Tidy evaluation

September 2017

Hadley Wickham
@hadleywickham
Chief Scientist, **RStudio**

Tidy eval = principled NSE used in the tidyverse

dplyr

www.rstudio.com

tidyr

www.rstudio.com

TIBBLE

www.rstudio.com

In dev version

ggplot2

www.rstudio.com

Motivation

Names usually refer to objects in an environment

```
f <- function(x, y) (x - 1) * (y + 1)
```

```
a <- 1:10
b <- runif(10)
```

```
f(a, b)
```

But in the tidyverse, they often refer to columns in data

```
ggplot(mtcars, aes(disp, mpg)) + geom_point()

mtcars %>%
group_by(cyl) %>%
summarise(
  mpg = mean(mpg),
  disp = mean(disp)
)
```

We need some new vocabulary

Evaluated using usual R rules

```
ggplot(mtcars, aes(disp, mpg)) + geom_point()
```

Quoted and evaluated in a "non-standard" way

```
mtcars %>%
  group_by(cyl) %>%
  summarise(
    mpg = mean(mpg),
    disp = mean(disp)
  )
```

Other base R functions also quote an argument

```
mtcars$cyl
```

```
library(ggplot2)
```

```
rm(mtcars)
```

```
subset(mtcars, cyl == 2)
```

A bunch of base R functions quote an input

Can rewrite every operation in prefix form

```
`$`(mtcars, cyl)
```

```
library(ggplot2)
```

```
rm(mtcars)
```

```
subset(mtcars, cyl == 2)
```

How do you make this code work?

```
var <- "cyl"
mtcars$var
```

```
package <- "ggplot2"
library(package)
```

```
obj <- "mtcars"
rm(obj)
```

And each uses a different technique to unquote

```r
# Alternative function
var <- "cyl"
mtcars[[var]]

# Argument controls quoting vs evaluating
package <- "ggplot2"
library(package, character.only = TRUE)

# Alternative argument
obj <- "mtcars"
rm(list = obj)
```

Quasiquotation provides a consistent approach

Short for quote; we'll learn more about it later

```
var <- quo(cyl)

mtcars %>% group_by(!!var) %>% tally(vs)

mtcars %>% nest(!!var)

# in the dev version
ggplot(mtcars, aes(!!var, mpg)) + geom_point()
```

Particularly powerful because can unnest anywhere

```
var <- quo(mpg)

mtcars %>%
  group_by(cyl) %>%
  summarise(
    mean = mean(!!var, na.rm = TRUE),
    sd = sd(!!var, na.rm = TRUE)
  )
```

Why is this so important?

```
df1 %>%
  group_by(g1) %>%
  summarise(mean = mean(a))

df2 %>%
  group_by(g2) %>%
  summarise(mean = mean(b))

df3 %>%
  group_by(g3) %>%
  summarise(mean = mean(c))

df4 %>%
  group_by(g4) %>%
  summarise(mean = mean(d))
```

We want this

```
df1 %>% grouped_mean(g1, a)
df2 %>% grouped_mean(g2, b)
df3 %>% grouped_mean(g3, c)
df4 %>% grouped_mean(g4, d)
```

But we can't write

```
grouped_mean <- function(df, group_var, mean_var) {
 df %>%
    group_by(group_var) %>%
    summarise(mean = mean(mean_var))
}

df <- data.frame(x = c(1, 1, 2), y = c(1, 2, 3))
df %>% grouped_mean(x, y)
```

