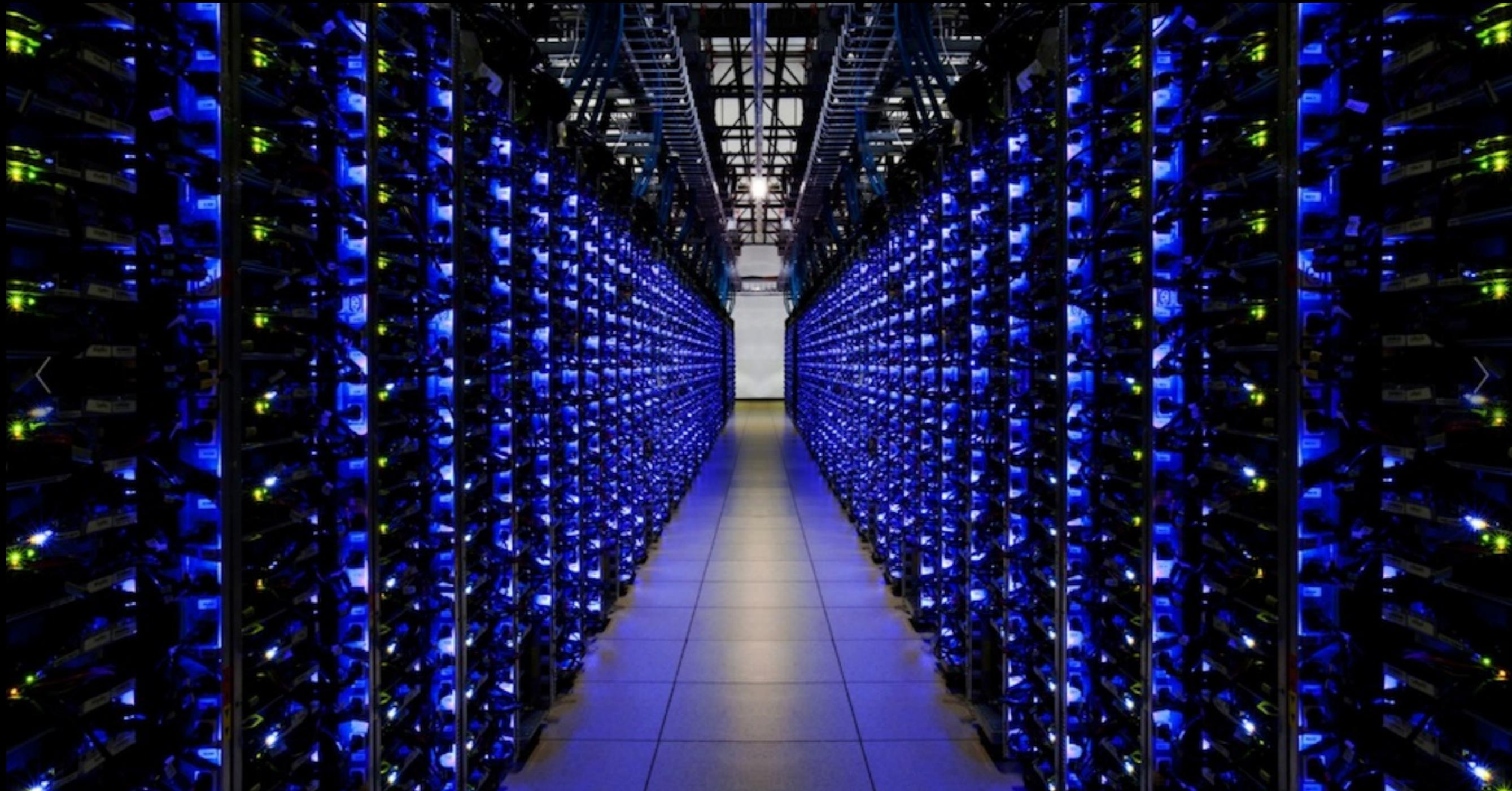


The other side of Go

Programming Pictures

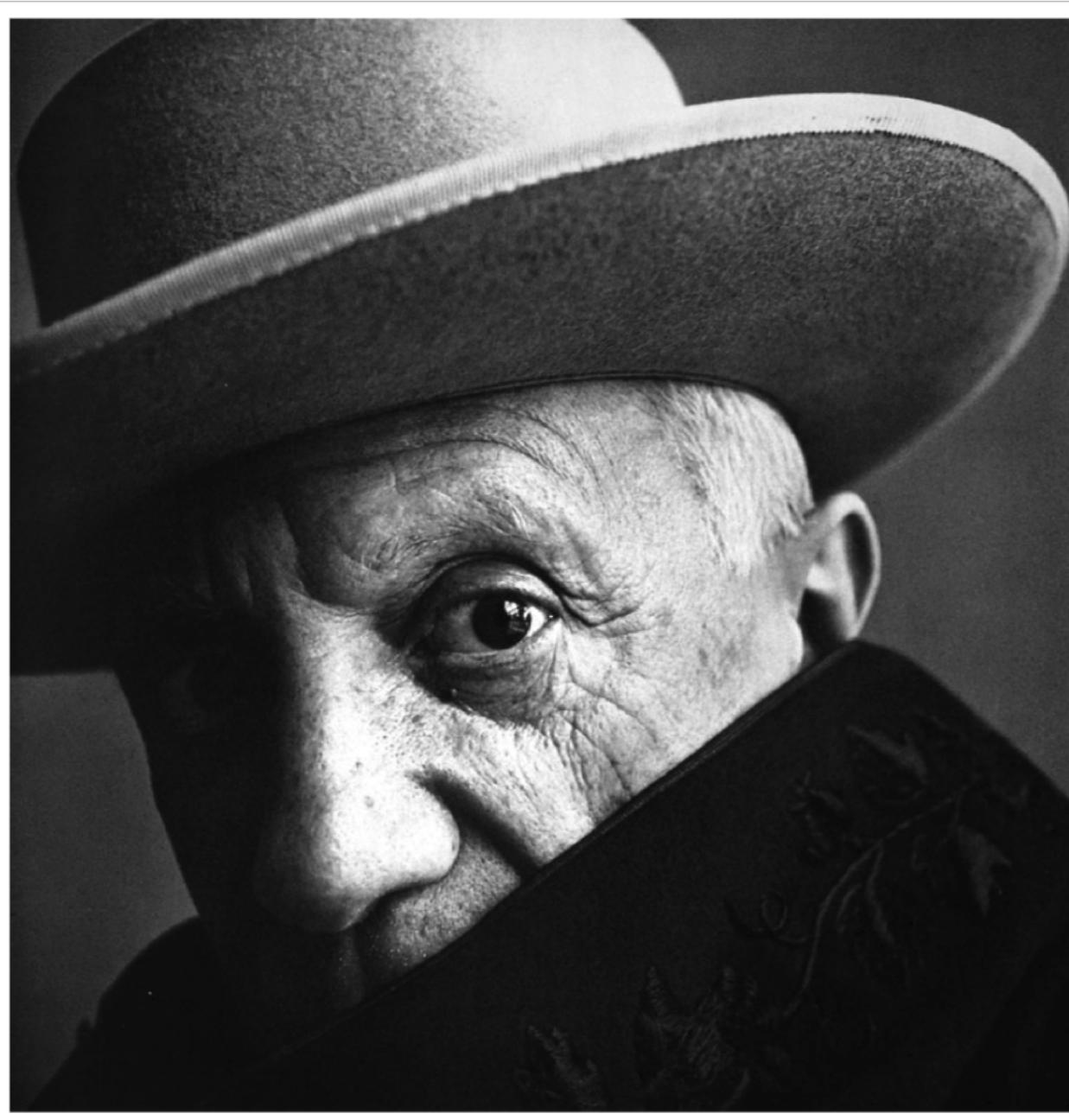
Anthony Starks

Go is great in the back end



A photograph of a dramatic sky. The upper portion is filled with large, white, billowing cumulus clouds against a dark blue background. In the lower portion, there are darker, more textured clouds, possibly cumulonimbus or stratus. The overall mood is serene and majestic.

But sometimes it's about the picture



Picasso



Turing

API Design

Client Program Design

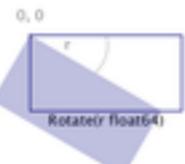
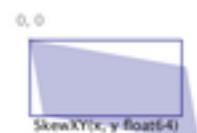
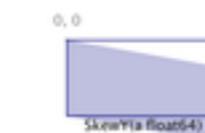
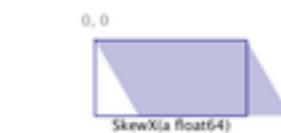
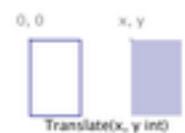
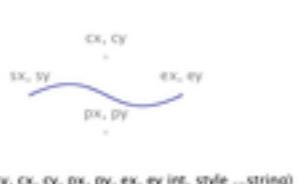
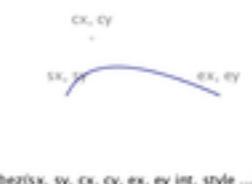
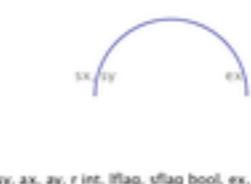
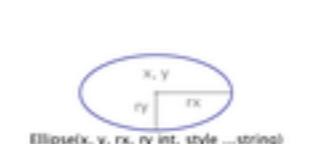
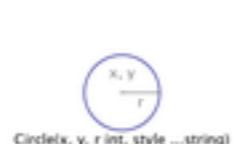
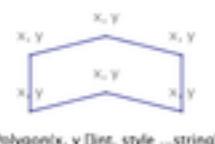
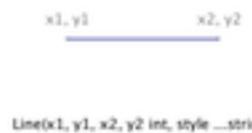
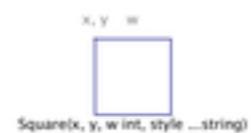
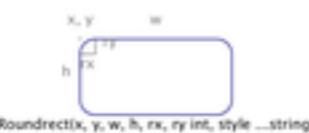
Visual Design and Relationships



SVG

SVG Go Library

github.com/ajstarks/svg



hello, this is SVG

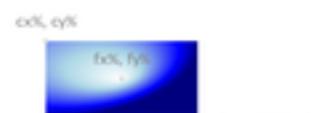
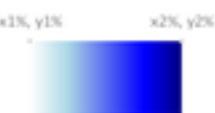
Text(x, y int, s string, style ...string)

It's time to draw something

Textpath(s, pathid string, style ...string)

W3C

Path(s string, style ...string)



NewFile io.Writer
Start(w, h int, options ...string)(End)
StartViewbox, h, minx, miny, w, vh int
Group(s ...string)(End)
Gstyle(s string)(End)
Transform(s string)(End)
Gid(id string)(End)
ClipPath(s ...string)(Clip(End))
Def0(DefEnd)
Marker0(MarkerEnd)
Pattern0(PatternEnd)
Desct(s string)
Title(s string)
Script(type, data ...string)
Mask(id string, x,y,w,h int, style ...string)(MaskEnd)
Link(href string, title string)(LinkEnd)
Use(x int, y int, link string, style ...string)

specify destination
begin/end the document
begin/end the document with viewport
begin/end group with attributes
begin/end group style
begin/end group transform
begin/end group id
begin/end clip path
begin/end a definition block
begin/end markers
begin/end pattern
set the description element
set the title element
define a script
begin/end mask element
begin/end link to href, with a title
use defined objects
Textlines(x, y int, s []string, size, spacing int, fill, align string)

Element

|

Rect

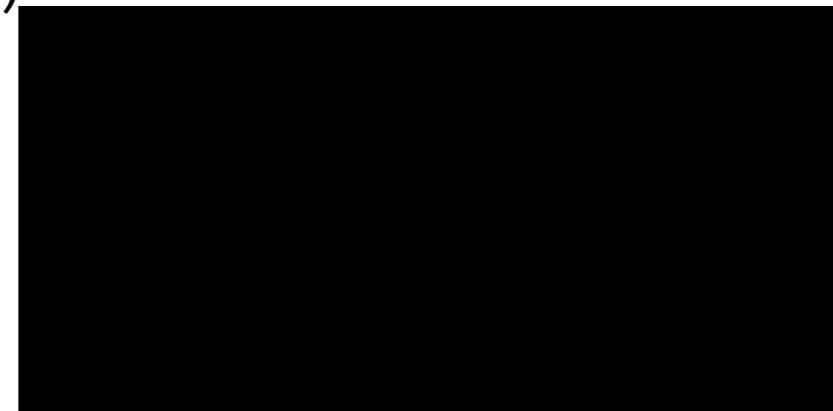
Arguments

|

(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 := 600
    height := 338
    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()
}
```



