SOFTWARE MODELING

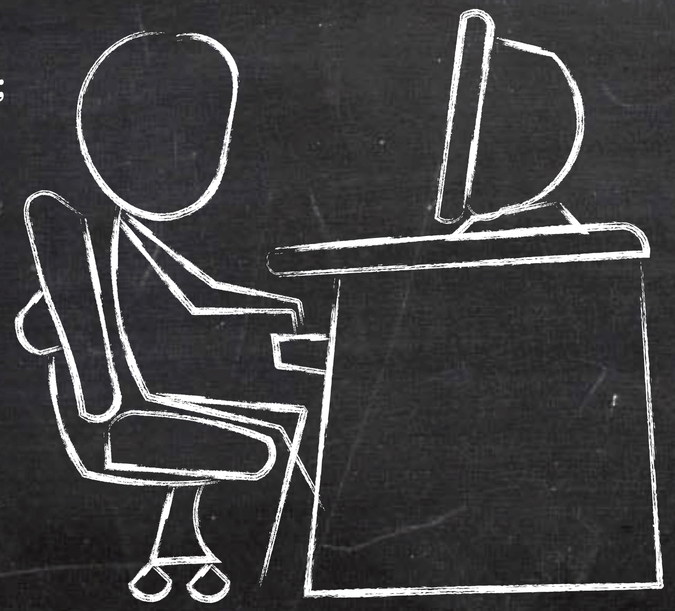
Motivation

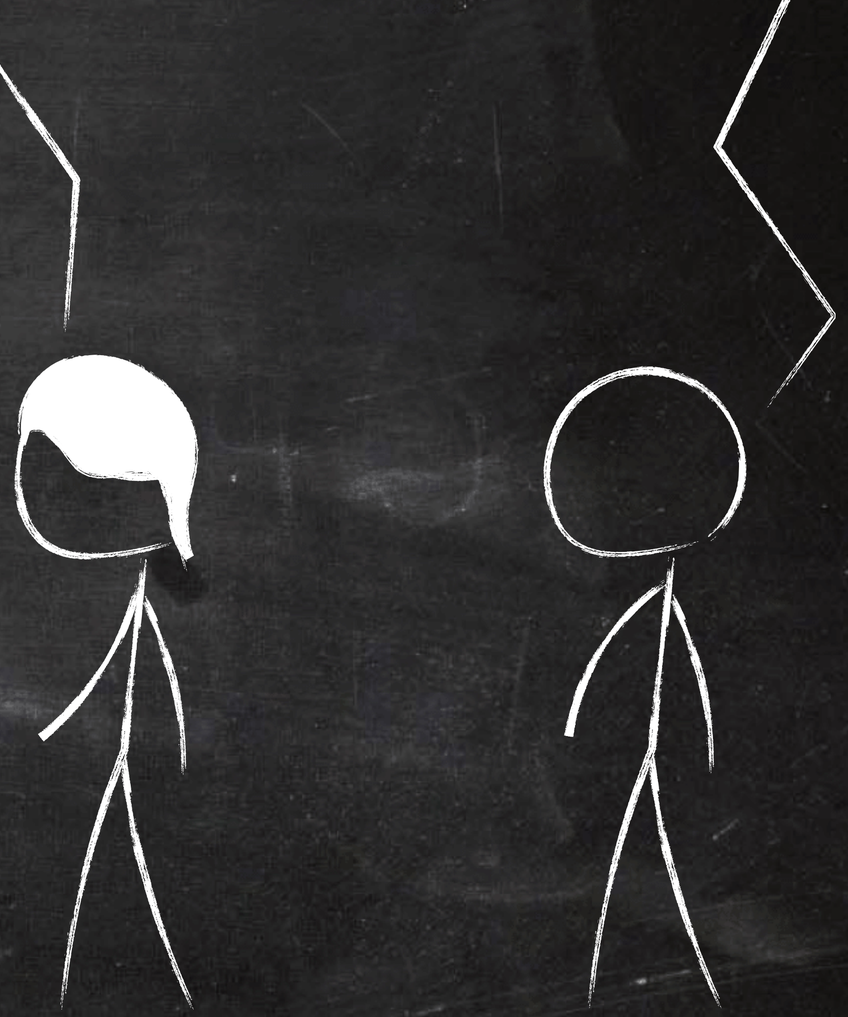
HOW TO ACCESS ALL ITEMS IN A LIST?

```
@Override public Iterator<T> iterator() {
        return new Iterator<T>() {
            private int idx = 0;
            @Override public boolean hasNext() { return idx < size && list[idx] != null; }
            @Override public Type next() { return list[currentIndex++]; }
        }; }
```

