tf/idf computation

Florian Thomas, Christian Reß
Map/Reduce Algorithms on Hadoop
6. Juli 2009

tf/idf computation

- **Was ist tf/idf?**
- **Verschiedene Implementierungen**
- **Map/Reduce-Aufbau**
- **Implementierungsbesonderheiten**
- **Analyse**
- **Fazit**

Was ist tf/idf?

term frequency

$$\mathbf{tf}_{i,j} = \frac{freq_{i,j}}{\sum_k freq_{k,j}}$$

---

inverse document frequency

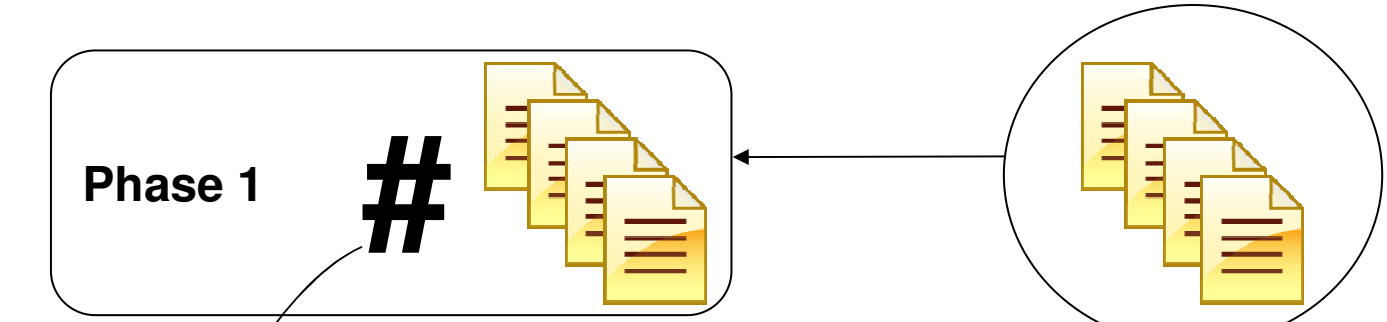
$$\mathbf{idf}_i = \log\left(\frac{|D|}{|\{d|t_i \in d\}|}\right)$$

