Algorithm Engineering für grundlegende Datenstrukturen und Algorithmen

Peter Sanders

Was sind die schnellsten implementierten Algorithmen

für das 1×1 der Algorithmik:

Listen, Sortieren, Prioritätslisten, Sortierte Listen, Hashtabellen,

Graphenalgorithemen?

Nützliche Vorkenntnisse

- Informatik I/II
- Algorithmentechnik (gleichzeitig hören vermutlich OK)
- etwa Rechnerarchitektur (oder Ct lesen ;-)
- passive Kenntnisse von C/C++

Vertiefungsgebiet: Algorithmik

Material

- Folien
- wissenschaftliche Aufsätze. Siehe Vorlesungshomepage
- Basiskenntnisse: Algorithmenlehrbücher, z.B. Cormen Leiserson Rivest???, Mehlhorn, Sedgewick, sowie ein Manuskript zu einem neuen Lehrbuch
- Mehlhorn Näher: The LEDA Platform of Combinatorial and Geometric Computing. Gut für die fortgeschrittenen Papiere.

Überblick

- Was ist Algorithm Engineering, Modelle, . . .
- Erste Schritte: Arrays, verkettete Listen, Stacks, FIFOs,. . .
- Sortieren rauf und runter
- Prioritätslisten
- Sortierte Listen
- Hashtabellen
- Minimale Spannbäume
- Kürzeste Wege
- Ausgewählte fortgeschrittene Algorithmen, z.B. maximale Flüsse

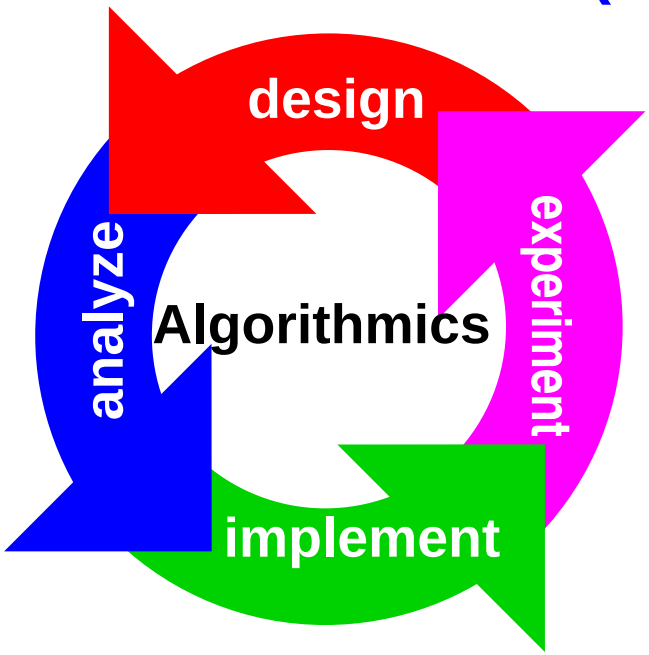
Methodik: in Exkursen

Algorithm Engineering
An Attempt at a Definition

Peter Sanders

Institut für Theoretische Informatik (ITI)

Universität Karlsruhe (TH)



Joint work with:

Kurt Mehlhorn

Rolf Möhring

Burkhardt Monien

Petra Mutzel

Dorothea Wagner

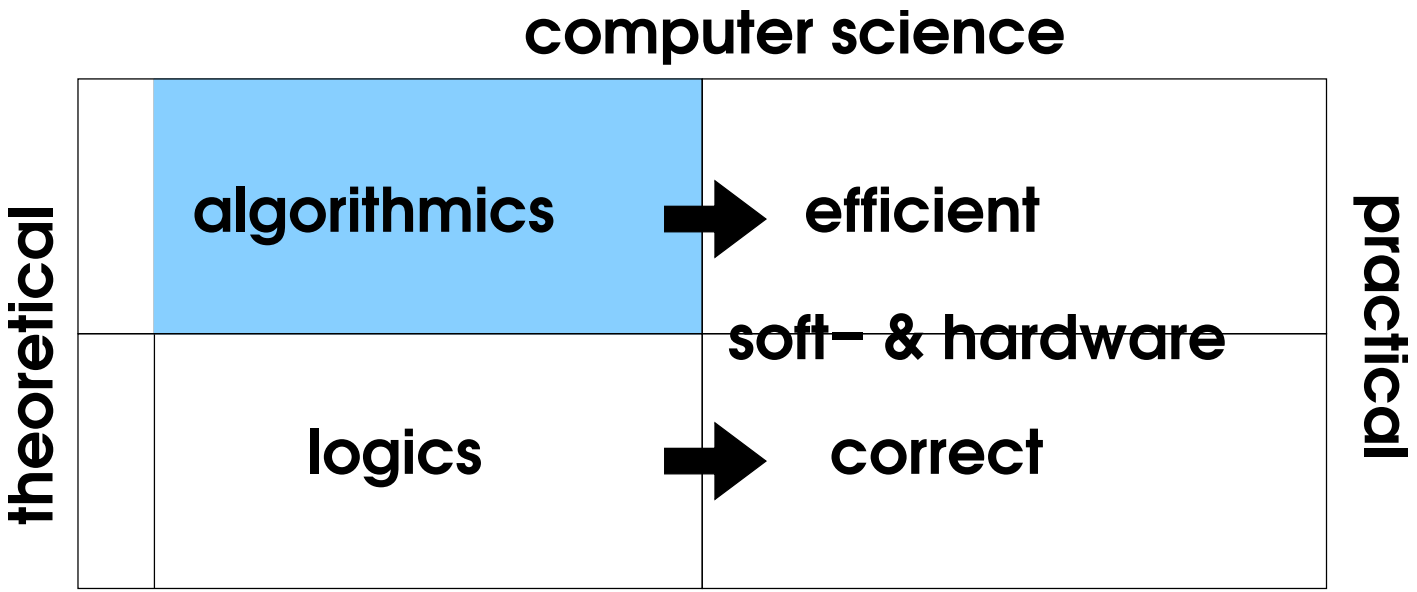
Source: Proposal for a DFG Focus Program.

