UCL

CEGE0096: Geospatial Programming

Lecture 6: Geospatial Modelling

November 11, 2019

Aldo Lipani

GEOSPATIAL MODELLING

Definition of GIS

"A Geographic Information System is a computer-based information system that enables capture, modelling, storage, retrieval, sharing, manipulation, analysis and presentation of geographically referenced data."

Worboys and Duckham, 2004

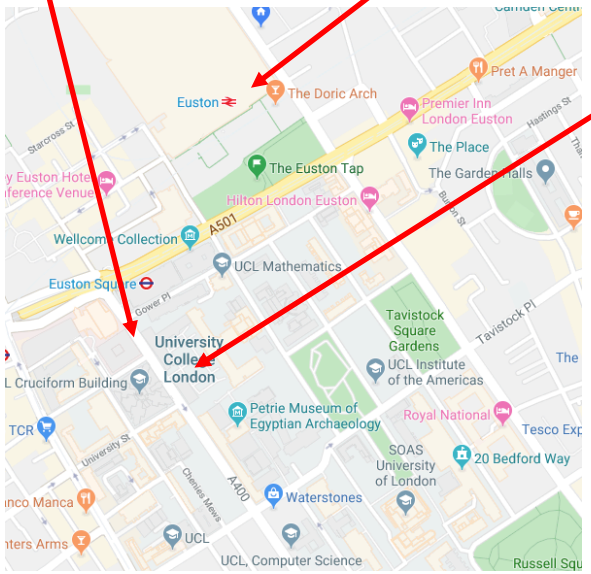
In order to store, share, manipulate, analyze and present we first need to capture and model.

Capturing and modelling is dependent on our view of reality.

How do we Represent Reality?

On a piece of paper, draw three separate maps of:

- **Gower Street** and its potholes;
- The path to get from **Euston Station** to **Chadwick Building**;
- **Gower Street** in a map of the **UK**.



How do we Represent Reality?

Each map demands a different way of representing Gower Street.

Each representation has advantages and disadvantages.

The representation is a simplification of the real world feature.

Choosing the correct representation is an important decision.

Real vs. Digitally Represented Worlds

The real world:

- No straight lines
- No hard edges
- 3D + Time
- Ambiguous
- Missing information
- Unusual unexpected features
- Locations can be a function of probability

Digitally represented world:

- Simplified geometry
- 2D (paper)
- 2D + Time (screen)
- 3D + Time (VR)
- Specified time
- Locations may not be precise
- Locations may not be accurate

Accuracy vs. Precision

Accuracy is the closeness of **a measured value** to a standard or known value.

E.g. if you measure a distance of 3.2 meters but the actual or known distance is 10 meters:

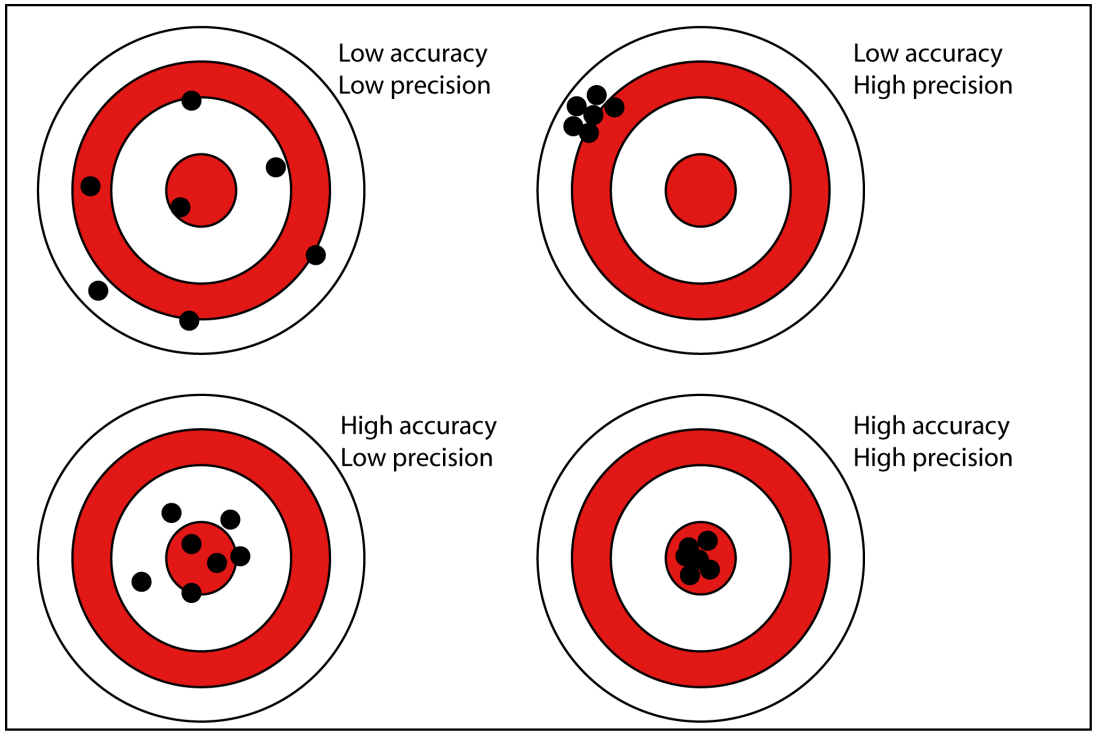
- Your measurement is not accurate.
- It is not close to the known value.

