Algorithm Engineering

An Attempt at a Definition

Using Parallel (External) Sorting as an Example

Peter Sanders

Karlsruhe Institute of Technology

Overview

- a general definition

  [with Kurt Mehlhorn, Rolf Möhring, Petra Mutzel, Dorothea Wagner]

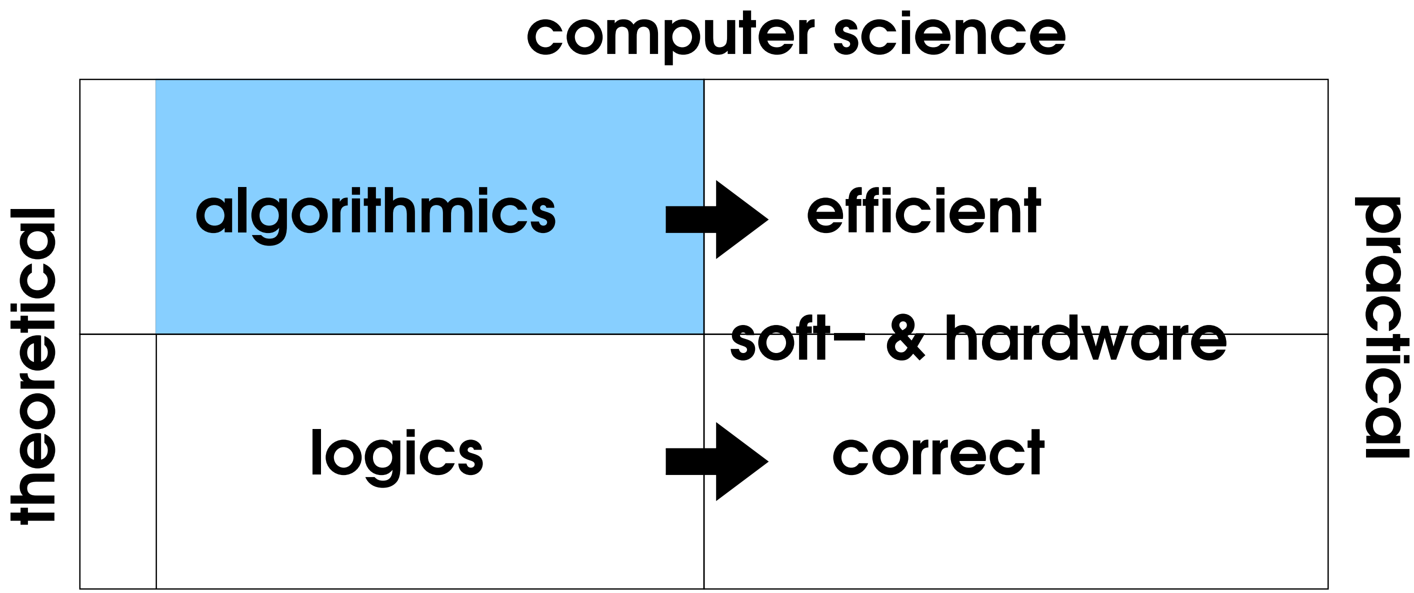
- main challenges

- Parallel (external) sorting as an example

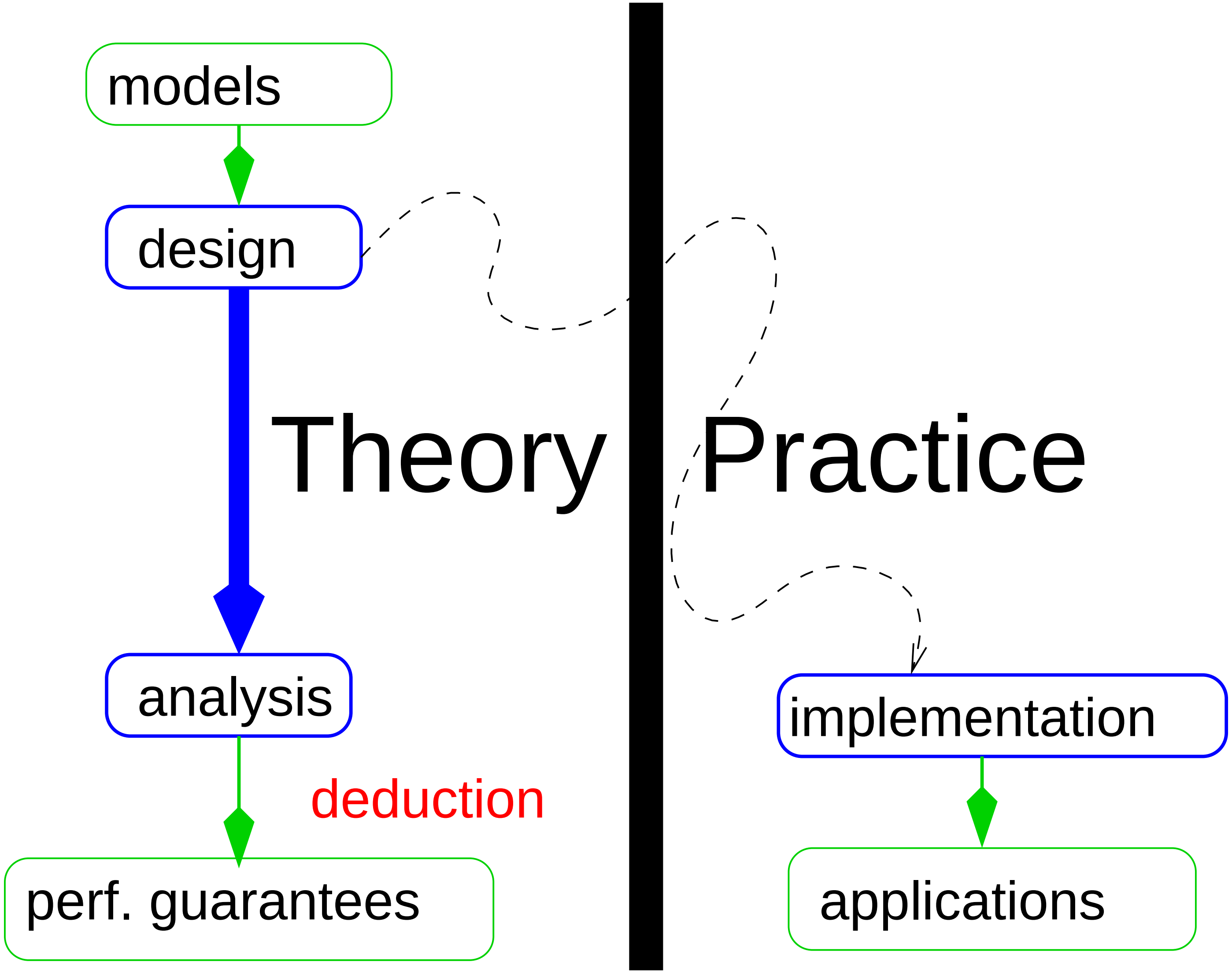
  [with Andreas Beckmann, Roman Dementiev, David Hutchinson, Kanela Kaligosi, Nicolai Leischner, Ulrich Meyer, Vitaly Osipov, Mirko Rahn, Johannes Singler, Jeff Vitter, Sebastian Winkel]

Algorithmics

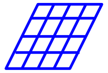
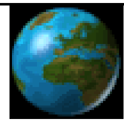
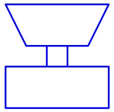
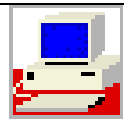
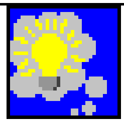
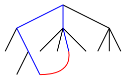
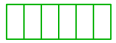
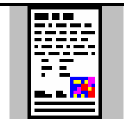
$=$ the systematic design of efficient software and hardware



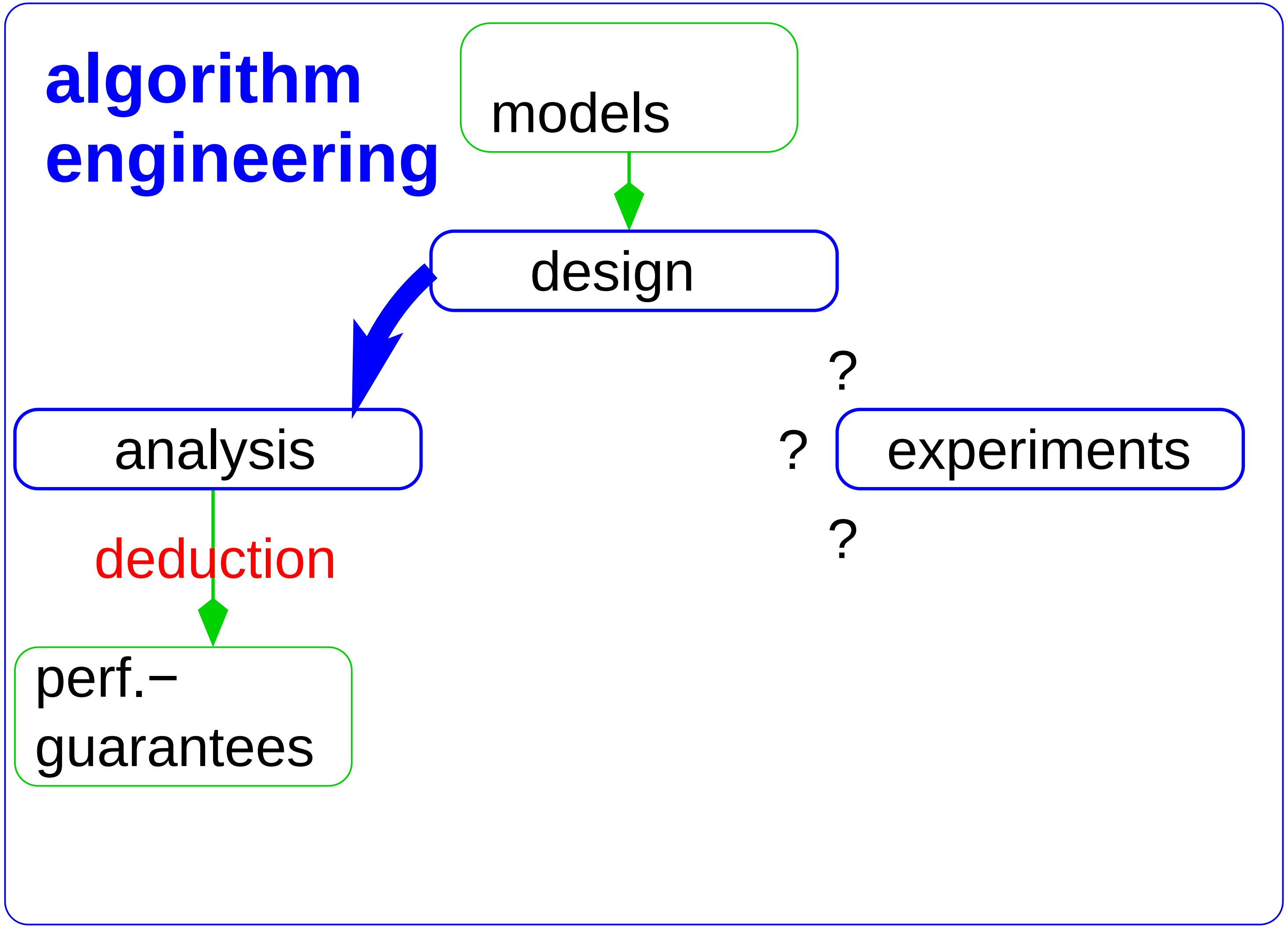
(Caricatured) Traditional View: Algorithm Theory