NO, NO. IT'S MORE LIKE THIS…

```
template <class Derived, class Value, class CategoryOrTraversal, class Reference =
Value&, class Difference = ptrdiff_t>
class iterator_facade {
 public:
   typedef remove_const<Value>::type value_type;
   typedef Reference reference;
   typedef Value* pointer;
   typedef Difference difference_type;
   reference operator*() const;
  Derived& operator++();
   Derived operator++(int);
   Derived& operator—();
…
```





NEED FOR SIMPLE IMPLEMENTATION— INDEPENDENT WAY TO PRESENT STRUCTURE, BEHAVIOR, INTERACTIONS,…

sloppy based on xkcd.com

MODELING LANGUAGES

Modeling languages are artificial languages with certain rules to express…

- … information & knowledge
- … structure, dependencies & workflow
- … *and more*

at an abstract (i.e., implementation-independent) level.

Used to provide multiple stakeholders such as

- … software guys (engineers, architects, analyzers & testers), and
- … non-software guys (users, customers, management,…)

insights into a (complex) system or a process without giving overwhelming (technical) details.

MODELING LANGUAGES (2)

Pros & Cons

Modeling languages help to…

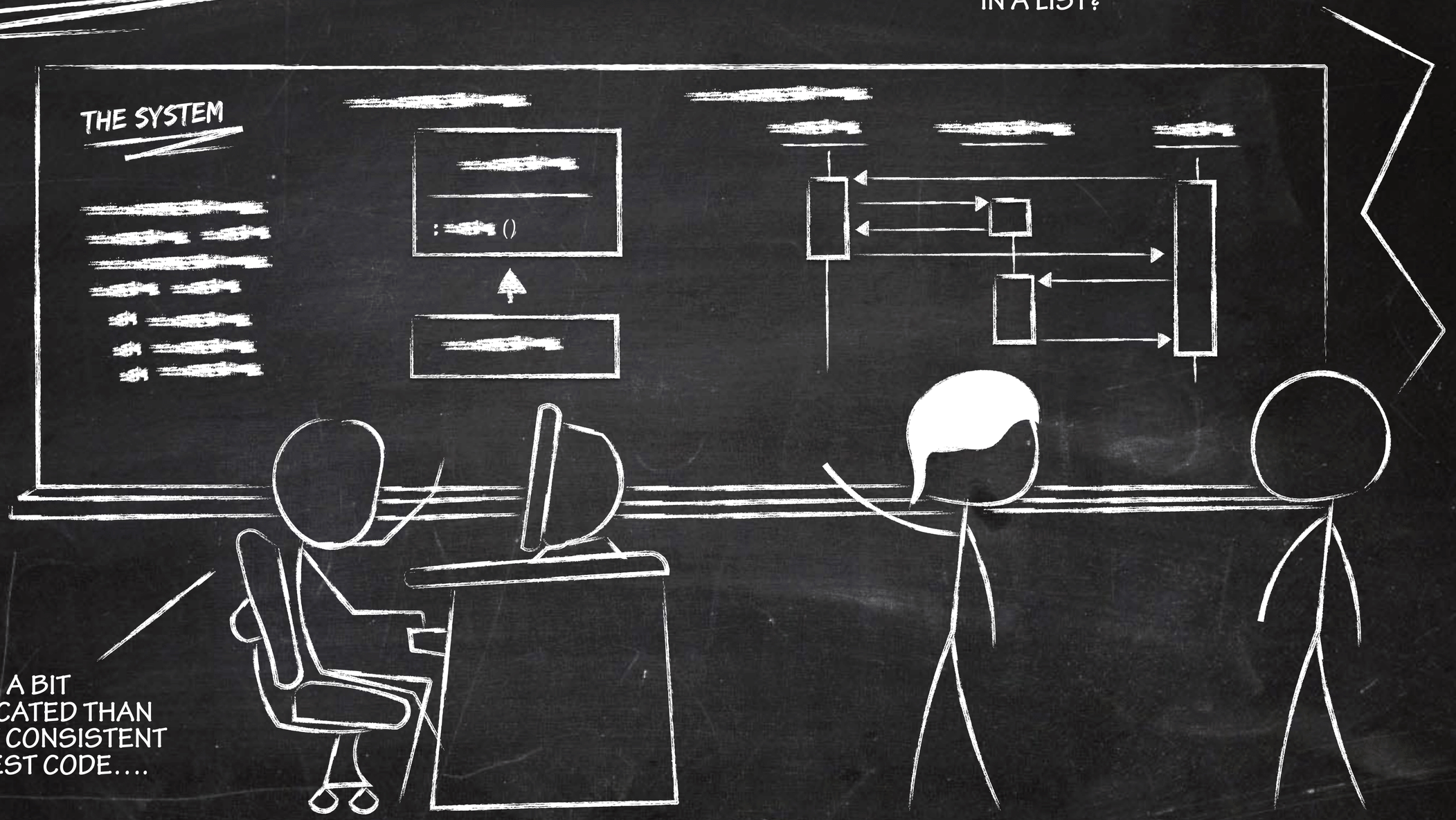
- PRO … avoid need for „reverse engineering & analyzing of code“
- PRO … getting you started in a complex system (i.e., better orientation)
- PRO … find dependencies & understand program flow at a glance
- PRO … communicate with other involved people on a common level

