Technische Universität Braunschweig

Algorithmen und Datenstrukturen 2 – Übung #5

Komplexität, Reduktionen und Hashing

Ramin Kosfeld und Chek-Manh Loi

02.07.2025

Heute

Heute

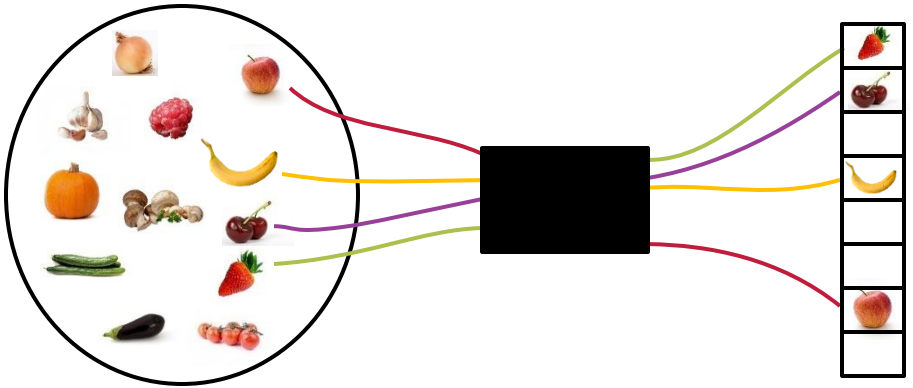
- Komplexität
- Reduktionen

$$\varphi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee x_4)$$

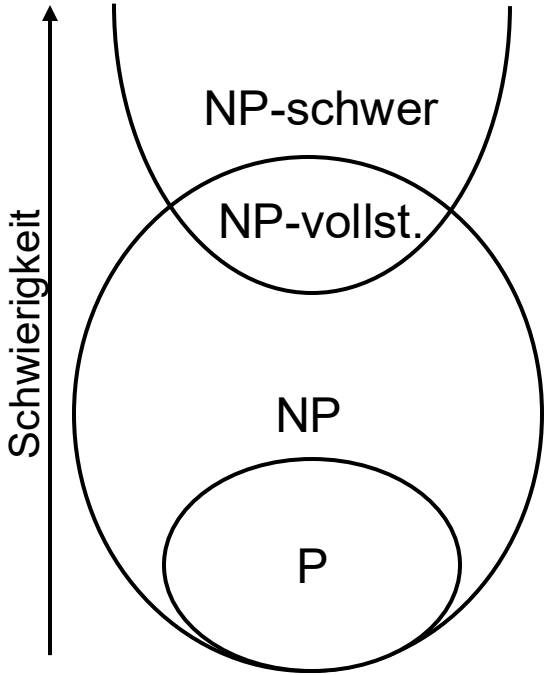
Heute

- Komplexität
- Reduktionen
- Hashing
- Modulo

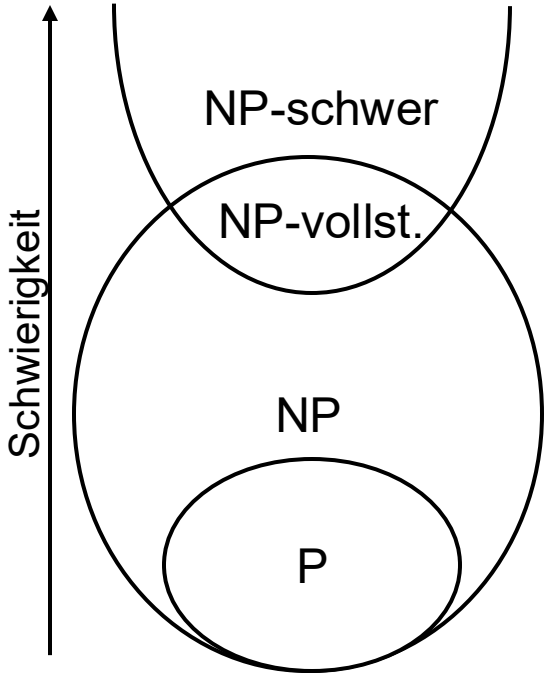
$$\varphi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee x_4)$$



Komplexitätsklassen

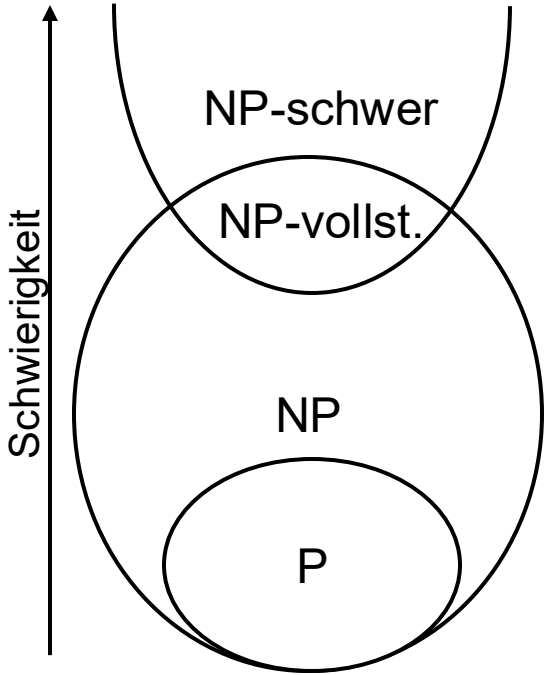


Komplexitätsklassen



P: Probleme lassen sich effizient lösen.

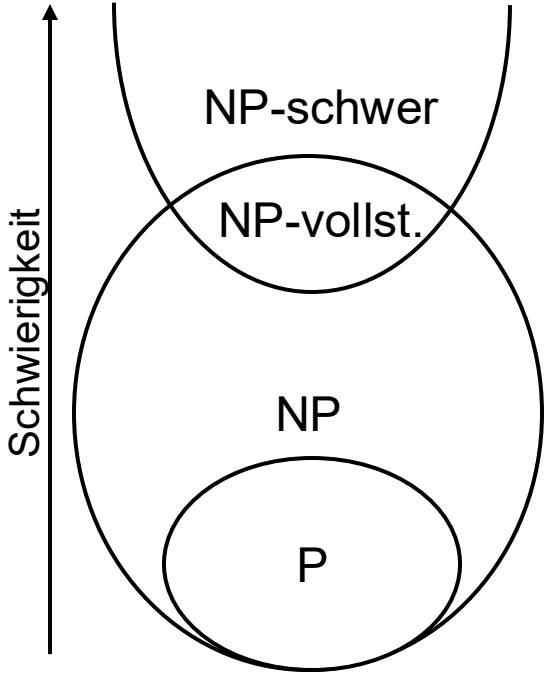
Komplexitätsklassen



P: Probleme lassen sich effizient lösen.

NP: Lösungen können effizient verifiziert werden.

Komplexitätsklassen

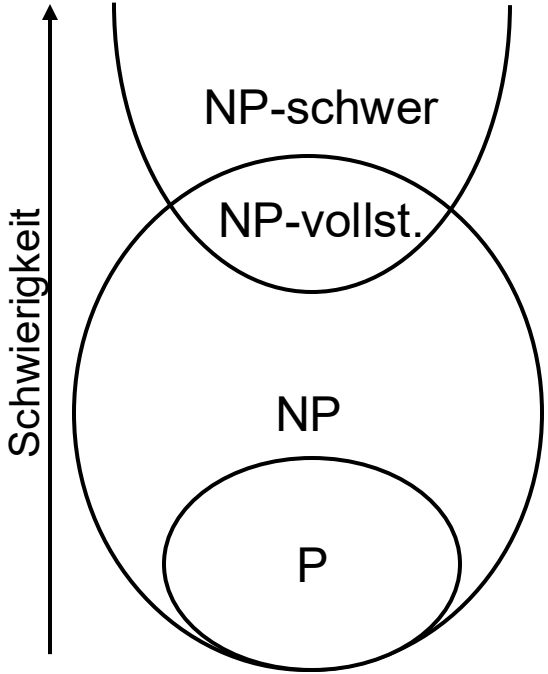


P: Probleme lassen sich effizient lösen.

NP: Lösungen können effizient verifiziert werden.

15

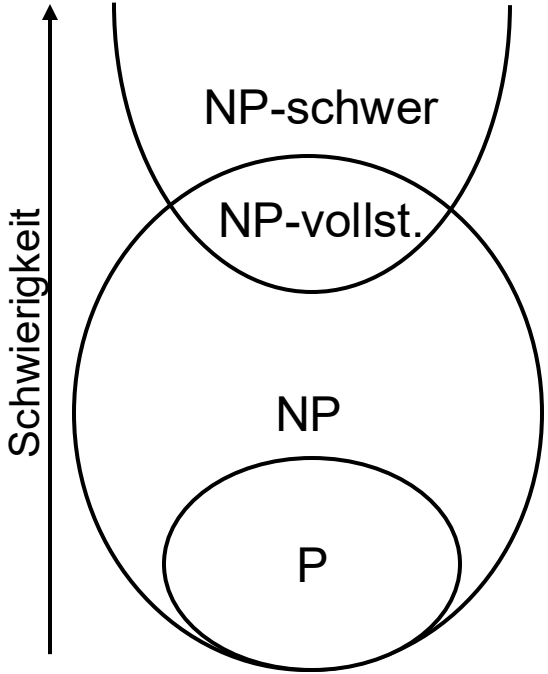
Komplexitätsklassen



P: Probleme lassen sich effizient lösen.

NP: Lösungen können effizient verifiziert werden.

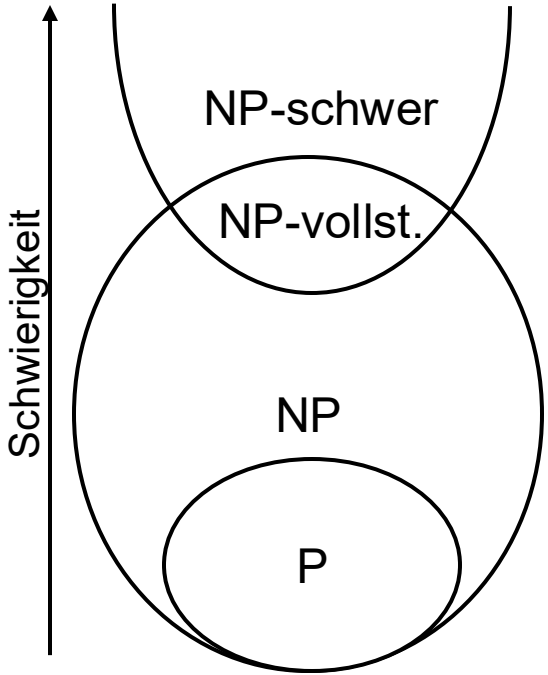
Komplexitätsklassen



P: Probleme lassen sich effizient lösen.

NP: Lösungen können effizient verifiziert werden.

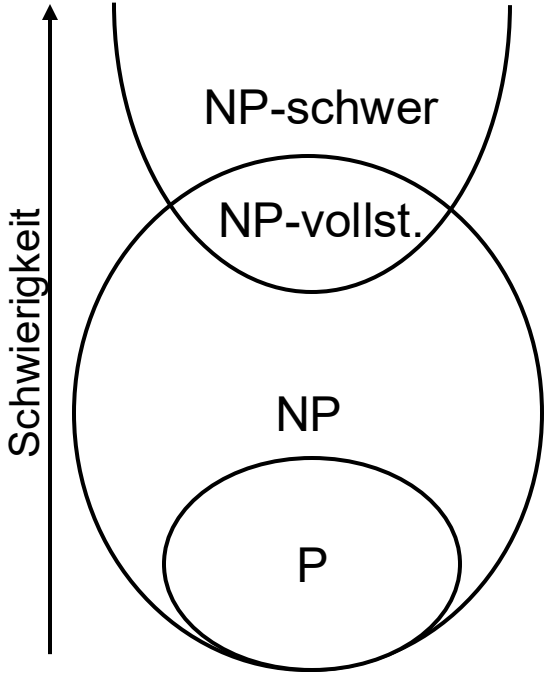
Komplexitätsklassen



P: Probleme lassen sich effizient lösen.

NP: Lösungen können effizient verifiziert werden.

Komplexitätsklassen

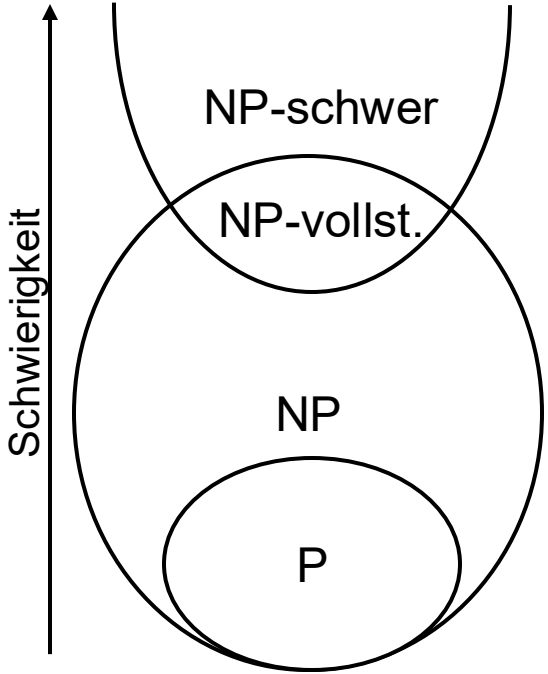


P: Probleme lassen sich effizient lösen.

NP: Lösungen können effizient verifiziert werden.

NP-schwer: Wenn das Problem in P liegt, gilt P=NP.

Komplexitätsklassen



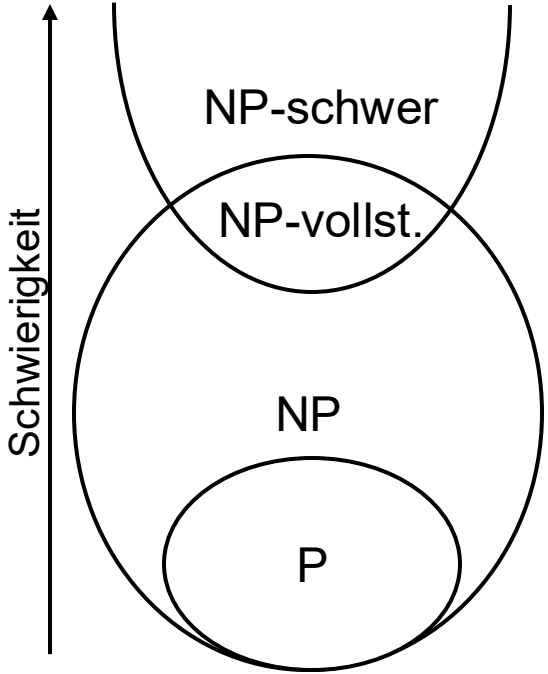
P: Probleme lassen sich effizient lösen.

NP: Lösungen können effizient verifiziert werden.

NP-schwer: Wenn das Problem in P liegt, gilt P=NP.

NP-vollständig: Problem liegt in NP und ist NP-schwer.

Komplexitätsklassen



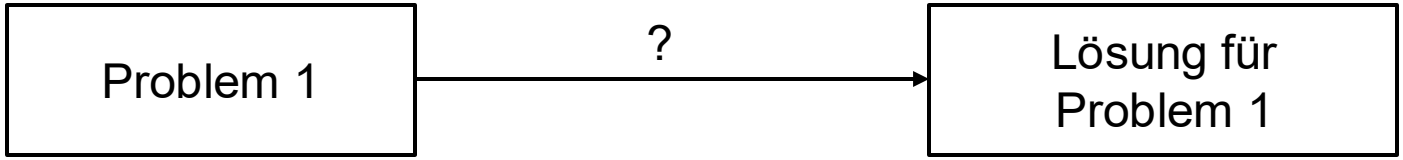
P: Probleme lassen sich effizient lösen.

NP: Lösungen können effizient verifiziert werden.

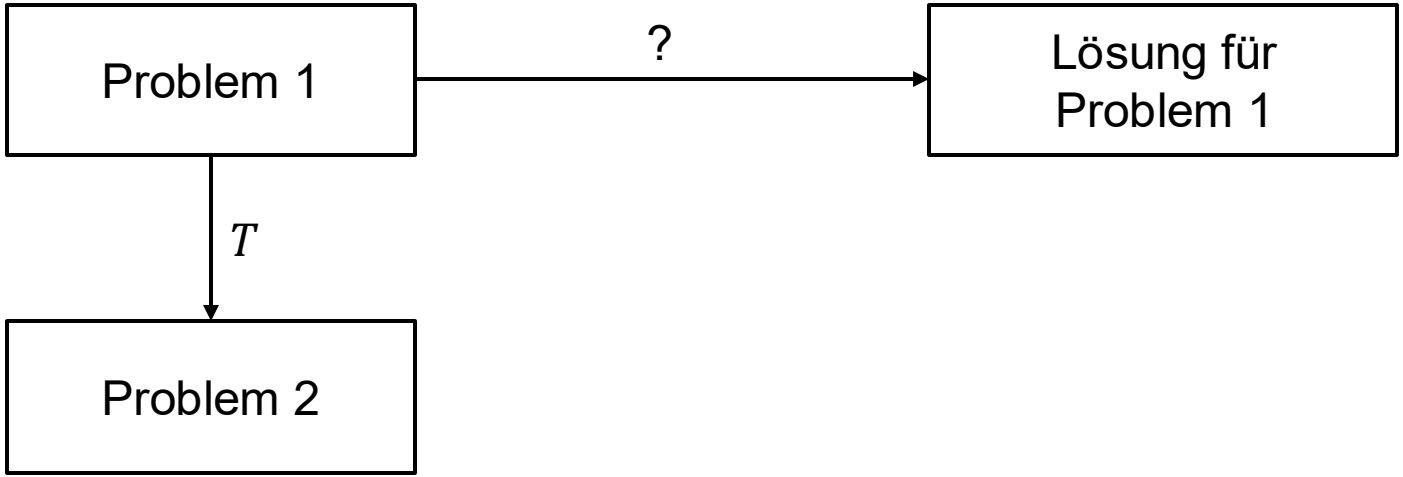
NP-schwer: Wenn das Problem in P liegt, gilt P=NP.

NP-vollständig: Problem liegt in NP und ist NP-schwer.

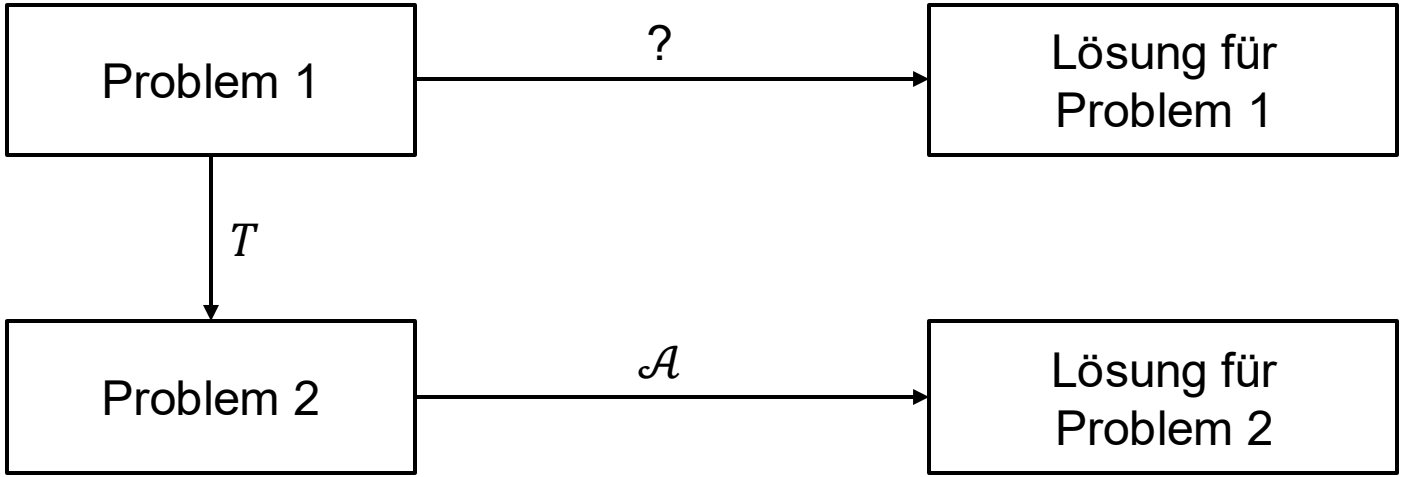
Reduktionen



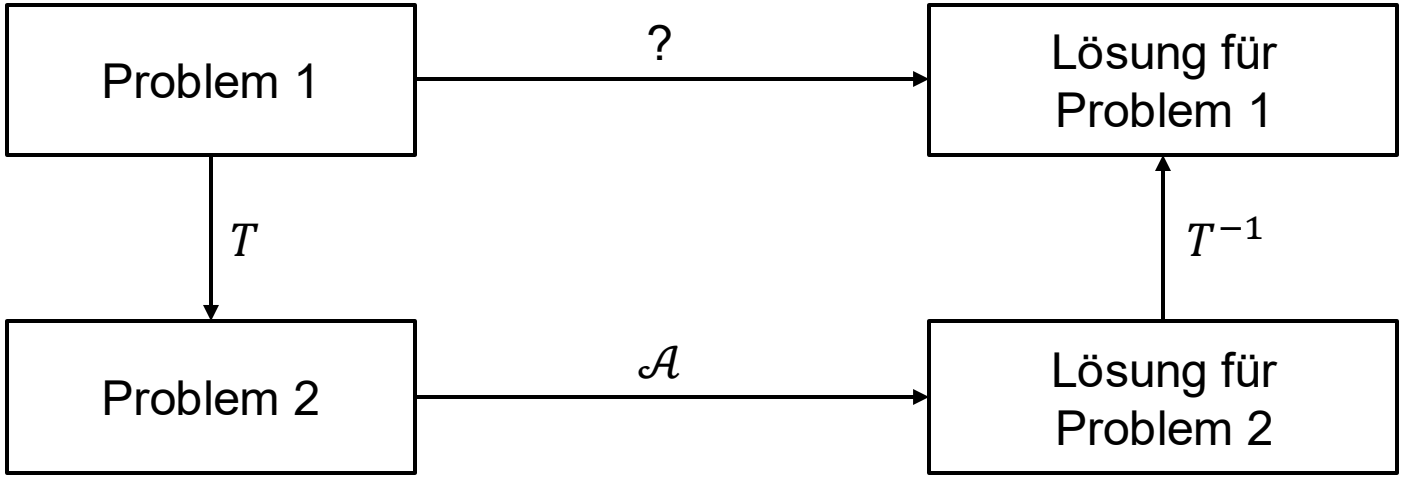
Reduktionen



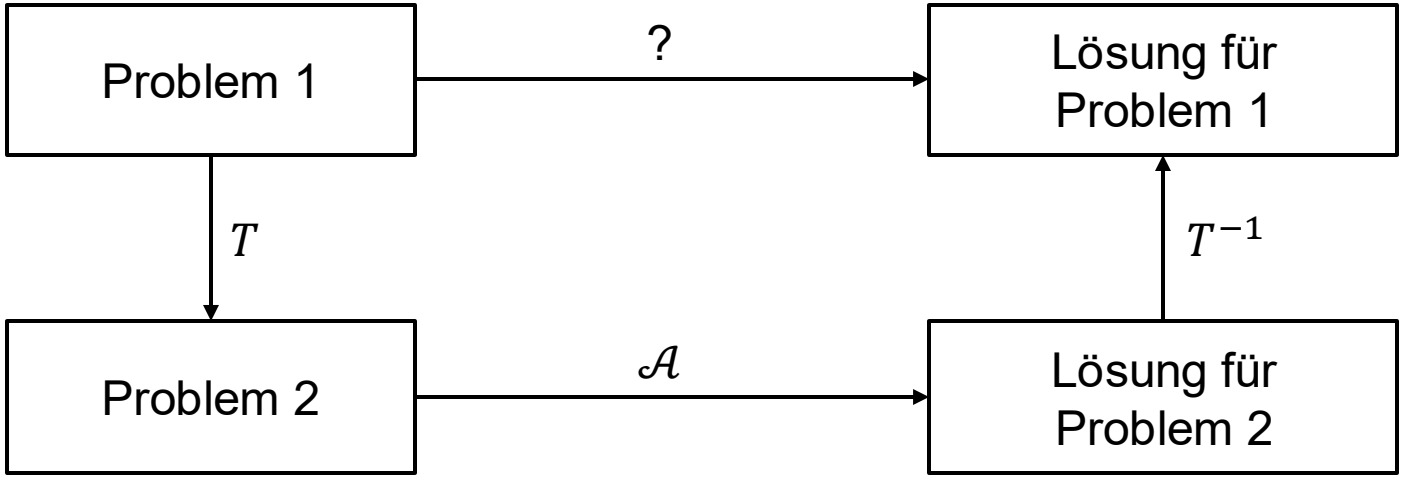
Reduktionen



Reduktionen

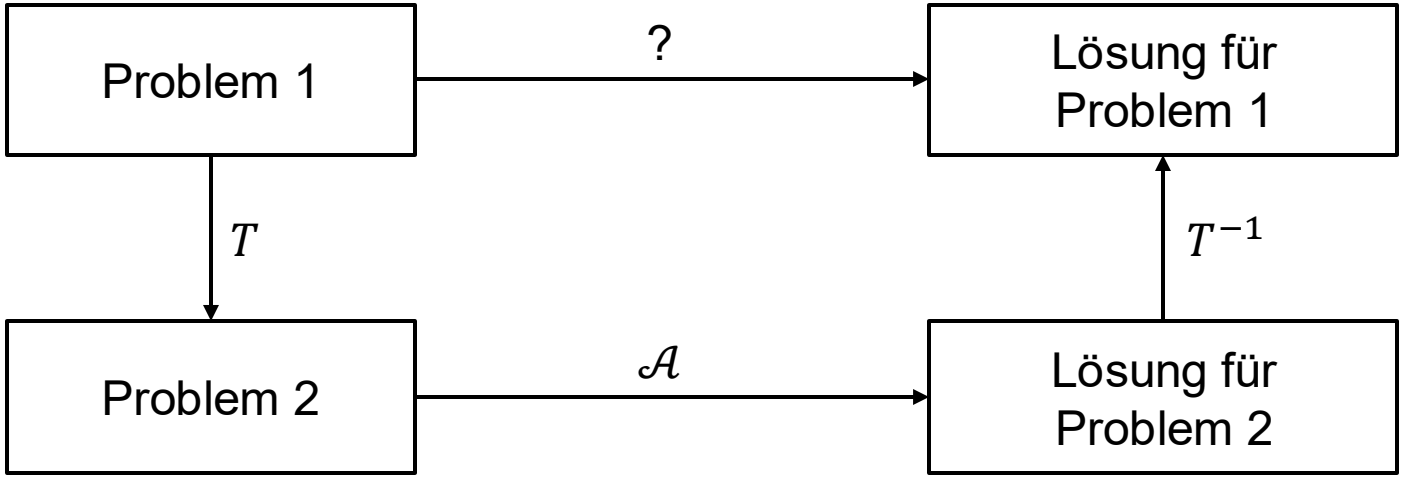


Reduktionen



Besitzen T, T^{-1} und $\mathcal{A}$ polynomielle Laufzeit,
kann Problem 1 in polynomieller Zeit gelöst werden.

Reduktionen



Ist Problem 1 NP-schwer und besitzen T und T^{-1} polynomielle Laufzeit, dann muss Problem 2 auch NP-schwer sein.

NP-Schwere

NP-Schwere

Ein Problem Π heißt *NP-schwer*, falls für jedes Problem $\Pi' \in$ NP eine Polynomialzeit-Reduktion von Π' auf Π existiert.

NP-Schwere

Ein Problem Π heißt *NP-schwer*, falls für jedes Problem $\Pi' \in$ NP eine Polynomialzeit-Reduktion von Π' auf Π existiert.

In der Literatur wird oft auch $\Pi' \leq_p \Pi$ geschrieben, wenn es eine Polynomialzeit-Reduktion von Π' auf Π gibt.

NP-Schwere

Ein Problem Π heißt *NP-schwer*, falls für jedes Problem $\Pi' \in$ NP eine Polynomialzeit-Reduktion von Π' auf Π existiert.

In der Literatur wird oft auch $\Pi' \leq_p \Pi$ geschrieben, wenn es eine Polynomialzeit-Reduktion von Π' auf Π gibt.

$\Pi' \leq_p \Pi$ heißt, man kann in polynomieller Zeit eine Instanz I' von Π' in eine Instanz I von Π transformieren, die eine Ja-Instanz ist gdw. I' eine Ja-Instanz ist.

NP-Schwere

Ein Problem Π heißt *NP-schwer*, falls für jedes Problem $\Pi' \in$ NP eine Polynomialzeit-Reduktion von Π' auf Π existiert.

In der Literatur wird oft auch $\Pi' \leq_p \Pi$ geschrieben, wenn es eine Polynomialzeit-Reduktion von Π' auf Π gibt.

$\Pi' \leq_p \Pi$ heißt, man kann in polynomieller Zeit eine Instanz I' von Π' in eine Instanz I von Π transformieren, die eine Ja-Instanz ist gdw. I' eine Ja-Instanz ist.

$\Pi' \leq_p \Pi$: „Π' ist höchstens so schwer wie Π“

NP-Schwere

Ein Problem Π heißt *NP-schwer*, falls für jedes Problem $\Pi' \in$ NP eine Polynomialzeit-Reduktion von Π' auf Π existiert.

In der Literatur wird oft auch $\Pi' \leq_p \Pi$ geschrieben, wenn es eine Polynomialzeit-Reduktion von Π' auf Π gibt.

$\Pi' \leq_p \Pi$ heißt, man kann in polynomieller Zeit eine Instanz I' von Π' in eine Instanz I von Π transformieren, die eine Ja-Instanz ist gdw. I' eine Ja-Instanz ist.

$\Pi' \leq_p \Pi$: „Π' ist höchstens so schwer wie Π“

Die Relation $\leq_p$ ist transitiv, da man polynomielle Reduktionen verketten kann.

NP-Schwere

Ein Problem Π heißt *NP-schwer*, falls für jedes Problem $\Pi' \in$ NP eine Polynomialzeit-Reduktion von Π' auf Π existiert.

In der Literatur wird oft auch $\Pi' \leq_p \Pi$ geschrieben, wenn es eine Polynomialzeit-Reduktion von Π' auf Π gibt.

$\Pi' \leq_p \Pi$ heißt, man kann in polynomieller Zeit eine Instanz I' von Π' in eine Instanz I von Π transformieren, die eine Ja-Instanz ist gdw. I' eine Ja-Instanz ist.

$\Pi' \leq_p \Pi$: „Π' ist höchstens so schwer wie Π“

Die Relation $\leq_p$ ist transitiv, da man polynomielle Reduktionen verketten kann.

Um zu zeigen, dass ein Problem Π NP-schwer ist, reicht es also, eine Reduktion **von einem** als NP-schwer bekannten Problem Π‘ auf Π durchzuführen.

Ein paar Probleme

Ein paar Probleme

3-SAT

Ein paar Probleme

3-SAT

Gegeben

Ein paar Probleme

3-SAT

Gegeben

Logische Formel φ in konjunktiver Normalform mit

Ein paar Probleme

3-SAT

$$\varphi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee x_4)$$

Gegeben

Logische Formel φ in konjunktiver Normalform mit

Ein paar Probleme

3-SAT

$$\varphi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee x_4)$$

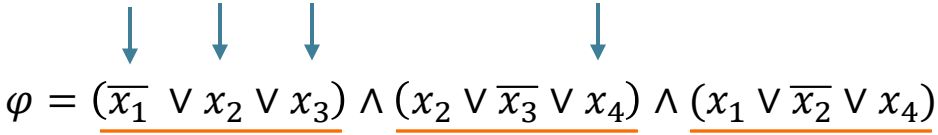
Gegeben

Logische Formel φ in konjunktiver Normalform mit

- m Klauseln

Ein paar Probleme

3-SAT



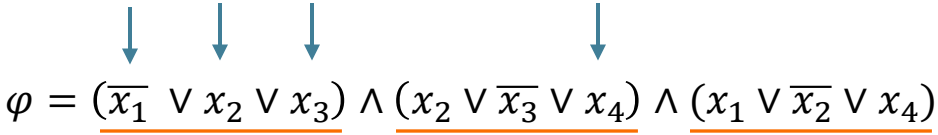
Gegeben

Logische Formel φ in konjunktiver Normalform mit

- m Klauseln
- n Variablen

Ein paar Probleme

3-SAT



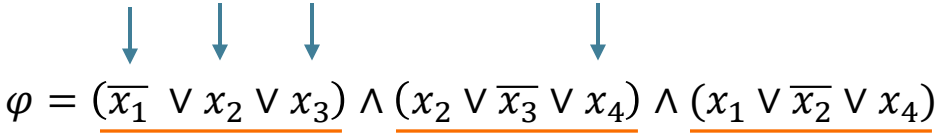
Gegeben

Logische Formel φ in konjunktiver Normalform mit

- m Klauseln
- n Variablen
- Genau drei Literale pro Klausel

Ein paar Probleme

3-SAT



Gegeben

Logische Formel φ in konjunktiver Normalform mit

- m Klauseln
- n Variablen
- Genau drei Literale pro Klausel

Frage

Gibt es eine φ erfüllende Belegung der Variablen?

Ein paar Probleme

3-SAT

Ein paar Probleme

3-SAT

Undirected Hamiltonian Cycle HC

Ein paar Probleme

3-SAT

Undirected Hamiltonian Cycle HC

Gegeben

Ungerichteter Graph $\mathrm{G} = (V, E)$

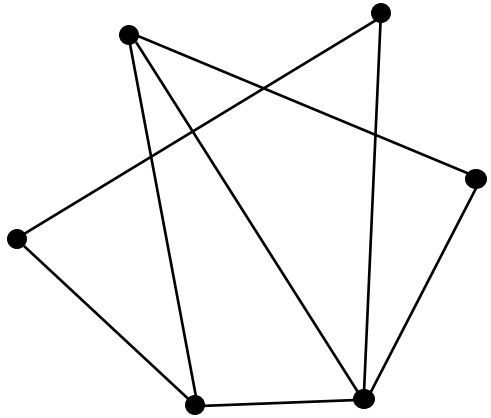
Ein paar Probleme

3-SAT

Undirected Hamiltonian Cycle HC

Gegeben

Ungerichteter Graph $\mathrm{G} = (V, E)$



Ein paar Probleme

3-SAT

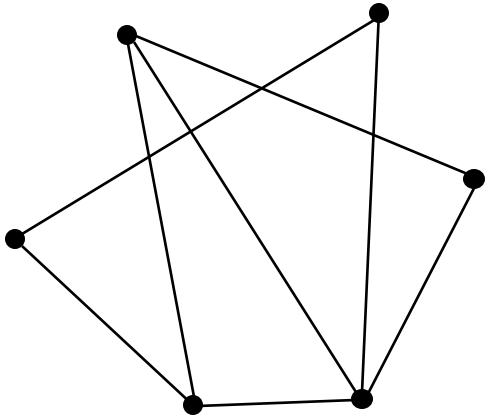
Undirected Hamiltonian Cycle HC

Gegeben

Ungerichteter Graph $\mathrm{G} = (V, E)$

Frage

Gibt es einen Hamiltonkreis in G?



Ein paar Probleme

3-SAT

HC

Ein paar Probleme

3-SAT

HC

Directed Hamiltonian Cycle DHC

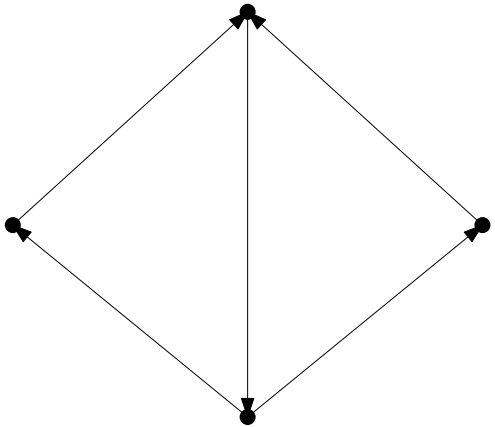
Ein paar Probleme

3-SAT **HC**

Directed Hamiltonian Cycle DHC

Gegeben

Gerichteter Graph $\mathrm{D} = (V, E)$



Ein paar Probleme

3-SAT **HC**

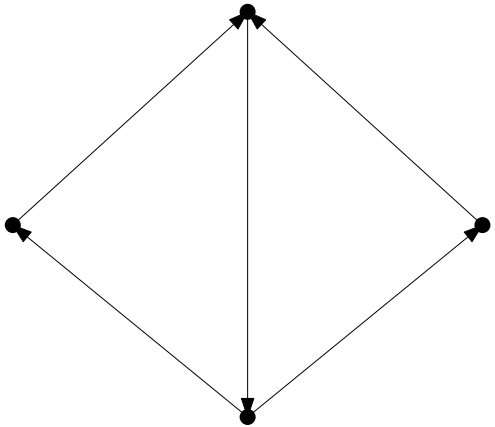
Directed Hamiltonian Cycle DHC

Gegeben

Gerichteter Graph $\mathrm{D} = (V, E)$

Frage

Gibt es einen gerichteten Hamiltonkreis in D?



Ein paar Probleme

3-SAT **DHC** **HC**

Ein paar Probleme

3-SAT **DHC** **HC**

Traveling Salesman Problem TSP

Ein paar Probleme

3-SAT **DHC** **HC**

Traveling Salesman Problem TSP

Gegeben

Vollsändiger Graph $G = (V, E)$ mit Kantenkosten $c: E \to \mathbb{R}^+$
und eine Zahl $k \in \mathbb{R}^+$

Ein paar Probleme

3-SAT **DHC** **HC**

Traveling Salesman Problem TSP

Gegeben

Vollsändiger Graph $\mathrm{G} = (V, E)$ mit Kantenkosten $c: E \to \mathbb{R}^+$ und eine Zahl $k \in \mathbb{R}^+$

Frage

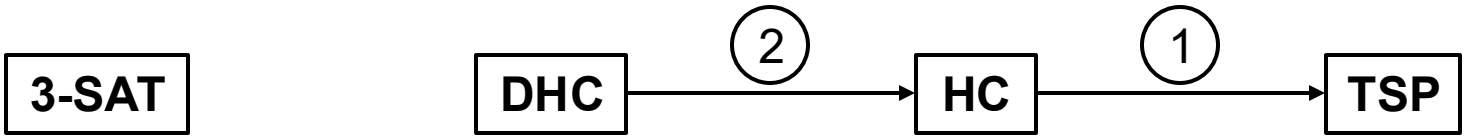
Gibt es eine Tour in G mit Kantenkosten maximal k?

