Data Types and Vectors in R

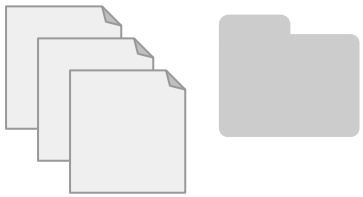
Stat 133 by Gaston Sanchez

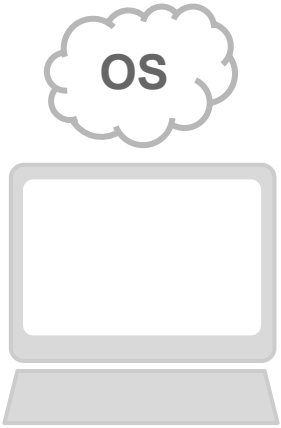
DCD
Data Computing Diagram

Data Sets

Software & Languages

Code, Scripts, Programs





Computers

Analyst /Scientist

We'll be working with "Data"

How do statisticians / analysts think of data?

How do computers treat data?

How do data sets get stored?

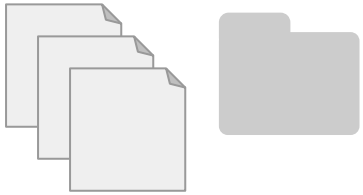
How do programs "understand" data?

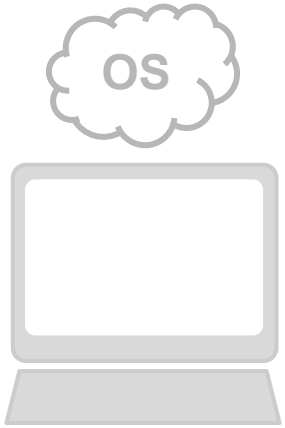
How do statisticians/scientists think about data?

Software &
Languages

Data
Sets

Code, Scripts,
Programs





Questions
Hypothesis

Computers

Ideas,
concepts,
abstractions

Analyst /Scientist

Data (as in Statistics)

Variables observed on some individuals

Variable: characteristic, feature, descriptor

Individual: objects, animals, humans, etc

Typical representation in tabular format

Some data set

first	last	gender	born	halfblood
Harry	Potter	male	1980	true
Hermione	Granger	female	1979	false
Luna	Lovegood	female	1980	true
Ron	Weasley	male	1981	false

Typical tabular form: rows for individuals, columns for variables

Referring to variables in statistics ...