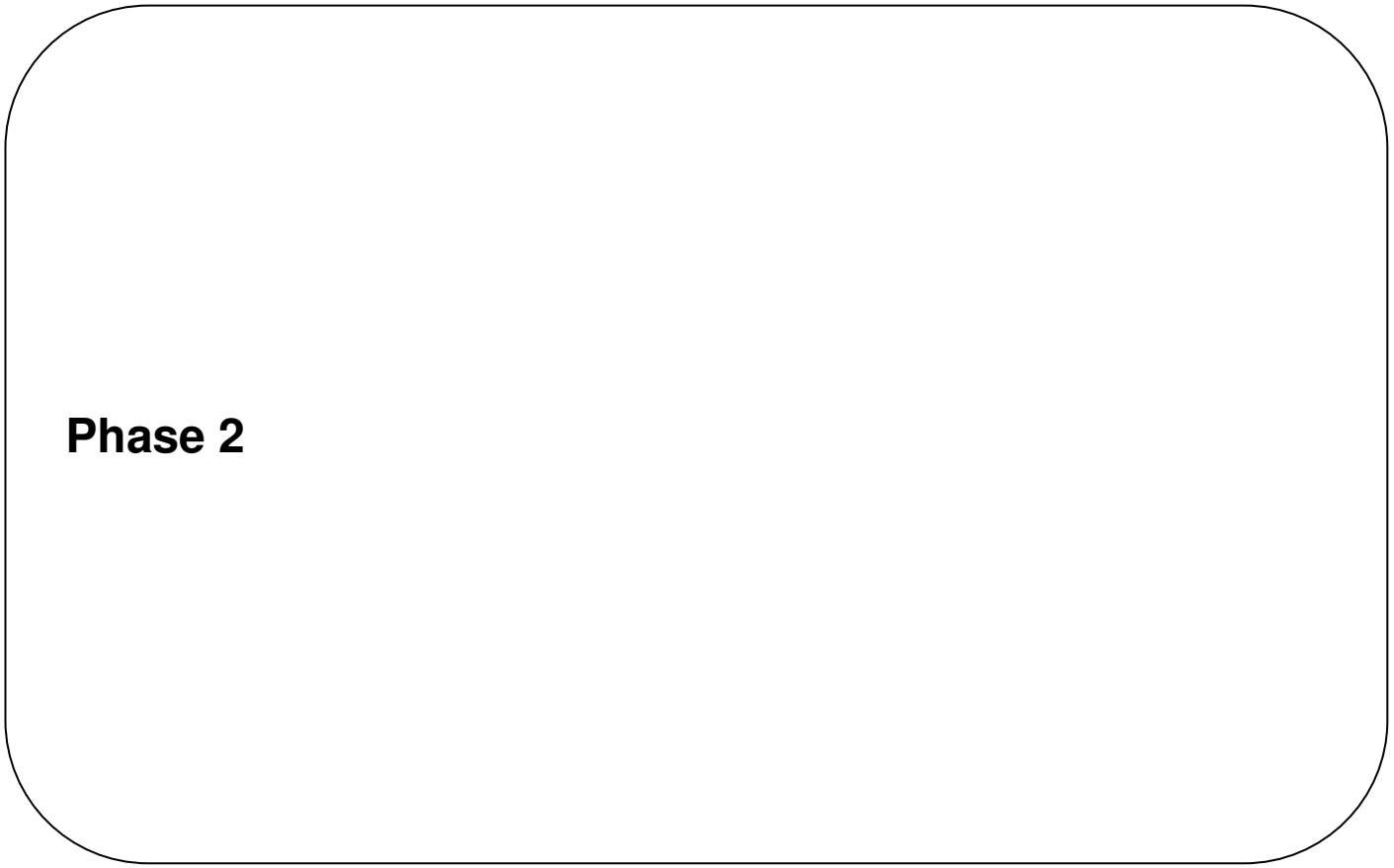
---

tf/idf

$$\mathbf{(tf - idf)}_{\mathbf{i,j}} = tf_{i,j} \cdot idf_i$$

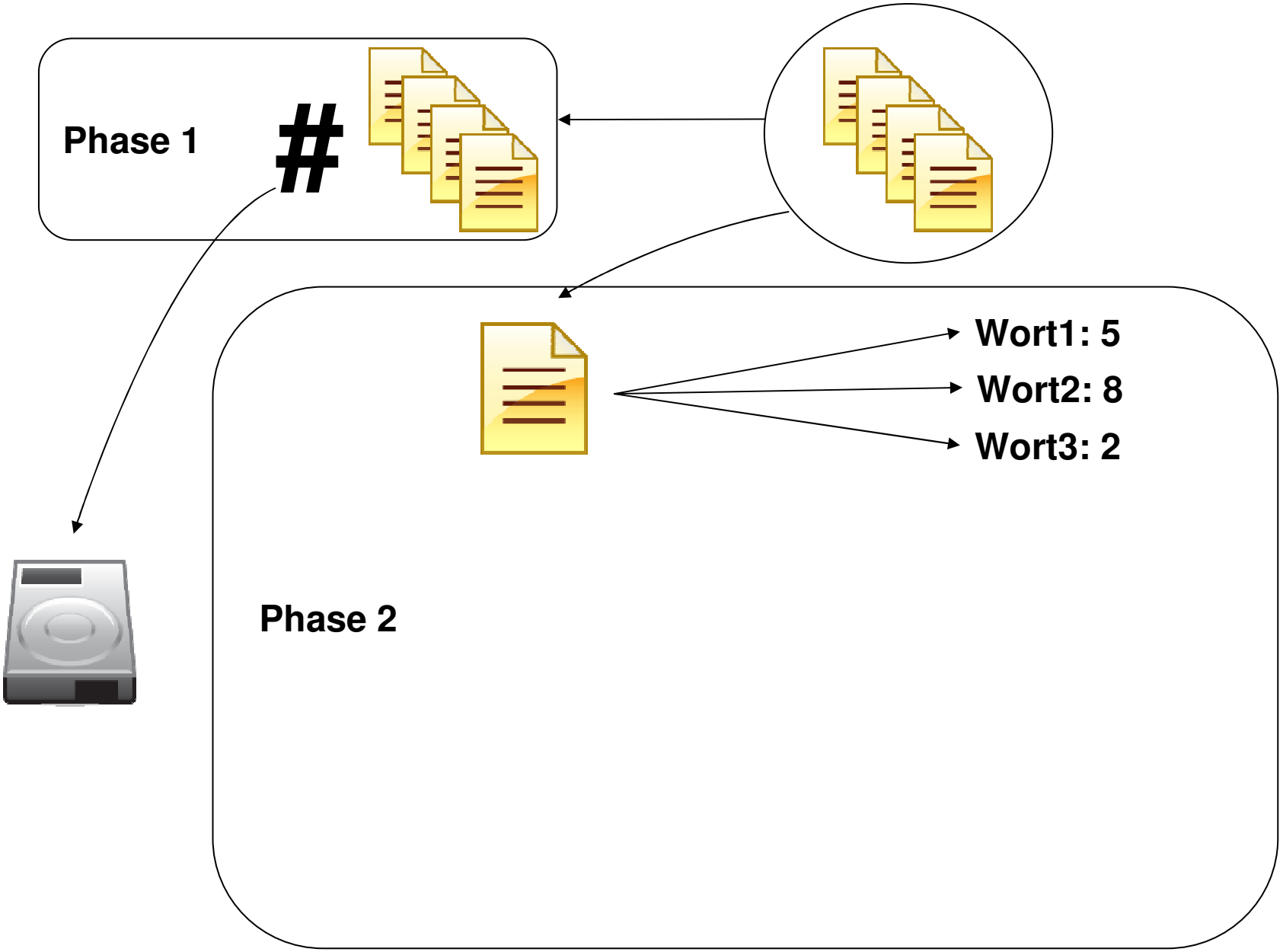
Implementierungen – 2 Phasen





$$idf_i = \log\left(\frac{|D|}{|\{d|t_i \in d\}|}\right)$$

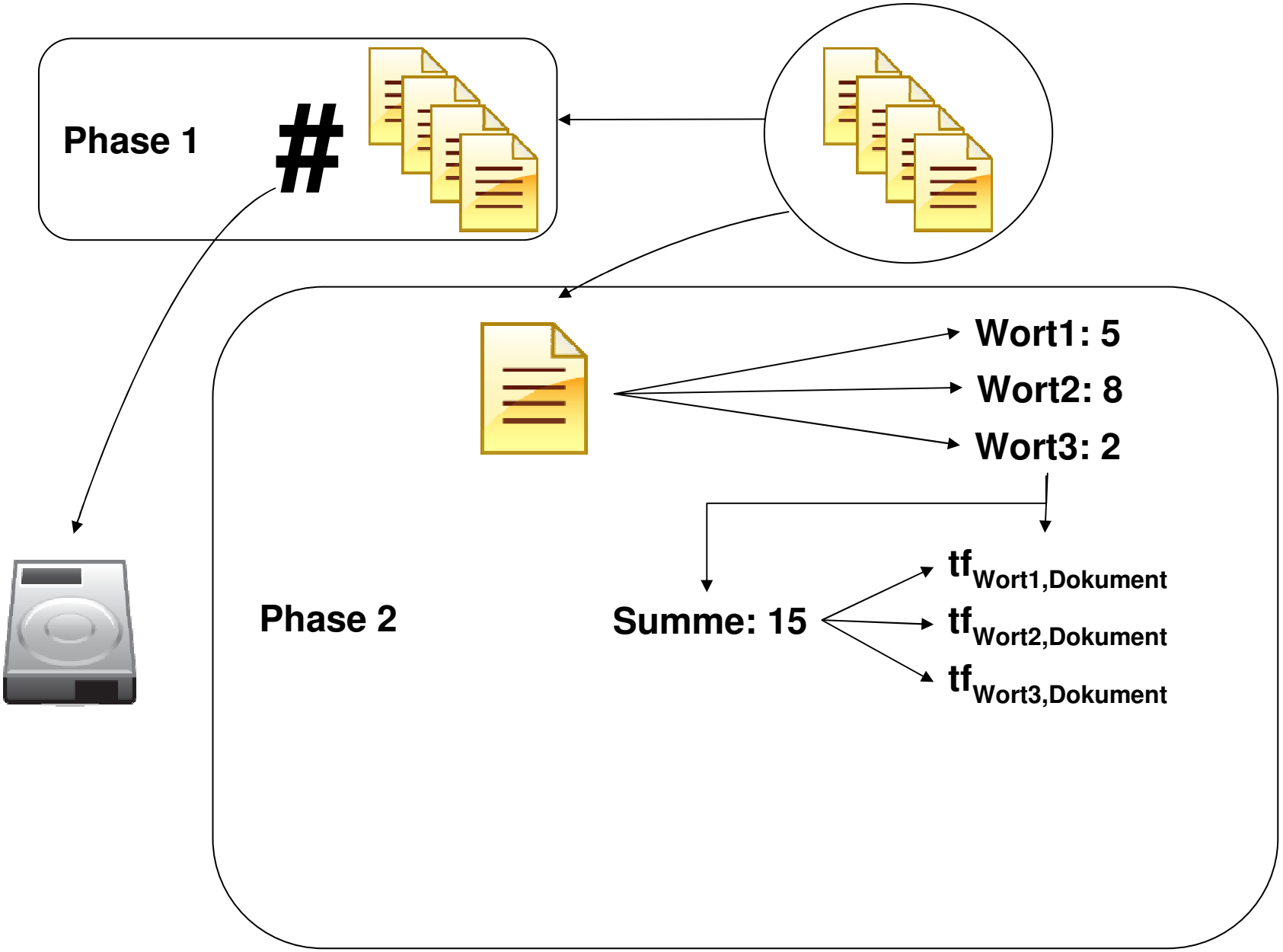
Implementierungen – 2 Phasen



$$idf_i = \log\left(\frac{|D|}{|\{d|t_i \in d\}|}\right)$$

$$tf_{i,j} = \frac{freq_{i,j}}{\sum_k freq_{k,j}}$$

Implementierungen – 2 Phasen

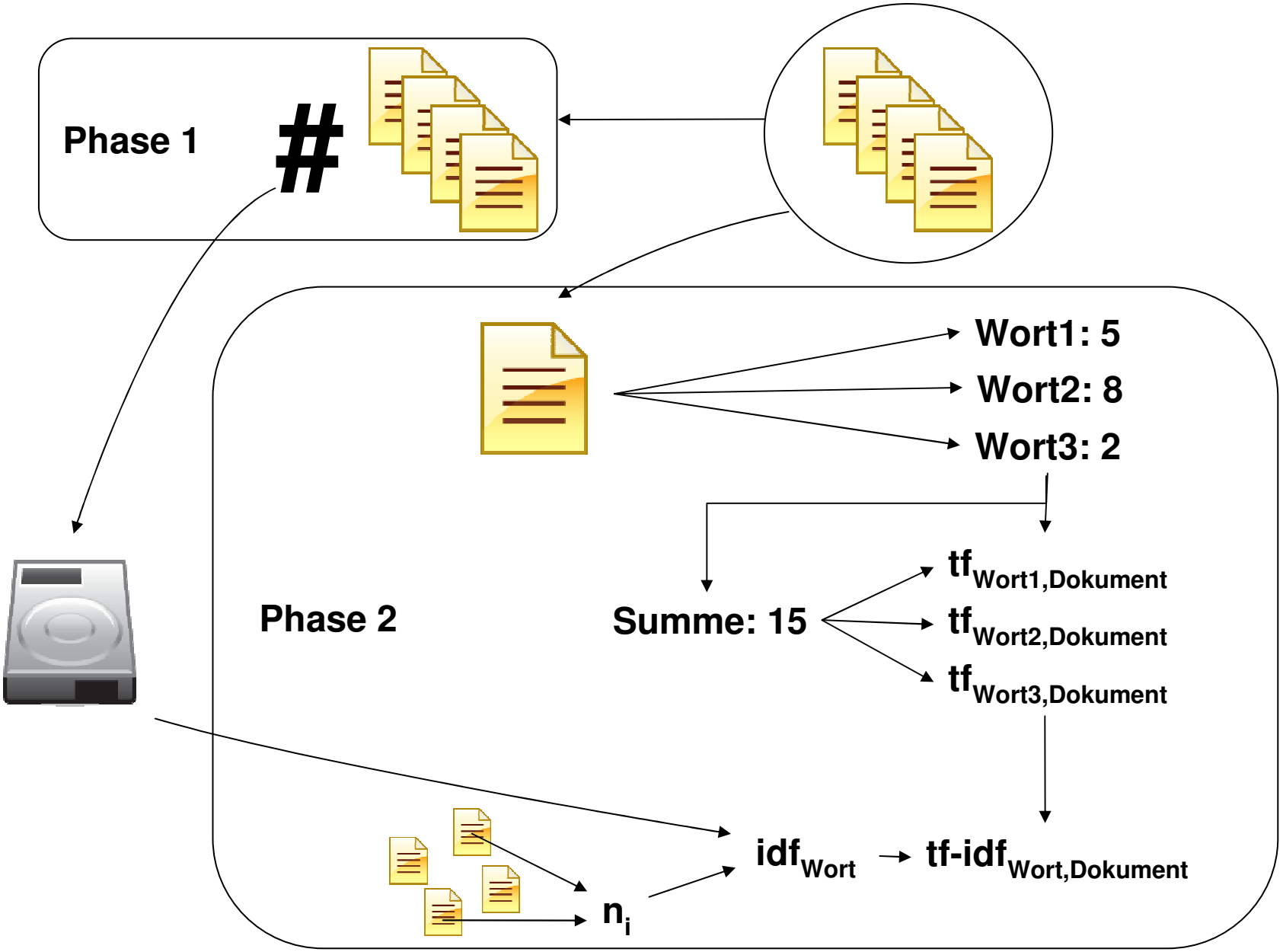


$$idf_i = \log\left(\frac{|D|}{|\{d|t_i \in d\}|}\right)$$

$$tf_{i,j} = \frac{freq_{i,j}}{\sum_k freq_{k,j}}$$

