Software Engineering Seminar

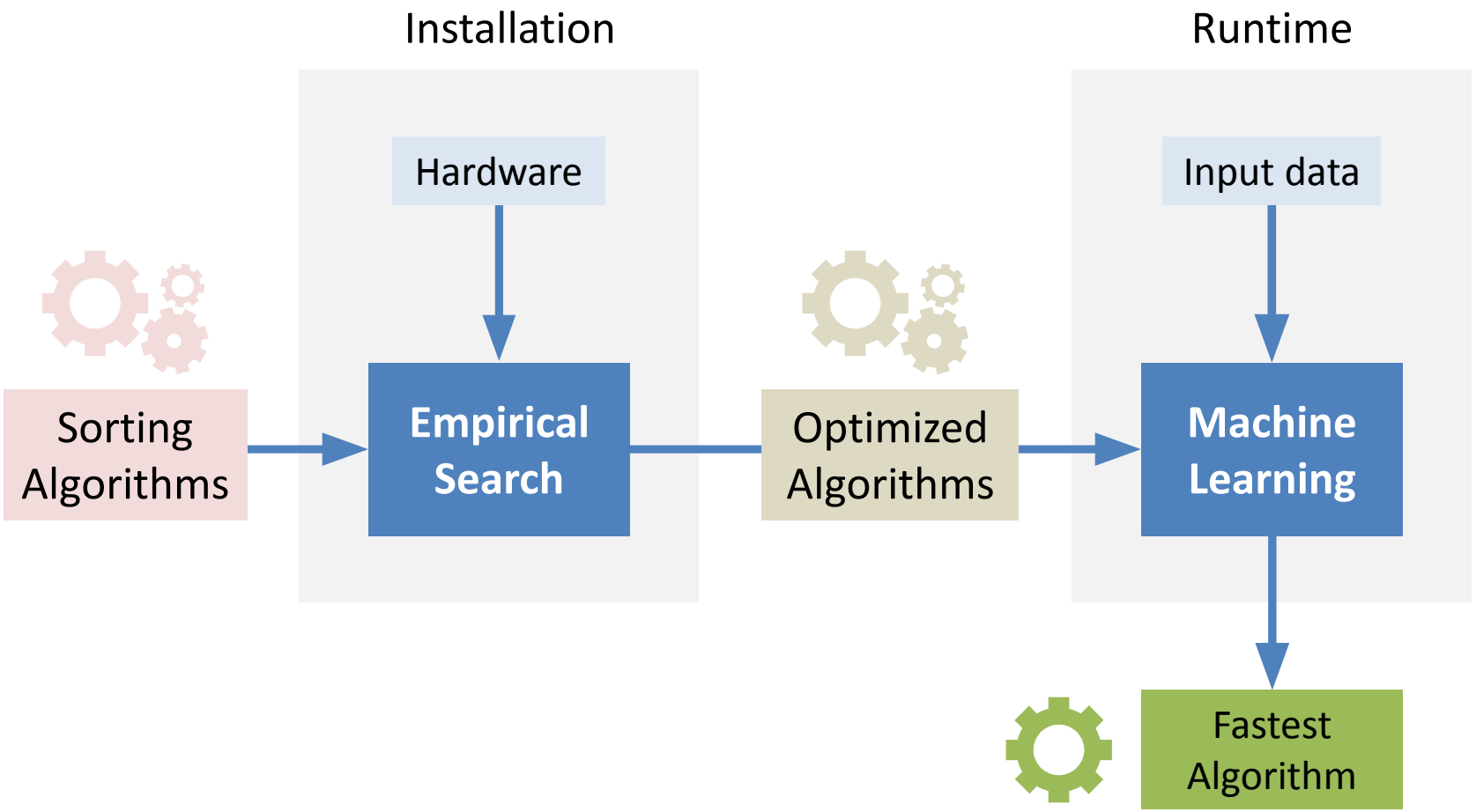
Stephan Semmler

A Dynamically Tuned Sorting Library

Xiaoming Li, María Jesús Garzarán, and David Padua

2004

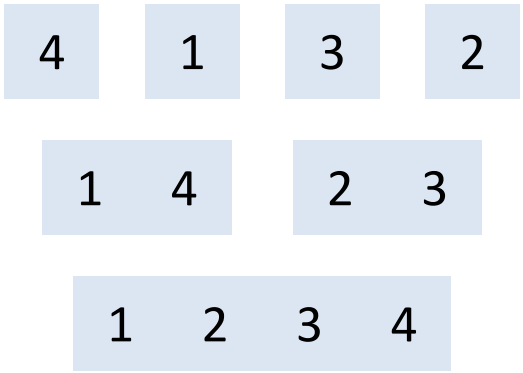
The Sorting Library



Overview

- Sorting Algorithms
- Optimizing the Algorithms
- Factors of Performance
- The Library
- Results
- Final Words