Overview

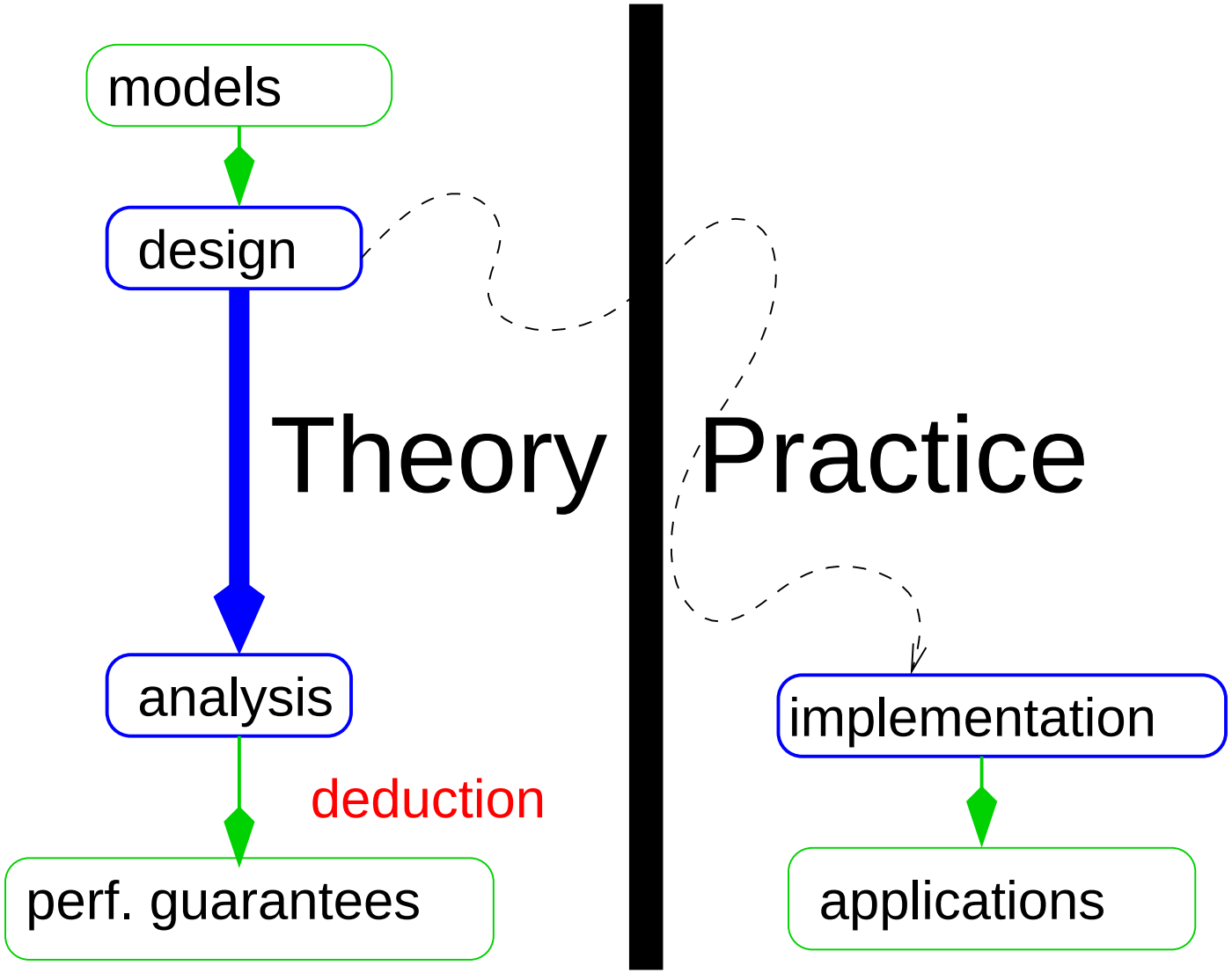
- Motivation
- Pieces of Methodology:
  - Examples
  - Challenges
- A Case Study: Car route planning
- Conclusion

Algorithmics

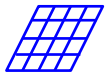
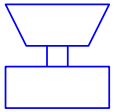
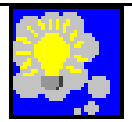
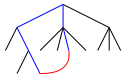
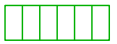
= the systematic design of efficient software and hardware



(Caricatured) Traditional View: Algorithm Theory



Gaps Between Theory & Practice

Theory	$\longleftrightarrow$	**Practice**
simple 	**appl. model**	 complex
simple 	**machine model**	 real
complex 	**algorithms**	FOR simple
advanced 	**data structures**	 arrays,. . .
worst case max	**complexity measure**	 inputs
asympt. $\mathcal{O}(\cdot)$	**efficiency**	42% constant factors

Why Bridge Gaps?

- With growing problems size asymptotics will eventually win
- Worst case bounds
  - ⇝ performance guarantees
  - ⇝ quality, real time properties
- Theory in the natural science means:

  Theory explains reality

Warum diese Vorlesung?

- Jeder Informatiker kennt einige Lehrbuchalgorithmen

  $\rightsquigarrow$ wir können gleich mit Algorithm Engineering loslegen

- Viele Anwendungen profitieren

- Es ist frappierend, dass es hier noch Neuland gibt

- Basis für Studien- Diplomarbeiten

Was diese Vorlesung nicht ist:

Keine wiedergekäute Algorithmentechnik o.Ä.

- Grundvorlesungen “vereinfachen” die Wahrheit oft
- z.T. fortgeschrittene Algorithmen
- steilere Lernkurve
- Implementierungsdetails
- Betonung von Messergebnissen

Was diese Vorlesung nicht ist:

