Algorithmen & Datenstrukturen

Sebastian Wild
wild@cs.uni-kl.de

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

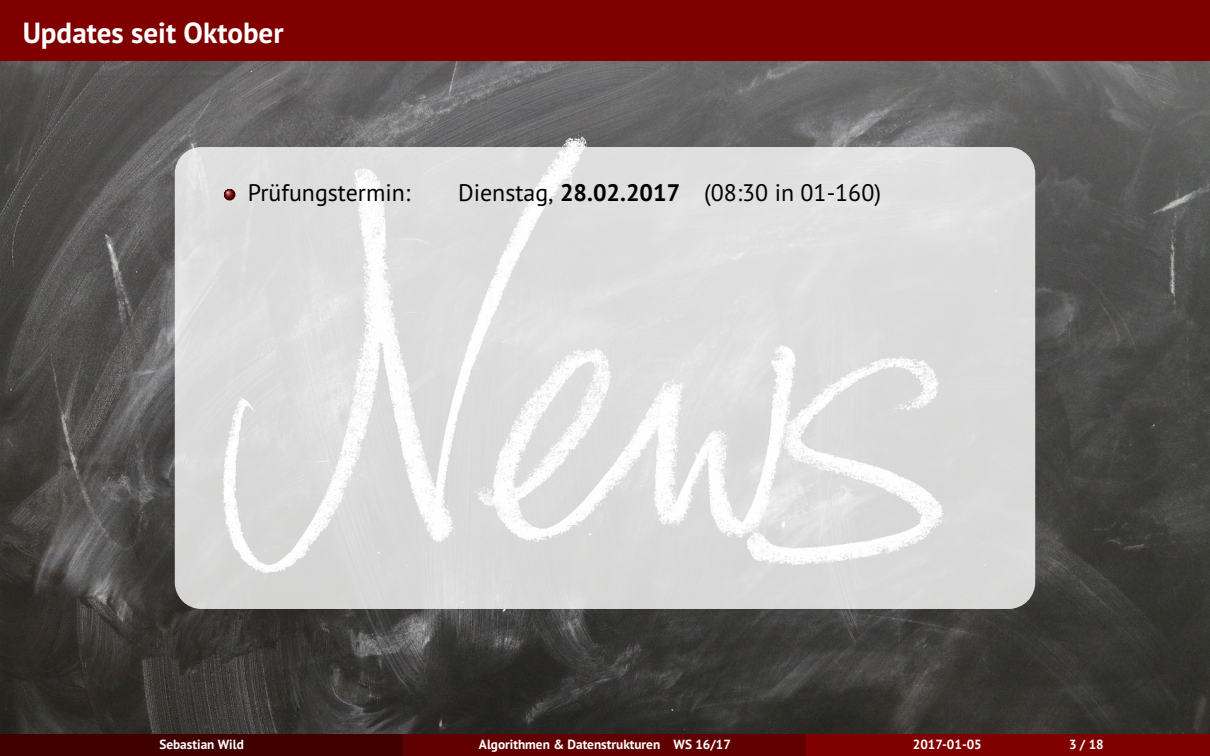
Fachbereich Informatik

05. Januar 2017


0
Organisatorisches

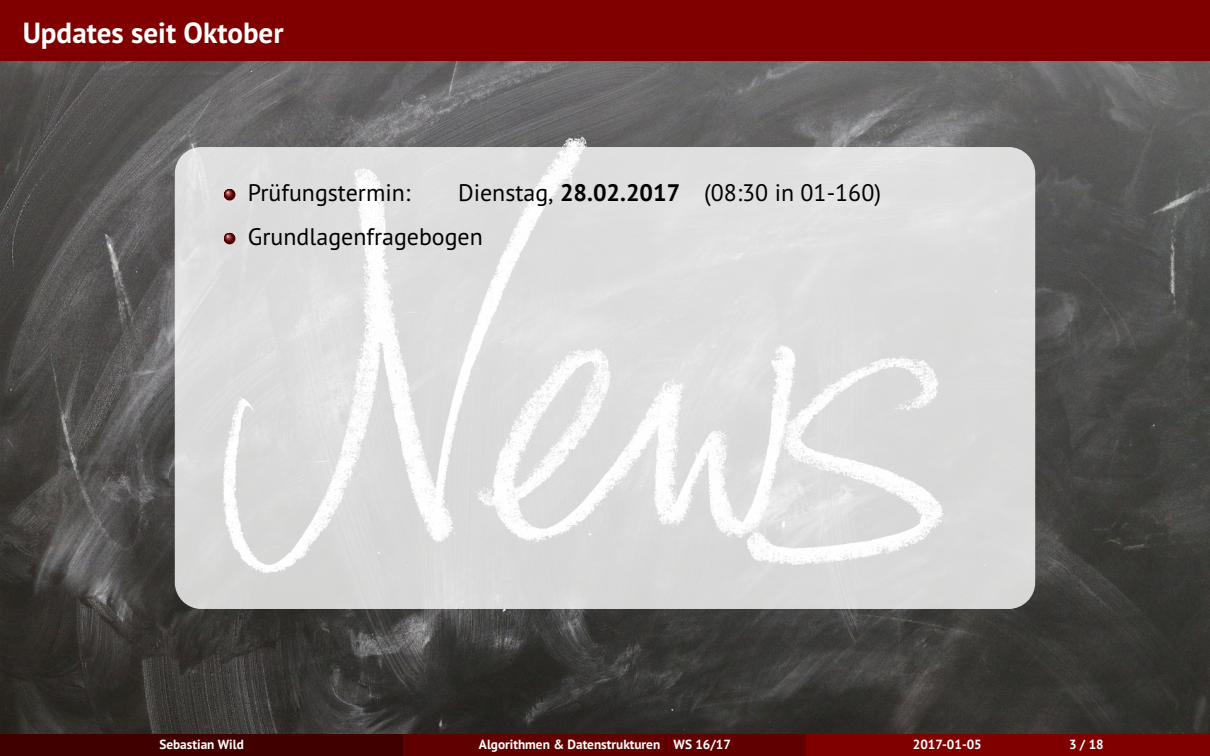
Updates seit Oktober

Updates seit Oktober



- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)

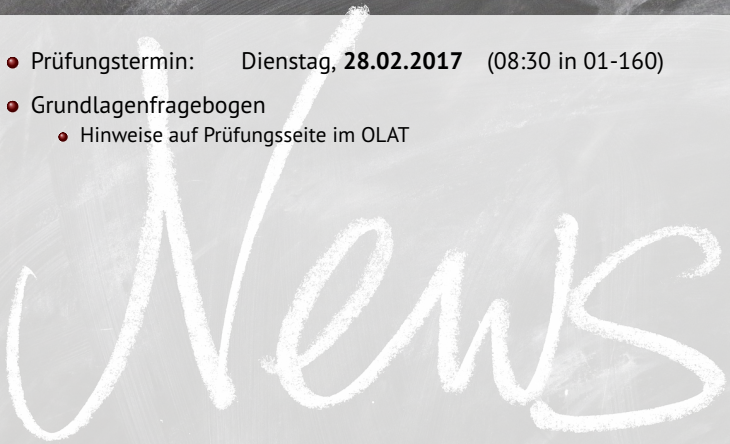
Updates seit Oktober



- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen

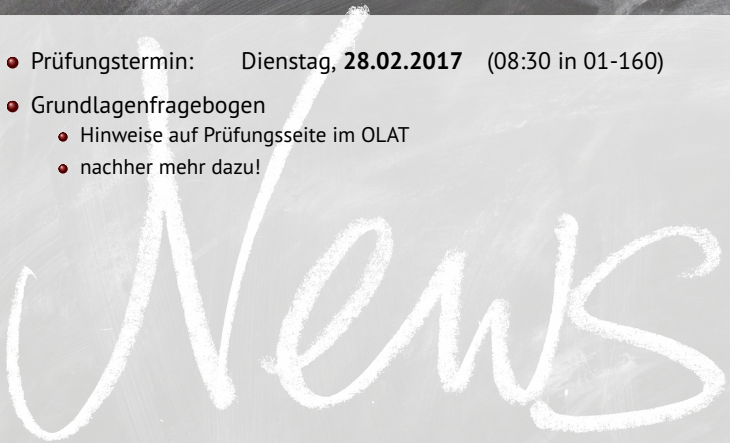
Updates seit Oktober

- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen
  - Hinweise auf Prüfungsseite im OLAT



Updates seit Oktober

- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen
  - Hinweise auf Prüfungsseite im OLAT
  - nachher mehr dazu!



Updates seit Oktober

- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen
  - Hinweise auf Prüfungsseite im OLAT
  - nachher mehr dazu!
- Programmierprojekt 

Updates seit Oktober

- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen
  - Hinweise auf Prüfungsseite im OLAT
  - nachher mehr dazu!
- Programmierprojekt 
  - Maze Competition **nächsten Mittwoch** (11.01.) in 46-260.

Updates seit Oktober

- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen
  - Hinweise auf Prüfungsseite im OLAT
  - nachher mehr dazu!
- Programmierprojekt 
  - Maze Competition **nächsten Mittwoch** (11.01.) in 46-260.
  - danach in diesem Slot: Übungsstunde

Updates seit Oktober

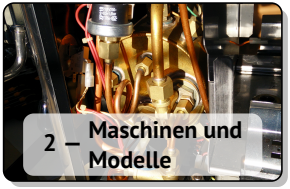
- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen
  - Hinweise auf Prüfungsseite im OLAT
  - nachher mehr dazu!
- Programmierprojekt 
  - Maze Competition **nächsten Mittwoch** (11.01.) in 46-260.
  - danach in diesem Slot: Übungsstunde
- voraussichtlich kein Donnerstagstermin für Vorlesungen (außer heute)

Updates seit Oktober

- Prüfungstermin: Dienstag, **28.02.2017** (08:30 in 01-160)
- Grundlagenfragebogen
  - Hinweise auf Prüfungsseite im OLAT
  - nachher mehr dazu!
- Programmierprojekt 
  - Maze Competition **nächsten Mittwoch** (11.01.) in 46-260.
  - danach in diesem Slot: Übungsstunde
- voraussichtlich kein Donnerstagstermin für Vorlesungen (außer heute)
- jetzt offiziell: 

