

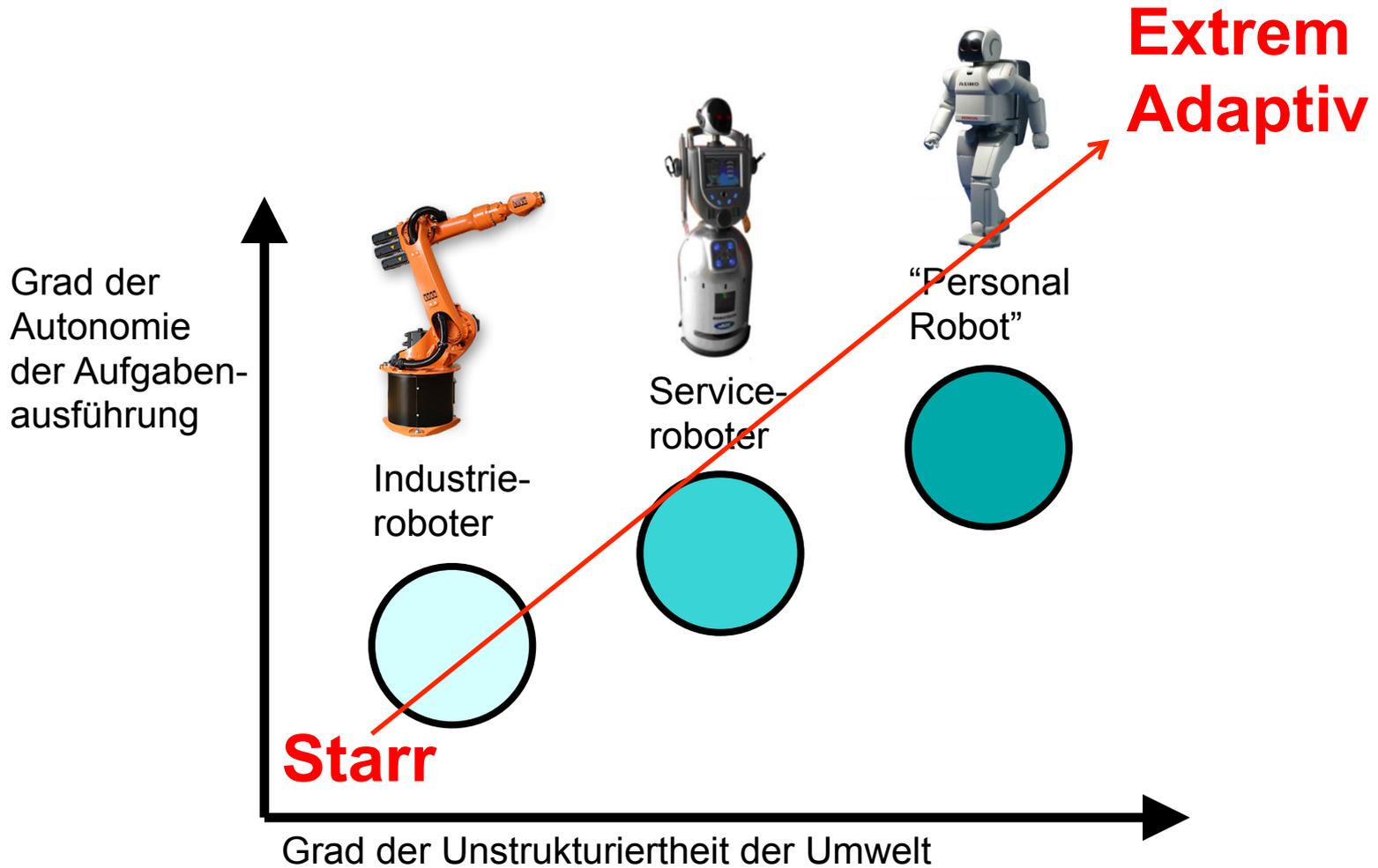
# Autonome Roboter: Beispiele, Architekturen und Planung

Prof. Dr.-Ing. Rüdiger Dillmann

Dr.-Ing. Sven R. Schmidt-Rohr

Dr.-Ing. Rainer Jäkel

- **Motivation: Autonome Roboter**
- Beispiele für Serviceroboter
- Architekturen für die Informationsverarbeitung bei autonomen Robotern
- Symbolische Planung für autonomes Handeln



- Motivation: Autonome Roboter
- **Beispiele für Serviceroboter**
- Architekturen für die Informationsverarbeitung bei autonomen Robotern
- Symbolische Planung für autonomes Handeln

## Einsatzgebiete

- Baugewerbe und Dienstleistungen
- Kommunalwesen, Umweltschutz und Landwirtschaft
- Sicherheit und Katastrophenschutz
- Medizin
- Rehabilitation
- Hotel und Gastronomie
- Hobby, Freizeit und Entertainment
- Weltraum- und Unterwassertechnik

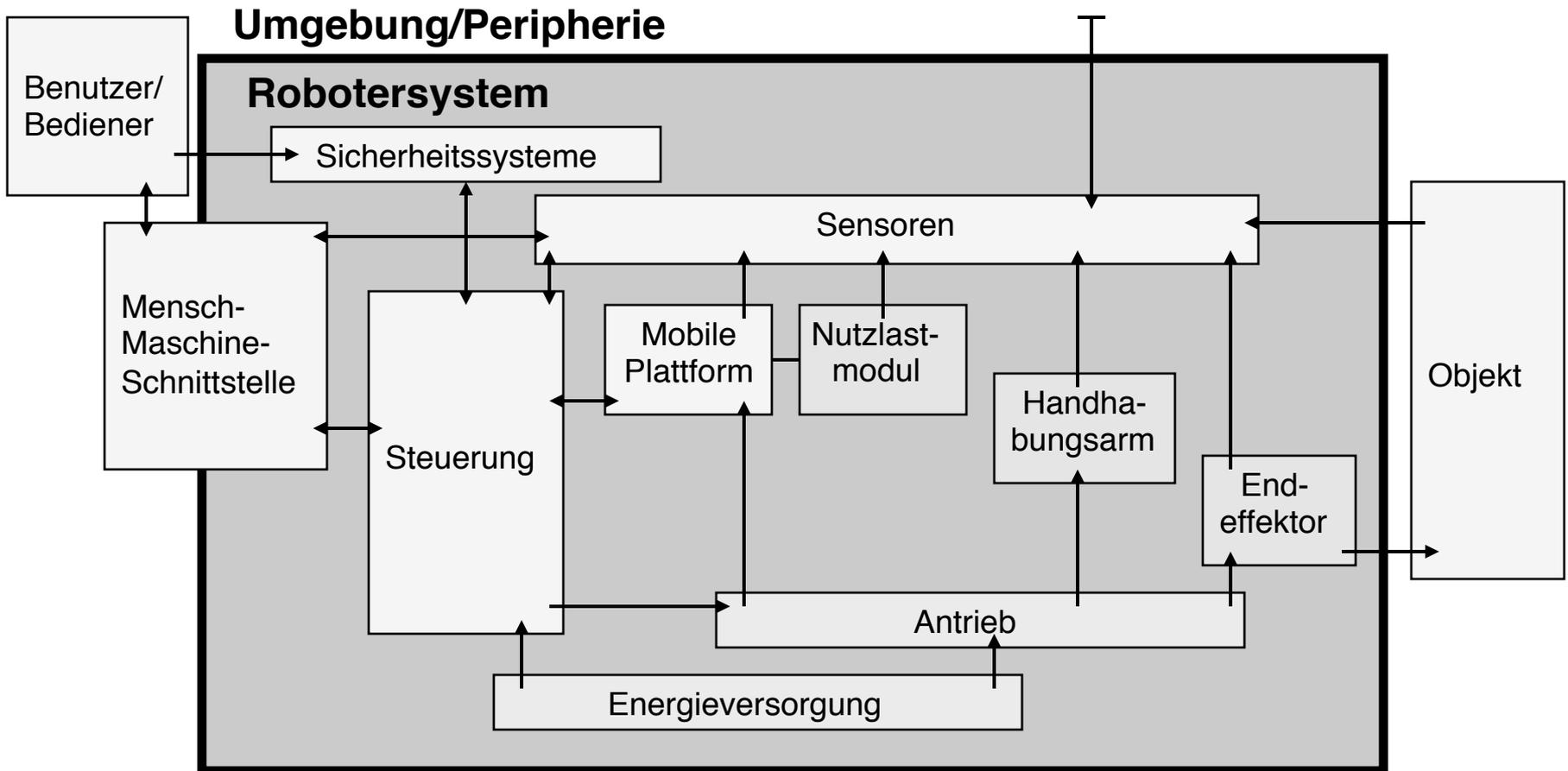
## Basisfunktionen

- Orientierung in Bewegung
- Umgebungserfassung
- Umweltmodellierung
- Ortung
- Bahnplanung
- Bewegungsführung
- Sicherheit und Kollisionsschutz
- Handhabung

## Teilsysteme

- Handhabungsarm
- Endeffektor
- Mobile Plattformen
- Antriebstechnik
- Steuerung
- Sensorik
- Mensch-Maschine-Schnittstelle
- Sicherheitssystem

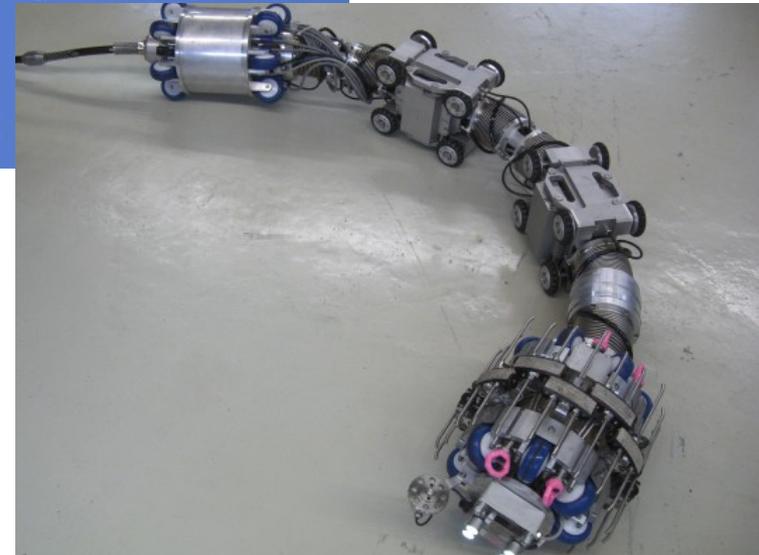
## Teilsysteme (2)



## Flexibler Produktions/Handwerksassistent



## Spezielle Inspektionsroboter



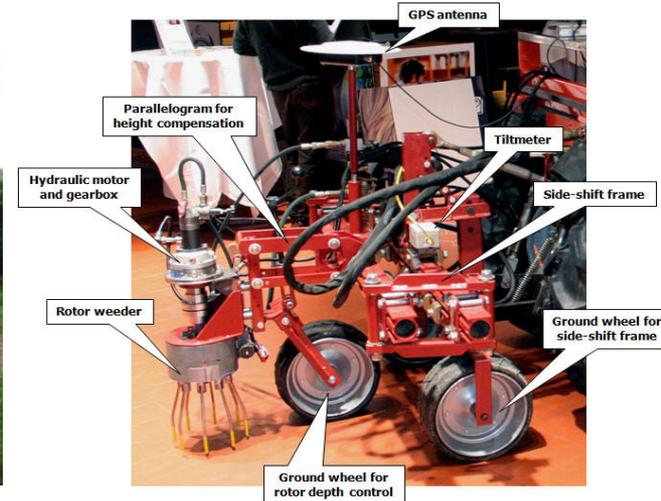
## Über- und unterwasserdrohnen (zivil und militärisch)



## Flugdrohnen (zivil und militärisch)



## Autonome, flexible Erntemaschinen



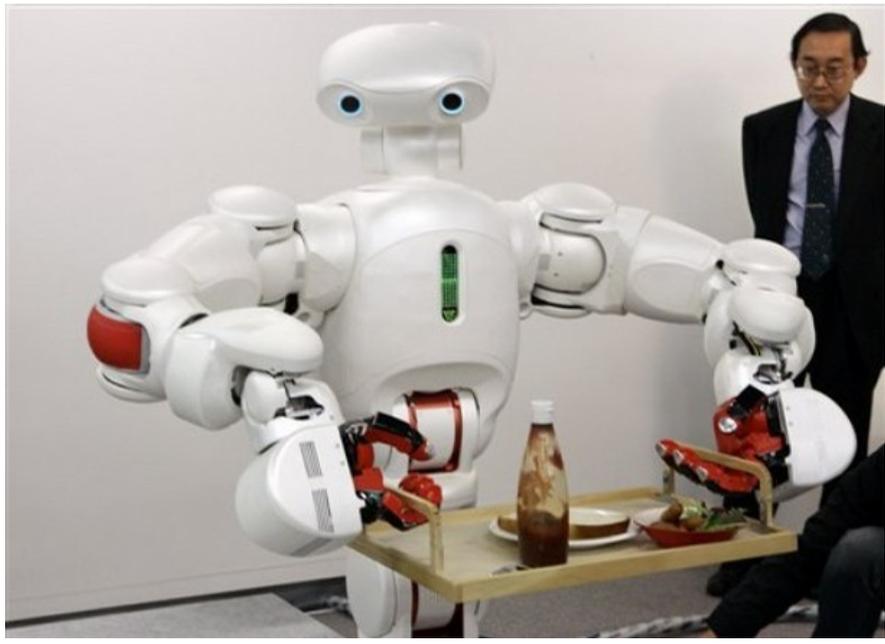
## Teil- bzw. vollautonome Kampfroboter (Infanterie)



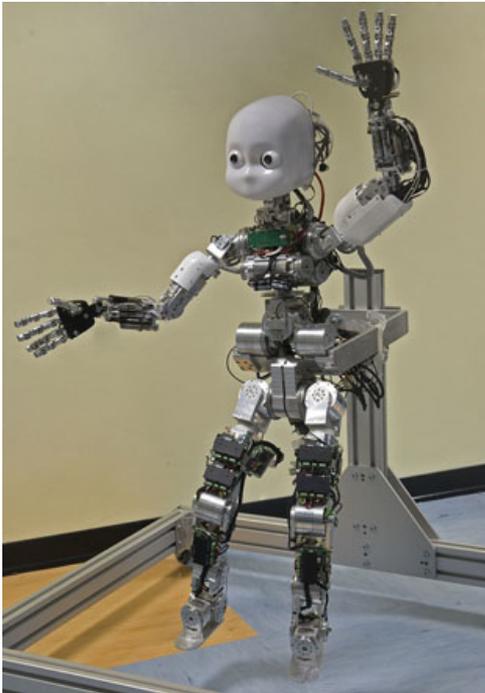
## Pflege von Alten und Kranken (Antwort auf das Demografieproblem)



## Persönlicher Diener im Haushalt



## Androiden: belebte Puppen bzw. künstliche Menschen



## PR2 Making Popcorn at Automatica Fair 2012



## Artificial Cognitive System in DEXMART: Learning from demonstrations and Execution

Learning manipulation strategies from human demonstrations for autonomous execution combined with decomposition planning and reachability analysis



EC Integrated Project in the  
Sventh Framework Programme (FP7/2007–2013)  
under grand agreement no. 126239

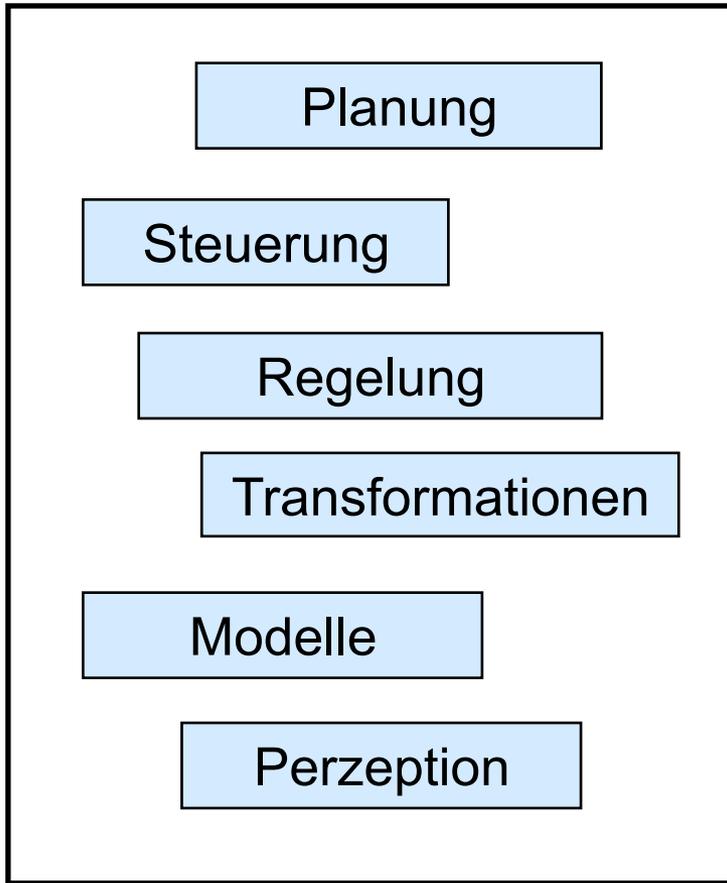


- Motivation: Autonome Roboter
- Beispiele für Serviceroboter
- **Architekturen für die Informationsverarbeitung bei autonomen Robotern**
- Symbolische Planung für autonomes Handeln