The downside is…

- CON maintenance costs (time/money): keeping model & code consistent is effort
- CON waste effort if wrong audience: hardliner engineers may not need models
- CON every (required) detail (even abstractly) lead to complex diagrams
- CON over-engineering: investing in complex design for simple problems

… also in practice, it's rarely the case that models satisfy 100% of the UML specification
… also in data-intensive system remember design & abstraction introduces (often) overhead that comes with performance penalties

Motivation (2)



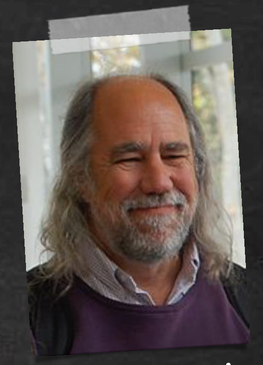
sloppy based on xkcd.com

MODELING LANGUAGES (3)

Important modeling languages:

UML | ER-MODEL | FLOW CHARTS | ...

Unified Modeling Language (UML)


Grady Booch


James Rumbaugh


Ivar Jacobson

UML is a kind of visual language rooted back to mid-1990's

- standardized 1997 (UML 1.0)
- de-facto industry standard
- Current version UML 2.5

Set of graphic notations for visual models on different aspects of modularized/object-oriented systems:

- Structural diagrams
  class, object, component, composite structure, package, deployment,...
- Behavior diagrams
  use-case, activity, state machine,...
- Interactions diagrams
  sequence, communication, timing, interaction overview,...

see all details: http://www.uml.org

UML Structural Diagrams (1)

Class Diagrams (1)

Purpose

Class diagrams:

- show (subset) of classes (or interfaces) incl. their members, and their relationships

classes: shared structure and shared behavior of certain objects (aka instances)

- type name (aka class name)
- type related functions (aka methods)
- contained variables (aka members)
- …

relationships:

- parent-child (aka inheritance)
- dependencies
- extension of methods
- …

Purposes

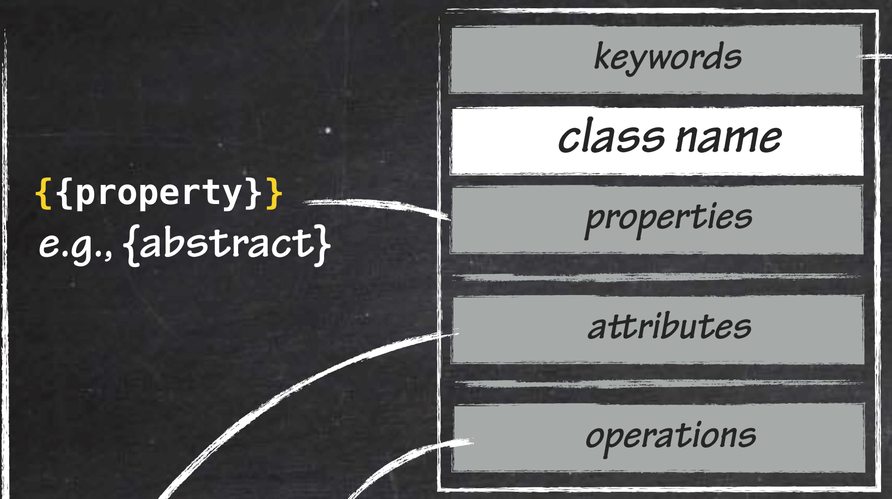
- Design of the system in a static view
- Showing responsibilities in the system
- Required for component and deployment diagrams

UML Structural Diagrams (1)

like blueprints for objects

Class Diagrams (2)

Notation and building blocks



<<{property,…}>>
e.g., <<interface>>

{{property}}
e.g., {abstract}

[visibility] name [({parameters})] [: result type]

[direction] name :type [multiplier] [= default value]

in, out, or inout

[visibility] [/] name [:type] [multiplier] [= initial value]

scoping: underlining represents „same for all instances“ (aka static)

+	public
-	private
#	protected
/	derived
~	package

Generalization

B has implicitly all properties of A

Association

A and B have relationship R

Part-whole relationships

Aggregation

A is part of B

Composition

A is part of B but A *cannot exists* without B

Visibility

Class

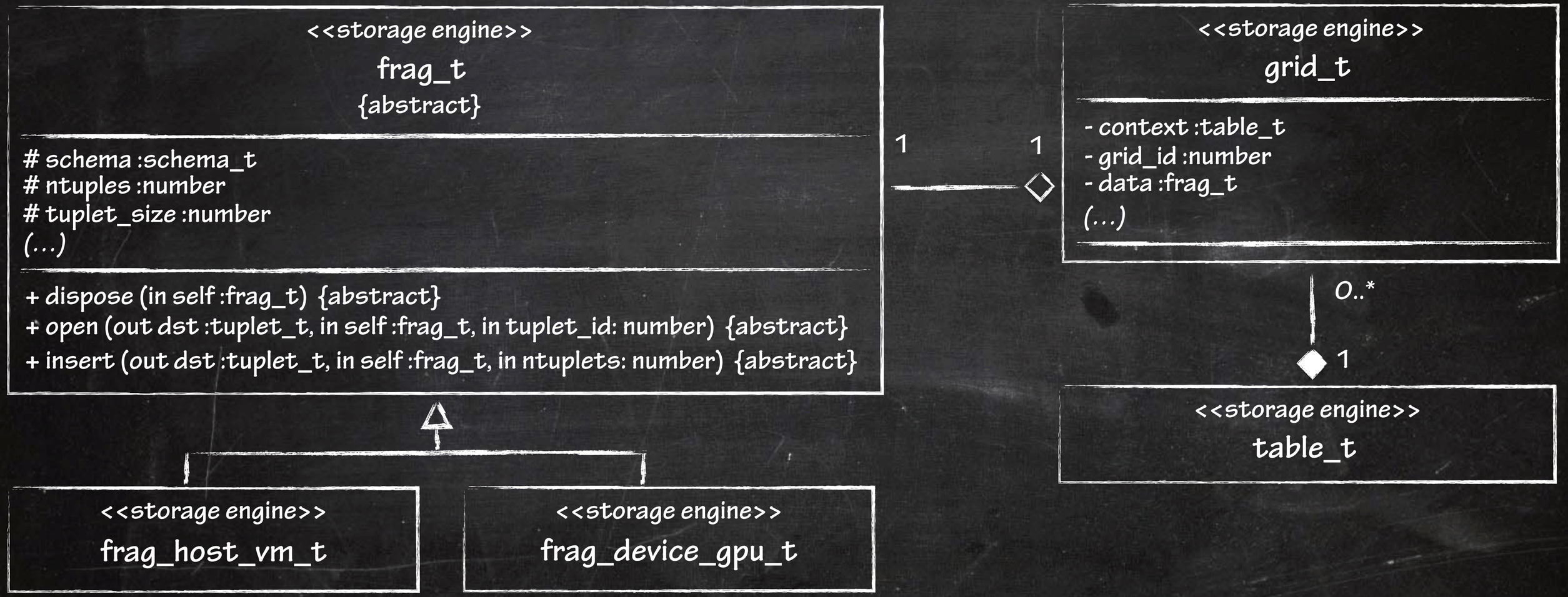
Relationships

Mandatory Optional [ ] Optional

UML Structural Diagrams (1)

