Algorithmen & Datenstrukturen

Sebastian Wild
wild@cs.uni-kl.de

Fachbereich Informatik

10. Januar 2017

3 Analyse von Algorithmen

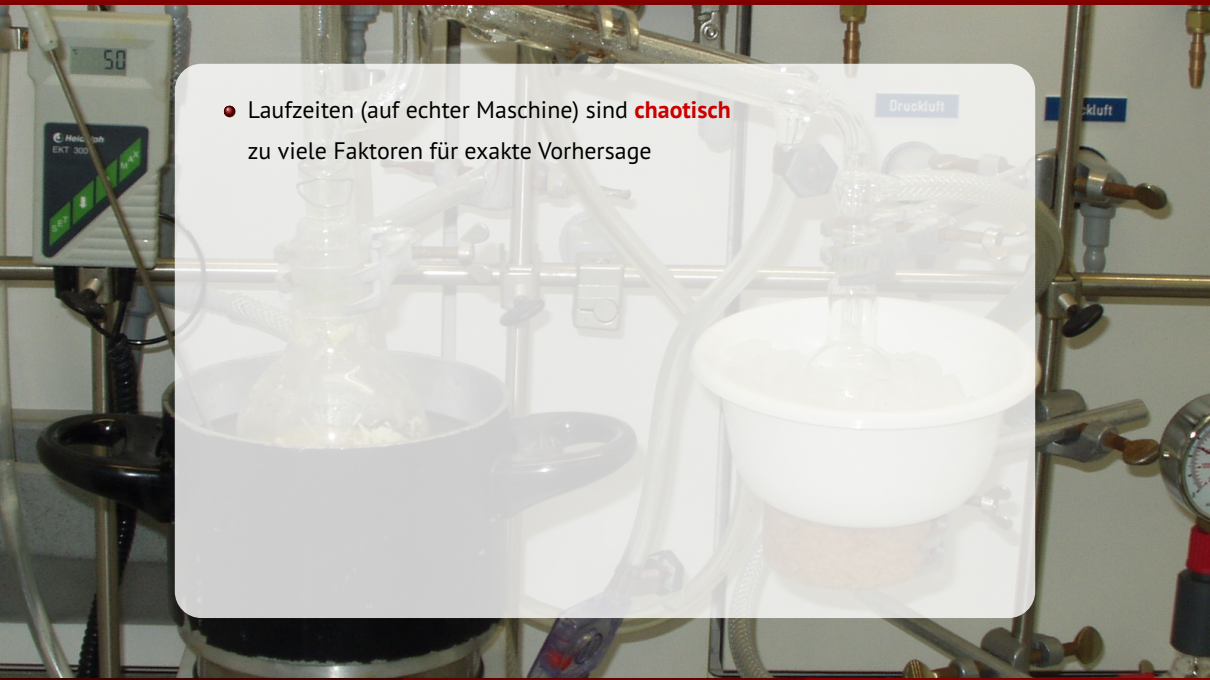
3 Analyse von Algorithmen

Bemerkungen und Ergänzungen*

*eigentliche Inhalte in Videos

Laufzeit vs. Modelle

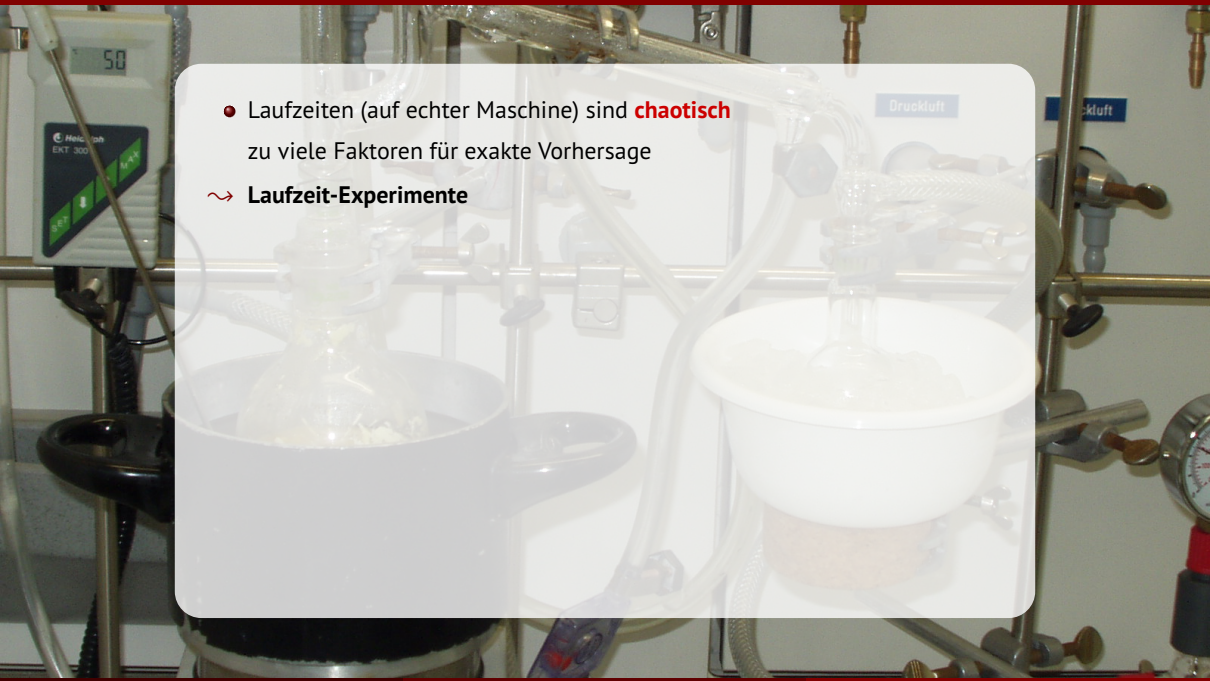
Laufzeit vs. Modelle



- Laufzeiten (auf echter Maschine) sind **chaotisch**

  zu viele Faktoren für exakte Vorhersage

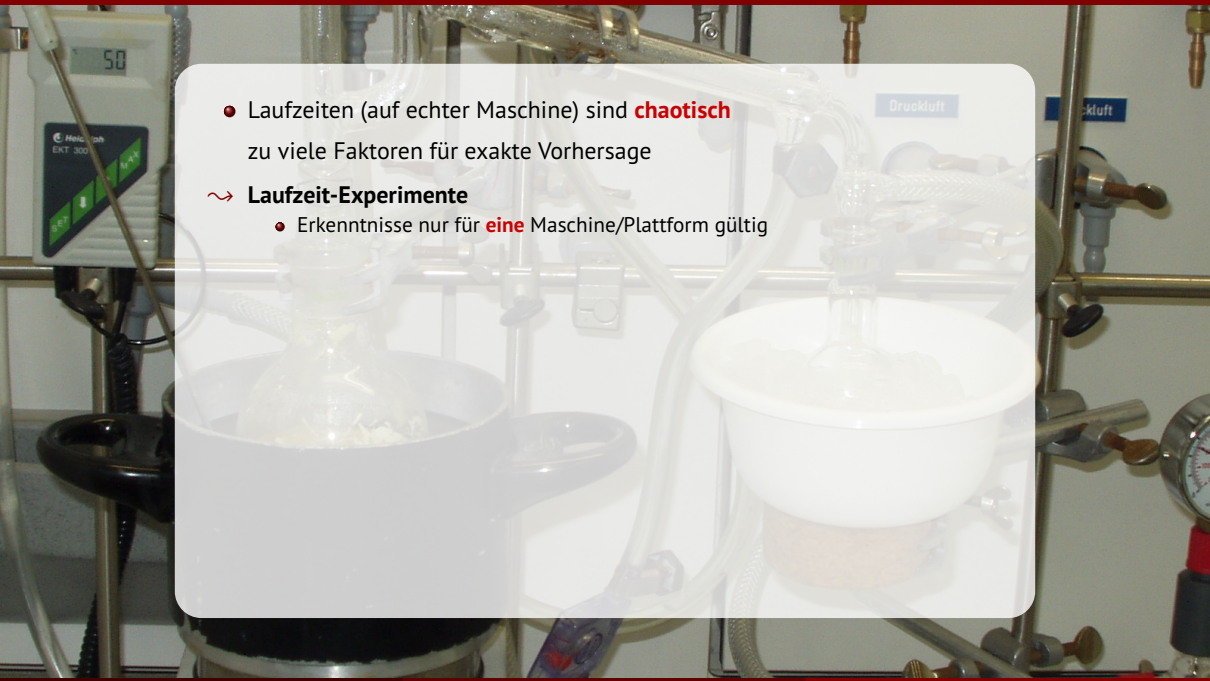
Laufzeit vs. Modelle



- Laufzeiten (auf echter Maschine) sind **chaotisch**
  zu viele Faktoren für exakte Vorhersage

