

"The city of Salzburg as a sketch" als Eingabe zu to **DALL-E Mini / OpenAI DALL-E**

<https://huggingface.co/spaces/dalle-mini/dalle-mini>



AI Eingangswerkstatt

Roland Kwitt, Wolfgang Trutschnig



**Herzlich Willkommen zum
Bachelorstudium Artificial Intelligence**

Administratives zur Lehrveranstaltung

Administratives zur Lehrveranstaltung (LV)

Lehrveranstaltungsleiter

Webex

Univ.-Prof. Dr. Roland Kwitt

Professor für Maschinelles Lernen

Vorsitzender Curricularkommision

stv . Fachbereichsleiter FB AIHI



VO Teil der Lehrveranstaltung

Univ.-Prof. Dr. Wolfgang Trutschnig

Professor für Statistik / Stochastik

Direktor IDA Lab Salzburg

stv . Fachbereichsleiter FB AIHI



UE Teil der Lehrveranstaltung

Administratives zur Lehrveranstaltung (LV)

Personalia (Kwitt)

Email: roland.kwitt@plus.ac.at

Web: <http://rkwitt.org> (weiter zu “Teaching”)

Büro: Raum 1.18b (Jakob-Haringer Str. 2, Itzling, 1.ter Stock)

Sprechstunden: auf Vereinbarung



Administratives zur Lehrveranstaltung (LV)

- **Abhaltung:**
 - Vorlesung (VO, 536.101): Montags **9:00 - 10:30** (Itzling)
 - Übung (UE, 536.102): Donnerstags 16:00 - 18:00 (Hellbrunnerstrasse)
- **Benotung (VO Teil):** Prüfung am Ende des Semesters
- Die VO ist Teil der **STEOP** (Studieneingangs- und orientierungsphase)
- **Unterlagen** (auch in PLUS Online, sieht QR Code): <https://github.com/rkwitt/teaching>

Die Unterlagen zur Lehrveranstaltung (VO) sind die **Slides**.



Administratives zur Lehrveranstaltung (LV)

Anmerkungen zu den Folien

Auf den Folien (Teil Kwitt) sind Referenzen/Quellen meist in der Form (Autor(en), Jahr) angegeben; dazugehörige detaillierte Referenzen finden Sie als Fußnoten. Nahezu alle Referenzen sind im Internet frei verfügbar.

Administratives zur Lehrveranstaltung

Empfohlene Literatur (Hauptlehrbuch zur LV)

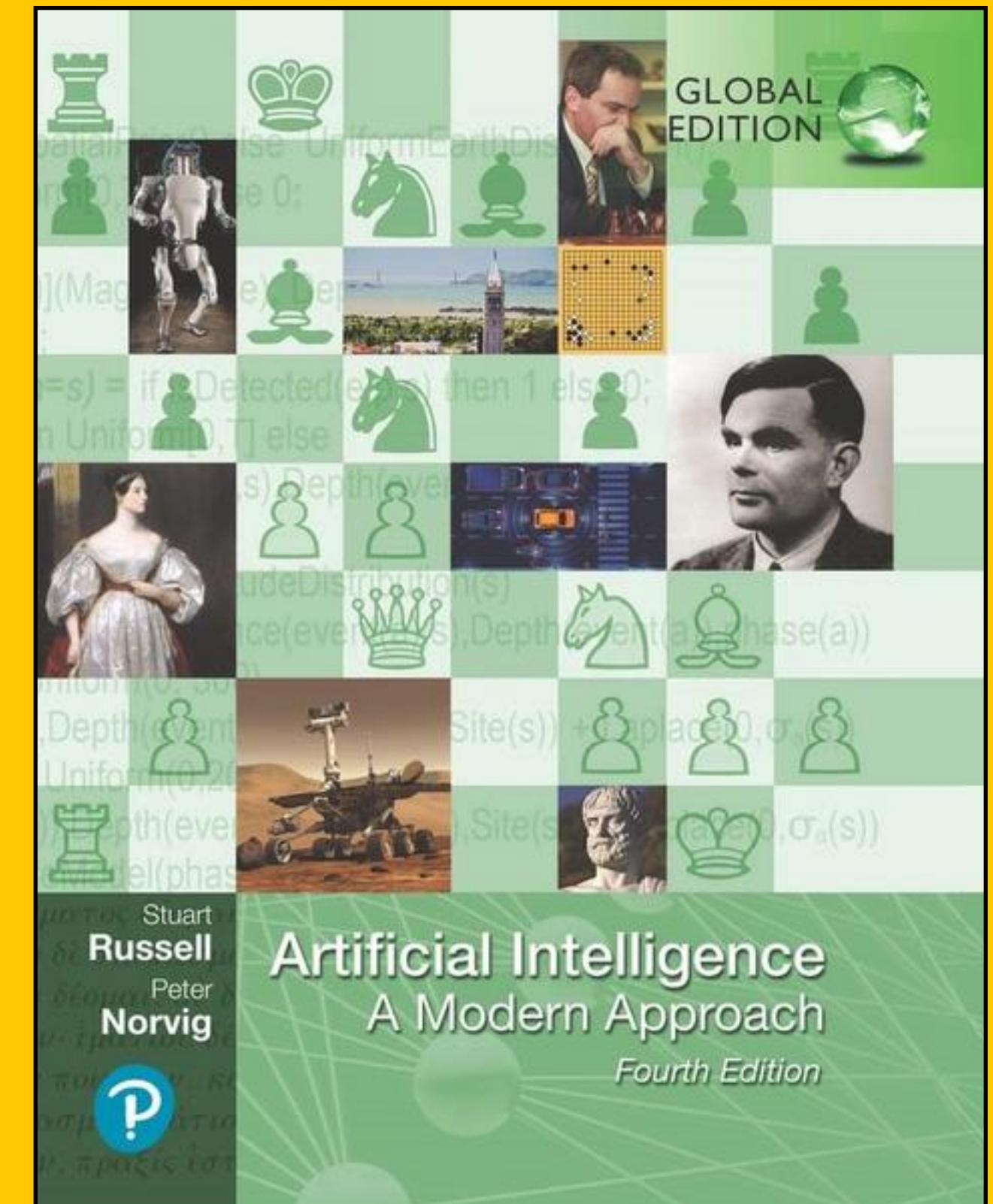
Stuart Russel & Peter Norvig

Artificial Intelligence - A Modern Approach (4th ed.)

Pearson Education Limited (2021)

es gibt auch eine Version in Deutsch, wir empfehlen
jedoch die 2021 erschienene Originalversion.

Auf den Folien als (RN) abgekürzt!



Aufbau / Ablauf des Studiums

Aufbau / Ablauf des Studiums

Einige Grunddaten

- **Bachelorstudium** (neu an der PLUS seit WS 2022/2023; Abschluss mit **BSc**)
- Studiendauer (Regelstudiendauer): **6 Semester**
- Studenausmaß: **180 ECTS**
- **Hauptinformationsquelle:** Curriculum (siehe QR Code)



Aufbau / Ablauf des Studiums

Stundenplan

Hier der aktuelle **Stundenplanvorschlag** für das **erste** Semester.

Weitere Informationen finden Sie im PLUS Online!

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
08:00 Uhr					
08:15 Uhr					
08:30 Uhr					
08:45 Uhr					
09:00 Uhr					
09:15 Uhr					
09:30 Uhr					
09:45 Uhr					
10:00 Uhr					
10:15 Uhr					
10:30 Uhr					
10:45 Uhr					
11:00 Uhr					
11:15 Uhr					
11:30 Uhr					
11:45 Uhr					
12:00 Uhr					
12:15 Uhr					
12:30 Uhr					
12:45 Uhr					
13:00 Uhr					
13:15 Uhr					
13:30 Uhr					
13:45 Uhr					
14:00 Uhr					
14:15 Uhr					
14:30 Uhr					
14:45 Uhr					
15:00 Uhr					
15:15 Uhr					
15:30 Uhr					
15:45 Uhr					
16:00 Uhr					
16:15 Uhr					
16:30 Uhr					
16:45 Uhr					
17:00 Uhr					
17:15 Uhr					
17:30 Uhr					
17:45 Uhr					
18:00 Uhr					
18:15 Uhr					
18:30 Uhr					
18:45 Uhr					
19:00 Uhr					

Aufbau / Ablauf des Studiums

STEOP

- Studieneingangs- und Orientierungsphase
- vermittelt einen Überblick über wesentliche Inhalte des Studiums
- **muss bestanden werden**, um weitere LV abschließen zu können
 - **Ausnahme**: LV im Ausmaß von 22 ECTS können vorgezogen werden (Informationen hierzu folgen noch)
- Im Bachelorstudium Artificial Intelligence besteht die STEOP* aus:
 - AI Eingangswerkstatt (VO, 2 ECTS)
 - Einführung in die Programmierung (VO, 3 ECTS)
 - Grundlagen der Mathematik (VU, 3 ECTS)

* die STEOP LV bei Studieneinstieg im Sommersemester sind unterschiedlich; siehe Curriculum

Aufbau / Ablauf des Studiums

LV Typen

- **Vorlesung (VO)**: Präsentation des Stoffs, Prüfungstermine nach Beendigung der LV
- **Übung (UE)**: Anwesenheitspflicht, Tests, Aufgaben vorführen / besprechen, Vorträge
- **Proseminar (PS)**: Anwesenheitspflicht, Tests, Aufgaben vorführen / besprechen, Vorträge
- **Übung mit Vorlesung (UV)**: Mischung aus VO+UE, Anwesenheitspflicht
- **Seminar (SE)**: Anwesenheitspflicht, eingehende Bearbeitung eines Themas mittels Vorträgen und Diskussionen

Eine genaue Beschreibung der verschiedenen LV Typen finden sie im [Curriculum](#).



Studienrecht

Studienrecht

Die folgenden Angaben beziehen sich auf das [Universitätsgesetz 2022](#) (kurz UG 02) und fassen einige (mMn) relevante Punkte zusammen.

Ich empfehle jedoch die entsprechend angegebenen Paragraphen zu lesen, da etwaige Nuancen aufgrund der verkürzten Darstellung möglicherweise nicht den juristisch geltenden Maßstäben entsprechen.

Studienrecht

Rechte und Pflichten der Studierenden

In **§ 59 (1)** heißt es „Den Studierenden steht nach Maßgabe der gesetzlichen Bestimmungen Lernfreiheit zu.“ Dies bedeutet unter anderem:

- Nach Maßgabe des Lehrangebotes und nach Maßgabe der Curricula kann zwischen dem Lehrpersonal ausgewählt werden.
- Facheinschlägige Lehr- und Forschungseinrichtungen und die Bibliothek an der Universität können nach Maßgabe der Benützungsordnungen benutzt werden.

Studienrecht

Rechte und Pflichten der Studierenden

Über die Pflichten gibt **§ 59 (2)** Auskunft:

- Namens- und Adressenänderungen sind unverzüglich bekanntzugeben.
- Die Fortsetzung des Studiums ist jedes Semester während der allgemeinen Zulassungsfrist (oder der Nachfrist) zu melden.
- Bei vorhersehbarer Studieninaktivität ist eine zeitgerechte Abmeldung vom Studium durchzuführen.
- Fristgerechte An- und Abmeldung zu den Prüfungen, nicht erscheinen ohne ordnungsgemäße Abmeldung: Ablegung frühestens nach 40 Kalendertagen möglich (Satzung der PLUS, § 15).

Studienrecht

Erlöschen der Zulassung

Die wesentlichen Gründe, damit die Zulassung erlischt sind (siehe **§ 68**):

- Abmeldung vom Studium
- Die Meldung der Fortsetzung des Studiums unterbleibt, ohne beurlaubt zu sein.
- Die letzte zulässige Wiederholung einer vorgeschriebenen Prüfung wird negativ beurteilt.
- Das Studium wurde durch die positive Beurteilung bei der letzten vorgeschriebenen Prüfung abgeschlossen.

Studienrecht

Feststellung des Studienerfolges (1)

Generell heißt es dazu im **§ 72**: „Der Studienerfolg ist durch die Prüfungen und die Beurteilung wissenschaftlicher Arbeiten (Master-, Diplomarbeiten und Dissertationen) festzustellen.“

Der positive Erfolg von Prüfungen und wissenschaftlichen Arbeiten ist mit „sehr gut“ (1), „gut“ (2), „befriedigend“ (3) oder „genügend“ (4), der negative Erfolg ist mit „nicht genügend“ (5) zu beurteilen ... wenn diese Form der Beurteilung bei Lehrveranstaltungsprüfungen unmöglich oder unzweckmäßig ist, hat die positive Beurteilung „mit Erfolg teilgenommen“, die negative Beurteilung „ohne Erfolg teilgenommen“ zu lauten.

Studienrecht

Feststellung des Studienerfolges (2)

Zur Ablegung einer Prüfung ist eine Anmeldung erforderlich (Internet, LV-Leiter, . . .). Die entsprechenden Zeugnisse sind bis spätestens vier Wochen nach Erbringung der zu beurteilenden Leistung auszustellen.

Studienrecht

Feststellung des Studienerfolges (3)

Prüfungen sind für **nichtig** zu erklären (**§ 73**) ...

- ... wenn die Anmeldung zur Prüfung erschlichen wurde.
- ... wenn die Beurteilung erschlichen wurde – insbesondere durch unerlaubte Hilfsmittel, unerlaubter Weise einer anderen Person bedienen, Daten/Ergebnisse erfunden/gefälscht, Plagiat).
- Die Prüfung, deren Beurteilung für nichtig erklärt wurde, ist auf die Gesamtanzahl der Wiederholungen anzurechnen.
- Prüfungen, die außerhalb des Wirkungsbereichs einer Fortsetzungsmeldung abgelegt wurden, sind absolut nichtig.

Studienrecht

Wiederholung von Prüfungen (1)

Positiv beurteilte Prüfungen können bis zwölf Monate nach der Ablegung einmal wiederholt werden. Die positiv beurteilte Prüfung wird mit dem Antreten zur Wiederholungsprüfung nichtig.

Negativ beurteilte Prüfungen können dreimal wiederholt werden. Ab der dritten Wiederholung einer Prüfung ist diese kommissionell abzuhalten, wenn die Prüfung in Form eines einzigen Prüfungsvorganges durchgeführt wird. Auf Antrag der Studierenden bzw. des Studierenden gilt dies auch für die zweite Wiederholung ([Satzung der PLUS](#), § 21 Absatz 1). Bei negativer Beurteilung der letzten Wiederholung der letzten Prüfung des Studiums sind die Studierenden berechtigt, diese ein weiteres Mal zu wiederholen.

Studienrecht

Wiederholung von Prüfungen (2)

Gegen die Beurteilung einer Prüfung ist kein Rechtsmittel zulässig (**§ 79**). Es kann jedoch eine **negativ beurteilte** Prüfung aufgehoben werden, wenn sie einen schweren Mangel aufweist. Ein entsprechender Antrag ist innerhalb von vier Wochen ab Bekanntgabe der Beurteilung einzubringen und der schwere Mangel glaubhaft zu machen.

Innerhalb von sechs Monaten ab Bekanntgabe der Beurteilung ist den Studierenden **Einsicht in die entsprechenden Unterlagen** zu gewähren, falls sie ihnen nicht ausgehändigt wurden.

Organisatorische Struktur der PLUS

Organisatorische Struktur der PLUS

Universitätsleitung — Rektorat

- Prof. Dr. Dr. h.c. **Hendrik Lehnert** (Rektor)
- Dr. Barbara Romauer (VR Finanzen)
- Ao.Univ.-Prof. Dr. Martin Weichbold (VR Lehre)
- Univ.-Prof. Dr. Nicola Hüsing (VR Forschung)

Organisatorische Struktur der PLUS

Fakultäten

Organisatorisch ist die PLUS in **6 Fakultäten** gegliedert:

DAS	GW	NLW	RWW	KW	KTH
Fakultät für Digitale und Analytische Wissenschaften	Gesellschaftswissenschaftliche Fakultät	Natur- und Lebenswissenschaftliche Fakultät	Rechts- und Wirtschaftswissenschaftliche Fakultät	Kulturwissenschaftliche Fakultät	Katholisch-Theologische Fakultät

An den Fakultäten sind die **Fachbereiche (FB)** angesiedelt.

Organisatorische Struktur der PLUS

Fachbereiche an der DAS Fakultät

An der DAS Fakultät gibt es aktuell **4 Fachbereiche (FB)**:

FB Artificial Intelligence & Human Interfaces (AIHI)	FB Informatik	FB Geoinformatik	FB Mathematik
--	------------------	---------------------	------------------

Das **Bachelorstudium Artificial Intelligence** wird vom FB AIHI betreut.

Organisatorische Struktur der PLUS

Zuständigkeit Curriculum

Für das Curriculum des BA Artificial Intelligence ist die **Curricularkommission** zuständig.

Aktuell setzt sich diese folgendermaßen zusammen (Anm.: ab Okt. 2022 neu):

- Univ.-Prof. Dr. Roland Kwitt (Vorsitz)
- Univ.-Prof. Dr. Wolfgang Trutschnig
- Univ.-Prof. Dr. Clemens Fuchs
- Dr. Mag. Ulrike Ruprecht
- Assoz. Prof. Dr. Ana Sokolova (stv. Vorsitzende)
- Dipl.-Ing. Bettina Sereinig
- Lea Maislinger, BSc
- Selina Milla, BSc
- Julius Sula

Was ist eigentlich künstliche Intelligenz?

Was ist eigentlich künstliche Intelligenz?

Es gibt **keine allgemein anerkannte Definition**. Folgende Definitionsansätze unterscheiden sich in ihrer jeweiligen Sichtweise (mit Überschneidungen).

orientiert am **menschlichen Denken**

[The automation of] activities that we associate with human thinking, activities such as decision-making, problem-solving, learning,

(Bellman, 1978)

Was ist eigentlich künstliche Intelligenz?

