MONASH University

Advanced Data Visualisation with R

R Graphics using $grid$

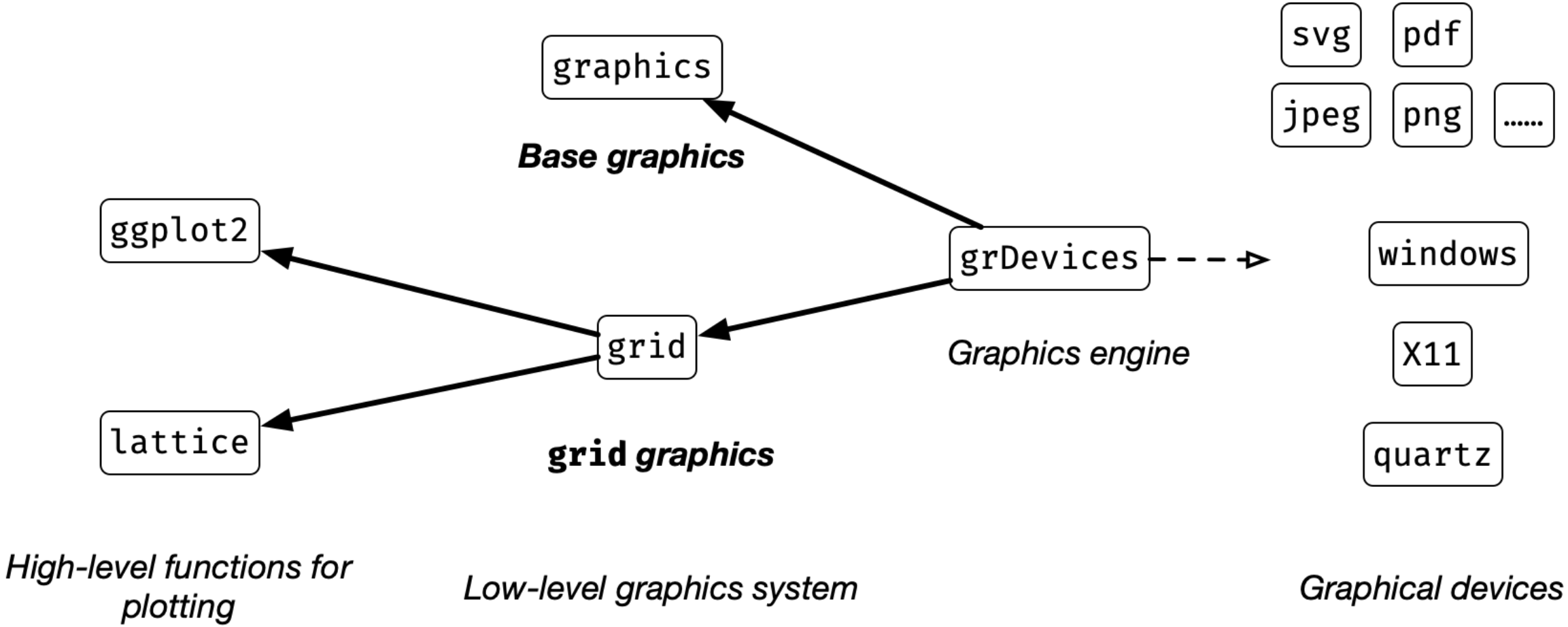
Instructor: *Emi Tanaka*

emi.tanaka@monash.edu

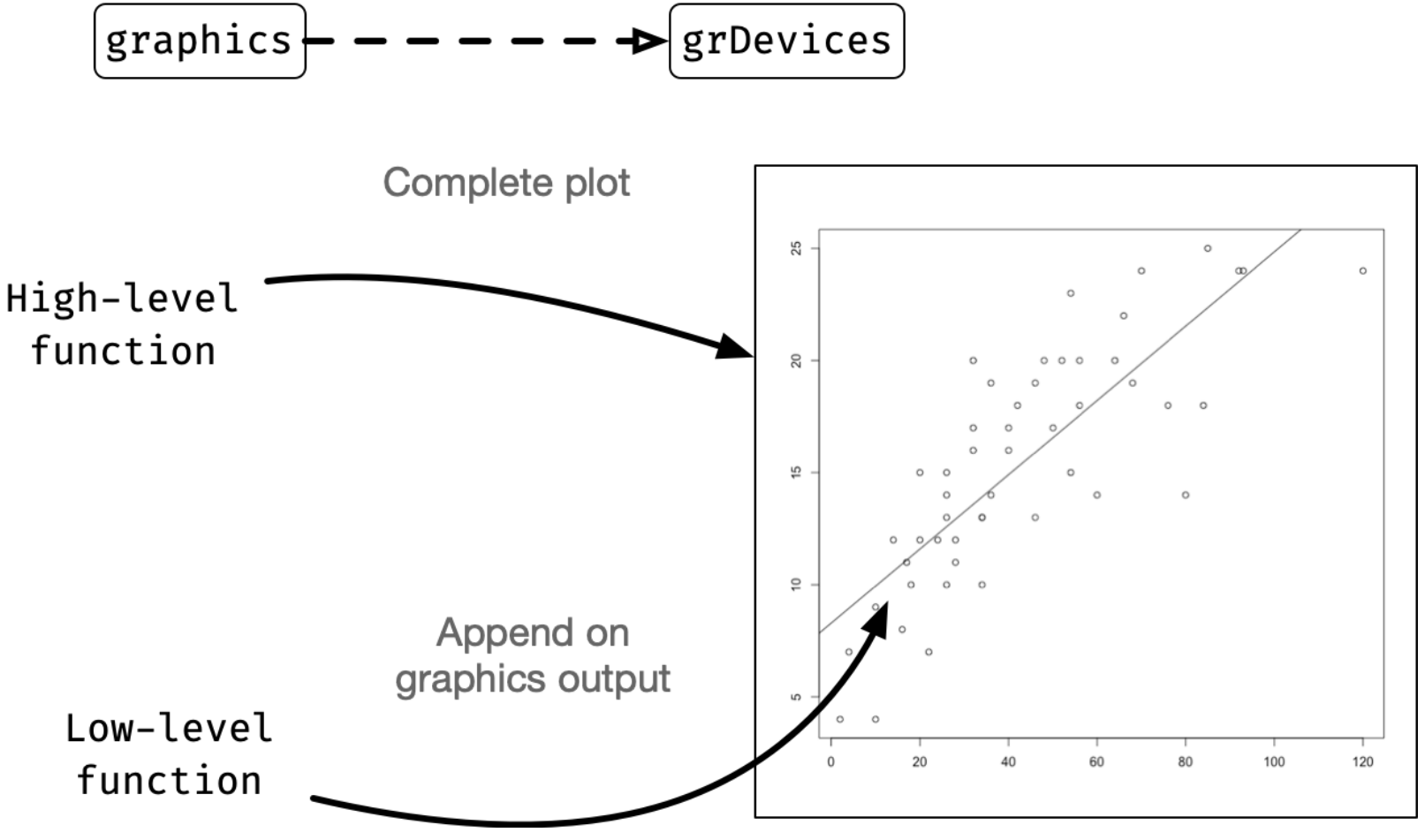
8th Dec 2021 @ Statistical Society of Australia Canberra Branch | Zoom

Murrell (2019) R Graphics. CRC Press.

