The top screenshot shows a PDF viewer window titled "deck-fira-4x3.pdf (page 53 of 57)". It displays a quote in white text on a dark blue background: "Python and Ruby programmers come to Go because they don't have to surrender much expressiveness, but gain performance and get to play with concurrency." Below the quote, it says "Less is exponentially more" and "Rob Pike". The bottom screenshot shows a web browser window titled "command center: Less is e...". The URL in the address bar is "commandcenter.blogspot.com/2012/06/less-is-exp...". The page content is identical to the quote in the PDF, along with some additional text: "What you're given is a set of powerful but easy to understand, easy to use building blocks from which you can assemble—compose—a solution to your problem. It might not end up quite as fast or as sophisticated or as ideologically motivated as the solution you'd write in some of those other languages, but it'll almost certainly be easier to write, easier to read, easier to understand, easier to maintain, and maybe safer." Below this, there is another quote: "To put it another way, oversimplifying of course: Python and Ruby programmers come to Go because they don't have to surrender much expressiveness, but gain performance and get to play with concurrency." At the bottom, there is a final note: "C++ programmers *don't* come to Go because they have fought hard to gain exquisite control of their programming domain, and don't want to surrender any of it. To them, software isn't just about getting the job done, it's about doing it a certain way." The issue is then summarized as "The issue, then, is that Go's success would contradict their world view."

Python and Ruby programmers come to Go because they don't have to surrender much expressiveness, but gain performance and get to play with concurrency.

Less is exponentially more

Rob Pike

What you're given is a set of powerful but easy to understand, easy to use building blocks from which you can assemble—compose—a solution to your problem. It might not end up quite as fast or as sophisticated or as ideologically motivated as the solution you'd write in some of those other languages, but it'll almost certainly be easier to write, easier to read, easier to understand, easier to maintain, and maybe safer.

To put it another way, oversimplifying of course:

Python and Ruby programmers come to Go because they don't have to surrender much expressiveness, but gain performance and get to play with concurrency.

C++ programmers *don't* come to Go because they have fought hard to gain exquisite control of their programming domain, and don't want to surrender any of it. To them, software isn't just about getting the job done, it's about doing it a certain way.

The issue, then, is that Go's success would contradict their world view.

BOS



SFO

Virgin America 351

Gate B38

8:35am

On Time

JFK



IND

US Airways 1207

Gate C31C

5:35pm

Delayed

AAPL	119.08	3.58	(3.10%)
AMZN	599.03	35.12	(6.23%)
FB	102.19	2.52	(2.53%)
GOOG	702.00	50.21	(7.70%)
MSFT	52.87	4.84	(10.08%)

# go

---

<b>build</b>	compile packages and dependencies
<b>clean</b>	remove object files
<b>doc</b>	show documentation for package or symbol
<b>env</b>	print Go environment information
<b>fix</b>	run go tool fix on packages
<b>fmt</b>	run gofmt on package sources
<b>generate</b>	generate Go files by processing source
<b>get</b>	download and install packages and dependencies
<b>install</b>	compile and install packages and dependencies
<b>list</b>	list packages
<b>run</b>	compile and run Go program
<b>test</b>	test packages
<b>tool</b>	run specified go tool
<b>version</b>	print Go version
<b>vet</b>	run go tool vet on packages

This is not a index card

Rich

Can't buy me love

Bliss

Worse

Better

Misery

We have each other

Poor

So, the next time you're  
about to make a subclass,  
think hard and ask yourself  
**what would Go do**

Andrew Mackenzie-Ross



**FOR, LO,**

the winter is past,  
the rain is over and gone;  
The flowers appear on the earth;  
the time for the singing of birds is come,  
and the voice of the turtle is heard in our land.

