Mini Course Algorithm Engineering

Bergen 2008

Overview

- An attempt at a definition
- Sorting
- Approximate weighted matching
- Route planning
- Presenting experimental data
- External suffix sorting
- More (external algorithms) ?

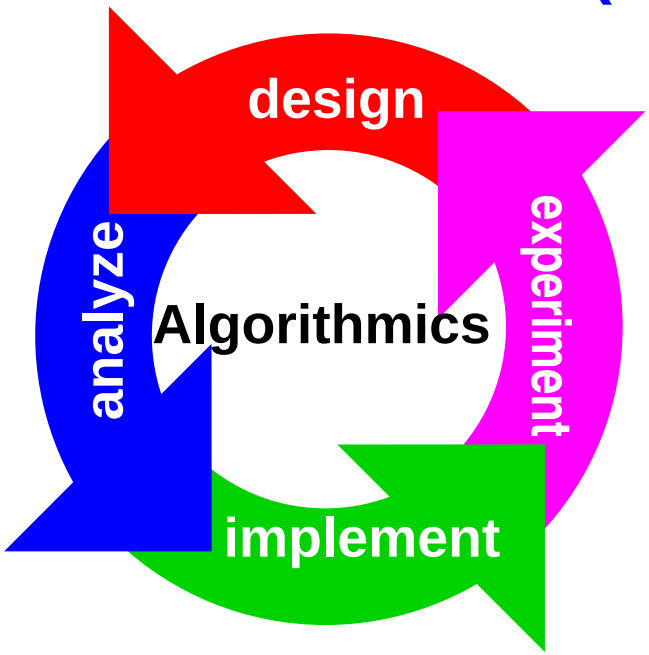
Algorithm Engineering

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

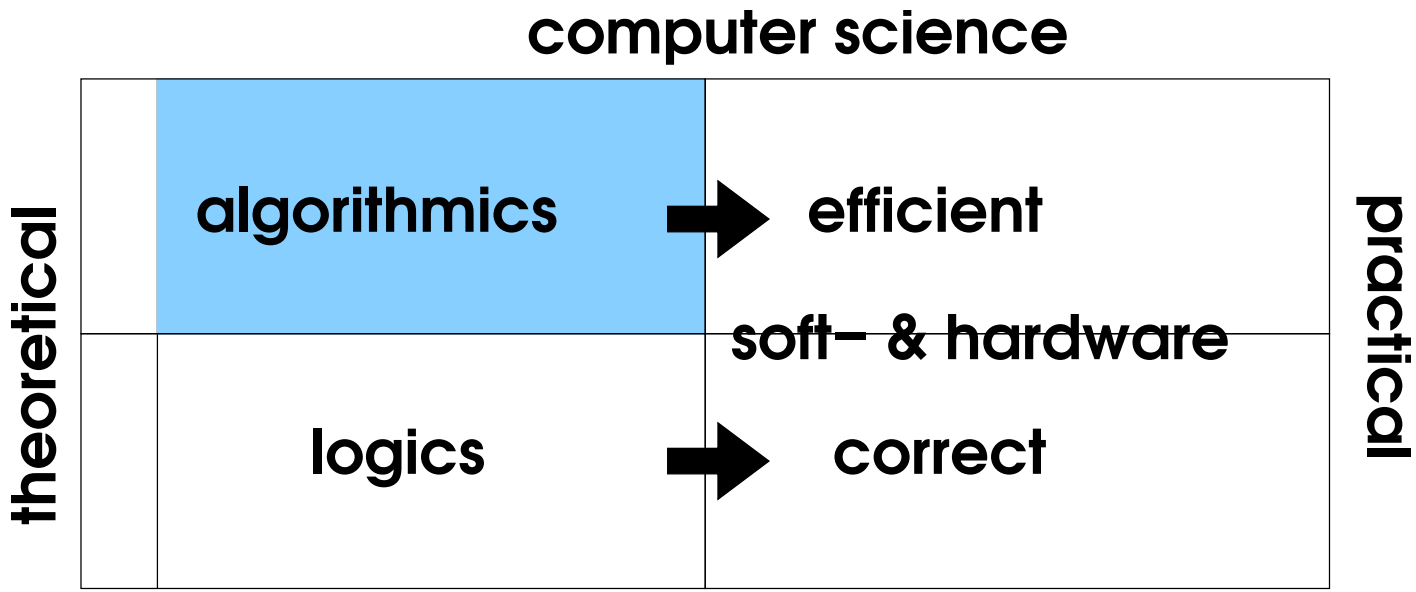
- in general
- simple examples, mostly external MSTs

More algorithm engineering in my group

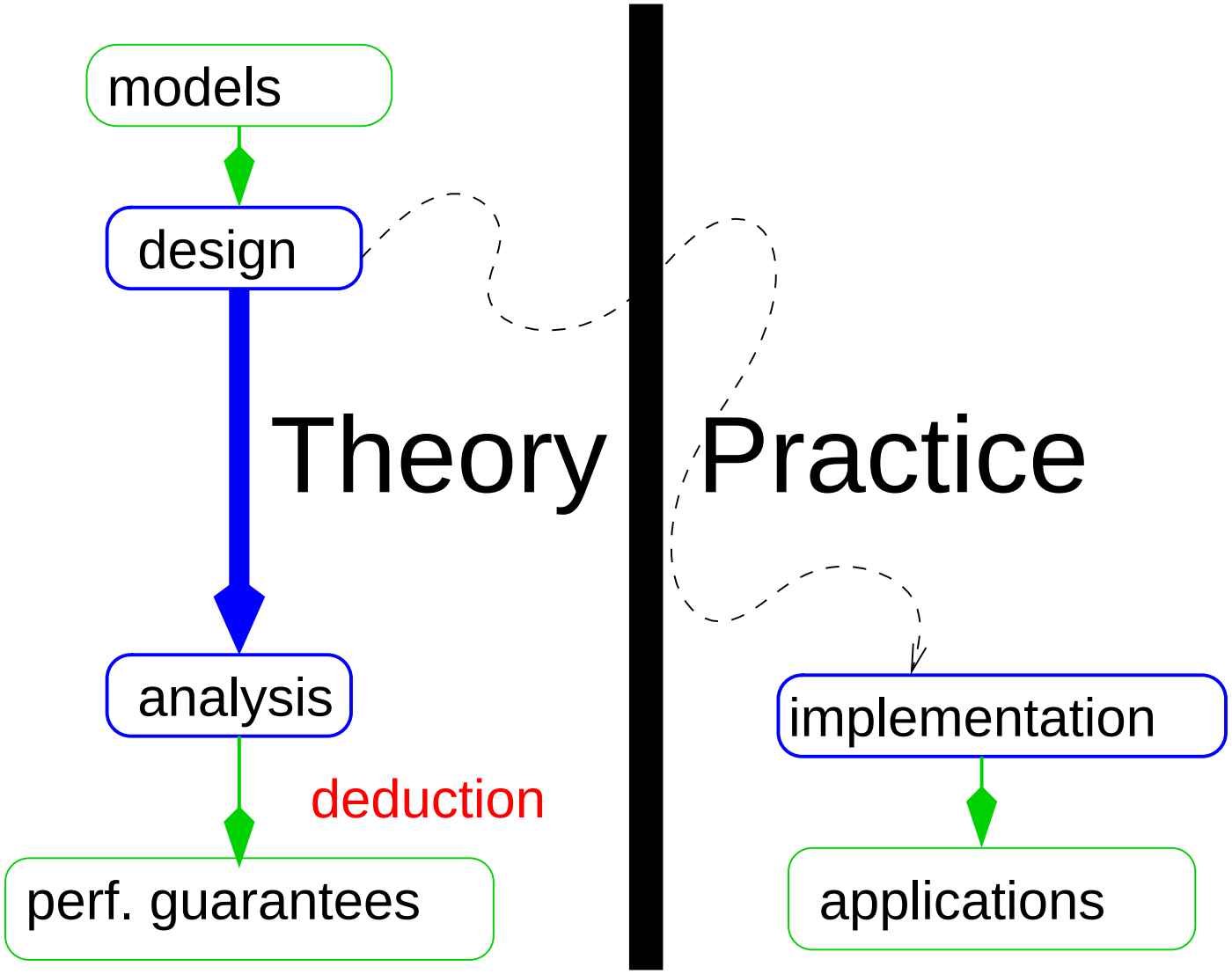
- The basic toolbox
- Various applications