```
package main

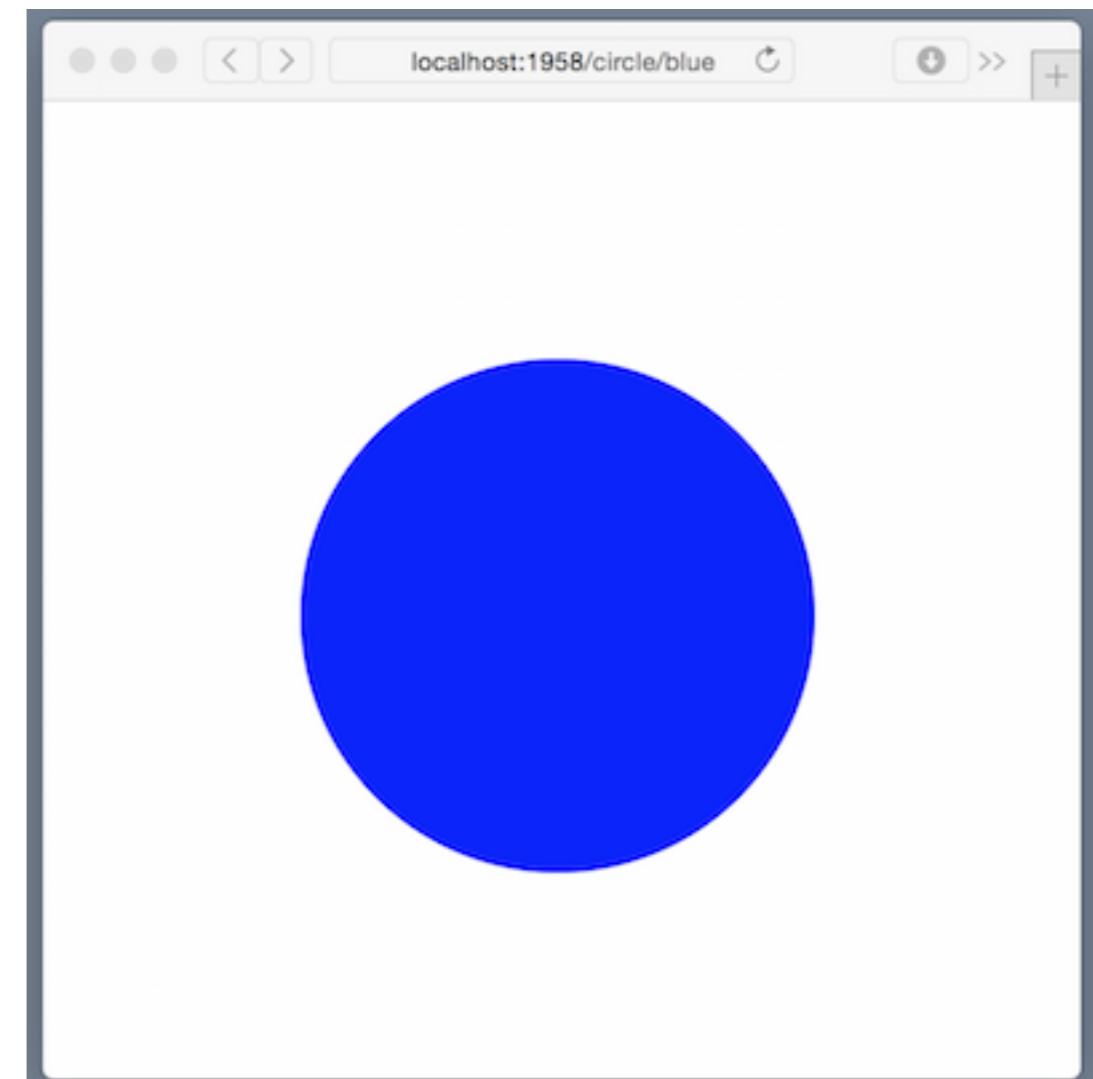
import (
    "github.com/ajstarks/svg"
    "log"
    "net/http"
    "strings"
)

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
}
```

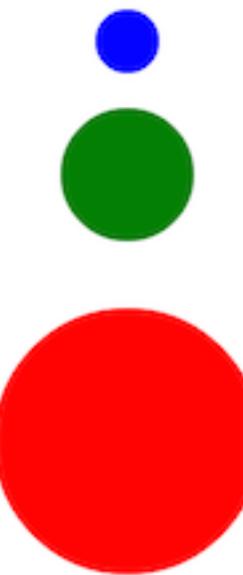


Read/Parse/Draw Pattern

Data



Picture



```
<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>
```

Little:This is small/blue

Med:This is medium/green

Big:This is large/red

```
package main

// Imports
import (
    "encoding/xml"
    "flag"
    "fmt"
    "io"
    "os"

    "github.com/ajstarks/svg"
)

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"`
}

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

// Open the file
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)
}

// Read the file, loading the defined structure
func readthing(r io.Reader) {
    var t Thing
    if err := xml.NewDecoder(r).Decode(&t); err != nil {
        fmt.Fprintf(os.Stderr, "%v\n", err)
        return
    }
    drawthing(t)
}

// use the items of "thing" to make the picture
func drawthing(t Thing) {
    x := t.Left
    y := t.Top
    for _, v := range t.Item {
        style := fmt.Sprintf("font-size:%dpx;fill:%s", v.Width/2, v.Color)
        canvas.Circle(x, y, v.Height/4, "fill:"+v.Color)
        canvas.Text(x+t.Sep, y, v.Name+":"+v.Text+"/"+v.Color, style)
        y += v.Height
    }
}

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

Imports

Define the input data structures

Define flags and output destination

Read the input

Parse and load the data structures

Draw

Main

Imports

```
import (
    "encoding/xml"
    "flag"
    "fmt"
    "io"
    "os"

    "github.com/ajstarks/svg"
)
```

The main

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

Define the input 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"`
}

<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>
```

Define flags and output destination

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

Read the input

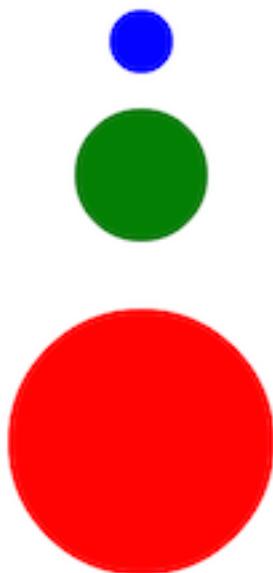
```
func dothing(location string) {
    f, err := os.Open(location)
    if err != nil {
        fmt.Fprintf(os.Stderr, "%v\n", err)
        return
    }
    defer f.Close()
    readthing(f)
}
```

Parse and load the data structures

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

Use the items of "thing" to make the picture

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



Little:This is small/blue

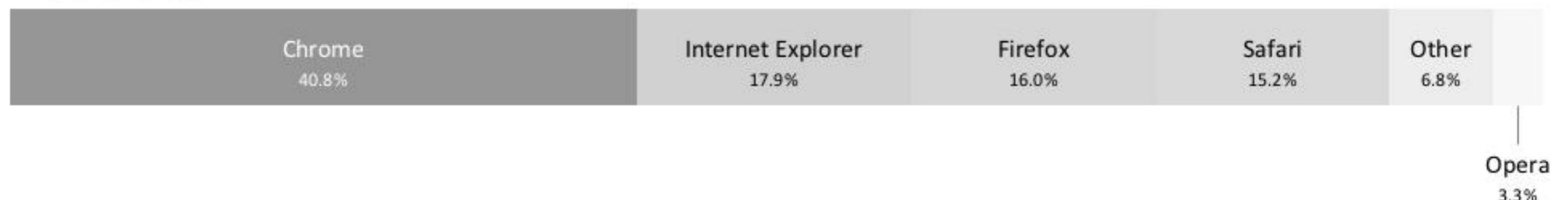
Med:This is medium/green

Big:This is large/red

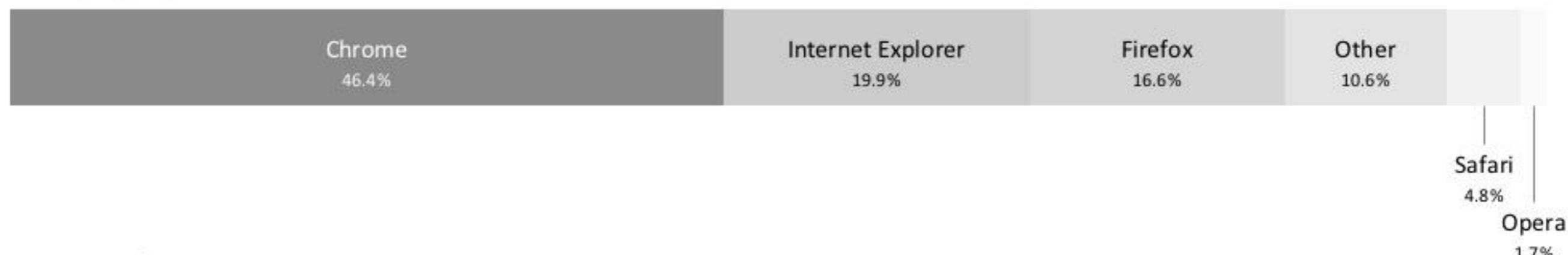
StatCounter



W3C Counter



Wikimedia



Net Applications



Browser Market Share

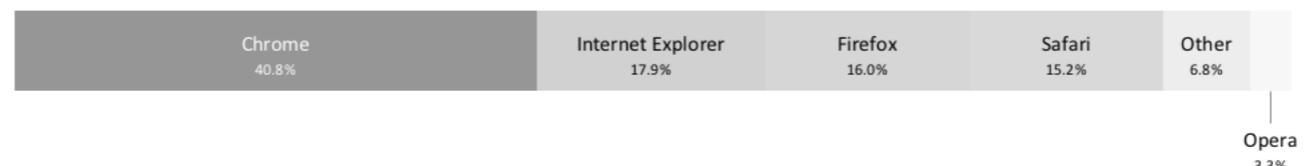
```
pmap -stagger -showtitle -p -t "Browser Market Share" bs.xml
```