Keine Theorievorlesung

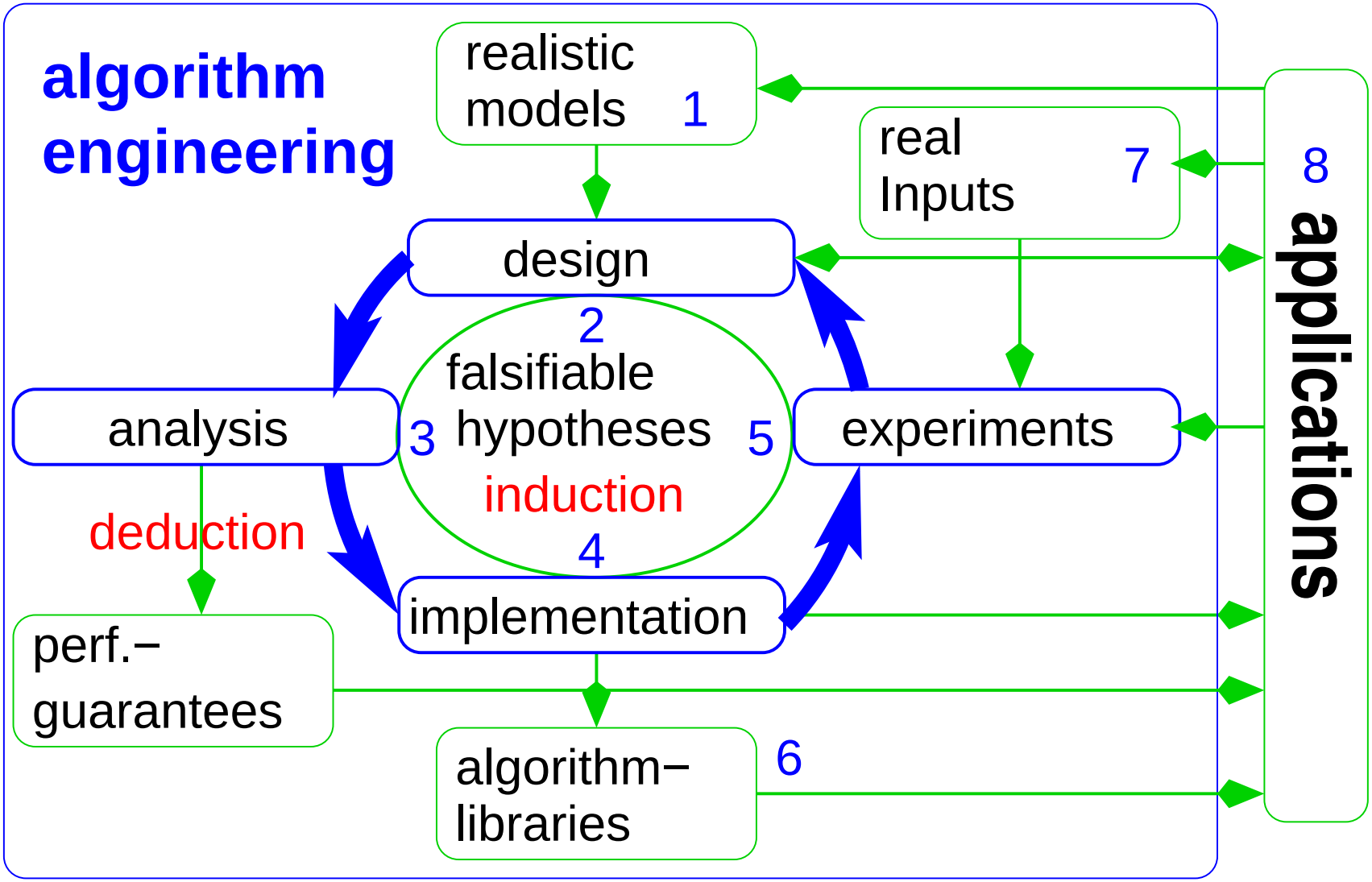
- keine (wenig?) Beweise
- Reale Leistung vor Asymptotik

Was diese Vorlesung nicht ist:

Keine Implementierungsvorlesung

- Etwas Algorithmenanalyse,. . .
- Wenig Software Engineering
- Keine Implementierungsübungen (aber, Praktikum!)

Algorithmics as Algorithm Engineering



Goals

- bridge gaps between theory and practice
- accelerate transfer of algorithmic results into applications
- keep the advantages of theoretical treatment:
  generality of solutions and
  reliabiltiy, predictabilty from performance guarantees

Exkurs: Maschinenmodelle

RAM/von Neumann Modell

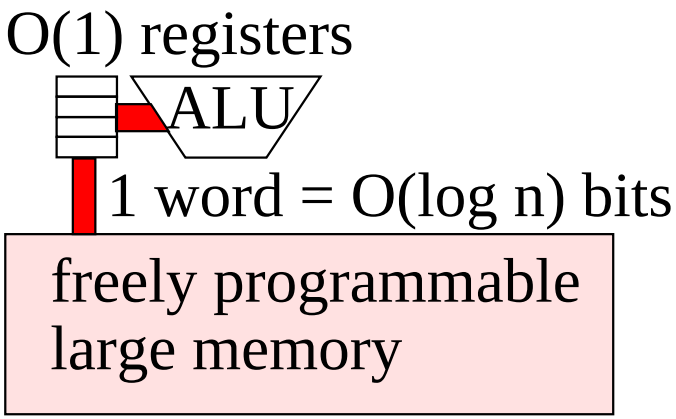
Analyse: zähle Maschinenbefehle —

load, store, Arithmetik, Branch,. . .

