HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI
MATEMAATTIS-LUONNONTIETEELLINEN TIEDEKUNTA
MATEMATISK-NATURVETENSKAPLIGA FAKULTETEN
FACULTY OF SCIENCE

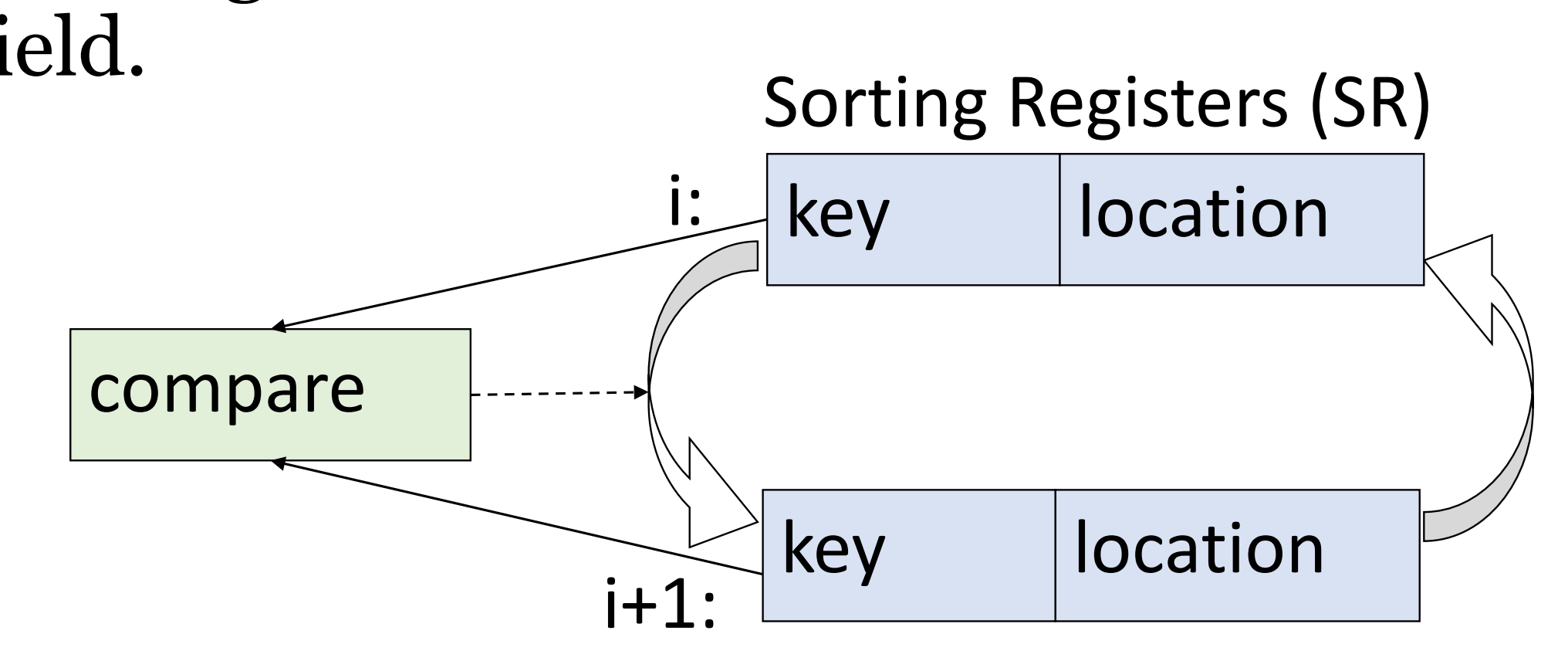
LINEAR TIME SORTING WITH SPECIALIZED PROCESSOR

Teemu Kerola, Department of Computer Science, University of Helsinki

SORTING REGISTERS

Specialized processor contains N (e.g., 1 billion or 1 trillion) specialized sorting registers to sort any data set (smaller than the number of sorting registers) in linear time. The keys may be (e.g.) 64 bit integers, floating points, or short strings.

