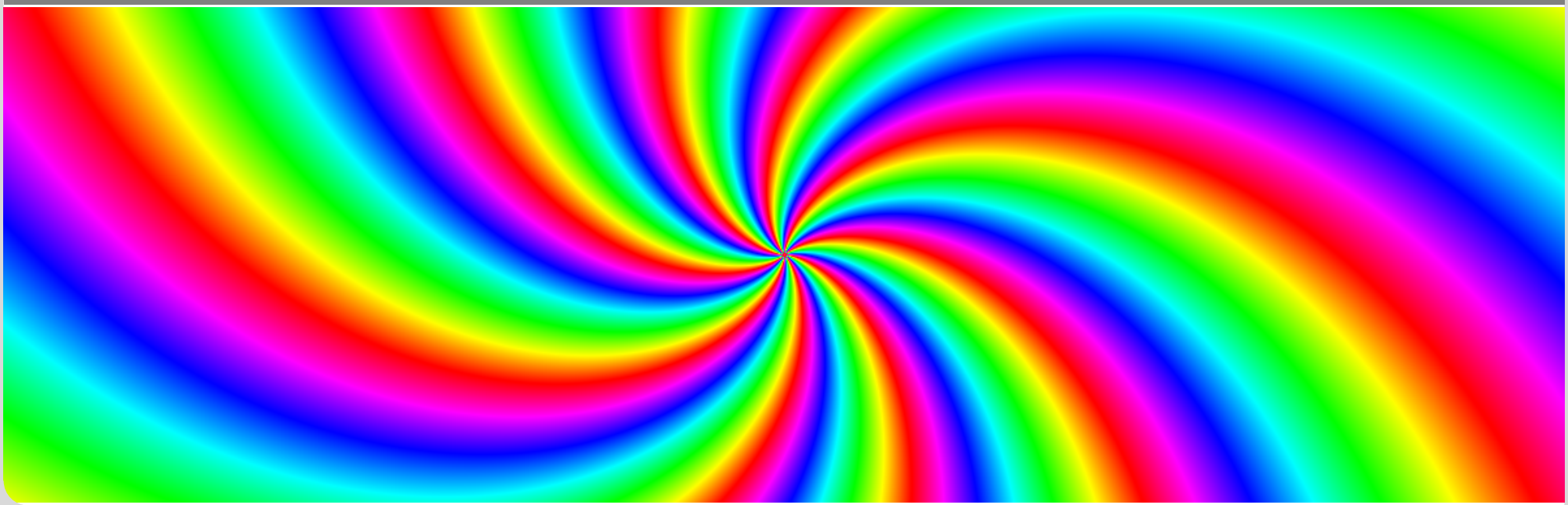
Karlsruhe Institute of Technology

Thrill :

Distributed Big Data Batch Processing in C++

Michael Axtmann, Timo Bingmann, Peter Sanders, Sebastian Schlag, and 6 Students | 2016-01-19

INSTITUTE OF THEORETICAL INFORMATICS – ALGORITHMICS



Example $T =$ [dbadcbccbabdcc$]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
SA_i	$T_{SA_i \ldots n}$														
14	$														
9	a	b	d	c	c	$									
2	a	d	c	b	c	c	b	a	b	d	c	c	$		
8	b	a	b	d	c	c	$								
1	b	a	d	c	b	c	c	b	a	b	d	c	c	$	
5	b	c	c	b	a	b	d	c	c	$					
10	b	d	c	c	$										
13	c	$													
7	c	b	a	b	d	c	c	$							
4	c	b	c	c	b	a	b	d	c	c	$				
12	c	c	$												
6	c	c	b	a	b	d	c	c	$						
0	d	b	a	d	c	b	c	c	b	a	b	d	c	c	$
3	d	c	b	c	c	b	a	b	d	c	c	$			
11	d	c	c	$											


bwUniCluster
512 x 16 cores, 64 GB RAM
© KIT (SCC)

Suffix Sorting with DC3: Example

	0	1	2	3	4	5	6	7	8	9	10
$T =$	d	b	a	c	b	a	c	b	d	$	$

$= [\, t_i \,]_{i=0,\ldots,n-1}$

triples	(bac,1),	(bac,4),	(bd$,7),	(acb,2)	(acb,5),	(d$$,8)
sorted	(acb,2)	(acb,5),	(bac,1),	(bac,4),	(bd$,7),	(d$$,8)
equal 0/1	0	0	1	0	1	1
prefix sum	0	0	1	1	2	3

$R =$ 1 1 2 0 0 3 $

r_1 r_4 r_7 r_2 r_5 r_8

$\mathrm{SA}_R =$ 3 4 0 1 2 5 $ $\quad \mathrm{ISA}_R =$ 2 3 4 0 1 5 $

$S_0 = [(\mathtt{d},\mathtt{b},2,0,0),(\mathtt{c},\mathtt{b},3,1,3),(\mathtt{c},\mathtt{b},4,5,6)]$ $\quad (t_i, t_{i+1}, r_{i+1}, r_{i+2}, i)$

$S_1 = [(2,\mathtt{b},0,1),(3,\mathtt{b},1,4),(4,\mathtt{b},5,7)]$ $\quad (r_{i+1}, t_{i+1}, r_{i+2}, i+1)$

$S_2 = [(0,\mathtt{a},\mathtt{c},3,2),(1,\mathtt{a},\mathtt{c},4,5),(5,\mathtt{d},\$,6,8)]$ $\quad (r_{i+2}, t_{i+2}, t'_{i+3}, r'_{i+4}, i+2)$

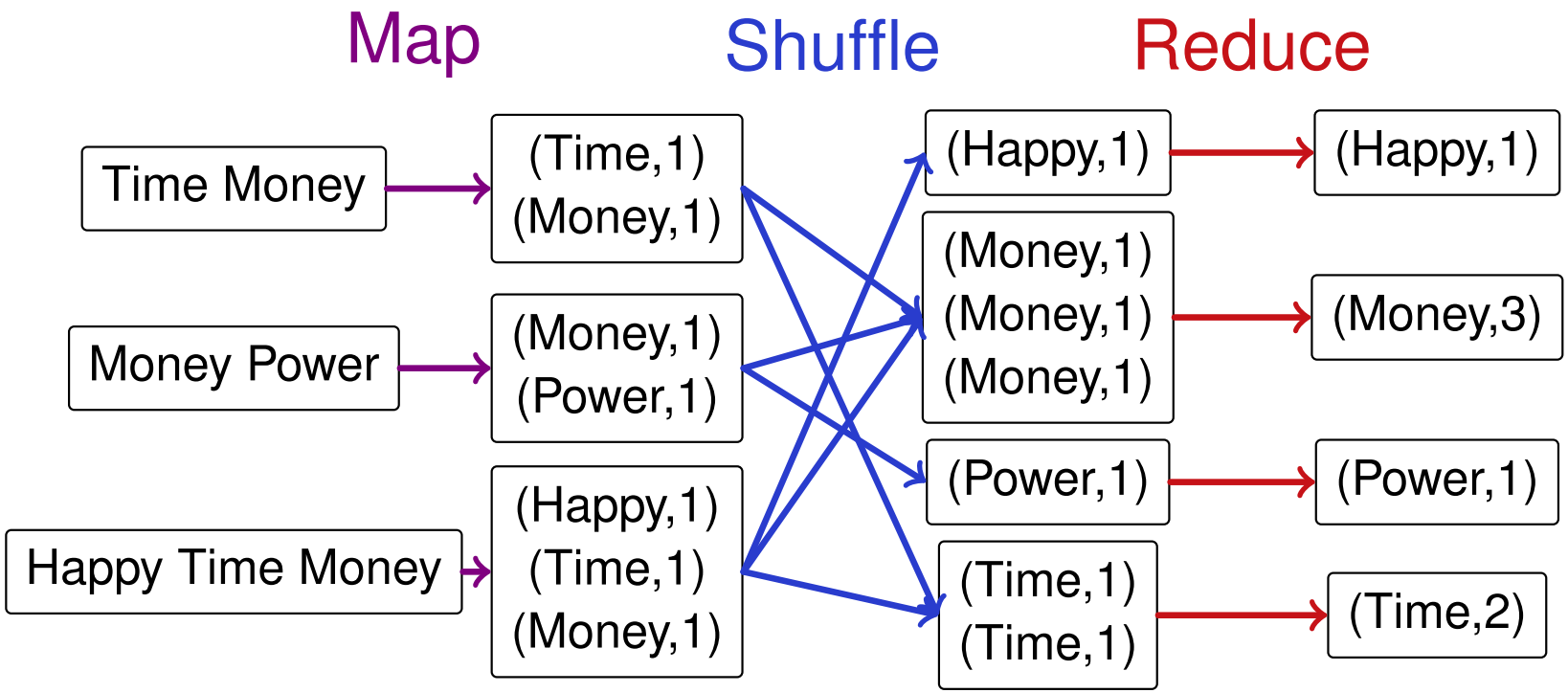
$\mathrm{SA}_T = \mathrm{Merge}(\mathrm{Sort}(S_0), \mathrm{Sort}(S_1), \mathrm{Sort}(S_2))$

$\Theta(\mathrm{sort}(n))$

Flavours of Big Data Frameworks

- High Performance Computing (Supercomputers)
  MPI
- Batch Processing
  Google's MapReduce, Hadoop MapReduce  , Apache Spark  , Apache Flink  (Stratosphere), Google's FlumeJava.
- Real-time Stream Processing
  Apache Storm  , Apache Spark Streaming, Google's MillWheel.
- Interactive Cached Queries
  Google's Dremel, Powerdrill and BigQuery, Apache Drill  .
- Sharded (NoSQL) Databases and Data Warehouses
  MongoDB  , Apache Cassandra, Apache Hive, Google BigTable, Hypertable, Amazon RedShift, FoundationDB.
- Graph Processing
  Google's Pregel, GraphLab  , Giraph  , GraphChi.
- Time-based Distributed Processing
  Microsoft's Dryad, Microsoft's Naiad.

