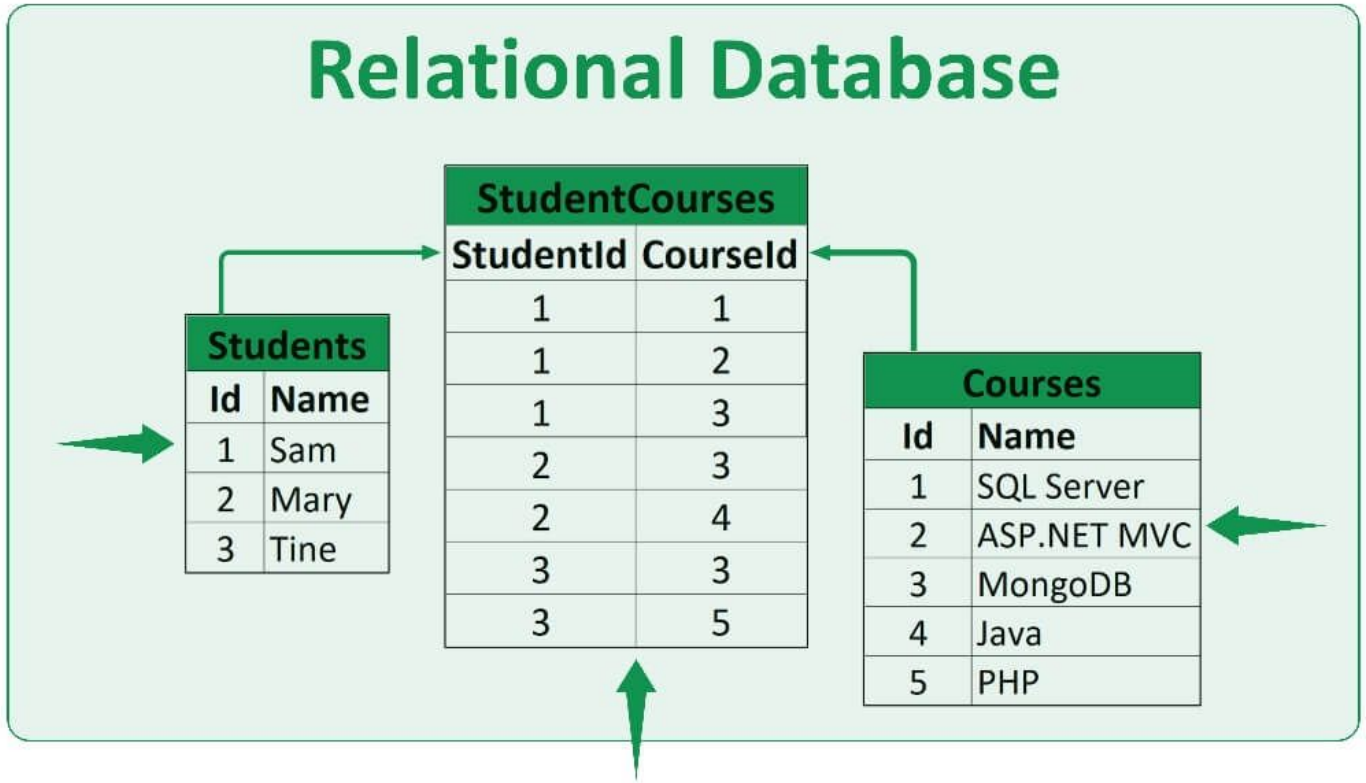
Afternoon stretch: How are these tables related?



Time, Dates, and Joins

Dates and Time

- tidyverse data types:
  - <date> : date, can take many formats
  - <time> : time, can include time zone
  - <dttm> : date plus time, a unique instant in time
    - Base R calls it POSIXct, but it is the same thing
- always use the simplest possible data type

```
flights |>
  select(year, month, day, hour, minute) |>
  mutate(departure = make_datetime(year, month, day, hour, minute))
#> # A tibble: 336,776 × 6
#>    year month   day  hour minute departure
#>   <int> <int> <int> <dbl>  <dbl> <dttm>
#> 1  2013     1     1     5     15 2013-01-01 05:15:00
#> 2  2013     1     1     5     29 2013-01-01 05:29:00
#> 3  2013     1     1     5     40 2013-01-01 05:40:00
```

Math with Date-times — Lubridate provides three classes of timespans to facilitate math with dates and date-times.