```
<pmap>
  <pdata legend="StatCounter">
    <item value="51.3">Chrome</item>
    <item value="21.3">Internet Explorer</item>
    <item value="18.8">Firefox</item>
    <item value="5.1">Safari</item>
    <item value="2.1">Other</item>
    <item value="1.4">Opera</item>
  </pdata>
  ...
</pmap>
```

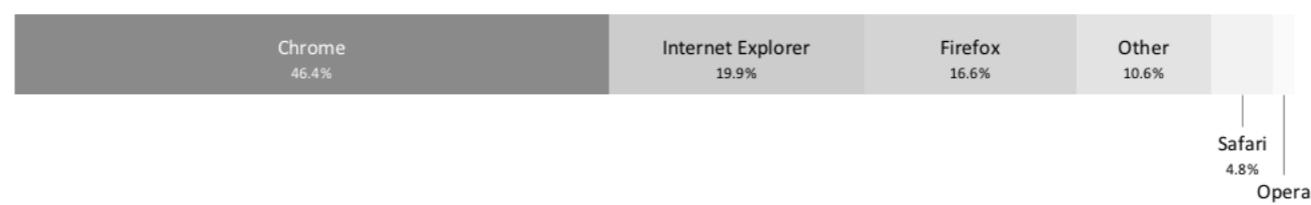
StatCounter



W3C Counter



Wikimedia

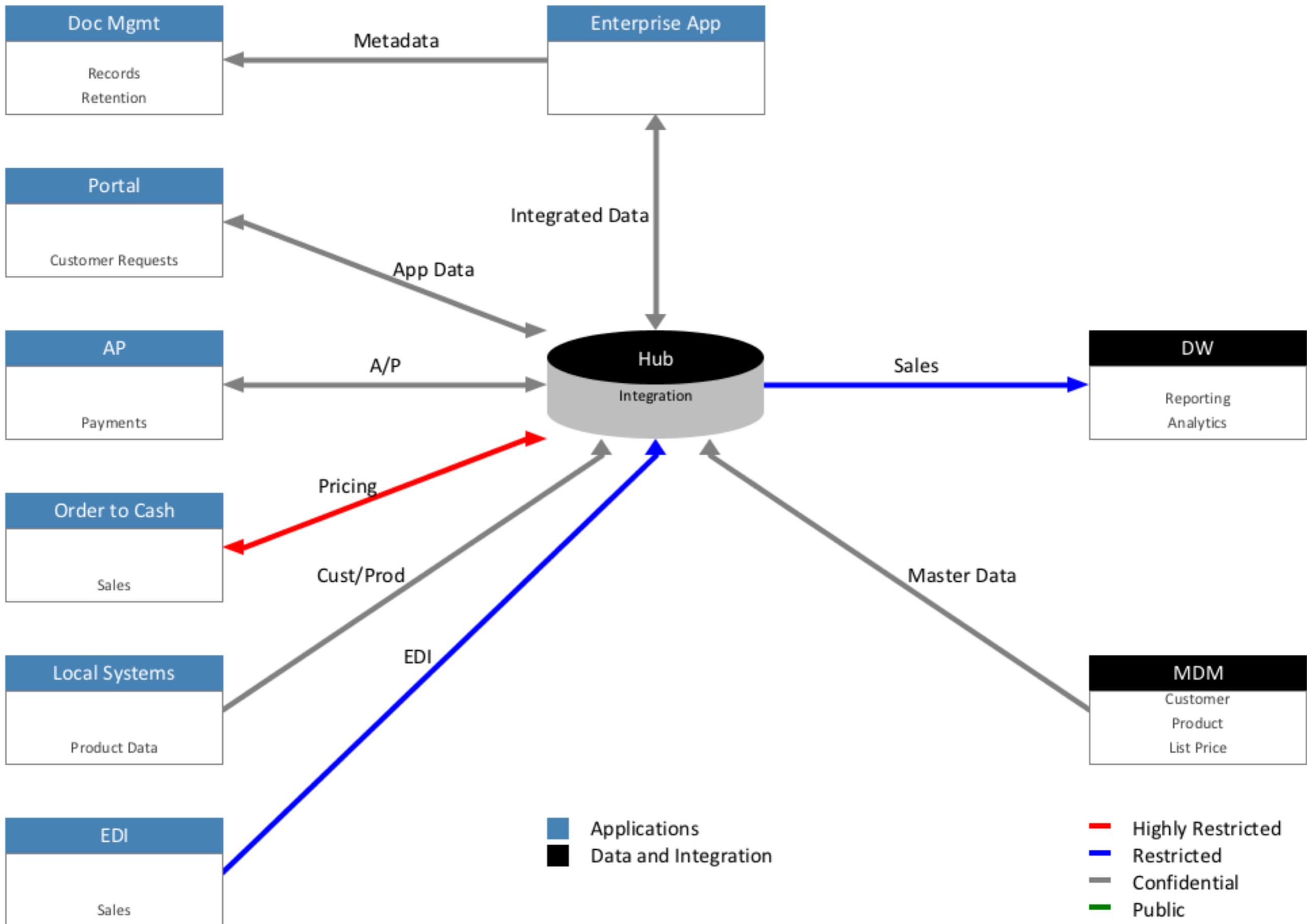


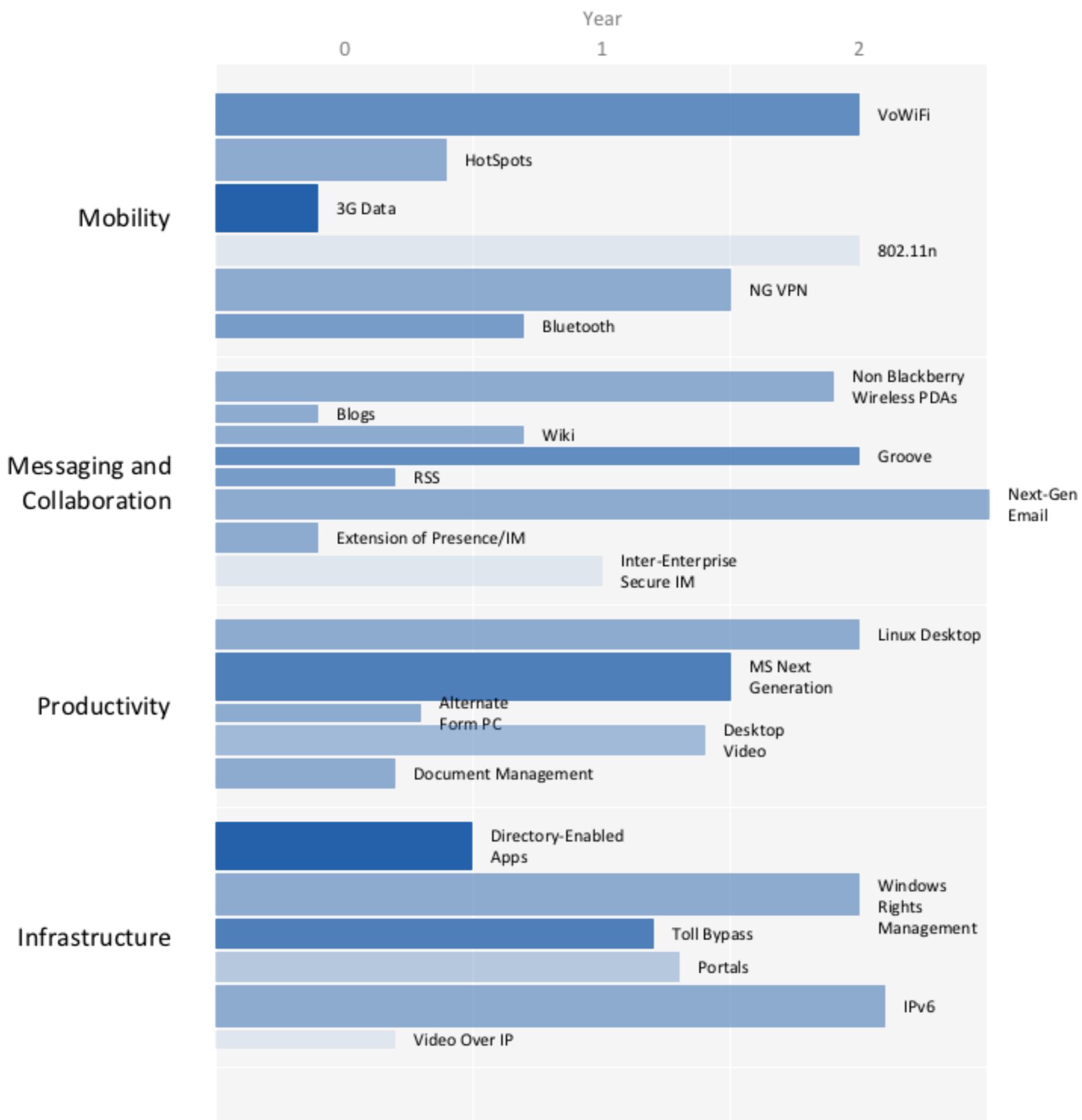
Net Applications

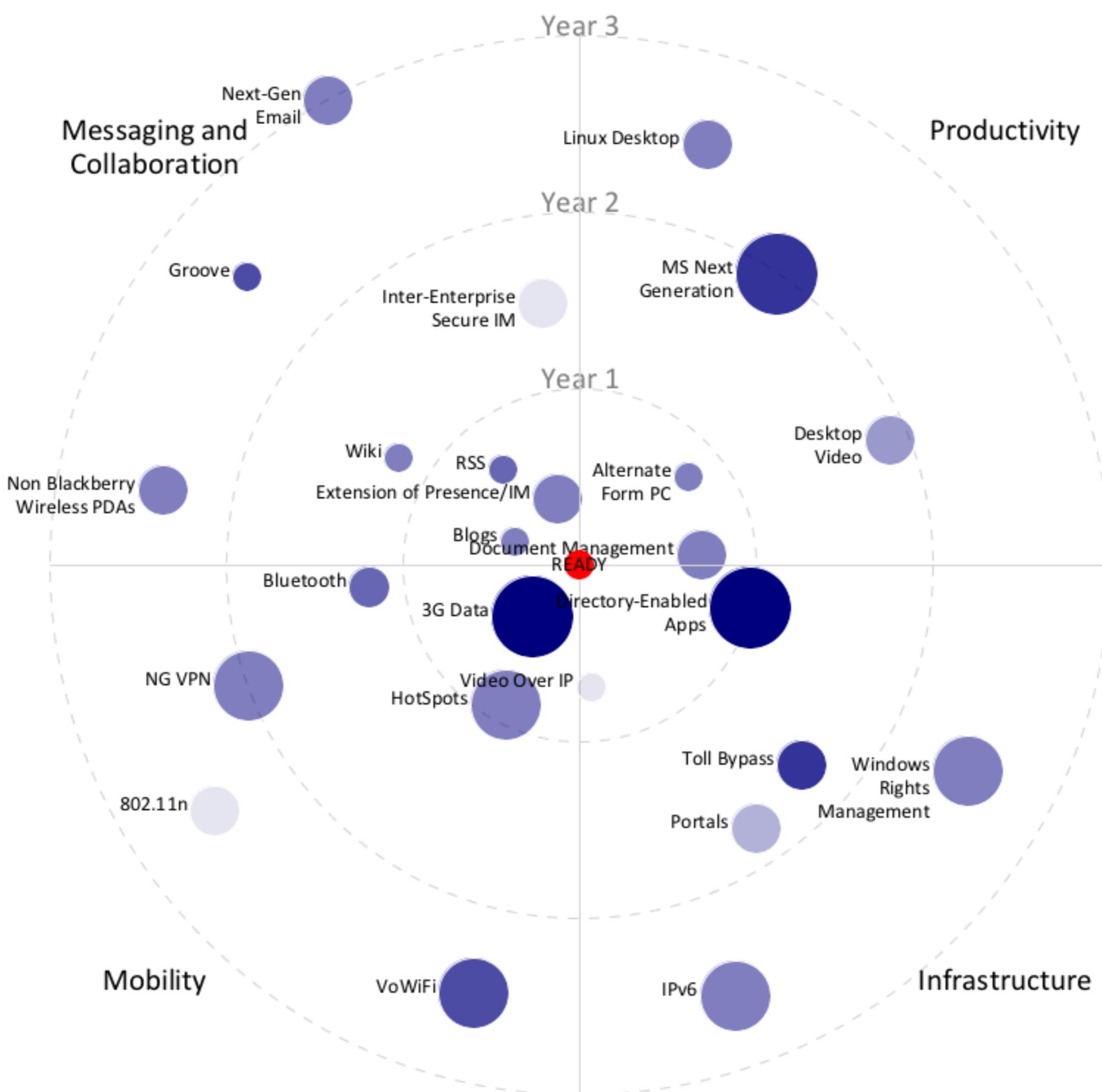


Browser Market Share

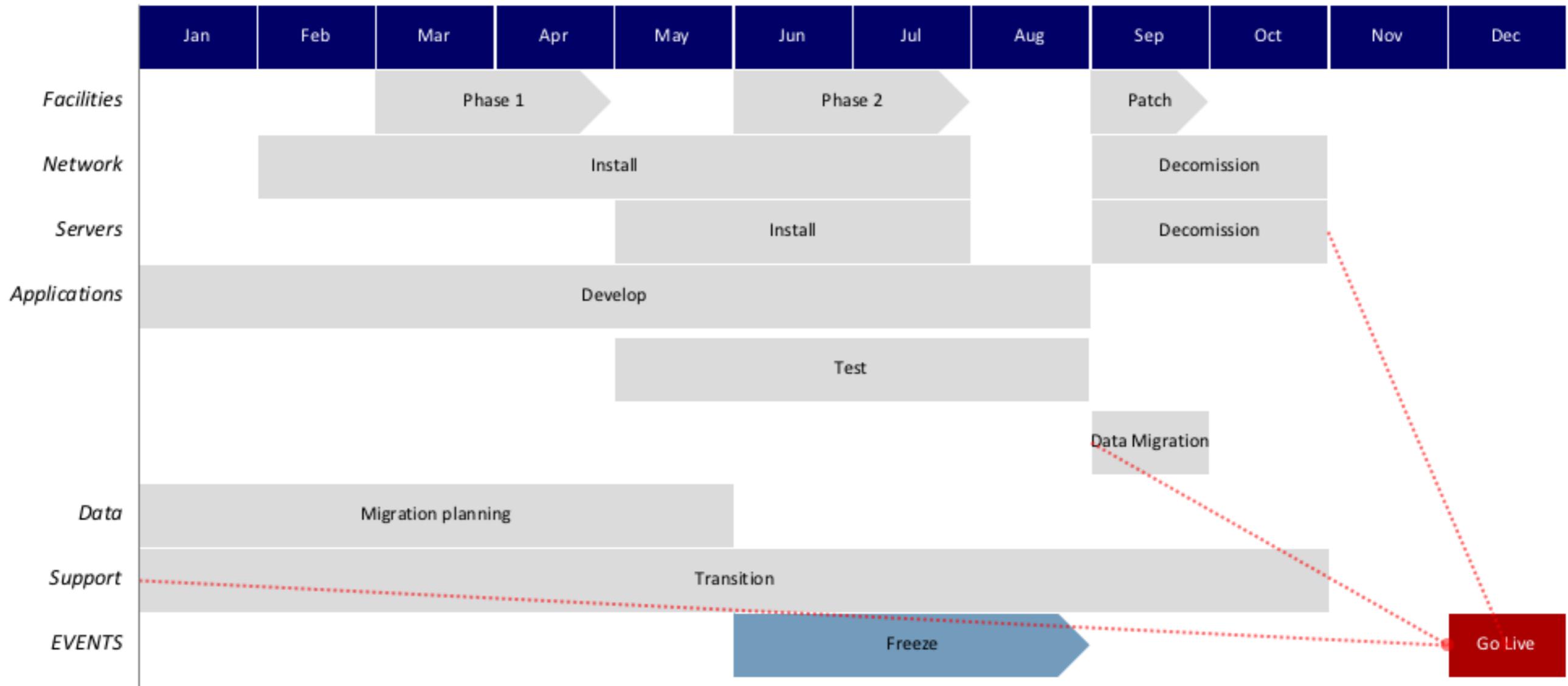
SVGo Clients





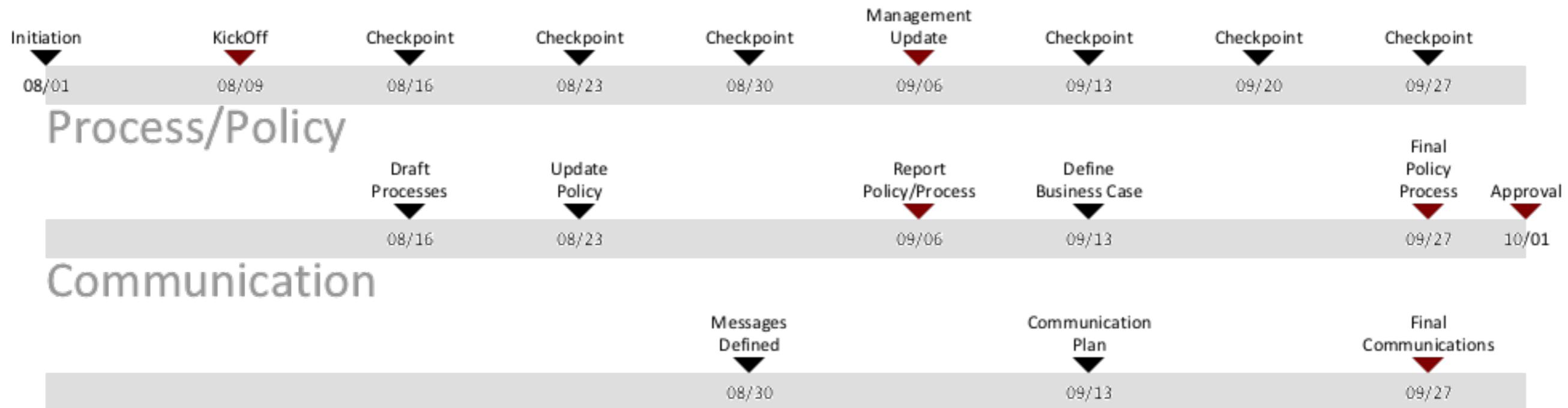


PLANNING TO PLAN



Project Timeline

Meetings



Ability to Execute

Definite

Thing 13

Item 16

Thing 9

Thing 1

Item 18

Thing 19

Likely

Thing 8
Thing 10

Thing 12

Thing 2

Thing 7

Moderate

Thing 14

Item 15

Thing 6

Item 17
Item 20

Thing 3

Unlikely

Thing 4

Thing 5

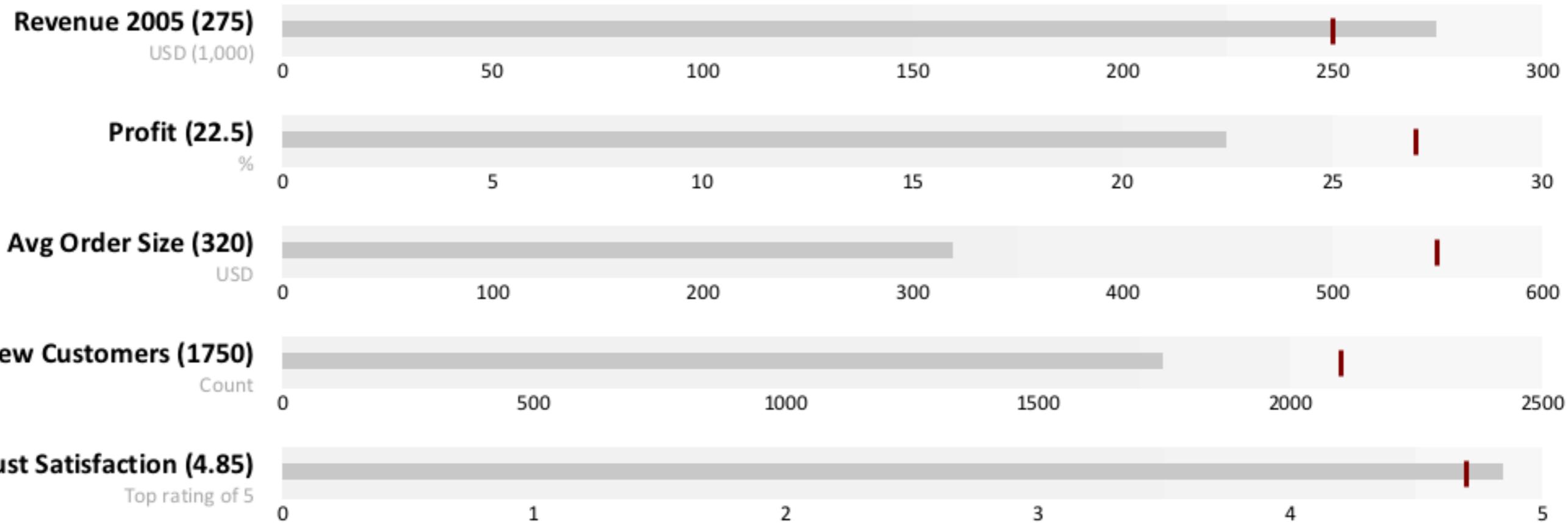
Weak

Moderate

Strong

Extereme

Strategic Alignment

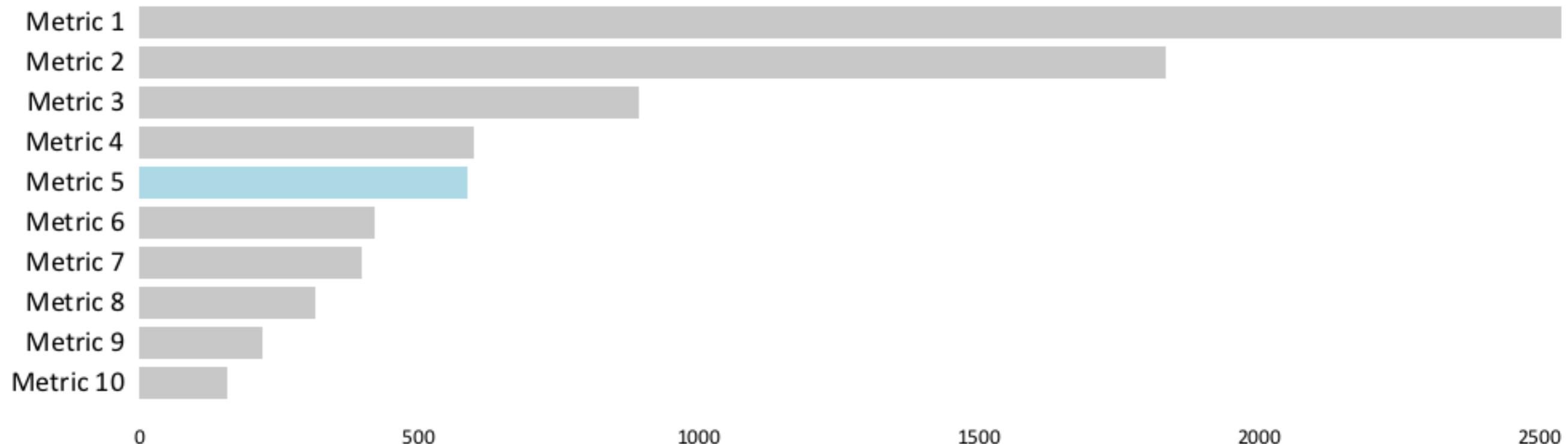


Sample Bullet Graph

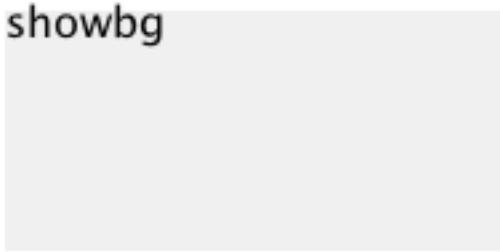
The bullet graph features a single, primary measure (for example, current year-to-date revenue) compares that measure to one or more other measures to enrich its meaning, for example, compared to a target), and displays it in the context of qualitative ranges of performance, such as poor, satisfactory,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