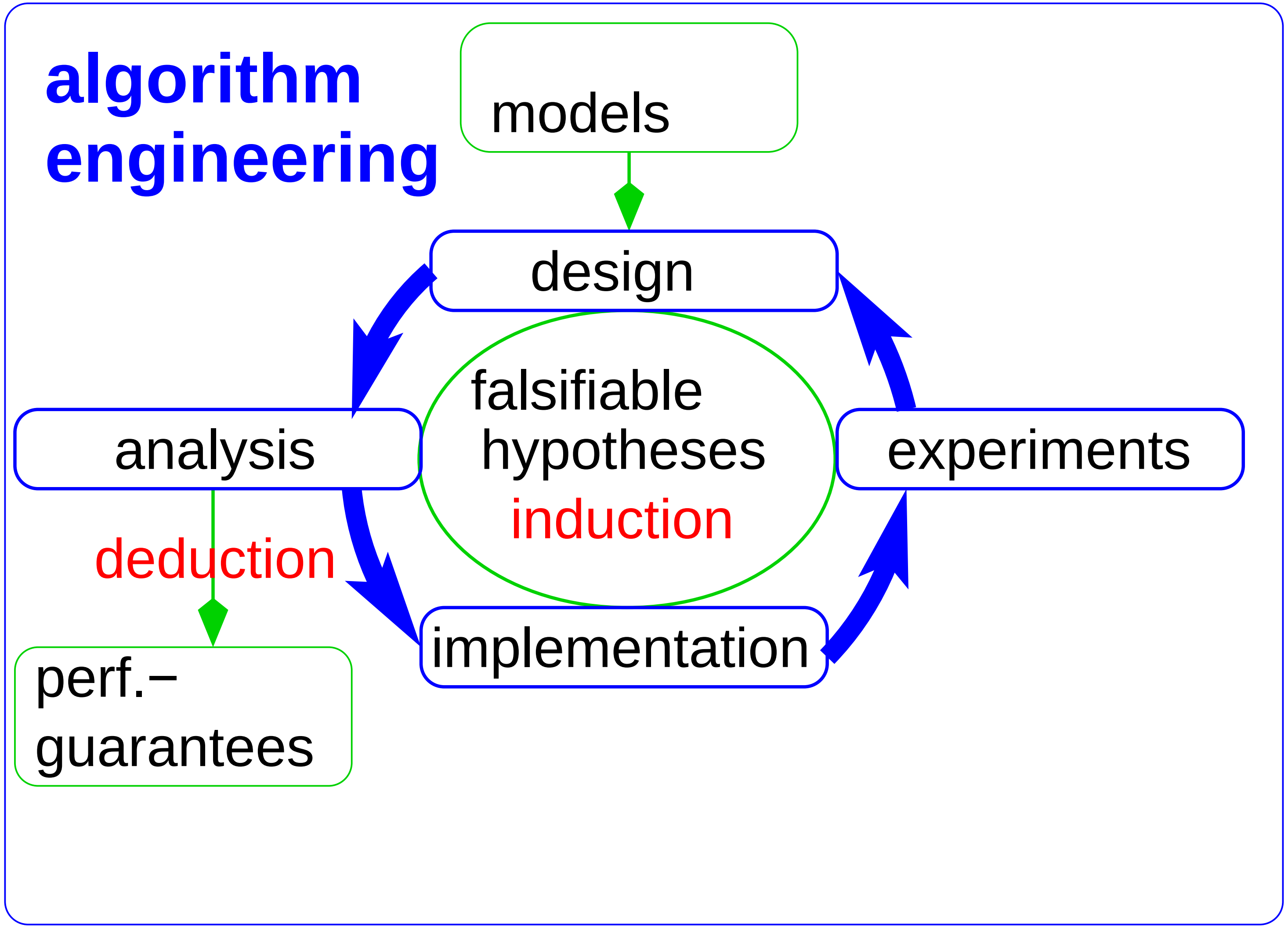
Gaps Between Theory & Practice

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

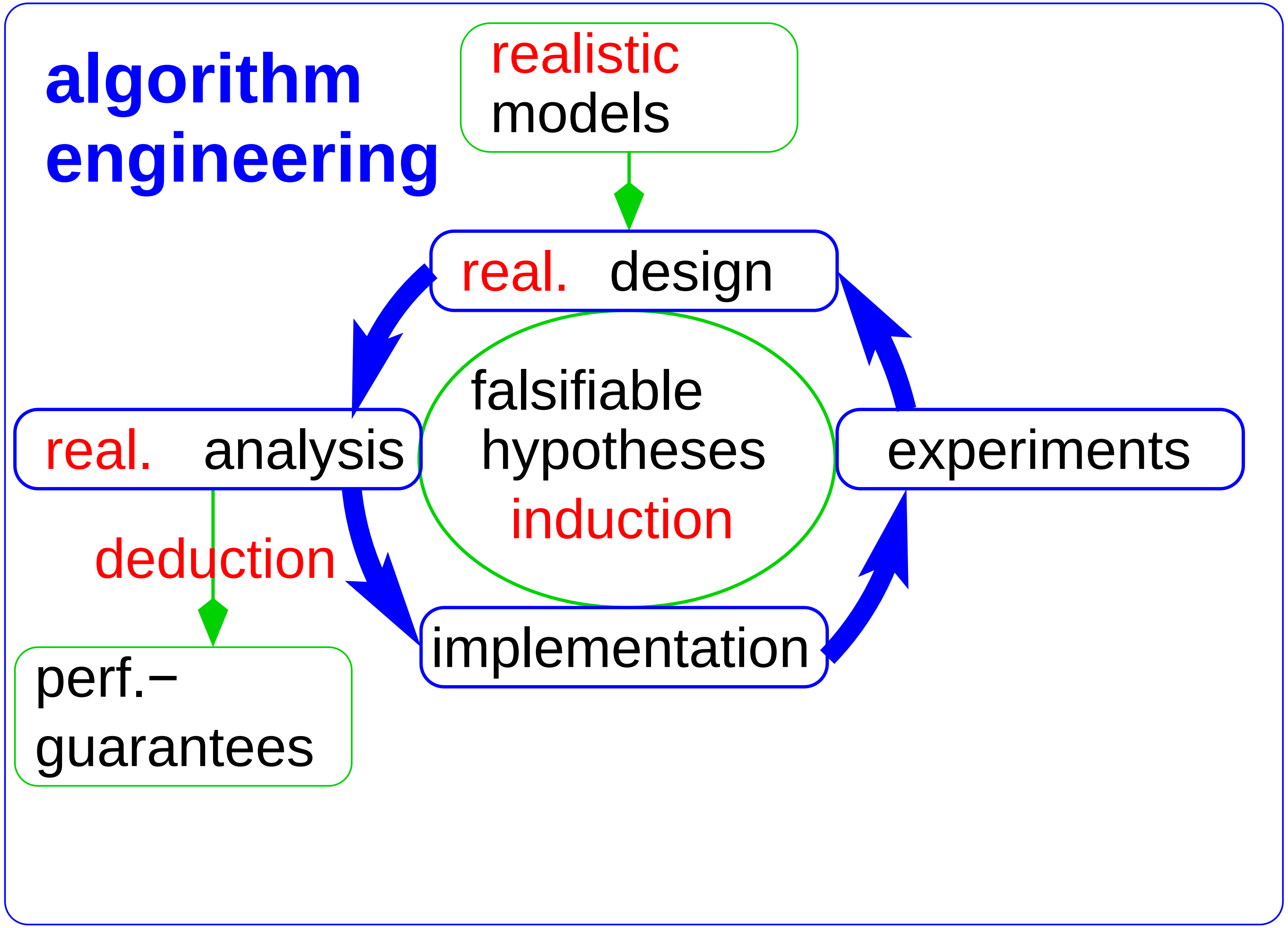
Algorithmics as Algorithm Engineering



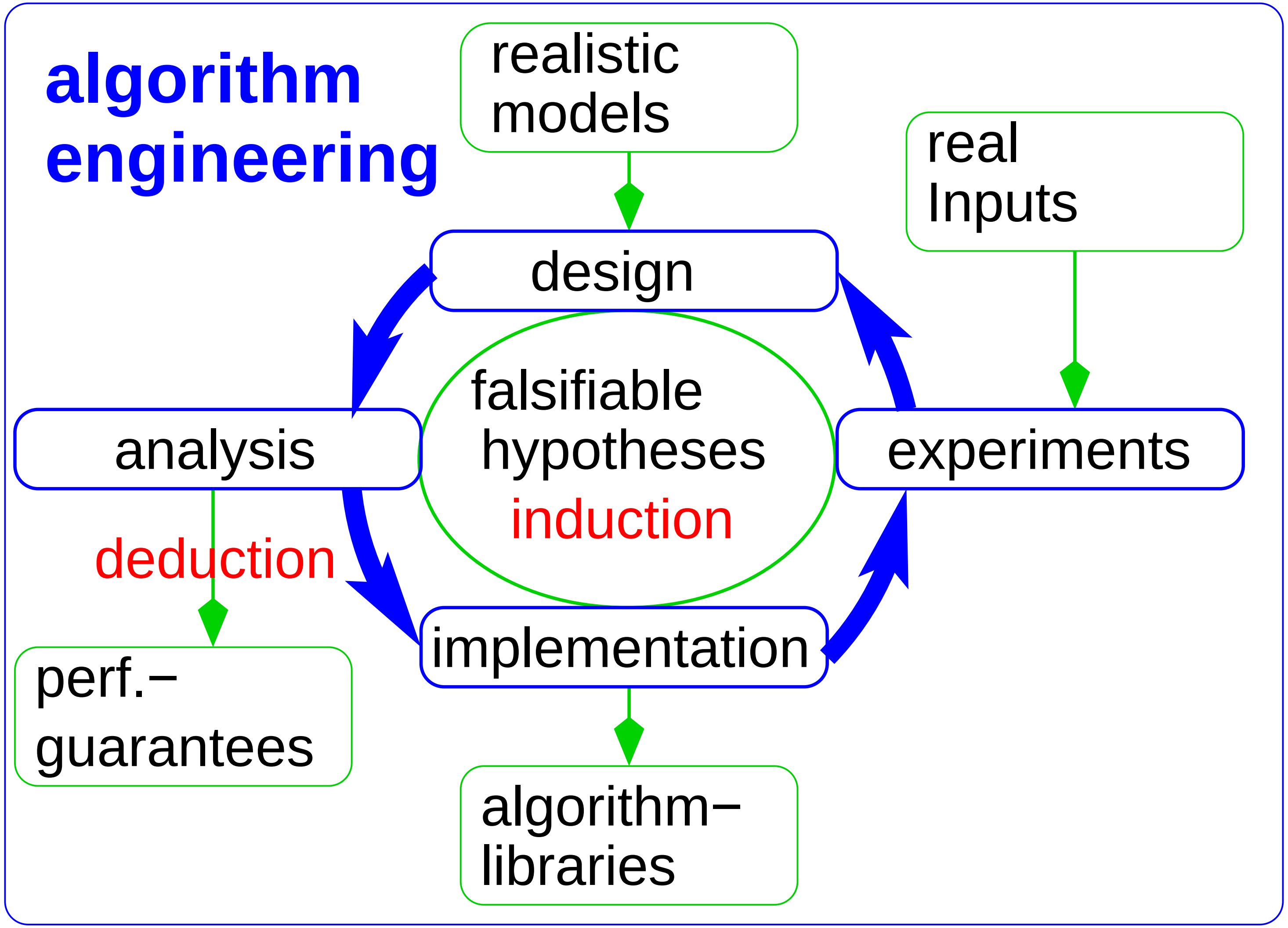
Algorithmics as Algorithm Engineering



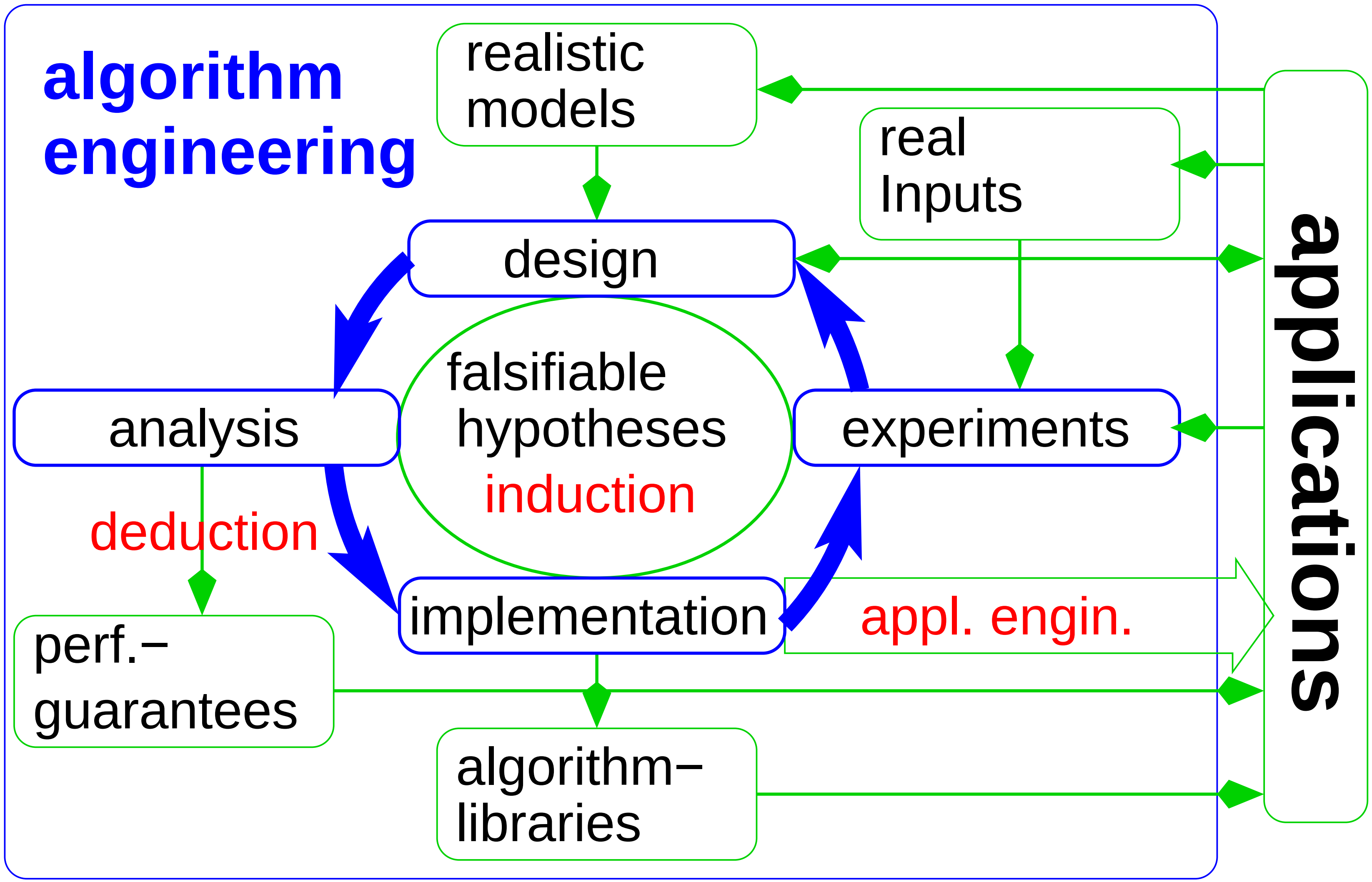
Algorithmics as Algorithm Engineering



Algorithmics as Algorithm Engineering



Algorithmics as Algorithm Engineering

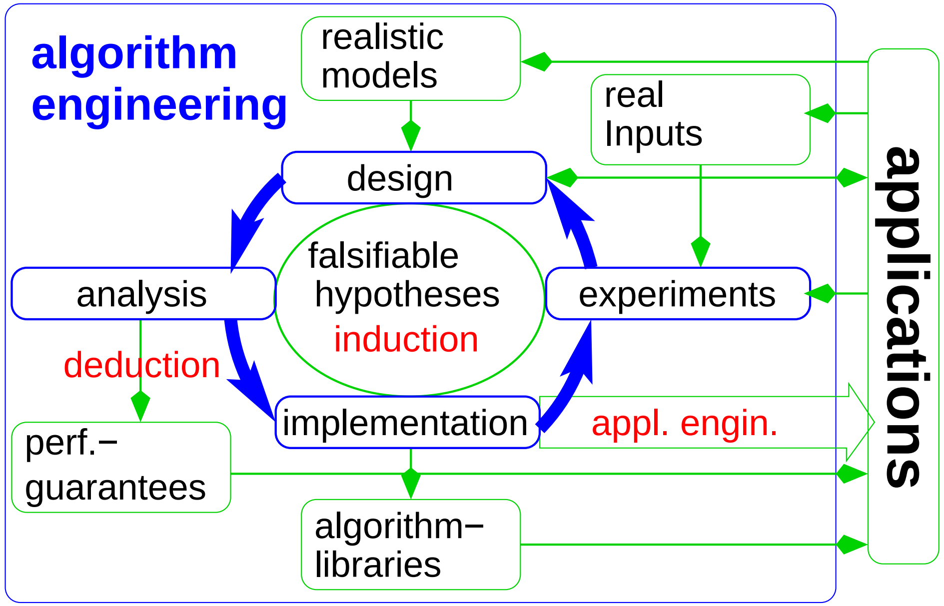


Goals

- bridge gaps between theory and practice
- accelerate transfer of algorithmic results into applications
- keep the advantages of theoretical treatment:

  generality of solutions and

  reliability, predictability from performance guarantees



Bits of History

1843– Algorithms in theory and practice

1950s,1960s Still infancy

1970s,1980s Paper and pencil algorithm theory.

Exceptions exist, e.g., [J. Bentley, D. Johnson]

1986 Term used by [T. Beth],

lecture “Algorithmentechnik” in Karlsruhe.

1988– Library of Efficient Data Types and Algorithms

(LEDA) [K. Mehlhorn, S. Näher]

1990– DIMACS Implementation Challenges [D. Johnson]

1997– Workshop on Algorithm Engineering

$\rightsquigarrow$ ESA applied track [G. Italiano]

1997 Term used in US policy paper [Aho, Johnson, Karp, et. al]

1998 Alex workshop in Italy $\rightsquigarrow$ ALENEX

Commercial Break [Bader, Sanders, Wagner]

10th DIMACS Implementation Challenge

Two related challenges:

- (Balanced) **Graph Partitioning** (cut minimization) and
- **Clustering** (modularity, others)

Variants welcome for the workshop

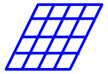
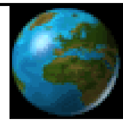
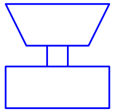
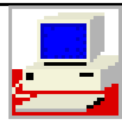
Atlanta February 13/14, 2012