Each sorting register (SR) has two fields, key and location. Sorting is done based on the key value, and original data item is found with the location field.

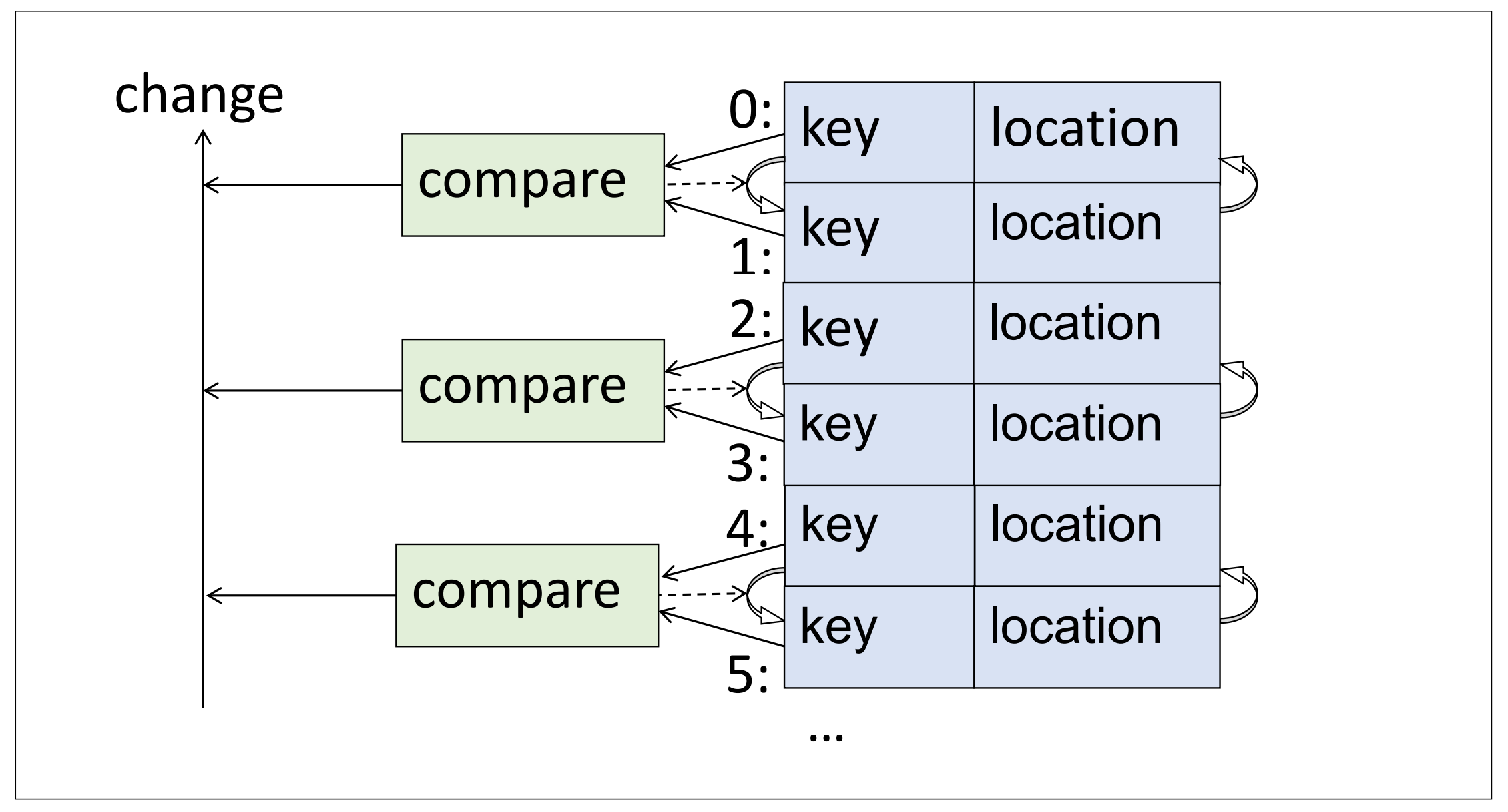


Parallel Bubble Sort sorting element

PARALLEL BUBBLE SORTING

Sorting is done alternating even and odd steps. In even step, each key in even numbered SR is compared to the key in the following SR, and the register contents are interchanged if needed. Pairwise comparisons and possible interchanges are independent of each other, and they can be performed massively in parallel. In odd step the same is done for all odd numbered SRs. The same compare circuits can be used both in even and odd steps.