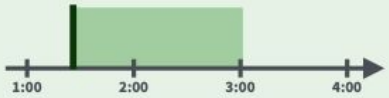
Math with date-times relies on the **timeline**, which behaves inconsistently. Consider how the timeline behaves during:

A normal day

```
nor <- ymd_hms("2018-01-01 01:30:00",tz="US/Eastern")
```

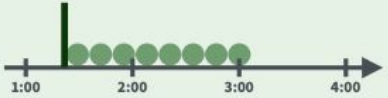
Periods track changes in clock times, which ignore time line irregularities.

nor + minutes(90)



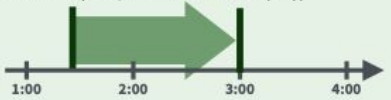
Durations track the passage of physical time, which deviates from clock time when irregularities occur.

nor + dminutes(90)



Intervals represent specific intervals of the timeline, bounded by start and end date-times.

interval(nor, nor + minutes(90))



Math with Date-times — Lubridate provides three classes of timespans to facilitate math with dates and date-times.

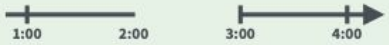
Math with date-times relies on the **timeline**, which behaves inconsistently. Consider how the timeline behaves during:

A normal day

```
nor <- ymd_hms("2018-01-01 01:30:00",tz="US/Eastern")
```

The start of daylight savings (spring forward)

```
gap <- ymd_hms("2018-03-11 01:30:00",tz="US/Eastern")
```



The end of daylight savings (fall back)

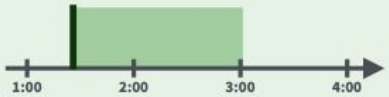
```
lap <- ymd_hms("2018-11-04 00:30:00",tz="US/Eastern")
```

Leap years and leap seconds

```
leap <- ymd("2019-03-01")
```

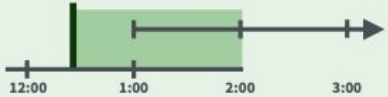
Periods track changes in clock times, which ignore time line irregularities.

nor + minutes(90)

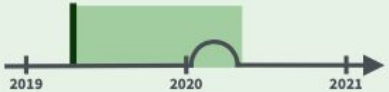


gap + minutes(90)

lap + minutes(90)

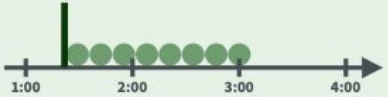


leap + years(1)



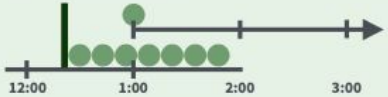
Durations track the passage of physical time, which deviates from clock time when irregularities occur.

nor + dminutes(90)



gap + dminutes(90)

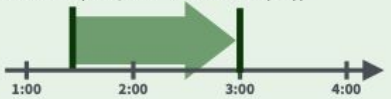
lap + dminutes(90)



leap + dyears(1)

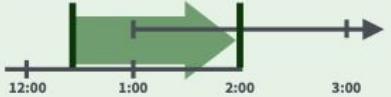
Intervals represent specific intervals of the timeline, bounded by start and end date-times.

interval(nor, nor + minutes(90))



interval(gap, gap + minutes(90))

interval(lap, lap + minutes(90))



interval(leap, leap + years(1))

Unix Epoch: January 1, 1970

The way R is able to add or subtract dates is by converting it to time since the Unix epoch.

Returned when converting date class to numeric class.

Give it a try, what do these calls return?