Class Diagrams (3)

Example



UML Structural Diagrams (2)

Object Diagrams (1)

Purpose

Object diagrams:

- show a detailed snapshot of certain instances (incl. values of certain attributes) at a point in time.

it's like pausing your program and take a look into the variable value assignments at a particular time during runtime

Class diagram and object diagram are closely related

- class diagrams focus on classes
- object diagrams focus on instances of these classes (aka objects)

Purposes

- Shows objects in the system during runtime
- Static view on interaction of objects
- Object behavior and relationships at particular moment
- Prototyping a system (cf., fake/simulate data and environment)

cf. debugging

UML Structural Diagrams (2)

concrete things of classes

Object Diagrams (2)

Notation and building blocks

same as for class diagrams with the exception that

- it's about objects not classes

- lines („links“) connect objects

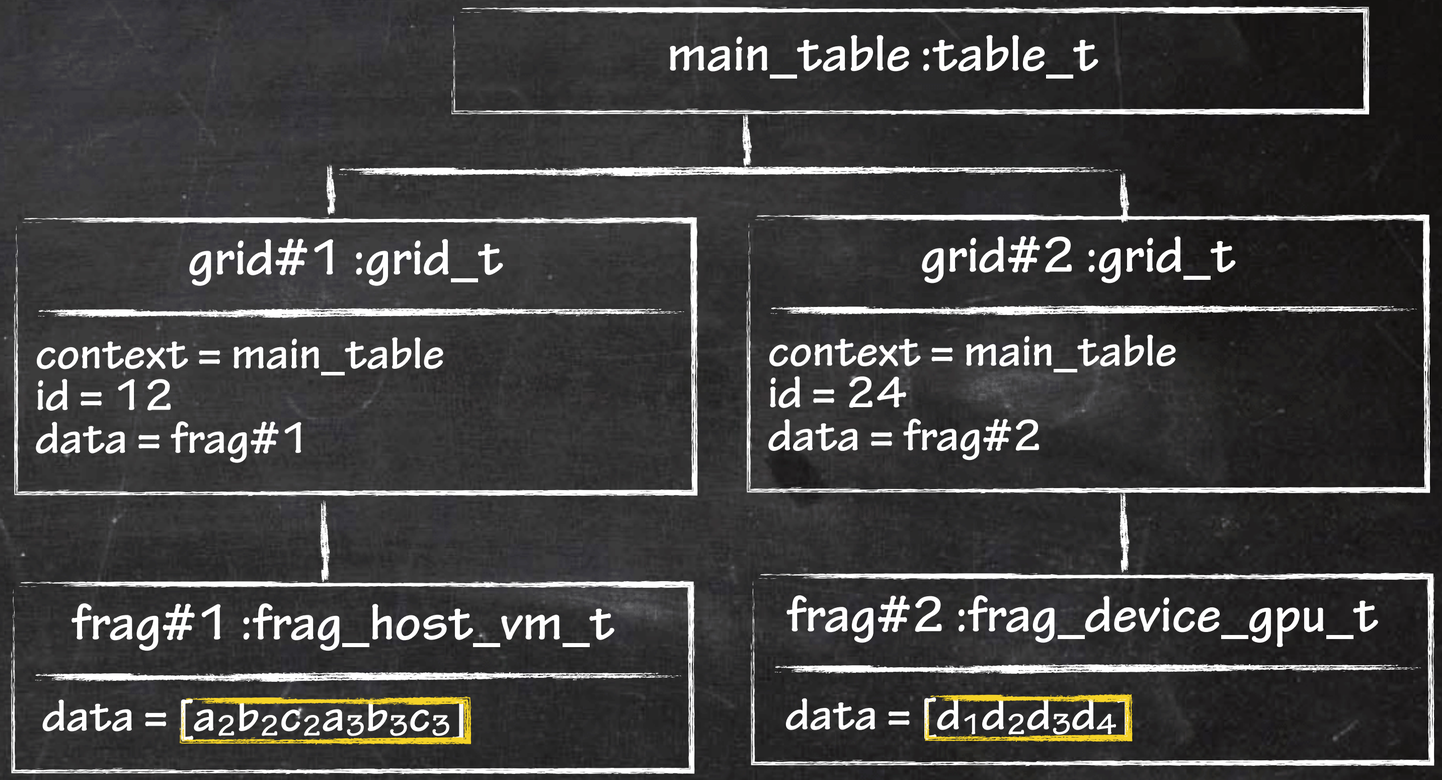
UML Structural Diagrams (2)