Überblick über den Kurs

1 — Motivation


2 — Maschinen und Modelle

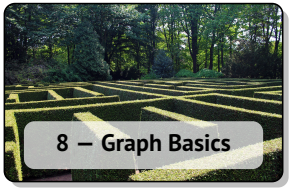

3 — Analyse von Algorithmen

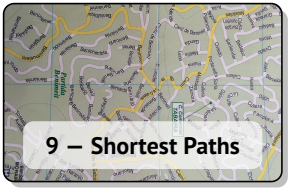

4 — Elementare Datenstrukturen


5 — Sortieren


6 — Binäre Suchbäume


7 — Hashing


8 — Graph Basics

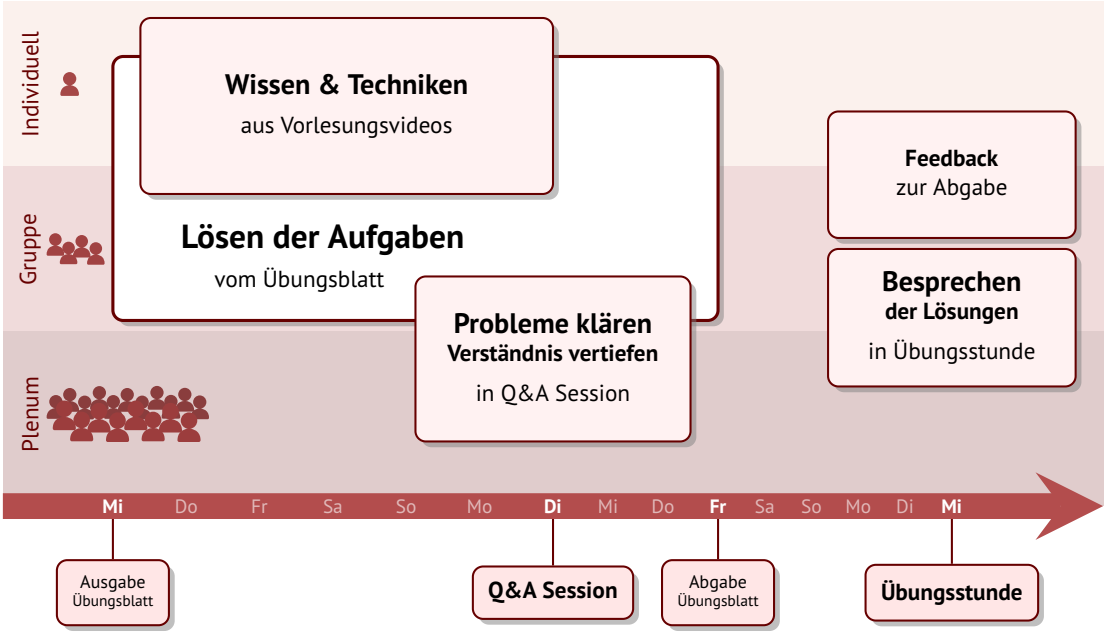

9 — Shortest Paths


10 — Dynamic Programming


11 — Computability & Intractability

Images from https://www.flickr.com/photos/drywall/747943436/, pixabay, http://www.palmpedia.net/wiki/Hyphaene_compressa

Ablaufplan


Individuell
Gruppe
Plenum
Wissen & Techniken
aus Vorlesungsvideos
Lösen der Aufgaben
vom Übungsblatt
Probleme klären
Verständnis vertiefen
in Q&A Session
Feedback
zur Abgabe
Besprechen
der Lösungen
in Übungsstunde
Mi Do Fr Sa So Mo Di Mi Do Fr Sa So Mo Di Mi
Ausgabe Übungsblatt
Q&A Session
Abgabe Übungsblatt
Übungsstunde

Komponenten und ihre Ziele

- **Aufgaben auf Übungsblättern**
  Problem-zentriertes Arbeiten (⟶ deshalb Übungsblatt ganz am Anfang),
  Erwerb von **Fähigkeiten** beim Lösen in Kleingruppe

Komponenten und ihre Ziele

- **Aufgaben auf Übungsblättern**
  Problem-zentriertes Arbeiten (⟶ deshalb Übungsblatt ganz am Anfang),
  Erwerb von **Fähigkeiten** beim Lösen in Kleingruppe
- **Videos** (Alternative: Kapitel aus Lehrbuch) ← gibts auf deutsch
  Vermittlung von **Wissen**, *jeder in seinem Tempo* (innerhalb einer Lektion)

Komponenten und ihre Ziele

- **Aufgaben auf Übungsblättern**
  Problem-zentriertes Arbeiten (⟶ deshalb Übungsblatt ganz am Anfang),
  Erwerb von **Fähigkeiten** beim Lösen in Kleingruppe
- **Videos** (Alternative: Kapitel aus Lehrbuch) ← gibts auf deutsch
  Vermittlung von **Wissen**, *jeder in seinem Tempo* (innerhalb einer Lektion)
- **„Aufgaben zur Vorbereitung“** zu Videos
  Mindestaktivierung während des Zuhörens
  „das solltest ihr *mindestens* aus den Videos mitnehmen“; oft etwas oberflächlich

  Antworten mitschreiben! Nicht passiv berieseln lassen („Hirn aus“)
  (sonst ist die Zeit vertan)

Komponenten und ihre Ziele

- **Aufgaben auf Übungsblättern**
  Problem-zentriertes Arbeiten (⟶ deshalb Übungsblatt ganz am Anfang),
  Erwerb von **Fähigkeiten** beim Lösen in Kleingruppe
- **Videos** (Alternative: Kapitel aus Lehrbuch) ← gibts auf deutsch
  Vermittlung von **Wissen**, *jeder in seinem Tempo* (innerhalb einer Lektion)
- **„Aufgaben zur Vorbereitung“** zu Videos
  Mindestaktivierung während des Zuhörens
  „das solltest ihr *mindestens* aus den Videos mitnehmen“; oft etwas oberflächlich
  Antworten mitschreiben! Nicht passiv berieseln lassen („Hirn aus“)
  (sonst ist die Zeit vertan)
- **Kollaborative Fragensammlung** (Piratenpad)
  Unklarheiten und **weiterführende Fragen** sammeln, direkt beim Schauen
  Feedback zu Fragen und Antworten **von Mitstudis** (geben und bekommen!)
  Ranking der Fragen für Q&A Session

Komponenten und ihre Ziele

- **Aufgaben auf Übungsblättern**
  Problem-zentriertes Arbeiten (⟶ deshalb Übungsblatt ganz am Anfang),
  Erwerb von **Fähigkeiten** beim Lösen in Kleingruppe
- **Videos** (Alternative: Kapitel aus Lehrbuch) ← gibts auf deutsch
  Vermittlung von **Wissen**, *jeder in seinem Tempo* (innerhalb einer Lektion)
- **„Aufgaben zur Vorbereitung“** zu Videos
  Mindestaktivierung während des Zuhörens
  „das solltest ihr *mindestens* aus den Videos mitnehmen“; oft etwas oberflächlich

  Antworten mitschreiben! Nicht passiv berieseln lassen („Hirn aus“)
  (sonst ist die Zeit vertan)
- **Kollaborative Fragensammlung** (Piratenpad)
  Unklarheiten und **weiterführende Fragen** sammeln, direkt beim Schauen
  Feedback zu Fragen und Antworten **von Mitstudis** (geben und bekommen!)
  Ranking der Fragen für Q&A Session

  Evtl. fortführen während Q&A Session als Gedächtnisstütze / grobes Protokoll?

Komponenten und ihre Ziele

- **Aufgaben auf Übungsblättern**
  Problem-zentriertes Arbeiten (→ deshalb Übungsblatt ganz am Anfang),
  Erwerb von **Fähigkeiten** beim Lösen in Kleingruppe
- **Videos** (Alternative: Kapitel aus Lehrbuch) ← gibts auf deutsch
  Vermittlung von **Wissen**, *jeder in seinem Tempo* (innerhalb einer Lektion)
- **„Aufgaben zur Vorbereitung“** zu Videos
  Mindestaktivierung während des Zuhörens
  „das solltest ihr *mindestens* aus den Videos mitnehmen“; oft etwas oberflächlich

  Antworten mitschreiben! Nicht passiv berieseln lassen („Hirn aus“)
  (sonst ist die Zeit vertan)
- **Kollaborative Fragensammlung** (Piratenpad)
  Unklarheiten und **weiterführende Fragen** sammeln, direkt beim Schauen
  Feedback zu Fragen und Antworten **von Mitstudis** (geben und bekommen!)
  Ranking der Fragen für Q&A Session

  Evtl. fortführen während Q&A Session als Gedächtnisstütze / grobes Protokoll?
- **Q&A Sessions**
  Klären von schwierigen Fragen, **Vertiefen** des Verständnisses
  nach den Videos, während Bearbeitung der Übungsaufgaben → ihr seid „im Stoff“

Q&A-Modus

Ziel: Verständnis vertiefen $\rightsquigarrow$ Ihr seid **aktiv**.

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.
  inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$ hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet $\rightsquigarrow$ keine Scheu!
- Wir sind alle erwachsen
  $\rightsquigarrow$ keine Anwesenheitspflicht
  (auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

Q&A-Modus

Ziel: Verständnis vertiefen $\rightsquigarrow$ Ihr seid **aktiv**.

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.
  inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$ hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet $\rightsquigarrow$ keine Scheu!
- Wir sind alle erwachsen
  $\rightsquigarrow$ keine Anwesenheitspflicht
  (auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

Q&A-Modus

Ziel: Verständnis vertiefen $\rightsquigarrow$ Ihr seid **aktiv**.

≠ alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.
  inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$ hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet $\rightsquigarrow$ keine Scheu!
- Wir sind alle erwachsen
  $\rightsquigarrow$ keine Anwesenheitspflicht
  (auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

Q&A-Modus

Ziel: Verständnis vertiefen $\rightsquigarrow$ Ihr seid **aktiv**.

$\neq$ alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.
  inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$ hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet $\rightsquigarrow$ keine Scheu!
- Wir sind alle erwachsen
  $\rightsquigarrow$ keine Anwesenheitspflicht
  (auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

Q&A-Modus

Ziel: Verständnis vertiefen $\rightsquigarrow$ Ihr seid **aktiv**.

$\neq$ alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.
  inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$ hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet $\rightsquigarrow$ keine Scheu!
- Wir sind alle erwachsen
  $\rightsquigarrow$ keine Anwesenheitspflicht
  (auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

Q&A-Modus

Ziel: Verständnis vertiefen $\rightsquigarrow$ Ihr seid **aktiv**.

$\neq$ alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.
  inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$ hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet $\rightsquigarrow$ keine Scheu!
- Wir sind alle erwachsen
  $\rightsquigarrow$ keine Anwesenheitspflicht
  (auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

Q&A-Modus

Ziel: Verständnis vertiefen $\rightsquigarrow$ Ihr seid **aktiv**.

≠ alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.
  inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$ hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet $\rightsquigarrow$ keine Scheu!
- Wir sind alle erwachsen
  $\rightsquigarrow$ keine Anwesenheitspflicht
  (auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

Q&A-Modus

Ziel: Verständnis vertiefen $\rightsquigarrow$ Ihr seid **aktiv**.

≠ alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.
  inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$ hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet $\rightsquigarrow$ keine Scheu!
- Wir sind alle erwachsen
  $\rightsquigarrow$ keine Anwesenheitspflicht
  (auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

Q&A-Modus

Ziel: Verständnis vertiefen $\rightsquigarrow$ Ihr seid **aktiv**.

$\neq$ alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.

  inklusive „Aufgaben zur Vorbereitung“!

$\rightsquigarrow$ hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet $\rightsquigarrow$ keine Scheu!
- Wir sind alle erwachsen

  $\rightsquigarrow$ keine Anwesenheitspflicht

  (auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

Q&A-Modus

Ziel: Verständnis vertiefen ⇝ Ihr seid **aktiv**.

≠ alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.
  inklusive „Aufgaben zur Vorbereitung“!

⇝ hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet ⇝ keine Scheu!
- Wir sind alle erwachsen
  ⇝ keine Anwesenheitspflicht
  (auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

Vorlesungmodus

Ziel: neuen Stoff lernen
⇝ konzentriertes **Mitdenken** ermöglichen

- keine Laptops, Tablets, Handys,
  Smartphones, Phablets, Smartwatches, Mobilkonsolen, ...

Übungen

- ähnlich zu Q&A-Sessions
- **Zitieren statt Plagiieren!**
  Ich habe (etwas) Verständnis für Faulheit,
  aber keins für Ideenklau!

Q&A-Modus

Ziel: Verständnis vertiefen ⇝ Ihr seid **aktiv**.

≠ alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.
  inklusive „Aufgaben zur Vorbereitung“!

⇝ hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet ⇝ keine Scheu!
- Wir sind alle erwachsen
  ⇝ keine Anwesenheitspflicht
  (auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

Vorlesungmodus

Ziel: neuen Stoff lernen
⇝ konzentriertes **Mitdenken** ermöglichen

- keine Laptops, Tablets, Handys,
  Smartphones, Phablets, Smartwatches, Mobilkonsolen, ...

Übungen

- ähnlich zu Q&A-Sessions
- **Zitieren statt Plagiieren!**
  Ich habe (etwas) Verständnis für Faulheit,
  aber keins für Ideenklau!