⇝ **Laufzeit-Experimente**

Laufzeit vs. Modelle



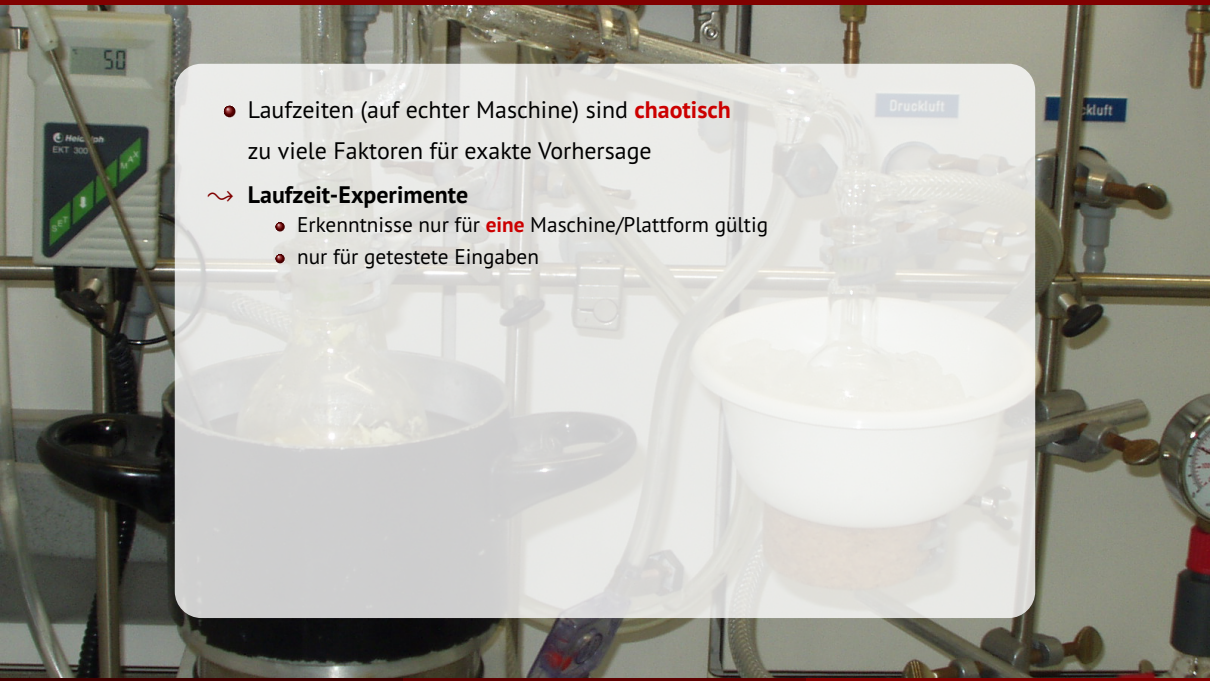
- Laufzeiten (auf echter Maschine) sind **chaotisch**

  zu viele Faktoren für exakte Vorhersage

⇝ **Laufzeit-Experimente**

- Erkenntnisse nur für **eine** Maschine/Plattform gültig

Laufzeit vs. Modelle



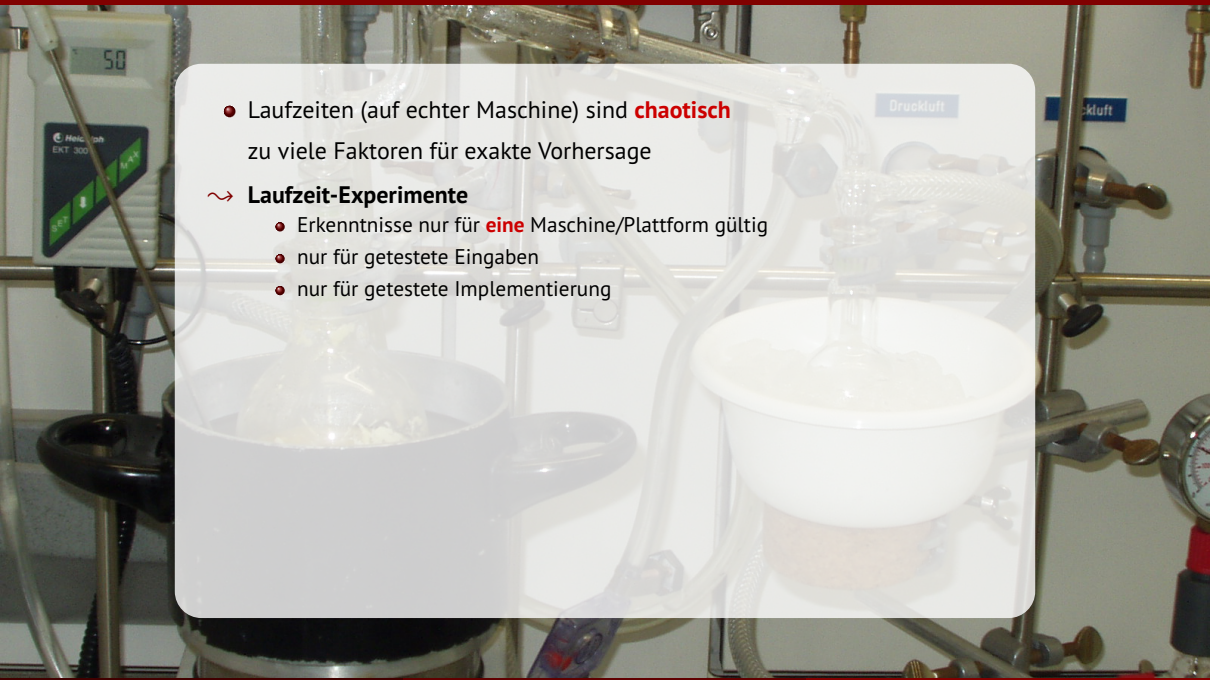
- Laufzeiten (auf echter Maschine) sind **chaotisch**

  zu viele Faktoren für exakte Vorhersage

⇝ **Laufzeit-Experimente**

- Erkenntnisse nur für **eine** Maschine/Plattform gültig
- nur für getestete Eingaben

Laufzeit vs. Modelle



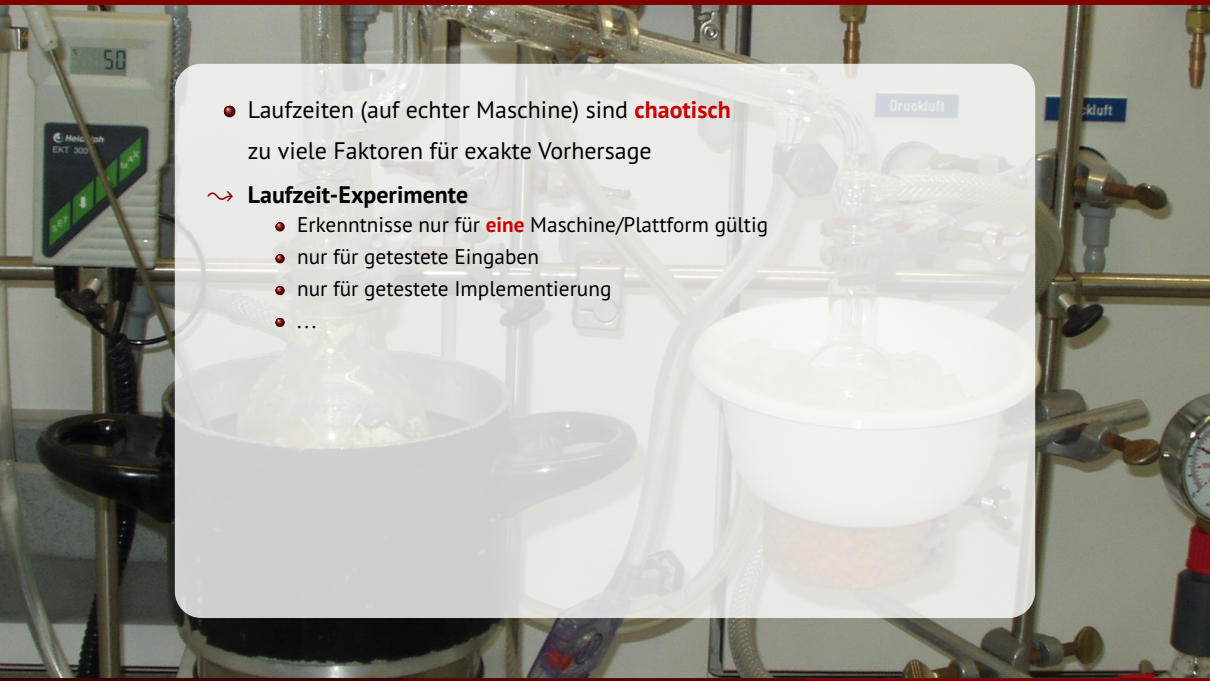
- Laufzeiten (auf echter Maschine) sind **chaotisch**

  zu viele Faktoren für exakte Vorhersage

⇝ **Laufzeit-Experimente**

- Erkenntnisse nur für **eine** Maschine/Plattform gültig
- nur für getestete Eingaben
- nur für getestete Implementierung

