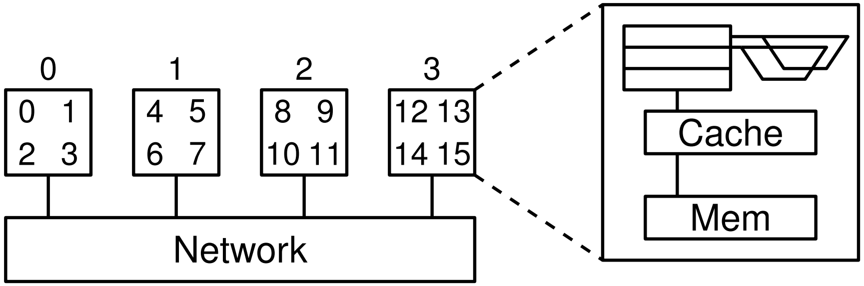
Karlsruhe Institute of Technology

Practical Massively Parallel Sorting

Michael Axtmann, Timo Bingmann, Peter Sanders, Christian Schulz

23th November 2015 @ TU WIEN (SPAA 2015)

Institute of Theoretical Informatics – Algorithmics

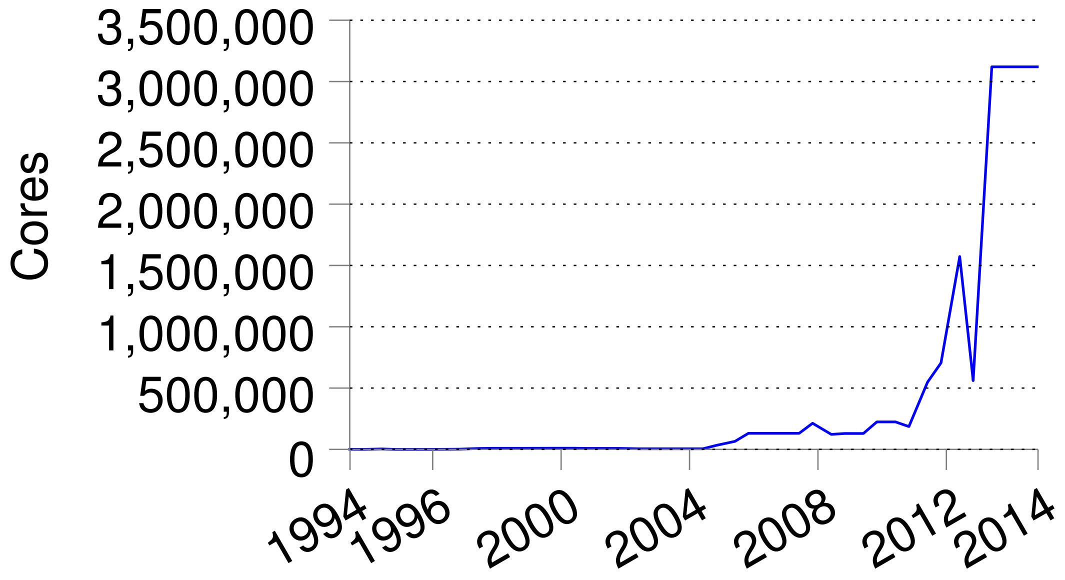


Motivation

Example

- Space-filling curves for load balancing in supercomputers
- Relatively small input

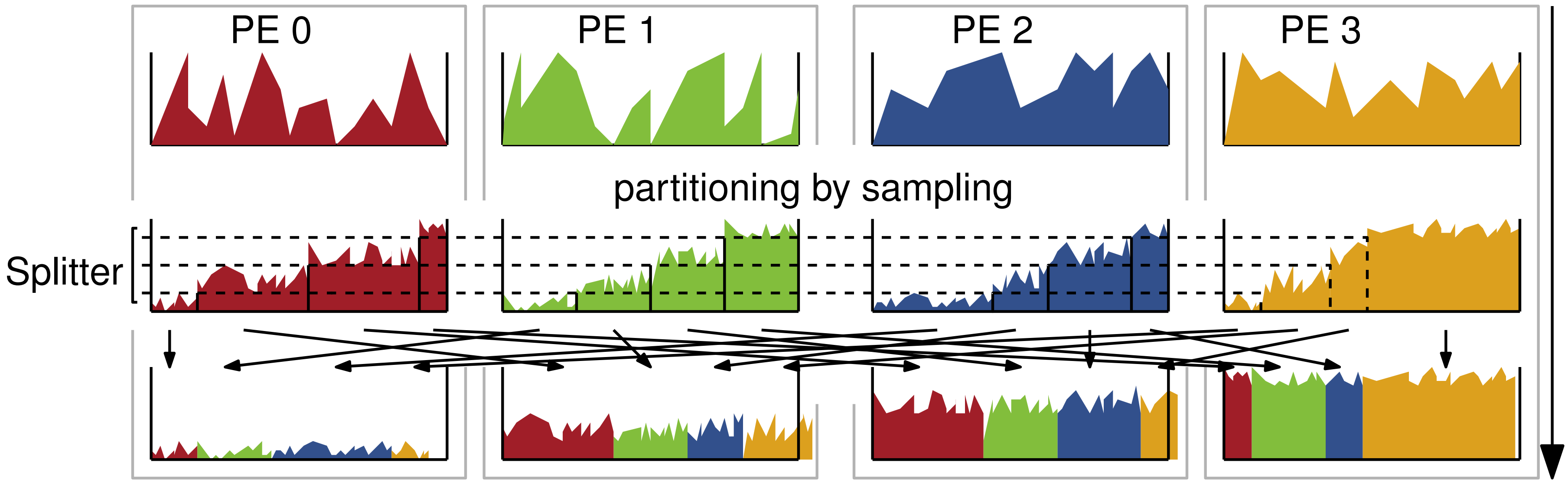
Development over time: cores of the #1 supercomputer



Data source: TOP500 November 2014

p-way Sample Sort

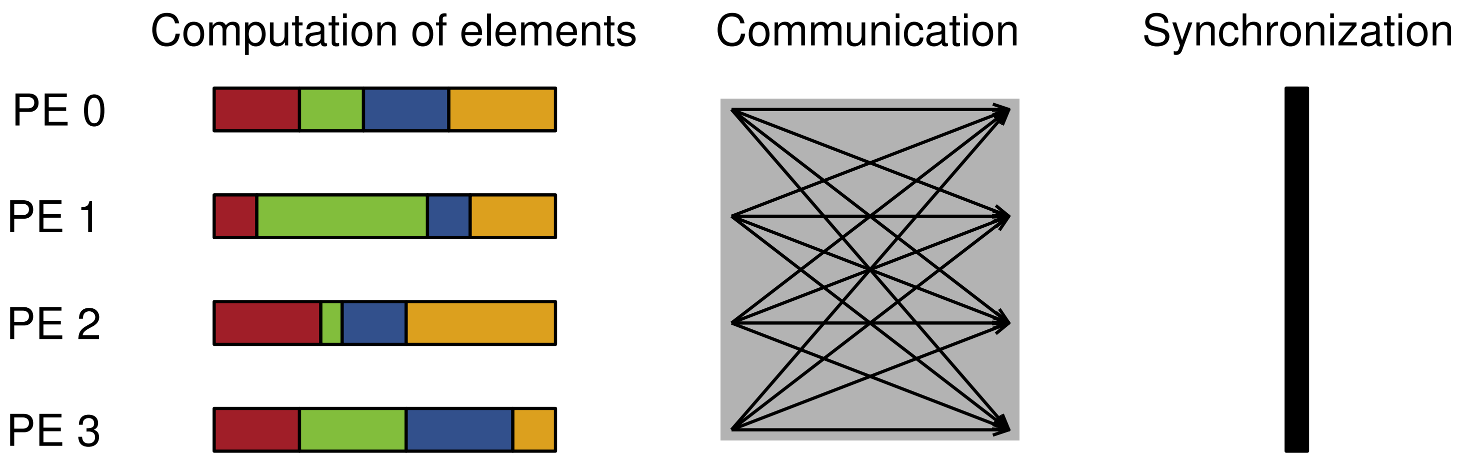
- Input: large n
- Many processing elements (PE) p
- Delivering data once



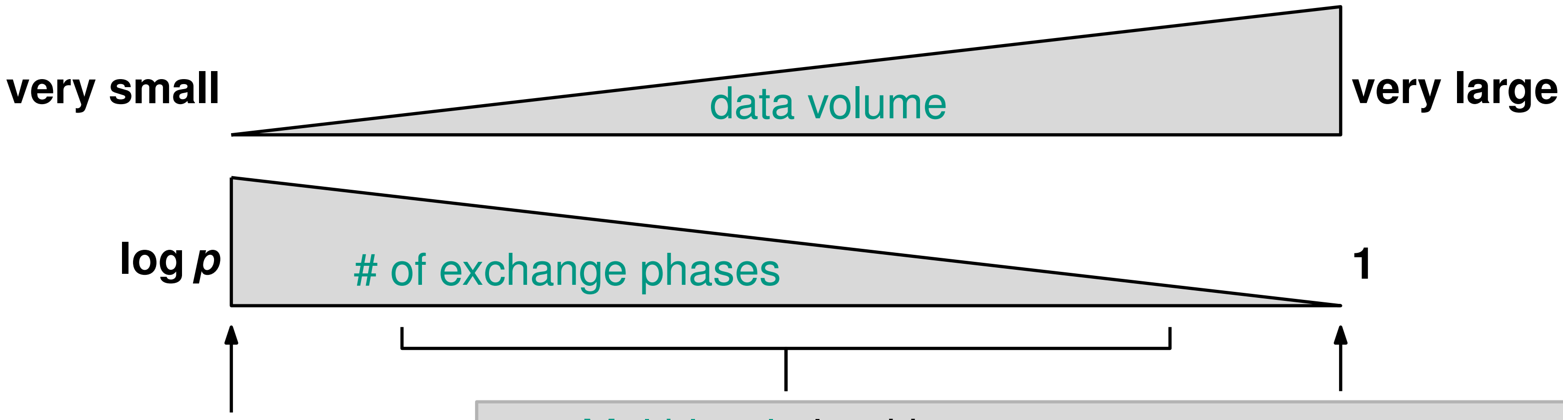
G. E. Blelloch et al. *3rd SPAA*

BSP Model

- Bulk synchronous
- Data exchange: p startups in practice



Massively Parallel Sorting Algorithms



Merge sort, quick sort [1]

Multi-level algorithms in the BSP model [2, 3]

p-way parallel sample sort [4]

!

