

Forms of Data

Mirco Schönfeld
University of Bayreuth

mirco.schoenfeld@uni-bayreuth.de
[@TWlY29](https://twitter.com/TWlY29)



Primitive!

The most basic data types:

- Integers 1, 2, 3, 1234, -5
- Floating Point Decimals 1.23, 2.34, -5.0
- Character 'a', 'b', 'c'
- Boolean True, False
- Strings "abc"



Complexity Increased

Complex data types can be constructed out of primitive data types.

Example: a date.

2020-11-24

More complex data structures can be constructed out of basic structures

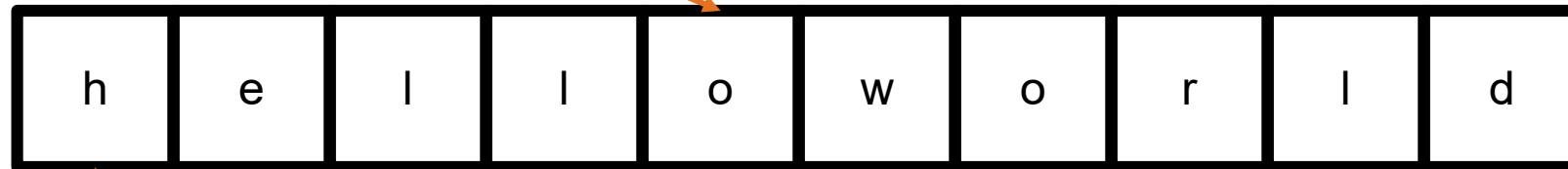
Array, one-dimensional

- One of the oldest and most important data structures:
Almost every program uses arrays at some point.
- Mostly some consecutive space in memory.
- Used to implement a lot of other data structures.
- Accessing elements requires a single subscript.

Indices are restricted to a consecutive range of integers: $B + c \times i$

```
my_array[4]  
> 'o'
```

i : element's index, zero-based



B : Base address

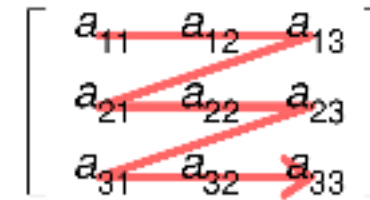
c : address increment

Array, multi-dimensional

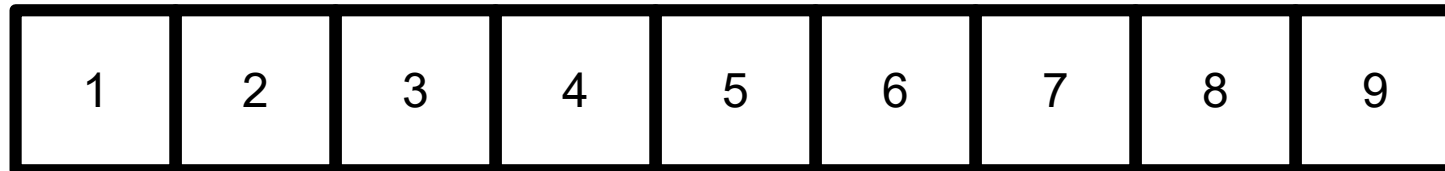
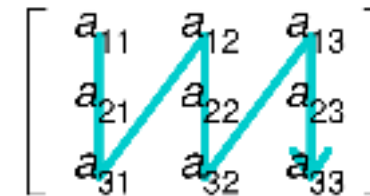
- Used to model a notion of matrices
- Still some consecutive space in memory
- Order either row-major or column-major

$my_matrix = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$
 `my_matrix[1][2]`
 `> 6`

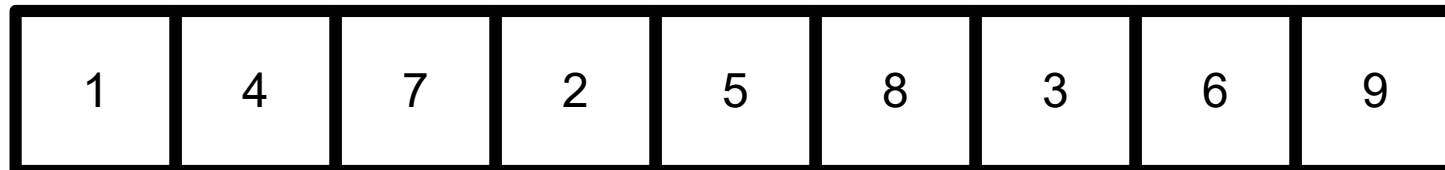
Row-major order



Column-major order



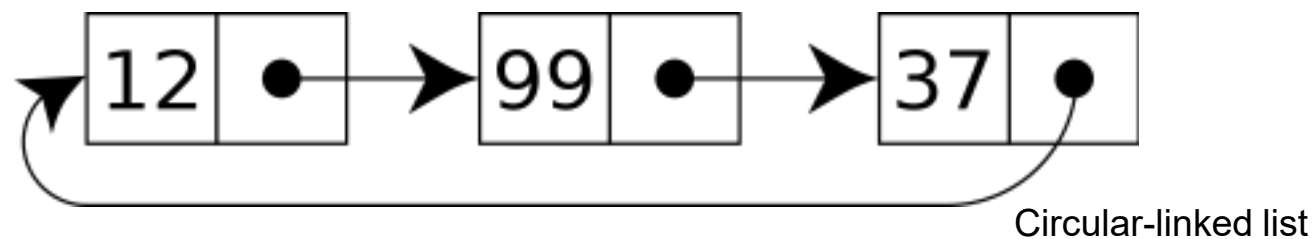
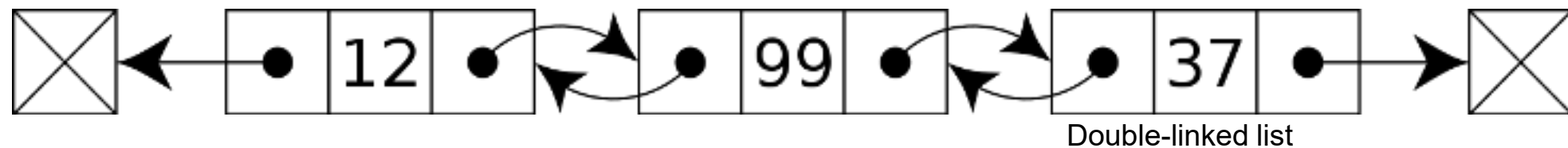
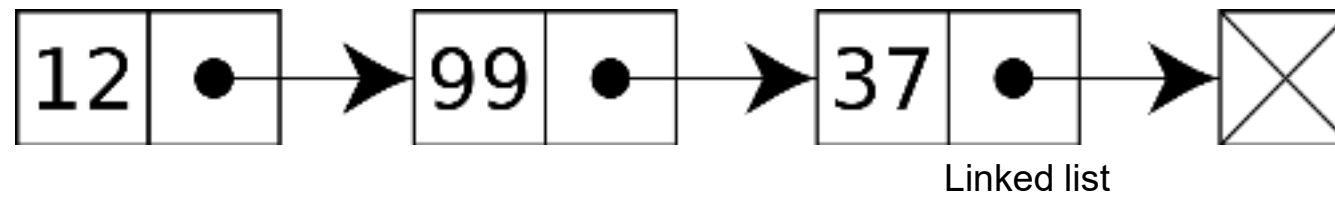
Row-major order



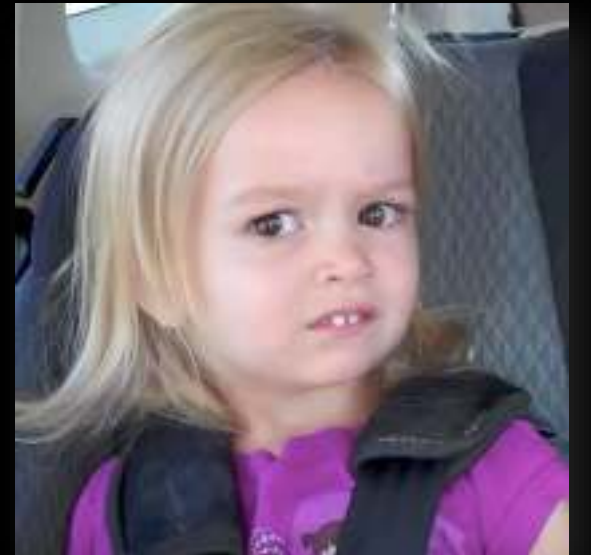
Column-major order

Lists

- A problem with arrays: Their consecutive space in memory is fixed. That makes insertion and deletion of elements a “costly operation” – sometimes the whole array has to be copied to a larger or smaller space in memory.
- Linked lists allow insertion and deletion at any point in the list



