Entwurf von Softwarekomponenten

aus dem Schwerpunkt

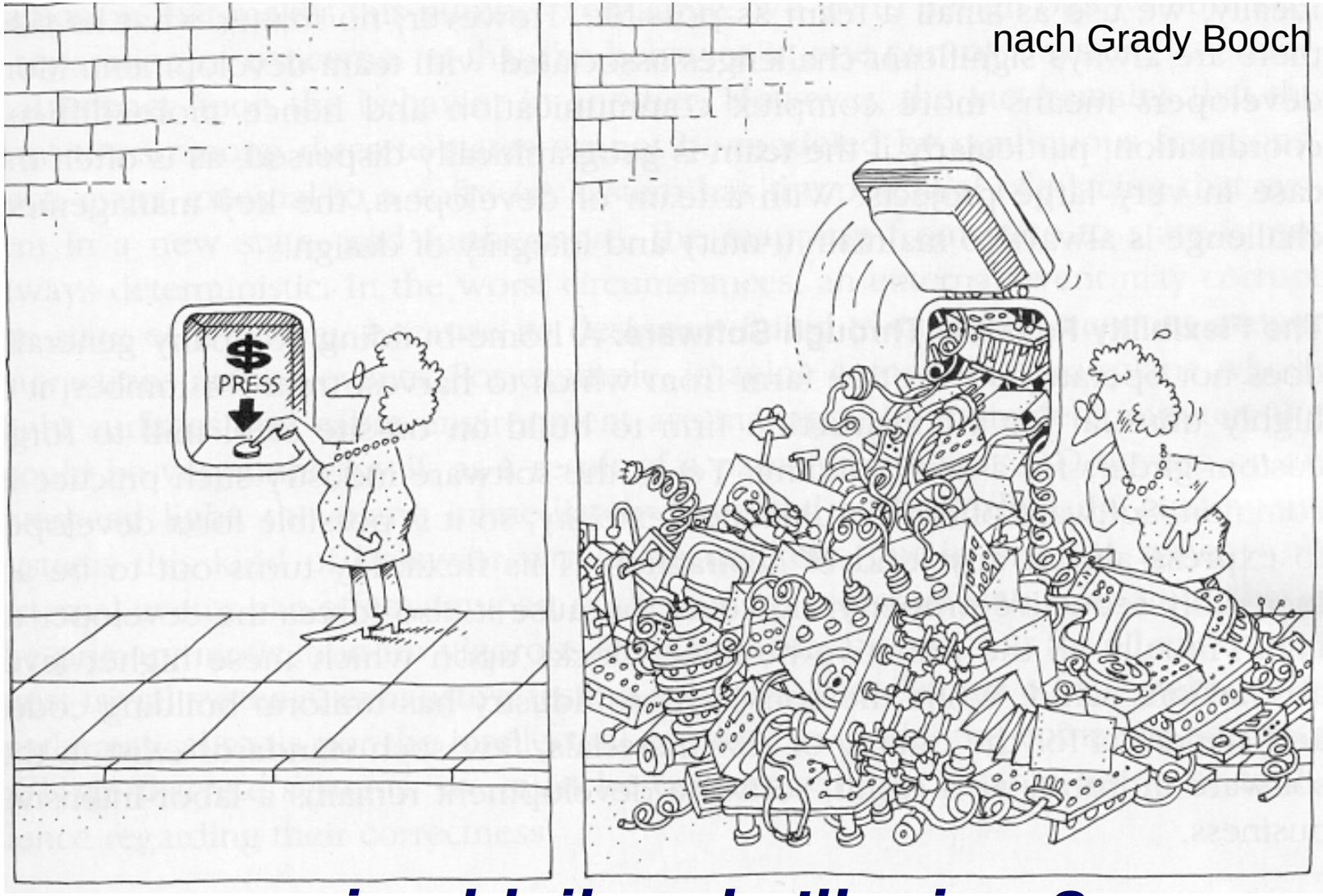
Modellbasierte Software-Entwicklung

Harald Böhme, Joachim Fischer
boehme/fischer@informatik.hu-berlin.de

Zum heutigen Ablauf

1. Motivation
2. Middleware, CORBA
3. Vom CORBA-Objektmodell zum CORBA-Komponentenmodell
4. Modellbasierte Komponentenentwicklung
5. evtl.: Beispiel
6. Zusammenfassung, Ausblick

Vereinfachung der Softwareentwicklung



– eine bleibende Illusion ?

Lang gehegter Traum der Softwareentwicklung

- **Wiederverwendung** von bereits Entwickeltem
- Zusammensetzung von Software aus vorgefertigten **Bausteinen**
- Konzentration auf das Wesentliche: die **Geschäftslogik**

Lang gehegter Traum der Softwareentwicklung

- **Wiederverwendung** von bereits Entwickeltem
- Zusammensetzung von Software aus vorgefertigten **Bausteinen**
- Konzentration auf das Wesentliche, die **Geschäftslogik**

ABER

- trotz 30 Jahre intensiver Bemühungen nur **partielle Fortschritte** (Produktivität, beherrschbare Komplexität)
- **Detailvielfalt** blieb

  Beispiele: Transaktionsmodelle, Persistenz
- entstehender **Programmcode** verschleiert die eigentliche Geschäftslogik

➔ **NEUER ANSATZ: komponentenbasierte Softwareentwicklung**
ein weiterer Schritt zur Traumverwirklichung

Motivation

- Entwickler, die die **Geschäftslogik** verstehen, sollen
  - vordefinierte Komponenten zusammensetzen können
  - ohne sich um programmiersprachliche Details zu kümmern

Motivation

- Entwickler, die die **Geschäftslogik** verstehen, sollen
  - vordefinierte Komponenten zusammensetzen können
  - ohne sich um programmiersprachliche Details zu kümmern

- aber **wer kümmert sich** um diese Details ?
  - Komponentenentwickler
  - Software-Systeme
    (Middleware mit spezifischer Komponentenunterstützung)

Motivation

- Entwickler, die die **Geschäftslogik** verstehen, sollen
  - vordefinierte Komponenten zusammensetzen können
  - ohne sich um programmiersprachliche Details zu kümmern

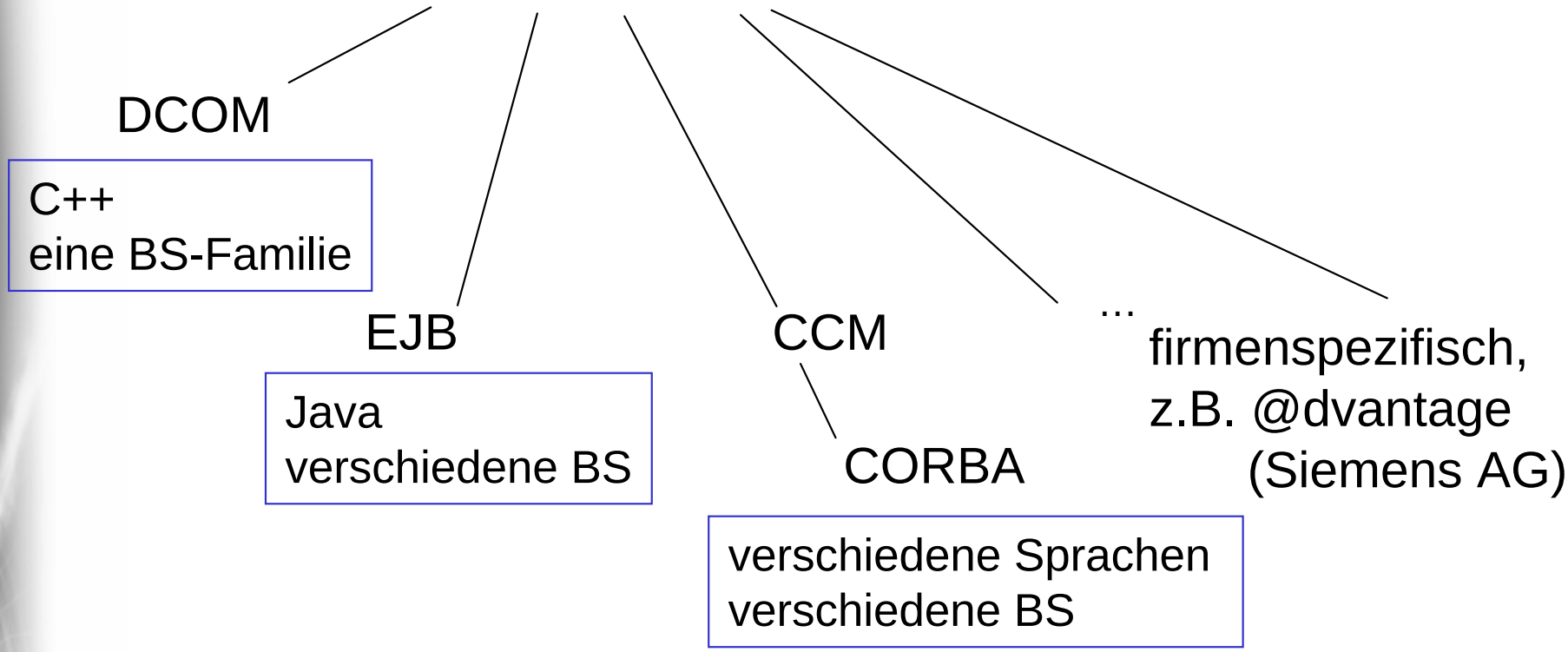
- aber **wer kümmert sich** um diese Details ?
  - Komponentenentwickler
  - Software-Systeme
    (Middleware mit spezifischer Komponentenunterstützung)

- Grundlage ist immer ein **Komponentenmodell**

  als Vereinbarung über
  - Komponentenaufbau,
  - Art von Services

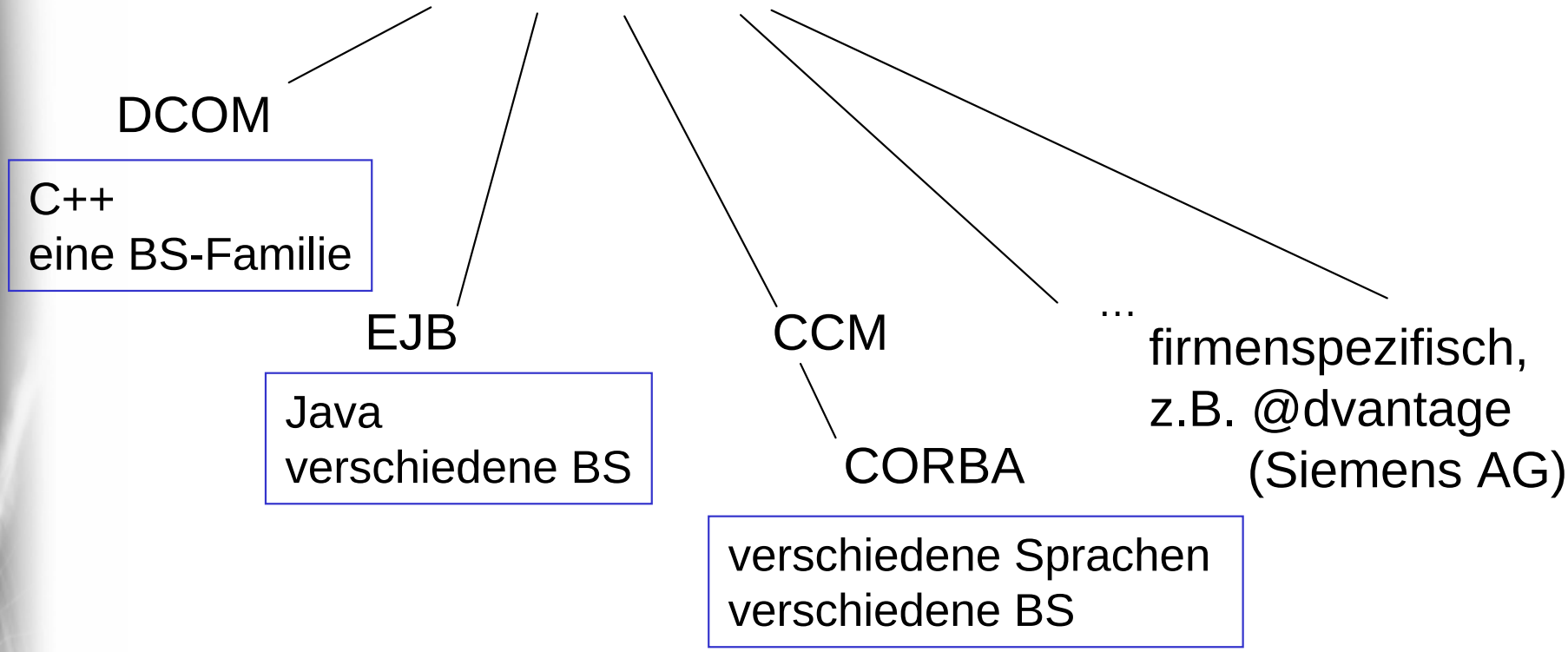
  ➔de facto- Standards (mit spezifischen Vor- und Nachteilen)

Komponentenplattformen



- substanzielle Unterschiede

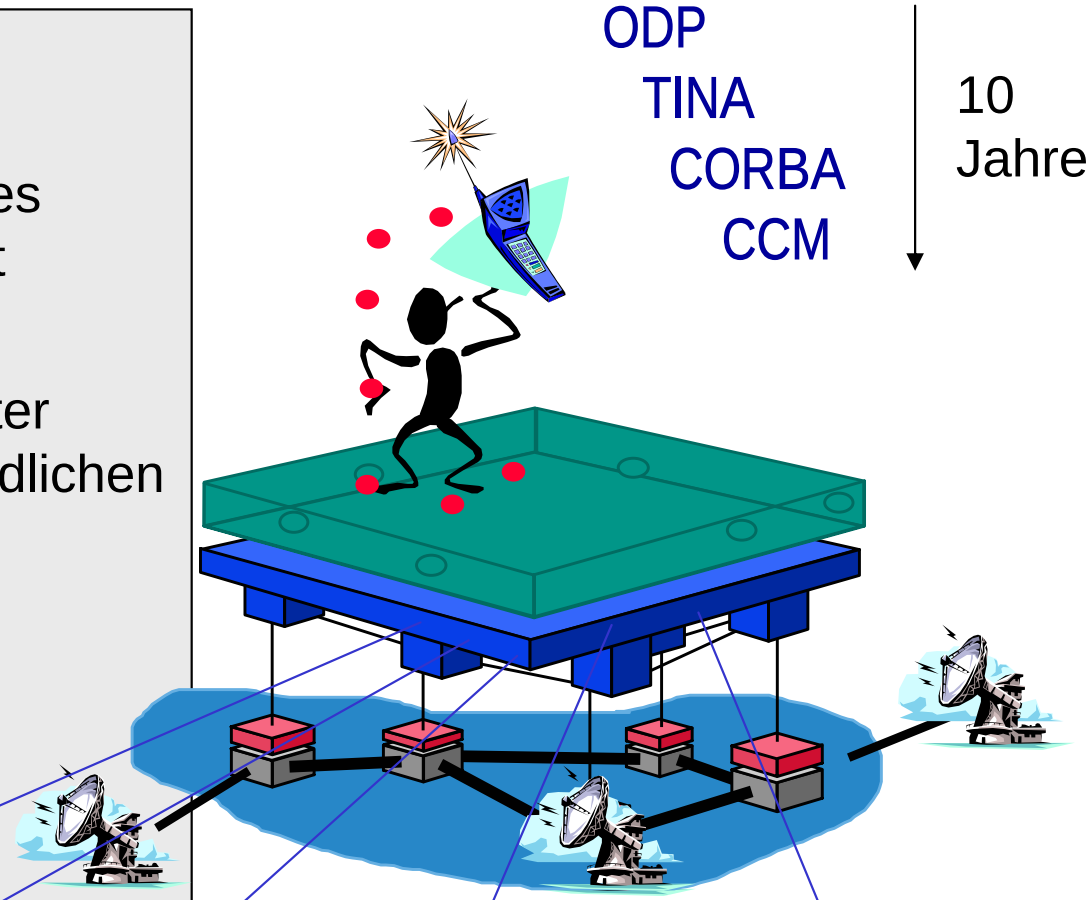
Komponentenplattformen



- substanzielle Unterschiede
- Entwicklung erfolgte in Verbindung

DOT: Distributed Object Technologies

- Objekte kooperieren über geeignete Infrastrukturen
- Kapselung verbirgt komplexes Verhalten, Interface-Konzept
- Interoperabilität
- Wiederverwendung kompletter Applikationen auf unterschiedlichen Betriebssystemplattformen
- Erweiterbarkeit
- Implementations-sprachunabhängigkeit
- ...



PLATINUM | CATS | @dvantage | COACH | CCM – OGSA ???

Auswahl von Projekten am Lehrstuhl für Systemanalyse

… weiterer Ablauf

1. Motivation
2. Middleware, CORBA
3. Vom CORBA-Objektmodell zum CORBA-Komponentenmodell
4. Modellbasierte Komponentenentwicklung
5. evtl.: Beispiel
6. Zusammenfassung, Ausblick

Was ist CORBA?

- CORBA = *Common Object Request Broker Architecture*

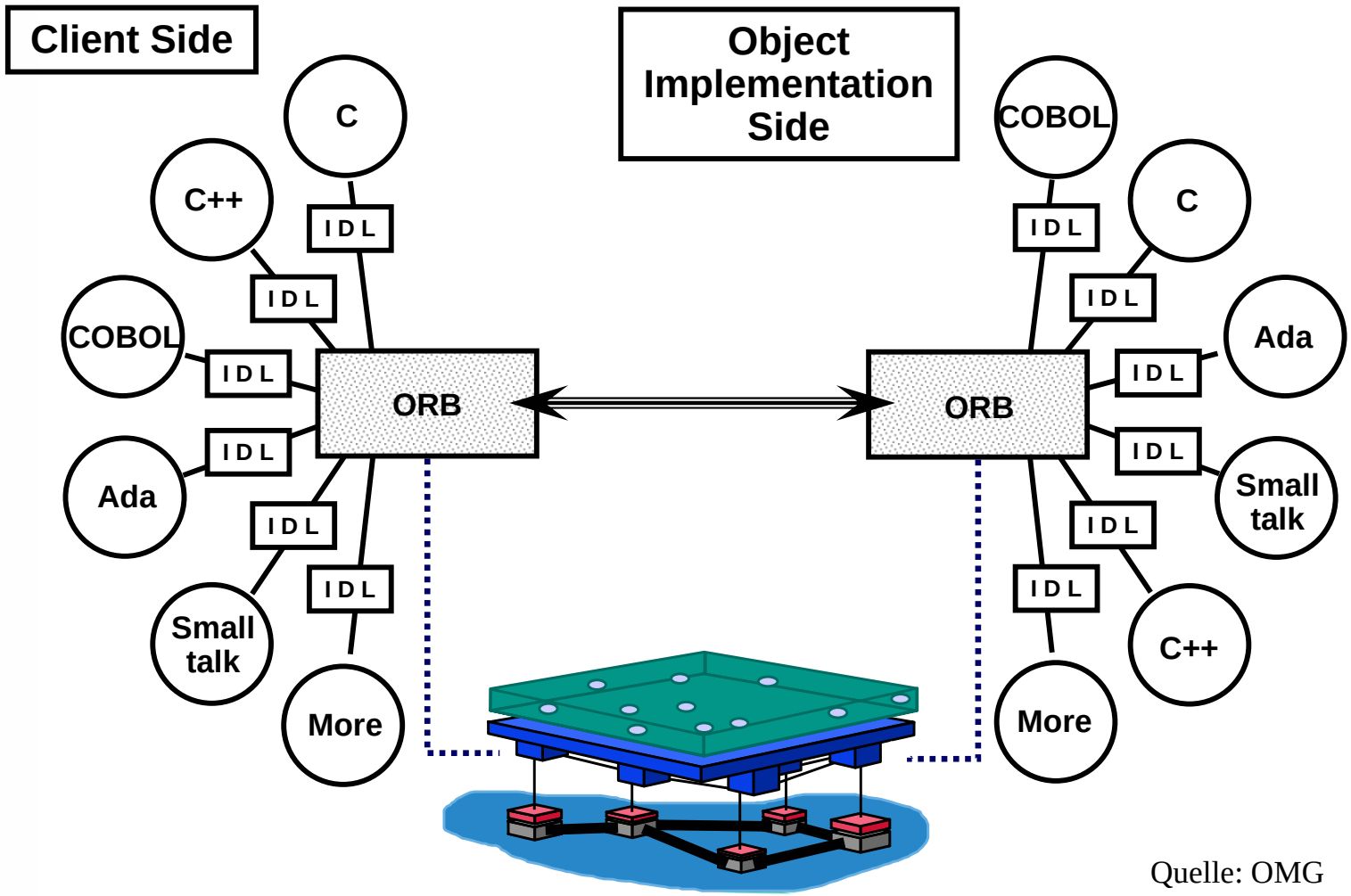
- ist eine **verteilte objekt-orientierte Client/Server- Plattform**
  - objekt-orientierter RPC (*Remote Procedure Call*)
  - Objekt-Dienste (*Naming, Trading, …*)
  - Unterstützung verschiedenster Programmiersprachen per Sprachabbildung
  - interoperable Protokolle
  - Programmierrichtlinien und Programmier-Muster