June 1 2011: testbed creation

Oct 21 2011: paper deadline

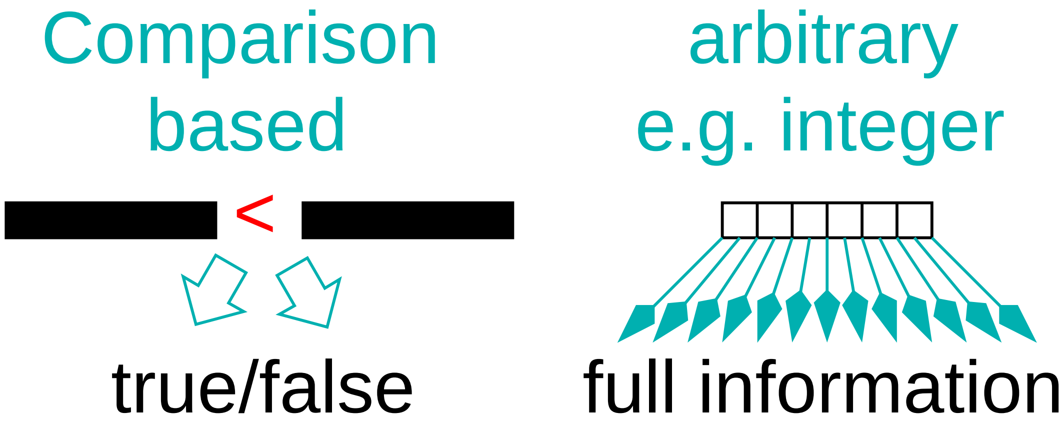
`http://www.cc.gatech.edu/dimacs10/`

Realistic Models – The Beauty and the Beast

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

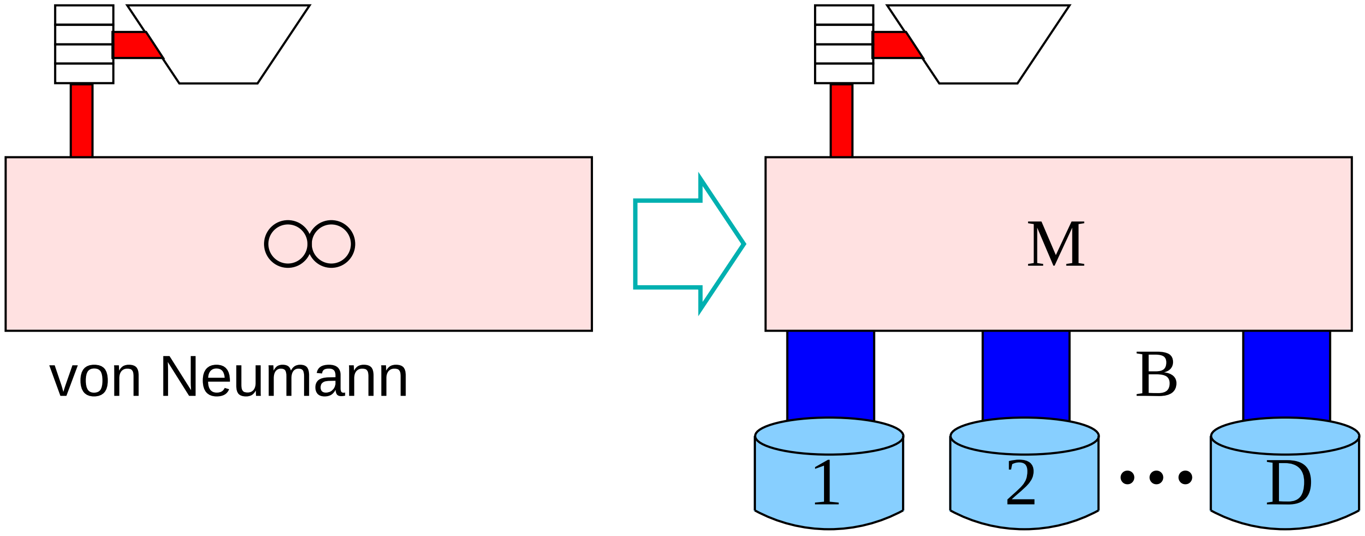
- Careful refinements
- Try to preserve (partial) analyzability / simple results