Algorithmics

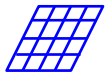
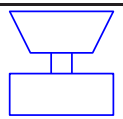
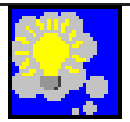
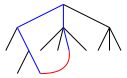
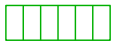
= the systematic design of efficient software and hardware



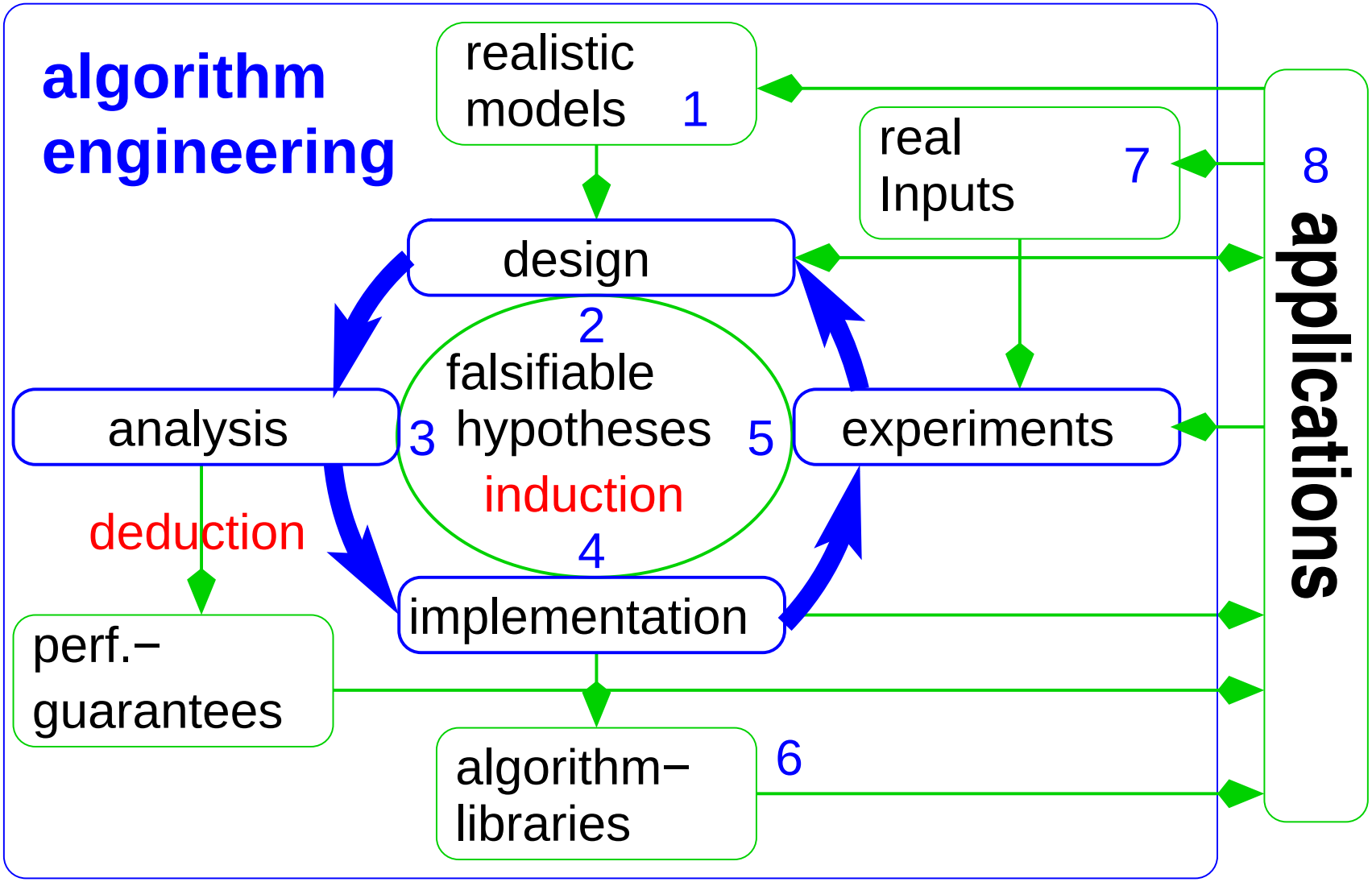
(Caricatured) Traditional View: Algorithm Theory



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



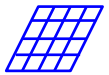
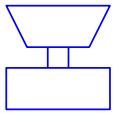
Goals

