4 – Komplexitätstheorie

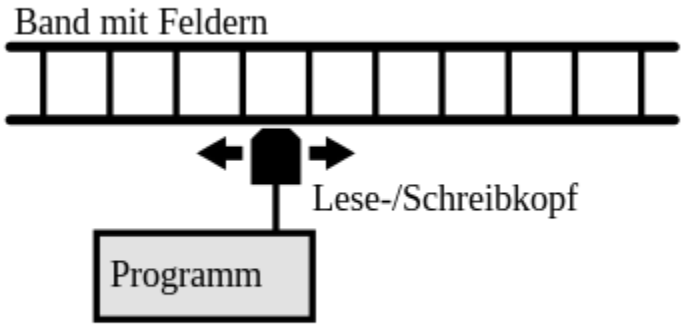
Wie geht es weiter?
Grundsatzfrage

Können alle berechenbaren Probleme auf einem Parallelrechner effizient gelöst werden?

Theoretische Rechnermodelle
Die Turingmaschine

Eine Turingmaschine ist ein theoretisches Rechnermodell mit…

- Einem unendlich langen Speicherband, welches pro Feld genau ein Zeichen aus dem Bandalphabet speichern kann
- Einem Bandalphabet Γ
- Einem „Leerzeichen“ B
- Einem Lese-/Schreibkopf, der sich feldweise auf dem Speicherband bewegen kann
- Einem Zustandsraum Q



Quelle: http://de.wikipedia.org/wiki/Turingmaschine

Theoretische Rechnermodelle
Die Turingmaschine

Ein Programm auf einer Turingmaschine besteht aus …

- Eine Anfangszustand $q_0 \in Q$
- Einem Stoppzustand $\bar{q} \in Q$
- Einer Eingabe, welche auf dem Speicherband steht
- Einer Zustandsübergangsfunktion $\delta: (Q \setminus \{\bar{q}\}) \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$
  - Eine Auswertung von δ sieht wie folgt aus: $\delta(q, a) = (q', a', d)$
    - $q \in \mathrm{Q}$, aktueller Zustand der Turingmaschine
    - $a \in \Gamma$, Zeichen im aktuellen Feld
    - $q' \in \mathrm{Q}$, Folgezustand der Turingmaschine
    - $a' \in \Gamma$, Zeichen welches im aktuellen Feld gespeichert wird
    - $d \in \{R, L, N\}$, Bewegung des Lese-/Schreibkopfes

Zu Beginn des Programms steht der Lese-/Schreibkopf auf dem ersten Zeichen der Eingabe. Danach wird δ immer wieder aufgerufen, bis $\bar{q}$ erreicht ist.

Theoretische Rechnermodelle
Die Turingmaschine - Beispiel

Es seien $Q = \{q_0, \bar{q}\}$, $\Gamma = \{0, 1, B\}$ und δ wie folgt:

δ	0	1	B
q_0	$(\bar{q}, 0, N)$	(q_0, B, R)	$(\bar{q}, 1, N)$

Die Turingmaschine prüft, ob die Eingabe nur aus „1" besteht

- Ist eine „0" in der Eingabe, terminiert das Programm mit „0"
- Sind nur „1" in der Eingabe, terminiert das Programm mit „1"
- Alle „1" in der Eingabe werden mit Leerzeichen überschrieben

Theoretische Rechnermodelle
Die Registermaschine

Eine Registermaschine ist ein theoretisches Rechnermodell mit...

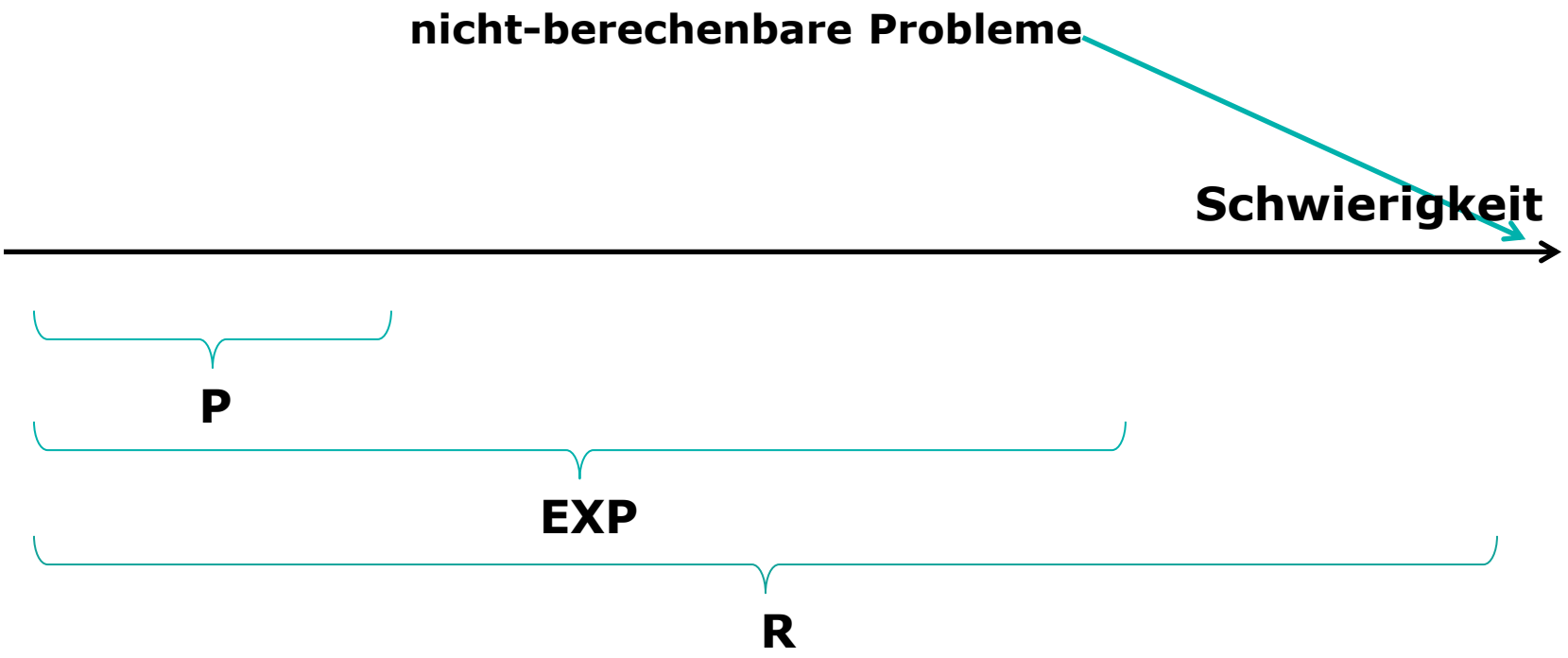
- Einer bestimmten Anzahl von Registern, welche natürliche Zahlen speichern können
- Drei Operationen, welche auf einem Register ausgeführt werden können
  - > Inkrementierung eines Registers
  - > Dekrementierung eines Registers, falls das Register noch nicht 0 enthält
  - > Test, ob ein Register eine „0“ enthält

