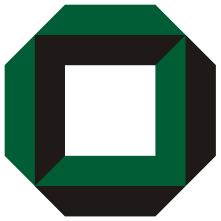
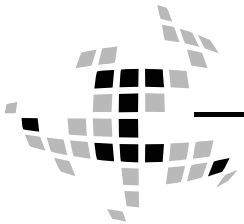


Universität Karlsruhe (TH)

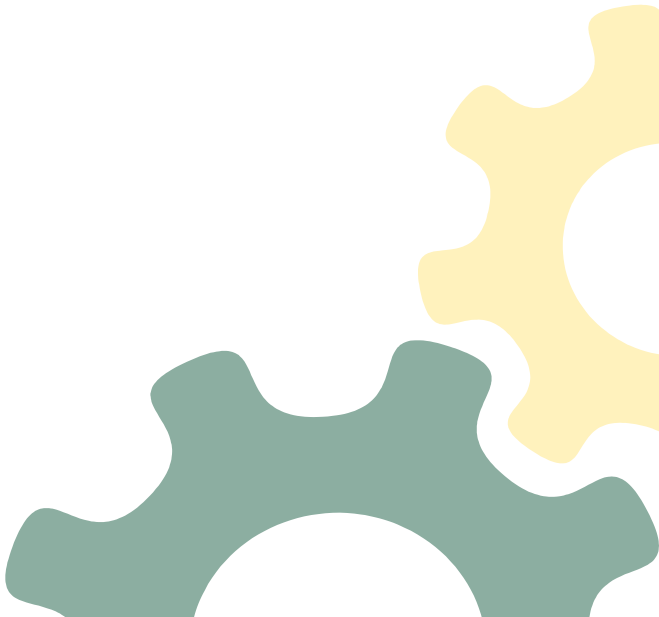
Forschungsuniversität · gegründet 1825

3.4 Architekturstile



Fakultät für **Informatik**

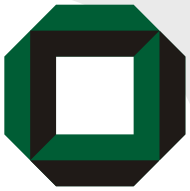
Lehrstuhl für Programmiersysteme



Abstrakte Maschine und Programmfamilie

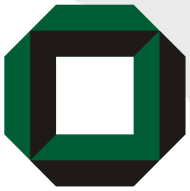
Def.: Eine **abstrakte Maschine** oder **virtuelle Maschine** ist eine Menge von Softwarebefehlen und -objekten, die auf einer darunterliegenden (abstrakten oder realen) Maschine aufbauen und diese ganz oder teilweise verdecken können.

Die Benutztrelation zwischen abstrakten Maschinen ist hierarchisch, d. h. zyklenfrei. Beispiele für abstrakte Maschinen: Programmiersprache, Betriebssystem, Anwendungskern, GUI-Bibliothek, Java VM.



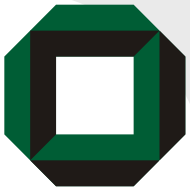
Abstrakte Maschine

- Eine abstrakte Maschine wird in der Regel von einem oder mehreren Modulen oder Paketen implementiert. Dabei werden die Softwarebefehle und -objekte von den Schnittstellen dieser Module bereitgestellt.
- Die darunter liegende Maschine muss ganz oder teilweise verdeckt werden, um Inkonsistenzen der beiden Maschinen zu vermeiden.
- Die Befehle der abstrakten Maschine sollen so gewählt werden, dass sie in einer Vielzahl von Programmen verwendet werden können.
- Eine abstrakte Maschine ist of als eine Programmbibliothek oder Application Programming Interface (API) verwirklicht.



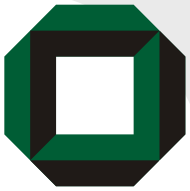
Beispiele für abstrakte Maschinen

- **Betriebssystem** stellt eine mächtige abstrakte Maschine zur Verfügung mit
  - Prozessverwaltung,
  - virtuellem Speicher,
  - Datenhaltung auf Hintergrundspeicher,
  - Kommunikation
  - Kommandosprachen, grafische Benutzeroberfläche
- Betriebssystem verdeckt gewisse privilegierte Befehle, die zur Implementierung der Betriebssystemdienste benötigt werden.
- Die Nicht-Nutzung der privilegierten Befehle wird zur Laufzeit überprüft. (führen zu einer Unterbrechung oder werden ignoriert (Pentium))



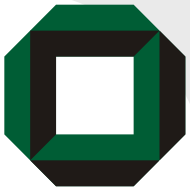
Beispiele für abstrakte Maschinen

- Java virtuelle Maschine: interpretiert Bytecode
- Java-Übersetzer
  - Bietet abstrakte Maschine, die in Java programmiert wird
  - Dabei sind Maschinenbefehle, Bytecode verdeckt
  - Die Verdeckung wird vom Übersetzer überprüft. (Analog: Interpretierer, Dokumentformatierer)



Beispiele für abstrakte Maschinen

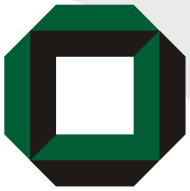
- **System zur Bearbeitung elektronischer Post:**
  Bietet Befehle zum Empfangen und Senden von Nachrichten, verdeckt die darunterliegenden Protokolle (Pufferung, Stückelung und Zusammensetzung von langen Nachrichten, Quittungen, Fehlerbehandlung), die für zuverlässigen Nachrichtenaustausch über Kommunikationskanäle erforderlich sind.
- **Makros:**
  Sind i. d. R. ungeeignet, die unterliegende Maschine zu verdecken. Z. B. können Call-Return-Assembler-Makros nicht verhindern, dass andere Instruktionen die Invarianten bez. des Aufrufkellers stören.



Programmfamilie/ SW Produktlinie

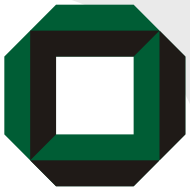
Def.: Eine **Programmfamilie** oder **Software-Produktline** ist eine Menge von Programmen, die erhebliche Anteile von Anforderungen, Entwurfsbestandteilen oder Softwarekomponenten gemeinsam haben.

Neue Mitglieder der Produktlinie können rasch entwickelt werden, in dem man Anteile von Anforderungen, Entwurf, Bibliotheken oder Komponenten wieder verwendet (evtl. leicht abgewandelt). Es ist also wesentlich kostengünstiger, ein neues Mitglied der Produktlinie zu erzeugen, als das Programm völlig von vorne zu entwickeln.



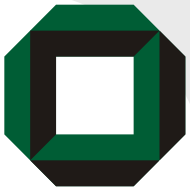
Programmfamilie/ SW Produktlinie

- Ziel:
  - Ausnutzung der Gemeinsamkeiten,
  - Wiederverwendung von Entwürfen, Spezifikationen und Anforderungen, Softwarekomponenten, Bibliotheken
  - zur Reduktion der Entwicklungs- und Wartungskosten.
- Variationspunkte:
  - Stellen in der Architektur, an denen Alternativen in Funktionalität, benutzte Plattform, etc. eingeführt werden können.



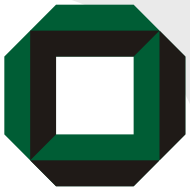
Wie unterscheiden sich Mitglieder einer Programmfamilie?

- Extern:
  - Sie laufen auf verschiedenen Hardware- und Betriebssystem-Konfigurationen.
  - Sie unterscheiden sich im Format der Ein-/Ausgabe.
  - Sie unterscheiden sich im Funktionsumfang, da manche Benutzer nur eine Teilmenge der Funktionen benötigen (und nicht gezwungen sein sollten, für nicht benötigte Funktionalität zu bezahlen, Betriebsmittel bereitzustellen oder Effizienzeinbußen hinzunehmen.)



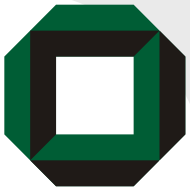
Wie unterscheiden sich Mitglieder einer Programmfamilie?

- Intern:
  - Sie unterscheiden sich in Datenstrukturen und Algorithmen, wegen Unterschieden in den zur Verfügung stehenden Betriebsmitteln oder Unterschieden in den Leistungsanforderungen.
  - Sie unterscheiden sich in Datenstrukturen und Algorithmen, wegen Unterschieden im Umfang der Eingabedaten oder der relativen Häufigkeit gewisser Ereignisse (z.B. Anfragen an das System).



Programmfamilie/ SW Produktlinie

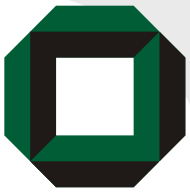
- Wichtiger Unterschied zwischen Allgemeinheit und Flexibilität:
  - Ein Programm ist allgemein, falls es in vielen Situationen ohne Änderung benutzt werden kann. (Allgemeinheit bedeutet meist hohe Laufzeit- und Speicherplatzkosten.)
  - Ein Programm ist flexibel, falls es leicht für viele Situationen abgeändert werden kann. (Flexibilität erfordert höhere Entwurfskosten.)
- Alle Überlegungen bis jetzt zielen darauf ab, ein System flexibel zu machen und dadurch Wartung zu vereinfachen.
- Ein flexibler Entwurf ist änderungsfreundlich.



