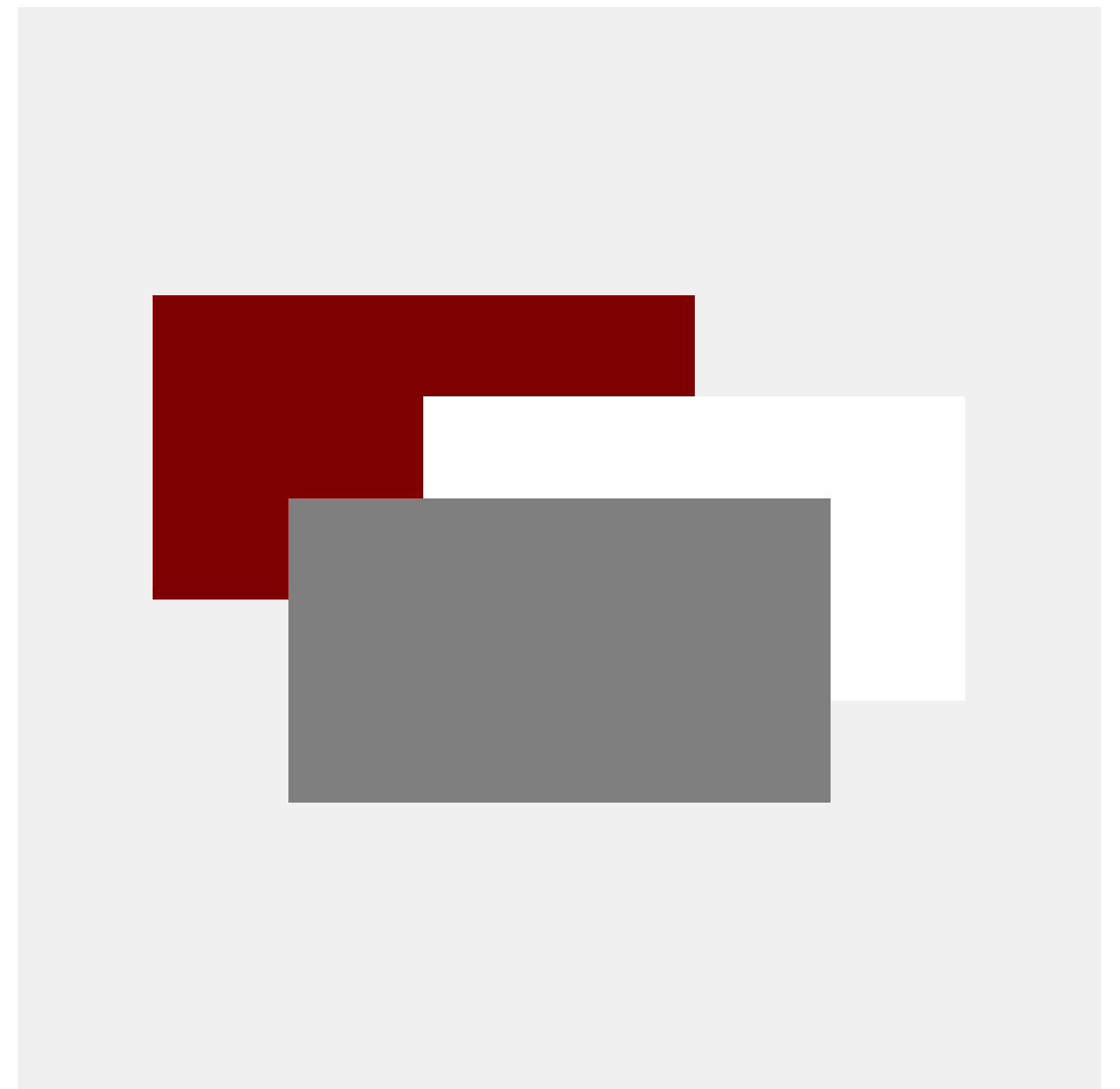


Deck



a Go package for presentations

DECK: a package for presentations

Deck is a package written in Go

That uses a singular markup language

With elements for text, lists, code, and graphics

All layout and sizes are expressed as percentages

Clients are interactive or create formats like PDF or SVG

Servers use a RESTful API to list, upload, stop, start, remove decks

Elements

text element

Hello, World (plain text)

A block of text, word-wrapped to a specified width. You may specify size, font, color, and opacity.

```
package main

import "fmt"

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

list element

- Point A
- Point B
- Point C
- Point D

First item

Second item

The third item

the last thing

1. This

2. That

3. The other

4. One more

```
<list xp="5" yp="70" sp="3"
  type="bullet"
  font="sans"
  color="rgb(0,127,0)">
<li>Point A</li>
<li>Point B</li>
<li>Point C</li>
<li>Point D</li>
</list>
```

```
<list xp="35" yp="70" sp="3"
  type="plain"
  font="serif"
  color="rgb(0,0,127)">
<li>First item</li>
<li>Second item</li>
<li>The third item</li>
<li>the last thing</li>
</list>
```

```
<list xp="70" yp="70" sp="3"
  type="number"
  font="mono"
  color="black">
<li>This</li>
<li>That</li>
<li>The other</li>
<li>One more</li>
</list>
```

image element

height

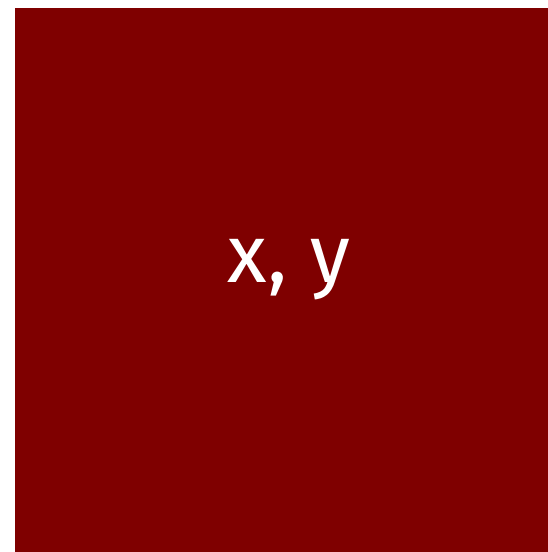


width

