Defensa de tesis para optar al grado de Doctor en Ciencias de la Computación

Compact data structures and query processing for temporal graphs

Candidato: Diego Caro Alarcón

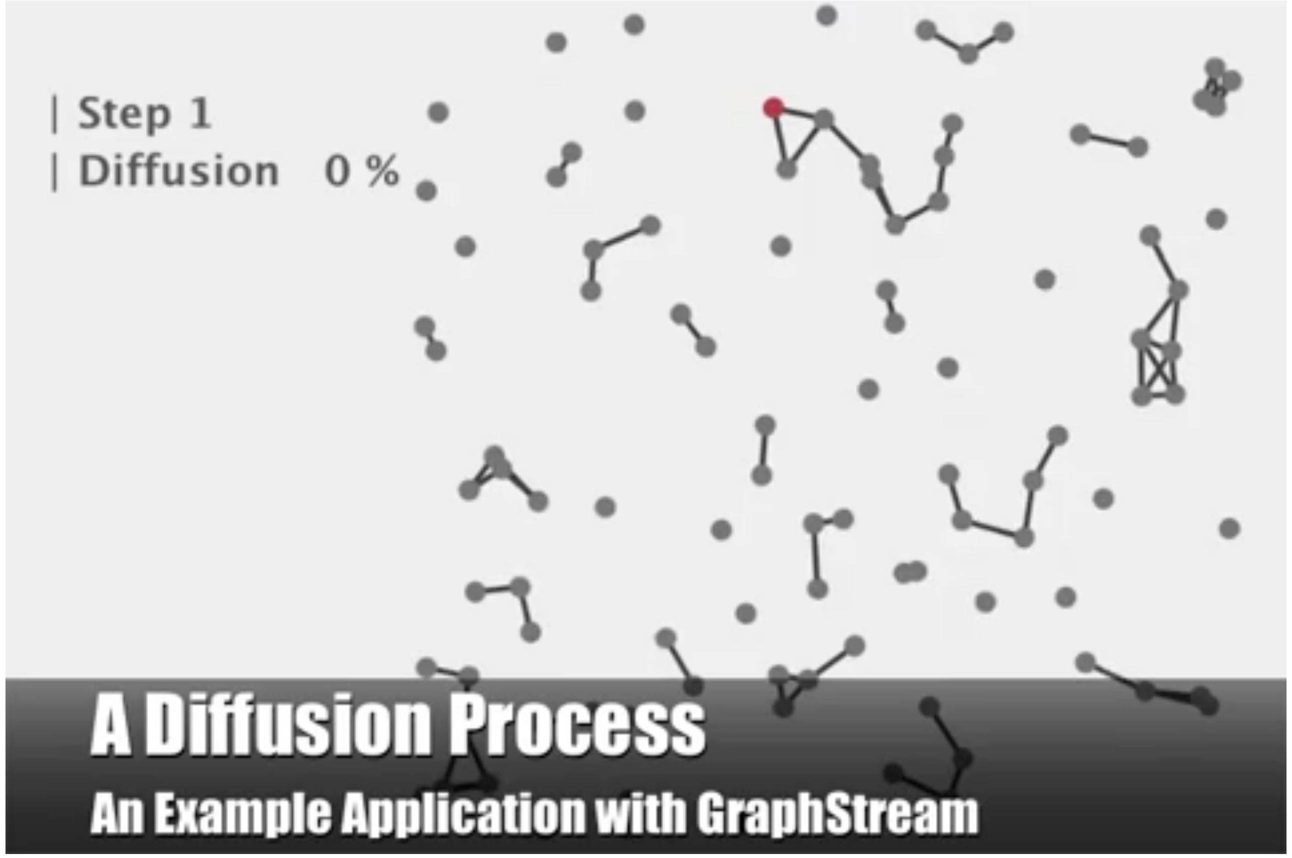
Profesoras guía:
- Dra. M. Andrea Rodríguez
- Dra. Nieves R. Brisaboa

Outline

- Definition and Motivation.
- Previous works about temporal graphs.
- Compression of temporal graphs.
- Contributions.
- Evaluation.
- Conclusions and future works.

Motivation for Temporal Graphs

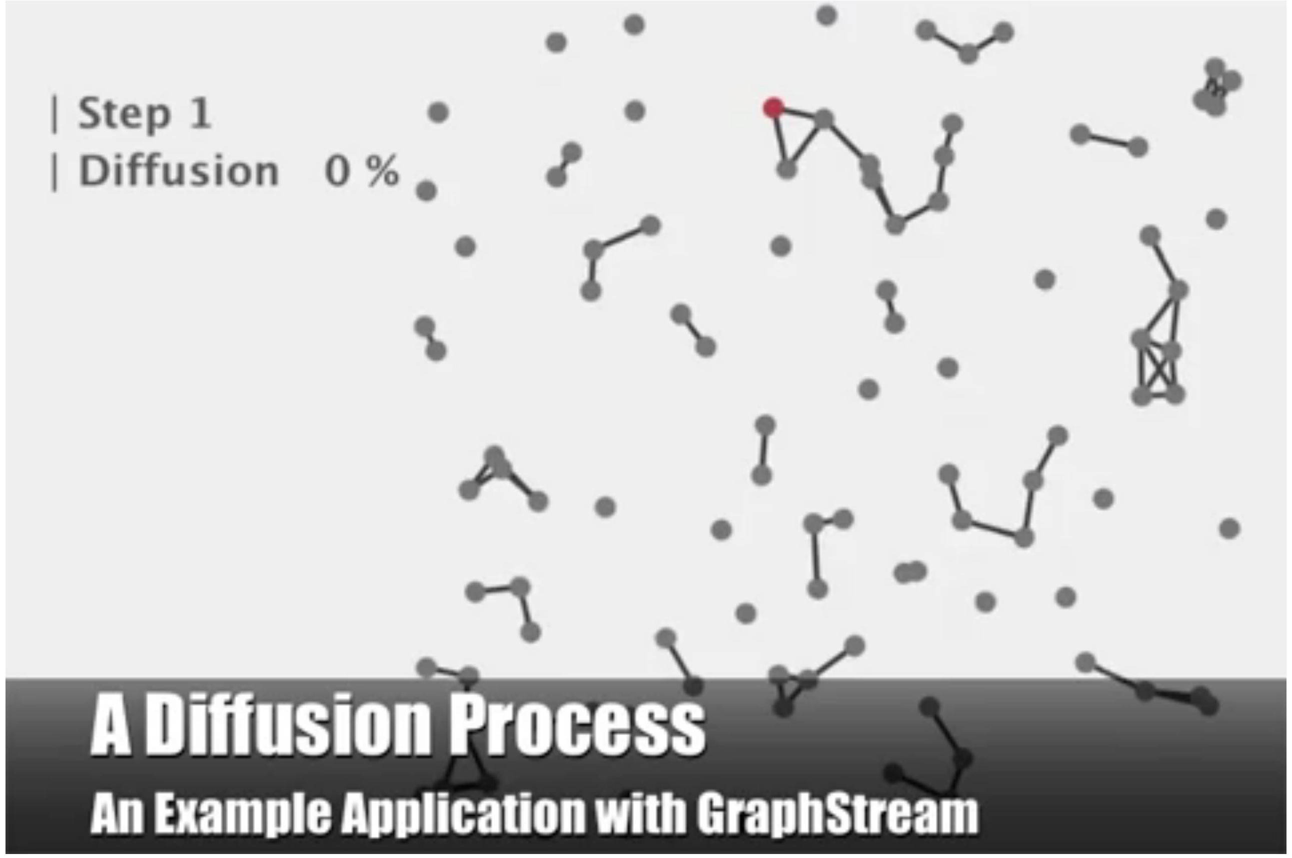
- Temporal graphs are graphs whose edges appear and disappear along time.
  - Diffusion in a network.
  - Evolution of friendship in social networks.
  - Evolution of links between web pages.
- The interest is not only the current state, but also the historical states of the graph.



https://www.youtube.com/watch?v=wjUnT2dxPyk

Motivation for Temporal Graphs

- Temporal graphs are graphs whose edges appear and disappear along time.
  - Diffusion in a network.
  - Evolution of friendship in social networks.
  - Evolution of links between web pages.
- The interest is not only the current state, but also the historical states of the graph.



https://www.youtube.com/watch?v=wjUnT2dxPyk

Temporal graphs concepts

- Temporal graph: set of contacts between a pair of vertices (edges).
- An edge (u,v) is active at time instant t if there is a contact (u,v,t_1,t_2) where t is in [t_1,t_2).

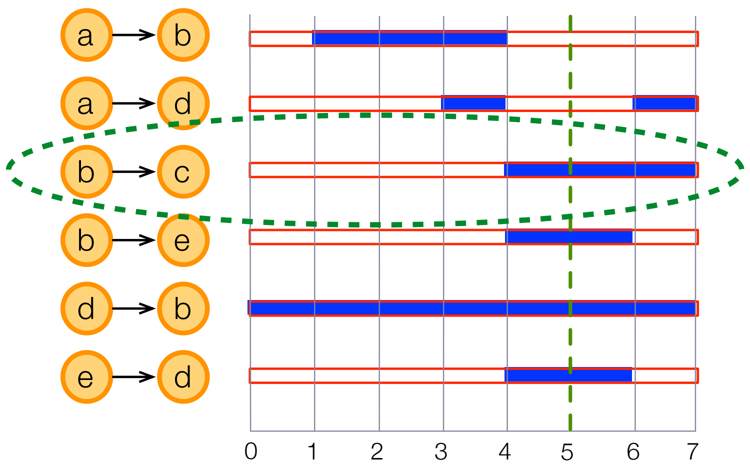
Vertices V = {a,b,c,d,e}

Edges E = {ab, ad, bc,
be, db, ed}

Time T = {0,1,2,3,4,5,6,7}

Contacts

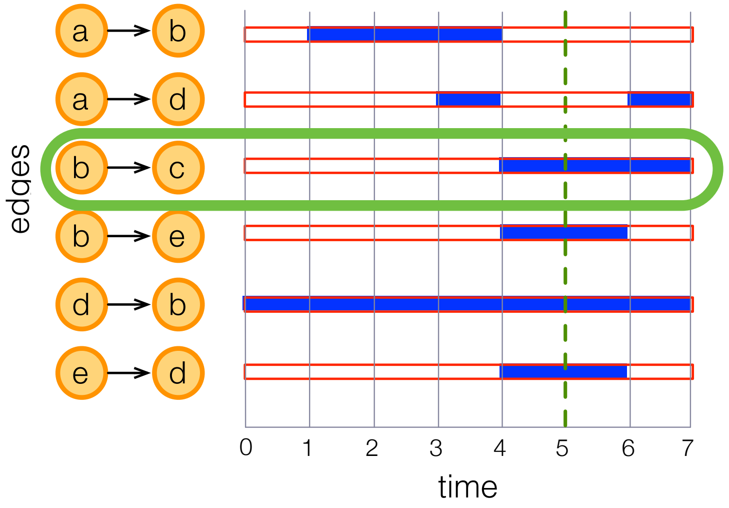
C = {(a,b,1,4),
(a,d,3,4),
(a,d,6,7),
(b,c,4,7),
(b,e,4,6),
(d,b,0,7),
(e,d,4,6)}



Operations over Temporal Graphs

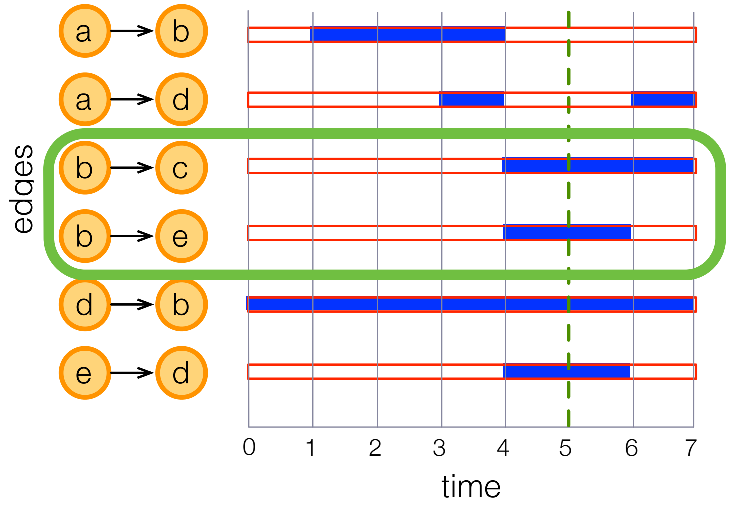
Operations over Temporal Graphs

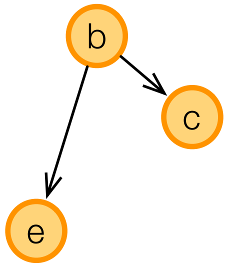
1. About **edges**: check if an edge is active at a time instant or time interval.



Operations over Temporal Graphs

1. About **edges**: check if an edge is active at a time instant or time interval.
2. About **vertices**: recover direct or reverse active neighbors at a time instant or time interval.

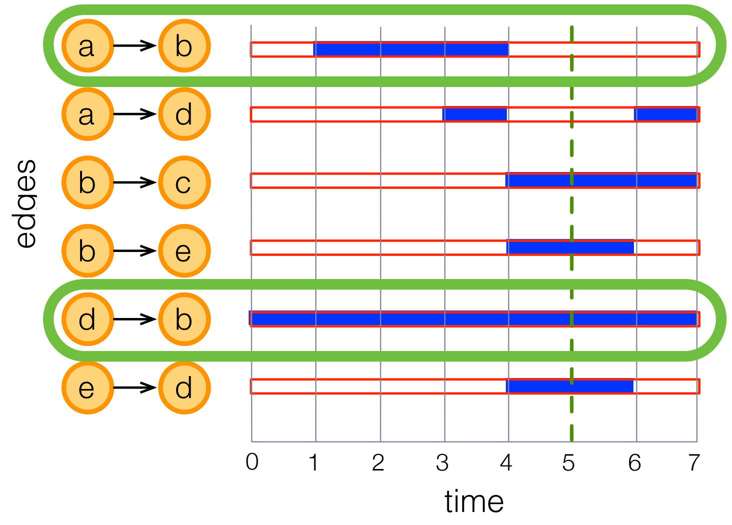


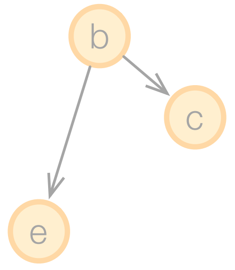


Direct Neighbors of **b** at time **5**

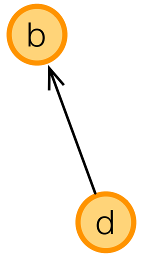
Operations over Temporal Graphs

1. About **edges**: check if an edge is active at a time instant or time interval.
2. About **vertices**: recover direct or reverse active neighbors at a time instant or time interval.





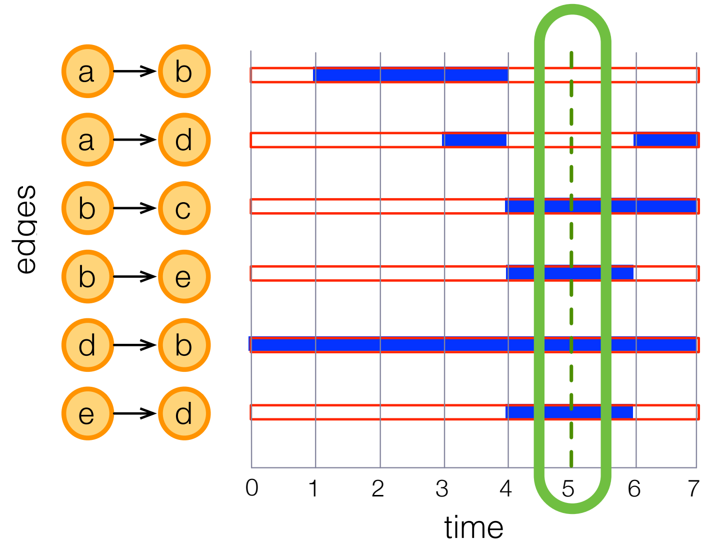
Direct Neighbors of **b** at time **5**

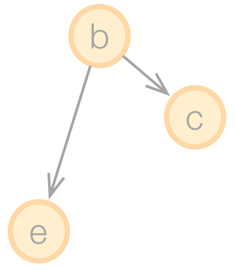


Reverse Neighbors of **b** at time **5**

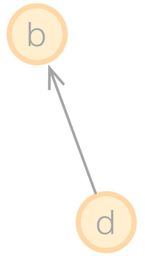
Operations over Temporal Graphs

1. About **edges**: check if an edge is active at a time instant or time interval.
2. About **vertices**: recover direct or reverse active neighbors at a time instant or time interval.
3. About the **state** of the graph: recover the set of active edges at a time instant (recover the snapshot).

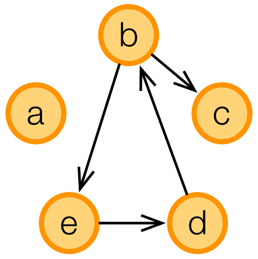




Direct Neighbors of **b** at time **5**



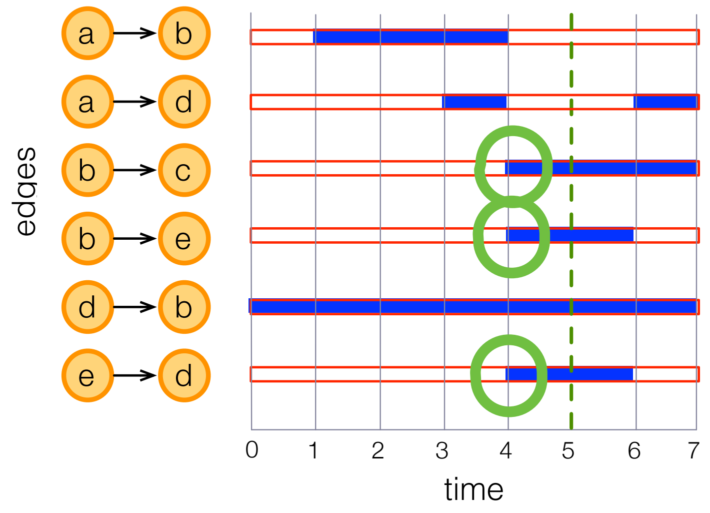
Reverse Neighbors of **b** at time **5**

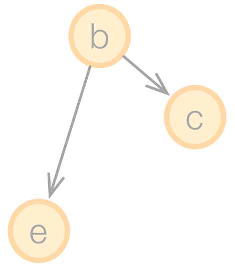


Snapshot at t=5

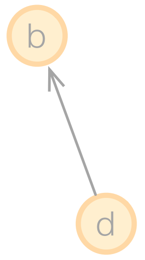
Operations over Temporal Graphs

1. About **edges**: check if an edge is active at a time instant or time interval.
2. About **vertices**: recover direct or reverse active neighbors at a time instant or time interval.
3. About the **state** of the graph: recover the set of active edges at a time instant (recover the snapshot).
4. About **events**: retrieve which edges were activated/deactivated at a time instant.

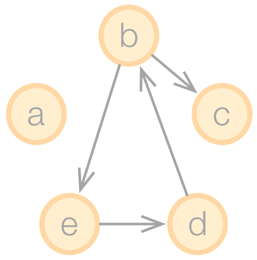




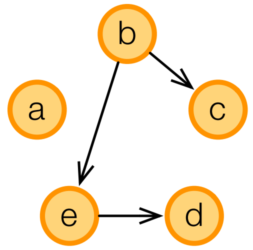
Direct Neighbors of **b** at time **5**



Reverse Neighbors of **b** at time **5**



Snapshot at t=5



Edges activated at time instant **4**

Outline

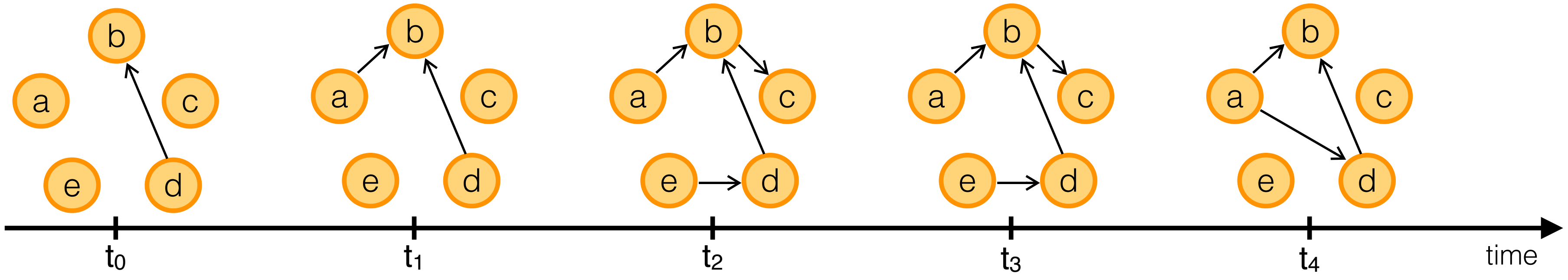
✓ Definition and Motivation.

- Previous works about temporal graphs.
- Compression of temporal graphs.
- Contributions.
- Evaluation.
- Conclusions and future works.

Typical representations

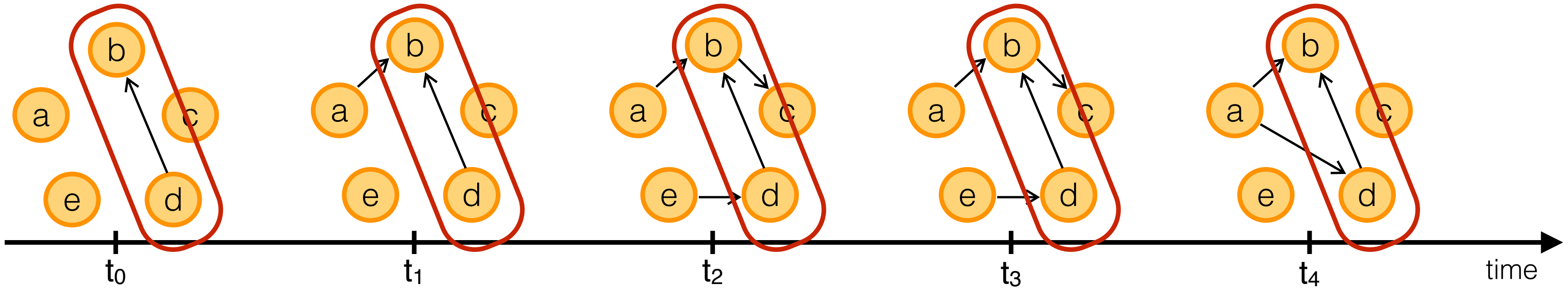
Typical representations

Snapshots:



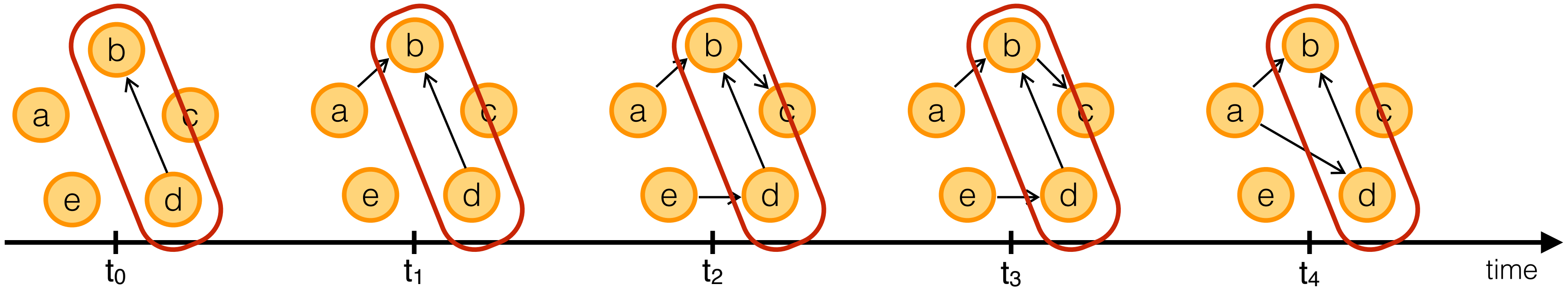
Typical representations

Snapshots:

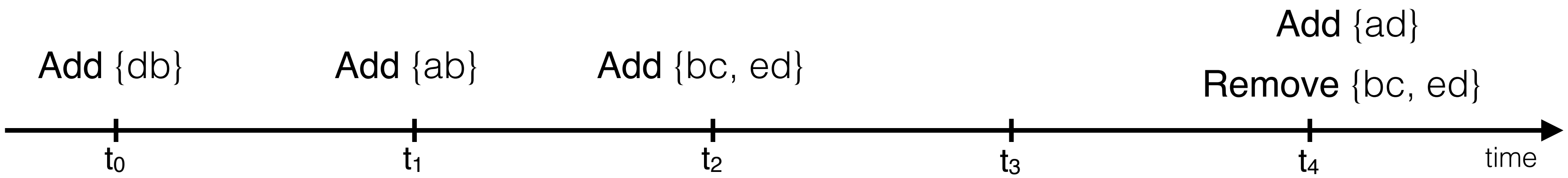


Typical representations

Snapshots:



Temporal Log:



Add {db}

Add {ab}

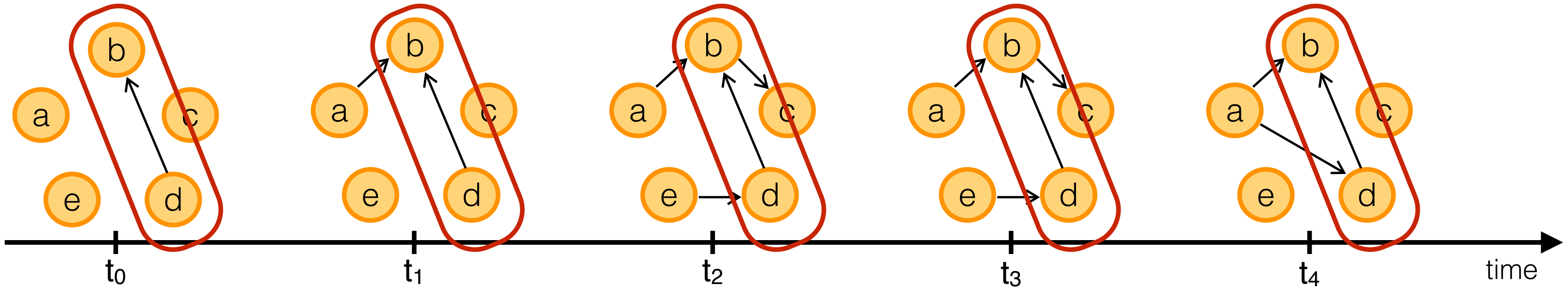
Add {bc, ed}

Add {ad}

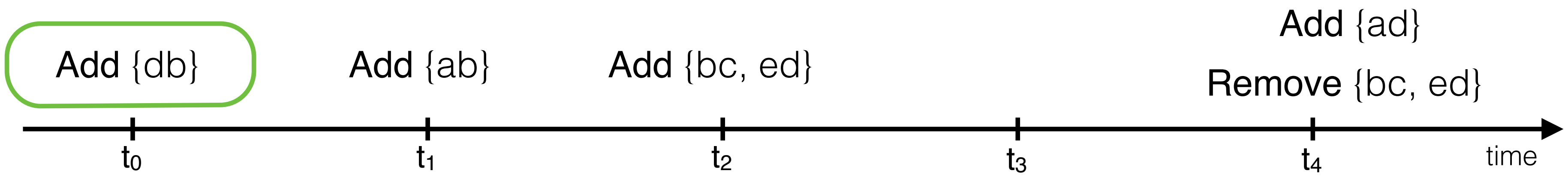
Remove {bc, ed}

Typical representations

Snapshots:



Temporal Log:



Add {db}

Add {ab}

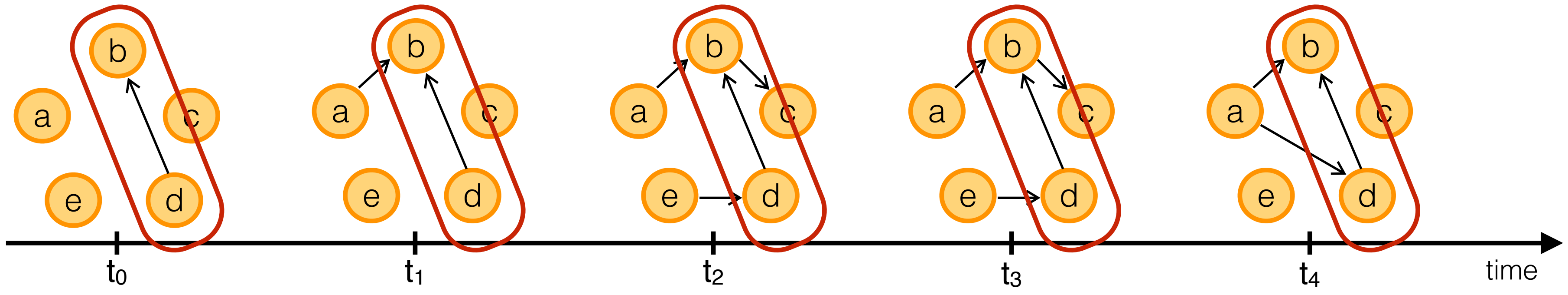
Add {bc, ed}

Add {ad}

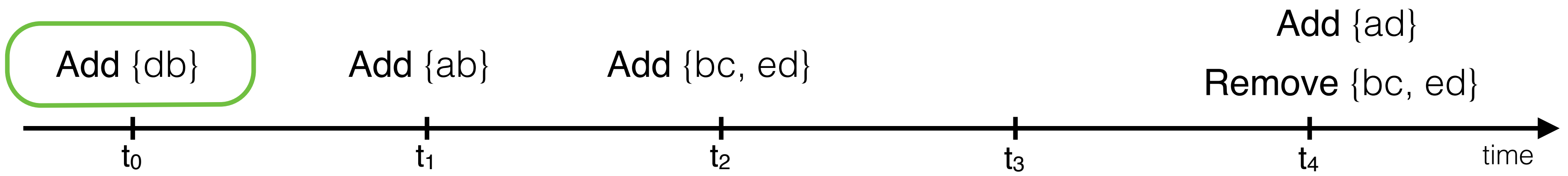
Remove {bc, ed}

Typical representations

Snapshots:



Temporal Log:



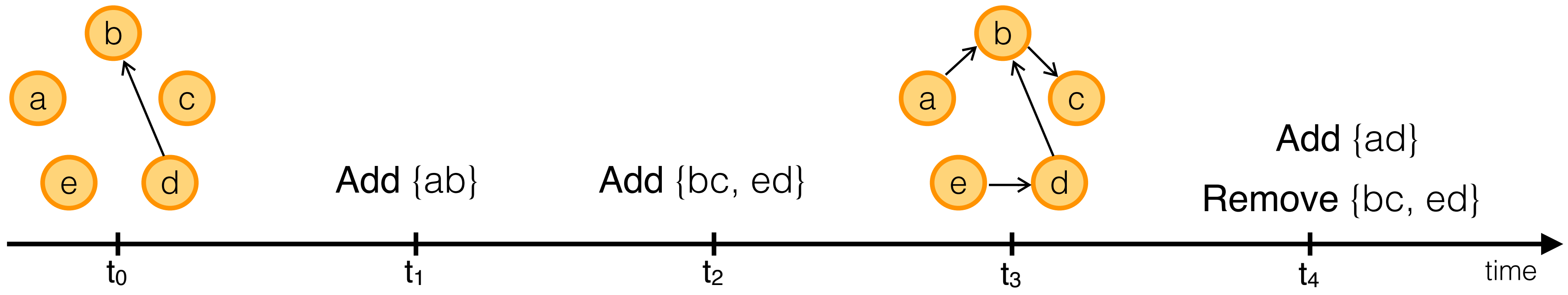
Add {db}

Add {ab}

Add {bc, ed}

Add {ad}
Remove {bc, ed}

Snapshot
+
Temporal Log



Add {ab}

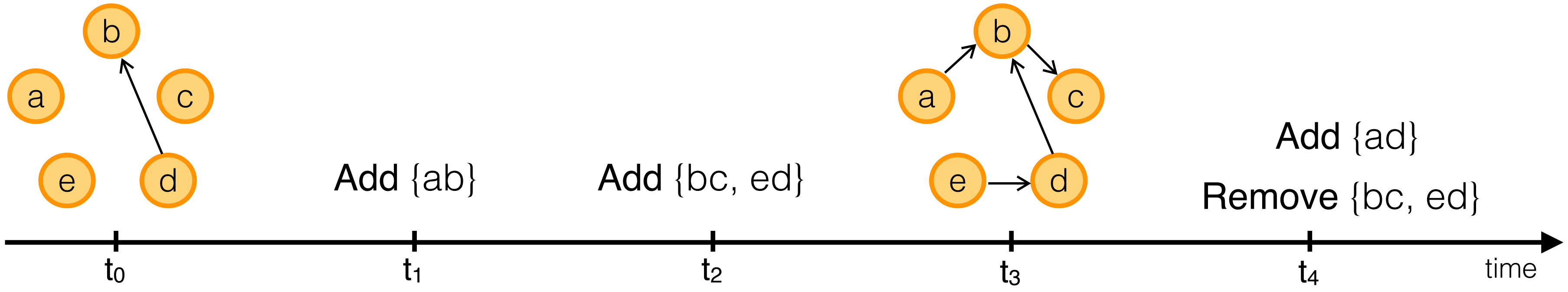
Add {bc, ed}

Add {ad}
Remove {bc, ed}

Previous works

Previous works

Khurana and Deshpande (2013).



Add {ab}

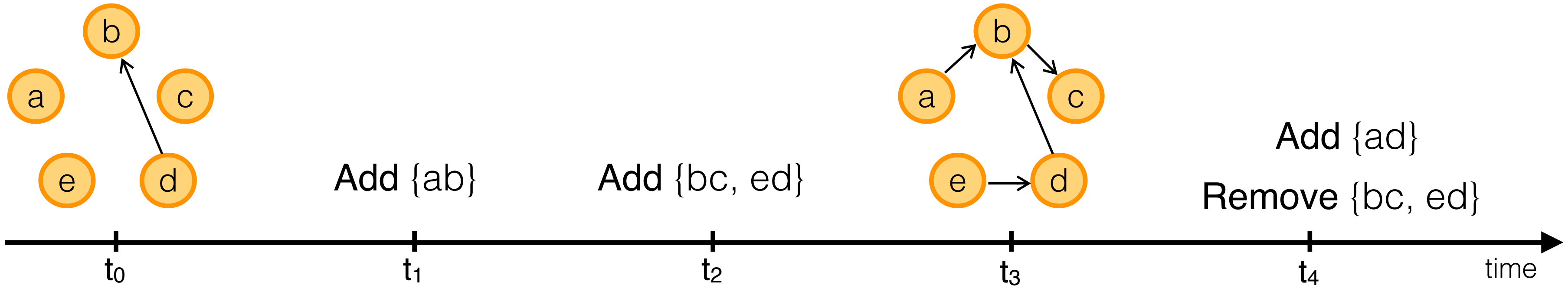
Add {bc, ed}

Add {ad}

Remove {bc, ed}

Previous works

Khurana and Deshpande (2013).



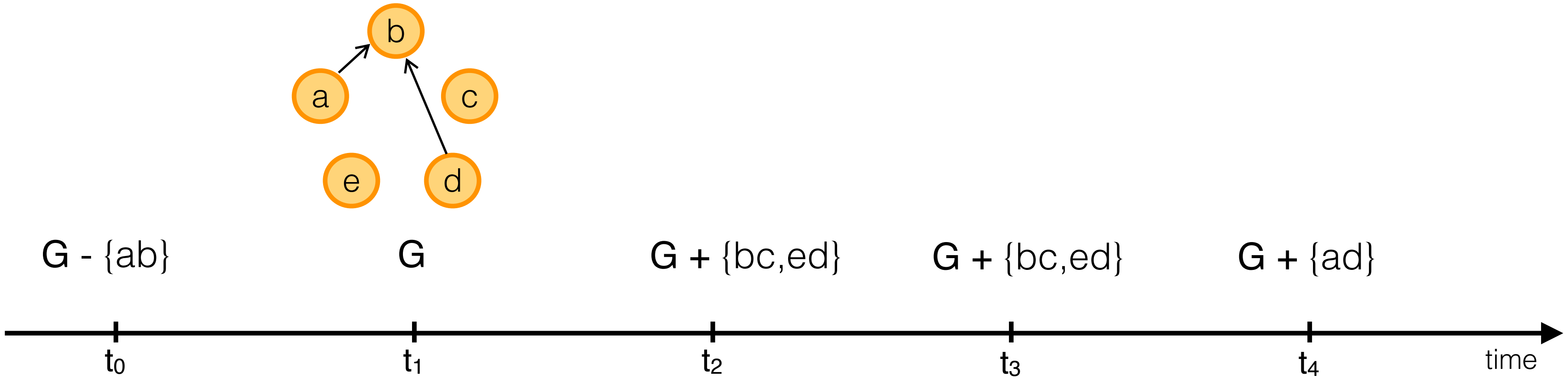
Add {ab}

Add {bc, ed}

Add {ad}

Remove {bc, ed}

Ren et al. (2011)



G - {ab}

G

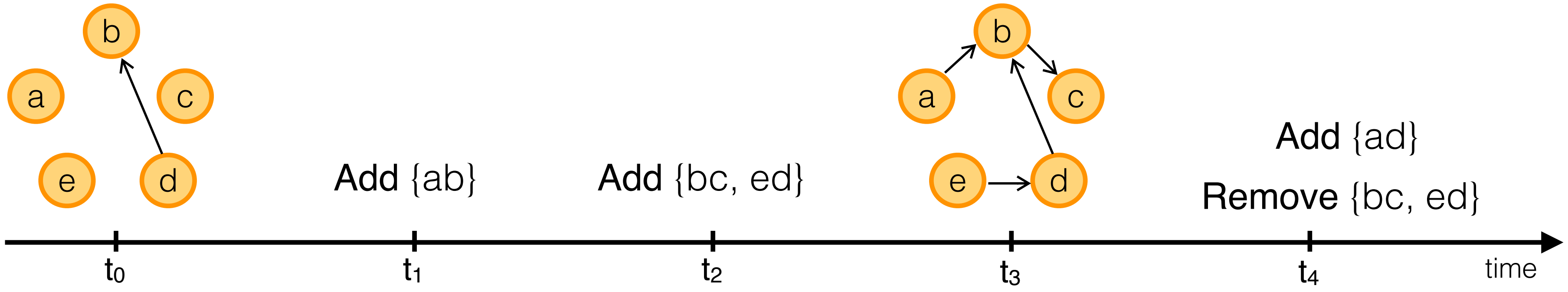
G + {bc,ed}

G + {bc,ed}

G + {ad}

Previous works

Khurana and Deshpande (2013).



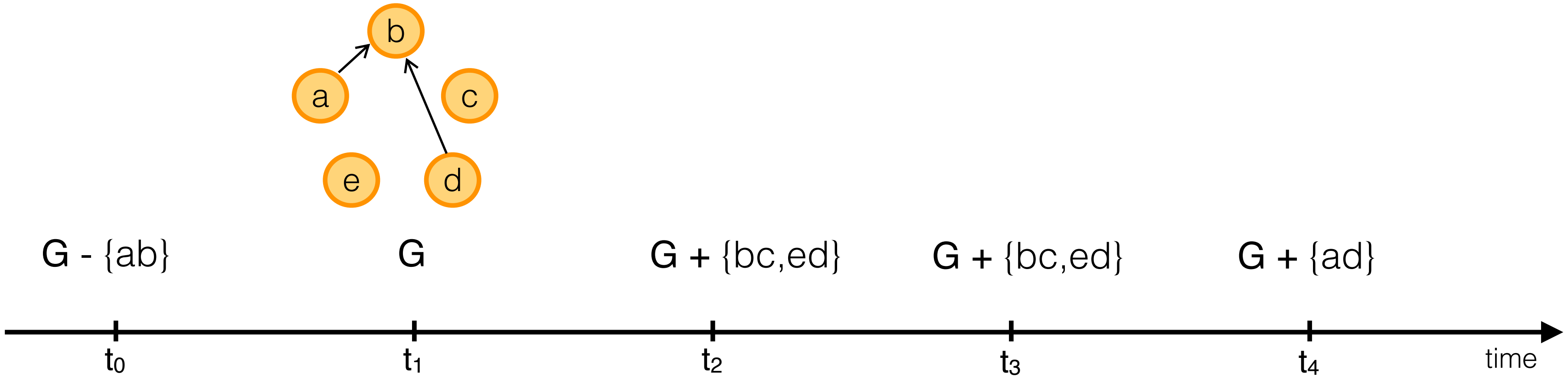
Add {ab}

Add {bc, ed}

Add {ad}

Remove {bc, ed}

Ren et al. (2011)



G - {ab}

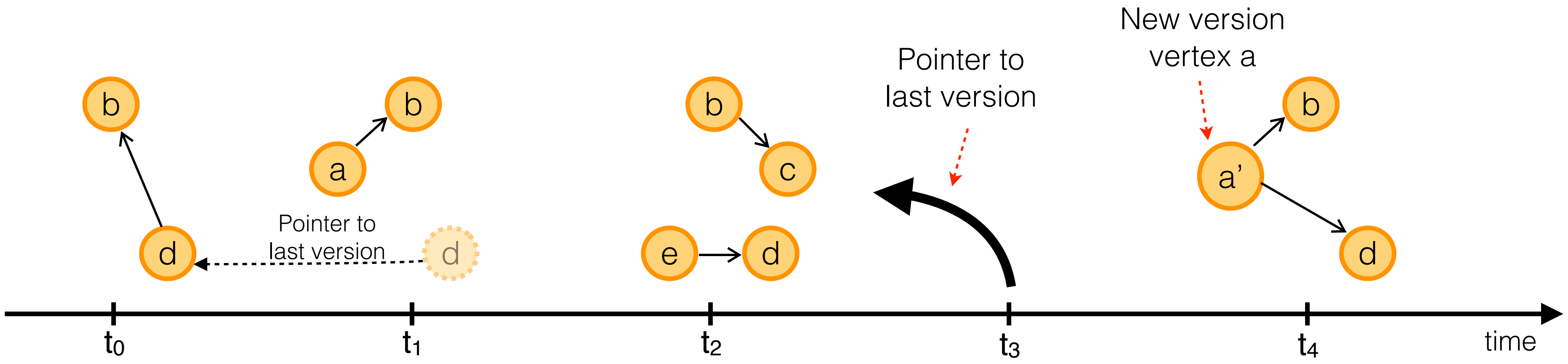
G

G + {bc,ed}

G + {bc,ed}

G + {ad}

Labouseur et al. (2014).

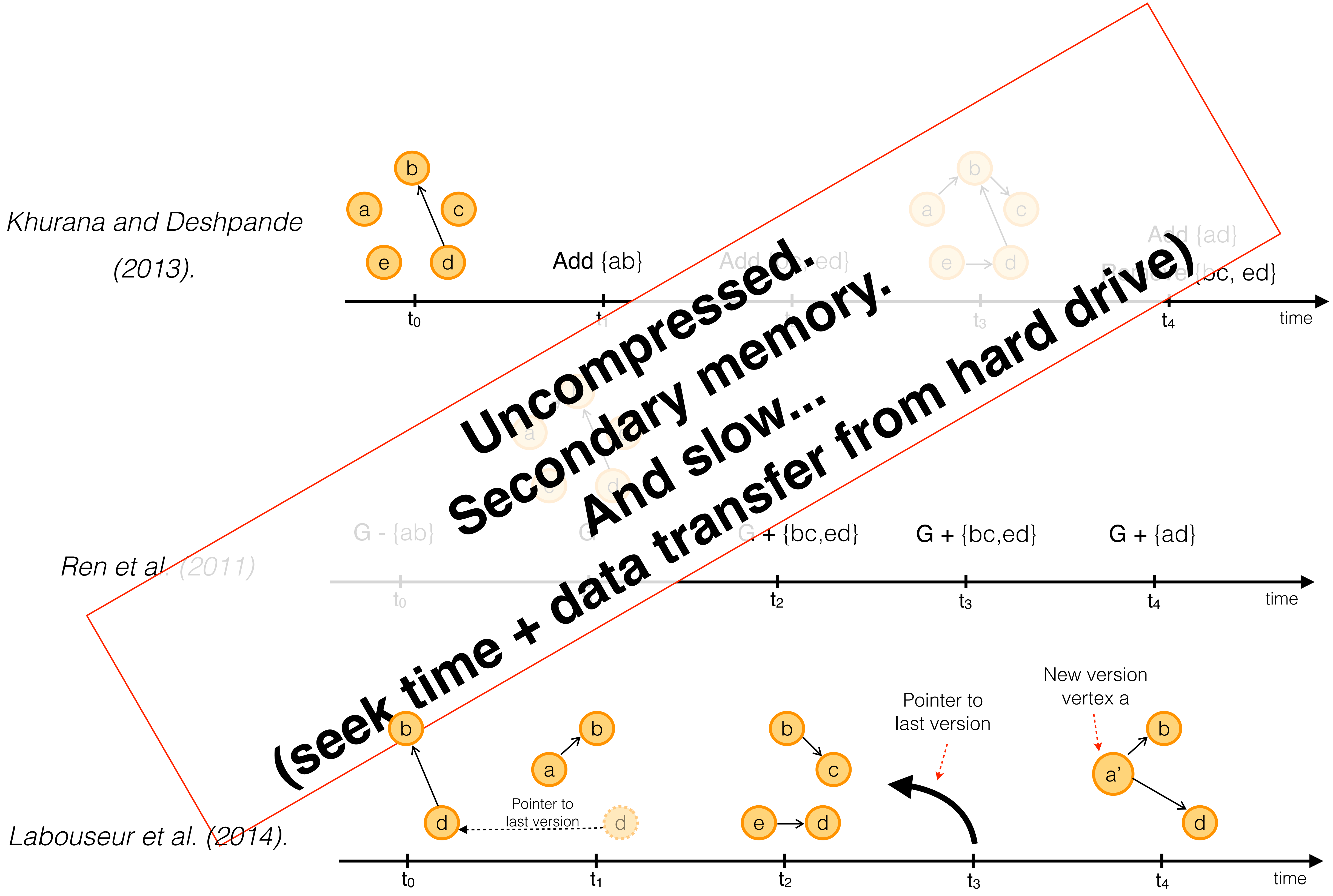


Pointer to last version

New version vertex a

Pointer to last version

Previous works



Khurana and Deshpande (2013).

Ren et al. (2011)

Labouseur et al. (2014).

Uncompressed.
Secondary memory.
And slow…
(seek time + data transfer from hard drive)

Outline

- ✓ Definition and Motivation.
- ✓ Previous works about temporal graphs.
- Compression of temporal graphs.
- Contributions.
- Evaluation.
- Conclusions and future works.

Goal & Hypothesis

- Goal:
    - To design new compact data structures for temporal graphs.
- Hypothesis:
    - Compact data structures for temporal graphs based on logs of changes and multidimensional representation use less space and similar access time than structures based on snapshot representations.

Compact data structures

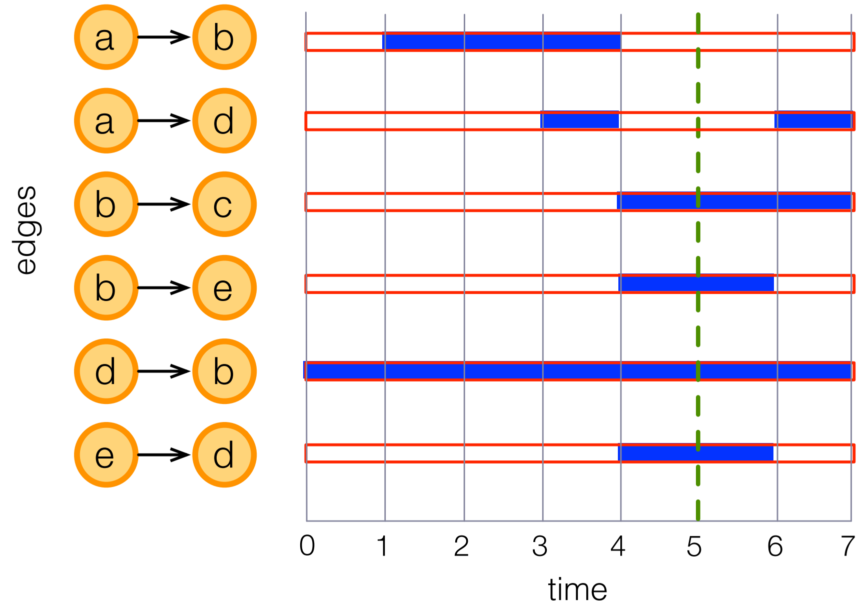
- State of the art in compressed graphs:
  - The Web Graph: k^2-tree, Webgraph, Repair Graph, etc…
  - Binary relations.
  - None of them consider time.
- But, there are many other tools and compact data structures available:
  - Sequence compression: Wavelet Tree.
  - Text compression: Compressed suffix array for pattern matching in bioinformatics.
  - Inverted indexes: Web search.
  - Compressed bitmaps, and many others!

Outline

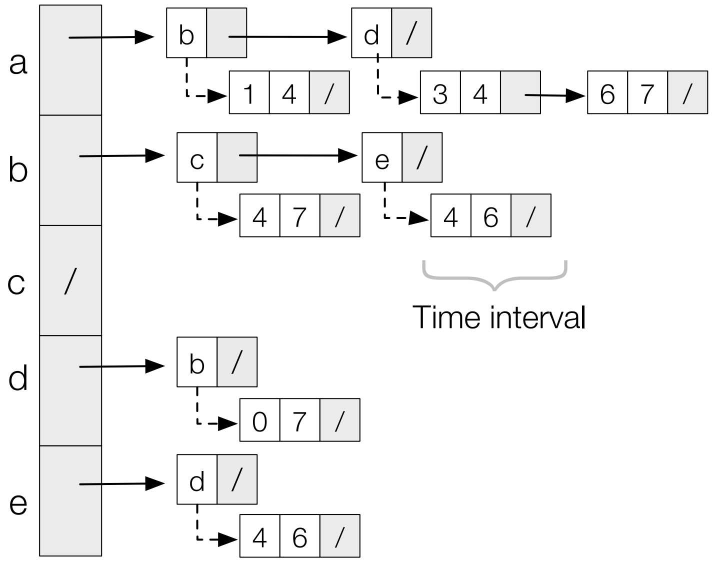
- ✓ Definition and Motivation.
- ✓ Previous works about temporal graphs.
- ✓ Compression of temporal graphs.
- Contributions
    - Based on inverted indexes:
        - EdgeLog
        - EveLog
    - Based on Wavelet Trees:
        - Compact Adjacency Sequence (CAS)
        - Compact Events ordered by Time (CET)
    - Based on the Compressed Suffix Array:
        - Temporal Graph CSA
    - Based on the multidimensional k^d-tree
        - The Compressed k^d-tree
- Evaluation.
- Conclusions and future works.

EdgeLog

- EdgeLog: Temporal Log as a list of time intervals per edge.
    - Edges are stored as adjacency lists.
    - Adjacency list and time intervals are compressed as inverted lists.
    - Operations need to decompress the adjacency lists and time intervals.
- Reverse neighbors require extra space.

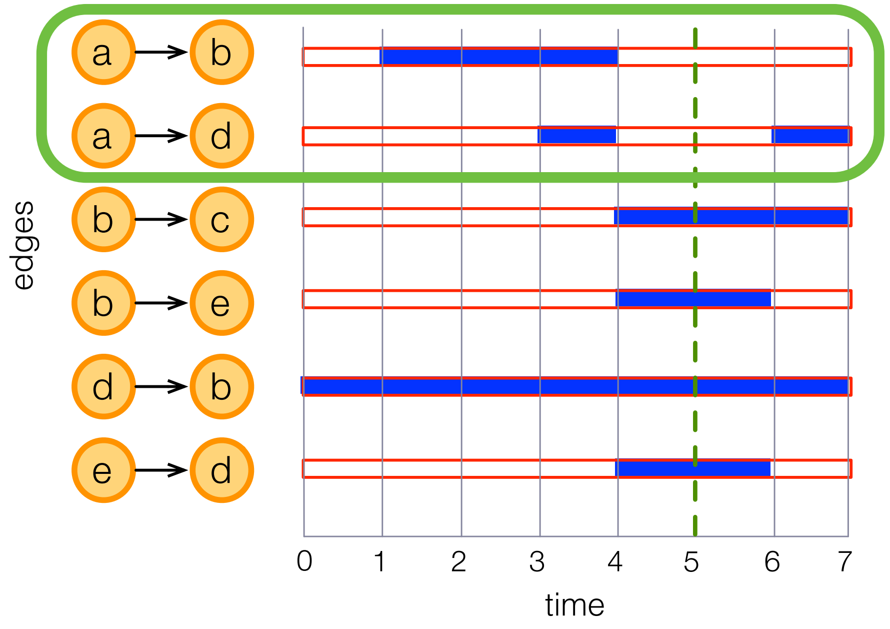


EdgeLog

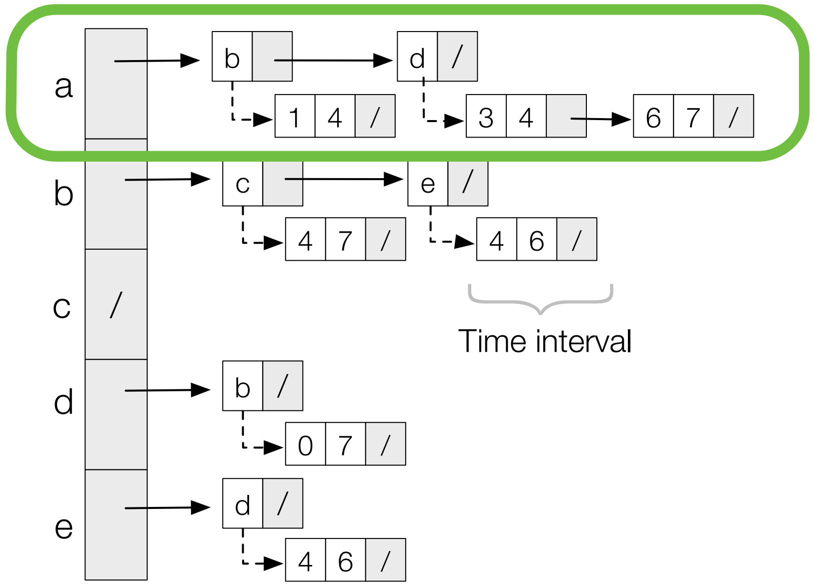


EdgeLog

- EdgeLog: Temporal Log as a list of time intervals per edge.
  - Edges are stored as adjacency lists.
  - Adjacency list and time intervals are compressed as inverted lists.
  - Operations need to decompress the adjacency lists and time intervals.
- Reverse neighbors require extra space.