Precision is the closeness of **two or more** measurements to each other.

Precision is independent of accuracy.

E.g. if you measure a distance five times, and each time you measure 3.2 meters, then your measurement is precise. Even if the distance is actually 10 meters.

Accuracy vs. Precision



Modelling

"A model is an artificial construction in which **parts of one domain**, termed the source domain, are **represented in another domain**, termed the target domain."

Worboys and Duckham, 2004

A model is a simplification and an abstraction away from the source domain.

"All models are wrong, but some are useful."

George E. P. Box

Examples include: paper and online maps, flight simulators, health models like hospital planning, meteorological models like weather forecasting, and geological models.

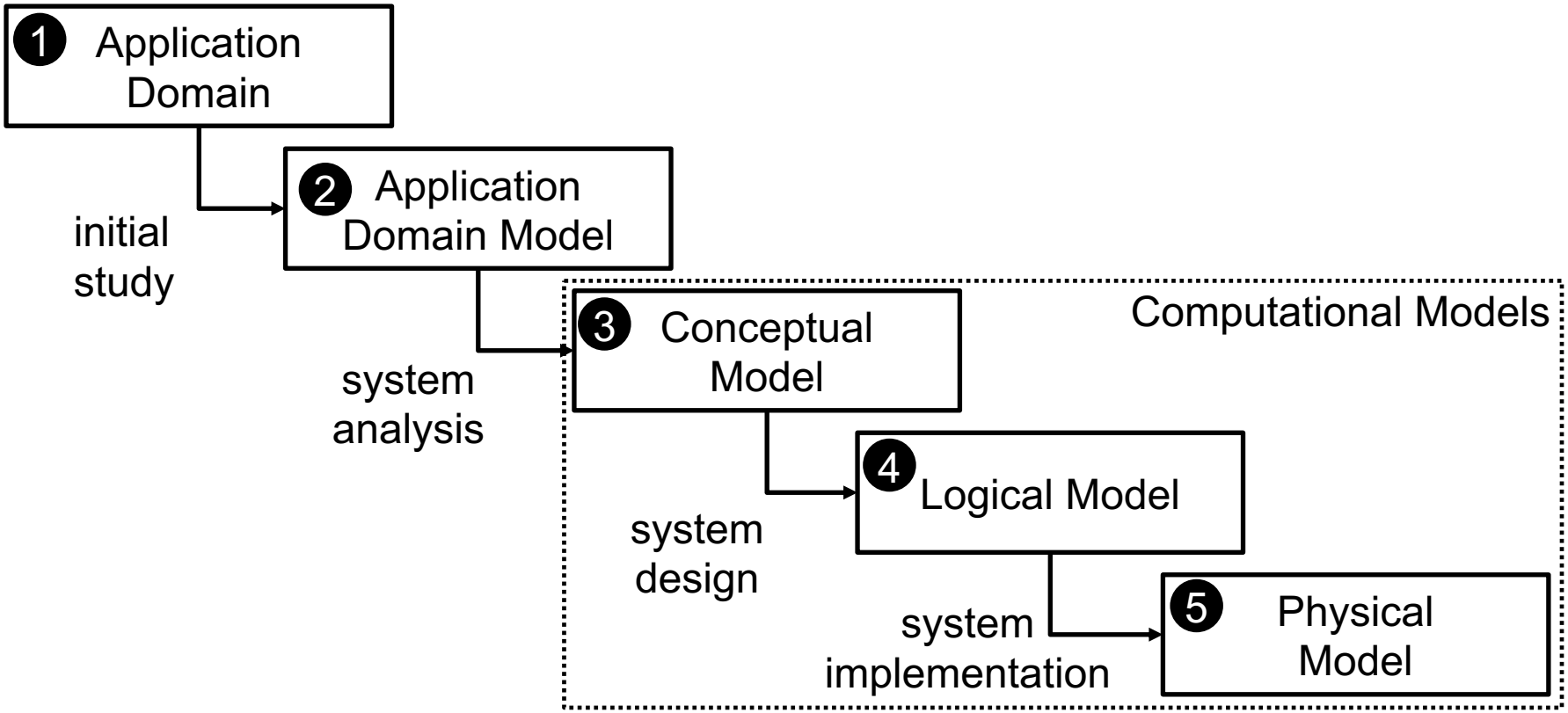
Fidelity vs Implementability

The usefulness of a model is trade-off between **fidelity** and **implementability**.

Fidelity is the degree to which the model represents the world;

Implementability is the degree to which the model can be used for storage, analysis, etc.

The Modelling Process – Waterfall Model



Application Domain 1

The real world is referred to as the Application domain.

Observe the real world.