```
<image xp="50" yp="50" width="360" height="203" name="desert2.jpg"/>
```

rect element

height (relative
to element
or canvas
width)

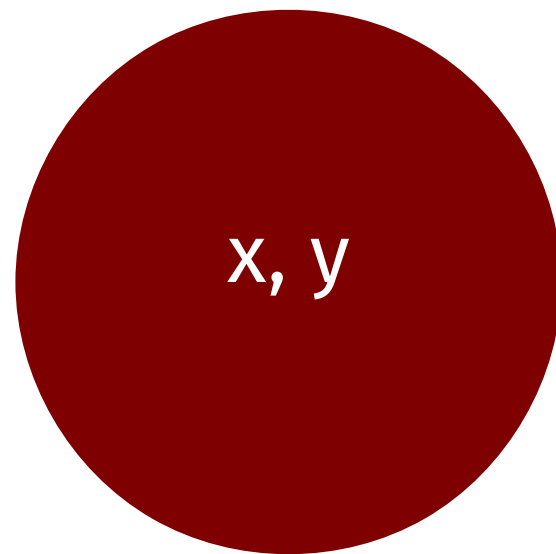


width

```
<rect xp="50" yp="50" wp="20" hr="100"/>
```

ellipse element

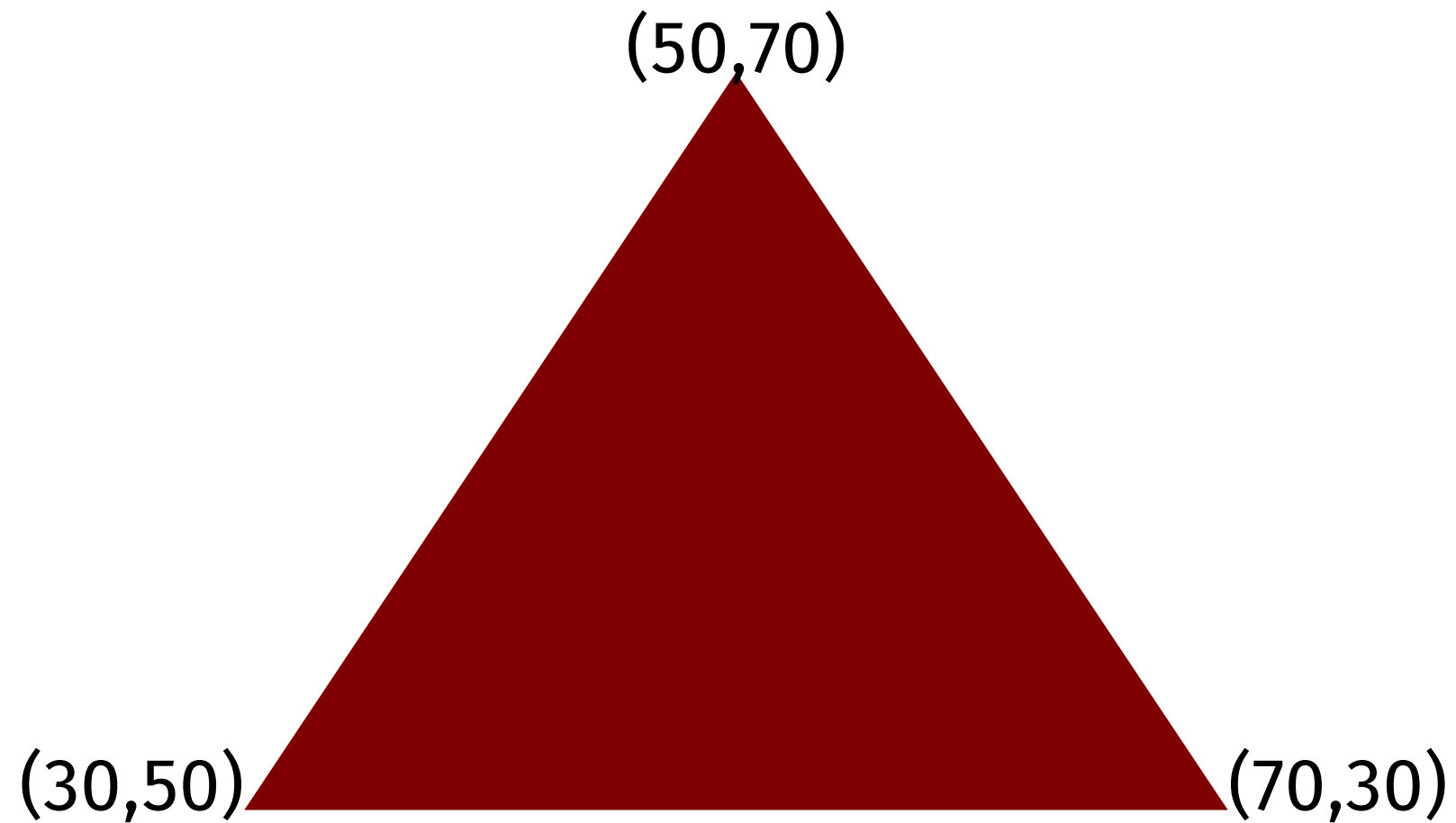
height (relative
to element
or canvas
width)



width

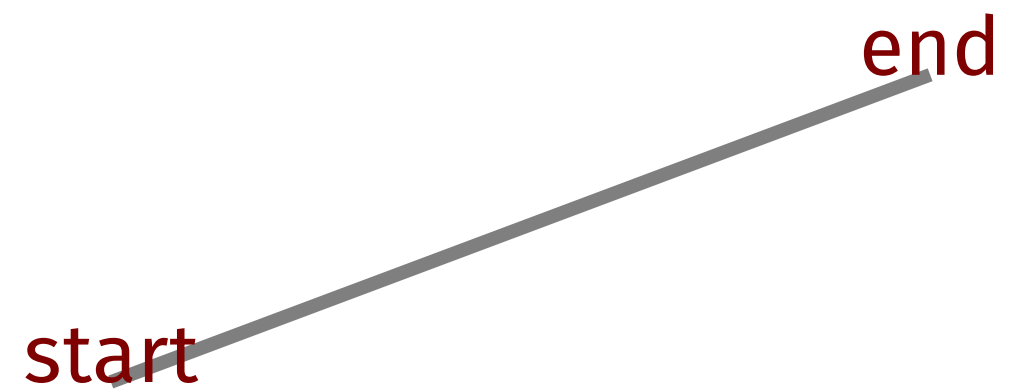
```
<ellipse xp="50" yp="50" wp="20" hr="100"/>
```


polygon element



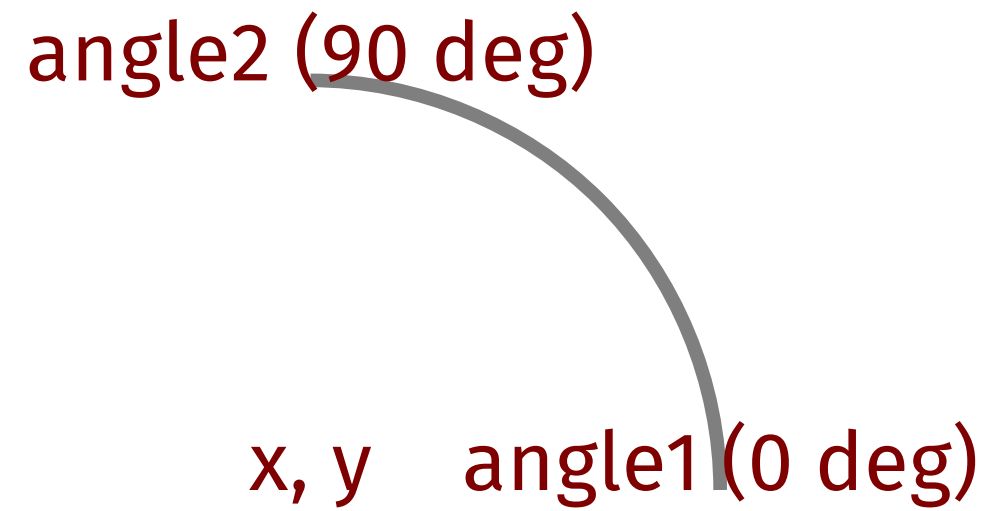
```
<polygon xc="30 50 70" yc="30 70 30"/>
```

line element



