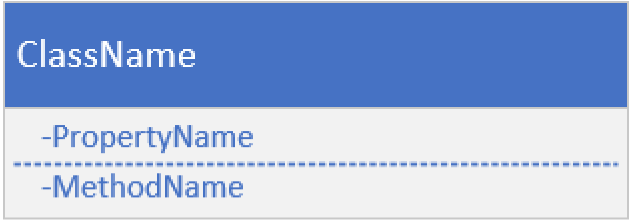
Lecture 8: Matlab APP

Class definition in Matlab

```
classdef (ClassAttributes) ClassName < SuperClass1 & SuperClass2
    properties (PropertyAttributes)
        ...
    end
    methods (MethodAttributes)
        ...
    end
    events (EventAttributes)
        EventName
    end
end
```



Class Attributes

- Abstract
  - If specified as true, this class is an abstract class (cannot be instantiated).
  - classdef (Abstract = true) ClassName
- Sealed
  - If true, this class cannot be subclassed.

Value Class vs. Handle Class

- Value Class
- Each assignment creates a new copy of the object

```
classdef NumValue
  properties
    Number = 1
  end
end
```

- a = NumValue;
- b=a;
- a.Number = 7;
- b.Number
  - ans=1

- Handle Class
- Upon construction a reference to the object is created

```
classdef NumHandle < handle
  properties
    Number = 1
  end
end
```

- a = NumHandle;
- b=a;
- a.Number = 7;
- b.Number
  - ans=7

Value Class vs. Handle Class (cont.)

- When object passed into a function
  - Value object: a new copy of the object is created inside function workspace
  - Handle object: a copy of the handle (reference) is created instead of the object

- Deleting a handle object
  - Delete(NumHandle)

Object equality

Value object

- Can only evaluate whether value of the objects are the same

- a = NumValue;
- b = NumValue;
- isequal(a,b)
- ans=1

Handle object

- Can check whether they are the same object as well as their value equality

- a = NumHandle;
- b = a;
- a == b (same object?)
  - ans=1;
- isequal(a,b) (same value?)
  - ans=1;

a = NumHandle;

b = NumHandle;

a == b

ans=0;

isequal(a,b)

ans=1;

Class Members Access

- public — Unrestricted access
- protected — Access from methods in class or subclasses
- private — Access by class methods only (not from subclasses)
- List classes (and their subclasses) have access to this member
  - (Access = {?ClassName1,?ClassName2,...})

Property Attributes

- Read and write access
  - GetAccess
  - SetAccess
    - properties(GetAccess = 'public', SetAccess = 'private')
    - % public read access, but private write access.
    - end
    - SetAccess = immutable: set during construction, cannot be changed afterwards
- Constant

```
properties(Constant = true)
    DAYS_PER_YEAR =  365;
end
```

- Dependent
  - depend on other values
  - calculated only when needed.
  - i.e. area of a square depends on the width property

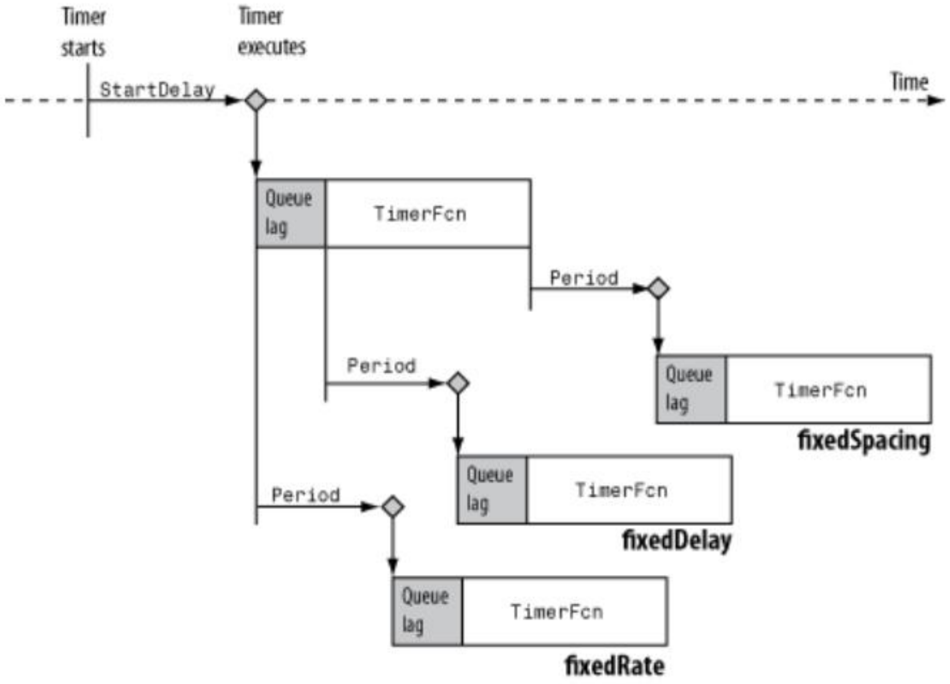
Class Constructor Method

- There is a default class constructor without input arguments
- We can define class constructor that overrides the default one
- Method with the same name as the class name

```
classdef ConstructorDesign < BaseClass1
methods
    function obj = ConstructorDesign(a,b,c)
    end
end
```