- Ein Programm ist eine Aneinanderreihung von Operationen auf Registern.
- Die Registermaschine ist sehr ähnlich zu einem modernen Computer.
- Registermaschine und Turingmaschine sind äquivalente Modelle, da sie sich gegenseitig in polynomieller Laufzeit simulieren können.

Es gibt verschiedene Komplexitätsklassen, welche die „Schwierigkeit" von Problemen beschreiben

- **P**
  - > Alle Probleme, die in polynomieller Laufzeit zu berechnen sind
  - > $O(n^c)$
- **EXP**
  - > Alle Probleme, die in exponentieller Laufzeit zu berechnen sind
  - > $O(2^{n^c})$
- **R**
  - > Alle Probleme, die in endlicher Laufzeit zu berechnen sind
  - > „rekursive" Probleme = berechenbare Probleme

Komplexitätstheorie
Komplexitätsklassen



$$P \subset EXP \subset R$$

Komplexitätstheorie
Beispiele

Sortieralgorithmen sind in P

- Laufzeit von $O(n)$ bis $O(n^2)$ [bzw. $O(n^{\frac{\log(n)}{2}})$...]

Schach ist in EXP, aber nicht in P

- Der Spielbaum hat schätzungsweise 10^{123} Knoten, sofern man zyklische Spielzüge nicht beachtet

Klassen von Problemen
Nicht-berechenbare Probleme

Ein Problem heißt nicht-berechenbar (nicht-"entscheidbar") wenn es keinen Algorithmus (allgemeiner: keine Turing-Maschine) gibt, welcher das Problem in endlich vielen Schritten löst („berechnet").

Solche Probleme sind nicht in der Komplexitätsklasse R.

Beispiel: Das Halteproblem
Es gibt keinen Algorithmus, welcher entscheidet, ob ein **beliebiger** anderer Algorithmus terminiert.

Klassen von Problemen
Durchführbare bzw. nicht-durchführbare Probleme

Selbst wenn ein Problem berechenbar ist, bedeutet das nicht, dass es „praktisch" berechenbar ist.

- Viele Probleme benötigen für die Lösung zu viel Zeit oder Speicherplatz, um einen realistischen Problemfall berechnen zu können

Beispiel: Das Problem des Handlungsreisenden ist für realistische Problemgrößen nicht „durchführbar".

Entscheidungsprobleme
Allgemeines

Ein Entscheidungsproblem ist eine „Frage", zu der „JA" und eine „NEIN" Antworten existieren

- Die tatsächliche Ausgabe hängt von den Parametern des Funktionsaufrufs ab
- Parameter sind als Binärstring bzw. natürliche Zahlen darstellbar
- Formal:
  - $f(n) = \{0, 1\}, \quad n \in \mathbb{N}$

Beispiel: Teilt ein X ein Y ohne Rest?

Hinweis:

Die Turingmaschine aus dem Beispiel löst ein Entscheidungsproblem.

Behauptung:

Die meisten Entscheidungsprobleme sind nicht berechenbar.

Entscheidungsprobleme
Berechenbarkeit

Vorüberlegungen:

- Programme sind darstellbar als Binärstring $\triangleq n \in \mathbb{N}$
- Entscheidungsprobleme sind darstellbar als Funktion $f(n) = \{0,1\},\ n \in \mathbb{N}$
- Entscheidungsprobleme sind aber auch als Tabelle darstellbar:

Eingabe	**0**	**1**	**2**	**3**	**4**	**…**
Ausgabe	**{0,1}**	**{0,1}**	**{0,1}**	**{0,1}**	**{0,1}**	**…**

Entscheidungsprobleme sind also eine unendlich lange Folge von Bits.

Entscheidungsprobleme
Berechenbarkeit

Eingabe	0	1	2	3	4	...
Ausgabe	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}	...

Ausgehend von solch einer Tabelle können Entscheidungs-probleme wie in folgendem Beispiel gezeigt als reelle Zahlen dargestellt werden:

Eingabe	0	1	2	3	4	...
Ausgabe	1	0	1	1	0	...

Die entsprechende reelle Zahl ist 0,10110...