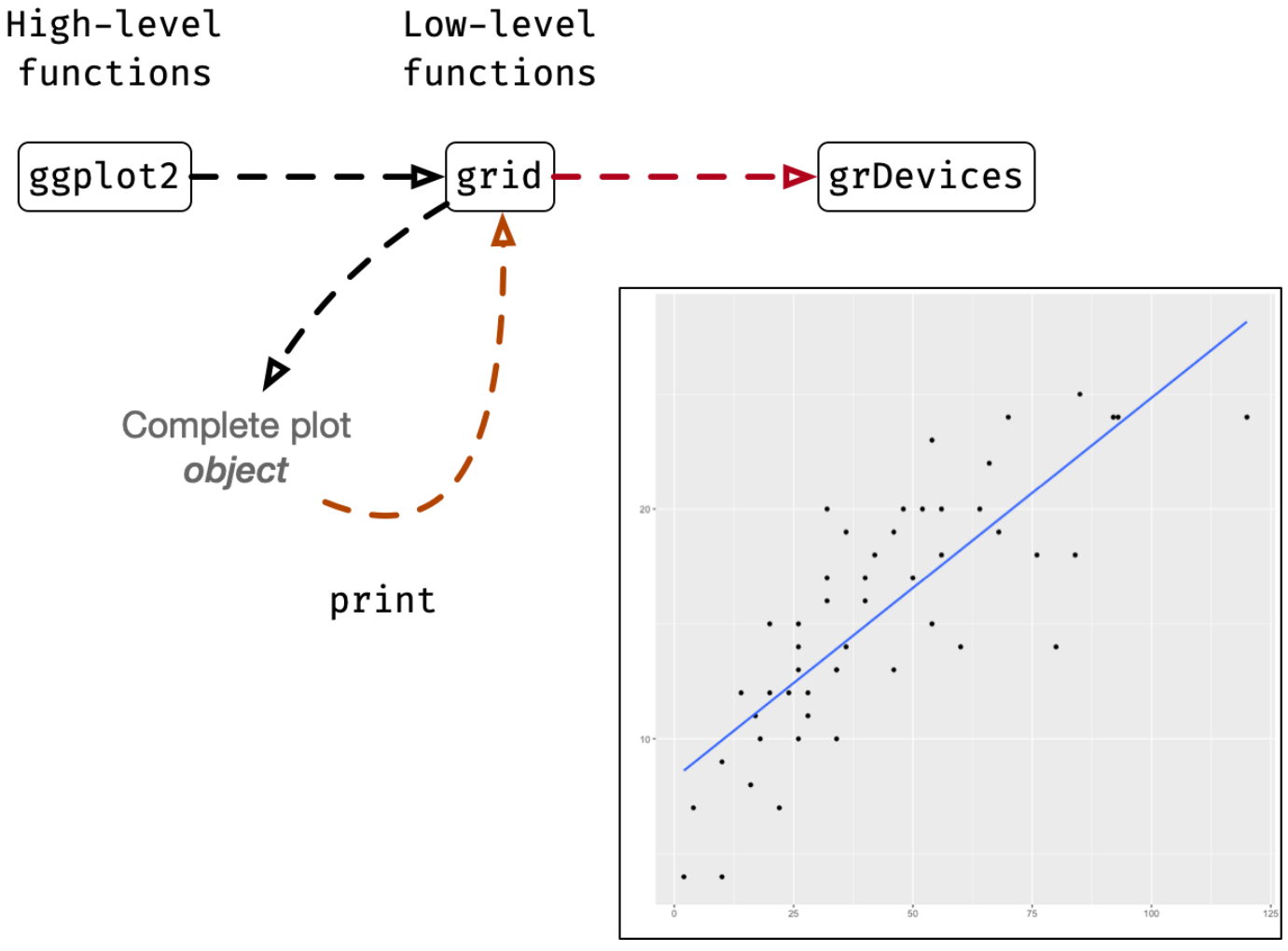
R graphics system



Base graphics



ggplot graphics



Graphics model in R

- There are two main **graphics models** in R

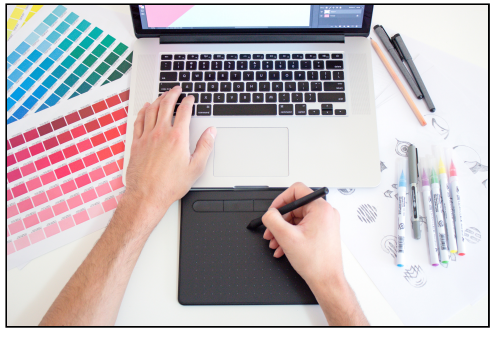
Base `graphics`

`library(graphics)`

(typically already loaded)

`grid` graphics

`library(grid)`



`grid` R-package

- The `grid` package contains the low-level functions to create, manipulate and draw **gr**aphical **ob**jects (grob).
- The `grid` package is a **base package** and so is included in standard R installations

```
packageDescription("grid")
```

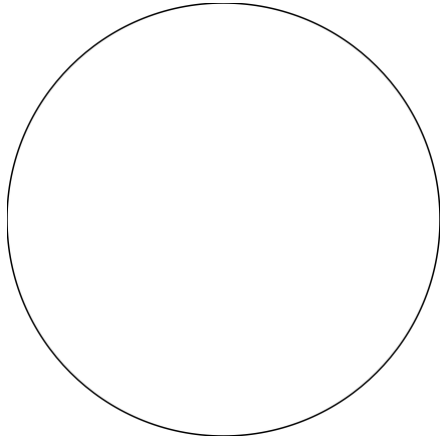
```
Package: grid
Version: 4.1.2
Priority: base
Title: The Grid Graphics Package
Author: Paul Murrell <paul@stat.auckland.ac.nz>
Maintainer: R Core Team <do-use-Contact-address@r-project.org>
Contact: R-help mailing list <r-help@r-project.org>
Description: A rewrite of the graphics layout capabilities, plus some
        support for interaction.
Imports: grDevices, utils
License: Part of R 4.1.2
NeedsCompilation: yes
Built: R 4.1.2; x86_64-apple-darwin17.0; 2021-11-01 20:58:55 UTC; unix
```

Graphical objects a.k.a. "Grobs"

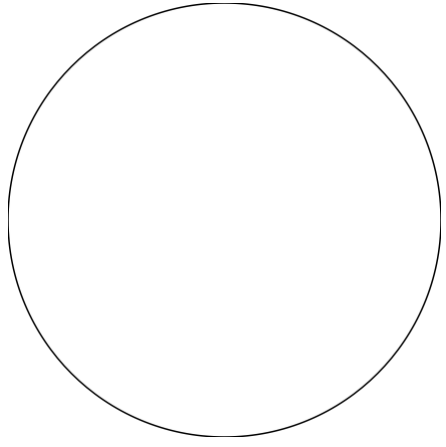
- The base packages `stats`, `utils`, `datasets`, `methods`, `base`, `graphics` and **`grDevices`** are automatically loaded in a standard launch of R.

```
library(grid)
```

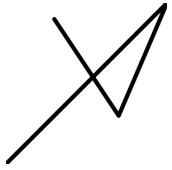
```
grid.circle()
```



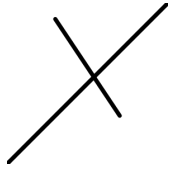
```
circleGrob()

## circle[GRID.circle.94]

grid.draw(circleGrob())
```



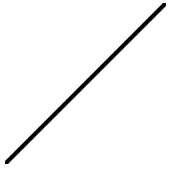
Graphical primitives in `grid`



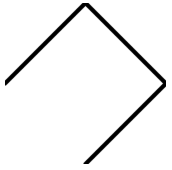
`grid.lines()`
`linesGrob()`



`grid.polyline()`
`polylineGrob()`



`grid.segments()`
`segmentsGrob()`

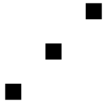


`grid.xspline()`
`xsplineGrob()`

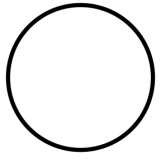
`grid.bezier()`
`bezierGrob()`

Label

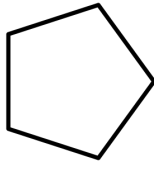
`grid.text()`
`textGrob()`



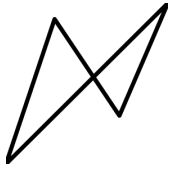
`grid.raster()`
`rasterGrob()`



`grid.circle()`
`circleGrob()`

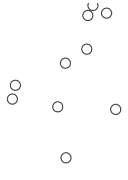


`grid.polygon()`
`polygonGrob()`

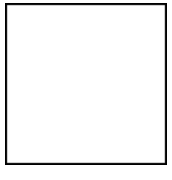


`grid.path()`
`pathGrob()`

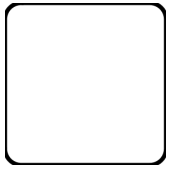
`grid.curve()`
`curveGrob()`



`grid.points()`
`pointsGrob()`



`grid.rect()`
`rectGrob()`



`grid.roundrect()`
`roundrectGrob()`

Convention of function names:

`grid.primitive()`
`primitiveGrob()`

Higher-level functions

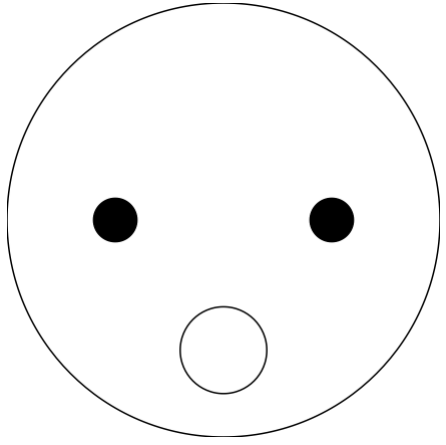
- Most functions in `grid` are low-level functions but *selected number of higher-level functions exist*
- The output of these functions combine multiple graphical primitives

```
pushViewport(plotViewport())
pushViewport(viewport(xscale = c(0, 100)))
grid.xaxis()
```

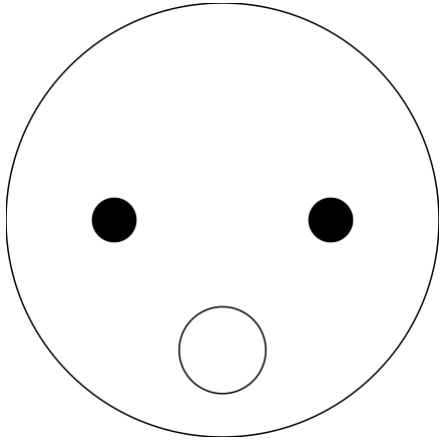
Making your own grobs from graphical primitive

```
shape <- circleGrob(x = 0.5, y = 0.5, r = 0.5, name = "shape")
eyes <- circleGrob(x = c(0.25, 0.75), y = 0.5, r = 0.05,
                   gp = gpar(fill = "black"), name = "eyes")
mouth <- circleGrob(x = 0.5, y = 0.2, r = 0.1, name = "mouth")

face1 <- gList(shape, eyes, mouth)
face2 <- grobTree(shape, eyes, mouth, name = "face")
```

```
grid.draw(face1)
```



```
grid.draw(face2)
```



```
grid.ls(face1)
```

```
## shape
## eyes
## mouth
```

```
grid.ls(face2)
```

```
## face
##   shape
##   eyes
##   mouth
```

```
class(face1)
```

```
## [1] "gList"
```

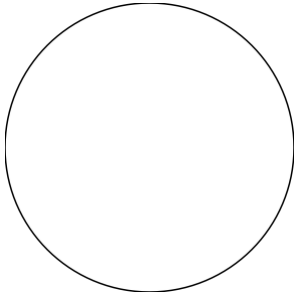
```
class(face2)
```

```
## [1] "gTree" "grob"  "gDesc"
```

Basic creators of grobs

- Drawing details for graphical primitives are written in C

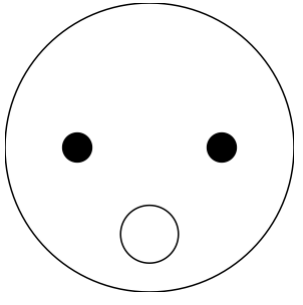
```
g1 <- grob(x = unit(0.5, "npc"),
           y = unit(0.5, "npc"),
           r = unit(0.5, "npc"),
           cl = "circle")
grid.draw(g1)
```



- You likely won't ever use this approach

- Creating a flexible tree structure with grobs

```
g2 <- gTree(children = gList(shape,
                             eyes,
                             mouth))
grid.draw(g2)
```



- Note: `grobTree(x)` is essentially `gTree(children = x)`

Working with graphical outputs

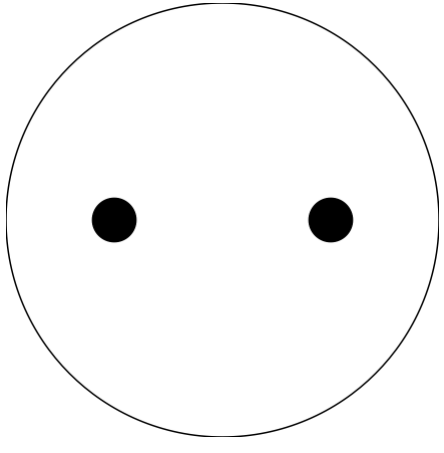
Work with output	Work with grobs	Description
`grid.get()`	`getGrob()`	Return a copy of grobs
`grid.edit()`	`editGrob()`	Modifies grobs
`grid.add()`	`addGrob()`	Add a grob
`grid.remove()`	`removeGrob()`	Remove grobs
`grid.set()`	`setGrob()`	Replace grobs

Along with some other helpful functions like :

- `grid.newpage()` to erase current device or move to a new page
- `grid.grep()` to find all matching grobs

Working with graphical outputs in action

```
grid.ls(face2)

## face
##   shape
##   eyes
##   mouth

grid.draw(removeGrob(face2, "mouth"))
```



```
xaxis <- xaxisGrob()

grid.ls(xaxis)

## GRID.xaxis.99
```

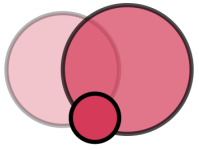
- This should include primitives?
- The `xaxis` contains no children
- This is because the tick marks are only drawn when signalled to draw

```
xaxisf <- grid.force(xaxis)
grid.ls(xaxisf)

## GRID.xaxis.99
##   major
##   ticks
##   labels
```

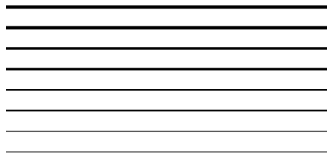
Graphical parameters in `grid`

`col`
color

`fill`
fill

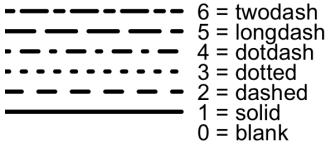


`alpha`
opacity

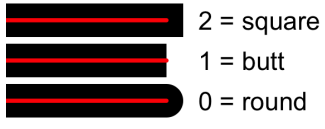


`lwd`
line width

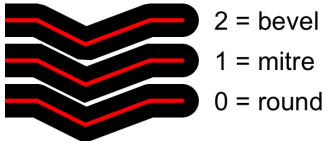
`lex`
line width expansion



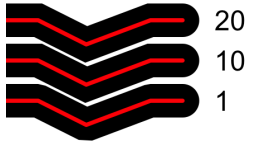
`lty`
line type



`lineend`
line end style



`linejoin`
line join style



`linemitre`
line mitre limit

All graphical primitives have a `gp` argument to parse with `grid::gpar()`

abcdef
abcdef
abcdef

`cex`
character expansion

abcdef
abcdef
abcdef

`fontsize`
font size

abcdef 4 = "bold.italic"
abcdef 3 = "italic"
abcdef 2 = "bold"
abcdef 1 = "plain"