Timer Class

- t = timer;
- Properties
  - ‘ExecutionMode’
  - ‘Period’: Time between timer functions
  - ‘TimerFcn’: Function handle
- t.TimerFcn=@callback;
- Function callback(hObj,src,event)



Demo: Traffic Light

Example: Information system for restaurants

- The owner of restaurant A would like to improve service efficiency

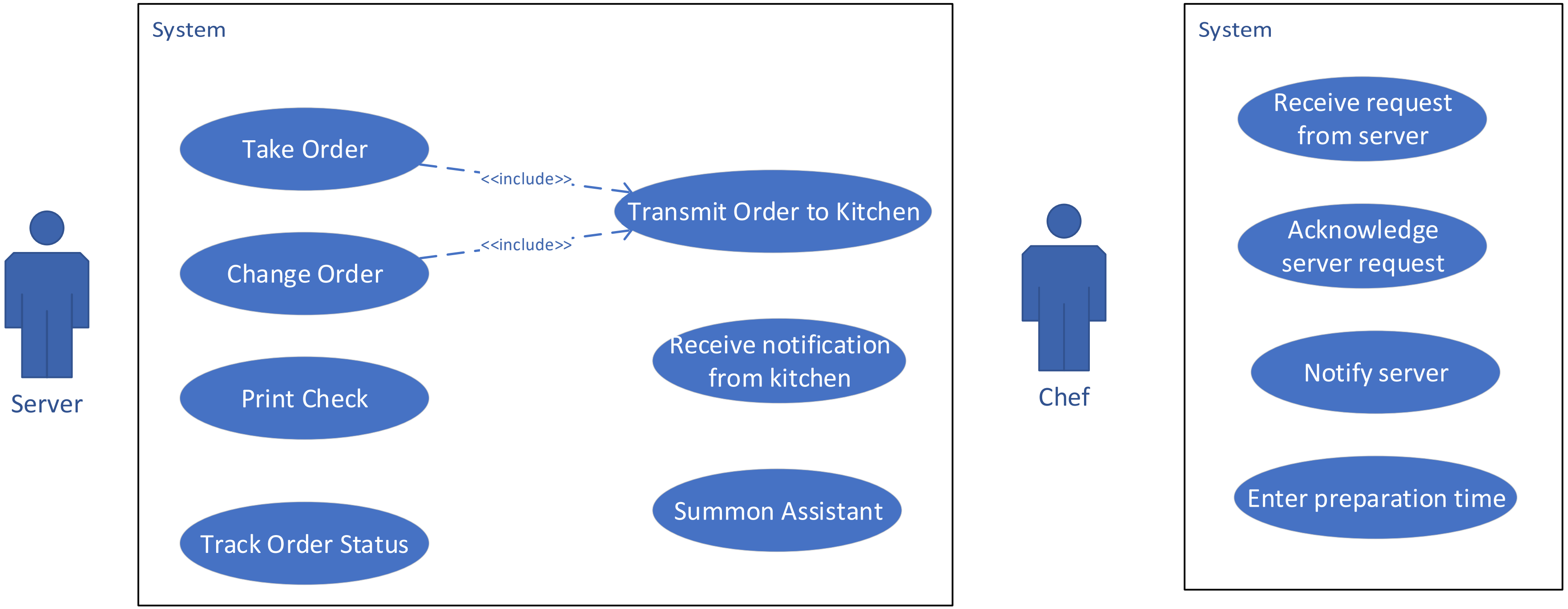

Customer


Server


Chef

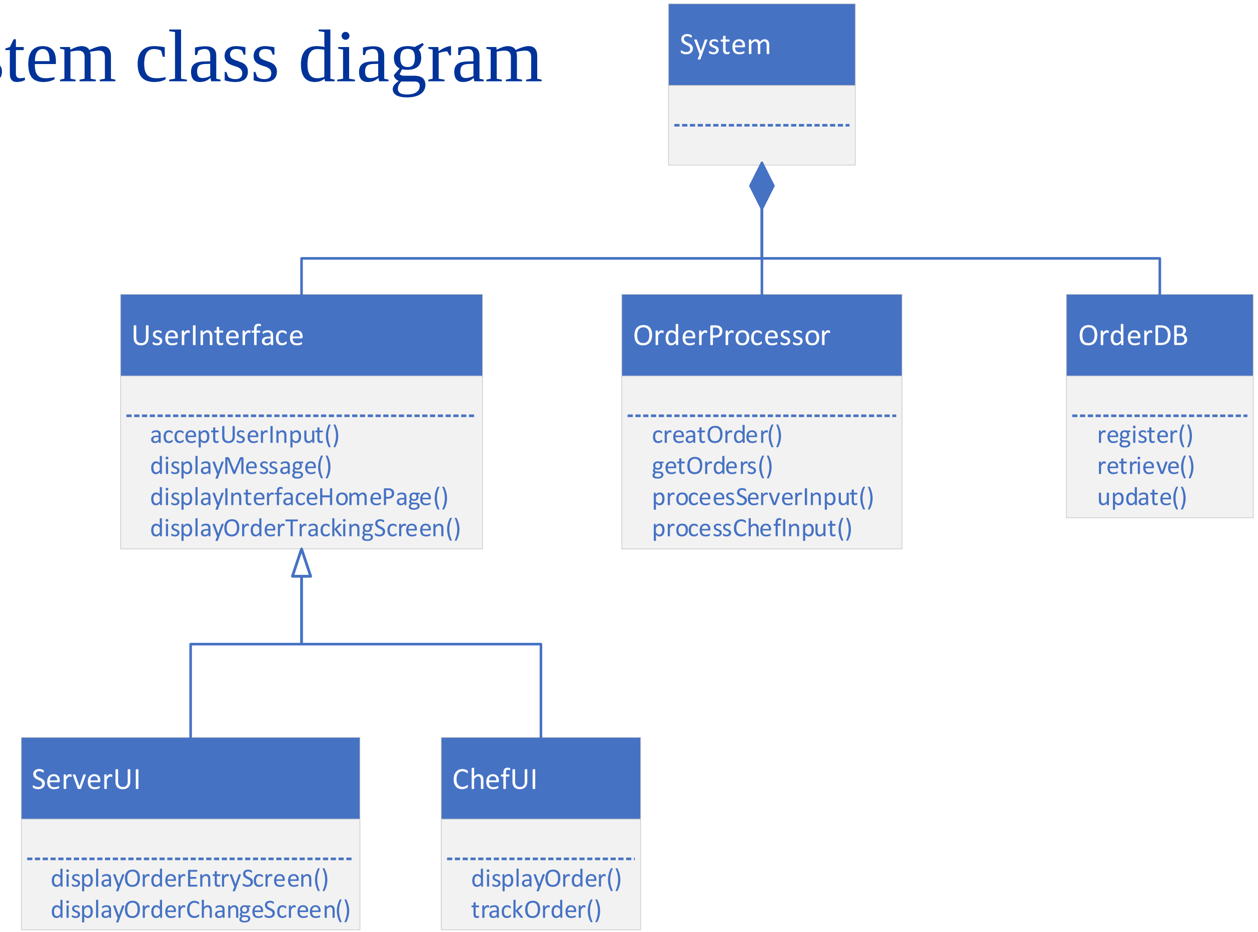
Discover system requirements (cont.)

