

---

# An Incrementally Maintainable Index for Approximate Lookups in Hierarchical Data

**Nikolaus Augsten<sup>a</sup>, Michael Böhlen, Johann Gamper**

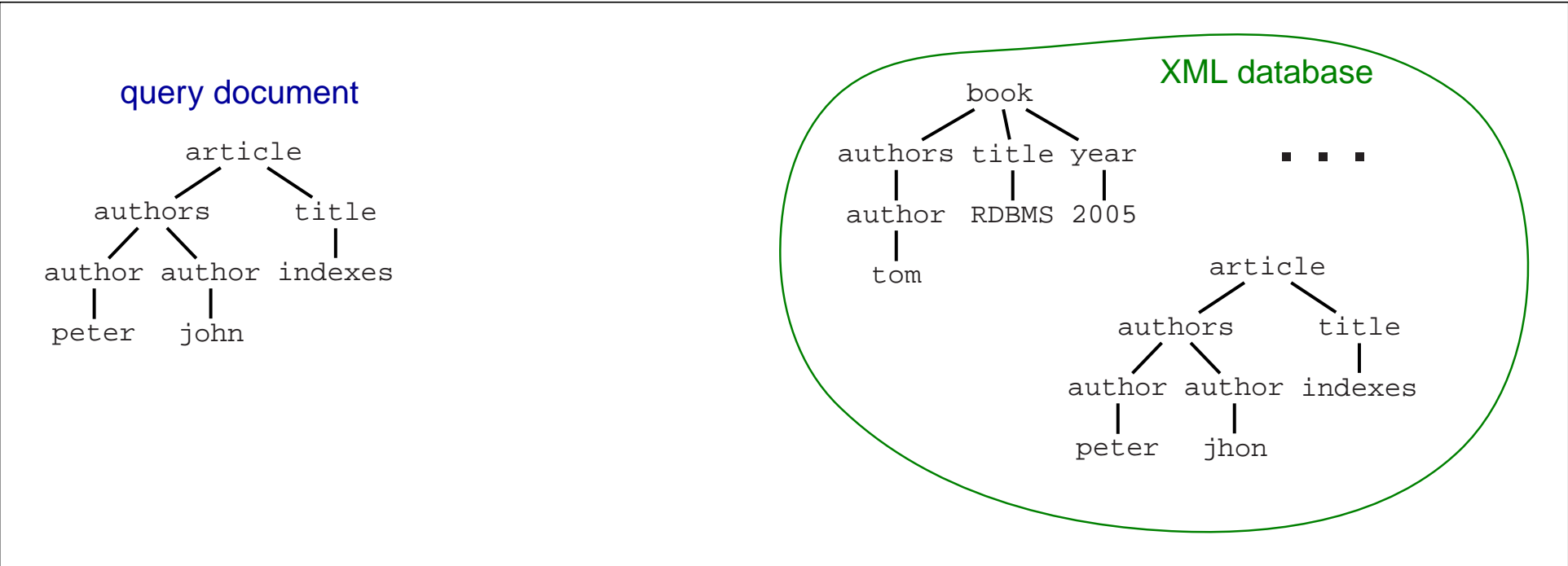
DIS - Center for Database and Information Systems

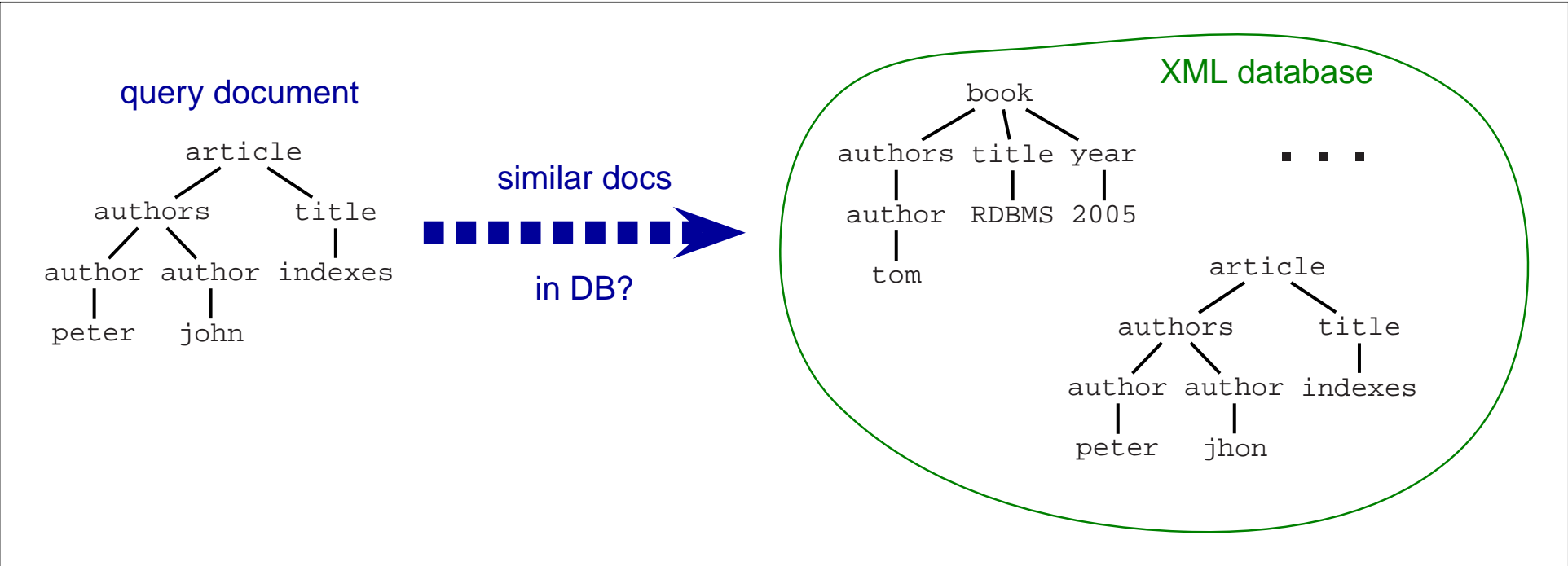
Free University of Bozen-Bolzano, Italy

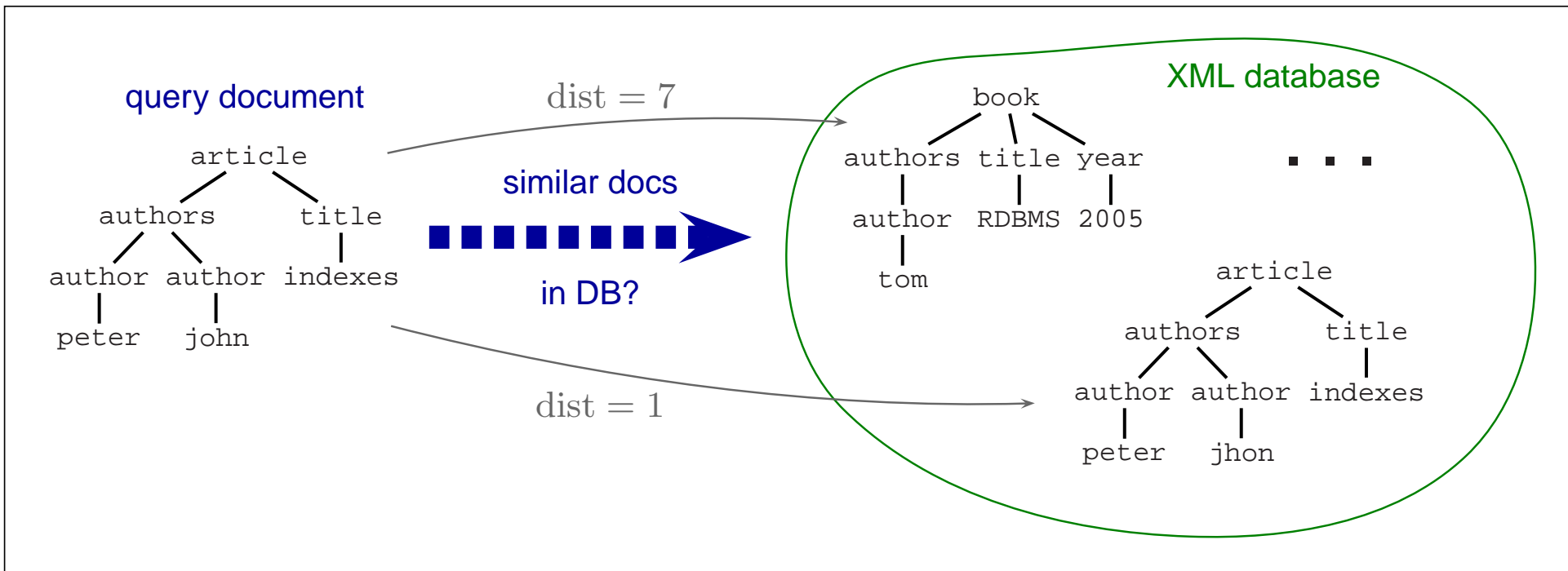
[www.inf.unibz.it](http://www.inf.unibz.it)

---

<sup>a</sup>Supported by the Municipality of Bozen-Bolzano.

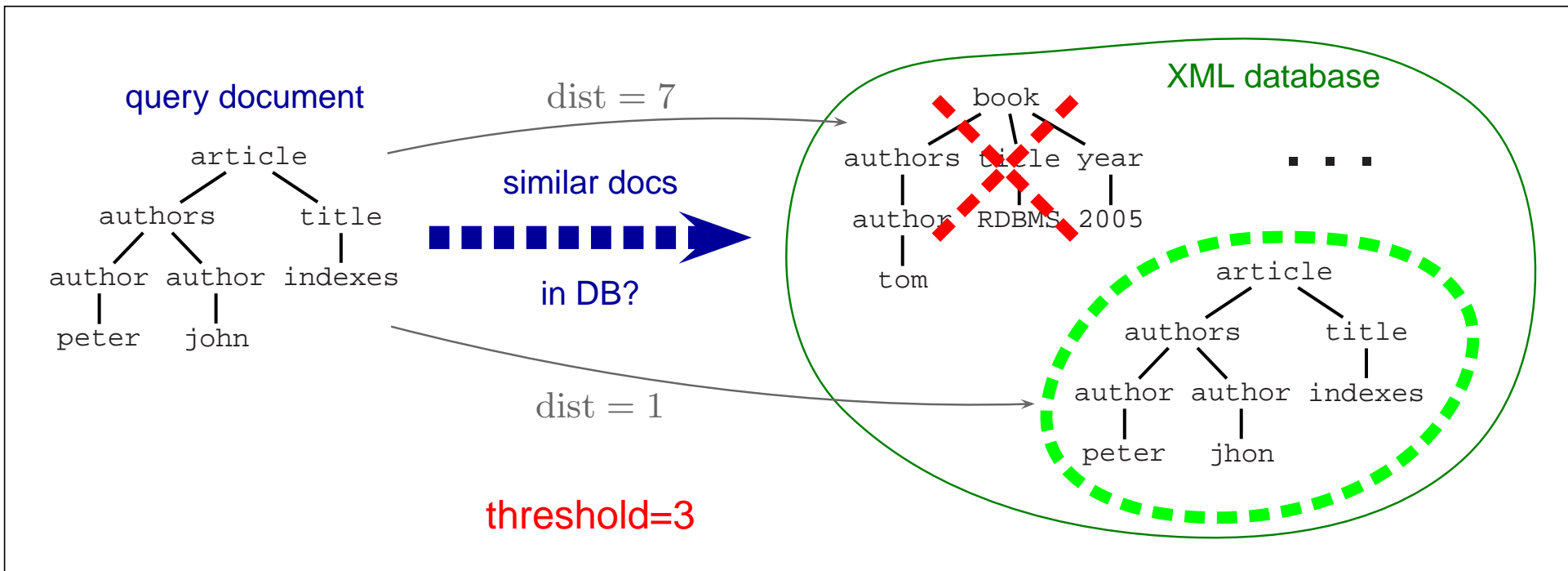






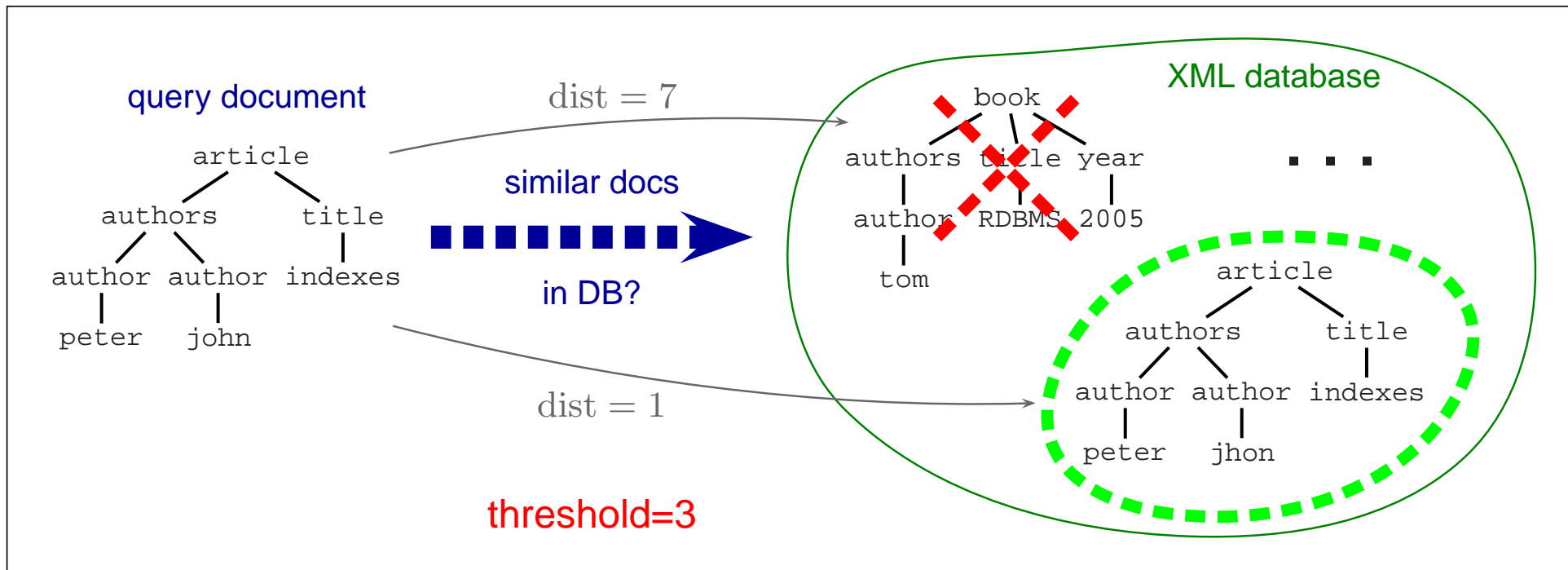
👉 **Simple approach:**

1. compute **distance** between the **query document** and **each document in DB**



## 👉 Simple approach:

1. compute **distance** between the **query document** and **each document in DB**
2. choose documents within **threshold**

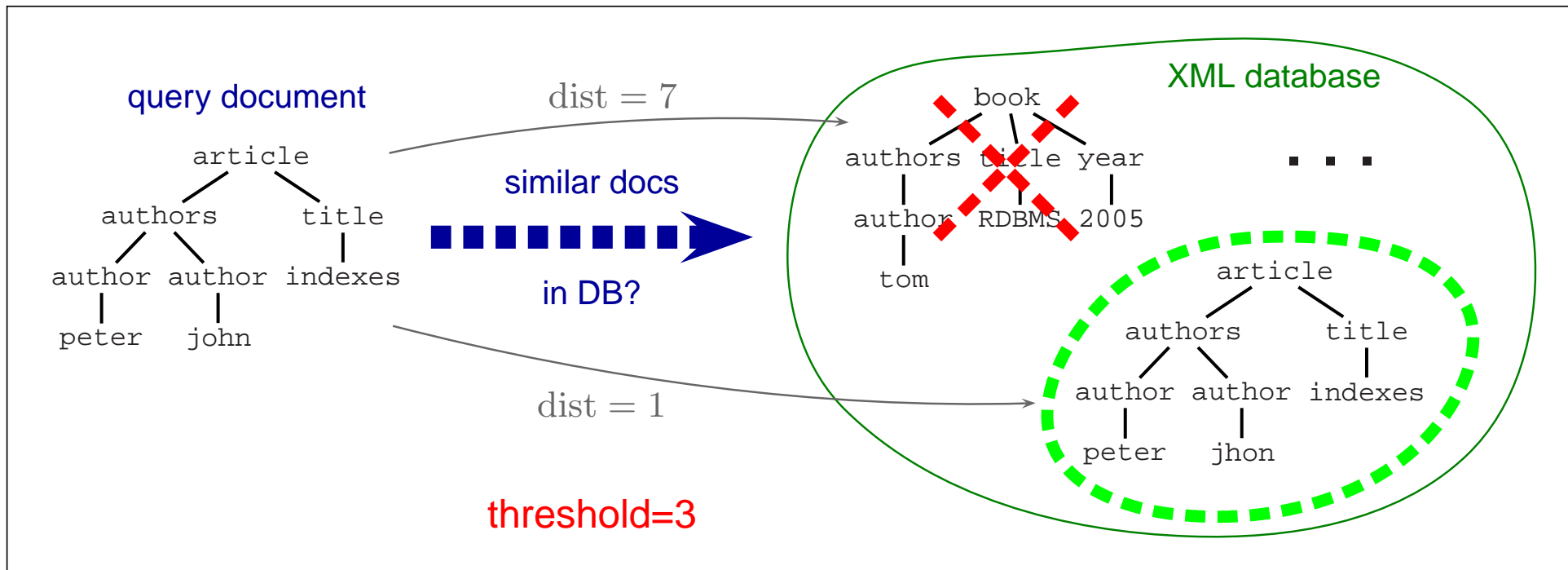


## 👉 Simple approach:

1. compute **distance** between the **query document** and **each document in DB**
2. choose documents within **threshold**

## 👉 Very expensive

- ▣➡ scan whole DB
- ▣➡ compute distance to each XML document



## 👉 Simple approach:

1. compute **distance** between the **query document** and **each document in DB**
2. choose documents within **threshold**

## 👉 Very expensive

- ▣➡ scan whole DB
- ▣➡ compute distance to each XML document

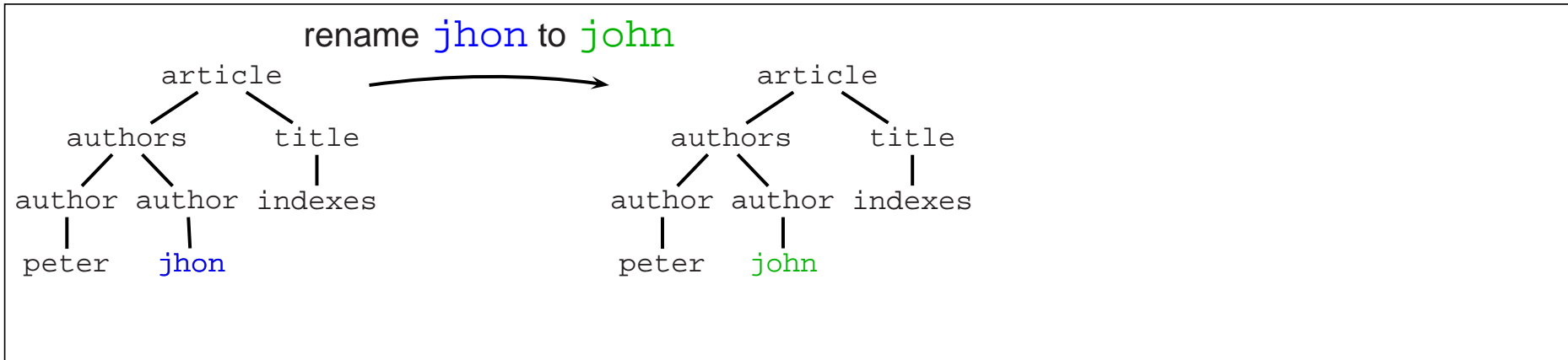
## 👉 **Index** for approximate lookups

👉 **Documents** in database change



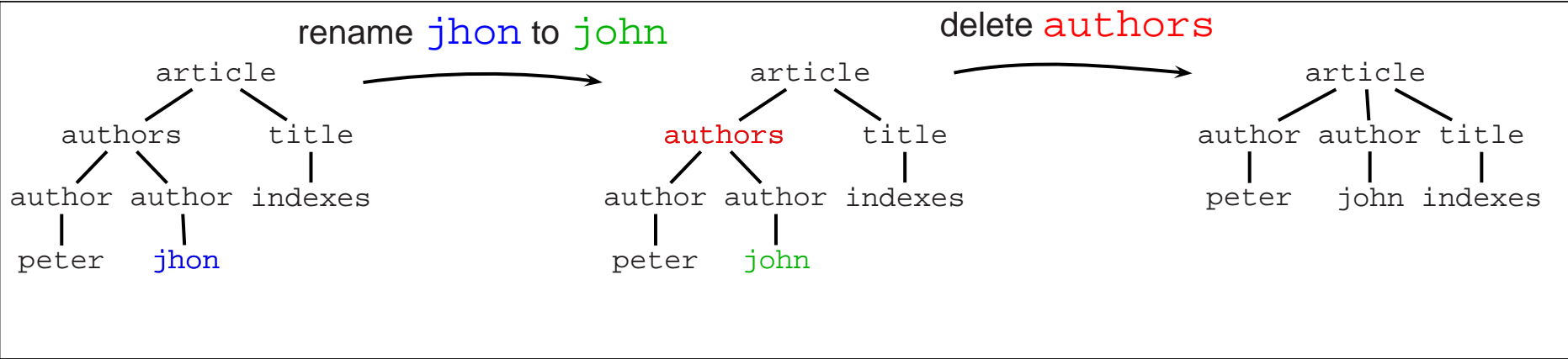
☞ **Documents** in database **change**

▣ changes are **node edit operations**: rename, delete, insert



👉 **Documents in database change**

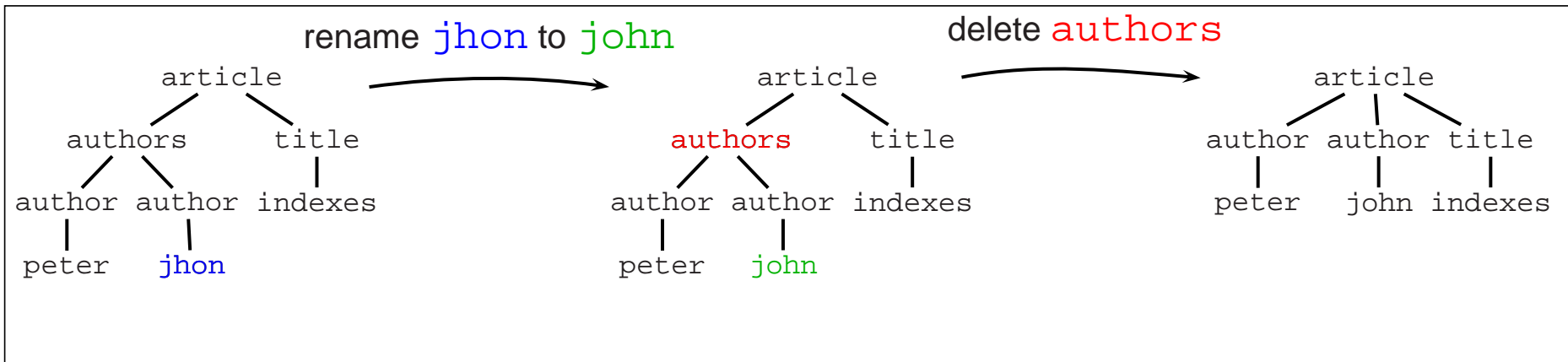
▣ changes are **node edit operations**: rename, delete, insert



☞ **Documents** in database **change**

▣ changes are **node edit operations**: rename, delete, insert

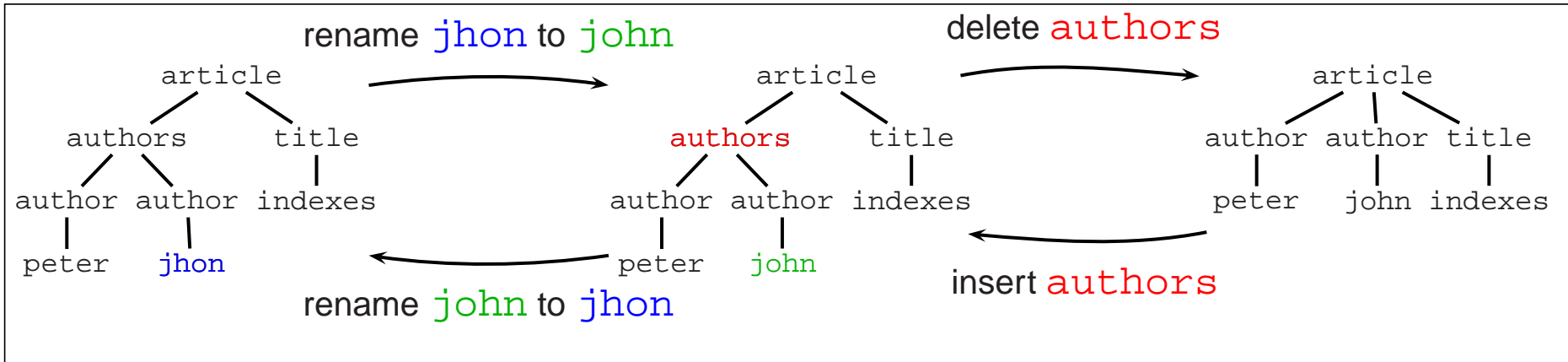
▣ **inverse** edit operation **undoes** the edit operation



☞ **Documents** in database change

▣ changes are **node edit operations**: rename, delete, insert

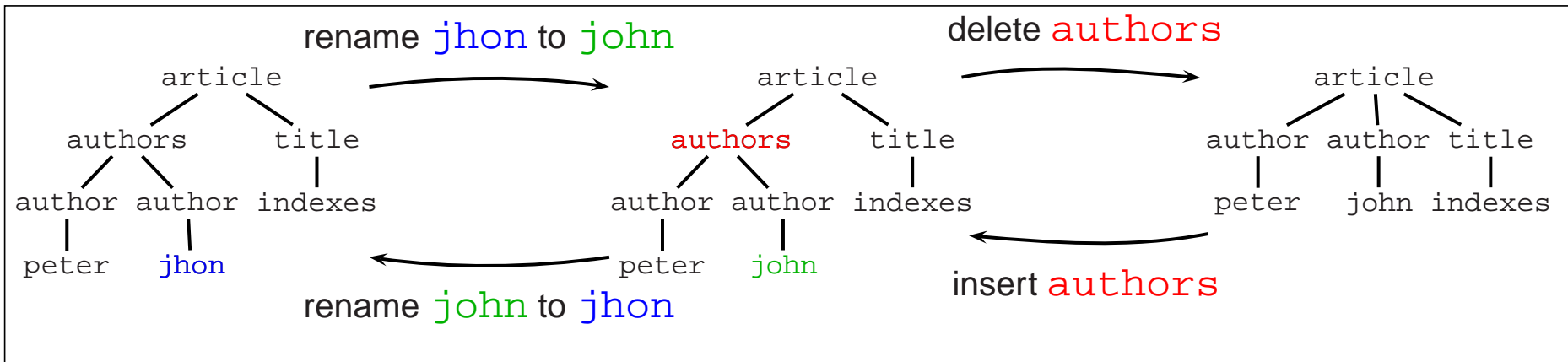
▣ **inverse** edit operation **undoes** the edit operation