What is Map/Reduce?



Computation model popularized in 2004
by Google with the name MapReduce.

Why Map/Reduce?

- Changes the perspective from the number of processors to how data is processed.

- A simple algorithmic and programming abstraction with
  - automatic parallelization of independent operations (map) and aggregation (reduce),
  - automatic distribution and balancing of data and work,
  - automatic fault tolerance versus hardware errors.

$\Rightarrow$ **MapReduce framework**

Where is Map/Reduce?

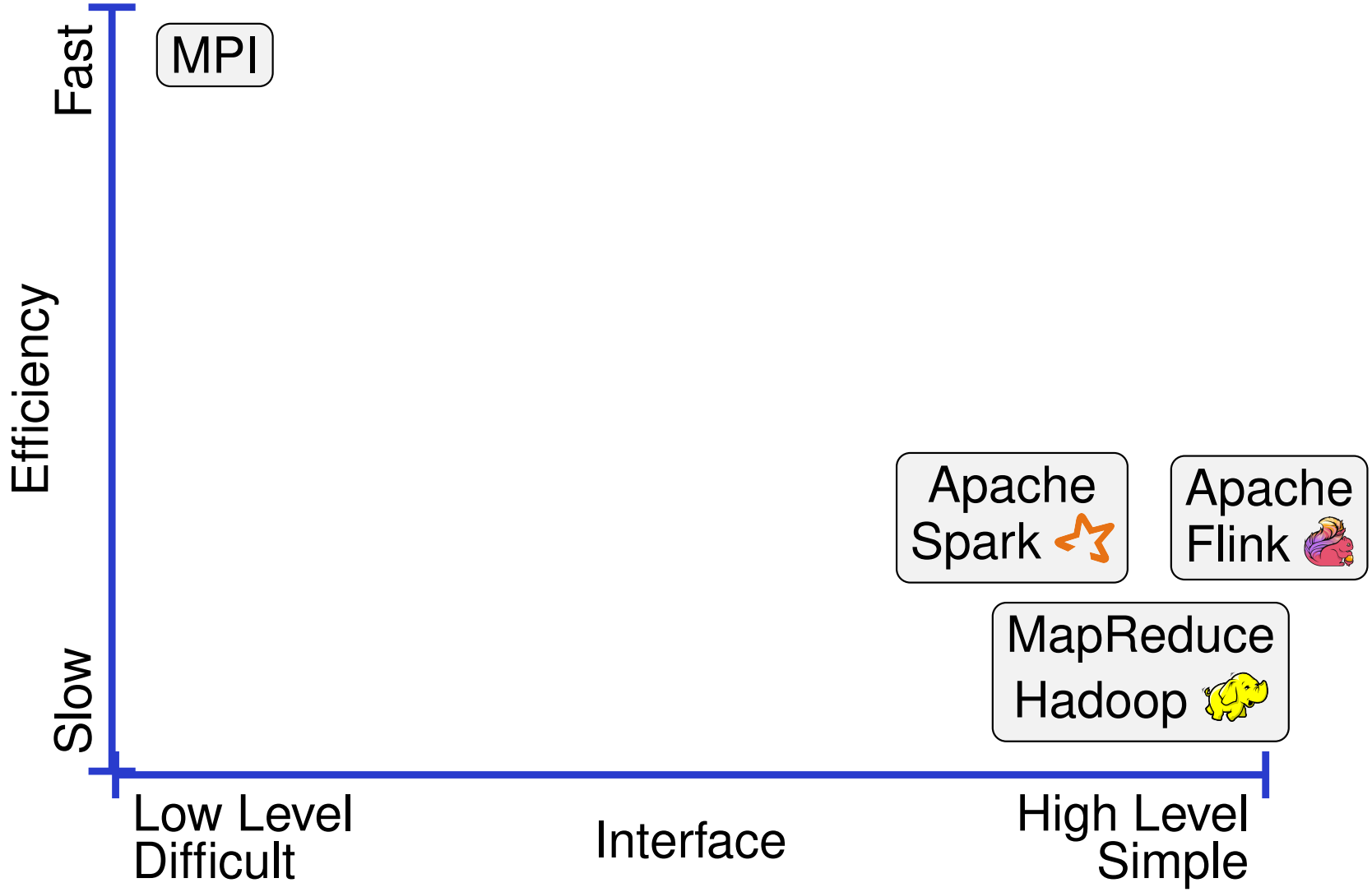
- Programming model for (highly) distributed system
- Implementations, e.g., in   and other experimental frameworks (some also in C++): (Boost.MapReduce, Sector/Sphere, mapreduce-lite).
- for calculations like: PageRank (spare matrix multiplication), parallel image processing, aggregation of statistics, machine learning, etc.
- NOT the same as distributed file systems or (distributed) databases   .
- And now, our's: Thrill  .

Why another Big Data Framework?

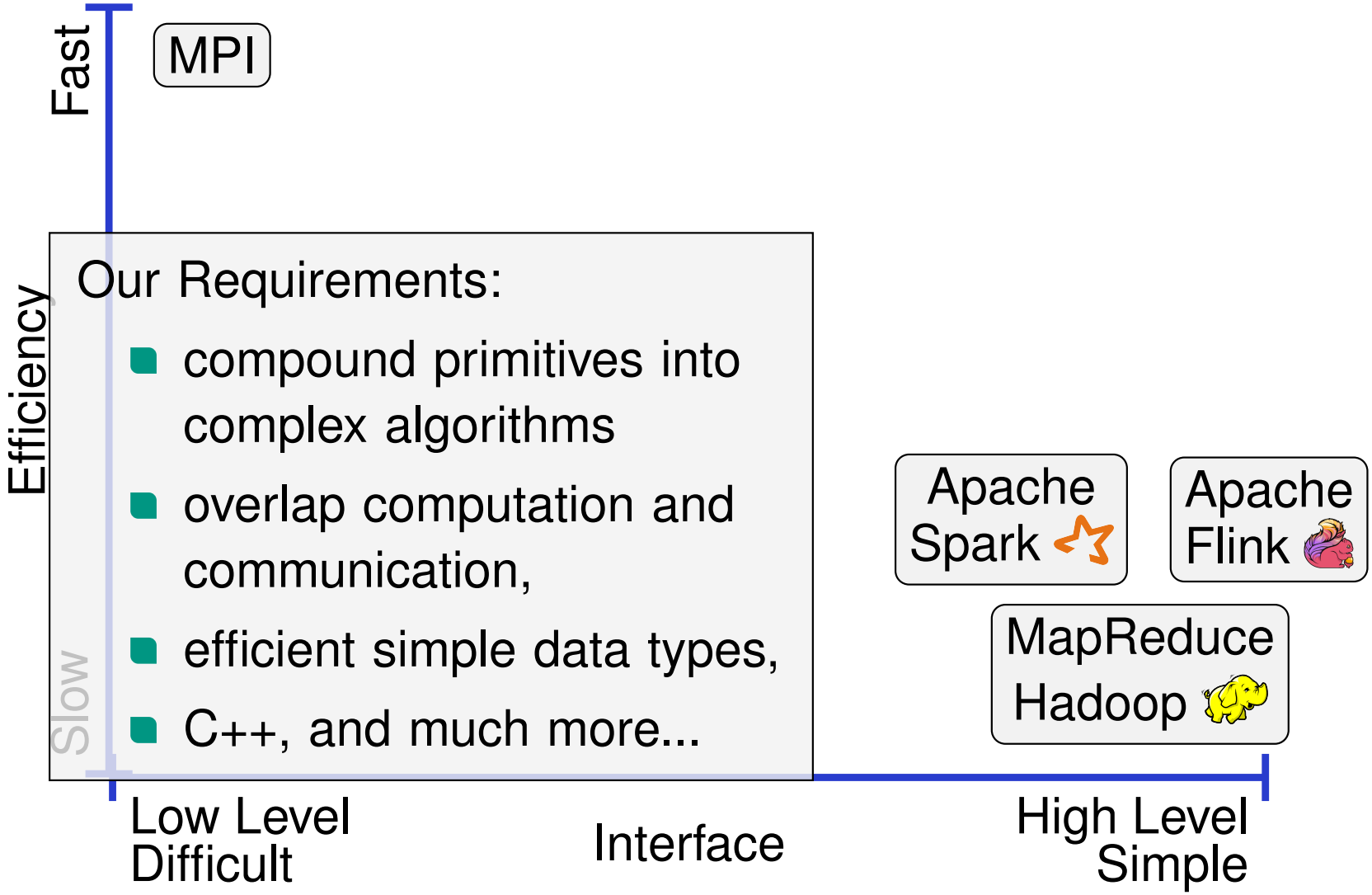
	Hadoop World Record	Spark 100 TB	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400	6592	6080
# Reducers	10 000	29 000	250 000
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	11.2 MB/sec	345 MB/sec	375 MB/sec
Daytona Rules	Yes	Yes	No
Environment	dedicated	EC2 (i2.8xlarge)	

