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,

Graphenalgorithemen? $+$ Exkurse in die ITI-Forschung: z.B.

Routenplanung

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, . . .
- Exkurs: Routenplanung
- Erste Schritte: Arrays, verkettete Listen, Stacks, FIFOs,. . .
- Sortieren rauf und runter
- Prioritätslisten
- Sortierte Listen
- Minimale Spannbäume
- Volltextindices
- Kürzeste Wege
- Ausgewählte fortgeschrittene Algorithmen, z.B. maximale Flüsse

Methodik: in Exkursen

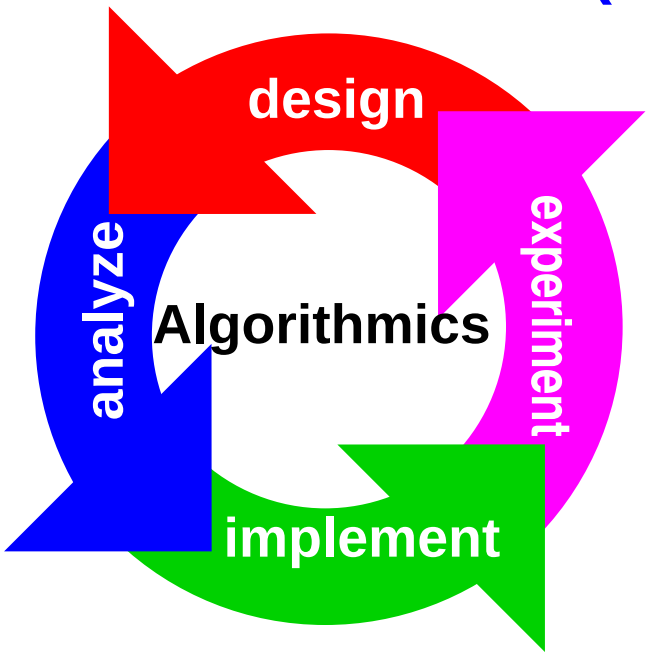
Algorithm Engineering

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

More: DFG Focus Program `www.algorithm-engineering.de`

Overview

A detailed definition

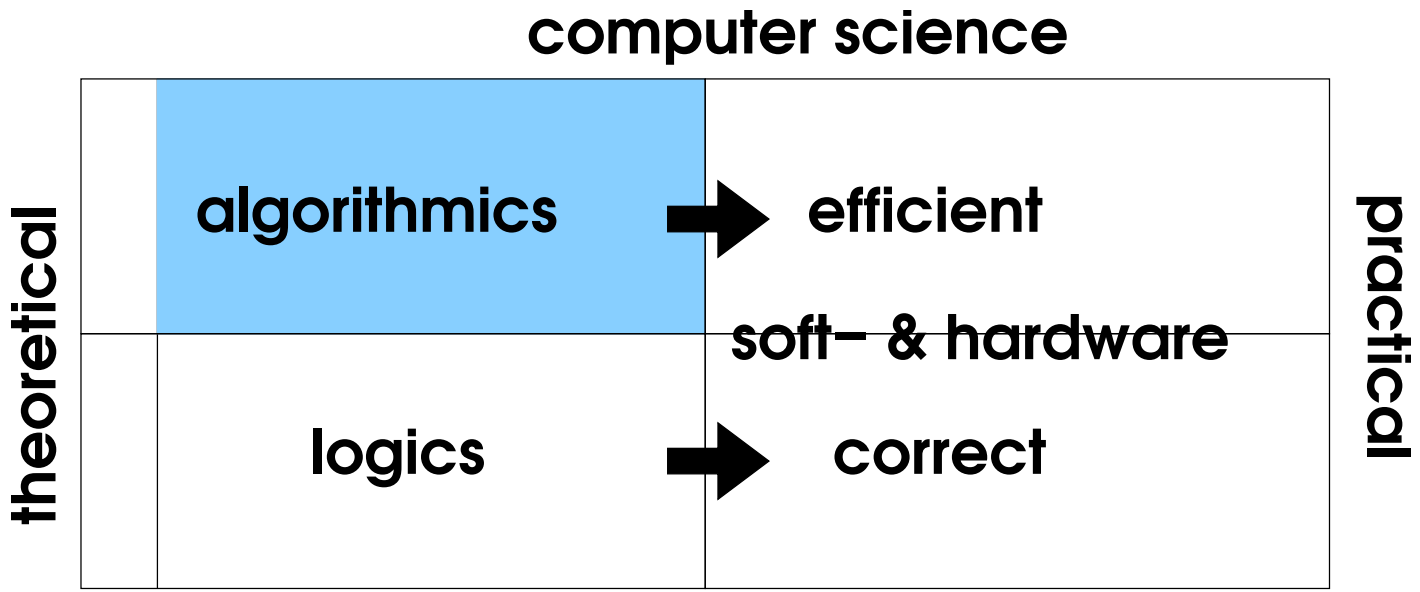
- in general
- simple examples, mostly external MSTs