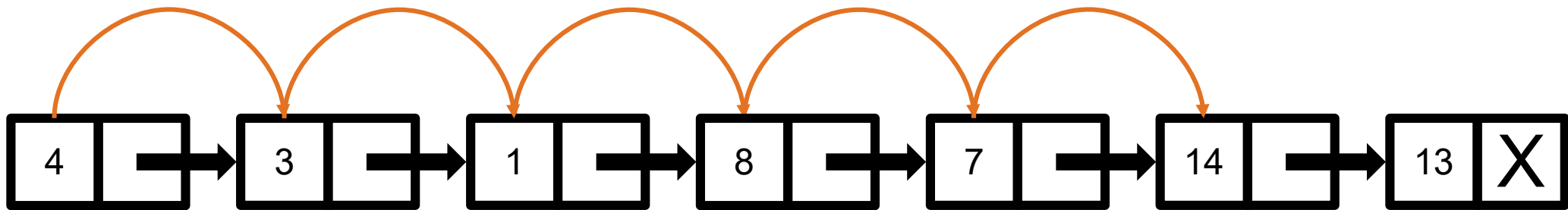
Why should we care?



Data structures make a difference

Choosing an appropriate data structure for your task will make a difference!

Example: Find the element '14'!



Data structure supports only to proceed to next element like a linked list.

Linear search – in worst case we have to visit all elements of the list!

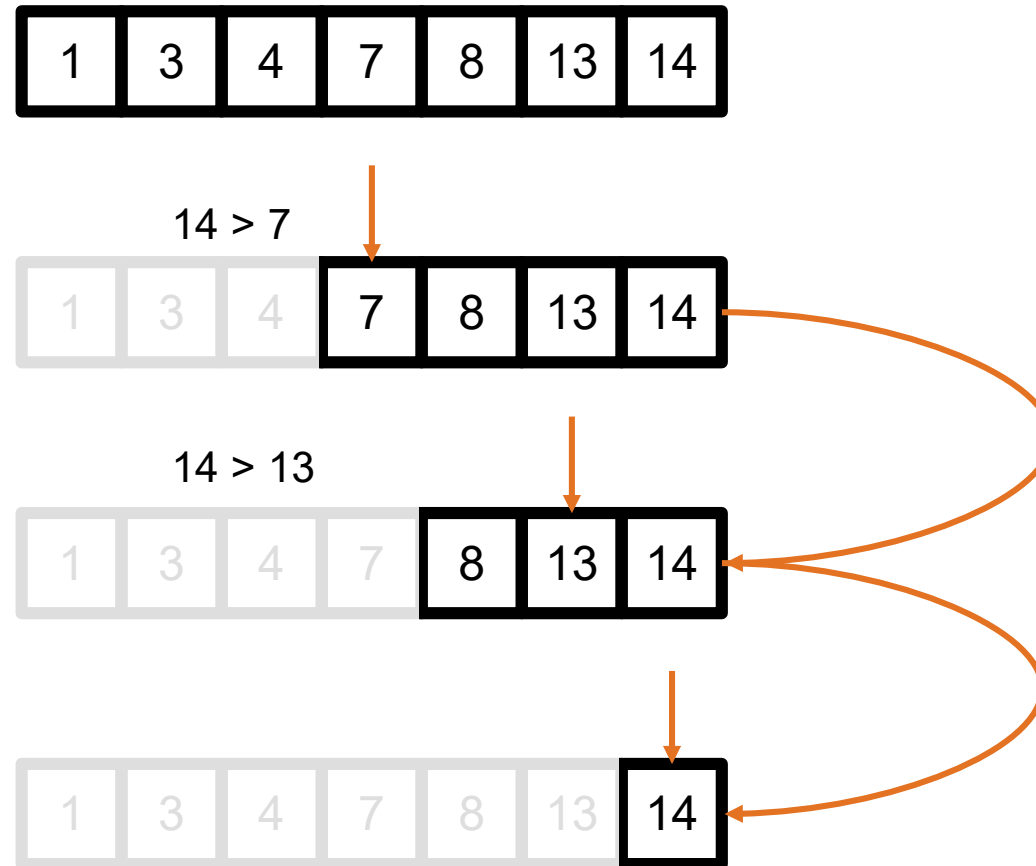
Sorting the list first doesn't increase lookup speed.

Data structures make a difference

Choosing an appropriate data structure for your task will make a difference!

Example: Find the element '14'!

Data structure supports accessing specific positions of the list like an array. This boils down to **binary search**. With each operation we can eliminate half of the remaining list. Even in worst case, we don't have to visit all elements in the list.

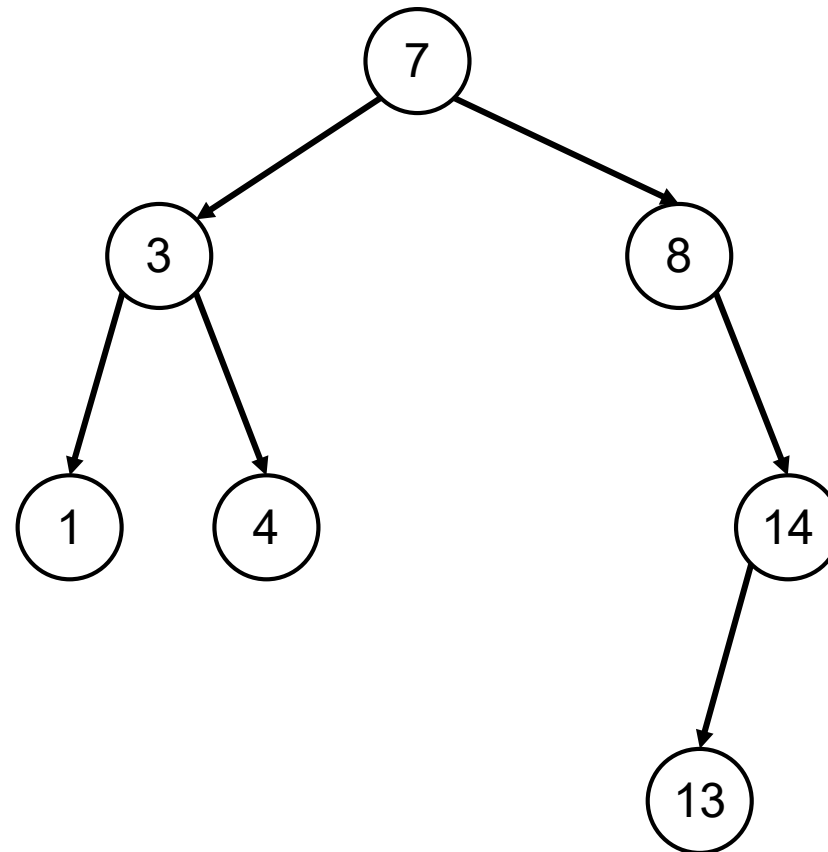


Data structures make a difference

Choosing an appropriate data structure for your task will make a difference!

Example: Find the element '14'!

For binary search, the array has to be sorted. The logical structure represented within the array is a binary search tree.



A binary search tree is a rooted binary tree data structure whose internal nodes each store a key greater than all the keys in the node's left subtree and less than those in its right subtree.

As a binary tree, each node has at most two children.