Quantitative -vs- Qualitative

Continuous -vs- Discrete

Numerical -vs- Categorical

Scales: Ratio, Interval, Ordinal, Nominal

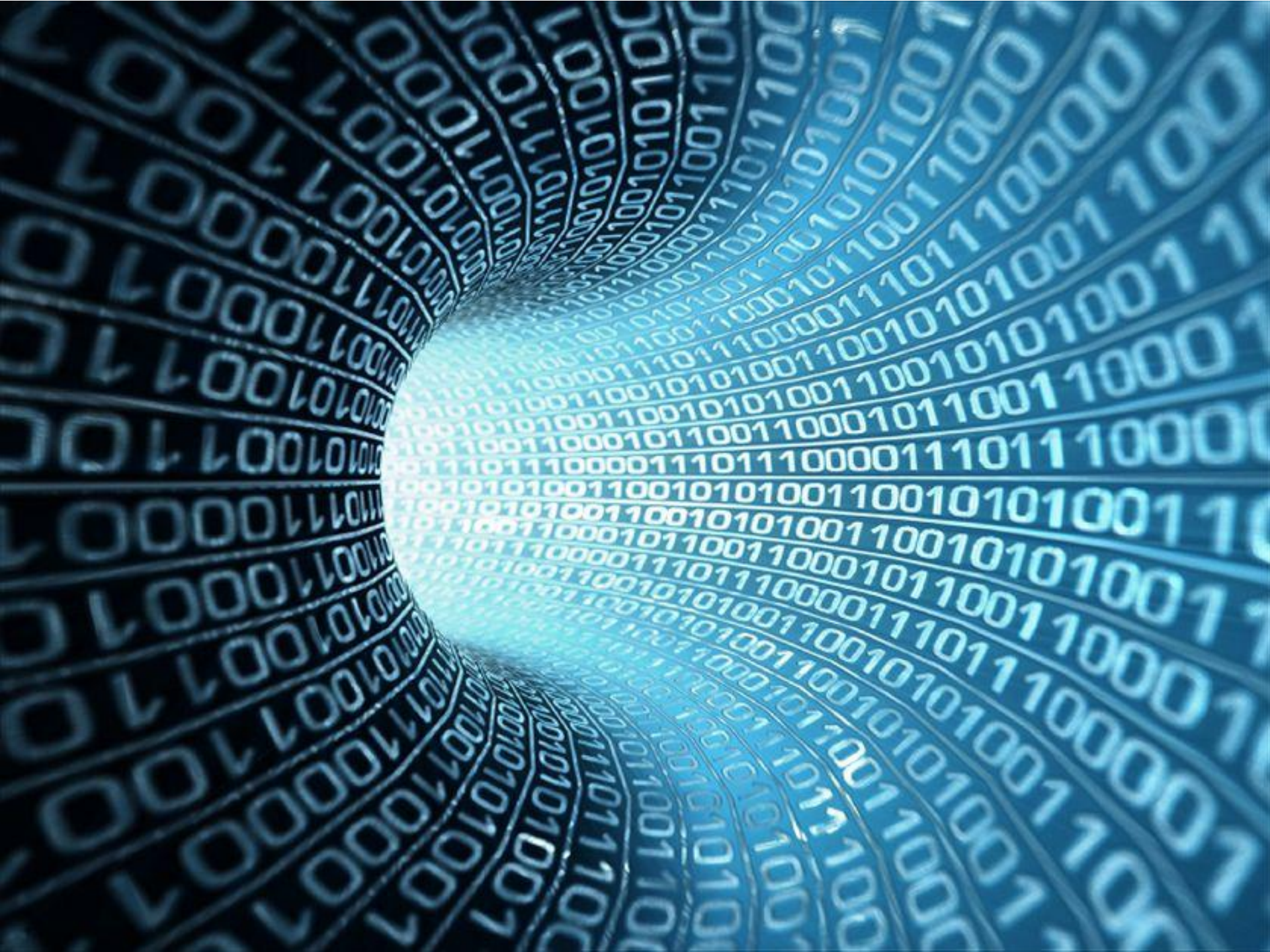
Dependent -vs- Independent

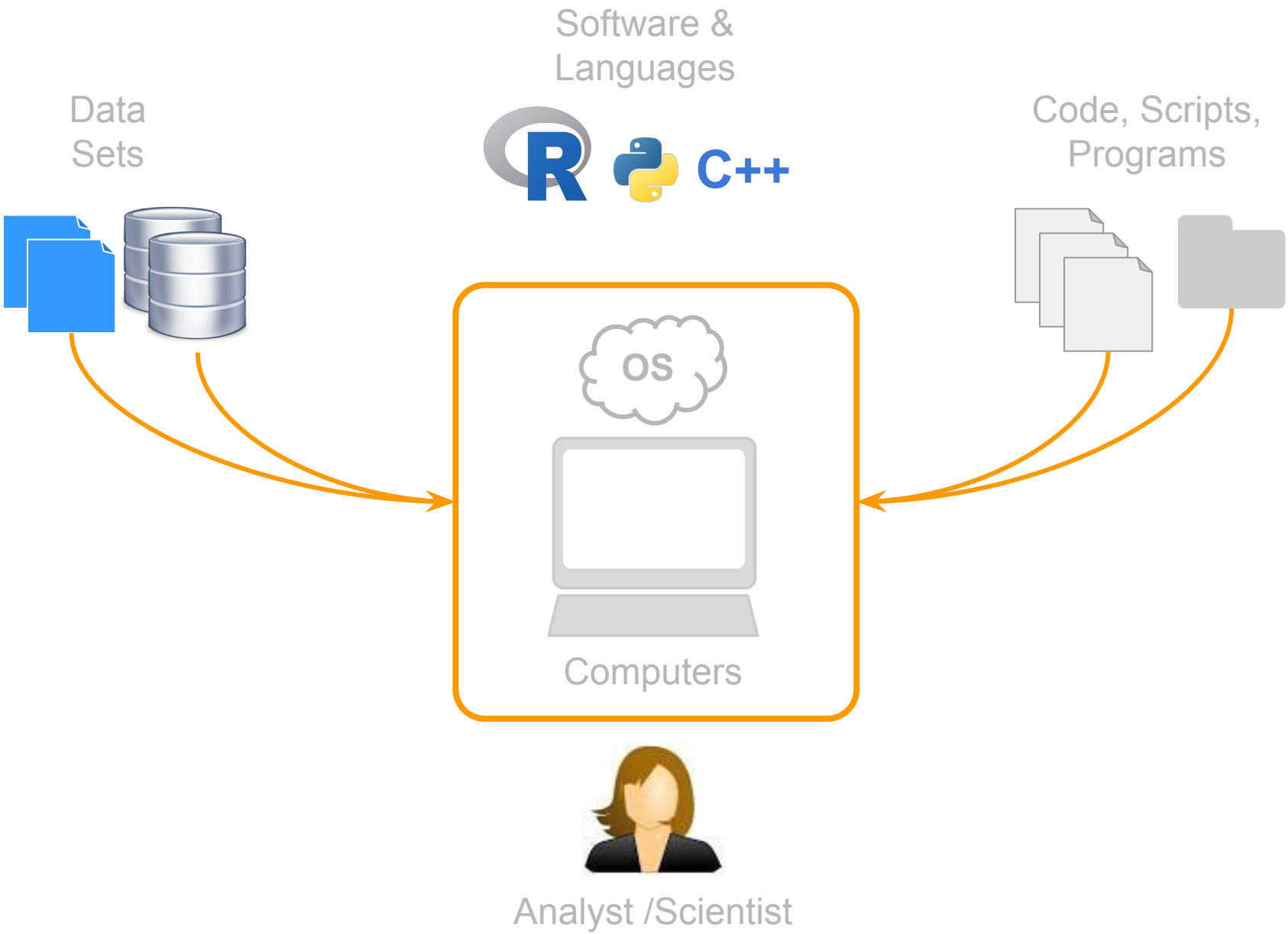
Descriptors (predictors) -vs- Response

Input -vs- Output

Missing values, Censored

How do computers treat data?