The qualitative ranges are displayed as varying intensities of a single hue to make them discernible by those who are color blind and to restrict the use of colors on the dashboard to a minimum.

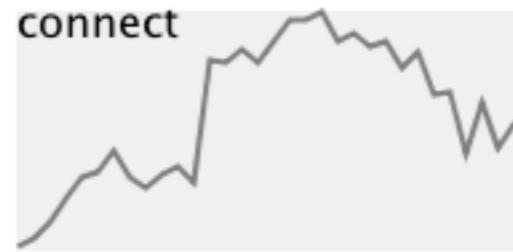
IS Metrics Usage (Feb-Sep 2011)



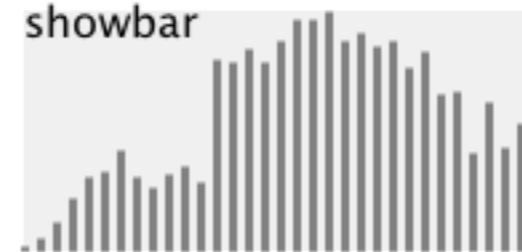
showbg



connect



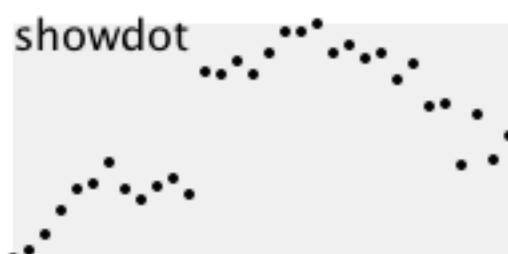
showbar



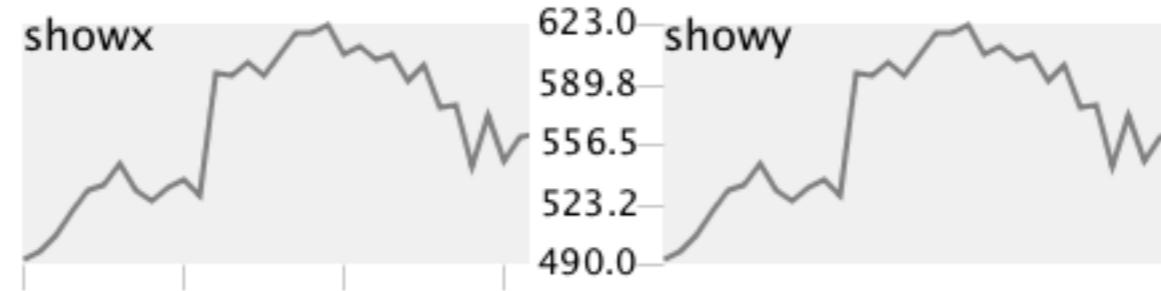
area



showdot



showx



showy



test.d



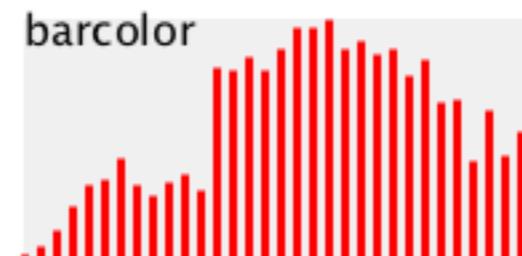
bgcolor



linecolor



barcolor



hcolor



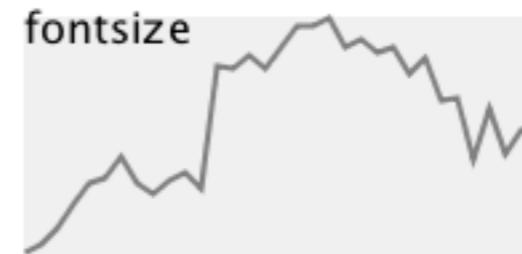
dotcolor



labelcolor



fontsize



font



JONES MONTIEL

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



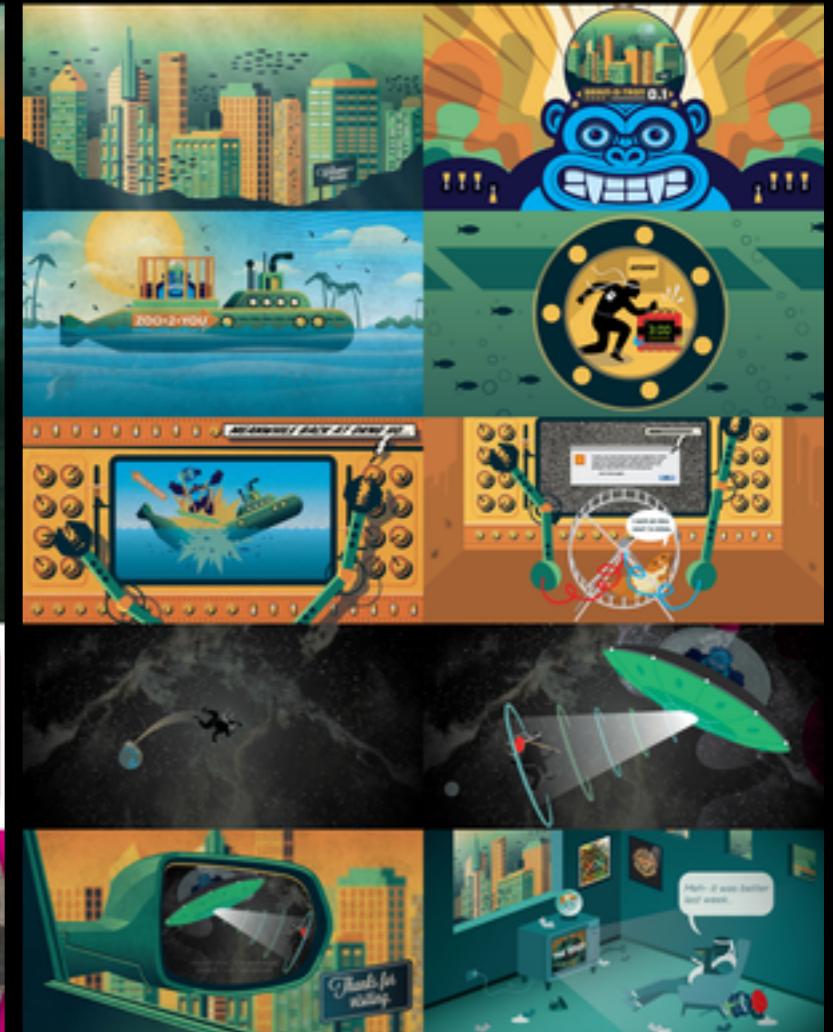
ANDERSON DKNG

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

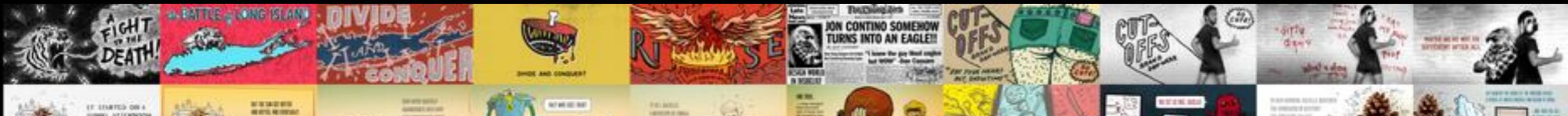


DKNG DDL

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



CONTINO
CASSARO



PUTNAM
STOCKS



WHITE
TAYLOR



HALL
WARREN



STRAWBERRY LUNA
DOUBLEXAUT



JONES
MONTIEL



DKNG
DDL.



SHAWNA X
STEVENS



TAYLOR
JONES



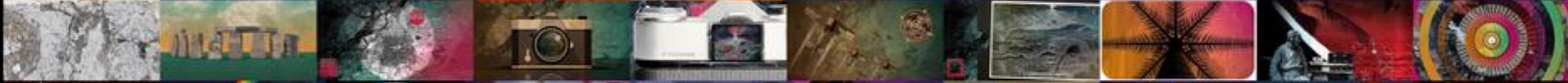
PUTNAM
RAJKUMAR



REYES
WHITE



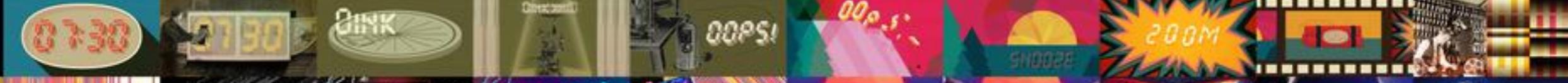
ANDERSON
DKNG



TAYLOR
WHITE



RAJKUMAR
ANDERSON

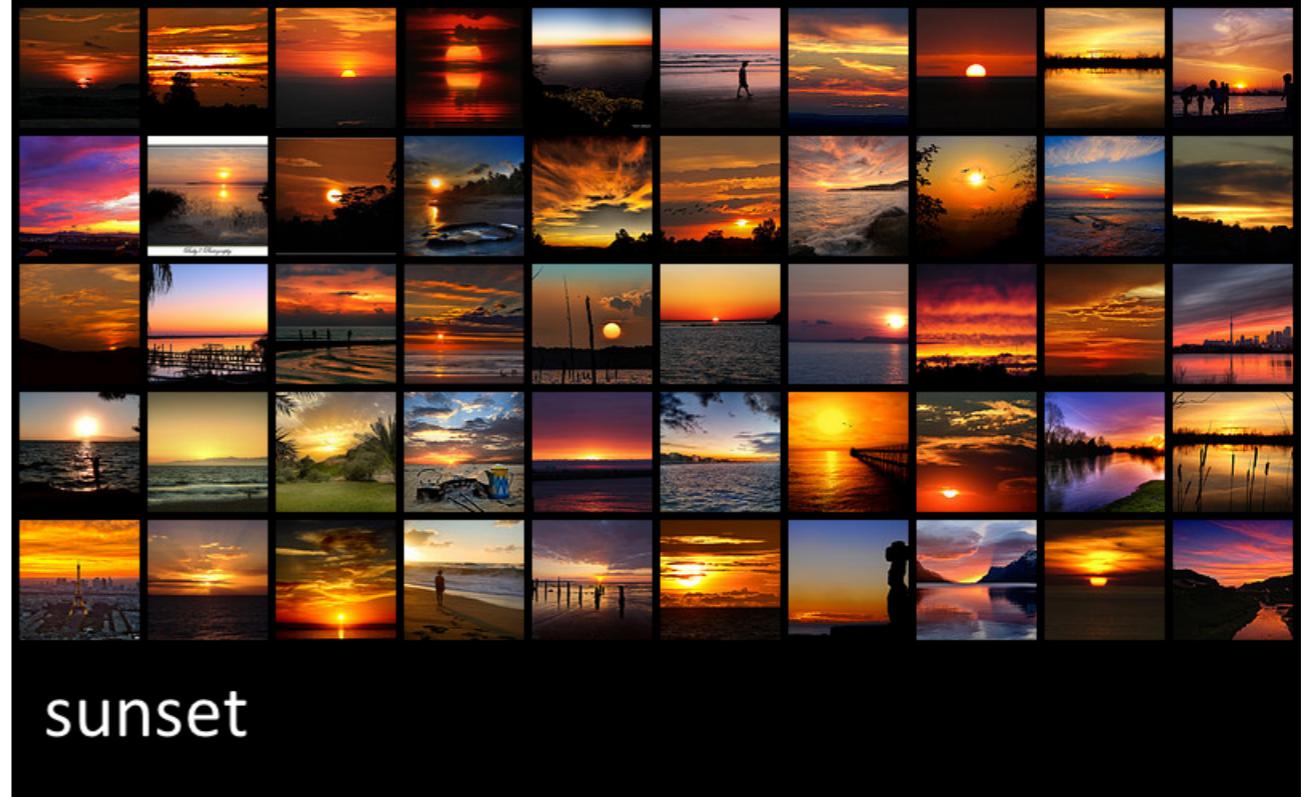


ANDERSON
WHITE

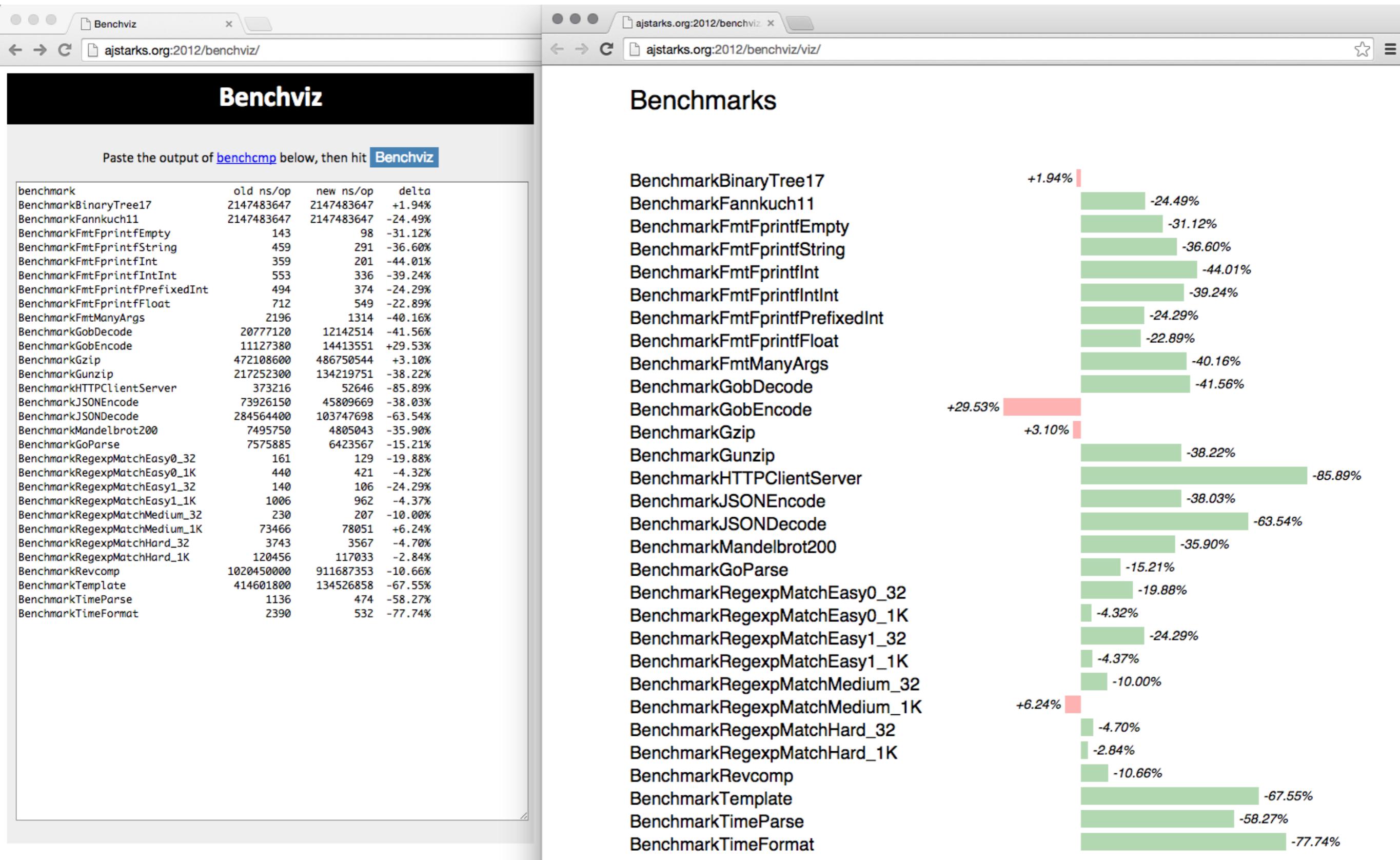


f50 sunset