- ersetzt ad-hoc- Mechanismen (wie Socket- Kommunikation) durch
  eine offene, standardisierte, skalierbare und portable Plattform

Client/Server- Systeme

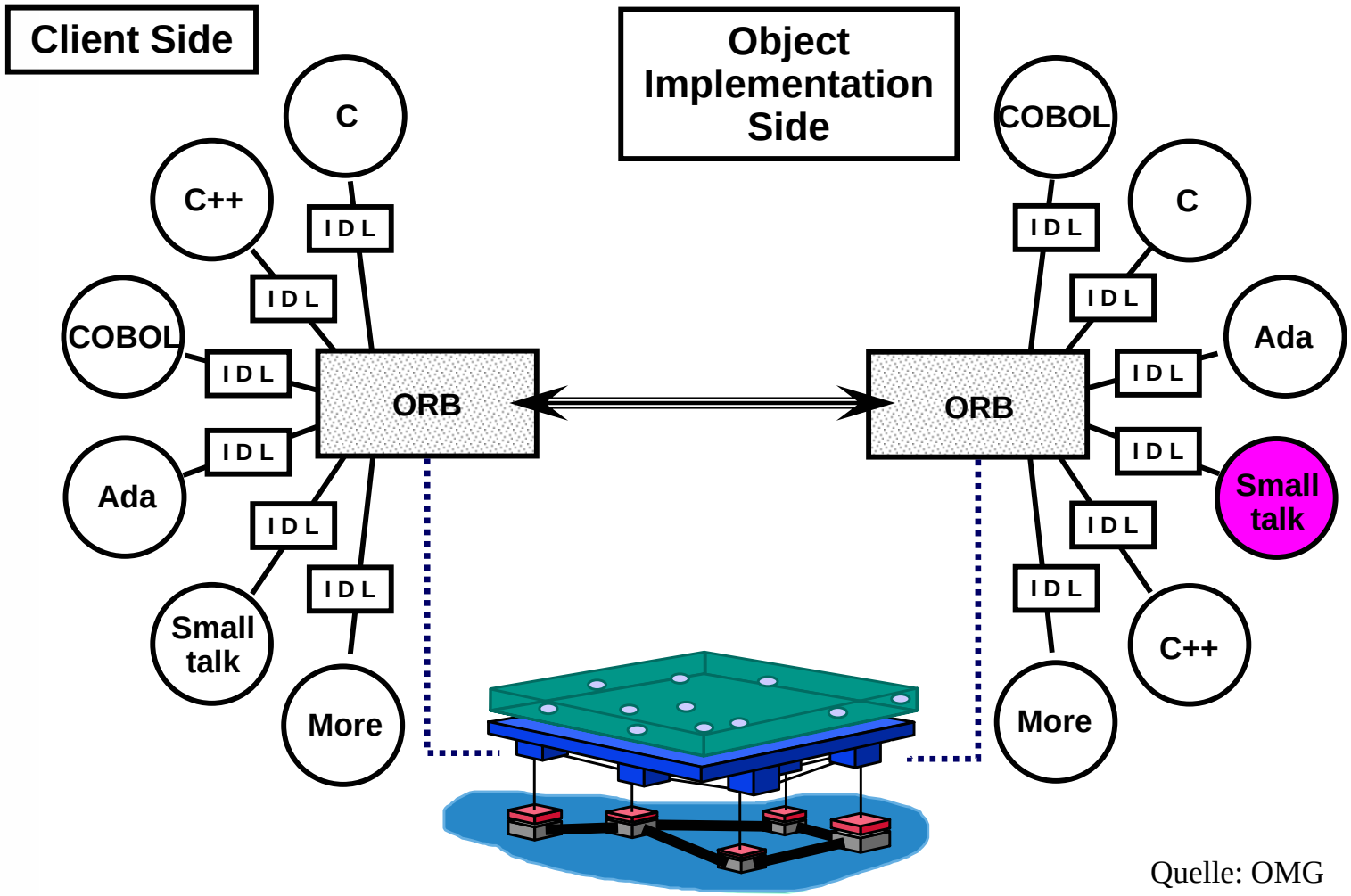
- eine gewisse Anzahl von Klienten und Diensterbringern kooperieren, um gemeinsam eine Aufgabe zu bearbeiten
- Diensterbringer sind passiv, bieten Dienst(e) an, warten auf Anforderungen der Klienten zur Diensterbringung
- Klienten sind aktiv, nehmen Dienstleistung in Anspruch
- Klienten und Server laufen als Prozesse i. allg. auf verschiedenen Maschinen
- objekt-orientierte *Client/Server*- Systeme nutzen OO-Paradigmen
  (Interface, Nachrichten, Vererbung, Polymorphie)

Interface Definition Language



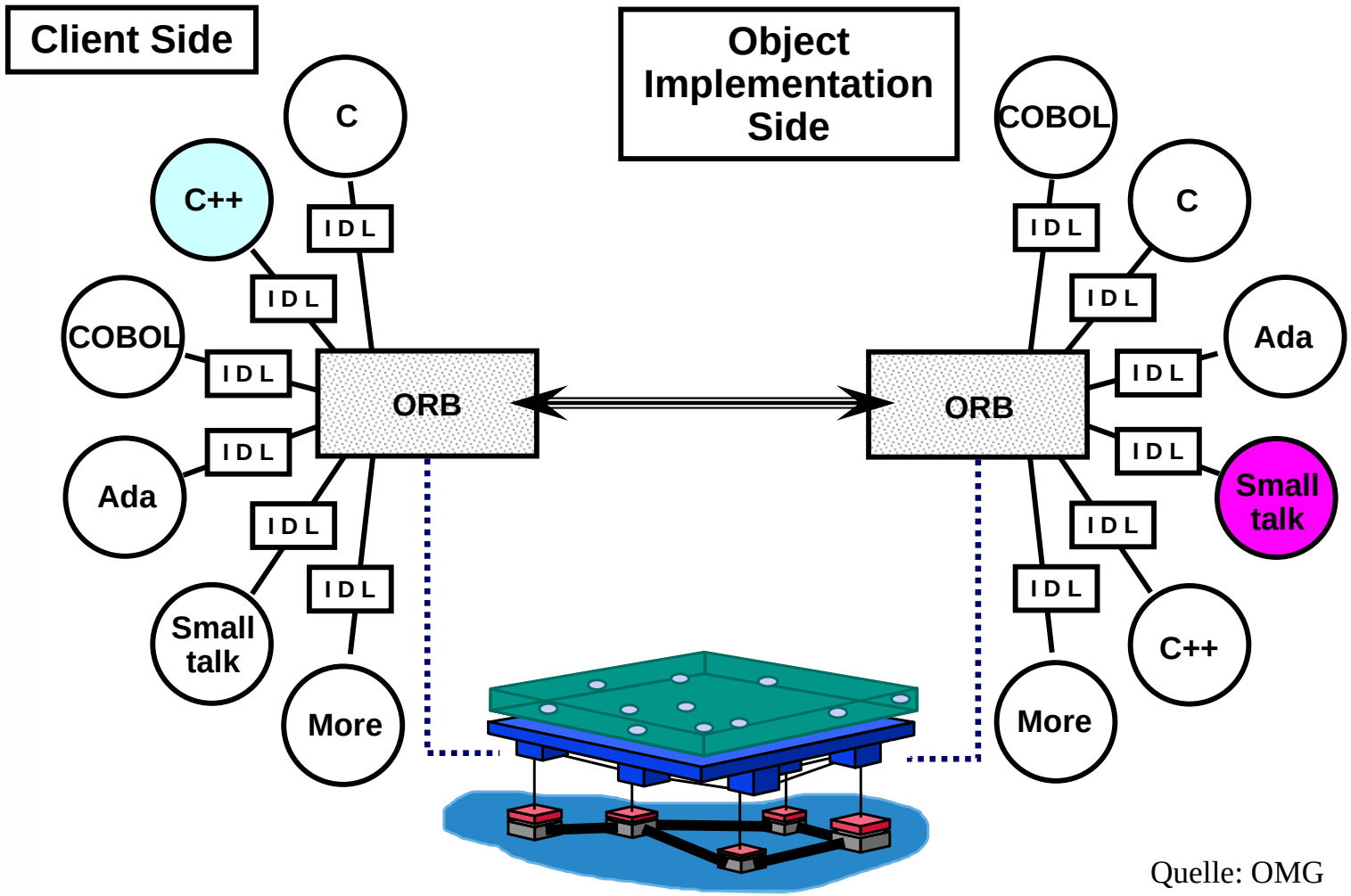
Quelle: OMG

Interface Definition Language



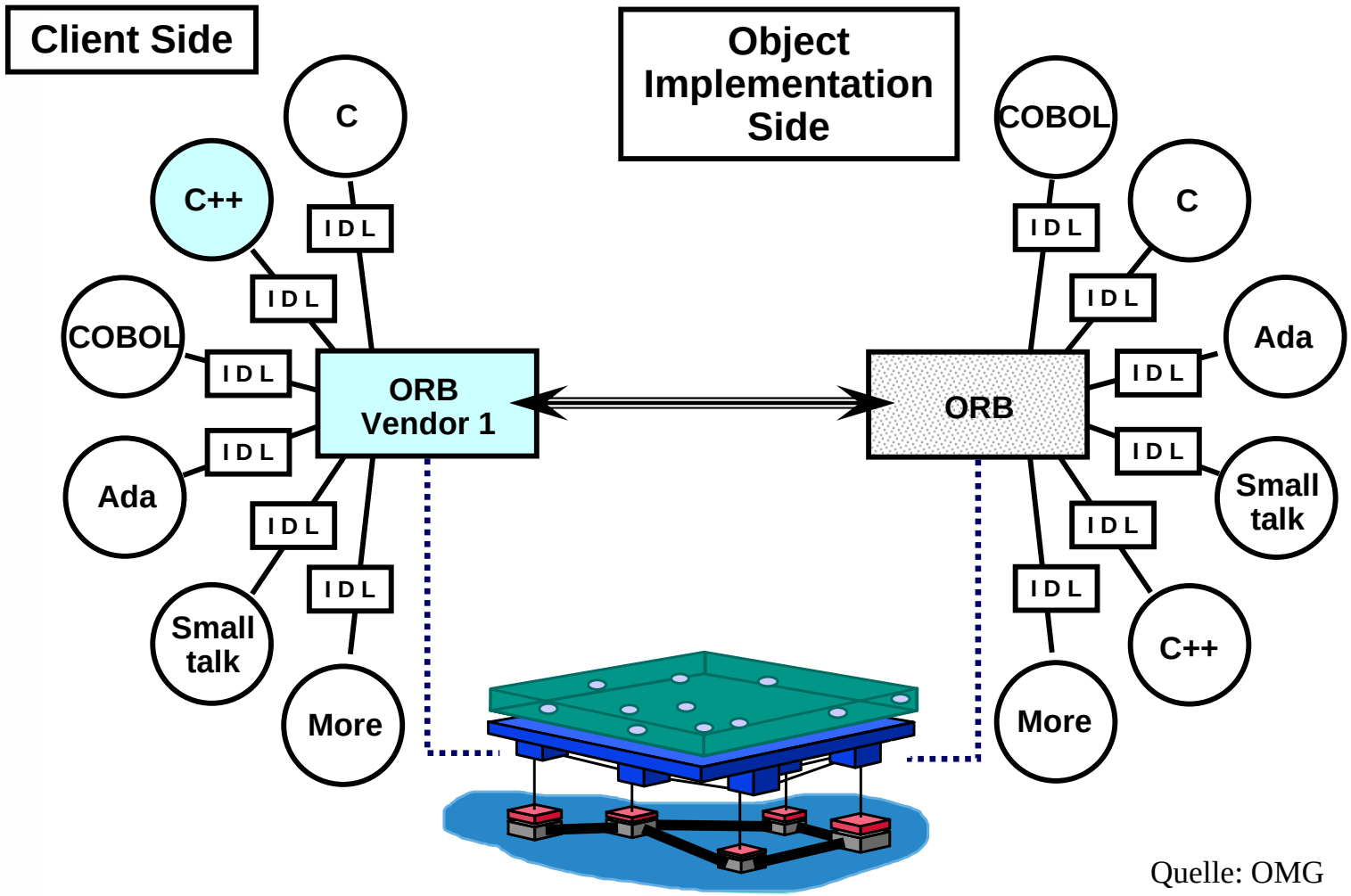
Quelle: OMG

Interface Definition Language



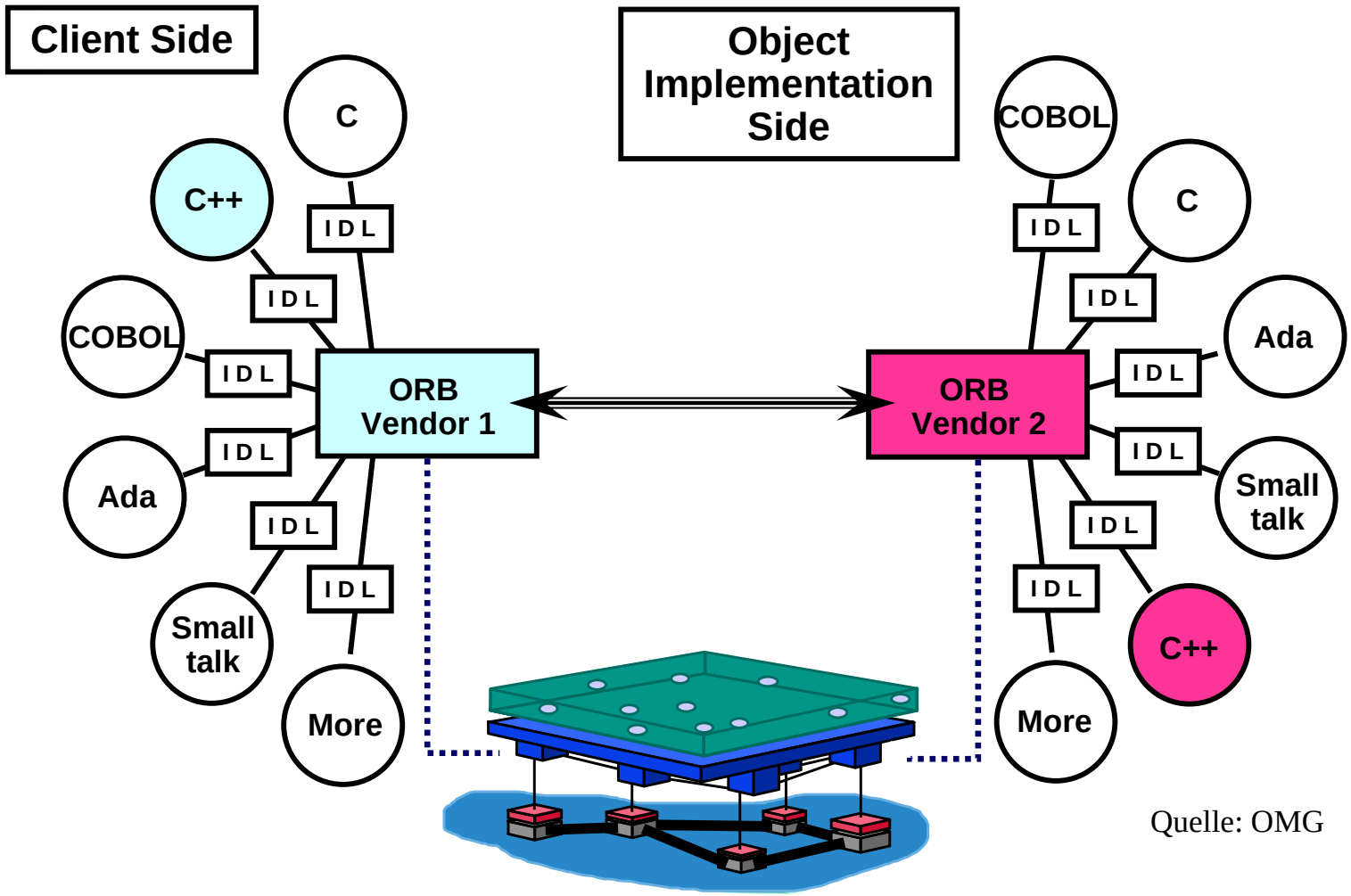
Quelle: OMG

Interface Definition Language



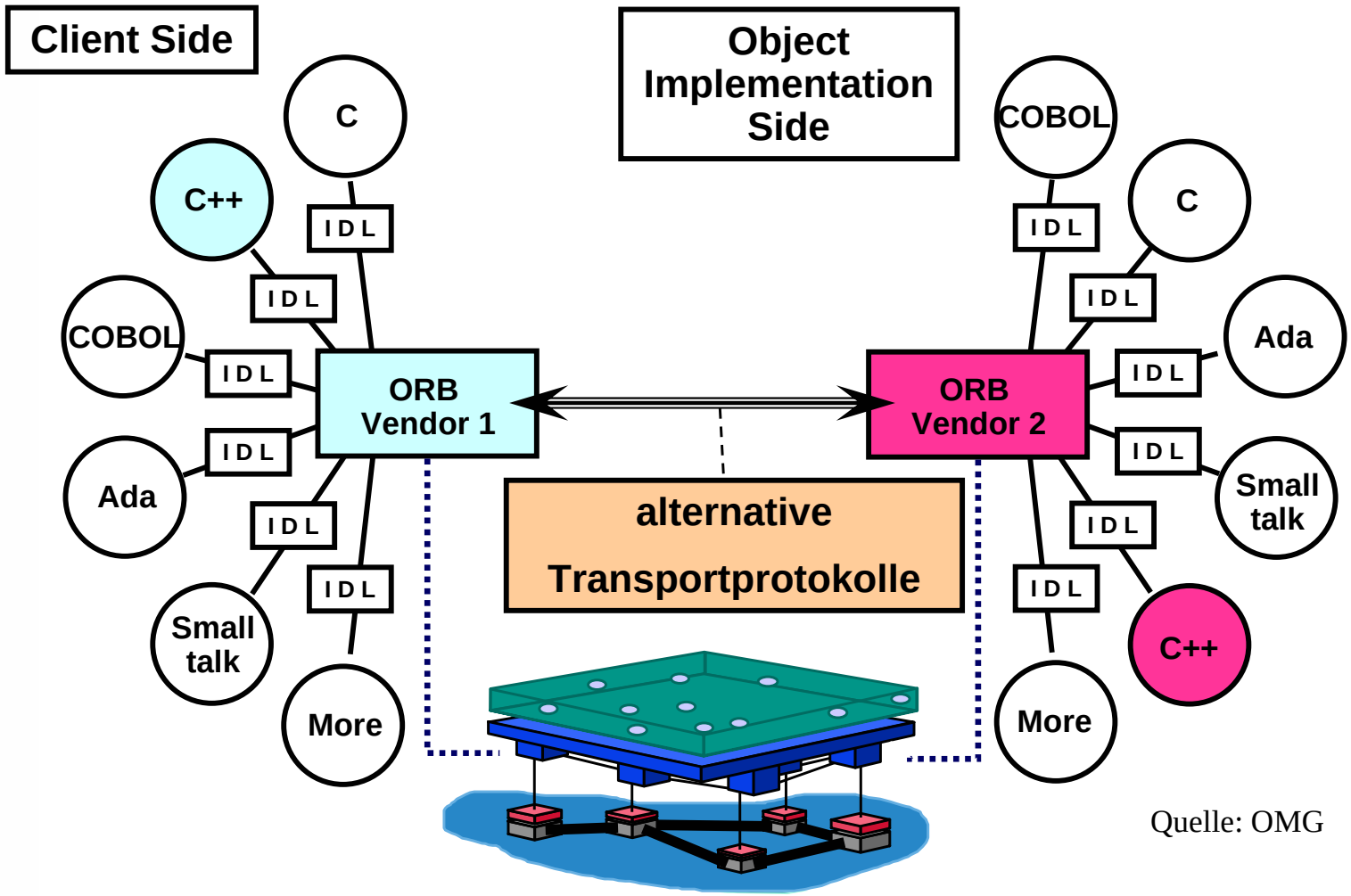
Quelle: OMG

Interface Definition Language



Quelle: OMG

Interface Definition Language



Quelle: OMG

Nützliche CORBA-Dienste

Lifecycle Service - Operationen für die Konstruktion, Destruktion, Kopieren und Verschieben von Objekten