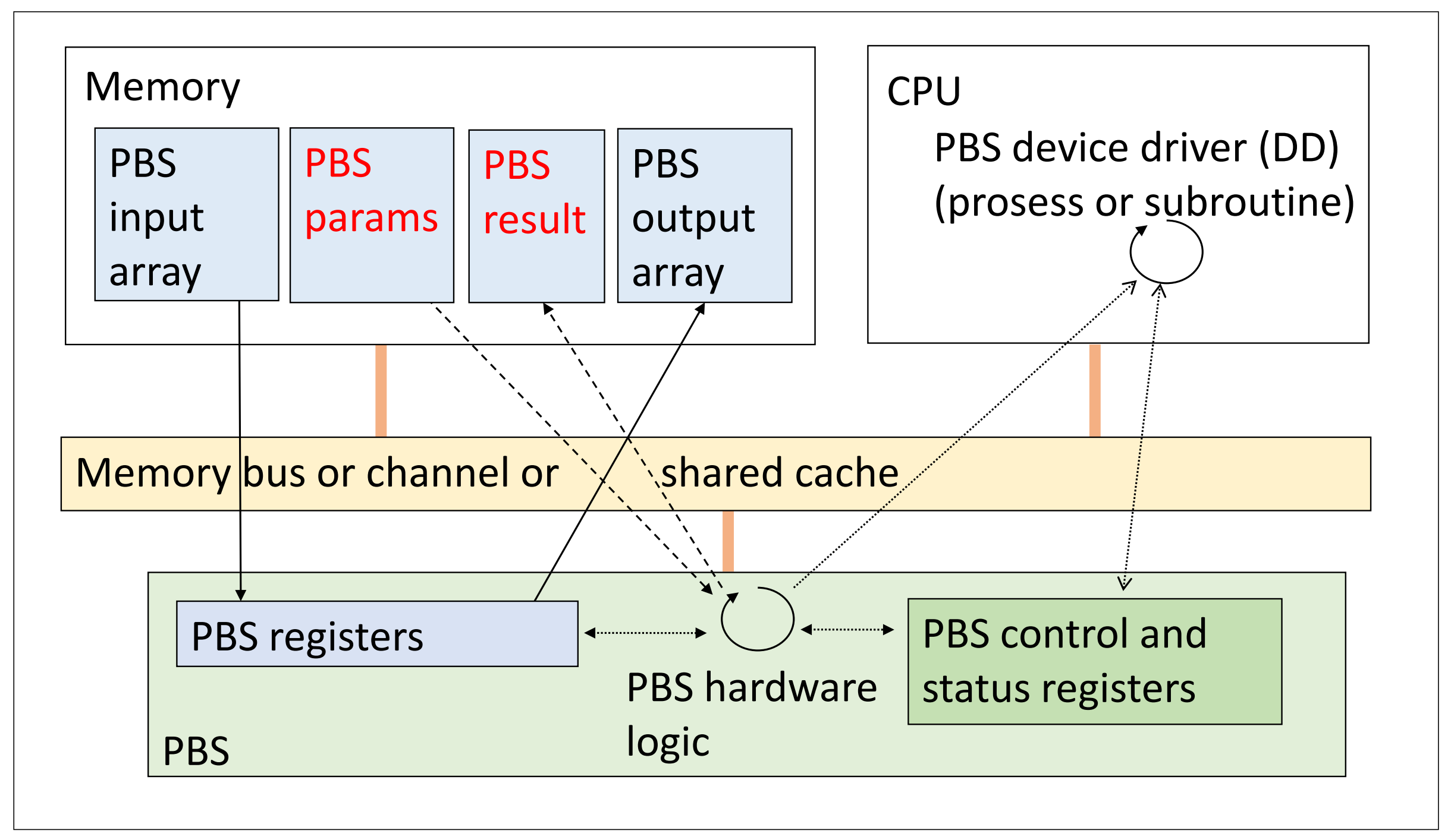
Sorting is completed in at most n steps (Habermann 1972). Sorting may also be completed earlier, which is observed when two subsequent sorting steps happen without interchanges. If the key set is already sorted, sorting terminates immediately after the 1st two steps. The termination test is easy to implement. If any one of the sorting elements needs a value interchange, it asserts the shared change signal. The change-signal value is stored and two subsequent not-asserted values indicate sorting to be completed.



