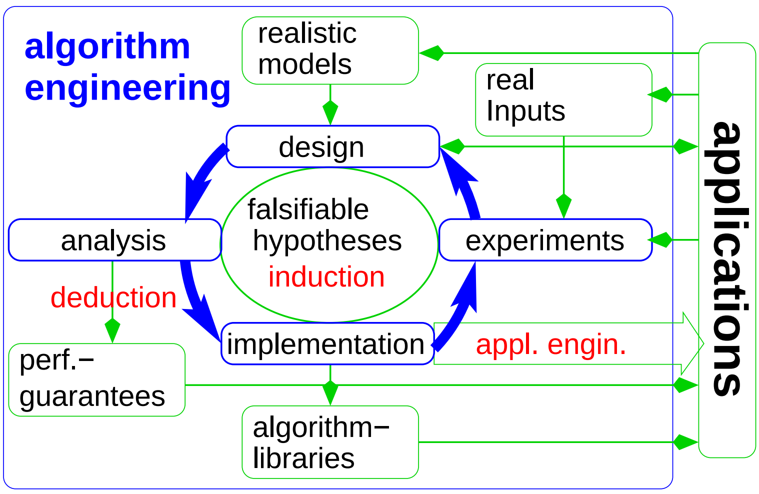
Tutorial: Algorithm Engineering for Big Data

Peter Sanders, Karlsruhe Institute of Technology

Efficient algorithms are at the heart of any nontrivial computer application. But how can we obtain innovative algorithmic solutions for demanding application problems with exploding input sizes using complex modern hardware and advanced algorithmic techniques?

This tutorial proposes algorithm engineering as a methodology for taking all these issues into account. Algorithm engineering tightly integrates modeling, algorithm design, analysis, implementation and experimental evaluation into a cycle resembling the scientific method used in the natural sciences. Reusable, robust, flexible, and efficient implementations are put into algorithm libraries. Benchmark instances provide further coupling to applications.



We begin with examples representing fundamental algorithms and data structures with a particular emphasis on large data sets. We first look at **sorting** in detail. Then we will have shorter examples for **full text indices**, **priority queue** data structures, **route planning**, **graph partitioning**, and **minimum spanning trees**. We will also give examples of future challenges centered on particular big data applications like **genome sequencing** and phylogenetic tree reconstruction, **particle tracking** at the CERN LHC, and the SAP-HANA **data base**,

Further Information

Duration: half-day

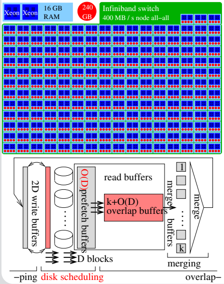
Intended Audience: Practitioners with some basic background in algorithms (2nd semester computer science in most German universities)

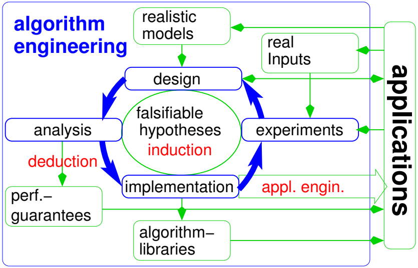
Slides are attached. Some images with unclear copyright are removed

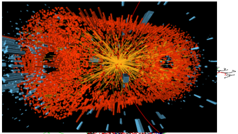
Algorithm Engineering for Big Data

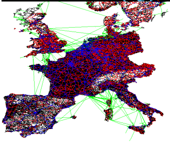
Peter Sanders

Institute of Theoretical Informatics - Algorithmics









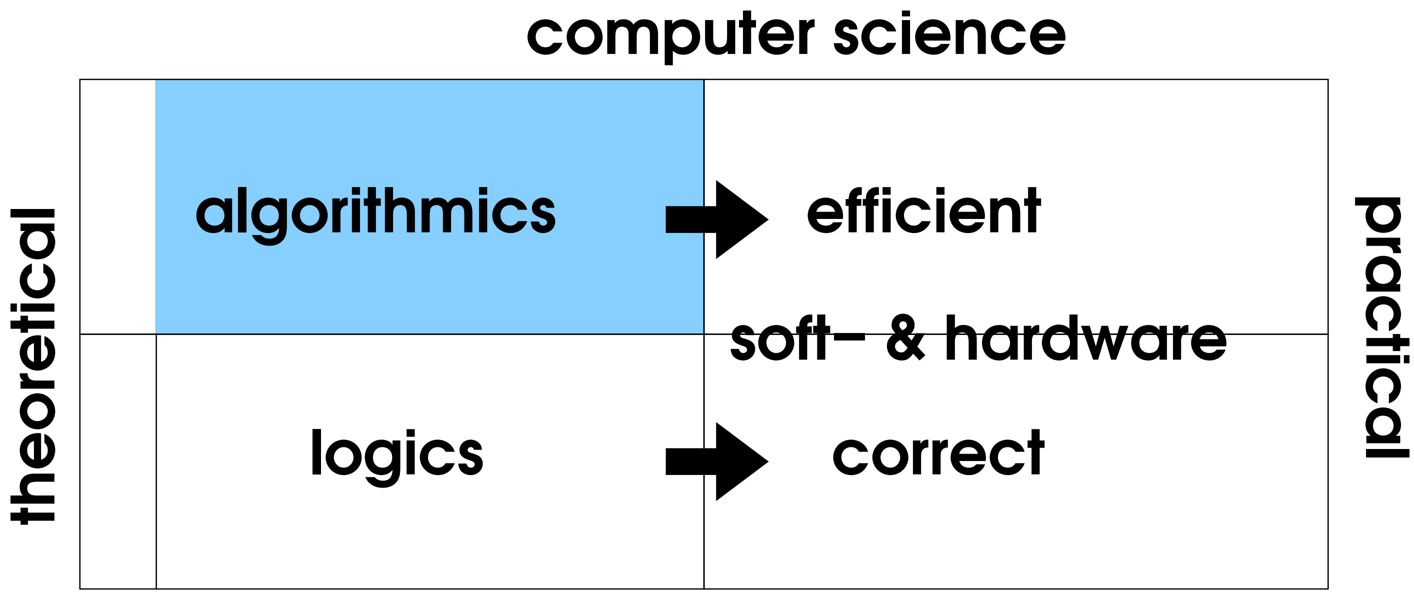
Overview

- A detailed explanation of algorithm engineering with sorting for (more or less) big inputs as a throughgoing example
- More Big Data examples from my group

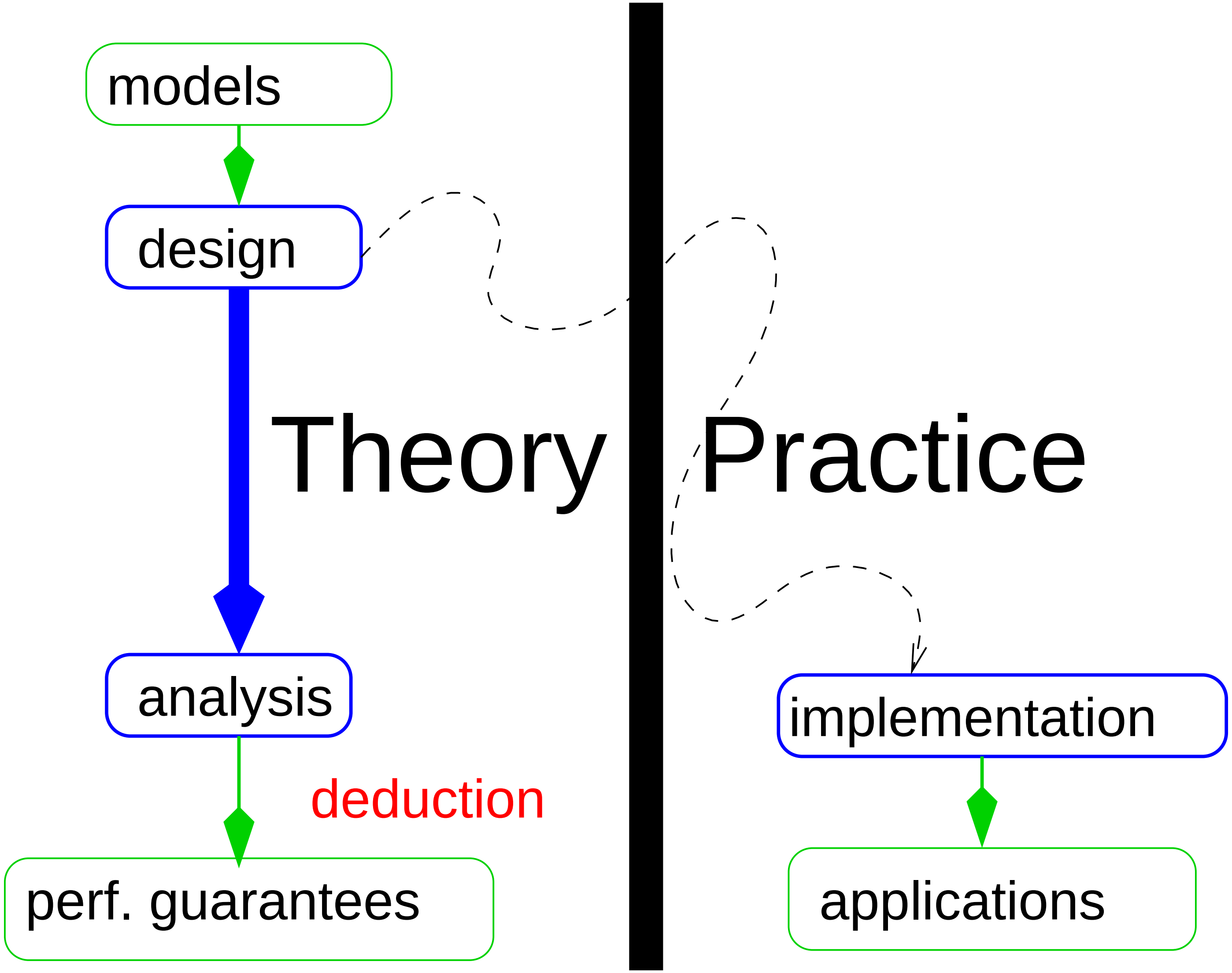
[with: David Bader, Veit Batz, Andreas Beckmann, Timo Bingmann, Stefan Burkhardt, Jonathan Dees, Daniel Delling, Roman Dementiev, Daniel Funke, Robert Geisberger, David Hutchinson, Juha Kärkkäinen, Lutz Kettner, Moritz Kobitzsch, Nicolai Leischner, Dennis Luxen, Kurt Mehlhorn, Ulrich Meyer, Henning Meyerhenke, Rolf Möhring, Ingo Müller, Petra Mutzel, Vitaly Osipov, Felix Putze, Günther Quast, Mirko Rahn, Dennis Schieferdecker, Sebastian Schlag, Dominik Schultes, Christian Schulz, Jop Sibeyn, Johannes Singler, Jeff Vitter, Dorothea Wagner, Jan Wassenberg, Martin Weidner, Sebastian Winkel, Emmanuel Ziegler]

Algorithmics

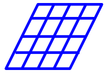
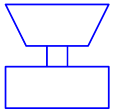
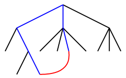
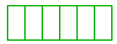
= the systematic design of efficient software and hardware



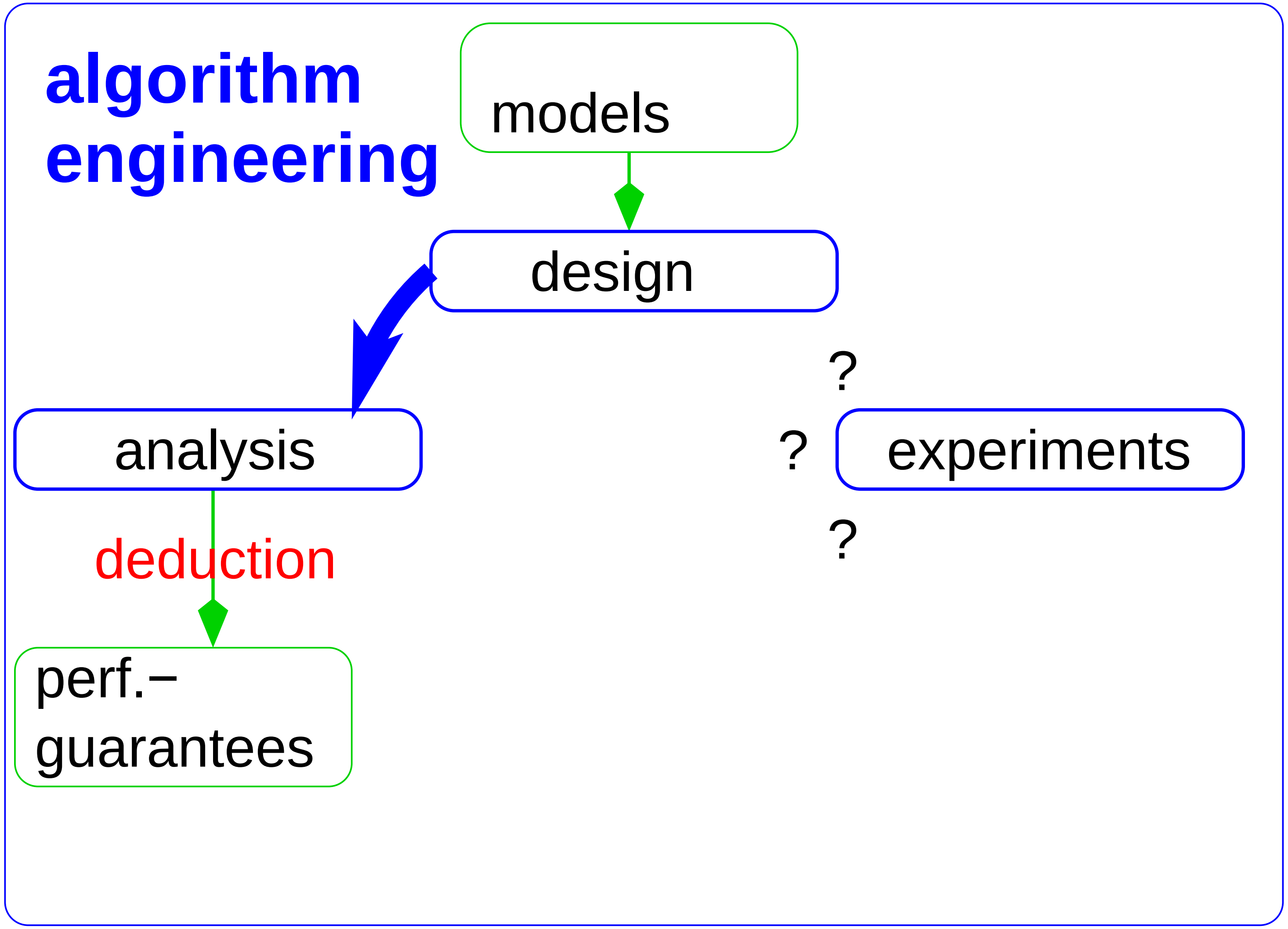
(Caricatured) Traditional View: Algorithm Theory



Gaps Between Theory & Practice

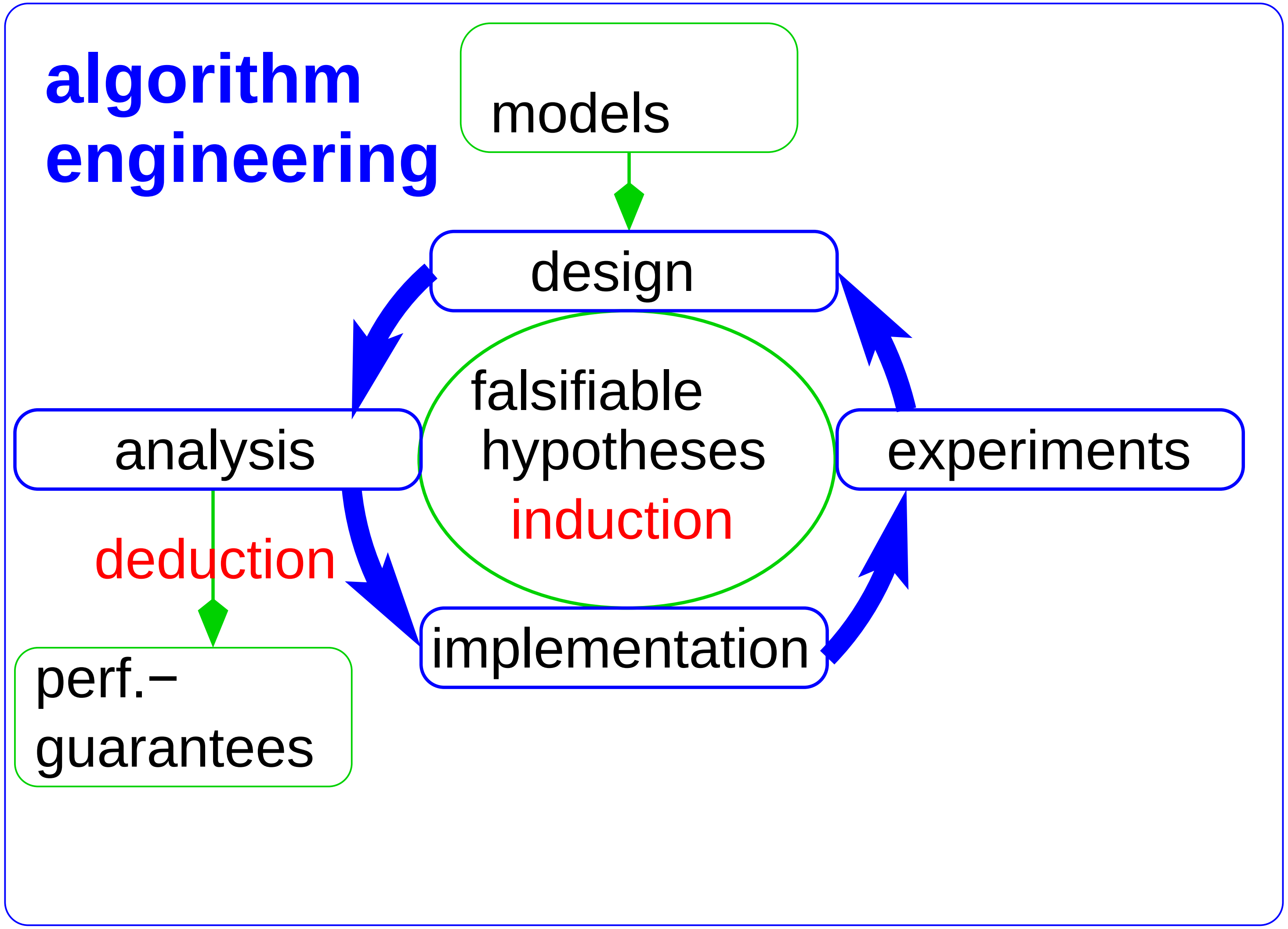
Theory	$\longleftrightarrow$	**Practice**
simple 	**appl. model**	complex
simple 	**machine model**	real
complex	**algorithms**	$\boxed{\text{FOR}}$ simple
advanced 	**data structures**	 arrays, . . .
worst case $\boxed{\max}$	**complexity measure**	inputs
asympt. $\boxed{\mathcal{O}(\cdot)}$	**efficiency**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

Algorithmics as Algorithm Engineering



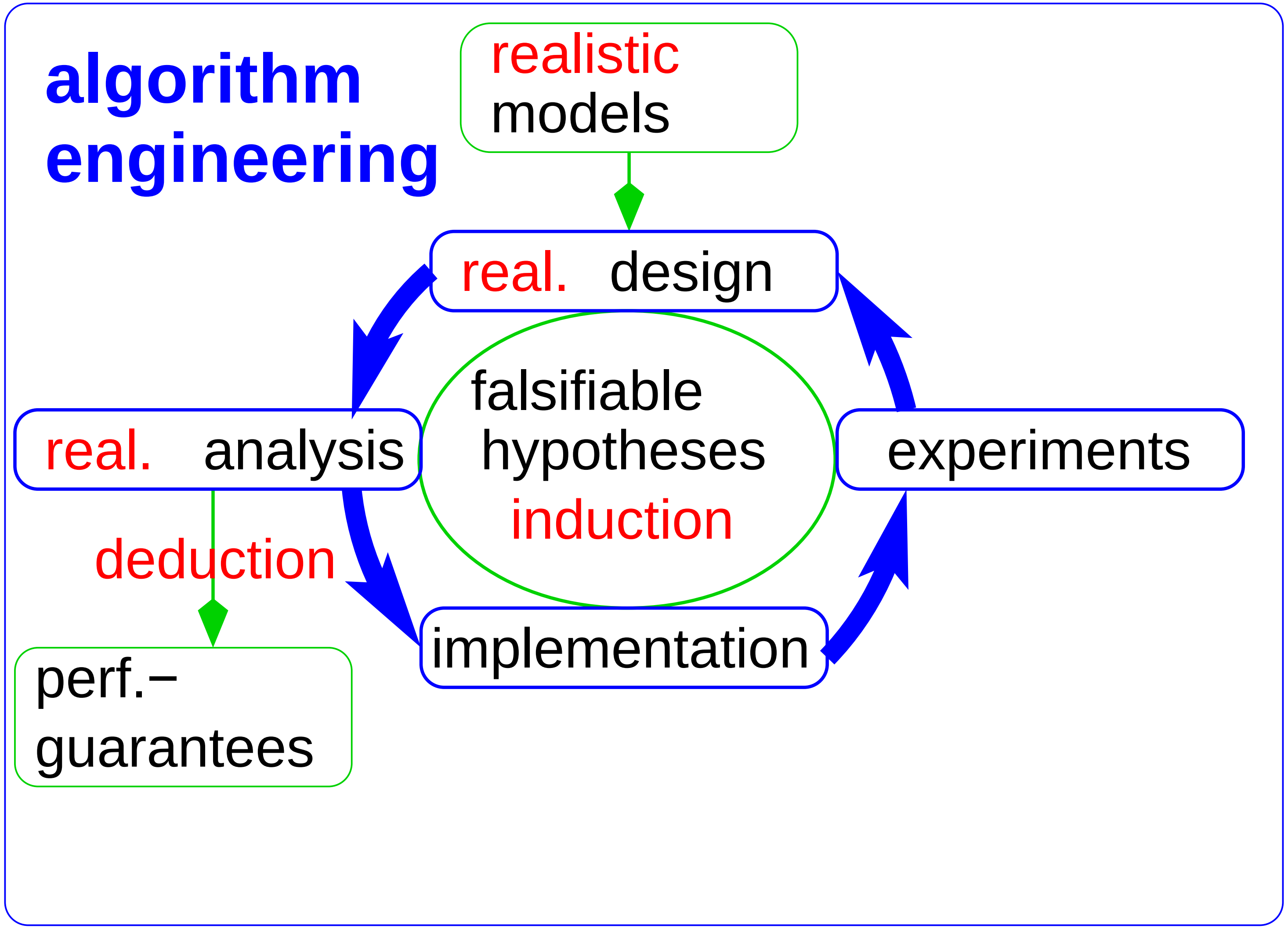
[1]

