Forschungszentrum Karlsruhe
in der Helmholtz-Gemeinschaft

Universität Karlsruhe

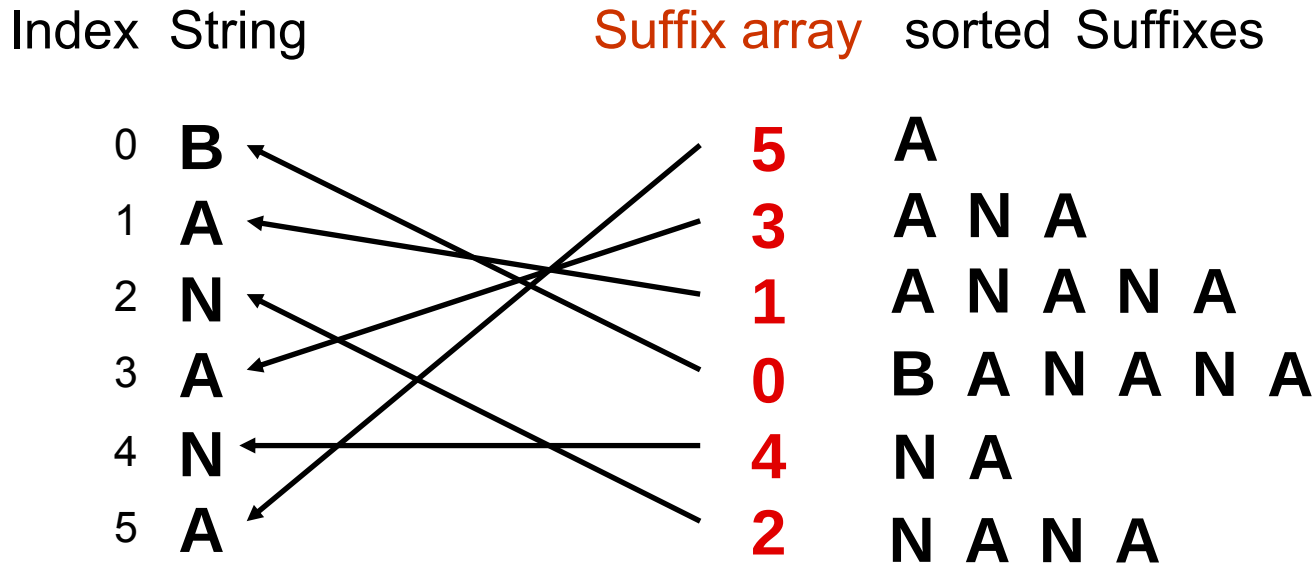
Scalable Parallel Suffix Array Construction

Fabian Kulla and Peter Sanders
Karlsruhe Institute of Technology

Outline

- Suffix array
- Motivation
- Parallel Difference Cover 3 algorithm
- Evaluation

Suffix array (example)



Motivation

- Fast phrase search in text databases in logarithmical time
  - (Find “to be or not to be”)
- Data compression
  - (e.g. Burrows Wheeler transform)
- Bioinformatics
  - Genome analysis

Sample sort (deterministic)

Input uniformly distributed

1. Sort local
2. Select samples
3. Collect and sort samples
4. Get / Broadcast splitters
5. Partition with splitters
6. Collective exchange of buckets
   all-to-all
7. Merge buckets

	P0	P1	P2
	47 11 0 8 15 9	11 3 50 45 38 60	24 12 34 53 37 28
1. Sort local 2. Select samples	0 8 9 11 15 47	3 11 38 45 50 60	12 24 28 34 37 53
3. Collect and sort samples 4. Get / Broadcast splitters	0 3 11 12 34 45		
5. Partition with splitters	0 8 9 11 15 47	3 11 38 45 50 60	12 24 28 34 37 53
6. Collective exchange of buckets all-to-all	0 8 9 11 3 11	15 38 45 12 24 28 34 37	47 50 60 53
7. Merge buckets	0 3 8 9 11 11	12 15 24 28 34 37 38 45	47 50 53 60