Parallel Bubble Sorter, even step

PBS DEVICE DRIVER

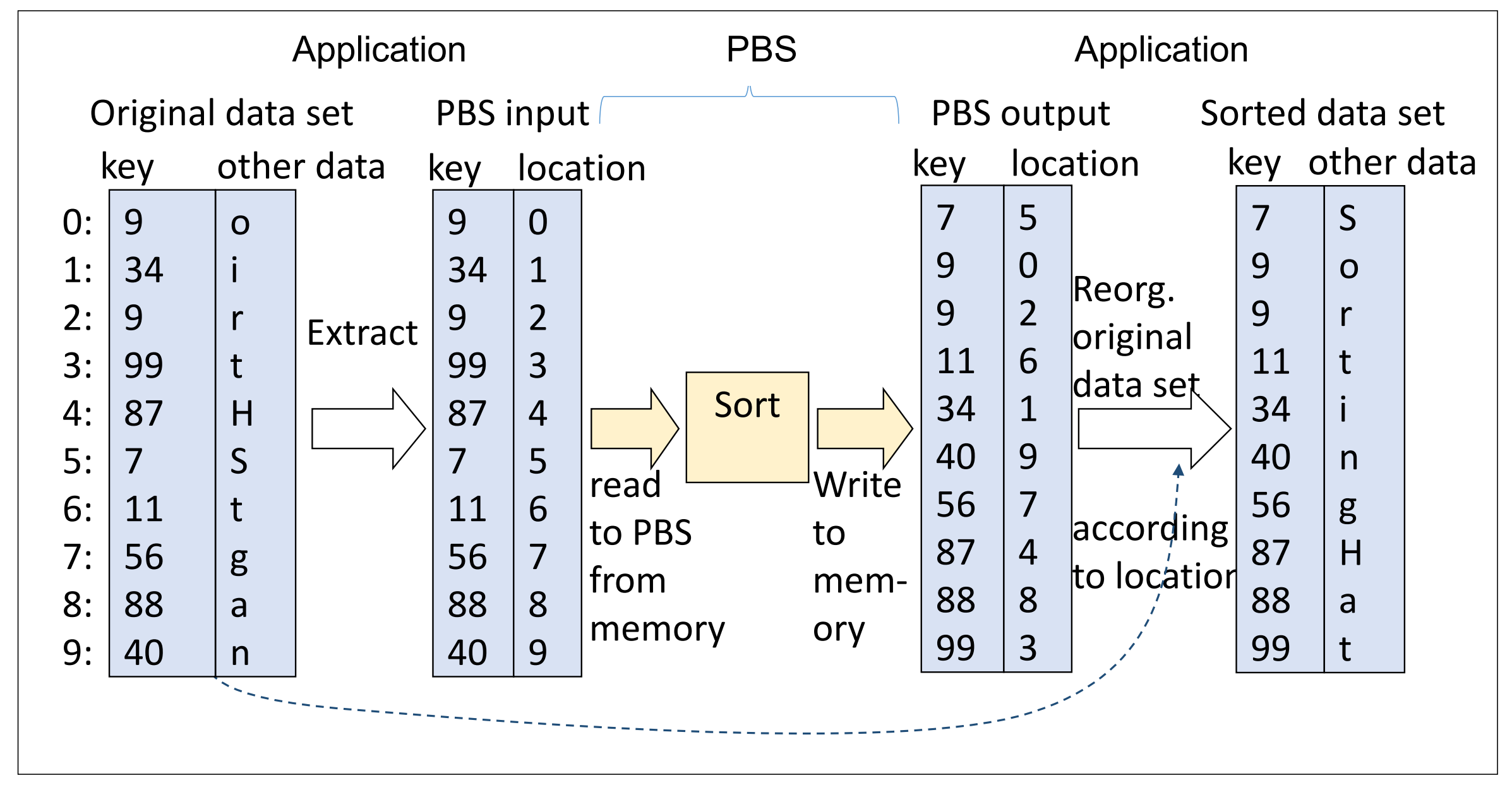
PBS can be implemented in the system (e.g.) as a coprocessor, same way as GPUs. PBS could use DMA or shared cache for fast main memory access, and it would have its own operating system device driver (DD). The DD gives PBS the key count, key and location field types, sorting order, and the memory location of input array, which contains all {key, location} pairs, as well as the address of the resulting output array. The PBS reads the input array from memory to sorting registers, sorts the key set and writes the sorted registers back to memory to the designated output array. It also returns the number of steps used in the sort.