Würden wir uns an dieser Definition orientieren, müsste man unweigerlich auch menschliches Denken verstehen. Mögliche Ansätze hierzu sind

- psychologische Experimente
- Beobachtung von Denkprozessen anhand medizinischer Bildgebungsverfahren
- Selbstbeobachtung

Hätte man eine “ausreichend präzise Theorie” menschlichen Denkens, könnte man versuchen diese Theorie als Computerprogramm zu realisieren.

Beispiel: “General Problem Solver” ([Newell and Simon, 1961](#)) — Weniger am tatsächlich korrekten “Problemlösen” interessiert, als an der Sequenz von Schlussfolgerungen der Maschine im Vergleich zum Menschen.

([Newell and Simon, 1961](#)) . Newell A. and Simon, H.A. GPS, a program that simulates human thought.
In: Feigenbaum E. And Feldmann J., Computers and Thought (1995)

Was ist eigentlich künstliche Intelligenz?

orientiert am **menschlichen Handeln**

“The art of creating machines that perform functions that require intelligence when performed by people.”

(Kurzweil, 1990)

Beispiel: Turing Test (Turing, 1950)

Gegeben eine Menge von (**geschriebenen**) **Fragestellungen** (einer Befragungsperson) mit entsprechenden (**geschriebenen**) **Antworten** einer Maschine und eines Menschen, kann die Befragungsperson zwischen Mensch und Maschine unterscheiden?

Ist dies **nicht** möglich, könnte man argumentieren die Maschine besäße menschliches Denkvermögen.

Was ist eigentlich künstliche Intelligenz?

Obwohl der (klassische) Turing Test oft kritisiert wird, würde er zumindest folgende Fähigkeiten erfordern:

- Verstehen natürlicher Sprache (NLP - **natural language processing**)
- Repräsentation von Wissen in geeigneter Art und Weise (**knowledge representation**)
- Automatisierte Schlussfolgerungen (**automated reasoning**)
- Maschinelles Lernen um sich an neue Situationen anzupassen und zu extrapolieren (**machine learning**)

Was ist eigentlich künstliche Intelligenz?

orientiert am **rationalen Denken**

“The study of computations that make it possible to perceive, reason and act.”

(Winston, 1992)

Bereits Aristoteles versuchte “richtiges Denken” zu formalisieren anhand bestimmter Typen logischer Schlüsse (sog. **Syllogismen**). Bei gegebenen korrekten Prämissen (Vorraussetzungen), stellten diese Syllogismen quasi Muster dar anhand derer korrekte Schlüsse gezogen werden konnten.

Beispiel: Aus “Alle Menschen sind sterblich.” und “Alle Griechen sind Menschen.” folgt die Schlussfolgerung “Alle Griechen sind sterblich”.

Was ist eigentlich künstliche Intelligenz?

Logiker des 19. ten Jahrhunderts entwickelten eine präzise Notation für Aussagen über Objekte und Beziehungen zwischen diesen Objekten.

Der **Logizismus** hat im Großgebiet der künstlichen Intelligenz eine starke Tradition, mit der Hoffnung intelligente Systeme auf dem Fundament der Logik zu erschaffen.

Konventionell gesehen würde dies jedoch (zweifelsfreies!) Wissen über die Welt erfordern, also eine Anforderung benötigen die wohl kaum jemals erfüllt werden kann. Die **Wahrscheinlichkeitstheorie** füllt diese Lücke und erlaubt Schlussfolgerungen auf Basis unsicherer Information.

Was ist eigentlich künstliche Intelligenz?

orientiert am **rationalen Handeln**

“Computational Intelligence is the study of the design of intelligent agents.”

(Poole et al., 1998)

Führt zur Definition von **Agenten**, also im Wesentlichen autonomen Systemen die (1) handeln, (2) ihre Umgebung wahrnehmen, (2) über eine gewisse Zeit bestehen, (3) sich ihrer Umgebung anpassen und (4) gewisse Ziele verfolgen.

Rationale Agenten handeln in einer Art und Weise, um das beste Ergebnis zu erzielen, oder — in Anbetracht unsicherer Informationen — **in Erwartung** das beste Ergebnis zu erzielen.

Fundament & Teilgebiete

Fundamentale Ideen zur künstlichen Intelligenz finden sich in diversen Wissenschaftsgebieten, wie

- Philosophie
- Mathematik
- Informatik
- Wirtschaftswissenschaften (economics)
- Regelungstheorie (control theory)
- Linguistik
- Neurowissenschaften (neuroscience)
- etc.

Historie

siehe Kapitel 1.3 in (RN)

State-of-the-Art

siehe [aiindex.org](#) (Artificial Intelligence Index Report 2022)

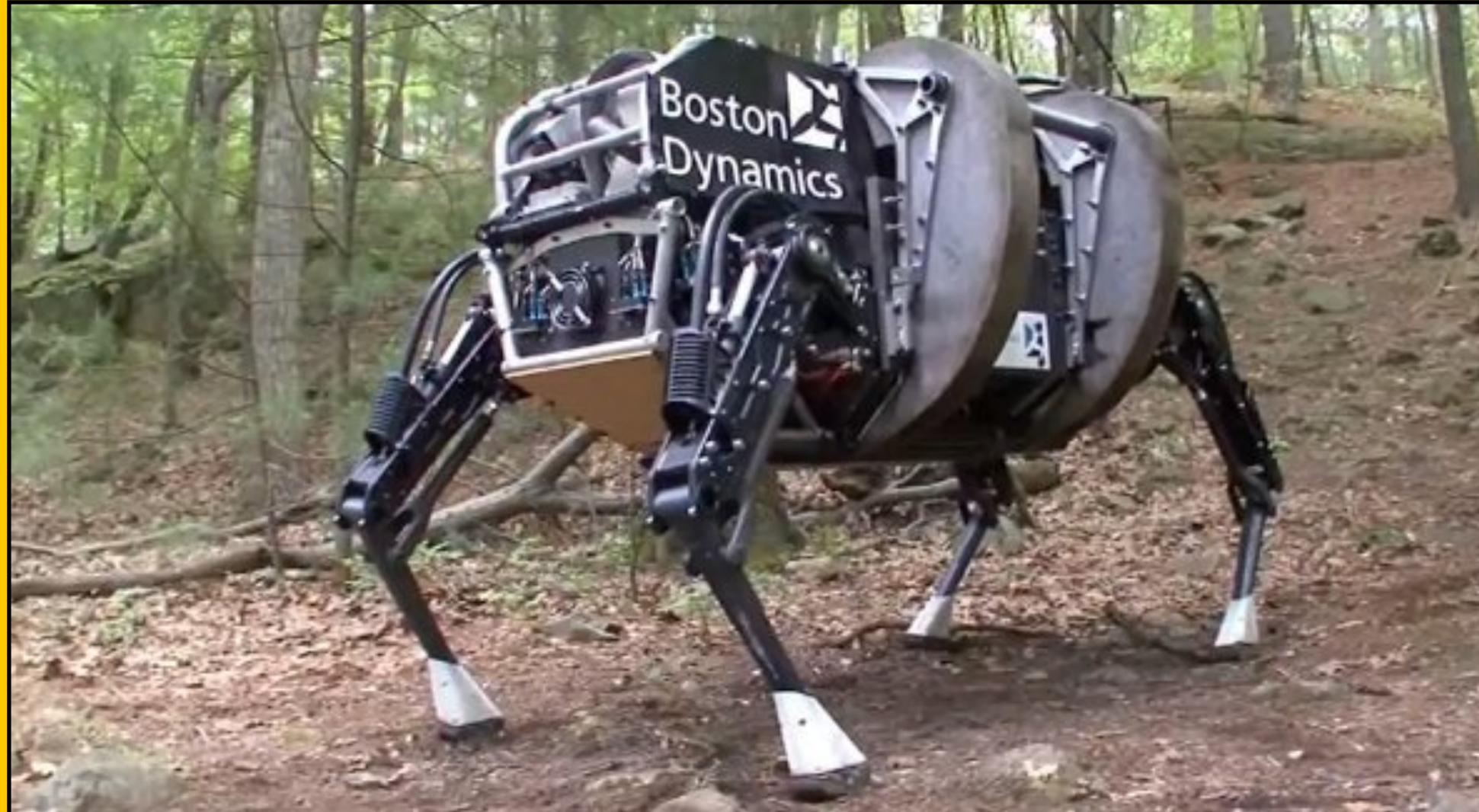
Einige **Key-Facts** (möglicherweise bereits heut schon *out-of-date* :)

- Anstieg an Publikationen (verdoppelt) im Großgebiet AI zwischen 2010 und 2021
- Enormer Anstieg an AI Startups in den USA
- “Computing Power” im Kontext von AI Systemen verdoppelt sich alle **3.4** Monate
- “Human-Level Performance” bereits im Kontext von Objekterkennungsproblemen, Go, Poker, Schach, Atari Spiele, Haut- u. Prostatakrebs Erkennung, Proteinfaltung, etc.

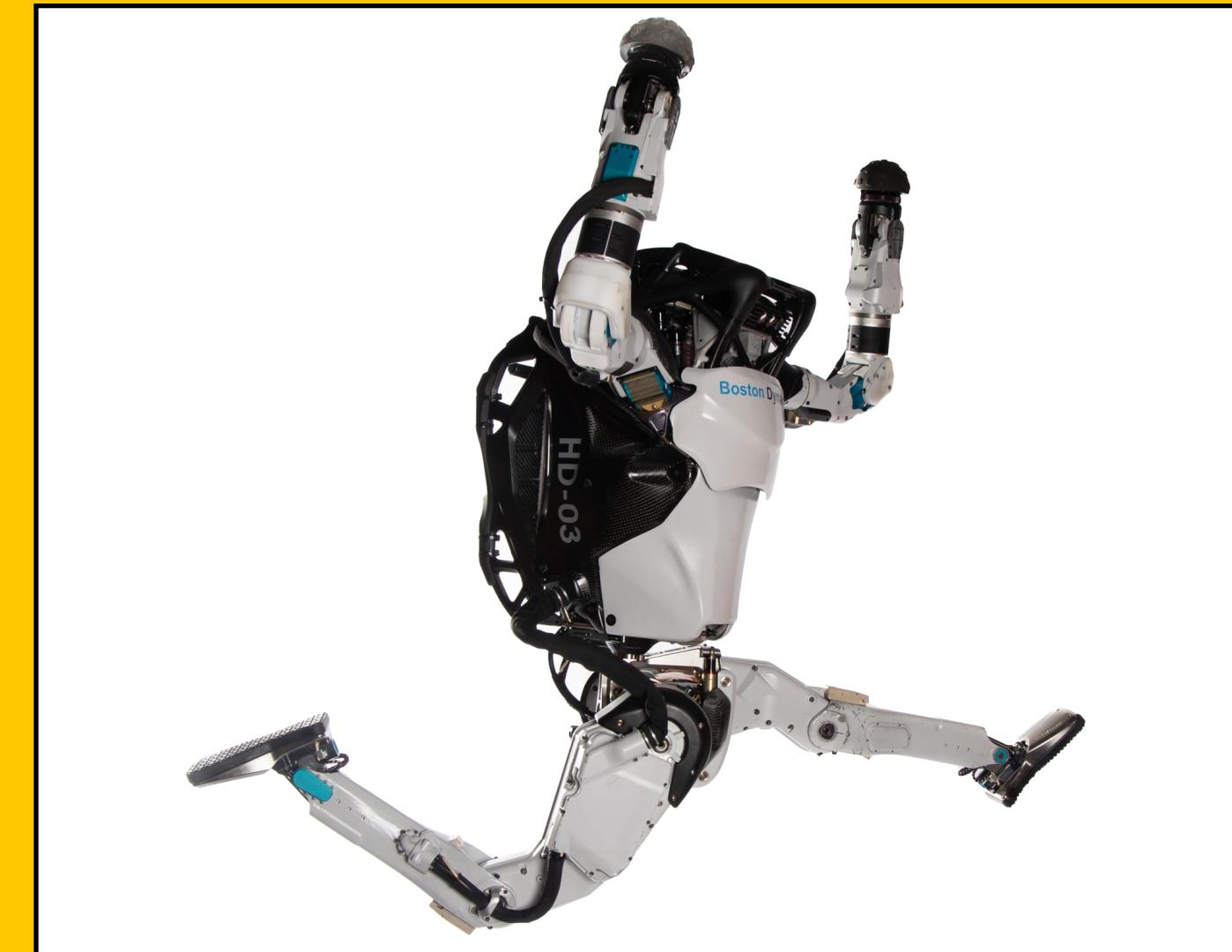
State-of-the-Art

Beispiele

- Fortbewegung auf Beinen (Legged Locomotion)



BigDog (Robert et al., 2008)

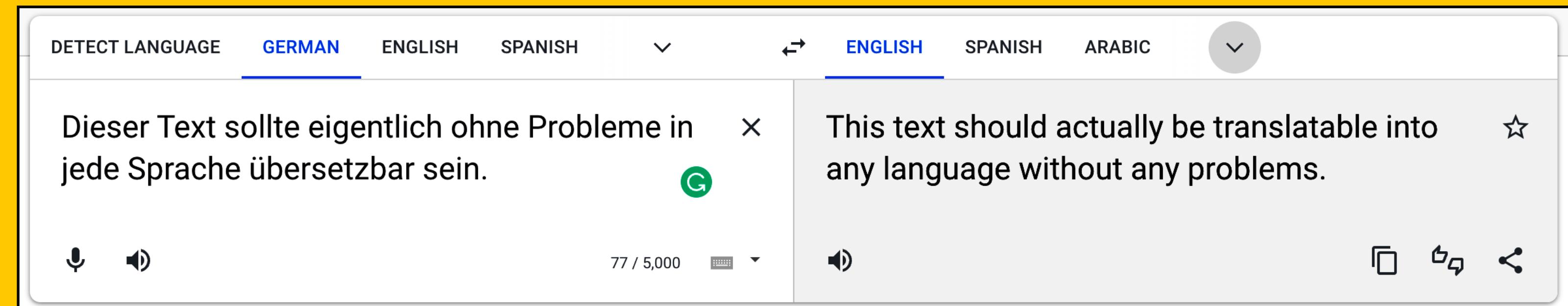


Atlas (Ackermann and Guizzo, 2016)

State-of-the-Art

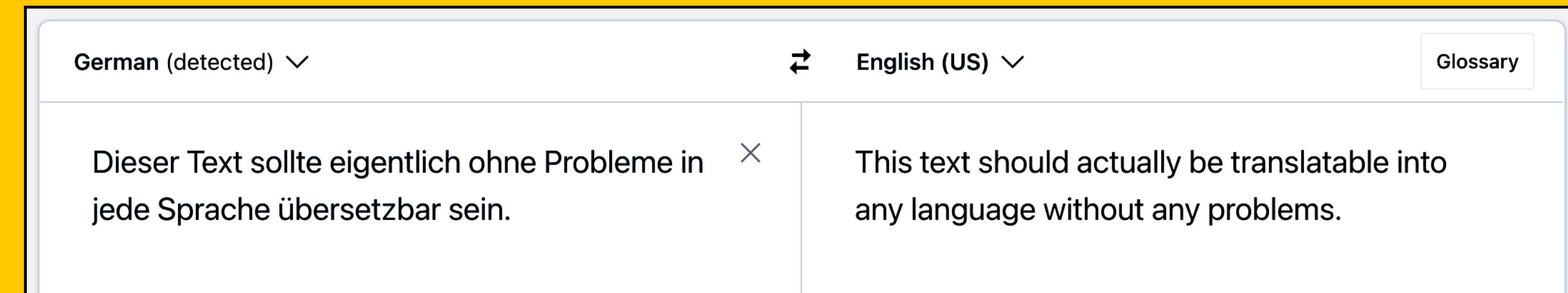
Beispiele

- Übersetzung (Machine Translation, Natural Language Processing)



The screenshot shows the Google Translate web interface. The source text "Dieser Text sollte eigentlich ohne Probleme in jede Sprache übersetzbar sein." is in German. The target language is English (US). The translated text "This text should actually be translatable into any language without any problems." is displayed. The interface includes language detection, a glossary button, and sharing options.

Quelle: translate.google.com (Sep. 2022)



The screenshot shows the DeepL web interface. The source text "Dieser Text sollte eigentlich ohne Probleme in jede Sprache übersetzbar sein." is in German (detected). The target language is English (US). The translated text "This text should actually be translatable into any language without any problems." is displayed. The interface includes a glossary button.