Worst case: receive $\Theta(p)$ messages

[1] J. Jaja. *An Introduction to Parallel Algorithms*, 1992

[2] A. Gerbessiotis and L. Valiant. *JPDC*, 1994

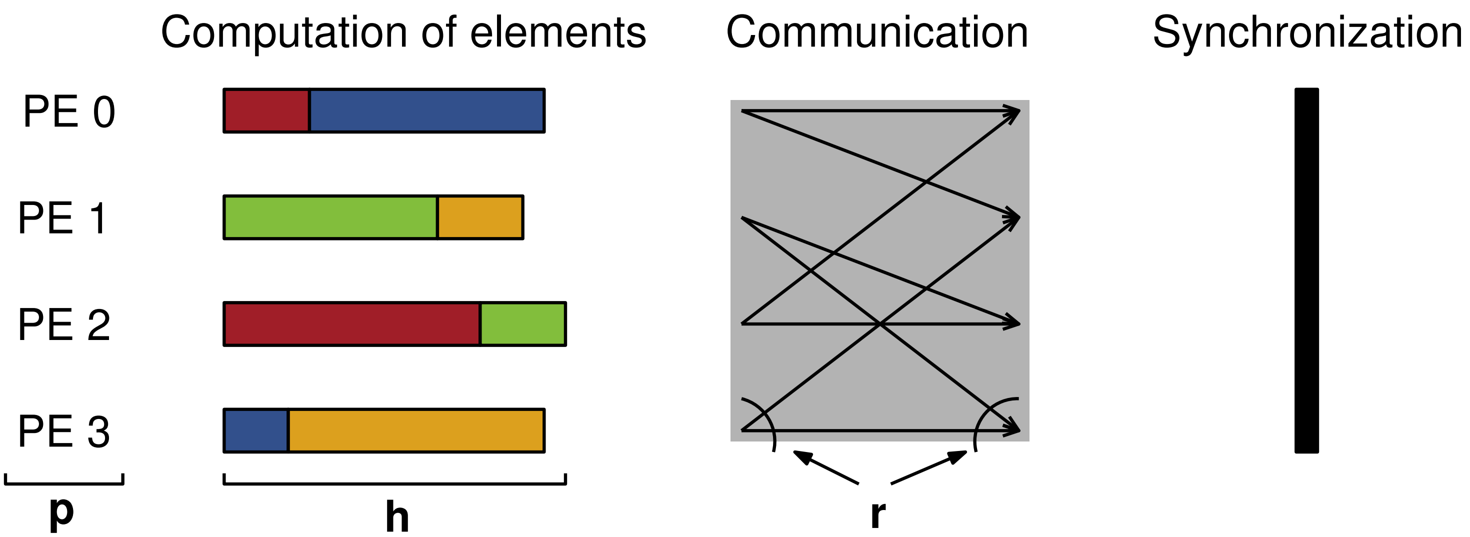
[3] M. T. Goodrich. *SICOMP*, 1999

[4] G. E. Blelloch et al. *3rd SPAA*, 1991

Model of Computation

BSP model generalization

- Data exchange function: $\text{Exch}(p, h, r)$
  - p ← involved PEs
  - h ← max send/receive volume per PE
  - r ← send/recv messages per PE



Comparison

Assumptions

- Number of levels $k \in \mathcal{O}(1)$
- Single-ported message passing
  - Sending of ℓ machine words: $\alpha + \beta\ell$

Algorithm	**Isoefficiency function**
p-way parallel sample sort [1]	$\mathcal{O}(p^2 \cdot \frac{1}{\log p})$
Multi-level BSP-based [2,3]	$\Omega(p^2 \cdot \frac{1}{\log p})$ in our model

[1] G. E. Blelloch et al. *3rd SPAA*, 1991
[2] A. Gerbessiotis and L. Valiant. *JPDC*, 1994
[3] M. T. Goodrich. *SICOMP*, 1999

Comparison

Karlsruhe Institute of Technology

Assumptions

- Number of levels $k \in \mathcal{O}(1)$
- Single-ported message passing
  - Sending of ℓ machine words: $\alpha + \beta\ell$

Algorithm	Isoefficiency function
p-way parallel sample sort [1]	$\mathcal{O}(p^2 \cdot \frac{1}{\log p})$
Multi-level BSP-based [2,3]	$\Omega(p^2 \cdot \frac{1}{\log p})$ in our model
Multi-level merge sort	$\mathcal{O}(p^{1+\frac{1}{k}} \cdot \log p)$
Multi-level sample sort	$\mathcal{O}(p^{1+\frac{1}{k}} \cdot \frac{1}{\log p})$

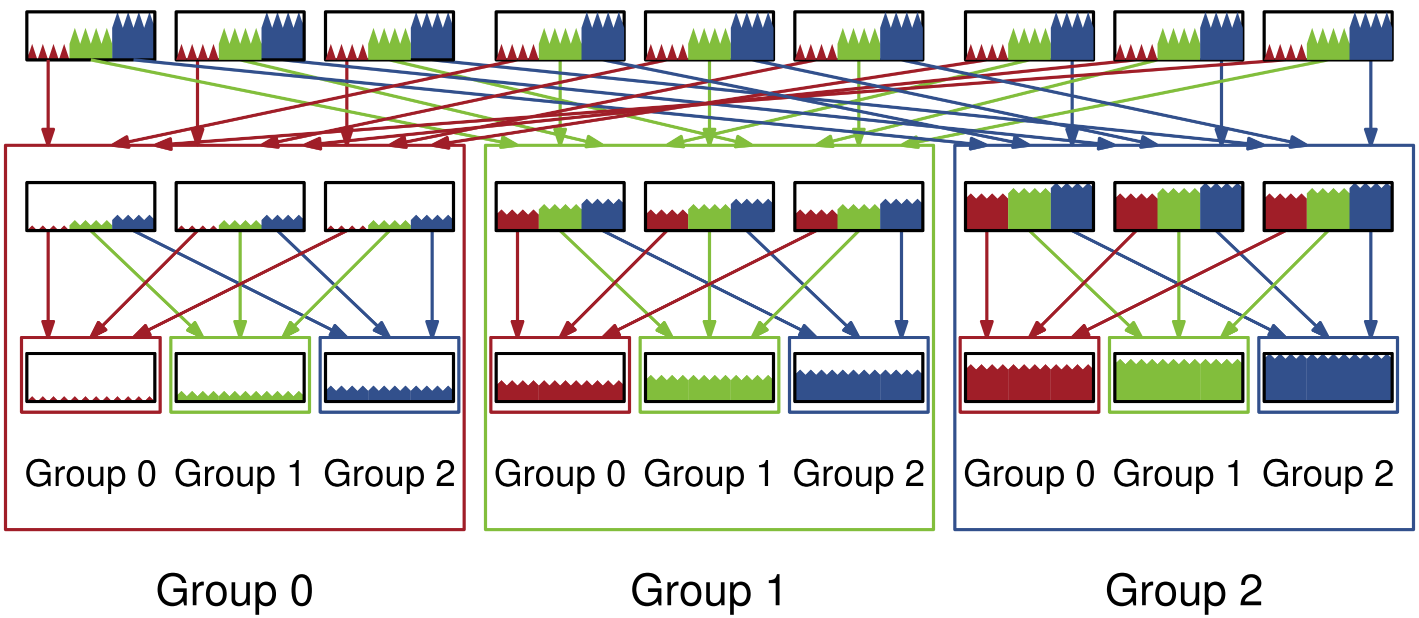
[1] G. E. Blelloch et al. *3rd SPAA*, 1991

[2] A. Gerbessiotis and L. Valiant. *JPDC*, 1994

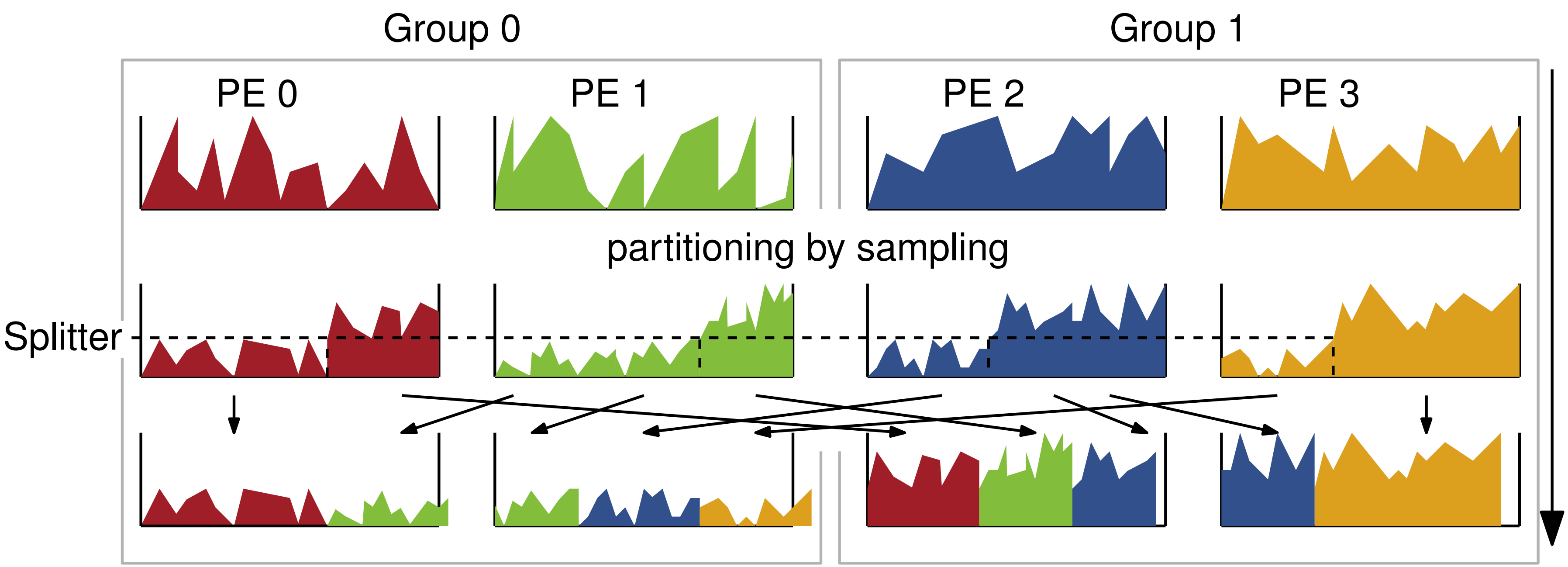
[3] M. T. Goodrich. *SICOMP*, 1999

Multi-Level Sorting Approach

- Subdivide PEs into groups
- Move data to suitable group
- *k* levels of recursion
  - Groups $r \approx \sqrt[k]{p}$