Naming Service - Finden registrierter Objekte

Event Service - Nutzung von Kanälen zum asynchronen Nachrichtenaustausch

Object Property Service – Operationen, um Name-Werte-Paare mit beliebigen CORBA-Objekten zu verknüpfen

Object Transaction Service – Zwei-Phasen-Commit-Protokoll für zurücksetzbare CORBA-Objekte (flache u. verschachtelte Transaktionen)

Trading Service - „Gelbe Seiten“-Dienst

Weitere CORBA-Dienste

Licensing Service – Abrechnung der Benutzung von Corba-Objekten (noch nicht in Aktion)

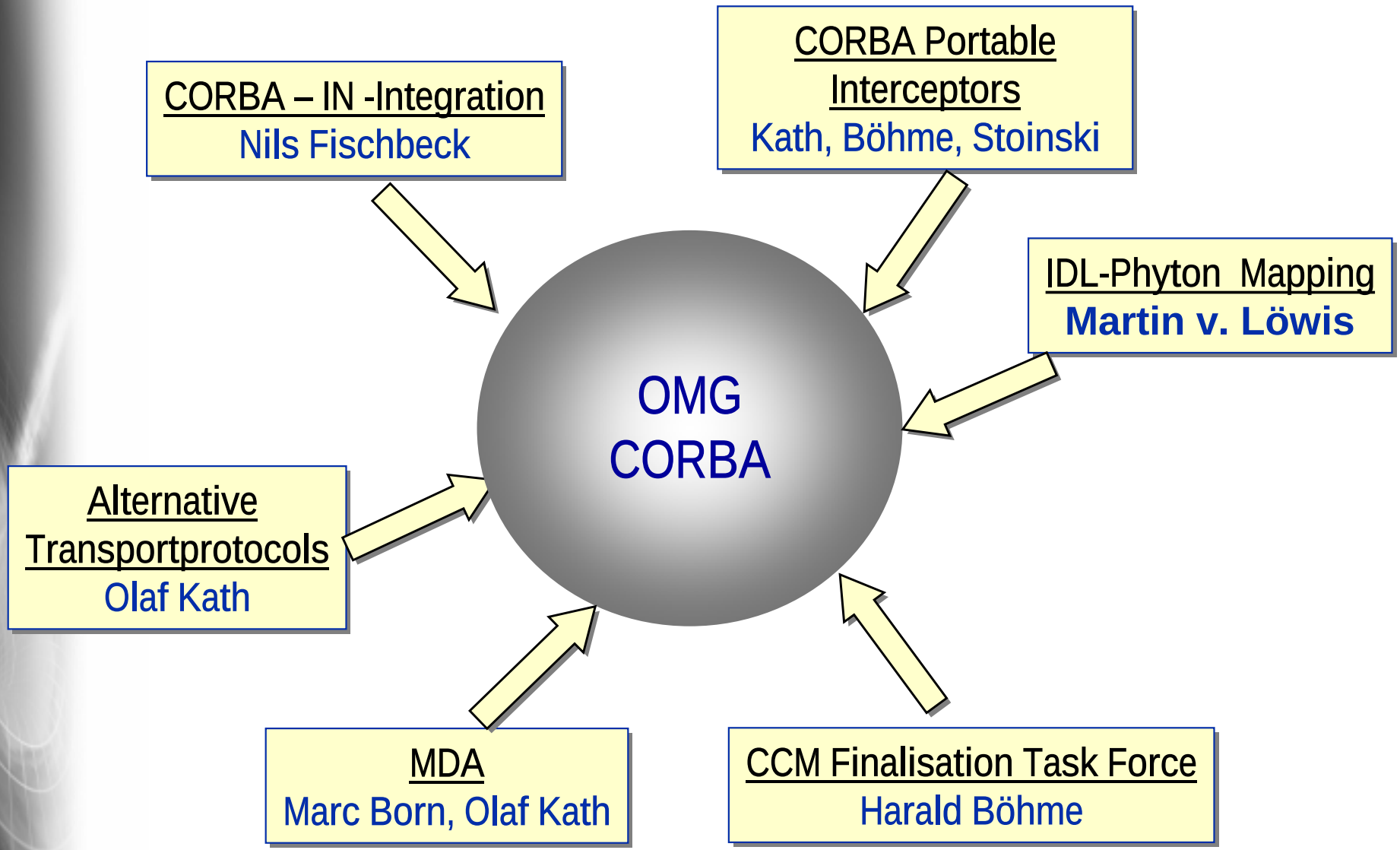
Security Service – einheitlicher Sicherheitsmechanismus mit Authentifikation, Autorisierung, Vertraulichkeit, Datenintegrität (nicht vollständig implementierbar)

Object Query Service - Abfragesprache für CORBA-Objekte á la SQL und OQL (nicht genutzt)

Object Collection Service – Bildung von Objekt-Strukturen

Object Startup Service – unterstützt den Object Transaction Service, Hochfahren eines Servers nach Absturz

Beiträge des Lehrstuhls



… weiterer Ablauf

1. Motivation
2. Middleware, CORBA
3. Vom CORBA-Objektmodell zum CORBA-Komponentenmodell
4. Modellbasierte Komponentenentwicklung
5. evtl.: Beispiel
6. Zusammenfassung, Ausblick

Von CORBA 2.x ...

- CORBA liefert verteiltes objekt-orientiertes Modell
  - Heterogenität durch IDL
  - Portabel durch standardisierte Sprachabbildungen
  - Interoperabel durch GIOP/IIOP
- **keine** standardisierten Packaging- und Deployment-Möglichkeiten
- **notwendige** explizite Programmierung nicht-funktionaler Aspekte
  - Lifecycle, Transaktionen, Persistenz, Sicherheit (aber Services sind da)

... zum CORBA Component Model (CCM)

- CCM liefert verteiltes komponenten-orientiertes Modell
  - Architektur zur Definition von Komponenten und ihrer Interaktionen
  - Paketierungstechnologie zur Unterstützung des Deployment-Prozesses
  - Container-Framework zur Integration nicht-funktionaler Aspekte wie Lifecycle, Transaktionen, Persistenz, ...
    mit unterschiedlichen Automatisierungsgrad
    (voll, I/f zu Diensten)
- CCM ist damit der erste betriebssystem- und sprachunabhängige Komponentenstandard

Komponentenbegriff

Allgemein Software-Komponente

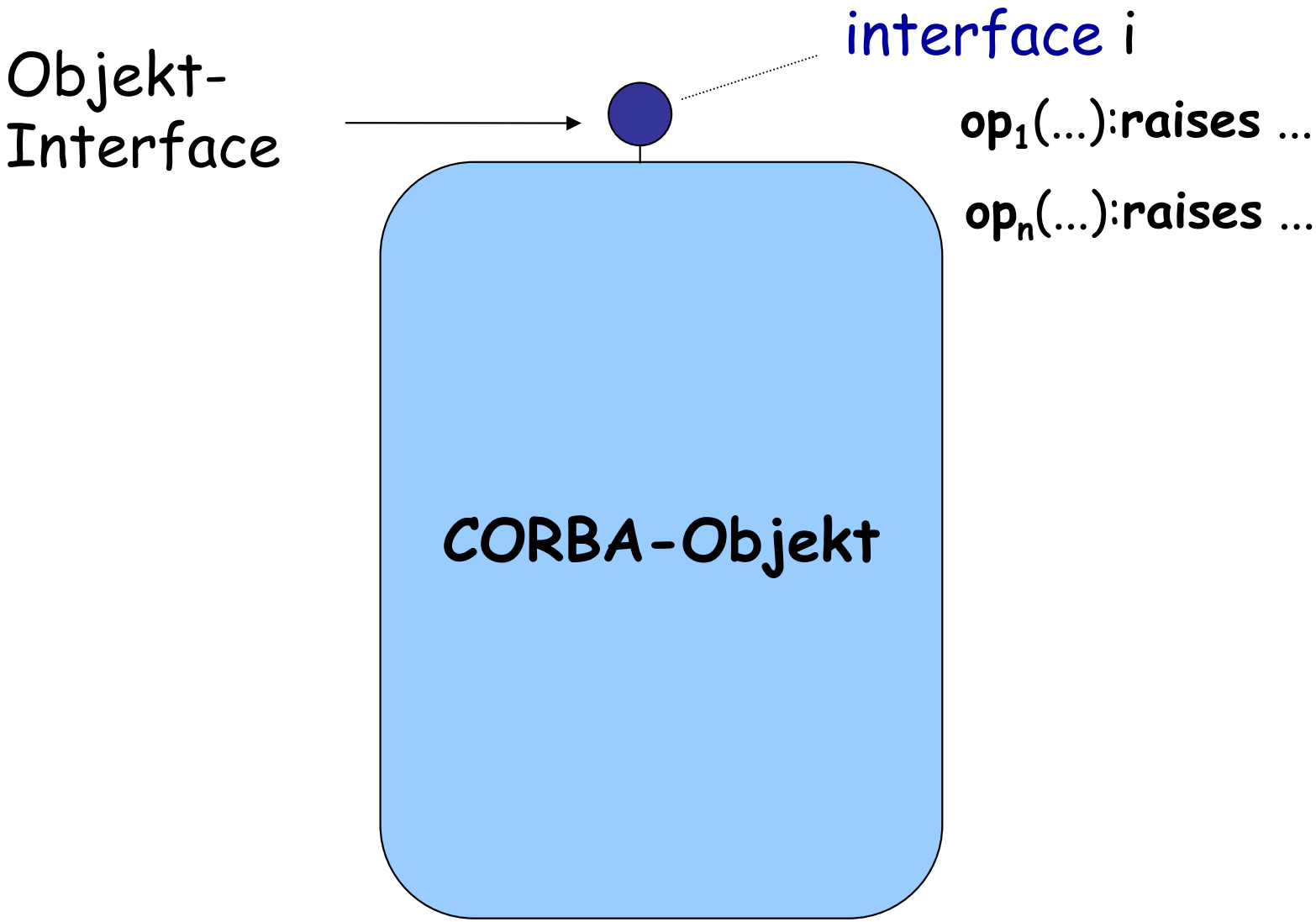
- modulare, in ihrer Umgebung austauschbare Einheit mit wohldefinierten Schnittstellen
- implementiert durch Software-Artefakte
- Erklärung benutzter und bereitgestellter Schnittstellen
- Stück Software in binärer Form (Java Beans, CORBA Components, …)

➔ **stets für die Verwendung innerhalb eines konkreten Komponentenmodells ausgelegt**

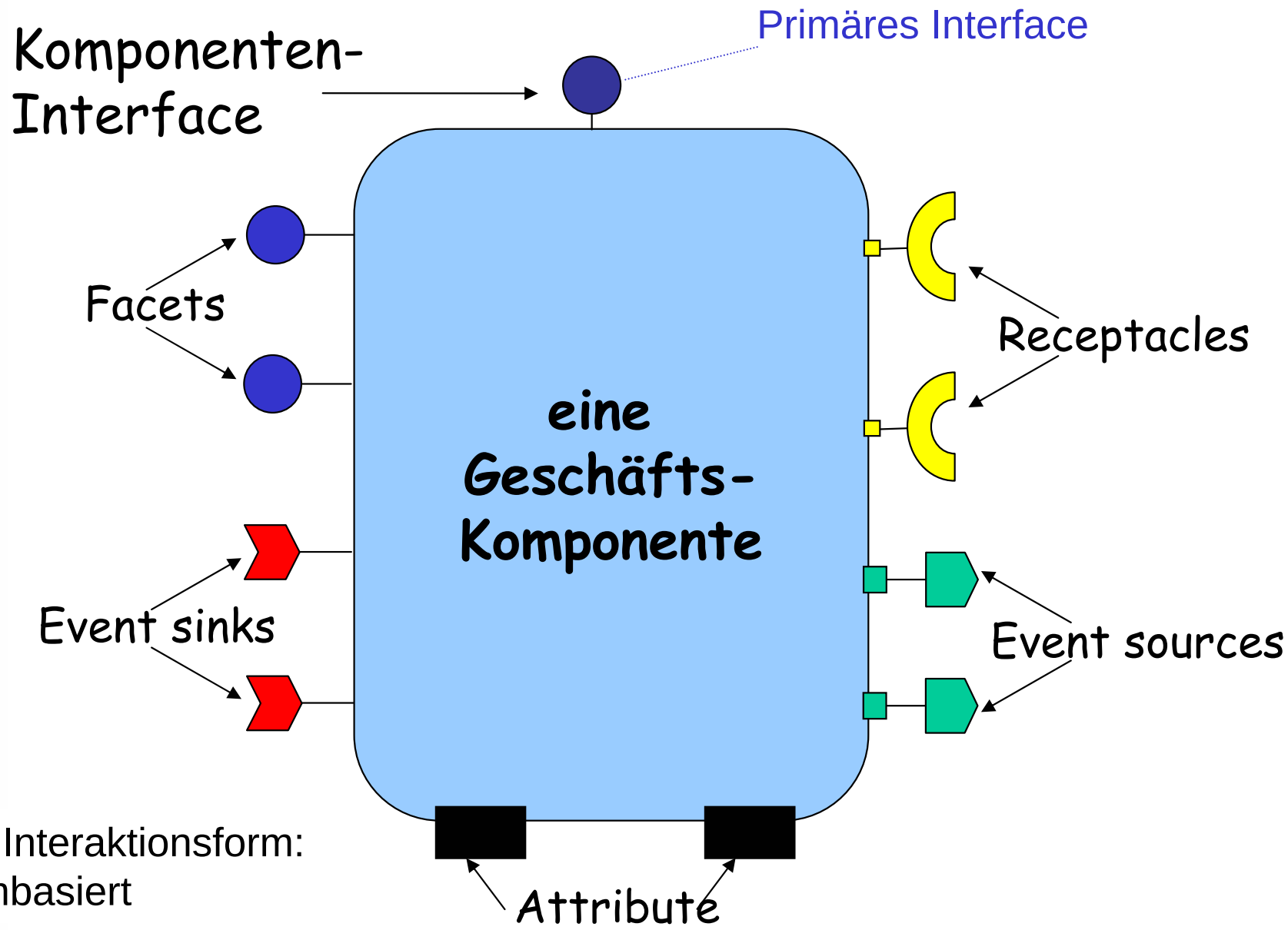
CCM-Komponente

- Software-Komponente
- Kommunikation über CCM/CORBA

CORBA-Objekt – CORBA-Komponente

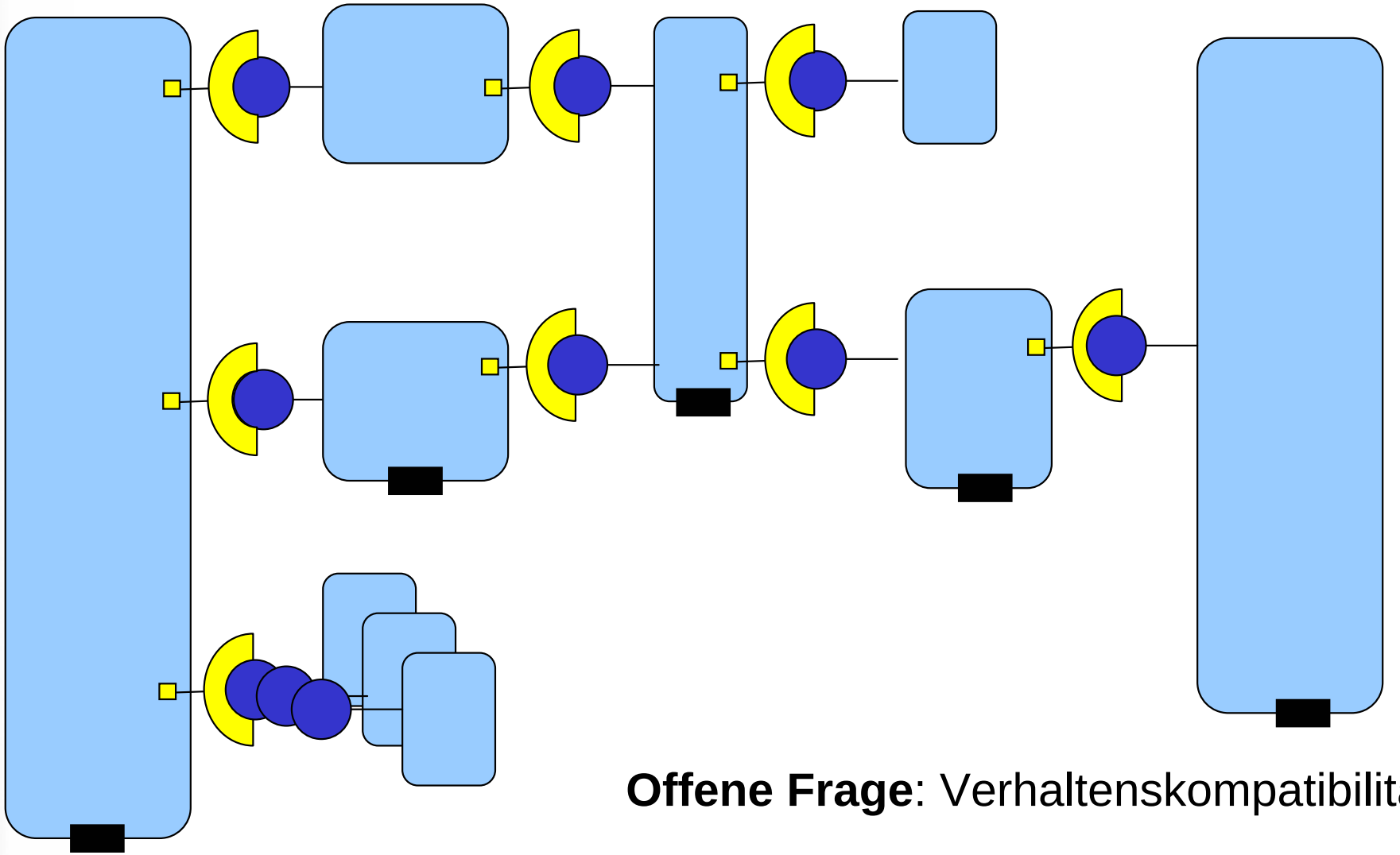


CORBA-Objekt – CORBA-Komponente



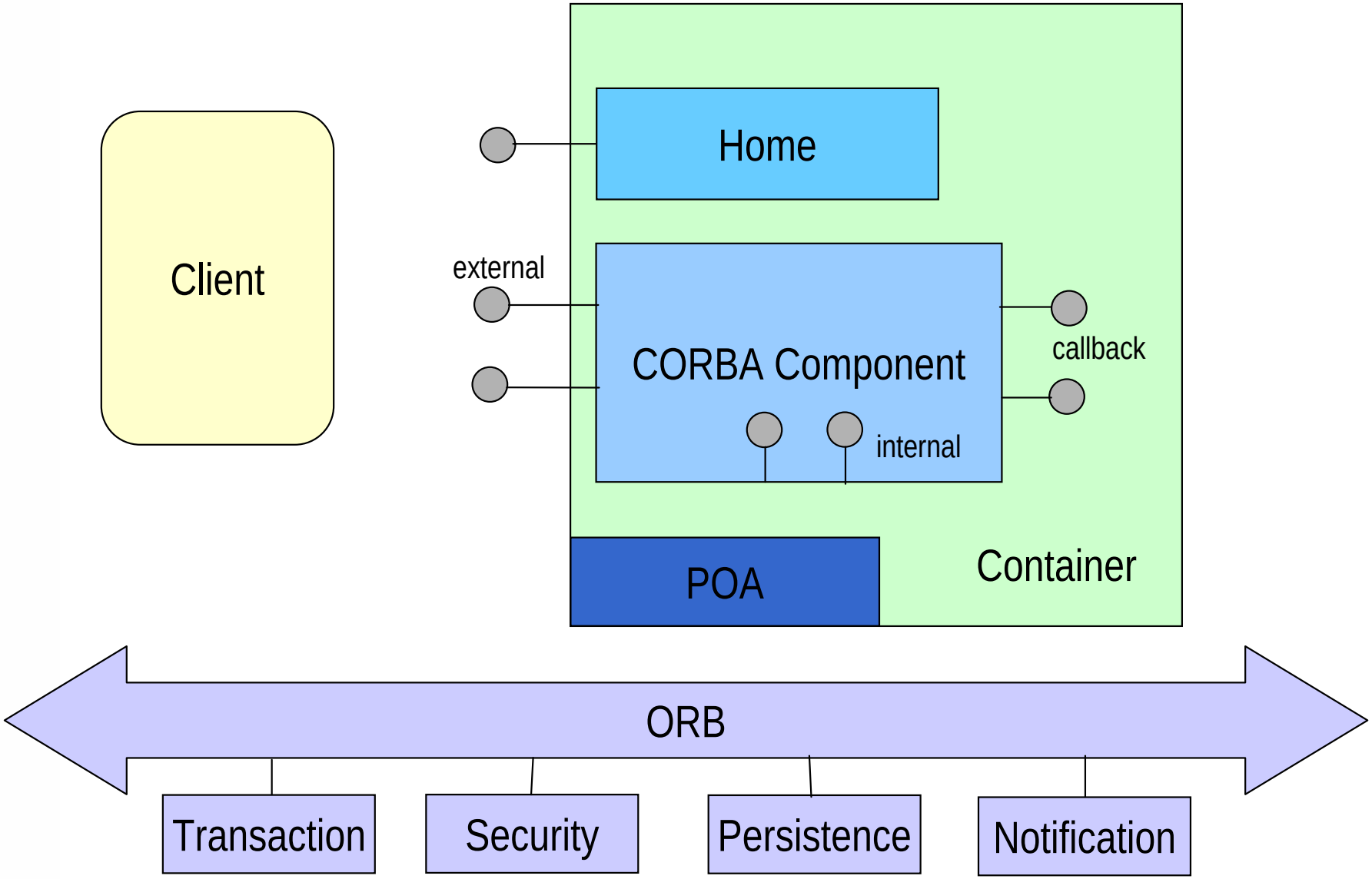
weitere Interaktionsform:
- streambasiert