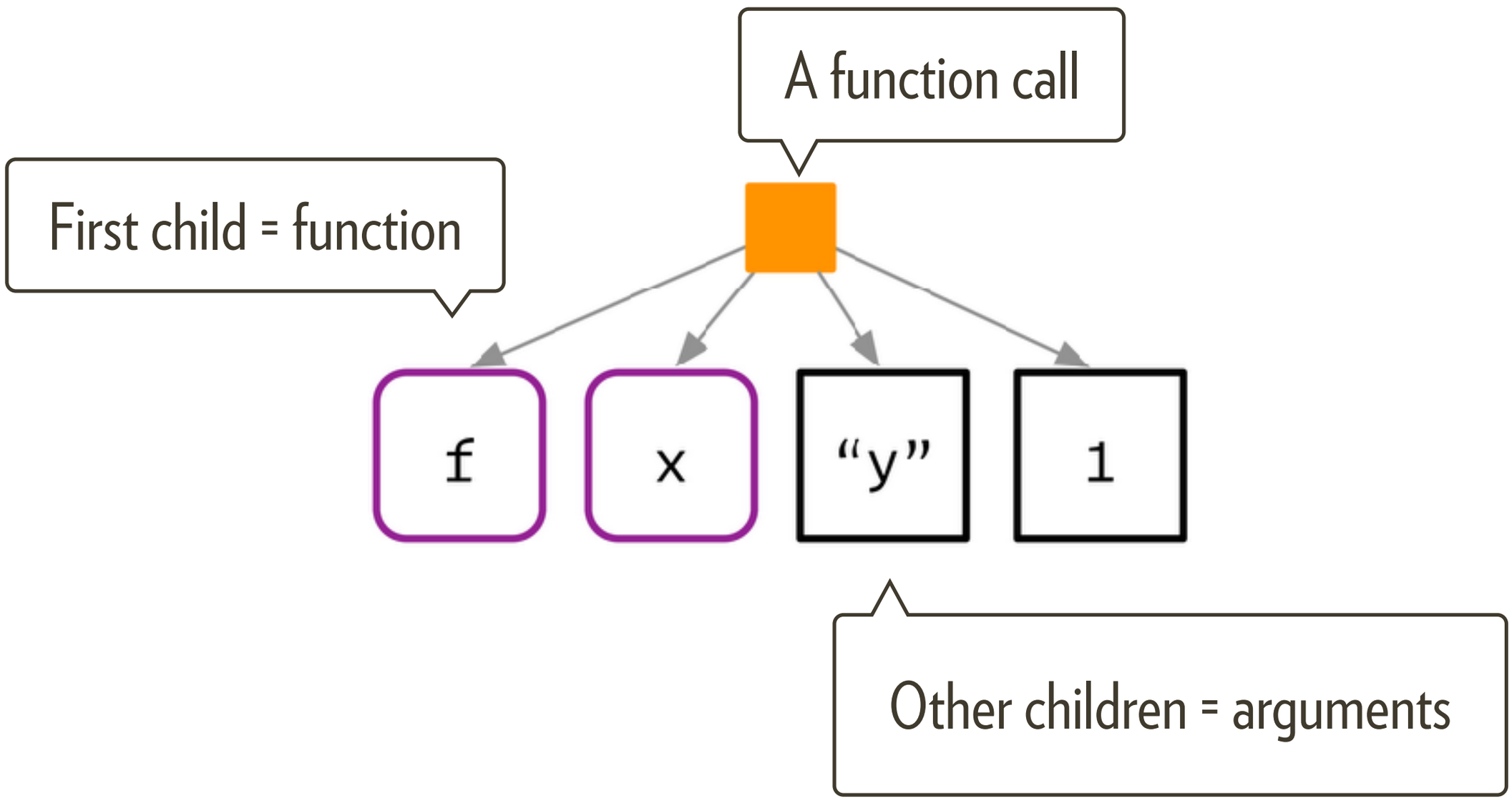
Need to learn three big ideas

R code is a tree