Architekturstile

Architekturstile legen den Grobaufbau eines SW-Systems fest. Wir behandeln folgende Stile:

- Schichtenarchitektur
- Klient/Dienstgeber (engl. client/server)
- Partnernetze (engl. peer-to-peer)
- Datenablage (engl. repository)
- Model/View/Controller
- Fließband (engl. pipeline)
- Rahmenarchitektur (engl. framework)



Schichtenarchitektur (engl. Layered Architecture)

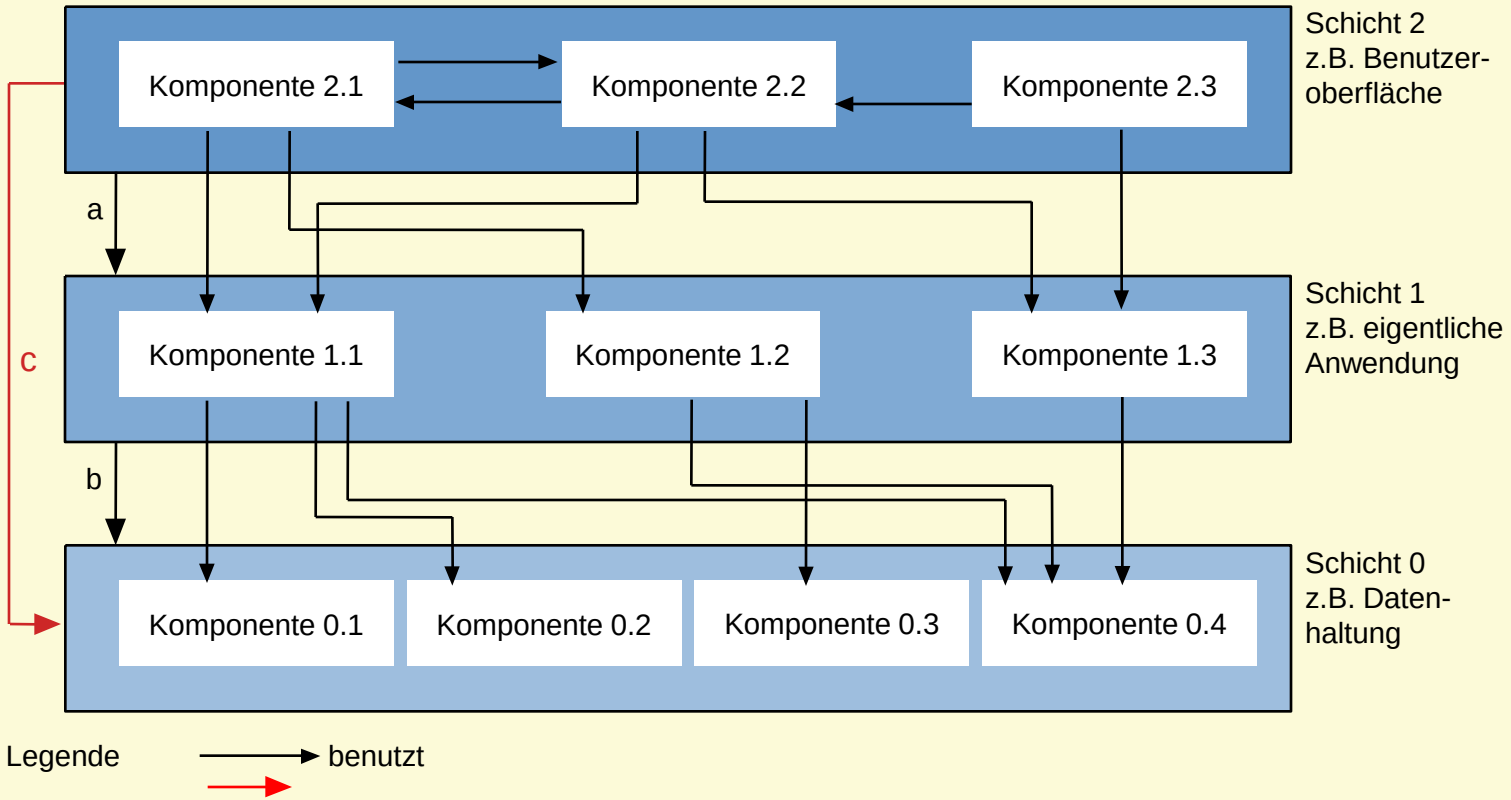
Def.: Eine **Schichtenarchitektur** ist die Gliederung einer Softwarearchitektur in hierarchisch geordnete Schichten.

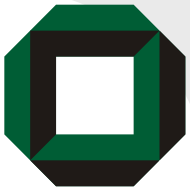
Eine Schicht besteht aus einer Menge von Software-Komponenten (Module, Klassen, Objekte, Pakete) mit einer wohldefinierten Schnittstelle, nutzt die darunterliegenden Schichten und stellt seine Dienste darüberliegenden Schichten zur Verfügung.

Zwischen den einzelnen Schichten ist die Benutztrelation linear, baumartig, oder ein azyklischer Graph. Innerhalb einer Schicht ist die Benutztrelation beliebig.

Schichtenarchitektur

- Beispiel für eine Drei-Schichten-Architektur



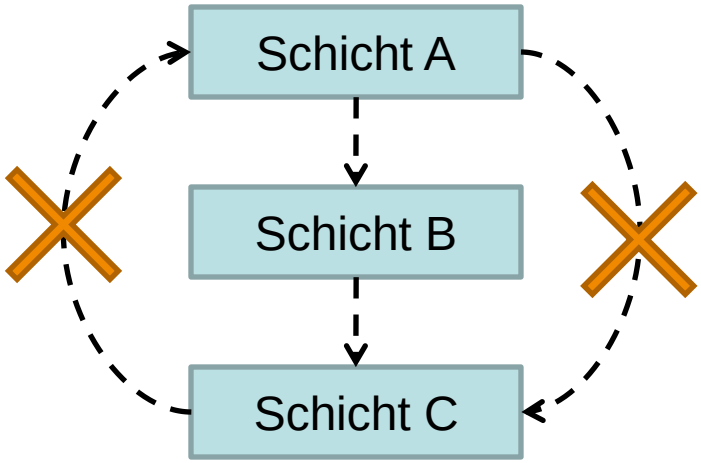


Schichten

- Eine Schicht (engl. layer, tier) ist ein Subsystem, welches Dienste für andere Schichten zur Verfügung stellt, mit folgenden Einschränkungen:
  - Eine Schicht nutzt nur Dienste von niedrigeren Schichten.
  - Eine Schicht nutzt keine höheren Schichten.
- Eine Schicht kann horizontal in mehrere, unabhängige Subsysteme, auch Partitionen genannt, aufgeteilt werden.
  - Partitionen bieten Dienste für andere Partitionen der gleichen Schicht an.

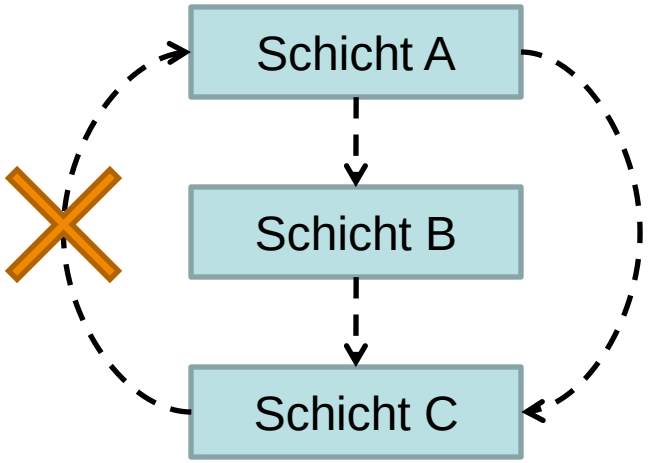
Intransparente Schichtenarchitektur

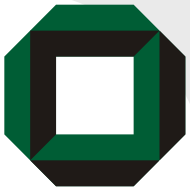
- Bei einer intransparenten Schichtenarchitektur werden undurchlässige Schichten (engl. opaque layers) verwendet.
- Eine Schicht in einer solchen Architektur kann nur auf Dienste der Schicht direkt unter ihr zugreifen.



Transparente Schichtenarchitektur

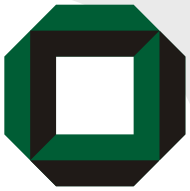
- Bei einer transparenten Schichtenarchitektur werden durchlässige Schichten (engl. transparent layers) verwendet.
- Eine Schicht in einer solchen Architektur kann auf Dienste jeder Schicht unter ihr zugreifen.





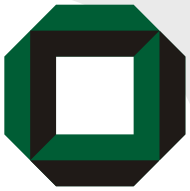
Schichtenarchitektur

- Schichtenarchitektur, Vorteile:
  - Übersichtliche Strukturierung in Abstraktionsebenen oder virtuelle Maschinen (die Schichten stellen abstrakte Maschinen dar).
  - Keine zu starke Einschränkung des Entwerfenden, da er neben einer strengen Hierarchie noch eine liberale Strukturierungsmöglichkeit innerhalb der Schichten besitzt.
  - Es werden die Wiederverwendbarkeit, die Änderbarkeit, die Wartbarkeit, die Portabilität und die Testbarkeit unterstützt. (Schichten können ausgetauscht, hinzugefügt, portiert, verbessert und wieder verwendet werden; Testen von unten oder von oben her).



