

# Software-Technologie: Stand der Kunst und Herausforderungen

O.Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Pree

Software & Systems Research Group

Universität Salzburg

[cs.uni-salzburg.at](http://cs.uni-salzburg.at)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

# Kontext

- Das Phänomen Software
- Wie kann Software ingenieurmäßig entwickelt werden?
- Softwaretechnik – Quo vadis?

# Das Phänomen Software

# Die Universalmaschine Computer macht Software allgegenwärtig



Flugzeug-/Raketensteuerungen



ca. 70 Prozessoren  
im Auto



# Qualität ist wesentlich schlechter als bei anderen Produkten

- **Softwarefehler/-mängel** mit drastischen Auswirkungen:
  - | Y2K, €-Umstellung
  - | fehlerhafte Finanztransaktionen
  - | Abstürze (zB Ariane: \$ 800 Mio.)
  - | . . .



# Was ist besonders an Software?

Die Probleme bei der Herstellung von Software resultieren aus der Komplexität der zu realisierenden Produkte

- **Spezifikation der Anforderungen** ← Prototyping
- **Beherrschung der Komplexität** ← Programmiermodelle
- **Wiederverwendung/PlugIns, Änderbarkeit und Erweiterbarkeit** ← Entwurfsmuster, Frameworks
- **Automatisierung im Herstellungsprozeß**
- **Portabilität**
- **Dokumentation**
- **Produktergonomie (Mensch-Computer-Schnittstelle)** ← Psychologie (zB Piaget)
- **Projektorganisation u. -kontrolle**
- **Qualitätssicherung und -bewertung**
- **Personenunabhängigkeit**
- **Kostenabschätzung**

# Beispiel: Problem der exakten Spezifikation

# Eine exakte Spezifikation ist oft unpraktikabel

geg.:  $n \geq 3$ ,

$$L: N_n \rightarrow N$$

ges.: Ein Programm P, sodass

$a: N_3 \xrightarrow{\text{inj}} N_n$ , sodass

$$\begin{array}{ccc} \wedge & \wedge & L(a_i) \geq L(a_j) \\ 1 \leq i \leq 3 & j \in N_n \setminus \cup \{a_k\} & \\ & 1 \leq k \leq j & \end{array}$$

... im Vergleich zur nicht exakten verbalen Spezifikation

Gegeben ist eine Liste mit mindestens drei positiven Zahlen.

Gesucht ist ein Programm  $P$ , das die Indizes der drei größten Elemente der Liste liefert.

# Meisterung der Komplexität

Bei klassischen Ingenieurdisziplinen gilt:

- Schlechte Qualität läßt sich kaum verbergen
  - | Tür zu einem Raum geht nicht gut auf
  - | unnötige Schnörkel fallen auf
    - „5. Rad am Wagen“
- Die Ressourcen sind beschränkt
  - | ingenieurmäßiges Herangehen bedeutet, unter den gegebenen Rahmenbedingungen zu optimieren

Bei Software hingegen ist schlechte Qualität nicht unmittelbar sichtbar

- schlechte Strukturierung
  - | „Spaghetti“-Programmcode:  
Radwechsel => Motor funktioniert nicht mehr
  - | replizierter Programmcode
- kaum Wiederverwendung
  - | das Rad wird immer neu erfunden

# Ingenieurmäßiges Vorgehen scheint sich nicht auszuzahlen

- Hardware-Ressourcen werden nach *Moore's Law* potenter; der gedankenlose Umgang damit führt zu
  - unnötiger Komplexität
  - nicht mehr verstehbaren Artefakten

OberonOS (ETH ZH)  
30.000 Zeilen  
Programmcode

4,1 cm

27,5 m

Windows XP:  
20,000.000 (!!)  
Zeilen  
Programmcode

# Wie kann Software ingenieurmäßig entwickelt werden?

# von Einzelteilen zu Komponenten

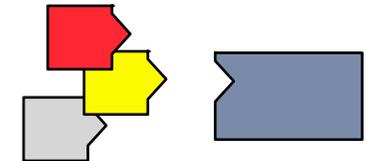
## 50er Jahre

- Maschinen-/Assembler-Programme: auf bestimmten Prozessor zugeschnitten

## 60er/70er Jahre

- höhere Programmiersprachen (wie Pascal, C)
- **Anweisungen können zu Funktionen/Prozeduren zusammengefasst werden**

→ Einzelteile, Schrauben, etc.