source: `http://databricks.com/blog/2014/10/10/spark-petabyte-sort.html`

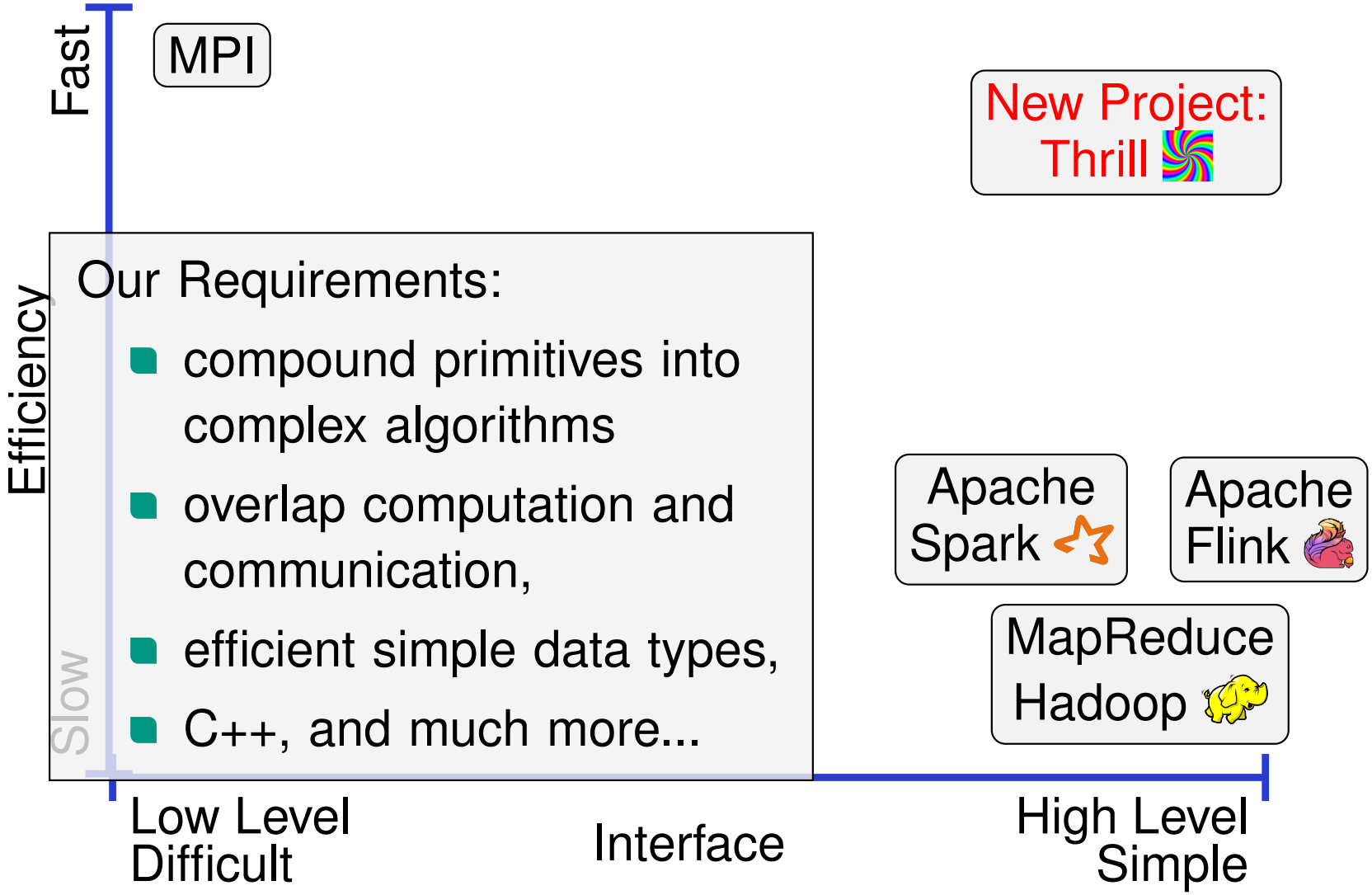
Big Data Batch Processing



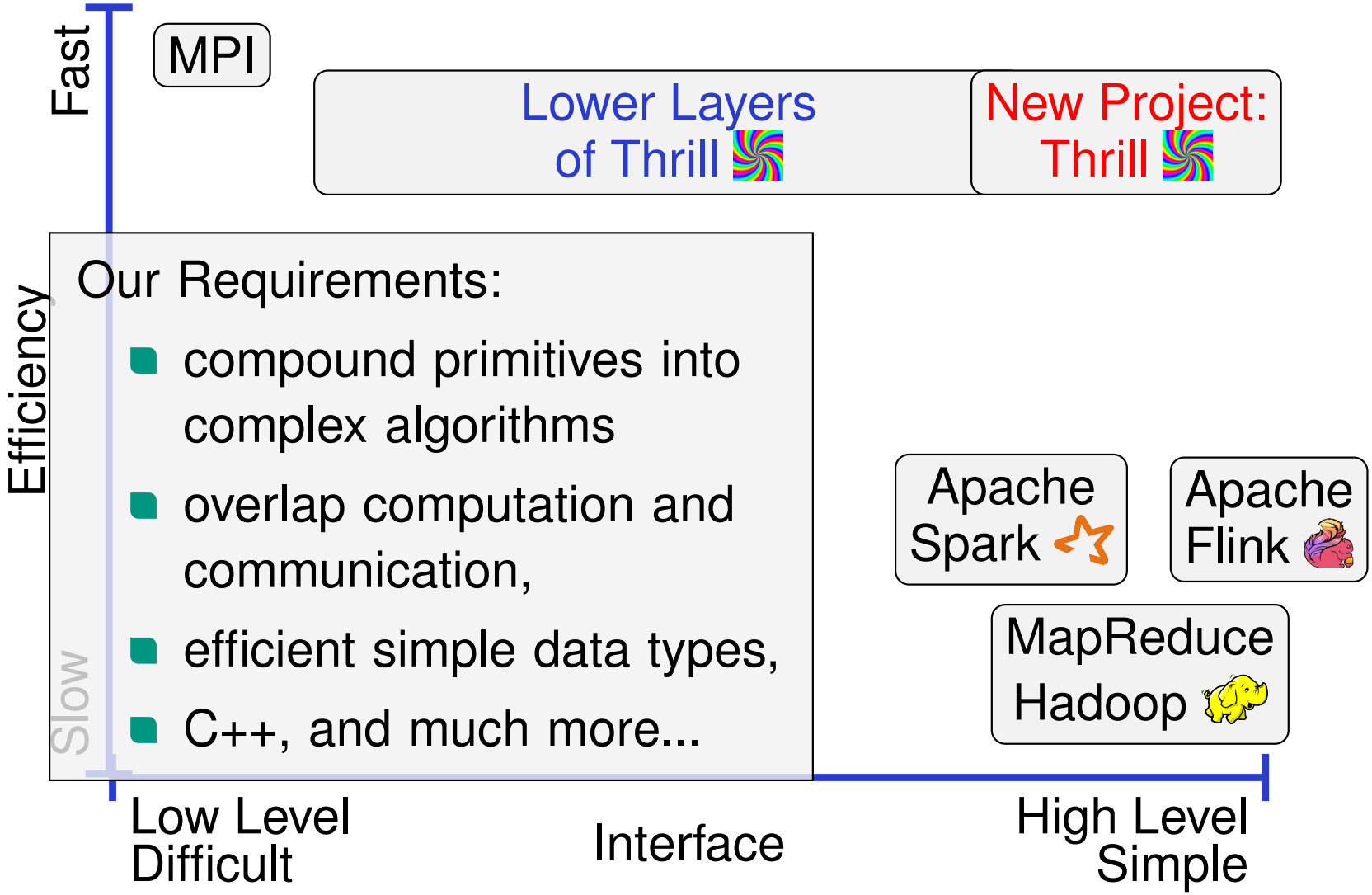
Big Data Batch Processing



Big Data Batch Processing



Big Data Batch Processing



Thrill's Design Goals

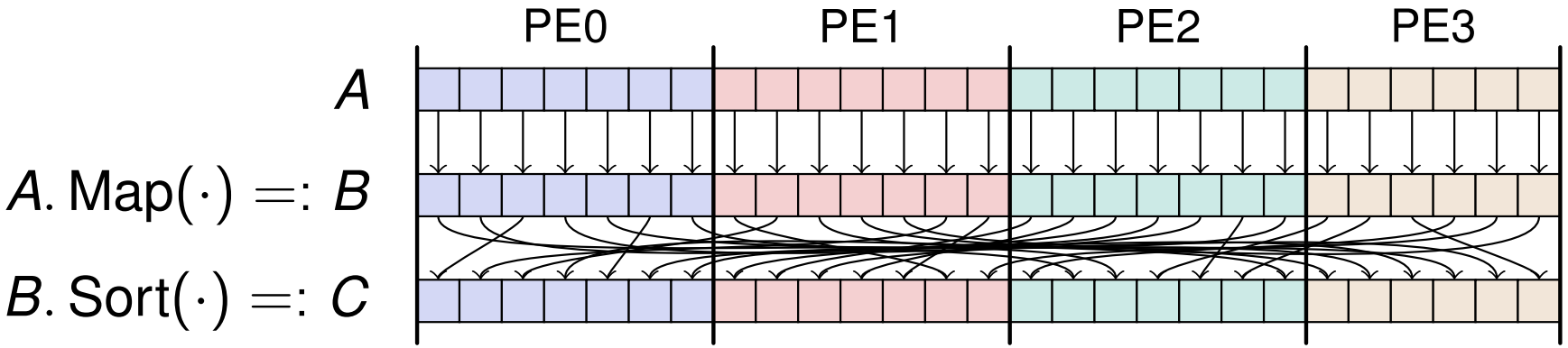
- Distributed arrays of small items (characters or integers).
- High-performance, parallelized C++ operations.
- Locality-aware, in-memory computation.
- Transparently use disk if needed
  $\Rightarrow$ external memory algorithms.
- Avoid all unnecessary round trips of data to memory (or disk).
- Optimize chaining of local operations.

Current Status:

- Open-Source at `http://project-thrill.org` and Github.
- Status: prototypes of many DOps work reasonably well.
- Near future: extension to distributed LCP array construction.

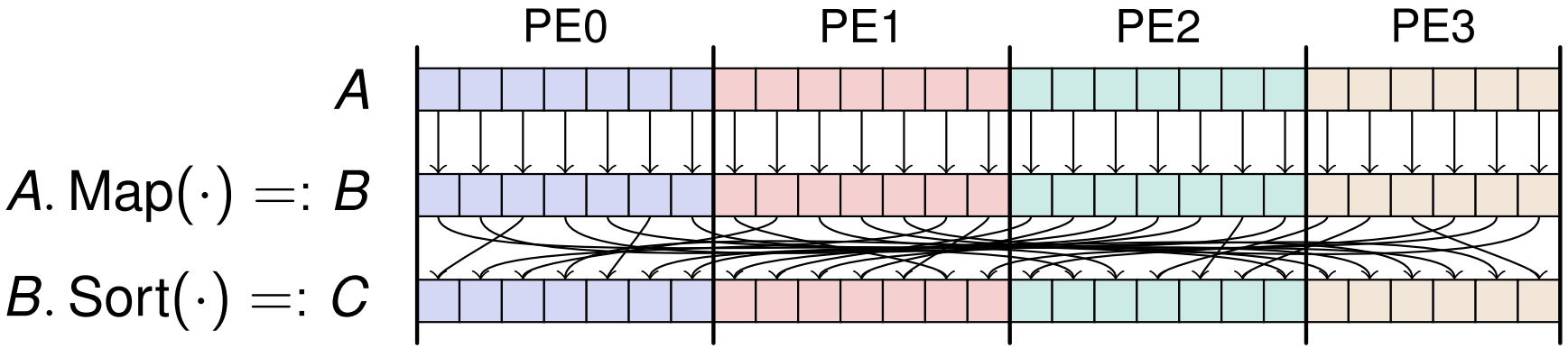
Distributed Immutable Array (DIA)

- User Programmer's View:
  - `DIA<T>` = result of an operation (local or distributed).
  - Model: distributed array of items T on the cluster