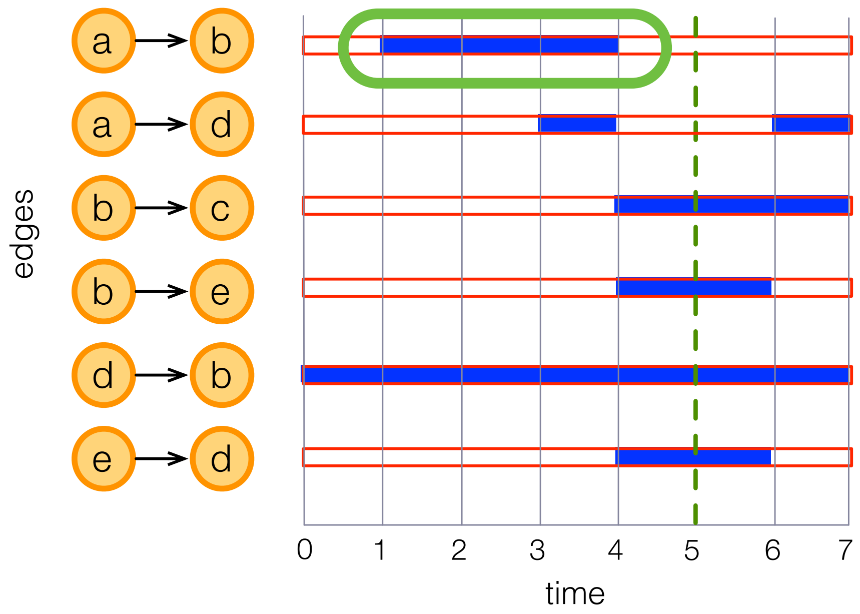


EdgeLog

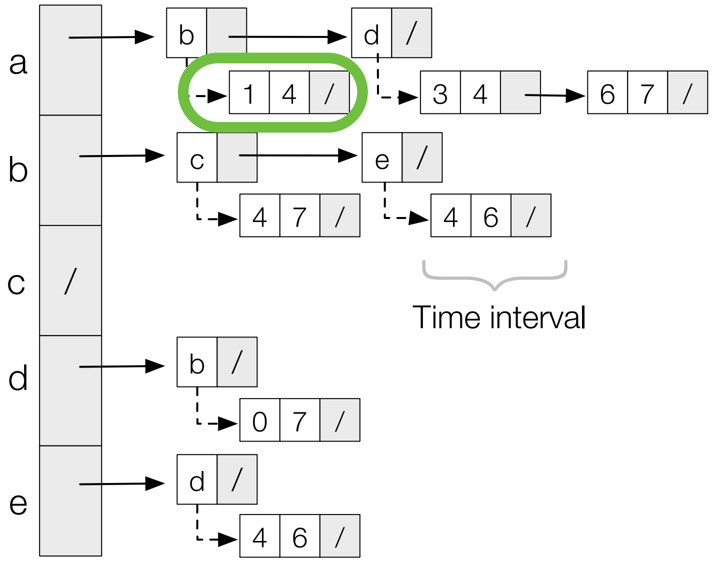


EdgeLog

- EdgeLog: Temporal Log as a list of time intervals per edge.
    - Edges are stored as adjacency lists.
    - Adjacency list and time intervals are compressed as inverted lists.
    - Operations need to decompress the adjacency lists and time intervals.
- Reverse neighbors require extra space.

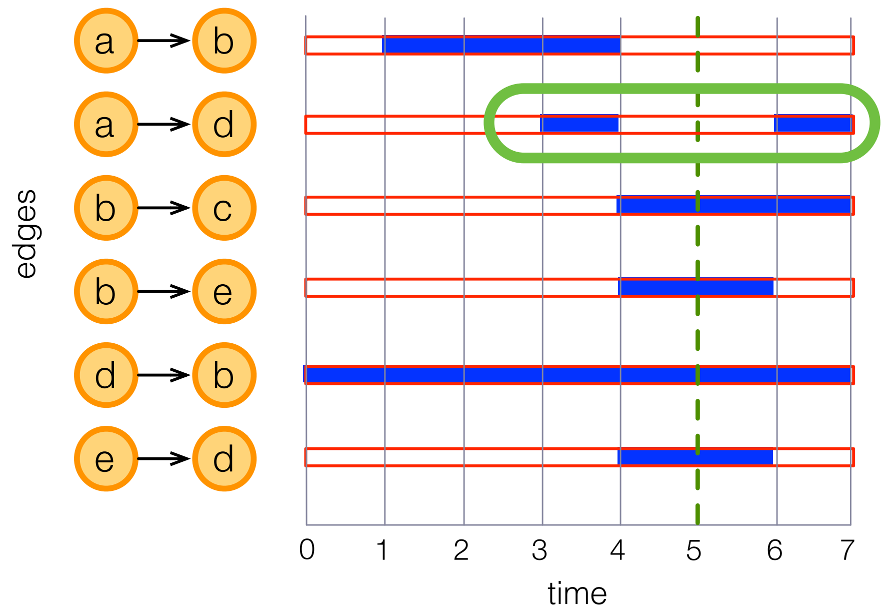


EdgeLog

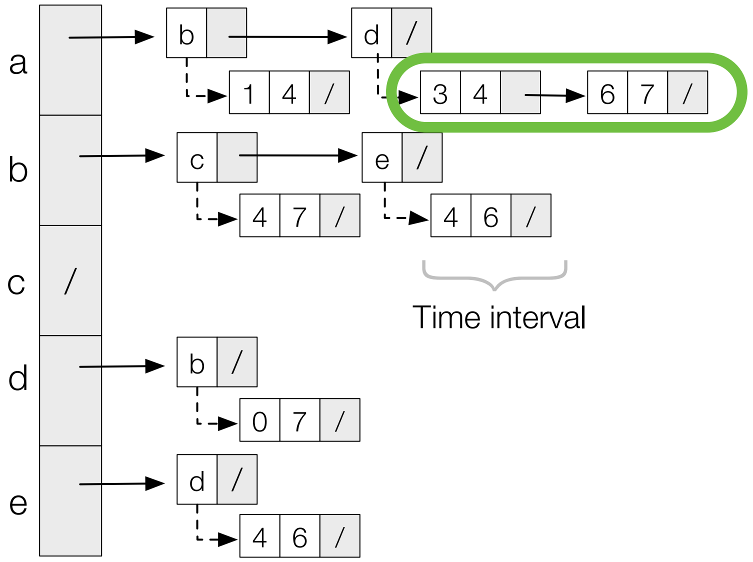


EdgeLog

- EdgeLog: Temporal Log as a list of time intervals per edge.
  - Edges are stored as adjacency lists.
  - Adjacency list and time intervals are compressed as inverted lists.
  - Operations need to decompress the adjacency lists and time intervals.
- Reverse neighbors require extra space.

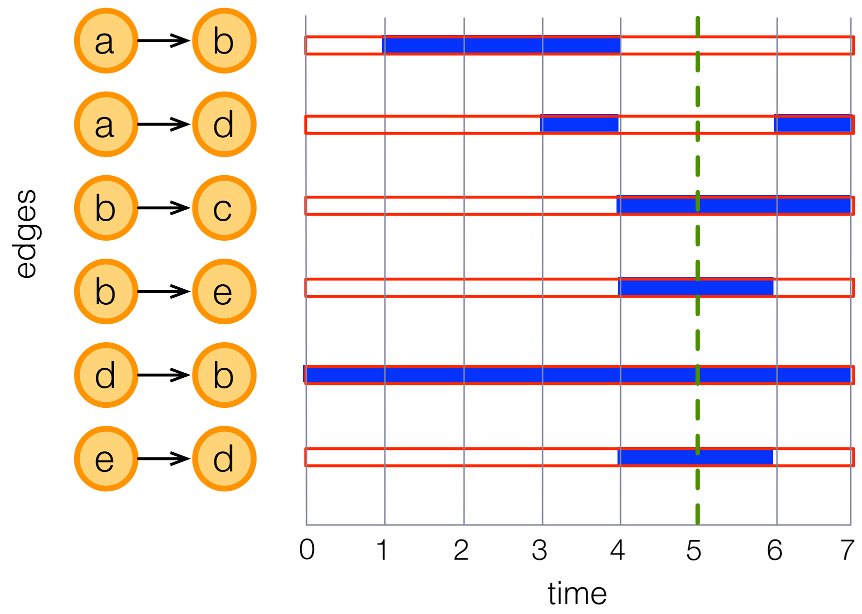


EdgeLog

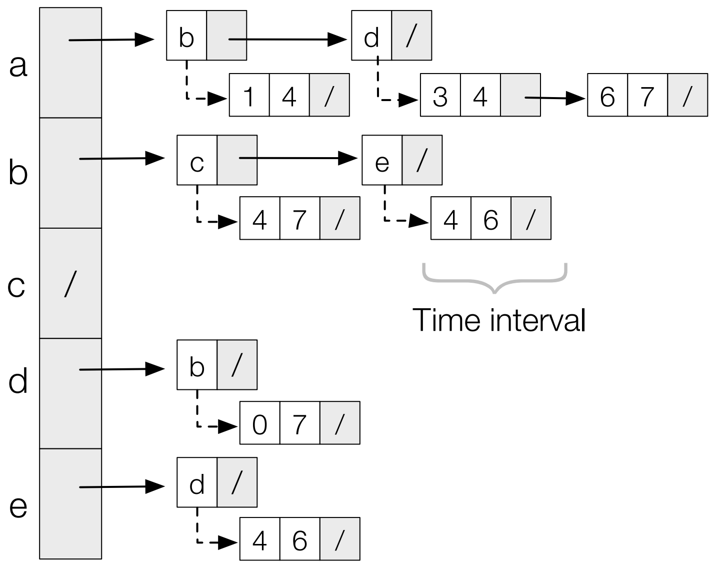


EdgeLog

- EdgeLog: Temporal Log as a list of time intervals per edge.
  - Edges are stored as adjacency lists.
  - Adjacency list and time intervals are compressed as inverted lists.
  - Operations need to decompress the adjacency lists and time intervals.
- Reverse neighbors require extra space.



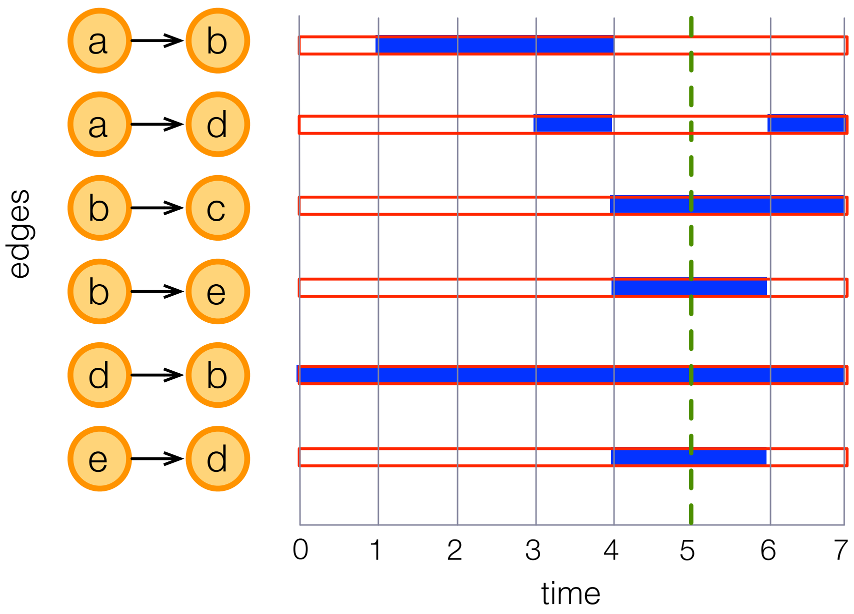
EdgeLog



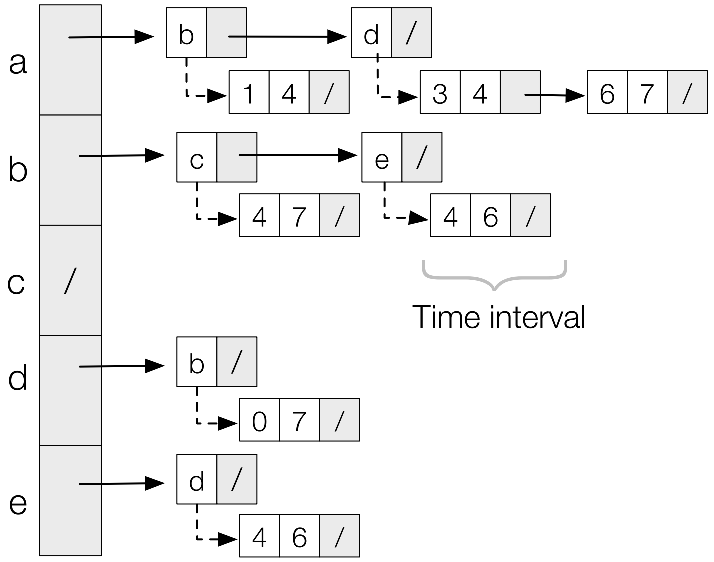
EdgeLog

- EdgeLog: Temporal Log as a list of time intervals per edge.
    - Edges are stored as adjacency lists.
    - Adjacency list and time intervals are compressed as inverted lists.
    - Operations need to decompress the adjacency lists and time intervals.
- Reverse neighbors require extra space.

edge(ab,t=1)?



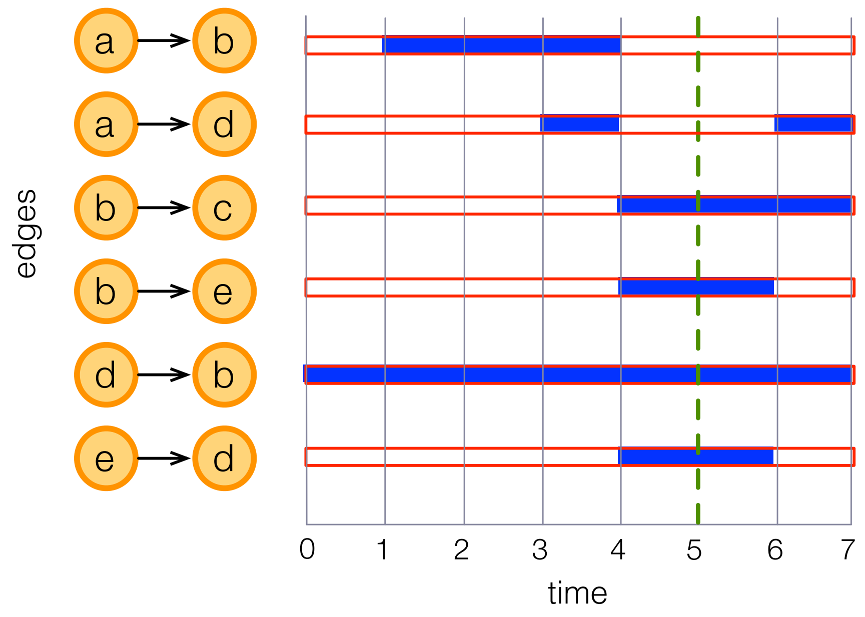
EdgeLog



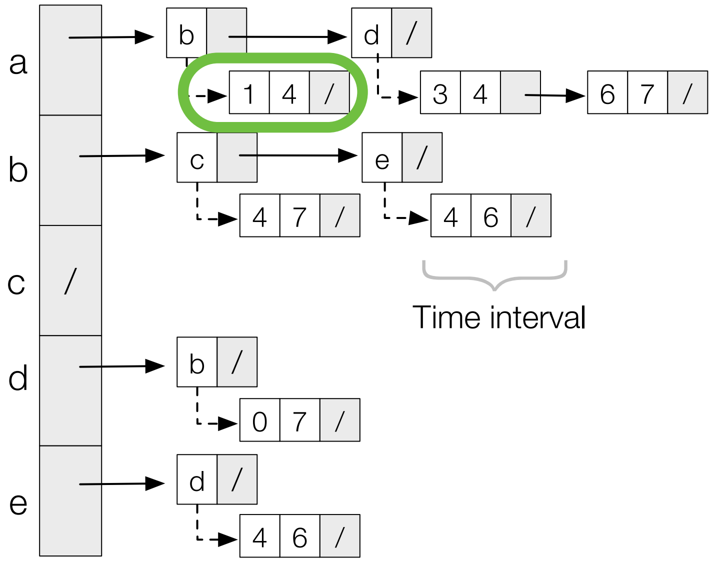
EdgeLog

- EdgeLog: Temporal Log as a list of time intervals per edge.
  - Edges are stored as adjacency lists.
  - Adjacency list and time intervals are compressed as inverted lists.
  - Operations need to decompress the adjacency lists and time intervals.
- Reverse neighbors require extra space.

edge(ab,t=1)?

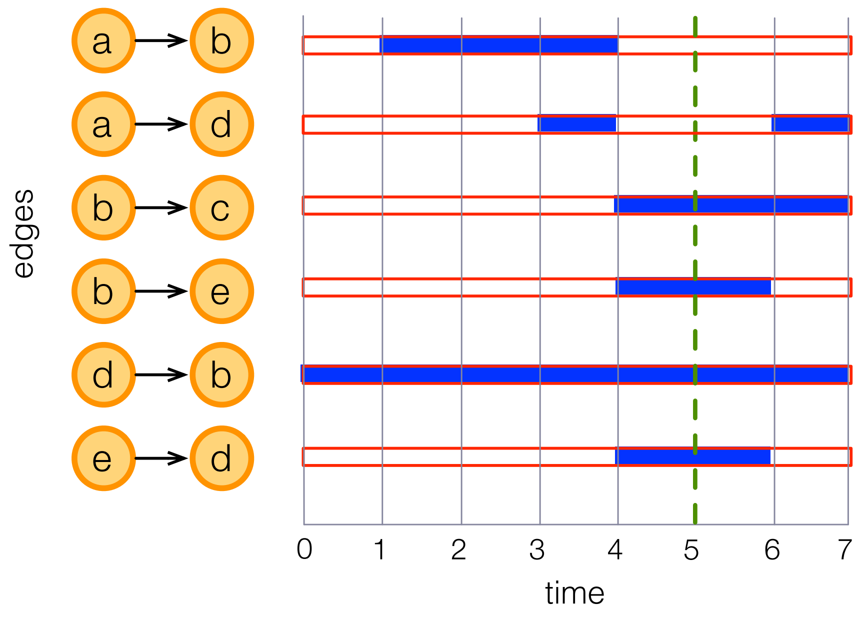


EdgeLog

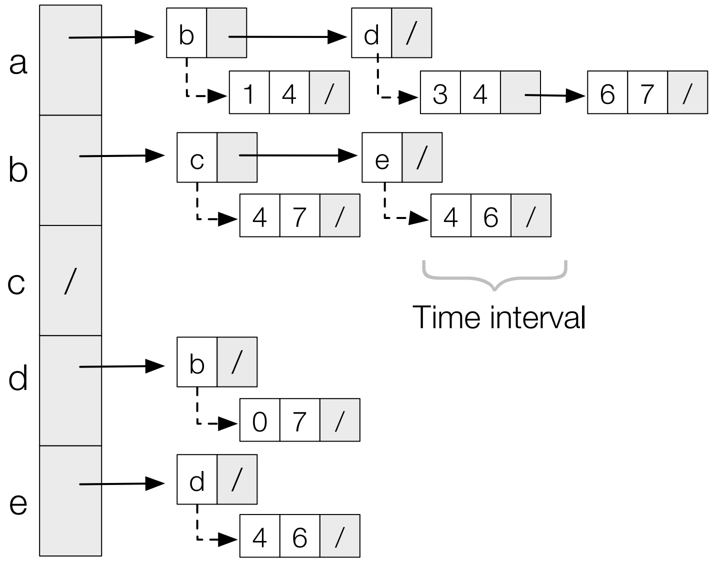


EdgeLog

- EdgeLog: Temporal Log as a list of time intervals per edge.
  - Edges are stored as adjacency lists.
  - Adjacency list and time intervals are compressed as inverted lists.
  - Operations need to decompress the adjacency lists and time intervals.
- Reverse neighbors require extra space.



EdgeLog

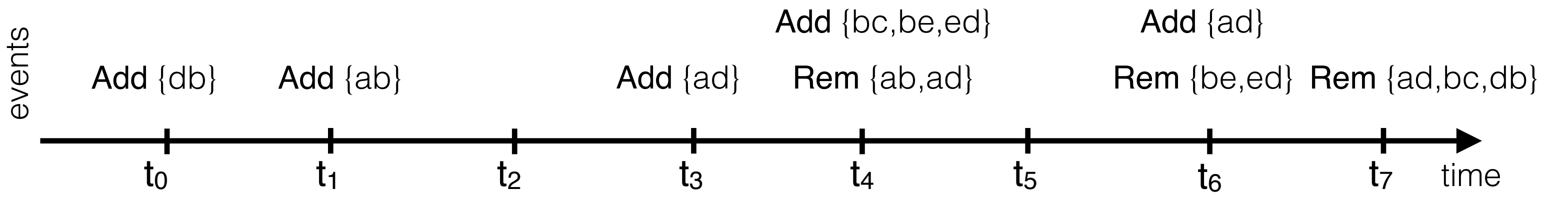


Outline

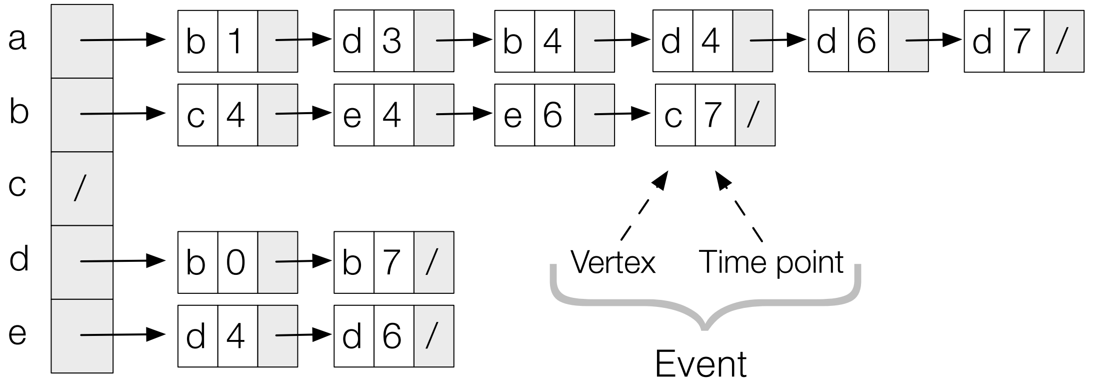
- ✓ Definition and Motivation.
- ✓ Previous works about temporal graphs.
- ✓ Compression of temporal graphs.
- Contributions
  - Based on inverted indexes:
    - ✓ EdgeLog
    - EveLog
  - Based on Wavelet Trees:
    - Compact Adjacency Sequence (CAS)
    - Compact Events ordered by Time (CET)
  - Based on the Compressed Suffix Array:
    - Temporal Graph CSA
  - Based on the multidimensional k^d-tree
    - The Compressed k^d-tree
- Evaluation.
- Conclusions and future works.

EveLog

- EveLog: Temporal Log as a list of events per vertex.
  - Events are sorted by the time they occur.
  - Operations count how many changes there are per edge.
    - Parity property: even times, the edge is active. Otherwise is inactive.
  - List of events are compressed using inverted lists.
- Reverse neighbors also require extra space.

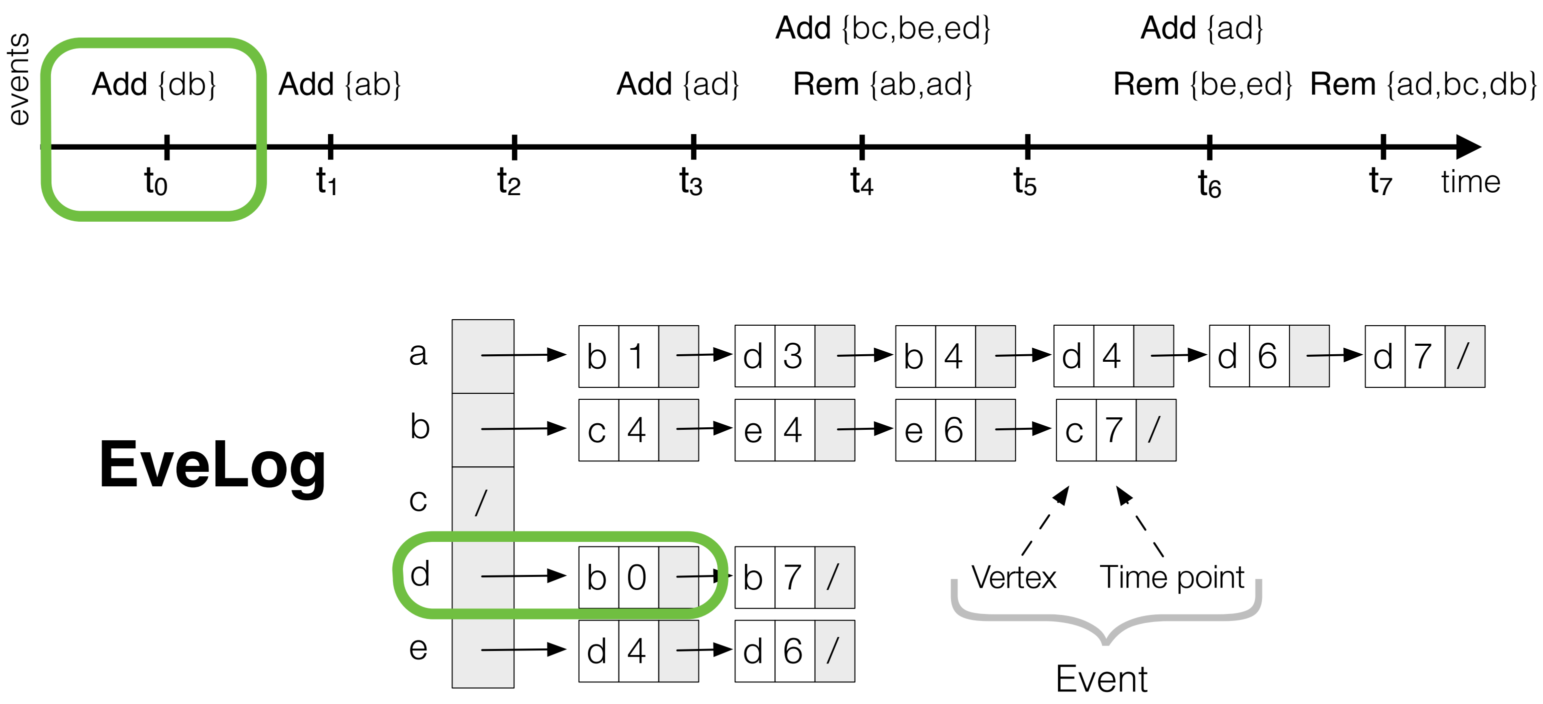


EveLog



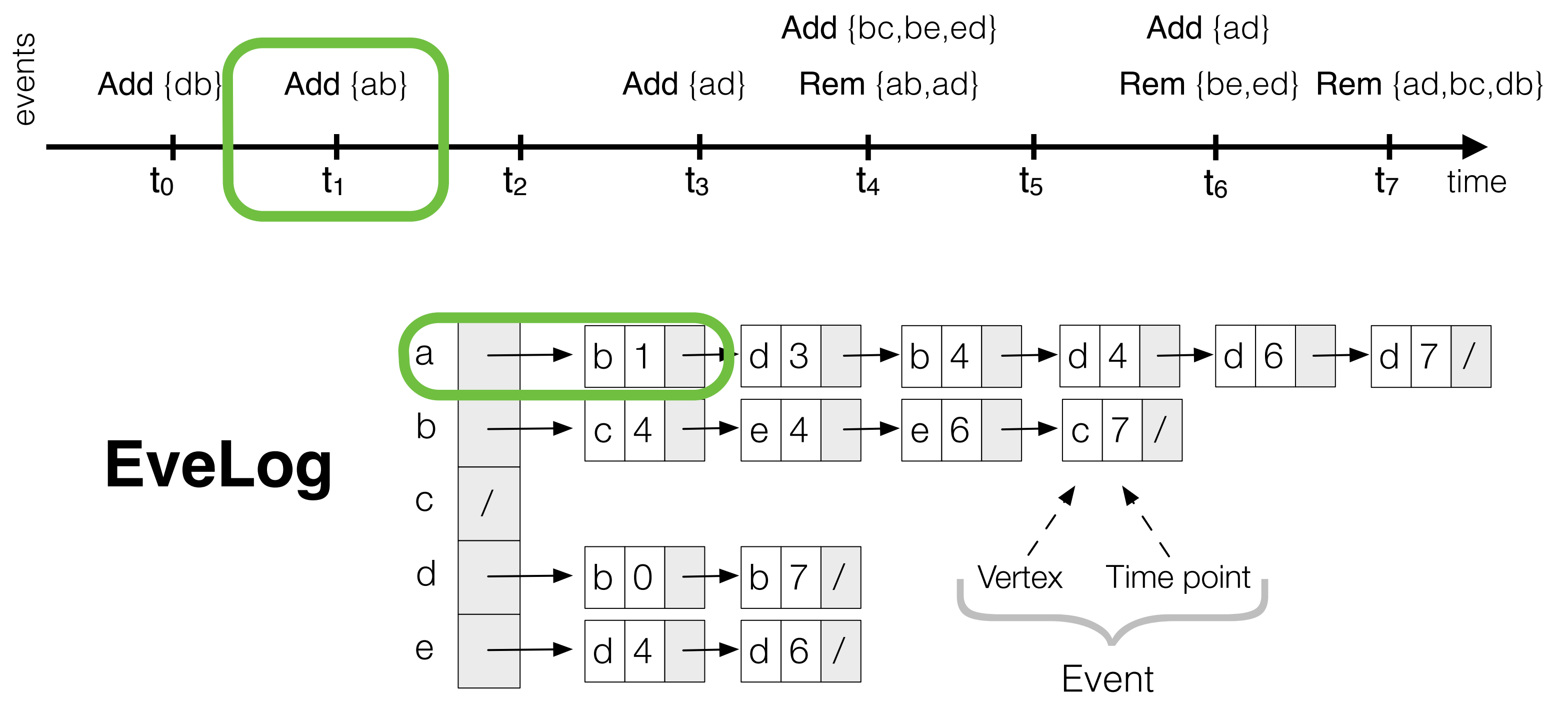
EveLog

- EveLog: Temporal Log as a list of events per vertex.
  - Events are sorted by the time they occur.
  - Operations count how many changes there are per edge.
    - Parity property: even times, the edge is active. Otherwise is inactive.
  - List of events are compressed using inverted lists.
- Reverse neighbors also require extra space.



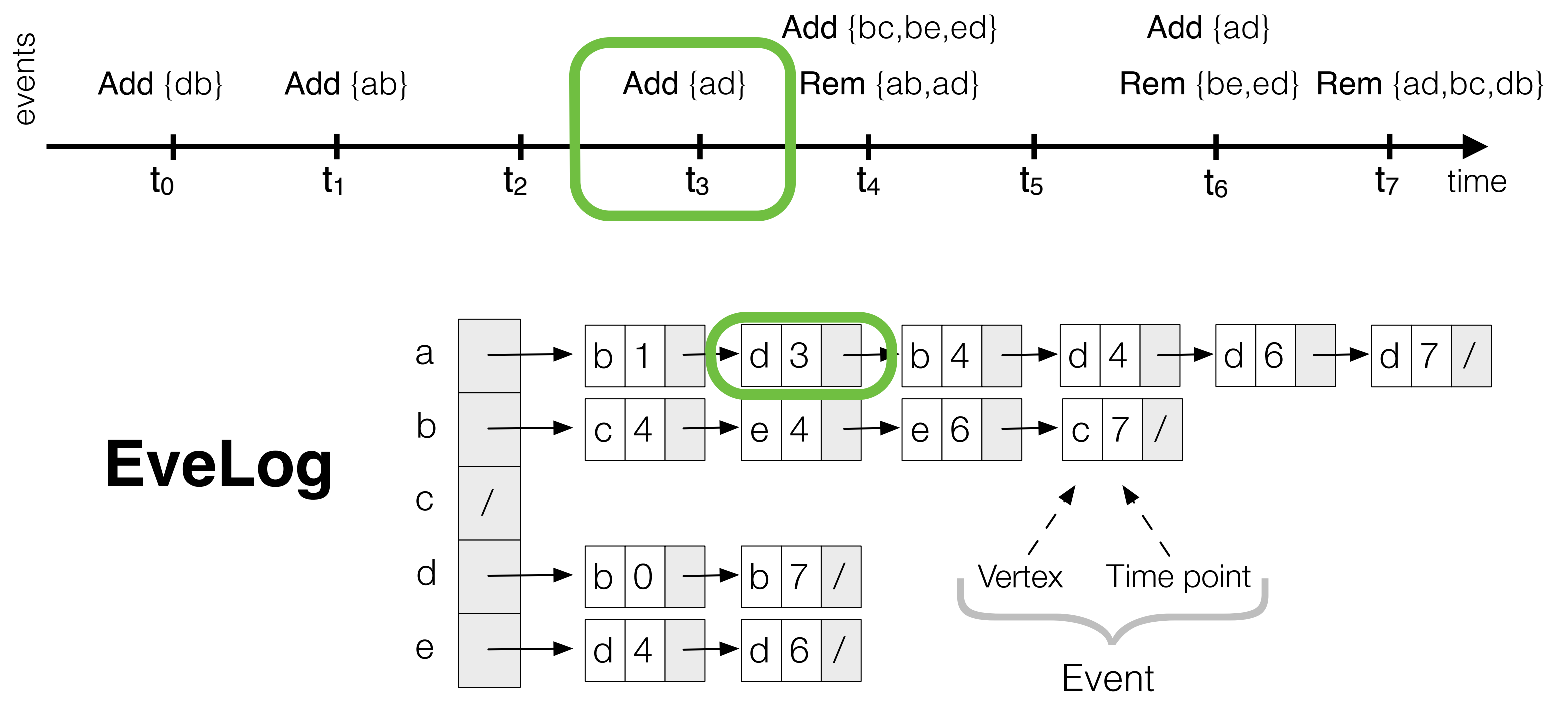
EveLog

- EveLog: Temporal Log as a list of events per vertex.
  - Events are sorted by the time they occur.
  - Operations count how many changes there are per edge.
    - Parity property: even times, the edge is active. Otherwise is inactive.
  - List of events are compressed using inverted lists.
- Reverse neighbors also require extra space.



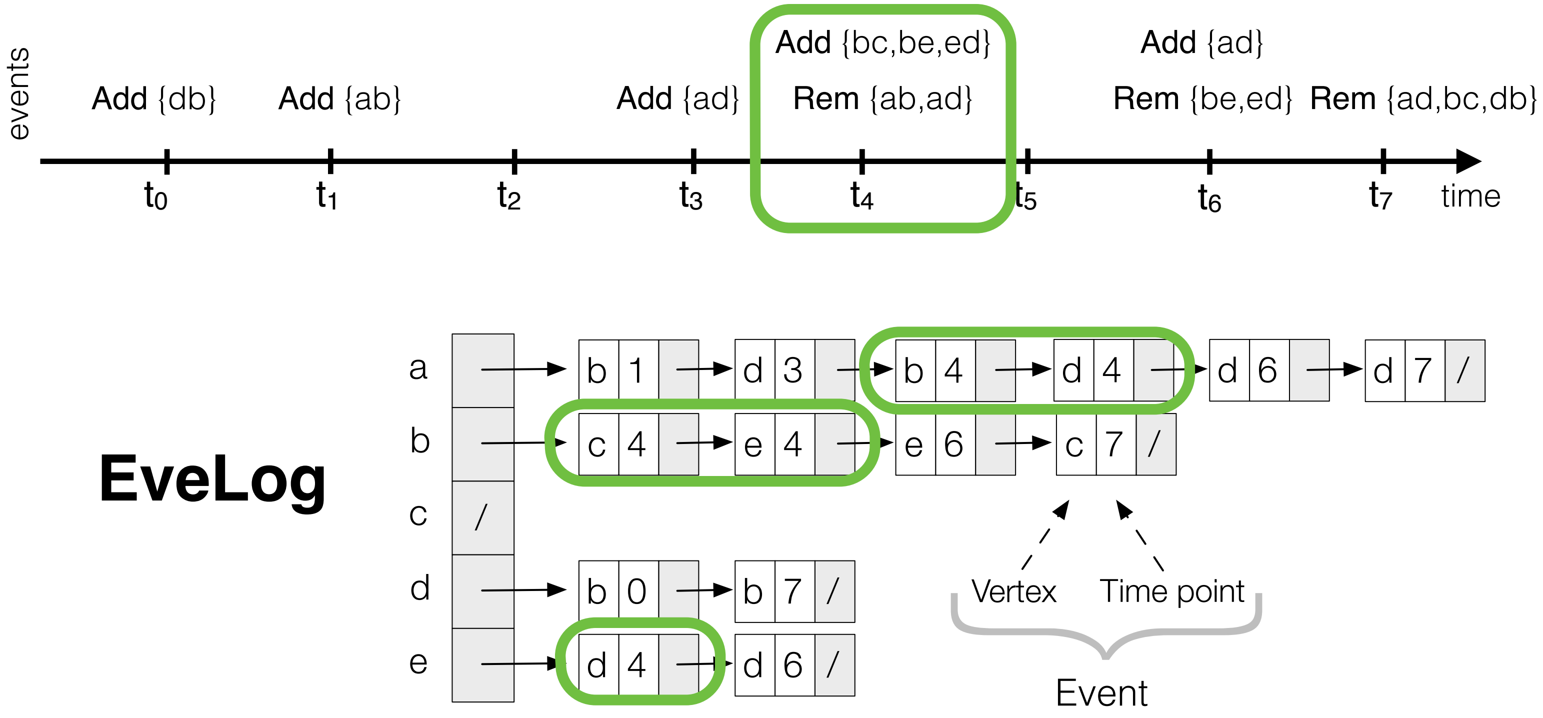
EveLog

- EveLog: Temporal Log as a list of events per vertex.
    - Events are sorted by the time they occur.
    - Operations count how many changes there are per edge.
        - Parity property: even times, the edge is active. Otherwise is inactive.
    - List of events are compressed using inverted lists.
- Reverse neighbors also require extra space.



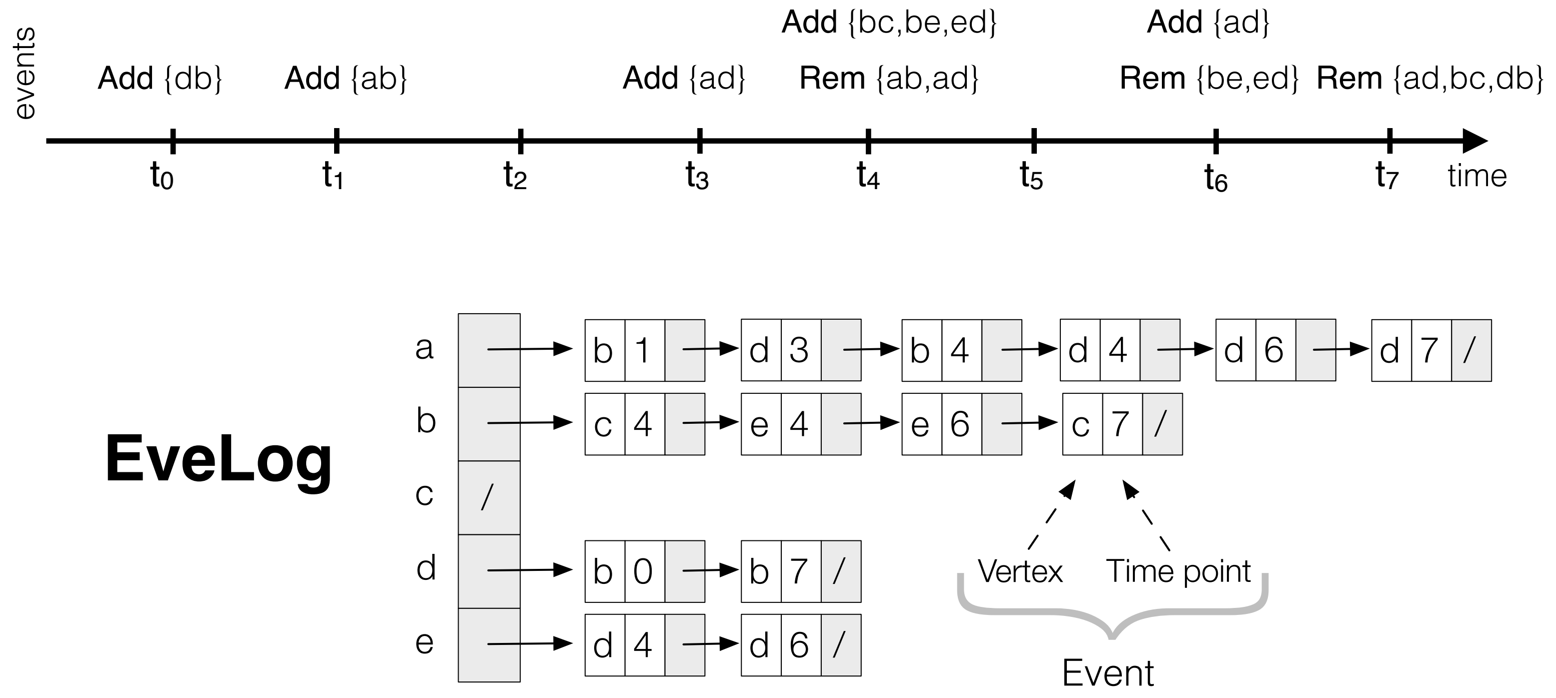
EveLog

- EveLog: Temporal Log as a list of events per vertex.
  - Events are sorted by the time they occur.
  - Operations count how many changes there are per edge.
    - Parity property: even times, the edge is active. Otherwise is inactive.
  - List of events are compressed using inverted lists.
- Reverse neighbors also require extra space.



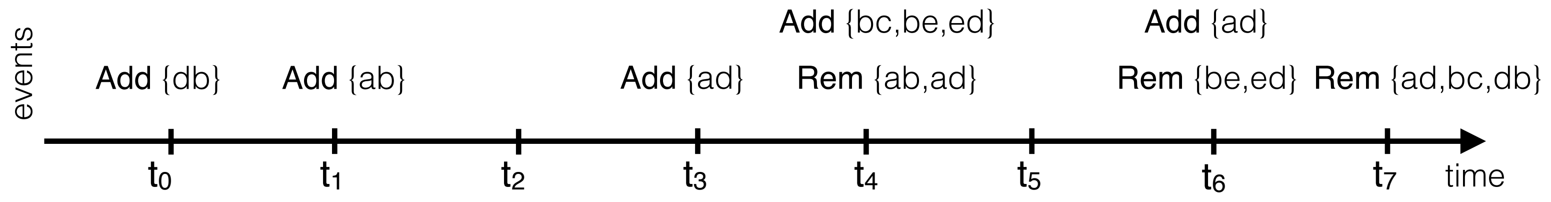
EveLog

- EveLog: Temporal Log as a list of events per vertex.
  - Events are sorted by the time they occur.
  - Operations count how many changes there are per edge.
    - Parity property: even times, the edge is active. Otherwise is inactive.
  - List of events are compressed using inverted lists.
- Reverse neighbors also require extra space.



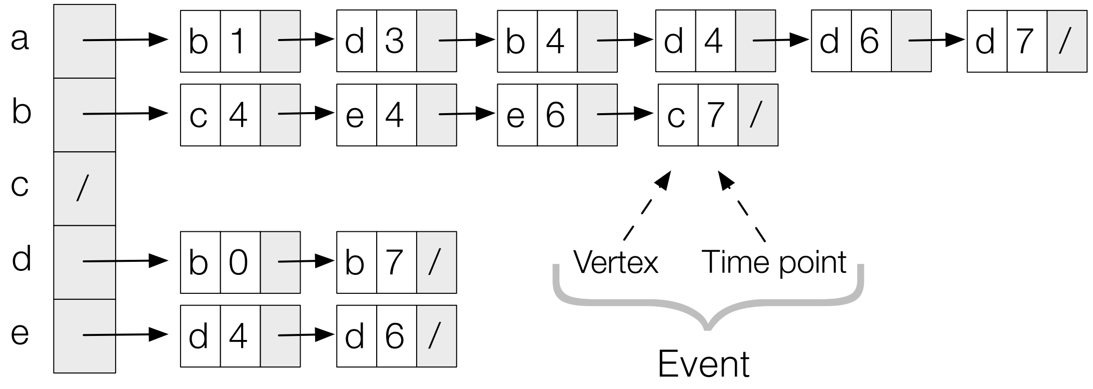
EveLog

- EveLog: Temporal Log as a list of events per vertex.
  - Events are sorted by the time they occur.
  - Operations count how many changes there are per edge.
    - Parity property: even times, the edge is active. Otherwise is inactive.
  - List of events are compressed using inverted lists.
- Reverse neighbors also require extra space.



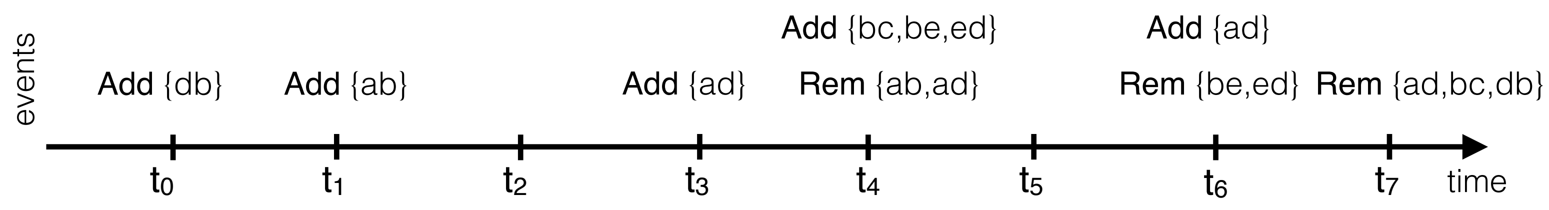
edge(ab,t=1)?

EveLog



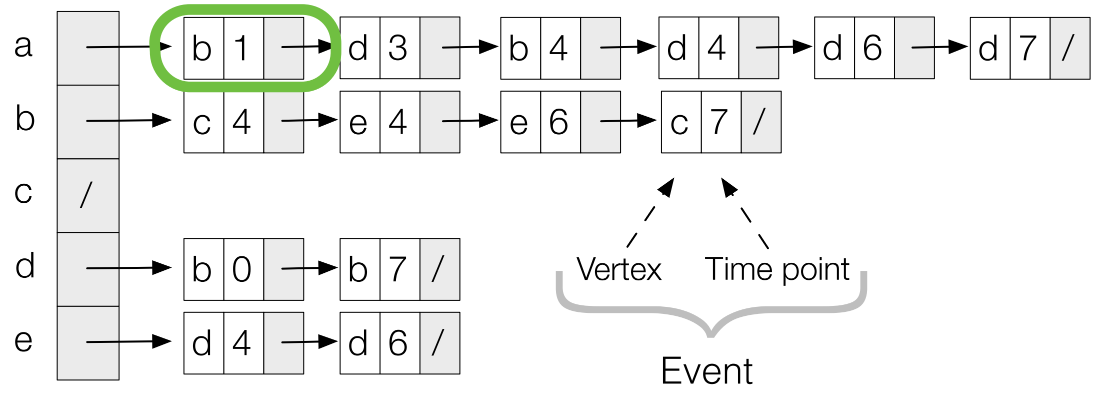
EveLog

- EveLog: Temporal Log as a list of events per vertex.
  - Events are sorted by the time they occur.
  - Operations count how many changes there are per edge.
    - Parity property: even times, the edge is active. Otherwise is inactive.
  - List of events are compressed using inverted lists.
- Reverse neighbors also require extra space.



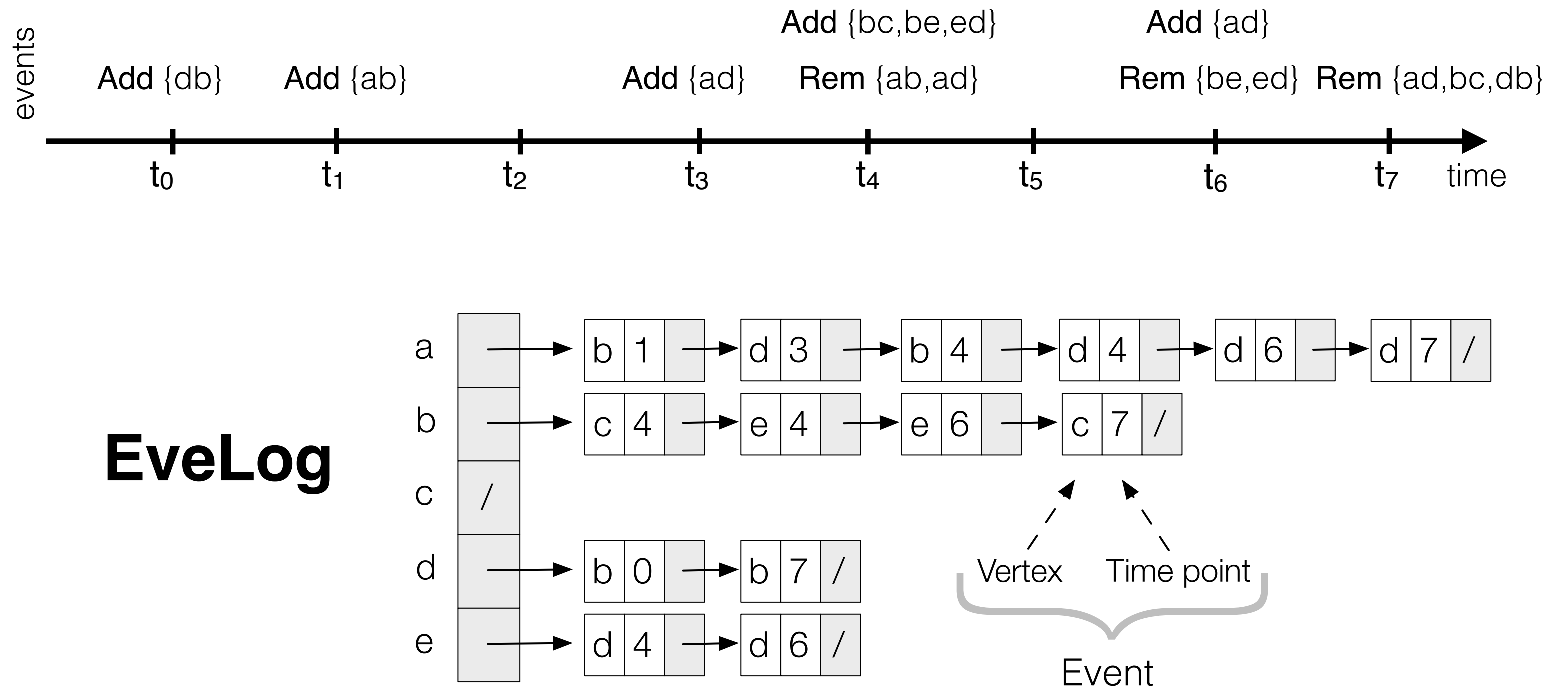
edge(ab,t=1)?

EveLog



EveLog

- EveLog: Temporal Log as a list of events per vertex.
  - Events are sorted by the time they occur.
  - Operations count how many changes there are per edge.
    - Parity property: even times, the edge is active. Otherwise is inactive.
  - List of events are compressed using inverted lists.
- Reverse neighbors also require extra space.

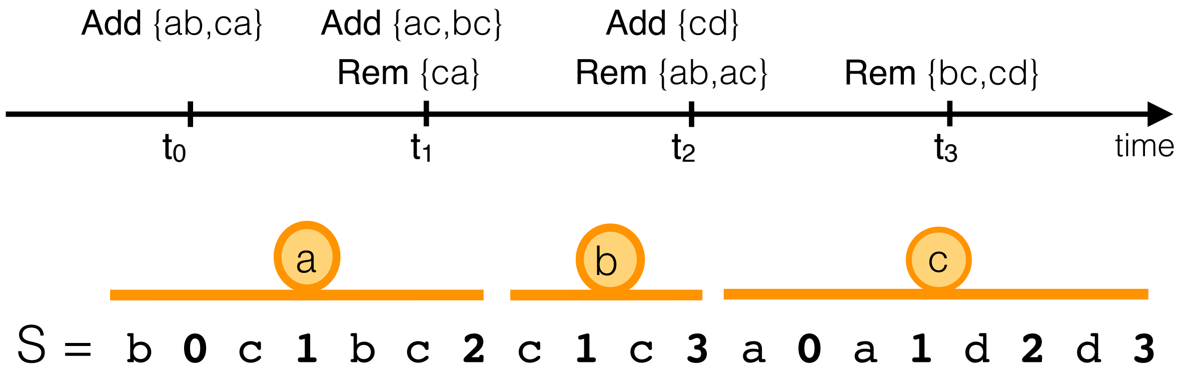


Outline

- ✓ Definition and Motivation.
- ✓ Previous works about temporal graphs.
- ✓ Compression of temporal graphs.
- Contributions
    - ✓ Based on inverted indexes:
        - ✓ EdgeLog
        - ✓ EveLog
    - Based on Wavelet Trees:
        - Compact Adjacency Sequence (CAS)
        - Compact Events ordered by Time (CET)
    - Based on the Compressed Suffix Array:
        - Temporal Graph CSA
    - Based on the multidimensional k^d-tree
        - The Compressed k^d-tree
- Evaluation.
- Conclusions and future works.