Q&A-Modus

Ziel: Verständnis vertiefen ⇝ Ihr seid **aktiv**.

≠ alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.
  inklusive „Aufgaben zur Vorbereitung“!

⇝ hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet ⇝ keine Scheu!
- Wir sind alle erwachsen
  ⇝ keine Anwesenheitspflicht
  (auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

Vorlesungmodus

Ziel: neuen Stoff lernen
⇝ konzentriertes **Mitdenken** ermöglichen

- keine Laptops, Tablets, Handys,
  Smartphones, Phablets, Smartwatches, Mobilkonsolen, ...

Übungen

- ähnlich zu Q&A-Sessions
- **Zitieren statt Plagiieren!**
  Ich habe (etwas) Verständnis für Faulheit,
  aber keins für Ideenklau!

Q&A-Modus

Ziel: Verständnis vertiefen ⇝ Ihr seid **aktiv**.

≠ alles verstanden

- Ich setze voraus, dass ihr Videos / Lesestoff kennt.
  inklusive „Aufgaben zur Vorbereitung“!

⇝ hier: Verständnisfragen klären

- Wechsel zwischen Gruppendiskussion und Plenum
- Beiträge werden nicht bewertet ⇝ keine Scheu!
- Wir sind alle erwachsen
  ⇝ keine Anwesenheitspflicht
  (auch keine Bestehensgarantie ohne Anstrengung)
- Wir fangen pünktlich an.

Vorlesungmodus

Ziel: neuen Stoff lernen
⇝ konzentriertes **Mitdenken** ermöglichen

- keine Laptops, Tablets, Handys,
  Smartphones, Phablets, Smartwatches, Mobilkonsolen, ...

Übungen

- ähnlich zu Q&A-Sessions
- **Zitieren statt Plagiieren!**
  Ich habe (etwas) Verständnis für Faulheit,
  aber keins für Ideenklau!


1
START

Wozu Algorithmik?

Why study algorithms?

Their impact is broad and far-reaching.

Internet. Web search, packet routing, distributed file sharing, ...

Biology. Human genome project, protein folding, ...

Computers. Circuit layout, file system, compilers, ...

Computer graphics. Movies, video games, virtual reality, ...

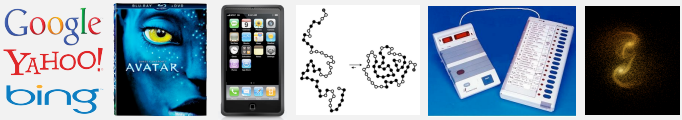
Security. Cell phones, e-commerce, voting machines, ...

Multimedia. MP3, JPG, DivX, HDTV, face recognition, ...

Social networks. Recommendations, news feeds, advertisements, ...

Physics. N-body simulation, particle collision simulation, ...

⋮



3

http://algs4.cs.princeton.edu/lectures/00Intro.pdf, page 3

Why study algorithms?

Their impact is broad and far-reaching.

Mysterious algorithm was 4% of trading activity last week

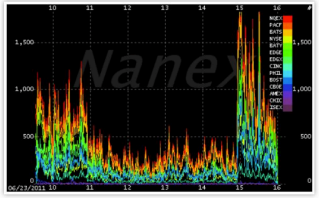
October 11, 2012

A single mysterious computer program that placed orders — and then subsequently canceled them — made up 4 percent of all quote traffic in the U.S. stock market last week, according to the top tracker of high-frequency trading activity.

The motive of the algorithm is still unclear, *CNBC* reports.

The program placed orders in 25-millisecond bursts involving about 500 stocks, according to Nanex, a market data firm. The algorithm never executed a single trade, and it abruptly ended at about 10:30 a.m. ET Friday.

[+]



Generic high frequency rrading chart (credit: Nanex)

"My guess is that the algo was testing the market, as high-frequency frequently does," says Jon Najarian, co-founder of TradeMonster.com. "As soon as they add bandwidth, the HFT crowd sees how quickly they can top out to create latency." (*Read More*: Unclear What Caused Kraft Spike: Nanex Founder.)

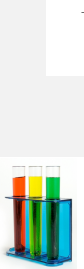
4

http://algs4.cs.princeton.edu/lectures/00Intro.pdf, page 4

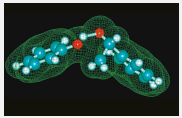
Why study algorithms?

They may unlock the secrets of life and of the universe.

> *" Computer models mirroring real life have become crucial for most advances made in chemistry today.... Today the computer is just as important a tool for chemists as the test tube. "*
>
> — *Royal Swedish Academy of Sciences*
> (*Nobel Prize in Chemistry 2013*)

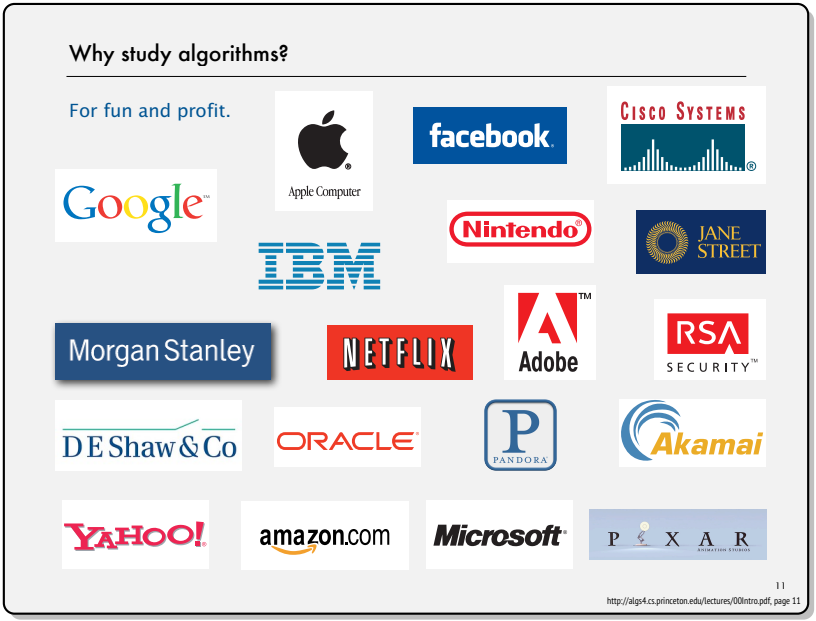


Martin Karplus, Michael Levitt, and Arieh Warshel



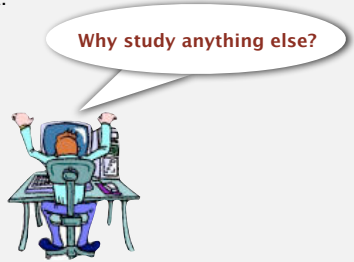
8

http://algs4.cs.princeton.edu/lectures/00Intro.pdf, page 8


Why study algorithms?
For fun and profit.
Google
Apple Computer
facebook
Cisco Systems
IBM
Nintendo
Jane Street
Morgan Stanley
NETFLIX
Adobe
RSA Security
DE Shaw & Co
ORACLE
Pandora
Akamai
YAHOO!
amazon.com
Microsoft
PIXAR
11
http://algs4.cs.princeton.edu/lectures/00Intro.pdf, page 11

Why study algorithms?

- Their impact is broad and far-reaching.
- Old roots, new opportunities.
- For intellectual stimulation.
- To become a proficient programmer.
- They may unlock the secrets of life and of the universe.
- To solve problems that could not otherwise be addressed.
- Everybody else is doing it.
- For fun and profit.



12

http://algs4.cs.princeton.edu/lectures/00Intro.pdf, page 12

Wozu Algorithmik?

Ziele dieser Vorlesung:

1. **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
   - **best practices:** Liste von A. & DS. für Standardprobleme
   - intellektuelle Errungenschaften der Informatik
2. Algorithmen **bewerten** können
3. mit anderen Experten über Algorithmen **reden**
   - Computational Thinking prägt unsere Welt.
   - Vokabular aufbauen ($\rightsquigarrow$ Baukasten)
   - präzise Beschreibungen verstehen und erstellen, z. B. Javadoc der Java API (siehe nächste Folie)
   - erkennen, wenn jemand „Mist" erzählt

Wozu Algorithmik?

Ziele dieser Vorlesung:

1. **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
   - **best practices:** Liste von A. & DS. für Standardprobleme
   - intellektuelle Errungenschaften der Informatik
2. Algorithmen **bewerten** können
3. mit anderen Experten über Algorithmen **reden**
   - Computational Thinking prägt unsere Welt.
   - Vokabular aufbauen ($\rightsquigarrow$ Baukasten)
   - präzise Beschreibungen verstehen und erstellen, z. B. Javadoc der Java API (siehe nächste Folie)
   - erkennen, wenn jemand „Mist" erzählt

Wozu Algorithmik?

Ziele dieser Vorlesung:

1. **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
   - **best practices:** Liste von A. & DS. für Standardprobleme
   - intellektuelle Errungenschaften der Informatik
2. Algorithmen **bewerten** können
3. mit anderen Experten über Algorithmen **reden**
   - Computational Thinking prägt unsere Welt.
   - Vokabular aufbauen ($\rightsquigarrow$ Baukasten)
   - präzise Beschreibungen verstehen und erstellen, z. B. Javadoc der Java API (siehe nächste Folie)
   - erkennen, wenn jemand „Mist" erzählt

Wozu Algorithmik?

Ziele dieser Vorlesung:

1. **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
   - **best practices:** Liste von A. & DS. für Standardprobleme
   - intellektuelle Errungenschaften der Informatik
2. Algorithmen **bewerten** können
3. mit anderen Experten über Algorithmen **reden**
   - Computational Thinking prägt unsere Welt.
   - Vokabular aufbauen ($\rightsquigarrow$ Baukasten)
   - präzise Beschreibungen verstehen und erstellen, z. B. Javadoc der Java API (siehe nächste Folie)
   - erkennen, wenn jemand „Mist“ erzählt

Wozu Algorithmik?

Ziele dieser Vorlesung:

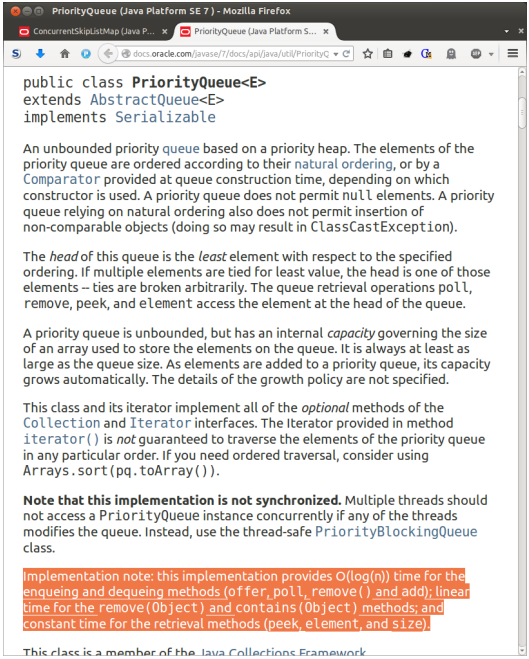
1. **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
   - **best practices:** Liste von A. & DS. für Standardprobleme
   - intellektuelle Errungenschaften der Informatik
2. Algorithmen **bewerten** können
3. mit anderen Experten über Algorithmen **reden**
   - Computational Thinking prägt unsere Welt.
   - Vokabular aufbauen (⇝ Baukasten)
   - präzise Beschreibungen verstehen und erstellen, z. B. Javadoc der Java API (siehe nächste Folie)
   - erkennen, wenn jemand „Mist“ erzählt

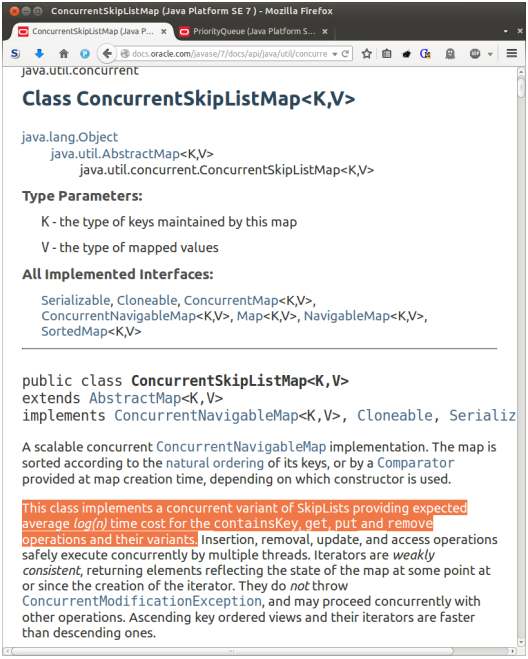
Wozu Algorithmik?