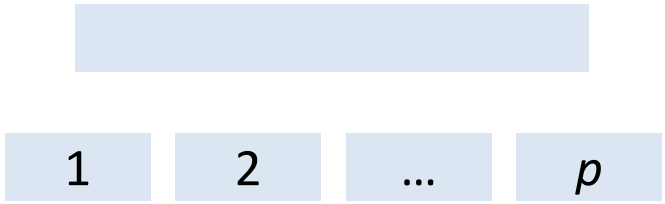
Merge Sort



- Divide and Conquer
- Runtime O(n·log(n))
- Needs additional memory to merge

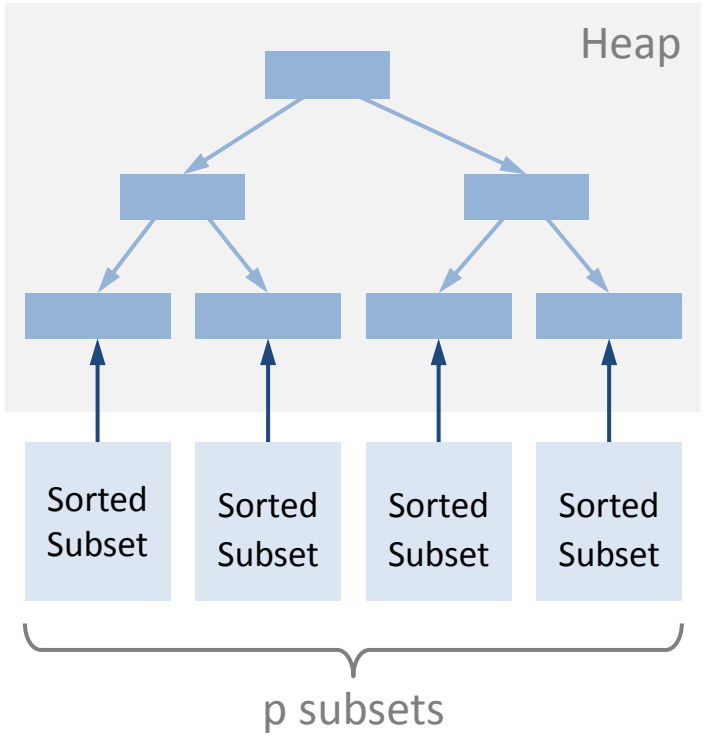
Multiway Merge Sort

- Partition data into *p* subsets

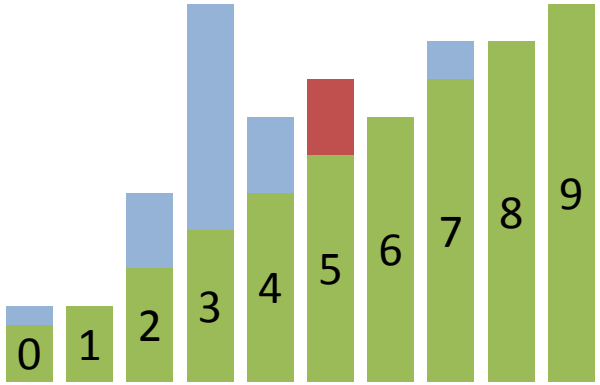


- Sort each subset

- Merge *p* subsets using a heap



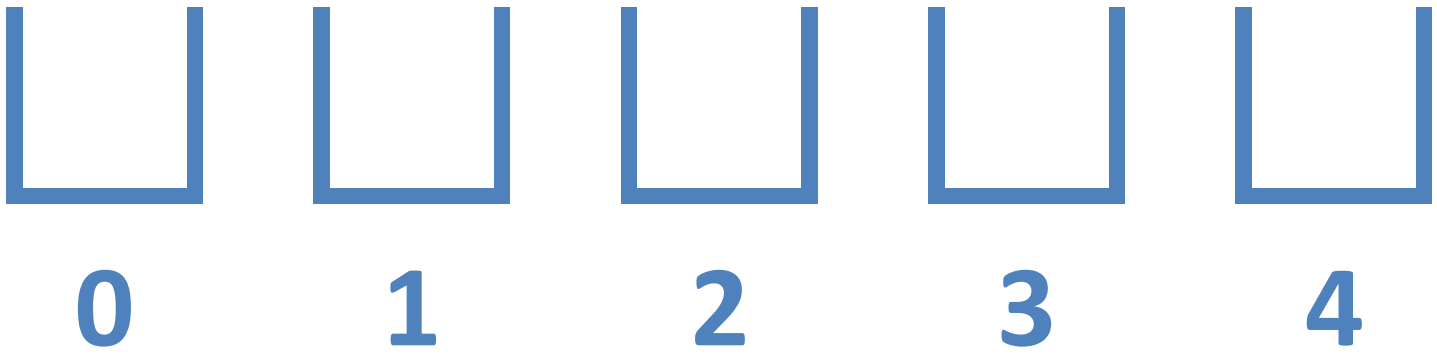
Quicksort



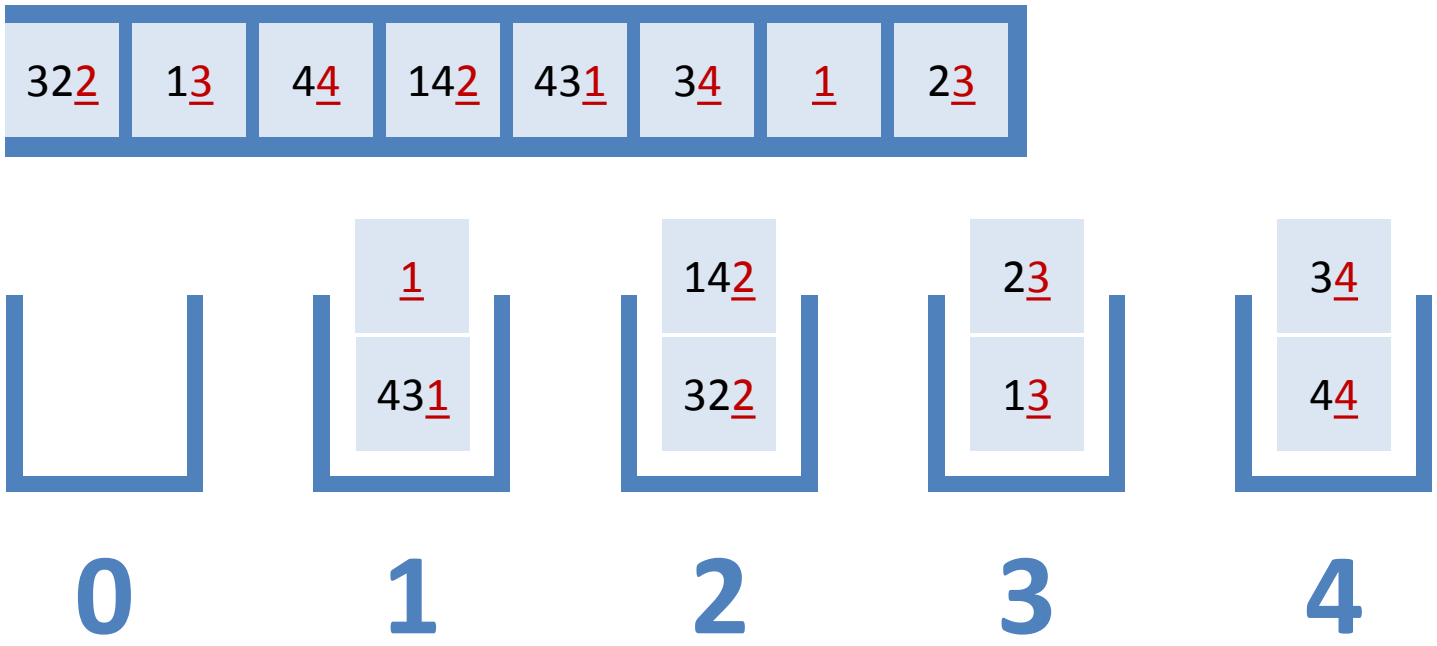
- Average runtime O(n·log(n))
- Inplace
- Worst Case runtime $O(n^2)$

