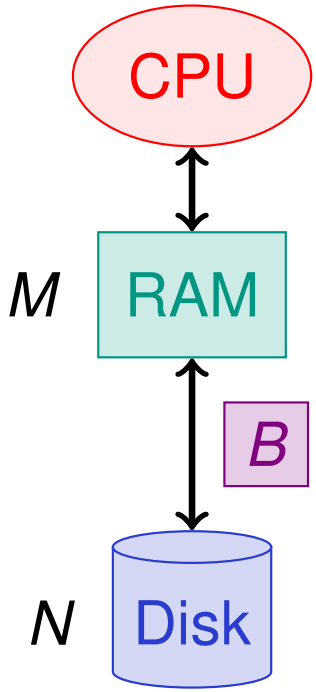
Karlsruhe Institute of Technology

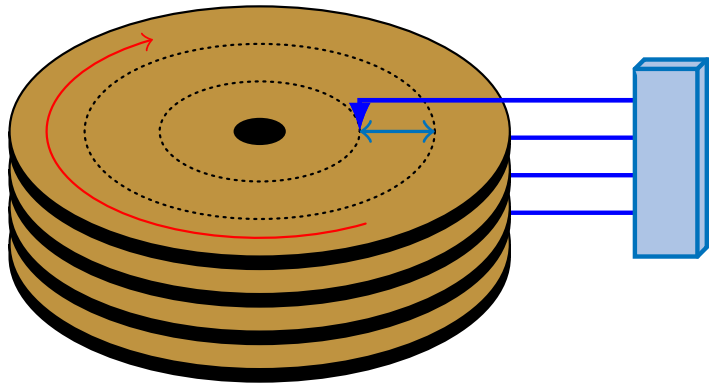
STXXL 1.4.0 and Beyond: External Memory Algorithms

Timo Bingmann | June 17th, 2014 @ 3rd LSDMA Topical Meeting

INSTITUTE OF THEORETICAL INFORMATICS – ALGORITHMICS

External Memory (EM) Model



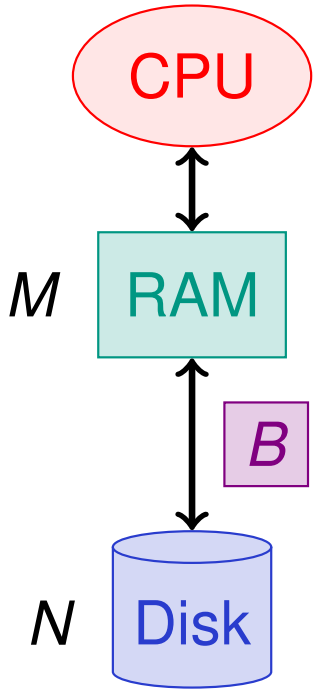


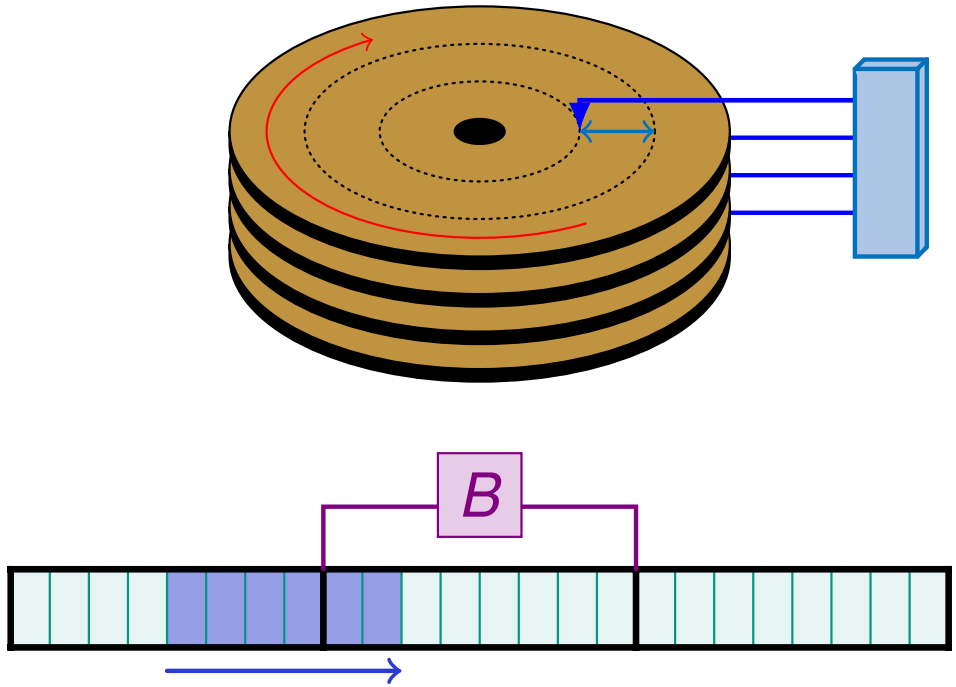
$$M = 1\,\text{GiB} = 2^{30}$$

$$B = 1\,\text{MiB} = 2^{20}$$

$$N = 1\,\text{TiB} = 2^{40}$$

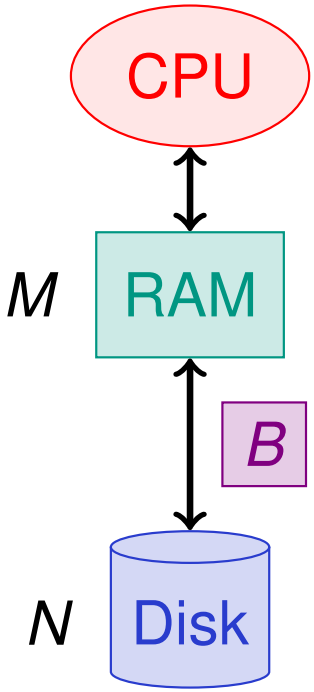
External Memory (EM) Model

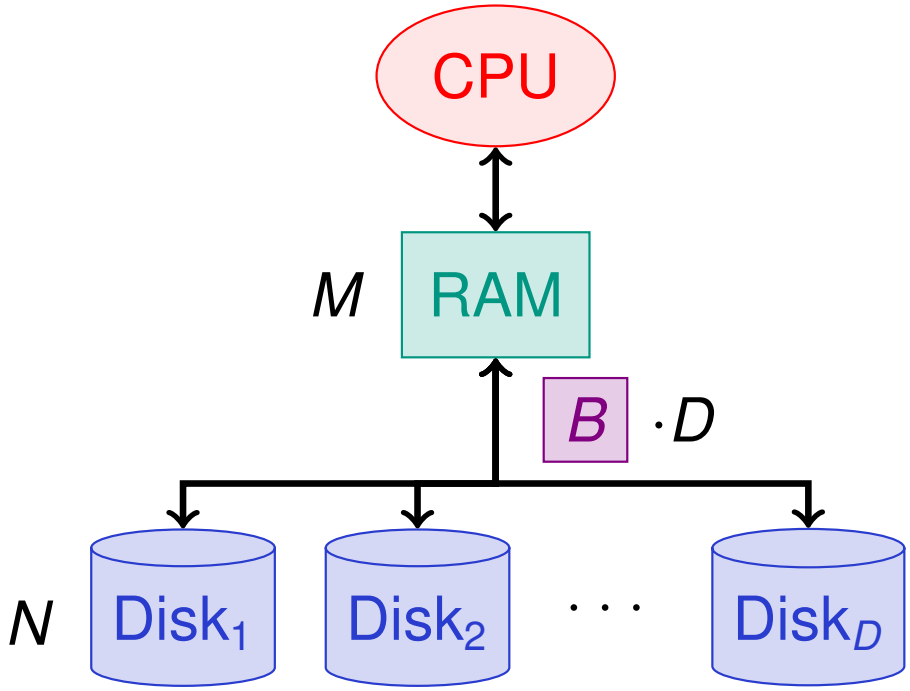




Scanning:	$\frac{N}{B}$ I/Os	$M = 1\,\text{GiB} = 2^{30}$
Random Access:	N I/Os	$B = 1\,\text{MiB} = 2^{20}$
Sorting:	$\mathcal{O}(\frac{N}{B}\lceil\log_{\frac{M}{B}}\frac{N}{M}\rceil)$ I/Os	$N = 1\,\text{TiB} = 2^{40}$