Ein paar Probleme

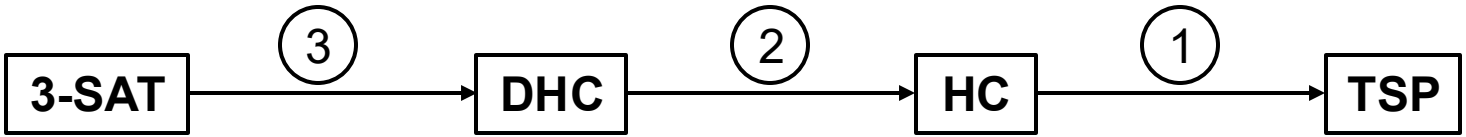
3-SAT **DHC** **HC** **TSP**

Ein paar Probleme

Ein paar Probleme

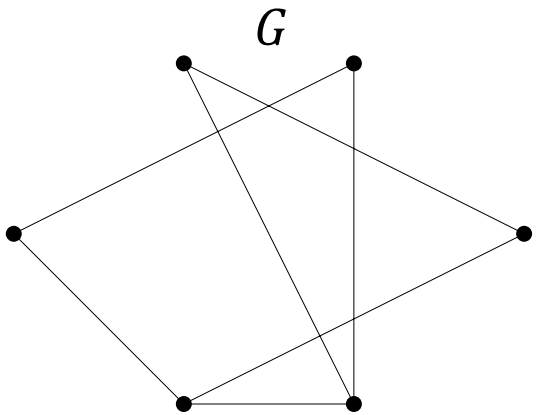


Ein paar Probleme

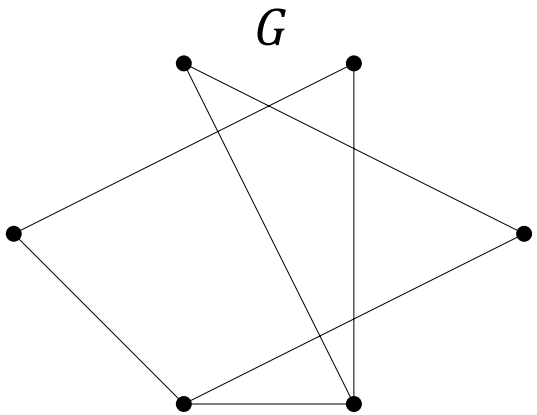


HC auf TSP

Reduktion von HC auf TSP

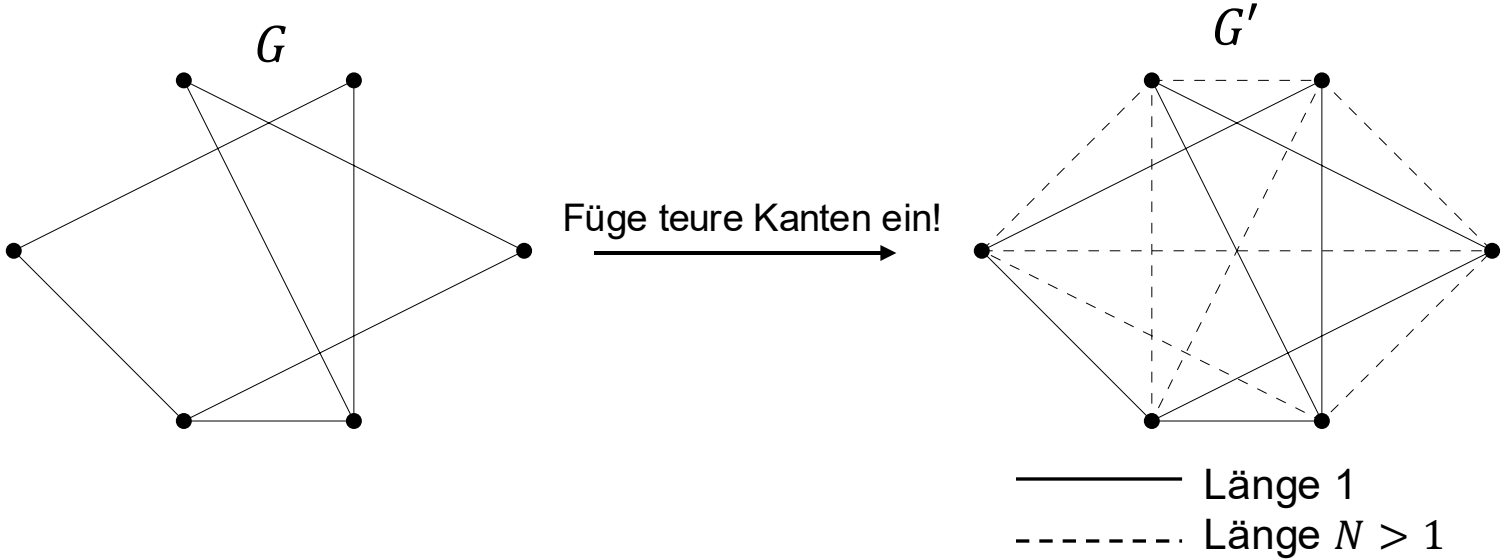


Reduktion von HC auf TSP

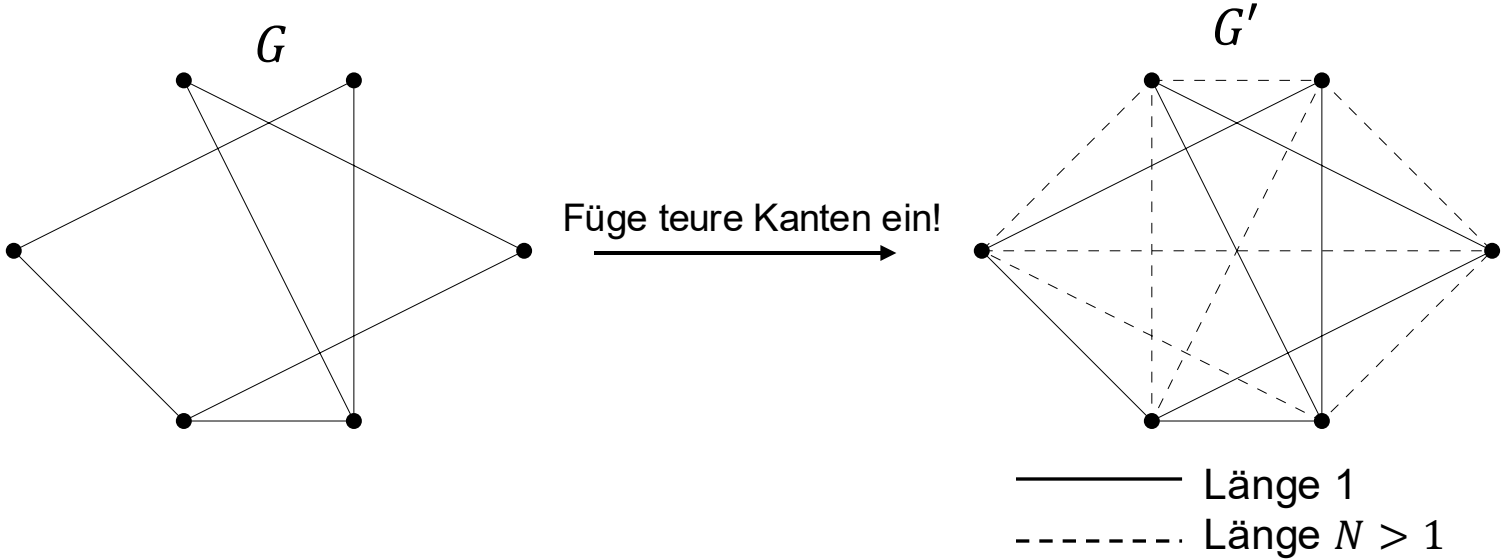


Füge teure Kanten ein!

Reduktion von HC auf TSP



Reduktion von HC auf TSP



Zu zeigen

G besitzt genau dann einen Hamitonkreis, wenn G' eine Tour der Länge $n := |V|$ besitzt.

Beweis Korrektheit

Beweis Korrektheit

„ ⇒ “

Beweis Korrektheit

„ ⇒ “

Wähle die gleichen Kanten von G in G'. Diese haben die Kosten n.

Beweis Korrektheit

„ ⇒ “

Wähle die gleichen Kanten von G in G'. Diese haben die Kosten n.

„ ⇐ “

Beweis Korrektheit

„ ⇒ “

Wähle die gleichen Kanten von G in G'. Diese haben die Kosten n.

„ ⇐ “

Besitzt G keinen Hamiltonkreis, so muss in G' mindestens eine Kante mit Gewicht N benutzt werden. Somit hat die Tour ein Gewicht von mindestens $n - 1 + N > n$, da $N > 1$.

Beweis Korrektheit

„ ⇒ “

Wähle die gleichen Kanten von G in G'. Diese haben die Kosten n.

„ ⇐ “

Besitzt G keinen Hamiltonkreis, so muss in G' mindestens eine Kante mit Gewicht N benutzt werden. Somit hat die Tour ein Gewicht von mindestens $n - 1 + N > n$, da $N > 1$.

Laufzeit

Beweis Korrektheit

„ ⇒ “

Wähle die gleichen Kanten von G in G'. Diese haben die Kosten n.

„ ⇐ “

Besitzt G keinen Hamiltonkreis, so muss in G' mindestens eine Kante mit Gewicht N benutzt werden. Somit hat die Tour ein Gewicht von mindestens $n - 1 + N > n$, da $N > 1$.

Laufzeit

Es müssen $O(n^2)$ Kanten hinzugefügt werden.

Beweis Korrektheit

„ ⇒ “

Wähle die gleichen Kanten von G in G'. Diese haben die Kosten n.

„ ⇐ “

Besitzt G keinen Hamiltonkreis, so muss in G' mindestens eine Kante mit Gewicht N benutzt werden. Somit hat die Tour ein Gewicht von mindestens $n - 1 + N > n$, da $N > 1$.

Laufzeit

Es müssen $O(n^2)$ Kanten hinzugefügt werden.

Alle $O(n^2)$ Kanten müssen mit Kosten versehen werden.

Beweis Korrektheit

„ ⇒ “

Wähle die gleichen Kanten von G in G'. Diese haben die Kosten n.

„ ⇐ “

Besitzt G keinen Hamiltonkreis, so muss in G' mindestens eine Kante mit Gewicht N benutzt werden. Somit hat die Tour ein Gewicht von mindestens $n - 1 + N > n$, da $N > 1$.

Laufzeit

Es müssen $O(n^2)$ Kanten hinzugefügt werden.

Alle $O(n^2)$ Kanten müssen mit Kosten versehen werden.

Insgesamt also eine Laufzeit von $O(n^2)$.

Beweis Korrektheit

„ ⇒ “

Wähle die gleichen Kanten von G in G'. Diese haben die Kosten n.

„ ⇐ “

Besitzt G keinen Hamiltonkreis, so muss in G' mindestens eine Kante mit Gewicht N benutzt werden. Somit hat die Tour ein Gewicht von mindestens $n - 1 + N > n$, da $N > 1$.

Laufzeit

Es müssen $O(n^2)$ Kanten hinzugefügt werden.

Alle $O(n^2)$ Kanten müssen mit Kosten versehen werden.

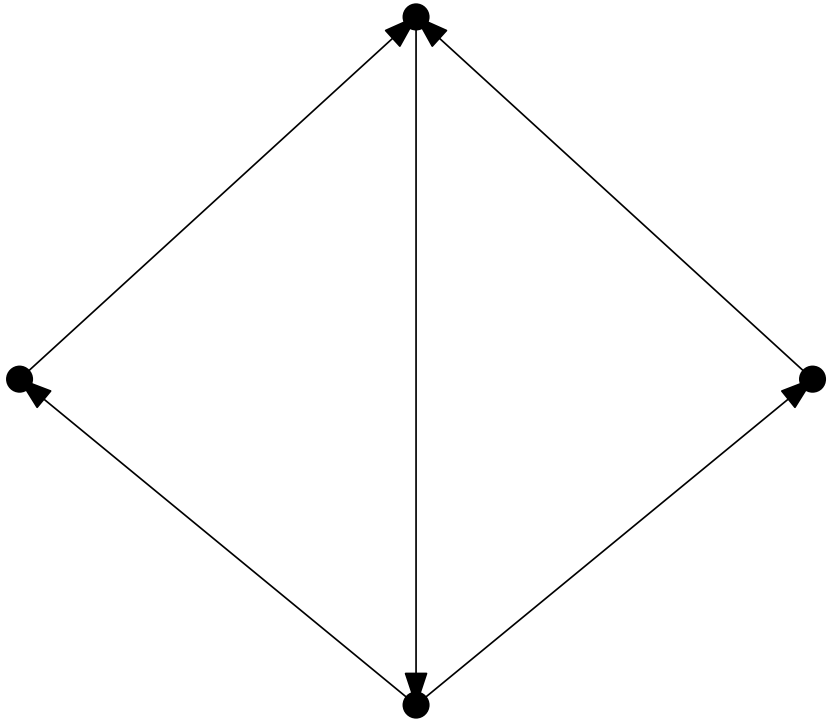
Insgesamt also eine Laufzeit von $O(n^2)$.

Konsequenz

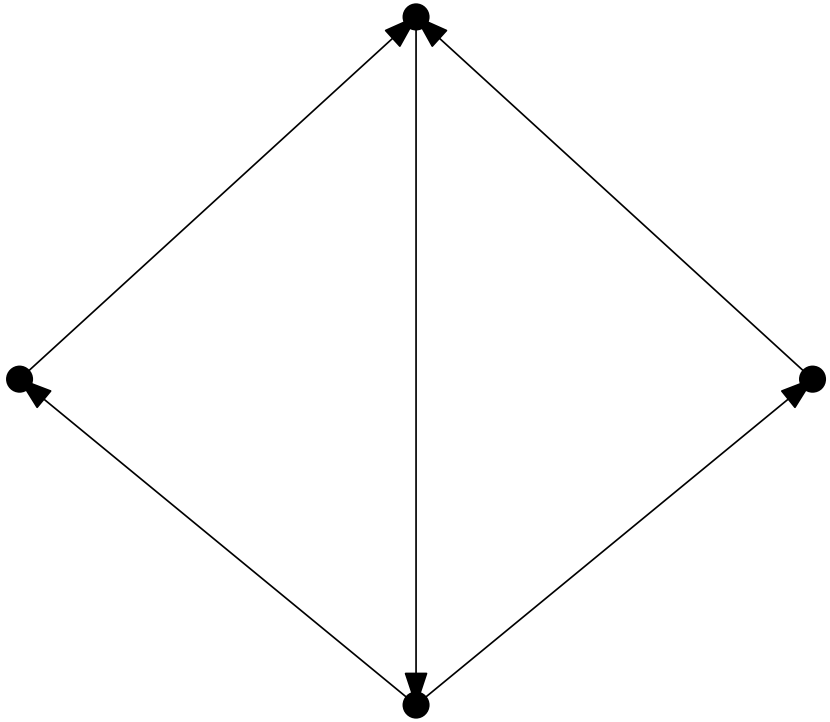
TSP kann nicht approximiert werden, es sei denn P = NP.

DHC auf HC

Reduktion von DHC auf HC

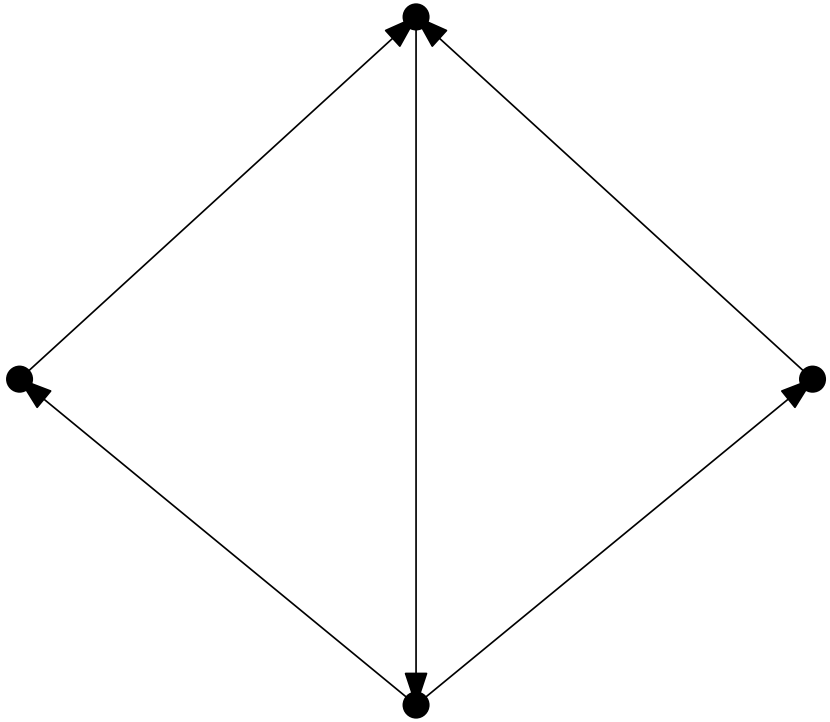


Reduktion von DHC auf HC



Wie kann man garantieren, dass im ungerichteten Graphen…

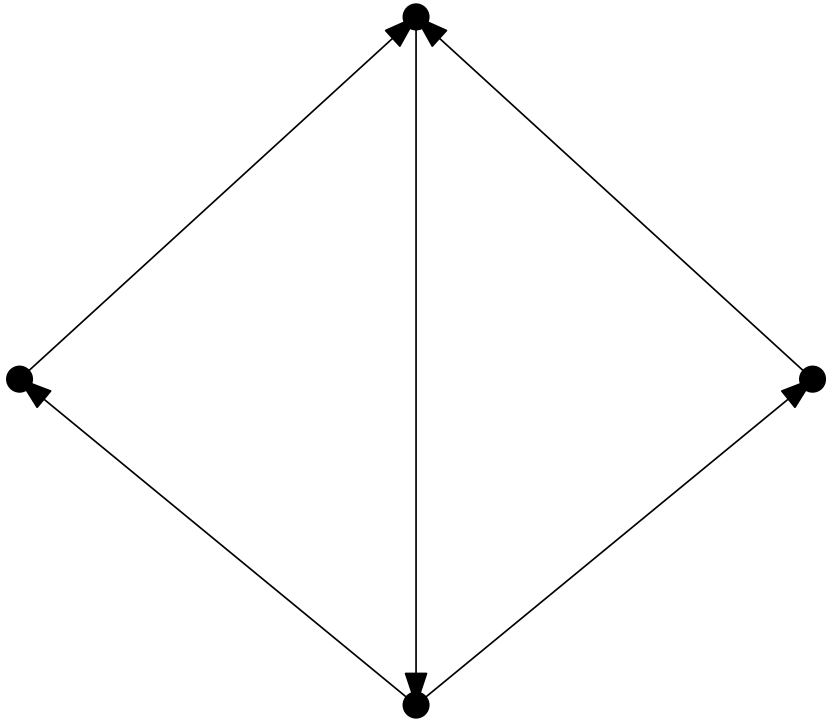
Reduktion von DHC auf HC



Wie kann man garantieren, dass im ungerichteten Graphen…

…nur eine der ursprünglich *eingehenden* Kanten verwendet wird?

Reduktion von DHC auf HC

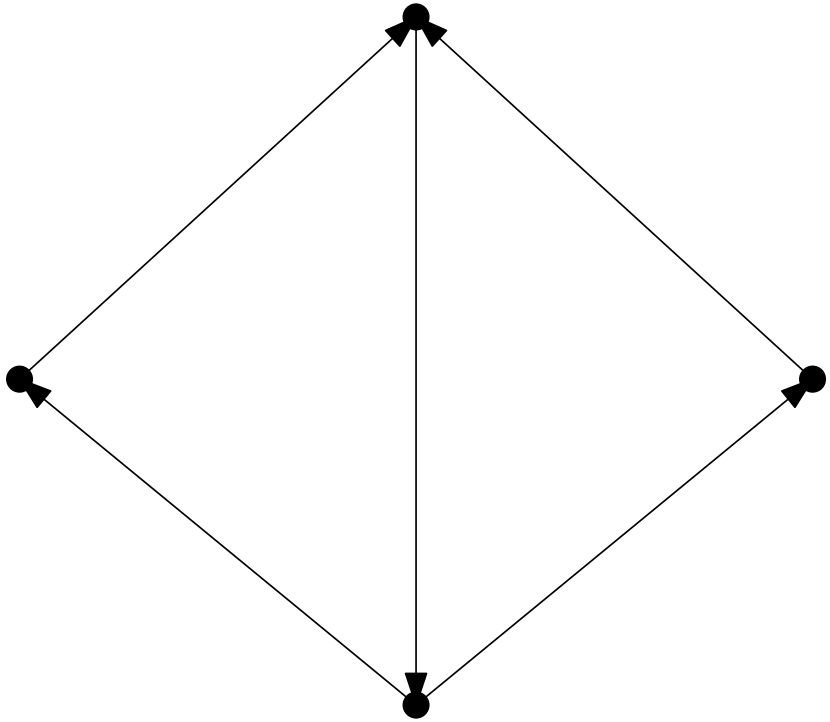


Wie kann man garantieren, dass im ungerichteten Graphen…

…nur eine der ursprünglich *eingehenden* Kanten verwendet wird?

…nur eine der ursprünglich *ausgehenden* Kanten verwendet wird?

Reduktion von DHC auf HC



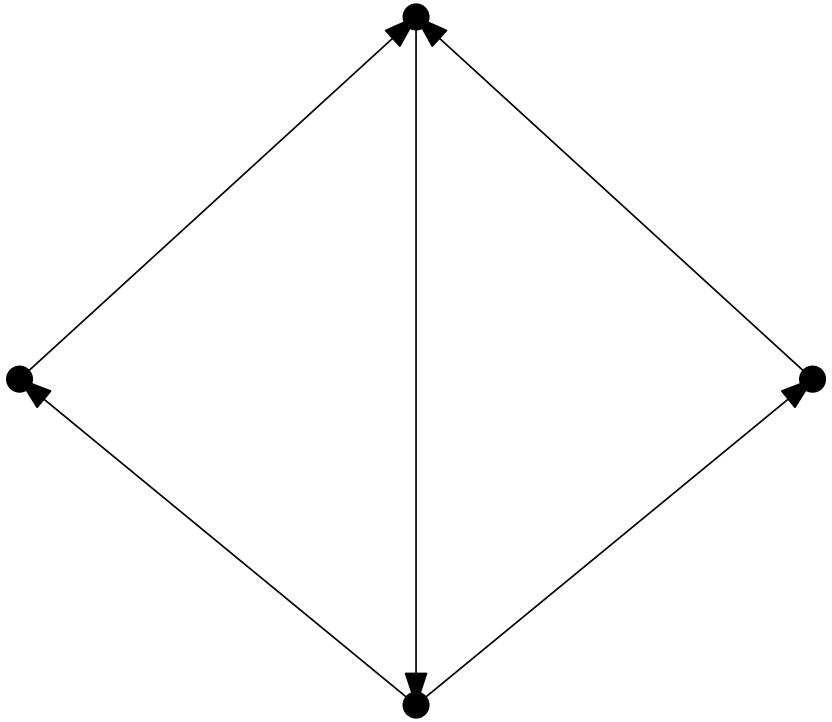
Wie kann man garantieren, dass im ungerichteten Graphen…

…nur eine der ursprünglich *eingehenden* Kanten verwendet wird?

…nur eine der ursprünglich *ausgehenden* Kanten verwendet wird?

Idee: Teile Knoten auf!

Reduktion von DHC auf HC

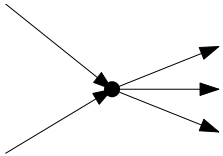


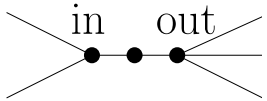
Wie kann man garantieren, dass im ungerichteten Graphen…

…nur eine der ursprünglich *eingehenden* Kanten verwendet wird?

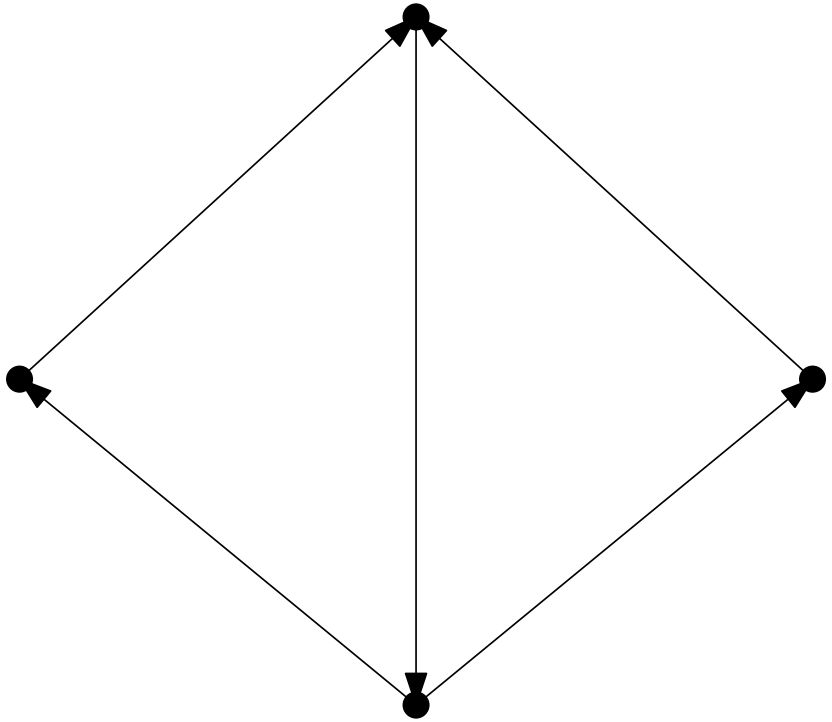
…nur eine der ursprünglich *ausgehenden* Kanten verwendet wird?

Idee: Teile Knoten auf!





Reduktion von DHC auf HC

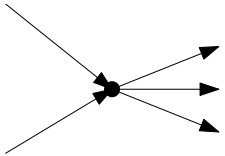


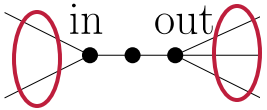
Wie kann man garantieren, dass im ungerichteten Graphen…

…nur eine der ursprünglich *eingehenden* Kanten verwendet wird?

…nur eine der ursprünglich *ausgehenden* Kanten verwendet wird?

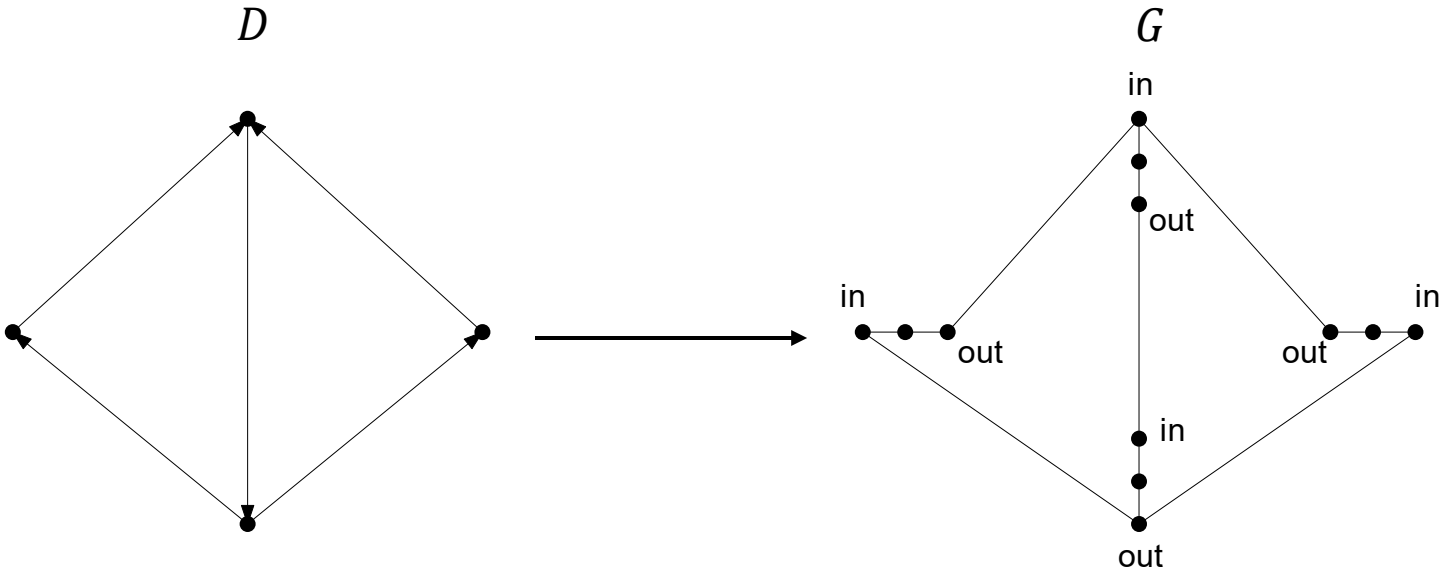
Idee: Teile Knoten auf!



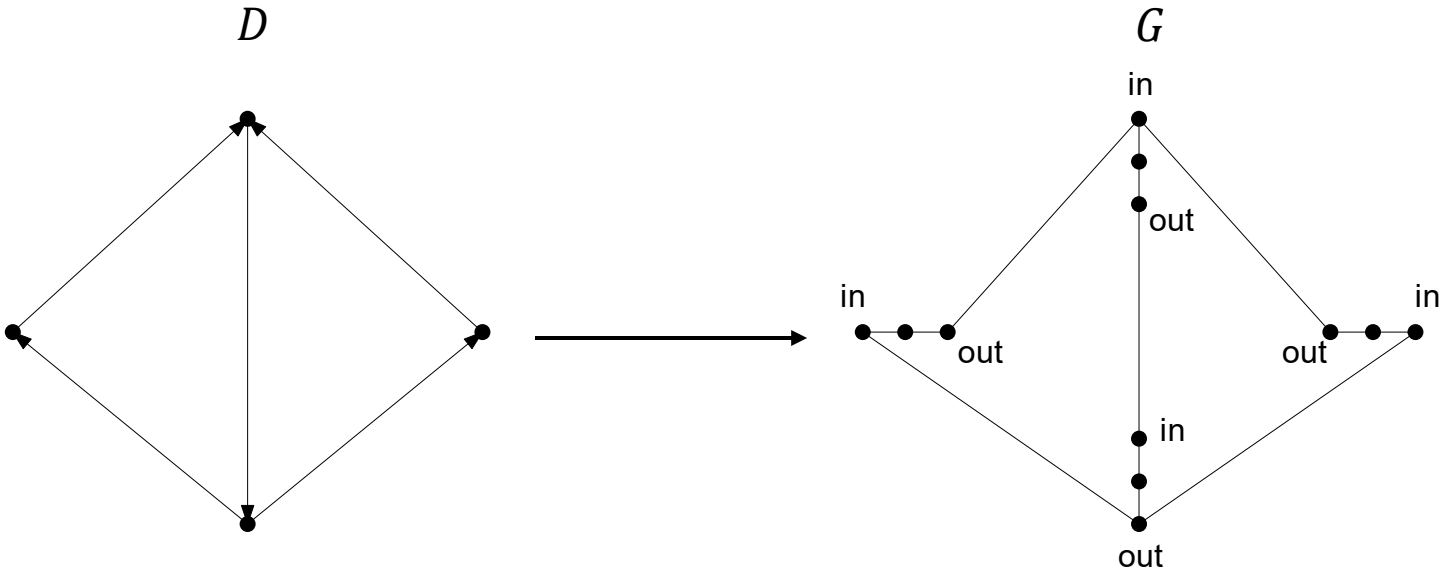


Jeweils nur eine möglich!

Reduktion von DHC auf HC



Reduktion von DHC auf HC



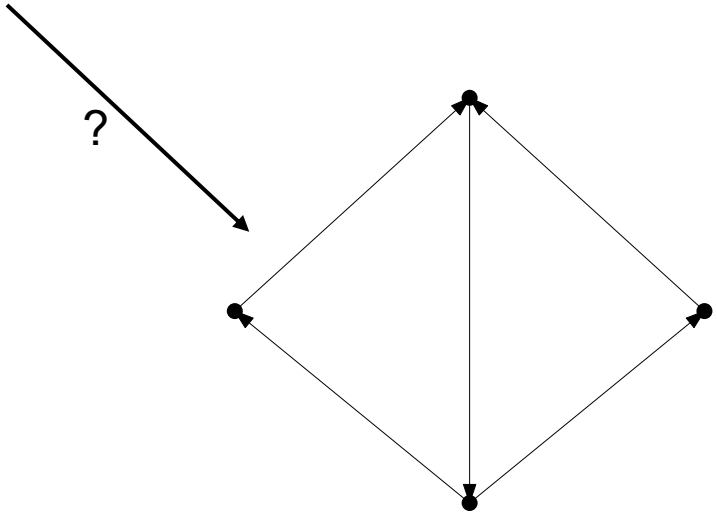
Zu zeigen

D besitzt genau dann einen gerichteten Hamiltonkreis, wenn G einen Hamiltonkreis besitzt.

3SAT auf DHC

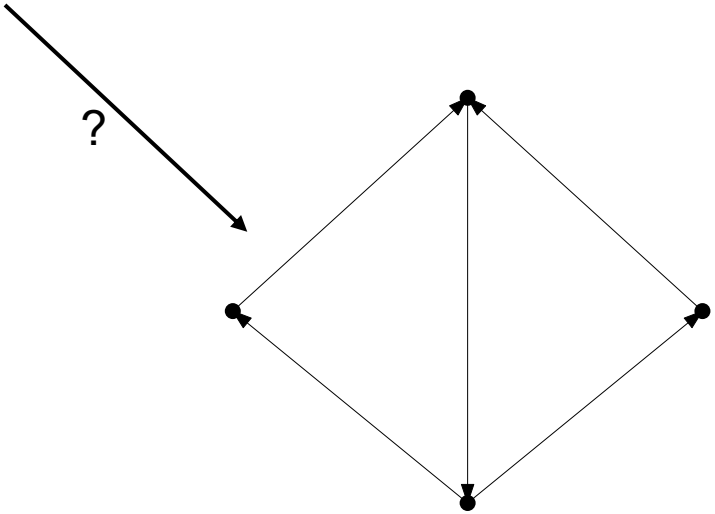
Reduktion 3SAT auf DHC

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3 \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee x_4)$$



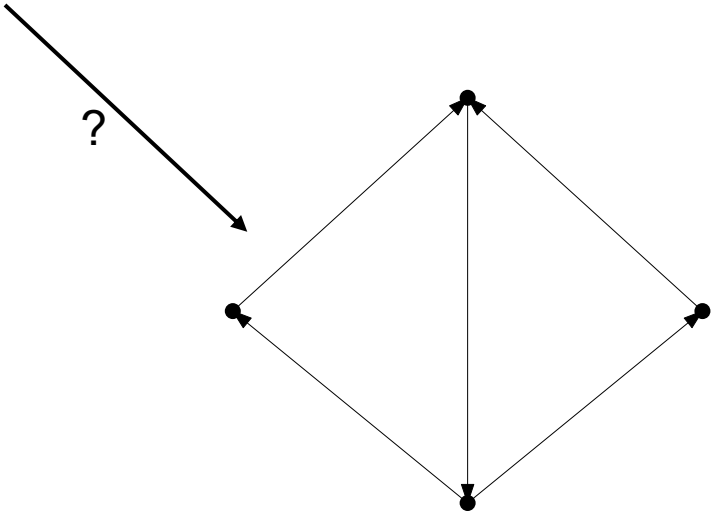
Reduktion 3SAT auf DHC

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3 \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee x_4)$$



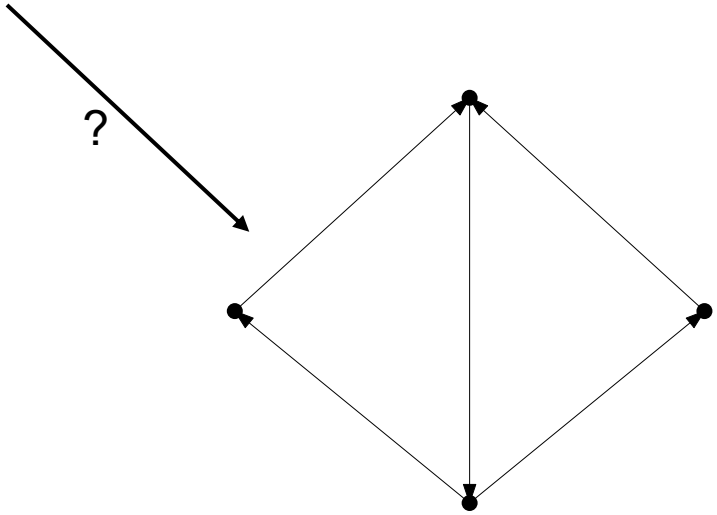
Reduktion 3SAT auf DHC

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3 \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee x_4)$$



Reduktion 3SAT auf DHC

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3 \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee x_4)$$



