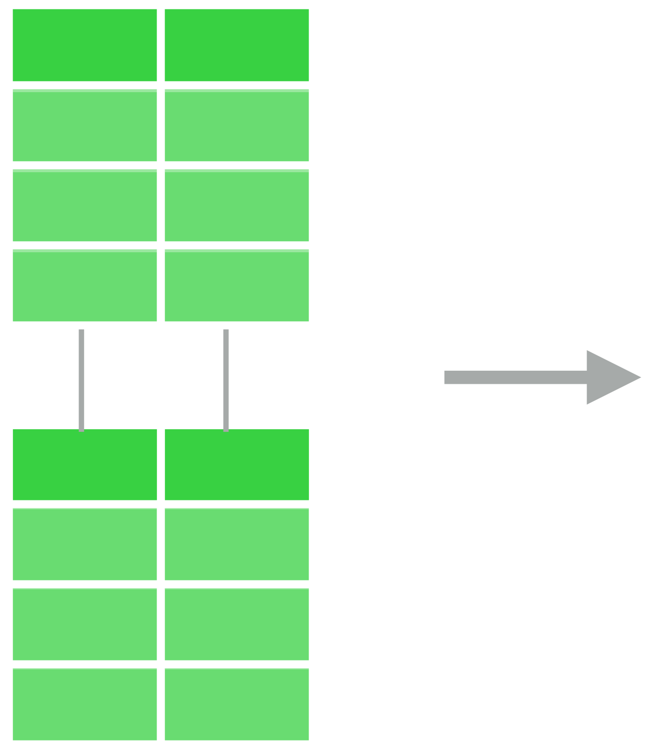
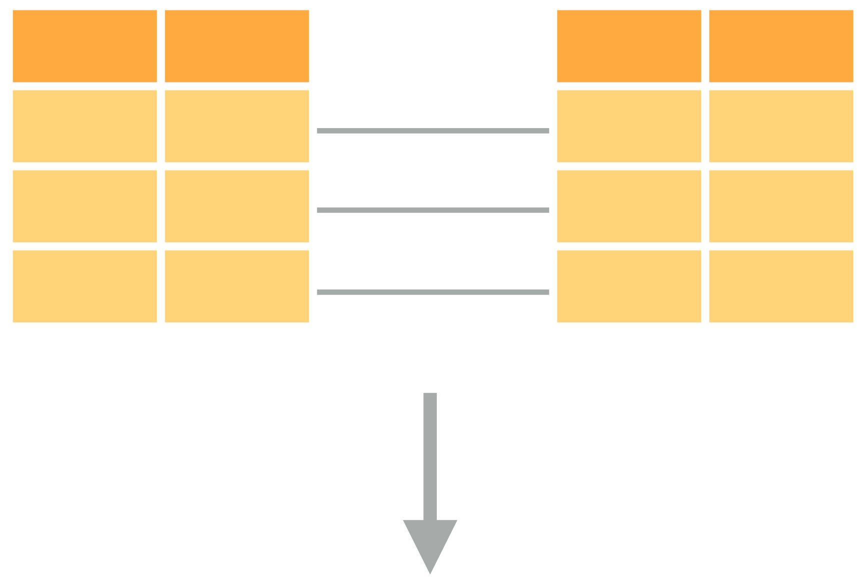
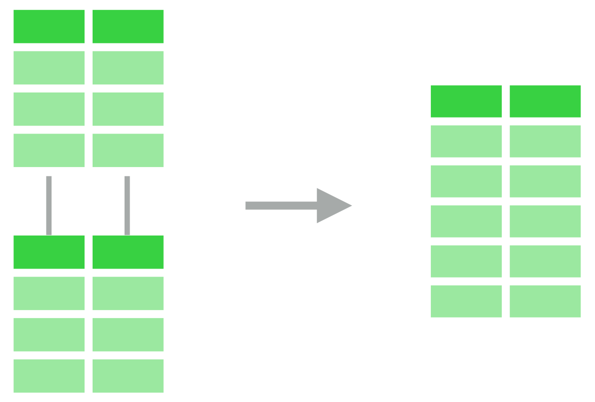
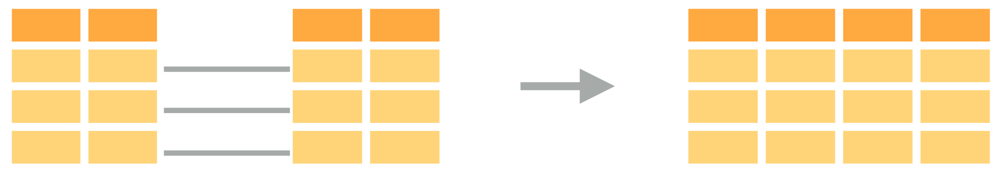
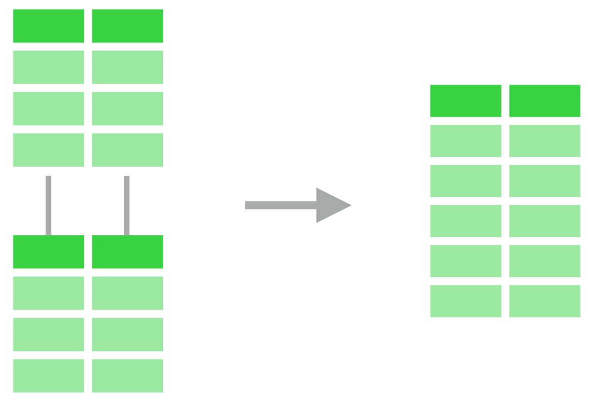
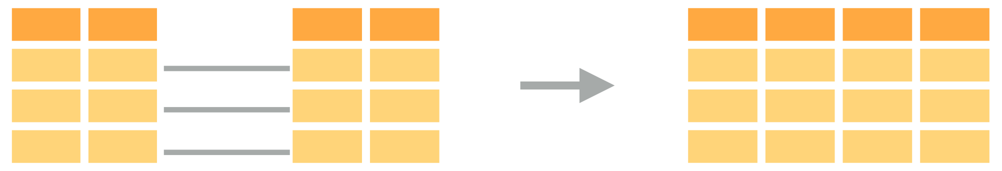
JOINING DATA IN R WITH DPLYR