Call	Returns
Sys.time()	
today()	
as.numeric(Sys.time())	
as.numeric(today())	

Unix Epoch: January 1, 1970

The way R is able to add or subtract dates is by converting it to time since the Unix epoch.

Returned when converting date class to numeric class.

Give it a try, what do these calls return?

Call	Returns
Sys.time()	“YYYY-MM-DD HH:MM:SS TZ”
today()	“YYYY-MM-DD”
as.numeric(Sys.time())	Seconds since epoch
as.numeric(today())	Days since epoch

ISO8601: international standard for writing dates from biggest to smallest separated by “-”

Standardizing Non- ISO8601 Formats

- Lot's of different ways to express dates and time
  - MM/DD/YY
  - DD - MM - YY
  - DD Month YY
  - AM/PM
  - 24 HR
  - Time zones
  - Daylight savings
  - …
- R can manipulate all of these types of dates by parsing the strings to create a uniform format ready for analysis

Non-ISO8601 Formats

- Lot's of different ways to express dates and time
  - MM/DD/YY
  - DD - MM - YY
  - DD Month YY
  - AM/PM
  - 24 HR
  - Time zones
  - Daylight savings
  - …
- R can manipulate all of these types of dates by parsing a string to create a uniform format ready for analysis

Defining Custom Formats

Similar to regex, there is a syntax to express the format of your data.

How would you express:

1. May 30, 1985
2. 05/30/85
3. 1985-05-30

Type	Code	Meaning	Example
Year	%Y	4 digit year	2021
	%y	2 digit year	21
Month	%m	Number	2
	%b	Abbreviated name	Feb
	%B	Full name	February
Day	%d	One or two digits	2
	%e	Two digits	02
Time	%H	24-hour hour	13
	%I	12-hour hour	1
	%p	AM/PM	pm
	%M	Minutes	35
	%S	Seconds	45
	%0S	Seconds with decimal component	45.35
	%Z	Time zone name	America/Chicago
	%z	Offset from UTC	+0800

https://r4ds.hadley.nz/datetimes#tbl-date-formats

Defining Custom Formats

Similar to regex, there is a syntax to express the format of your data.

How would you express:

May 30, 1985	%B %e %Y
05/30/85	%m/%d/%y
1985-05-30	%Y-%m-%d

https://r4ds.hadley.nz/datetimes#tbl-date-formats

Type	Code	Meaning	Example
Year	%Y	4 digit year	2021
	%y	2 digit year	21
Month	%m	Number	2
	%b	Abbreviated name	Feb
	%B	Full name	February
Day	%d	One or two digits	2
	%e	Two digits	02
Time	%H	24-hour hour	13
	%I	12-hour hour	1
	%p	AM/PM	pm
	%M	Minutes	35
	%S	Seconds	45
	%0S	Seconds with decimal component	45.35
	%Z	Time zone name	America/Chicago
	%z	Offset from UTC	+0800

Questions and switch to M6.R

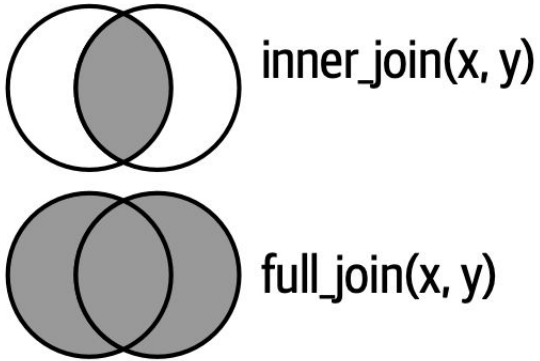
Joins: Because data is never in just one place

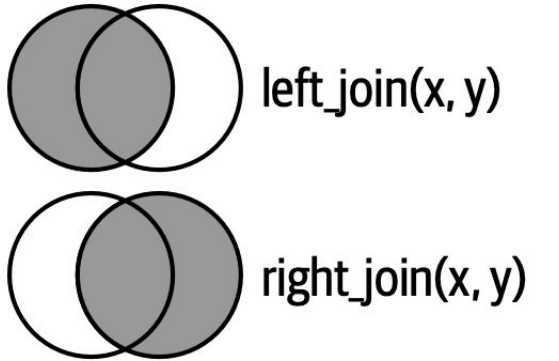
Joins are a super useful diagnostic tool if you have a strong conceptual understanding of how they work