- System requirements as use cases



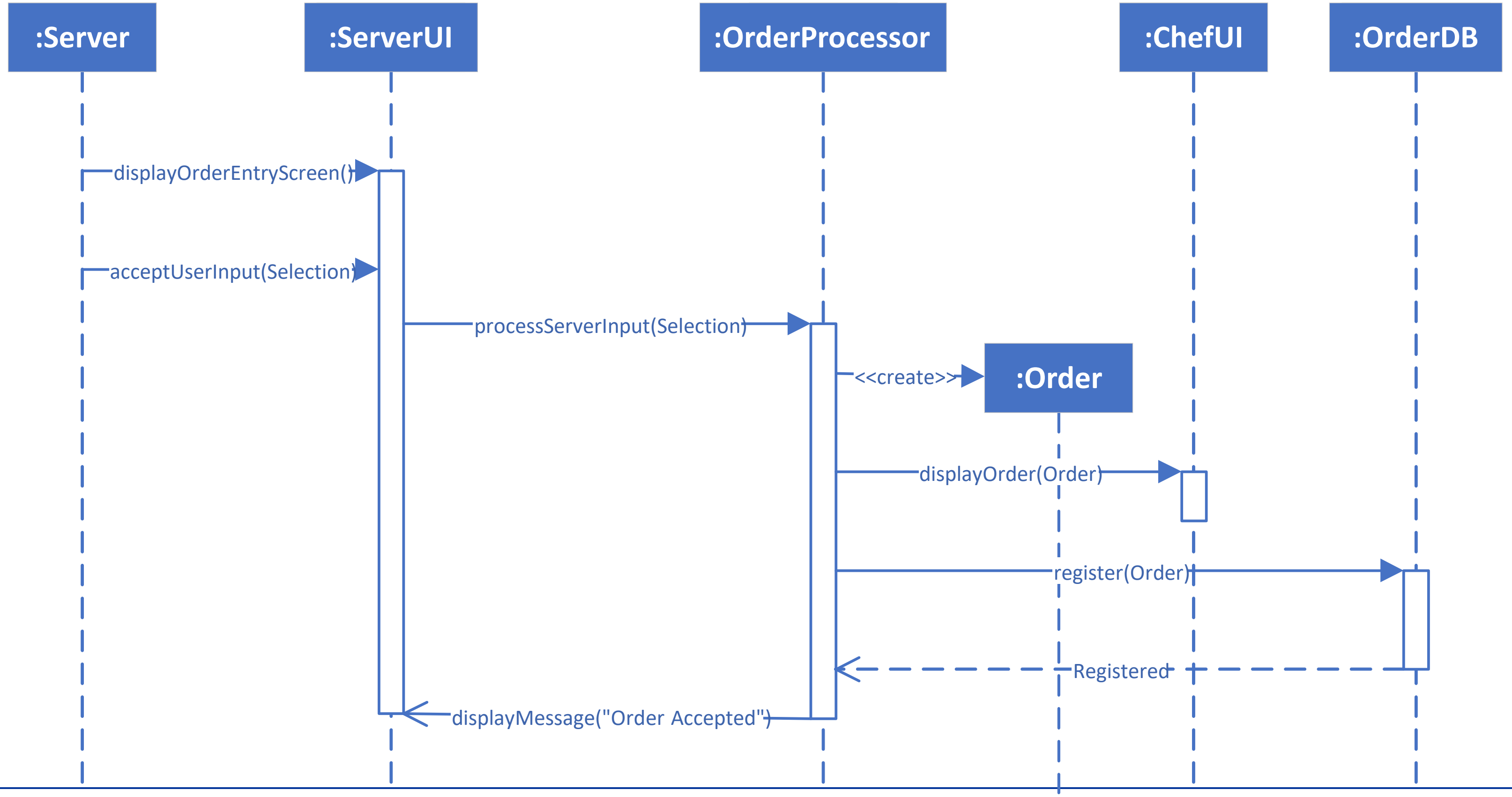
Discover system requirements (cont.)