Laufzeit vs. Modelle



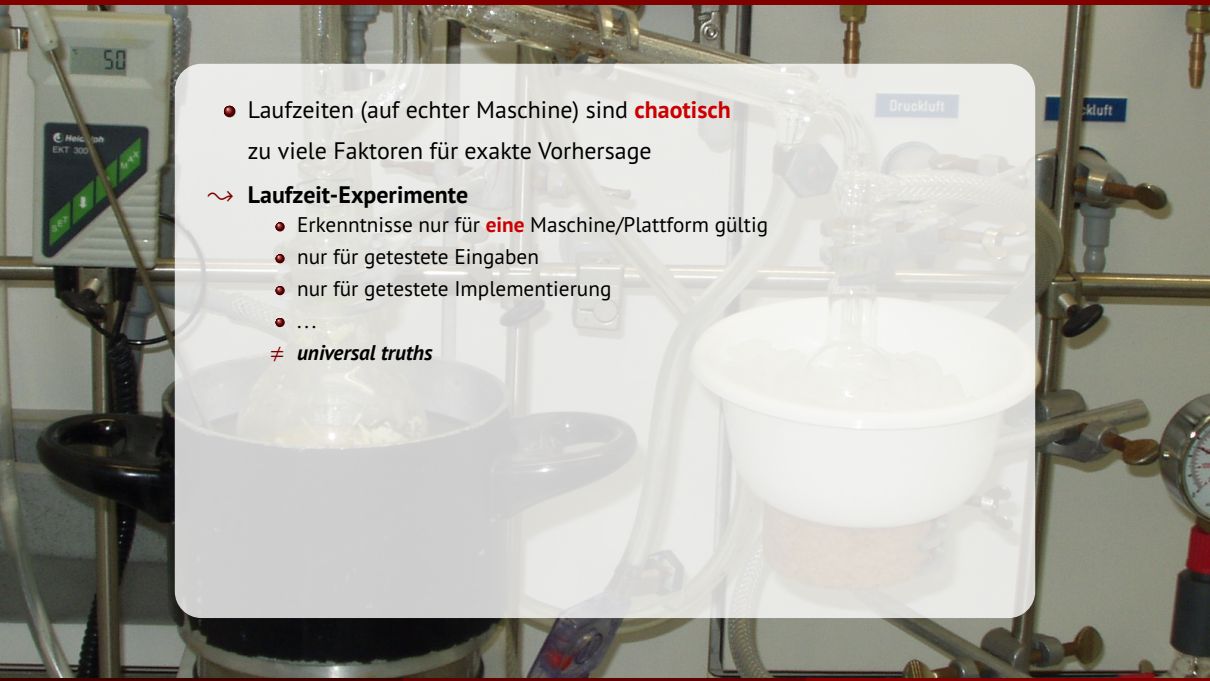
- Laufzeiten (auf echter Maschine) sind **chaotisch**

  zu viele Faktoren für exakte Vorhersage

⇝ **Laufzeit-Experimente**

- Erkenntnisse nur für **eine** Maschine/Plattform gültig
- nur für getestete Eingaben
- nur für getestete Implementierung
- …

Laufzeit vs. Modelle



- Laufzeiten (auf echter Maschine) sind **chaotisch**

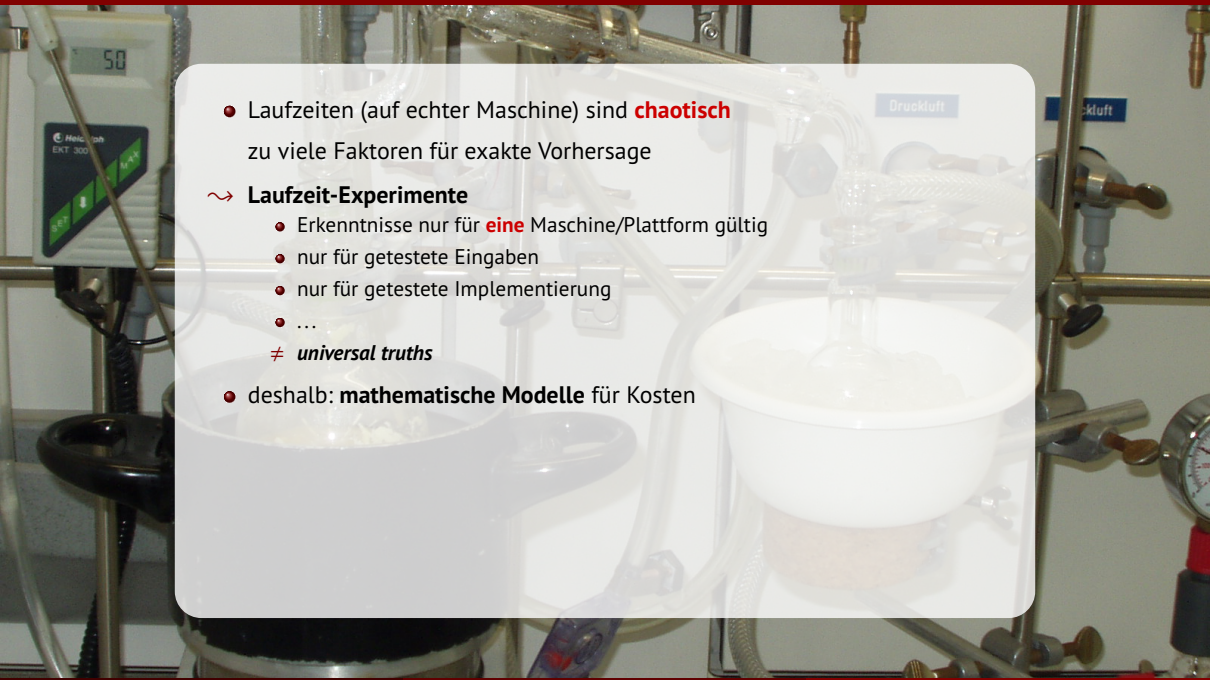
  zu viele Faktoren für exakte Vorhersage

⇝ **Laufzeit-Experimente**

- Erkenntnisse nur für **eine** Maschine/Plattform gültig
- nur für getestete Eingaben
- nur für getestete Implementierung
- …