  - Cannot access items directly, instead use actions.



Distributed Immutable Array (DIA)

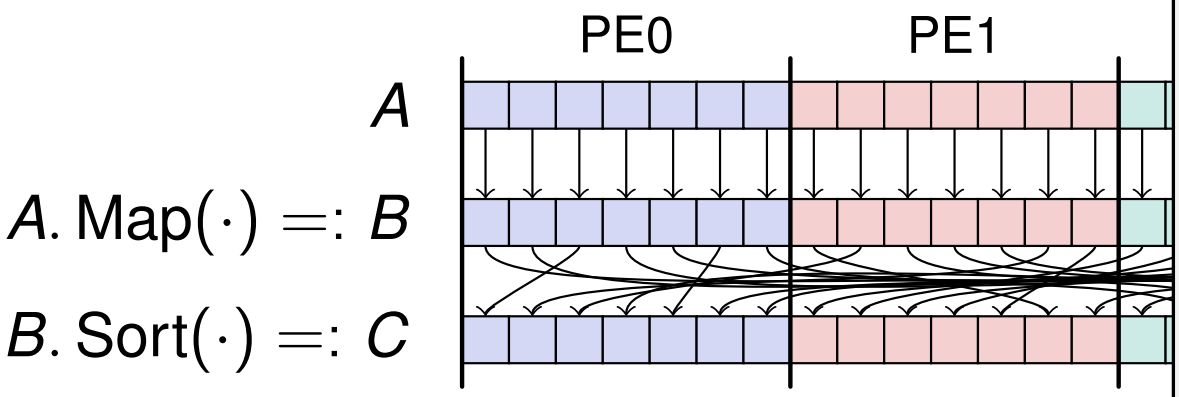
- User Programmer's View:
  - `DIA<T>` = result of an operation (local or distributed).
  - Model: distributed array of items T on the cluster
  - Cannot access items directly, instead use actions.

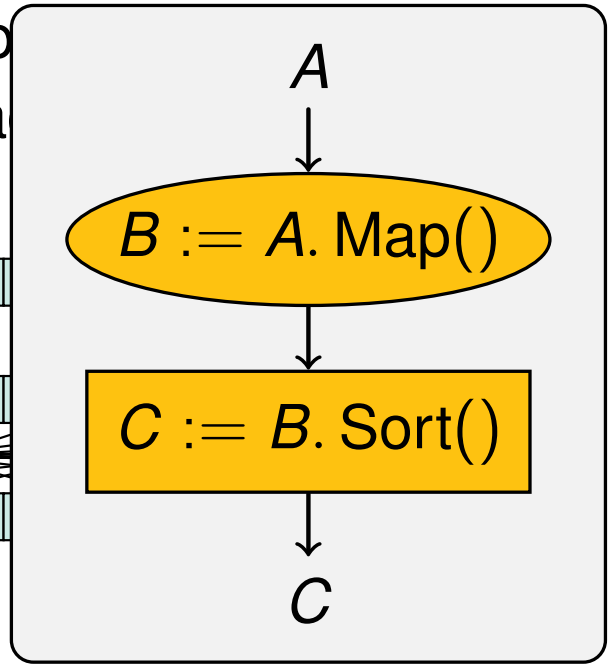


- Framework Designer's View:
  - Goals: distribute work, optimize execution on cluster, add redundancy where applicable. $\Longrightarrow$ build data-flow graph.
  - `DIA<T>` = chain of computation items
  - Let distributed operations choose "materialization".

Distributed Immutable Array (DIA)

- User Programmer's View:
  - `DIA<T>` = result of an operation (local or distributed).
  - Model: distributed array of items T o
  - Cannot access items directly, instea





- Framework Designer's View:
  - Goals: distribute work, optimize execution on cluster, add redundancy where applicable. $\Longrightarrow$ build data-flow graph.
  - `DIA<T>` = chain of computation items
  - Let distributed operations choose "materialization".