Adaptive Multi-Level Sample Sort



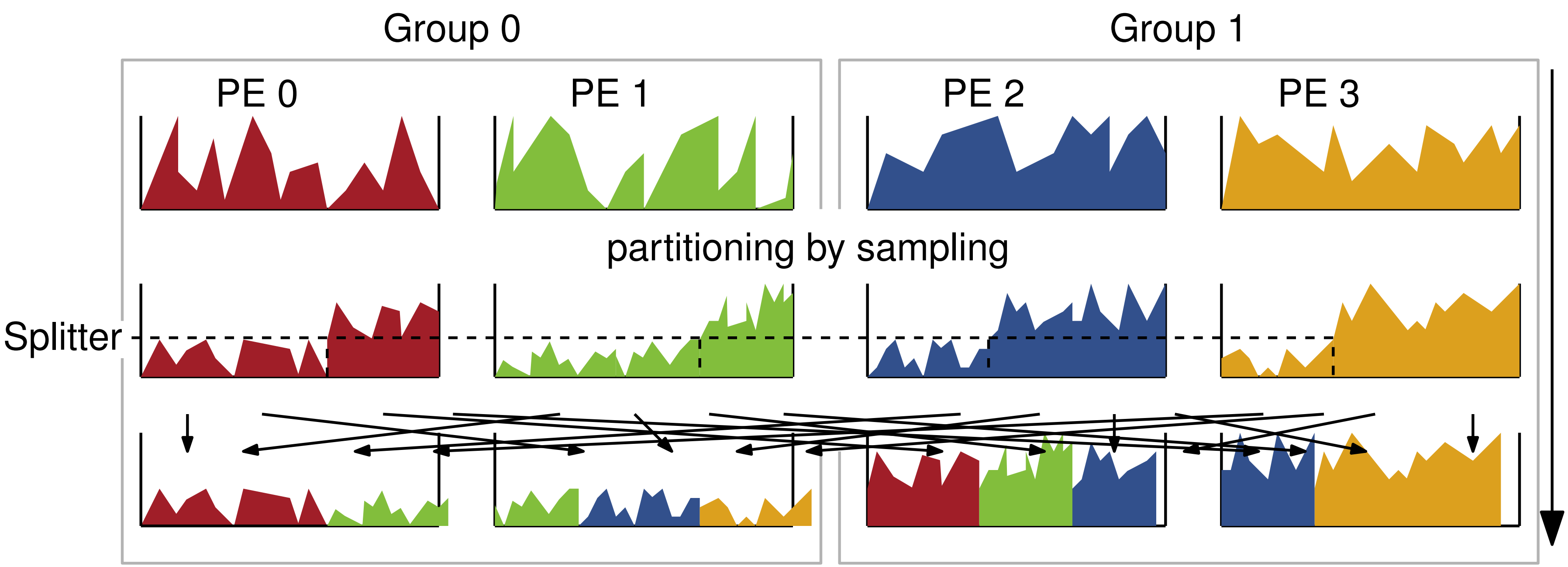
recurse on $\frac{n}{r}(1+\epsilon)$ items with $\frac{n}{r}$ PEs

recurse on $\frac{n}{r}(1-\epsilon)$ items with $\frac{n}{r}$ PEs

Requirements

- Fast parallel sorting of samples
- Sample reduction by overpartitioning
- Reduce startup overheads to $\mathcal{O}(k\sqrt[k]{p})$

Adaptive Multi-Level Sample Sort



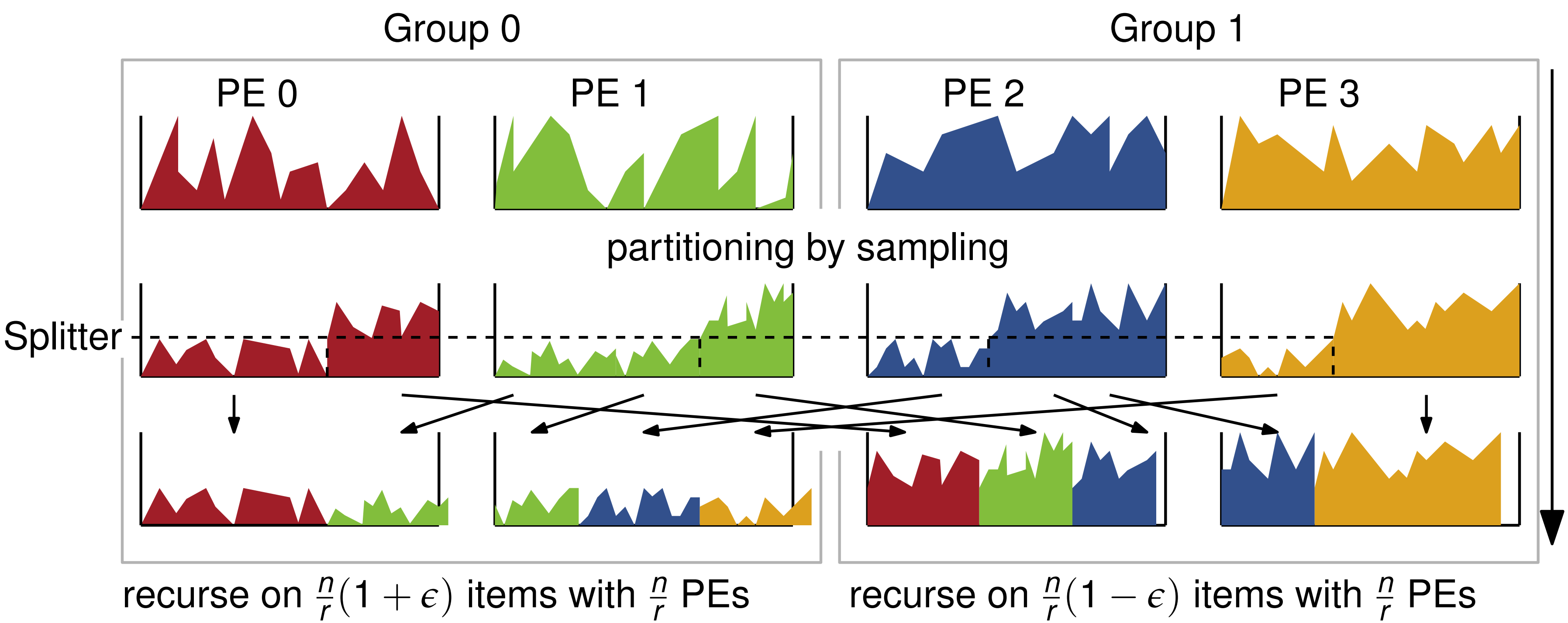
recurse on $\frac{n}{r}(1+\epsilon)$ items with $\frac{n}{r}$ PEs

recurse on $\frac{n}{r}(1-\epsilon)$ items with $\frac{n}{r}$ PEs

Requirements

- Fast parallel sorting of samples
- Sample reduction by overpartitioning
- Reduce startup overheads to $\mathcal{O}(k\sqrt[k]{p})$

Adaptive Multi-Level Sample Sort



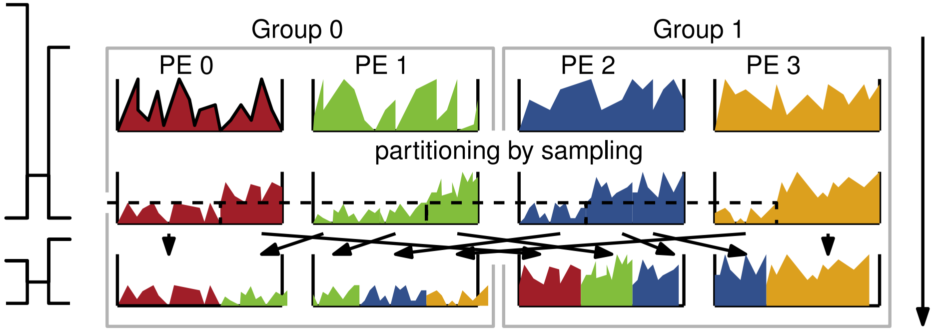
Requirements

- Fast parallel sorting of samples
- Sample reduction by overpartitioning
- Reduce startup overheads to $\mathcal{O}(k\sqrt[k]{p})$

Adaptive Multi-Level Sample Sort

Open submodules

1. Fast sample sorting
   - Oversampling
2. Optimal overpartitioning
3. Group-based data delivery



Fast Parallel Sample Sorting

- Parallel sorting of s samples
- Rectangular $a \times b$ array of PEs

PE 0 **v x**	PE 1 **d t**	PE 2 **h g**
PE 3 **o q**	PE 4 **r i**	PE 5 **l w**
PE 6 **f m**	PE 7 **p c**	PE 8 **b k**

Fast Parallel Sample Sorting

- Parallel sorting of s samples
- Rectangular $a \times b$ array of PEs

Local sort

PE 0 v x	PE 1 d t	PE 2 g h
PE 3 o q	PE 4 i r	PE 5 l w
PE 6 f m	PE 7 c p	PE 8 b k

Fast Parallel Sample Sorting

- Parallel sorting of s samples
- Rectangular $a \times b$ array of PEs

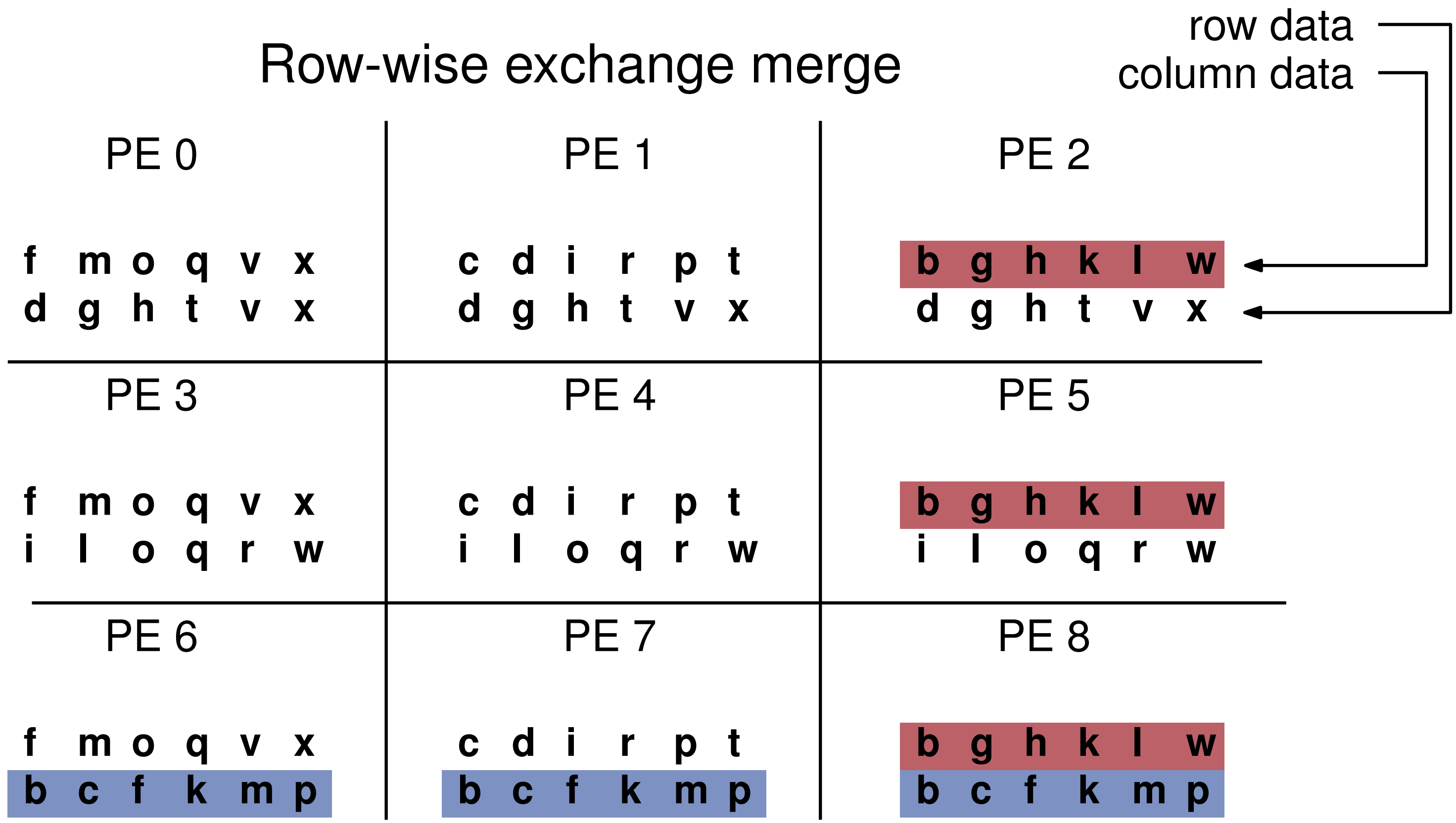
Column-wise exchange merge

column data →

PE 0	PE 1	PE 2
f m o q v x x v	c d i r p t d t	b g h k l w g h
PE 3	**PE 4**	**PE 5**
f m o q v x o q	c d i r p t i r	b g h k l w l w
PE 6	**PE 7**	**PE 8**
f m o q v x m f	c d i r p t p c	b g h k l w k b

