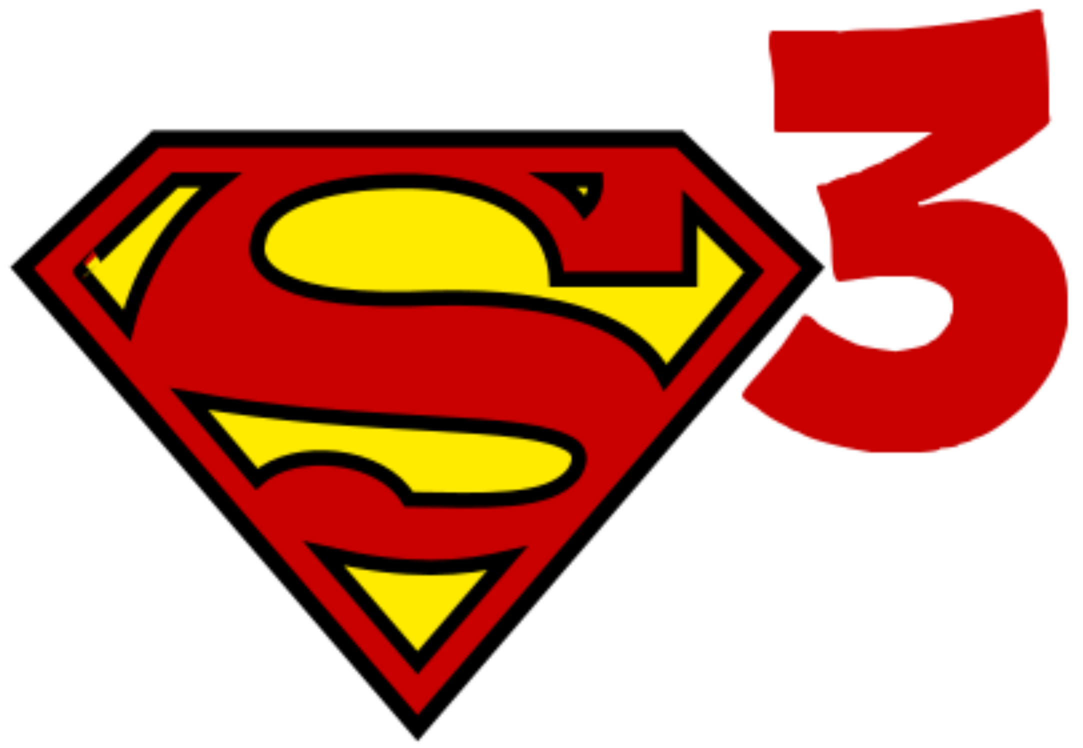
OBJECT-ORIENTED PROGRAMMING IN R: S3 & R6

Generics and Methods

```
> summary(c(TRUE, FALSE, NA, TRUE))
   Mode   FALSE    TRUE    NA's
logical       1       2       1
```

```
> summary(rgamma(1000, 1))
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.000354 0.276500 0.690300 1.020000 1.384000 9.664000
```

function overloading = input-dependent function behavior



$$\begin{tikzcd}
 & \text{generic} \\
\text{function} \arrow[ur] \arrow[dr] & \\
 & \text{method}
\end{tikzcd}$$

```
> print
function (x, ...)
UseMethod("print")
<bytecode: 0x1062f0870>
<environment: namespace:base>
```

Methods are named `generic.class`

- `print.Date`
- `summary.factor`
- `unique.array`

Method signatures contain generic signatures

```
> args(print)
function (x, ...)
NULL
```

```
> args(print.Date)
function (x, max = NULL, ...)
NULL
```

pass arguments between methods with ...
include it in both generic and methods

```
> print.function
function (x, useSource = TRUE, ...)
.Internal(print.function(x, useSource, ...))
```

```
> print.Date
function (x, max = NULL, ...)
{
    if (is.null(max))
        max <- getOption("max.print", 9999L)
    if (max < length(x)) {
        print(format(x[seq_len(max)]), max = max, ...)
        cat(" [ reached getOption(\"max.print\") --
omitted",
            length(x) - max, "entries ]\n")
    }
    else print(format(x), max = max, ...)
    invisible(x)
}
```