Software-System als Komposition von Komponenten



Offene Frage: Verhaltenskompatibilität

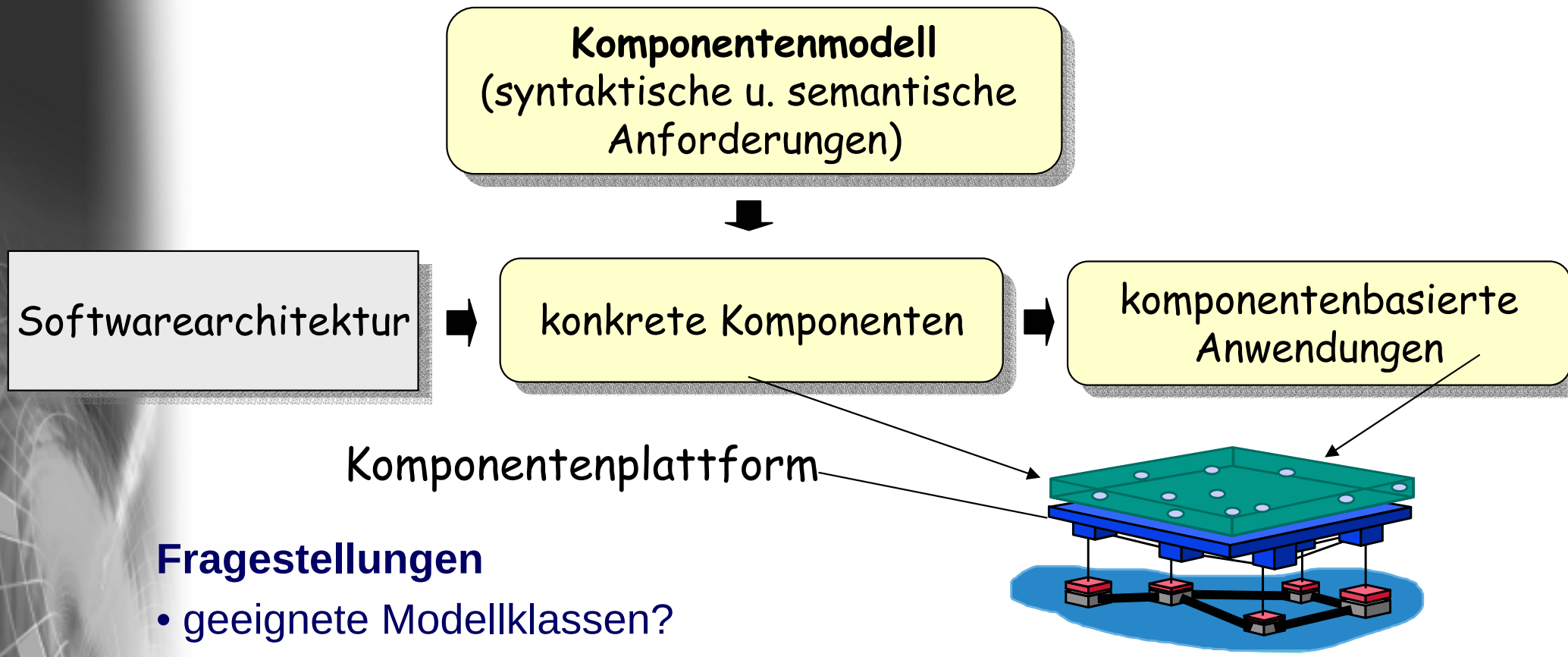
Container - Architektur



… weiterer Ablauf

1. Motivation
2. Middleware, CORBA
3. Vom CORBA-Objektmodell zum CORBA-Komponentenmodell
4. Modellbasierte Komponentenentwicklung
5. evtl.: Beispiel
6. Zusammenfassung, Ausblick

Modellbasierte Software- Komponentenentwicklung

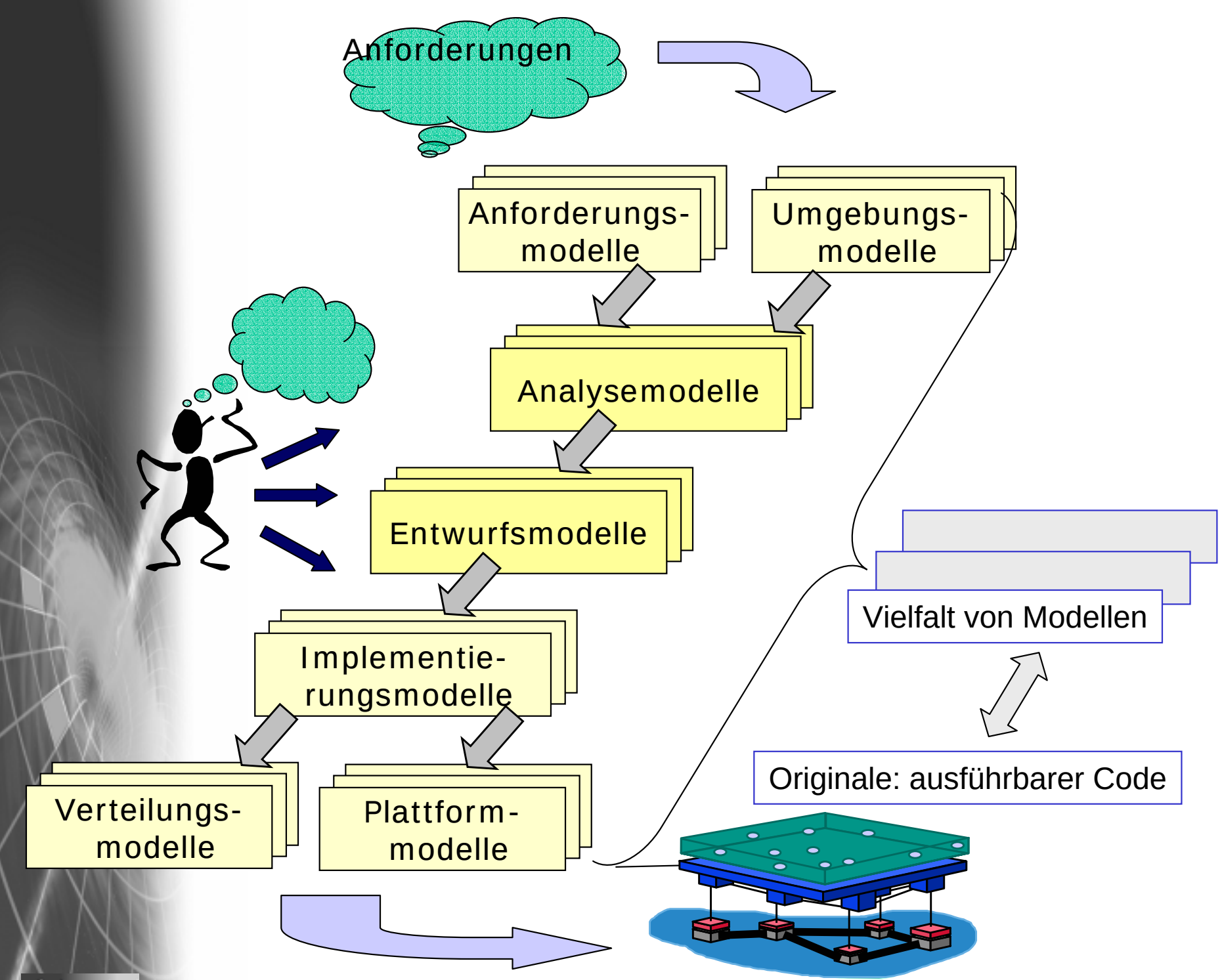


Fragestellungen

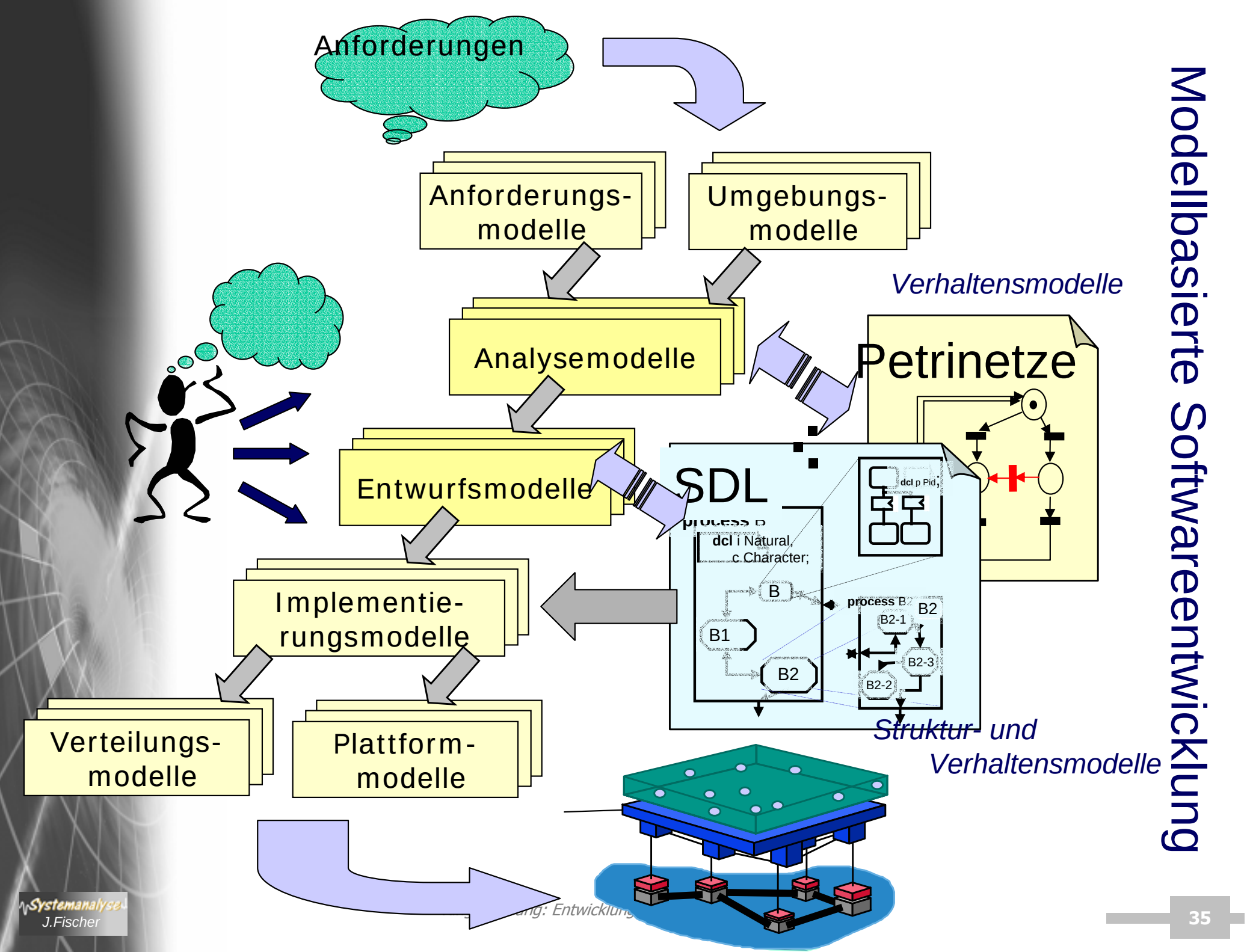
- geeignete Modellklassen?
- geeignete Werkzeugunterstützung: automatisierte Entwicklung?
- Relationserhaltung zwischen Modellen und Zielsoftwarekomponenten?
- Unabhängigkeit von Plattform und Komponentenmodell?
- Suche nach einem plattform-unhabhängigen Komponentenmodell?

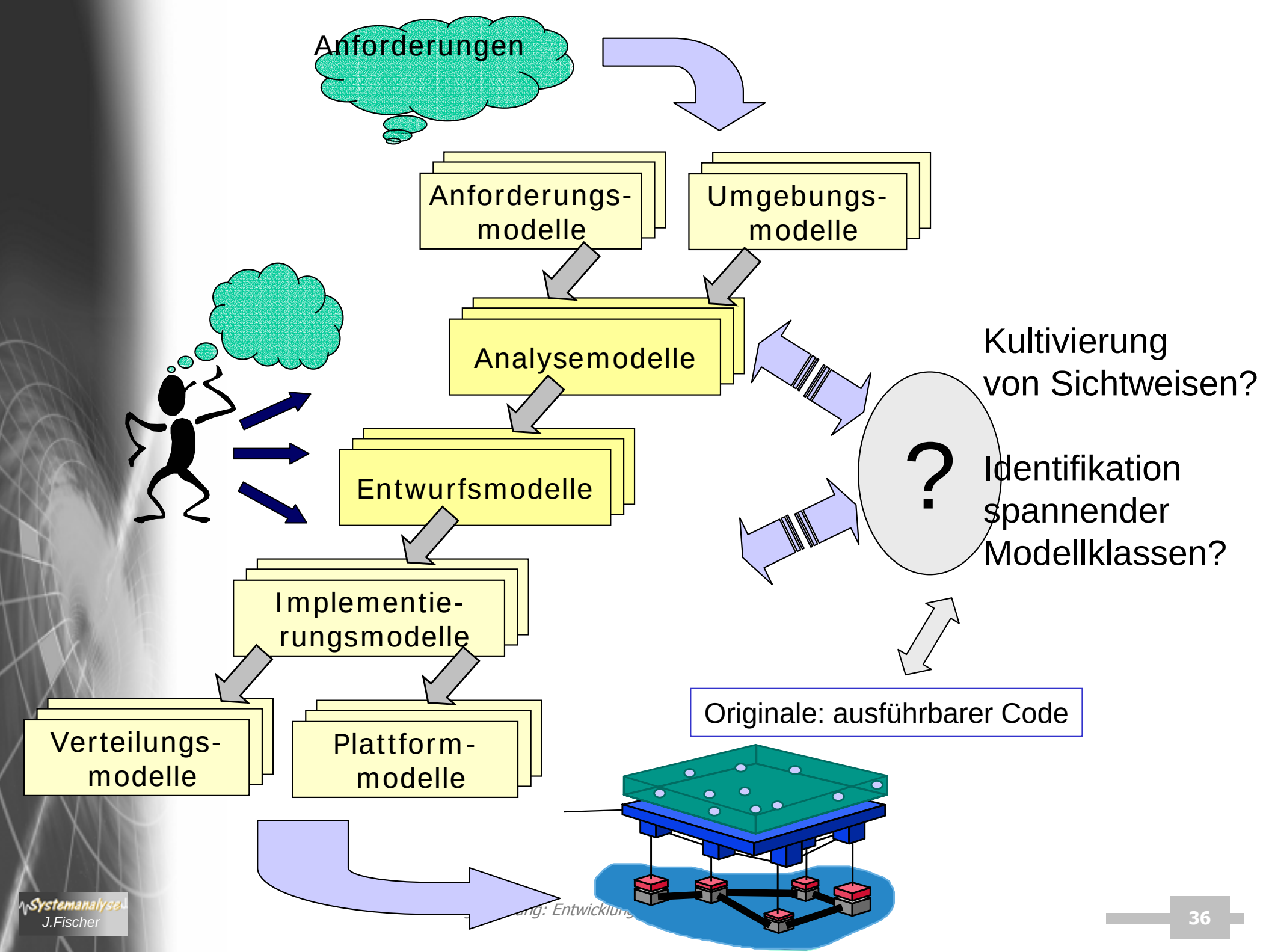
➔ **Model-Driven-Architecture (MDA)**

Modellbasierte Softwareentwicklung



Modellbasierte Softwareentwicklung




Anforderungen
Anforderungs-modelle
Umgebungs-modelle
Analysemodelle
Entwurfsmodelle
Implementie-rungsmodelle
Verteilungs-modelle
Plattform-modelle
?
Kultivierung von Sichtweisen?
Identifikation spannender Modellklassen?
Originale: ausführbarer Code

Modellsichten zur Entwicklung von Software-Komponenten

strukturelle Aspekte einer funktionalen Dekomposition:
CO
int
da

Verhaltensaspekte von CO-Typen (bei Neuentwicklung)
Blackbox, Greybox, Whitebox

Computational View

Konfigurationsaspekte der Software-Komponenten
port (provided, used)

Sichtweisen Und Modellkonzepte

Deployment View
Aspekte der Herstellung und Integration von Software-Komponenten: component, assembly, realize

Implementation View
Implementationsaspekte der Software-Komponenten
artefact, implementation element, state attribute

ungelöst: QoS-Aspekte
fallbasierte Beschreibung von Interaktions-Vorschriften und Anforderungen
binding, contract type, contract

Target Environment View
Eigenschaften von Knoten und Verbindungen

Konzepte zur Modellbeschreibung: eODL (ITU)

strukturelle Aspekte einer funktionalen Dekomposition:
CO
int
da

Verhaltensaspekte von CO-Typen (bei Neuentwicklung)
Blackbox, Greybox, Whitebox

Computational View

Konfigurationsaspekte der Software-Komponenten
port
(provided, used)

Sichtweisen Und Modellkonzepte

Deployment View
Aspekte der Herstellung und Integration von Software-Komponenten: component, assembly, realize

Implementation View
Implementationsaspekte der Software-Komponenten
artefact, implementation element, state attribute

ungelöst: QoS-Aspekte
fallbasierte Beschreibung von Interaktions-Vorschriften und Anforderungen
binding, contract type, contract

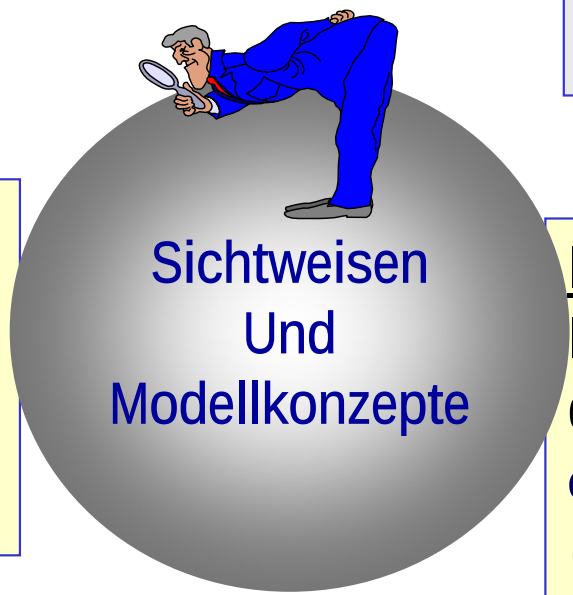
Target Environment View
Eigenschaften von Knoten und Verbindungen

Konzepte zur Modellbeschreibung: eODL (ITU)

strukturelle Aspekte einer funktionalen Dekomposition:
CO type, interface, interaction element, data types

Computational View

Konfigurationsaspekte der Software-Komponenten
port (provided, used)



Deployment View
Aspekte der Herstellung und Integration von Software-Komponenten: component, assembly, realize

Implementation View
Implementationsaspekte der Software-Komponenten
artefact, implementation element, state attribute

eODL= ODL+ = IDL++

SDL UML CIDL

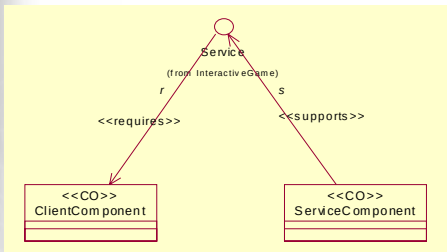
Target Environment View
Eigenschaften von Knoten und Verbindungungen

