2VO + 4UE
Algorithm Engineering

Vorlesungsablauf

Algorithm Engineering?

Algorithm Engineering

Theorie	Praxis
Vereinfachte Probleme	Komplexe Probleme
Komplizierte Algorithmen	Einfache Algorithmen
Einfaches Maschinenmodell	Speicherhierarchien
O-Notation	In Realität schneller, schnell genug,…
Exakte oder beweisbar gute Lösungen	Heuristiken

"Efforts must be made to ensure that promising algorithms discovered by the theory community are implemented, tested, and refined to the point where they can be usefully applied in practice. [...] to increase the impact of theory on key application areas."

Aho et al. [1997], Emerging Opportunities for Theoretical Computer Science

Damals…

1950er, 60er: **Anfänge der heutigen Algorithmik-Forschung**

Algorithmen wurden größtenteils auch implementiert und getestet

1970er, 80er: **Pen-and-Paper Ära**

Viele algorithmische Durchbrüche

Abstraktere High-Level Beschreibung der Algorithmen

I.d.R. keine Implementierungen/Experimente

Probleme:

- Es schleichen sich leicht Fehler ein
- Theoretisch bester Algorithmen vs. bester Algorithmus in der Praxis

"Beware of bugs in the above code; I have only proved it correct, not tried it."

D. E. Knuth

Fehler? Beispiele…

"If you don't make mistakes, you're not working on hard enough problems."

F. Wikzek

Dreizusammenhang von Graphen

1973	**Hopcroft, Tarjan**	Erster *O(n)* Algorithmus
2001	**Gutwenger, Mutzel**	Implementierung → Fehler gefunden und behoben

Planaritätstest

1961	**Auslander**	Erster polynomieller Algorithmus
1963	**Goldstein**	Fehler gefunden und behoben
1974	**Hopcroft, Tarjan**	Erster *O(n)* Algo; Einbettung extrahieren: Skizze.
1984	**Mehlhorn**	Einbettung extrahieren: Genauer beschrieben
1996	**Mehlhorn, Mutzel**	Implementierung → Fehler gefunden (in beiden obigen) und behoben

Laufzeit: Theorie vs. Praxis

Asymptotische Laufzeit ist wichtig, aber nicht alles!

$$O(f(n)) = \{ f'(n) \mid \exists\, c \geq 0, n_0: \ \forall\, n \geq n_0: \ \ f'(n) \leq c \cdot f(n) \}$$

$$g(n) = O(f(n)) \leftrightarrow g(n) \in O(f(n))$$

$$100 \cdot n \in O(n)$$
$$n \cdot \log(n) \notin O(n)$$
$$n \cdot \log\log(n) \notin O(n)$$

Konstanten vs. LOG-Laufzeiten

- O(log *n*)… zumeist *logarithmus dualis* $\log_2(n)$
- log(1.000.000) < 20
- log(1.000.000.000) < 30
- log log(*n*) < 6 für alle realistischen *n*…
- Die in der *O*-Notation versteckte Konstanten können also in der Realität weit höher sein…

Laufzeit: Theorie vs. Praxis

Asymptotische Laufzeit ist wichtig, aber nicht alles!

z.B.

Gleiche Laufzeit? **Planaritätstest**

- 1974 (Hopcroft, Tarjan) erster Linearzeit-Algo. **sehr** kompliziert
- viele weitere Linearzeit-Algorithmen
- 2003/04 (de Fraysseix, Ossona de Mendez; Boyer, Myrvold): auch Linearzeit aber: – vergleichsweise einfach (DFS-basiert), – **viel** schneller in der Praxis

Polynomiell vs. Exponentiell. **Lösen Linearer Programme**

- Simplex-Algorithmus (Dantzig, 1947) ist im Worst-Case exponentiell (Klee, Minty ‘72).
- Es gibt polynomielle Algorithmen (z.B. Ellipsoid-Methode; Khachiyan ‘79).
- **Simplex ist einfacher und in der Praxis schneller!**

Laufzeit: Theorie vs. Praxis

Asymptotische Laufzeit ist wichtig, aber nicht alles!

z.B.

Versteckte Konstanten. **Graph Minoren**

- Robertson-Seymour-Graphminor-Theorem (ca. 20 Artikel seit 1984): „Jede Minor-abgeschlossene Graphenklasse **K** lässt sich durch eine **fixe endliche** Menge **M** von verbotenen Minor-Teilgraphen charakterisieren.“
- Enthält ein Graph **G** einen fixen Graph **H** als Minor? – Test: $\mathbf{O(n^3)}$ Zeit

$\Rightarrow$ Prüfen, ob ein Graph zu einer Klasse **K** gehört, benötigt nur $\mathbf{O(n^3)}$ Zeit

Oft beliebt bei FPT Algorithmen.

Aber wie groß ist **M**, wie sieht **M** aus? Oft sehr groß, oder gar unbekannt!