- 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

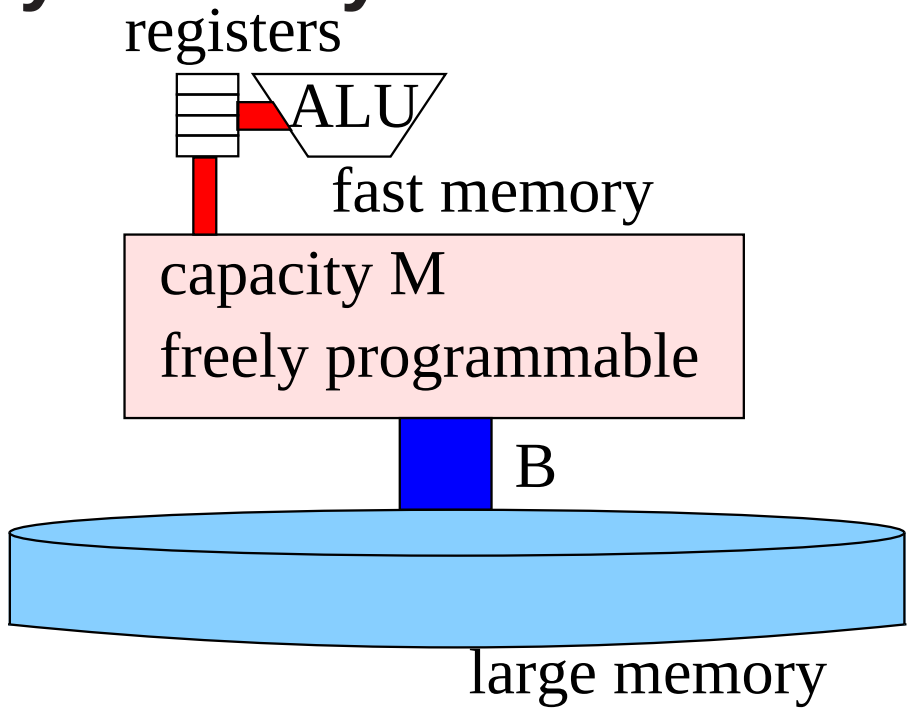
Theory	$\longleftrightarrow$	**Practice**
simple 	**appl. model**	 complex
simple 	**machine model**	 real

- 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

Example 0: Semiexternal Kruskal MST

Assumption: $n < M$ nodes

sort edges by increasing weight

set up union-find data structure for nodes

for each edge (u, v) by decreasing weight **do**

  if u and v are in different components **then**

    output (u, v) *//* in MST by cut property

    join components of u and v

  // Else not in MST due to cycle property

$\mathcal{O}(\text{sort}(m))$ I/Os

Observation: More cleverness only needed for huge, sparse graphs

Example 1

for i := n **downto** M **do**

pick a random node v

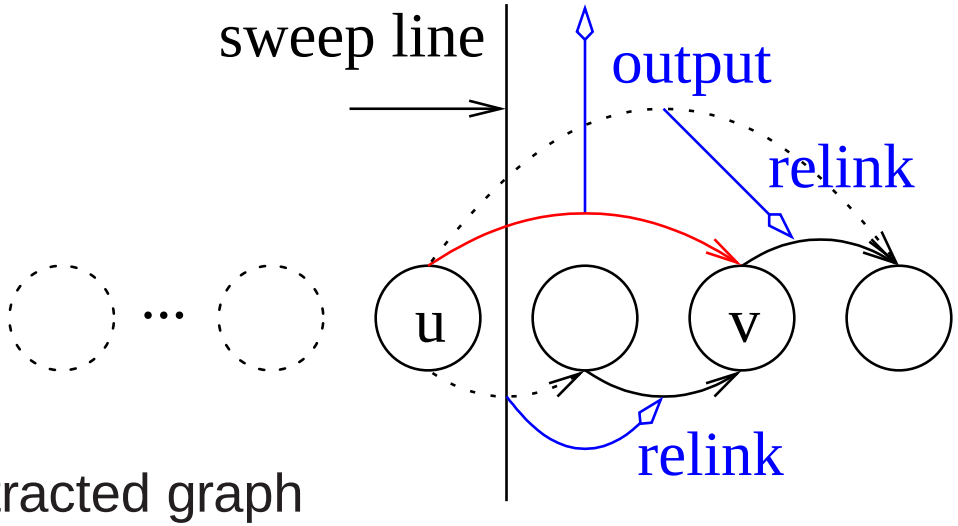
(u, v):= lightest edge out of v

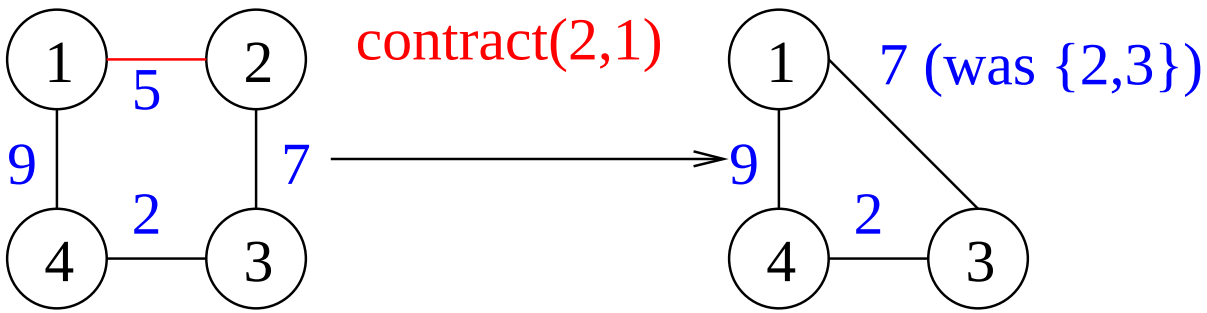
output (u, v)

contract (u, v)

run semiexternal Kruskal on the contracted graph

Sibeyn's external MST algorithm





[with: Roman Dementiev, Dominik Schultes, and Jop Sibeyn]

2: Design

Sibeyn's Algorithm

- ☐ simplicity — ≈ 20 lines
- ☐ reuse — using ext. PQ, sorter,. . . from STXXL
- ☐ constant factors — $\geq 3\times$ better than Boruvka based
- ☐ exploit easy instances — asymptotically opt. for planar graphs

3: Analysis

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

Example

Sibeyn's external MST algorithm

for i := n **downto** M **do**

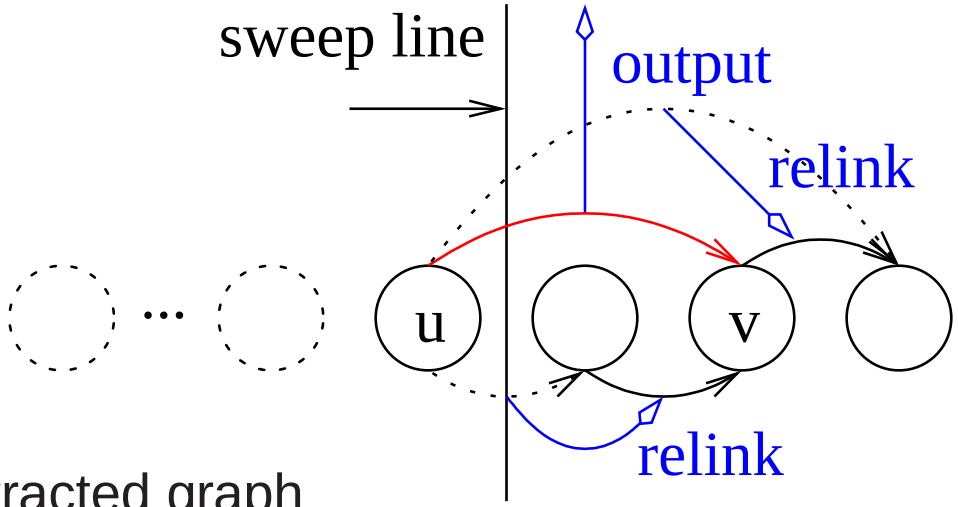
pick a random node v

(u, v):= lightest edge out of v

output (u, v)

contract (u, v)

run semiexternal Kruskal on the contracted graph



$$\mathrm{E}[\text{degree}(v)] \leq 2m/i$$

$$\sum_{M<i\leq n} \frac{2m}{i} = 2m \left(\sum_{0<i\leq n} \frac{1}{i} - \sum_{0<i\leq M} \frac{1}{i} \right) \approx 2m(\ln n - \ln M) = 2m \ln \frac{n}{M}$$

expected edge reling operations.