`fontface`
font face

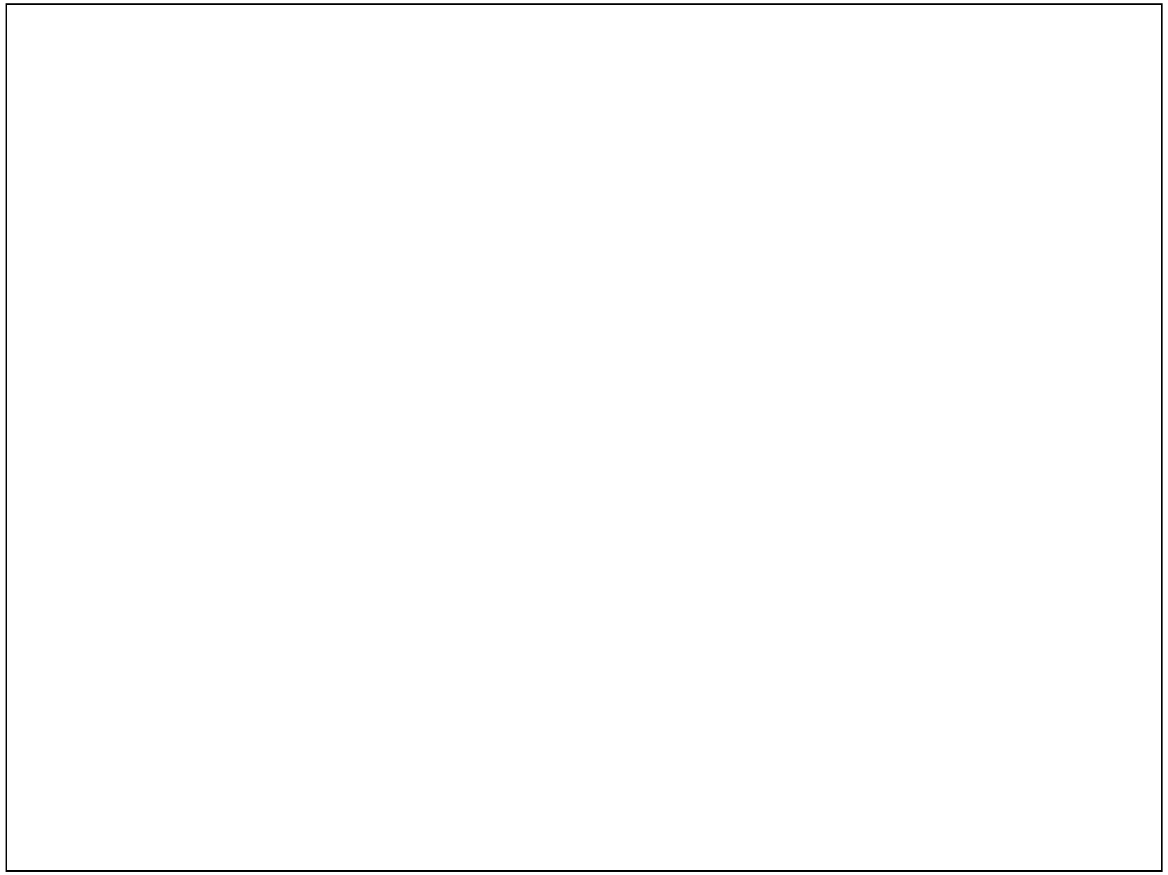
`abcdef` "mono"
abcdef "sans"
abcdef "serif"

`fontfamily`
font family

line1 line1 line1
line2 line2 line2
line3 line3 line3

`lineheight`
line height

The canvas

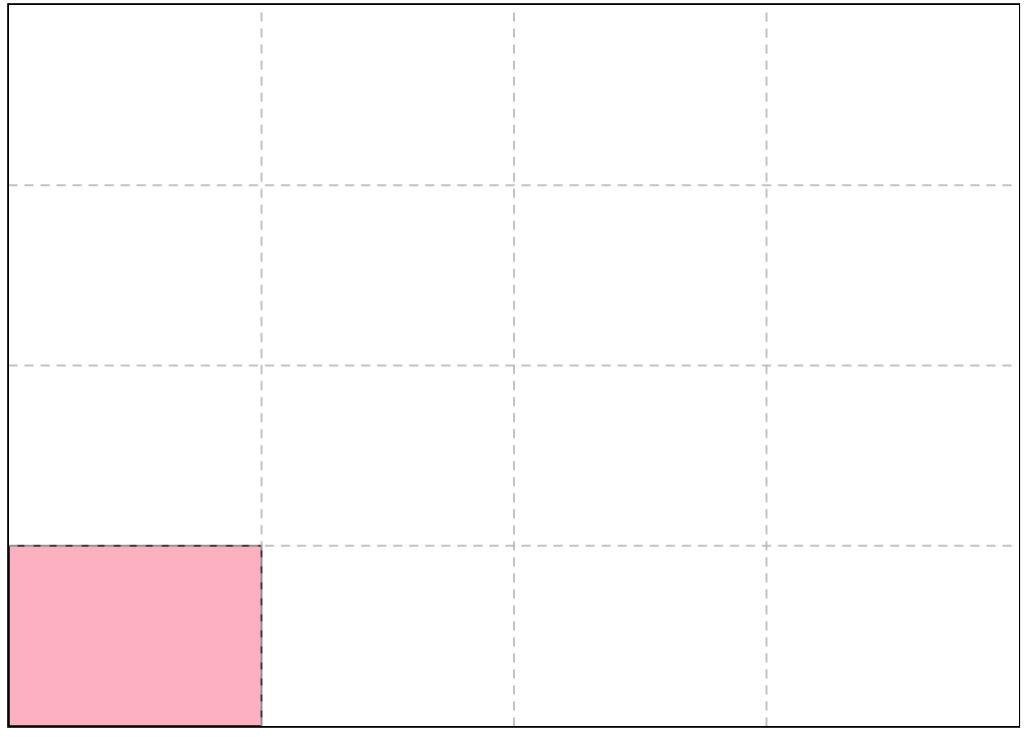


Coordinate systems

Coordinate system	Description
`"native"`	Relative to the scales of the current viewport
`"npc"`	Normalised parent coordinates
`"snpc"`	Square normalised parent coordinates
`"in"`, `"cm"`, `"mm"`	Physical inches, centimeters, millimeters
`"pt"`, `"bigpts"`, `"picas"`, `"dida"`, `"cicero"`, `"scaledpts"`	72.27 points = 1 inch, 72 big points = 1 inch, 12 points = 1 pica, 1157 dida = 1248 points, 1 cicero = 12 dida, 65536 scaled points = 1 point
`"char"`	Multiples of the current font size (`fontsize` and `cex`)
`"line"`	Multiples of the height of a text line (`fontsize`, `cex` and `lineheight`)
`"strwidth"`, `"strheight"`	Multiples of the width/height of a given string (`fontsize`, `cex`, `fontfamily`, and `fontface`)
`"grobx"`, `"groby"`	Multiples of the x- and y-location on the boundary of a given grob
`"grobwidth"`, `"grobheight"`	Multiples of the width/height of a given grob