Conceptual explanations often use diagrams like:

x	
1	x1
2	x2
3	x3

y	
1	y1
2	y2
4	y3





Joins: Because data is never in just one place

Combining data sets

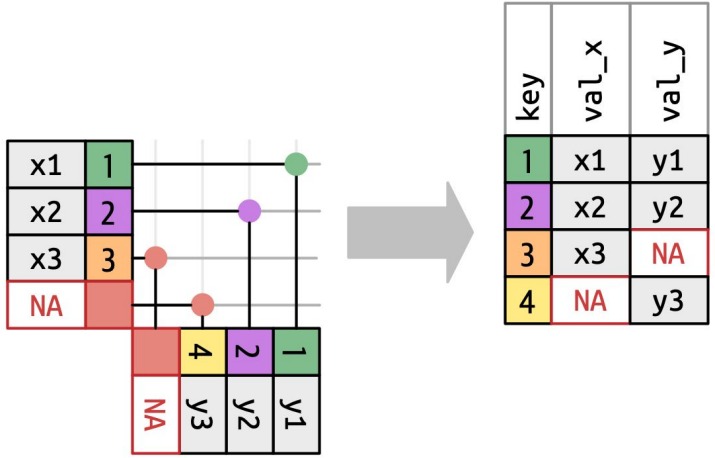
x

1	x1
2	x2
3	x3

y

1	y1
2	y2
4	y3

Mutation Joins: add new variables to one data frame by matching observations in another.



The key(s) is used to decide which observations (rows) go where.

Joins: Because data is never in just one place

Combining data sets

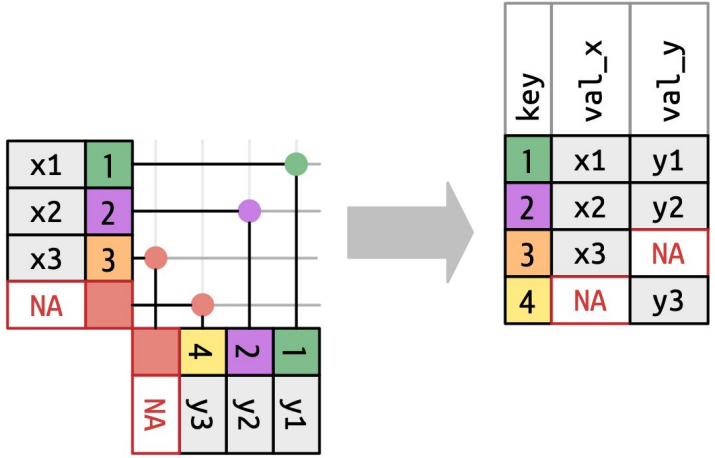
x

1	x1
2	x2
3	x3

y

1	y1
2	y2
4	y3

Mutation Joins: add new variables to one data frame by matching observations in another.



The key(s) is used to decide which observations (rows) go where.

What do you think would happen if:

- you had a typo in a key?
- a key matched multiple observations?