Everything happens in the context of the real world.

Application Domain Model ❷

Focus only on the parts of the real world that are import to your world.

It is important to identify what is important and what is not.

At this stage you would produce a document with descriptive notes, diagrams, tables, etc.

Conceptual Model 3

What will the system do?

Identify phenomena, their attributes and their relationships to the world.

The real world phenomena can be conceptualized as either:

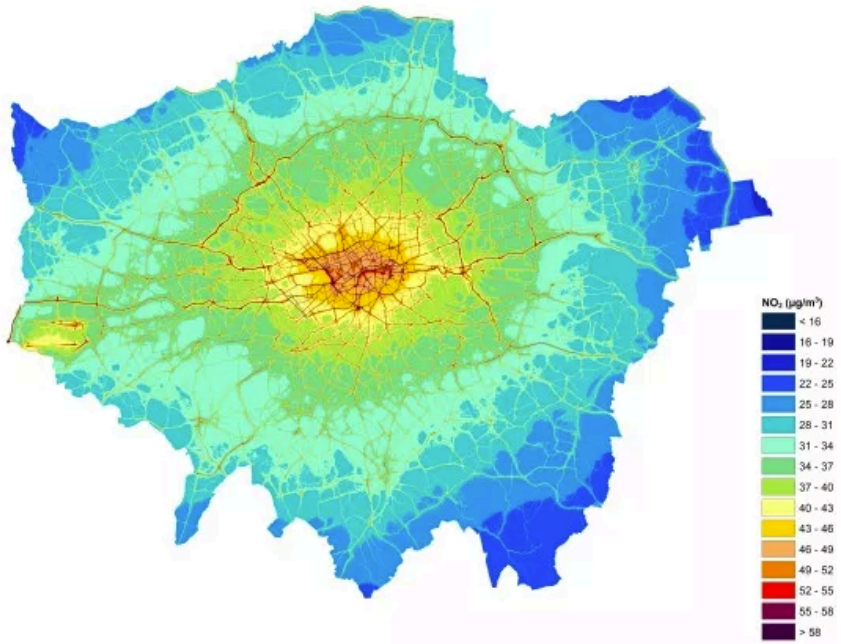
- Continuous fields
- Discrete features

Continuous Fields

A continuous field has a characteristic that is spatially continuous.

Values do not suddenly change at boundaries:

- Temperature
- NO_2 concentration



Discrete Features

A discrete feature (aka entity or object) is distinct from its environment and its neighbors due to an identifiable characteristic:

- Building
- Tree
- Country
- A post code

Discrete features can be **classified**.

Discrete features can be **aggregated**.

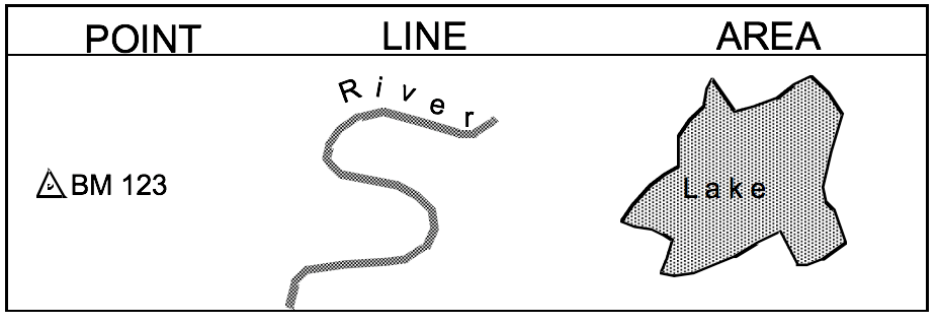
Dimensionality

Features can be conceptually modelled as:

- 0D entities i.e. Points;
- 1D entities i.e. Lines;
- 2D entities i.e. Areas, Surfaces;
- 3D entities i.e. Volumes, Solids.