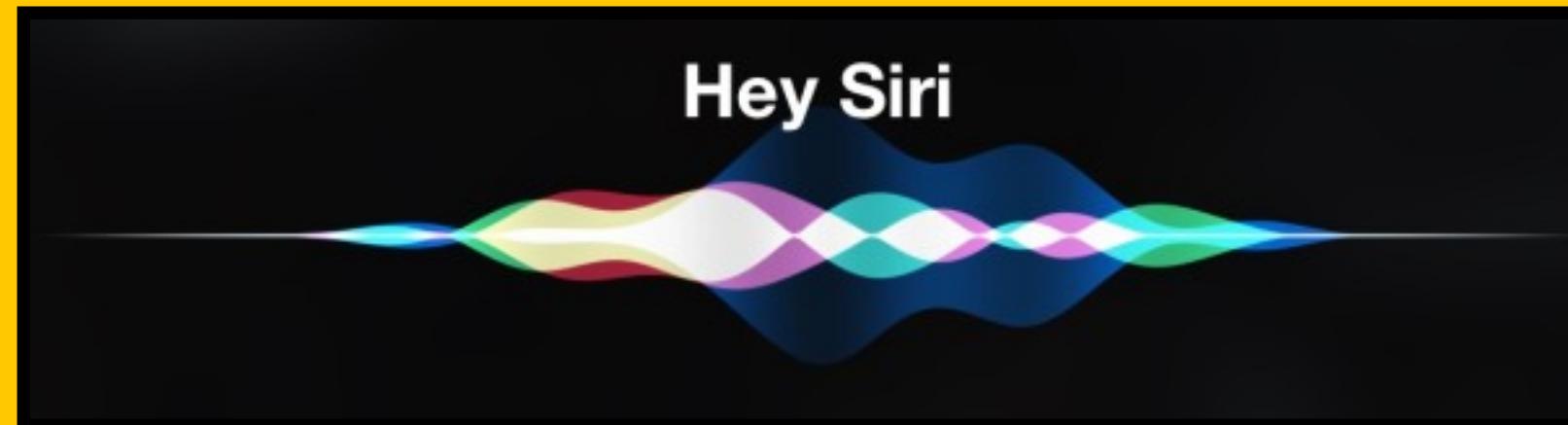
Quelle: deepl.com (Sep. 2022)

State-of-the-Art

Beispiele

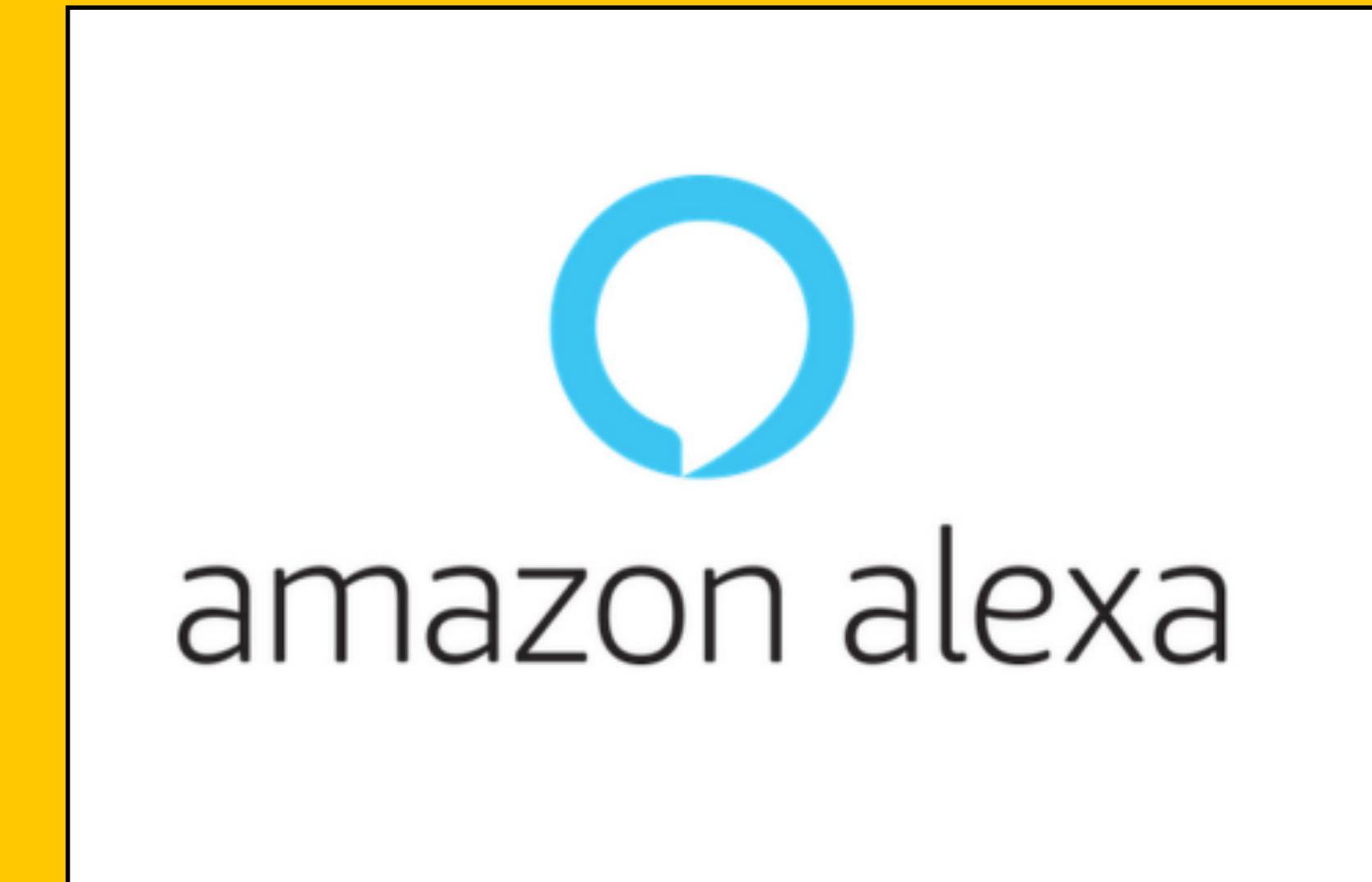
- Spracherkennung (Speech Recognition)

Apple Siri



Quelle: osxdaily.com

Amazon Alexa



Quelle: developer.amazon.com

State-of-the-Art

Beispiele

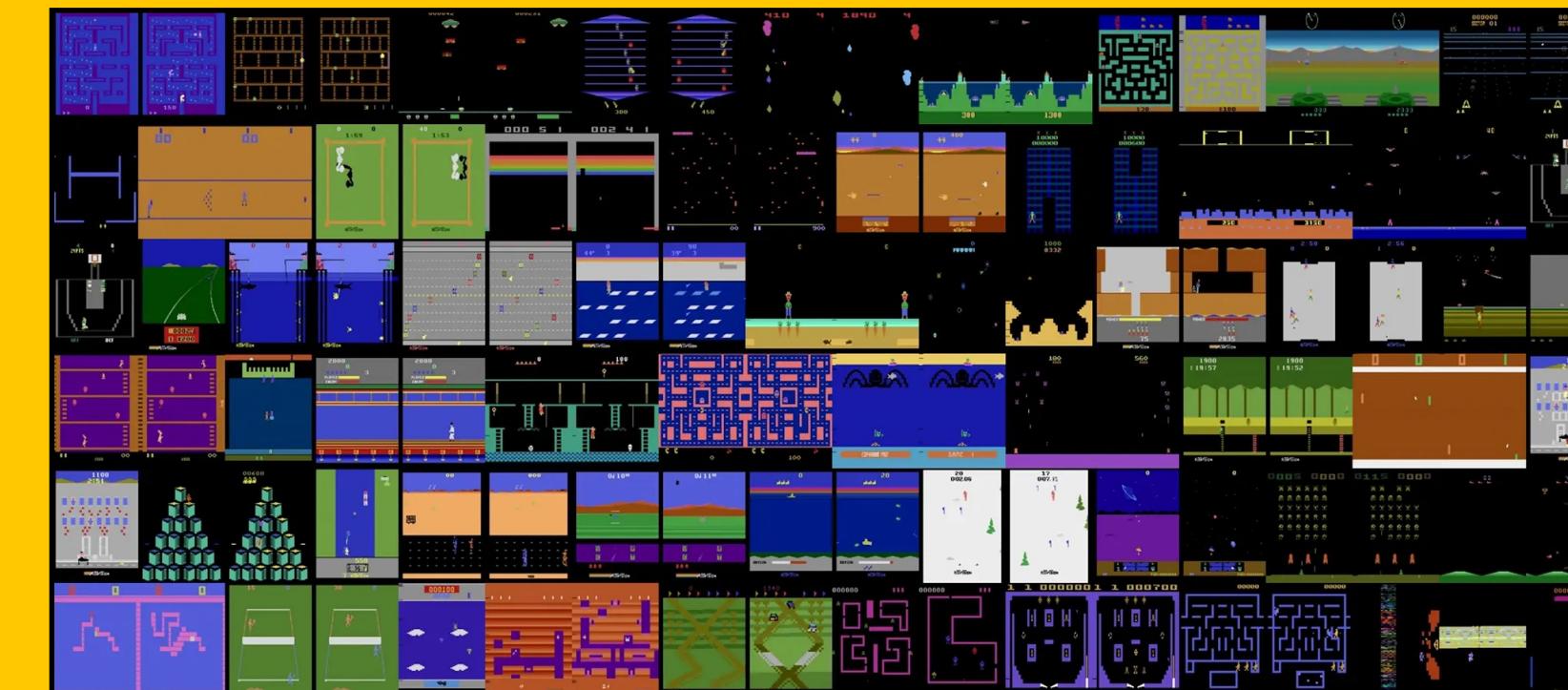
- “Spielen” (Game Playing)

Go



Quelle: Wikipedia

Alle 57 Atari Spiele



Quelle: deepmind.com

Schach



Quelle: Wikipedia

Jeopardy!



Quelle: NYTimes

AlphaGo

Agent57

DeepBlue

IBM Watson

State-of-the-Art

Beispiele

- **Maschinelles Sehen (Computer Vision)**

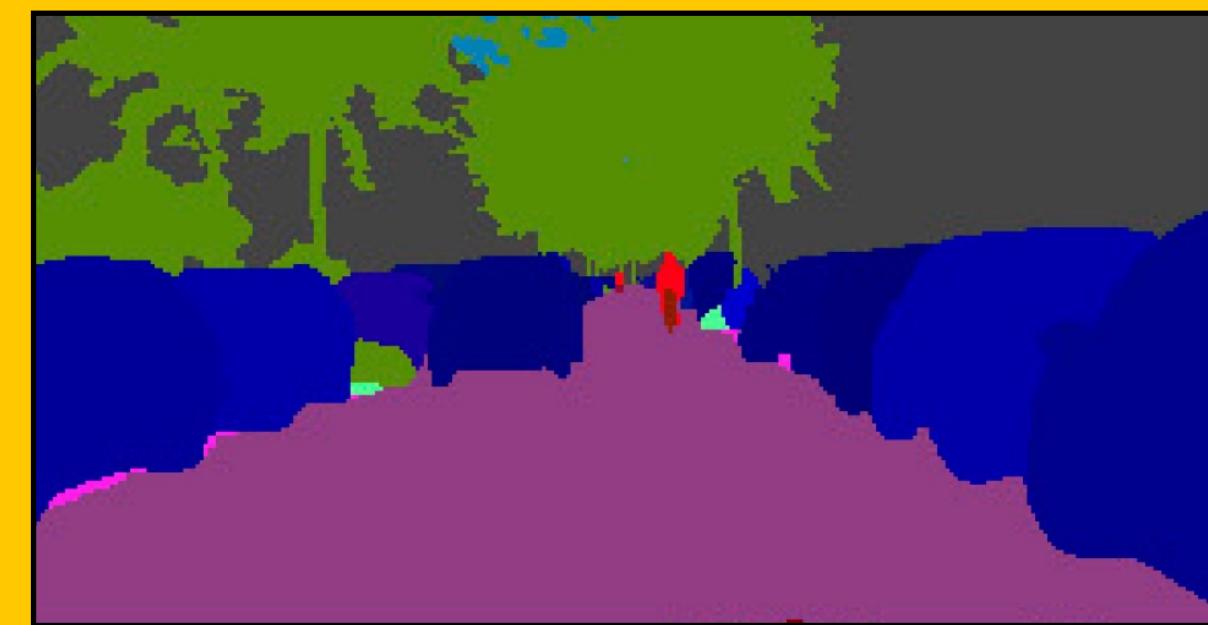
Text2Image



"A cat sweating while weightlifting in the gym"

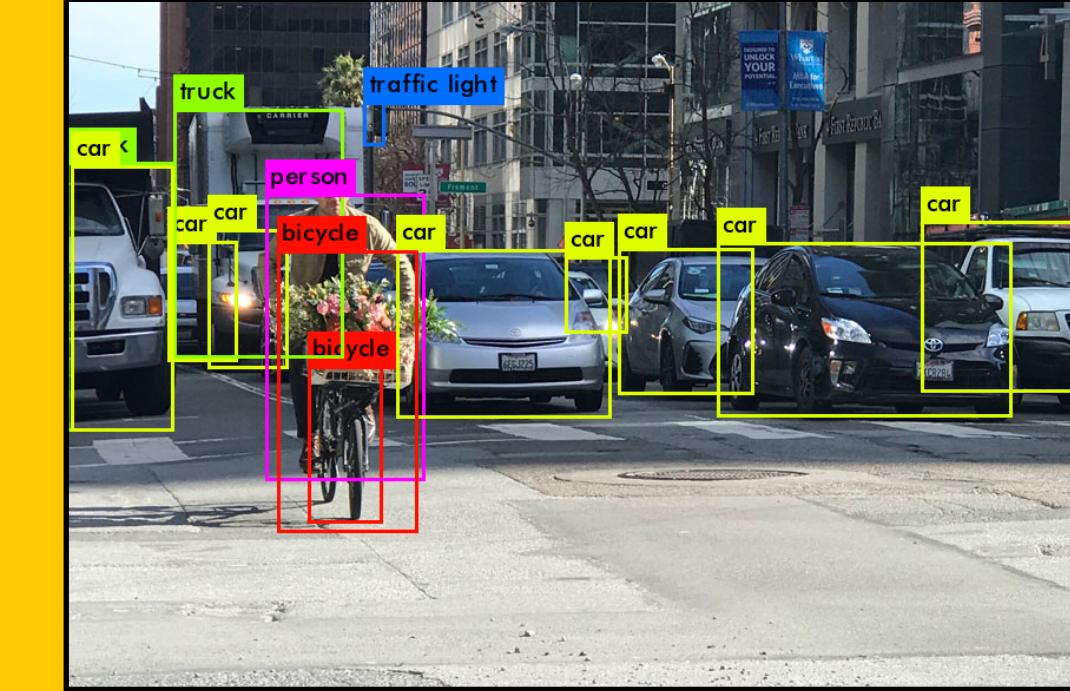
Quelle: OpenAI DALL-E

Segmentation



Quelle: Panoptic-DeepLab (Cheng et al., 2020)

Object Detection



Novel View Synthesis



Quelle: NERF

(Mildenhall et al., 2020)

State-of-the-Art

Beispiele

- **Maschinelles Sehen (Computer Vision)**

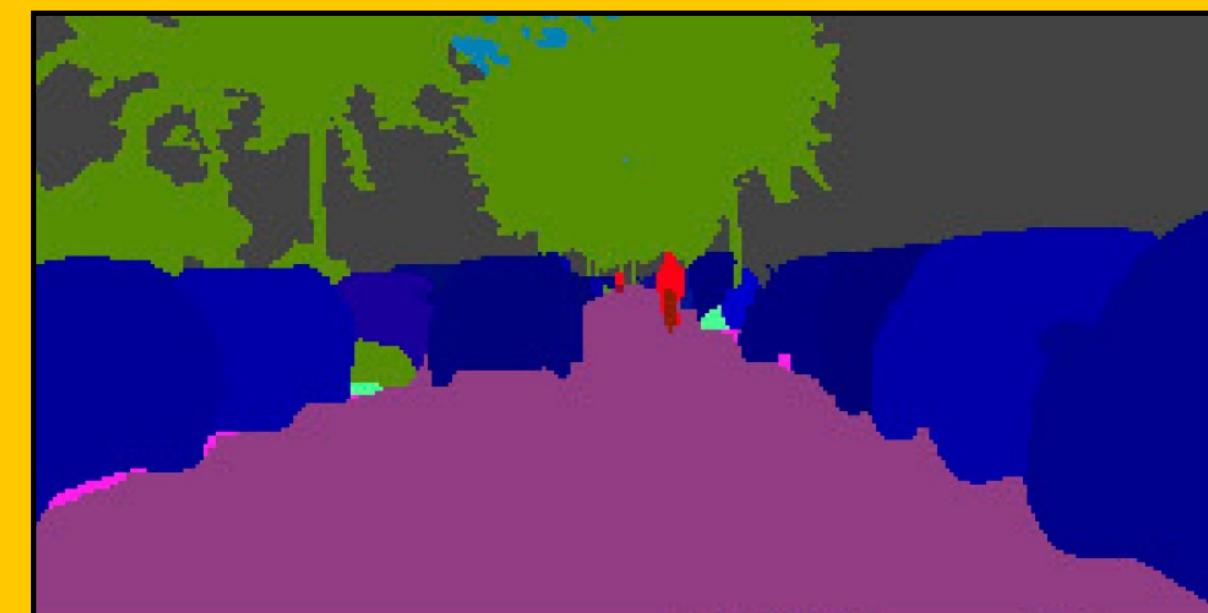
Text2Image



"A cat sweating while weightlifting in the gym"

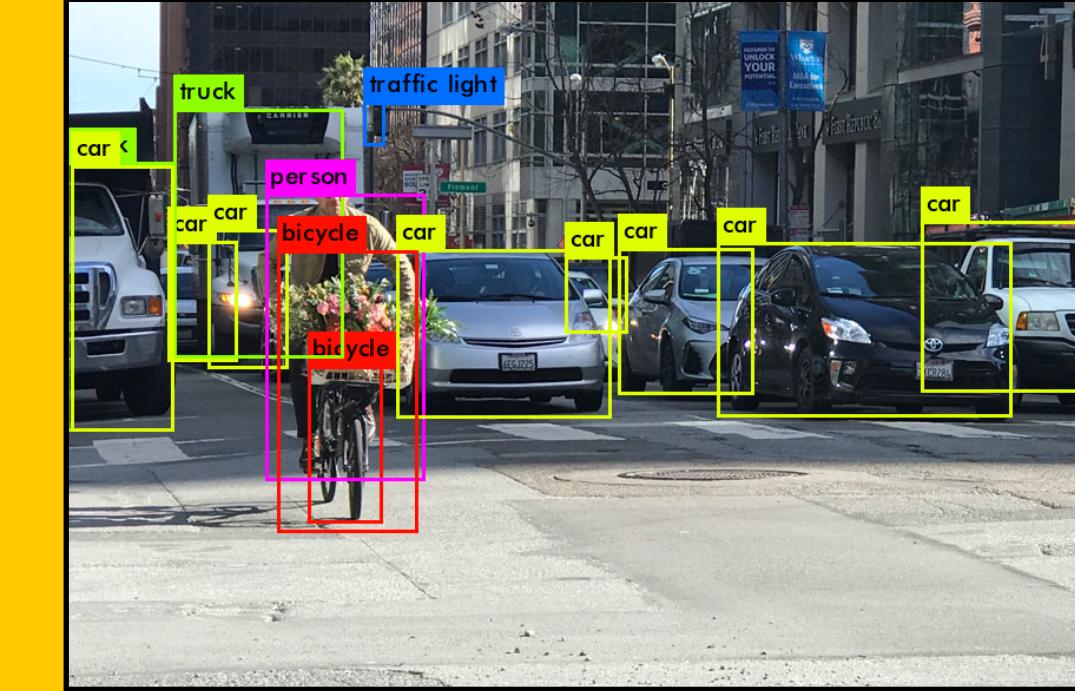
Quelle: OpenAI DALL-E

Segmentation



Quelle: Panoptic-DeepLab (Cheng et al., 2020)