≠ ***universal truths***

Laufzeit vs. Modelle



- Laufzeiten (auf echter Maschine) sind **chaotisch**

  zu viele Faktoren für exakte Vorhersage

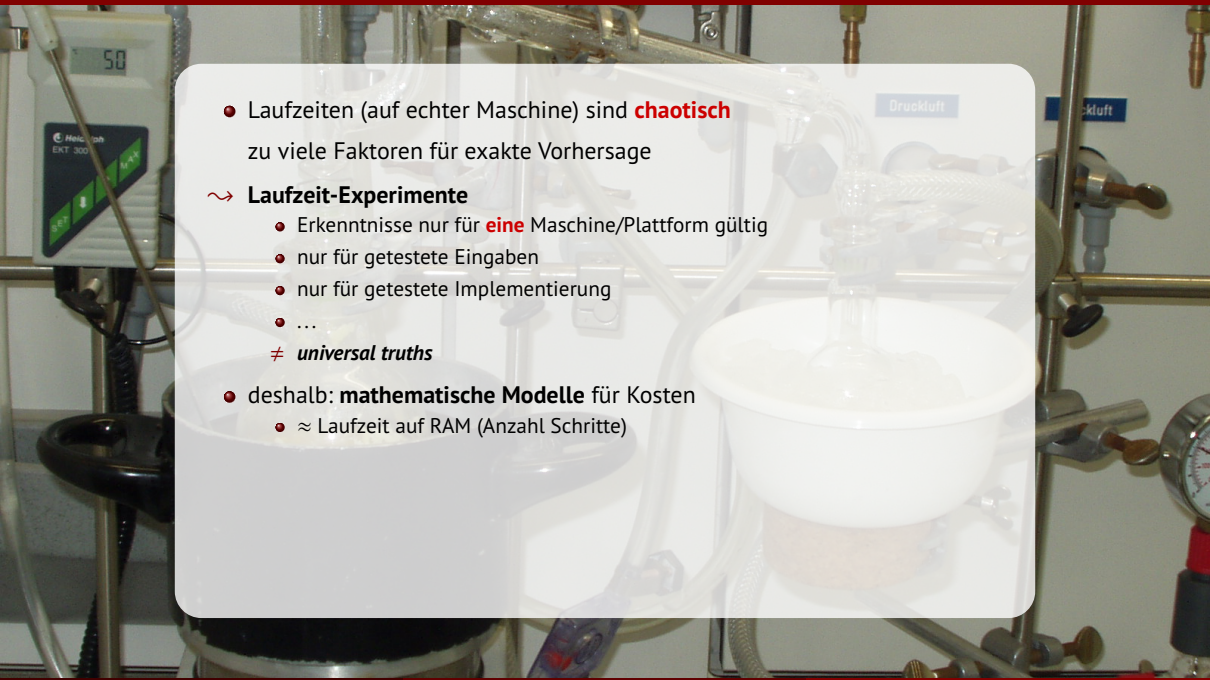
↝ **Laufzeit-Experimente**

- Erkenntnisse nur für **eine** Maschine/Plattform gültig
- nur für getestete Eingaben
- nur für getestete Implementierung
- …

≠ ***universal truths***

- deshalb: **mathematische Modelle** für Kosten

Laufzeit vs. Modelle



- Laufzeiten (auf echter Maschine) sind **chaotisch**

  zu viele Faktoren für exakte Vorhersage

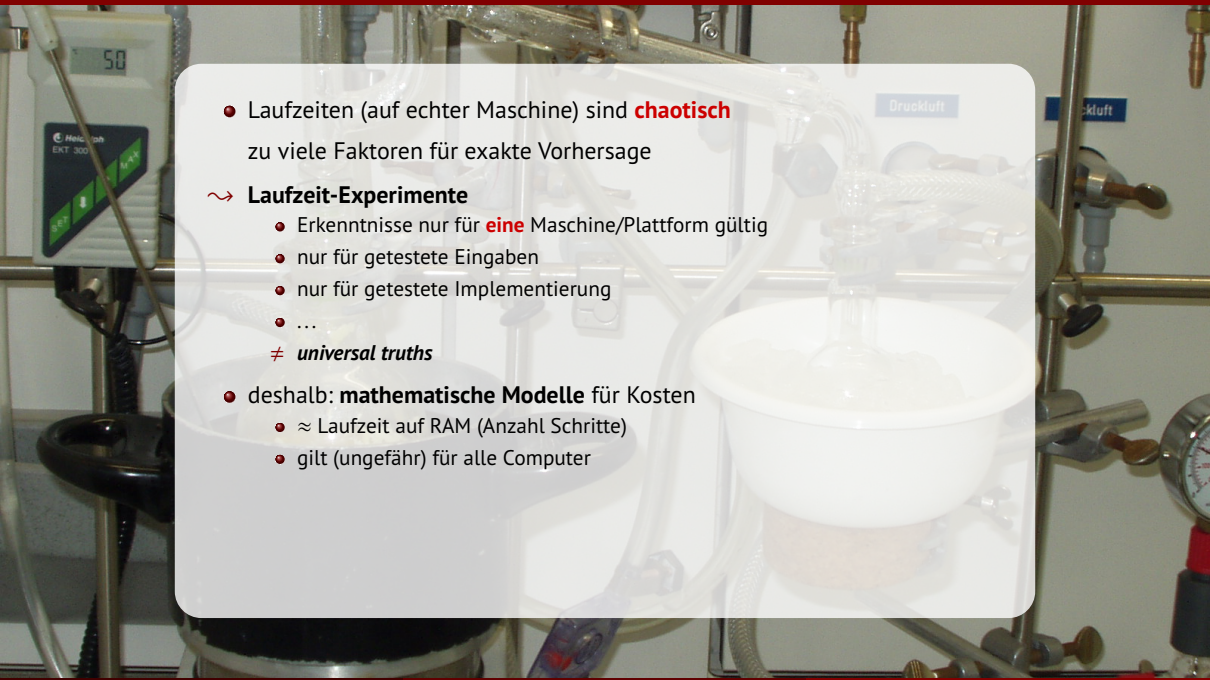
↝ **Laufzeit-Experimente**

- Erkenntnisse nur für **eine** Maschine/Plattform gültig
- nur für getestete Eingaben
- nur für getestete Implementierung
- …

≠ ***universal truths***

- deshalb: **mathematische Modelle** für Kosten
  - ≈ Laufzeit auf RAM (Anzahl Schritte)

Laufzeit vs. Modelle



- Laufzeiten (auf echter Maschine) sind **chaotisch**

  zu viele Faktoren für exakte Vorhersage

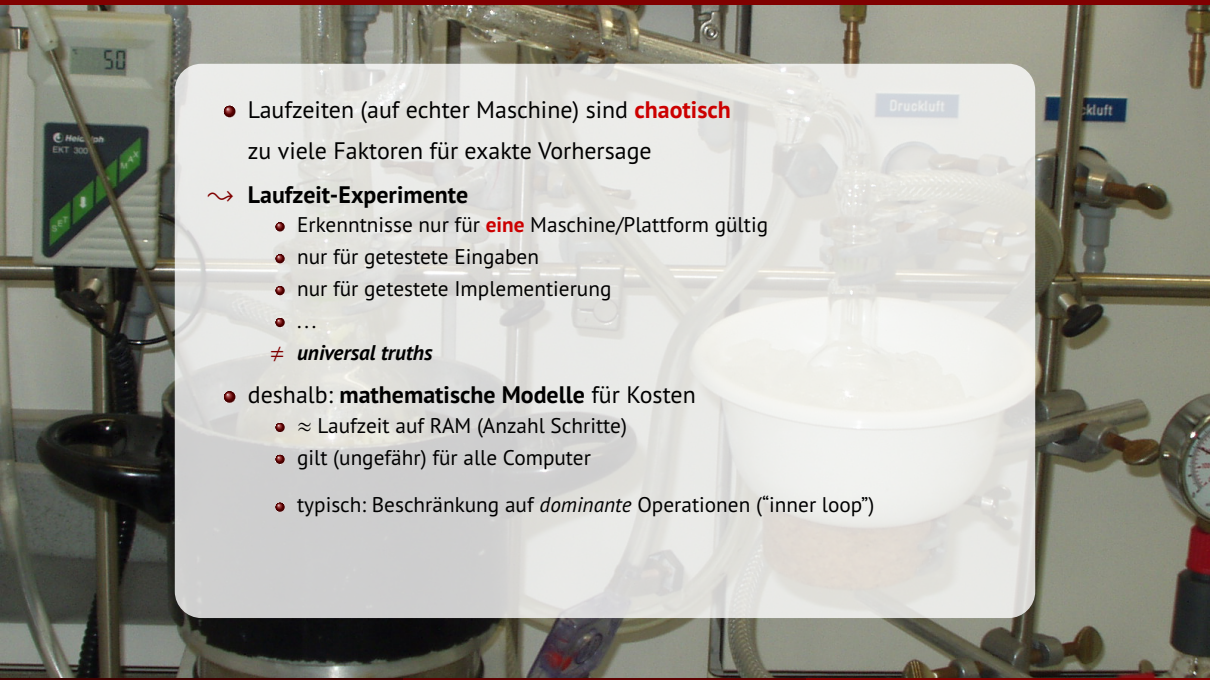
↝ **Laufzeit-Experimente**

- Erkenntnisse nur für **eine** Maschine/Plattform gültig
- nur für getestete Eingaben
- nur für getestete Implementierung
- …