Difference Cover 3 Algorithm (DC3)

- Sort suffixes starting at position $i \bmod 3 \neq 0$
- Sort remaining suffixes with help of already sorted suffix
- Merge both sorted suffix groups

Difference Cover 3 Algorithm (DC3)

- Sort suffixes starting at position *i mod* 3 ≠ 0
  - Sorting triples, recursion with 2/3 of input

```
0 1 2 3 4 5        mod1-suffix:   ANANA , NA
B A N A N A        mod2-suffix:    NANA ,  A
```

- Sort remaining suffixes with help of already sorted suffix
  - Sorting pairs

```
B A N A N A
  2 4   3 1
```

(A,3) < (B,2)

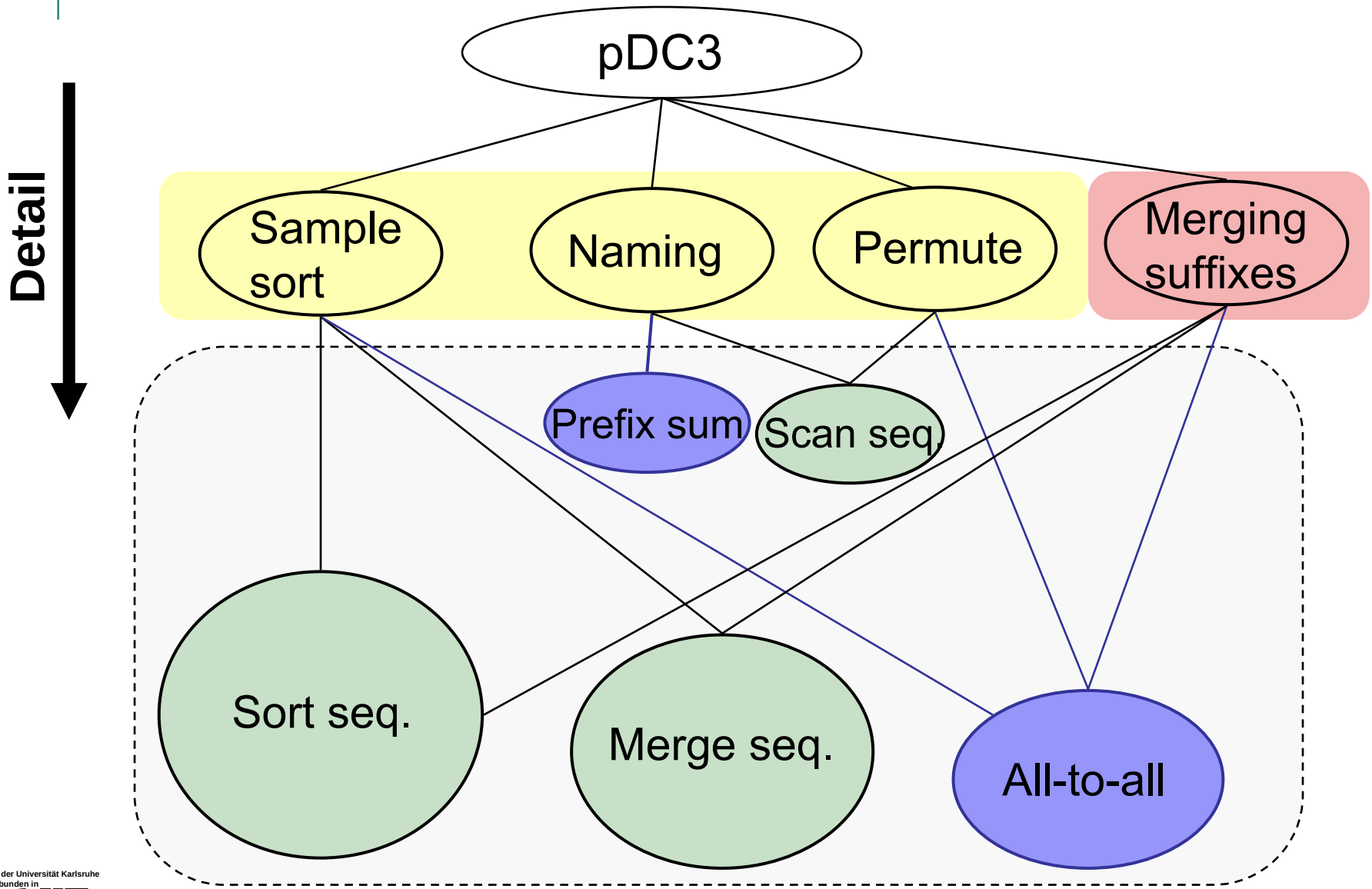
- Merge both sorted suffix groups
  - Comparing pairs and triples