Schichtenarchitektur

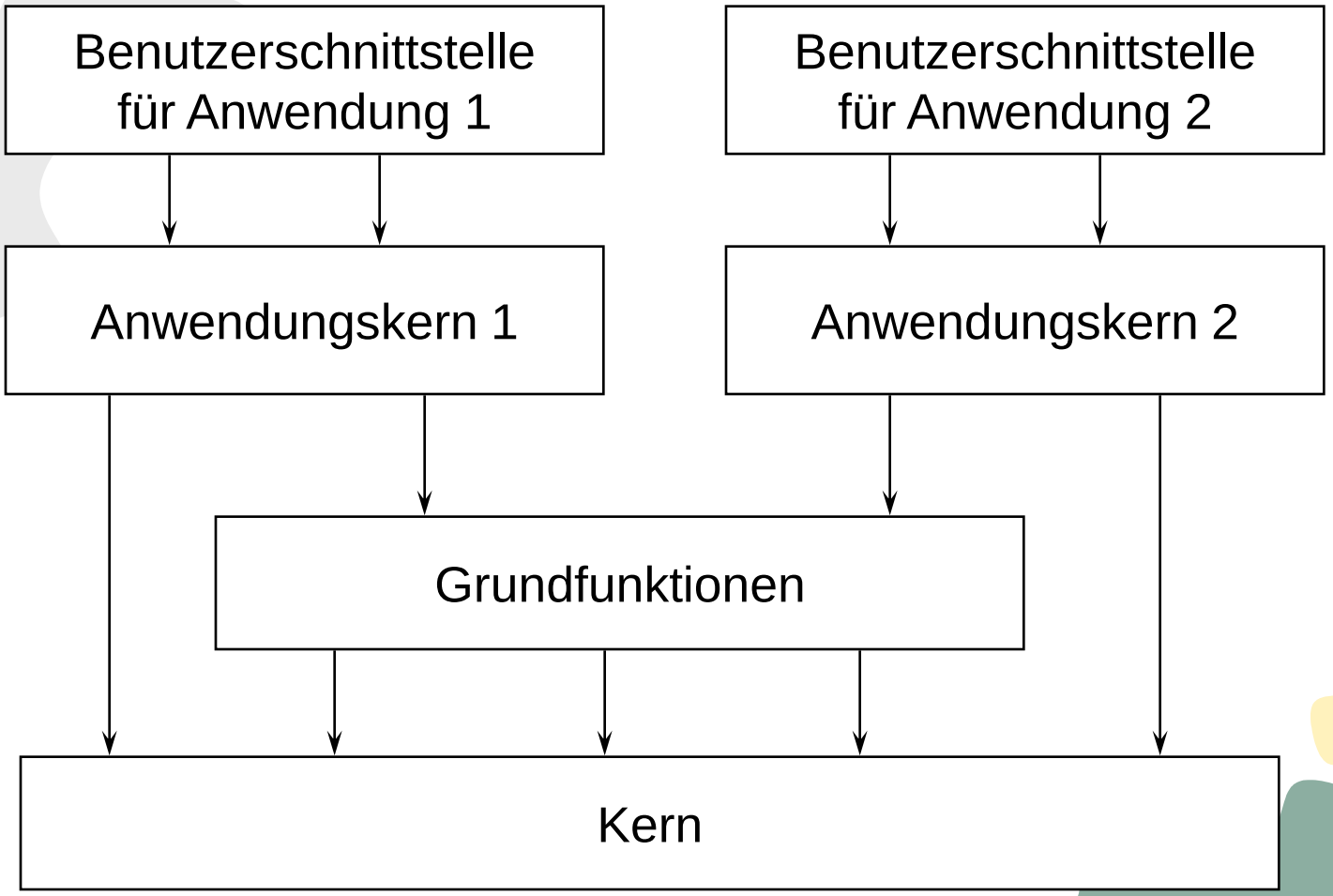
- Schichtenarchitektur, Nachteile/Probleme:
  - Bei intransparenter Schichtung kann es Effizienzverluste geben, da Aufrufe, die mehrere Schichten überschreiten, durch mehrere Schichten weitergereicht werden müssen, und mehrfache Parameterübergabe und Ergebnisrückgabe erfordern.
    - Dies gilt auch für Fehlermeldungen (mit catch/throw aber kein Problem mehr)
  - Eindeutig voneinander abgrenzbare Abstraktionsschichten lassen sich nicht immer definieren.
  - Innerhalb einer Schicht darf kein Chaos herrschen!

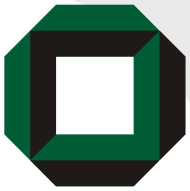


3-Schichten-Architektur (1)

- **Def.:** 3-Schichten-Architektur
  - Ein Architekturstil, bei welchem die Anwendung aus 3 hierarchisch geordneten Subsystemen besteht
    - Eine Benutzerschnittstelle, Anwendungskern und einem Datenbanksystem.
- **Def.:** 3-stufige Architektur (engl. 3-tier architecture)
  - Eine 3-Schichten-Architektur, bei welcher die Schichten auf unterschiedlichen Rechnern laufen.
    - Beispiel: Benutzerschicht läuft auf einem Klienten, Anwendungskern und Datenbank auf einem Dienstgeber

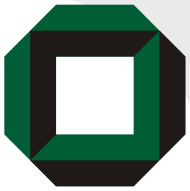
4-stufige Schichtenarchitektur





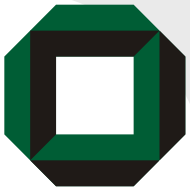
Noch eine 4-Schichten-Architektur

- 4-Schichten-Architekturen werden oft bei Webdiensten für elektronischen Handel verwendet:
  - Der Webbrowser stellt die Benutzerschnittstelle dar.
  - Der Webserver liefert statische HTML-Dateien.
  - Ein Anwendungs-Dienstgeber verwaltet Sitzungen (engl. Sessions) und erzeugt dynamischen HTML-Seiten.
  - Eine Datenbank verwaltet die Daten.



Weitere Beispiele

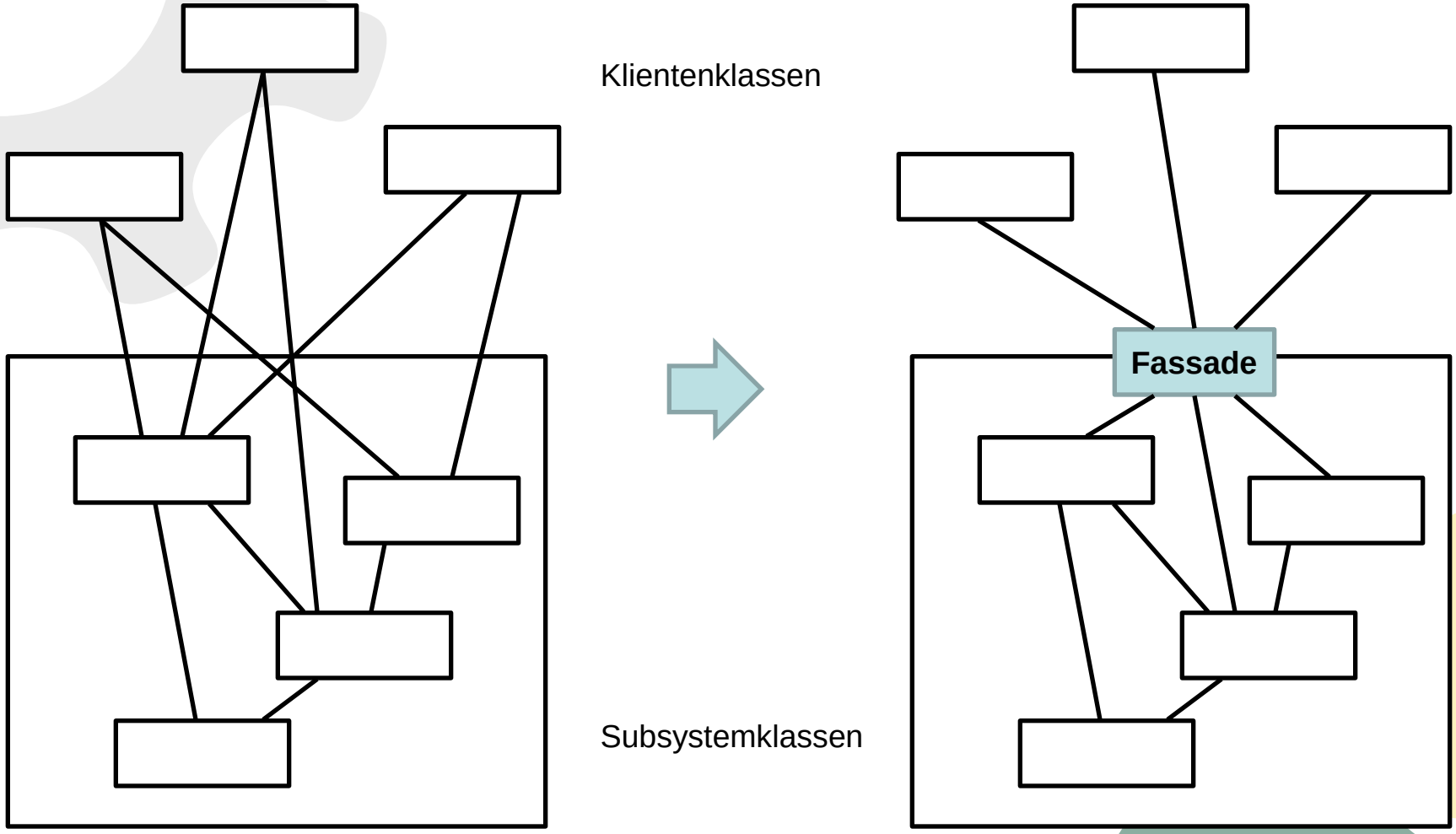
- Betriebssysteme (die wichtigsten Schichten sind (von unten): Prozessverwaltung, Speicherverwaltung, Dateiverwaltung, Kommunikation, graphische Benutzeroberfläche, Anwendungen)
- Protokolltürme bei der Datenfernübertragung
- Informationssysteme (auf Datenbanken aufbauend)