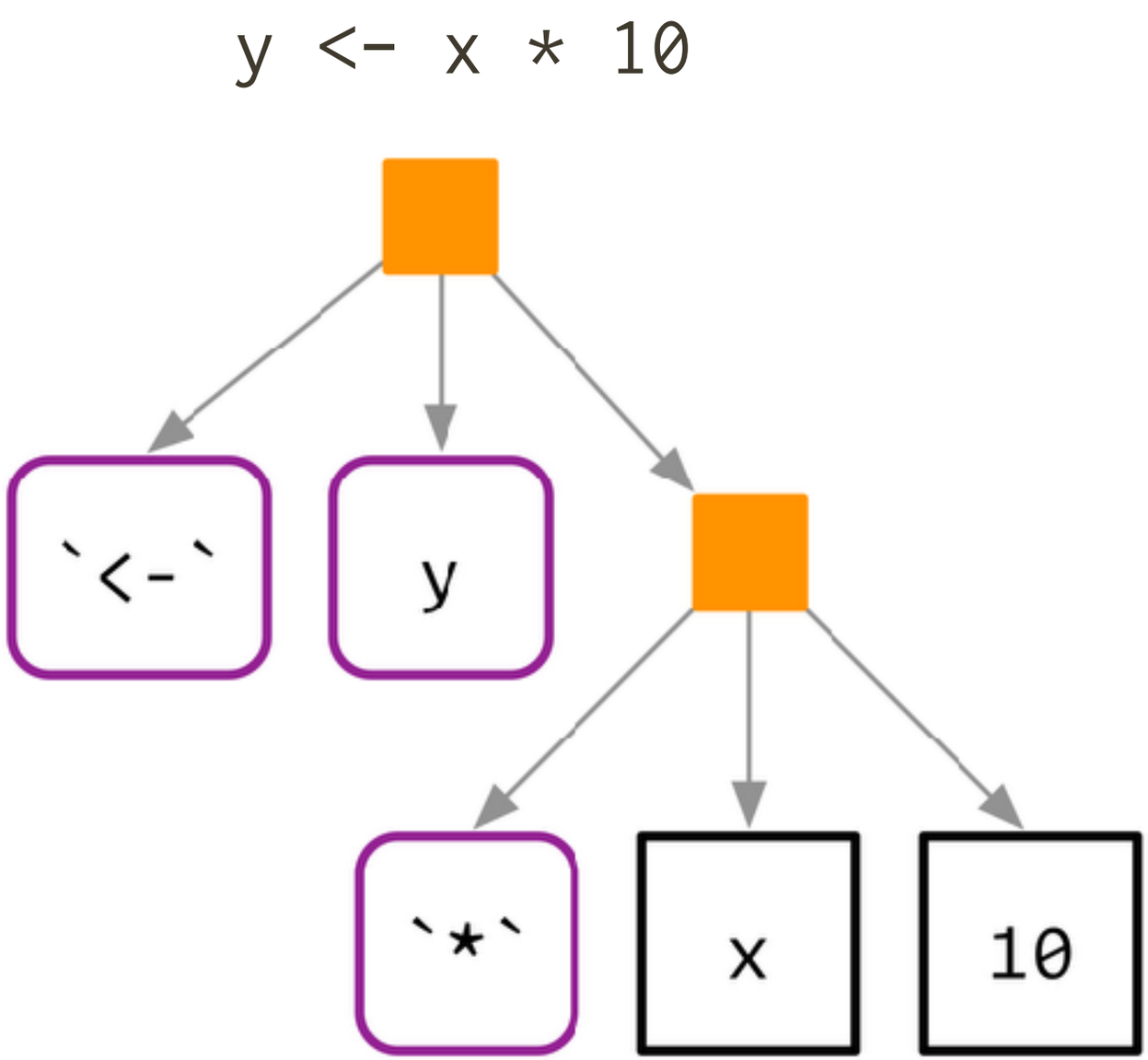
Quoting & unquoting

Data masks

Abstract syntax tree (AST)

