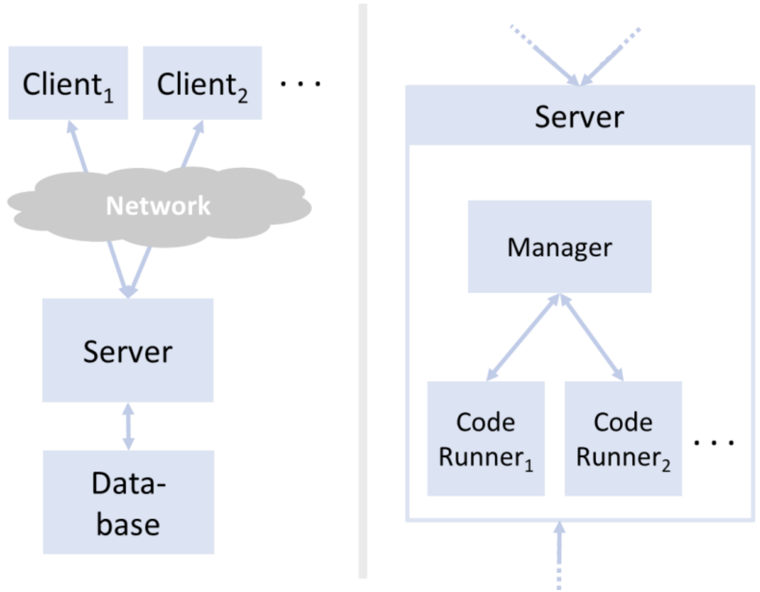
Software Engineering
(D-MATH CSE)

(Informal) Modeling with the UML

Marcel Lüthi, Malte Schwerhoff

Autumn Semester 2025

Zooming Out




```
// computes √x
double sqrt (double x) {
    double guess = x / 2.0;
    double last_guess = 0;
    while (std::abs(guess - last_guess) > 1e-12) {
        last_guess = guess;
        guess = (guess + x / guess) / 2.0’;
    }
    return guess;
}
```

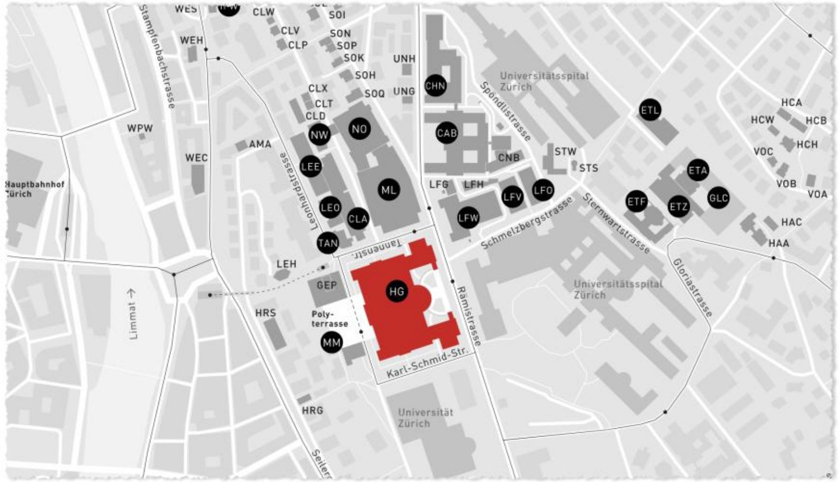
What do the diagram and the code comments have in common?

⇨ They abstract over an actual system, actual code

⇨ They provide us with a model of the reality

Examples of models

- A Map of ETH



- Newton’s Law

$$\mathbf{F} = m\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = m\mathbf{a}\,.$$

Discussion

"All models are wrong – but some are useful" – George Box

Discuss the following questions:

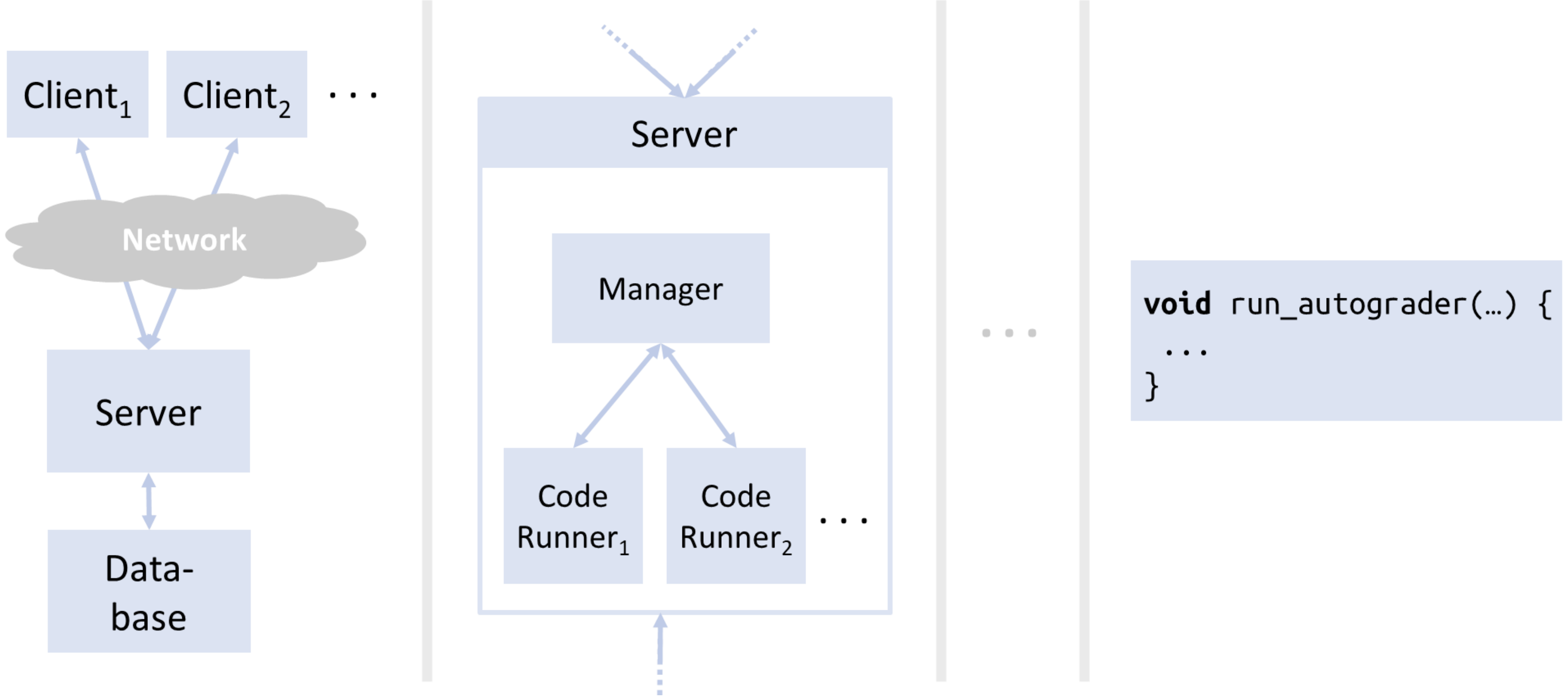
- Why are all models "wrong"?
- What does it mean for a model to be useful?
- Is a more accurate model always more useful? Why or why not?

What is Modeling?

- Building an abstraction of reality
  - Abstractions from things, people, processes, …
  - Relationships between these abstractions
- Abstractions are simplifications
  - They ignore irrelevant details
  - What is relevant or irrelevant depends on the purpose of the model
- Modeling is a means for dealing with complexity

Ideally enables drawing complicated conclusions about the reality through simple steps in the model

Modeling and Programming



- Gradually refine “black box components” to code

Modeling and Programming

Advantages of models over code:

- Focus on essential aspects, omit irrelevant details
- Postpone decisions until they must be taken: intentional underspecification

Focus (Examples)

- Many software engineering tasks require specific views on the system

- Examples
  - Software architecture: Which components depend on each other?
  - Test data generation: Is the algorithm expected to handle cyclic data structures?
  - Security audit: Is the communication protocol prone to attacks?
  - Deployment: Which software component runs on which hardware?