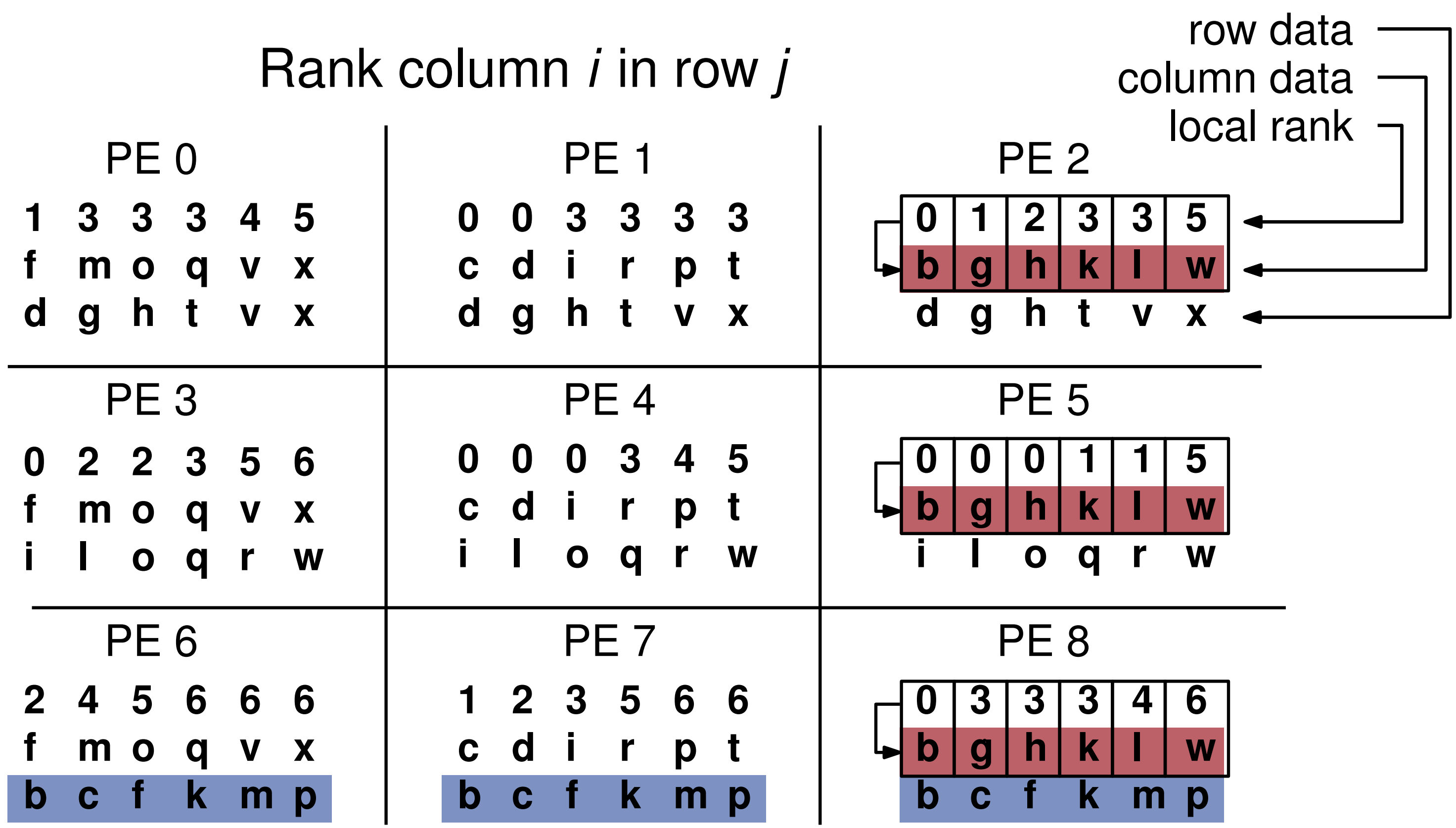
Fast Parallel Sample Sorting

- Parallel sorting of s samples
- Rectangular $a \times b$ array of PEs



Fast Parallel Sample Sorting

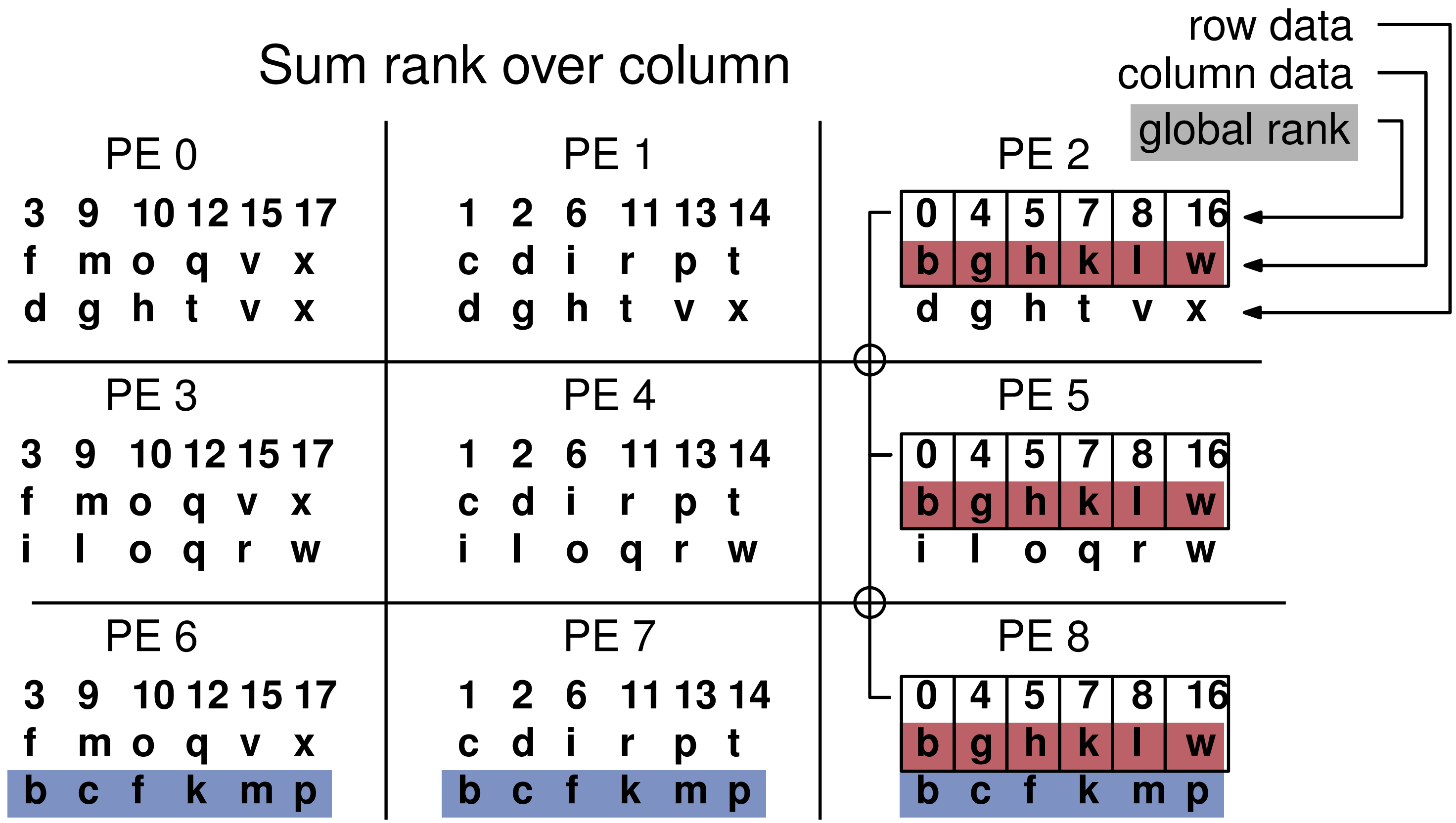
- Parallel sorting of s samples
- Rectangular $a \times b$ array of PEs



Fast Parallel Sample Sorting

- Parallel sorting of s samples
- Rectangular $a \times b$ array of PEs

Sum rank over column



Fast Parallel Sample Sorting

Single-ported message passing

- Sending of ℓ machine words: $\alpha + \beta\ell$

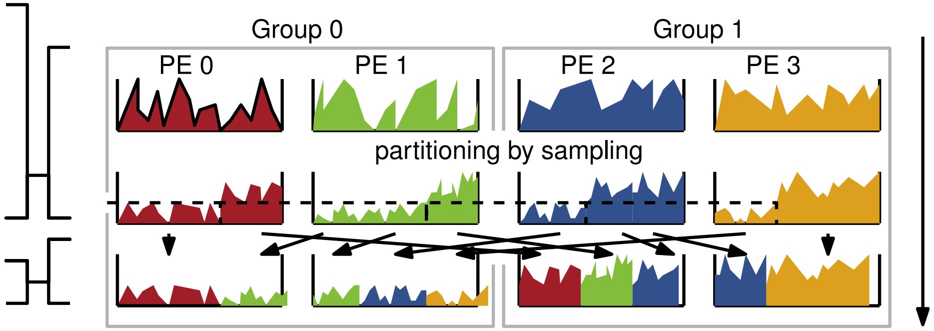
- Local sort
- Column-wise allgather merge
- Row-wise allgather merge
- Rank column *i* in row *j*
- Sum rank over column

$$\mathcal{O}\left(\alpha \log p + \beta\frac{s}{\sqrt{p}} + \frac{s}{p}\log\frac{s}{p}\right)$$