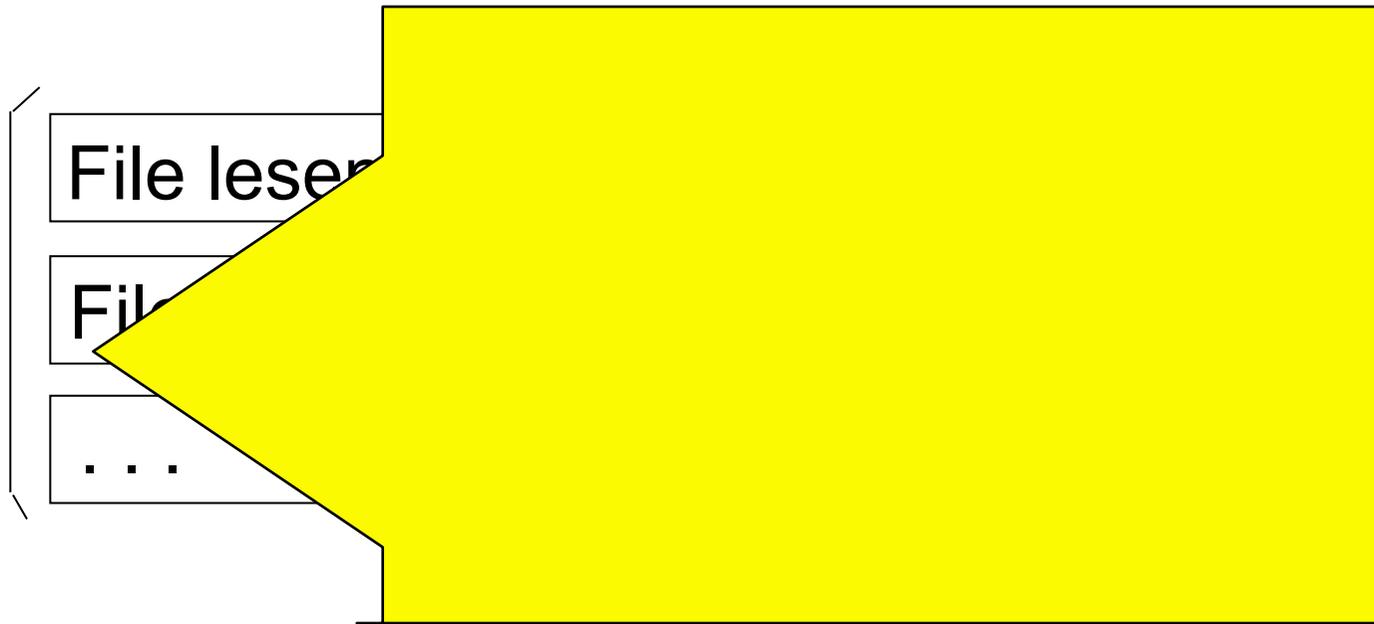


## 80er/90er Jahre

- **Funktionen/Prozeduren werden zu Modulen zusammengefasst** (Modula, Oberon, C++, Java, C#)
- Software-Komponenten

# Beispiel: Komponente File-Handler

einfache Schnittstelle



**versteckte Implementierungsdetails:**

Zugriff auf Festplatte

Aufsplitten des Inhalts eines Files

etc.

# Architektur-Patterns



# Software-Patterns

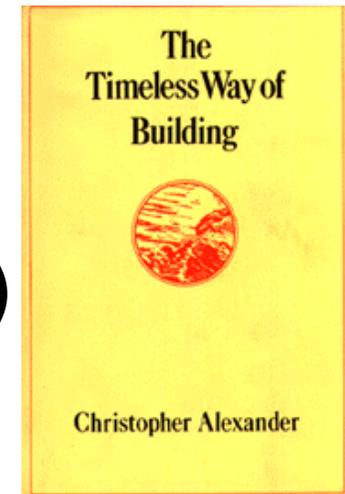
# The Timeless Way of Building

**Christopher Alexander**, Professor of Architecture, Univ. of California, Berkeley:

1979 erschienene Bücher:

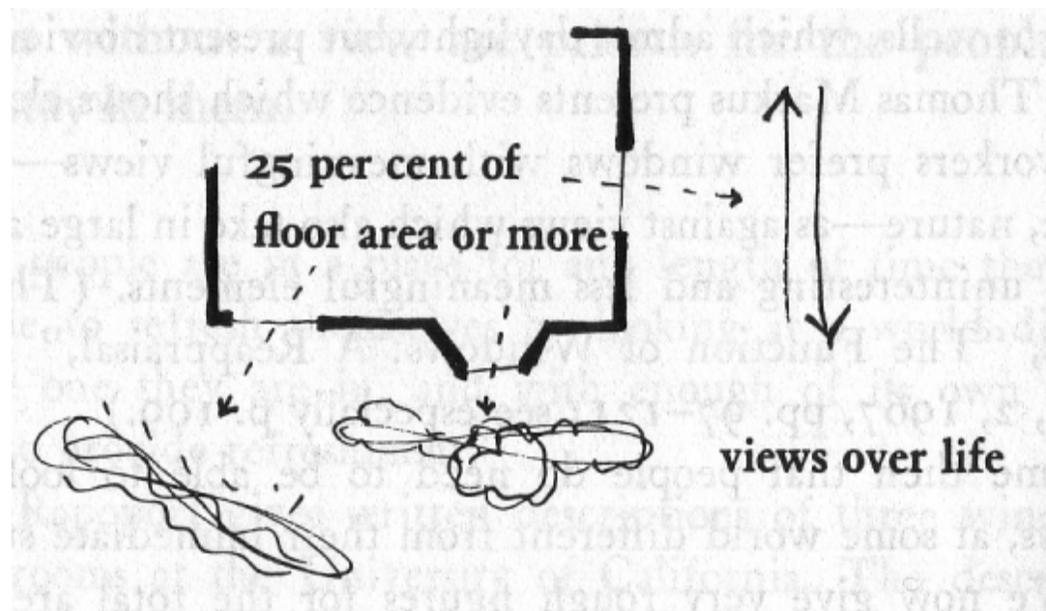
**The Timeless Way of Building**  
**A Pattern Language (253 Patterns)**

*Quality without a name*



1991 von der Software-Community entdeckt

# Beispiel: Windows Overlooking Life

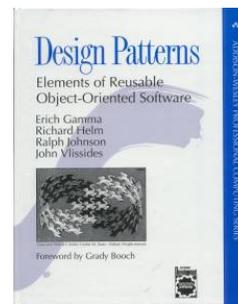


# Beispiele für Software Patterns

# Wie können SW-PlugIn-Architekturen geschaffen werden?

## Beschrieben in Architektur-Handbüchern (1995):

- E. Gamma, R. Helm, R. Johnson, J. Vlissides:  
Design Patterns: Elements of Reusable Software



- W. Pree:  
Design Patterns for Object-Oriented Software Development



## Was sind PlugIn-Architekturen?

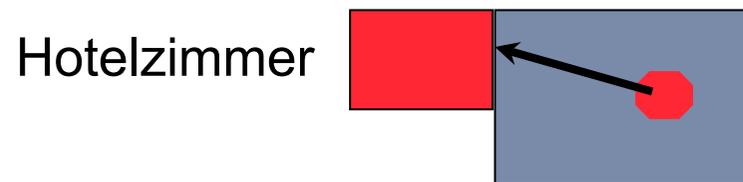
- Küchenmaschine: durch Einstecken einer Komponente wird das vorhandene „**PlugIn-System**“ zum fertigen Mixer oder Fleischwolf
- neue Automodelle gleichen meist „im Kern“ (Chassis, Getriebe, Motorpalette) den Vorgängermodellen

# SW-Beispiele

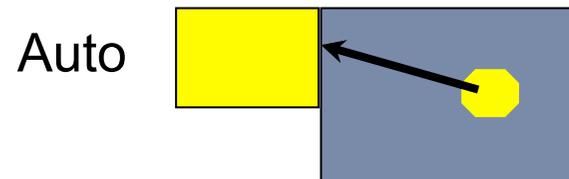
- Einwegsoftware:
  - Hotelreservierungssystem
  - Autovermietungssystem
  - Schiverleihsystem
  - Motorradverleihsystem
  - etc.
- PlugIn-Architektur:
  - Reservierungssystem  
(Mietgegenstand)