Relevant information difficult to extract from code

Underspecification (Examples)

- Relationships

```
class University {
  set<Student> students;
  ...
}
```

```
class Student {
  Program major;
  ...
}
```

```
class University {
  map<Student, Program>
    enrollment;
  ...
}
```

```
class Student {
  ...
}
```

One university, many students,
each of which has a major → decision for concrete data structure too early

- Conditions

```
class BankAccount {
  bool flagged;

  void detect_irregularities(...) {
    if (/* something fishy */)
      flagged = true;
  }
}
```

Difficult to leave definition of “something fishy” open. Premature implementation may influence further discussions and decisions.

The Unified Modeling Language UML

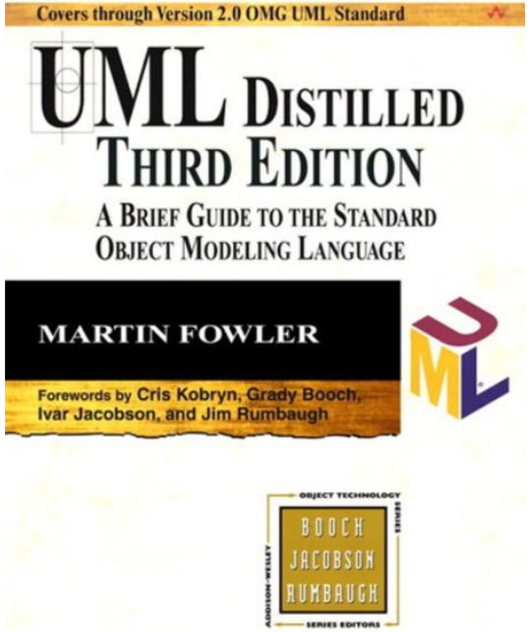
- UML is a modeling language
  - Using text and graphical notation
  - For documenting specification, analysis, design, and implementation
- Relevancy
  - De facto standard in industrial software development
  - Unified and standardized various older modeling approaches
  - First version from 1994, ISO standard since 2005, last version from 2017 (UML 2.5.1)

UML Notations

- Use case diagrams – requirements of a system
- Class diagrams – structure of a system
- Interaction diagrams – message passing
  - Sequence diagrams
  - Collaboration diagrams
- State and activity diagrams – actions of an object
- Implementation diagrams
  - Component model – dependencies between code
  - Deployment model – structure of the runtime system
- Object constraint language (OCL)

UML Resources

- Many books available, e.g. from ETH library
- Standard publicly available: https://www.omg.org/spec/UML/ (PDF has >700 pages!)
- Many UML summaries/cheatsheets can be found online, e.g.
  - https://modeling-languages.com/best-uml-cheatsheets-and-reference-guides/
  - https://pl.cs.jhu.edu/oose/resources/uml-cheatsheet.pdf



Modeling and Specification

- Informal Models
  - Static Models
  - Dynamic Models
  - Mapping Models to Code

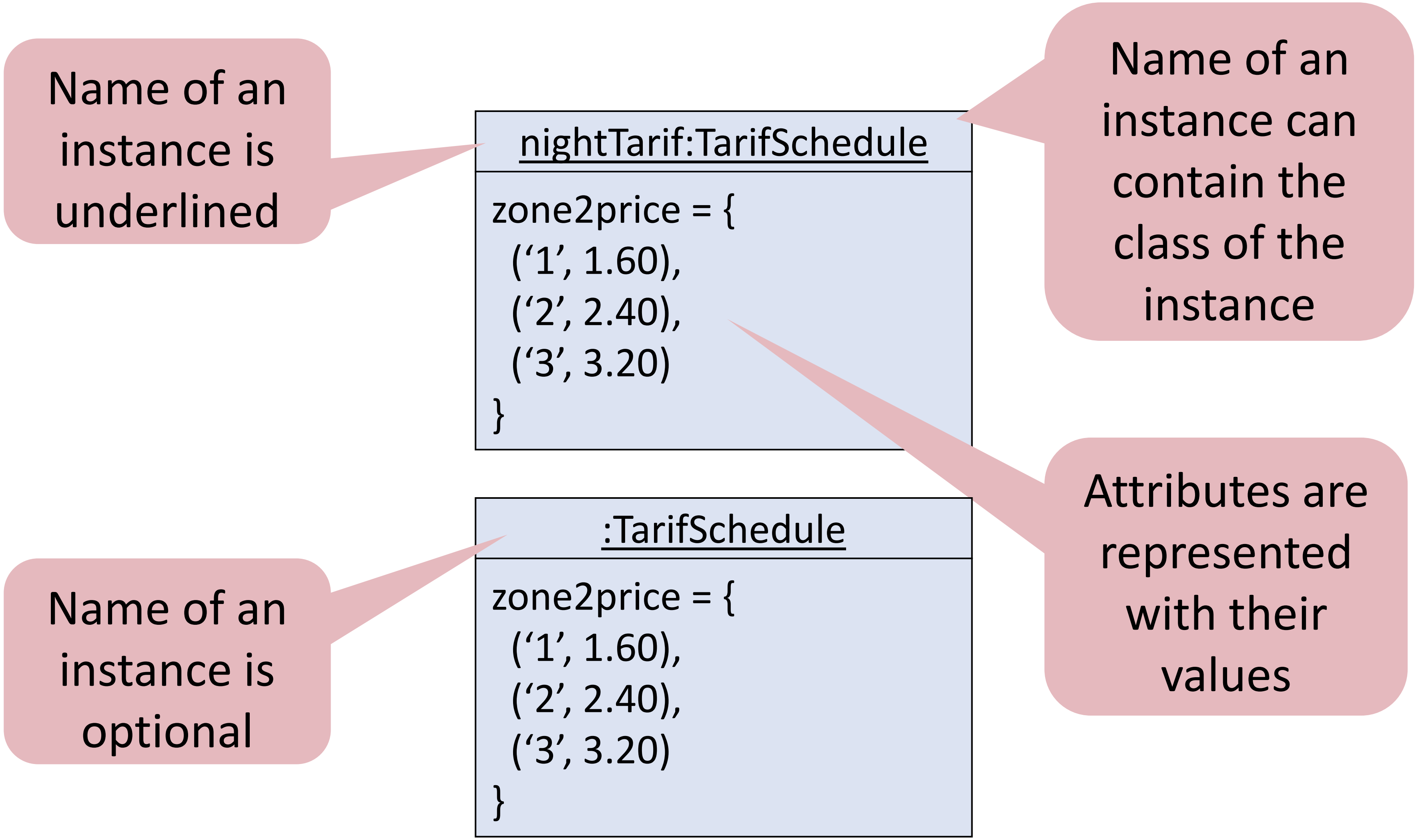
Classes

- A class includes state (attributes) and behavior (operations)
  - Each attribute has a type
  - Each operation has a signature