≠ ***universal truths***

- deshalb: **mathematische Modelle** für Kosten
  - ≈ Laufzeit auf RAM (Anzahl Schritte)
  - gilt (ungefähr) für alle Computer

Laufzeit vs. Modelle



- Laufzeiten (auf echter Maschine) sind **chaotisch**

  zu viele Faktoren für exakte Vorhersage

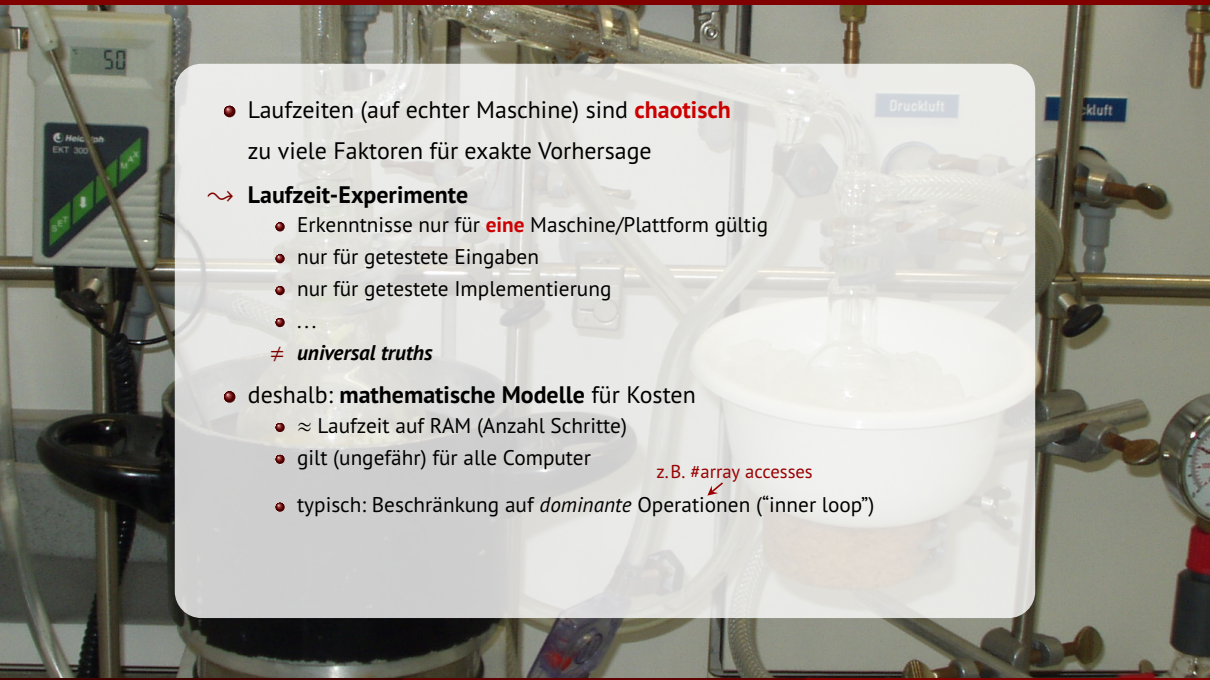
↝ **Laufzeit-Experimente**

- Erkenntnisse nur für **eine** Maschine/Plattform gültig
- nur für getestete Eingaben
- nur für getestete Implementierung
- …

≠ ***universal truths***

- deshalb: **mathematische Modelle** für Kosten
  - ≈ Laufzeit auf RAM (Anzahl Schritte)
  - gilt (ungefähr) für alle Computer
  - typisch: Beschränkung auf *dominante* Operationen (“inner loop”)

Laufzeit vs. Modelle



- Laufzeiten (auf echter Maschine) sind **chaotisch**

  zu viele Faktoren für exakte Vorhersage

↝ **Laufzeit-Experimente**

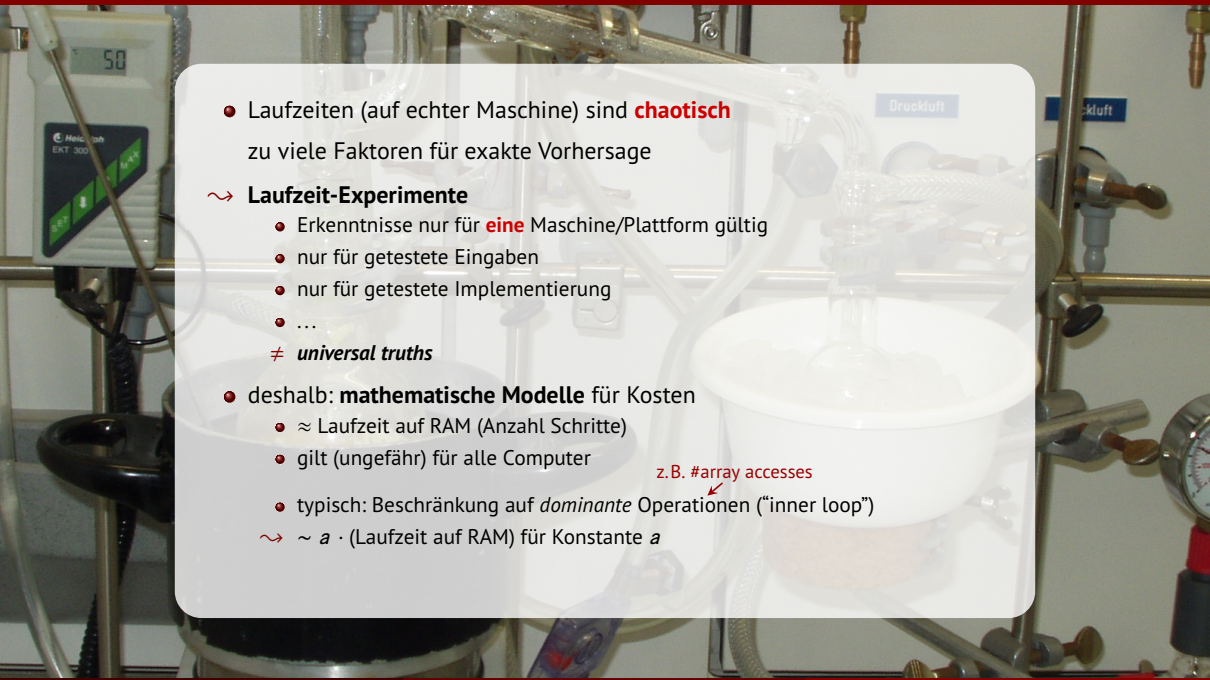
- Erkenntnisse nur für **eine** Maschine/Plattform gültig
- nur für getestete Eingaben
- nur für getestete Implementierung
- …

≠ ***universal truths***

- deshalb: **mathematische Modelle** für Kosten
  - ≈ Laufzeit auf RAM (Anzahl Schritte)
  - gilt (ungefähr) für alle Computer
  - typisch: Beschränkung auf *dominante* Operationen (“inner loop”)

z.B. #array accesses

Laufzeit vs. Modelle



- Laufzeiten (auf echter Maschine) sind **chaotisch**

  zu viele Faktoren für exakte Vorhersage

↝ **Laufzeit-Experimente**

- Erkenntnisse nur für **eine** Maschine/Plattform gültig
- nur für getestete Eingaben
- nur für getestete Implementierung
- …