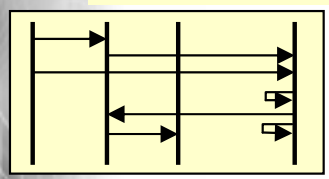
eODL ist technologie-unabhängig

Meta-modell

Konzept Raum
von eODL
Modellkonzepte und Relationen

Syntaktische Repräsentation der Konzepte



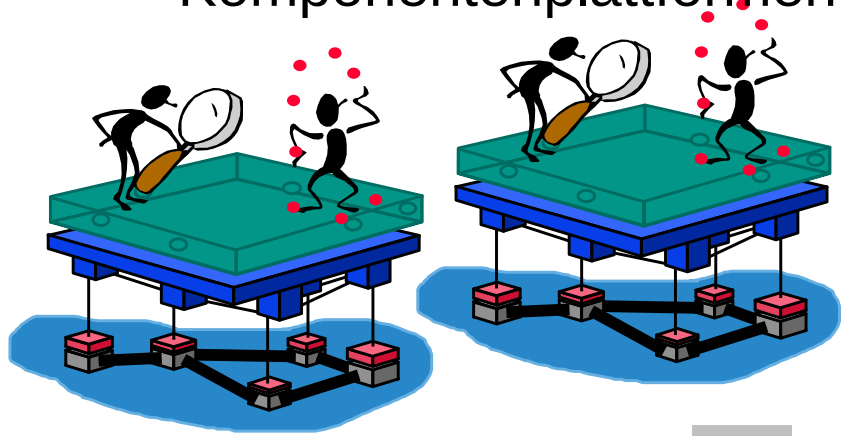


XXXXX
XXXXXX
XX
XXXXXX
XXXX

verschiedene Notationen

Abbildung der Konzepte (Code-Generierung)

verschiedene Komponentenplattformen



eODL hat verschiedene Repräsentationsformen

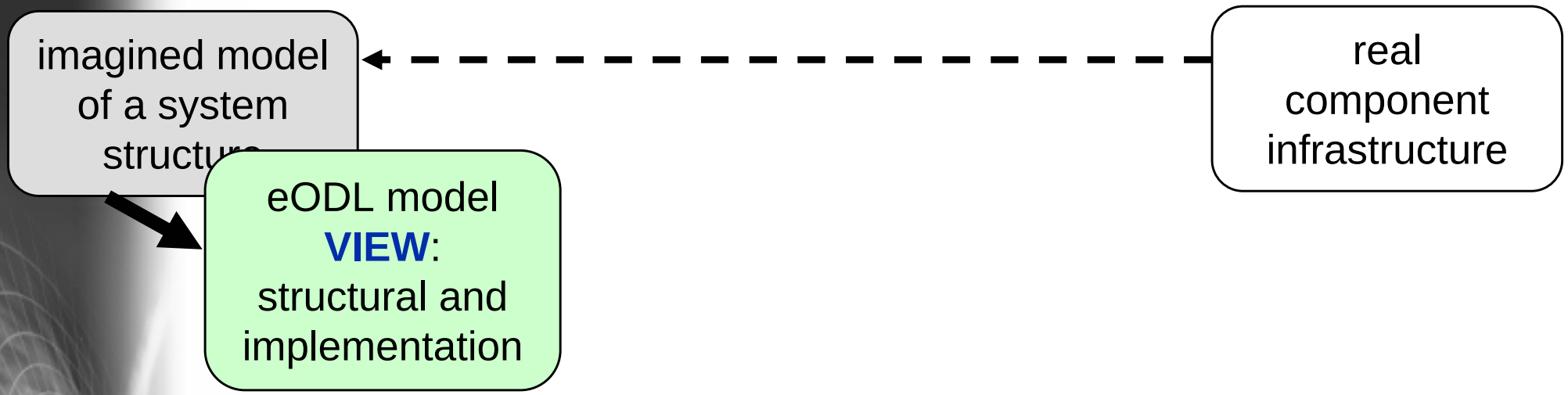
(1) textuell, IDL-like

(2) XML als Austauschformat (definiert durch XMI)