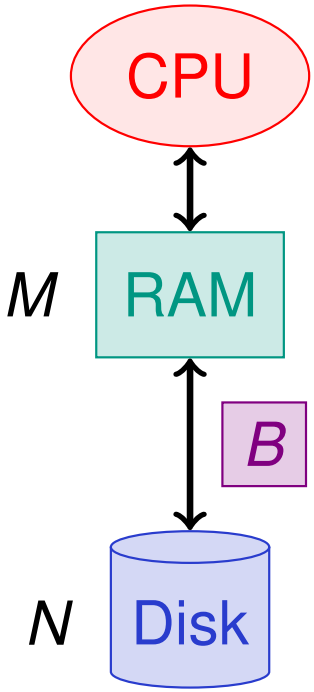
External Memory (EM) Model

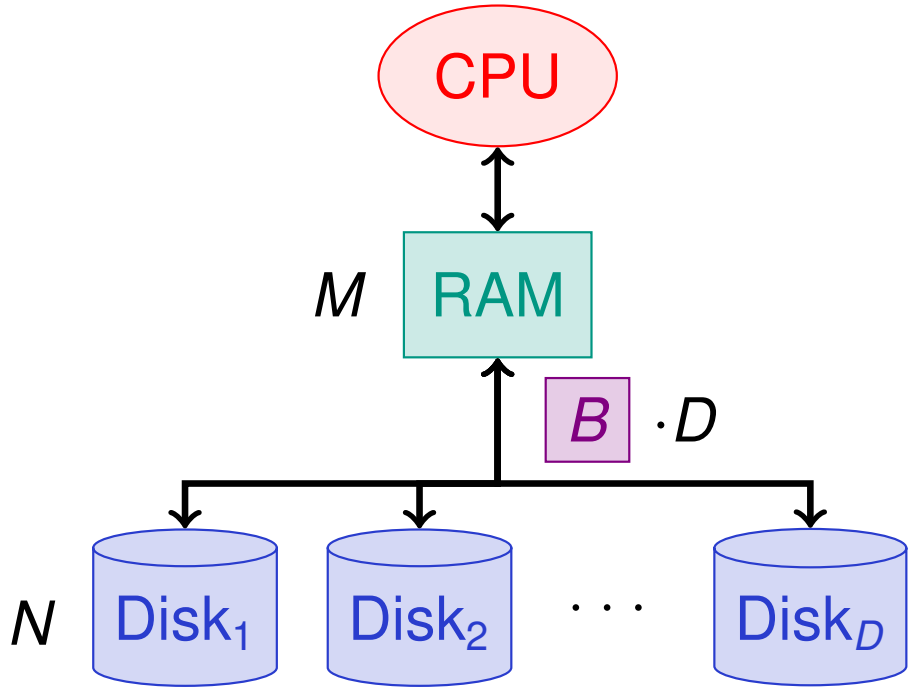




Scanning:	$\frac{N}{DB}$ I/Os	$M = 1\,\text{GiB} = 2^{30}$
Random Access:	N I/Os	$B = 1\,\text{MiB} = 2^{20}$
Sorting:	$\mathcal{O}(\frac{N}{DB}\lceil\log_{\frac{M}{B}}\frac{N}{M}\rceil)$ I/Os	$N = 1\,\text{TiB} = 2^{40}$

External Memory (EM) Model





Scanning:	$\mathcal{O}(\frac{N}{DB})$ I/Os	$M = 1\,\text{GiB} = 2^{30}$
Random Access:	$\mathcal{O}(N)$ I/Os	$B = 1\,\text{MiB} = 2^{20}$
Sorting:	$\mathcal{O}(\frac{N}{DB}\lceil\log_{\frac{M}{B}}\frac{N}{M}\rceil)$ I/Os	$N = 1\,\text{TiB} = 2^{40}$

STXXL Overview

Basic Properties:

- C++ template library of efficient external memory algorithms
- Licensed under the Boost Software License, Version 1.0
- Supports Linux, Windows, and Mac OS X.
- Primary authors: Roman Dementiev, Johannes Singler, Andreas Beckmann. Initiator: Peter Sanders.

STXXL Overview

Main Features:

- STL-compatible containers and algorithms.
- Efficient, highly optimized sorting implementation.
- Transparent parallel disk support.
- Pipelined Sorting, efficient priority queue, matrix operations.
- Partially parallelized shared memory algorithms.

Well-Known C++ STL Interfaces

Containers:

- `std::vector<T>`
  Operations: `push_back()`, `operator[]`, `begin()`, ...
- `std::stack<T>`, `std::queue<T>`, `std::deque<T>`
  Operations: `push()`, `top()`, `pop()`, ...
- `std::priority_queue<T,C,Cmp>`
  Operations: `push()`, `top()`, `pop()`, ...
- `std::map<K,V>`, `std::set<K>`
  Operations: `insert()`, `find()`, ...

Algorithms:

- `std::sort(begin,end,Cmp)`
- `std::for_each(begin,end,Functor)`

Well-Known C++ STL Interfaces

Containers:

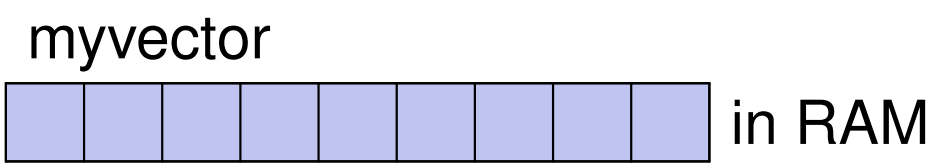
- `stxxl::vector<T>`
  Operations: `push_back()`, `operator[]`, `begin()`, ...
- `stxxl::stack<T>`, `stxxl::queue<T>`, `stxxl::deque<T>`
  Operations: `push()`, `top()`, `pop()`, ...
- `stxxl::priority_queue<T,C,Cmp>`
  Operations: `push()`, `top()`, `pop()`, ...
- `stxxl::map<K,V>`, `stxxl::set<K>`
  Operations: `insert()`, `find()`, ...

Algorithms:

- `stxxl::sort(begin,end,Cmp,memory)`
- `stxxl::for_each(begin,end,Functor,buffers)`

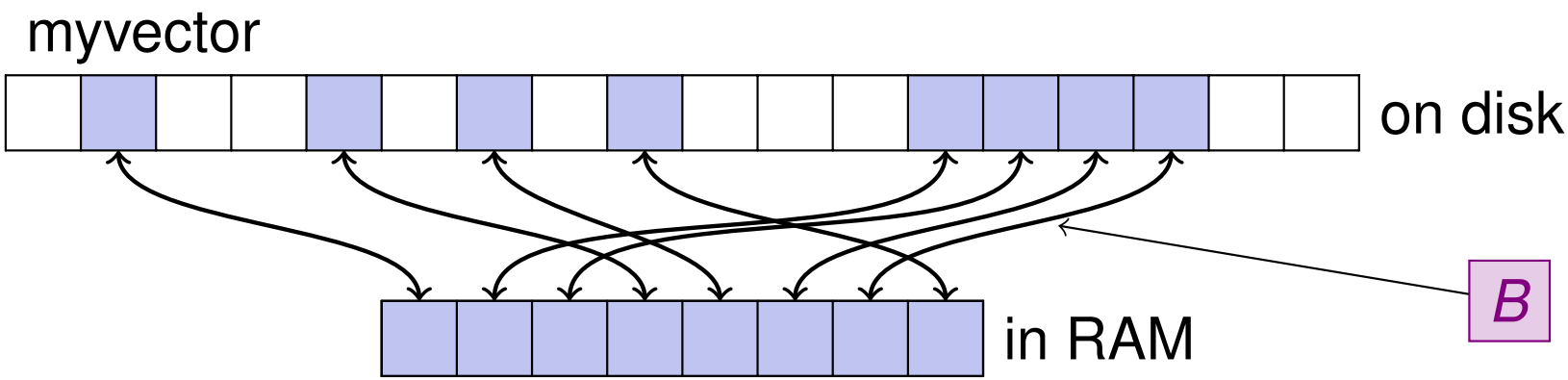
stxxl::vector Architecture

```
struct MyData { int a, b; };
std::vector<MyData> myvector;
for (int i = 0; i < N; i++) {
    myvector[i].a = 42 + i;
}
```

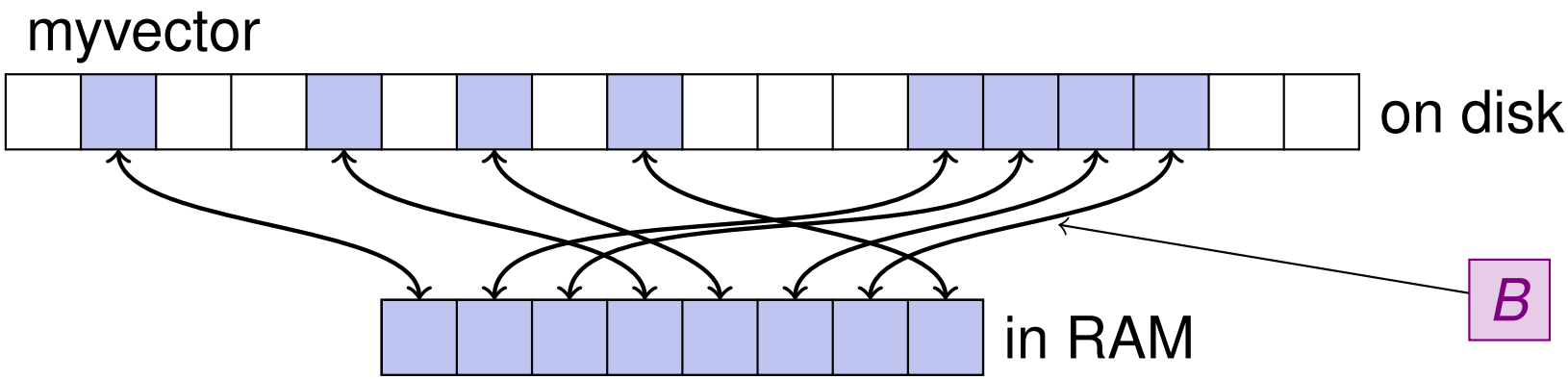


stxxl::vector Architecture

```
struct MyData { int a, b; };
stxxl::VECTOR_GENERATOR<MyData>::result myvector;
for (int i = 0; i < N; i++) {
    myvector[i].a = 42 + i;
}
```