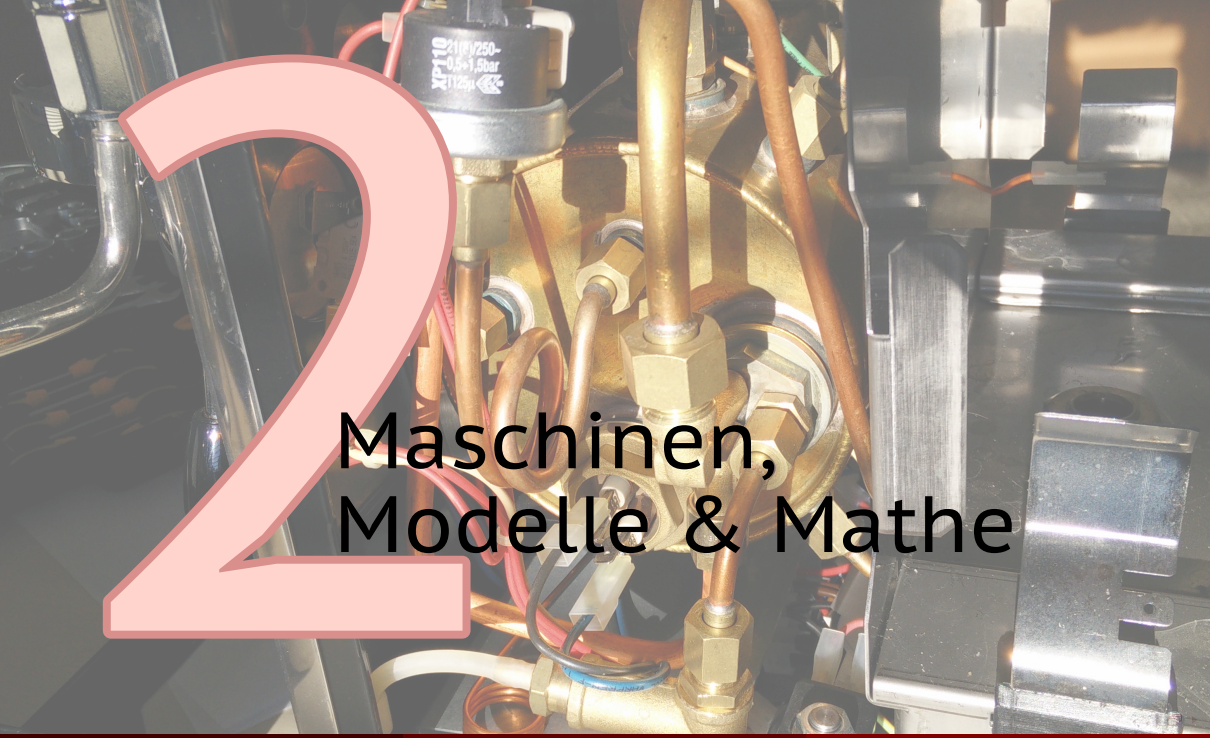
Ziele dieser Vorlesung:

1. **Baukasten** aus Algorithmen und Datenstrukturen aufbauen
   - **best practices:** Liste von A. & DS. für Standardprobleme
   - intellektuelle Errungenschaften der Informatik
2. Algorithmen **bewerten** können
3. mit anderen Experten über Algorithmen **reden**
   - Computational Thinking prägt unsere Welt.
   - Vokabular aufbauen ($\rightsquigarrow$ Baukasten)
   - präzise Beschreibungen verstehen und erstellen, z. B. Javadoc der Java API (siehe nächste Folie)
   - erkennen, wenn jemand „Mist“ erzählt

Beispiele aus der Java API






2
Maschinen,
Modelle & Mathe

Philosophie

A&DS ist Teil eines **wissenschaftlichen** Studiengangs

Philosophie

A&DS ist Teil eines **wissenschaftlichen** Studiengangs

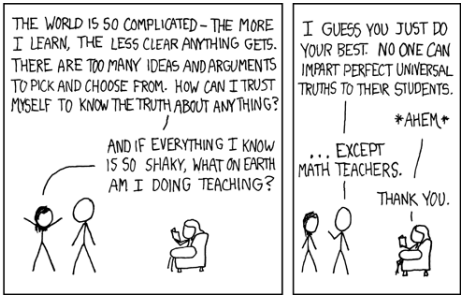
Weniger …

Philosophie

A&DS ist Teil eines **wissenschaftlichen** Studiengangs

Weniger …

…und mehr

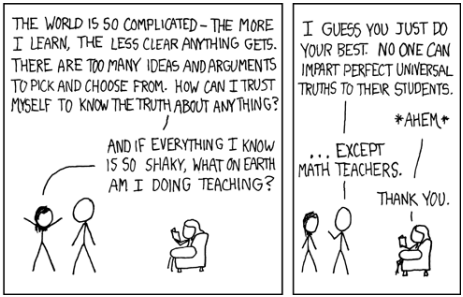


Philosophie

A&DS ist Teil eines **wissenschaftlichen** Studiengangs

Weniger …

…und mehr



↝ Fokus auf "universal truths" der Algorithmik

- Model der Realität
- quantitative Vorhersagen
- Validierung der Modelle in Experimenten

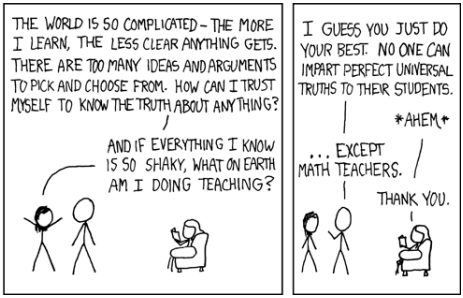
↝ Wir brauchen ein paar mathematische / theoretische Grundlagen.

Philosophie

A&DS ist Teil eines **wissenschaftlichen** Studiengangs

Weniger …

…und mehr



↝ Fokus auf "universal truths" der Algorithmik

- Model der Realität
- quantitative Vorhersagen
- Validierung der Modelle in Experimenten

↝ Wir brauchen ein paar mathematische / theoretische Grundlagen.

Was ist ein Algorithmus?

Was ist ein Algorithmus?

Welche Definitionen für „Algorithmus“ gaben die Videos?

Was ist ein Algorithmus?

Anschaulich:
Folge von Anweisungen $\approx$ Kochrezept

Was ist ein Algorithmus?

Anschaulich:
Folge von Anweisungen ≈ Kochrezept

Genauer:

1. mechanisch nachvollziehbar
   ⇝ kein „gesunder Menschenverstand“ nötig
2. endliche Beschreibung
3. löst ein Problem, d. h. ganze Klasse von Probleminstanzen

Was ist ein Algorithmus?

Anschaulich:
Folge von Anweisungen $\approx$ Kochrezept

Genauer:

1. mechanisch nachvollziehbar
   $\rightsquigarrow$ kein „gesunder Menschenverstand“ nötig
2. endliche Beschreibung
3. löst ein Problem, d. h. ganze Klasse von Probleminstanzen

Was ist ein Algorithmus?

Anschaulich:
Folge von Anweisungen $\approx$ Kochrezept

Genauer:

1. mechanisch nachvollziehbar
   $\rightsquigarrow$ kein „gesunder Menschenverstand“ nötig
2. endliche Beschreibung
3. löst ein Problem, d. h. ganze Klasse von Probleminstanzen

Was ist ein Algorithmus?

Anschaulich:
Folge von Anweisungen $\approx$ Kochrezept

Genauer:

1. mechanisch nachvollziehbar
   $\rightsquigarrow$ kein „gesunder Menschenverstand“ nötig
2. endliche Beschreibung $\neq$ endliche Berechnung!
3. löst ein Problem, d. h. ganze Klasse von Probleminstanzen

Was ist ein Algorithmus?

Anschaulich:
Folge von Anweisungen $\approx$ Kochrezept

Genauer:

1. mechanisch nachvollziehbar
   $\rightsquigarrow$ kein „gesunder Menschenverstand“ nötig
2. endliche Beschreibung $\neq$ endliche Berechnung!
3. löst ein Problem, d. h. ganze Klasse von Probleminstanzen

Was ist ein Algorithmus?

Anschaulich:
Folge von Anweisungen $\approx$ Kochrezept

Genauer:

z. B. Java Programm

1. mechanisch nachvollziehbar
   $\rightsquigarrow$ kein „gesunder Menschenverstand“ nötig
2. endliche Beschreibung $\neq$ endliche Berechnung!
3. löst ein Problem, d. h. ganze Klasse von Probleminstanzen

Was ist ein Algorithmus?

Anschaulich:
Folge von Anweisungen ≈ Kochrezept

Genauer:

z. B. Java Programm

1. mechanisch nachvollziehbar
   ⇝ kein „gesunder Menschenverstand“ nötig
2. endliche Beschreibung ≠ endliche Berechnung!
3. löst ein Problem, d. h. ganze Klasse von Probleminstanzen
   $x + y$, nicht nur $17 + 4$

Was ist ein Algorithmus?

Anschaulich:
Folge von Anweisungen ≈ Kochrezept

Genauer:

z. B. Java Programm

1. mechanisch nachvollziehbar
   ⇝ kein „gesunder Menschenverstand“ nötig
2. endliche Beschreibung ≠ endliche Berechnung!
3. löst ein Problem, d. h. ganze Klasse von Probleminstanzen
   $x + y$, nicht nur $17 + 4$

typisches Beispiel: *Bubblesort*

nicht ein spezielles Programm,
sondern die zugrundeliegende Idee

Was ist ein Algorithmus?