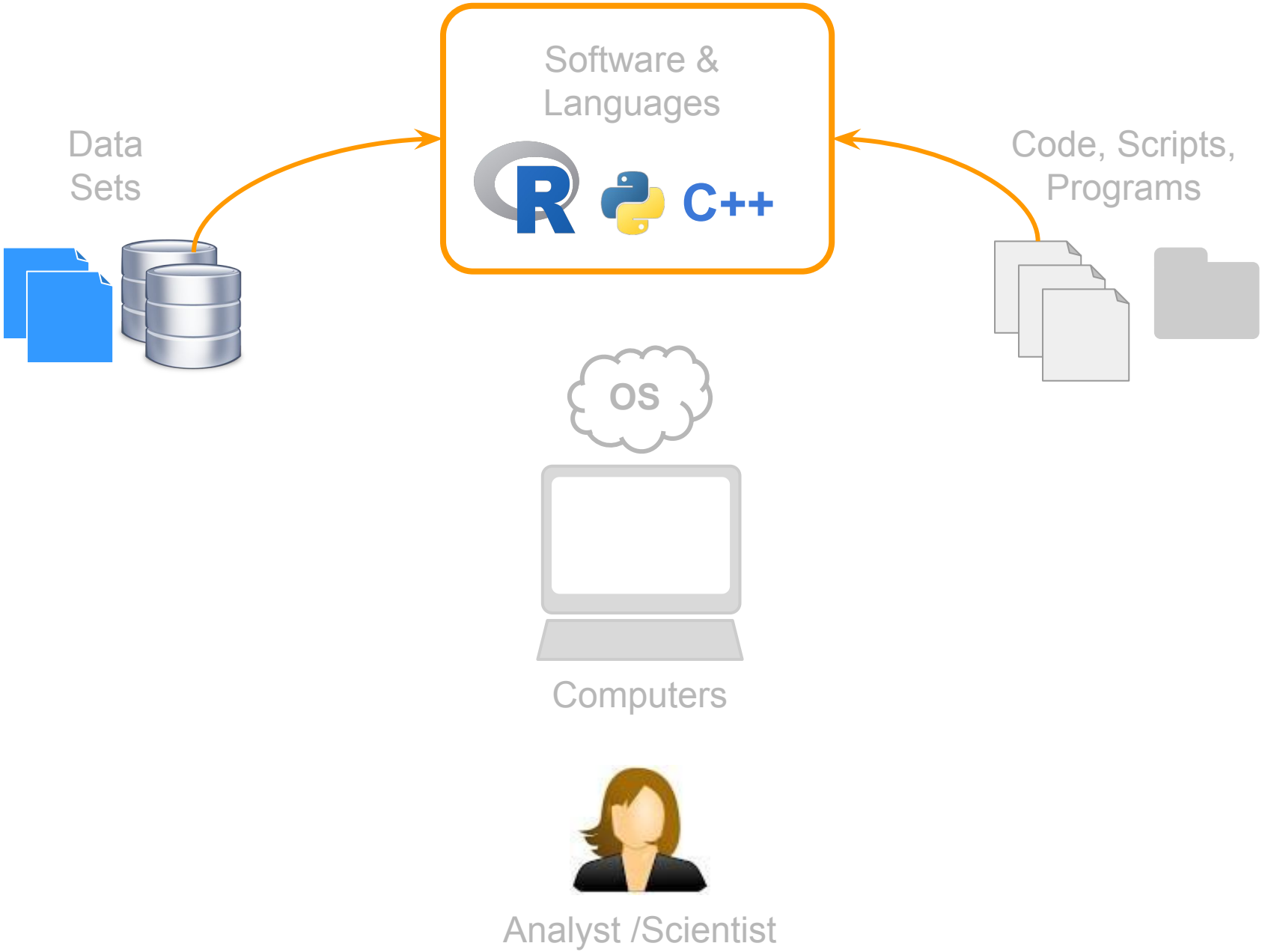

Software &
Languages
Data
Sets
Code, Scripts,
Programs
C++
OS
Computers
Analyst /Scientist

Data (for computers)

At the lowest level, computers treat all kinds of data in binary format:

0's and **1**'s

How do programming languages handle data?


Software &
Languages
C++
Data
Sets
Code, Scripts,
Programs
OS
Computers
Analyst /Scientist

Data for Software / Languages?

Data Types

Basic kinds

Data Structures

Containers

Data Types (for programming languages)

Also refer to as *data primitives* or primitive types

They serve as the building blocks (i.e. they are like the atoms)

Common Data Types (for programming languages)

- Integers (i.e. whole numbers)
- Real numbers (i.e. decimal numbers)