Drawing with `grid` 1

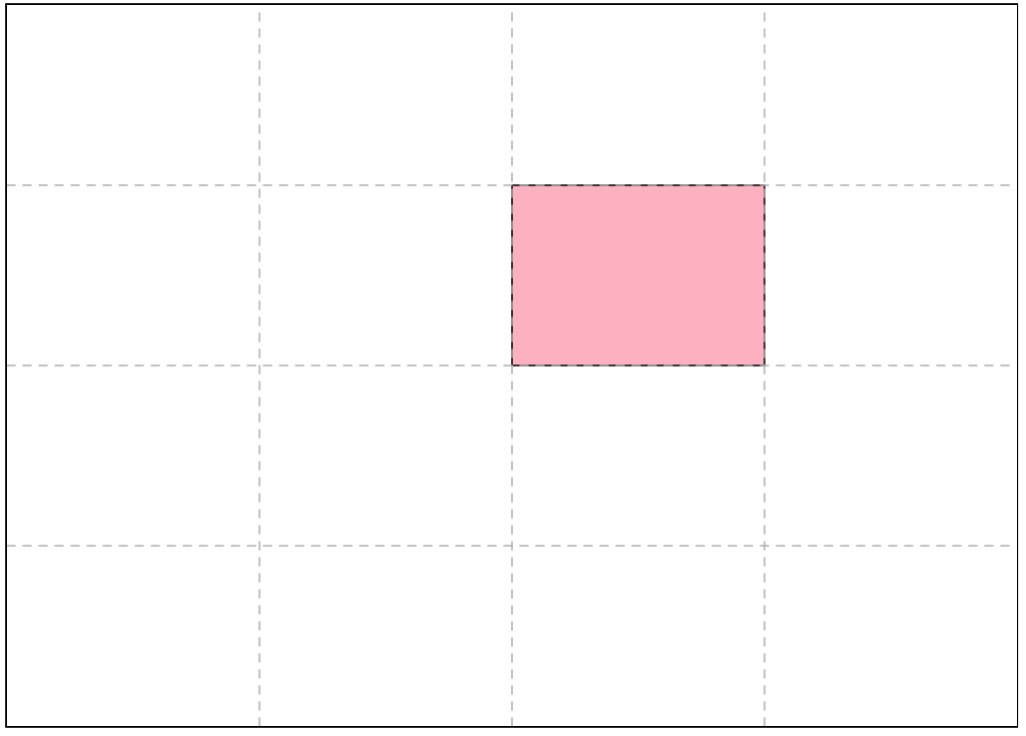
```
grid.rect(x = 0,
          y = 0,
          width = 0.25,
          height = 0.25,
          default.units = "npc",
          just = c("left", "bottom"),
          gp = gpar(fill = "pink"),
          vp = NULL)
```



- The grid lines are drawn for your convenience

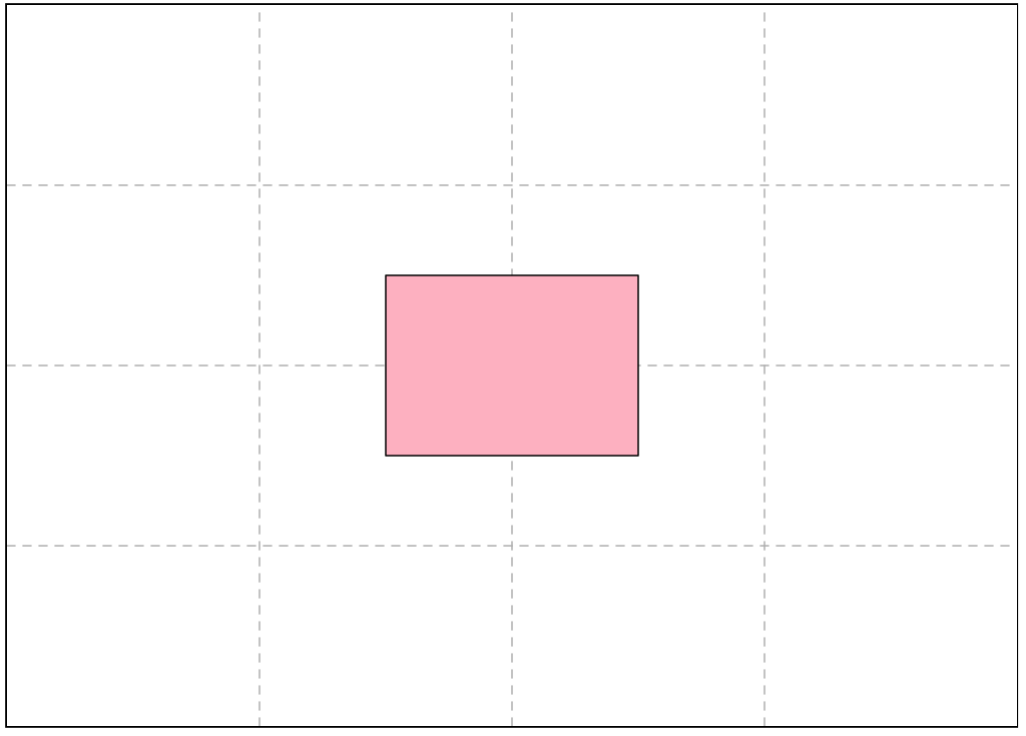
Drawing with `grid` 2

```
grid.rect(x = 0.5,
          y = 0.5,
          width = 0.25,
          height = 0.25,
          default.units = "npc",
          just = c("left", "bottom"),
          gp = gpar(fill = "pink"),
          vp = NULL)
```



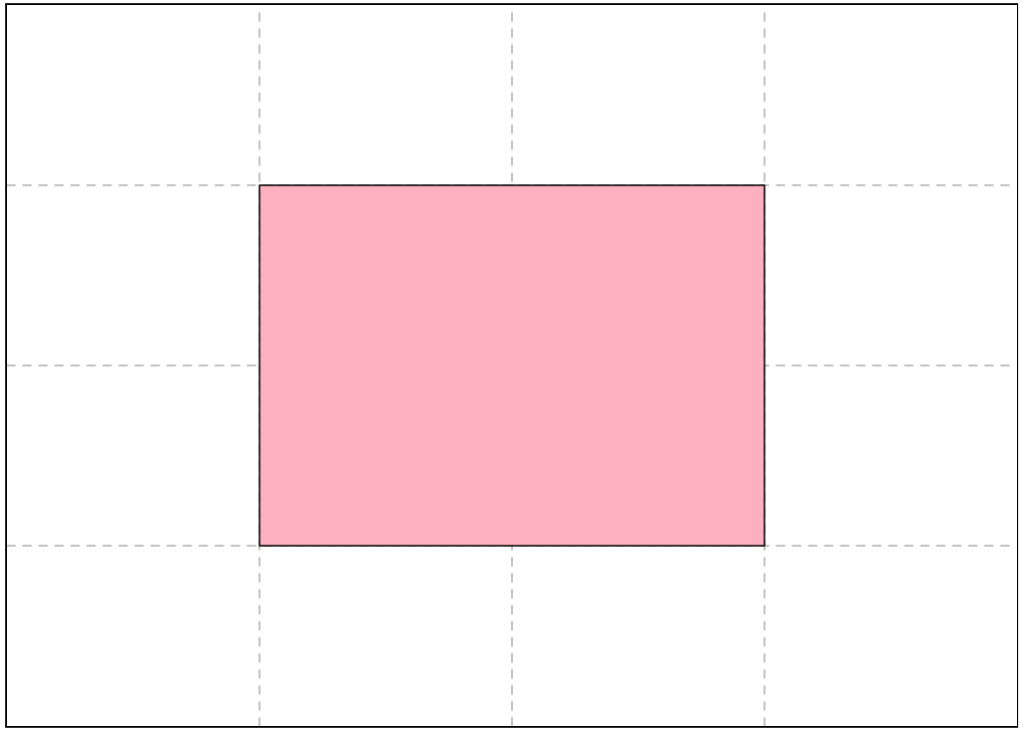
Drawing with `grid` 3

```
grid.rect(x = 0.5,
          y = 0.5,
          width = 0.25,
          height = 0.25,
          default.units = "npc",
          just = c("center", "center"),
          gp = gpar(fill = "pink"),
          vp = NULL)
```