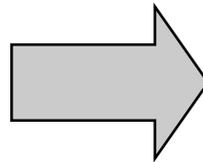
- Einführung
- Funktionsorientierte Architekturen
- Verhaltensorientierte Architekturen
- Hybride Architekturen
- Kognitive Architekturen

## Motivation

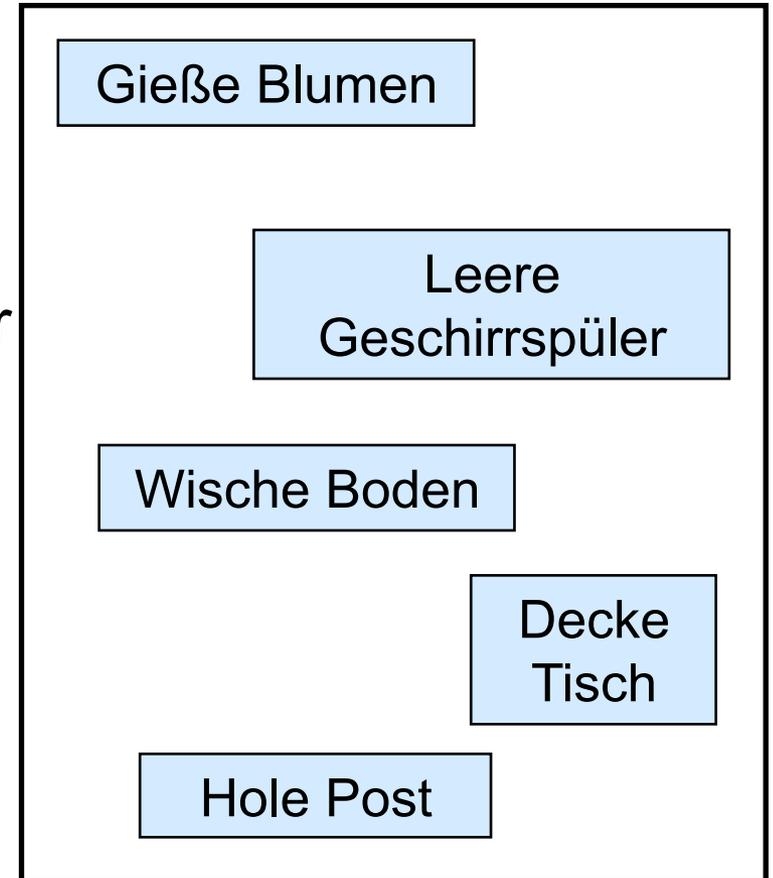
### Komponenten



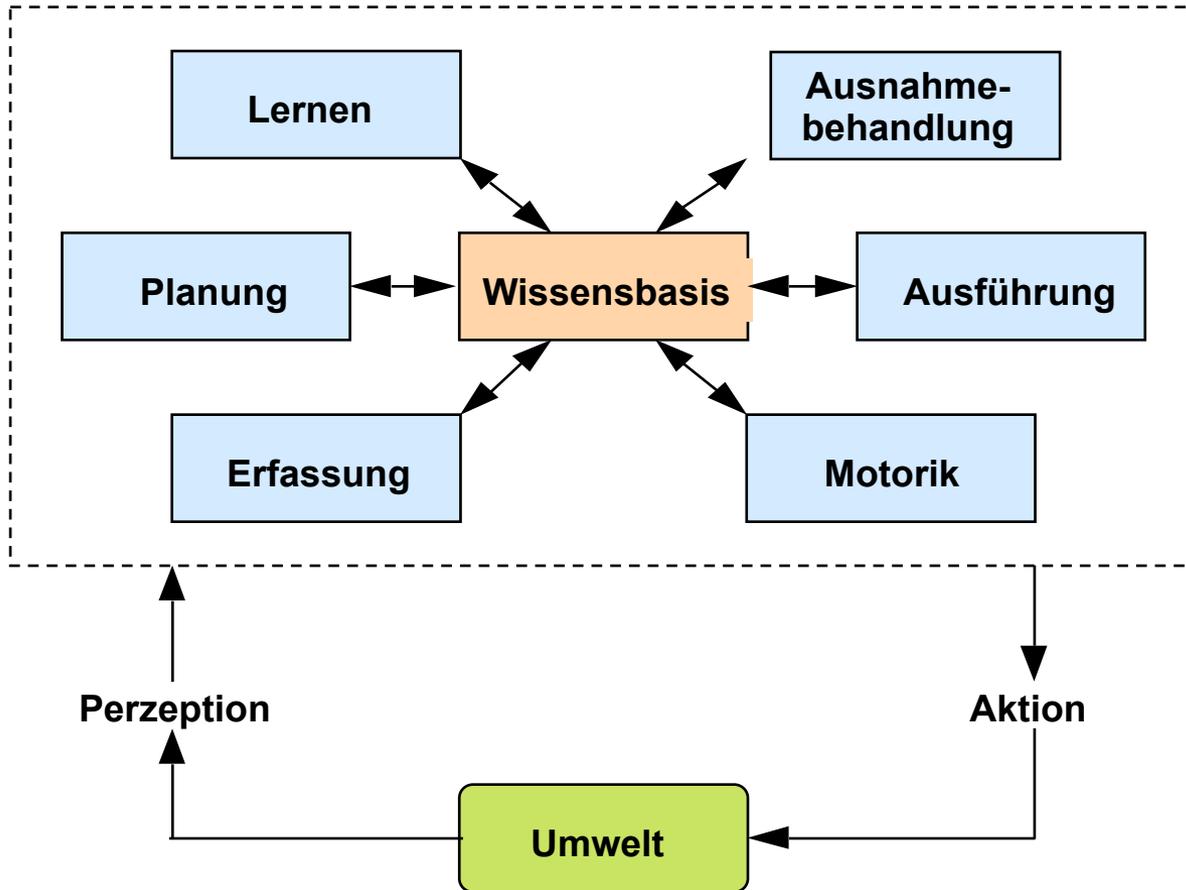
### Architektur



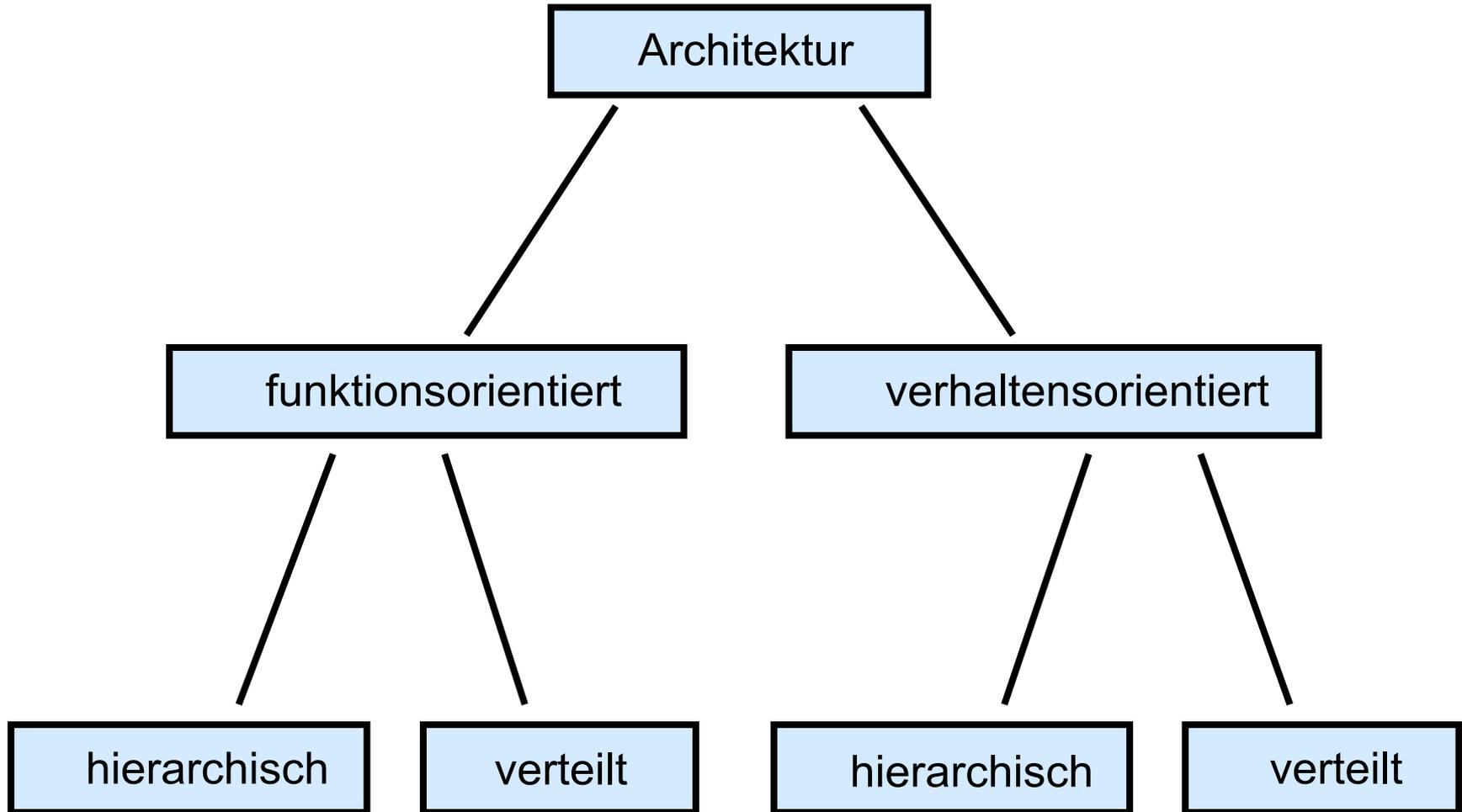
### Aufgaben eines Assistenzroboters



## Fähigkeiten eines autonomen Robotersystems



## Klassifikation von Architekturen



- Einführung
- Funktionsorientierte Architekturen
  - Hierarchische funktionsorientierte Architekturen
  - Verteilt funktionsorientierte Architekturen
- Verhaltensorientierte Architekturen
- Hybride Architekturen
- Kognitive Architekturen

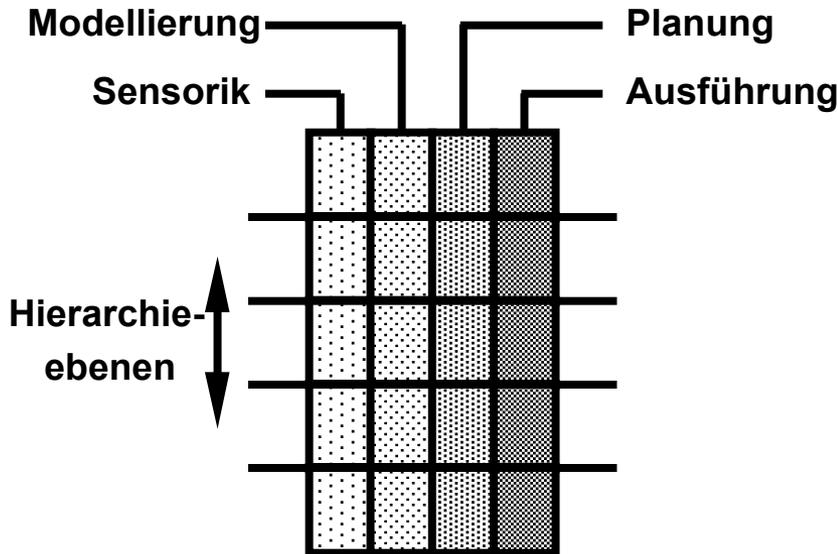
## Genereller Aufbau



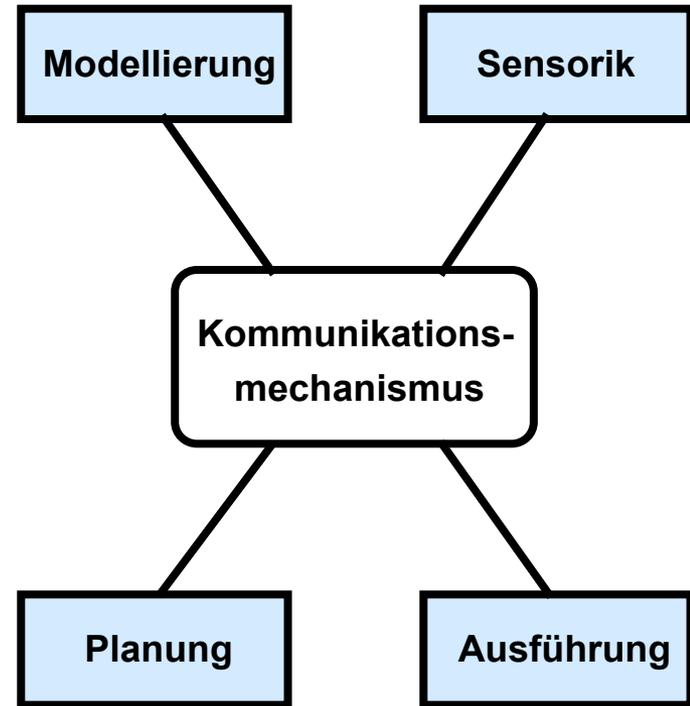
## Eigenschaften:

- Unidirektionaler Informationsfluss
- Schlecht für dynamische Umgebungen, Interaktion
- Vorteil: Einfacher Aufbau

## Typen von funktionsorientierten Architekturen



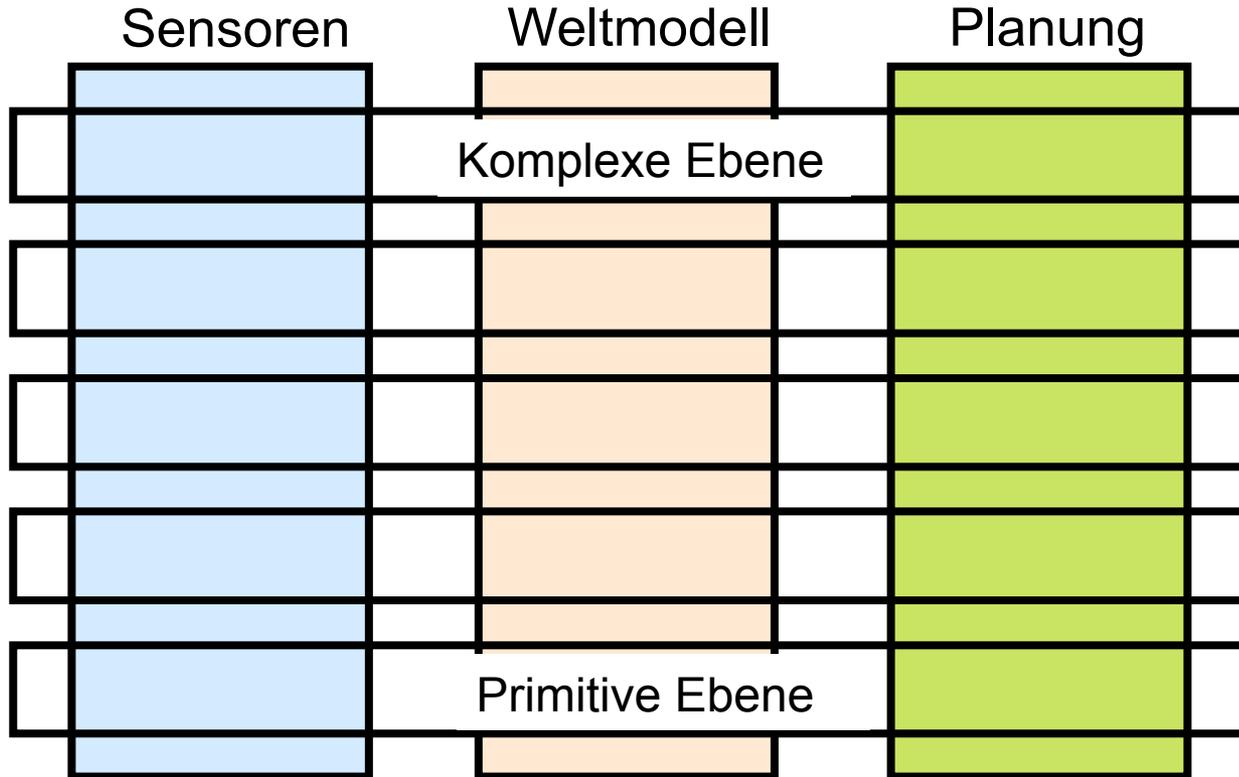
hierarchisch funktionsorientiert



verteilt funktionsorientiert

- Einführung
- Funktionsorientierte Architekturen
  - Hierarchische funktionsorientierte Architekturen
  - Verteilte funktionsorientierte Architekturen
- Verhaltensorientierte Architekturen
- Hybride Architekturen
- Kognitive Architekturen

