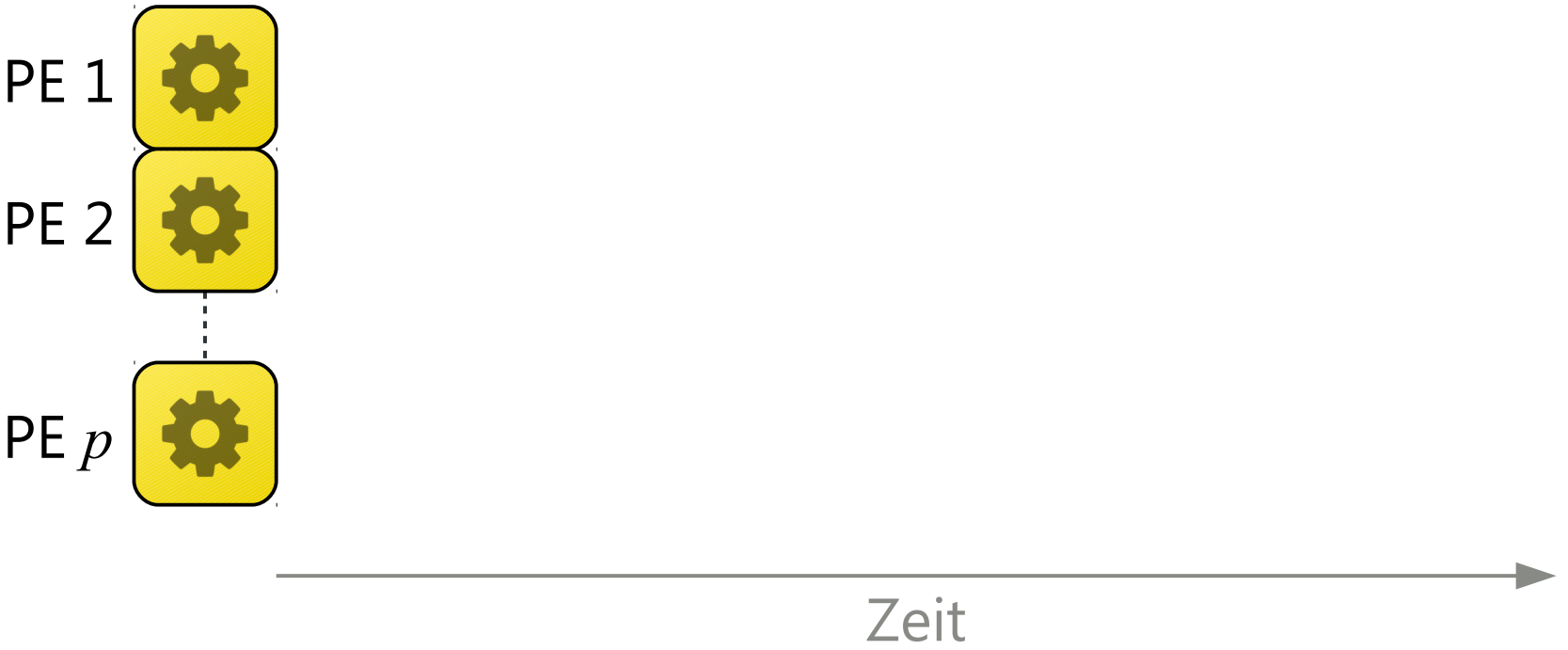
Master-Abschlussvortrag

Distributed Wavelet Tree Construction

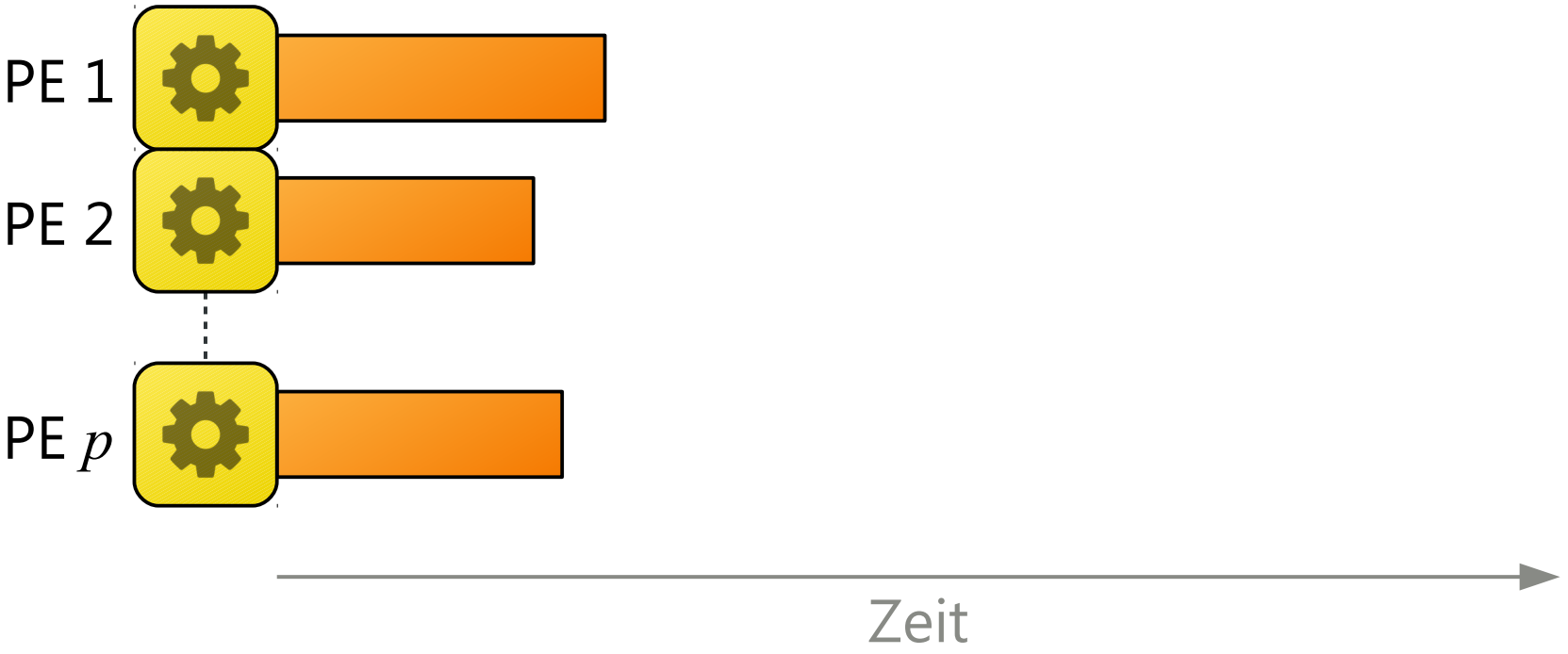
Patrick Dinklage

Gutachter:
Prof. Dr. Johannes Fischer
M. Sc. Florian Kurpicz

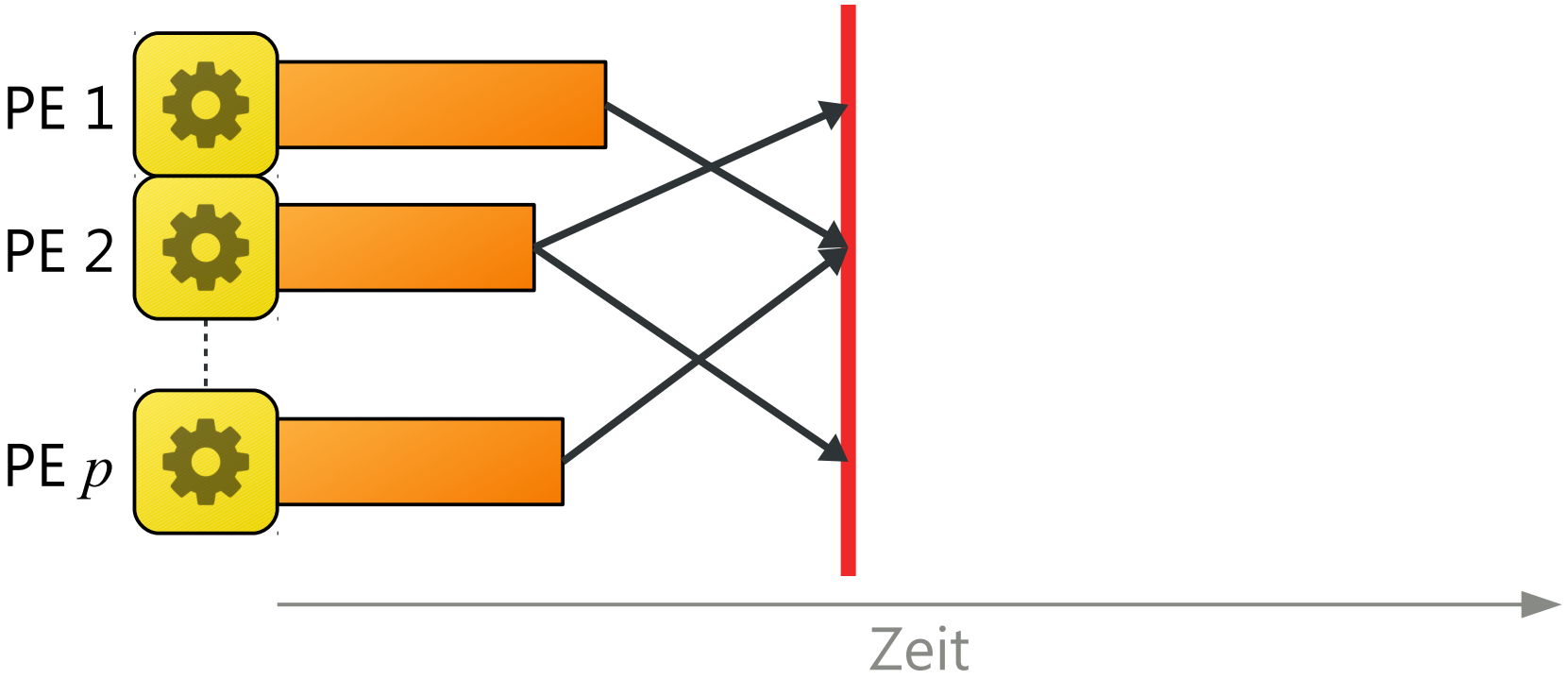
Das BSP-Modell



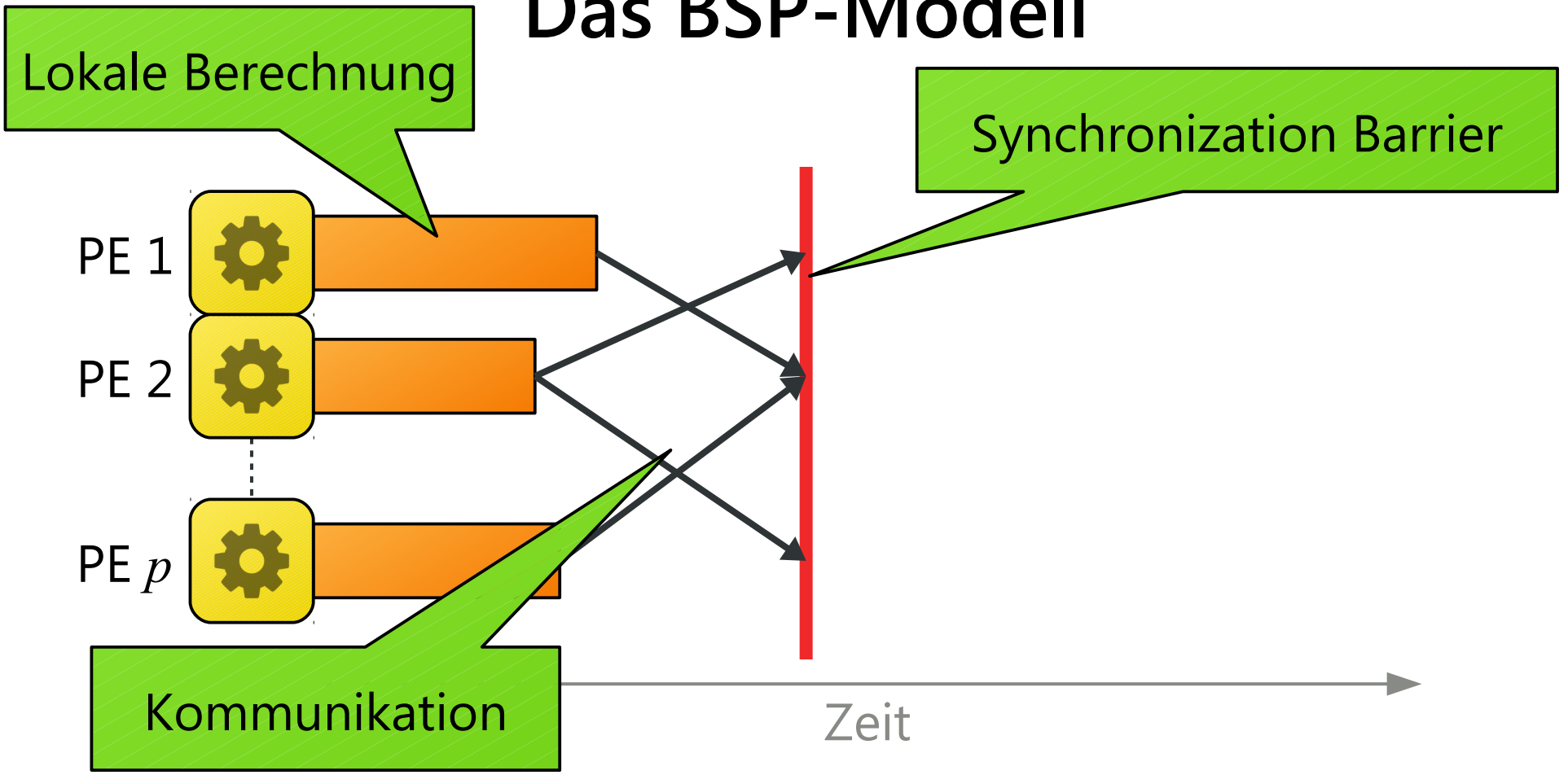
Das BSP-Modell



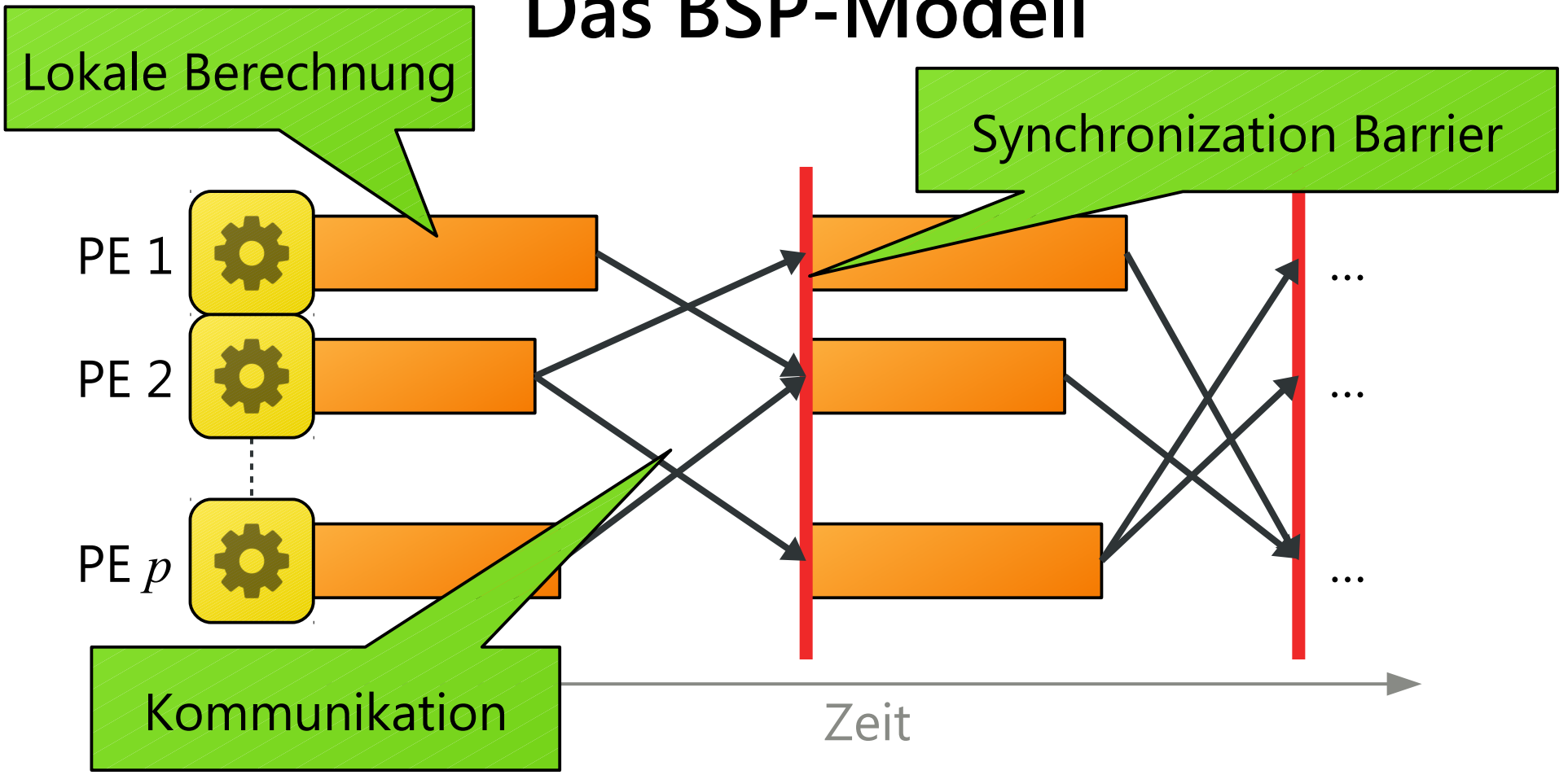
Das BSP-Modell



Das BSP-Modell



Das BSP-Modell



BSP-Kostenanalyse

Anzahl lokaler Berechnungsschritte

Anzahl versendeter Computerwörter

Anzahl Synchronization Barrier

Wavelet Trees

wavelettree

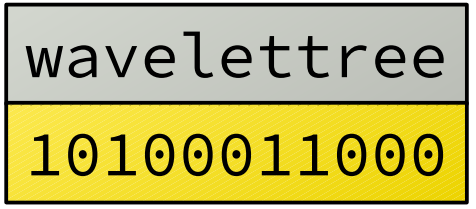
Wavelet Trees

$\Sigma = \{$a, e, l, r, ␣, t, v, w$\}$

wavelettree

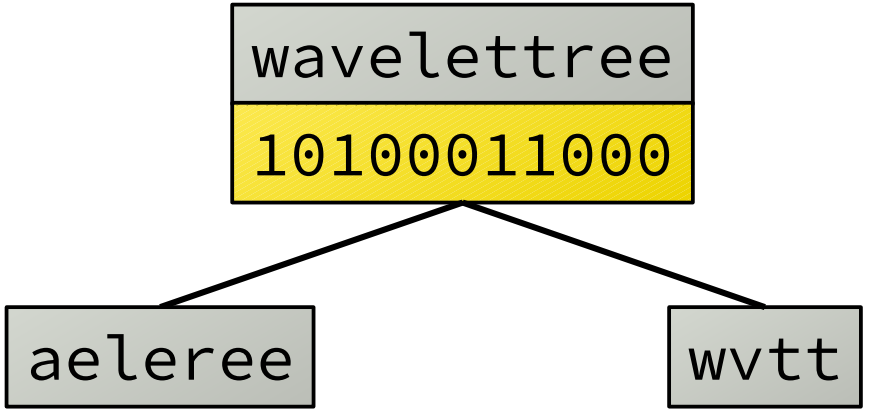
Wavelet Trees

$\Sigma = \{a, e, l, r, \ , t, v, w\}$