Object Detection



Novel View Synthesis



Quelle: NERF

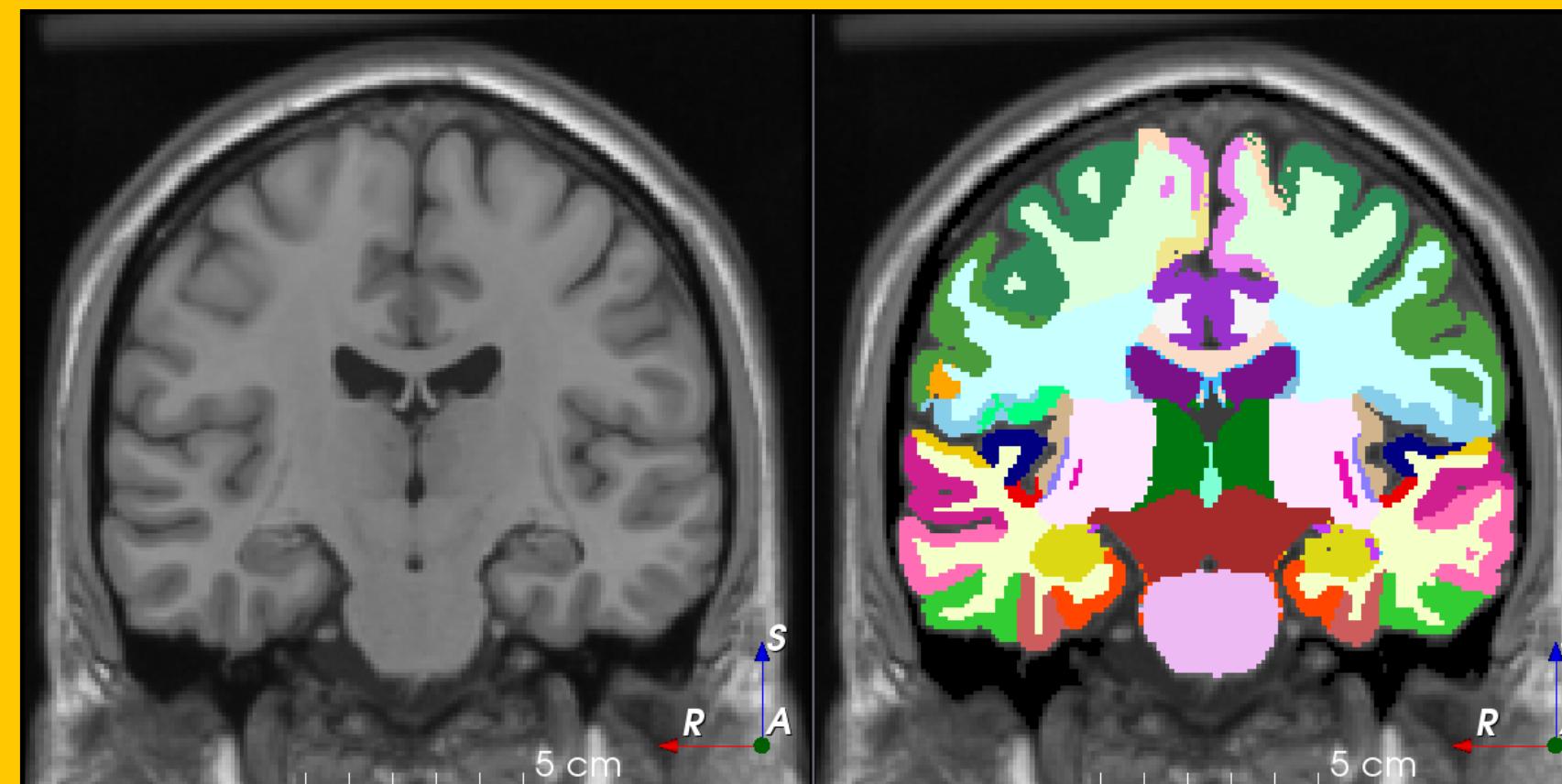
(Mildenhall et al., 2020)

State-of-the-Art

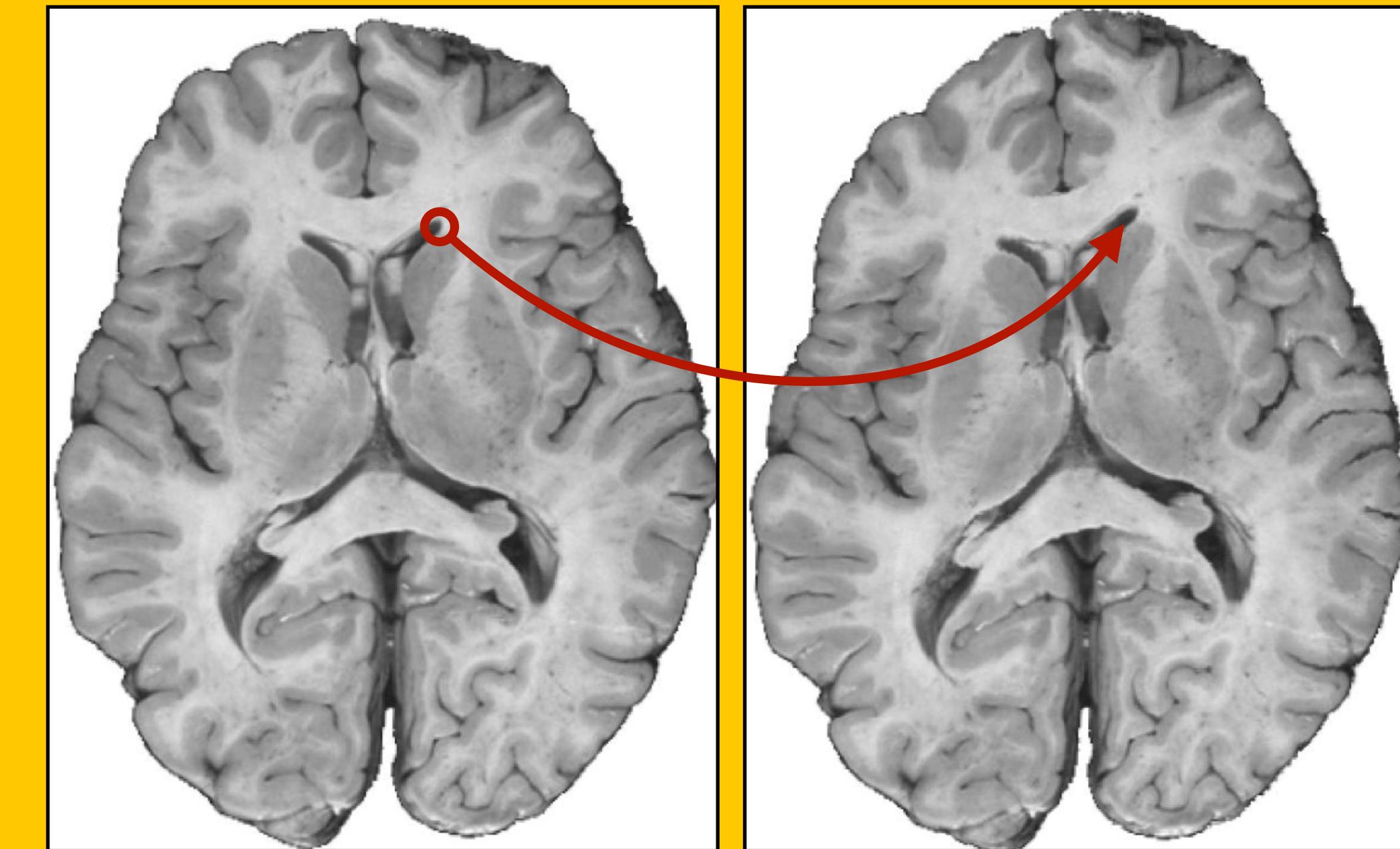
Beispiele

- **Medizinische Bildverarbeitung**

Segmentierung



Registrierung



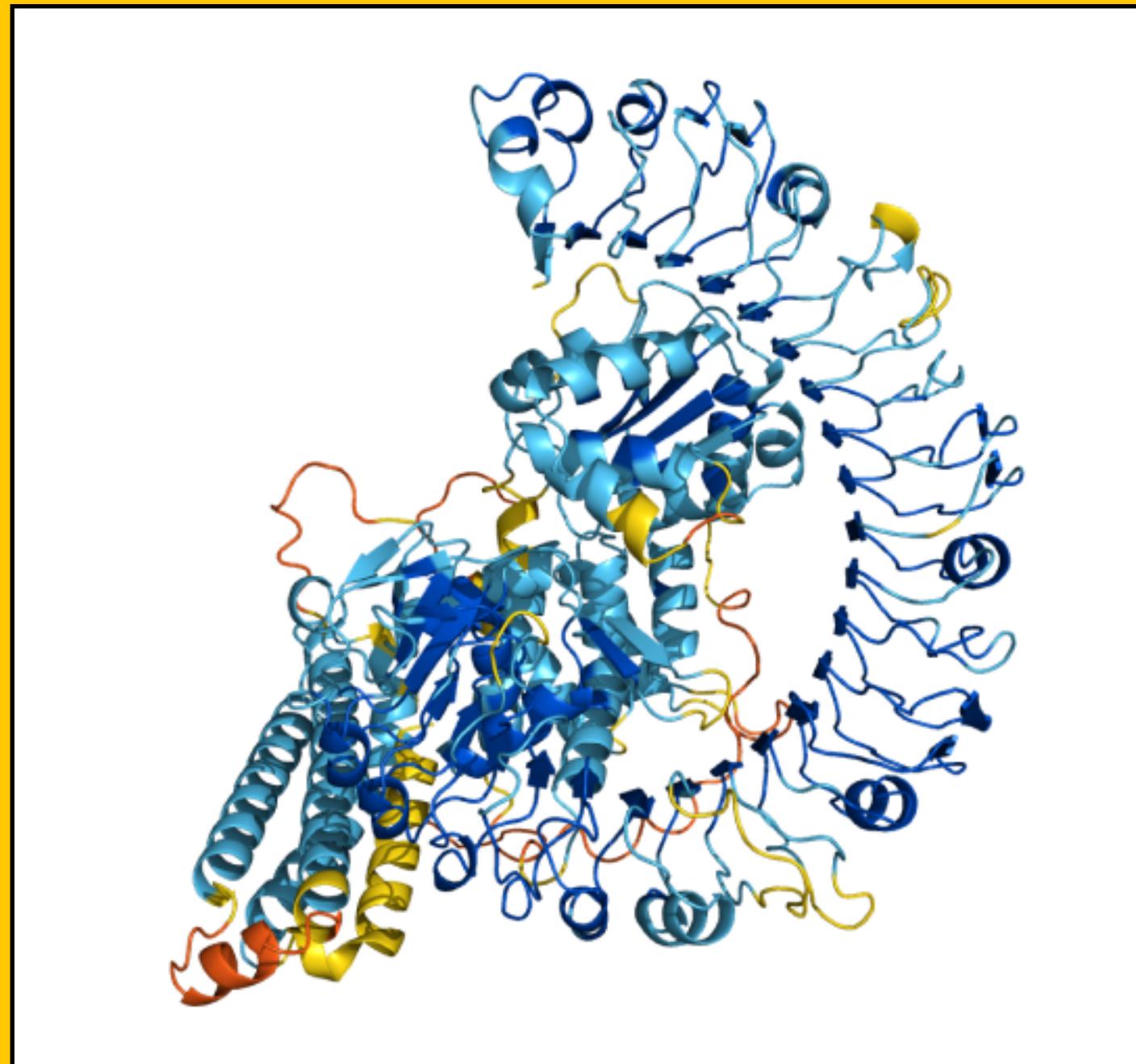
Quelle: Marc Niethammer

State-of-the-Art

Beispiele

- **Biologie**

3D Modelle von Protein Strukturen (auf Basis der Sequenz von Aminosäuren)

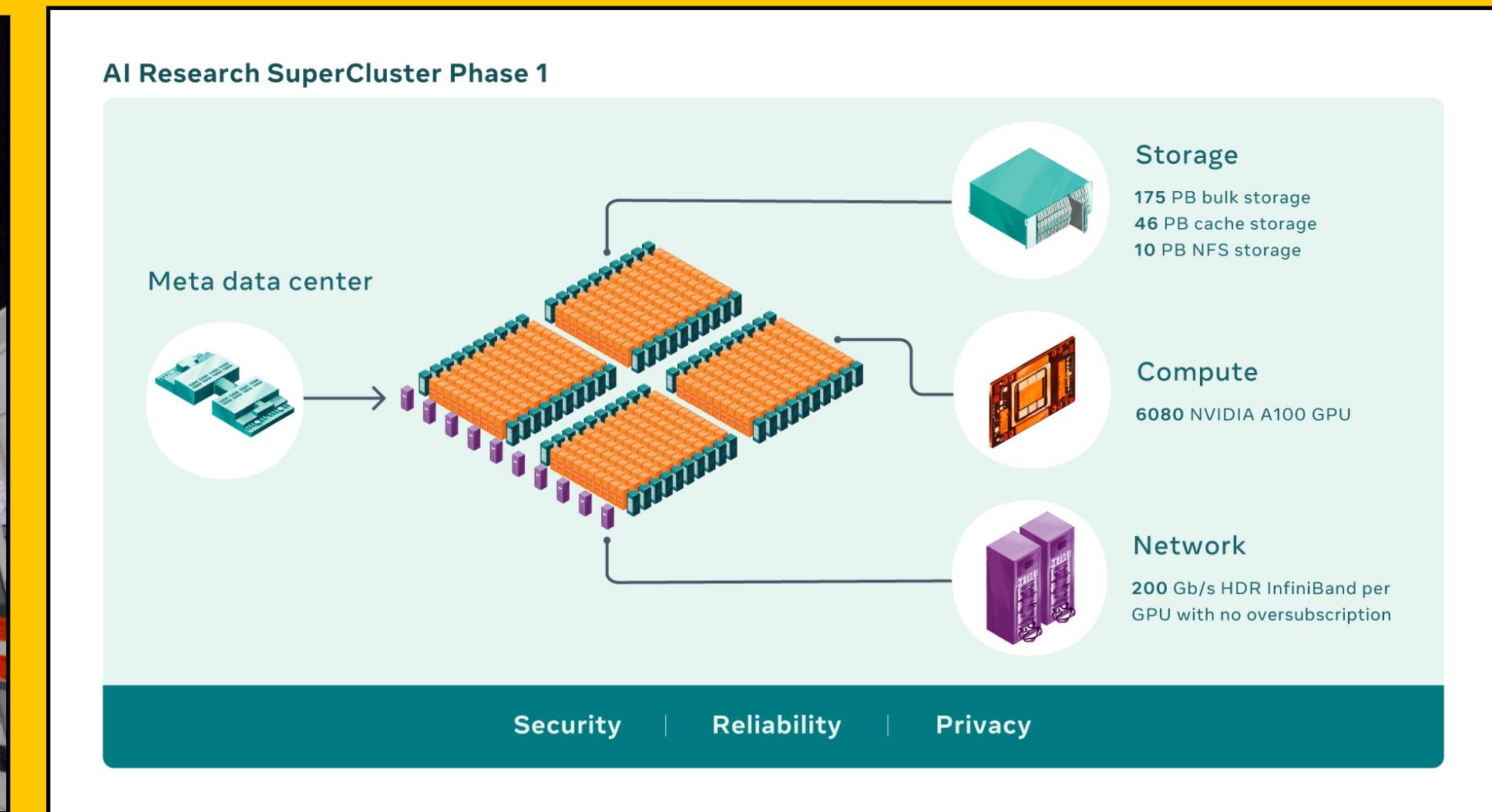


AlphaFold

State-of-the-Art

Hardware spielt eine essentielle Rolle

z.B. Meta AI **Research SuperCluster** (RSC)

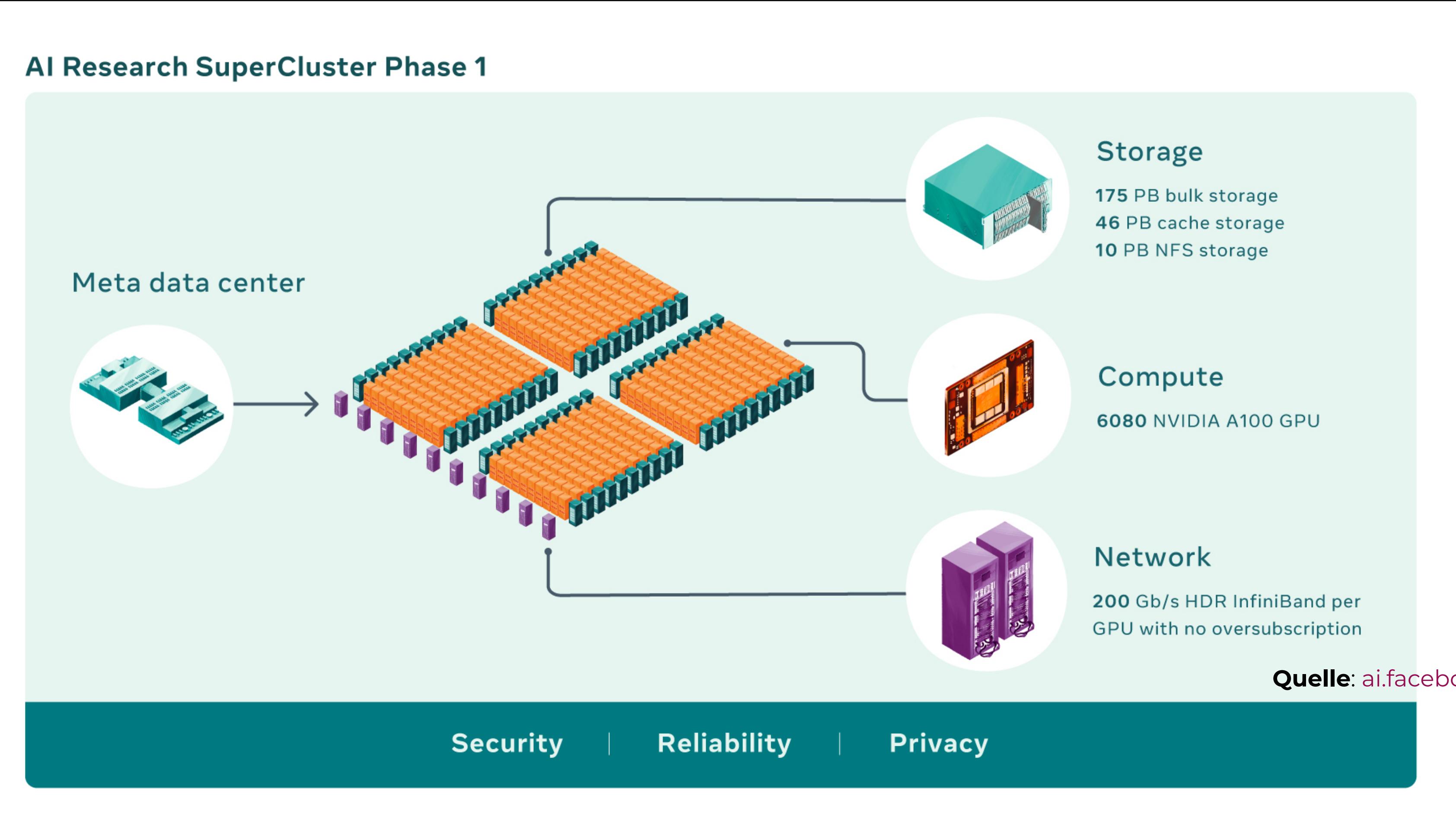


Quelle: ai.facebook.com

State-of-the-Art

Hardware speed limit in artificial intelligence

z.B. Meta AI



Intelligente Agenten

Kapitel 2 ([RN](#))

Agenten & Umgebungen

Ein **Agent** nimmt seine Umgebung (environment) über Sensoren wahr und agiert in dieser Umgebung über sogenannte **Aktuatoren**.