- Enriched system class diagram



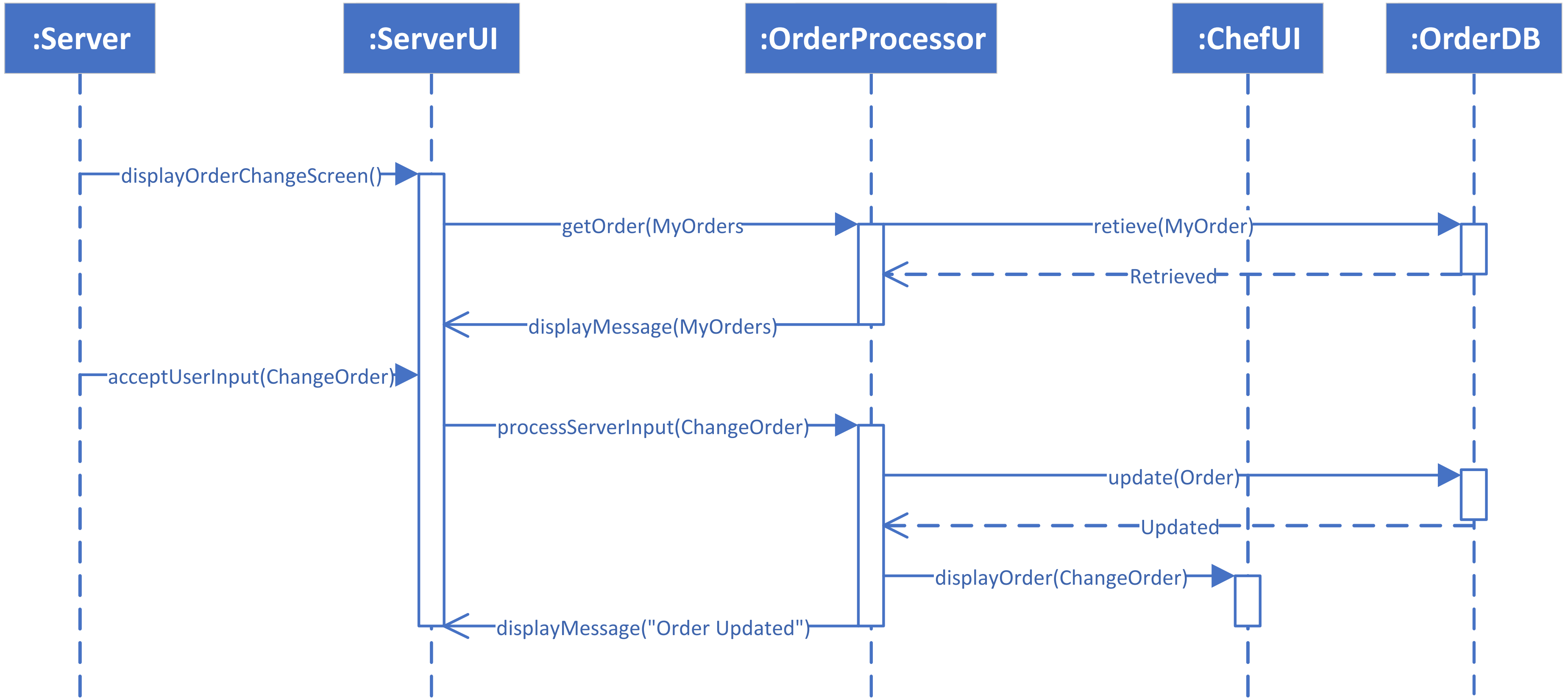
Identify interactions

- Use case “Take an order”



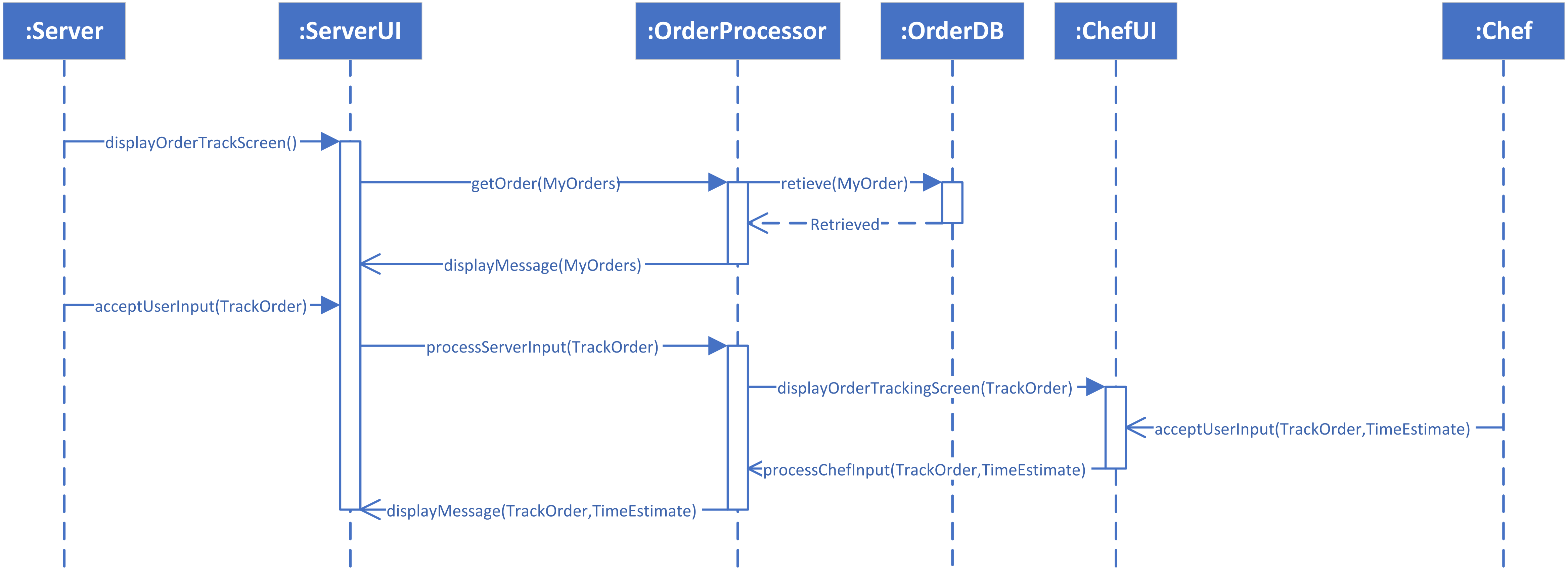
Identify interactions (cont.)

- Use case "Change an order"



Identify interactions (cont.)

- Use case “Track an order”



Why do we need models?