Adaptive Multi-Level Sample Sort

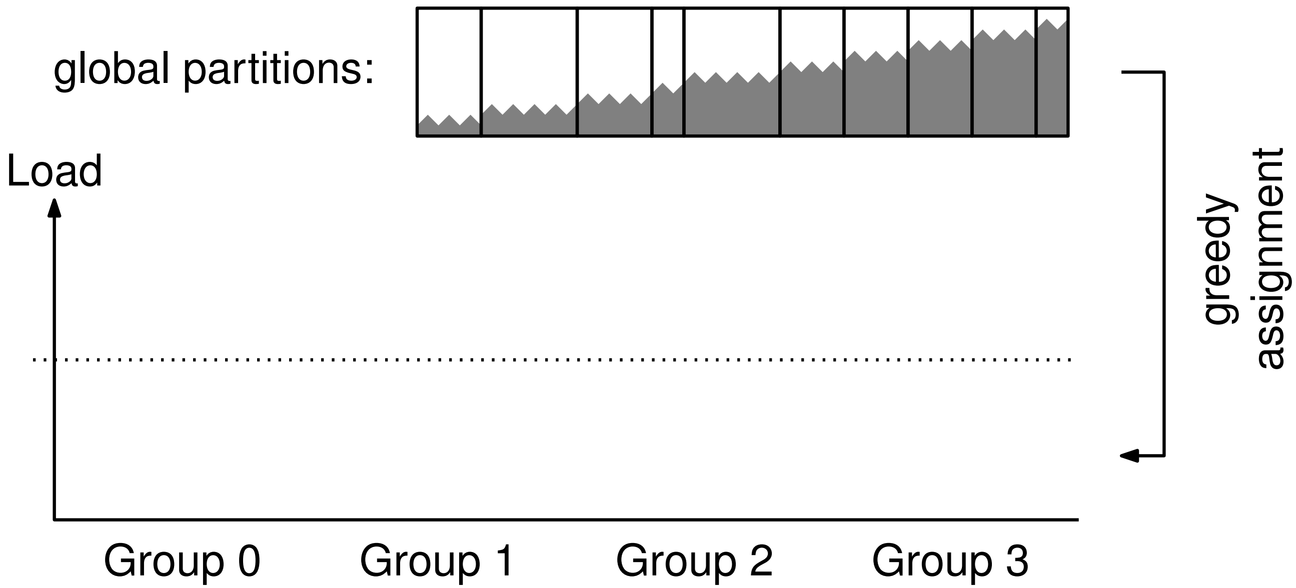
Open submodules

1. Fast sample sorting
   - Oversampling
2. Optimal overpartitioning
3. Group-based data delivery



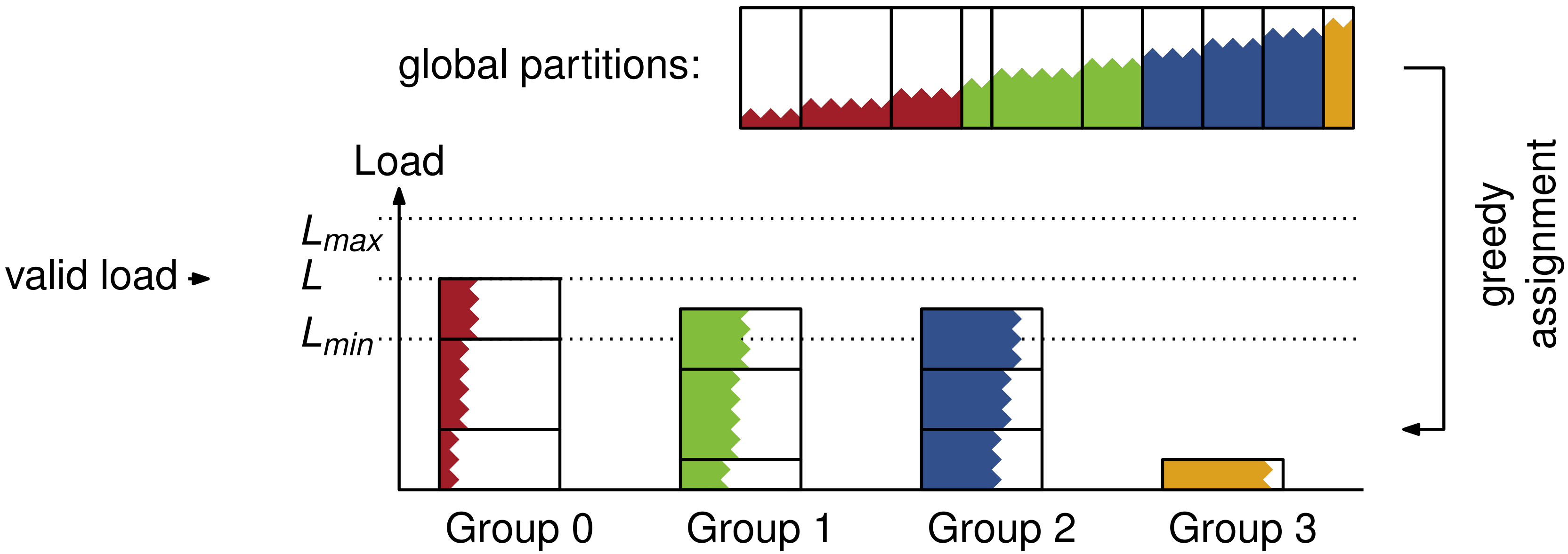
Optimal Overpartitioning

- Requirement: $L_{max} = (1 + \epsilon)\frac{n}{r}$ with high probability
- Oversampling: a
- Overpartitioning: $b \in \Theta(\frac{1}{\epsilon})$



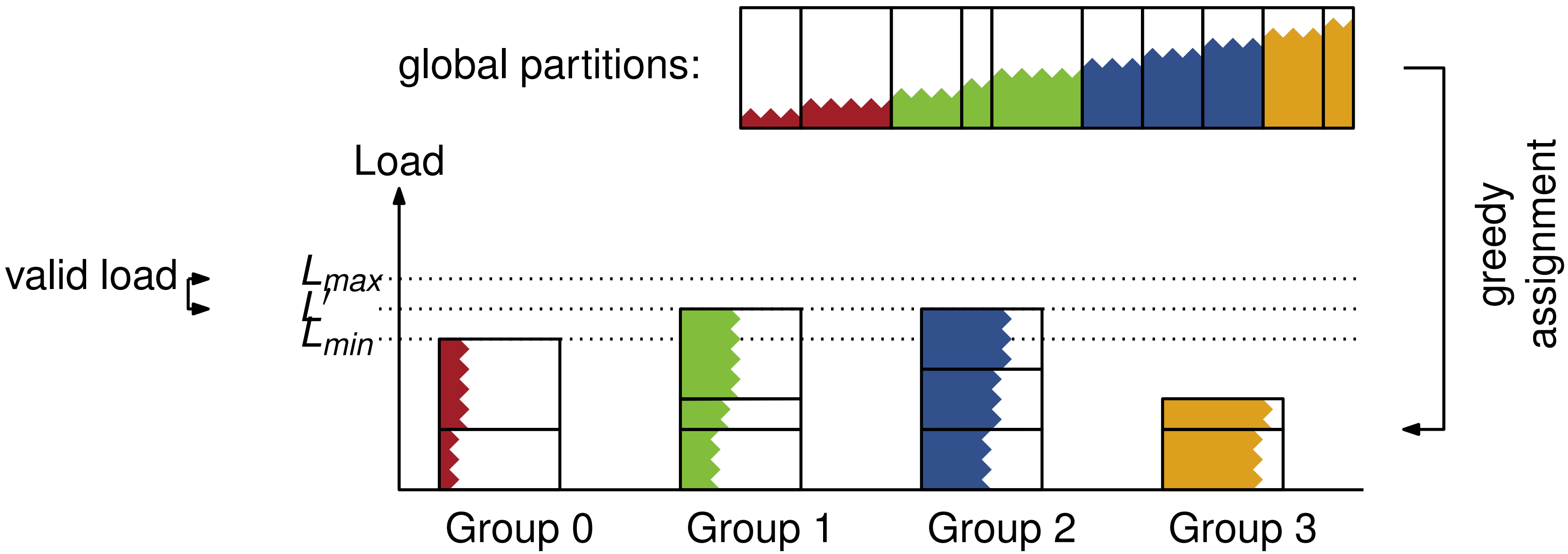
Optimal Overpartitioning

- Requirement: $L_{max} = (1 + \epsilon)\frac{n}{r}$ with high probability
- Oversampling: a
- Overpartitioning: $b \in \Theta(\frac{1}{\epsilon})$



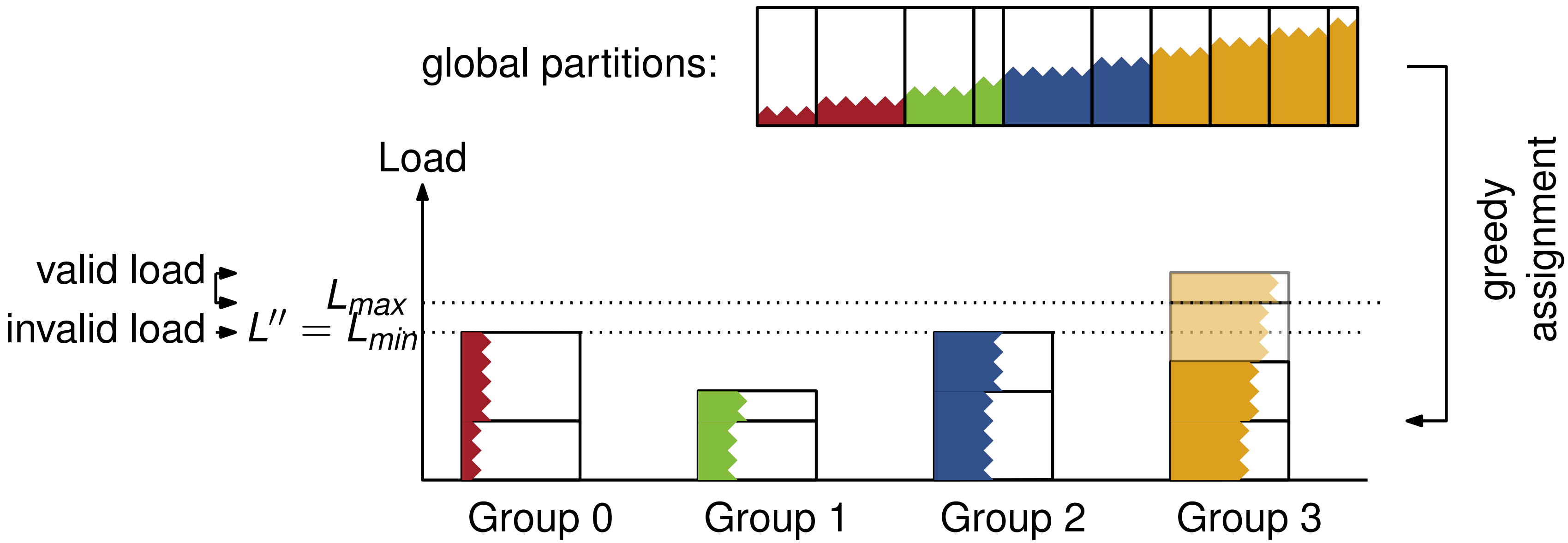
Optimal Overpartitioning

- Requirement: $L_{max} = (1 + \epsilon)\frac{n}{r}$ with high probability
- Oversampling: a
- Overpartitioning: $b \in \Theta(\frac{1}{\epsilon})$



