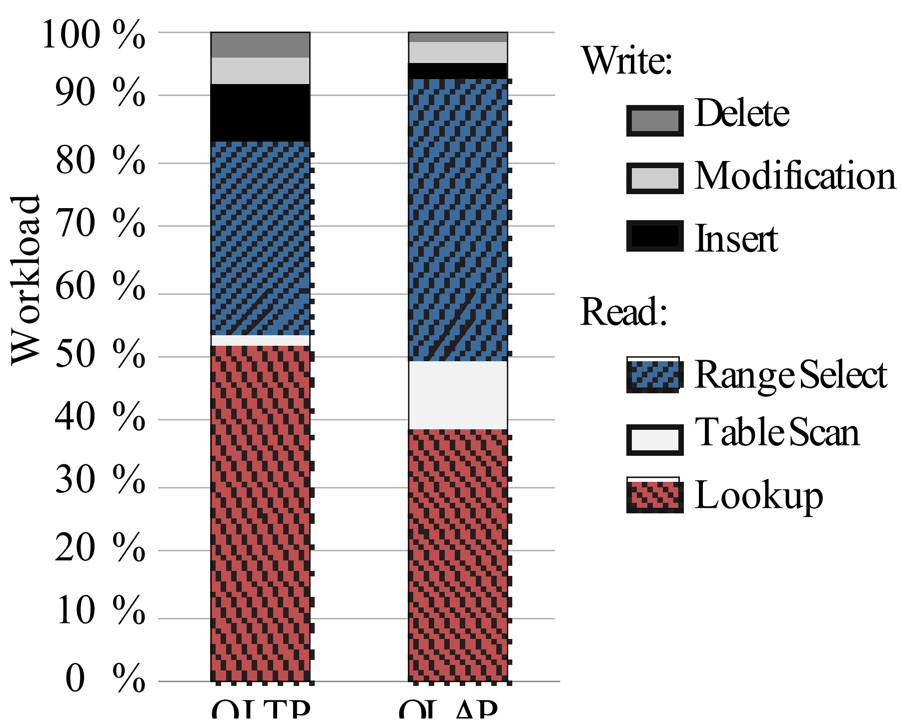
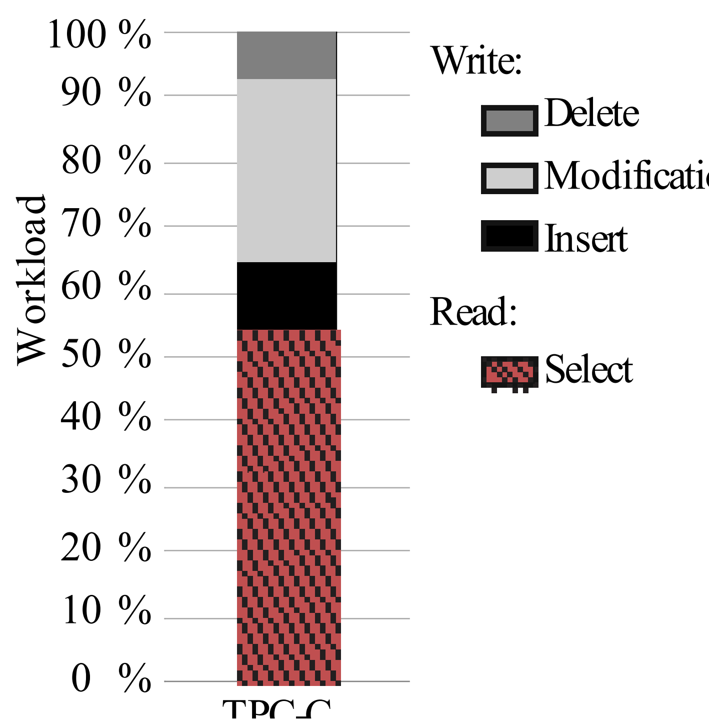
Applicability of GPU Computing for Efficient Merge in In-Memory Databases

Jens Krueger, Martin Grund, Ingo Jaeckel,
Alexander Zeier, Hasso Plattner
Hasso Plattner Institute for IT Systems Engineering
University of Potsdam
Potsdam, Germany

Introduction (1)

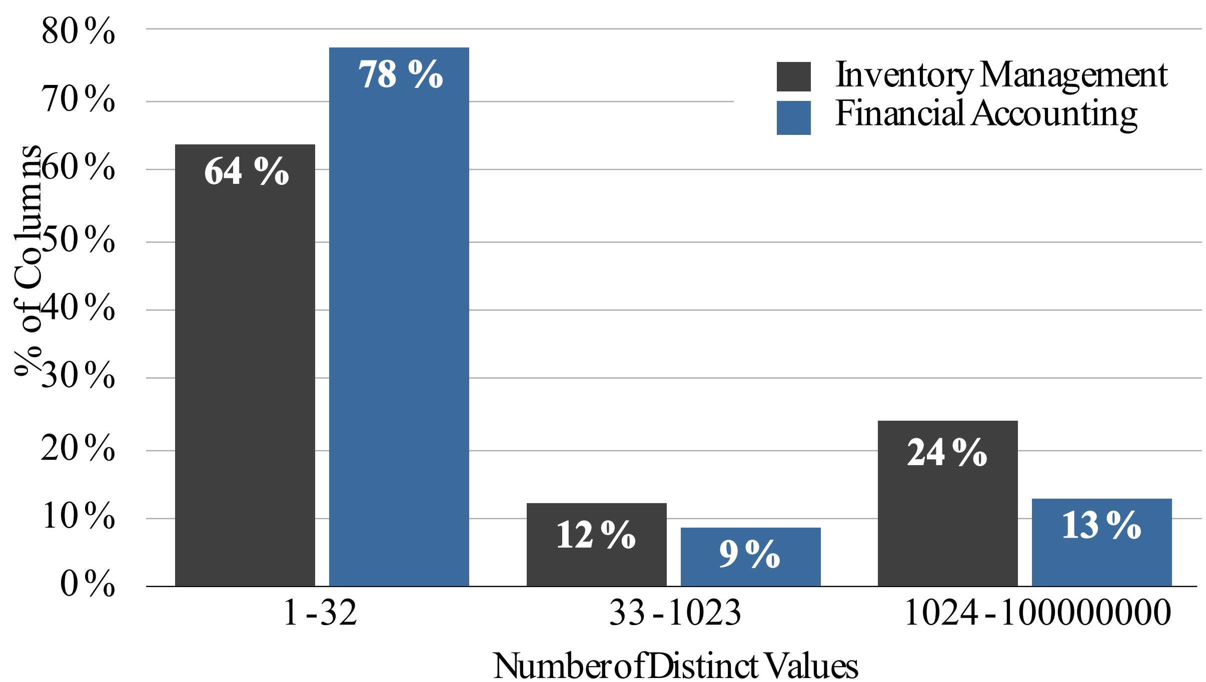
- Enterprise applications have evolved: not just OLAP vs. OLTP
- Range selects occur often
- Real world is more complicated than single tuple access