$$tf_{i,j} = \frac{freq_{i,j}}{\sum_k freq_{k,j}}$$

Implementierungen – 2 Phasen



$$idf_i = \log\left(\frac{|D|}{|\{d|t_i \in d\}|}\right)$$

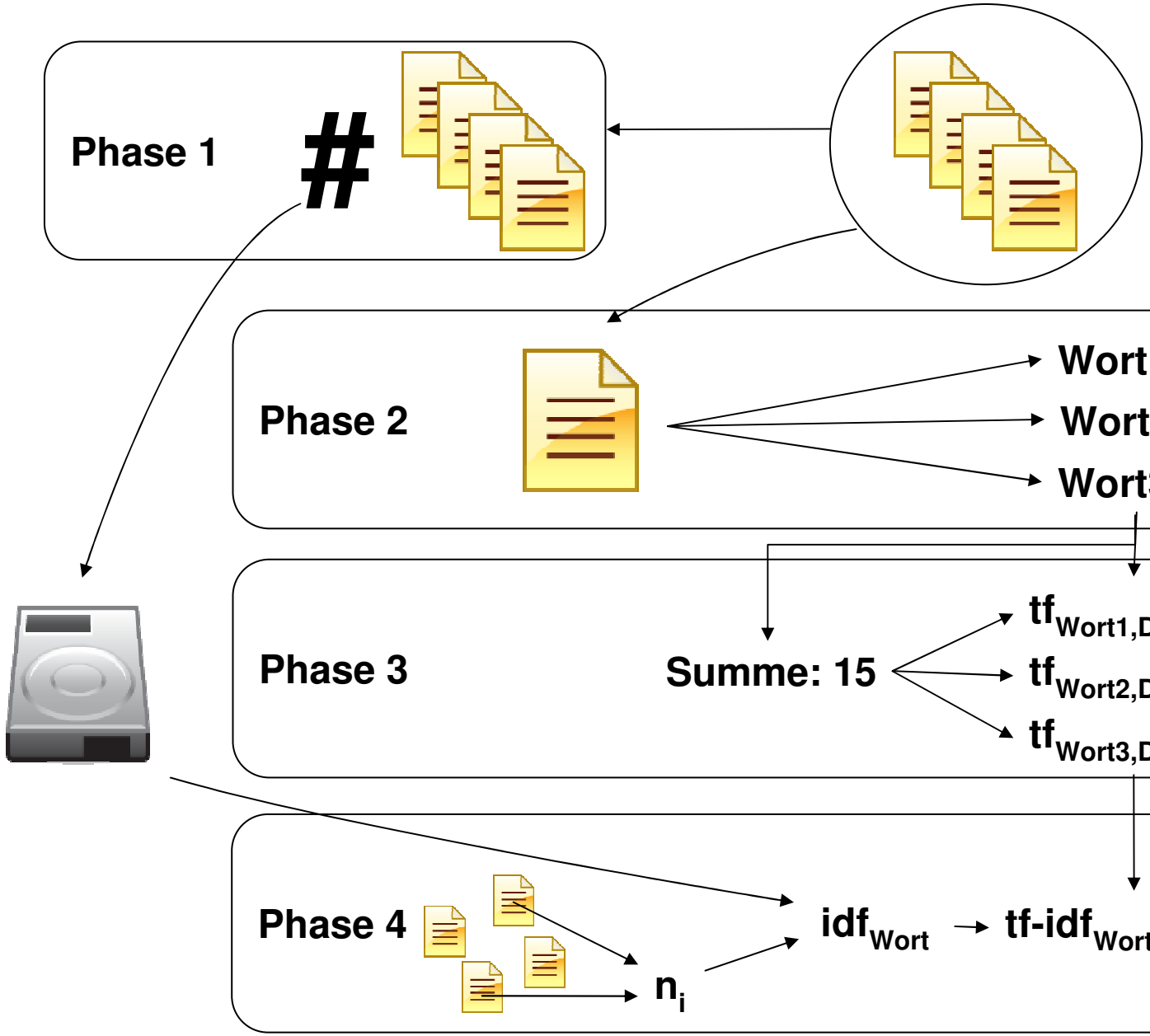
$$tf_{i,j} = \frac{freq_{i,j}}{\sum_k freq_{k,j}}$$

$$tf_{i,j} = \frac{freq_{i,j}}{\sum_k freq_{k,j}}$$

$$(tf\text{-}idf)_{i,j} = tf_{i,j} \cdot idf_i$$

$$= tf_{i,j} \cdot \log\left(\frac{|D|}{|\{d \mid t_i \in d\}|}\right)$$

Implementierungen – 4 Phasen



$$idf_i = \log\left(\frac{|D|}{|\{d|t_i \in d\}|}\right)$$

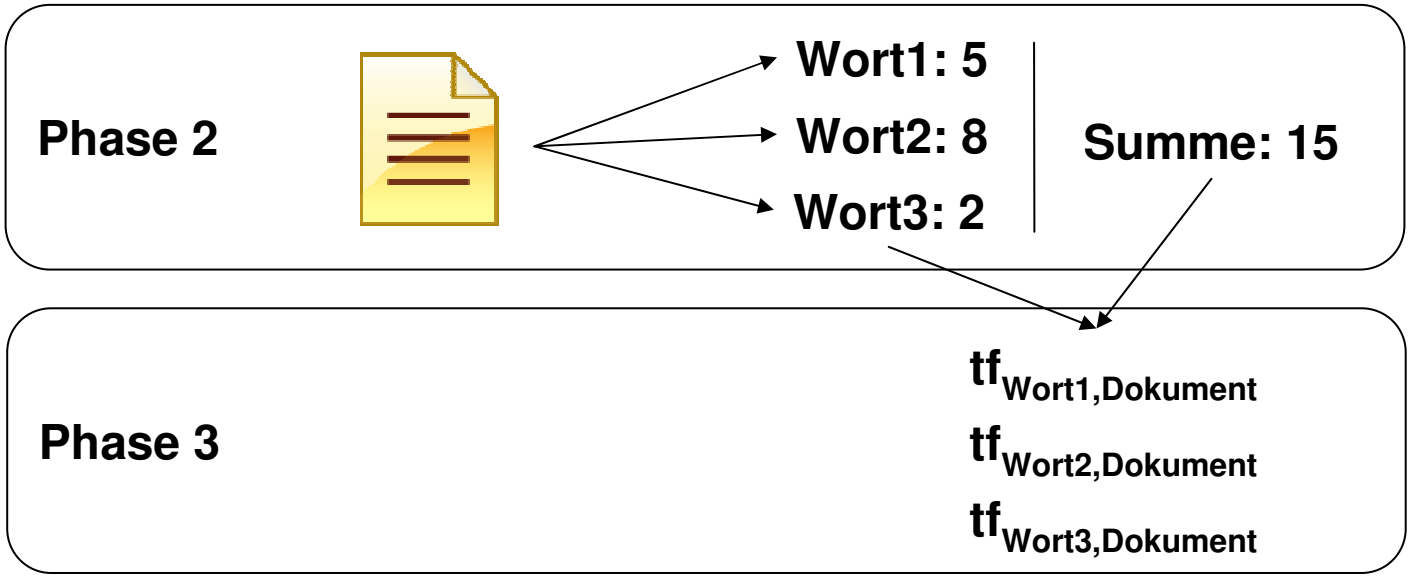
$$tf_{i,j} = \frac{freq_{i,j}}{\sum_k freq_{k,j}}$$