👉 **Documents in database change**

▣ changes are **node edit operations**: rename, delete, insert

▣ **inverse** edit operation **undoes** the edit operation

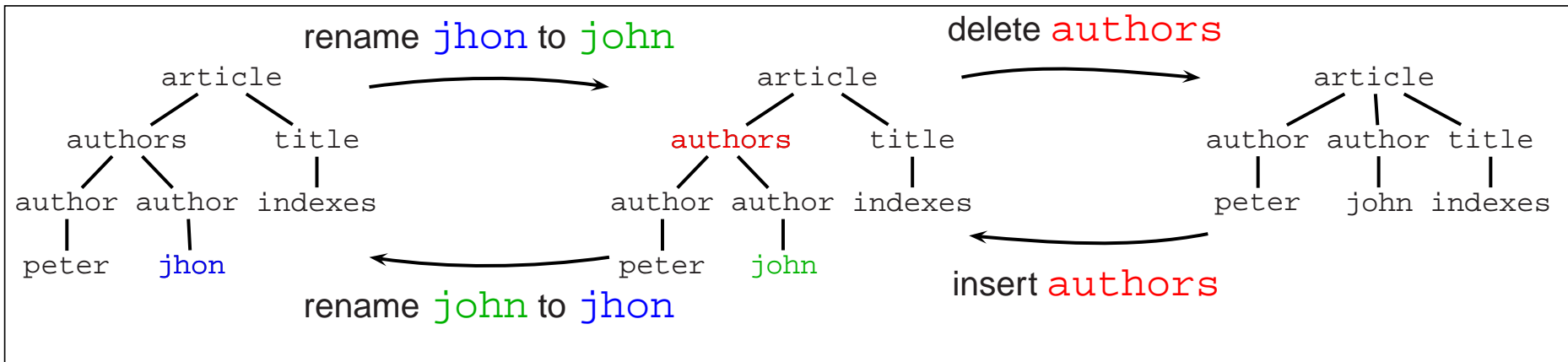


👉 **Index update required**

👉 **Documents** in database **change**

▣ changes are **node edit operations**: rename, delete, insert

▣ **inverse** edit operation **undoes** the edit operation



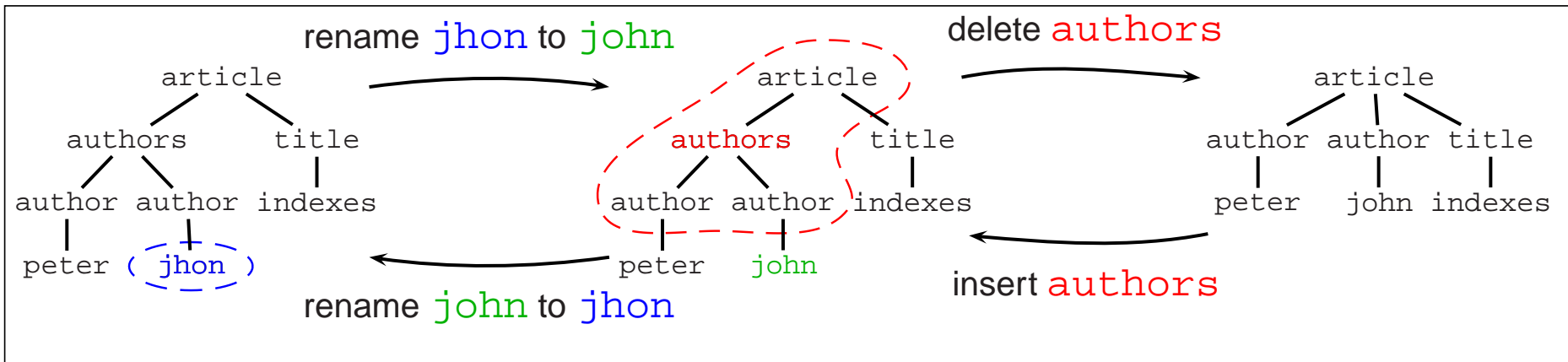
👉 **Index update** required

👉 **Problem**: computation **from scratch expensive** (> 3 hours for DBLP)

👉 **Documents in database change**

▣ changes are **node edit operations**: rename, delete, insert

▣ **inverse** edit operation **undoes** the edit operation



👉 **Index update required**

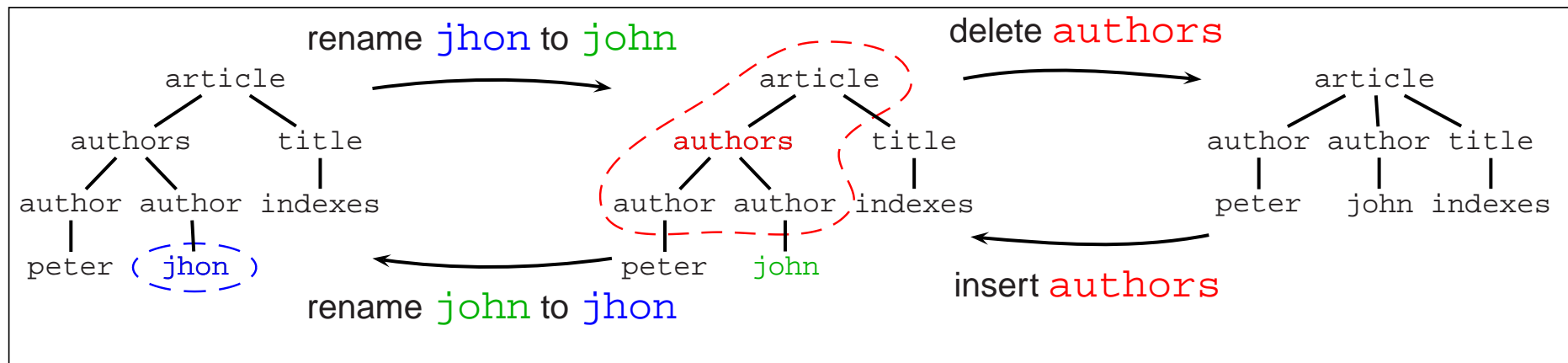
👉 **Problem:** computation **from scratch expensive** (> 3 hours for DBLP)

👉 Only **small subtrees change...**

👉 **Documents in database change**

▣ changes are **node edit operations**: rename, delete, insert

▣ **inverse** edit operation **undoes** the edit operation

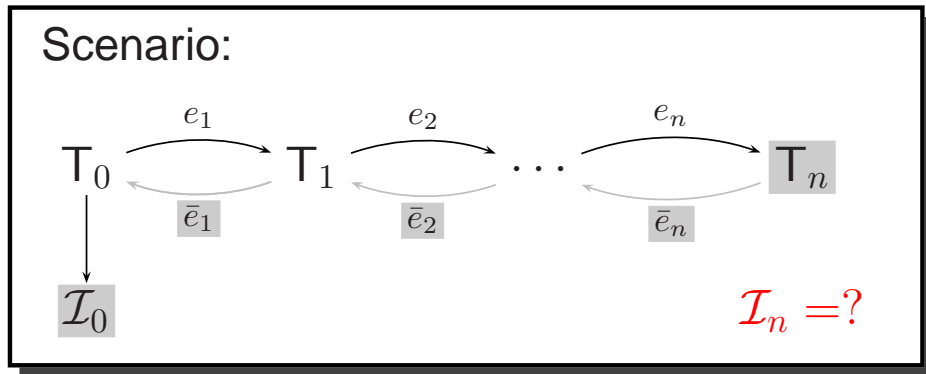


👉 **Index update** required

👉 **Problem:** computation **from scratch expensive** (> 3 hours for DBLP)

👉 Only **small subtrees change...**

👉 **Solution:** incrementally **update changing parts** in index

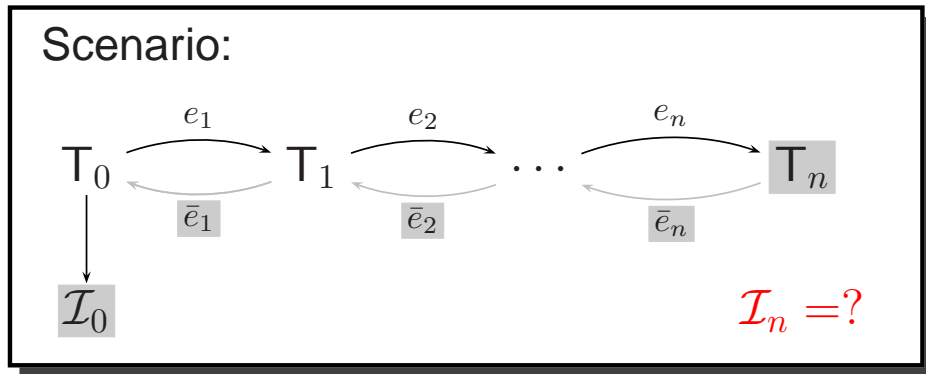


☞ **Update index** based on

- ▣ old index  $\mathcal{I}_0$
- ▣ log of inverse edit operations  $(\bar{e}_1, \dots, \bar{e}_n)$
- ▣ resulting tree  $T_n$

☞ **Do not compute  $\mathcal{I}_n$  from scratch.**

☞ **Do not compute intermediate trees.**



➡ **Update index** based on

- ➡ old index  $\mathcal{I}_0$
- ➡ log of inverse edit operations  $(\bar{e}_1, \dots, \bar{e}_n)$
- ➡ resulting tree  $T_n$

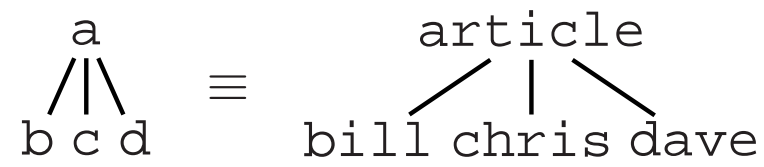
➡ **Do not compute  $\mathcal{I}_n$  from scratch.**

➡ **Do not compute intermediate trees.**

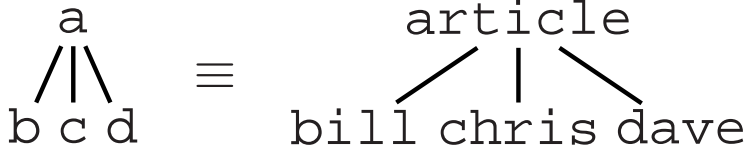
➡ **Example:**

- ➡ index for DBLP
- ➡ 1000 updates
- ➡ incrementally update index!

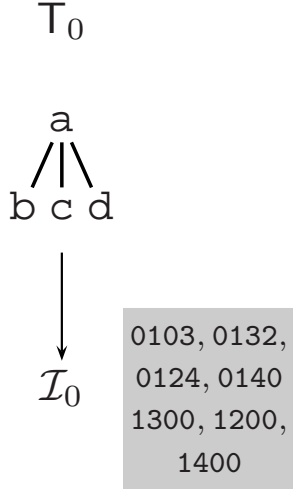
Example tree:



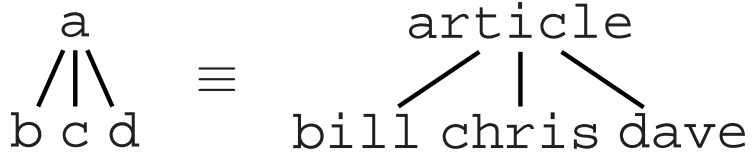
Example tree:



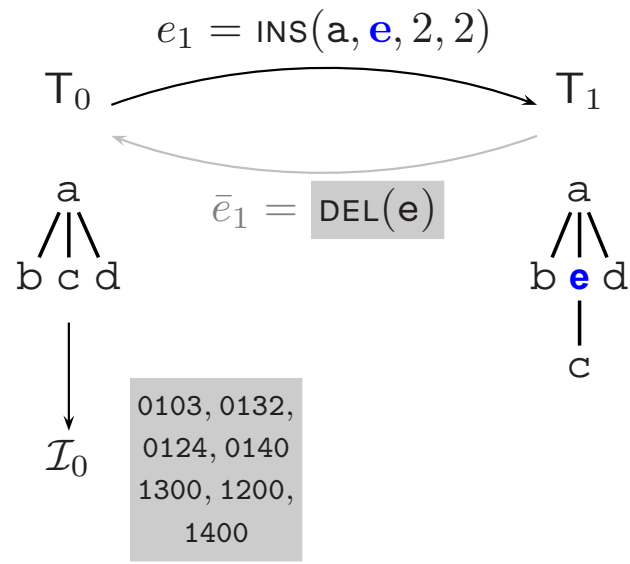
Example Scenario:



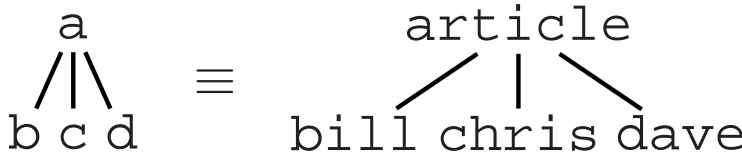
Example tree:



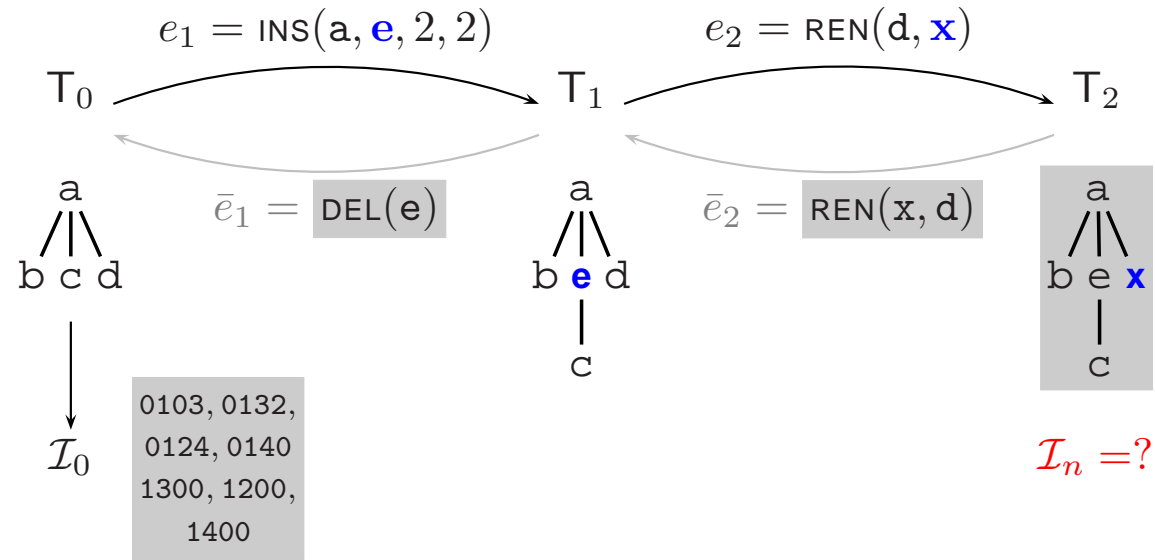
Example Scenario:



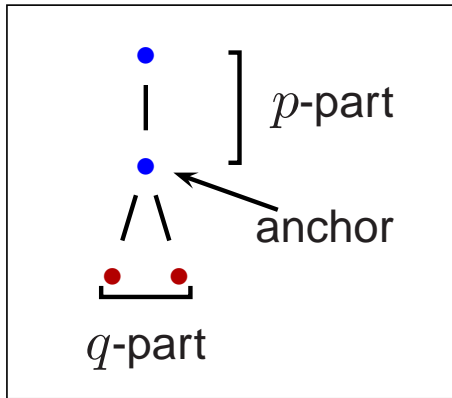
Example tree:



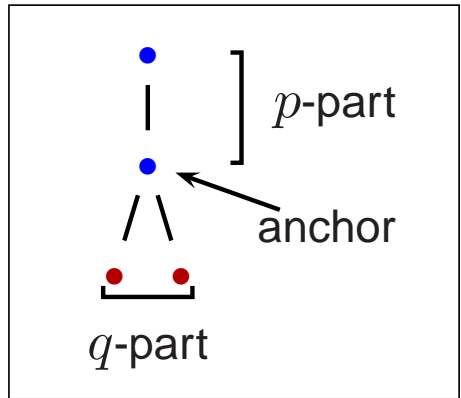
Example Scenario:



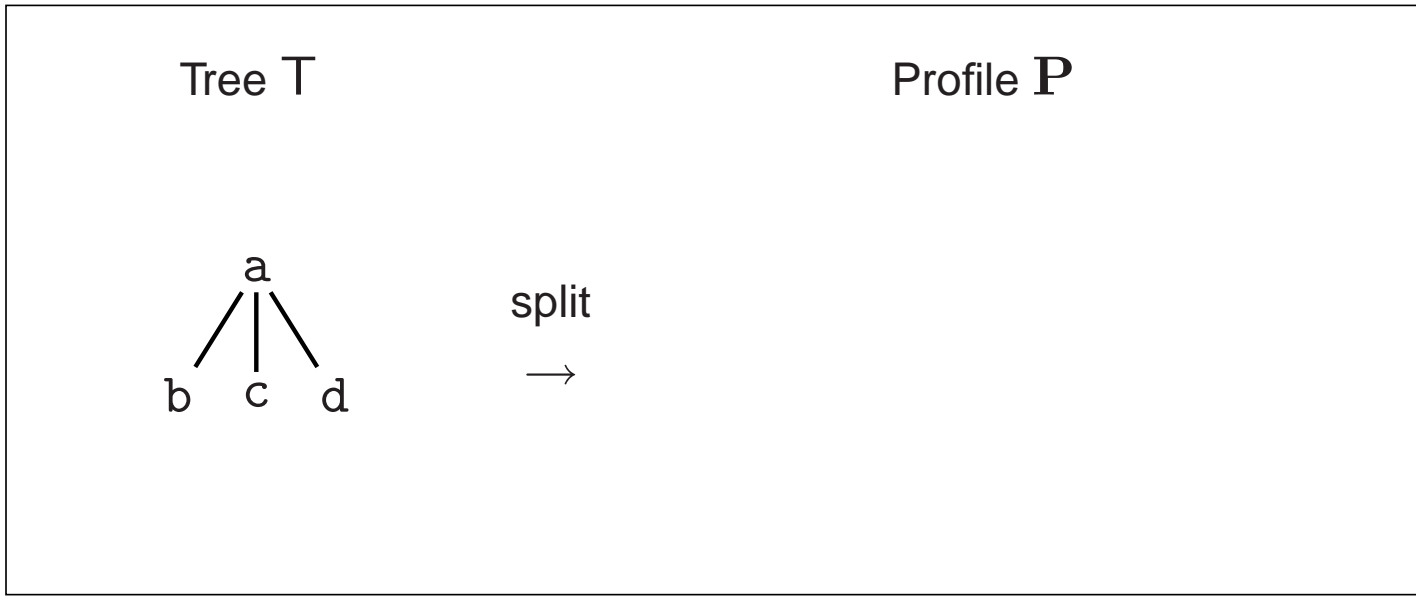
☞ *pq*-Gram pattern:



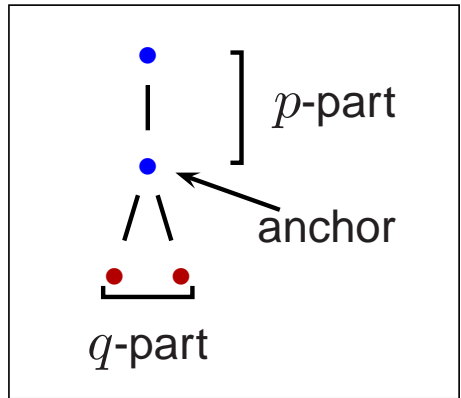
☞ pq-Gram pattern:



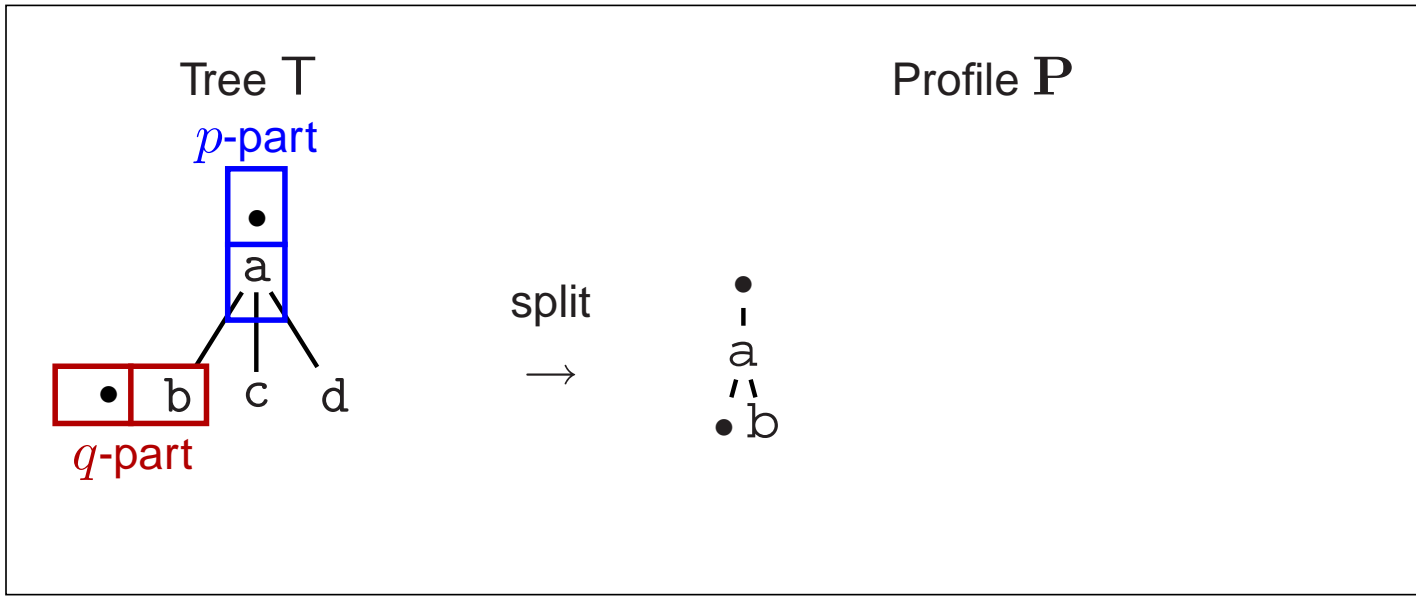
(1) Splitting the tree into 2,2-grams:



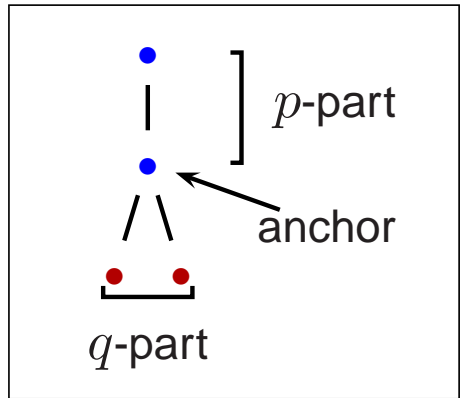
👉 pq-Gram pattern:



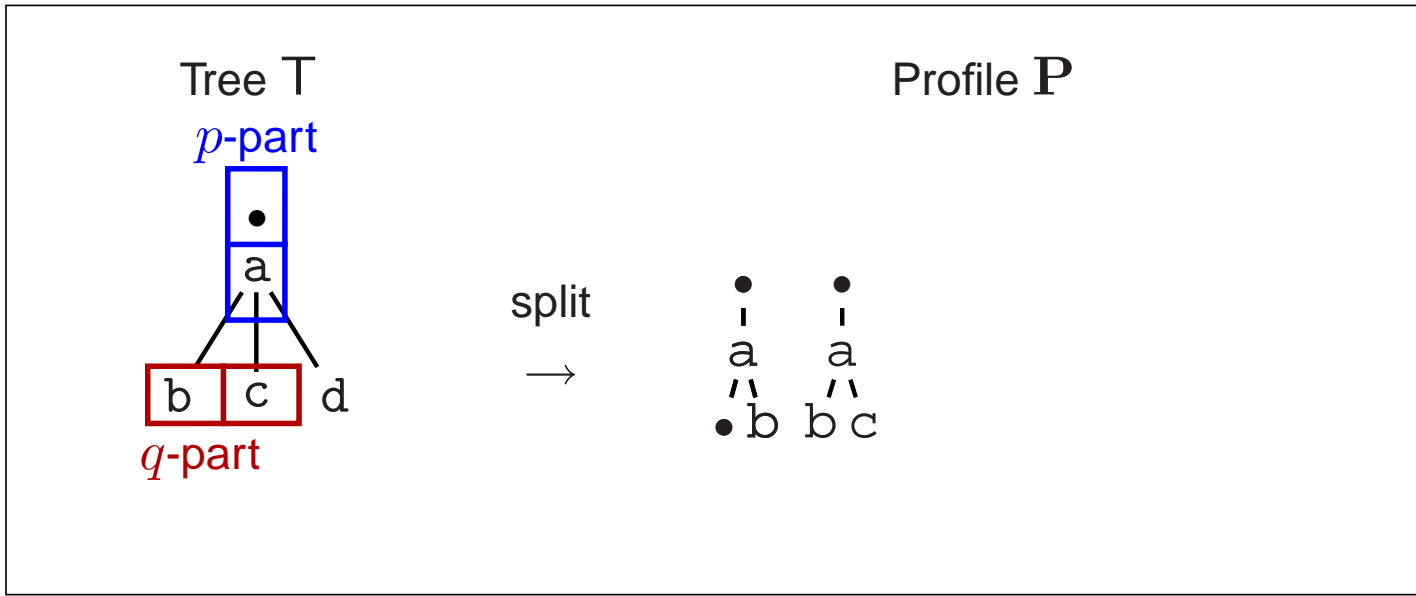
(1) Splitting the tree into 2,2-grams:



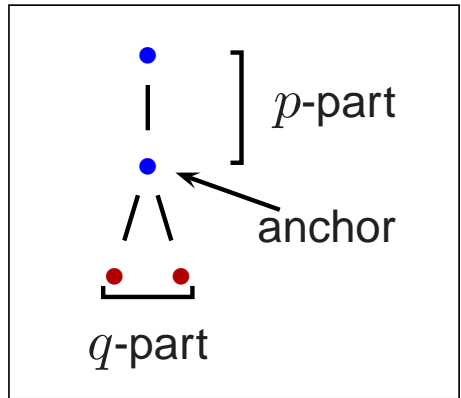
👉 pq-Gram pattern:



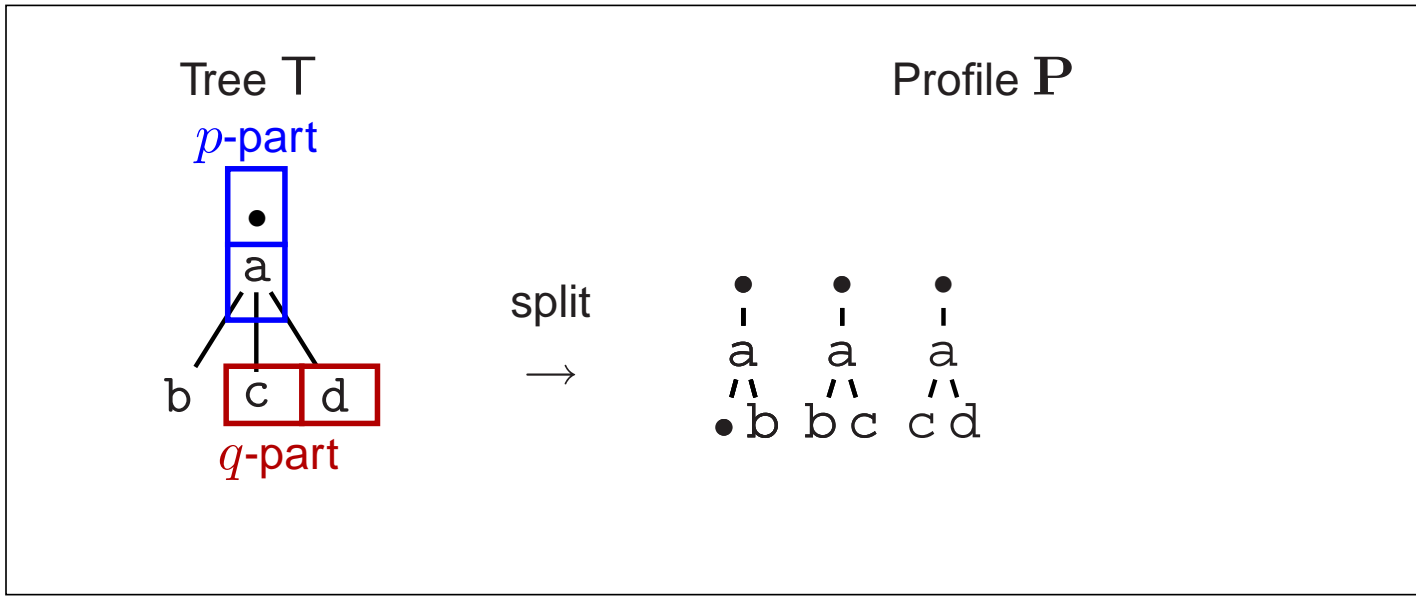
(1) Splitting the tree into 2,2-grams:



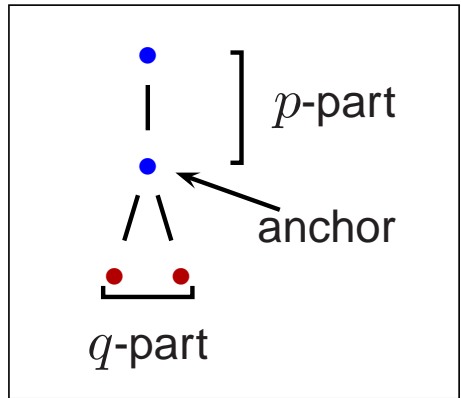
👉 pq-Gram pattern:



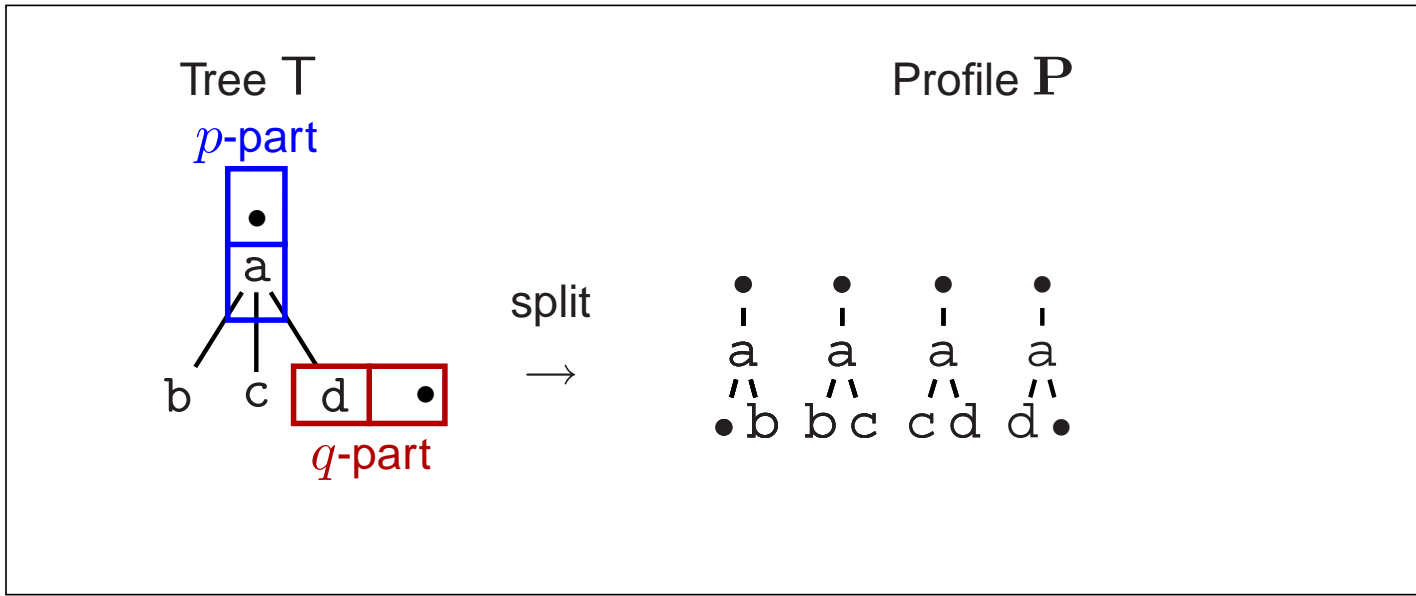
(1) Splitting the tree into 2,2-grams:



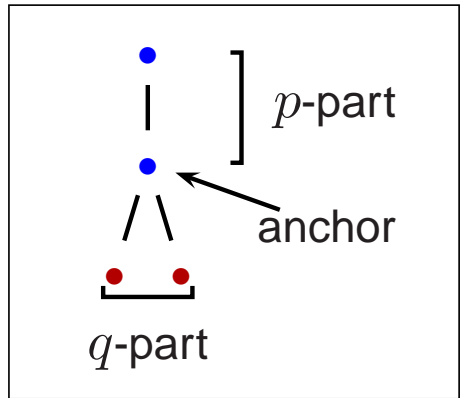
👉 pq-Gram pattern:



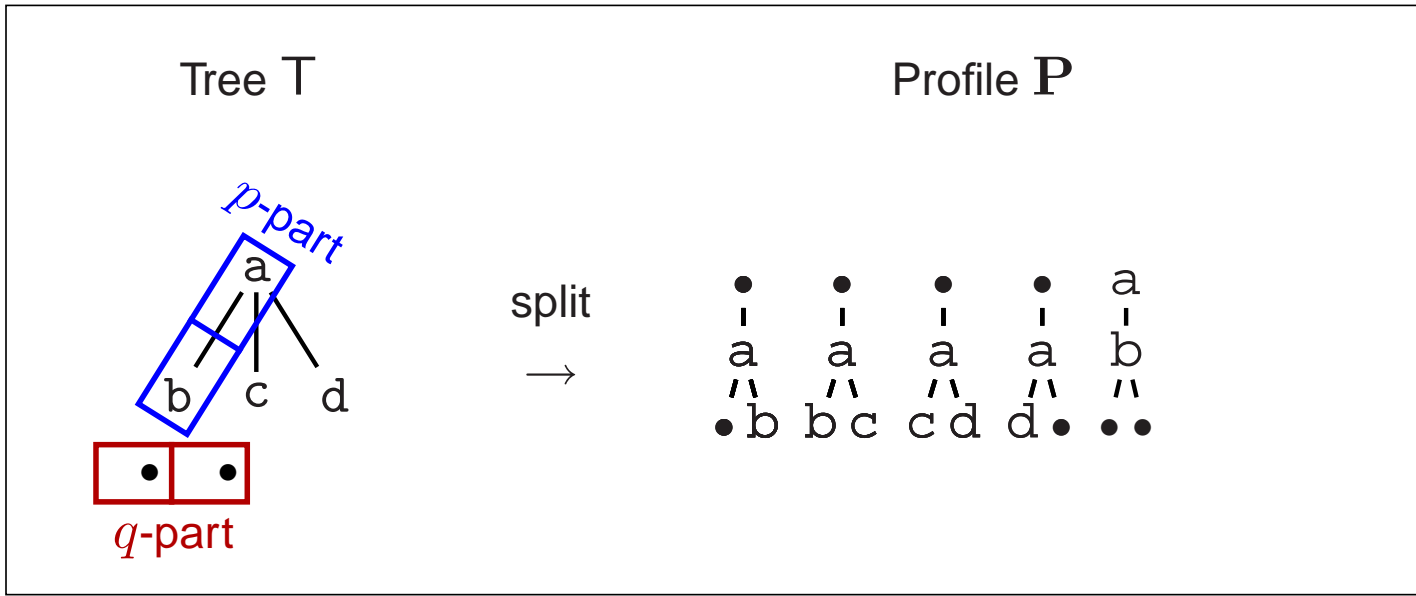
(1) Splitting the tree into 2,2-grams:



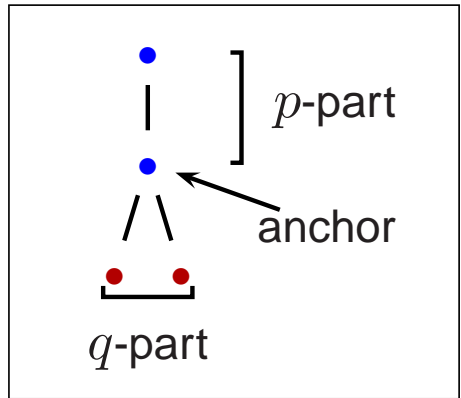
👉 pq-Gram pattern:



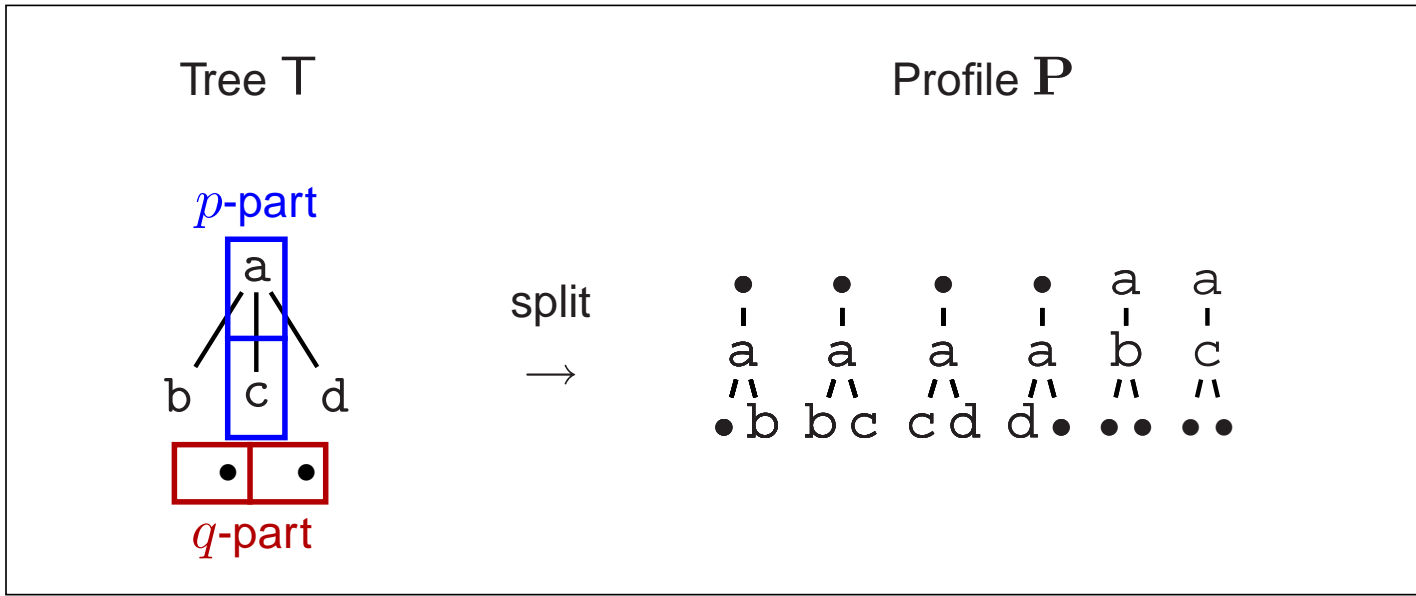
(1) Splitting the tree into 2,2-grams:



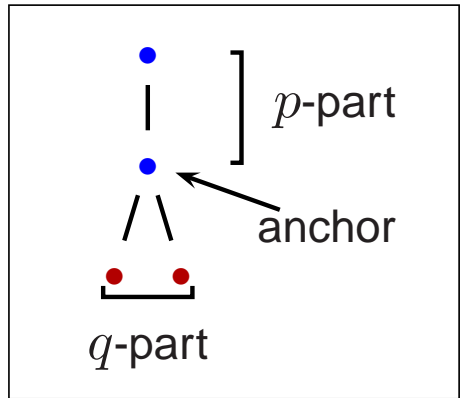
👉 pq-Gram pattern:



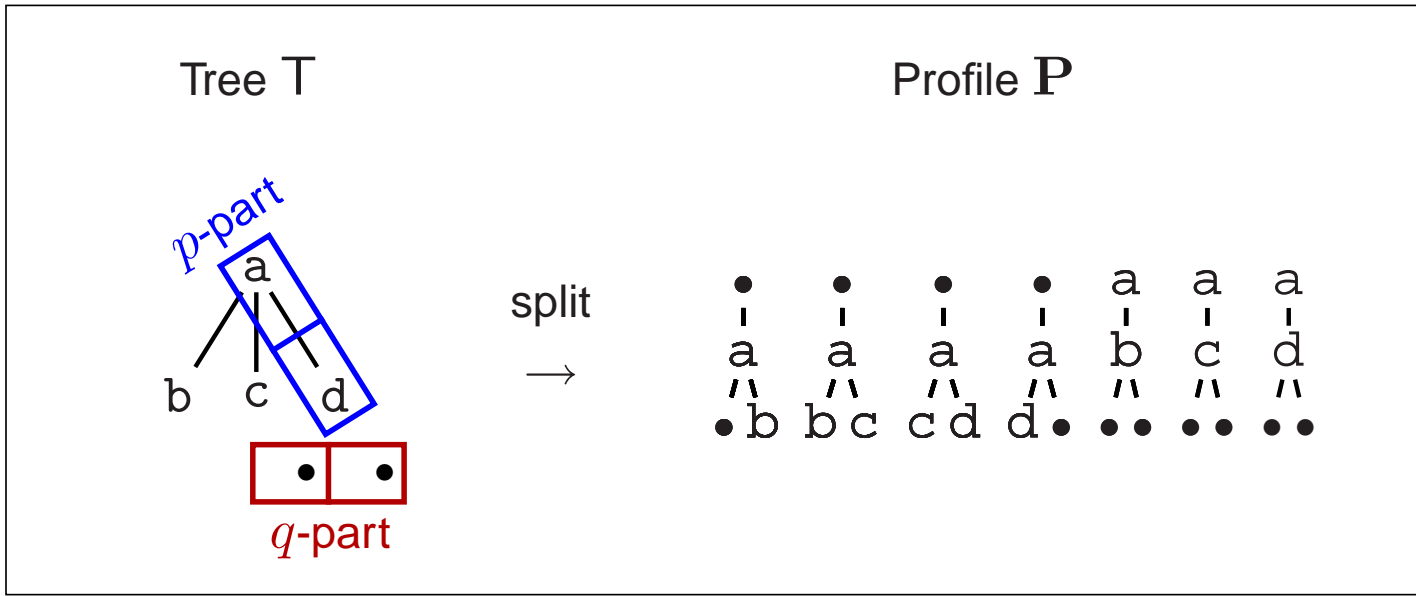
(1) Splitting the tree into 2,2-grams:



👉 pq-Gram pattern:



(1) Splitting the tree into 2,2-grams:

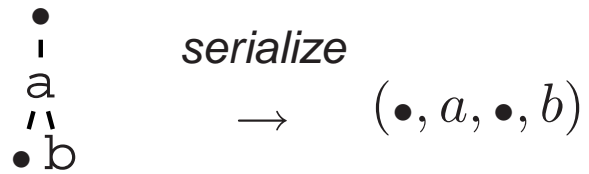


(2) Preorder serialization: *pq*-gram  $\rightarrow$  serialized *pq*-gram

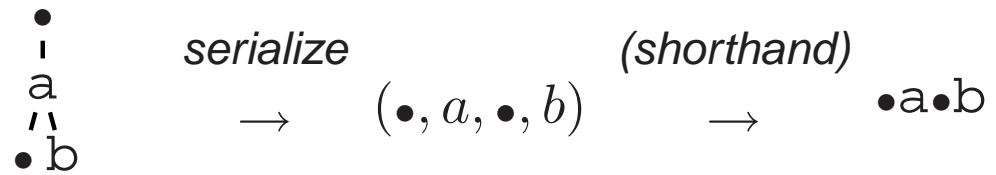
(2) Preorder serialization: *pq*-gram  $\rightarrow$  serialized *pq*-gram



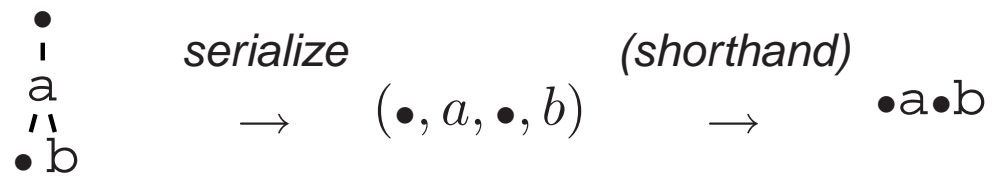
(2) Preorder serialization:  $pq$ -gram  $\rightarrow$  serialized  $pq$ -gram



(2) Preorder serialization:  $pq$ -gram  $\rightarrow$  serialized  $pq$ -gram

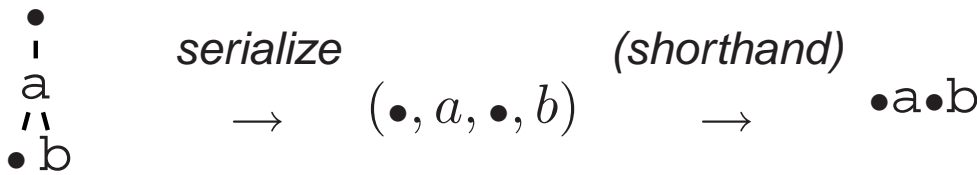


(2) **Preorder serialization:**  $pq$ -gram  $\rightarrow$  serialized  $pq$ -gram



(3) **Hashing:** serialized  $pq$ -gram  $\rightarrow$  hash string

(2) Preorder serialization: pq-gram → serialized pq-gram



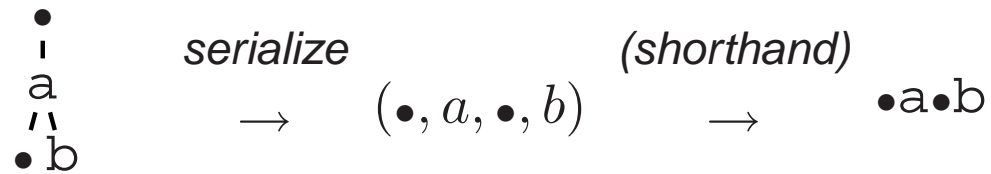
(3) Hashing: serialized pq-gram → hash string

👉 Fingerprint hash function:

Hash Function  $\lambda()$ :

node n	$\lambda(n)$
•	0
a	1
b	3
c	2
d	4

(2) Preorder serialization:  $pq$ -gram  $\rightarrow$  serialized  $pq$ -gram



(3) Hashing: serialized  $pq$ -gram  $\rightarrow$  hash string

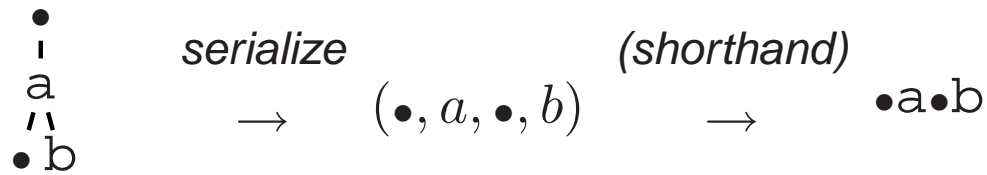
☞ Fingerprint hash function:

	node n	$\lambda(n)$
Hash Function $\lambda()$ :	$\bullet$	0
	a	1
	b	3
	c	2
	d	4

☞ Note:  $\bullet a \bullet b$  stands for  $(\bullet, \text{article}, \bullet, \text{bill})$

☛ **string tuple is mapped to integer**

(2) Preorder serialization:  $pq$ -gram  $\rightarrow$  serialized  $pq$ -gram



(3) Hashing: serialized  $pq$ -gram  $\rightarrow$  hash string

👉 Fingerprint hash function:

node n	$\lambda(n)$
•	0
a	1
b	3
c	2
d	4

Hash Function  $\lambda()$ :

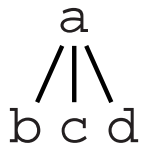
👉 Note:  $\bullet a \bullet b$  stands for  $(\bullet, \text{article}, \bullet, \text{bill})$

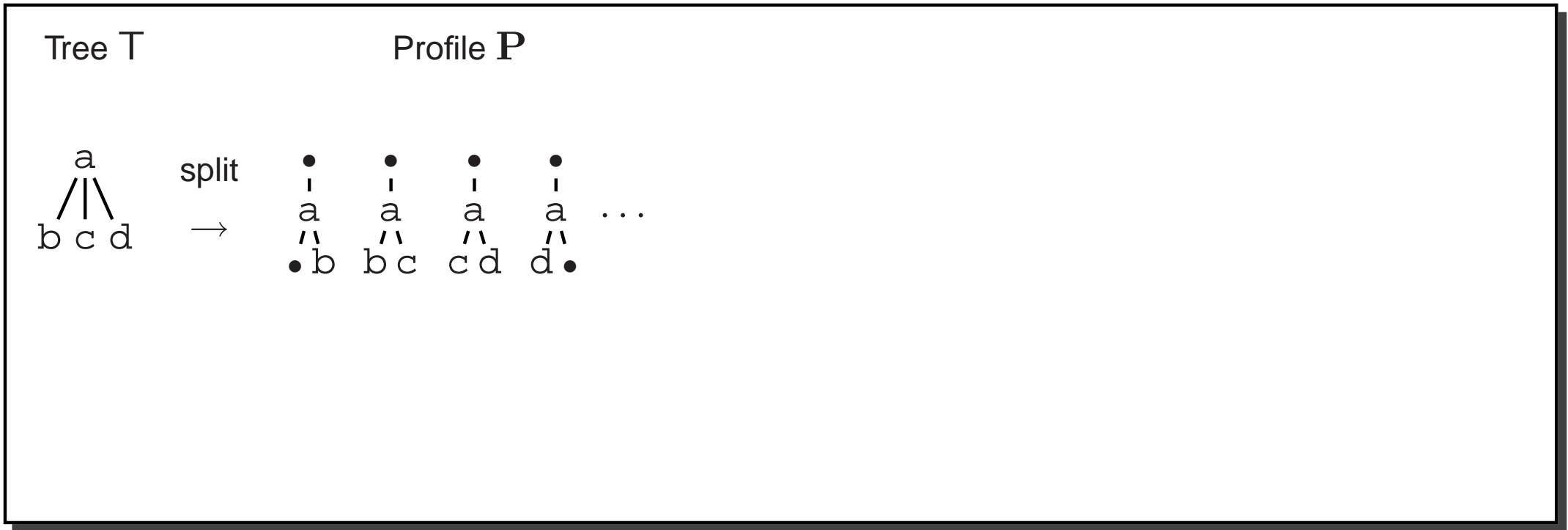
👉 **string tuple is mapped to integer**

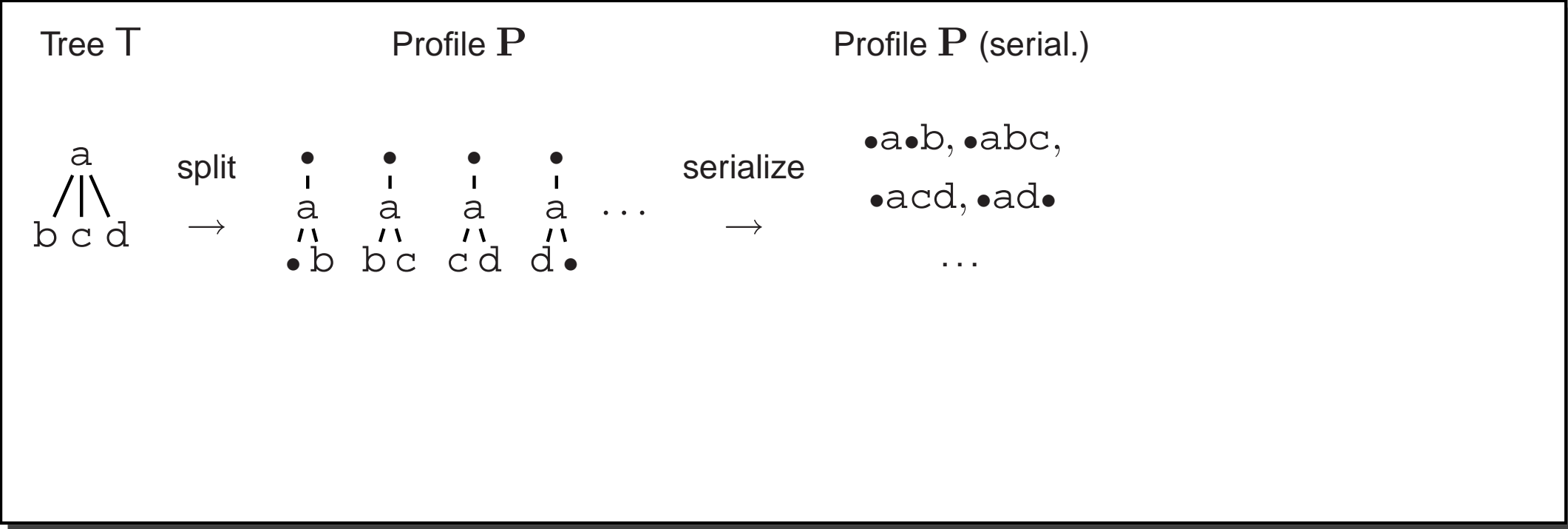
$$\begin{array}{l}
 (\bullet, \text{article}, \bullet, \text{bill}) \rightarrow 0103 \\
 (\bullet, \text{article}, \text{bill}, \text{chris}) \rightarrow 0132 \\
 \dots
 \end{array}$$

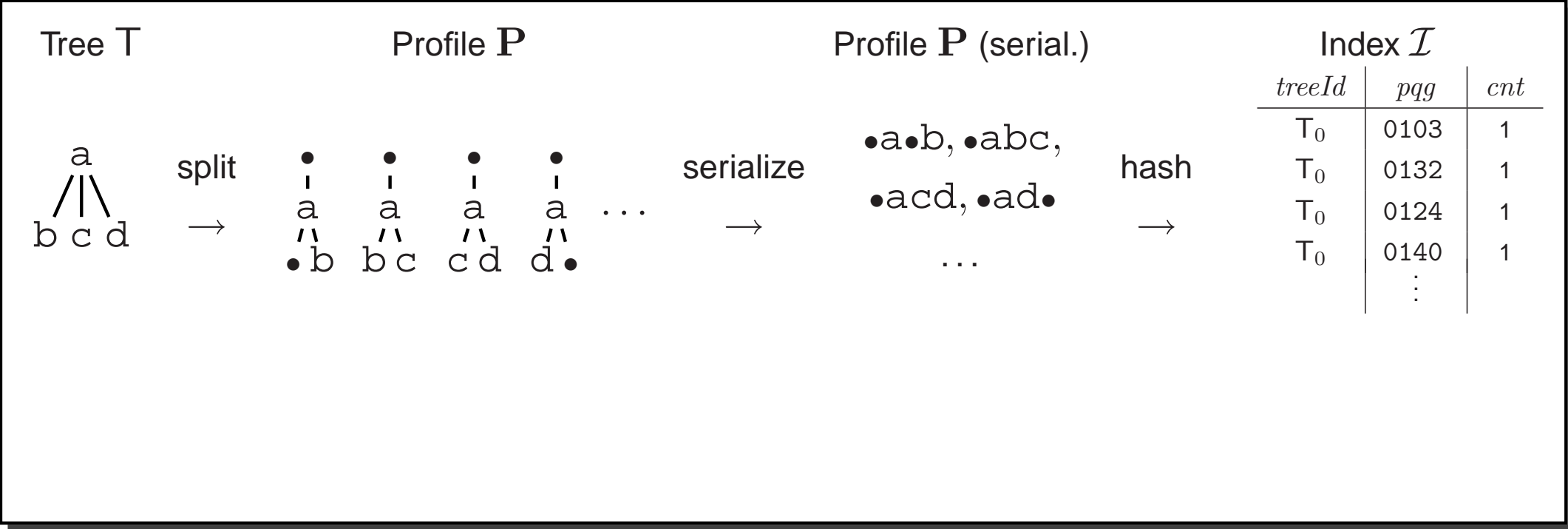


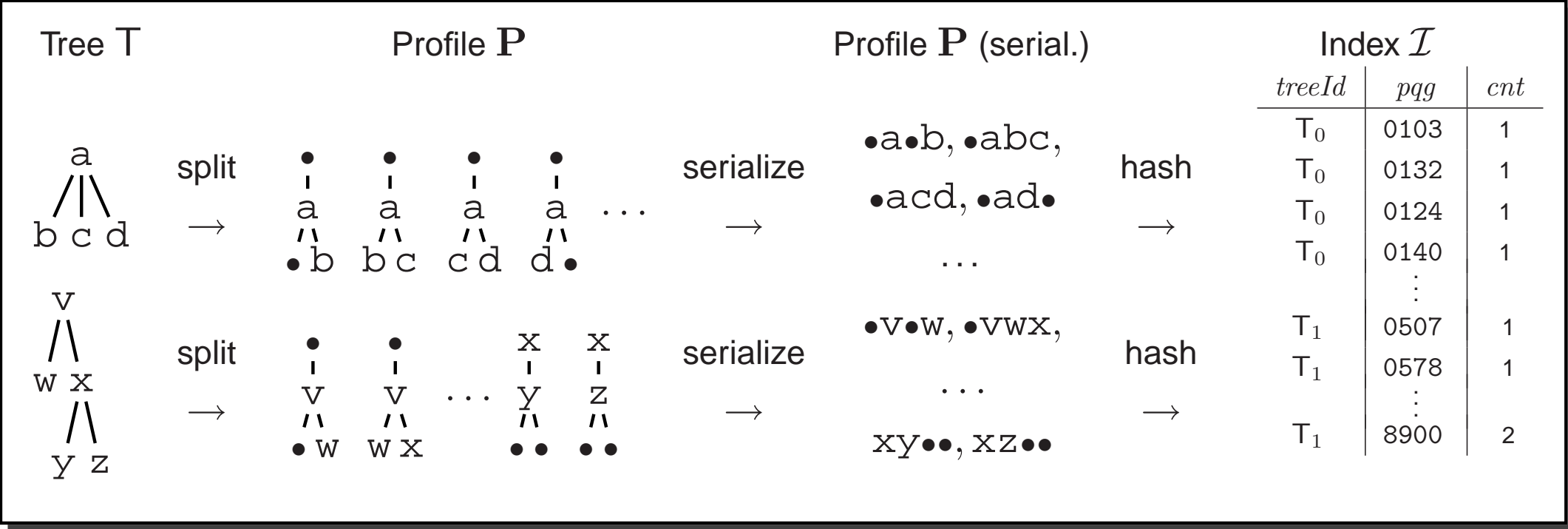
Tree T



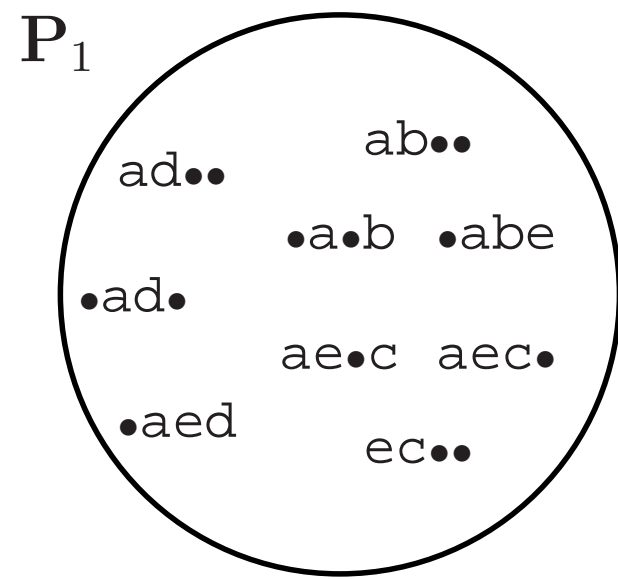


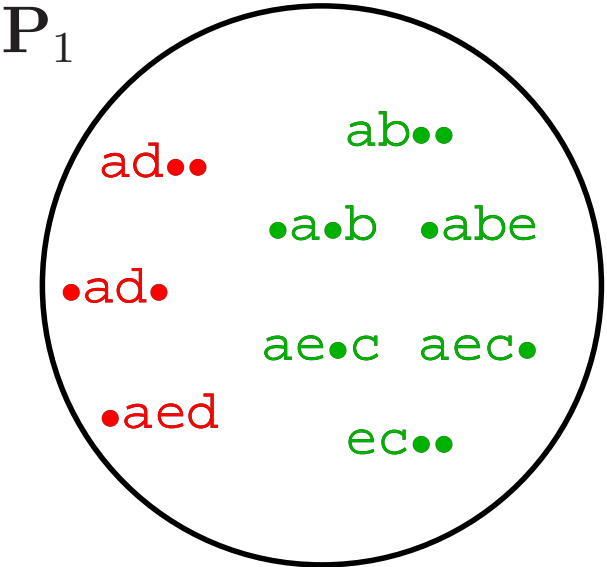
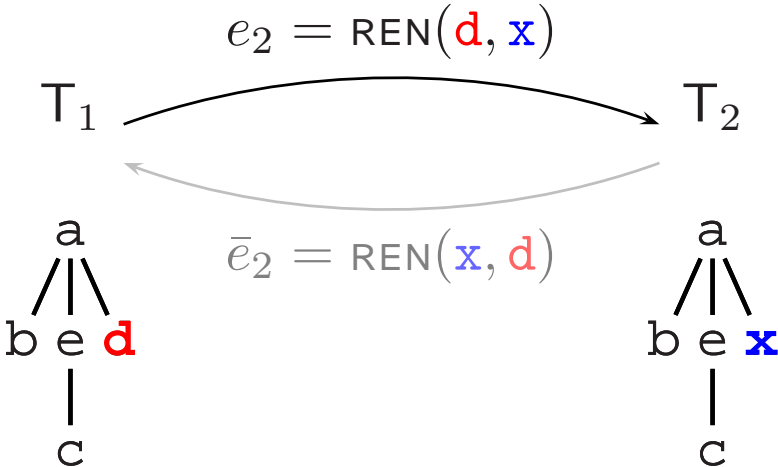