## NASREM-Modell von Albus



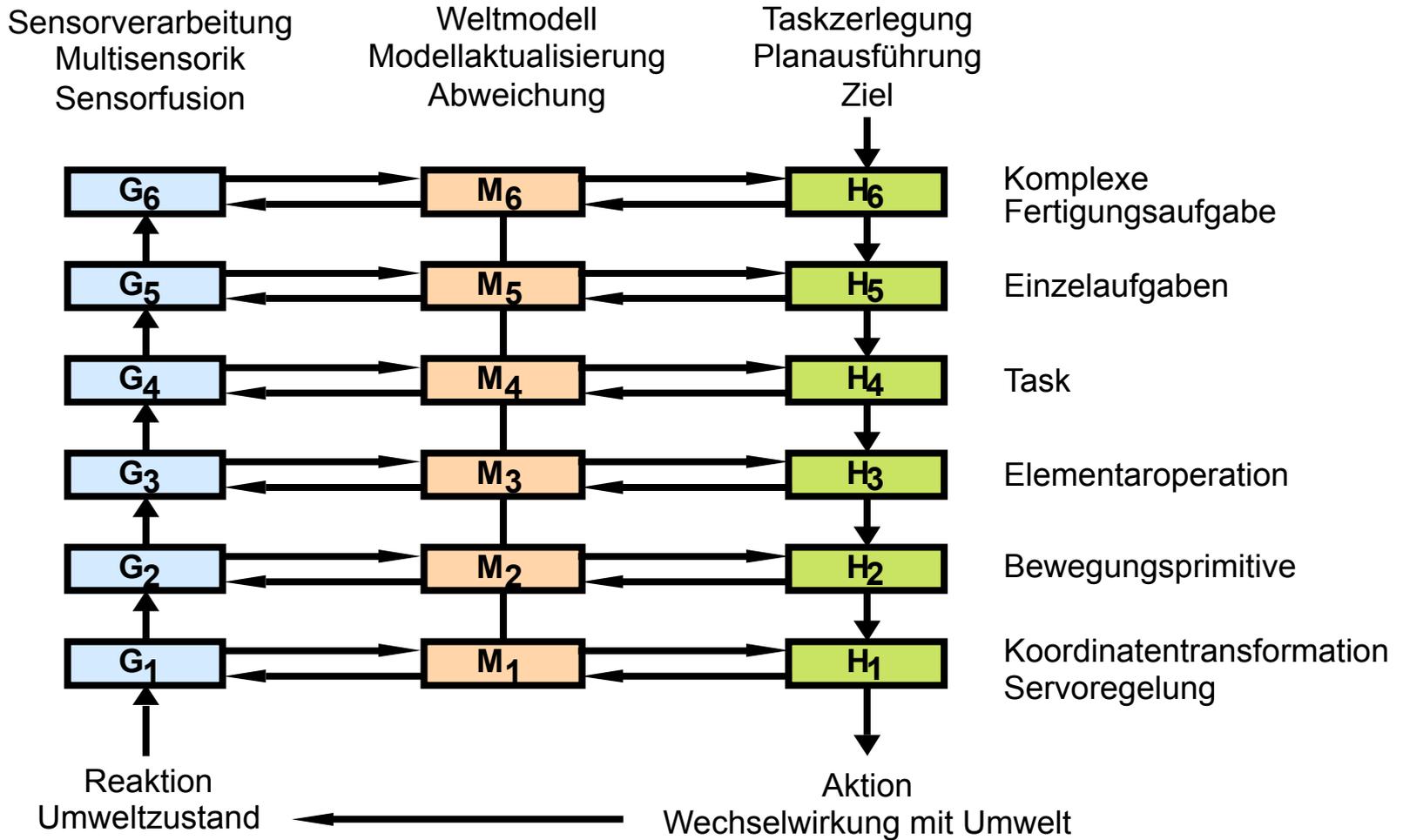
Albus, J. S.; McCain, H. G.; Lumini, R; "NASA / NBS Standard Reference Model for Telerobot Control System Architecture" NBS Technical Note 1235, 1987.

## Aufbau des NASREM-Modells (I)

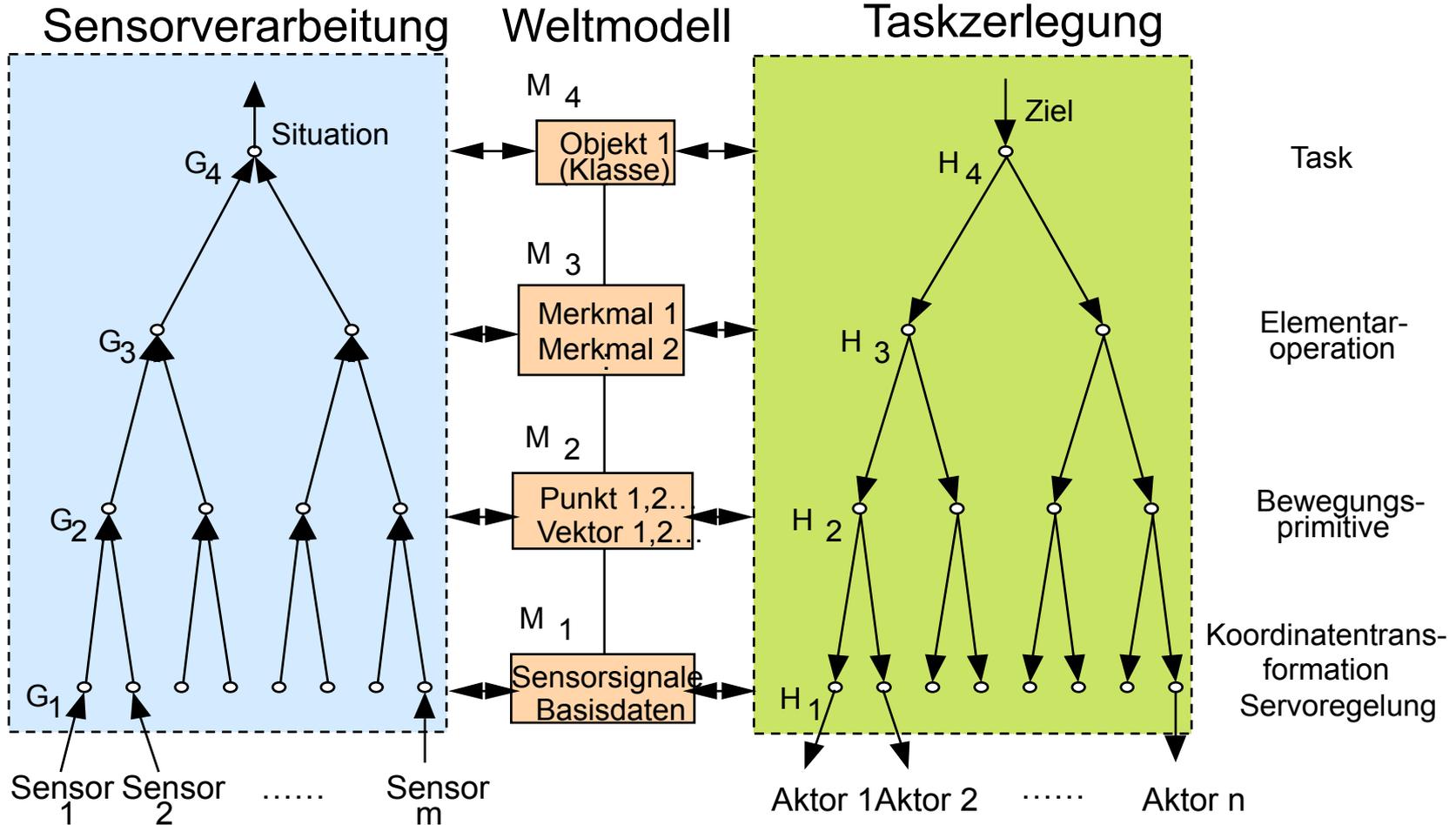
- Aufteilung in 4 bis 6 Ebenen
- je Ebene 3 sog. “Module”
  - $G_i$ : Sensorverarbeitungsmodul
  - $M_i$ : Weltmodell- und Referenzdatenmodul
  - $H_i$ : Taskzerlegungs-, Planungs- und Ausführungsmodul

Jede Modul-Art ist durch die Ebeneneinteilung hierarchisch geordnet

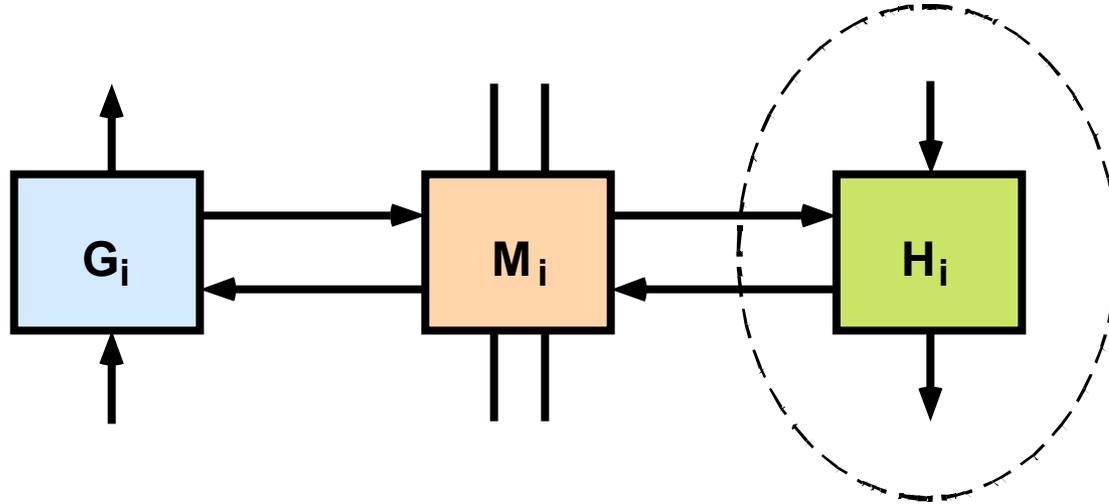
## Aufbau des NASREM-Modells (II)



## Informationsfluss im NASREM-Modell

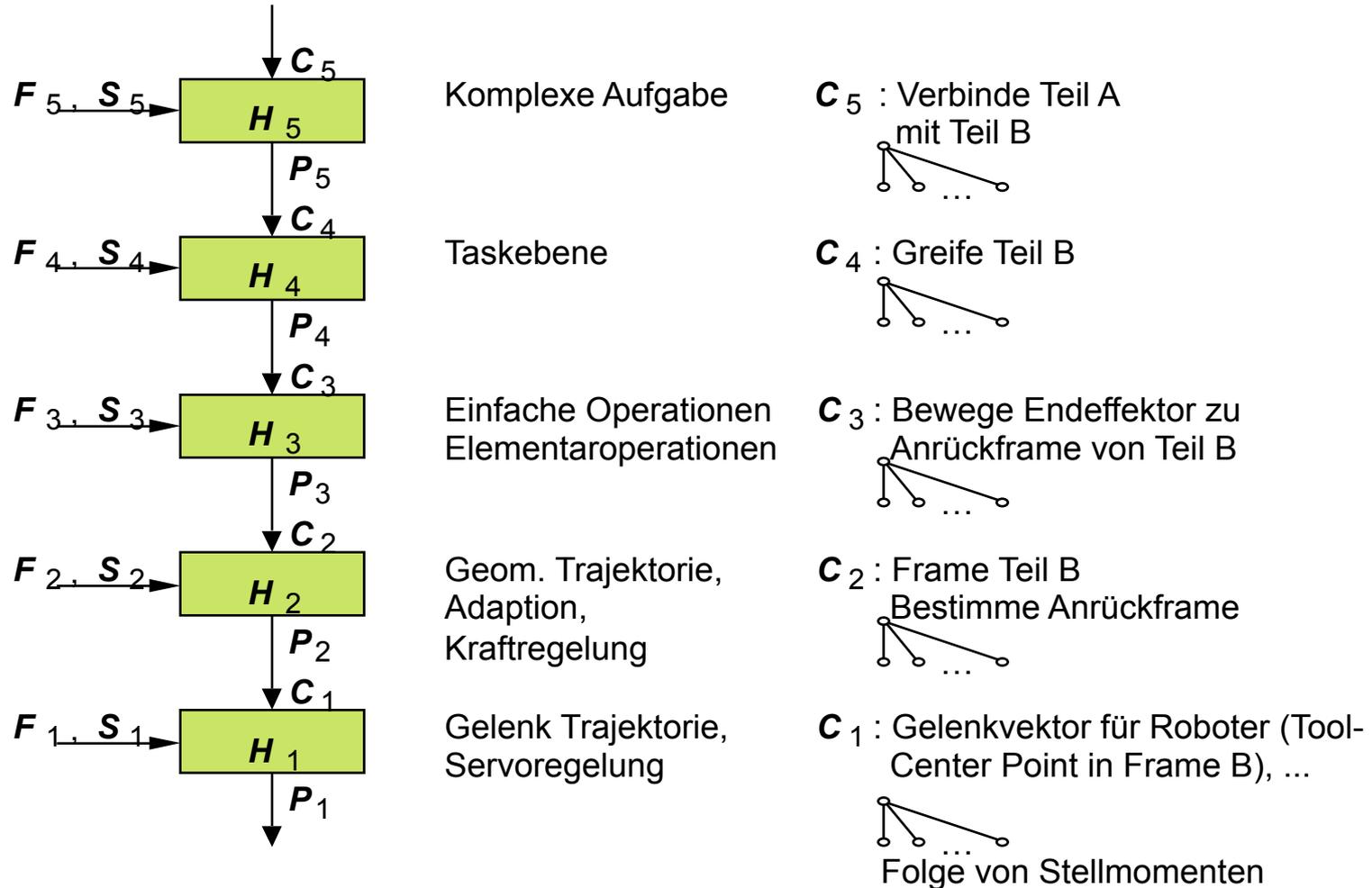


## Das H-Modul

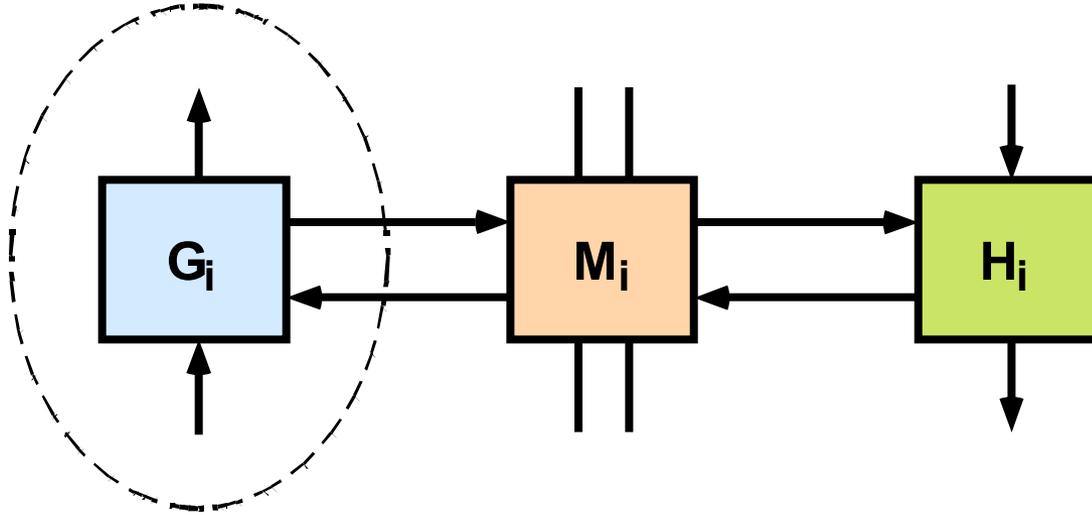


1. Das H-Modul erhält eine Aufgabe aus der oberen Ebene.
2. Mit den Daten aus dem M-Modul wird die Aufgabe in Teilaufgaben zerlegt.
3. Das M-Modul wird aktualisiert.
4. Die Teilaufgaben werden (einzeln) an die untere Ebene weitergereicht.