```
<line xp1="35" yp1="50" xp2="65" yp2="65"/>
```

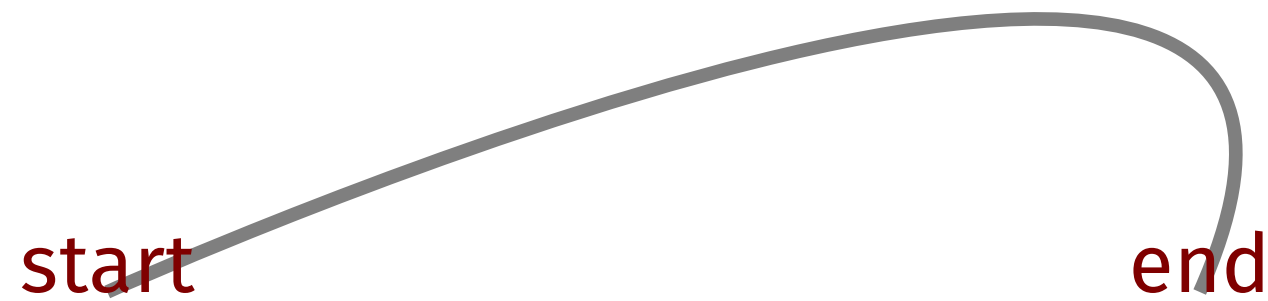
arc element



```
<arc xp="50" yp="50" wp="30" hp="30" a1="0" a2="90"/>
```

curve element

control