Modelled either in 2D or 3D, e.g. a point in 3D space or a point on a 2D surface.

Features can be modelled in time.



Logical Model 4

How will the system implement the conceptual computational model?

Identify how phenomena might be represented in an information system.

What datatypes would be best to represent attributes?

E.g. How to represent colors?

- “red” -- string
- integer
- #FF0000 -- hexadecimal integer representing RBP
- (1.0, 0, 0) – floating point RBP tuple

Spatial Representations

Identify how the geometry of features and fields can be represented in the information system:

- Vector representation;
- Raster representation.

Vector Representation

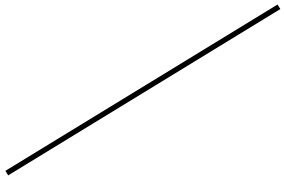
A vector is a line segment (edge, wire) defined by its end points (vertices).

Line segments may be curved.

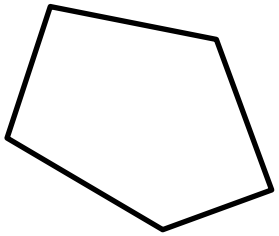
Requires less storage than raster for same fidelity.

A 0D geometric entity

A 1D geometric entity.



A 2D geometric entity.



Raster Representation

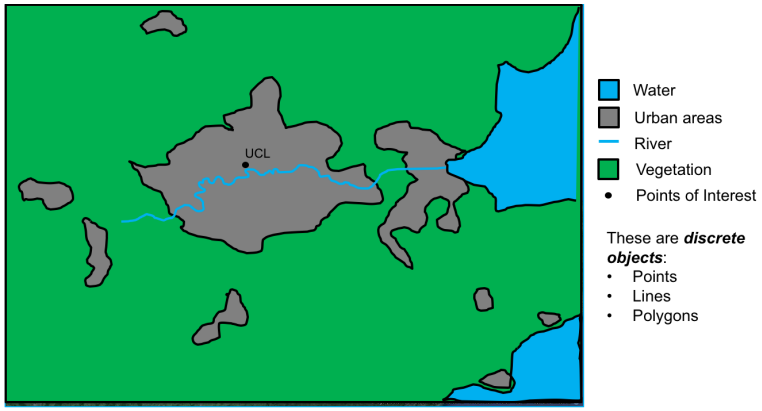
Structured as an array or grid of cells, referred to as pixels (or grid cells).

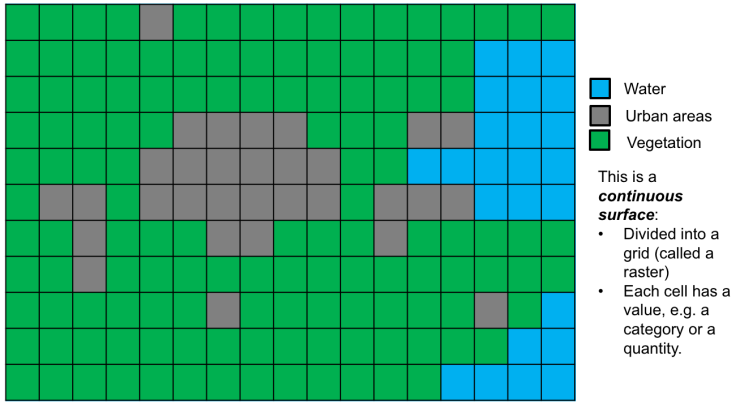
The 3D equivalent is a 3D array of cubic cells, called voxels.

Each raster cell is addressed by its position in the array.

Each cell can store a number as an attribute.

Attributes can be binary, integer, or floating point.





Raster Representation (II)

Each cell in a raster will have a fixed dimension (the **resolution**).

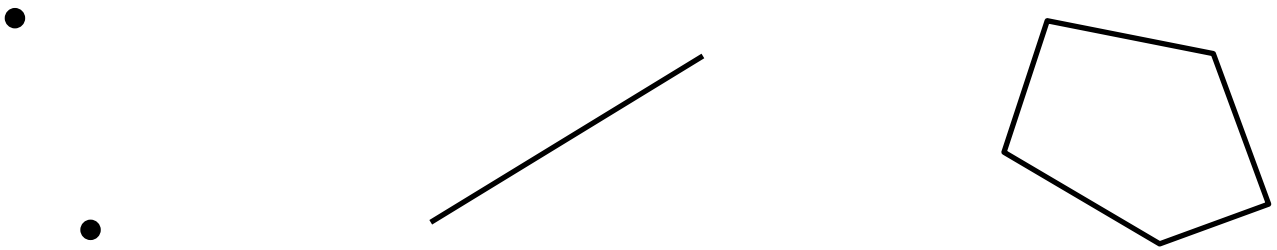
The higher the resolution, the more detail can be represented in the raster at the expense of memory.