- Einfach
- Sehr erfolgreich
- zunehmrend unrealistisch

  weil reale Hardware

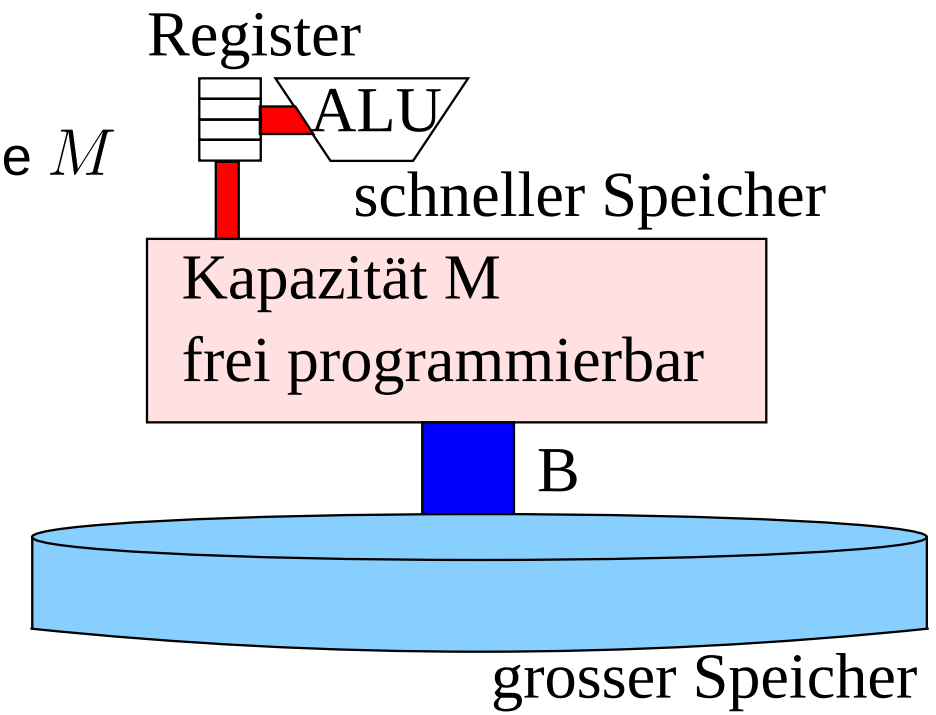
  immer komplexer wird



Das Sekundärspeichermodell

M: Schneller Speicher der Größe M

B: Blockgröße



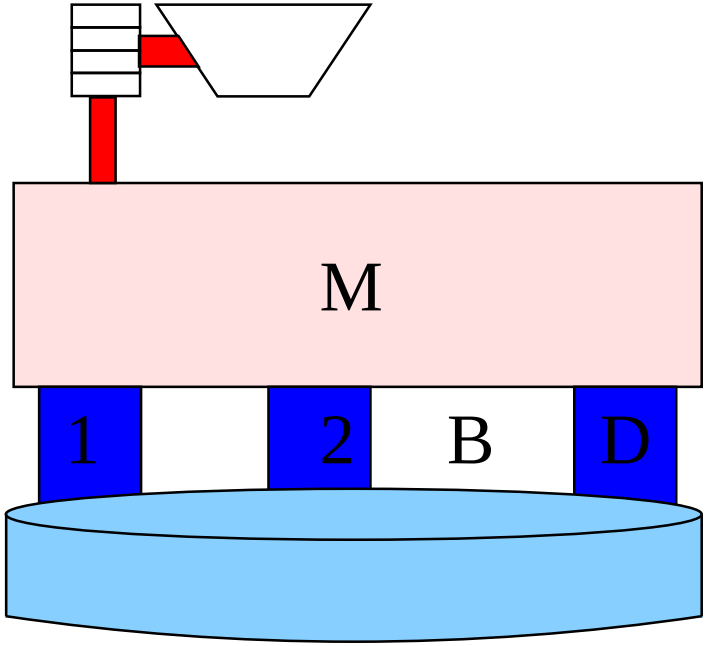
Analyse: zähle (nur?) Blockzugriffe (I/Os)

Interpretationen des Sekundärspeichermodelle

	Externspeicher	Caches
großer Speicher	Platte(n)	Hauptspeicher
M	Hauptspeicher	ein cache level
B	Plattenblock (MBytes!)	Cache Block (16–256) Byte

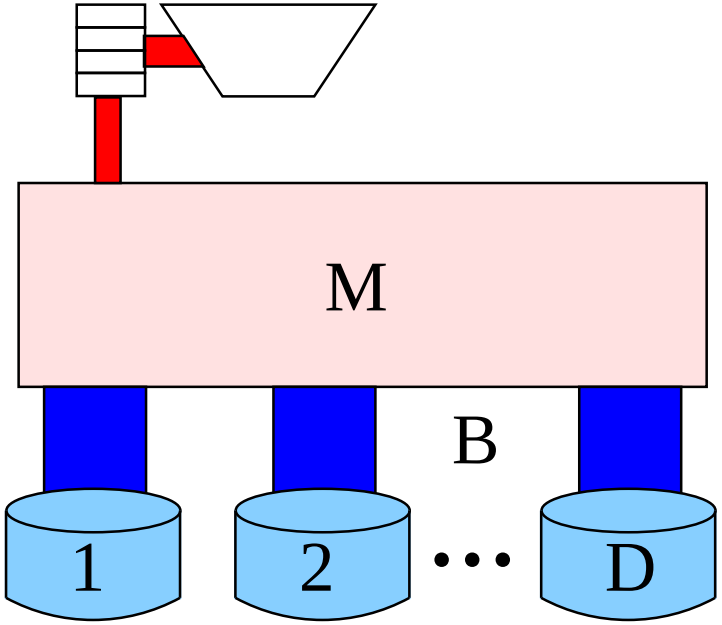
Ggf. auch zwei cache levels.

Parallele Platten



Mehrkopfmodell

[Aggarwal Vitter 88]



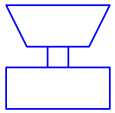
unabhängige Platten

[Vitter Shriver 94]

Mehr Modellaspekte

- Instruktionsparallelismus (Superscalar, VLIW, EPIC,SIMD,...)
- Pipelining
- Was kostet branch misprediction?
- Multilevel Caches (gegenwärtig 2–3 levels) $\rightsquigarrow$ “cache oblivious algorithms”
- Parallele Cores/Prozessoren, Multithreading $\rightsquigarrow$ Vorlesung Parallele Algorithmen
- Kommunikationsnetzwerke
- ...

1a: Realistic Models for Machines

Theory	⟷	**Practice**
simple 	**machine model**	 real

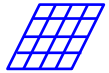
Driving force for many interesting areas:

Memory Hierarchies: I/O- model, cache oblivious model

Network: Streaming algorithms

Parallelism: BSP,. . . But what about processor hierarchies?

1b: Realistic Models for Problems

Theory	⟷	**Practice**
simple 	**appl. model**	 complex

Incomplete information: online algorithms, competitive ratio, ressource augmentation, stochastic optimization,. . .

Selfish Users: traffic planning, internet applications,. . .

Flow Problems: flows over time,. . .

2: Algorithm Design

- simplicity
- reuse
- constant factors
- exploit easy instances

2: Design Example

simple randomized external

Minimum Spanning Tree alg.