stxxl::vector Interface



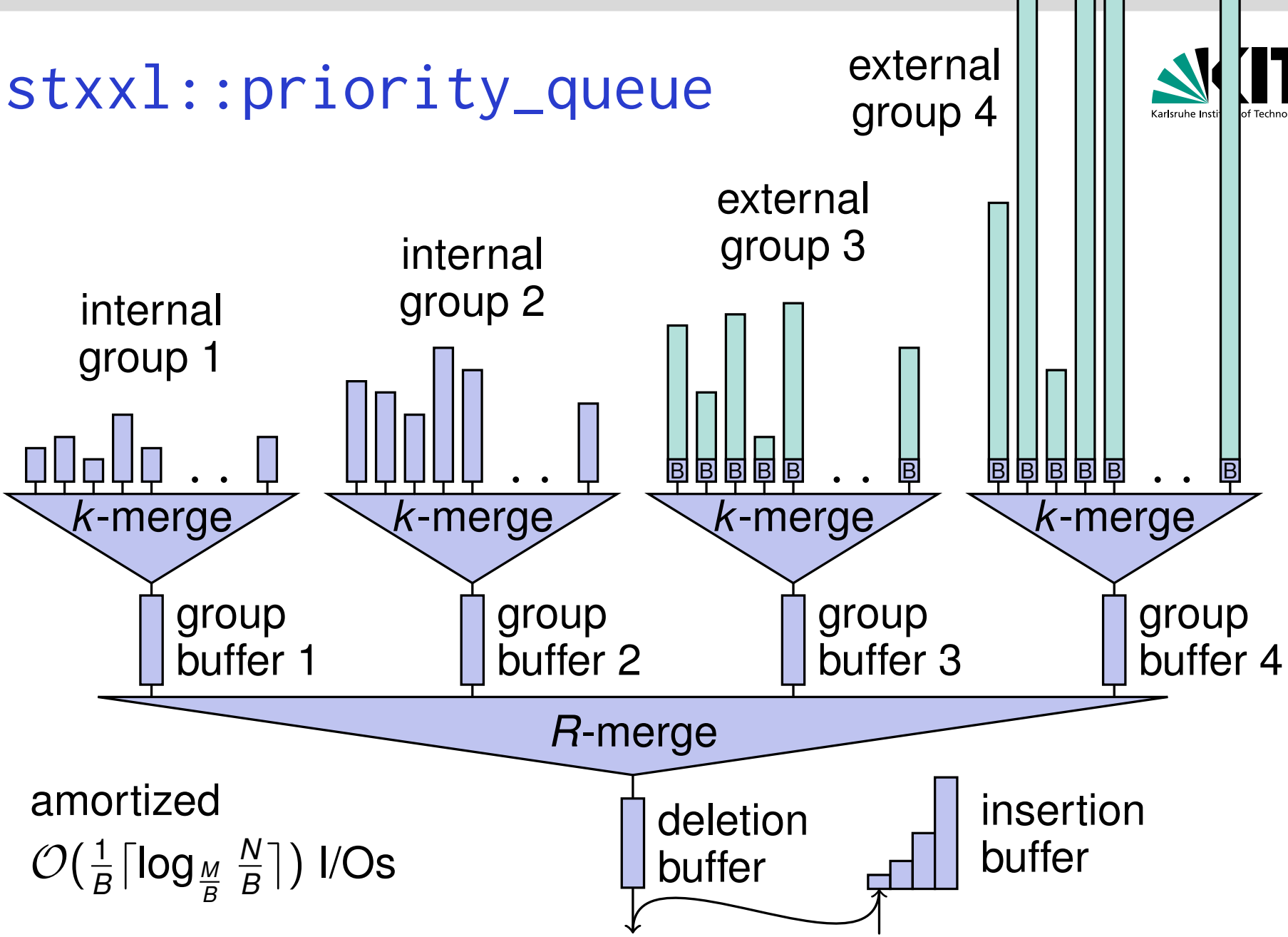

```
typedef stxxl::VECTOR_GENERATOR<MyData>::result vector_type;

vector_type vec;
vec.push_back(data);   vec[42] = data;
vec.resize(1024);      vec.size();

for (vector_type::iterator it = vec.begin();
     it != vec.end(); ++it)
{ *it = 42; }
```

stxxl::stack and stxxl::queue

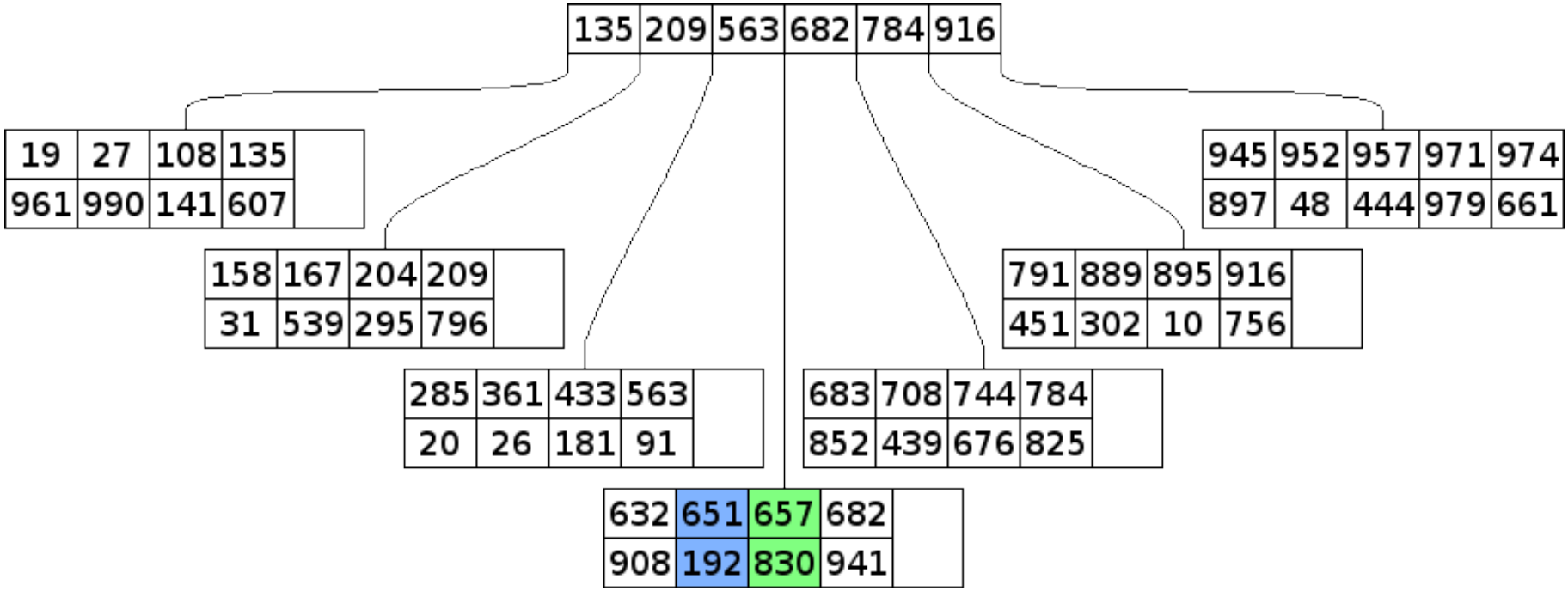
```
stxxl::stack<MyData> mystack;
```


```
stxxl::queue<MyData> myqueue;
```

stxxl::priority_queue



stxxl::map – a B-Tree

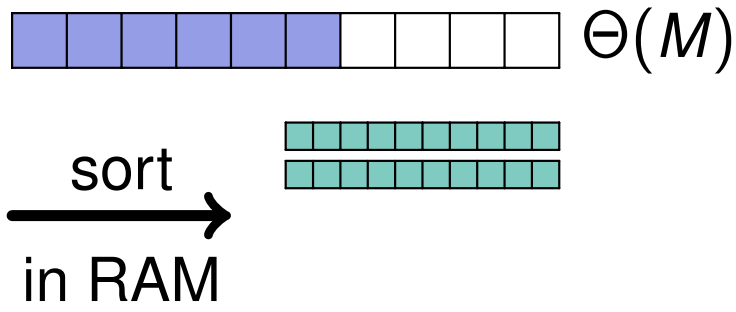


Sorting in STXXL

```
struct MyData { int a, b; }; MyData data = { 42, 6*9 };
stxxl::sorter<MyData, ComparisonFunctor> sorter;

// push all data into sorter
for (int i = 0; i < N; ++i)
    sorter.push(data);

// switch to reading state
sorter.sort();

// get back, in sorted order
while ( !sorter.empty() )
    data = *sorter, ++sorter;
```