More on Classes

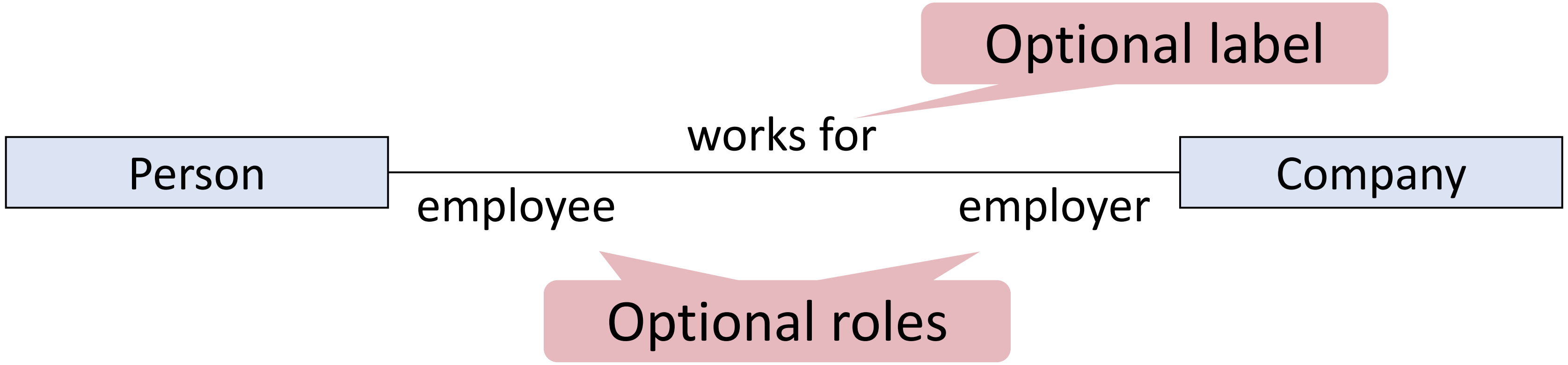
- The class name is the only mandatory information
- Examples of valid UML class diagrams

Instances (Objects)



Associations

- A link represents a connection between two objects
  - Object A can send messages to/communicate with object B
  - Object A has an attribute whose value is B
  - Object A creates object B
  - …
- Associations denote relationships between classes



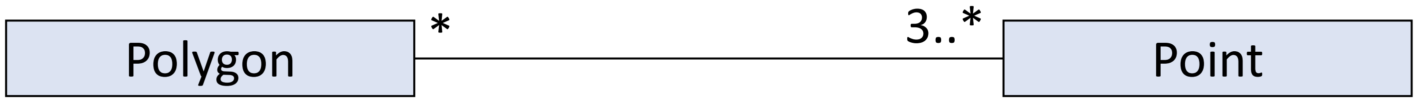
For the brave at heart: see https://www.omg.org/spec/UML/2.5.1/PDF#page=226 and https://www.omg.org/spec/UML/2.5.1/PDF#page=241

Multiplicity of Associations

- The multiplicity of an association end denotes how many objects the source object can reference
  - Exact number: 1, 2, etc.
  - Arbitrary number: * (zero or more)
  - Range: 1..3, 1..*