Radix Sort

322	13	44	142	431	34	1	23



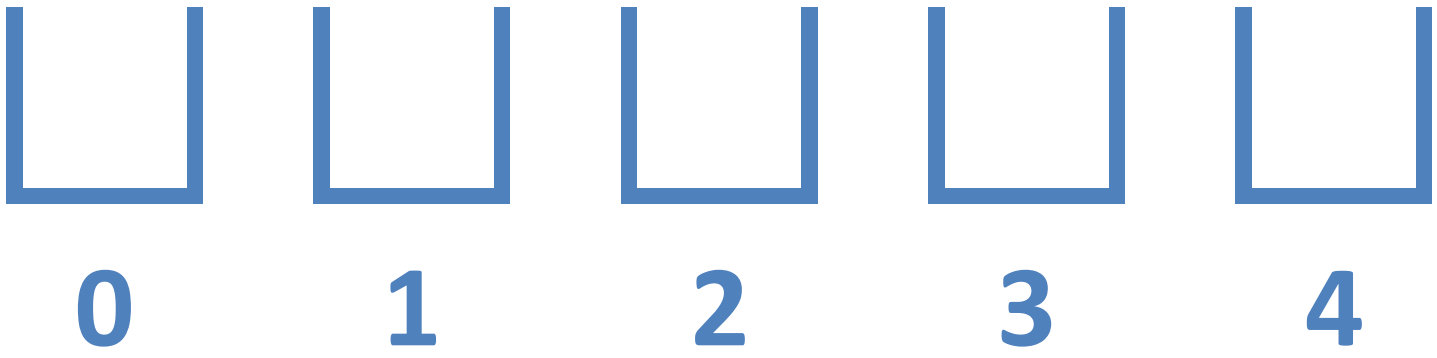
Radix Sort



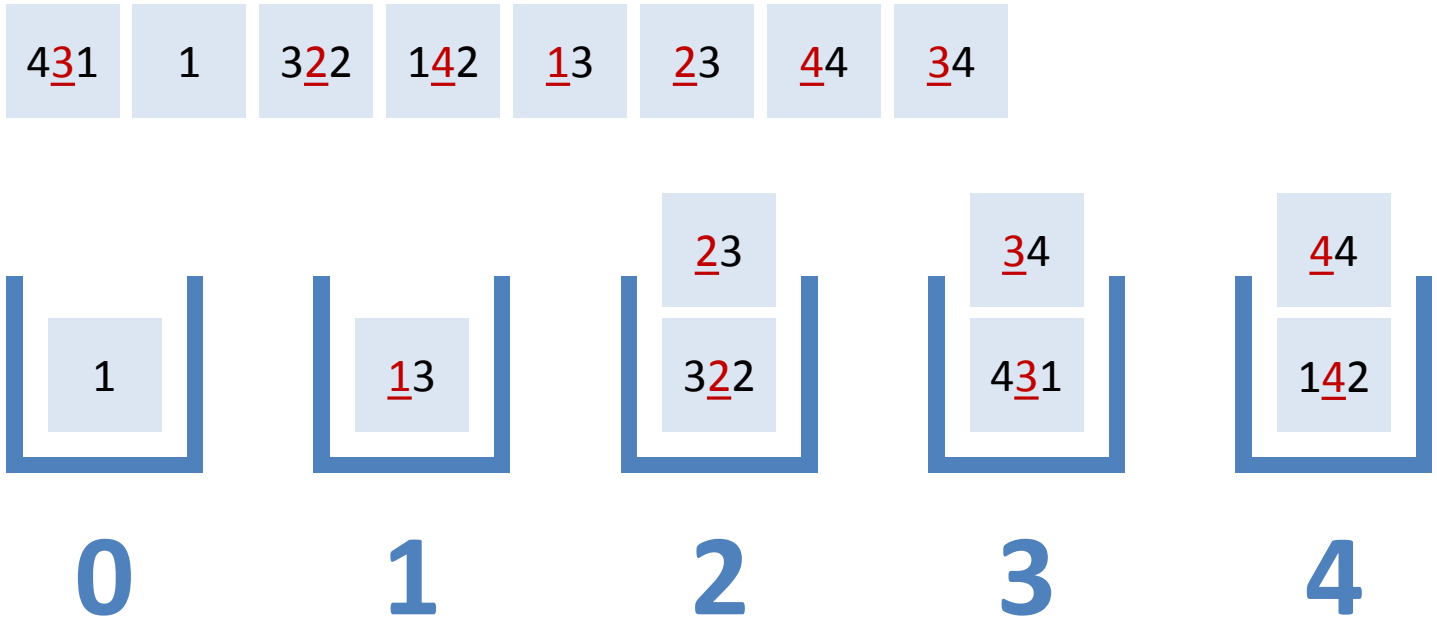
431 1 322 142 13 23 44 34

Radix Sort

431	1	322	142	13	23	44	34

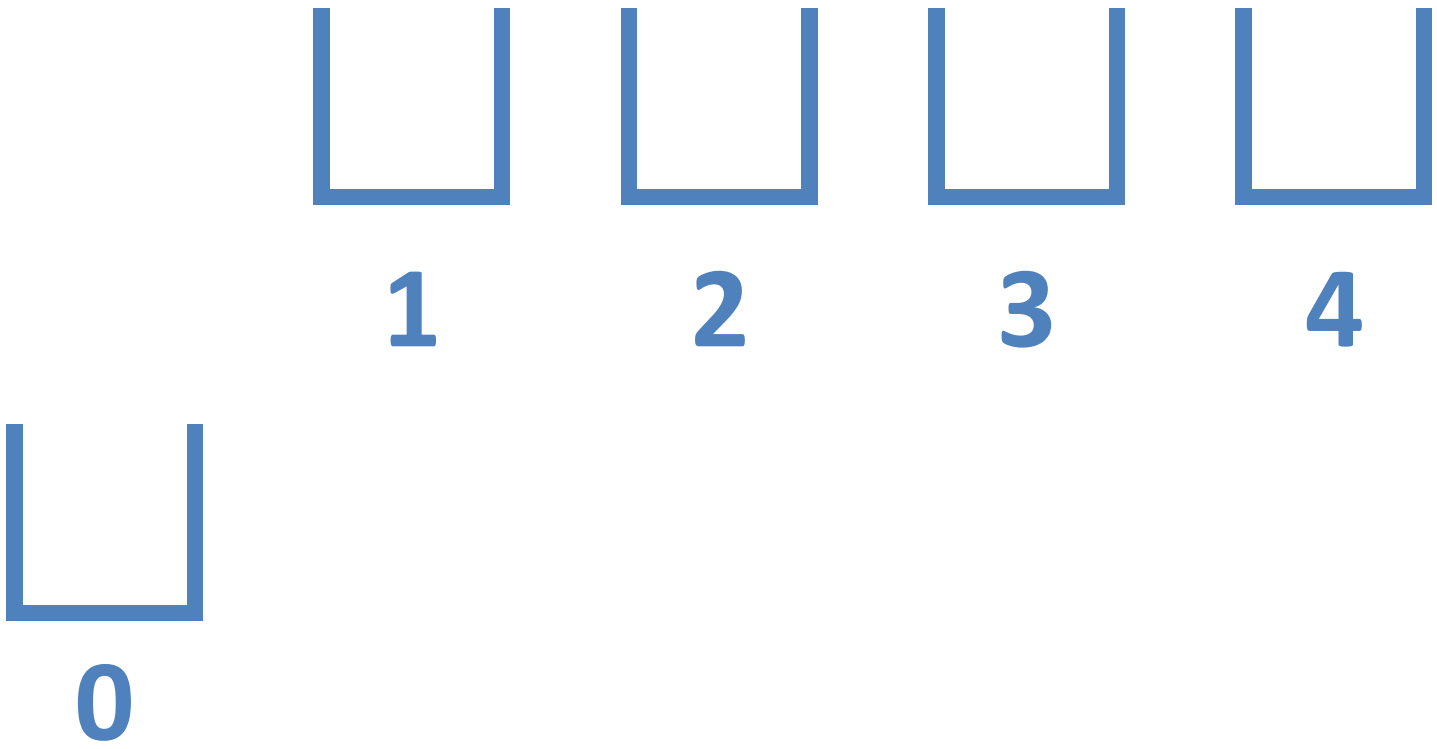


Radix Sort

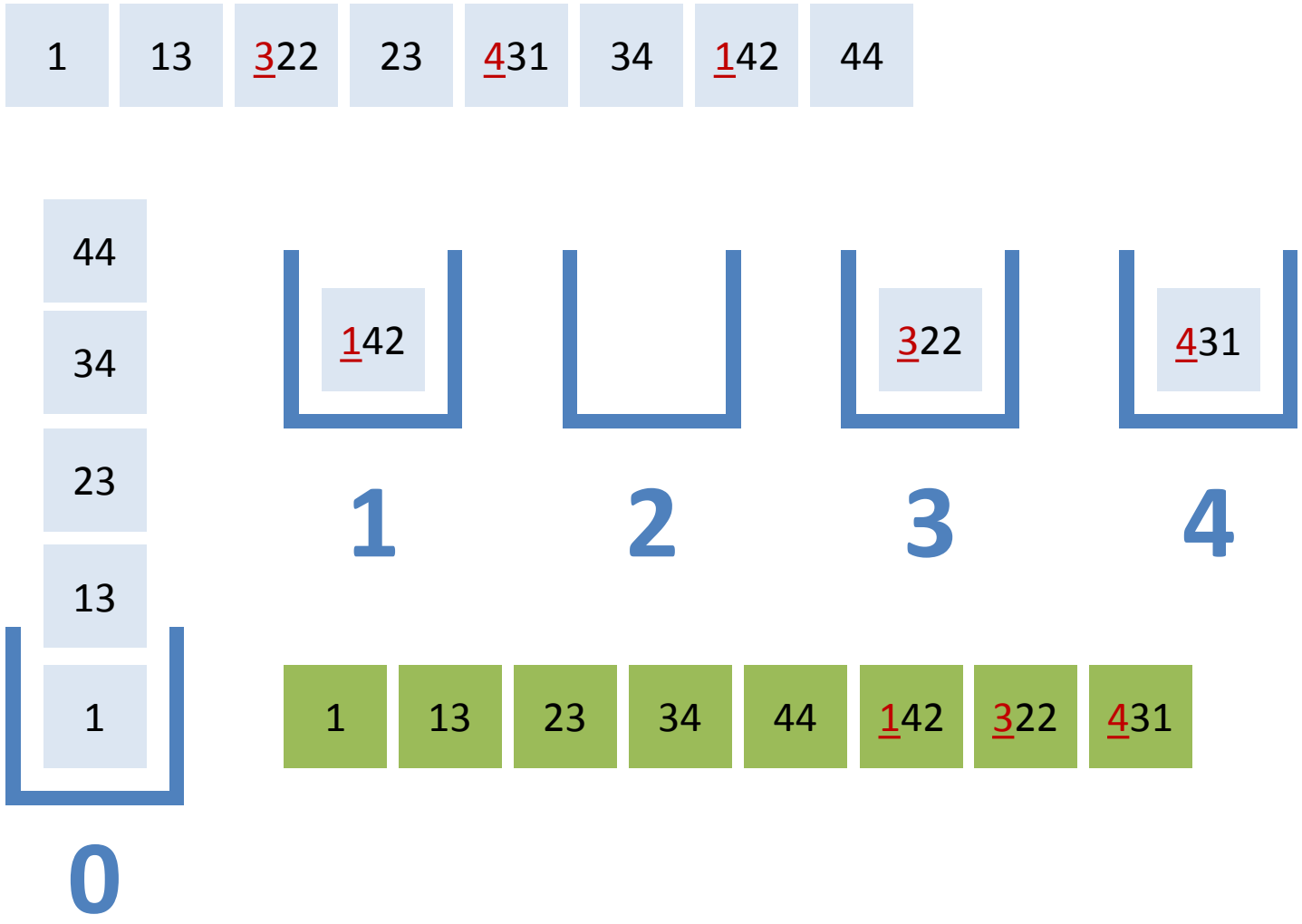

431 1 322 142 13 23 44 34
1
13
23
322
34
431
44
142
0
1
2
3
4
1 13 322 23 431 34 142 44

Radix Sort