```
<curve xp1="30" yp1="50" xp2="80" yp2="80" xp3="70" yp3="50"/>
```

Markup and Layout

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
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



Desk



Text and List Markup

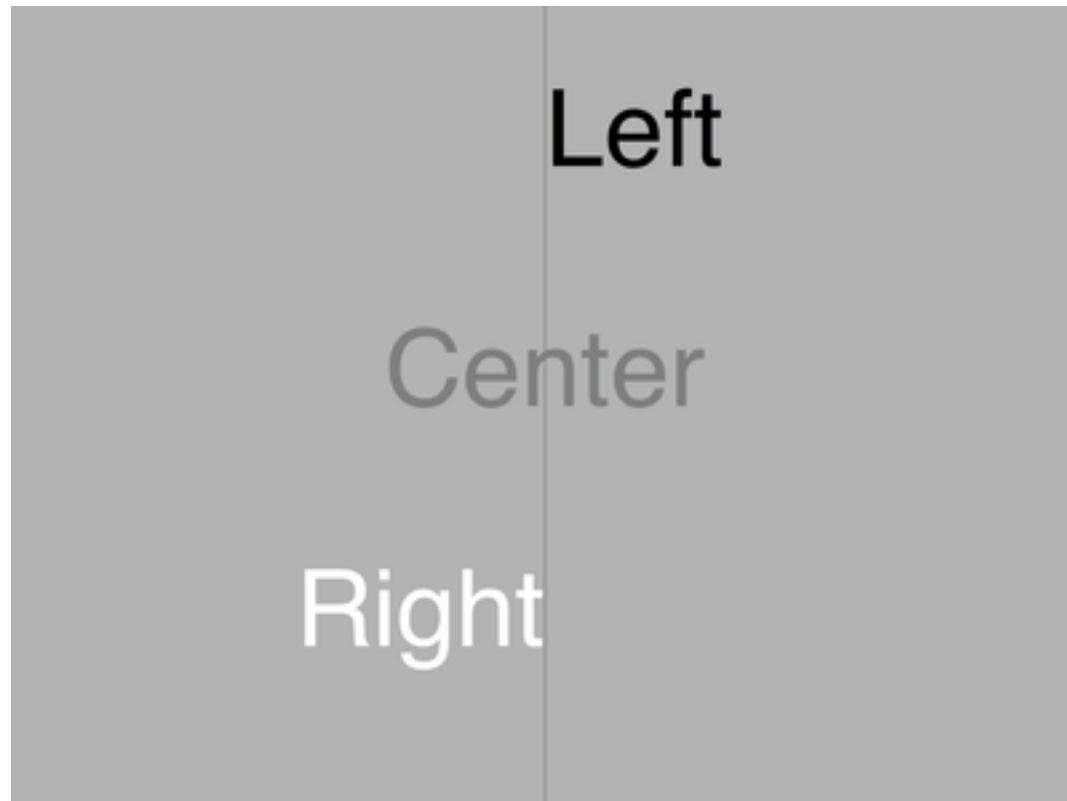
Position, size	<code><text xp="..." yp="..." sp="..."></code>
Block of text	<code><text ... type="block"></code>
Lines of code	<code><text ... type="code"></code>
Attributes	<code><text ... color="..." opacity="..." font="..." align="..." link="..."></code>

Position, size	<code><list xp="..." yp="..." sp="..."></code>
Bullet list	<code><list ... type="bullet"></code>
Numbered list	<code><list ... type="number"></code>
Attributes	<code><list ... color="..." opacity="..." font="..." align="..." link="..."></code>

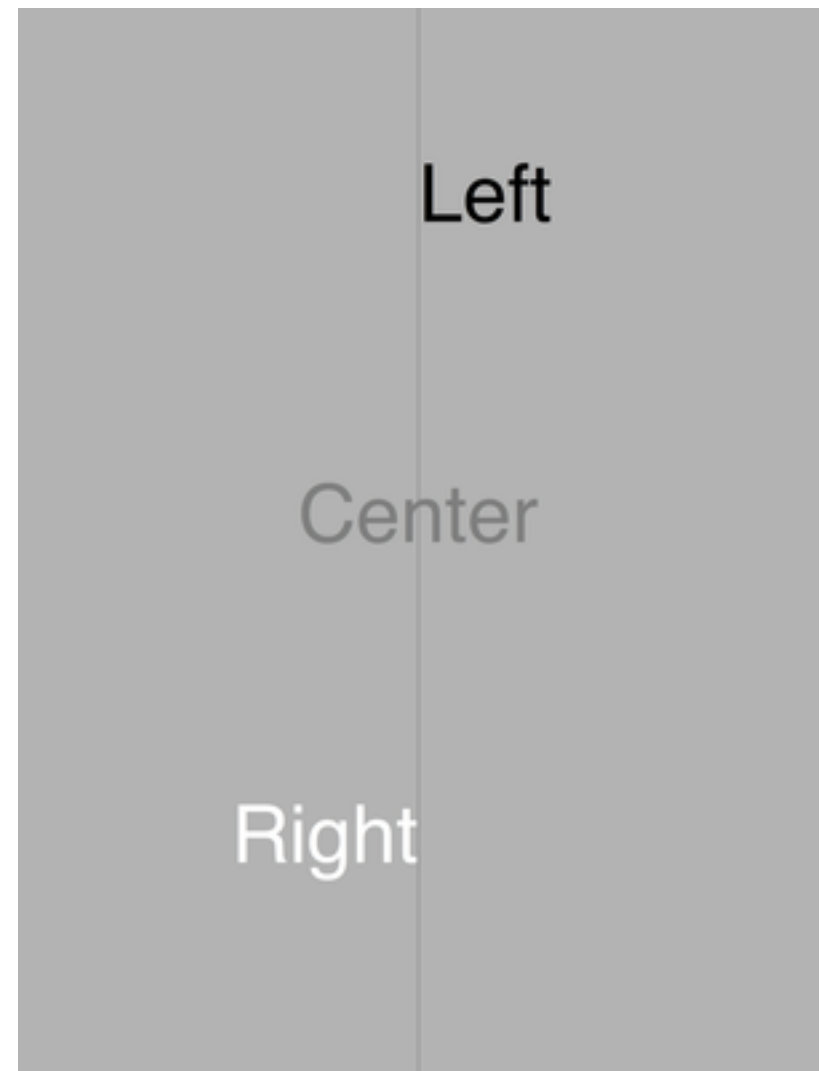
Common Attributes for text and list

xp	horizontal percentage
yp	vertical percentage
sp	font size percentage
type	"bullet", "number" (list), "block", "code" (text)
align	"left", "middle", "end"
color	SVG names ("maroon"), or RGB "rgb(127,0,0)"
opacity	percent opacity (0-100, transparent - opaque)
font	"sans", "serif", "mono"
link	URL

Scaling the canvas



Landscape



Portrait

[illegible]

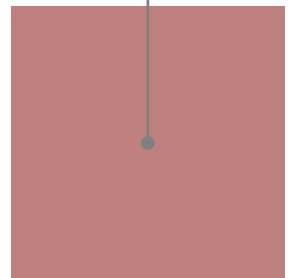
10%, 50%

Hello

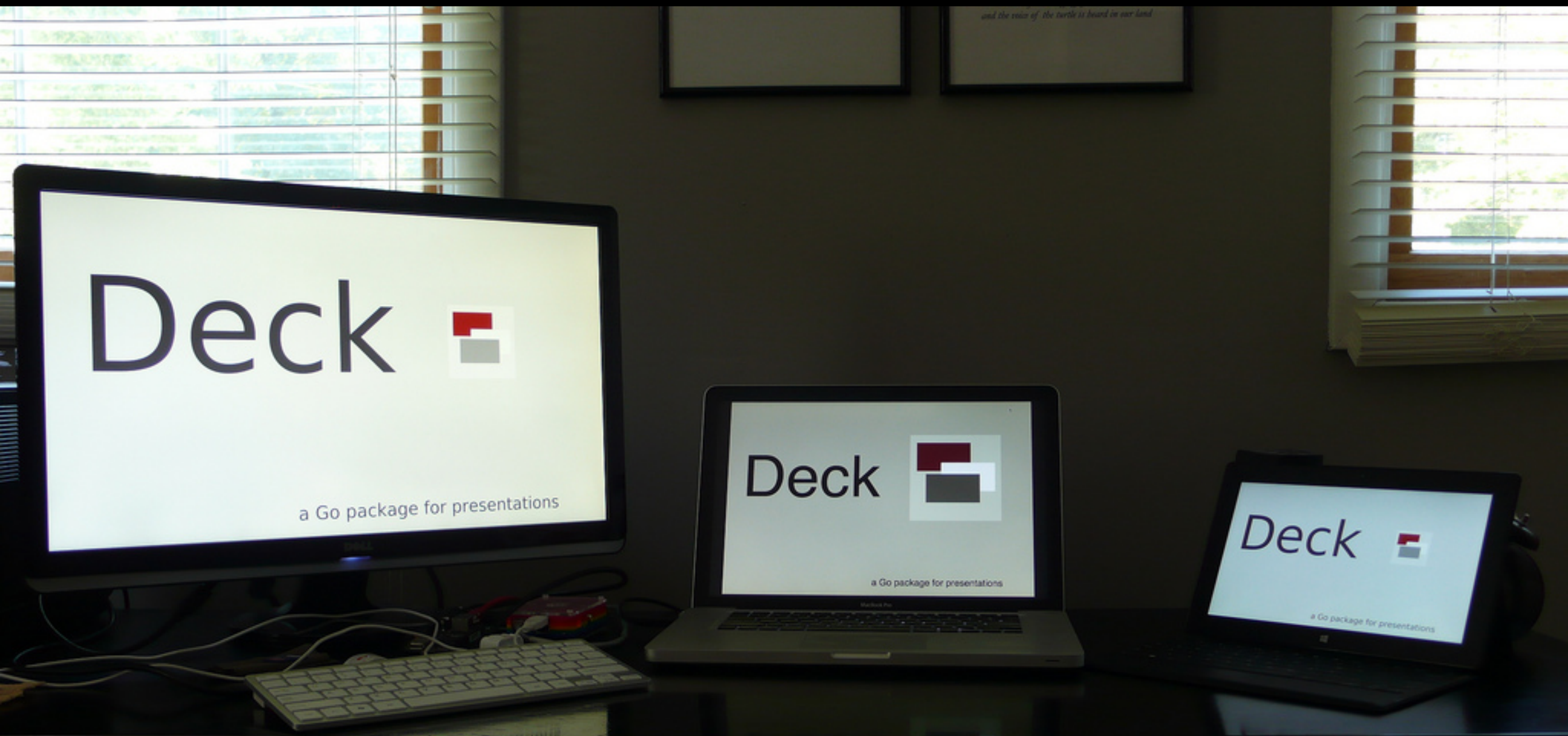
50%, 50%



90%, 50%



Percentage-based layout



Clients