Variablen-Gadgets

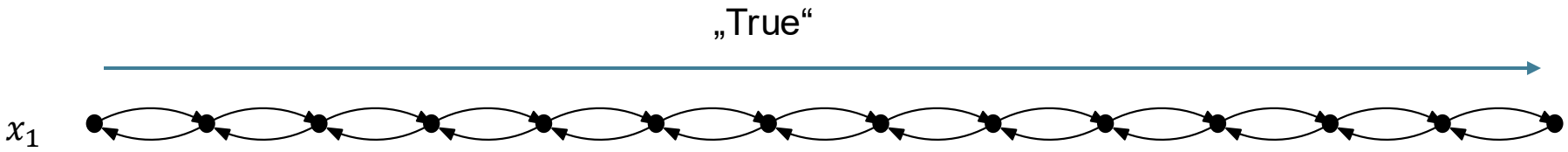
Variablen-Gadgets

x_1

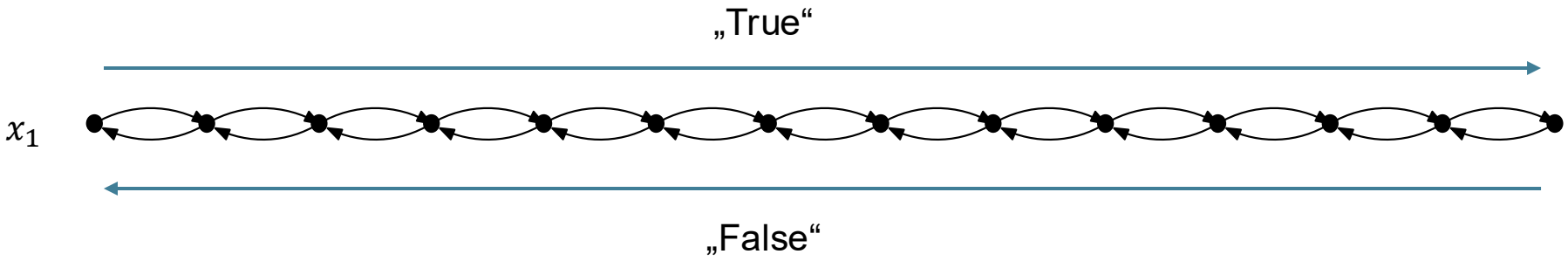
Variablen-Gadgets

x_1

Variablen-Gadgets



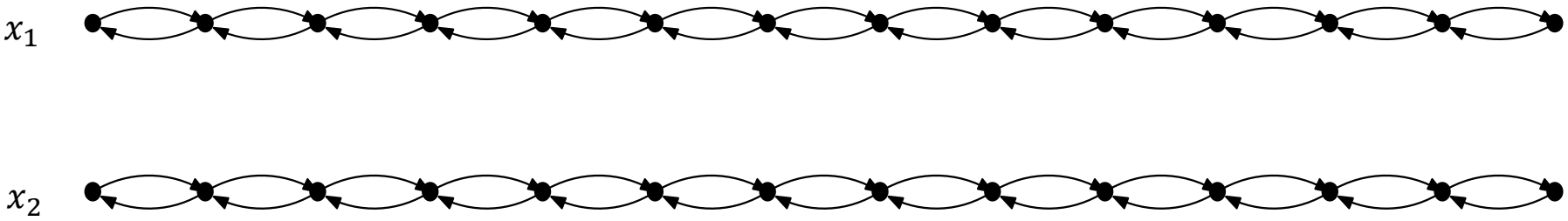
Variablen-Gadgets



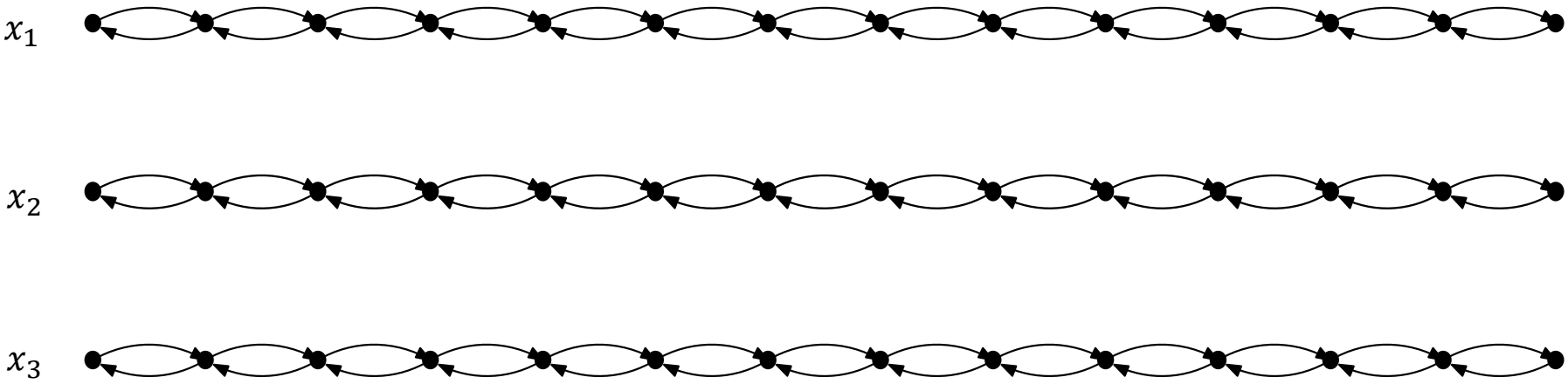
Variablen-Gadgets

x_1

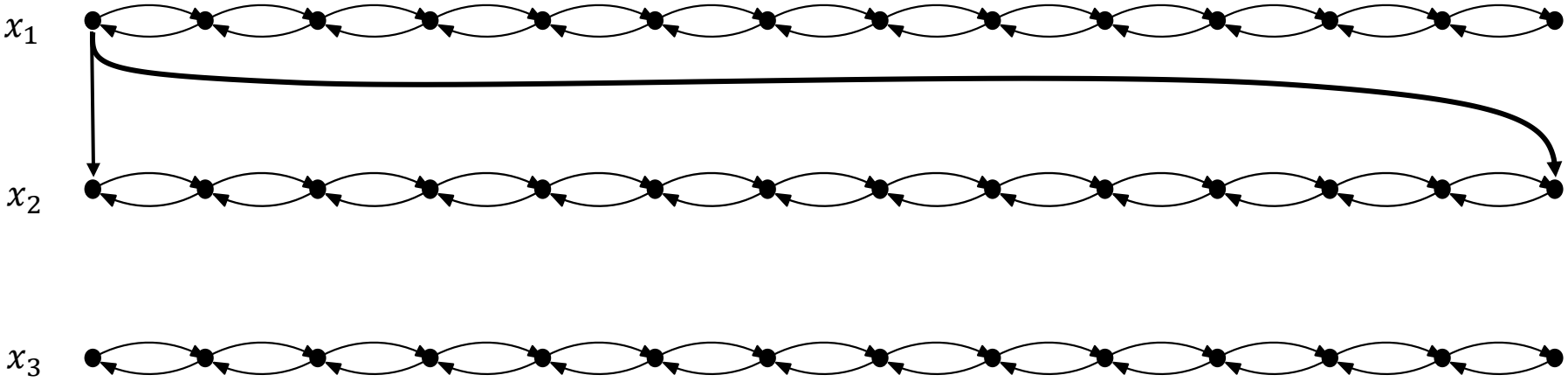
Variablen-Gadgets



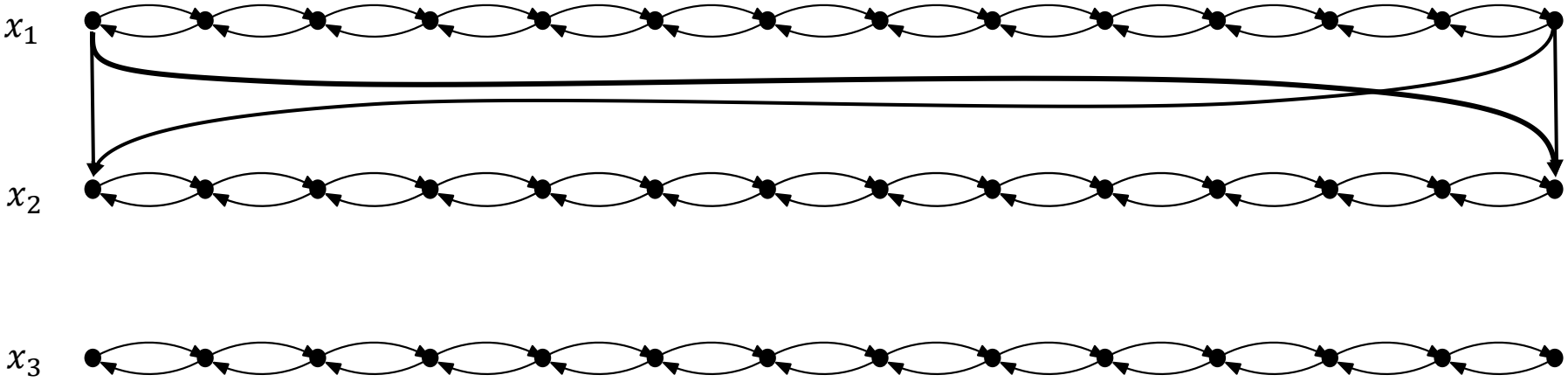
Variablen-Gadgets



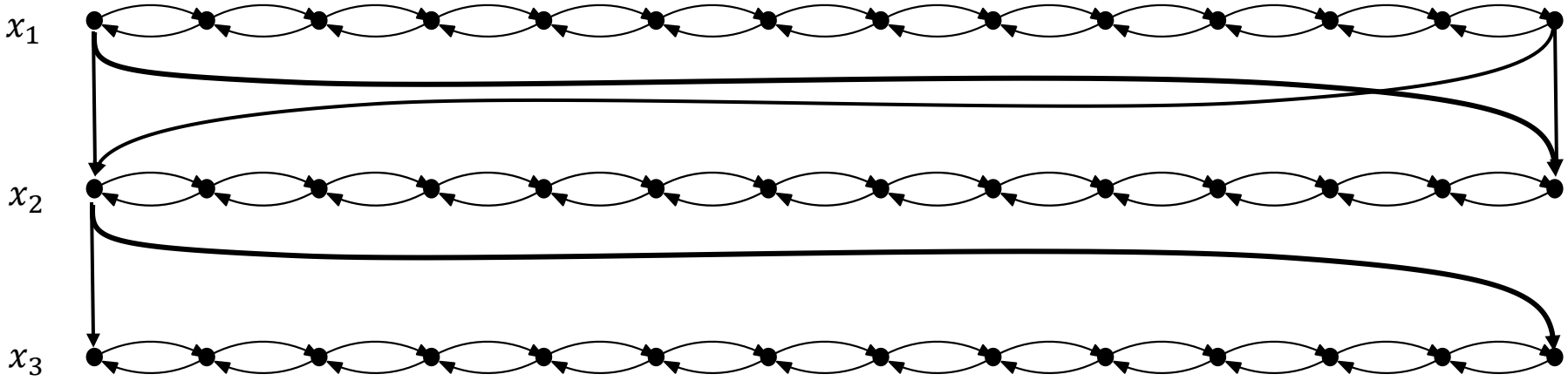
Variablen-Gadgets



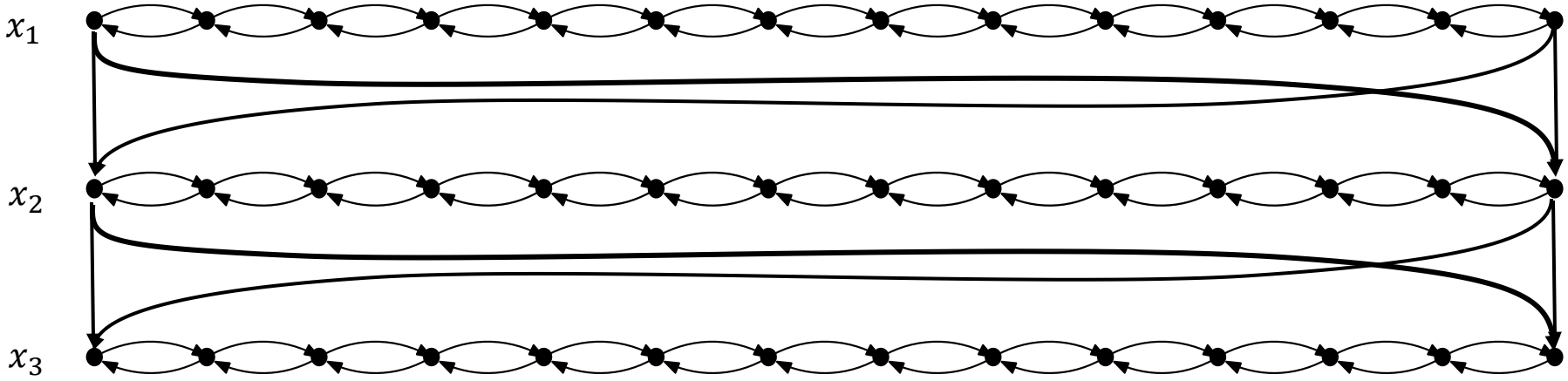
Variablen-Gadgets



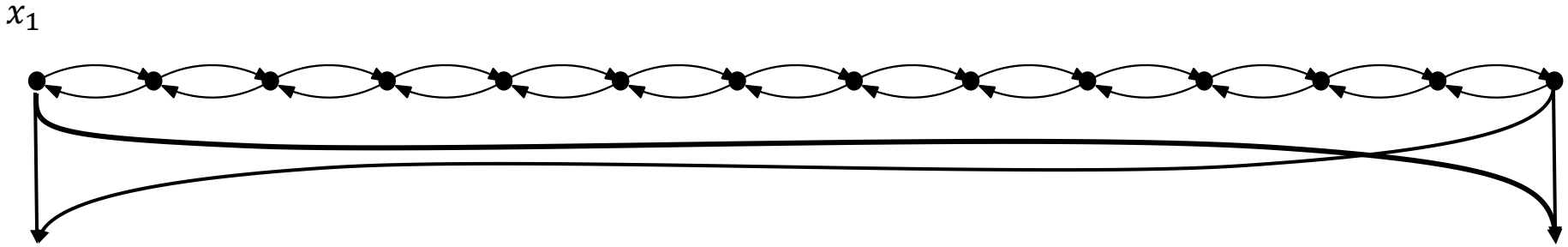
Variablen-Gadgets



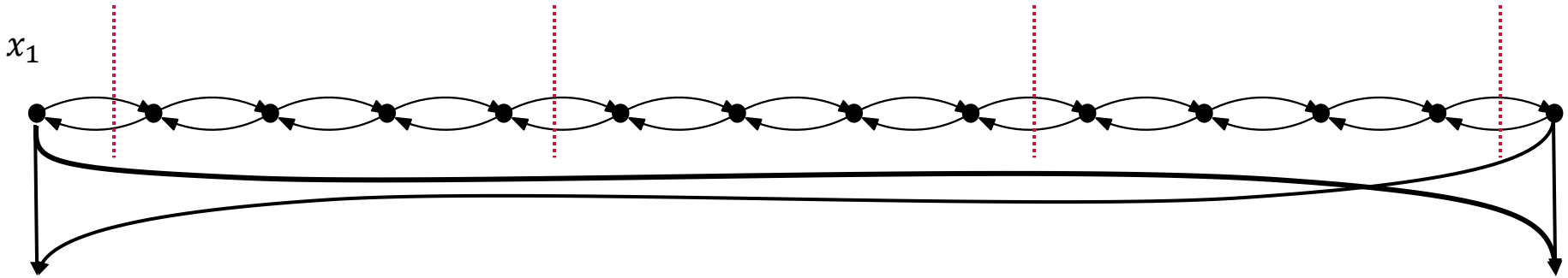
Variablen-Gadgets



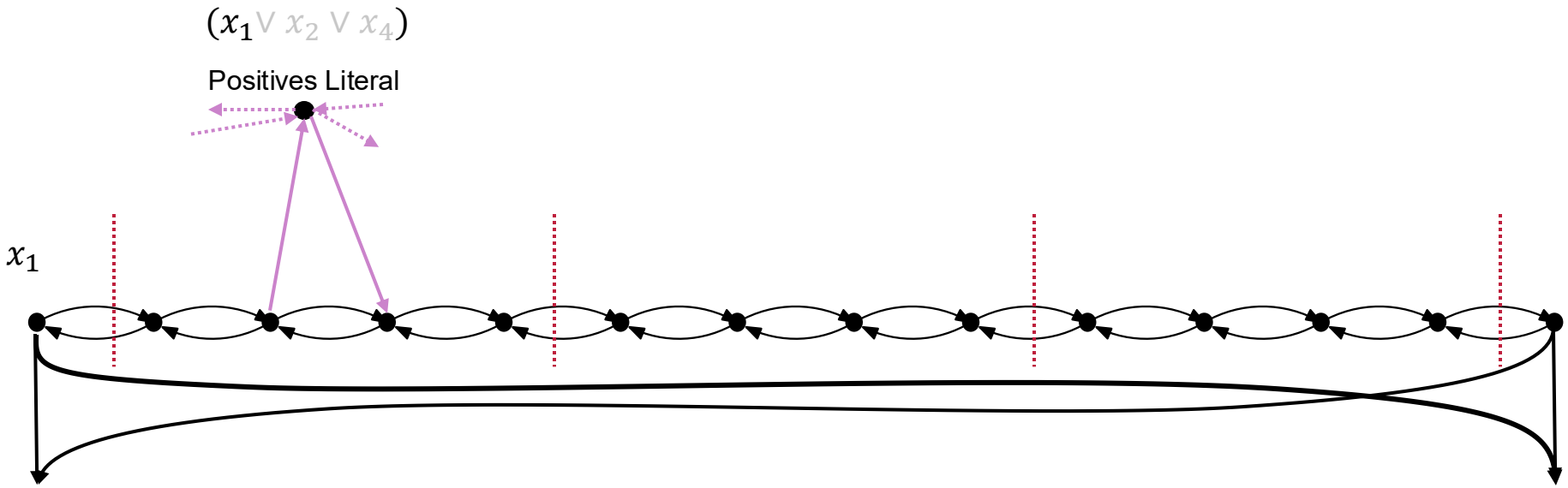
Klausel-Gadgets



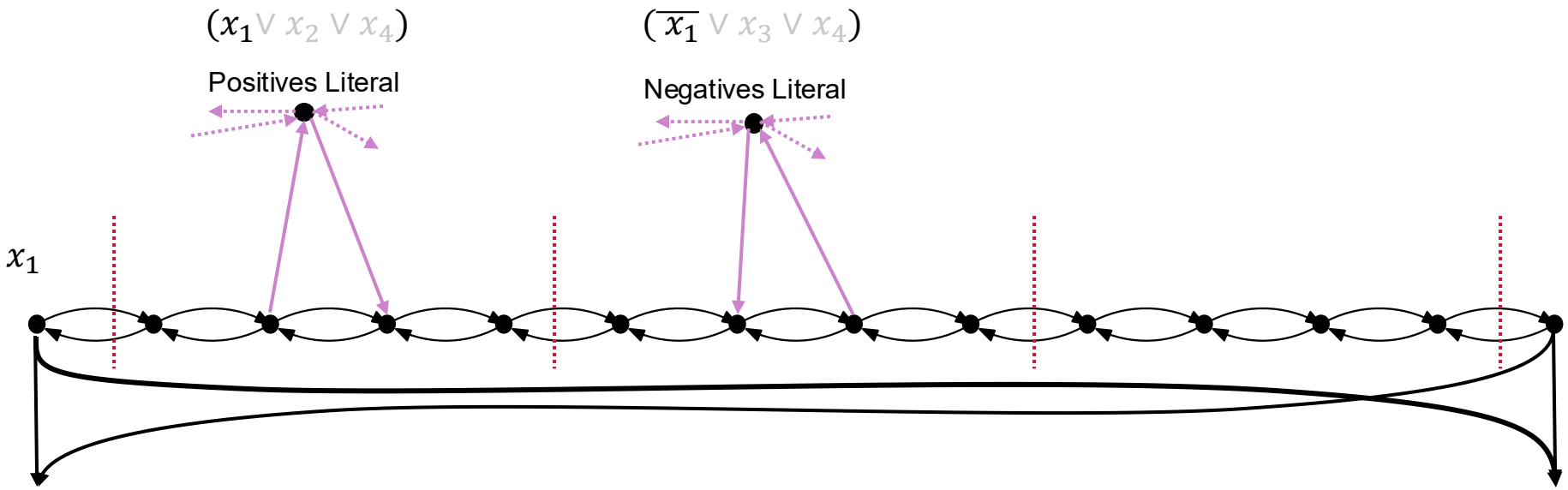
Klausel-Gadgets



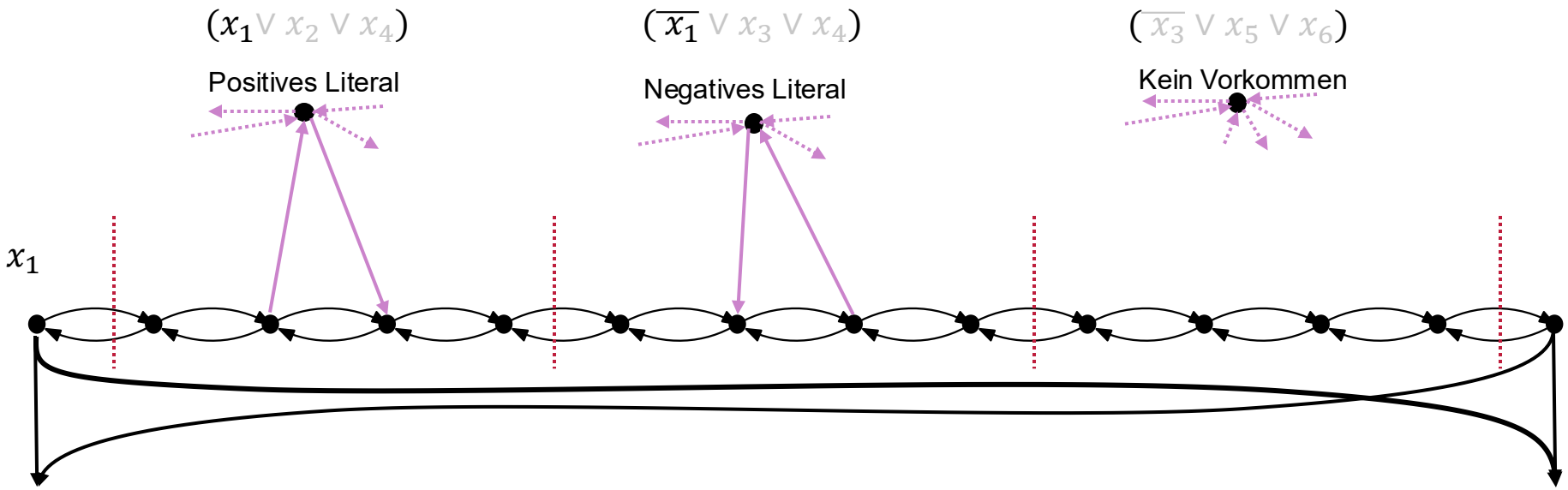
Klausel-Gadgets



Klausel-Gadgets



Klausel-Gadgets

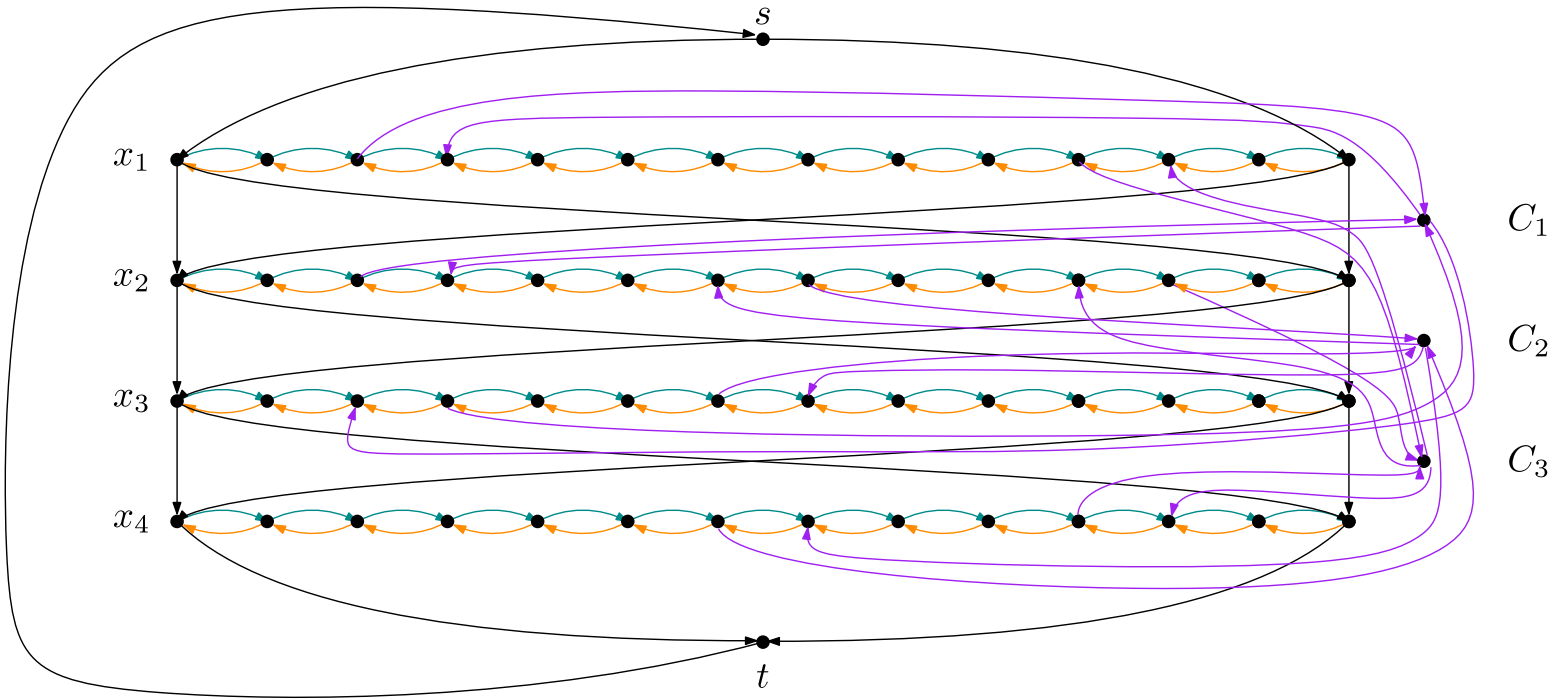


Beispiel der Reduktion

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3 \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee x_4)$$

Beispiel der Reduktion

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3 \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee x_4)$$



Korrektheit

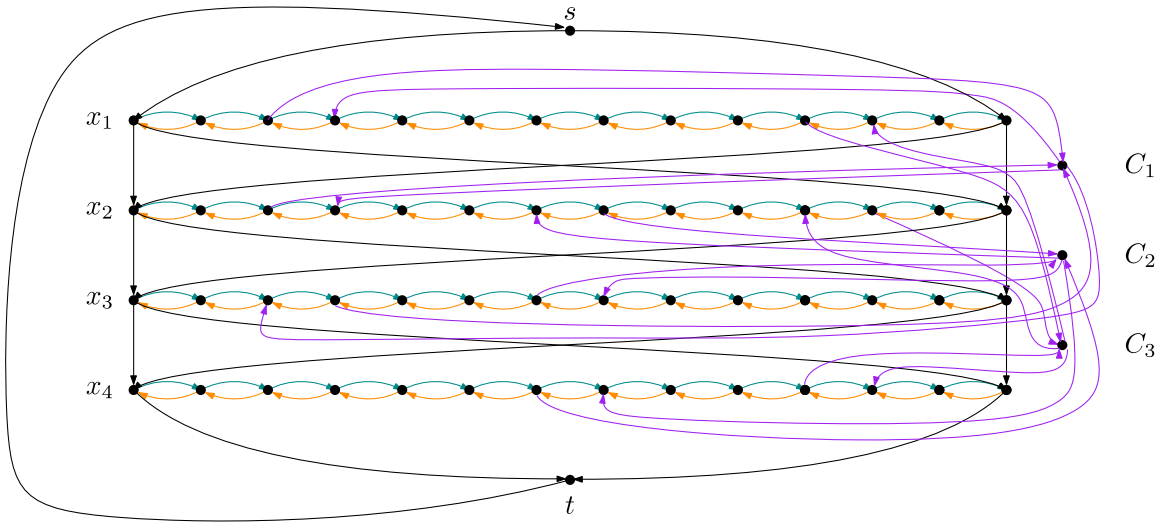
Korrektheit

„Wenn die Formel erfüllbar ist, dann gibt es einen Hamiltonkreis.“

Korrektheit

„Wenn die Formel erfüllbar ist, dann gibt es einen Hamiltonkreis.“

Laufe die Variablen Gadgets in der richtigen Richtung ab und laufe zwischendurch die Klauseln ab. (Nach Konstruktion möglich)



Korrektheit

„Wenn die Formel erfüllbar ist, dann gibt es einen Hamiltonkreis.“

Laufe die Variablen Gadgets in der richtigen Richtung ab und laufe zwischendurch die Klauseln ab. (Nach Konstruktion möglich)

„Wenn es einen Hamiltonkreis gibt, dann ist die Formel erfüllbar.“

Korrektheit

„Wenn die Formel erfüllbar ist, dann gibt es einen Hamiltonkreis.“

Laufe die Variablen Gadgets in der richtigen Richtung ab und laufe zwischendurch die Klauseln ab. (Nach Konstruktion möglich)

„Wenn es einen Hamiltonkreis gibt, dann ist die Formel erfüllbar.“

Dazu müssen wir zeigen:

Korrektheit

„Wenn die Formel erfüllbar ist, dann gibt es einen Hamiltonkreis.“

Laufe die Variablen Gadgets in der richtigen Richtung ab und laufe zwischendurch die Klauseln ab. (Nach Konstruktion möglich)

„Wenn es einen Hamiltonkreis gibt, dann ist die Formel erfüllbar.“

Dazu müssen wir zeigen:

1. Geht man zu einer Klausel, muss man zur selben Variable zurück.

Korrektheit

„Wenn die Formel erfüllbar ist, dann gibt es einen Hamiltonkreis.“

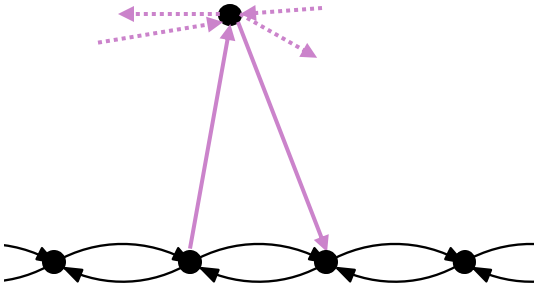
Laufe die Variablen Gadgets in der richtigen Richtung ab und laufe zwischendurch die Klauseln ab. (Nach Konstruktion möglich)

„Wenn es einen Hamiltonkreis gibt, dann ist die Formel erfüllbar.“

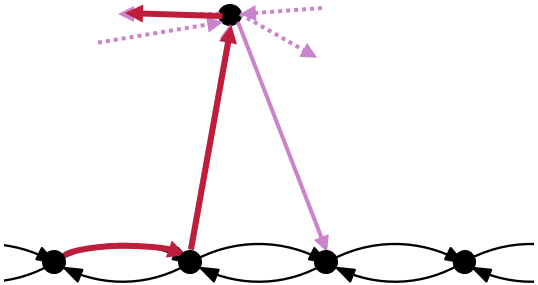
Dazu müssen wir zeigen:

1. Geht man zu einer Klausel, muss man zur selben Variable zurück.
2. Man darf nur Klauseln ablaufen, wenn die richtige Richtung gewählt wurde.

Man muss wieder zurück…

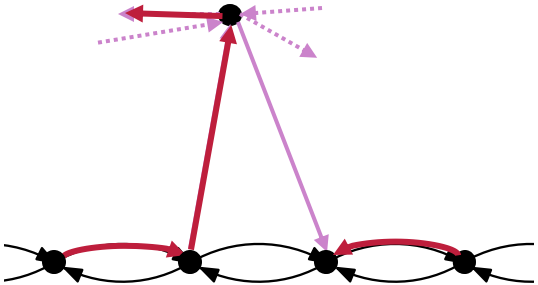


Man muss wieder zurück…



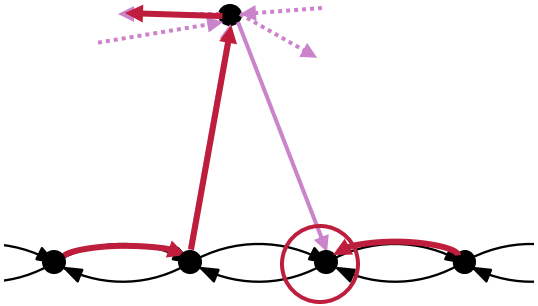
Annahme: Man geht nicht zurück.

Man muss wieder zurück…



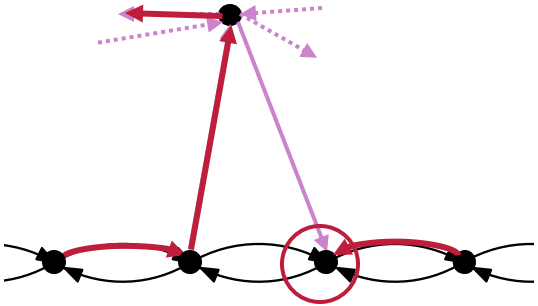
Annahme: Man geht nicht zurück.

Man muss wieder zurück…



Annahme: Man geht nicht zurück.

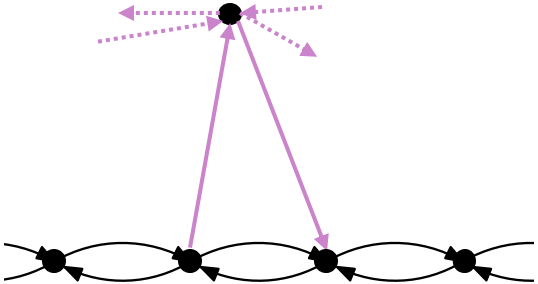
Man muss wieder zurück…



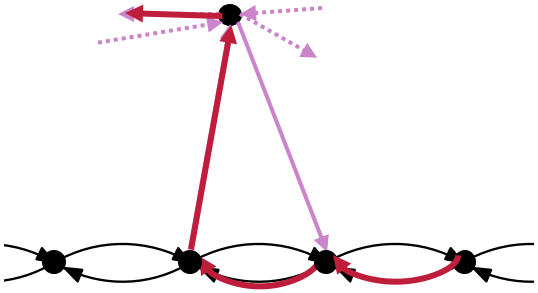
Man kommt nicht mehr weg…

Annahme: Man geht nicht zurück.

Man muss richtig rum laufen…

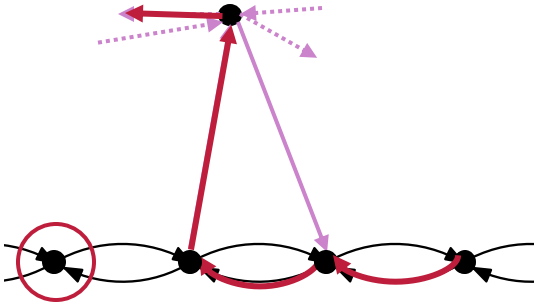


Man muss richtig rum laufen…



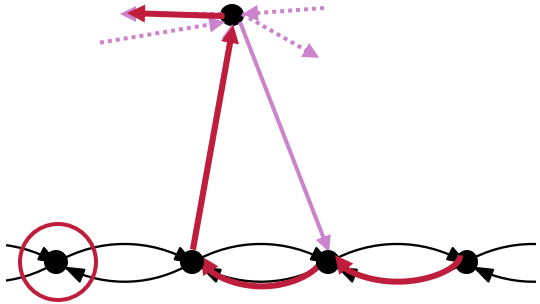
Annahme: Man läuft Klauseln falsch ab.

Man muss richtig rum laufen…



Annahme: Man läuft Klauseln falsch ab.

Man muss richtig rum laufen…



Man kommt nicht mehr weg…

Annahme: Man läuft Klauseln falsch ab.

Korrektheit

„Wenn es einen Hamiltonkreis gibt, dann ist die Formel erfüllbar.“

Dazu müssen wir zeigen:

1. Geht man zu einer Klausel, muss man zur selben Variable zurück.
2. Man darf nur Klauseln ablaufen, wenn die richtige Richtung gewählt wurde.

Korrektheit

„Wenn es einen Hamiltonkreis gibt, dann ist die Formel erfüllbar.“

Dazu müssen wir zeigen:

1. Geht man zu einer Klausel, muss man zur selben Variable zurück. ✓
2. Man darf nur Klauseln ablaufen, wenn die richtige Richtung gewählt wurde.

Korrektheit

„Wenn es einen Hamiltonkreis gibt, dann ist die Formel erfüllbar.“

Dazu müssen wir zeigen:

1. Geht man zu einer Klausel, muss man zur selben Variable zurück. ✓
2. Man darf nur Klauseln ablaufen, wenn die richtige Richtung gewählt wurde. ✓

Korrektheit

„Wenn es einen Hamiltonkreis gibt, dann ist die Formel erfüllbar.“

Dazu müssen wir zeigen:

1. Geht man zu einer Klausel, muss man zur selben Variable zurück. ✓
2. Man darf nur Klauseln ablaufen, wenn die richtige Richtung gewählt wurde. ✓

Also:

Korrektheit

„Wenn es einen Hamiltonkreis gibt, dann ist die Formel erfüllbar.“

Dazu müssen wir zeigen:

1. Geht man zu einer Klausel, muss man zur selben Variable zurück. ✓
2. Man darf nur Klauseln ablaufen, wenn die richtige Richtung gewählt wurde. ✓

Also:

1. Variablen-Gadgets werden in einem Zug durchlaufen

Korrektheit

„Wenn es einen Hamiltonkreis gibt, dann ist die Formel erfüllbar.“

Dazu müssen wir zeigen:

1. Geht man zu einer Klausel, muss man zur selben Variable zurück. ✓
2. Man darf nur Klauseln ablaufen, wenn die richtige Richtung gewählt wurde. ✓

Also:

1. Variablen-Gadgets werden in einem Zug durchlaufen
    - Die Richtung gibt uns *true* oder *false*

Korrektheit

„Wenn es einen Hamiltonkreis gibt, dann ist die Formel erfüllbar.“

Dazu müssen wir zeigen:

1. Geht man zu einer Klausel, muss man zur selben Variable zurück. ✓
2. Man darf nur Klauseln ablaufen, wenn die richtige Richtung gewählt wurde. ✓

Also:

1. Variablen-Gadgets werden in einem Zug durchlaufen
   - Die Richtung gibt uns *true* oder *false*
2. Gibt es keine Belegung der Variablen, sodass φ erfüllt wird, so gibt es immer mindestens ein Klausel-Gadget, dass nicht abgelaufen werden kann.

Korrektheit

„Wenn es einen Hamiltonkreis gibt, dann ist die Formel erfüllbar.“

Dazu müssen wir zeigen:

1. Geht man zu einer Klausel, muss man zur selben Variable zurück. ✓
2. Man darf nur Klauseln ablaufen, wenn die richtige Richtung gewählt wurde. ✓

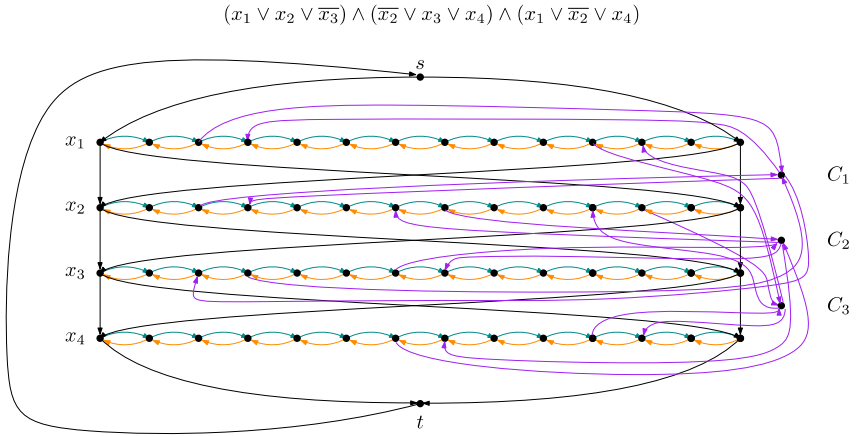
Also:

1. Variablen-Gadgets werden in einem Zug durchlaufen
   - Die Richtung gibt uns *true* oder *false*
2. Gibt es keine Belegung der Variablen, sodass φ erfüllt wird, so gibt es immer mindestens ein Klausel-Gadget, dass nicht abgelaufen werden kann.
   - Es gibt also keinen Hamiltonkreis.

Laufzeit der Transformation

Laufzeit der Transformation

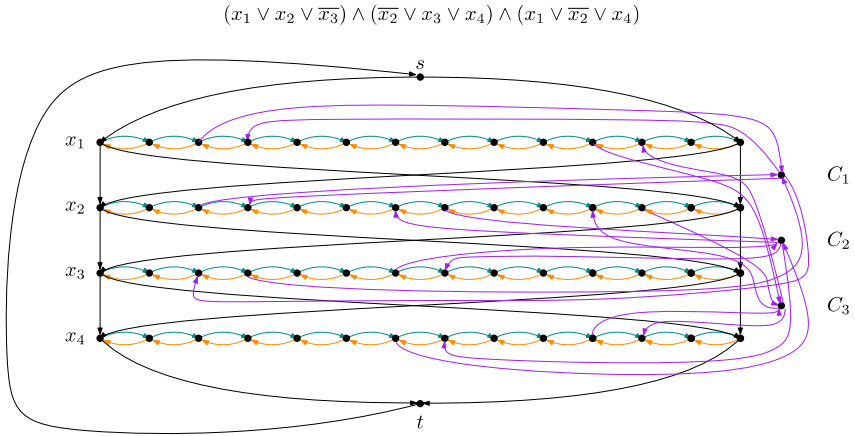
Wir erstellen einen Graphen mit



Laufzeit der Transformation

Wir erstellen einen Graphen mit

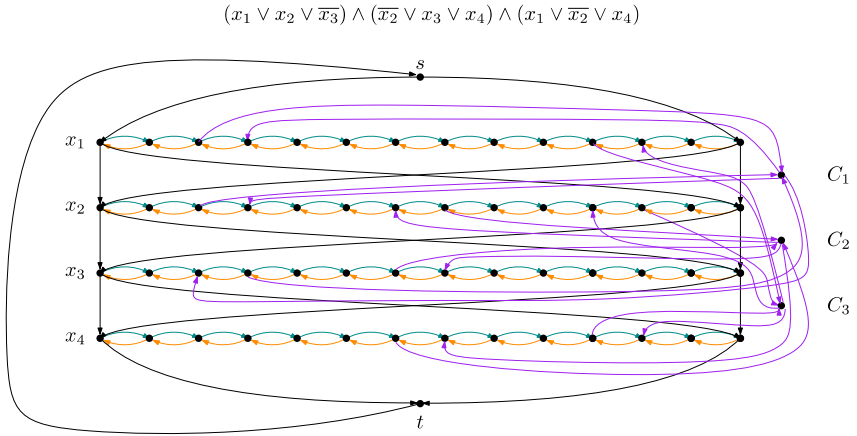
- $4nm + m + 2$ Knoten



Laufzeit der Transformation

Wir erstellen einen Graphen mit

- $4nm + m + 2$ Knoten
- $O(nm)$ vielen Kanten

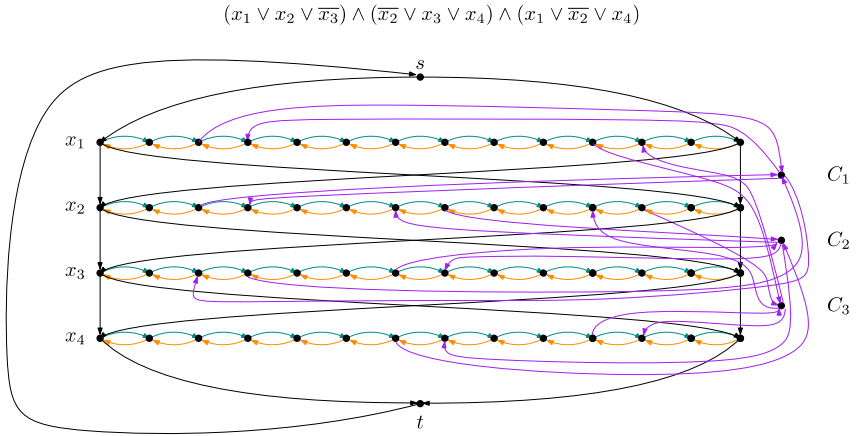


Laufzeit der Transformation

Wir erstellen einen Graphen mit

- $4nm + m + 2$ Knoten
- $O(nm)$ vielen Kanten

Wir brauchen also $O(nm)$ Zeit, den Graphen zu erstellen.



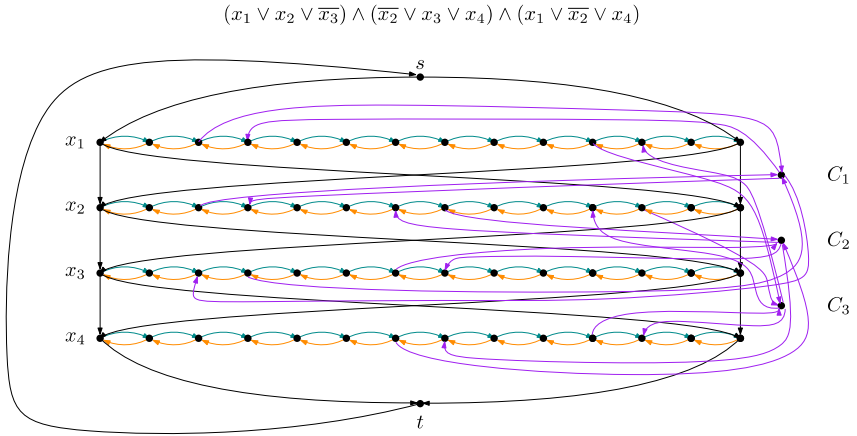
Laufzeit der Transformation

Wir erstellen einen Graphen mit

- $4nm + m + 2$ Knoten
- $O(nm)$ vielen Kanten

Wir brauchen also $O(nm)$ Zeit, den Graphen zu erstellen.

Lösung für 3SAT berechnen



Laufzeit der Transformation

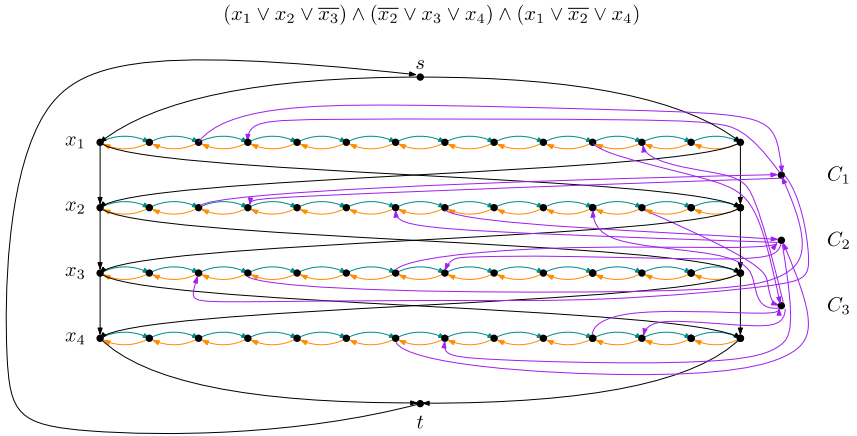
Wir erstellen einen Graphen mit

- $4nm + m + 2$ Knoten
- $O(nm)$ vielen Kanten

Wir brauchen also $O(nm)$ Zeit, den Graphen zu erstellen.

Lösung für 3SAT berechnen

- O(1)



Laufzeit der Transformation

Wir erstellen einen Graphen mit

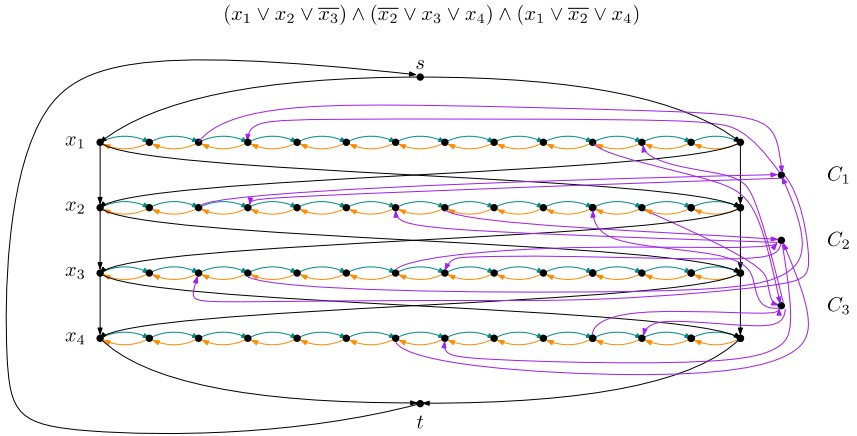
- $4nm + m + 2$ Knoten
- $O(nm)$ vielen Kanten

Wir brauchen also $O(nm)$ Zeit, den Graphen zu erstellen.

Lösung für 3SAT berechnen

- O(1)

Uns interessiert nur, **ob** die Formel erfüllbar ist!



Schritte für einen NP-Schwere Beweis

Schritte für einen NP-Schwere Beweis

Zeige: Problem Π ist NP-schwer

Schritte für einen NP-Schwere Beweis

Zeige: Problem Π ist NP-schwer

- Suche ein geeignetes NP-schweres Problem Π'.

Schritte für einen NP-Schwere Beweis

Zeige: Problem Π ist NP-schwer

- Suche ein geeignetes NP-schweres Problem Π'.
- Reduziere **von** Π' **auf** Π: $\Pi' \leq_p \Pi$

Schritte für einen NP-Schwere Beweis

Zeige: Problem Π ist NP-schwer

- Suche ein geeignetes NP-schweres Problem Π'.
- Reduziere **von** Π' **auf** Π: $\Pi' \leq_p \Pi$
- Beweise Korrektheit:

Schritte für einen NP-Schwere Beweis

Zeige: Problem Π ist NP-schwer

- Suche ein geeignetes NP-schweres Problem Π'.
- Reduziere **von** Π' **auf** Π: $\Pi' \leq_p \Pi$
- Beweise Korrektheit:

  „Für jede Instanz $I_{\Pi'}$ von Π' gibt es genau dann eine Lösung, wenn es für die Instanz $T(I_{\Pi'})$ von Π eine Lösung gibt.“

Schritte für einen NP-Schwere Beweis

Zeige: Problem Π ist NP-schwer

- Suche ein geeignetes NP-schweres Problem Π'.
- Reduziere **von** Π' **auf** Π: $\Pi' \leq_p \Pi$
- Beweise Korrektheit:

  „Für jede Instanz $I_{\Pi'}$ von Π' gibt es genau dann eine Lösung, wenn es für die Instanz $T(I_{\Pi'})$ von Π eine Lösung gibt.“

- Beweise polynomielle Laufzeit von T und T^{-1}.