Binds





- `rbind()` 
- `cbind()` 

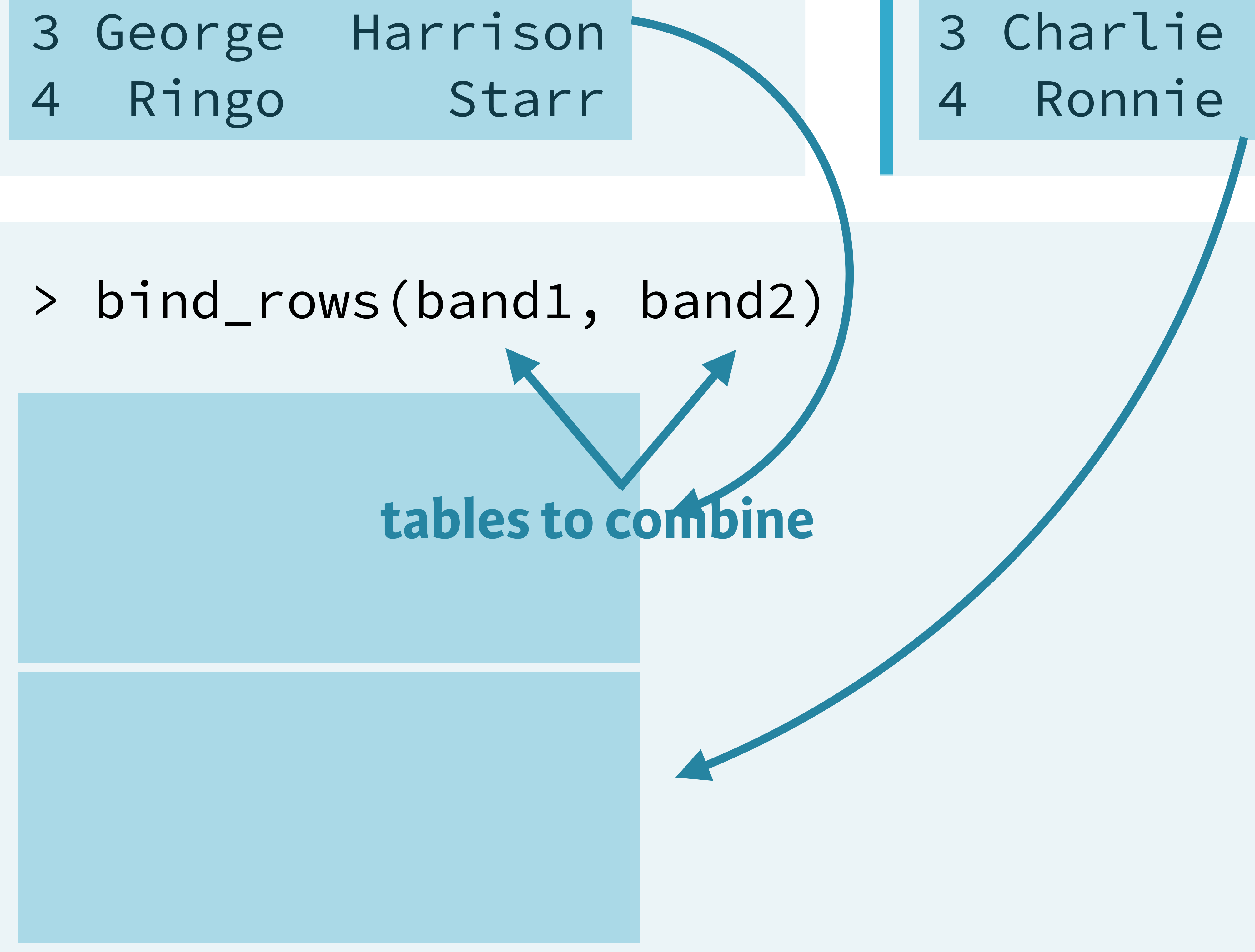
- `bind_rows()` 
- `bind_cols()` 

bind_rows()

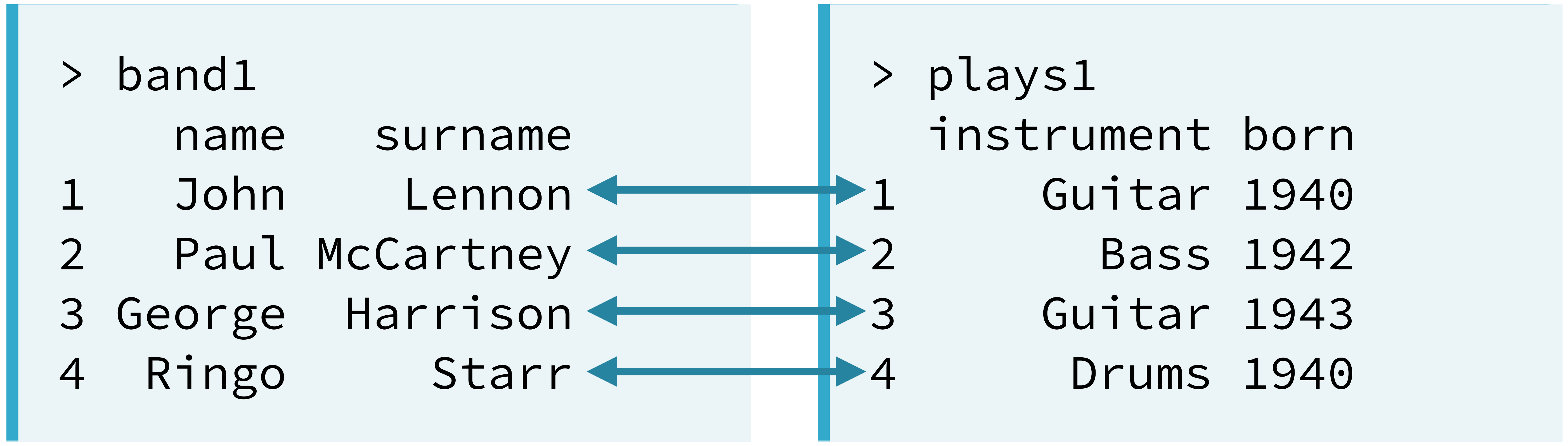
```
> band1
    name   surname
1   John    Lennon
2   Paul McCartney
3 George  Harrison
4  Ringo     Starr
```

```
> band2
     name  surname
1    Mick   Jagger
2   Keith Richards
3 Charlie    Watts
4  Ronnie     Wood
```

```
> bind_rows(band1, band2)
```



bind_cols()



```
> band1
     name   surname
1    John    Lennon
2    Paul McCartney
3 George  Harrison
4  Ringo     Starr
```

```
> plays1
  instrument born
1     Guitar 1940
2       Bass 1942
3     Guitar 1943
4      Drums 1940
```

```
> bind_cols(band1, plays1)
     name   surname instrument born
1    John    Lennon     Guitar 1940
2    Paul McCartney       Bass 1942
3 George  Harrison     Guitar 1943
4  Ringo     Starr      Drums 1940
```