ANA ⇔ ANANA : (A,3) ⇔ (A,4)

ANA ⇔ NANA : (A,N,1) ⇔ (N,A,3)

Note: only parts of suffixes (triples/pairs) are compared

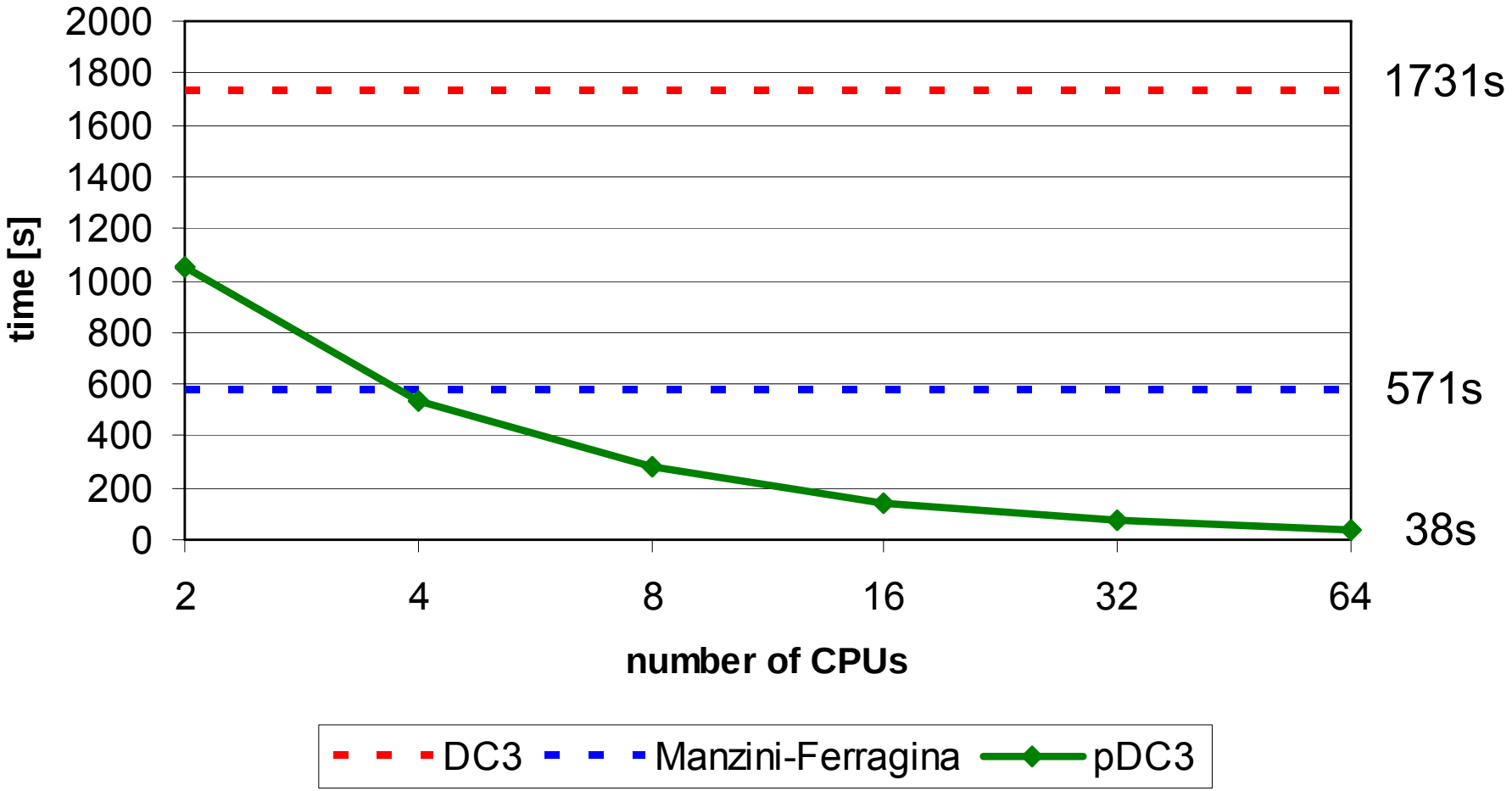
Overview Implementation