# Einwegsoftware

**Abhängigkeit zwischen den Komponenten ist im Programm Quelltext:**

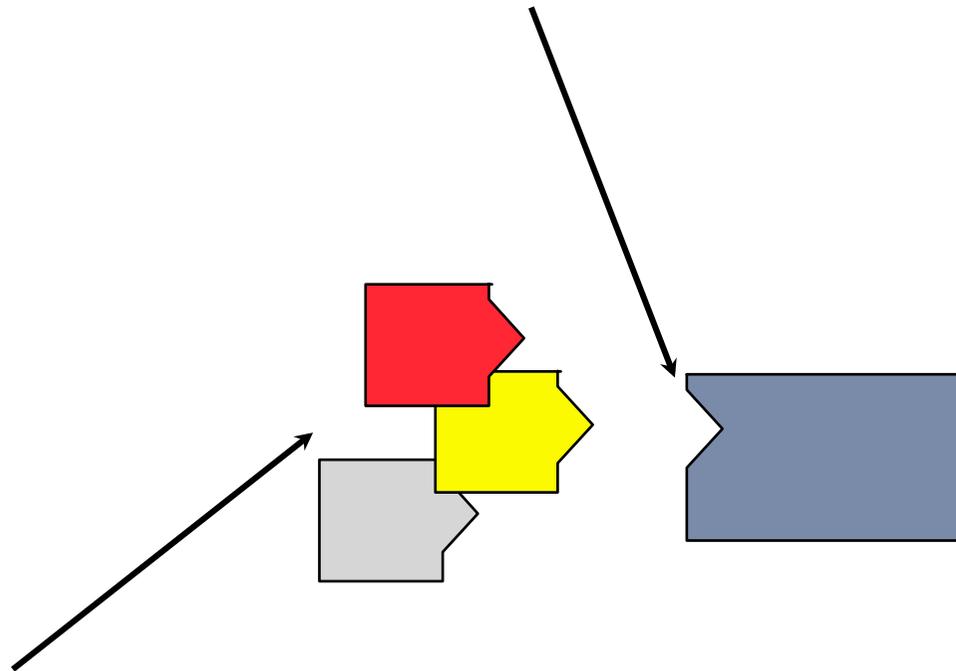


**Kopplung mit einer anderen Komponente erfordert Änderungen:**



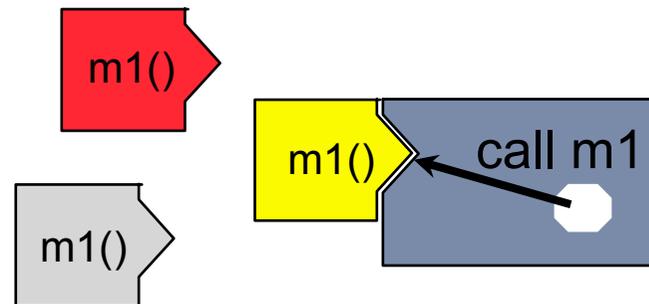
# Pattern: PlugIn-Architekturen erfordern die Definition von „Steckern“

## Stecker „Mietgegenstand“



Stecker-kompatible  
Komponenten

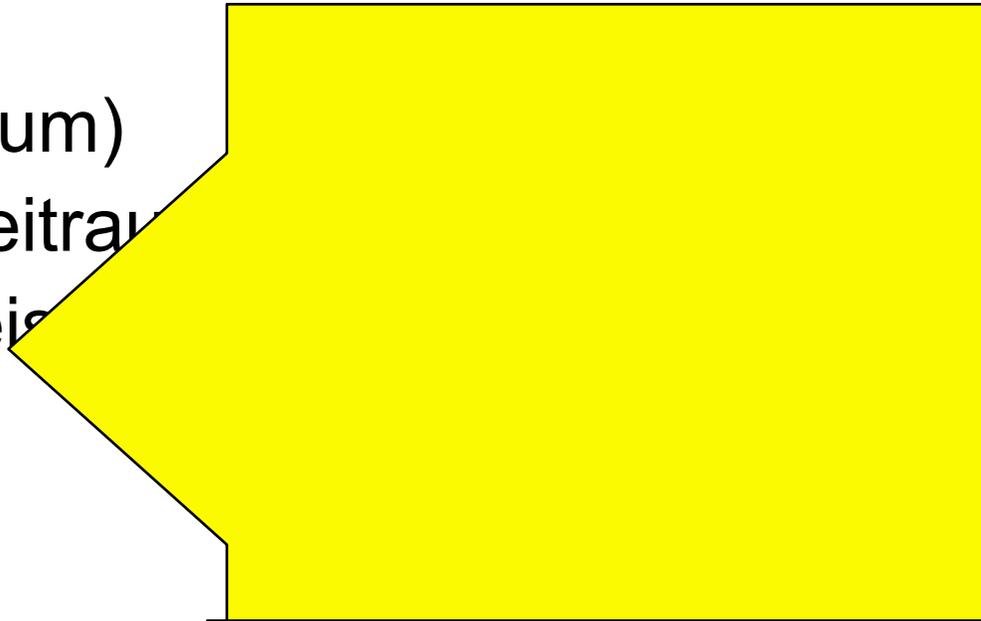
# sogenannte dynamische Bindung von Aufrufen macht Änderungen im Source-Code obsolet