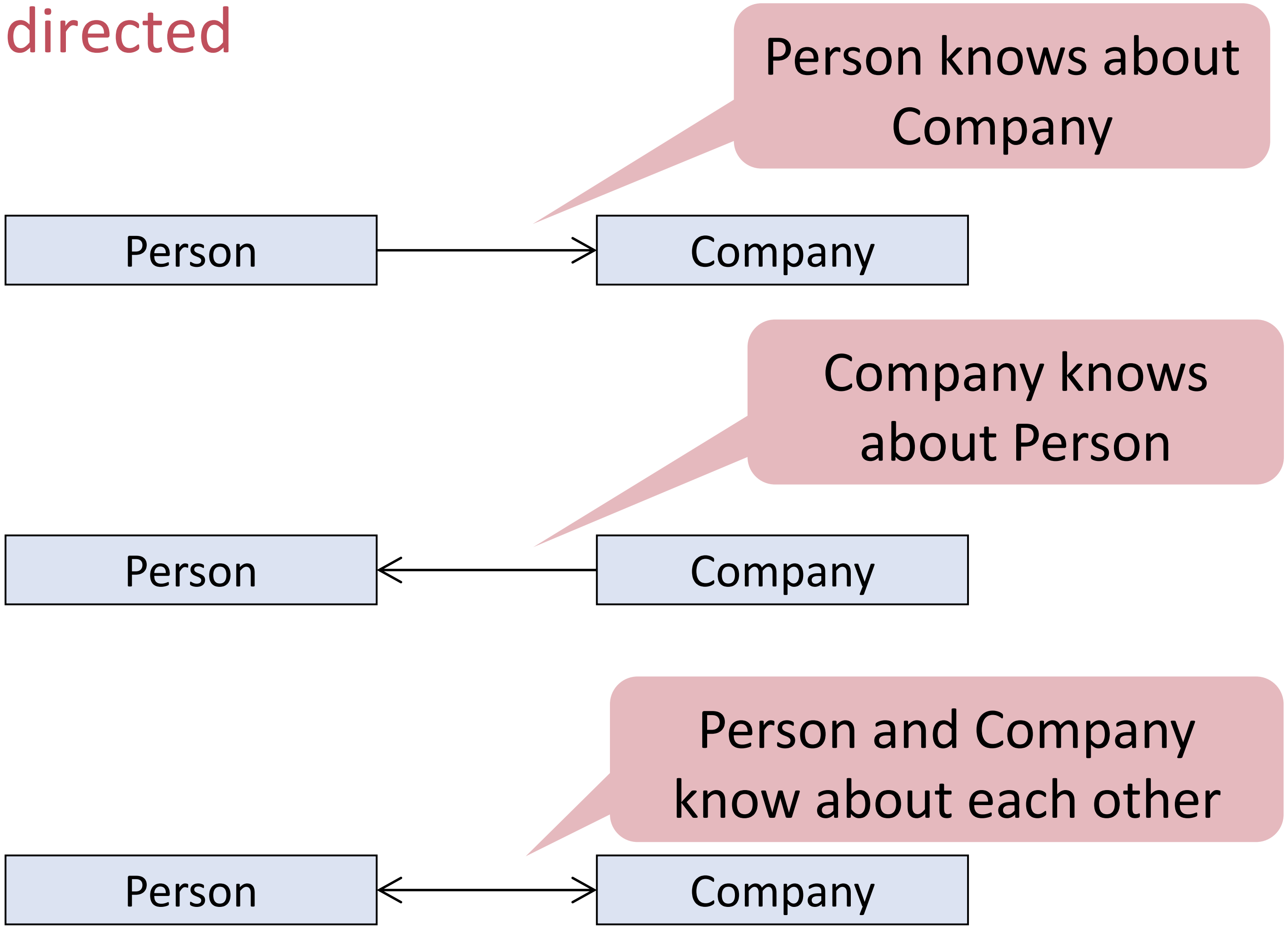
- 1-to-(at most) 1 association

- Many-to-many association



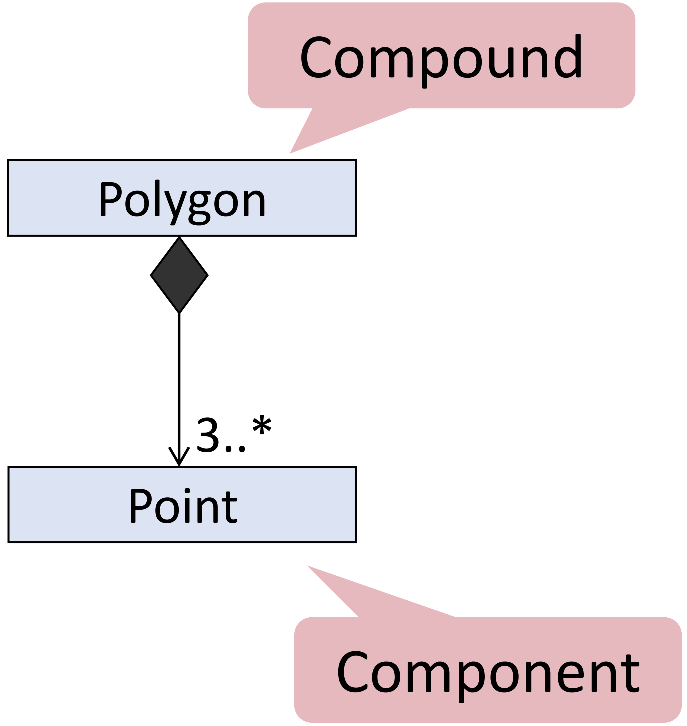
Navigability

- Associations can be directed



Composition

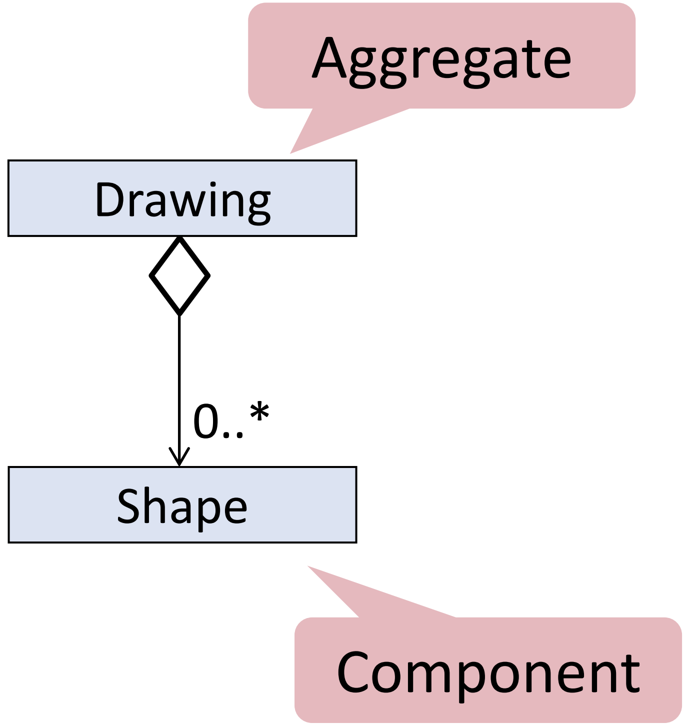
- Composition expresses an exclusive part-of (“has-a”) relationship
  - Special form of association
  - Composed parts (components) cannot exists independently (exclusive ownership)
    - Implies no sharing
- Composition can be decorated like other associations: multiplicity, label, roles



Aggregation

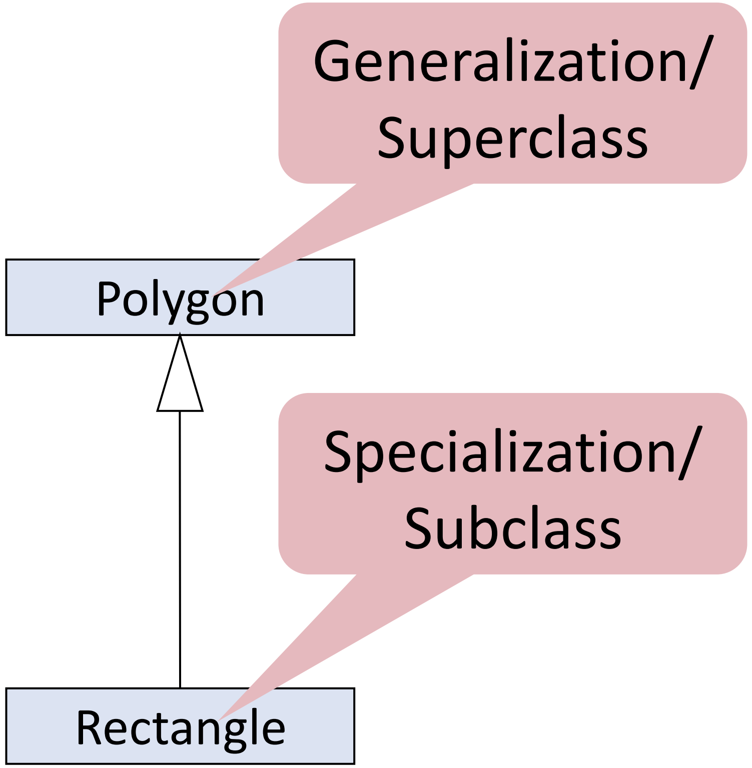
- UML also has aggregation (hollow diamond):
  - Aggregated parts can exist independently

If difference is not crucial, use aggregation



Generalization and Specialization

- Generalization expresses a kind-of ("is-a") relationship
- Generalization is implemented by inheritance
  - The child classes inherit the attributes and operations of the parent class
- Generalization simplifies the model by eliminating redundancy



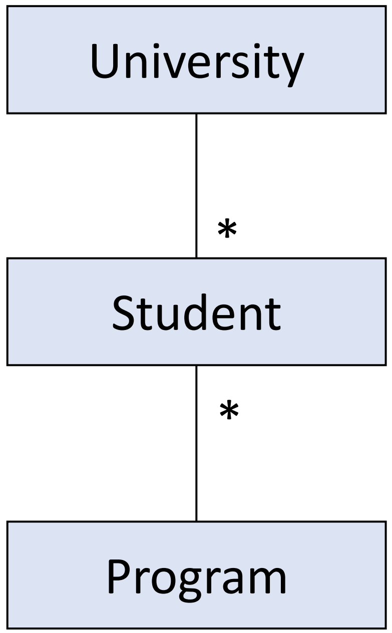
Recall Underspecification Discussion

```
class University {
  set<Student> students;
  ...
}
```

```
class Student {
  Program major;
  ...
}
```

```
class University {
  map<Student, Program>
    enrollment;
  ...
}
```

```
class Student {
  ...
}
```



- The class diagram leaves the choice of data structure unspecified

Mini exercise

- Draw an UML class diagram to model this simple statement“
  “A person can be married to another person”

Modeling and Specification

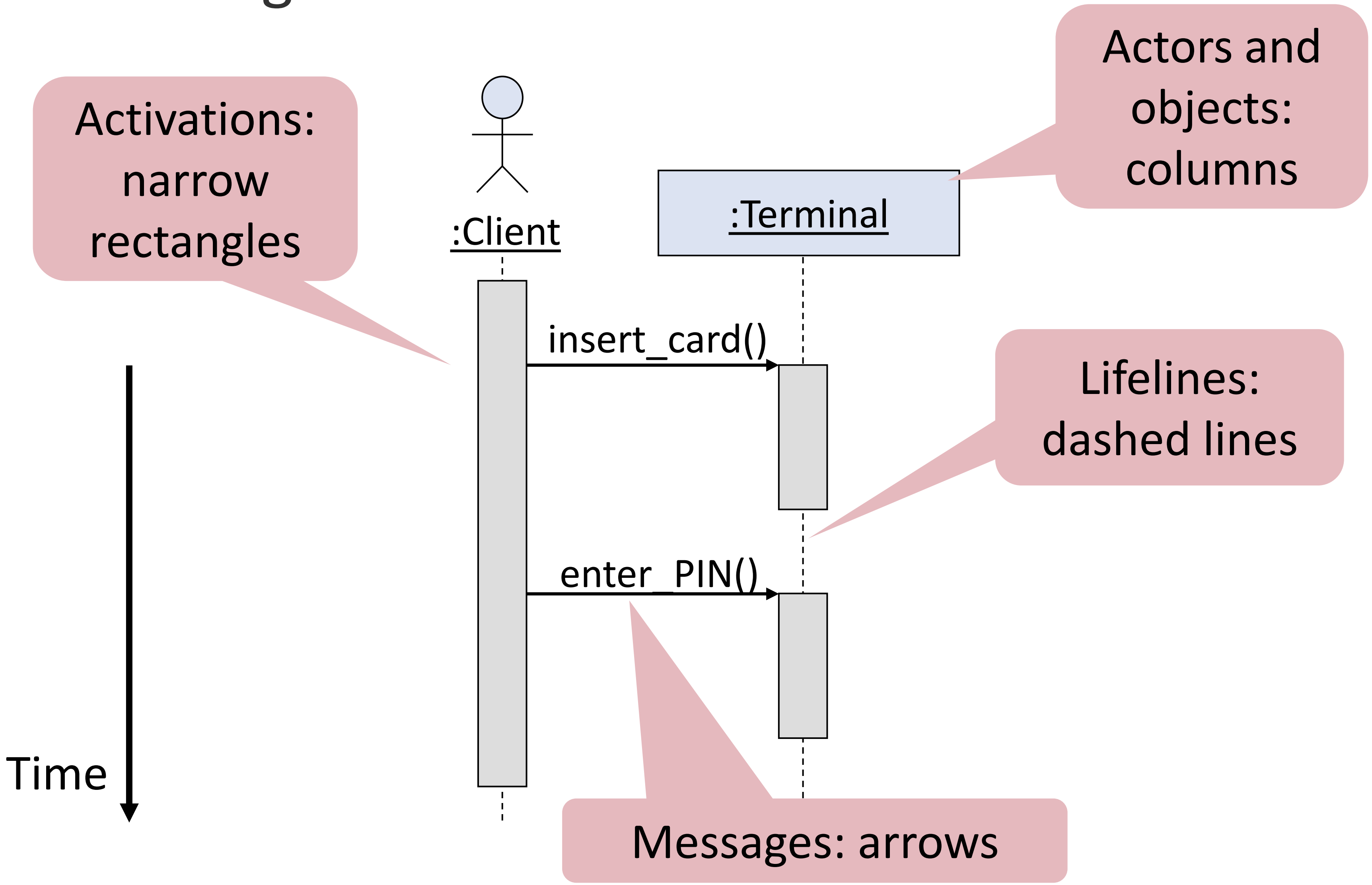
- Code Documentation
- Informal Models
  - Static Models
  - Dynamic Models
  - Mapping Models to Code