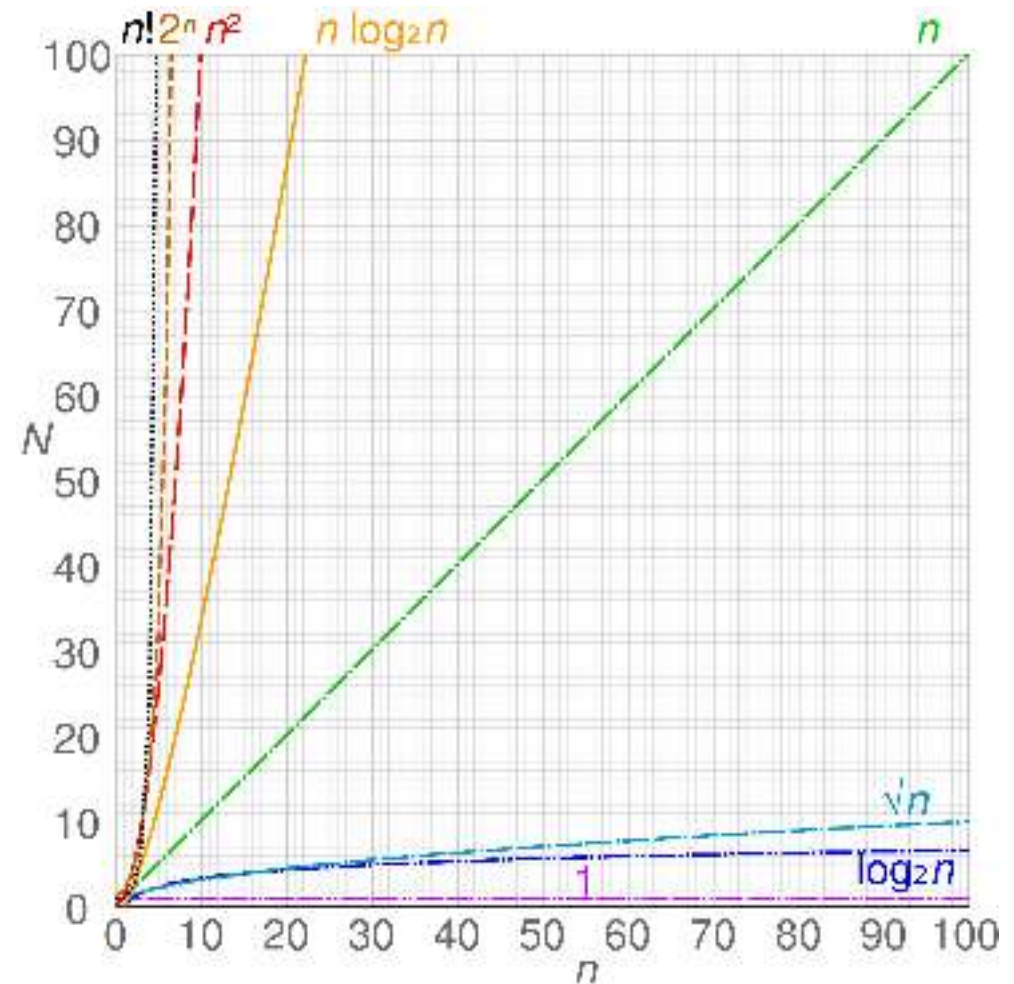
Do we even notice?

Classification of

- Linear search: $O(n)$
- Binary search: $O(\log n)$

Big-O-Notation

- Classifies algorithms according to how runtime grows with growing input size
- Usually provides an upper bound on the growth rate
- Picture shows some important classes
- $O(1)$ means “in constant time”, i.e. not depending on input size at all



Do we even notice?

Let's assume $n = 100$ elements

- $O(n)$: 100 operations required to find an element
- $O(\log n)$: 10 operations required

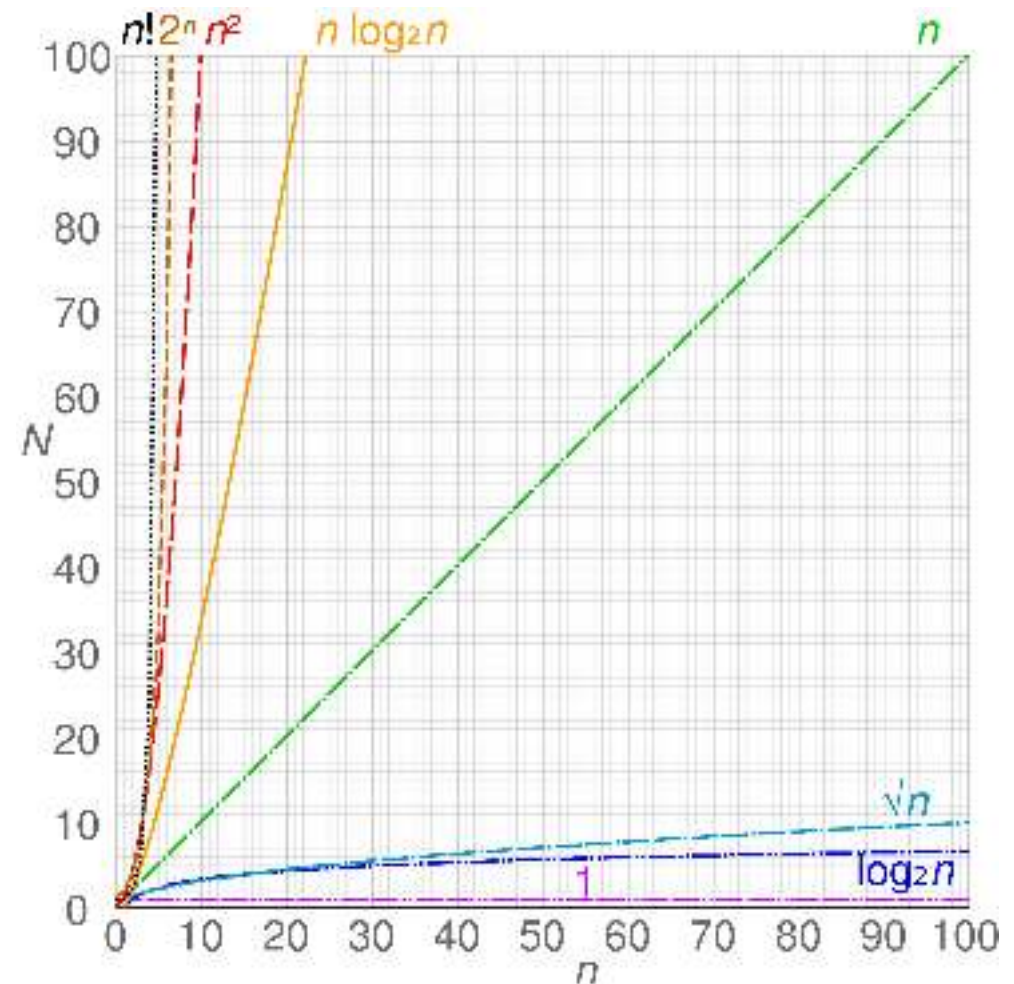
And for $n = 10.000.000$ (i.e. a factor of 100.000)?

- $O(n)$: 10.000.000 operations required
- $O(\log n)$: ~24 operations required.
That's a factor of ~2,4!

And now imagine each operation takes one second to finish...

10.000.000 sec ~ 2777,78 hours ~ 115,74 days

Google wouldn't be your friend anymore...