Compact Adjacency Sequence (CAS)

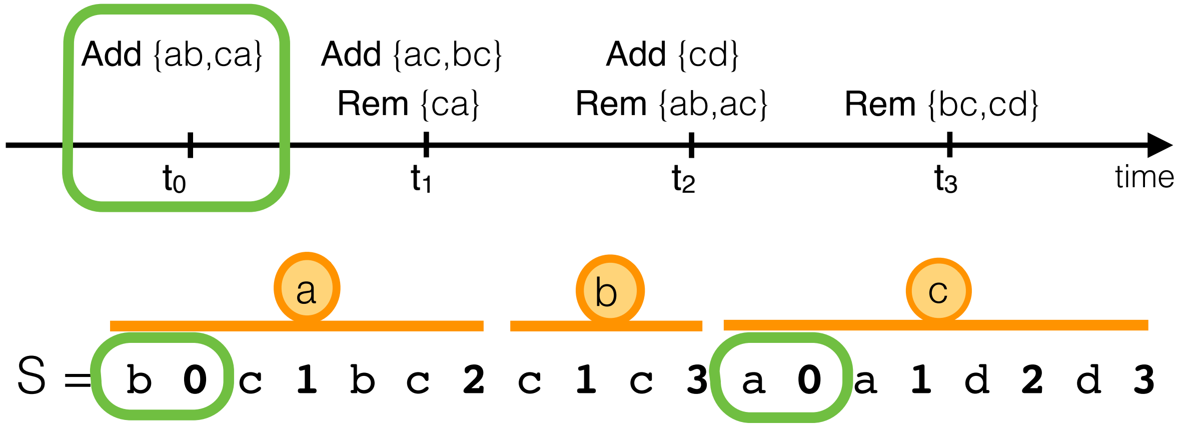
- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.



S = b **0** c **1** b c **2** c **1** c **3** a **0** a **1** d **2** d **3**

Compact Adjacency Sequence (CAS)

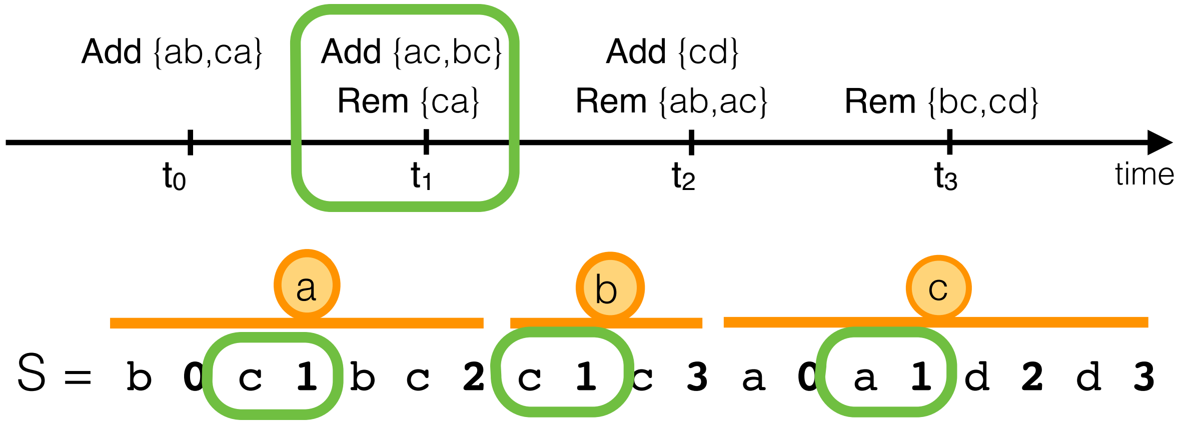
- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.



S = b **0** c **1** b c **2** c **1** c **3** a **0** a **1** d **2** d **3**

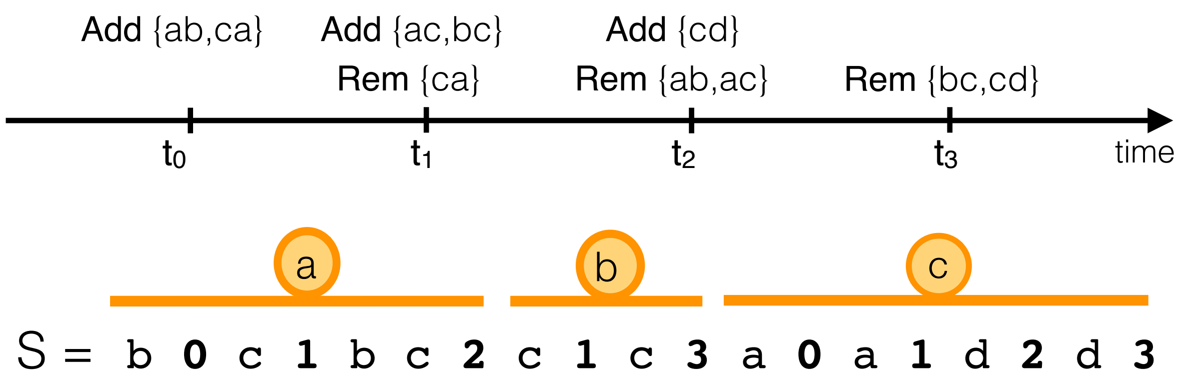
Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
    - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.



Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
    - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.



Add {ab,ca}

Add {ac,bc}
Rem {ca}

Add {cd}
Rem {ab,ac}

Rem {bc,cd}

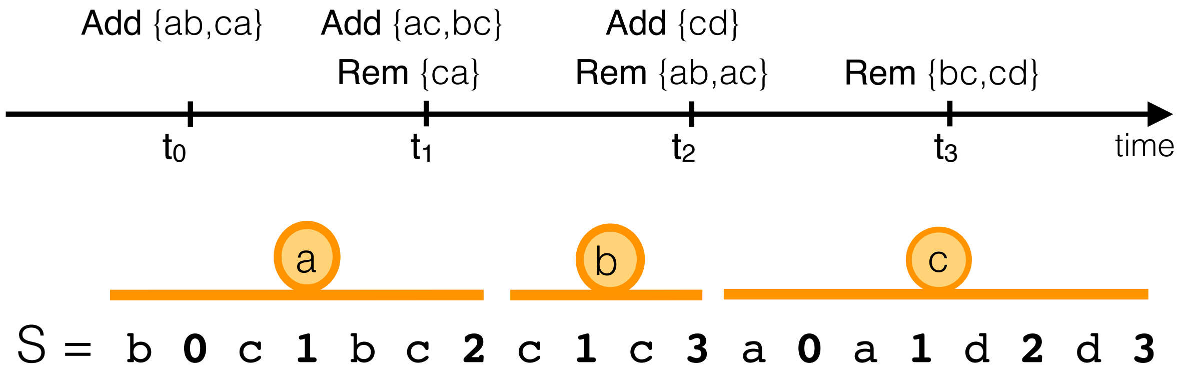
t_0 t_1 t_2 t_3 time

a b c

S = b **0** c **1** b c **2** c **1** c **3** a **0** a **1** d **2** d **3**

Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
    - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

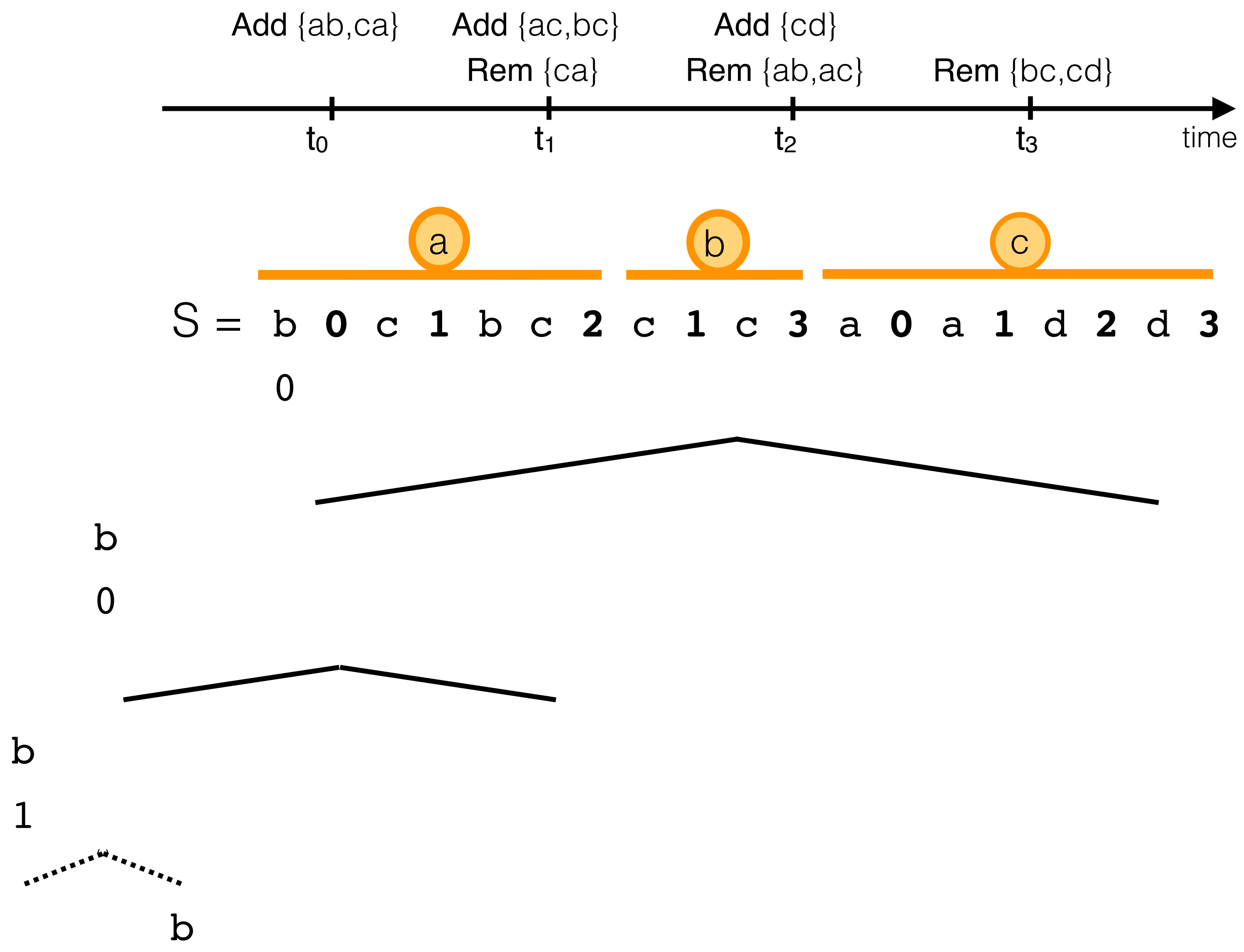


S = b **0** c **1** b c **2** c **1** c **3** a **0** a **1** d **2** d **3**

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111

Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

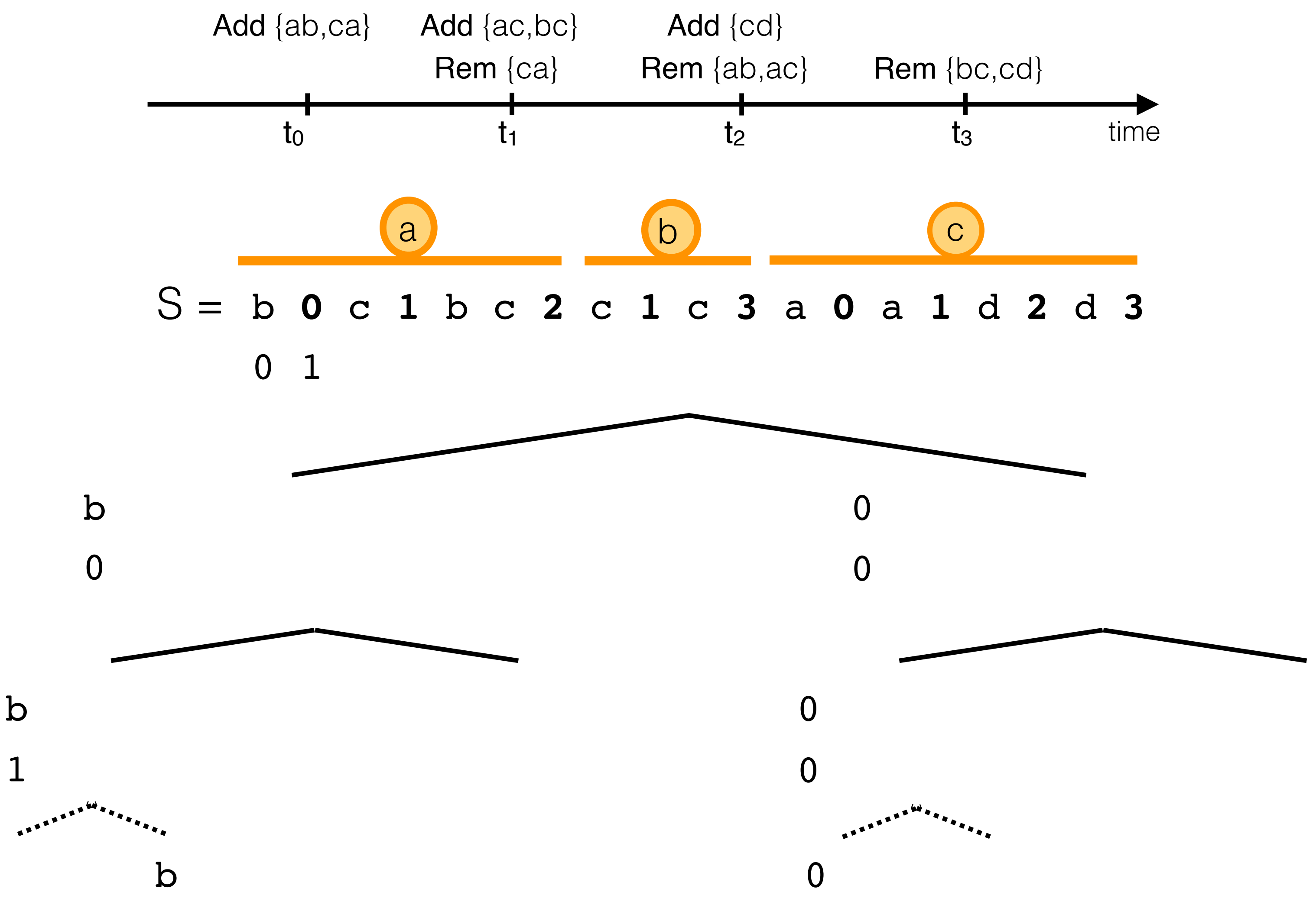


Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111

Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
    - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111

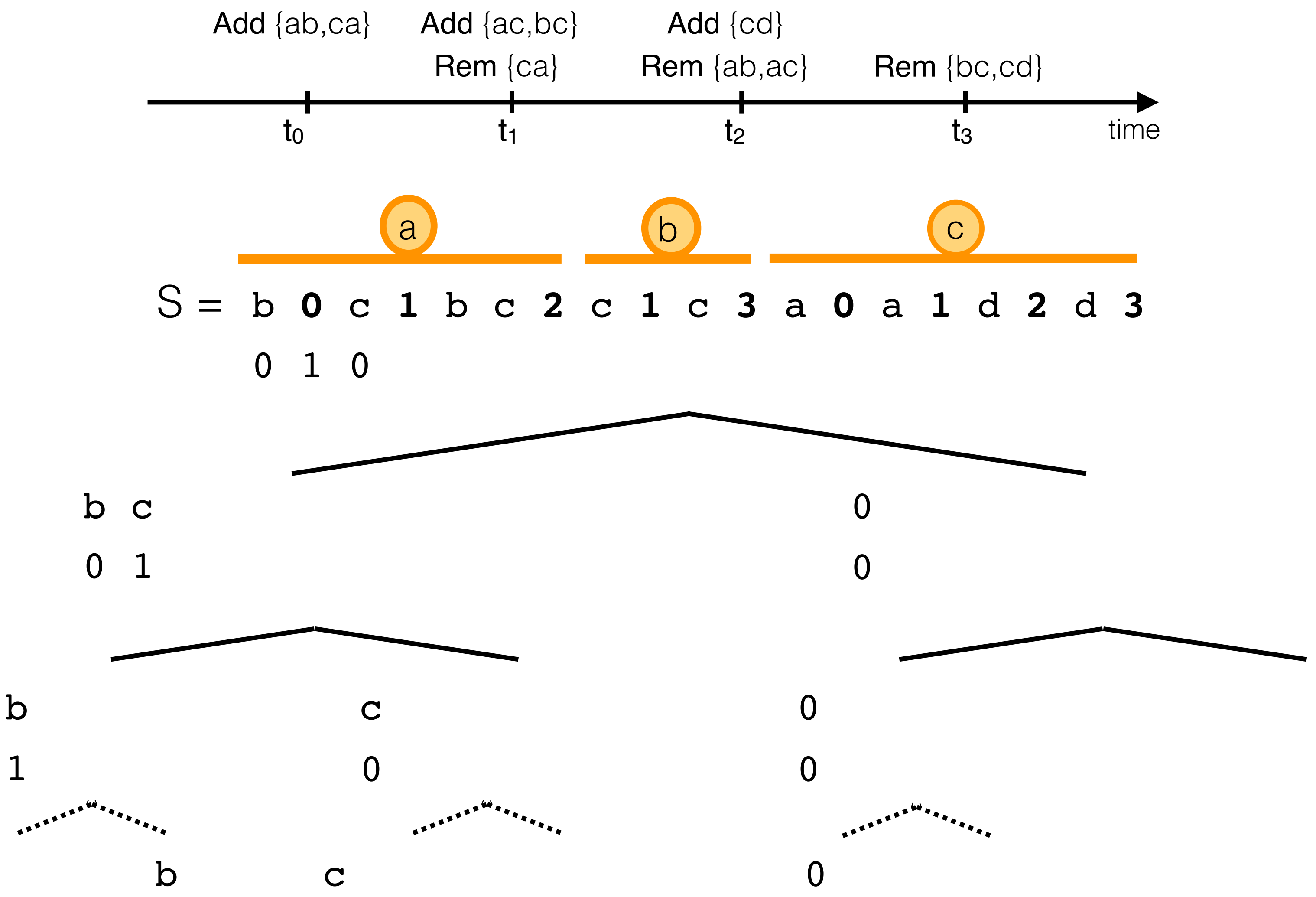


S = b **0** c **1** b c **2** c **1** c **3** a **0** a **1** d **2** d **3**

Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

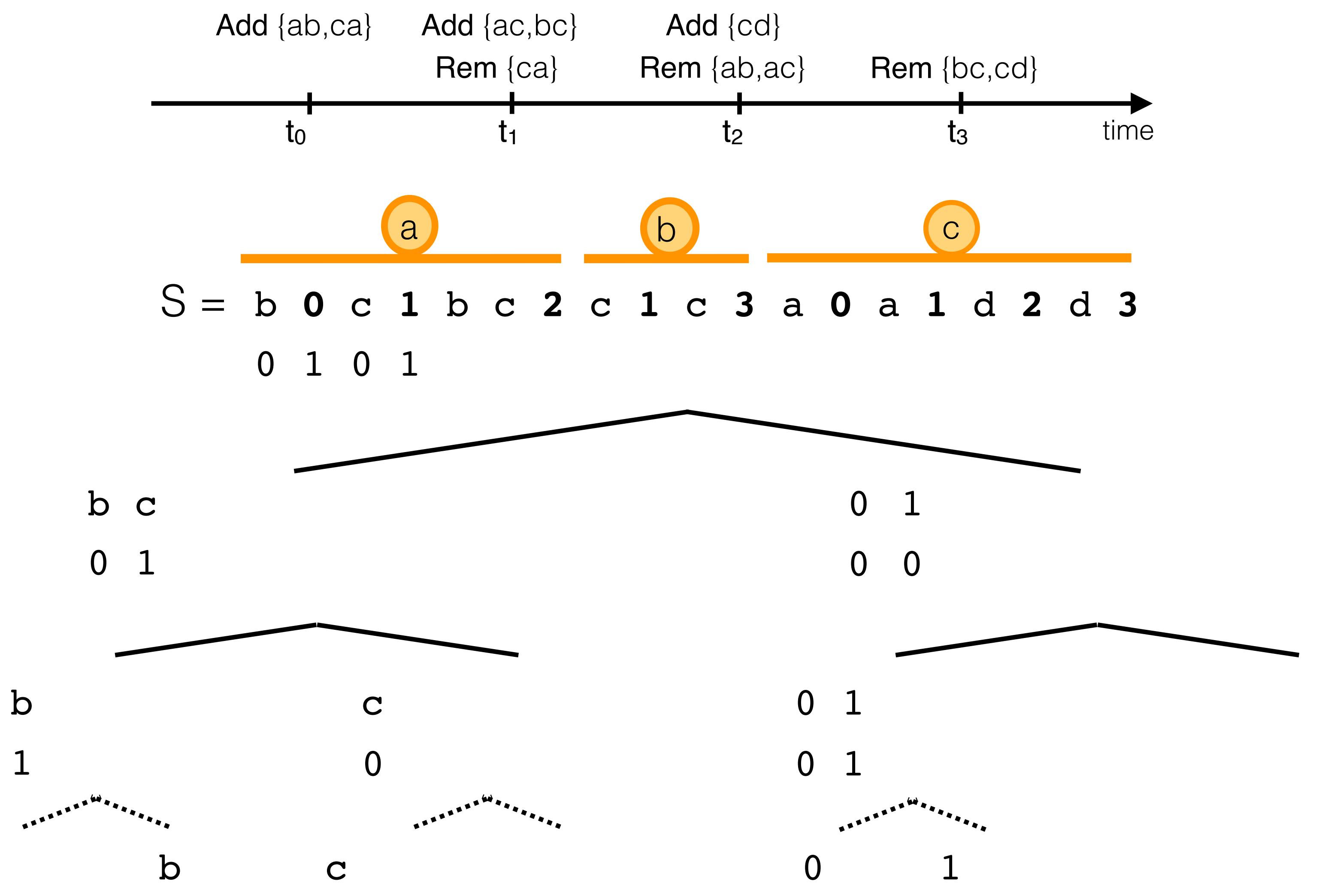
Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111



Compact Adjacency Sequence (CAS)

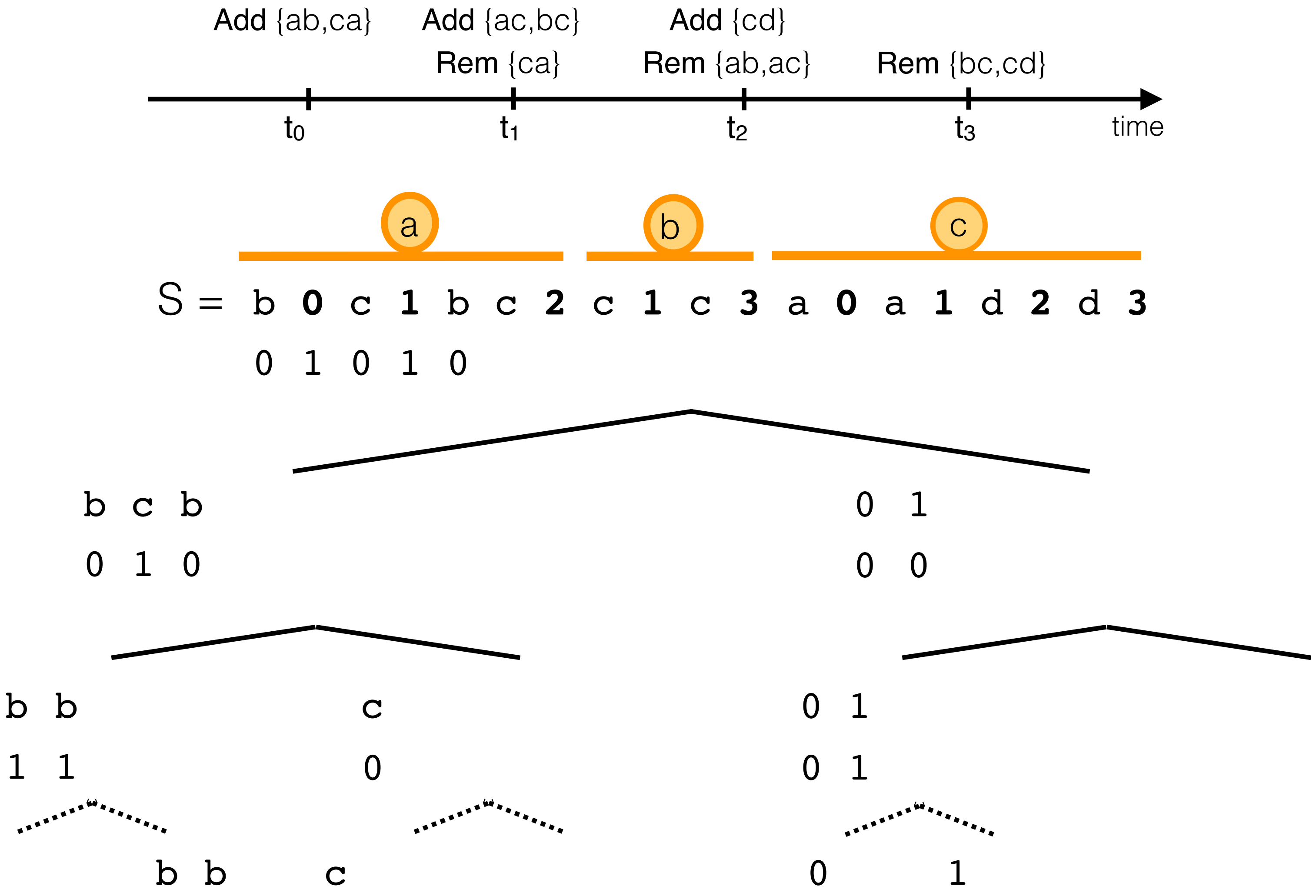
- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111



Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

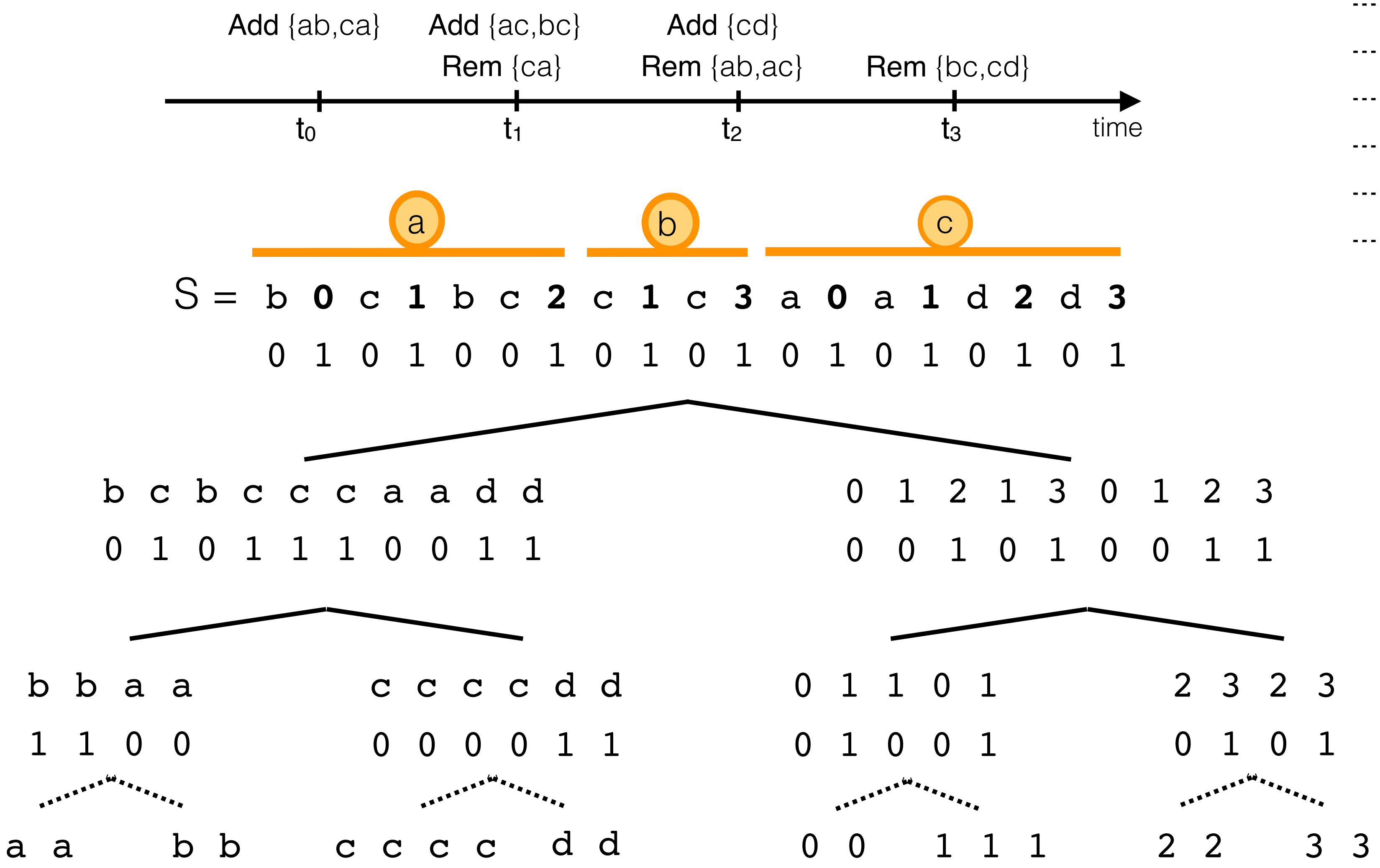


Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111

Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111



Add {ab,ca}

Add {ac,bc}
Rem {ca}

Add {cd}
Rem {ab,ac}

Rem {bc,cd}

t_0 t_1 t_2 t_3 time

a b c

S = b **0** c **1** b c **2** c **1** c **3** a **0** a **1** d **2** d **3**

0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1

b c b c c c a a d d

0 1 0 1 1 1 0 0 1 1

0 1 2 1 3 0 1 2 3

0 0 1 0 1 0 0 1 1

b b a a

1 1 0 0

c c c c d d

0 0 0 0 1 1

0 1 1 0 1

0 1 0 0 1

2 3 2 3

0 1 0 1

a a b b

c c c c d d

0 0 1 1 1

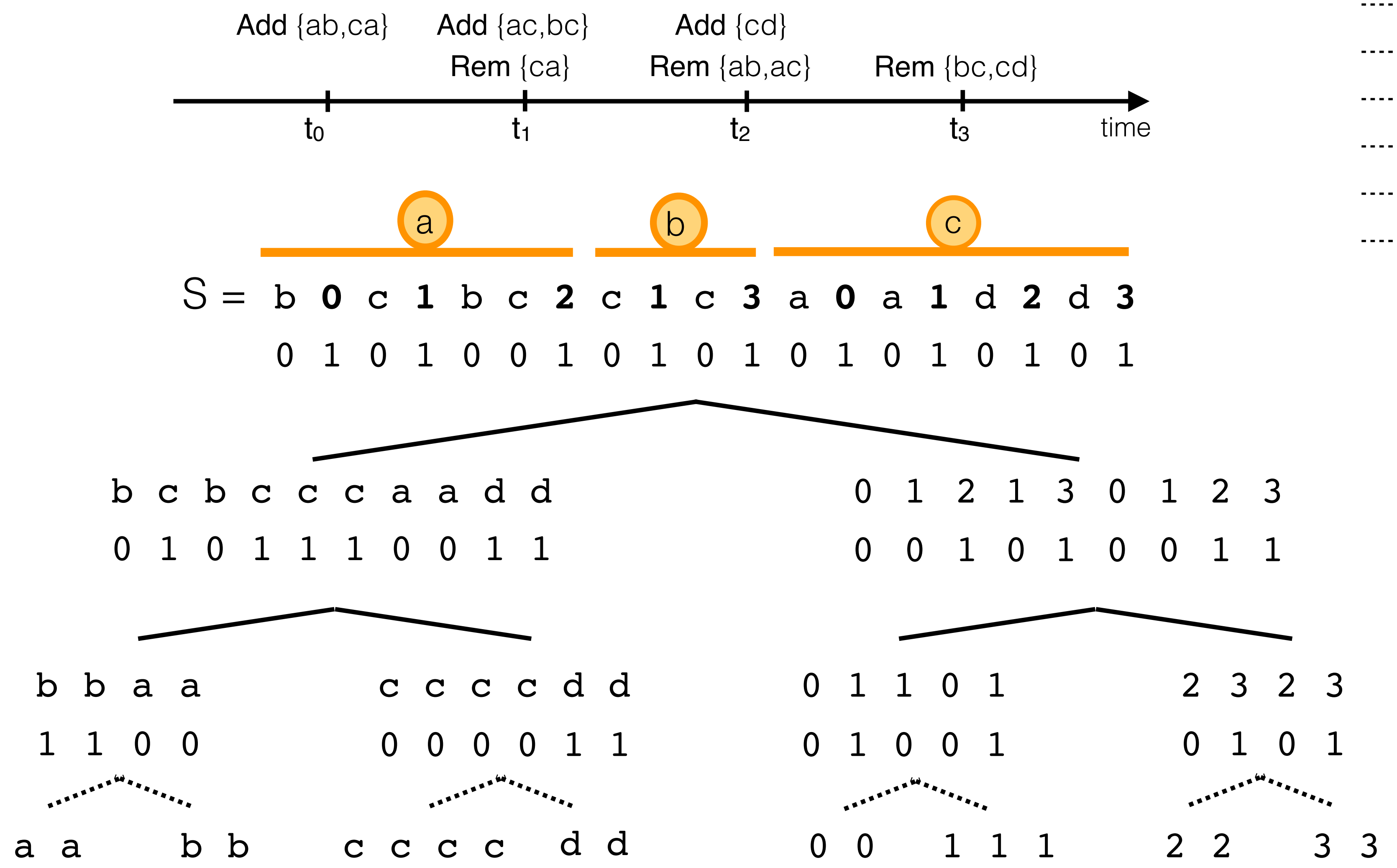
2 2 3 3

Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111

edge(ab,t=1)?



Add {ab,ca}

Add {ac,bc}
Rem {ca}

Add {cd}
Rem {ab,ac}

Rem {bc,cd}

t_0 t_1 t_2 t_3 time

a b c

S = b **0** c **1** b c **2** c **1** c **3** a **0** a **1** d **2** d **3**

0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1

b c b c c c a a d d

0 1 0 1 1 1 0 0 1 1

0 1 2 1 3 0 1 2 3

0 0 1 0 1 0 0 1 1

b b a a

1 1 0 0

c c c c d d

0 0 0 0 1 1

0 1 1 0 1

0 1 0 0 1

2 3 2 3

0 1 0 1

a a b b

c c c c d d

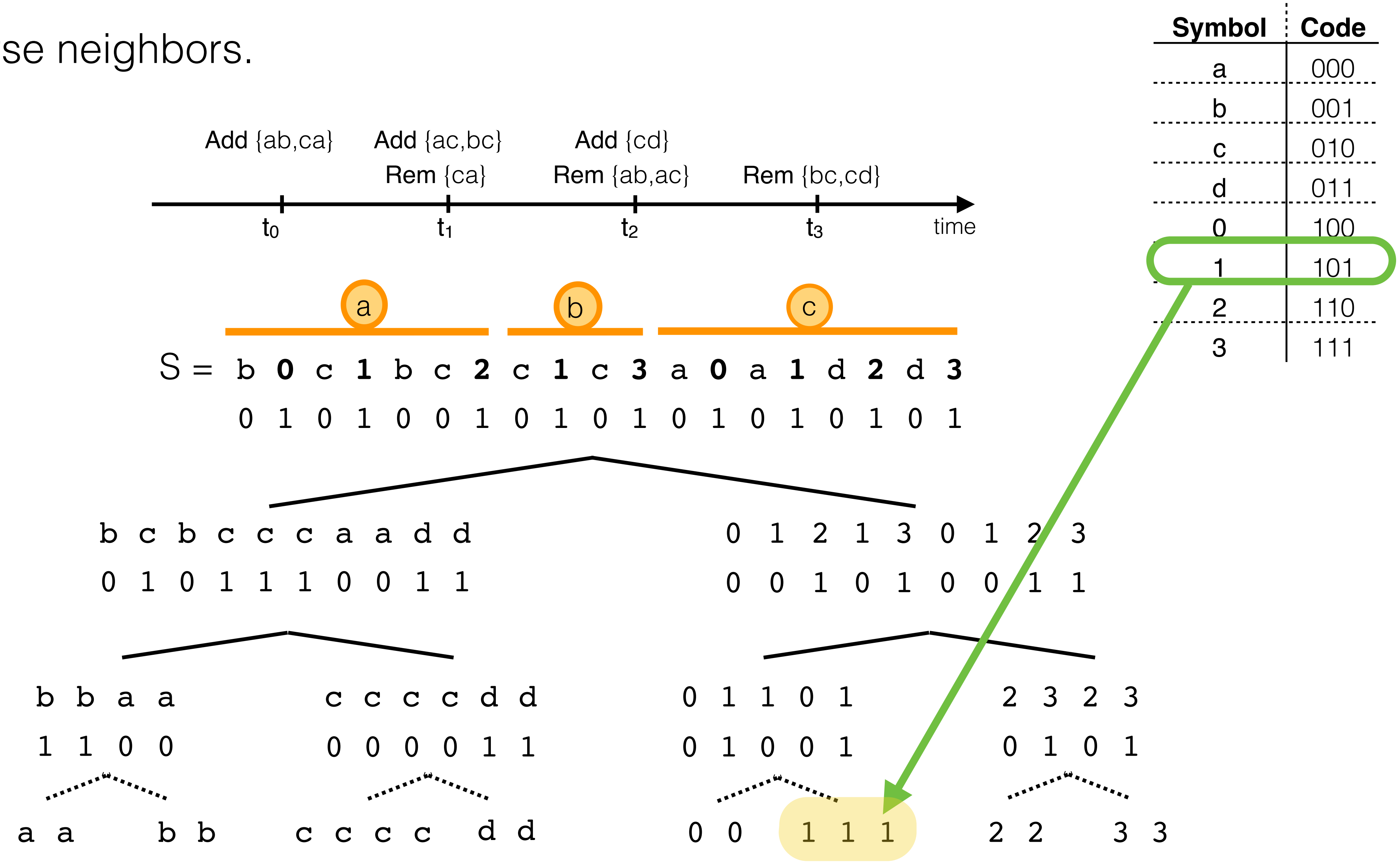
0 0 1 1 1

2 2 3 3

Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

edge(ab,t=1)?

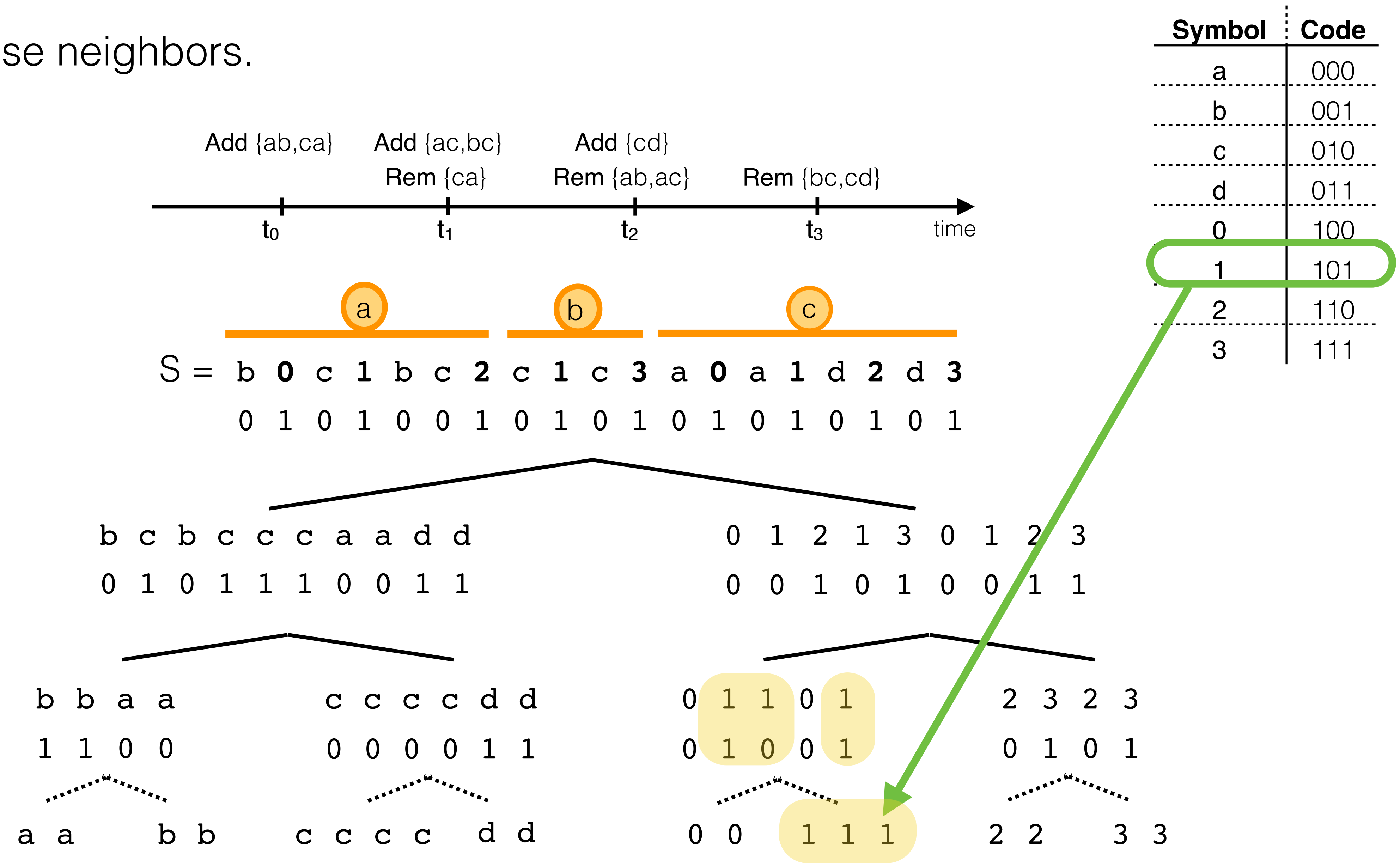


Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111

Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

edge(ab,t=1)?



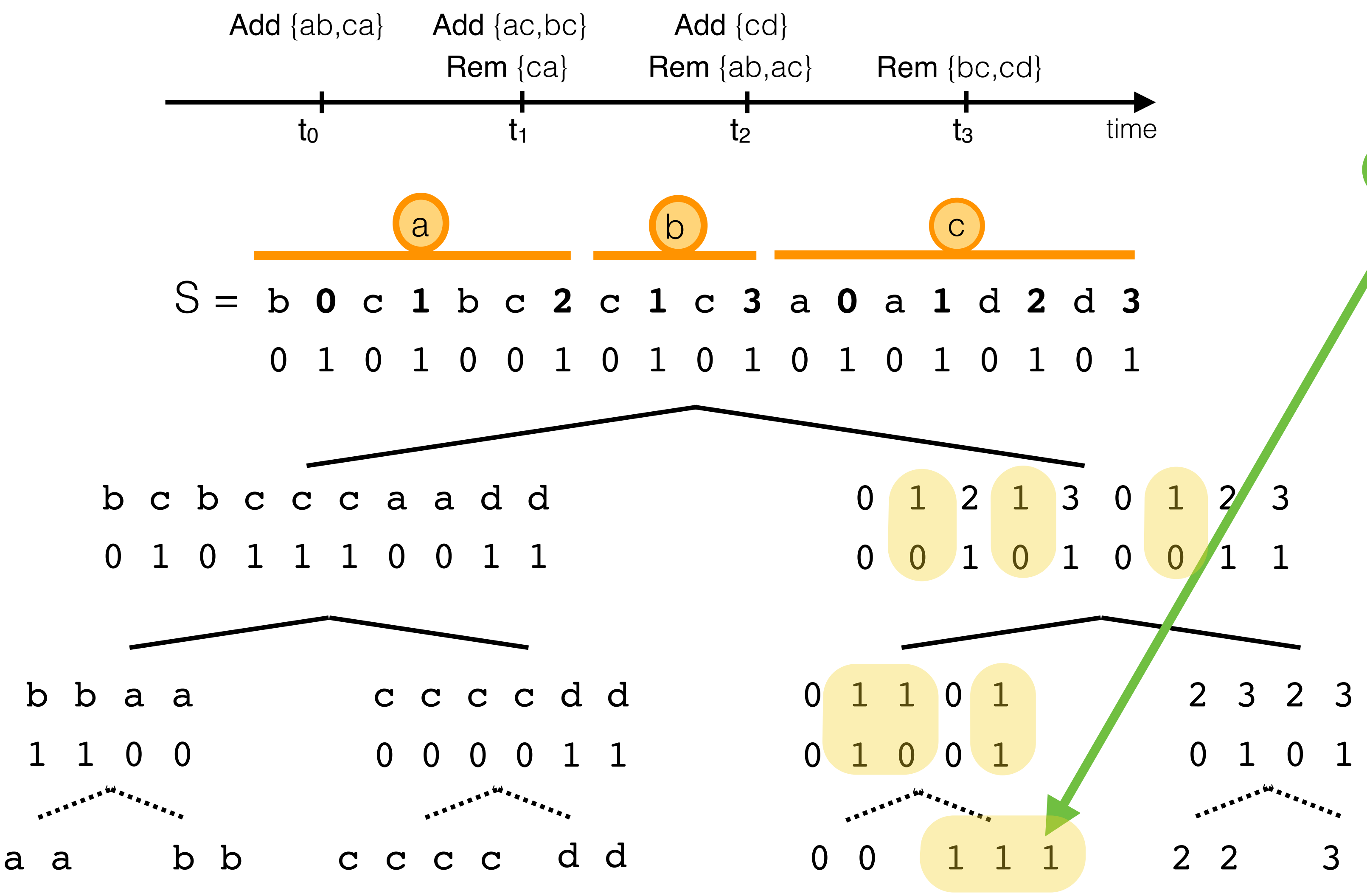
Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111

Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111

edge(ab,t=1)?

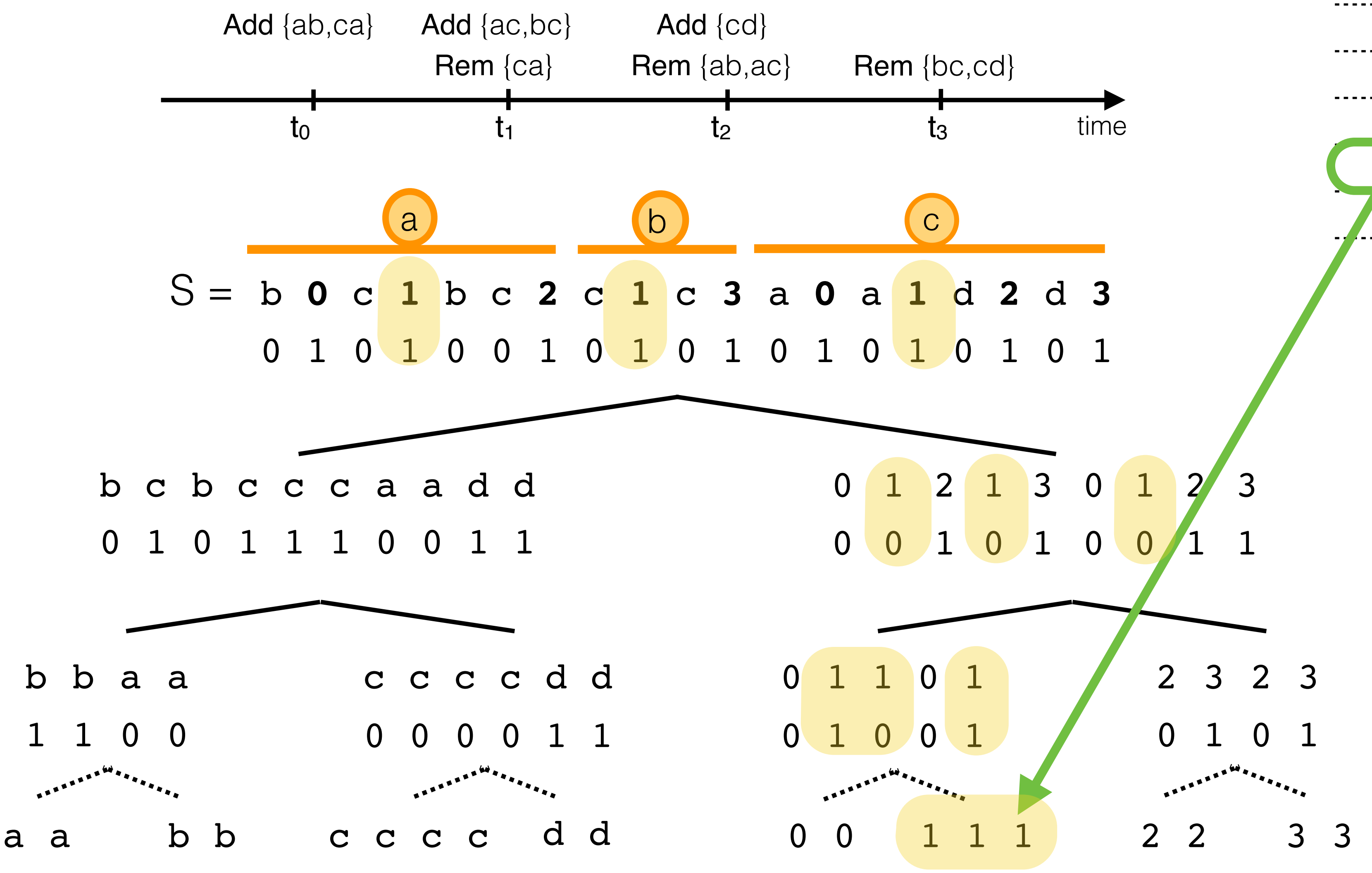


Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
    - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111

edge(ab,t=1)?



Add {ab,ca}

Add {ac,bc}
Rem {ca}

Add {cd}
Rem {ab,ac}

Rem {bc,cd}

t_0 t_1 t_2 t_3 time

a b c

S = b **0** c **1** b c **2** c **1** c **3** a **0** a **1** d **2** d **3**

0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1

b c b c c c a a d d

0 1 0 1 1 1 0 0 1 1

0 1 2 1 3 0 1 2 3

0 0 1 0 1 0 0 1 1

b b a a

1 1 0 0

c c c c d d

0 0 0 0 1 1

0 1 1 0 1

0 1 0 0 1

2 3 2 3

0 1 0 1

a a b b

c c c c d d

0 0 1 1 1

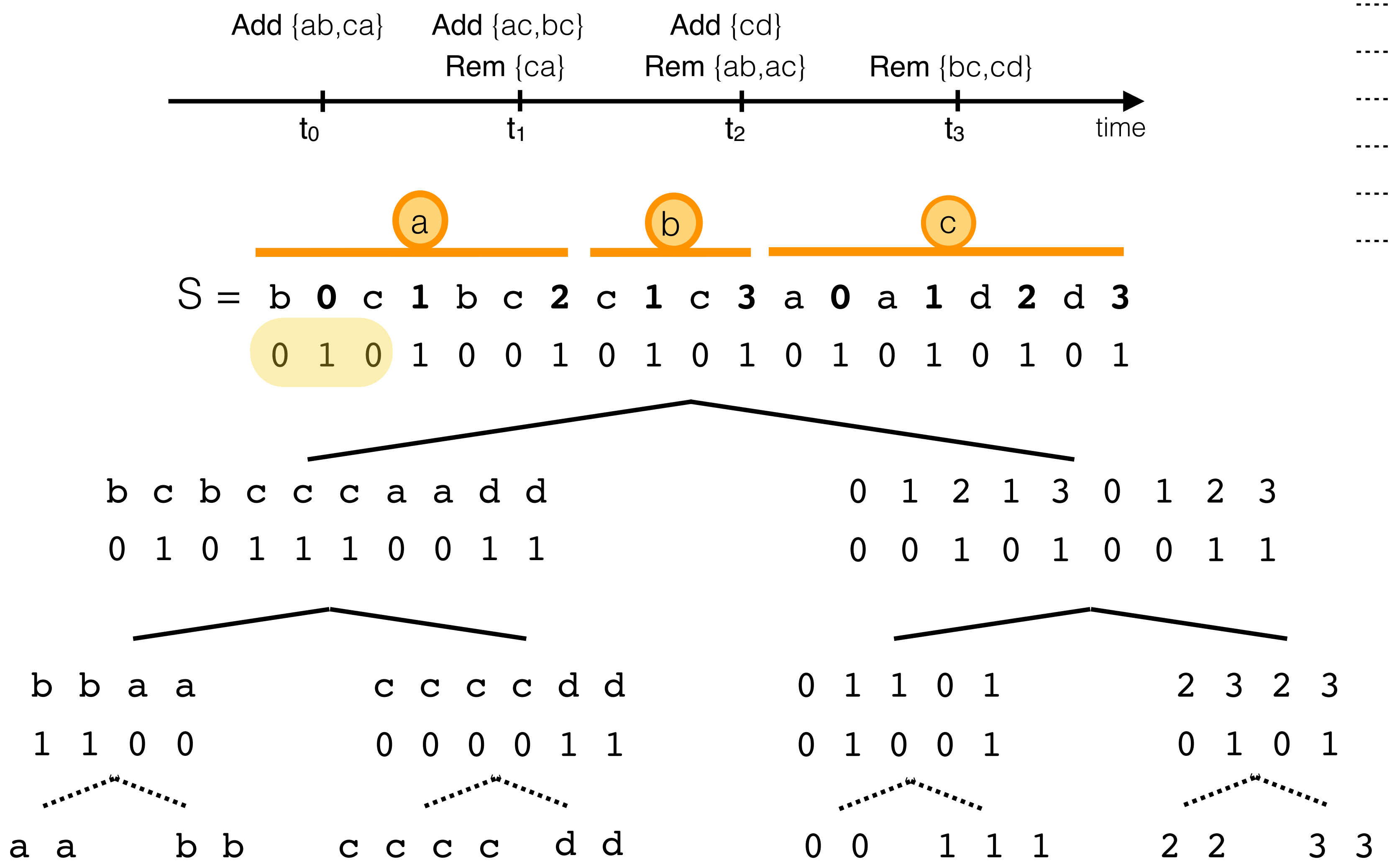
2 2 3 3

Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
    - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111

edge(ab,t=1)?