Optimal Overpartitioning

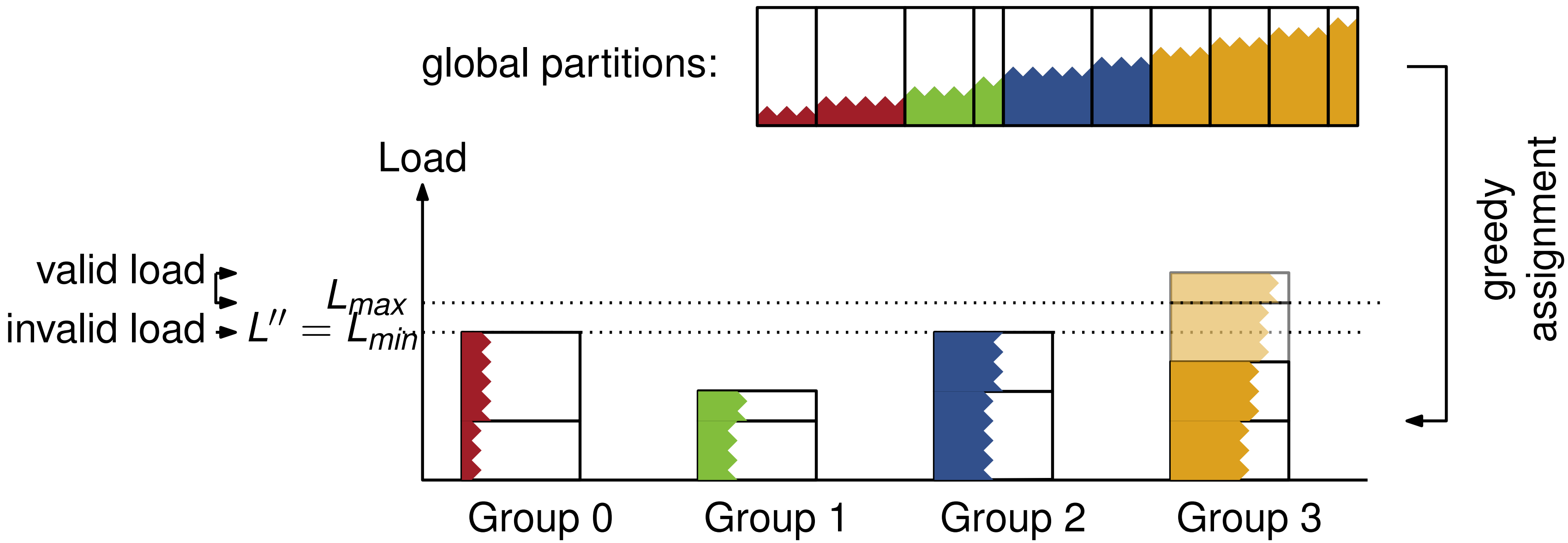
- Requirement: $L_{max} = (1+\epsilon)\frac{n}{r}$ with high probability
- Oversampling: a
- Overpartitioning: $b \in \Theta(\frac{1}{\epsilon})$



Optimal Overpartitioning

- Requirement: $L_{max} = (1+\epsilon)\frac{n}{r}$ with high probability
- Oversampling: a
- Overpartitioning: $b \in \Theta(\frac{1}{\epsilon})$
- Fewer samples $abr \in \Theta(r \log r)$

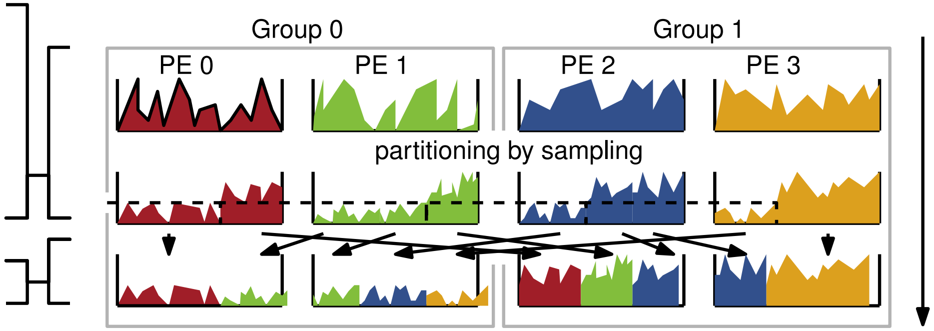
$$\mathcal{O}(br + \alpha \log p)$$



Adaptive Multi-Level Sample Sort

Open submodules

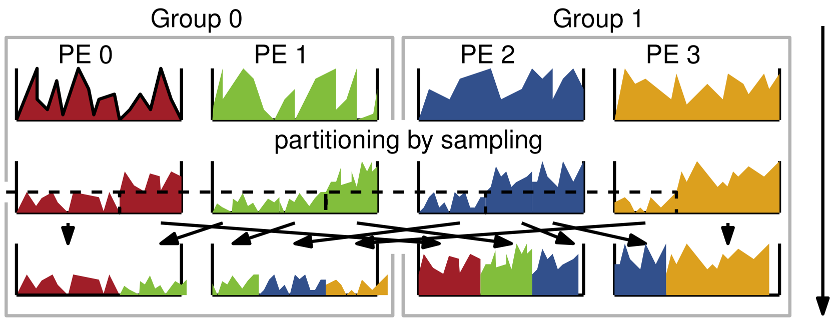
1. Fast sample sorting
   - Oversampling
2. Optimal overpartitioning
3. Group-based data delivery



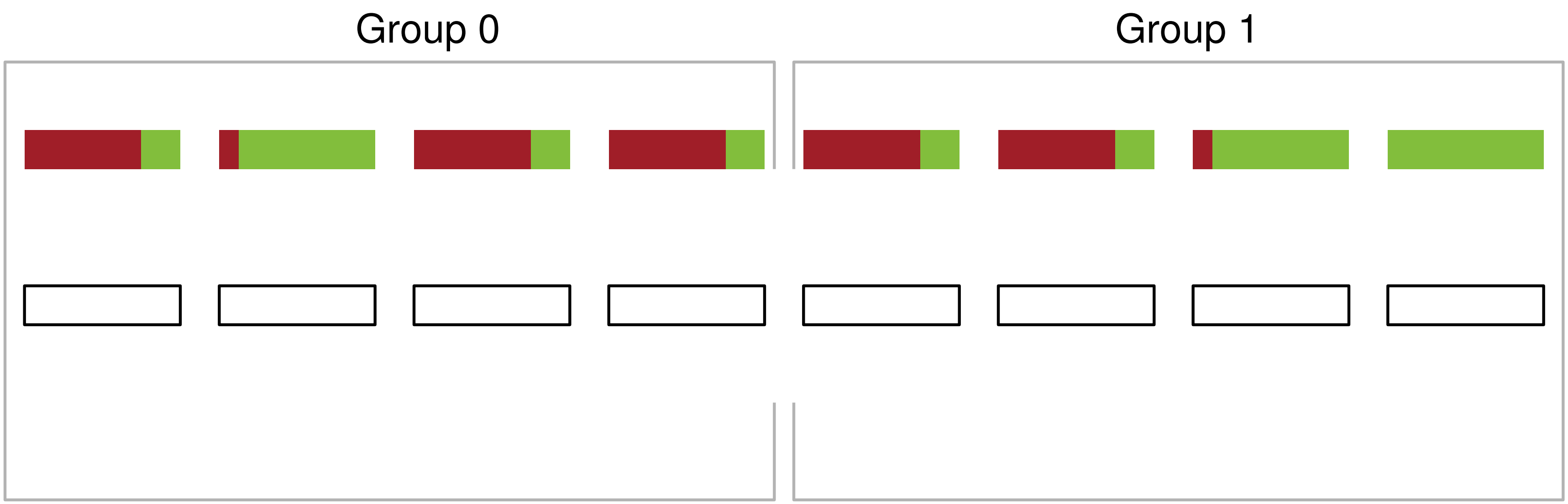
Group-Based Data Delivery

Goal

- Partition i to group i
- Each PE in group receives same amount of data
- $(1 + o(1))\text{Exch}(p, \frac{n}{p}, \mathcal{O}(r))$
  - p: involved PEs
  - $\frac{n}{p}$: max send/receive volume per PE
  - $\mathcal{O}(r)$: send/recv messages per PE
- Reduce startup overheads to $\mathcal{O}(\sqrt[k]{p})$