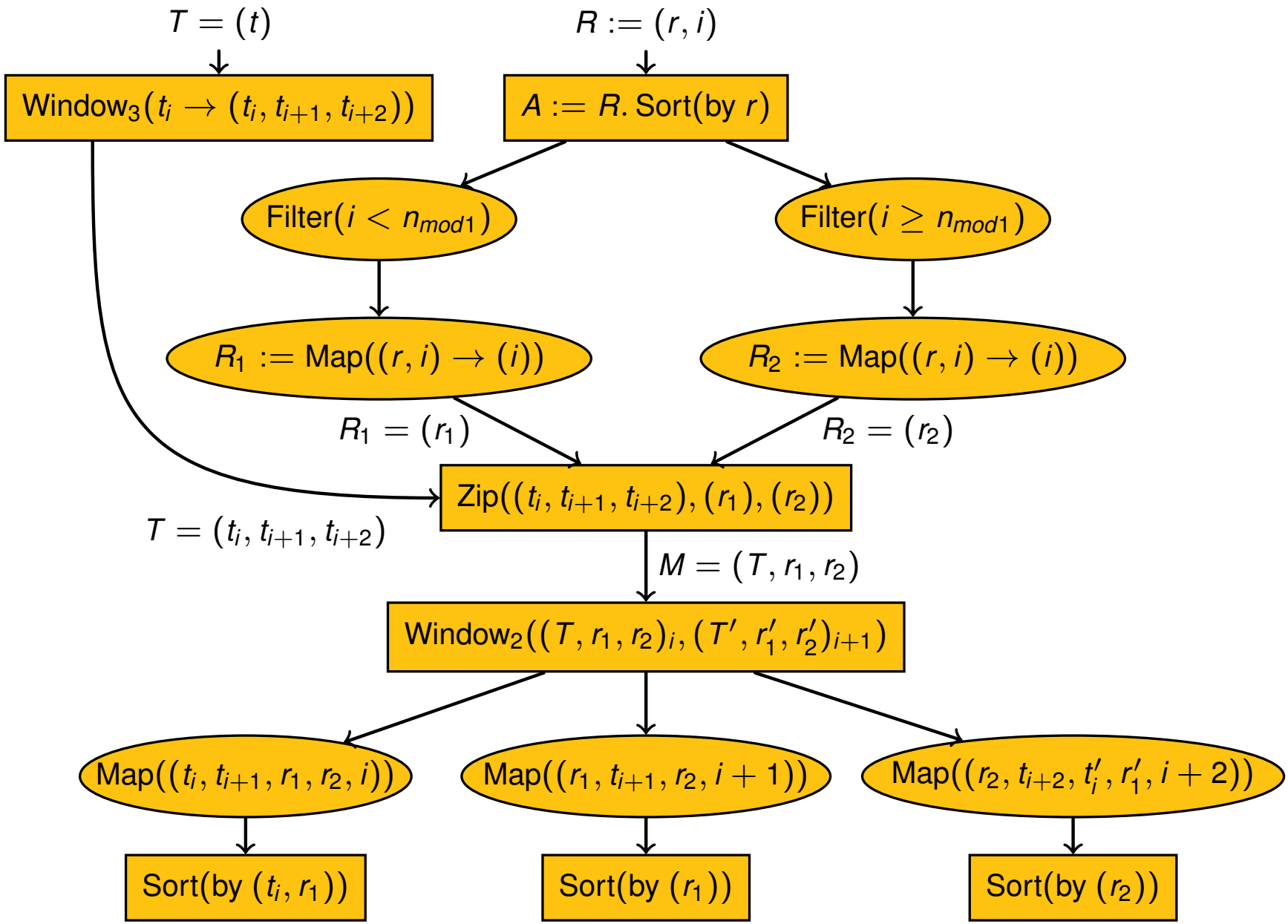
List of Primitives

- Local Operations (LOp): input is one item, output ≥ 0 items.
  Map(), Filter(), FlatMap().
- Distributed Operations (DOp): input is a DIA, output is a DIA.

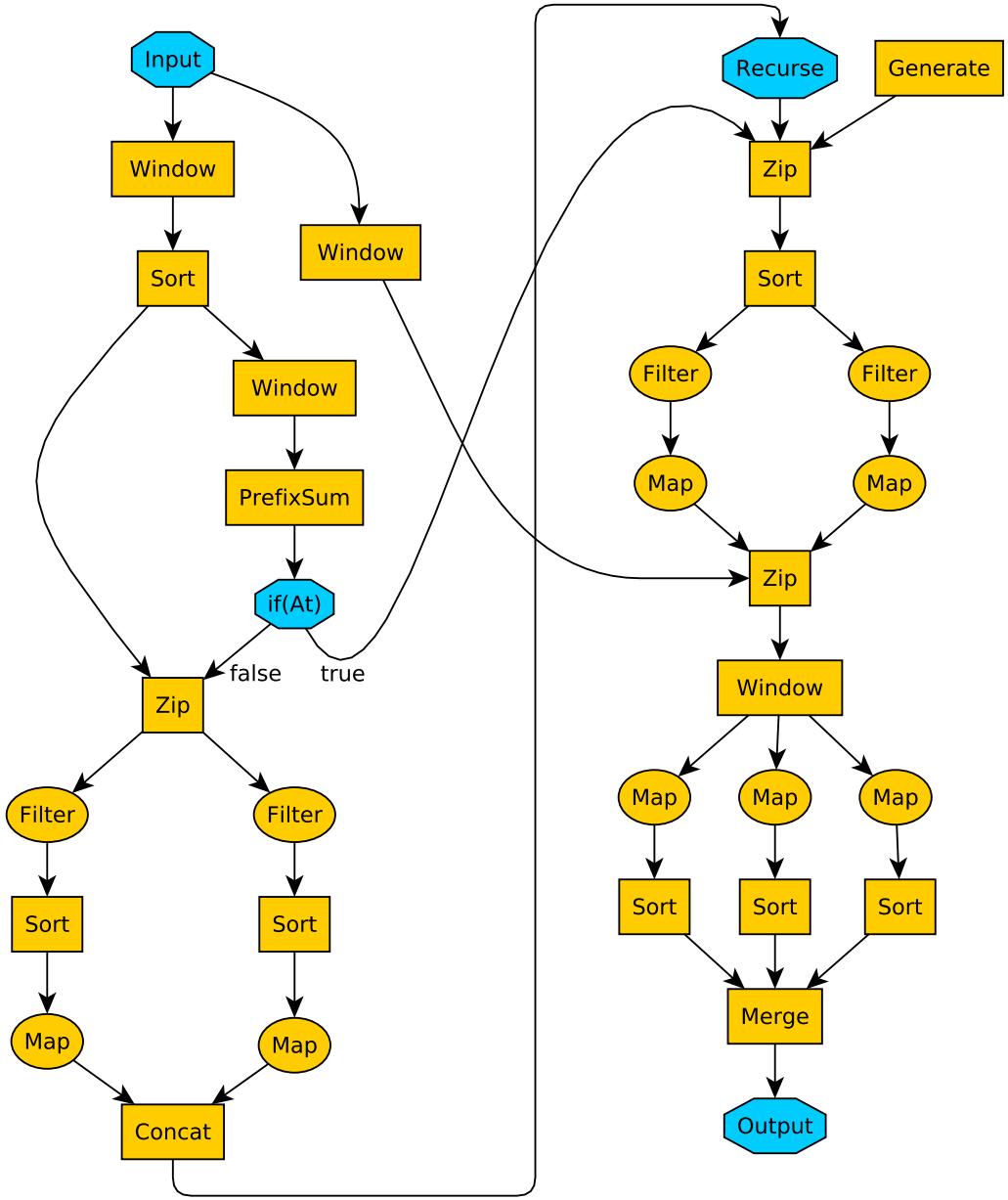
Sort()	Sort a DIA using comparisons.
ShuffleReduce()	Shuffle with Key Extractor, Hasher, and associative Reducer.
PrefixSum()	Compute (generalized) prefix sum on DIA.
Window$_k$()	Scan all k consecutive DIA items.
Concat()	Concatenate two or more DIAs of equal type.
Zip()	Combine equal sized DIAs item-wise.
Merge()	Merge equal typed DIAs using comparisons.

- Actions: input is a DIA, output: ≥ 0 items on master.
  At(), Min(), Max(), Sum(), Sample(), pretty much still open.