Beispiel: Mensch (Sensoren={Augen, Ohren, ...}; Aktuatoren={Beine, Hände, ...})

Als **Wahrnehmungsobjekt** (percept) bezeichnet man den Inhalt, den der Agent über seine Sensoren wahrnimmt. Als eine **Sequenz von Wahrnehmungsobjekten** (percept sequence) bezeichnet man die gesamte Historie die der Agent jemals wahrgenommen hat.

Agenten & Umgebungen

Das Verhalten eines Agenten wird über die **Agentenfunktion** (agent function) beschrieben, die jede Sequenz an Wahrnehmungsobjekten in eine Aktion abbildet.

Technisch gesehen, realisiert ein **Agentenprogramm** (agent program) diese Abbildung (welche man theoretisch auch tabellarisch aufschreiben könnte).

Agenten & Umgebungen

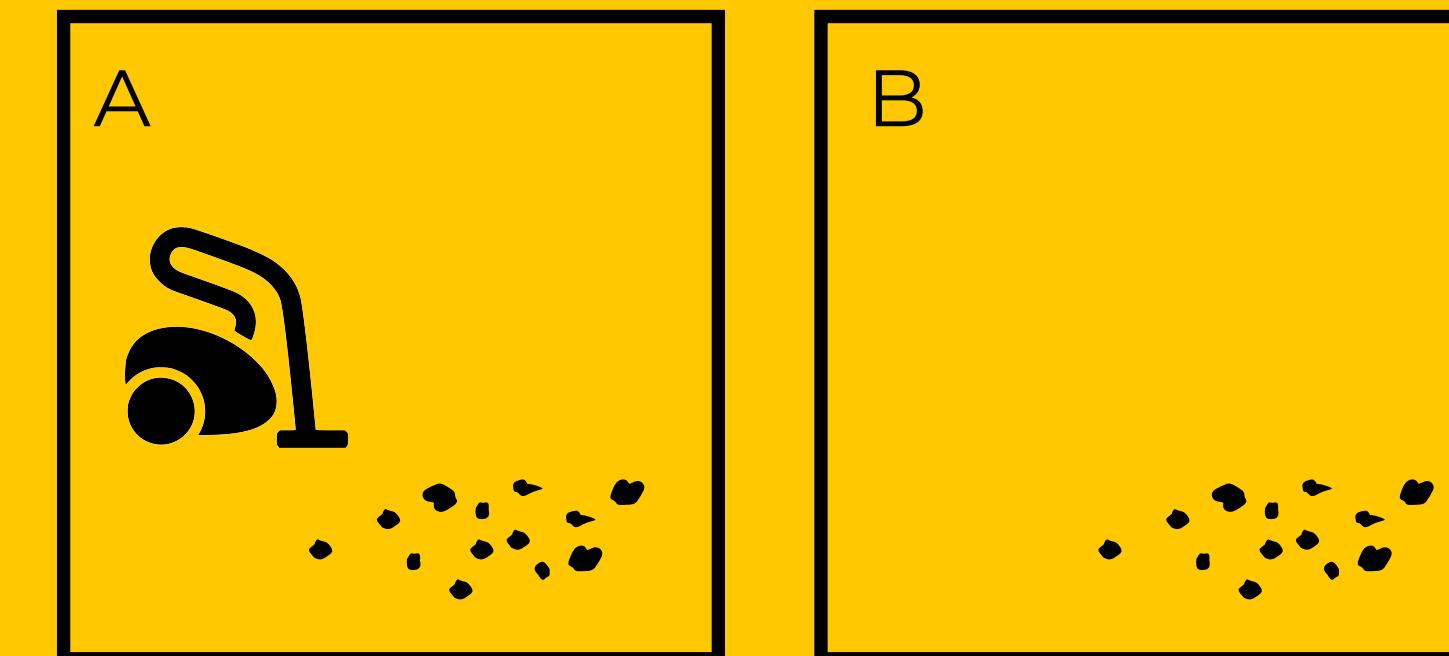
Beispiel: “Staubsauger Welt”

Die “Welt” dieses Agenten besteht aus Kacheln (im Beispiel folgend A & B), die entweder schmutzig oder sauber sind. Der Agent nimmt wahr in welchem Quadrat er ist und ob sich darin Schmutz befindet oder nicht. Der Agent reinigt die Kachel (saugt also) wenn Schmutz vorhanden ist, und bewegt sich dann zur nächsten Kachel.

Verfügbare Aktionen:

- “bewege nach rechts”
- “bewege nach links”
- “saugen”
- “Nichts tun”

Agenten & Umgebungen



Beispiel: Umgebung mit 2 Kacheln (A, B)

[A, Sauber]	bewege nach rechts
[A, Schmutzig]	saugen
[B, sauber]	bewege nach links
[B, schmutzig]	saugen
[A, sauber], [A, sauber]	bewege nach rechts
[A, sauber], [A, schmutzig]	saugen
...	...

Ausschnitt aus der Tabelle einer einfachen Agentenfunktion für die "Staubsauger Welt"

Rationalität

Ein **rationaler Agent** handelt richtig.

Um “richtig handeln” zu quantifizieren, benötigen wir ein **Performanzmaß** (performance measure). Im Kontext von AI beurteilen wir das Verhalten eines Agenten (zumeist) anhand der entstandenen Konsequenzen (consequentialism).

D.h., ein Agent generiert eine Sequenz an Aktionen als Resultat einer Sequenz von Wahrnehmungsobjekten. Dies ändert den Zustand der Umgebung. Die Frage ist nun, ob der Zustand nach einer Sequenz an Aktionen wünschenswert ist. Ein Performanzmaß quantifiziert den Zustand der Umgebung (z.B., wünschenswert oder eben nicht).

Rationalität

Als allgemeine Regel kann man formulieren, dass man sich bei dem Design des Performanzmaßes daran orientieren sollte, ob das gewünschte Ziel erreicht wird und weniger daran wie man glaubt das sich der Agent verhalten sollte.

Was tatsächlich zu einem gegebenen Zeitpunkt rational ist, hängt von folgenden vier Punkten ab:

1. dem Performanzmaß, welches uns angibt ob ein Ziel erreicht wurde
2. dem Vorwissen (prior knowledge) des Agenten über die Umgebung
3. den Aktionen die ein Agent durchführen kann
4. der Sequenz an Wahrnehmungsobjekten bis zum gegebenen Zeitpunkt

Rationalität

Definition (rationaler Agent): Für jede mögliche Sequenz an Wahrnehmungsobjekten sollte ein rationaler Agent jene Aktion wählen von der zu erwarten ist das es das gewählte Performanzmaß maximiert, gegeben der Evidenz durch die Sequenz an Wahrnehmungsobjekten und des vorliegenden Wissens über die Umgebung.

Rationalität

zurück zum “Staubsauger Welt” Beispiel!

Nehmen wir folgendes an:

- das Performanzmaß vergibt, zu jedem Zeitpunkt, **einen Punkt für eine saubere Kachel (+1)**, über eines Lebensdauer von 1000 Zeitpunkten
- wir kennen die Geometrie der Umgebung (also die Kacheln in dem Beispiel), nicht aber die Verteilung des Schmutzes oder die Startposition des Staubsaugers
- saubere Kachel bleiben sauber
- die einzigen Aktionen sind “saugen”, “bewege nach rechts / links”
- die Aktion “saugen” reinigt die aktuelle Kachel
- Der Agent nimmt seine Position wahr, sowie ob Schmutz vorhanden ist od. nicht

Rationalität

Der Staubsauger Agent handelt (unter diesen Annahmen) also **rational**.

Sind alle Kachel sauber, würde der Agent jedoch zwischen den Kacheln hin und her oszillieren. Unter einem anderen Performanzmaß (beispielsweise bei Vergabe von “Minus” Punkten bei jeder *Bewegung*), würde er jedoch recht schlecht abschneiden.

Rationalität ist jedoch nicht gleich Perfektion. Ein rationaler Agent maximiert den erwarteten Erfolg, ein perfekter Agent den tatsächliche Erfolg.

Aufgabenumgebungen (Task environments)

Bevor wir rationale Agenten entwerfen können, müssen wir uns also über die **Aufgabenumgebung** (task environment) Gedanken machen. In anderen Worten, die Problemstellung spezifizieren welche ein Agent lösen sollte.

Im vorherigen Beispiel hatten wir das gemacht, indem wir das Performanzmaß, die Umgebung, die Aktionen sowie die Sensoren spezifiziert hatten, typischerweise abgekürzt als die **PEAS** Beschreibung: **P**(erformance), **E**(nvironment), **A**(ctions), **S**(ensors).

Im Folgenden, betrachten wir verschiedene Eigenschaften von Aufgabenumgebungen, die es uns erlauben, Aufgabenumgebungen zu kategorisieren.

Aufgabenumgebungen — Eigenschaften

Ist der vollständige Zustand der Umgebung zu jedem Zeitpunkt über die Sensoren zugänglich, bezeichnen wir dies als **voll beobachtbar** (fully observable). In anderen Worten, wir kennen alle Aspekte die relevant sind um entsprechende Aktionen auszuwählen (natürlich vom Performanzmaß abhängig).

Ist die Sensorik mit Rauschen behaftet, ungenau, od. erfasst schlichtweg nicht den vollständigen Zustand der Umgebung, bezeichnen wir dies als **teilweise beobachtbar** (partially observable).

Aufgabenumgebungen — Eigenschaften

Wir unterscheiden weiter zwischen **Einzel-** und **Multiagenten** Umgebungen (single vs. multiagent environments). Sudoku zu Spielen ist ein Beispiel für eine Einzelagenten Umgebung, wohingegen Schach eine Multiagenten (konkret 2) Umgebung wäre.

Die Unterscheidung kommt oft auf das konkrete Problem an, konkret auf die Eigenschaft ob beispielsweise das Verhalten eines Agenten B durch Maximierung eines Performanzmaßes abhängig vom Verhalten des Agenten A beschrieben werden kann.

Prinzipiell würde man hier dann auch noch zwischen **kooperativen** (cooperative) und **kompetitiven** (competitive) Multiagentenumgebungen unterscheiden. Als Beispiel für Letzteres wäre Schach zu nennen.

Aufgabenumgebungen — Eigenschaften

Ist der nächste Zustand einer Umgebung und die nächste Aktion eines Agenten vollständig durch den aktuellen Zustand charakterisiert, bezeichnen wir dies als **deterministisch**, ansonsten als **nicht-deterministisch**.

Das “Staubsauger Welt” Beispiel ist deterministisch. Erlauben wir jedoch, dass die “saugen” Aktion nicht zuverlässig funktioniert, oder beispielsweise Schmutz zufällig auftritt, würde es sich um eine nicht-deterministische Aufgabenumgebung handeln.

Aufgabenumgebungen — Eigenschaften

In **episodischen** Aufgabenumgebungen (episodic task environment), erfährt ein Agent ein Wahrnehmungsobjekt und führt eine Handlung aus (dies ist eine Episode). Die Handlung in der nächsten Episode ist nicht von der Handlung der vorhergehenden Episode abhängig.

In **sequentiellen** Aufgabenumgebungen (sequential task environments) könnte die aktuelle Handlung die Folgehandlungen (auch alle) beeinflussen.

Aufgabenumgebungen — Eigenschaften

Kann sich die Umgebung ändern während der Agent “überlegt” welche Entscheidung er fällt (also wie er handelt), so nennen wir dies eine **dynamische** Aufgabenumgebung (dynamic task environment). In dynamischen Umgebungen wird der Agent fortlaufend gefragt, welche Handlung er setzen möchte.

Ändert sich die Umgebung nicht, nennen wir dies eine **statische** Aufgabenumgebung (static task environment).

Zusätzlich könnte es auch sein, dass sich das Performanzmaß des Agenten über die Zeit hinweg ändert, jedoch nicht die Umgebung. Dies nennt man **semi-dynamisch** (ein Beispiel hierfür wäre Schach mit Schachuhr).

Aufgabenumgebungen — Eigenschaften

Zusätzlich können wir verschiedene Bestandteile von Aufgabenumgebungen, wie Handlungen, die Wahrnehmungen, oder auch den Zustand der Umgebung dahingehend unterscheiden, als dass es sich um **diskrete** oder **stetige** Bestandteile handeln kann.

Betrachtet man beispielsweise “Taxi fahren”, sind sowohl Geschwindigkeit als auch Ort sich stetig über die Zeit hinweg ändernde Objekte. Ebenso können Handlungen eines “Taxi Agenten” (z.B., Steuerungswinkel) stetig sein.

Aufgabenumgebungen — Eigenschaften

Bezeichnet man eine Aufgabenumgebung als **bekannt** (known) oder **unbekannt** (unknown), bezieht man sich dabei auf das Wissen des Agenten-Designers hinsichtlich der in der Umgebung geltenden Gesetze (z.B., die Gesetze der Physik).

Ist die Aufgabenumgebung **bekannt**, sind die Ergebnisse aller Handlungen bekannt (od. eben die entsprechenden Wahrscheinlichkeiten, sofern es sich um eine nicht-deterministische Aufgabenumgebung handelt). Im Fall einer **unbekannten** Aufgabenumgebung muss der Agent erst die Ergebnisse seiner Handlungen erlernen.

Beispiel (Solitaire): man kennt die Regeln, sieht aber nur jene Karten die man bereits aufgedeckt hat (ergo, bekannte aber nur teilweise beobachtbare Aufgabenumgebung).

Aufgabenumgebungen — Eigenschaften

“Raffineriesteuerung” als weiters PEAS Beispiel

Performanzmaß: Sicherheit, Reinheit, Ertrag

Umgebung: Raffinerie, Rohstoffe, Betreiber

Aktuatoren: Pumpen, Displays, Ventile, Rührer, etc.

Sensoren: Temperatur-, Druck-, Durchfluss- und chemische Sensoren

Eigenschaften: teilweise beobachtbar, Einzel-Agent, stochastisch, sequenziell, dynamisch, stetig (bekannt vs. unbekannt: schwer zu sagen :)

Struktur von Agenten

Unser Ziel ist es ein **Agentenprogramm** (agent program) zu entwerfen, welches die Agentenfunktion implementiert, also die Abbildung von Wahrnehmungsobjekten zu Handlungen.

Dieses Agentenprogramm läuft auf einer “Maschine” mit physischen Sensoren und Aktuatoren. Wir nennen dies die **Agentenarchitektur** (agent architecture) und die Kombination von Architektur und dem Agentenprogramm nennen bezeichnen wir als den Agenten.

Agent = Agentenarchitektur + Agentenprogramm

Struktur von Agenten

Das **Bauprinzip** eines Agentenprogramms wird in weiterer Folge immer das Gleiche sein: empfange das **aktuelle** Wahrnehmungsobjekt von den Sensoren und bestimme dann eine Aktion, welche über die Aktuatoren ausgeführt wird.

Hinweis: die Agentenfunktion (siehe frühere Folien) hingegen bildet jede Sequenz von Wahrnehmungsobjekten auf eine Aktion/Handlung ab.

Anmerkung: Benötigt der Agent vorherige Wahrnehmungsobjekte um eine Handlung zu selektieren, müssen eben die vorherigen Wahrnehmungsobjekte gespeichert werden.