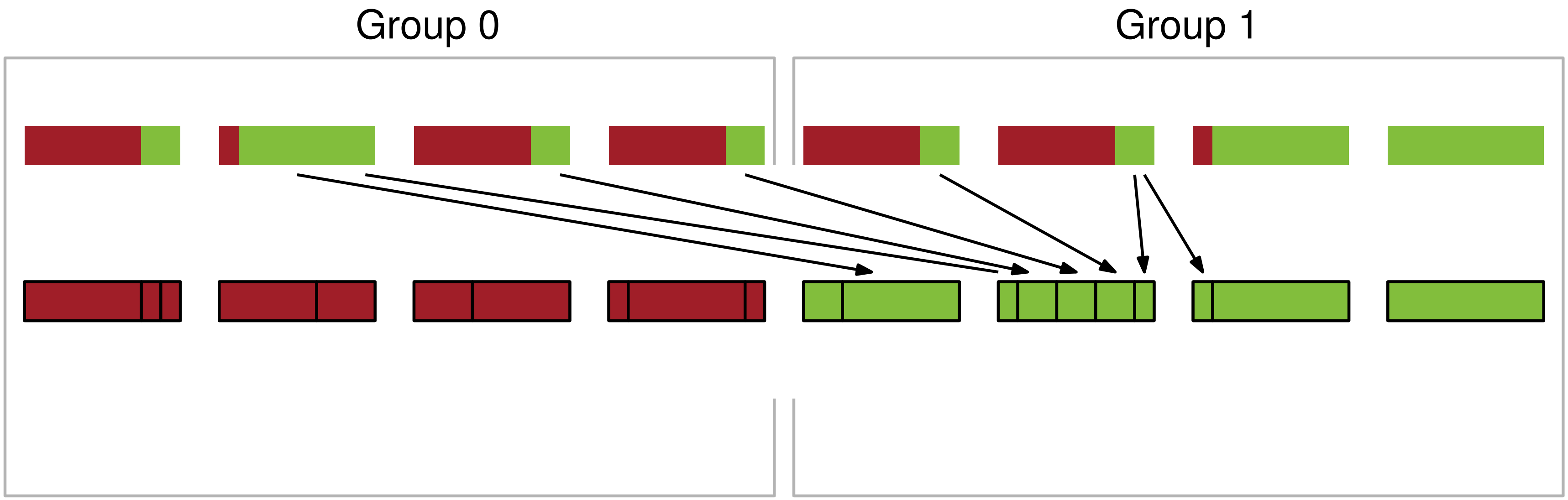
Group-Based Data Delivery



Group-Based Data Delivery

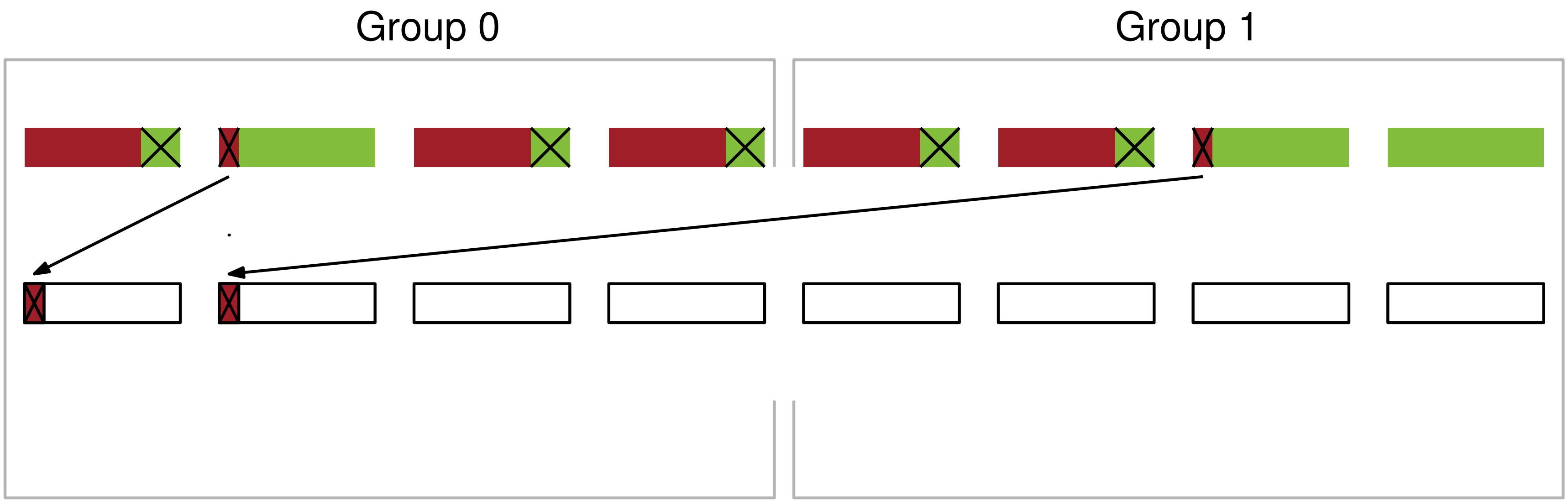
Karlsruhe Institute of Technology

Trivial approach



Group-Based Data Delivery

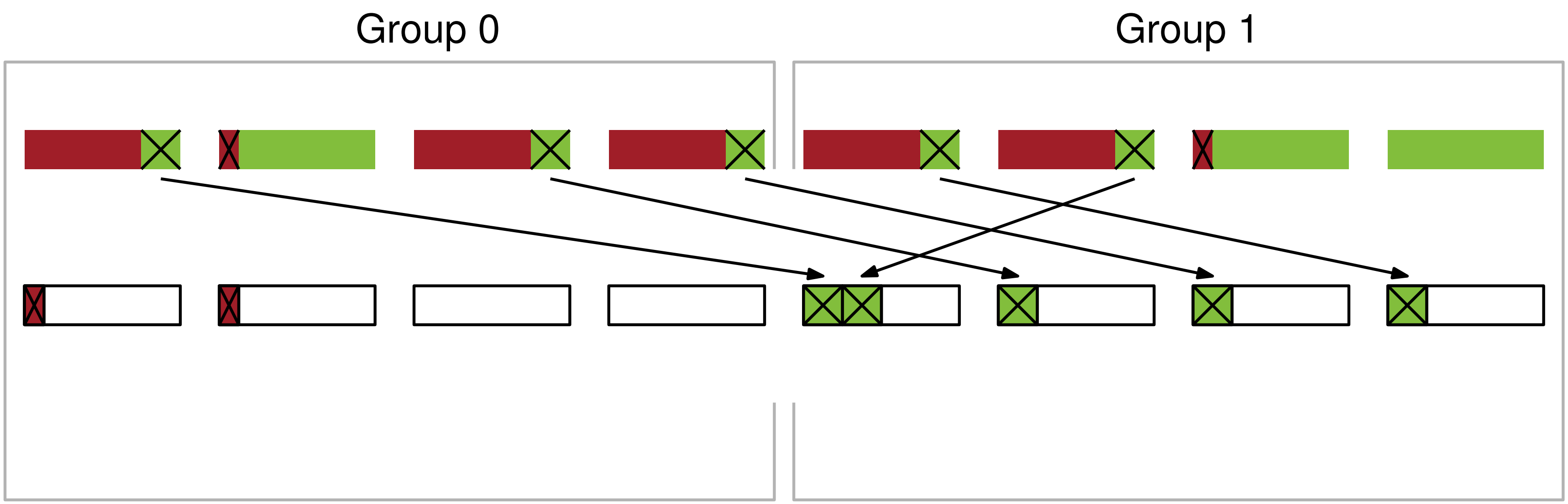
Our approach



- Distribution of small pieces $|\cdot| \leq \frac{n}{2pr}$ // round-robin

Group-Based Data Delivery

Our approach

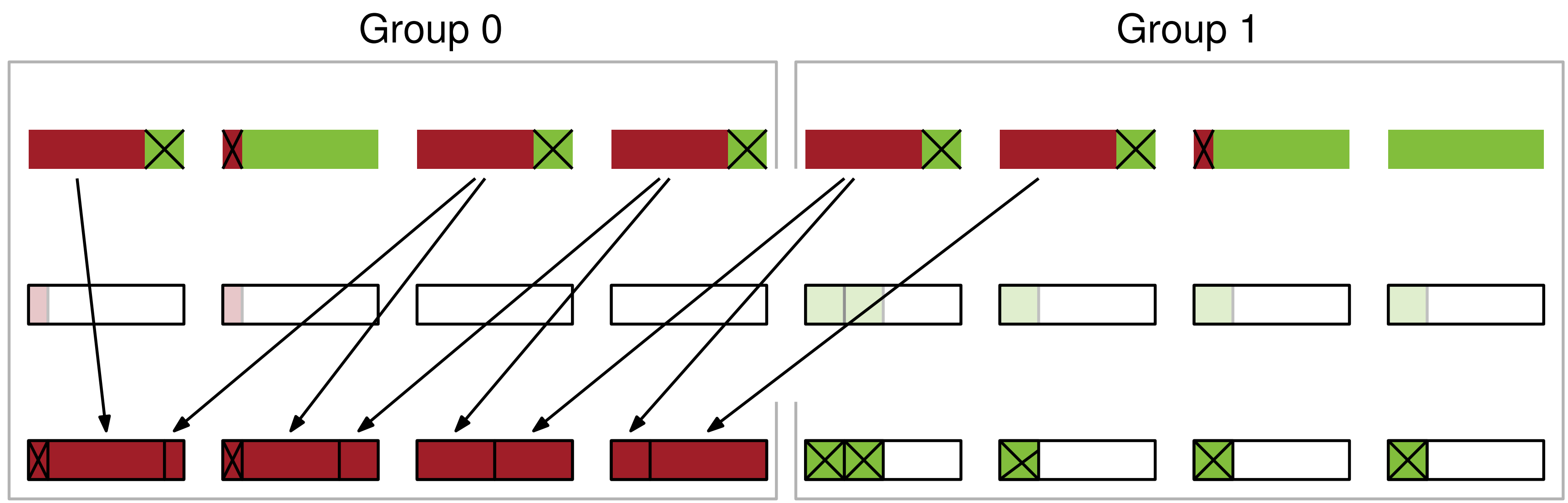


- Distribution of small pieces $|\cdot| \leq \frac{n}{2pr}$ // round-robin

Group-Based Data Delivery

Karlsruhe Institute of Technology

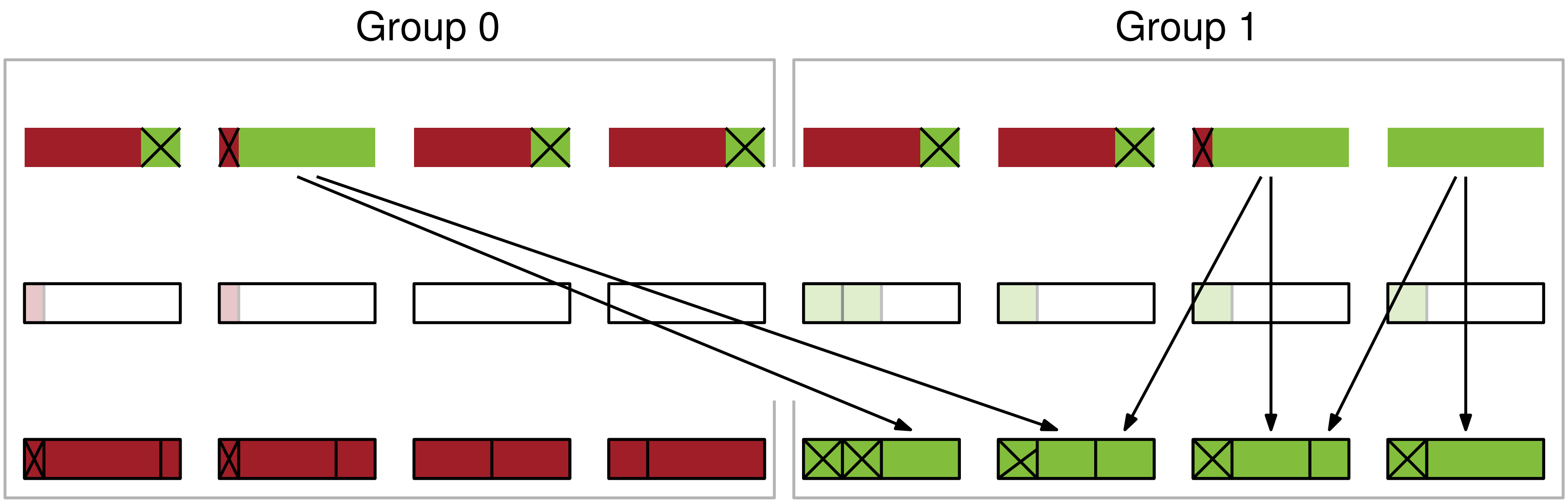
Our approach



- Distribution of small pieces $|\cdot| \leq \frac{n}{2pr}$ // round-robin
- Distribution of large pieces // prefix-sum and merging