Benefits of `bind_rows()` and `bind_cols()`

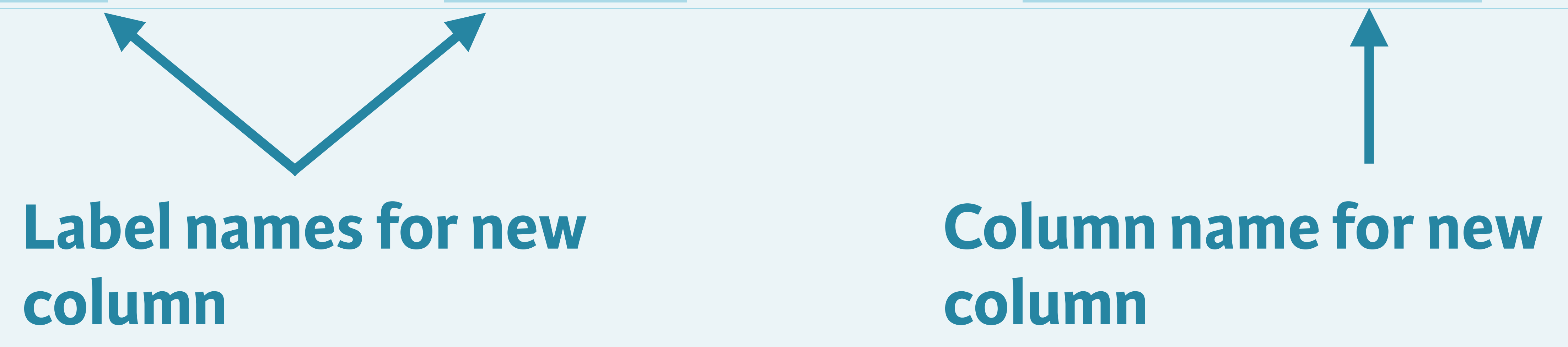
- Faster
- Return a tibble
- Can handle lists of data frames
- `.id`

bind_rows()

```
> band1
    name   surname
1   John    Lennon
2   Paul McCartney
3 George  Harrison
4  Ringo     Starr
```

```
> band2
     name  surname
1    Mick   Jagger
2   Keith Richards
3 Charlie    Watts
4  Ronnie     Wood
```

```
> bind_rows(Beatles = band1, Stones = band2, .id = "band")
```



Let's practice!

Build a better data frame

- `data.frame()`
- `as.data.frame()`

- `data_frame()`
- `as_data_frame()`

`data.frame()` defaults

- Changes strings to factors
- Adds row names
- Changes unusual column names

data_frame()

```
> data_frame(
+   Beatles = c("John", "Paul", "George", "Ringo"),
+   Stones = c("Mick", "Keith", "Charlie", "Ronnie"),
+   Zeppelins = c("Robert", "Jimmy", "John Paul", "John")
+ )
```

`data_frame()`

`data_frame()` will not...

- Change the data type of vectors (e.g. strings to factors)
- Add row names
- Change column names
- Recycle vectors greater than length one

data_frame()

- Evaluates arguments lazily, in order

```
> data_frame(
+   numbers = 1:5,
+   squares = numbers ^ 2
+ )
# A tibble: 5 × 2
  numbers squares
    <int>   <dbl>
1       1       1
2       2       4
3       3       9
4       4      16
5       5      25
```