Schichtenarchitektur und das Entwurfsmuster Fassade (engl. Facade)

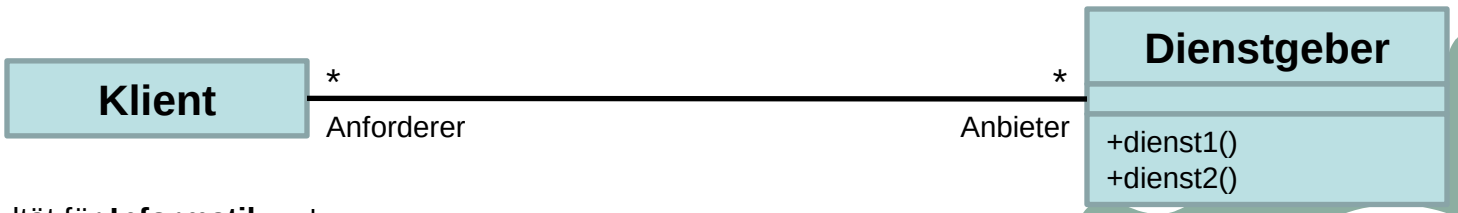
- Eine Schicht besteht oft aus einer Vielzahl von Elementen (Pakete, Module, Klassen, Objekte und Funktionen), die nicht alle an die darüberliegenden Schichten zur Verfügung gestellt werden sollen (weil zu komplex in der Bedienung, oder nicht nötig)
- Um eine bereinigte und vereinfachte Schnittstelle zur Verfügung zu stellen, setzt man eine Fassade ein.
- Die Fassade ist eine oder mehrere Klassen, die nur die zur Verfügung stehenden Elemente enthält und an die eigentlichen Elemente in der Schicht delegiert.

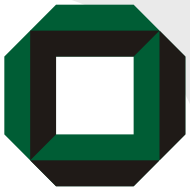
Schichtenarchitektur und das Entwurfsmuster Fassade



Klient/Dienstgeber (engl. Client/Server)

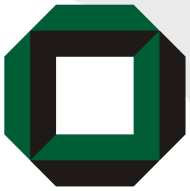
- Ein oder mehrere Dienstgeber bieten Dienste für andere Subsysteme, Klienten genannt, an.
- Jeder Klient ruft eine Funktion des Dienstgebers auf, welcher den gewünschten Dienst ausführt und das Ergebnis zurückliefert.
  - Dazu muss der Klient die Schnittstelle des Dienstgebers kennen.
  - Umgekehrt muss der Dienstgeber die Schnittstelle des Klienten nicht kennen.
- Ein Beispiel einer 2-stufigen, verteilten Architektur





Klient/Dienstgeber

- Wird oft beim Entwurf von Datenbank-Systemen verwendet:
  - Front-End: Benutzeroberfläche für den Benutzer (Klient)
  - Back-End: Datenbankzugriff und Manipulation (Dienstgeber)
- Funktionen, die der Klient ausführt:
  - Eingaben des Benutzers entgegennehmen
  - Vorverarbeitung der Eingaben
- Funktionen, die der Dienstgeber ausführt:
  - Datenverwaltung
  - Datenintegrität und -Konsistenz
  - Sicherheit
- Klient und Dienstgeber laufen i.d.R. auf unterschiedlichen Rechnern, aber nicht unbedingt.

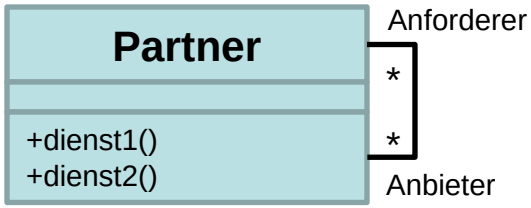


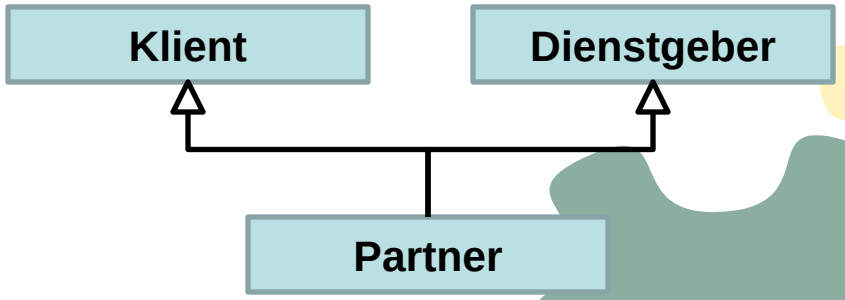
Klient/Dienstgeber: Beispiel

- Dateiübertragung auf einen FTP-Server:
  - Der Klient (bspw. ein FTP-Programm wie Filezilla) initiiert das Herauf- bzw. Herunterladen einer Datei.
  - Der Dienstgeber reagiert auf die Anfrage des Klienten und empfängt bzw. sendet die Datei.

Partnernetze (engl. Peer-to-Peer Networks)

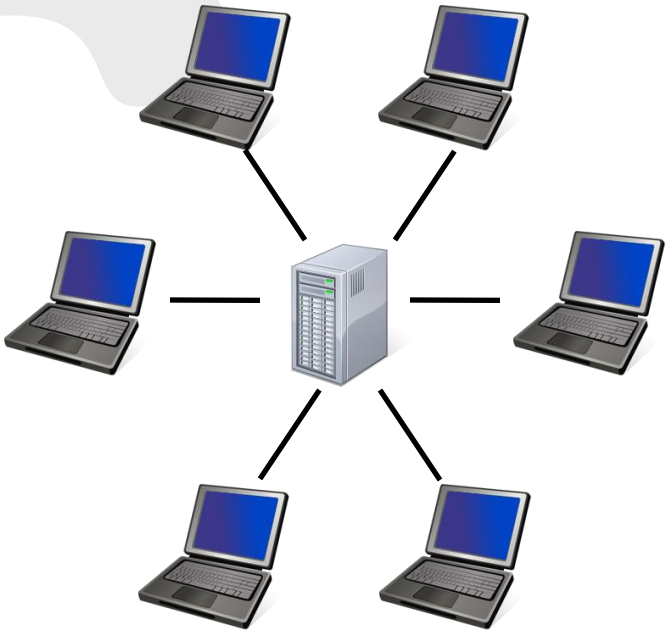
- Verallgemeinerung des Klient/ Dienstgeber-Architekturstils.
- Bei einem Partnernetz sind alle Subsysteme gleichberechtigt („Peer“ bedeutet „Gleichgestellter“ , „Gleichaltriger“, s.a. „peer pressure“). Partner laufen auf unterschiedlichen Rechnern.
- Vereinfacht dargestellt:



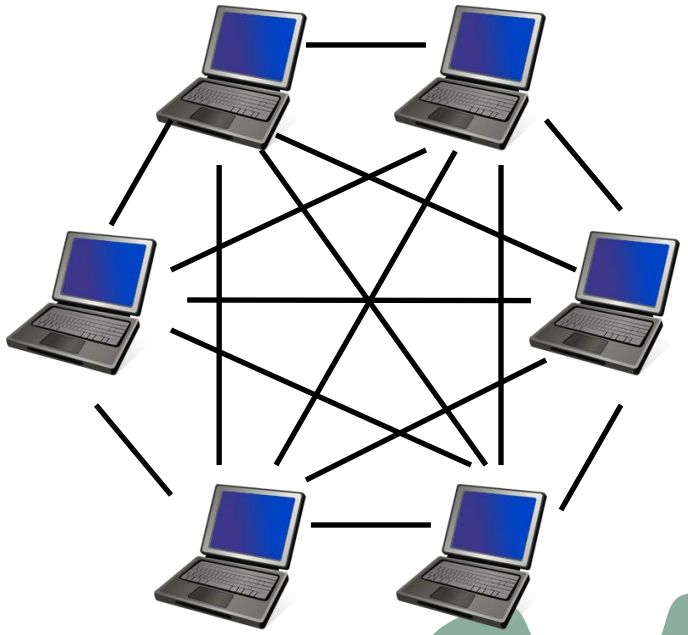


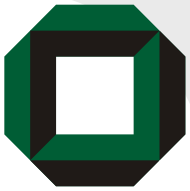
Partnernetz vs. Klient/Dienstgeber

Klient/Dienstgeber



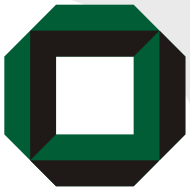
Partnernetz





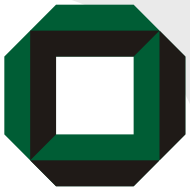
Partnernetz– Eigenschaften (1)

- Rollensymmetrie:
  - Jeder Partner ist sowohl Klient als auch Dienstgeber.
- Dezentralisierung:
  - Es gibt keine zentrale Koordination und keine zentrale Datenbasis. Jeder Partner kennt i.d.R. nur eine Untermenge der übrigen Partner (seine „Nachbarschaft“).
- Selbstorganisation:
  - Das Gesamtverhalten des Systems setzt sich aus der Interaktion zwischen dein einzelnen Partnern zusammen.