Entscheidungsprobleme
Berechenbarkeit

- **Entscheidungsprobleme können also als reelle Zahlen in Form von unendlich langen Binärstrings dargestellt werden.**

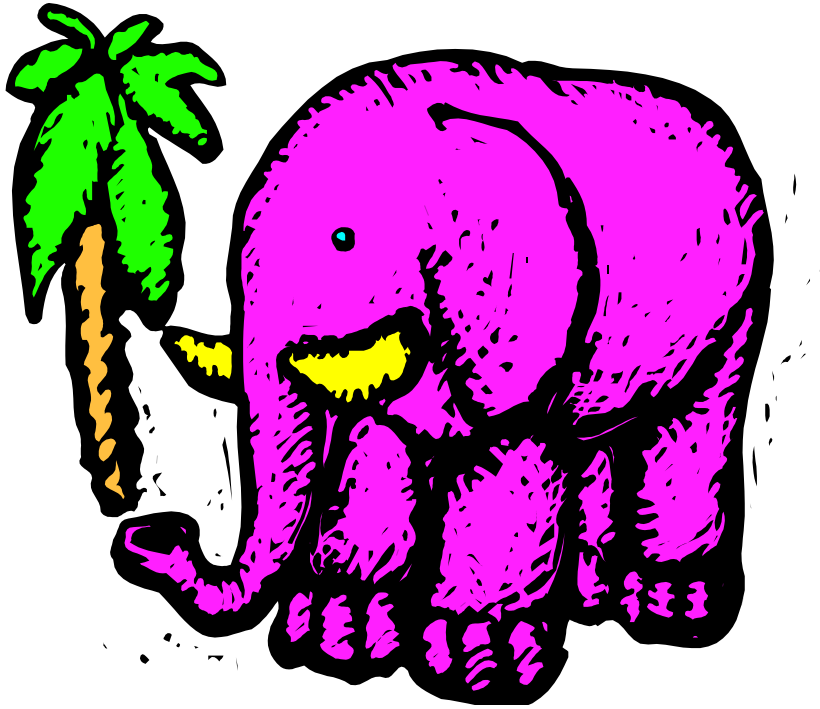
- **Programme hingegen werden als endlich lange, natürliche Zahl dargestellt.**

- **Da - wie bekannt - gilt, dass $|\mathbb{R}| \gg |\mathbb{N}|$, gibt es deutlich mehr Entscheidungsprobleme, als Programme zur Lösung dieser Probleme.**

NP
Definition

NP ist die Komplexitätsklasse, welche diejenigen Probleme enthält, welche von einer nicht-deterministischen Turingmaschine in polynomieller Zeit gelöst werden können.

Aber:
Was bedeutet das?



NP
Nicht-deterministische Turingmaschine

Eine nicht-deterministische Turingmaschine kann beim Lösen eines Problems „raten" und rät immer die korrekte Alternative aus.

Beispiel: Baumsuche

- Entscheidungsproblem: Existiert ein Blatt mit Wert X?
- Falls kein Blatt existiert wird „NEIN" ausgegeben
- Falls ein solches Blatt existiert wird „JA" ausgegeben
  - > Bei der Suche wird die Turingmaschine sich auf jeder Ebene des Baumes für ein Kindknoten entscheiden, welcher auf einem Pfad zum gesuchten Wert liegt

NP
Alternative Definition

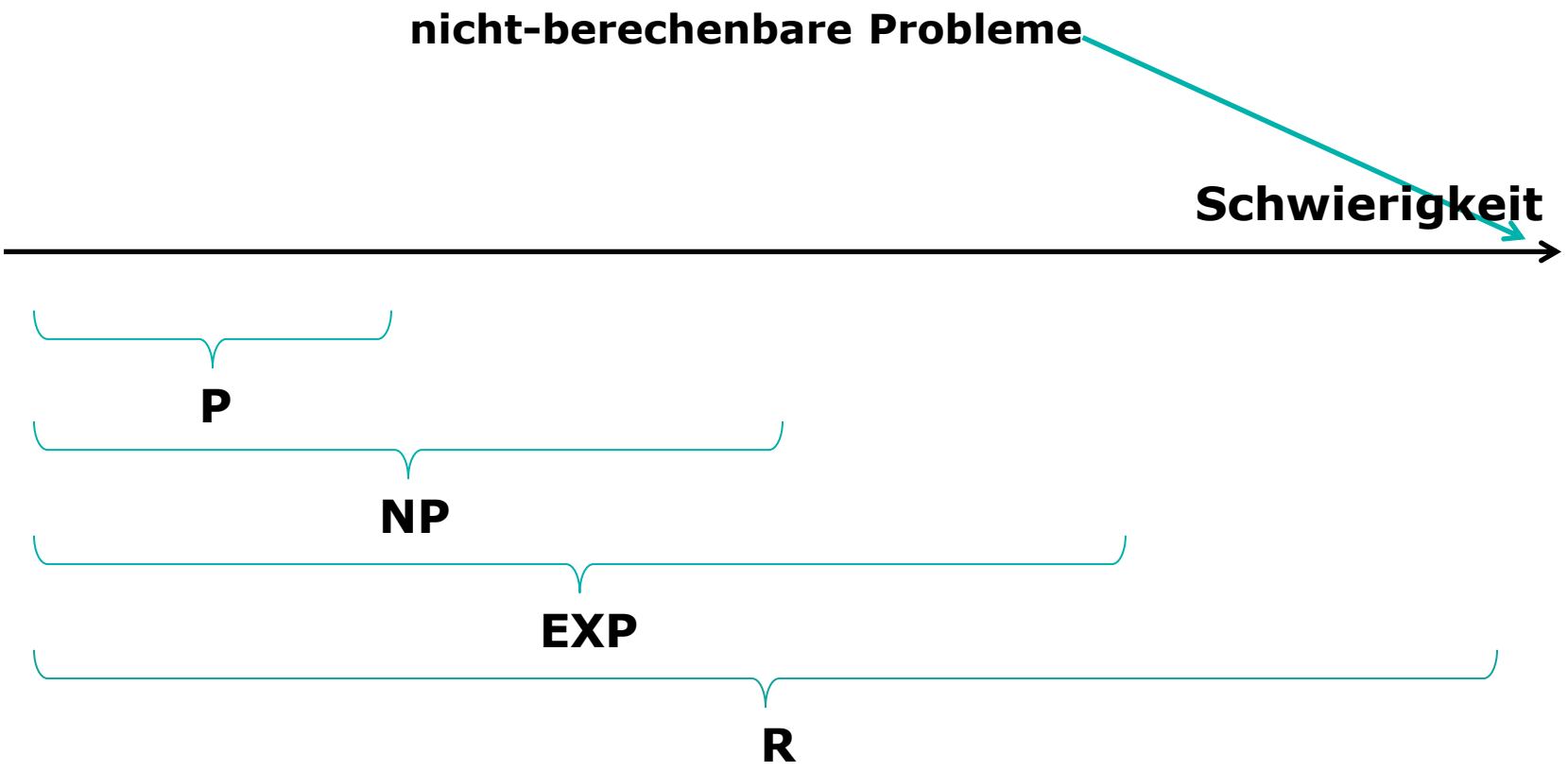
In der Klasse NP liegen diejenigen Probleme, deren Lösungskandidaten sich in polynomieller Laufzeit überprüfen lassen.