More algorithm engineering in my group

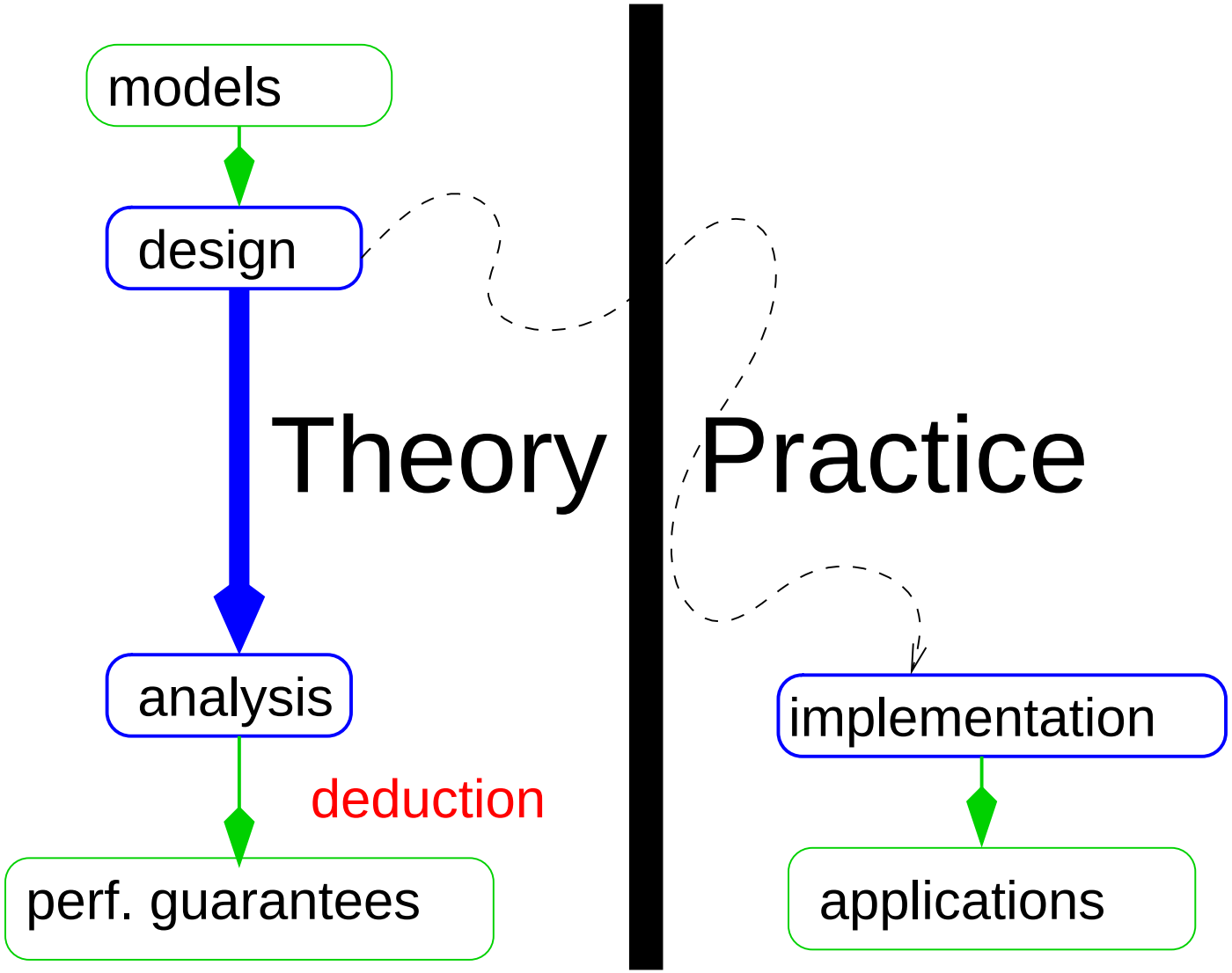
- The basic toolbox
- Various applications