Add {ab,ca}

Add {ac,bc}
Rem {ca}

Add {cd}
Rem {ab,ac}

Rem {bc,cd}

t_0 t_1 t_2 t_3 time

a b c

S = b **0** c **1** b c **2** c **1** c **3** a **0** a **1** d **2** d **3**

0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1

b c b c c c a a d d
0 1 0 1 1 1 0 0 1 1

0 1 2 1 3 0 1 2 3
0 0 1 0 1 0 0 1 1

b b a a
1 1 0 0

c c c c d d
0 0 0 0 1 1

0 1 1 0 1
0 1 0 0 1

2 3 2 3
0 1 0 1

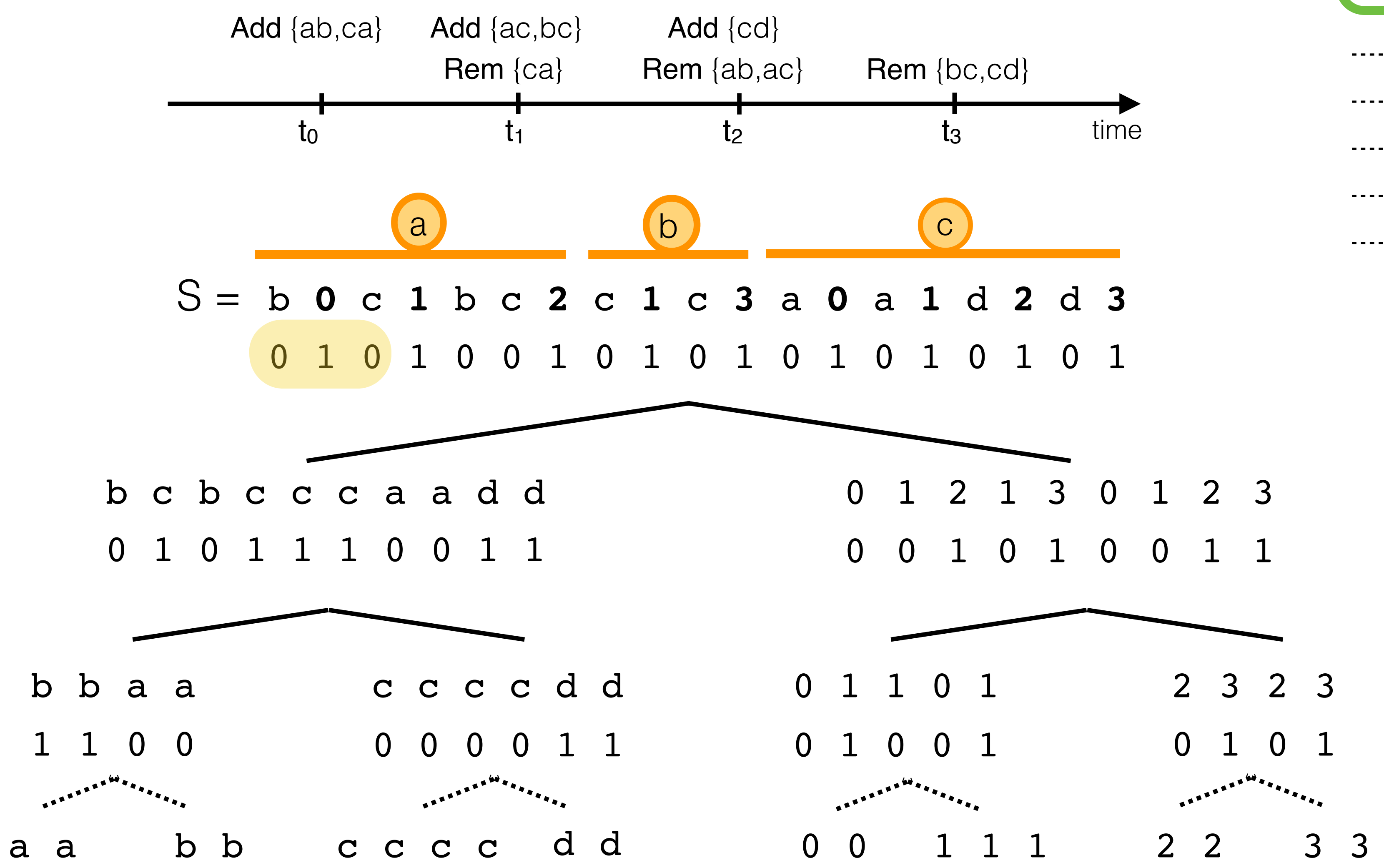
a a b b c c c c d d 0 0 1 1 1 2 2 3 3

Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
    - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111

edge(ab,t=1)?

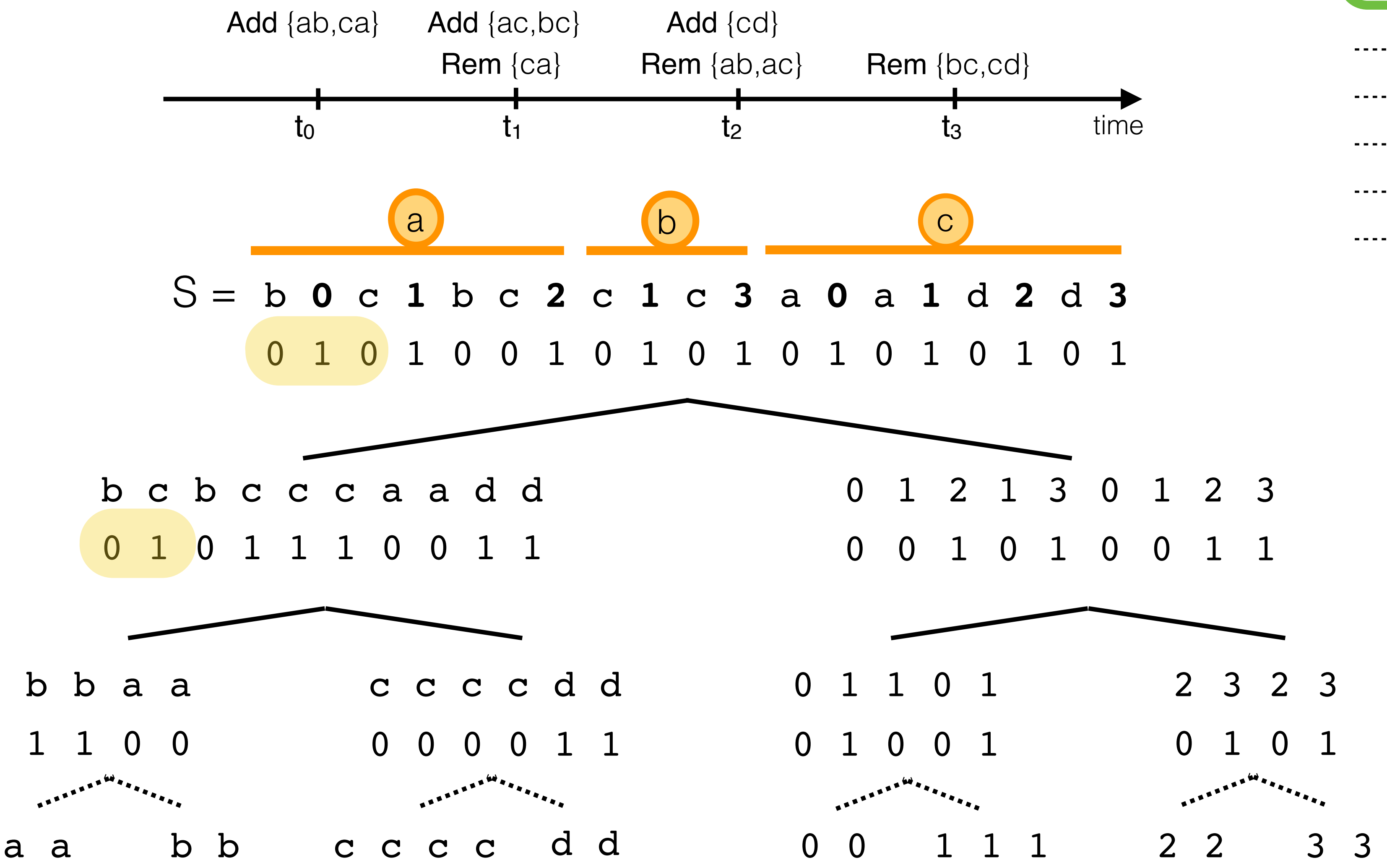


Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111

edge(ab,t=1)?



Add {ab,ca}

Add {ac,bc}
Rem {ca}

Add {cd}
Rem {ab,ac}

Rem {bc,cd}

t_0 t_1 t_2 t_3 time

a b c

S = b **0** c **1** b c **2** c **1** c **3** a **0** a **1** d **2** d **3**

0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1

b c b c c c a a d d
0 1 0 1 1 1 0 0 1 1

0 1 2 1 3 0 1 2 3
0 0 1 0 1 0 0 1 1

b b a a
1 1 0 0

c c c c d d
0 0 0 0 1 1

0 1 1 0 1
0 1 0 0 1

2 3 2 3
0 1 0 1

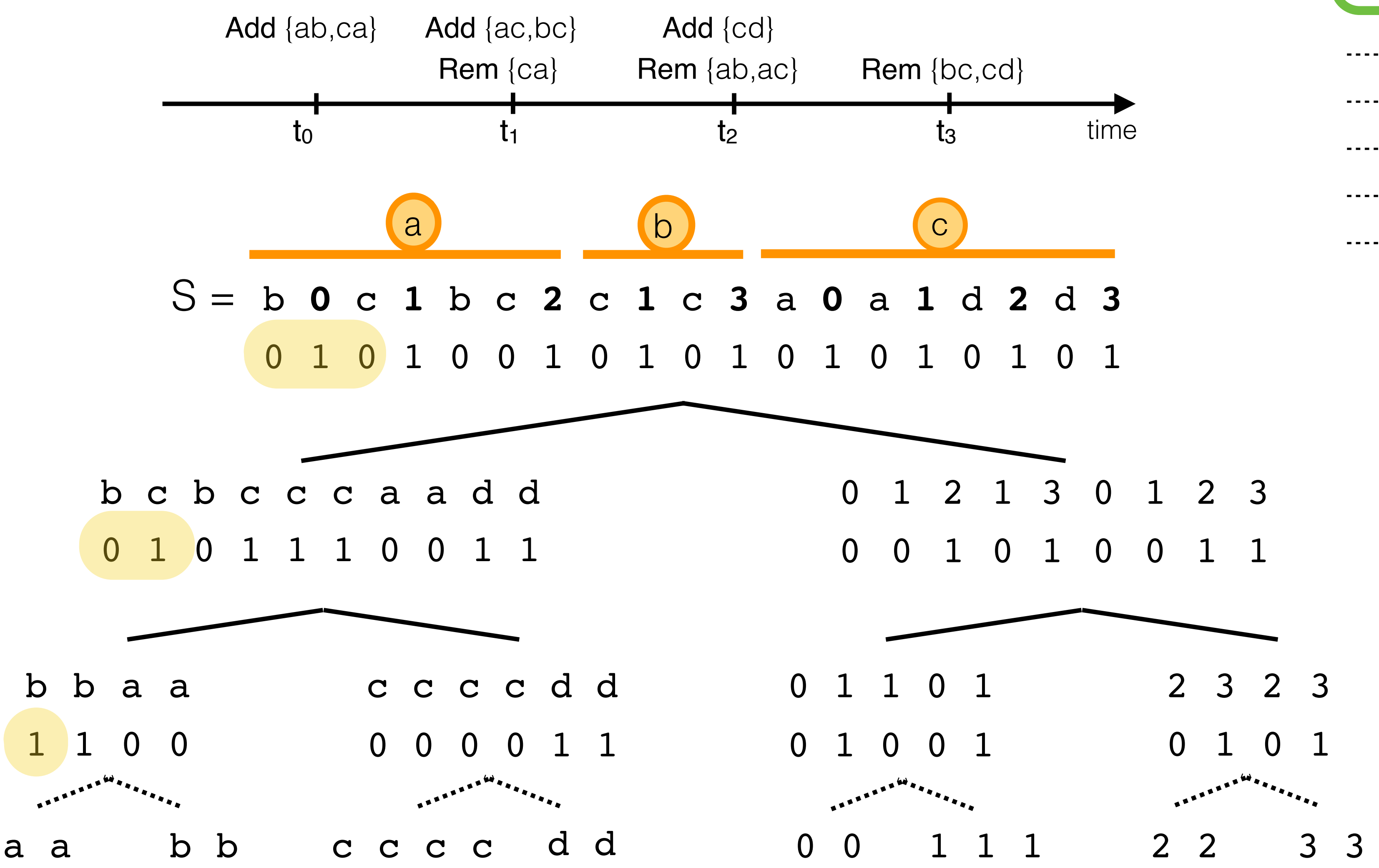
a a b b c c c c d d 0 0 1 1 1 2 2 3 3

Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111

edge(ab,t=1)?

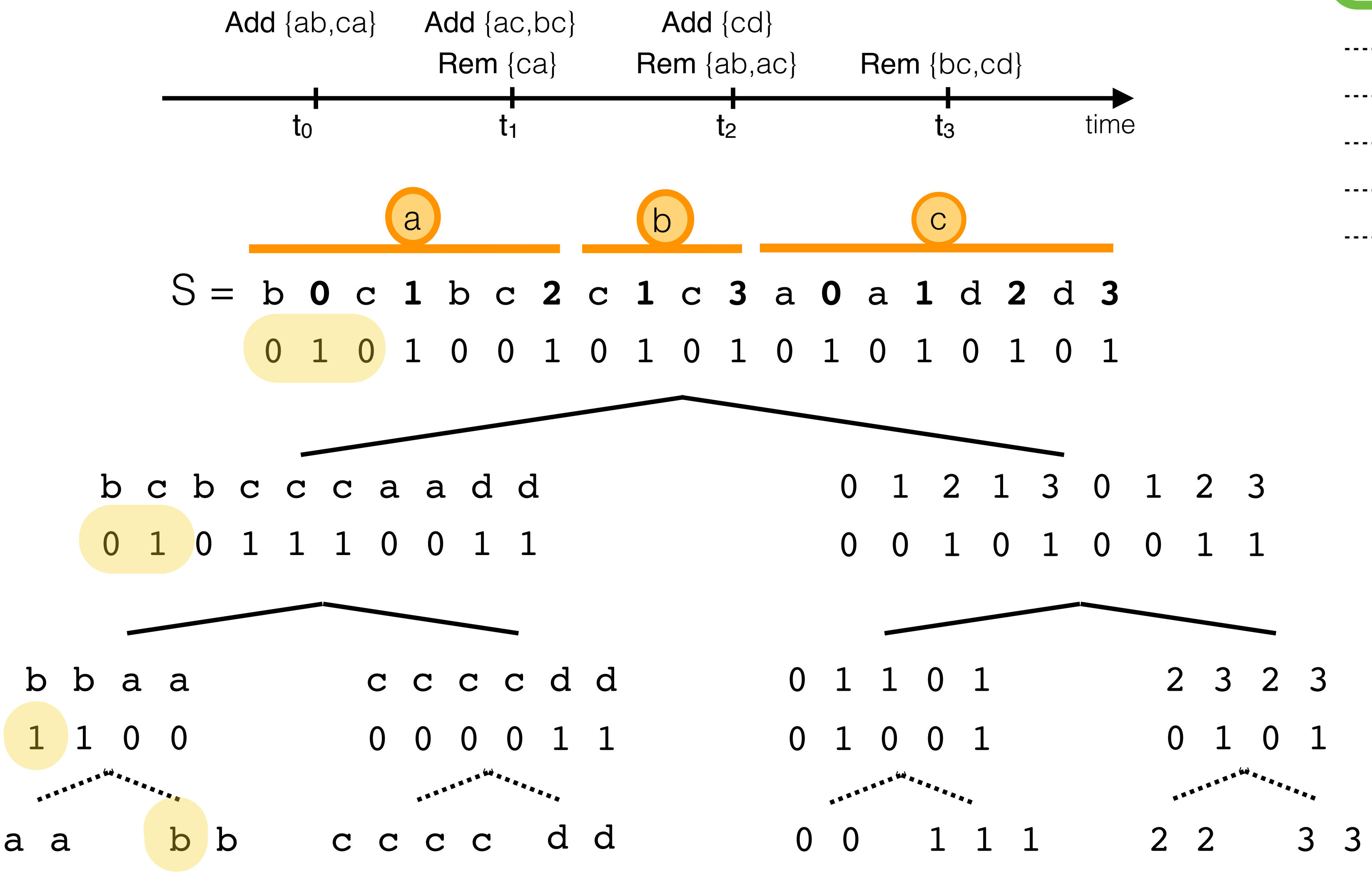


Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111

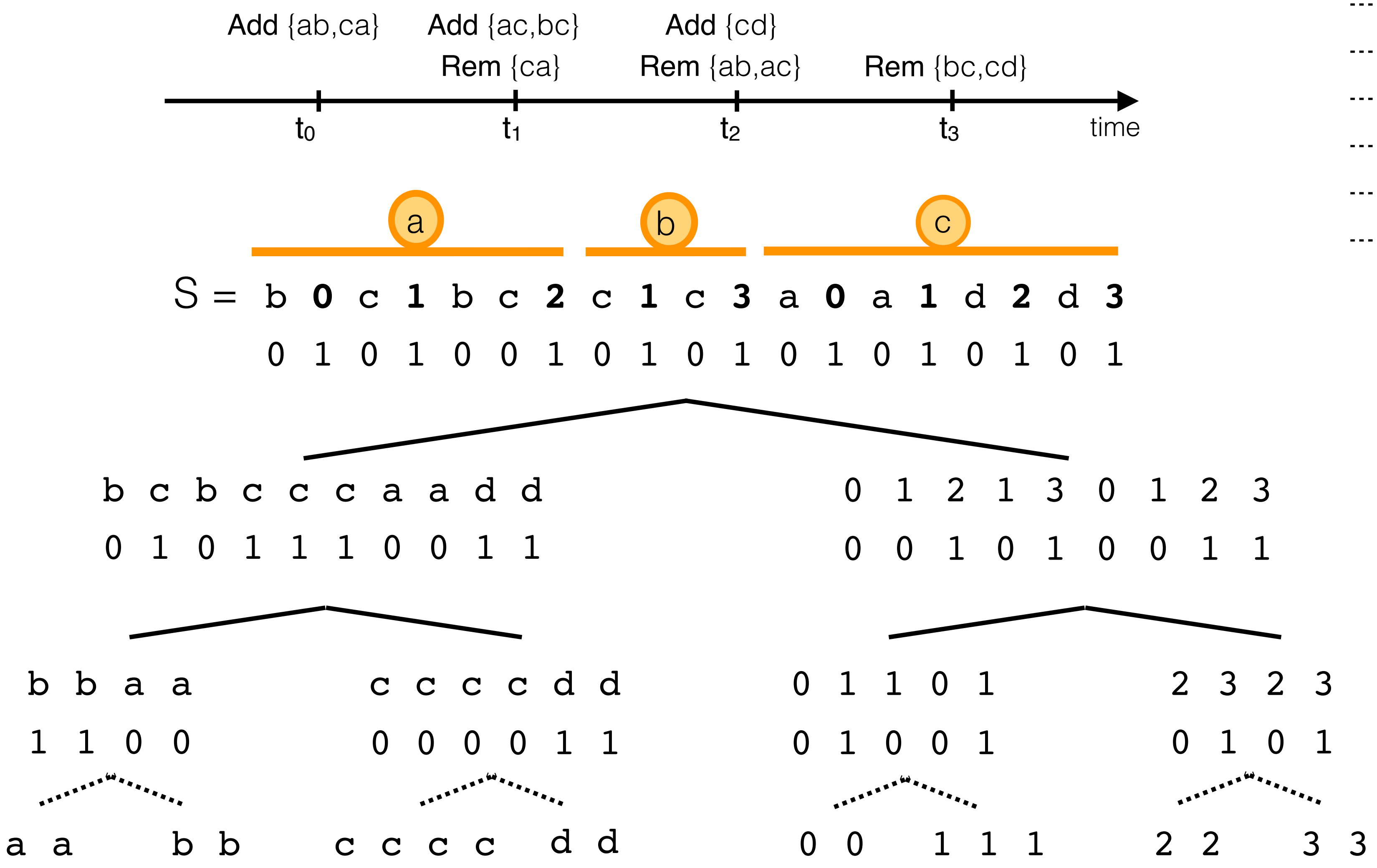
edge(ab,t=1)?



Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111



Add {ab,ca}
Add {ac,bc}
Rem {ca}
Add {cd}
Rem {ab,ac}
Rem {bc,cd}

t_0 t_1 t_2 t_3 time

a b c

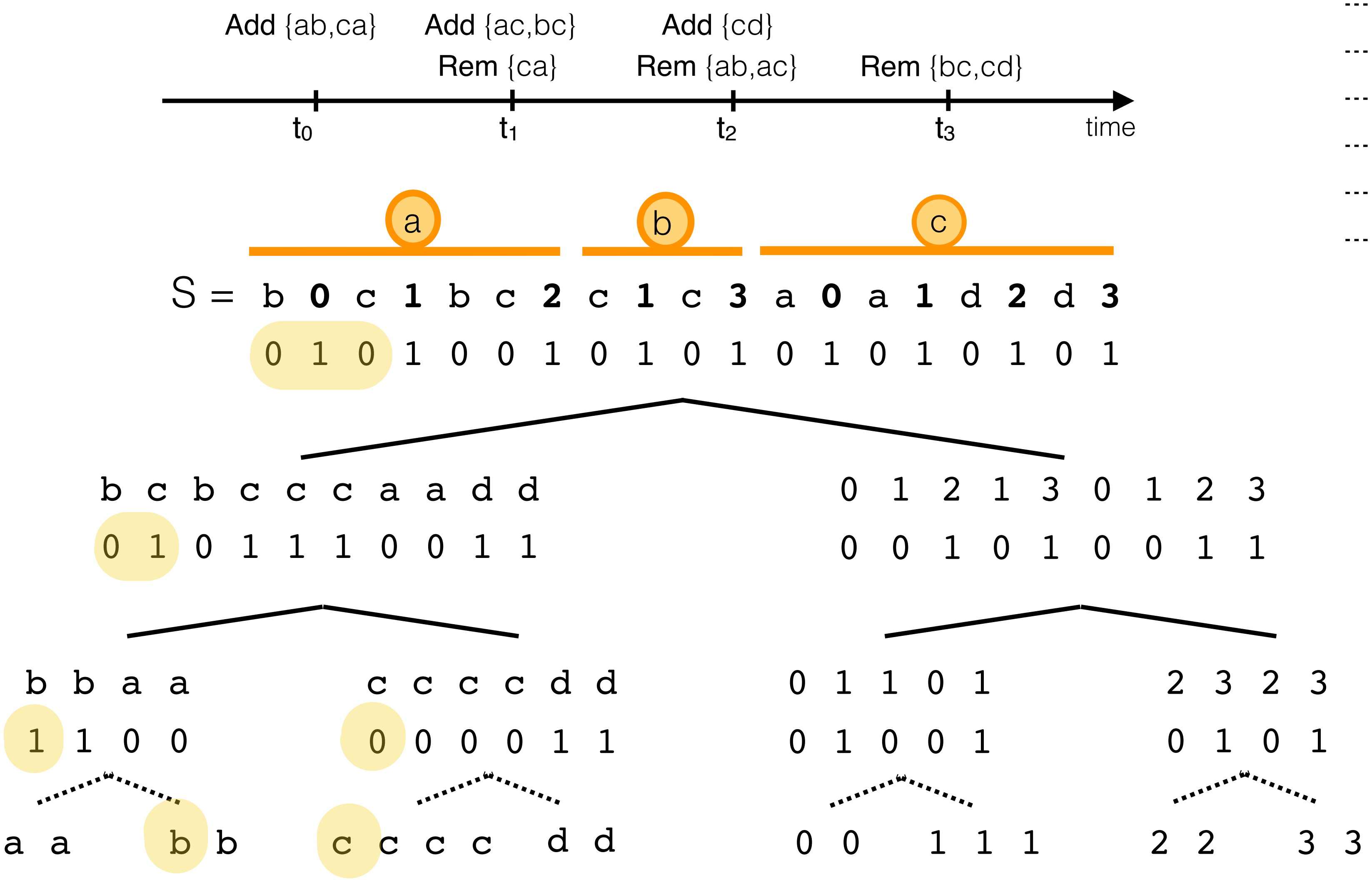
```
S = b 0 c 1 b c 2 c 1 c 3 a 0 a 1 d 2 d 3
    0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1

b c b c c c a a d d        0 1 2 1 3 0 1 2 3
0 1 0 1 1 1 0 0 1 1        0 0 1 0 1 0 0 1 1

b b a a    c c c c d d    0 1 1 0 1    2 3 2 3
1 1 0 0    0 0 0 0 1 1    0 1 0 0 1    0 1 0 1

a a  b b   c c c c  d d   0 0  1 1 1   2 2  3 3
```

dirnei(a,t=1)?

Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
    - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111



S = b **0** c **1** b c **2** c **1** c **3** a **0** a **1** d **2** d **3**

0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1

b c b c c c a a d d

0 1 0 1 1 1 0 0 1 1

0 1 2 1 3 0 1 2 3

0 0 1 0 1 0 0 1 1

b b a a

1 1 0 0

c c c c d d

0 0 0 0 1 1

0 1 1 0 1

0 1 0 0 1

2 3 2 3

0 1 0 1

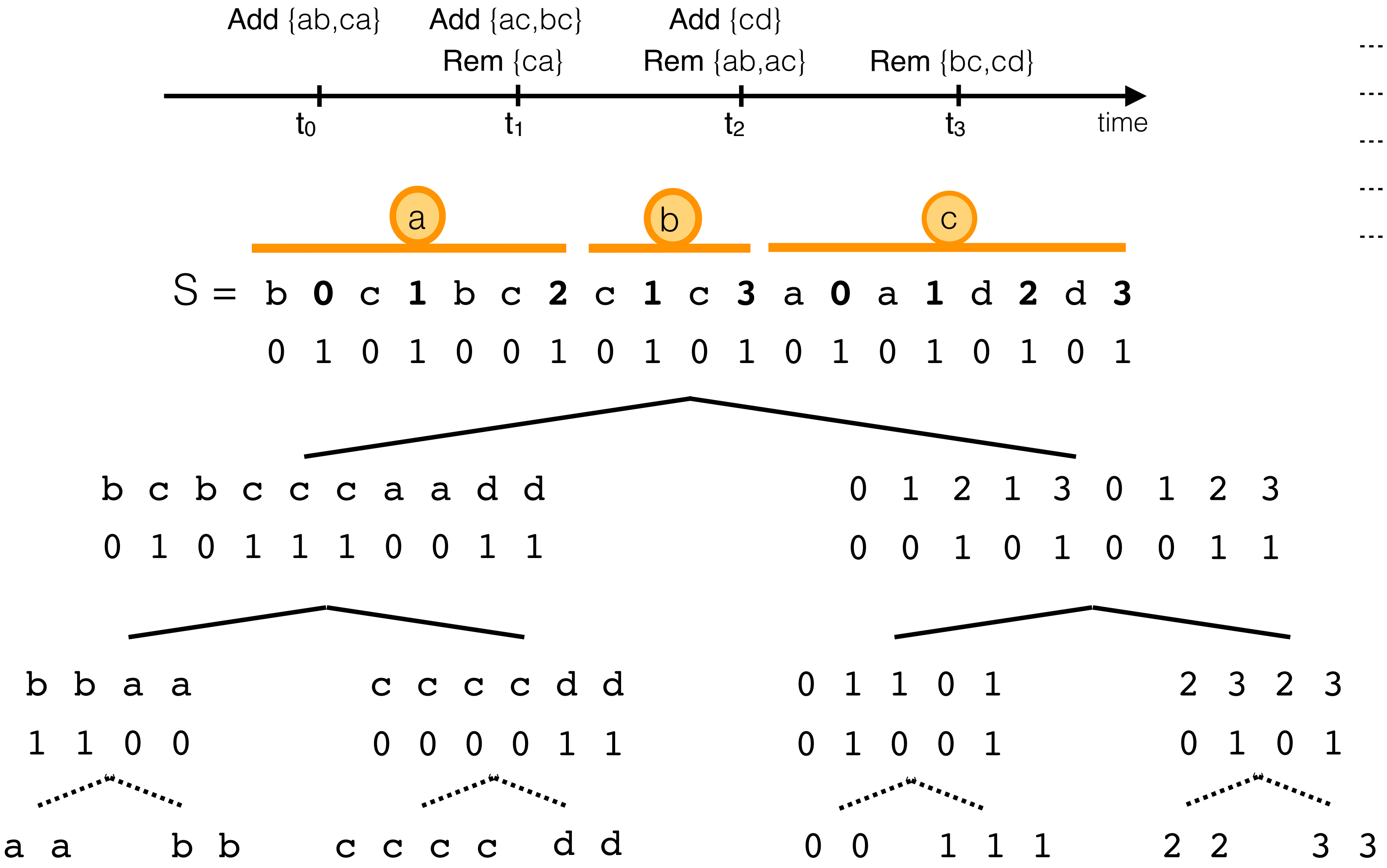
a a b b c c c c d d 0 0 1 1 1 2 2 3 3

dirnei(a,t=1)?

Compact Adjacency Sequence (CAS)

- Transform the Temporal Log into a sequence in a Wavelet Tree.
  - Events ordered by vertex and then, by time into a long sequence.
- Parity property is replaced by count in Wavelet Tree.
- Slow for reverse neighbors.

Symbol	Code
a	000
b	001
c	010
d	011
0	100
1	101
2	110
3	111



S = b **0** c **1** b c **2** c **1** c **3** a **0** a **1** d **2** d **3**

0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1

b c b c c c a a d d

0 1 0 1 1 1 0 0 1 1

0 1 2 1 3 0 1 2 3

0 0 1 0 1 0 0 1 1

b b a a

1 1 0 0

c c c c d d

0 0 0 0 1 1

0 1 1 0 1

0 1 0 0 1

2 3 2 3

0 1 0 1

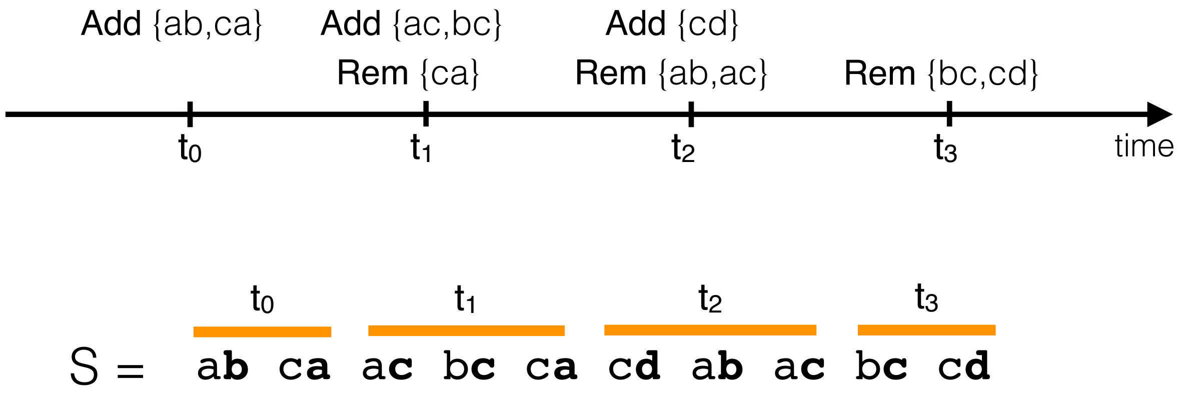
a a b b c c c c d d 0 0 1 1 1 2 2 3 3

Outline

- ✓ Definition and Motivation.
- ✓ Previous works about temporal graphs.
- ✓ Compression of temporal graphs.
- Contributions
    - ✓ Based on inverted indexes:
        - ✓ EdgeLog
        - ✓ EveLog
    - Based on Wavelet Trees:
        - ✓ Compact Adjacency Sequence (CAS)
        - Compact Events ordered by Time (CET)
    - Based on the Compressed Suffix Array:
        - Temporal Graph CSA
    - Based on the multidimensional k^d-tree
        - The Compressed k^d-tree
- Evaluation.
- Conclusions and future works.

Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
    - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.



Add {ab,ca}

Add {ac,bc}
Rem {ca}

Add {cd}
Rem {ab,ac}

Rem {bc,cd}

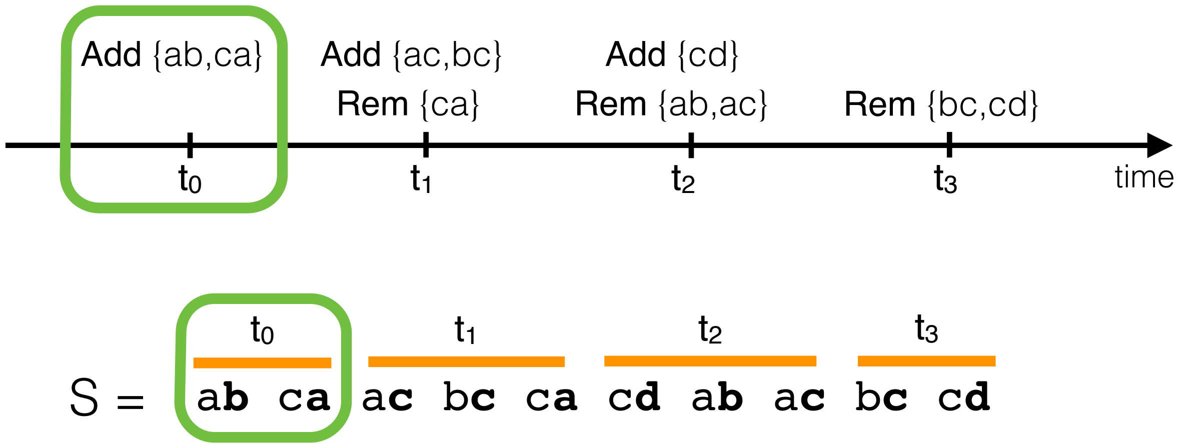
t_0 t_1 t_2 t_3 time

t_0 t_1 t_2 t_3

S = **ab ca ac bc ca cd ab ac bc cd**

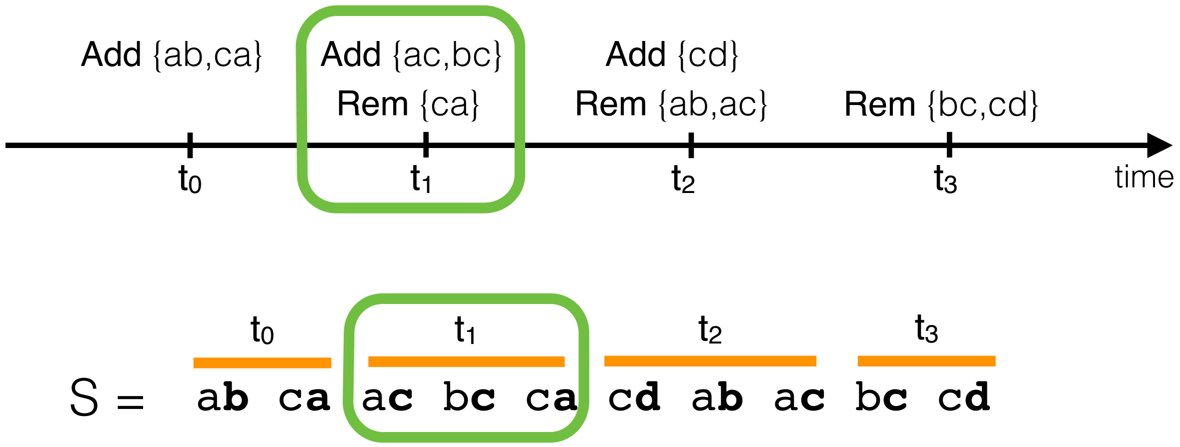
Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.



Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.



Add {ab,ca}

Add {ac,bc}
Rem {ca}

Add {cd}
Rem {ab,ac}

Rem {bc,cd}

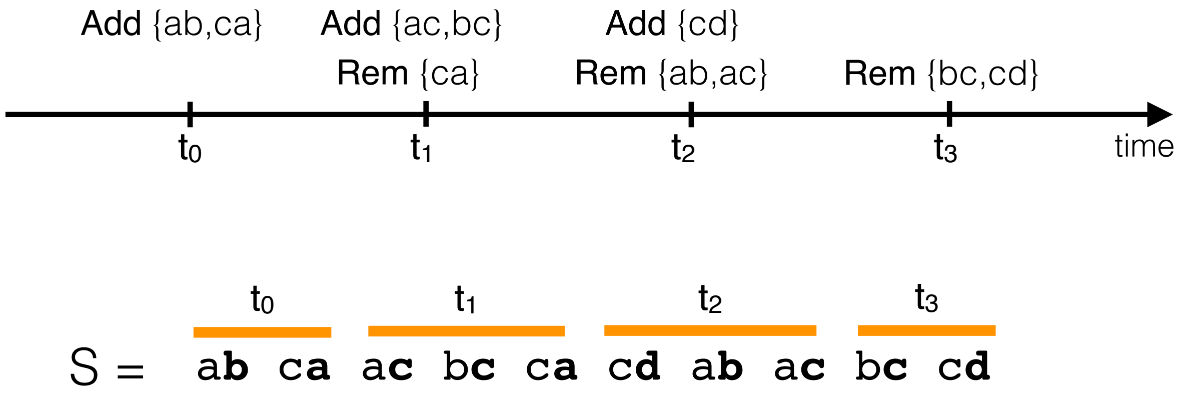
t_0 t_1 t_2 t_3 time

t_0 t_1 t_2 t_3

S = **ab ca ac bc ca cd ab ac bc cd**

Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
    - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.



Add {ab,ca}

Add {ac,bc}
Rem {ca}

Add {cd}
Rem {ab,ac}

Rem {bc,cd}

t_0 t_1 t_2 t_3 time

t_0 t_1 t_2 t_3

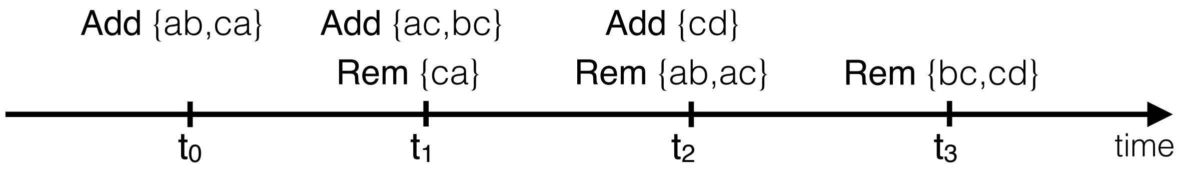
S = **ab ca ac bc ca cd ab ac bc cd**

Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**



t_0		t_1			t_2			t_3	
ab	**ca**	**ac**	**bc**	**ca**	**cd**	**ab**	**ac**	**bc**	**cd**

S = **ab ca ac bc ca cd ab ac bc cd**

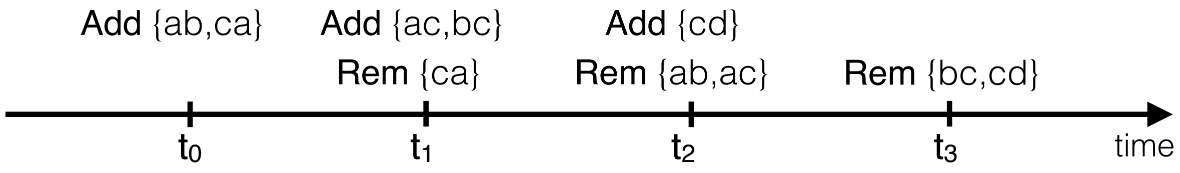
Interleaving Code	
ab	0**0**0**1**
ac	0**1**0**0**
bc	0**1**1**0**
ca	1**0**0**0**
cd	1**1**0**1**

Compact Events ordered by Time (CET)

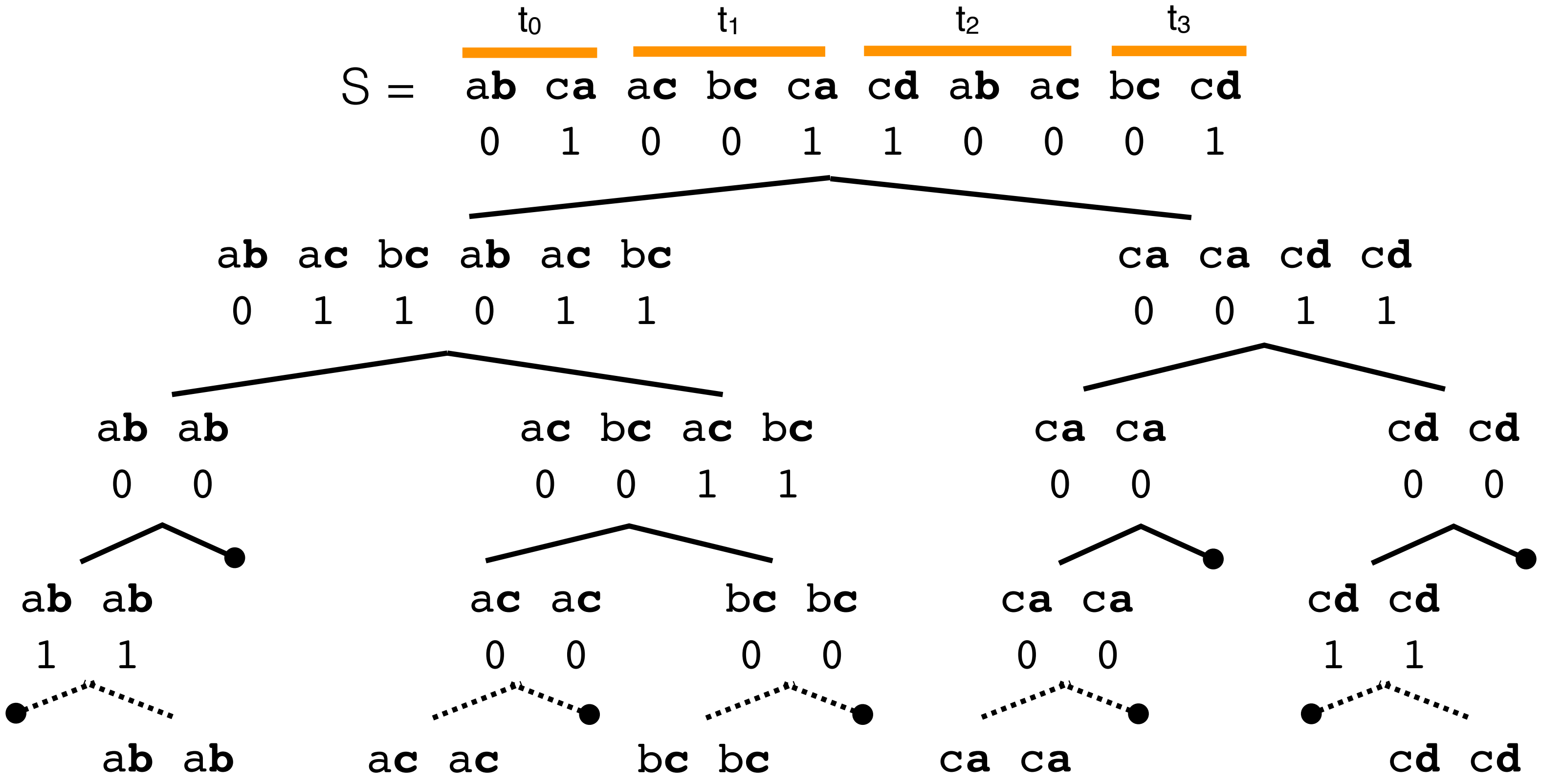
- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
    - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**

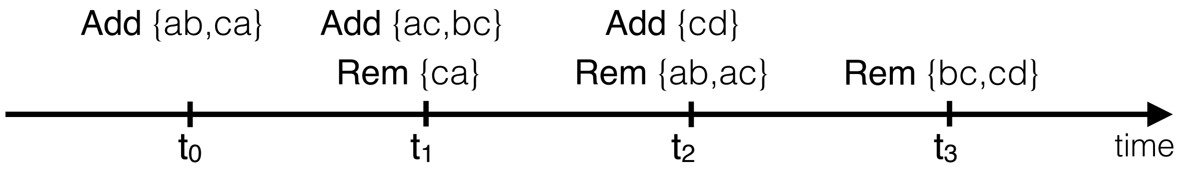


Interleaving Code	
ab	0**0**0**1**
ac	0**1**0**0**
bc	0**1**1**0**
ca	1**0**0**0**
cd	1**1**0**1**



Compact Events ordered by Time (CET)

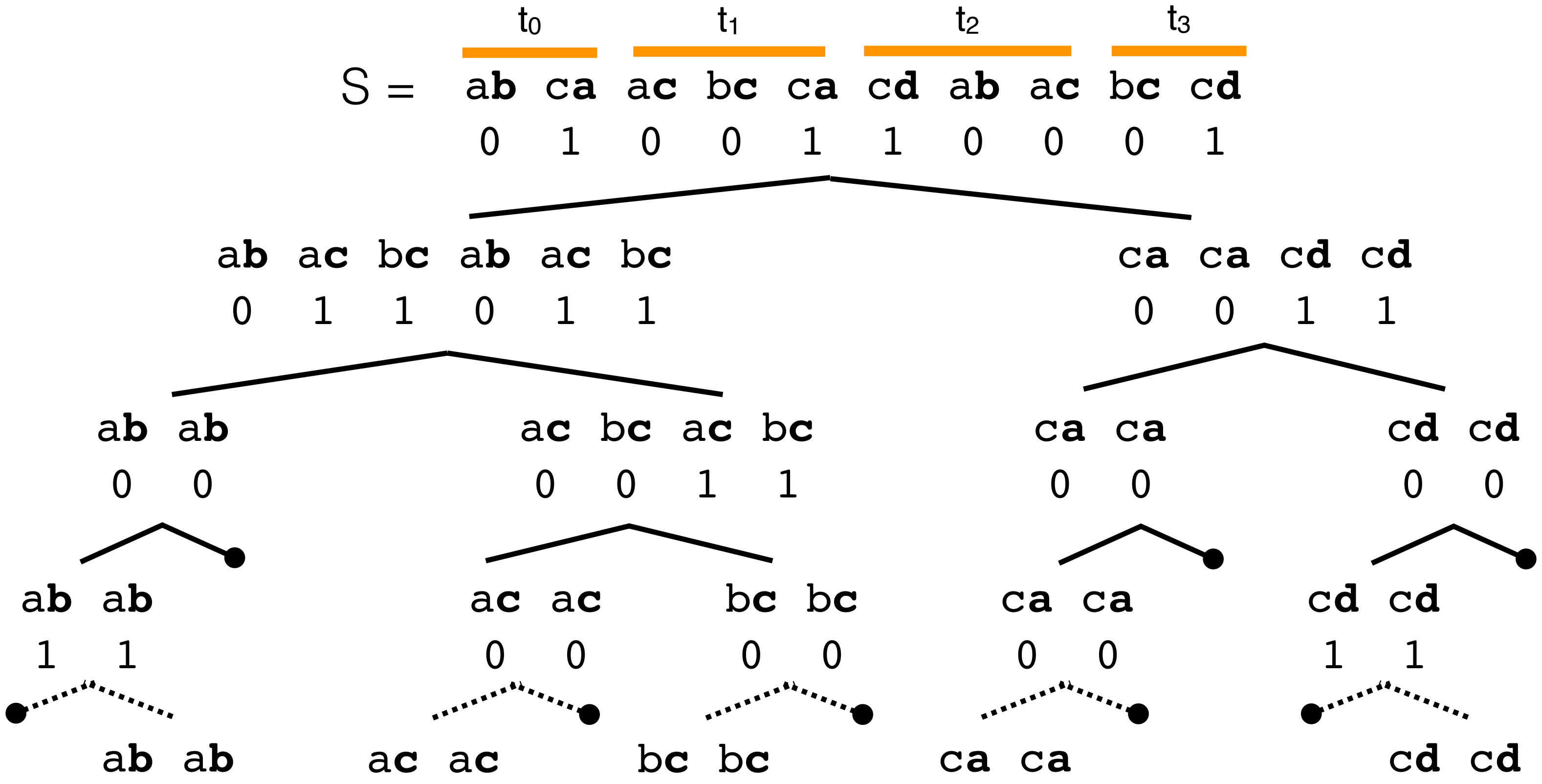
- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
    - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.



Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**

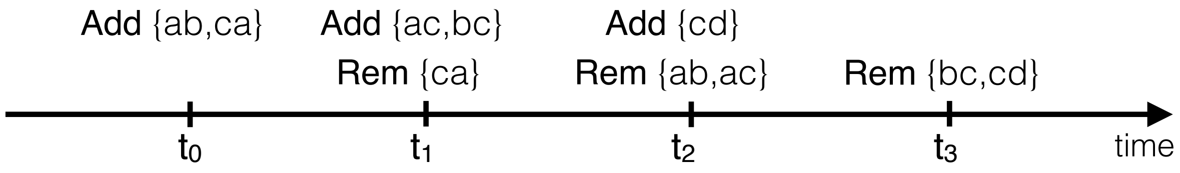
edge(ab,t=2)?



Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

Compact Events ordered by Time (CET)

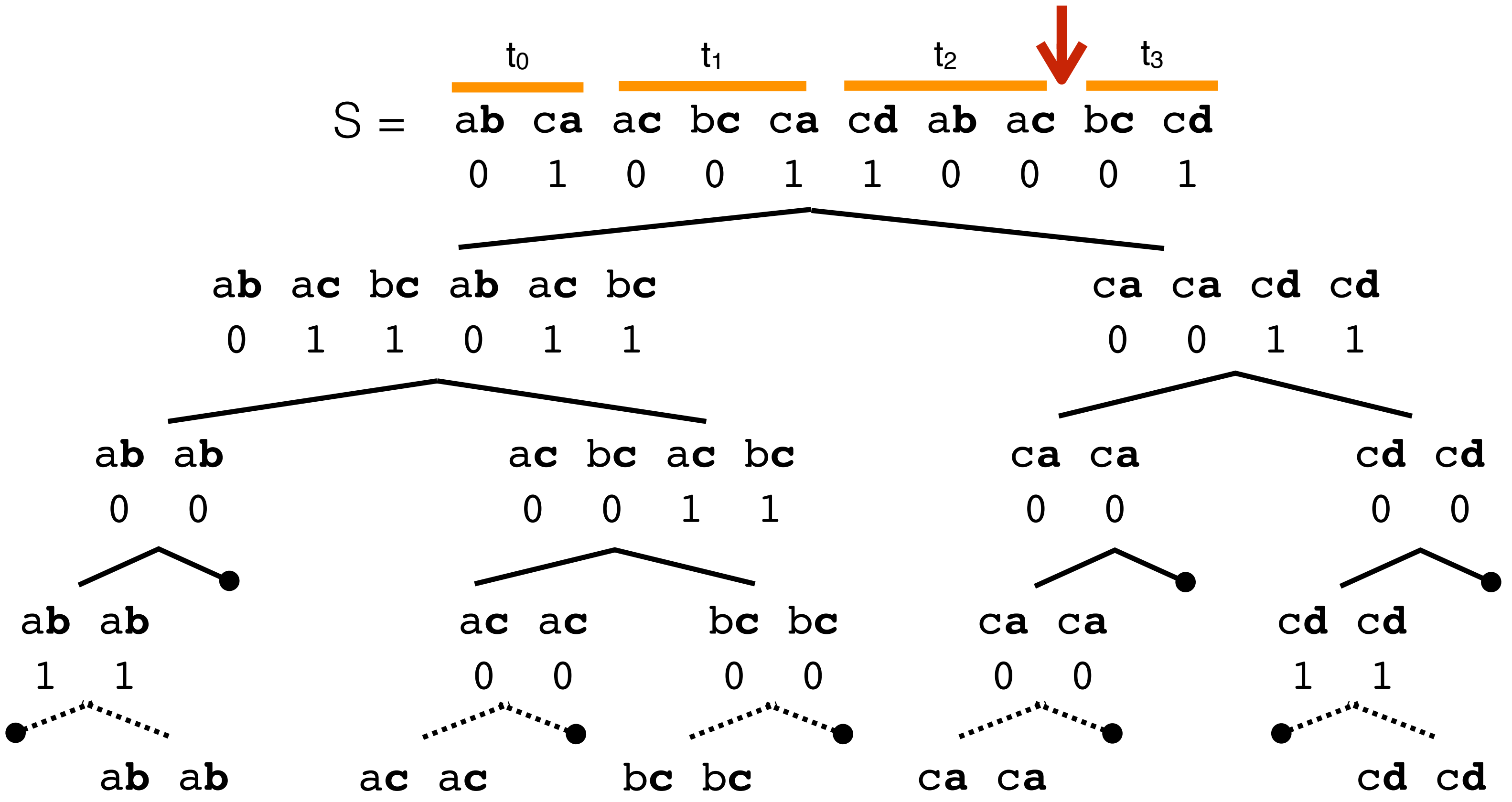
- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.



Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**

edge(ab,t=2)?



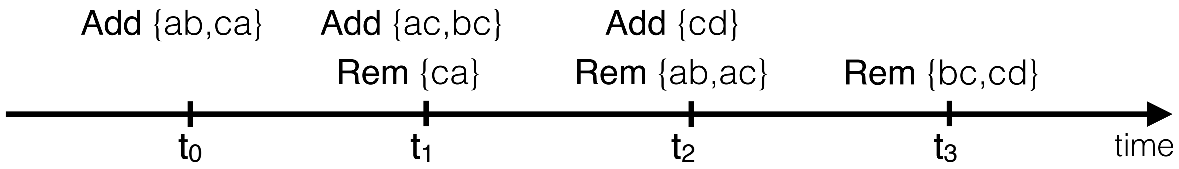
Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

Compact Events ordered by Time (CET)

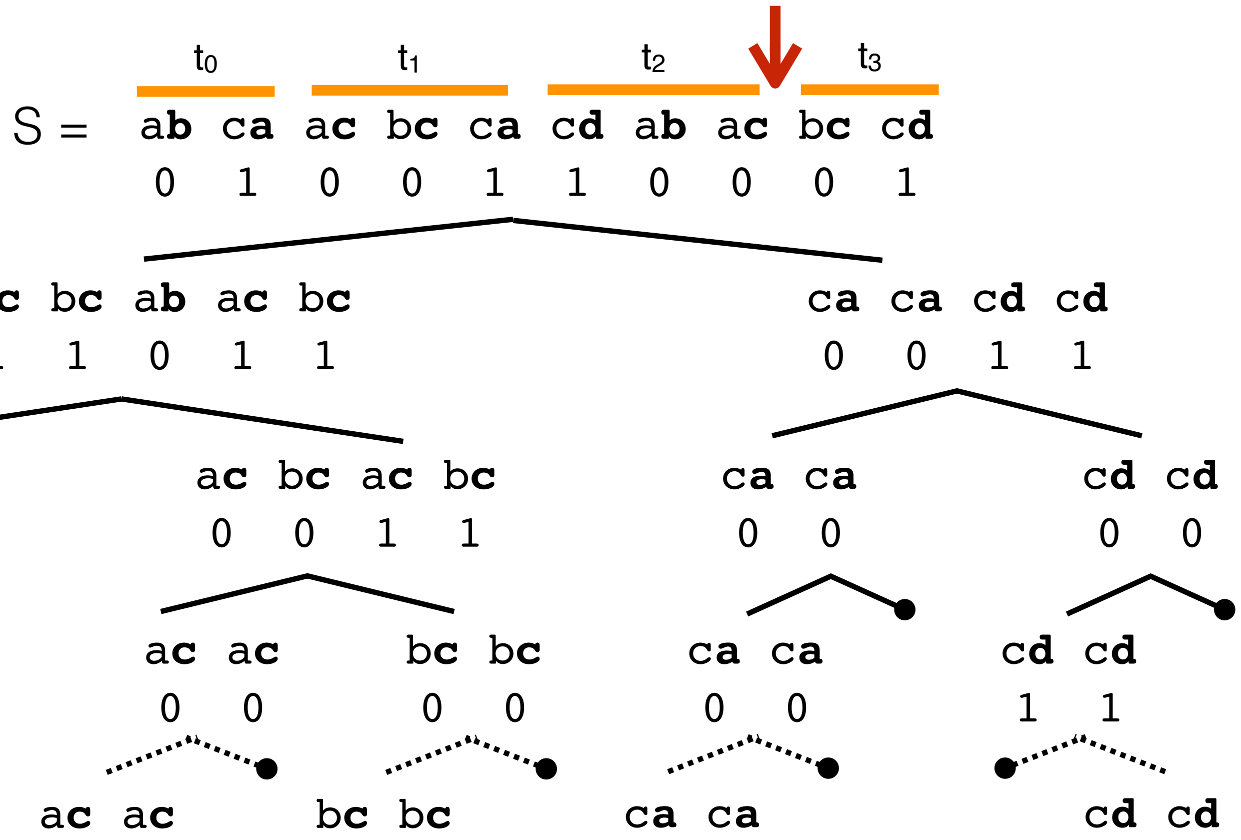
- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
    - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**



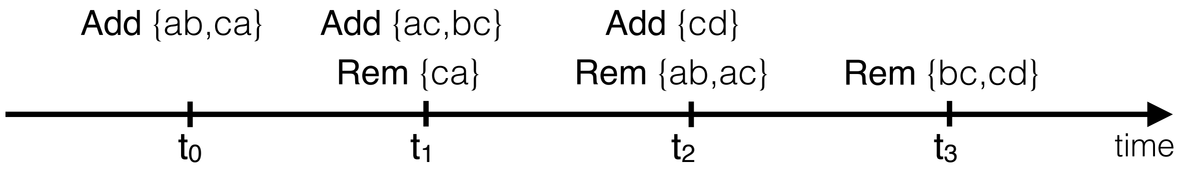
edge(ab,t=2)?



Interleaving Code	
ab	0**0**0**1**
ac	0**1**0**0**
bc	0**1**1**0**
ca	1**0**0**0**
cd	1**1**0**1**

Compact Events ordered by Time (CET)

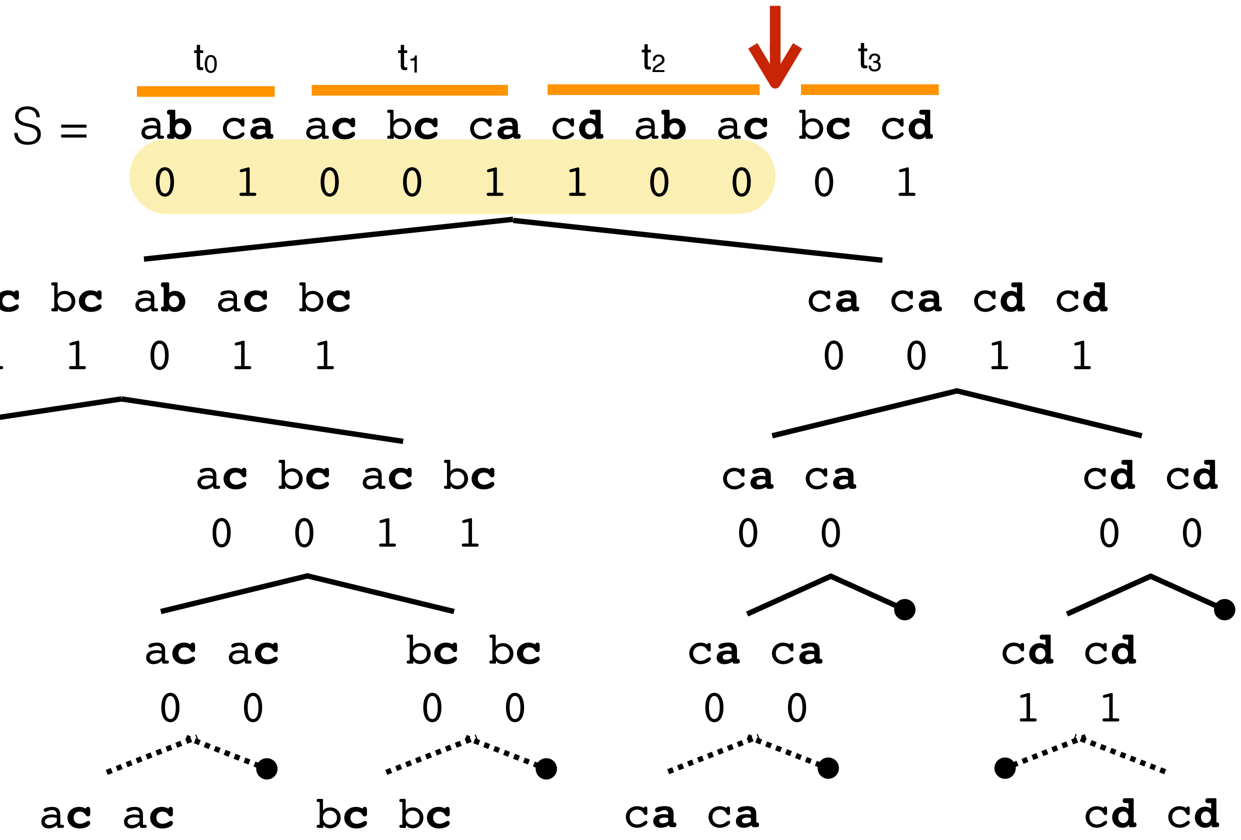
- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.



Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**

edge(ab,t=2)?

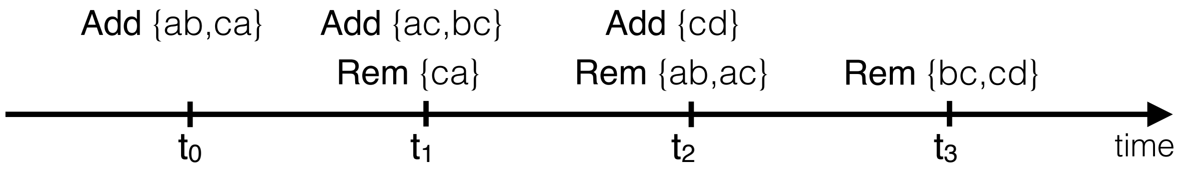


Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

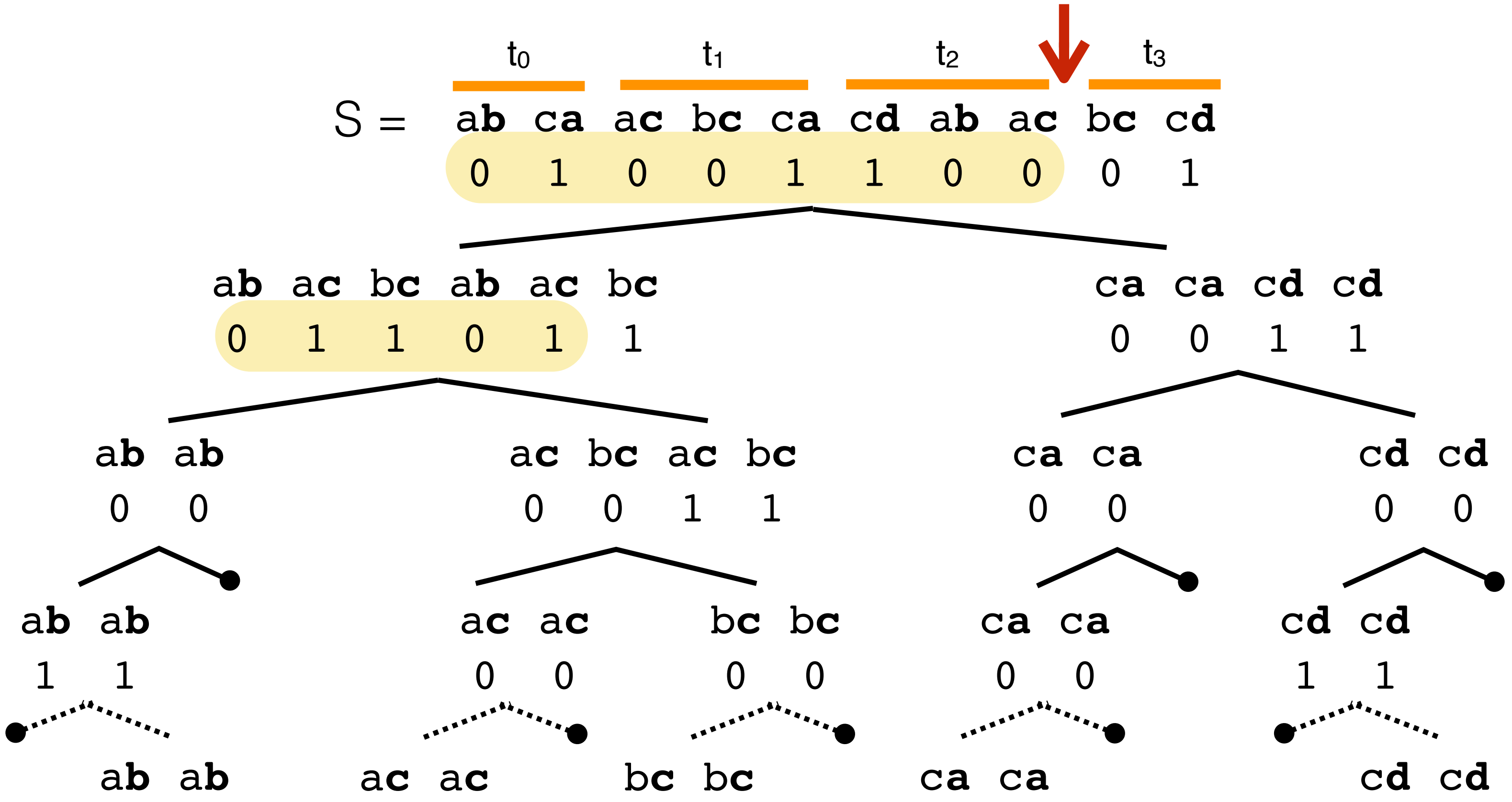
Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
    - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex		Target Vertex	
a	00	a	**00**
b	01	b	**01**
c	10	c	**10**
d	11	d	**11**



edge(ab,t=2)?

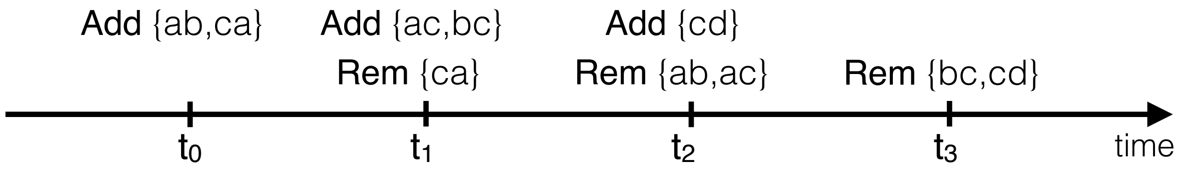


Interleaving Code

ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

Compact Events ordered by Time (CET)

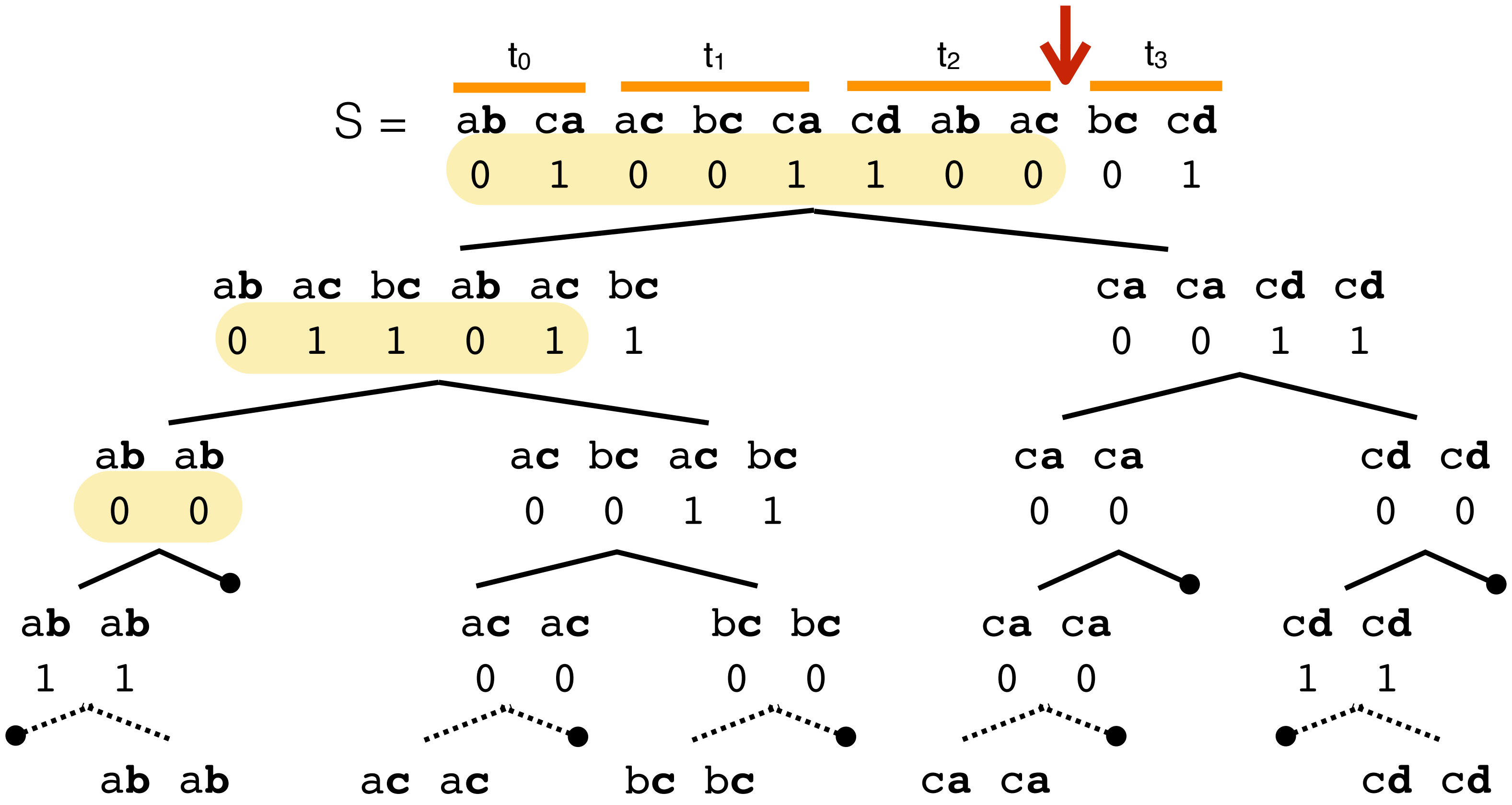
- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
    - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.



Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**

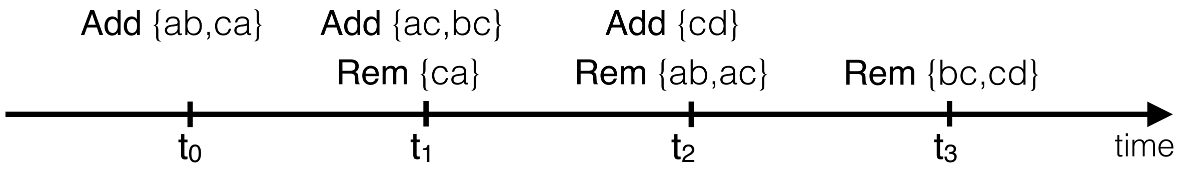
edge(ab,t=2)?



Interleaving Code	
ab	0**0**0**1**
ac	0**1**0**0**
bc	0**1**1**0**
ca	1**0**0**0**
cd	1**1**0**1**

Compact Events ordered by Time (CET)

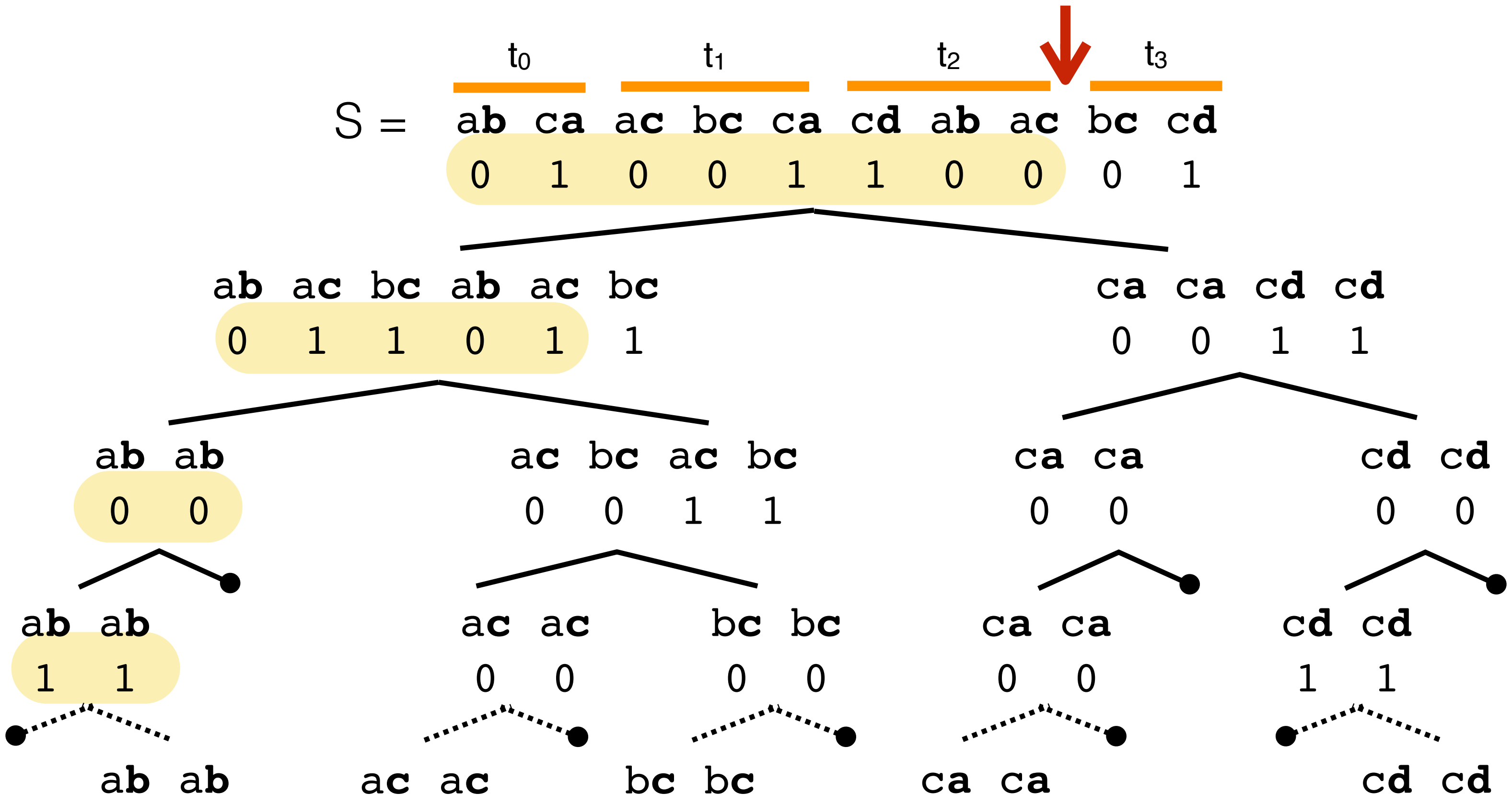
- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.



Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**

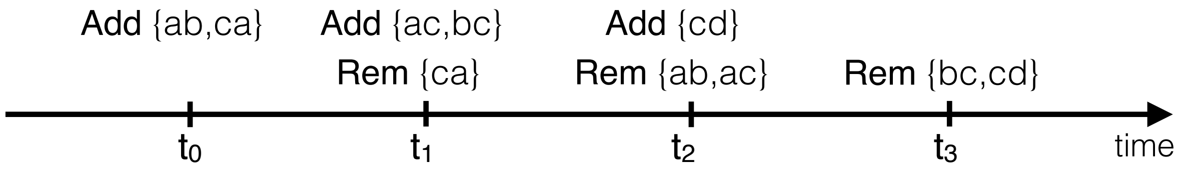
edge(ab,t=2)?



Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

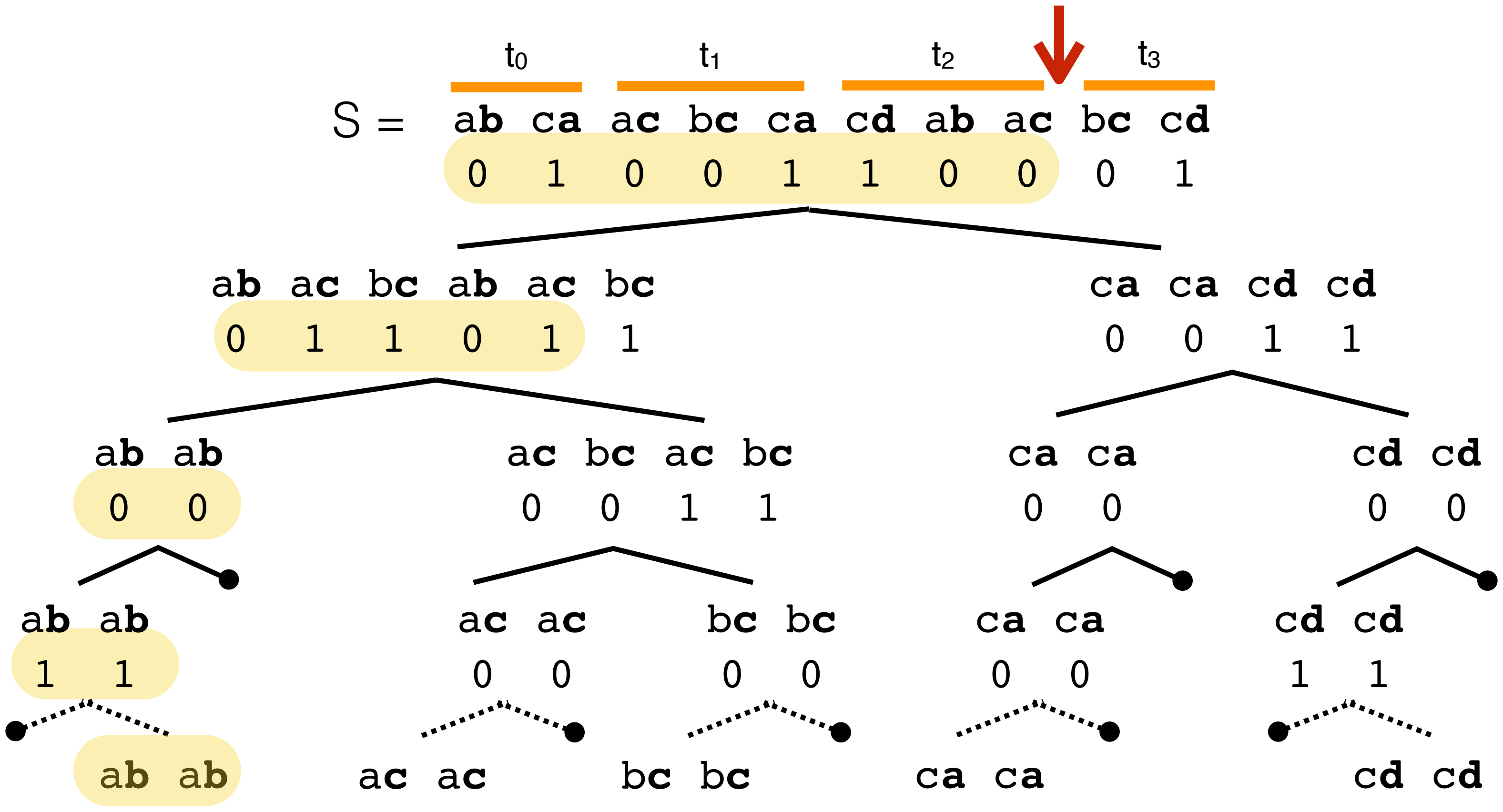
Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.



Source Vertex		**Target Vertex**	
a	00	a	**00**
b	01	b	**01**
c	10	c	**10**
d	11	d	**11**

edge(ab,t=2)?



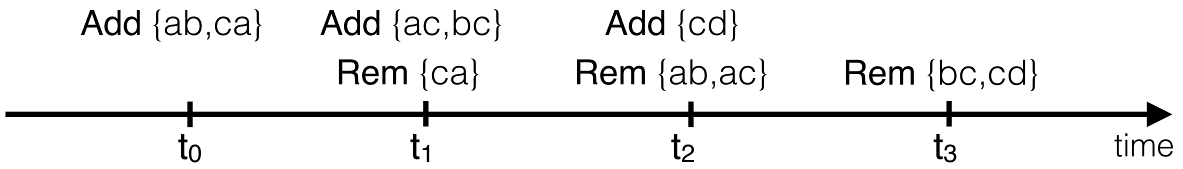
Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

Compact Events ordered by Time (CET)

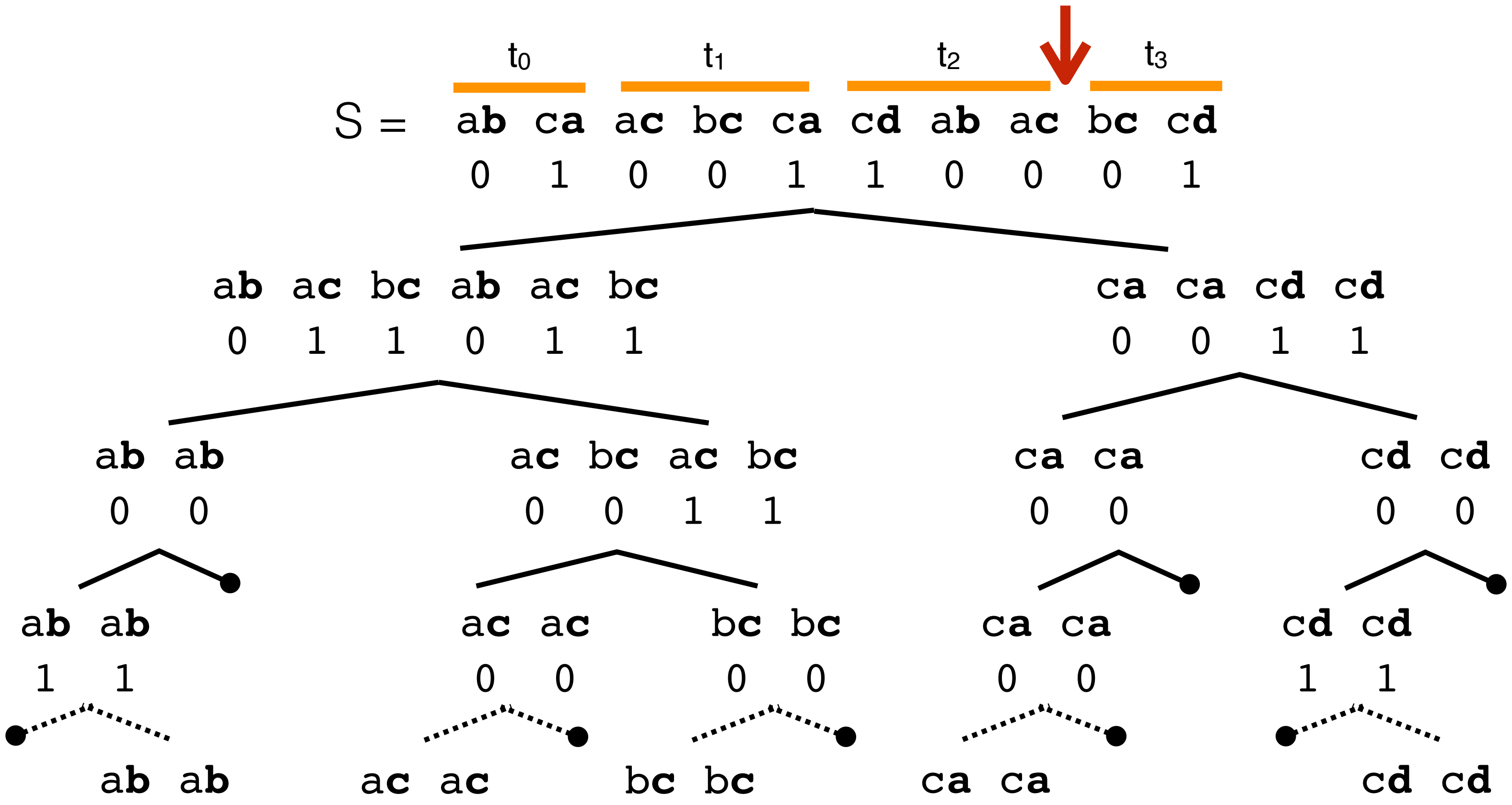
- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**



dirnei(a,t=2)?



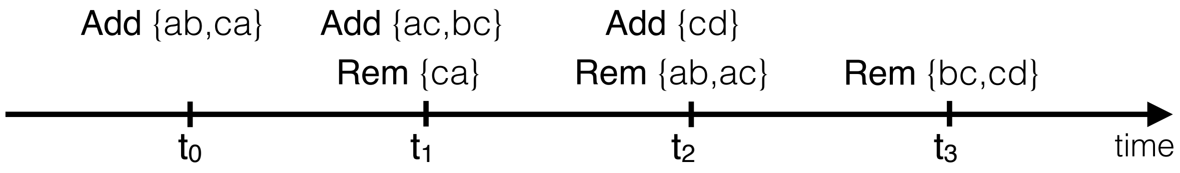
Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

Compact Events ordered by Time (CET)

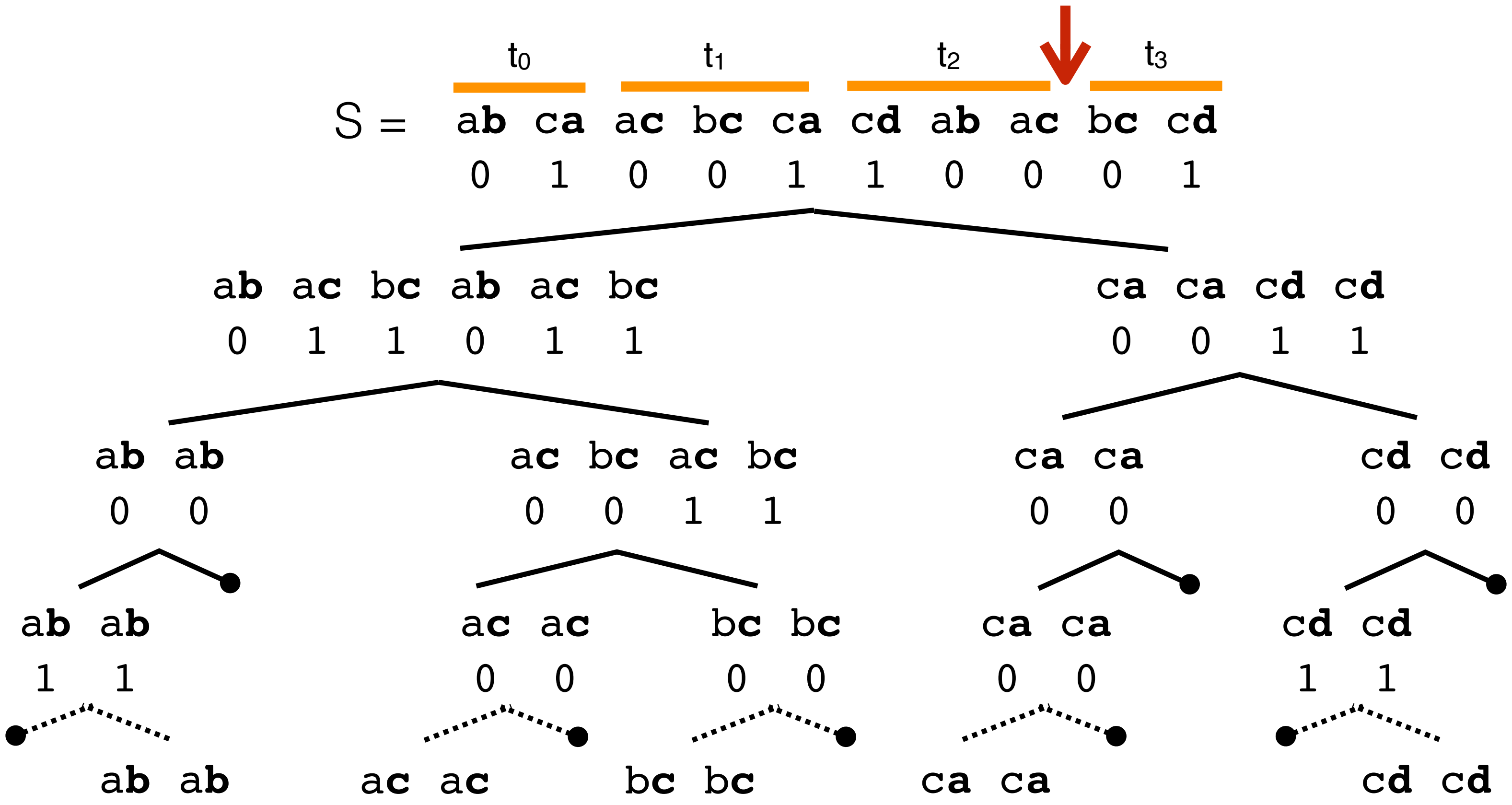
- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
    - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	00
b	01
c	10
d	11



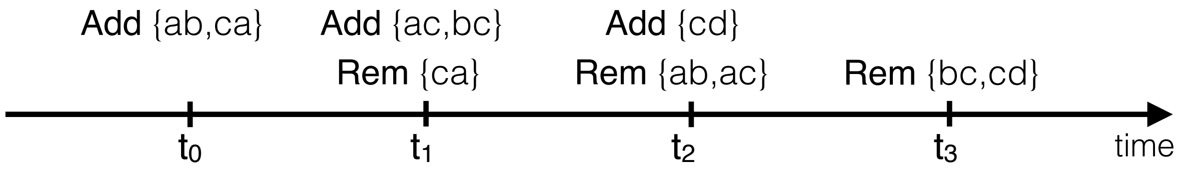
dirnei(a,t=2)?



Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

Compact Events ordered by Time (CET)

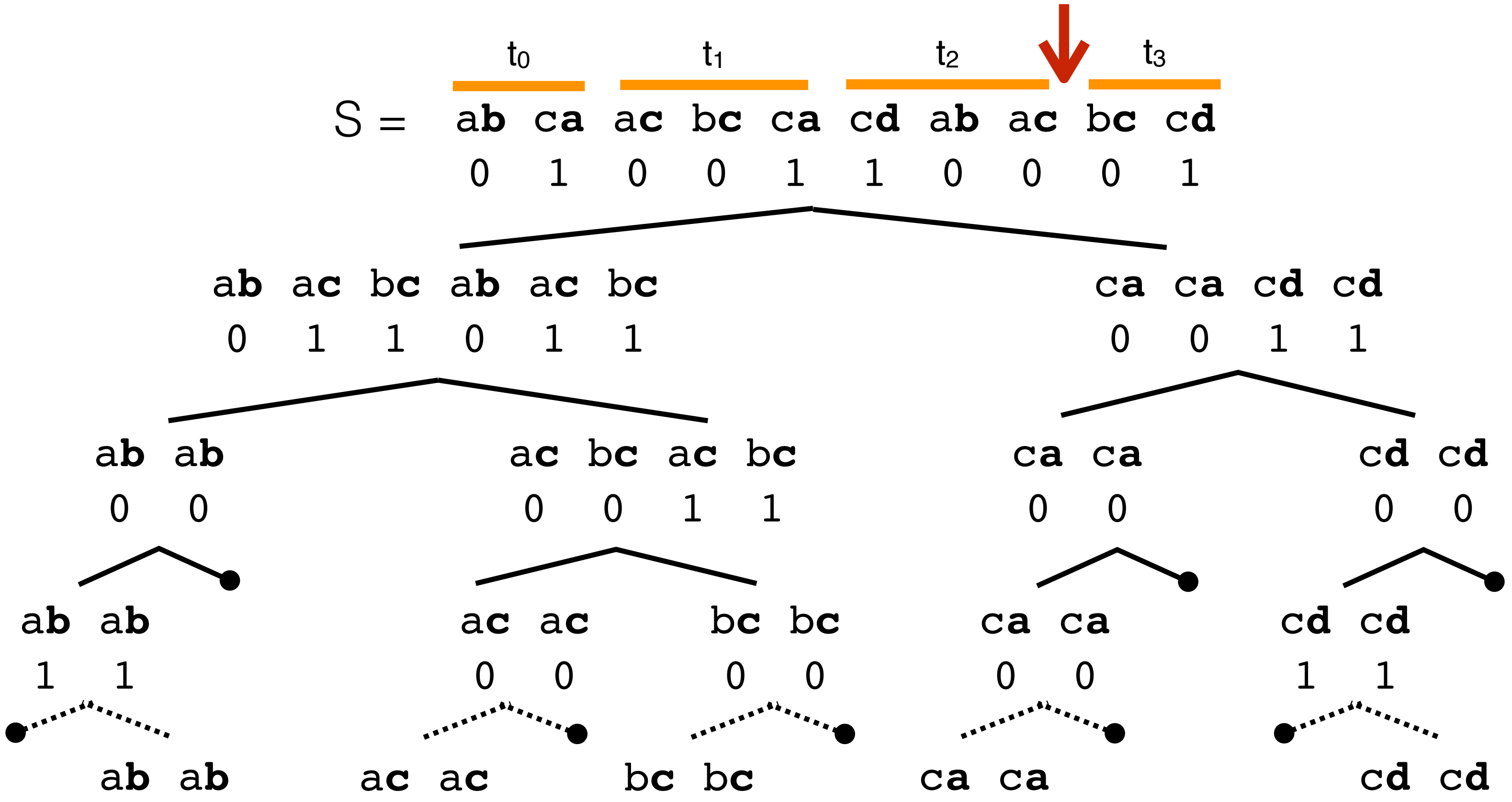
- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.



Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**

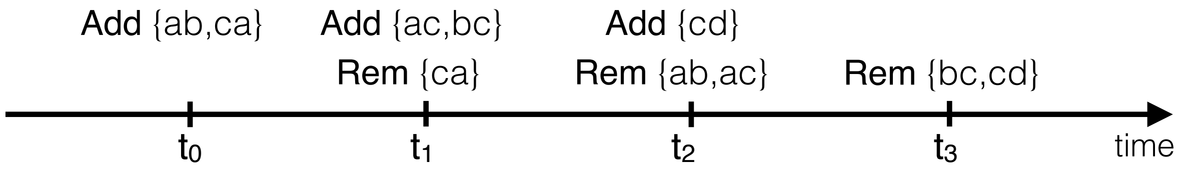
dirnei(a,t=2)?



Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

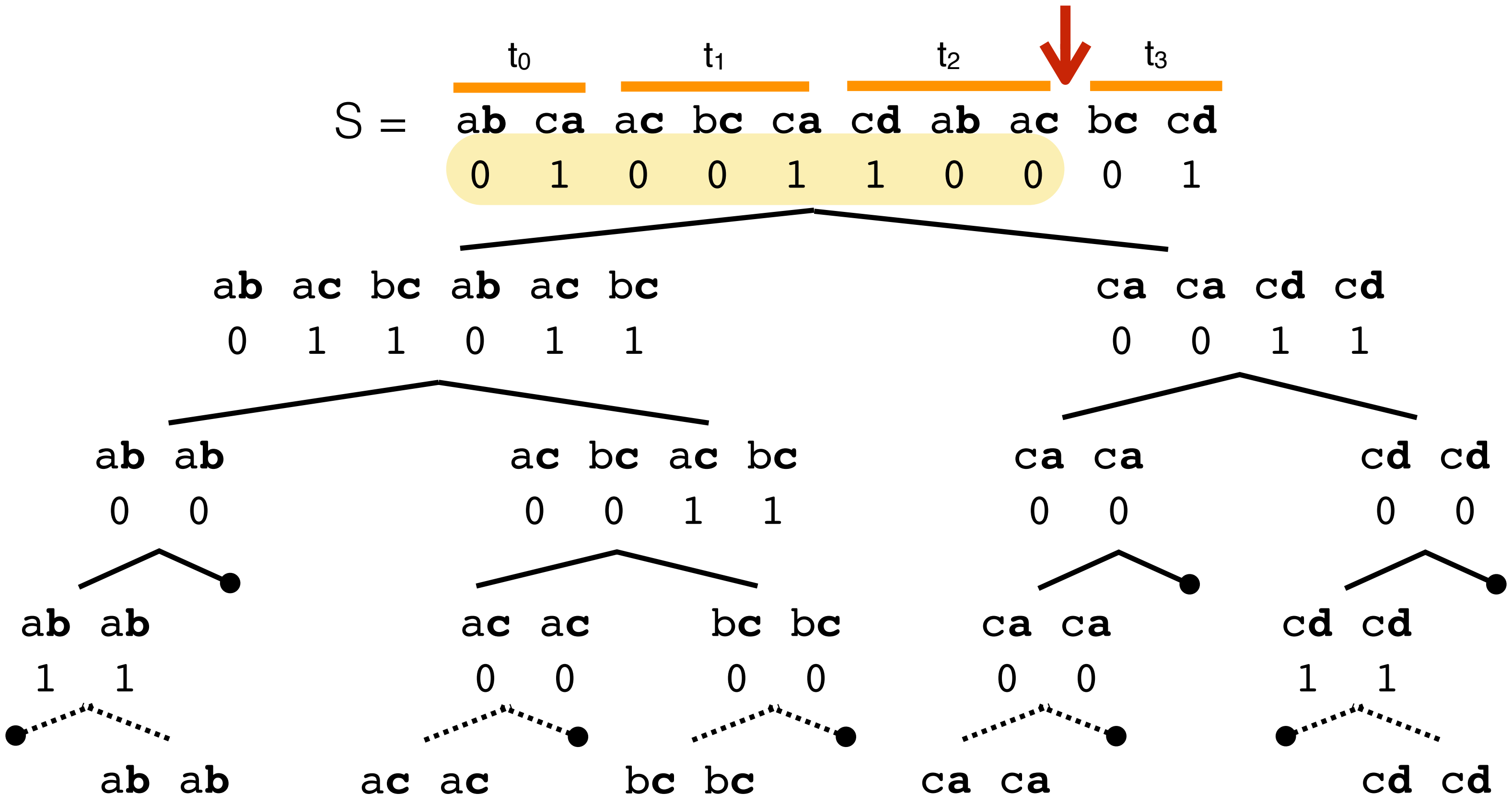
Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.



Source Vertex		**Target Vertex**	
a	00	a	**00**
b	01	b	**01**
c	10	c	**10**
d	11	d	**11**

dirnei(a,t=2)?



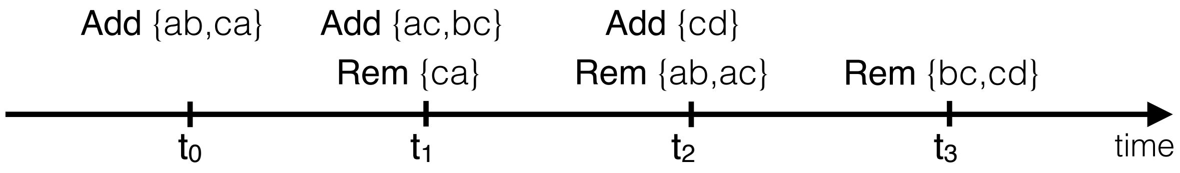
Interleaving Code	
ab	0**0**0**1**
ac	0**1**0**0**
bc	0**1**1**0**
ca	1**0**0**0**
cd	1**1**0**1**

Compact Events ordered by Time (CET)

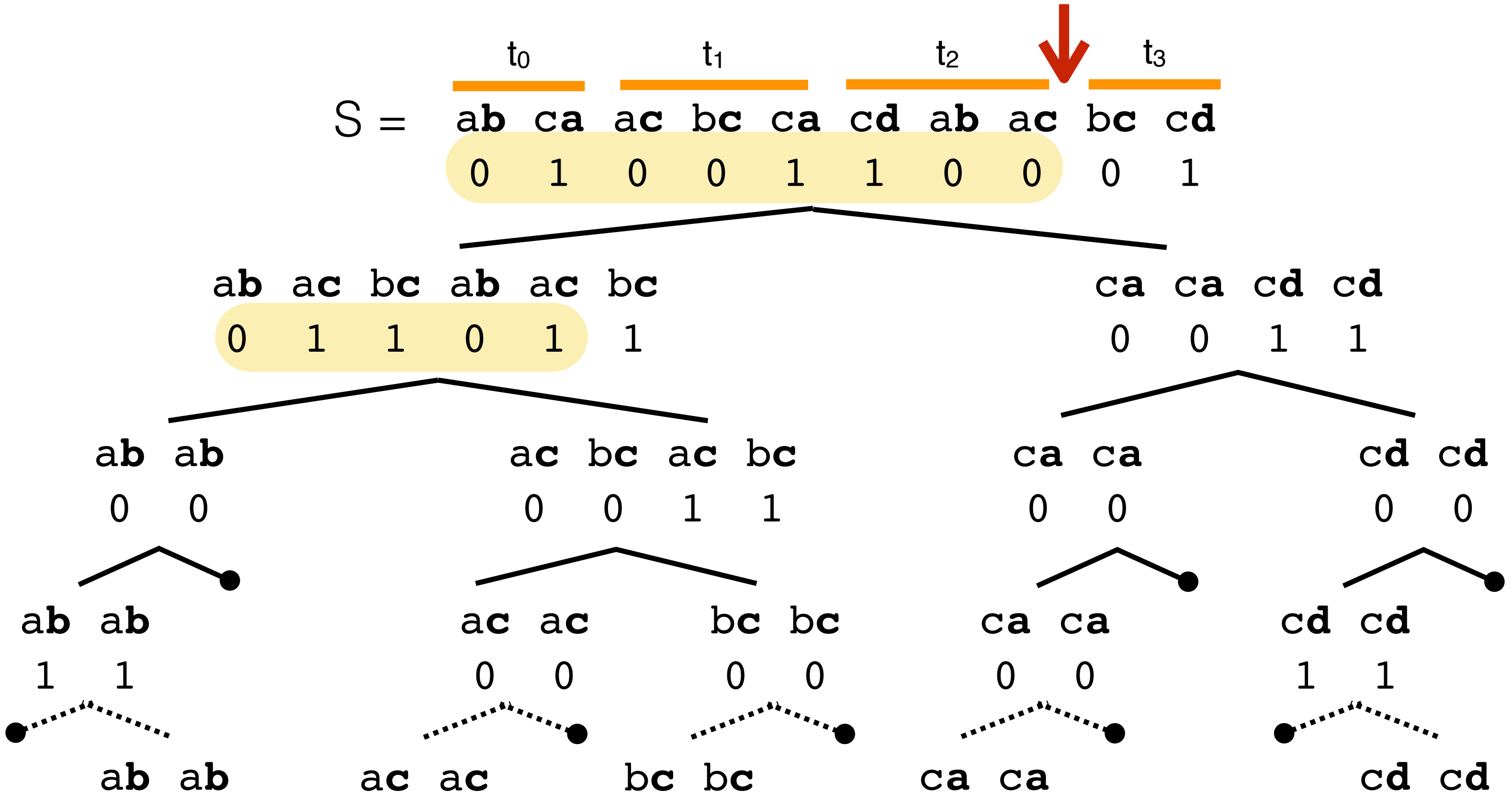
- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
    - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**



dirnei(a,t=2)?



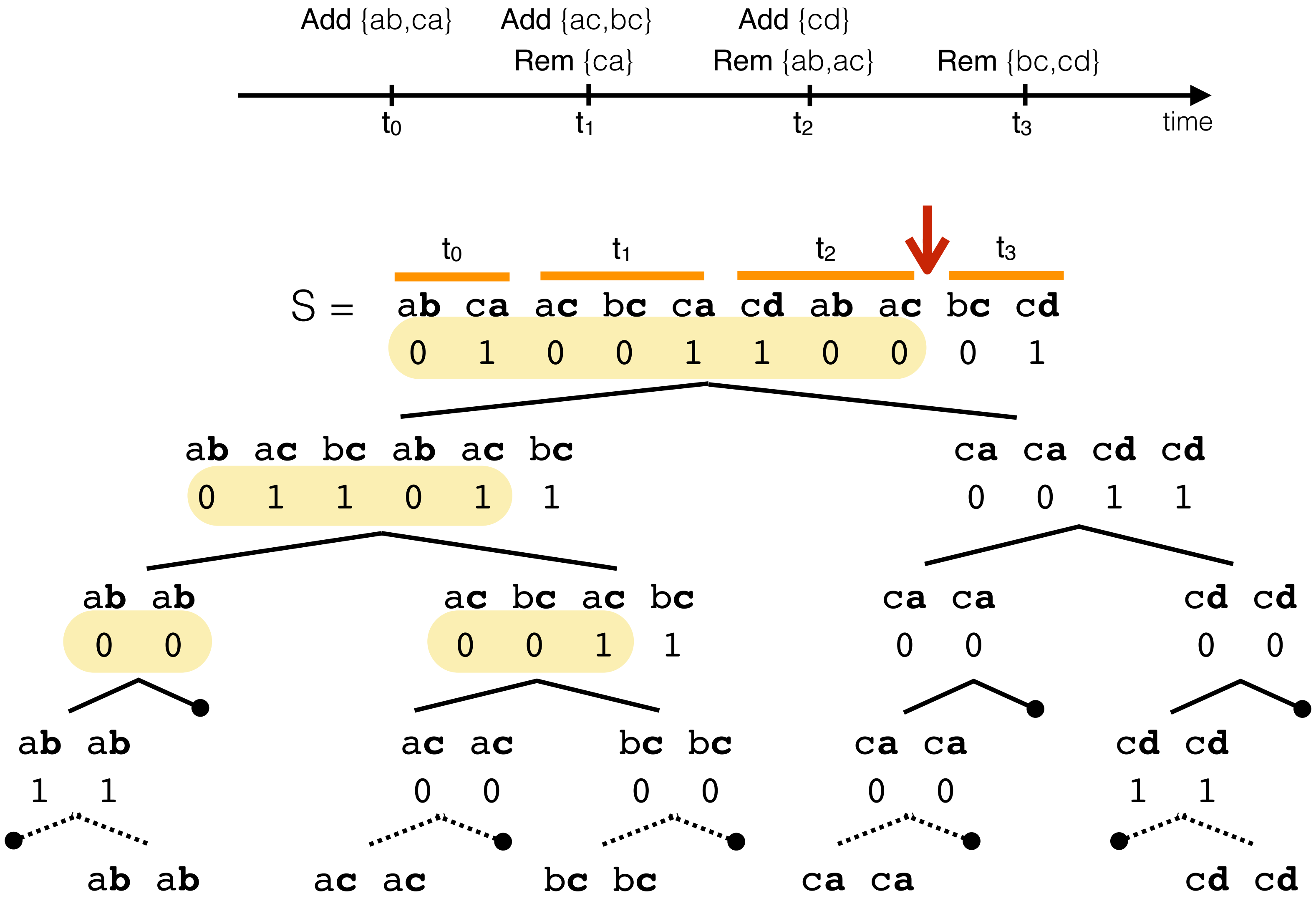
Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
    - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex		Target Vertex	
a	00	a	00
b	01	b	01
c	10	c	10
d	11	d	11

dirnei(a,t=2)?