Beispiel

K = Graphen die man auf einem Torus ohne Kreuzungen zeichnen kann

$\Rightarrow$ 16 629 Elemente bekannt. Unbekannt wieviele mehr…

Laufzeit: Theorie vs. Praxis

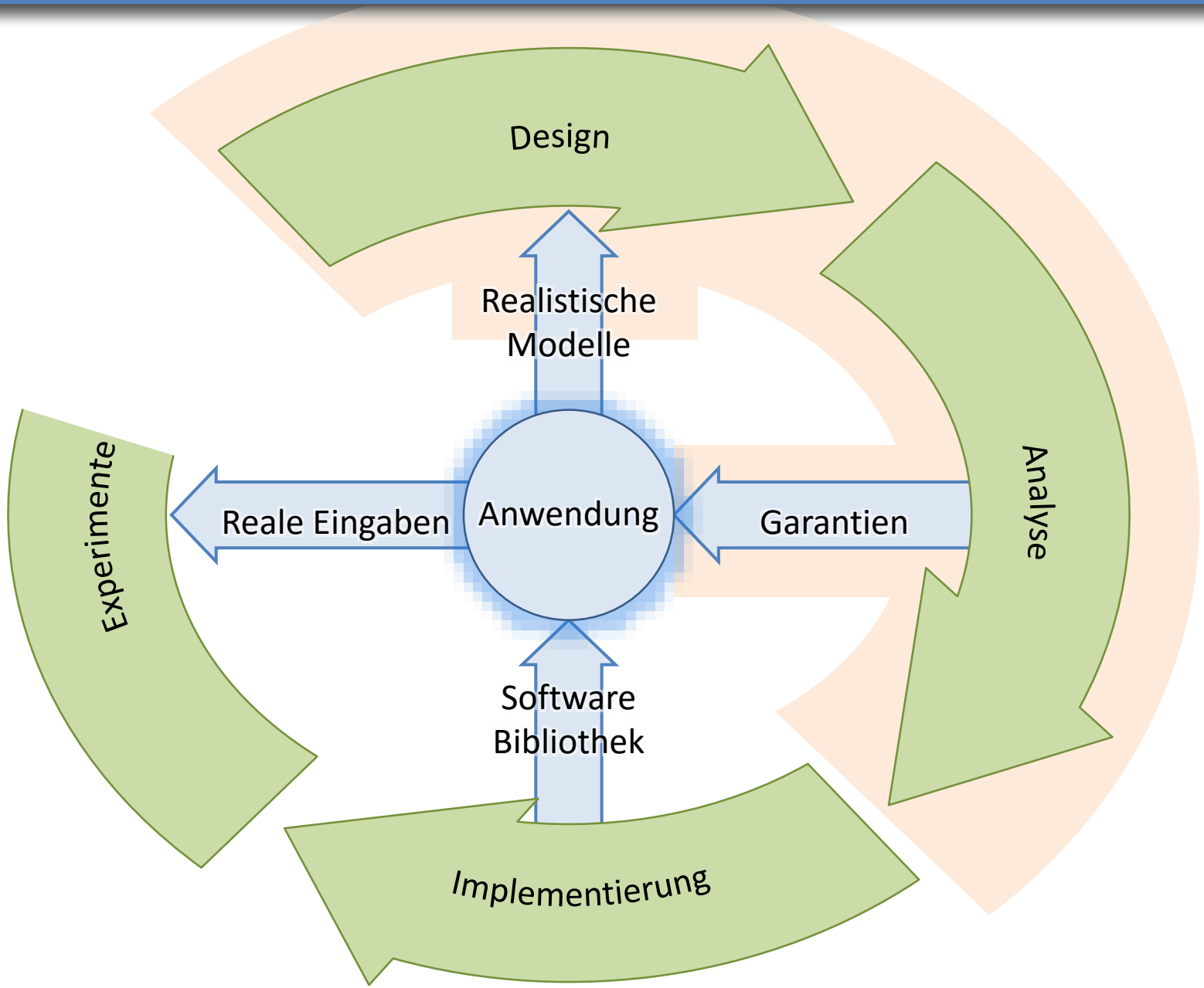
Asymptotische Laufzeit ist wichtig, aber nicht alles!

Maschinen-Modell.

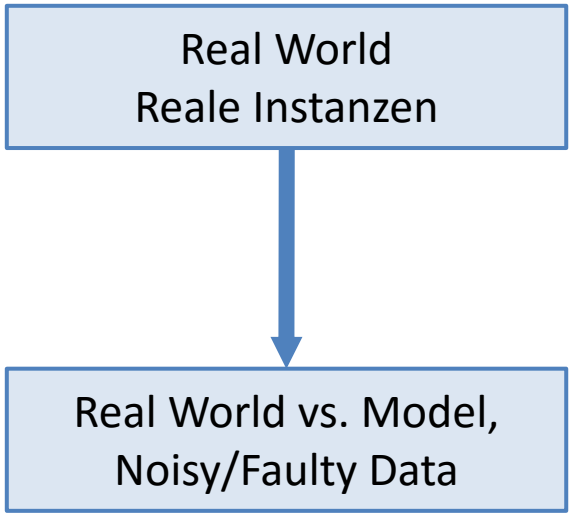
- Traditionelle Algorithmik – **von-Neumann-Modell:**
  - Jede Operation benötigt 1 Zeiteinheit
  - Speicher läßt sich direkt ohne Zeitverlust ansprechen
  - Keine Probleme mit MAX_INT, Präzision von Gleitkommazahlen,…

- Realität:
  - Verschiedene Chiparchitekturen die verschiedene Operationen direkt unterstützen. Einige Operatoren werden simuliert. Andere Operatoren können parallel ausgeführt werden (z.B. auf GPU).
  - Speicher ist hierarchisch organisiert: schnelle kleine Caches, flotter RAM, langsame Festplatte, etc.