~~`lower.leopard.case`~~

`lower_snake_case`

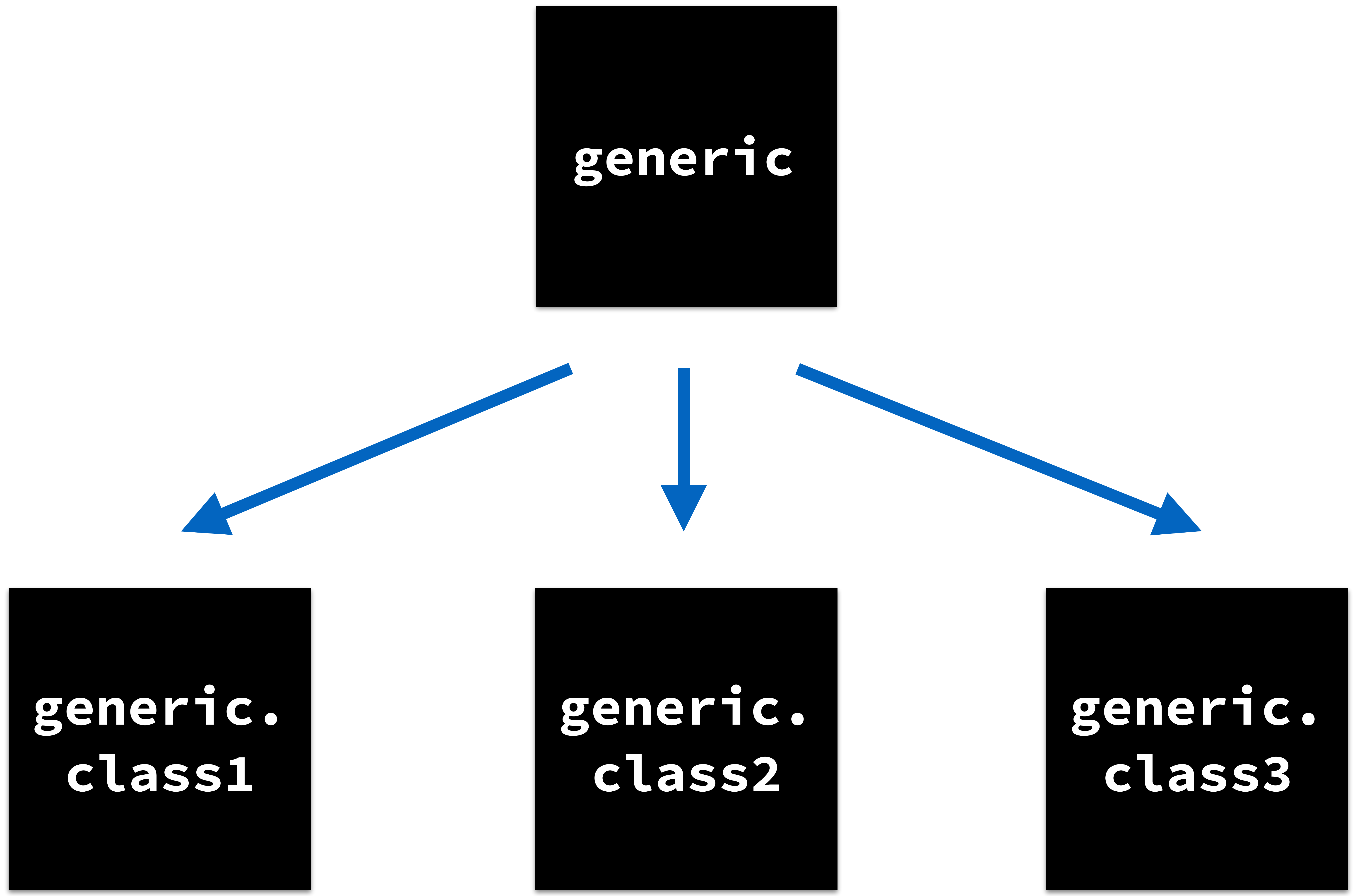
`lowerCamelCase`

Summary

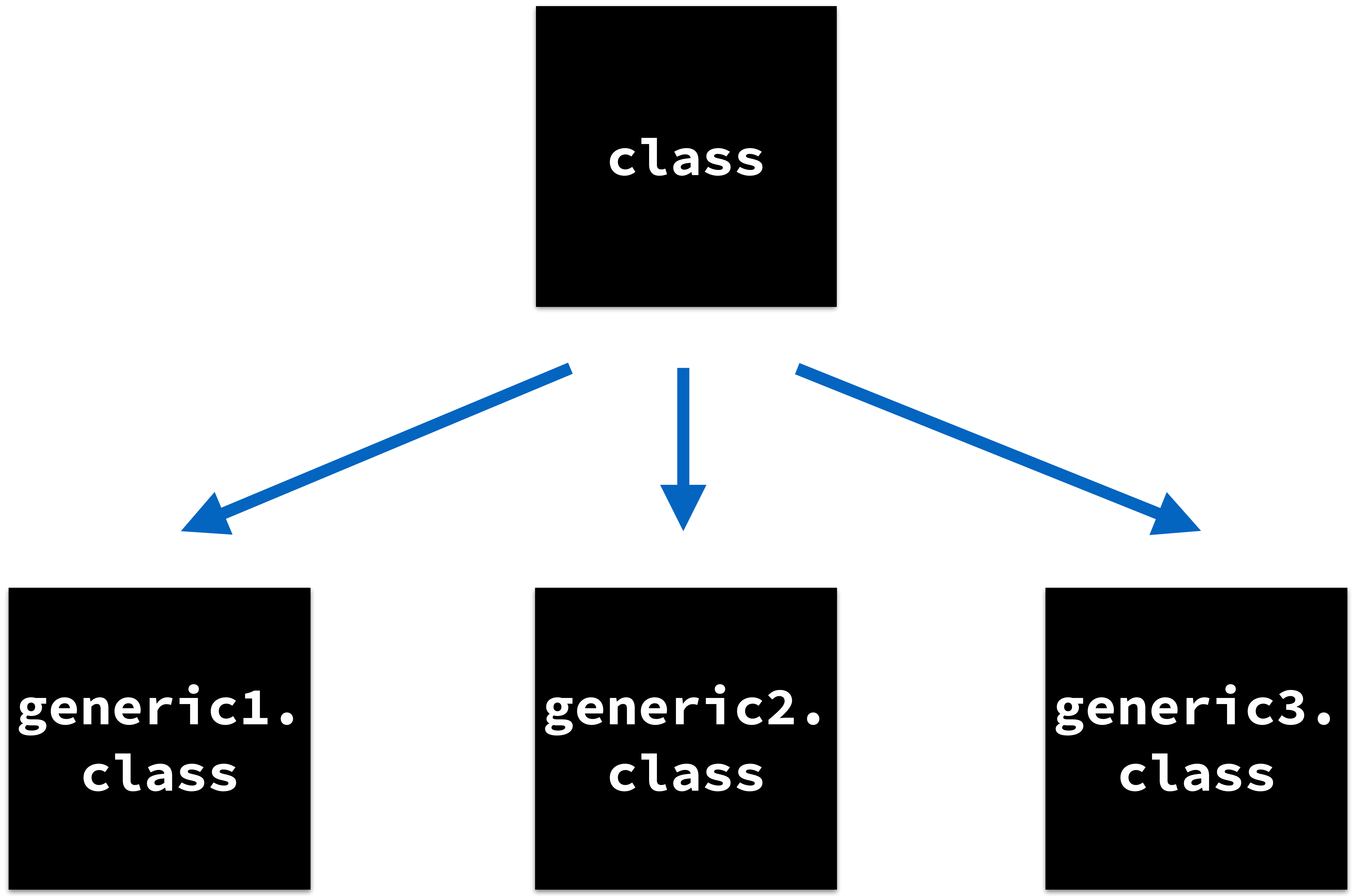
- Functions **split** into **generic** + **method**
- Methods named **`generic.class`**
- Method args **contain generic** args
- Include a **...** arg
- Use **`lower_snake_case`** or **`lowerCamelCase`**

Let’s practice!