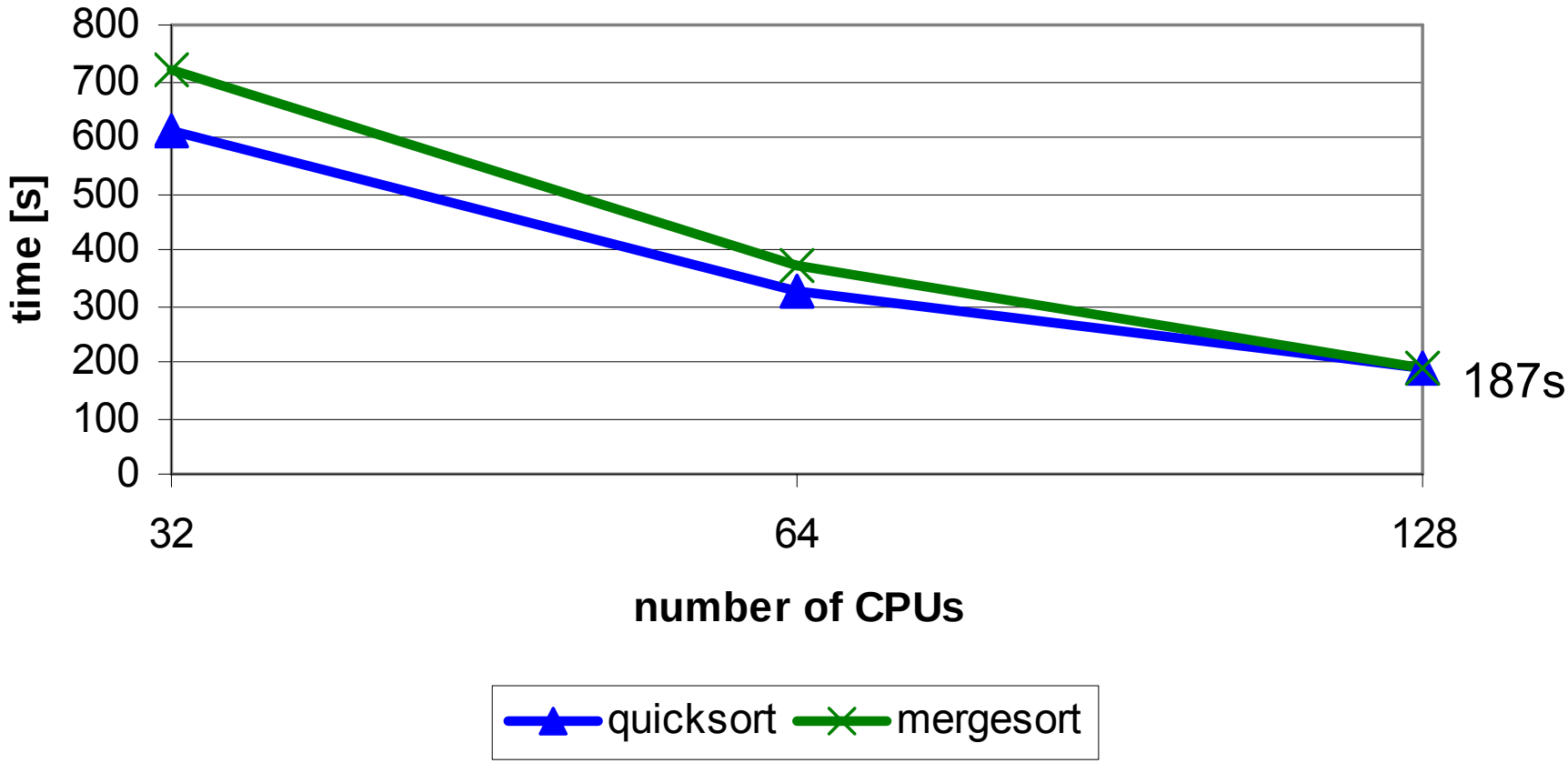
Karlsruhe

- Itanium cluster - University Karlsruhe (XC-Cluster)
  (64 dual processor nodes available;
  each node: 2x 1.5 GHz Itanium 2 and 12 GB main memory)