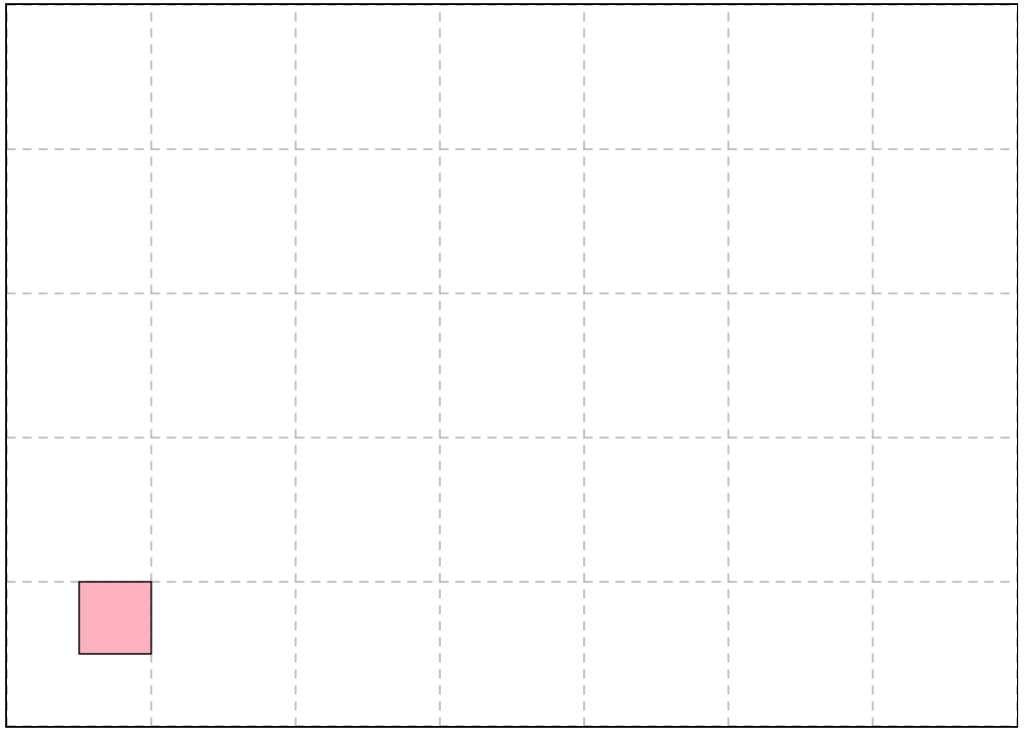
Drawing with `grid` 4

```
grid.rect(x = 0.5,
          y = 0.5,
          width = 0.5,
          height = 0.5,
          default.units = "npc",
          just = c("center", "center"),
          gp = gpar(fill = "pink"),
          vp = NULL)
```



Drawing with `grid` 5

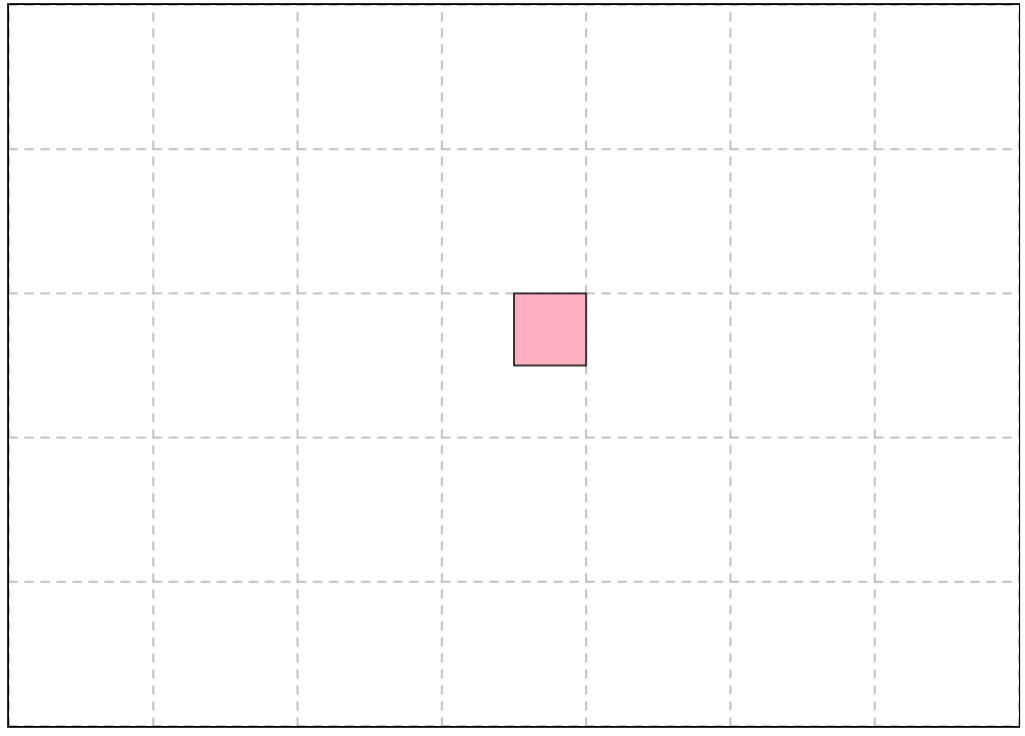
```
grid.rect(x = 0.5,
          y = 0.5,
          width = 0.5,
          height = 0.5,
          default.units = "in",
          just = c("left", "bottom"),
          gp = gpar(fill = "pink"),
          vp = NULL)
```



This canvas is 5 inches high and 7 inches wide

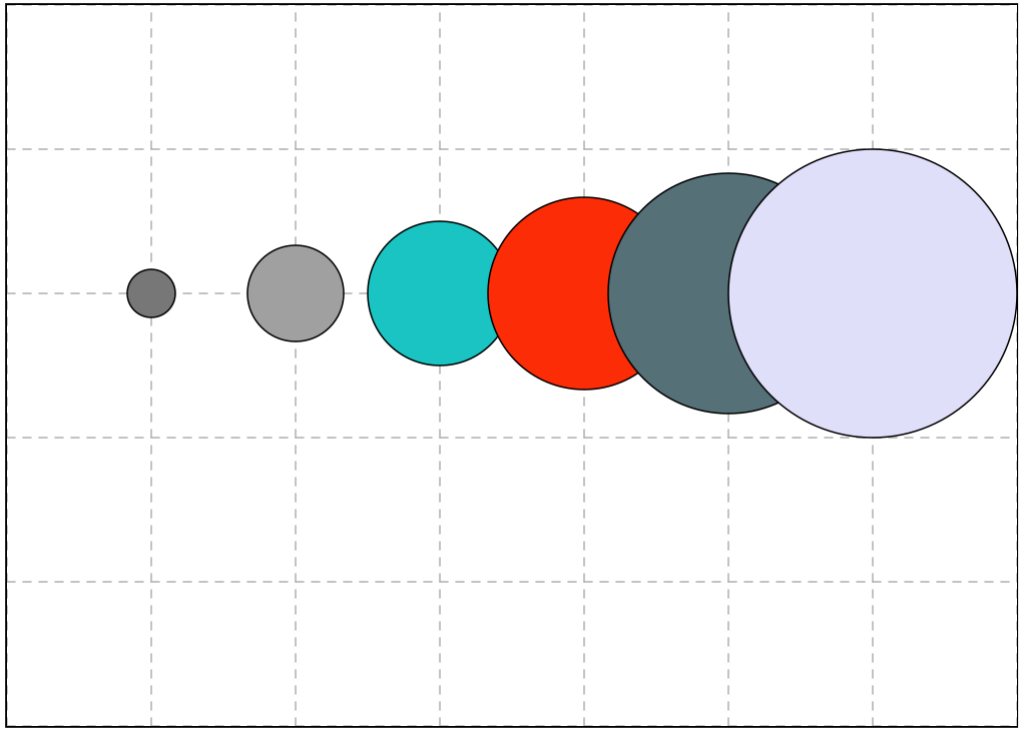
Drawing with `grid` 6

```
grid.rect(x = unit(0.5, "npc"),
          y = unit(0.5, "npc"),
          width = 0.5,
          height = 0.5,
          default.units = "in",
          just = c("left", "bottom"),
          gp = gpar(fill = "pink"),
          vp = NULL)
```