## Das H-Modul: Zerlegung der Taskanweisung

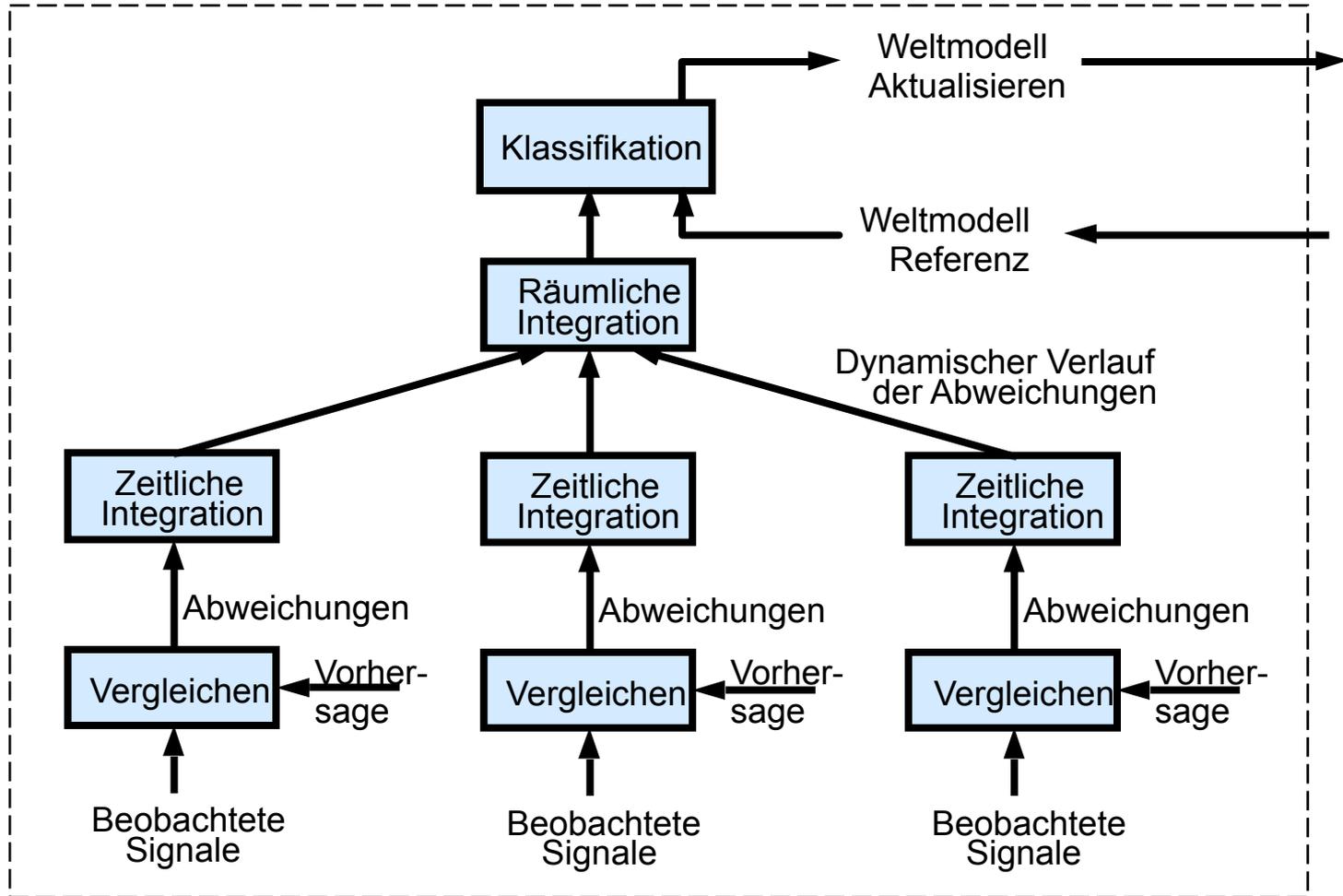


## Das G-Modul

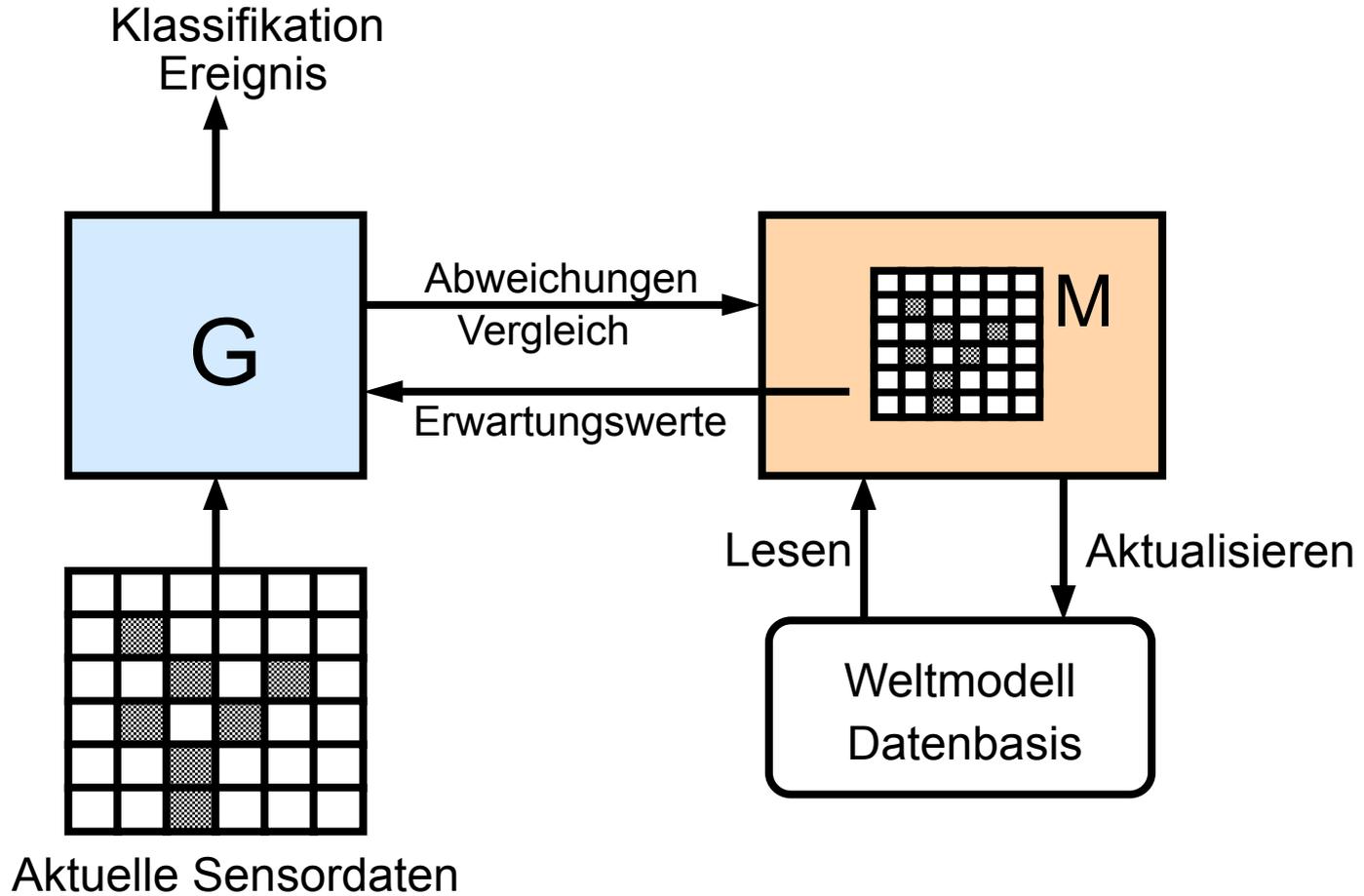


1. Das G-Modul bekommt Sensor-Informationen aus der unteren Ebene.
2. Mit dem M-Modul werden diese Daten verarbeitet.
3. Das M-Modul wird aktualisiert.
4. Die Informationen werden der oberen Ebene zur Verfügung gestellt.

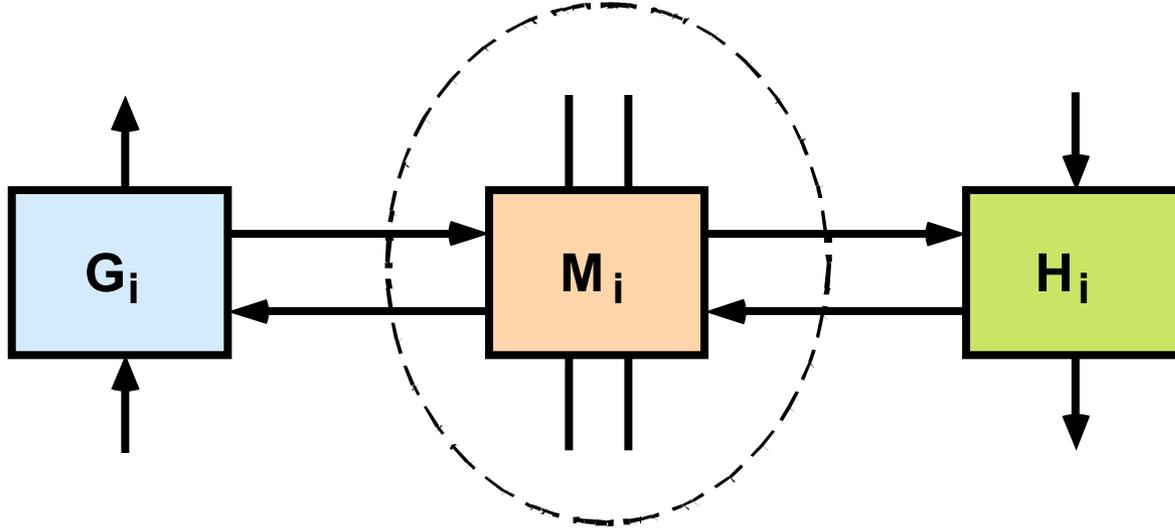
## Aufbau des G-Moduls (I)



## Aufbau des G-Moduls (II)

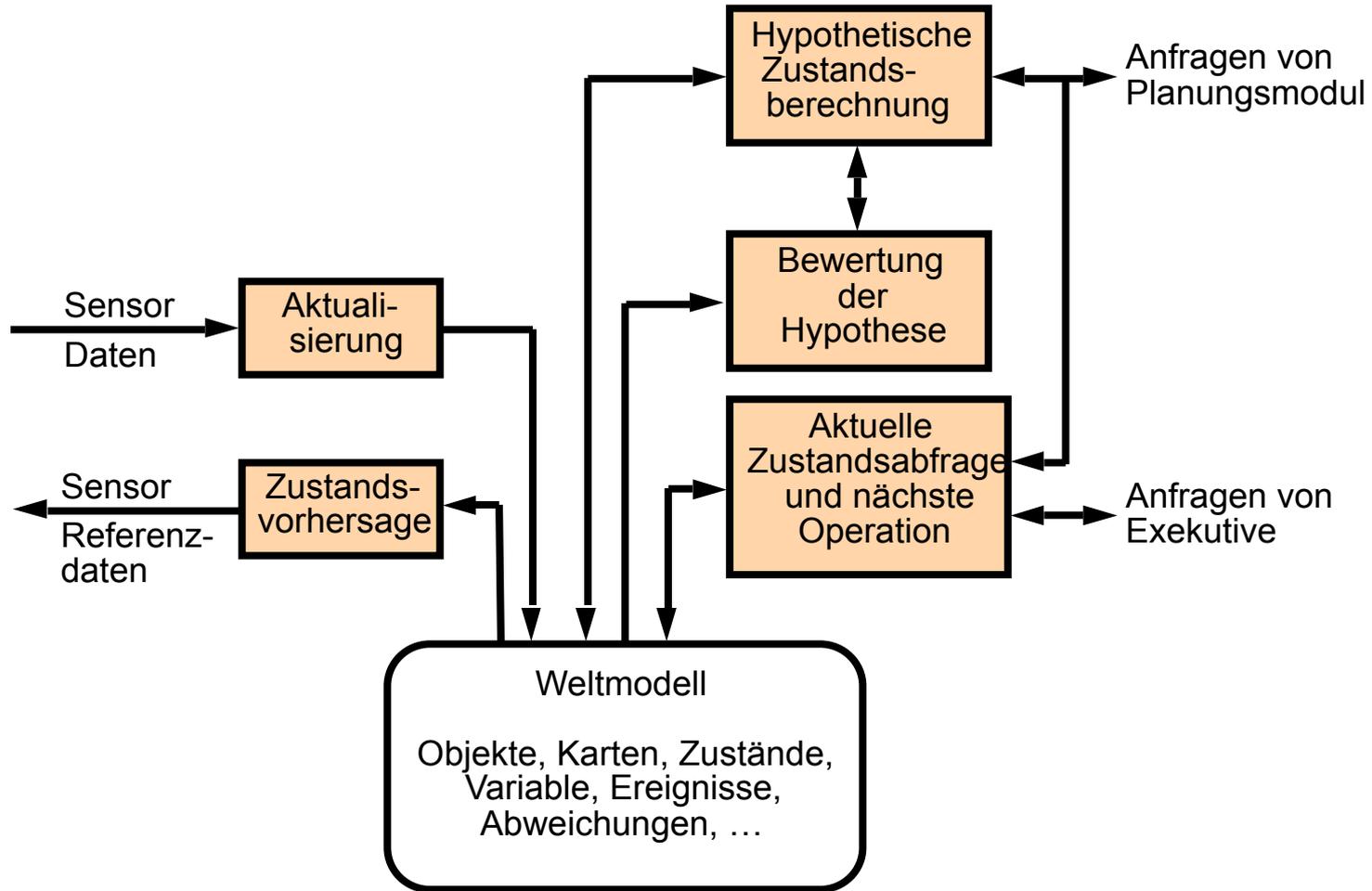


## Das M-Modul



1. Das M-Modul besteht aus einer einzigen Komponente, mit welcher die G- und H-Module Daten mit dem Abstraktionsgrad der jeweiligen Ebene austauschen.
2. Aufgrund einer Differenz zwischen den Solldaten (M-Modul) und den Ist-Daten (G-Modul) wird eine Störung erkannt.

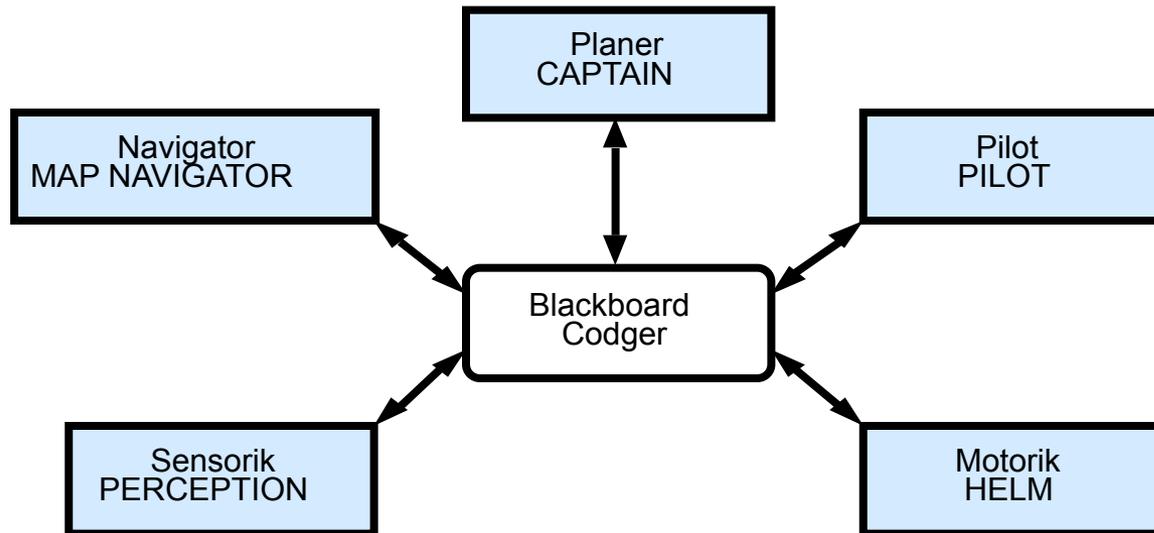
## Typische Operationen auf dem Weltmodell



- Einführung
- Funktionsorientierte Architekturen
  - Hierarchische funktionsorientierte Architekturen
  - Verteilte funktionsorientierte Architekturen
- Verhaltensorientierte Architekturen
- Hybride Architekturen
- Kognitive Architekturen