Rasters can be made on-the-fly using a mathematical **function**.

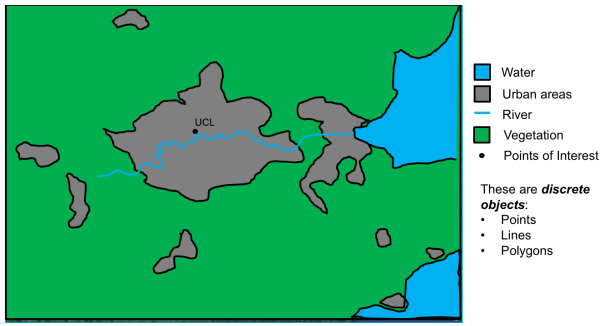
E.g. Inverse distance squared values from a point feature.

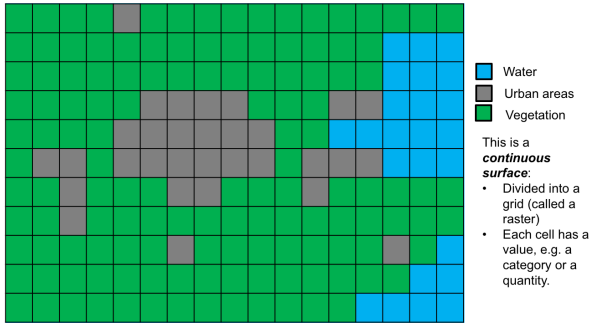
Exercise (in Moodle)

How would you implement the Vector representation in Python?



How would you implement the Raster representation in Python?





Layers (aka Levels)

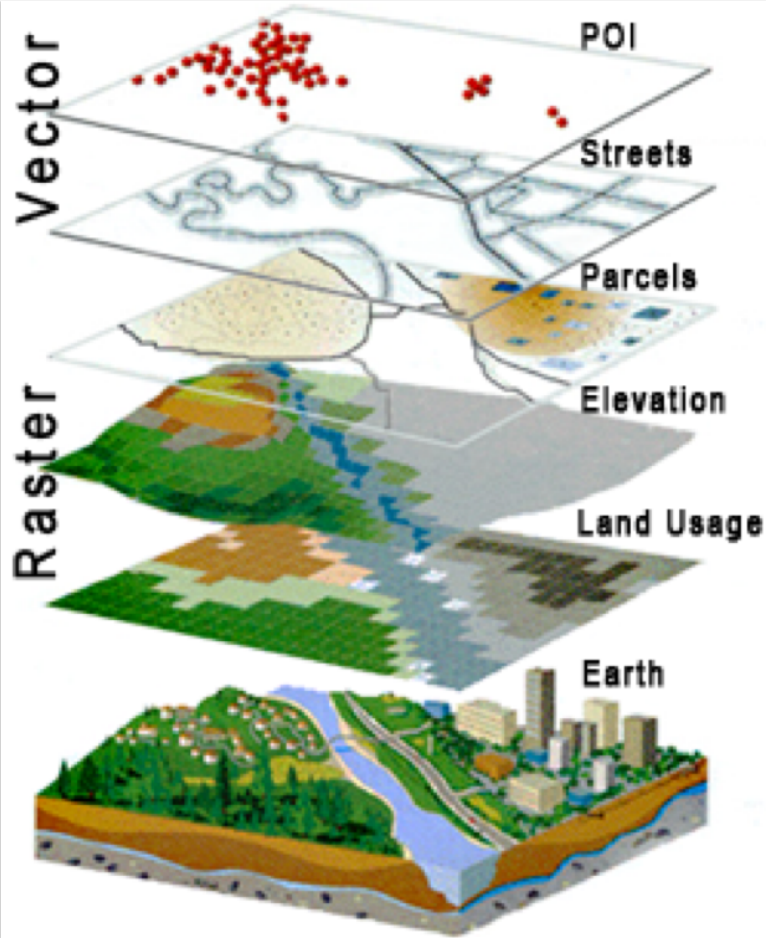
Geospatial fields and features representing identical phenomena are collated together in layers.

In computer aided design, levels are used primarily for visualization.

Levels are less disciplined as different object types may exist together in the same level.