```
https://api.flickr.com/services/rest/  
?method=flickr.photos.search  
&api_key=...  
&text=sunset  
&per_page=50  
&sort=interestingness-desc
```



```
<?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>
```



```
package main

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

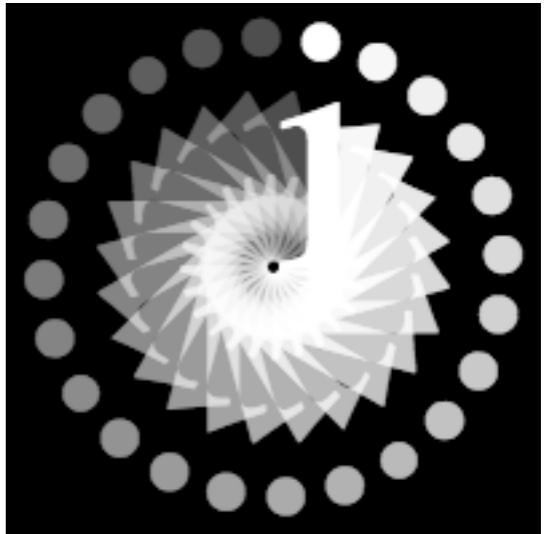
func main() {

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

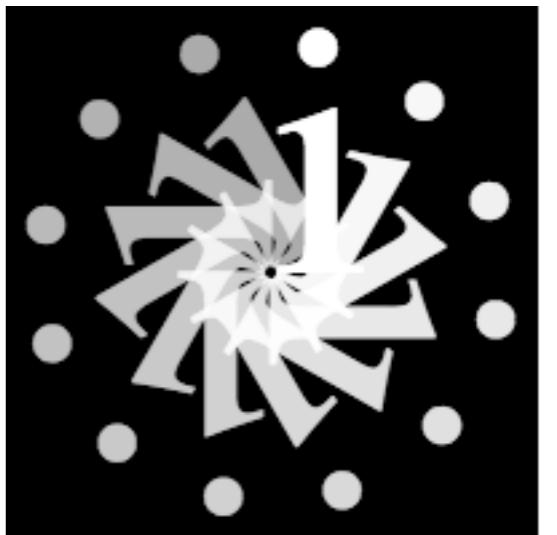
    canvas := svg.New(os.Stdout)
    canvas.Start(width, height)
    canvas.Rect(0, 0, width, height)
    canvas.Gstyle("font-family:serif;font-size:100pt")