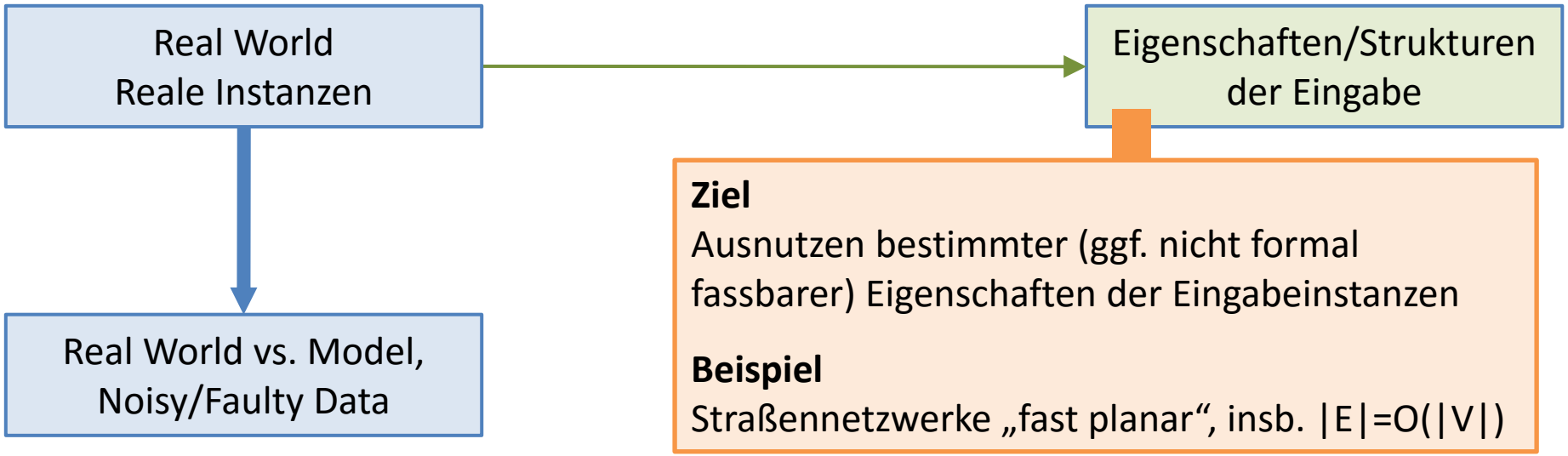
Algorithm Engineering Zyklus



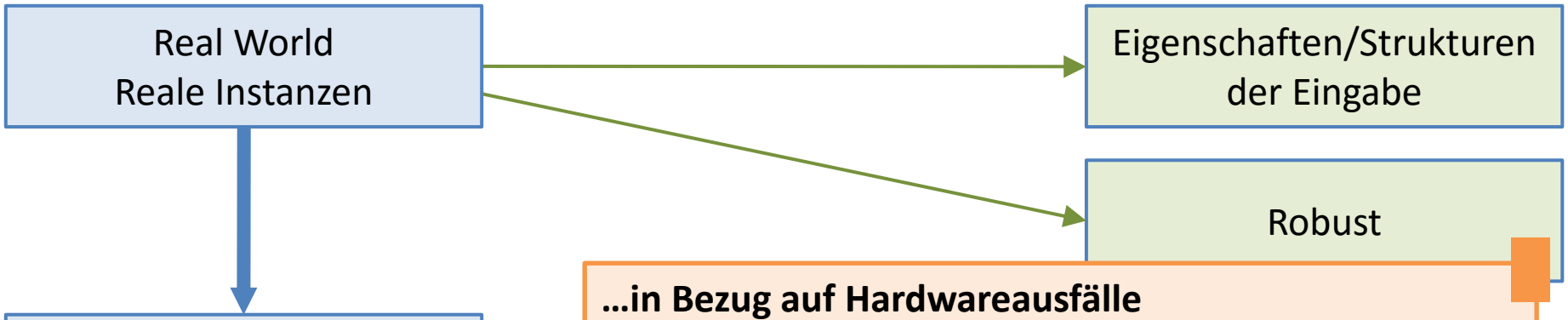
Große Themen



Große Themen



Große Themen



…in Bezug auf Hardwareausfälle
Volle Redundanz ist meist zu teuer!

RAM Fehler (random bit flips):
statistisch relevant für Server-Farmen!

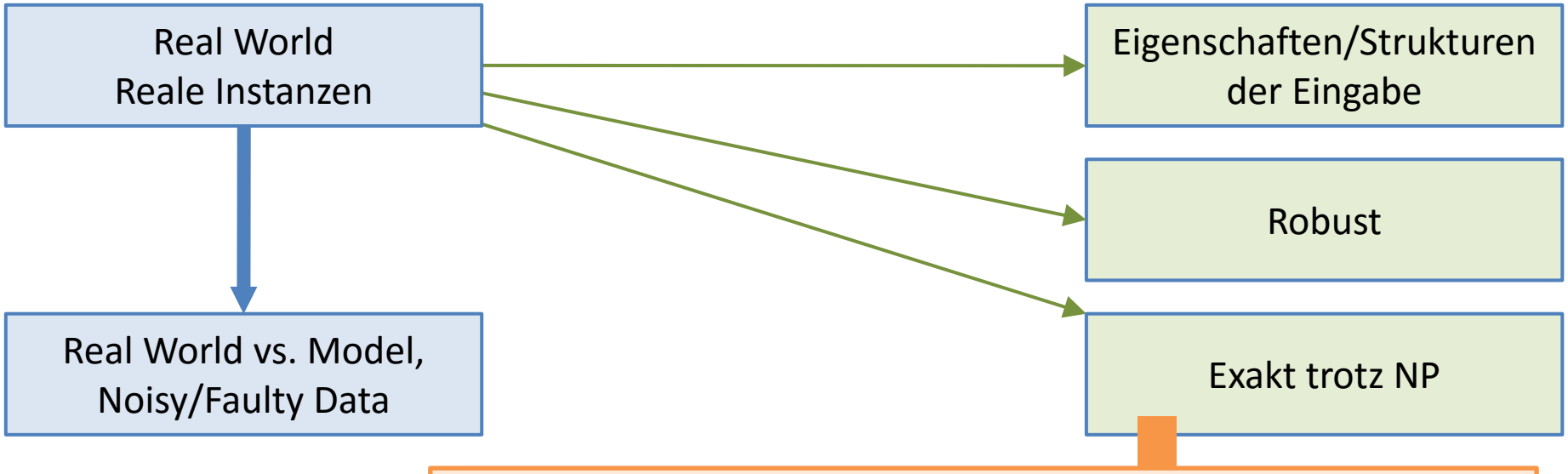
„Resilient“ Datenstrukturen/Algorithmen:
funktionieren noch korrekt bei bis zu *k* falschen bits.

…in Bezug auf Eingabewerte
Eingabe ist oft fehlerbehaftet, nur probabilistisch, etc.