Layers can be combined together with other layers for analysis and visualization.

A fully-capable GIS is able to combine data from different layers.



Physical Model

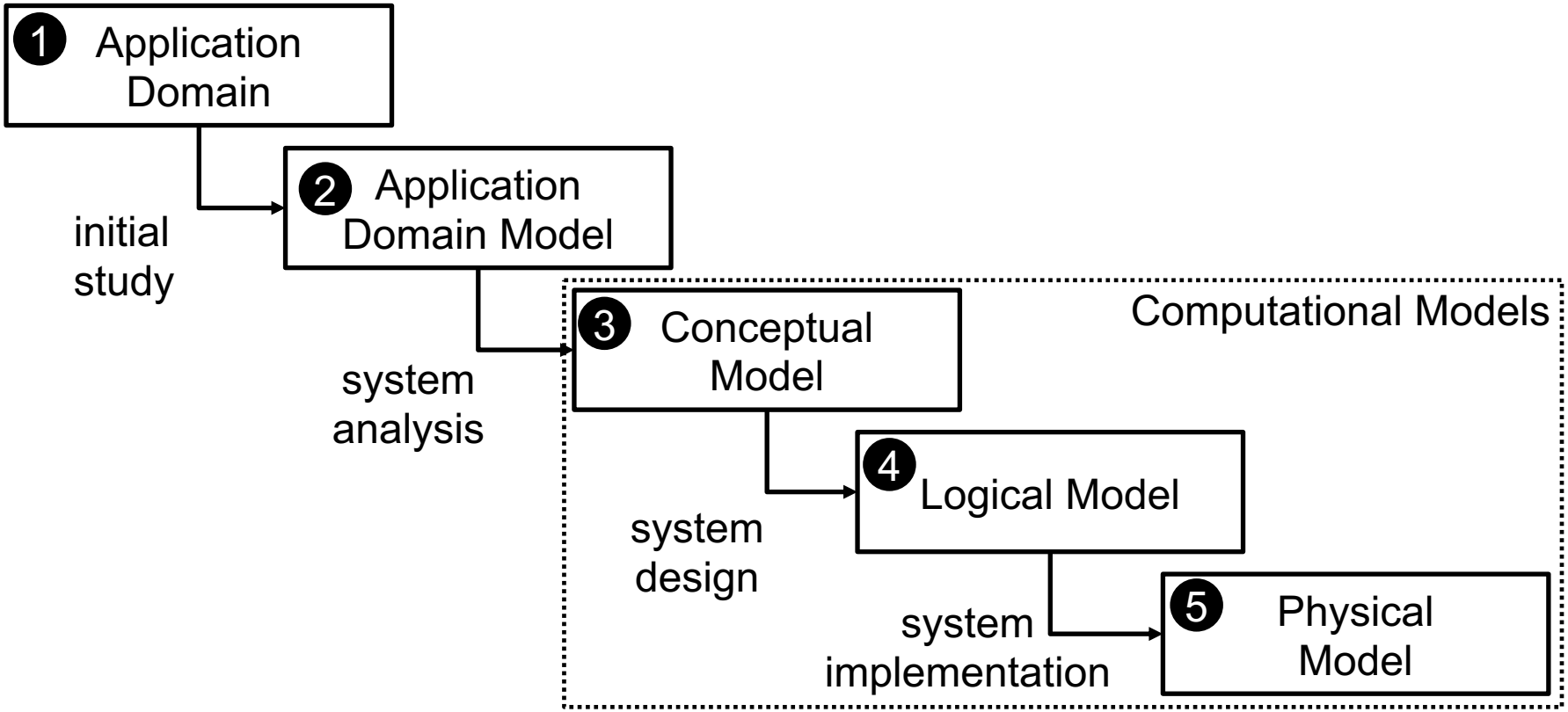
Transcribe the logical model into a data file.

Load the data file into a program.