Anschaulich:
Folge von Anweisungen ≈ Kochrezept

Genauer:

z. B. Java Programm

1. mechanisch nachvollziehbar
2. endliche Beschreibung ≠ endliche Berechnung!
3. löst ein Problem, d. h. ganze Klasse von Probleminstanzen

$x + y$, nicht nur $17 + 4$

Welche Algorithmen wurden in den Videos vorgestellt?

typisches Beispiel: *Bubblesort*

nicht ein spezielles Programm,
sondern die zugrundeliegende Idee

Was ist ein Algorithmus?

al-Chwarizmi
Namensgeber für Algorithmus

Anschaulich:
Folge von Anweisungen ≈ Kochrezept

Genauer:

z. B. Java Programm

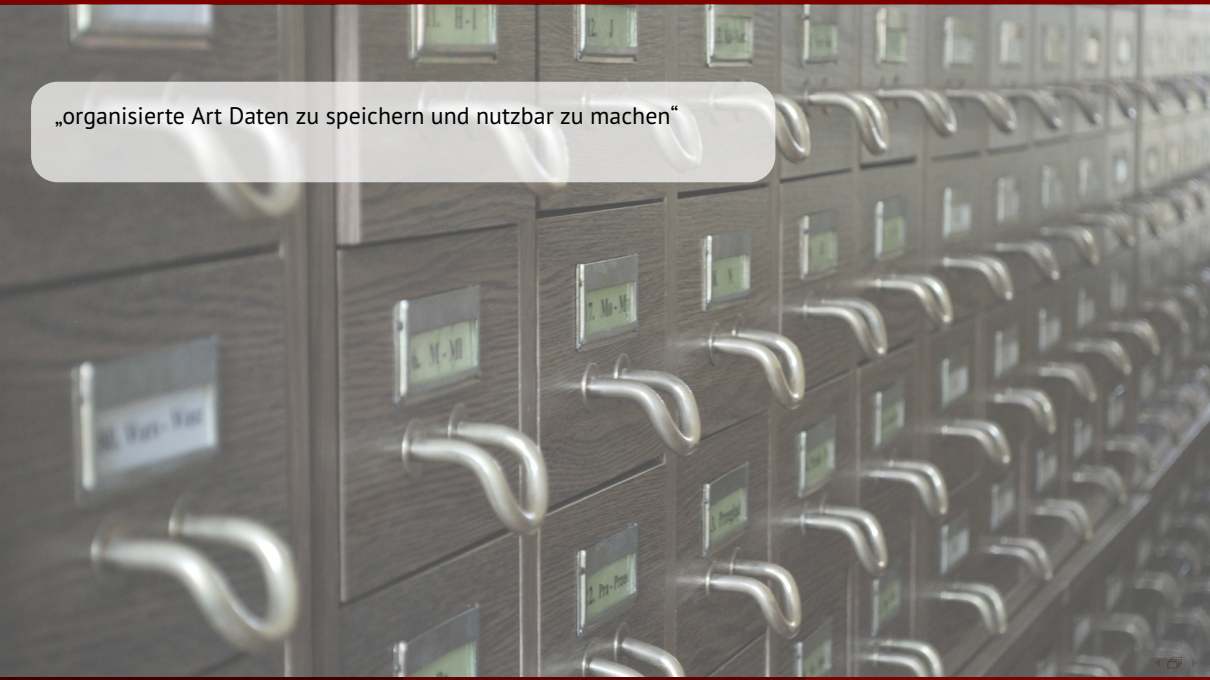
1. mechanisch nachvollziehbar
   ⇝ kein „gesunder Menschenverstand“ nötig
2. endliche Beschreibung ≠ endliche Berechnung!
3. löst ein Problem, d. h. ganze Klasse von Probleminstanzen
   $x + y$, nicht nur $17 + 4$

typisches Beispiel: *Bubblesort*

nicht ein spezielles Programm,
sondern die zugrundeliegende Idee

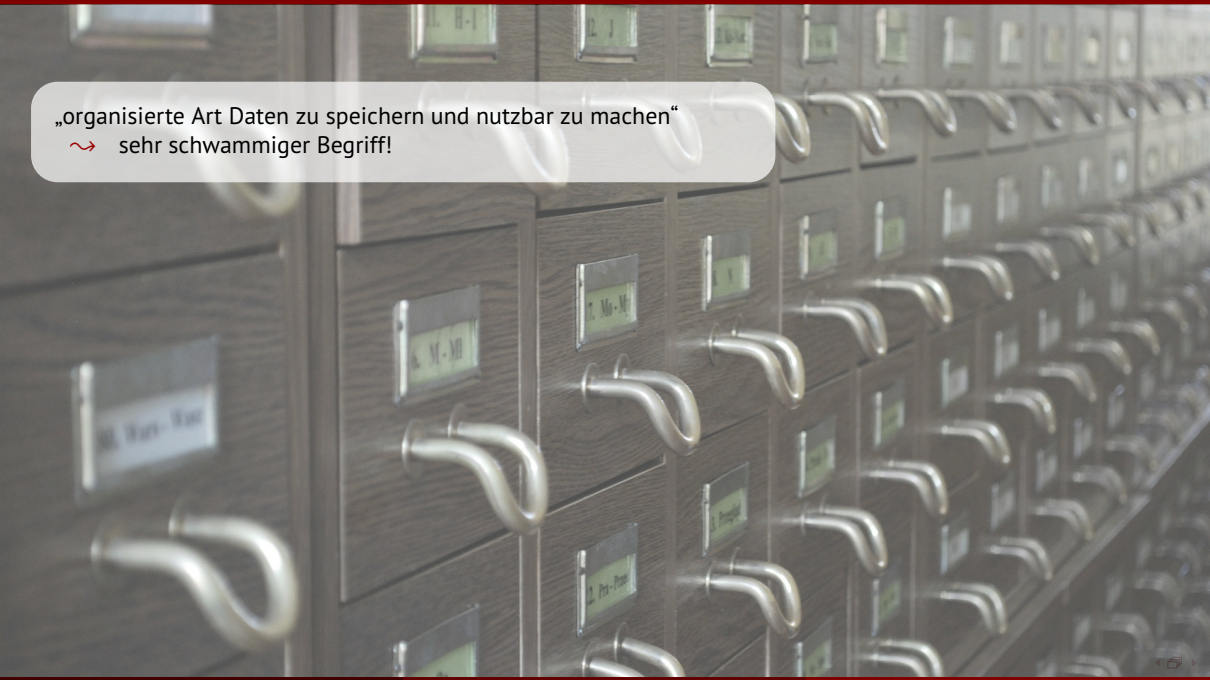
Was ist eine Datenstruktur?

Was ist eine Datenstruktur?



„organisierte Art Daten zu speichern und nutzbar zu machen“

Was ist eine Datenstruktur?



„organisierte Art Daten zu speichern und nutzbar zu machen“

$\rightsquigarrow$ sehr schwammiger Begriff!

Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“

↝ sehr schwammiger Begriff!

Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“

↝ sehr schwammiger Begriff!

Für uns:

Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“
↝ sehr schwammiger Begriff!

Für uns:

application programming interface

- **Interface:** Spezifikation von Operationen API
  = *abstract data type (ADT)*

Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“
↝ sehr schwammiger Begriff!

Für uns:

application programming interface

- **Interface:** Spezifikation von Operationen API
  = *abstract data type (ADT)*

  Beispiel: *Union-Find*

Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“
↝ sehr schwammiger Begriff!

Für uns:

application programming interface

- **Interface:** Spezifikation von Operationen API
  = *abstract data type (ADT)*
  Beispiel: *Union-Find*

- **Implementierung** eines Interface:
  pro Operation ein Algorithmus, der die Spezifikation erfüllt

Was ist eine Datenstruktur?

„organisierte Art Daten zu speichern und nutzbar zu machen“
↝ sehr schwammiger Begriff!

Für uns:

application programming interface

- **Interface:** Spezifikation von Operationen API
  = *abstract data type (ADT)*
  Beispiel: *Union-Find*

- **Implementierung** eines Interface:
  pro Operation ein Algorithmus, der die Spezifikation erfüllt
  Beispiel: *Quick Union*

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/…“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

  Sind am Verhalten für große Eingaben interessiert!

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/…“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert!

Mensch (zu Fuß) oder Zug: Wer ist schneller?

(beide stehen zu Beginn)

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

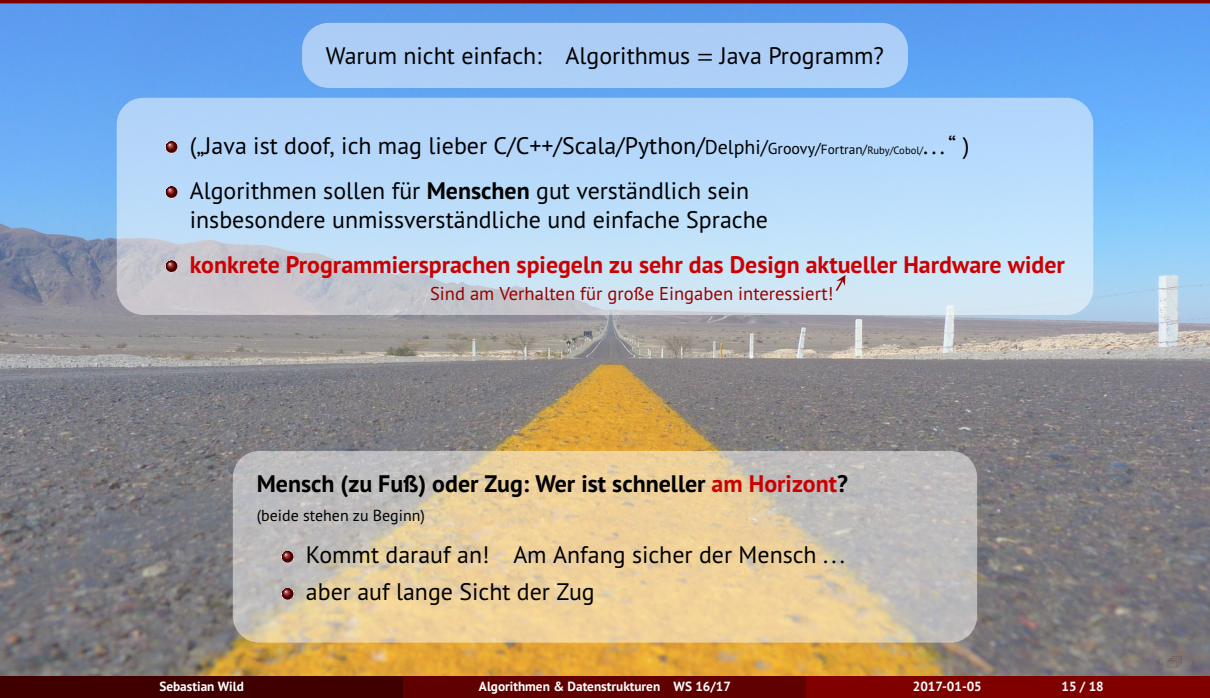
- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/…“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**
  Sind am Verhalten für große Eingaben interessiert!

Mensch (zu Fuß) oder Zug: Wer ist schneller?

(beide stehen zu Beginn)

- Kommt darauf an! Am Anfang sicher der Mensch …

Maschinenmodell



Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/…“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**
  Sind am Verhalten für große Eingaben interessiert!

Mensch (zu Fuß) oder Zug: Wer ist schneller am Horizont?

(beide stehen zu Beginn)

- Kommt darauf an! Am Anfang sicher der Mensch …
- aber auf lange Sicht der Zug

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

  Sind am Verhalten für große Eingaben interessiert!

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/…“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert!

Beispiel: Wie viel Speicher brauchen n natürliche Zahlen?

als Funktion in n

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**
  Sind am Verhalten für große Eingaben interessiert!

Beispiel: Wie viel Speicher brauchen n natürliche Zahlen?

- **Java:** $32n$ bit, da `int` eine 32 Bit Zahl ist.

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert!

Beispiel: Wie viel Speicher brauchen n natürliche Zahlen?

- **Java:** $32n$ bit, da `int` eine 32 Bit Zahl ist.
  = typische Wortgröße aktueller Hardware ← damit kann in einem Schritt gerechnet werden

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert!

Beispiel: Wie viel Speicher brauchen n natürliche Zahlen?

- **Java:** $32n$ bit, da `int` eine 32 Bit Zahl ist.
  = typische Wortgröße aktueller Hardware ← damit kann in einem Schritt gerechnet werden
- **Theorie:** Zahl im Bereich $0..M$ braucht $\lceil \mathrm{ld}(M+1) \rceil$ bits
  also $n \cdot \lceil \mathrm{ld}(M+1) \rceil$ bits

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert!

Beispiel: Wie viel Speicher brauchen n natürliche Zahlen?