- Prediction
  - We know the low-level mechanisms but we want to understand how they affect higher-level behaviors
  - Use simulation instead of testing on the real system

- Explain the data
  - Make assumptions and use our knowledge to explain mechanisms that we don’t understand

- Classification
  - i.e. definitions, machine learning algorithms

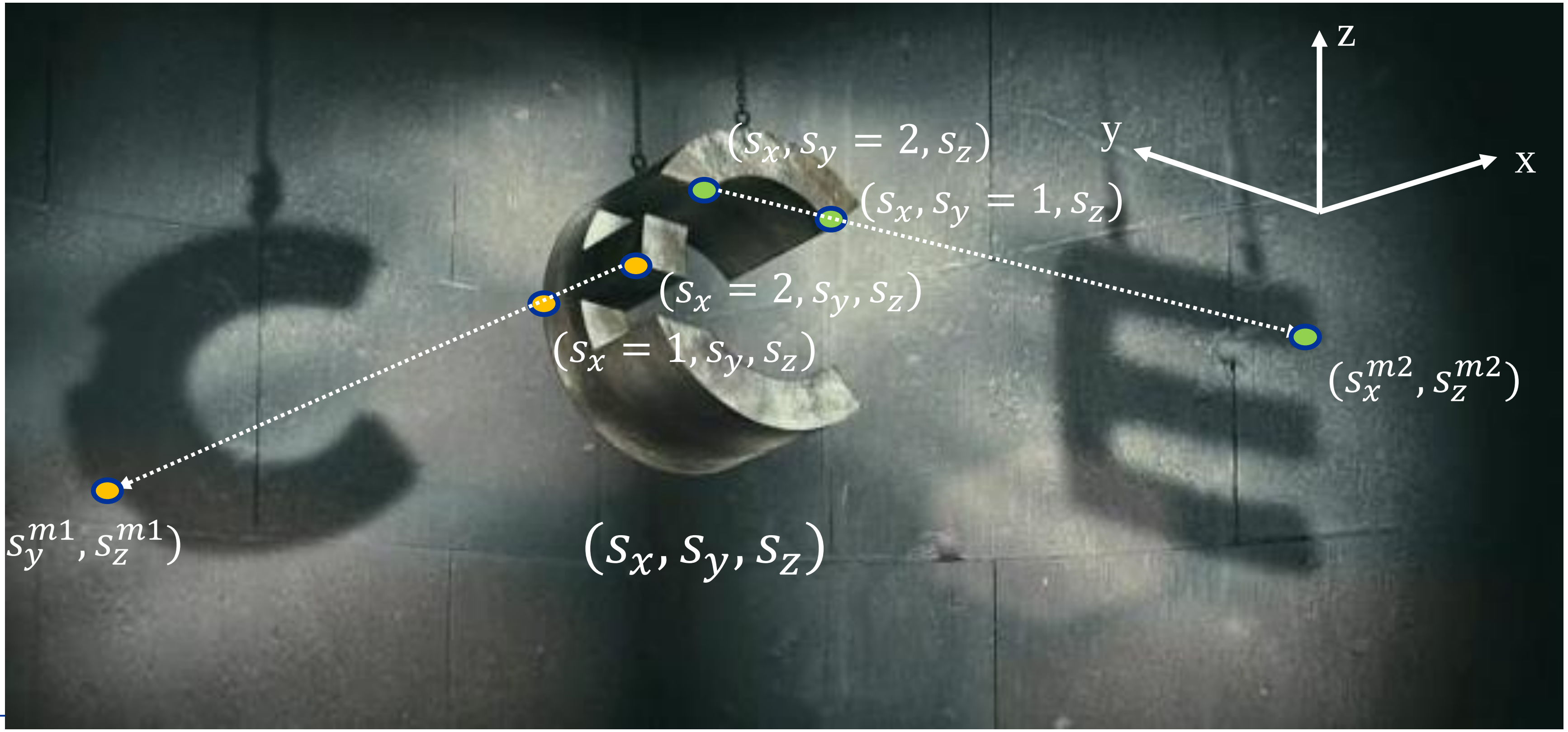
What are models?

- A system: (S, I, T, O)
  - S: States $s_1, s_2 \dots s_n$
  - I: Inputs (could be $\emptyset$)
  - T: Transitions $S \times I \times S$
  - O: Observations $f(S_o), S_o \subseteq S$

- Model of the system (S^m, I^m, T^m)
  - S^m: Abstraction/interpolation of S
    - Much fewer state variables
  - I^{m}: abstraction of I (could be $\emptyset$)
  - T^m: Transitions $S^m \times I^m \times S^m$