    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 = 15



ti = 30

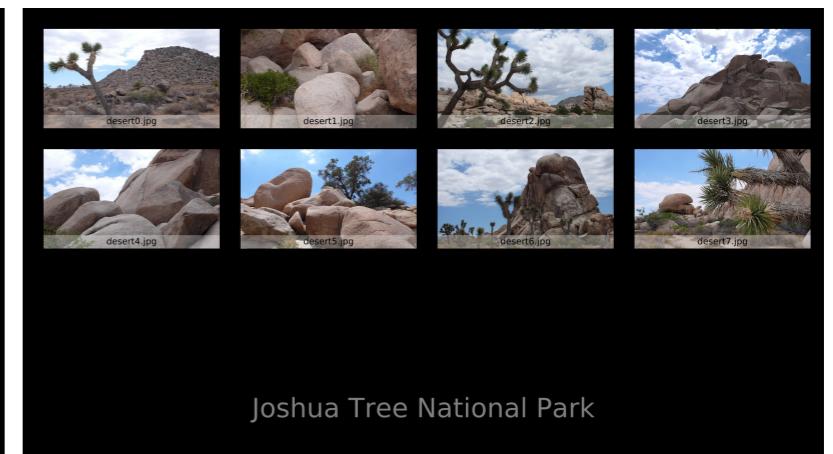
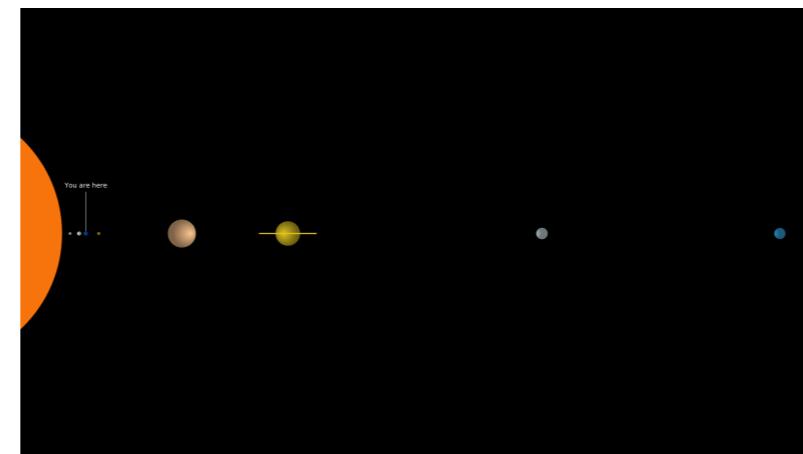
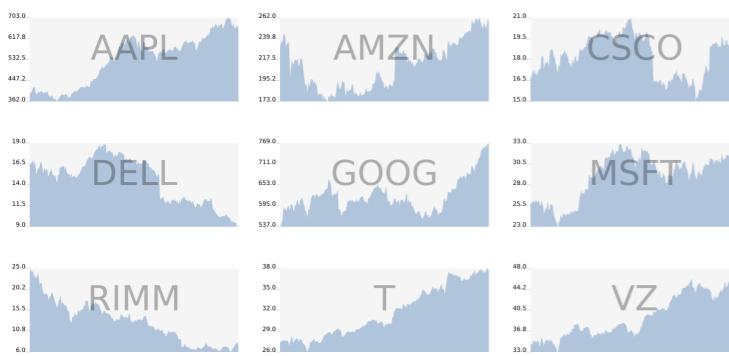
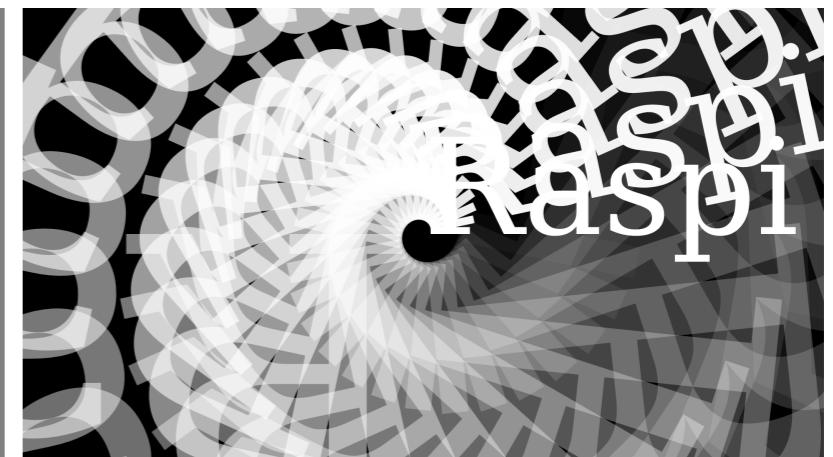
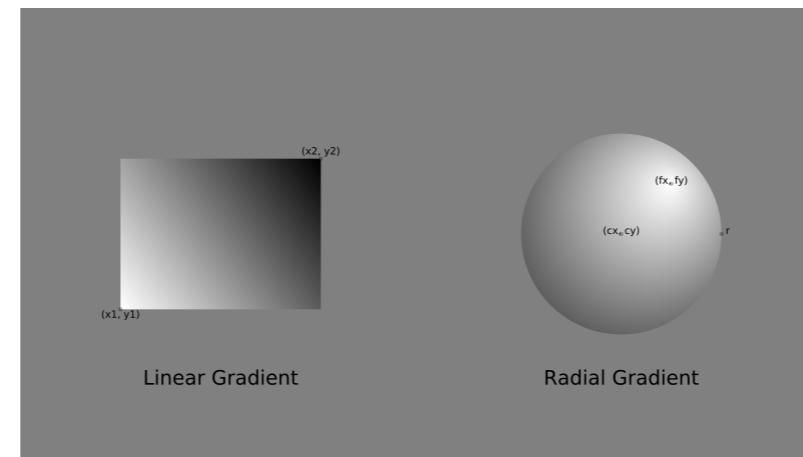


ti = 45



-  Circle
-  Ellipse
-  Rectangle
-  Rounded Rectangle
-  Line
-  Polyline
-  Polygon
-  Arc
-  Quadratic Bezier
-  Cubic Bezier
-  Image

OpenVG on the Raspberry Pi



Joshua Tree National Park

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.