Introduction (2)

- Enterprise data is wide and sparse
- Most columns are empty or have a low cardinality of distinct values
- Sparse distribution facilitates high compression



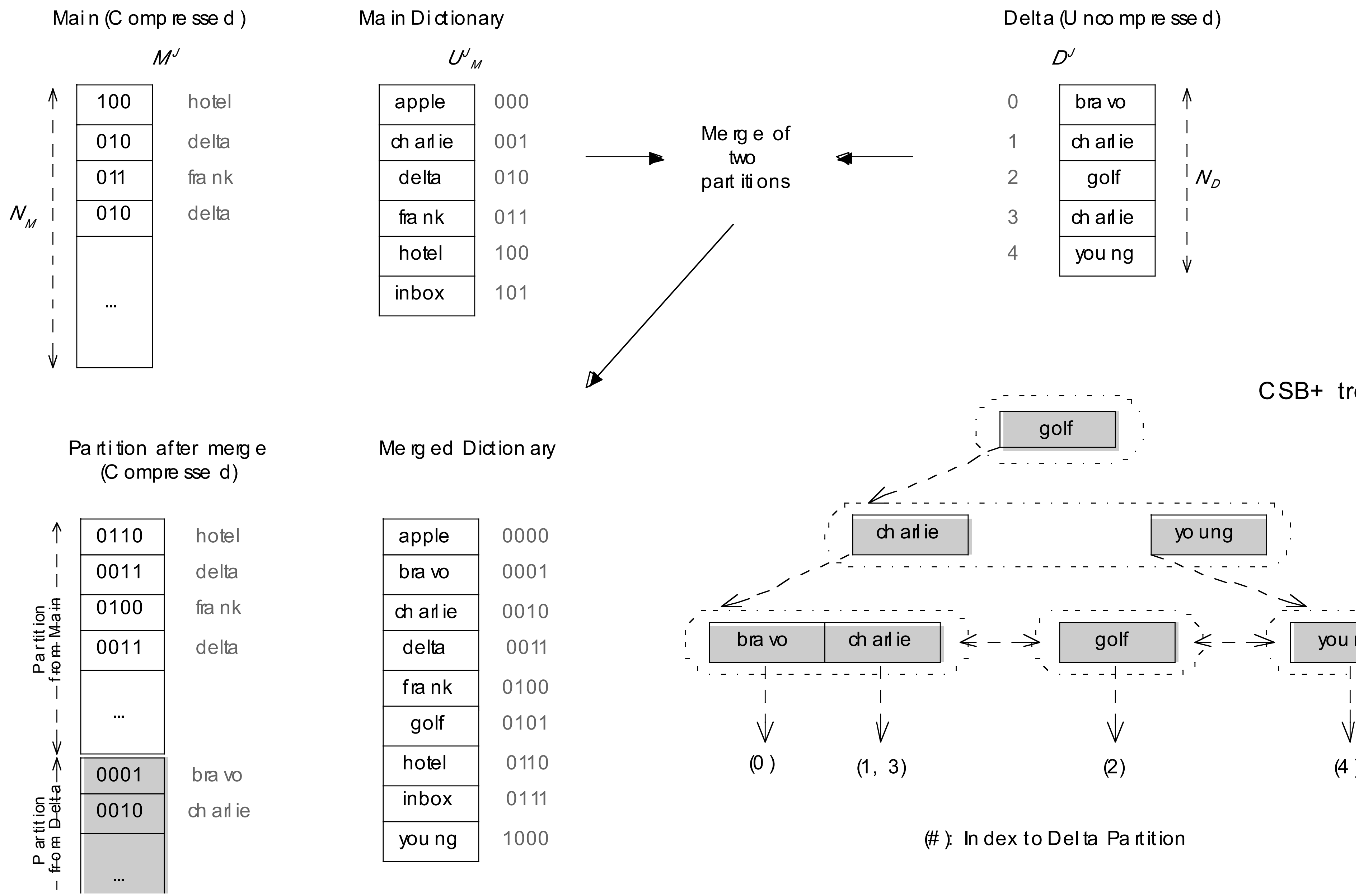
System Overview

- Based on **HYRISE**:
  In-memory compressed vertical partitionable database engine
  - Completely in main memory
  - Organizes data column-oriented
  - Applies dictionary compression with a order-preserving dictionary and a bit-compressed attribute vector
  - Uses a differential store concept to support data modifications
- Efficiently executes both OLTP and OLAP requests on structured enterprise data