$$tf_{i,j} = \frac{freq_{i,j}}{\sum_k freq_{k,j}}$$

$$(tf\text{ - }idf)_{i,j} = tf_{i,j} \cdot idf_i$$

$$= tf_{i,j} \cdot \log\left(\frac{|D|}{|\{d|t_i \in d\}|}\right)$$

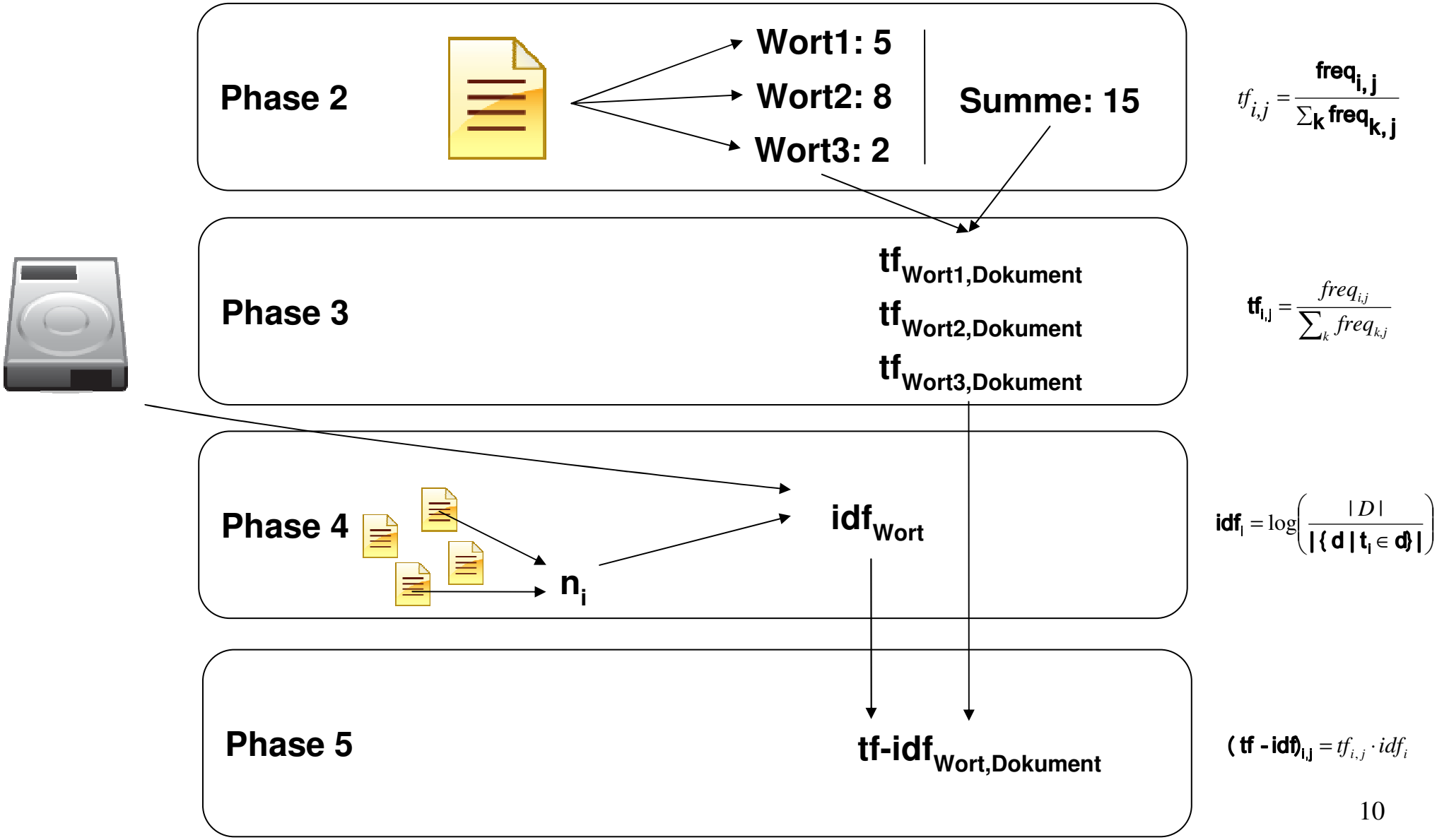
Implementierungen – 5 Phasen



$$tf_{i,j} = \frac{freq_{i,j}}{\sum_k freq_{k,j}}$$

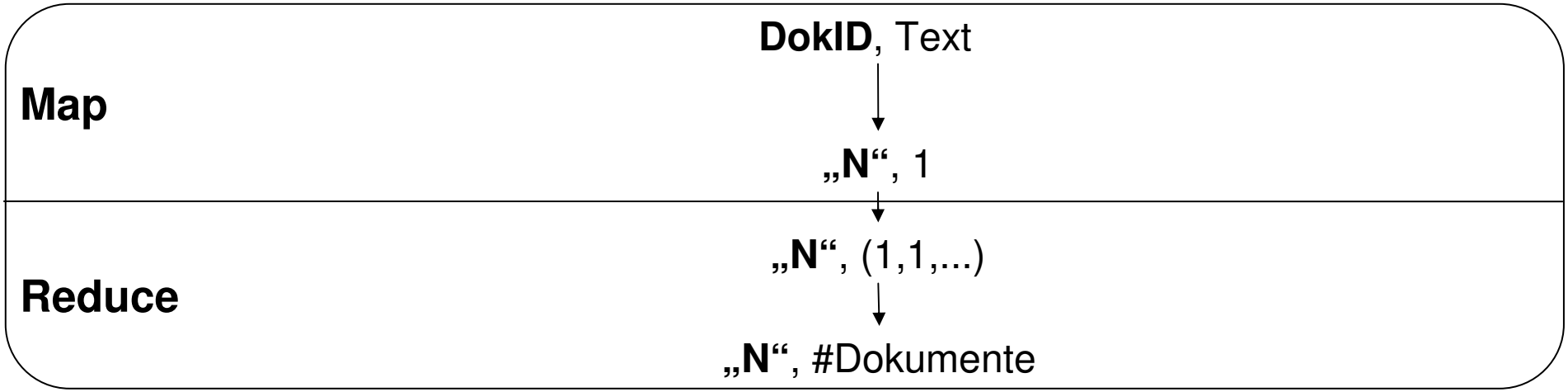
$$tf_{i,j} = \frac{freq_{i,j}}{\sum_k freq_{k,j}}$$

Implementierungen – 5 Phasen



Map/Reduce-Aufbau – 5 Phasen

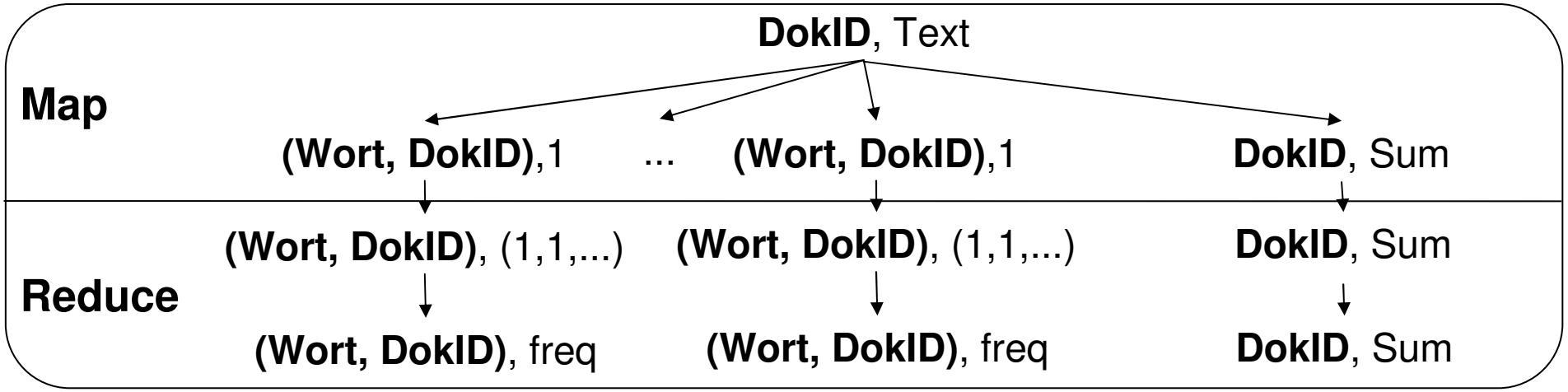
Phase 1: Dokumente zählen



$$idf_i = \log\left(\frac{|D|}{|\{d|t_i \in d\}|}\right)$$

Map/Reduce-Aufbau – 5 Phasen

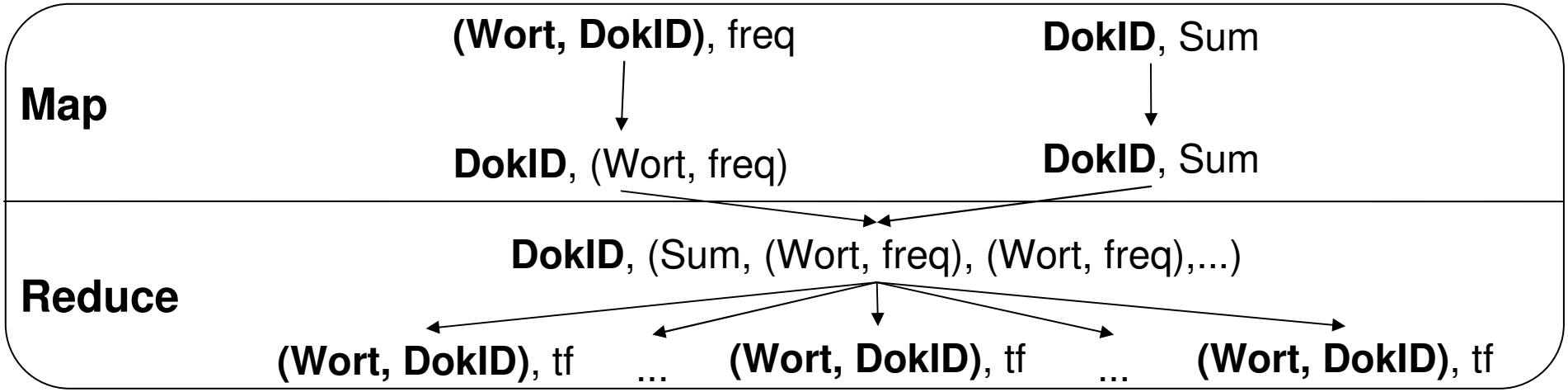
Phase 2: Wörter zählen in jedem Dokument