3: Analysis

Sibeyn's Algorithm

- Constant factors matter — here: beat asympt. factors!
- Beyond worst case analysis — graphs sparse under edge contr.
- Practical algorithms might be difficult to analyze

  (randomization, meta heuristics,. . .)

  Open problem: variant for connected components $\mathcal{O}\left(m \log\log \frac{n}{M}\right)$?

4: Implementation

sanity check for algorithms !

Challenges

Semantic gaps:

Abstract algorithm

$\longleftrightarrow$

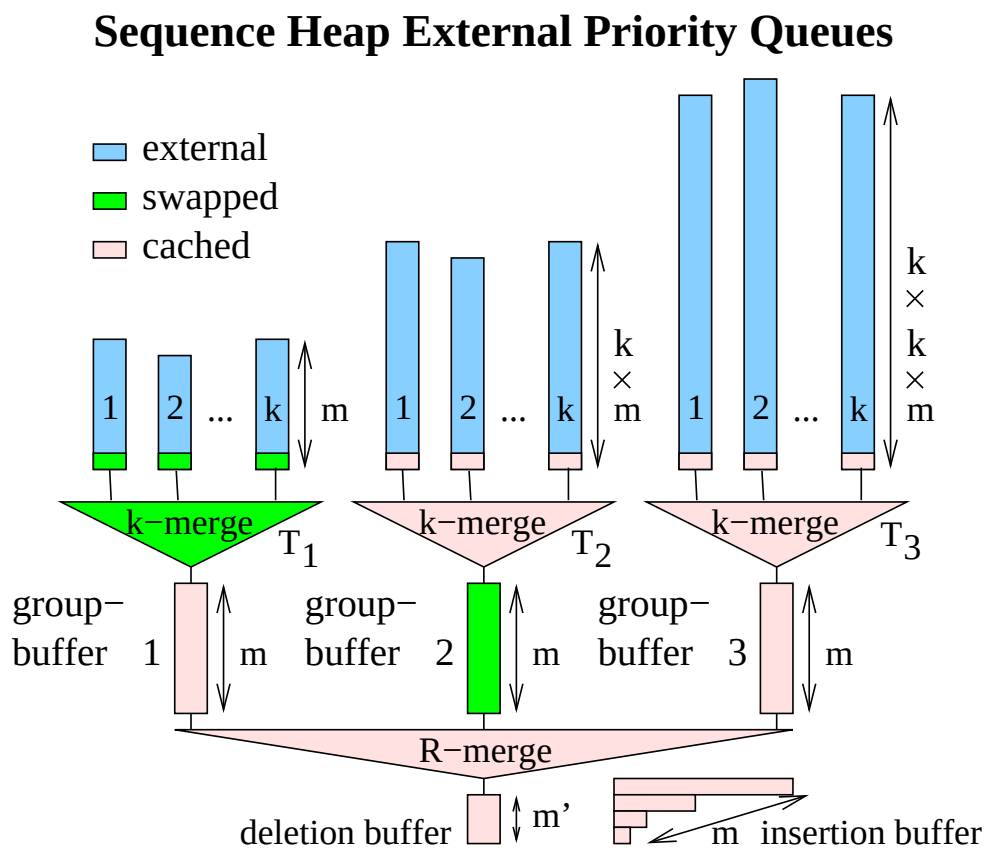
C++. . .

$\longleftrightarrow$

hardware

Small constant factors:

compare highly tuned competitors,



Example: Sibeyn's Algorithm Using Priority Queues

π : random permutation

Q: priority queue *//* Order: max node, then min edge weight

foreach $(\{u,v\},c) \in E$ **do** Q.insert$((\{\pi(u),\pi(v)\},c,\{u,v\}))$

current := $n+1$

loop

$(\{u,v\},c,\{u_0,v_0\})$:= Q.deleteMin$()$

if current$\neq \max\{u,v\}$**then**

if current$= M+1$ **then return**

output $\{u_0,v_0\},c$

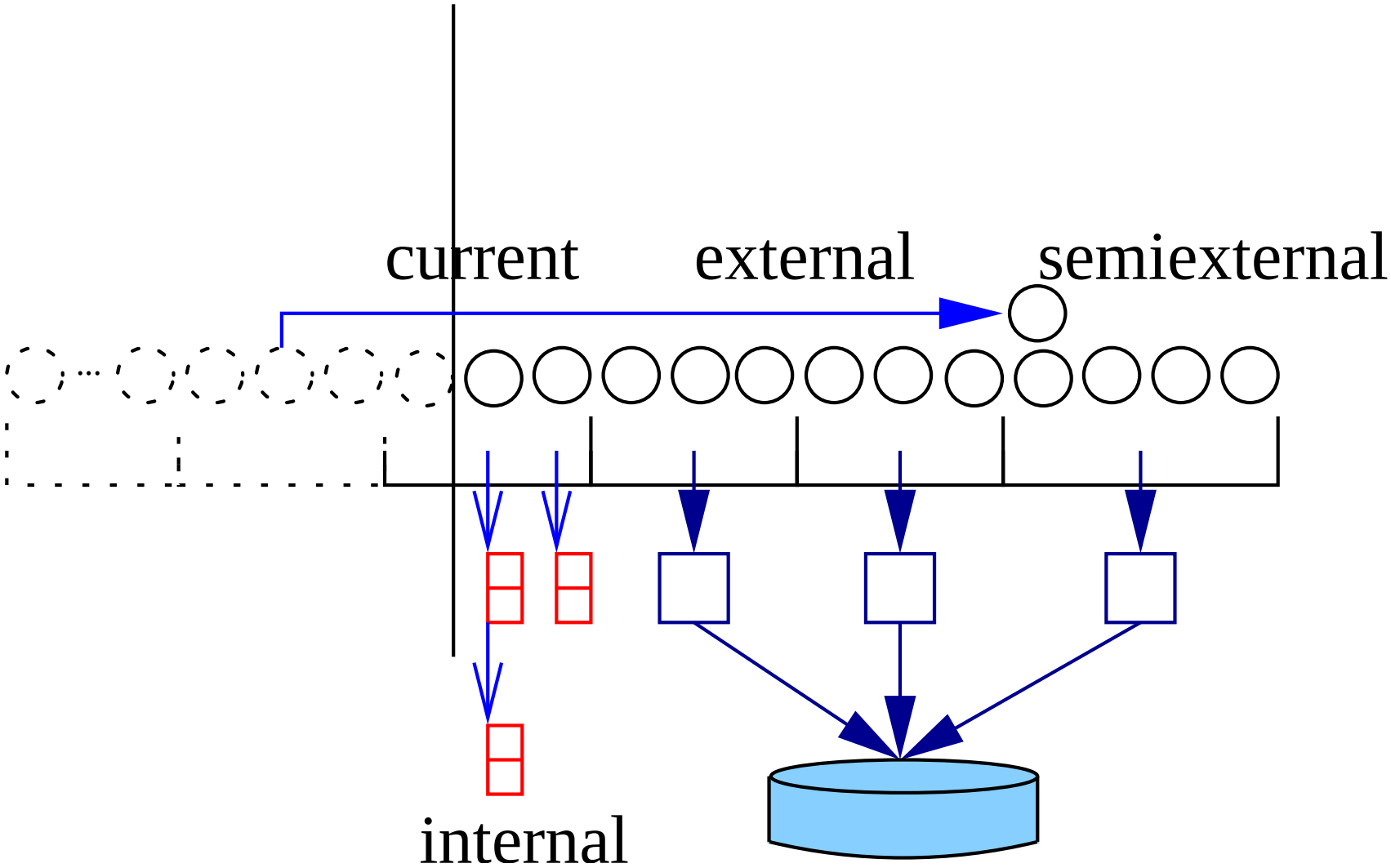
current := $\max\{u,v\}$

connect := $\min\{u,v\}$

else Q.insert$((\{\min\{u,v\}$, connect$\},c,\{u_0,v_0\}))$

Problem: Compute bound

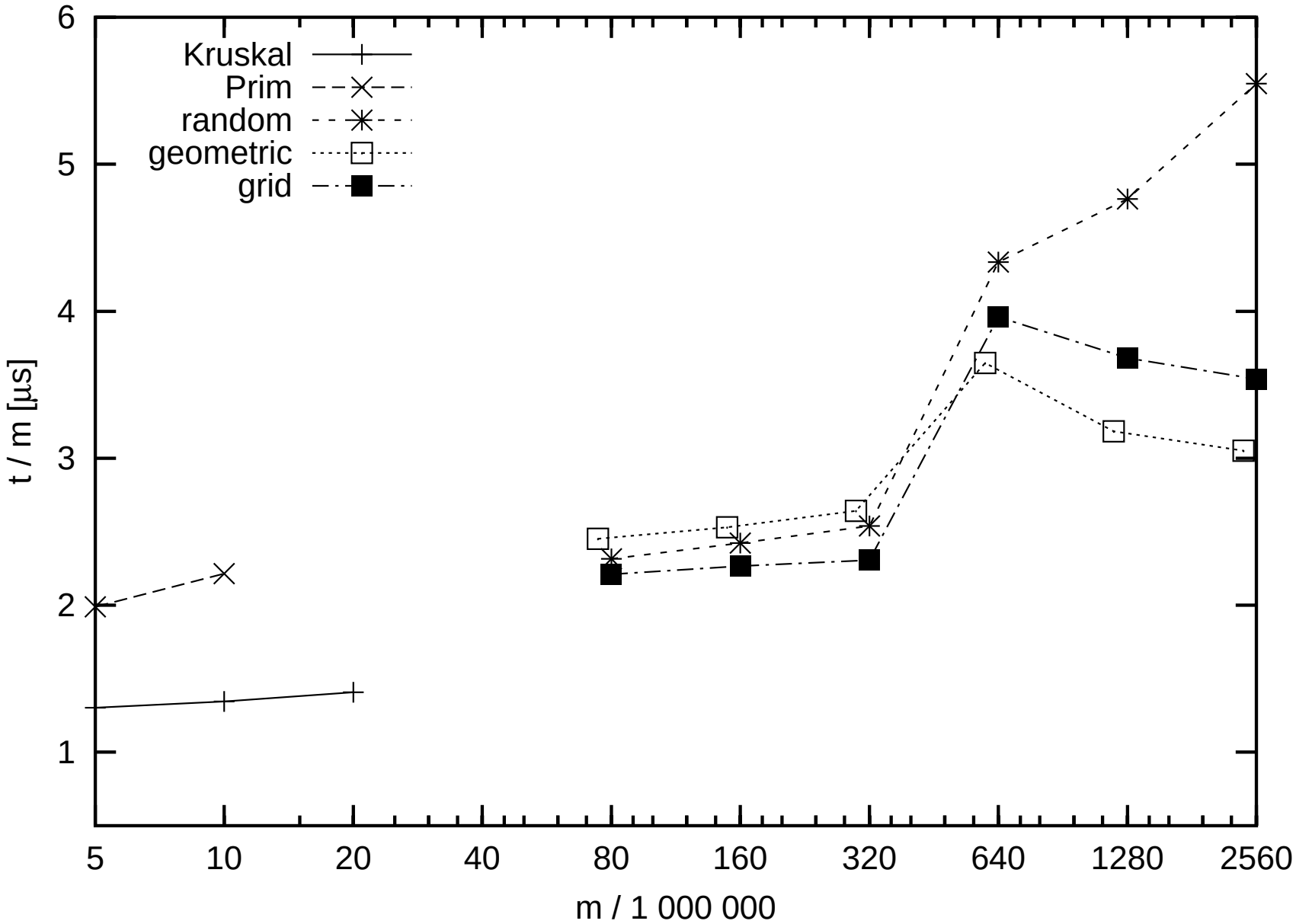
Bucket PQs for Sibeyn's Algorithm



5: Experiments

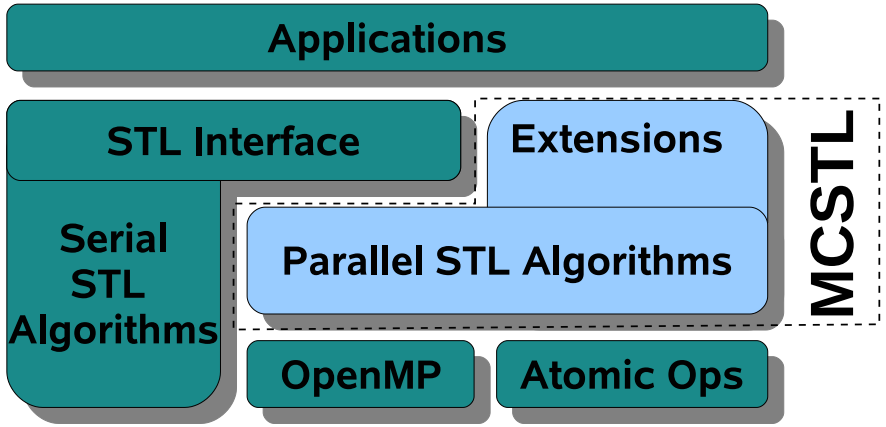
- sometimes a good surrogate for analysis
- too much rather than too little output data
- reproducibility (10 years!)
- software engineering