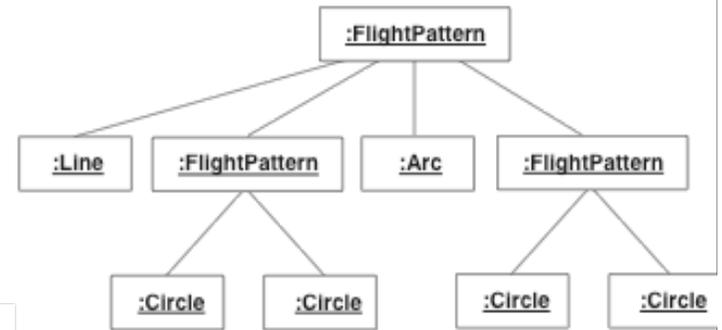
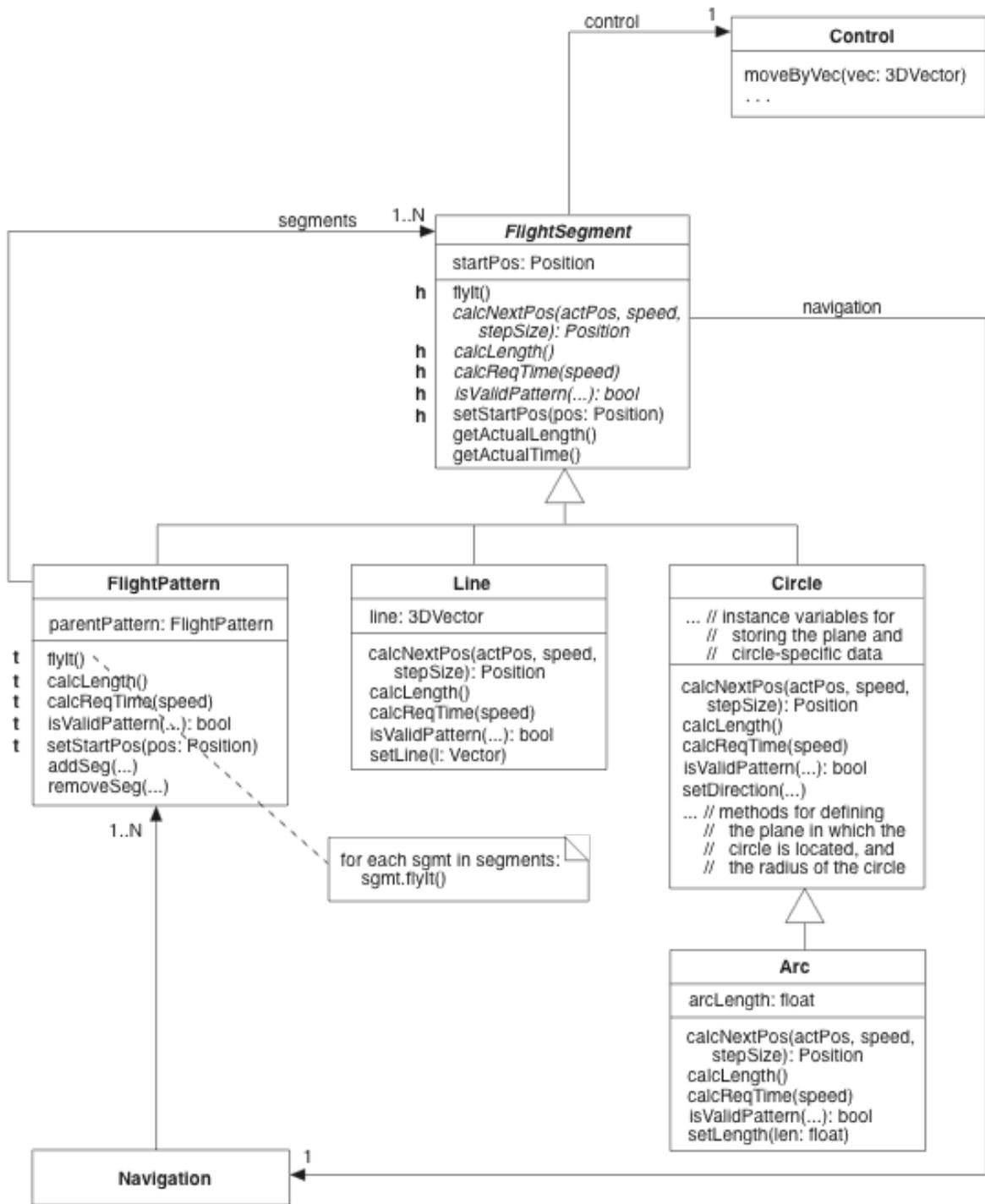


# „Stecker“ Mietgegenstand

## Definiert allgemeine, abstrakte Eigenschaften:

- istFrei(Zeitraum)
- reserviere(Zeitraum)
- berechnePreis
- etc.