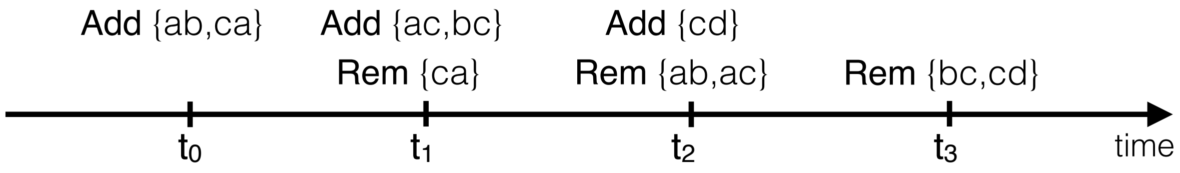


Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
    - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex		Target Vertex	
a	00	a	**00**
b	01	b	**01**
c	10	c	**10**
d	11	d	**11**

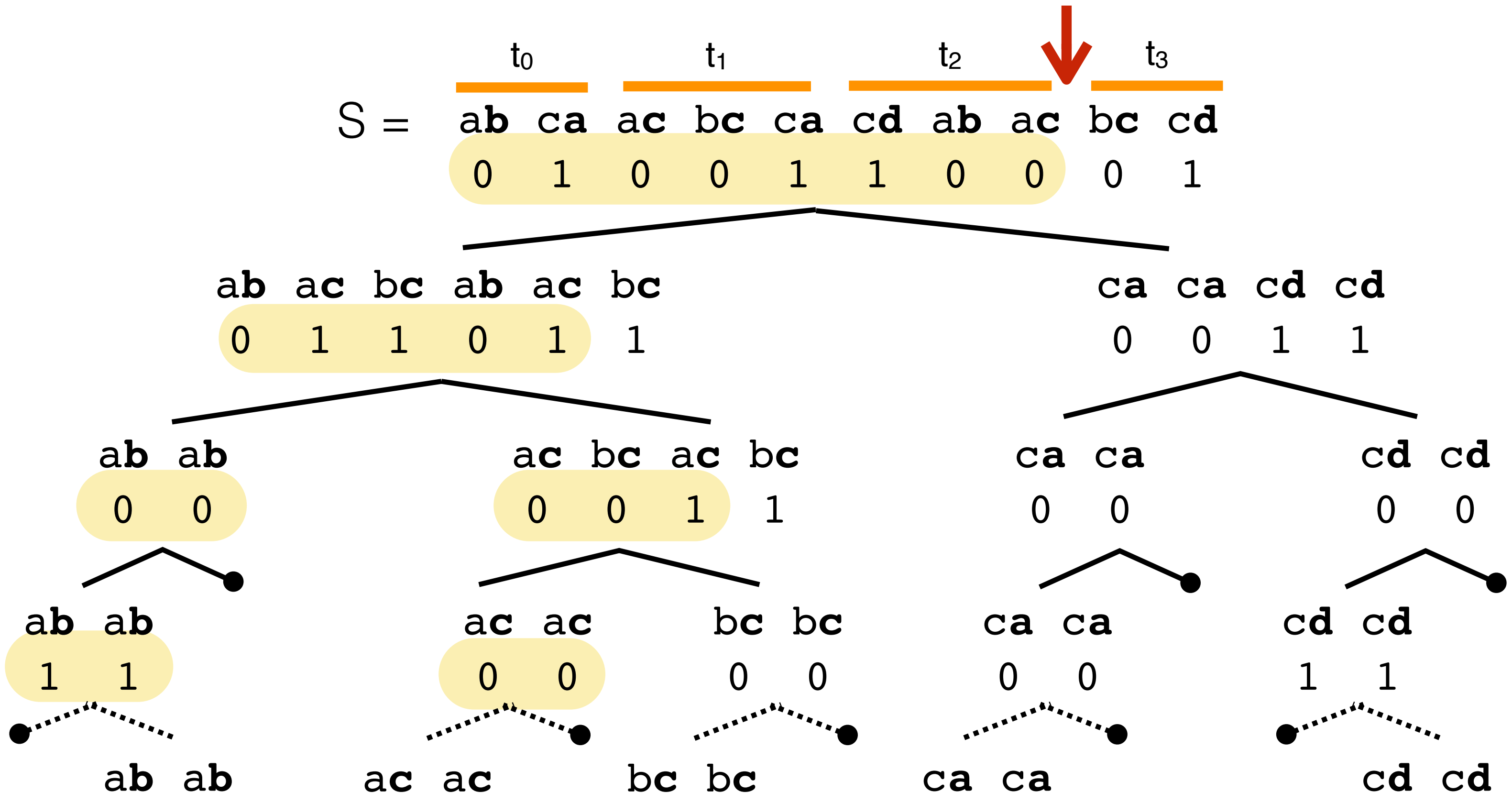

Add {ab,ca}

Add {ac,bc}
Rem {ca}

Add {cd}
Rem {ab,ac}

Rem {bc,cd}

dirnei(a,t=2)?

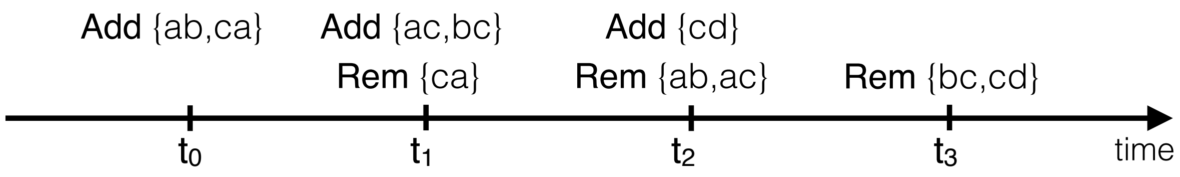


Interleaving Code

ab	0**0**0**1**
ac	0**1**0**0**
bc	0**1**1**0**
ca	1**0**0**0**
cd	1**1**0**1**

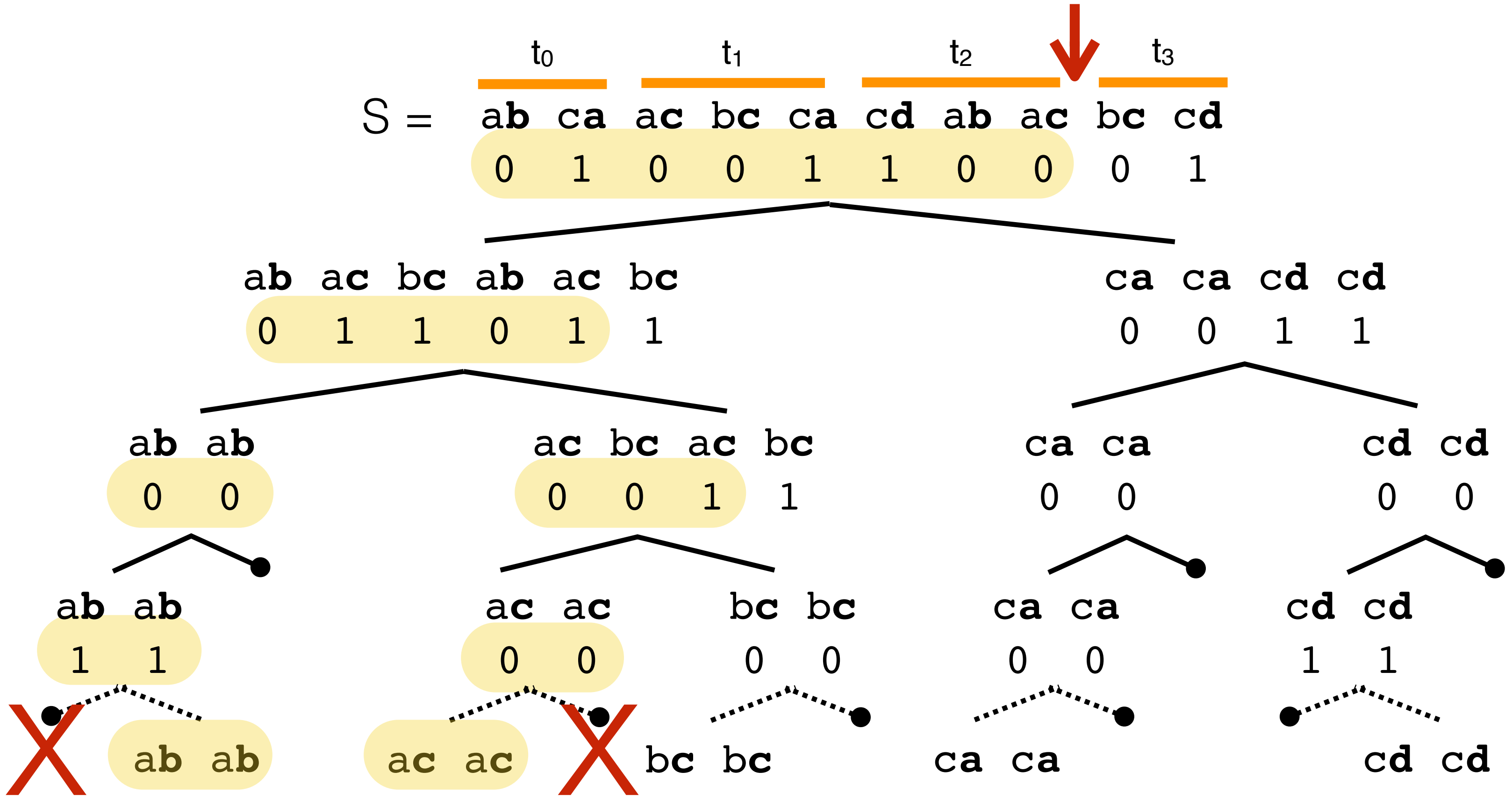
Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
    - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.



Source Vertex		Target Vertex	
a	00	a	**00**
b	01	b	**01**
c	10	c	**10**
d	11	d	**11**

dirnei(a,t=2)?



Interleaving Code

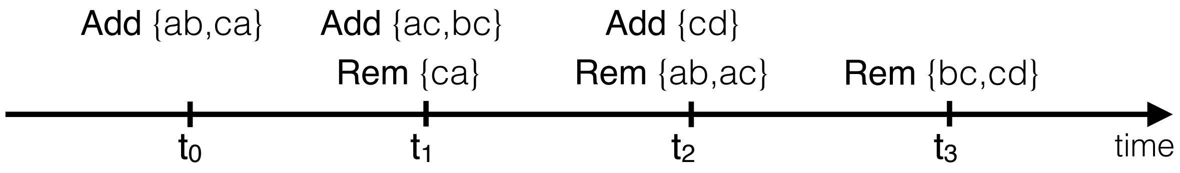
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

Compact Events ordered by Time (CET)

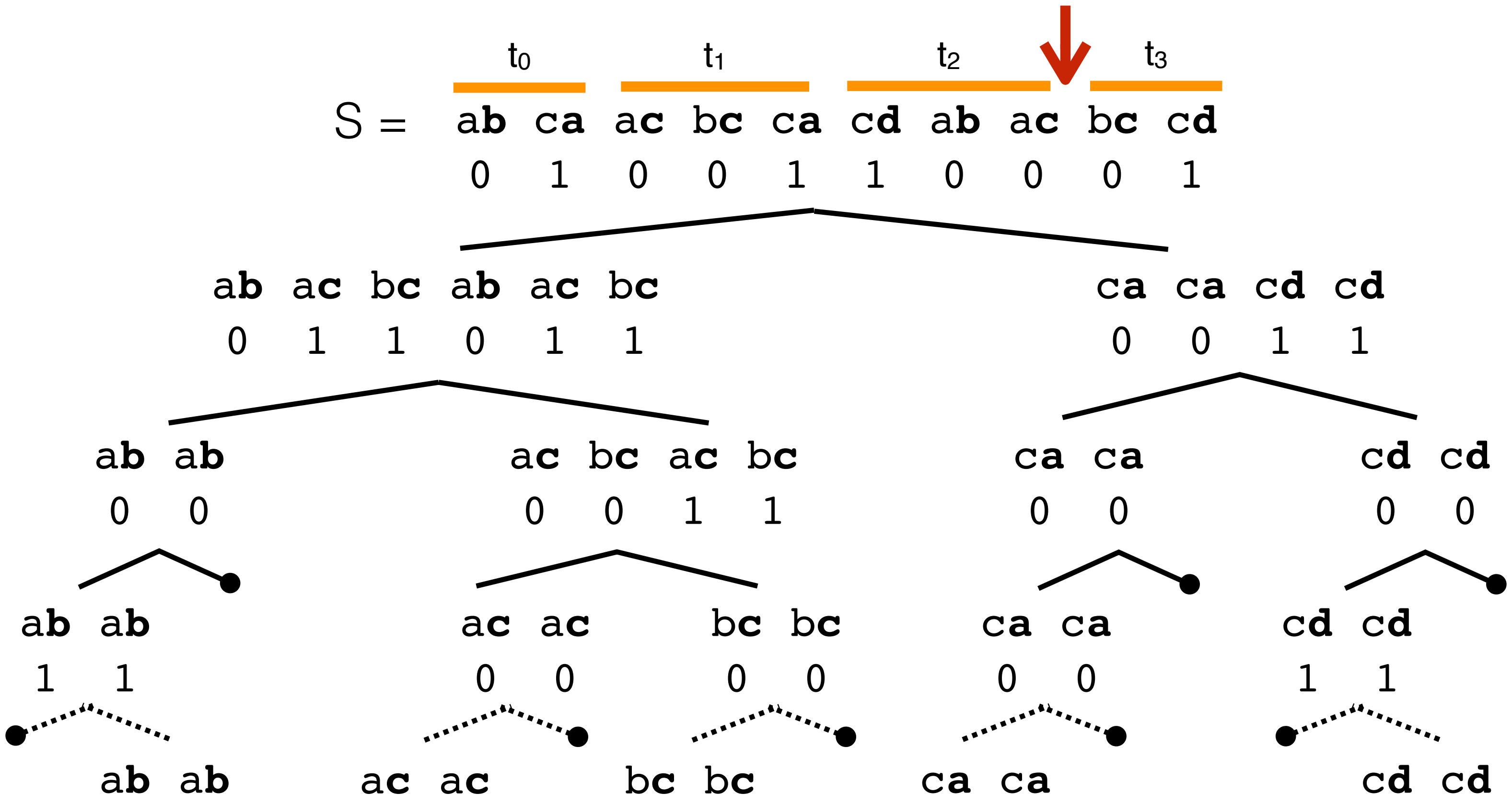
- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**



revnei(c,t=2)?



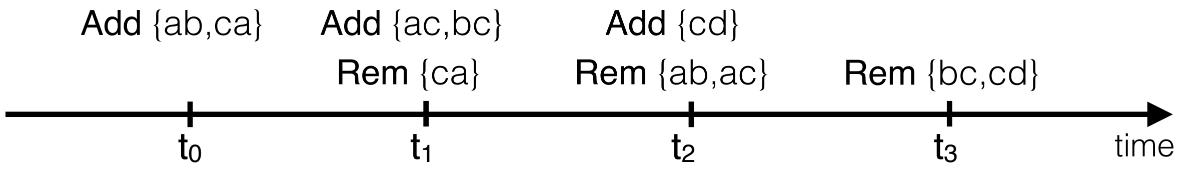
Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

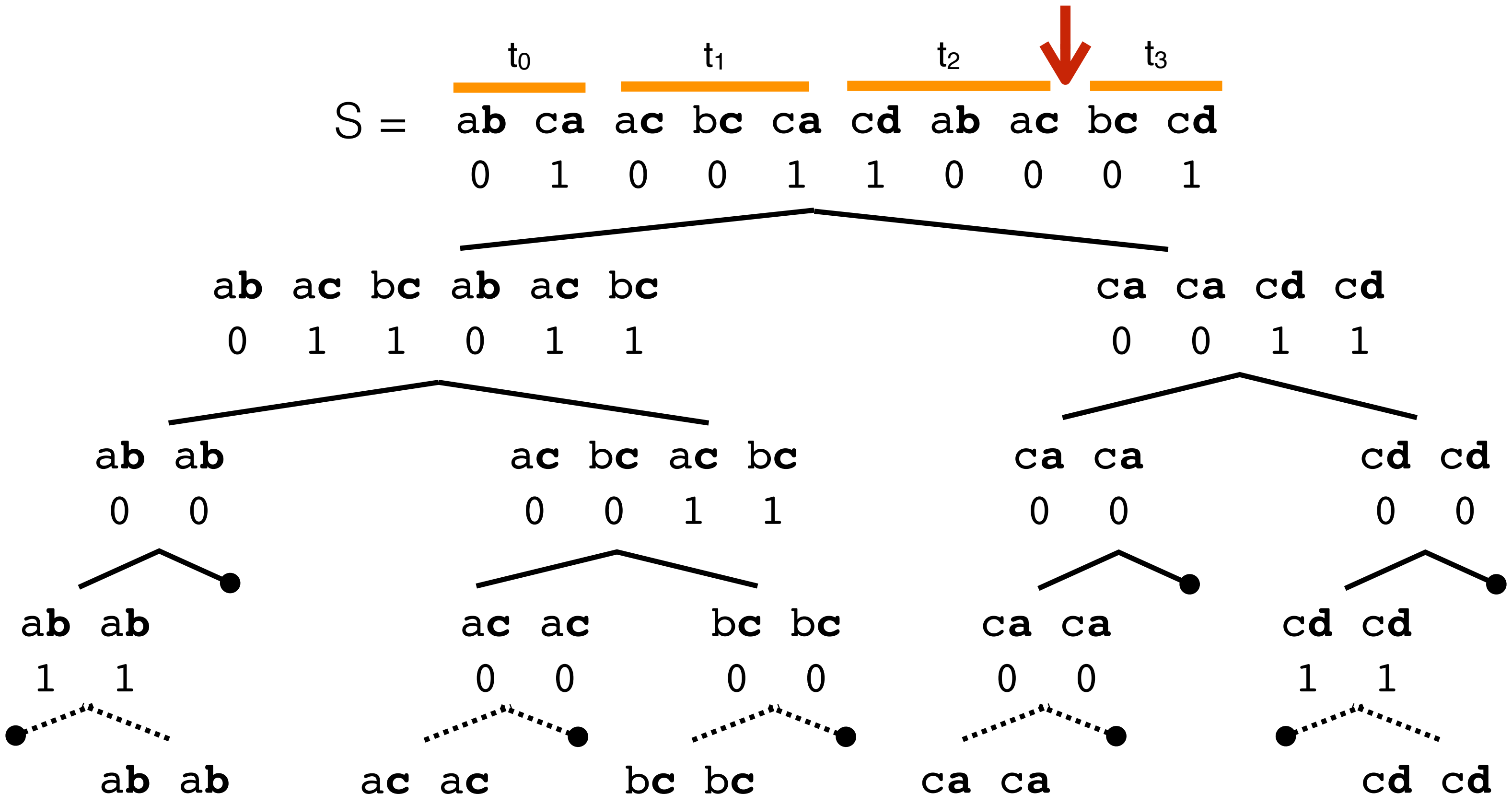
Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**



Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

revnei(c,t=2)?

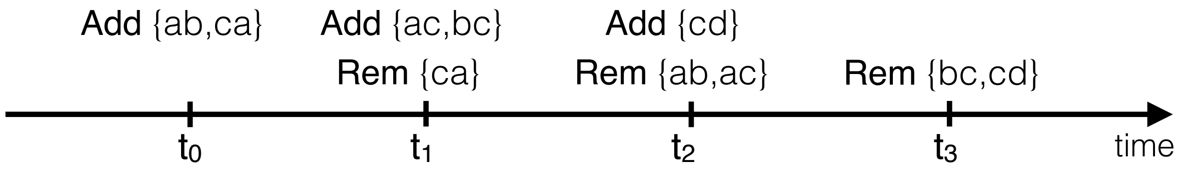


Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
    - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex	
a	00
b	01
c	10
d	11

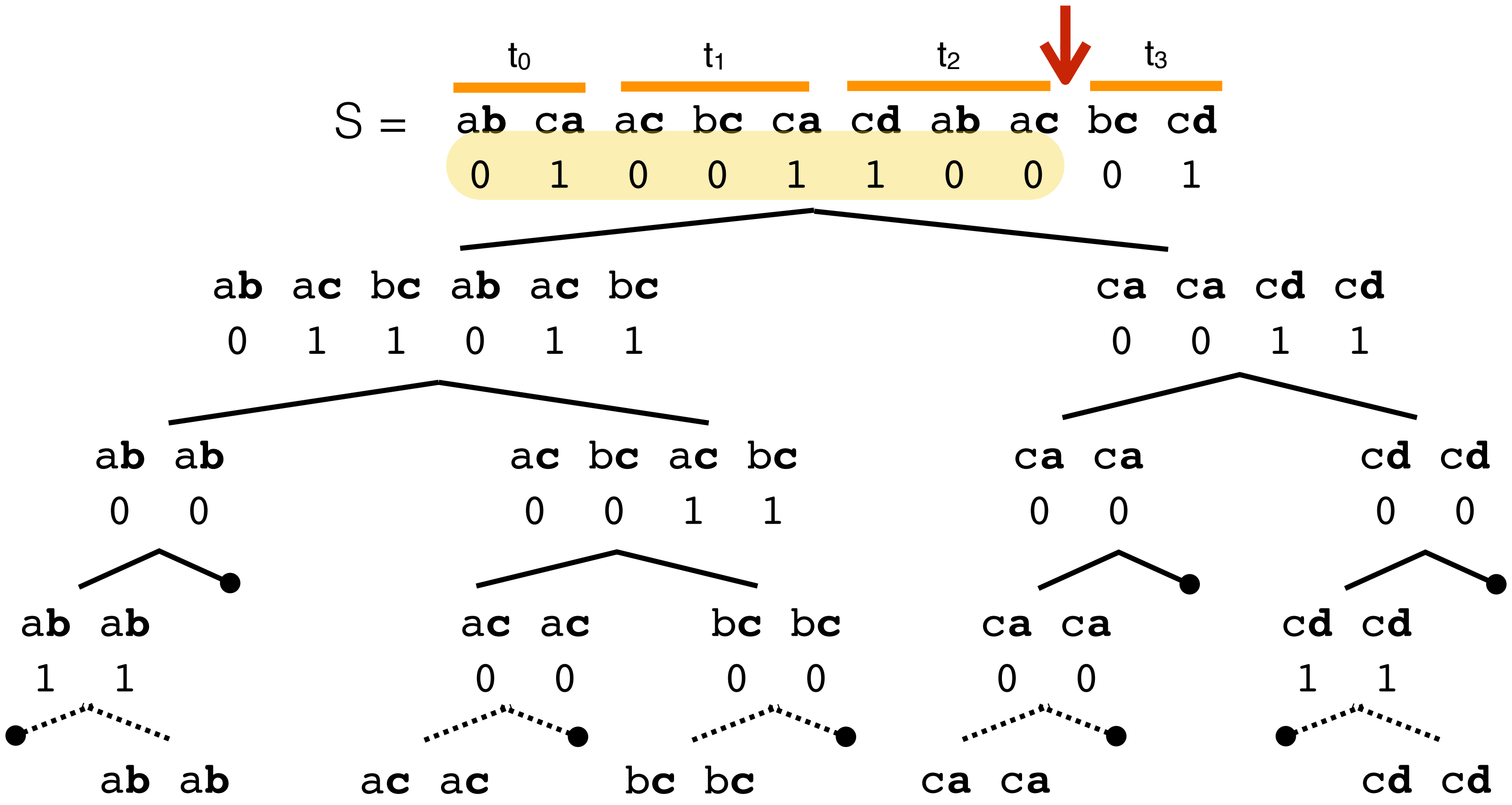
Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**



Interleaving Code

ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

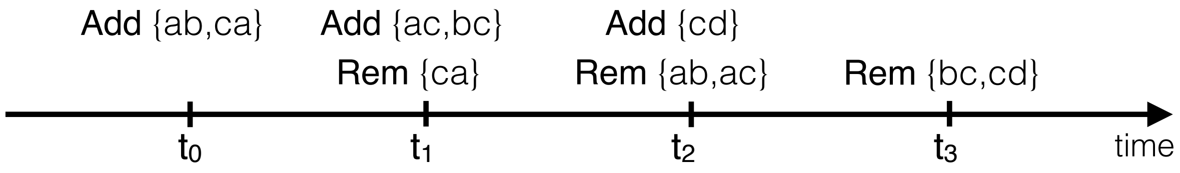
revnei(c,t=2)?



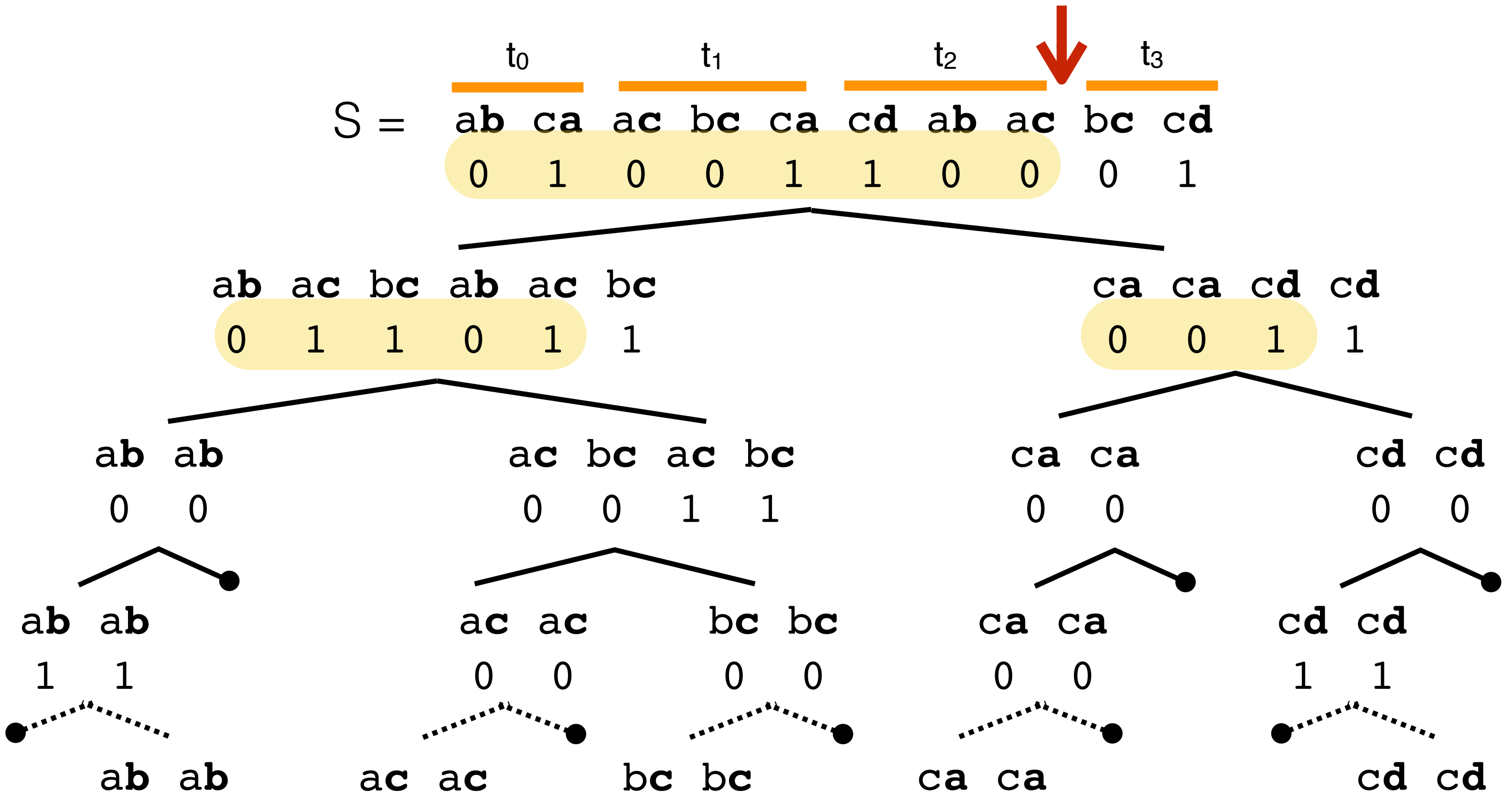
Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex		Target Vertex	
a	00	a	**00**
b	01	b	**01**
c	10	c	**10**
d	11	d	**11**



revnei(c,t=2)?



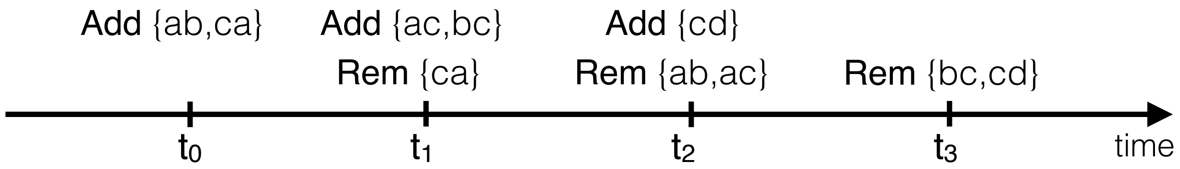
Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
    - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex	
a	00
b	01
c	10
d	11

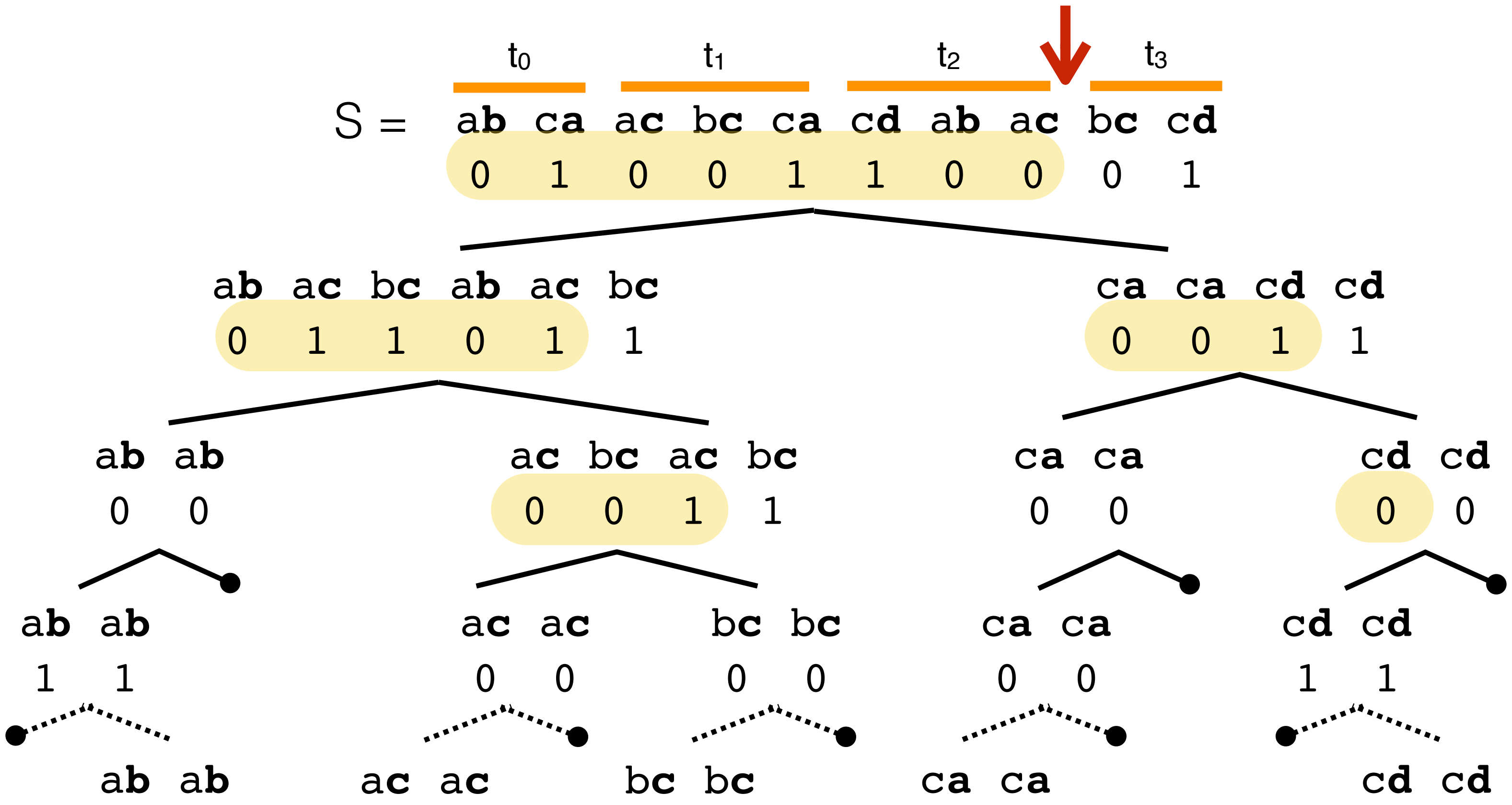
Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**



Interleaving Code

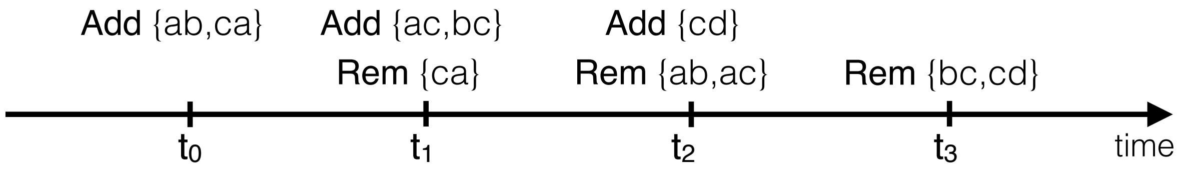
ab	00**01**
ac	01**00**
bc	01**10**
ca	10**00**
cd	11**01**

revnei(c,t=2)?



Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

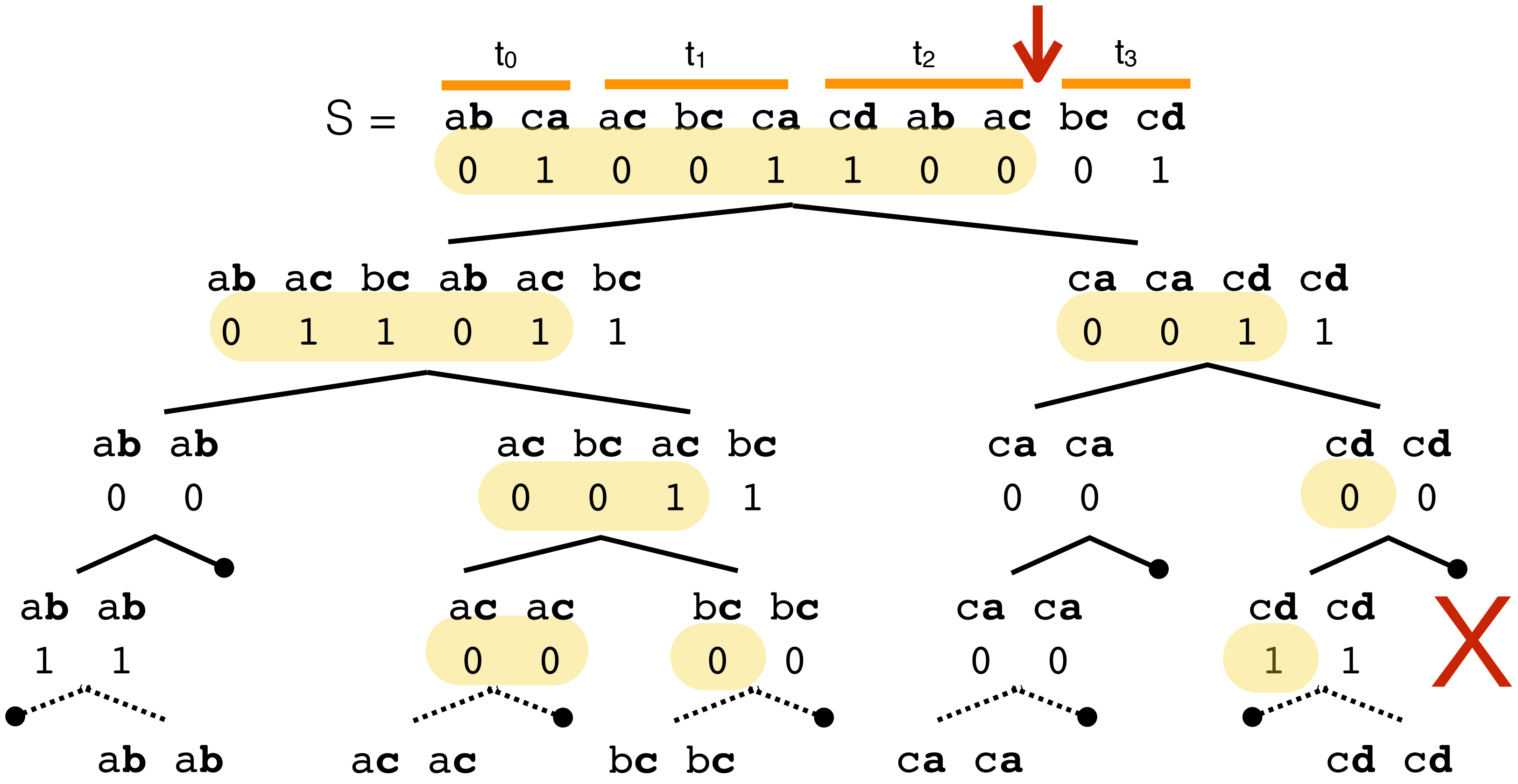


Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**

Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

revnei(c,t=2)?

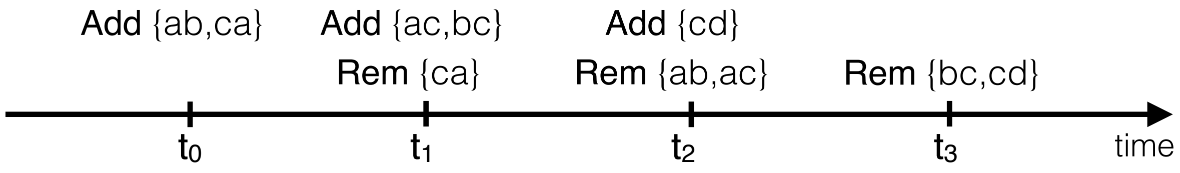


Compact Events ordered by Time (CET)

- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex	
a	00
b	01
c	10
d	11

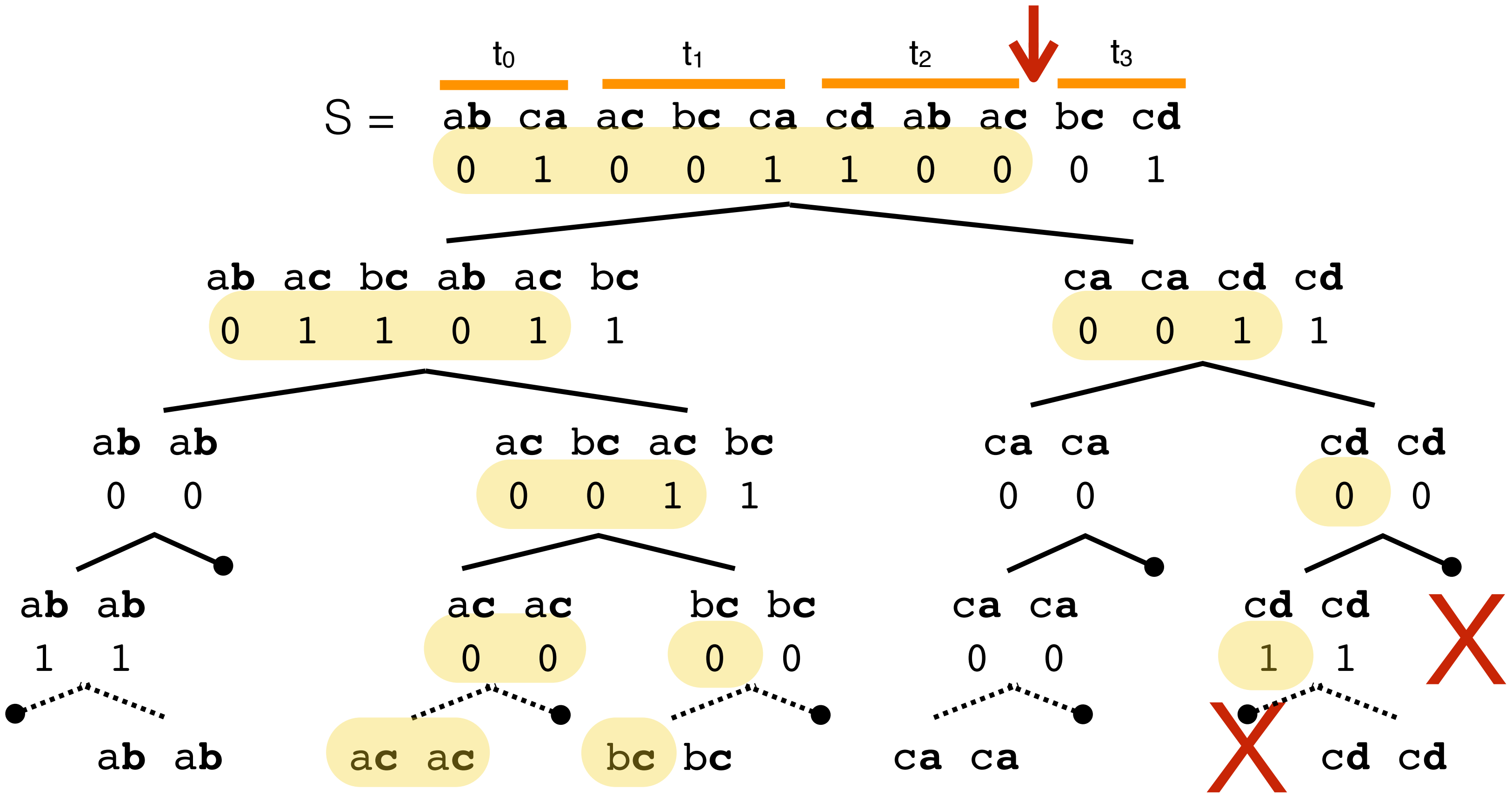
Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**



Interleaving Code

ab	0001
ac	0100
bc	0110
ca	1000
cd	1101

revnei(c,t=2)?

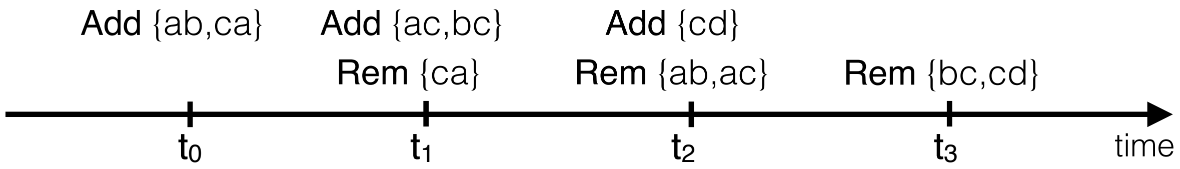


Compact Events ordered by Time (CET)

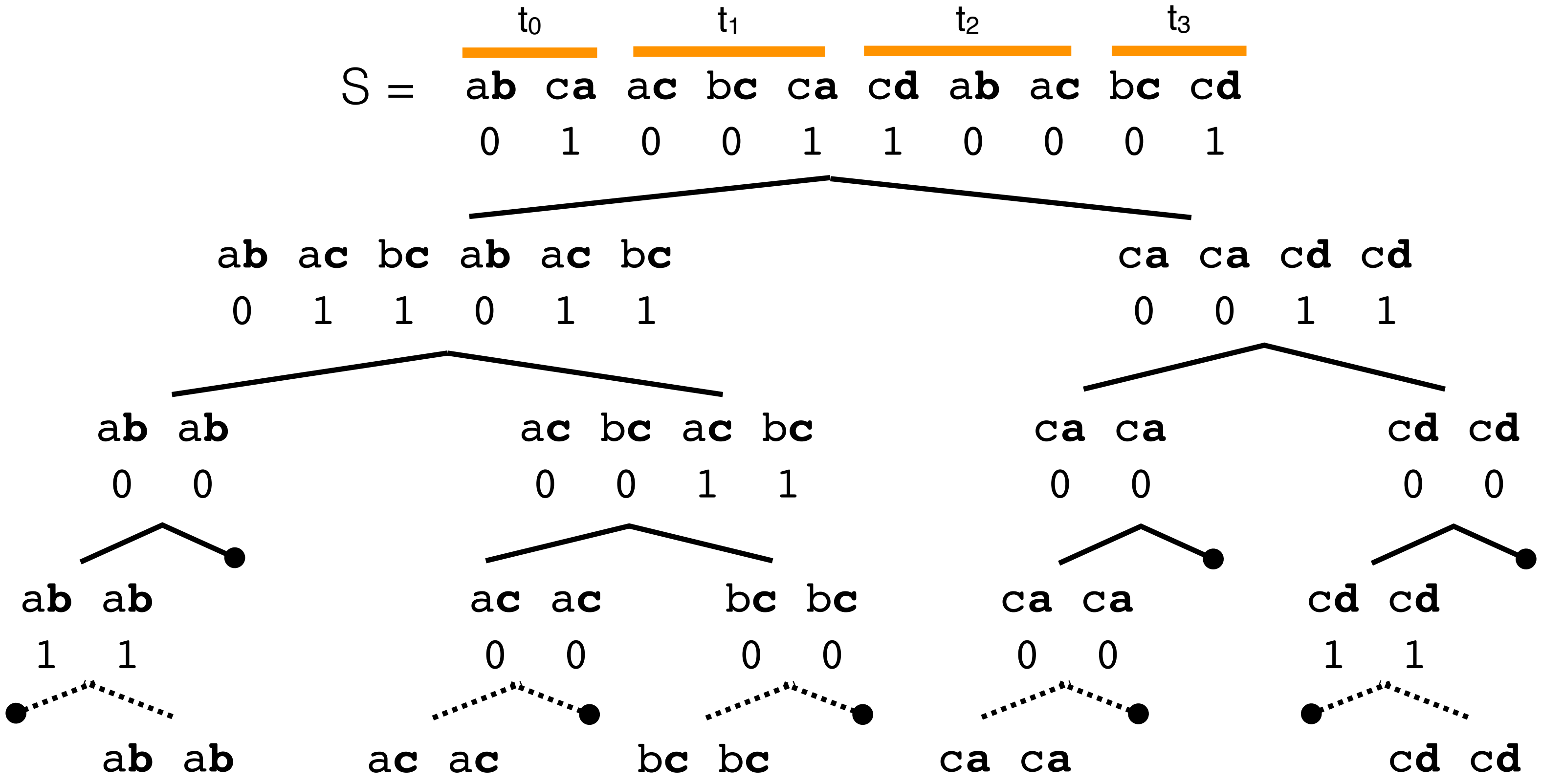
- Temporal Log as a sequence of bi-dimensional symbols representing edges in a Wavelet Tree.
  - Events are ordered by time instant.
- Direct and reverse neighbors are solved in the same time performance.

Source Vertex	
a	00
b	01
c	10
d	11

Target Vertex	
a	**00**
b	**01**
c	**10**
d	**11**



Interleaving Code	
ab	0001
ac	0100
bc	0110
ca	1000
cd	1101



Outline

- ✓ Definition and Motivation.
- ✓ Previous works about temporal graphs.
- ✓ Compression of temporal graphs.
- Contributions
    - ✓ Based on inverted indexes:
        - ✓ EdgeLog
        - ✓ EveLog
    - ✓ Based on Wavelet Trees:
        - ✓ Compact Adjacency Sequence (CAS)
        - ✓ Compact Events ordered by Time (CET)
    - Based on the Compressed Suffix Array:
        - Temporal Graph CSA
    - Based on the multidimensional k^d-tree
        - The Compressed k^d-tree
- Evaluation.
- Conclusions and future works.

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
    - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

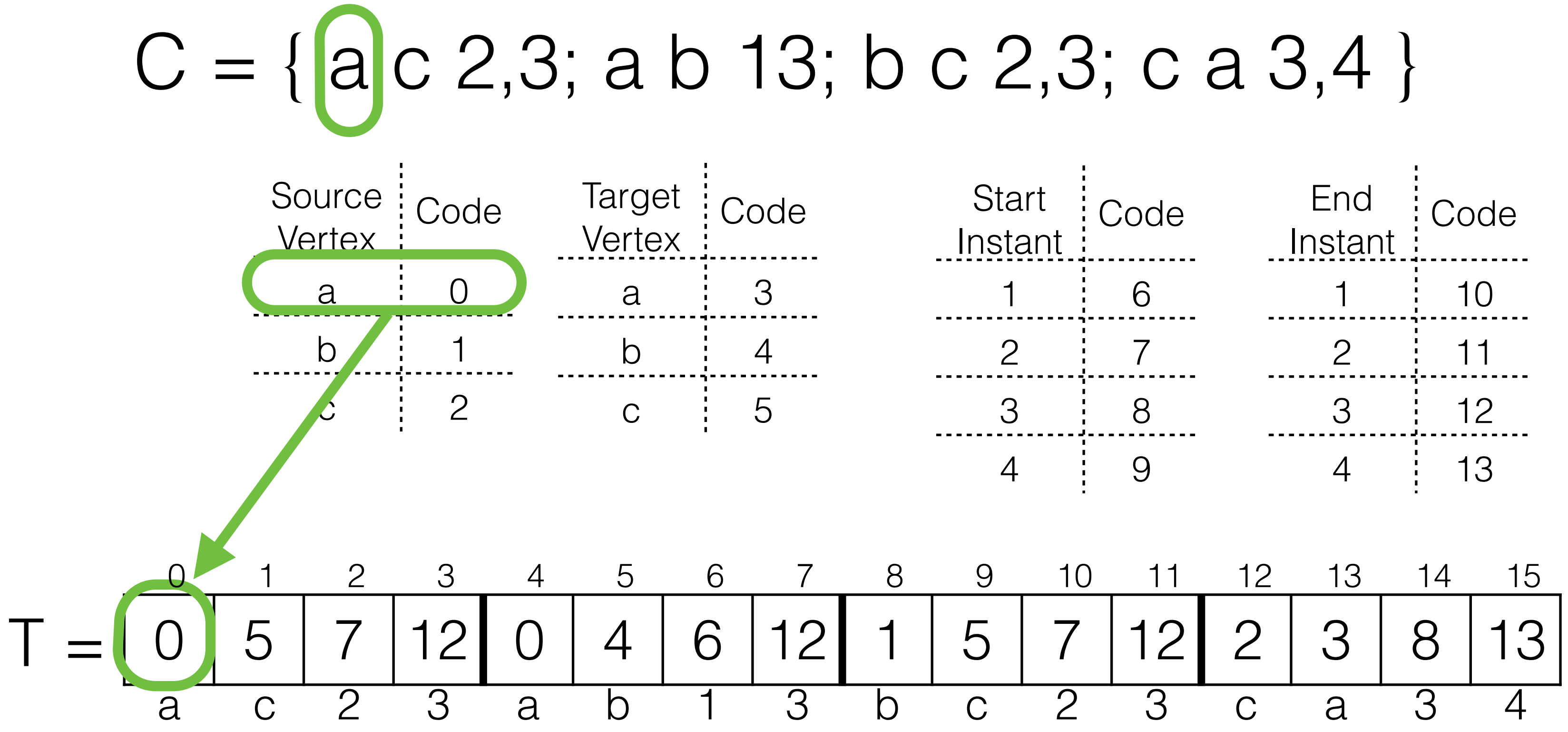
Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
    - Operations solved via pattern matching.