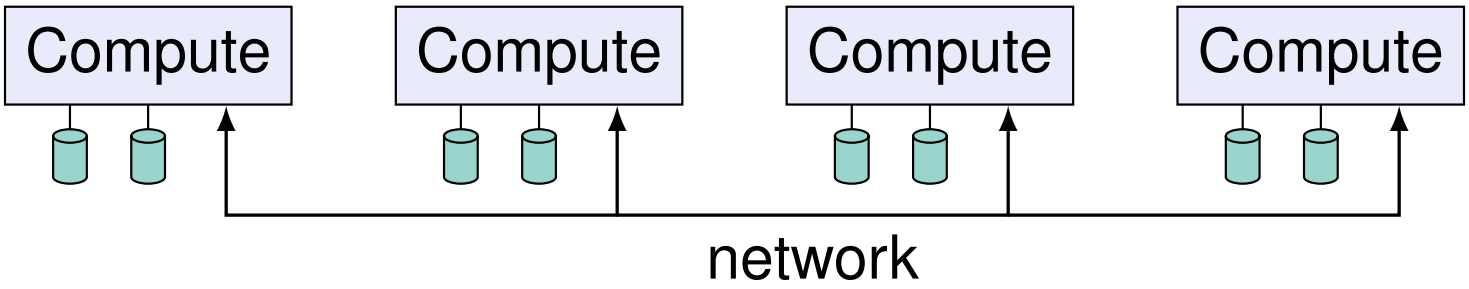
Exert of DC3's Data-Flow Graph



A Suffix Sorting Algorithm: DC3



Execution on Cluster



- Compile program into one binary, running on all nodes.
- Collective coordination of work on compute nodes, like MPI.
- Control flow is decided on by using C++ statements.
- Runs on MPI clusters and on Amazon's EC2 cloud.

Layers of Thrill

<table>
<tr><td colspan="2">api: High-level User Interface
DIA<T>, Map, FlatMap, Filter, Reduce, Sort, Merge, ...</td></tr>
<tr><td colspan="2">core: Internal Algorithms
reducing hash tables (bucket and linear probing), multiway merge, stage executor</td></tr>
<tr><td>data: Data Layer
Block, File, BlockQueue, Reader, Writer, Multiplexer, Streams, BlockPool (paging)</td><td rowspan="2">net: Network Layer
(Binomial Tree) Broadcast, Reduce, AllReduce, Async-Send/Recv, Dispatcher
Backends: mock, TCP, MPI</td></tr>
<tr><td>io: Async File I/O
borrowed from STXXL</td></tr>
<tr><td>common: Common Tools
Logger, Delegates, Math, ...</td><td>mem: Memory Limitation
Allocators, Counting</td></tr>
</table>

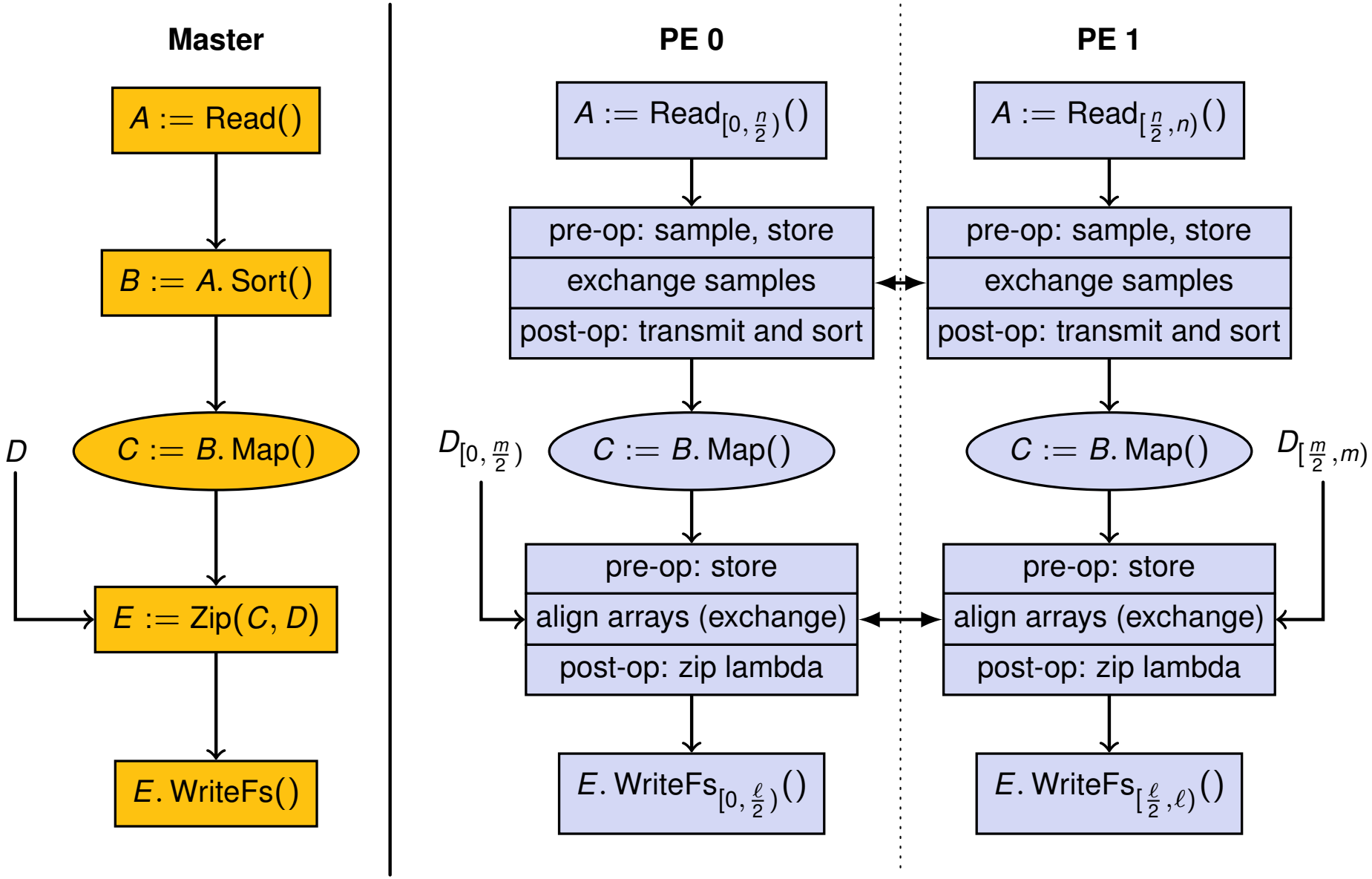
Parallel Algorithms Lecture and Thrill

Thrill contains many things you learned in this lecture:

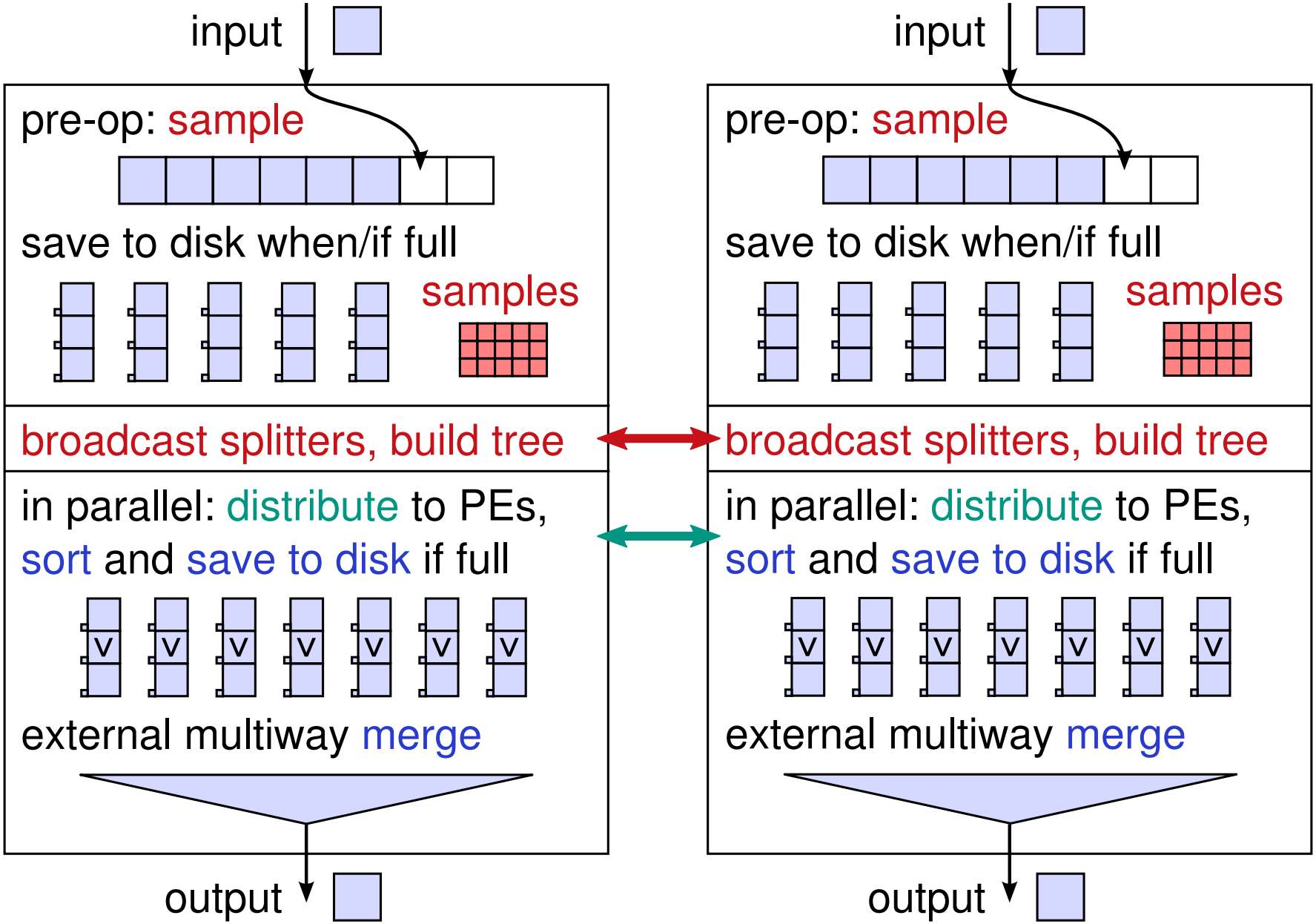
- DOps are BSP-style communications primitives. LOps are inlined into them.
- Network layer's Broadcast(), Reduce(), etc are binomial tree algorithms. Prefixsum() is a hypercube algorithm.
- Sort() is a distributed external sample sort.
- Merge() uses distributed multisequence selection to balance result.