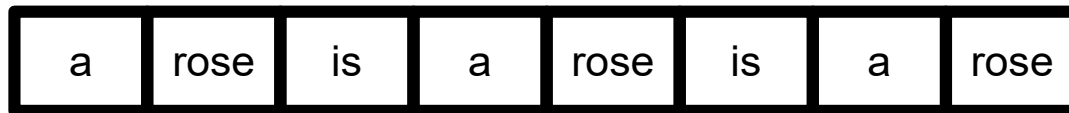


More on data structures

Sets

- Arrays maintain the order among elements but elements are not necessarily unique
- Sets store unique values without particular order
- Usually used to test a value for membership in a set

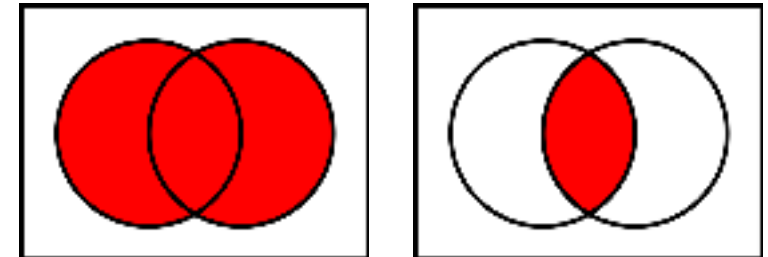
A poem as an array:



and as a set:

{ a, rose, is }

Sets allow fundamental operations to create new sets from given sets, e.g. the union of two sets and their intersection

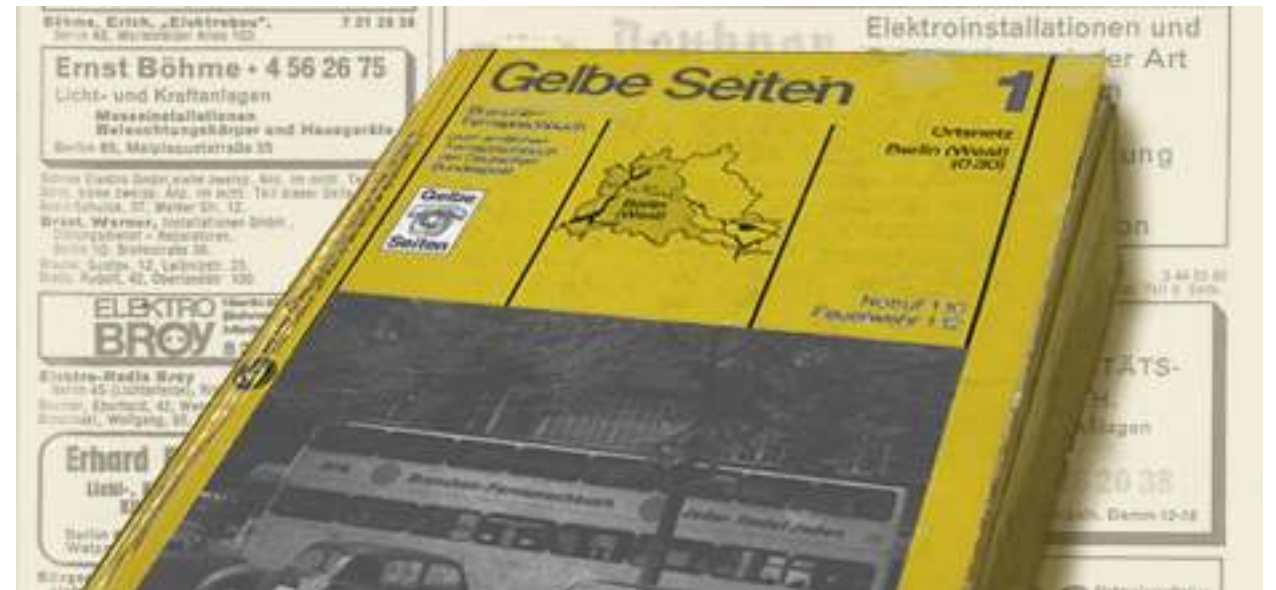


and more complex ones...



Keys and values

- Arrays allow accessing elements by index, i.e. their position in the list.
- Accessing elements by a semantically meaningful key requires dictionaries a.k.a. associative arrays
- Keys must be unique
- Can be implemented such that finding elements is possible in $O(1)$



Ancient key-value-store in paperback. ~1900

Keys and values

Most high-level programming languages offer dictionaries as a primitive, built-in data structure:

- dict (Python)
- dataframe (R)
- Map (Java)
- `std::map` (C++)

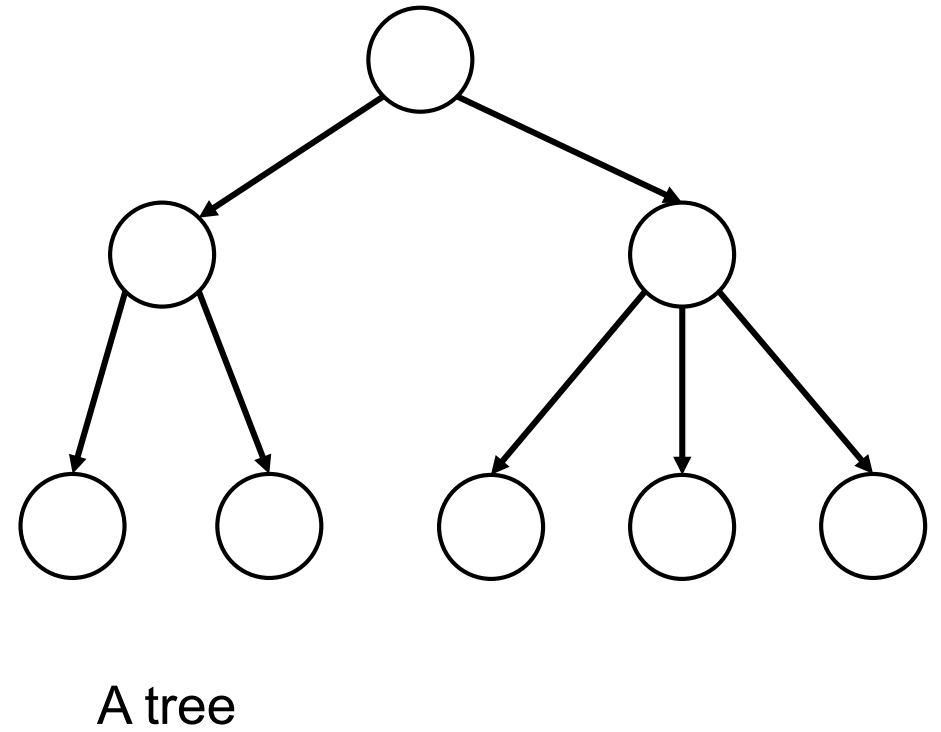
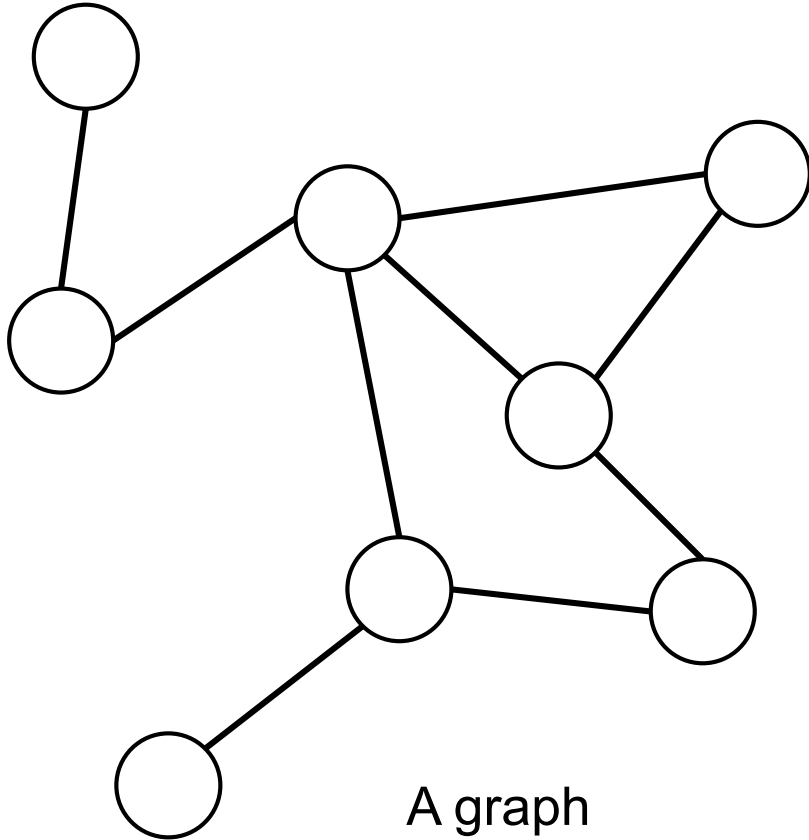
Differences in syntax, but functionality is basically the same

```
capitals = {  
    "France" : "Paris",  
    "Germany" : "Berlin",  
    "Brazil" : "Brasilia"  
}
```