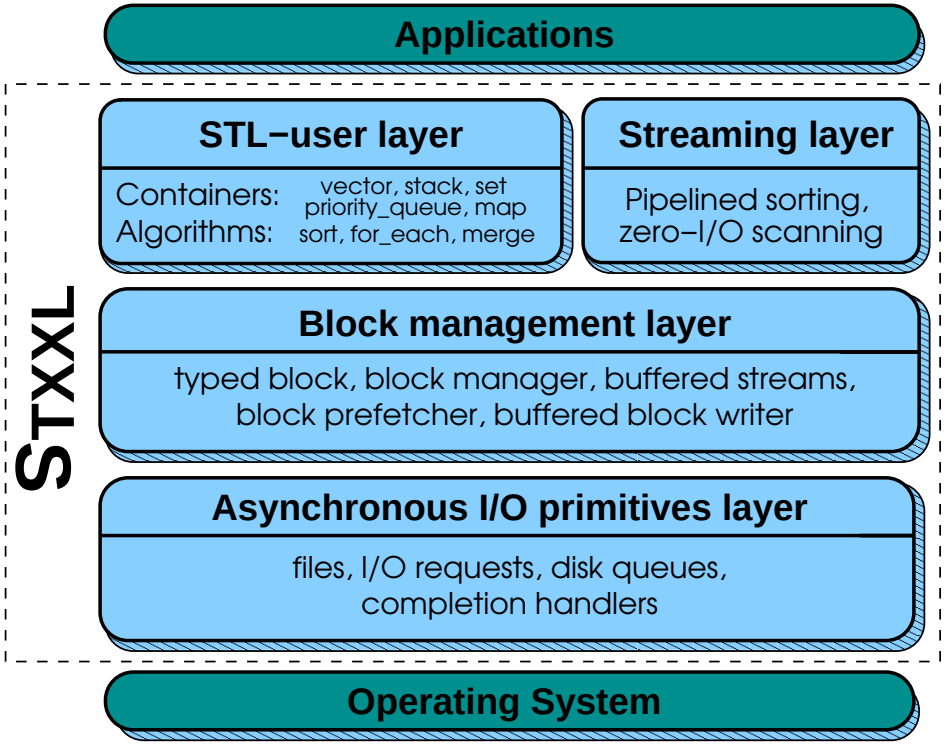
Example, Sibeyn's algorithm, $m \approx 2n$



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

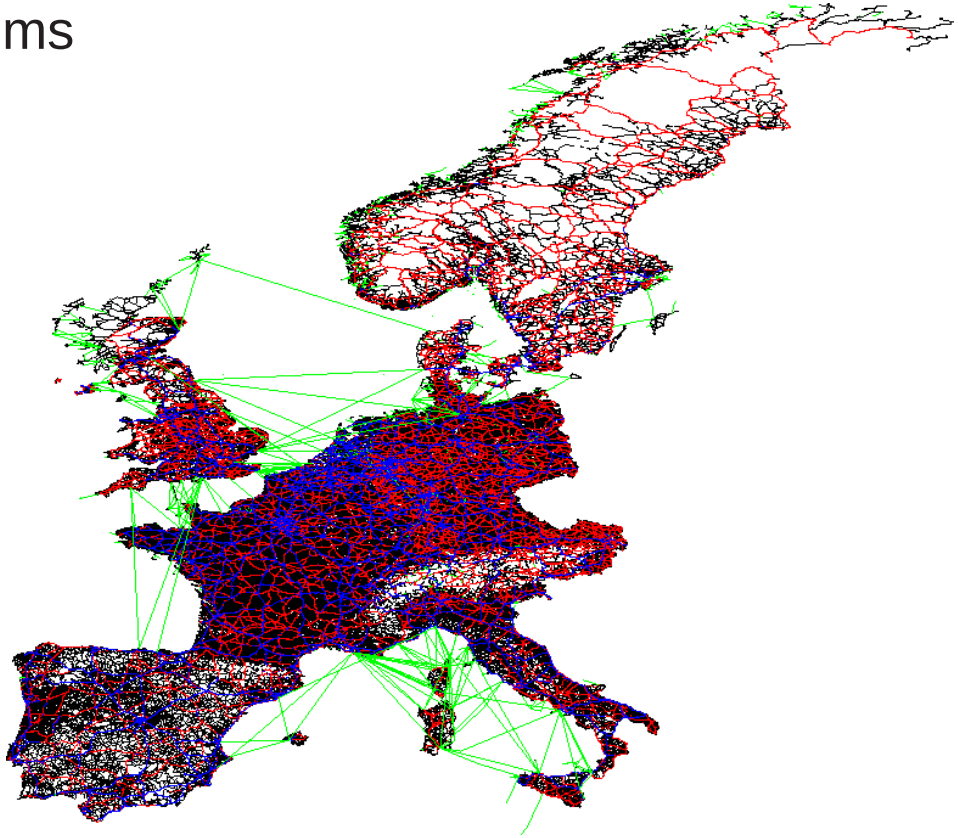




7: Problem Instances

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...)

Example: Huge Real World Instances for MST?

- Point clouds for surface reconstruction ?
- Aerial images for image segmentation ?

Is the pre/post processing the real issue ?

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,. . .

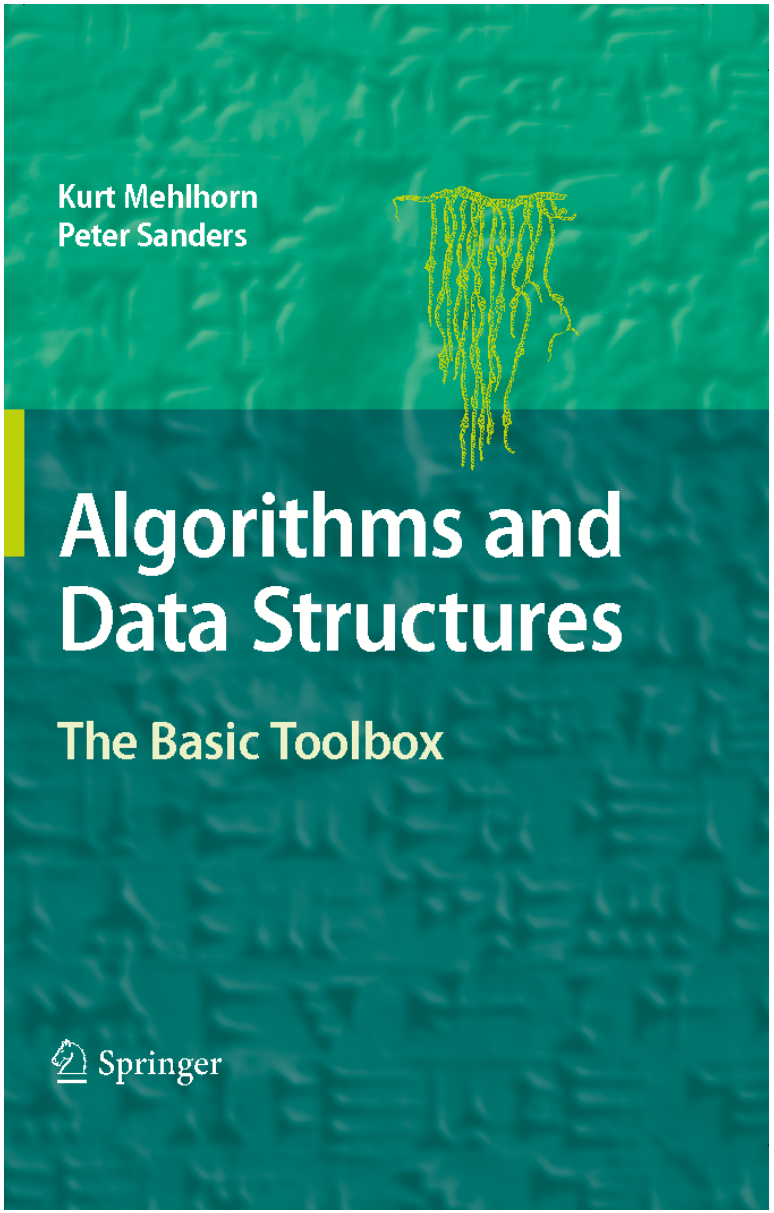
Conclusion:

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



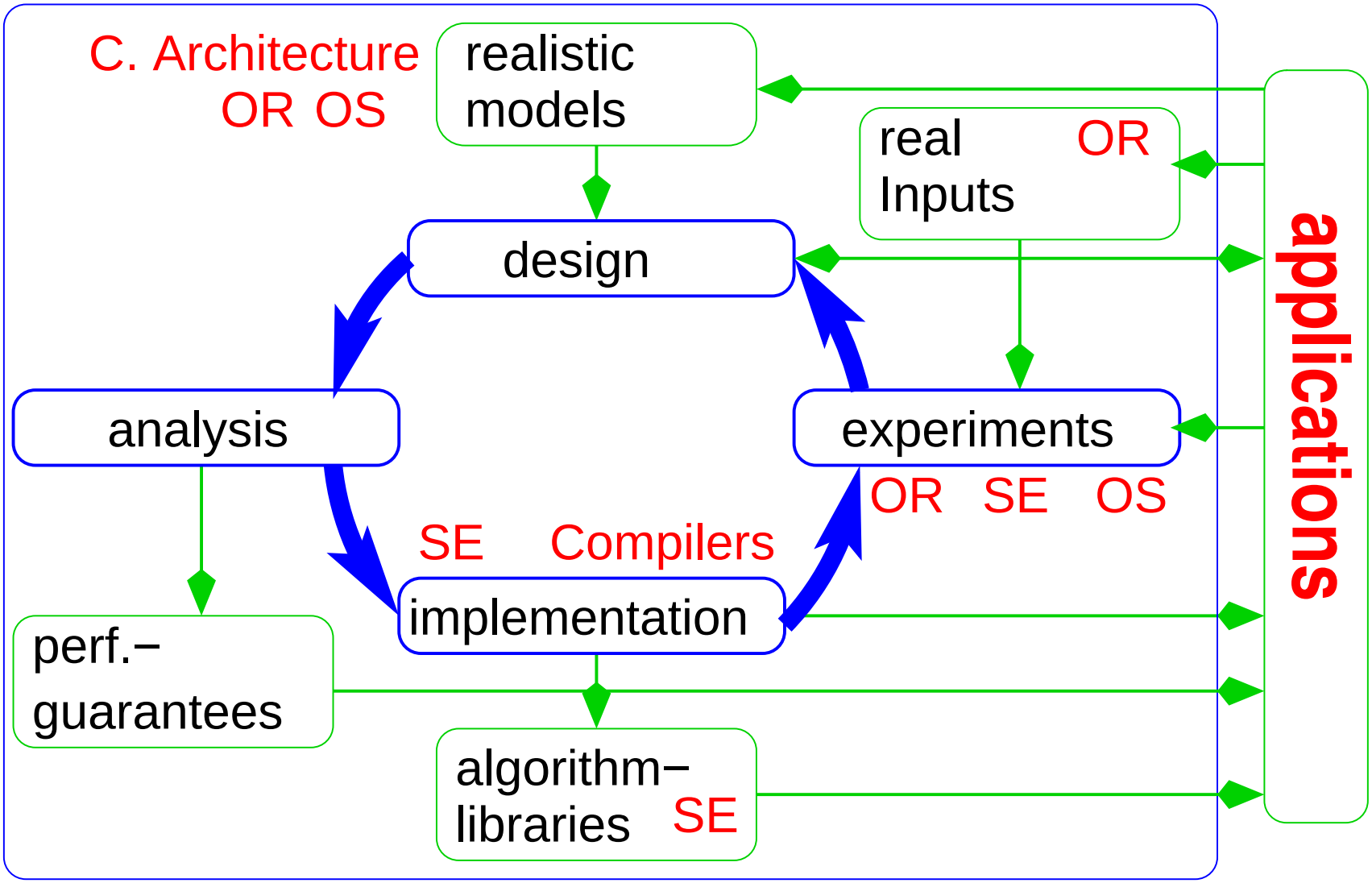
Application Areas

- combinatorial optimization
- parallel computing
  - C++ Standard Library for Multicore
  - MPI communication primitives
  - load balancing
- external memory computations
- scalable (virtual) storage servers
- information retrieval $\longrightarrow$ Cooperation with SAP TREX group