- Boolean (i.e. logical)
- Character (i.e. strings)

Common Data Types (for programming languages)

In many programming languages, everytime you create an object or a variable, you are forced to declare its type:

```
char first_name

int age
```

(you don't have to do this in R)

Data Types in R

Data types in R

- **Integer** (whole numbers)
- **Double** (real, decimal numbers)
- **Logical** (boolean)
- **Character** (or strings)
- **Complex (rarely used)*
- **Raw (rarely used)*

Data Types (primitives)

```
1L       # integer
2.5      # double (real)
TRUE     # logical
"hello"  # character
1 + 3i   # complex
```

Vectors in R

R vectors

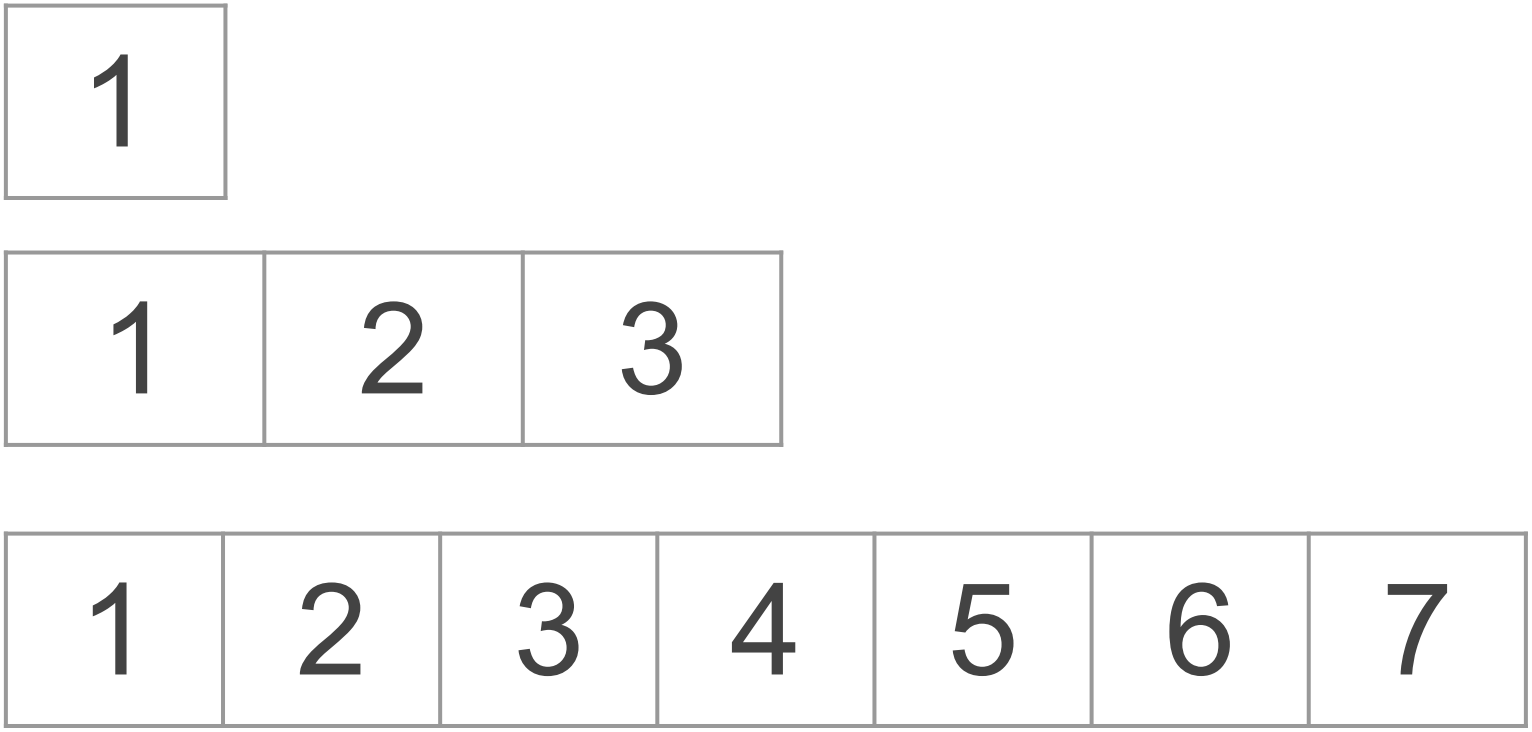
A vector is the **most basic** data structure in R