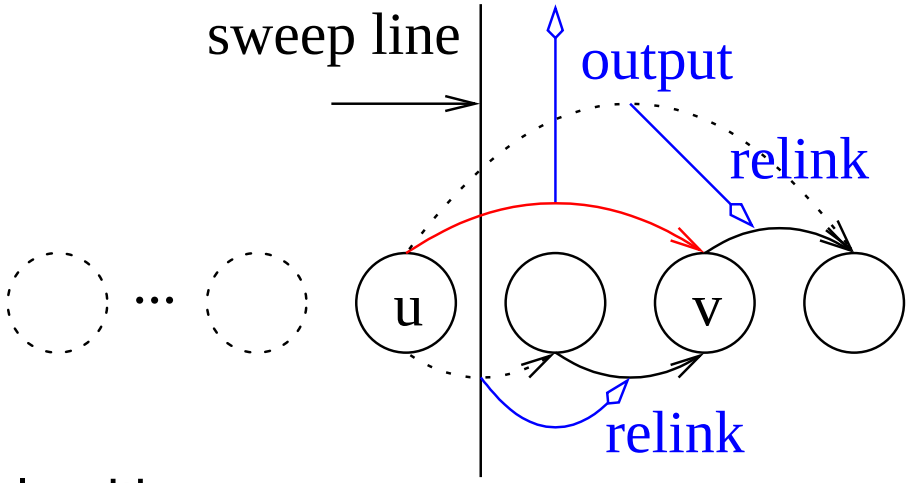
reuses priority Queue.

constant factors: $\mathcal{O}\left(\log \frac{|\text{Edges}|}{\text{cacheSize}}\right)$

more I/Os than best theoretical algorithm.

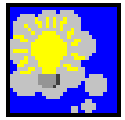
Factor ≥ 3 better for realistic hardware

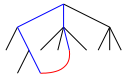
[Dementiev Sanders Schultes Sibeyn 03].



instances: asymptotically optimal for planar graphs

2: Design Challenges





FOR

- Reliably exploiting integer keys
  ($n\sqrt{\log\log n}$ sorting,... ???) $\leftrightarrow$ quicksort
- Computational geometry $\leftrightarrow$ grid and pixel algs.
- Approximation algorithms $\leftrightarrow$ OR approaches

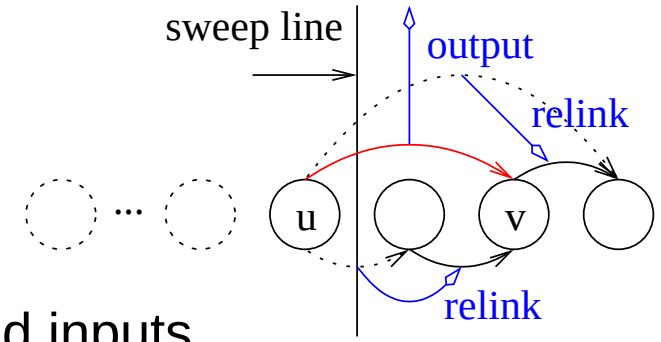
3: Analysis — Issues and Examples

constant factors: external MST

randomized algorithms: Open Problem:

sweeping algorithm for external

Connected Components [Sibeyn]



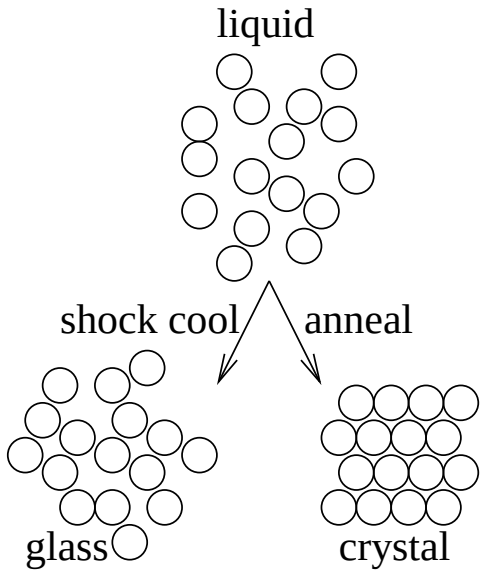
smoothed analysis: Analysis for randomly perturbed inputs.

Simplex algorithm [Spielmann Teng],

Knapsack [Beier Vöcking]

meta heuristics: genetic, annealing, ant colony. . . ,

e.g. for multi-level graph partitioning



4: Implementation

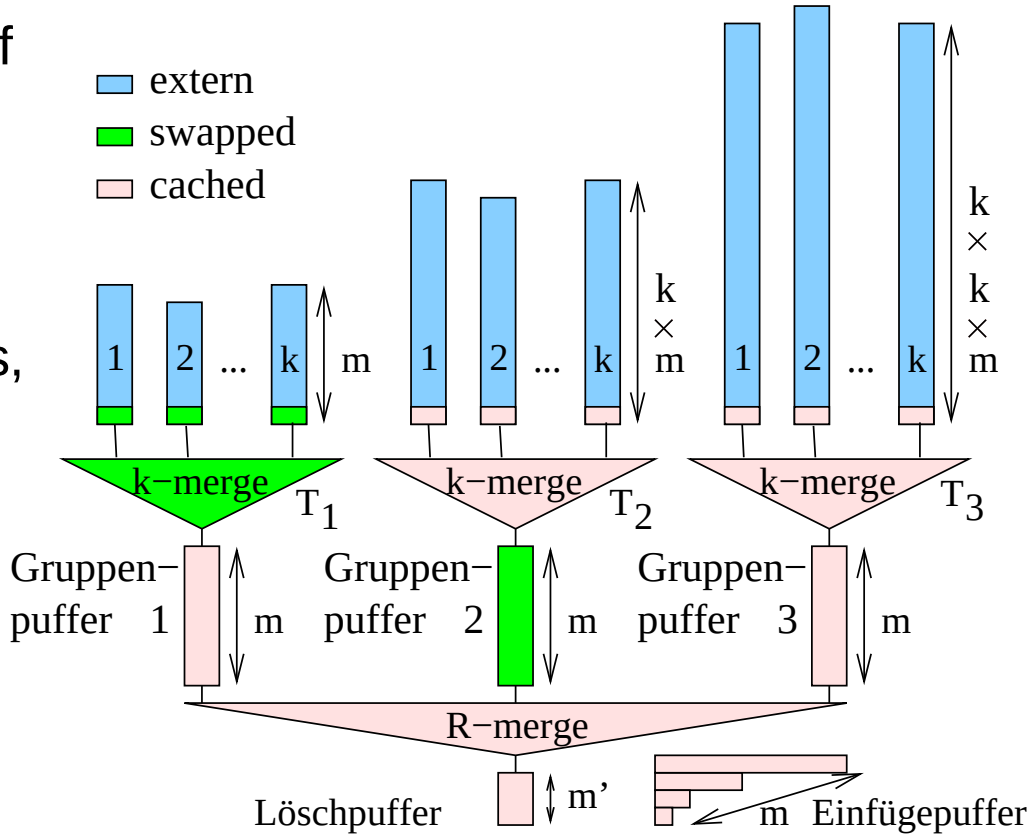
Semantic gaps: Abstract algorithm ⟷ C++... ⟷ hardware