Schritte für einen NP-Schwere Beweis

Zeige: Problem Π ist NP-schwer

- Suche ein geeignetes NP-schweres Problem Π'.
- Reduziere **von** Π' **auf** Π: $\Pi' \leq_p \Pi$
- Beweise Korrektheit:

  „Für jede Instanz $I_{\Pi'}$ von Π' gibt es genau dann eine Lösung, wenn es für die Instanz $T(I_{\Pi'})$ von Π eine Lösung gibt.“
- Beweise polynomielle Laufzeit von T und T^{-1}.

Schritte für einen NP-Schwere Beweis

Zeige: Problem Π ist NP-schwer

- Suche ein geeignetes NP-schweres Problem Π'.
- Reduziere **von** Π' **auf** Π: $\Pi' \leq_p \Pi$
- Beweise Korrektheit:

  „Für jede Instanz $I_{\Pi'}$ von Π' gibt es genau dann eine Lösung, wenn es für die Instanz $T(I_{\Pi'})$ von Π eine Lösung gibt.“
- Beweise polynomielle Laufzeit von T und T^{-1}.

Beliebte Fehler

Schritte für einen NP-Schwere Beweis

Zeige: Problem Π ist NP-schwer

- Suche ein geeignetes NP-schweres Problem Π'.
- Reduziere **von** Π' **auf** Π: $\Pi' \leq_p \Pi$
- Beweise Korrektheit:

  „Für jede Instanz $I_{\Pi'}$ von Π' gibt es genau dann eine Lösung, wenn es für die Instanz $T(I_{\Pi'})$ von Π eine Lösung gibt.“
- Beweise polynomielle Laufzeit von T und T^{-1}.

Beliebte Fehler

- Nur für ein Beispiel gezeigt.

Schritte für einen NP-Schwere Beweis

Zeige: Problem Π ist NP-schwer

- Suche ein geeignetes NP-schweres Problem Π'.
- Reduziere **von** Π' **auf** Π: $\Pi' \leq_p \Pi$
- Beweise Korrektheit:

  „Für jede Instanz $I_{\Pi'}$ von Π' gibt es genau dann eine Lösung, wenn es für die Instanz $T(I_{\Pi'})$ von Π eine Lösung gibt.“
- Beweise polynomielle Laufzeit von T und T^{-1}.

Beliebte Fehler

- Nur für ein Beispiel gezeigt.
- Reduktion falsch herum.

Schritte für einen NP-Schwere Beweis

Zeige: Problem Π ist NP-schwer

- Suche ein geeignetes NP-schweres Problem Π'.
- Reduziere **von** Π' **auf** Π: $\Pi' \leq_p \Pi$
- Beweise Korrektheit:

  „Für jede Instanz $I_{\Pi'}$ von Π' gibt es genau dann eine Lösung, wenn es für die Instanz $T(I_{\Pi'})$ von Π eine Lösung gibt.“
- Beweise polynomielle Laufzeit von T und T^{-1}.

Beliebte Fehler

- Nur für ein Beispiel gezeigt.
- Reduktion falsch herum.
- Im Korrektheitsbeweis nur eine Richtung gezeigt.

Schritte für einen NP-Schwere Beweis

Zeige: Problem Π ist NP-schwer

- Suche ein geeignetes NP-schweres Problem Π'.
- Reduziere **von** Π' **auf** Π: $\Pi' \leq_p \Pi$
- Beweise Korrektheit:

  „Für jede Instanz $I_{\Pi'}$ von Π' gibt es genau dann eine Lösung, wenn es für die Instanz $T(I_{\Pi'})$ von Π eine Lösung gibt.“
- Beweise polynomielle Laufzeit von T und T^{-1}.

Beliebte Fehler

- Nur für ein Beispiel gezeigt.
- Reduktion falsch herum.
- Im Korrektheitsbeweis nur eine Richtung gezeigt.
- Laufzeit der Transformation nicht berücksichtigt.

Schritte für einen NP-Schwere Beweis

Zeige: Problem Π ist NP-schwer

- Suche ein geeignetes NP-schweres Problem Π'.
- Reduziere **von** Π' **auf** Π: $\Pi' \leq_p \Pi$
- Beweise Korrektheit:

  „Für jede Instanz $I_{\Pi'}$ von Π' gibt es genau dann eine Lösung, wenn es für die Instanz $T(I_{\Pi'})$ von Π eine Lösung gibt.“
- Beweise polynomielle Laufzeit von T und T^{-1}.

Beliebte Fehler

- Nur für ein Beispiel gezeigt.
- Reduktion falsch herum.
- Im Korrektheitsbeweis nur eine Richtung gezeigt.
- Laufzeit der Transformation nicht berücksichtigt.
- Codierungsgröße von $T(I_{\Pi'})$ ist nicht polynomiell durch die Größe von $I_{\Pi'}$ beschränkt.

Hashing

Hashing

(Symbolbild)

Hashing

Hashing

Hashfunktion

Hashing

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Hashing

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

Hashing

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

Hashing

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

590a 23b8

Hashing

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

Hashing


42
“Hello World”
Beliebige (Zahlen) Daten beliebiger Länge
Hashfunktion
Zahl (Hash) aus endlicher Menge von Zahlen
8410 b221
590a 23b8

Hashing

“192.168.21.101”

42

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

8410 b221 590a 23b8

Hashing

“192.168.21.101”

42

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

8410 b221 590a 23b8

78bc 08ff

Hashing

“192.168.21.101”

password.txt

42

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

8410 b221 590a 23b8

78bc 08ff

Hashing

“192.168.21.101”

password.txt

42

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

8410 b221 590a 23b8

78bc 08ff 3bcd f783

Hashing

“192.168.21.101”

“Was hast du gerade über mich gesagt, du kleiner Studi? Du solltest wissen, ich habe mein Pharmazie Studium in Regelstudienzeit abgeschlossen, war in vielen geheimen Meetings gegen die Studierendenschaft und habe über 300 bestätigte Abbrecher! […]”

password.txt

42

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

8410 b221 590a 23b8

78bc 08ff 3bcd f783

Hashing

“192.168.21.101”

“Was hast du gerade über mich gesagt, du kleiner Studi? Du solltest wissen, ich habe mein Pharmazie Studium in Regelstudienzeit abgeschlossen, war in vielen geheimen Meetings gegen die Studierendenschaft und habe über 300 bestätigte Abbrecher! […]”

password.txt

42

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

8410 b221 590a 23b8 2401 ba98

78bc 08ff 3bcd f783

Hashing

“192.168.21.101”

data.zip (3.1GB)

“Was hast du gerade über mich gesagt, du kleiner Studi? Du solltest wissen, ich habe mein Pharmazie Studium in Regelstudienzeit abgeschlossen, war in vielen geheimen Meetings gegen die Studierendenschaft und habe über 300 bestätigte Abbrecher! […]”

password.txt

42

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

8410 b221 590a 23b8 2401 ba98

78bc 08ff 3bcd f783

Hashing

“192.168.21.101”

data.zip (3.1GB)

“Was hast du gerade über mich gesagt, du kleiner Studi? Du solltest wissen, ich habe mein Pharmazie Studium in Regelstudienzeit abgeschlossen, war in vielen geheimen Meetings gegen die Studierendenschaft und habe über 300 bestätigte Abbrecher! […]”

password.txt

42

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

8410 b221 590a 23b8 2401 ba98

b0a1 83f1 78bc 08ff 3bcd f783

Hashing

“192.168.21.101”

data.zip (3.1GB)

“Was hast du gerade über mich gesagt, du kleiner Studi? Du solltest wissen, ich habe mein Pharmazie Studium in Regelstudienzeit abgeschlossen, war in vielen geheimen Meetings gegen die Studierendenschaft und habe über 300 bestätigte Abbrecher! […]”

password.txt

42

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

8410 b221 590a 23b8 2401 ba98

b0a1 83f1 78bc 08ff 3bcd f783

Hashing

“192.168.21.101”

data.zip (3.1GB)

“Was hast du gerade über mich gesagt, du kleiner Studi? Du solltest wissen, ich habe mein Pharmazie Studium in Regelstudienzeit abgeschlossen, war in vielen geheimen Meetings gegen die Studierendenschaft und habe über 300 bestätigte Abbrecher! […]”

password.txt

42

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

890b 23a8 8410 b221 590a 23b8 2401 ba98

b0a1 83f1 78bc 08ff 3bcd f783

Hashing

“192.168.21.101”

data.zip (3.1GB)

“Was hast du gerade über mich gesagt, du kleiner Studi? Du solltest wissen, ich habe mein Pharmazie Studium in Regelstudienzeit abgeschlossen, war in vielen geheimen Meetings gegen die Studierendenschaft und habe über 300 bestätigte Abbrecher! […]”

password.txt

42

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

890b 23a8 8410 b221 590a 23b8 2401 ba98

b0a1 83f1 78bc 08ff 3bcd f783

Hashing

“192.168.21.101”

data.zip (3.1GB)

“Was hast du gerade über mich gesagt, du kleiner Studi? Du solltest wissen, ich habe mein Pharmazie Studium in Regelstudienzeit abgeschlossen, war in vielen geheimen Meetings gegen die Studierendenschaft und habe über 300 bestätigte Abbrecher! […]”

password.txt

42

Beliebige (Zahlen) Daten beliebiger Länge

Hashfunktion

Zahl (Hash) aus endlicher Menge von Zahlen

890b 23a8	8410 b221	590a 23b8	2401 ba98
b0a1 83f1	09c8 89dd	78bc 08ff	3bcd f783

Hashing: Kontexte

Hashing: Kontexte

Diese Veranstaltung

Hashing: Kontexte

Diese Veranstaltung
- Hash-Tabellen

Hashing: Kontexte

Diese Veranstaltung
- Hash-Tabellen

Weitere Kontexte / Anwendungen

Hashing: Kontexte

Diese Veranstaltung
- Hash-Tabellen

Weitere Kontexte / Anwendungen
- Checksums

Hashing: Kontexte

Diese Veranstaltung
- Hash-Tabellen

Weitere Kontexte / Anwendungen
- Checksums
- Kryptografie

Hashing: Kontexte

Diese Veranstaltung
- Hash-Tabellen

Weitere Kontexte / Anwendungen
- Checksums
- Kryptografie
- Datenbanken

Hashing: Kontexte

Diese Veranstaltung
- Hash-Tabellen

Weitere Kontexte / Anwendungen
- Checksums
- Kryptografie
- Datenbanken
- Caches

Hashing: Kontexte

Diese Veranstaltung
- Hash-Tabellen

Weitere Kontexte / Anwendungen
- Checksums
- Kryptografie
- Datenbanken
- Caches
- …

Dictionaries

Dictionaries

Assoziative Datenstruktur, speichere Elemente (Value) unter einem Schlüssel (Key).

Dictionaries

Assoziative Datenstruktur, speichere Elemente (Value) unter einem Schlüssel (Key).

Beispiel

Dictionaries

Assoziative Datenstruktur, speichere Elemente (Value) unter einem Schlüssel (Key).

Beispiel

Domainnamen

Dictionaries

Assoziative Datenstruktur, speichere Elemente (Value) unter einem Schlüssel (Key).

Beispiel

Domainnamen

IP-Adressen

Dictionaries

Assoziative Datenstruktur, speichere Elemente (Value) unter einem Schlüssel (Key).

Beispiel

Domainnamen

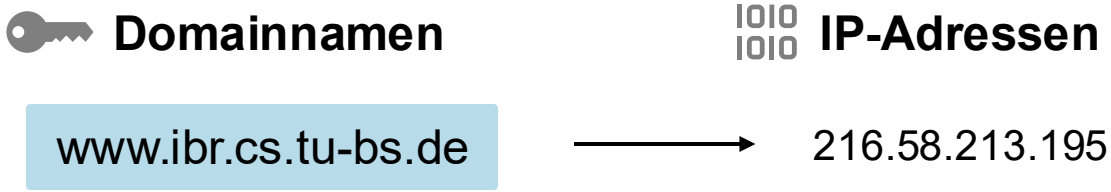
IP-Adressen

www.ibr.cs.tu-bs.de

Dictionaries

Assoziative Datenstruktur, speichere Elemente (Value) unter einem Schlüssel (Key).

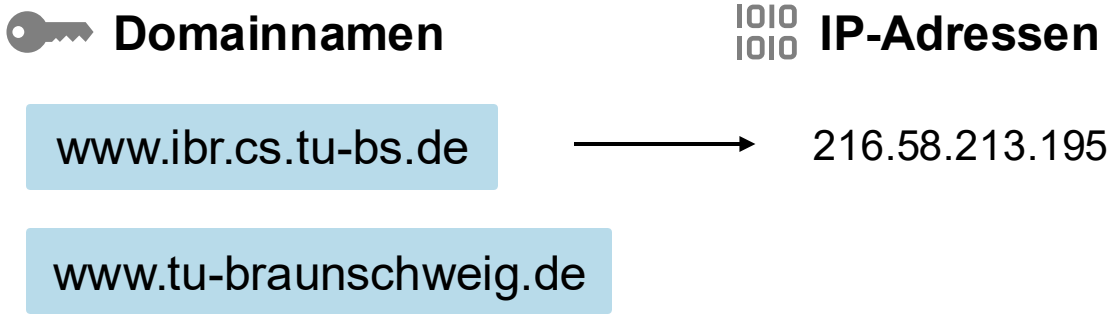
Beispiel



Dictionaries

Assoziative Datenstruktur, speichere Elemente (Value) unter einem Schlüssel (Key).

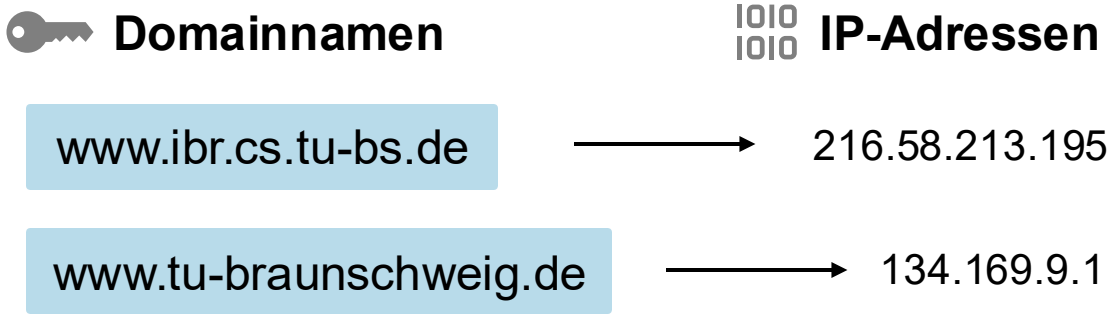
Beispiel



Dictionaries

Assoziative Datenstruktur, speichere Elemente (Value) unter einem Schlüssel (Key).

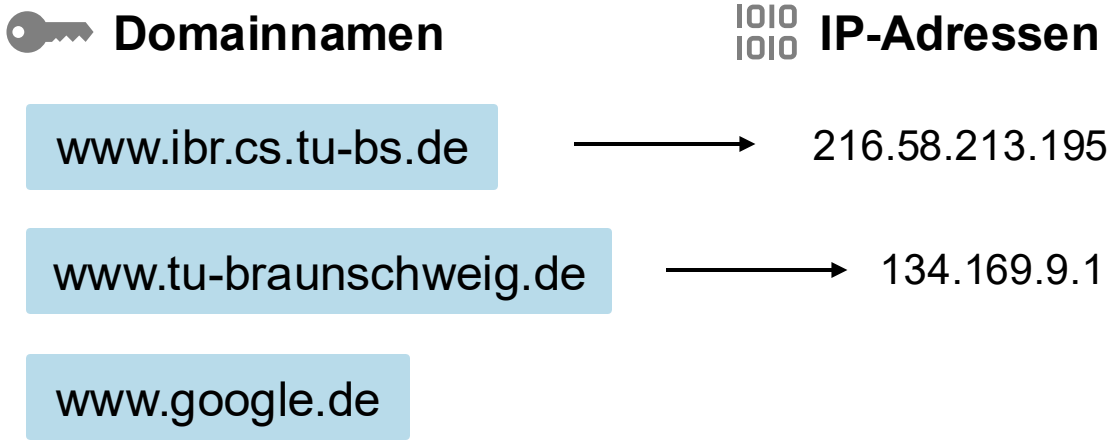
Beispiel



Dictionaries

Assoziative Datenstruktur, speichere Elemente (Value) unter einem Schlüssel (Key).

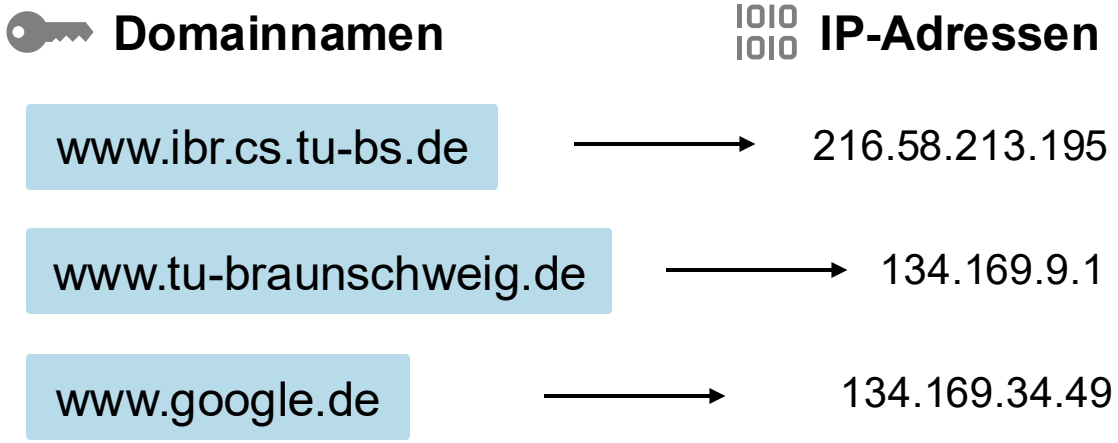
Beispiel



Dictionaries

Assoziative Datenstruktur, speichere Elemente (Value) unter einem Schlüssel (Key).

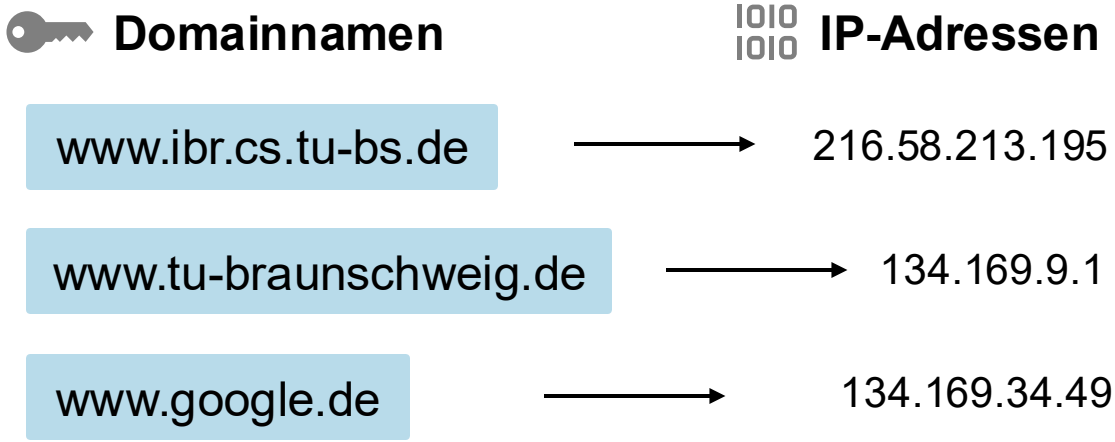
Beispiel



Dictionaries

Assoziative Datenstruktur, speichere Elemente (Value) unter einem Schlüssel (Key).

Beispiel



Keine Reihenfolge garantiert. Hier kann man nicht sortieren, Vorgänger finden etc.

Ein Blick zurück zu AuD 1

Ein Blick zurück zu AuD 1

Wir könnten Dictionaries mit Datenstrukturen aus AuD1 bauen ...

Ein Blick zurück zu AuD 1

Wir könnten Dictionaries mit Datenstrukturen aus AuD1 bauen ...

Laufzeiten
Suchen
Einfügen
Löschen

Ein Blick zurück zu AuD 1

Wir könnten Dictionaries mit Datenstrukturen aus AuD1 bauen ...

Laufzeiten	**Verkettete Liste**
Suchen	
Einfügen	
Löschen	

Ein Blick zurück zu AuD 1

Wir könnten Dictionaries mit Datenstrukturen aus AuD1 bauen ...

Laufzeiten	**Verkettete Liste**
Suchen	$\mathcal{O}(n)$
Einfügen	$\mathcal{O}(1)$
Löschen	$\mathcal{O}(n)$ / $\mathcal{O}(1)$

Ein Blick zurück zu AuD 1

Wir könnten Dictionaries mit Datenstrukturen aus AuD1 bauen ...

Laufzeiten	Verkettete Liste	Binäre Bäume
Suchen	$\mathcal{O}(n)$	
Einfügen	$\mathcal{O}(1)$	
Löschen	$\mathcal{O}(n)$ / $\mathcal{O}(1)$	

Ein Blick zurück zu AuD 1

Wir könnten Dictionaries mit Datenstrukturen aus AuD1 bauen ...

Laufzeiten	Verkettete Liste	Binäre Bäume
Suchen	$\mathcal{O}(n)$	$\mathcal{O}(h)$
Einfügen	$\mathcal{O}(1)$	$\mathcal{O}(h)$
Löschen	$\mathcal{O}(n)$ / $\mathcal{O}(1)$	$\mathcal{O}(h)$

Ein Blick zurück zu AuD 1

Wir könnten Dictionaries mit Datenstrukturen aus AuD1 bauen ...

Laufzeiten	Verkettete Liste	Binäre Bäume	AVL-Bäume
Suchen	$\mathcal{O}(n)$	$\mathcal{O}(h)$	
Einfügen	$\mathcal{O}(1)$	$\mathcal{O}(h)$	
Löschen	$\mathcal{O}(n)$ / $\mathcal{O}(1)$	$\mathcal{O}(h)$	

Ein Blick zurück zu AuD 1

Wir könnten Dictionaries mit Datenstrukturen aus AuD1 bauen ...

Laufzeiten	Verkettete Liste	Binäre Bäume	AVL-Bäume
Suchen	$\mathcal{O}(n)$	$\mathcal{O}(h)$	$\mathcal{O}(\log n)$
Einfügen	$\mathcal{O}(1)$	$\mathcal{O}(h)$	$\mathcal{O}(\log n)$
Löschen	$\mathcal{O}(n)$ / $\mathcal{O}(1)$	$\mathcal{O}(h)$	$\mathcal{O}(\log n)$

Ein Blick zurück zu AuD 1

Wir könnten Dictionaries mit Datenstrukturen aus AuD1 bauen ...

Laufzeiten	Verkettete Liste	Binäre Bäume	AVL-Bäume	(Max) Heaps
Suchen	$\mathcal{O}(n)$	$\mathcal{O}(h)$	$\mathcal{O}(\log n)$	
Einfügen	$\mathcal{O}(1)$	$\mathcal{O}(h)$	$\mathcal{O}(\log n)$	
Löschen	$\mathcal{O}(n)$ / $\mathcal{O}(1)$	$\mathcal{O}(h)$	$\mathcal{O}(\log n)$	

Ein Blick zurück zu AuD 1

Wir könnten Dictionaries mit Datenstrukturen aus AuD1 bauen ...

Laufzeiten	Verkettete Liste	Binäre Bäume	AVL-Bäume	(Max) Heaps
Suchen	$\mathcal{O}(n)$	$\mathcal{O}(h)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Einfügen	$\mathcal{O}(1)$	$\mathcal{O}(h)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Löschen	$\mathcal{O}(n)$ / $\mathcal{O}(1)$	$\mathcal{O}(h)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Ein Blick zurück zu AuD 1

Wir könnten Dictionaries mit Datenstrukturen aus AuD1 bauen ...

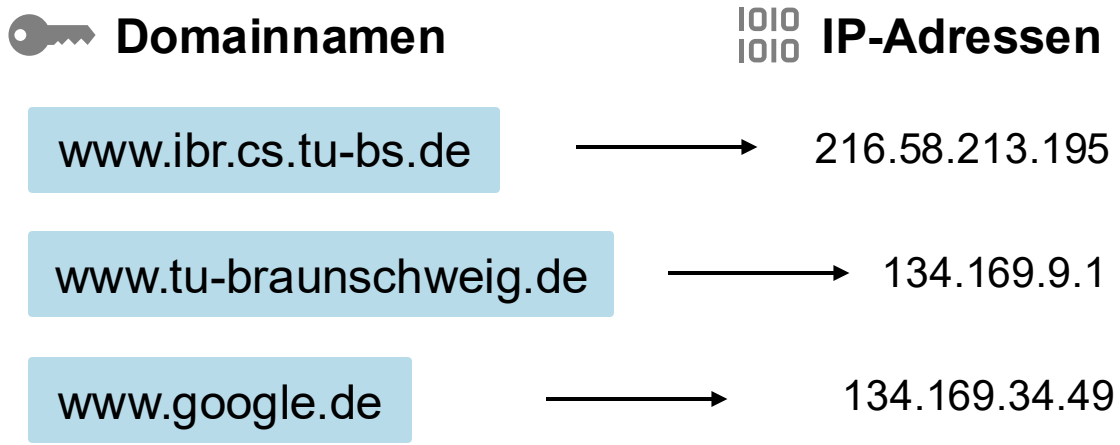
Laufzeiten	Verkettete Liste	Binäre Bäume	AVL-Bäume	(Max) Heaps
Suchen	$\mathcal{O}(n)$	$\mathcal{O}(h)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Einfügen	$\mathcal{O}(1)$	$\mathcal{O}(h)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Löschen	$\mathcal{O}(n)$ / $\mathcal{O}(1)$	$\mathcal{O}(h)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Mit Hashing geht das tatsächlich noch schneller!

Hash-Tabellen

Assoziative Datenstruktur mit extrem schnellen Operationen für Suchen, Einfügen und Löschen.

Beispiel



Hash-Tabellen

Schlüssel

www.ibr.cs.tu-bs.de

www.tu-braunschweig.de

www.google.de

Hash-Tabellen

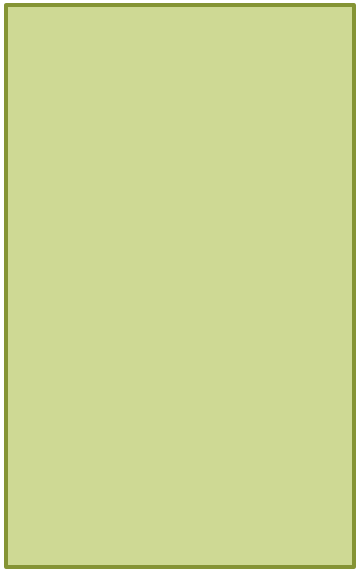
Schlüssel

www.ibr.cs.tu-bs.de

www.tu-braunschweig.de

www.google.de

Hashfunktion



Hash-Tabellen

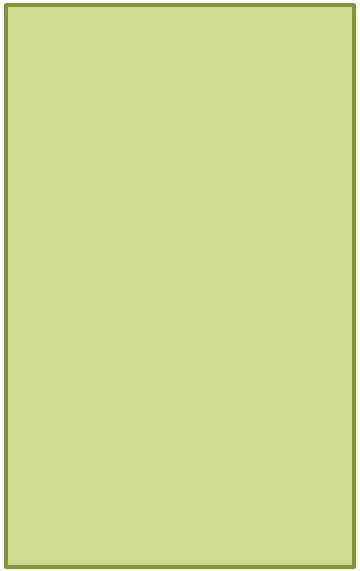
Schlüssel

www.ibr.cs.tu-bs.de

www.tu-braunschweig.de

www.google.de

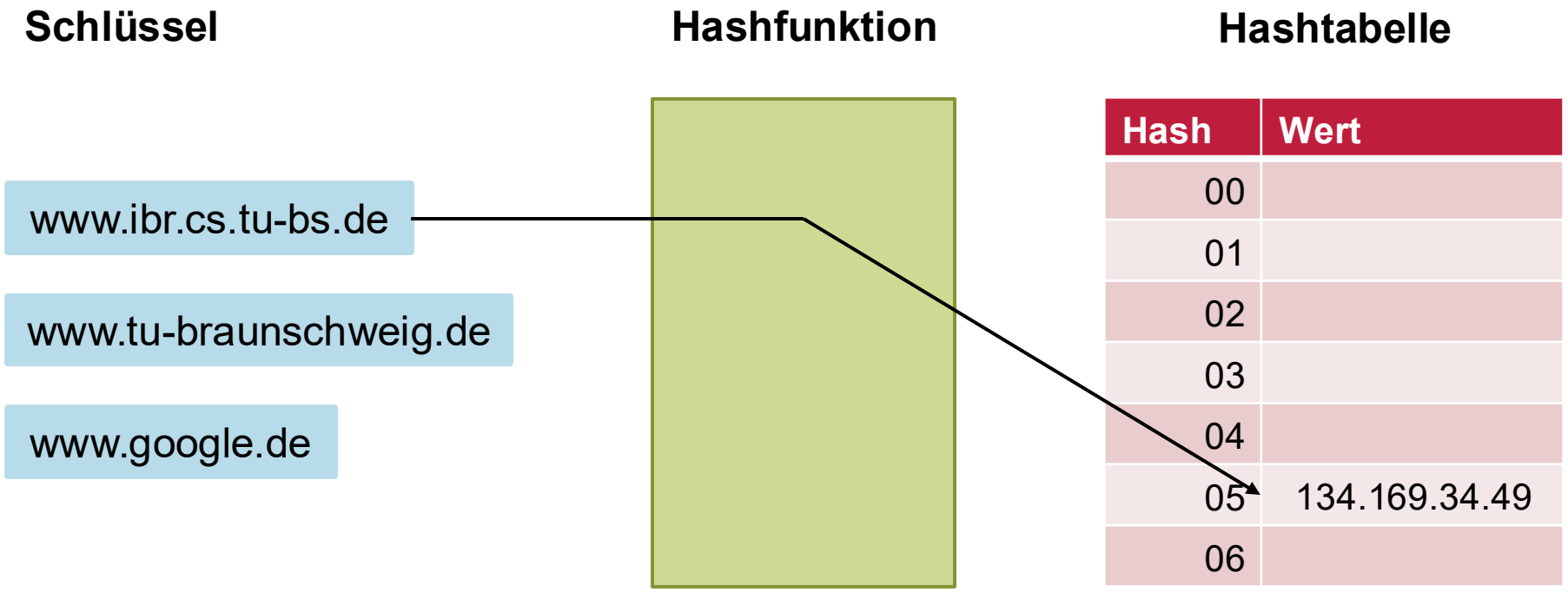
Hashfunktion



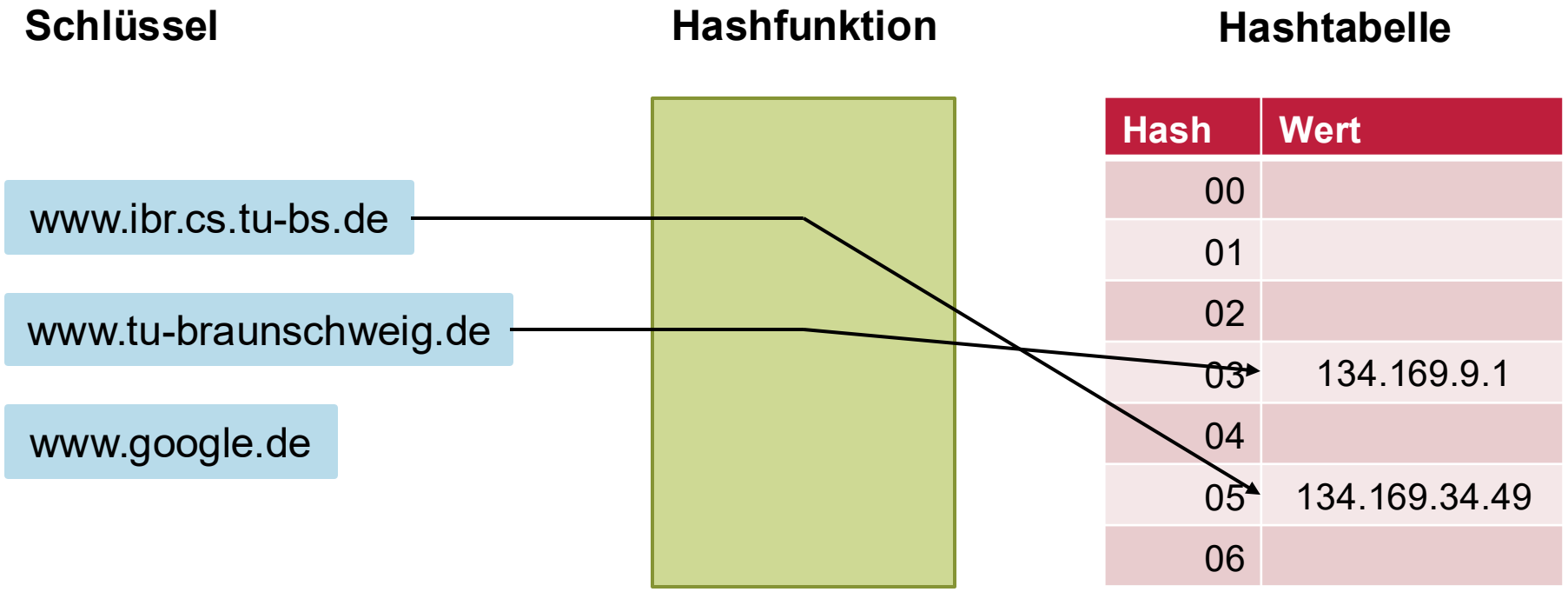
Hashtabelle

Hash	Wert
00	
01	
02	
03	
04	
05	
06	

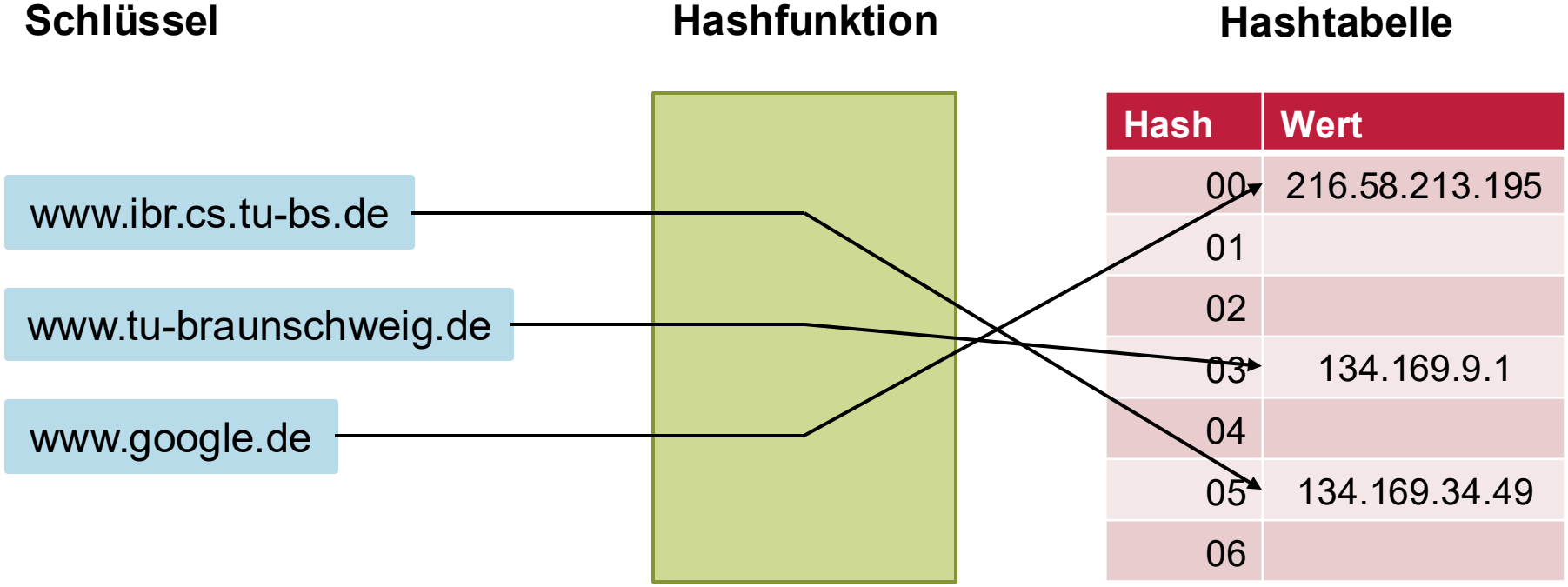
Hash-Tabellen



Hash-Tabellen



Hash-Tabellen



Hashing

Universum

Hashing

Jedes Objekt besitzt einen Schlüssel

Universum

Hashing

Jedes Objekt besitzt einen Schlüssel

Universum

Black-Box wandelt Schlüssel in eine Position in der Hashtabelle um.

Black-Box

Hashfunktion

Hashing

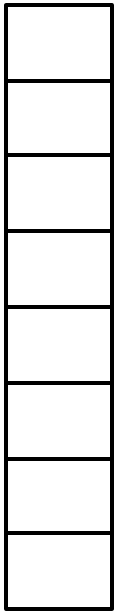
Jedes Objekt besitzt einen Schlüssel

Universum

Black-Box wandelt Schlüssel in eine Position in der Hashtabelle um.

Black-Box

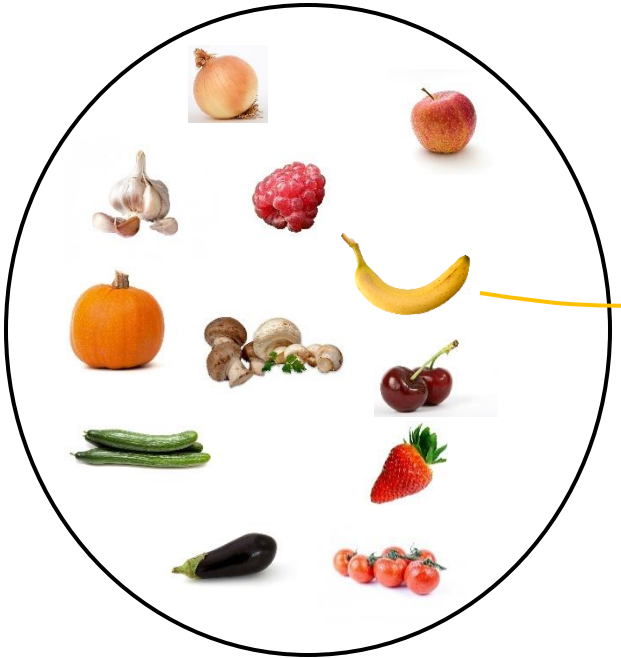
Hashfunktion



Hashtabelle

Hashing

Jedes Objekt besitzt einen Schlüssel

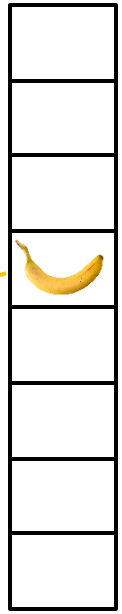


Universum

Black-Box wandelt Schlüssel in eine Position in der Hashtabelle um.