Algorithmics as Algorithm Engineering



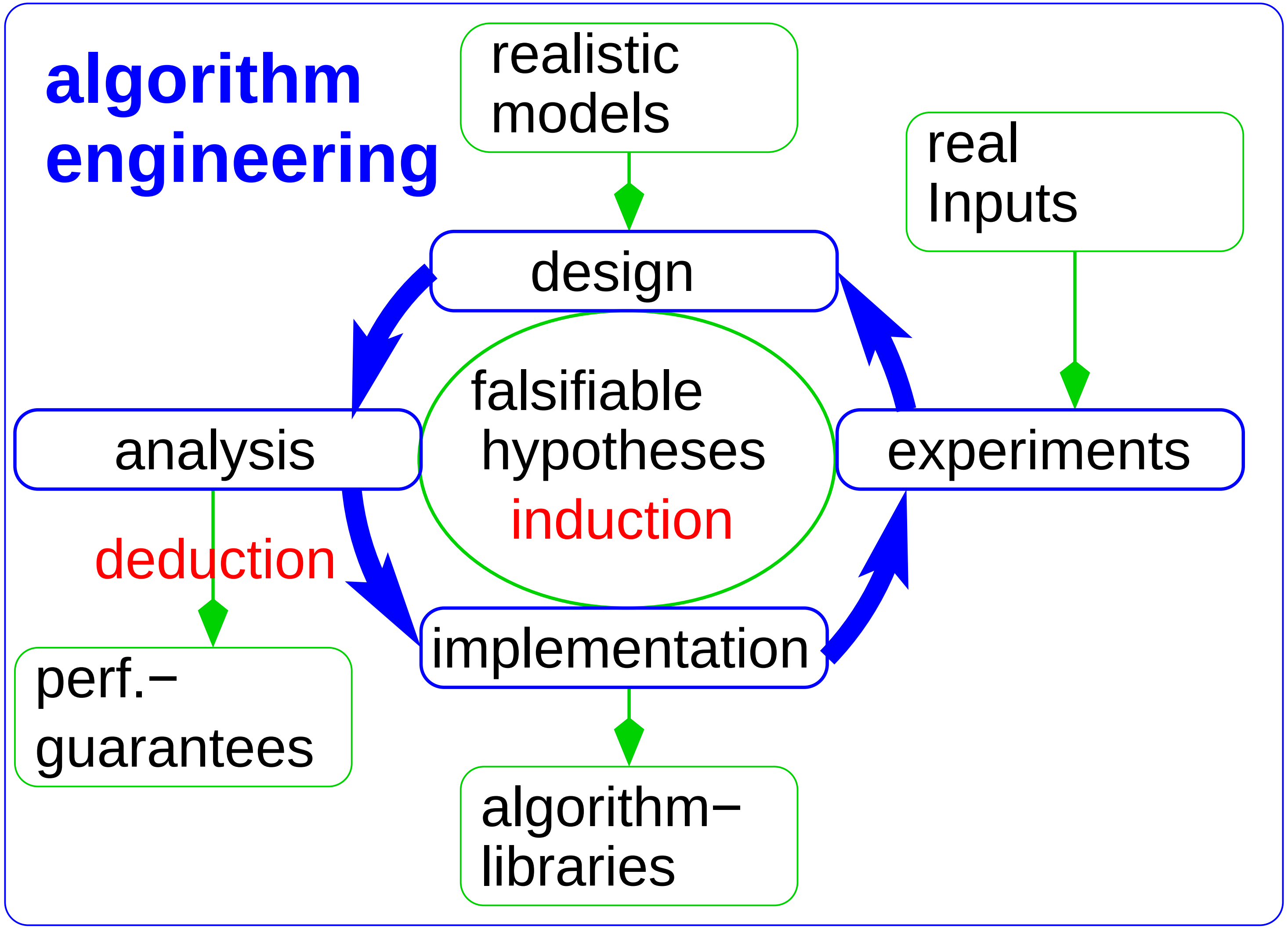
[1]

Algorithmics as Algorithm Engineering



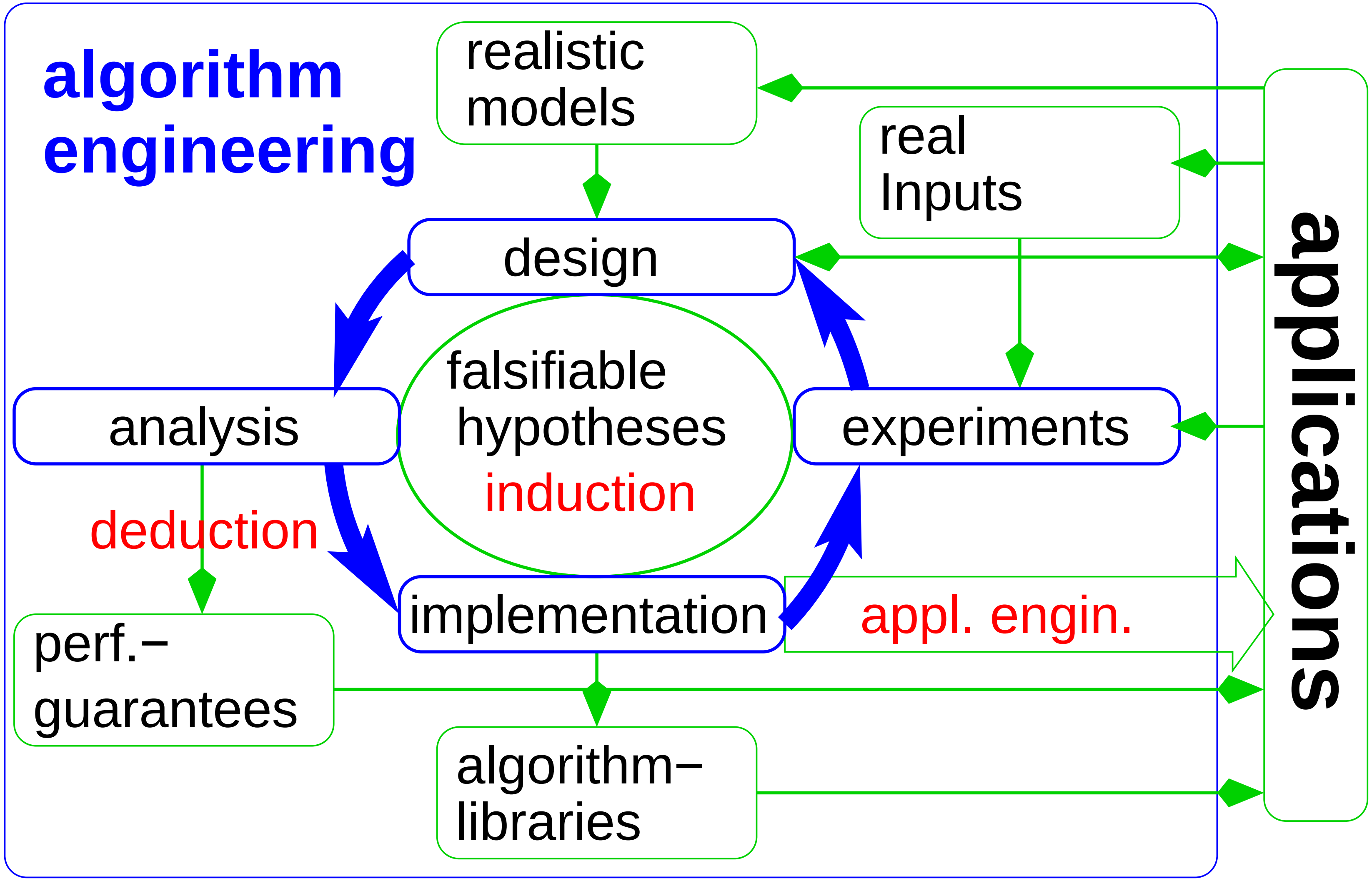
[1]

Algorithmics as Algorithm Engineering



[1]

Algorithmics as Algorithm Engineering



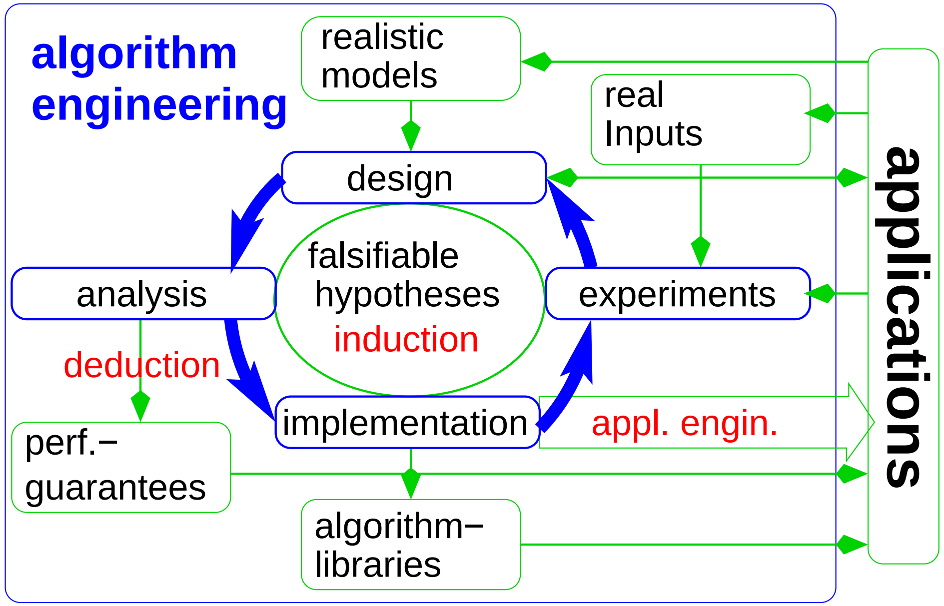
[1]

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



Bits of History

1843– Algorithms in theory and practice

1950s,1960s Still infancy

1970s,1980s Paper and pencil algorithm theory.

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

1986 Term used by [T. Beth],

lecture “Algorithmentechnik” in Karlsruhe.

1988– Library of Efficient Data Types and Algorithms

(LEDA) [2]

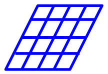
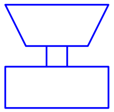
1997– Workshop on Algorithm Engineering

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

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

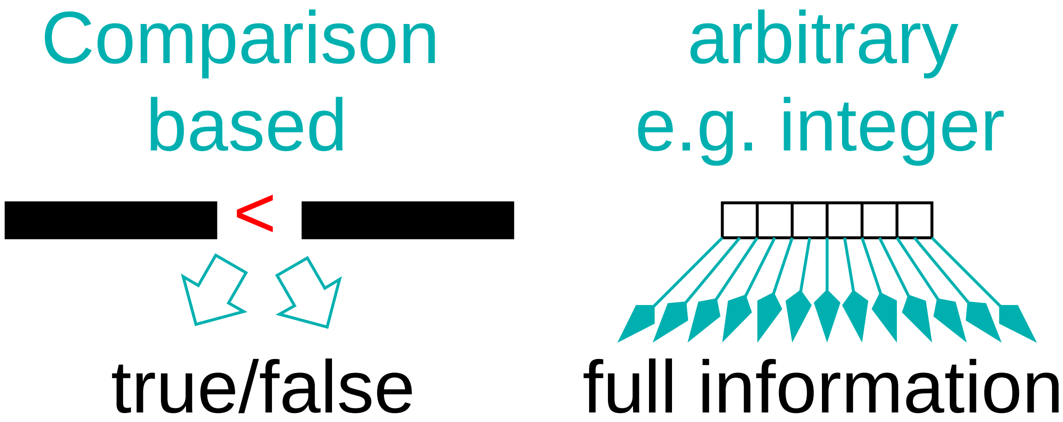
1998 Alex workshop in Italy $\rightsquigarrow$ ALENEX

Realistic Models

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

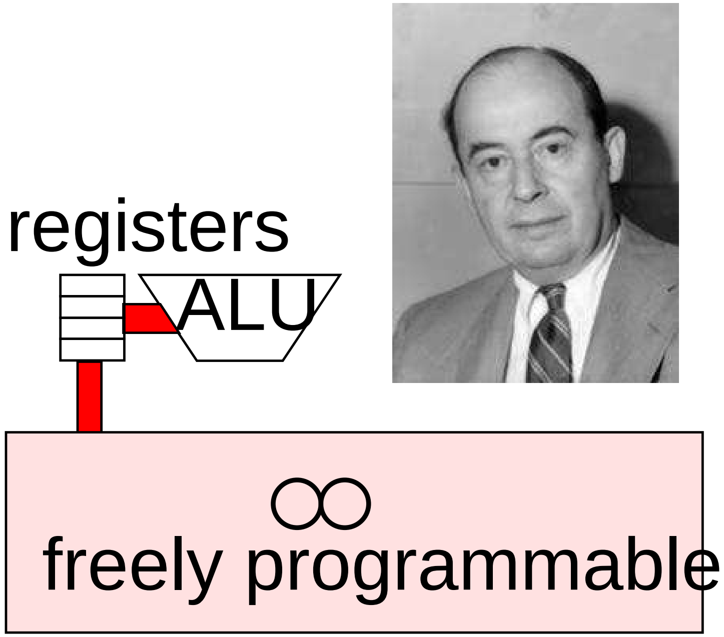
- Careful refinements
- Try to preserve (partial) analyzability / simple results

Sorting – Model



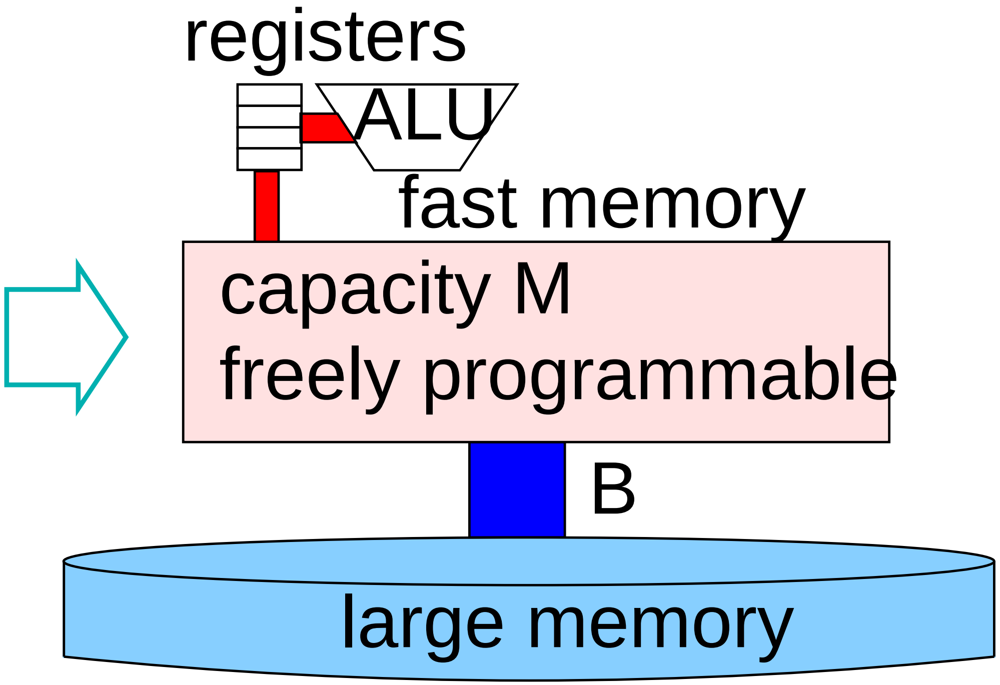
Advanced Machine Models[3]

RAM / von Neumann



count instructions

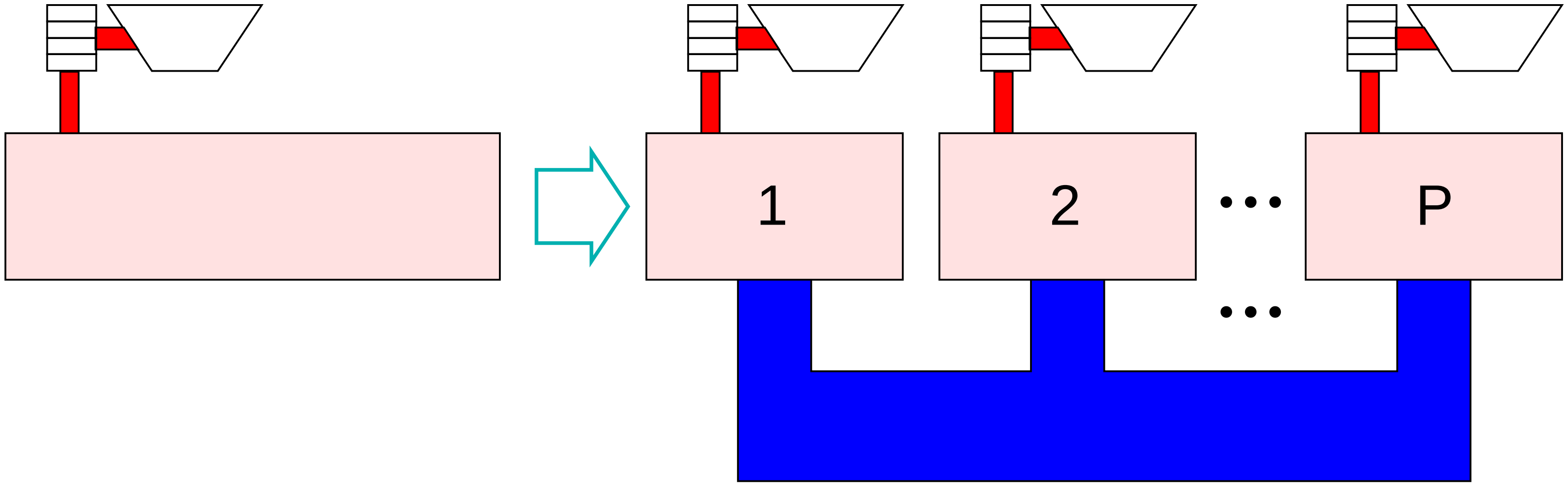
External



(also) count (block) I/Os

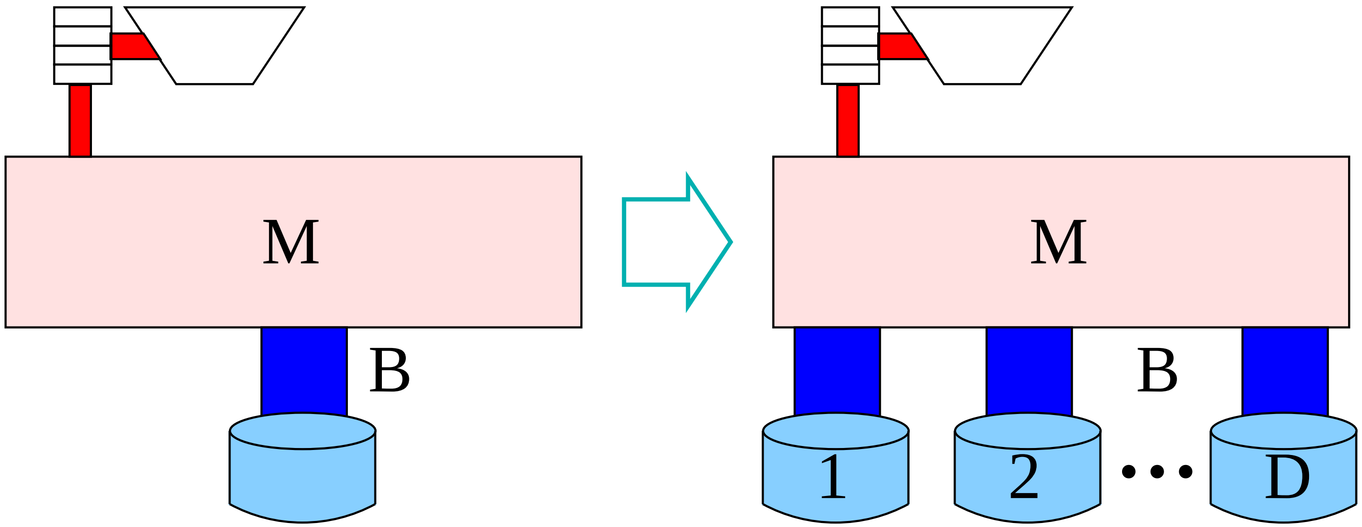
[3]

Distributed Memory [4]