Methodical Thinking


generic
generic.
class1
generic.
class2
generic.
class3

```
> methods("mean") # or methods(mean)
[1] mean.Date     mean.default  mean.difftime mean.POSIXct
[5] mean.POSIXlt
see '?methods' for accessing help and source code
```


class
generic1.class
generic2.class
generic3.class

```
> methods(class = "glm") # or methods(class = glm)
 [1] add1           anova          coerce
 [4] confint        cooks.distance deviance
 [7] drop1          effects        extractAIC
[10] family         formula        influence
[13] initialize     logLik         model.frame
[16] nobs           predict        print
[19] residuals      rstandard      rstudent
[22] show           slotsFromS3    summary
[25] vcov           weights
see '?methods' for accessing help and source code
```

`methods()` returns **S3** *and* **S4** methods

```
> .S3methods(class = "glm")
 [1] add1            anova           confint
 [4] cooks.distance  deviance        drop1
 [7] effects         extractAIC      family
[10] formula         influence       logLik
[13] model.frame     nobs            predict
[16] print           residuals       rstandard
[19] rstudent        summary         vcov
[22] weights
see '?methods' for accessing help and source code
```

```
> .S4methods(class = "glm")
[1] coerce      initialize  show        slotsFromS3
see '?methods' for accessing help and source code
```

Summary

- `methods()` **finds methods** for a generic
- ... or for a **class**
- `.S3methods()` finds **only S3** methods

Let's practice!