Terminology

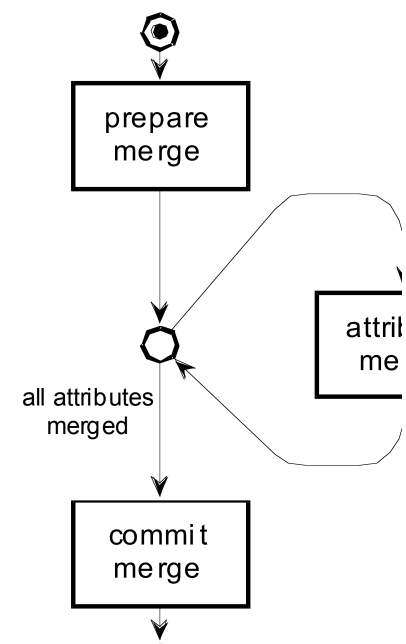
- **Table**: A relation table with NC columns, with one writ (delta) and one read-optimized (main) partition.
- **Update**: Any modification operation on the table resulting in an entry in the delta partition.
- **Main Partition**: Compressed and read-optimized part o the column. Consists of a ***order-preserving dictionary*** and a ***attribute vector*** with ***bit-compressed value ids***
- **Delta Partition**: Uncompressed write-optimized part of the column where all updates are stored until the merg process is completed.
- **Merge Process**: Applies compression to delta and mai partition to create new main partition

System Overview (2)



Merge Process

- Transfer **updates** from uncompressed delta partition into main partition
- Requirements
  - has to be performed while the system is operational, … hence works on a copy
  - minimal time of increased resource utilization
- Phases:
  - Prepare merge
  - Attribute merge
    1. Merge dictionaries
       1.a Build delta dictionary
       1.b Merge main and delta dictionary
    2. Update compressed values
       2.a Compute new compressed value length
       2.b Create new compressed main
- Commit merge



Attribute Merge

For the j^{th} column, the input for the merging algorithm consists of $\mathbf{M}^j$, $\mathbf{D}^j$ and $\mathrm{U}^j_{\mathrm{M}}$, while the output consists of $\mathbf{M}'^j$ and $\mathrm{U}'^j_{\mathrm{M}}$.

Runtime complexity:

Step 1: $|\mathbf{U}'^j_{\mathbf{M}}| = |\mathbf{D}^j \cup \mathbf{U}^j_{\mathbf{M}}|$

Step 2: $N_{M'} = N_M + N_D$

As Step 2 is already bandwidth bound [1], we focus on Step

[1] Krueger et al., "Fast Updates on Read-Optimized Databases Using Multi-Core CPUs," VLDB 2012 (t

Motivation / Trade-offs

- GPUs offer up to two orders of magnitude more cores than a CPU
  - Increases the maximum possible speedup through parallelization accordingly
- **But:** the in-/output data needs to be transferred over the PCI-Express bus which has a limited bandwidth
  - to be faster, GPU implementations have to finish befor CPU implementations **including the data transfers**

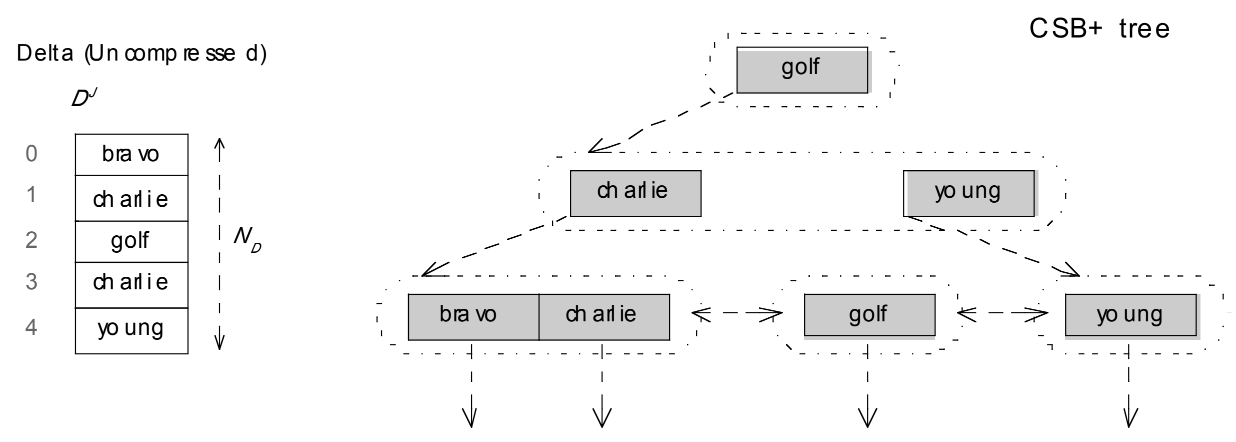
NVIDIA Thrust

- *Thrust* is a CUDA library of parallel algorithms resembling the C· STL
- **Assumption**:
  An implementation that uses operations provided by a matu CUDA library can provide better performance than a custon made CUDA kernel
  - e.g. thrust::sort, thrust::unique, thrust::reduce, thrust::lower_bound

Year	Device	Rate	
		CPU	GPU
2009	GTX 280		200
2010	Knights Ferry vs. GTX 280	**560**	176
2010	Core i7 vs. GTX 280	250	200
2009	Tesla C1060		300
2010	GTX 285		550
2011	GTX 480		**1005**