Song of Solomon 2:11-12

# Good Design

is innovative

makes a product useful

is aesthetic

makes a product understandable

is unobtrusive

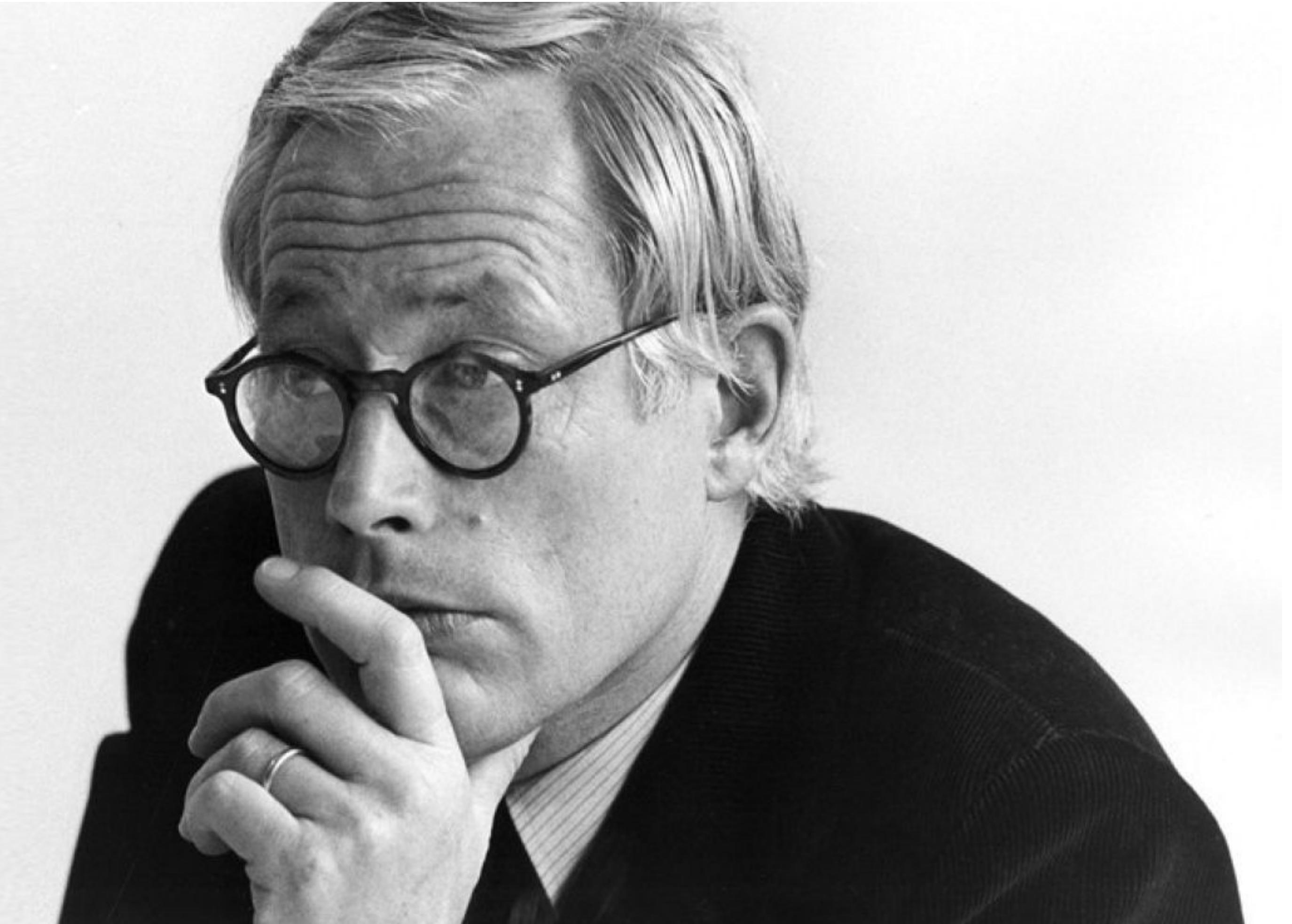
is honest

is long-lasting

is thorough down to the last detail

is environmentally-friendly

is as little design as possible



# What is it about Go?

f m t

func

Writer  
Reader

net/http

encoding/xml

encoding/json

cgo

# The Community

Re: [go-nuts] Visualizing Random Number Generators:  
/dev/urandom vs. Go rand package



Inbox x



Russ Cox rsc@golang.org via google.com

3/5/10



to me ▾

are you going to share the library or just  
tease us with pictures? ;-)

Thompson wanted to create a comfortable computing environment constructed according to his own design, using whatever means were available.