In future (like bachelor/master theses): add more advanced things like distributed hashing, Bloom filters, etc to Thrill.

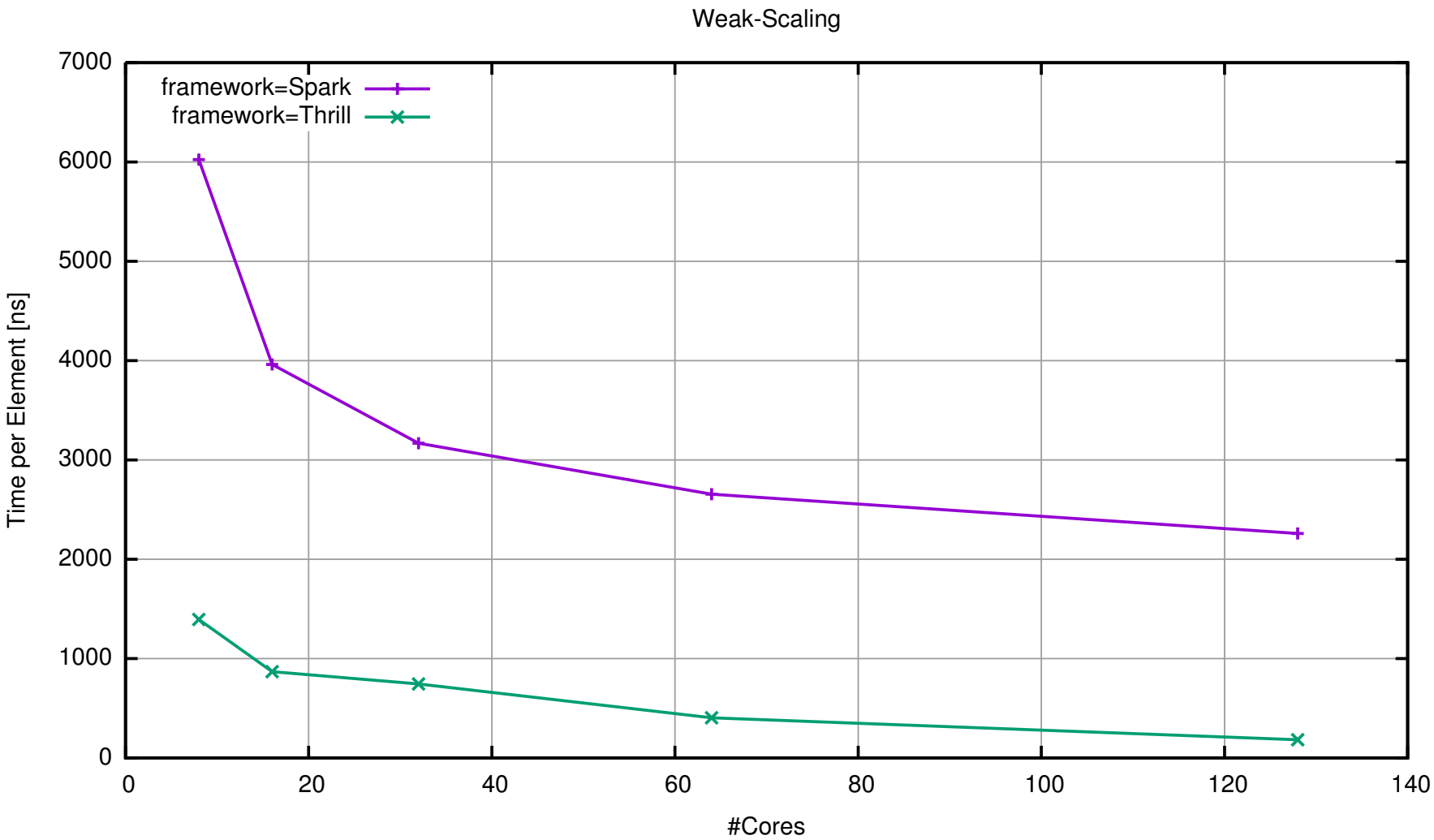
Mapping Data-Flow Nodes to Cluster



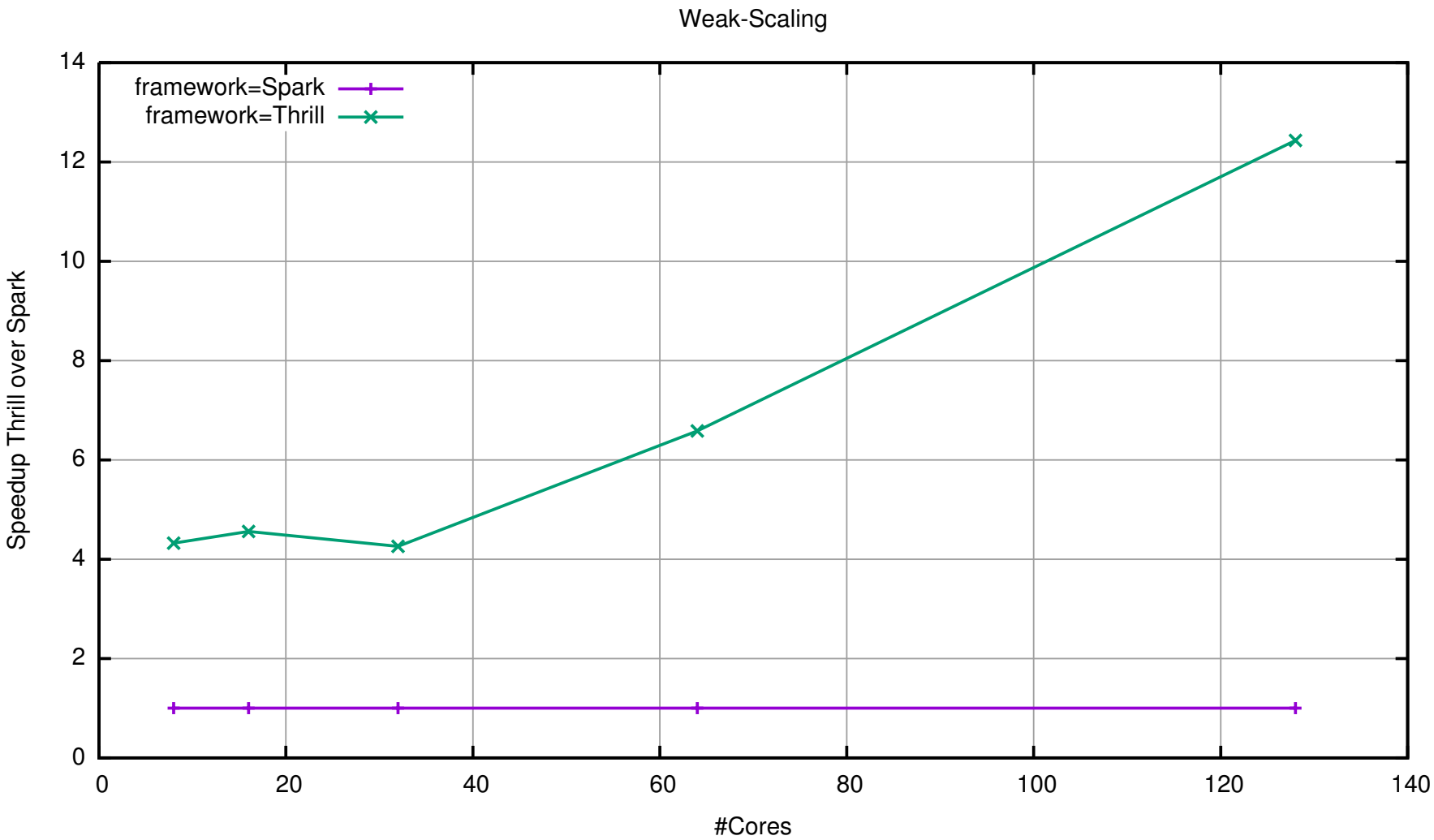
Sorting DOp


input
pre-op: sample
save to disk when/if full
samples
broadcast splitters, build tree
in parallel: distribute to PEs,
sort and save to disk if full
external multiway merge
output
input
pre-op: sample
save to disk when/if full
samples
broadcast splitters, build tree
in parallel: distribute to PEs,
sort and save to disk if full
external multiway merge
output