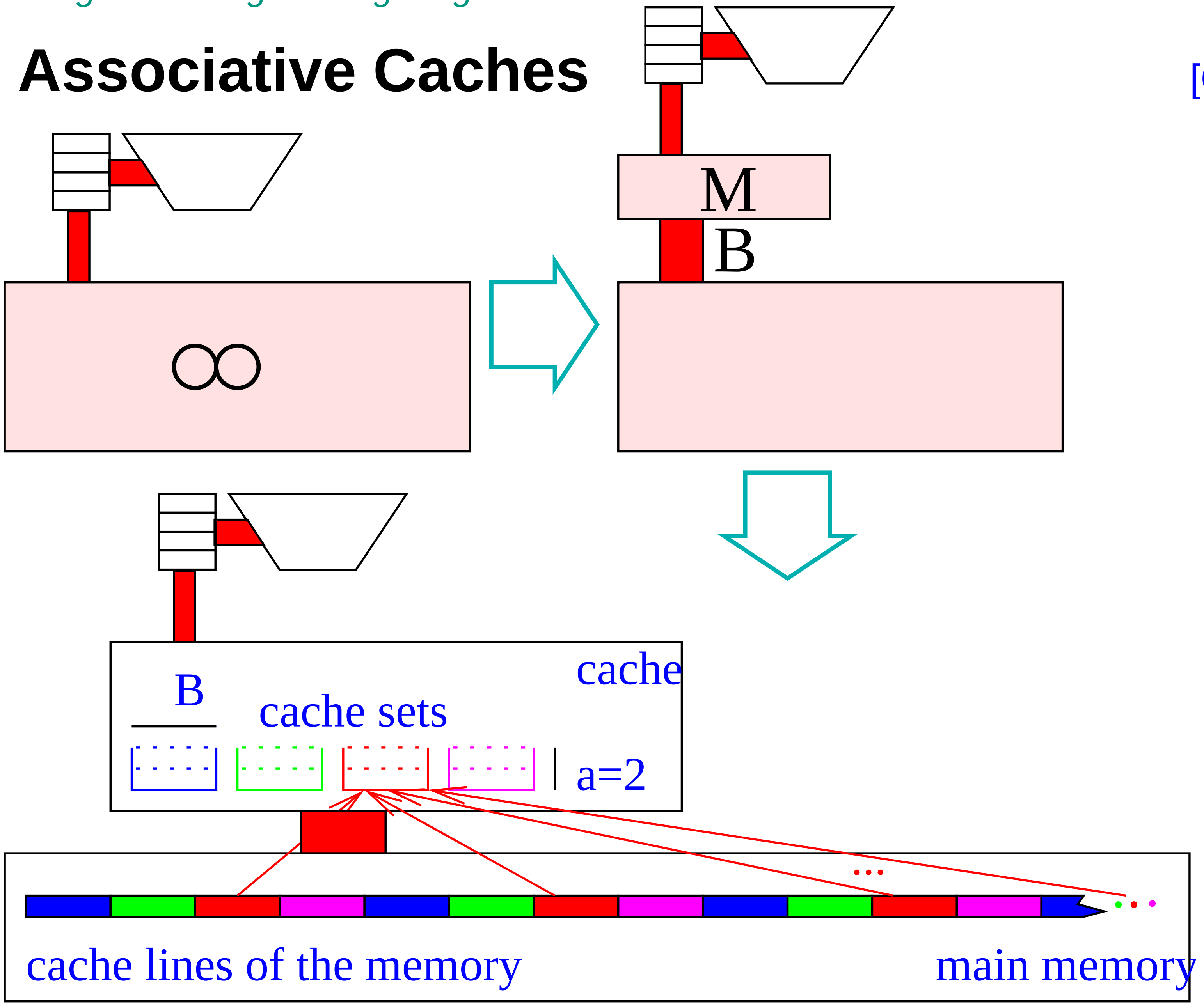
(also) determine communication volume

Parallel Disks [5]



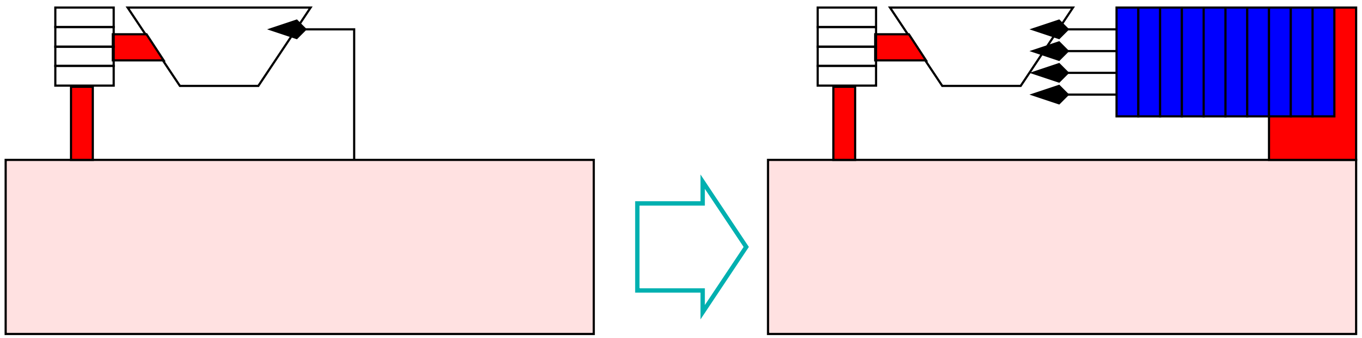
Set Associative Caches

[6]

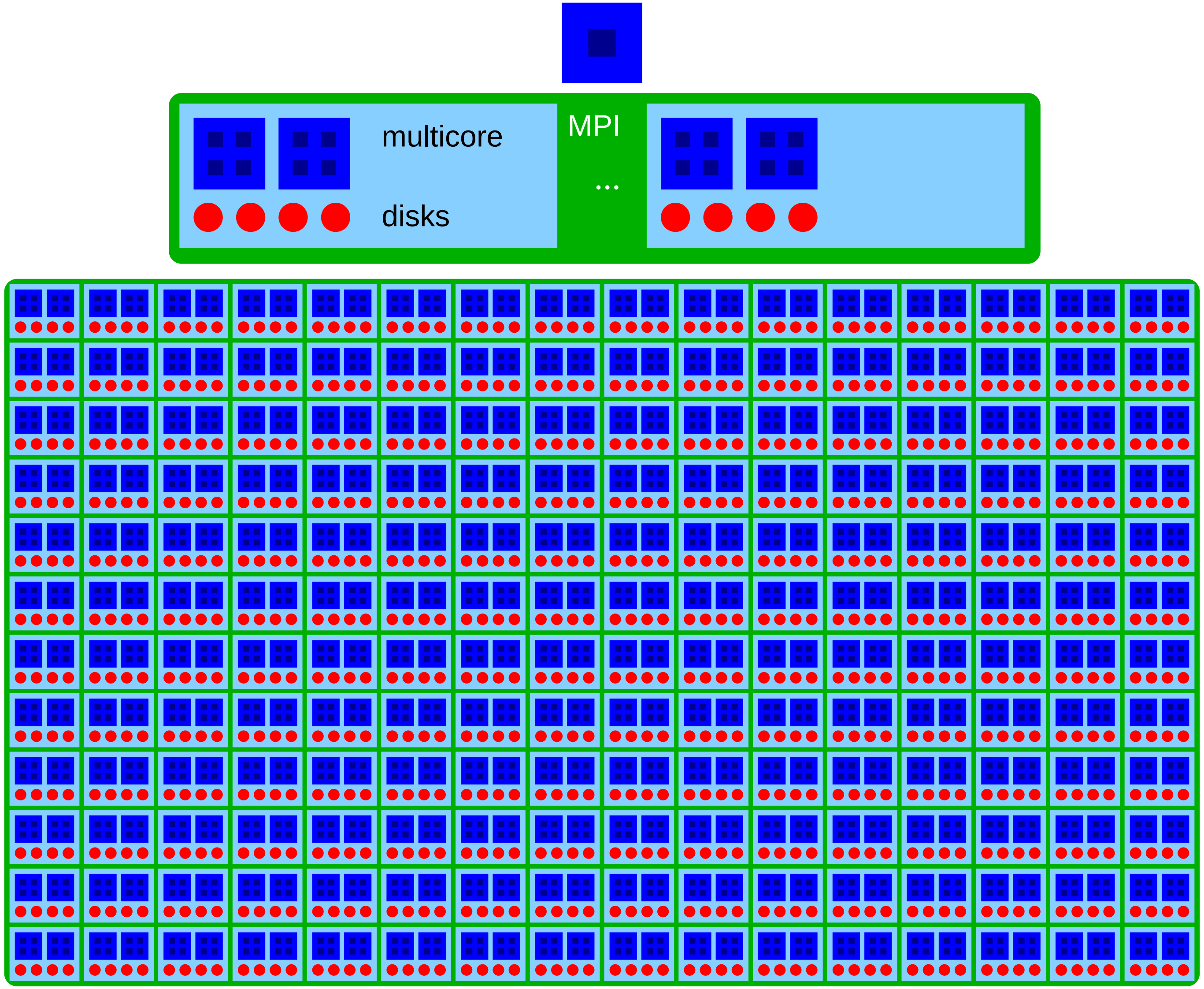


Branch Prediction

[7]



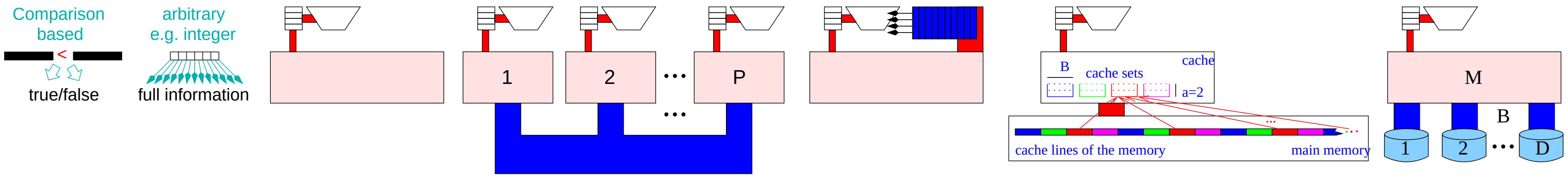
Hierarchical Parallel External Memory [8]



Graphics Processing Units [9]

Combininig Models?

- design / analyze one aspect at a time
- hierarchical combination
- autotuning ?

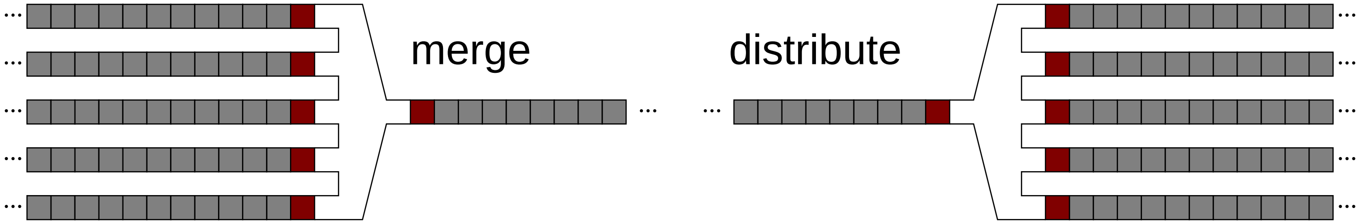


Design

of algorithms that work well in practice

- simplicity
- reuse
- constant factors
- exploit easy instances

Design – Sorting



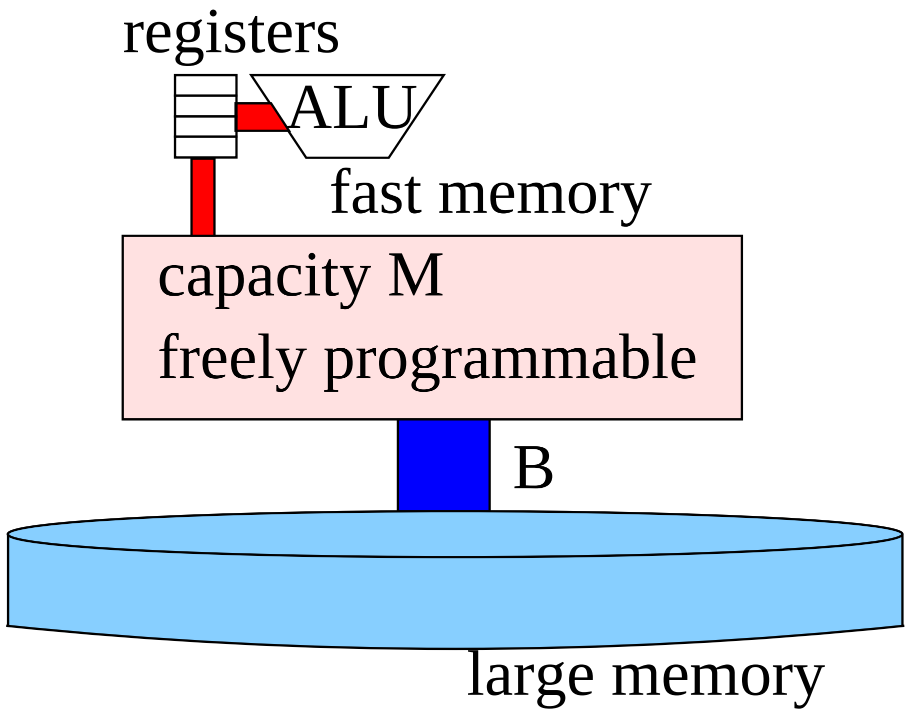
- simplicity
- reuse — disk scheduling, prefetching, load balancing, sequence partitioning [10, 5, 11, 8]
- constant factors — detailed machine model· (caches, TLBs, registers, branch prediction, ILP) [3, 7]
- instances — randomization for difficult instances [5, 8]

Example: External Sorting [12]

n: input size

M: internal memory size

B: block size



Procedure externalMerge$(a, b, c$:File **of** Element$)$

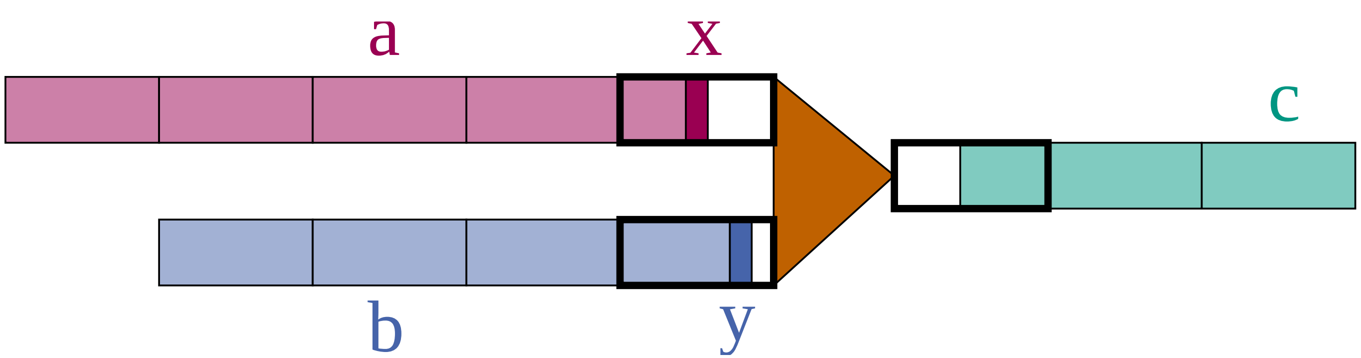
x := a.readElement // Assume emptyFile.readElement$= \infty$

y := b.readElement

for j := 1 **to** $|a| + |b|$ **do**

if $x \le y$ **then** c.writeElement(x); x := a.readElement

else c.writeElement(y); y := b.readElement



External Binary Merging

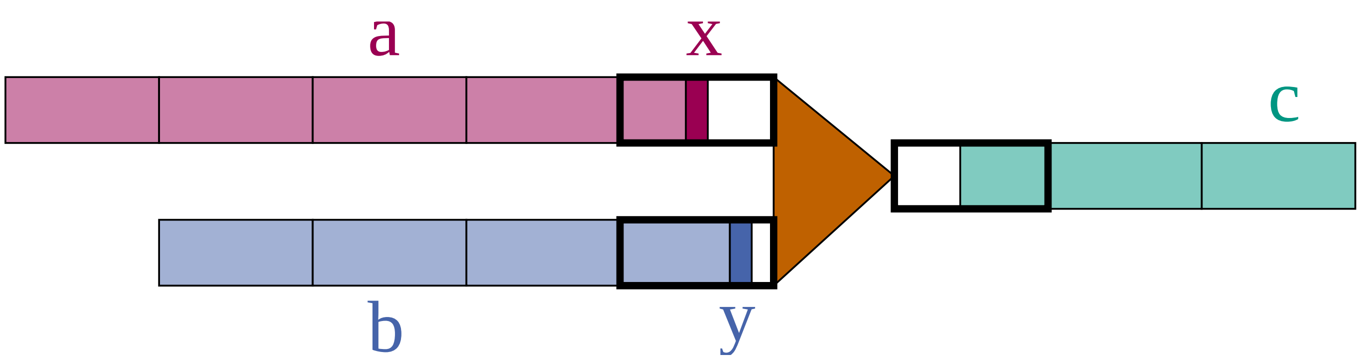
read file a: $\approx |a|/B$.

read file b: $\approx |b|/B$.

write file c: $\approx (|a|+|b|)/B$.

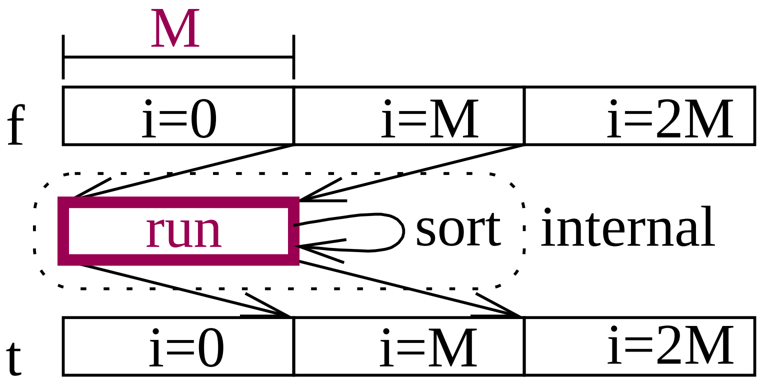
overall:

$$\approx 2\frac{|a|+|b|}{B}$$



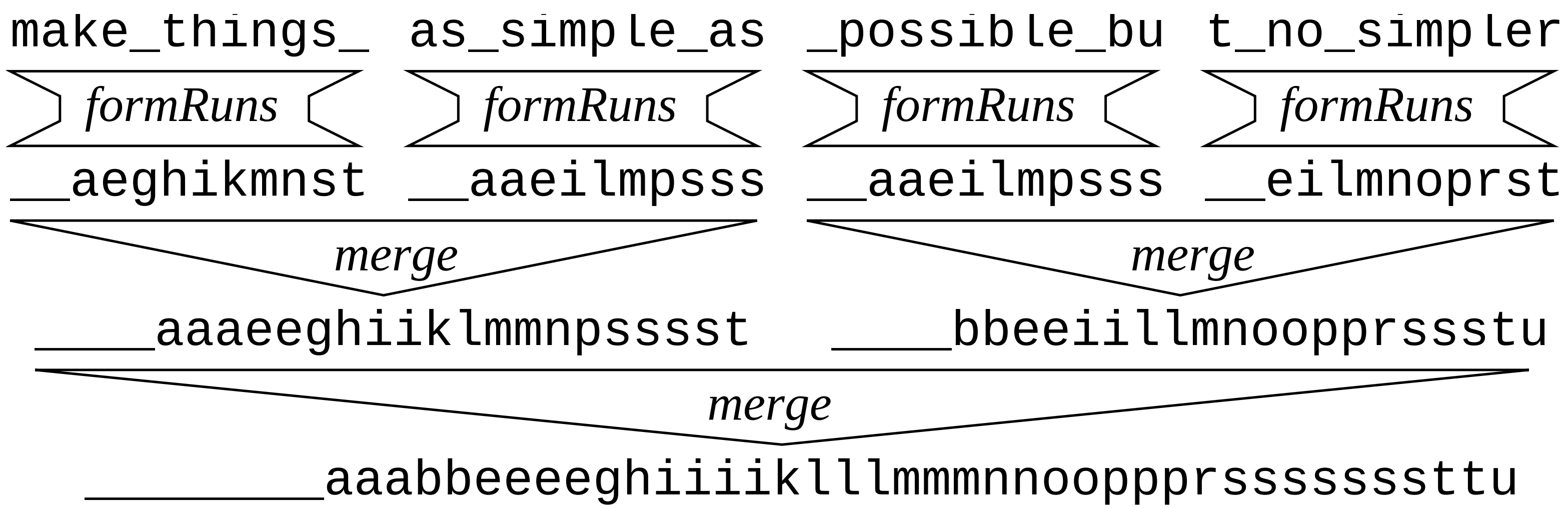
Run Formation

Sort input pieces of size M



I/Os: $\approx 2\frac{n}{B}$

Sorting by External Binary Merging



Procedure externalBinaryMergeSort // I/Os: $\approx$

run formation // $2n/B$

while more than one run left **do** // $\left\lceil \log \frac{n}{M} \right\rceil \times$

merge pairs of runs // 2n/B

output remaining run // $\sum : 2\frac{n}{B}\left(1 + \left\lceil \log \frac{n}{M} \right\rceil\right)$

Example Numbers: PC 2013

$n = 2^{40}$ Byte (1 TB)

$M = 2^{33}$ Byte (8 GB)

$B = 2^{22}$ Byte (4 MB)

one I/O needs 2^{-5} s (31.25 ms)

$$\begin{aligned}\text{time } &= 2\frac{n}{B}\left(1 + \left\lceil \log \frac{n}{M} \right\rceil\right) \cdot 2^{-5}\text{s} \\ &= 2 \cdot 2^{18} \cdot (1+7) \cdot 2^{-5}\text{s} = 2^{17}\text{s} \approx 36\text{h}\end{aligned}$$