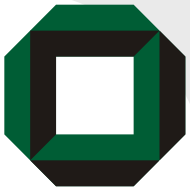
Partnernetze– Eigenschaften (2)

- Autonomie:
  - Partner treffen Entscheidungen autonom und Verhalten sich autonom.
- Zuverlässigkeit:
  - Partner sind unzuverlässig (z.B. nicht immer angeschaltet). D.h. es müssen Mechanismen gefunden werden, welche diese Unzuverlässigkeit kompensieren.
- Verfügbarkeit:
  - Alle im System gespeicherten Daten müssen redundant verfügbar sein, wegen Unzuverlässigkeit (mancher) Partner.



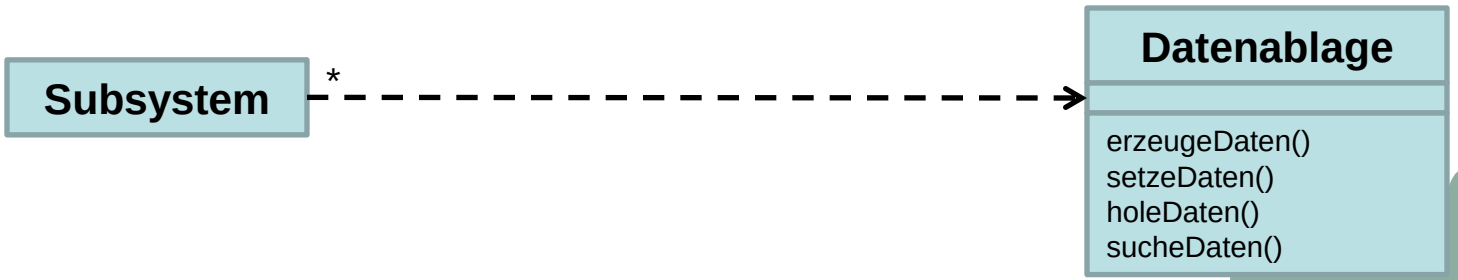
Partnernetz: Beispiel

- Auf Dienstebene: Austausch von Dateien in einem Partnernetz (bspw. Bittorrent):
  - Ein Partner ist sowohl Klient als auch Dienstgeber:
    - Er kann Dateien von anderen Partnern anfordern (→ Klient)
    - Er kann anderen Partnern Dateien (und weitere Dienste) anbieten (→ Dienstgeber)
- Auf Netzwerkebene: TCP/IP, DNS:
  - Anfragen werden anwendungsunabhängig über das physikalische Netzwerk verteilt.

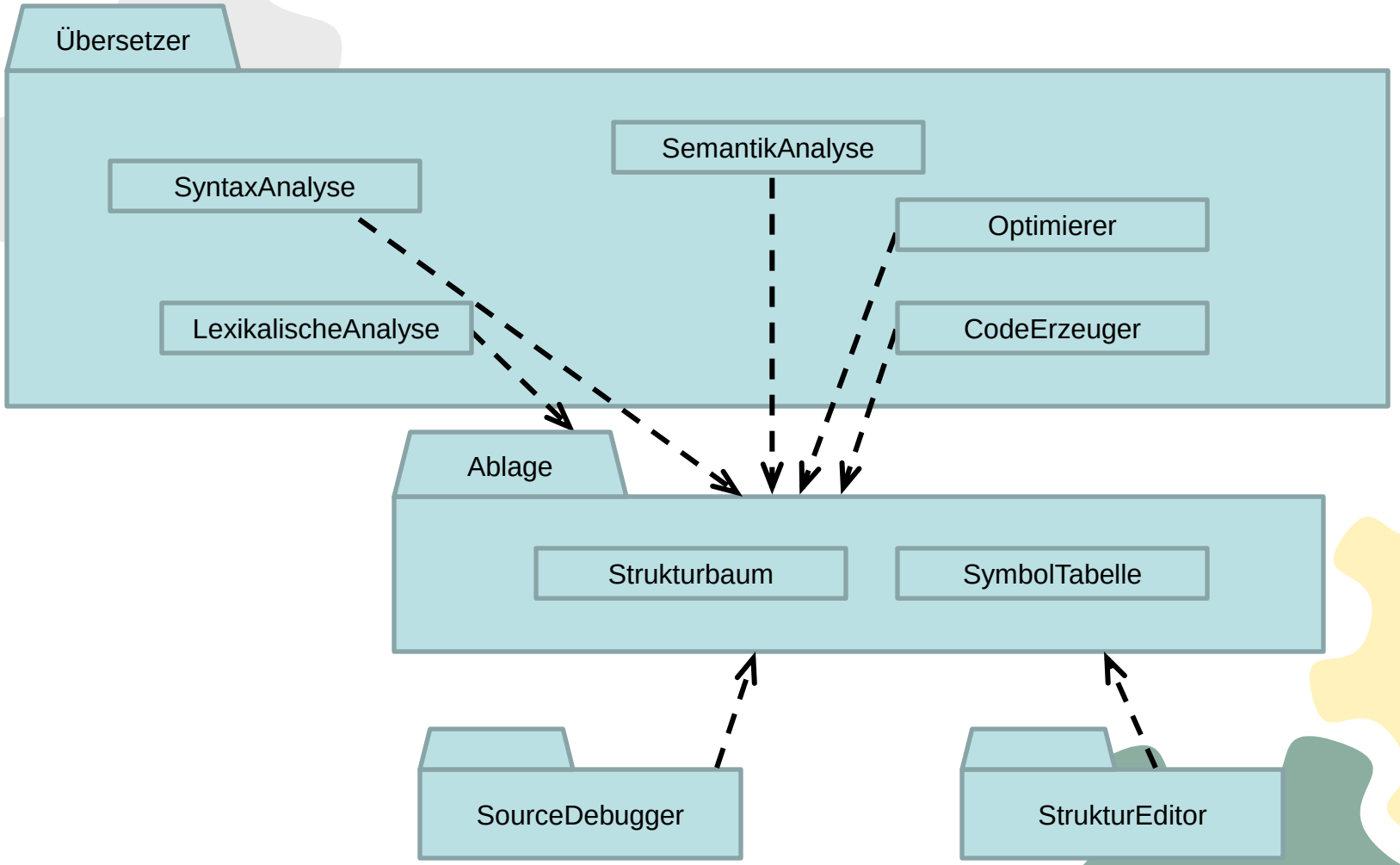


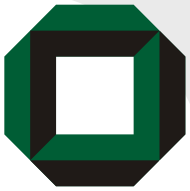
Datenablage (engl. Repository)

- Subsysteme modifizieren Daten von einer zentralen Datenstruktur, der Datenablage.
- Subsysteme sind lose gekoppelt und interagieren nur über die Datenablage.
- Subsysteme können parallel oder hintereinander auf die Datenablage zugreifen. In parallelem Fall ist Synchronisation nötig (siehe Datenbanken).



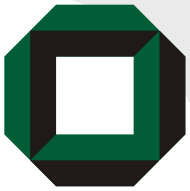
Datenablage: Beispiel (1)





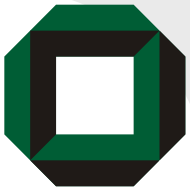
Datenablage: Beispiel (2)

- Erklärung:
  - Die drei Werkzeuge werden vom Benutzer in beliebiger Reihenfolge, auch parallel ausgeführt.
  - Die Datenablage stellt sicher, dass gleichzeitige Zugriffe auf gemeinsame Daten geordnet werden.
  - Der Zustand der Datenablage kann Einfluss auf die Ausführung der verschiedenen Werkzeuge haben.
    - Z.B. kann das Bearbeiten von Dateien mit dem Editor während eines Übersetzvorgangs verboten werden.