C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
    - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

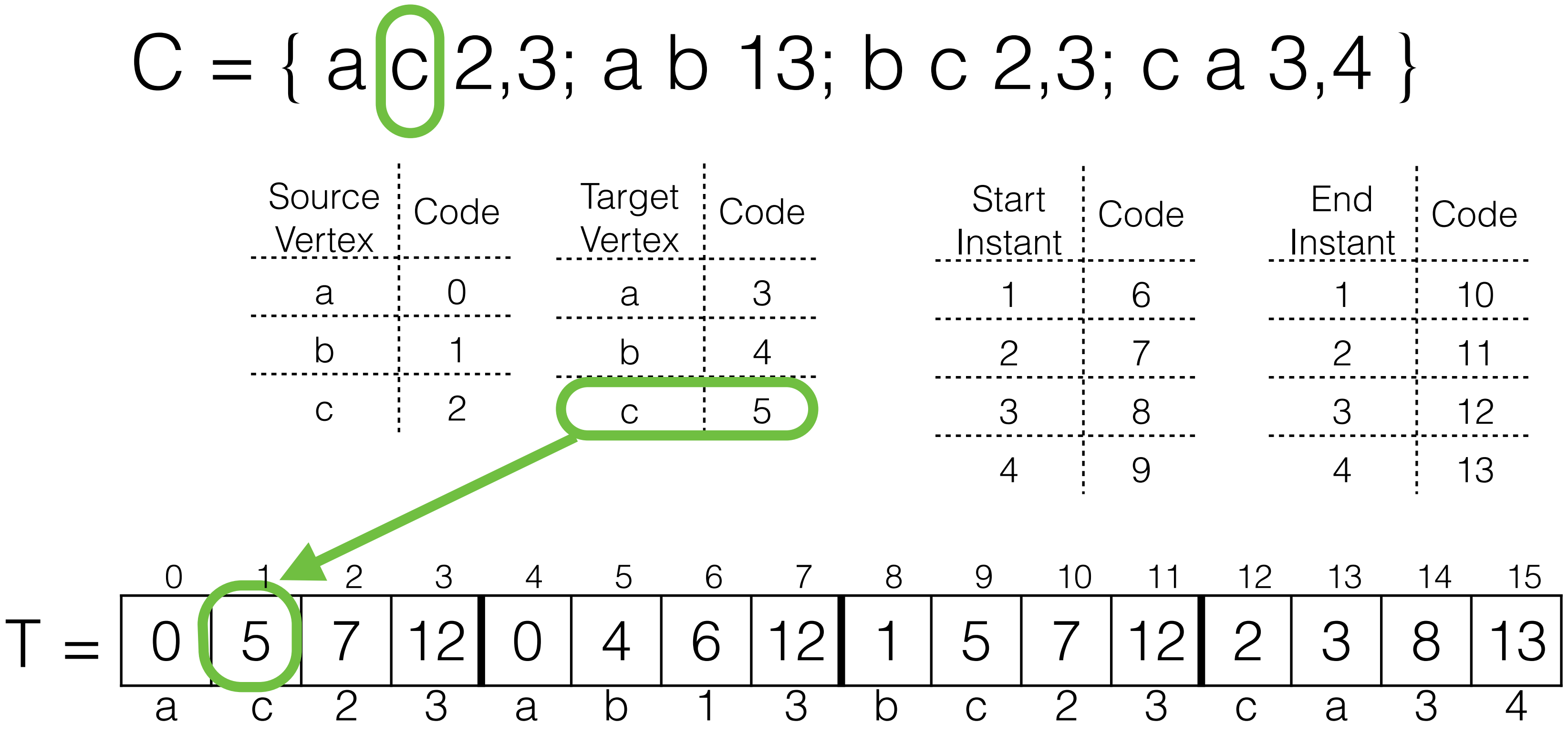
End Instant	Code
1	10
2	11
3	12
4	13

T =

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.



C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

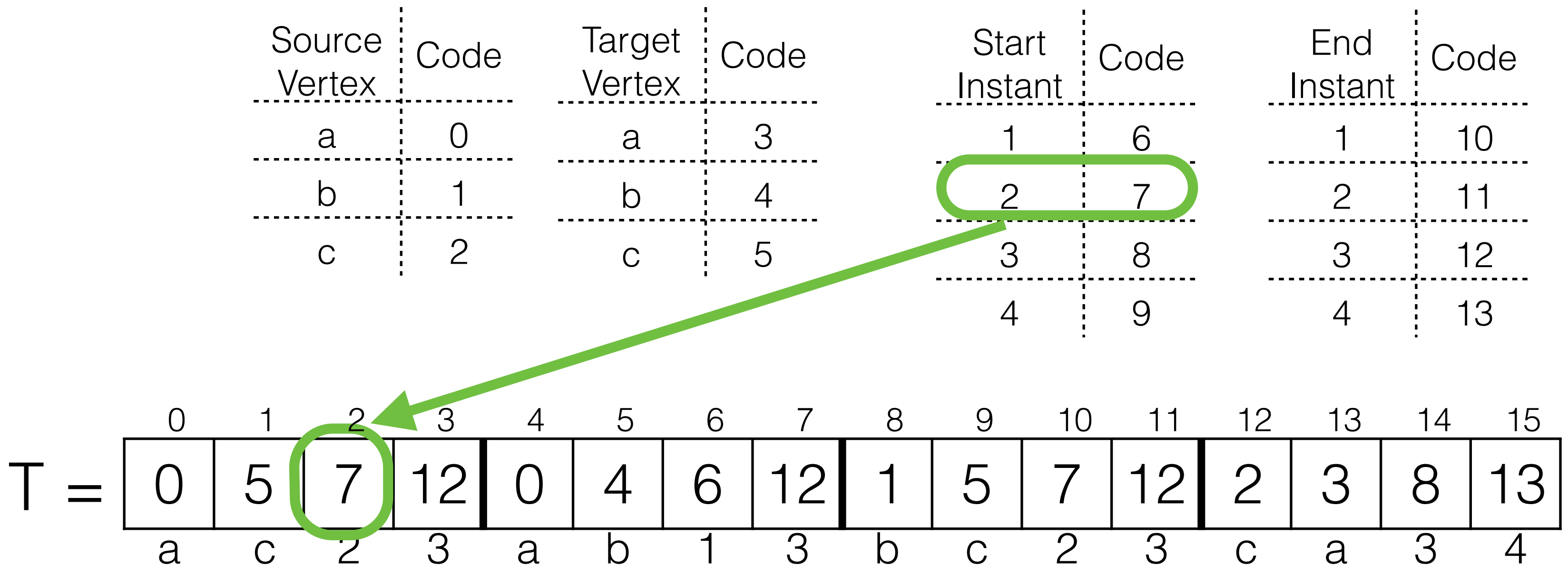
T =

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }



Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

T =

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

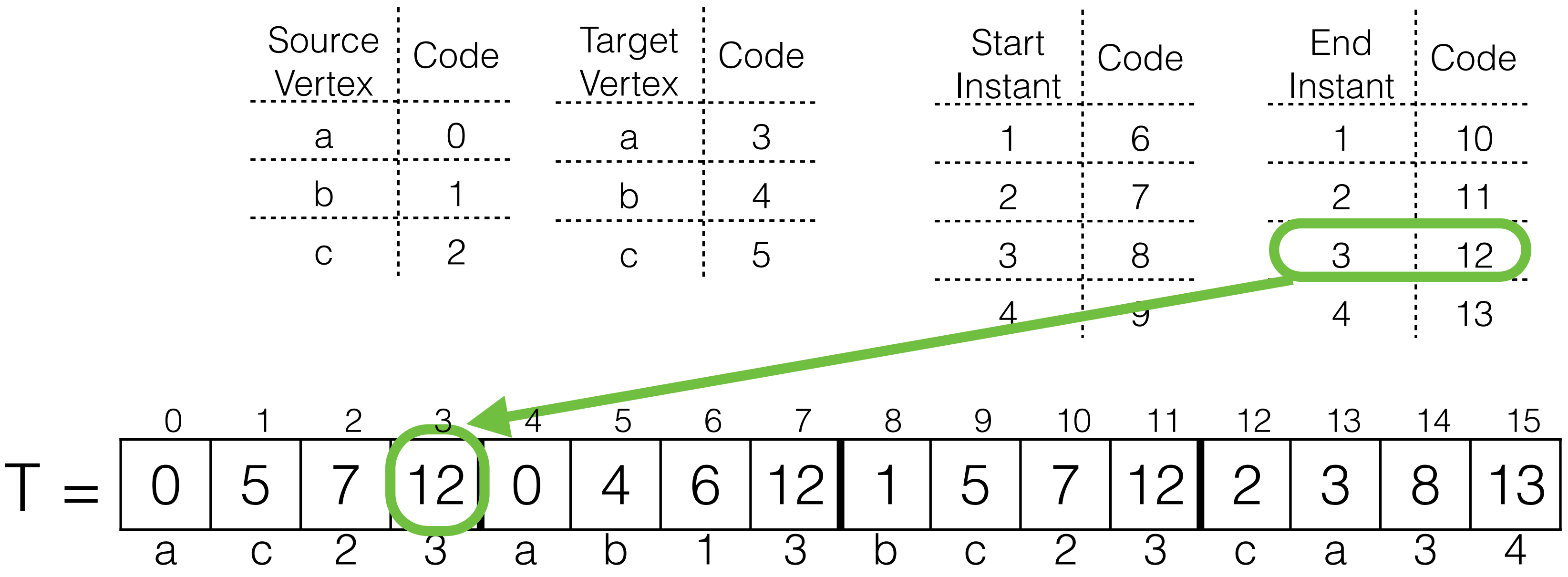
T =

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
    - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }



Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
    - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
    - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

T =

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

A =

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =

5	6	7	4	11	8	9	10	12	13	14	15	2	0	3	1
a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4
Source vertices				Target vertices				Start Instant				End Instant			

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
    - Operations solved via pattern matching.

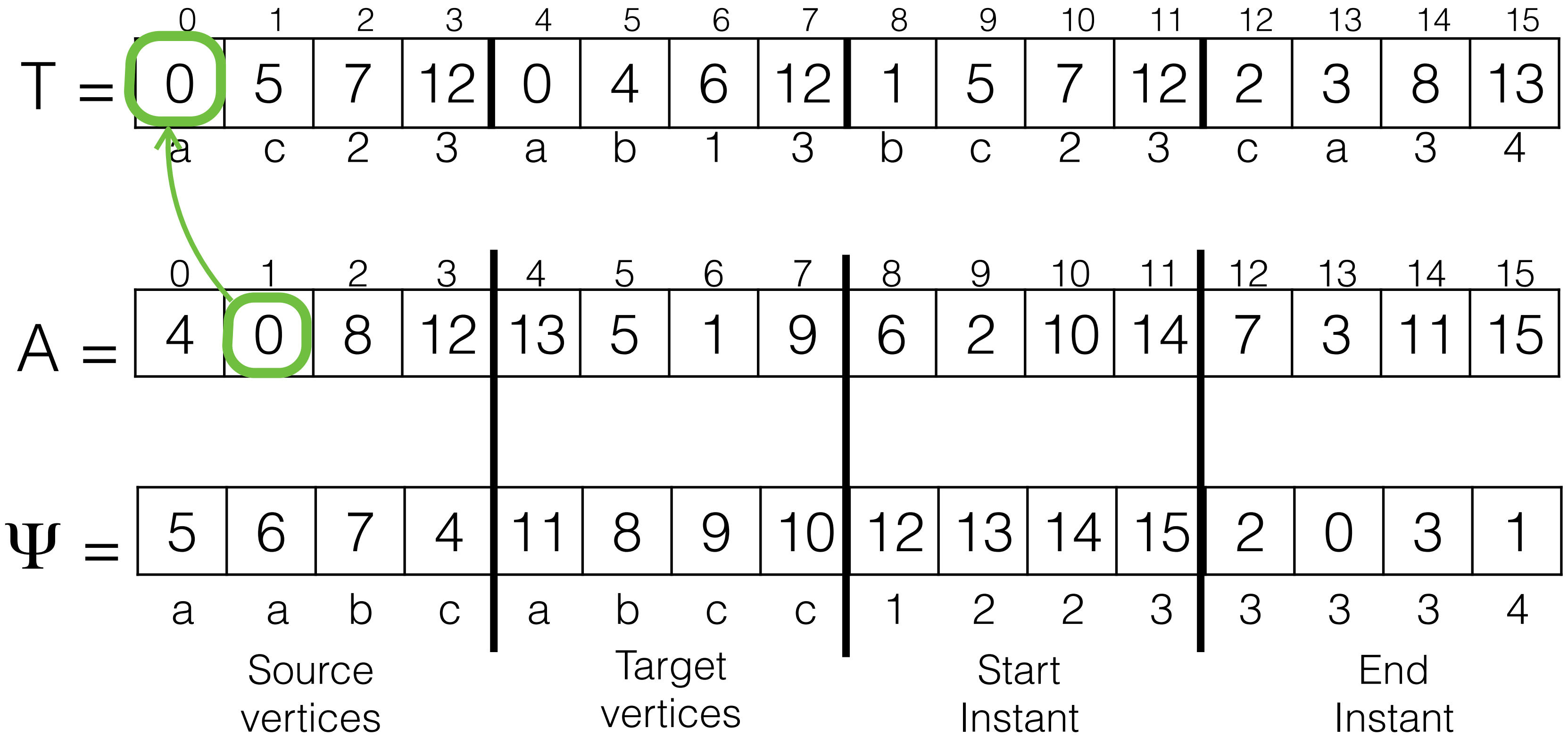
C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	2	0	3	1
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

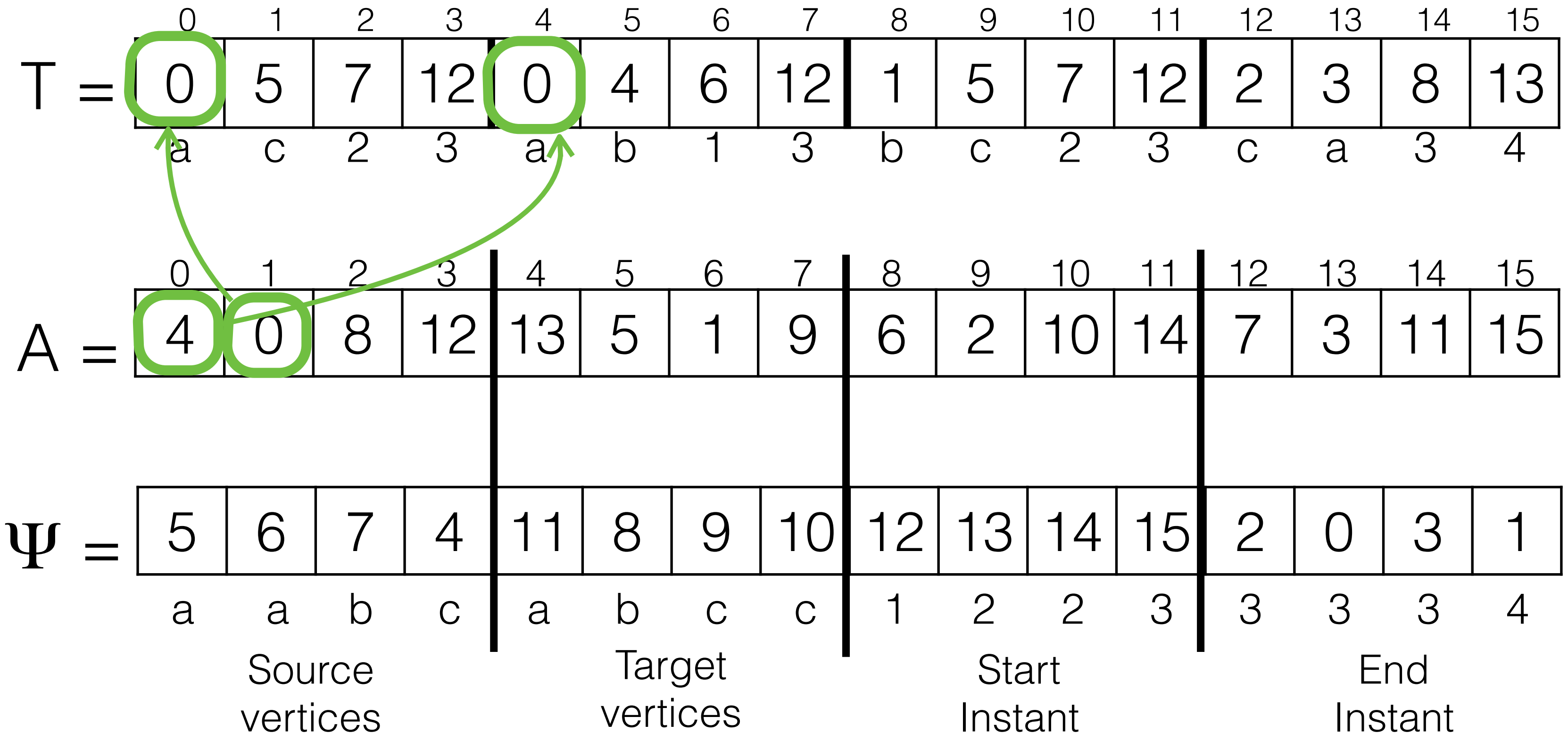
C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	2	0	3	1
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
    - Operations solved via pattern matching.

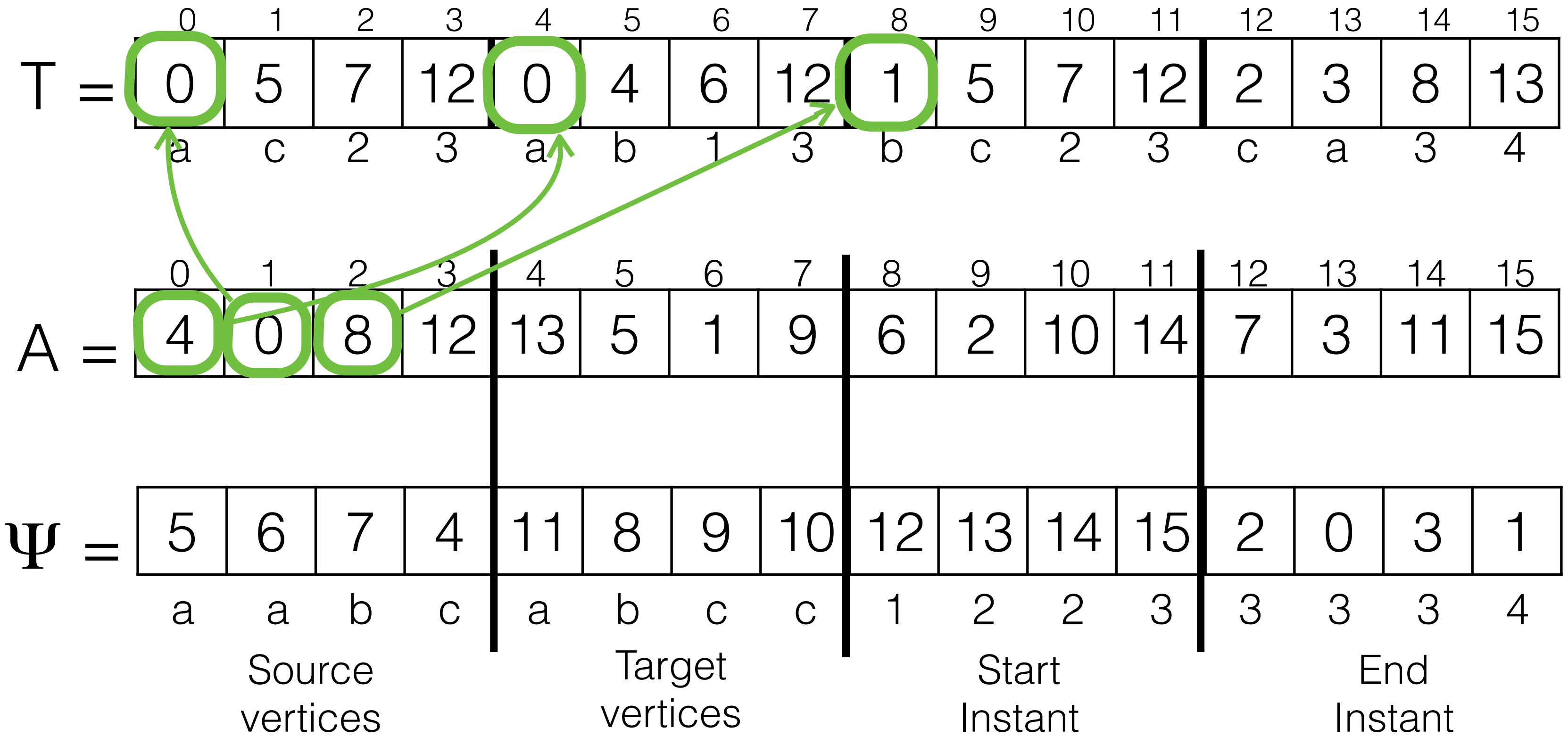
C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	2	0	3	1
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

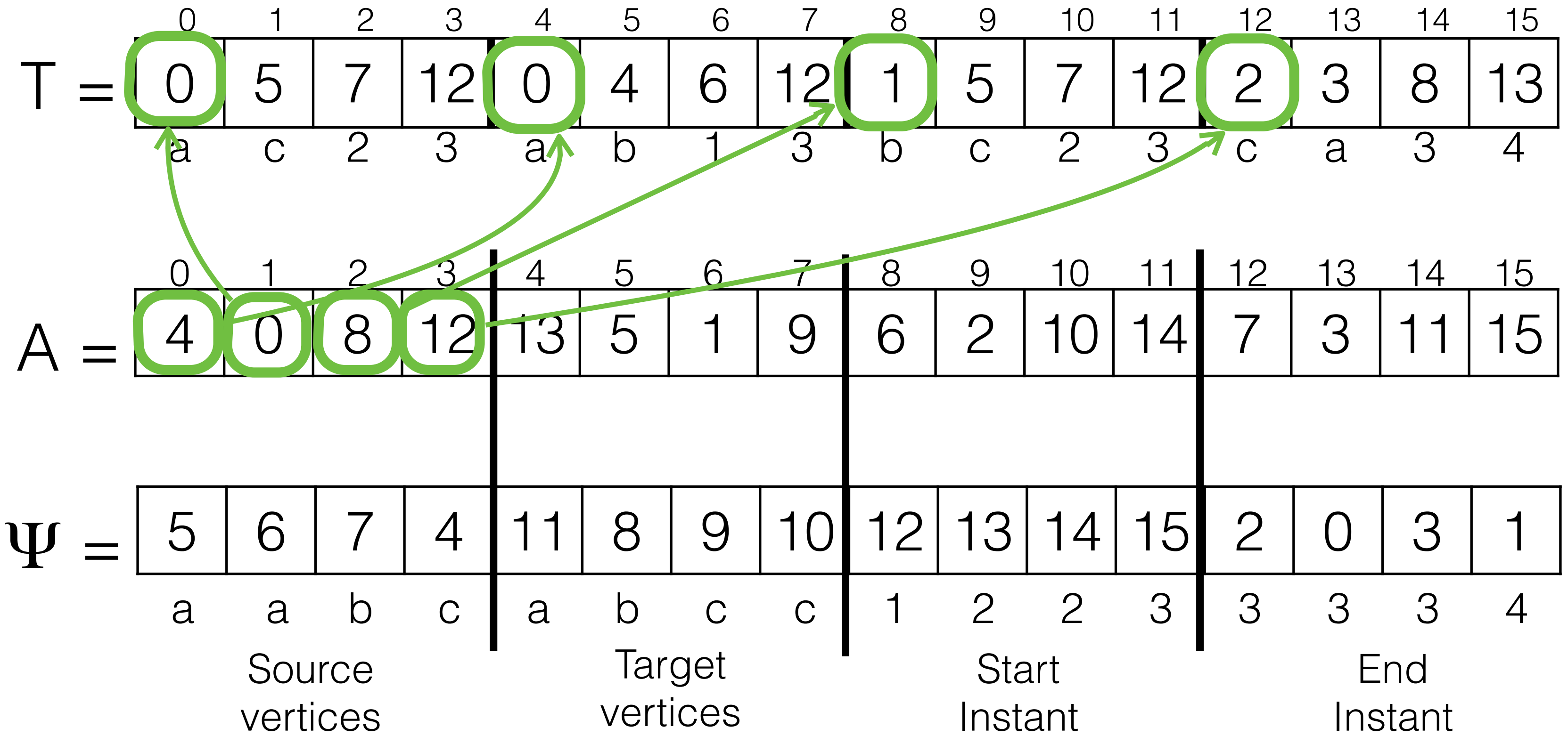
$$C = \{\ a\ c\ 2,3;\ a\ b\ 13;\ b\ c\ 2,3;\ c\ a\ 3,4\ \}$$

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	2	0	3	1
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15
Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	2	0	3	1
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4
	Source vertices				Target vertices				Start Instant				End Instant			

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

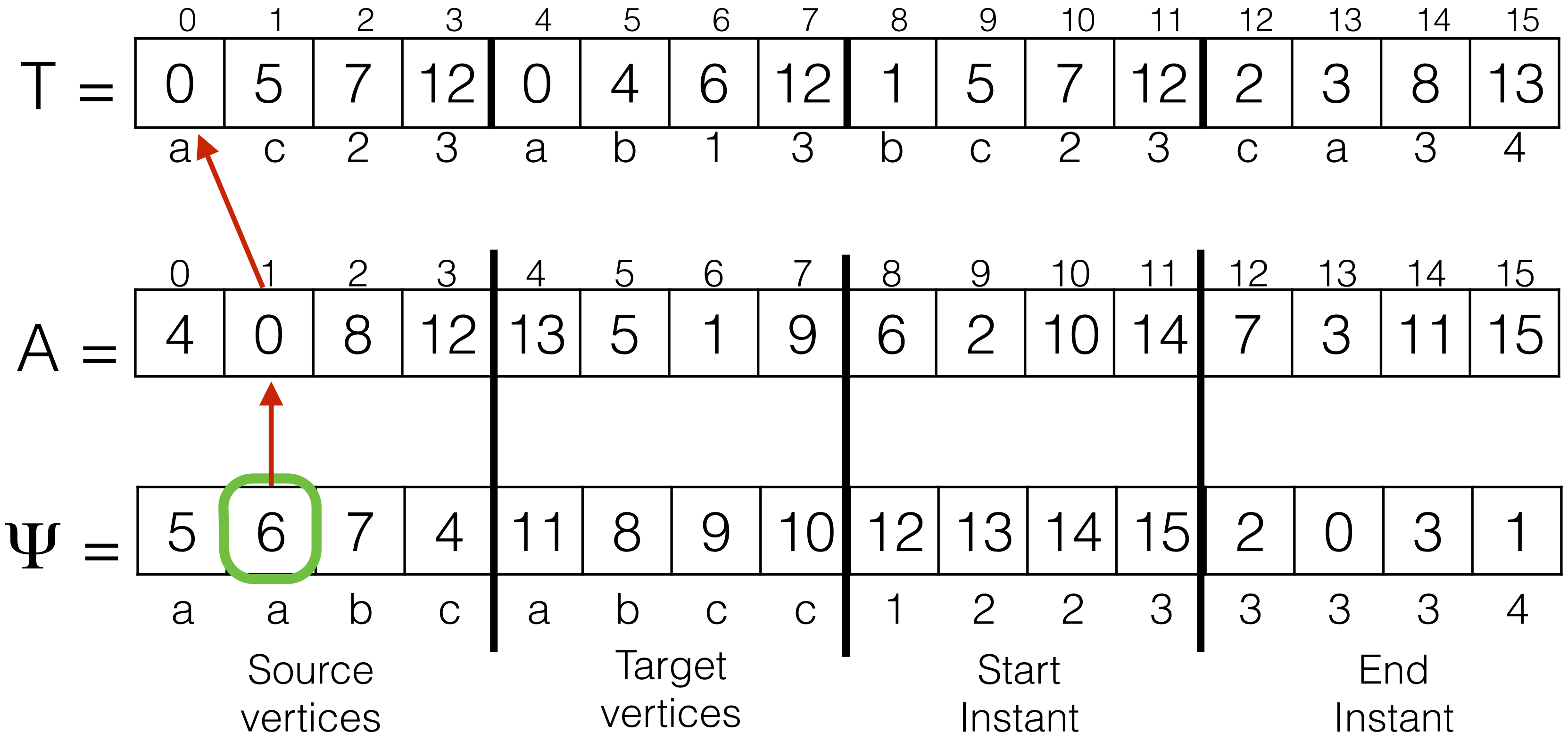
$$C = \{ a\ c\ 2,3;\ a\ b\ 13;\ b\ c\ 2,3;\ c\ a\ 3,4 \}$$

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	2	0	3	1
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

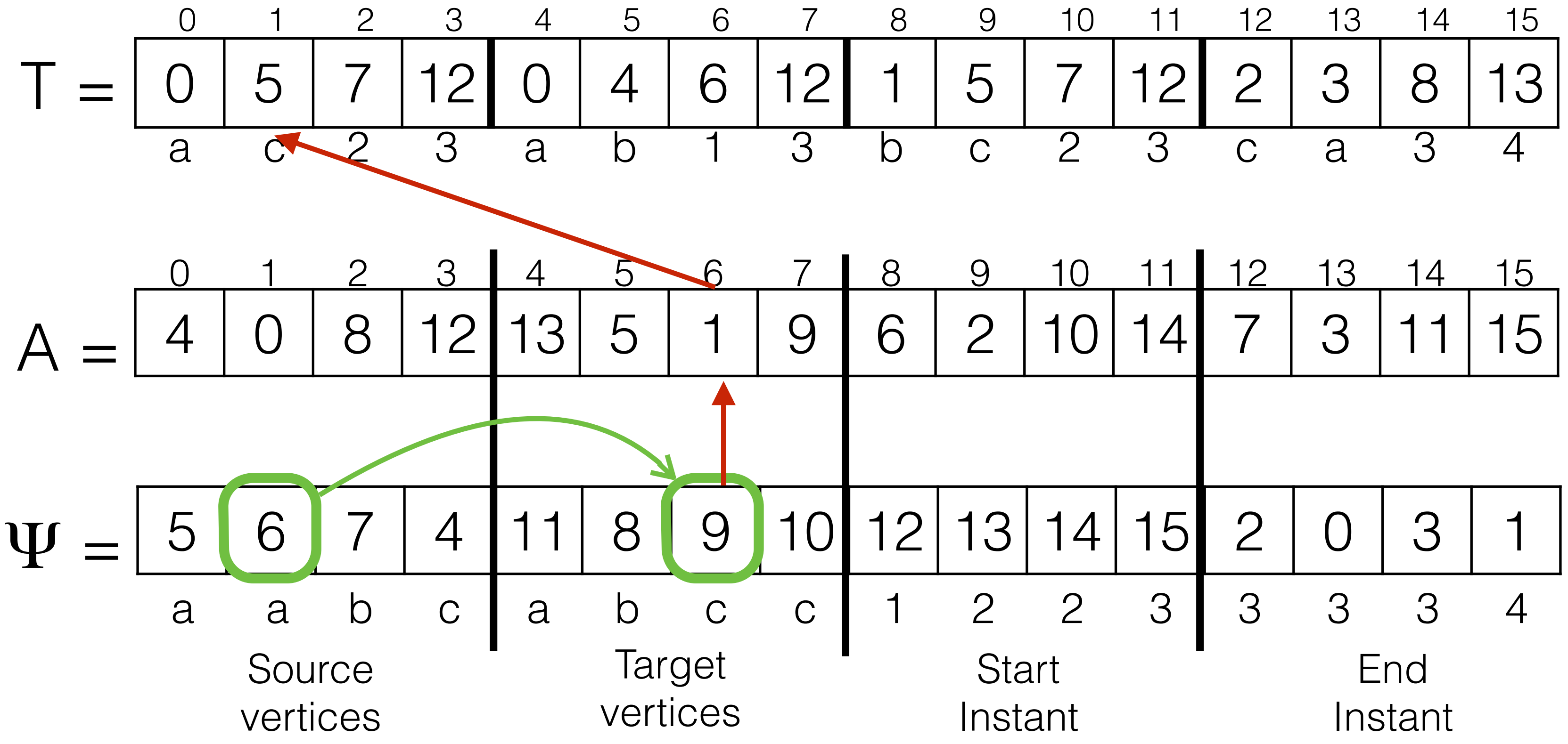
$$C = \{ a\ c\ 2,3;\ a\ b\ 13;\ b\ c\ 2,3;\ c\ a\ 3,4 \}$$

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	2	0	3	1
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

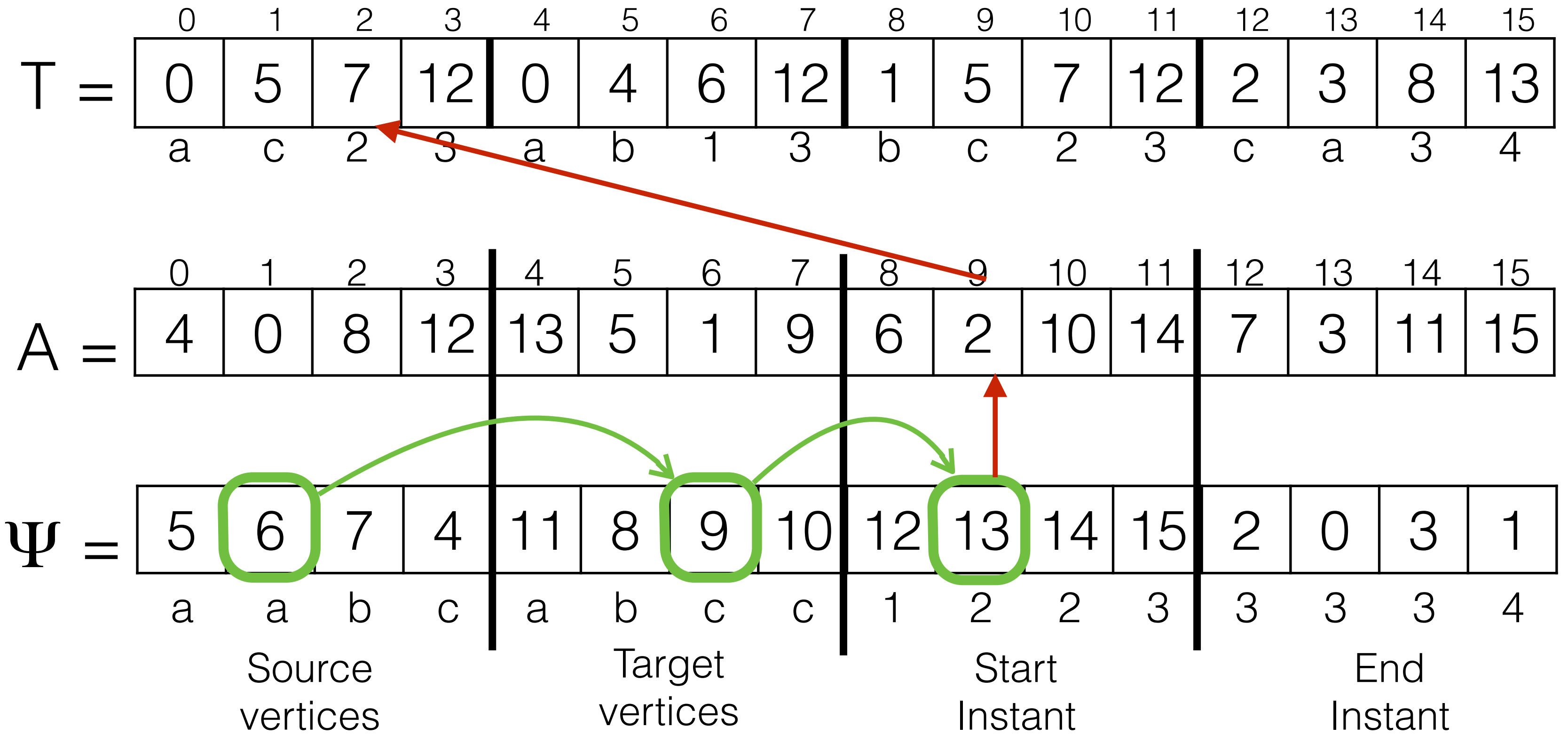
$$C = \{ a\ c\ 2,3;\ a\ b\ 13;\ b\ c\ 2,3;\ c\ a\ 3,4 \}$$

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	2	0	3	1
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

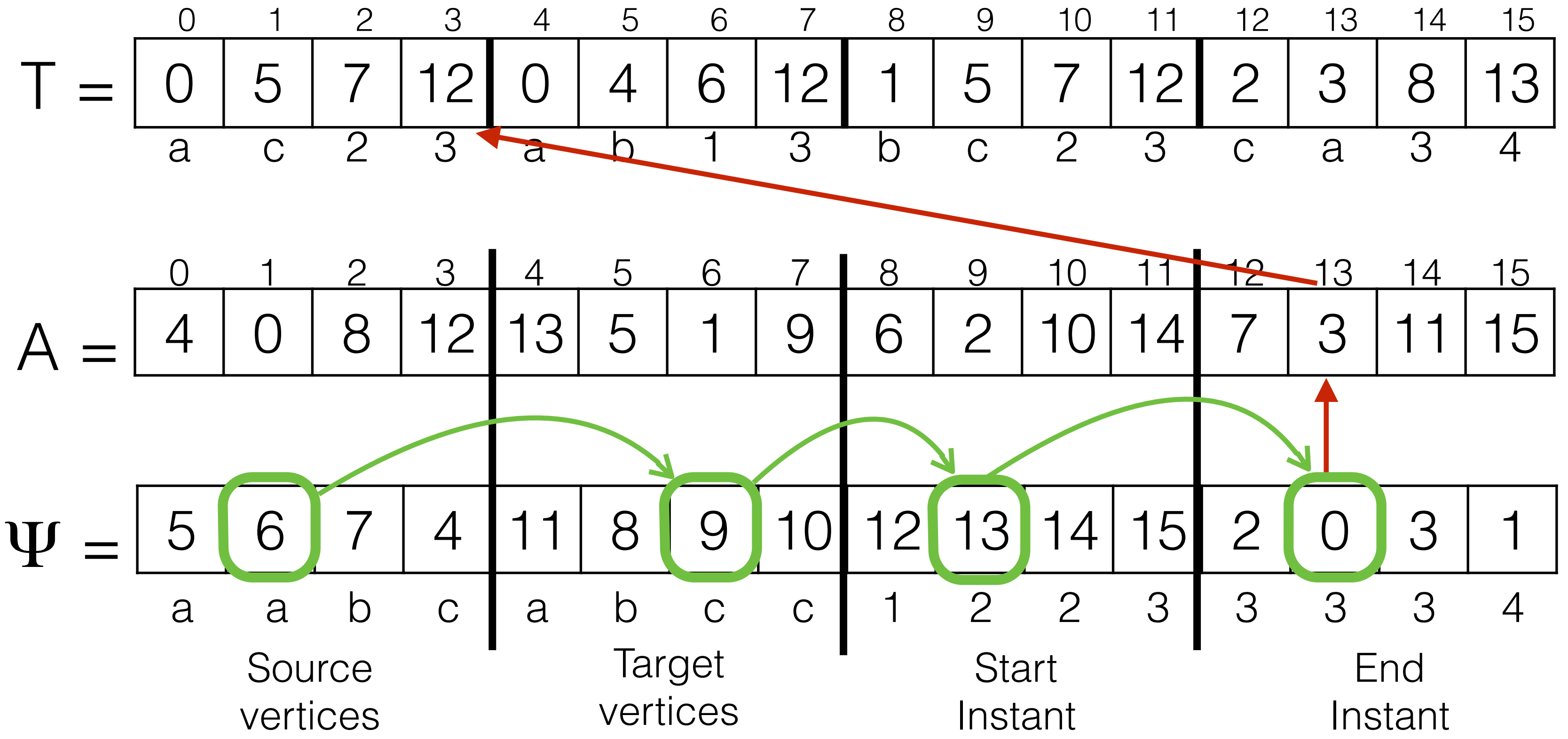
$$C = \{ a\ c\ 2,3;\ a\ b\ 13;\ b\ c\ 2,3;\ c\ a\ 3,4 \}$$

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	2	0	3	1
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

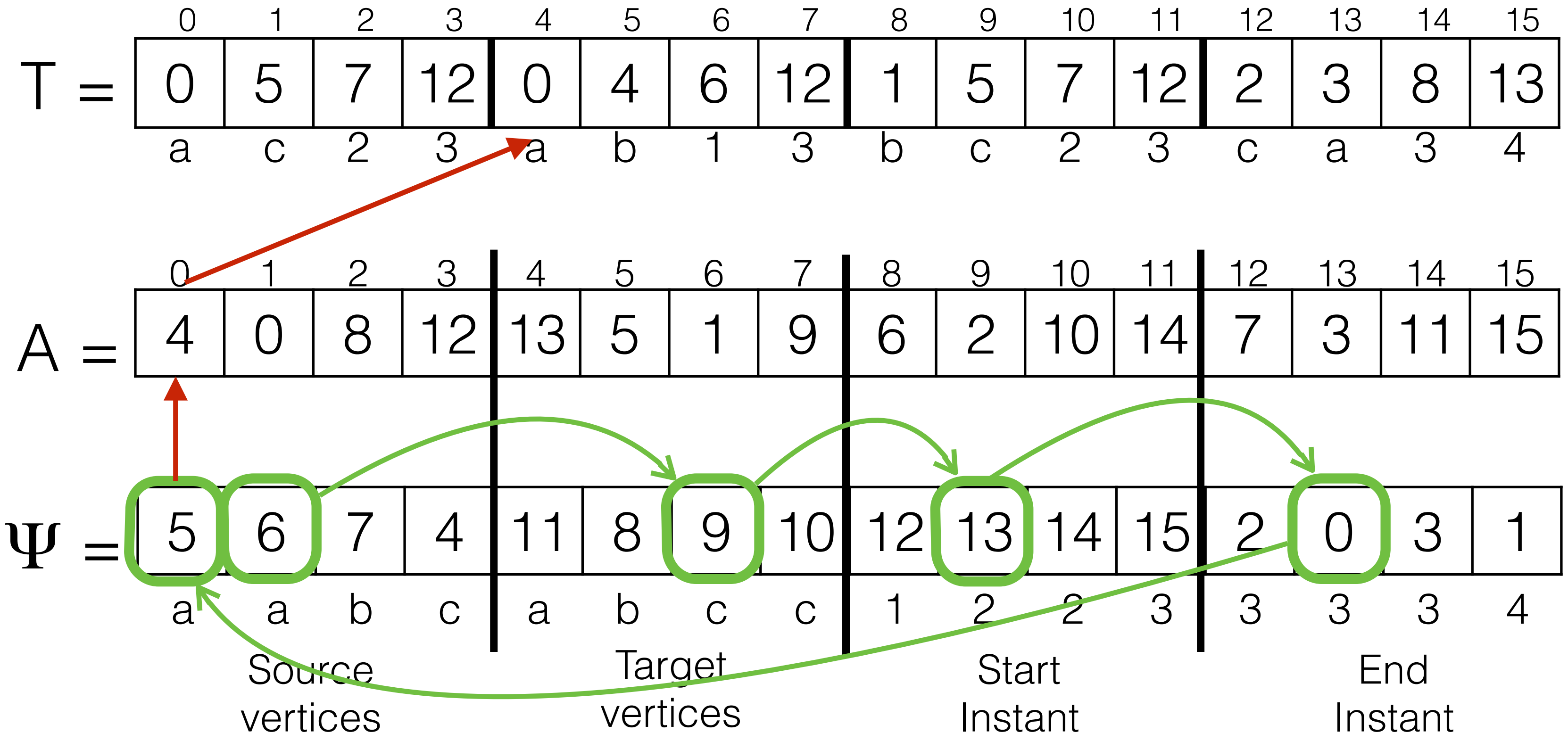
C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	2	0	3	1
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
    - Operations solved via pattern matching.

$$C = \{\ a\ c\ 2,3;\ a\ b\ 13;\ b\ c\ 2,3;\ c\ a\ 3,4\ \}$$

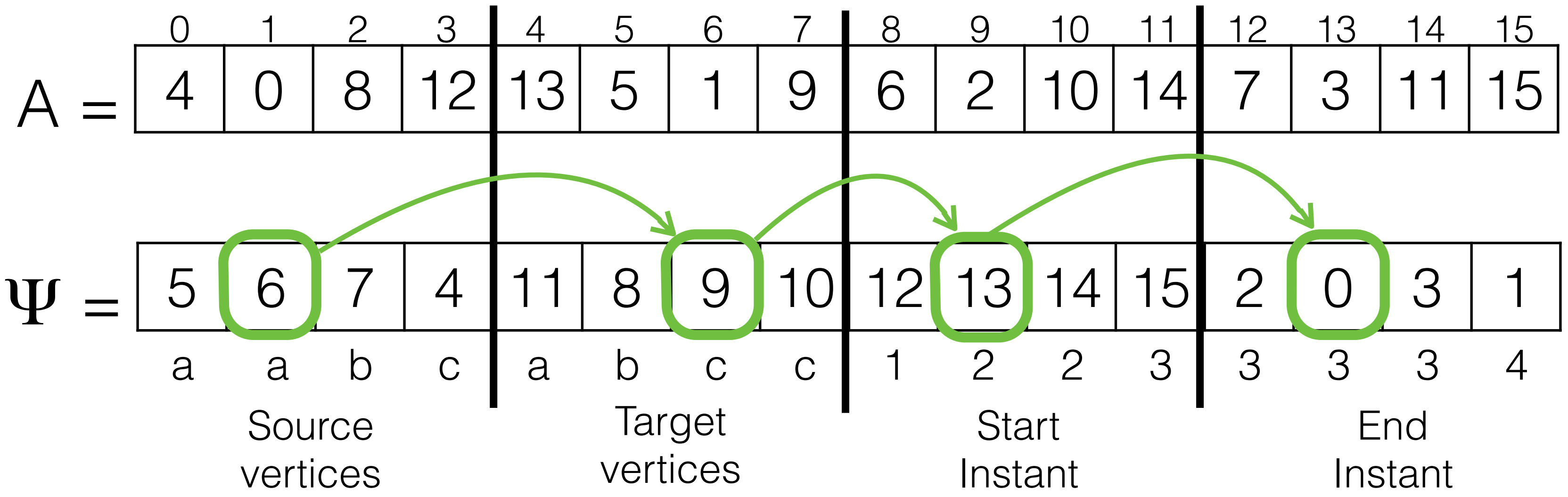
Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15
Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	2	0	3	1
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

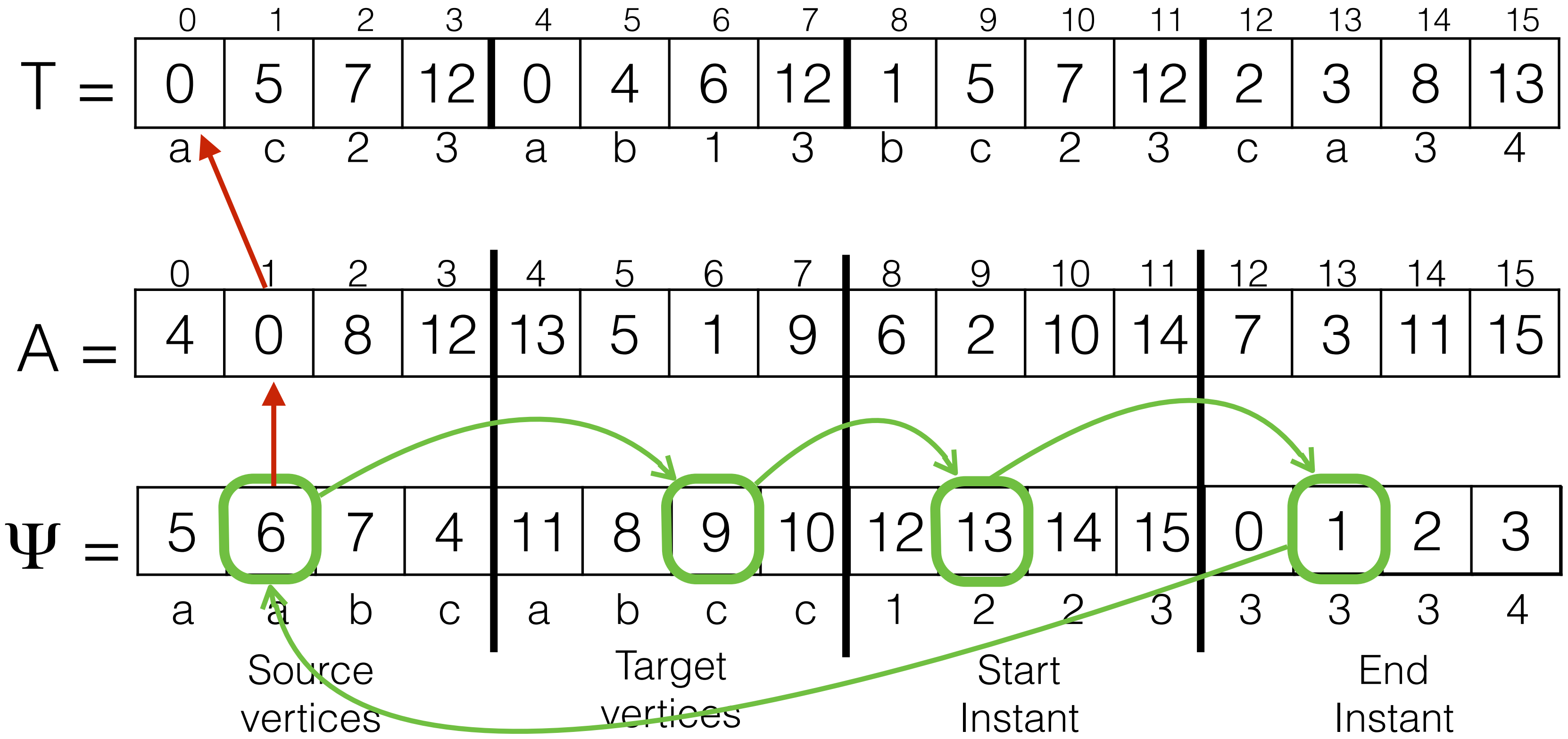
$$C = \{ a\ c\ 2{,}3;\ a\ b\ 13;\ b\ c\ 2{,}3;\ c\ a\ 3{,}4 \}$$

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	0	1	2	3
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15
Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	0	1	2	3
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4
	Source vertices				Target vertices				Start Instant				End Instant			

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
    - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

edge(ab,t=2)?

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	0	1	2	3
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4
	Source vertices				Target vertices				Start Instant				End Instant			

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

edge(ab,t=2)?

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15
Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	0	1	2	3
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4
	Source vertices				Target vertices				Start Instant				End Instant			

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

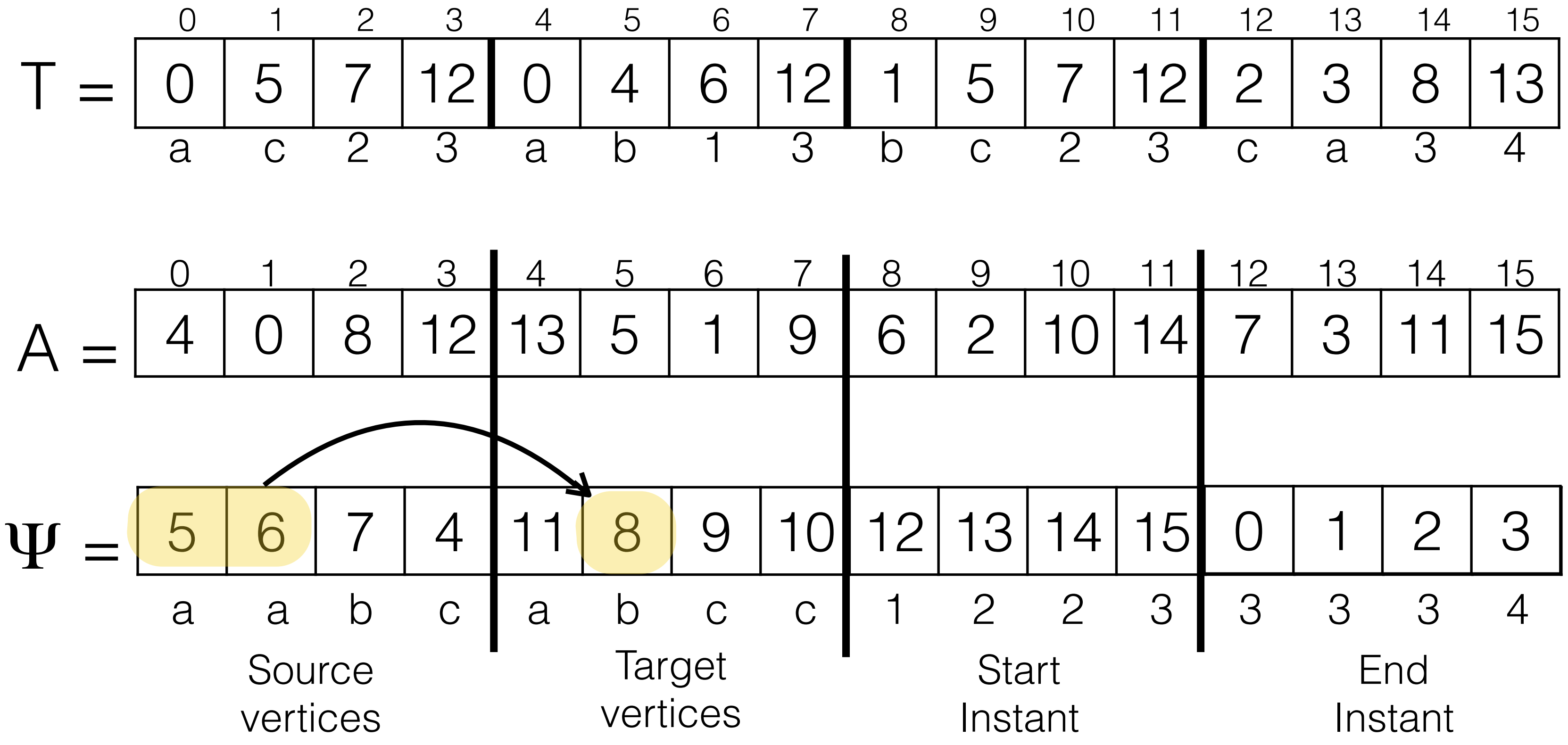
Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

edge(ab,t=2)?