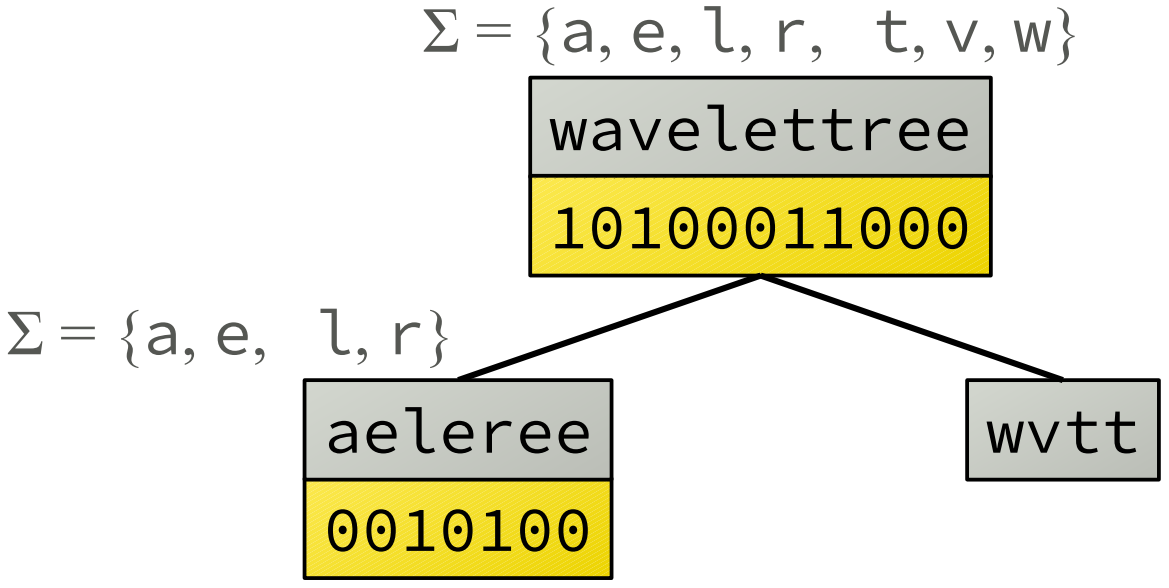


Wavelet Trees

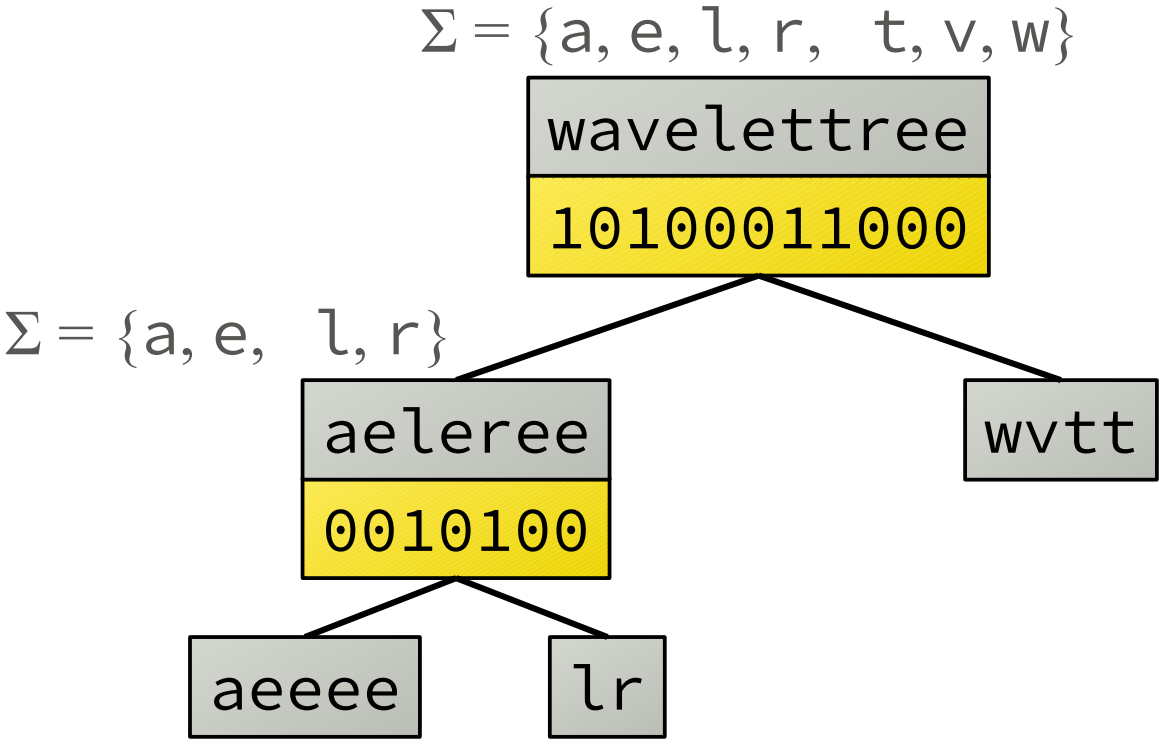
$\Sigma = \{a, e, l, r, \ t, v, w\}$



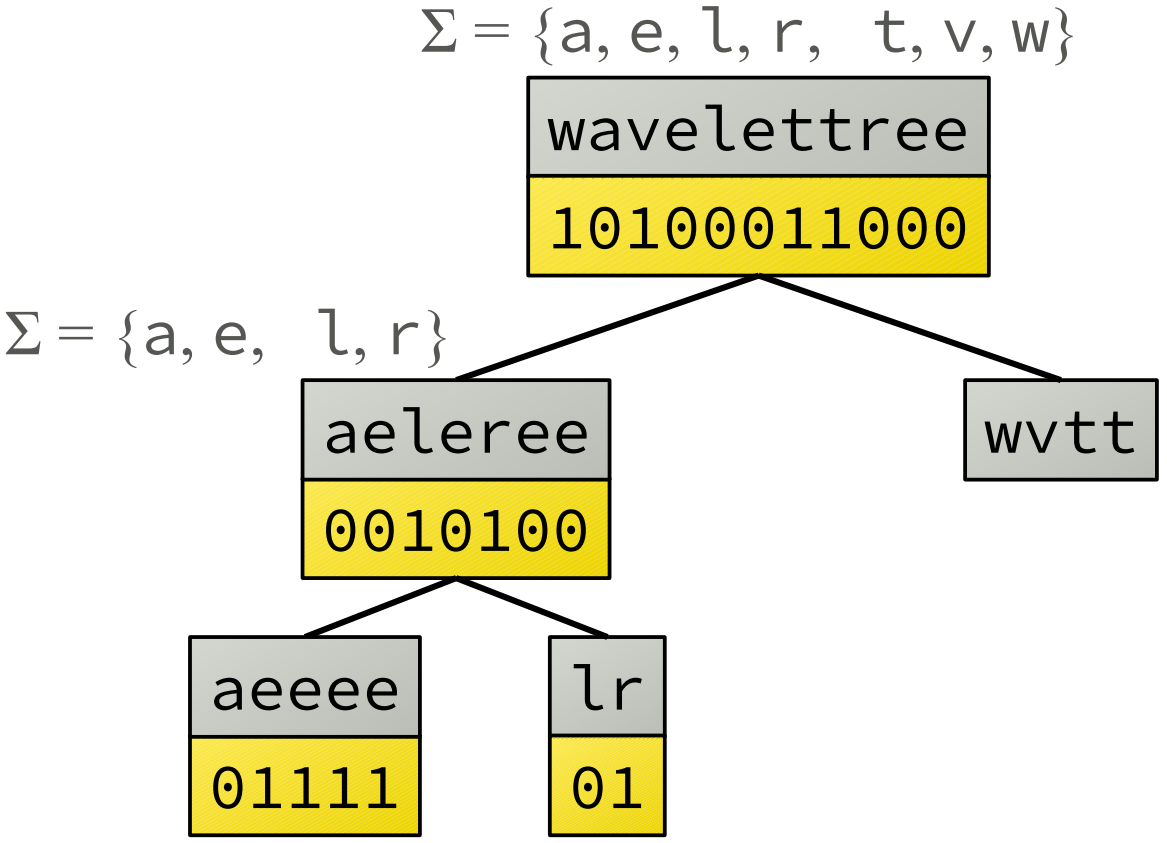
Wavelet Trees



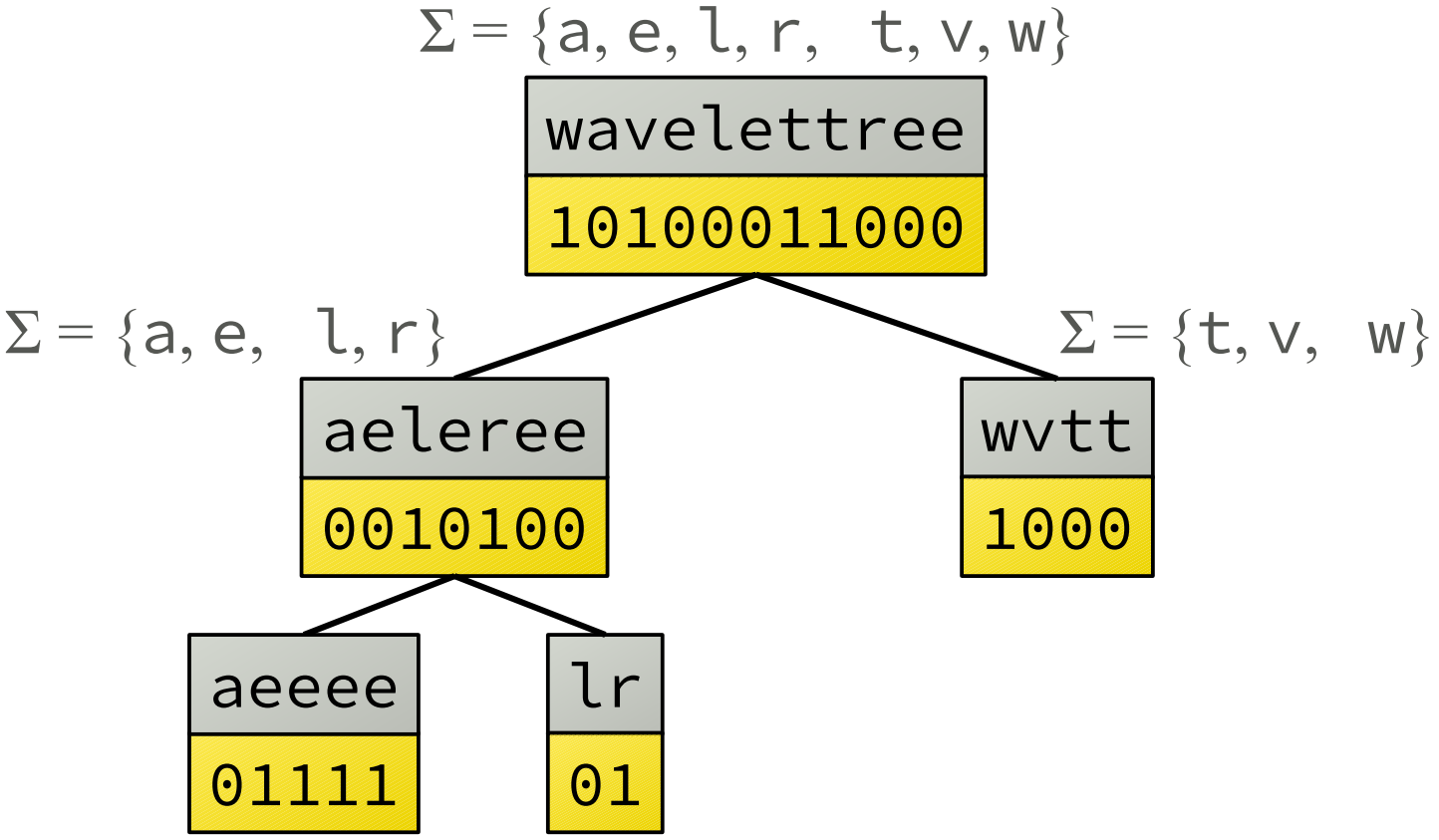
Wavelet Trees



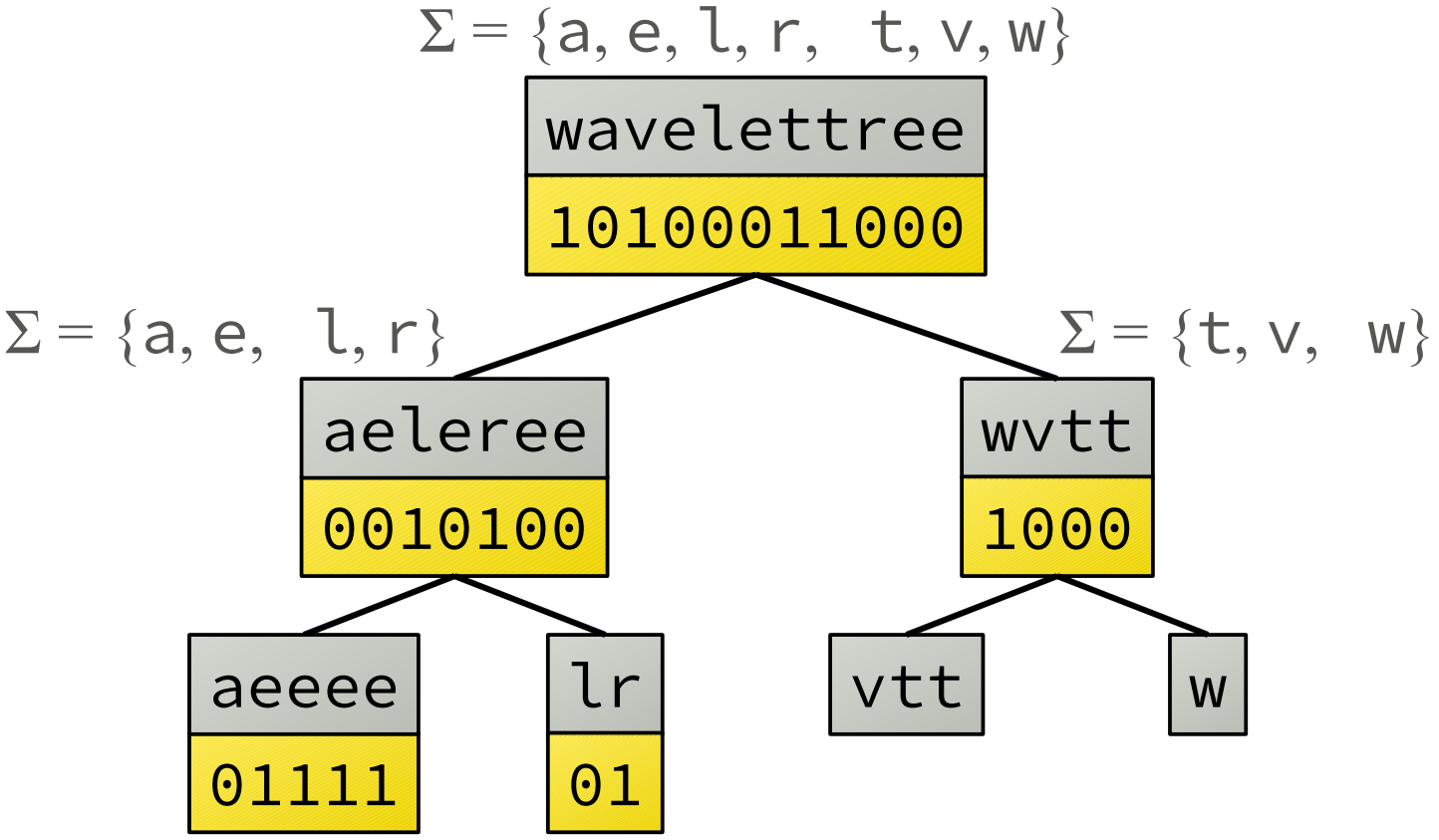
Wavelet Trees



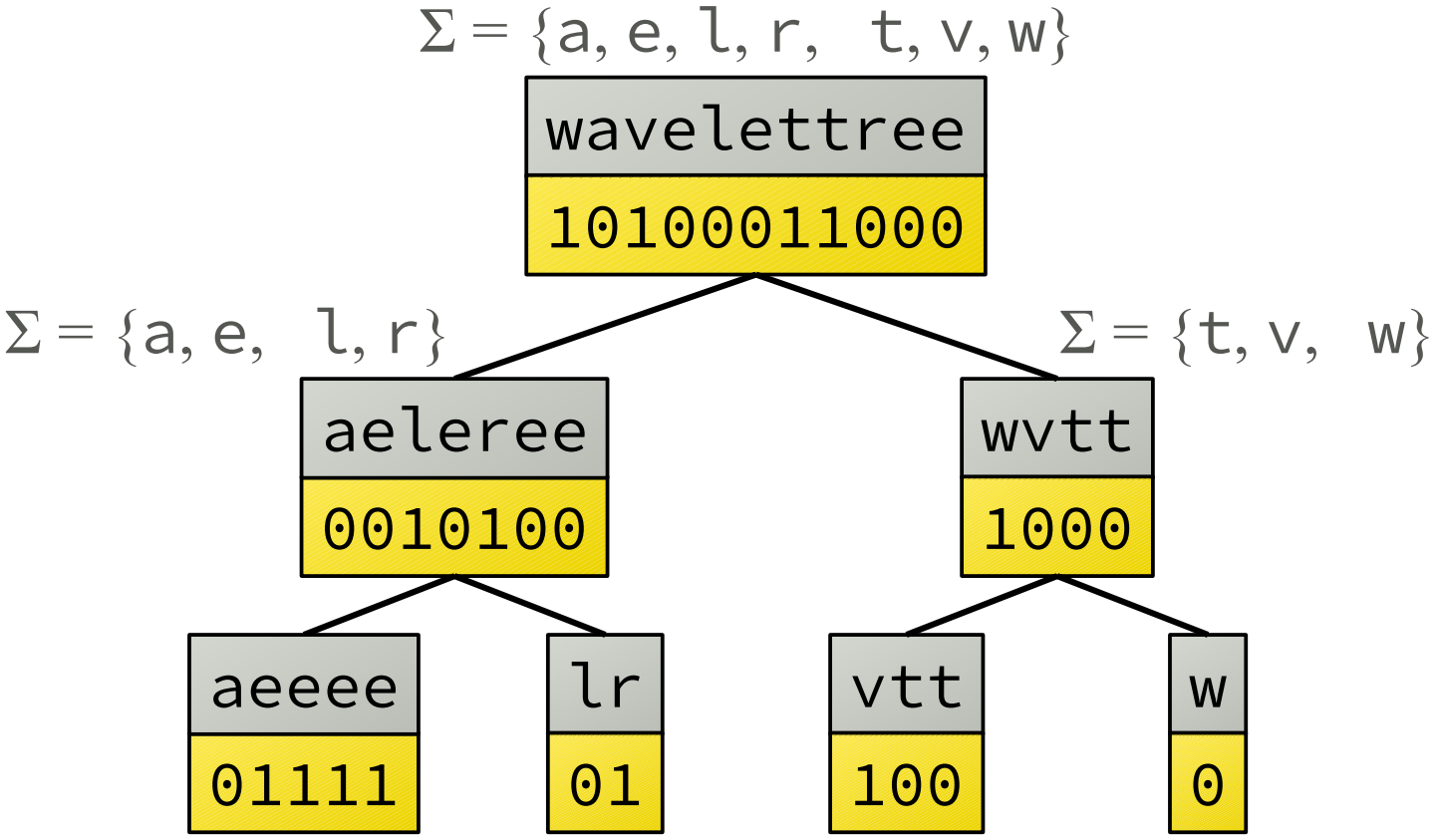
Wavelet Trees



Wavelet Trees

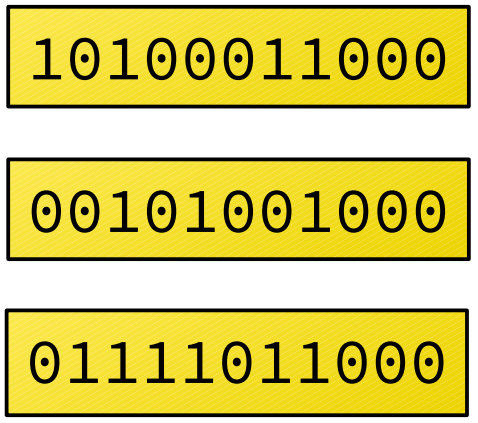


Wavelet Trees



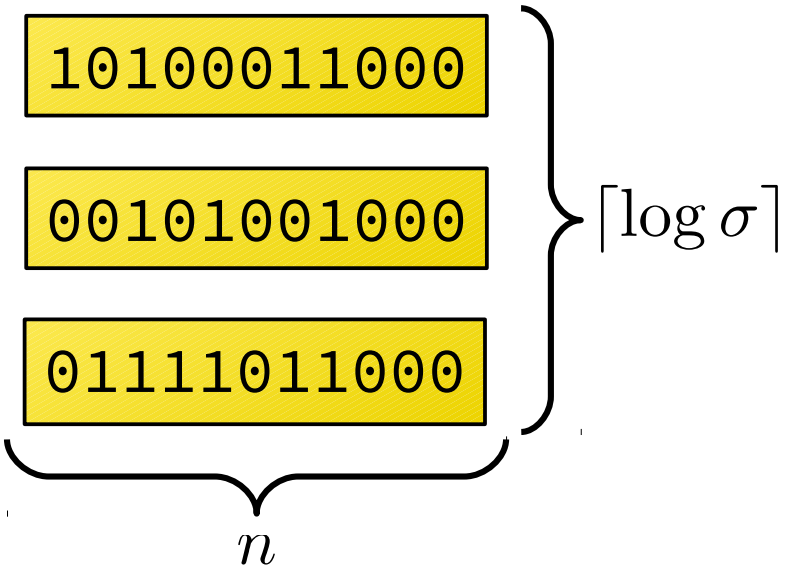
Wavelet Trees

Levelweise Darstellung



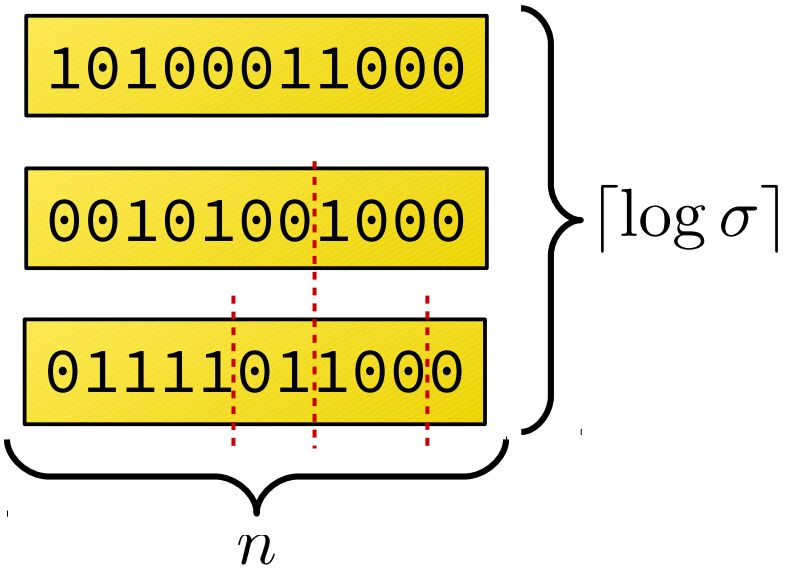
Wavelet Trees

Levelweise Darstellung



Wavelet Trees

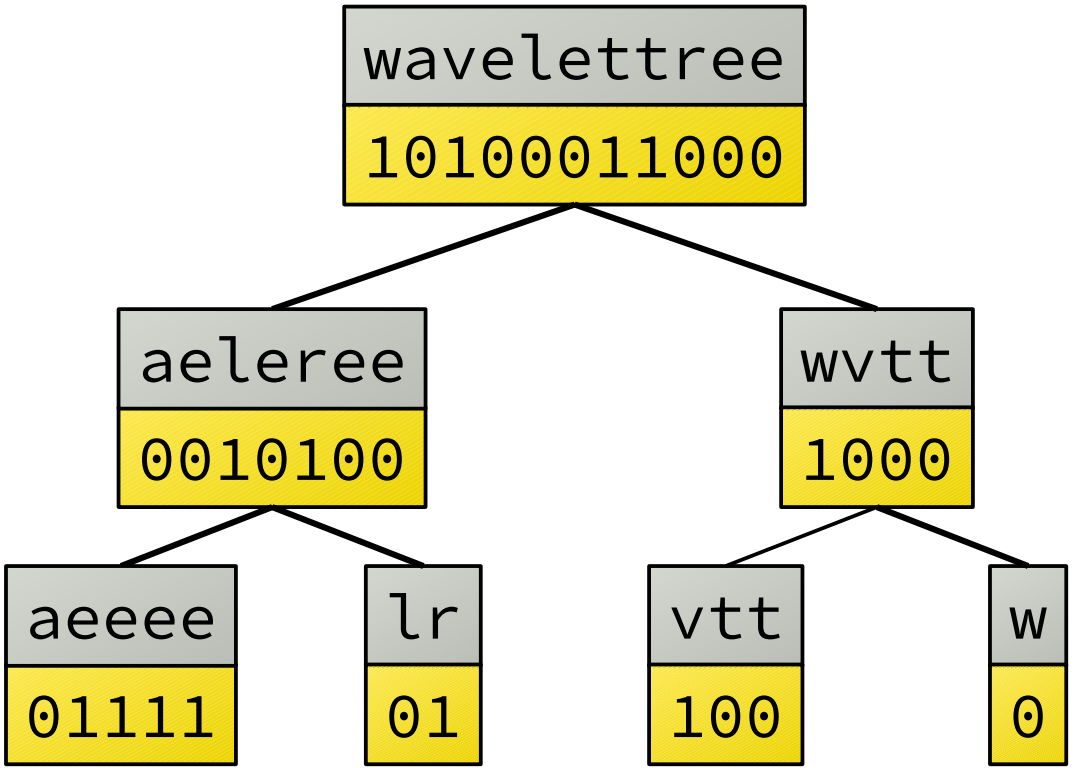
Levelweise Darstellung



+ rank/select je Level

Wavelet Trees

Bit-Pfade



Wavelet Trees

Effektives Alphabet:

a ↦ 0 = 000_b

e ↦ 1 = 001_b

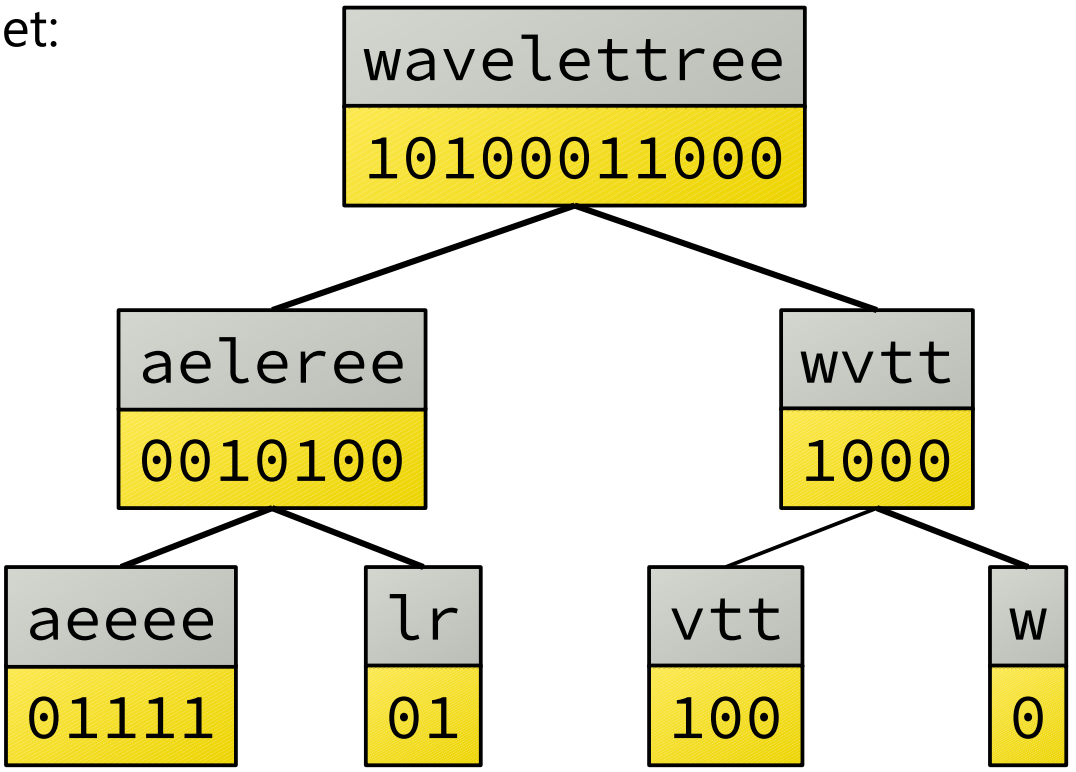
l ↦ 2 = 010_b

r ↦ 3 = 011_b

t ↦ 4 = 100_b

v ↦ 5 = 101_b

w ↦ 6 = 110_b



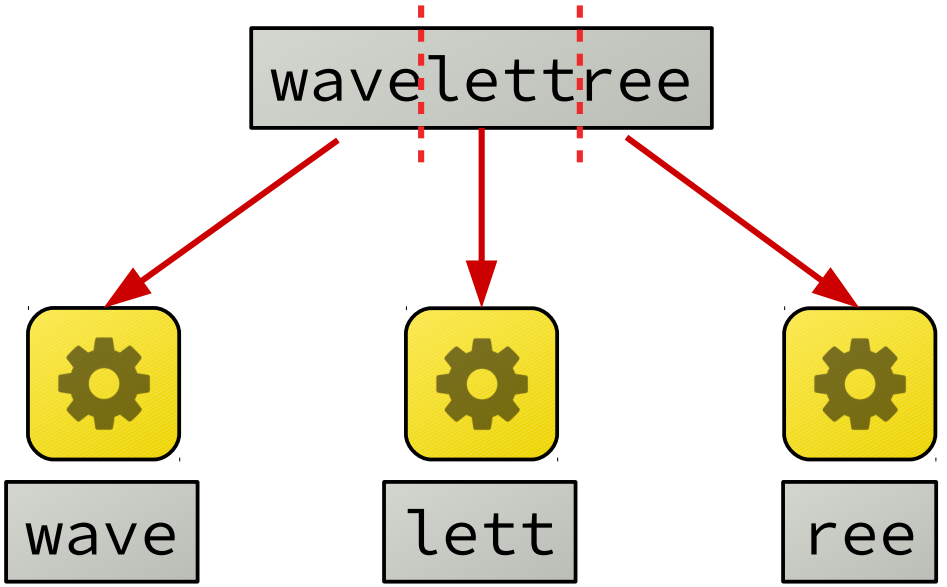
Verteile Wavelet-Tree-Konstruktion

Partitionierung der Eingabe

wavelettree

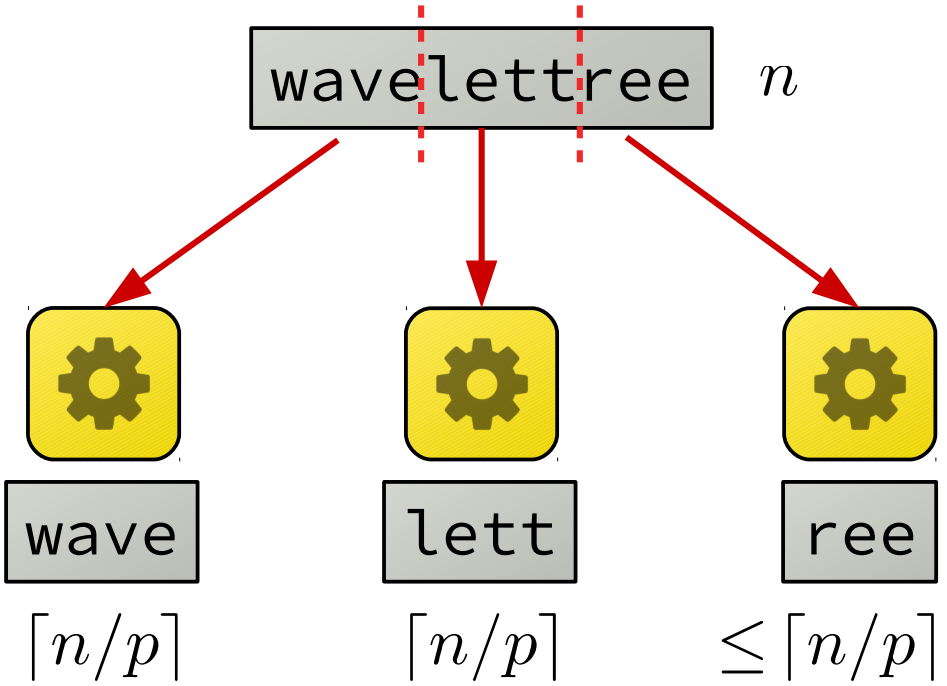
Verteile Wavelet-Tree-Konstruktion

Partitionierung der Eingabe



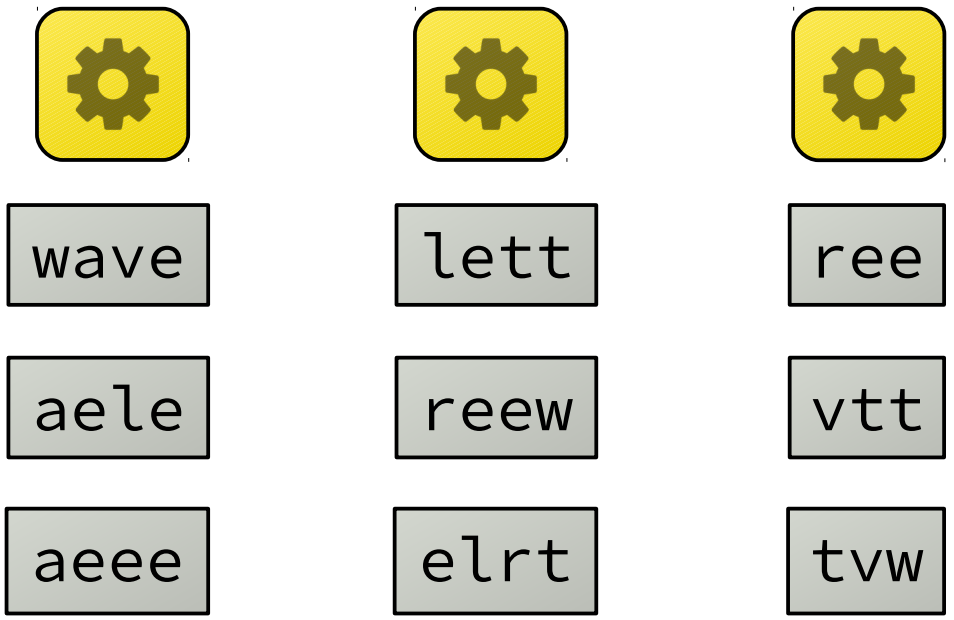
Verteile Wavelet-Tree-Konstruktion

Partitionierung der Eingabe



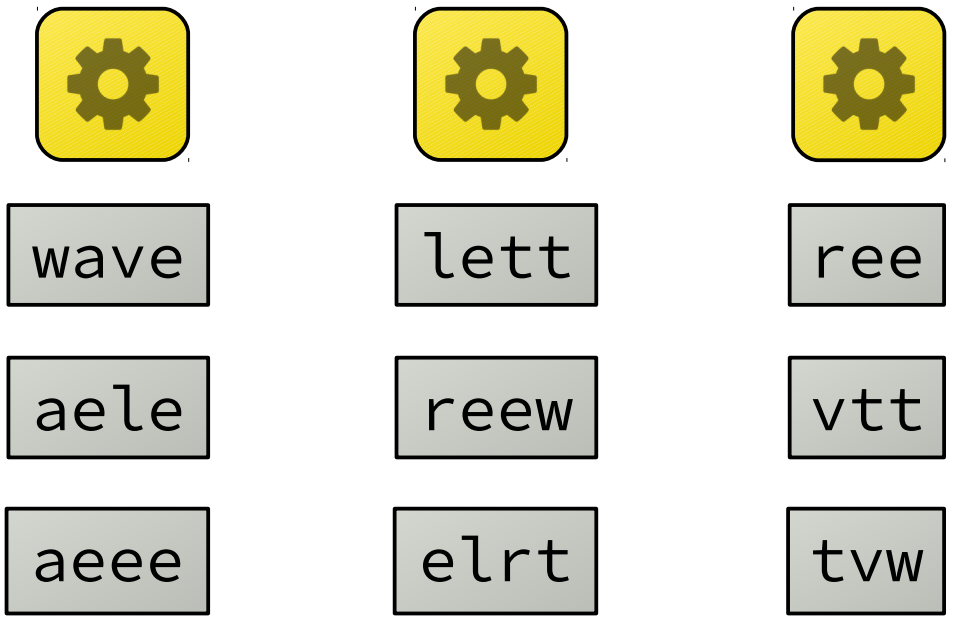
Verteile Wavelet-Tree-Konstruktion

Ausgabeformat



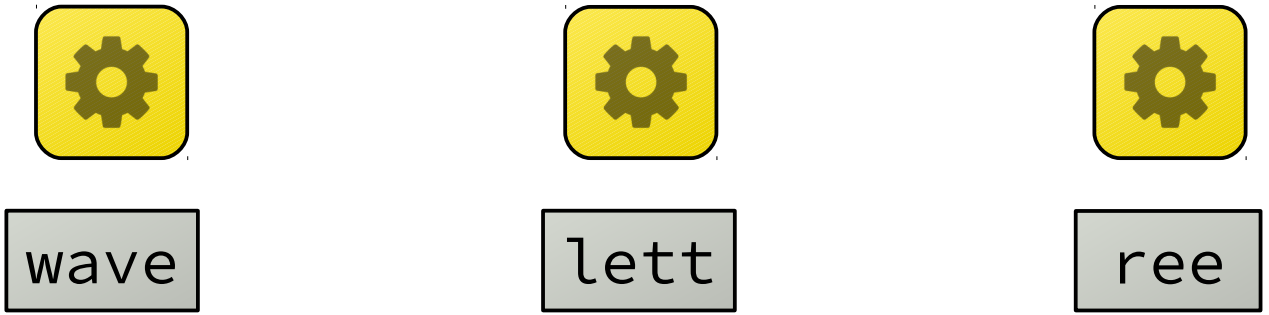
Verteile Wavelet-Tree-Konstruktion

Ausgabeformat

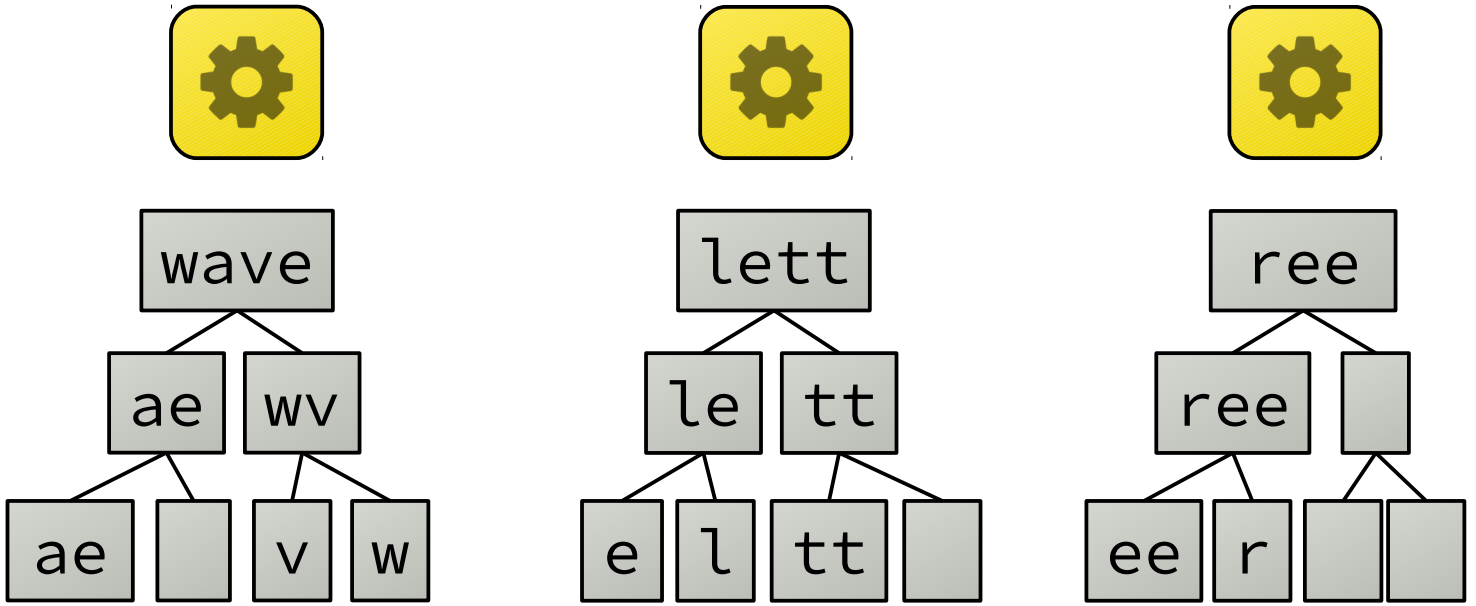


→ Partitionierte levelweise Darstellung

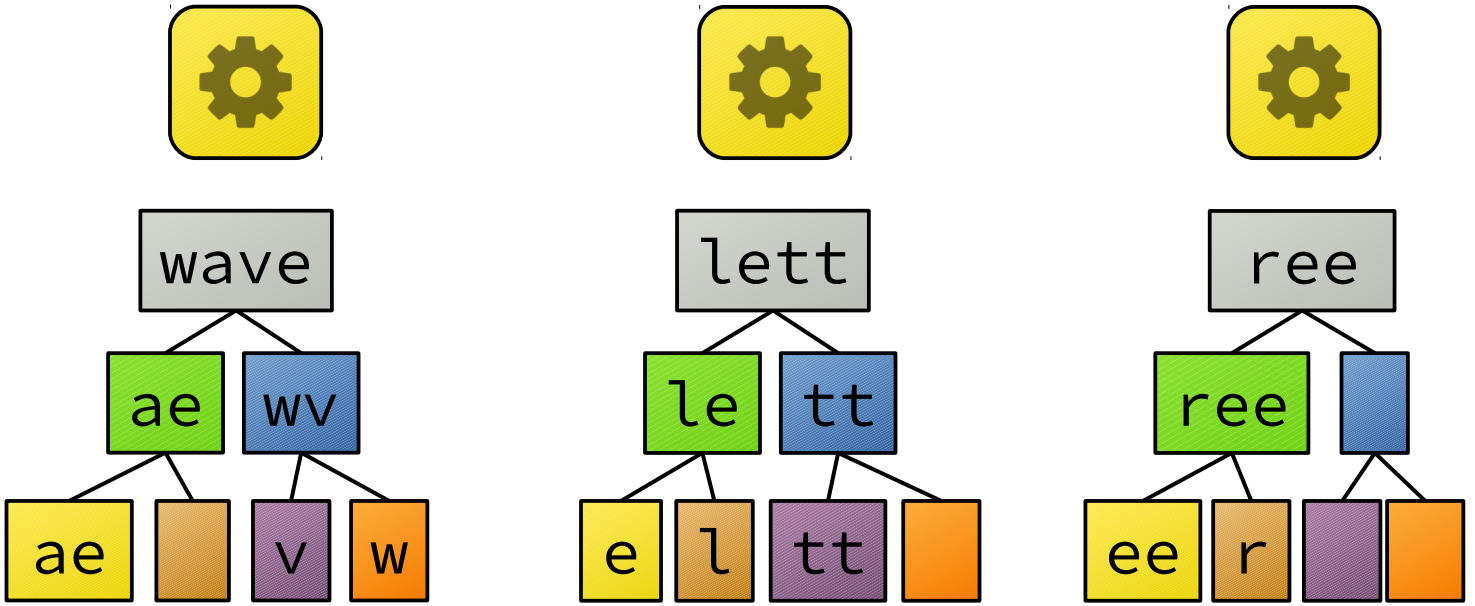
Domain Decomposition



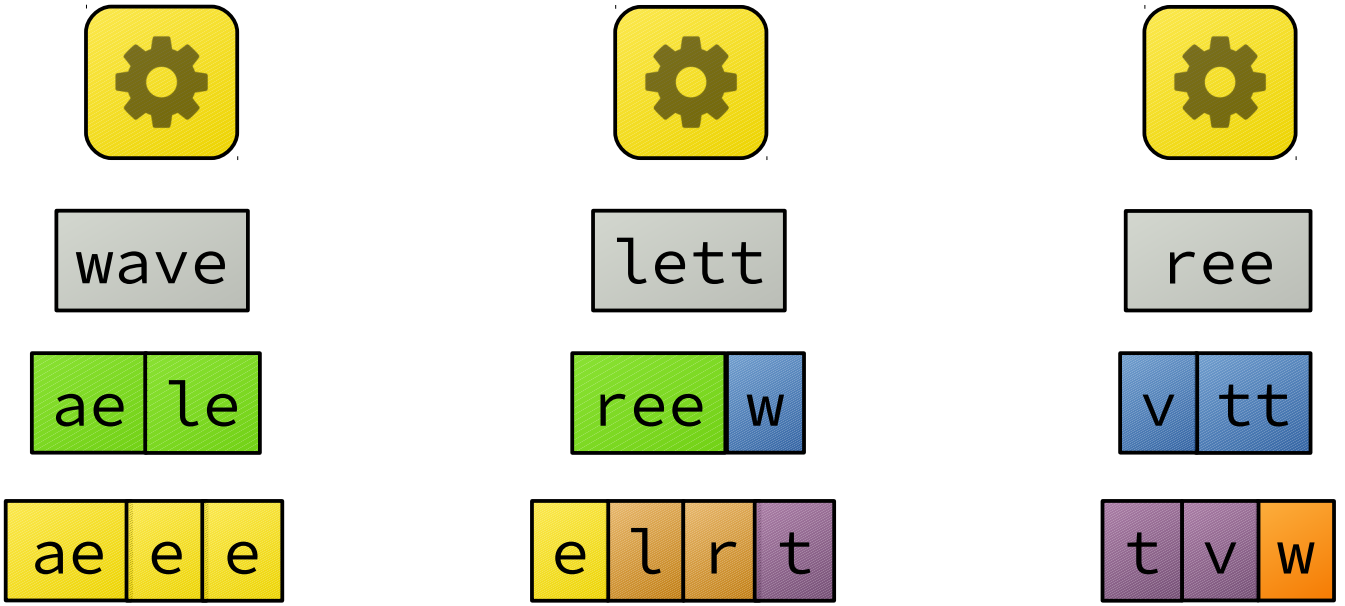
Domain Decomposition



Domain Decomposition



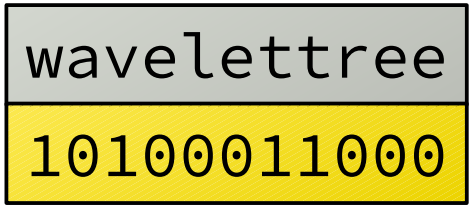
Domain Decomposition



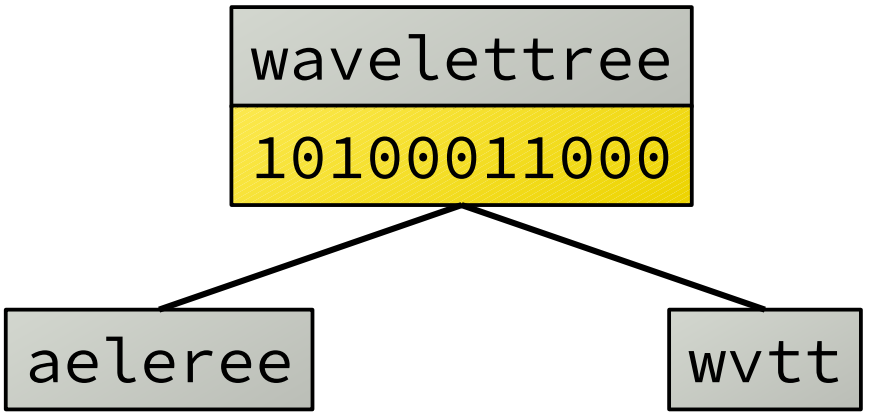
Verteiltes Aufsplitten

wavelettree

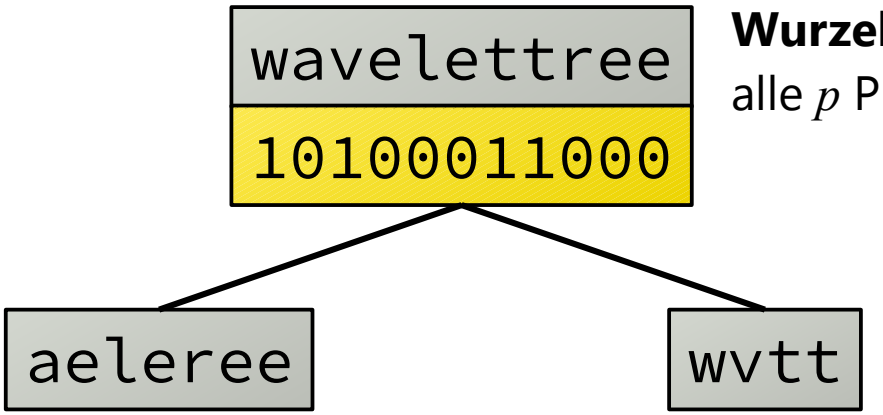
Verteiltes Aufsplitten



Verteiltes Aufsplitten

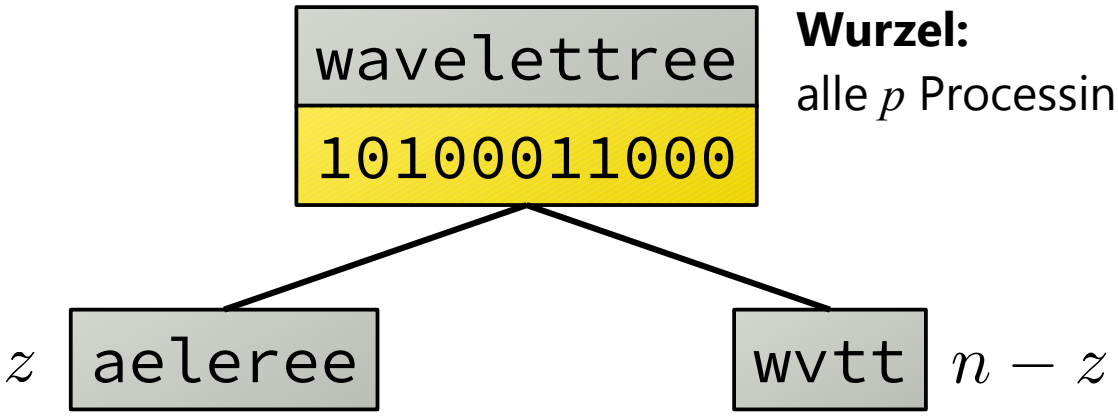


Verteiltes Aufsplitten

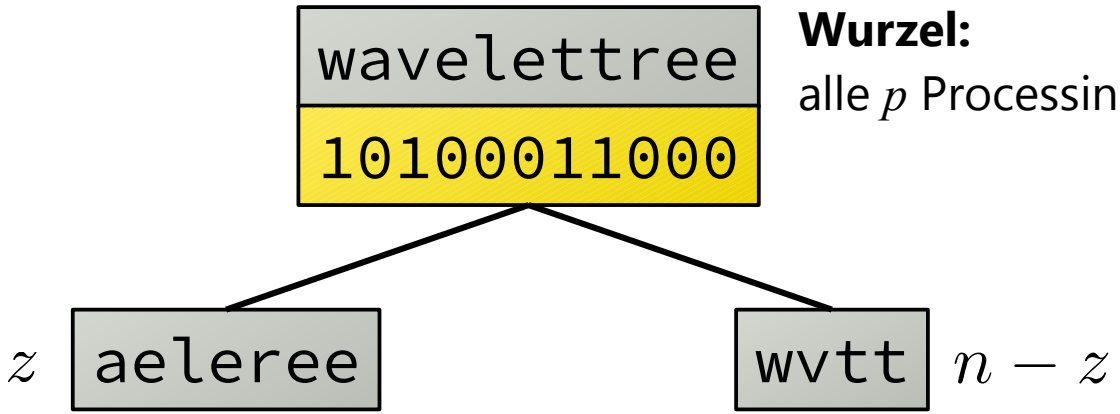


Wurzel:
alle p Processing-Elemente

Verteiltes Aufsplitten



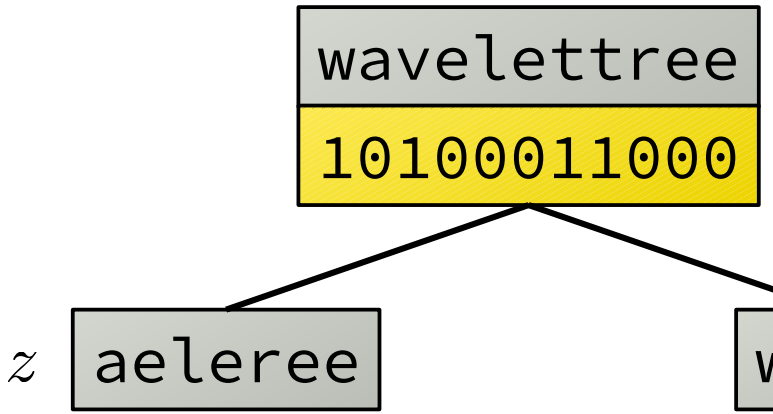
Verteiltes Aufsplitten



→ $p\frac{z}{n}$ Processing-Elemente

→ $p\frac{n-z}{n}$ Processing-Elemente

Verteiltes Aufsplitten



Wurzel:
alle p Processing-Elemente

→ $p\frac{z}{n}$ Processing-Elemente

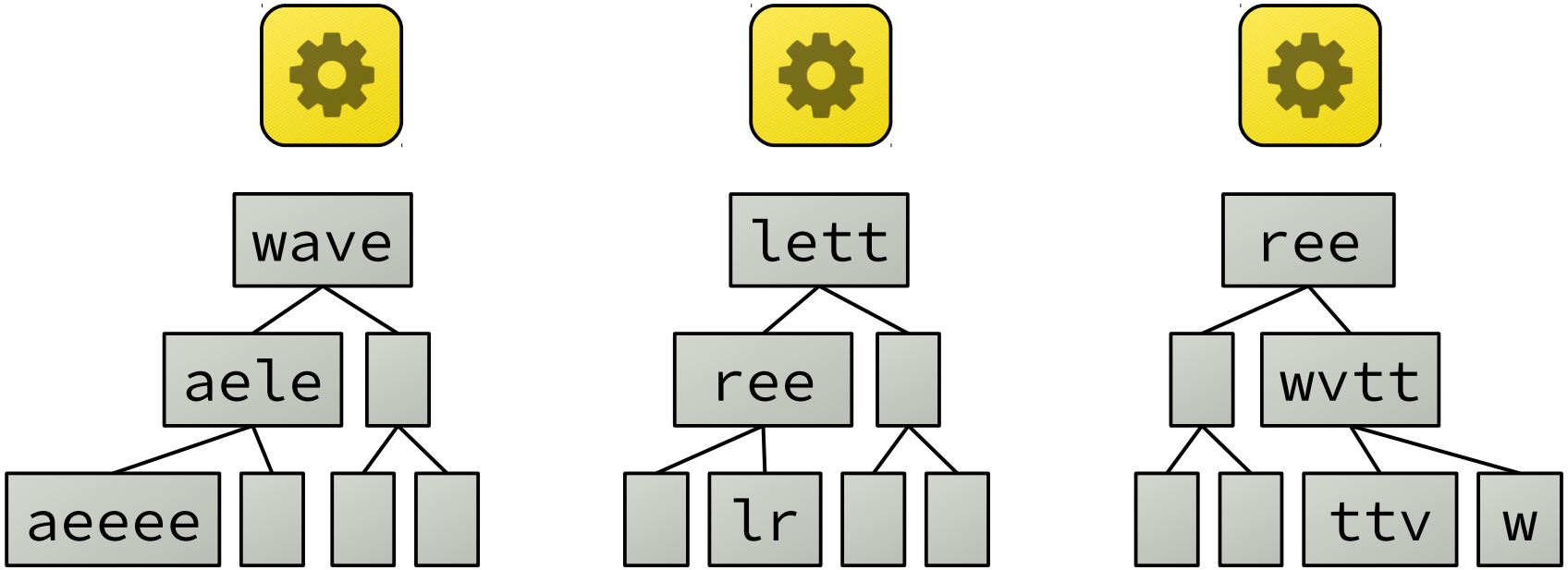
$z = 7 \Rightarrow \frac{z}{n} \approx \frac{2}{3}$ → 2 PEs