- **Java:** $32n$ bit, da `int` eine 32 Bit Zahl ist.
  = typische Wortgröße aktueller Hardware ← damit kann in einem Schritt gerechnet werden
- **Theorie:** Zahl im Bereich $0..M$ braucht $\lceil \mathrm{ld}(M+1) \rceil$ bits
  also $n \cdot \lceil \mathrm{ld}(M+1) \rceil$ bits
- Wenn M klein ist, würde Programmierer aus Effizienzgründen trotzdem 32-bit `int`s wählen.

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert!

Beispiel: Wie viel Speicher brauchen n natürliche Zahlen?

- **Java:** $32n$ bit, da `int` eine 32 Bit Zahl ist.
  = typische Wortgröße aktueller Hardware ← damit kann in einem Schritt gerechnet werden
- **Theorie:** Zahl im Bereich $0..M$ braucht $\lceil \mathrm{ld}(M+1) \rceil$ bits
  also $n \cdot \lceil \mathrm{ld}(M+1) \rceil$ bits
- Wenn M klein ist, würde Programmierer aus
  Effizienzgründen trotzdem 32-bit `int`s wählen.

$\rightsquigarrow$ Halten Wortbreite variabel.

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert!

Beispiel: Wie viel Speicher brauchen n natürliche Zahlen?

- **Java:** $32n$ bit, da `int` eine 32 Bit Zahl ist.
  = typische Wortgröße aktueller Hardware ← damit kann in einem Schritt gerechnet werden
- **Theorie:** Zahl im Bereich $0..M$ braucht $\lceil \mathrm{ld}(M+1) \rceil$ bits
  also $n \cdot \lceil \mathrm{ld}(M+1) \rceil$ bits
- Wenn M klein ist, würde Programmierer aus
  Effizienzgründen trotzdem 32-bit `int`s wählen.

$\rightsquigarrow$ Halten Wortbreite variabel. ← kann sogar mit n mitwachsen

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert!

Beispiel: Wie viel Speicher brauchen n natürliche Zahlen?

- **Java:** $32n$ bit, da `int` eine 32 Bit Zahl ist.
  = typische Wortgröße aktueller Hardware ← damit kann in einem Schritt gerechnet werden
- **Theorie:** Zahl im Bereich $0..M$ braucht $\lceil \mathrm{ld}(M+1) \rceil$ bits
  also $n \cdot \lceil \mathrm{ld}(M+1) \rceil$ bits
- Wenn M klein ist, würde Programmierer aus
  Effizienzgründen trotzdem 32-bit `int`s wählen.

⇝ Halten Wortbreite variabel. ← kann sogar mit n mitwachsen

„Überlebt Pentium 4.“

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/…“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert!

Beispiel: Wie viel Speicher brauchen n natürliche Zahlen?

- **Java:** $32n$ bit, da `int` eine 32 Bit Zahl ist.
  = typische Wortgröße aktueller Hardware ← damit kann in einem Schritt gerechnet werden
- **Theorie:** Zahl im Bereich $0..M$ braucht $\lceil \mathrm{ld}(M+1) \rceil$ bits
  also $n \cdot \lceil \mathrm{ld}(M+1) \rceil$ bits
- Wenn M klein ist, würde Programmierer aus Effizienzgründen trotzdem 32-bit `int`s wählen.

⇝ Halten Wortbreite variabel. ← kann sogar mit n mitwachsen
← „Überlebt Pentium 4.“

Dinge, die wir **nicht** erlauben

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/…“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert!

Beispiel: Wie viel Speicher brauchen n natürliche Zahlen?

- **Java:** $32n$ bit, da `int` eine 32 Bit Zahl ist.
  = typische Wortgröße aktueller Hardware ← damit kann in einem Schritt gerechnet werden
- **Theorie:** Zahl im Bereich $0..M$ braucht $\lceil \mathrm{ld}(M+1) \rceil$ bits
  also $n \cdot \lceil \mathrm{ld}(M+1) \rceil$ bits
- Wenn M klein ist, würde Programmierer aus Effizienzgründen trotzdem 32-bit `int`s wählen.

⇝ Halten Wortbreite variabel. ← kann sogar mit n mitwachsen
← „Überlebt Pentium 4.“

Dinge, die wir **nicht** erlauben

- selbst-modifizierender Code

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

  Sind am Verhalten für große Eingaben interessiert!

Beispiel: Wie viel Speicher brauchen n natürliche Zahlen?

- **Java:** $32n$ bit, da `int` eine 32 Bit Zahl ist.
  = typische Wortgröße aktueller Hardware ← damit kann in einem Schritt gerechnet werden
- **Theorie:** Zahl im Bereich $0..M$ braucht $\lceil \mathrm{ld}(M+1) \rceil$ bits
  also $n \cdot \lceil \mathrm{ld}(M+1) \rceil$ bits
- Wenn M klein ist, würde Programmierer aus Effizienzgründen trotzdem 32-bit `int`s wählen.

⇝ Halten Wortbreite variabel. ← kann sogar mit n mitwachsen ← „Überlebt Pentium 4.“

Dinge, die wir **nicht** erlauben

- selbst-modifizierender Code
- Rechnen mit reellen Zahlen (≠ floating-point)

Maschinenmodell

Warum nicht einfach: Algorithmus = Java Programm?

- („Java ist doof, ich mag lieber C/C++/Scala/Python/Delphi/Groovy/Fortran/Ruby/Cobol/...“)
- Algorithmen sollen für **Menschen** gut verständlich sein
  insbesondere unmissverständliche und einfache Sprache
- **konkrete Programmiersprachen spiegeln zu sehr das Design aktueller Hardware wider**

Sind am Verhalten für große Eingaben interessiert!

Beispiel: Wie viel Speicher brauchen n natürliche Zahlen?

- **Java:** $32n$ bit, da `int` eine 32 Bit Zahl ist.
  = typische Wortgröße aktueller Hardware ← damit kann in einem Schritt gerechnet werden
- **Theorie:** Zahl im Bereich $0..M$ braucht $\lceil \mathrm{ld}(M+1) \rceil$ bits
  also $n \cdot \lceil \mathrm{ld}(M+1) \rceil$ bits
- Wenn M klein ist, würde Programmierer aus Effizienzgründen trotzdem 32-bit `int`s wählen.

⇝ Halten Wortbreite variabel. ← kann sogar mit n mitwachsen ← „Überlebt Pentium 4.“

Dinge, die wir **nicht** erlauben

- selbst-modifizierender Code
- Rechnen mit reellen Zahlen (≠ floating-point)
- Magie

Abstraktion

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren
  - $\rightsquigarrow$ erst über Algorithmen nachdenken

Abstraktion

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren
  - ⇝ erst über Algorithmen nachdenken

Abstraktion

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.

- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren

  ⇝ erst über Algorithmen nachdenken

Abstraktion

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.

- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren

  $\rightsquigarrow$ erst über Algorithmen nachdenken

Abstraktion

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.

- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren

  ⇝ erst über Algorithmen nachdenken

Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank? Antwort

⇝ “look into the box”

top down funktioniert nur, wenn wir überblicken können,
dass und wie Schritte implementiert werden können!

Abstraktion

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.

- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren

  ⇝ erst über Algorithmen nachdenken

Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank? Antwort

⇝ "look into the box"

top down funktioniert nur, wenn wir überblicken können,
dass und wie Schritte implementiert werden können!

Abstraktion

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.

- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren

  ⇝ erst über Algorithmen nachdenken

Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank? Antwort

⇝ “look into the box”

top down funktioniert nur, wenn wir überblicken können,
dass und wie Schritte implementiert werden können!

Abstraktion

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.

- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren
  - ⇝ erst über Algorithmen nachdenken

Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank? Antwort

⇝ "look into the box"

top down funktioniert nur, wenn wir überblicken können, dass und wie Schritte implementiert werden können!

Pseudocode

- für Informatiker lesbar und mit etwas Aufwand in (Java/…)-Code übersetzbar
- keine strenge Syntax, aber eindeutig

⇝ nicht direkt verwendbar für Computer, also *pseudo* code

Abstraktion

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.

- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren

  ⇝ erst über Algorithmen nachdenken

Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank? Antwort

⇝ "look into the box"

top down funktioniert nur, wenn wir überblicken können, dass und wie Schritte implementiert werden können!

Pseudocode

- für Informatiker lesbar und mit etwas Aufwand in (Java/…)-Code übersetzbar
- keine strenge Syntax, aber eindeutig

⇝ nicht direkt verwendbar für Computer, also *pseudo* code

Abstraktion

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.
- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren

  ⇝ erst über Algorithmen nachdenken

Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank? Antwort

⇝ "look into the box"

top down funktioniert nur, wenn wir überblicken können, dass und wie Schritte implementiert werden können!

Pseudocode

"Überlebt Java."

- für Informatiker lesbar und mit etwas Aufwand in (Java/…)-Code übersetzbar
- keine strenge Syntax, aber eindeutig

⇝ nicht direkt verwendbar für Computer, also *pseudo* code

Abstraktion

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.

- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren

  ⇝ erst über Algorithmen nachdenken

Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank? Antwort

⇝ "look into the box"

top down funktioniert nur, wenn wir überblicken können, dass und wie Schritte implementiert werden können!

Pseudocode

"Überlebt Java."

- für Informatiker lesbar und mit etwas Aufwand in (Java/…)-Code übersetzbar
- keine strenge Syntax, aber eindeutig

⇝ nicht direkt verwendbar für Computer, also *pseudo* code

Abstraktion

- Algorithmen und Datenstrukturen sind Abstraktionen von Programmen.

- typische Vorgehensweise beim Programmieren: **top down**
  - erst grobe Schritte, dann diese präzisieren

  ⇝ erst über Algorithmen nachdenken

Gefahren der Abstraktion

Wie bekommt man eine Giraffe in einen Kühlschrank? Antwort

⇝ "look into the box"

top down funktioniert nur, wenn wir überblicken können, dass und wie Schritte implementiert werden können!

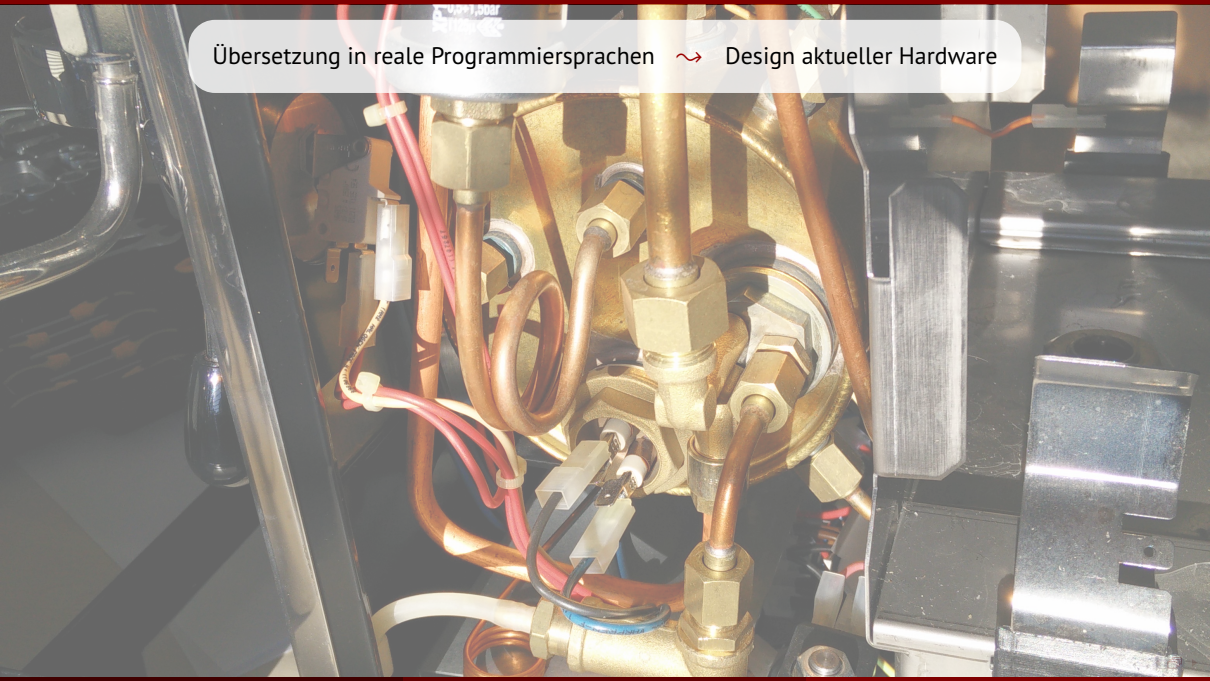
Pseudocode

"Überlebt Java."