Idea: 8 passes $\rightsquigarrow$ 2 passes

Multiway Merging

Procedure multiwayMerge$(a_1, \ldots, a_k, c$:File **of** Element$)$

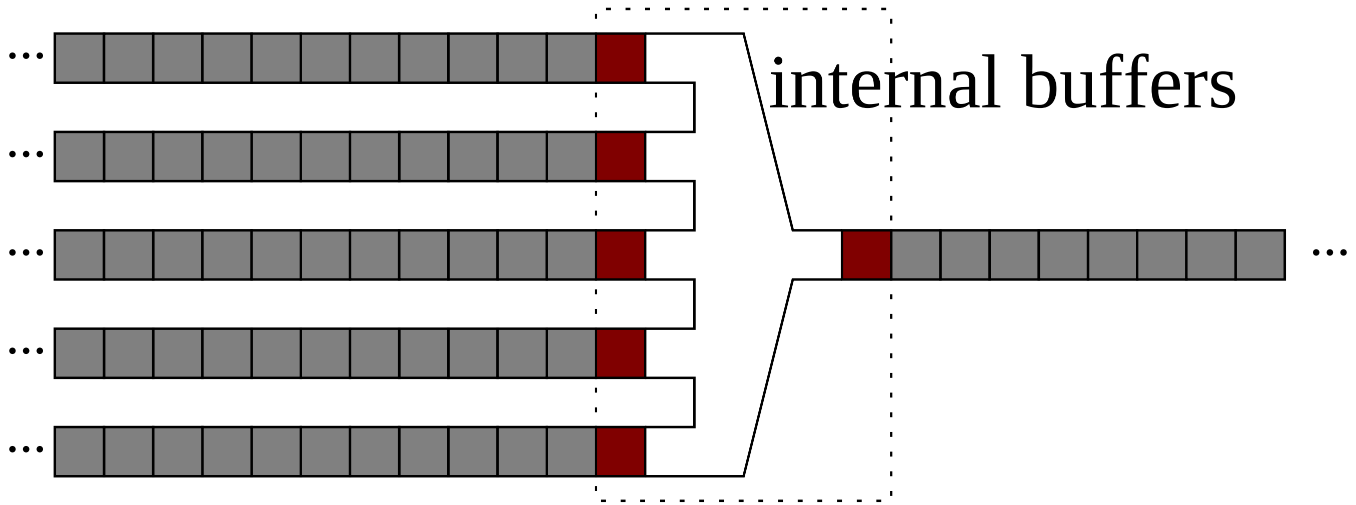
for i := 1 **to** k **do** x_i:= a_i.readElement

for j := 1 **to** $\sum_{i=1}^{k} |a_i|$ **do**

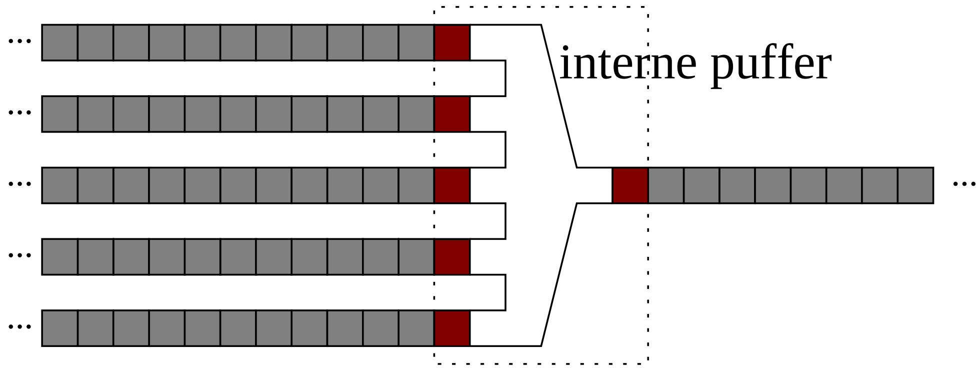
find $i \in 1..k$ that minimizes x_i // no I/Os!, $\mathcal{O}(\log k)$ time

c.writeElement(x_i)

x_i := a_i.readElement



Mulitway Merging – Analysis



I/Os: read file a_i: $\approx |a_i|/B$.

write file c: $\approx \sum_{i=1}^{k} |a_i|/B$

overall:

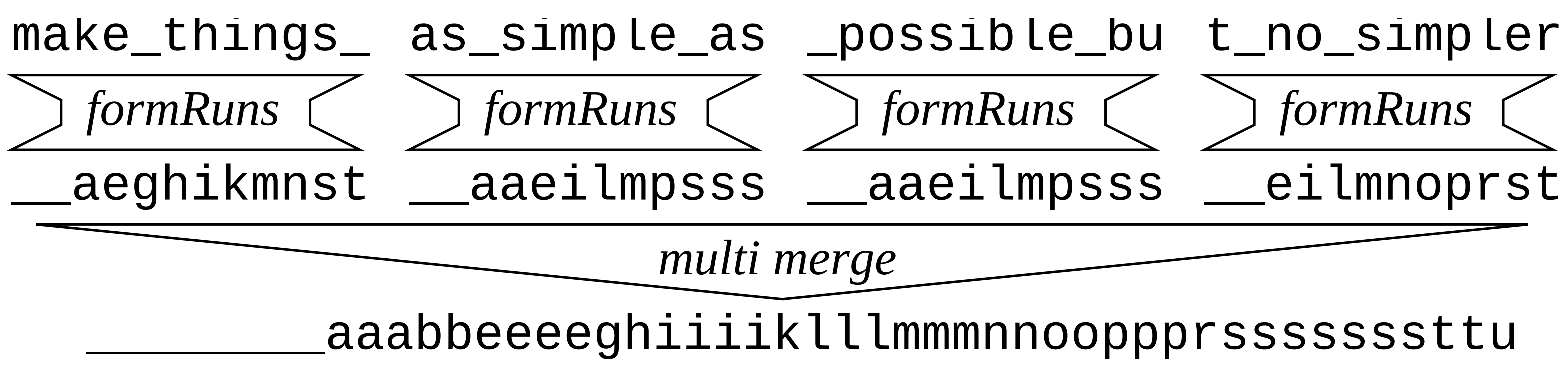
$$\leq\approx 2\frac{\sum_{i=1}^{k} |a_i|}{B}$$

constraint: We need $k+1$ buffer blocks, i.e., $k+1 < M/B$

Sorting by Multiway-Merging

- sort $\lceil n/M \rceil$ runs with M elements each $2n/B$ I/Os
- merge M/B runs at a time $2n/B$ I/Os
- unit a single run remains $\times \left\lceil \log_{M/B} \frac{n}{M} \right\rceil$ merging phases

overall $\mathrm{sort}(n) := \frac{2n}{B}\left(1 + \left\lceil \log_{M/B} \frac{n}{M} \right\rceil\right)$ I/Os



External Sorting by Multiway-Merging

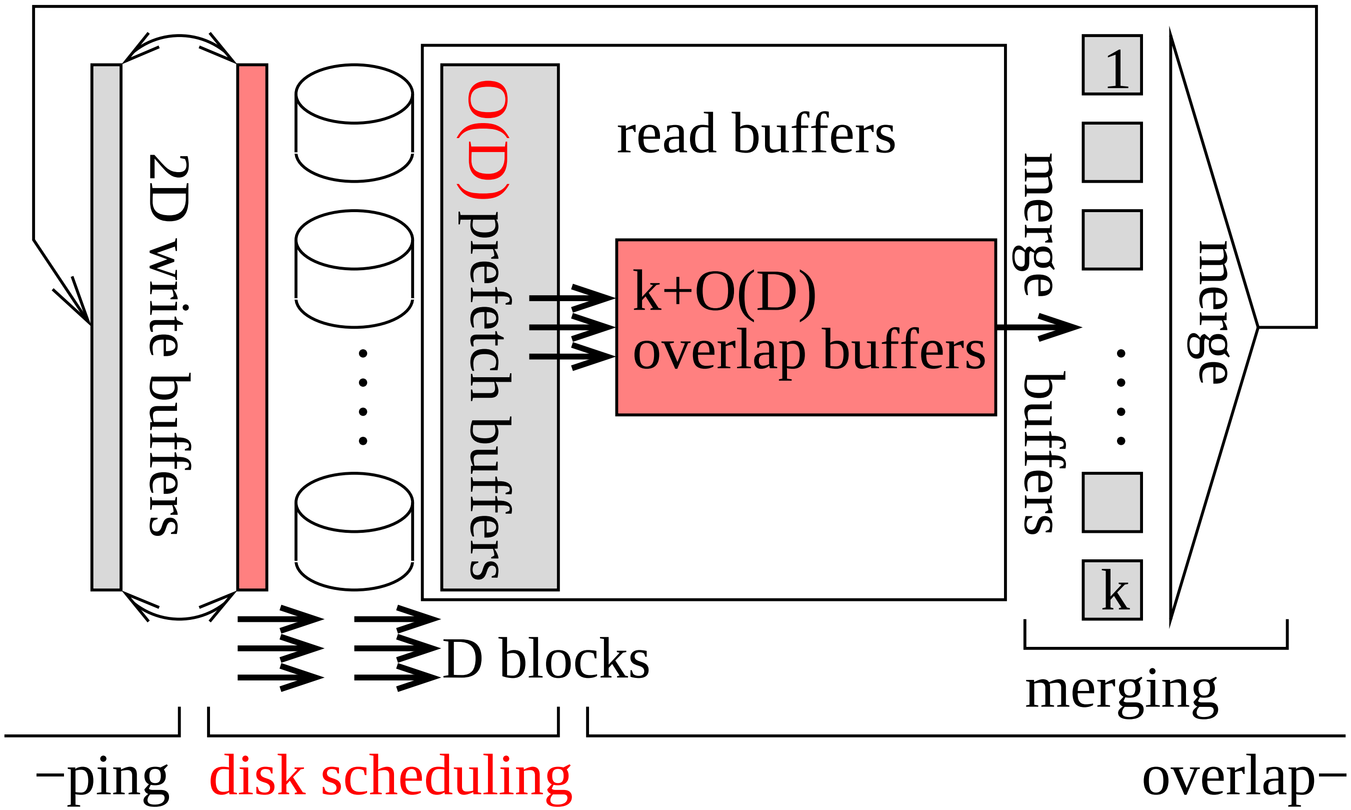
More than one merging phase?:

Not for the hierarchy main memory, hard disk.

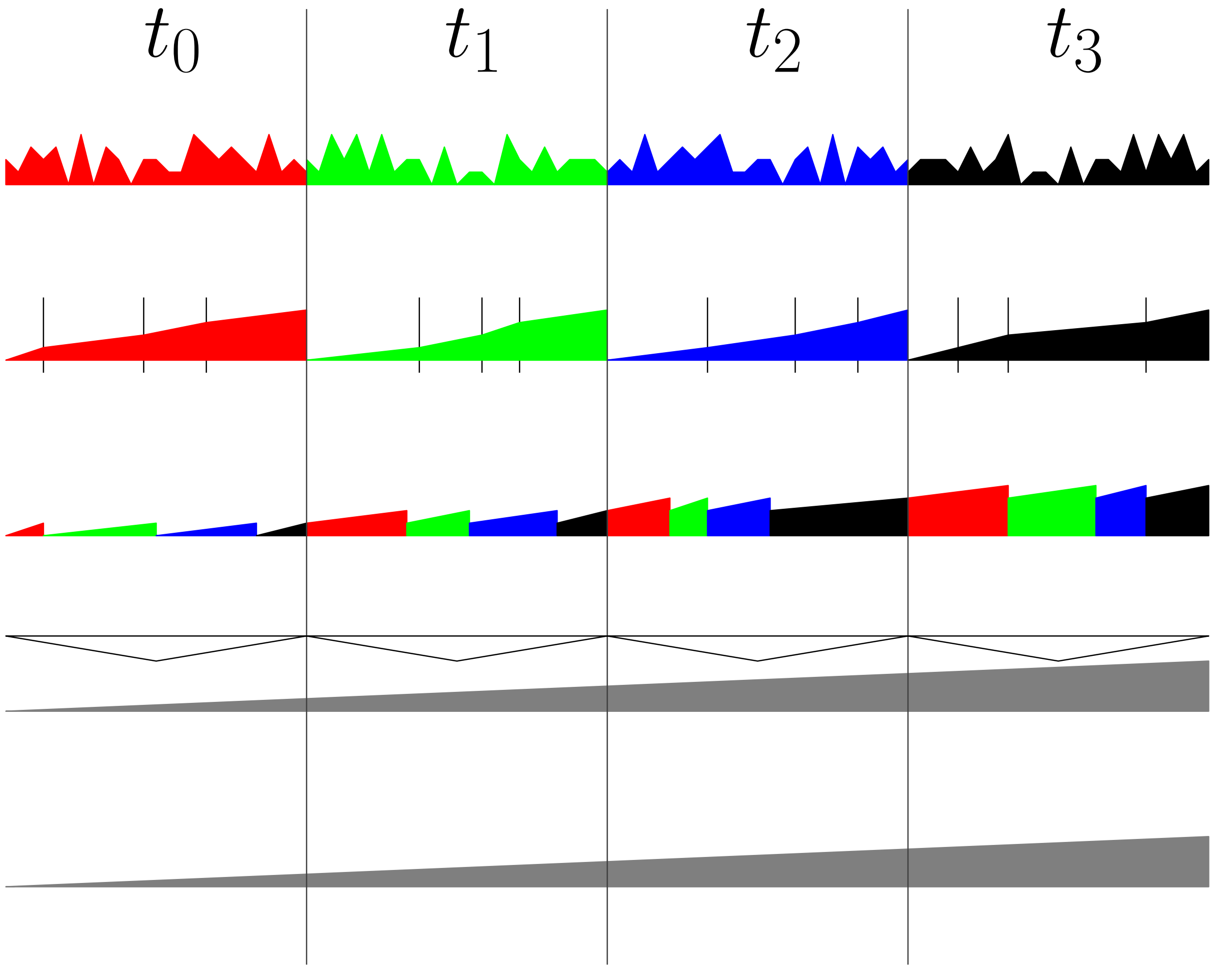
reason: $$\overbrace{\frac{M}{B}}^{>2000} > \overbrace{\frac{\text{RAM Euro}/\text{bit}}{\text{Platte Euro}/\text{bit}}}^{\approx 200}$$

More on Multiway Mergesort – Parallel Disks

- Randomized Striping [5]
- Optimal Prefetching [5]
- Overlapping of I/O and Computation [10]



Shared Memory Multiway Mergesort [11]



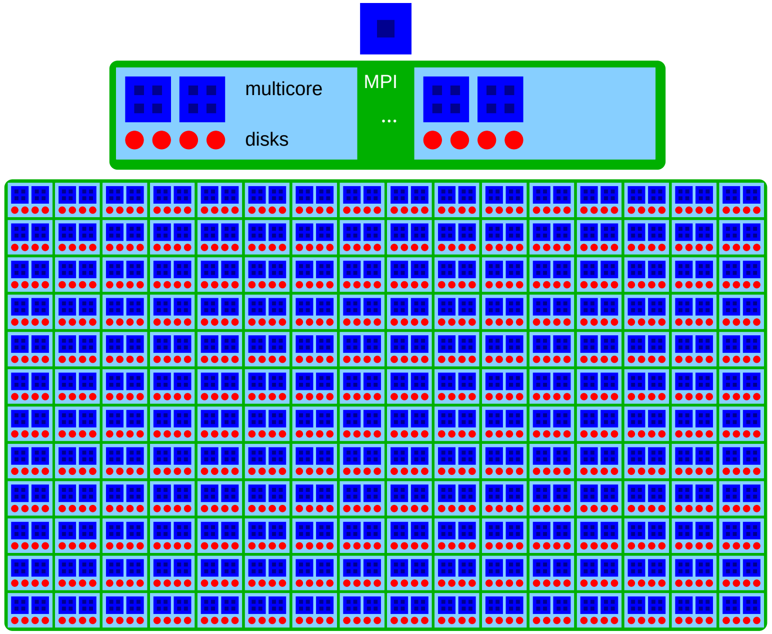
Combinations

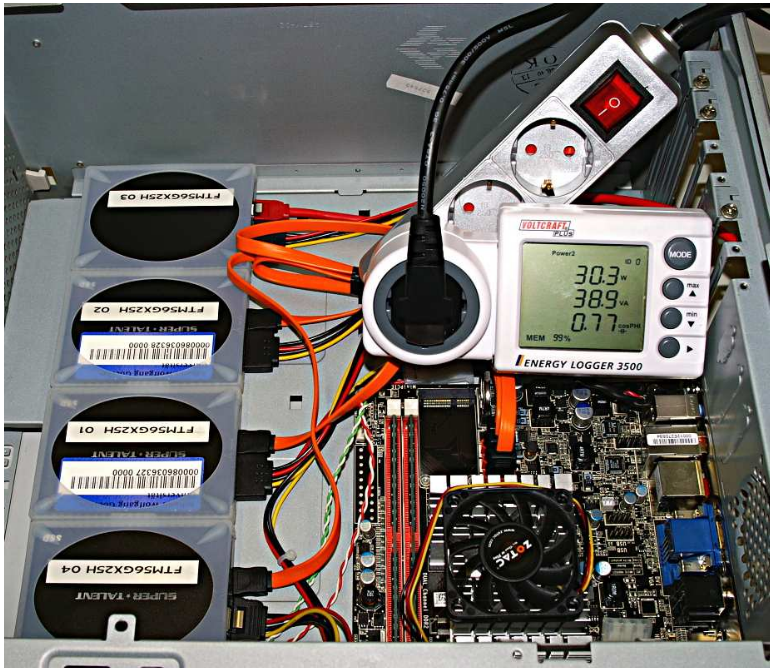
parallel disk + shared memory: [13]

+ distributed memory: [8] stay tuned

load balancing, randomization, collective communication

+ energy: [14] stay tuned





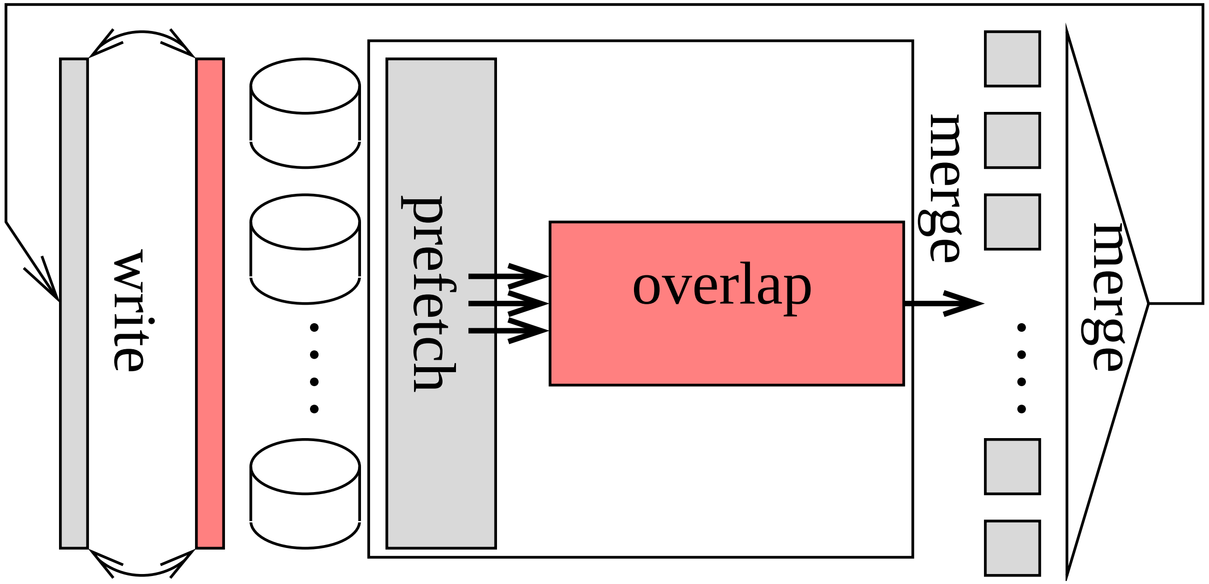
Analysis

- Constant factors matter

- Beyond worst case analysis

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

Analysis – Sorting



- Constant factors matter $(1+o(1))\times$lower bound
  [5, 8] I/Os for parallel (disk) external sorting

- Beyond worst case analysis

- Practical algorithms might be difficult to analyze Open Problem:
  [5] greedy algorithm for parallel disk prefetching [Knuth@48]

Implementation

sanity check for algorithms !

Challenges

Semantic gaps:

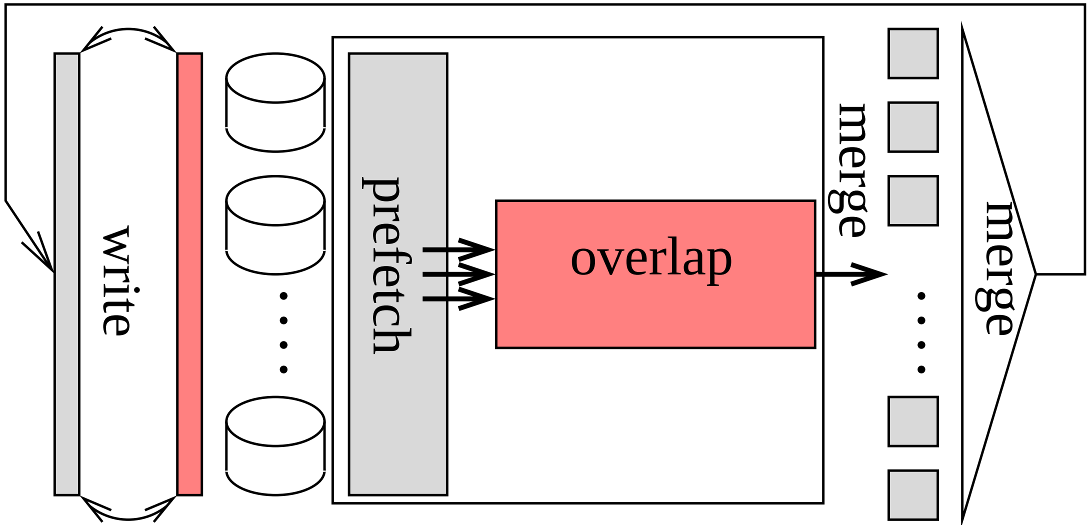
Abstract algorithm

$\leftrightarrow$

C++...