  - Linux source code (540 MB)
  - Human genome (~3 GB)
  - Gutenberg project (~3.2 GB)

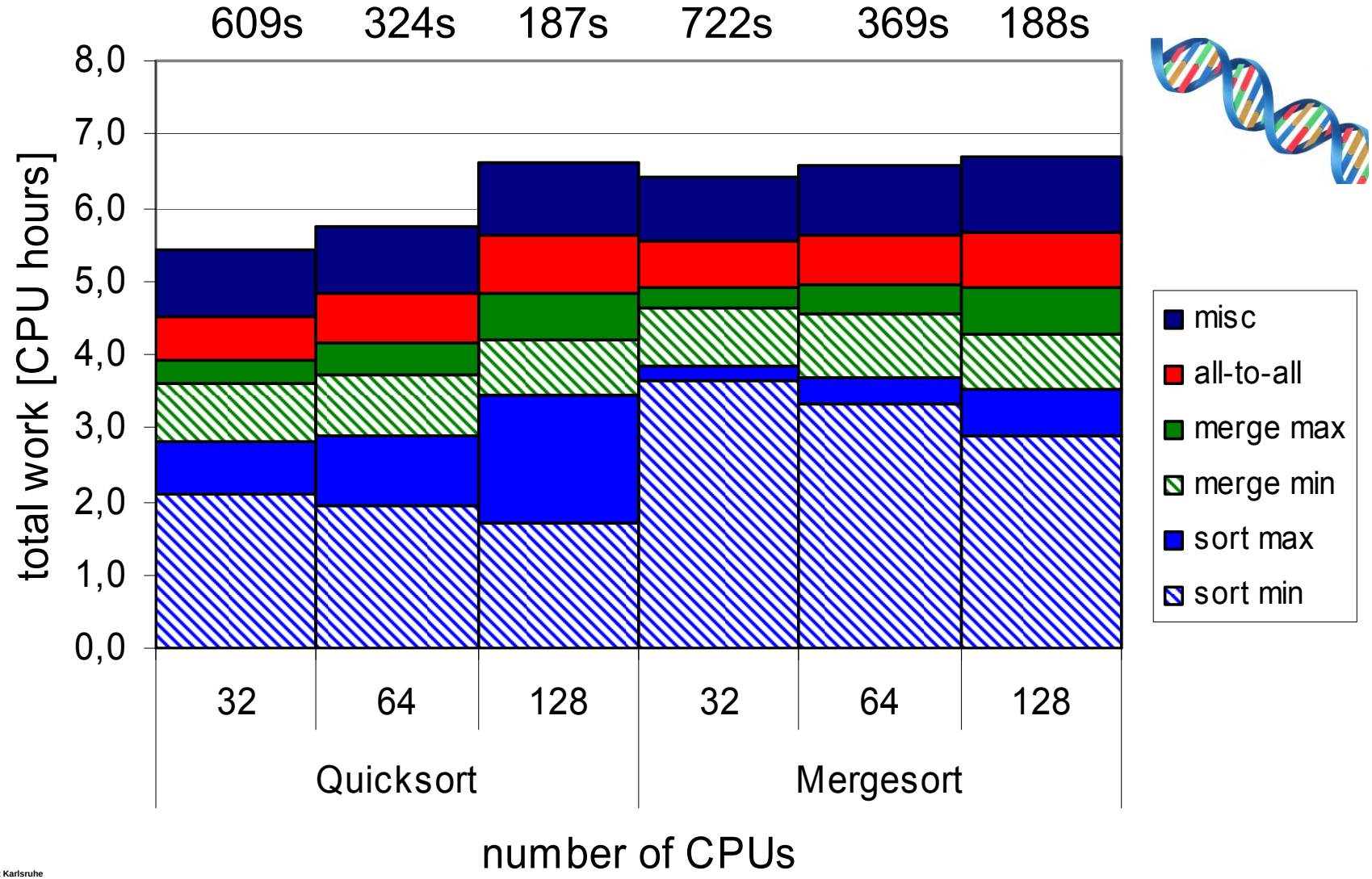
Linux source code (540 MB)