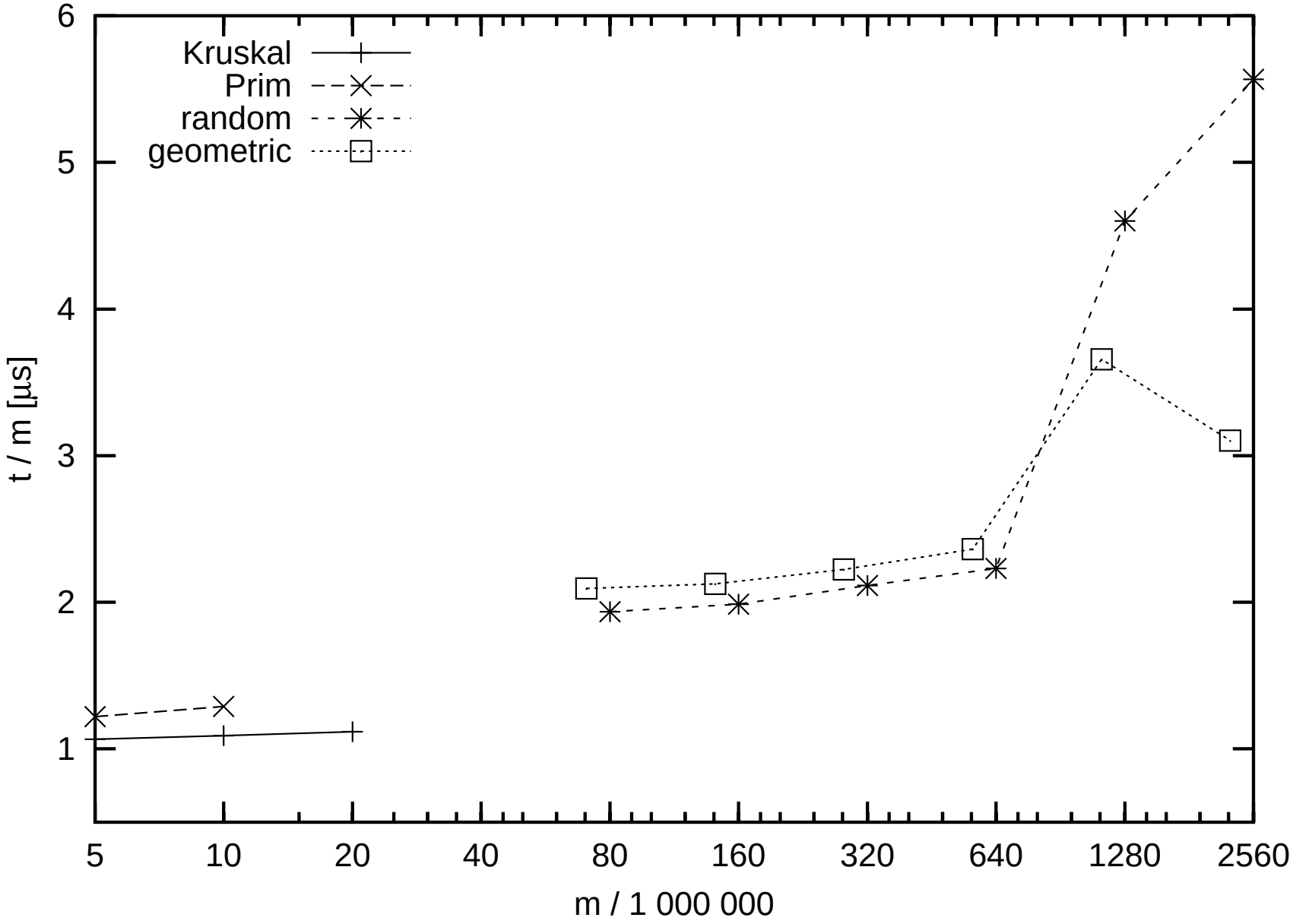
here

worked on

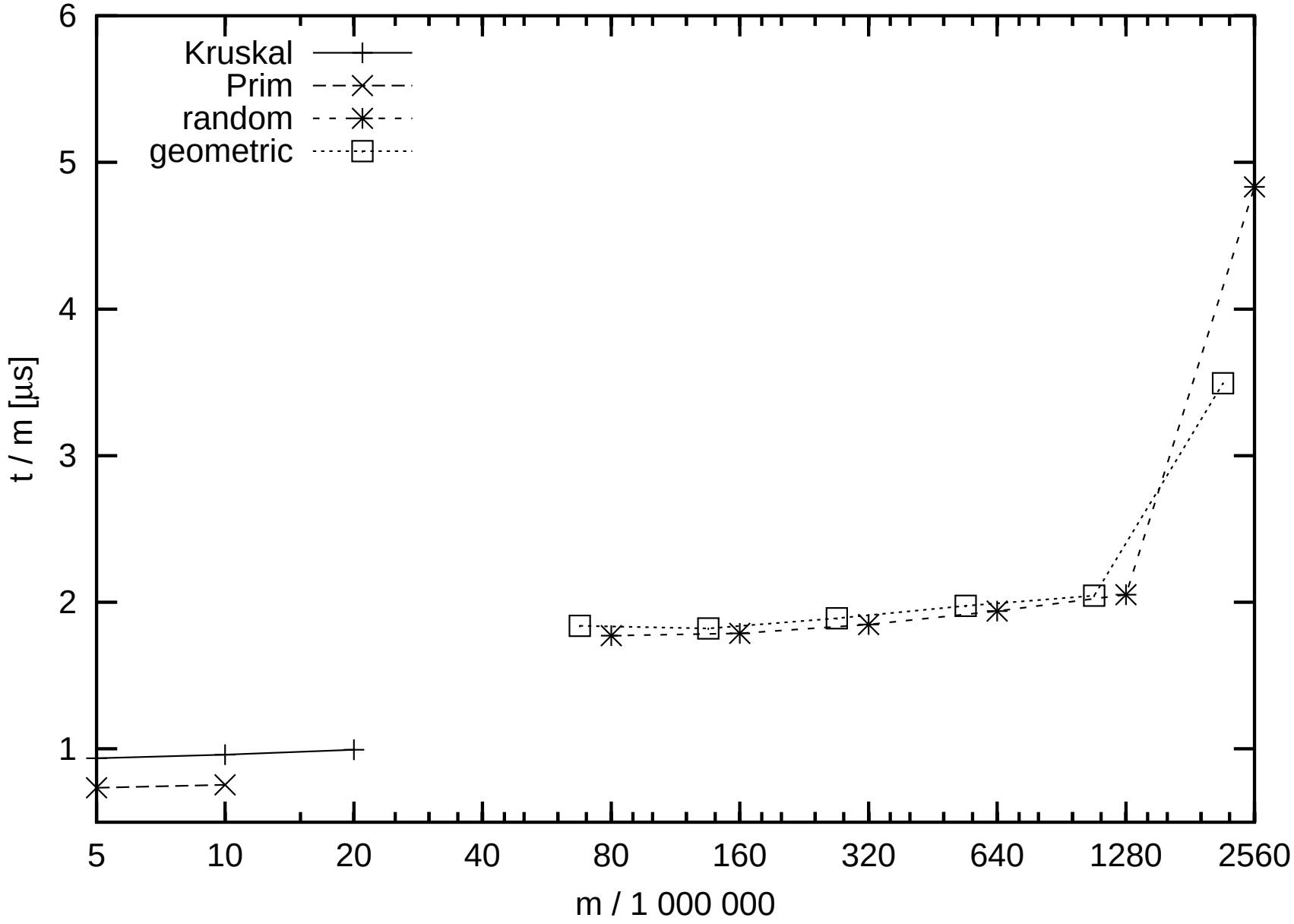
Interactions with other (Sub)disciplines



$m \approx 4n$


Kruskal
Prim
random
geometric
t / m [µs]
6
5
4
3
2
1
5
10
20
40
80
160
320
640
1280
2560
m / 1 000 000

$m \approx 8n$


Kruskal
Prim
random
geometric
t / m [µs]
1
2
3
4
5
6
5
10
20
40
80
160
320
640
1280
2560
m / 1 000 000