openvg

```
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 string, font string, size int)
TextMid	(x, y VGfloat, s string, font string, size int)
TextEnd	(x, y VGfloat, s string, 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="256" height="179" name="work.png" caption="Desk"/>
Draw some text	<text xp="20" yp="80" sp="3" link="http://goo.gl/Wm05Ex">Deck elements</text>
Make a bullet list	<list xp="20" yp="70" sp="2" type="bullet"> text, list, image line, rect, ellipse arc, curve, polygon </list>
End the 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>

Deck elements

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



Desk



Percent Grid

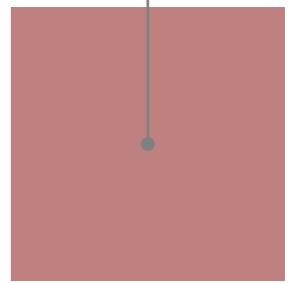
10%, 50%

Hello

50%, 50%

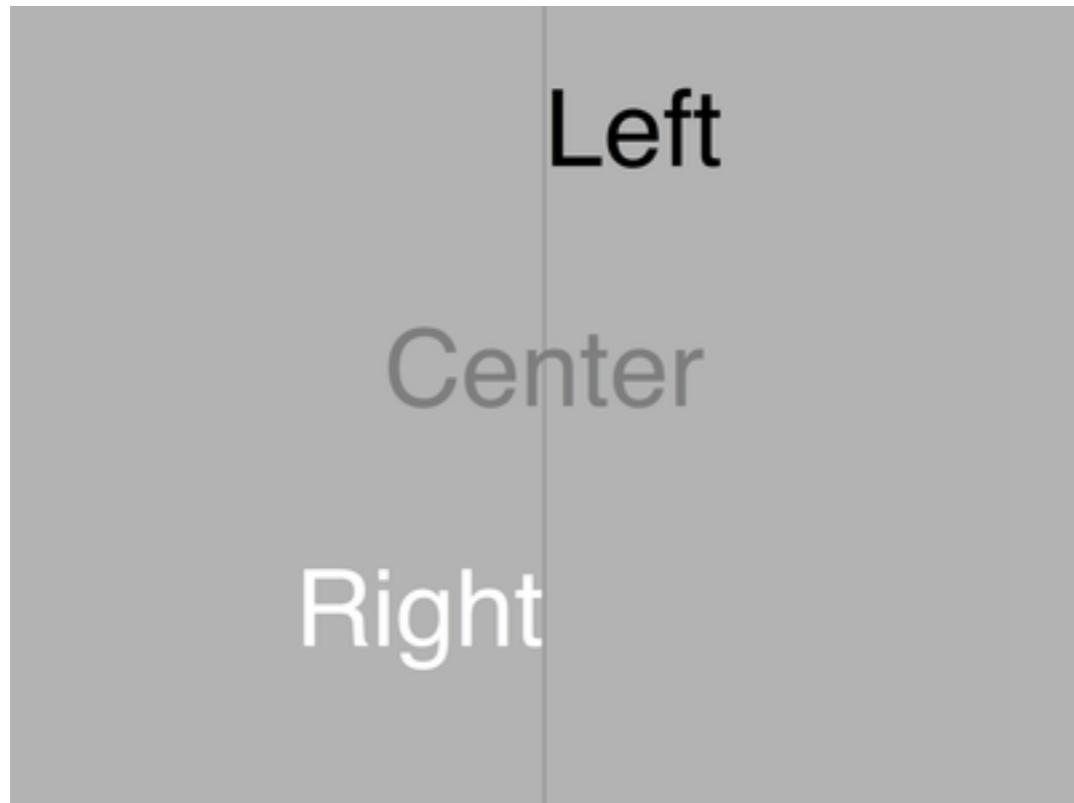


90%, 50%

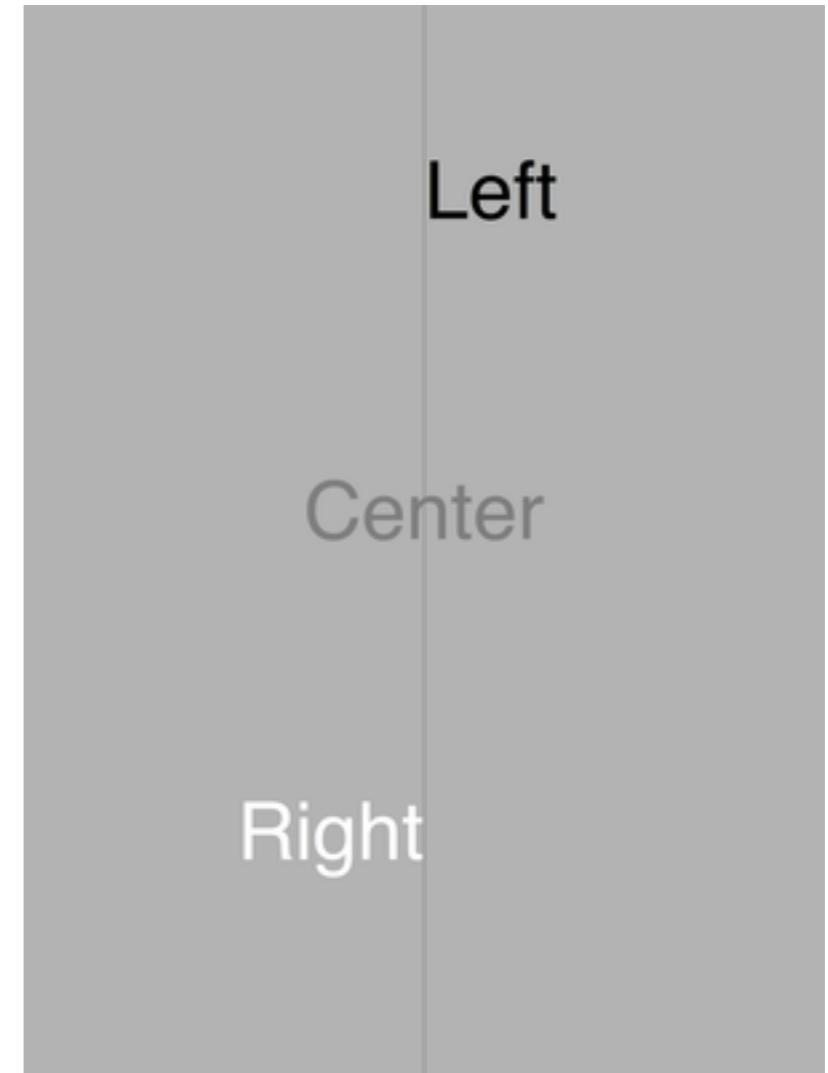


Percentage-based layout

Scaling the canvas



Landscape



Portrait

Process



deck code

interactive

PDF

SVG

NewSlides	(where io.Writer, w, h int)	
StartDeck()/EndDeck()		
StartSlide	(colors ...string)	EndSlide()
Arc	(x, y, w, h, size, a1, a2 float64, color string, opacity ...float64)	
Circle	(x, y, w float64, color string, opacity ...float64)	
Code	(x, y float64, s string, size, margin float64, color string, opacity ...float64)	
Curve	(x1, y1, x2, y2, x3, y3, size float64, color string, opacity ...float64)	
Ellipse	(x, y, w, h float64, color string, opacity ...float64)	
Image	(x, y float64, w, h int, name string)	
Line	(x1, y1, x2, y2, size float64, color string, opacity ...float64)	
List	(x, y, size float64, items []string, ltype, font, color string)	
Polygon	(x, y []float64, color string, opacity ...float64)	
Rect	(x, y, w, h float64, color string, opacity ...float64)	
Square	(x, y, w float64, color string, opacity ...float64)	
Text	(x, y float64, s, font string, size float64, color string, opacity ...float64)	
TextBlock	(x, y float64, s, font string, size, margin float64, color string, opacity ...float64)	
TextEnd	(x, y float64, s, font string, size float64, color string, opacity ...float64)	
TextMid	(x, y float64, s, font string, size 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

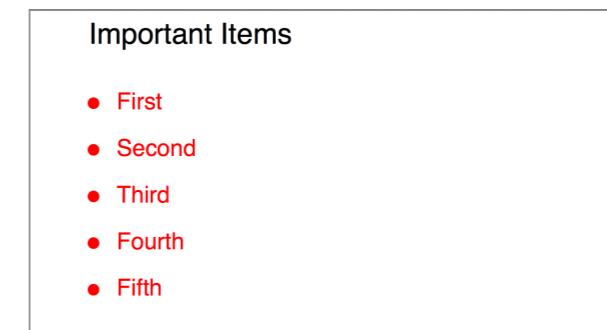
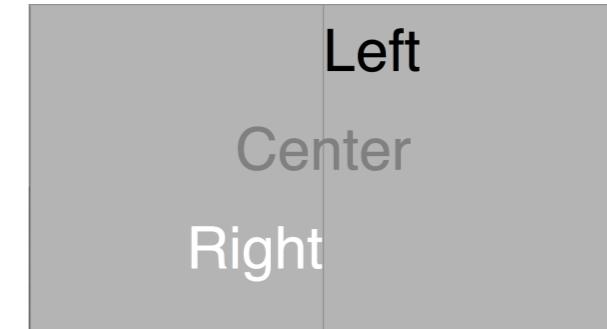
    // Text alignment
    deck.StartSlide("rgb(180,180,180)")
    deck.Text(50, 80, "Left", "sans", 10, "black")
    deck.TextMid(50, 50, "Center", "sans", 10, "gray")
    deck.TextEnd(50, 20, "Right", "sans", 10, "white")
    deck.Line(50, 100, 50, 0, 0.2, "black", 20)
    deck.EndSlide()

    // 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()

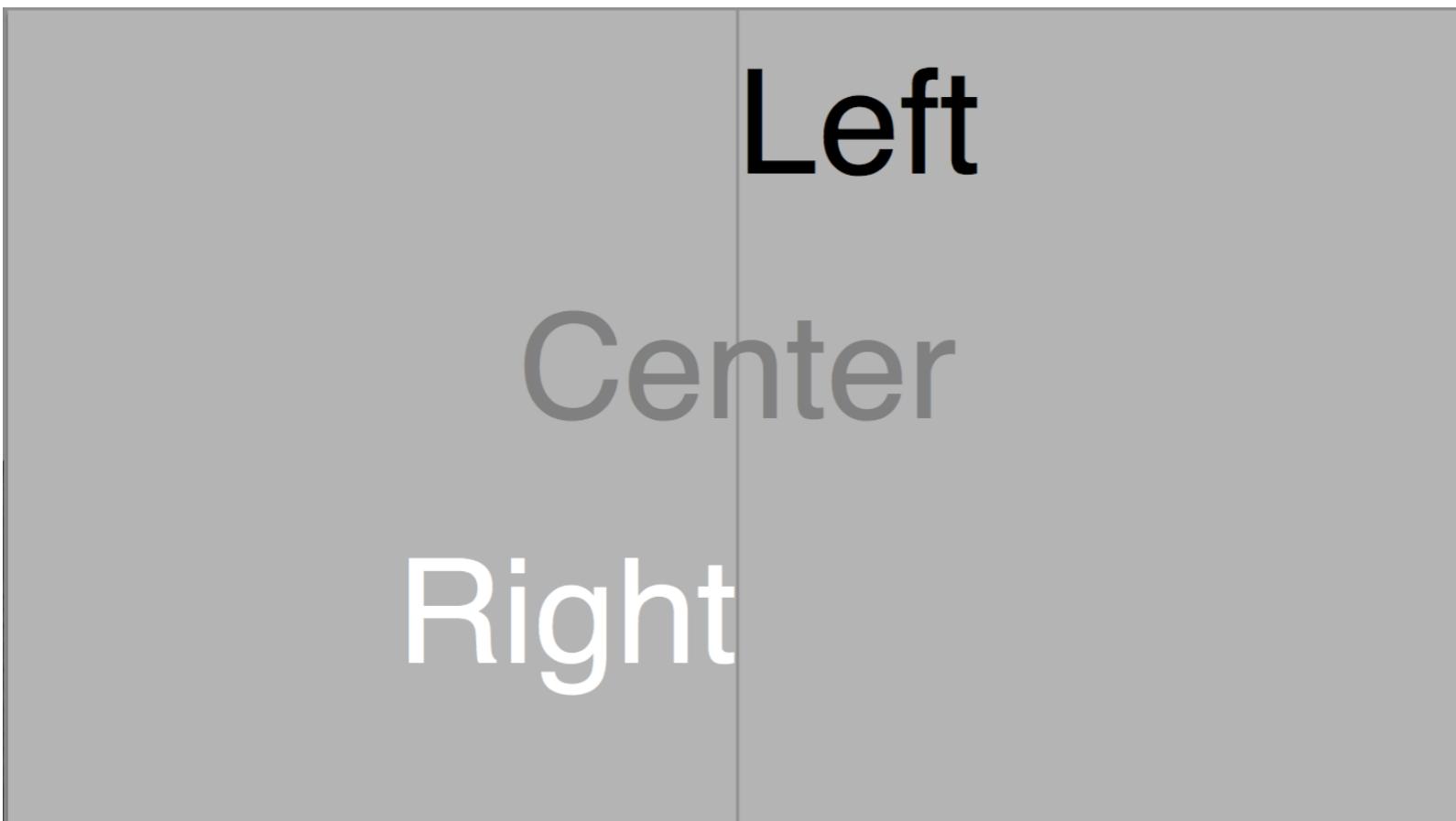
    // Picture with text annotation
    quote := "Yours is some tepid, off-brand, generic 'cola'. What I'm making is \"Classic Coke\""
    person := "Heisenberg"
    deck.StartSlide("black", "white")
    deck.Image(50, 50, 1440, 900, "classic-coke.png")
    deck.TextBlock(10, 80, quote, "sans", 2.5, 30,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

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

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.

Design Examples

Left

Top

Right

Bottom

10%

30%

70%

10%

Header (top 10%)

Summary
(30%)

Detail
(70%)

Footer (bottom 10%)

My Story

Section

One

Two

Three

Four

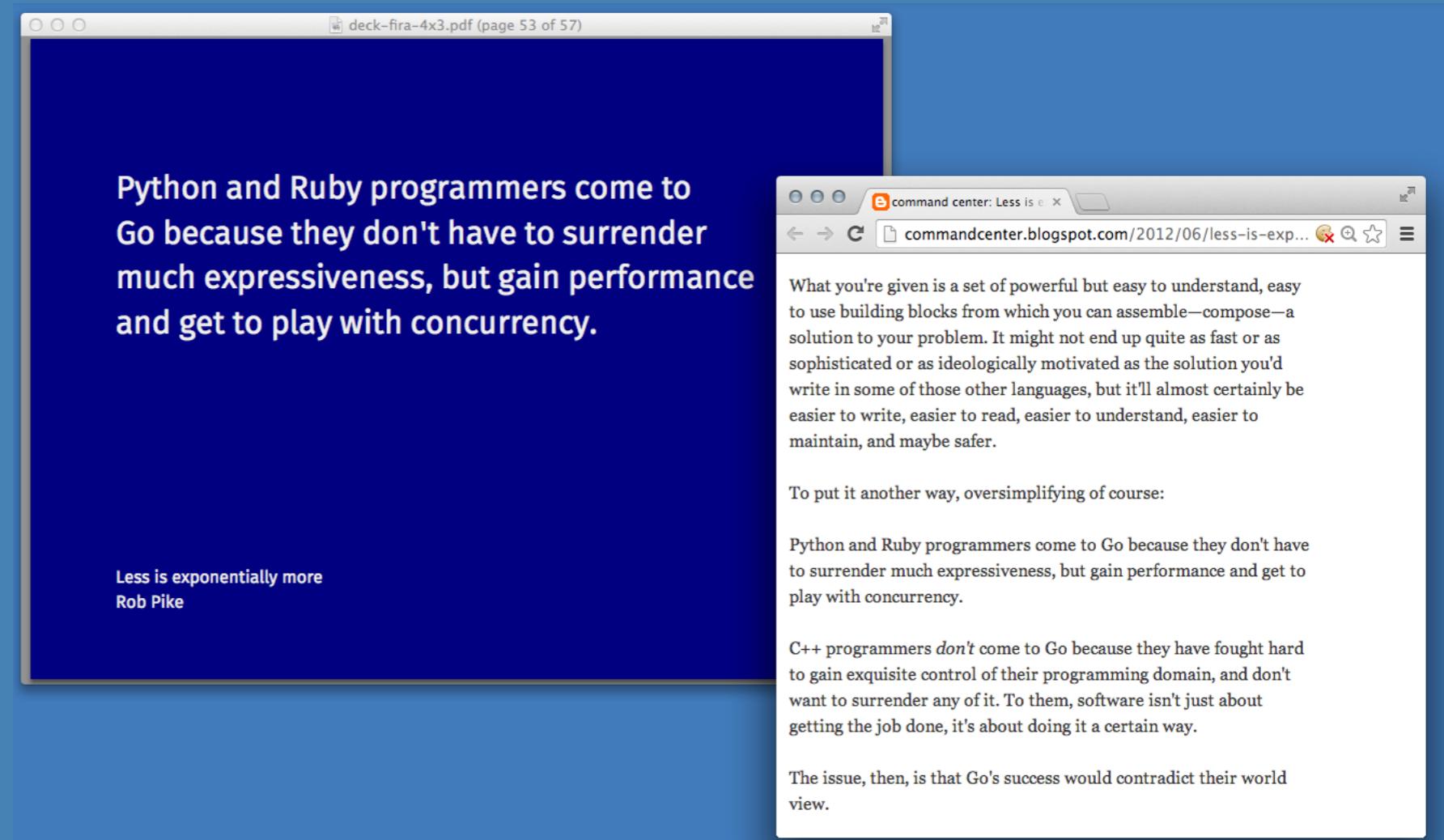
Five

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.



BOS



SFO

Virgin America 351

Gate B38

8:35am

On Time

JFK



IND

US Airways 1207

Gate C31C

5:35pm

Delayed

AAPL

110.22

+0.97 (0.89%)

AMZN

294.74

+3.33 (1.14%)

FB

76.45

-0.27 (0.35%)

GOOG

496.18

+3.63 (0.74%)

MSFT