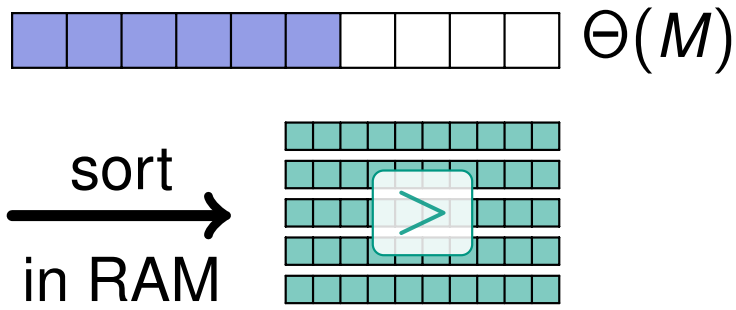
Sorting in STXXL

```
struct MyData { int a, b; }; MyData data = { 42, 6*9 };
stxxl::sorter<MyData, ComparisonFunctor> sorter;

// push all data into sorter
for (int i = 0; i < N; ++i)
    sorter.push(data);

// switch to reading state
sorter.sort();

// get back, in sorted order
while ( !sorter.empty() )
    data = *sorter, ++sorter;
```



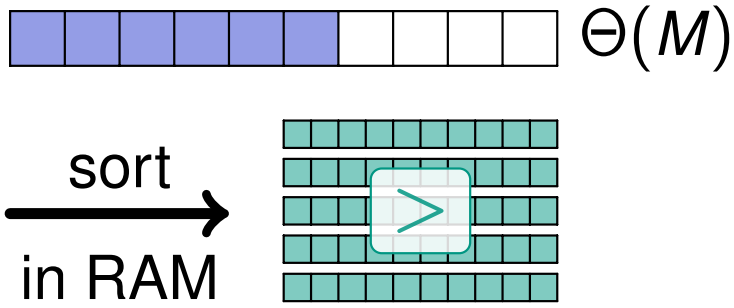
Sorting in STXXL

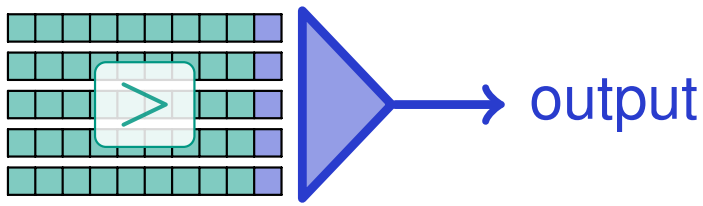
```
struct MyData { int a, b; }; MyData data = { 42, 6*9 };
stxxl::sorter<MyData, ComparisonFunctor> sorter;

// push all data into sorter
for (int i = 0; i < N; ++i)
    sorter.push(data);

// switch to reading state
sorter.sort();

// get back, in sorted order
while ( !sorter.empty() )
    data = *sorter, ++sorter;
```



Sorting in STXXL

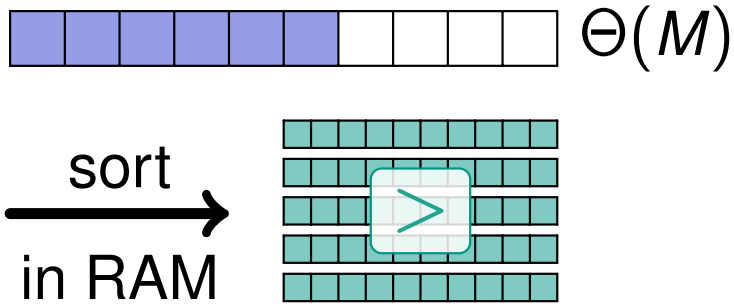
```
struct MyData { int a, b; }; MyData data = { 42, 6*9 };
stxxl::sorter<MyData, ComparisonFunctor> sorter;

// push all data into sorter
for (int i = 0; i < N; ++i)
    sorter.push(data);

// switch to reading state
sorter.sort();

// get back, in sorted order
while ( !sorter.empty() )
    data = *sorter, ++sorter;
```



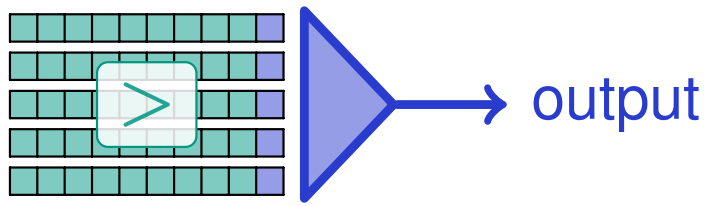


Sorting in STXXL

```
struct MyData { int a, b; }; MyData data = { 42, 6*9 };
stxxl::sorter<MyData, ComparisonFunctor> sorter;

// push all data into sorter
for (int i = 0; i < N; ++i)
    sorter.push(data);

// switch to reading state
sorter.sort();

// get back, in sorted order
while ( !sorter.empty() )
    data = *sorter, ++sorter;
```



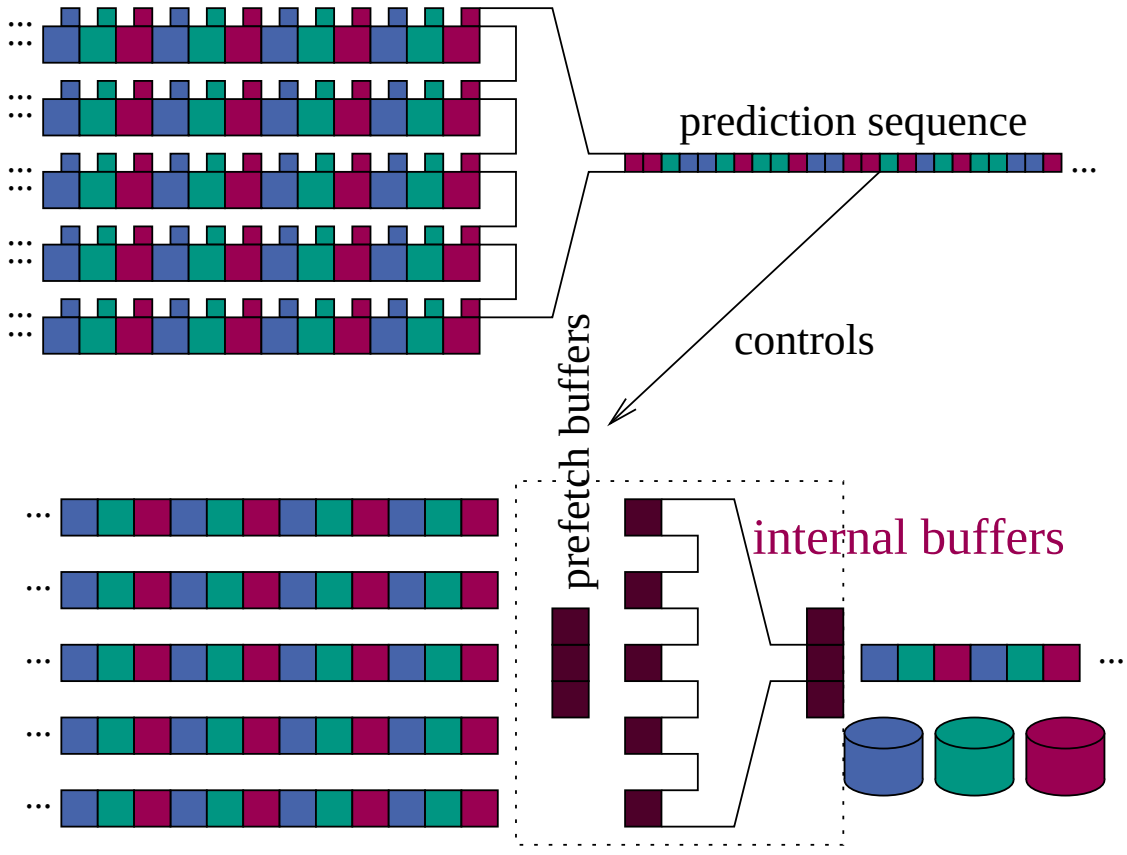
$\mathcal{O}(\frac{N}{B}\lceil\log_{\frac{M}{B}}\frac{N}{M}\rceil)$ I/Os, usually $2\frac{N}{B}$ I/Os



Prediction

[Folklore, Knuth]
Smallest Element
of each block
triggers fetch.