→ $p\frac{n-z}{n}$ Processing-Elemente

$\frac{n-z}{n} \approx \frac{1}{3}$ → 1 PE

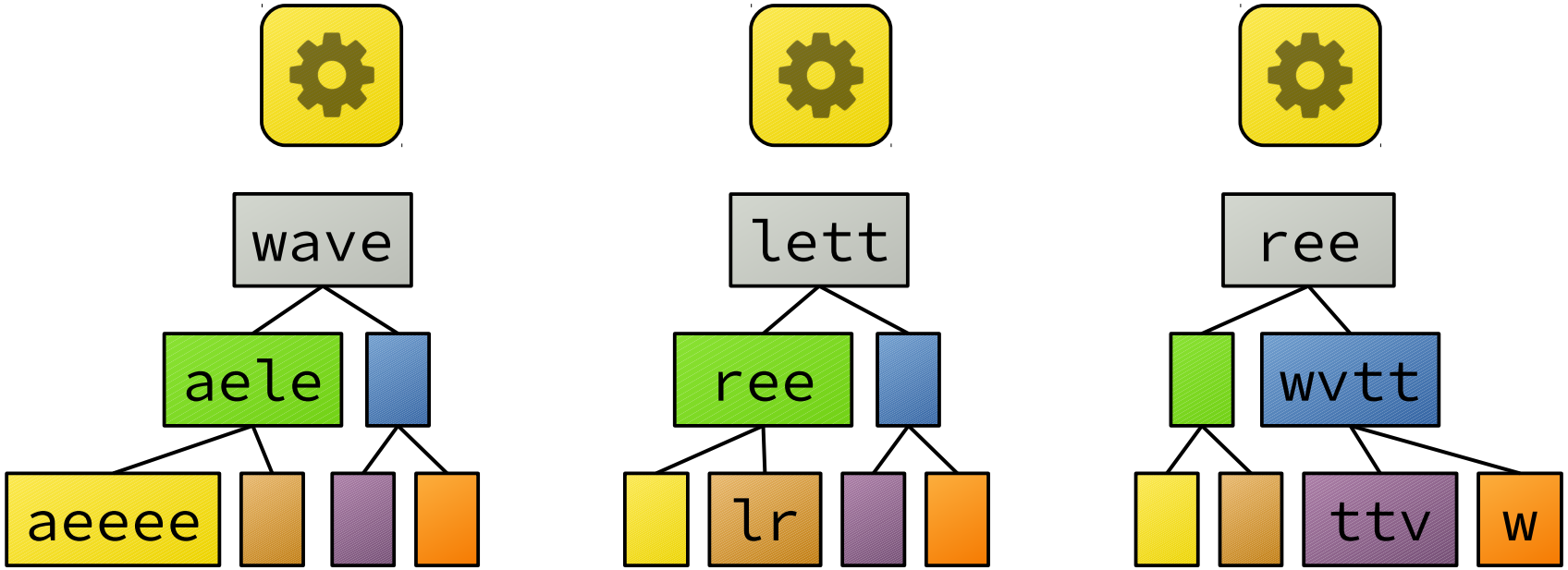
Verteiltes Aufsplitten

Merge



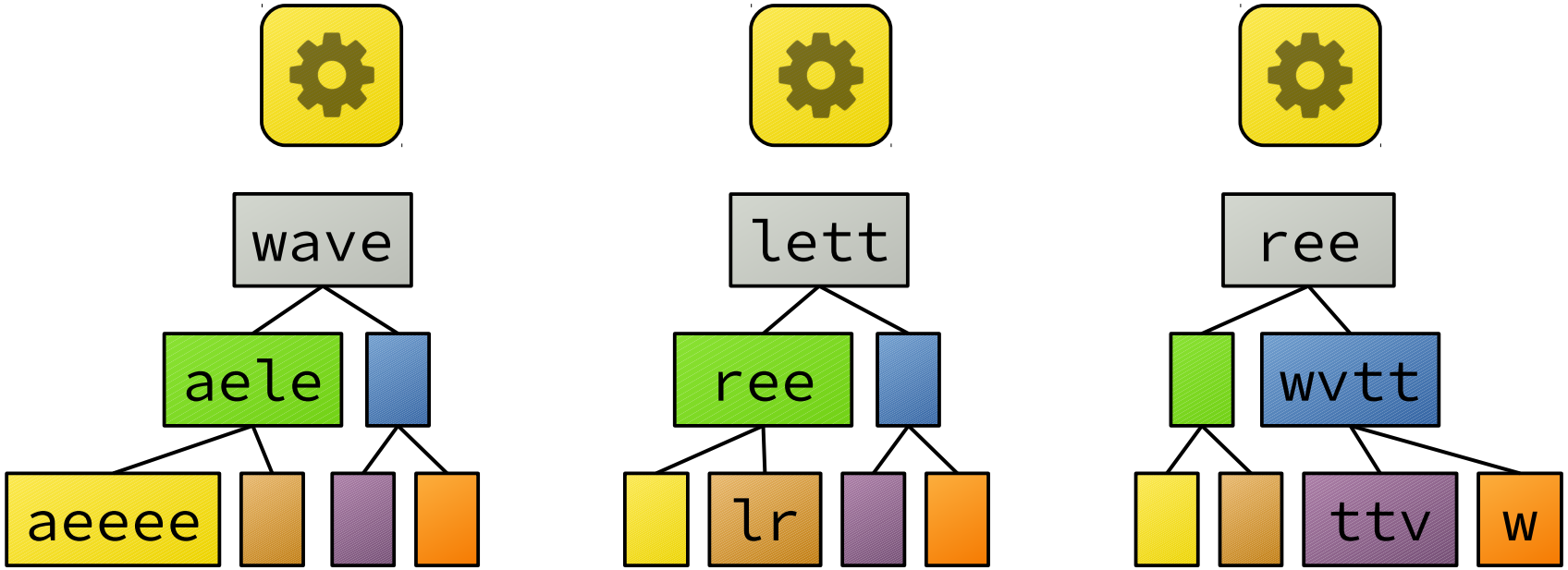
Verteiltes Aufsplitten

Merge



Verteiltes Aufsplitten

Merge



→ Variante des Szenarios bei der Domain Decomposition!

Stabiles Sortieren

wavelettree

Stabiles Sortieren

`wavelettree`

Effektives Alphabet:

a ↦ 0 = 000_b

e ↦ 1 = 001_b

l ↦ 2 = 010_b

r ↦ 3 = 011_b

t ↦ 4 = 100_b

v ↦ 5 = 101_b

w ↦ 6 = 110_b

Stabiles Sortieren

w	a	v	e	l	e	t	t	r	e	e
110	000	101	001	010	001	100	100	011	001	001

Effektives Alphabet:

a ↦ 0 = 000_b

e ↦ 1 = 001_b

l ↦ 2 = 010_b

r ↦ 3 = 011_b

t ↦ 4 = 100_b

v ↦ 5 = 101_b

w ↦ 6 = 110_b

Stabiles Sortieren

w	a	v	e	l	e	t	t	r	e	e
110	**0**00	**1**01	**0**01	**0**10	**0**01	**1**00	**1**00	**0**11	**0**01	**0**01

Effektives Alphabet:

a ↦ 0 = 000_b

e ↦ 1 = 001_b

l ↦ 2 = 010_b

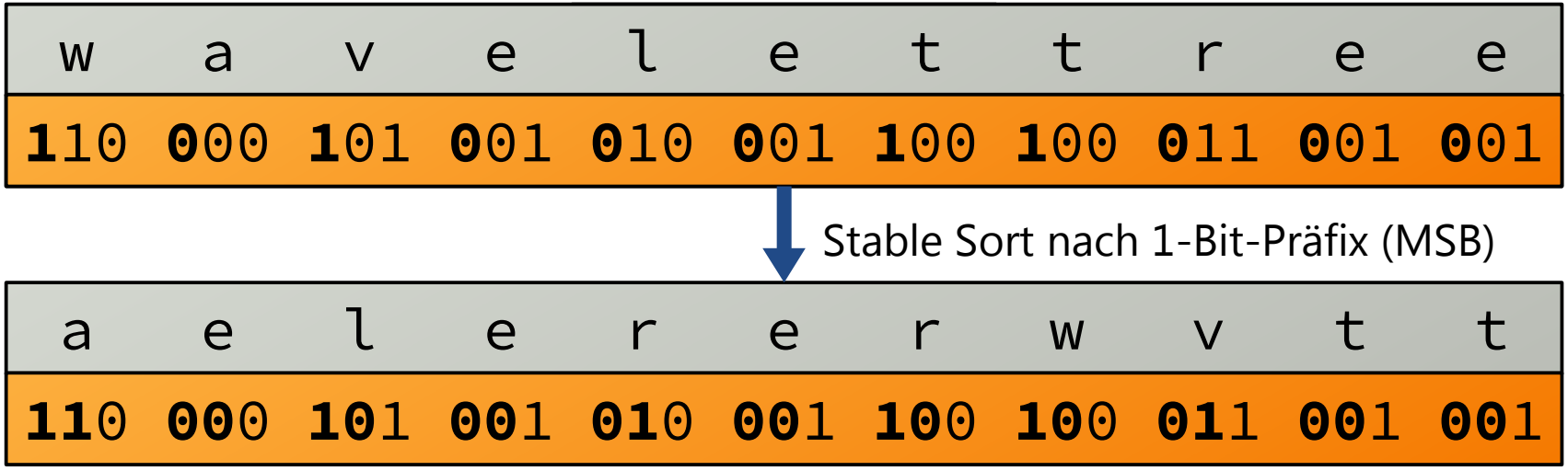
r ↦ 3 = 011_b

t ↦ 4 = 100_b

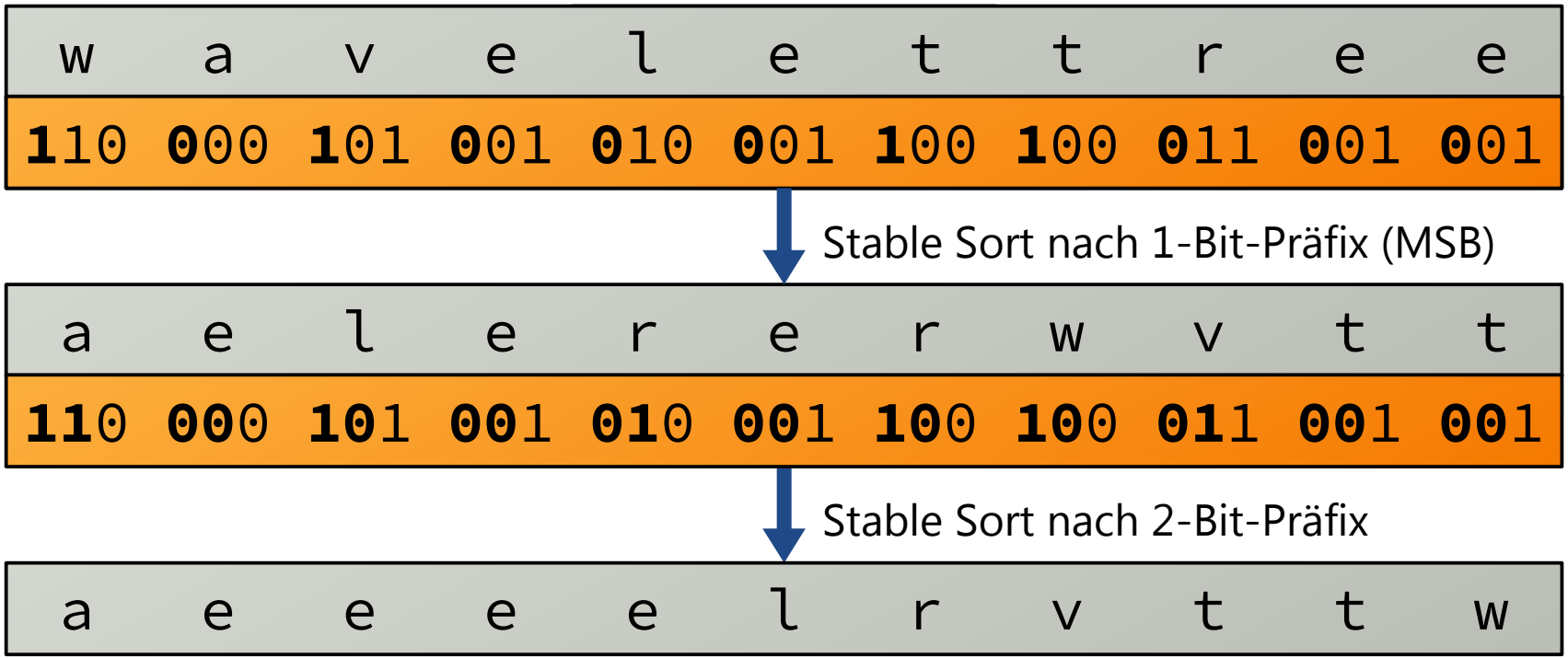
v ↦ 5 = 101_b

w ↦ 6 = 110_b

Stabiles Sortieren

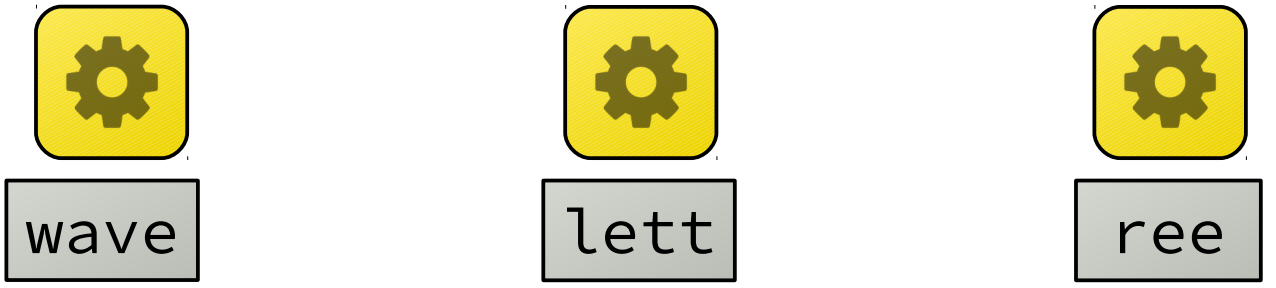


Stabiles Sortieren



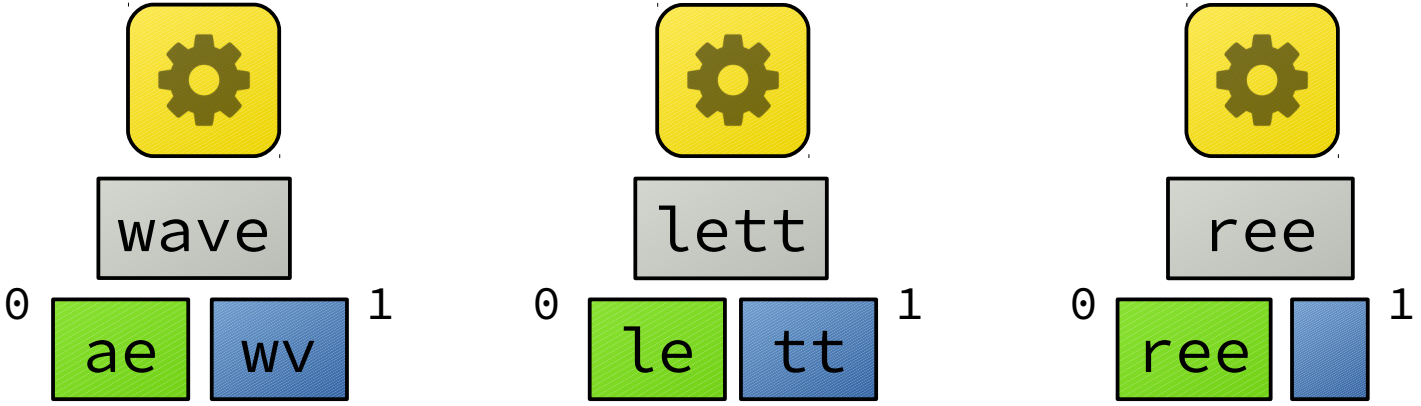
Stabiles Sortieren

Verteilter Bucket Sort



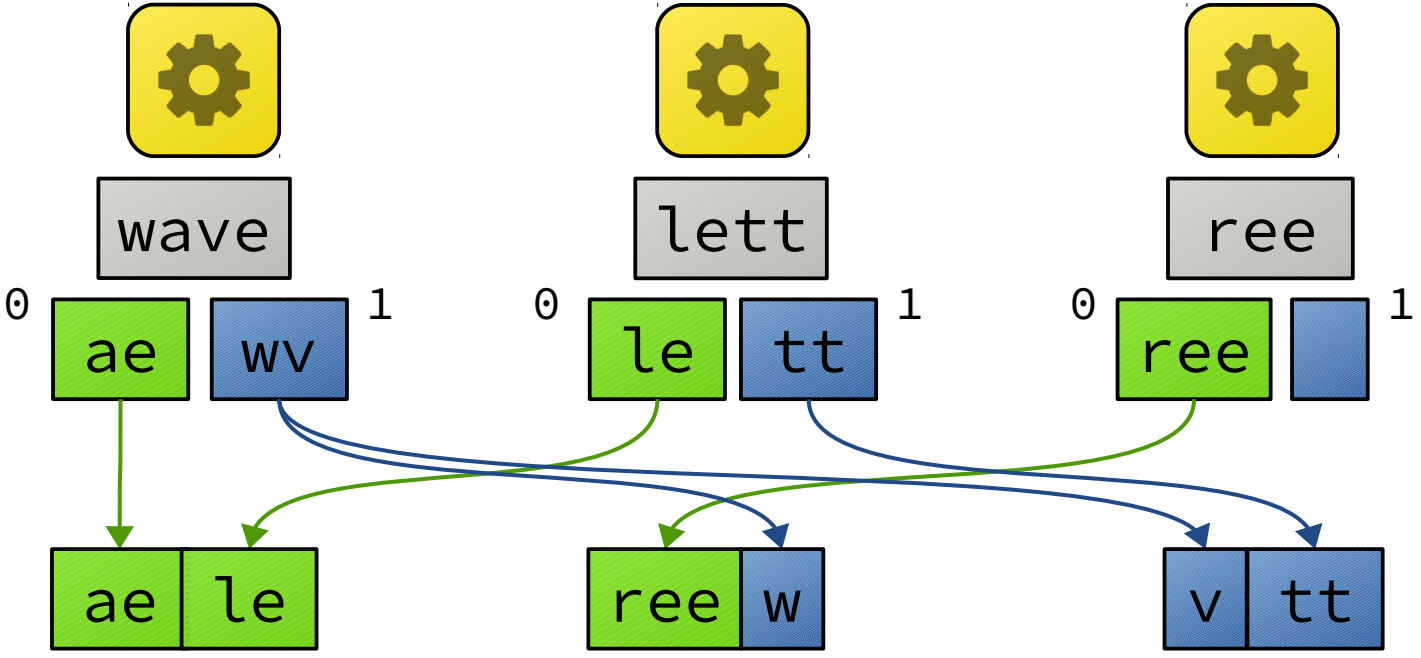
Stabiles Sortieren

Verteilter Bucket Sort



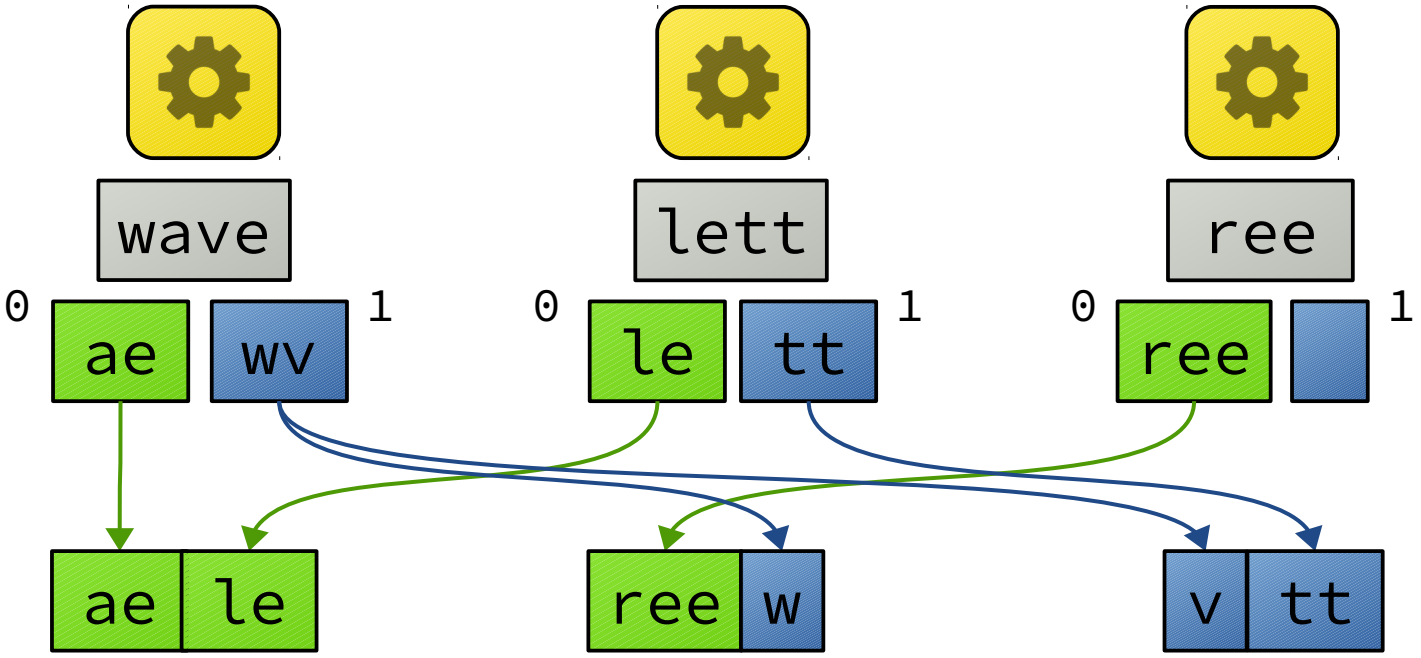
Stabiles Sortieren

Verteilter Bucket Sort



Stabiles Sortieren

Verteilter Bucket Sort



→ *Levelweise Domain Decomposition!*

Übersicht BSP-Kosten

$\mathcal{O}(\dots)$



DD

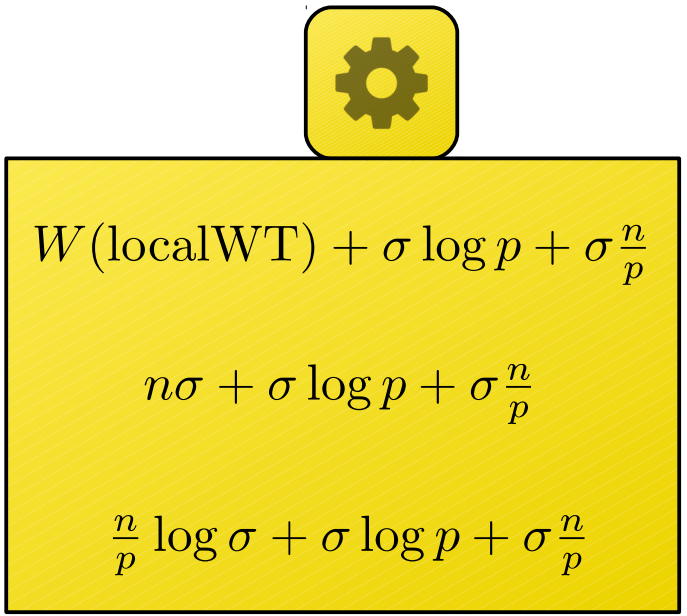
Split

$$p\sigma + n \log \sigma + H(\mathrm{DD})$$

Sort

Übersicht BSP-Kosten

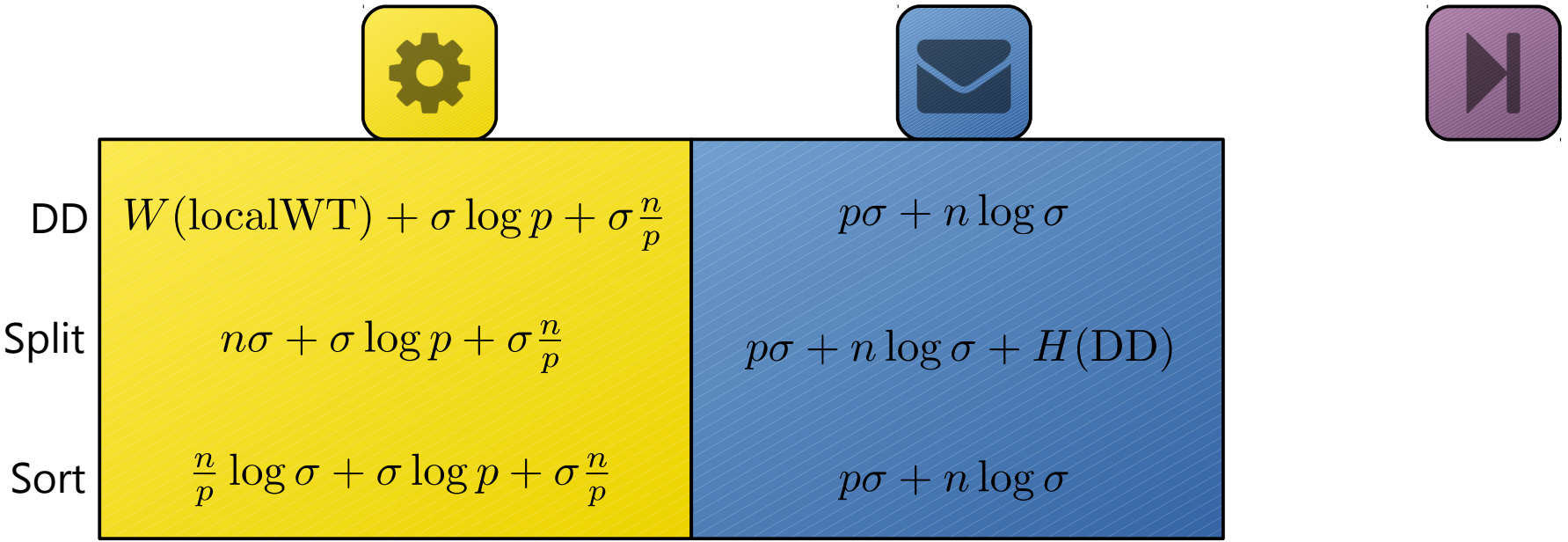
$\mathcal{O}(\dots)$



$p\sigma + n \log \sigma + H(\text{DD})$

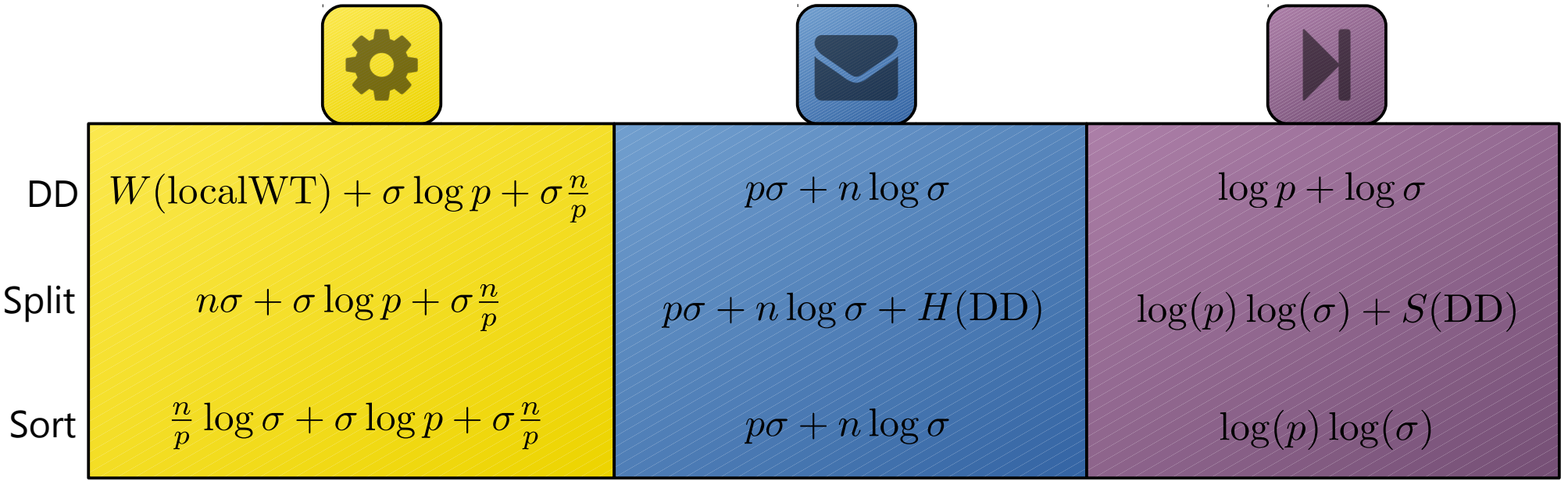
Übersicht BSP-Kosten

$\mathcal{O}(\dots)$



Übersicht BSP-Kosten

$\mathcal{O}(\dots)$



Implementierung

Implementierung

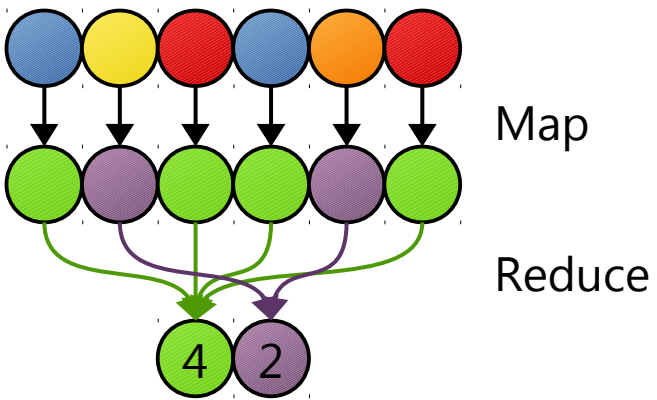
Thrill

- Framework in C++14
- Distributed Immutable Arrays
- Datenflussbasierte High-Level-Operationen auf Sequenzen

Implementierung

Thrill

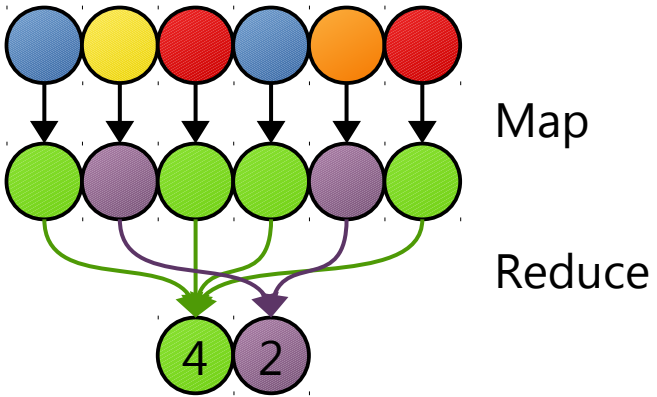
- Framework in C++14
- Distributed Immutable Arrays
- Datenflussbasierte High-Level-Operationen auf Sequenzen



Implementierung

Thrill

- Framework in C++14
- Distributed Immutable Arrays
- Datenflussbasierte High-Level-Operationen auf Sequenzen



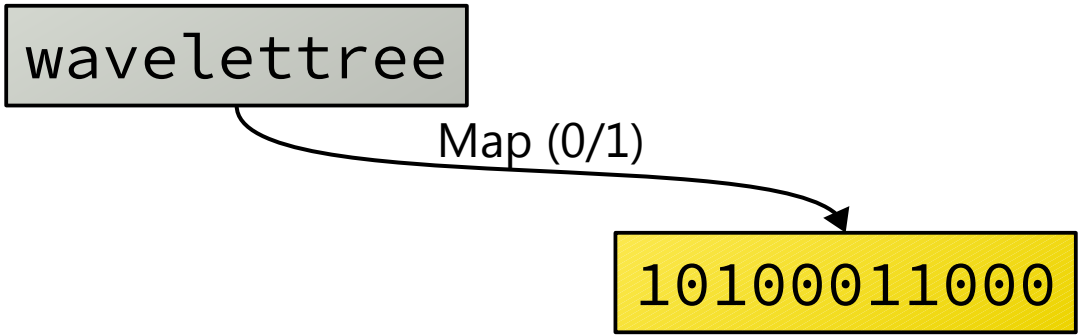
MPI (Message Passing Interface)

- C-Bibliothek
- Low-Level-Schnittstelle zum Nachrichtenaustausch
- Primitive: Send / Recv / Probe
- Komplex: Allreduce, Exscan