Dennis M. Ritchie, “The Development of the C Language”

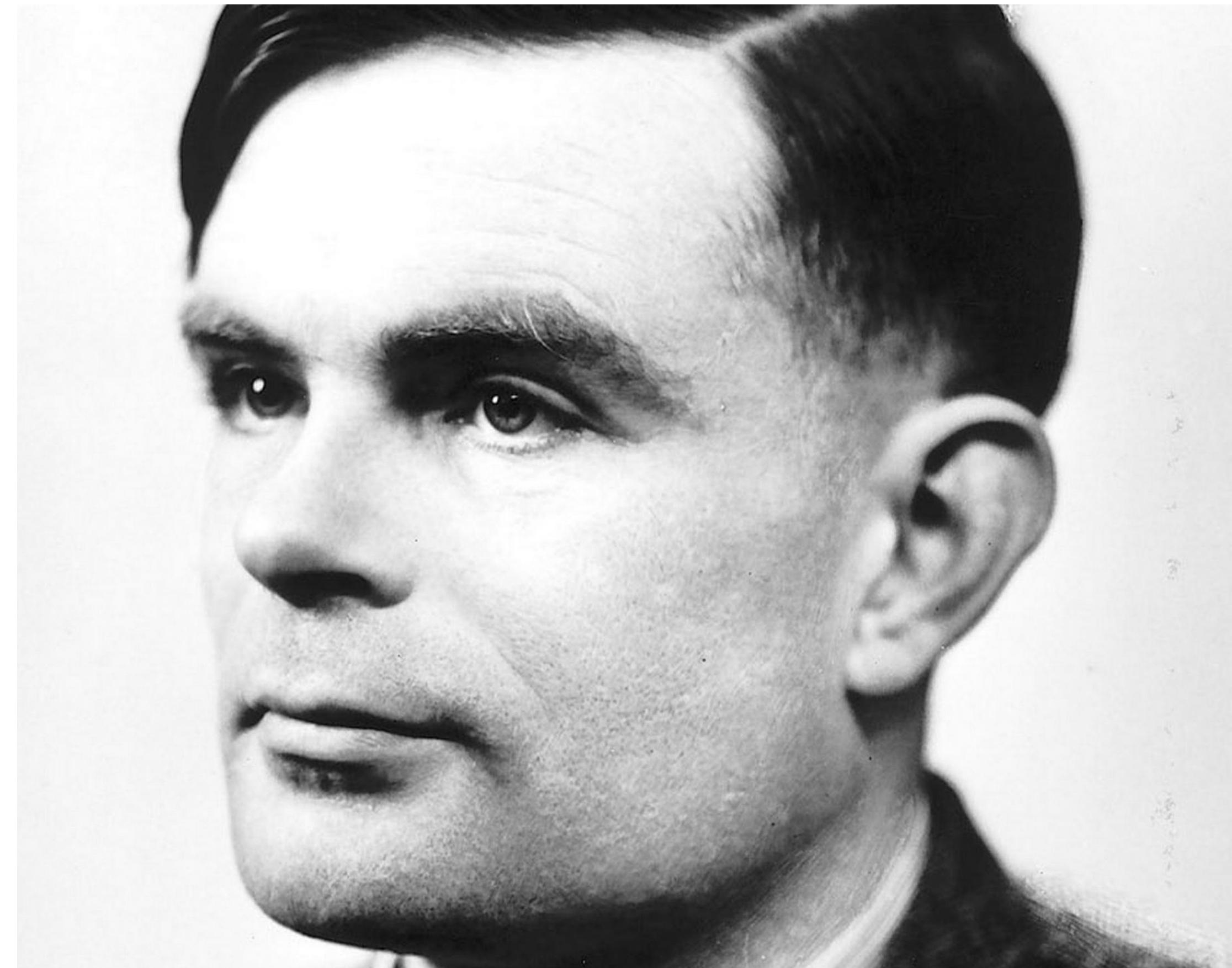
Go is not the product of a Whiggish development process. We were just trying to get something that worked for us.