Extreme example: assumption of exact arithmetics in geometry

Small constant factors:

compare highly tuned competitors,

e.g. Priority queues [Sanders 00]



4: Implementations

provide evidence of correctness

Example:

Embedding of a planar graph

[Hopcroft Tarjan 74]$\leftrightarrow$[Mehlhorn Mutzel 96]

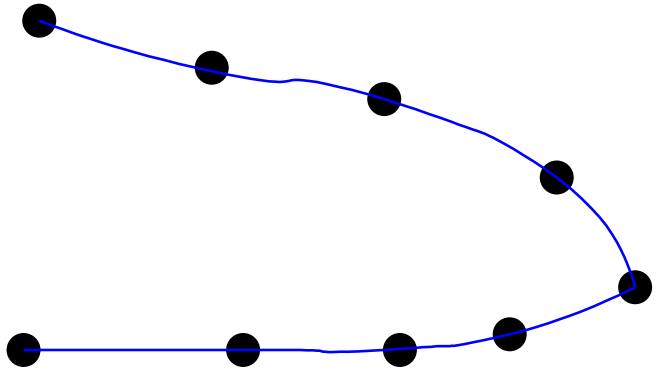
5: Experiments — Examples

Actually falsify hypotheses: Crossing minimization [Jünger Mutzel 97]

$\rightsquigarrow$ realistic instances filter out algs. of limited relevance

Avoid undue pessimism: Curve reconstruction [Althaus Mehlhorn 02]

$\rightsquigarrow$ easy instances of TSP



Suggest performance bounds: Disk scheduling [San. Egner Korst 00]

$\rightsquigarrow$ (almost) ideal performance

5: Experiments — Challenges

- too much rather than too little data
- reproducibility (10 years!)
- software engineering

Build best practice examples and tools, e.g.,

`http://explab.sourceforge.net/`

6: Algorithm Libraries — Examples

Basics: LEDA, Boost, java.util

established but still room for high performance implementations

External: STXXL/TPIE

beginning to get useful

Parallization: MPI, Thread libraries

big gap between known practical algorithms and implementations

Geometry: CGAL

Graph drawing: AGD

6: Algorithm Libraries — Challenges

- software engineering , e.g. CGAL
- standardization, e.g. java.util, C++ STL and BOOST
- applications are a priori unknown
- performance ↔ generality ↔ simplicity
- result checking, verification

7: Problem Instances

Benchmark instances for NP-hard problems

- TSP
- Steiner-Tree
- SAT
- set covering
- graph partioning
- ...

have proved essential for develpment of practical algorithms

Strange: much less real world instances for polynomial problems

(MST, shortest path, max flow, matching...)

8: Applications that “Change the World”

Algorithmics has the potential to SHAPE applications

(not just the other way round) [G. Myers]

Bioinformatics: sequencing, proteomics, phylogenetic trees,. . .

Information Retrieval: Searching, ranking,. . .