# PlugIn-Architektur für Satellitensteuerungen



in Kooperation mit der European Space Agency (ESA): 1998 – 2002

# Automatische Generierung von Software aus „Bauplänen“ (Modellen)

# Compiler: DIE Erfolgsgeschichte der Softwaretechnik

- Programmtext (Pascal, C#, xUML)  
→ ausführbares (Maschinen-)Programm

a+b als Maschinenprogramm:

010 100 hole die Zahl vom Speicherplatz 100(a) in einen Puffer  
015 101 addiere dazu den Inhalt von Speicherplatz 101(b)  
011 102 speichere das Ergebnis in Zelle 102(a+b)

- für das Zeitverhalten von Embedded Systems noch ausständig
- für andere Anwendungsbereiche denkbar?

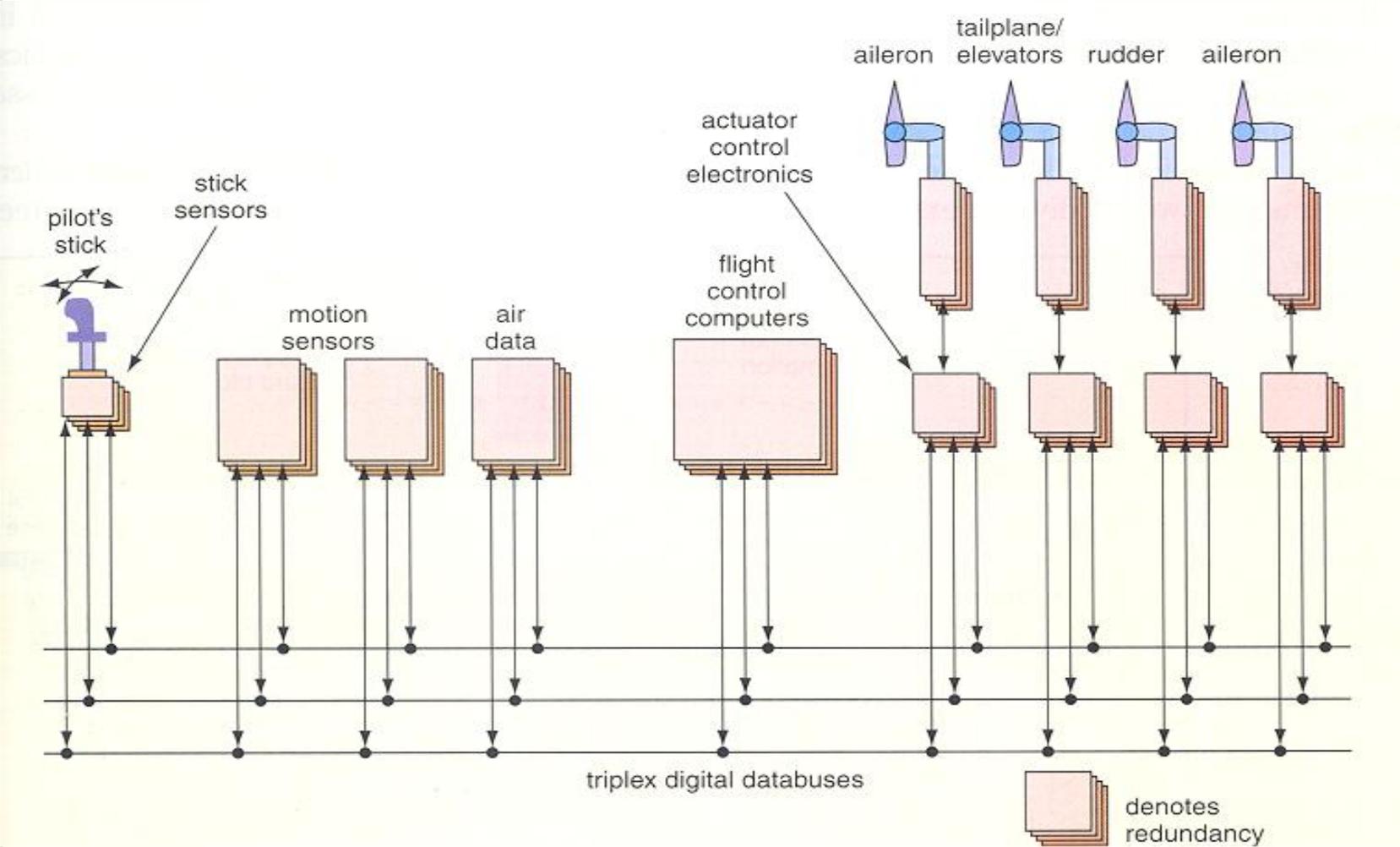
# Beispiel: Helicopter Control System (I)