PBS implementation in HW and in OS

METHOD TO SORT LARGE DATA SETS

Sorting is done (e.g.) in multiple phases. The application first defines and extracts the input array with n {key, location} pairs from the original data set. The key can be any field in the data set. Original data set may be (e.g.) a (multi-dimensional) array or any (distributed) data structure, in memory or in disk. The original data set can contain much more data than just the key fields. The application calls the PBS device driver, which invokes the PBS to do the sorting. The PBS reads input array from memory to sorting registers, sorts it, and saves the resulting output array back to memory.



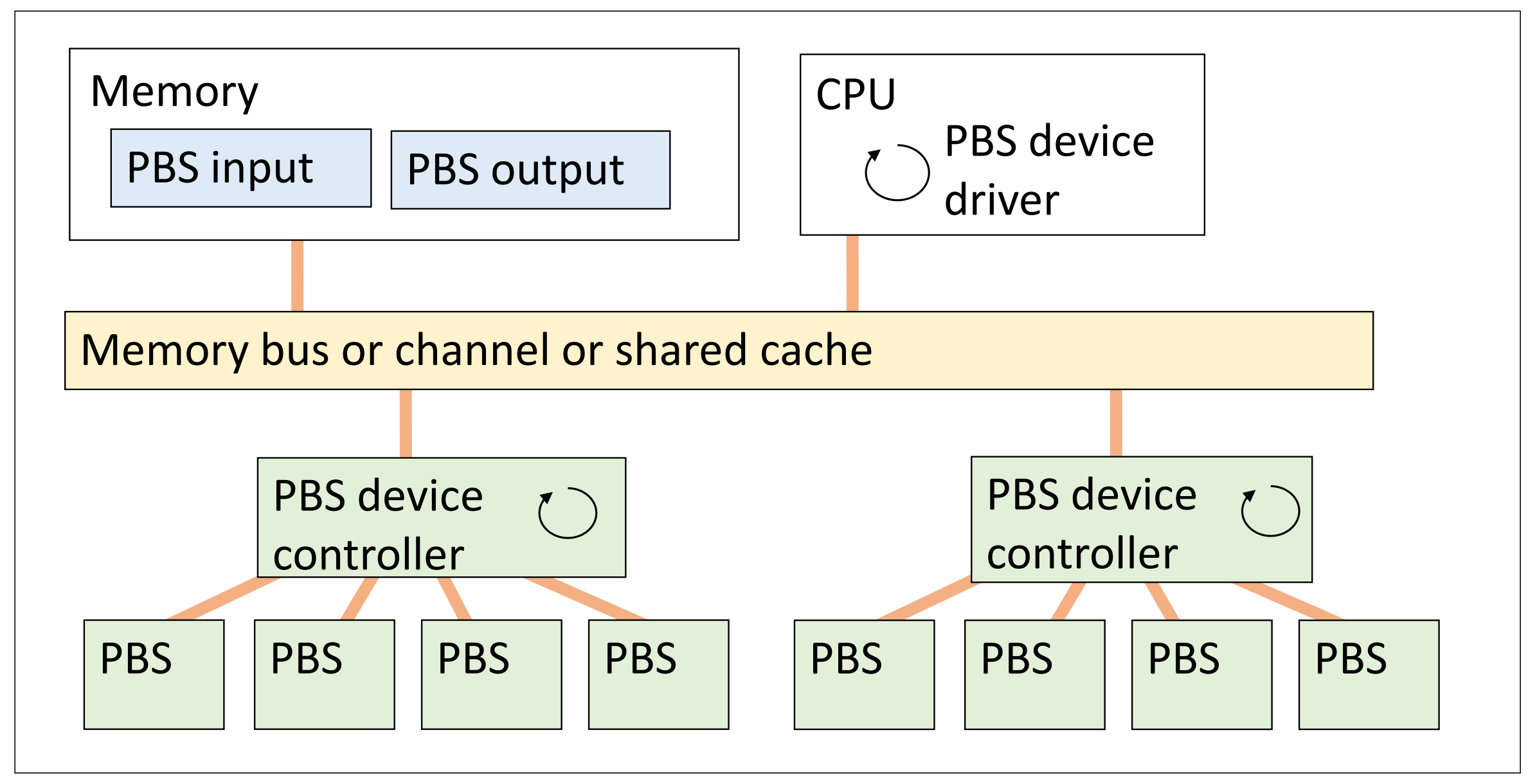
	Original data set key	Original data set other data	PBS input key	PBS input location	PBS output key	PBS output location	Sorted data set key	Sorted data set other data
0:	9	o	9	0	7	5	7	S
1:	34	i	34	1	9	0	9	o
2:	9	r	9	2	9	2	9	r
3:	99	t	99	3	11	6	11	t
4:	87	H	87	4	34	1	34	i
5:	7	S	7	5	40	9	40	n
6:	11	t	11	6	56	7	56	g
7:	56	g	56	7	87	4	87	H
8:	88	a	88	8	88	8	88	a
9:	40	n	40	9	99	3	99	t

Sorting with PBS, when n≤N

If needed, the device driver will split very large input array (n>N) in size N blocks, sorts them with PBS, and merges the results into one output array. Finally, the application reorganizes the original data set according to the location fields in the output array. Alternatively, the output array can be retained as an index to the original data set. If the data set was originally already in wanted order, the device driver will inform the application about it and the application will not need to do any actions on the original data set.

SYSTEMS WITH MANY PBSs

If the key sets are very large, sorting can be speeded up with many PBSs. They could be controlled by the same device driver which first spreads the work by giving each PBS their own block to sort, and then merges the output arrays from multiple PBSs before returning the control to calling application. Another alternative would be to implement them as separate devices under higher level PBS device controller, just like one device controller may control multiple hard disks. The device controller could perform the merge operations for the PBSs that it controls. Either way, each individual PBS should be implemented as large as technology allows, because sorting in PBS has time complexity O(n), whereas merge has time complexity O(n log(n/N)). You could expand this even further to have multiple PBS device controllers, each with their own set of PBSs, and then the final sort could be left to the device driver.



PBS device controllers, each with multiple PBSs

REFERENCES

Nico Habermann, 1972, *Parallel neighbor-sort (or the glory of the induction principle)*, Carnegie-Mellon University, Technical report, August 1972, 12 p.