$\leftrightarrow$

hardware

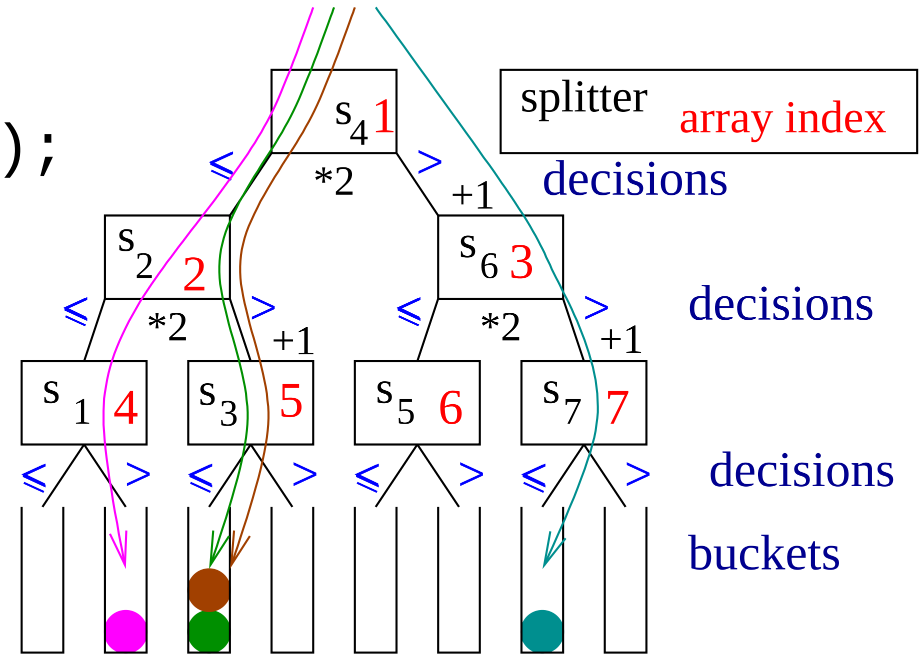


Small constant factors:

compare highly tuned competitors

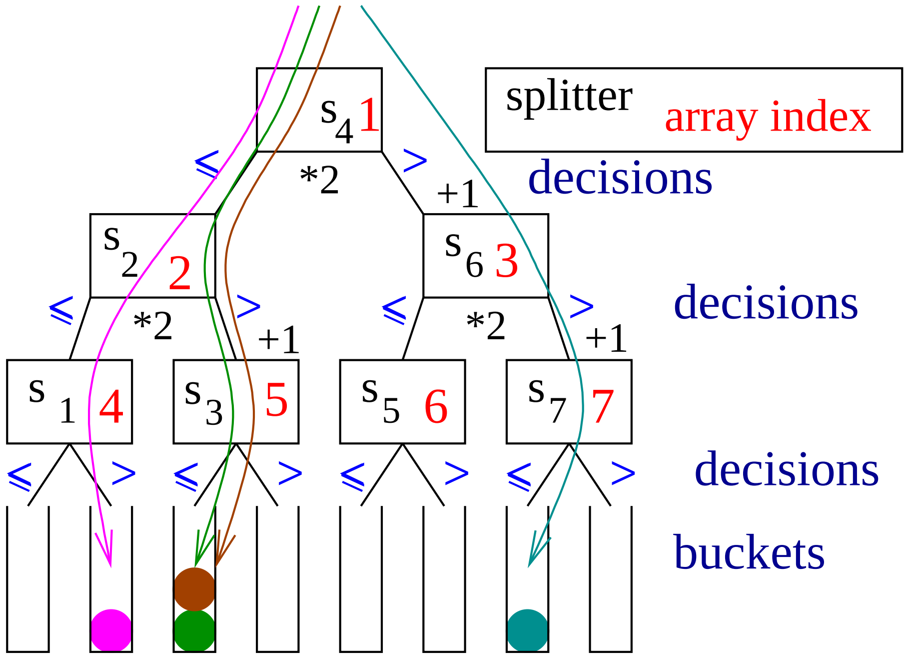
Example: Inner Loops Sample Sort [7]

```
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 [7]

```
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]);
    j = j*2 + (a[i] > s[j]);
    int b = j-k;
    bucket[b]++;
    oracle[i] = b;
} }
```



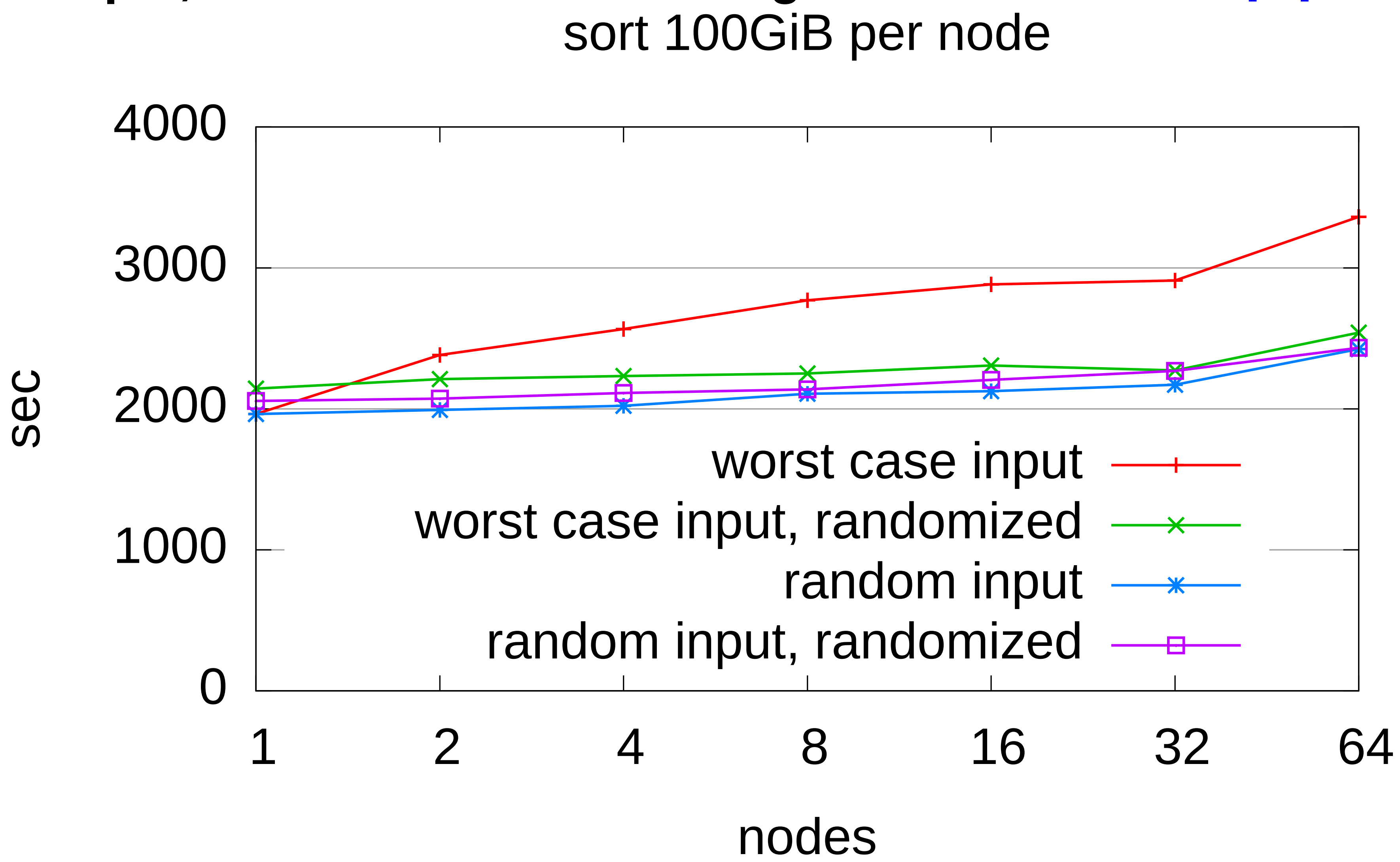
Example: Inner Loops Sample Sort [7]