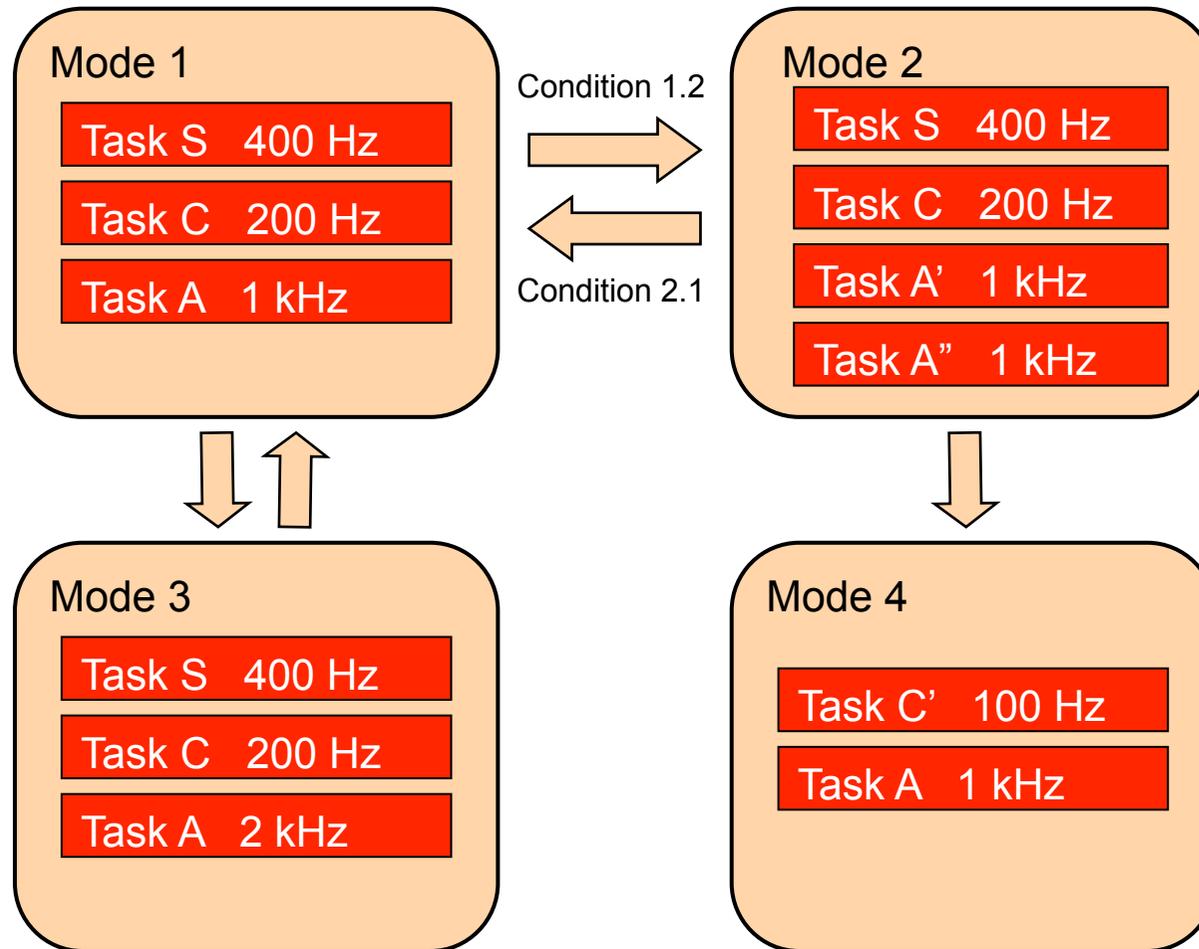
Henzinger, Kirsch, Pree, Sanvido (UC Berkeley)

Schaufelberger, Wirth (ETH Zürich)