Abstraction – removal of state variables

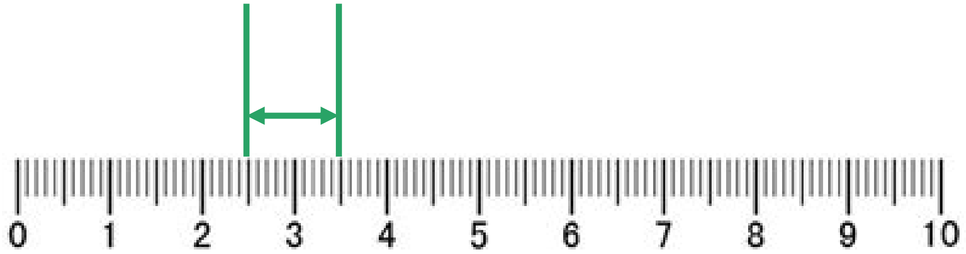
- States (s_x, s_y, s_z) *are abstracted to* (s_y^{m1}, s_z^{m1})
  - $(s_x, s_y, s_z) \rightarrow (s_y^{m1}, s_z^{m1})$
- Loss of information



Abstraction: Approximation of state variable values

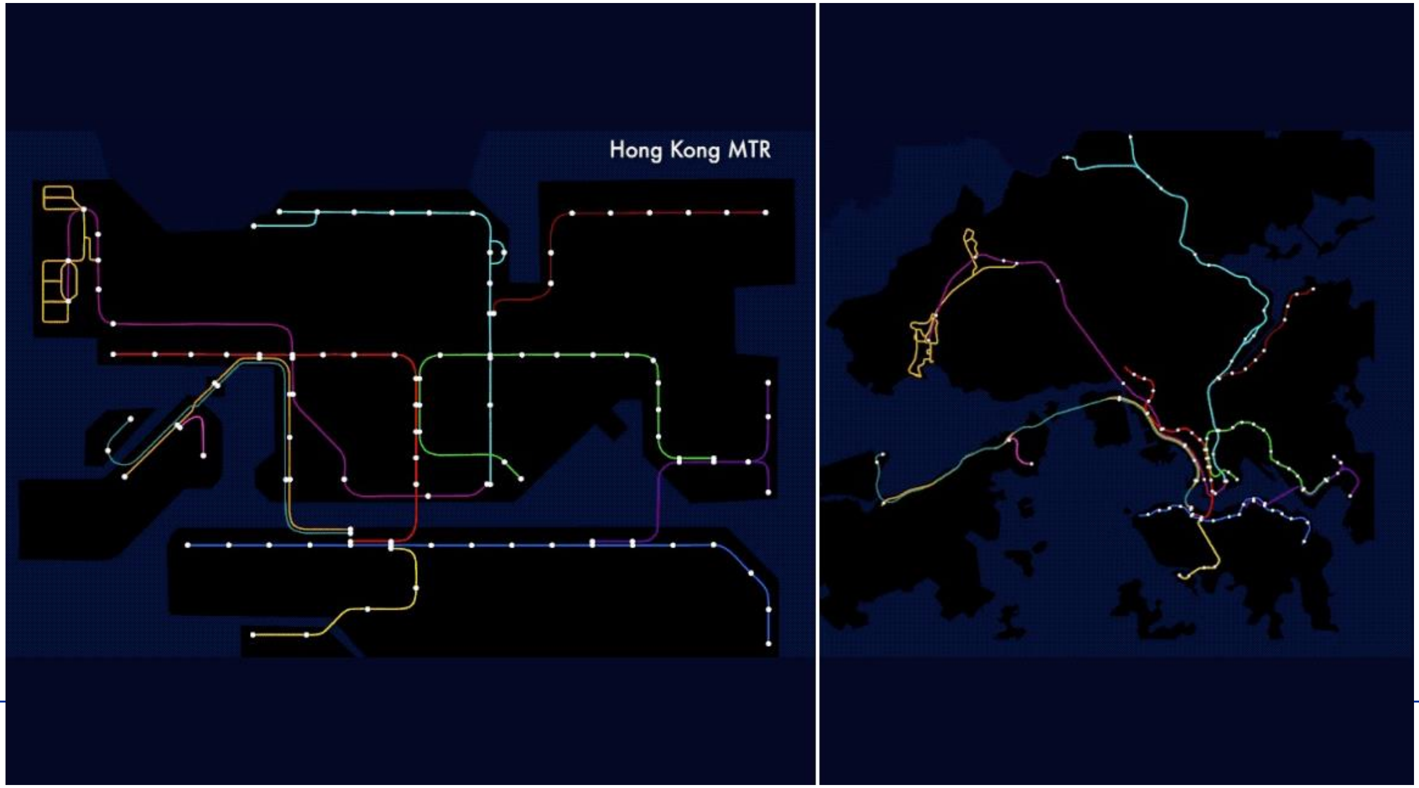
- Irrational numbers
  - $\pi \approx 3.1415$
  - $\sqrt{2} \approx 1.414$