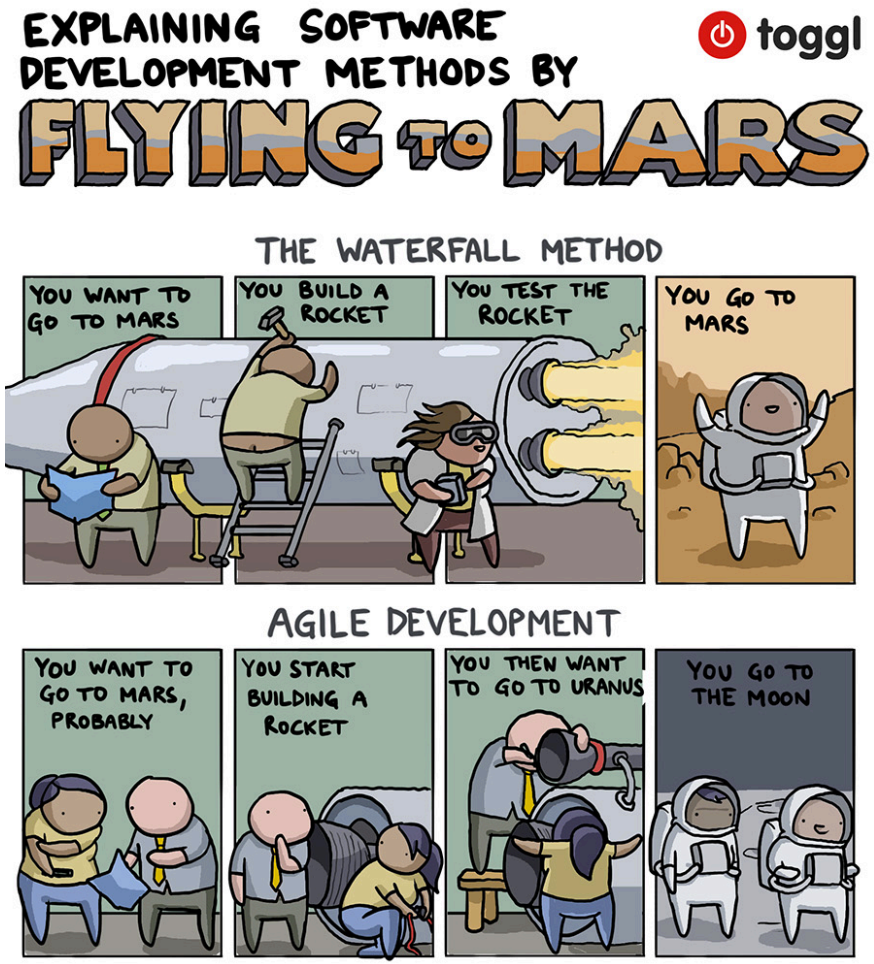
The implemented physical model can be simulated, analyzed and visualized.

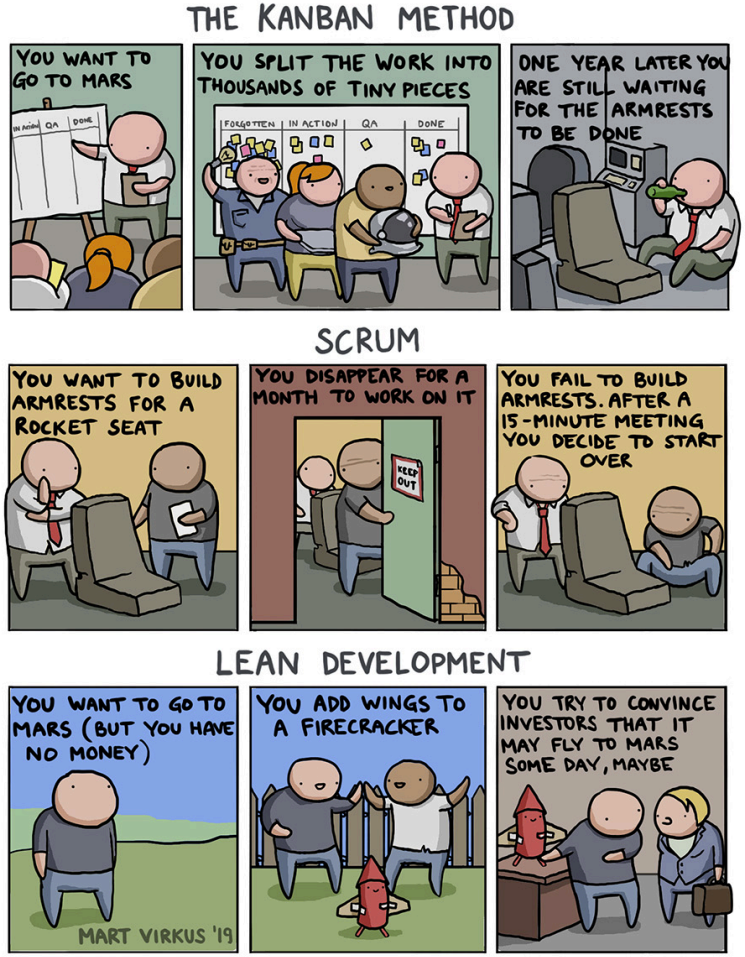
Exercise (Reflection on the Assignment)

Identify the modelling steps we took in the 1st Assignment to get from the Application Domain (the problem) to the Physical Model (the solution).