- Wenn die Lösung eines Entscheidungsproblemes „JA“ ist, so lässt sich die Lösung in polynomieller Laufzeit verifizieren

Beispiel: Baumsuche

- Sobald eine nicht-deterministische Turingmaschine einen Pfad zu einem Blatt mit dem gesuchten Wert X berechnet hat kann dieser Pfad von einer normalen Turingmaschine überprüft werden

Komplexitätstheorie
Komplexitätsklassen mit NP



$$P \subset NP \subset EXP \subset R$$

NP
Begrifflichkeiten

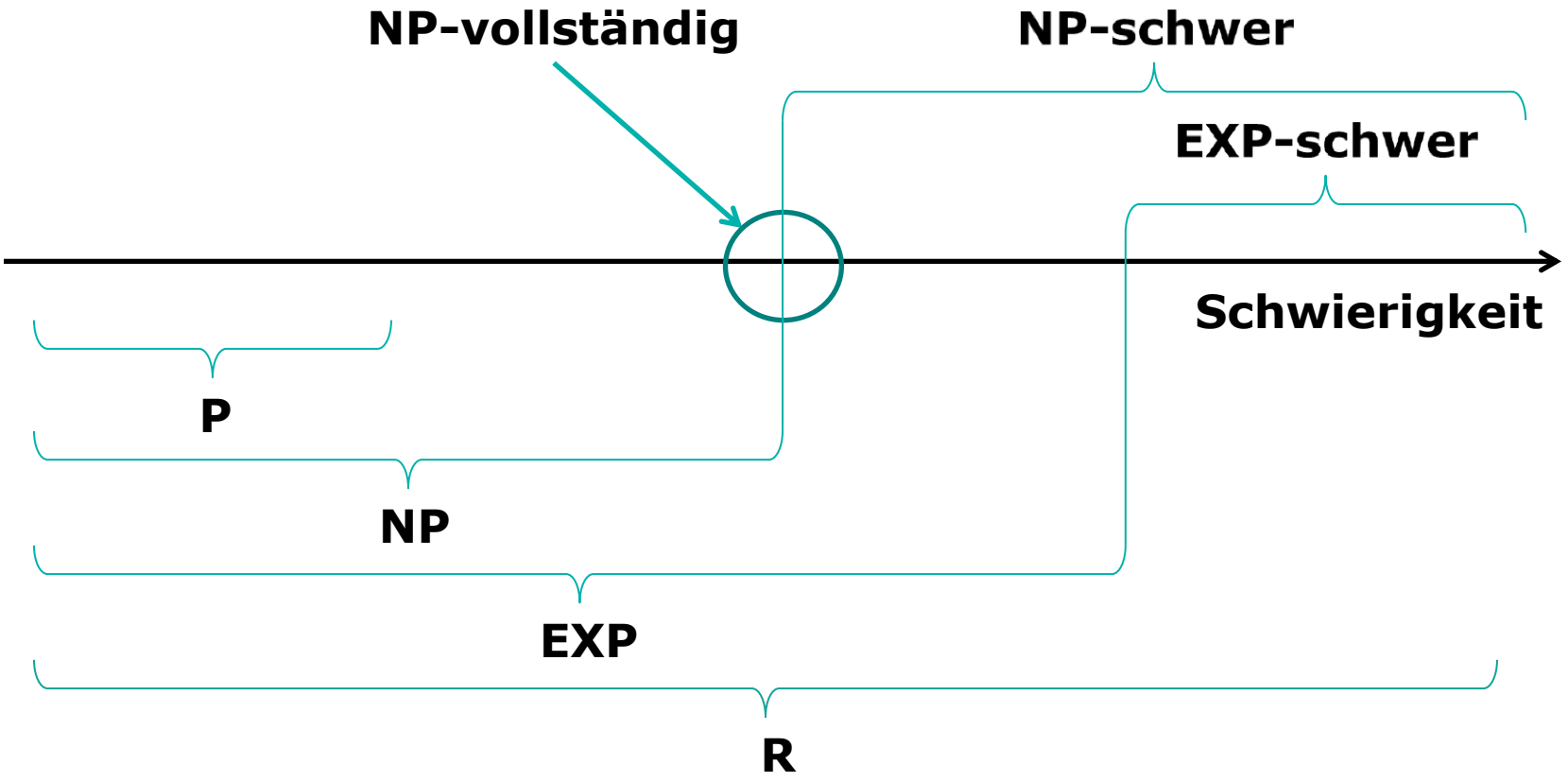
NP-schwer

- Ein Problem ist NP-schwer, wenn es mindestens so „schwer" ist, wie ein beliebiges Problem in NP
- Dieses Problem muss NICHT in NP liegen, es kann auch in EXP oder R liegen

NP-vollständig

- Ein NP-schweres Problem, welches selbst in NP liegt, wird NP-vollständig genannt

Komplexitätstheorie
Komplexitätsklassen mit NP



$$P \subset NP \subset EXP \subset R$$

NP
Beispiele für Probleme in NP

- **Problem des Handlungsreisenden**
- **Rucksack-Problem**
- **Graph-Coloring**
- **...**

Führe ein ungelöstes Problem A auf ein anderes Problem B zurück, dessen Lösung bereits bekannt ist.