1	13	322	23	431	34	142	44



Radix Sort


1
13
322
23
431
34
142
44
44
34
23
13
1
142
322
431
1
2
3
4
1
13
23
34
44
142
322
431
0

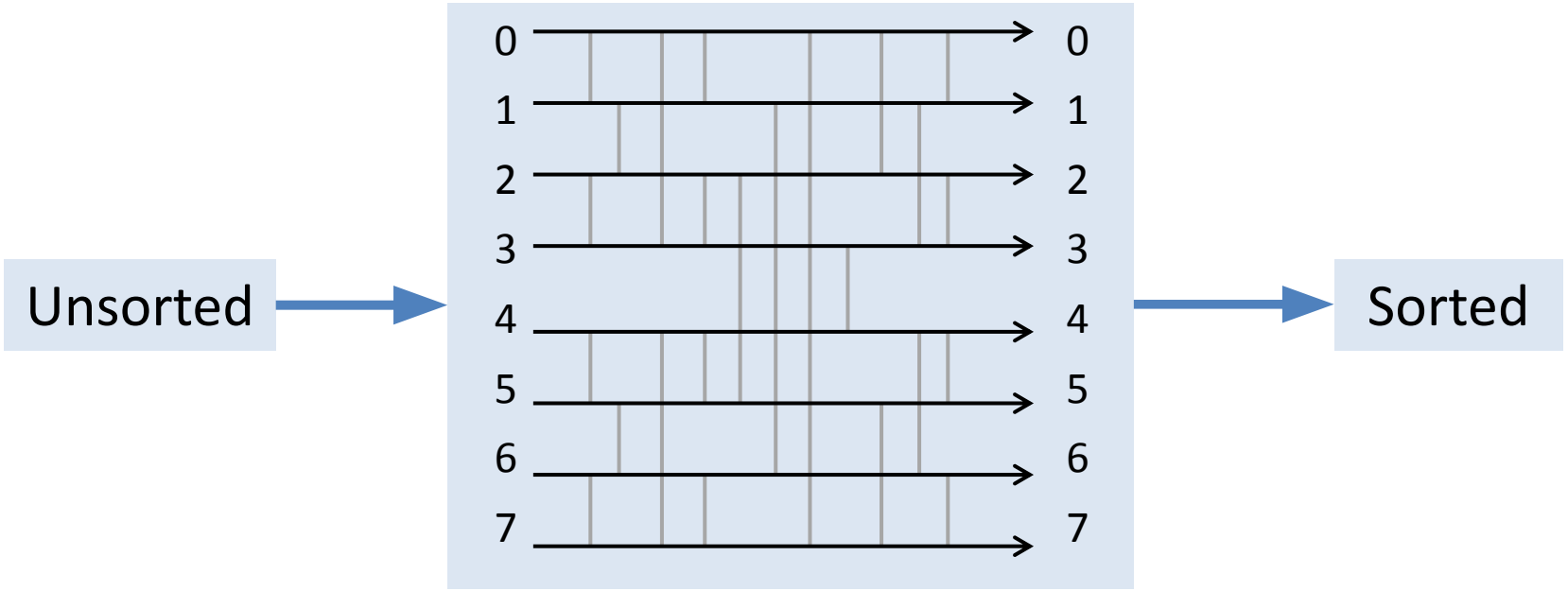
Radix Sort

- Non-comparative sorting algorithm
- Needs Integer Keys
- Linear Time Complexity O(n)
- Highly dependent on key distribution

Insertion Sort

- Average case $O(n^2)$
- Best case O(n) for sorted data
- Good for small partitions