Drawing with `grid` 7

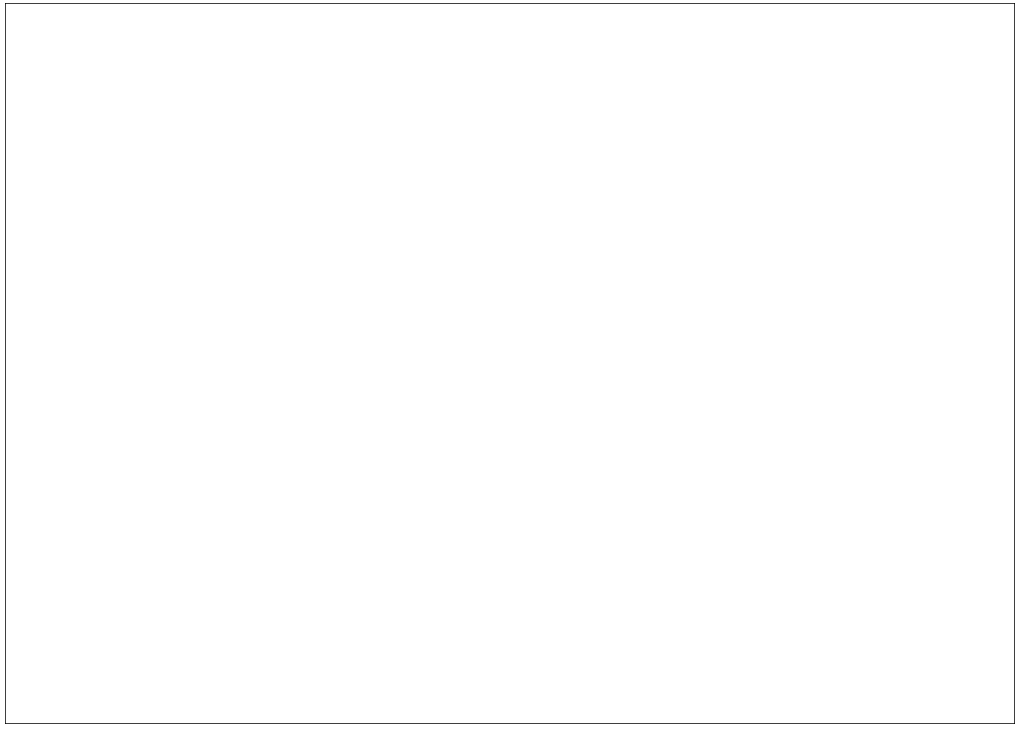
```
grid.circle(
  x = 1:6,
  y = 3,
  r = 1:6/6,
  default.units = "in",
  gp = gpar(fill = sample(colors(), 6)),
  vp = NULL)
```



- Arguments can be vectorised
- The Z-order is determined by the order of the input

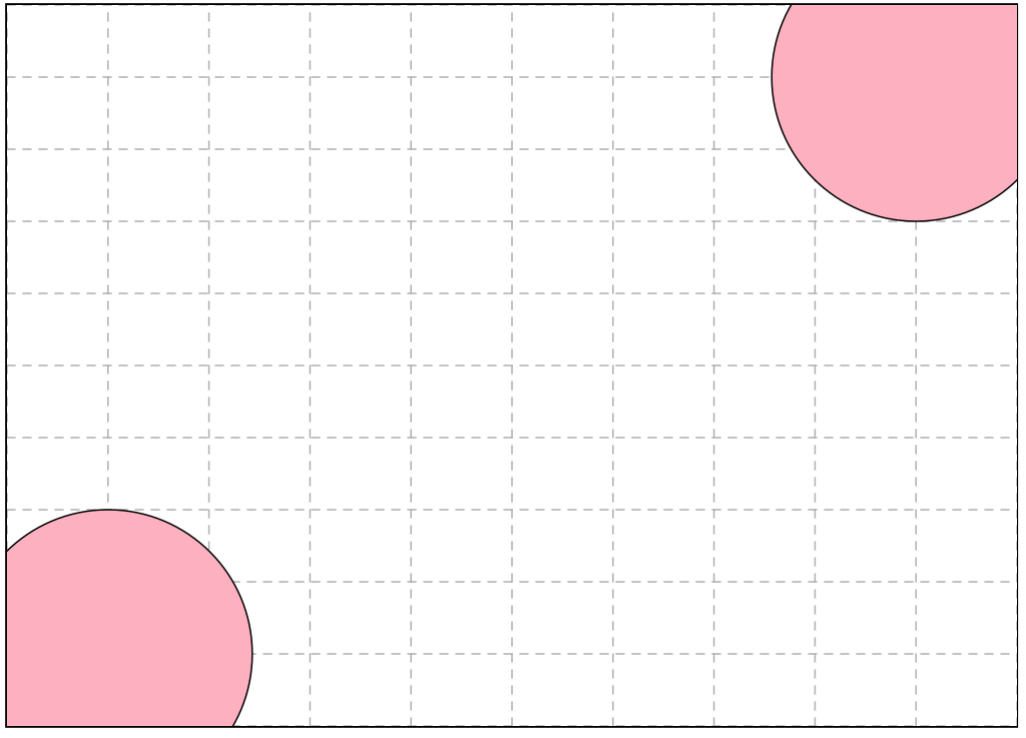
The root viewport

- A **viewport** is a polygon (often rectangular) viewing region in computer graphics.
- When `vp = NULL`, this refers to the root viewport, i.e. the canvas



Viewport in `grid` 1

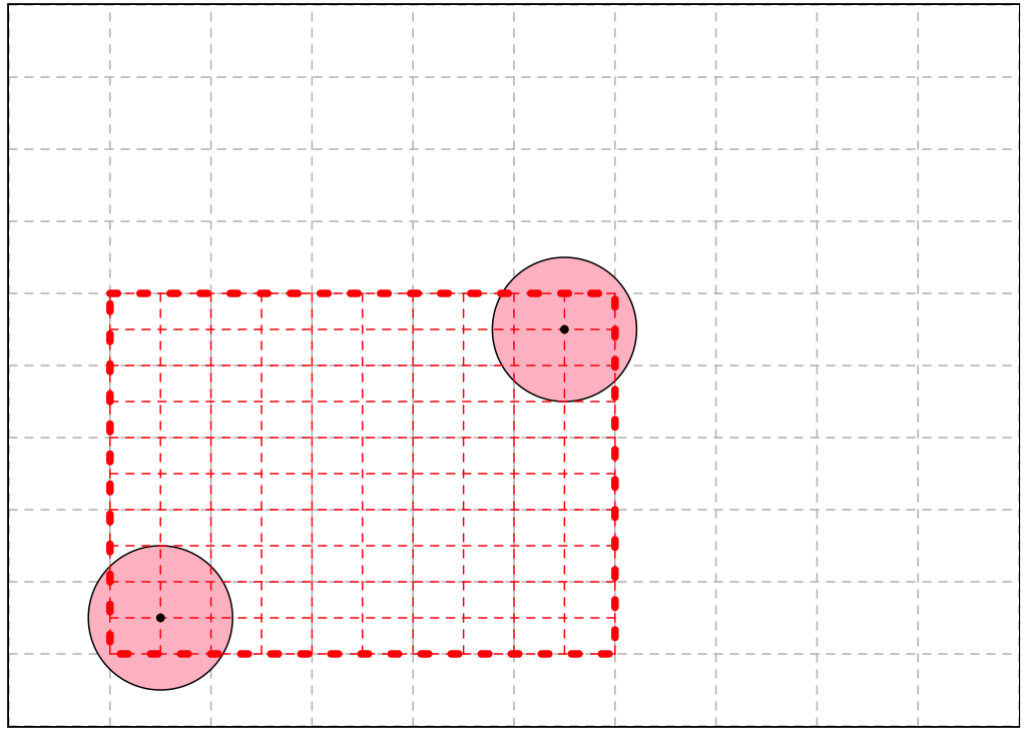
```
grid.circle(x = c(0.1, 0.9),
            y = c(0.1, 0.9),
            r = 0.2,
            default.units = "npc",
            gp = gpar(fill = "pink"),
            vp = NULL)
```



Graphical output outside of the root Viewport are not visible.