Lösung sollte auch bei leicht veränderten Eingaben (Zielfunktion) noch „gut“ sein.
zB: Kantenkosten mit Wahrscheinlichkeitsverteilung;
kürzester Weg sollte keine „Risiko-Kanten“ enthalten

sensitivity analysis, robust optimization, stochastic optimization

Große Themen



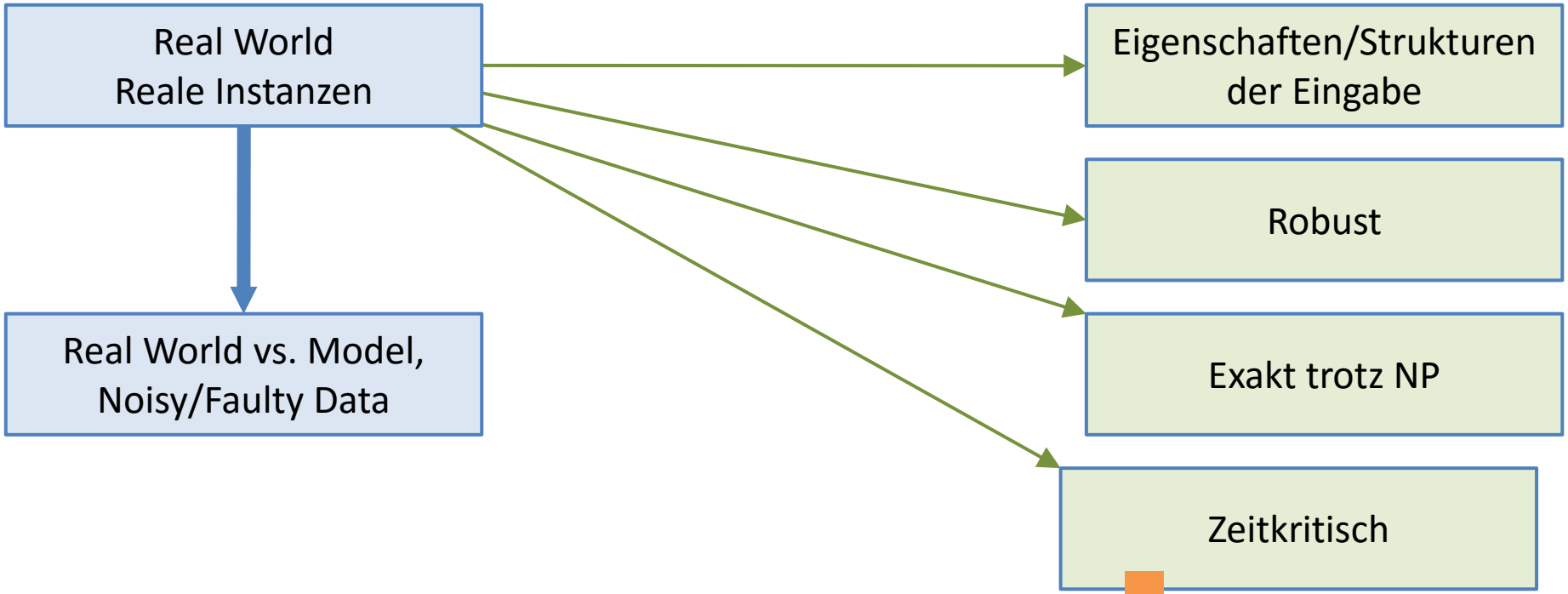
NP-Beweise benutzen riesige Gadgets und Hilfskonstruktionen, die in der Realität nicht vorkommen.

- Echte Instanzen sind oft nicht so kompliziert
- Mit den richtigen Methoden können kleine exponentielle Funktionen gutmütig sein

zB: Traveling Salesman, Steinerbaum, *k*-Cardinality Tree, Linear Ordering & Varianten, Vertex Cover,…

Fixed Parameter Tractable
Ganzzahlige Lineare Programme (ILPs)

Große Themen



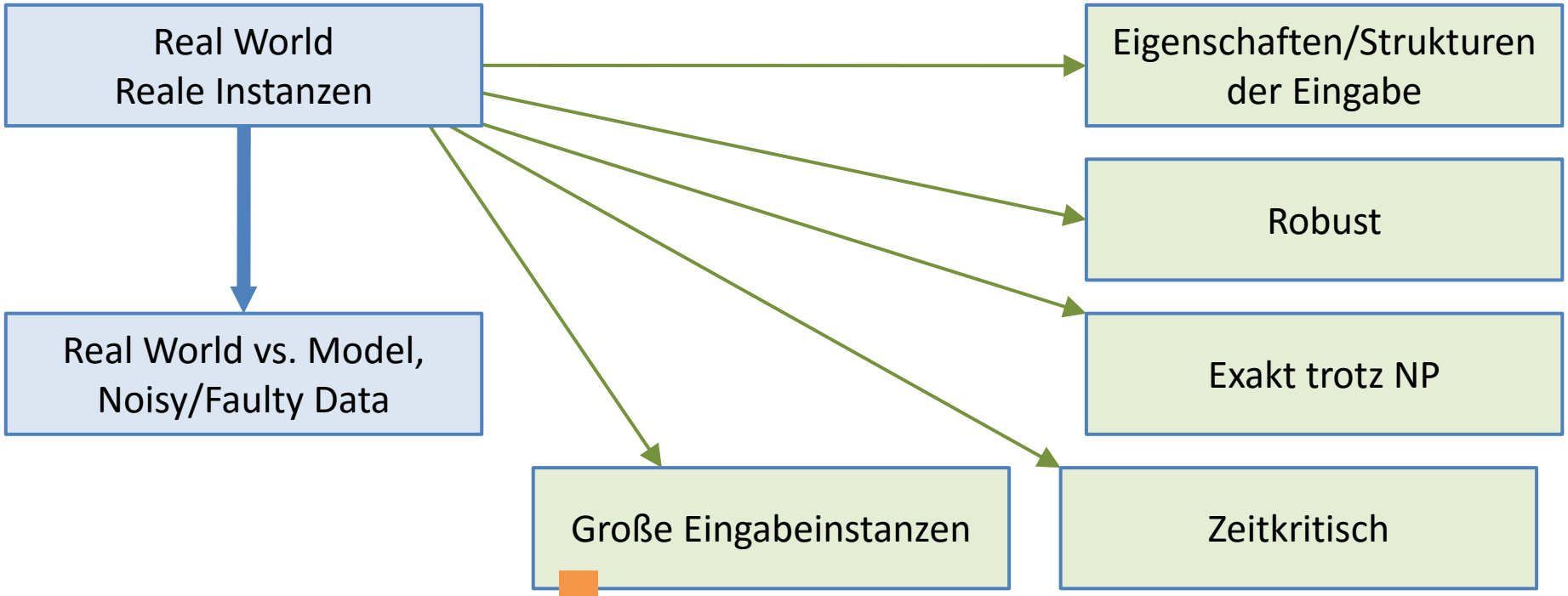
Real-Time Systems

Ein Problem muss innerhalb eines strikten Zeitintervalls gelöst sein.