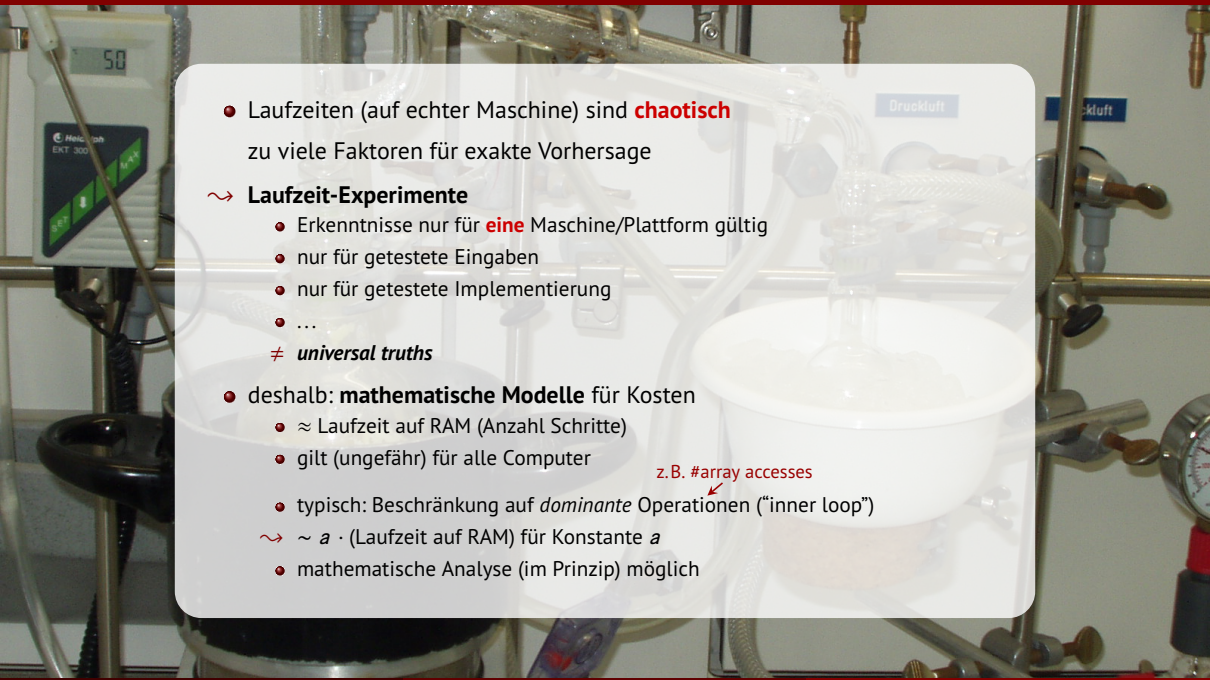
≠ ***universal truths***

- deshalb: **mathematische Modelle** für Kosten
  - ≈ Laufzeit auf RAM (Anzahl Schritte)
  - gilt (ungefähr) für alle Computer
  - typisch: Beschränkung auf *dominante* Operationen (“inner loop”)

    z.B. #array accesses

  ↝ $\sim a \cdot$ (Laufzeit auf RAM) für Konstante a

Laufzeit vs. Modelle



- Laufzeiten (auf echter Maschine) sind **chaotisch**

  zu viele Faktoren für exakte Vorhersage

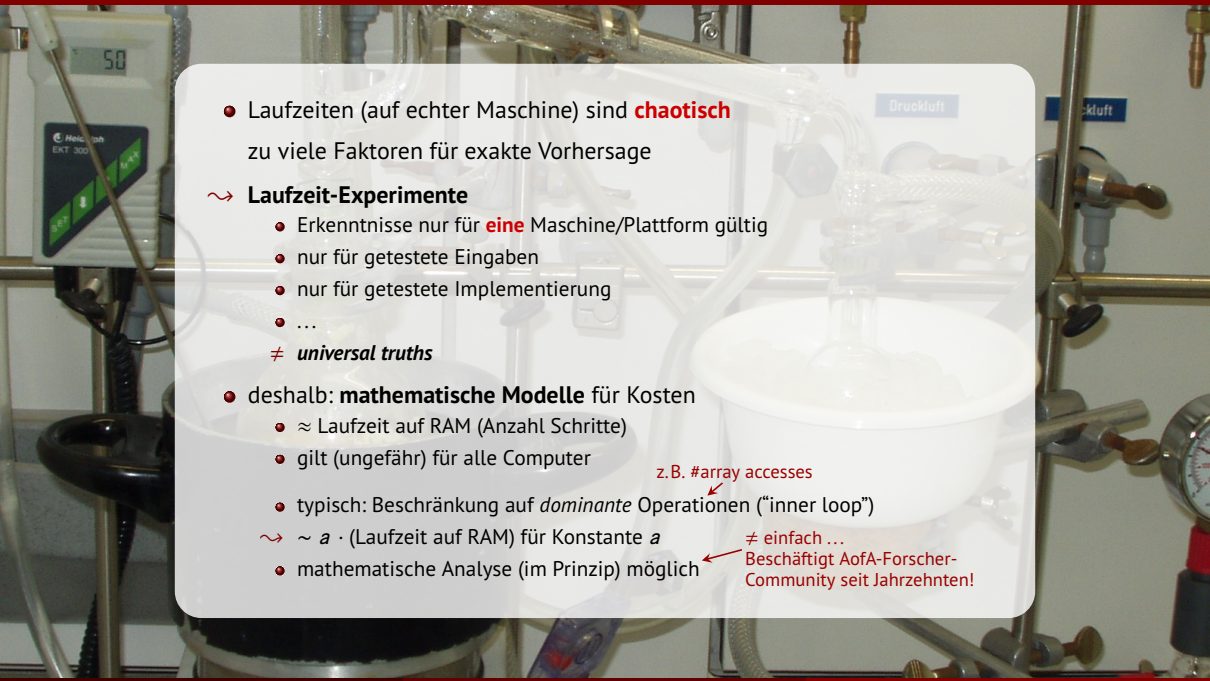
↝ **Laufzeit-Experimente**

  - Erkenntnisse nur für **eine** Maschine/Plattform gültig
  - nur für getestete Eingaben
  - nur für getestete Implementierung
  - …
  - $\neq$ ***universal truths***

- deshalb: **mathematische Modelle** für Kosten
  - $\approx$ Laufzeit auf RAM (Anzahl Schritte)
  - gilt (ungefähr) für alle Computer
  - typisch: Beschränkung auf *dominante* Operationen (“inner loop”)

    z.B. #array accesses
  - ↝ $\sim a \cdot$ (Laufzeit auf RAM) für Konstante a
  - mathematische Analyse (im Prinzip) möglich

Laufzeit vs. Modelle



- Laufzeiten (auf echter Maschine) sind **chaotisch**

  zu viele Faktoren für exakte Vorhersage

↝ **Laufzeit-Experimente**

  - Erkenntnisse nur für **eine** Maschine/Plattform gültig
  - nur für getestete Eingaben
  - nur für getestete Implementierung
  - …
  - ≠ ***universal truths***

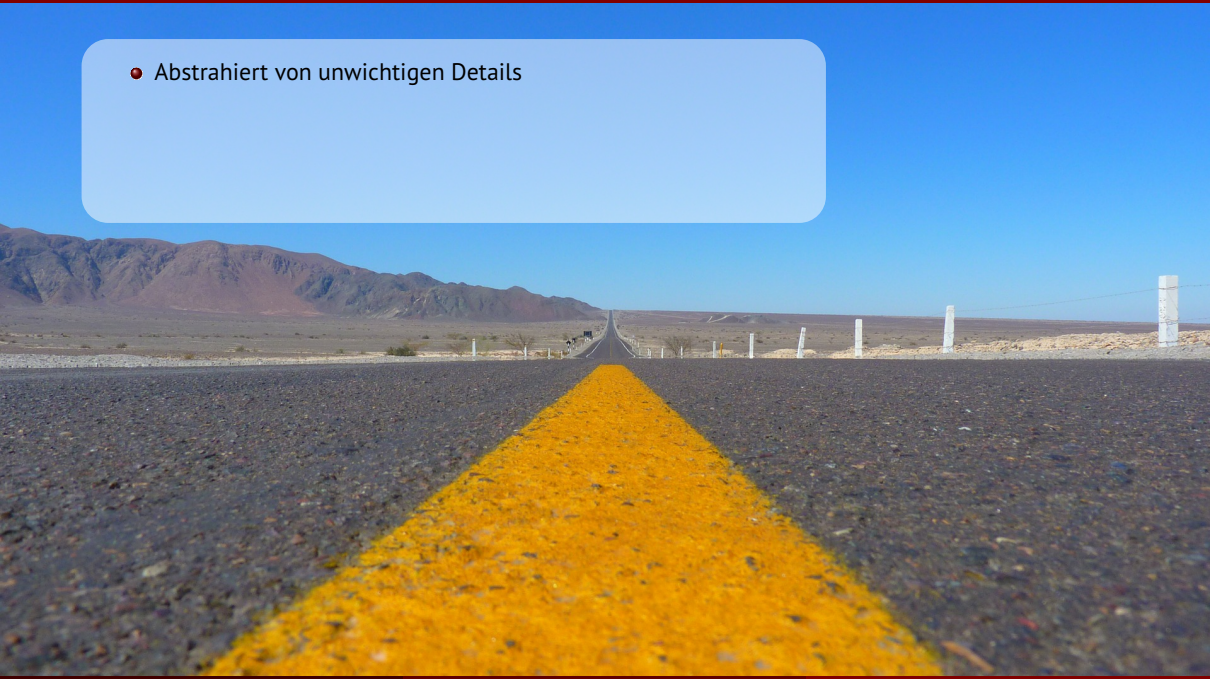
- deshalb: **mathematische Modelle** für Kosten
  - ≈ Laufzeit auf RAM (Anzahl Schritte)
  - gilt (ungefähr) für alle Computer
  - typisch: Beschränkung auf *dominante* Operationen (“inner loop”)

    z.B. #array accesses
  - ↝ $\sim a \cdot$ (Laufzeit auf RAM) für Konstante a
  - mathematische Analyse (im Prinzip) möglich

    ≠ einfach …
    Beschäftigt AofA-Forscher-Community seit Jahrzehnten!