Prefetch buffers
allow parallel access
of next blocks



Tournament Trees for Multiway Merging

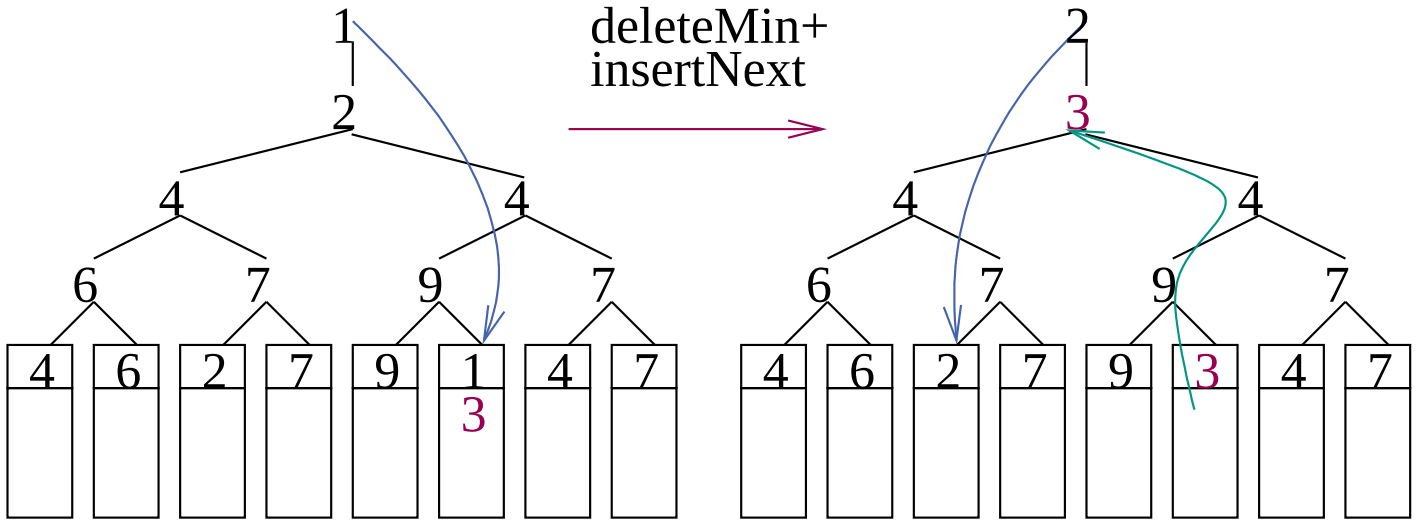
Assume $k = 2^K$ runs

K level complete binary tree

Leaves: smallest current element of each run

Internal nodes: loser of a competition for being smallest

Above root: global winner



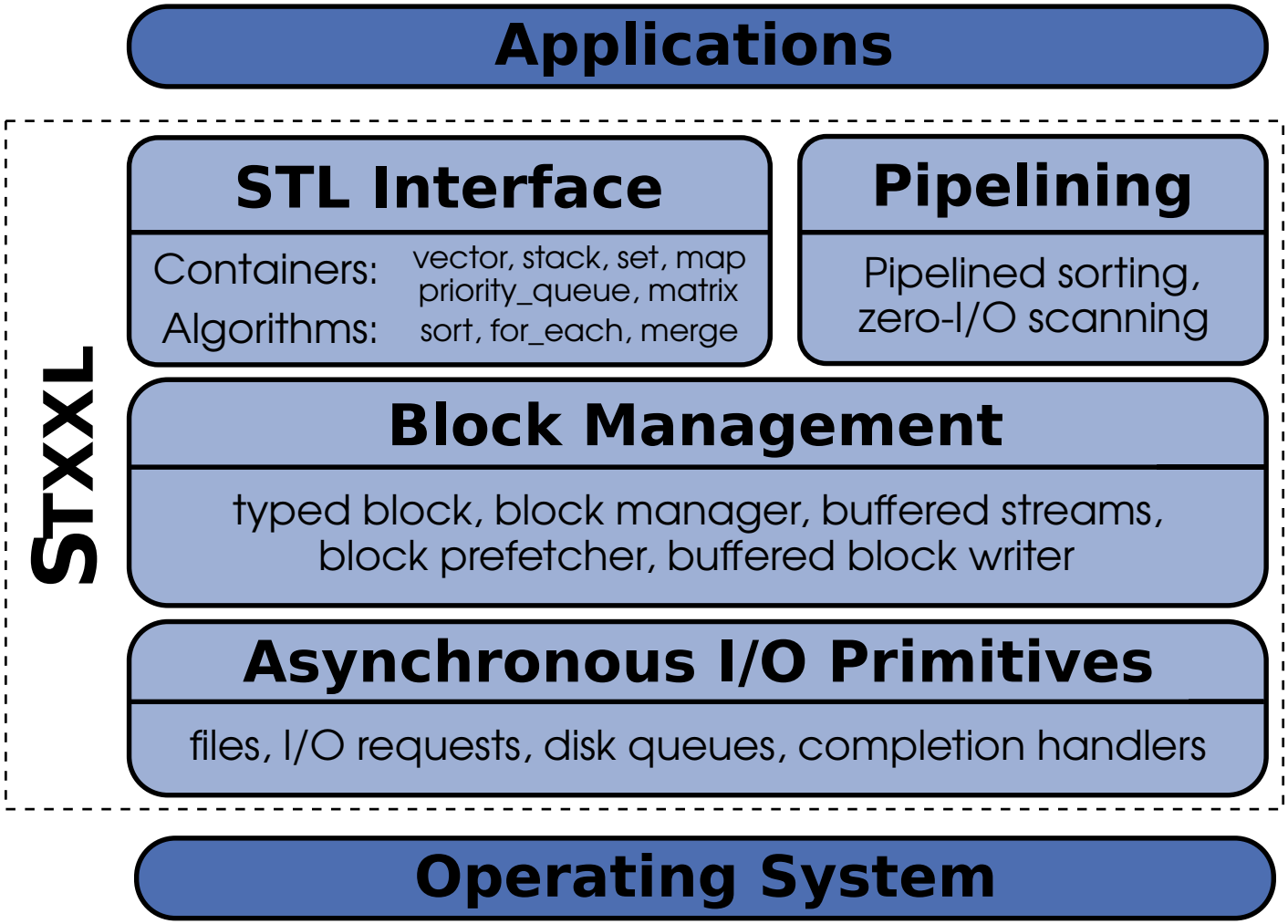
Sorting in STXXL – Rotational Disks

```
$ stxxl_tool benchmark_sort 256gib -M 16gib
Parameter size set to 274877906944.
Option -M, --ram set to 17179869184.
[STXXL-MSG] STXXL v1.4.99 (prerelease/Release) (git 79ceaa1fe6542db0d3fa8489d
[STXXL-MSG] Disk '/dev/sde1' is allocated, space: 2861587 MiB, I/O implementa
[STXXL-MSG] Disk '/dev/sdf1' is allocated, space: 2861587 MiB, I/O implementa
[STXXL-MSG] Disk '/dev/sdg1' is allocated, space: 2861587 MiB, I/O implementa
[STXXL-MSG] Disk '/dev/sdh1' is allocated, space: 2765655 MiB, I/O implementa
[STXXL-MSG] In total 4 disks are allocated, space: 11350417 MiB
#!!! running sorting test with pair of uint64 = 16 bytes.
# materialize random_stream into vector of size 17179869184
finished in 380.801 seconds @ 688.401 MiB/s
# stxxl::sort vector of size 17179869184
finished in 1766.61 seconds @ 148.388 MiB/s
# stxxl::ksort vector of size 17179869184
finished in 2065.61 seconds @ 126.909 MiB/s
# stxxl::stream::sort of size 17179869184
finished in 1198.99 seconds @ 218.637 MiB/s
```

Sorting in STXXL – SSD Disks

```
$ stxxl_tool benchmark_sort 256gib -M 16gib
Parameter size set to 274877906944.
Option -M, --ram set to 17179869184.
[STXXL-MSG] STXXL v1.4.99 (prerelease/Release) (git 79ceaa1fe6542db0d3fa8489d
[STXXL-MSG] Disk '/dev/sda1' is allocated, space: 953868 MiB, I/O implementat
[STXXL-MSG] Disk '/dev/sdb1' is allocated, space: 953868 MiB, I/O implementat
[STXXL-MSG] Disk '/dev/sdc1' is allocated, space: 953868 MiB, I/O implementat
[STXXL-MSG] Disk '/dev/sdd1' is allocated, space: 953868 MiB, I/O implementat
[STXXL-MSG] In total 4 disks are allocated, space: 3815474 MiB
#!!! running sorting test with pair of uint64 = 16 bytes.
# materialize random_stream into vector of size 17179869184
finished in 170.389 seconds @ 1538.5 MiB/s
# stxxl::sort vector of size 17179869184
finished in 832.635 seconds @ 314.837 MiB/s
# stxxl::ksort vector of size 17179869184
finished in 990.998 seconds @ 264.525 MiB/s
# stxxl::stream::sort of size 17179869184
finished in 988.763 seconds @ 265.123 MiB/s
```

STXXL Design – Layers



Pipelining – Materialize Sorted Stream

```
typedef std::pair<uint64_t,uint64_t> MyData
stxxl::vector<MyData> myvector(N);
for (int i = 0; i < N; ++i)
    myvector[i] = std::make_pair(rand(),rand());
stxxl::sort(myvector.begin(), myvector.end());
```

Pipelining – Materialize Sorted Stream

```
typedef std::pair<uint64_t,uint64_t> MyData
stxxl::vector<MyData> myvector(N);
for (int i = 0; i < N; ++i)
    myvector[i] = std::make_pair(rand(),rand());
stxxl::sort(myvector.begin(), myvector.end());
```

or we can ...

```
stxxl::sorter<MyData,MyCompare> mysorter(N);
for (int i = 0; i < N; ++i)
    mysorter.push( std::make_pair(rand(),rand()) );
stxxl::vector<MyData> myvector(N);
stxxl::stream::materialize(mysorter,
    myvector.begin(), myvector.end());
```