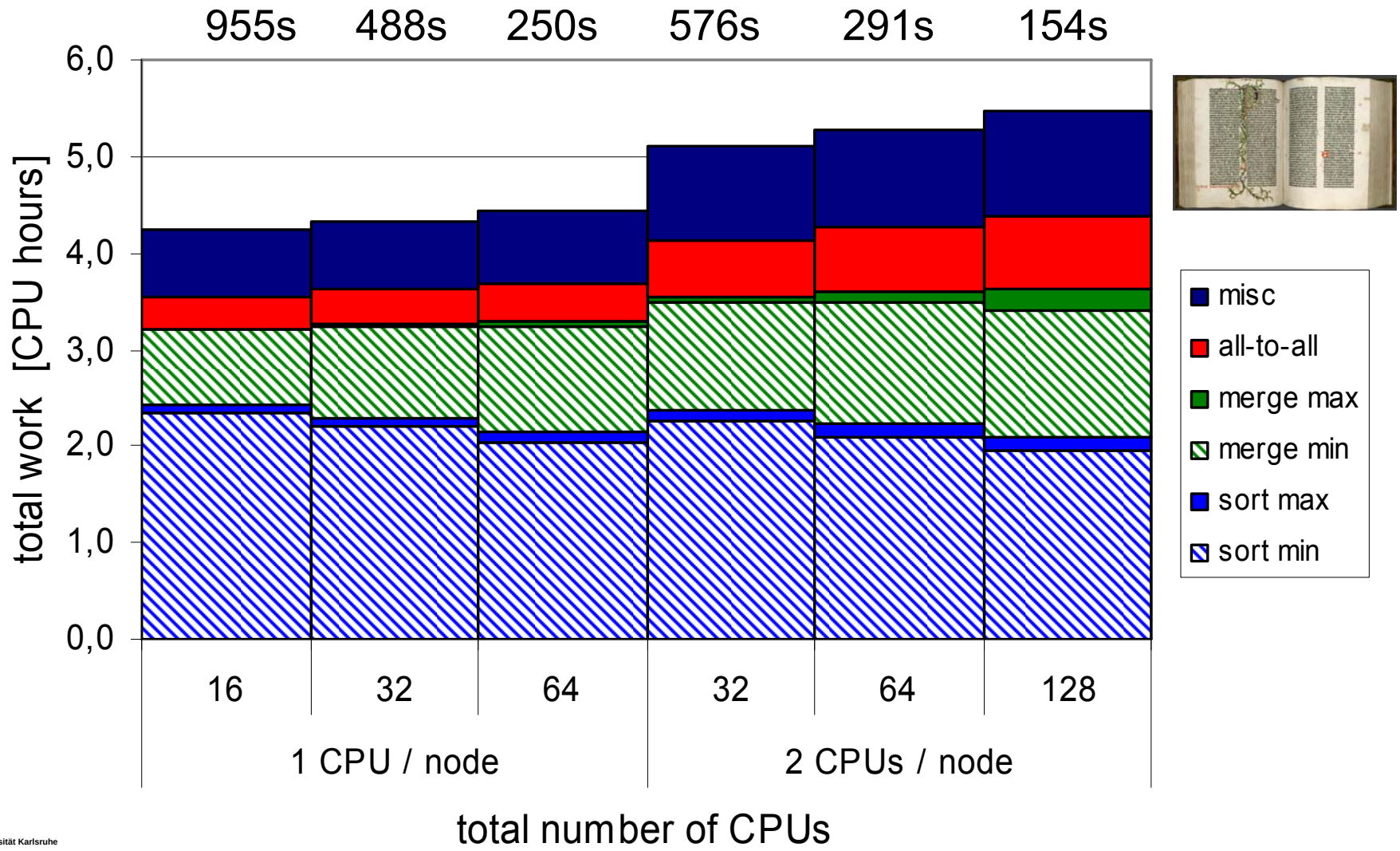
Human genome (~3 GB)