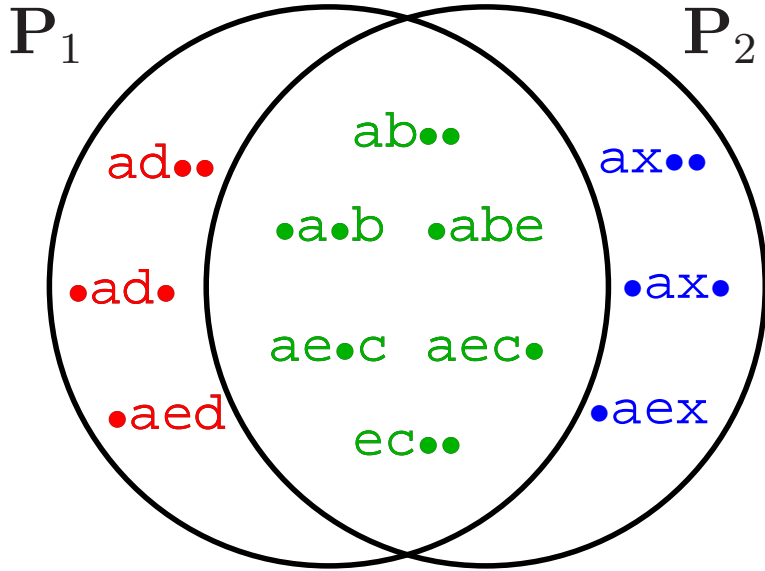
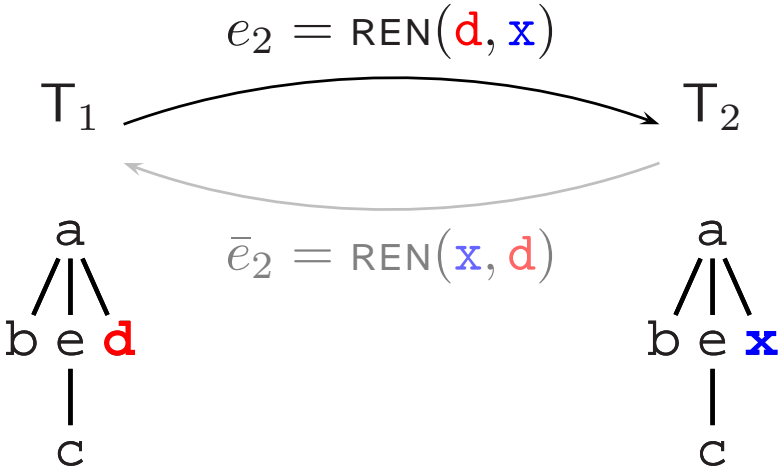


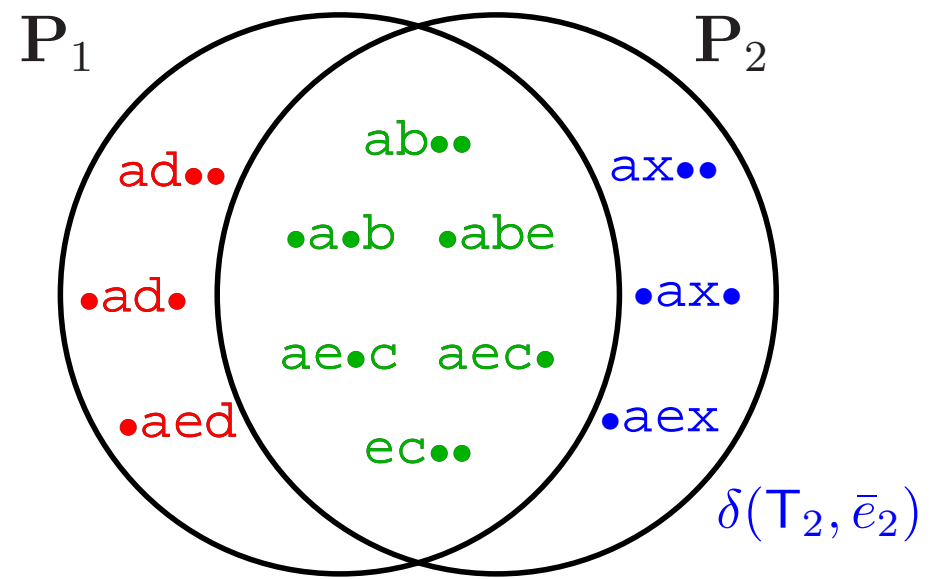
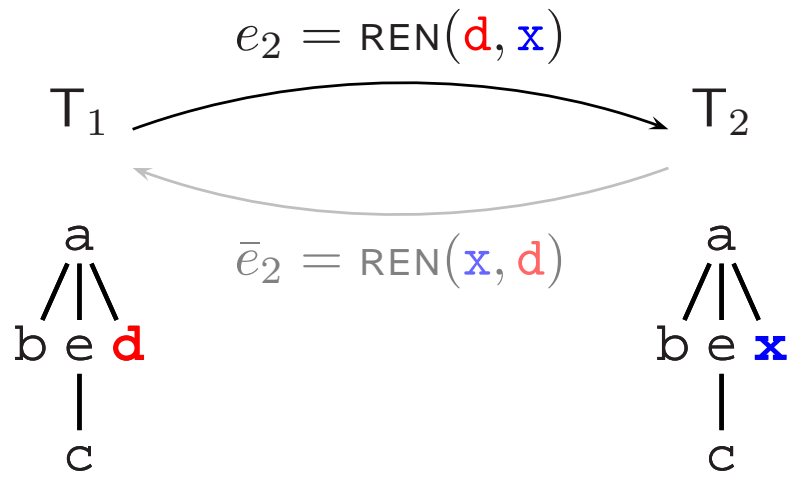






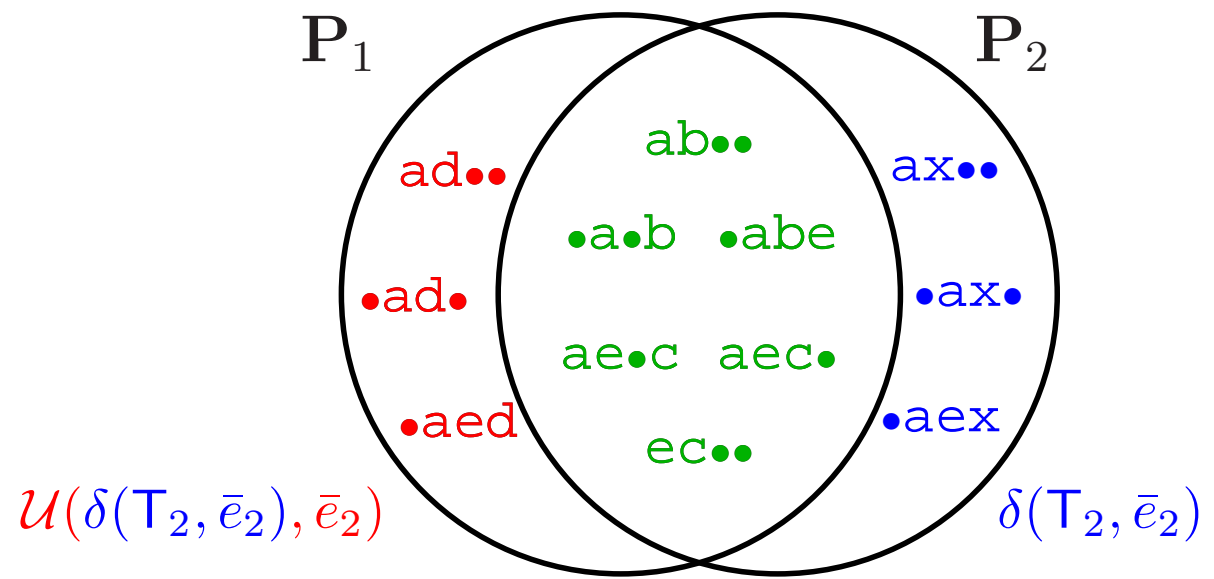
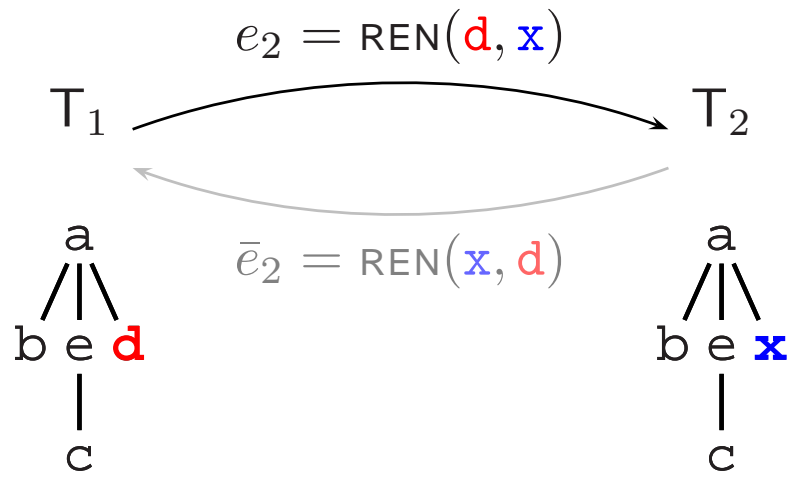






**Delta Function** (new *pq*-grams)

$$\delta(T_2, \bar{e}_2) = \begin{cases} P_2 \setminus P_1 & \text{if } \bar{e}_2 \text{ is defined on } T_2 \\ \emptyset & \text{otherwise} \end{cases}$$



**Profile Update Function (old *pq*-grams)**

$$U(\delta(T_2, \bar{e}_2), \bar{e}_2) = \mathbf{P}_1 \setminus \mathbf{P}_2$$

**Delta Function (new *pq*-grams)**

$$\delta(T_2, \bar{e}_2) = \begin{cases} \mathbf{P}_2 \setminus \mathbf{P}_1 & \text{if } \bar{e}_2 \text{ is defined on } T_2 \\ \emptyset & \text{otherwise} \end{cases}$$

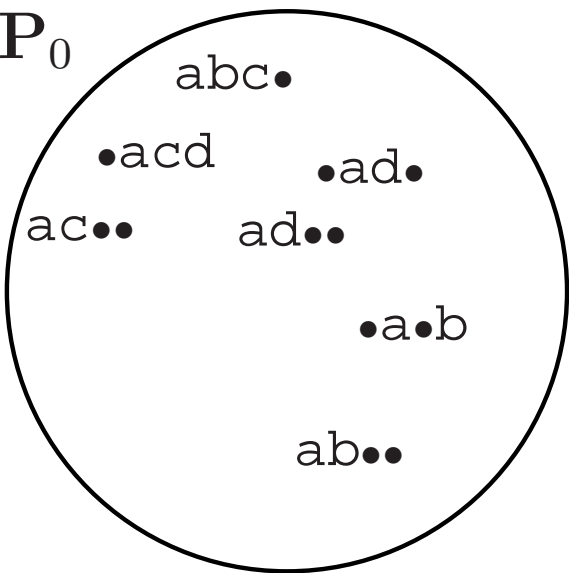
$T_0$

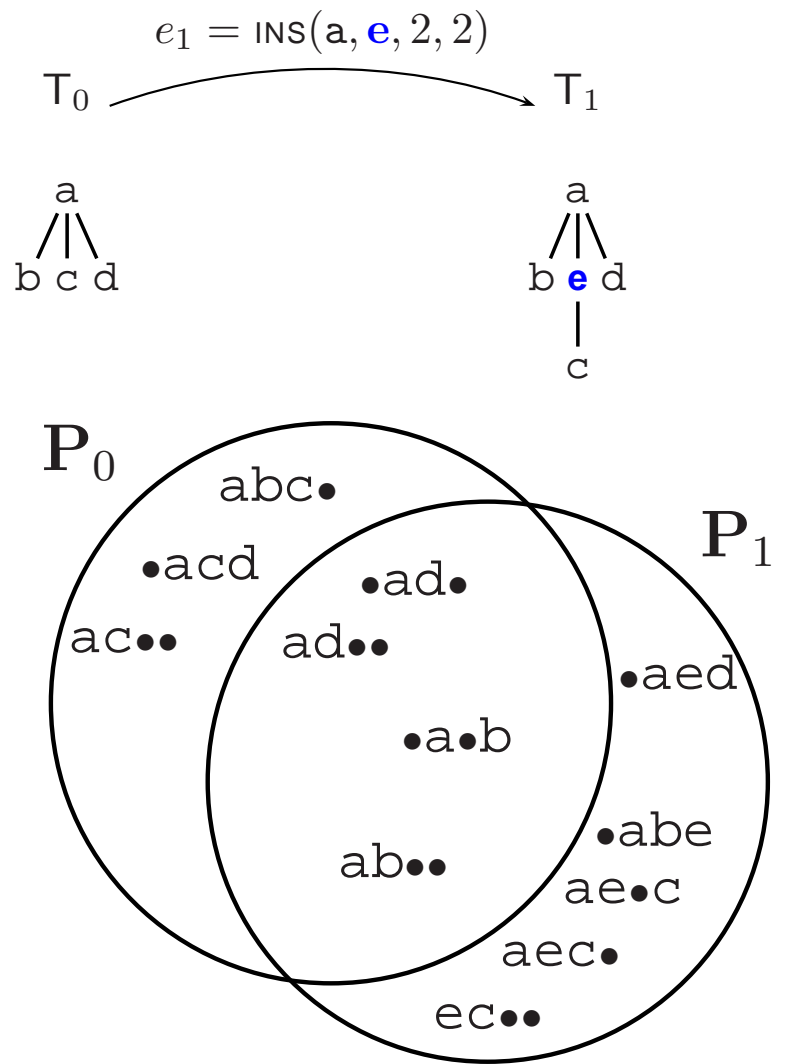


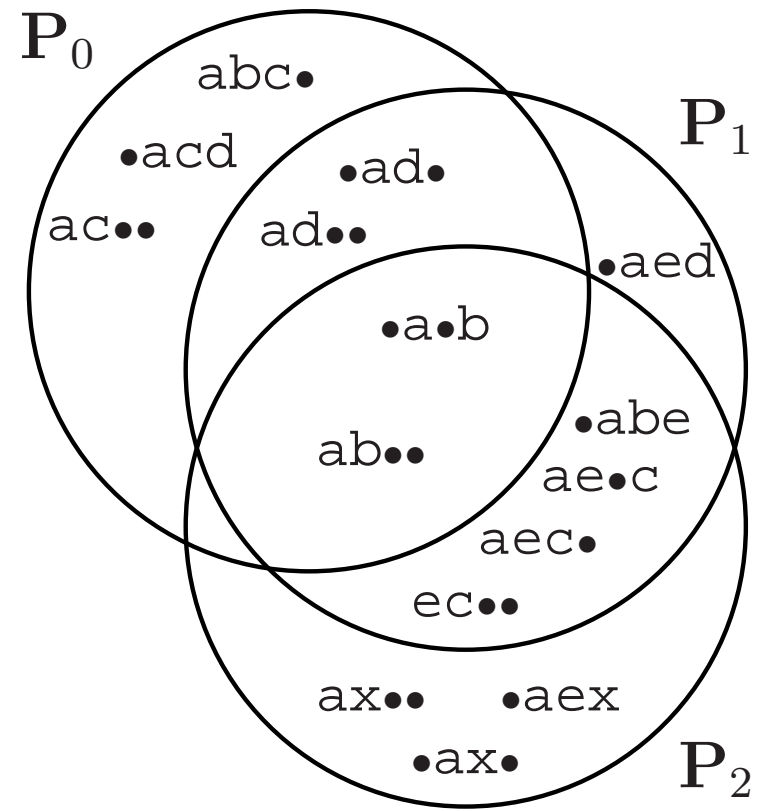
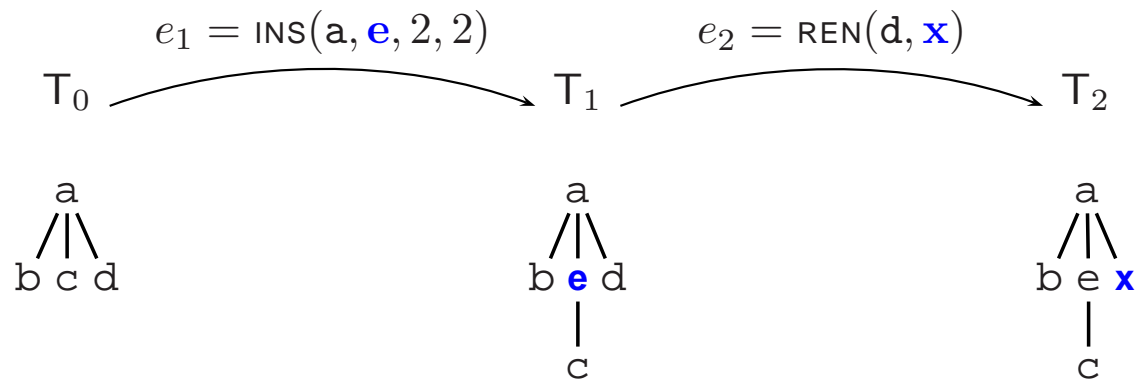
$T_0$

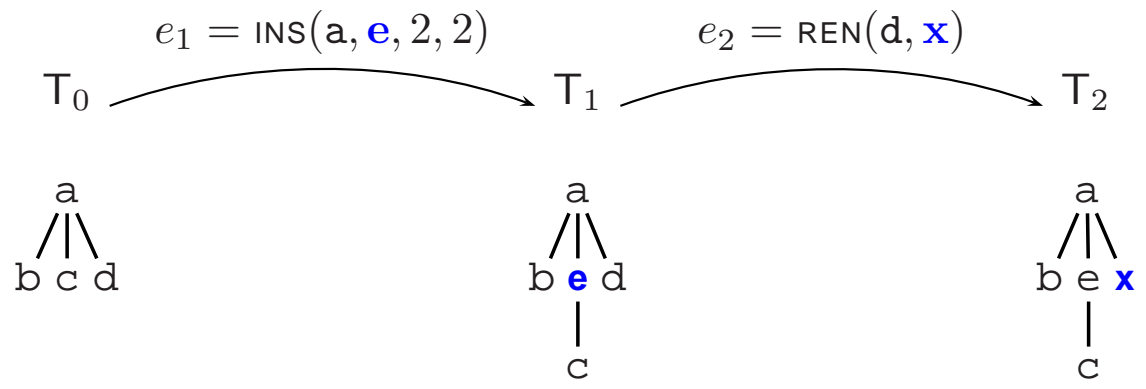


$P_0$

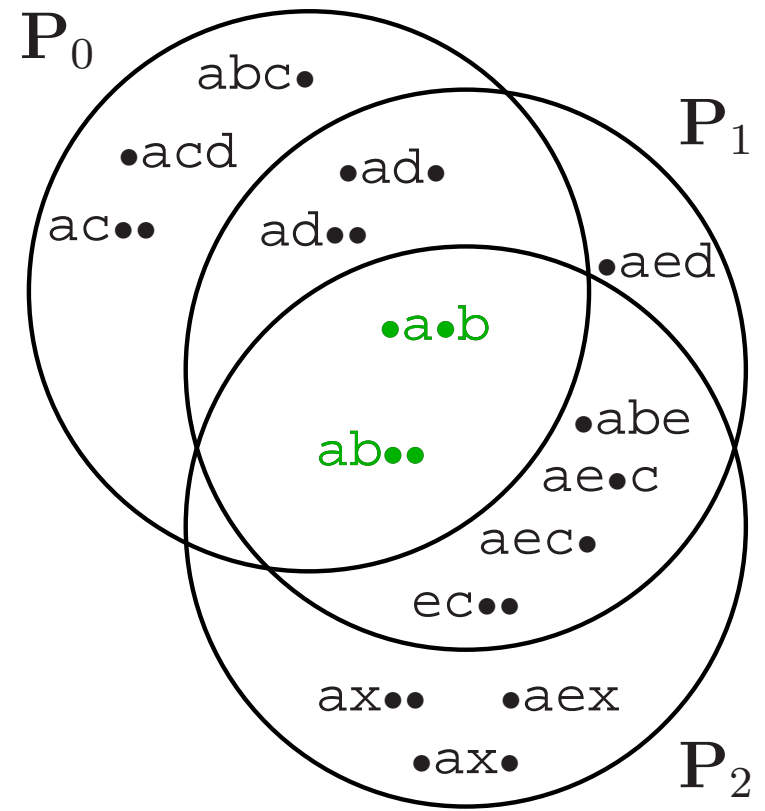




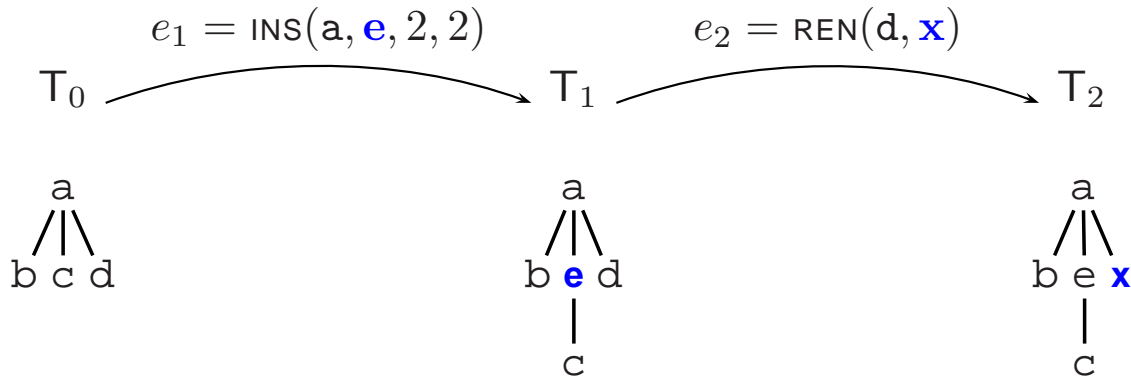




Invariant *pq*-Grams (no need for update)



$$C_n = P_0 \cap \dots \cap P_n$$

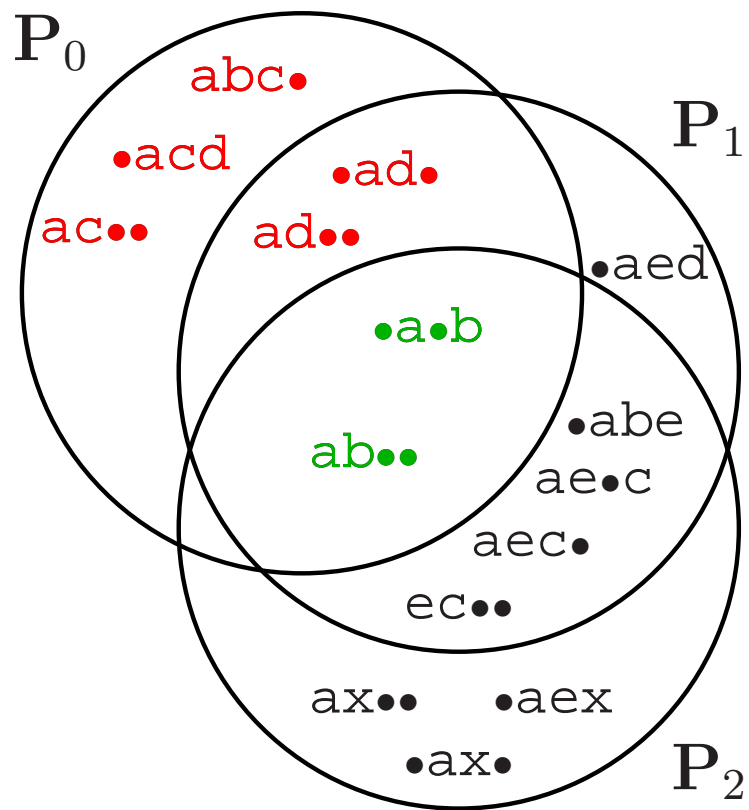


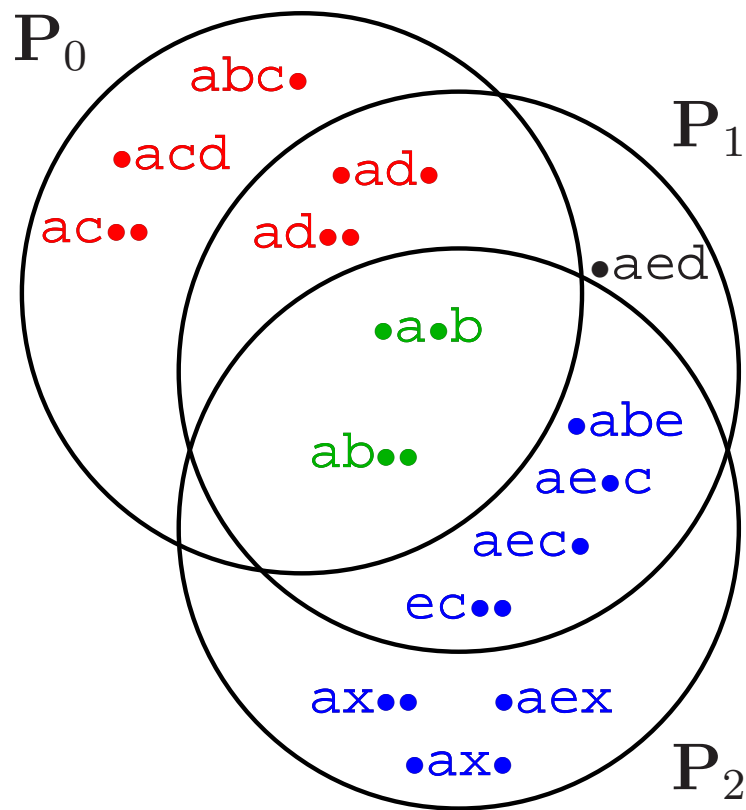
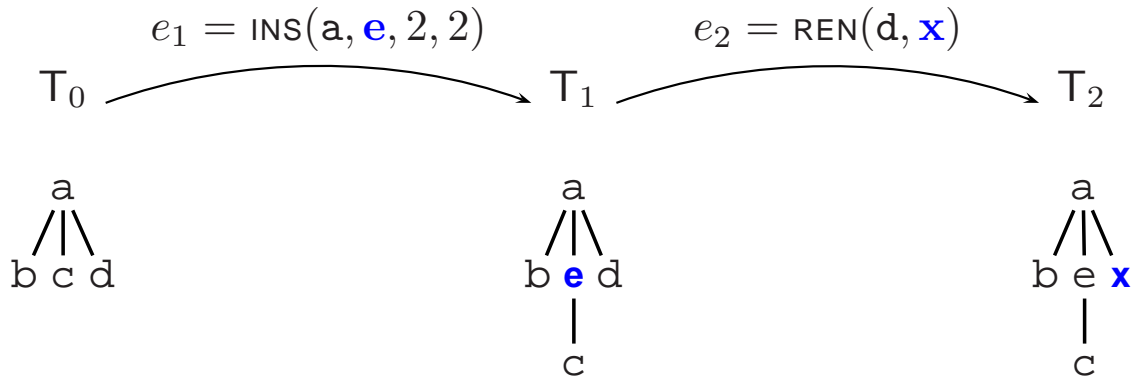
➡ Invariant *pq*-Grams (no need for update)

$$C_n = P_0 \cap \dots \cap P_n$$

➡ Old *pq*-Grams (deleted from old index  $\mathcal{I}_0$ )

$$\Delta_n^- = P_0 \setminus C_n$$





➡ Invariant  $pq$ -Grams (no need for update)