```
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]);
      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;
} }
```

Experiments

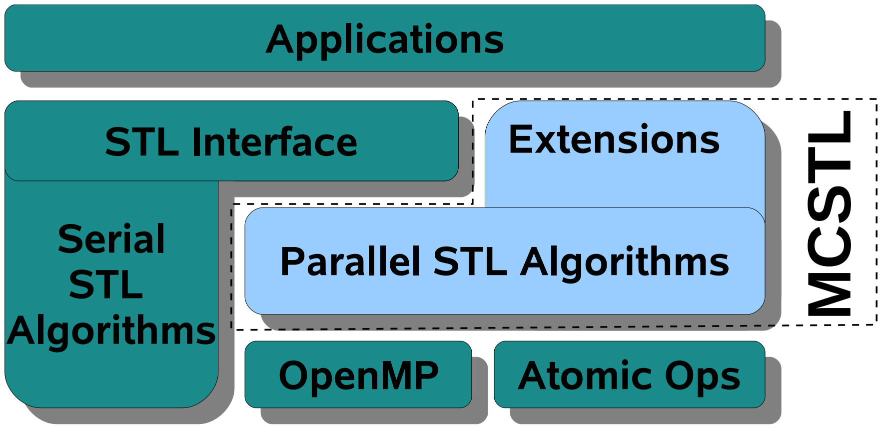
- sometimes a good surrogate for analysis
- too much rather than too little output data
- reproducibility (10 years!)
- software engineering

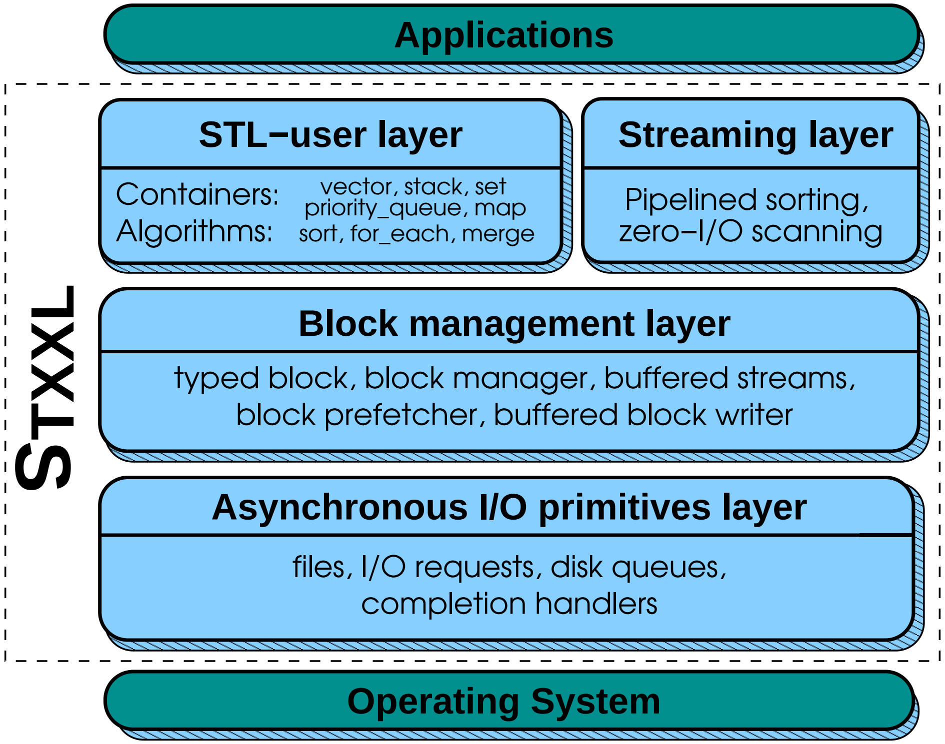
Example, Parallel External Sorting [10]



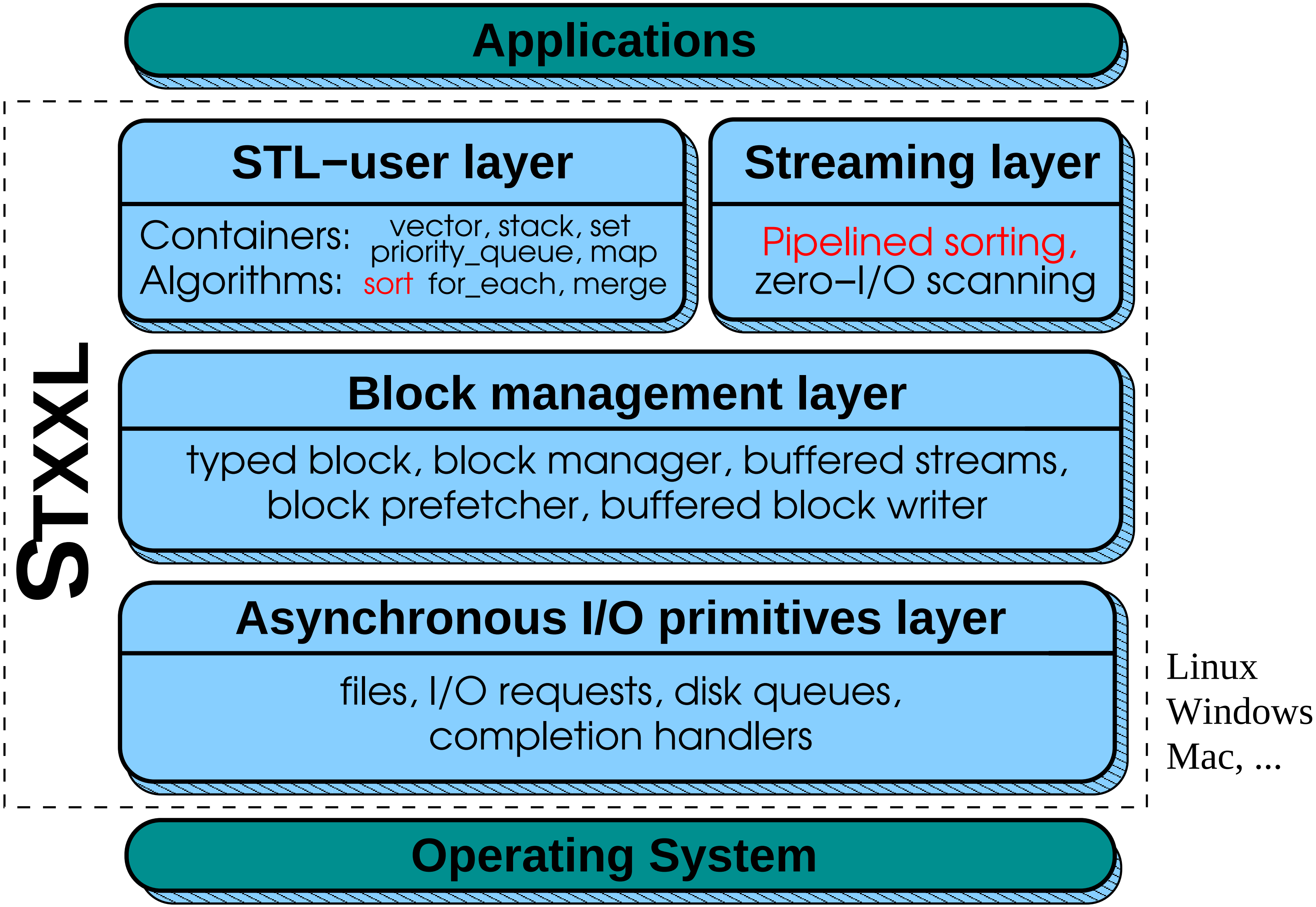
Algorithm Libraries — Challenges

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

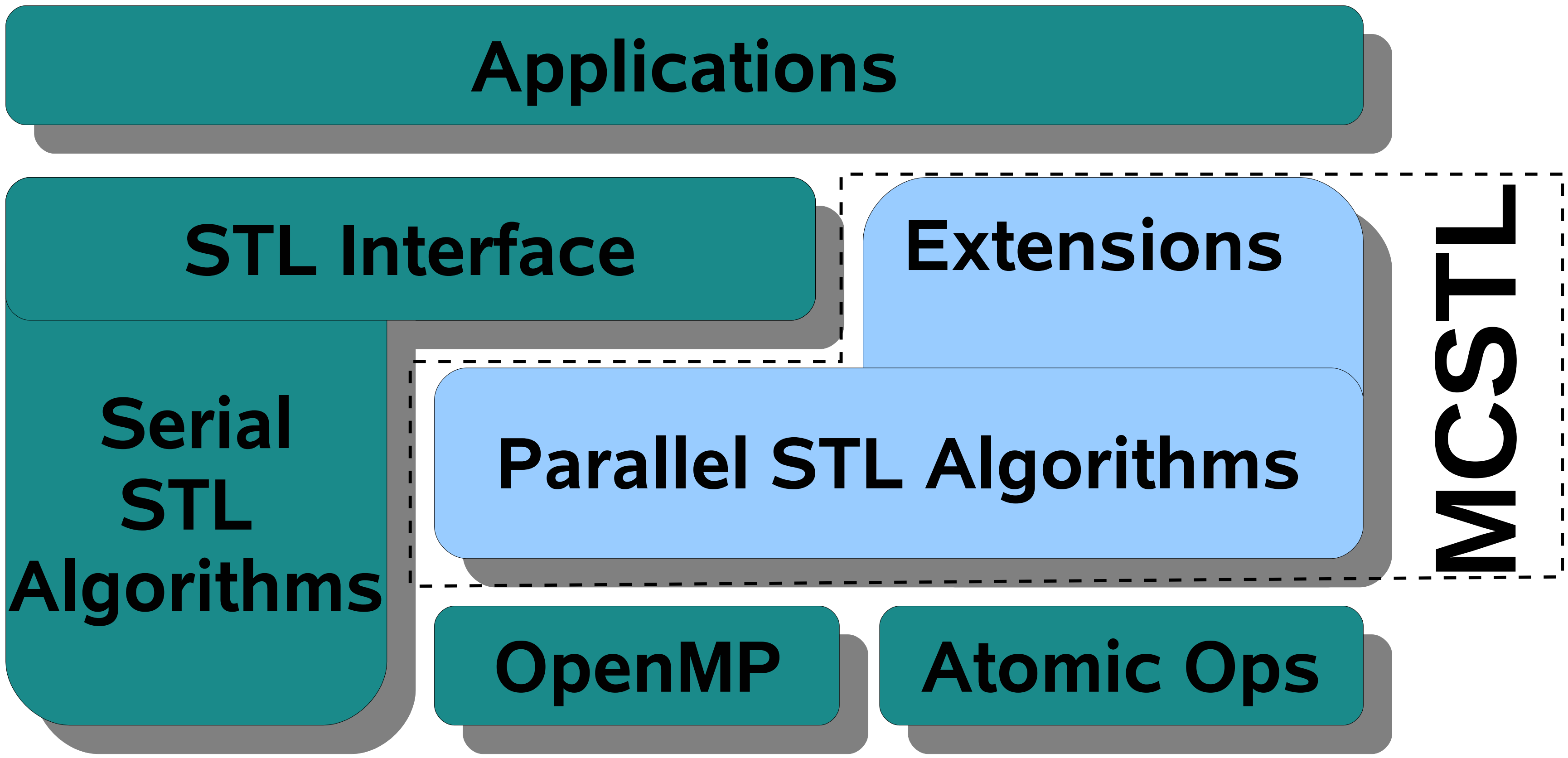




Example: External Sorting [10, 15]



Example: Shared Memory Sorting [11, 16]

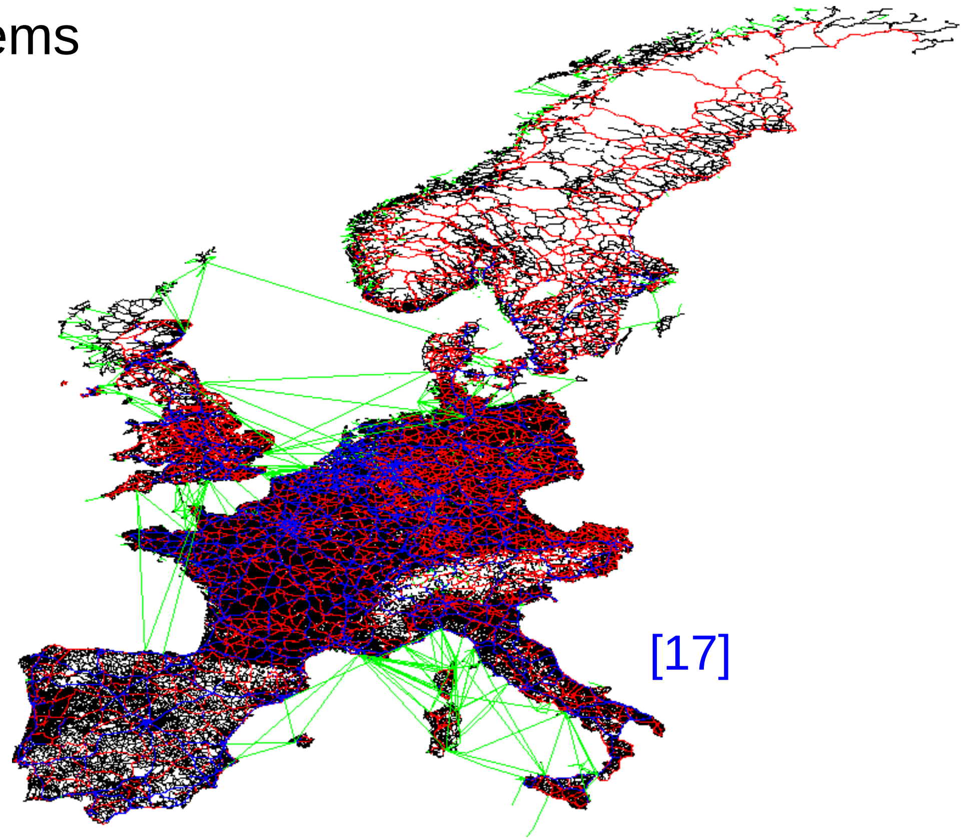


STL-alike $\ll$ STL-integrated

Problem Instances

Benchmark instances for NP-hard problems

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



[17]

have proved essential for development of practical algorithms

Strange: much less real world instances for polynomial problems

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

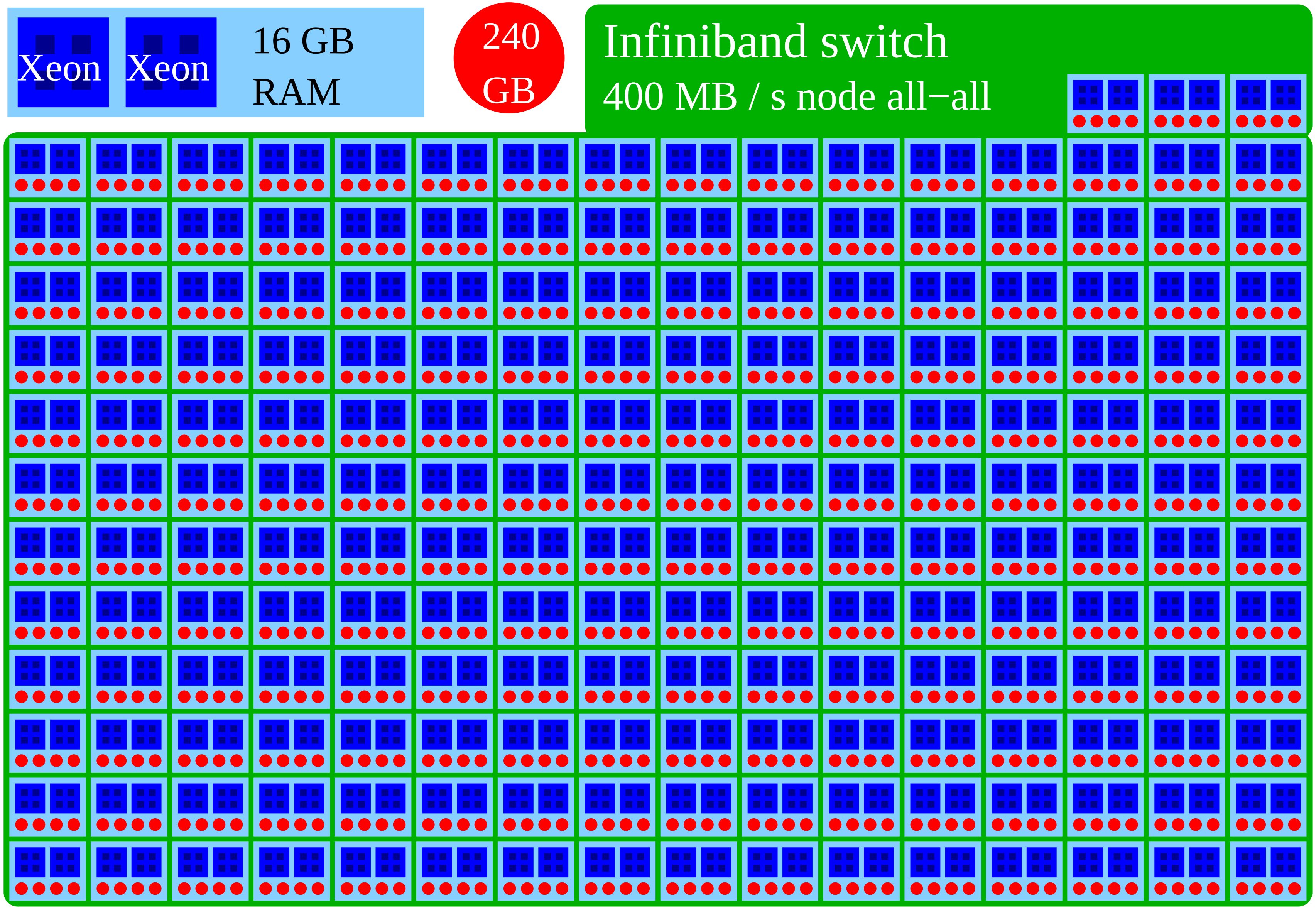
Example: Sorting Benchmark (Indy) [8, 14]

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

Category	data volume	performance	improvement
GraySort	100 TB	564 GB / min	$17\times$
MinuteSort	955 GB	955 GB / min	$> 10\times$
JouleSort	1 000 GB	13 400 Recs/Joule	$4\times$
JouleSort	100 GB	35 500 Recs/Joule	$3\times$
JouleSort	10 GB	34 300 Recs/Joule	$3\times$

Also: PennySort

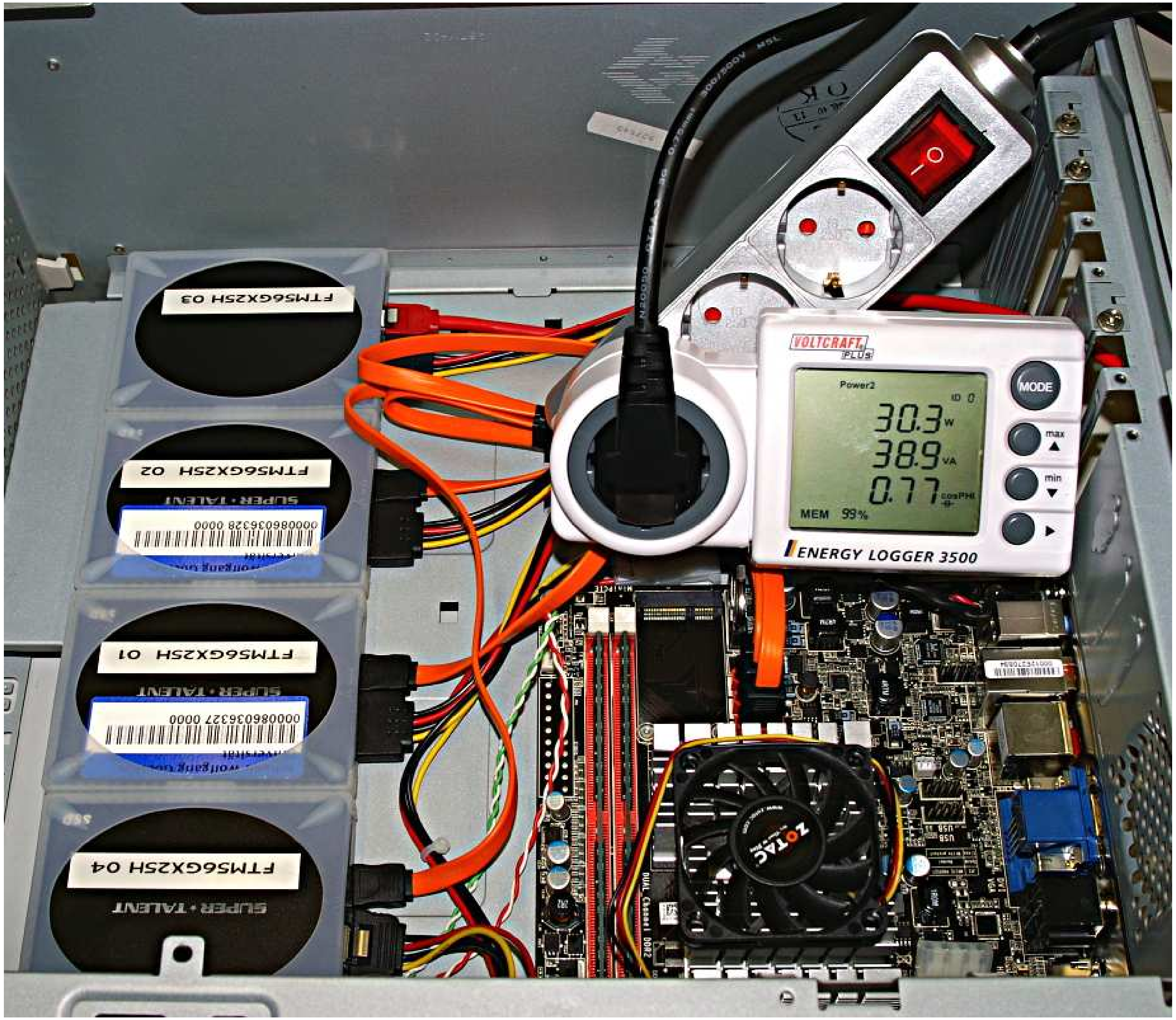
GraySort: inplace multiway mergesort, exact splitting [8]



JouleSort [14]

- 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

Geographic Information Systems: agriculture, environmental protection,
disaster management, tourism,. . .

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

AE for Big Data

Applications

- sensor data
- genomes
- data bases
- www
- GIS
- mobile
- ...

Techniques

- data structures
- graphs
- geometry
- strings
- coding theory
- ...

AE

Technology

- parallelism
- memory hierarchies
- communication
- fault tolerance
- energy

experience
PS

Larger Sorting Problems

- millions of processors

  $\rightsquigarrow$ multipass algorithms

- fault tolerance

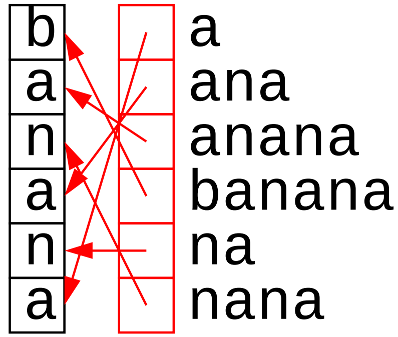
- still energy $\sim$ time?

Higly related to MapReduce, index construction,...

More Big Data Examples From my Group

- Suffix Sorting and its applications
- Main Memory Data Bases
- Graph Partitioning
- Track Reconstruction at CERN
- Route Planning
- Genome Sequencing
- Image Processing
- Priority Queues

Suffix Sorting



sort suffixes $s_i \cdots s_n$ of string

$S = s_1 \cdots s_n$, $s_i \in \{1..n\}$.

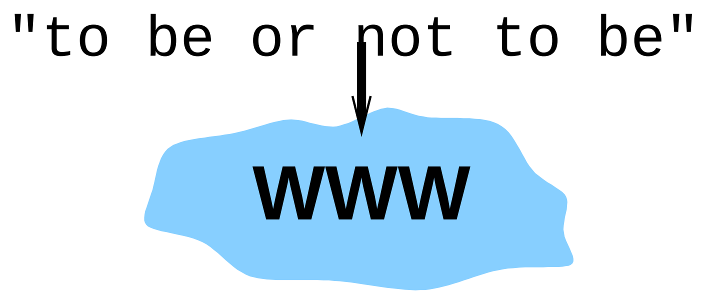
Applications: full text search,

Burrows-Wheeler text compression, bioinformatics,...

E.g. phrase search in time logarithmic

or even independent of input size.

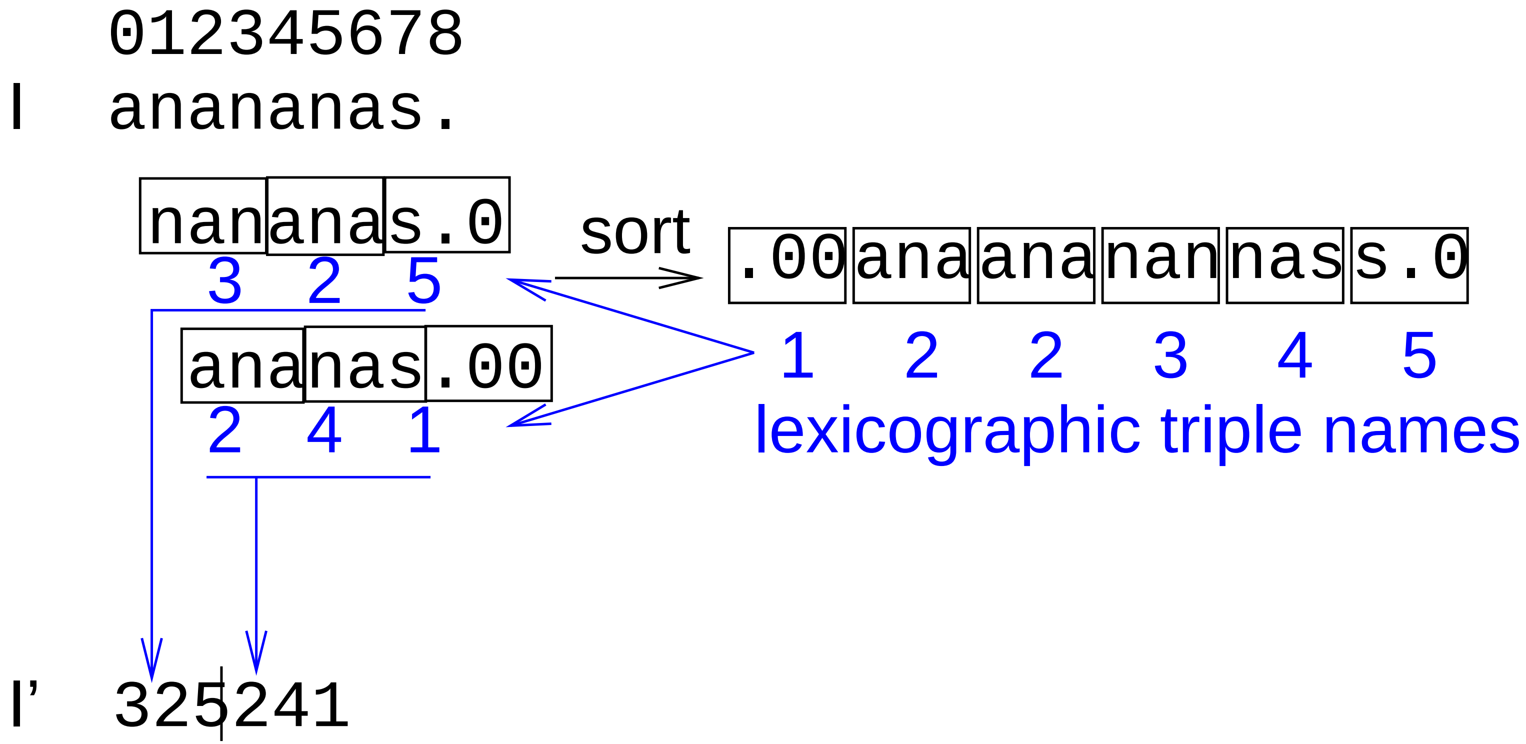
$\rightsquigarrow$ particularly interesting for large data



Linear Work Suffix Sorting [18]

simple: Radix-Sort + linear recursion + merging.

$\leadsto$ trivial external [19], parallel [20] adaptation

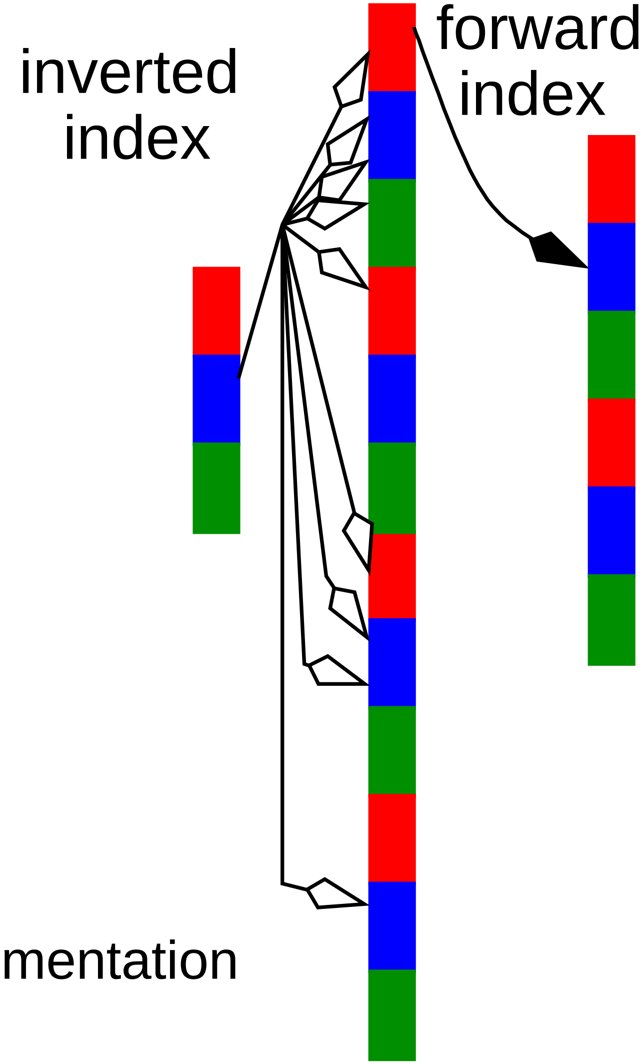


Current Work

- distributed memory (external) query
- parallel distributed construction of query data structure
  (longest common prefixes,. . .)

Data Bases – Our Approach [21, 22]

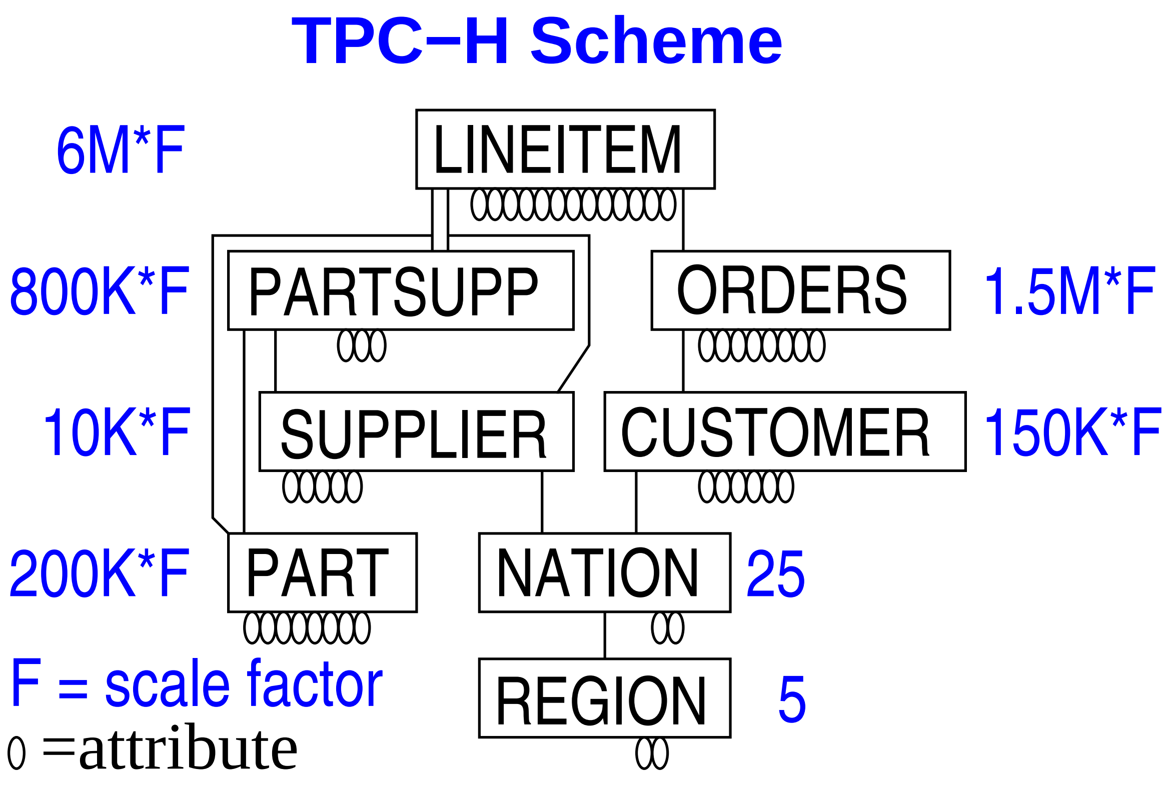
[with SAP HANA team, PhD students Dees, Müller]



- main memory based
- column based
- many-core machines
- NUMA-aware
- no precomputed aggregates
- aggressive indexing
- generate C++ code close to tuned manual implementation

TPC-H Decision Support Benchmark

- 22 realistic queries of varying complexity
- pseudorealistic random data
- F GByte space



Typical TPC-H Queries

Q1: Revenue etc. of all shipped LINEITEMs

(aggregated into 6 categories)

⇝ plain flat scan of all LINEITEMs

Q9: Sum profit for all LINEITEMs with a given color

for each nation and order year.

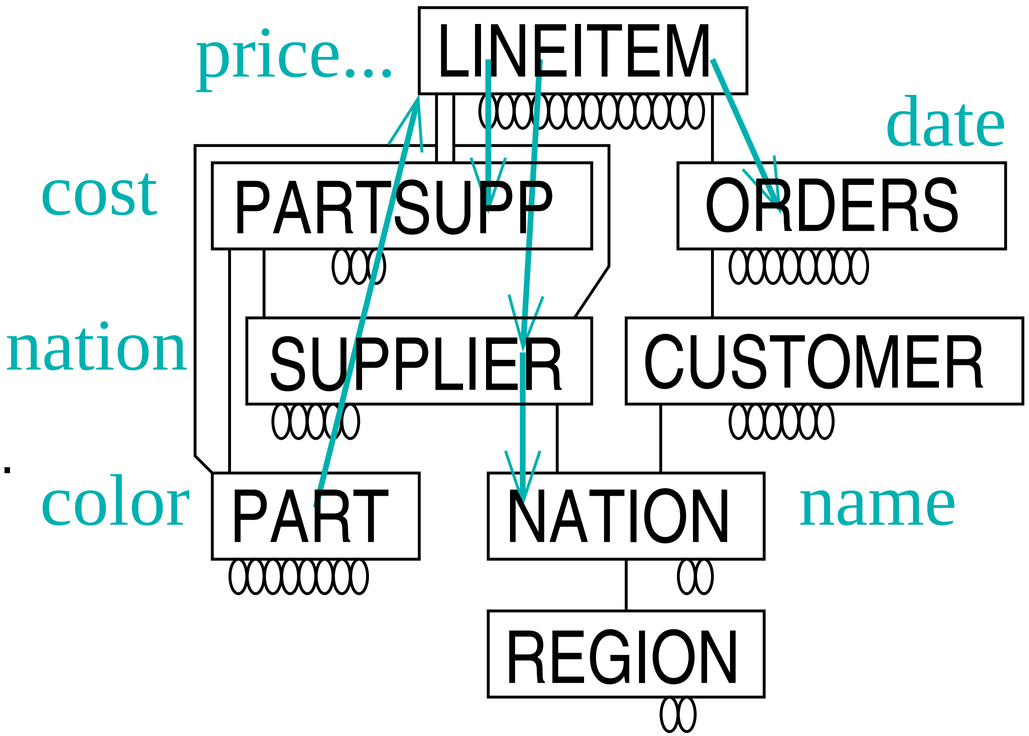
⇝ scan PARTs,

use inverted index

to access matching LINEITEMs

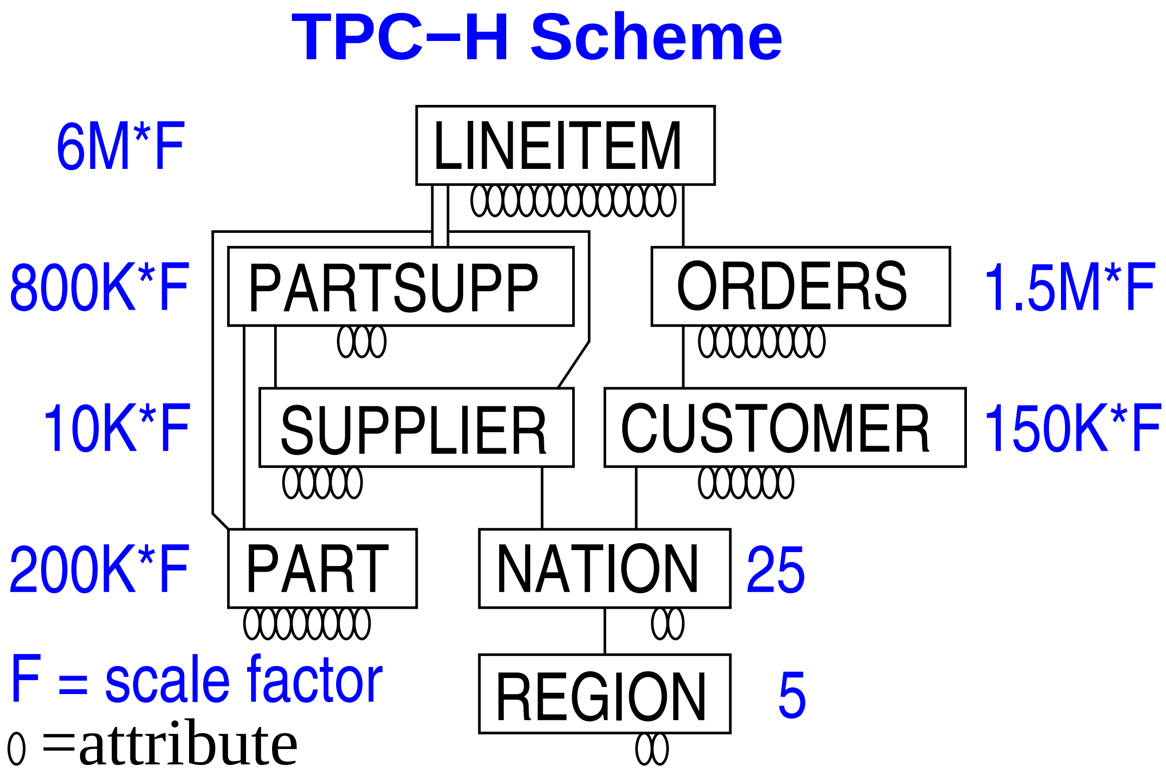
Go down from there

using forward indices (≈pointers).

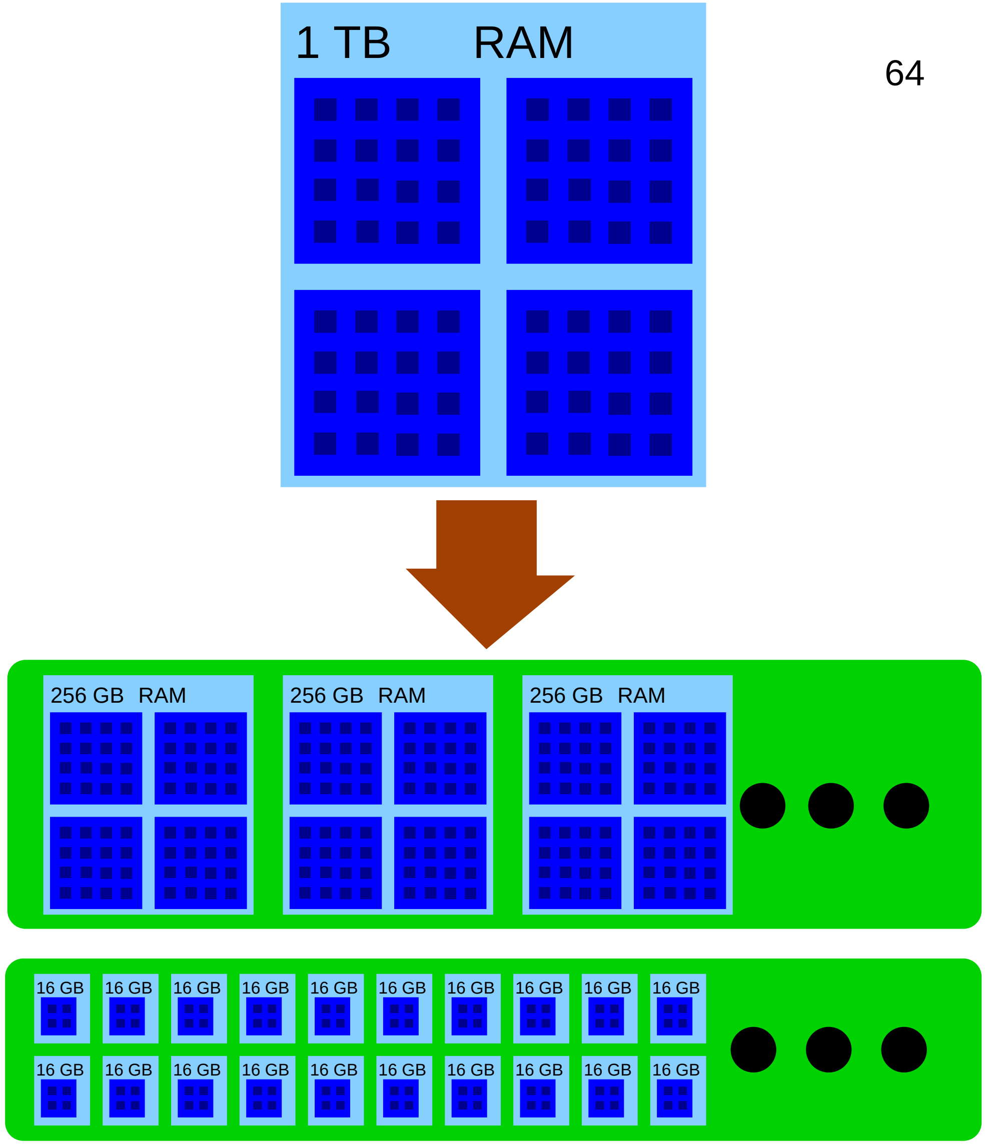


First Results [21]

- $\approx 30\times$ faster than current record in 300GB category (manual implementation)
- Compiler: seems to be largely orthogonal to algorithmic and parallelization issues



Larger Inputs



- Already needed by some large customers of SAP
- Move to clusters
  Master thesis Martin Weidner seems to give positive results (5 TPC-H queries) [22]
- fault tolerance beyond recovery?
- energy efficiency using many small nodes (ARM)?

Algorithmic Meat: Randomization, collective communication, communication complexity, sorting, data structures, multi-level memory hierarchies, coding theory

Graph Partitionierung [23, 24]

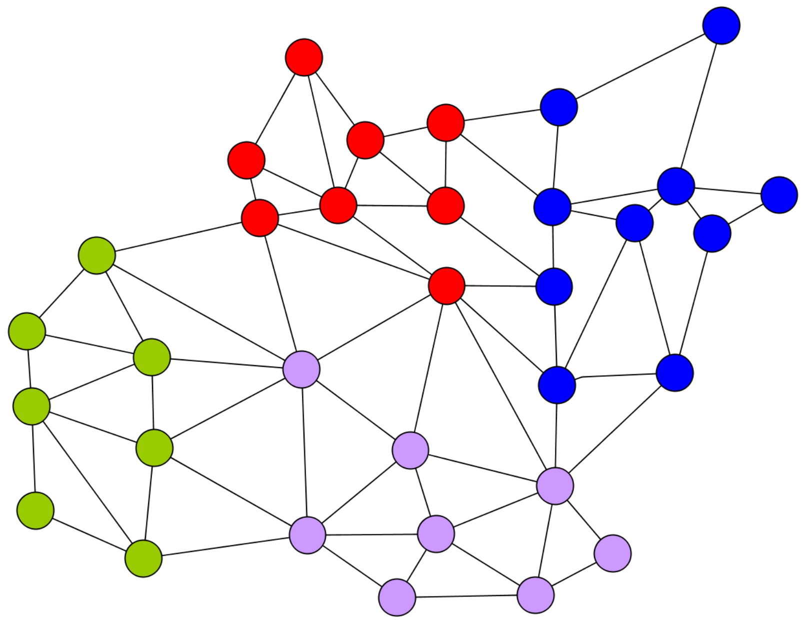
Input: Graph (V, E) (possibly with node and edge weights), ϵ, k