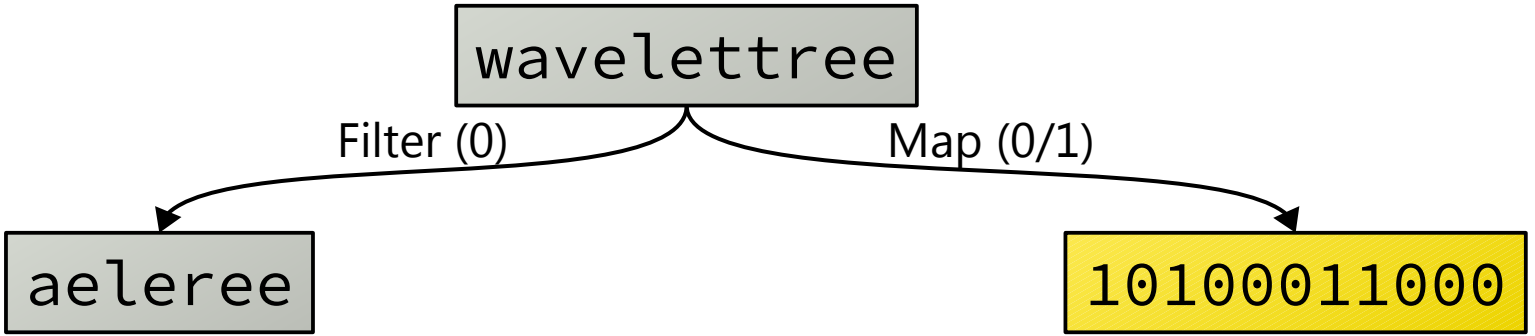
Thrill

```
wavelettree
```

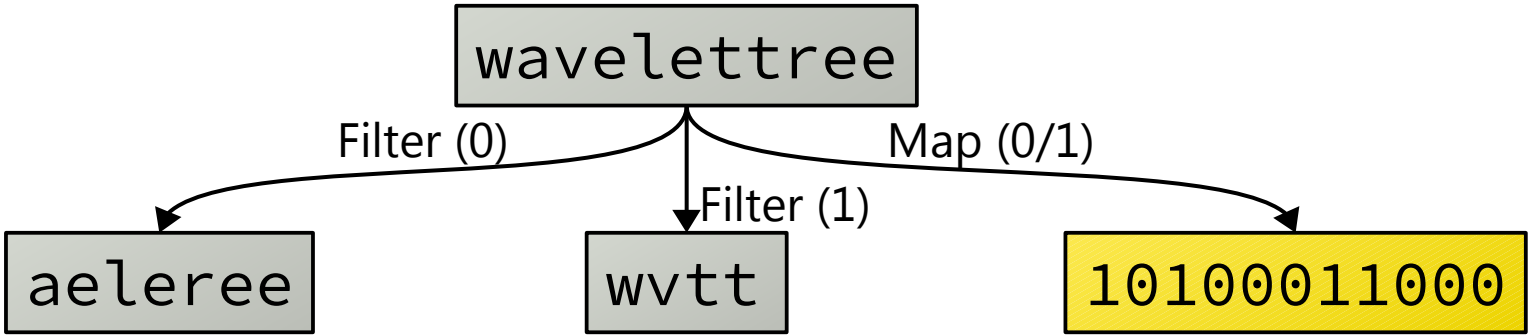
Thrill



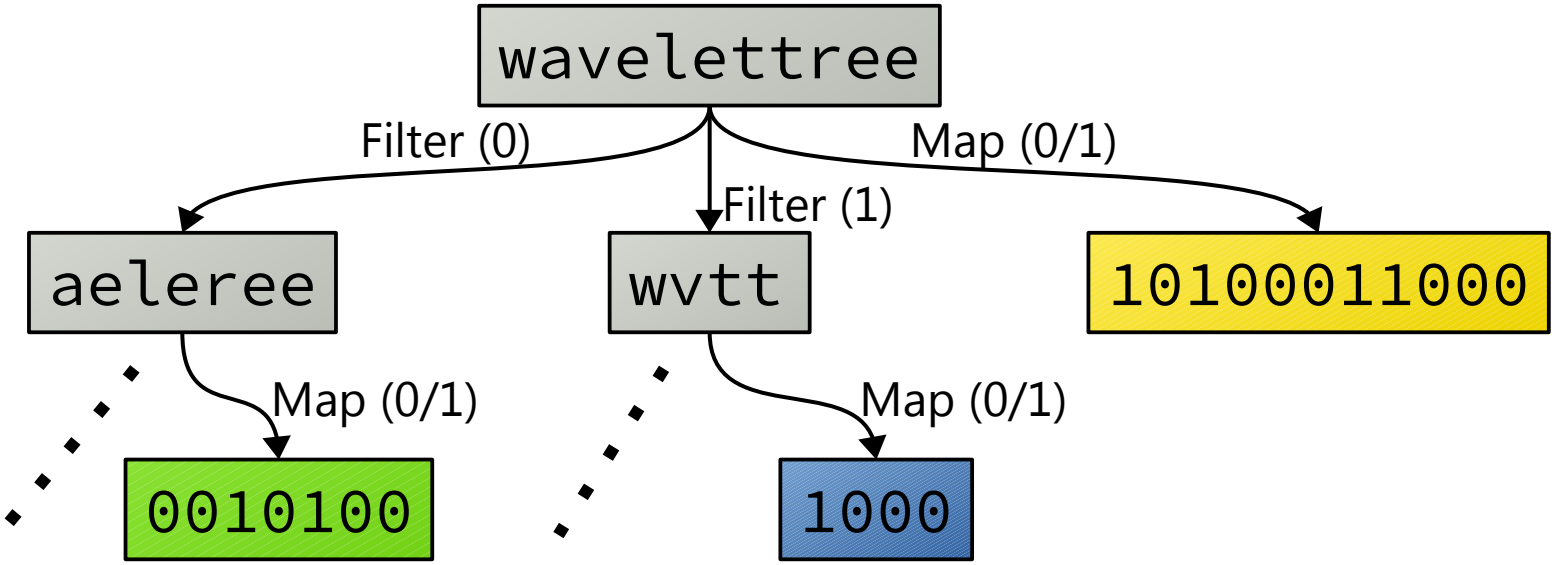
Thrill



Thrill

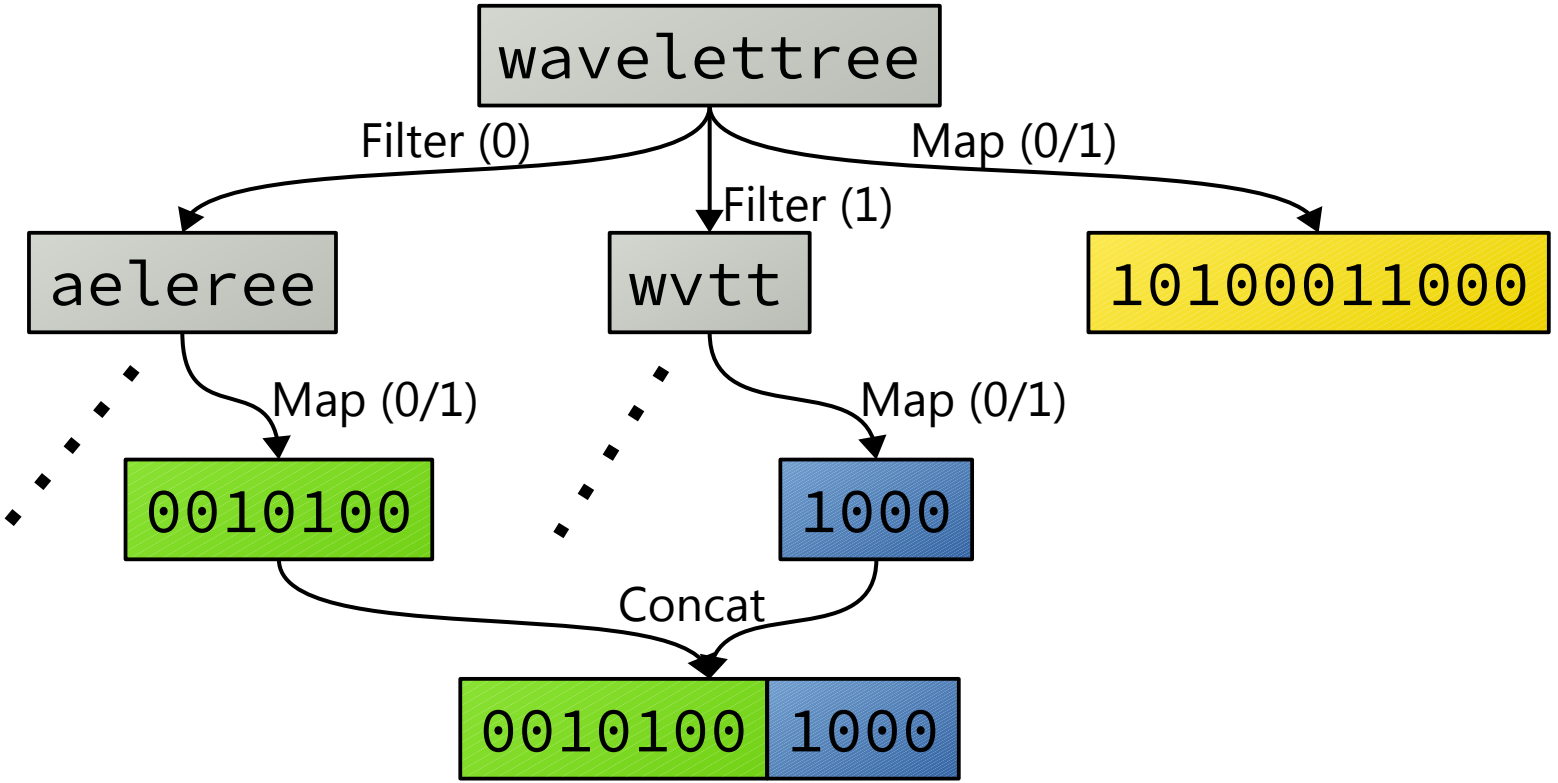

wavelettree
Filter (0)
Filter (1)
Map (0/1)
aeleree
wvtt
10100011000

Thrill


wavelettree
Filter (0)
Filter (1)
Map (0/1)
aeleree
wvtt
10100011000
Map (0/1)
Map (0/1)
0010100
1000