- Lieber eine schlechte Lösung als gar keine
- „echte“ Garantien für Güte und Laufzeit (nicht nur *O*-Notation)

zB: Fahrelektronik im Auto (Bremsassistenten, ESP,…)

Große Themen

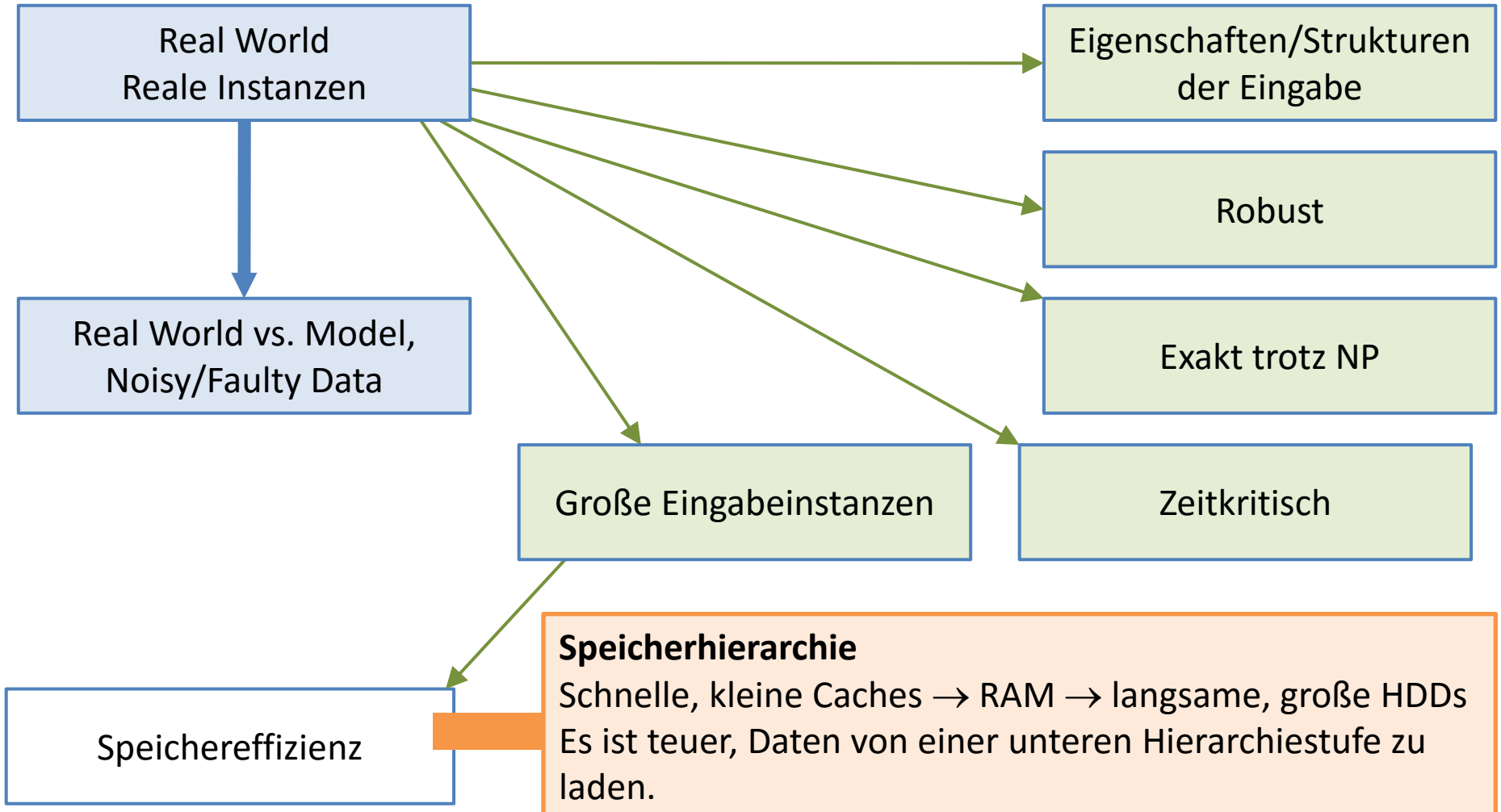


"One of the few resources increasing faster than the speed of computer hardware is the amount of data to be processed."

[IEEE InfoVis 2003 Call-For-Papers]