Struktur von Agenten

Skizze eines **Tabellen-basierten** Agentenprogramms (Pseudocode):

```
function TABLE-DRIVEN-AGENT(percept) returns an action
    persistent: percepts, a sequence, initially empty
                table, a table of actions, indexed by percept sequences, initially fully specified

    append percept to the end of percepts
    action <- LOOKUP(percepts, table)
    return action
```

In diesem Fall hier werden alle Wahrnehmungsobjekte (über den Lebenszeitraum des Agenten) gespeichert und die Handlung immer auf Basis aller bisherigen Wahrnehmungsobjekte entschieden (anhand der Funktion LOOKUP).

Struktur von Agenten

Hätte man nun eine Menge, \mathcal{P} , an möglichen Wahrnehmungsobjekten und einen Zeitraum T (also #Zeitpunkte an denen Wahrnehmungsobjekte empfangen werden) ergäbe sich eine Tabelle mit

$$\sum_{t=1}^T |\mathcal{P}|^t$$

Einträgen (der Form: Sequenz von Wahrnehmungsobjekten -> Handlung). Selbst bei sehr einfachen Problemstellungen würde diese Vorgehensweise extrem impraktikabel sein (und wir könnten die Tabelle schlichtweg nicht mehr speichern).

Struktur von Agenten

Man kann also behaupten, dass es unser Ziel sein muss, Programme zu realisieren die rationales Verhalten produzieren, jedoch ohne massive Tabellen benutzen zu müssen.

Im Folgenden werden wir vier Arten von Agentenprogrammen besprechen, die dieses Prinzip in verschiedenen Ausprägungen verkörpern:

1. **Einfache Reflex Agenten** (simple reflex agents)
2. **Modell-basierte Reflex Agenten** (model-based agents)
3. **Ziel-basierte Agenten** (goal-based agents)
4. **“Utility”-basierte Agenten** (utility-based agents)

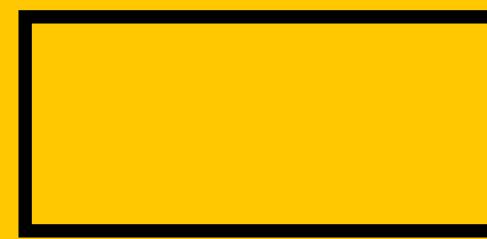
Struktur von Agenten

Einfache Reflex Agenten (simple reflex agents)

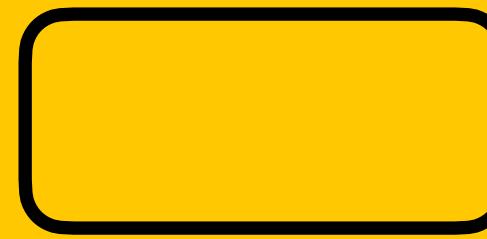
Selektieren Handlungen auf Basis des **aktuellen** Wahrnehmungsobjektes.

Der “Staubsauger” Agent aus unserem Beispiel wäre ein Repräsentant eines solchen Agenten. Wir hatten 4 mögliche Wahrnehmungsobjekte (<{Schmutzig, Sauber} x {Kachel A, Kachel B}) pro Zeitschritt. Da alles vorherigen Zeitschritte ignoriert werden, also eine Gesamtanzahl an 4 relevanten Wahrnehmungsobjekten.

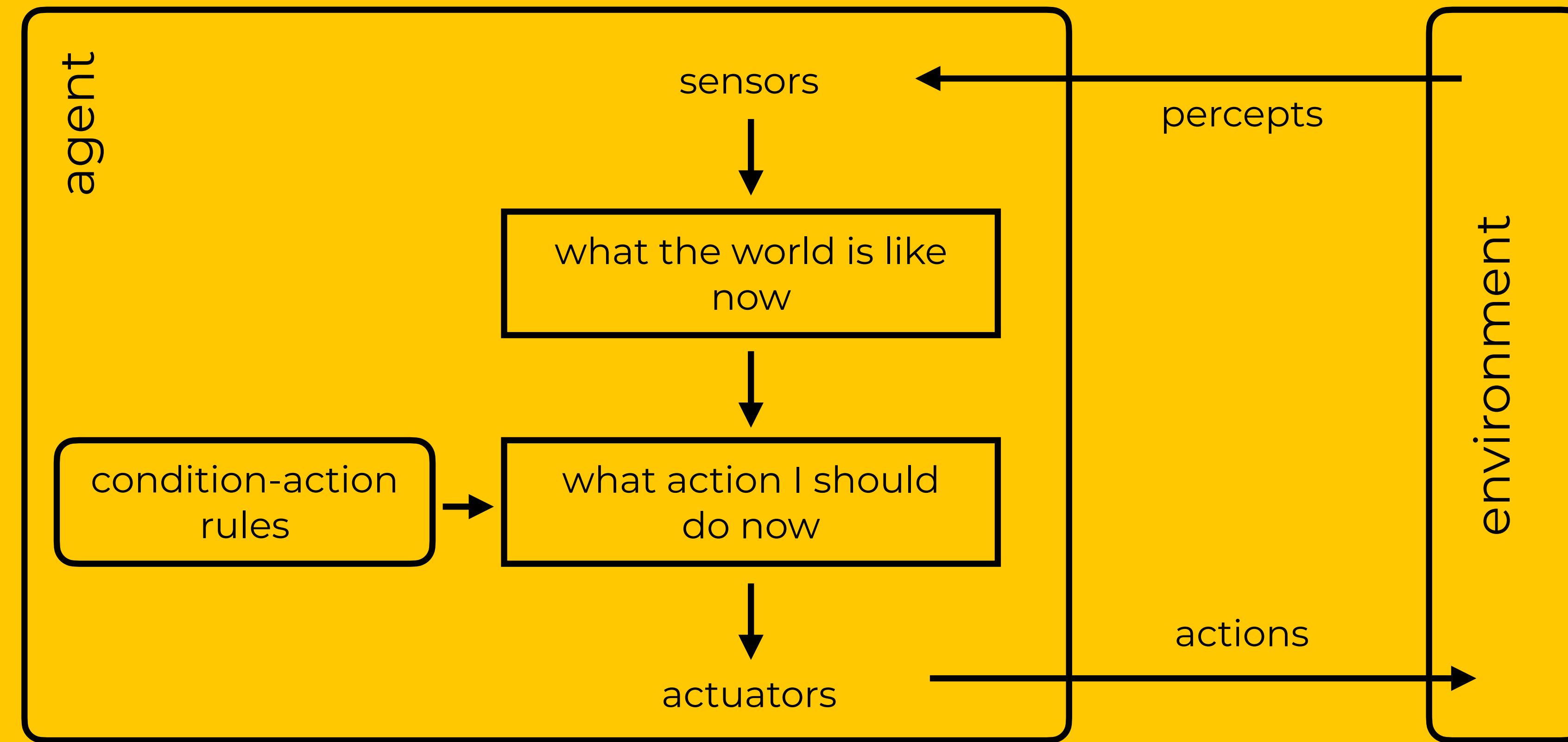
Struktur von Agenten



Interner Zustand des Agenten während des Entscheidungsprozesses (current internal state)



Hintergrundwissen (background information) während des Entscheidungsprozesses



Struktur von Agenten

```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  persistent: rules, a set of condition-action rules

  state <- INTERPRET-INPUT(percept)
  rule <- RULE-MATCH(state, rules)
  action <- rule.ACTION
  return action
```

Struktur von Agenten

Einfaches Reflex-Verhalten findet man auch in deutlich komplexeren Szenarien, z.B., dem Bremsen beim Lenken eines Fahrzeugs:

```
if car-in-front-is-braking then initiate-braking
```

Einfache Reflex-Agenten sind natürlich limitiert, z.B., funktioniert er/sie nur dann, wenn die korrekte Entscheidung auf Basis des aktuellen Wahrnehmungsobjektes tatsächlich getroffen werden kann (also in voll-beobachtbaren Aufgabenumgebungen).

Wir könnten uns beispielsweise fragen, ob wir `car-in-front-is-braking` überhaupt (mit Sicherheit) auf Basis eines einzelnen Video-Frames feststellen können.

Struktur von Agenten

In unserem “Staubsauger” Beispiel würde eine problematische Situation unmittelbar eintreten sobald der Ort-Sensor nicht mehr funktioniert.

In dem konkreten Fall, würde der Agent nur mehr [Schmutzig] / [Sauber] wahrnehmen (anstatt z.B. [A, Sauber]). Dies könnte zu einer Endlosschleife führen wenn sich der Agent in Kachel A befindet (aber dies nicht bekannt ist) und [Sauber] wahrnimmt. Da wir uns in einer deterministischen Aufgabenumgebung befinden, kann es durchaus sein, dass die Aktion “nach-links bewegen” gewählt wird, diese aber jedoch für immer fehlschlägt.

Ein Ausweg (aus der Endlosschleife) wäre, die Aktion zu **randomisieren** (wie viele Schritte würde der Agent in Erwartung benötigen um die andere Kachel erreichen?)

Struktur von Agenten

Modell-basierte Reflex Agenten (model-based reflex agents)

Um mit teilweiser Beobachtbarkeit (partial observability) umgehen zu können, sollte der Agent einen **internen Zustand** (internal state) aufrecht erhalten, welcher von der bisherigen Sequenz an Wahrnehmungsobjekten abhängt.

Um den Zustand aktualisieren zu können, benötigen wir

- (1) ein sogenanntes **Übergangsmodell** (transition model) der “Welt”, sowie
- (2) ein **Modell der Sensorik** (sensor model).

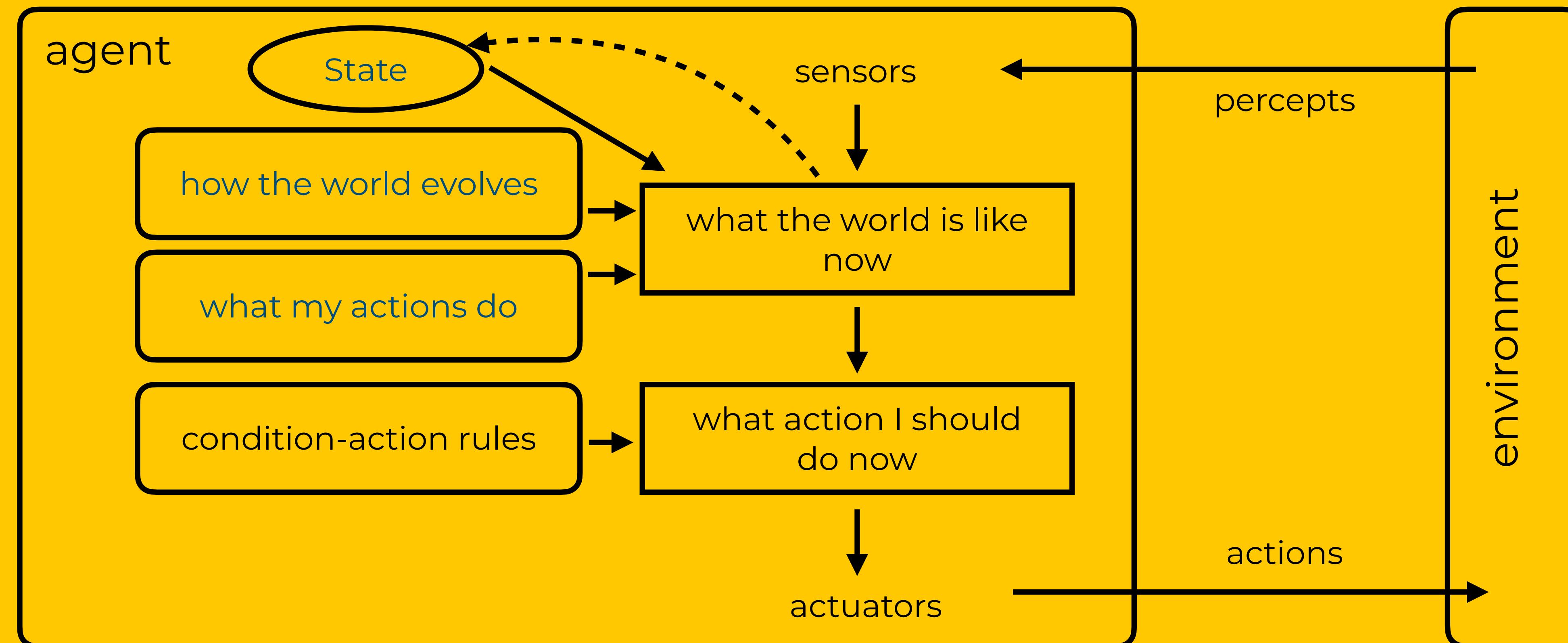
Struktur von Agenten

Das **Übergangsmodell** beschreibt wie sich die “Welt” über die Zeit hinweg verändert. Dies umfasst sowohl die Veränderungen bzgl. der Aktionen des Agenten, sowie jene Veränderungen unabhängig von diesen Aktionen.

Das **Modell der Sensorik** umfasst, wie sich der Zustand der “Welt” in den Wahrnehmungsobjekten widerspiegelt (bremst beispielsweise ein Auto vor unserem Agenten und hat dieser Kameras, würde sich dieses Bremsen als rote Regionen in den empfangenen Kamerabildern widerspiegeln).

Agenten die solche Übergangsmodelle und Modelle der Sensorik nutzen nennt man **Modell-basierte Reflex Agenten**.

Struktur von Agenten



Struktur von Agenten

```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
    persistent: state, the agent's current conception of the world state
              transition model, a description of how the next state depends on the current state and action
              sensor model, a description of how the current world state is reflected in the agent's percepts
              rules, a set of condition-action rules
              action, the most recent action, initially none

    state <- UPDATE-STATE(state, action, percept, transition model, sensor model)
    rule <- RULE-MATCH(state, rules)
    action <- rule.ACTION
    return action
```

Struktur von Agenten

Allgemein ist zu sagen, dass es dem Agenten selten möglich ist den Zustand in teilweise beobachtbaren (partially observable) Aufgabenumgebungen exakt zu bestimmen. Meist handelt es sich dabei (egal wie auch immer technisch realisiert) um einen “best guess”.

Struktur von Agenten

Ziel-basierte Agenten (goal-based agents)

Etwas über den aktuellen Zustand der Umgebung zu wissen um eine Entscheidung zu treffen reicht häufig nicht aus. In vielen Aufgabenstellungen benötigt der Agent, zusätzlich zum aktuellen Zustand der Umgebung Informationen über das **Ziel** (goal) seiner Handlungen.

Beispiel: steht ein Agent einer Kreuzung um mit den Handlungsoptionen “nach links”, “nach rechts”, “geradeaus” zu gehen/fahren, wäre es natürlich hilfreich die finale Destination (z.B. ein konkreter Ort) zu kennen (zusätzlich zum aktuellen Zustand der Umgebung).

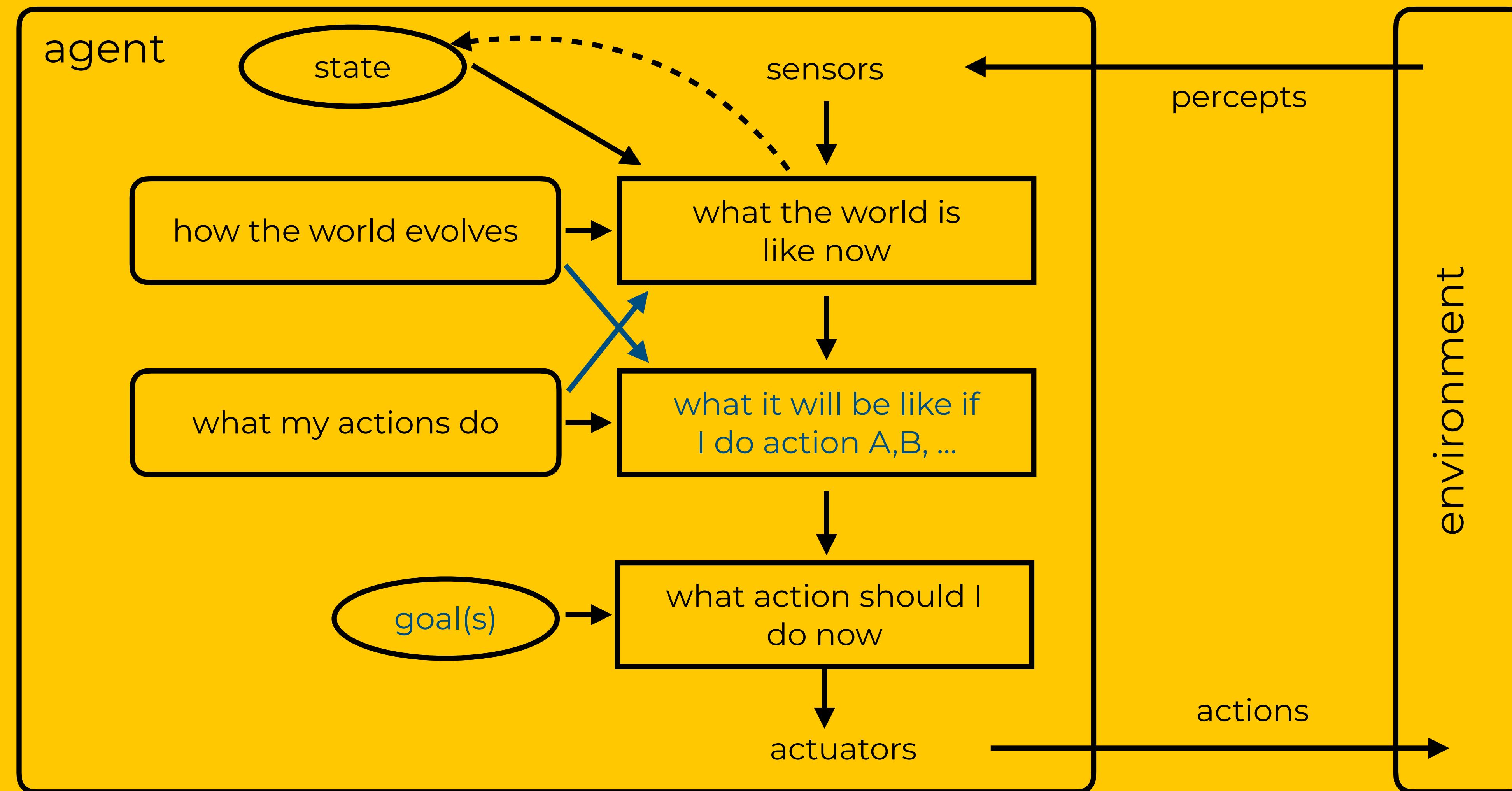
Struktur von Agenten

Klarerweise ist ein Ziel oft nicht unmittelbar in einer einzigen Handlung zu erreichen, sondern wir müssen Sequenzen von Handlungen betrachten um unser Ziel zu erreichen. Dies fällt thematisch in die AI Gebiete **Suche** (search) und **Planung** (planning).