```
package main

import (
    "github.com/ajstarks/deck"
    "log"
)

func main() {
    presentation, err := deck.Read("deck.xml", 1024, 768) // open the deck
    if err != nil {
        log.Fatal(err)
    }
    for _, slide := range presentation.Slide { // for every slide...
        for _, t := range slide.Text { // process the text elements
            x, y, size := deck.Dimen(presentation.Canvas, t.Xp, t.Yp, t.Sp)
            slideText(x, y, size, t)
        }
        for _, l := range slide.List { // process the list elements
            x, y, size := deck.Dimen(presentation.Canvas, l.Xp, l.Yp, l.Sp)
            slideList(x, y, size, l)
        }
    }
}
```

A Deck Client

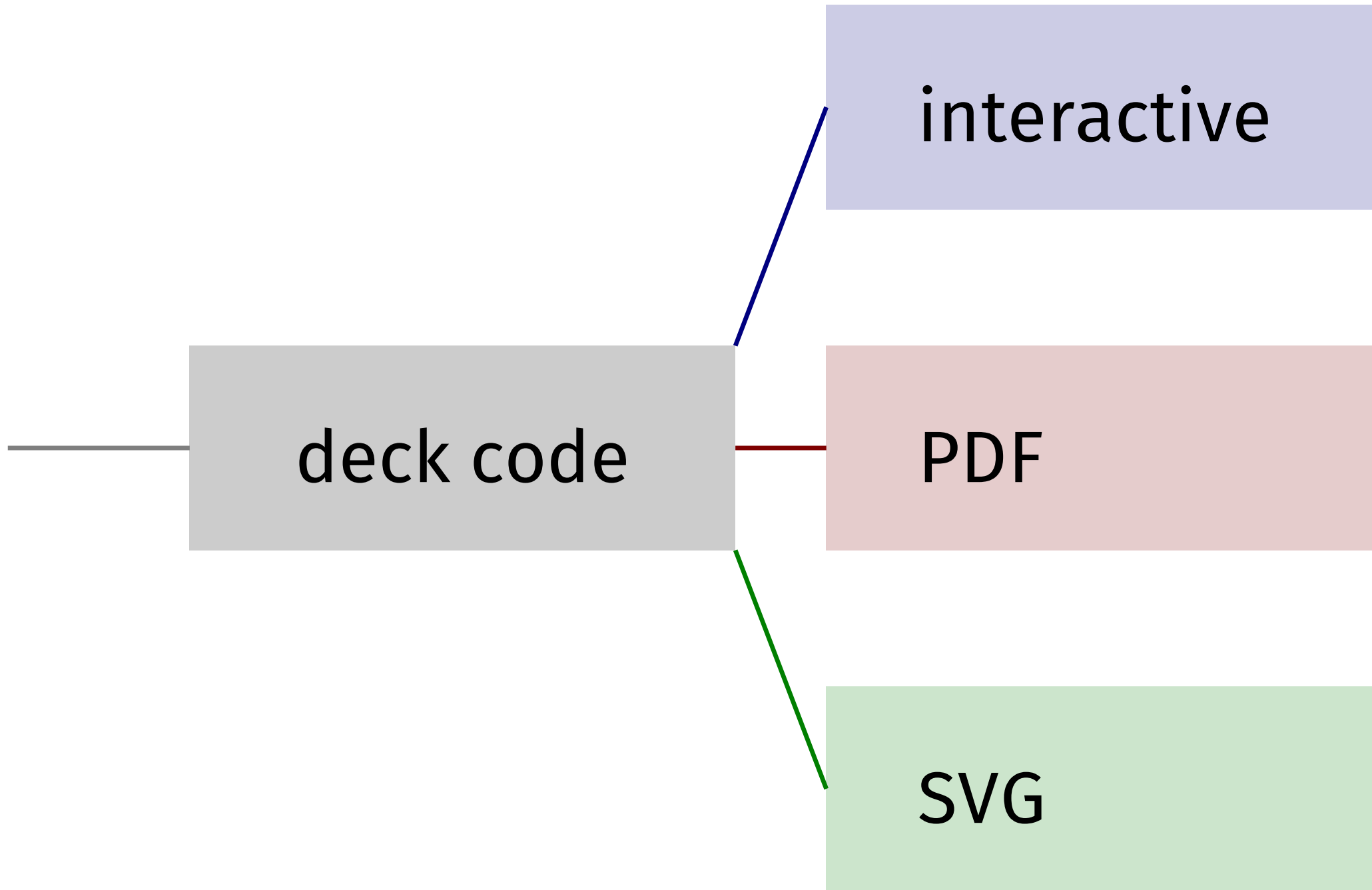
Process

deck code

interactive

PDF

SVG



Generating a Barchart