Dynamic Models

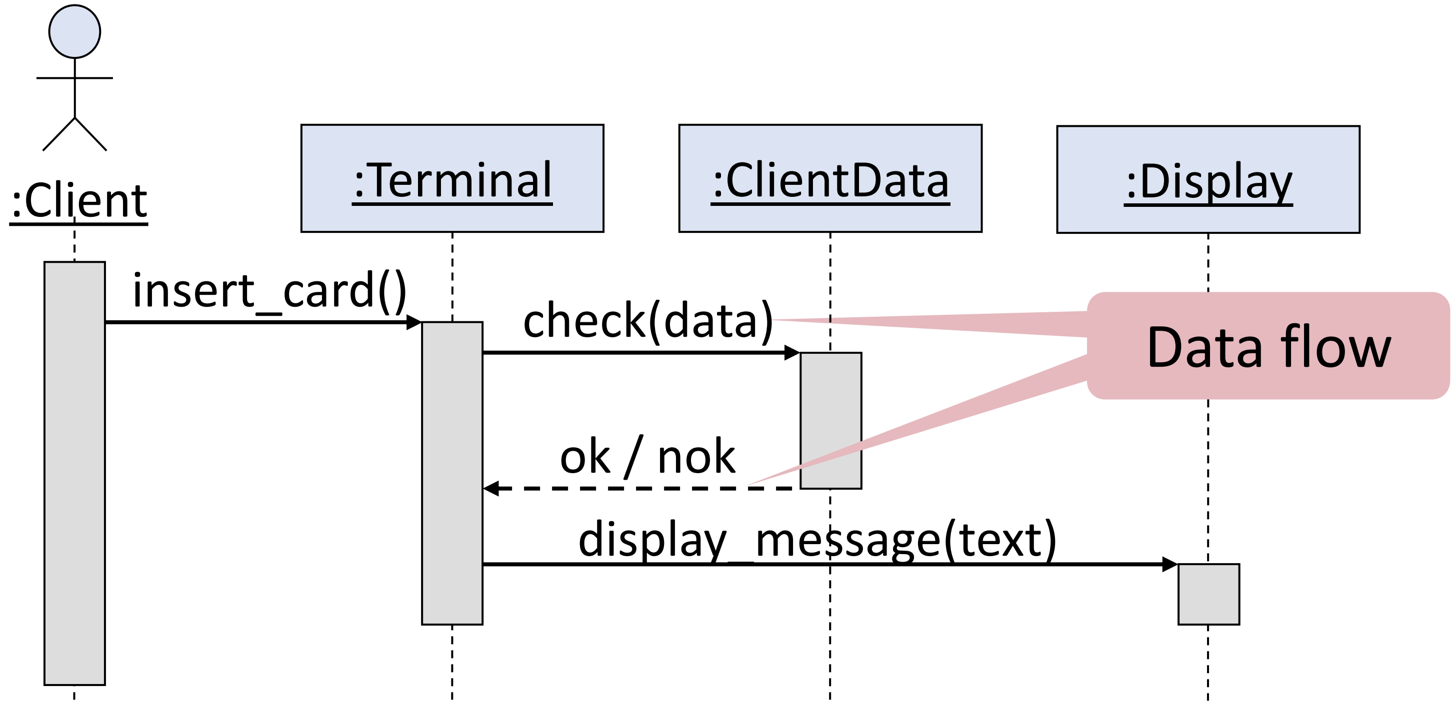
- Static models describe the structure of a system
- Dynamic models describe its behavior

Sequence diagrams describe collaboration between objects	State diagrams describe the lifetime of a single object

UML Sequence Diagrams

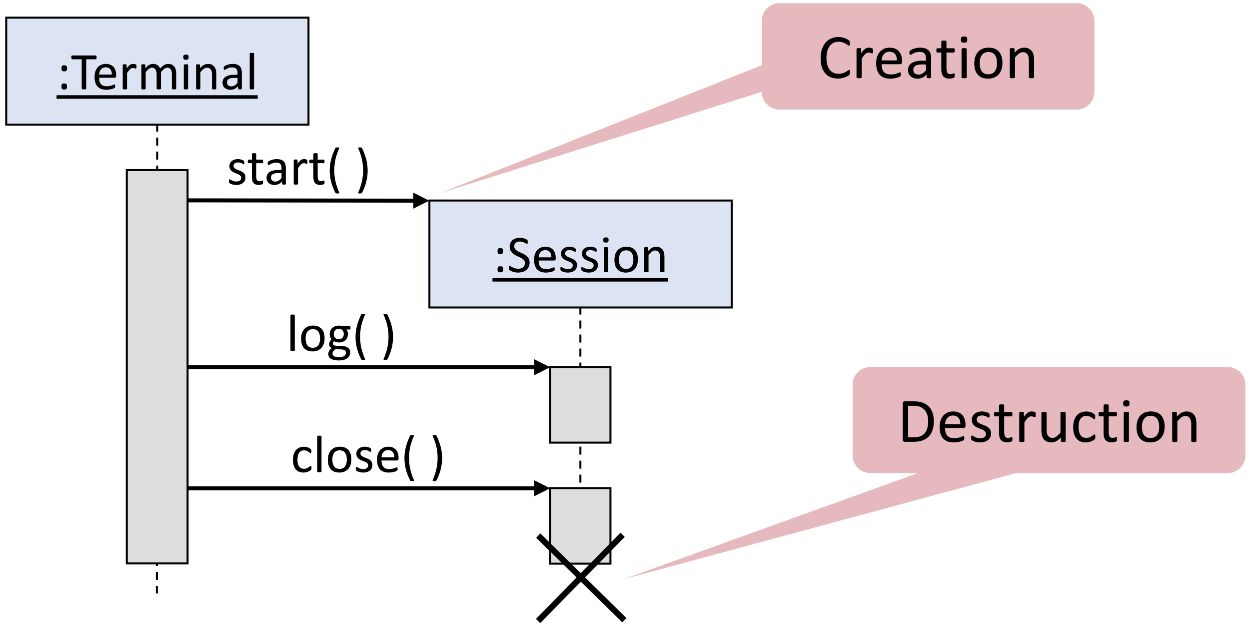


Nested Messages



- The source of an arrow indicates the activation which sent the message
- An activation is at least as long as all nested activations