- Sofern diese Reduktion möglich ist, ist B mindestens so schwer wie A
- B kann auch schwerer sein, daher bringt nicht jede Reduktion einen Vorteil
- Formal: $A \preccurlyeq B$

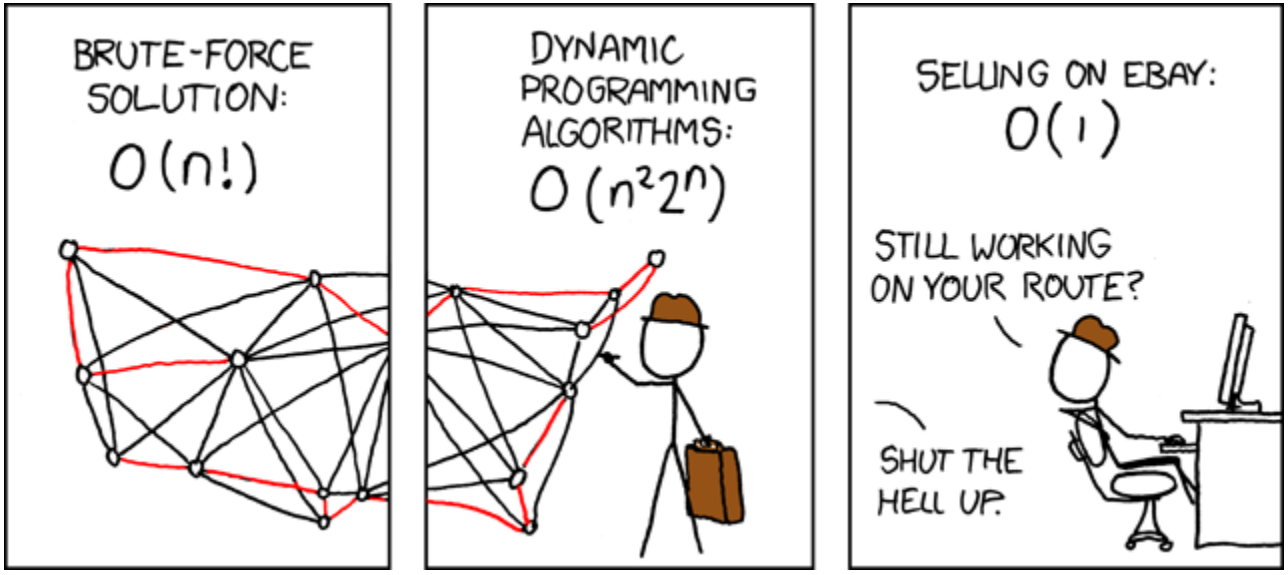
Beispiel: Quadrieren von Zahlen

Eine Zahl zu Quadrieren kann auf die Multiplikation von Zahlen zurückgeführt werden, indem man für die Multiplikation zwei gleiche Zahlen verwendet.

Ein Problem A ist also NP-schwer, wenn jedes Problem aus NP auf A reduzierbar ist.

Was bedeuten Komplexitätsklassen für den Alltag?

- Scharfe Trennung zwischen P und NP in der Theorie vorhanden
  - > In der Realität sind bereits viele Probleme in P kaum praktisch lösbar
  - > „Kleine" Instanzen von Problemen in NP sind in der Realität durchaus berechenbar