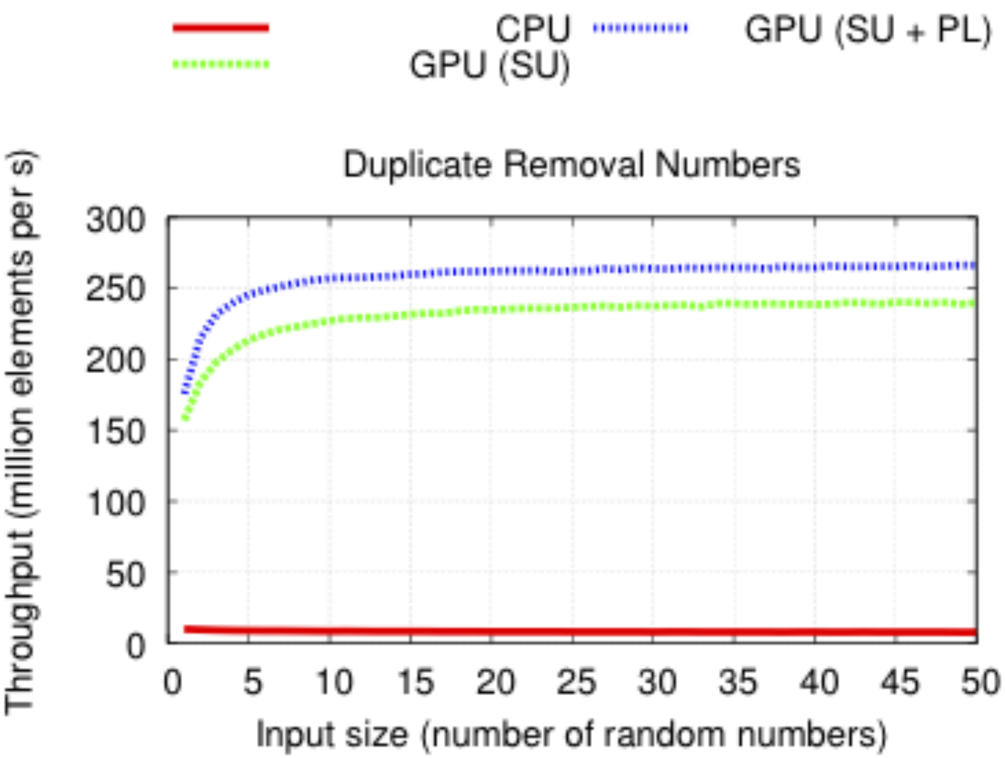
GPU Duplicate Removal

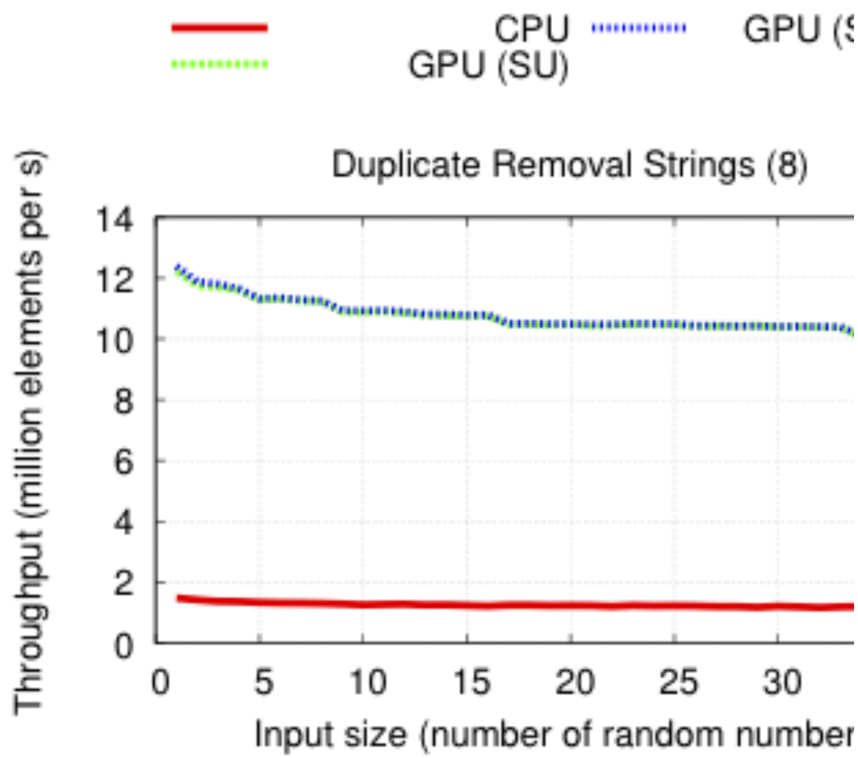
- **Trade-off:** Delta partition insert costs vs. costs for creatir delta dictionary during merge process
- Assuming that inserting into the CSB+ structure is too expensive in insert/update-intensive workloads
- Without the CSB+ structure duplicates have to be remove to create a dictionary



Duplicate Removal (2)

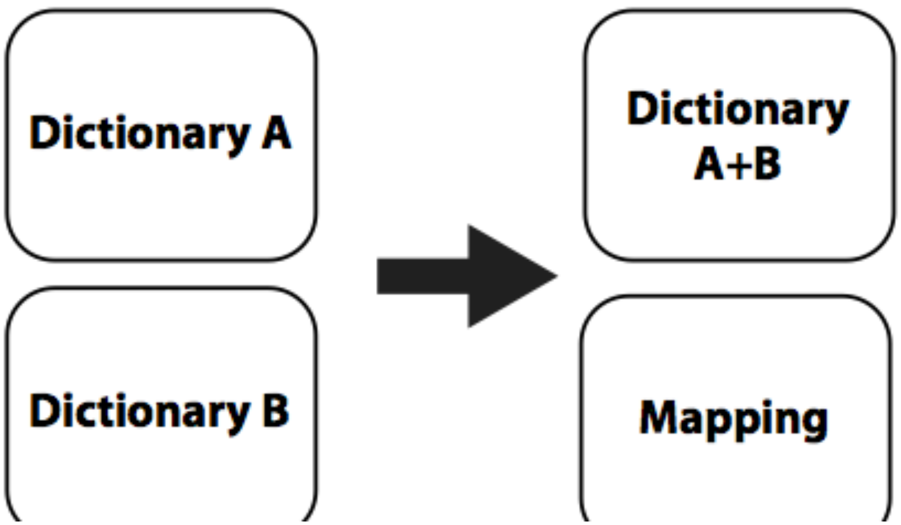
- Remove duplicates by sorting and removing subsequent duplicates with *thrust::sort* and *thrust::unique*
- Up to 27 times faster than naïve *std::sort* and *std::unique*





Dictionary Merge

- Propose a custom kernel for merging dictionaries
  - Block-Wise Parallel Slice Merge (**BWS**)
- … And *Thrust*-supported approaches that reuses the duplicate removal approach
  - Concatenate-Sort-Unique-Binary Search (**CSUBS**)
  - Merge-Unique-Binary-Search (**MUBS**)