Output: $V_1 \dot\cup \cdots \dot\cup V_k$ mit $|V_i| \leq (1+\epsilon)\left\lceil \frac{|V|}{k} \right\rceil$

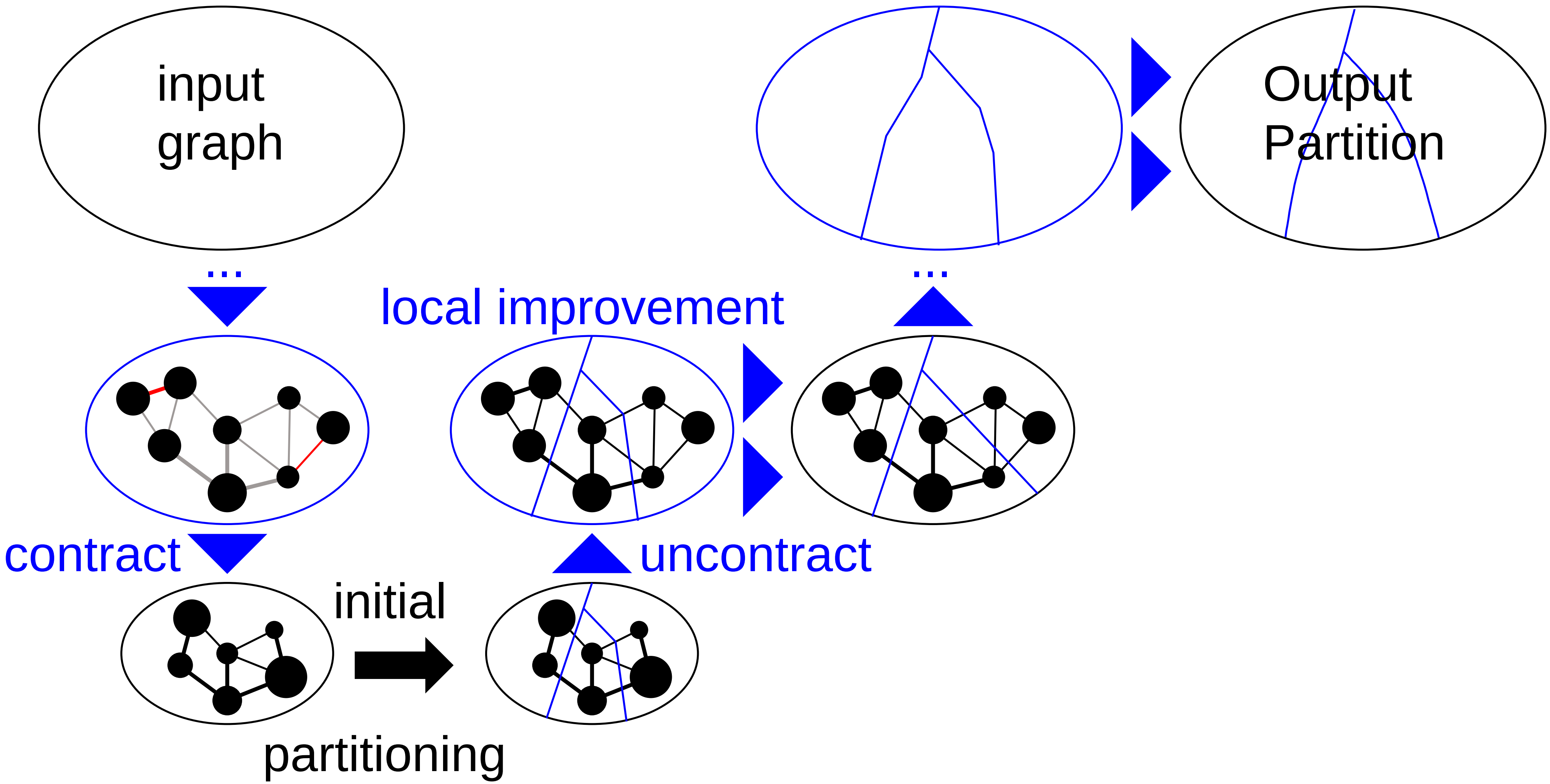
Objective Function: minimize cut

Applications: finite element simulations, VLSI-design, route planning,...

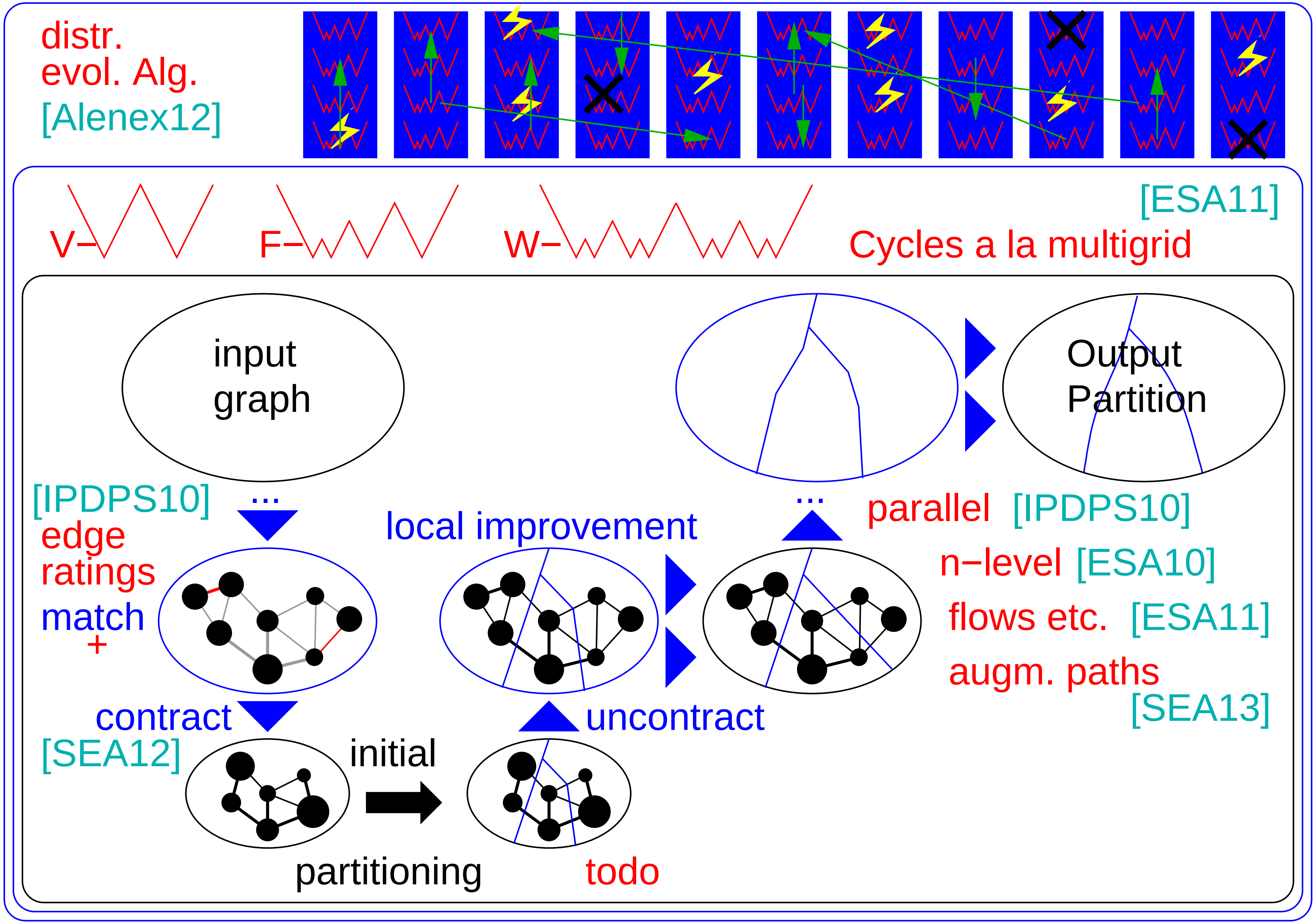
Variants: hypergraphs, clustering, different objective functions,...



Multilevel Graph Partitioning



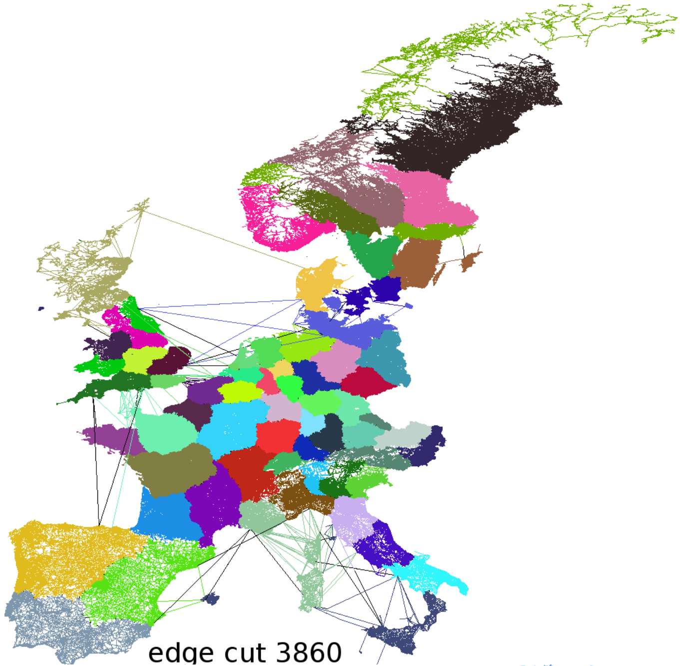
Reengineering Multilevel Graph Partitioning

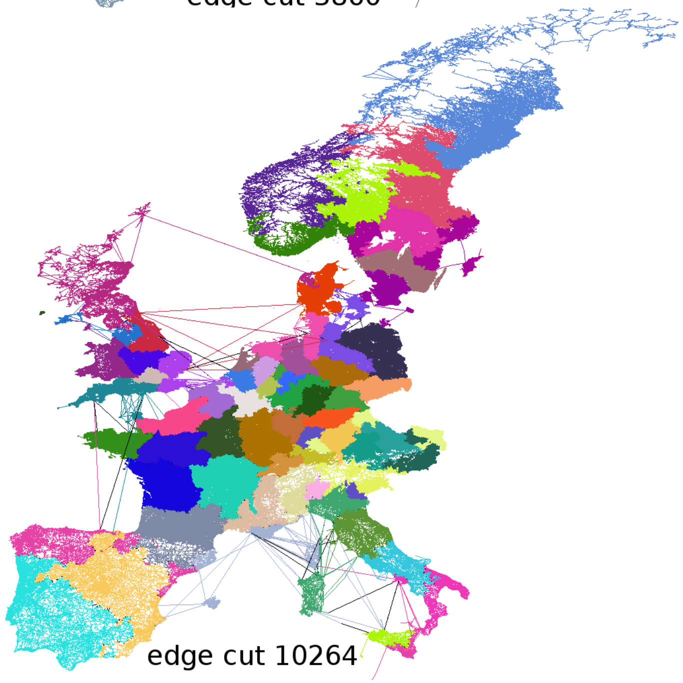


Our Contribution

- scalable parallelization KaPPa
  (matching, edge coloring, evolutionary)
- thorough reengineering of multilevel approch
  (use flows, SCCs, BFS, matching, edge coloring, negative cycle detection, . . .)

⇝ high quality (e.g. 90–99% entries in Walshaw's benchmark)





Large Data Graph Partitioning

- difficult inputs: social networks, WWW, 3D/4D models, VLSI, knowledge graph?
- more difficult parallelization

Future Work

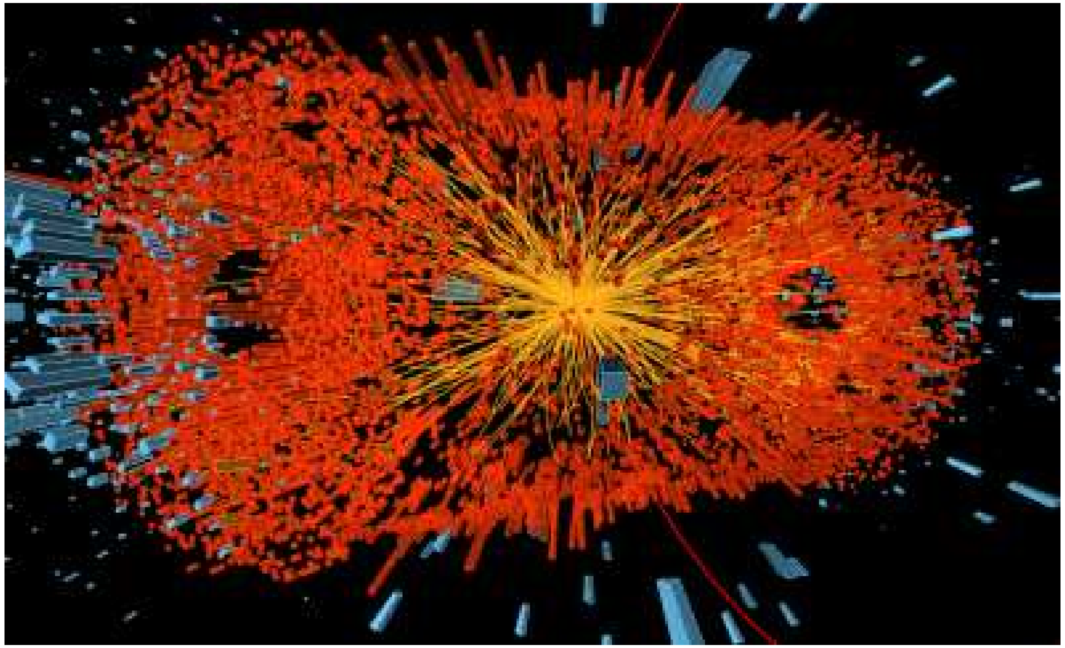
- parallel external
- other variants
- fault tolerant
- component of a graph processing framework

Track reconstruction [25]

Input: clouds of $\approx 10^4$ 3D points

Output: $< 10^3$ spiral tracks of high energy particles

Also cluster tracks by emergence point



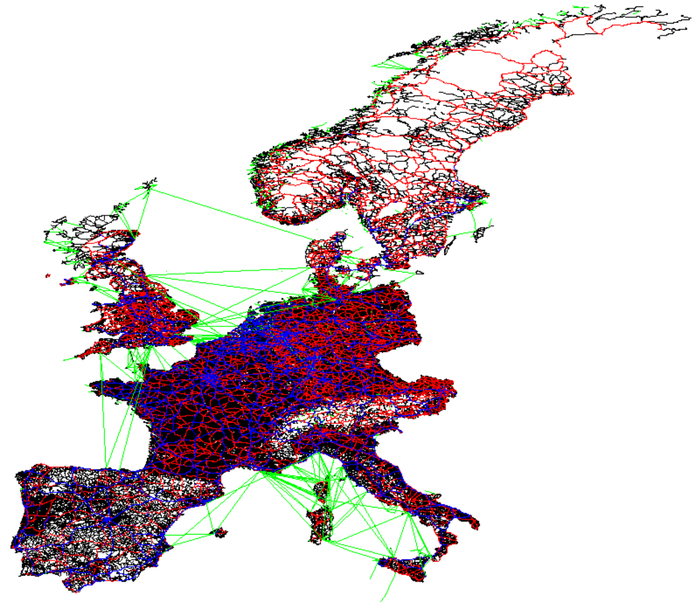
Large Data???

- up to 10^5 instances / s
- cost of processors / energy
- memory constrained
- exploit SIMD/GPU parallelism?

Algorithmic Meat:

Geometric data structures, parallelization, clustering

Route Planning



Large Data 2004: Western European network (18M nodes).

Dijkstra's algorithm needs $6s$.

- too much time for servers
- too much memory for mobile devices

$\rightsquigarrow$ inaccurate heuristics with tedious "manual preprocessing"

Our contribution: Automatic preprocessing techniques

- 10^4–10^6 times faster exact query on servers
- still "instantaneous" on mobile devices (external implementation)

Large Data 2013

- 1.6G nodes OpenStreetMap routing graph (edge based)
- billions of GPS traces
  (+ road based sensors + elevation data)
- public transportation

Potential use:

- time-dependent edge weights [27]
- detailed traffic jam detection Google, TomTom,...
- multi-modal route planning [28]
- probabilistic route planning attempts
- really useful detours around traffic jams ???
  use real time traffic simulation??

Genome Sequencing

[29]: 20 000 CPU hours for shotgun sequencing of the human genome

($3 \cdot 10^9$ base pairs, 5–10 times oversampling.

Prototypical large data problem?

Today: a few minutes on a work station [ZieglerDFMS work in progr.]

(use template, modern hardware, AE $+$ cheap sequencing)

$\rightsquigarrow$ routine use for personal medicine

New Challenge:

processing many sequences

Phylogenetic Tree Reconstruction

Image Processing [30]

Gigapixel aerial images.

Filters, Segmentation, Change detection

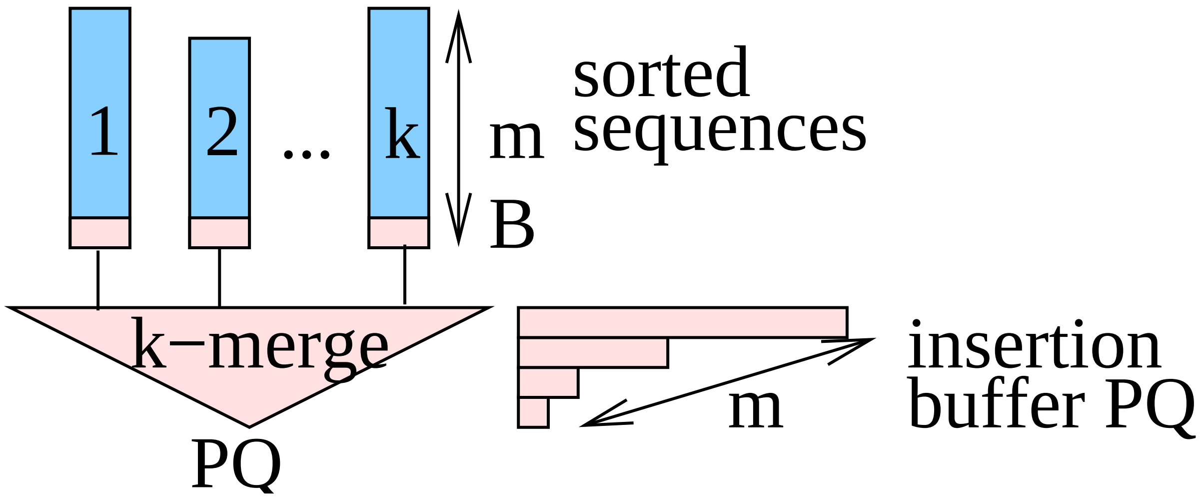
Algorithmic meat: Graph algorithms, parallelization, memory hierarchies, range minimum data structures,...

External Priority Queues

Problem: Binary heaps need
$\Theta\left(\log \frac{n}{M}\right)$ I/Os per deleteMin

We would rather have:
$\Theta\left(\frac{1}{B} \log_{M/B} \frac{n}{M}\right)$ I/Os (amortized)

Medium Size PQs – $km \ll M^2/B$ Insertions



Insert: Initially into insertion buffer.

Overflow $\longrightarrow$

sort; flush; smallest key is now in merge PQ

Delete-Min: deleteMin from the PQ with smaller $\min$

Large Queues

$$\approx \frac{2n}{B}\left(1+\left\lceil\log_{M/B}\frac{n}{M}\right\rceil\right)$$

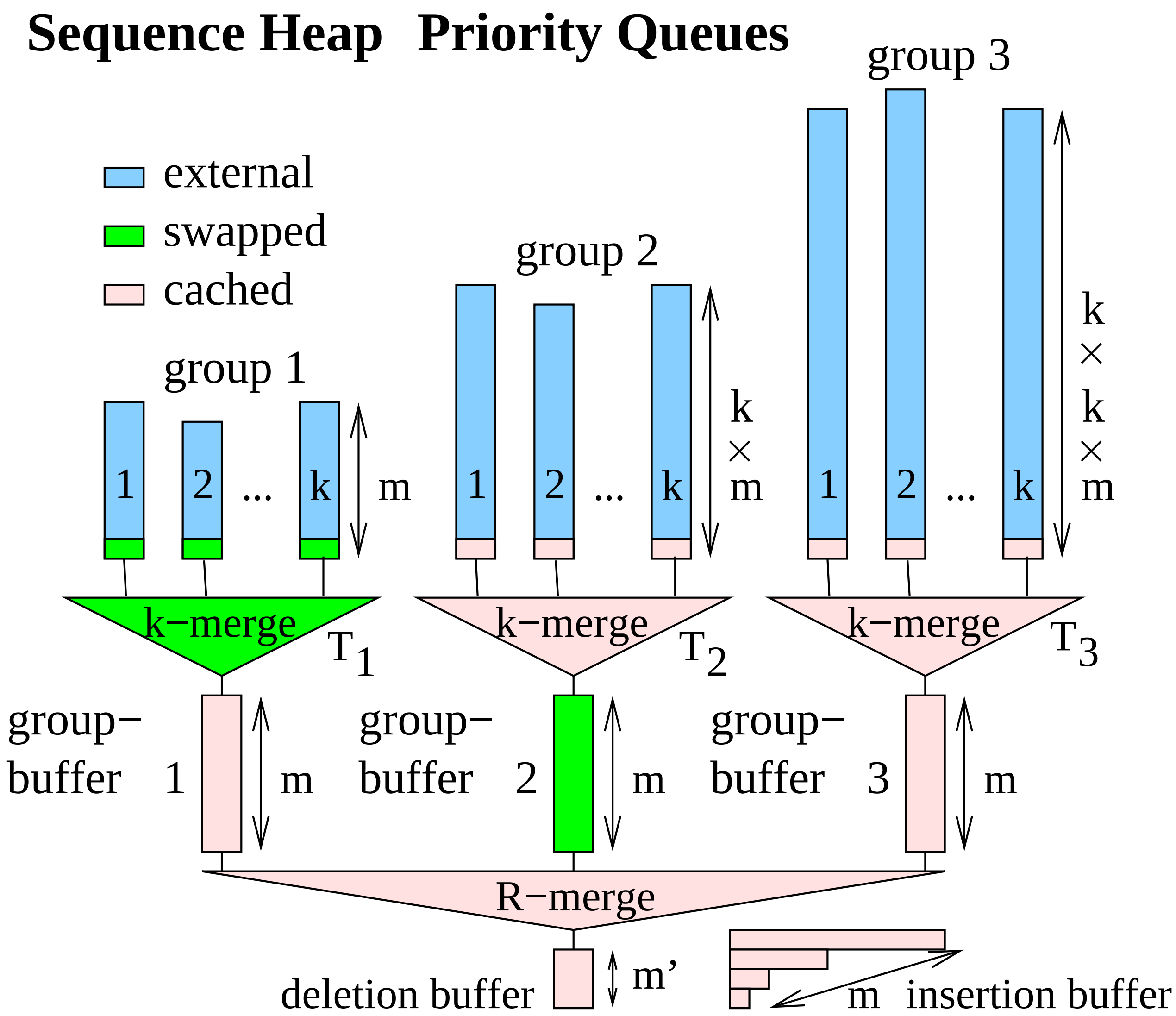
I/Os for n insertiosn

$\mathcal{O}(n\log n)$ Arbeit.

[31].

deleteMin:

“amortisiert umsonst”.



Experiments

Keys: random 32 bit integers

Associated information: 32 dummy bits

Deletion buffer size: 32

Group buffer size: 256

Merging degree k: 128

Near optimal
: performance on
all machines tried!

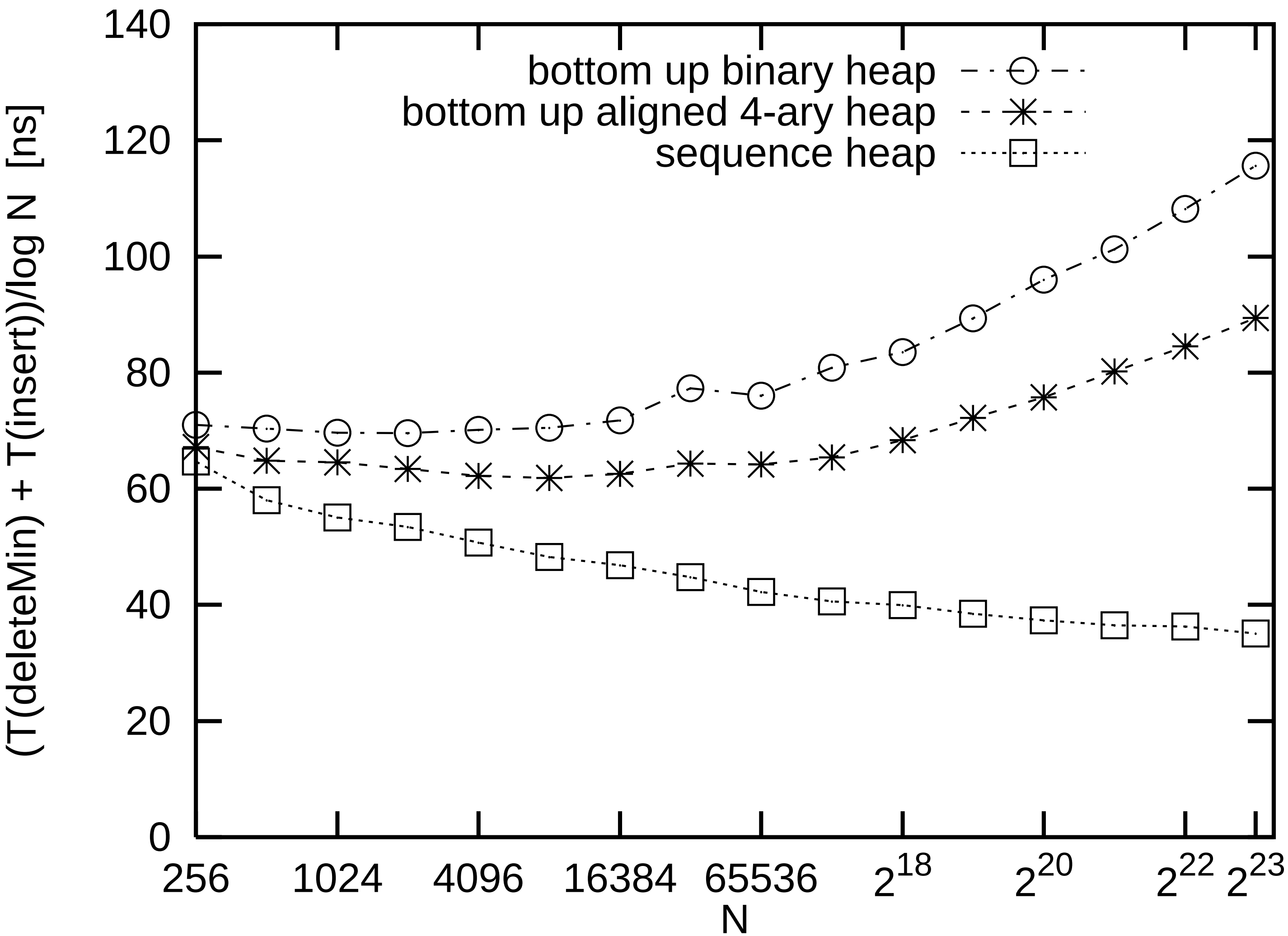
Compiler flags: Highly optimizing, nothing advanced

Operation Sequence:

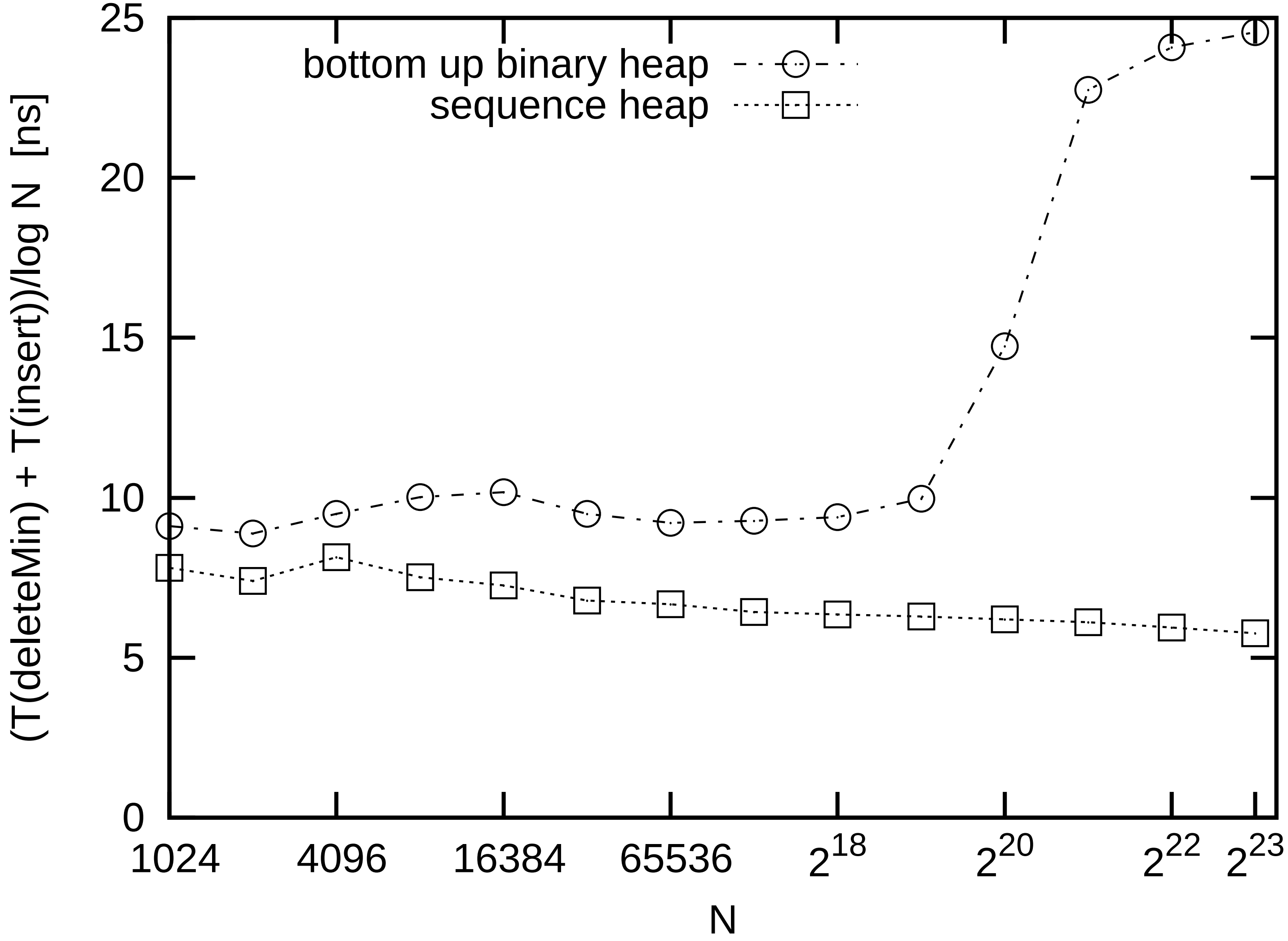
$$(\text{Insert-DeleteMin-Insert})^N(\text{DeleteMin-Insert-DeleteMin})^N$$

Near optimal performance on all machines tried!

Alpha-21164, 533 MHz



Core2 Duo Notebook, 1.??? GHz



Future Work

- see above
- find more algorithmic application problems
- algorithmic cores of application independent libraries and tools

  data structures, MapReduce, graphs, data bases,. . .
- distributed memory external algorithms
- back to massive parallelism including exascale
- fault tolerance

Commercial Break

I am hiring

PhD students, Postdocs in algorithm engineering.

Desirable Skills:

- Desire to bridge gaps between theory and practice
- Algorithmics
- Performance oriented C++ programming
- Parallelization, e.g., MPI, OpenMP,. . .

Literatur

[1] P. Sanders. Algorithm engineering – an attempt at a definition. In *Efficient Algorithms*, volume 5760 of *LNCS*, pages 321–340. Springer, 2009.