```
package main

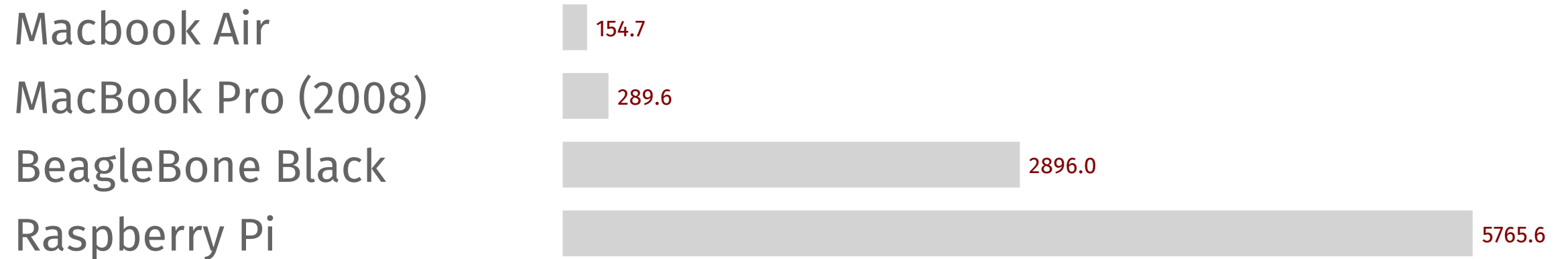
import (
    "fmt"
    "github.com/ajstarks/deck/generate"
    "os"
)

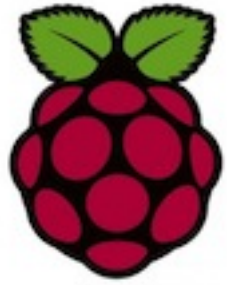
type Bardata struct {
    label string
    value float64
}

func vmap(value float64, low1 float64, high1 float64, low2 float64, high2 float64) float64 {
    return low2 + (high2-low2)*(value-low1)/(high1-low1)
}

func main() {
    benchmarks := []Bardata{
        {"Macbook Air", 154.701}, {"MacBook Pro (2008)", 289.603}, {"BeagleBone Black", 2896.037}, {"Raspberry Pi", 5765.568},
    }
    maxdata := 5800.0
    ts := 2.5
    hts := ts / 2
    x, y := 10.0, 60.0
    bx1 := x + (ts * 12)
    bx2 := bx1 + 50.0
    linespacing := ts * 2.0
    deck := generate.NewSlides(os.Stdout, 0, 0)
    deck.StartDeck()
    deck.StartSlide("rgb(255,255,255)")
    deck.Text(x, y+20, "Go 1.1.2 Build and Test Times", "sans", ts*2, "black")
    for _, data := range benchmarks {
        deck.Text(x, y, data.label, "sans", ts, "rgb(100,100,100)")
        bv := vmap(data.value, 0, maxdata, bx1, bx2)
        deck.Line(bx1, y+hts, bv, y+hts, ts, "lightgray")
        deck.Text(bv+0.5, y+(hts/2), fmt.Sprintf("%.1f", data.value), "sans", hts, "rgb(127,0,0)")
        y -= linespacing
    }
    deck.EndSlide()
    deck.EndDeck()
}
```


Go 1.1.2 Build and Test Times





```
go get github.com/ajstarks/deck/cmd/vgdeck
```



```
go get github.com/ajstarks/deck/cmd/pdfdeck
```



```
go get github.com/ajstarks/deck/cmd/svgdeck
```

pdfdeck [options] file.xml...

- sans, -serif, -mono [font] specify fonts
- pagesize [w,h, or Letter, Legal, Tabloid, A2-A5, ArchA, Index, 4R, Widescreen]
- stdout (output to standard out)
- outdir [directory] directory for PDF output
- fontdir [directory] directory containing font information
- author [author name] set the document author
- title [title text] set the document title
- grid [percent] draw a percent grid on each slide

svgdeck [options] file.xml...

- sans, -serif, -mono [font] specify fonts
- pagesize [Letter, Legal, A3, A4, A5]
- pagewidth [canvas width]
- pageheight [canvas height]
- stdout (output to standard out)
- outdir [directory] directory for PDF output
- title [title text] set the document title
- grid [percent] draw a percent grid on each slide

vgdeck [options] file.xml...

- loop [duration] loop, pausing [duration] between slides
- slide [number] start at slide number
- w [width] canvas width
- h [height] canvas height
- g [percent] draw a percent grid

vgdeck Commands

`+`, `Ctrl-N`, `[Return]`

Next slide

`-`, `Ctrl-P`, `[Backspace]`

Previous slide

`^`, `Ctrl-A`

First slide

`$`, `Ctrl-E`

Last slide

`r`, `Ctrl-R`

Reload

`x`, `Ctrl-X`

X-Ray

`/`, `Ctrl-F` `[text]`

Search

`s`, `Ctrl-S`

Save

`q`

Quit

Deck Web API