- für Informatiker lesbar und mit etwas Aufwand in (Java/…)-Code übersetzbar
- keine strenge Syntax, aber eindeutig

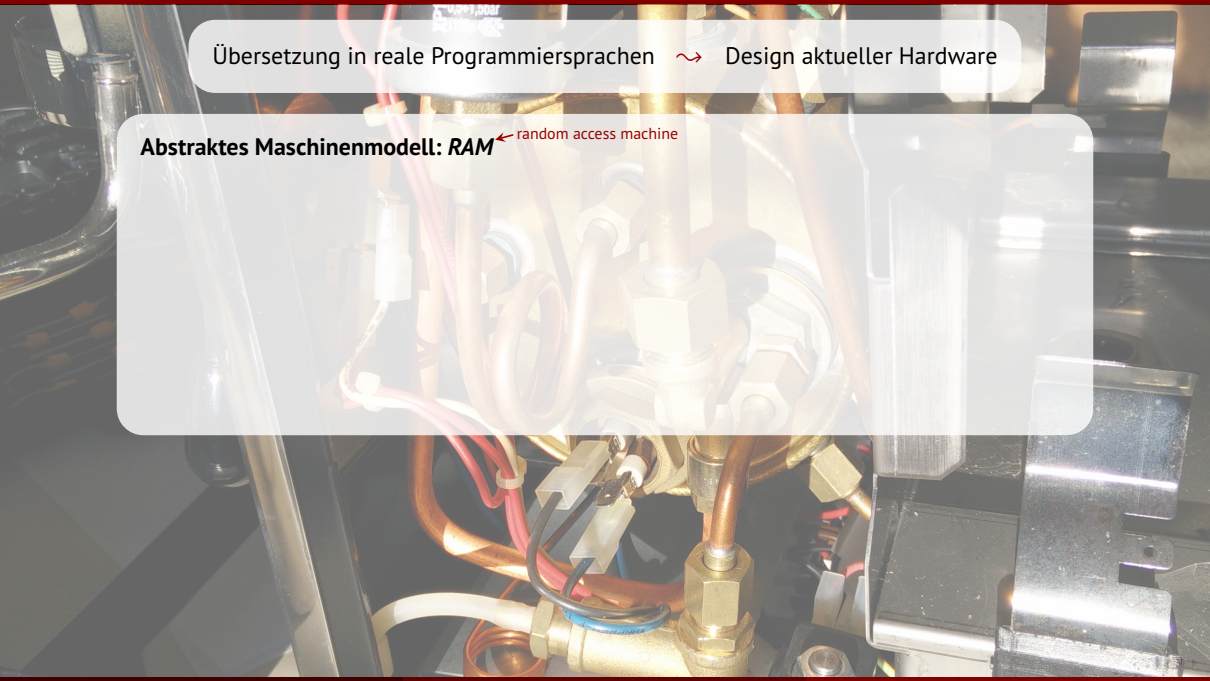
⇝ nicht direkt verwendbar für Computer, also *pseudo* code

Random Access Machines



Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

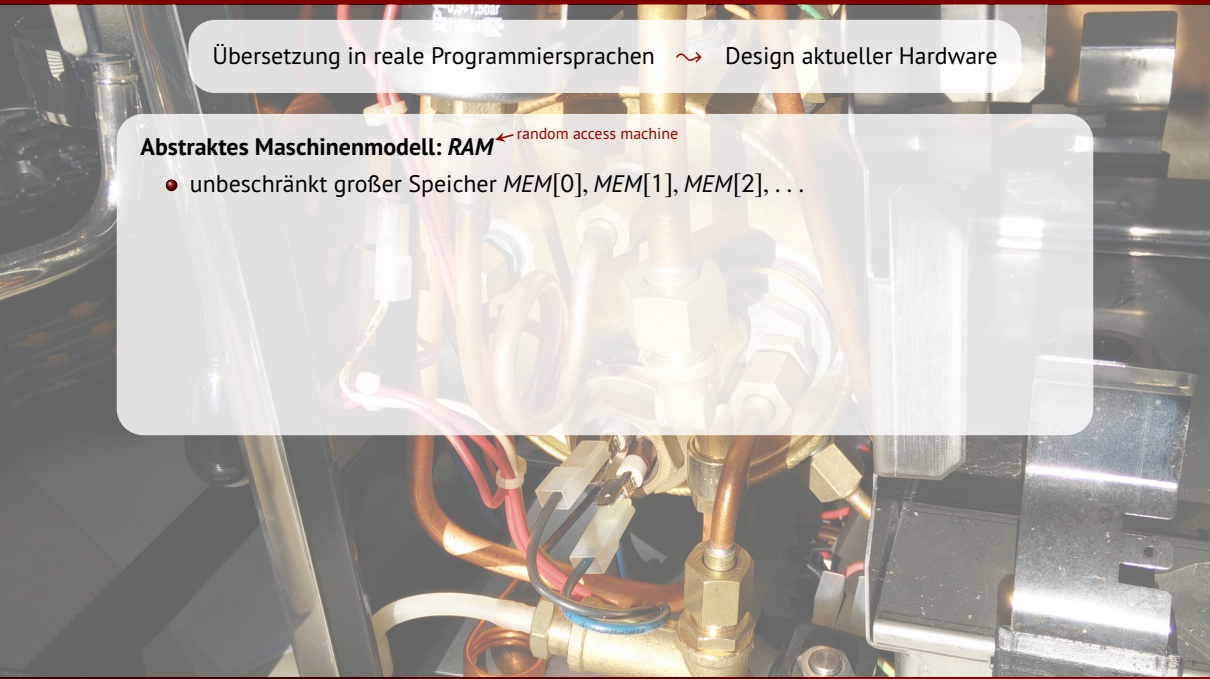
Random Access Machines



Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

Abstraktes Maschinenmodell: *RAM* ← random access machine

Random Access Machines



Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

Abstraktes Maschinenmodell: *RAM* ← random access machine

- unbeschränkt großer Speicher $MEM[0], MEM[1], MEM[2], \ldots$

Random Access Machines

Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

Abstraktes Maschinenmodell: *RAM* ← random access machine

- unbeschränkt großer Speicher $MEM[0], MEM[1], MEM[2], \ldots$
- jede Zelle $MEM[i]$ speichert eine w-Bit Zahl, also ganze Zahl aus $[0..2^w - 1]$
  w ist die Wortbreite; typischerweise $2^w \approx n$, die Eingabegröße

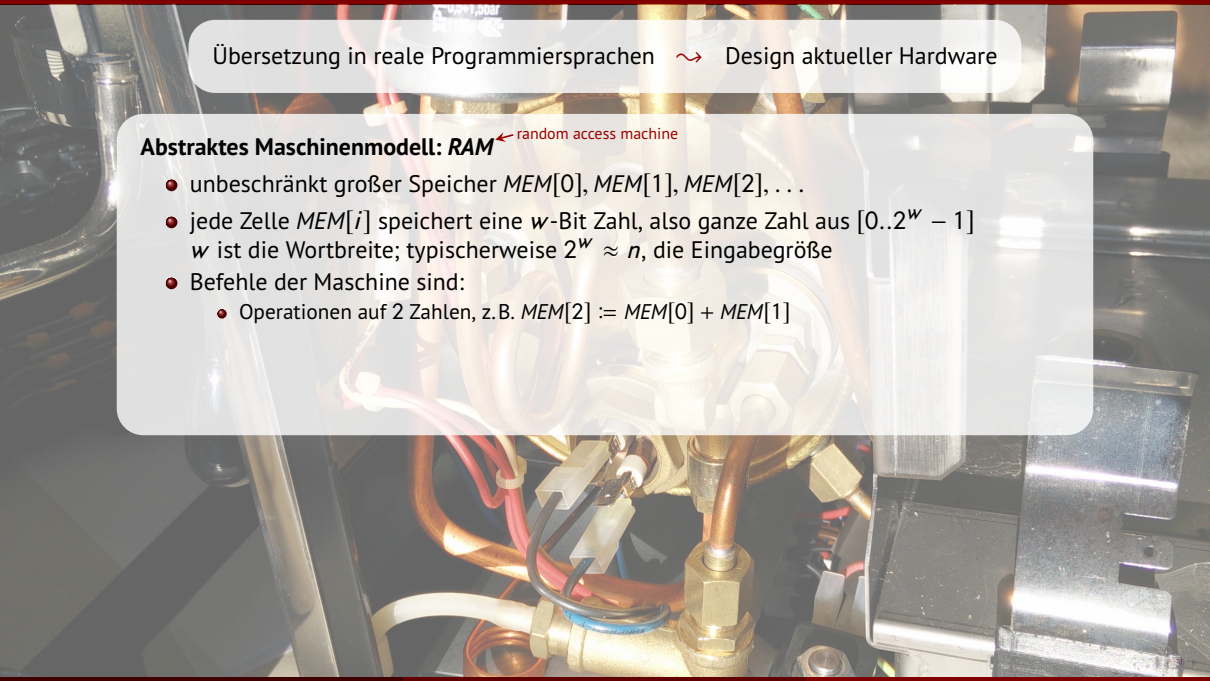
Random Access Machines

Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

Abstraktes Maschinenmodell: *RAM* ← random access machine

- unbeschränkt großer Speicher $MEM[0], MEM[1], MEM[2], \ldots$
- jede Zelle $MEM[i]$ speichert eine w-Bit Zahl, also ganze Zahl aus $[0..2^w - 1]$
  w ist die Wortbreite; typischerweise $2^w \approx n$, die Eingabegröße
- Befehle der Maschine sind:

Random Access Machines

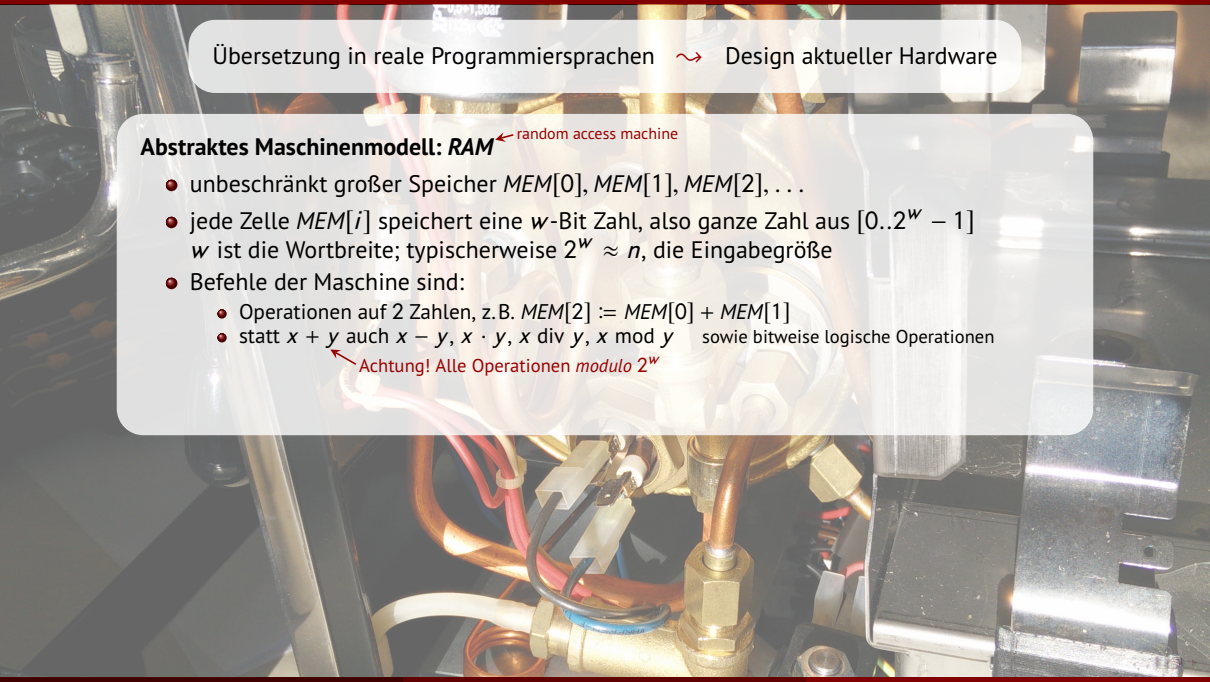


Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

Abstraktes Maschinenmodell: *RAM* ← random access machine

- unbeschränkt großer Speicher $MEM[0], MEM[1], MEM[2], \ldots$
- jede Zelle $MEM[i]$ speichert eine w-Bit Zahl, also ganze Zahl aus $[0..2^w - 1]$
  w ist die Wortbreite; typischerweise $2^w \approx n$, die Eingabegröße
- Befehle der Maschine sind:
  - Operationen auf 2 Zahlen, z. B. $MEM[2] := MEM[0] + MEM[1]$

Random Access Machines



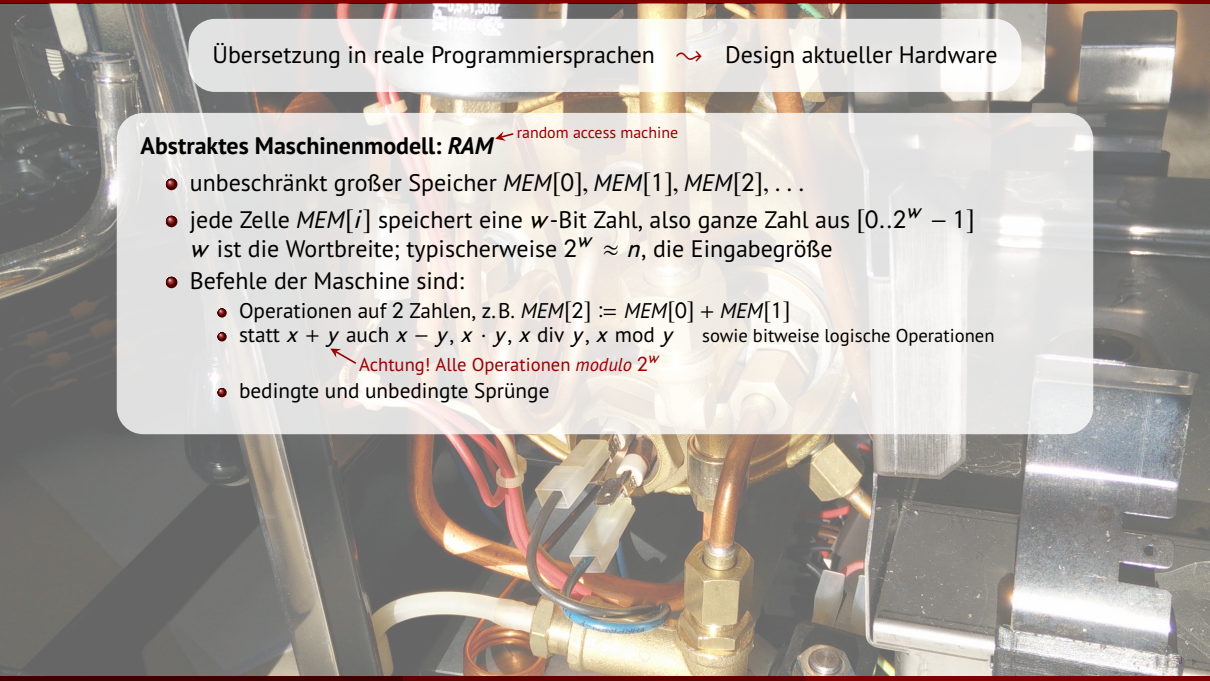
Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

Abstraktes Maschinenmodell: *RAM* ← random access machine

- unbeschränkt großer Speicher $MEM[0], MEM[1], MEM[2], \ldots$
- jede Zelle $MEM[i]$ speichert eine w-Bit Zahl, also ganze Zahl aus $[0..2^w - 1]$
  w ist die Wortbreite; typischerweise $2^w \approx n$, die Eingabegröße
- Befehle der Maschine sind:
  - Operationen auf 2 Zahlen, z.B. $MEM[2] := MEM[0] + MEM[1]$
  - statt $x + y$ auch $x - y$, $x \cdot y$, x div y, x mod y sowie bitweise logische Operationen

Achtung! Alle Operationen *modulo* 2^w

Random Access Machines



Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

Abstraktes Maschinenmodell: *RAM* ← random access machine

- unbeschränkt großer Speicher $MEM[0], MEM[1], MEM[2], \ldots$
- jede Zelle $MEM[i]$ speichert eine w-Bit Zahl, also ganze Zahl aus $[0..2^w - 1]$
  w ist die Wortbreite; typischerweise $2^w \approx n$, die Eingabegröße
- Befehle der Maschine sind:
  - Operationen auf 2 Zahlen, z.B. $MEM[2] := MEM[0] + MEM[1]$
  - statt $x + y$ auch $x - y$, $x \cdot y$, x div y, x mod y sowie bitweise logische Operationen
    ← Achtung! Alle Operationen *modulo* 2^w
  - bedingte und unbedingte Sprünge

Random Access Machines

Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

Abstraktes Maschinenmodell: *RAM* ← random access machine

- unbeschränkt großer Speicher $MEM[0], MEM[1], MEM[2], \ldots$
- jede Zelle $MEM[i]$ speichert eine w-Bit Zahl, also ganze Zahl aus $[0..2^w - 1]$
  w ist die Wortbreite; typischerweise $2^w \approx n$, die Eingabegröße
- Befehle der Maschine sind:
  - Operationen auf 2 Zahlen, z.B. $MEM[2] := MEM[0] + MEM[1]$
  - statt $x + y$ auch $x - y$, $x \cdot y$, x div y, x mod y sowie bitweise logische Operationen
    ← Achtung! Alle Operationen *modulo* 2^w
  - bedingte und unbedingte Sprünge

oben: vereinfachte Version ohne Register

Random Access Machines

Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

Abstraktes Maschinenmodell: *RAM* ← random access machine

- unbeschränkt großer Speicher $MEM[0], MEM[1], MEM[2], \ldots$
- jede Zelle $MEM[i]$ speichert eine w-Bit Zahl, also ganze Zahl aus $[0..2^w - 1]$
  w ist die Wortbreite; typischerweise $2^w \approx n$, die Eingabegröße
- Befehle der Maschine sind:
  - Operationen auf 2 Zahlen, z.B. $MEM[2] := MEM[0] + MEM[1]$
  - statt $x + y$ auch $x - y$, $x \cdot y$, x div y, x mod y sowie bitweise logische Operationen
    ← Achtung! Alle Operationen *modulo* 2^w
  - bedingte und unbedingte Sprünge

oben: vereinfachte Version ohne Register

alle Details: Kapitel 2.2 aus
Dietzfelbinger, Mehlhorn, Sanders: *Algorithmen und Datenstrukturen*

Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

Abstraktes Maschinenmodell: *RAM* ← random access machine

- unbeschränkt großer Speicher $MEM[0], MEM[1], MEM[2], \ldots$
- jede Zelle $MEM[i]$ speichert eine w-Bit Zahl, also ganze Zahl aus $[0..2^w - 1]$
  w ist die Wortbreite; typischerweise $2^w \approx n$, die Eingabegröße
- Befehle der Maschine sind:

- RAM = voller Detailgrad …
  So will man nicht programmieren müssen!
- Meistens reicht uns eine abstraktere Sichtweise.
- **Aber:** Wenn nötig, können wir Details im präzisen Modell klären!

$\rightsquigarrow$ große Errungenschaft der theoretischen Informatik,
da wir mathematische Beweise führen können.

eternal truths

oben: v
alle De

Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

Abstraktes Maschinenmodell: *RAM* ← random access machine

- unbeschränkt großer Speicher $MEM[0], MEM[1], MEM[2], \ldots$
- jede Zelle $MEM[i]$ speichert eine w-Bit Zahl, also ganze Zahl aus $[0..2^w - 1]$
  w ist die Wortbreite; typischerweise $2^w \approx n$, die Eingabegröße
- Befehle der Maschine sind:

- RAM = voller Detailgrad …
  So will man nicht programmieren müssen!
- Meistens reicht uns eine abstraktere Sichtweise.
- **Aber:** Wenn nötig, können wir Details im präzisen Modell klären!

$\rightsquigarrow$ große Errungenschaft der theoretischen Informatik,
da wir mathematische Beweise führen können.

eternal truths

oben: v
alle De

Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

Abstraktes Maschinenmodell: *RAM* ← random access machine

- unbeschränkt großer Speicher $MEM[0], MEM[1], MEM[2], \ldots$
- jede Zelle $MEM[i]$ speichert eine w-Bit Zahl, also ganze Zahl aus $[0..2^w - 1]$
  w ist die Wortbreite; typischerweise $2^w \approx n$, die Eingabegröße
- Befehle der Maschine sind:

- RAM = voller Detailgrad …
  So will man nicht programmieren müssen!
- Meistens reicht uns eine abstraktere Sichtweise.

- **Aber:** Wenn nötig, können wir Details im präzisen Modell klären!

$\rightsquigarrow$ große Errungenschaft der theoretischen Informatik,
da wir mathematische Beweise führen können.

eternal truths

oben: v
alle De

Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

Abstraktes Maschinenmodell: *RAM* ← random access machine

- unbeschränkt großer Speicher $MEM[0], MEM[1], MEM[2], \ldots$
- jede Zelle $MEM[i]$ speichert eine w-Bit Zahl, also ganze Zahl aus $[0..2^w - 1]$
  w ist die Wortbreite; typischerweise $2^w \approx n$, die Eingabegröße
- Befehle der Maschine sind:

- RAM = voller Detailgrad …
  So will man nicht programmieren müssen!
- Meistens reicht uns eine abstraktere Sichtweise.
- **Aber:** Wenn nötig, können wir Details im präzisen Modell klären!

$\rightsquigarrow$ große Errungenschaft der theoretischen Informatik,
da wir mathematische Beweise führen können.

eternal truths

oben: v
alle De

Übersetzung in reale Programmiersprachen $\rightsquigarrow$ Design aktueller Hardware

Abstraktes Maschinenmodell: *RAM* ← random access machine

- unbeschränkt großer Speicher $MEM[0], MEM[1], MEM[2], \ldots$
- jede Zelle $MEM[i]$ speichert eine w-Bit Zahl, also ganze Zahl aus $[0..2^w - 1]$
  w ist die Wortbreite; typischerweise $2^w \approx n$, die Eingabegröße
- Befehle der Maschine sind:

oben: v
alle De

- RAM = voller Detailgrad …
  So will man nicht programmieren müssen!
- Meistens reicht uns eine abstraktere Sichtweise.

- **Aber:** Wenn nötig, können wir Details im präzisen Modell klären!

$\rightsquigarrow$ große Errungenschaft der theoretischen Informatik,
da wir mathematische Beweise führen können.
↖ eternal truths