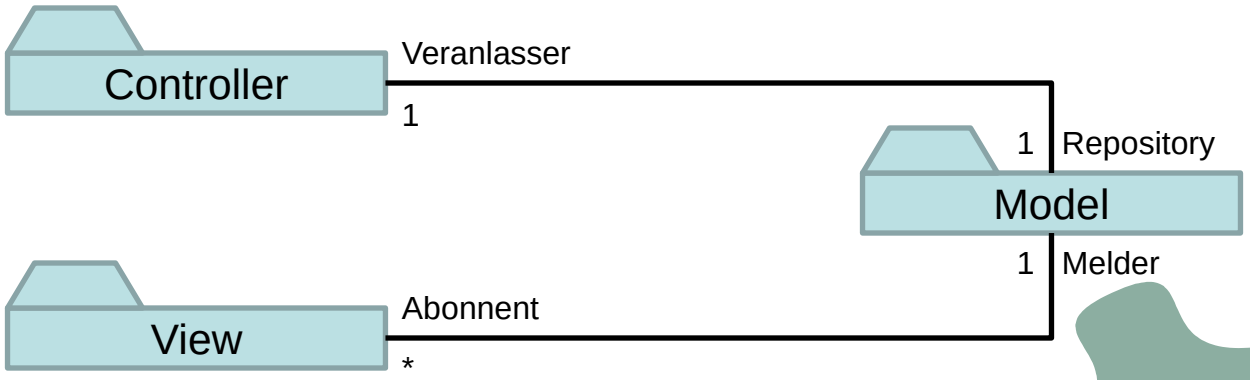
Model/View/Controller (MVC)

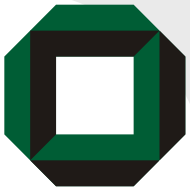
- **Problem:** Angenommen, ein geg. System hat eine hohe Kopplung, so dass Änderungen an der Benutzerschnittstelle zu Änderungen an den Daten-Objekten.
  - Die Benutzerschnittstelle kann nicht neu implementiert werden, ohne die Darstellung der Daten-Objekte zu ändern.
  - Die Daten-Objekte können nicht neu organisiert werden, ohne die Benutzerschnittstelle zu ändern.
- **Lösung:** Der Architekturstil „Model/View/Controller“, welcher Daten von deren Darstellung trennt.
  - Das Subsystem zur Darstellung der Daten wird View (Sicht) genannt.
  - Das Subsystem zur Datenspeicherung heißt Modell.
  - Das Subsystem zur Interaktion zwischen Sicht und Modell wird Controller (Steuerung) genannt.



Model/View/Controller

- **Model**: Verantwortlich für anwendungsspezifisches Daten.
- **View**: Verantwortlich für die Darstellung der Objekte der Anwendung.
- **Controller**: Verantwortlich für die Verarbeitung der Interaktionen des Benutzers und das Melden von Änderungen der Modelldaten an die Sicht(en).



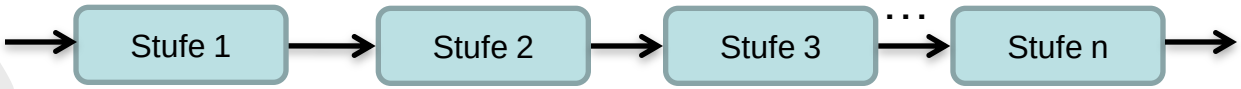


Vergleich: MVC vs. 3-Schichten-Architektur

- Die MVC-Architektur ist nicht hierarchisch:
  - Sicht sendet Aktualisierungen an Steuerung
  - Steuerung aktualisiert Modell
  - Modell aktualisiert Sicht

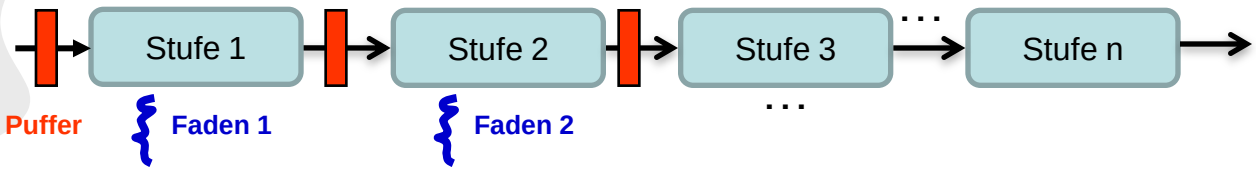
  ➔„Dreiecksbeziehung“
- Die 3-Schichten-Architektur ist hierarchisch:
  - Die Darstellungsschicht kommuniziert nicht direkt mit der Datenschicht
  - Jede Art von Kommunikation muss durch die mittlere Schicht erfolgen.

Fließband (engl. Pipeline, Pipe and Filter)



- Jede Stufe oder Filter ist ein eigenständig ablaufender Prozess oder Faden, mit eigenem Befehlszähler
- Daten fließen durch das Fließband, wobei jede Stufe von der vorigen Daten empfängt, sie verarbeitet, und an die nächste weiterreicht.
- Aufeinanderfolgende Stufen sind mit einem größenbeschränkten Puffer verbunden, um Geschwindigkeitsschwankungen auszugleichen.

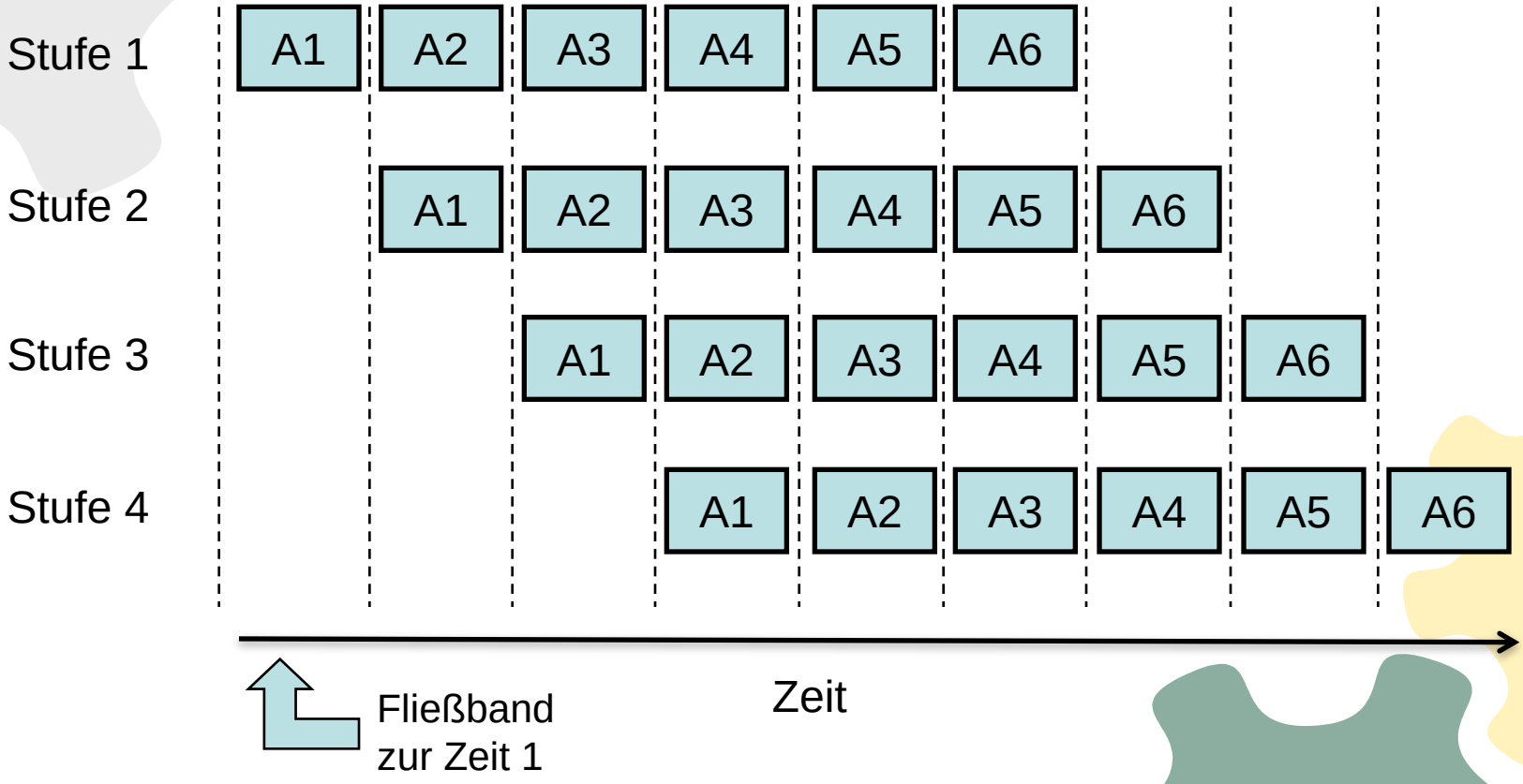
Fließband



- Bei Parallelrechnern können die einzelnen Stufen echt parallel ablaufen und damit zu einer Beschleunigung führen.
- Bei sequentiellen Rechnern werden die einzelnen Prozesse abwechselnd ausgeführt.

Fließband

- Verarbeitungsprinzip



Beispiele für Fließband

- Ein Beispiel für diesen Architekturstil ist die Unix-Shell.