Sorting – Model



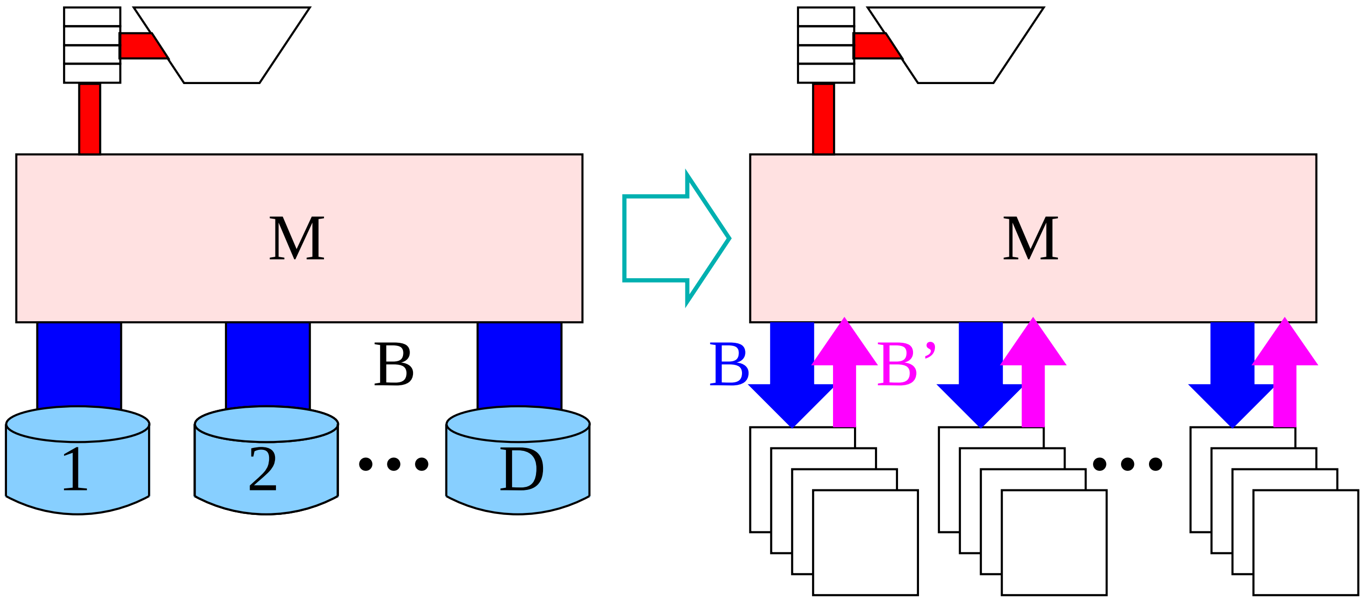
Advanced Machine Models

Parallel Disks

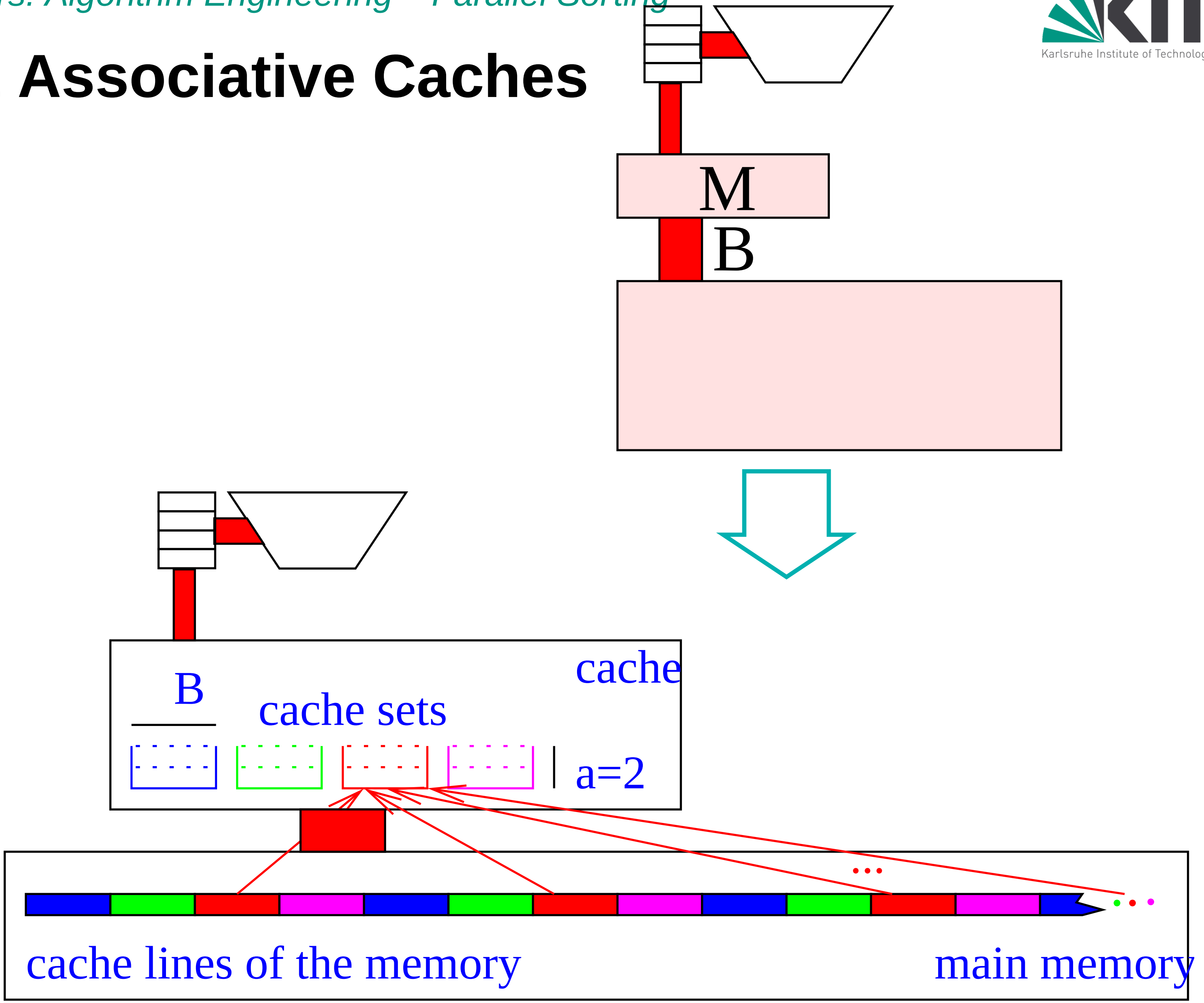


Advanced Machine Models

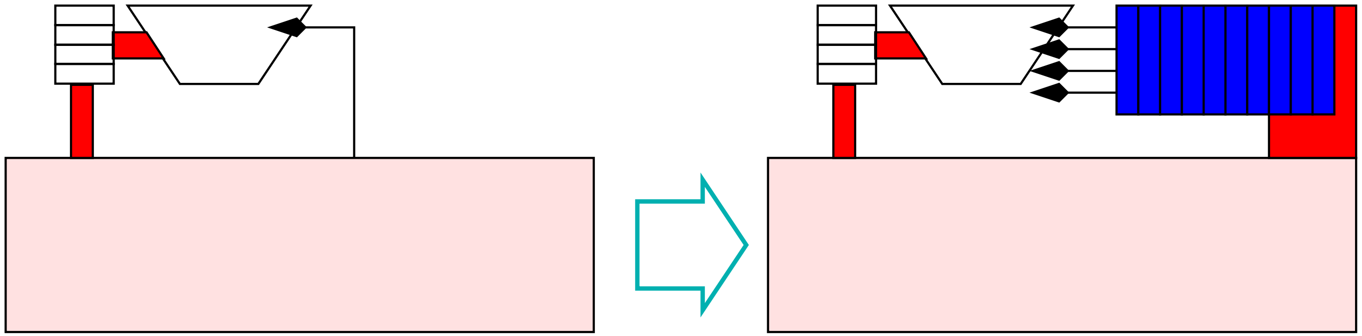
SSDs



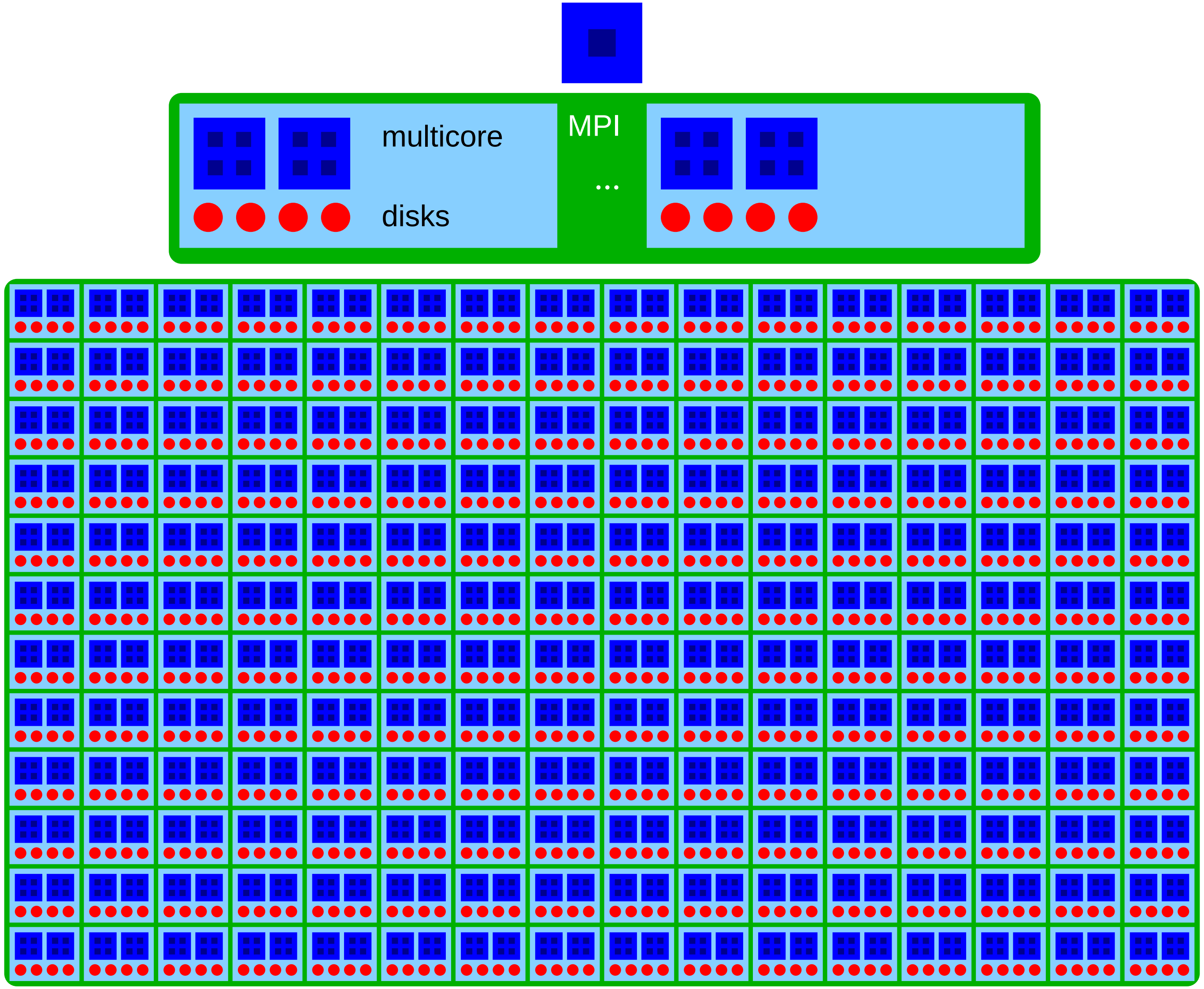
Set Associative Caches



Branch Prediction



Hierarchical Parallel External Memory



Our Cost Model for Parallel External Sorting

"sequential" aspects: Comparisons, cache faults, branch mispredictions, ILP

shared memory: remote cache accesses (not here: synchronizations,...)

disk: I/Os, overlapping, tune block size

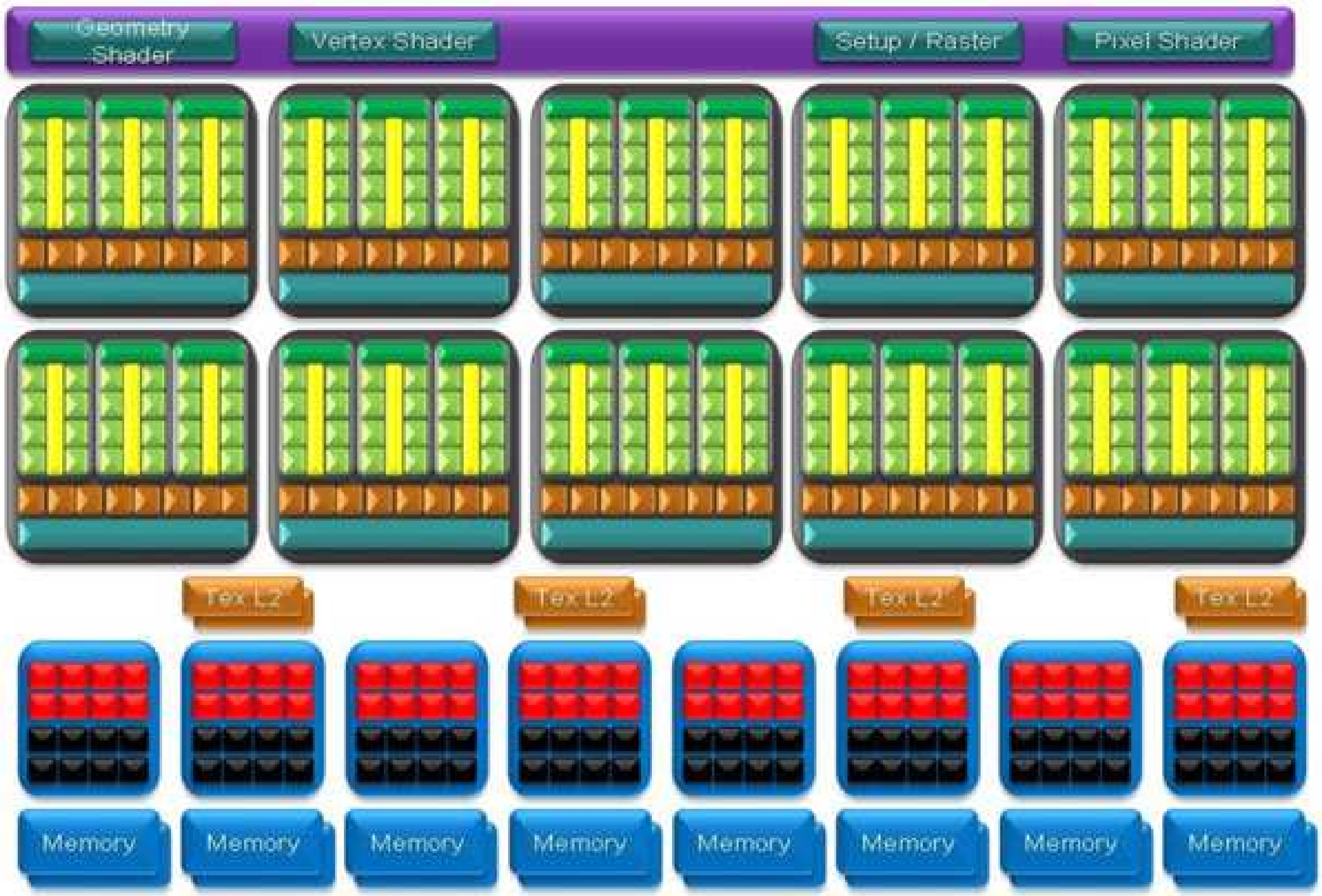
distributed memory: communication volume (with alltoallv) (not here latency, collectives)

overall: time, energy

partly plug-and-play of previous results.

Mix of formal and informal consideration

Graphics Processing Units



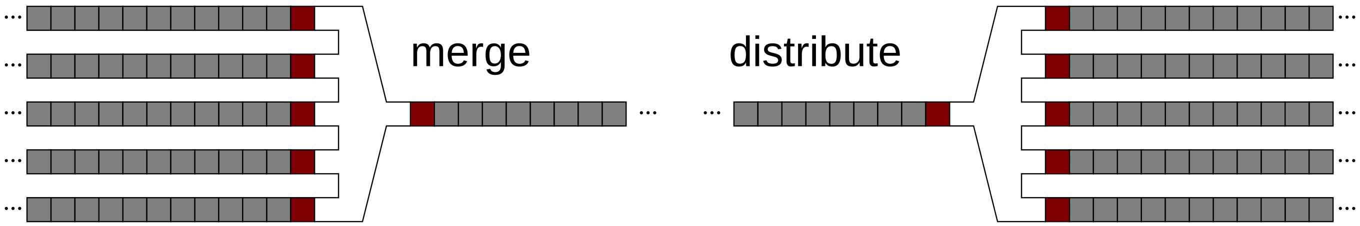
not here

Design

of algorithms that work well in practice

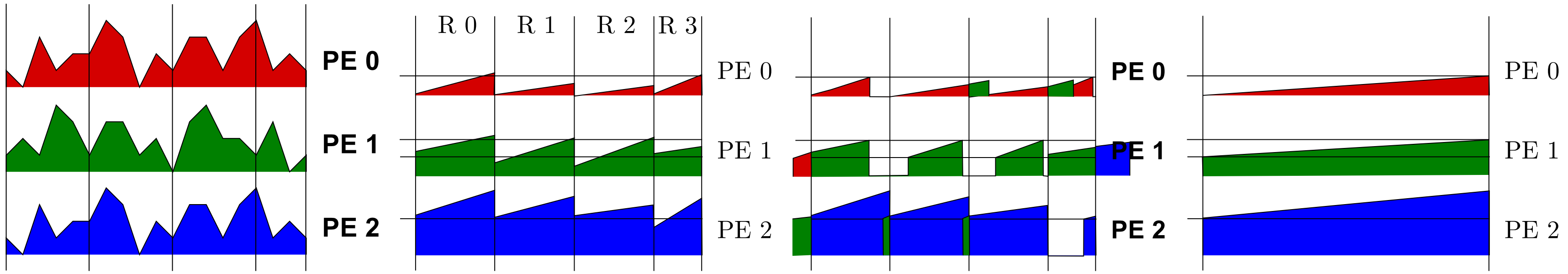
- simplicity
- reuse
- constant factors
- exploit easy instances