Pipelining – Materialize Sorted Stream

```
typedef std::pair<uint64_t,uint64_t> MyData
stxxl::vector<MyData> myvector(N);
for (int i = 0; i < N; ++i)
    myvector[i] = std::make_pair(rand(),ran
stxxl::sort(myvector.begin(), myvector.end(
```

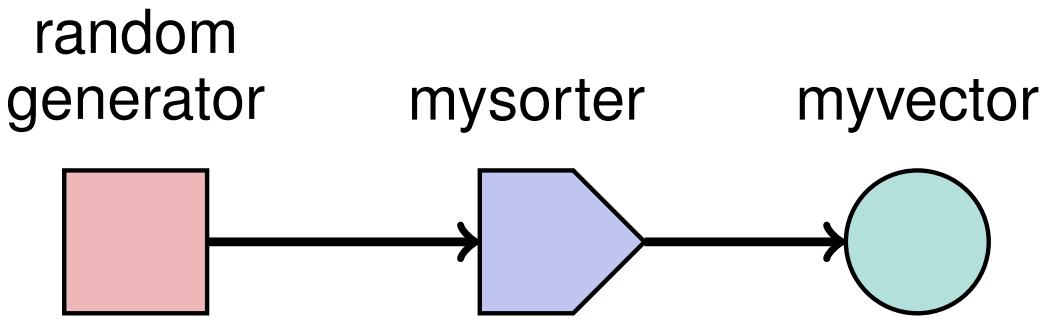
Write +
2x Read
2x Write

or we can ...

```
stxxl::sorter<MyData,MyCompare> mysorter(N);
for (int i = 0; i < N; ++i)
    mysorter.push( std::make_pair(rand(),ra
stxxl::vector<MyData> myvector(N);
stxxl::stream::materialize(mysorter,
    myvector.begin(), myvector.end());
```

1x Write
1x Read
+ Write

Pipelining – Materialize Sorted Stream




```
stxxl::sorter<MyData,MyCompare> mysorter(N);
for (int i = 0; i < N; ++i)
    mysorter.push( std::make_pair(rand(),ra
stxxl::vector<MyData> myvector(N);
stxxl::stream::materialize(mysorter,
    myvector.begin(), myvector.end());
```

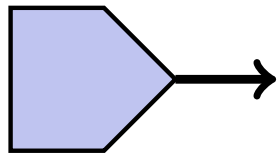
1x Write
1x Read
+ Write

Pipelining – Stream Concept

```
concept Stream {
    const value_type & operator* () const;
    Stream & operator++ ();
    bool empty();
};
```

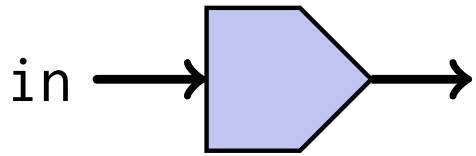
Pipelining – Stream Concept

```
concept Stream {
    const value_type & operator* () const;
    Stream & operator++ ();
    bool empty();
};


stxxl::sorter<MyData, CmpType> sorter;
// ... fill sorter

// get back, in sorted order
while ( !sorter.empty() )
    data = *sorter, ++sorter;
```



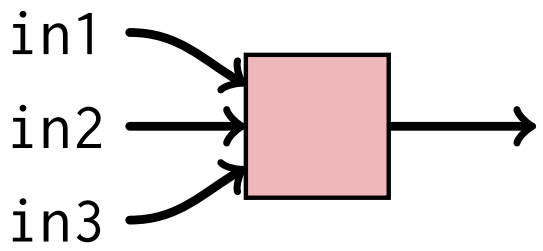
Pipelining – Stream Concept

```
concept Stream {
    const value_type & operator* () const;
    Stream & operator++ ();
    bool empty();
};


class stxxl::stream::sort : Stream {
    typedef Input::value_type value_type;
    sort(Input in, CmpType cmp);
    // implements Stream methods
};
```



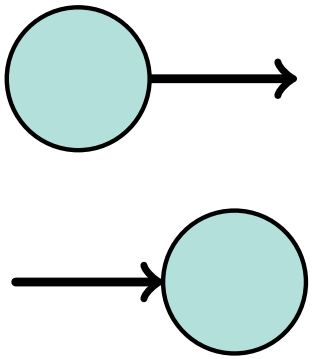
Pipelining – Stream Concept

```
concept Stream {
    const value_type & operator* () const;
    Stream & operator++ ();
    bool empty();
};

class stxxl::stream::transform : Stream {
    typedef Operation::value_type value_type;
    transform(Operation op,
              Input1 in1, Input2 in2, Input3 in3, ...);
    // implements Stream methods
};
```

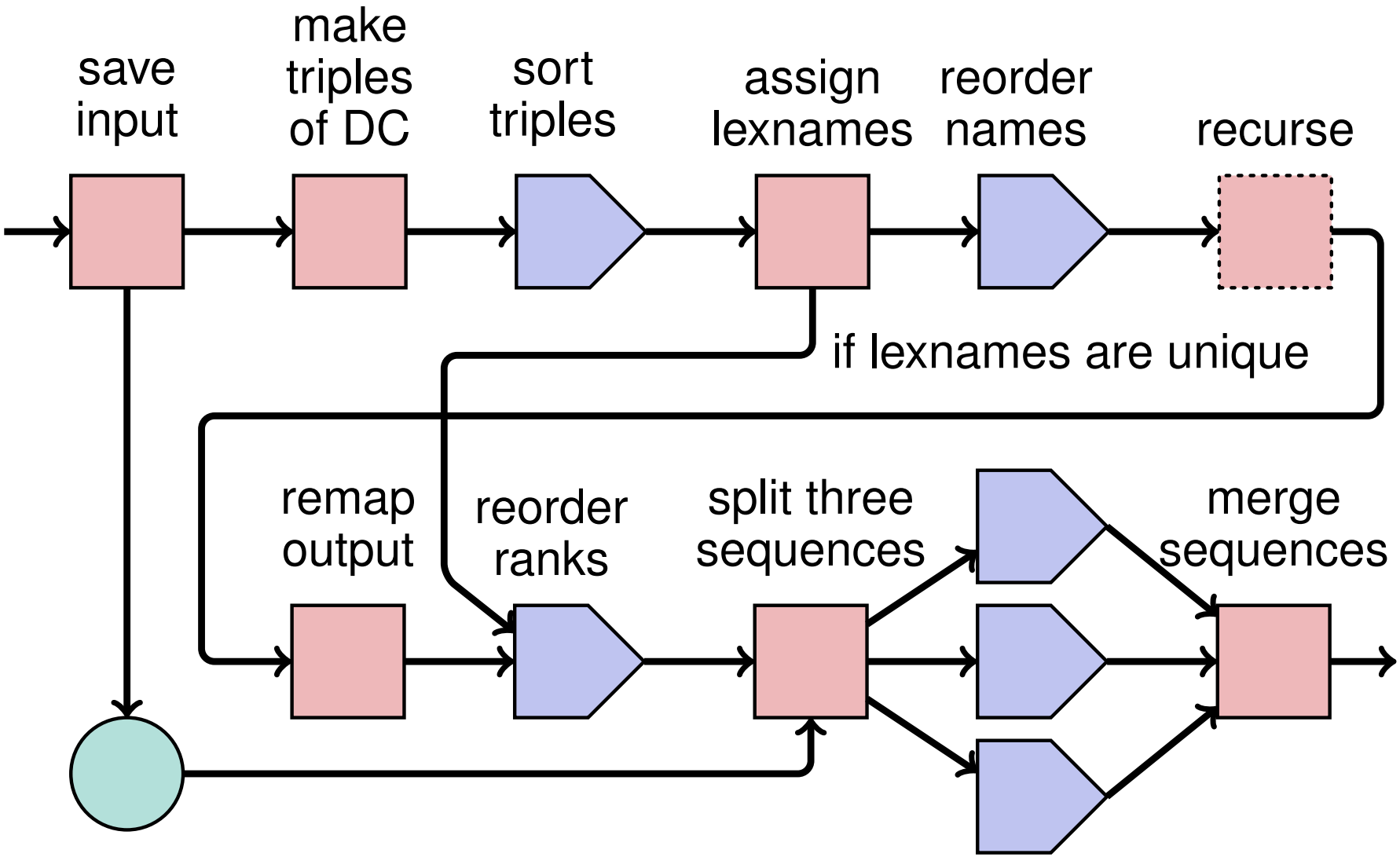


Pipelining – Stream Concept

```
concept Stream {
    const value_type & operator* () const;
    Stream & operator++ ();
    bool empty();
};

class stxxl::stream::vector_iterator2stream : Stream {
    typedef Iterator::value_type value_type;
    vector_iterator2stream(Iterator begin,
                           Iterator end);
    // implements Stream methods
};

stxxl::stream::materialize(stream, begin, end);
```