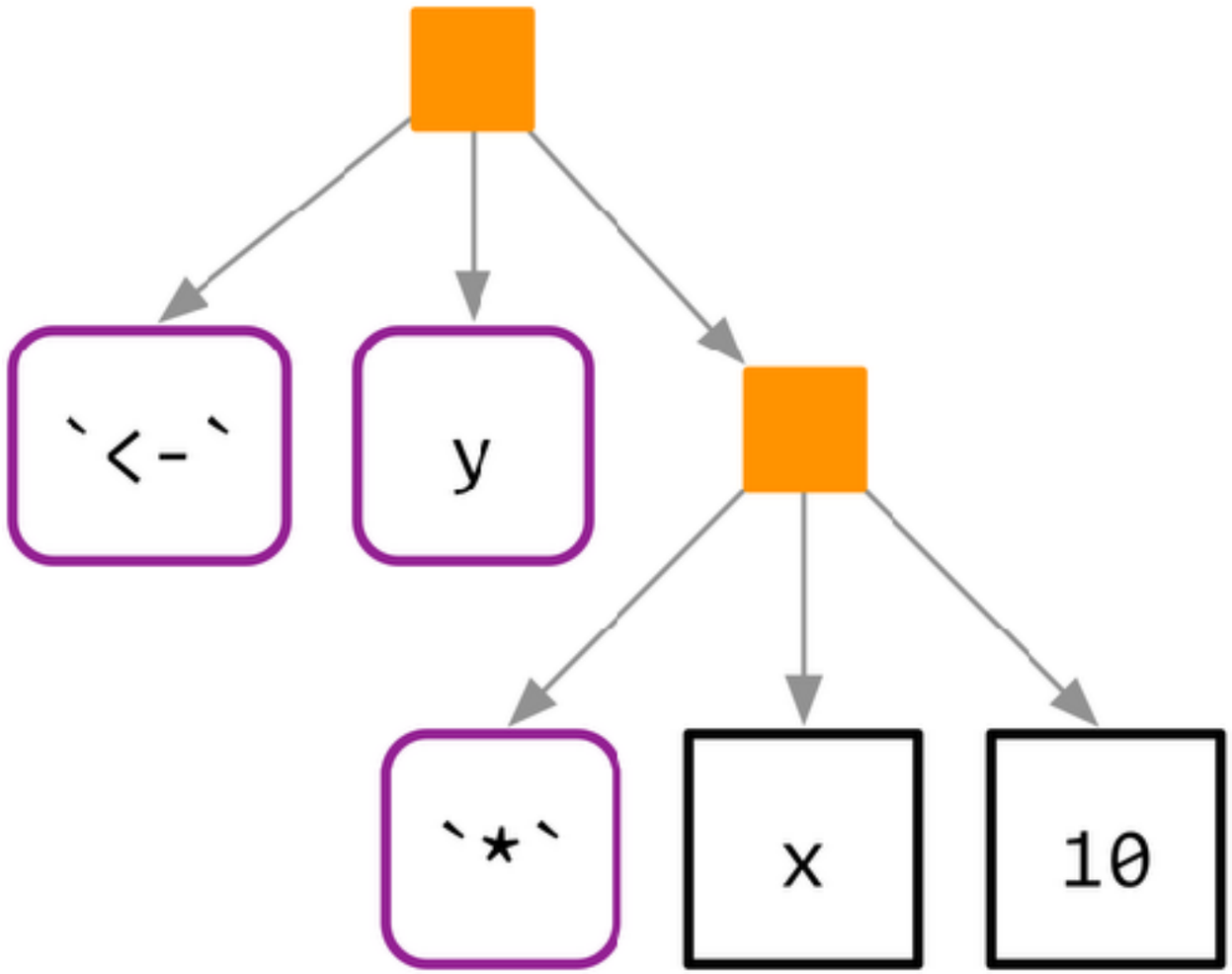

f(x, "y", 1)
A function call
First child = function
f
x
"y"
1
Other children = arguments

Every call has this form



Every call has this form

```
`<-`(y, `*`(x, 10))
```



Your turn

```
library(lobstr)
# devtools::install_github("hadley/r-lib")

# Compare to my hand drawn diagrams
ast(f(x, "y", 1)
ast(y <- x * 10)
```

```
# What does this tree tell you?
lobstr::ast(function(x, y) {
  if (x > y) {
    x
  } else {
    y
  }
})
```

Quoting &
Unquoting

Four related functions

```
# Capture your expressions
expr(x + 1)
exprs(x, y, z)

# Capture users expressions
f2 <- function(x) enexpr(x)
f1 <- function(...) enexpr(...)

# These are best for exploration, but for
# live code you should use the quo() versions;
# we'll get to those later
```