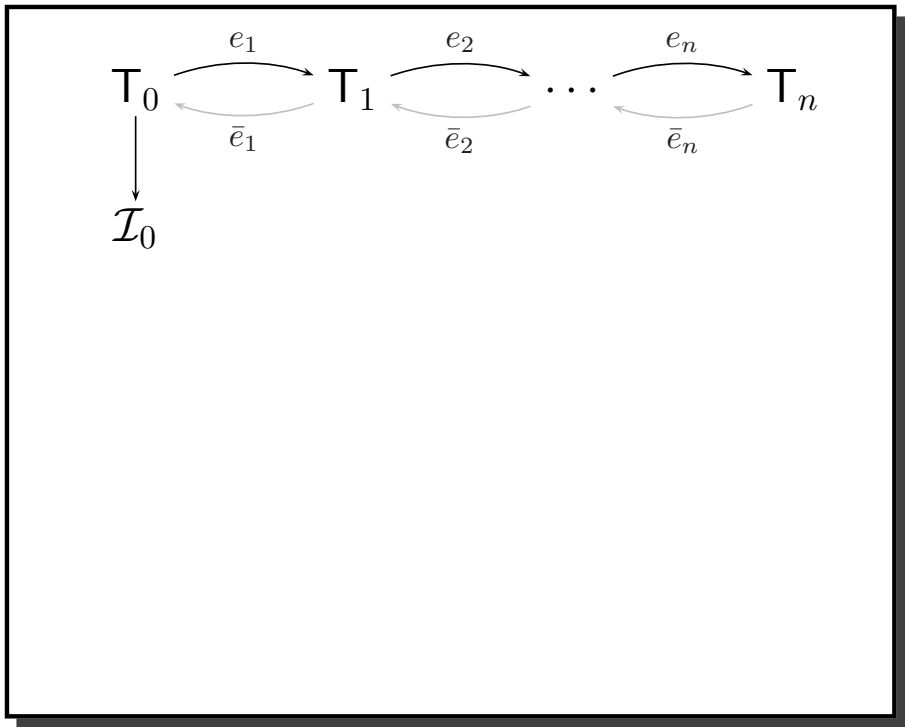
$$C_n = P_0 \cap \dots \cap P_n$$

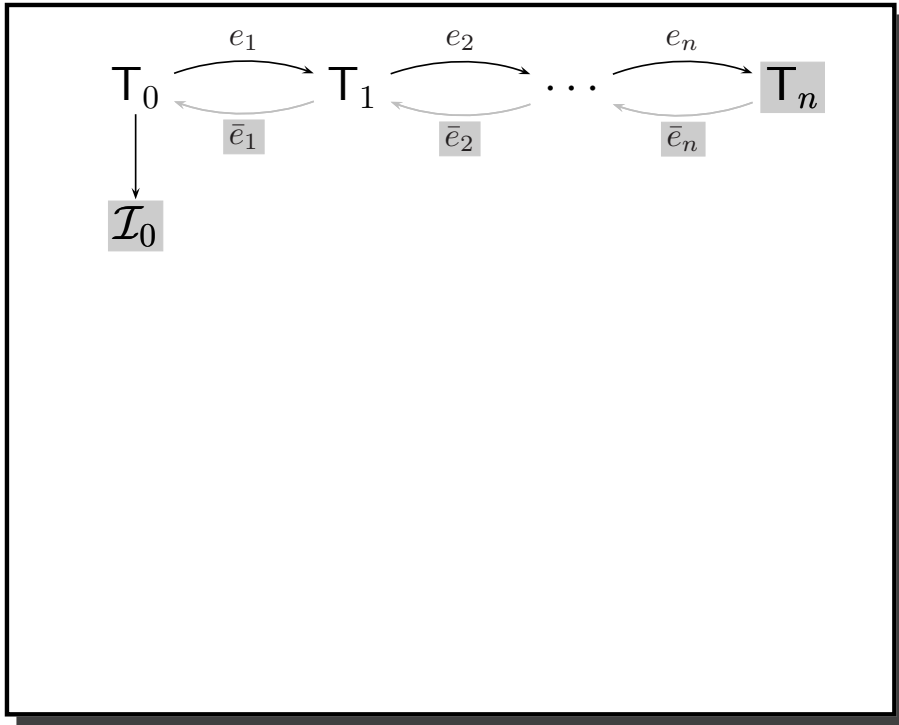
➡ Old  $pq$ -Grams (deleted from old index  $\mathcal{I}_0$ )

$$\Delta_n^- = P_0 \setminus C_n$$

➡ New  $pq$ -Grams (inserted into new index  $\mathcal{I}_n$ )

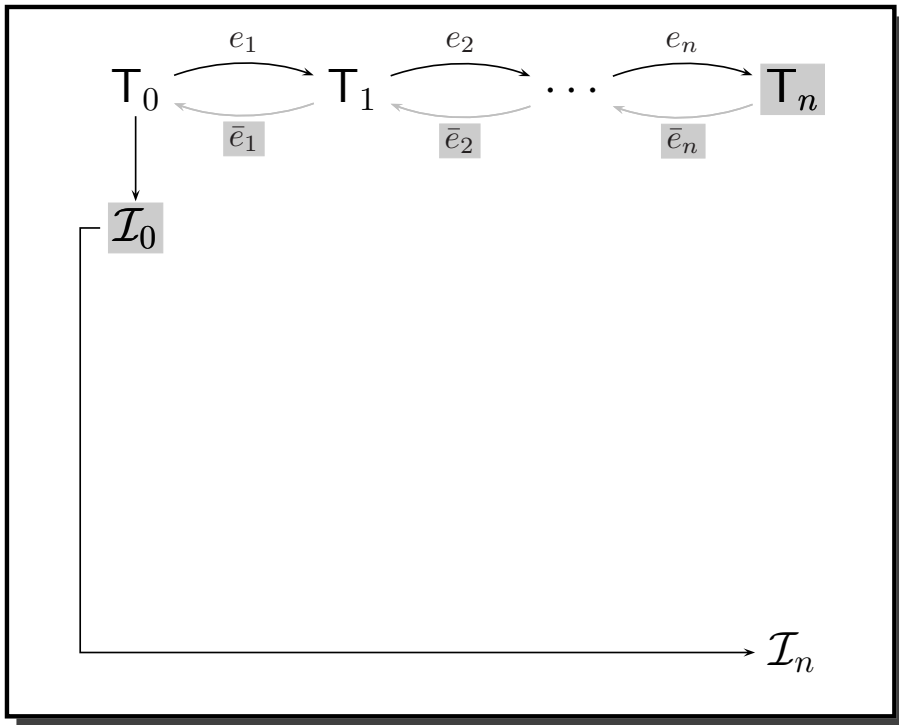
$$\Delta_n^+ = P_n \setminus C_n$$





👉 **Input:**

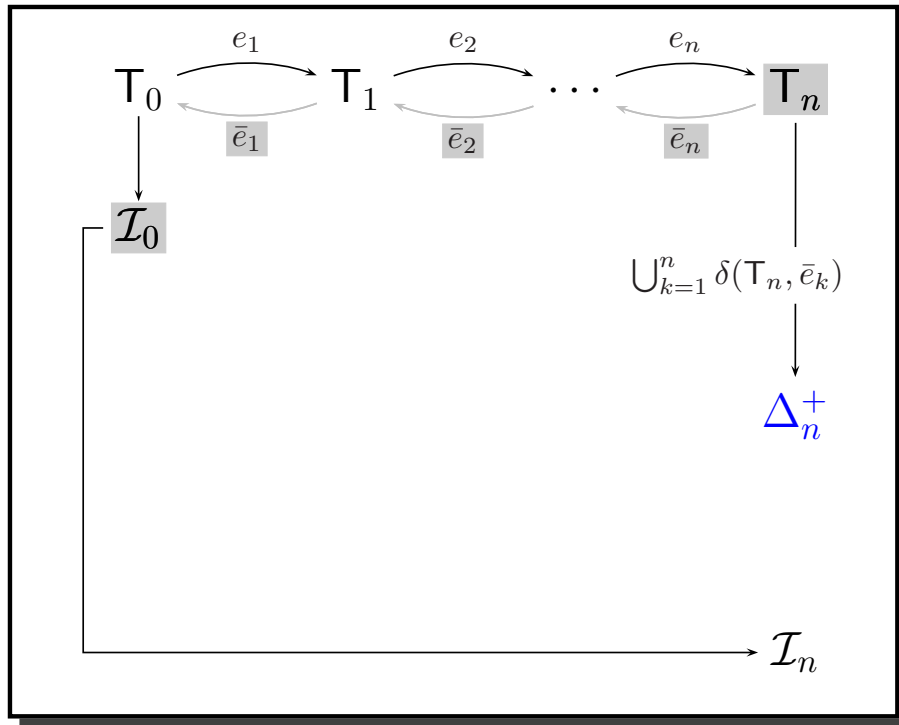
1. old index  $\mathcal{I}_0$
2. log of inverse edit operations  $(\bar{e}_n, \dots, \bar{e}_1)$
3. resulting tree  $T_n$



👉 **Input:**

1. old index  $\mathcal{I}_0$
2. log of inverse edit operations  $(\bar{e}_n, \dots, \bar{e}_1)$
3. resulting tree  $T_n$

👉 **Output:** new index  $\mathcal{I}_n$



👉 **Input:**

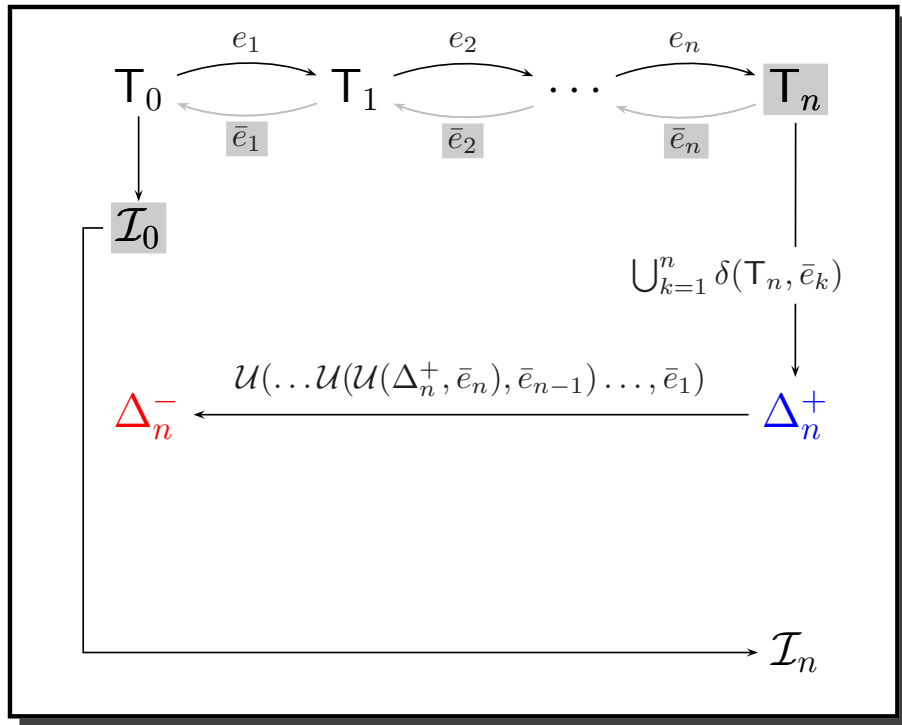
1. old index  $\mathcal{I}_0$
2. log of inverse edit operations  $(\bar{e}_n, \dots, \bar{e}_1)$
3. resulting tree  $T_n$

👉 **Output:** new index  $\mathcal{I}_n$

👉 **Solution:** 3 Steps

1. compute **new pq-grams**:

$$\Delta_n^+ = \delta(T_n, \bar{e}_1) \cup \dots \cup \delta(T_n, \bar{e}_n)$$



☞ **Input:**

1. old index  $\mathcal{I}_0$
2. log of inverse edit operations  $(\bar{e}_n, \dots, \bar{e}_1)$
3. resulting tree  $T_n$

☞ **Output:** new index  $\mathcal{I}_n$

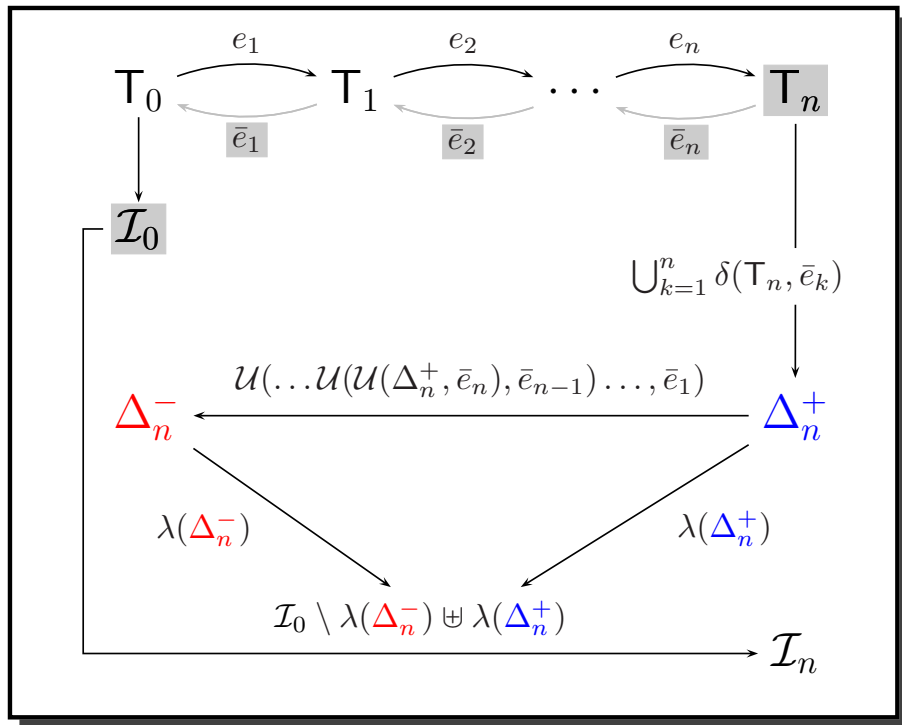
☞ **Solution:** 3 Steps

1. compute **new pq-grams**:

$$\Delta_n^+ = \delta(T_n, \bar{e}_1) \cup \dots \cup \delta(T_n, \bar{e}_n)$$

2. compute **old pq-grams**:

$$\Delta_n^- = \mathcal{U}(\dots \mathcal{U}(\mathcal{U}(\Delta_n^+, \bar{e}_n), \bar{e}_{n-1}) \dots, \bar{e}_1)$$



👉 **Input:**

1. old index  $\mathcal{I}_0$
2. log of inverse edit operations  $(\bar{e}_n, \dots, \bar{e}_1)$
3. resulting tree  $T_n$

👉 **Output:** new index  $\mathcal{I}_n$

👉 **Solution:** 3 Steps

1. compute **new pq-grams**:

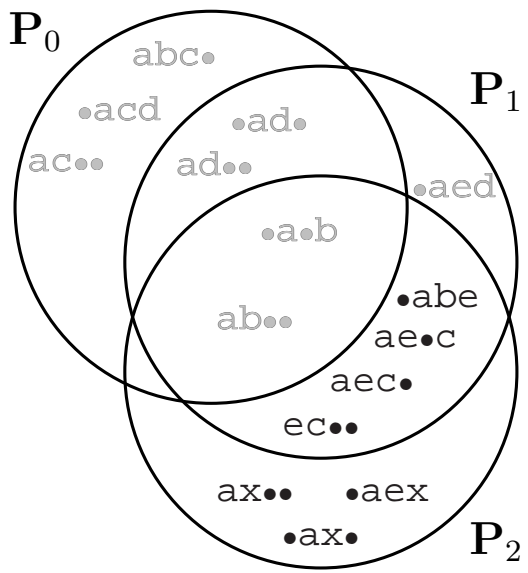
$$\Delta_n^+ = \delta(T_n, \bar{e}_1) \cup \dots \cup \delta(T_n, \bar{e}_n)$$

2. compute **old pq-grams**:

$$\Delta_n^- = \mathcal{U}(\dots \mathcal{U}(\mathcal{U}(\Delta_n^+, \bar{e}_n), \bar{e}_{n-1}) \dots, \bar{e}_1)$$

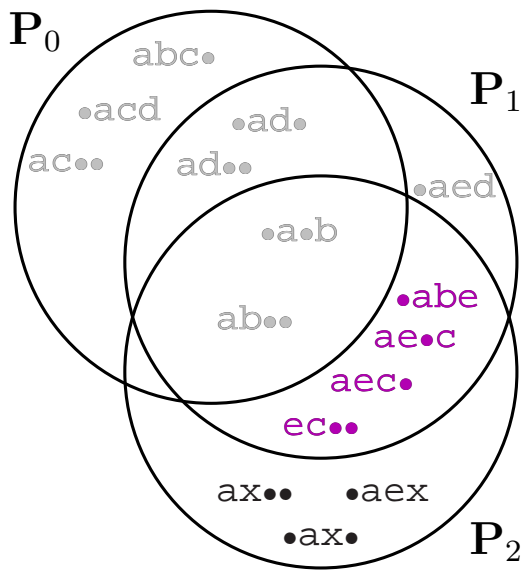
3. **update index:**

$$\mathcal{I}_n = \mathcal{I}_0 \setminus \lambda(\Delta_n^-) \uplus \lambda(\Delta_n^+)$$

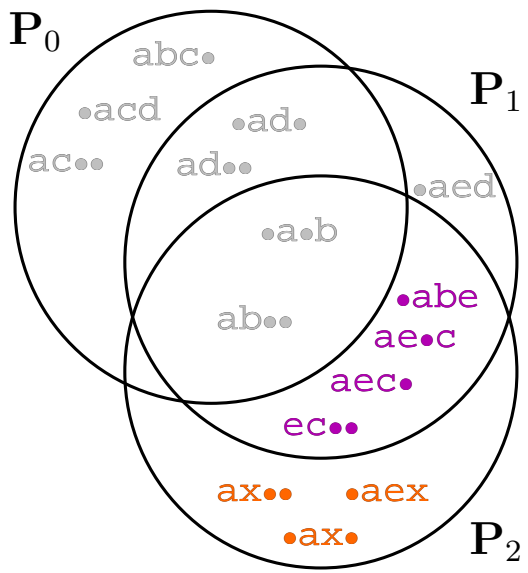


$$\Delta_2^+ =$$

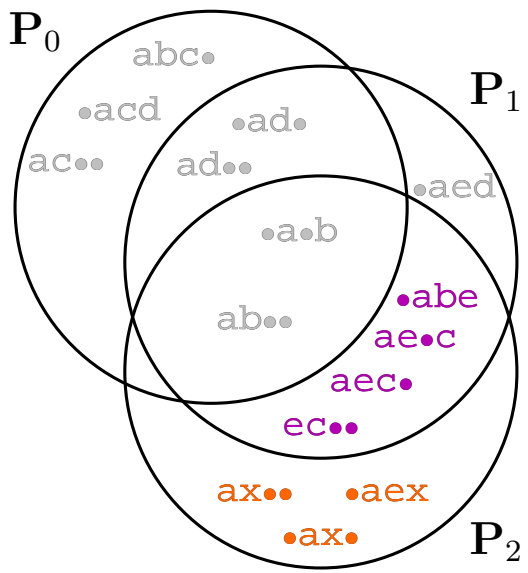
$$\Delta_2^+ = [\delta(T_1, \bar{e}_1) \cap P_2]$$



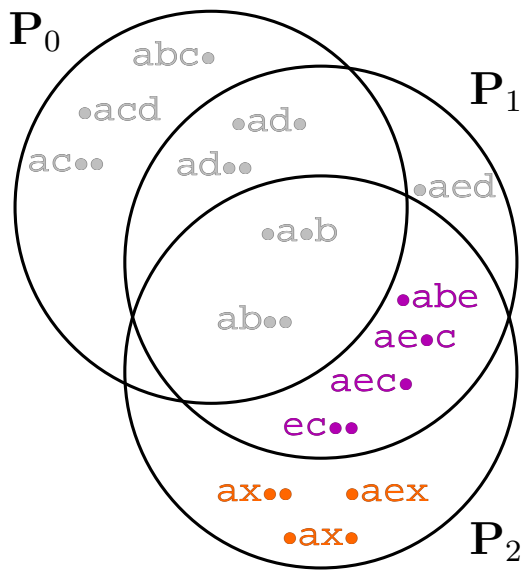




$$\Delta_2^+ = [\delta(T_1, \bar{e}_1) \cap P_2] \cup \underbrace{\delta(T_2, \bar{e}_2)}_{\text{OK}}$$

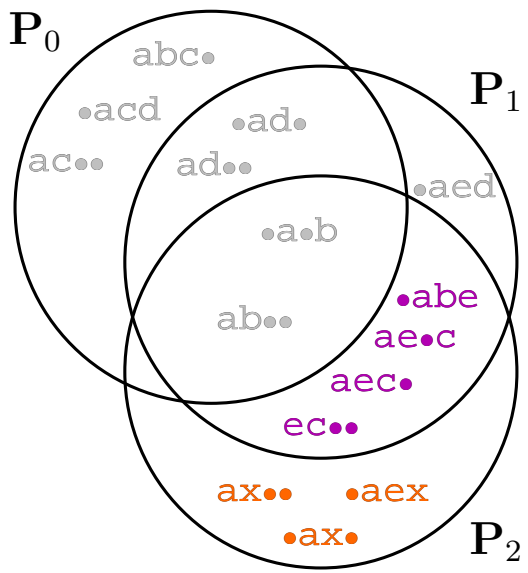


$$\Delta_2^+ = \underbrace{[\delta(T_1, \bar{e}_1) \cap P_2]}_{T_1 \text{ not given!}} \cup \underbrace{\delta(T_2, \bar{e}_2)}_{\text{OK}}$$



$$\Delta_2^+ = \underbrace{[\delta(T_1, \bar{e}_1) \cap P_2]}_{T_1 \text{ not given!}} \cup \underbrace{\delta(T_2, \bar{e}_2)}_{\text{OK}}$$

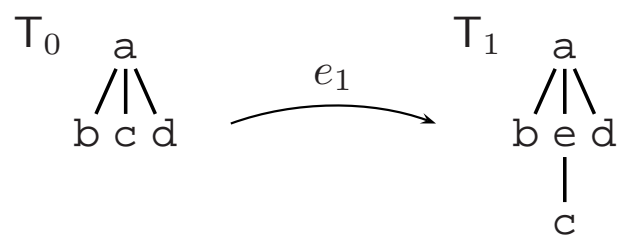
➔ Compute  $\delta$  on wrong tree ( $T_2$ ) and fix it

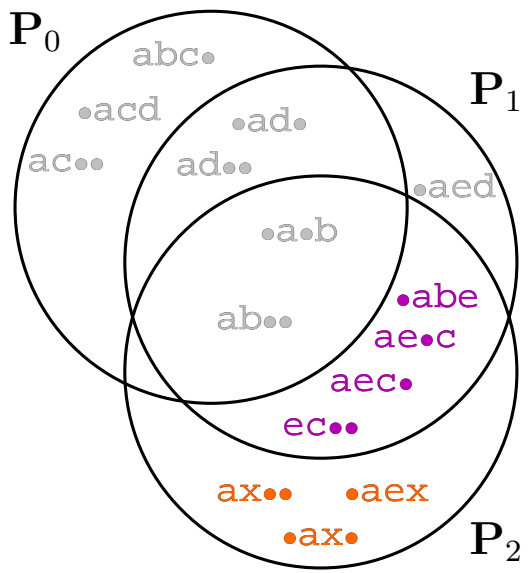


$$\Delta_2^+ = \underbrace{[\delta(T_1, \bar{e}_1) \cap P_2]}_{T_1 \text{ not given!}} \cup \underbrace{\delta(T_2, \bar{e}_2)}_{\text{OK}}$$

👉 Compute  $\delta$  on wrong tree ( $T_2$ ) and fix it

Reality:

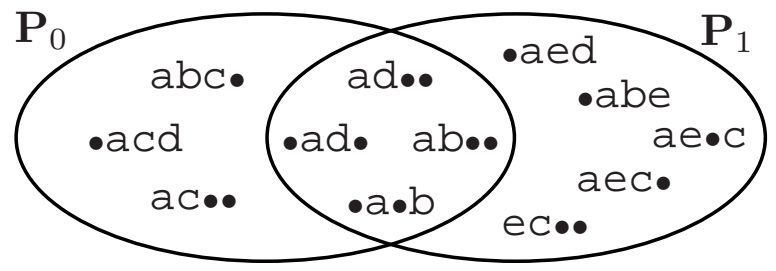
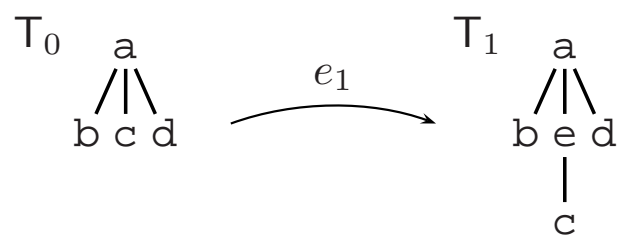


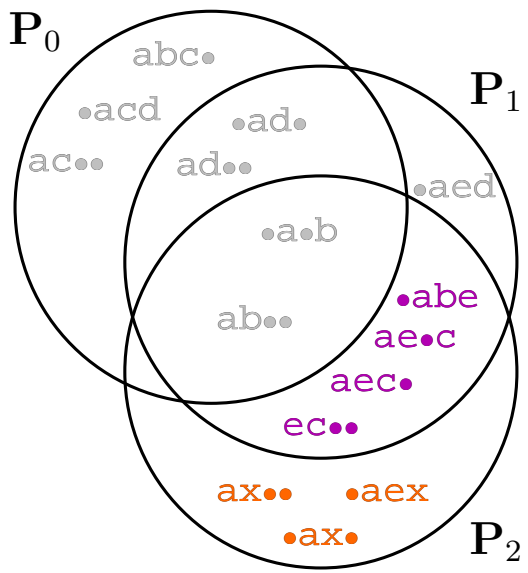


$$\Delta_2^+ = \underbrace{[\delta(T_1, \bar{e}_1) \cap P_2]}_{T_1 \text{ not given!}} \cup \underbrace{\delta(T_2, \bar{e}_2)}_{\text{OK}}$$

👉 Compute  $\delta$  on wrong tree ( $T_2$ ) and fix it

Reality:

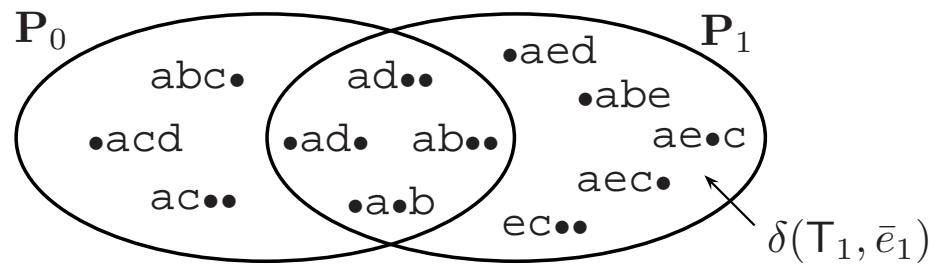
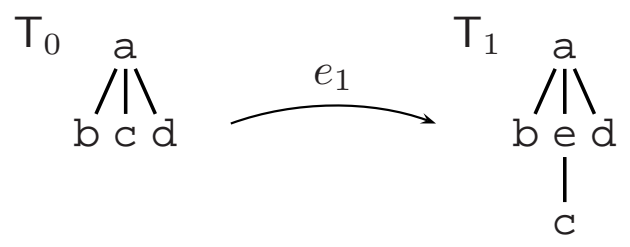


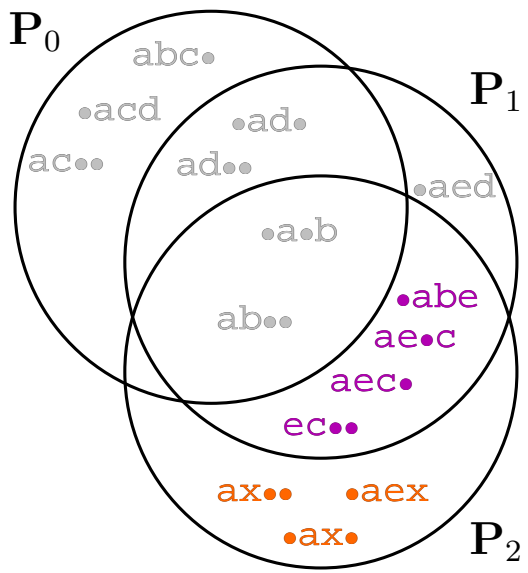


$$\Delta_2^+ = \underbrace{[\delta(T_1, \bar{e}_1) \cap P_2]}_{T_1 \text{ not given!}} \cup \underbrace{\delta(T_2, \bar{e}_2)}_{\text{OK}}$$