Traffic Planning: navigation, flow optimization,

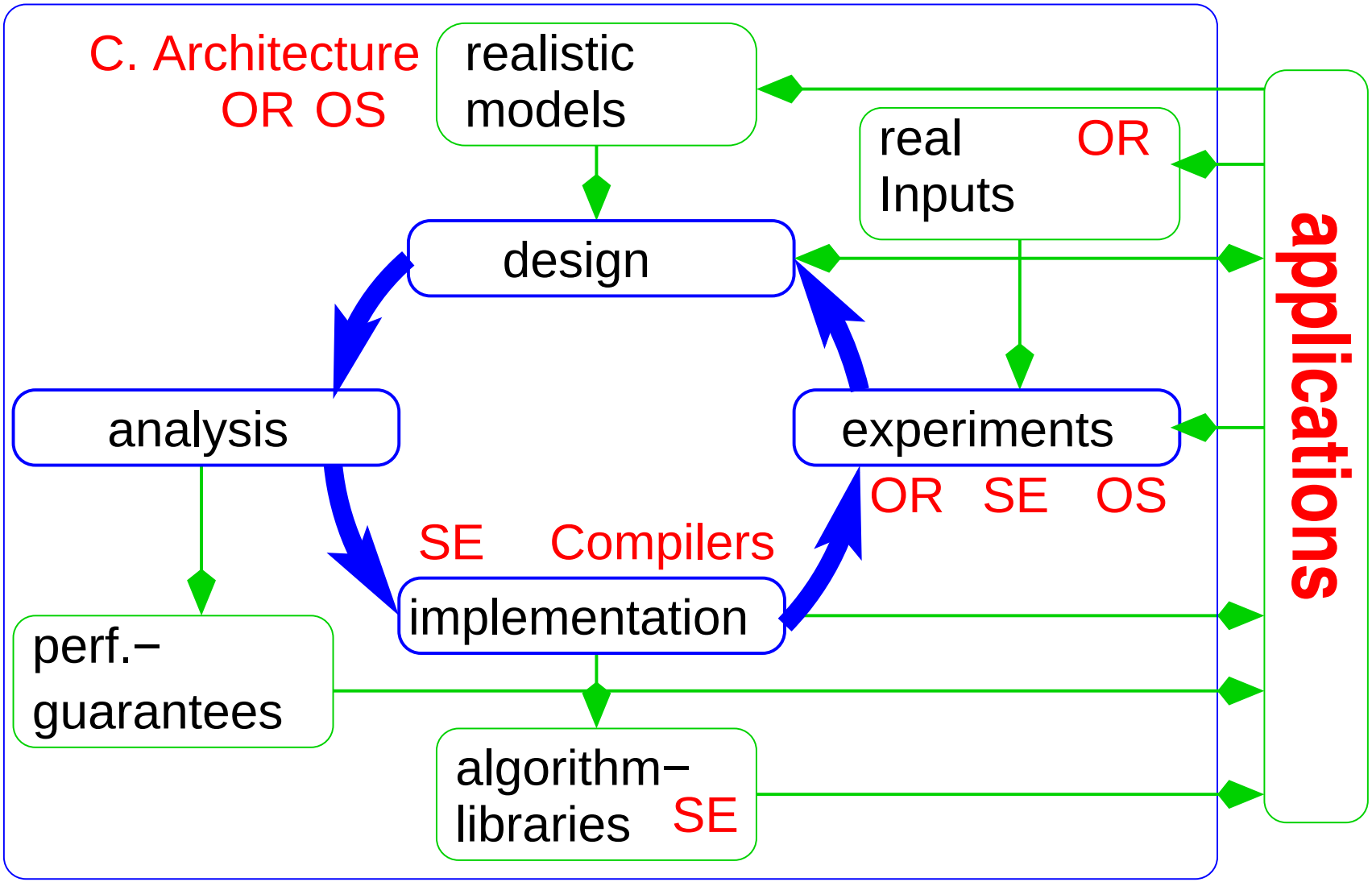
adaptive toll, disruption management

Geographic Information Systems: agriculture, environmental protection,

disaster management, tourism,. . .

Communication Networks: mobile, P2P, grid, selfish users,. . .

Interactions with other (Sub)disciplines



Case Study: Car Route Planning

Model:

Many s-t query in a directed graph $G = (V, E)$ with nonnegative edge weights.

Huge, sparse graphs.

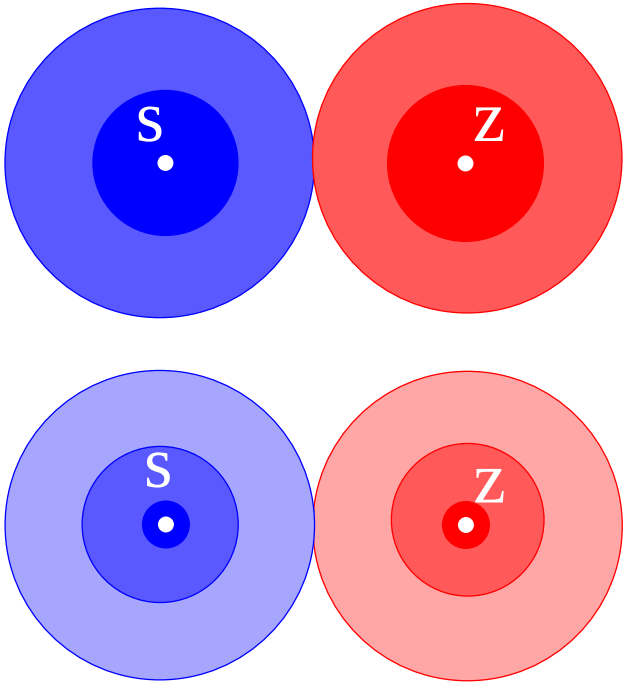
fast, space efficient preprocessing allowed.

Challenges:

- Edge weight updates (traffic jams,...)
- Tunable objective functions (time, distance, fuel, toll,...)
- Time dependent edge weights
- Interactions with flow optimization

Design: Exact Highway Hierarchies

- complete search in local area
- search in (more sparse) highway network
- iterate $\rightsquigarrow$ highway hierarchy



Defining the highway network:

minimal network,

that preserves all shortest paths.