$$tf_{i,j} = \frac{\mathbf{freq_{i,j}}}{\sum_{\mathbf{k}} \mathbf{freq_{k,j}}}$$

Map/Reduce-Aufbau – 5 Phasen

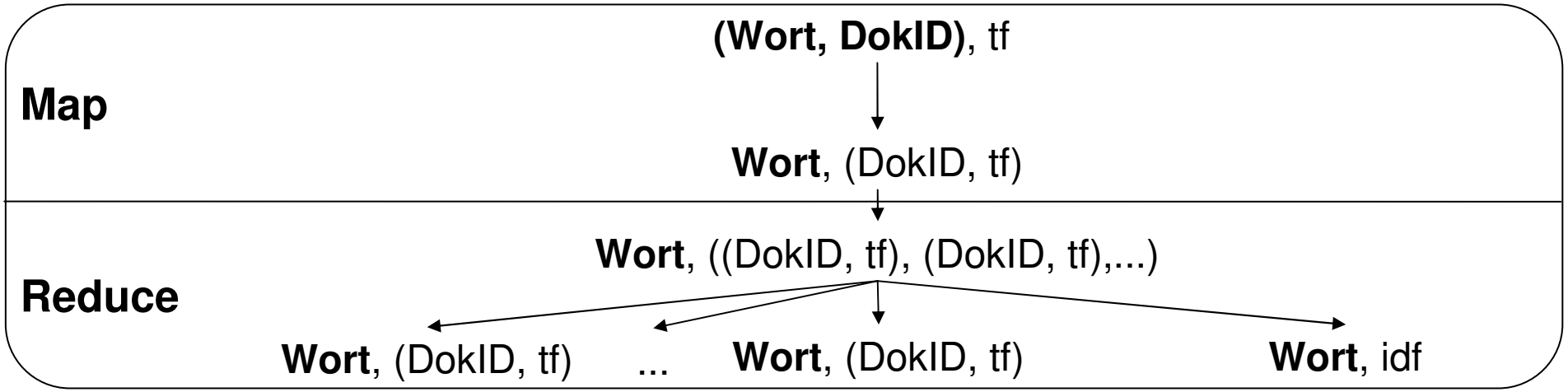
Phase 3: tf berechnen



$$\mathbf{tf_{i,j}} = \frac{freq_{i,j}}{\sum_k freq_{k,j}}$$

Map/Reduce-Aufbau – 5 Phasen

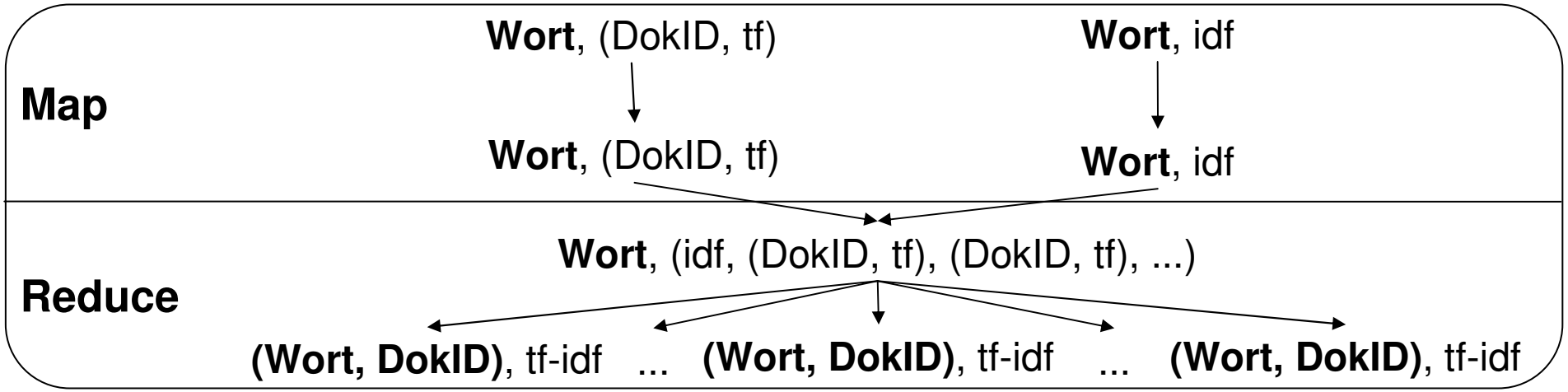
Phase 4: #Dokumente$_{Wort}$ bestimmen, idf berechnen



$$\mathbf{idf}_i = \log\left(\frac{|D|}{|\{\mathbf{d} \mid \mathbf{t}_i \in \mathbf{d}\}|}\right)$$

Map/Reduce-Aufbau – 5 Phasen

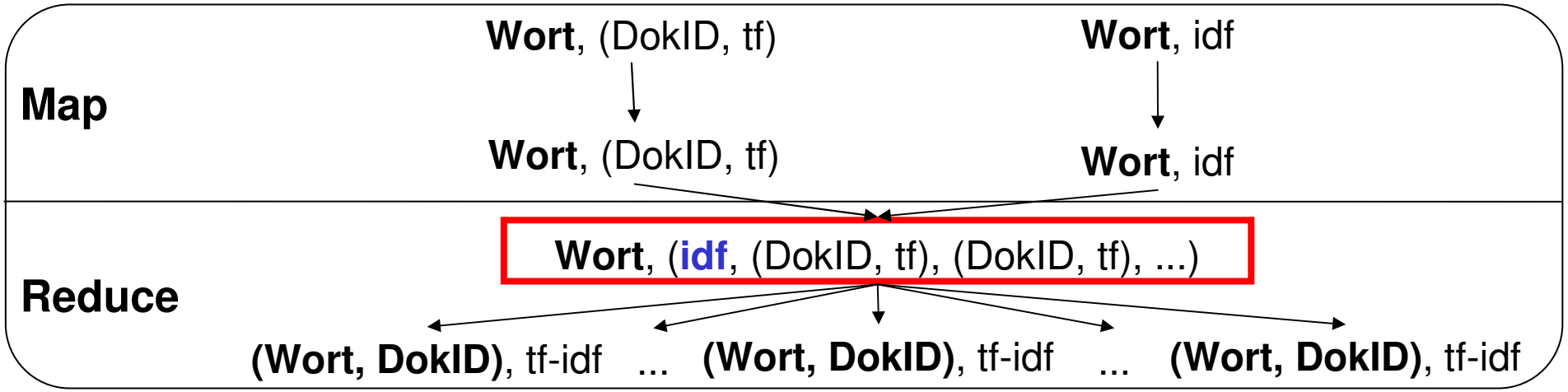
Phase 5: tf-idf berechnen



$$\mathbf{(tf - idf)_{i,j}} = tf_{i,j} \cdot idf_i$$

Implementierung – Secondary Sort

Motivation: Phase 5



Normalerweise: **Wort**, ((DokID, tf), (DokID, tf), ..., **idf**, …)

Implementierung – Secondary Sort

Hadoop – normales Verhalten

Wort	(DokID, tf)
mental	(1, 0.4)
mental	(3, 0.9)
mental	(5, 0.3)
mental	(0, 0.7)
mental	**1.8**
mental	(2, 0.6)
retardation	(8, 0.5)
retardation	**5.1**
retardation	(6, 0.1)
retardation	(9, 0.4)
retardation	(4, 0.8)

Implementierung – Secondary Sort

Hadoop – normales Verhalten

Wort	(DokID, tf)	
mental	(1, 0.4)	→ **mental**, ((1, 0.4), (3, 0.9), (5, 0.3), (0, 0.7), **1.8**, (2, 0.6))
mental	(3, 0.9)	
mental	(5, 0.3)	
mental	(0, 0.7)	
mental	**1.8**	
mental	(2, 0.6)	
retardation	(8, 0.5)	→ **retardation**, ((8, 0.5), **5.1**, (6, 0.1), (9, 0.4), (4, 0.8))
retardation	**5.1**	
retardation	(6, 0.1)	
retardation	(9, 0.4)	
retardation	(4, 0.8)	

Implementierung – Secondary Sort

Hadoop – gewünschtes Verhalten

Wort	(DokID, tf)
mental 1	(1, 0.4)
mental 1	(3, 0.9)
mental 1	(5, 0.3)
mental 1	(0, 0.7)
mental 0	**1.8**
mental 1	(2, 0.6)
retardation 1	(8, 0.5)
retardation 0	**5.1**
retardation 1	(6, 0.1)
retardation 1	(9, 0.4)
retardation 1	(4, 0.8)

- **Anhängen eines Index-Wertes an den Schlüssel**

Implementierung – Secondary Sort

Hadoop – gewünschtes Verhalten

Wort	(DokID, tf)
mental 0	**1.8**
mental 1	(1, 0.4)
mental 1	(3, 0.9)
mental 1	(5, 0.3)
mental 1	(0, 0.7)
mental 1	(2, 0.6)
retardation 0	**5.1**
retardation 1	(8, 0.5)
retardation 1	(6, 0.1)
retardation 1	(9, 0.4)
retardation 1	(4, 0.8)