Geographische Systeme, Bioinformatik/Systembiologie,…

Große Themen



Große Themen

Real World
Reale Instanzen

Eigenschaften/Strukturen der Eingabe

Robust

Exakt trotz NP

Zeitkritisch

Real World vs. Model, Noisy/Faulty Data

Große Eingabeinstanzen

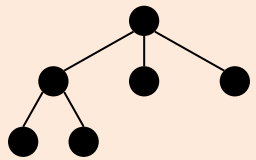
Speichereffizienz

„Succinct“ Datastrukturen

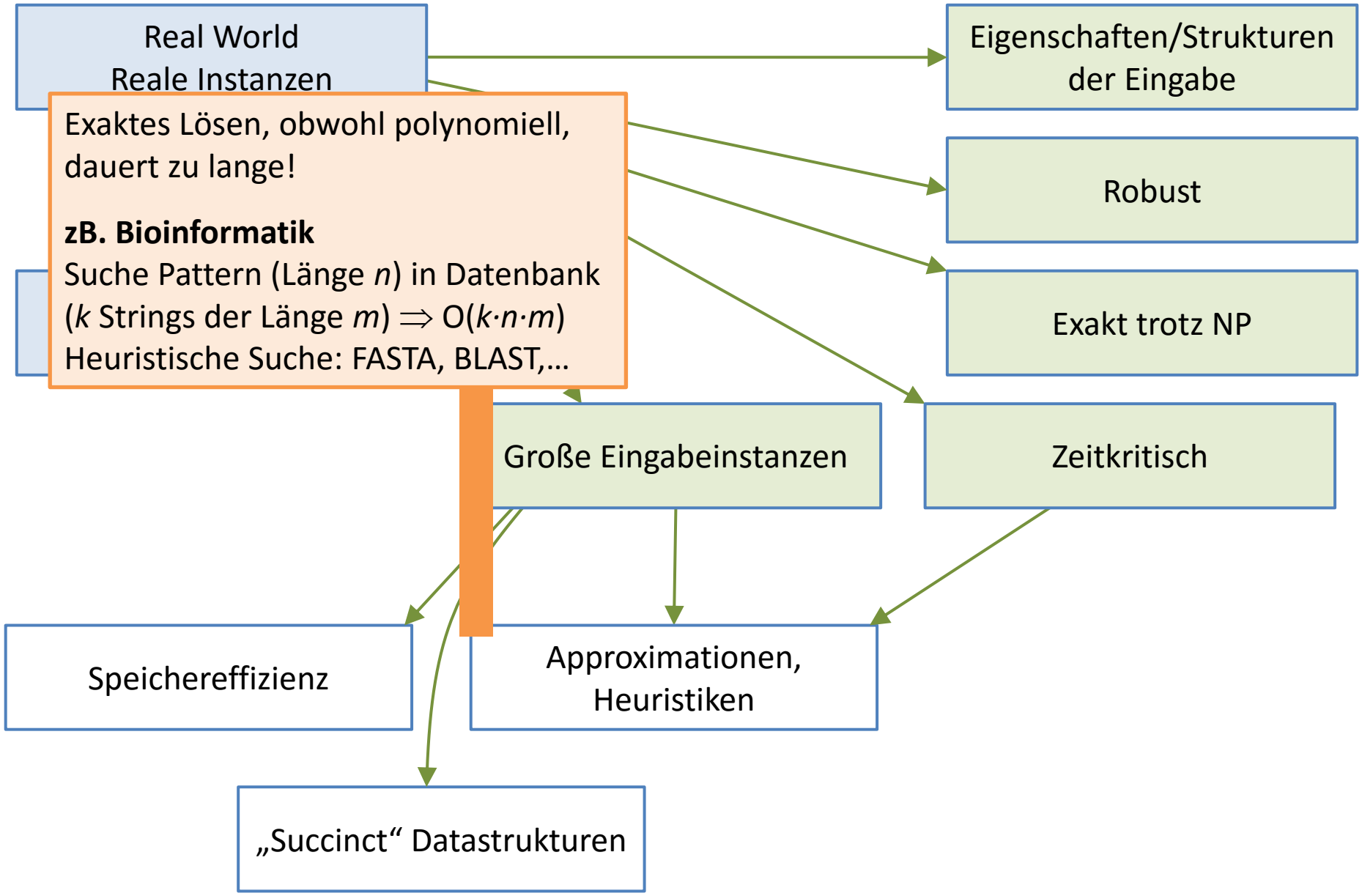
Speichere Datenstrukturen mit möglichst wenigen Bits, ohne die Laufzeit zu verschlechtern (≠ Komprimierung)