(3) grafisch
noch nicht vorhanden, aber möglich
Kandidat: UML-Profile für eODL

klarer Vorteil: UML-Tools sind verfügbar

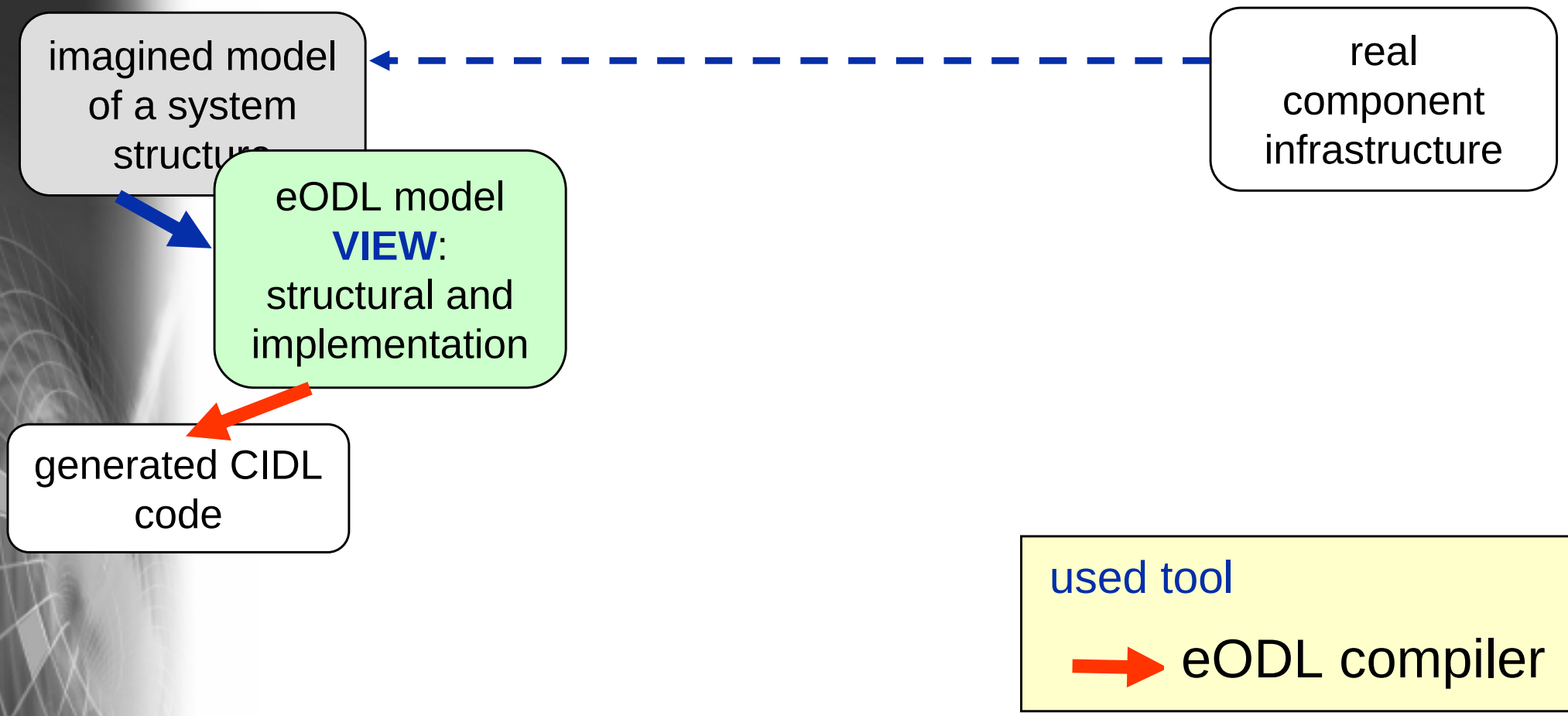
Komponenten-Entwicklung mit eODL



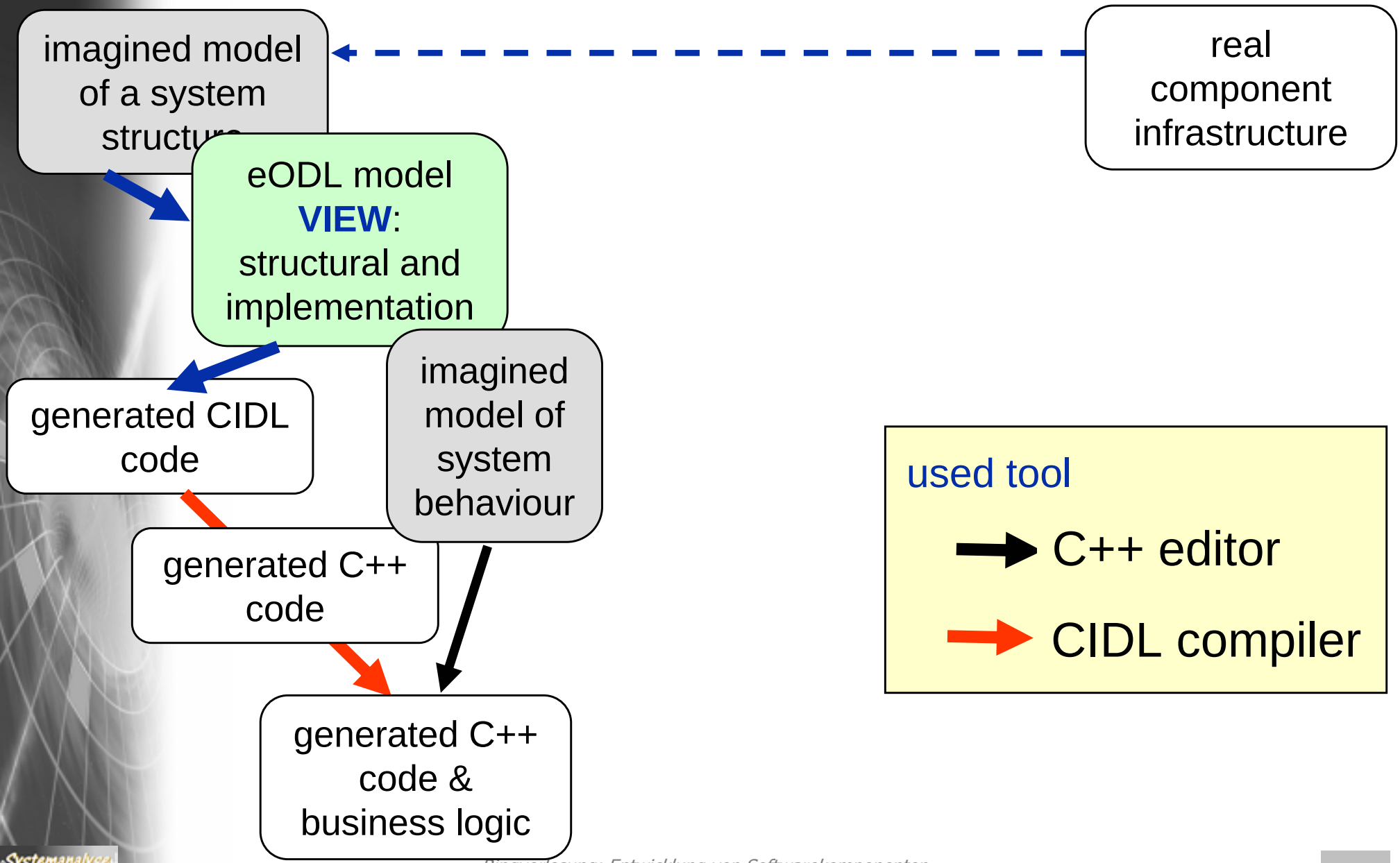
used tool

→ eODL editor

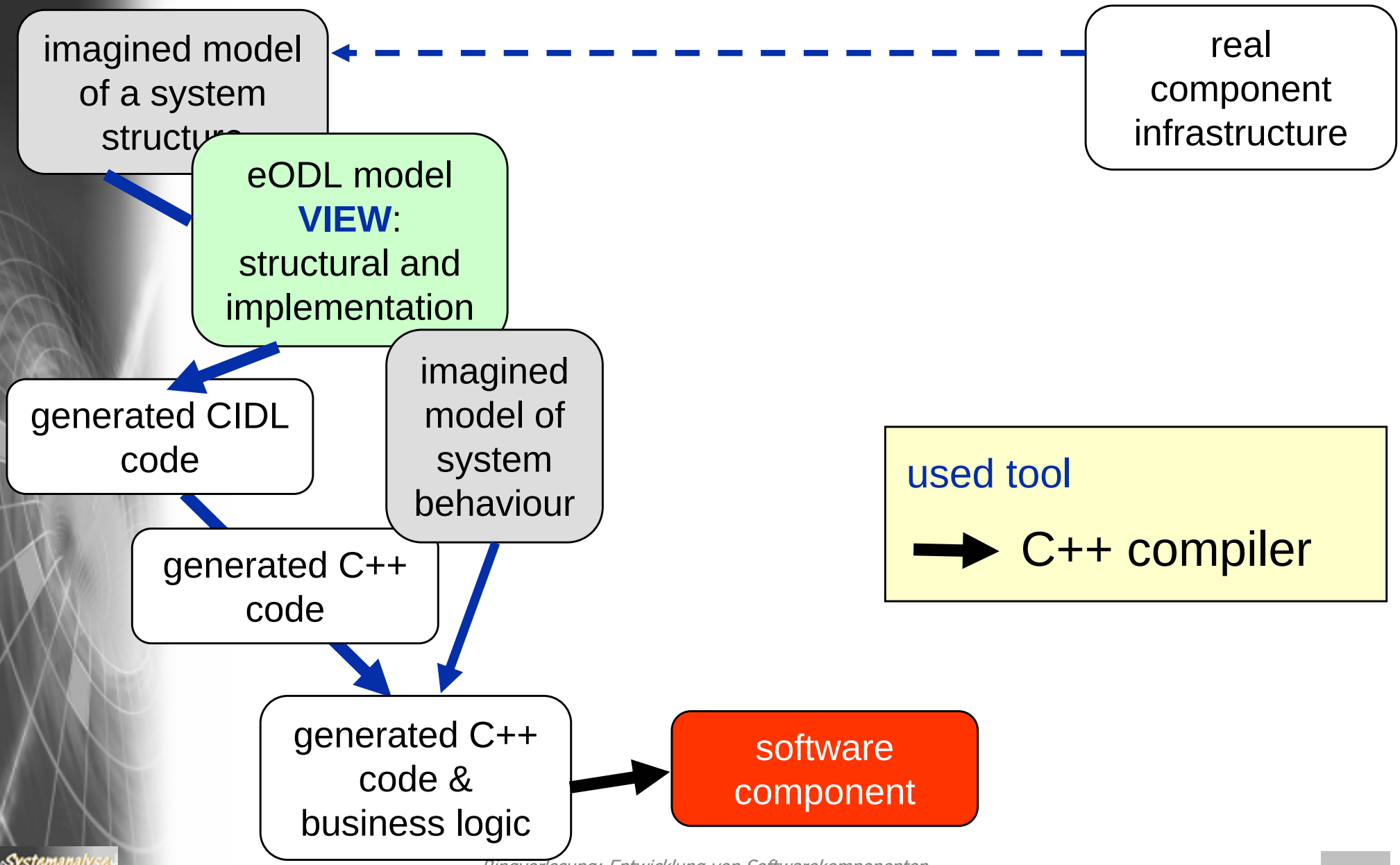
Component Development



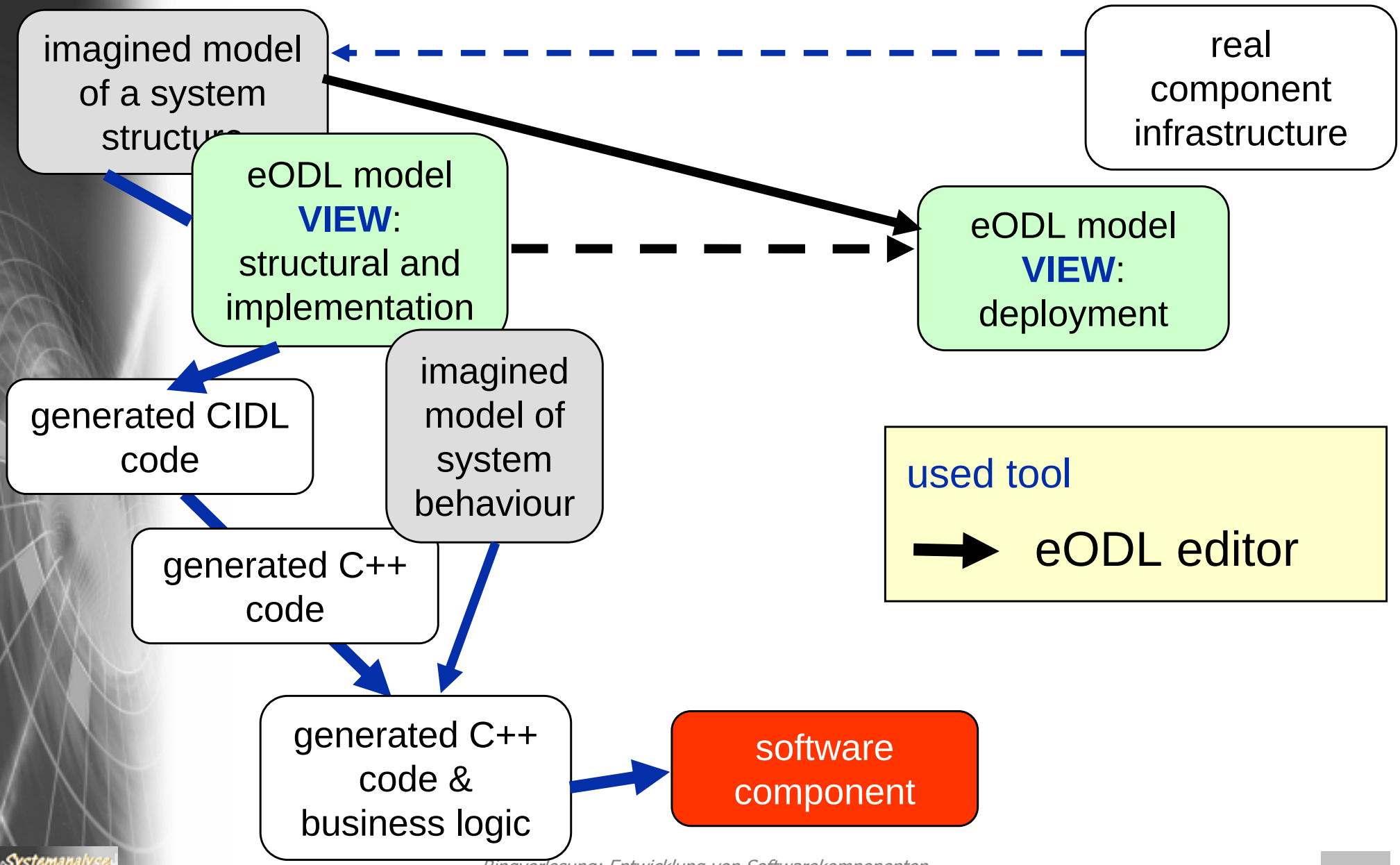
Component Development



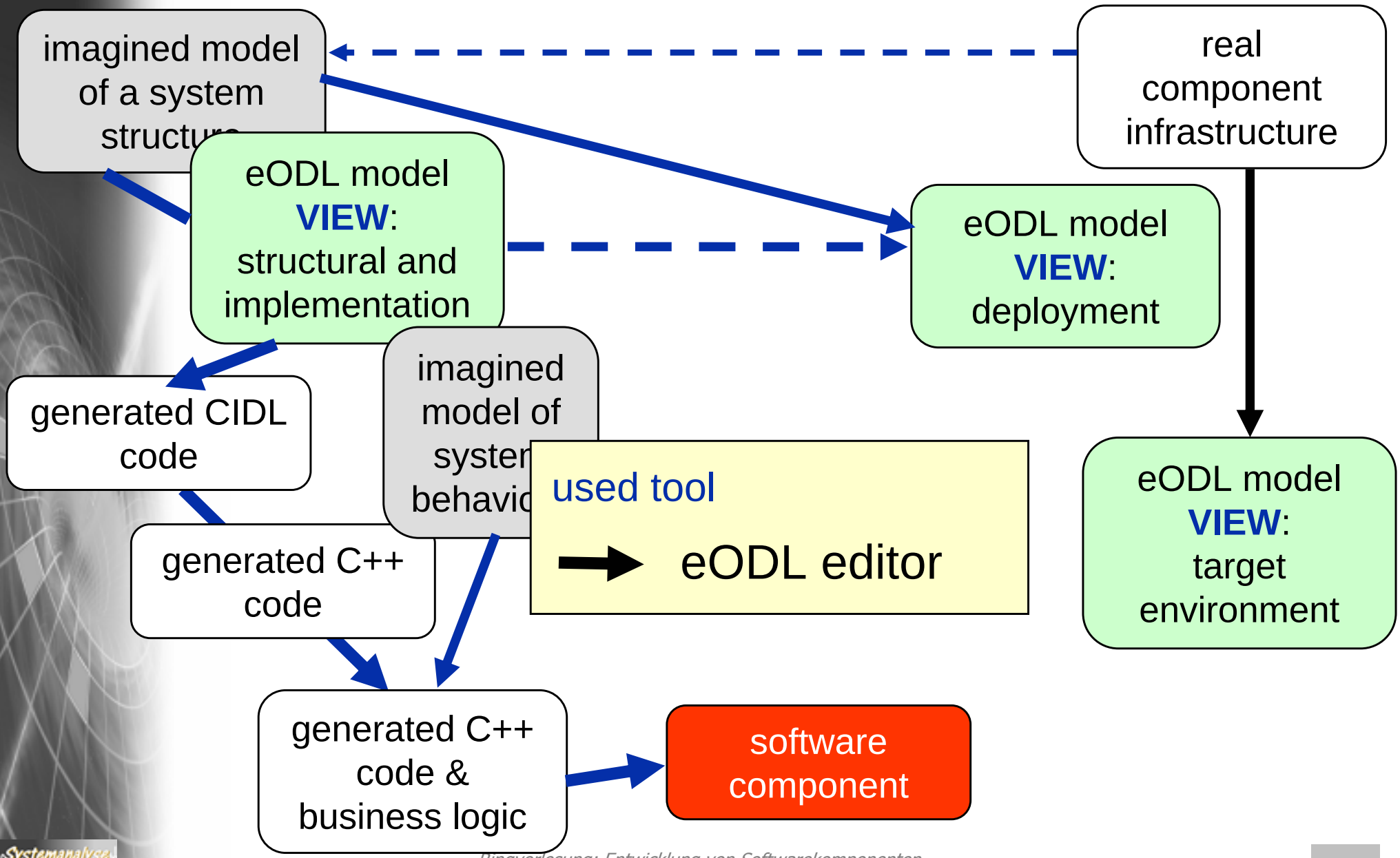
Component Development



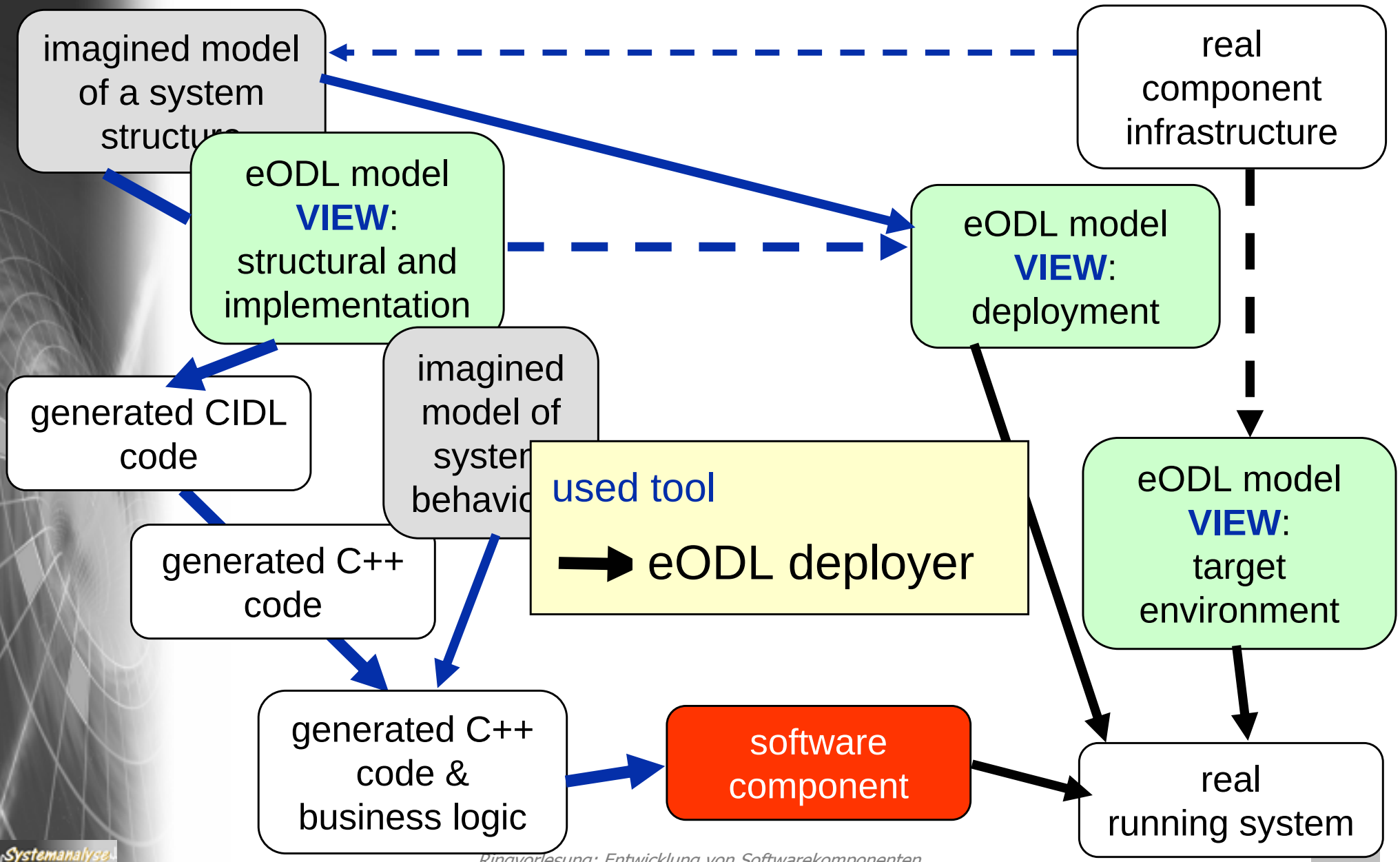
Component Development



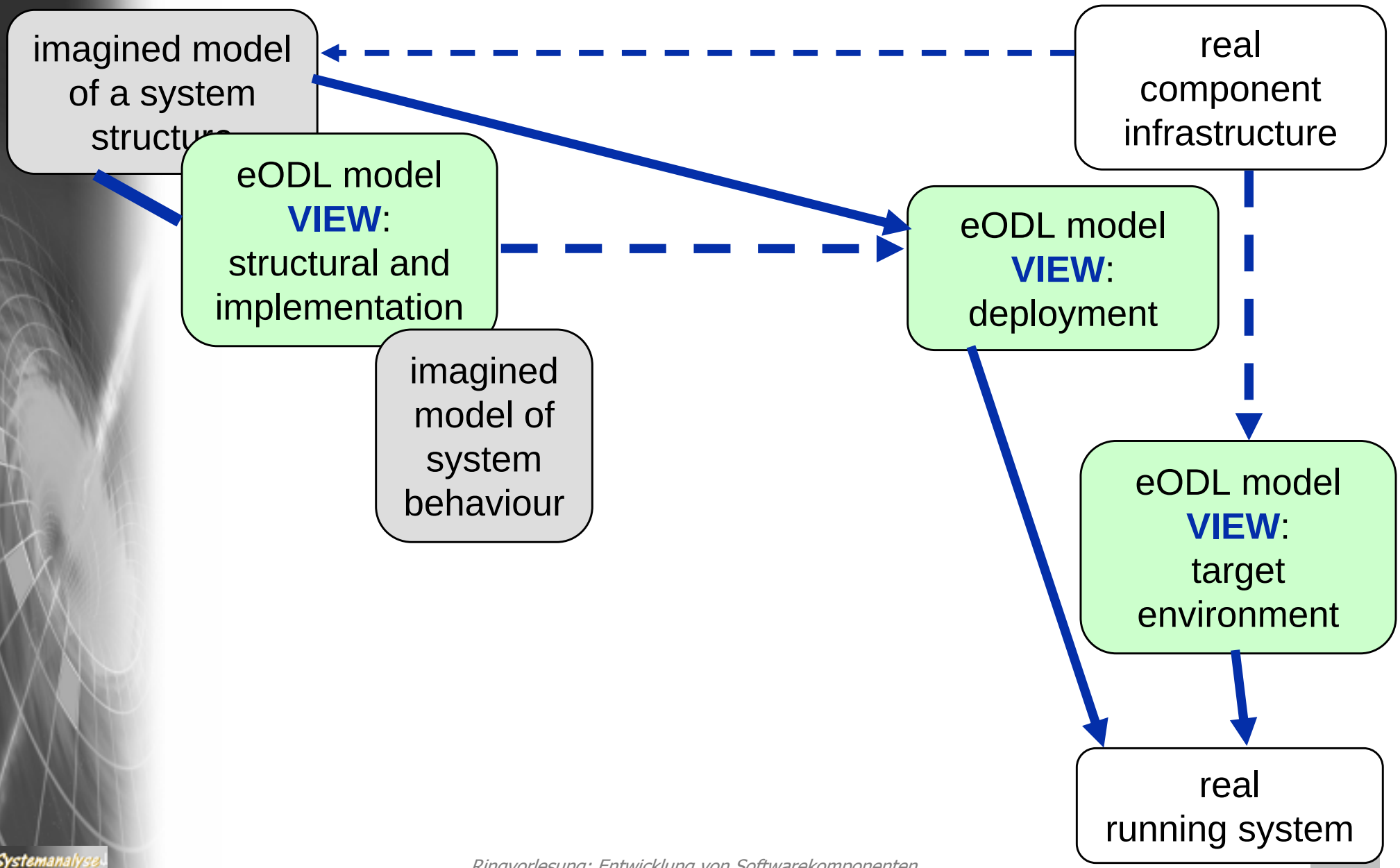
Component Development



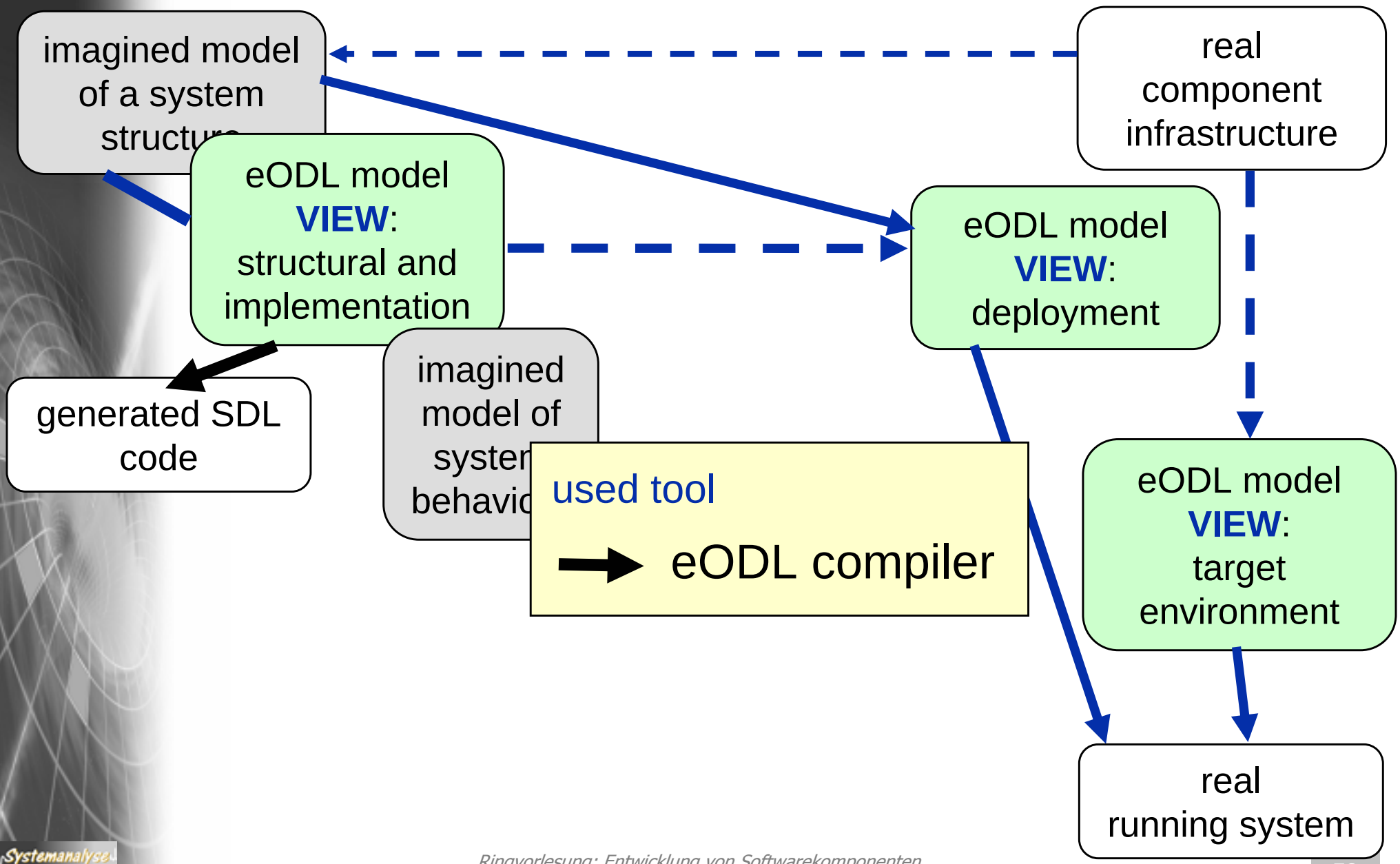
Component Development



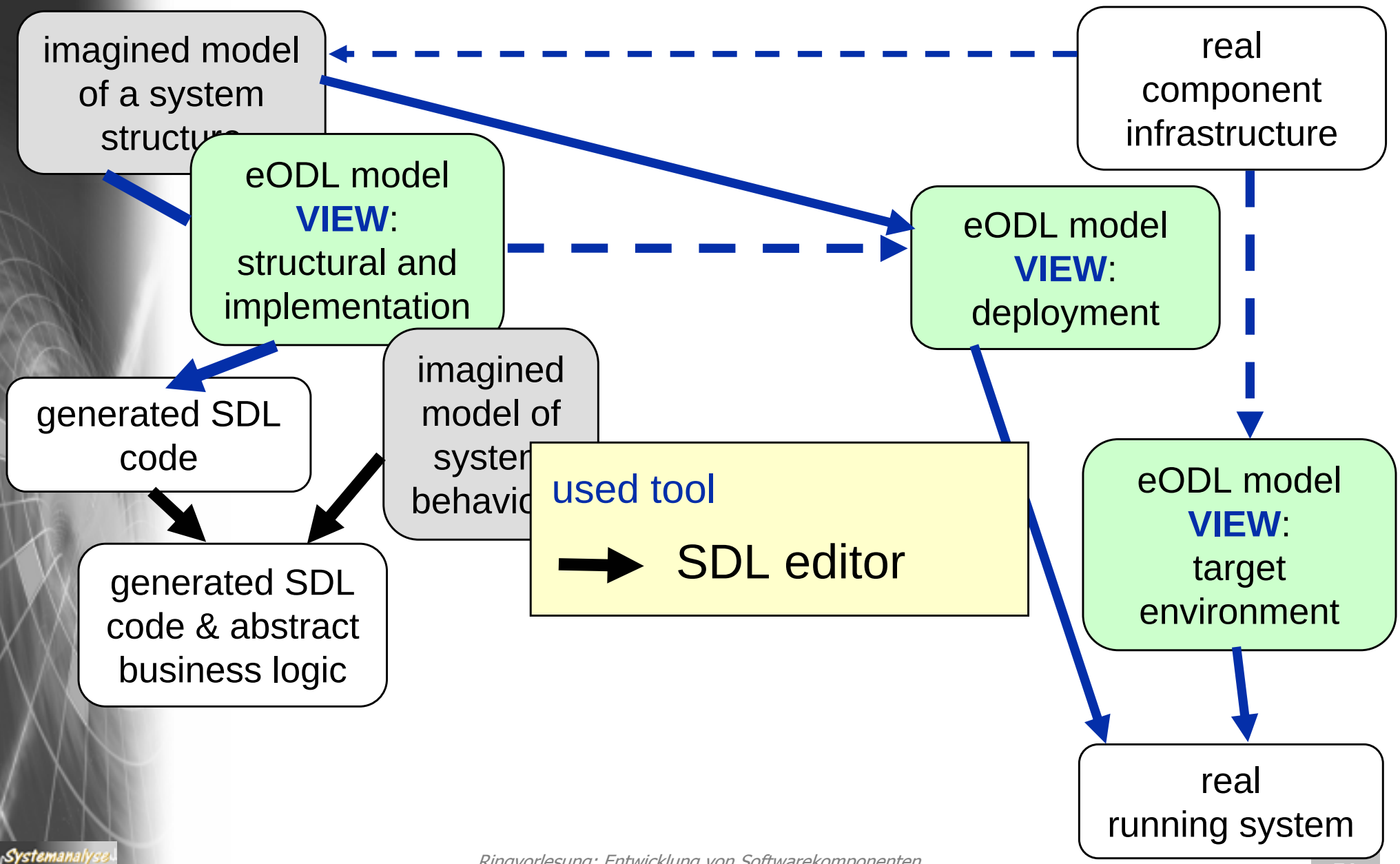
Component Development (alternative)



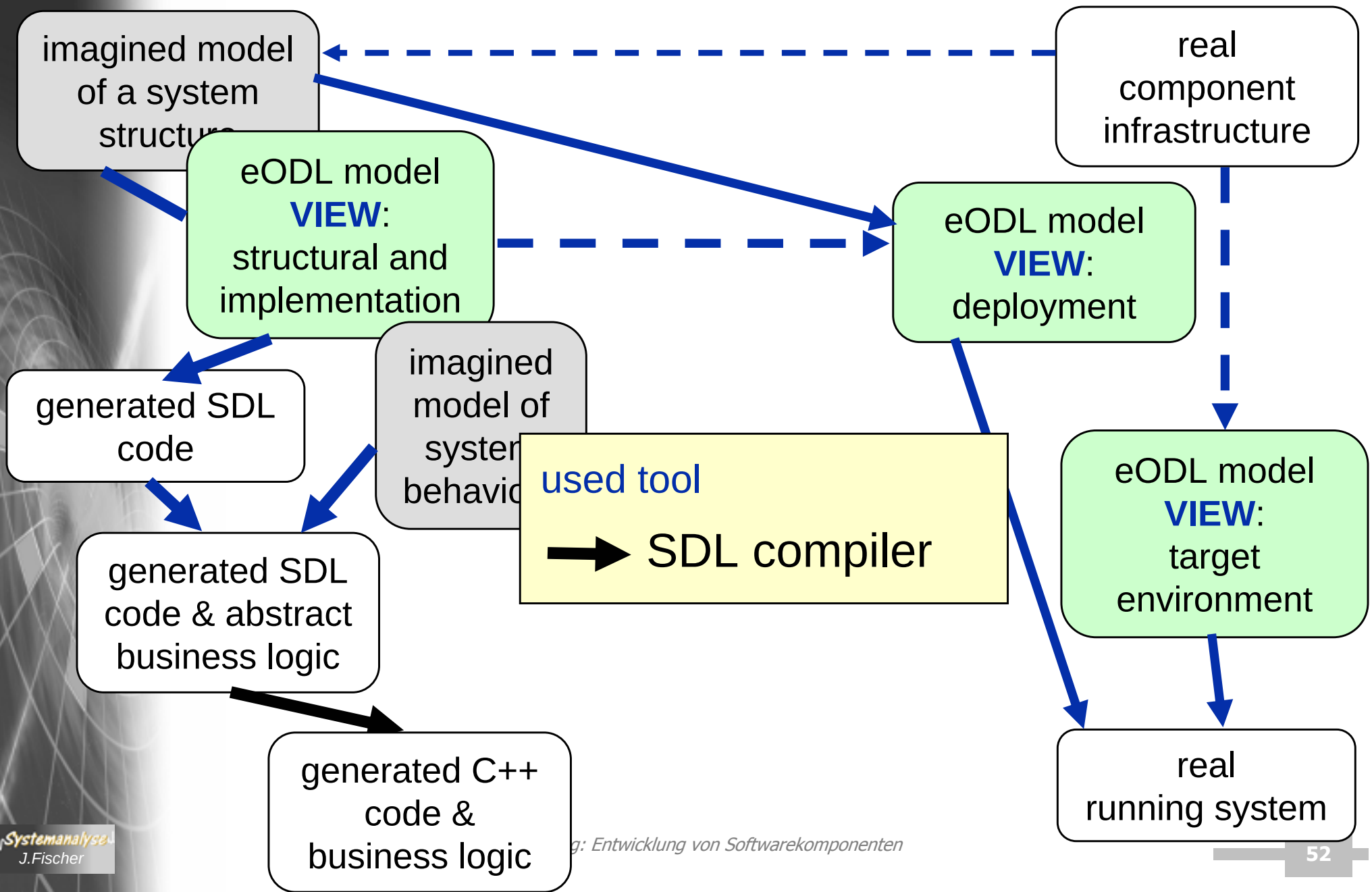
Component Development (alternative)