Algorithmics

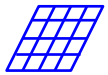
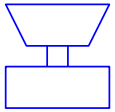
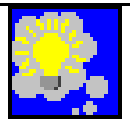
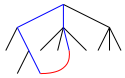
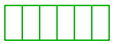
= the systematic design of efficient software and hardware



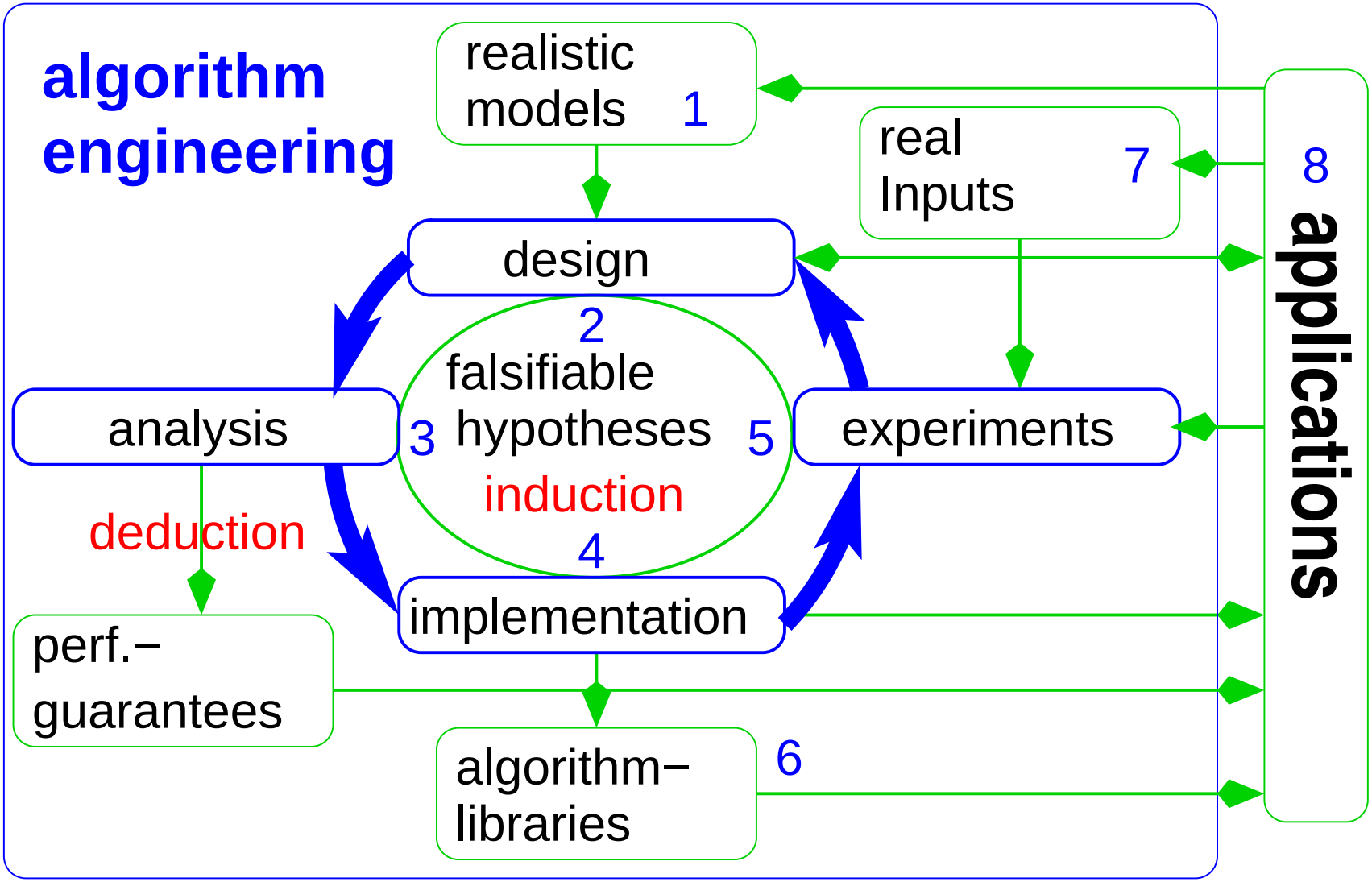
(Caricatured) Traditional View: Algorithm Theory