Taken from XKCD.com

Komplexitätstheorie
Abschließende Frage

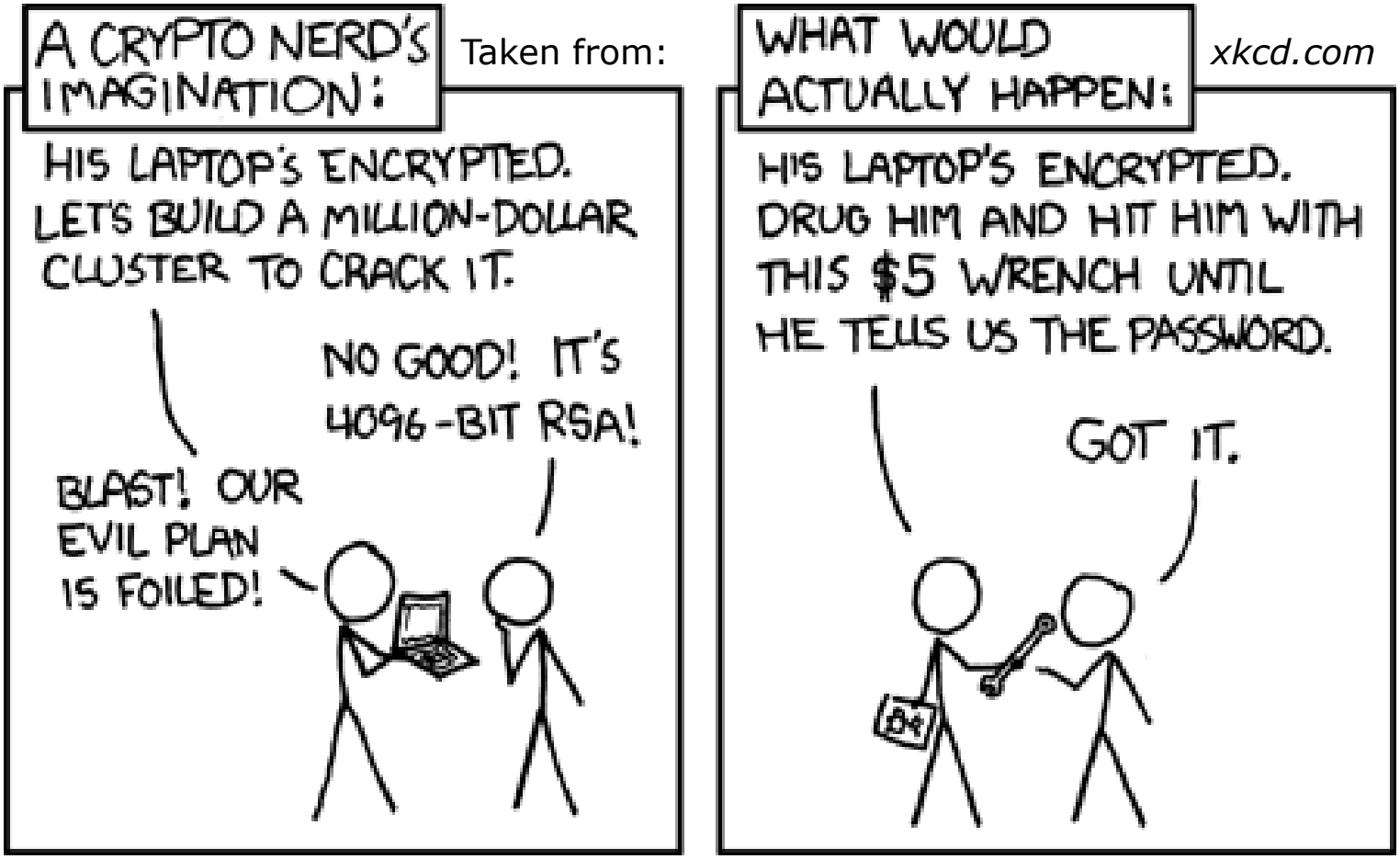
P = NP?

(Ungelöst; Die Lösung ist 10^6 $ wert)

4 – Komplexitätstheorie

Anwendung: Kryptographie

Kryptographie



Kryptographie
Wiederholung

- Klartext M
  > Unverschlüsselte Nachricht

- Geheimtext (Chiffretext) C
  > Verschlüsselte Nachricht

- (De-) Chiffrierung
  > Ent- bzw. Verschlüsselung von Informationen

- Schlüssel k
  > Vertrauliche Information zum Ver- bzw. Entschlüsseln

- Einwegfunktion
  > Mathematische Funktion, welche effizient zu berechnen ist; die Umkehrung der Funktion ist **nicht** effizient zu berechnen

Einwegfunktion sind zur Verschlüsselung wünschenswert

- Verschlüsselung soll einfach zu berechnen sein
- Entschlüsselung ohne den korrekten Schlüssel soll schwer zu berechnen sein
- Schwierigkeit liegt in der Tatsache, dass eine Entschlüsselung mit dem korrekten Schlüssel ebenfalls einfach zu berechnen sein soll

Wunschvorstellung:

- Verschlüsselung / Entschlüsselung mit Schlüssel in **P**
- Entschlüsselung ohne Schlüssel in **NP**

Problem

- Es ist nicht bekannt, ob echte Einwegfunktionen existieren, oder ob die Algorithmen zur effizienten Umkehrung nur noch unbekannt sind
- Sollten echte Einwegfunktionen existieren wäre gleichzeitig **P ≠ NP** bewiesen

Kryptographie
Einwegfunktionen – Zwei Beispiele

Integerfaktorisierung $\mathbf{m = a_1 * a_2}$

- Zwei (Prim-)Zahlen a_1 und a_2 miteinander zu multiplizieren hat nur polynomiellen Zeitaufwand
- Für das Zerlegen des erzeugte Produkts m in seine Primfaktoren, ohne Kenntnis mindestens eines Faktors zu haben, ist jedoch kein Polynomialzeitalgorithmus bekannt

Diskreter Logarithmus $b = r^i \, mod \, p$

- Das potenzieren einer Primitivwurzel r in einer primen Restklassengruppe hat nur polynomiellen Zeitaufwand
- Für das Berechnen des Logarithmus von b zur Basis r ist jedoch kein Polynomialzeitalgorithmus bekannt

Zum Ver- und Entschlüsseln werde verschiedene Schlüssel benutzt

- Privater Schlüssel („private key“) zum Entschlüsseln
- Öffentlicher Schlüssel („public key“) zum Verschlüsseln