Black-Box

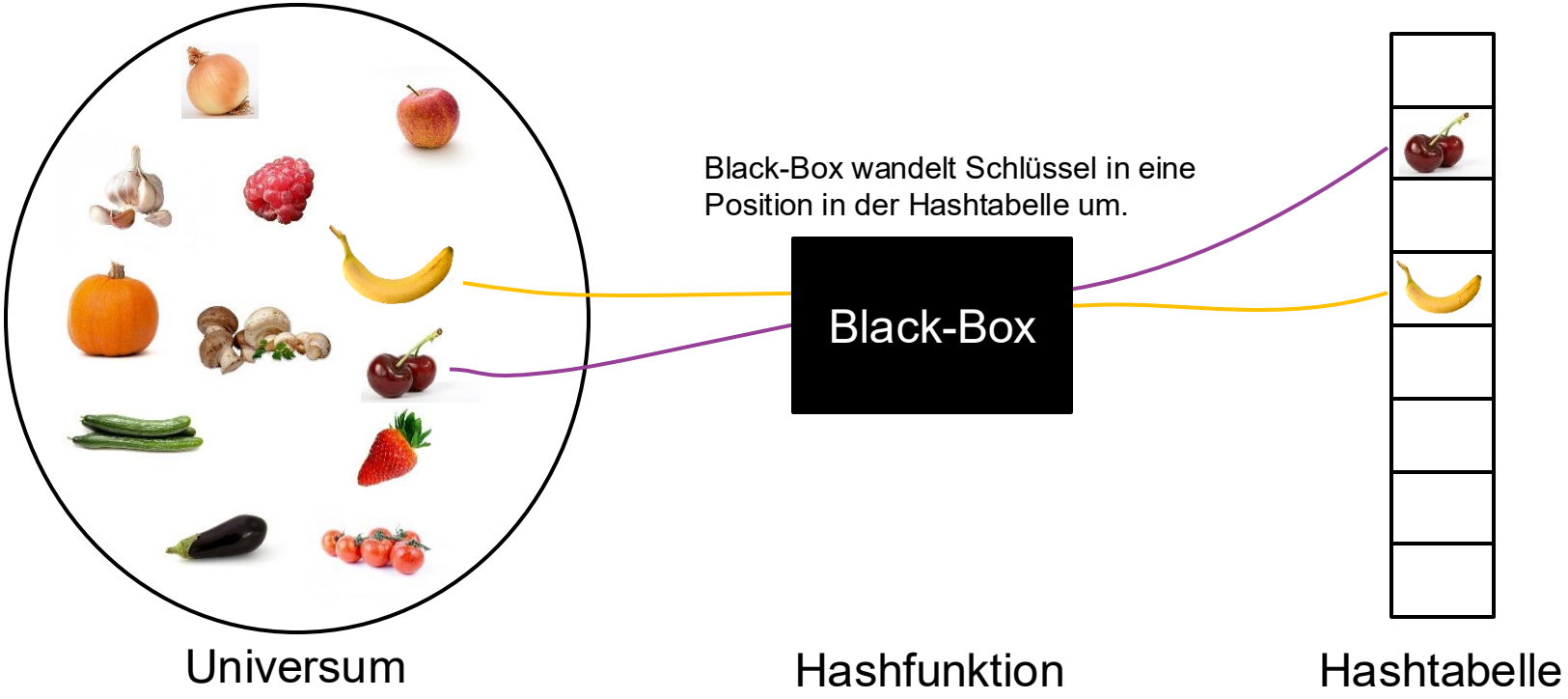
Hashfunktion



Hashtabelle

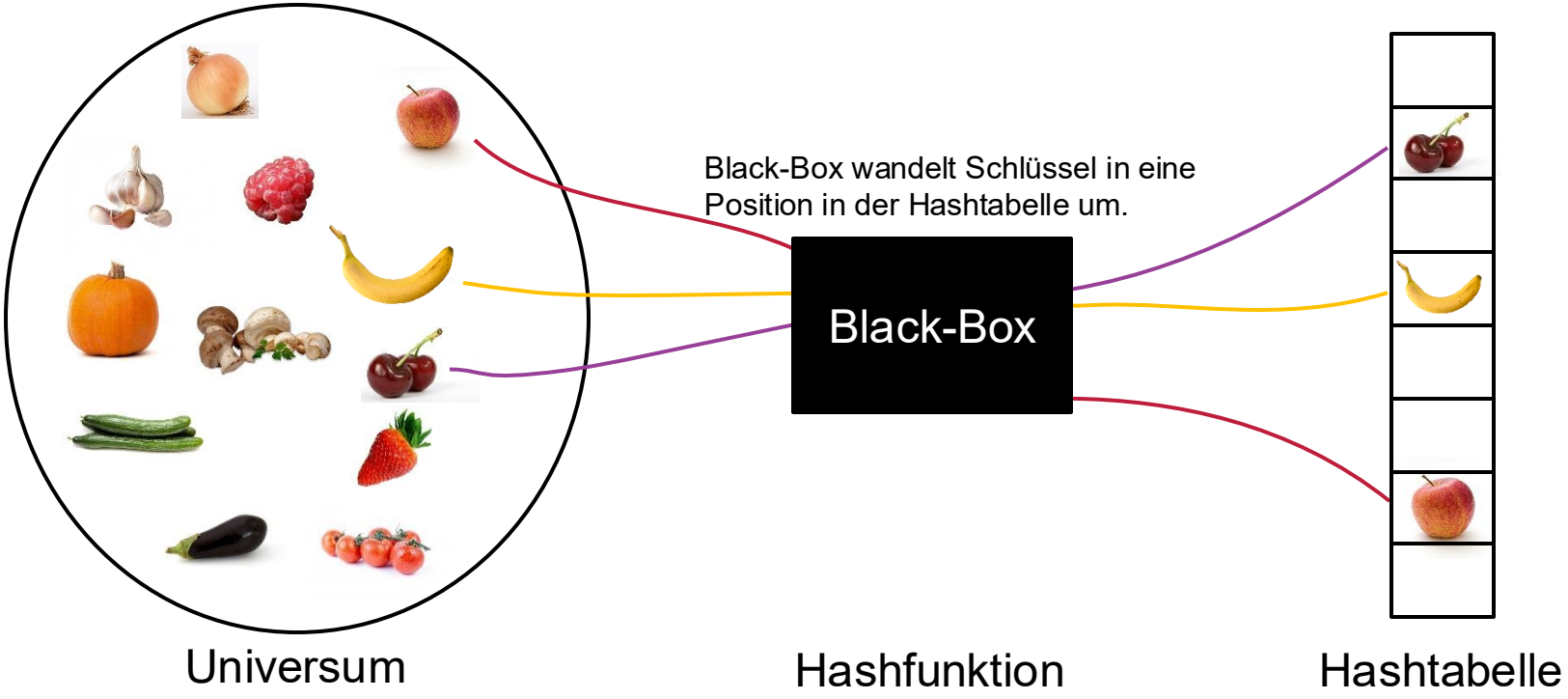
Hashing

Jedes Objekt besitzt einen Schlüssel



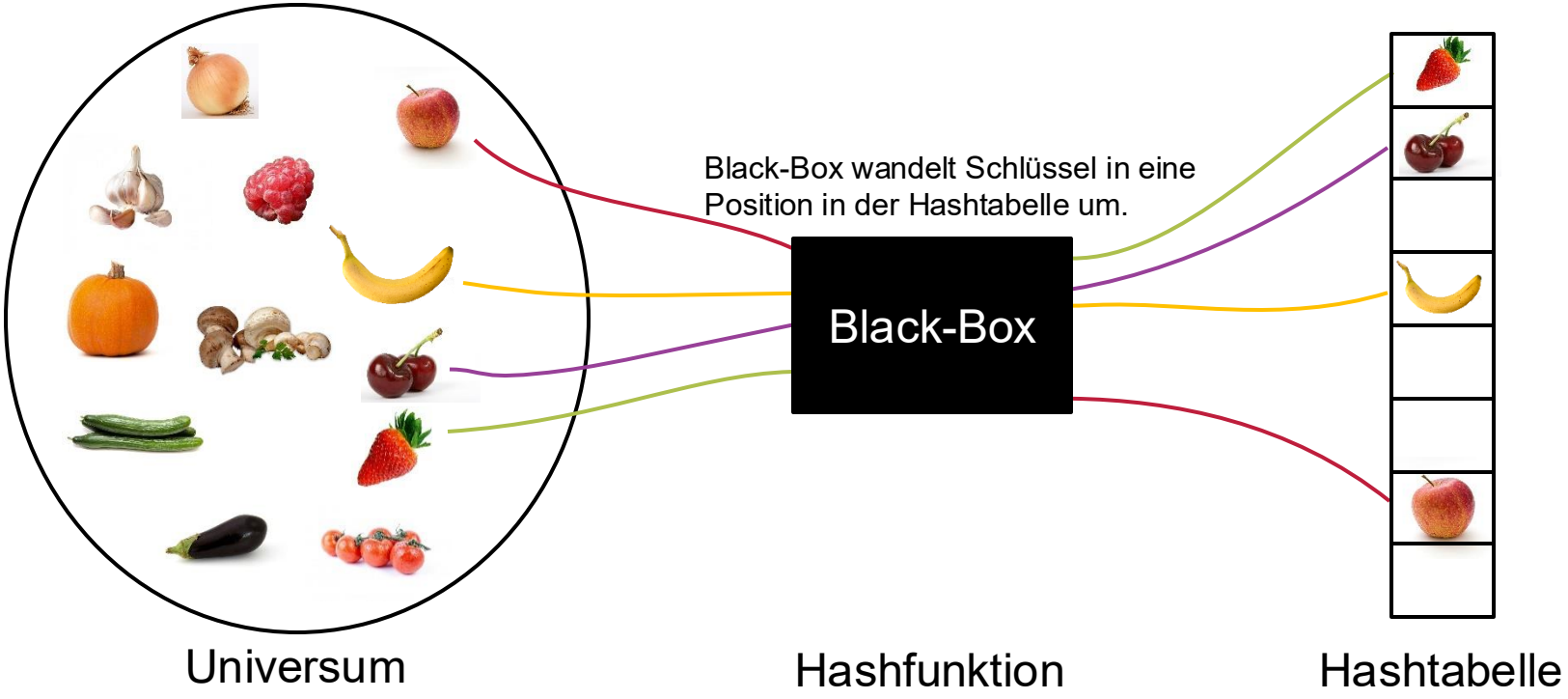
Hashing

Jedes Objekt besitzt einen Schlüssel



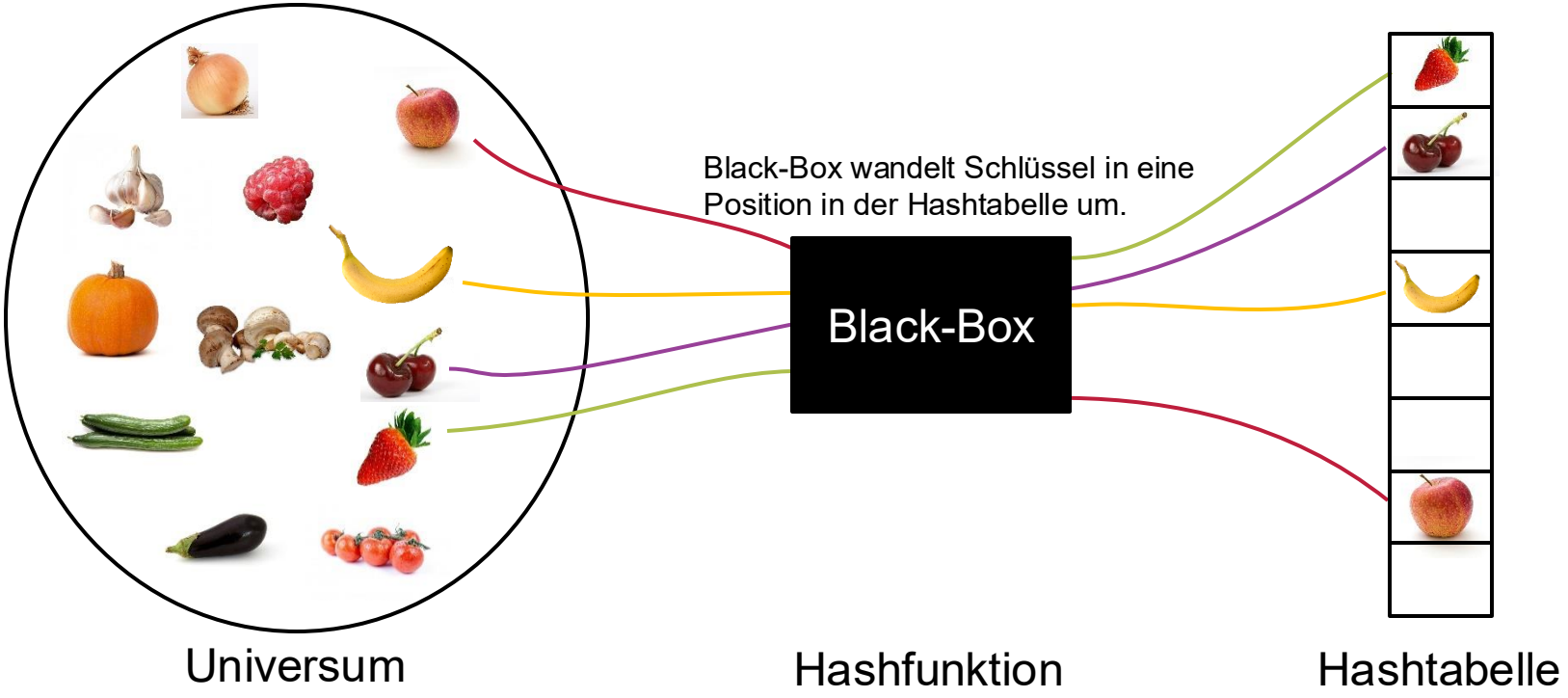
Hashing

Jedes Objekt besitzt einen Schlüssel



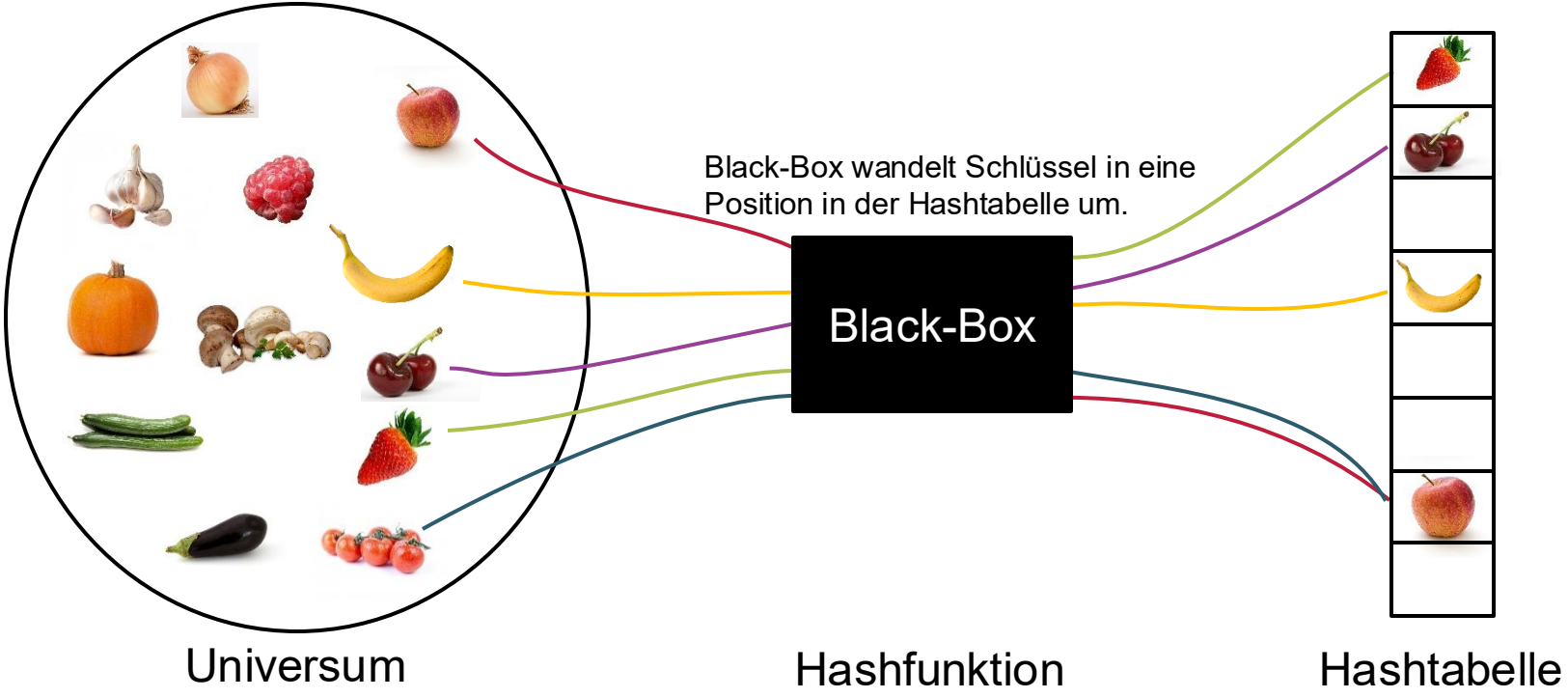
Hashing

Jedes Objekt besitzt einen Schlüssel



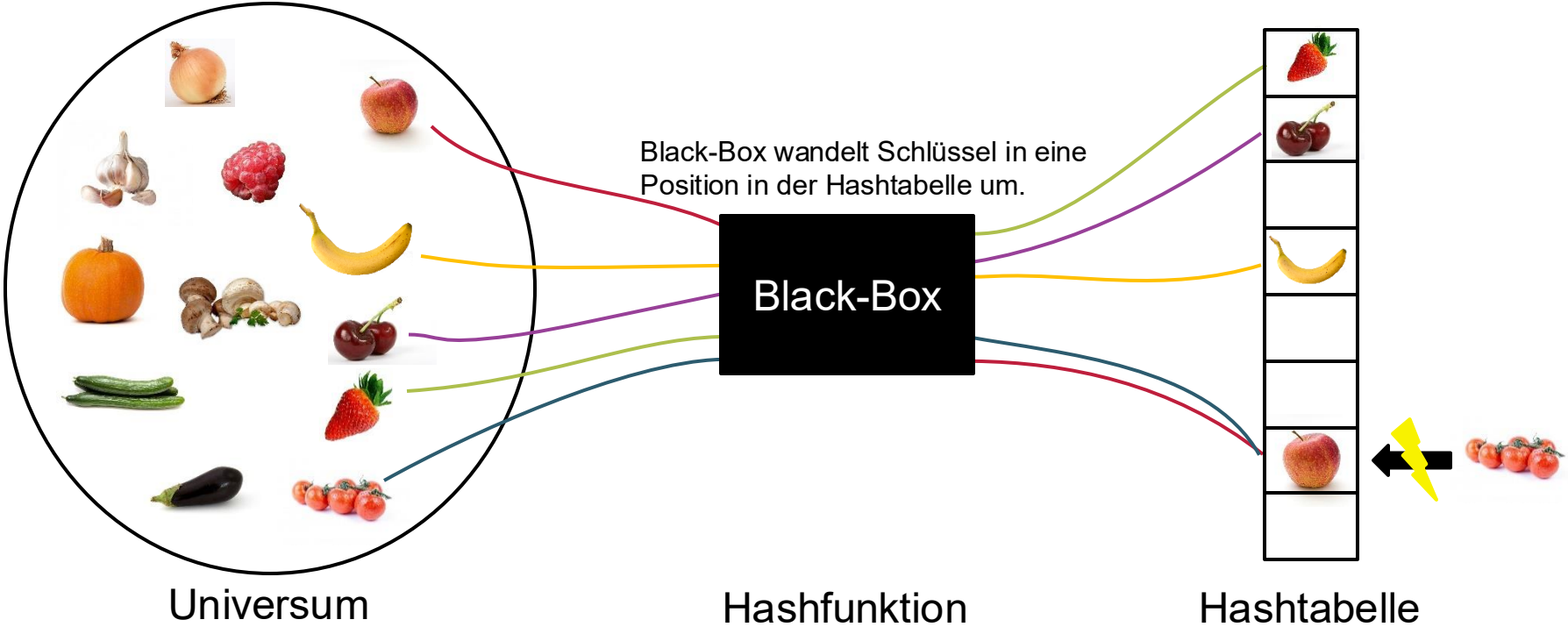
Hashing

Jedes Objekt besitzt einen Schlüssel



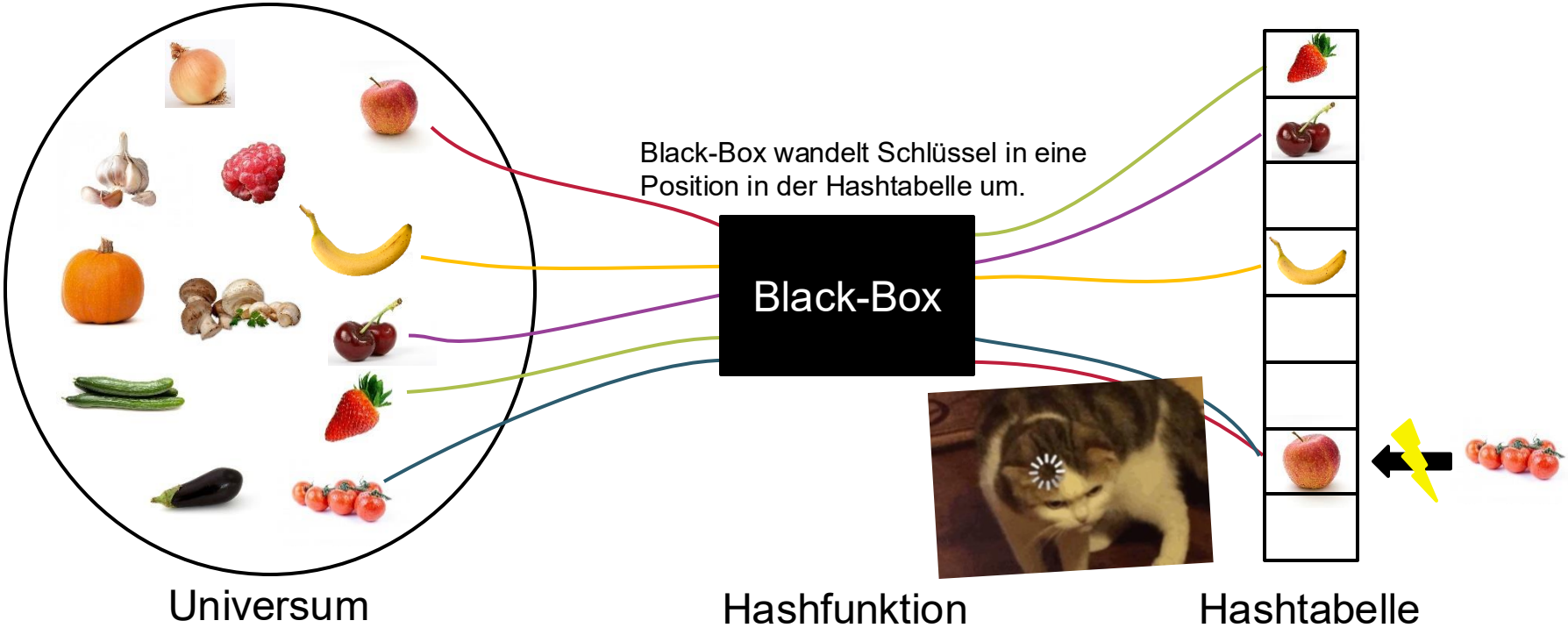
Hashing – Kollision

Jedes Objekt besitzt einen Schlüssel



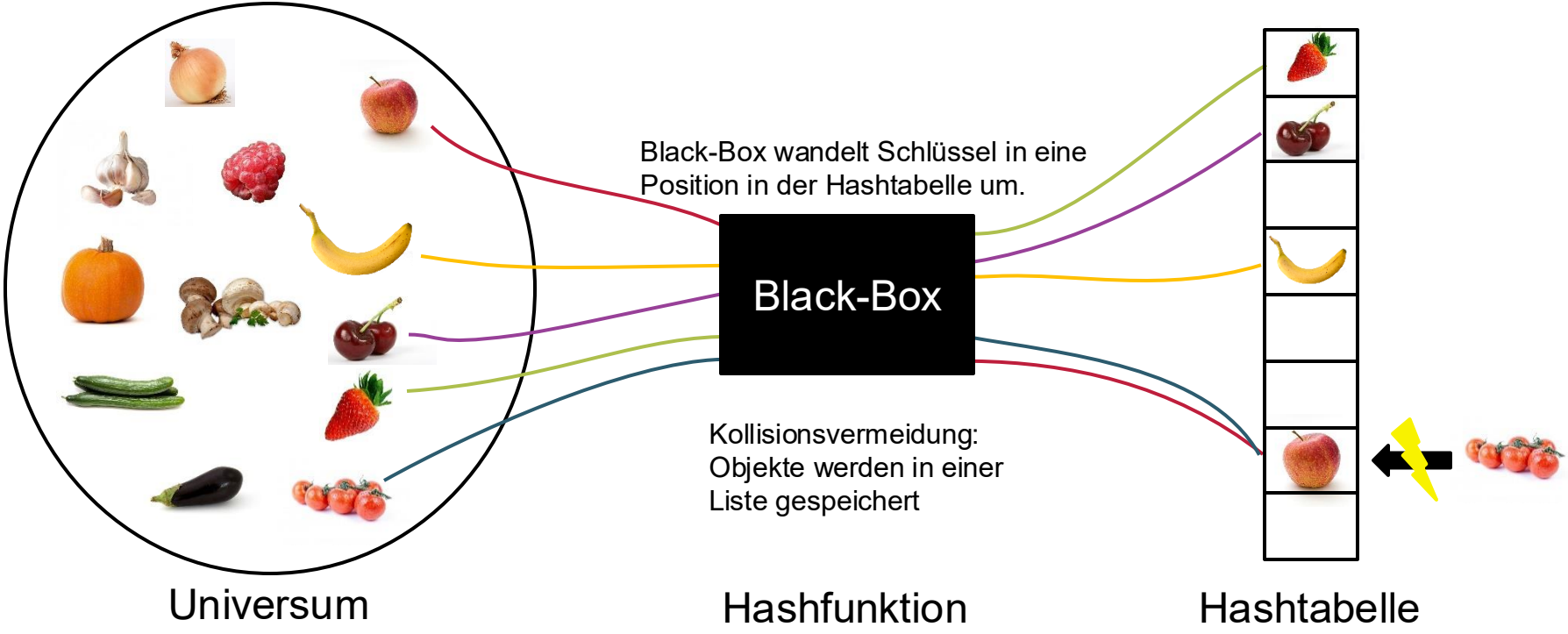
Hashing – Kollision

Jedes Objekt besitzt einen Schlüssel



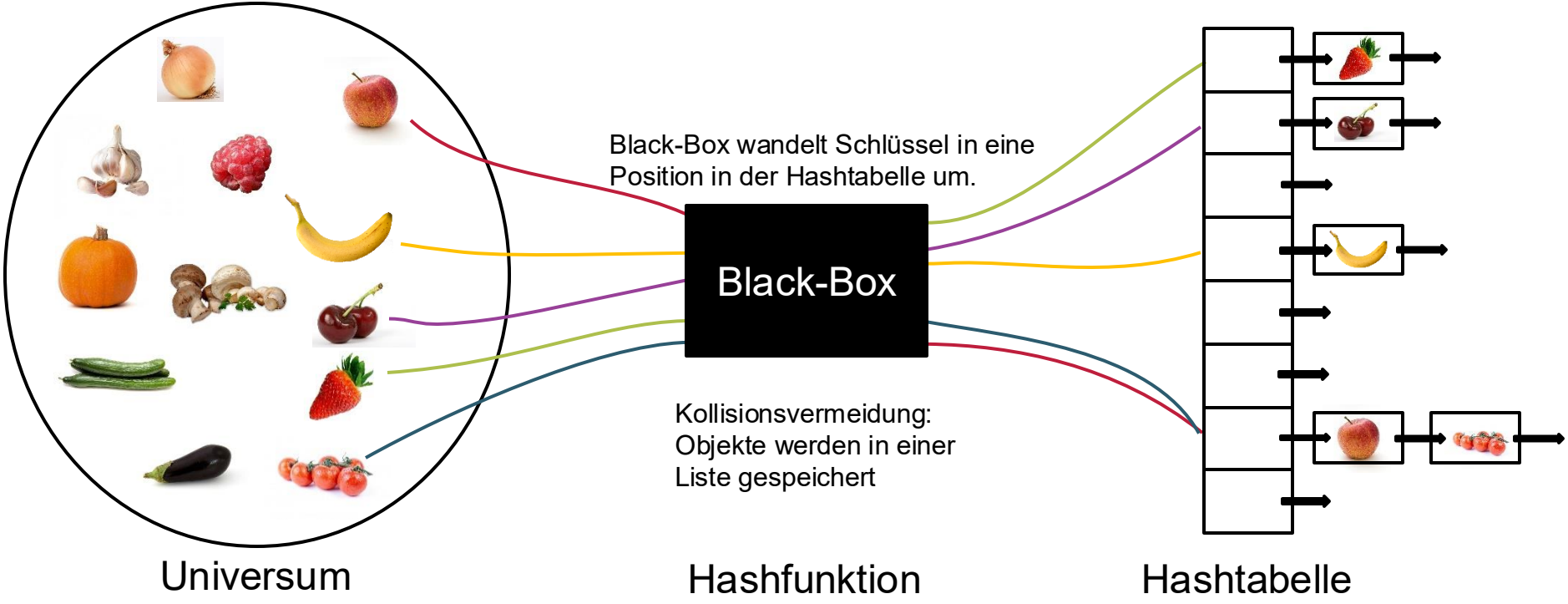
Hashing – Kollision

Jedes Objekt besitzt einen Schlüssel



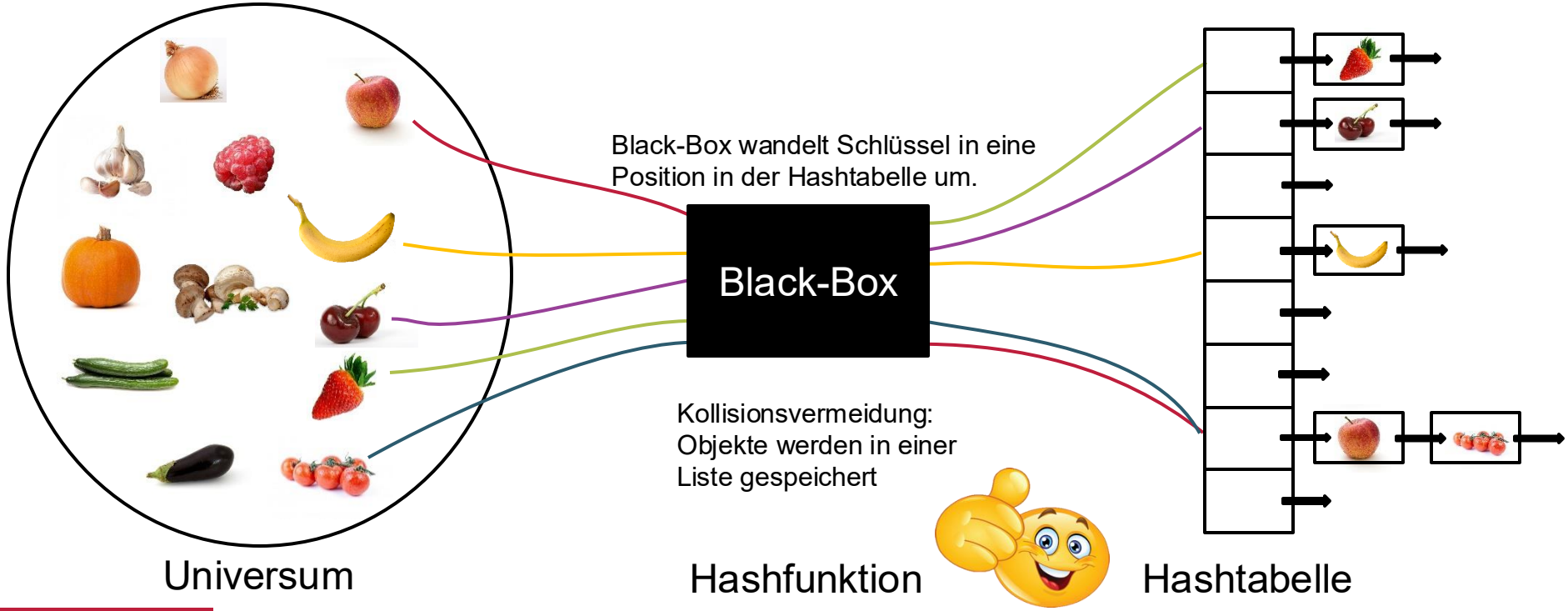
Hashing – Listen

Jedes Objekt besitzt einen Schlüssel



Hashing – Listen

Jedes Objekt besitzt einen Schlüssel



Hashing – Listen in der Praxis

Hashing – Listen in der Praxis

Vorteile

Hashing – Listen in der Praxis

Vorteile

- Einfach zu implementieren

Hashing – Listen in der Praxis

Vorteile

- Einfach zu implementieren
- Robust

Hashing – Listen in der Praxis

Vorteile

- Einfach zu implementieren
- Robust
- Stabilität der Adressen der Einträge selbst bei Rehashing (z.B. Vergrößerung der Tabelle)

Hashing – Listen in der Praxis

Vorteile

- Einfach zu implementieren
- Robust
- Stabilität der Adressen der Einträge selbst bei Rehashing (z.B. Vergrößerung der Tabelle)

Nachteile

Hashing – Listen in der Praxis

Vorteile

- Einfach zu implementieren
- Robust
- Stabilität der Adressen der Einträge selbst bei Rehashing (z.B. Vergrößerung der Tabelle)

Nachteile

- Zusätzliche Indirektion (siehe Pointer nach, siehe dann Element nach, …): Laufzeitkosten

Hashing – Listen in der Praxis

Vorteile

- Einfach zu implementieren
- Robust
- Stabilität der Adressen der Einträge selbst bei Rehashing (z.B. Vergrößerung der Tabelle)

Nachteile

- Zusätzliche Indirektion (siehe Pointer nach, siehe dann Element nach, …): Laufzeitkosten
- Zusätzlicher Platzbedarf

Hashing – Listen in der Praxis

Vorteile

- Einfach zu implementieren
- Robust
- Stabilität der Adressen der Einträge selbst bei Rehashing (z.B. Vergrößerung der Tabelle)

Nachteile

- Zusätzliche Indirektion (siehe Pointer nach, siehe dann Element nach, …): Laufzeitkosten
- Zusätzlicher Platzbedarf
- Zusätzliche dynamische Allokationen (kann man in den Griff kriegen)

Hashing – Listen in der Praxis

Vorteile

- Einfach zu implementieren
- Robust
- Stabilität der Adressen der Einträge selbst bei Rehashing (z.B. Vergrößerung der Tabelle)

Nachteile

- Zusätzliche Indirektion (siehe Pointer nach, siehe dann Element nach, …): Laufzeitkosten
- Zusätzlicher Platzbedarf
- Zusätzliche dynamische Allokationen (kann man in den Griff kriegen)
- Iteration der Tabelle ohne Extraaufwand langsam:

Hashing – Listen in der Praxis

Vorteile

- Einfach zu implementieren
- Robust
- Stabilität der Adressen der Einträge selbst bei Rehashing (z.B. Vergrößerung der Tabelle)

Nachteile

- Zusätzliche Indirektion (siehe Pointer nach, siehe dann Element nach, …): Laufzeitkosten
- Zusätzlicher Platzbedarf
- Zusätzliche dynamische Allokationen (kann man in den Griff kriegen)
- Iteration der Tabelle ohne Extraaufwand langsam:
  - Betrachte jeden Slot (‘Bucket’)

Hashing – Listen in der Praxis

Vorteile

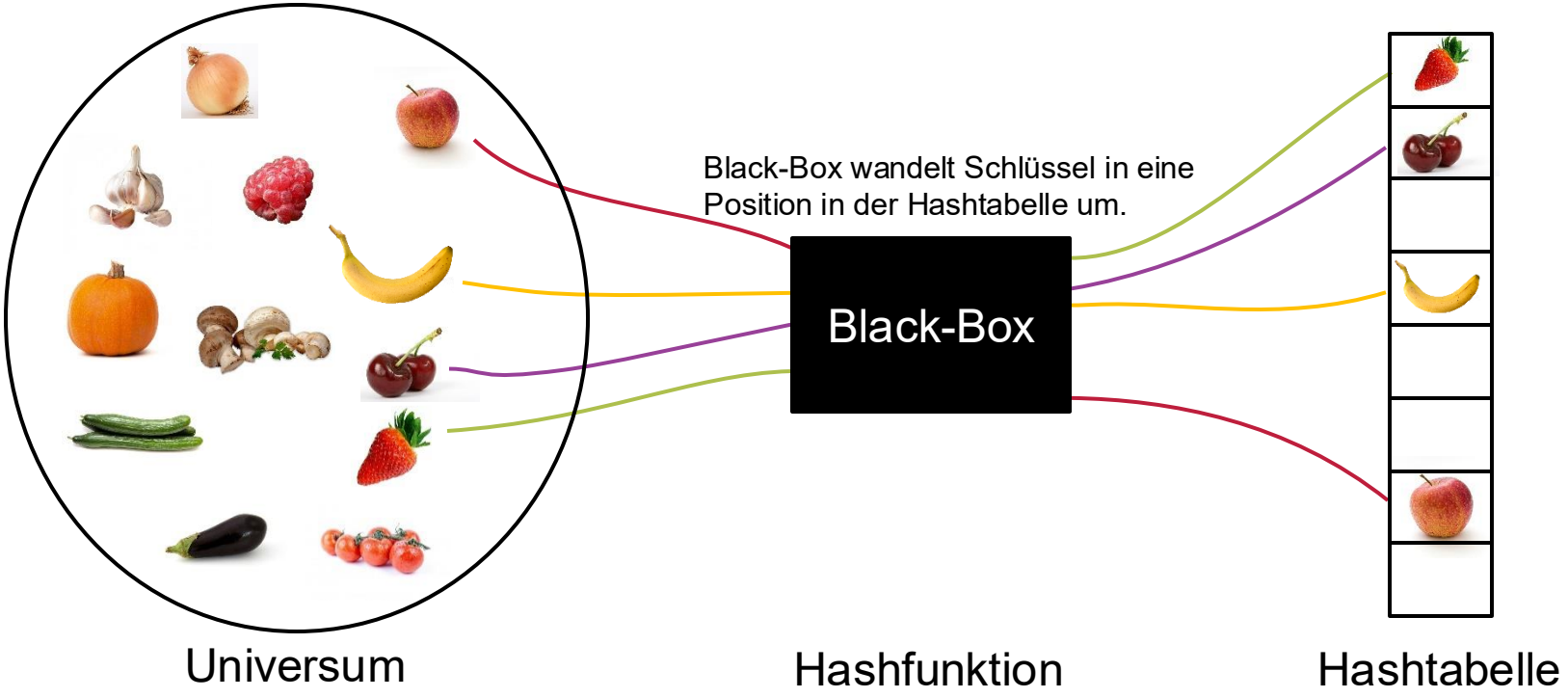
- Einfach zu implementieren
- Robust
- Stabilität der Adressen der Einträge selbst bei Rehashing (z.B. Vergrößerung der Tabelle)

Nachteile

- Zusätzliche Indirektion (siehe Pointer nach, siehe dann Element nach, …): Laufzeitkosten
- Zusätzlicher Platzbedarf
- Zusätzliche dynamische Allokationen (kann man in den Griff kriegen)
- Iteration der Tabelle ohne Extraaufwand langsam:
  - Betrachte jeden Slot (‘Bucket’)
  - Dann jeweils Iteration durch verkettete Liste

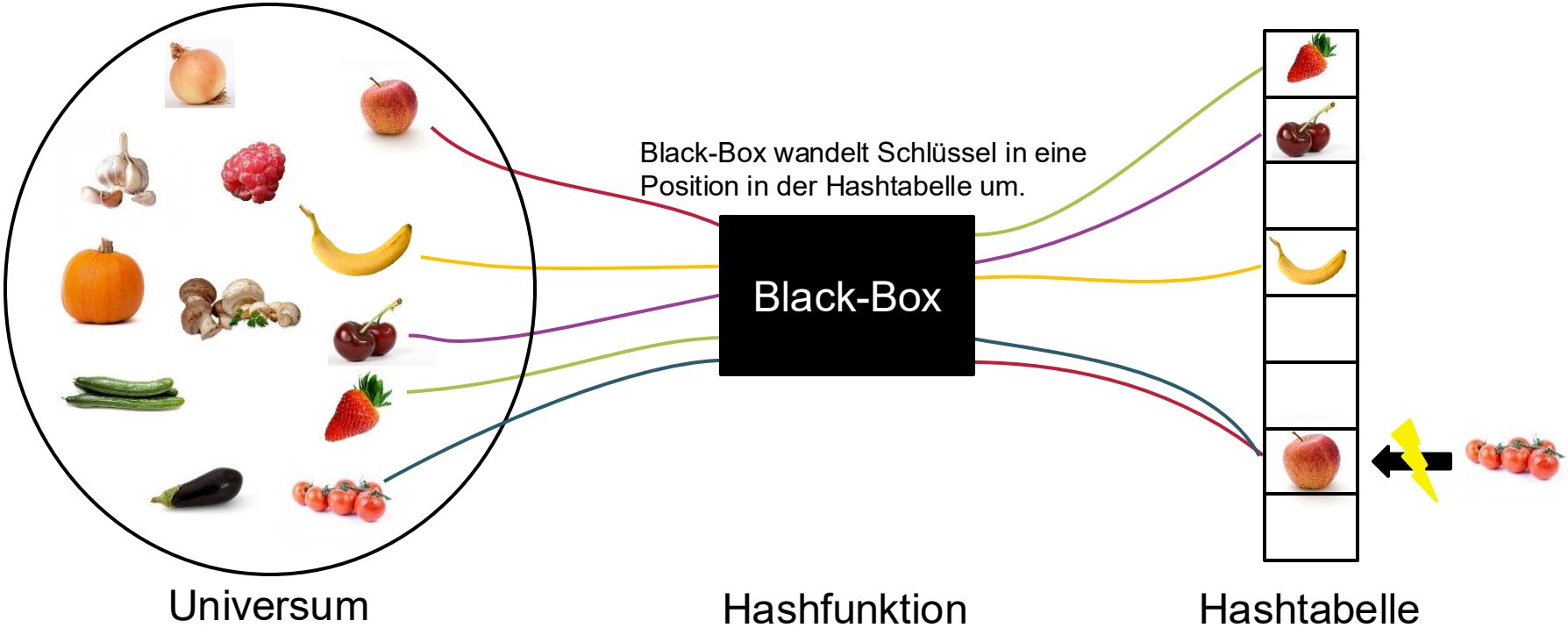
Hashing – offene Adressierung (offenes Hashing)

Jedes Objekt besitzt einen Schlüssel



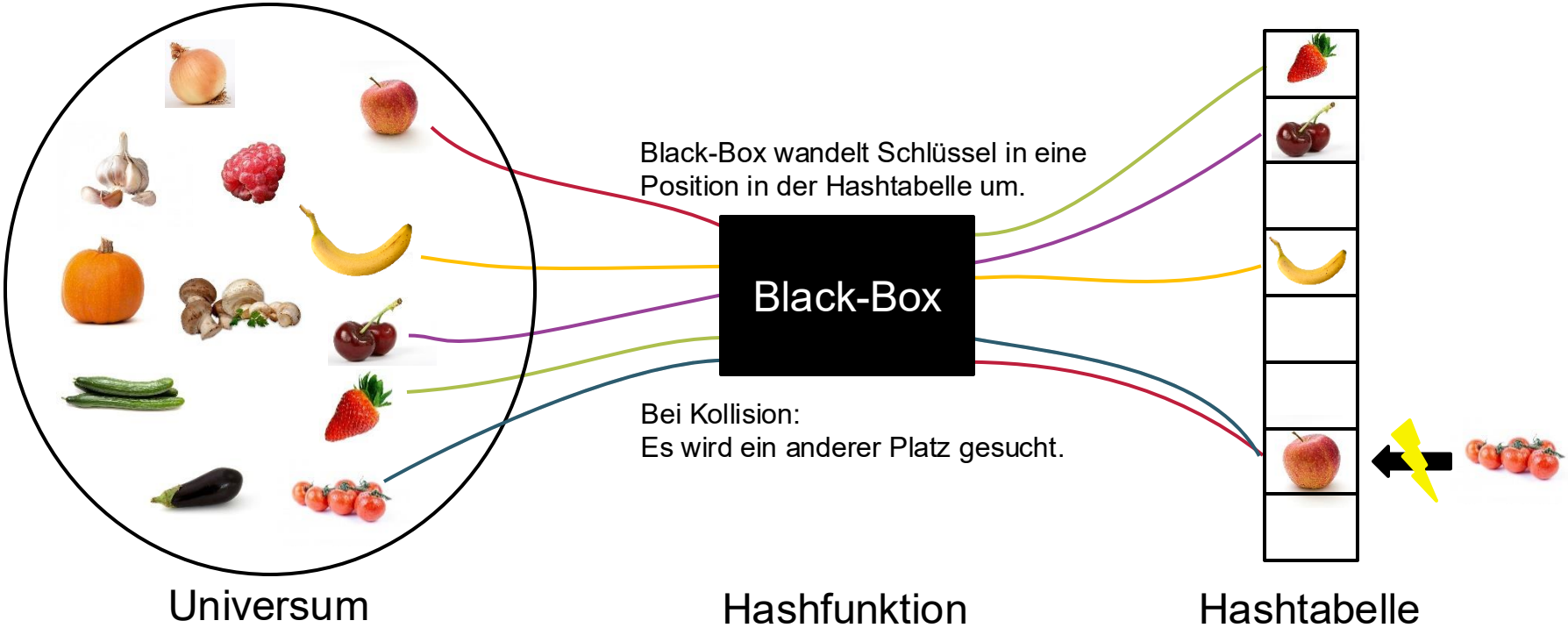
Hashing – offene Adressierung (offenes Hashing)