Design – Sorting



- simplicity
- reuse — disk scheduling, prefetching, load balancing, sequence partitioning
- constant factors — detailed machine model· (caches, TLBs, registers, branch prediction, ILP, communication)
- instances — randomization for difficult instances

Design – Parallel External Multiway Mergesort



- run formation: internal parallel sorting

  (multi-core parallel subroutines).

  shuffle blocks between runs randomly

- data redistribution by

  external inplace all-to-all

- node-local multi-core-parallel merging

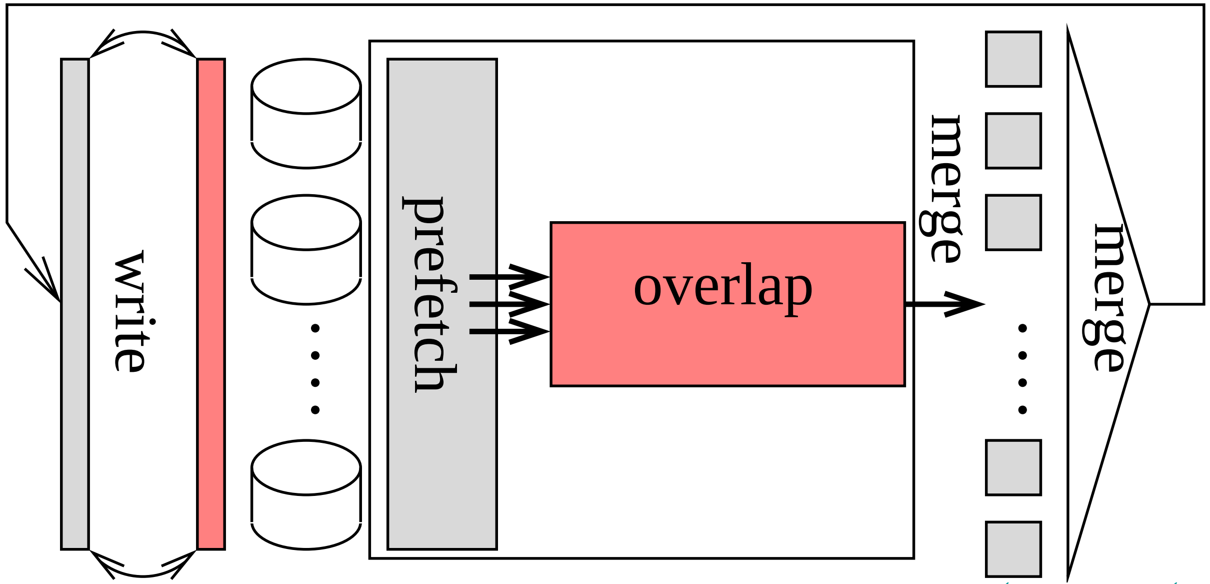
Analysis

- Constant factors matter

- Beyond worst case analysis

- Practical algorithms might be difficult to analyze
  (randomization, meta heuristics,. . .)

Analysis – Sorting



- Constant factors matter $(1+o(1))\mathrm{sort}(\mathrm{n})$

  I/Os for parallel (disk) external sorting

  Open Problem: optimal I/O AND communication volume

- Beyond worst case analysis Open Problem:

  quicksort: avg. case analysis of branch mispredictions

- Practical algorithms might be difficult to analyze Open Problem:

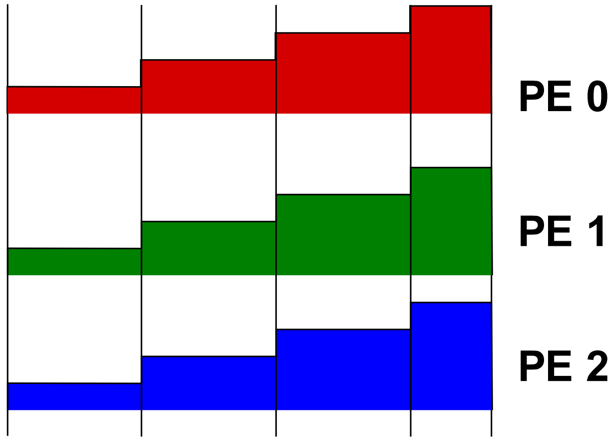
  greedy algorithm for parallel disk prefetching [Knuth@48]

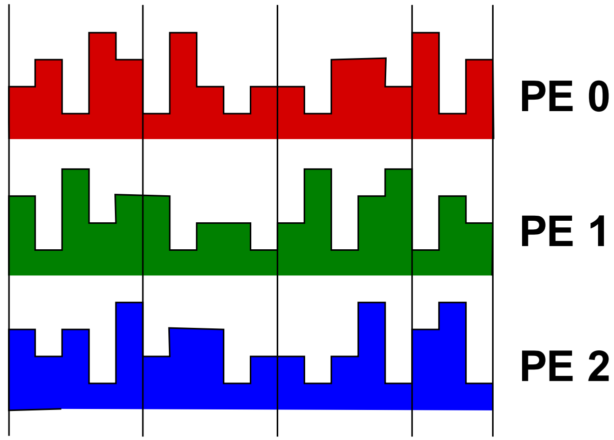
Analysis – Parallel External Sorting

Best case: 2 I/O passes, $1\times$communication (optimal)

Worst case: one additional I/O, communication pass

Expected case: much less extra I/O/comm. even for arbitrarily skewed inputs





Implementation

sanity check for algorithms !

Challenges

Semantic gaps:

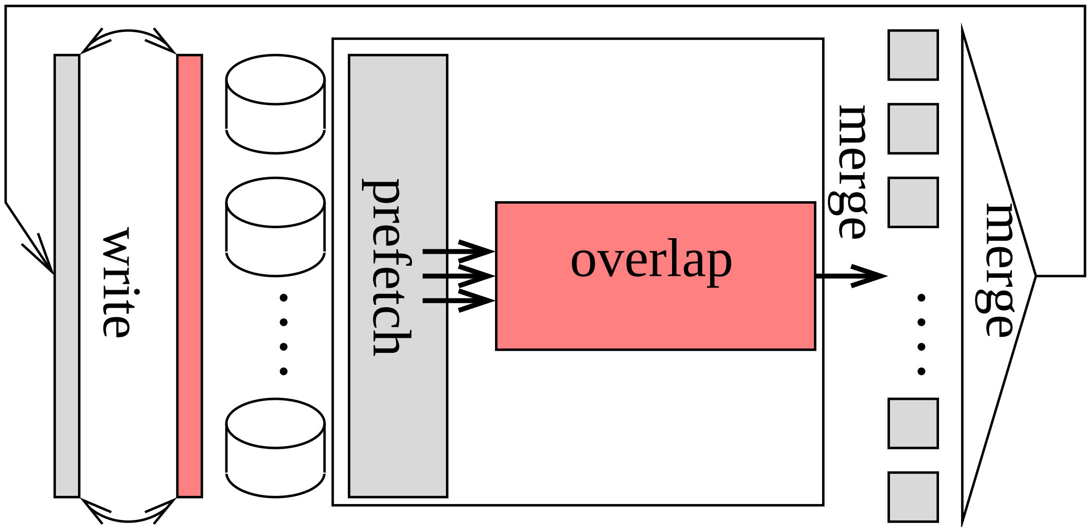
Abstract algorithm

$\leftrightarrow$

C++, OpenMP, MPI,. . .

$\leftrightarrow$

hardware

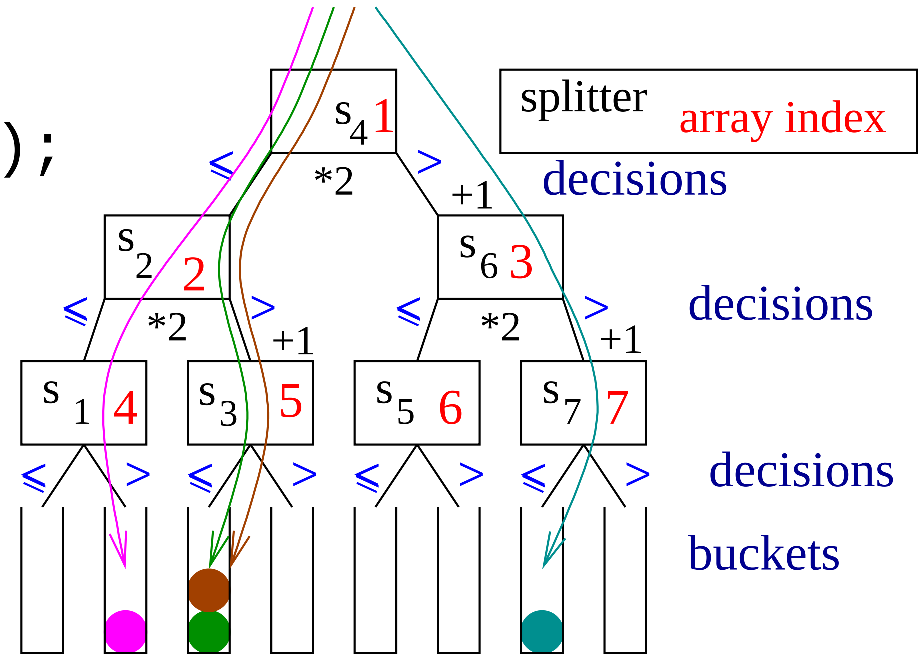


Small constant factors:

compare highly tuned competitors

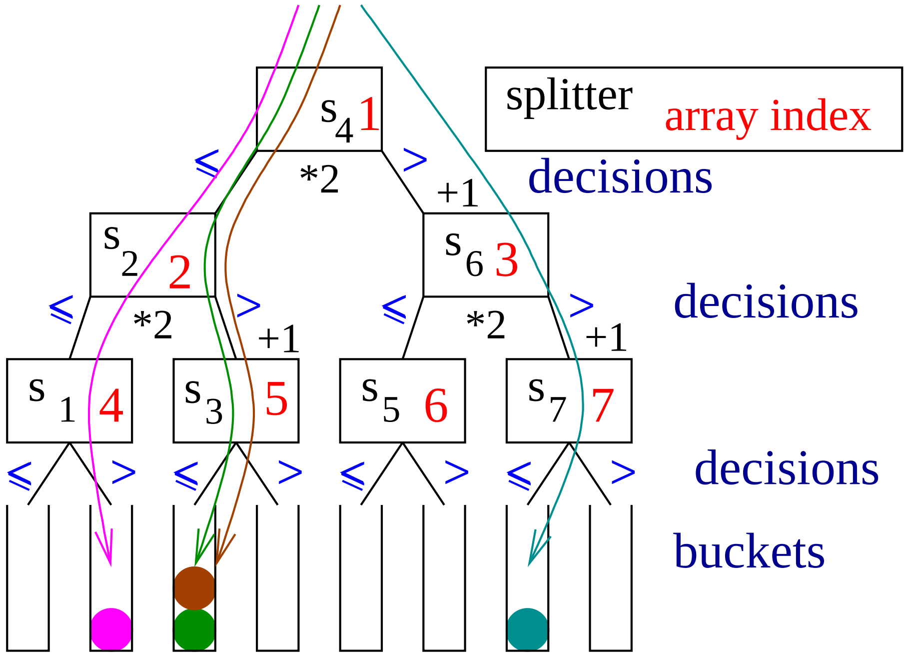
Example: Inner Loops Sample Sort

```
template <class T>
void findOraclesAndCount(const T* const a,
    const int n, const int k, const T* const s,
    Oracle* const oracle, int* const bucket) {
{ for (int i = 0;  i < n;  i++)
    int j = 1;
    while (j < k) {
      j = j*2 + (a[i] > s[j]);
    }
    int b = j-k;
    bucket[b]++;
    oracle[i] = b;
  }
}
```