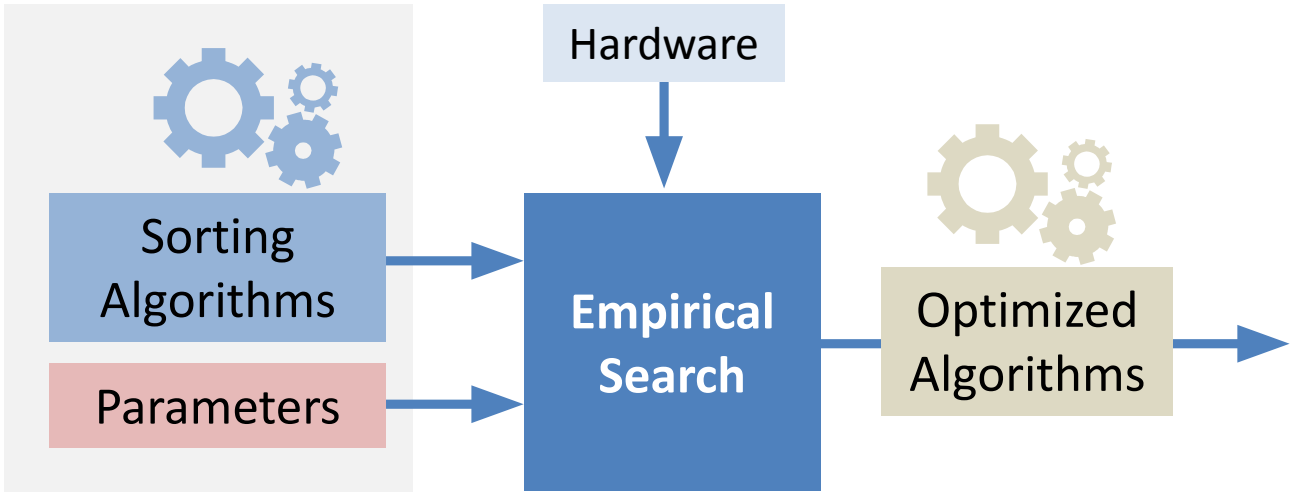
Sorting Networks



- Like hardwired
- Only appropriate for very small amount of data

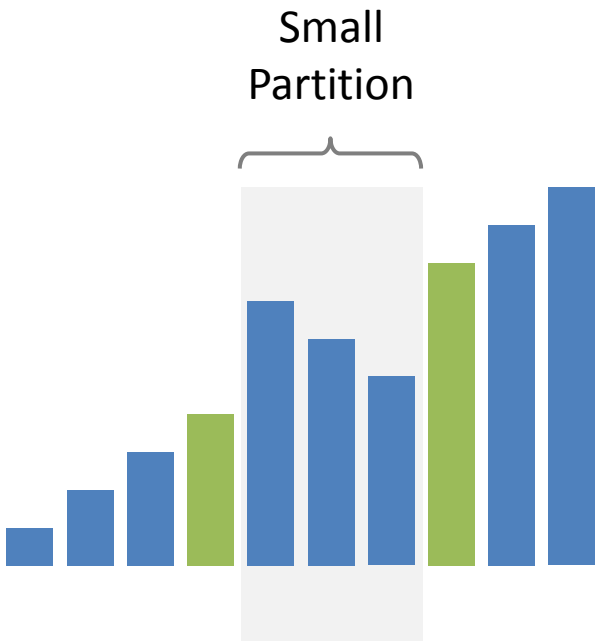
Optimizing Algorithms

- For a given Architecture
  - Cache Size
  - Registers
  - Cache Line Size
- Which Parameters to tune?



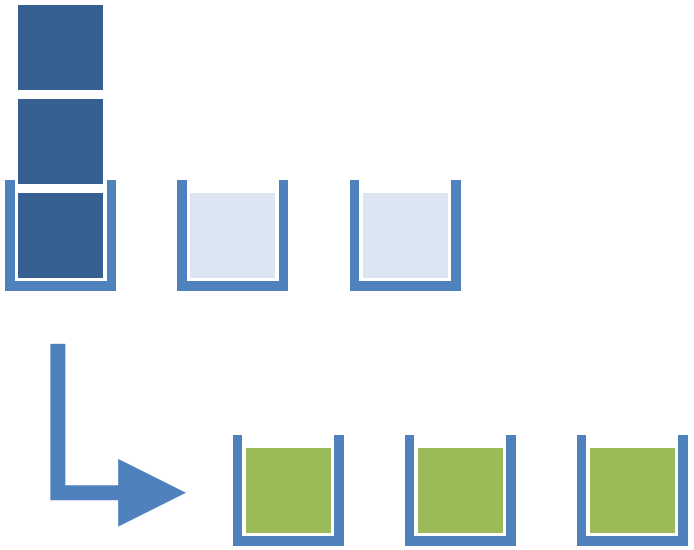
Tuning Quicksort

- Small partitions
  - Insertion Sort
  - Sorting Networks
- Threshold for small partitions
- Apply immediately
  or at the end



Tuning Radix sort (CC-radix sort)

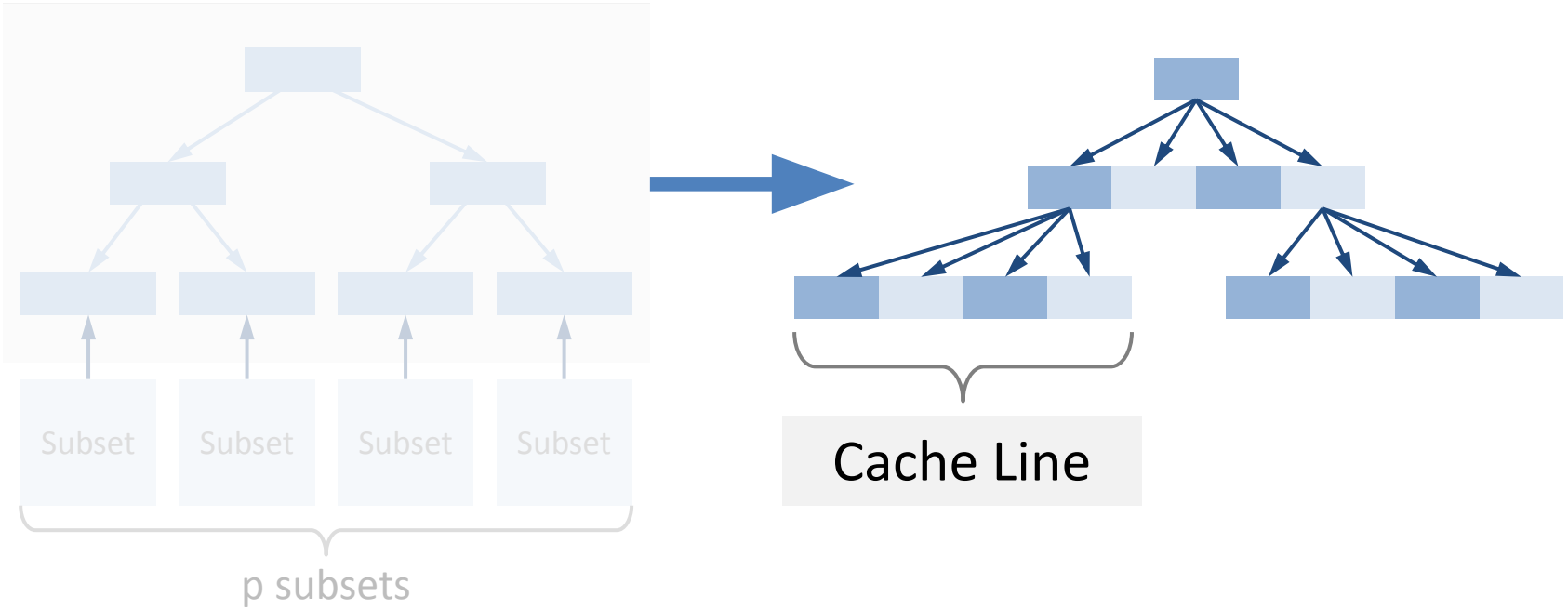
- Create sub-buckets if data is too large for cache