Vectors are contiguous cells containing data

1	2	3	4	5

R vectors

Can be of any length (including zero)



Different kinds of vectors

1	2	3	4	5

numeric

TRUE	FALSE	TRUE	FALSE

logical

"I"	"you"	"we"	"they"

character

Common *(and not so common*)* data types in R

An **integer** vector stores integers

A **double** vector stores regular (real) numbers

A **character** vector stores character strings

A **logical** vector stores TRUE and FALSE values

A **complex vector stores complex numbers*

A **raw vector stores raw bytes*

“Scalars” = one element vectors

```
x <- 1L       # integer
y <- 2.5      # real
z <- TRUE     # logical
w <- "hello"  # character
u <- 1 + 3i   # complex
```

Atomicity

Vectors are **atomic** structures

Examples

```
x <- c(1, 2, 3, 4, 5)

y <- c("one", "two", "three")

z <- c(TRUE, FALSE, TRUE)
```

Atomic vectors

Vectors are atomic structures

The values in a vector must be **ALL** of the same type!

Either all integers, or reals, or complex, or characters, of logicals

You CANNOT have a vector of different data types

R parlance: Types and Modes

The function `typeof()` returns the type of data: this is how the values are stored internally in R.

In S terminology, instead of talking about **types** we talk about **modes**.

The function `mode()` returns the "mode" of an R object.

Data types and modes

A bit confusing at the beginning

value	example	mode	type
integer	`1L, 2L`	`numeric`	`integer`
real	`1, -0.5`	`numeric`	`double`
complex	`3 + 5i`	`complex`	`complex`
logical	`TRUE, FALSE`	`logical`	`logical`
character	`"hello"`	`character`	`character`

You will typically be using the **mode**

Special Values

There are some special data values in R

`NULL` = null object

`NA` = Not available (missing value)

`Inf` = positive infinite

`-Inf` = negative infinite

`NaN` = Not a Number (different from NA)

Coercion

What happens if you mix different data values in a vector?

Mixing data types within a vector?

```
x <- c(1, 2, 3, "four", "five")
```

```
y <- c(TRUE, FALSE, 3, 4)
```

```
z <- c(TRUE, 1L, 2 + 3i, pi)
```

Implicit Coercion

If you mix different data values, R will **implicitly coerce** them so they are ALL of the same type

```
x <- c(1, 2, 3, "four", "five")
```

```
y <- c(TRUE, FALSE, 3, 4)
```

How does R coerce data types in vectors?