- **Anhängen eines Index-Wertes an den Schlüssel**

Implementierung – Secondary Sort

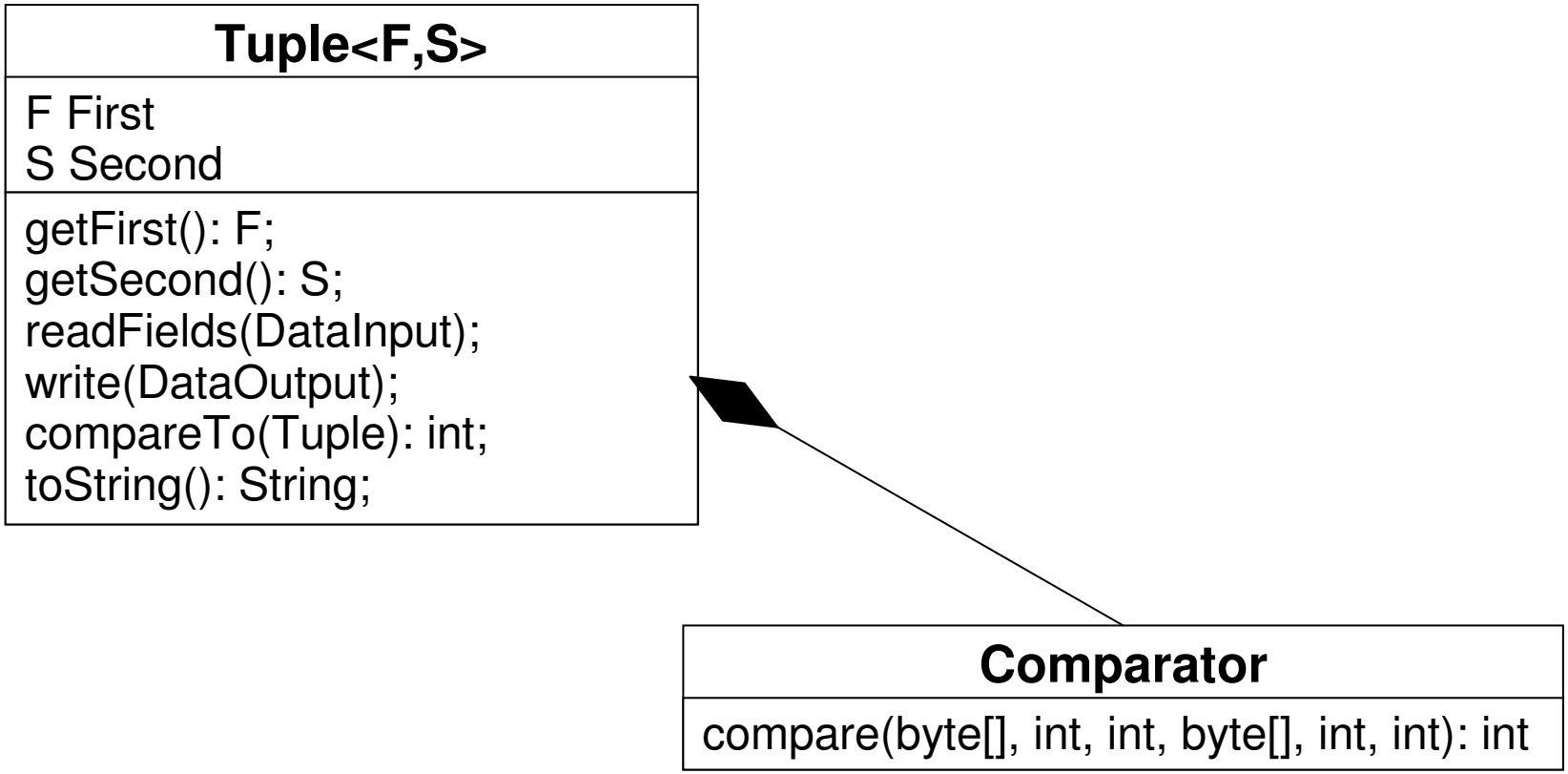
Hadoop – gewünschtes Verhalten

Wort	(DokID, tf)	
mental 0	**1.8**	
mental 1	(1, 0.4)	→ **mental**, (**1.8**, (1, 0.4), (3, 0.9), (5, 0.3), (0, 0.7), (2, 0.6))
mental 1	(3, 0.9)	
mental 1	(5, 0.3)	
mental 1	(0, 0.7)	
mental 1	(2, 0.6)	
retardation 0	**5.1**	
retardation 1	(8, 0.5)	→ **retardation**, (**5.1**, (8, 0.5), (6, 0.1), (9, 0.4), (4, 0.8))
retardation 1	(6, 0.1)	
retardation 1	(9, 0.4)	
retardation 1	(4, 0.8)	

- **Anhängen eines Index-Wertes an den Schlüssel**
- **Anpassen der Gruppierung**

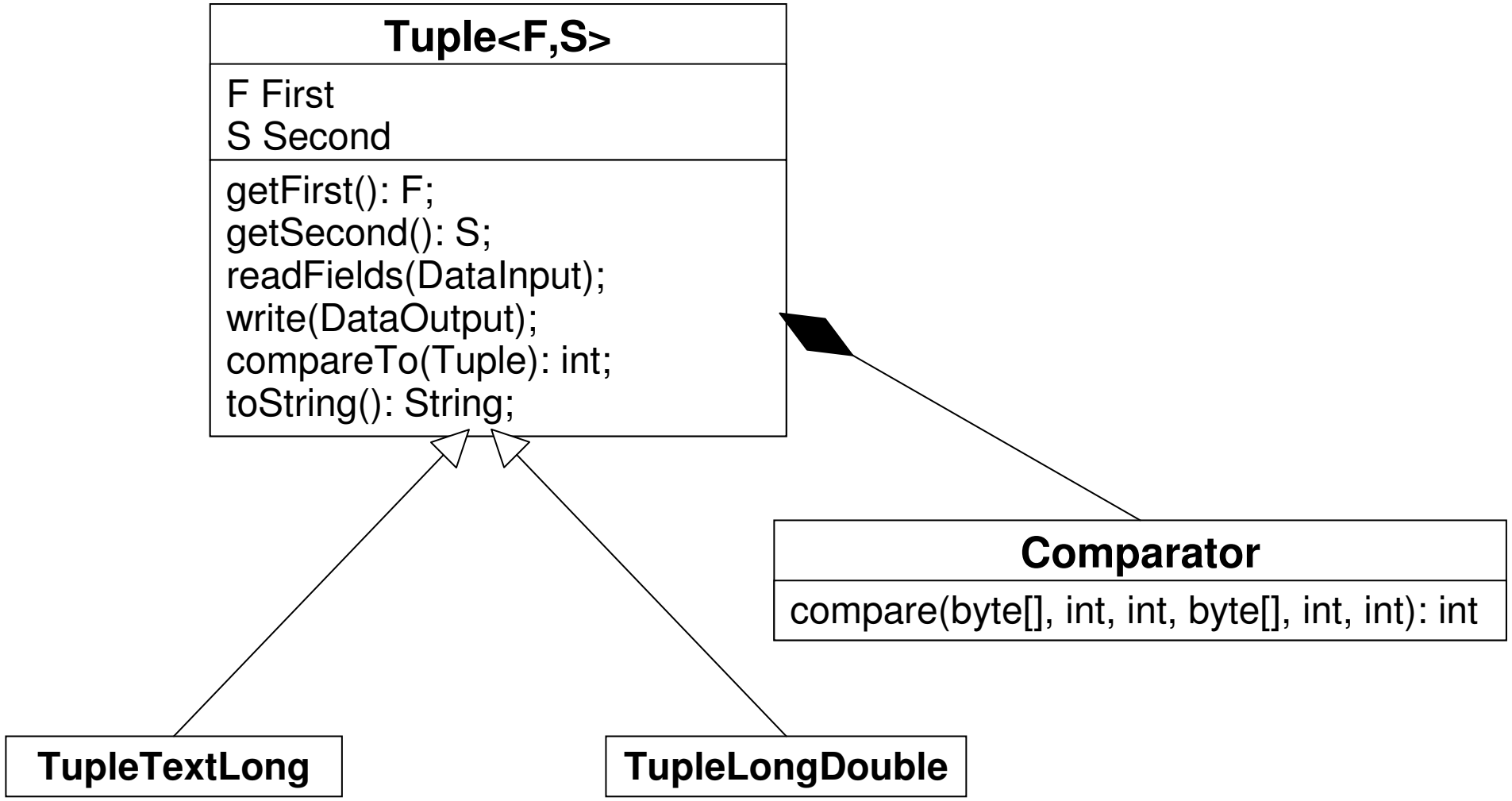
Implementierung – Eigene Datentypen

Tupel



Implementierung – Eigene Datentypen

Tupel



Speicherauslastung – 2 Phasen

- Phase 1
  - Map: konstant
  - Reduce: konstant – 1 long

- Phase 2
  - Map: 1 Dokument, pro Wort ein float
  - Reduce: | {Wort ∈ Korpus} | * (Wort, Long, Double)

  - Abschätzung Obergrenze:

    | {Wort ∈ Korpus} | = | Korpus |

    1’000’000 * (10+8+8 Byte) ≈ 25 MiB
    3 GiB RAM ≈ 3,2 Mrd. Dokumente

Speicherauslastung – 4 Phasen

- Phase 1
  - Map: konstant
  - Reduce: konstant – 1 long
- Phase 2
  - Map: 1 Dokument, 1 int
  - Reduce: konstant, 1 int
- Phase 3
  - Map: 1 Tupel (String, Long, Double)
  - Reduce: { (Wort, tf) } eines Dokuments
- Phase 4
  - Map: 1 Tupel (String, Long, Double)
  - Reduce: | {Wort ∈ Korpus} | * (String, Double)

Speicherauslastung – 5 Phasen

- Phase 1
  - Map: konstant
  - Reduce: konstant – 1 long
- Phase 2
  - Map: 1 Dokument, 1 int
  - Reduce: konstant, 1 int
- Phase 3
  - Map: 1 Tupel (String, Long, Long)
  - Reduce: 1 Tupel (String, Long, Long)
- Phase 4 & Phase 5
  - Map: 1 Tupel (String, Long, Double)
  - Reduce: 1 Tupel (String, Long, Double)