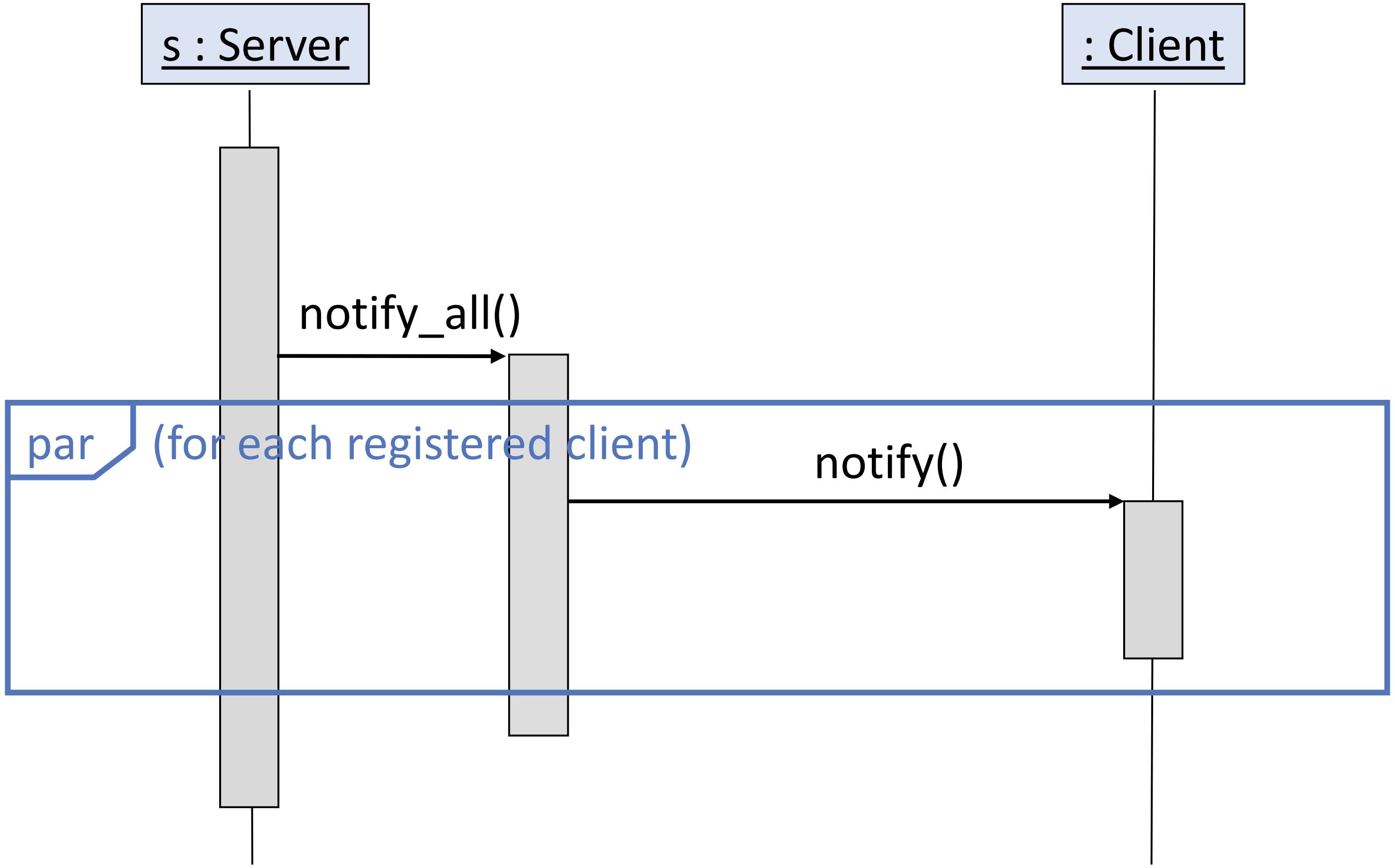
Creation and Destruction



- Creation is denoted by a message arrow pointing to the new object
- Destruction is marked by an X at the end of the object’s lifeline

Example: Underspecification and Views

- Underspecification: par = potentially parallel, not (yet) committed to an order
- View:
  - Server-centric (server vs. client activation)
  - Core protocol (notifications); no data & data format, unclear how server & client establish first contact



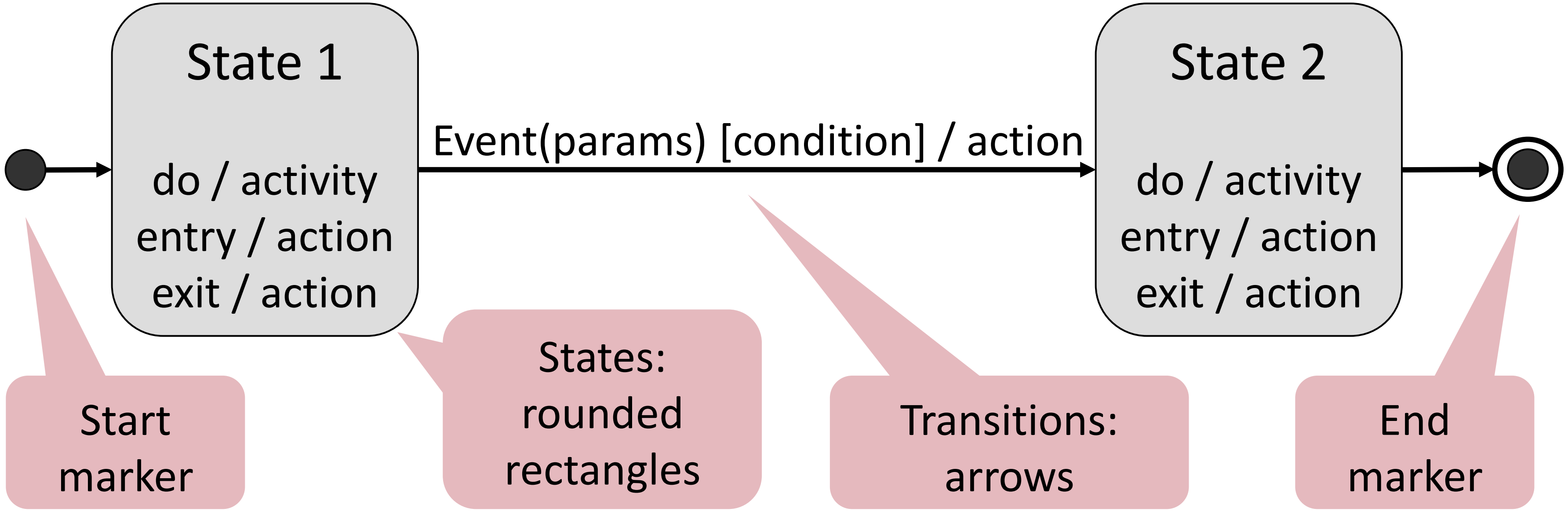
One minute Challenge:

- Draw an UML sequence diagram to model this simple statement:
  “Bob is proposing to Alice to get married. Alice thinks then answers yes”

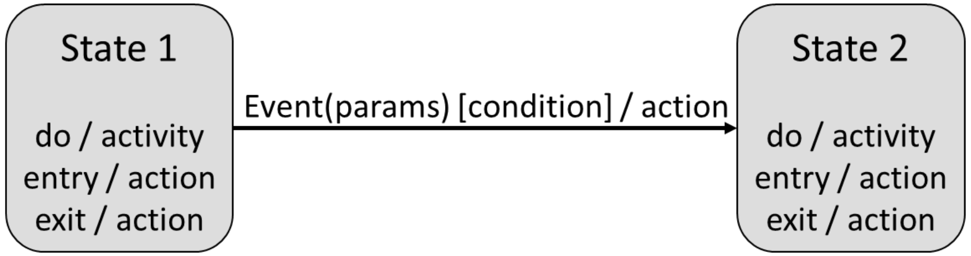
State

- An abstraction of the attribute values of an object
- A state is an equivalence class of all those attribute values and links that do not need to be distinguished for the control structure of the class
- Example: State of a bank account
  - An account is in state open, flagged, closed
  - Omissions: account number, owner, etc.
  - All open accounts are in the same equivalence class, independent of their number, owner, etc.