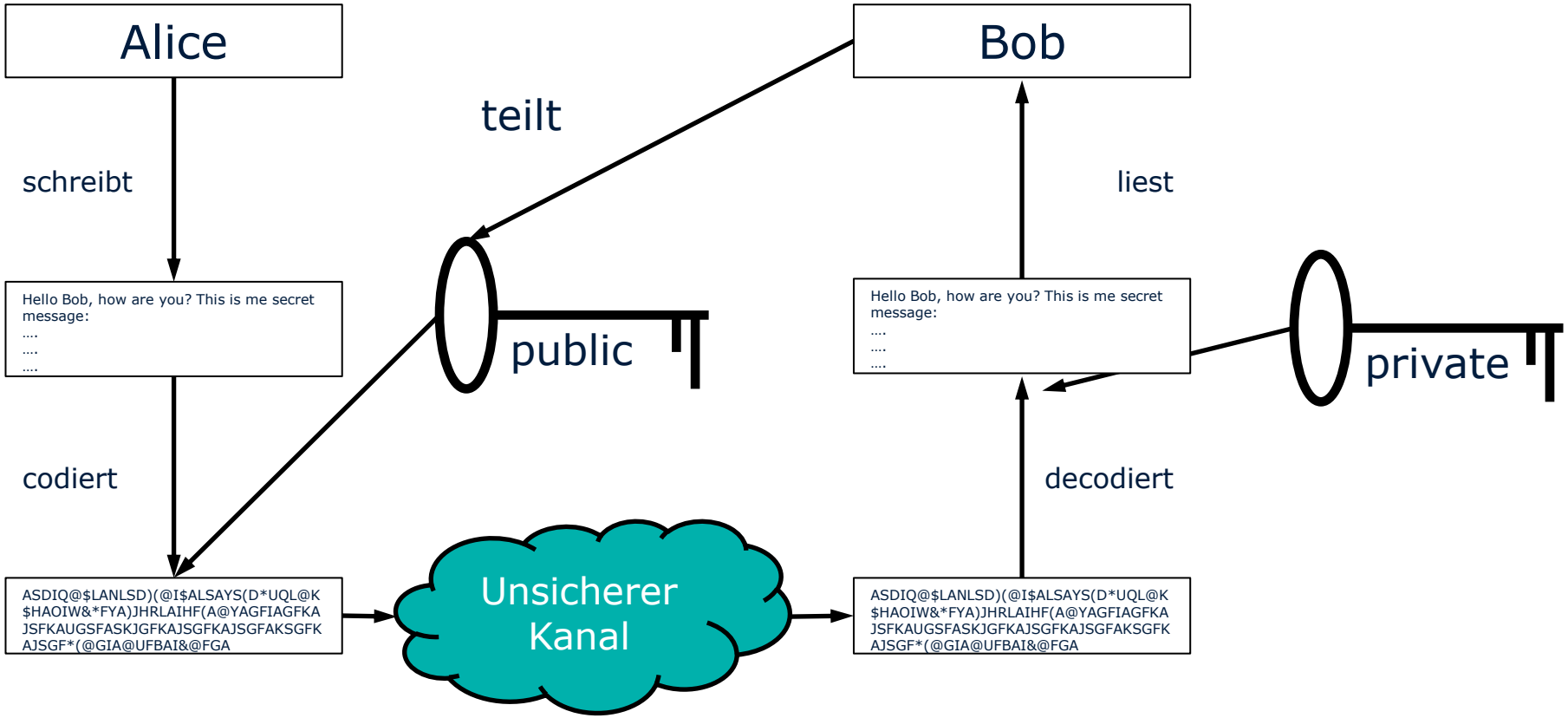
Vorteile

- Privater Schlüssel muss nicht übertragen werden

Nachteile

- Rechenaufwand höher als bei symmetrischen Verfahren (~ Faktor 103)

Kryptographie
Asymmetrische Verfahren – Wiederholung



Die Sicherheit von RSA basiert auf der Annahme, dass Primfaktorzerlegungen nicht effizient berechnet werden können

- Schlüsselpaar
  - > Privater Schlüssel (d, N)
  - > Öffentlicher Schlüssel (e, N)

- Algorithmus
  - > Verschlüsselung $C = M^e \; mod \; N$
  - > Entschlüsselung $M = C^d \; mod \; N$

Kryptographie
RSA - Schlüsselerzeugung

1. Wähle 2 Primzahlen p und q mit $p \neq q$
2. Berechne $N = p * q$
3. Berechne $\varphi(N) = (p-1) * (q-1)$
4. Wähle ein zu $\varphi(N)$ teilerfremdes e mit $1 < e < \varphi(N)$
5. Berechne ein d sodass $e * d \equiv 1 \, mod \, \varphi(N)$
6. p, q, k und $\varphi(N)$ werden nicht länger benötigt und sollten sicher(!) vernichtet werden

Das Schlüsselpaar ist (d, N) und (e, N)

1. $p = 11, q = 13$

2. $N = p * q = 11 * 13 = 143$

3. $\varphi(N) = (p-1) * (q-1) = 10 * 12 = 120$

4. $W\ddot{a}hle\ e = 23$

5. $23 * d + k * 120 = 1 \rightarrow d = 47, k = -9$

Das Schlüsselpaar ist (47, 143) und (23, 143)

Die Ver- bzw. Entschlüsselung bei RSA läuft wie folgt ab

1. Übersetzung der Nachricht in eine Zahl, sofern nötig
   - z.B. durch Bitdarstellung eines Textes
   - Beispiel: ‚*‘ $\triangleq 00101010_b = 42_d$

2. Berechnen des Geheimtextes
   - Beispiel:
     > Klartext $M = 42$, Schlüssel $k = (23, 143)$
     > $C = 42^{23} \, mod \, 143 = 113$

3. Erneutes Berechnen des Klartextes
   - Beispiel:
     > Geheimtext $C = 113$, Schlüssel $k = (47, 143)$
     > $M = 113^{47} \, mod \, 143 = 42$

Die Ver- bzw. Entschlüsselung bei RSA ist gleich aufwändig.

Doch wie sieht es mit dem simplen Umkehren der Verschlüsselung, d.h. dem Brechen der Verschlüsselung, aus?

- Die Berechnung von M aus $C = M^e \, mod \, N$ entspricht der Suche nach $\sqrt[e]{C}$, also der e-ten Wurzel von C, unter dem gegebenen Modul. Hierfür ist kein effizienter Algorithmus bekannt.
- Alternativ kann versucht werden das N in seine zwei Primfaktoren p und q zu zerlegen und unter Zuhilfenahme des Algorithmus zur Schlüssel-erzeugung den privaten Schlüssel (d, N) zu berechnen. Dies ist eine Integerfaktorisierung, für die ebenfalls kein effizienter Algorithmus bekannt ist.