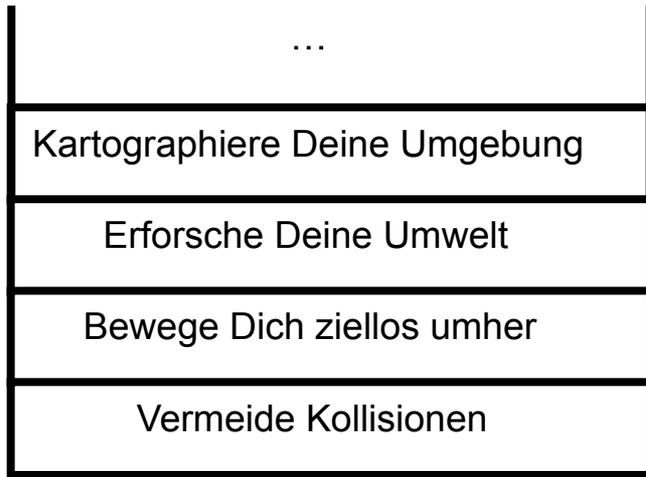
## Nav-Lab System der CMU

- Menge spezialisierter Teilsysteme
- Kommunikation über eine zentrale Kompetenz

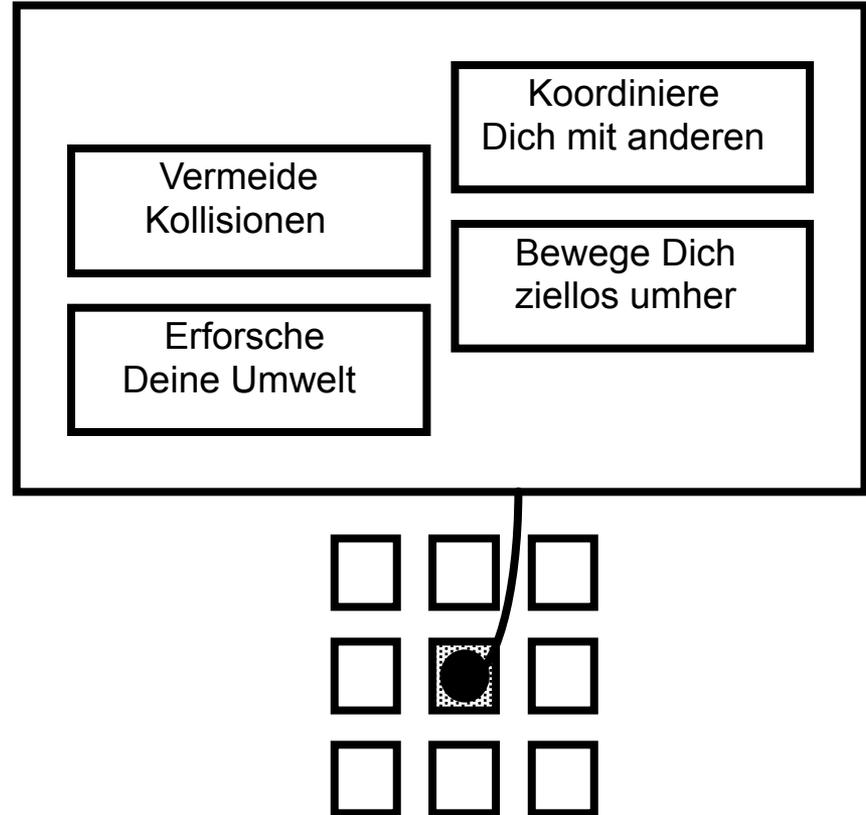


- Einführung
- Funktionsorientierte Architekturen
- **Verhaltensorientierte Architekturen**
  - Hierarchische verhaltensorientierte Architekturen
  - Verteilte verhaltensorientierte Architekturen
- Hybride Architekturen
- Kognitive Architekturen

## Typen von verhaltensorientierten Architekturen



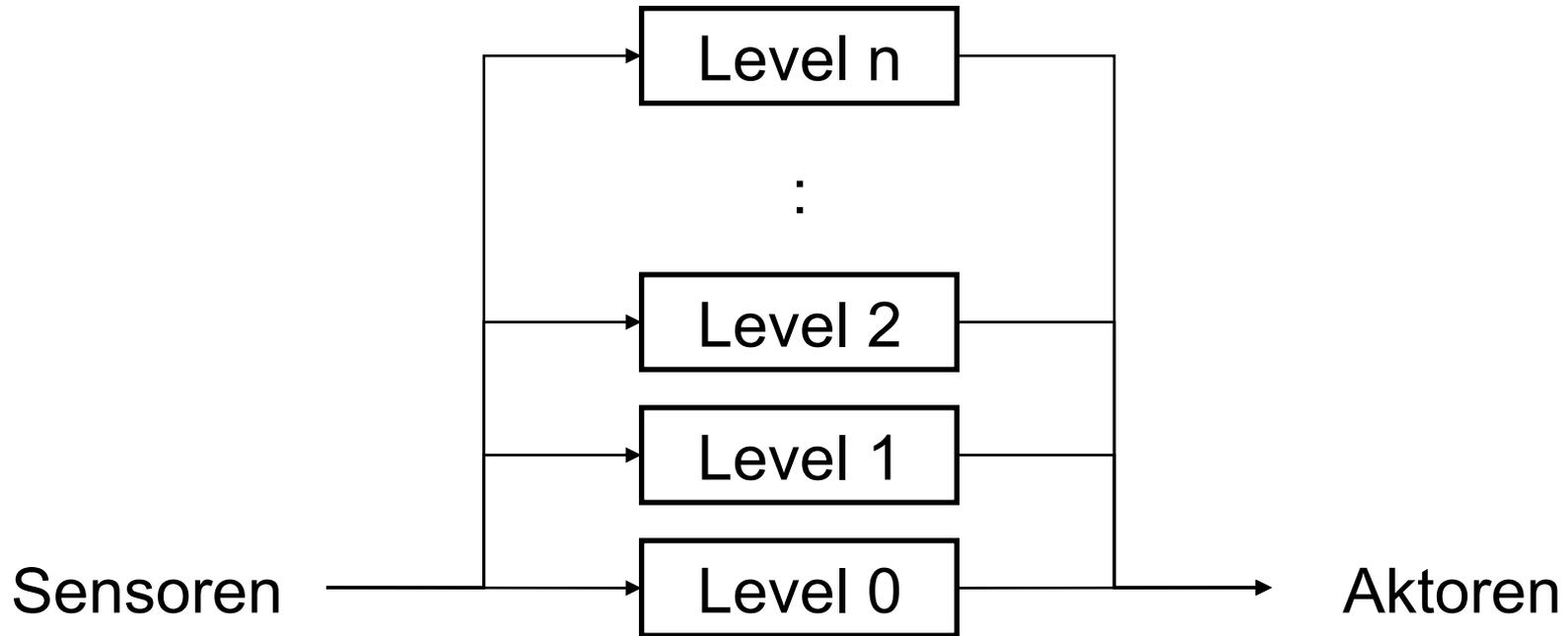
hierarchisch verhaltensorientiert



verteilt verhaltensorientiert

## Subsumption

- Hierarchische Anordnung der Einzelverhalten
- Übergeordnete Verhalten können die Ausgabe darunter liegender Verhalten hemmen



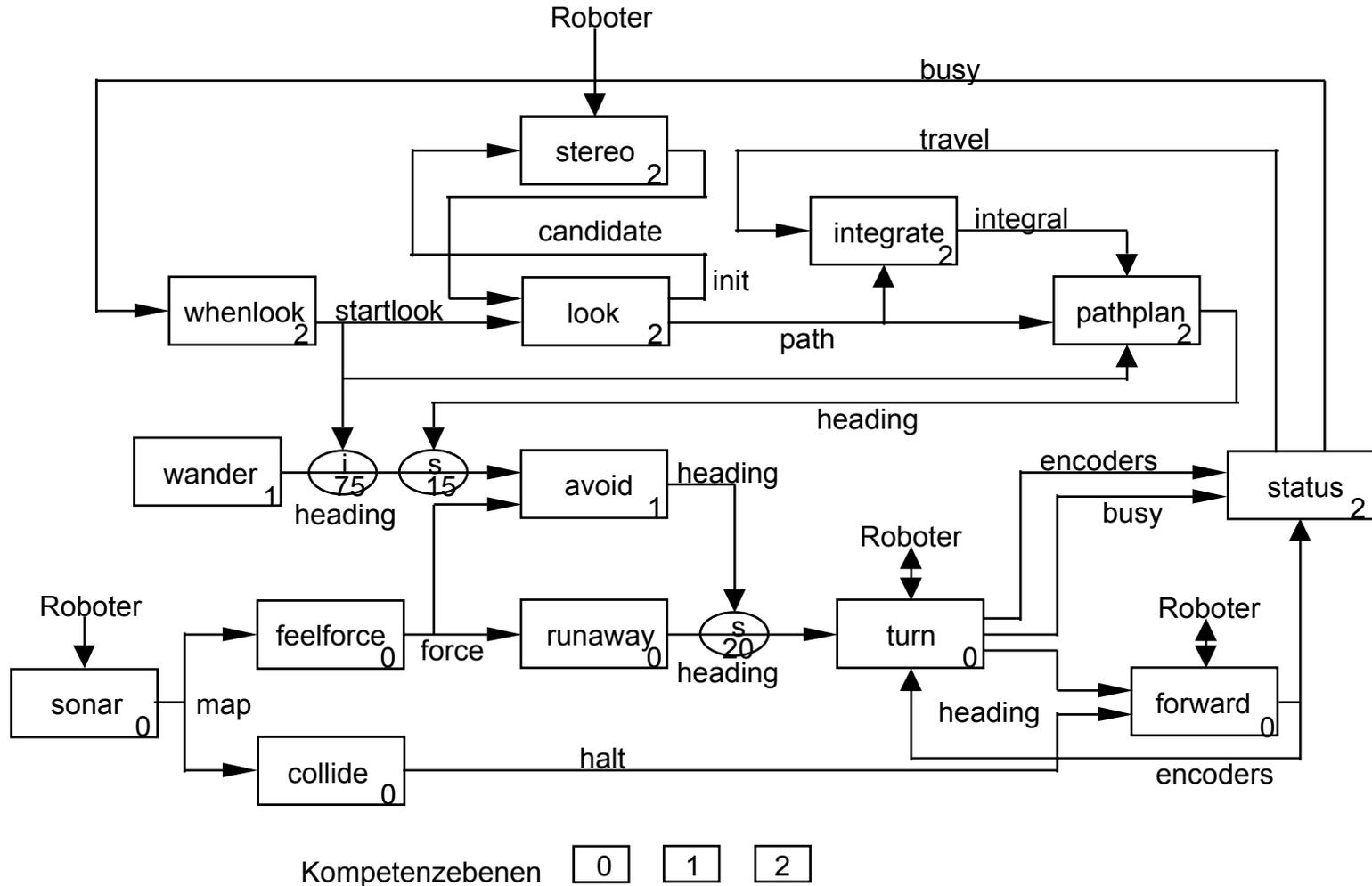
## Eigenschaften

- Steuerung durch Verhaltensmuster bzw. Reflexe
- Muster: Reaktion des Systems auf bestimmte Sensorstimuli (Umweltsituation)
- Anordnung in Verhaltensebenen
- Hierarchische Struktur der Kompetenzebenen
- Beispiel: Subsumption Architecture (R. Brooks, MIT)  
Verhaltensschemata als endliche Zustandsautomatennetze

## Beispiel: Hierarchische Verhalten bei Fahrzeug

1. Vermeidung von Kontakt mit Objekten
2. Ziellose Bewegung in der Umgebung/Erforschung der Umwelt
3. Erstellung einer Karte der Umgebung und Durchführung einer Wegplanung
4. Erfassung von Veränderungen in der lokalen Umgebung
5. Ziehen von Schlussfolgerungen über die Umwelt im Hinblick auf erkennbare Objekte und Ausführung objektbezogener Aufgaben
6. Erstellung und Ausführung von Plänen, welche die Umwelt in einer definierten Weise verändern
7. Ziehen von Schlussfolgerungen über das Verhalten von Objekten in der Umgebung und entsprechende Modifikation von Plänen

## Beispiel: Steuerungsarchitektur des Robot-Systems



- Einführung
- Funktionsorientierte Architekturen
- **Verhaltensorientierte Architekturen**
  - Hierarchische verhaltensorientierte Architekturen
  - **Verteilte verhaltensorientierte Architekturen**
- Hybride Architekturen
- Kognitive Architekturen

## Eigenschaften

- Menge von unabhängigen Teilsystemen mit identischen Verhaltensmustern
- Koordination erfolgt über ein Verhaltensmuster

## Beispiel

- Multi-Agenten-Systeme

Eigenschaften:

- Selbstorganisation
- assoziative Speicherung von Information
- Erkennung von bestimmten Mustern

- Einführung
- Funktionsorientierte Architekturen
- Verhaltensorientierte Architekturen
- **Hybride Architekturen**
- Kognitive Architekturen

## Aufbau

### Meistens: 3-Schichten Modell

- Obere Ebene: Planung
- Mittlere Ebene: Sequentialisierung (Ausführen/Abspielen von Plänen)
- Untere Ebene: Verhaltensebene

## Beispiel: PACO+ Architektur

### High Level

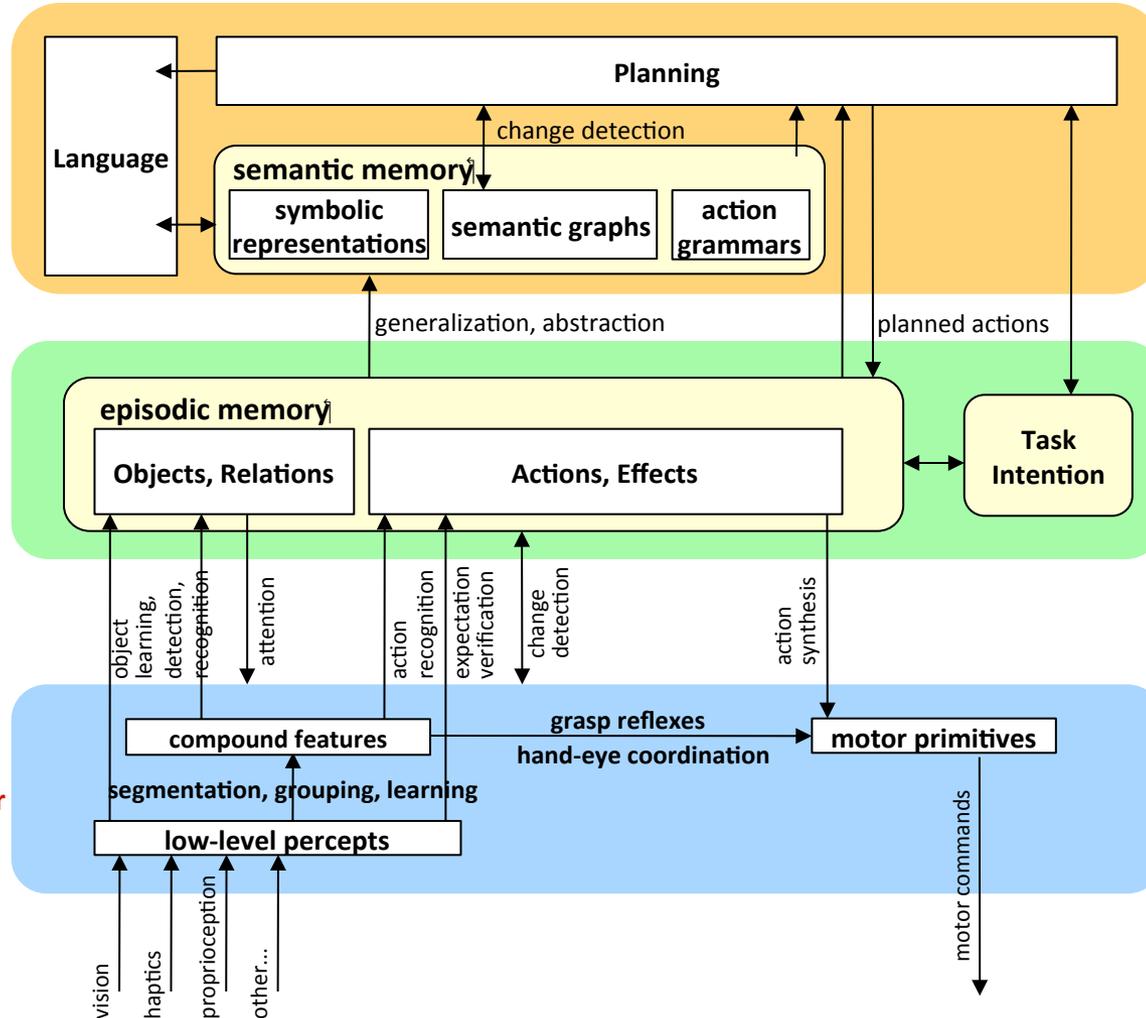
- Reasoning
- Planning
- Language

### Mid Level

- Recognition
- Memory
- Consolidation
- Action
- Selection

### Low Level

- Online Sensorimotor Processing



- Einführung
- Funktionsorientierte Architekturen
- Verhaltensorientierte Architekturen
- Hybride Architekturen
- Kognitive Architekturen