- Approximation is another way of abstraction

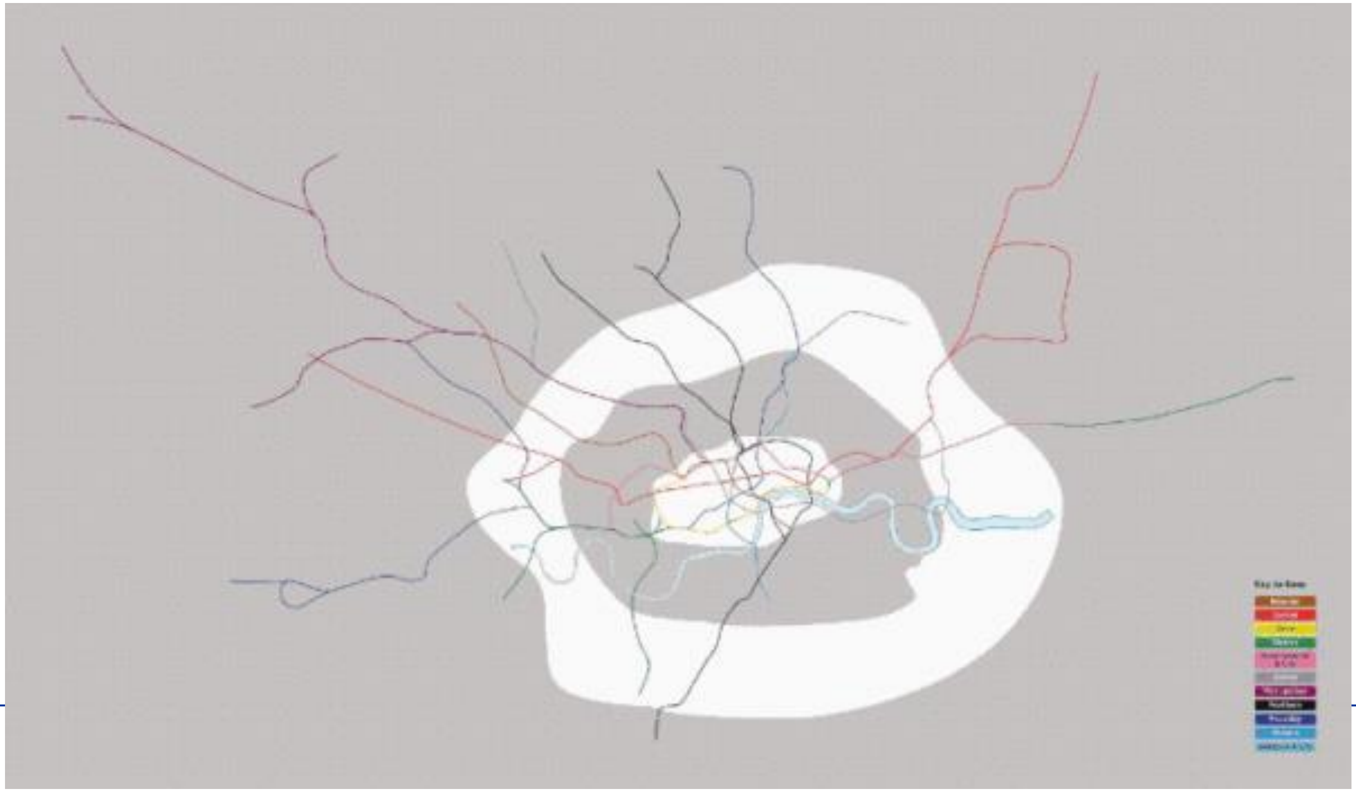


Interpolation: extracting interpretable information

- Locational information -> topological information
- $S^m = f(S_p), S_p \subseteq S$



More Interpolation: London MTR



What is considered as a “good” model?

- Accuracy
  - All models are wrong!
  - Error accumulates over time
  - Initial condition of the model cannot be determined due to limited observability

- Generality
  - The capability to explain not only training data, but also testing data

- Identifiability
  - Model parameters can be identified from data

- Interpretability
  - S^m are meaningful and interpretable by human

Newton vs. Einstein

- Newtonian physics is suitable for macro level objects at low speed
- $L = L_0\sqrt{1 - \frac{v^2}{c^2}}$
- A model can only be "good" within the context of its designated application
- The definition of "goodness" is changing over time

Modeling methodologies

- Bottom-up modeling
  – “White-box” model
  – Using first principles
  – Pros:
    - Interpretable
    - Convincing
  – Cons:
    - State space explosion
    - Difficult to be general
    - Low identifiability

- Data driven models (i.e. Neural networks)
  – “Black-box” model
  – From observable data
  – Pros:
    - No need to know domain knowledge
  – Cons:
    - Large and uninterpretable S^m
    - Depends highly on the quality and quantity of data