OBJECT-ORIENTED PROGRAMMING IN R: S3 & R6

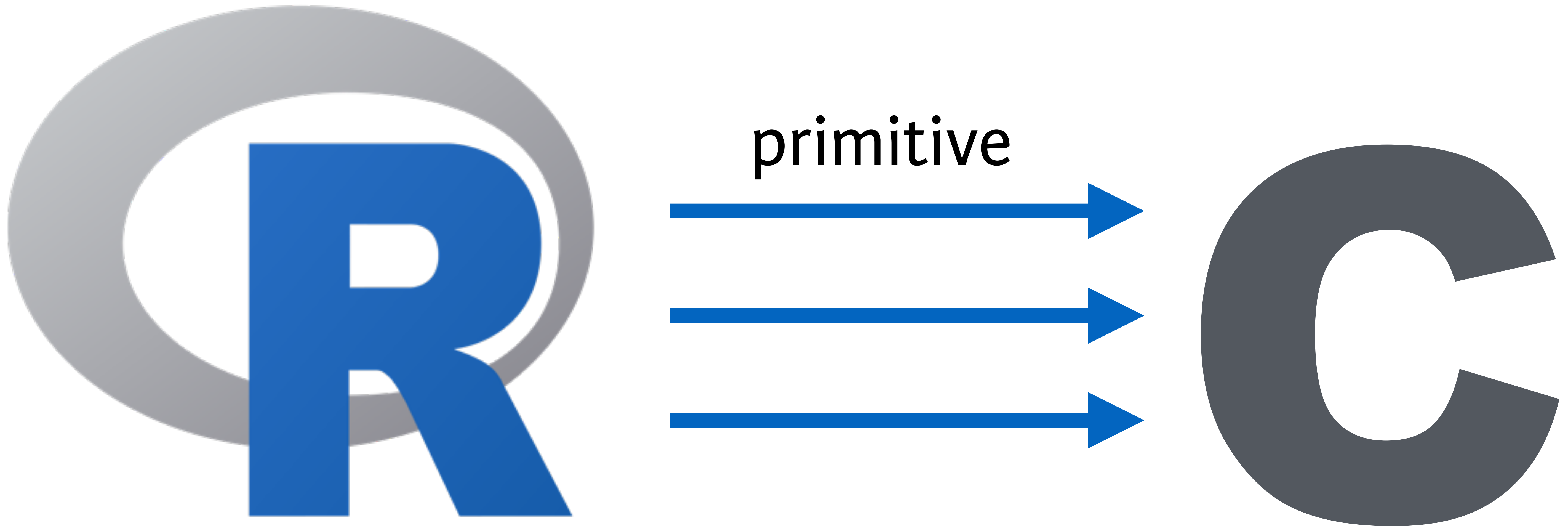
Method Lookup for Primitive Generics

- Writing code
- Debugging code
- Maintaining code

- Running code

R vs. C

- **C** code often **runs** faster
- **R** code is usually easier to **write**
- ... and easier to **debug**


primitive

```
> exp
function (x)  .Primitive("exp")
> sin
function (x)  .Primitive("sin")
```

```
> `+`
function (e1, e2)  .Primitive("+")
> `-`
function (e1, e2)  .Primitive("-")
```

```
> `if`
.Primitive("if")
> `for`
.Primitive("for")
```

```
> .S3PrimitiveGenerics
 [1] "anyNA"          "as.character"   "as.complex"
 [4] "as.double"      "as.environment" "as.integer"
 [7] "as.logical"     "as.numeric"     "as.raw"
[10] "c"              "dim"            "dim<-"
[13] "dimnames"       "dimnames<-"     "is.array"
[16] "is.finite"      "is.infinite"    "is.matrix"
[19] "is.na"          "is.nan"         "is.numeric"
[22] "length"         "length<-"       "levels<-"
[25] "names"          "names<-"        "rep"
[28] "seq.int"        "xtfrm"
```

```
> all_of_time <- c("1970-01-01", "2012-12-21")
> as.Date(all_of_time)
[1] "1970-01-01" "2012-12-21"
```

```
> class(all_of_time) <- "date_strings"
> as.Date(all_of_time)
Error in as.Date.default(all_of_time) :
  do not know how to convert 'all_of_time' to class “Date”
```

```
> length(all_of_time)
[1] 2
```

Summary

- Some R functions are actually **written in C**
- The **primitive** interface gives **best performance**
- `.S3PrimitiveGenerics` lists **primitive S3 generics**
- Primitive generics **don't throw an error** when no method is found

Let's practice!

Too Much Class

```
> x <- c(1, 3, 6, 10, 15)
> class(x) <- c(
    "triangular_numbers", "natural_numbers", "numeric"
  )
```

```
> is.numeric(x)
[1] TRUE
```

```
> is.triangular_numbers(x)
Error: could not find function "is.triangular_numbers"
```

```
> inherits(x, "triangular_numbers")
[1] TRUE
> inherits(x, "natural_numbers")
[1] TRUE
> inherits(x, "numeric")
[1] TRUE
```

```r
what_am_i <- function(x, …) {
  UseMethod("what_am_i")
}
```

```r
what_am_i.triangular_numbers <- function(x, ...) {
  message("I'm triangular numbers")
  NextMethod("what_am_i")
}
```

```r
what_am_i.natural_numbers <- function(x, ...) {
  message("I'm natural numbers")
  NextMethod("what_am_i")
}
```

```r
what_am_i.numeric <- function(x, ...) {
  message("I'm numeric")
}
```

```
> what_am_i(x)
I'm triangular numbers
I'm natural numbers
I'm numeric
```

$$\begin{tikzcd}
\texttt{what\_am\_i} \arrow[d] \\
\texttt{what\_am\_i.triangular\_numbers} \arrow[d] \\
\texttt{what\_am\_i.natural\_numbers} \arrow[d] \\
\texttt{what\_am\_i.numeric}
\end{tikzcd}$$

Summary

- **Multiple classes** are allowed
- Use `inherits()` to **test for arbitrary classes**
- Use `NextMethod()` to **chain method calls**

Let's practice!