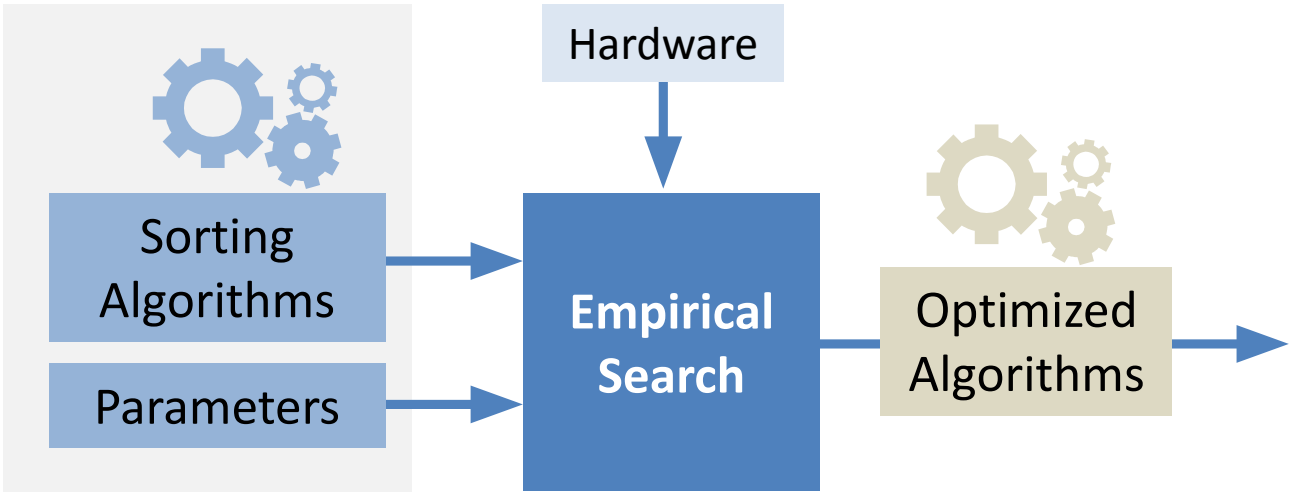


- Apply radix sort for sub-buckets
- Insertion sort / Sorting networks for small partitions

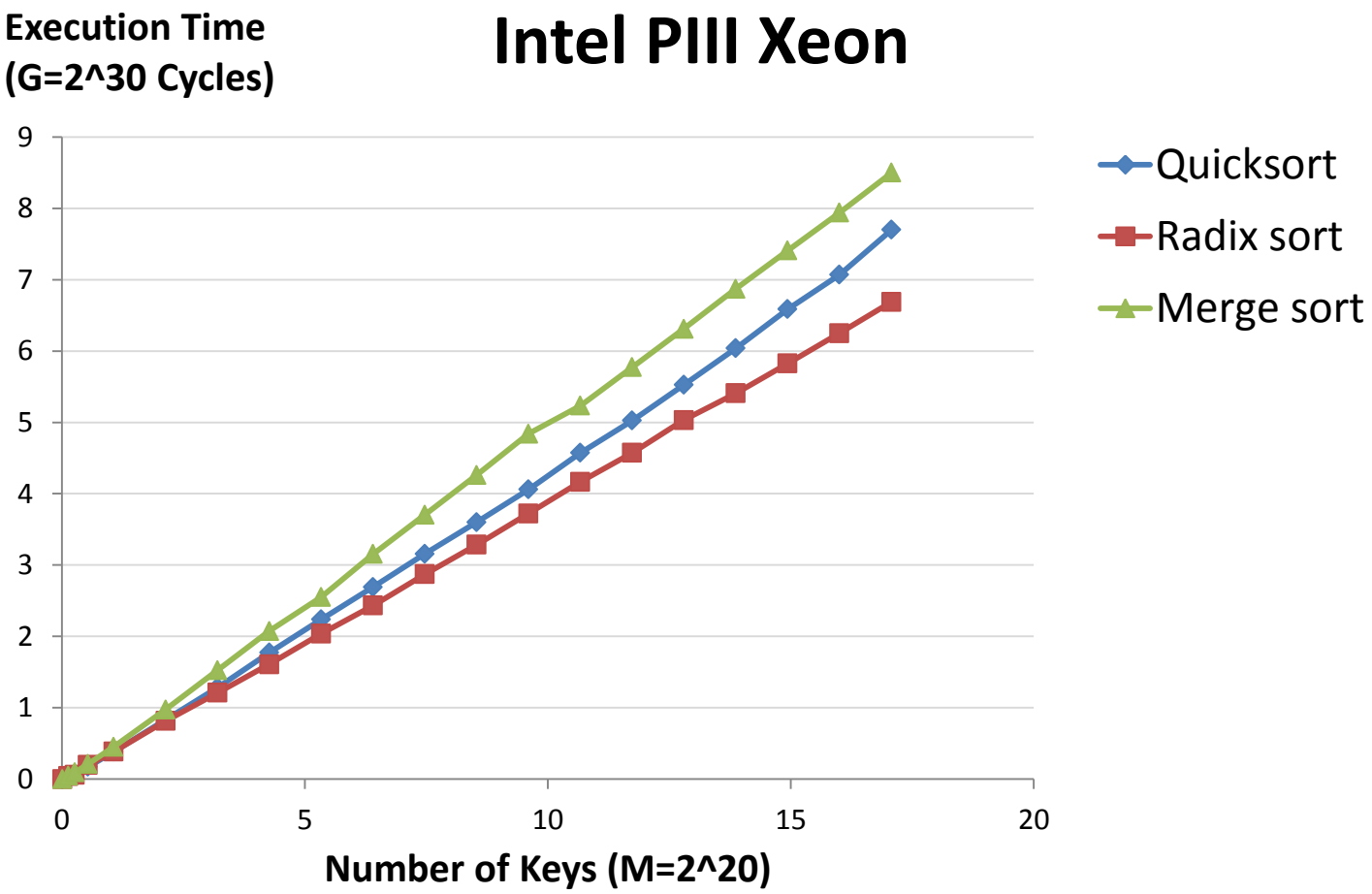
Tuning Multiway Merge sort

- Number of subset *p*
- Operation on heap: find smallest child
  - Adapt fanout such that children fit into Cache Line