Compress trees and nodes of degree two

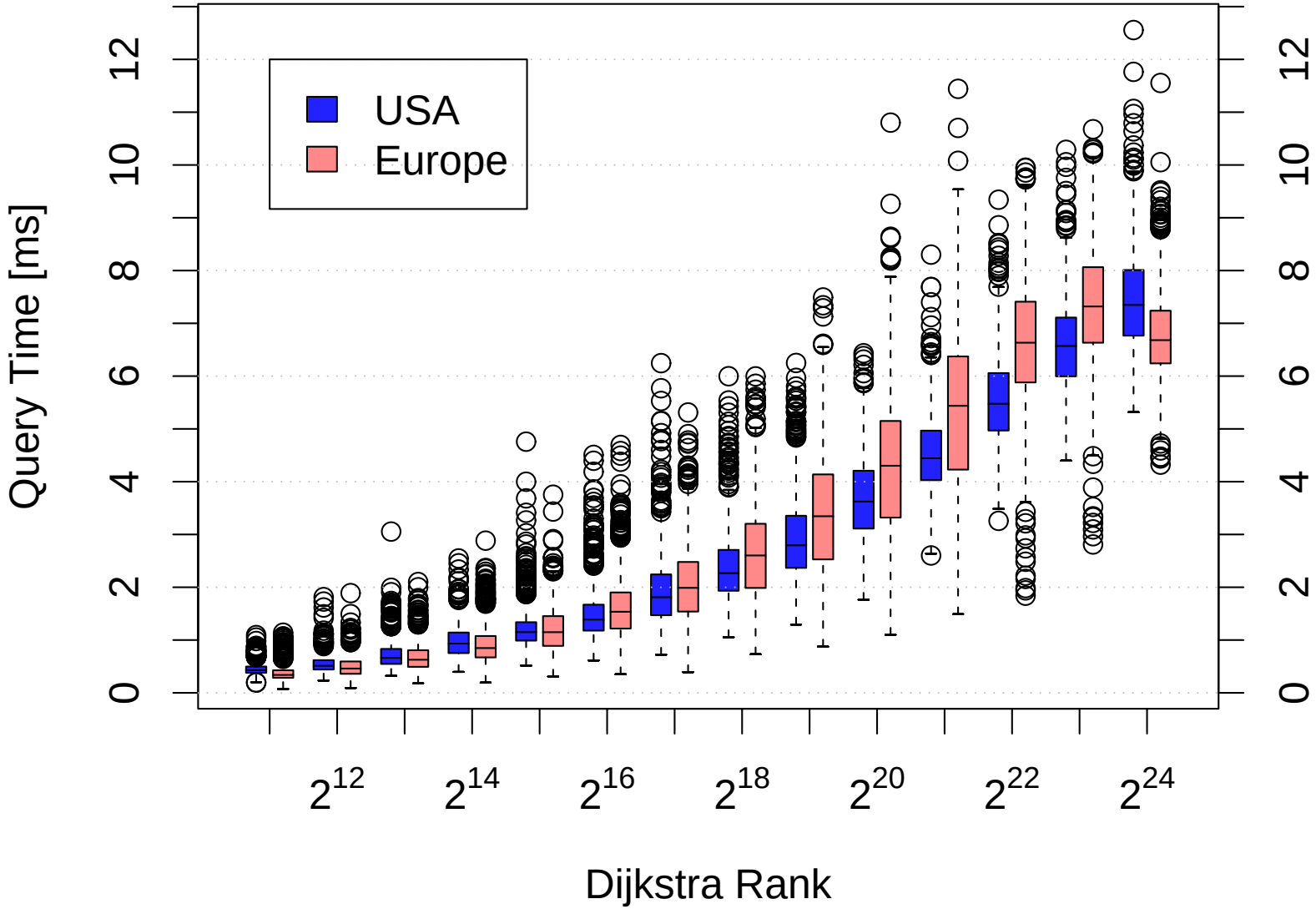
Car Route Planning: Analysis

- Prove optimality of pruned search
- query time for model instances
- Analyze space consumption, preprocessing time,...

Car Route Planning: Implementation

- priority queues are much LESS relevant than in naive Dijkstra
- space efficiency much sought after by practitioners

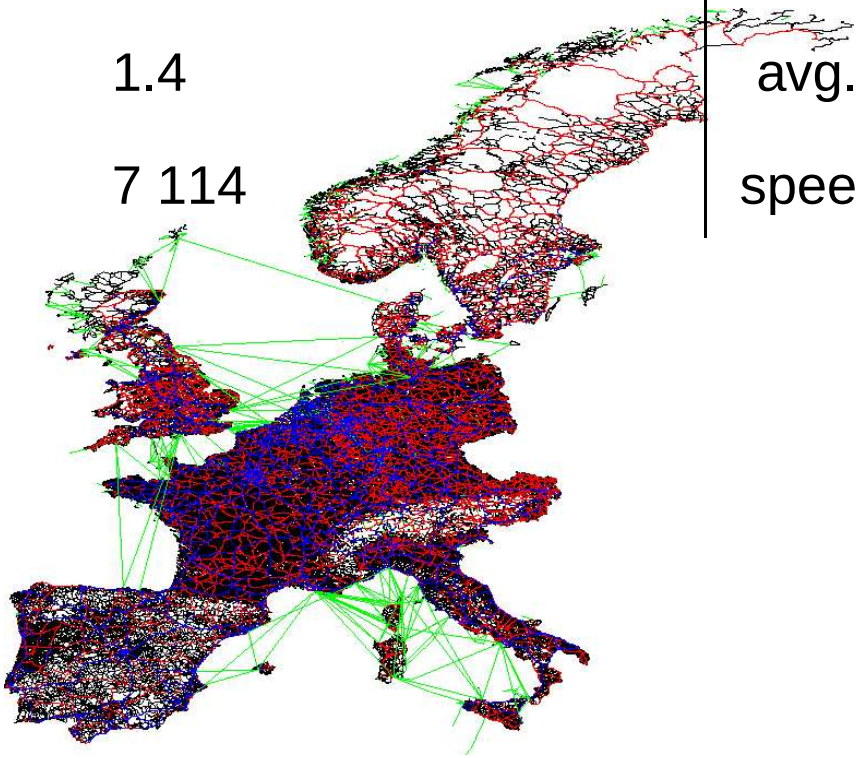
Experiments: Query Time Distribution

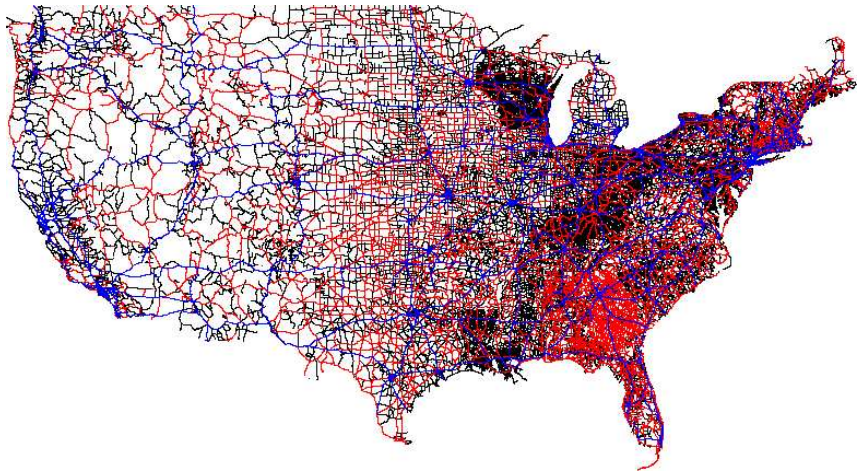


Algorithm Libraries for Car Route Planning?

- generic priority queues provides several variants (break ties for equal length paths during preprocessing)
- Can we decouple implementation from representation?
  (so customers can keep old data-base and graphics code)

W. Europe (PTV)	**Instances**	USA (Tiger Line)
18 029 721	#nodes	24 278 285
42 199 587	#edges	58 213 192
21	preproc time [h]	24
1.4	avg. query time [ms]	1.75
7 114	speedup over Dijkstra	6 106





Application Requirements

- space
- turn restrictions
- compatibility with existing representations

Algorithm Engineering versus Algorithm Theory

- algorithm engineering is a wider view on algorithmics
  (but no revolution. None of the ingredients is really new)
- rich methodology
- better coupling to applications
- experimental algorithmics $\ll$ algorithm engineering
- algorithm theory $\subset$ algorithm engineering
- sometimes different theoretical questions
- algorithm theory may still yield the strongest, deepest and most persistent results within algorithm engineering