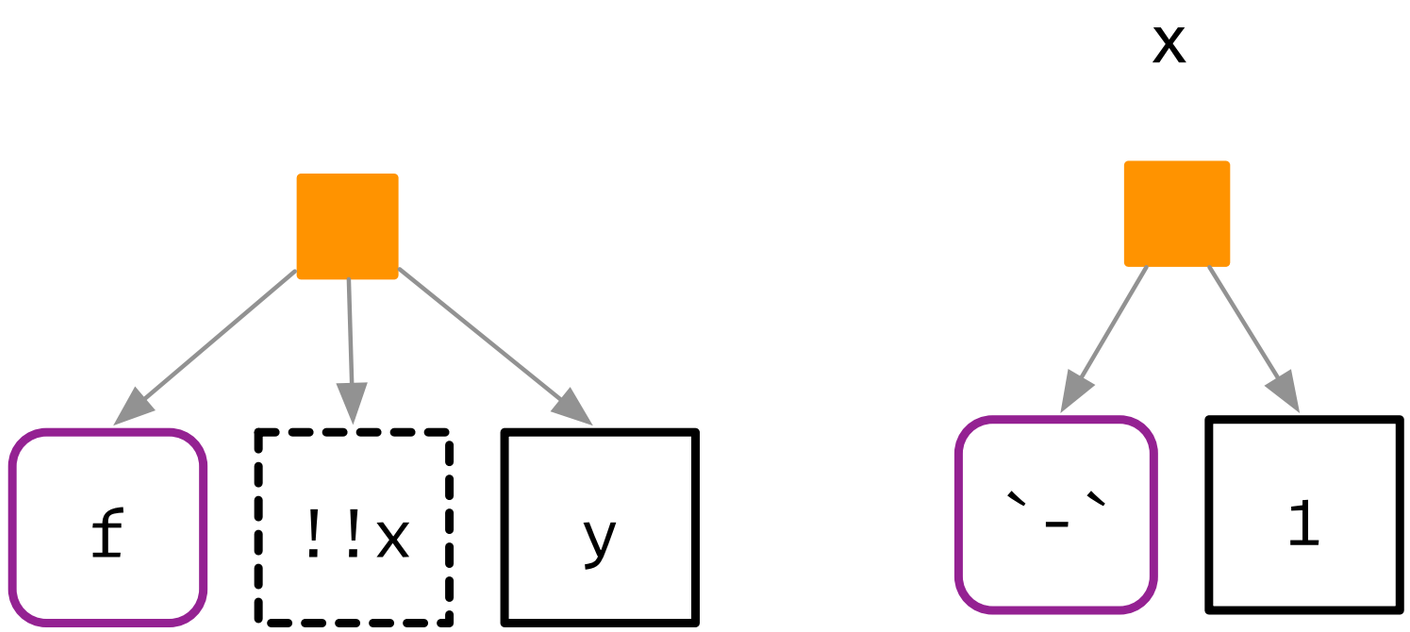
One opposite of quoting is unquoting

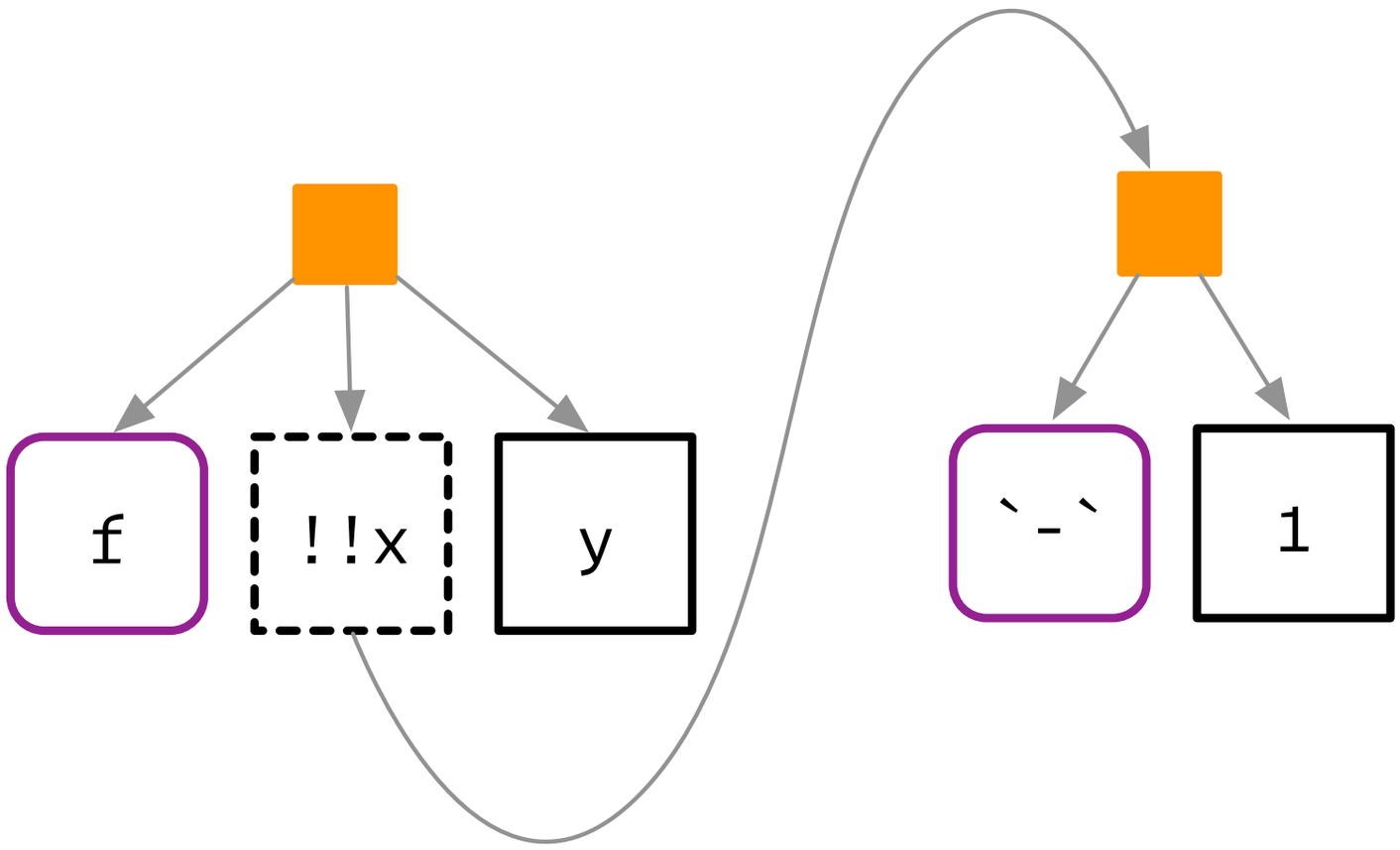
```
x1 <- expr(f(x, "y", 1)
x2 <- expr(y <- x * 10)
```