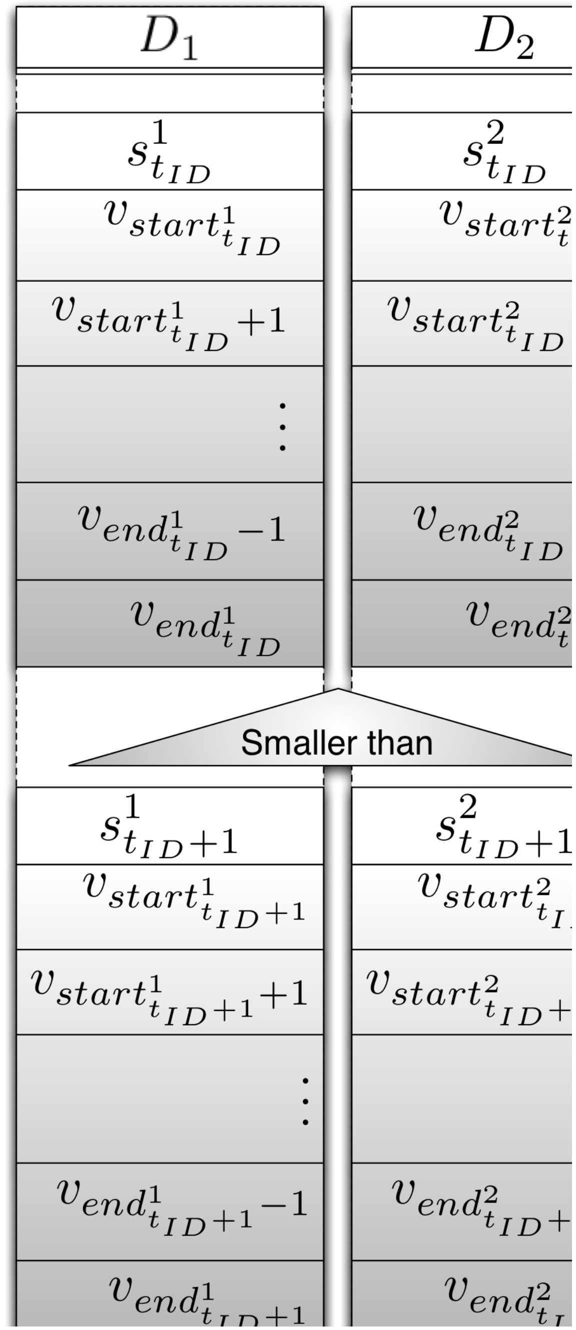


BWS Merge

- Merge two sorted lists
- All values in a list are distinct
- **But:** a value can appear in both lists at the same time

Idea:

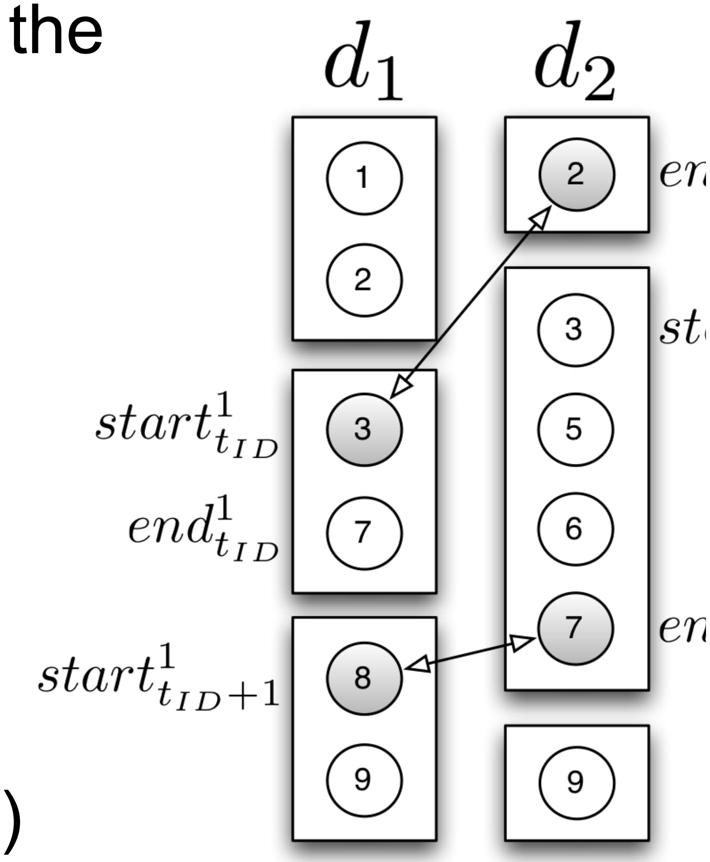
- ○ Partition both input lists into slices
- ○ All values of a slice are smaller than the values of the subsequent slice
- ○ Static partitioning is not sufficient since it allows duplicates



BWS Merge (2)

- First list: partition into equally sized slices
- Second list: determine boundaries of the slices with binary search

- Partition both dictionaries (CPU)
- Merge slices (GPU)
  - Determine number of unique values per thread block
  - Inform other threads of local unique value count
  - Write unique values
- Concatenate block-wise output (CPU)
- Use parallel binary search to fill auxiliary structures or: set them on the GPU



CSUBS Merge

- **C**oncatenate, **S**ort, **U**nique, **B**inary **S**earch
- Use *Thrust* **primitives** to implement dictionary merge:
  - Concatenate dictionaries in GPU memory with thrust::copy
  - Sort concatenated dictionaries with *thrust::sort*
  - Remove subsequent duplicates with *thrust::unique*
  - Map values to their new position with *thrust::lower_bou*
    - create auxiliary structures with binary search for eac value of both dictionaries