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

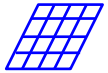
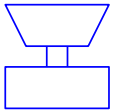
Algorithmics as Algorithm Engineering



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

1: Realistic Models

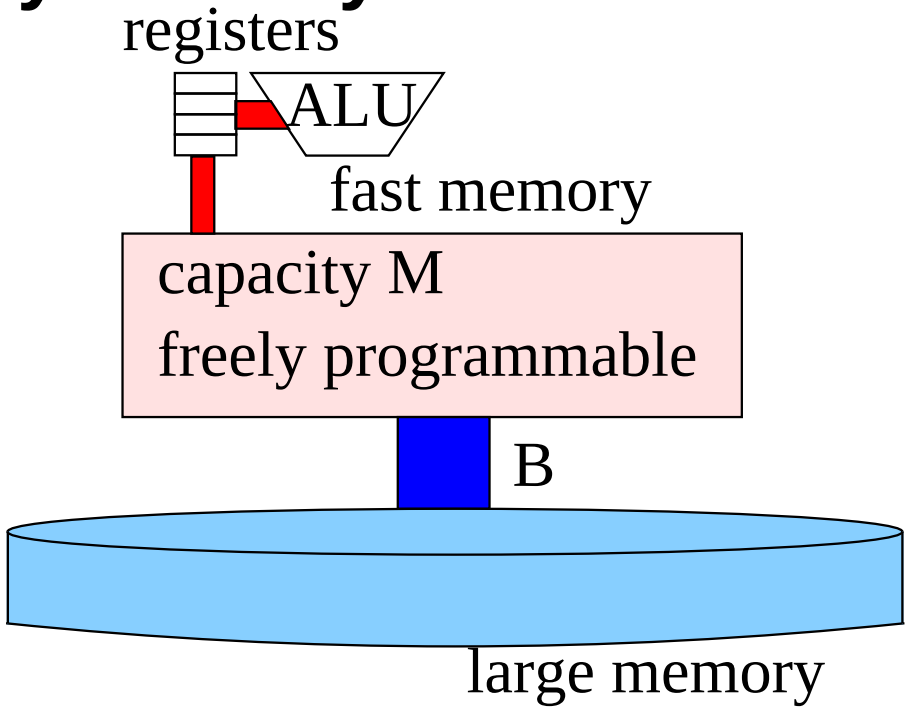
Theory	⟷	**Practice**
simple 	**appl. model**	 complex
simple 	**machine model**	 real

- Careful refinements
- Try to preserve (partial) analyzability / simple results

Example: The Secondary Memory Model

M: Fast memory of size M

B: Block size (?)



Analysis: count block accesses (I/Os)

e.g. Sorting in $\mathrm{sort}(n) := \frac{2n}{DB}\left(1 + \left\lceil \log_{M/B} \frac{n}{M} \right\rceil\right)$ I/Os

Variants: D parallel disks, cache oblivious (M, B unknown)

Challenge: parallel, hierarchical memory model

2: Design

of algorithms that work well in practice

- simplicity
- reuse
- constant factors
- exploit easy instances

3: Analysis

- Constant factors matter
- Beyond worst case analysis
- Practical algorithms might be difficult to analyze (randomization, meta heuristics,. . .)

4: Implementation

sanity check for algorithms !

Challenges

Semantic gaps:

Abstract algorithm

$\longleftrightarrow$

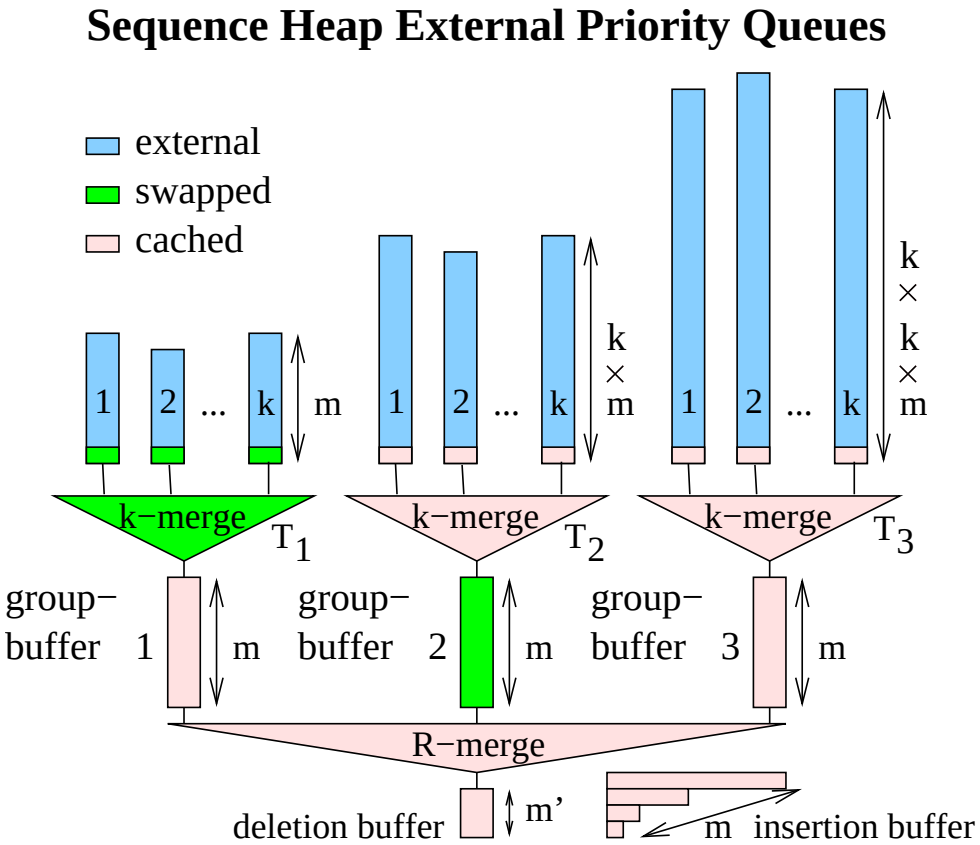
C++...

$\longleftrightarrow$

hardware

Small constant factors:

compare highly tuned competitors,

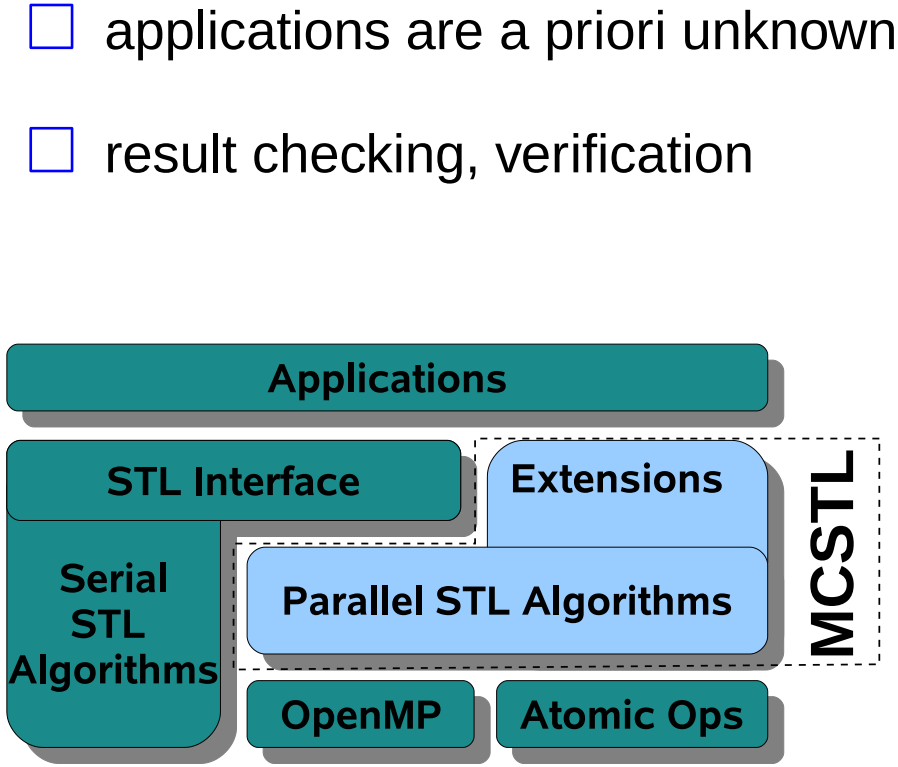


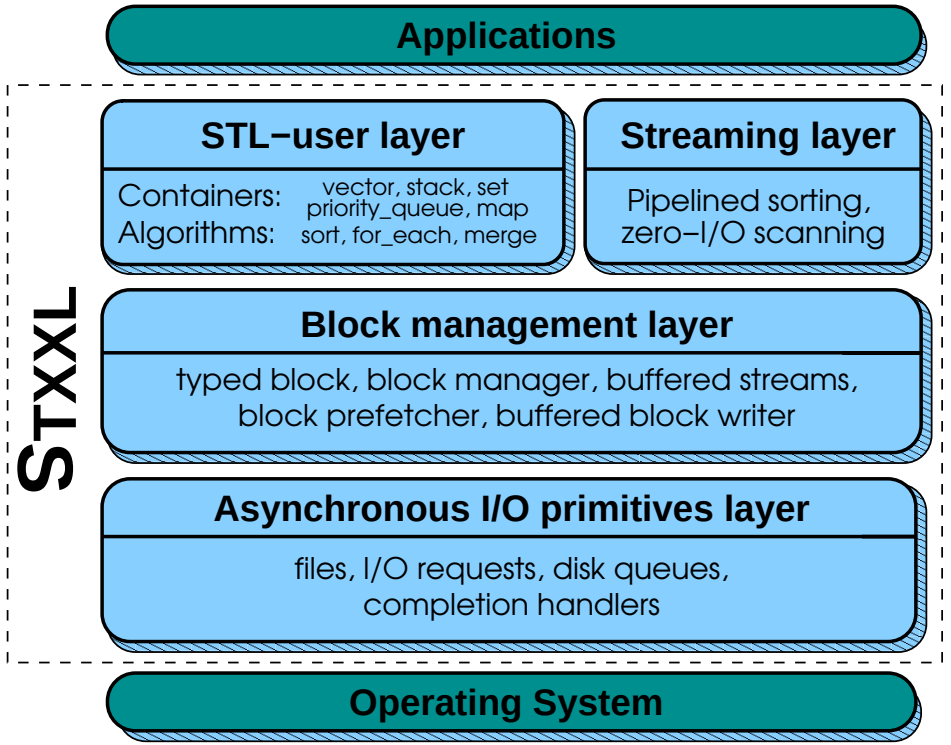
5: Experiments

- sometimes a good surrogate for analysis
- too much rather than too little output data
- reproducibility (10 years!)
- software engineering

6: Algorithm Libraries — Challenges

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

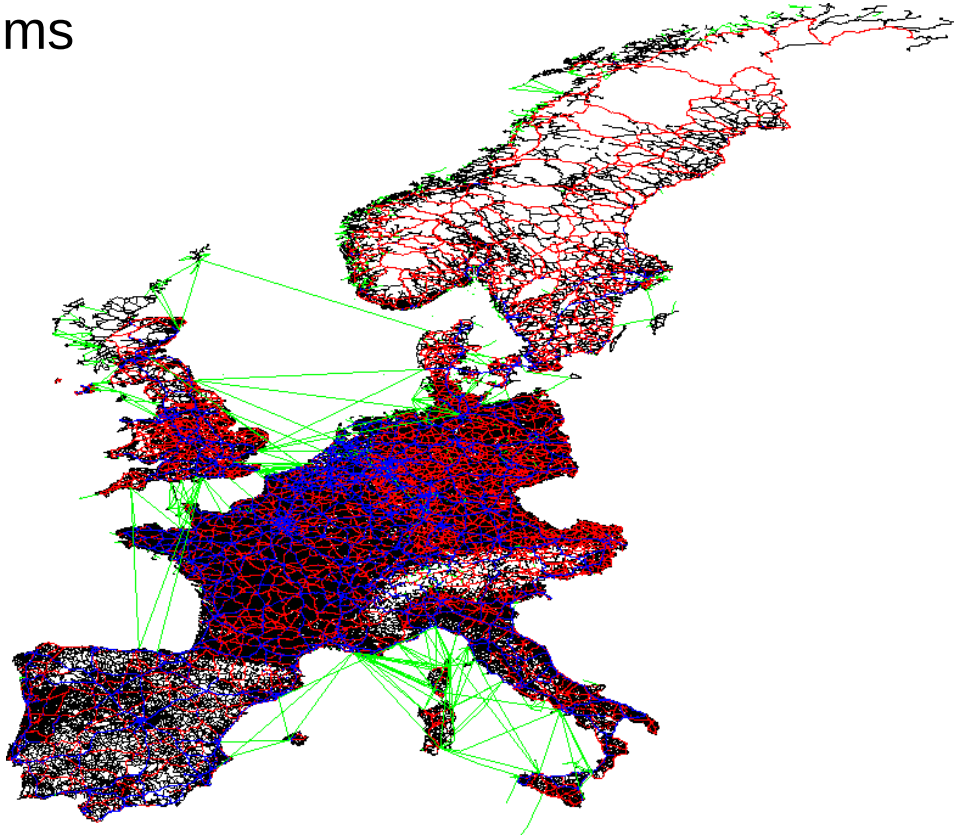




7: Problem Instances

Benchmark instances for NP-hard problems

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



have proved essential for development 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,. . .