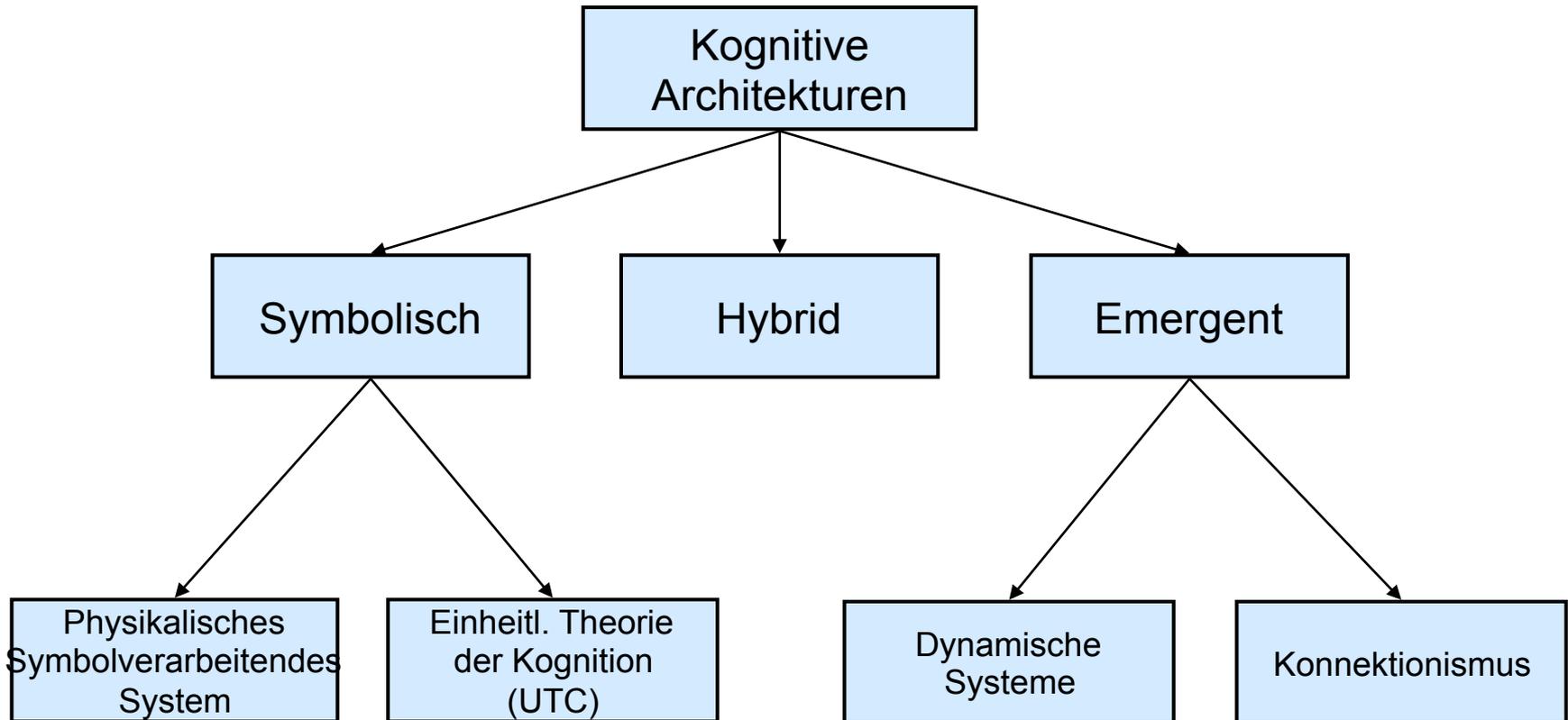
## Definition (P. Langley, CMU, 2006):

„A cognitive architecture specifies the **infrastructure for an intelligent system** that remains **constant across different domains** and knowledge bases.

This infrastructure includes a commitment to formalisms for **representing** knowledge, **memories** for storing this domain content, and **processes that utilize and acquire** the knowledge.“

- Problem:
  - Lernfähigkeit des Roboters
  - Hinzufügen neuen Handlungswissens
- Keine eindeutige Definition
  - Vorbild: kognitive Prozesse des Menschen
  - Ziel: Eigendynamik, Erweiterbarkeit, Generalisierung des Systems

## Übersicht



## Symbolische Architekturen

- Basieren auf einer symbolische Repräsentation
- Vorschriften zur Verarbeitung der Symbole, oft in Form von Regeln
- Beispiele:
  - Soar (Symbolic cognitive architecture), J. Laird + A. Newell + P. Rosenbloom, 1987
  - ACT-R (Adaptive Control of Thought - Rational), Anderson, J. R. (1990)

## Emergenz

Spontane Herausbildung von Phänomenen oder Strukturen auf der Makroebene eines Systems auf der Grundlage des Zusammenspiels seiner Elemente

## Emergente Systeme

- Basierend auf einfachen Grundelementen, komplexe Funktionen durch Verschaltung der Elemente
- Repräsentationen sind subsymbolisch und verteilt
- Beispiele:
  - Neuronale Netze

- Motivation: Autonome Roboter
- Beispiele für Serviceroboter
- Architekturen für die Informationsverarbeitung bei autonomen Robotern
- **Symbolische Planung für autonomes Handeln**

Warum planen?

Was planen?

Wie planen?

## Warum planen?

- Planung dort, wo mehr als ein Schritt nötig ist, um zum Ziel zu kommen (eine Aufgabe zu erledigen).

## Was planen?

- Aktionsplanung
  - Welche Aktionen werden aus Aufgabe abgeleitet?
- Reihenfolgeplanung
  - Zeitliche Beziehungen zwischen Aktionen
- Ressourcenplanung
  - Welche Roboter, Geräte führen Aktionen aus
- Zeitplanung
  - Wann werden Aktionen gestartet
- Durchführungsplanung
  - Welche Parameter für Aktionsausführung (Bahn- und Greifplanung)
- Scheduling
  - Ressourcenplanung und Zeitplanung

## Wie planen?

Unter einem **Planungssystem** für Roboter versteht man allgemein ein System, das ausgehend von einem **Anfangszustand** und der Beschreibung eines gewünschten **Zielzustandes** eine Folge von **Aktionen** (Aktionsplan) generiert, die das betrachtete System von seinem Anfangszustand schrittweise in den gewünschten Zielzustand überführen.

- Überwachung des Plans mit Sensoren
- bei Abweichung umplanen

- Motivation
- Planungsverfahren
  - Vorgehensweise
  - Suchverfahren
- Planung in der Robotik
- STRIPS

## Definitionen

- Die von einem Planer erzeugten Aktionssequenzen oder Graphen werden als **Plan** bezeichnet.
- Der Plan beschreibt im Zustandsraum eine Menge von **Zustandsänderungen**, die zum gewünschten Zielzustand führen.
- Diese Zustandsänderungen können **sequentiell, nebenläufig oder koordiniert nebenläufig** sein.
- Generell können Pläne in Teilpläne zerlegt werden
  - Plansegmente: vollständige Teilpläne
  - Planskelette: unvollständige Teilpläne
- Ein nicht weiter zerlegbares Plansegment bzw. der Zustandsübergang wird als Aktionsprimitiv oder **Planprimitiv** bezeichnet.

## Vorgehensweise

### 1. Ziel-Formulierung

- Gesucht: klare Definition, woraus das Ziel besteht
- Definition „Ziel“: Exakt die Menge von Weltzuständen, die das Ziel erfüllen

### 2. Problem-Formulierung

- Gesucht: Zustände und Aktionen, um zum Ziel zu kommen
- Frage der Abstraktion

## Vorgehensweise

### Hintergrundwissen

- Ohne Hintergrundwissen keine Planung möglich
- In Form von Karten, Handlungswissen, ...
- Verschiedene Sequenzen von Aktionen erzeugen:
  - Mögliche, gültige Sequenzen ermitteln
  - „Beste“ Sequenz auswählen

### 3. Suche

### 4. Ausführung

## Genereller Ablauf

while (true) do

Zustand := UPDATESTATE(Zustand, Perzeption)

Ziel := FORMULATEGOAL(Zustand)

Problem := FORMULATEPROBLEM(Zustand, Ziel)

Sequenz := SOLVEPROBLEMBYSEARCH(Problem)

EXECUTE(Sequenz)

end while

## Genereller Ablauf

- Im einfachen Fall ist ein Plan eine einfache Folge (oder ein azyklischer Graph) von Aktionen.
- Im allgemeinen Fall sind alle (aus einer Programmiersprache bekannten) Kontrollstrukturen möglich.
- Vorgehensweise



- Wiederholung
  - Wahrnehmung des momentanen Zustands der Umgebung
  - Generierung einer Handlungsfolge
  - Ausführung der nächsten Handlung

## Formale Definition von Problemen und Lösungen

- **Initialer Zustand**  $s^0$
- **Nachfolgerfunktion**, Liste möglicher Nachfolgezustände und Aktionen aus Zustand  $s^t$   
$$\text{SUCC-FN}(s^t) = \{(a^{t+1}, s^{t+1}); s^t \text{ geht in } s^{t+1} \text{ über mit Aktion } a^{t+1}\}$$
- **Zustandsraum**, als Graph:
  - Knoten: Zustände
  - Kanten: Zustandsübergänge, Aktionen
- **Pfad**
  - Folge von Zuständen, verknüpft durch Aktionen

## Formale Definition von Problemen und Lösungen

- **Zieltest**
  - Menge aller Zielzustände
  - Funktion (z.B. „Schachmatt“)
- **Pfadkosten**
  - Zeit, Wegstrecke, Kosten, ...
- **Schrittkosten**
  - Kosten entlang einer Kante
- **Lösung**
  - Pfad vom Startzustand zu einem Zielzustand
  - Optimal: geringste Pfadkosten

## Beispiele für Probleme

- 8-Damen Problem
- TSP (Travelling Salesman Problem)
- Türme von Hanoi
  
- VLSI-Layout
- Navigation für Roboter
- Automatic Assembly Sequencing

## Unterschiedliche Problemstellungen

Vollständiges Wissen ↔ Unvollständiges Wissen

Determinismus ↔ Nichtdeterminismus

Endlicher Zustandsraum ↔ Unendlicher Zustandsraum

Diskreter Zustandsraum ↔ Kontinuierlicher Zustandsraum

Statischer Zustandsraum ↔ Dynamischer Zustandsraum

- Motivation
- Planungsverfahren
  - Vorgehensweise
  - Suchverfahren
- Planung in der Robotik
- STRIPS

## Problemstellung

### Gegeben:

- Problem-Formulierung (Zustandsraum als Graph od. SUCC-FN)
- Ziel-Formulierung (Zielzustände od. Zieltest-Funktion)
- Initialer Zustand

### Gesucht:

Pfad vom initialen Zustand zum Ziel unter bestimmten Nebenbedingungen (z.B. minimale Kosten).

→ Suche im Zustandsraum notwendig

## Allgemeines Verfahren (I)

Baumsuche, Wurzelknoten ist Startzustand

Ablauf:

1. Vergleiche aktuellen Zustand mit Zielzustand
2. Expandiere aktuellen Zustand durch Anwenden der Nachfolgerfunktion  $SUCC-FN(x)$ 
  - Erzeugt neue Menge von Zuständen
3. Wähle einen der neuen Zustände aus (merke alte) und gehe zu 1

Bemerkungen:

- Suchbaum  $\neq$  Zustandsraum
- Knoten im Baum  $\neq$  Zustand
- Speichern der erzeugten Knoten als Liste

## Allgemeines Verfahren (II)

FUNCTION BaumSuche(Problem) RETURN Lösung oder Fehler

Liste.Insert(Startzustand)

WHILE (true) DO

    IF(Liste.IsEmpty())

        THEN RETURN Fehler

    Knoten = Liste.FirstElement()

    IF (ZielTest(Knoten, Problem))

        THEN RETURN Solution(Knoten)

    Liste.Insert(Expand(Knoten, Problem))

## Allgemeines Verfahren (III)

FUNCTION Expand(Knoten, Problem) RETURN Liste von Knoten

Rückgabeliste = Leere Liste

FOR EACH <Aktion, Ergebnis> IN SUCC-FN(Knoten.Zustand) DO

    Knoten s = neuer Knoten

    s.Zustand = Ergebnis

    s.Elternknoten = Knoten

    s.Aktion = Aktion

    s.Pfadkosten = Knoten.Pfadkosten + Schrittkosten(Knoten, Aktion, s)

    s.Tiefe = Knoten.Tiefe + 1

    Rückgabeliste.Add(s)

RETURN Rückgabeliste

## Bewertungskriterien

- Vollständigkeit
  - Wird Lösung gefunden, falls eine existiert?
- Optimalität
  - Wird optimale Lösung gefunden?
- Zeitkomplexität
- Raumkomplexität, Speicheraufwand

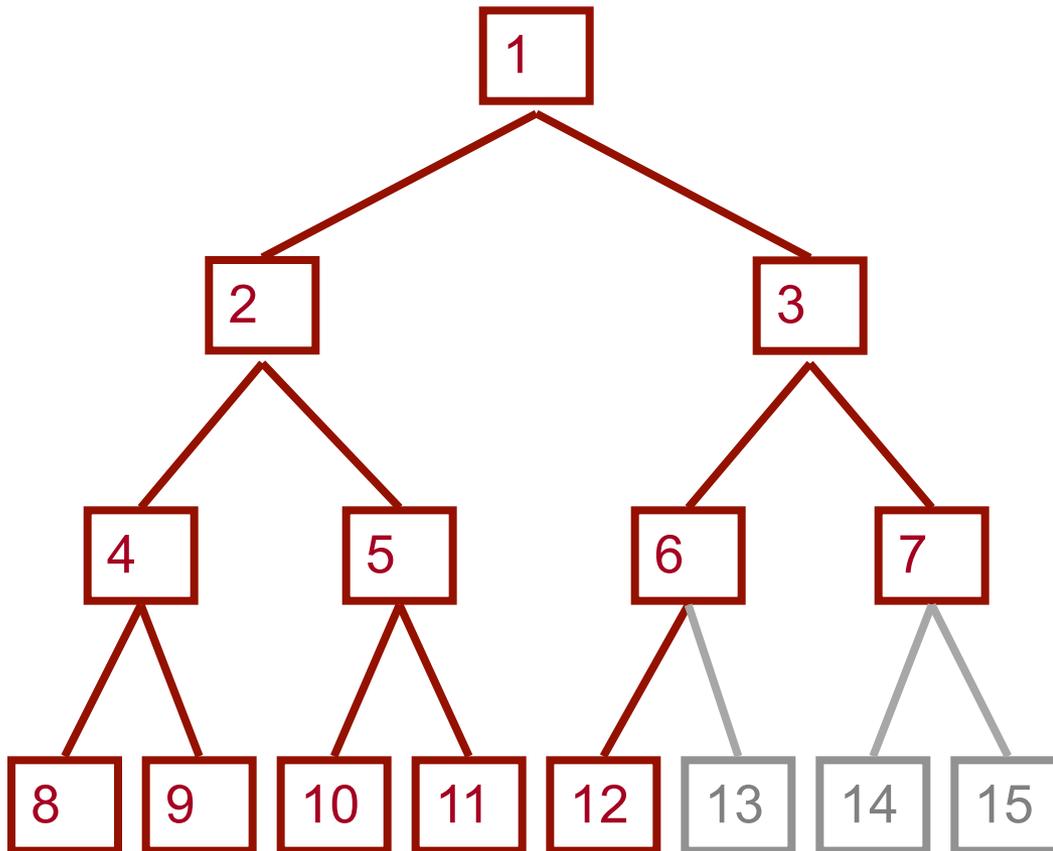
## Komplexitätsmaße:

- $b$ : Verzweigungsfaktor (branching factor)
- $d$ : Tiefe der Lösung (depth)
- $m$ : maximale Tiefe des Baums

## Übersicht

- Analytische Verfahren
  - Breitensuche
  - Tiefensuche
  - Iterative Deepening
  - Bidirektionale Suche
- Heuristische Verfahren
  - Dekomposition
  - A\*