MUBS Merge

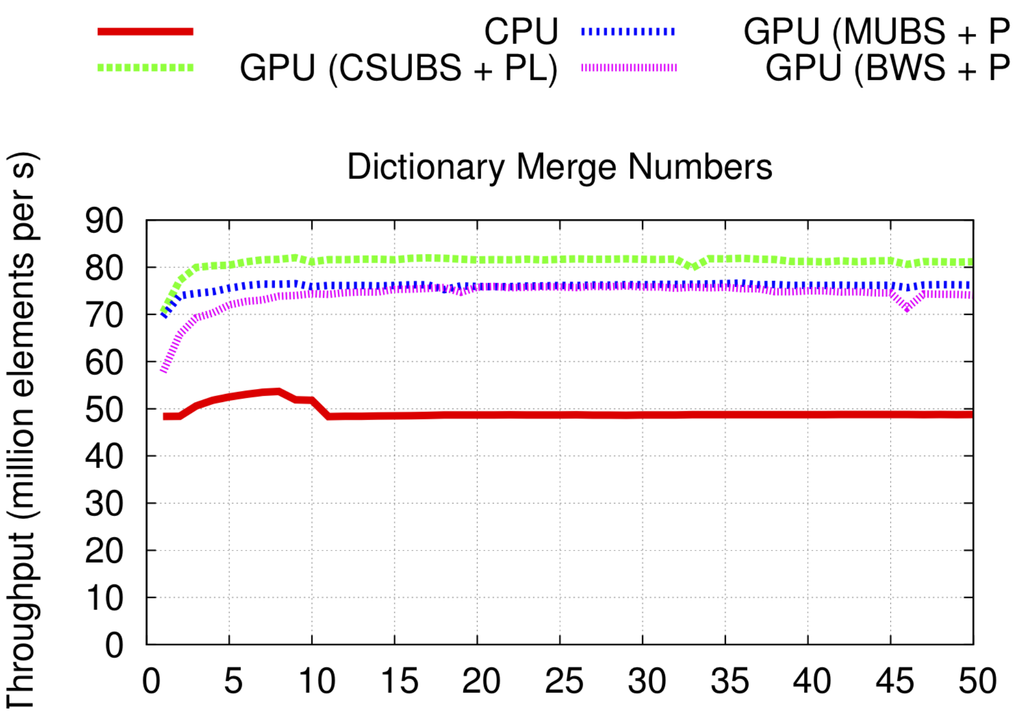
- **M**erge, **U**nique, **B**inary **S**earch
- CSUBS approach does not exhibit the fact that both lists a already sorted
- Rather than concatenating and sorting use *Thrust*'s merge primitive
  - Merge both sorted lists into a new list *thrust::merge*
  - Remove subsequent duplicates with *thrust::unique*
  - Map values to their new position with *thrust::lower_bou*

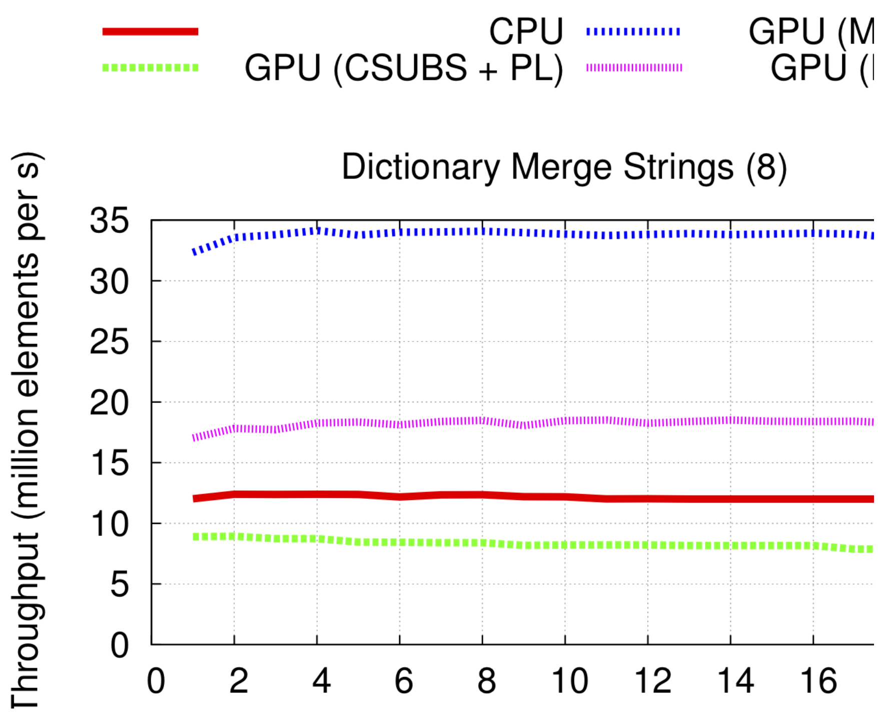
Evaluation - Environment

- GPU: Tesla C2050 GPU, 3GB memory
- CPU: single core of a Xeon E5620 processor as baseline
  - Used STL implementations, e.g. *std::sort, std::unique*, a
    the default merge implementation applied in HYRISE
- Data:
  - Single column
  - 32-bit integer values and
  - Strings with a length of up to eight characters