👉 Compute  $\delta$  on wrong tree ( $T_2$ ) and fix it

Reality:

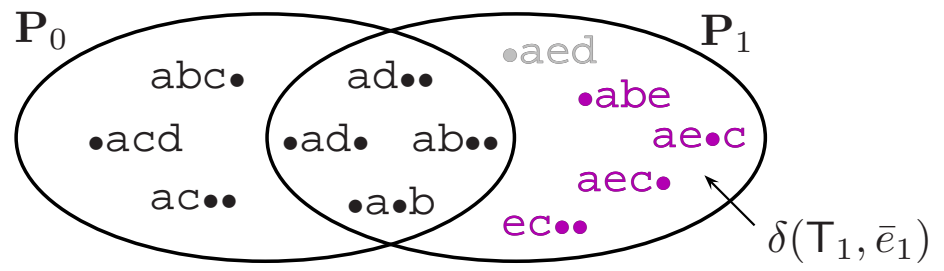
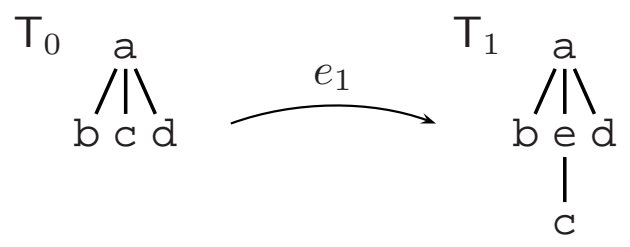


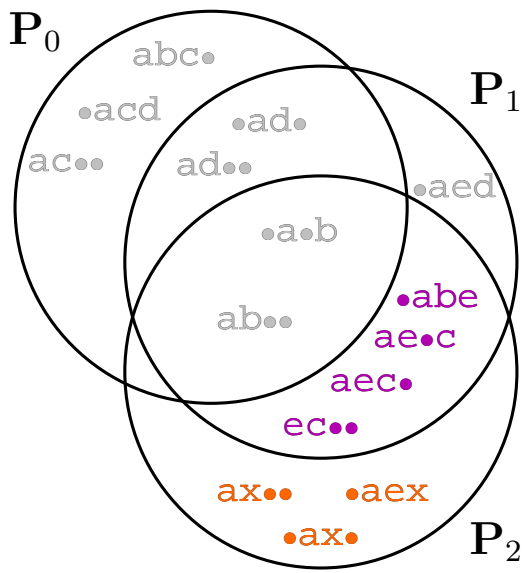


$$\Delta_2^+ = \underbrace{[\delta(T_1, \bar{e}_1) \cap P_2]}_{T_1 \text{ not given!}} \cup \underbrace{\delta(T_2, \bar{e}_2)}_{\text{OK}}$$

👉 Compute  $\delta$  on wrong tree ( $T_2$ ) and fix it

Reality:

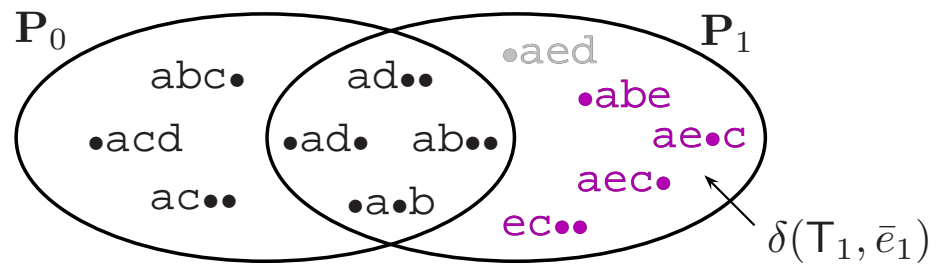
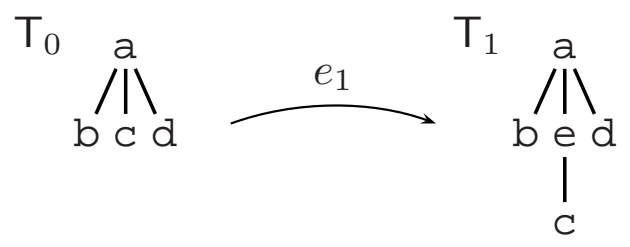




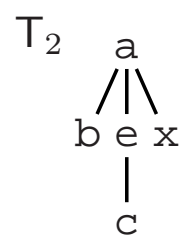
$$\Delta_2^+ = \underbrace{[\delta(T_1, \bar{e}_1) \cap P_2]}_{T_1 \text{ not given!}} \cup \underbrace{\delta(T_2, \bar{e}_2)}_{\text{OK}}$$

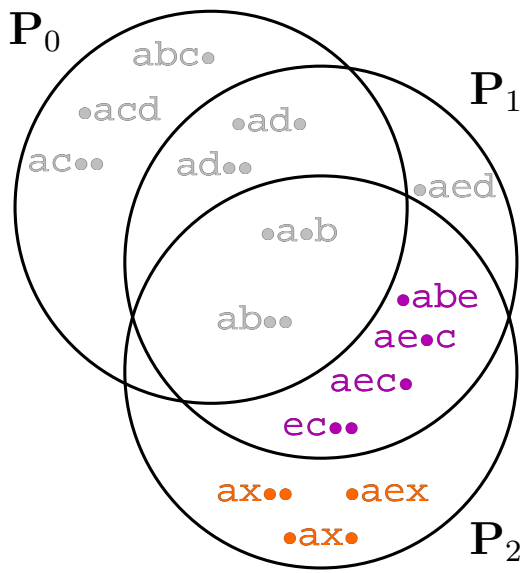
👉 Compute  $\delta$  on wrong tree ( $T_2$ ) and fix it

Reality:



Computation:

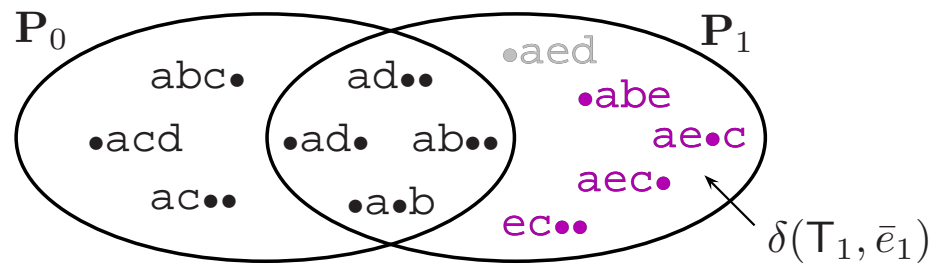
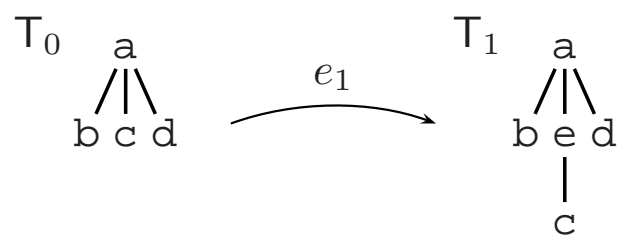




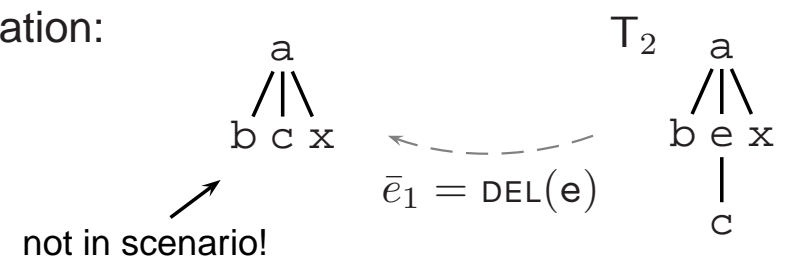
$$\Delta_2^+ = \underbrace{[\delta(T_1, \bar{e}_1) \cap P_2]}_{T_1 \text{ not given!}} \cup \underbrace{\delta(T_2, \bar{e}_2)}_{\text{OK}}$$

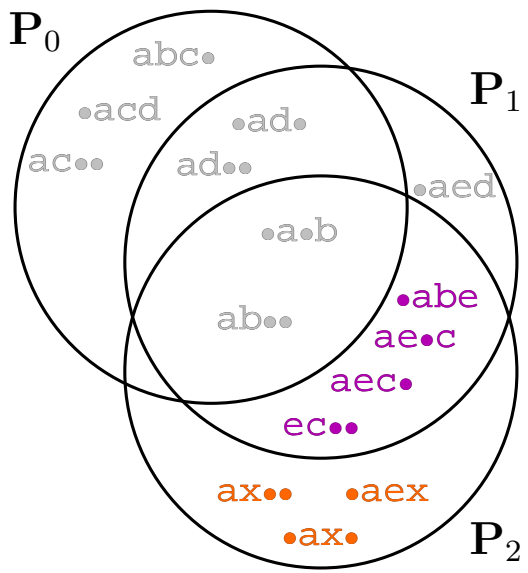
👉 Compute  $\delta$  on wrong tree ( $T_2$ ) and fix it

Reality:



Computation:

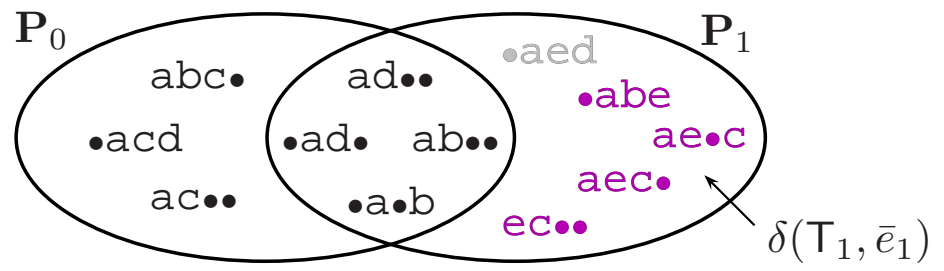
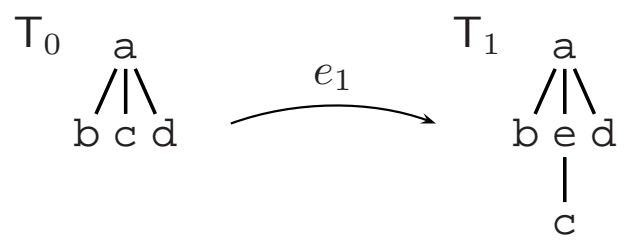




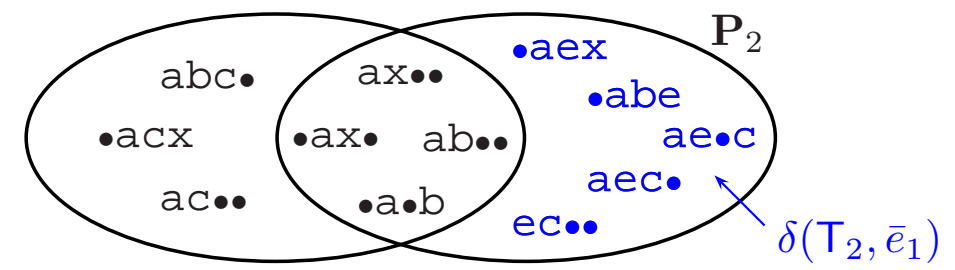
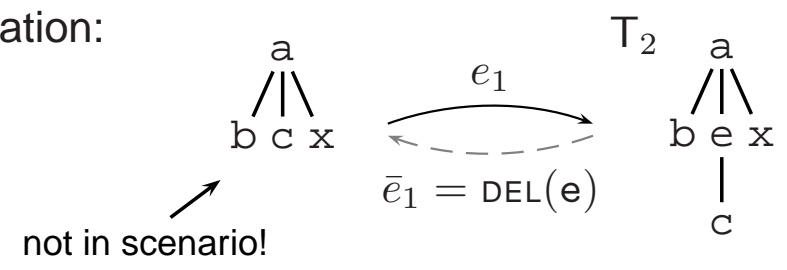
$$\Delta_2^+ = \underbrace{[\delta(T_1, \bar{e}_1) \cap P_2]}_{T_1 \text{ not given!}} \cup \underbrace{\delta(T_2, \bar{e}_2)}_{\text{OK}}$$

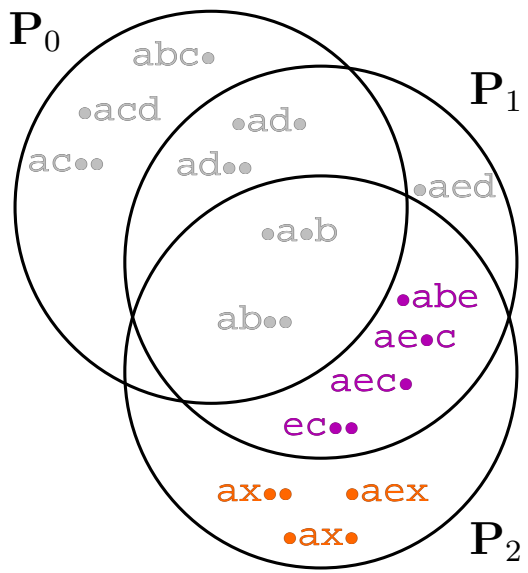
👉 Compute  $\delta$  on wrong tree ( $T_2$ ) and fix it

Reality:



Computation:

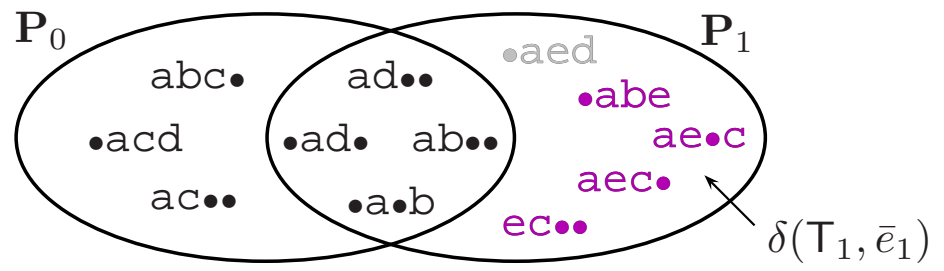
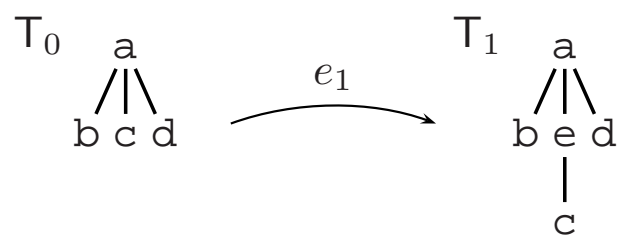




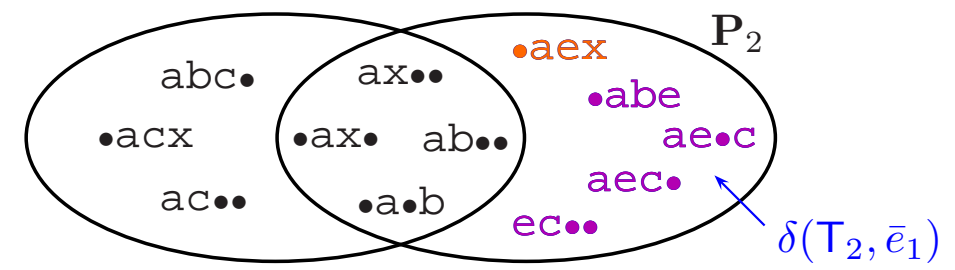
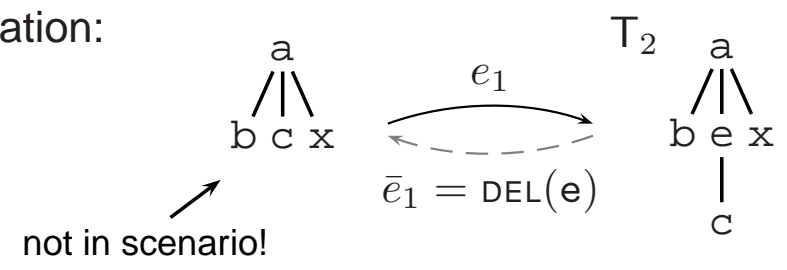
$$\Delta_2^+ = \underbrace{[\delta(T_1, \bar{e}_1) \cap P_2]}_{T_1 \text{ not given!}} \cup \underbrace{\delta(T_2, \bar{e}_2)}_{\text{OK}}$$

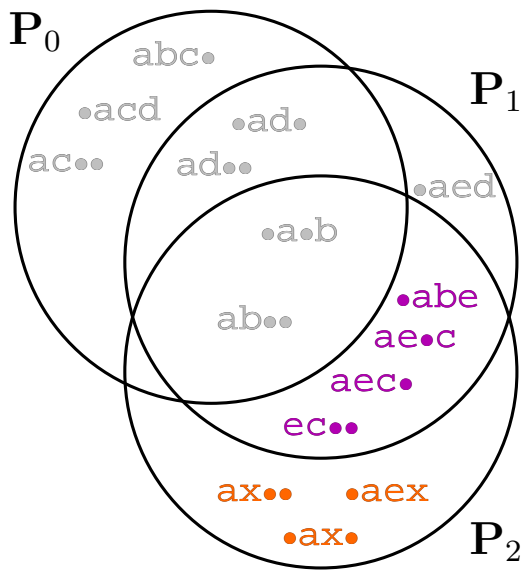
👉 Compute  $\delta$  on wrong tree ( $T_2$ ) and fix it

Reality:



Computation:



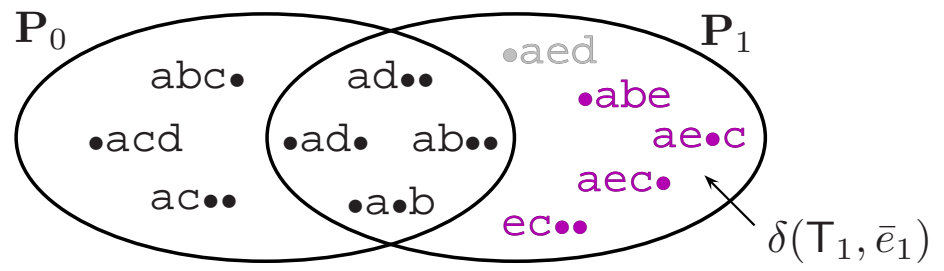
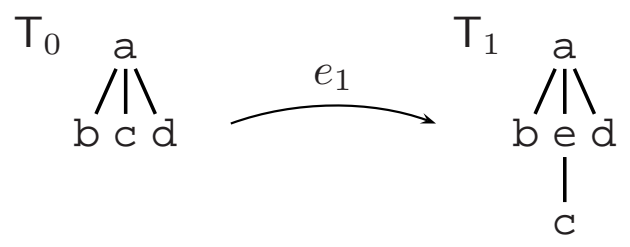


$$\Delta_2^+ = \underbrace{[\delta(T_1, \bar{e}_1) \cap P_2]}_{T_1 \text{ not given!}} \cup \underbrace{\delta(T_2, \bar{e}_2)}_{\text{OK}}$$

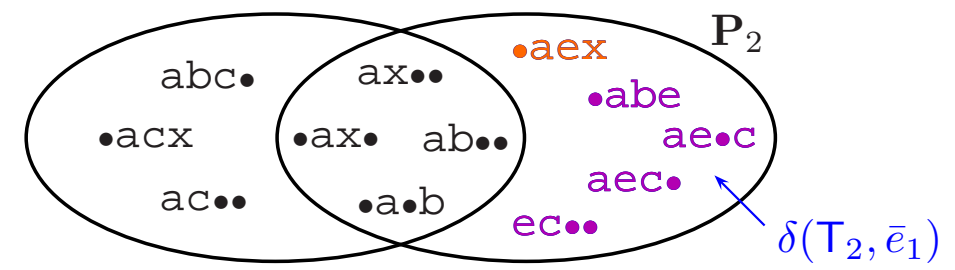
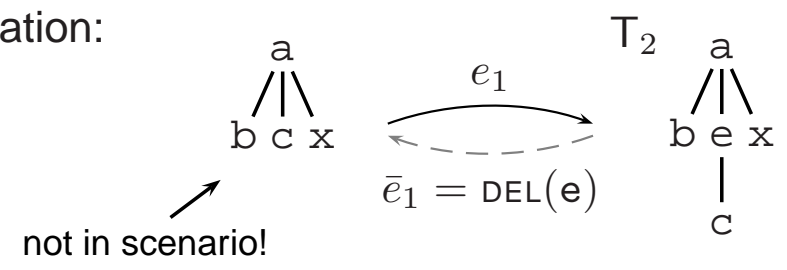
→ Compute  $\delta$  on wrong tree ( $T_2$ ) and fix it

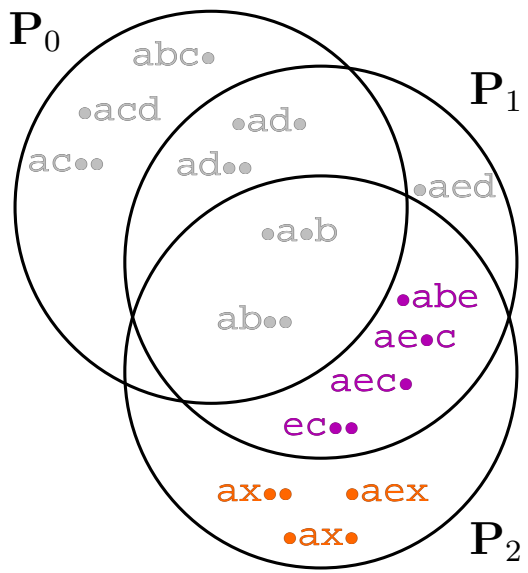
$$\delta(T_1, \bar{e}_1) \cap P_2 = \delta(T_2, \bar{e}_1) \setminus \delta(T_2, \bar{e}_2)$$

Reality:



Computation:





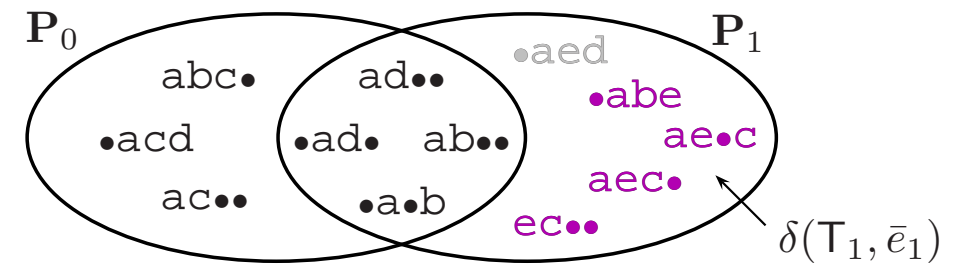
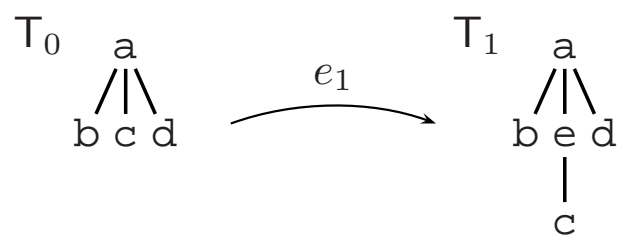
$$\Delta_2^+ = \underbrace{[\delta(T_1, \bar{e}_1) \cap P_2]}_{T_1 \text{ not given!}} \cup \underbrace{\delta(T_2, \bar{e}_2)}_{\text{OK}}$$

→ Compute  $\delta$  on wrong tree ( $T_2$ ) and fix it

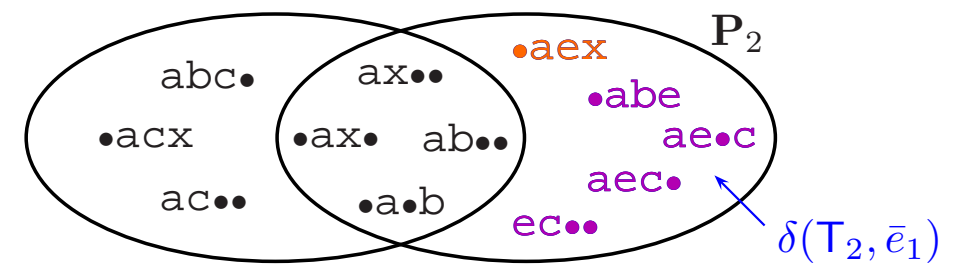
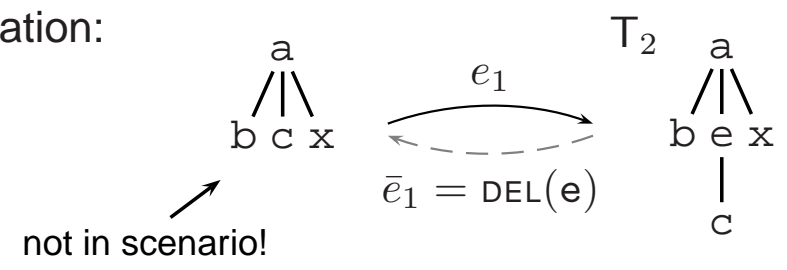
$$\delta(T_1, \bar{e}_1) \cap P_2 = \delta(T_2, \bar{e}_1) \setminus \delta(T_2, \bar{e}_2)$$

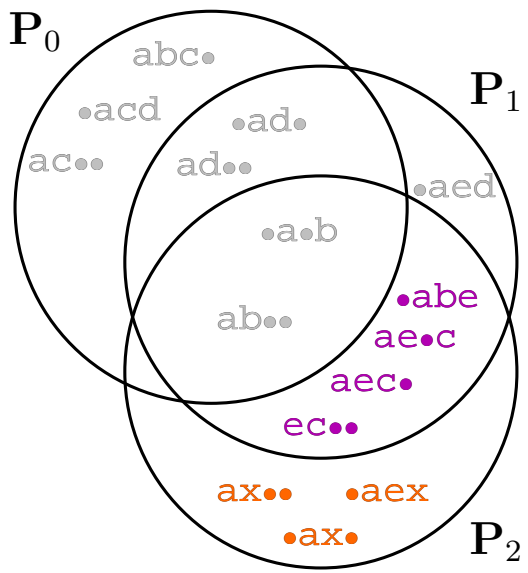
$$\Delta_2^+ = \delta(T_2, \bar{e}_1) \setminus \delta(T_2, \bar{e}_2) \cup \delta(T_2, \bar{e}_2)$$

Reality:



Computation:





$$\Delta_2^+ = \underbrace{[\delta(T_1, \bar{e}_1) \cap P_2]}_{T_1 \text{ not given!}} \cup \underbrace{\delta(T_2, \bar{e}_2)}_{\text{OK}}$$

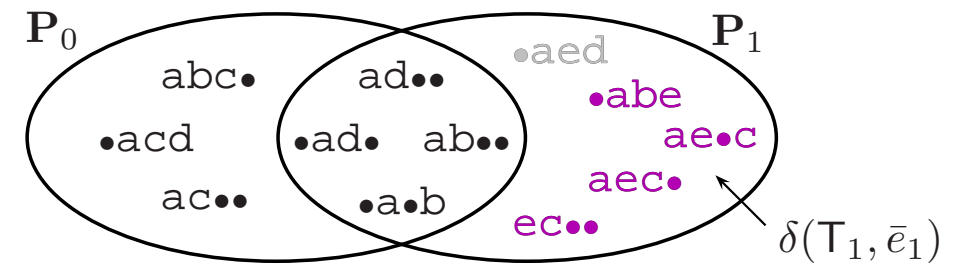
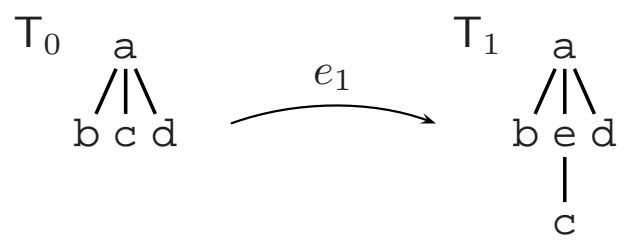
→ Compute  $\delta$  on wrong tree ( $T_2$ ) and fix it

$$\delta(T_1, \bar{e}_1) \cap P_2 = \delta(T_2, \bar{e}_1) \setminus \delta(T_2, \bar{e}_2)$$