Viewport in `grid` 2

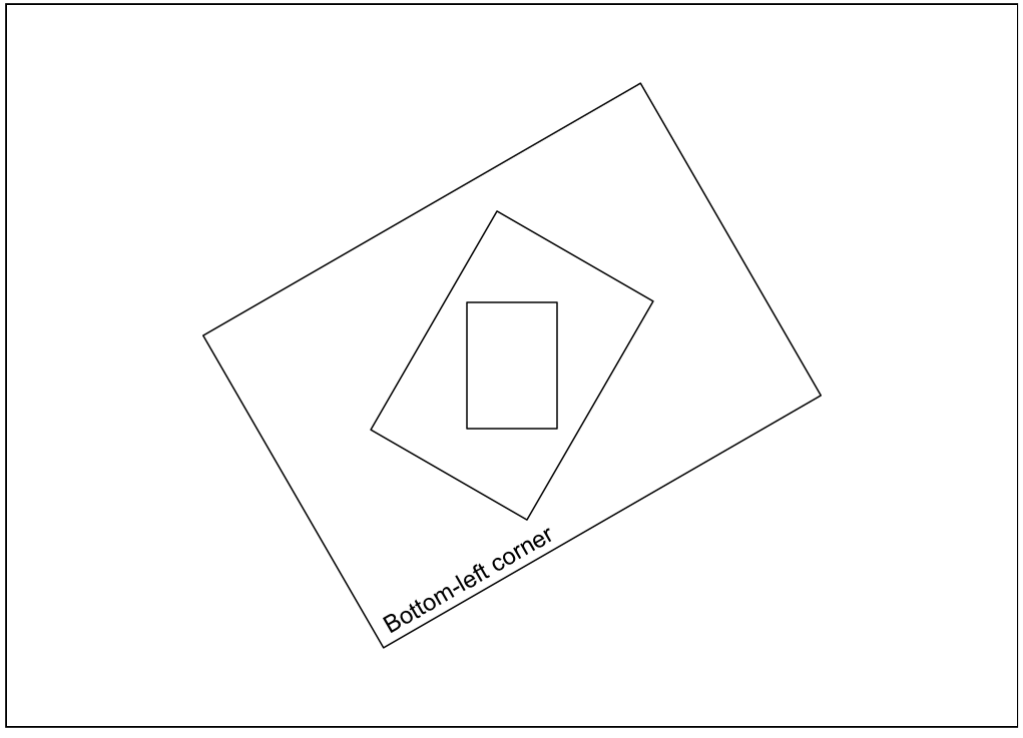
```
vp1 <- viewport(x = 0.1, y = 0.1,
         width = 0.5,
         height = 0.5,
         just = c("left", "bottom"))

grid.circle(x = c(0.1, 0.9),
            y = c(0.1, 0.9),
            r = 0.2,
            default.units = "npc",
            gp = gpar(fill = "pink"),
            vp = vp1)
```



Graphical object is drawn relative to Viewport vp1.

Viewport in `grid` 3

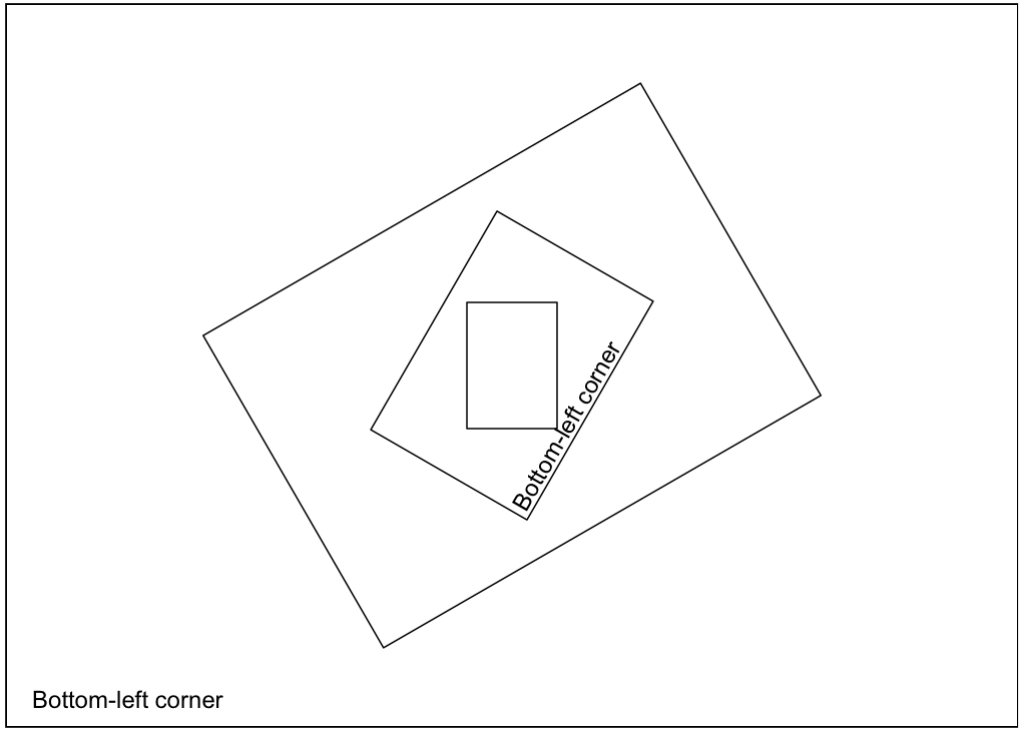
```
vp2 <- viewport(x = 0.5, y = 0.5,
         width = 0.5,
         height = 0.5,
         just = "center",
         angle = 30, name = "vp-2")
grid.rect(vp = vp2)
pushViewport(vp2)
grid.rect(vp = vp2)
pushViewport(vp2)
grid.rect(vp = vp2)
popViewport()
grid.text("Bottom-left corner",
           x = 0.025, y = 0.025,
           just = c("left", "bottom"))
```



- `pushViewport(vp)` changes all subsequent drawing context to the `viewport` object `vp`
- `popViewport()` removes current viewport and reverts to the previous drawing context

Viewport in grid 4

```
vp3 <- viewport(x = 0.5, y = 0.5,
         width = 0.5,
         height = 0.5,
         just = "center",
         angle = 30, name = "vp-3")
grid.rect(vp = vp2)
pushViewport(vp2)
grid.rect(vp = vp2)
pushViewport(vp3)
grid.rect(vp = vp2)
upViewport(n = 2)
grid.text("Bottom-left corner",
          x = 0.025, y = 0.025,
          just = c("left", "bottom"))
downViewport("vp-3")
grid.text("Bottom-left corner",
          x = 0.025, y = 0.025,
          just = c("left", "bottom"))
```



- upViewport()
- downViewport()

Resources

See Murrell (2019) "R Graphics" book for more about `grid`

</> Open day1-exercise-01.Rmd

15:00

Session Information

```
devtools::session_info()
```

```
## ─ Session info  🤗  💤  🇲🇩 ──────────────────────────────────────────────
##  hash: smiling face with open hands, zzz, flag: Moldova
##
##  setting  value
##  version  R version 4.1.2 (2021-11-01)
##  os       macOS Catalina 10.15.7
##  system   x86_64, darwin17.0
##  ui       X11
##  language (EN)
##  collate  en_AU.UTF-8
##  ctype    en_AU.UTF-8
##  tz       Australia/Melbourne
##  date     2021-12-07
```