Pipelining – DC3 Suffix Sorting



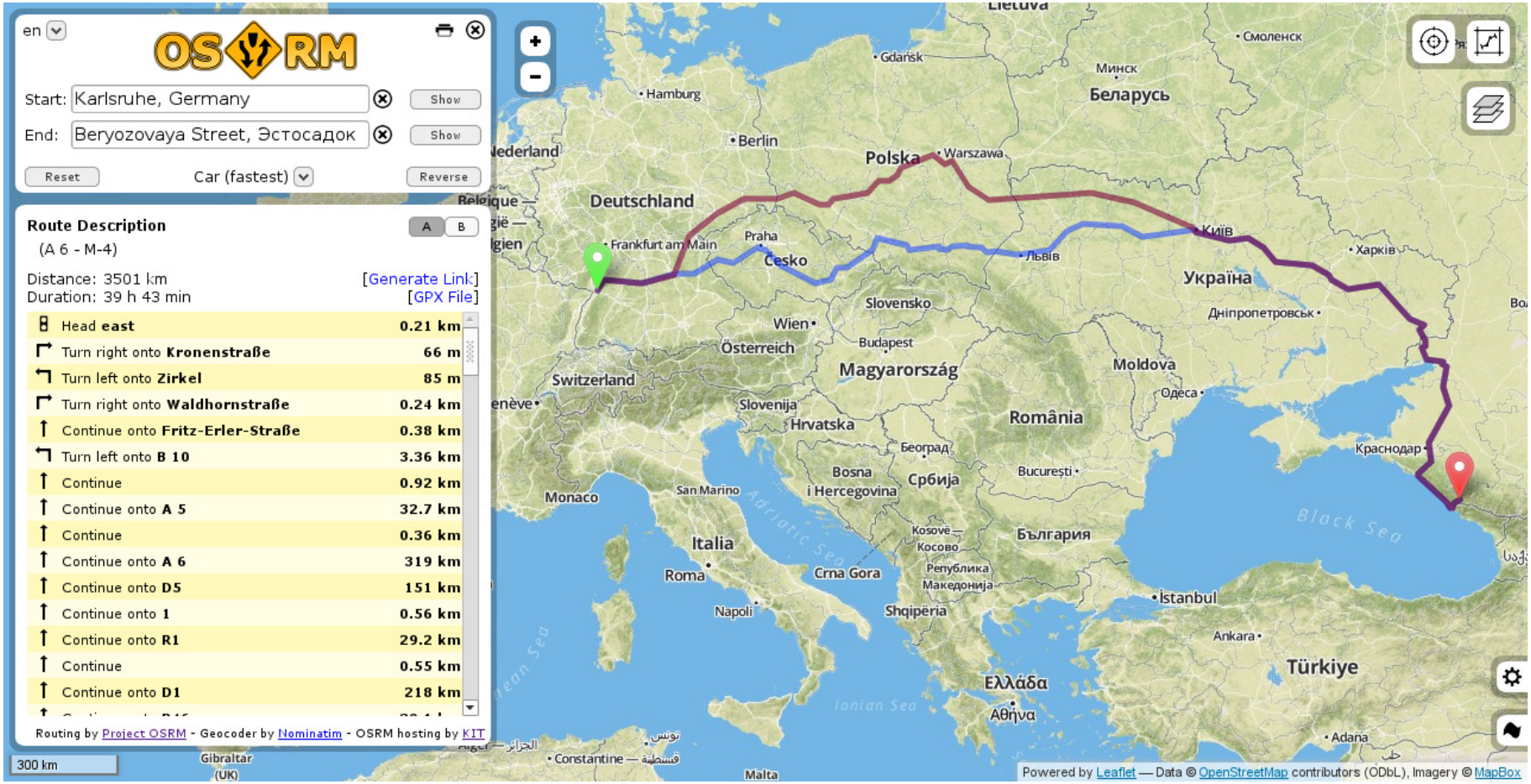
Applications using STXXL

Past:

- Minimum spanning tree and breadth-first search in EM.
- Large matrix operations: multiplication and inversion.
- EM suffix array construction with DC3.

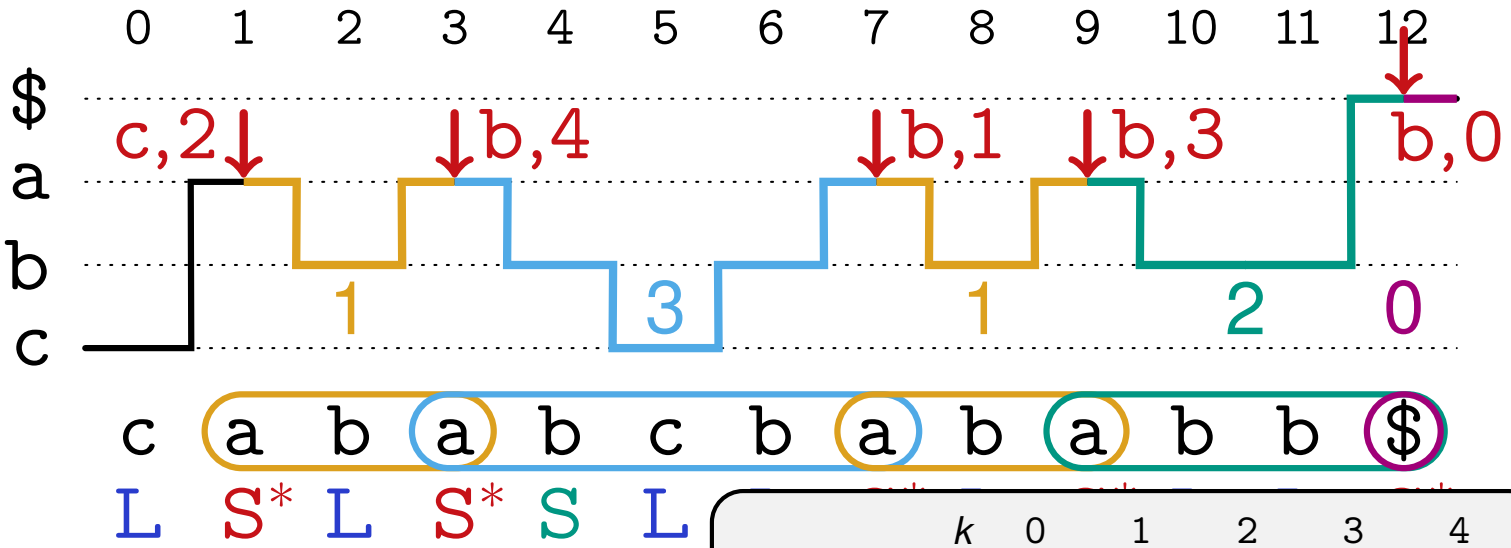
Present and Future of STXXL

- OSRM – Open Source Routing Machine:
  make shortest path preprocessing scale to OpenStreetMap.



Present and Future of STXXL

- eSAIS – induced suffix and LCP construction for text indexes.