Hierin besteht auch schon der fundamentale Unterschied zu den “condition-action” Regeln einfacher & Modell-basierter Reflex Agenten. Um in einer Umgebung so zu handeln, dass ein Ziel erreicht wird, müssen im Allgemeinen Überlegungen über die Zukunft angestellt werden (d.h. von der Art “was passiert wenn ich so und so handle?”).

Ziel-basierte Agenten sind deutlich **flexibler** als Reflex-Agenten, da Ziele ersetzt / geändert werden können (diese Flexibilität ist im Falle der bereits bekannten “condition-action” Regeln nicht gegeben).

Struktur von Agenten



Struktur von Agenten

Utilitäts-basierte Agenten (utility-based agents)

Führen mehrere Wege (also konkrete Handlungssequenzen) zum Ziel, ist es im Allgemeinen nicht ausreichend/sinnvoll “nur” das Ziel zu definieren.

Ziele sind in gewisser Art und Weise binär (entweder erreicht oder eben nicht). Ein allgemeineres Maß würde es uns erlauben Zustände der Umgebung zu **vergleichen** und zu **bewerten** wie “zufrieden” der Agent mit einem dieser Zustände ist. Wissenschaftlicher ausgedrückt, fassen wir dies mit dem Begriff der **Utilität** (Nützlichkeit, utility).

Zusätzlich zum (bereits bekannten) Performanzmaß, haben wir nun eine (dem Agenten interne) **Nützlichkeits-Funktion** (utility function) die Nützlichkeit bewertet.

Struktur von Agenten

Stimmen Performanzmaß und diese Nützlichkeitsfunktion überein und wählt der Agent seine Handlungen so, daß die Nützlichkeitsfunktion maximiert wird, handelt der Agent rational (dies hängt ja, wie bekannt, vom Performanzmaß ab).

Sich an Nützlichkeit zu orientieren ist besonders in zwei Fällen sinnvoll: (1) wenn **Ziele widersprüchlich** sind (z.B. Vereinbarkeit von Sicherheit und Geschwindigkeit im Kontext autonom fahrender Systeme), bzw. (2) der Agent mehrere Ziele verfolgen könnte jedoch **keines der Ziele mit Sicherheit erreicht werden kann.**

Nützlichkeit kann dazu dienen Ziele gegeneinander abzuwägen (im Fall 1), bzw. die Erfolgswahrscheinlichkeit zur Erreichung eines Ziels gegen dessen Wichtigkeit zu gewichten (im Fall 2).

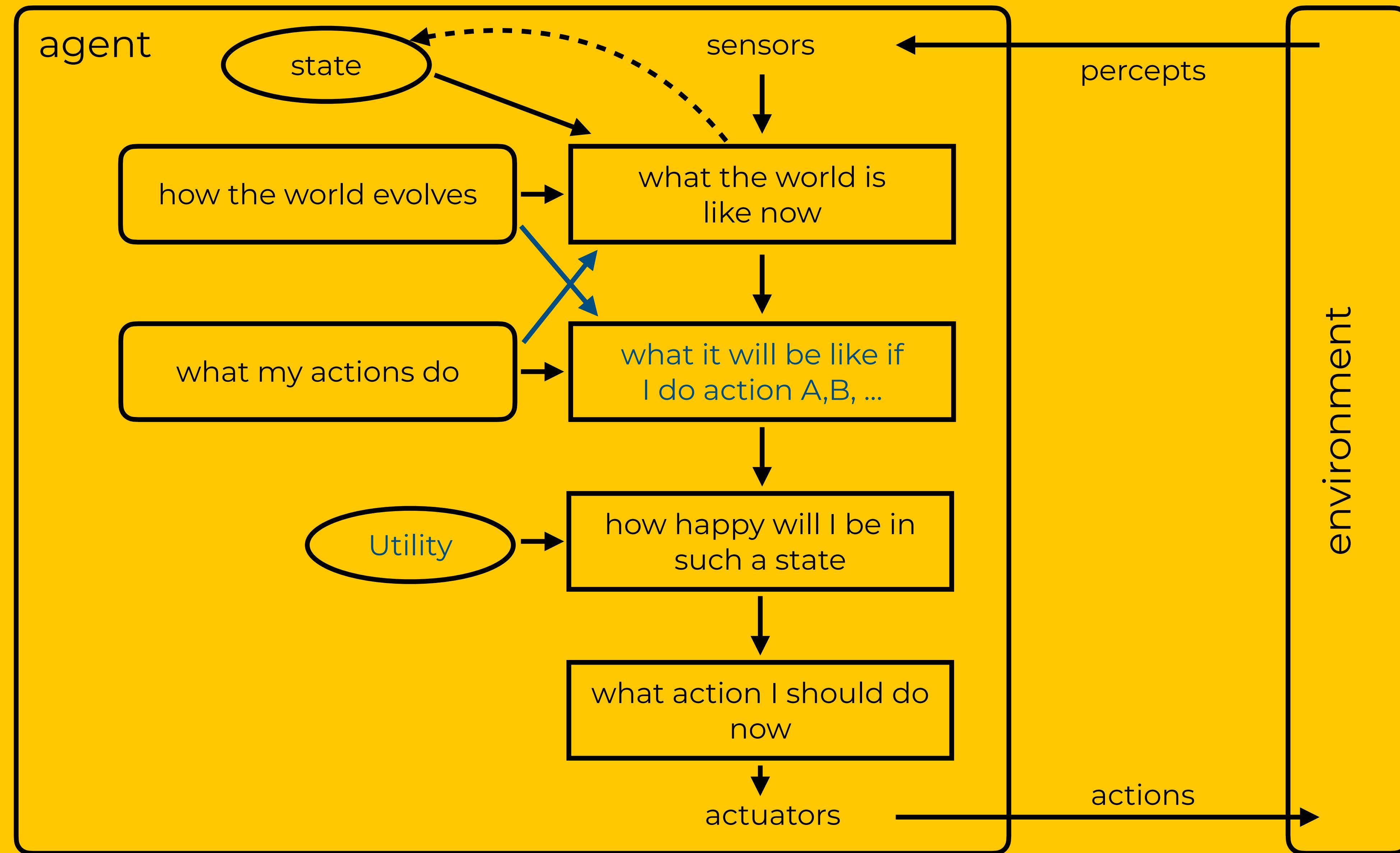
Struktur von Agenten

Speziell im Kontext teilweise-beobachtbarer, nicht-deterministischer (od. stochastischer) Aufgabenumgebungen (was wohl auf die meisten realen Aufgabenumgebungen zutreffen wird), müssen wir Entscheidungen unter **Unsicherheit** treffen.

Ein rationaler Utilitäts-basierter Agent wählt eine Handlung so, dass die **erwartete Nützlichkeit** (expected utility) des Handlungsresultats maximiert wird (vorausgesetzt wir haben eine Wahrscheinlichkeit und Nützlichkeit zu jedem Handlungsresultat verfügbar).

Nicht alle Utilitäts-basierte Agenten basieren auf einem Modell (wie in der Illustration auf der nächsten Folie suggeriert). Es gibt auch **Modell-freie Agenten** (model-free agents) die erlernen welche Handlung in einer bestimmten Situation gewählt werden soll, ohne tatsächlich zu erlernen wie genau diese Handlung die Umgebung ändert.

Struktur von Agenten



Struktur von Agenten

Lernagenten (learning agents)

Wie entsteht eigentlich das Agentenprogramm?

Anstatt intelligente Maschinen tatsächlich zu “programmieren”, findet sich in Turing’s Arbeit ([Turing, 1950](#)) folgender Satz: *“At my present rate of working I produce about a thousand digits of programme a day, so that about sixty workers, working steadily through the fifty years might accomplish the job, if nothing went into the waste-paper basket. Some more expeditious method seems desirable.”*

Struktur von Agenten

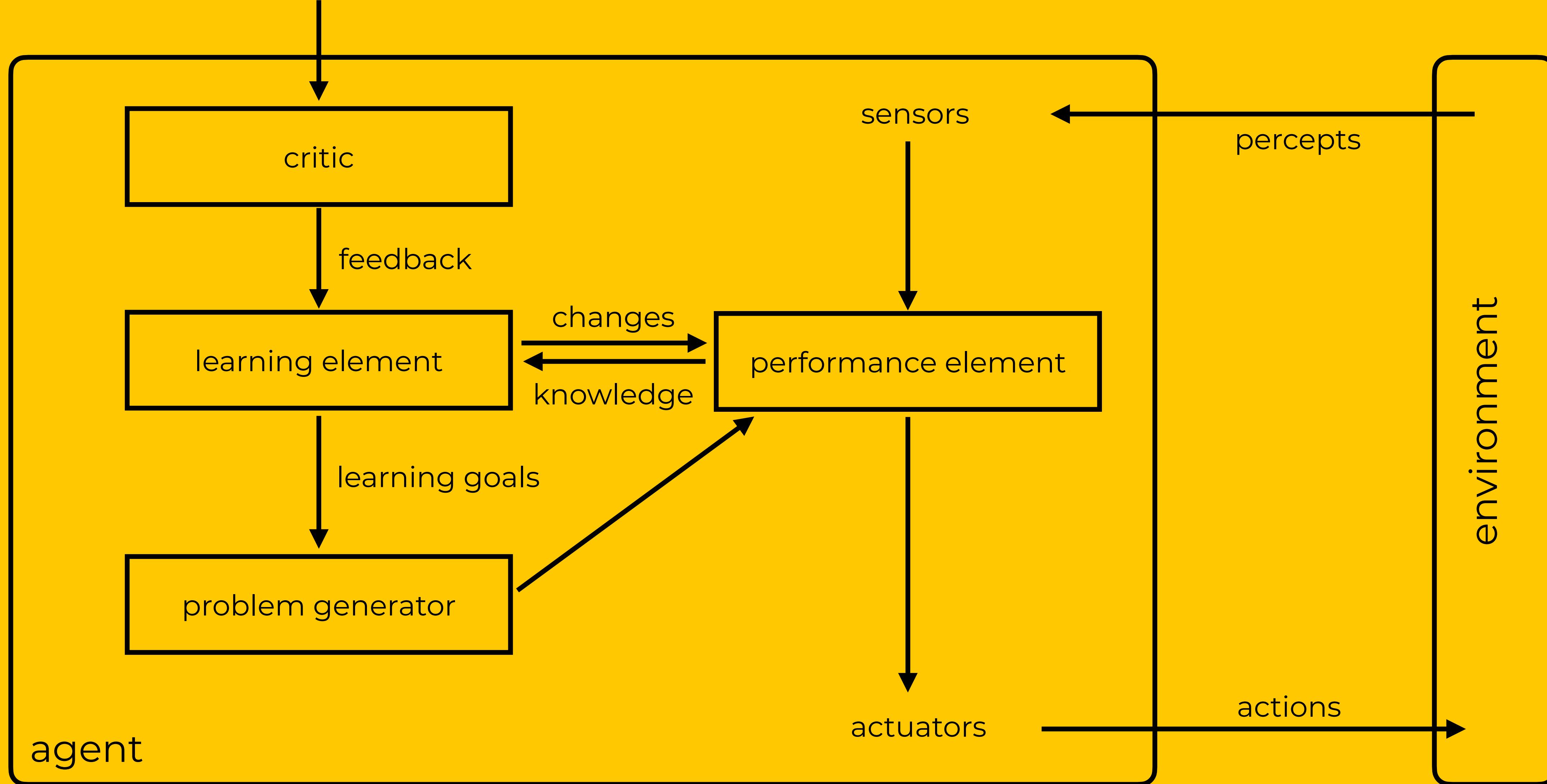
Mehr oder weniger handelt es sich hier (also in der Arbeit von Turing) um die Geburt der Idee zum **maschinellen Lernen** (machine learning).

Alle zuvor besprochenen Agenten (also einfache & Modell-basierte Reflex Agenten, Ziel-basierte Agenten und Utilitäts-basierte Agenten) lassen sich als Lernagenten realisieren. In nahezu jedem Gebiet von AI sind Lernagenten aktuell der de-facto Standard um State-of-the-Art Systeme umzusetzen.

Lernen hat zudem den Vorteil, dass Agenten in initial-unbekannten Umgebungen operieren können und über die Zeit immer kompetenter werden. Die Komponenten eines Lernagenten sind auf der folgenden Folie dargestellt.

Struktur von Agenten

(External) performance standard



Struktur von Agenten

Das **Lernelement** (learning element) ist verantwortlich Verbesserungen zu machen, das **Performanzelement** (performance element) wählt schlussendlich Handlungen aus. Im Wesentlichen subsummiert das Performanzelement alle Komponenten in unseren bisherigen Agenten.

Das Lernelement nutzt Feedback vom **Bewerter** (critic) über die Performanz des Agenten um das Performanzelement zu modifizieren (alle Elemente in den Diagrammen zu den vorherigen Agenten können modifiziert werden).

Struktur von Agenten

Der Bewerter nutzt den externen **Performanzstandard** (performance standard; nicht modifizierbar) um dieses Feedback zu geben. Die Sensorik kann hierzu ja nicht dienen (z.B. die Stellung Schachmatt im Schach soll zwar über Sensorik erkannt, aber nicht bewertet werden).

Sozusagen handelt es sich beim Performanzstandard um ein Element welches Teile einer Sequenz an Wahrnehmungsobjekten als Strafe (penalty) od. als Belohnung (reward) bewertet.

Der **Problemgenerator** (problem generator) schlägt Handlungen vor um neue informative Erfahrungen zu machen. Diese sind natürlich oftmals suboptimal, erlauben es jedoch, auf längere Sicht gesehen, bessere Handlungen zu finden.

Repräsentation der Umgebung

Bislang haben wir verschiedene Komponenten betrachtet, welche im Wesentlichen folgende Fragen (aus Sicht des Agenten) beantworten:

- Wie ist der aktuelle Zustand meiner Umgebung?
- Welche Handlung soll ich durchführen?
- Welche Konsequenzen haben meine Handlungen?

Ein wichtiger Punkt in diesem Kontext ist die Art und Weise wie unsere verschiedenen Komponenten die Umgebung des Agenten **repräsentieren**.

Repräsentation der Umgebung

Wir unterscheiden grob zwischen

- **atomaren**,
- **faktorisierten**, und
- **strukturierten**

Repräsentationen (gelistet in der Reihenfolge aufsteigender Komplexität).

Repräsentation der Umgebung

In einer **atomaren Repräsentation** ist der Zustand der Umgebung nicht aufteilbar, der Zustand hat sozusagen keine interne Struktur.

Beispiel: Wir werden später ein Routensuchproblem sehen, wo der Zustand der Umgebung schlichtweg durch den Namen einer Stadt repräsentiert ist. Dieser atomare Zustand unterscheidet sich von anderen atomaren Zuständen, ist aber quasi eine “Black-Box”.

Repräsentation der Umgebung

In einer **faktorisierten Repräsentation** ist der Zustand der Umgebung in eine fixe Menge an Variablen und Attributen aufgeteilt. Diese Variablen und Attribute können Werte annehmen.

Beispiel: im zuvor genannten Routensuchproblem könnte man den Zustand der Umgebung aufteilen in “Treibstoffstand”, “GPS Koordinaten”, “Ölwarnleuchte an/aus”, etc.

Repräsentation der Umgebung

In einer **strukturierten Repräsentation** ist der Zustand der Umgebung charakterisiert durch Dinge/Objekte und deren Relation zwischen einander; also nicht mehr alleine durch Variablen und Attribute mit Werten.

Vieles was wir in unserer natürlichen Sprache ausdrücken handelt von Objekten und deren Beziehungen.

Repräsentation der Umgebung

Von atomar hin zu faktorisiert und strukturiert steigt die **Ausdrucksstärke** (expressiveness) der Repräsentation.

In vielen Fällen kann man mit einer ausdrucksstärkeren Repräsentation Dinge deutlich knapper ausdrücken. Beispielsweise sind die Regeln von Schach auf ein paar Seiten in einer strukturierten Repräsentationssprache (z.B. Logik erster Stufe) beschreibbar, man benötigt aber $\approx 10^{40}$ Seiten würde man das Gleiche in einer atomaren Repräsentationssprache (z.B. in der eines endlichen Automaten) versuchen.

Problemlösen mittels Suche

Kapitel 3 ([RN](#))

Übersicht

Ist die richtige Handlung nicht unmittelbar ersichtlich, muss ein Agent eventuell **vorausplanen**, also Sequenzen von Handlungen betrachten, um sein Ziel zu erreichen.