46.35

-0.24 (0.53%)

Closing Price, 2015-01-13

Two Columns

One

Two

Three

Four

Five

Six

Seven

Eight



Tree and Sky



Rocks

go

build	compile packages and dependencies
clean	remove object files
env	print Go environment information
fix	run go tool fix on packages
fmt	run gobfmt on package sources
generate	generate Go files by processing source
get	download and install packages and dependencies
install	compile and install packages and dependencies
list	list packages
run	compile and run Go program
test	test packages
tool	run specified go tool
version	print Go version
vet	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

A few months ago, I had a look at the brainchild of a few serious heavyweights working at Google. Their project, the Go programming language, is a static typed, c lookalike, semicolon-less, self formatting, package managed, object oriented, easily parallelizable, cluster fuck of genius with an unique class inheritance system. It doesn't have one.

The Go Programming Language

is a static typed,
c lookalike,
semicolon-less,
self formatting,
package managed,
object oriented,
easily parallelizable,
cluster fuck of genius
with an unique class inheritance system.

The Go Programming Language

is a static typed,
c lookalike,
semicolon-less,
self formatting,
package managed,
object oriented,
easily parallelizable,
cluster fuck of genius
with an unique class inheritance system.

The Go Programming Language

is a static typed, c lookalike, semicolon-less, self formatting,
package managed, object oriented, easily parallelizable,
cluster fuck of genius with an unique class inheritance system

It doesn't have one.

So, the next time you're
about to make a subclass,
think hard and ask yourself

what would Go do

Andrew Mackenzie-Ross



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



You must not blame me if I do talk to the clouds.

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.

Good Design

is innovative

makes a product useful

is aesthetic

makes a product understandable

is unobtrusive

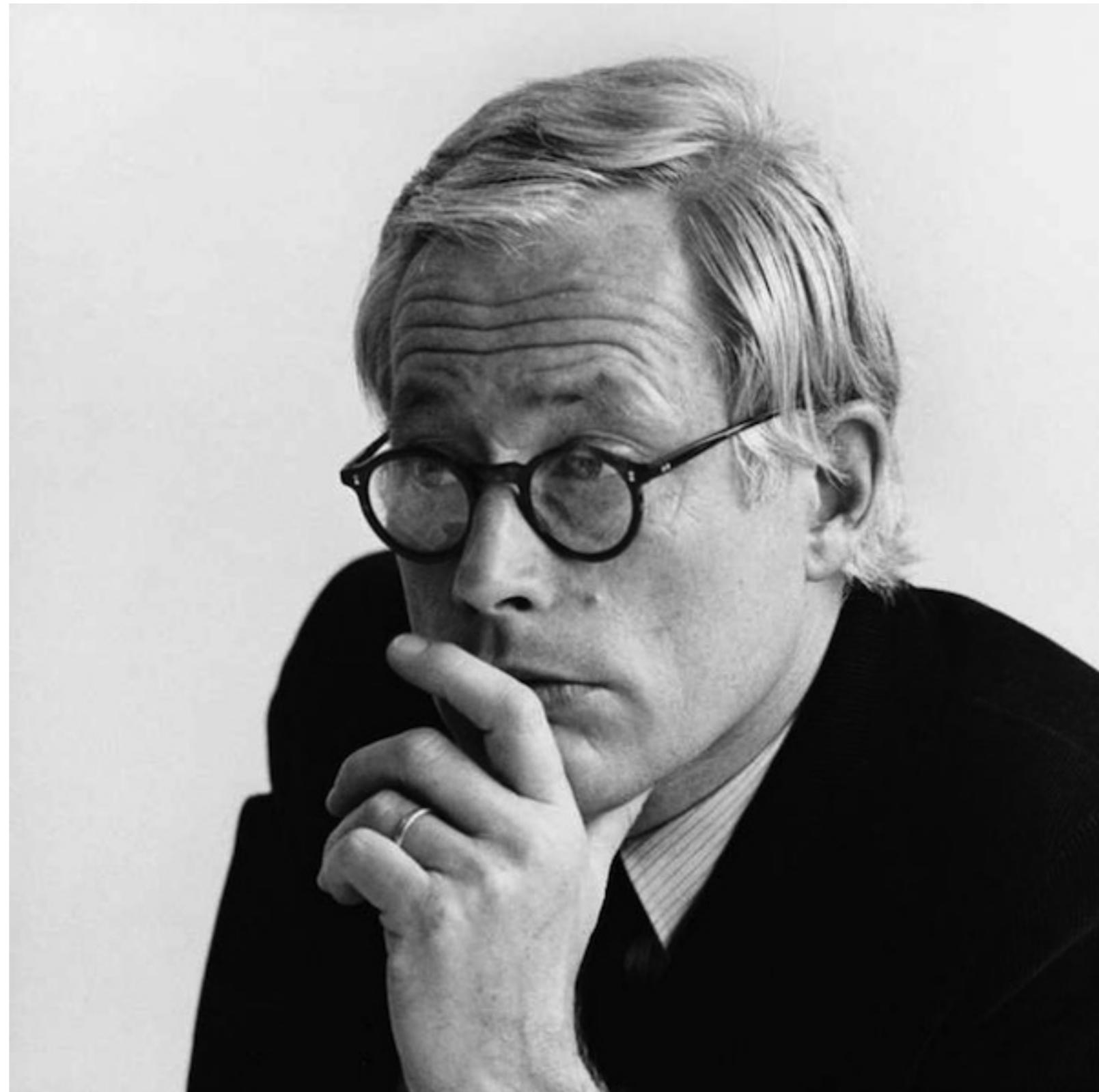
is honest

is long-lasting

is thorough down to the last detail

is environmentally-friendly

is as little design as possible

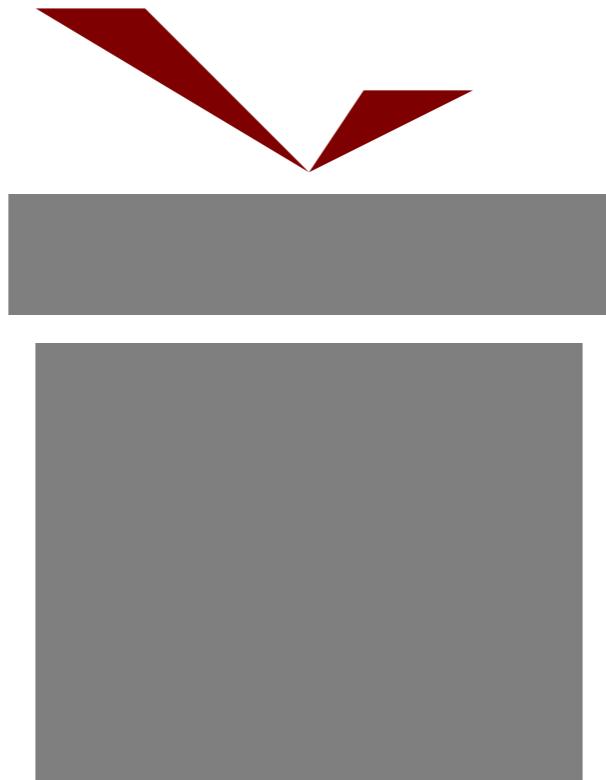


Dieter Rams



You've got to start with the customer experience and work back toward the technology, not the other way around.

Steve Jobs



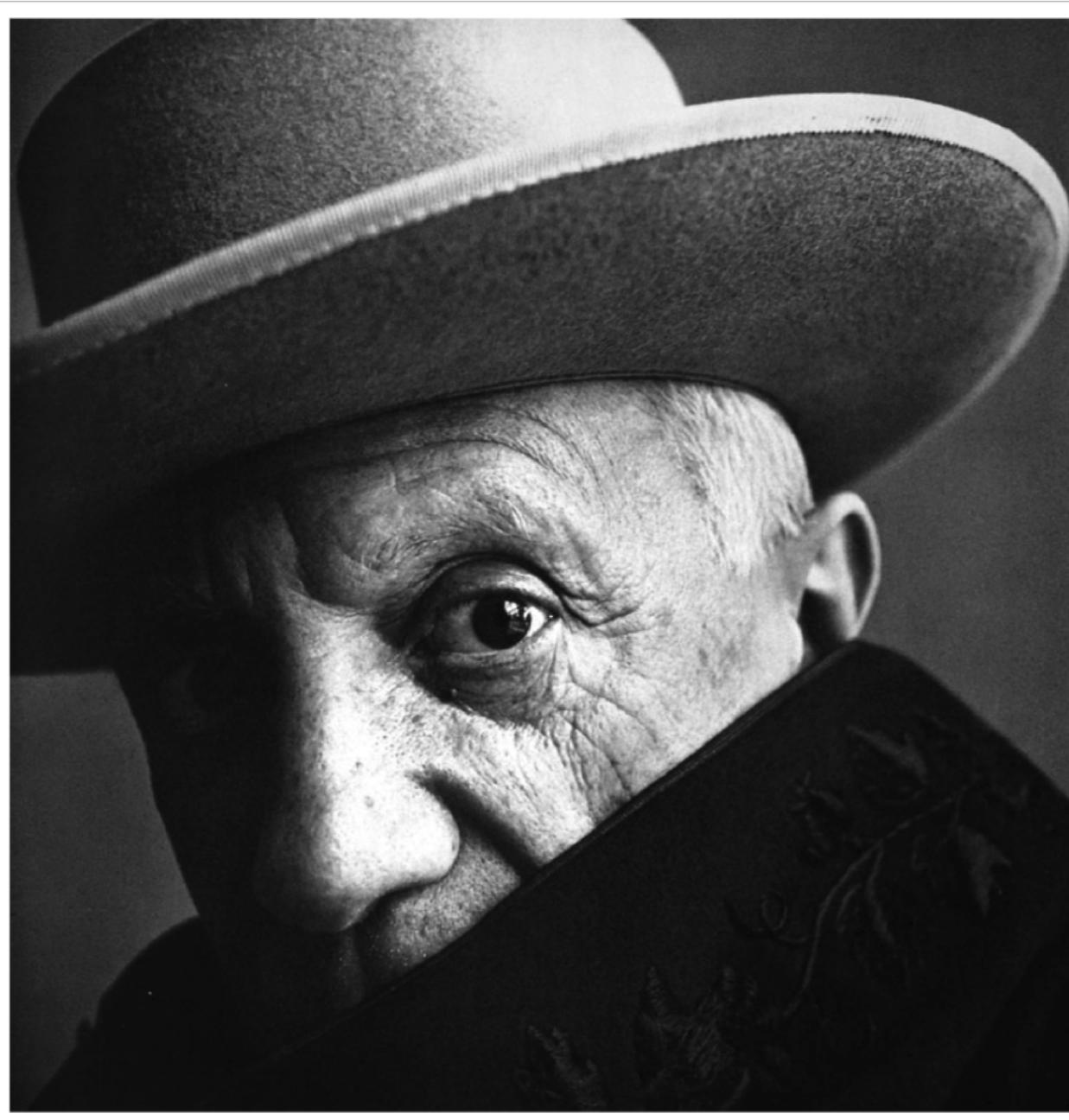
gift

Go Image Filtering Toolkit



gift -contrast 80 -blur 4 rocks.jpg > abstract.jpg

-blur=0: blur value	-max=0: local maximum (kernel size)
-brightness=-200: brightness value (-100, 100)	-mean=0: local mean filter (kernel size)
-contrast=-200: contrast value (-100, 100)	-median=0: local median filter (kernel size)
-crop="": crop x1,y1,x2,y2	-min=0: local minimum (kernel size)
-edge=false: edge	-resize="": resize w,h
-emboss=false: emboss	-rotate=0: rotate specified degrees counter-clockwise
-fliph=false: flip horizontal	-saturation=-200: saturation value (-100, 500)
-flipv=false: flip vertical	-sepia=-1: sepia percentage (0-100)
-gamma=0: gamma value	-sigmoid="": sigmoid contrast (midpoint,factor)
-gray=false: grayscale	-transpose=false: flip horizontally and rotate 90° counter-clockwise
-hue=-200: hue value (-180, 180)	-transverse=false: flips vertically and rotate 90° counter-clockwise
-invert=false: invert	-unsharp="": unsharp mask (sigma,amount,threshold)



Picasso



Turing

Thank you



github.com/ajstarks



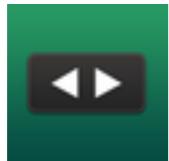
@ajstarks



ajstarks@gmail.com



flickr.com/photos/ajstarks



speakerdeck.com/ajstarks