zB. Baum: 2 bit pro Knoten

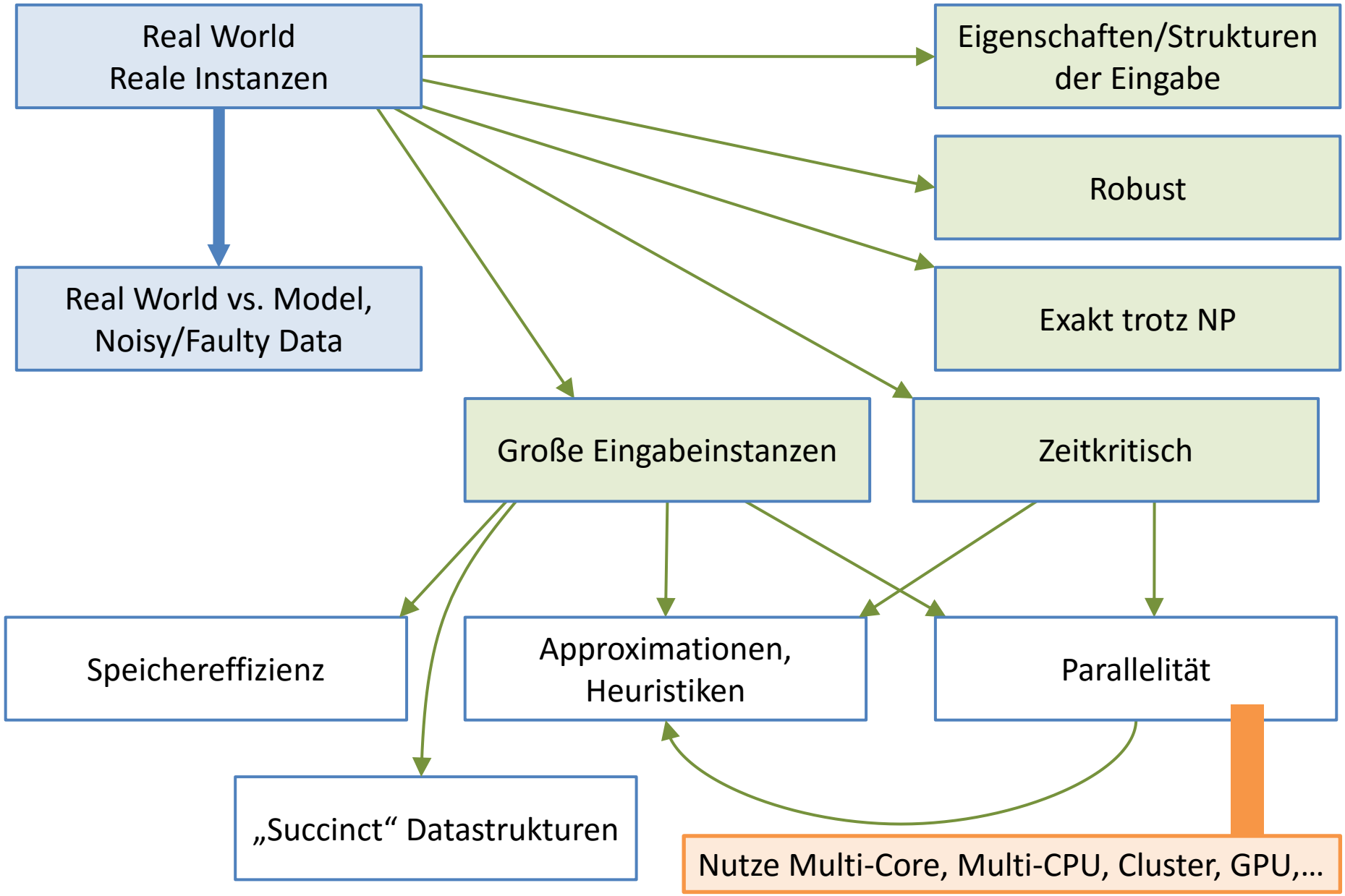
(((()())()()) ⇒



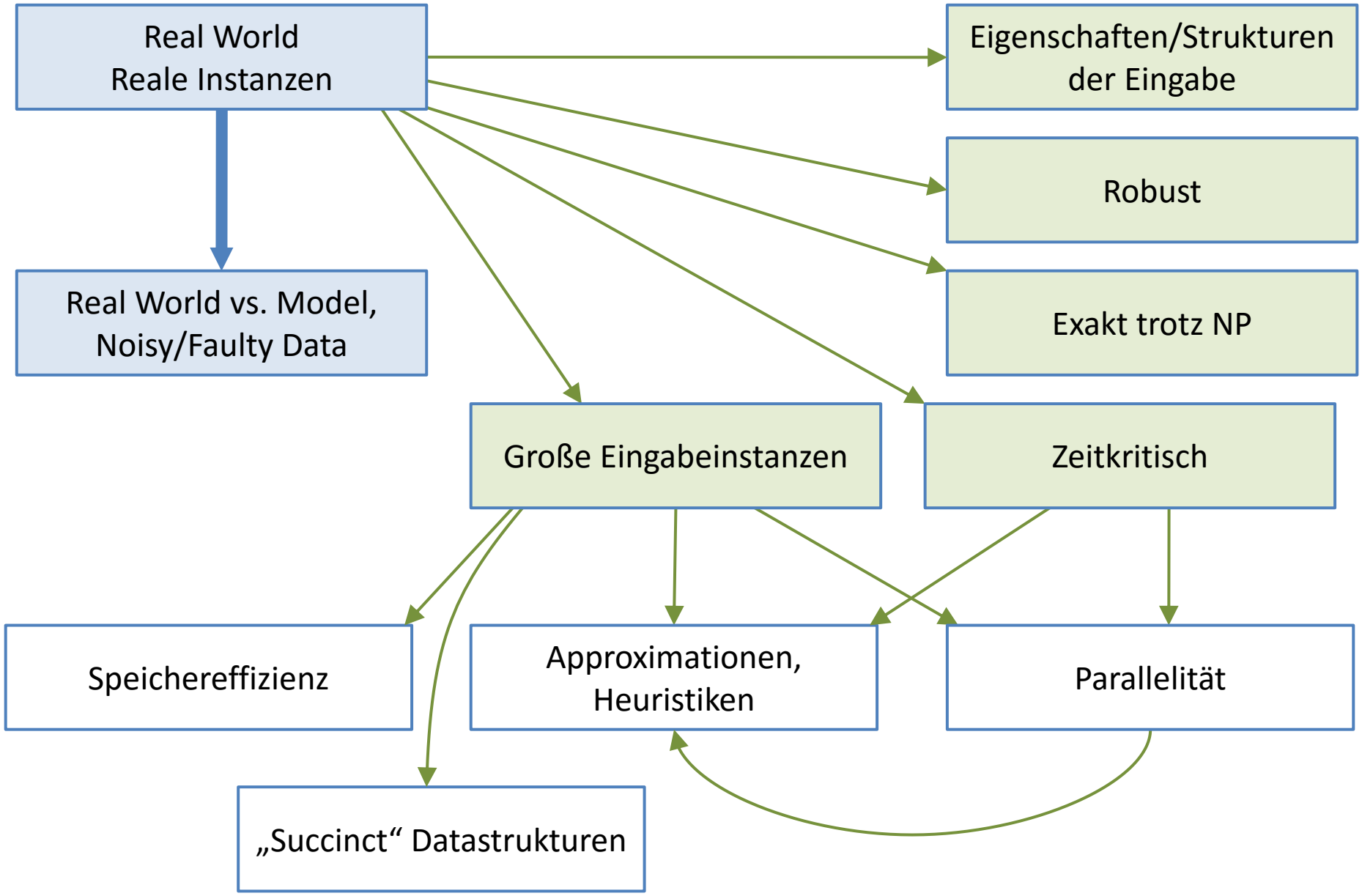
Große Themen



Große Themen



Große Themen



2VO + 4UE ?

Achtung, diese Lehrveranstaltung ist aufwendig!

2 SWS	Vorlesung	
2 SWS	Übung [Vorlesungsvertiefung]	} in 2-3er Gruppen
2 SWS	Übung [Projekt]	

Vorlesung:

- **Montags, 12:15–13:45, 69/E18**
- Auswahl subjektiv-spannender Themen aus diesen Bereichen,
- **zB.** Externspeicheralgorithmen, kürzeste Wege, Tourenplanung (TSP), *k*-Cardinality Trees, Textsuche/SuffixArrays, Kreuzungszahlen,…
- Folien auf **StudIP**

Prüfung

- Erfolgreiche Teilnahme (= Erledigen der zugewiesenen Aufgaben und aktive Mitarbeit) an **beiden** Teilen der Übung ist **Zulassungsvoraussetzung!**
- VO-Prüfung ist mündlich, 30min

2VO + 4UE ?

Achtung, diese Lehrveranstaltung ist aufwendig!