## Breitensuche (I)

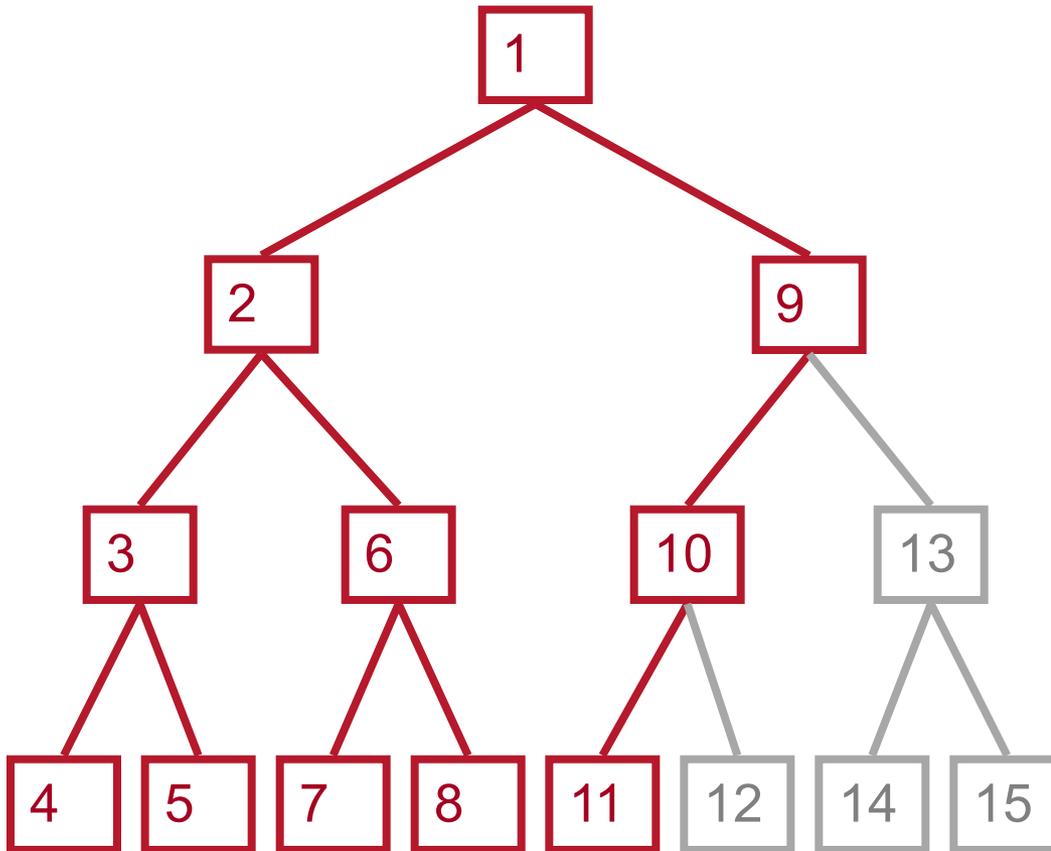


- Beginn am Ausgangszustand
- Prüfe alle Knoten gleicher Tiefe
- Gehe dann eine Stufe tiefer

## Breitensuche (II)

- Implementierung:
  - Liste als FIFO (neue Elemente hinten anhängen)
- Bewertung:
  - Vollständig
  - Optimal (falls alle Kantengewichte identisch)
  - Aufwand sehr hoch:  $O(b^{d+1})$

## Tiefensuche (I)

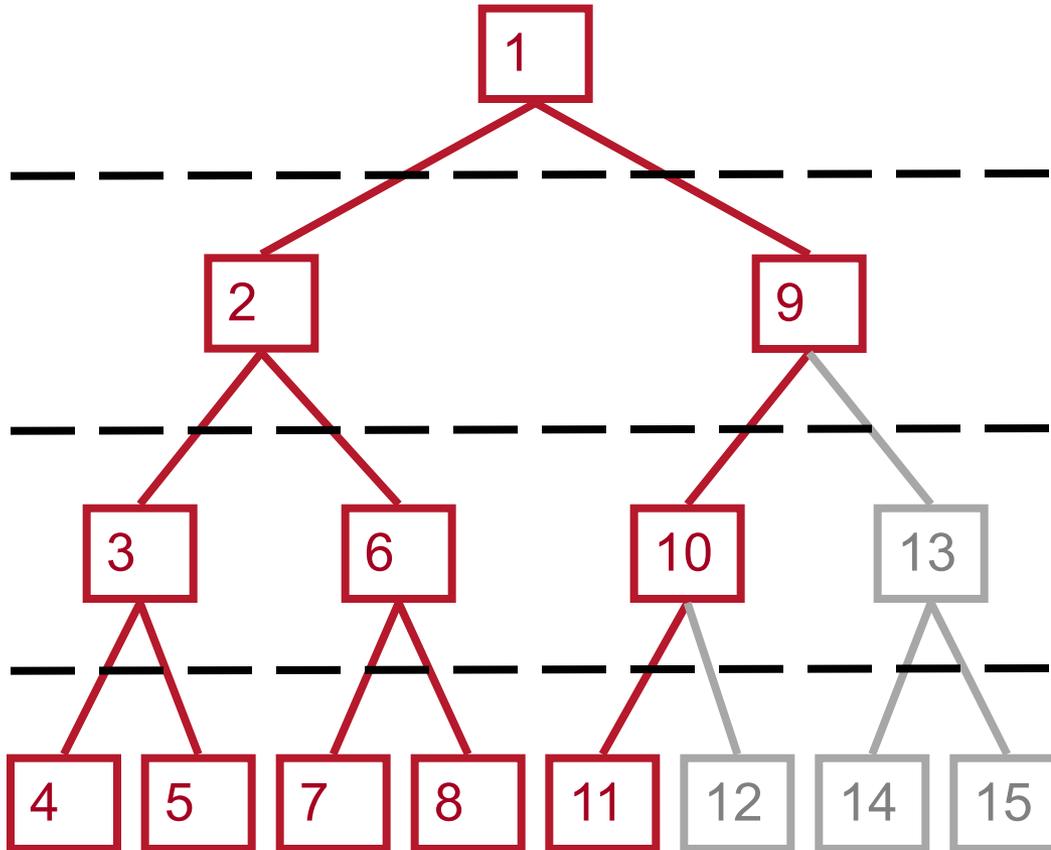


- Beginn am Ausgangszustand
- Suche bis zur maximalen Tiefe
- Weiter im tiefsten Knoten, der noch nicht-durchsuchte Kanten hat

## Tiefensuche (II)

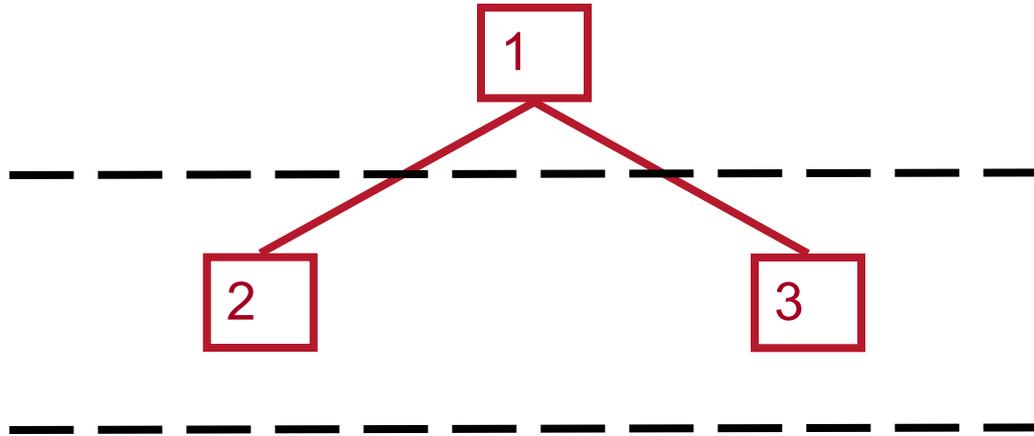
- Implementierung:
  - Liste als Stack (LIFO) (neueste Elemente vorne anhängen)
- Bewertung:
  - Vollständig? Ja und Nein (Zyklen).
  - Optimalität? Nein.
  - Aufwand:
    - Geringer Speicherbedarf
    - Hohe Laufzeit

## Iterative Deepening (I)



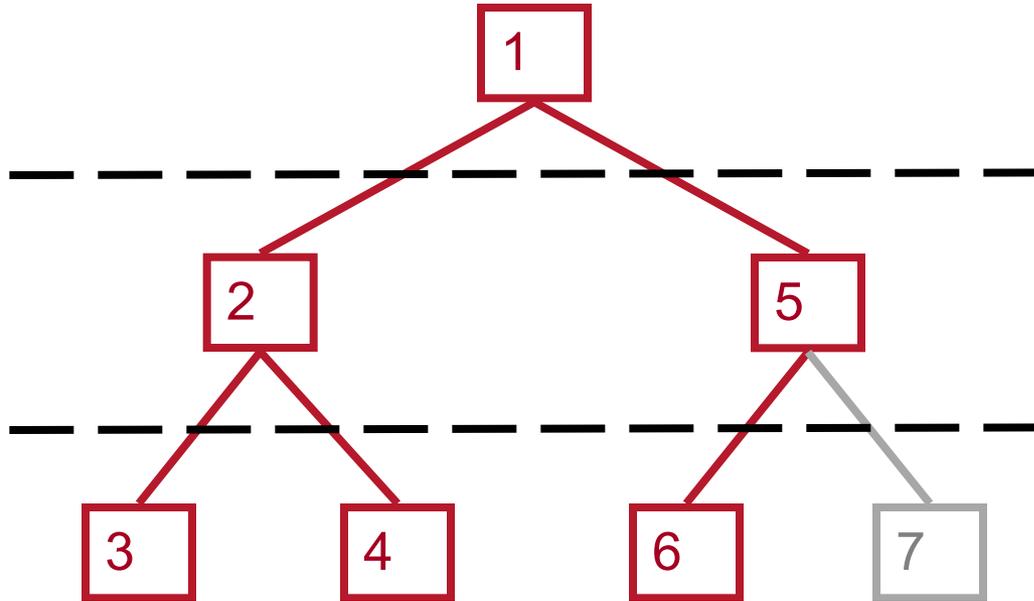
- Beginn am Ausgangszustand
- Tiefensuche bis zur Tiefengrenze
- Tiefengrenze inkrementieren, falls keine Lösung gefunden

## Iterative Deepening (II)



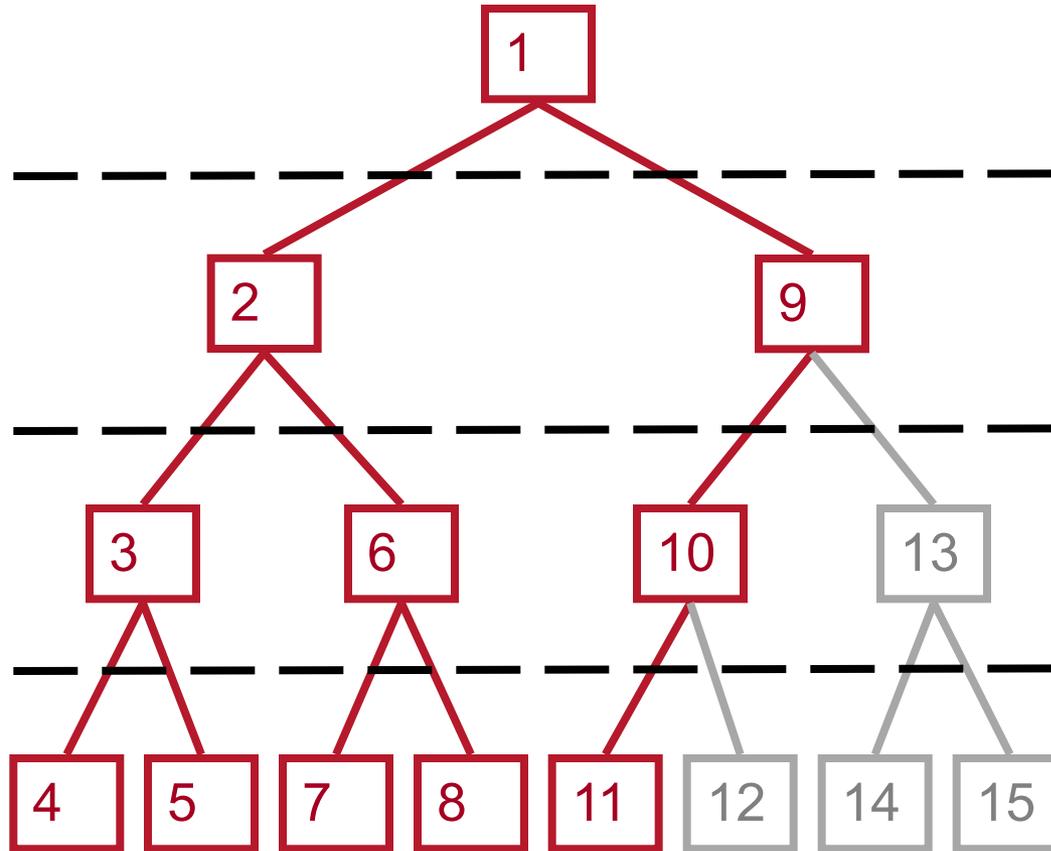
Tiefe 1

## Iterative Deepening (II)



Tiefe 2

## Iterative Deepening (II)

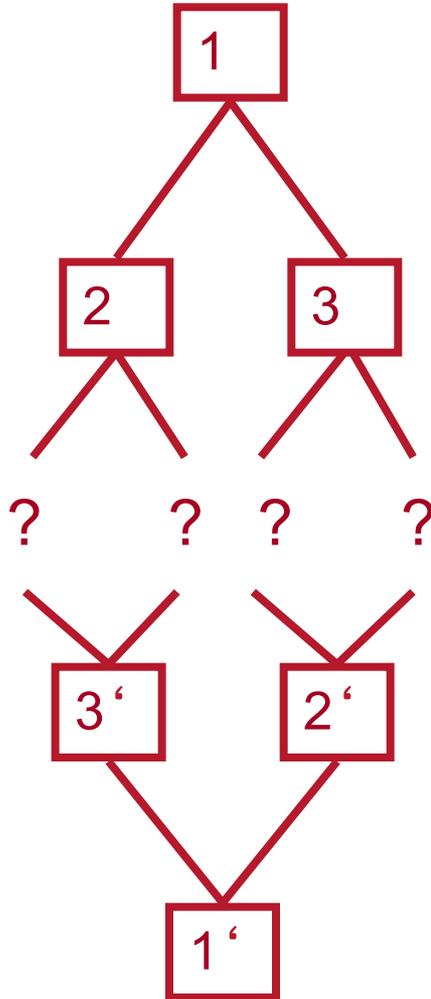


Tiefe 3

## Iterative Deepening (III)

- Erweiterung der Tiefensuche um maximale Suchtiefe
- Bewertung:
  - Methode der Wahl, wenn Suchraum groß und Tiefe a priori unbekannt

## Bidirektionale Suche



- Beginne eine Breitensuche am Ausgangszustand und eine weitere am gewünschten Zielzustand
- Halte, wenn bei beiden Suchen ein identischer Zustand erreicht wurde
- die Lösung setzt sich zusammen aus den beiden Pfaden, mit denen dieser identischer Zustand erreicht wurde
- Problem: Existenz einer Vorgängerfunktion (PRED-FN)

## Vergleich der analytischen Suchverfahren

	Breitensuche	Tiefensuche	Iterative Deepening	Bidirektionale Suche
Vollständigkeit	Ja	Nein	Ja	Ja
Laufzeit	$O(b^{d+1})$	$O(b^m)$	$O(b^d)$	$O(b^{d/2})$
Speicherbedarf	$O(b^{d+1})$	$O(bm)$	$O(bd)$	$O(b^{d/2})$
Optimal	Ja*	Nein	Ja*	-

- Legende:

- b: Verzweigungsgrad
- d: Tiefe der günstigsten Lösung
- m: max. Tiefe des Suchbaums

\* Falls Pfadkosten identisch in allen Kanten

Aus: Russel & Norvig:

Artificial Intelligence – A Modern Approach, 2nd Edition

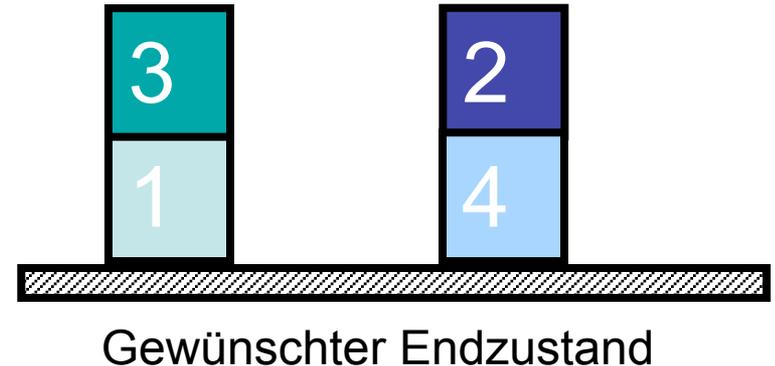
## Heuristiken

Möglich, wenn zusätzliches Wissen zugänglich ist, z.B.