R follows two basic rules of implicit coercion

1) If a character is present, R will coerce everything else to characters

2) If a vector contains logicals and numbers, R will convert the logicals to numbers (TRUE to 1, FALSE to 0)

Coercion functions

R provides a set of explicit coercion functions that allow you to "convert" one type of data into another

- `as.character()`
- `as.numeric()`
- `as.integer()`
- `as.logical()`

Subsetting and Indexing

Bracket notation for vectors

`object[index]`

Bracket Notation System

To extract values from R objects use brackets: **[ ]**

Inside the brackets specify vector(s) of indices

Use as many indices, separated by commas, as dimensions in the object

Vector(s) of indices can be numbers, logicals, and sometimes characters

Bracket Notation System

```
# some vector
x <- c(2, 4, 6, 8)

# adding names
names(x) <- letters[1:4]
```

Numeric index

```
# first element
x[1]

# second element
x[2]

# last element
x[length(x)]
```

Numeric index

```
# first 3 elements
x[1:3]

# non-consecutive elements
x[c(1, 3)]

# different order
x[c(3, 2, 4, 1)]
```

Logical index

```
# first element
x[c(TRUE, FALSE, FALSE, FALSE)]

# elements equal to 2
x[x == 2]

# elements different to 2
x[x != 2]
```

Logical index

```
# elements greater than 1
x[x > 1]

# try this
x[TRUE]

# what about this?
x[as.logical(c(0, 1, pi, -10))]
```

Character index

```
# element names "a"
x["a"]

# "b" and "d"
x[c("b", "d")]

# what about this?
x[rep("a", 5)]
```

Vectorization

Vectorization

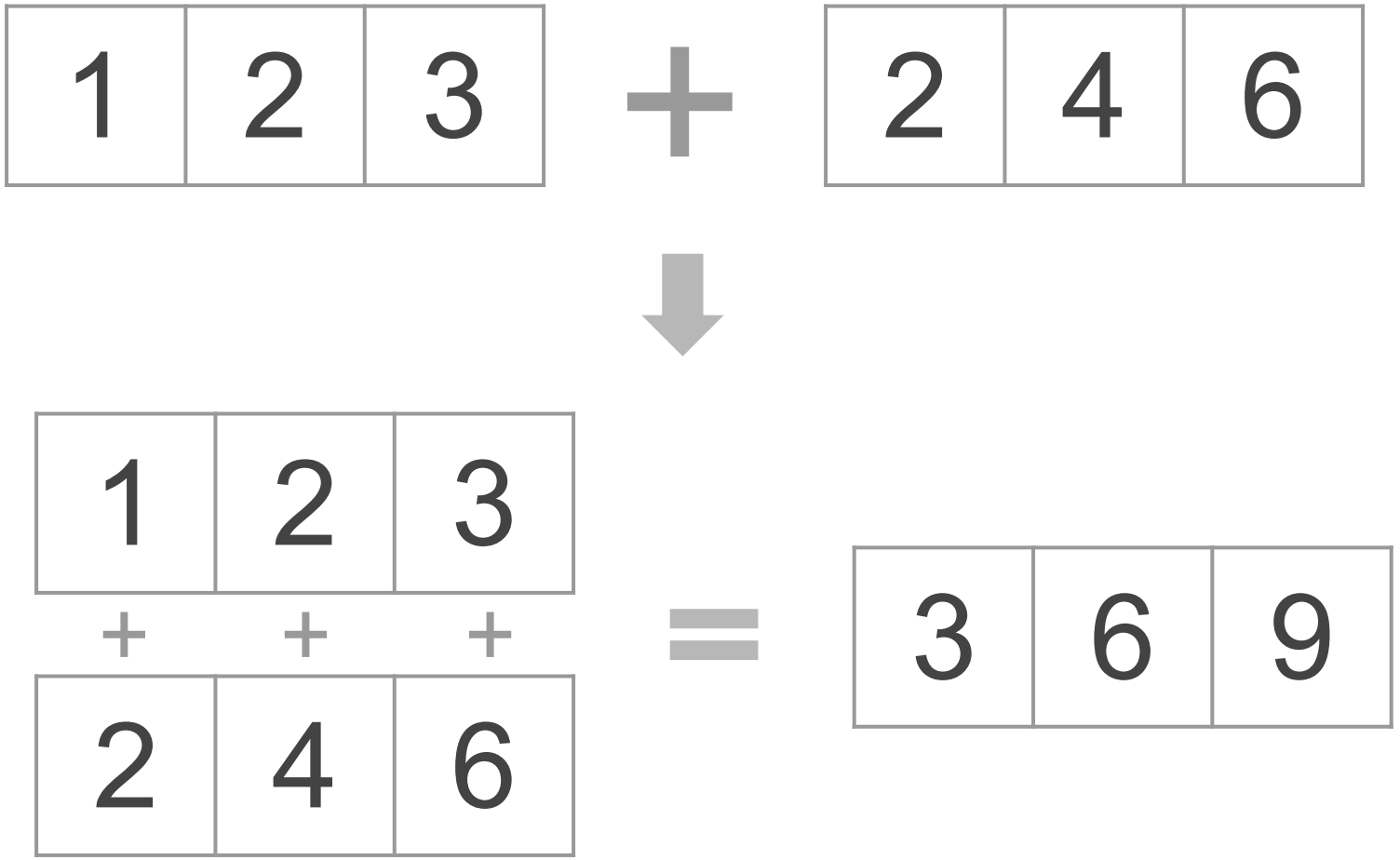
A **vectorized** computation is any computation that when applied to a vector operates on all of its elements

```
c(1, 2, 3) + c(3, 2, 1)

c(1, 2, 3) * c(3, 2, 1)

c(1, 2, 3) ^ c(3, 2, 1)
```