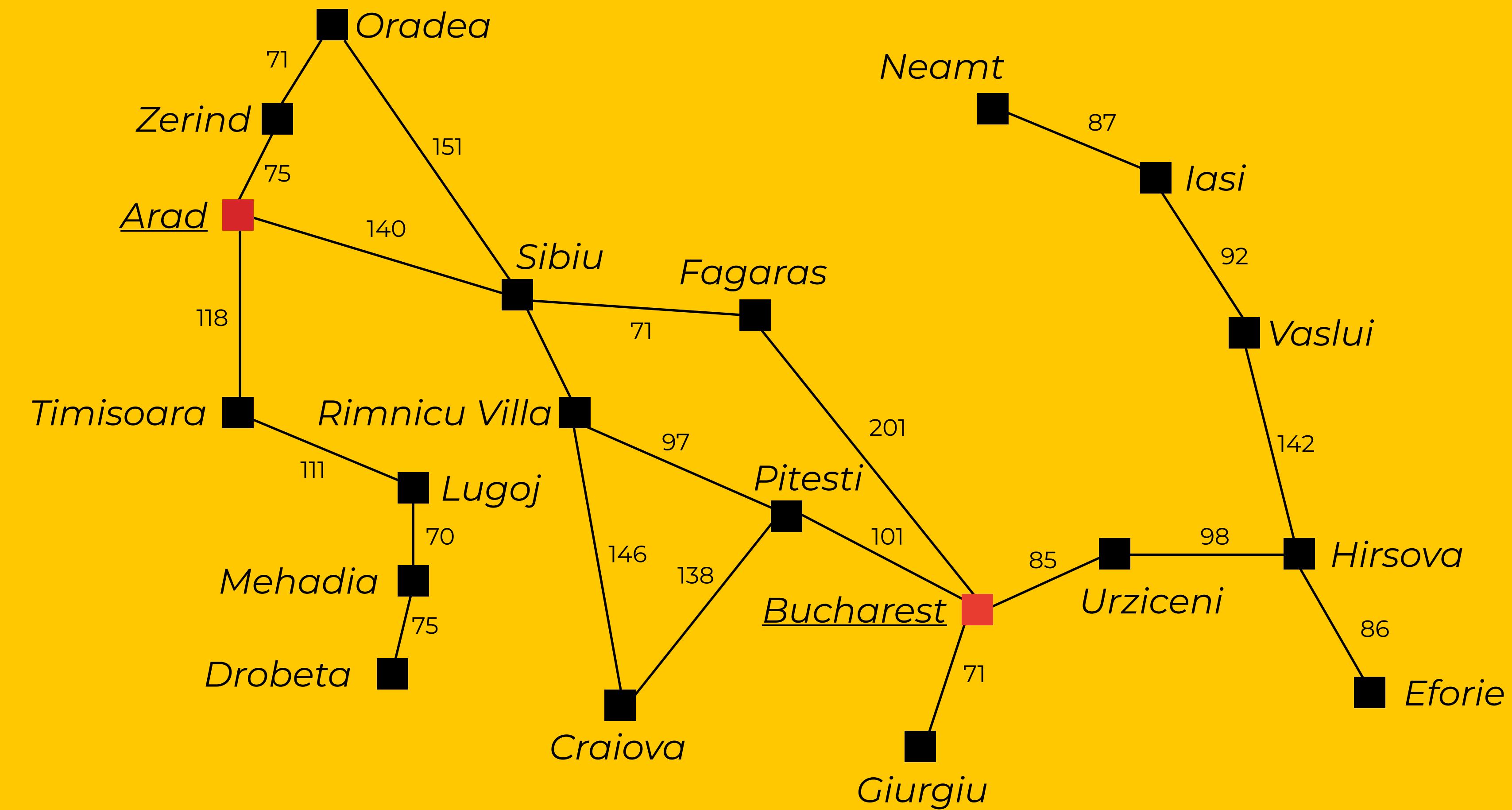
Solche Agenten nennt man **Problemlöseagenten** (problem-solving agents); diese nutzen **atomare** Repräsentationen, d.h. keine interne Struktur ist den Problemlösealgorithmen zugänglich.

Im Gegensatz dazu, nennt man Agenten die **faktorisierte** oder **strukturierte** Repräsentationen nutzen **Planungsagenten** (planning agents).

Im Folgenden betrachten wir Problemlöseagenten.

Problemlöseagenten

Beispiel: Tour durch Rumänien



Problemlöseagenten

Ist die Aufgabenumgebung **unbekannt**, kann der Agent, ausgehend von Arad, eigentlich nur zufällig eine der zwei Wege weg von Arad (nach Sibiu, oder Timisoara) zufällig wählen.

Wir nehmen jedoch an, die Aufgabenumgebung (also hier die Karte) ist bekannt. In diesem Fall, kann der Agent den folgenden 4 Stufen des Problemlöseprozesses folgen:

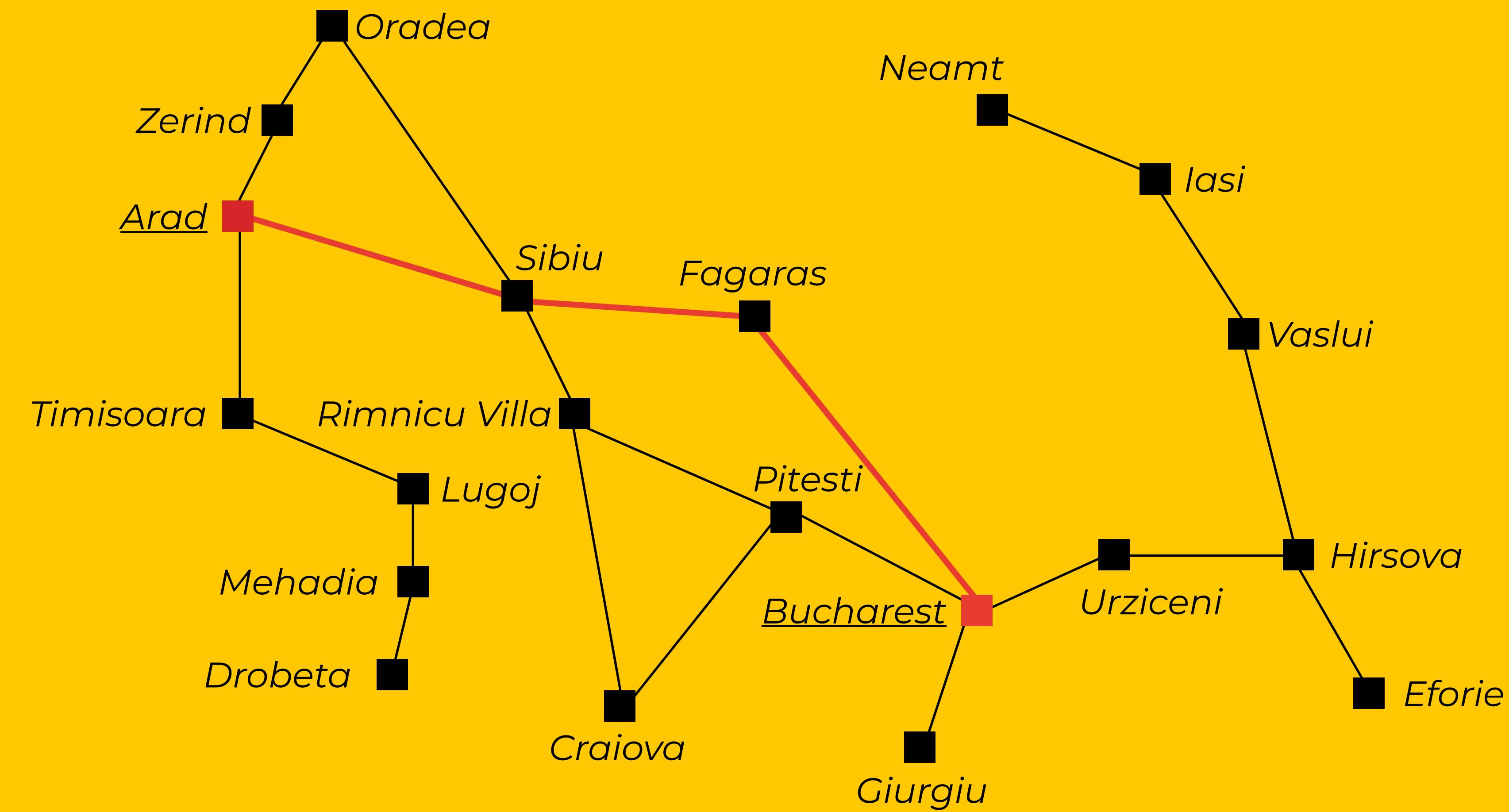
1. **Formulierung des Ziels** (goal formulation): Ziel ist Bucharest. Dies schränkt jedenfalls schon einmal die Handlungen die betrachtet werden ein.
2. **Formulierung des Problems** (problem formulation): der Agent erstellt eine abstraktes Modell der relevanten Teile seiner Umgebung (Zustände und Handlungen die notwendig sind um das Ziel zu erreichen).

Problemlöseagenten

Anm. zu 2) ein gutes Modell wäre nur Handlungen zu betrachten die den Agenten von einer Stadt in eine Nachbarstadt bringen. Der einzige Teil des Zustands seiner Umgebung der sich nach einer Handlung ändert ist also die aktuelle Stadt.

3. **Suche** (search): Vor einer Handlung, simuliert der Agent eine Sequenz von Handlungen in seiner Umgebung, mit dem Ziel eine Sequenz zu finden in der er sein Ziel erreicht. Eine solche Sequenz nennt man eine **Lösung** (solution).

Problemlöseagenten



Problemlöseagenten

4. **Durchführung** (execution): Durchführen der Handlungen in der gefundenen Lösung, Schritt für Schritt.

In voll-beobachtbaren, deterministischen und bekannten Aufgabenumgebungen ist die Lösung zu jedem Problem eine fixe Sequenz von Handlungen.

Die Wahrnehmungsobjekte könnten, bei gefundener Lösung, ignoriert werden. Man nennt dies auch ein **open-loop** System (Begriff aus der Kontrolltheorie), da es keine “Schleife” zwischen Agenten und Umgebung mehr gibt (siehe frühere Diagramme).

Problemlöseagenten

Ist die Aufgabenumgebung nicht deterministisch, od. nur teilweise beobachtbar, ist es keine gute Idee die Wahrnehmungsobjekte zu ignorieren. Bei teilweiser Beobachtbarkeit müsste eine Lösung ja Handlungsvorschläge auf Basis aktueller Wahrnehmungsobjekte beinhalten.

Suchprobleme und Lösungen

Ein **Suchproblem** kann folgendermaßen formal beschrieben werden:

- Eine **Menge an Umgebungszuständen** (state space)
- Ein **Initialzustand** (initial state)
- Eine **Menge an Zielzuständen** (goal states), z.B. $S = \{\text{Bucharest}\}$
- Eine **Menge an Handlungen** die dem Agenten zur Verfügung stehen. Bei gegebenen Zustand s , liefert $\text{ACTION}(s)$ liefert die Menge an Handlungen die in Zustand s durchgeführt werden können, z.B. $\text{ACTION}(\text{Arad}) = \{\text{ToSibiu}, \text{ToTimisoara}, \text{ToZerind}\}$
- Ein **Übergangsmodell** (transition model), welches beschreibt was in einem Zustand bei gegebener Handlung passiert, $\text{RESULT}(\text{Arad}, \text{ToZerind}) = \text{Zerind}$

Suchprobleme und Lösungen

- Eine **Handlungs-Kostenfunktion** (action cost function): $\text{ACTION-COST}(s, a, s')$ gibt uns die “Kosten” die Handlung **a** in Zustand **s** durchzuführen und Zustand **s'** zu erreichen. In unserem Beispiel könnte die Kostenfunktion die #Meilen/#Kilometer zurück liefern, oder z.B. die benötigte Zeit, etc.

Die Sequenz and Handlungen nennt man auch **Pfad** (path). Nehmen wir an die Kosten sind additiv, dann ist eine **optimale Lösung** (optimal solution) jene Lösung mit der geringsten Summe an Kosten entlang eines Pfades unter allen möglichen Lösungen.

Problemformulierung

Unsere Problemformulierung nach Bucharest zu kommen ist eine Modell, also eine **Abstraktion** der Wirklichkeit. Die atomare Zustandsbeschreibung von beispielsweise Arad umfasst natürlich nicht alle Details einer tatsächlichen Reise (z.B. Wetter, etc.).

Für unser Beispiel sind alle zusätzlichen Details jedoch irrelevant. Dieses “Weglassen” od. “Reduzieren” nicht notwendiger Details nennt man **Abstraktion**. Eine “gute” Problembeschreibung hat gerade den “richtigen” Detailgrad (oftmals die schwierigste Aufgabe).

Eine Abstraktion ist dann **brauchbar** (useful) wenn das Durchführen der Handlungen in der Lösung einfacher ist als im Originalproblem.

Problemformulierung

Eine Abstraktion ist **valide** (valid), wenn eine Lösung in der Abstraktion des Problems in eine Lösung in der detaillierteren Welt übergeführt werden kann.

Im Prozess ein konkretes Problem abstrahiert zu betrachten, versuchen wir also (1) soweit wie möglich Details zu entfernen, (2) Handlungen einfach durchführbar zu halten und (3) natürlich die Validität nicht zu verletzen.

Problem-Beispiele

Wir unterscheiden zwischen **standardisierten** und **“real-world”** Problemen.

Standardisierte Probleme dienen dazu, verschiedene Problemlösemethoden zu testen und zu evaluieren. “Real-world” Probleme sind jene die auch tatsächlich in der Realität vorkommen (meist individuelle Probleme mit bestimmten Sensoren, etc.).

Problem-Beispiele

Standardisierte Probleme

Grid-world: 2-dimensionales Array an quadratischen Kacheln; ein Agent kann sich zwischen diesen Kacheln bewegen, typischerweise horizontal od. vertikal von einer Kachel zu einer (hindernisfreien) anderen Kachel. In Kacheln können sich Objekte befinden, die der Agent bewegen od. verschieben kann (od. auf die er in sonnst einer Art und Weise einwirken kann). Wände od. Hindernisse verhindern, dass sich ein Agent von einer Kachel zur Nächsten bewegen kann.

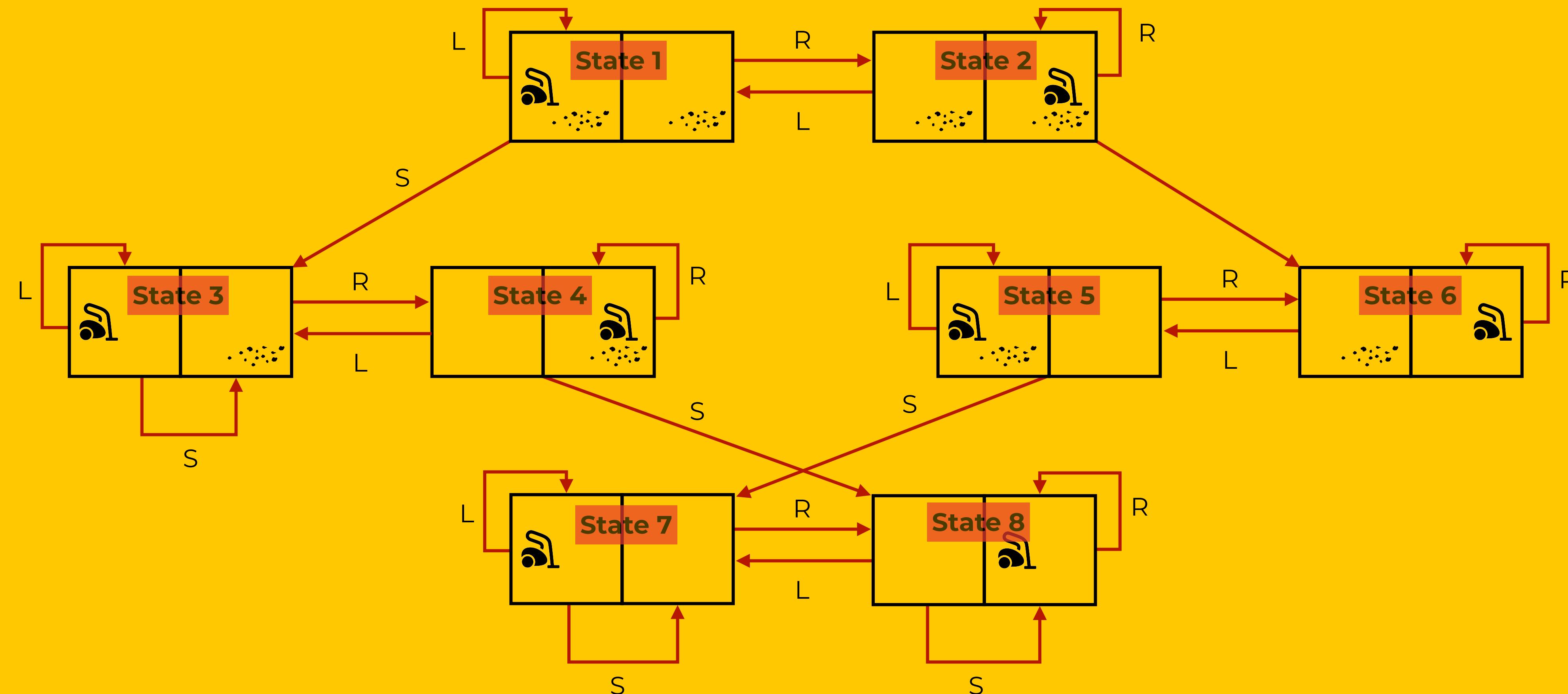
Ein klassisches Beispiel ist die Staubsaugerwelt von zuvor.

Problem-Beispiele

Formuliert als ein Grid-world problem, haben wir für unser Staubsaugerwelt Beispiel folgende Beschreibung:

- **Umgebungszustände** (states): ein Zustand beschreibt welche Objekte sich in welchen Kacheln befinden; Objekte sind hier der Agent und Schmutz. Bei zwei Kacheln (A, B) kann der Agent entweder in A od. B sein und A bzw. B können schmutzig od. sauber sein. Demnach haben wir $2 \cdot 2 \cdot 2 = 2^3 = 8$ Zustände (siehe nächste Folie).

Problem-Beispiele



Problem-Beispiele

- **Initialzustand** (initial state): Jeder Zustand kann Initialzustand sein
- **Handlungen** (actions): “saugen”, “bewege-links”, “bewege-rechts”
- **Übergangsmodell** (transition model): “saugen” entfernt Schmutz, etc.
- **Zielzustände** (goal states): Jene Zustände in denen alle Kacheln sauber sind
- **Handlungs-Kostenfunktion** (action costs): jede Handlung kostet +1.