UML State Diagrams

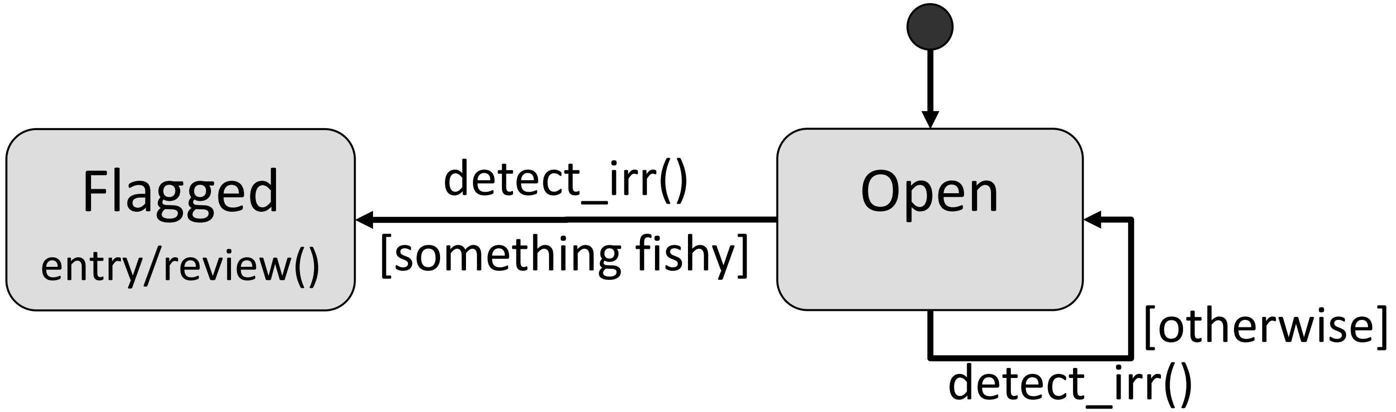


Events, Actions, and Activities



- Activity: Operation performed as long as object is in some state
  - Example: Object watches for file system changes, or refines a result (numerical computation)
- Event: Something that happens at a point in time
  - Examples: Receipt of a message, change event for a condition, time event
- Action: Operation in response to an event
  - Example: Object performs a computation upon receipt of a message

Example: Underspecification



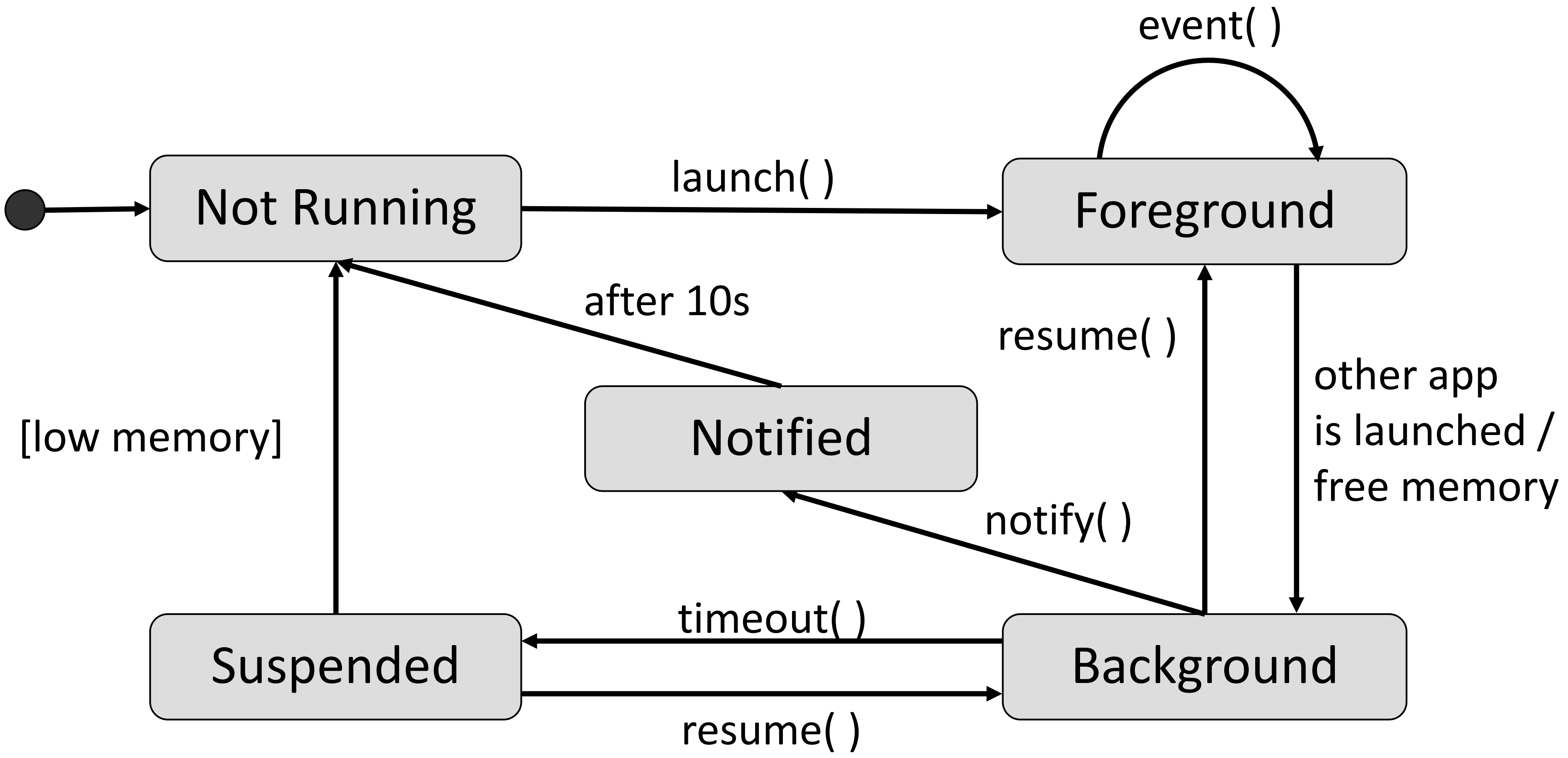

```
class BankAccount {
 bool flagged;

 void detect_irregularities(...) {
  if (/* something fishy */)
   flagged = true;
 }
}
```

- Conditional can be left informal
- Diagram is deliberately incomplete, e.g. post-review transitions missing
- Concrete implementation still needs to handle unexpected messages and message arguments

Example: Views

Simplified iOS app lifecycle diagram that answers the question
“May an app be terminated without prior notification?”



One minute Challenge:

- Draw a state diagram that illustrates Alice’s state (remember Bob’s proposal)

Modeling and Specification

- Code Documentation
- Informal Models
  - Static Models
  - Dynamic Models
  - Mapping Models to Code

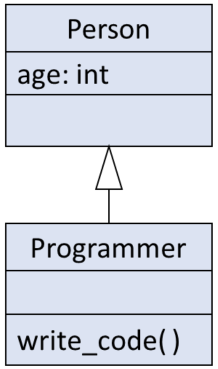
Discussion

- Discuss the following idea: Instead of writing code directly, programs are produces from a sequence of ever more detailed models, and thereby
  - raise the level of abstraction
  - make programming languages superfluous
  - make it easier for non-technical staff to write programs

Model-Driven Development: Idea

- Work on the level of design models
- Generate code automatically

- Advantages
  - Supports many implementation platforms
  - Frees programmers from recurring activities
  - Leads to uniform code
  - Useful to enforce coding conventions (e.g., getters and setters)
  - Models are no longer “mere” documentation