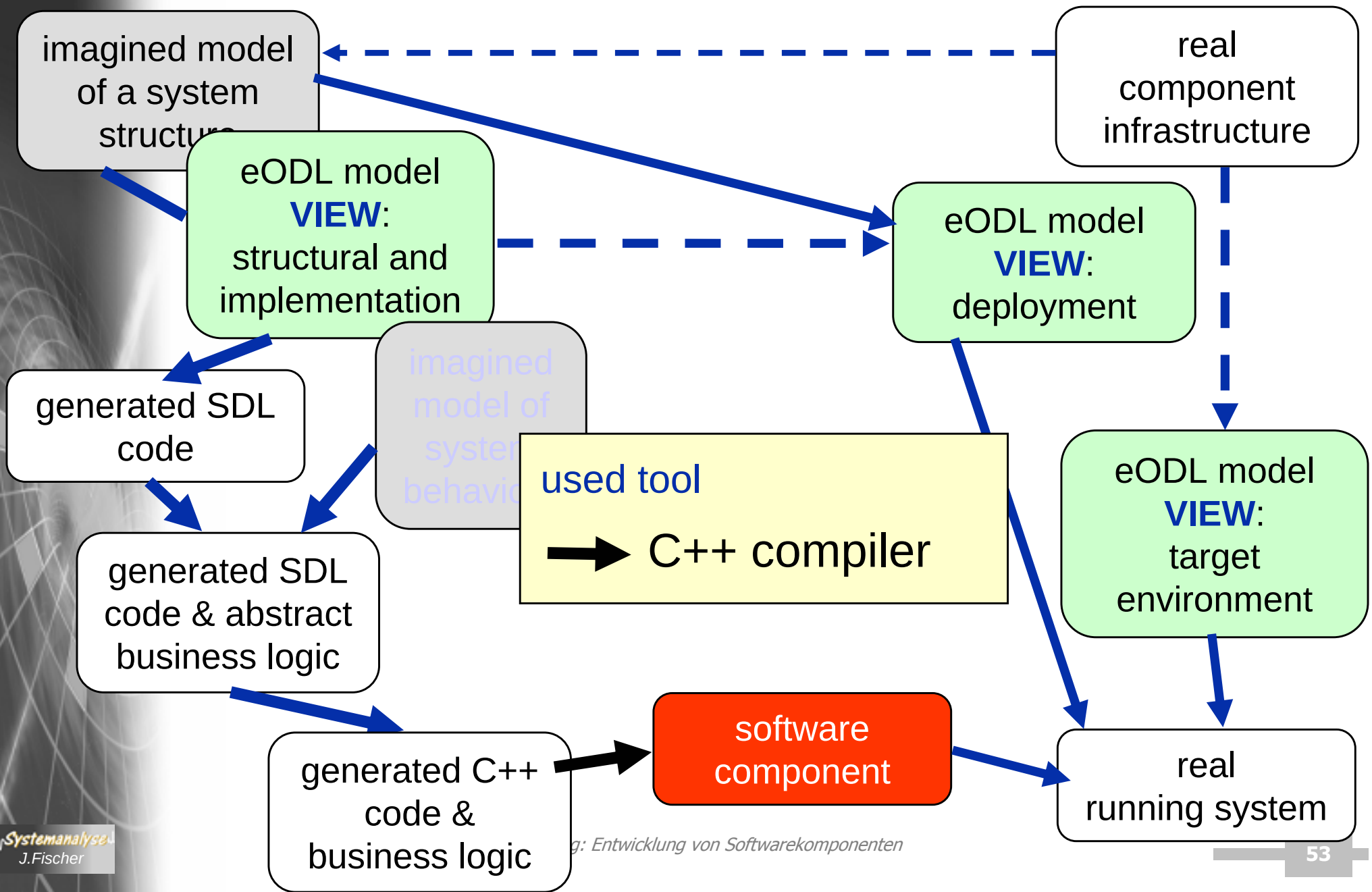
Component Development (alternative)



Component Development (alternative)



Component Development (alternative)

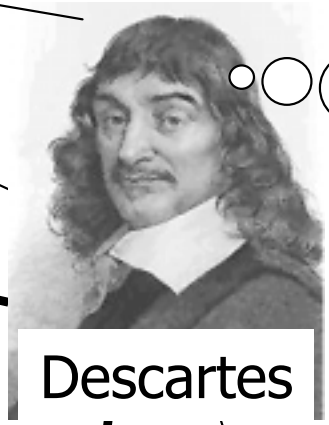


… weiterer Ablauf

1. Motivation
2. Middleware, CORBA
3. Vom CORBA-Objektmodell zum CORBA-Komponentenmodell
4. Modellbasierte Komponentenentwicklung
5. Beispiel
6. Zusammenfassung, Ausblick

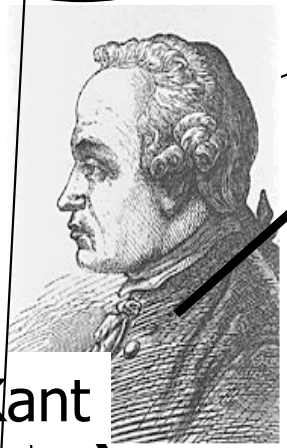
The Dining Philosophers Example

Thinking
Hungry
Starving
Eating
Dead



Descartes

Thinking
Hungry
Starving
Eating
Dead



Kant

Fork

Fork

Fork

Aristotle

Thinking
Hungry
Starving
Eating
Dead

as a simple distributed computer game

Modeling: System Structure by COs

Modeling: System Structure by COs

Philosopher

name = Kant

Philosopher

name = Descartes

Philosopher

name = Aristotle

Fork

Fork

Fork

Observer

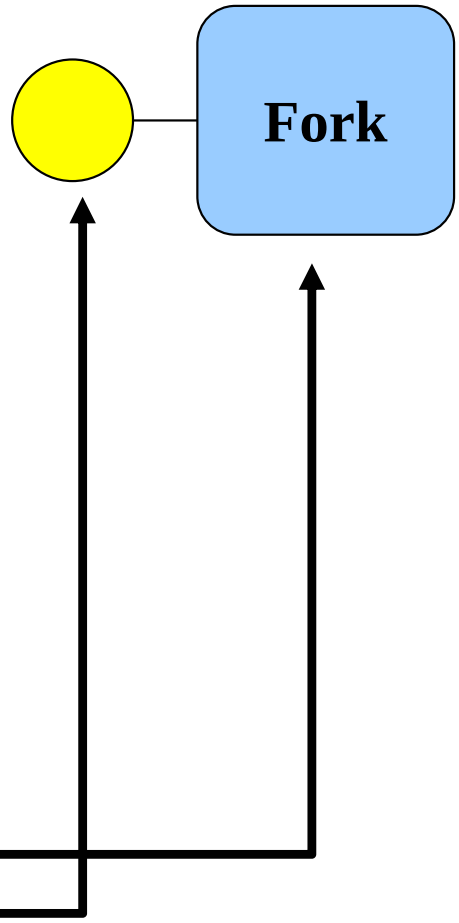
CO

provided port

used port

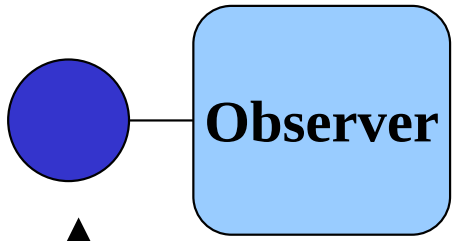
Fork Definition

```
exception ForkNotAvailable { } ;
exception NotTheEater { } ;

interface  i_Fork {
    void obtain_fork ( in o_Philosopher eater )
            raises ( ForkNotAvailable );
    void release_fork ( in o_Philosopher eater )
            raises ( NotTheEater );
};

CO  o_Fork {
    provide i_Fork fork;
};
```



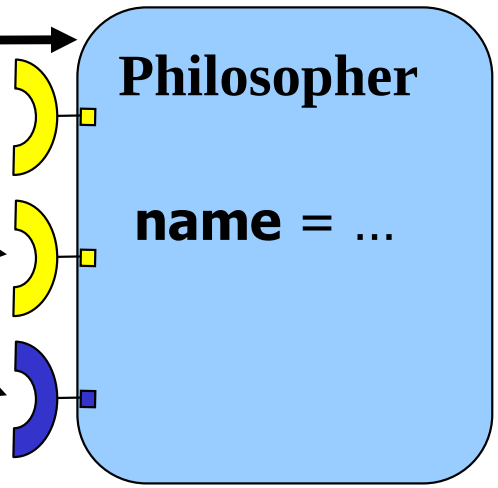
Observer Definition

```
valuetype Pstate {
    public e_PState state;
    public string name;
    public o_Philosopher philosoph;
};

signal PhilosopherState {
    PState carry_pstate;
};

interface i_Observer {
    consume PhilosopherState pstate;
};

CO o_Observer {
    provide i_Observer observer;
};
```



Philosopher Definition

```
enum e_Pstate {
    EATING, THINKING , SLEEPING,
    DEAD, CREATED, HUNGRY
} ;

CO o_Philosopher {
    use i_Fork left;
    use i_Fork right;
    use I_observer observer;
    attribute string name;
};
```



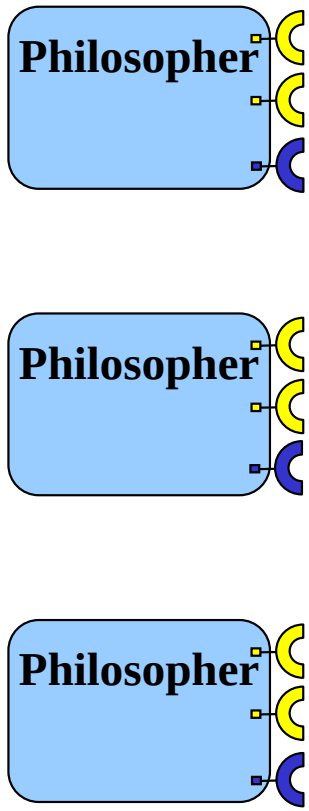
System Definition

assembly Dinner {

assembly (=system):
configuration of COs

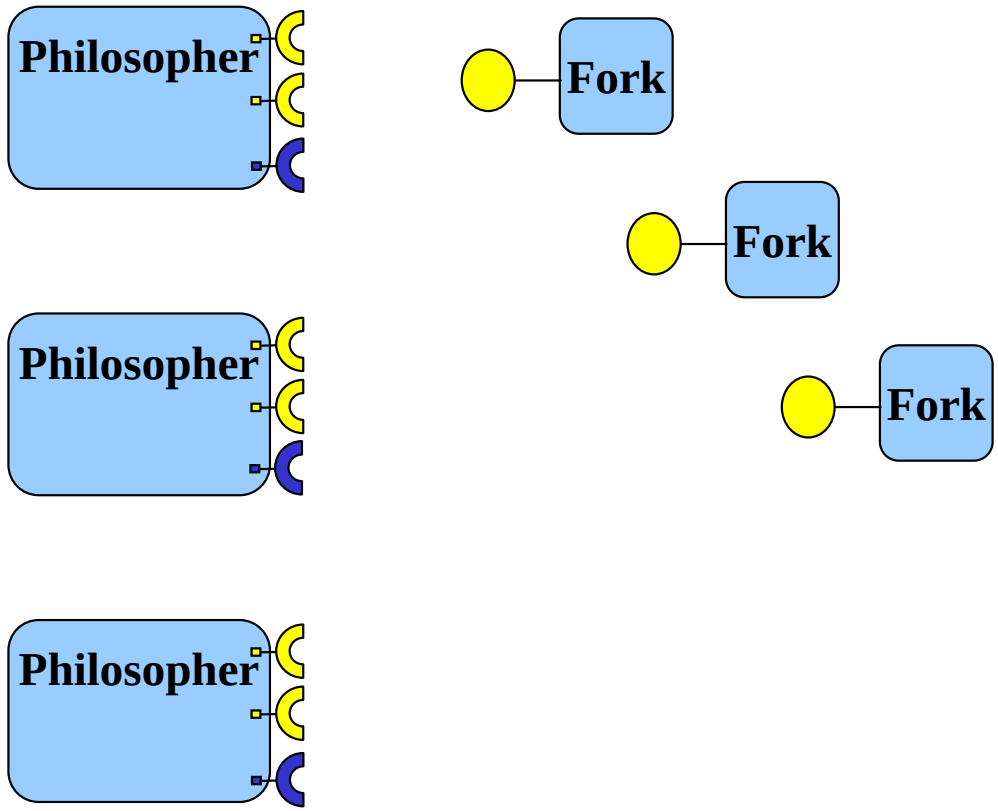
System Definition

```
assembly Dinner {
    p1 : o_Philosopher;
    p2 : o_Philosopher;
    p3 : o_Philosopher;
```



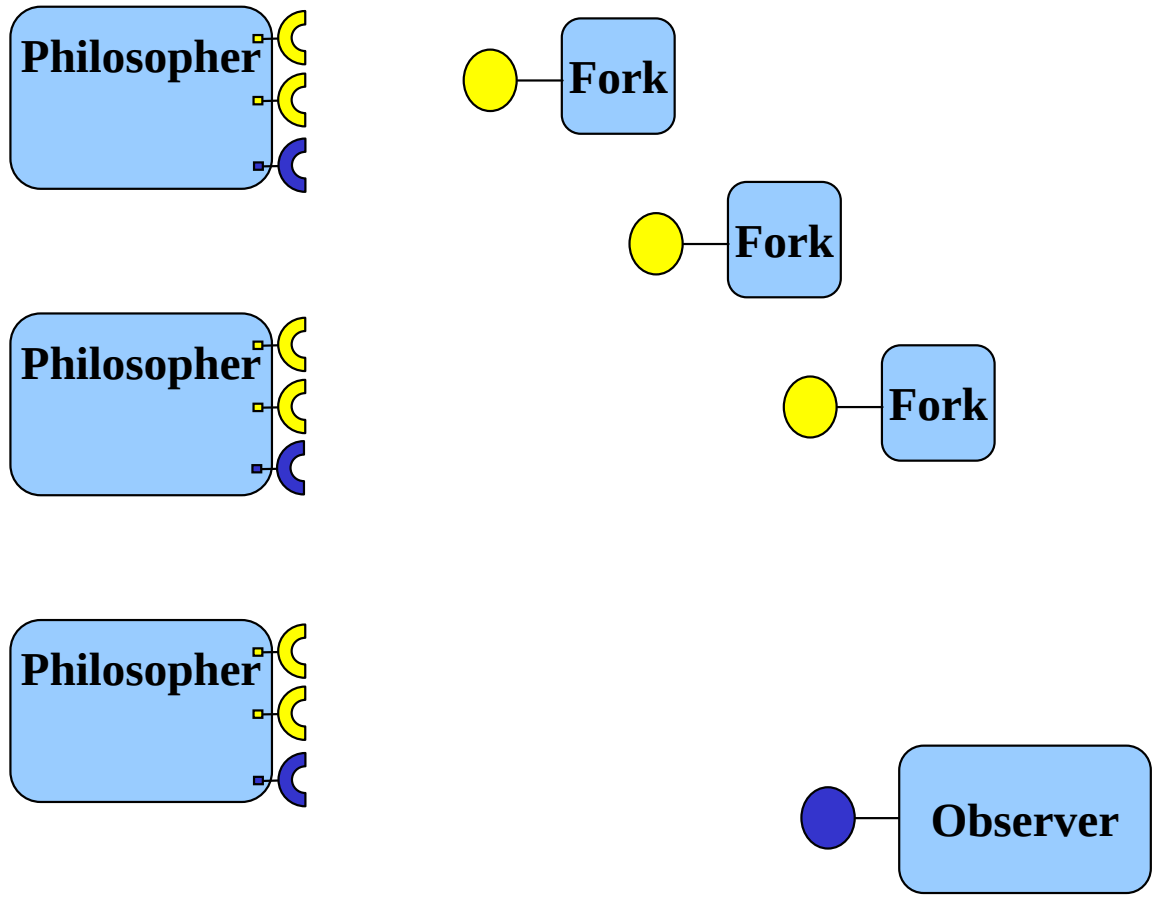
System Definition

```
assembly Dinner {
    p1 : o_Philosopher;
    p2 : o_Philosopher;
    p3 : o_Philosopher;
    f1 : o_Fork;
    f2 : o_Fork;
    f3 : o_Fork;
```



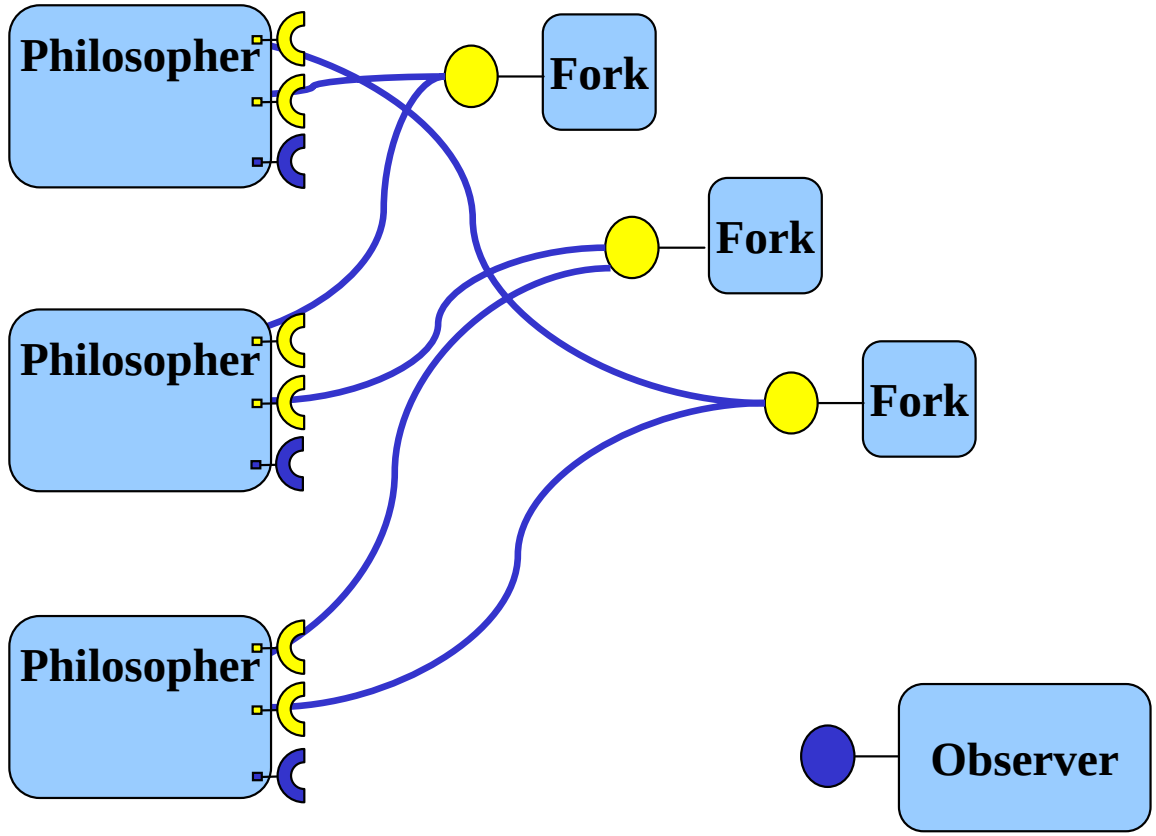
System Definition

```
assembly Dinner {
    p1 : o_Philosopher;
    p2 : o_Philosopher;
    p3 : o_Philosopher;
    f1 : o_Fork;
    f2 : o_Fork;
    f3 : o_Fork;
    o : o_Observer;
```