Joins: Because data is never in just one place

Combining data sets

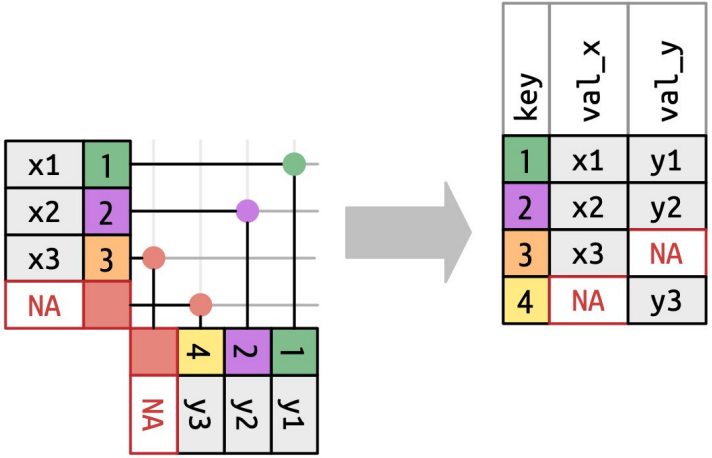
x

1	x1
2	x2
3	x3

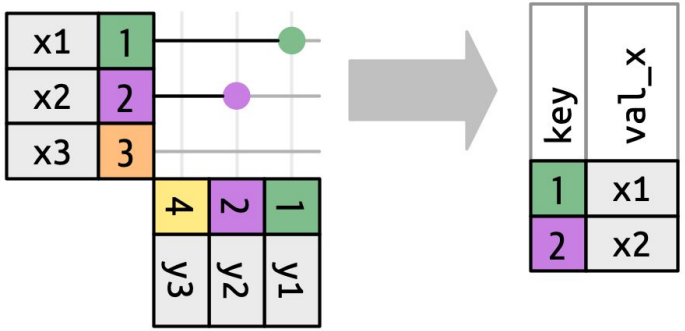
y

1	y1
2	y2
4	y3

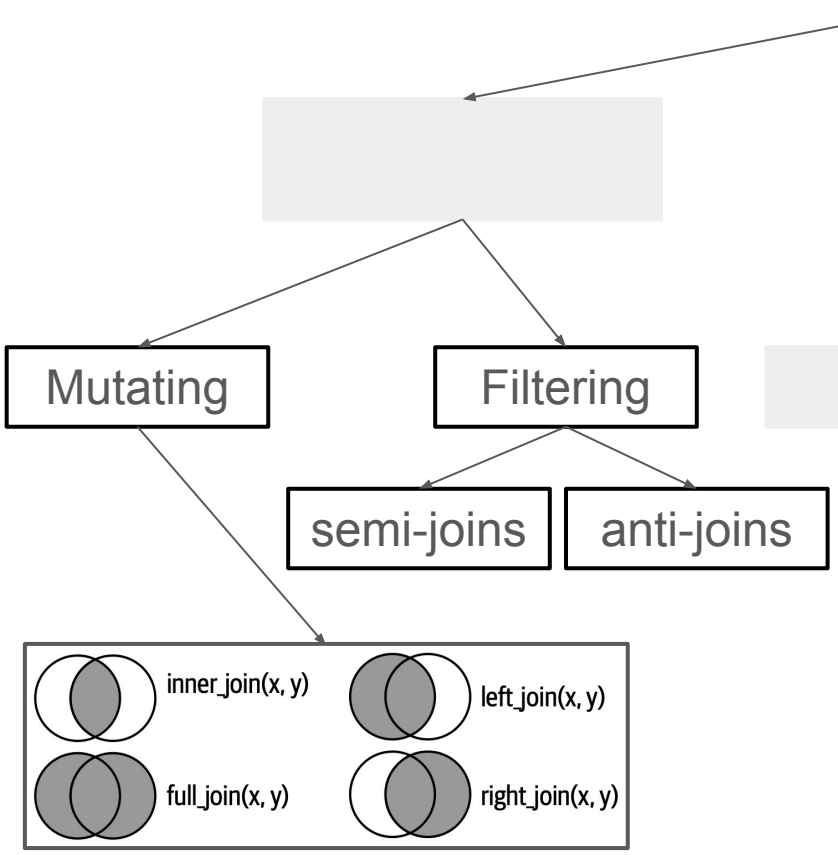
Mutation Joins: add new variables to one data frame by matching observations in another.



Filtering Joins: filter observations from one data frame based on whether or not they match an observation in another.