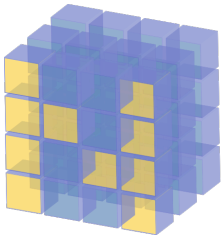


Other Modelling Methods





PYTHON PACKAGES



NumPy

NumPy

NumPy is an independent scientific computing package for Python.

Written in C for speed an efficiency. NumPy arrays (lists) occupy less memory and are faster than Python lists!

```
import numpy as np
```

The fundamental data structure of NumPy is the array:

```
arr = np.array([1, 2, 3, 4])
```

NumPy Arrays

Arrays can be 1D, 2D, 3D, or n-dimensional:

```
arr = np.array([[1, 2, 3, 4], [4, 3, 2, 1]])
```

Arrays can be created from any datatype. This is specified using the dtype keyword.

```
arr = np.array([1, 2, 3, 4], dtype=int)
```

You can also use C types:

```
arr = np.array([1, 2, 3, 4], dtype=np.int32)
```

Addressing Cells

The individual cells of an array can be addressed using [,].

For a 2D array we can address the the first element of the second row as:

```
arr = np.array([[1, 2, 3, 4], [4, 1, 2, 3]])

arr[1,0]
```

Stacking Arrays

Two arrays can be stacked together both vertically and horizontally.

Column-wise:

```
np.hstack([arr1, arr2])
```

Row wise:

```
np.vstack([arr1, arr2])
```

Arrays Operations

NumPy offers a new set of operations you can perform on arrays.

You can sum two arrays (`+`, `-`), compute the power exponentiation (`**`), conditions (`==`, `<`, `>`, `<=`, `=<`), and element-wise multiplication (`*`, `/`).

You can also perform matrix multiplications (`@`):

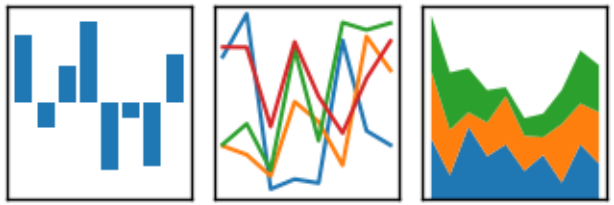
$$\overset{4\times 2 \text{ matrix}}{\begin{bmatrix} a_{11} & a_{12} \\ \cdot & \cdot \\ a_{31} & a_{32} \\ \cdot & \cdot \end{bmatrix}} \overset{2\times 3 \text{ matrix}}{\begin{bmatrix} \cdot & b_{12} & b_{13} \\ \cdot & b_{22} & b_{23} \end{bmatrix}} = \overset{4\times 3 \text{ matrix}}{\begin{bmatrix} \cdot & x_{12} & x_{13} \\ \cdot & \cdot & \cdot \\ \cdot & x_{32} & x_{33} \\ \cdot & \cdot & \cdot \end{bmatrix}}$$

Documentation

http://numpy.org

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Pandas

Pandas is a software library written for the Python for data manipulation and analysis.

```
import pandas as pd
```

It offers data structures and operations for manipulating **numerical tables** and **time series**.

These are called **Series** and **DataFrames**.

A Series is a column, and a DataFrame is a multi-dimensional table made up of a collection of Series.

Creating a DataFrame from Scratch

A way to create a DataFrame is from a python dictionary.

```
data = {
    'x': [0, 1, 2, 3],
    'y': [3, 2, 1, 0]
}
```

Then, pass it to the pandas DataFrame constructor:

```
df = pd.DataFrame(data)
```

Creating a DataFrame from CSVs

With CSV files you can use read_csv:

```
df = pd.read_csv('points.csv')
```

To save a data frame as a CSV file:

```
df.write_csv('points.csv')
```

Check a DataFrame

To see the first n rows of a DataFrame:

```
df.head()
```

To see the last n rows of a DataFrame:

```
df.tail()
```

To get info about your DataFrame:

```
df.info()
```

To get stats:

```
df.describe()
```

To get the dimensions of your DataFrame:

```
df.shape
```

Accessing Columns

To see the first n rows of a DataFrame:

```
df.columns
```

To access a column:

```
df['x']
```

To access a column and return a DataFrame:

```
df[['x']]
```

Accessing Rows

To see the first n rows of a DataFrame:

```
df.iloc[0]
```

To access a value:

```
df.iloc[0]['x']
```

Documentation

https://pandas.pydata.org/

See you on Thursday for the Practical

Christopher Ingold Building G20