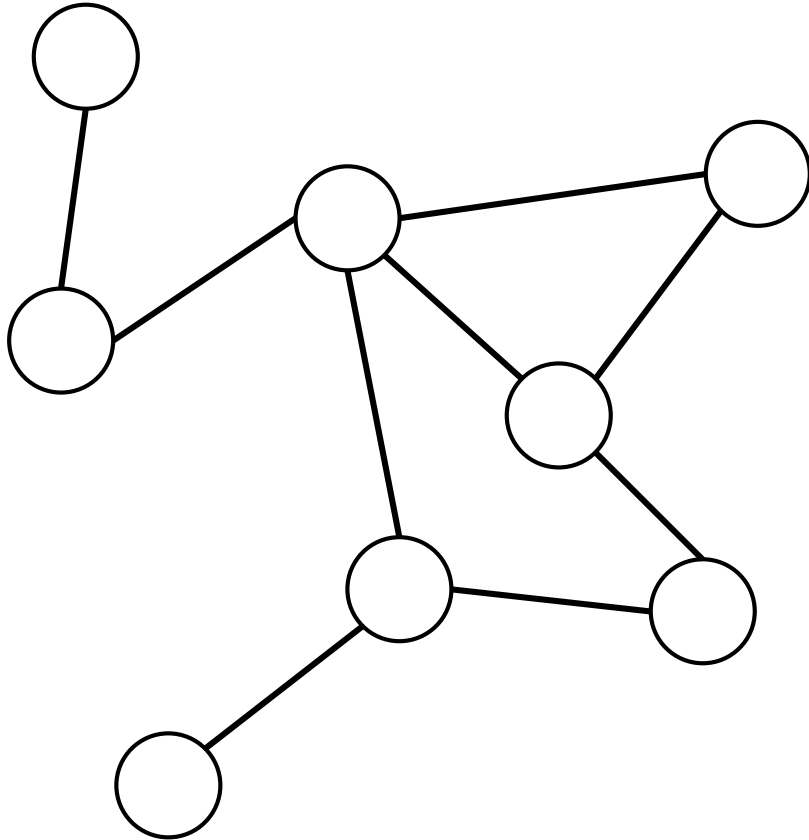
```
capitals["France"]  
> "Paris"
```

```
capitals["Mexico"] = "Mexico City"
```

Graphs & Trees



Graphs



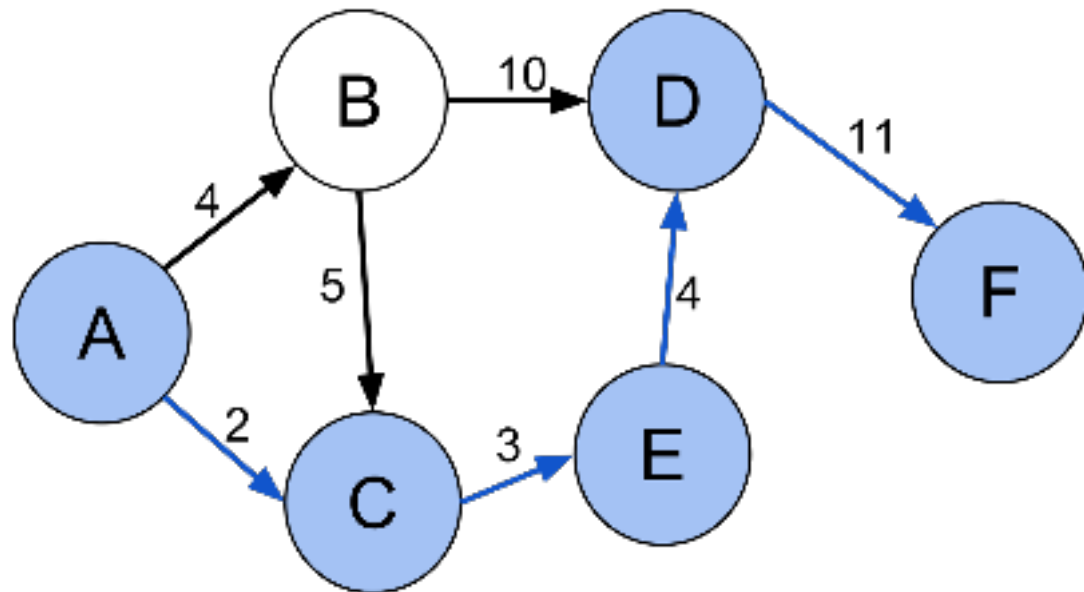
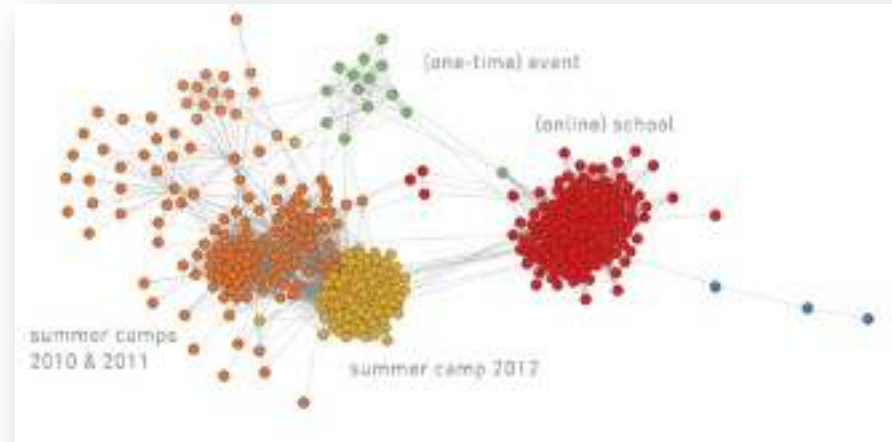
A graph in graph theory is a mathematical structure used to model pairwise relations between objects.

A graph consists of *vertices* (or *nodes*) and connecting *edges* (or *links*)

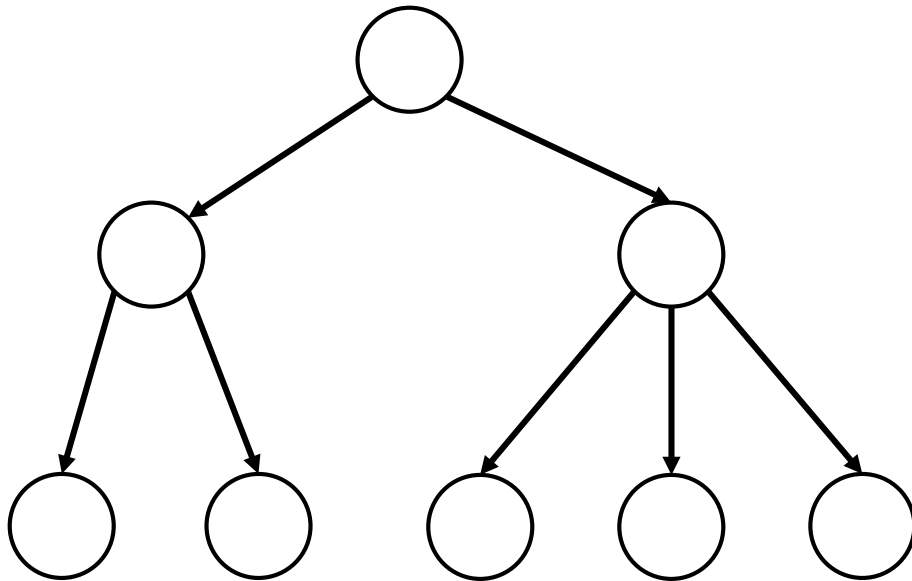
There is a large variety of types of graphs

- Directed / Undirected / Mixed
- Connected / Disconnected
- Bipartite
- Weighted
- Attributed
- Tree