Warum Asymptotik?

- Abstrahiert von unwichtigen Details



Warum Asymptotik?

- Abstrahiert von unwichtigen Details

  für Gesamtlaufzeit

Warum Asymptotik?

- Abstrahiert von unwichtigen Details

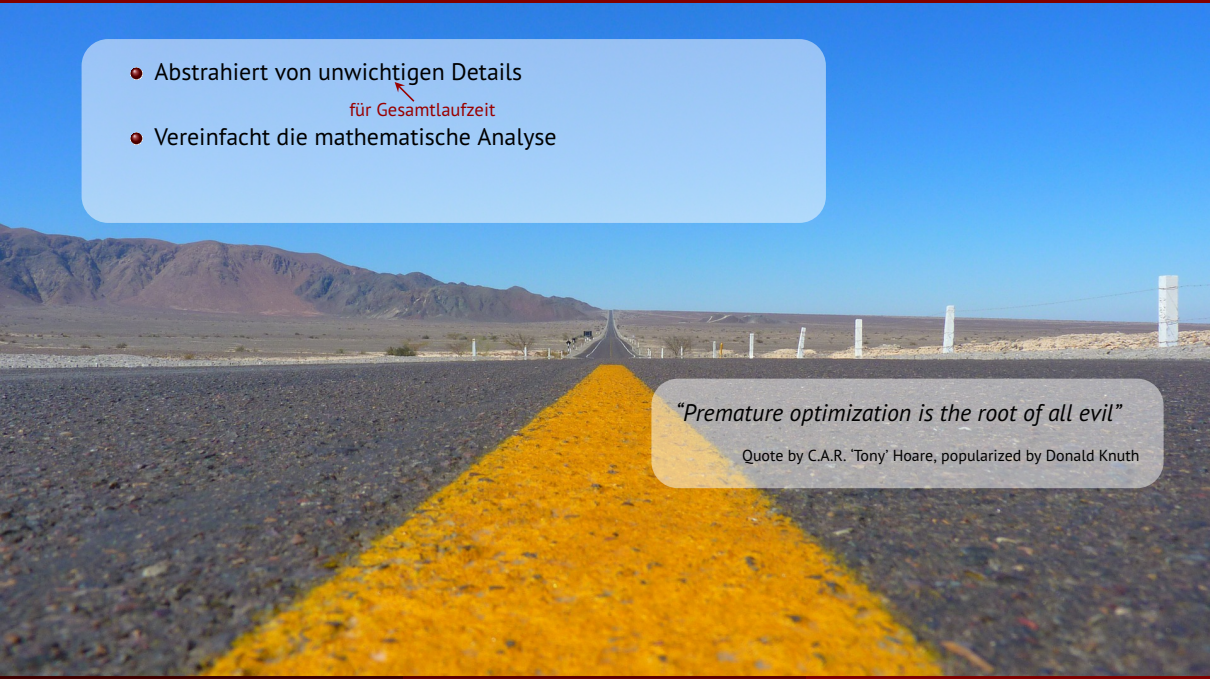
  für Gesamtlaufzeit

“Premature optimization is the root of all evil”

Quote by C.A.R. ‘Tony’ Hoare, popularized by Donald Knuth

Warum Asymptotik?

- Abstrahiert von unwichtigen Details
  für Gesamtlaufzeit
- Vereinfacht die mathematische Analyse



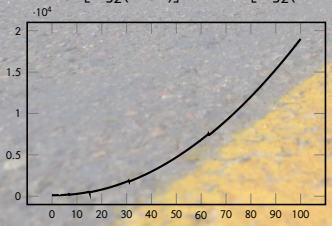
“Premature optimization is the root of all evil”

Quote by C.A.R. ‘Tony’ Hoare, popularized by Donald Knuth

Warum Asymptotik?

- Abstrahiert von unwichtigen Details
  für Gesamtlaufzeit
- Vereinfacht die mathematische Analyse

$$2n^2 - 3n\lfloor \log_2(n+1) \rfloor + 7n - 3\lfloor \log_2(n+1) \rfloor + 120$$



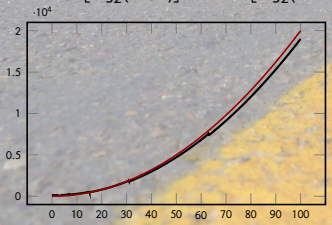
"Premature optimization is the root of all evil"

Quote by C.A.R. 'Tony' Hoare, popularized by Donald Knuth

Warum Asymptotik?

- Abstrahiert von unwichtigen Details
  für Gesamtlaufzeit
- Vereinfacht die mathematische Analyse

$2n^2 \quad \sim \quad 2n^2 - 3n\lfloor \log_2(n+1) \rfloor + 7n - 3\lfloor \log_2(n+1) \rfloor + 120$



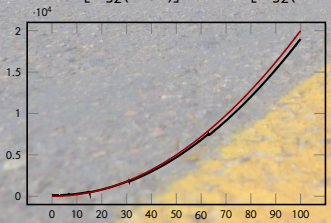
"Premature optimization is the root of all evil"

Quote by C.A.R. 'Tony' Hoare, popularized by Donald Knuth

Warum Asymptotik?

- Abstrahiert von unwichtigen Details
  für Gesamtlaufzeit
- Vereinfacht die mathematische Analyse
- Erlaubt oft erst sinnvollen Vergleich von Algorithmen

$2n^2 \quad \sim \quad 2n^2 - 3n\lfloor \log_2(n+1) \rfloor + 7n - 3\lfloor \log_2(n+1) \rfloor + 120$



"Premature optimization is the root of all evil"

Quote by C.A.R. 'Tony' Hoare, popularized by Donald Knuth

Mathematische Tools

- **“Tilde-Notation:”** $f(n) \sim g(n)$ gdw (genau dann, wenn) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$

  „f und g sind *asymptotisch äquivalent*”

- **"Tilde-Notation:"** $f(n) \sim g(n)$ gdw $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$

  genau dann, wenn

  „f und g sind *asymptotisch äquivalent*"

- **"Big-Oh Notation:"** $f(n) \in O(g(n))$ gdw $\left|\frac{f(n)}{g(n)}\right|$ beschränkt für $n \geq n_0$

  also write '=' instead

  gdw $\lim\sup_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < \infty$

  need supremum since limit might not exist!

  Varianten:

- **"Tilde-Notation:"** $f(n) \sim g(n)$ gdw (genau dann, wenn) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$

  „f und g sind *asymptotisch äquivalent*"

- **"Big-Oh Notation:"** $f(n) \in O(g(n))$ (also write '=' instead) gdw $\left|\frac{f(n)}{g(n)}\right|$ beschränkt für $n \geq n_0$

  gdw $\limsup_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < \infty$ (need supremum since limit might not exist!)

Varianten:

- $f(n) = \Omega(g(n))$ ("Big-Omega") gdw $g(n) = O(f(n))$

- **"Tilde-Notation:"** $f(n) \sim g(n)$ (genau dann, wenn) gdw $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$

  „f und g sind *asymptotisch äquivalent*"

- **"Big-Oh Notation:"** $f(n) \in O(g(n))$ (also write '=' instead) gdw $\left|\frac{f(n)}{g(n)}\right|$ beschränkt für $n \geq n_0$

  gdw $\limsup_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < \infty$ (need supremum since limit might not exist!)