Evaluation - Results

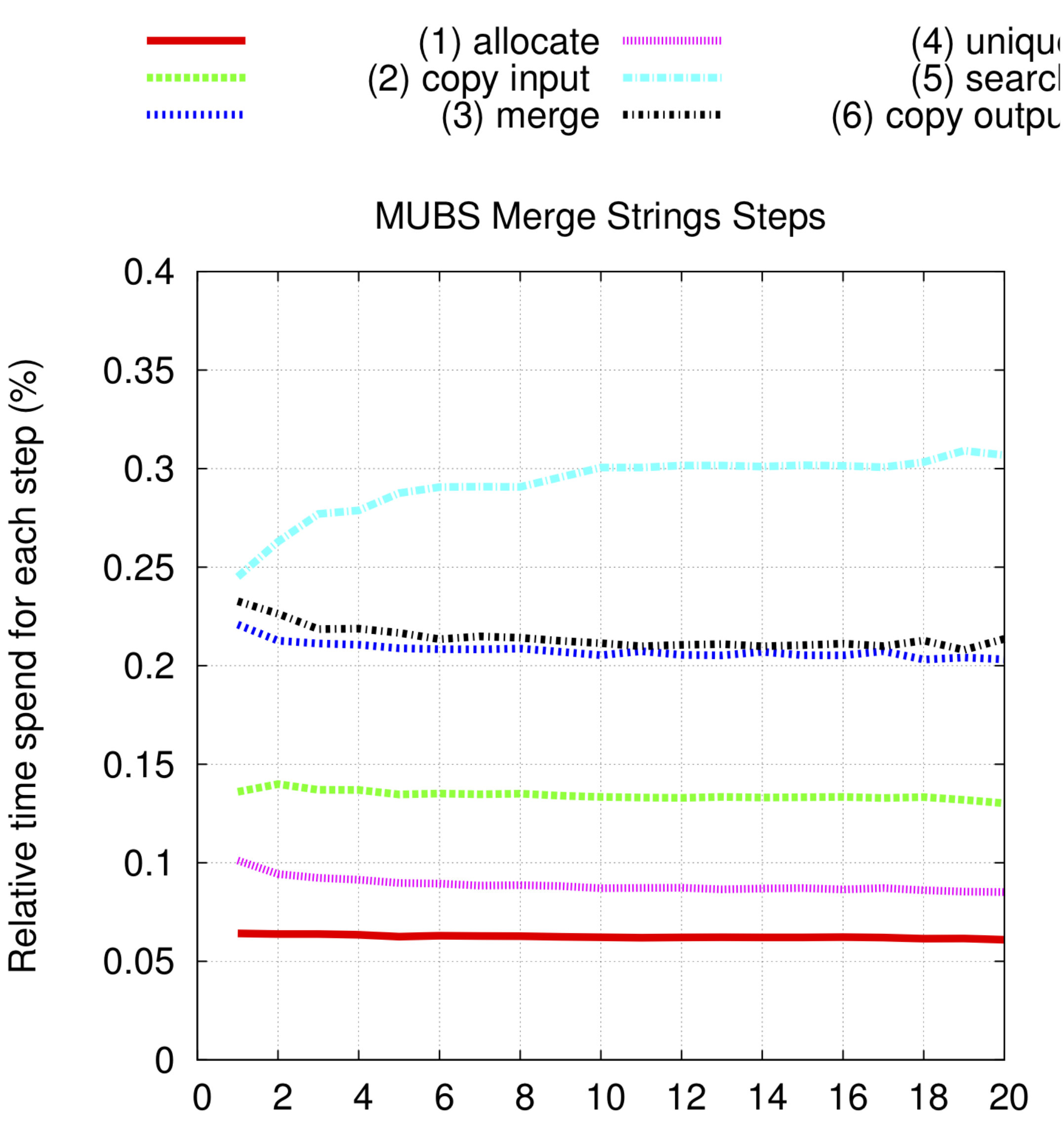
- Numbers:
  - Dictionary merge approaches are up to 40% faster
  - Duplicate removal is up to 27 times faster
  - Page-locked memory increases throughput by up to 10%
- Strings:
  - Throughput of all implementations is reduced
  - BWS and MUBS outperform the CPU implementation (sorting strings on a GPU is expensive)

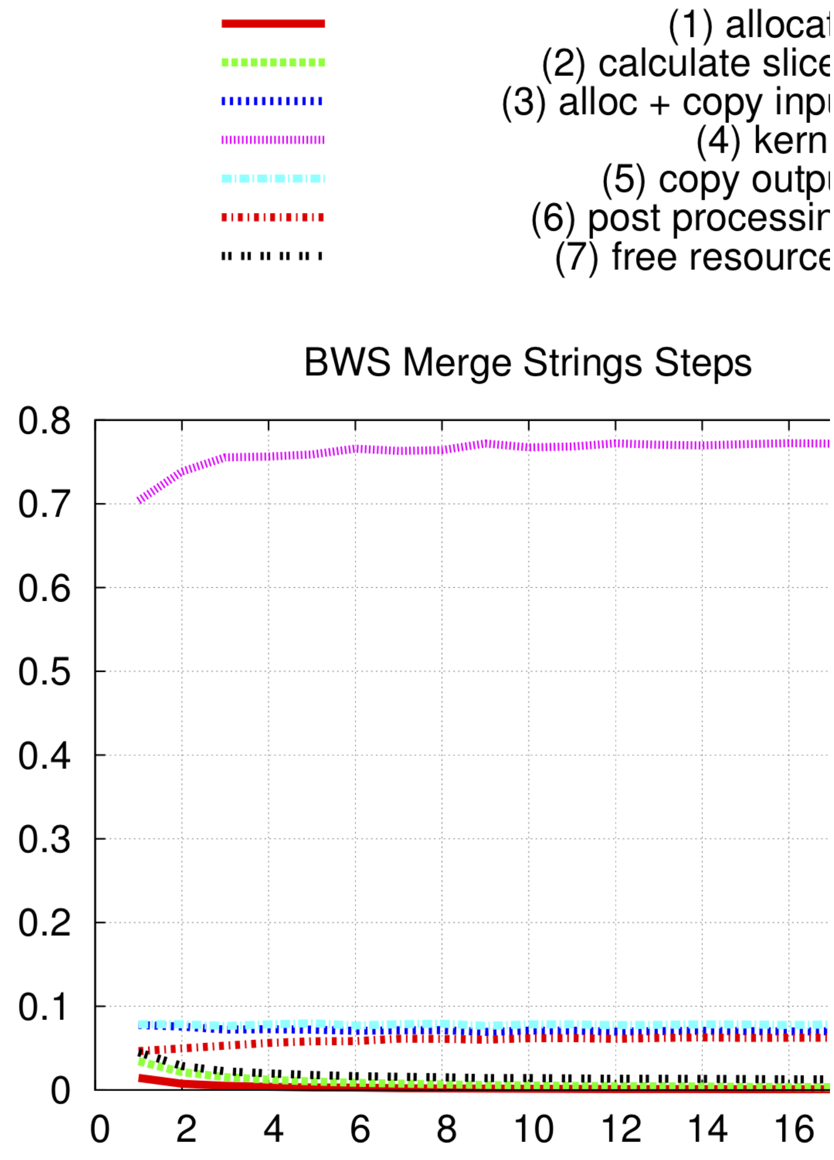




Evaluation - Breakdown

- Relative run-time of individual dictionary merge steps





Conclusion

- Architecture conscious optimizations are needed
- Merge run-time can be reduced with a GPU implementati
  - 27 times improvement on duplicate removal
  - 40% speed up on dictionary merge
- Data transfer is still a bottleneck
- String processing is expensive
- Limited global memory of the GPU compared to main memory

Thank You!