- Returns a tibble

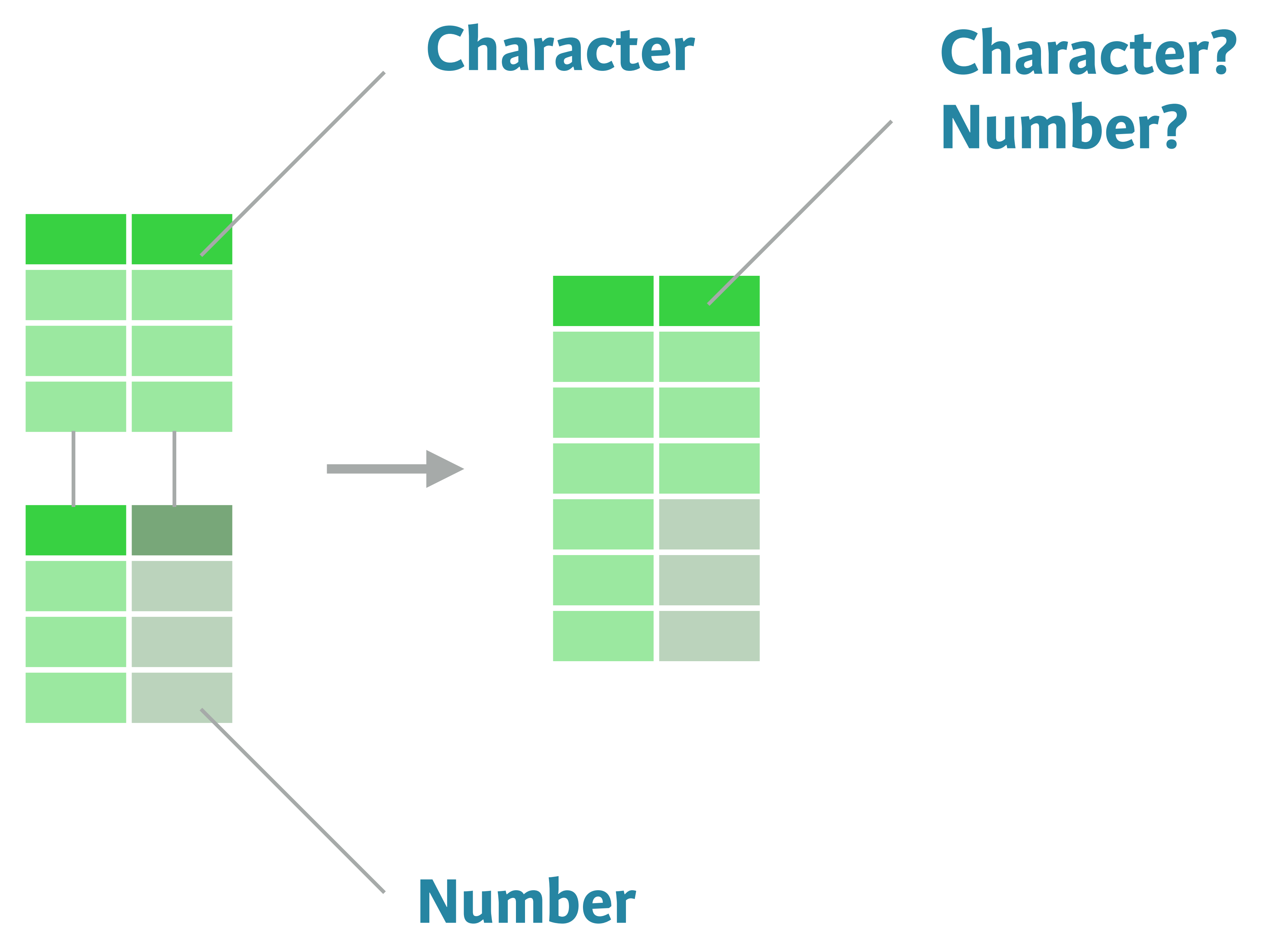
`as_data_frame()`

Let's practice!

Working with data types

```
> 1 + 1

> "one" + "one"
```


Character
Character?
Number?
Number

Atomic data types

```
> typeof(TRUE)            Logical
[1] "logical"

> typeof("hello")         Character (i.e. string)
[1] "character"

> typeof(3.14)            Double (i.e. numeric w/ decimal)
[1] "double"

> typeof(1L)              Integer (i.e. numeric w/o decimal)
[1] "integer"

> typeof(1 + 2i)          Complex
[1] "complex"

> typeof(raw(1))          Raw
[1] "raw"
```