Obergrenze – 5 Phasen

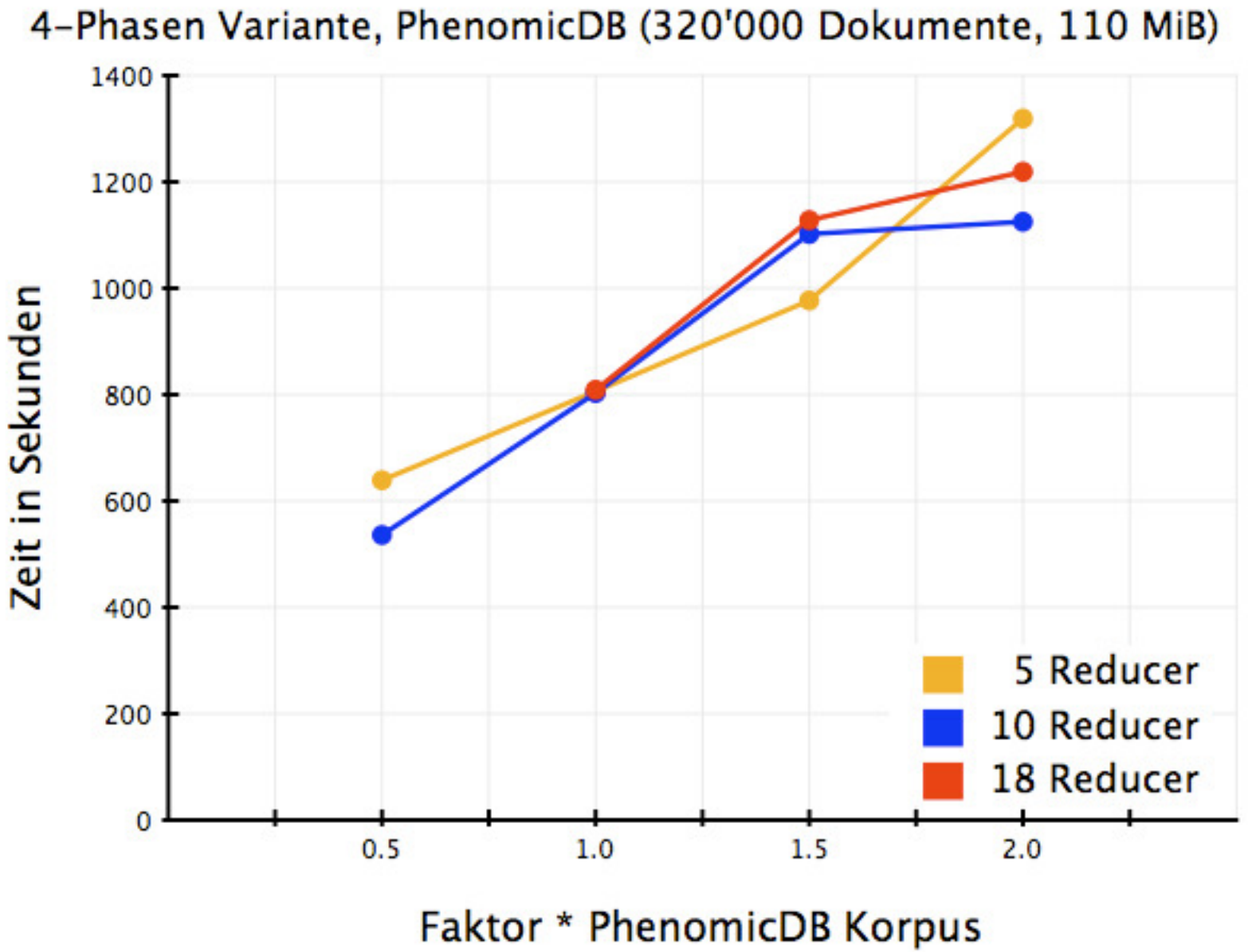
- durch Phase 2 beschränkt

- 1 Wort => max. INT_MAX Vorkommen im Korpus
- 1 Dokument => max. LONG_MAX Worte
- max. LONG_MAX Dokumente

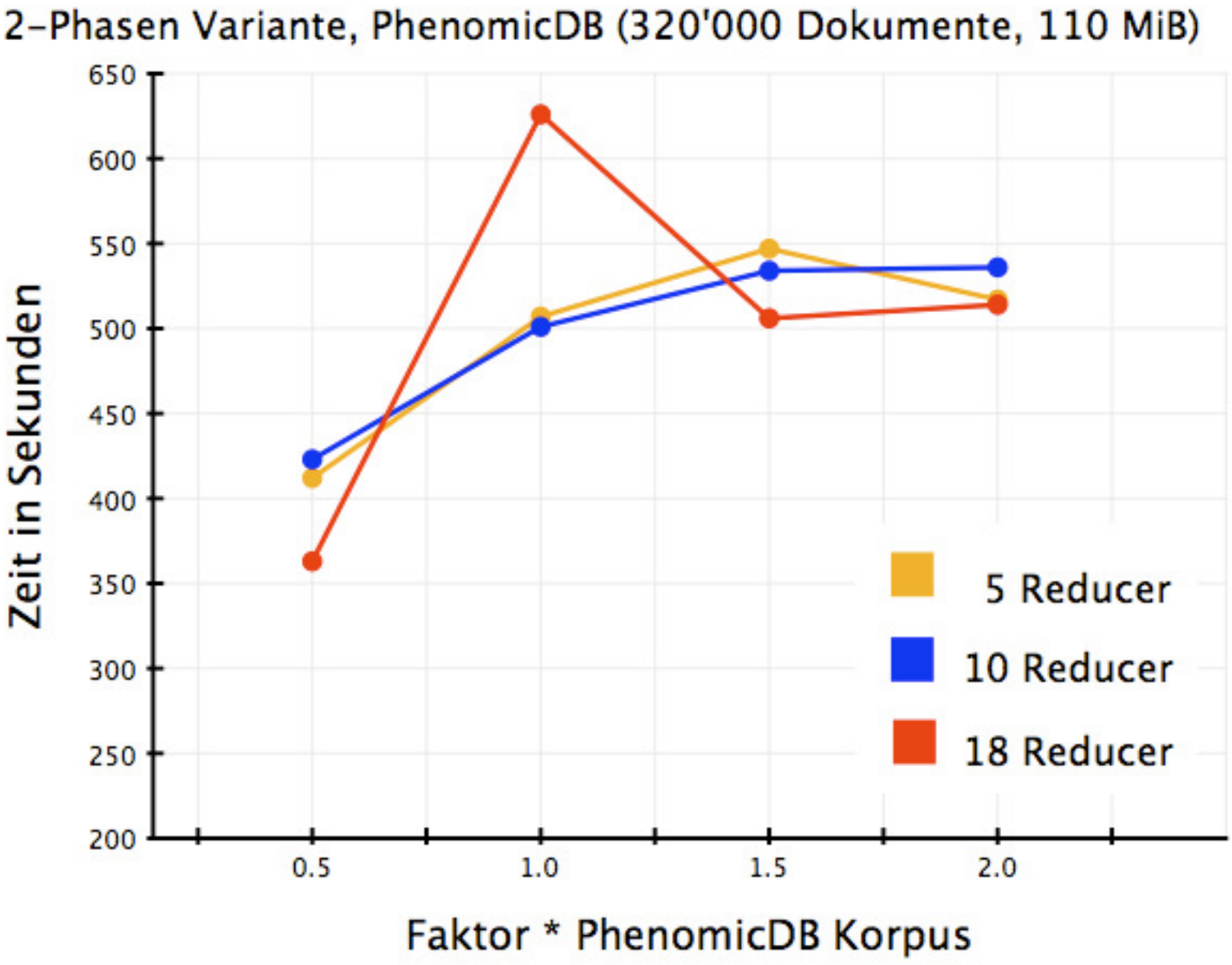
- Abschätzung: 10^19 * 10^19 * 10Byte = 10 ^ 39 Byte

=> unbegrenzt (10^80 Atome im Universum…)

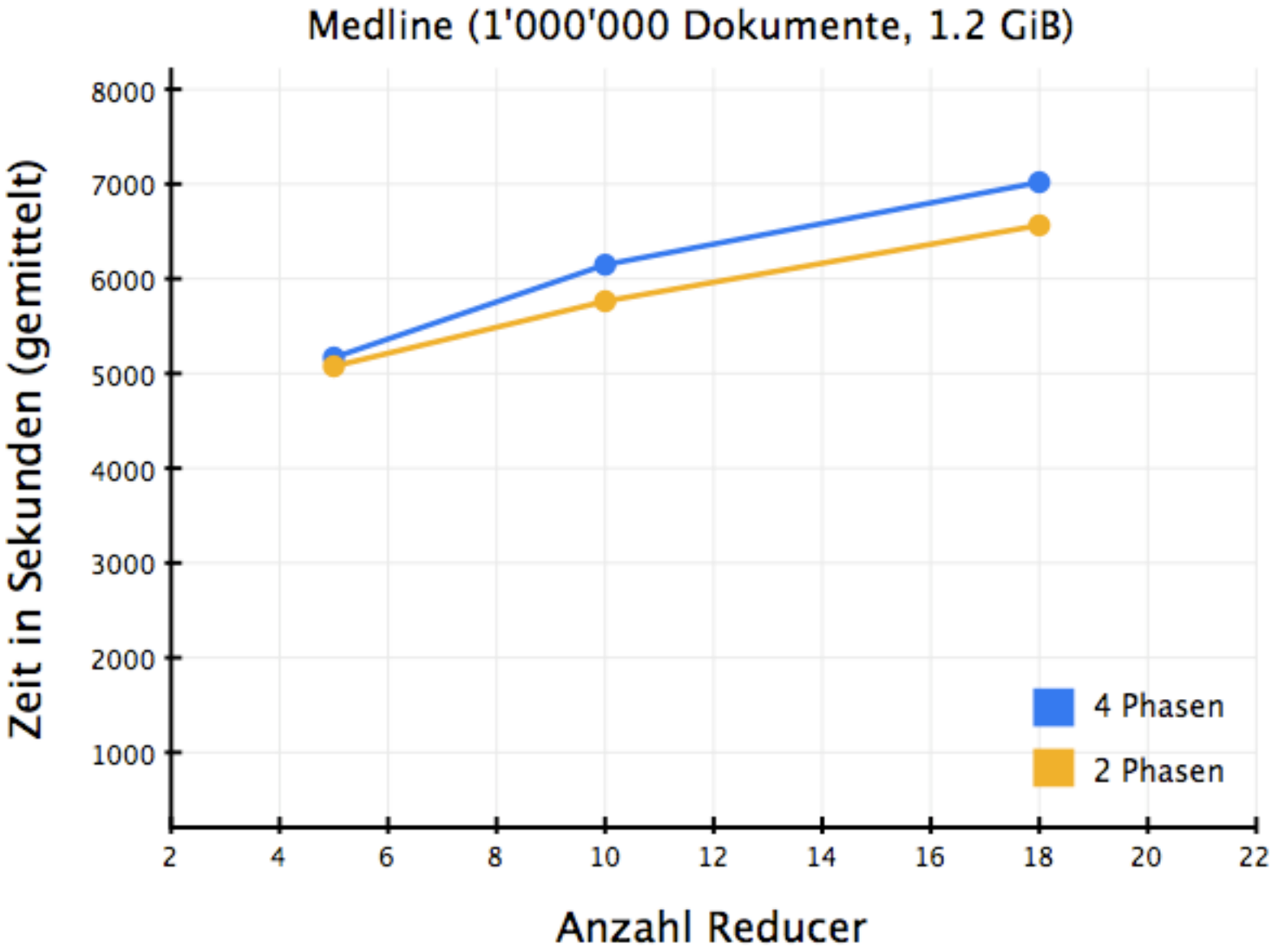
Laufzeit


4-Phasen Variante, PhenomicDB (320'000 Dokumente, 110 MiB)
Zeit in Sekunden
1400
1200
1000
800
600
400
200
0
0.5
1.0
1.5
2.0
Faktor * PhenomicDB Korpus
5 Reducer
10 Reducer
18 Reducer