Rob Pike, “Origin of Go’s interface design”, golang-nuts

# Making Tools

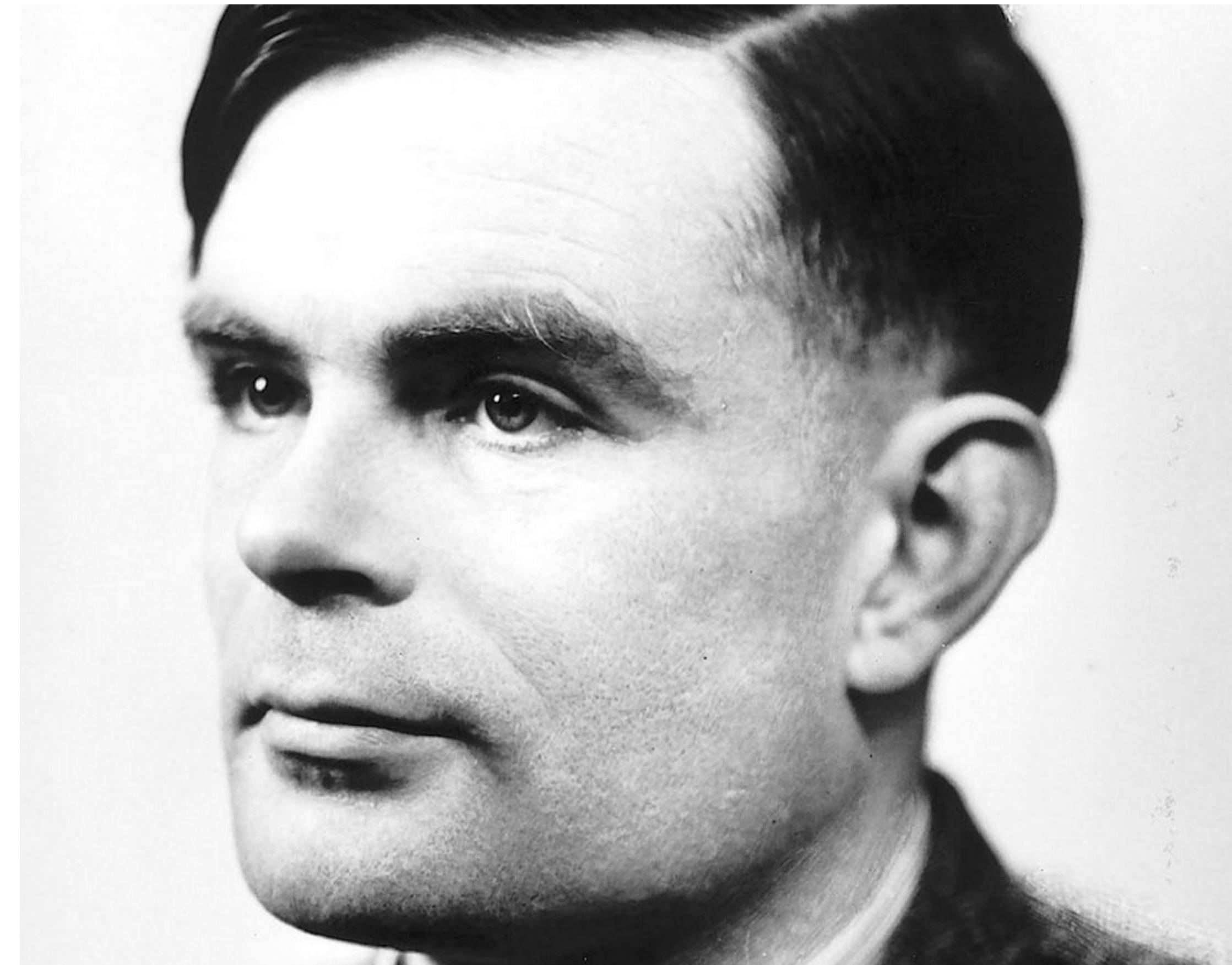
# Fun

Reducing the distance from  
the idea to the picture





Picasso



Turing