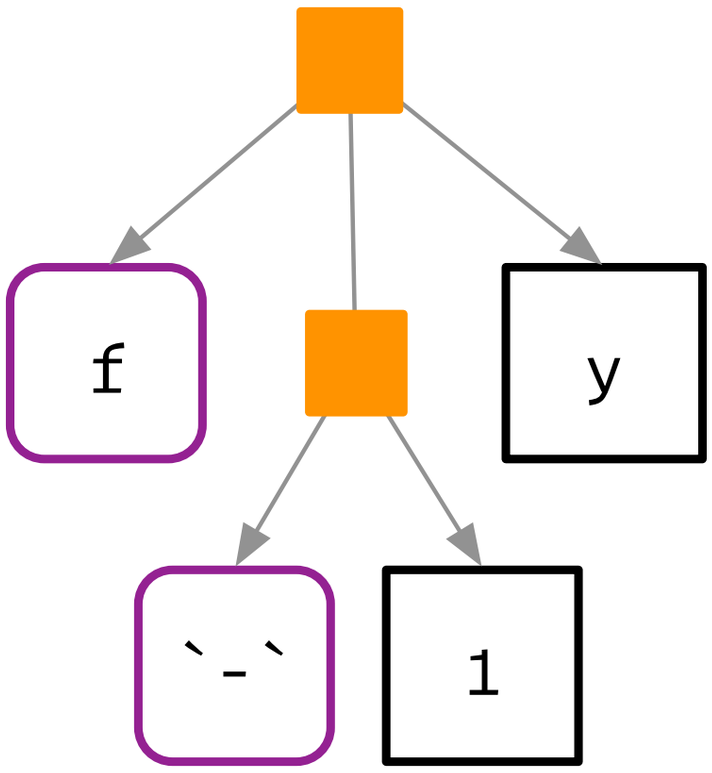
```
# Not useful!
ast(x1)

# But can use unquoting
ast(!!x1)
```

Unquoting allows you to build your own trees

```
x <- expr(a + b + c)
expr(f(!!x, y))
#> f(a + b + c, y)
```