Graphs



Trees



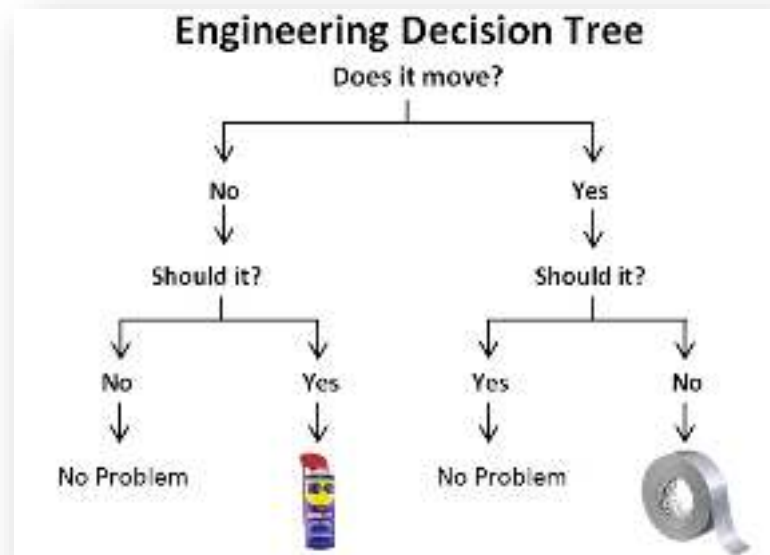
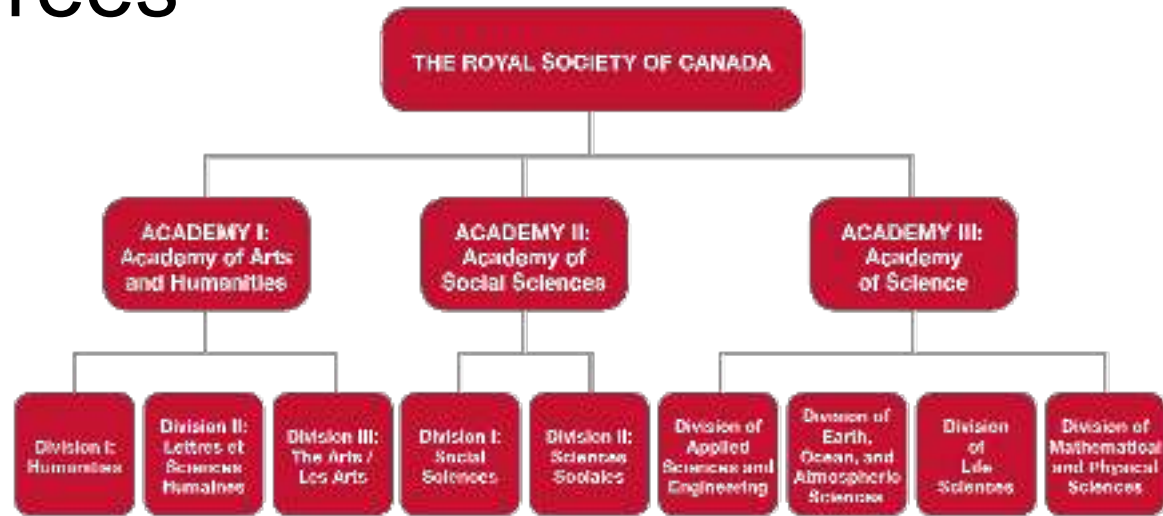
A tree in graph theory is an undirected graph in which any two vertices are connected by exactly one path, or equivalently a connected acyclic undirected graph.

A tree as an abstract data type simulates a hierarchical tree structure, with a root value and subtrees of children with a parent node.

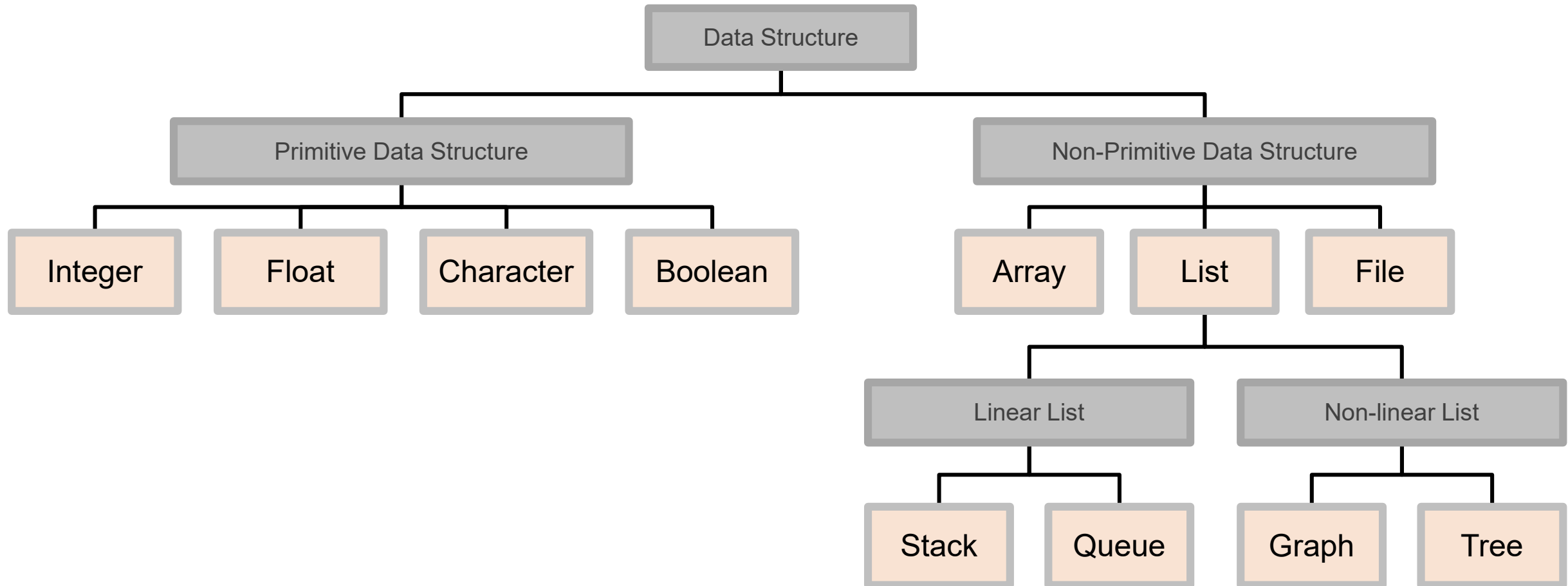
Trees as abstract data types are used to

- Represent hierarchical data (e.g. file systems)
- Represent evolutionary data (e.g. in biology, technology, ...)
- Represent Document Object Models of XML
- Implement efficient search

Trees



Data Structures: An Overview



Wait... files are data structures?



Important file types

Popular file types for exchanging data

- csv / tsv
- JSON
- XML
- GraphML

All of the above choices are essentially text files

Proprietary binary formats are bad choices for exchanging data

Why text files?

- Human-readable
- Require no proprietary software to view / edit
- Work well with version control systems
- Can easily be fed into algorithms
- Command-line tools are great in processing text files
 - Tools are available by default on Unix/Linux platforms
 - Individual tools are simple but powerful in combination
 - Allow to examine and explore data without any code

Examples:

- `wc` count word & line numbers
- `grep` filter & search files based on their content
- `sort` sort files
- `shuf` pick random lines out of files
- `uniq` handling duplicates
- `cut` access columns in tables
- `join` combine files
- `sed` modify files
- `awk` execute programs on every line of input



csv/tsv: Comma Separated Values

```
Sherlock Holmes,221B Baker Street,Detective  
James Moriarty,Reichenbach Falls,Villain
```

- Simple format
- Human-readable
- Widely used
- Problems if separator is part of a data field
- Separator customizable. **Strong recommendation:** use tsv (tab separated values)!

```
Sherlock Holmes      221B Baker Street  Detective  
James Moriarty      Reichenbach Falls  Villain
```



json: JavaScript Object Notation

```
[{"name":"Sherlock Holmes", "address":"221B Baker Street", "job":"Detective"},  
{"name":"James Moriarty", "address":"Reichenbach Falls", "job":"Villain"}]
```

- Format more standardized than csv/tsv
- Suitable for nested data and composited data
- Fairly easy to write
- Quite popular

xml: eXtensible Markup Language

```
<characters>
  <person name="Sherlock Holmes" address="221B Baker Street" job="Detective"/>
  <person name="James Moriarty" address="Reichenbach Falls" job="Villain"/>
</characters>
```

- Format standardized
- Not too friendly to write
- Very verbose, can be very complex
- Advanced features like DTD, XML Schema, XSL, ... (rarely used)