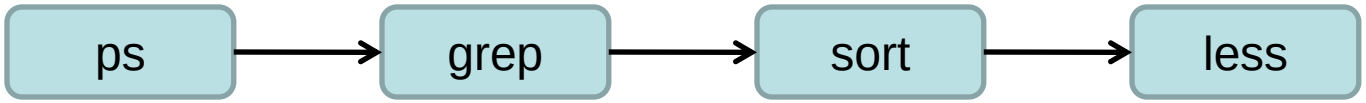
```
ps auxww | grep hans | sort | less
```

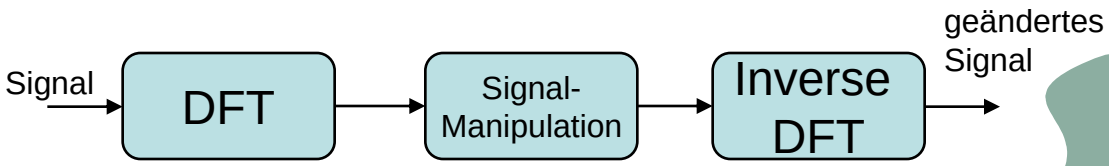
Zeigt alle laufenden Prozesse an

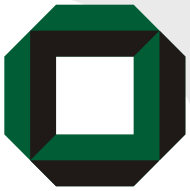
Filtert Zeilen mit „hans“ heraus

Sortiert Eingabe

Erlaubt blättern in Eingabe

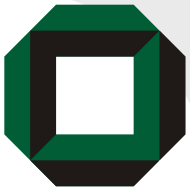






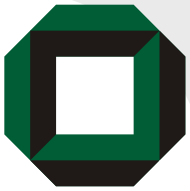
Fließband: Anwendbarkeit

- Gut geeignet für Verarbeiten von Datenströmen, z.B. für Videocodierung –Decodierung, Übersetzer, Stapelverarbeitung.
- Für gute Leistung auf Parallelrechnern sollten die einzelnen Stufen etwa gleich schnell laufen (unerheblich für Einzelprozessoren)



Rahmenarchitektur (engl. Framework)

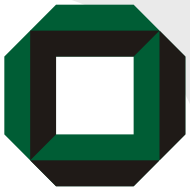
- Bietet ein (nahezu) vollständiges Programm, das durch Einfüllen geplanter „Lücken” oder Erweiterungspunkten erweitert werden kann. Es enthält die vollständige Anwendungslogik, meistens sogar ein komplettes Hauptprogramm. Von einigen der Klassen in dem Programm können Benutzer Unterklassen bilden und dabei Methoden überschreiben oder vordefinierte abstrakte Methoden implementieren.
  - Das Rahmenprogramm sieht vor, dass die vom Benutzer gelieferten Erweiterungen richtig aufgerufen werden.
  - Die Erweiterungen werden auch Einschübe (engl. Plug-ins) bezeichnet.



Rahmenarchitektur (engl. Framework)

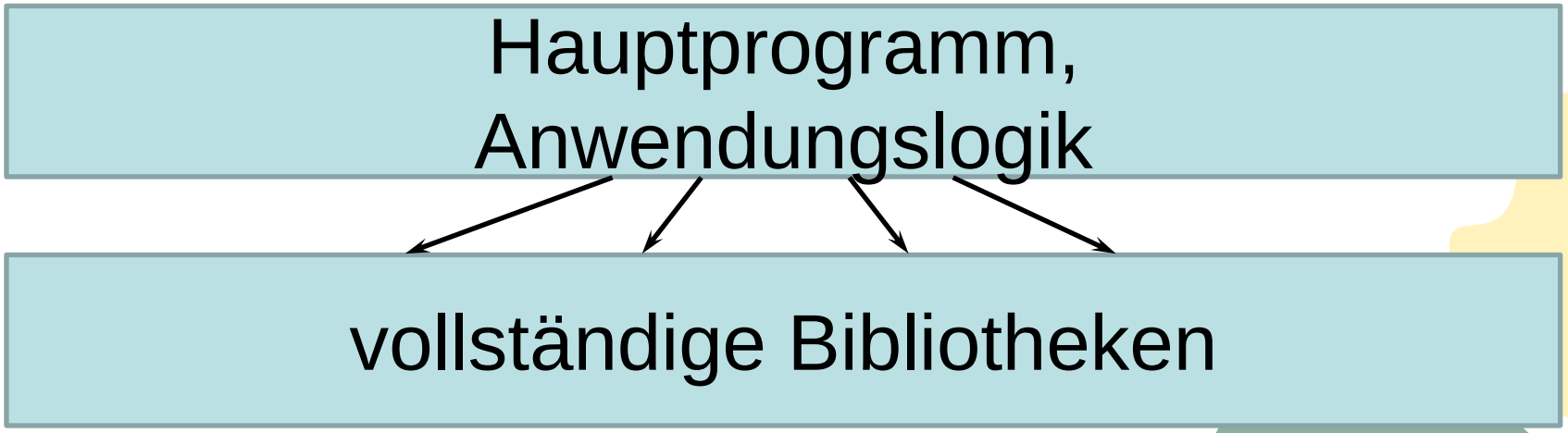
Eine Rahmenarchitektur (auch Rahmenprogramm) bietet ein (nahezu) vollständiges Programm, das durch Einfüllen geplanter „Lücken” oder Erweiterungspunkten erweitert werden kann. Es enthält die vollständige Anwendungslogik, meistens sogar ein komplettes Hauptprogramm.

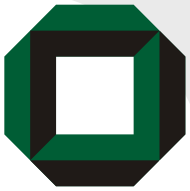
Die Erweiterungspunkte sind Klassen oder Schnittstellen. Von diesen können Benutzer Unterklassen bilden und dabei Methoden überschreiben oder vordefinierte abstrakte Methoden implementieren. Das Rahmenprogramm bewirkt, dass die vom Benutzer gelieferten Erweiterungen richtig aufgerufen werden.
Die Erweiterungen werden auch Einschübe (engl. Plug-ins) bezeichnet.



Rahmenarchitektur: Struktur (1)

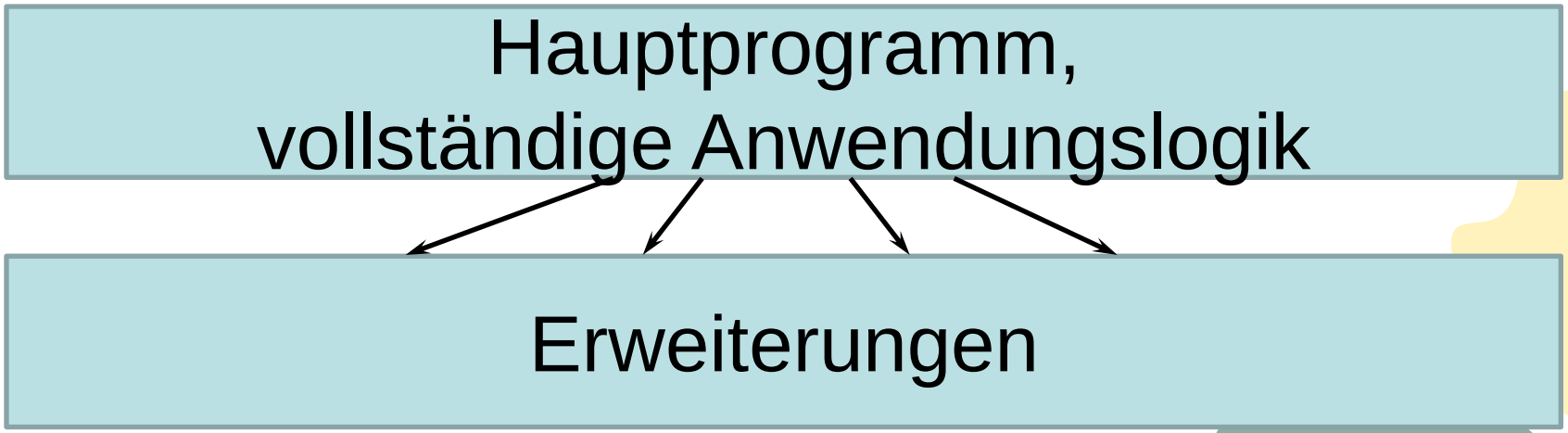
- Herkömmliche Systemstruktur:
  - Hersteller liefert Bibliotheken
  - Benutzer schreibt Hauptprogramm und Anwendungslogik

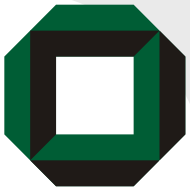




Rahmenarchitektur: Struktur (2)

- Rahmenarchitektur:
  - Ein Rahmenprogramm befolgt das „Hollywood-Prinzip”: „Don’t call us - we’ll call you”.
  - Das Hauptprogramm besteht bereits und ruft die Erweiterungen der Benutzer auf.