2 SWS	Vorlesung	
2 SWS	Übung [Vorlesungsvertiefung]	} in 2-3er Gruppen
2 SWS	Übung [Projekt]	

Übung [Vorlesungsvertiefung]:

- Jeden **zweiten** **Montag, 14:00–16:00, 69/E19**
  beginnend am 19.Okt (passiv) und 2.Nov (aktiv)
- Übung bringt 2 Stunden/**Woche**
  → mehr Zeit/Zeitaufwand in der Vorbereitung als bei gewöhnlichen wöchentlichen Übungen

Vertiefung des VO-Stoffes durch

- Denk- und Recherche-Fragen, Kurzvorträge zu weiterführenden Themen (auf Basis vorgegebener Veröffentlichungen)
- Kleine Implementierungen/Experimente
- Eine Aufgabe pro Übungsblatt und Gruppe

2VO + 4UE ?

Achtung, diese Lehrveranstaltung ist aufwendig!

2 SWS	Vorlesung	
2 SWS	Übung [Vorlesungsvertiefung]	} in 2-3er Gruppen
2 SWS	Übung [Projekt]	

Übung [Projekt]:

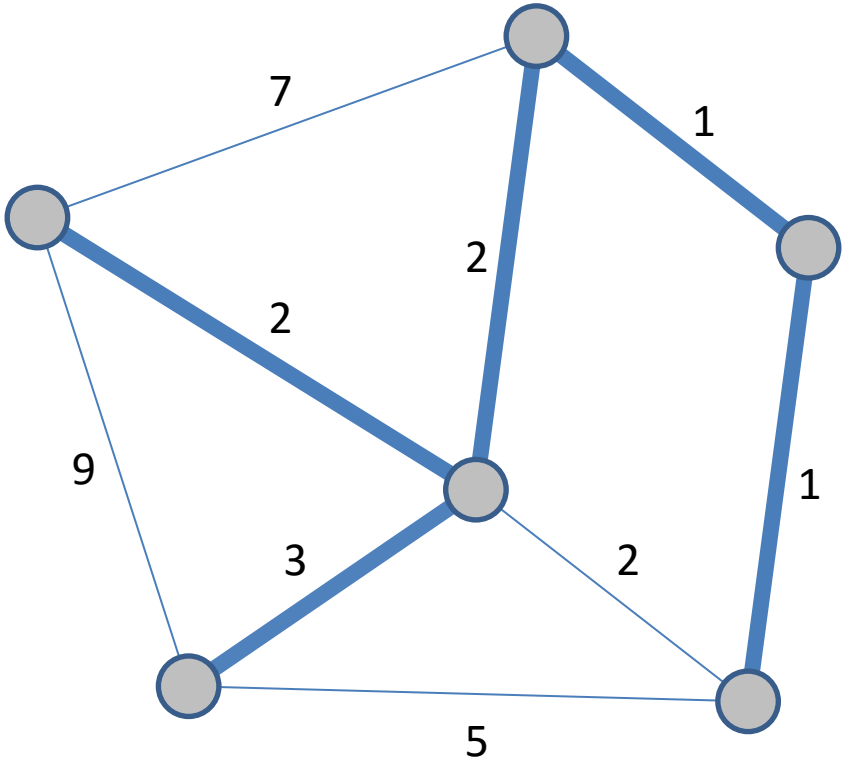
- Sie bekommen ein Optimierungsproblem als allgemeine Aufgabenstellung
- Lösen sie das Problem und betreiben Sie Algorithm Engineering…

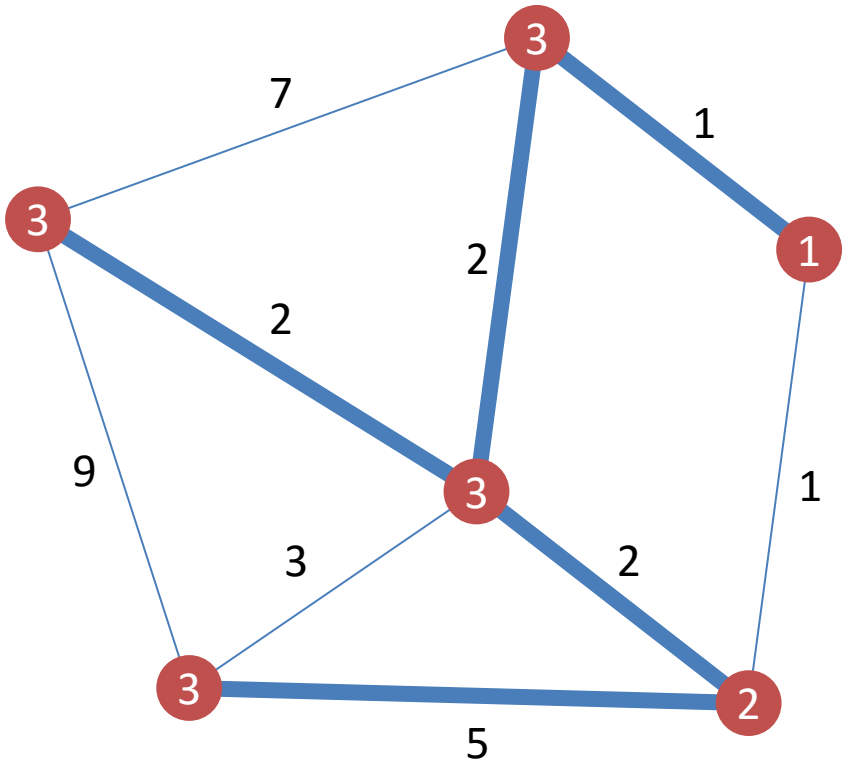
Zu tun:

- Recherche, Testinstanzen finden/generieren/…
- Heuristiken und exakte Verfahren überlegen
- Algorithmen implementieren [**C++**], testen, evaluieren,…
- Erkenntnisse gewinnen und in verbesserten Designs anwenden

Regelmäßige Treffen!
(Ca. alle zwei Wochen mit Betreuer, alle vier Wochen mit allen)

Ü-Projekt: Gradbeschr. Spannbäume





Für heute: Ende.

Bis nächsten Montag!