Human genome



Gutenberg project (~3.2 GB)

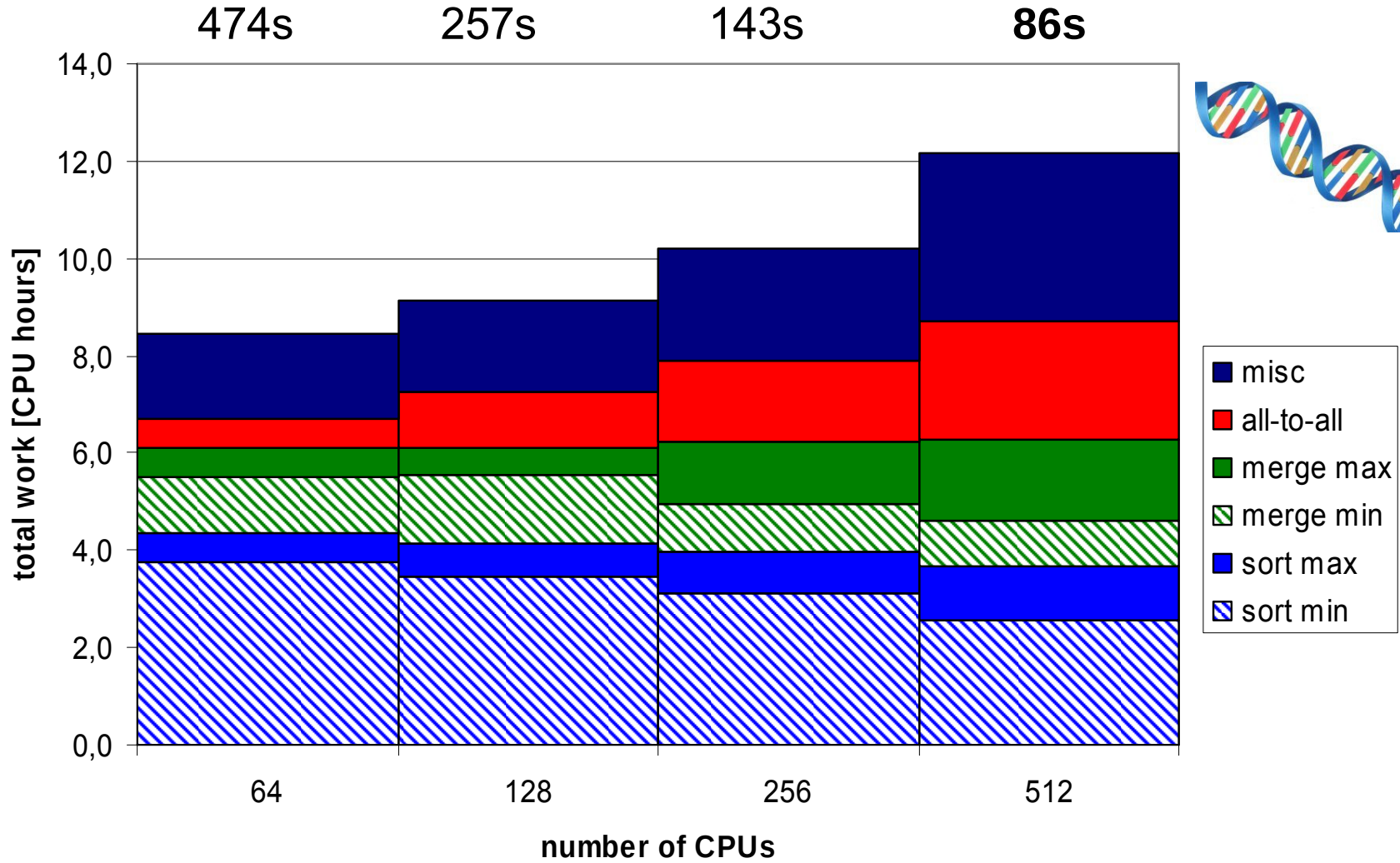


Jülich

- p690 IBM eServer Cluster1600 - FZ Jülich (JUMP-Cluster)
  (16 SMP nodes available;
  each node: 32x 1.7 GHz Power4+ and 128 GB main memory)

  – Human genome (~3 GB)

Human genome



Summary

- Suffix arrays:
  lexicographically sorted array of the suffixes of a string
- Sample sort
  Parallel sorting algorithm
- Overview pDC3
  Sorting tuples and recursion
- Evaluation:
  - Influence of different sequential sorting algorithms
  - Good scalability
  - **Suffix array construction of human genome: 1.5 minutes**

Thank you for your attention!
Questions?

Pseudo code of pDC3

Function $pDC3(T)$

$S := \langle((T[i, i+2]), i) : i \in [0, n), i \bmod 3 \neq 0\rangle$ 1

sort S *by the first component* 2

$P := name(S)$ 3

if *the names in* P *are not unique* **then** 4

permute the $(r, i) \in P$ *such that they are sorted by* $(i \bmod 3, i \operatorname{div} 3)$ 5

$SA^{12} := pDC3(\langle c : (c, i) \in P\rangle)$ 6

$P := \langle(j+1, SA^{12}[j]) : j \in [0, 2n/3)\rangle$ 7