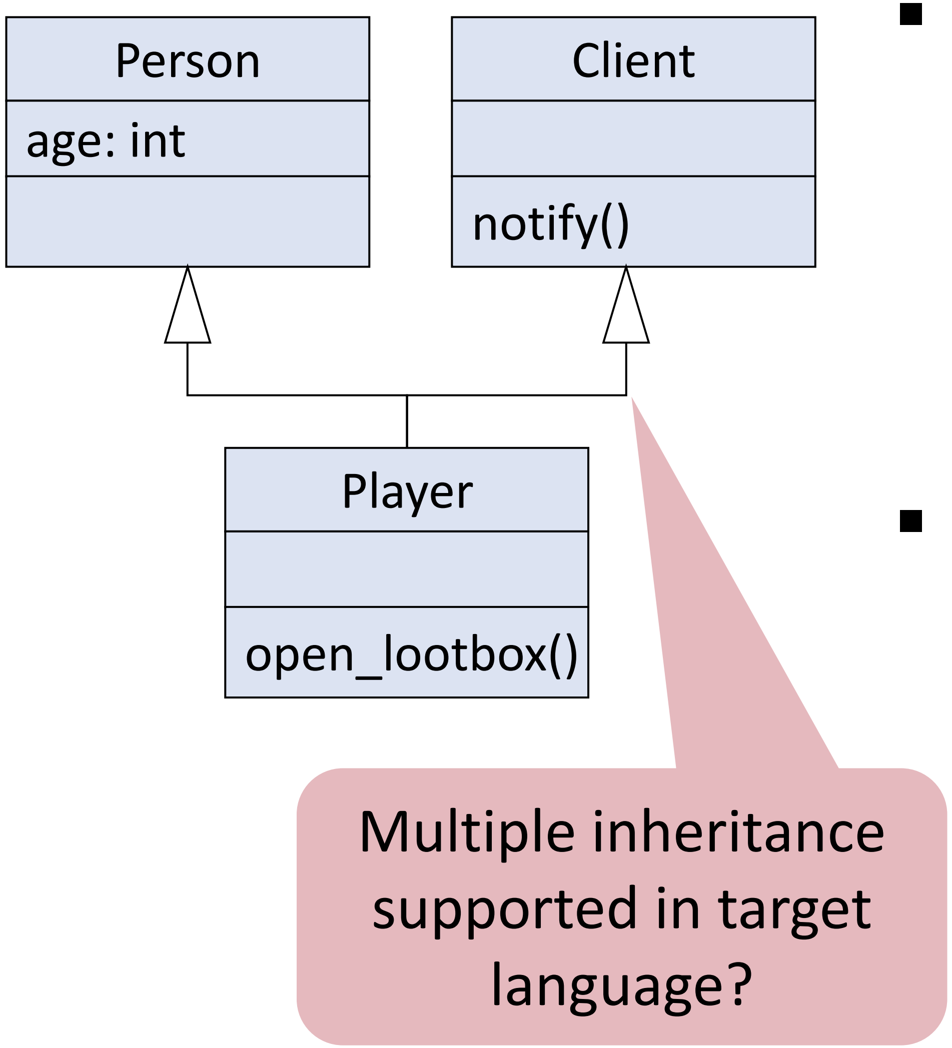



```
class Person {
  unsigned int age;

public:
  set_age(unsigned int a)
    { age = a; }
  unsigned int get_age()
    { return age; }
  ...
};
```

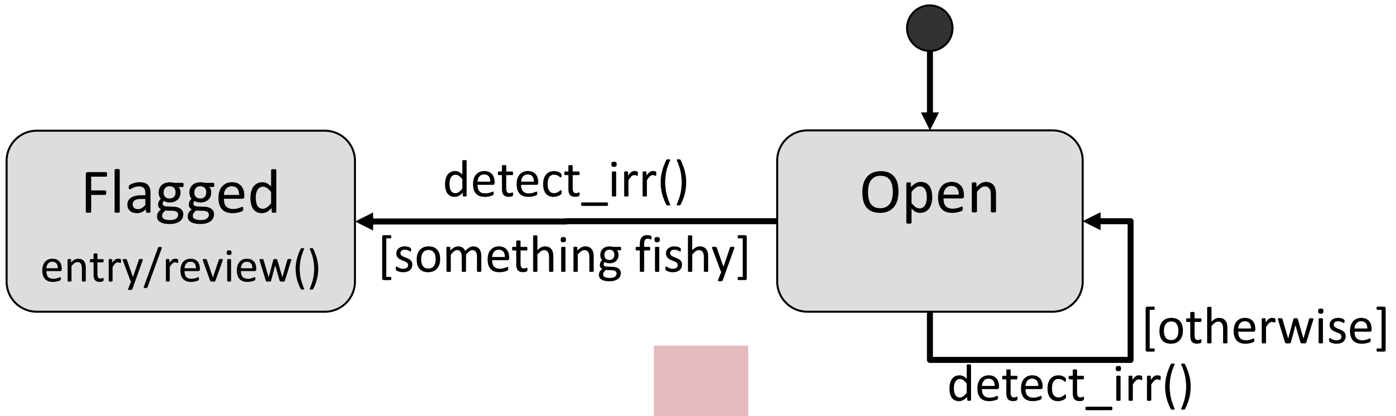
```
class Programmer: public Person {
public:
  void write_code() { ... }
  ...
};
```

MDD Problems: Abstraction Mismatch



- UML models
  - may use different abstractions than the programming language
  - should not depend on implementation language
  - e.g. Java (OOP) vs. Haskell (FP)
- Models, e.g. due to multiple inheritance, cannot always be mapped to code directly

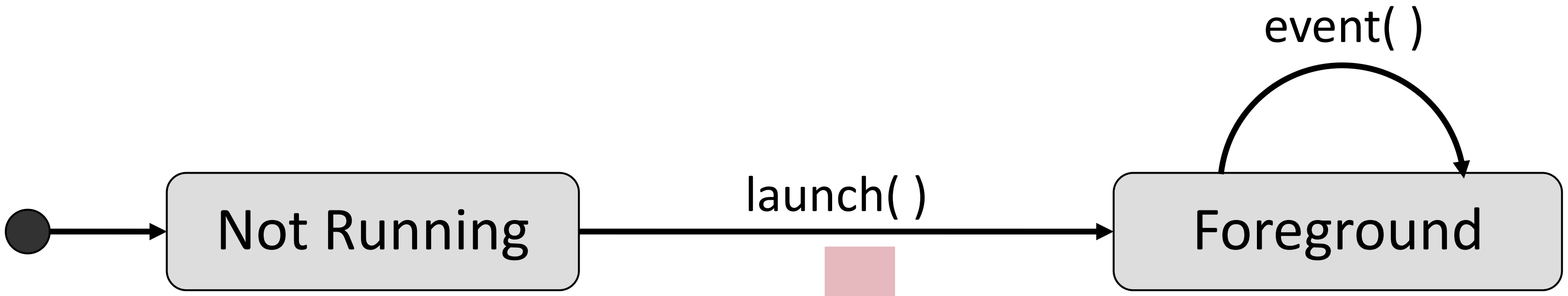
MDD Problems: Specifications may be Informal




```
void open() {
  assert(state == OPEN);
  assert(/* something fishy || otherwise*/);
  if (/* something fishy */)  {
    state = OPEN;
    review();
  }
}
```

How to map informal specifications?

MDD Problems: Specifications are Incomplete




```
class App {
  State state;
public:
  void launch() {
    assert(state == NOT_RUNNING);
    state = FOREGROUND;
  }
  public void event() {
    assert(state == FOREGROUND);
  }
}
```

Where is the interesting behavior?

MDD Problems: Switching between Models and Code

- Code has to be changed manually
  - Add interesting behavior
  - Clarify informal specifications
  - Implement incomplete specifications
  - Optimizations (memory, runtime, user experience, …)

- Modification of code requires complicated synchronization between code and models

Model-Driven Development: Reality

- Works in specific domains, e.g.,
  - Business Process Modeling (BPML)
  - Event-driven graphical user interfaces
  - Stereotypical games: 2D platformers, point & click adventures, …
- Code generation works for basic, standardized properties
- Interesting code is still implemented manually
- Problems
  - Mapping manual code changes back to the model is difficult, often impossible
  - Maintaining code that has no models (reverse-engineering)

Informal Modeling: Summary

Strengths

- Describes particular views on the overall system
- Abstracts information
- Allows to specify information informally
- Graphical notation facilitates communication and informal reasoning

Weaknesses

- Precise meaning of models is often unclear
- Incomplete and informal models hamper tool support
- Graphical notation limits level of detail (they become hard to depict)

Next week: Formal modelling with Alloy