Jedes Objekt besitzt einen Schlüssel



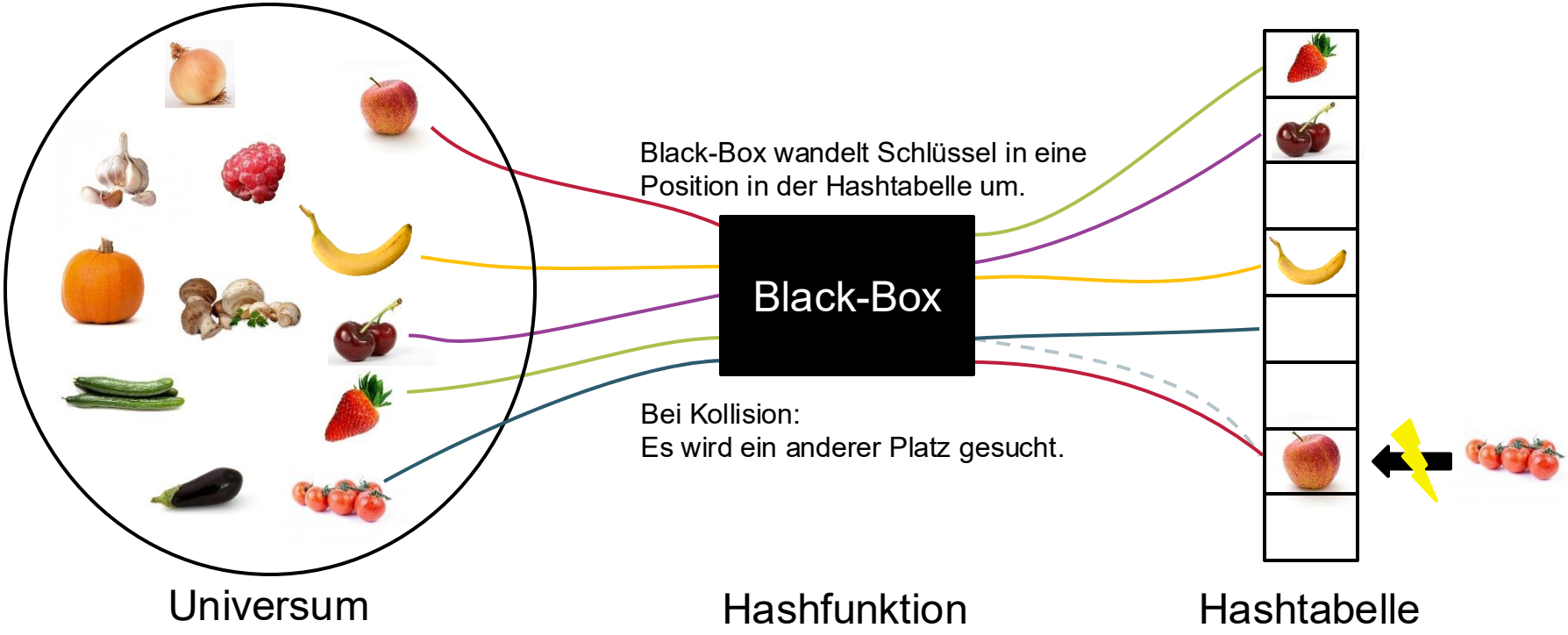
Hashing – offene Adressierung (offenes Hashing)

Jedes Objekt besitzt einen Schlüssel



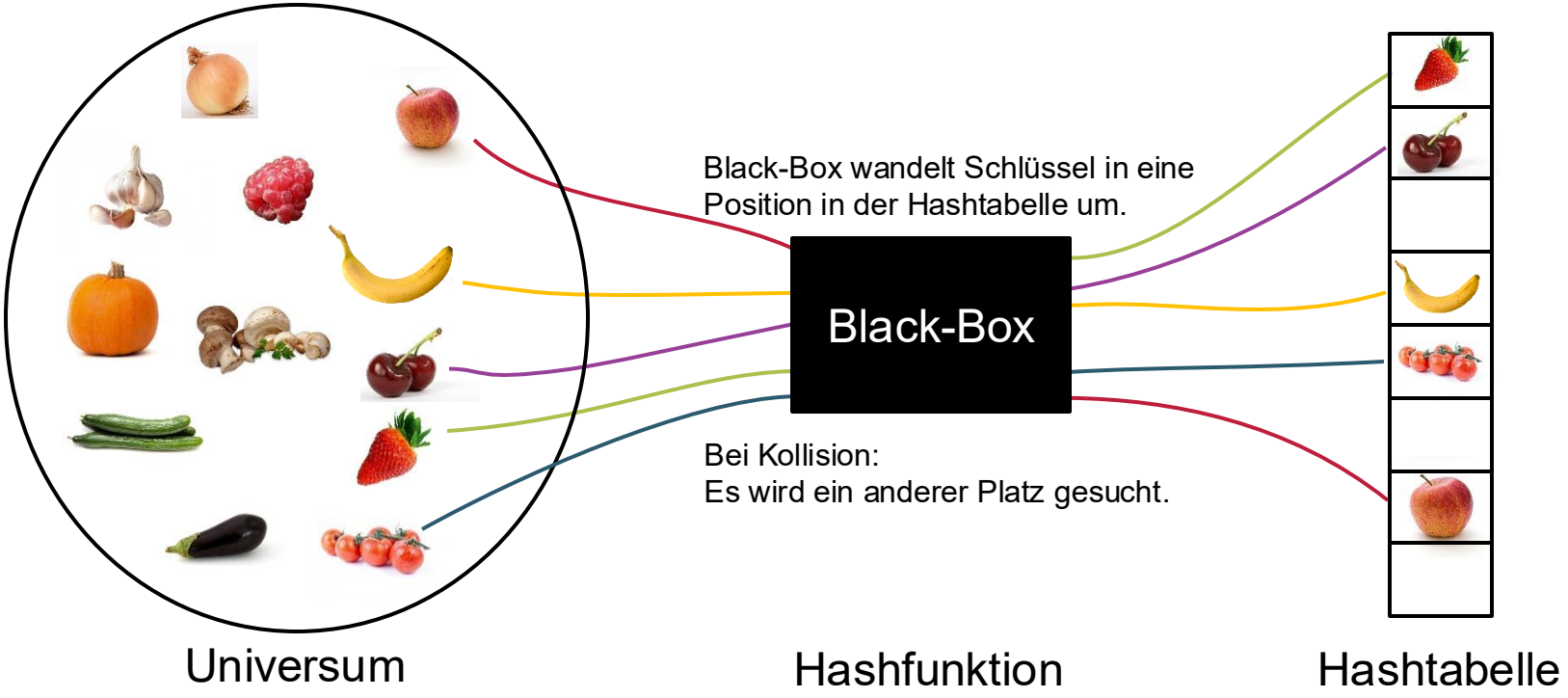
Hashing – offene Adressierung (offenes Hashing)

Jedes Objekt besitzt einen Schlüssel



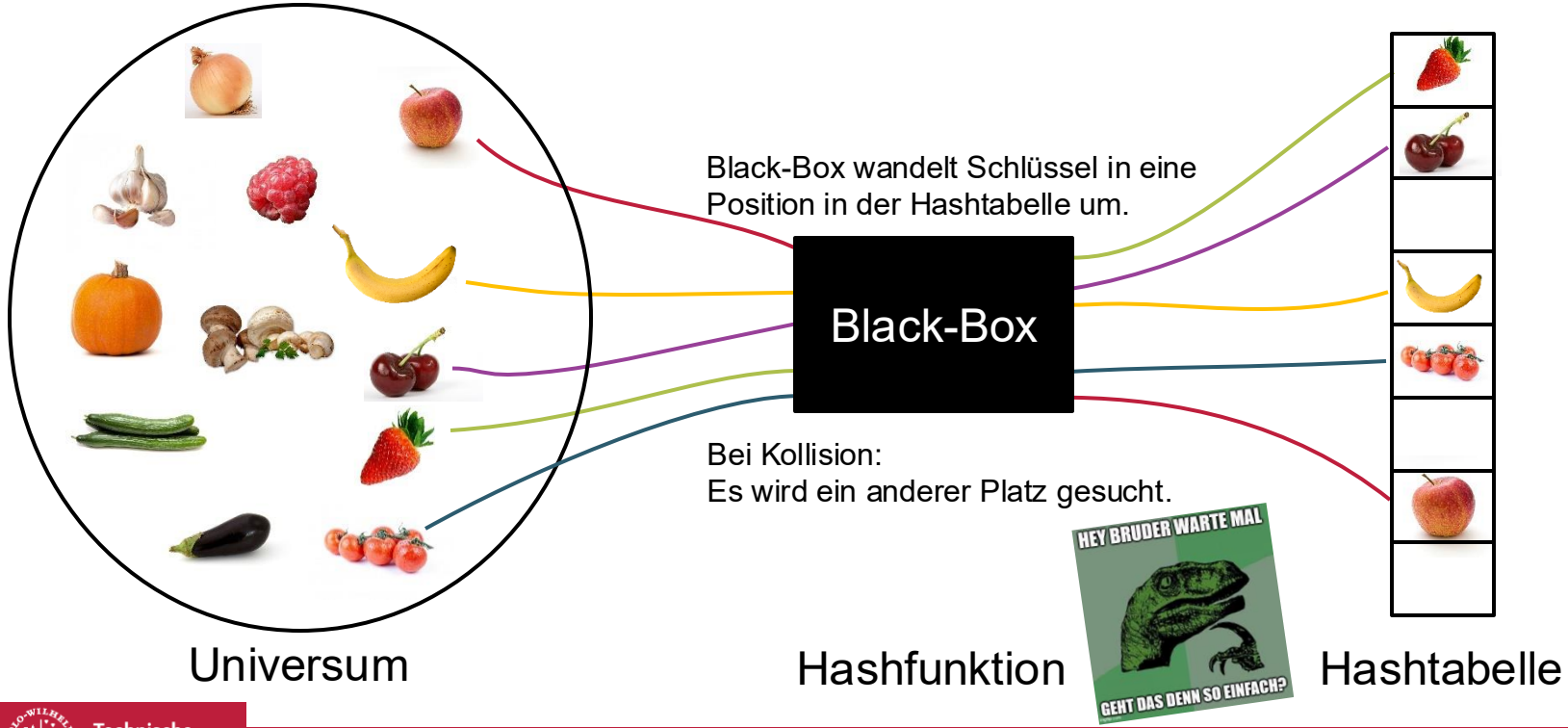
Hashing – offene Adressierung (offenes Hashing)

Jedes Objekt besitzt einen Schlüssel



Hashing – offene Adressierung (offenes Hashing)

Jedes Objekt besitzt einen Schlüssel



… aber hat das nicht Konsequenzen?

… aber hat das nicht Konsequenzen?

- Bei offenem Hashing ist nicht mehr eindeutig klar, wo in der Tabelle ein Schlüssel landet

… aber hat das nicht Konsequenzen?

- Bei offenem Hashing ist nicht mehr eindeutig klar, wo in der Tabelle ein Schlüssel landet
- … das heißt, alle drei Operationen

… aber hat das nicht Konsequenzen?

- Bei offenem Hashing ist nicht mehr eindeutig klar, wo in der Tabelle ein Schlüssel landet
- … das heißt, alle drei Operationen
  - INSERT

… aber hat das nicht Konsequenzen?

- Bei offenem Hashing ist nicht mehr eindeutig klar, wo in der Tabelle ein Schlüssel landet
- … das heißt, alle drei Operationen
  - INSERT
  - SEARCH

… aber hat das nicht Konsequenzen?

- Bei offenem Hashing ist nicht mehr eindeutig klar, wo in der Tabelle ein Schlüssel landet
- … das heißt, alle drei Operationen
  - INSERT
  - SEARCH
  - DELETE

… aber hat das nicht Konsequenzen?

- Bei offenem Hashing ist nicht mehr eindeutig klar, wo in der Tabelle ein Schlüssel landet
- … das heißt, alle drei Operationen
  - INSERT
  - SEARCH
  - DELETE
- … müssen angepasst werden!

Hashfunktionen für offenes Hashing

Hashfunktionen für offenes Hashing

Hashfunktion nicht mehr nur Mapping Key → Hash, $x \mapsto h(x)$

Hashfunktionen für offenes Hashing

Hashfunktion nicht mehr nur Mapping Key → Hash, $x \mapsto h(x)$

Sondern: Wir müssen auch bestimmen, wie wir sondieren

Hashfunktionen für offenes Hashing

Hashfunktion nicht mehr nur Mapping Key → Hash, $x \mapsto h(x)$

Sondern: Wir müssen auch bestimmen, wie wir sondieren

Also: Wo schauen wir nach i fehlgeschlagenen Lookups?

Hashfunktionen für offenes Hashing

Hashfunktion nicht mehr nur Mapping Key → Hash, $x \mapsto h(x)$

Sondern: Wir müssen auch bestimmen, wie wir sondieren

Also: Wo schauen wir nach i fehlgeschlagenen Lookups?

Hashfunktion hier also jetzt Mapping (Key, i) → Index, $(x, i) \mapsto t(x, i)$

Hashfunktionen für offenes Hashing

Hashfunktion nicht mehr nur Mapping Key → Hash, $x \mapsto h(x)$

Sondern: Wir müssen auch bestimmen, wie wir sondieren

Also: Wo schauen wir nach i fehlgeschlagenen Lookups?

Hashfunktion hier also jetzt Mapping (Key, i) → Index, $(x, i) \mapsto t(x, i)$

Wer werden verschiedene Ideen in der Vorlesung sehen:

Hashfunktionen für offenes Hashing

Hashfunktion nicht mehr nur Mapping Key → Hash, $x \mapsto h(x)$

Sondern: Wir müssen auch bestimmen, wie wir sondieren

Also: Wo schauen wir nach i fehlgeschlagenen Lookups?

Hashfunktion hier also jetzt Mapping (Key, i) → Index, $(x, i) \mapsto t(x, i)$

Wer werden verschiedene Ideen in der Vorlesung sehen:

- Lineare Sondierung

Hashfunktionen für offenes Hashing

Hashfunktion nicht mehr nur Mapping Key → Hash, $x \mapsto h(x)$

Sondern: Wir müssen auch bestimmen, wie wir sondieren

Also: Wo schauen wir nach i fehlgeschlagenen Lookups?

Hashfunktion hier also jetzt Mapping (Key, i) → Index, $(x, i) \mapsto t(x, i)$

Wer werden verschiedene Ideen in der Vorlesung sehen:

- Lineare Sondierung
- Quadratische Sondierung

Hashfunktionen für offenes Hashing

Hashfunktion nicht mehr nur Mapping Key → Hash, $x \mapsto h(x)$

Sondern: Wir müssen auch bestimmen, wie wir sondieren

Also: Wo schauen wir nach i fehlgeschlagenen Lookups?

Hashfunktion hier also jetzt Mapping (Key, i) → Index, $(x, i) \mapsto t(x, i)$

Wer werden verschiedene Ideen in der Vorlesung sehen:

- Lineare Sondierung
- Quadratische Sondierung
- Multiplikative Sondierung

Hashfunktionen für offenes Hashing

Hashfunktion nicht mehr nur Mapping Key → Hash, $x \mapsto h(x)$

Sondern: Wir müssen auch bestimmen, wie wir sondieren

Also: Wo schauen wir nach i fehlgeschlagenen Lookups?

Hashfunktion hier also jetzt Mapping (Key, i) → Index, $(x, i) \mapsto t(x, i)$

Wer werden verschiedene Ideen in der Vorlesung sehen:

- Lineare Sondierung
- Quadratische Sondierung
- Multiplikative Sondierung
- Doppeltes Hashing

Hashfunktionen für offenes Hashing

Hashfunktion nicht mehr nur Mapping Key → Hash, $x \mapsto h(x)$

Sondern: Wir müssen auch bestimmen, wie wir sondieren

Also: Wo schauen wir nach i fehlgeschlagenen Lookups?

Hashfunktion hier also jetzt Mapping (Key, i) → Index, $(x, i) \mapsto t(x, i)$

Wer werden verschiedene Ideen in der Vorlesung sehen:

- Lineare Sondierung
- Quadratische Sondierung
- Multiplikative Sondierung
- Doppeltes Hashing

In Aufgaben: Üblicherweise explizit angegeben

Hashfunktionen – Modulo

Hashfunktionen – Modulo

Für $m \in \mathbb{N}, x \in \mathbb{Z}$ und $r \in \{0, \dots, m-1\}$ ist

Hashfunktionen – Modulo

Für $m \in \mathbb{N}, x \in \mathbb{Z}$ und $r \in \{0, \dots, m-1\}$ ist

$$x \bmod m = r,$$

Hashfunktionen – Modulo

Für $m \in \mathbb{N}, x \in \mathbb{Z}$ und $r \in \{0, \dots, m-1\}$ ist

$$x \bmod m = r,$$

falls eine Zahl $q \in \mathbb{Z}$ existiert, sodass gilt

Hashfunktionen – Modulo

Für $m \in \mathbb{N}, x \in \mathbb{Z}$ und $r \in \{0, \dots, m-1\}$ ist

$$x \bmod m = r,$$

falls eine Zahl $q \in \mathbb{Z}$ existiert, sodass gilt

$$x = q \cdot m + r.$$

Hashfunktionen – Modulo

Für $m \in \mathbb{N}, x \in \mathbb{Z}$ und $r \in \{0, \dots, m-1\}$ ist

$$x \bmod m = r,$$

falls eine Zahl $q \in \mathbb{Z}$ existiert, sodass gilt

$$x = q \cdot m + r.$$

Damit ist auch für beliebiges $i \in \mathbb{Z}$

Hashfunktionen – Modulo

Für $m \in \mathbb{N}, x \in \mathbb{Z}$ und $r \in \{0, \dots, m-1\}$ ist

$$x \bmod m = r,$$

falls eine Zahl $q \in \mathbb{Z}$ existiert, sodass gilt

$$x = q \cdot m + r.$$

Damit ist auch für beliebiges $i \in \mathbb{Z}$

$$(x + i \cdot m) \bmod m = x \bmod m.$$

Hashfunktionen – Modulo

Für $m \in \mathbb{N}, x \in \mathbb{Z}$ und $r \in \{0, \dots, m-1\}$ ist

$$x \bmod m = r,$$

falls eine Zahl $q \in \mathbb{Z}$ existiert, sodass gilt

$$x = q \cdot m + r.$$

Damit ist auch für beliebiges $i \in \mathbb{Z}$

$$(x + i \cdot m) \bmod m = x \bmod m.$$

Beispiel:

Hashfunktionen – Modulo

Für $m \in \mathbb{N}, x \in \mathbb{Z}$ und $r \in \{0, \dots, m-1\}$ ist

$$x \bmod m = r,$$

falls eine Zahl $q \in \mathbb{Z}$ existiert, sodass gilt

$$x = q \cdot m + r.$$

Damit ist auch für beliebiges $i \in \mathbb{Z}$

$$(x + i \cdot m) \bmod m = x \bmod m.$$

Beispiel:

$$27 \bmod 7 = 6$$

Hashfunktionen – Modulo

Für $m \in \mathbb{N}, x \in \mathbb{Z}$ und $r \in \{0, \dots, m-1\}$ ist

$$x \bmod m = r,$$

falls eine Zahl $q \in \mathbb{Z}$ existiert, sodass gilt

$$x = q \cdot m + r.$$

Damit ist auch für beliebiges $i \in \mathbb{Z}$

$$(x + i \cdot m) \bmod m = x \bmod m.$$

Beispiel:

$$27 \bmod 7 = 6$$
$$31 \bmod 9 = 4$$

Hashfunktionen – Modulo

Für $m \in \mathbb{N}, x \in \mathbb{Z}$ und $r \in \{0, \dots, m-1\}$ ist

$$x \bmod m = r,$$

falls eine Zahl $q \in \mathbb{Z}$ existiert, sodass gilt

$$x = q \cdot m + r.$$

Damit ist auch für beliebiges $i \in \mathbb{Z}$

$$(x + i \cdot m) \bmod m = x \bmod m.$$

Beispiel:

$$27 \bmod 7 = 6$$
$$31 \bmod 9 = 4$$

Man kann zeigen:

Hashfunktionen – Modulo

Für $m \in \mathbb{N}, x \in \mathbb{Z}$ und $r \in \{0, \dots, m-1\}$ ist

$$x \bmod m = r,$$

falls eine Zahl $q \in \mathbb{Z}$ existiert, sodass gilt

$$x = q \cdot m + r.$$

Damit ist auch für beliebiges $i \in \mathbb{Z}$

$$(x + i \cdot m) \bmod m = x \bmod m.$$

Beispiel:

$$27 \bmod 7 = 6$$
$$31 \bmod 9 = 4$$

Man kann zeigen:

$$(a + b) \bmod m = \big((a \bmod m) + (b \bmod m)\big) \bmod m$$

Hashfunktionen – Modulo

Für $m \in \mathbb{N}, x \in \mathbb{Z}$ und $r \in \{0, \dots, m-1\}$ ist

$$x \bmod m = r,$$

falls eine Zahl $q \in \mathbb{Z}$ existiert, sodass gilt

$$x = q \cdot m + r.$$

Damit ist auch für beliebiges $i \in \mathbb{Z}$

$$(x + i \cdot m) \bmod m = x \bmod m.$$

Beispiel:

$$27 \bmod 7 = 6$$
$$31 \bmod 9 = 4$$

Man kann zeigen:

$$(a + b) \bmod m = \big((a \bmod m) + (b \bmod m)\big) \bmod m$$
$$(a \cdot b) \bmod m = \big((a \bmod m) \cdot (b \bmod m)\big) \bmod m$$

Klassische Aufgabenstellung

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

Dabei ist x ein einzusetzender Schlüssel und i die Nummer des Versuches, x in eine unbesetzte Speicherzelle des Arrays zu schreiben, beginnend bei $i = 0$. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

Dabei ist x ein einzusetzender Schlüssel und i die Nummer des Versuches, x in eine unbesetzte Speicherzelle des Arrays zu schreiben, beginnend bei $i = 0$. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

26, 152, 4

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

Dabei ist x ein einzusetzender Schlüssel und i die Nummer des Versuches, x in eine unbesetzte Speicherzelle des Arrays zu schreiben, beginnend bei $i = 0$. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

26, 152, 4

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

Dabei ist x ein einzusetzender Schlüssel und i die Nummer des Versuches, x in eine unbesetzte Speicherzelle des Arrays zu schreiben, beginnend bei $i = 0$. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

26, 152, 4

- Was für ein Sondieren ist das hier?

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

Dabei ist x ein einzusetzender Schlüssel und i die Nummer des Versuches, x in eine unbesetzte Speicherzelle des Arrays zu schreiben, beginnend bei $i = 0$. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

26, 152, 4

- Was für ein Sondieren ist das hier?
  - Lineares Sondieren

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

Dabei ist x ein einzusetzender Schlüssel und i die Nummer des Versuches, x in eine unbesetzte Speicherzelle des Arrays zu schreiben, beginnend bei $i = 0$. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

26, 152, 4

- Was für ein Sondieren ist das hier?
  - Lineares Sondieren
  - Quadratisches Sondieren

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

Dabei ist x ein einzusetzender Schlüssel und i die Nummer des Versuches, x in eine unbesetzte Speicherzelle des Arrays zu schreiben, beginnend bei $i = 0$. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

26, 152, 4

- Was für ein Sondieren ist das hier?
  - Lineares Sondieren
  - Quadratisches Sondieren
  - Multiplikatives Sondieren

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

Dabei ist x ein einzusetzender Schlüssel und i die Nummer des Versuches, x in eine unbesetzte Speicherzelle des Arrays zu schreiben, beginnend bei $i = 0$. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

26, 152, 4

- Was für ein Sondieren ist das hier?
  - Lineares Sondieren
  - Quadratisches Sondieren
  - Multiplikatives Sondieren
  - Doppeltes Hashing

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

Dabei ist x ein einzusetzender Schlüssel und i die Nummer des Versuches, x in eine unbesetzte Speicherzelle des Arrays zu schreiben, beginnend bei $i = 0$. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

26, 152, 4

- Was für ein Sondieren ist das hier?
  - Lineares Sondieren
  - Quadratisches Sondieren
  - Multiplikatives Sondieren
  - Doppeltes Hashing

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

Dabei ist x ein einzusetzender Schlüssel und i die Nummer des Versuches, x in eine unbesetzte Speicherzelle des Arrays zu schreiben, beginnend bei $i = 0$. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

26, 152, 4

- Was für ein Sondieren ist das hier?
  - Lineares Sondieren
  - Quadratisches Sondieren
  - Multiplikatives Sondieren
  - Doppeltes Hashing
- Welches Problem hat die zweite Hashfunktion $(3x + 1) \cdot i \bmod 11$?

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

Dabei ist x ein einzusetzender Schlüssel und i die Nummer des Versuches, x in eine unbesetzte Speicherzelle des Arrays zu schreiben, beginnend bei $i = 0$. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

26, 152, 4

- Was für ein Sondieren ist das hier?
  - Lineares Sondieren
  - Quadratisches Sondieren
  - Multiplikatives Sondieren
  - Doppeltes Hashing
- Welches Problem hat die zweite Hashfunktion $(3x + 1) \cdot i \bmod 11$?
  - Kann 0 sein, dann finden wir keine alternativen Positionen!

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

Dabei ist x ein einzusetzender Schlüssel und i die Nummer des Versuches, x in eine unbesetzte Speicherzelle des Arrays zu schreiben, beginnend bei $i = 0$. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

26, 152, 4

- Was für ein Sondieren ist das hier?
  - Lineares Sondieren
  - Quadratisches Sondieren
  - Multiplikatives Sondieren
  - Doppeltes Hashing
- Welches Problem hat die zweite Hashfunktion $(3x + 1) \cdot i \bmod 11$?
  - Kann 0 sein, dann finden wir keine alternativen Positionen!
  - Hier zu einfach gehalten aus Demonstrationszwecken

Klassische Aufgabenstellung

Betrachte ein anfangs leeres Array A der Größe 11, es gibt also die Speicherzellen $A[0], \ldots, A[10]$. In diesem Array führen wir offenes Hashing mit der folgenden Hashfunktion durch:

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

Dabei ist x ein einzusetzender Schlüssel und i die Nummer des Versuches, x in eine unbesetzte Speicherzelle des Arrays zu schreiben, beginnend bei $i = 0$. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

26, 152, 4

- Was für ein Sondieren ist das hier?
  - Lineares Sondieren
  - Quadratisches Sondieren
  - Multiplikatives Sondieren
  - Doppeltes Hashing
- Welches Problem hat die zweite Hashfunktion $(3x + 1) \cdot i \bmod 11$?
  - Kann 0 sein, dann finden wir keine alternativen Positionen!
  - Hier zu einfach gehalten aus Demonstrationszwecken
  - Passt auf, falls ihr so etwas mal implementiert :)

Klassische Aufgabenstellung

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

Klassische Aufgabenstellung

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]

INSERT 26:

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]

INSERT 26:

- $t(0,26) = 6$

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]

INSERT 26:

- $t(0,26) = 6$
- $7 \cdot 26 = 182$

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]

INSERT 26:

- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]

INSERT 26:

- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26				

INSERT 26:

- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26				

INSERT 26:

- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26				

INSERT 26:

- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:

- $t(0,152) = 8$

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26				

INSERT 26:

- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:

- $t(0,152) = 8$
- $7 \cdot 152 = 1064$

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26				

INSERT 26:

- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:

- $t(0,152) = 8$
- $7 \cdot 152 = 1064$
- $1064 \bmod 11 = 8$

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26				

INSERT 26:

- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:

- $t(0,152) = 8$
- $7 \cdot 152 = 1064$
- $1064 \bmod 11 = 8$
- keine Kollision

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26		152		

INSERT 26:

- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:

- $t(0,152) = 8$
- $7 \cdot 152 = 1064$
- $1064 \bmod 11 = 8$
- keine Kollision

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26		152		

INSERT 26:

- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:

- $t(0,152) = 8$
- $7 \cdot 152 = 1064$
- $1064 \bmod 11 = 8$
- keine Kollision

INSERT 4:

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26		152		

INSERT 26:

- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:

- $t(0,152) = 8$
- $7 \cdot 152 = 1064$
- $1064 \bmod 11 = 8$
- keine Kollision

INSERT 4:

- $t(0,4) = 6$

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26		152		

INSERT 26:
- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:
- $t(0,152) = 8$
- $7 \cdot 152 = 1064$
- $1064 \bmod 11 = 8$
- keine Kollision

INSERT 4:
- $t(0,4) = 6$
- $4 \cdot 7 = 28$

Klassische Aufgabenstellung

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26		152		

INSERT 26:

- $t(0, 26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:

- $t(0, 152) = 8$
- $7 \cdot 152 = 1064$
- $1064 \bmod 11 = 8$
- keine Kollision

INSERT 4:

- $t(0, 4) = 6$
- $4 \cdot 7 = 28$
- $4 \cdot 7 \bmod 11 = 6$

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26		152		

INSERT 26:
- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:
- $t(0,152) = 8$
- $7 \cdot 152 = 1064$
- $1064 \bmod 11 = 8$
- keine Kollision

INSERT 4:
- $t(0,4) = 6$
- $4 \cdot 7 = 28$
- $4 \cdot 7 \bmod 11 = 6$
- Kollision!

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26		152		

INSERT 26:
- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:
- $t(0,152) = 8$
- $7 \cdot 152 = 1064$
- $1064 \bmod 11 = 8$
- keine Kollision

INSERT 4:
- $t(0,4) = 6$
- $4 \cdot 7 = 28$
- $4 \cdot 7 \bmod 11 = 6$
- Kollision!
- $t(1,4) = 8$

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26		152		

INSERT 26:
- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:
- $t(0,152) = 8$
- $7 \cdot 152 = 1064$
- $1064 \bmod 11 = 8$
- keine Kollision

INSERT 4:
- $t(0,4) = 6$
- $4 \cdot 7 = 28$
- $4 \cdot 7 \bmod 11 = 6$
- Kollision!
- $t(1,4) = 8$
- Kollision!

Klassische Aufgabenstellung

$$t(i, x) = 7x + (3x + 1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26		152		

INSERT 26:
- $t(0, 26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:
- $t(0, 152) = 8$
- $7 \cdot 152 = 1064$
- $1064 \bmod 11 = 8$
- keine Kollision

INSERT 4:
- $t(0, 4) = 6$
- $4 \cdot 7 = 28$
- $4 \cdot 7 \bmod 11 = 6$
- Kollision!
- $t(1, 4) = 8$
- Kollision!
- $t(2, 4) = 10$

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26		152		

INSERT 26:
- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:
- $t(0,152) = 8$
- $7 \cdot 152 = 1064$
- $1064 \bmod 11 = 8$
- keine Kollision

INSERT 4:
- $t(0,4) = 6$
- $4 \cdot 7 = 28$
- $4 \cdot 7 \bmod 11 = 6$
- Kollision!
- $t(1,4) = 8$
- Kollision!
- $t(2,4) = 10$
- Keine Kollision

Klassische Aufgabenstellung

$$t(i,x) = 7x + (3x+1) \cdot i \bmod 11$$

26, 152, 4

x: Schlüssel
i: Versuch (Start bei 0)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
						26		152		4

INSERT 26:
- $t(0,26) = 6$
- $7 \cdot 26 = 182$
- $182 \bmod 11 = 6$
- keine Kollision

INSERT 152:
- $t(0,152) = 8$
- $7 \cdot 152 = 1064$
- $1064 \bmod 11 = 8$
- keine Kollision

INSERT 4:
- $t(0,4) = 6$
- $4 \cdot 7 = 28$
- $4 \cdot 7 \bmod 11 = 6$
- Kollision!
- $t(1,4) = 8$
- Kollision!
- $t(2,4) = 10$
- Keine Kollision

Hashing in der Praxis

Übliche Vorgehensweise

Übliche Vorgehensweise

In der Praxis: Objekte implementieren Hashfunktion

Übliche Vorgehensweise

In der Praxis: Objekte implementieren Hashfunktion

- Kennen normalerweise nicht die Größe der Tabelle

Übliche Vorgehensweise

In der Praxis: Objekte implementieren Hashfunktion

- Kennen normalerweise nicht die Größe der Tabelle
- Liefern normalerweise Hashwerte im Bereich eines Hardware-Integers (z.B. $0 \leq h(x) < 2^{64}$)

Übliche Vorgehensweise

In der Praxis: Objekte implementieren Hashfunktion

- Kennen normalerweise nicht die Größe der Tabelle
- Liefern normalerweise Hashwerte im Bereich eines Hardware-Integers (z.B. $0 \leq h(x) < 2^{64}$)
- Normalerweise findet die Reduktion Modulo m separat statt

Übliche Vorgehensweise

In der Praxis: Objekte implementieren Hashfunktion

- Kennen normalerweise nicht die Größe der Tabelle
- Liefern normalerweise Hashwerte im Bereich eines Hardware-Integers (z.B. $0 \leq h(x) < 2^{64}$)
- Normalerweise findet die Reduktion Modulo m separat statt
- Dieser Teil wird dann normalerweise von der Hashtabelle implementiert

Übliche Vorgehensweise

In der Praxis: Objekte implementieren Hashfunktion

- Kennen normalerweise nicht die Größe der Tabelle
- Liefern normalerweise Hashwerte im Bereich eines Hardware-Integers (z.B. $0 \leq h(x) < 2^{64}$)
- Normalerweise findet die Reduktion Modulo m separat statt
- Dieser Teil wird dann normalerweise von der Hashtabelle implementiert
- Gleiches gilt üblicherweise für das Sondieren

Hashing in der Praxis

Hashing in der Praxis

In Programmiersprachen:

Hashing in der Praxis

In Programmiersprachen:

- Java (HashMap)

Hashing in der Praxis

In Programmiersprachen:

- Java (HashMap)
- C++ (std::unordered_map): üblicherweise mit Listen

Hashing in der Praxis

In Programmiersprachen:

- Java (HashMap)
- C++ (std::unordered_map): üblicherweise mit Listen
- C++ (absl::flat_hash_map): offen, quadratische Sondierung

Hashing in der Praxis

In Programmiersprachen:

- Java (`HashMap`)
- C++ (`std::unordered_map`): üblicherweise mit Listen
- C++ (`absl::flat_hash_map`): offen, quadratische Sondierung
- Python (`dict`): offen, spezielle Sondierungsfunktion

Hashing in der Praxis

In Programmiersprachen:

- Java (`HashMap`)
- C++ (`std::unordered_map`): üblicherweise mit Listen
- C++ (`absl::flat_hash_map`): offen, quadratische Sondierung
- Python (`dict`): offen, spezielle Sondierungsfunktion
- ...

Hashing in der Praxis

In Programmiersprachen:

- Java (`HashMap`)
- C++ (`std::unordered_map`): üblicherweise mit Listen
- C++ (`absl::flat_hash_map`): offen, quadratische Sondierung
- Python (`dict`): offen, spezielle Sondierungsfunktion
- ...

- Was, wenn wir überhaupt nicht wissen, wie viele Elemente in unserer Hash-Tabelle landen werden?

Hashing in der Praxis

In Programmiersprachen:

- Java (HashMap)
- C++ (std::unordered_map): üblicherweise mit Listen
- C++ (absl::flat_hash_map): offen, quadratische Sondierung
- Python (dict): offen, spezielle Sondierungsfunktion
- ...

- Was, wenn wir überhaupt nicht wissen, wie viele Elemente in unserer Hash-Tabelle landen werden?
- → Rehashing (mittendrin neue Tabelle erstellen, alle Elemente neu eintragen)

Coding-Interview

Coding-Interview

Coding-Interview

Element Uniqueness Problem:

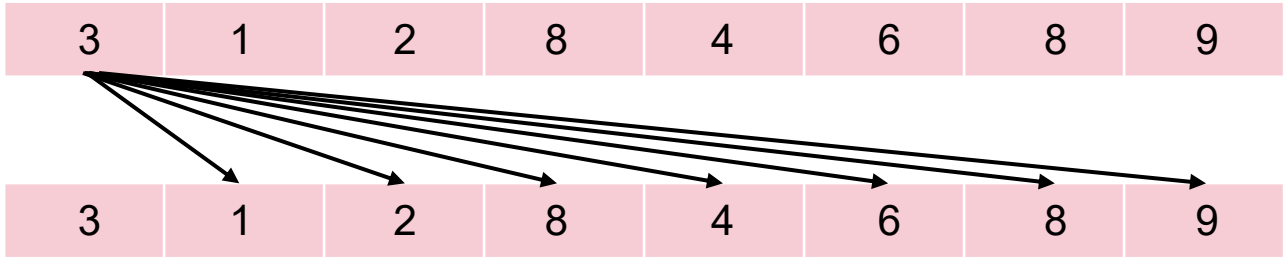
Gegeben: Liste aus Zahlen $l = \{a_1, a_2, \dots, a_n\}$.

Frage: Sind alle Zahlen unterschiedlich?

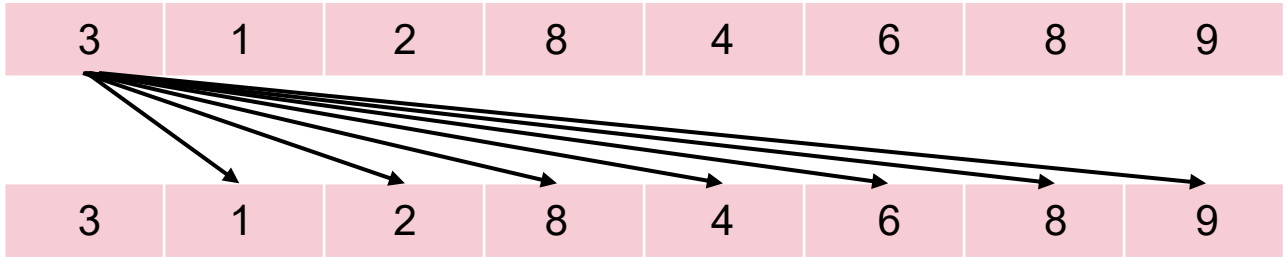
Coding-Interview

3	1	2	8	4	6	8	9

Coding-Interview



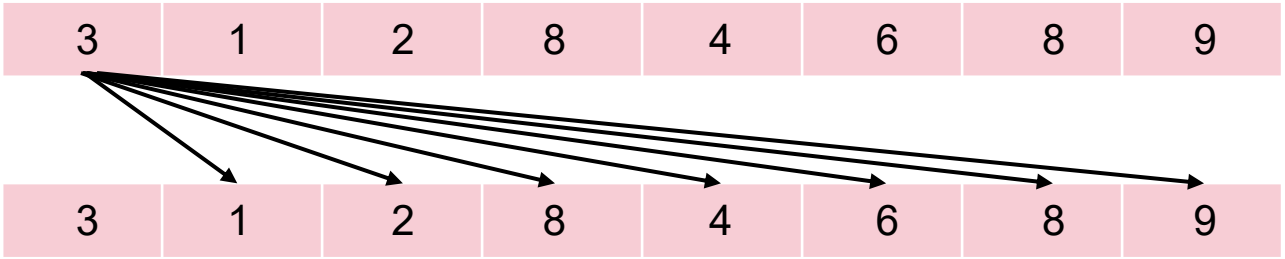
Coding-Interview



Naiver Ansatz:

$$
\begin{aligned}
&for\ i\ in\ 1, \dots, n: \\
&\quad for\ j\ in\ i+1, \dots, n: \\
&\quad\quad if\ l[i] == l[j]: \\
&\quad\quad\quad return\ True \\
&return\ False
\end{aligned}
$$

Coding-Interview



Naiver Ansatz:

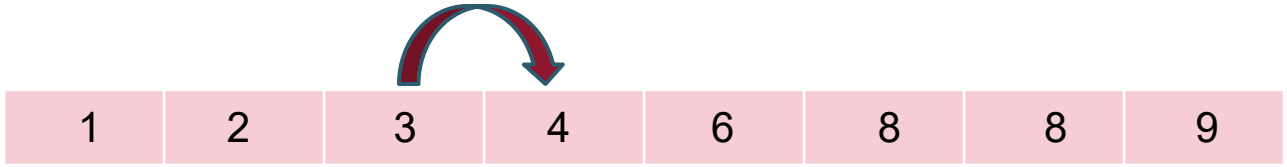
$$
\begin{aligned}
&for\ i\ in\ 1, \dots, n: \\
&\quad for\ j\ in\ i+1, \dots, n: \\
&\quad\quad if\ l[i] == l[j]: \\
&\quad\quad\quad return\ True \\
&return\ False
\end{aligned}
$$

$O(n^2)$

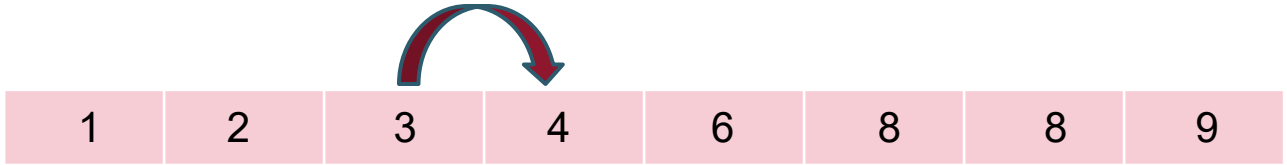
Coding-Interview

3	1	2	8	4	6	8	9

Coding-Interview



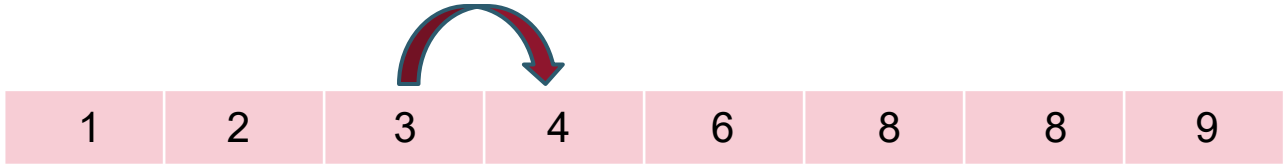
Coding-Interview



Besserer Ansatz:

$$
\begin{aligned}
&sort(l)\\
&for\ i\ in\ 1, \dots, n-1:\\
&\quad if\ l[i] == l[i+1]:\\
&\quad\quad return\ True\\
&return\ False
\end{aligned}
$$

Coding-Interview

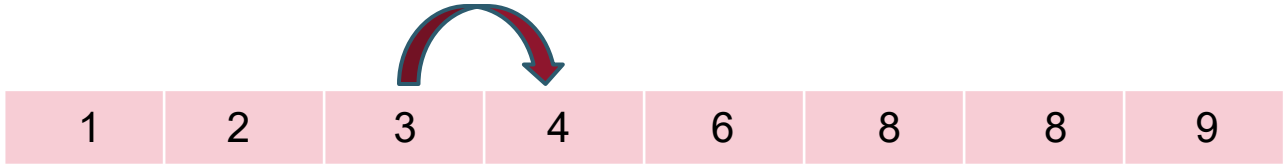


Besserer Ansatz:

$sort(l)$
$for\ i\ in\ 1, \dots, n-1:$
    $if\ l[i] == l[i+1]:$
        $return\ True$
$return\ False$

$O(n \log n)$

Coding-Interview

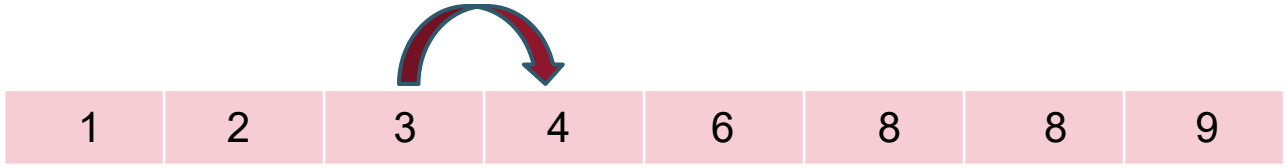


Besserer Ansatz:

Geht das noch besser?

```
if t[i] == t[i + 1]:
    return True
return False
```

Coding-Interview



Besserer Ansatz:

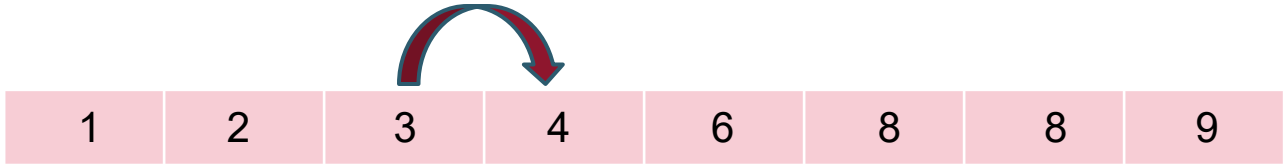
Geht das noch besser?