Example: Inner Loops Sample Sort

```
template <class T>
void findOraclesAndCountUnrolled([...]){
  for (int i = 0;  i < n;  i++)
    int j = 1;
    j = j*2 + (a[i] > s[j]);
    j = j*2 + (a[i] > s[j]);
    j = j*2 + (a[i] > s[j]);
    int b = j-k;
    bucket[b]++;
    oracle[i] = b;
} }
```



Example: Inner Loops Sample Sort

```
template <class T>
void findOraclesAndCountUnrolled2([...]){
   for (int i = n & 1;  i < n;  i+=2) {\
      int j0 = 1;               int j1 = 1;
      T  ai0 = a[i];            T ai1  = a[i+1];
      j0=j0*2+(ai0>s[j0]);  j1=j1*2+(ai1>s[j1]);
      j0=j0*2+(ai0>s[j0]);  j1=j1*2+(ai1>s[j1]);
      j0=j0*2+(ai0>s[j0]);  j1=j1*2+(ai1>s[j1]);
      int b0 = j0-k;            int b1 = j1-k;
      bucket[b0]++;             bucket[b1]++;
      oracle[i] = b0;           oracle[i+1] = b1;
} }
```

Implementation – Parallel External Sorting

shared memory: g++ STL parallel mode (parallel multiway mergesort) (developed by Johannes Singler)

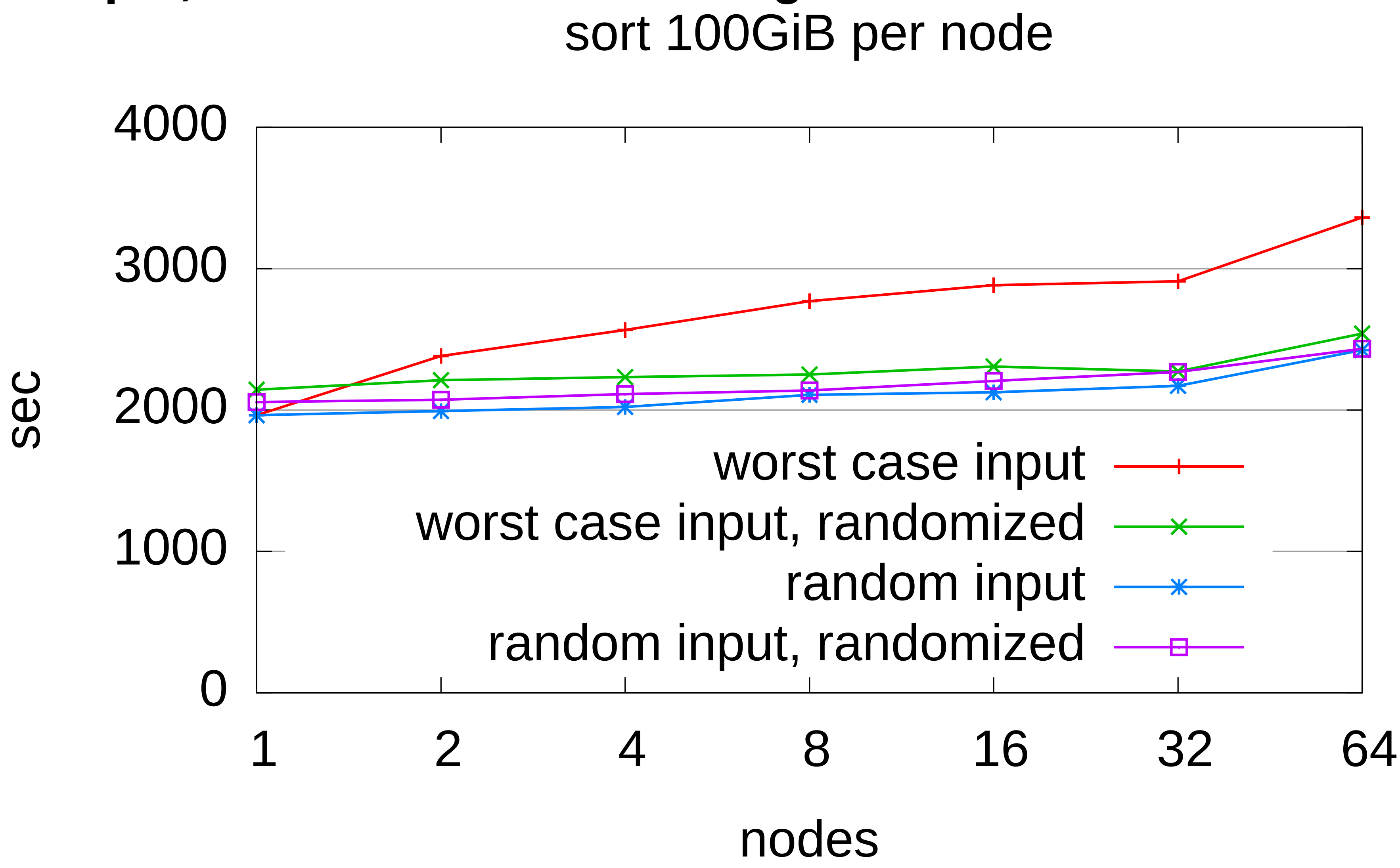
disk: STXXL by Roman Dementiev et al. overlapping, disk scheduling

distributed memory: MPI $+$ fix 32 bit problems + inplace external alltoallv

Experiments

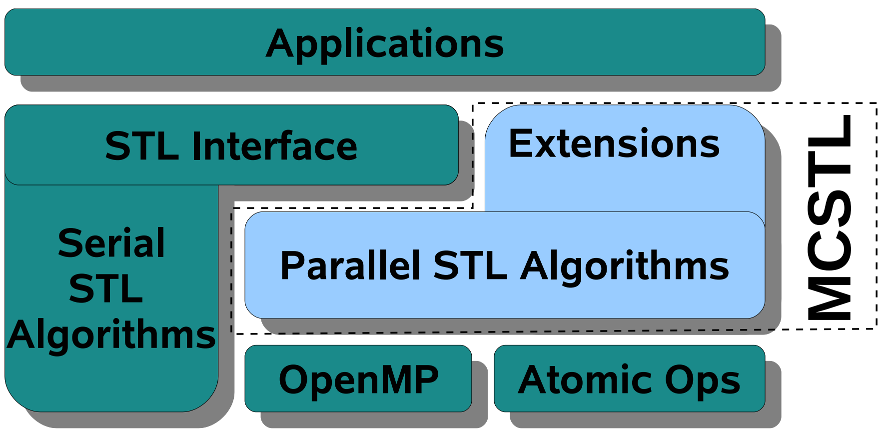
- sometimes a good surrogate for analysis
- too much rather than too little output data
- reproducibility (10 years!)
- software engineering
- reliable parallel running time measurements