# Beispiel: Helicopter Control System (II)

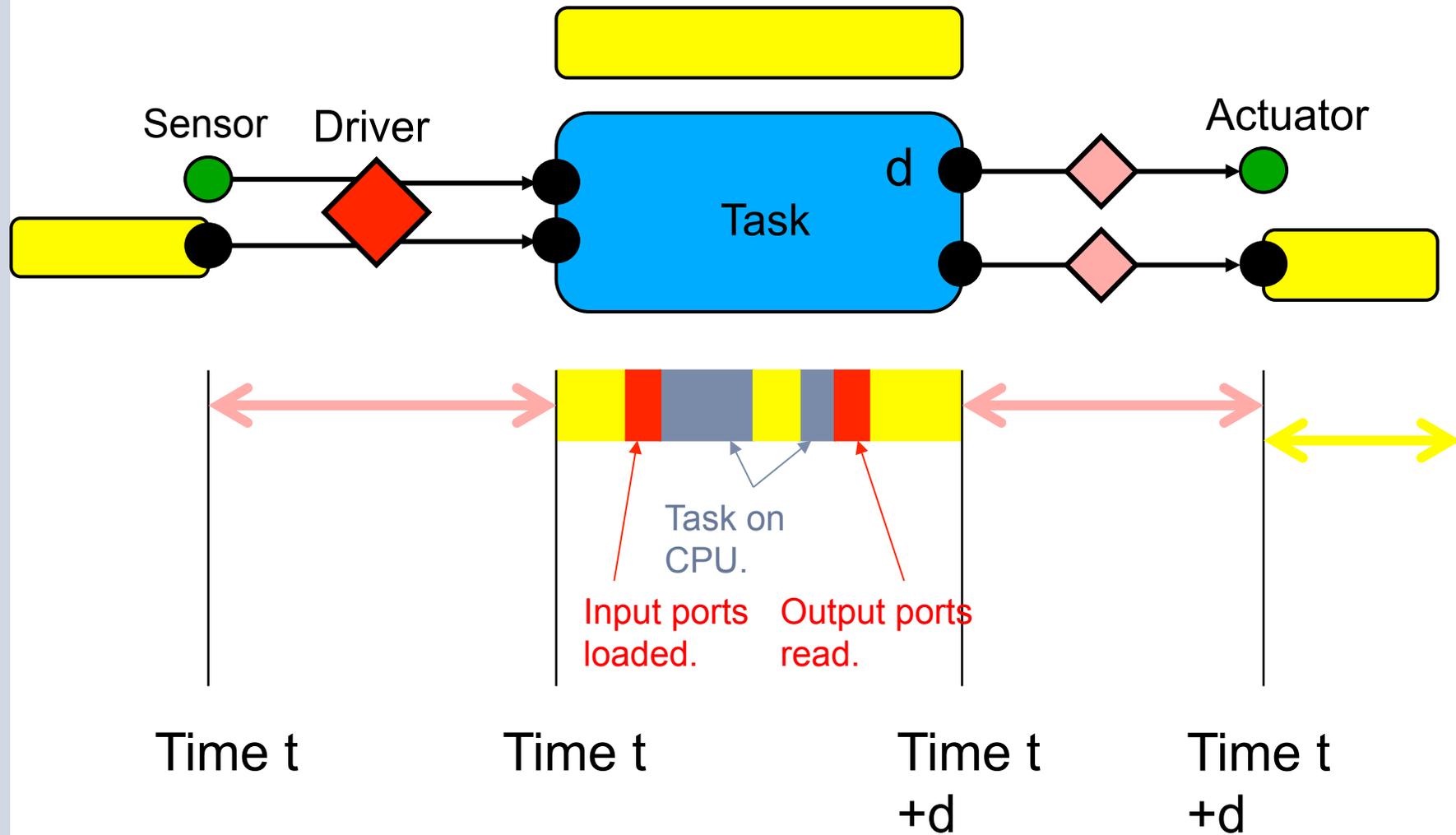


# Giotto als „höhere Programmiersprache“ = Bauplan = Modell für Zeitaspekte



**=> spezif. HW-Plattform wird irrelevant**

# Generierung der Software + deren Verteilung auf Prozessoren erfolgt automatisch



# Softwaretechnik – Quo vadis?

- kostenintensive Wartung von Software, die 20-30 Jahre alt ist
- ingenieurmäßige Herangehensweise wird sich zumindest in Teilbereichen etablieren, zB bei sicherheitskritischen Systemen

- simple, mechanische Weltsicht schwer skalierbar



- Vorbild biologische Systeme  
→ Internet wuchs um den Faktor 100 Mio.

# Softwaretechnik – Quo vadis?

*The End*

**Vielen Dank für Ihre  
Aufmerksamkeit!**