- vollkommene Unabhängigkeit bestimmter Teilaufgaben (Dekomposition der Gesamtaufgabe)
- Schätzung der ‚Distanz‘ zwischen einem Zwischenzustand und dem gewünschten Endzustand (A\*-Suche)
- Ignoranz irrelevanter Informationen und Ausschluss von Aktionen, die keine Annäherung zum Zielzustand bringen

## Dekomposition

- Hier: Zwei unabhängig voneinander erreichbare Teilaufgaben
- Lösung:
  - Finde einen Plan für jede Teillösung
- Probleme:
  - Teillösungen können sich gegenseitig blockieren oder aufheben



## A\*-Verfahren (I)

### Eigenschaften

- Baumsuche
- Suche nicht systematisch entlang der Baumstruktur
- Heuristik  $h(n)$ 
  - Funktion, die die ‚Distanz‘ (Kosten) des Zustandes  $n$  zum Zielzustand schätzt
- Funktion  $g(n)$ 
  - Ermittelt die tatsächlichen Kosten vom Ausgangszustand zum Zustand  $n$
- Suche jeweils in dem Knoten fortsetzen, in dem  $f(n) = g(n) + h(n)$  minimal ist

## A\*-Verfahren (II)

Ziel:

Finde einen Weg von Feld 2  
nach Feld 13 (nur waagrecht  
und senkrecht)

Wegkosten:

nach grau = 1  
nach gelb = 4

Heuristik:

$h(n)$  = Manhattan-Distanz von  $n$   
nach 13 (z.B.  $h(11) = 2$ )

- $g(n)$  = tatsächliche Wegkosten  
des zurückgelegten Weges

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15

- Motivation
- Planungsverfahren
- Planung in der Robotik
- STRIPS

Um zu einer geeigneten Repräsentation des Hintergrundwissens zu gelangen müssen verschiedene Bedingungen berücksichtigt werden:

- Rand- und Vorbedingungen
- Nebenbedingungen

➔ Notwendigkeit der Modellierung dieser Bedingungen

## Vorbedingungen

Nicht alle Aktionen in bestimmten Situationen möglich

### **Beispiel: „Greifen“**

Roboter kann Objekt greifen, wenn

- Objekt zugänglich ist
- Masse des Objekts vom Roboter bewegt werden kann
- Greifer geöffnet ist
- Greifer sich am Greifpunkt des Objekts befindet
- Greifer zum Greifen des Objekts geeignet ist

## Nebenbedingungen

Bestimmte Restriktionen müssen eingehalten werden

### Beispiel: „Montageplanung“

- min. Bearbeitungszeit oder in fester Zeitspanne
- möglichst wenig Werkzeugwechsel
- mechanische Stabilität der teilmontierten Komponenten
- Verbindungsform (z.B. Einhalten der Abbindzeiten beim Kleben)
- Materialeigenschaften (z.B. verformbar)
- Beobachtbarkeit (z.B. Kamera)
- Genauigkeit bei der gegenseitigen Positionierung von Teilen

- Motivation
- Planungsverfahren
- Planung in der Robotik
- STRIPS

## STRIPS:

Stanford Research Institut Problem Solver

Entwickelt 1971 von Fikes und Nilsson

## Sprache aus 2 Komponenten:

- Sprache zur Beschreibung der Welt  
(Zustandssprache)
- Sprache zur Beschreibung, wie sich die Welt ändert  
(Operatorsprache)

## STRIPS - Zustandssprache

- **Relationale und konstante Symbole**

Beispiel:  $\text{auf}(a, b)$

$\text{auf}(,)$ : Relation mit Wertigkeit 2

$a, b$ : Konstanten

- Konstanten heißen auch **Terme**
- **Atom:**  
 $p(t_1, \dots, t_i)$ , mit  $p$  Relation,  $t_1, \dots, t_i$  Konstanten
- **Formeln:**  
Konjunktion von Atomen
- Konstanten beschreiben Objekte der Welt
- Relationen beschreiben Zustände der Welt

## STRIPS - Operatorsprache

- Zustände werden über **Operatoren**  $o$  oder über aus Operatoren bestehende **Pläne**  $p$  verändert
- Ein Operator ist ein Quadrupel  $o = (u, cond, add, sub)$ 
  - *Bezeichnung*  $u$ : semantische Bedeutung des Operators für die Zustandsänderung
  - *Vorbedingung*  $cond$ : logische Formel, ob für aktuellen Zustand Operator anwendbar ist
  - *Eigenschaftsadditionsmenge*  $add$ : zusätzliche Formeln des neuen Zustands gegenüber ursprünglicher Zustands
  - *Eigenschaftssubtraktionsmenge*  $sub$ : fehlende Formeln im neuen Zustand gegenüber ursprünglicher Zustand

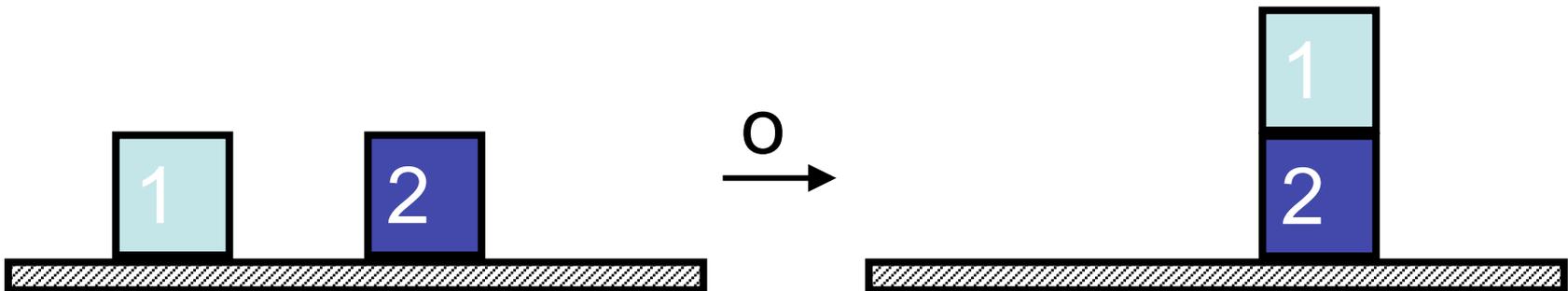
## Beispiel: Anwendung eines Operators (I)

- Aktueller Zustand:

$$x(y) = \{frei(1), frei(2)\}$$

- Operator-Quadrupel:

$$o = ( lege\_auf(1,2), \{frei(1),frei(2)\}, \{frei(1),auf(1,2)\}, \{frei(2)\} )$$

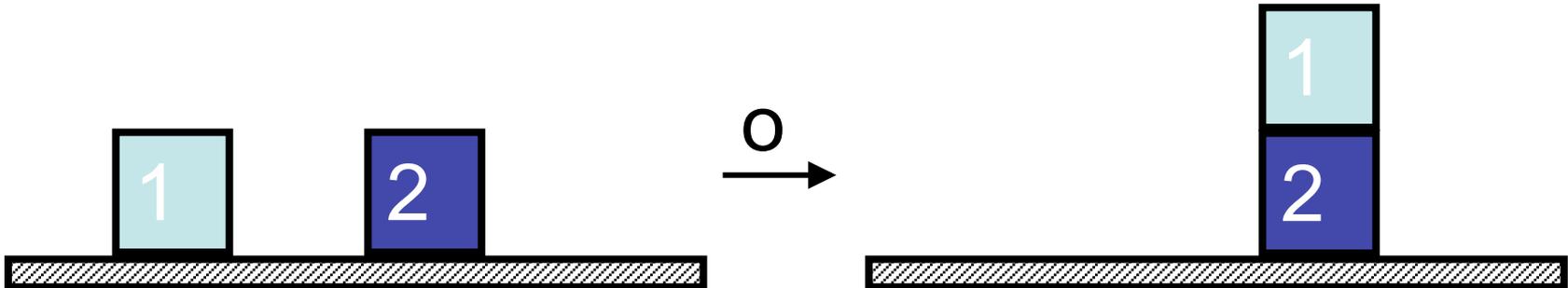


## Beispiel: Anwendung eines Operators (II)

$o = ( lege\_auf(1,2), \{frei(1),frei(2)\}, \{frei(1),auf(1,2)\}, \{frei(2)\} )$

- *lege\_auf(1,2)* darf angewendet werden, da  $\{frei(1),frei(2)\}$  zutrifft
- Es gilt weiterhin *frei(1)*, zusätzlich gilt *auf(1,2)*
- *frei(2)* gilt nicht mehr

Das Ergebnis der Anwendung des Operators ist:  $x(y) = \{frei(1), auf(1,2)\}$



## Eigenschaften der STRIPS - Operatoren

- In der *Additionsliste* müssen alle Aussageformeln stehen, die im neuen Zustand als geltend feststellbar sein müssen.  
Würde *frei(1)* fehlen, könnte *lege\_auf(1,2)* in einen Zustand führen, in dem *frei(1)* nicht gilt.
- In der *Subtraktionsliste* (auch *Delete-Liste*) müssen alle Aussageformeln stehen, die im neuen Zustand nicht mehr als geltend feststellbar sein dürfen.

## Problemdefinition mit STRIPS

Problem  $P$  in STRIPS ist gegeben durch Tupel:

$$P = \langle L_S, O_S, I_S, G_S \rangle$$

mit

$L_S$  : Zustandssprache

$O_S$  : definierte Operatoren (Operatorsprache)

$I_S$  : Menge von Atomen, die den initialen Zustand beschreiben

$G_S$  : Menge von Atomen, die die Zielzustände beschreiben

## Zustandsraummodell (I)

Zustandsraummodell eines Problems  $P$  mit STRIPS:

$$S(P) = \langle S, s_0, S_G, A, next \rangle$$

mit

$S$  : endliche Menge von Zuständen

$s_0 \in S$  : Startzustand

$S_G \subseteq S$  : nichtleere Menge von Zielzuständen

$A(s)$  : Menge von Aktionen, die auf den Zustand  $s$  anwendbar sind

$next$  : Übergangsfunktion, die den Zustand  $s$  in den Nachfolgezustand  $s_a = next(a, s)$  überführt für jede Aktion  $a \in A(s)$

## Zustandsraummodell (II)

- Lösung des Modells ist eine Sequenz von Aktionen  $a_0, a_1, \dots, a_m$ , die den Startzustand  $s_0$  in einen Zielzustand  $s \in S_G$  überführen
- Die Sequenz generiert eine Folge von Zuständen  $s_i, i = 0, \dots, n+1$  mit  $s_{i+1} = \text{next}(a_i, s_i)$ ,  $a_i \in A(s_i)$  und  $s_{n+1} \in S_G$

## Planen mit STRIPS

Abbildung des Planungsproblems  $P$  auf ein Zustandsraummodell

$$P = \langle L_S, O_S, I_S, G_S \rangle \Rightarrow S(P) = \langle S, s_0, S_G, A, next \rangle$$

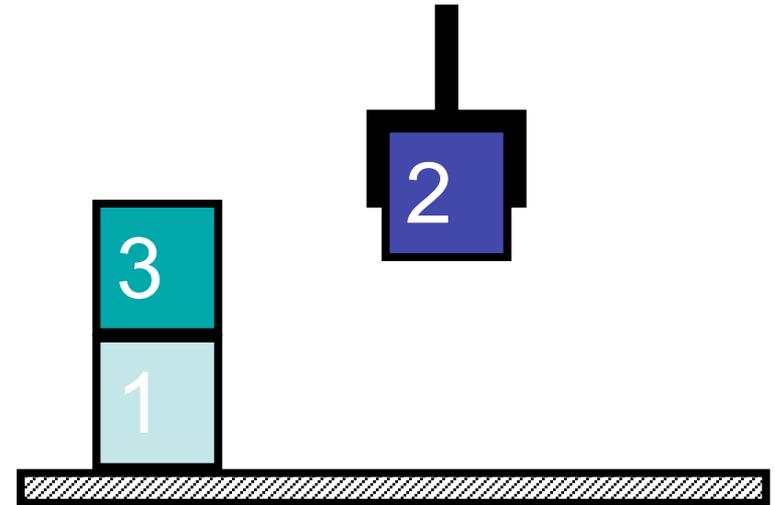
mit

- Zustände  $s$  sind Mengen von Atomen aus  $L_S$
- Der Startzustand  $s_0$  ist  $I_S$
- Die Zielzustände sind alle Zustände  $s$  mit  $G_S \subseteq s$
- $A(s)$  ist die Teilmenge von Operatoren  $op \in O_S$ , mit  $Prec(op) \subseteq s$
- Für die Überföhrungsfunktion  $next$  gilt:  
 $next(a, s) = s + Add(a) - Del(a)$ , für  $a \in A(s)$

## Beispiel (I)

Operator  $o = ( lege\_auf(X, Y), \{frei(X) \wedge frei(Y)\}, \{frei(X) \wedge auf(X, Y)\}, \{frei(Y)\} )$

- Handlungschema  
 $lege\_auf(X, Y)$
- Vorbedingungen  
 $frei(X) \wedge frei(Y)$
- Effekt
  - Add =  $\{ frei(X) \wedge auf(X, Y) \}$
  - Delete =  $\{ frei(Y) \}$



## Beispiel (II)

Aktueller Zustand:

$auf(Block1, Tisch) \wedge$

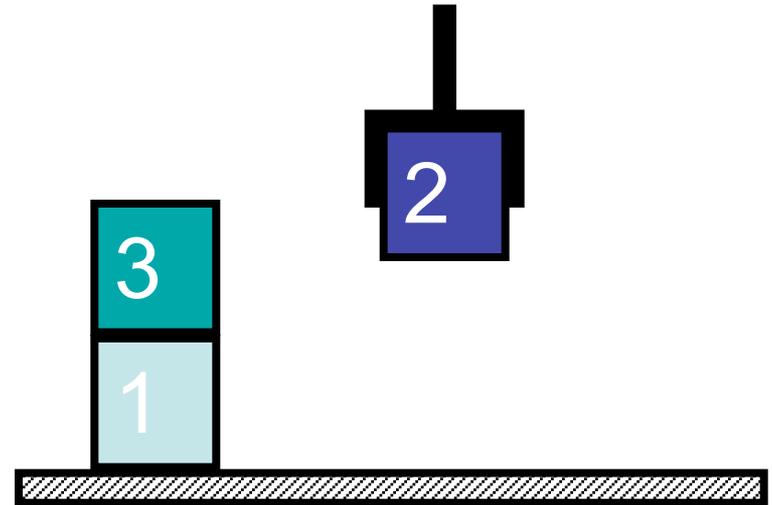
$auf(Block3, Block1) \wedge$

$frei(Block3) \wedge frei(Block2)$

Handlungsanweisung:

$lege\_auf(Block2, Block3)$

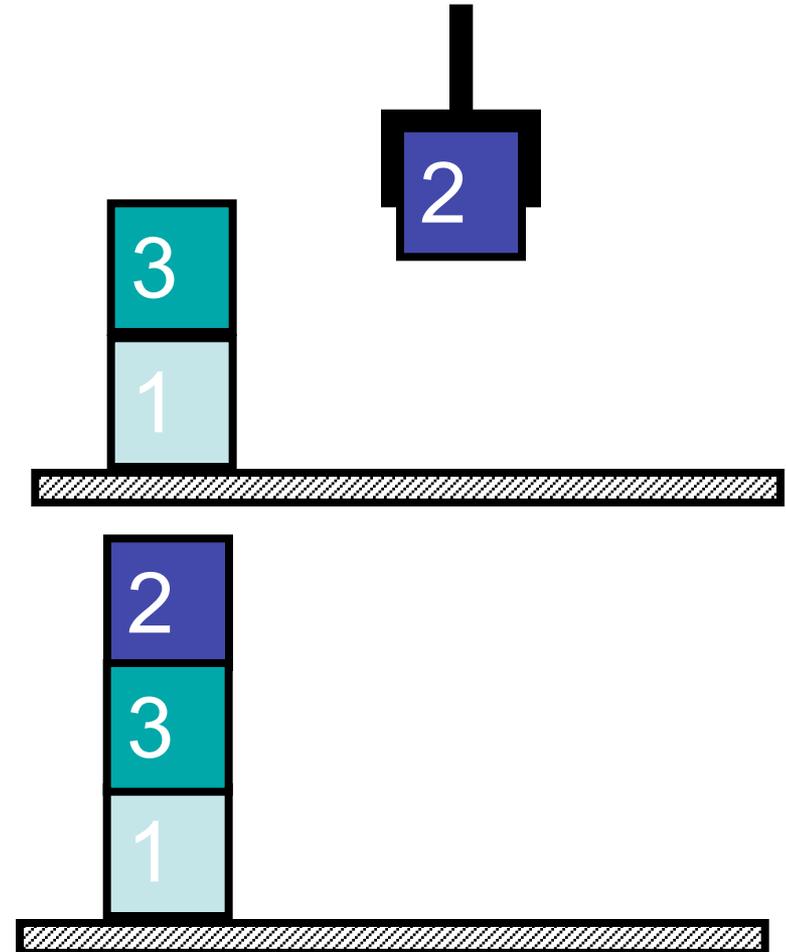
Zielzustand ?



## Beispiel (III)

Zielzustand:

$\text{auf}(\text{Block1}, \text{Tisch}) \wedge$   
 $\text{auf}(\text{Block3}, \text{Block1}) \wedge$   
 $\text{auf}(\text{Block2}, \text{Block3}) \wedge$   
 $\text{frei}(\text{Block2})$



- Russel, Norvig, „Artificial Intelligence“, Kapitel 3,4
- Fikes, Nilsson, „STRIPS: A new approach to the application of theorem proving to problem solving“, In *Artificial Intelligence*. Vol. 1, Seiten 27-120, 1971
- Geffner, „Functional STRIPS: A more flexible language for planning and problem solving“

- Praxisvorlesung nächste Woche zeigt auch komplexere Systeme
- Bachelor-/Masterarbeiten!
- Fachpraktika!
- VL Robotik 2, Robotik 3 im SoSe