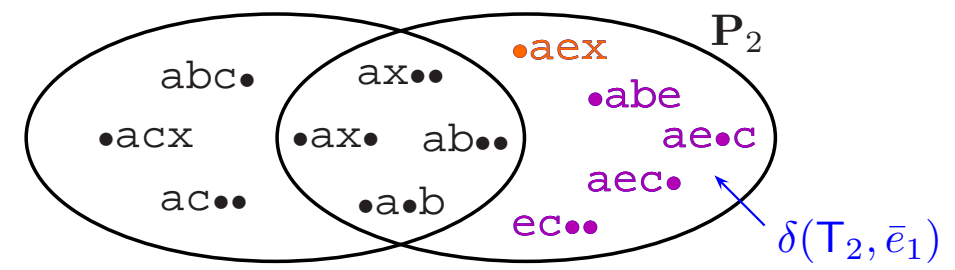
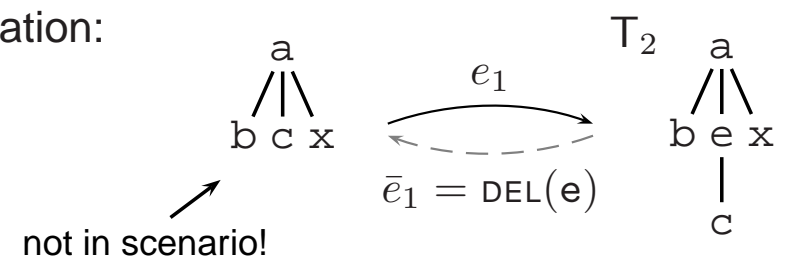
$$\Delta_2^+ = \delta(T_2, \bar{e}_1) \setminus \delta(T_2, \bar{e}_2) \cup \delta(T_2, \bar{e}_2)$$

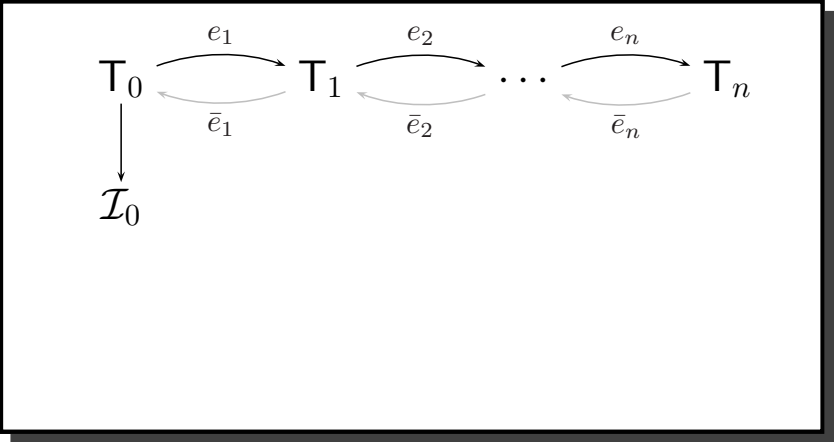
$$\Delta_2^+ = \delta(T_2, \bar{e}_1) \cup \delta(T_2, \bar{e}_2)$$

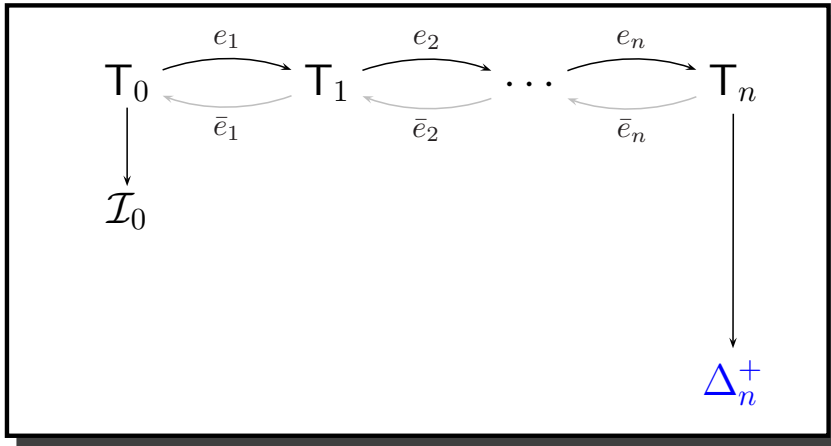
Reality:



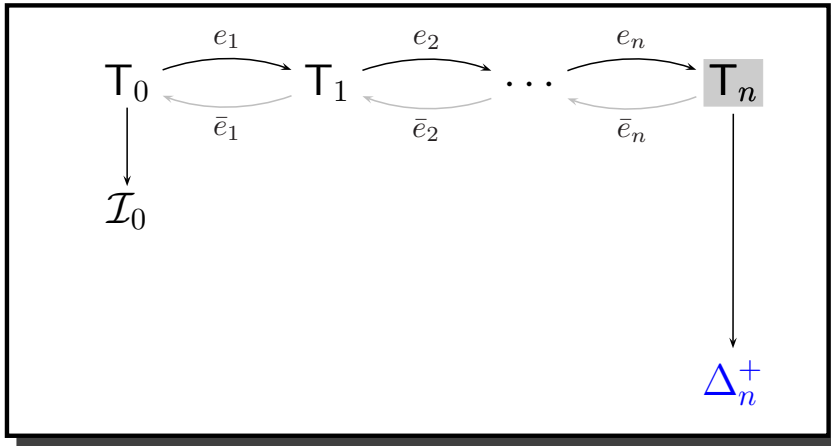
Computation:





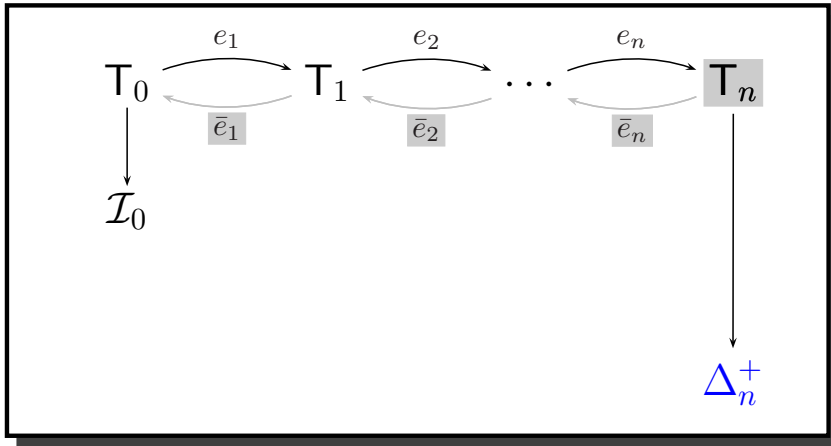


→ new *pq*-grams can be computed using:



⇒ new *pq*-grams can be computed using:

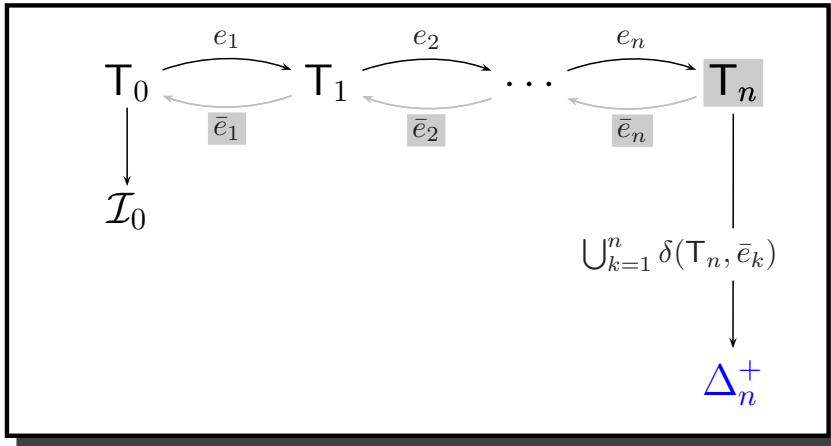
⇒ the resulting tree  $T_n$



⇒ **new  $pq$ -grams** can be computed using:

⇒ the resulting tree  $T_n$

⇒ the log  $(\bar{e}_1, \dots, \bar{e}_n)$



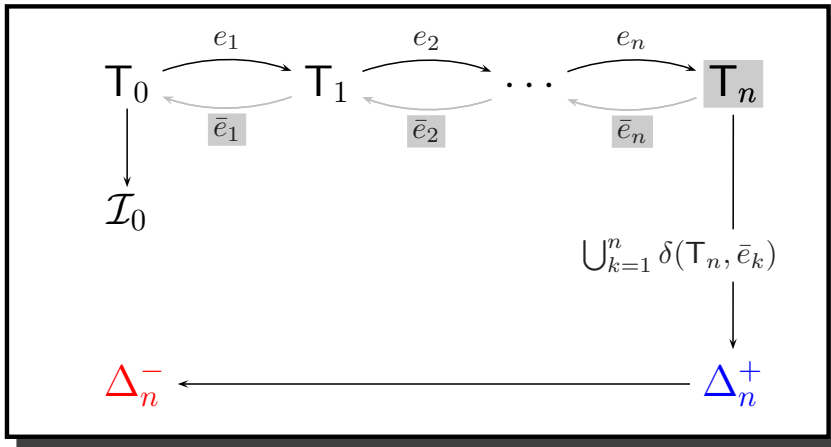
→ new *pq*-grams can be computed using:

⇒ the resulting tree  $T_n$

⇒ the log  $(\bar{e}_1, \dots, \bar{e}_n)$

**Theorem 1** The set of *new pq*-grams can be computed on  $T_n$  as

$$\Delta_n^+ = \bigcup_{k=1}^n \delta(T_n, \bar{e}_k)$$



→ new *pq-grams* can be computed using:

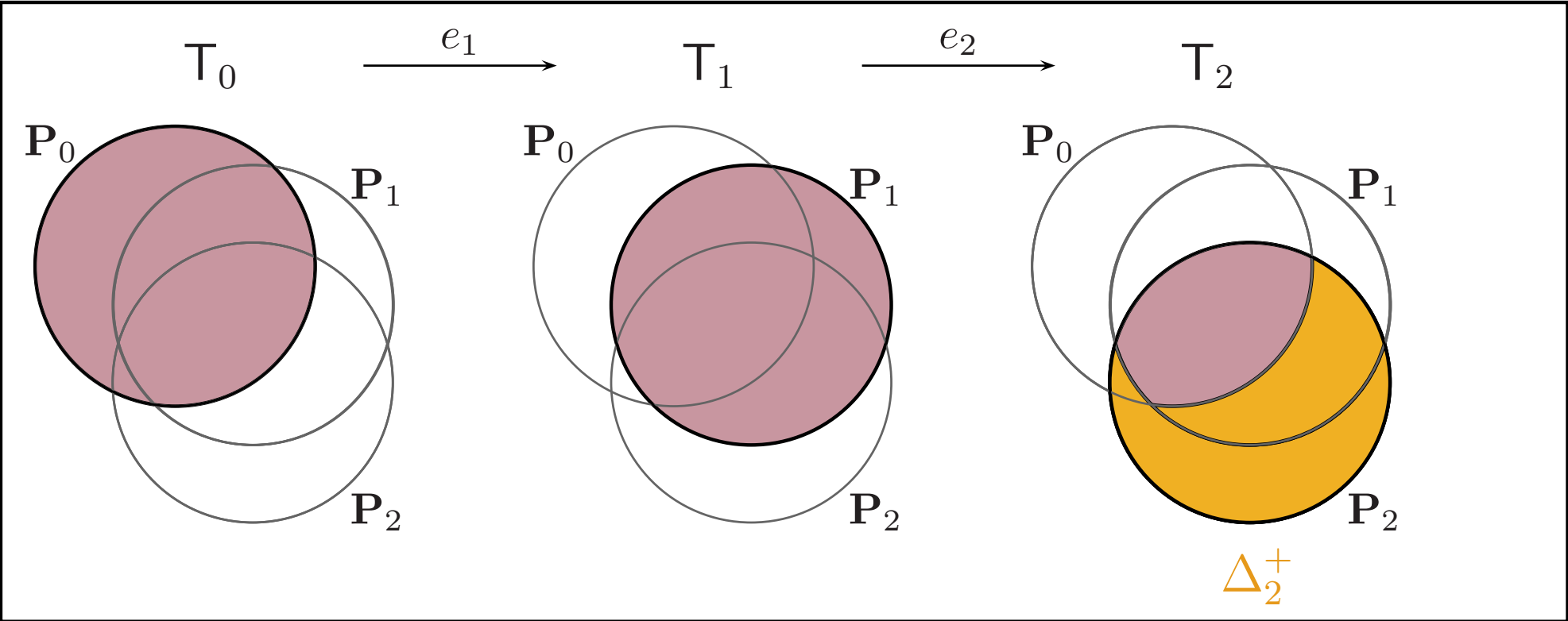
⇒ the resulting tree  $T_n$

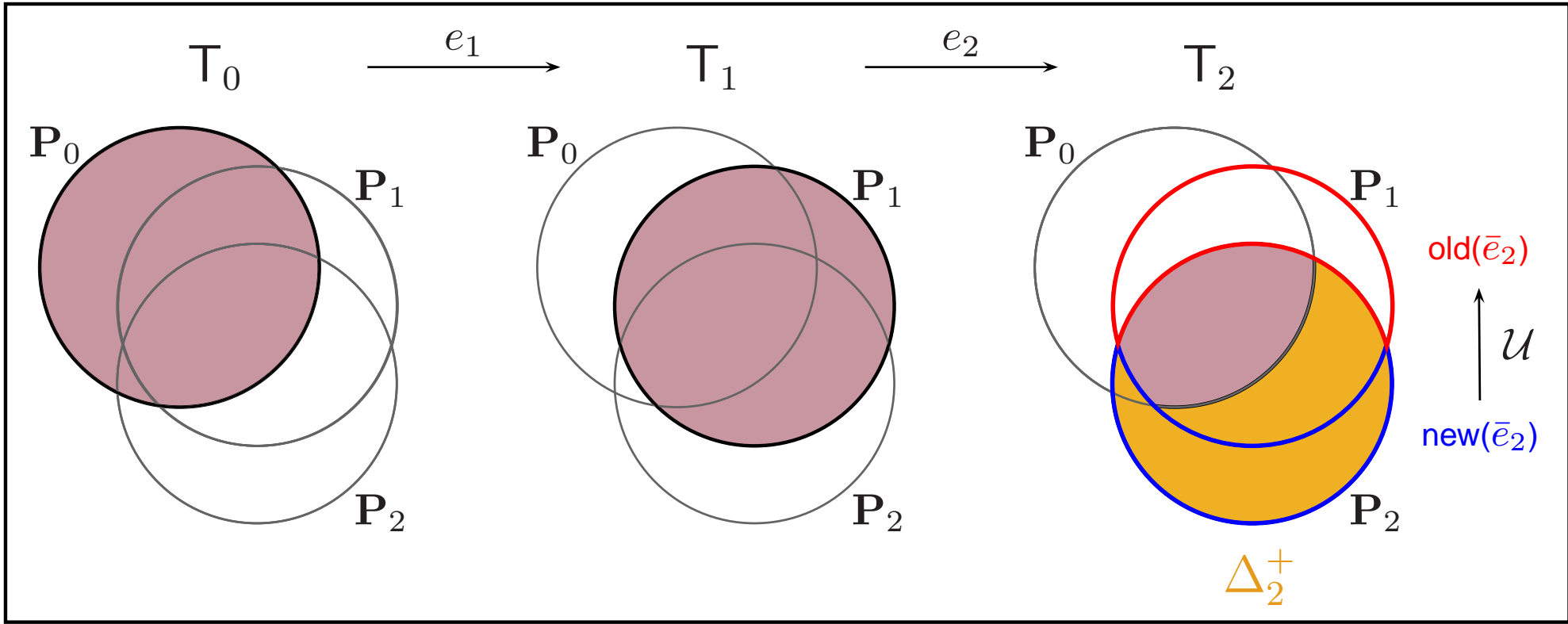
⇒ the log  $(\bar{e}_1, \dots, \bar{e}_n)$

**Theorem 1** The set of *new pq-grams* can be computed on  $T_n$  as

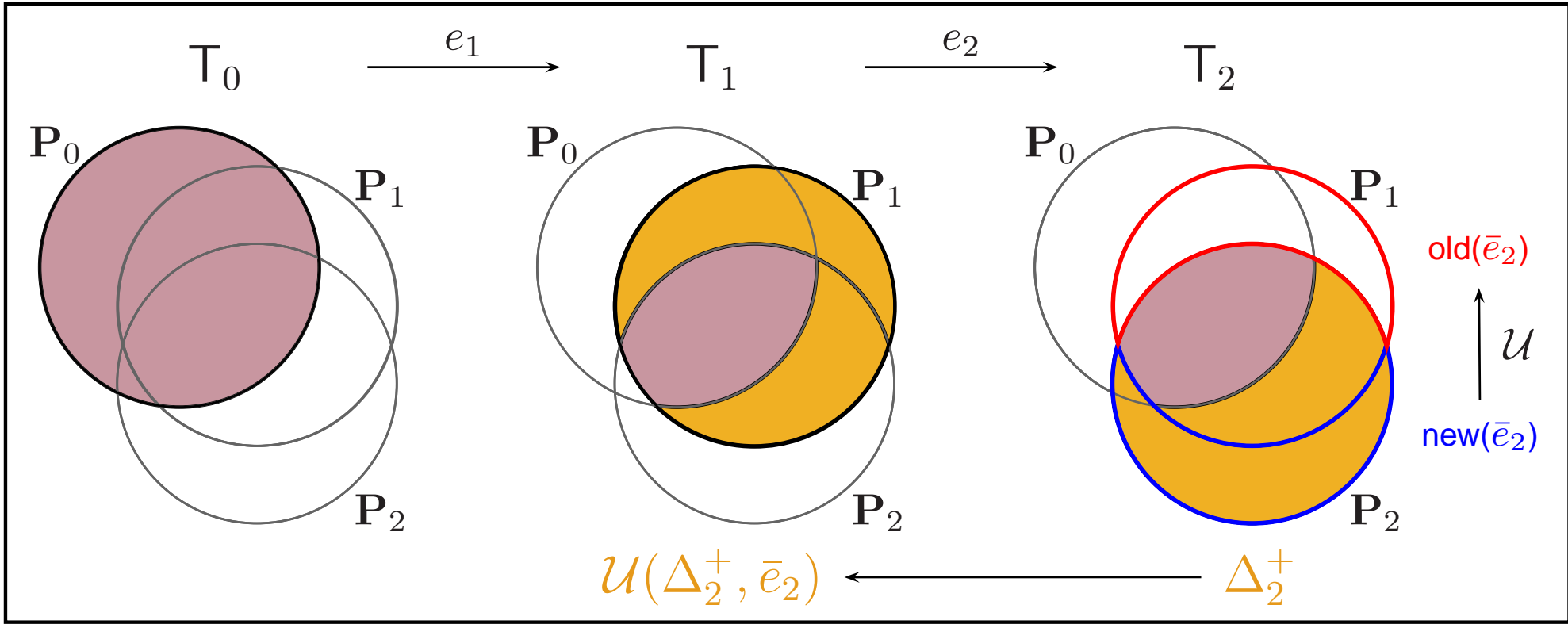
$$\Delta_n^+ = \bigcup_{k=1}^n \delta(T_n, \bar{e}_k)$$

→ **Next step:** compute *old pq-grams*  $\Delta_n^-$

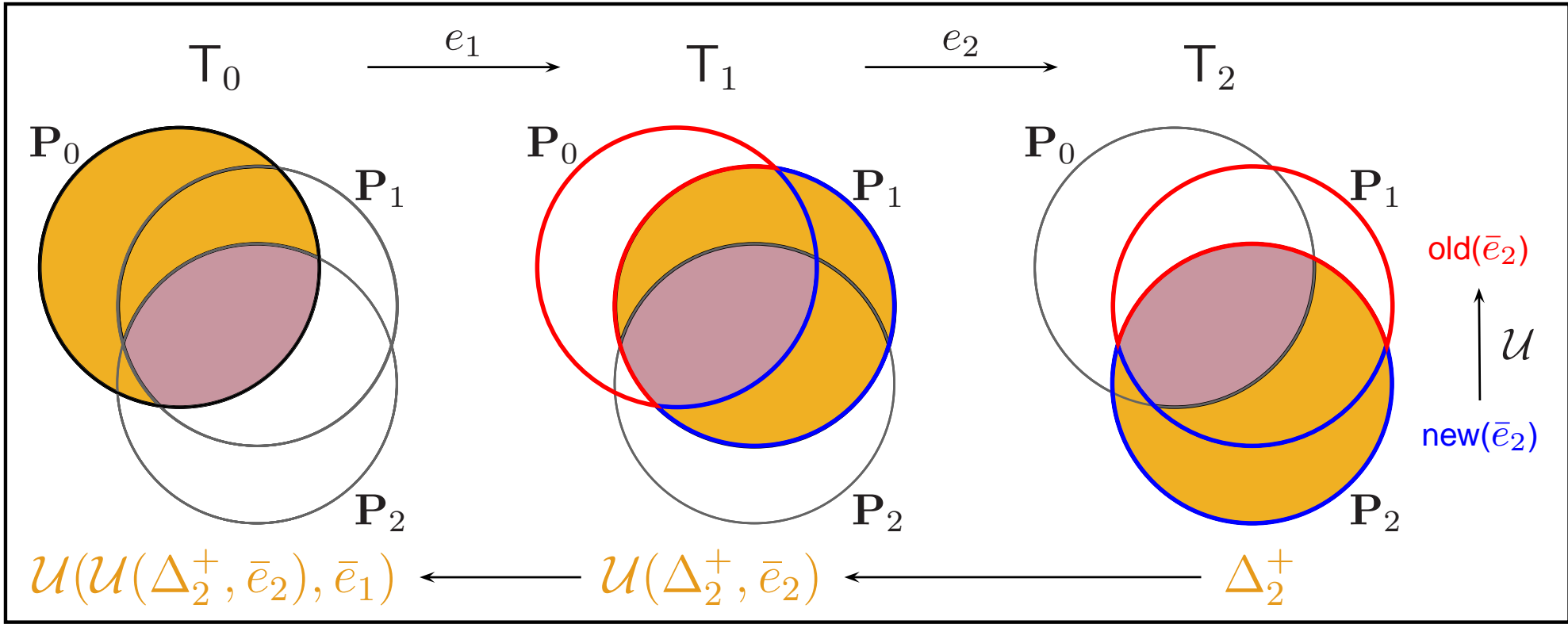




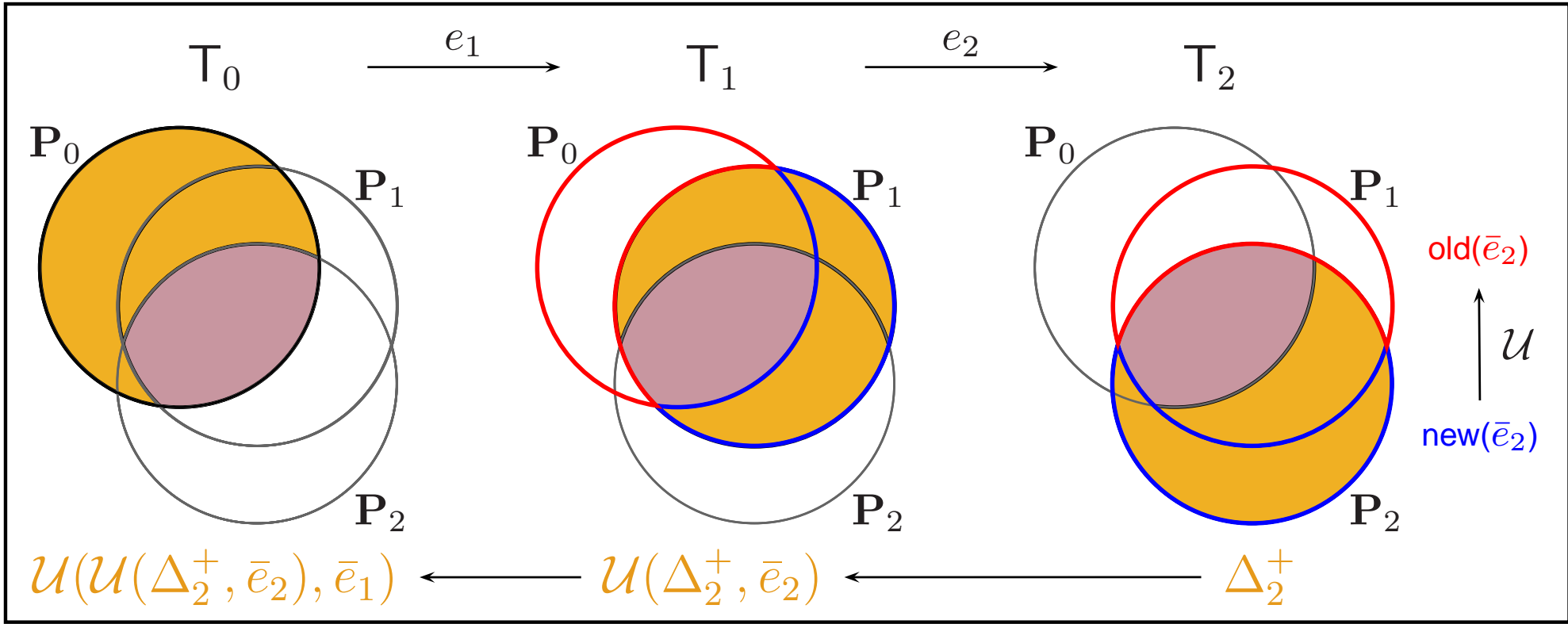
👉 Profile update function  $\mathcal{U}$ : transforms new  $pq$ -grams into old  $pq$ -grams.



👉 Profile update function  $\mathcal{U}$ : transforms new  $pq$ -grams into old  $pq$ -grams.



☞ **Profile update function  $\mathcal{U}$ :** transforms **new**  $pq$ -grams into **old**  $pq$ -grams.



👉 Profile update function  $\mathcal{U}$ : transforms new  $pq$ -grams into old  $pq$ -grams.