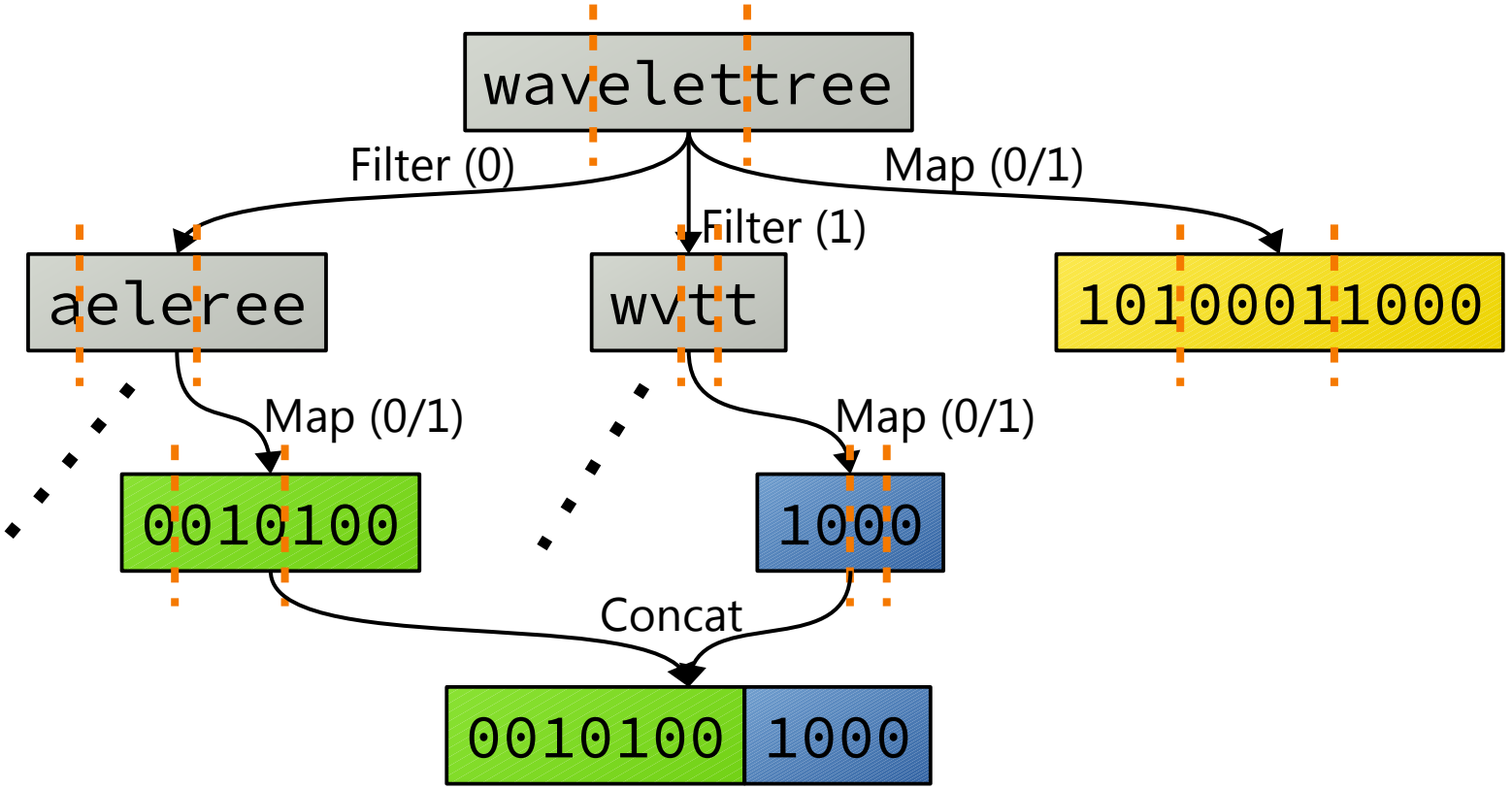
Thrill

Domain Decomposition



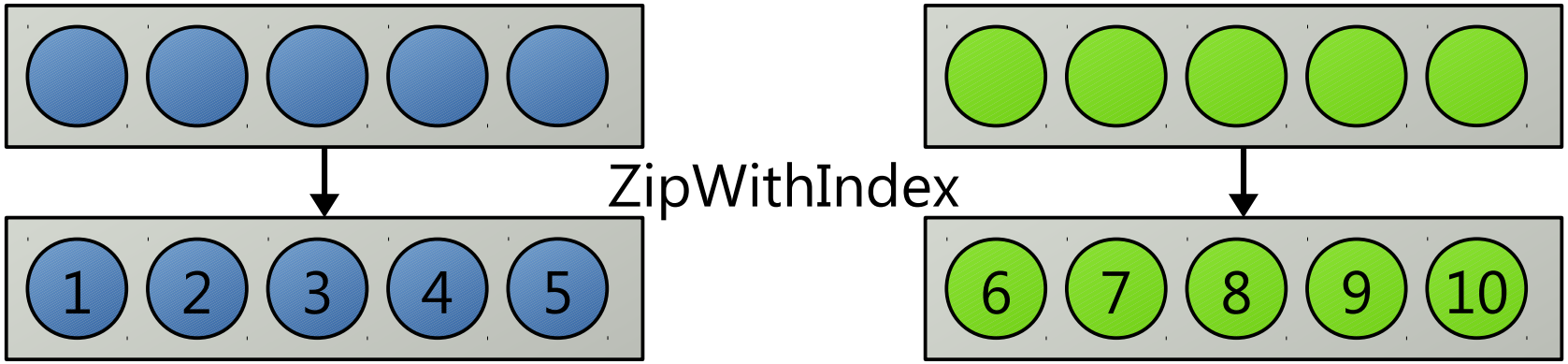
Thrill

Domain Decomposition

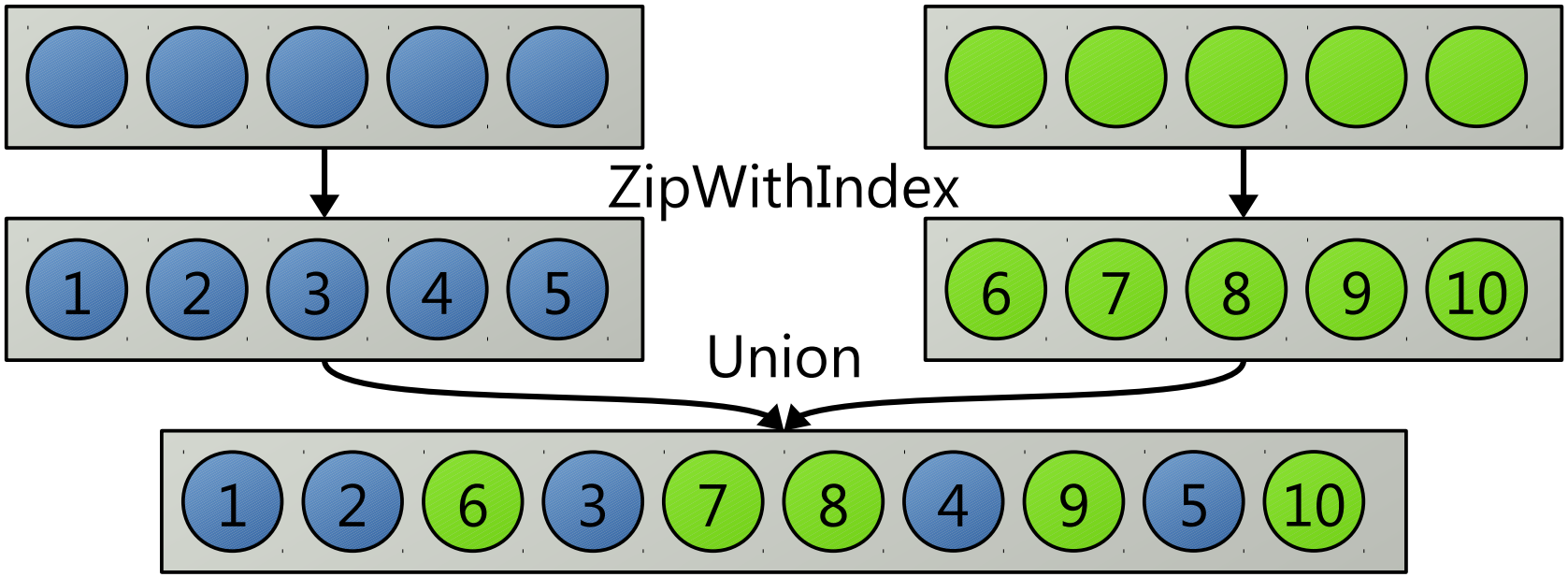


Concat in Thrill

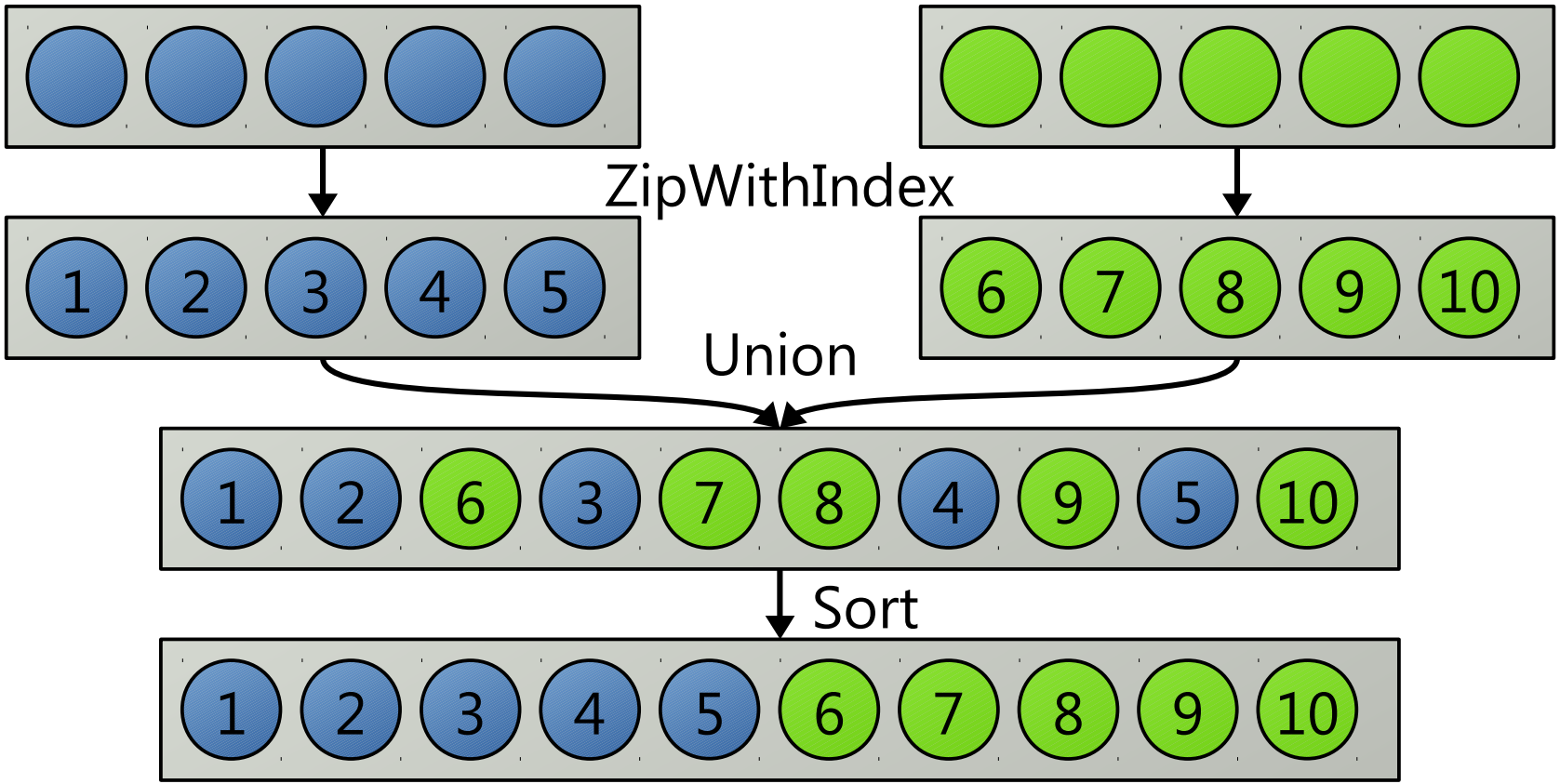
Concat in Thrill



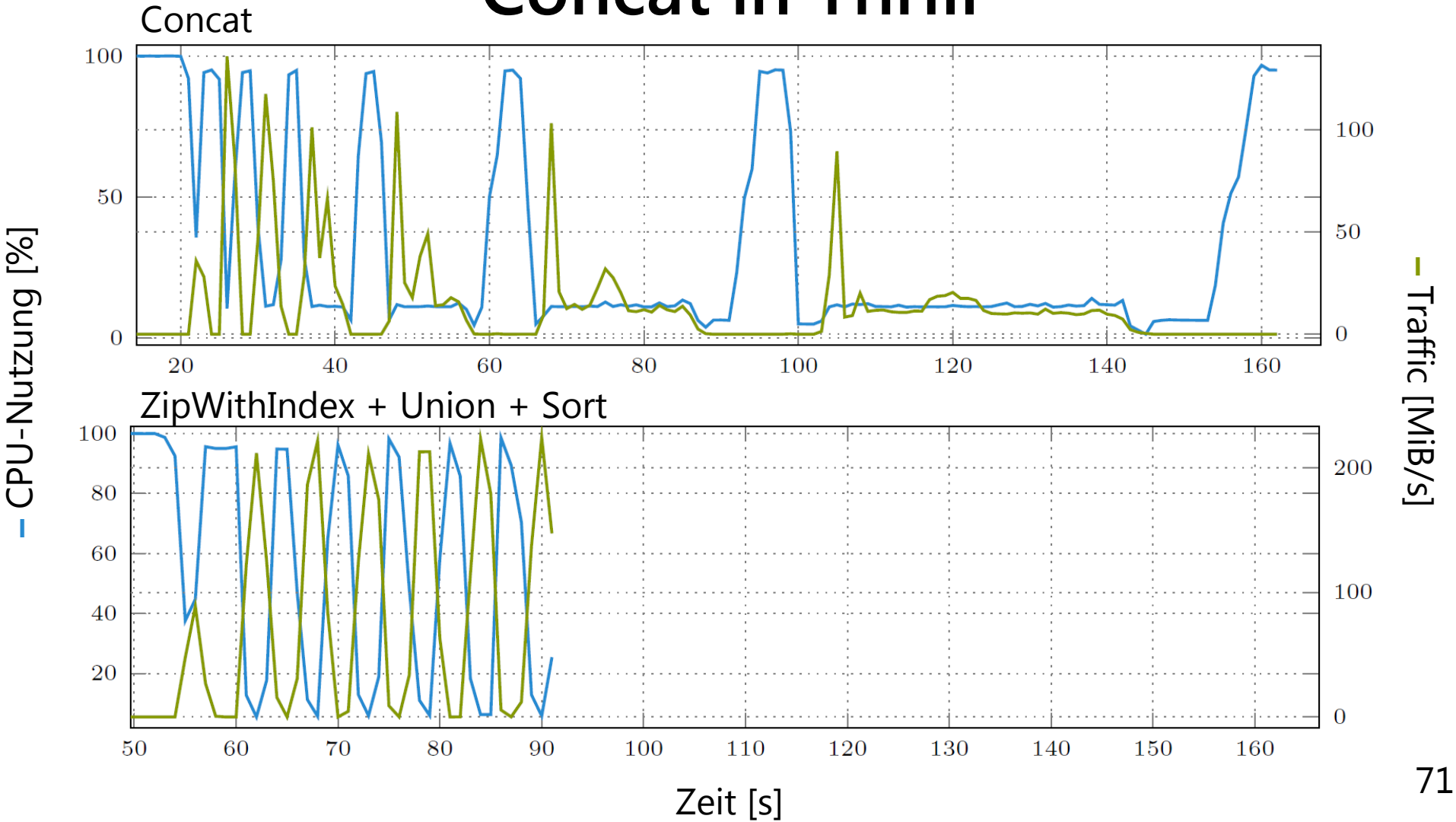
Concat in Thrill



Concat in Thrill



Concat in Thrill

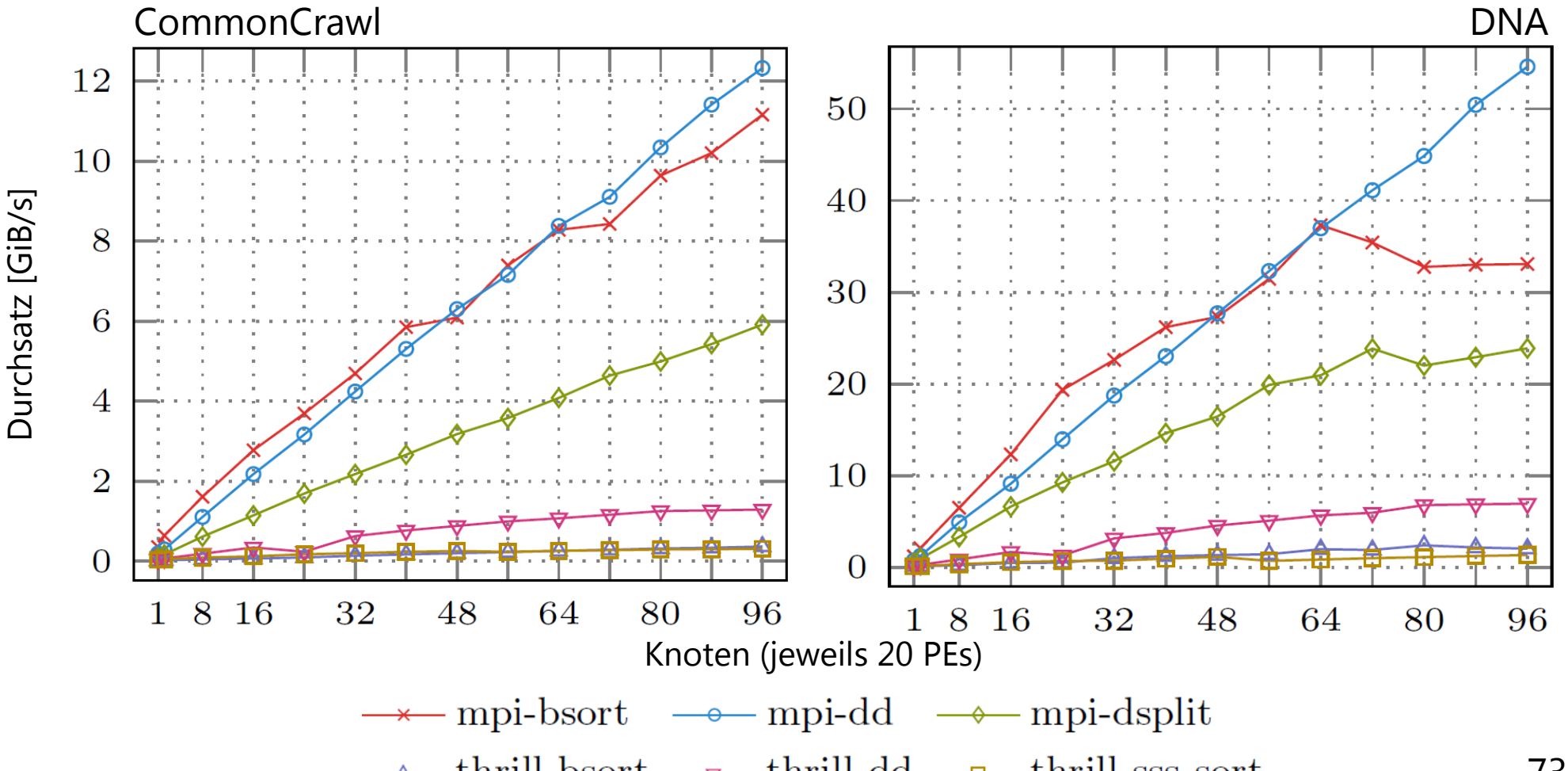


Evaluation

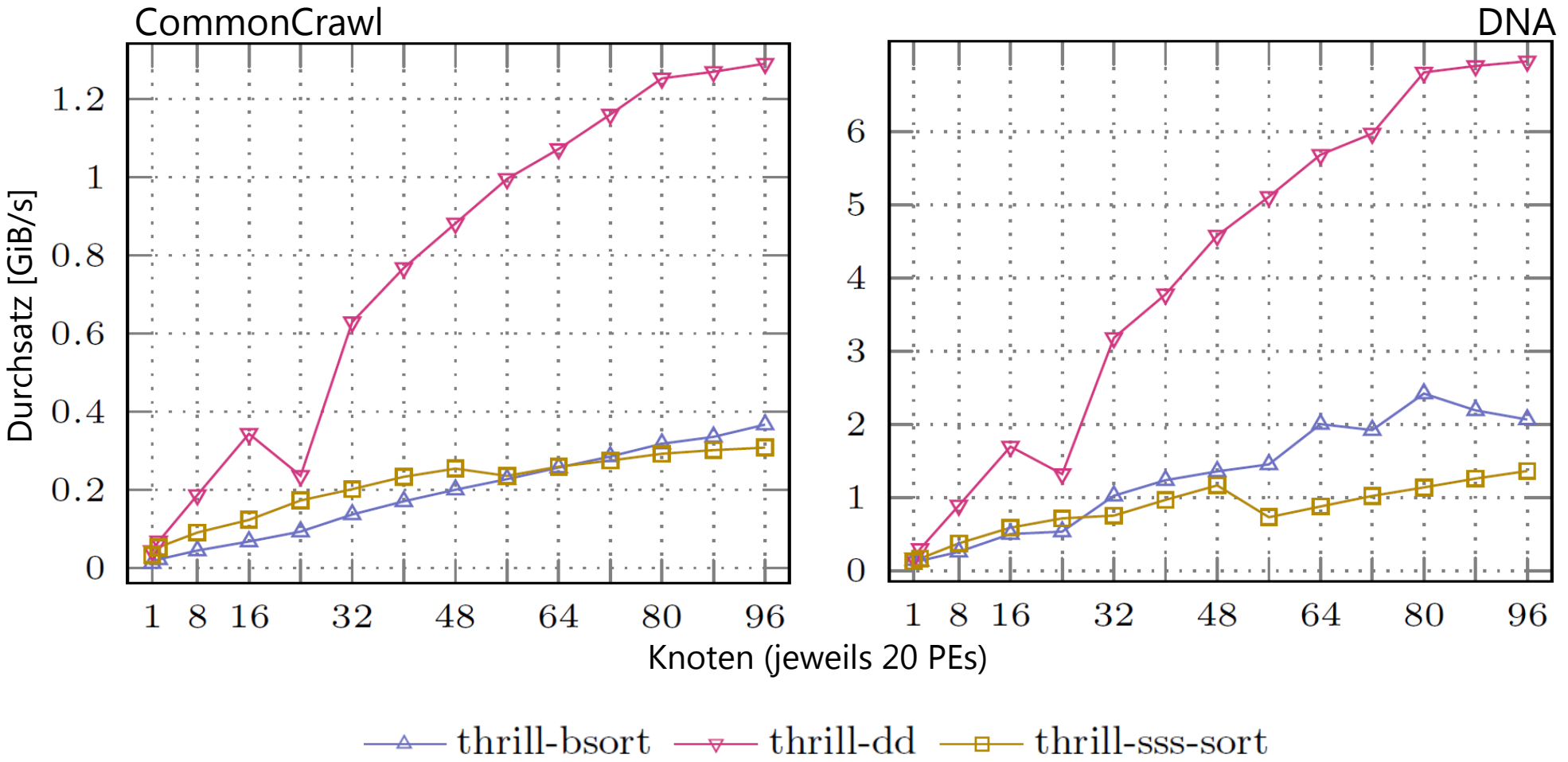
Weak-Scaling-Experimente

- LiDO3-Cluster (20 PEs je Knoten)