4 – Komplexitätstheorie

Randomisierte Algorithmen

Randomisierte Algorithmen
Grundidee

Es gibt Algorithmen, deren Ablauf in bestimmtem Maße zufällig gesteuert wird

- Zufallsbits sind neben den „normalen" Parametern auch eine Eingabe des Algorithmus
- Laufzeit kann eine Zufallsvariable sein
- Verwendeter Speicherplatz kann eine Zufallsvariable sein
- Ergebnis kann auch eine Zufallsvariable sein

Will man das?

(Und wozu überhaupt?)

Randomisierte Algorithmen
Wozu überhaupt?

Randomisierte Algorithmen sind manchmal, …

- Einfacher bzw. leichter zu implementieren
- Die einzige Möglichkeit überhaupt – effizient - ein Ergebnis zu berechnen

Randomisierte Algorithmen
Einfacher bzw. leichter zu implementieren?

In den meisten Fällen ist die Struktur eines randomisierten Algorithmus bedeutend einfacher, als die Struktur eines „schlaueren" deterministischen Algorithmus

(Nicht ganz ernst gemeintes) **Beispiel 1:** Sortieralgorithmen

- Randomisiert und im Mittel langsam, aber einfach:
  Bogosort
  - > Average Case: $O(n * n!)$
  - > Best Case: $O(n)$
  - > Worst Case: $O(\infty)$
- Deterministisch und im Mittel schnell, aber kompliziert:
  Quicksort
  - > Average Case: $O(n * \log n)$
  - > Best Case: $O(n * \log n)$
  - > Worst Case: $O(n^2)$

Beispiel 2: Berechnung von π ($\rightarrow$ Netbeans)

Randomisierte Algorithmen
Die einzige Möglichkeit… ?

Speziell Optimierungsprobleme sind teilweise derart rechenintensiv, dass ein Ergebnis für realistische Szenarien mit deterministischen Algorithmen nicht in annehmbarer Zeit zu berechnen ist.

- Randomisierte Algorithmen liefern Näherungen der korrekten Lösung in einem Bruchteil der Zeit
- Näherungen in der Praxis oft ausreichend genau

Beispiel: Traveling-Salesman-Problem

- Lösung durch Monte-Carlo-Algorithmen
- Lösung durch Heuristiken

Randomisierte Algorithmen
Monte-Carlo-Algorithmus?

Ein Monte-Carlo-Algorithmus ist ein randomisierter Algorithmus, welcher mit einer beschränkten Wahrscheinlichkeit ein falsches Ergebnis liefern darf

- Durch Wiederholung des Algorithmus mit neuen Zufallsbits kann die Fehlerwahrscheinlichkeit beliebig gesenkt werden

- Es gibt Monte-Carlo-Algorithmen für Suchprobleme und Entscheidungsprobleme
  - > Absoluter bzw. relativer Fehler bei Suchproblemen
  - > Ein- bzw zweiseitiger Fehler bei Entscheidungsproblemen (*false negatives, false positives*)

- Komplexitätsklassen bei Entscheidungsproblemen
  - > **BPP** (*bounded error probabilistic polynomial time*): Das korrekte Ergebnis wird mit mindestens mit Wahrscheinlichkeit $p = \frac{2}{3}$ ausgegeben
  - > **RP** (randomized polynomial time): „JA" wird mit mindestens $p = 0.5$ korrekt ausgegeben. „NEIN" wird mit $p = 1$ korrekt ausgegeben.

Beispiele

1. Miller-Rabin-Primzahltest
   > Der Miller-Rabin-Primzahltest bestimmt, ob eine gegebene Zahl prim ist. Zusammengesetze Zahlen werden immer korrekt erkannt, Primzahlen nur mit einer Wahrscheinlichkeit von 75%.

2. Probabilistische Bestimmung von π
   > Siehe Beispielprogramm aus der Vorlesung

3. Monte-Carlo-Tree-Search
   > Ein Suchverfahren aus der künstlichen Intelligenz, welches verwendet wird um „gute" Spielverläufe zu finden.

Randomisierte Algorithmen
Las-Vegas-Algorithmus

Ein Las-Vegas-Algorithmus ist ein randomisierter Algorithmus, welche nie ein falsches Ergebnis liefern darf

- Die Rechenzeit kann sehr groß werden
- Es gibt Varianten, in denen der Algorithmus „aufgeben" darf

Beispiel: Random Quicksort

- Das Pivot-Element beim Quicksort wird immer zufällig gewählt
- Die Sortierung ist stets korrekt, jedoch kann das Laufzeitverhalten sehr schlecht werden, ist im Mittel aber dieselbe wie bei der regulären Quicksort-Variante.

Randomisierte Algorithmen
Heuristik?

Funktionsweise ist ähnlich zu Monte-Carlo-Algorithmen, allerdings wird der Zufall durch „Faustregeln" bzw. „intelligentes Raten" beeinflusst

- Abhängig vom Problem stellen Heuristiken eine Verbesserung der „normalen" Monte-Carlo-Algorithmen dar
- Heuristiken kommen aus dem Themenfeld der künstlichen Intelligenz
- Heuristiken können bei Suchproblemen benutzt werden um Schranken abzuschätzen, sofern sie beweisbar eine gewisse Genauigkeit aufweisen

Beispiel: Spiele-KI

- Monte-Carlo-Tree-Search durchläuft den Spielbaum völlig zufällig. Es ist aber z.B. bei Schach nur sehr selten sinnvoll bewusst eine Figur zu opfern. Es ist also „intelligent" Knoten, bei denen der Gegner sich selbst schadet, weniger wahrscheinlich bei einem Durchlauf zu besuchen.