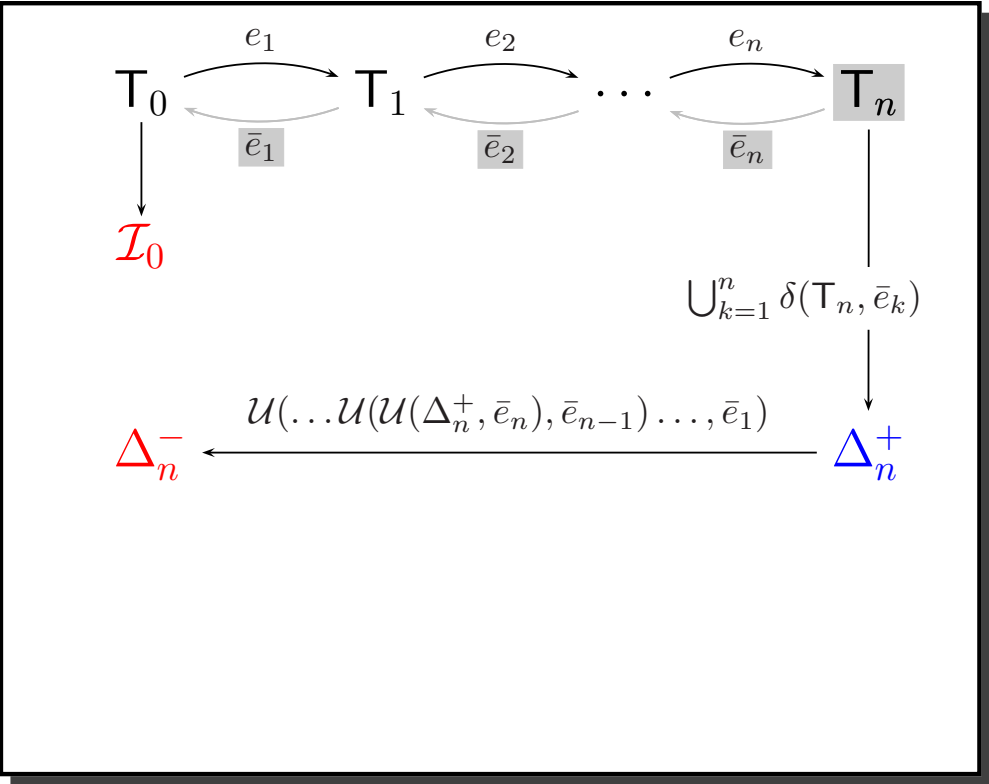
$$\mathcal{U}(\mathcal{U}(\Delta_2^+, \bar{e}_2), \bar{e}_1) = \Delta_2^-$$

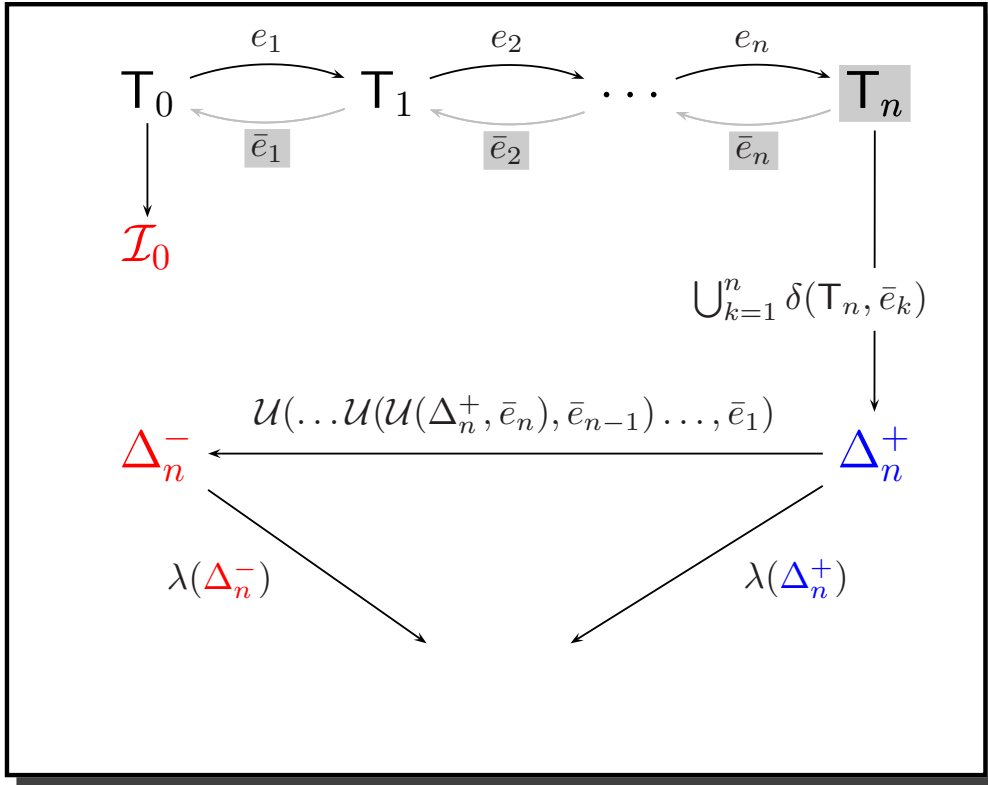
---

**Theorem 2** *The **old pq-grams**  $\Delta_n^-$  are computed by recursively applying the **update function**  $\mathcal{U}$  to the **new pq-grams**  $\Delta_n^+$ :*

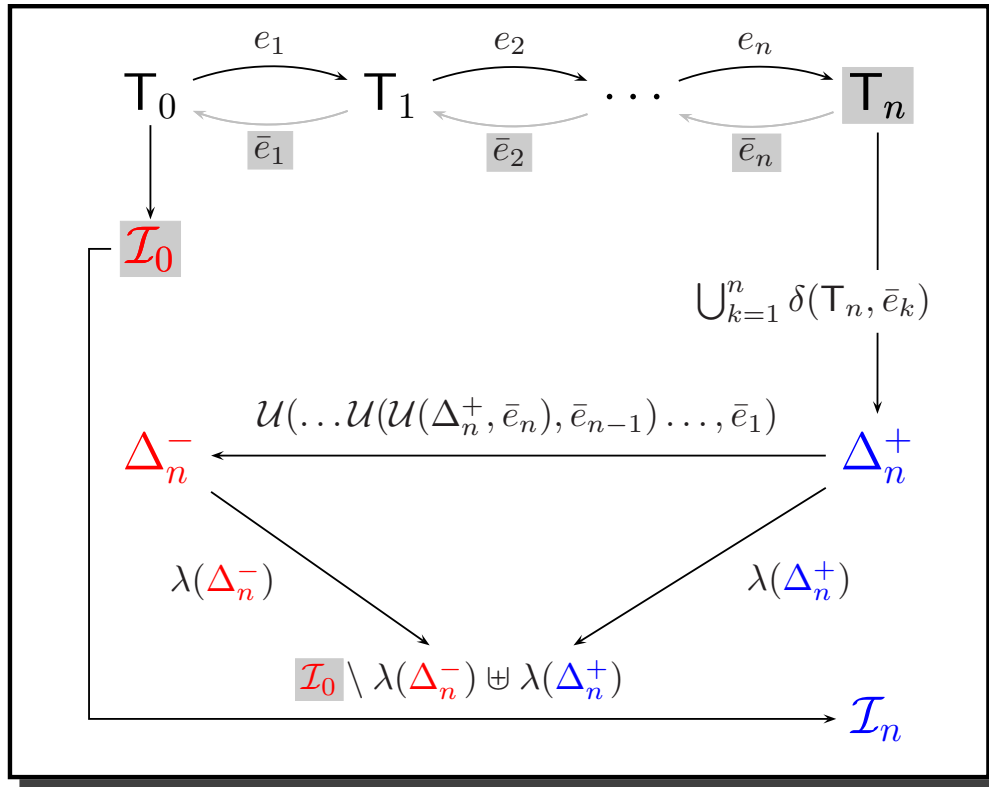
$$\Delta_n^- = \mathcal{U}(\dots \mathcal{U}(\mathcal{U}(\Delta_n^+, \bar{e}_n), \bar{e}_{n-1}) \dots, \bar{e}_1).$$

---





→ Hash old ( $\Delta_n^+$ ) and new ( $\Delta_n^-$ )  $pq$ -grams



➡ Hash old ( $\Delta_n^+$ ) and new ( $\Delta_n^-$ )  $pq$ -grams

➡ Update old index  $\mathcal{I}_0$

---

**Algorithm 1:**  $\text{updateIndex}(I_0, T_n, \log)$

---

---

---

**Algorithm 1:**  $\text{updateIndex}(I_0, T_n, \log)$ 

---

1  $\Delta_n^+ = \emptyset;$ 2 **foreach**  $\bar{e}_i \in \log$  **do**  $\Delta_n^+ \leftarrow \Delta_n^+ \cup \delta(T_n, \bar{e}_i);$

---

**Algorithm 1:**  $\text{updateIndex}(I_0, T_n, \log)$ 

---

- 1  $\Delta_n^+ = \emptyset;$
  - 2 **foreach**  $\bar{e}_i \in \log$  **do**  $\Delta_n^+ \leftarrow \Delta_n^+ \cup \delta(T_n, \bar{e}_i);$
  - 3  $\Delta_n^- \leftarrow \Delta_n^+;$
  - 4 **for**  $\bar{e}_i \leftarrow \bar{e}_n$  **downto**  $\bar{e}_1$  **do**  $\Delta_n^- \leftarrow \mathcal{U}(\Delta_n^-, \bar{e}_i);$
-

---

**Algorithm 1:** updateIndex( $I_0, T_n, log$ )

---

- 1  $\Delta_n^+ = \emptyset$ ;
  - 2 **foreach**  $\bar{e}_i \in log$  **do**  $\Delta_n^+ \leftarrow \Delta_n^+ \cup \delta(T_n, \bar{e}_i)$ ;
  - 3  $\Delta_n^- \leftarrow \Delta_n^+$ ;
  - 4 **for**  $\bar{e}_i \leftarrow \bar{e}_n$  **downto**  $\bar{e}_1$  **do**  $\Delta_n^- \leftarrow \mathcal{U}(\Delta_n^-, \bar{e}_i)$ ;
  - 5  $I^+ \leftarrow \lambda(\Delta_n^+)$ ;
  - 6  $I^- \leftarrow \lambda(\Delta_n^-)$ ;
-

---

**Algorithm 1:** updateIndex( $I_0, T_n, log$ )

---

```
1  $\Delta_n^+ = \emptyset$ ;  
2 foreach  $\bar{e}_i \in log$  do  $\Delta_n^+ \leftarrow \Delta_n^+ \cup \delta(T_n, \bar{e}_i)$ ;  
3  $\Delta_n^- \leftarrow \Delta_n^+$ ;  
4 for  $\bar{e}_i \leftarrow \bar{e}_n$  downto  $\bar{e}_1$  do  $\Delta_n^- \leftarrow \mathcal{U}(\Delta_n^-, \bar{e}_i)$ ;  
5  $I^+ \leftarrow \lambda(\Delta_n^+)$ ;  
6  $I^- \leftarrow \lambda(\Delta_n^-)$ ;  
7  $I_n \leftarrow I_0 \setminus I^- \cup I^+$ ;  
8 return  $I_n$ ;
```

---

---

**Algorithm 1:** updateIndex( $I_0, T_n, log$ )
 

---

```

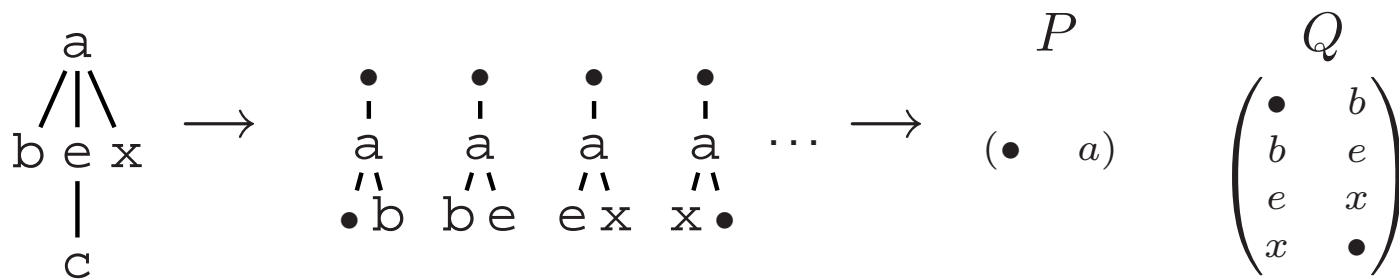
1  $\Delta_n^+ = \emptyset;$ 
2 foreach  $\bar{e}_i \in log$  do  $\Delta_n^+ \leftarrow \Delta_n^+ \cup \delta(T_n, \bar{e}_i);$ 
3  $\Delta_n^- \leftarrow \Delta_n^+;$ 
4 for  $\bar{e}_i \leftarrow \bar{e}_n$  downto  $\bar{e}_1$  do  $\Delta_n^- \leftarrow \mathcal{U}(\Delta_n^-, \bar{e}_i);$ 
5  $I^+ \leftarrow \lambda(\Delta_n^+);$ 
6  $I^- \leftarrow \lambda(\Delta_n^-);$ 
7  $I_n \leftarrow I_0 \setminus I^- \cup I^+;$ 
8 return  $I_n;$ 

```

---

### Implementation

☞ Store  $p$ -part and  $q$ -part in tables  $P$  and  $Q$



**Insert** node  $n$  as the  $k$ -th child of node  $v$ :  $\text{INS}(n, v, k, m)$

$$\delta(\mathbf{T}_j, \bar{e}) = P(v) \circ Q^{k..m}(v) \cup P(x) \circ Q(x)$$

$$\forall x \in \text{desc}_{p-2}(c_k, \dots, c_m)$$

$$\begin{aligned} \mathcal{U}(\delta(\mathbf{T}_j, \bar{e}), \bar{e}) &= P(v) \circ [Q^{k..m}(v) // D(n)] \cup P^{+n,0}(v) \circ \\ &[D(\bullet) // Q^{k..m}(v)] \cup P^{+n,d}(x) \circ Q(x) \end{aligned}$$

$$\forall x \in \text{desc}_{p-2}(c_k, \dots, c_m), d = \text{dist}(c_i, x) + 1$$

$c_i$  :  $i$ -th child of  $v$

**Delete** node  $n$ ,  $\text{DEL}(n)$ :

$$\delta(\mathbf{T}_j, \bar{e}) = P(v) \circ Q^{k..k}(v) \cup P(x) \circ Q(x)$$

$$\forall x \in \text{desc}_{p-1}(n)$$

$$\mathcal{U}(\delta(\mathbf{T}_j, \bar{e}), \bar{e}) = P(v) \circ [Q^{k..k}(v) // Q(n)] \cup P^{-n}(x) \circ Q(x)$$

$$\forall x \in \text{desc}_{p-1}(n) \setminus \{n\}$$

$v$  :  $n$  is the  $k$ -th child of  $v$

**Rename** node  $n$  to  $l'$ :  $\text{REN}(n, l')$

$$\delta(\mathbf{T}_j, \bar{e}) = P(v) \circ Q^{k..k}(v) \cup P(x) \circ Q(x)$$

$$\forall x \in \text{desc}_{p-1}(n)$$

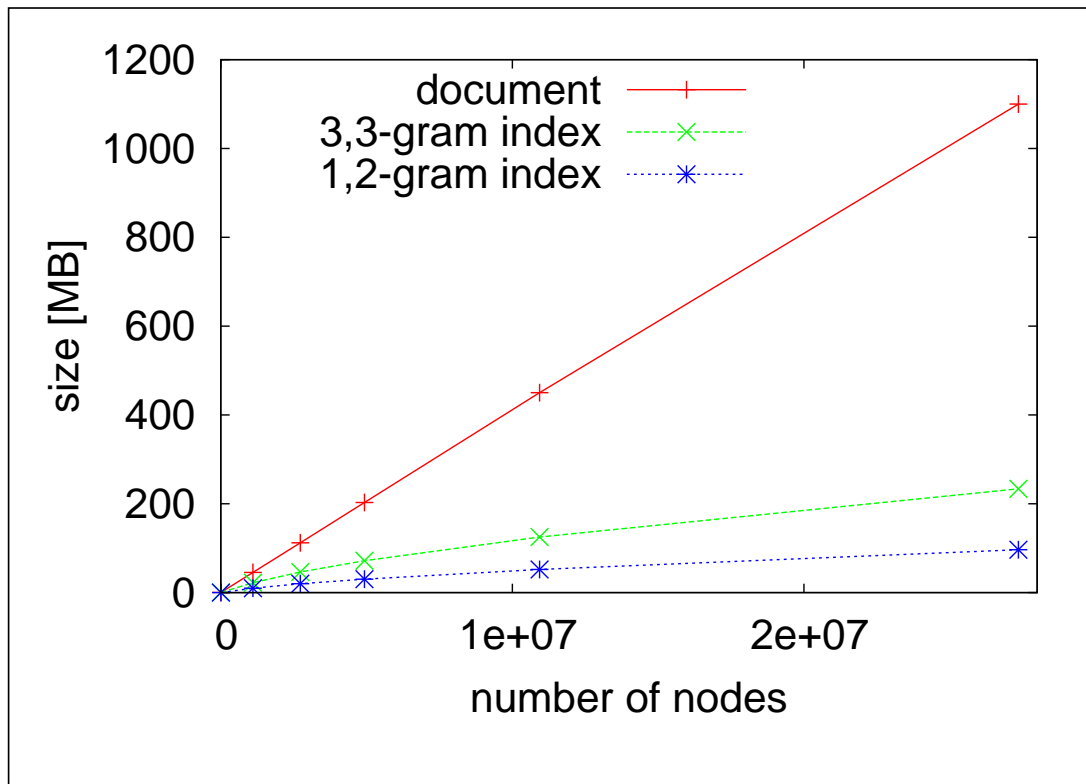
$$\mathcal{U}(\delta(\mathbf{T}_j, \bar{e}), \bar{e}) = P(v) \circ [Q^{k..k}(v) // D(m)] \cup P^{n/m}(x) \circ Q(x)$$

$$\forall x \in \text{desc}_{p-1}(n)$$

$m = (\text{id}(n), l')$   $v$  :  $n$  is the  $k$ -th child of  $v$



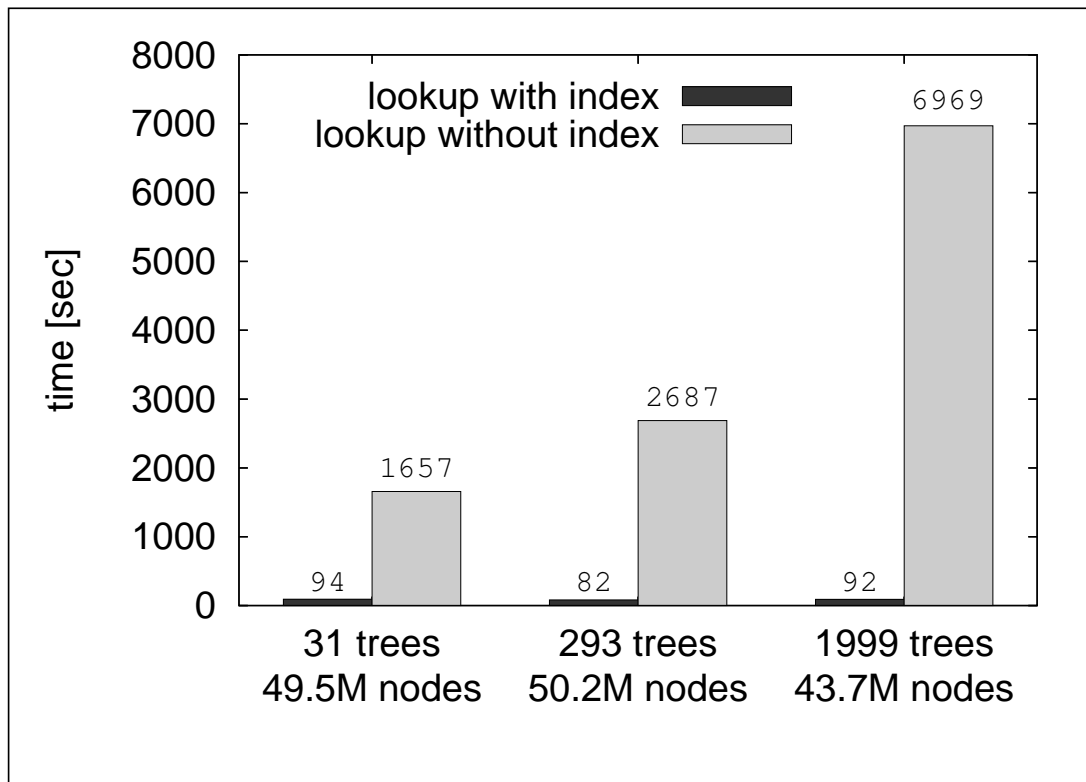
**Index size linear in tree size**



- ☞ Experiment: **synthetic** data (XMark)
- ▣ vary document size
- ▣ compute index
- ▣ compare index size with tree size



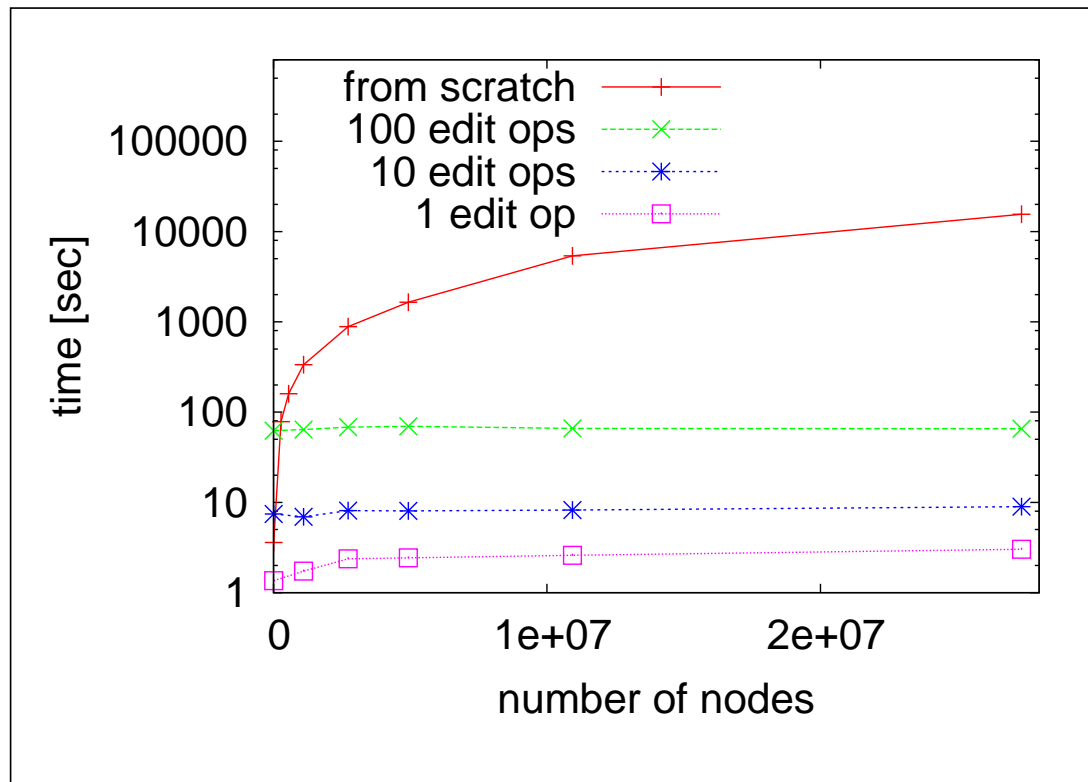
**Index greatly increases efficiency for lookup**



- 👉 Experiment: **synthetic** data (XMark)
- ▣➡ lookup with/without index
- ▣➡ different document sets of similar size
- ▣➡ measure wall clock time

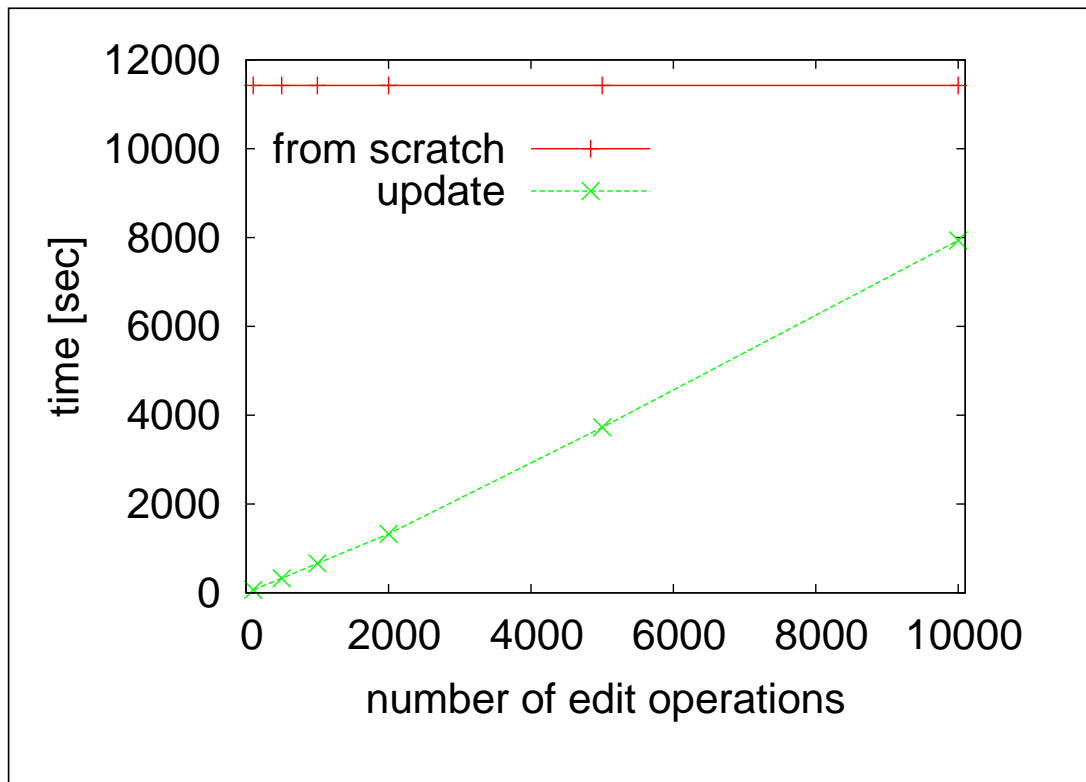


## Update independent of tree size



- Experiment: **synthetic** data (XMark)
- ⇒ vary tree size (up to 27M nodes)
- ⇒ compute incremental update
- ⇒ compute index from scratch
- ⇒ measure wall clock time



**Update linear in log size**

- 👉 Experiment: **DBLP** (211MB, 11M nodes)
- ⇒ vary number of edit operations
- ⇒ compute incremental update
- ⇒ compute index from scratch
- ⇒ measure wall clock time



Action	Number of edit operations			
	1	10	100	1000
$\Delta_n^+$	0.642s	3.903s	37.533s	391.513s
$I^+ = \lambda(\Delta_n^+)$	0.184s	0.199s	0.287s	0.443s
$\Delta_n^-$	0.196s	2.836s	27.967s	295.104s
$I^- = \lambda(\Delta_n^-)$	0.177s	0.191s	0.185s	0.383s
$I_0 \setminus I^- \cup I^+$	2.206s	2.770s	6.475s	19.780s
total	3.405s	9.900s	72.448s	707.224s

👉 Experiment: **DBLP** (211MB, 11M nodes)

⇒ vary number of edit operations

⇒ compute incremental update

⇒ wall clock time for each steps

$\Delta^+$ ,  $\Delta^-$ : main share

Action	Number of edit operations			
	1	10	100	1000
$\Delta_n^+$	0.642s	3.903s	37.533s	391.513s
$I^+ = \lambda(\Delta_n^+)$	0.184s	0.199s	0.287s	0.443s
$\Delta_n^-$	0.196s	2.836s	27.967s	295.104s
$I^- = \lambda(\Delta_n^-)$	0.177s	0.191s	0.185s	0.383s
$I_0 \setminus I^- \cup I^+$	2.206s	2.770s	6.475s	19.780s
total	3.405s	9.900s	72.448s	707.224s

Experiment: **DBLP** (211MB, 11M nodes)

⇒ vary number of edit operations

⇒ compute incremental update

⇒ wall clock time for each steps

$\Delta^+$ ,  $\Delta^-$ : main share

Hashing very cheap

Action	Number of edit operations			
	1	10	100	1000
$\Delta_n^+$	0.642s	3.903s	37.533s	391.513s
$I^+ = \lambda(\Delta_n^+)$	0.184s	0.199s	0.287s	0.443s
$\Delta_n^-$	0.196s	2.836s	27.967s	295.104s
$I^- = \lambda(\Delta_n^-)$	0.177s	0.191s	0.185s	0.383s
$I_0 \setminus I^- \cup I^+$	2.206s	2.770s	6.475s	19.780s
total	3.405s	9.900s	72.448s	707.224s

Experiment: **DBLP** (211MB, 11M nodes)

⇒ vary number of edit operations

⇒ compute incremental update

⇒ wall clock time for each steps

$\Delta^+, \Delta^-$ : main share

Hashing very cheap

$\mathcal{I}_n = \mathcal{I}_0 \setminus \mathcal{I}^- \cup \mathcal{I}^+$ : sublinear in log size

Action	Number of edit operations			
	1	10	100	1000
$\Delta_n^+$	0.642s	3.903s	37.533s	391.513s
$\mathcal{I}^+ = \lambda(\Delta_n^+)$	0.184s	0.199s	0.287s	0.443s
$\Delta_n^-$	0.196s	2.836s	27.967s	295.104s
$\mathcal{I}^- = \lambda(\Delta_n^-)$	0.177s	0.191s	0.185s	0.383s
$\mathcal{I}_0 \setminus \mathcal{I}^- \cup \mathcal{I}^+$	2.206s	2.770s	6.475s	19.780s
total	3.405s	9.900s	72.448s	707.224s

Experiment: **DBLP** (211MB, 11M nodes)

⇒ vary number of edit operations

⇒ compute incremental update

⇒ wall clock time for each steps



- ☞ ***pq*-Gram distance** (Augsten et al., VLDB 2005)
  - ▣ edit distance approximation
  - ▣ properties analyzed in (Augsten et al., VLDB 2005)

☞ ***pq*-Gram distance** (Augsten et al., VLDB 2005)

▣ edit distance approximation

▣ properties analyzed in (Augsten et al., VLDB 2005)

☞ **Other edit distance approximations:**

▣ (Garofalakis & Kumar, TODS 2005): *approximation guarantees* with tree edit distance embedding

▣ (Yang et al., SIGMOD 2005): *lower bound* for edit distance

▣ (Weis and Naumann, SIGMOD 2005): *XML duplicate detection*

▣ (Cobéna et al., ICDE 2002), (Lee et al., TKDE 2004): *XML change detection*

☞ ***pq*-Gram distance** (Augsten et al., VLDB 2005)

▣ edit distance approximation

▣ properties analyzed in (Augsten et al., VLDB 2005)

☞ **Other edit distance approximations:**

▣ (Garofalakis & Kumar, TODS 2005): *approximation guarantees* with tree edit distance embedding

▣ (Yang et al., SIGMOD 2005): *lower bound* for edit distance

▣ (Weis and Naumann, SIGMOD 2005): *XML duplicate detection*

▣ (Cobéna et al., ICDE 2002), (Lee et al., TKDE 2004): *XML change detection*

☞ **Approximate XML join** by (Guha et al., SIGMOD 2002), **index** by (Guha et al., ICDE 2003)

▣ optimize join based on tree edit distance using reference sets

▣ updates of reference sets not addressed



☞ **Proof for incremental update** of *pq*-gram index with

▣▣▣▣▶ old index

▣▣▣▣▶ resulting document

▣▣▣▣▶ edit-log

☞ **Proof for incremental update** of  $pq$ -gram index with

- ▣▶ old index
- ▣▶ resulting document
- ▣▶ edit-log

☞ **Update efficient**

- ▣▶ constant in tree size
- ▣▶ linear in edit-log size

☞ **Proof for incremental update** of  $pq$ -gram index with

- ▣▶ old index
- ▣▶ resulting document
- ▣▶ edit-log

☞ **Update efficient**

- ▣▶ constant in tree size
- ▣▶ linear in edit-log size

☞ **Future work**

- ▣▶ optimize edit-log (e.g. remove redundancy)
- ▣▶ subtree edit operations (e.g. subtree move)
- ▣▶ compute updates for edit ops in parallel