Varianten:

- $f(n) = \Omega(g(n))$ ("Big-Omega") gdw $g(n) = O(f(n))$
- $f(n) = \Theta(g(n))$ ("Big-Theta") gdw $f(n) = O(g(n))$ **und** $f(n) = \Omega(g(n))$

- **"Tilde-Notation:"** $f(n) \sim g(n)$ gdw $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$

  genau dann, wenn

  „f und g sind *asymptotisch äquivalent*"

- **"Big-Oh Notation:"** $f(n) \in O(g(n))$ gdw $\left|\frac{f(n)}{g(n)}\right|$ beschränkt für $n \geq n_0$

  also write '$=$' instead

  need supremum since limit might not exist!

  gdw $\limsup_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < \infty$

  Varianten:

  "Big-Omega"

  - $f(n) = \Omega(g(n))$ gdw $g(n) = O(f(n))$
  - $f(n) = \Theta(g(n))$ gdw $f(n) = O(g(n))$ **und** $f(n) = \Omega(g(n))$

    "Big-Theta"

- **"Little-Oh Notation:"** $f(n) \in o(g(n))$ gdw $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = 0$

  $\omega(g(n))$ wenn lim $= \infty$

- **"Tilde-Notation:"** $f(n) \sim g(n)$ gdw $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$

  genau dann, wenn

  „f und g sind *asymptotisch äquivalent*"

- **"Big-Oh Notation:"** $f(n) \in O(g(n))$ gdw $\left|\frac{f(n)}{g(n)}\right|$ beschränkt für $n \geq n_0$

  also write '$=$' instead

  need supremum since limit might not exist!

  gdw $\limsup_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < \infty$

  Varianten:

  "Big-Omega"

  - $f(n) = \Omega(g(n))$ gdw $g(n) = O(f(n))$
  - $f(n) = \Theta(g(n))$ gdw $f(n) = O(g(n))$ **und** $f(n) = \Omega(g(n))$

    "Big-Theta"

- **"Little-Oh Notation:"** $f(n) \in o(g(n))$ gdw $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = 0$

  $\omega(g(n))$ wenn lim $= \infty$

Beispiel

Beispiel: "Square-and-multiply" Algorithmus um x^m zu berechnen ($m \in \mathbb{N}$)

- m als Binärzahl gegeben; der Einfachheit halber: array der bits
- $n = \#\text{bits in } m$
- x als double gegeben

```
1  double pow(double base, boolean[] exponentBits) {
2      double res = 1;
3      for (final boolean bit : exponentBits) {
4          res *= res;
5          if (bit) res *= base;
6      }
7      return res;
8  }
```

Beispiel

Beispiel: "Square-and-multiply" Algorithmus um x^m zu berechnen ($m \in \mathbb{N}$)

- m als Binärzahl gegeben; der Einfachheit halber: array der bits
- $n = \#\text{bits in } m$
- x als double gegeben

```
1  double pow(double base, boolean[] exponentBits) {
2      double res = 1;
3      for (final boolean bit : exponentBits) {
4          res *= res;
5          if (bit) res *= base;
6      }
7      return res;
8  }
```

- Kosten: C = Anzahl Multiplikationen

Beispiel

Beispiel: “Square-and-multiply” Algorithmus um x^m zu berechnen ($m \in \mathbb{N}$)

- m als Binärzahl gegeben; der Einfachheit halber: array der bits
- $n = \#\text{bits in } m$
- x als double gegeben

```
1  double pow(double base, boolean[] exponentBits) {
2      double res = 1;
3      for (final boolean bit : exponentBits) {
4          res *= res;
5          if (bit) res *= b
6      }
7      return res;
8  }
```

Was ist die Anzahl Multiplikationen?

- Kosten: C = Anzahl Multiplikationen

Beispiel

Beispiel: "Square-and-multiply" Algorithmus um x^m zu berechnen ($m \in \mathbb{N}$)

- m als Binärzahl gegeben; der Einfachheit halber: array der bits
- $n = \#\text{bits in } m$
- x als double gegeben

```
1  double pow(double base, boolean[] exponentBits) {
2      double res = 1;
3      for (final boolean bit : exponentBits) {
4          res *= res;
5          if (bit) res *= base;
6      }
7      return res;
8  }
```

- Kosten: C = Anzahl Multiplikationen
- $C = n$ (Zeile 4) + #Eins-Bits in Binärdarstellung von m (Zeile 5)
  $\rightsquigarrow \quad n \leq C \leq 2n$

Beispiel

Beispiel: "Square-and-multiply" Algorithmus um x^m zu berechnen ($m \in \mathbb{N}$)

- m als Binärzahl gegeben; der Einfachheit halber: array der bits
- $n = \#\text{bits in } m$
- x als double gegeben

```
1  double pow(double base, boolean[] exponentBits) {
2      double res = 1;
3      for (final boolean bit : exponentBits) {
4          res *= res;
5          if (bit) res *= base;
6      }
7      return res;
8  }
```

Was ist die genaueste asymptotische Aussage über C (für $m \to \infty$), die wir treffen können?

- Kosten: C = Anzahl Multiplikationen
- $C = n$ (Zeile 4) + #Eins-Bits in Binärdarstellung von m (Zeile 5)
  $\rightsquigarrow \quad n \leq C \leq 2n$

Beispiel

Beispiel: "Square-and-multiply" Algorithmus um x^m zu berechnen ($m \in \mathbb{N}$)

- m als Binärzahl gegeben; der Einfachheit halber: array der bits
- $n = \#\text{bits in } m$
- x als double gegeben

```
1  double pow(double base, boolean[] exponentBits) {
2      double res = 1;
3      for (final boolean bit : exponentBits) {
4          res *= res;
5          if (bit) res *= base;
6      }
7      return res;
8  }
```

- Kosten: C = Anzahl Multiplikationen
- $C = n$ (Zeile 4) + #Eins-Bits in Binärdarstellung von m (Zeile 5)
  $\rightsquigarrow \quad n \leq C \leq 2n$

$\rightsquigarrow$ $C = \Theta(n) = \Theta(\log m)$

Beispiel

Beispiel: "Square-and-multiply" Algorithmus um x^m zu berechnen ($m \in \mathbb{N}$)

- m als Binärzahl gegeben; der Einfachheit halber: array der bits
- n = #bits in m
- x als double gegeben

```
1  double pow(double base, boolean[] exponentBits) {
2      double res = 1;
3      for (final boolean bit : exponentBits) {
4          res *= res;
5          if (bit) res *= base;
6      }
7      return res;
8  }
```

- Kosten: C = Anzahl Multiplikationen
- $C = n$ (Zeile 4) + #Eins-Bits in Binärdarstellung von m (Zeile 5)
  $\rightsquigarrow \quad n \leq C \leq 2n$

$\rightsquigarrow$ $C = \Theta(n) = \Theta(\log m)$

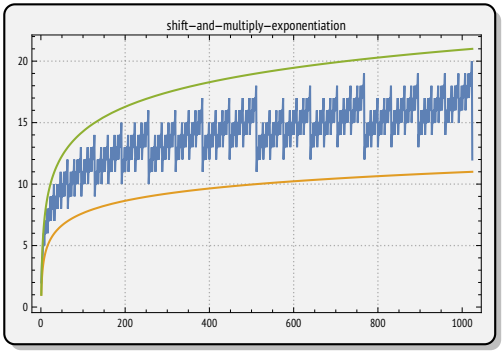
Achtung: Oft reicht die Intuition: „Θ ist wie $\sim$ mit unbekannter Konstante"
hier aber gibt es gar keine Konstante!

Beispiel

Beispiel: "Square-and-multiply" Algorithmus um x^m zu berechnen ($m \in \mathbb{N}$)

- m als Binärzahl gegeben; der Einfachheit halber: array der bits
- n = #bits in m
- x als double gegeben

```
1  double pow(double base, boolean[] exponentBits) {
2      double res = 1;
3      for (final boolean bit : exponentBits) {
4          res *= res;
5          if (bit) res *= base;
6      }
7      return res;
8  }
```



- Kosten: C = Anzahl Multiplikationen
- $C = n$ (Zeile 4) + #Eins-Bits in Binärdarstellung von m (Zeile 5)

  $\rightsquigarrow \quad n \leq C \leq 2n$

$\rightsquigarrow$ $C = \Theta(n) = \Theta(\log m)$

Achtung: Oft reicht die Intuition: „Θ ist wie $\sim$ mit unbekannter Konstante"
hier aber gibt es gar keine Konstante!