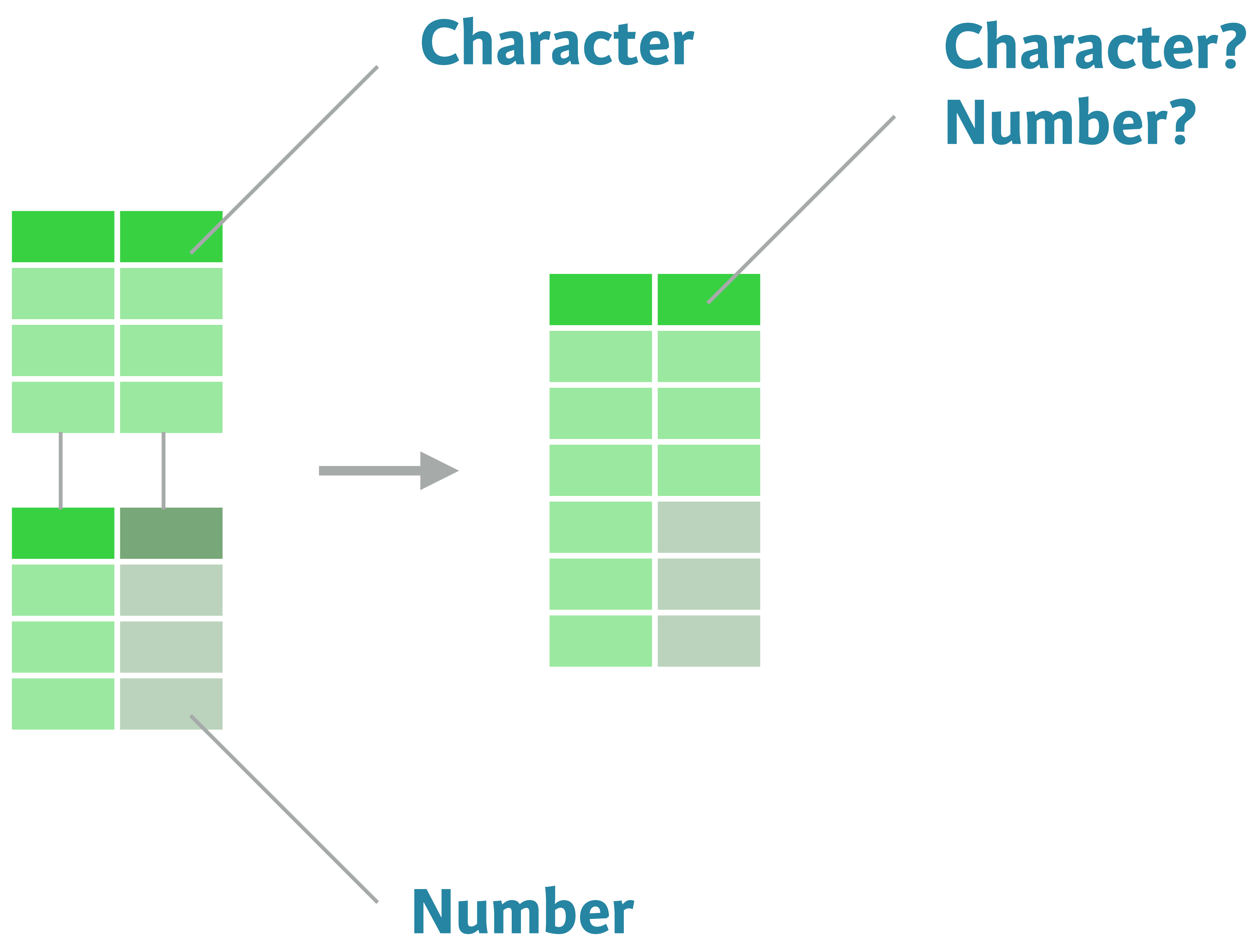
Classes

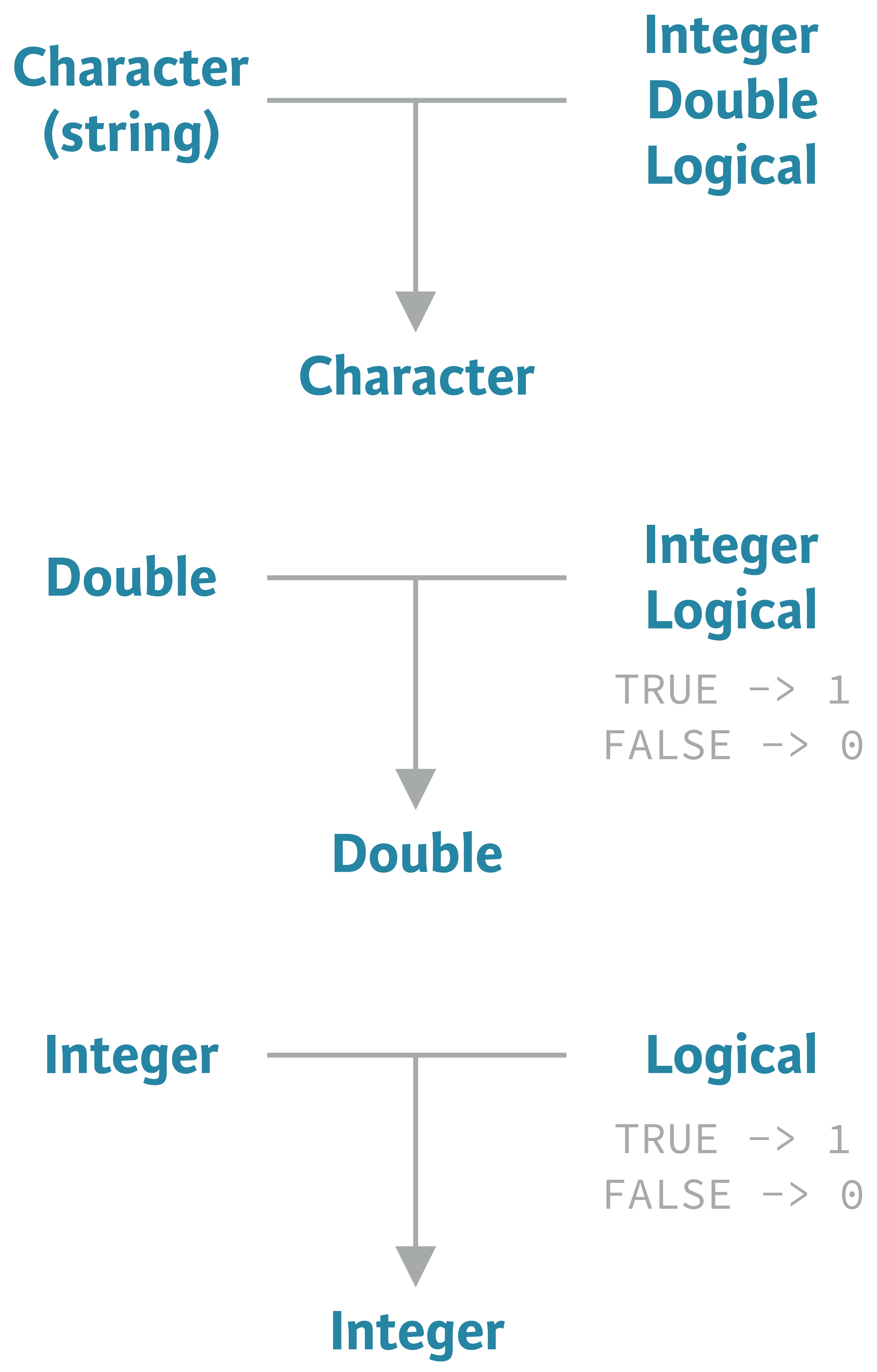
```
> x <- c(1L, 2L, 3L, 2L)
> x
[1] 1 2 3 2
> typeof(x)
[1] "integer"
> class(x)
[1] "integer"
```

```
1L = A
2L = B
3L = C
4L = D
```

Let's practice!

`dplyr`'s coercion rules


Character
Character?
Number?
Number



```
as.character()
```

```
as.numeric()
```

```
as.integer()
```

factors

```
# x is a factor
> x
[1] A B C B
Levels: A B C D

# How x is stored?
> unclass(x)
[1] 1 2 3 2
attr(,"levels")
[1] "A" "B" "C" "D"
```

```
> as.character(x)
[1] "A" "B" "C" "B"
> as.numeric(x)
[1] 1 2 3 2
```

factors

```
# y is a factor
> y <- factor(c(5, 6, 7, 6))
> y
[1] 5 6 7 6
Levels: 5 6 7

> unclass(y)
[1] 1 2 3 2
attr(,"levels")
[1] "5" "6" "7"
```

```
> as.character(y)
[1] "5" "6" "7" "6"
> as.numeric(y)
[1] 1 2 3 2
> as.numeric(as.character(y))
[1] 5 6 7 6
```

`dplyr`'s coercion behavior

- `dplyr` functions will not automatically coerce data types
    - Returns an error
    - Expects you to manually coerce data
- Exception: factors
    - `dplyr` converts non-aligning factors to strings
    - Gives warning message

Let's practice!