- CommonCrawl (σ=242 → 8 WT-Level)
- DNA (σ=4 → 2 WT-Level)
- 1 GiB je Knoten
- Thrill- und MPI-Implementierungen

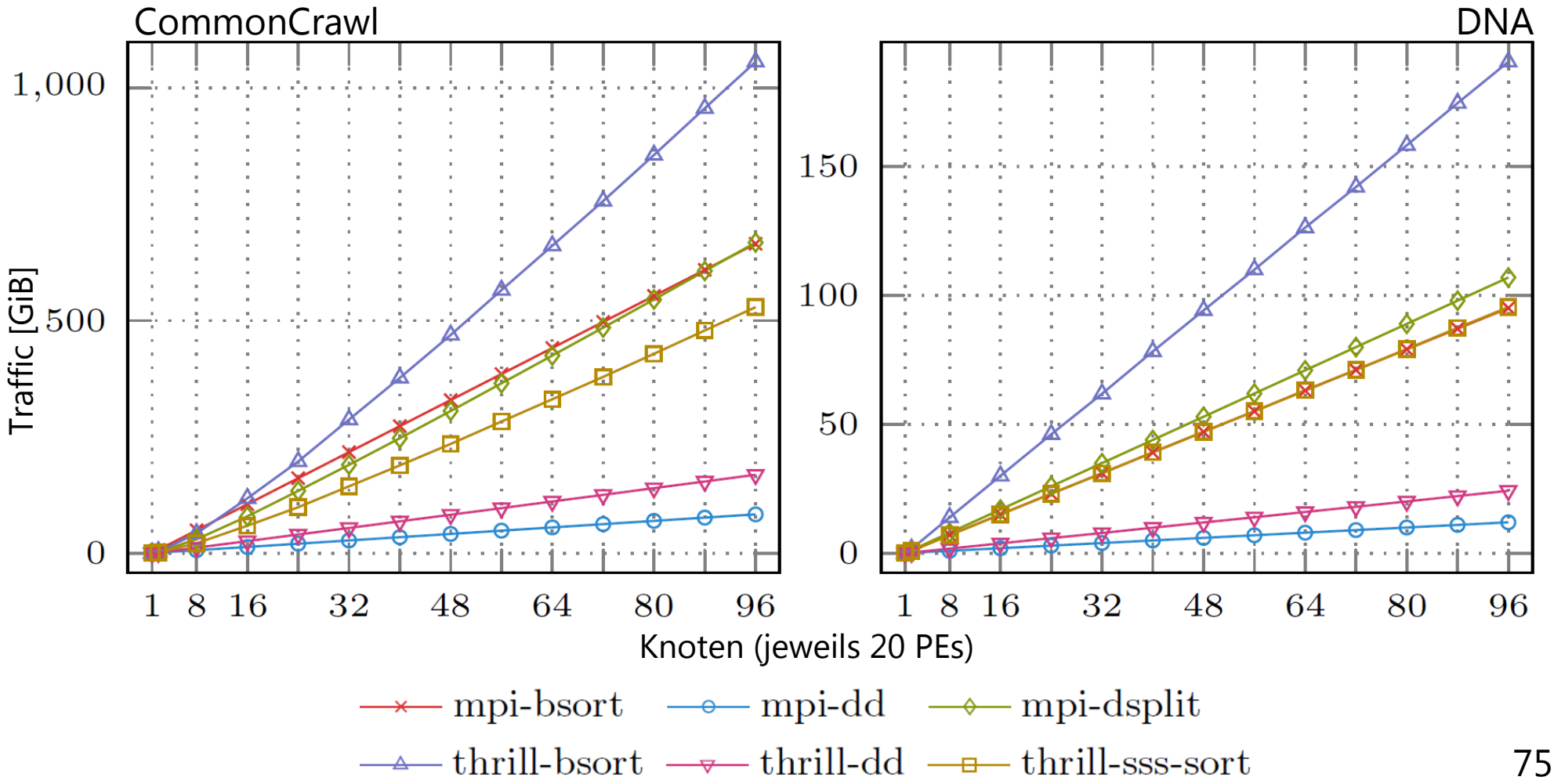
Weak Scaling – Durchsatz



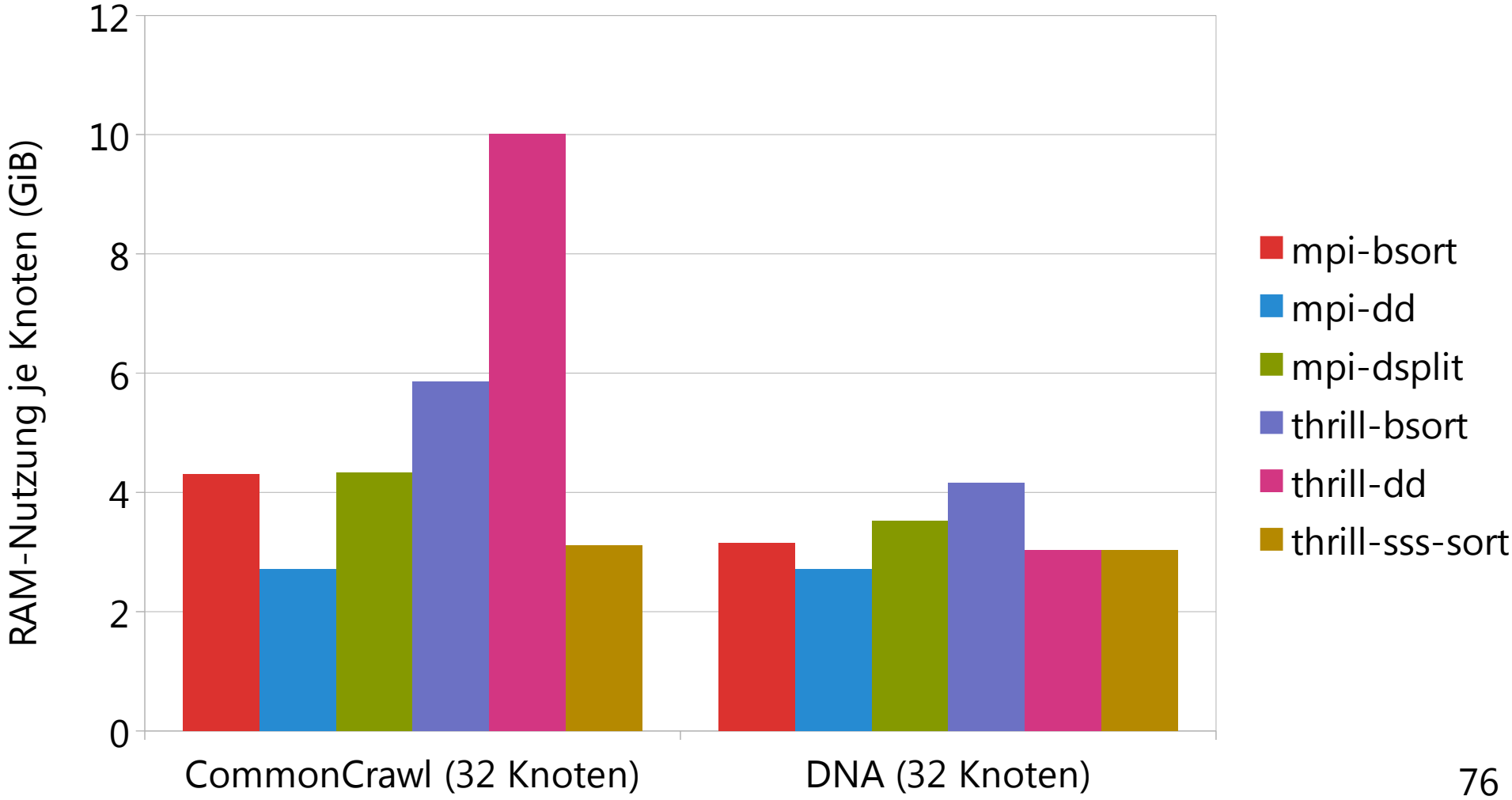
Weak Scaling – Durchsatz (Thrill)



Weak Scaling - Traffic



Weak Scaling - Speichernutzung



Fazit

→ Verteilte WT-Konstruktion skaliert im Allgemeinen recht gut
→ Domain Decomposition skaliert nahezu perfekt und benötigt den geringsten Traffic und Speicherplatz