```
sex -dir [start dir] -listen [address:port] -maxupload [bytes]
```

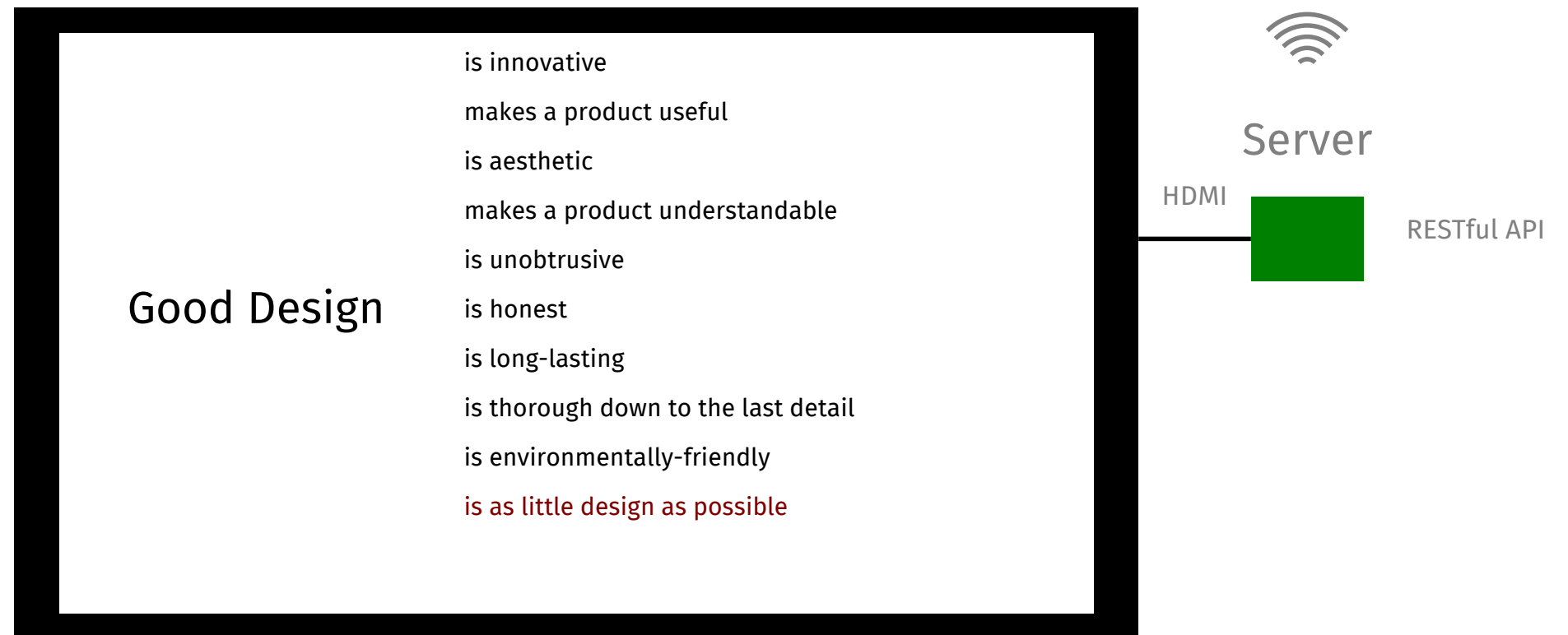
GET	/	List the API
GET	/deck/	List the content on the server
GET	/deck/?filter=[type]	List content filtered by deck, image, video
POST	/deck/content.xml?cmd=1s	Play a deck with the specified duration
POST	/deck/content.xml?cmd=stop	Stop playing a deck
POST	/deck/content.xml?slide=[num]	Play deck starting at a slide number
DELETE	/deck/content.xml	Remove content
POST	/upload/ Deck:content.xml	Upload content
POST	/table/ Deck:content.txt	Generate a table from a tab-separated list
POST	/table/?textsize=[size]	Specify the text size of the table
POST	/media/ Media:content.mov	Play the specified video

deck [command] [argument]

deck play file [duration]	Play a deck
deck stop	Stop playing a deck
deck list [deck image video]	List contents
deck upload file...	Upload content
deck remove file...	Remove content
deck video file	Play video
deck table file [textsize]	Make a table

\$ deck upload *.jpg	# upload images
\$ mkpicdeck *.jpg deck upload /dev/stdin	# generate the slide show deck
\$ deck play stdin	# play it

Display



Controller



Design Examples

A large, solid blue semi-circle is positioned in the lower half of the image, set against a dark gray background. The semi-circle's flat edge is at the bottom, and its curved edge is at the top.

hello, world

Top

Left

Right

Bottom

20%

30%

70%

20%

Header (top 20%)

Summary
(30%)

Detail
(70%)

Footer (bottom 20%)

bullet

- Point A
- Point B
- Point C
- Point D

plain

First item
Second item
The third item
the last thing

number

1. This
2. That
3. The other
4. One more

```
<list>...</list>
```

BOS



Virgin America 351

Gate B38

8:35am

SFO

On Time

JFK



IND

US Airways 1207

Gate C31C

5:35pm

Delayed

AAPL	503.73	-16.57 (3.18%)
------	--------	----------------

AMZN	274.03	+6.09 (2.27%)
------	--------	---------------

GOOG	727.58	-12.41 (1.68%)
------	--------	----------------

Two Columns

One

Two

Three

Four



Tree and Sky

Five

Six

Seven

Eight



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 gofmt on package sources
-----	------------------------------

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

Code

```
package main

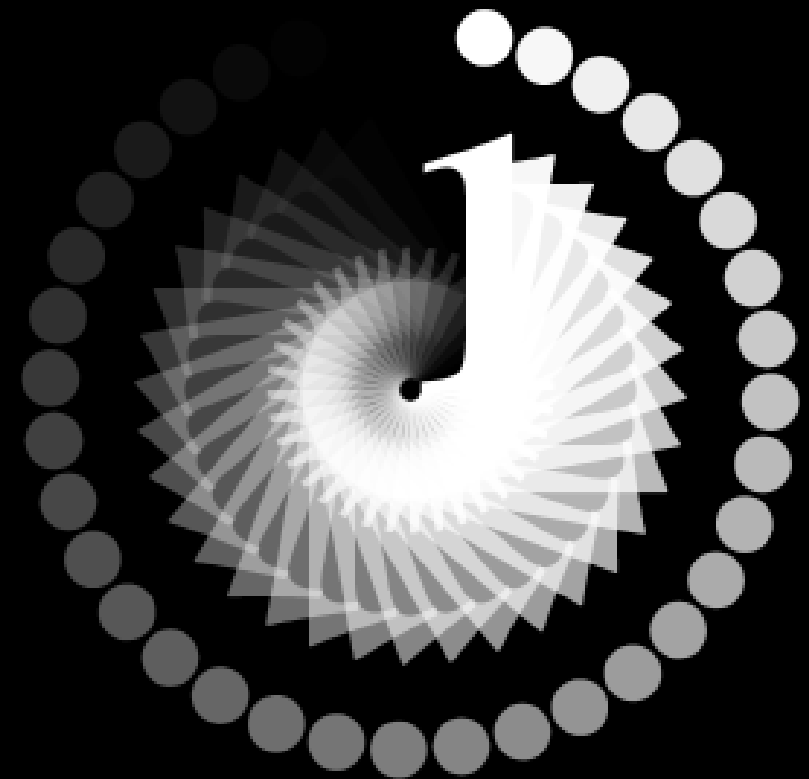
import (
    "github.com/ajstarks/svgo"
    "os"
)

func main() {
    canvas := svgo.New(os.Stdout)
    width, height := 500, 500
    a, ai, ti := 1.0, 0.03, 10.0

    canvas.Start(width, height)
    canvas.Rect(0, 0, width, height)
    canvas.Gstyle("font-family:serif;font-size:144pt")