Conclusion:

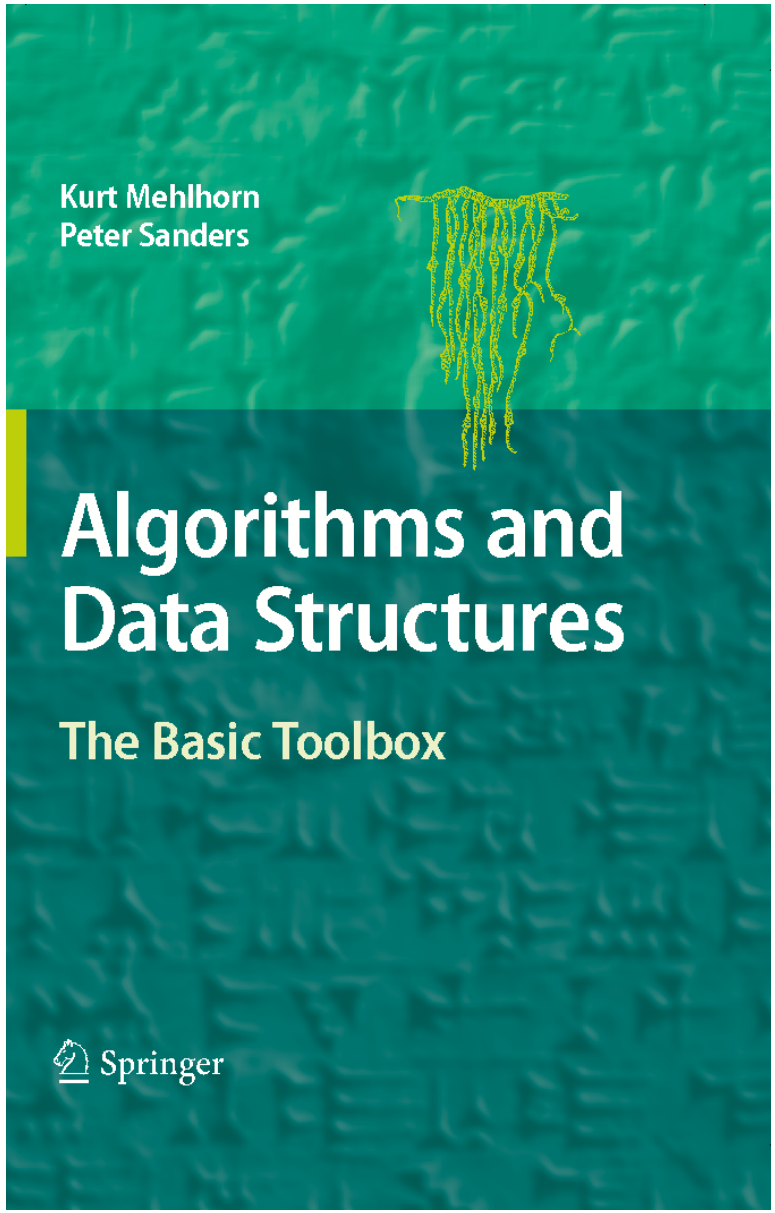
Algorithm Engineering $\leftrightarrow$ Algorithm Theory

- 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

Fundamental Algorithms

- lists, array, stacks, queues
- sorting
- priority queues
- sorted lists / search trees
- hash tables
- graph algorithms
  - graph traversal (DFS, BFS)
  - shortest paths
  - minimal spanning trees
  - flow problems
- strings



Application Areas

- combinatorial optimization
- parallel computing
  - C++ Standard Library for Multicore
  - MPI communication primitives
  - load balancing
- external memory computations
- scalable (virtual) storage servers
- information retrieval

here

worked on

$\longrightarrow$ Cooperation with SAP TREX group

Interactions with other (Sub)disciplines