$if\ i[i] == i[i+1]:$

$retur$

Ja! Hashing :)

Coding-Interview



Besserer Ansatz:

Geht das noch besser?

$if\ t[i] == t[i+1]:$

$retur$

Ja! Hashing :)

Coding-Interview

3	1	2	8	4	6	8	9

Coding-Interview

3	1	2	8	4	6	8	9

Coding-Interview

3	1	2	8	4	6	8	9

	3																						

Coding-Interview

3	1	2	8	4	6	8	9

	3							1															

Coding-Interview

3	1	2	8	4	6	8	9

	3							1		2														

Coding-Interview

3	1	2	8	4	6	8	9

	3							1		2							8							

Coding-Interview

3	1	2	8	4	6	8	9

	3			4				1		2							8							

Coding-Interview

3	1	2	8	4	6	8	9

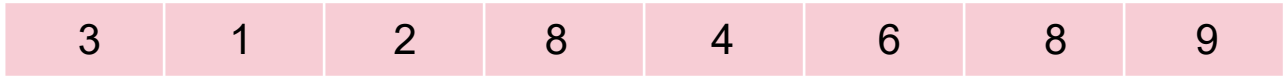
	3			4				1		2							8						6

Coding-Interview

3	1	2	8	4	6	8	9

	3			4				1		2							8						6

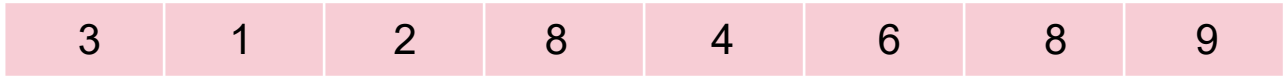
Coding-Interview



Noch besserer Ansatz:

$$h = HashMap()$$
$$for\ elem\ in\ l:$$
$$\quad if\ h.contains(elem):$$
$$\quad\quad return\ True$$
$$\quad h.insert(elem)$$
$$return\ False$$

Coding-Interview



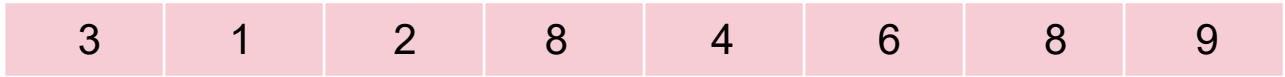
Noch besserer Ansatz:

$$h = HashMap()$$
$$for\ elem\ in\ l:$$
$$\quad if\ h.contains(elem):$$
$$\quad\quad return\ True$$
$$\quad h.insert(elem)$$
$$return\ False$$

Average Laufzeit:

$$O(n)$$

Coding-Interview



Noch besserer Ansatz:

$$h = HashMap()$$
$$for\ elem\ in\ l:$$
$$\quad if\ h.contains(elem):$$
$$\quad\quad return\ True$$
$$\quad h.insert(elem)$$
$$return\ False$$

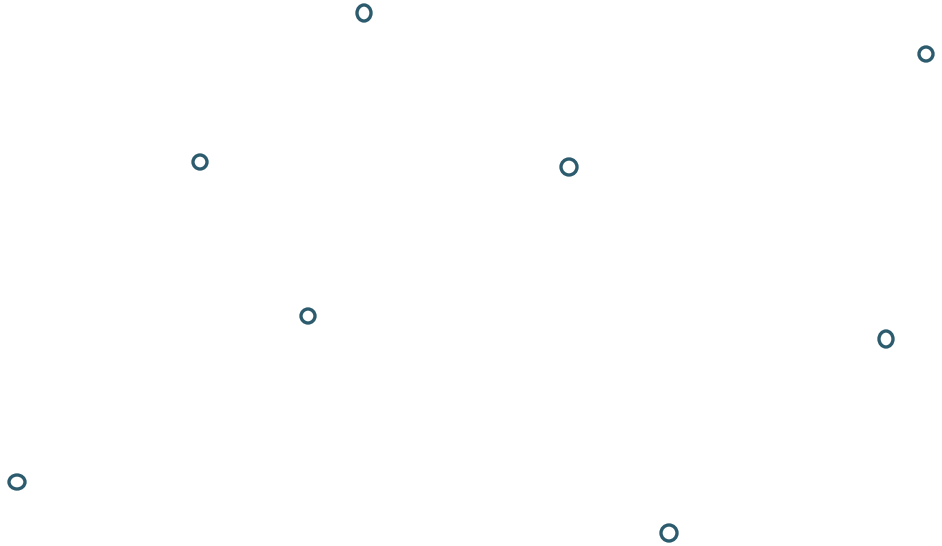
Average Laufzeit:

$$O(n)$$

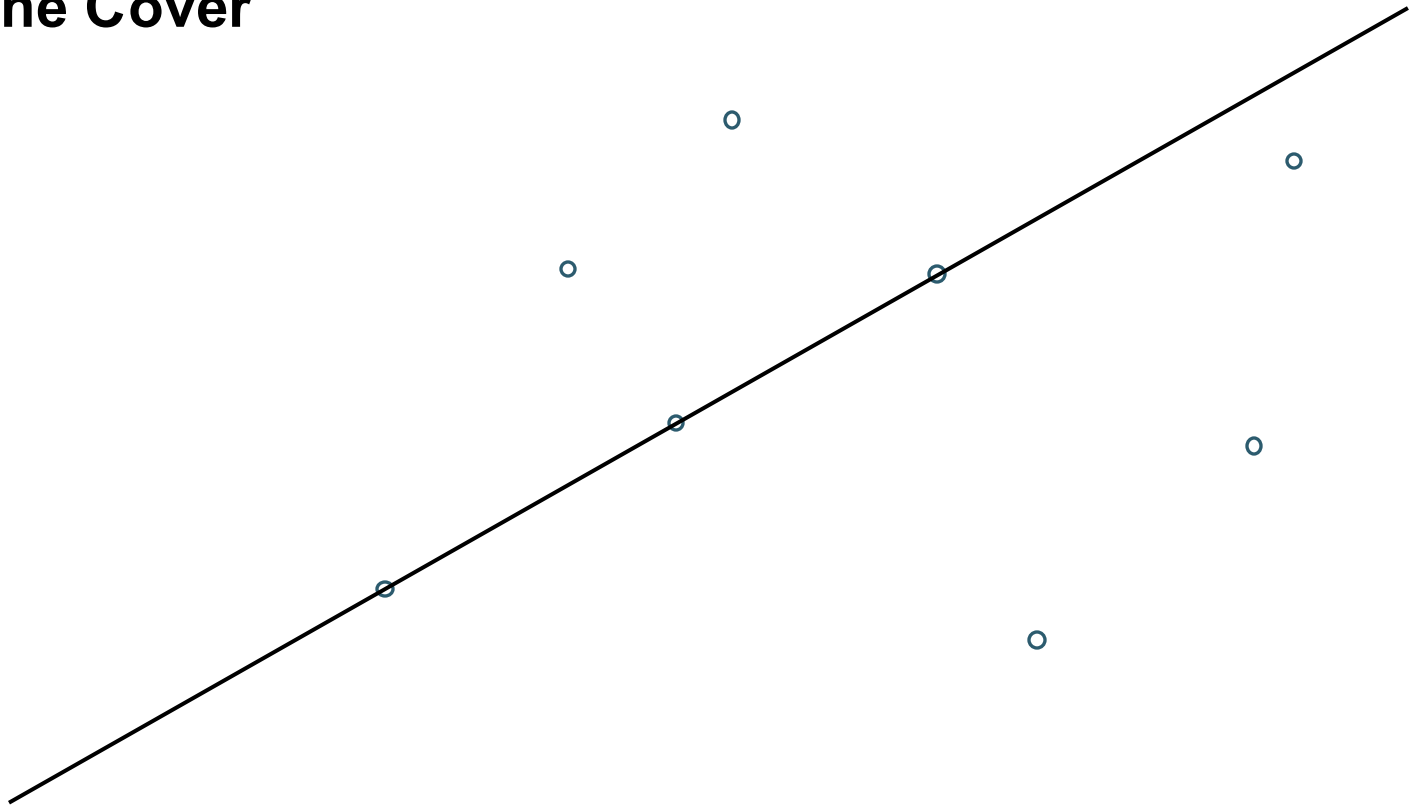
Worstcase Laufzeit:

$$O(n^2)$$

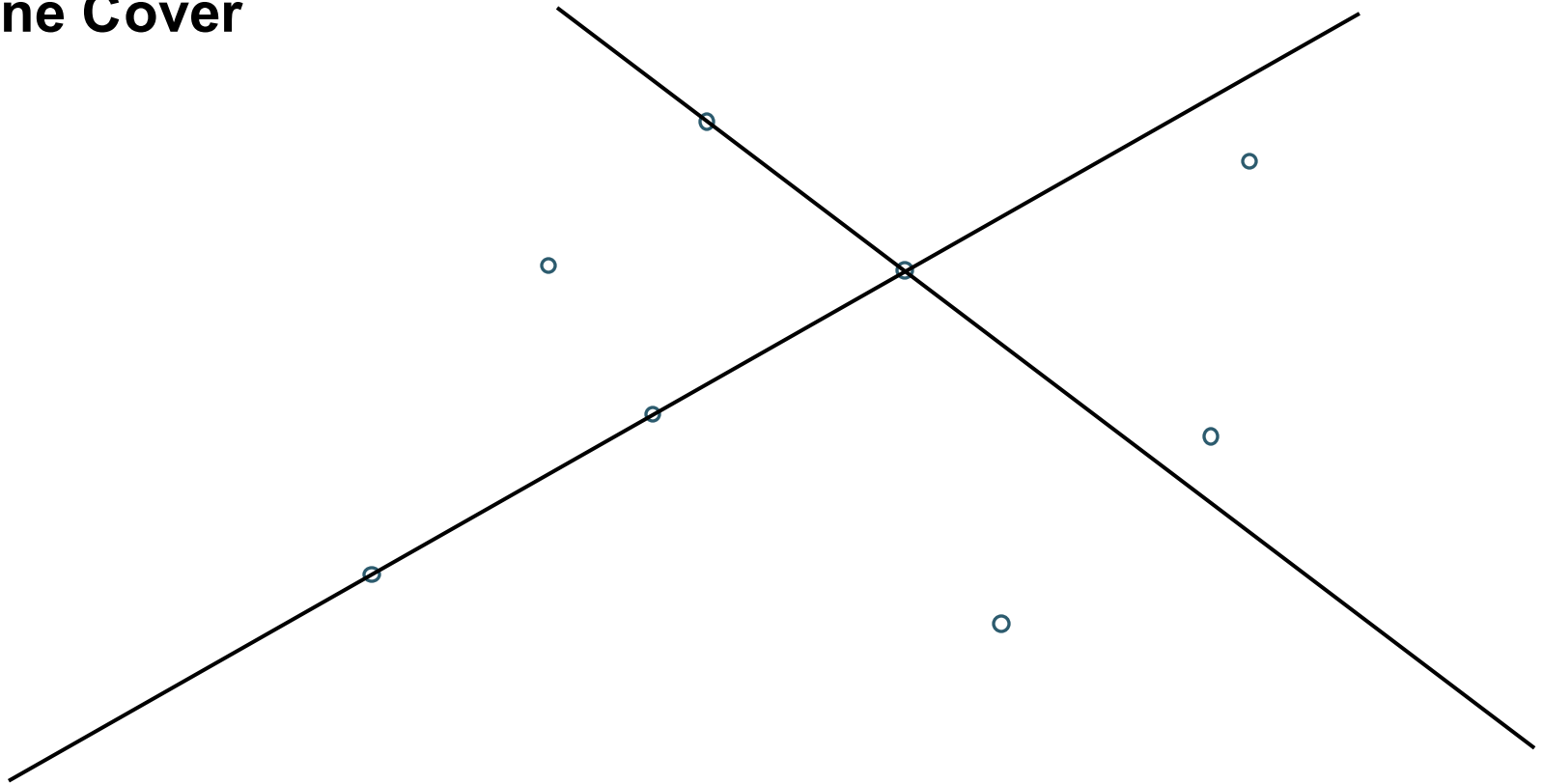
Point Line Cover



Point Line Cover

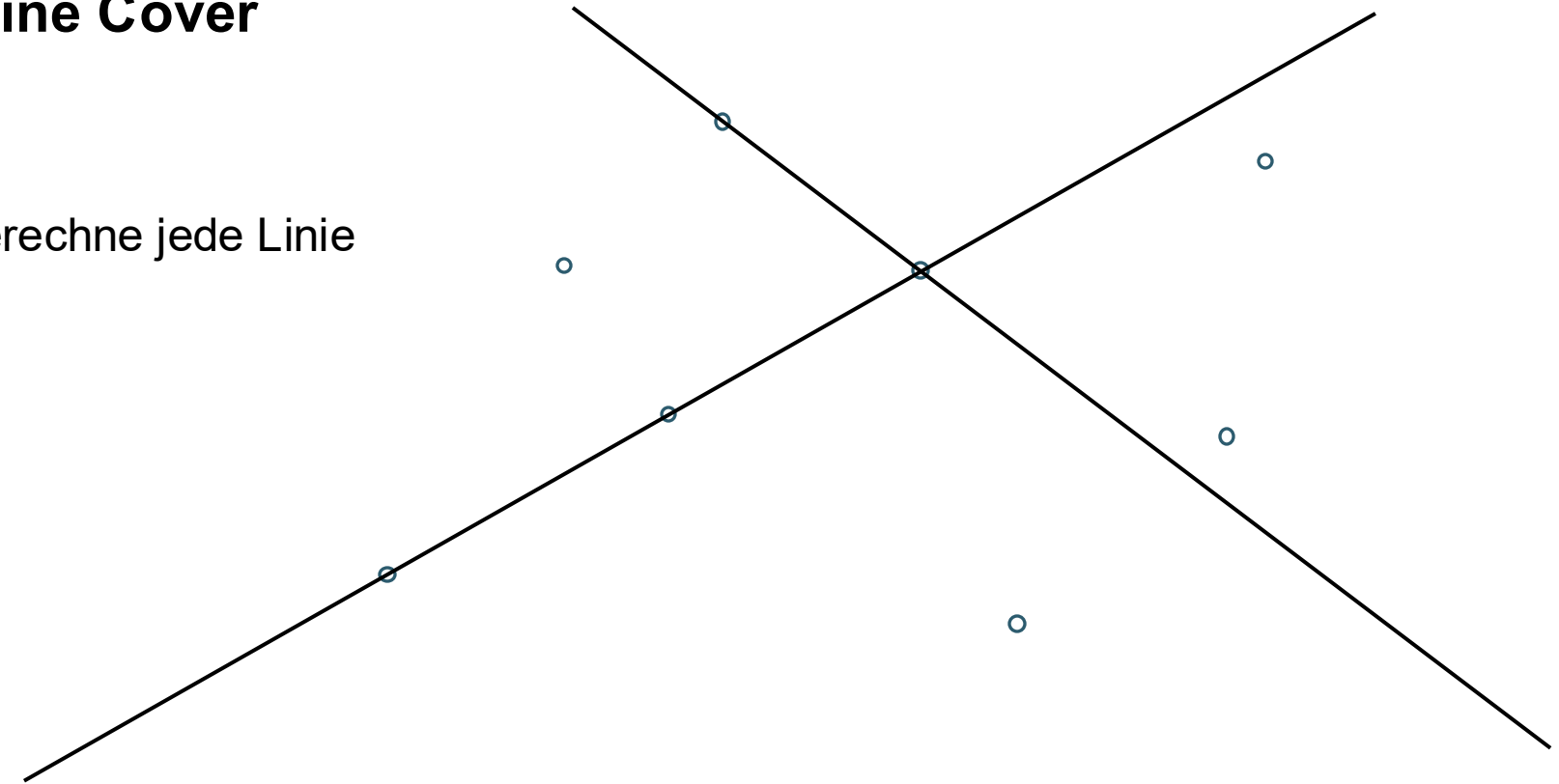


Point Line Cover



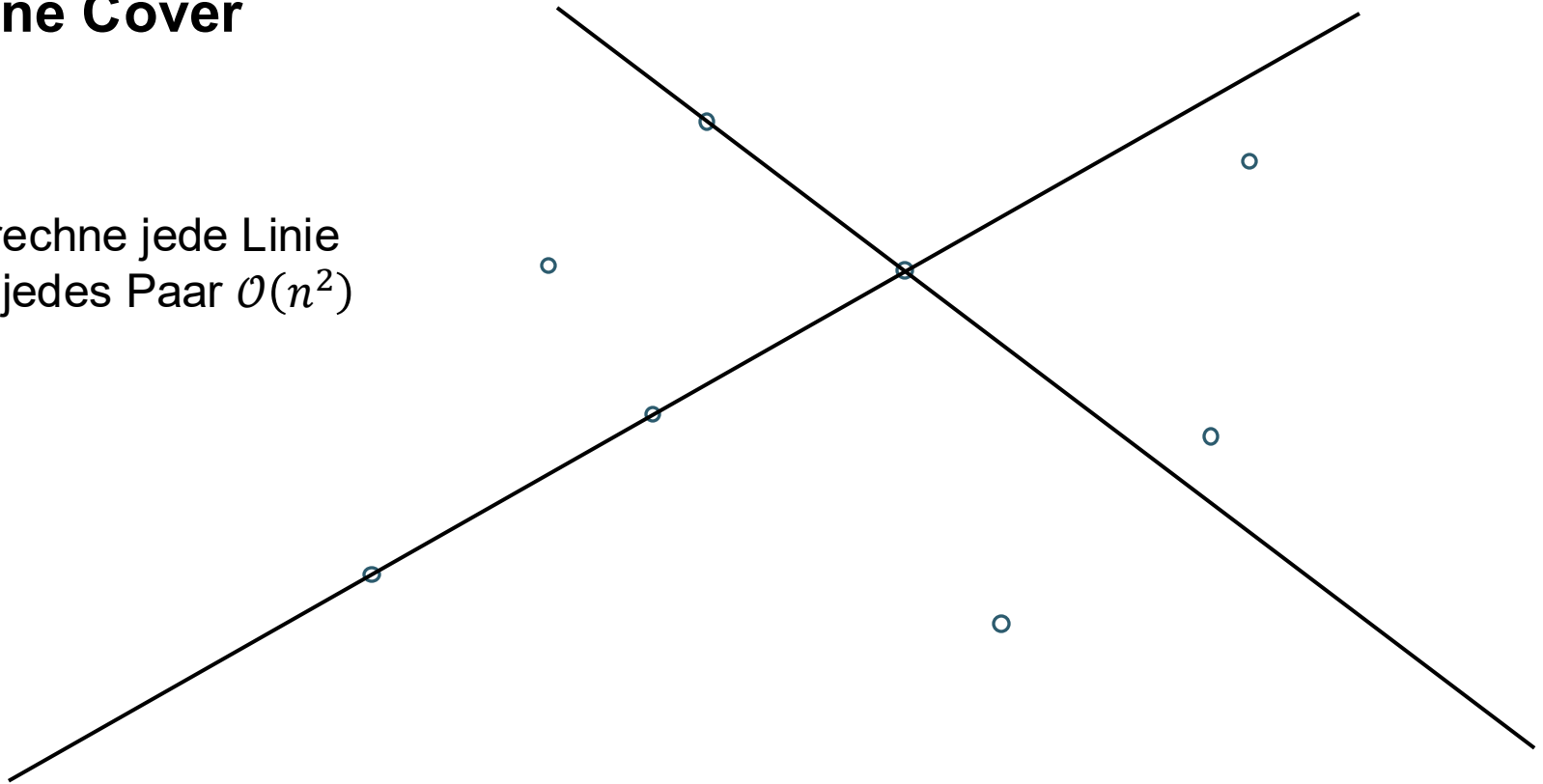
Point Line Cover

1. Berechne jede Linie



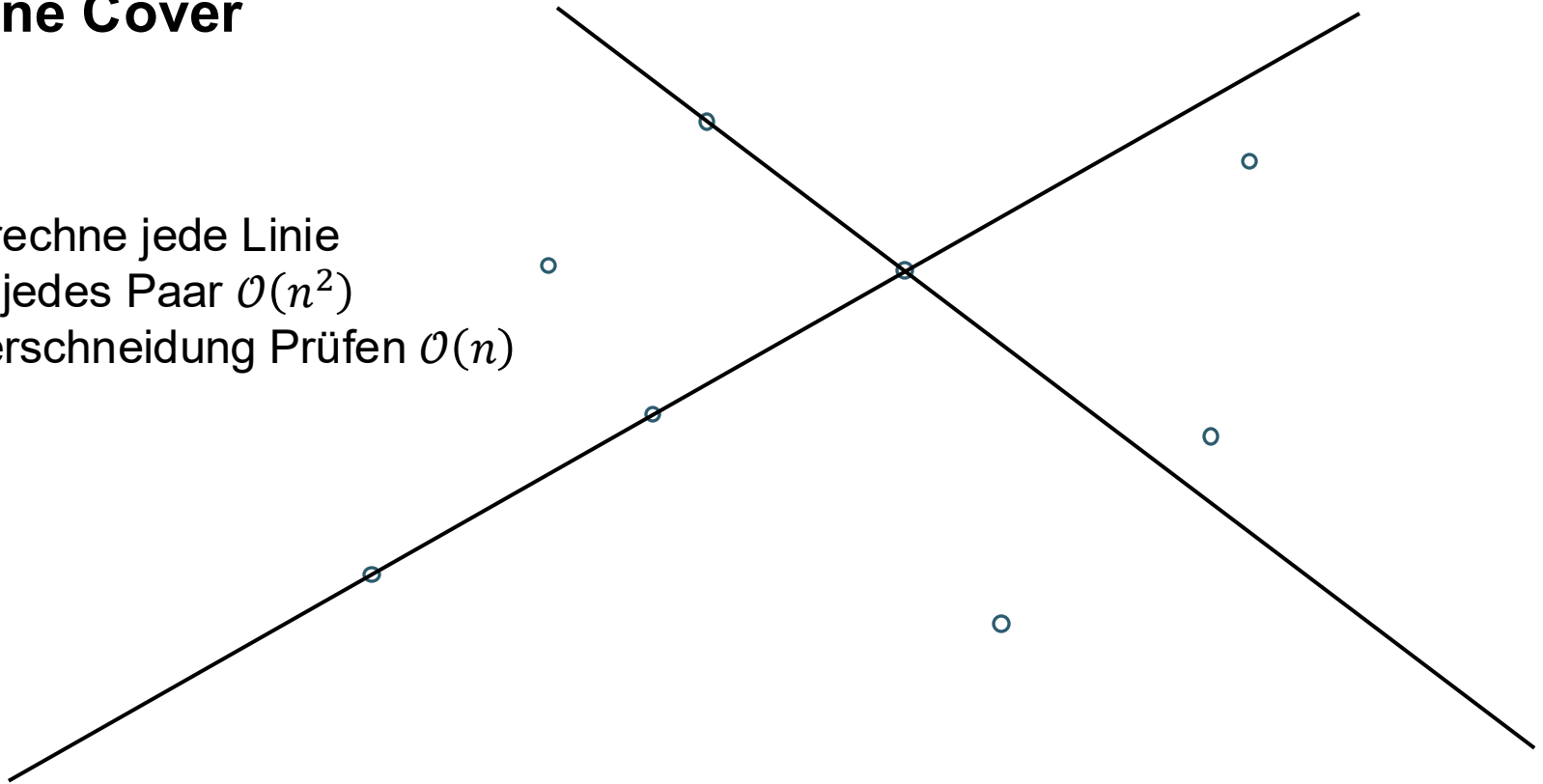
Point Line Cover

1. Berechne jede Linie
- Für jedes Paar $\mathcal{O}(n^2)$



Point Line Cover

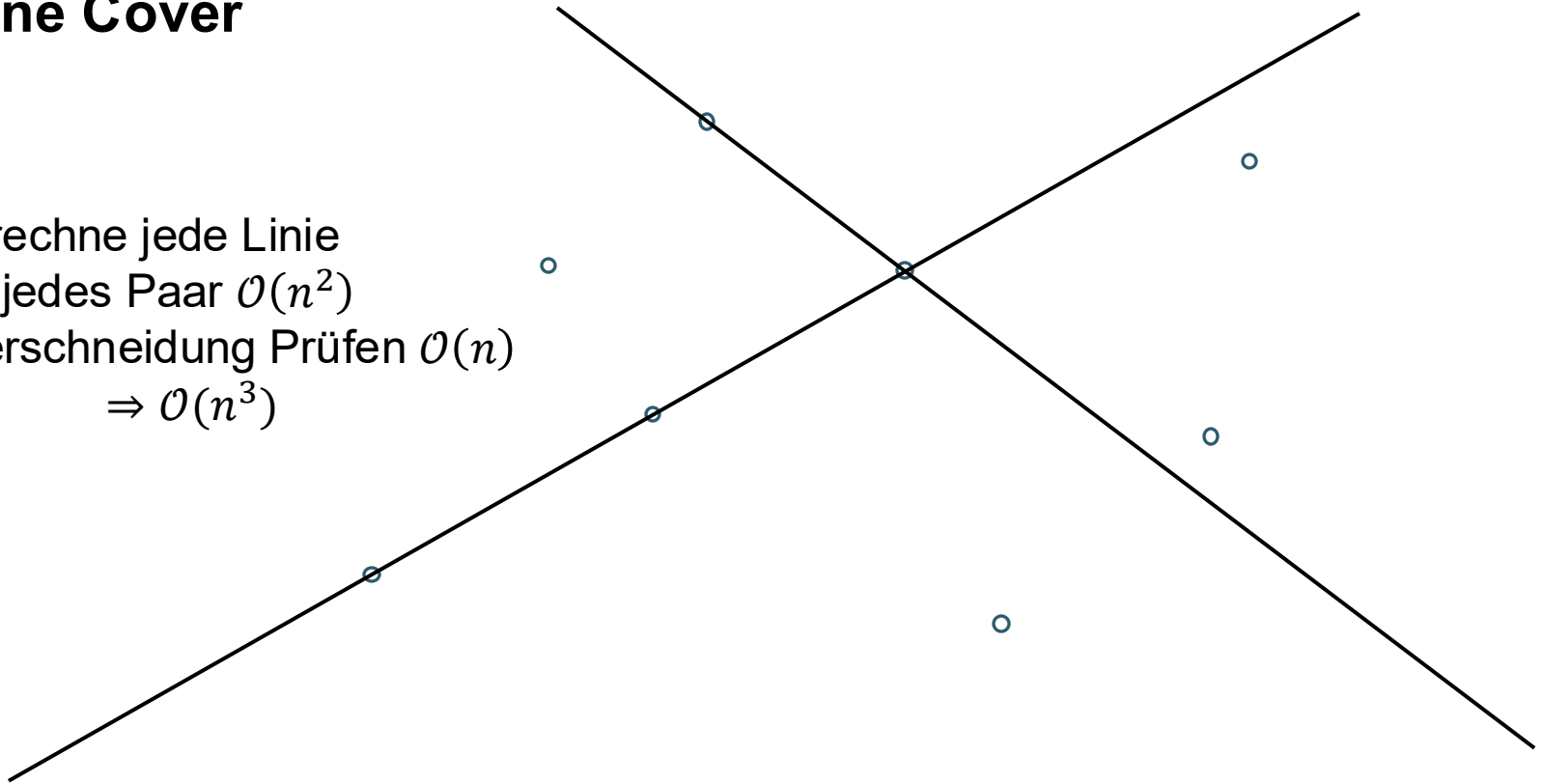
1. Berechne jede Linie
- Für jedes Paar $\mathcal{O}(n^2)$
- Überschneidung Prüfen $\mathcal{O}(n)$