GraphML

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="undirected">
    <node id="n0"/>
    <node id="n1"/>
    <edge id="e1" source="n0" target="n1"/>
  </graph>
</graphml>
```

- XML-based definition of graphs
- Supports the entire range of possible graph structure constellations
- Supports attributes on graphs, nodes, and edges

So... What is a data structure?

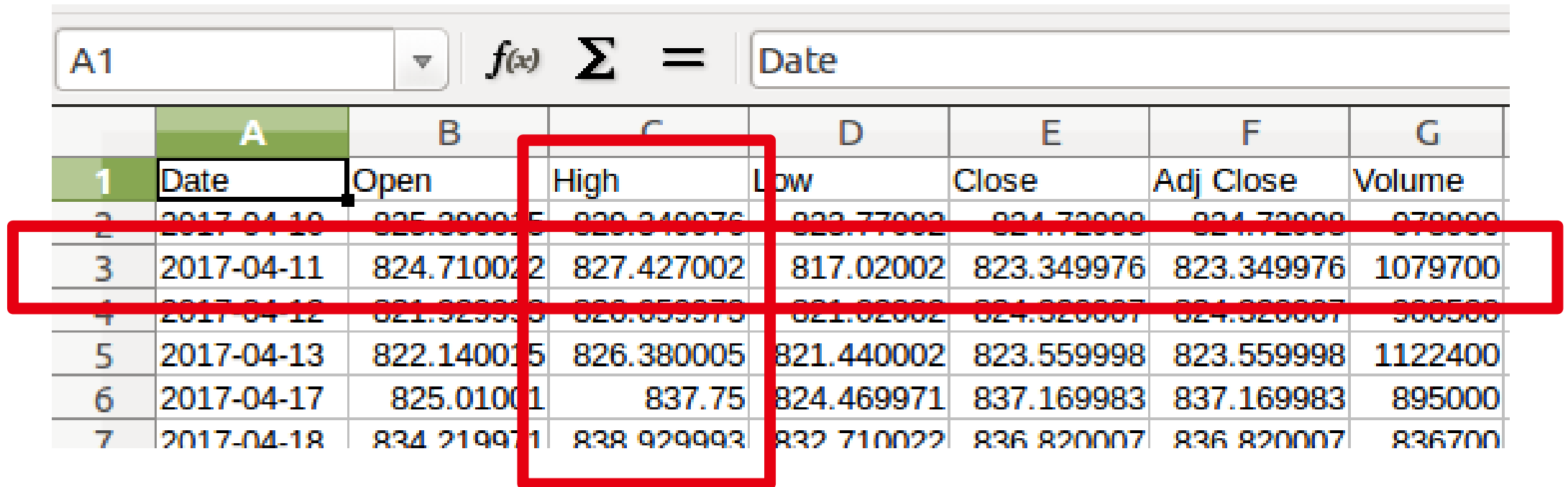


What is a Data Structure?

- Specialized format for organizing, processing, retrieving, and storing data.
- Designed to arrange data to suit a specific purpose
e.g. working with the data using an algorithm
- Contains information about data values, relationships between the data, and functions that can be applied to the data
- Maintains logical relationships between individual data elements
- Different operations on data structures:
 - Insertions
 - Deletions
 - Searching
 - Traversing

Tables

What are tables?



A screenshot of an Excel spreadsheet showing a table of stock data. The table has columns for Date, Open, High, Low, Close, Adj Close, and Volume. A red box highlights the data rows from 2 to 7, and another red box highlights the 'High' column.

| | A | B | C | D | E | F | G |
|---|------------|------------|------------|------------|------------|------------|---------|
| 1 | Date | Open | High | Low | Close | Adj Close | Volume |
| 2 | 2017-04-10 | 825.299995 | 829.349976 | 823.77002 | 824.72002 | 824.72002 | 979000 |
| 3 | 2017-04-11 | 824.710022 | 827.427002 | 817.02002 | 823.349976 | 823.349976 | 1079700 |
| 4 | 2017-04-12 | 821.929995 | 828.659976 | 821.82002 | 824.920007 | 824.920007 | 900500 |
| 5 | 2017-04-13 | 822.140015 | 826.380005 | 821.440002 | 823.559998 | 823.559998 | 1122400 |
| 6 | 2017-04-17 | 825.01001 | 837.75 | 824.469971 | 837.169983 | 837.169983 | 895000 |
| 7 | 2017-04-18 | 834.219971 | 838.929993 | 832.710022 | 836.820007 | 836.820007 | 836700 |

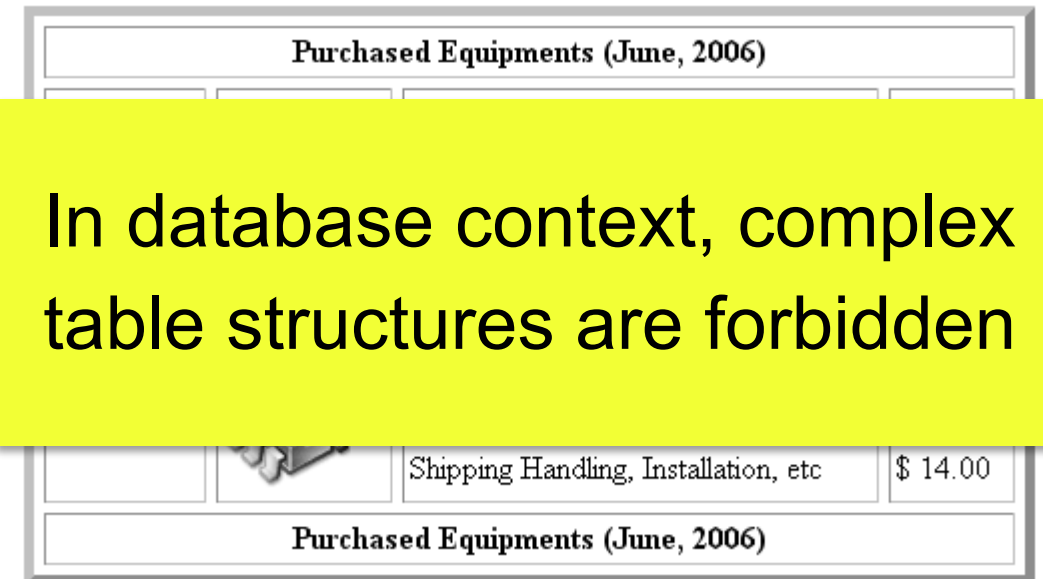
Important aspects of tables

Tables hold *related* data

Consist of rows & columns; their intersections is called cell

Tables have a specified number of columns and any number of rows