    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()
}
```

Output



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, <http://pocket.co/sSc56>



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

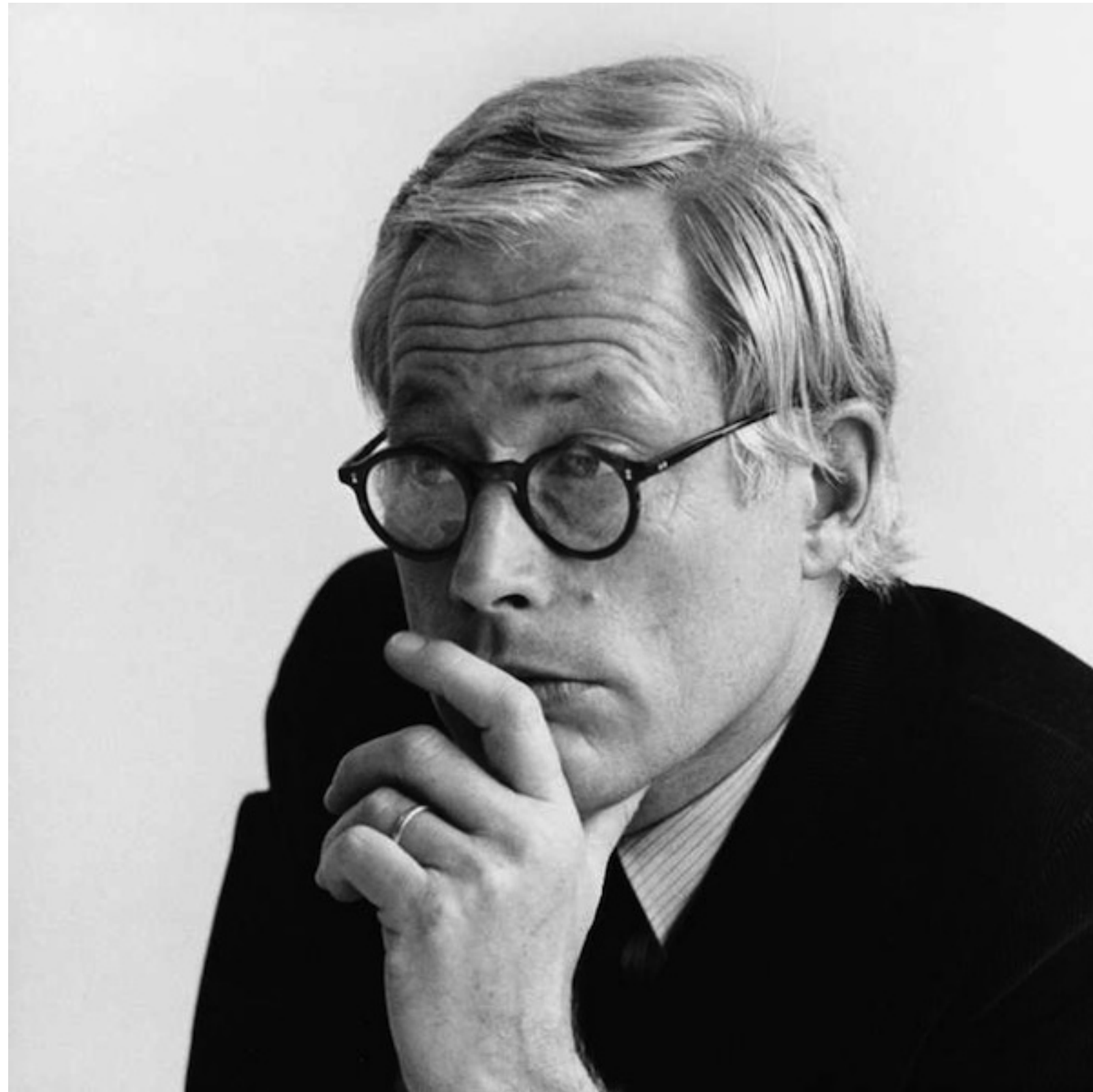
is honest

is long-lasting

is thorough down to the last detail

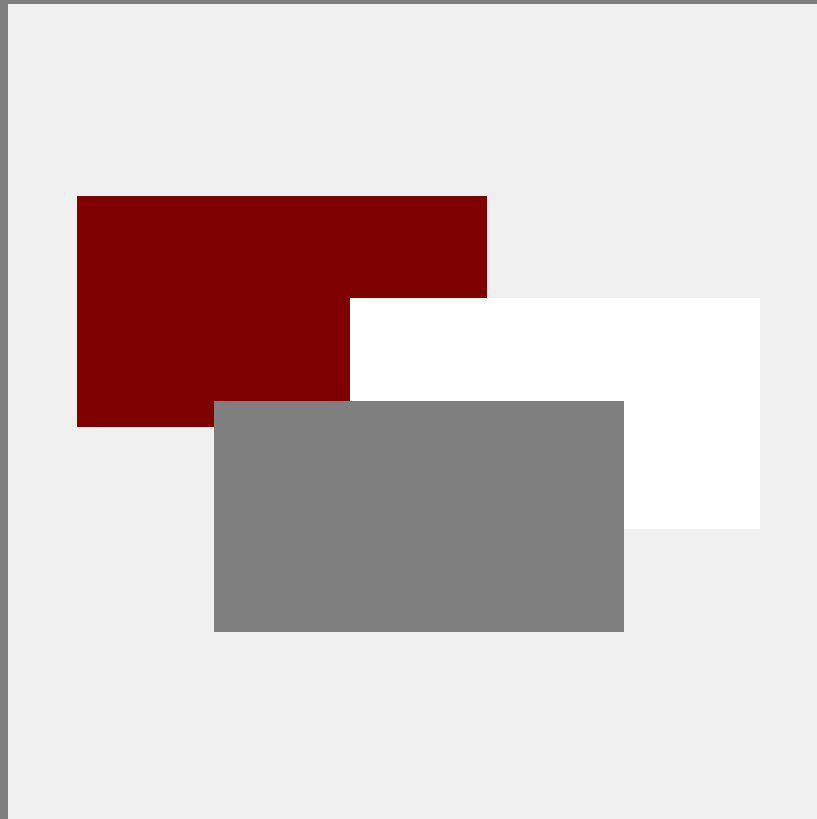
is environmentally-friendly

is as little design as possible



Dieter Rams

github.com/ajstarks/deck



ajstarks@gmail.com

@ajstarks