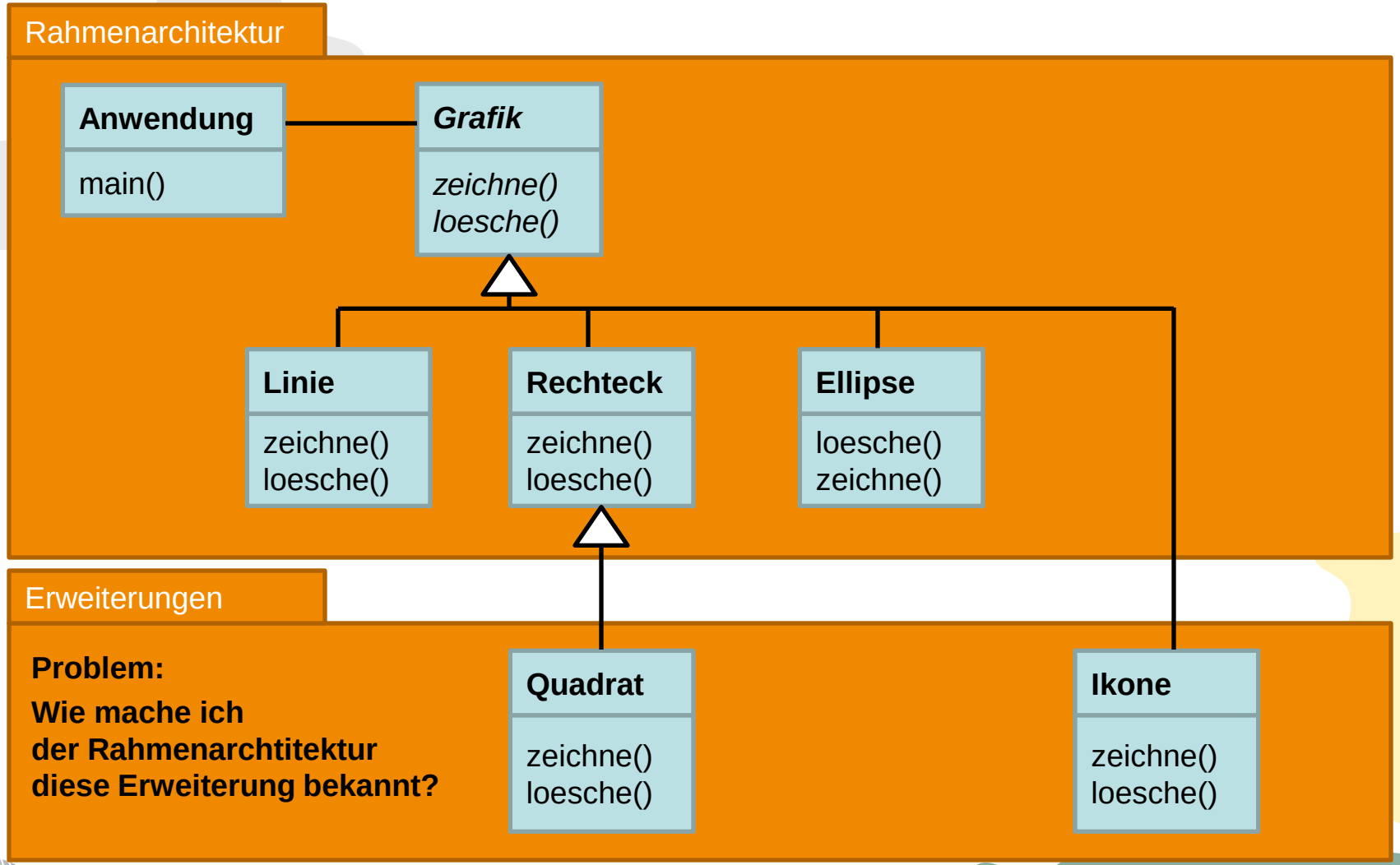


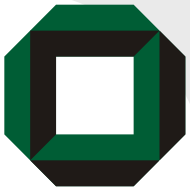


Rahmenarchitektur: Beispiel (1)

- Ein Zeichen-Rahmenprogramm sieht eine Klasse Grafik mit einigen Unterklassen (z.B. Linie, Rechteck, Ellipse, usw.) vor.
- Der Benutzer darf neue Unterklassen von Grafik und seinen Unterklassen bilden, z.B. Quadrat oder Ikone, muss dazu aber eine Methode zeichne bereitstellen.
- Das Rahmenprogramm sorgt dann dafür, dass Objekte der neuen Klassen richtig erzeugt, auf dem Zeichenbrett positioniert, verschoben, gesichert, usw. werden können.

Rahmenarchitektur: Beispiel (2)

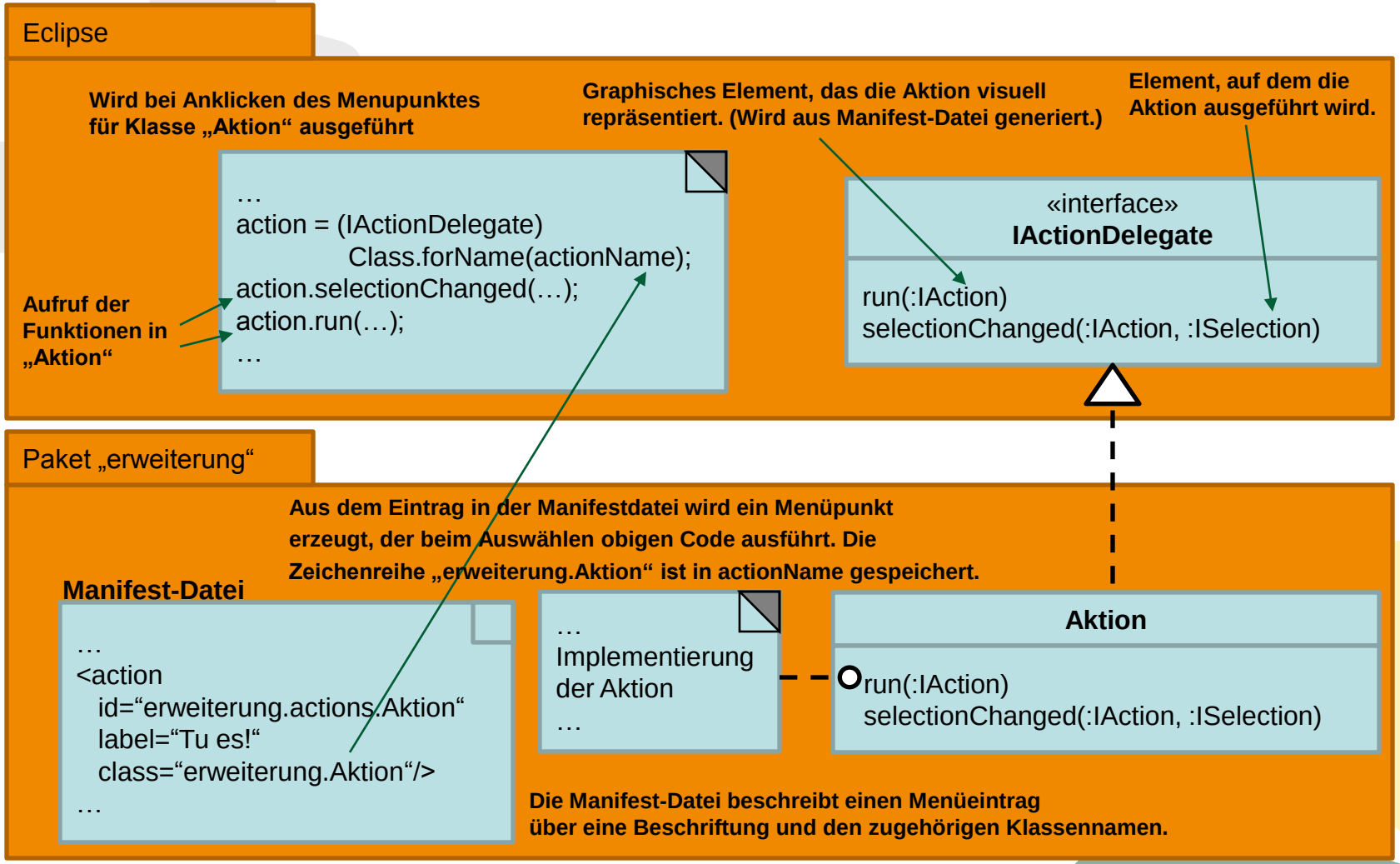


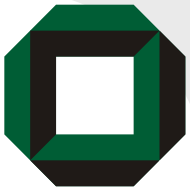


Rahmenarchitektur: Beispiel (3)

- **Eclipse:** Quelloffene Entwicklungsplattform
- Besteht aus einem relativ kleinen Kern mit einer großen Zahl an Erweiterungspunkten für Einschübe (Plug-Ins), die wiederum weitere Erweiterungspunkte bieten können.
- Große Teile der Standard-Distribution von Eclipse bestehen aus Einschüben.
- Einschübe werden als Jar-Dateien gespeichert, deren Manifest-Datei von dem Eclipse-Kern gelesen wird, um die Bezeichner der Erweiterungsklassen zu erhalten.

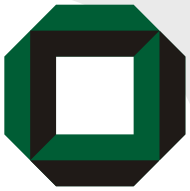
Rahmenarchitektur: Beispiel (4)





Rahmenarchitektur: Beispiel (5)

- Die Manifest-Datei enthält die Namen der Erweiterungsklassen, Bezeichnung der gewünschten Menüeinträge, und weitere Einzelheiten.
- Elemente wie Menüeinträge und Knöpfe in der Entwicklungsumgebung werden daraus generiert.
  - Daher ist die Darstellung dieser Elemente möglich ohne die Klassen sofort laden zu müssen. (→Vermeidet lange Wartezeiten beim Start)
- Wird eine neue Methode über einen Menüeintrag aufgerufen, dann lädt Eclipse die Klasse über class.forName(), initialisiert sie und ruft die in der Schnittstelle definierte Methode auf.
  - Bei der Initialisierung (selectionChanged) wird der Kontext des Methodenaufrufs übergeben. (Editor, Selektion, Projekt, …)



Rahmenarchitektur: Anwendbarkeit

- Wenn eine Grundversion der Anwendung schon funktionsfähig sein soll.
- Wenn Erweiterungen möglich sein sollen, die sich konsistent verhalten (Anwendungslogik im Rahmenprogramm).
- Wenn komplexe Anwendungslogik nicht neu programmiert werden soll.
- Die Entwurfsmuster Strategie, Fabrikmethode, abstrakte Fabrik und Schablonenmethode werden häufig in Rahmenarchitekturen benötigt (siehe nächster Abschnitt).