Group-Based Data Delivery

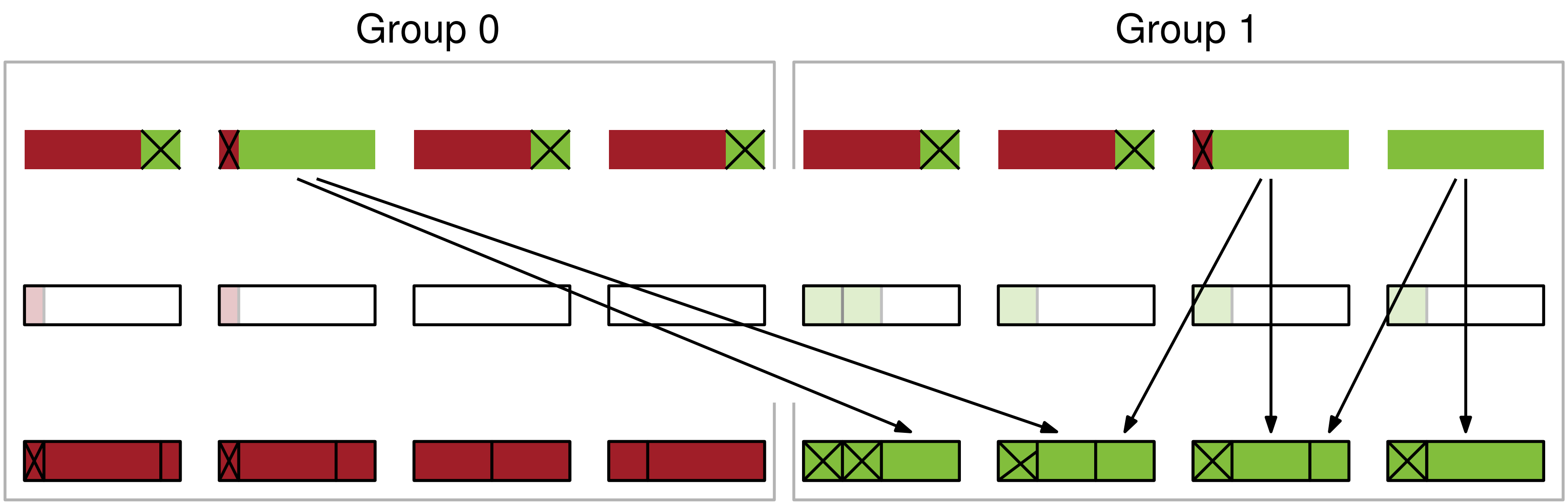
Our approach



- Distribution of small pieces $|\cdot| \leq \frac{n}{2pr}$ // round-robin
- Distribution of large pieces // prefix-sum and merging

Group-Based Data Delivery

Our approach



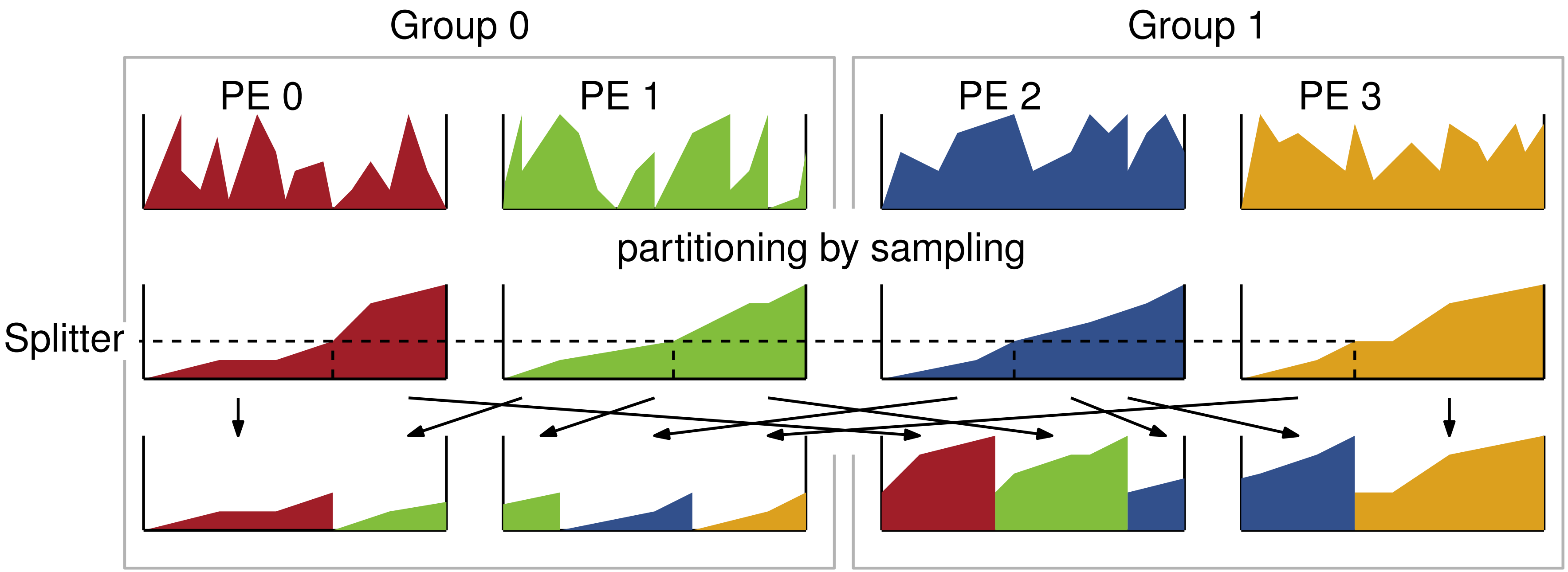
- Distribution of small pieces $|\cdot| \leq \frac{n}{2pr}$ // round-robin
- Distribution of large pieces // prefix-sum and merging

Reduces startup overheads to $\mathcal{O}(\sqrt[k]{p})$

Recurse Last Multiway Mergesort

Highlights

- Multisequence selection
- Perfect load balance
- Reduces startup overheads to $\mathcal{O}(k \sqrt[k]{p})$



recurse on $\frac{n}{r}$ items with $\frac{n}{r}$ PEs

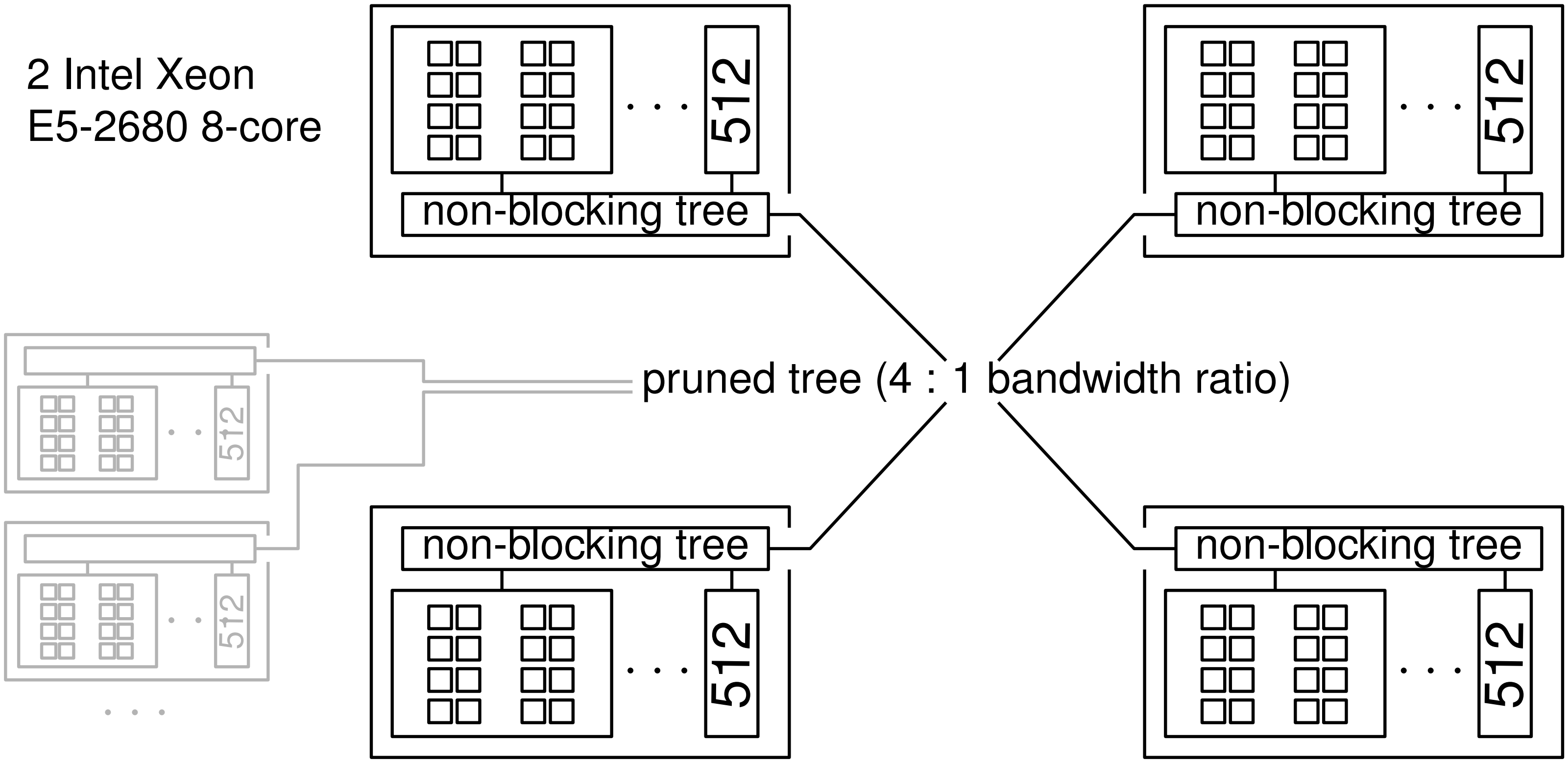
recurse on $\frac{n}{r}$ items with $\frac{n}{r}$ PEs

Experiments

System

SuperMUC in Munich

2 Intel Xeon
E5-2680 8-core



Cores used: 32 768 (4 islands)

Experiments

Karlsruhe Institute of Technology

Sample sort median wall-times in seconds

n/p	p = 512	2 048	8 192	32 768
10^5	0.0228	0.0277	0.0359	0.0707
10^6	0.2212	0.2589	0.2687	0.9171
10^7	2.6523	2.9797	4.0625	6.0932

Speedup of sample sort compared to sequential sort

n/p	p = 512	2 048	8 192	32 768
10^5	273	956	3 208	6 929
10^6	321	1 146	4 747	6 164
10^7	295	1 124	–	–

Level 1 Level 2 Level 3

Comparison to Literature

Solomonik and Kale [1]: CrayXT 4

- Slower processors, higher bandwidth
- $n = 8 \cdot 10^6 p$, up to $p = 2^{15}$ // vs. $n_{\text{ref}} = 10^7 p$
  - Similar performance

MP-sort [2]: Cray XE6

- $n = 10^5 p$ and $p = 2^{14}$ // vs. $p_{\text{ref}} = 2^{15}$
  - 289 times faster
- 6 times faster for large inputs

[1] Solomonik and Kale. *IPDPS 2010*

[2] Y. Feng et al. *2014*

Conclusion

Karlsruhe Institute of Technology

Result

- Scalable in theory and practice
- Improved wall-time: large p and moderate n
- Competitive: large p and large n

Future work

- Perform experiments with more PEs
- Shared memory on node-local level
- Better exchange algorithms
- Fault tolerance

Acknowledgement

The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC at Leibniz Supercomputing Centre (LRZ, www.lrz.de).