Vectorized code



Recycling

Recycling

When vectorized computations are applied, some problems may occur when dealing with two vectors of different length

```
c(2, 1) + c(1, 2, 3)
```

```
c(1, 2, 3, 4) + c(1, 2)
```

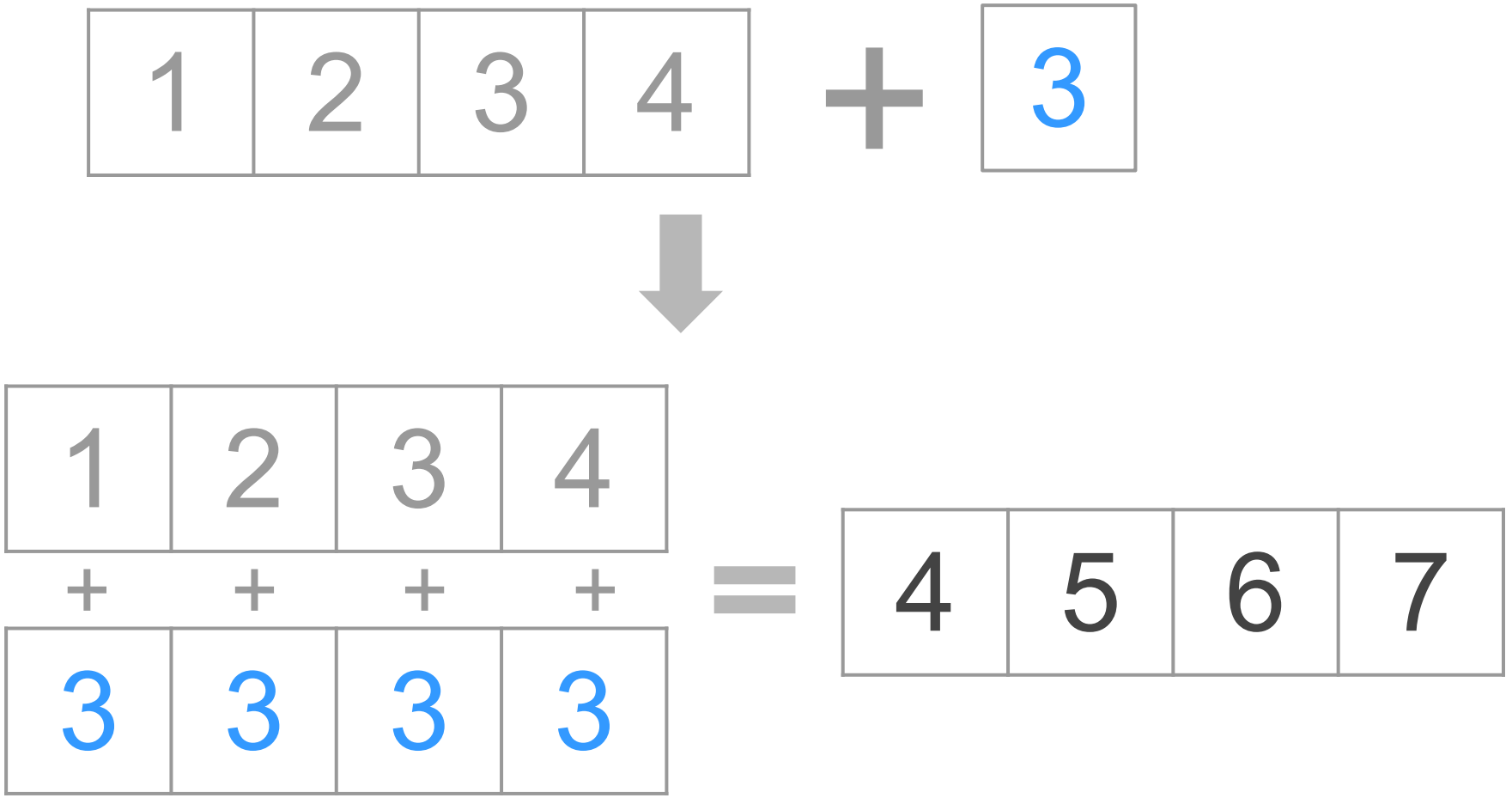
Recycling Rule

The recycling rule can be very useful, like when operating between a vector and a “scalar”