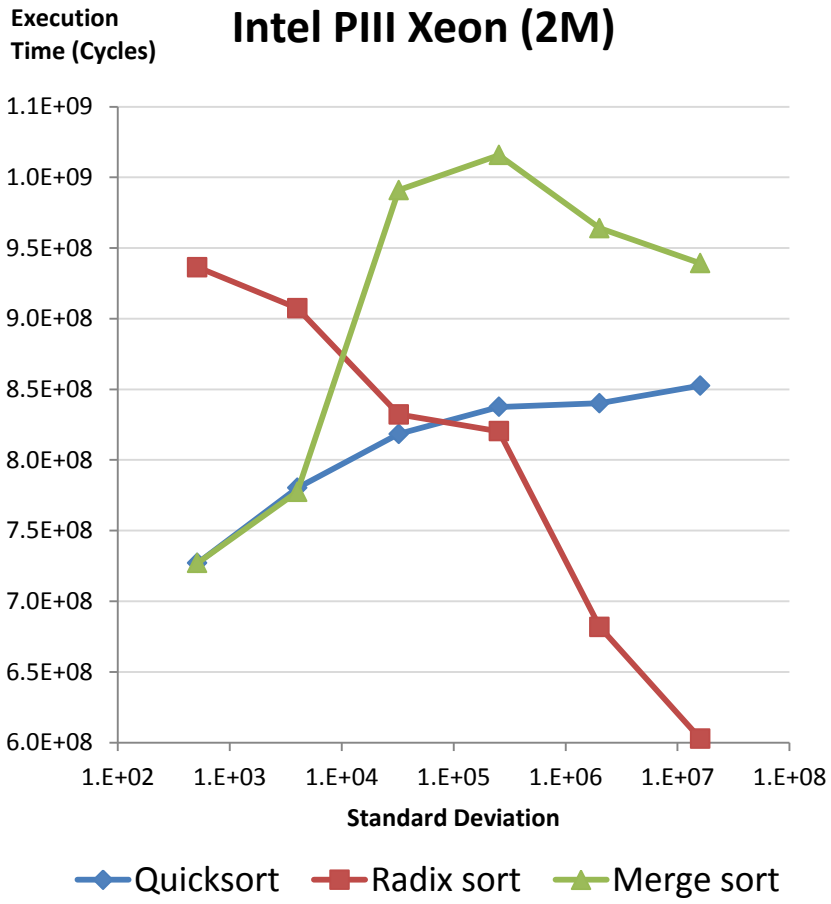


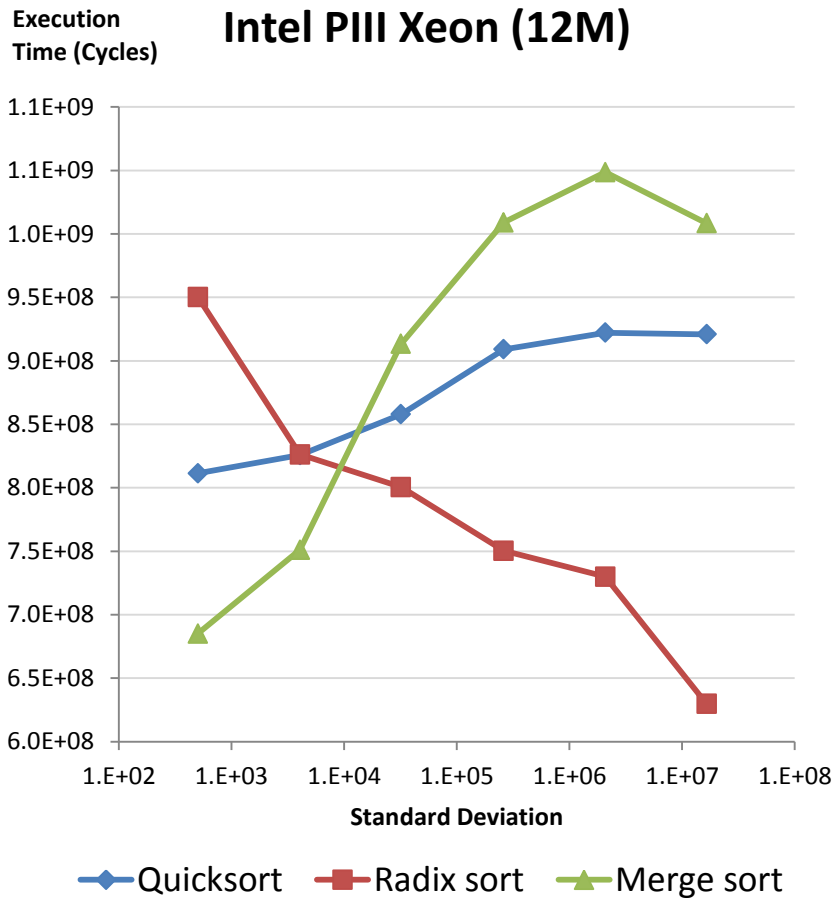

Hardware
Sorting Algorithms
Parameters
Empirical Search
Optimized Algorithms

Comparison of Sorting Algorithms



Varied Standard Deviations





Impact of Input Data

- Number of keys does not affect the relative performance
- Standard deviation matters!
  - Distribution among buckets in radix sort
  - Fewer operations on the heap in multiway merge sort
- Problems with Standard deviation
  - Only *related* to distribution of digit in keys
  - Expensive to compute
  - Use Entropy instead