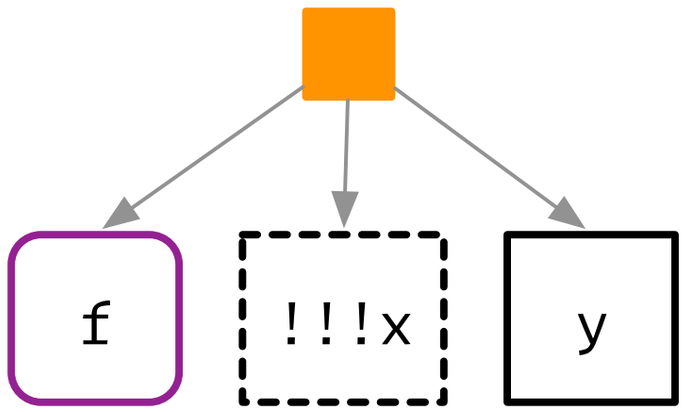

f
!!x
y
x
`-`
1

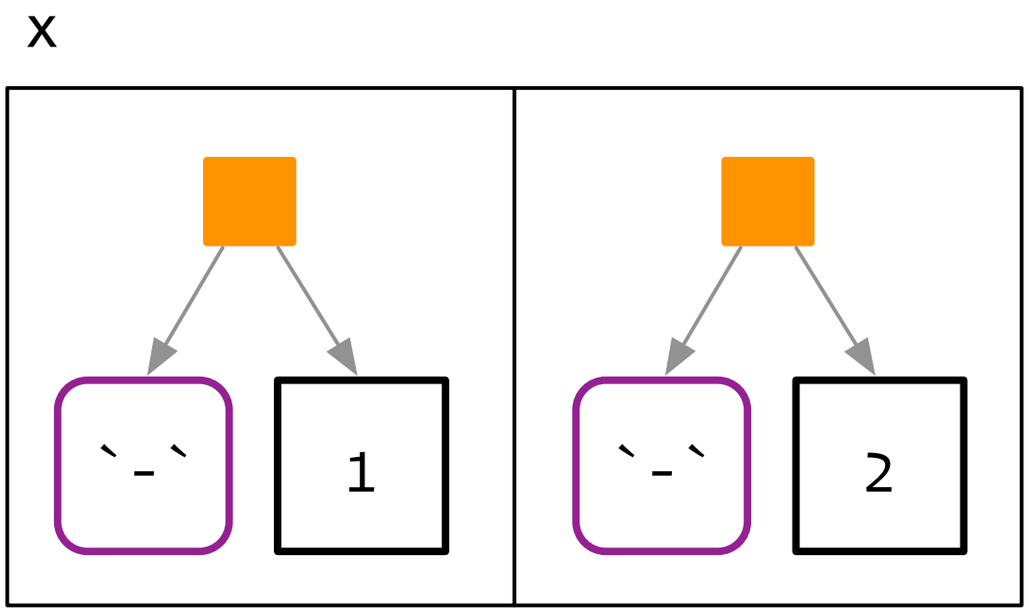

f
!!x
y
`-`
1

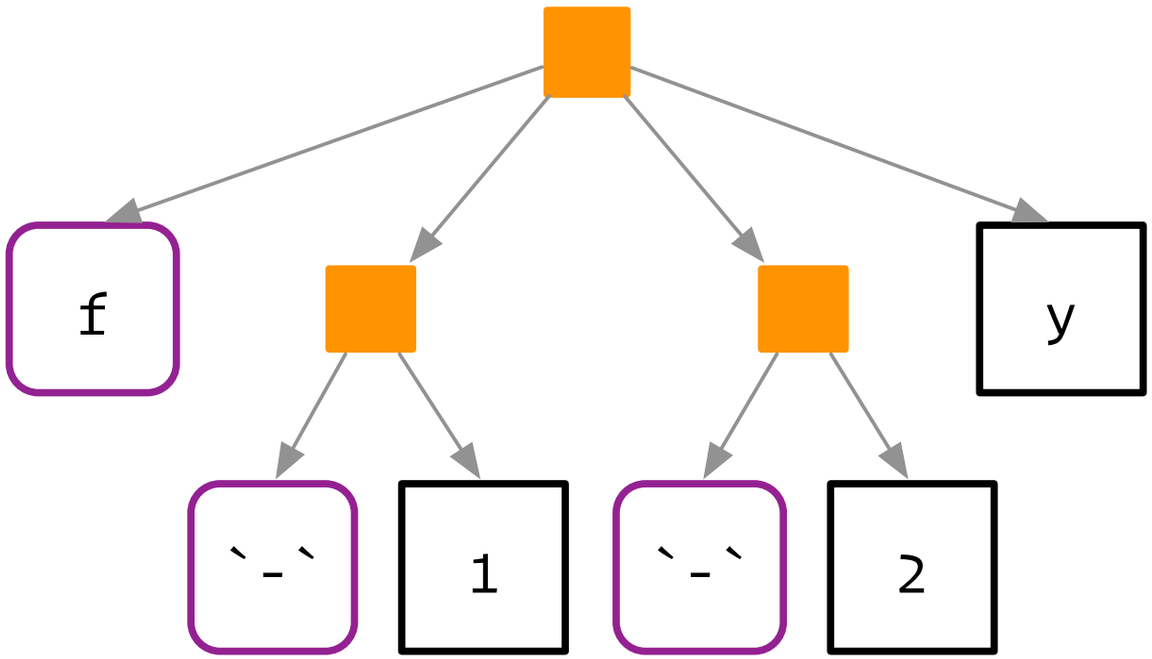

f
y
`_`
1

!! is 1-to-1; !!! is 1-to-many

```
x <- exprs(1, 2, 3, y = 10)
expr(f(!!!x, z = z))
#> f(1, 2, 3, y = 10, z = z)
```