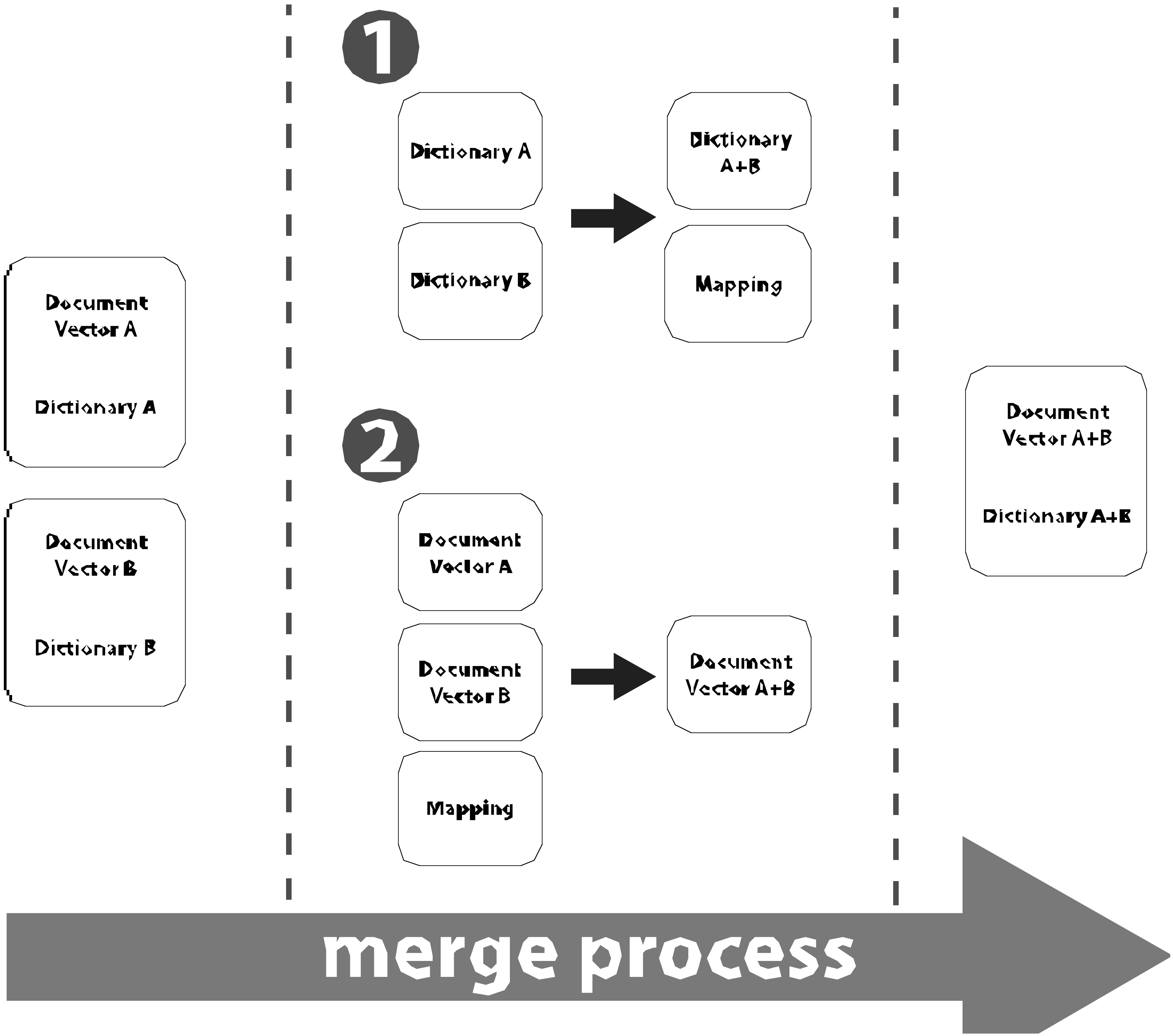
sources:

- J. Krueger, M. Grund, A. Zeier, and H. Plattner. Enterprise Application-Specific Data Management. In EDOC, pages 131–
  Computer Society, 2010.
- H. Plattner and A. Zeier. In-Memory Data Management: An Inflection Point for Enterprise Applications. Springer, 2011.
- Jens Krueger, Changkyu Kim, Martin Grund, Nadathur Satish, David Schwalb, Jatin Chhugani, Alexander Zeier, Pradeep
  and Hasso Plattner. Fast Updates on Read-Optimized Databases Using Multi-Core CPUs. VLDB, to appear 2012.
- D. Merrill and A. Grimshaw. High Performance and Scalable Radix Sorting: A Case Study of Implementing Dynamic Para
  GPU Computing. PPL, 2011.

Backup

Future Work

- Performance analysis for different data
  - Which speedup can be achieved for which data characteristics?
- Support for large numbers of distinct values
- More elaborate scheduling scheme
  - e.g. dynamic scheduling
- Dedicated merge server(s)
  - receives *merge tasks,* responds *merged tables*
  - may be shared across multiple databases
  - e.g. server with few CPU cores but many GPUs


1
Dictionary A
Dictionary B
Dictionary A+B
Mapping
2
Document Vector A
Document Vector B
Mapping
Document Vector A+B
Document Vector A
Dictionary A
Document Vector B
Dictionary B
Document Vector A+B
Dictionary A+B
merge process