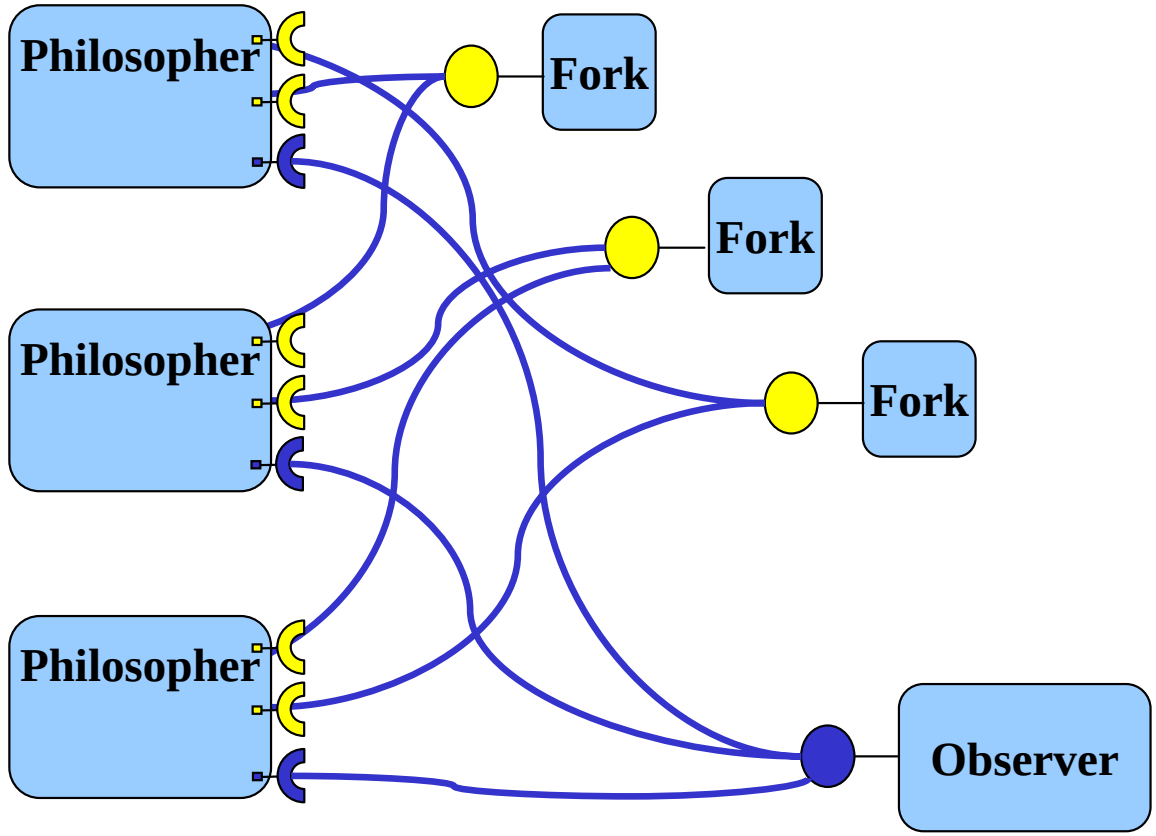
System Definition

```
assembly Dinner {
    p1 : o_Philosopher;
    p2 : o_Philosopher;
    p3 : o_Philosopher;
    f1 : o_Fork;
    f2 : o_Fork;
    f3 : o_Fork;
    o : o_Observer;
    connect c1{
            p1.left  = f1.fork;
            p2.right = f2.fork;
            ...
    };
```



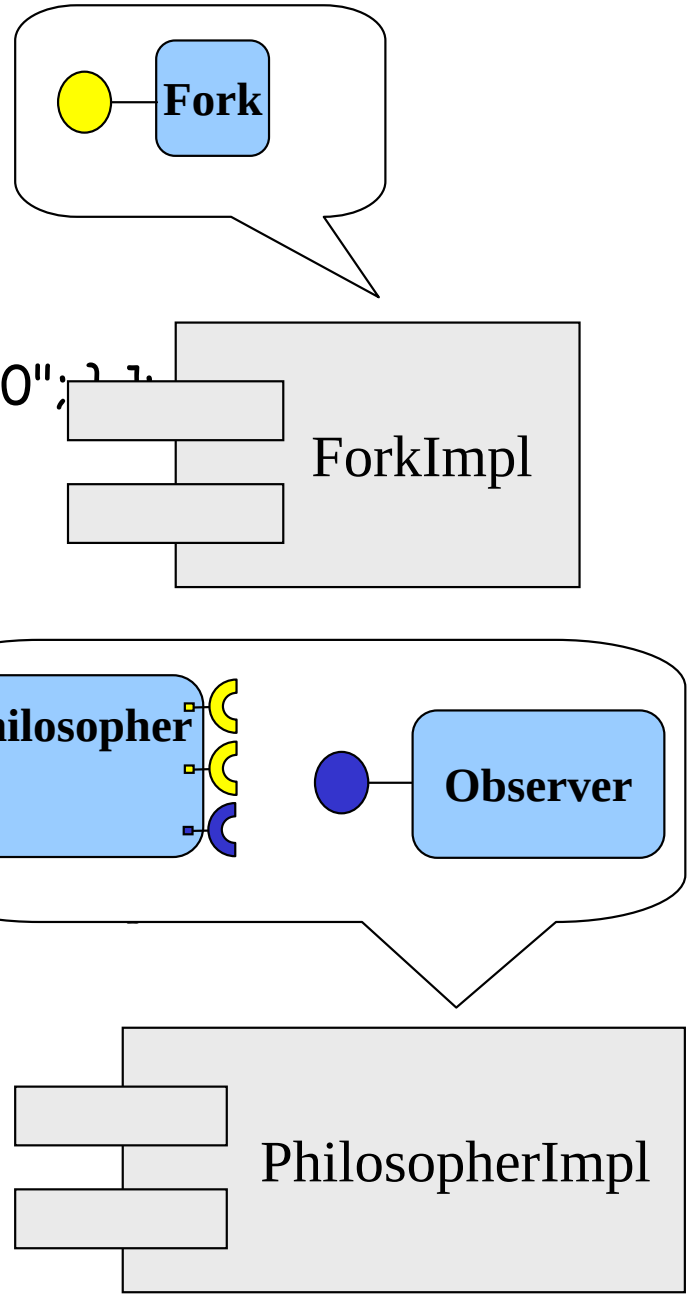
System Definition

```
assembly Dinner {
    p1 : o_Philosopher;
    p2 : o_Philosopher;
    p3 : o_Philosopher;
    f1 : o_Fork;
    f2 : o_Fork;
    f3 : o_Fork;
    o : o_Observer;
    connect c1{
            p1.left   = f1.fork;
            p2.right = f2.fork;
            ...
    };
    connect c2{ o.observer = p.observer; };
};
```



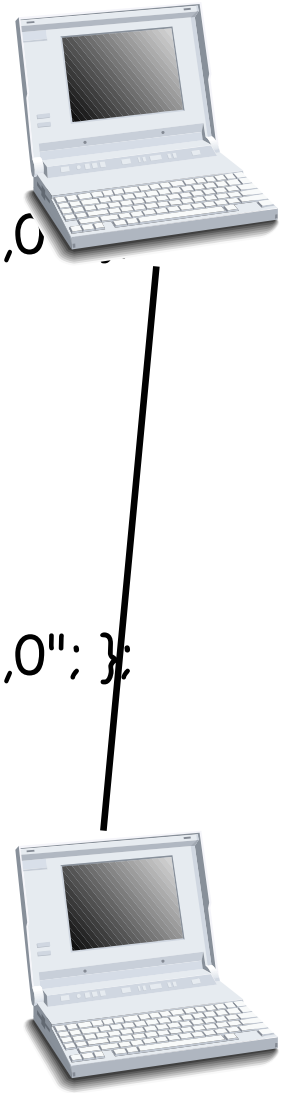
Definition of Software Components

```
softwarecomponent ForkImpl
realizes o_Fork {
    requires {
        property os =
        [ { name = "WINNT"; version = "4,0,0,0";
    };
};

softwarecomponent PhilosopherImpl
realizes o_Philosopher, o_Observer {
    requires {
        property os =
        [ { name = "WINNT"; version = "4,0,
    };
};
```



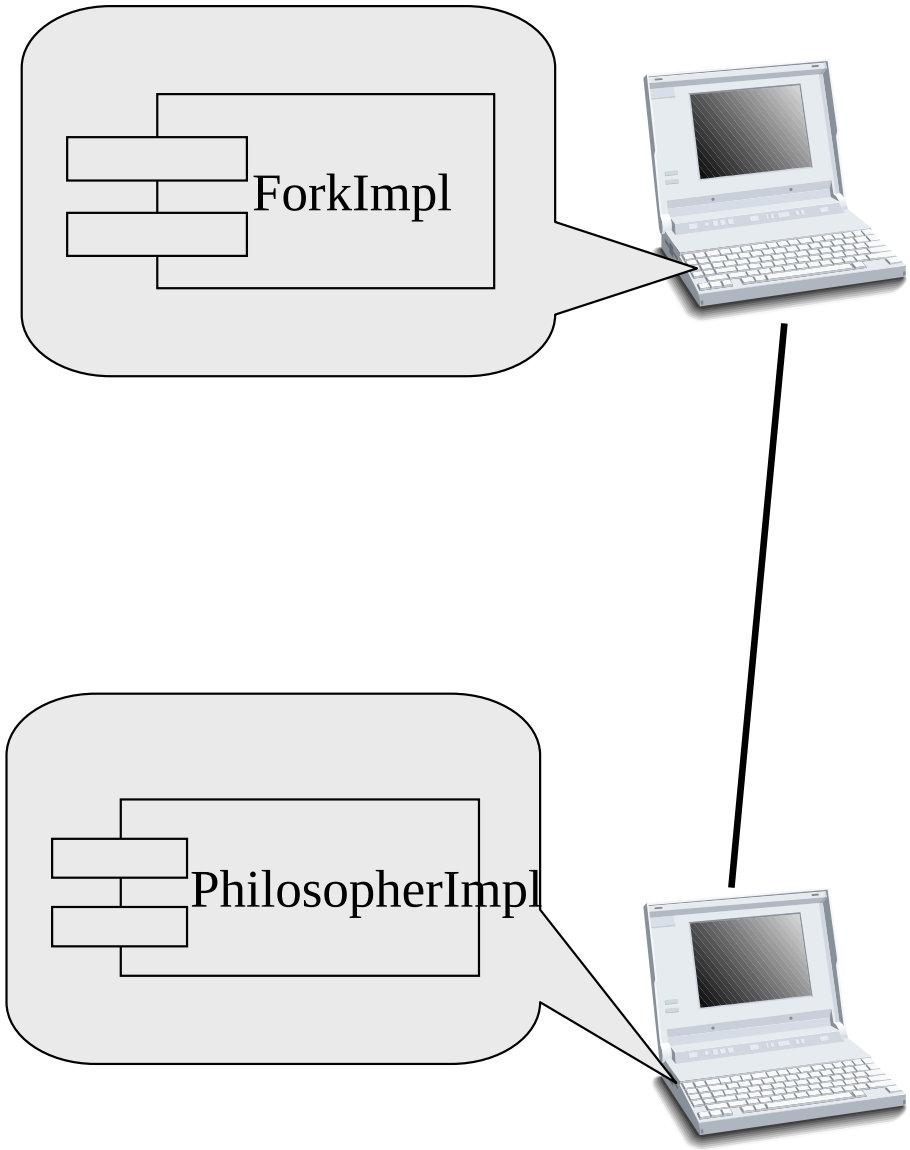
Definition of the Target Environment

```
environment MyEnv {
    node node1 {
        property os =
                { name = "WINNT"; version = "4,0,0,0
        property memory = 256;
    };

    node node2 {
        property os =
                { name = "WINNT"; version = "4,0,0,0"; };
        property memory = 128;
    };

    link link1 { node n1, n2; };
};
```



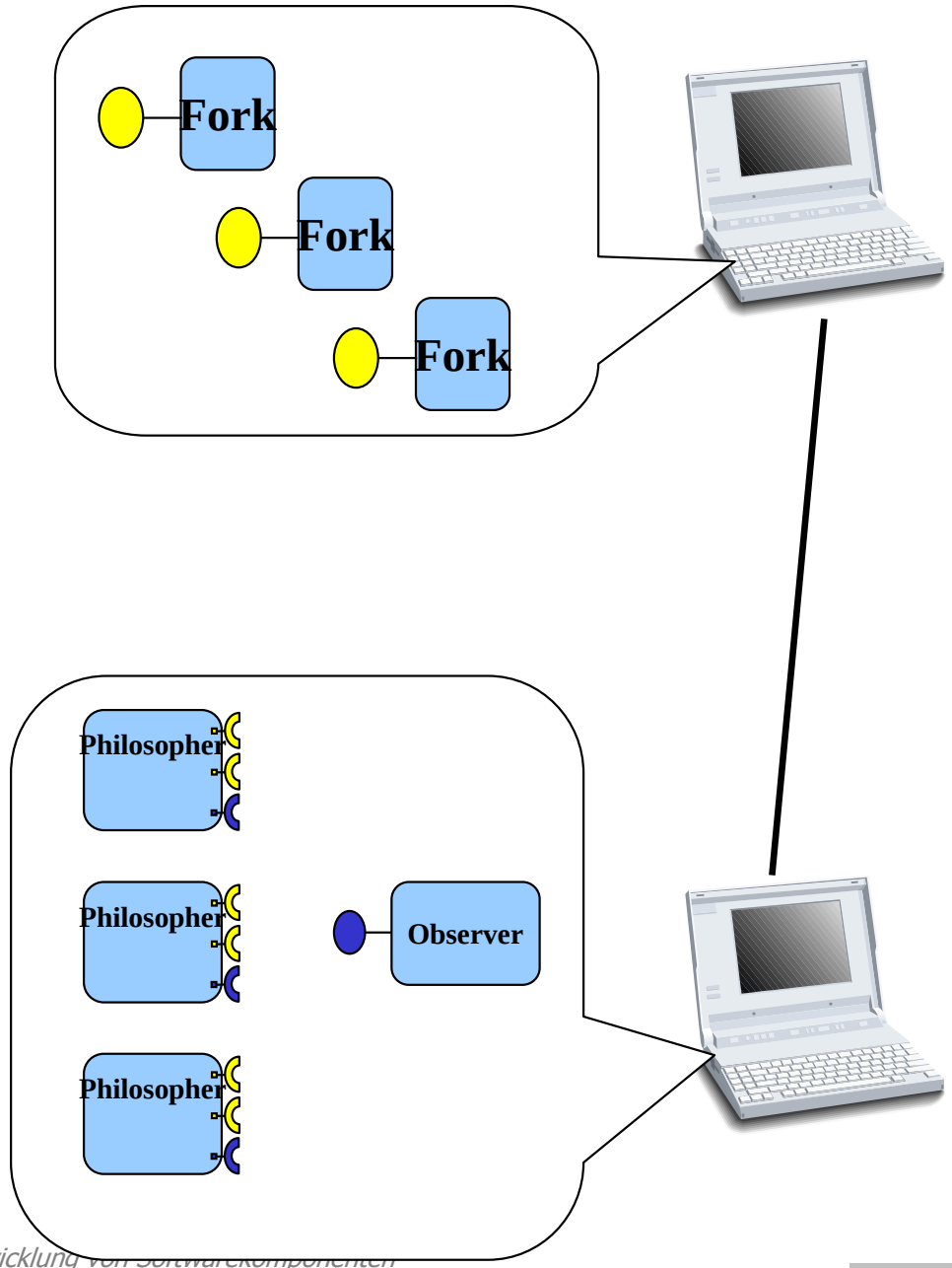
Definition of Installation

```
installation install
uses environment MyEnv {
    ForkImpl -> node1;
    PhilosopherImpl -> node2;
};
```



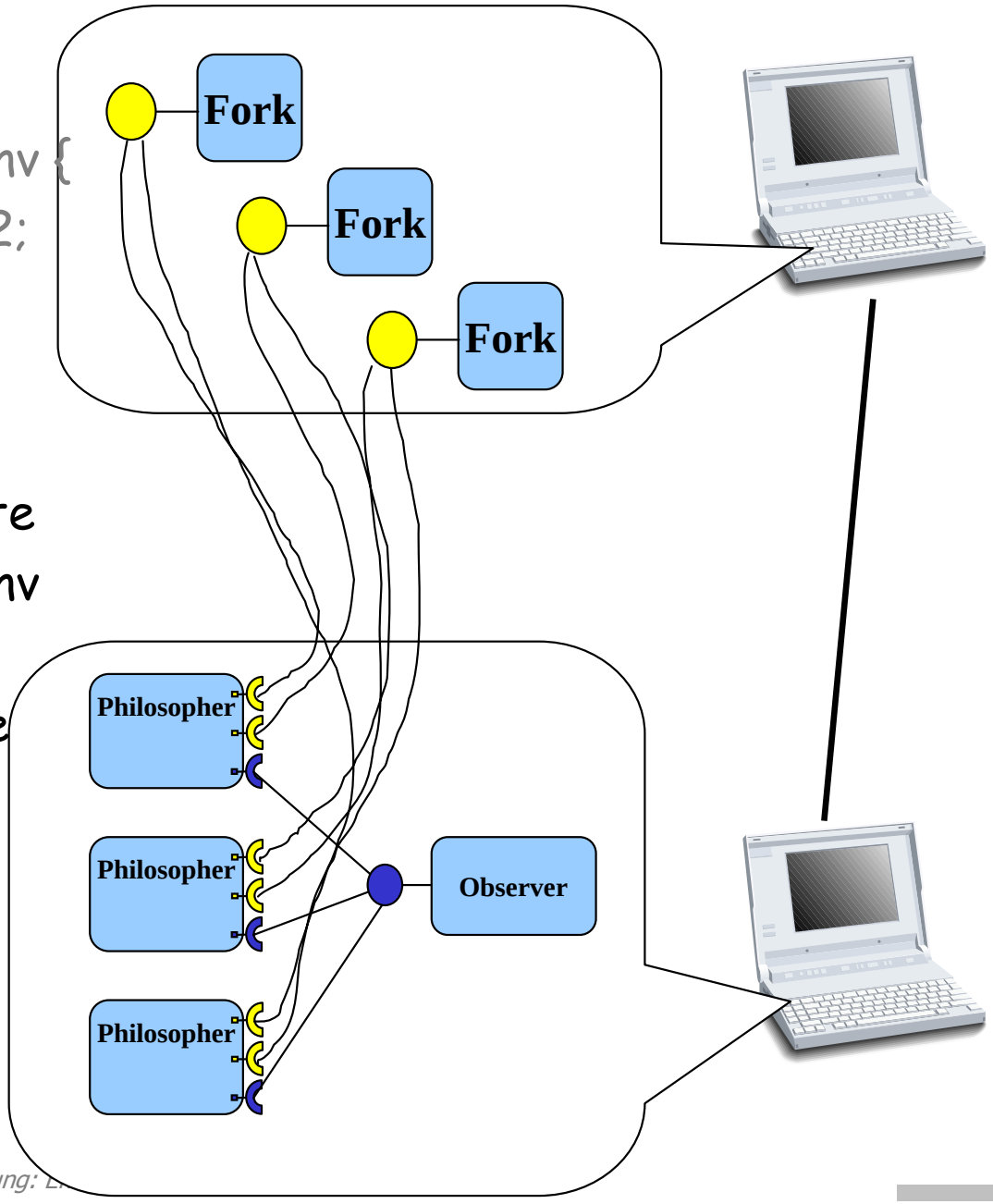
Definition of Instantiation

```
installation install
uses environment MyEnv {
    Philosopher -> node2;
    Fork -> node1;
};

instantiation instantiate
uses environment MyEnv
uses assembly Dinner {
    p1, p2, p3,  o -> node2;
    f1, f2, f3 -> node1;
};
```



Definition of Configuration

```
installation install
uses environment MyEnv {
    Philosopher -> node2;
    Fork -> node1;
};

instantiation instantiate
uses environment MyEnv
uses assembly Dinner {
    p1, p2, p3,  o -> node
    f1, f2, f3 -> node1;
};
```



… weiterer Ablauf

1. Motivation
2. Middleware, CORBA
3. Vom CORBA-Objektmodell zum CORBA-Komponentenmodell
4. Modellbasierte Komponentenentwicklung
5. Beispiel
6. Zusammenfassung, Ausblick

Zusammenfassung

- Bei Existenz geeigneter Komponentenmodelle und Infrastrukturen (inkl. Werkzeuge) ist eine komponentenbasierte Applikationsentwicklung realisierbar
- Lohnenswert ist die Untersuchung einer möglichen Trennung von technologie-abhängigen von technologie-unabhängigem Vorgehen
- Einbindung von Verhaltensbeschreibungen in die Komponentenentwicklung ist möglich, dennoch viele offene Probleme (Verhaltensäquivalenz)
- Modellbeschreibungssprachen auf der Basis einer gemeinsamen minimalen Meta-Modellplattform können einen großen Rationalisierungseffekt mit sich bringen

Ausblick

- Einbindung nicht-funktionaler Aspekte wichtig (➔ M. Malek)
- Einbindung von Testmodellen (➔ H. Schlingloff)
- Verhaltensäquivalenz (➔ W. Reisig)
- Ausweitung des MOF- Meta-Modellierungsansatzes (➔ E. Holz)
- Anwendung der CCM-Komponenten-Technologie zur OGSA-Emulation (➔ A. Reinefeld)

Fragen