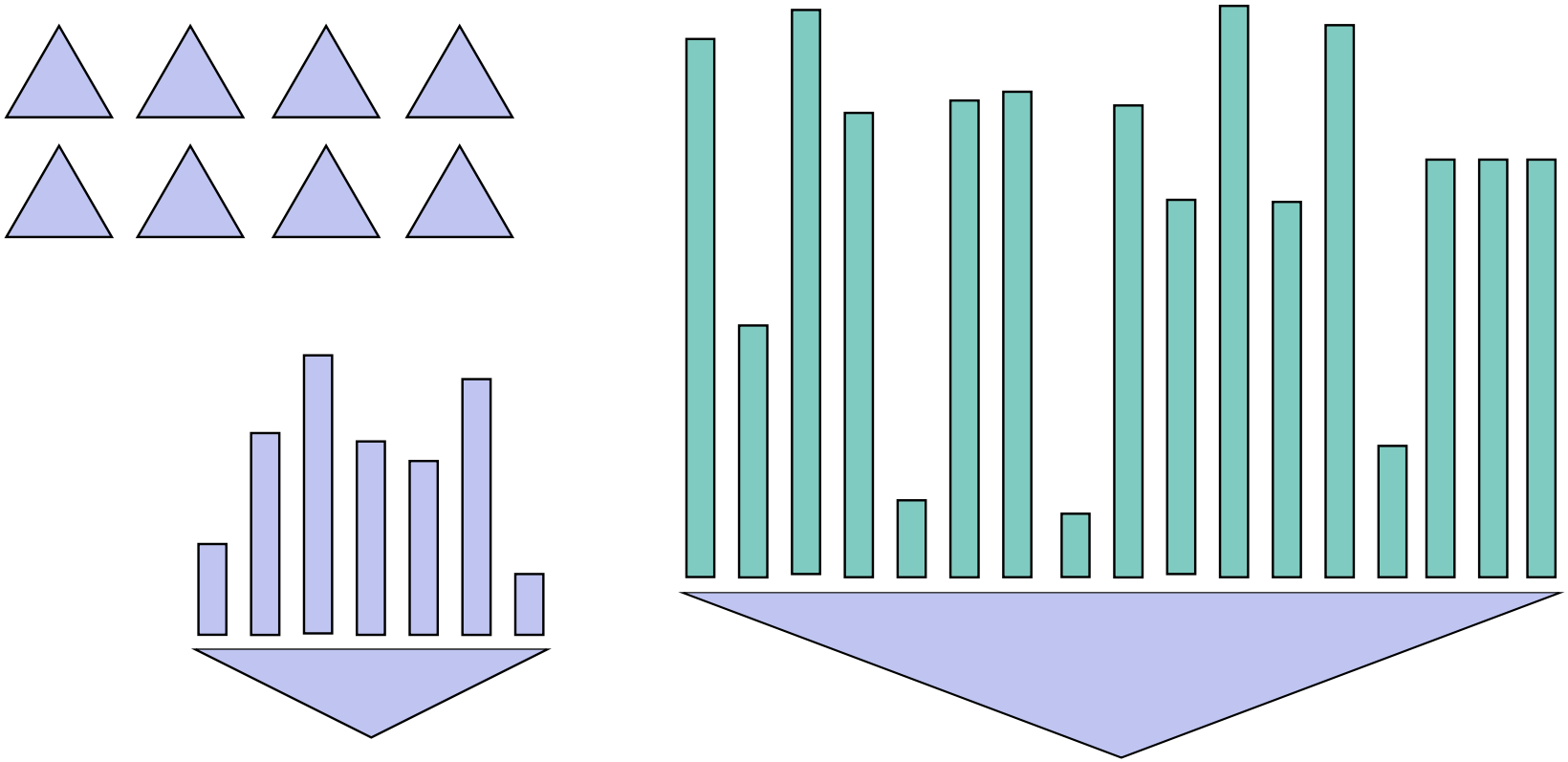


	$	a			b
SA	12	7	1	9	3
LCP	-	0	4	2	2

k	0	1	2	3	4
R	1	3	1	2	0
SA_R	4	2	0	3	1
ISA_R	2	4	1	3	0
LCP_R	-	0	1	0	0
LCP_{S^*}	-	0	4	2	2

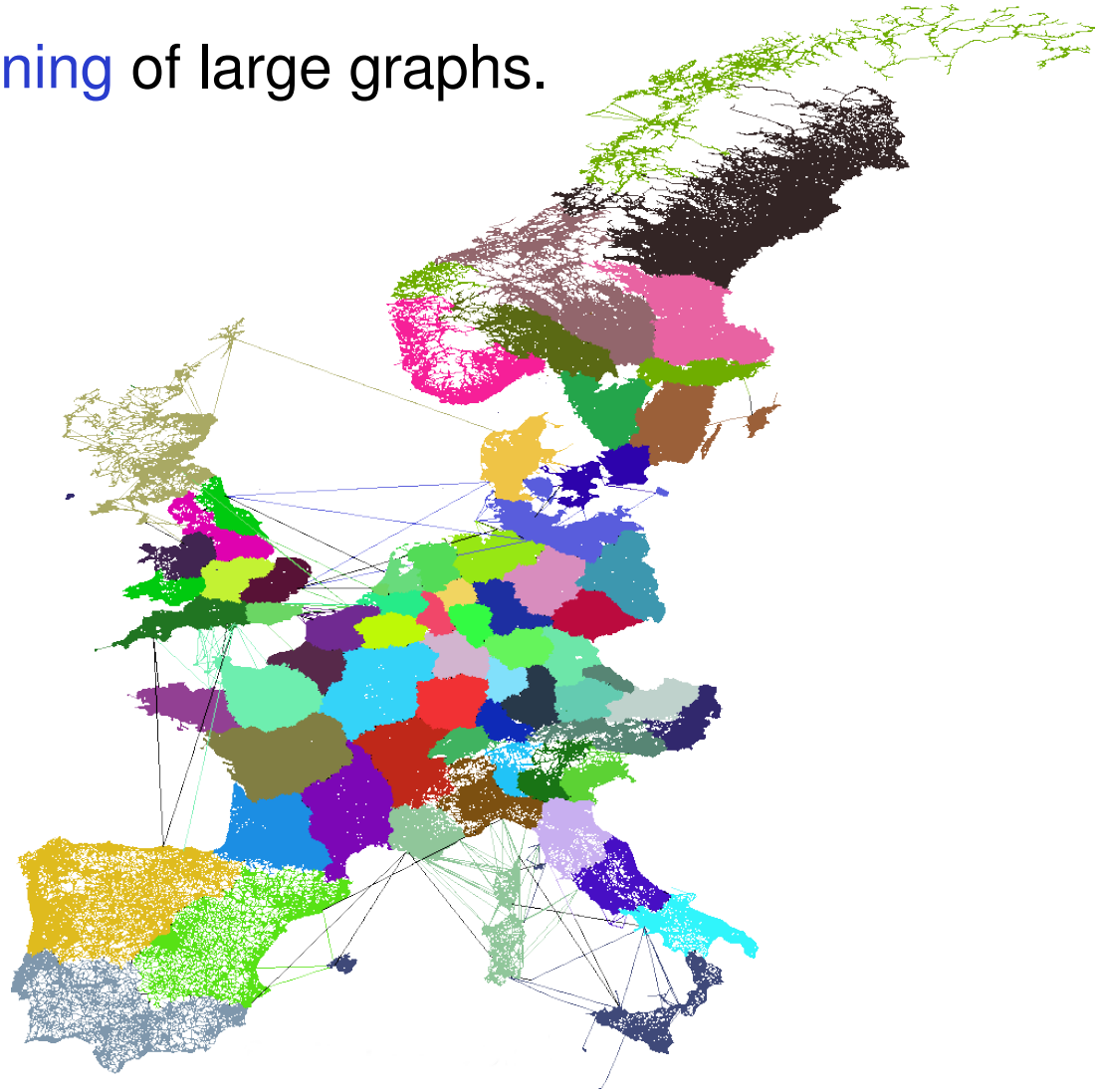
Present and Future of STXXL

- Parallel priority queue for fast multi-core algorithms.



Present and Future of STXXL

- EM graph partitioning of large graphs.



Development of STXXL

Present and Future:

- C++11 containers and additional functions.
- Direct support for Linux asynchronous I/O interface.
- Integration of EM hash_map and async pipelining branches.

MapReduce Projektpraktikum

WS 2014/15 + SS2015 (24 ECTS im Master)

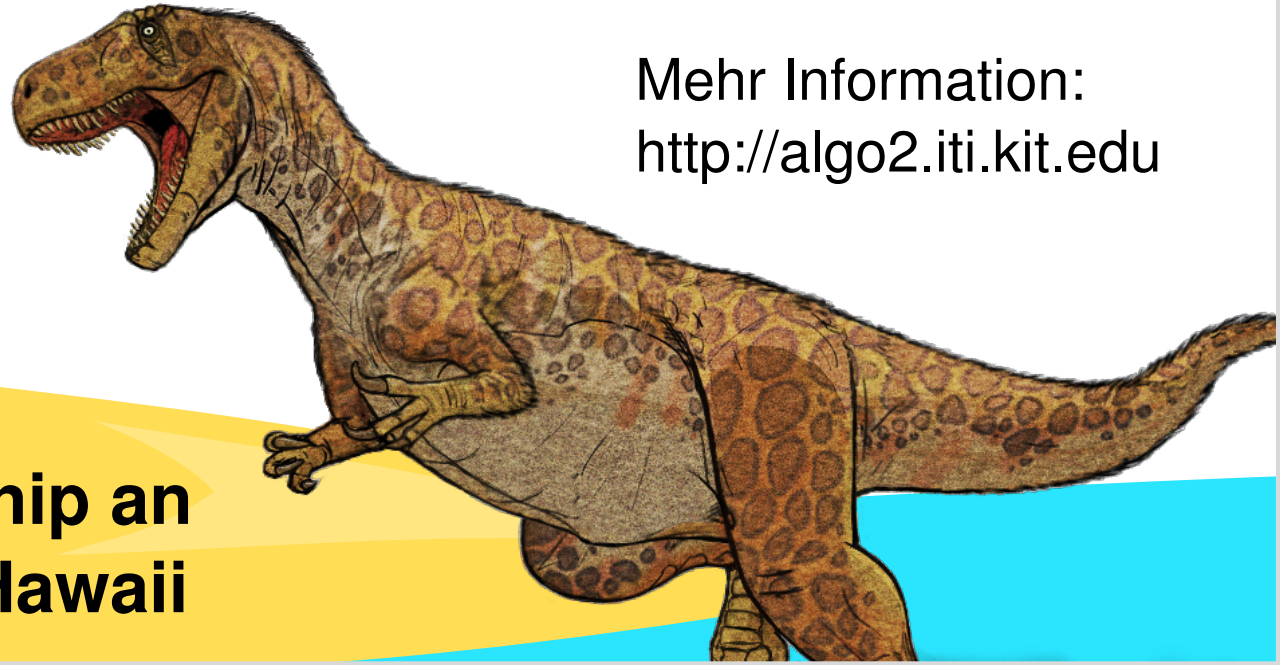
Karlsruher Institut für Technologie

Ziel: Hochleistungs C++ MapReduce Framework

zur massiv-parallelen Verarbeitung von "Big Data" mit C++.

Aufbauend auf Boost und effizienten parallelen Algorithmen für Hauptspeicher (MCSTL) oder Externspeicher (STXXL)

Entwicklung von neuen kommunikationseffizienten Datenaustausch und verteilen Prozess-Scheduling



Mehr Information:
http://algo2.iti.kit.edu

1. "Preis": Internship an der University of Hawaii

Karlsruhe Institute of Technology

Show documentation and demonstration.

Thank you for your attention!

Questions?