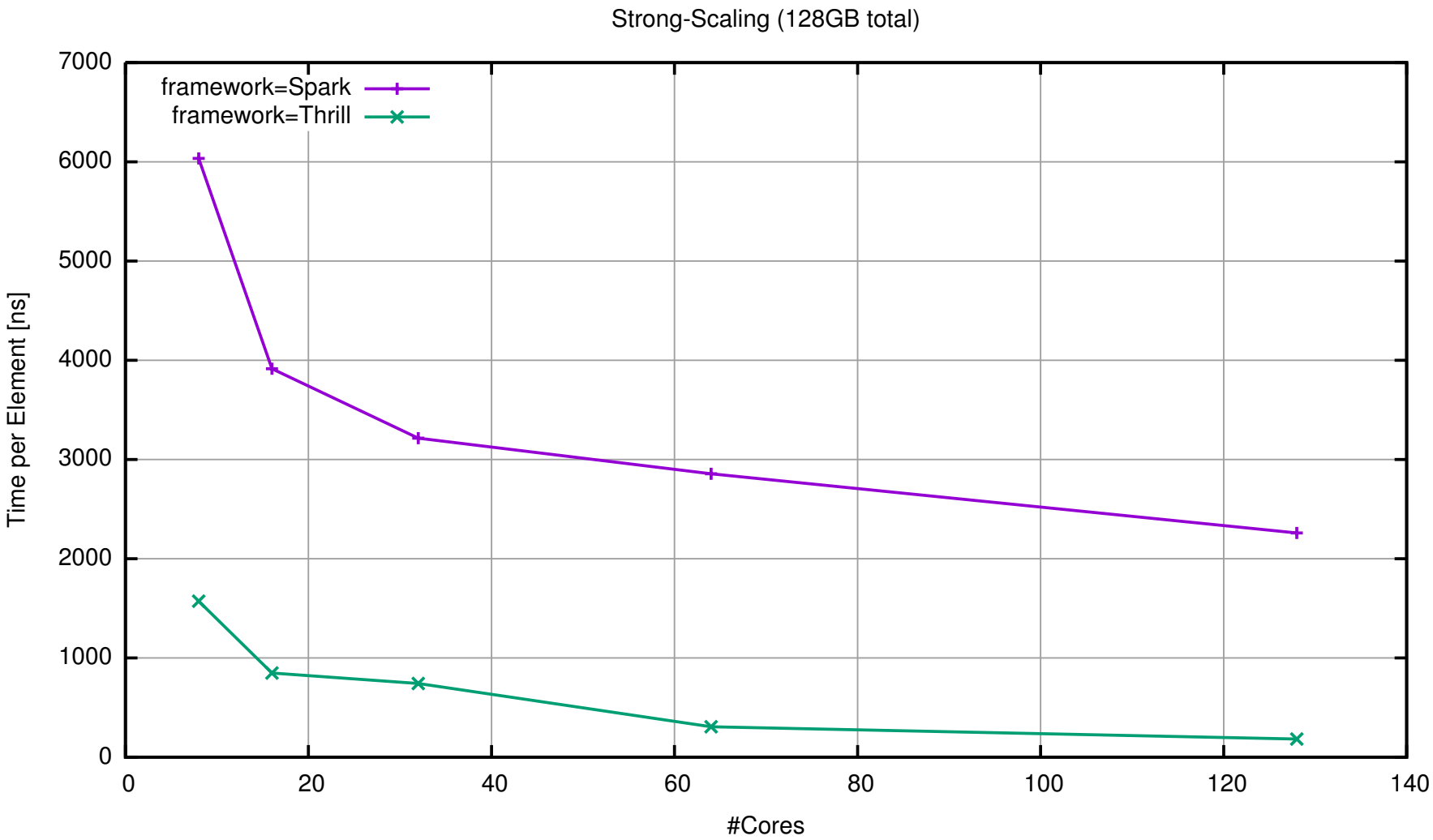
Timing: Word-Count Weak-Scaling



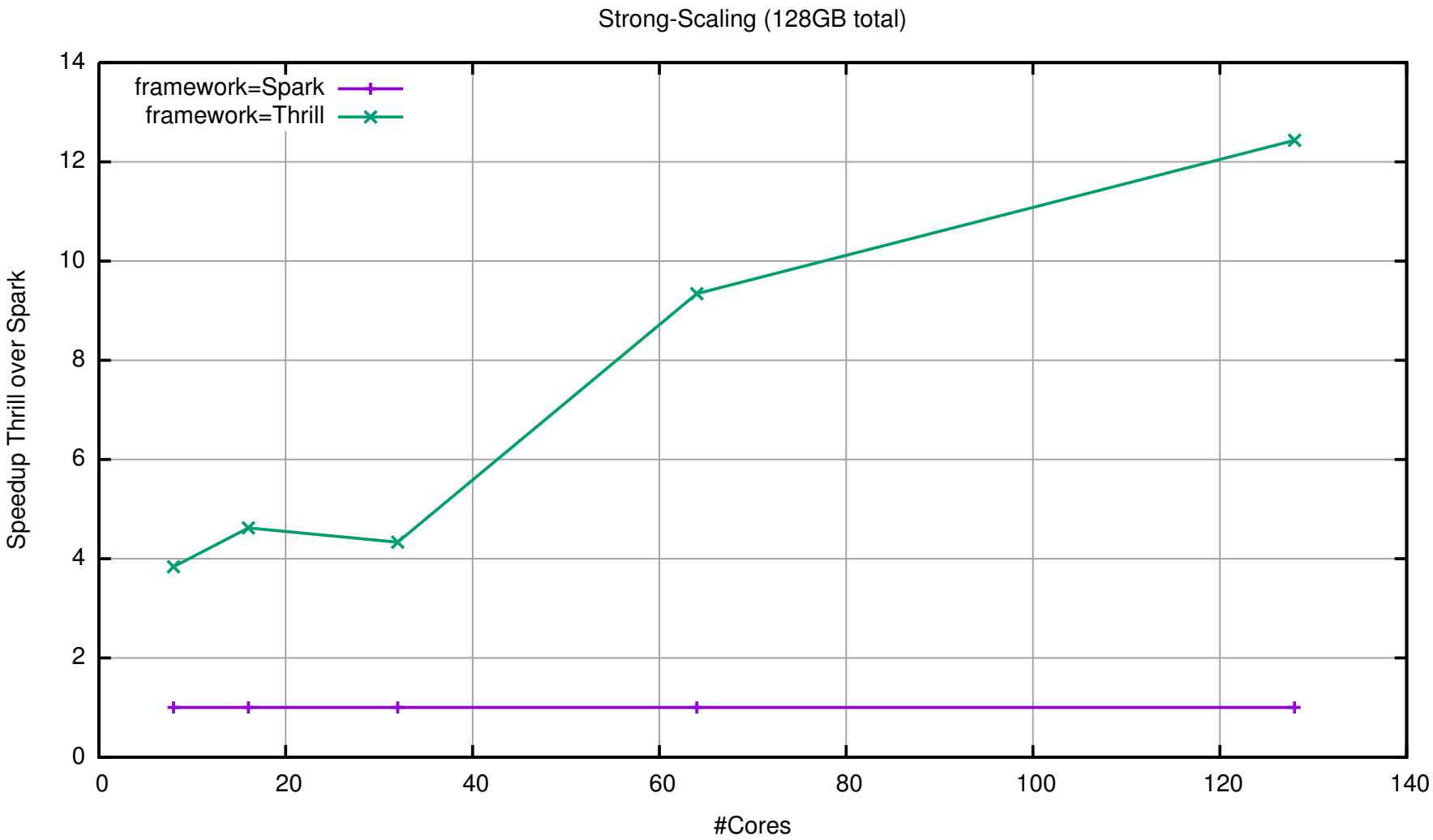
Speedup: Word-Count Weak-Scaling



Timing: Word-Count Strong-Scaling



Speedup: Word-Count Strong-Scaling



Current and Future Work

- Open-Source at http://project-thrill.org and Github.
- High quality, very modern C++14 code.

Some Master thesis ideas:

- Distributed rank()/select() and wavelet tree construction.
- Distributed query processing.
- Communication efficient distributed operations for Thrill.

Thank you for your attention!

Questions?