[2] K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[3] U. Meyer, P. Sanders, and J. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS Tutorial*. Springer, 2003.

[4] Peter Sanders, Sebastian Schlag, and Ingo Müller. Communication efficient algorithms for fundamental big data problems. In *IEEE Int. Conf. on Big Data*, 2013.

[5] D. A. Hutchinson, P. Sanders, and J. S. Vitter. Duality between prefetching and queued writing with parallel disks. *SIAM Journal on Computing*, 34(6):1443–1463, 2005.

[6] K. Mehlhorn and P. Sanders. Scanning multiple sequences via cache memory. *Algorithmica*, 35(1):75–93, 2003.

[7] P. Sanders and S. Winkel. Super scalar sample sort. In *12th European Symposium on Algorithms*, volume 3221 of *LNCS*, pages 784–796. Springer, 2004.

[8] M. Rahn, P. Sanders, and J. Singler. Scalable distributed-memory external sorting. In *26th IEEE International Conference on Data Engineering*, pages 685–688, 2010.

[9] N. Leischner, V. Osipov, and P. Sanders. GPU sample sort. *CoRR*, abs/0909.5649, 2009. submitted for publication.

[10] R. Dementiev and P. Sanders. Asynchronous parallel disk sorting. In *15th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 138–148, San Diego, 2003.

[11] J. Singler, P. Sanders, and F. Putze. MCSTL: The multi-core standard template library. In *13th International Euro-Par Conference*, volume 4641 of *LNCS*, pages 682–694. Springer, 2007.

[12] K. Mehlhorn and P. Sanders. *Algorithms and Data Structures — The Basic Toolbox*. Springer, 2008.

[13] A. Beckmann, R. Dementiev, and J. Singler. Building a parallel pipelined external memory algorithm library. In *23rd IEEE International Symposium on Parallel and Distributed Processing*, pages 1–10, 2009.

[14] A. Beckmann, U. Meyer, P. Sanders, and J. Singler. Energy-efficient sorting using solid state disks. In *1st International Green Computing Conference*, pages 191–202. IEEE, 2010.

[15] R. Dementiev, L. Kettner, and P. Sanders. STXXL: Standard Template Library for XXL data sets. *Software Practice & Experience*, 38(6):589–637, 2008.

[16] J. Singler and B. Kosnik. The libstdc++ parallel mode: Software engineering considerations. In *International Workshop on Multicore Software Engineering (IWMSE)*, 2008.

[17] David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, editors. *Graph Partitioning and Graph Clustering – 10th DIMACS Implementation Challenge Workshop*, volume 588 of *Contemporary Mathematics*. American Mathematical Society, 2013.

[18] J. Kärkkäinen, P. Sanders, and S. Burkhardt. Linear work suffix array construction. *Journal of the ACM*, 53(6):1–19, 2006.

[19] R. Dementiev, J. Kärkkäinen, J. Mehnert, and P. Sanders. Better external memory suffix array construction. *ACM Journal of Experimental Algorithmics*, 12, 2008. Special issue on Alenex 2005.

[20] F. Kulla and P. Sanders. Scalable parallel suffix array construction. *Parallel Computing*, 33:605–612, 2007. Special issue on Euro PVM/MPI 2006, distinguished paper.

[21] Jonathan Dees and Peter Sanders. Efficient many-core query execution in main memory column-stores. In *29th IEEE Conference on Data Engineering*, 2013.

[22] Martin Weidner, Jonathan Dees, and Peter Sanders. Fast olap query execution in main memory on large data in a cluster. In *IEEE Int. Conf. on Big Data*, 2013.

[23] Vitaly Osipov, Peter Sanders, and Christian Schulz. Engineering graph partitioning algorithms. In *11th International Symposium on Experimental Algorithms (SEA)*, volume 7276 of *LNCS*, pages 18–26. Springer, 2012.

[24] Christian Schulz. *High Quality Graph Partitioning*. PhD thesis, Karlsruhe Institute of Technology, 2013.

[25] Daniel Funke. Parallel triplet finding for particle track reconstruction. Master's thesis, Karlsruhe Institute of Technology, 2013.

[26] D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering route planning algorithms. In *Algorithmics of Large and Complex Networks*, volume 5515 of *LNCS State-of-the-Art Survey*, pages 117–139. Springer, 2009.

[27] Gernot Veit Batz and Peter Sanders. Time-dependent route planning with generalized objective functions. In *20th European Symposium on Algorithme (ESA)*, volume 7501 of *LNCS*, pages 169–180. Springer, 2012.

[28] Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast routing in very large public transportation networks using transfer patterns. In *18th European Symposium on Algorithms*, volume 6346 of *LNCS*, pages 290–301, 2010.

[29] J Craig Venter. Sequencing the human genome. In *Proceedings of the sixth annual international conference on Computational biology*, pages 309–309. ACM, 2002.

[30] Jan Wassenberg. *Efficient Algorithms for Large-Scale Image Analysis*. PhD thesis, Karlsruhe Institute of Technology, 2011.

[31] P. Sanders. Fast priority queues for cached memory. *ACM Journal of Experimental Algorithmics*, 5, 2000.