| company | division | sector | tryint |
|------------------------|-------------------------|-----------------------|--------|
| 00nil_Combined_Company | 00nil_Combined_Division | 00nil_Combined_Sector | 14625 |
| apple | 00nil_Combined_Division | 00nil_Combined_Sector | 10125 |
| apple | hardware | 00nil_Combined_Sector | 4500 |
| apple | hardware | business | 1350 |
| apple | hardware | consumer | 3150 |
| apple | software | 00nil_Combined_Sector | 5625 |
| apple | software | business | 4950 |
| apple | software | consumer | 675 |
| microsoft | 00nil_Combined_Division | 00nil_Combined_Sector | 4500 |
| microsoft | hardware | 00nil_Combined_Sector | 1890 |
| microsoft | hardware | business | 855 |
| microsoft | hardware | consumer | 1035 |
| microsoft | software | 00nil_Combined_Sector | 2610 |
| microsoft | software | business | 1215 |
| microsoft | software | consumer | 1395 |



Purchased Equipments (June, 2006)

In database context, complex table structures are forbidden

Shipping Handling, Installation, etc \$ 14.00

Purchased Equipments (June, 2006)



Tables as abstract data structures

Tables are usually multisets

i.e. duplicate rows are allowed

How could a table be implemented?

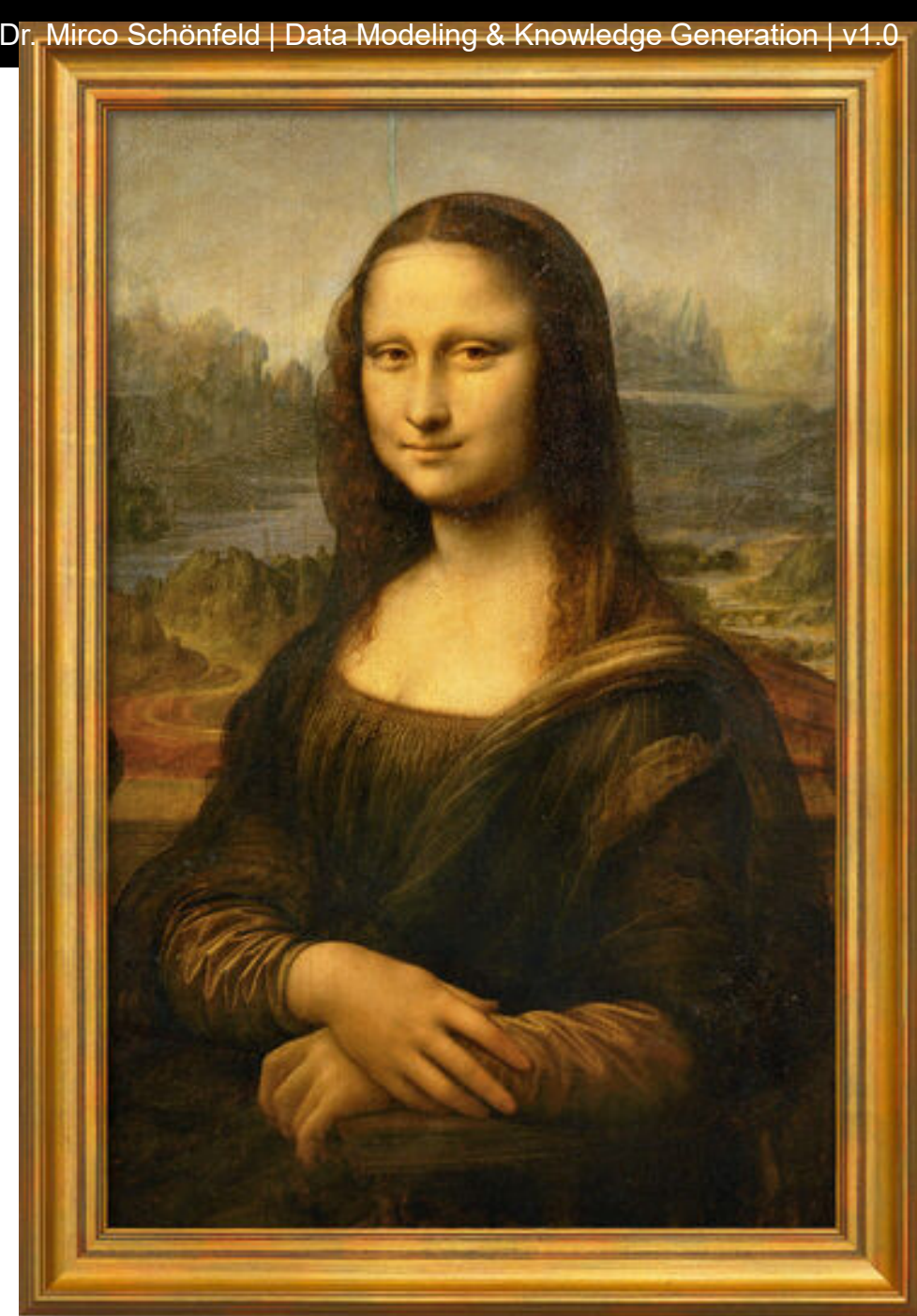
Difference to a spreadsheet: tables have a schema assigned

Schemes as logical data models specify which data types are allowed in which columns, for example

Type safety

Often, for specific data elements,
only certain types are allowed!

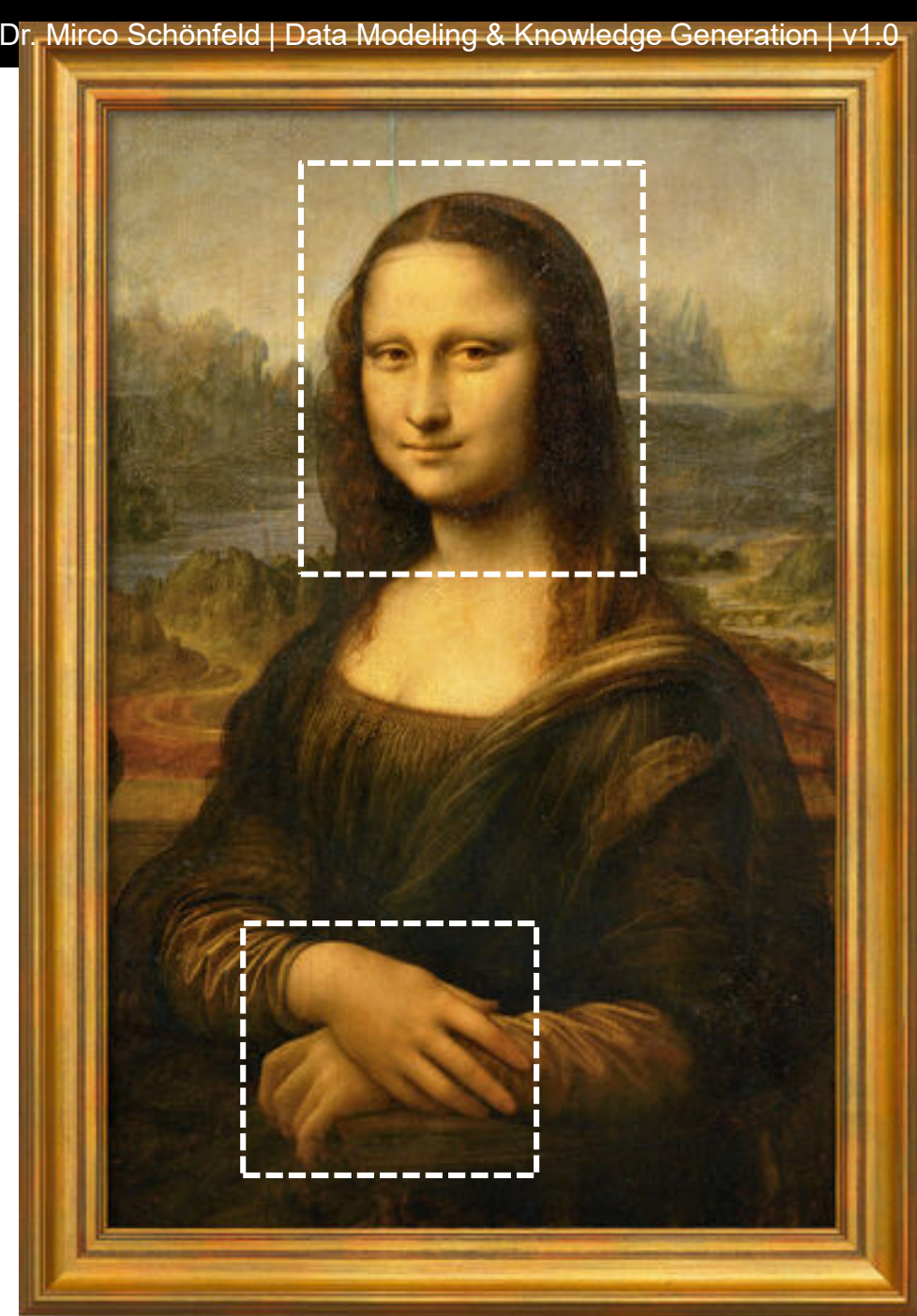
| ID | Painter | Year | Title |
|--------|-------------------|------|-----------------------|
| 124123 | Leonardo da Vinci | 1506 | Mona Lisa |
| 453723 | Peter Paul Rubens | 1606 | The Judgment of Paris |



Type safety

Often, for specific data elements,
only certain types are allowed!

| ID | Upper-left-x | Upper-left-y | Lower-right-x | Lower-right-y | Annotation |
|--------|--------------|--------------|---------------|---------------|------------|
| 124123 | 24 | 5 | 57 | 40 | Face |
| 124123 | 17 | 76 | 43 | 87 | Hands |



Thanks.

mirco.schoenfeld@uni-bayreuth.de