f
!!!x
y


x
`-`
1
`-`
2


f
y
`-`
1
`-`
2

Really beautiful in conjunction with purrr

```
intercept <- 10
coefs <- c(x1 = 5, x2 = -4)

coef_sym <- syms(names(coefs))
summands <- map2(coef_sym, coefs,
  ~ expr((!!.x * !!.y))
)
summands

summands <- c(intercept, summands)
summands

eq <- reduce(summands, ~ expr(!!.x + !!.y))
eq
```

Data masks

A **data mask** lets you refer to variables in data

```
filter(diamonds, x > 0 & y > 0 & z > 0)

# vs
diamonds[
  diamonds$x > 0 &
  diamonds$y > 0 &
  diamonds$z > 0,
]
```

How do we reduce the duplication here?

```
df1 %>%
  group_by(g1) %>%
  summarise(mean = mean(a))

df2 %>%
  group_by(g2) %>%
  summarise(mean = mean(b))

df3 %>%
  group_by(g3) %>%
  summarise(mean = mean(c))

df4 %>%
  group_by(g4) %>%
  summarise(mean = mean(d))
```

Why does this fail?

```
grouped_mean <- function(df, group_var, mean_var) {

  df %>%
    group_by(group_var) %>%
    summarise(mean = mean(mean_var))
}

df <- data.frame(x = c(1, 1, 2), y = c(1, 2, 3))
df %>% grouped_mean(x, y)
```

Why does this fail?

```
grouped_mean <- function(df, group_var, mean_var) {
  group_var <- expr(group_var)
  mean_var <- expr(mean_var)

  df %>%
    group_by(!!group_var) %>%
    summarise(mean = mean(!!mean_var))
}

df <- data.frame(x = c(1, 1, 2), y = c(1, 2, 3))
df %>% grouped_mean(x, y)
```

We need to capture *user* expression

```
grouped_mean <- function(df, group_var, mean_var) {
  group_var <- enexpr(group_var)
  mean_var <- enexpr(mean_var)

  df %>%
    group_by(!!group_var) %>%
    summarise(mean = mean(!!mean_var))
}

df <- data.frame(x = c(1, 1, 2), y = c(1, 2, 3))
df %>% grouped_mean(x, y)
```

Technically we need to capture quosures

```
grouped_mean <- function(df, group_var, mean_var) {
  group_var <- enquo(group_var)
  mean_var <- enquo(mean_var)

  df %>%
    group_by(!!group_var) %>%
    summarise(mean = mean(!!mean_var))
}

df <- data.frame(x = c(1, 1, 2), y = c(1, 2, 3))
df %>% grouped_mean(x, y)
```

Quosure = expression + environment

```
# Capture your expressions
quo(x + 1)
quos(x, y, z)

# Capture users expressions
f2 <- function(x) enquo(x)
f1 <- function(...) enquos(...)
```

Which means we need to understand this

```
grouped_mean <- function(df, group_var, mean_var) {
  group_var <- enquo(group_var)
  mean_var <- enquo(mean_var)

  df %>%
    group_by(!!group_var) %>%
    summarise(mean = mean(!!mean_var))
}

df <- data.frame(x = c(1, 1, 2), y = c(1, 2, 3))
df %>% grouped_mean(x, y)
```

Your turn: make a function

```
summarise(df,
  mean = mean(a),
  sum = sum(a),
  n = n()
)
summarise(df,
  mean = mean(b),
  sum = sum(b),
  n = n()
)
summarise(df,
  mean = mean(a * b),
  sum = sum(a * b),
  n = n()
)
```

Your turn: make a function

```
df %>% group_by(x) %>% summarise(n = n())
df %>% group_by(x, y) %>% summarise(n = n())
df %>% group_by(x, y, z) %>% summarise(n = n())
```

Learn more

Advanced R (2e)

https://adv-r.hadley.nz/meta.html

https://adv-r.hadley.nz/expressions.html

https://adv-r.hadley.nz/quasiquotation.html

https://adv-r.hadley.nz/evaluation-1.html

https://adv-r.hadley.nz/dsl.html