Object Diagrams (3)

Example

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2
a_3	b_3	c_3	d_3
a_4	b_4	c_4	d_4

(data view)



(object diagram)

UML Structural Diagrams (3)

Component Diagrams (1)

Purpose

Component diagrams:

- show organization, hierarchy and relationships of (logical) system components*.

larger design units that will be implemented using replaceable things

Purposes

- Visualize the systems logical components (w.r.t. its architecture)
- Communicate early overall view on the system (e.g., to key project stakeholders)

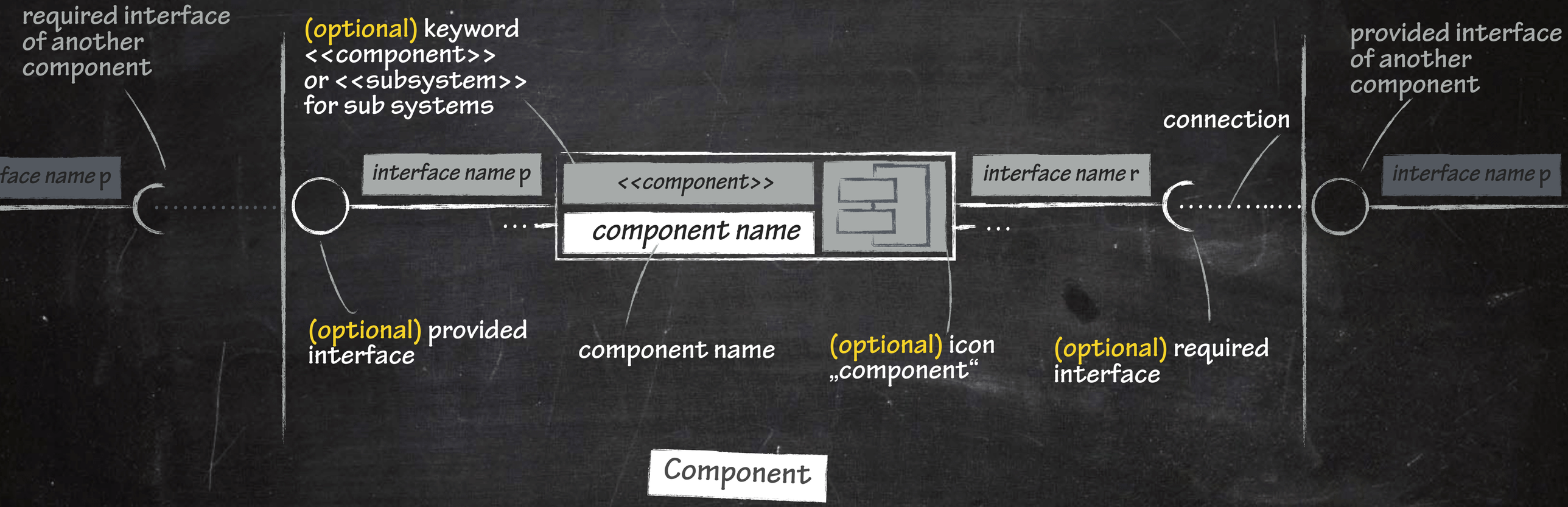
*since UML 2 the concept „component“ refers to autonomous, encapsulated units within a system that expose at least one interface

UML Structural Diagrams (3)

Component Diagrams (2)

Notation and building blocks

Since UML 2.0, the notation is similar to class diagrams (for better scaling)



Component

UML Structural Diagrams (3)

Component Diagrams (3)

Example