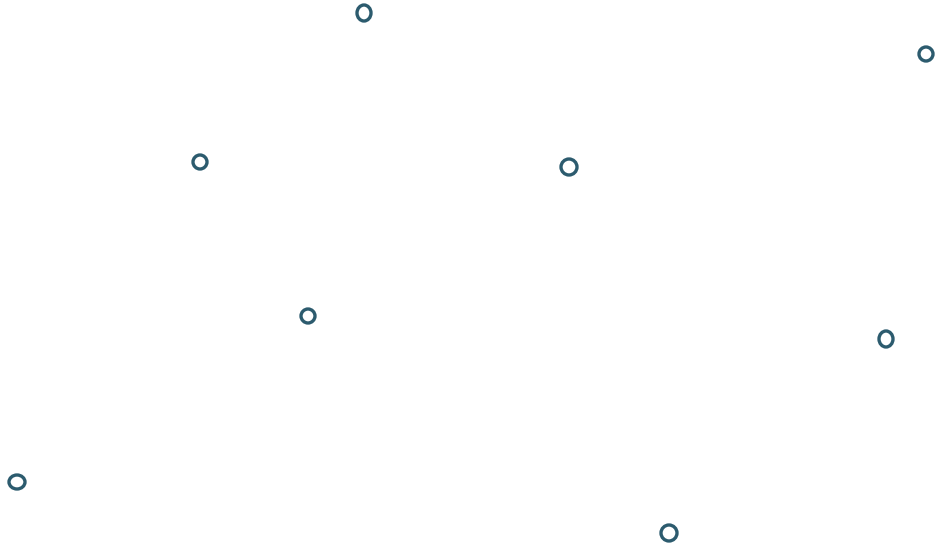
Point Line Cover

1. Berechne jede Linie
- Für jedes Paar $\mathcal{O}(n^2)$
- Überschneidung Prüfen $\mathcal{O}(n)$

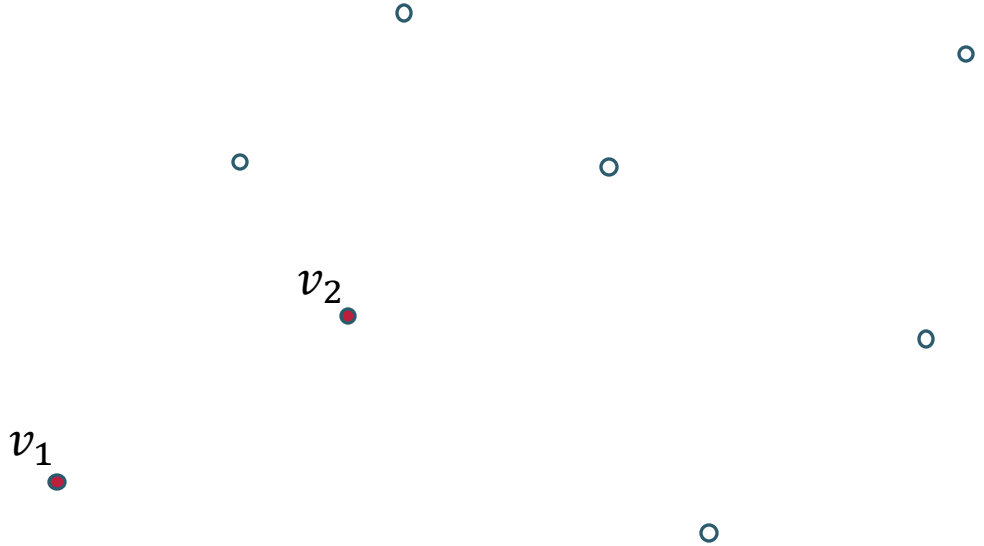
$$\Rightarrow \mathcal{O}(n^3)$$



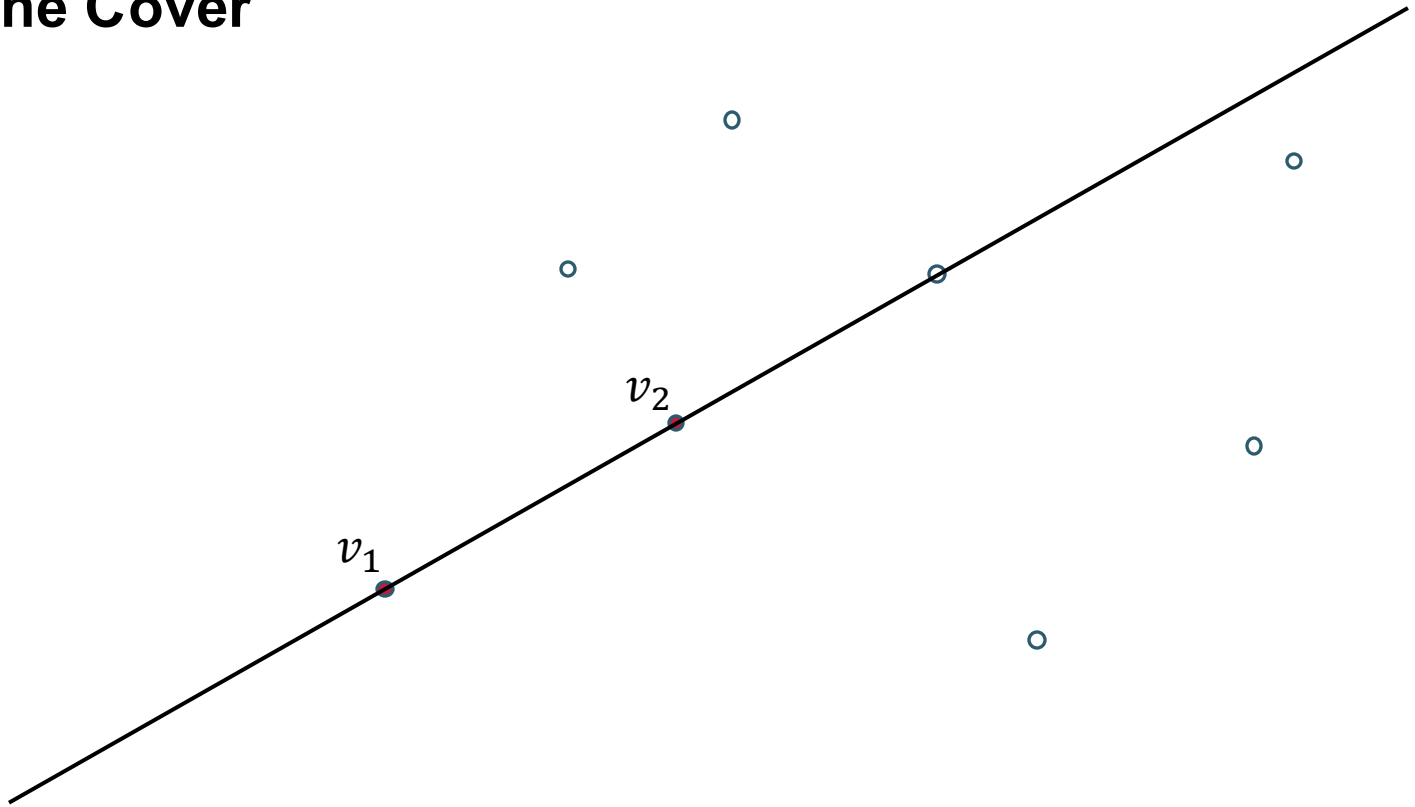
Point Line Cover



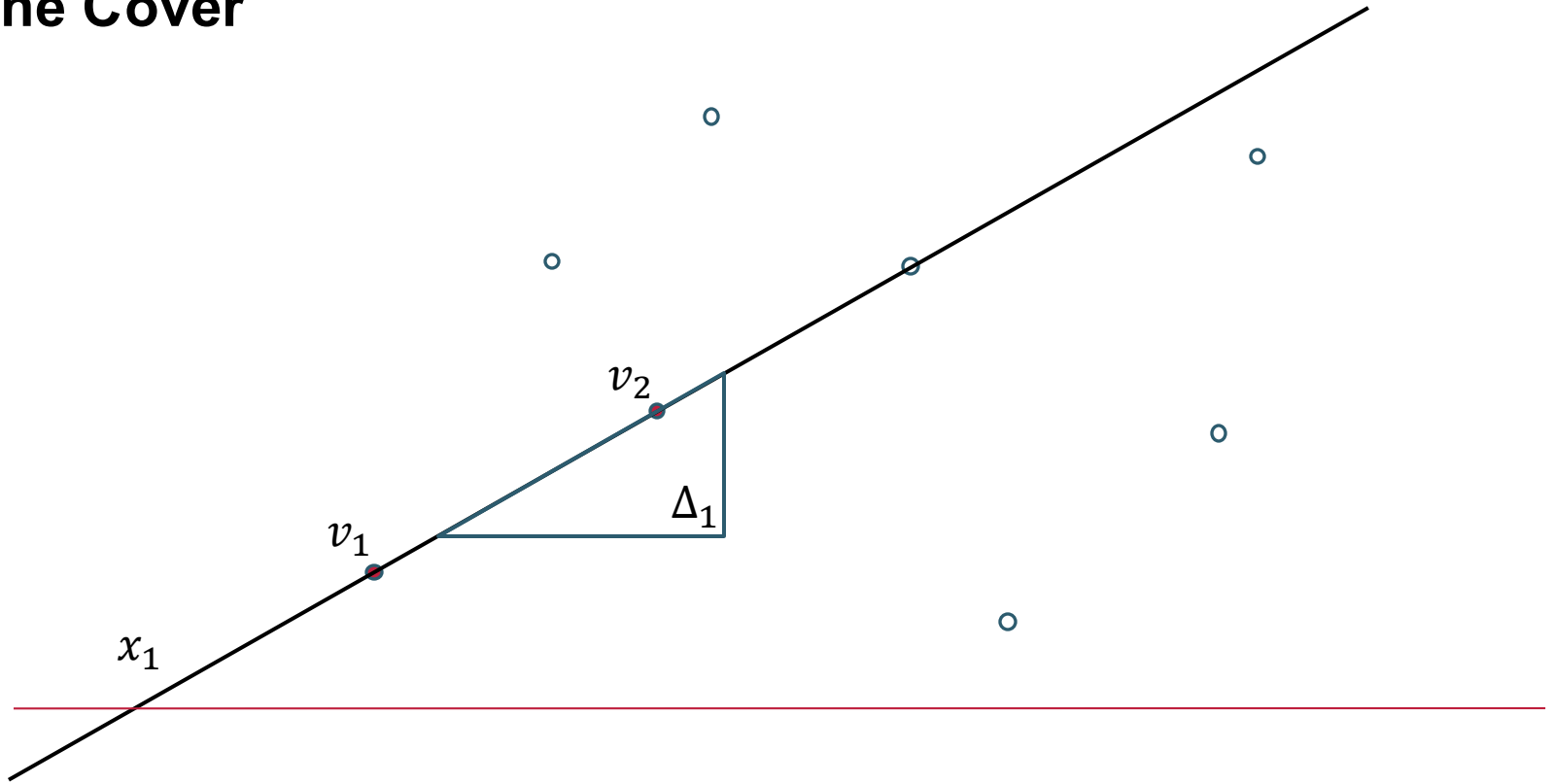
Point Line Cover



Point Line Cover

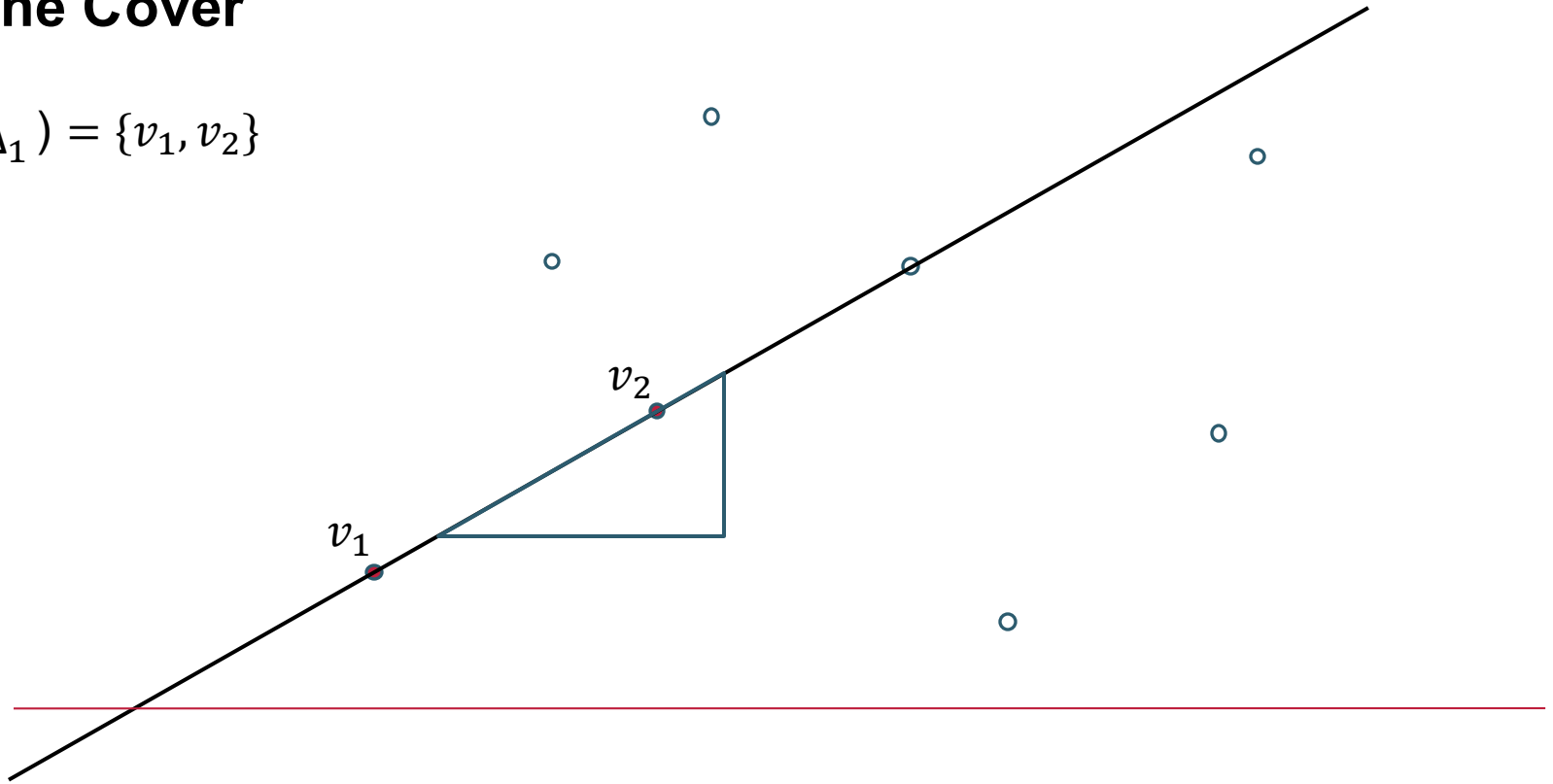


Point Line Cover



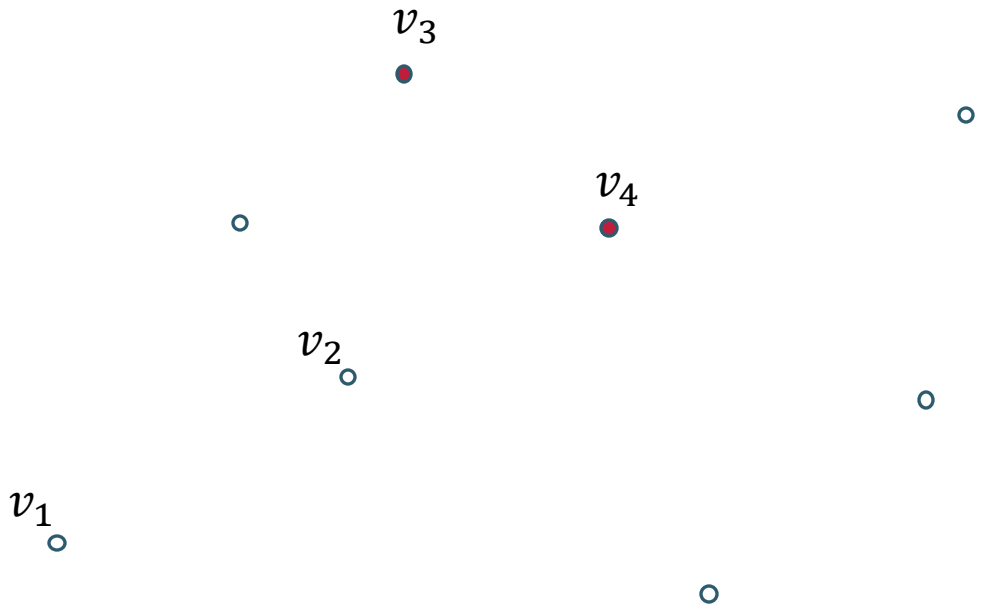
Point Line Cover

$$(x_1, \Delta_1) = \{v_1, v_2\}$$



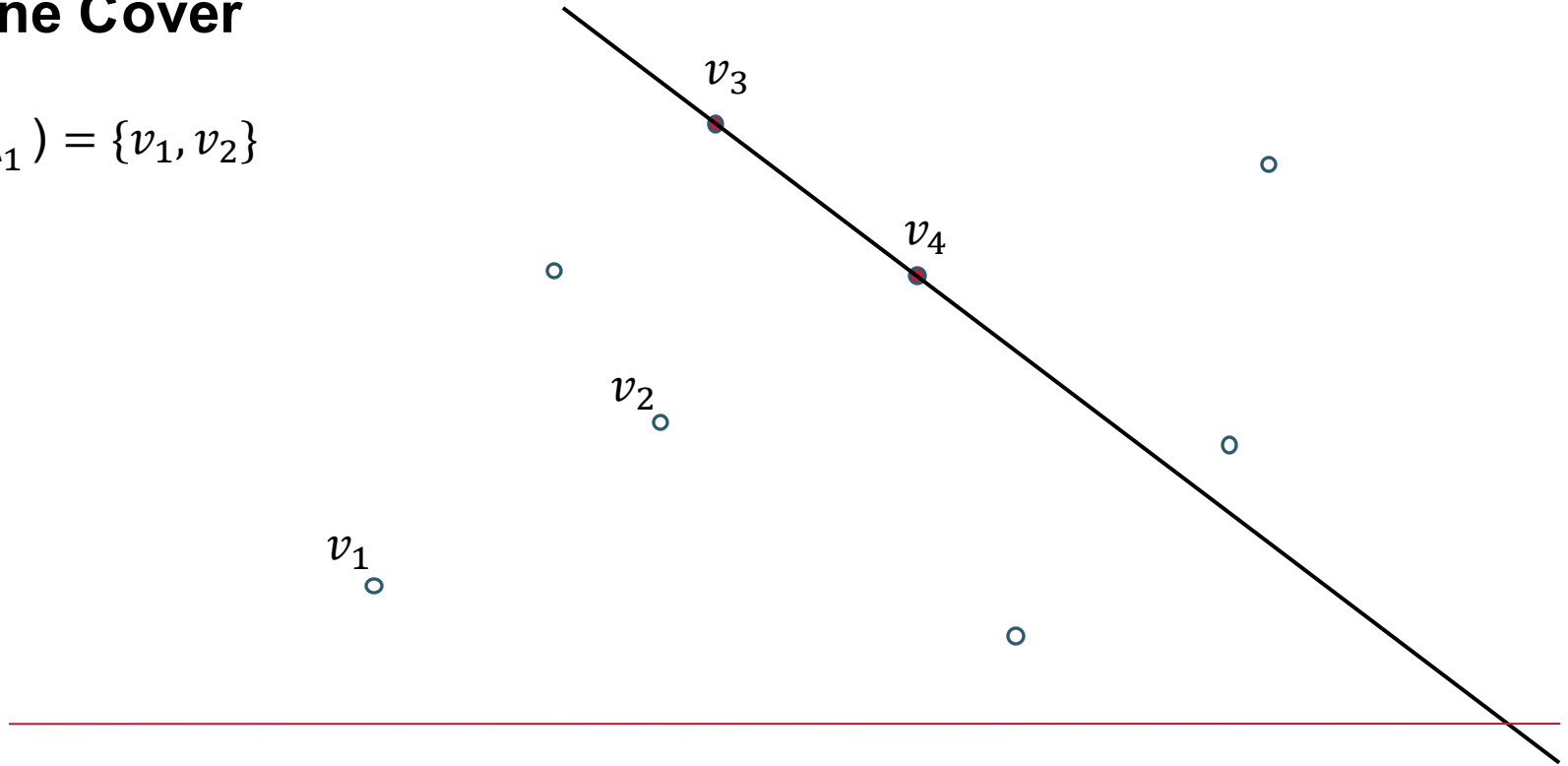
Point Line Cover

$$(x_1, \Delta_1) = \{v_1, v_2\}$$



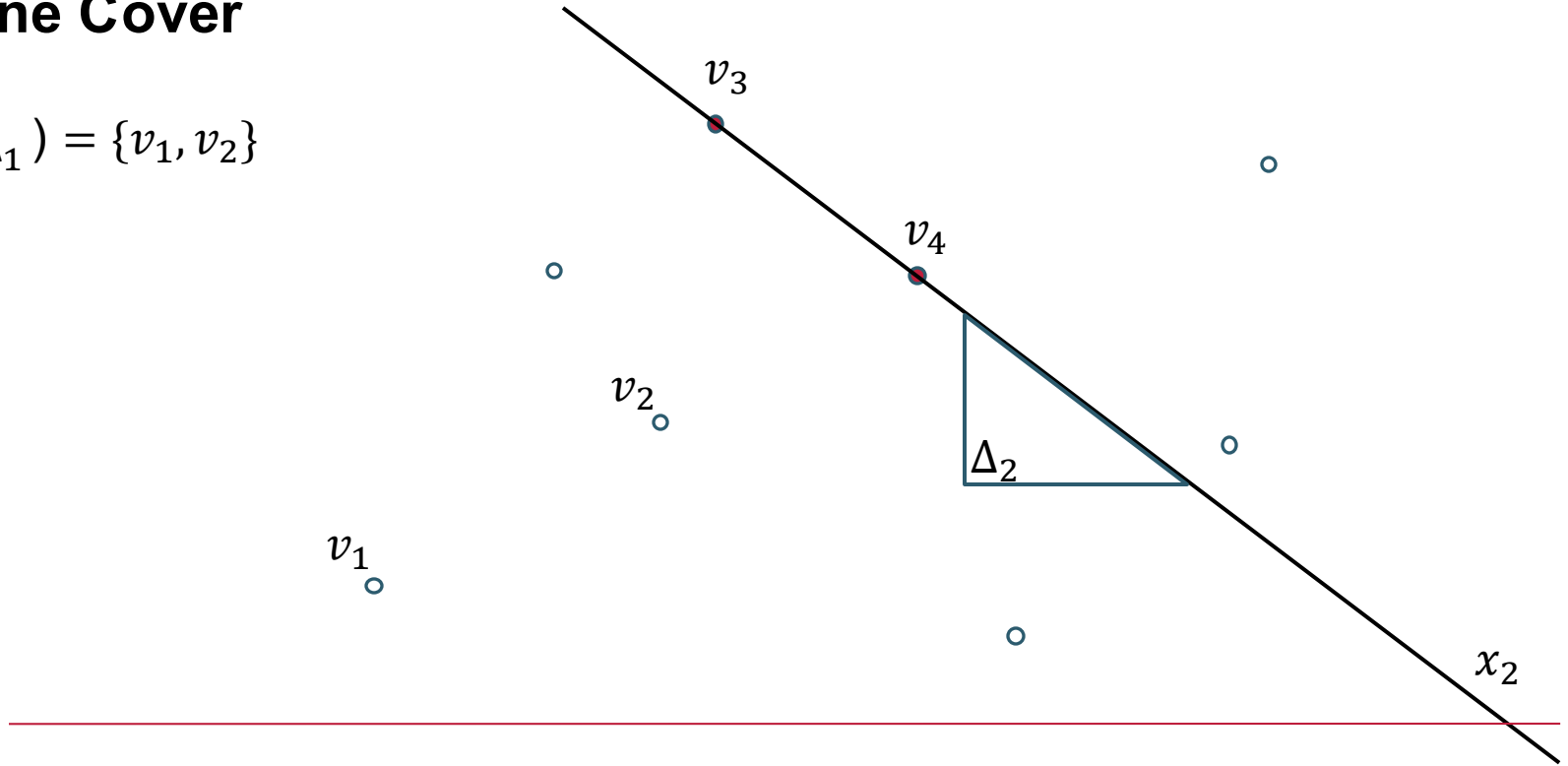
Point Line Cover

$(x_1, \Delta_1) = \{v_1, v_2\}$



Point Line Cover

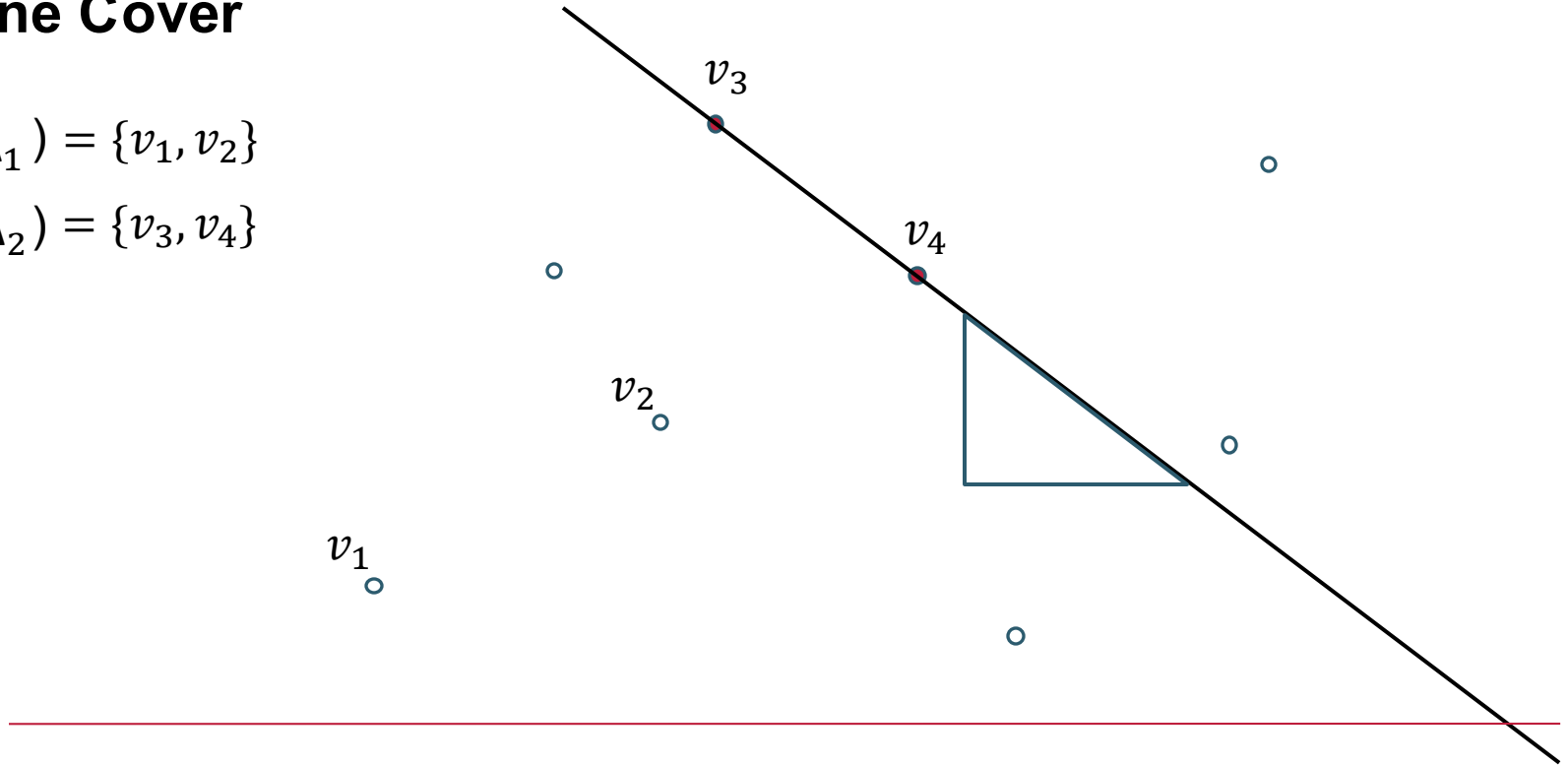
$(x_1, \Delta_1) = \{v_1, v_2\}$



Point Line Cover

$$(x_1, \Delta_1) = \{v_1, v_2\}$$

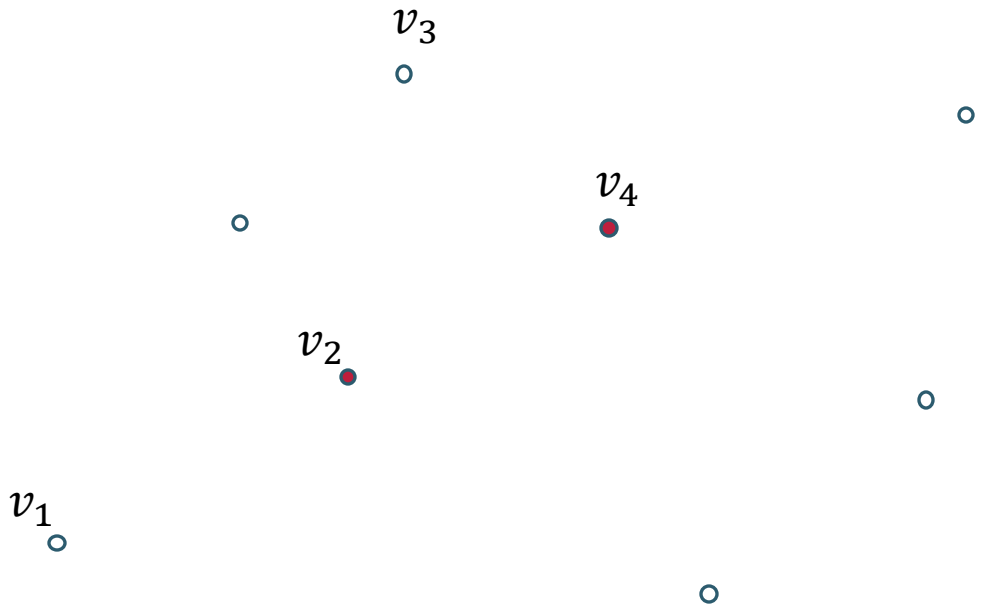
$$(x_2, \Delta_2) = \{v_3, v_4\}$$



Point Line Cover

$$(x_1, \Delta_1) = \{v_1, v_2\}$$

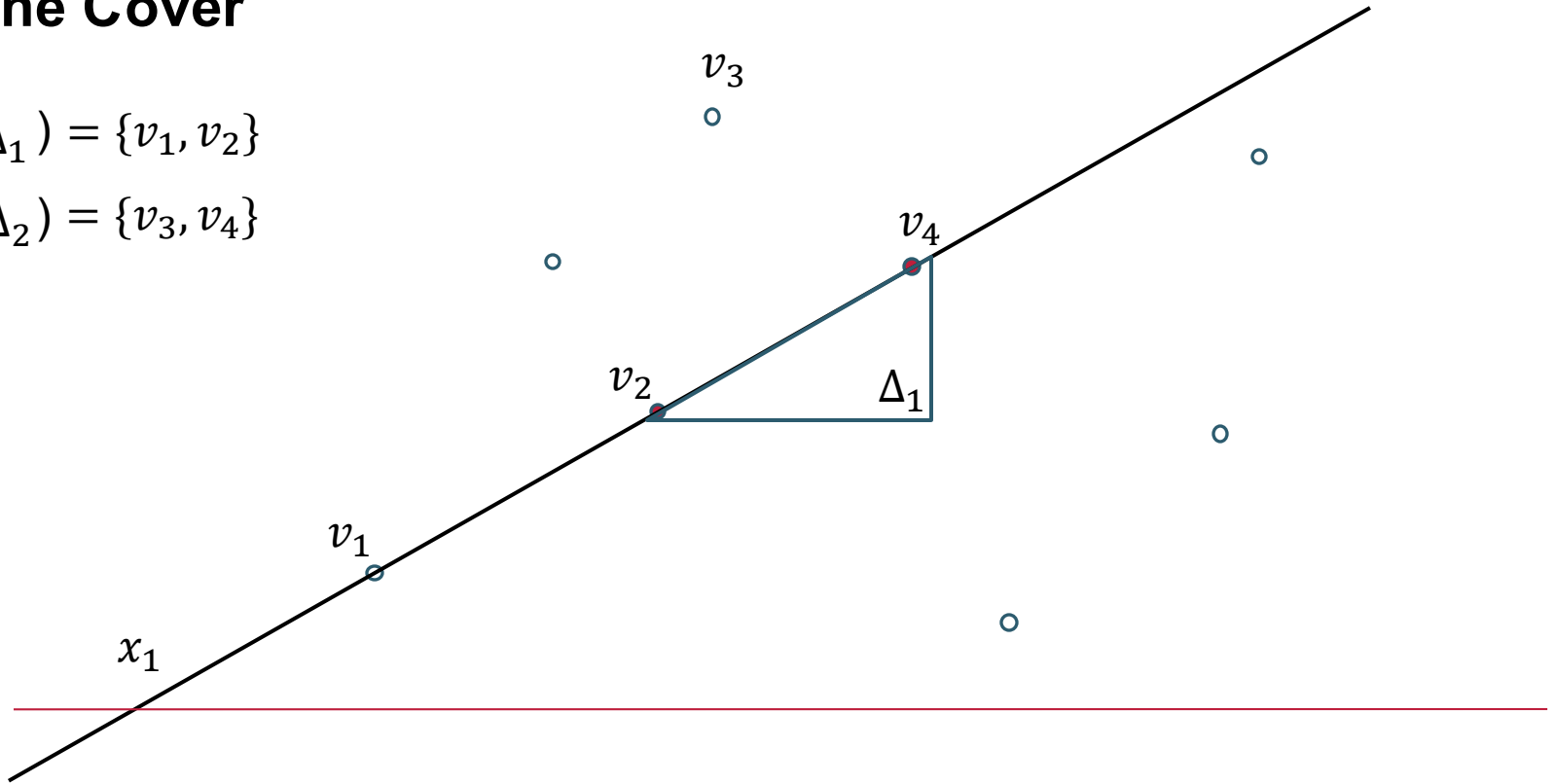
$$(x_2, \Delta_2) = \{v_3, v_4\}$$



Point Line Cover

$$(x_1, \Delta_1) = \{v_1, v_2\}$$

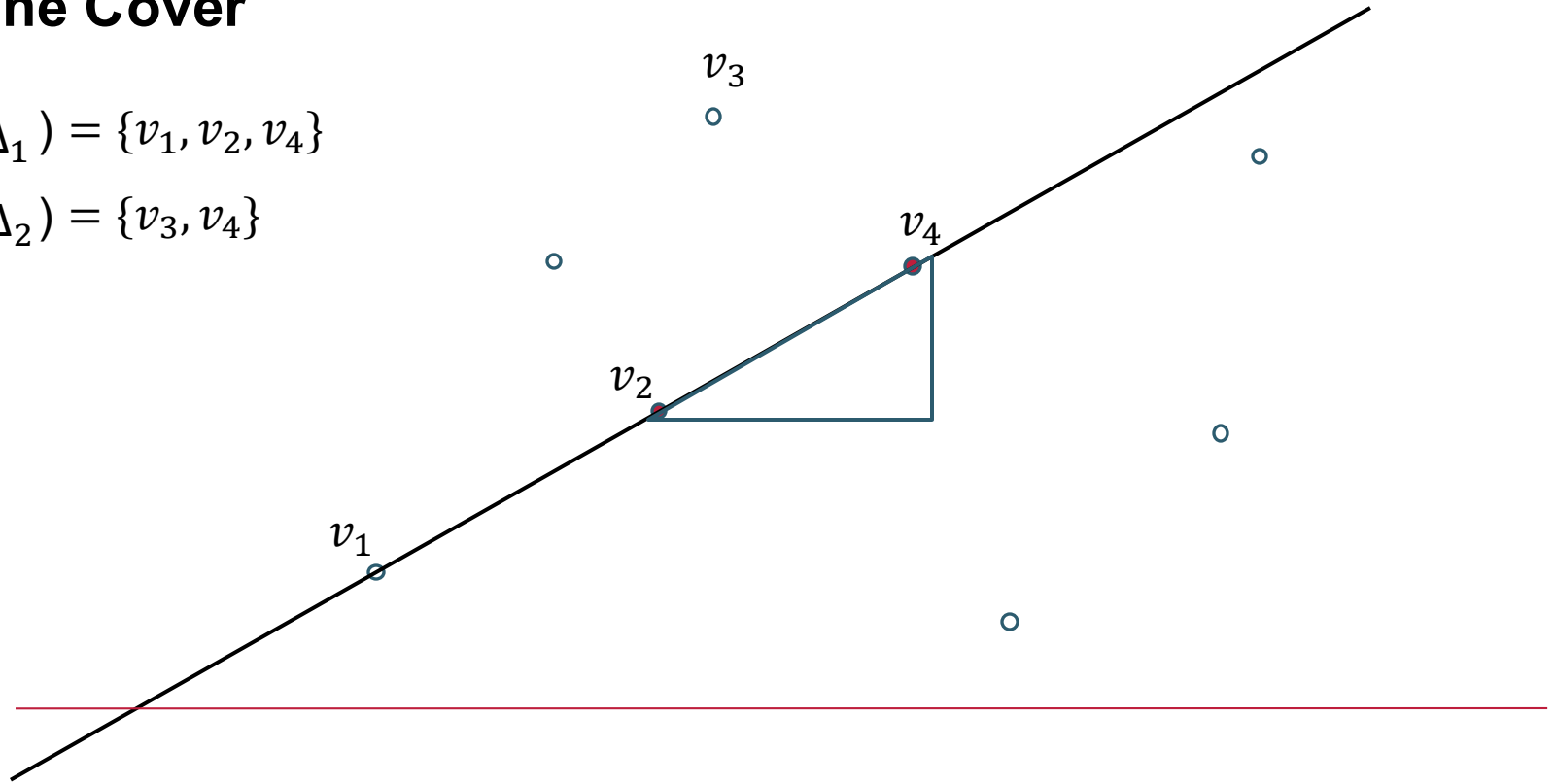
$$(x_2, \Delta_2) = \{v_3, v_4\}$$



Point Line Cover

$$(x_1, \Delta_1) = \{v_1, v_2, v_4\}$$

$$(x_2, \Delta_2) = \{v_3, v_4\}$$

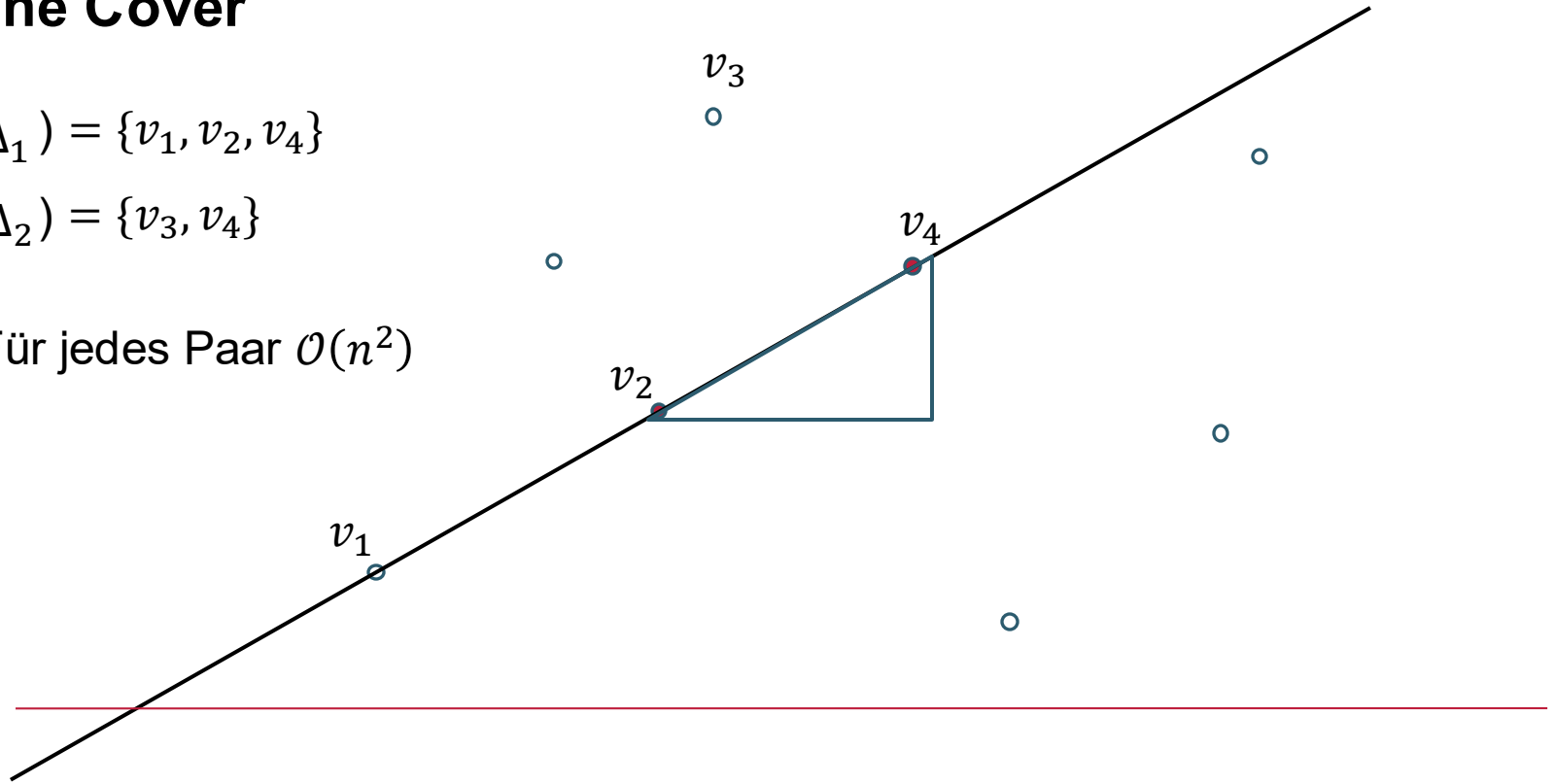


Point Line Cover

$(x_1, \Delta_1) = \{v_1, v_2, v_4\}$

$(x_2, \Delta_2) = \{v_3, v_4\}$

- Für jedes Paar $\mathcal{O}(n^2)$

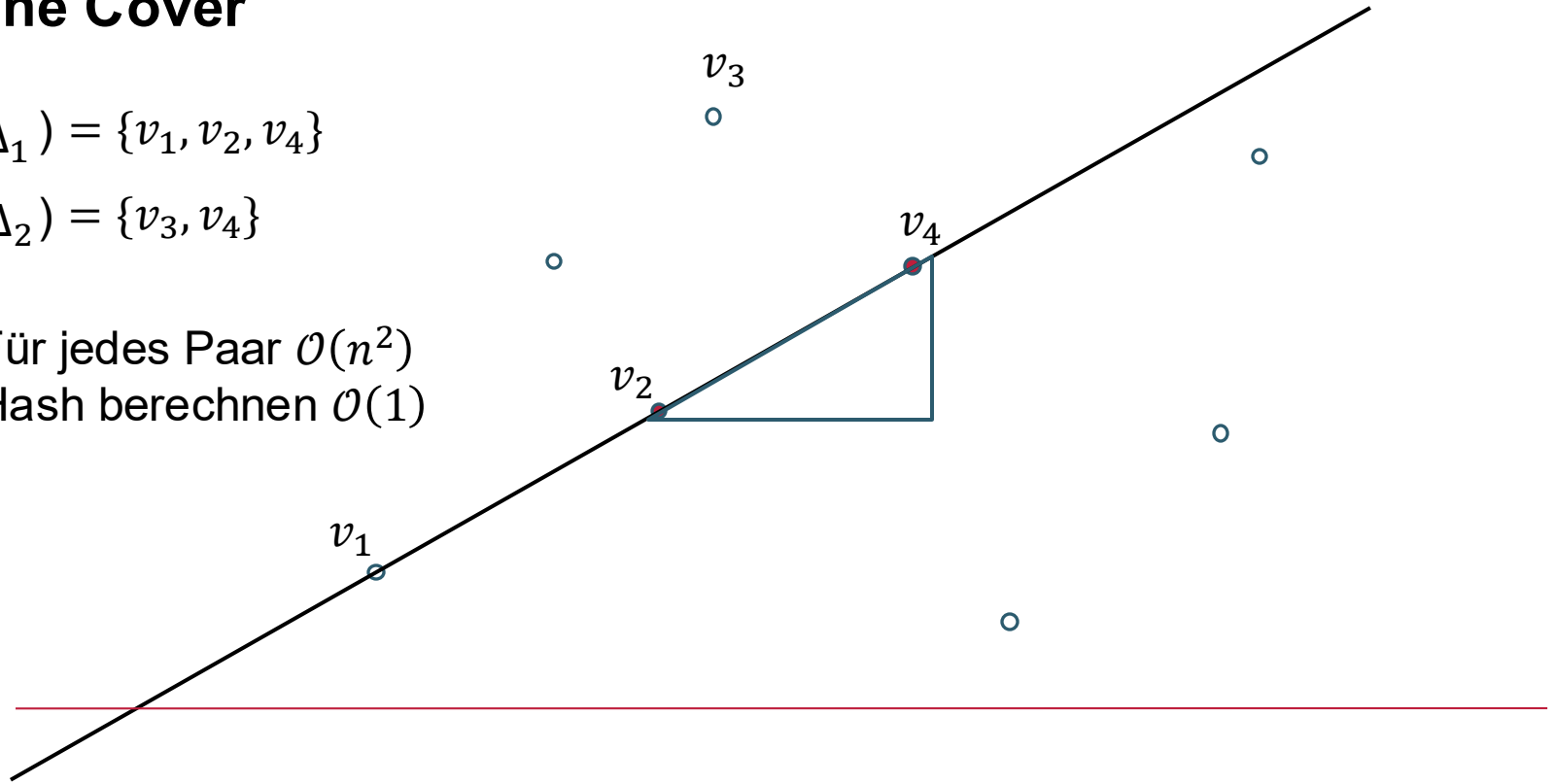


Point Line Cover

$(x_1, \Delta_1) = \{v_1, v_2, v_4\}$

$(x_2, \Delta_2) = \{v_3, v_4\}$

- Für jedes Paar $\mathcal{O}(n^2)$
- Hash berechnen $\mathcal{O}(1)$



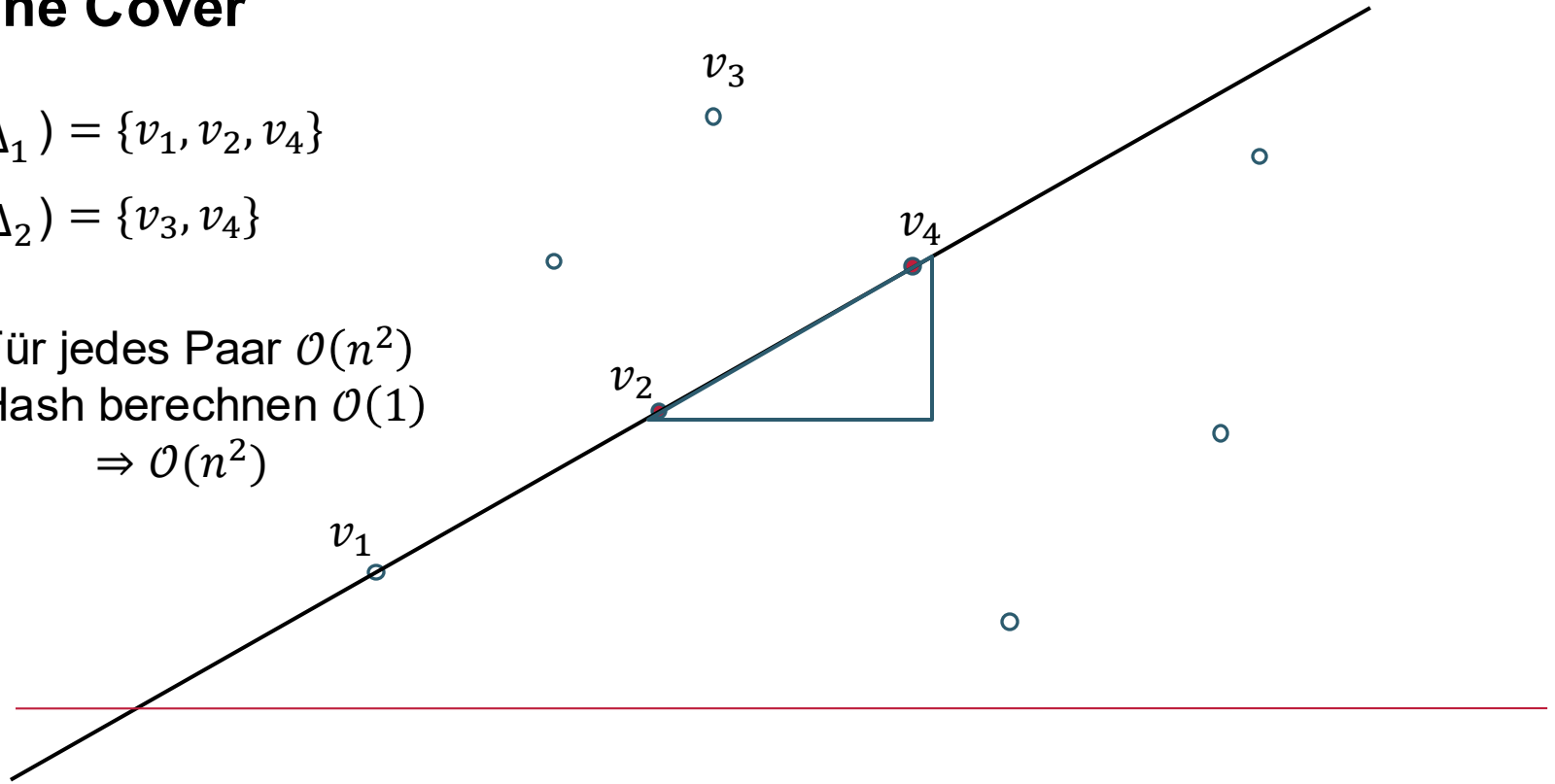
Point Line Cover

$(x_1, \Delta_1) = \{v_1, v_2, v_4\}$

$(x_2, \Delta_2) = \{v_3, v_4\}$

- Für jedes Paar $\mathcal{O}(n^2)$
- Hash berechnen $\mathcal{O}(1)$

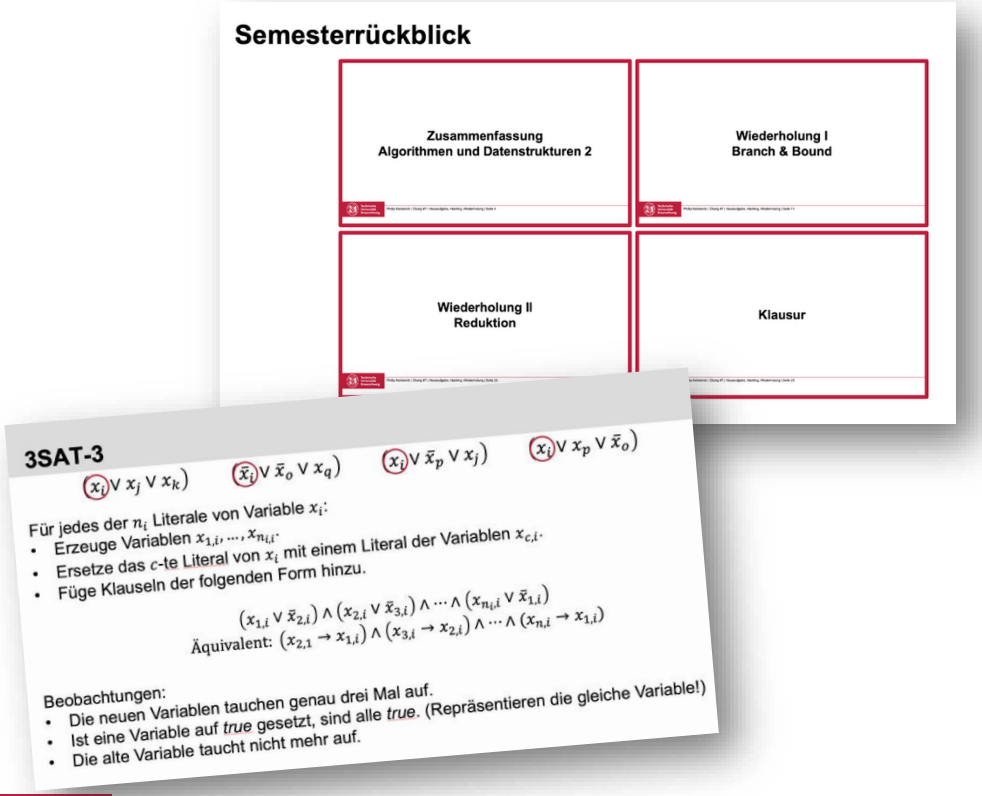
  $\Rightarrow \mathcal{O}(n^2)$



… und nächstes Mal

… und nächstes Mal

… und nächstes Mal



… und nächstes Mal

… und nächstes Mal

Wiederholung und Klausurvorbereitung :)

… und nächstes Mal

Wiederholung und Klausurvorbereitung :)

Am 16.07.2025!