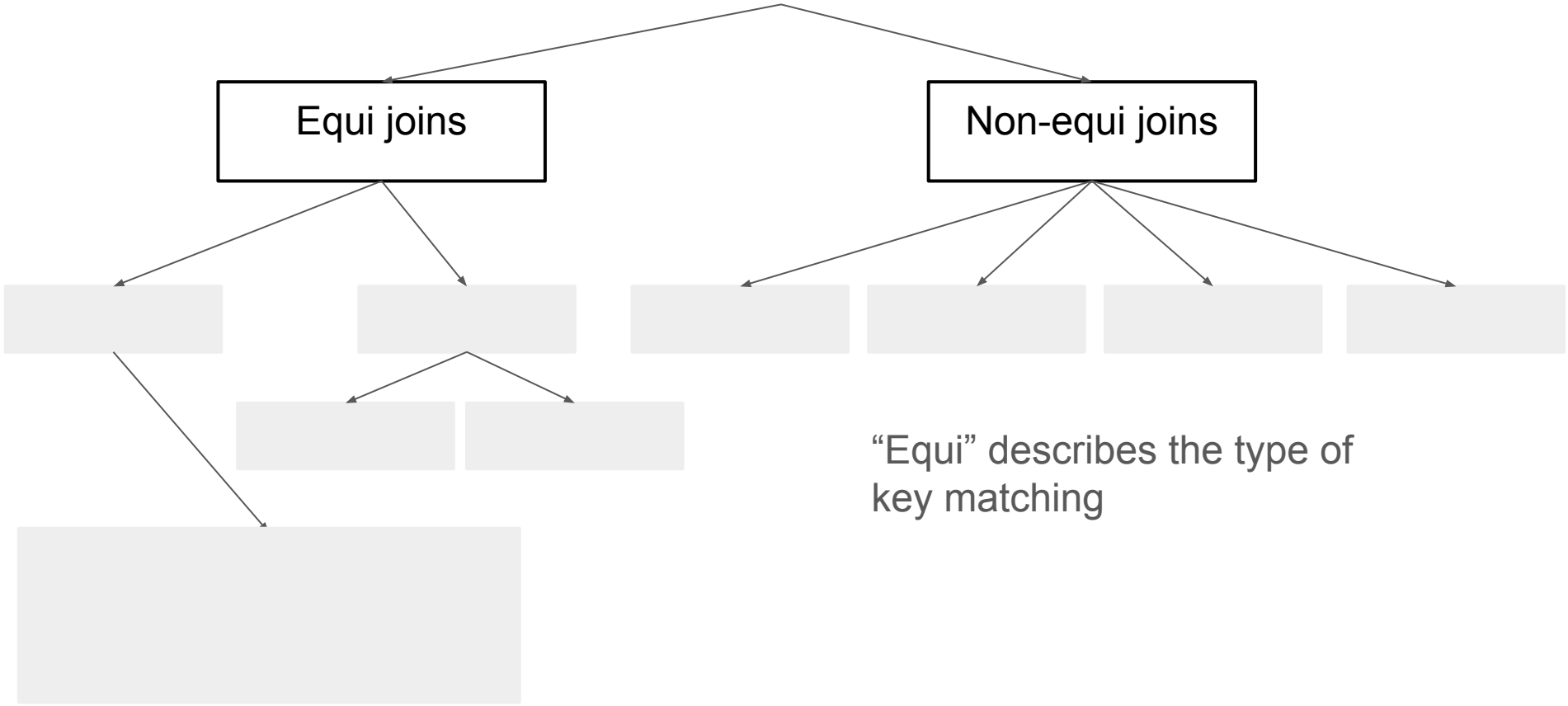
Types of Joins



Can filter on those that match or those that don't match

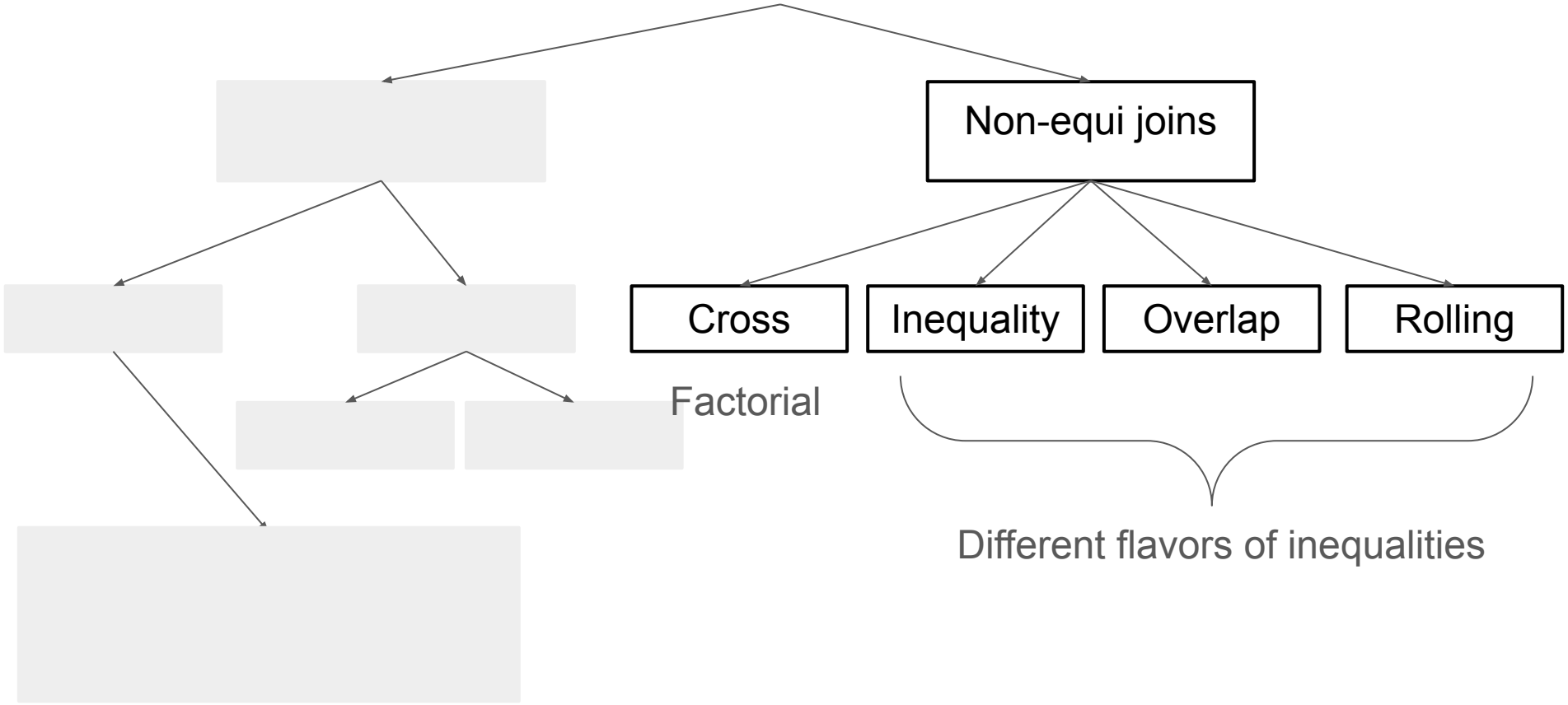
Super useful when mutating join is giving unexpected results

Types of Joins



“Equi” describes the type of key matching

Types of Joins



Types of Joins

- Equi joins
  - Mutating
    - inner_join(x, y)
    - left_join(x, y)
    - full_join(x, y)
    - right_join(x, y)
  - Filtering
    - semi-joins
    - anti-joins
- Non-equi joins
  - Cross
  - Inequality
  - Overlap
  - Rolling

Let’s Practice

X

Patient ID	Diag. Test
12	Y
25	N
72	Y

Y

Patient ID	Test Result
12	Positive
72	Negative
83	Negative

What is the key?

What is the output of an

inner_join(x,y)

anti_join(x,y)

anti_join(y,x)

full_join(x,y)

Questions and switch to M6.R