Entropy

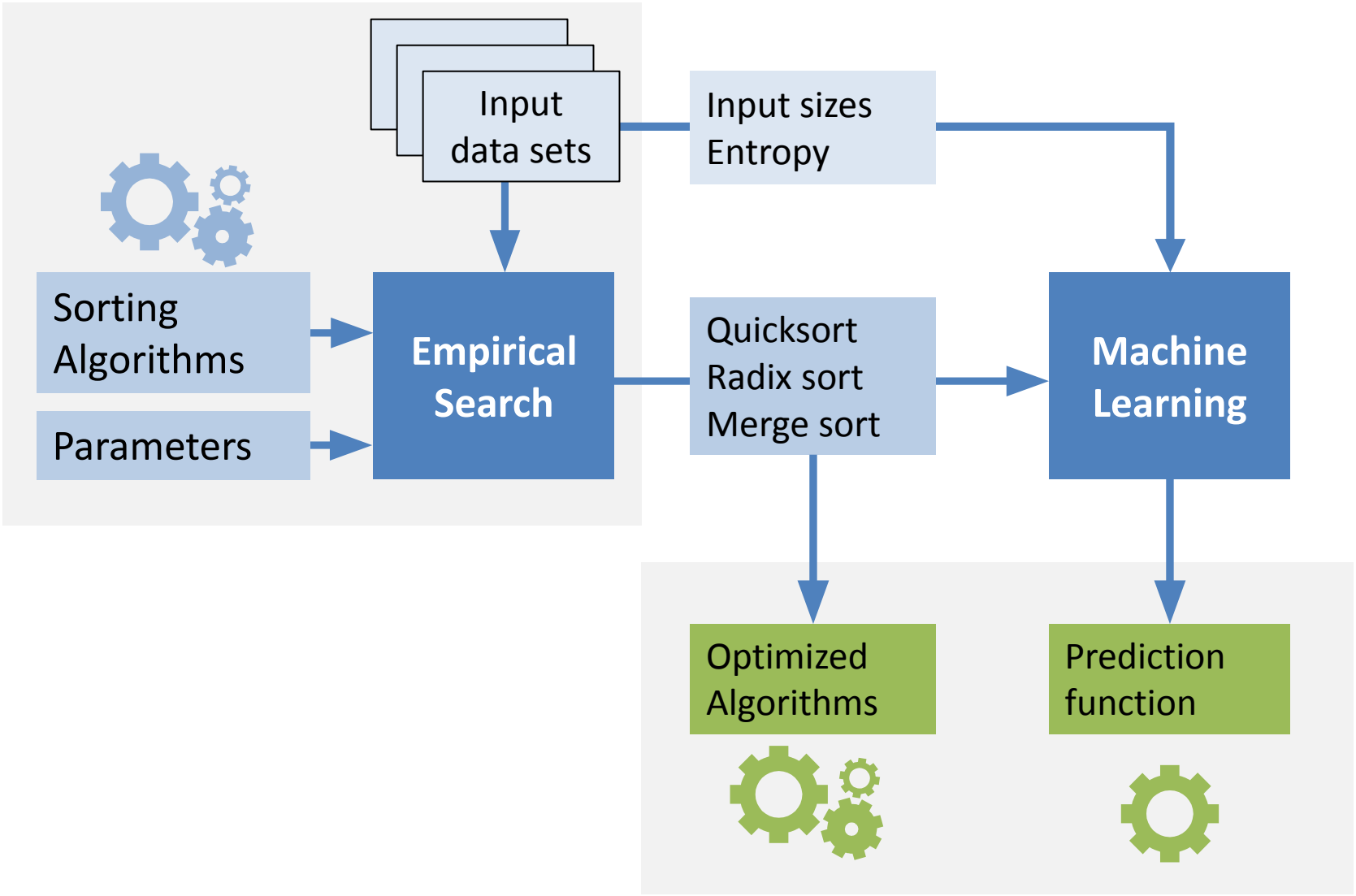
- Expected value of the information

1	2	4
1	3	5
1	3	6
0	0.9	1.58

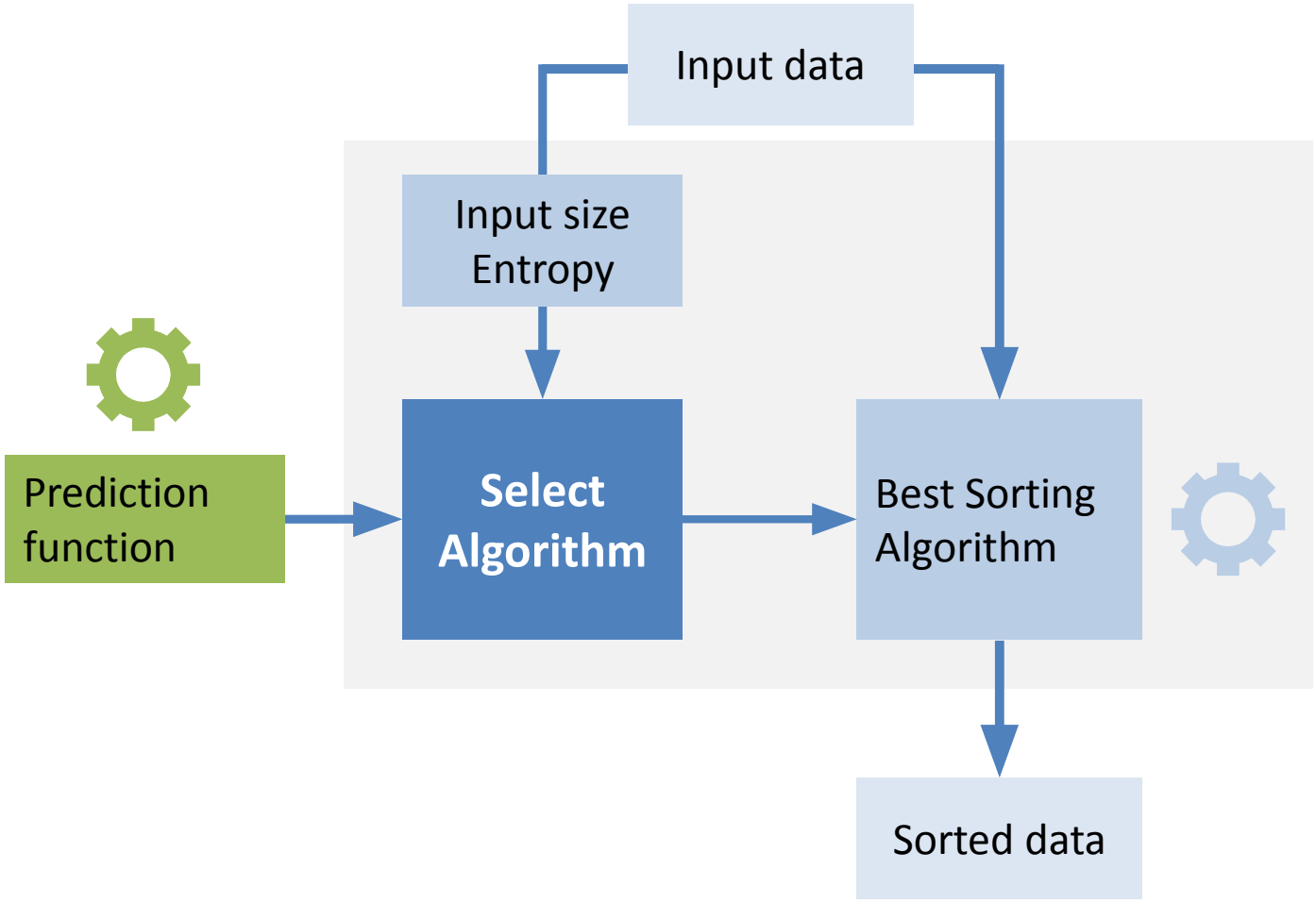
$$\sum_i -P_i * \log_2 P_i$$

Entropy vector

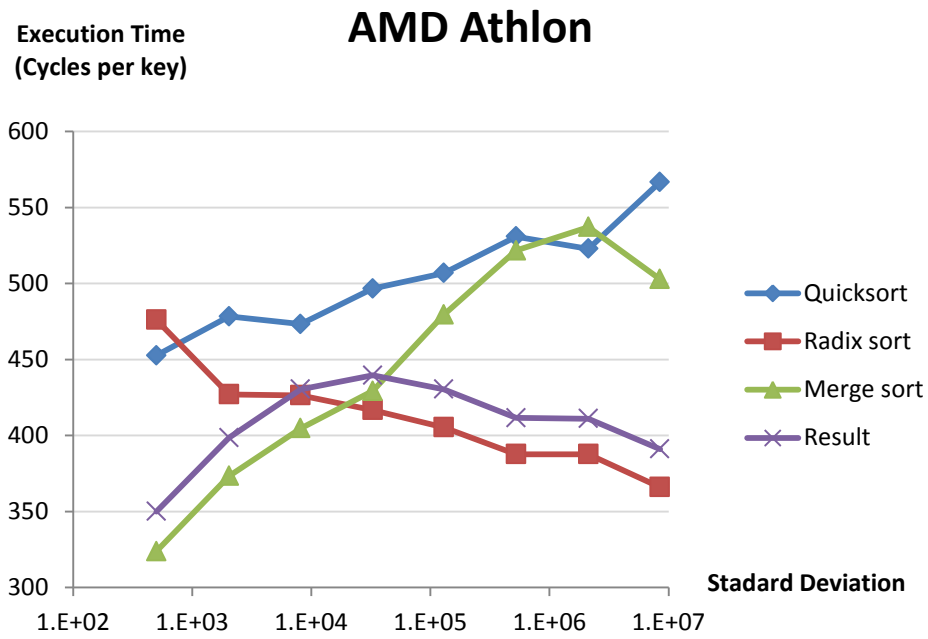
Building the Library



Runtime Procedure



Results



- Library chooses best algorithm
- Overhead of 5%
- On average 44% better than worst algorithm

Final words

- Optimizies Sorting Algorithms
- Works well for unknown input
- Overhead for known data

- Unclear degree of Optimization
- Hard-coded decisions

- Further work: See next presentation