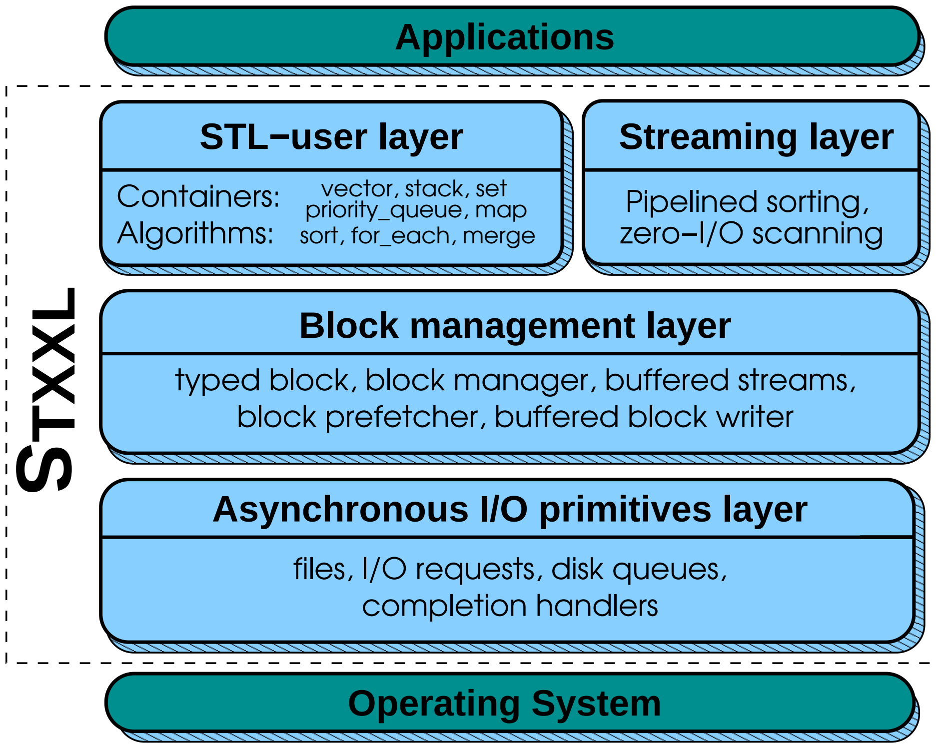
Example, Parallel External Sorting



Algorithm Libraries — Challenges

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

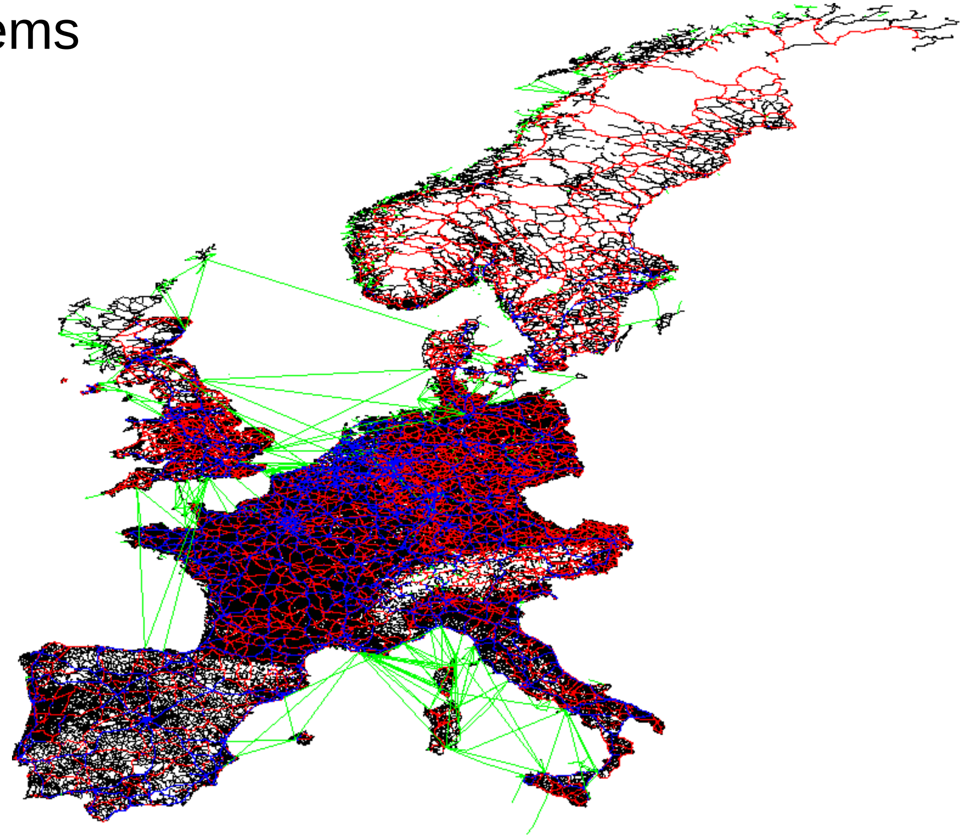




Problem Instances

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. . .)

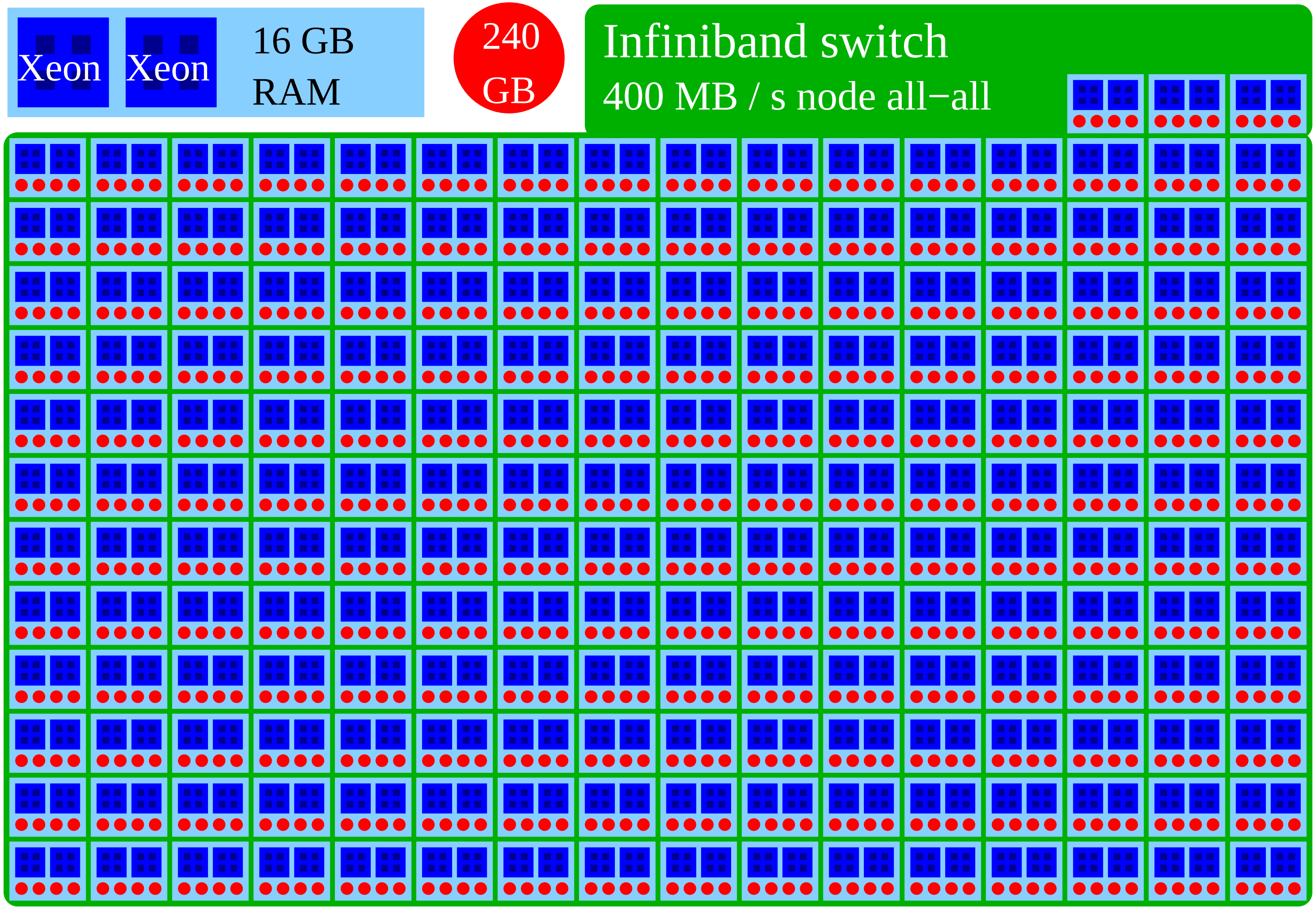
Example: Sorting Benchmark

100 byte records, 10 byte random keys, with file I/O

Category	data volume	performance	improvement
GraySort	100 000 GB	564 GB / min	$17\times$
MinuteSort	955 GB	955 GB / min	$> 10\times$
JouleSort	100 000 GB	3 400 Recs/Joule	$???\times$
JouleSort	1 000 GB	17 500 Recs/Joule	$5.1\times$
JouleSort	100 GB	39 800 Recs/Joule	$3.4\times$
JouleSort	10 GB	43 500 Recs/Joule	$5.7\times$

Also: PennySort

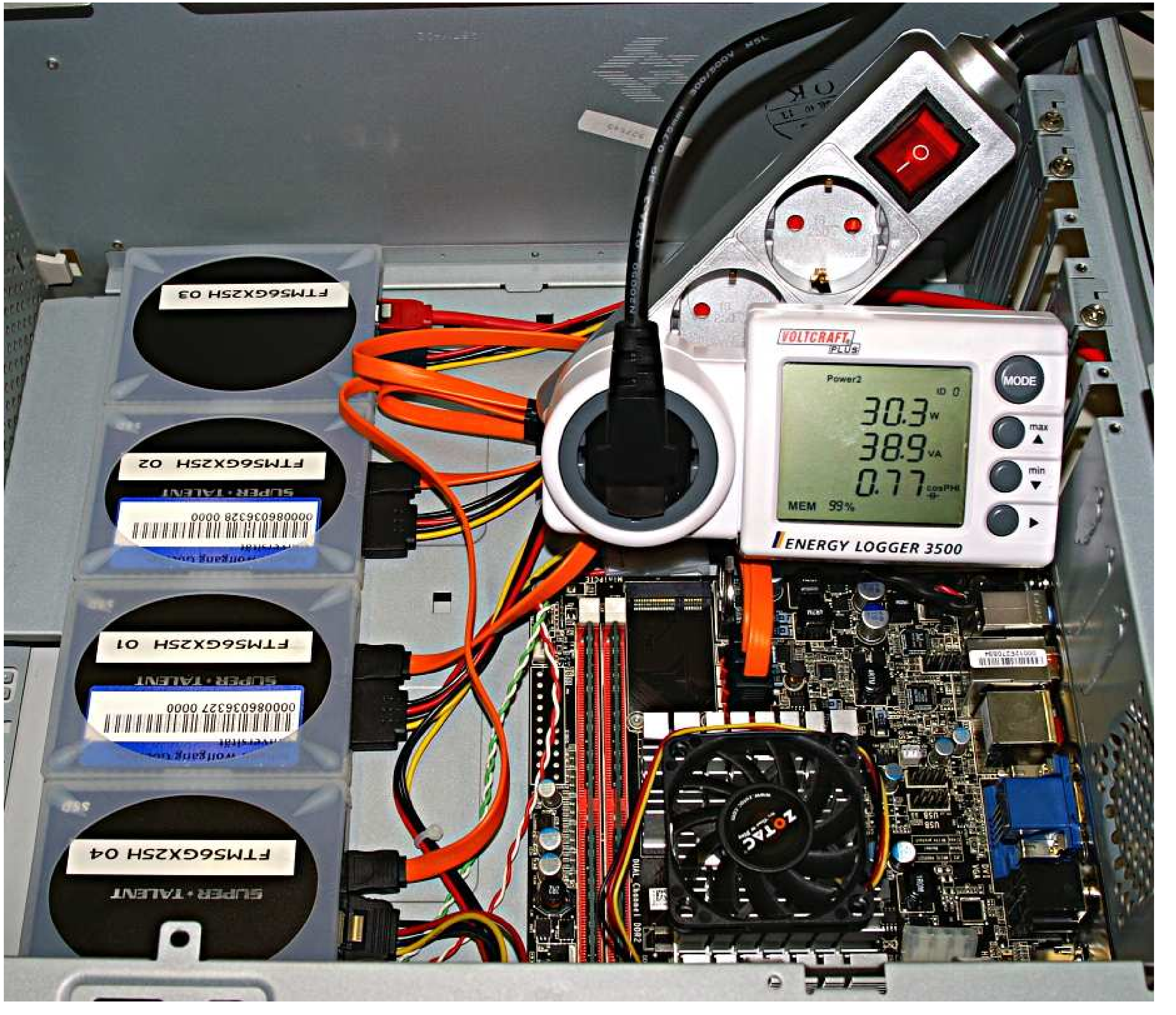
GraySort: inplace multiway mergesort, exact splitting



JouleSort

- Intel Atom N330
- 4 GB RAM
- 4×256 GB SSD (SuperTalent)

Algorithm similar to GraySort



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

Energy Grid: virtual powerplants (sun, wind, water, heat, negawatt),
disruption management,. . .

Communication Networks: mobile, cloud, 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

Interactions with other (Sub)disciplines