permute P *such that it is sorted by the second component* 8

$S_0 := \langle(T[i], T[i+1], c', c'', i) : i \bmod 3 = 0, (c', i+1), (c'', i+2) \in P\rangle$ 9

$S_1 := \langle(c, T[i], c', i) : i \bmod 3 = 1, (c, i), (c', i+1) \in P\rangle$ 10

$S_2 := \langle(c, T[i], T[i+1], c'', i) : i \bmod 3 = 2, (c, i), (c'', i+2) \in P\rangle$ 11

$S :=$ *sort* $S_0 \cup S_1 \cup S_2$ *using comparison function:* 12

$(c, \ldots) \in S_1 \cup S_2 \leq (d, \ldots) \in S_1 \cup S_2 \Leftrightarrow c \leq d$

$(t, t', c', c'', i) \in S_0 \leq (u, u', d', d'', j) \in S_0 \Leftrightarrow (t, c') \leq (u, d')$

$(t, t', c', c'', i) \in S_0 \leq (d, u, \quad d', j) \in S_1 \Leftrightarrow (t, c') \leq (u, d')$

$(t, t', c', c'', i) \in S_0 \leq (d, u, u', d'', j) \in S_2 \Leftrightarrow (t, t', c'') \leq (u, u', d'')$

return $\langle$*last component of* $s : s \in S\rangle$ 13

Difference Cover

A difference cover D modulo v is a subset of $[0, v)$ such that for all $i \in [0, v)$ there exist $j, k \in D$ with $i \equiv k - j \pmod{v}$.

Example:

{1,2} is a difference cover modulo 3.

{1,2,4} is a difference cover modulo 7.