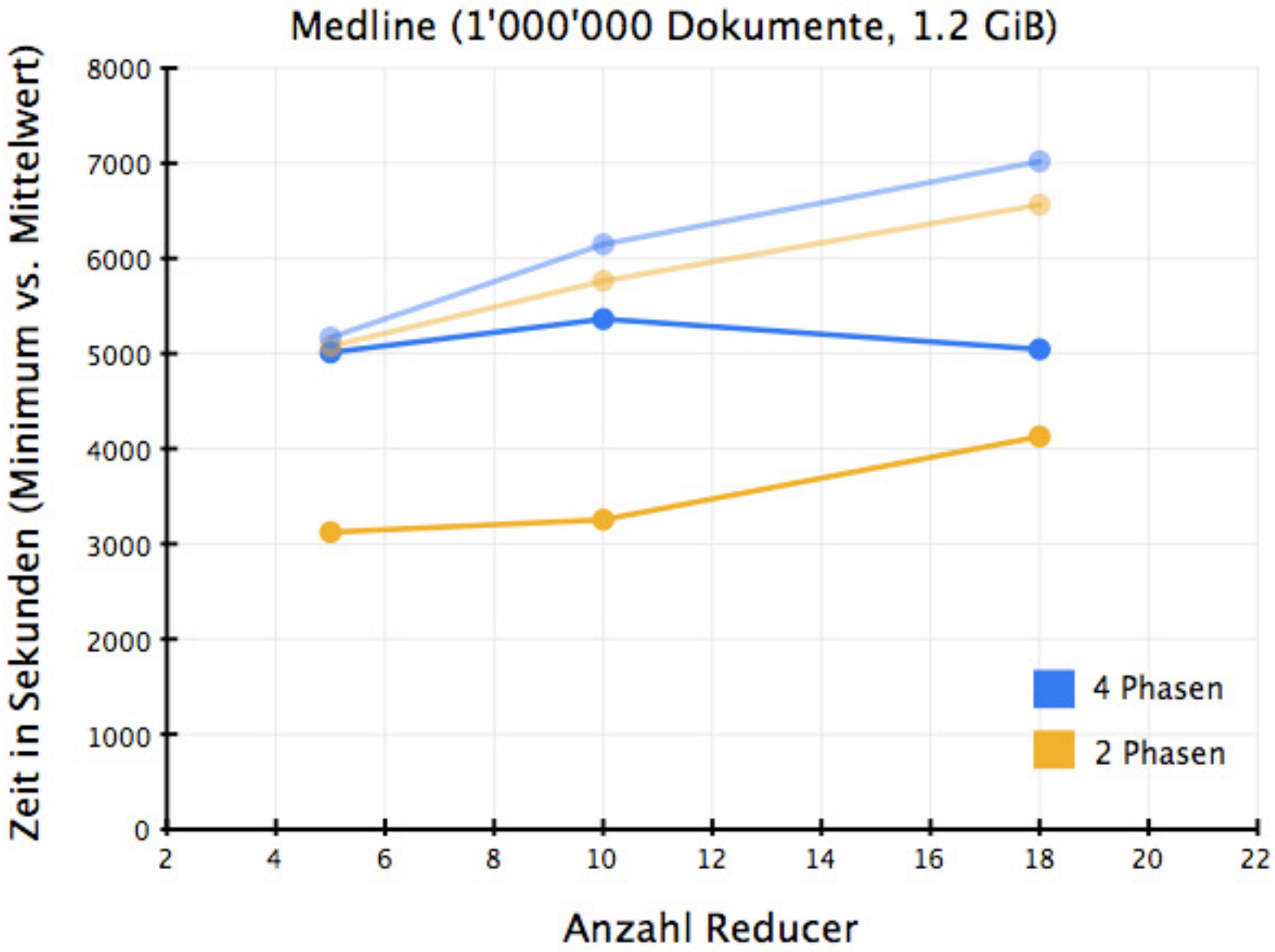
Laufzeit


2-Phasen Variante, PhenomicDB (320'000 Dokumente, 110 MiB)
Zeit in Sekunden
650
600
550
500
450
400
350
300
250
200
0.5
1.0
1.5
2.0
Faktor * PhenomicDB Korpus
5 Reducer
10 Reducer
18 Reducer

Laufzeit

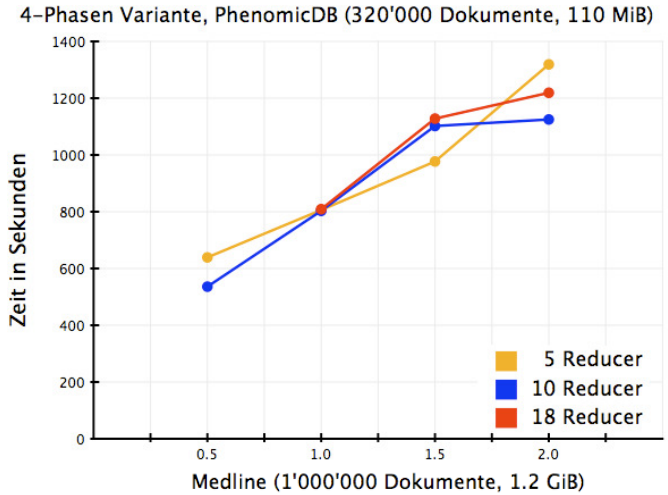

Medline (1'000'000 Dokumente, 1.2 GiB)
Zeit in Sekunden (gemittelt)
8000
7000
6000
5000
4000
3000
2000
1000
2
4
6
8
10
12
14
16
18
20
22
Anzahl Reducer
4 Phasen
2 Phasen

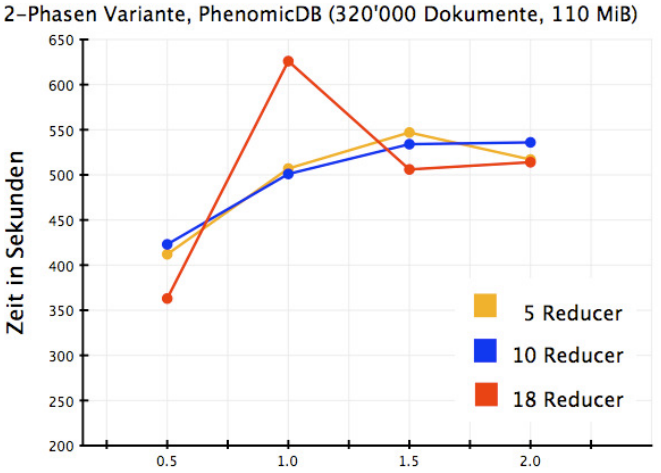
Laufzeit

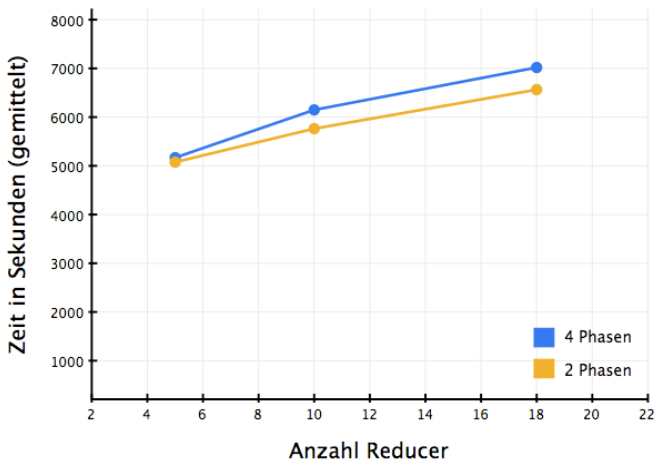

Medline (1'000'000 Dokumente, 1.2 GiB)
Zeit in Sekunden (Minimum vs. Mittelwert)
8000
7000
6000
5000
4000
3000
2000
1000
0
2
4
6
8
10
12
14
16
18
20
22
Anzahl Reducer
4 Phasen
2 Phasen

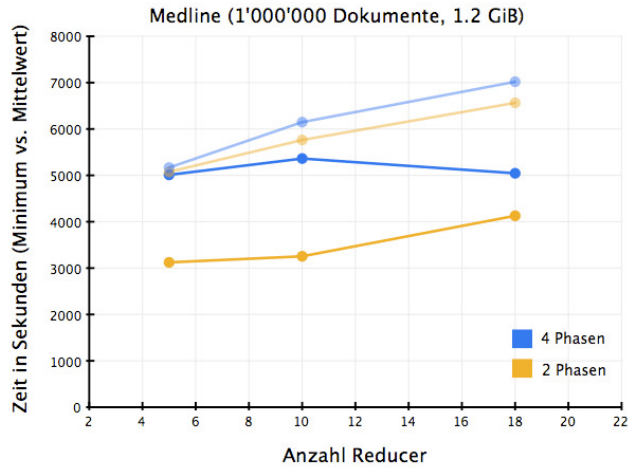
Fazit

- beide Varianten bei größeren Korpora anwendbar, max. O(N)
- Vergleich bei großen Korpora nötig (Wikipedia!)









Quellen

- Hadoop API Documentation
  http://hadoop.apache.org/core/docs/r0.20.0/api/
- Yahoo! Hadoop Tutorial
  http://developer.yahoo.com/hadoop/tutorial/
- Wikipedia
  http://en.wikipedia.org/wiki/Tf-idf
- Introduction to Information Retrieval
  http://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html