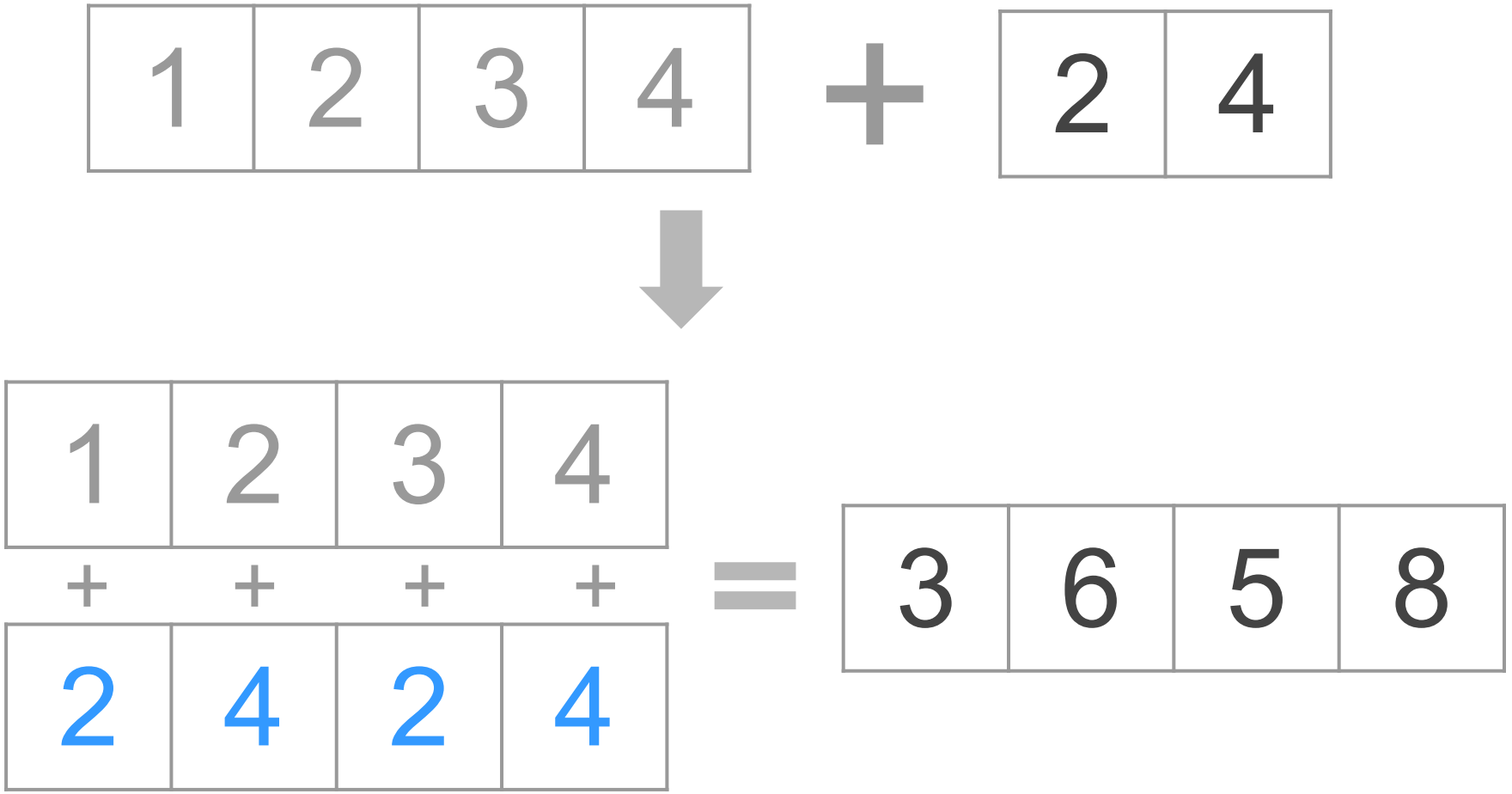
```
x <- c(2, 4, 6, 8)

x + 3
```

Recycling (and vectorization)



Recycling (and vectorization)



Recycling (and vectorization)