T =

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

A =

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =

5	6	7	4	11	8	9	10	12	13	14	15	0	1	2	3
a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

$$C = \{\ a\ c\ 2{,}3;\ a\ b\ 13;\ b\ c\ 2{,}3;\ c\ a\ 3{,}4\ \}$$

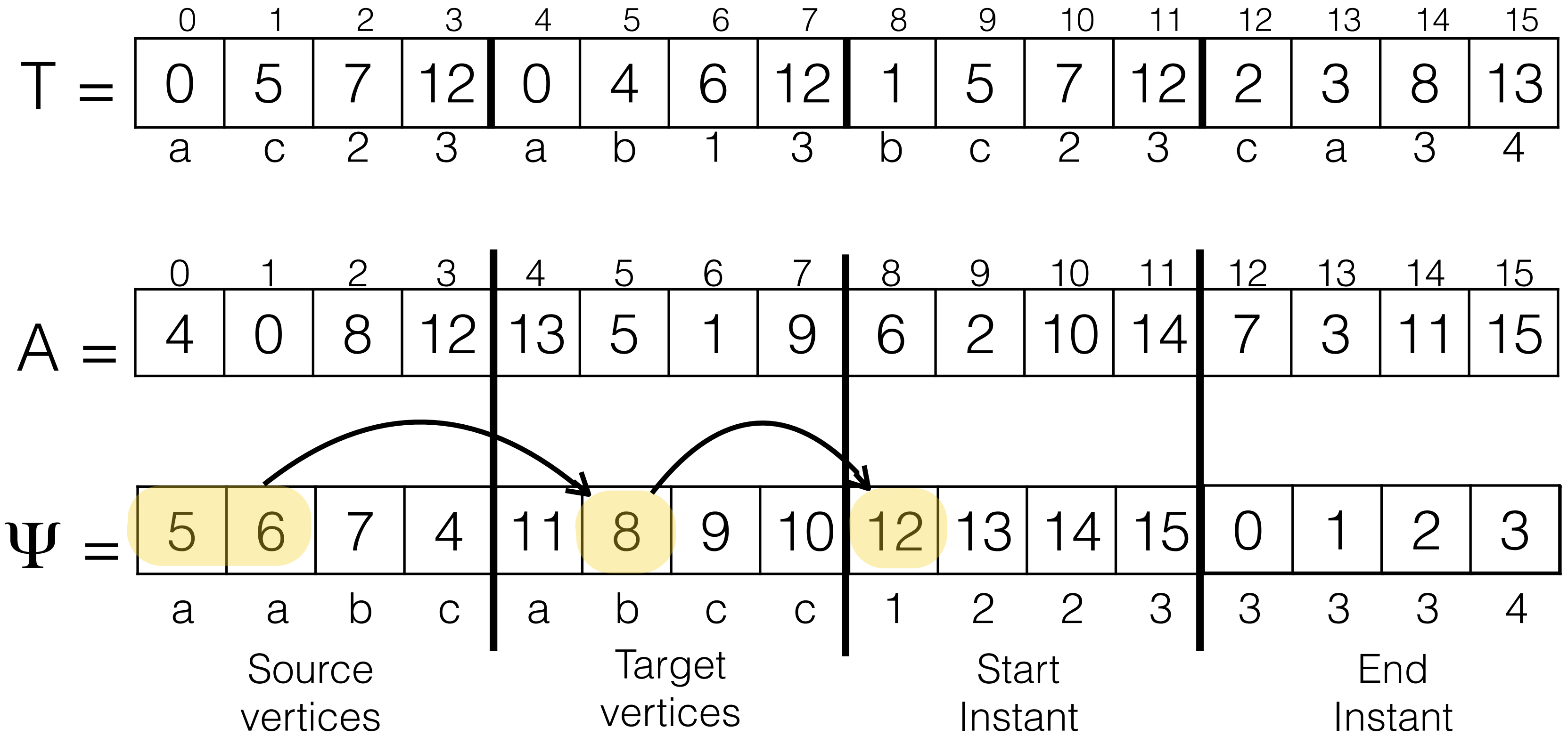
Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

edge(ab,t=2)?



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	0	1	2	3
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
  - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

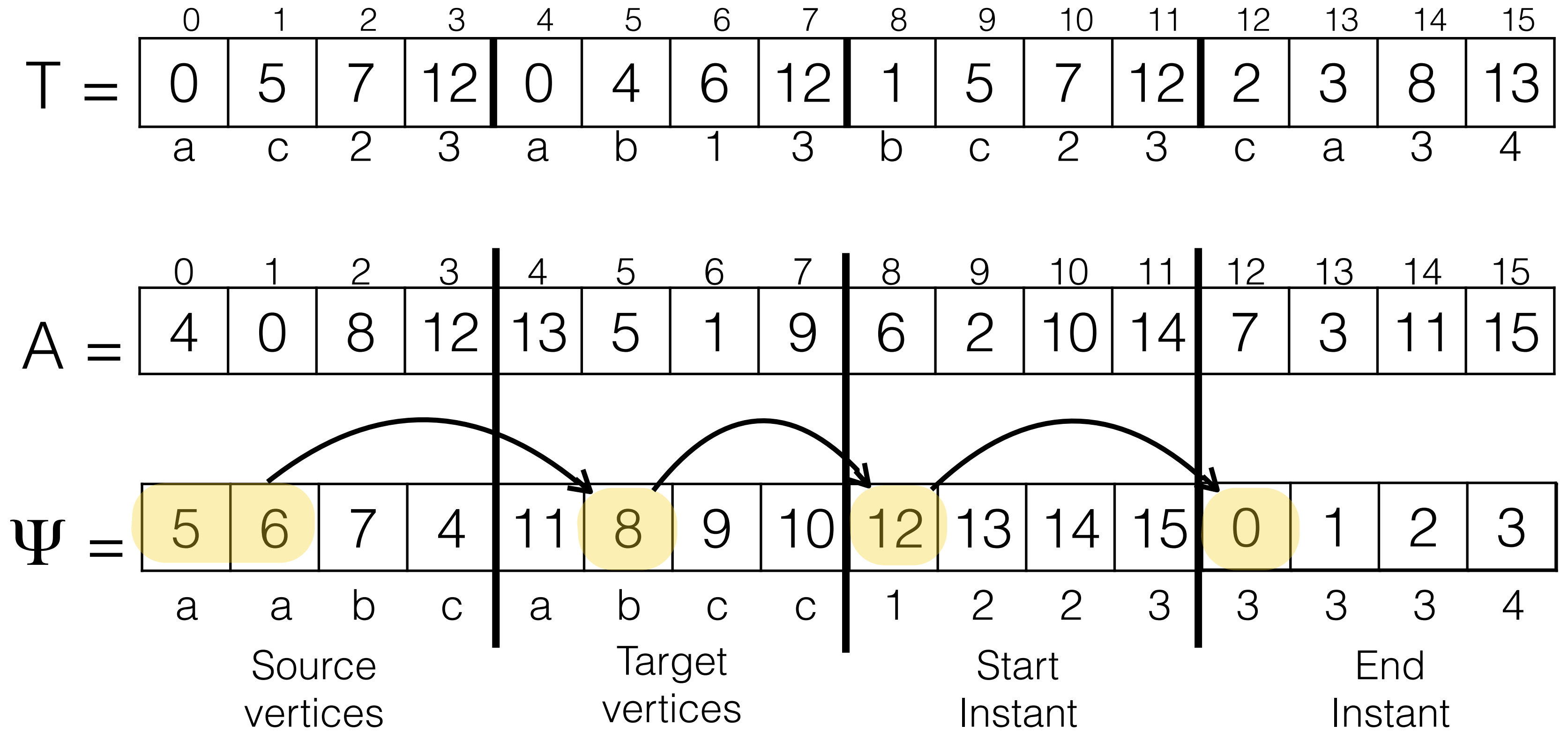
Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

edge(ab,t=2)?



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	0	1	2	3
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
    - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

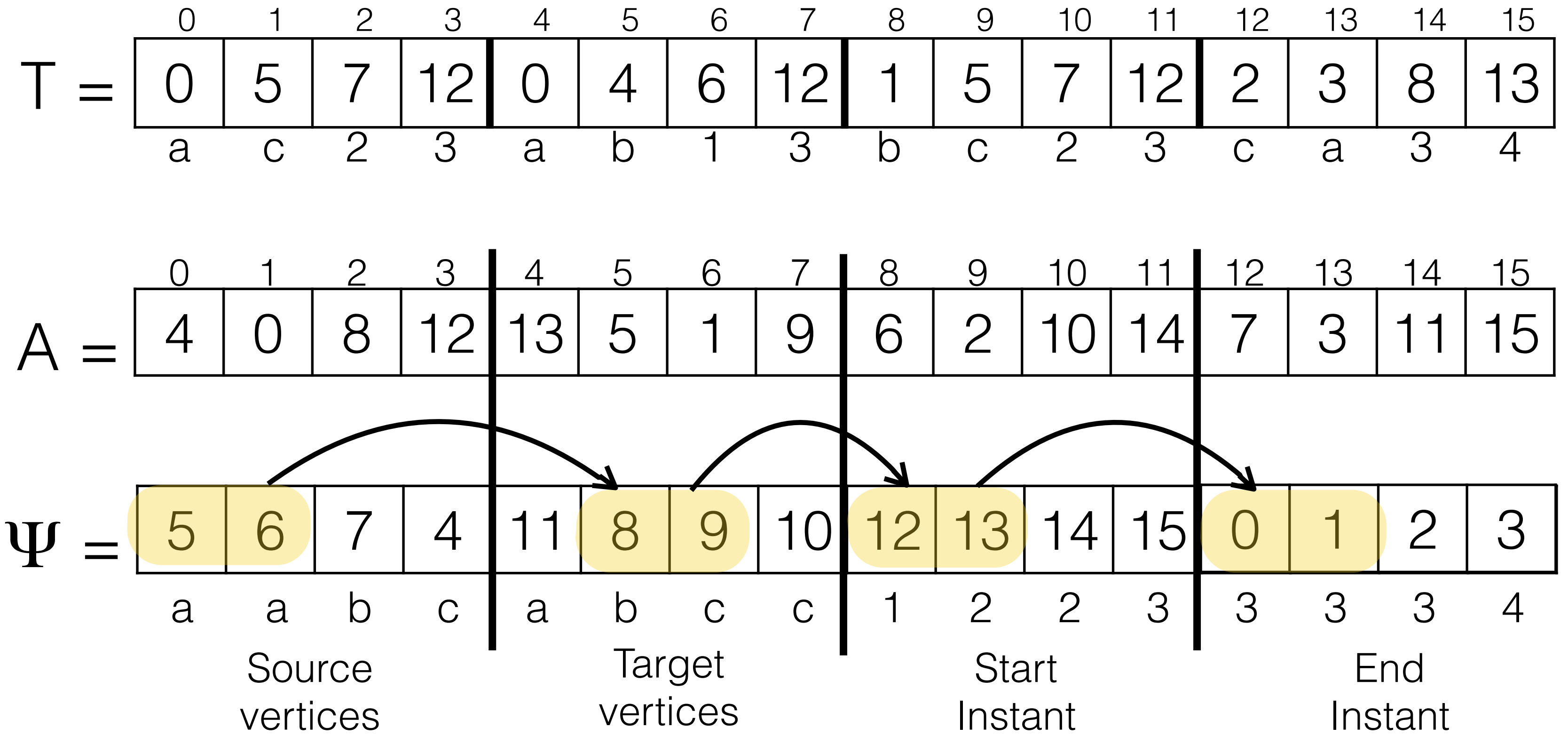
Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

dirnei(a,t=2)?



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15
Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	0	1	2	3
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Temporal Graph CSA (TG-CSA)

- Temporal Graph is transformed into a text that is the concatenation of contacts.
- Text is represented in a CSA
    - Operations solved via pattern matching.

C = { a c 2,3; a b 13; b c 2,3; c a 3,4 }

Source Vertex	Code
a	0
b	1
c	2

Target Vertex	Code
a	3
b	4
c	5

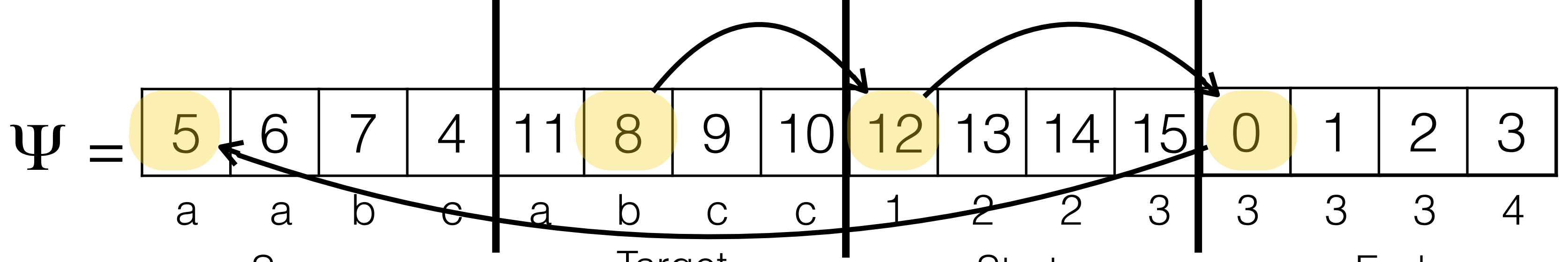
Start Instant	Code
1	6
2	7
3	8
4	9

End Instant	Code
1	10
2	11
3	12
4	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	0	5	7	12	0	4	6	12	1	5	7	12	2	3	8	13
	a	c	2	3	a	b	1	3	b	c	2	3	c	a	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A =	4	0	8	12	13	5	1	9	6	2	10	14	7	3	11	15

revnei(b,t=2)?



Ψ =	5	6	7	4	11	8	9	10	12	13	14	15	0	1	2	3
	a	a	b	c	a	b	c	c	1	2	2	3	3	3	3	4

Source vertices

Target vertices

Start Instant

End Instant

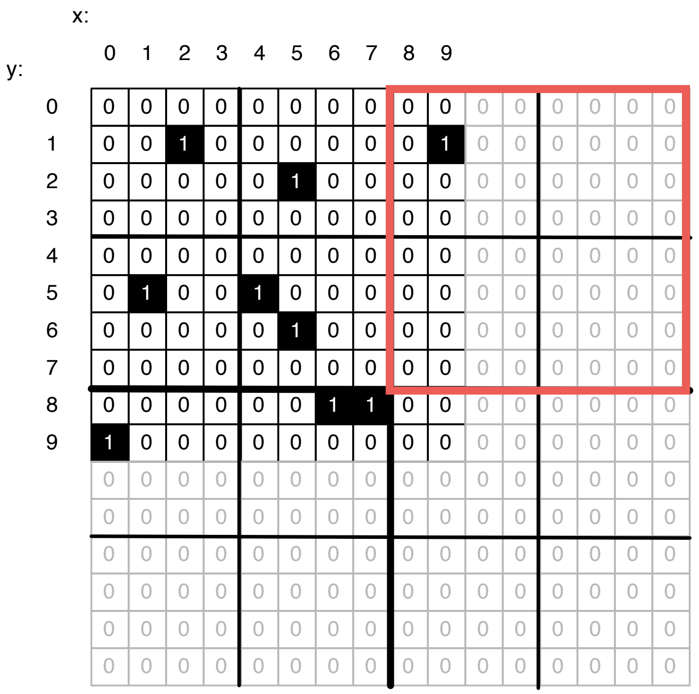
Outline

- ✓ Definition and Motivation.
- ✓ Previous works about temporal graphs.
- ✓ Compression of temporal graphs.
- Contributions
  - ✓ Based on inverted indexes:
    - ✓ EdgeLog
    - ✓ EveLog
  - ✓ Based on Wavelet Trees:
    - ✓ Compact Adjacency Sequence (CAS)
    - ✓ Compact Events ordered by Time (CET)
  - ✓ Based on the Compressed Suffix Array:
    - ✓ Temporal Graph CSA
  - Based on the multidimensional k^d-tree
    - The Compressed k^d-tree
- Evaluation.
- Conclusions and future works.

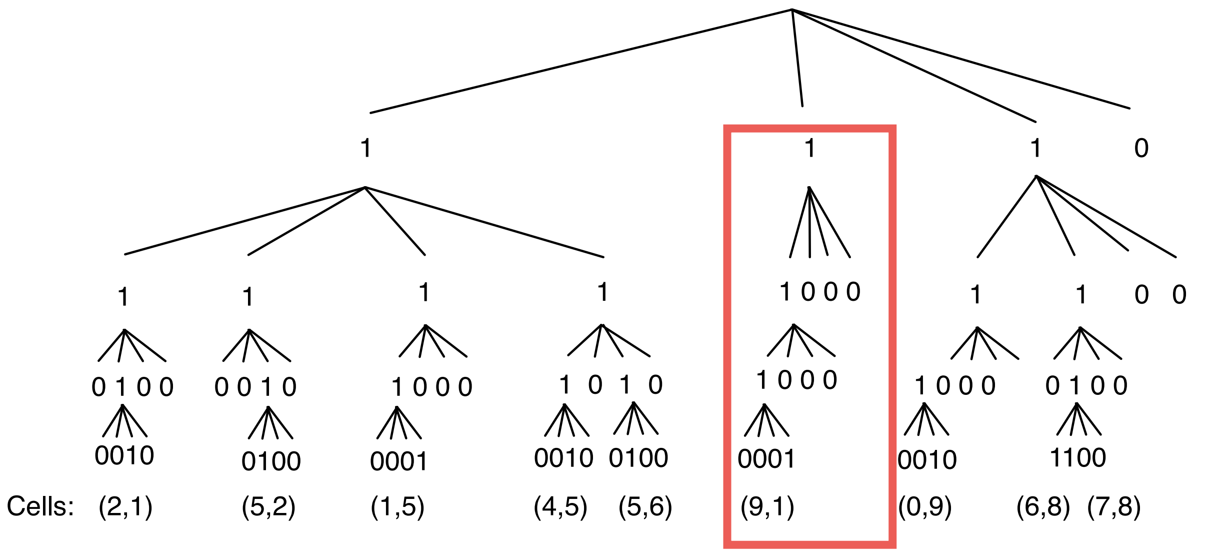
Temporal Graphs as multidimensional matrix

- The k^4-tree can be used to represent the multidimensional matrix.
  - But, compression only works when data is clustered.
  - Contacts in temporal graphs are not clustered.

2D:



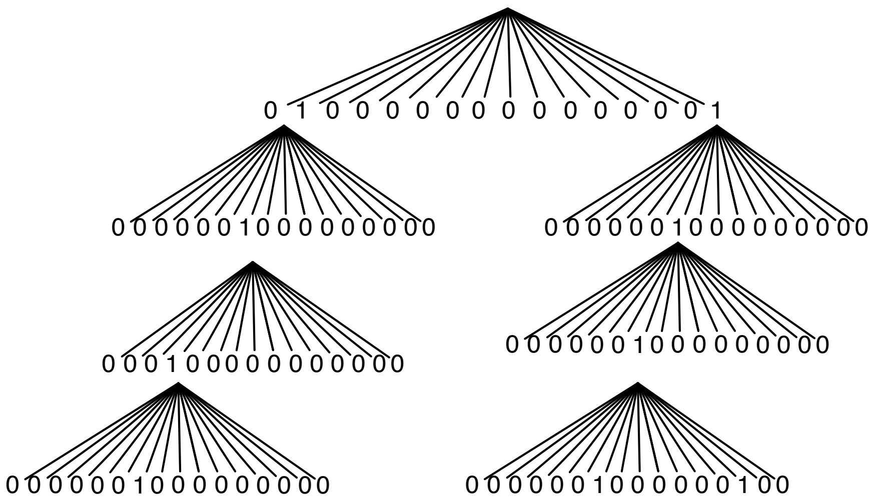
y: \ x:	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



T = 1110 1111 1000 1100 0100 0010 1000 1010 1000 1000 0100

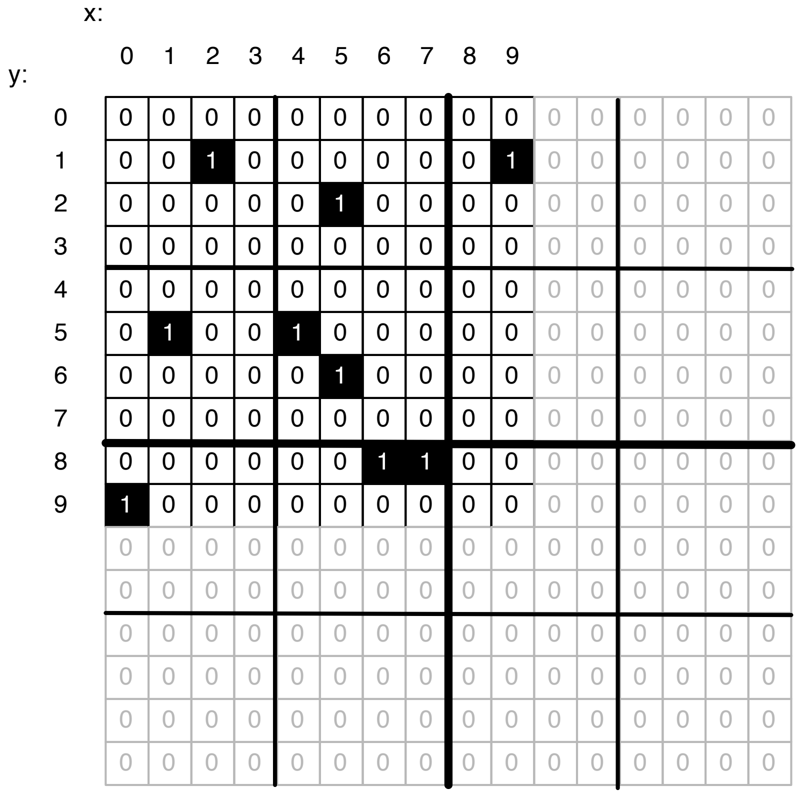
L = 0010 0100 0001 0010 0100 0001 0010 1100

4D:



Compressed k^4-tree (ck^4-tree)

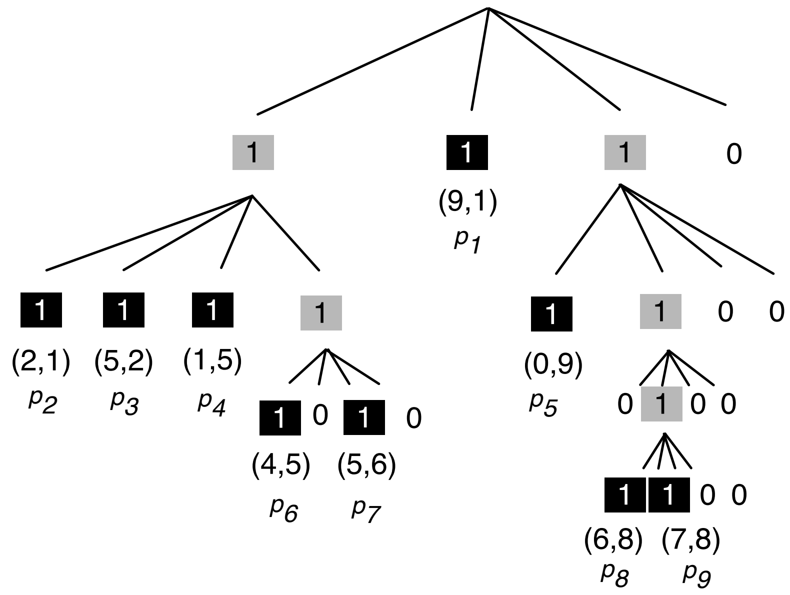
- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
  - Guarantee the information-theoretic lower bound.



x:

y:

	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



T = 1110 1111 1100 1010 0100 1100

B = 010 1110 10 1 1 0 11

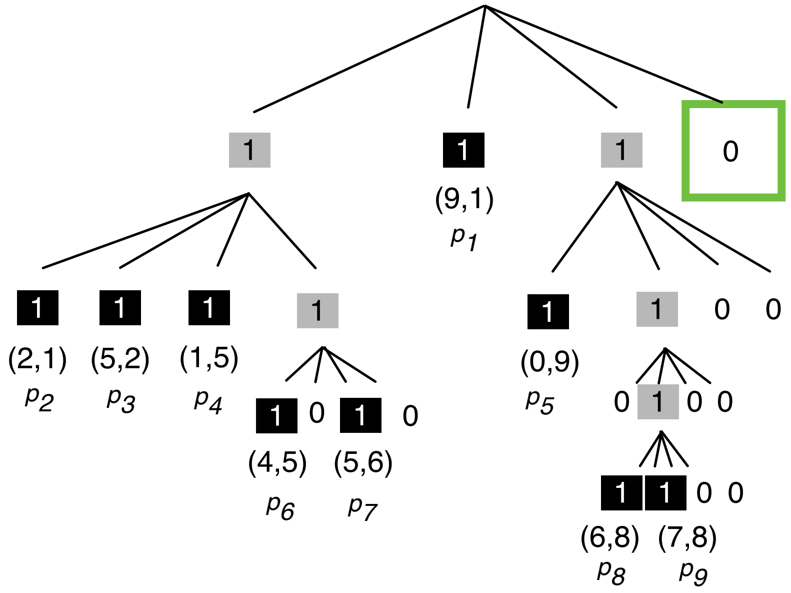
A = [(1',1'), (2',1'), (1', 2'), (1',1'), (0',1'), (0', 1') , (1',0')]

p_1 p_2 p_3 p_4 p_5 p_6 p_7

Compressed k^4-tree (ck^4-tree)

- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
    - Guarantee the information-theoretic lower bound.

y: \ x:	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



T = 1110 1111 1100 1010 0100 1100

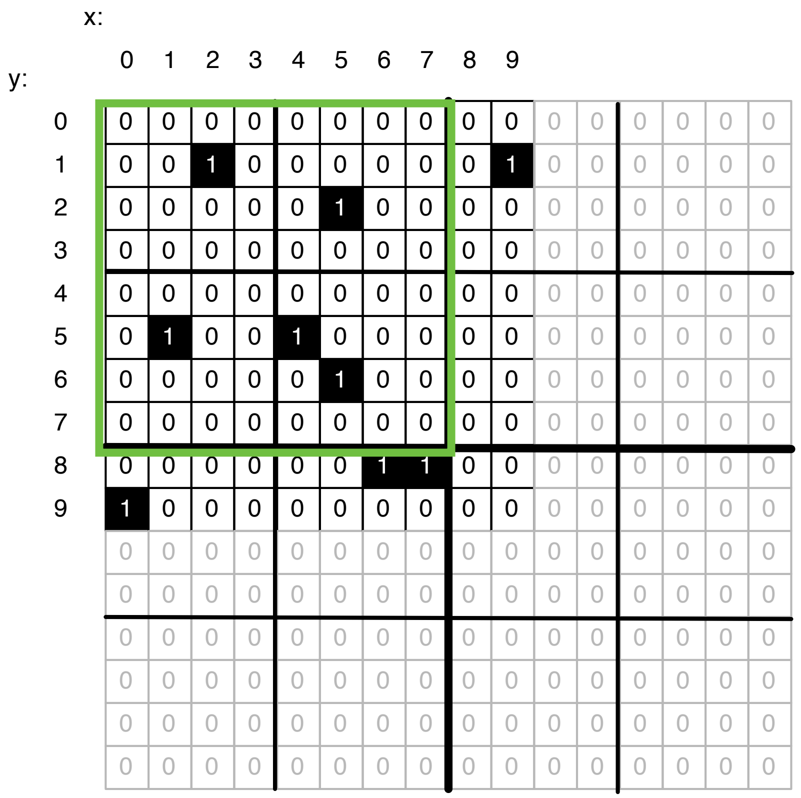
B = 010 1110 10 1 1 0 11

A = [(1',1'), (2',1'), (1', 2'), (1',1'), (0',1'), (0', 1') , (1',0')]

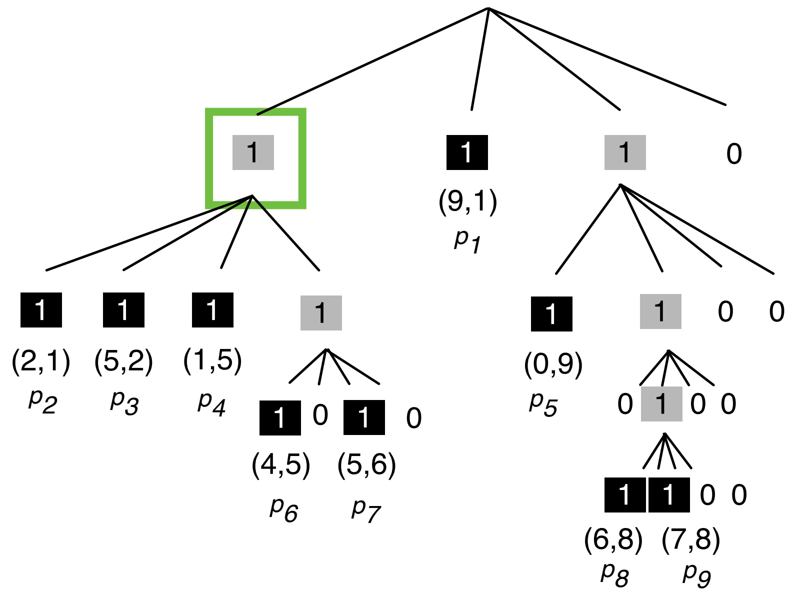
p_1 p_2 p_3 p_4 p_5 p_6 p_7

Compressed k^4-tree (ck^4-tree)

- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
  - Guarantee the information-theoretic lower bound.



y: \ x:	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



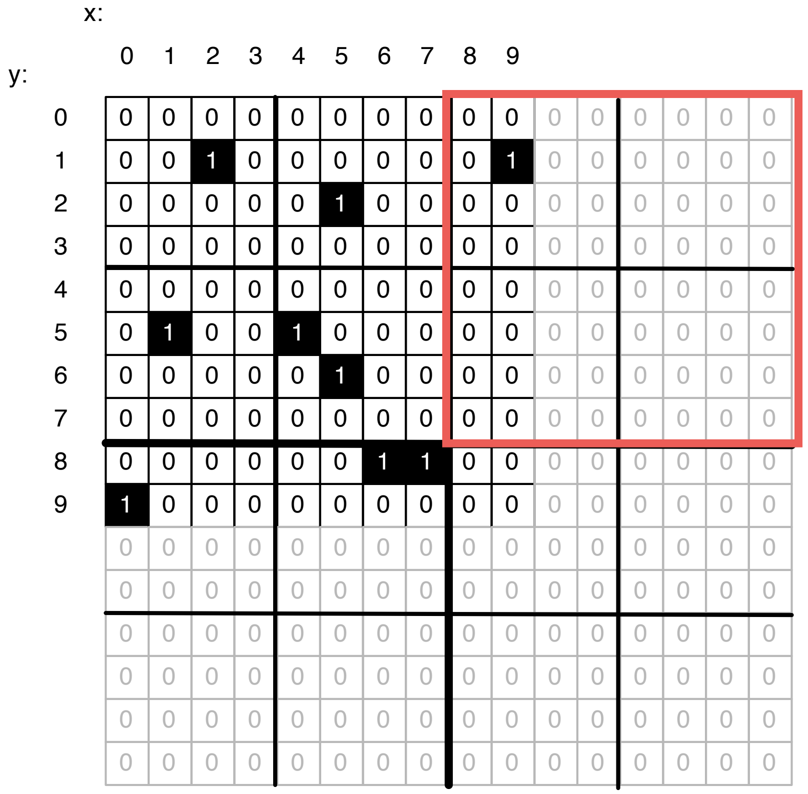
T = 1110 1111 1100 1010 0100 1100

B = 010 1110 10 1 1 0 11

A = [(1',1'), (2',1'), (1', 2'), (1',1'), (0',1'), (0', 1') , (1',0')]
p_1 p_2 p_3 p_4 p_5 p_6 p_7

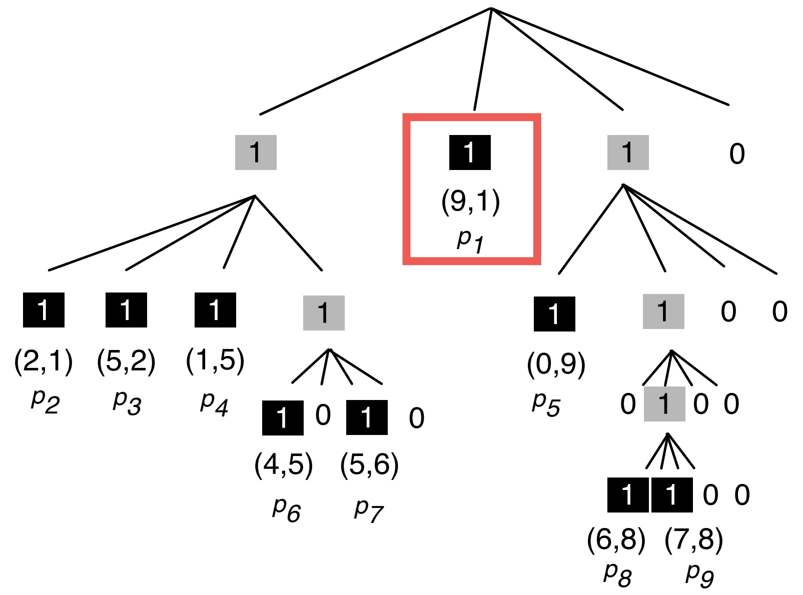
Compressed k^4-tree (ck^4-tree)

- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
    - Guarantee the information-theoretic lower bound.



x:

y:	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

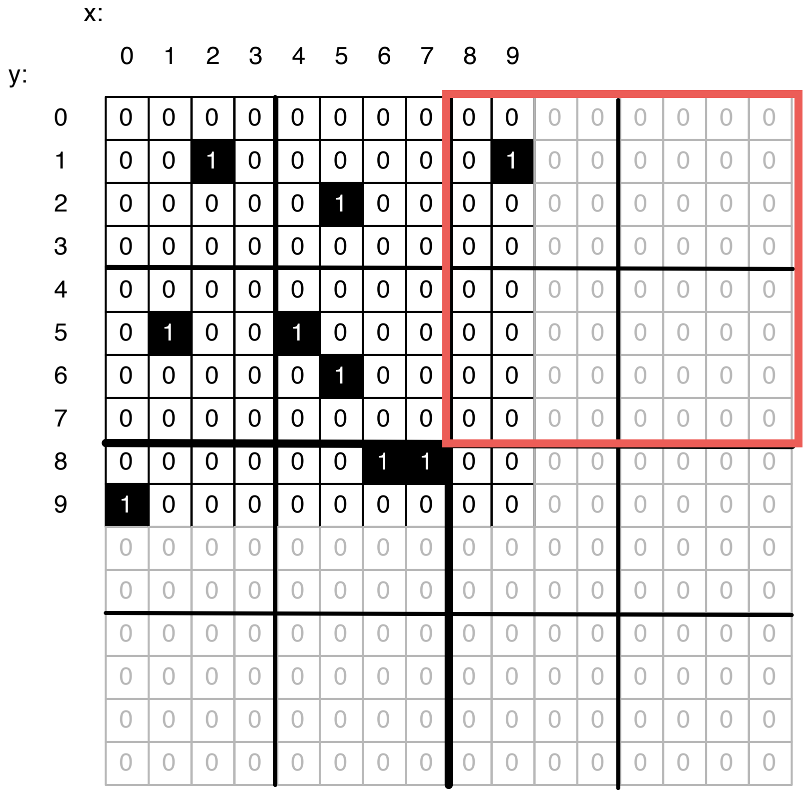


```
T = 1110 1111 1100 1010 0100 1100

B = 010  1110 10   1 1   0    11
```

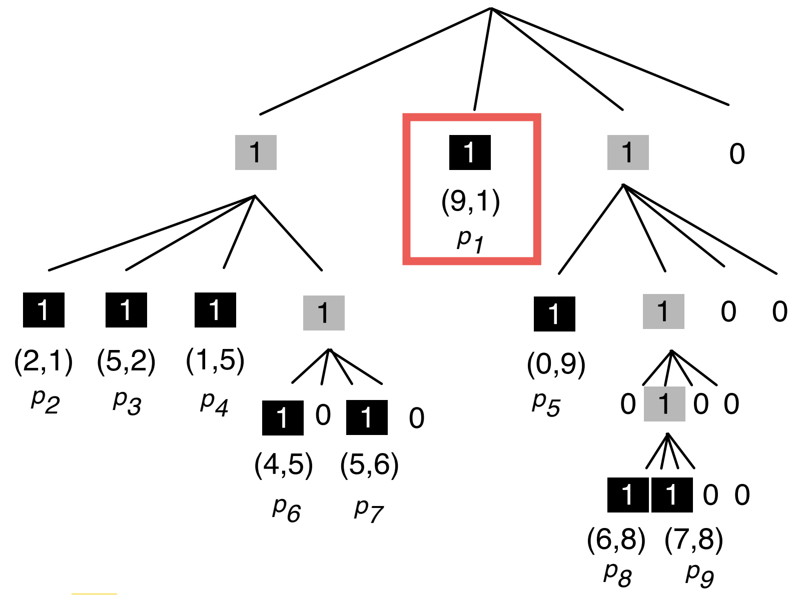
A = [(1',1'), (2',1'), (1', 2'), (1',1'), (0',1'), (0', 1') , (1',0')]
p_1 p_2 p_3 p_4 p_5 p_6 p_7

Compressed k^4-tree (ck^4-tree)

- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
  - Guarantee the information-theoretic lower bound.



y: \ x:	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



T = 1110 1111 1100 1010 0100 1100

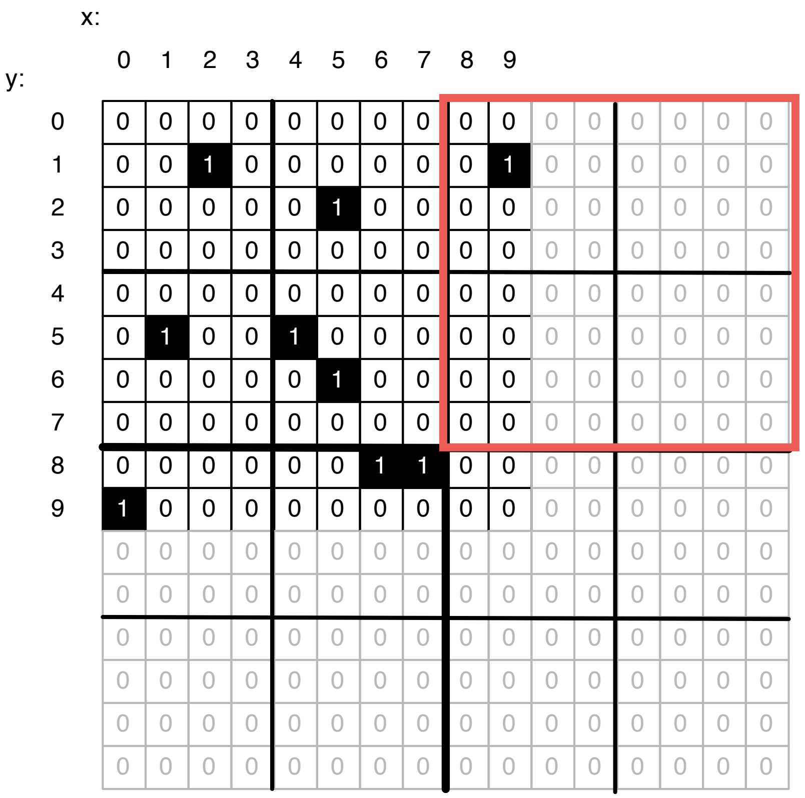
B = 010 1110 10 1 1 0 11

A = [(1',1'), (2',1'), (1', 2'), (1',1'), (0',1'), (0', 1') , (1',0')]

p_1 p_2 p_3 p_4 p_5 p_6 p_7

Compressed k^4-tree (ck^4-tree)

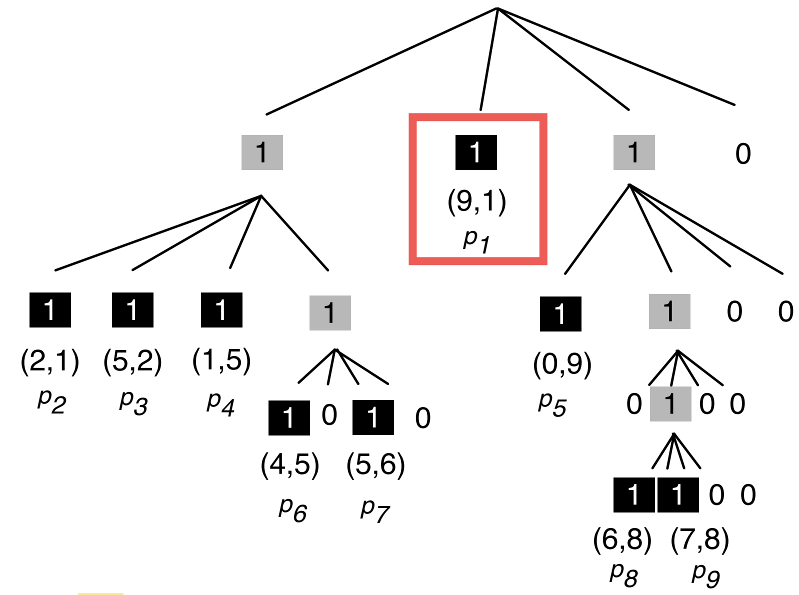
- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
    - Guarantee the information-theoretic lower bound.



x:

y:

	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

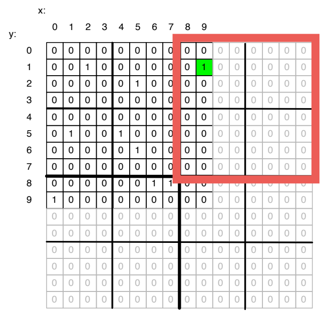


T = 1110 1111 1100 1010 0100 1100

B = 010 1110 10 1 1 0 11

A = [(1',1'), (2',1'), (1', 2'), (1',1'), (0',1'), (0', 1') , (1',0')]

p_1 p_2 p_3 p_4 p_5 p_6 p_7

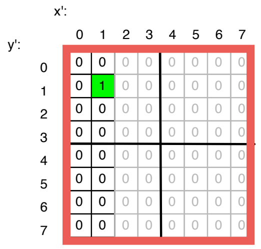


x=1
y=9

→

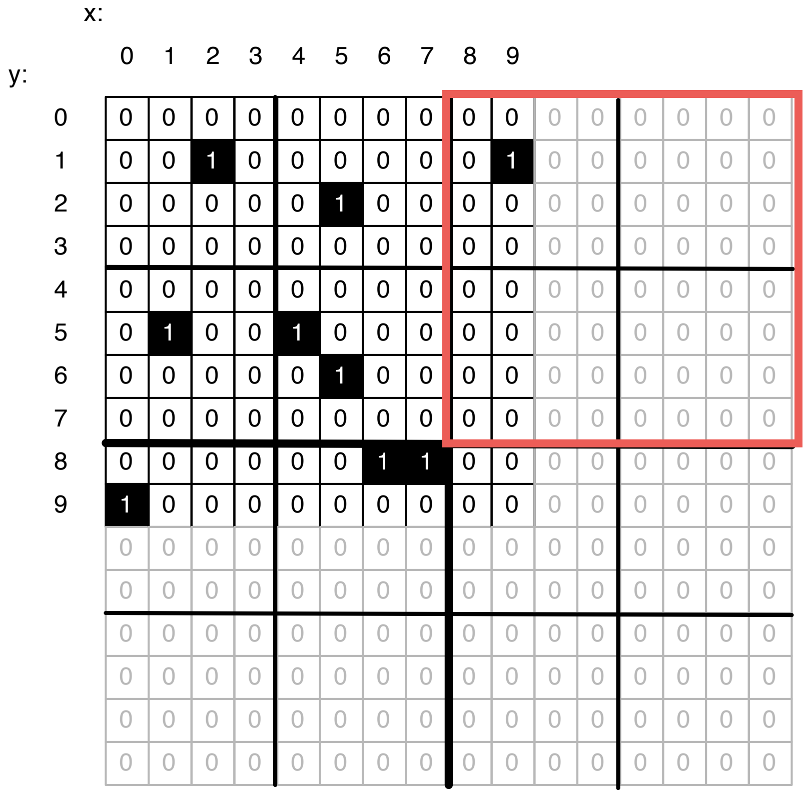
x'=1
y'=1

Position relative to submatrix

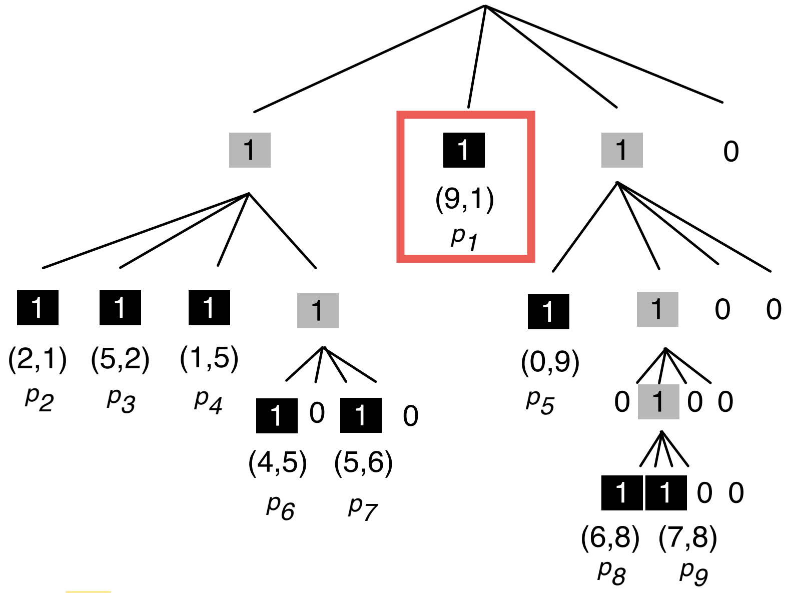


Compressed k^4-tree (ck^4-tree)

- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
    - Guarantee the information-theoretic lower bound.



y: \ x:	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



```
T = 1110 1111 1100 1010 0100 1100
B = 010  1110 10    1 1   0    11
```

A = [(1',1'), (2',1'), (1', 2'), (1',1'), (0',1'), (0', 1') , (1',0')]

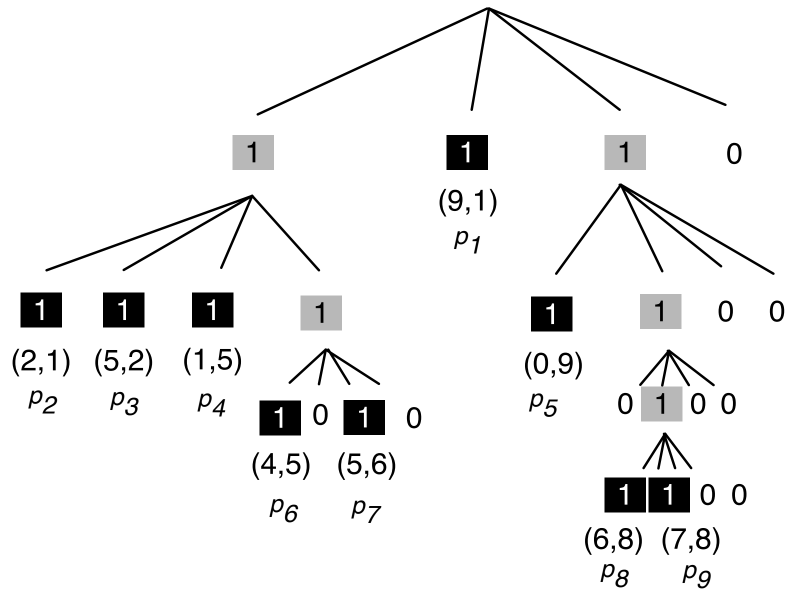
p_1 p_2 p_3 p_4 p_5 p_6 p_7

Compressed k^4-tree (ck^4-tree)

- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
  - Guarantee the information-theoretic lower bound.

x:

y:	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



T = 1110 1111 1100 1010 0100 1100

B = 010 1110 10 1 1 0 11

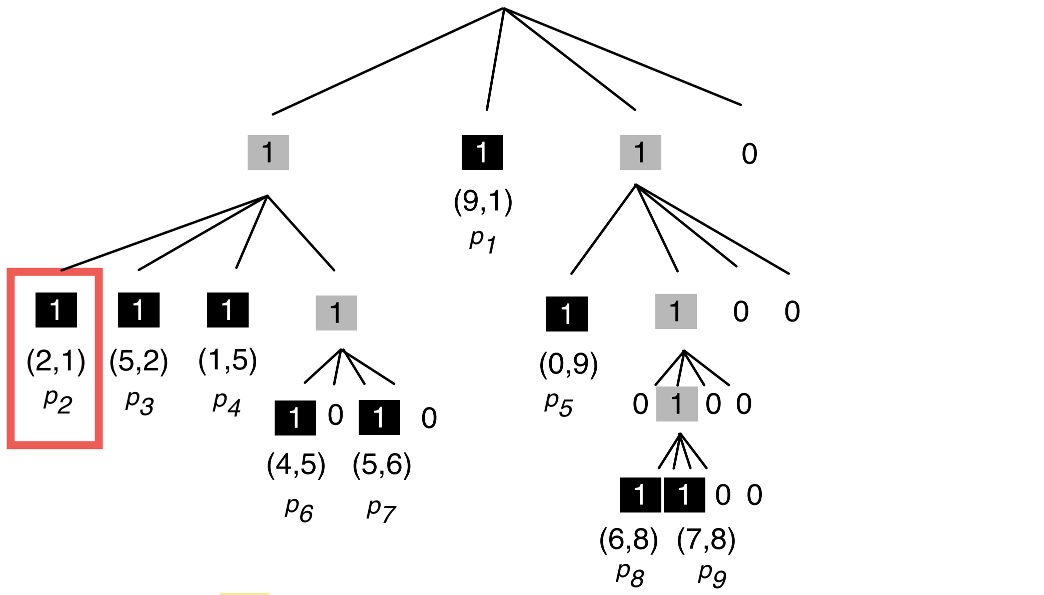
A = [$\underset{p_1}{(1',1')}$, $\underset{p_2}{(2',1')}$, $\underset{p_3}{(1', 2')}$, $\underset{p_4}{(1',1')}$, $\underset{p_5}{(0',1')}$, $\underset{p_6}{(0', 1')}$, $\underset{p_7}{(1',0')}$]

Compressed k^4-tree (ck^4-tree)

- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
  - Guarantee the information-theoretic lower bound.

x:

y:	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



T = 1110 1111 1100 1010 0100 1100

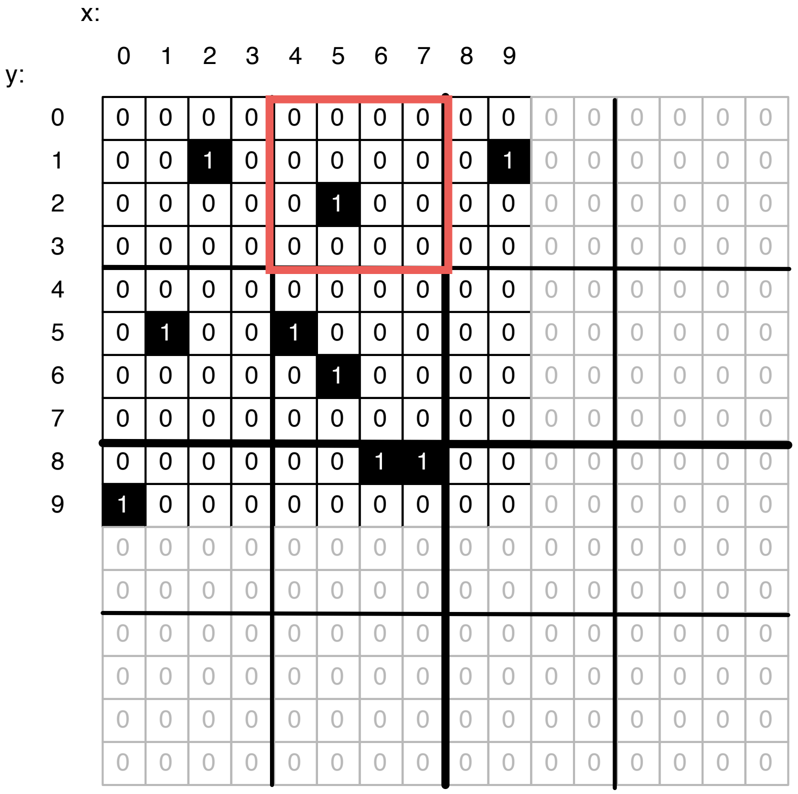
B = 010 1110 10 1 1 0 11

A = [(1',1'), (2',1') (1', 2'), (1',1'), (0',1'), (0', 1') , (1',0')]

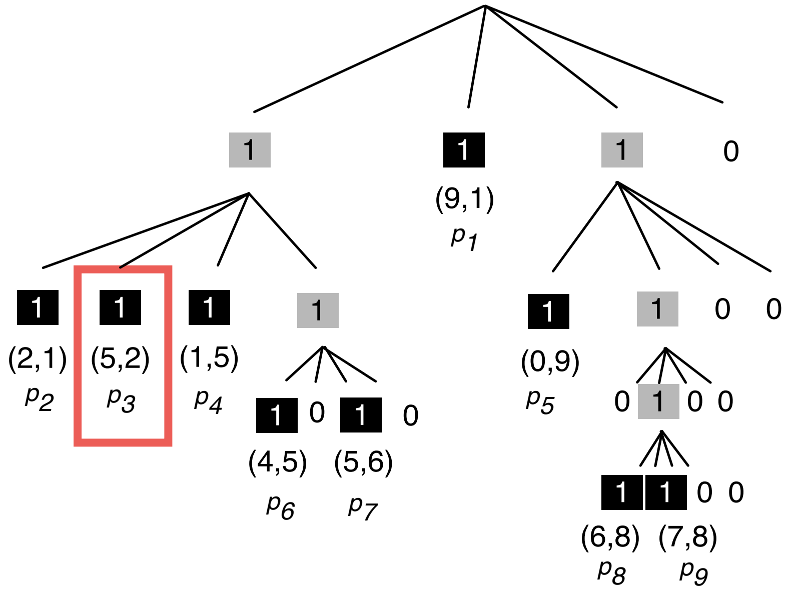
p_1 p_2 p_3 p_4 p_5 p_6 p_7

Compressed k^4-tree (ck^4-tree)

- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
    - Guarantee the information-theoretic lower bound.



y \ x	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



T = 1110 1111 1100 1010 0100 1100

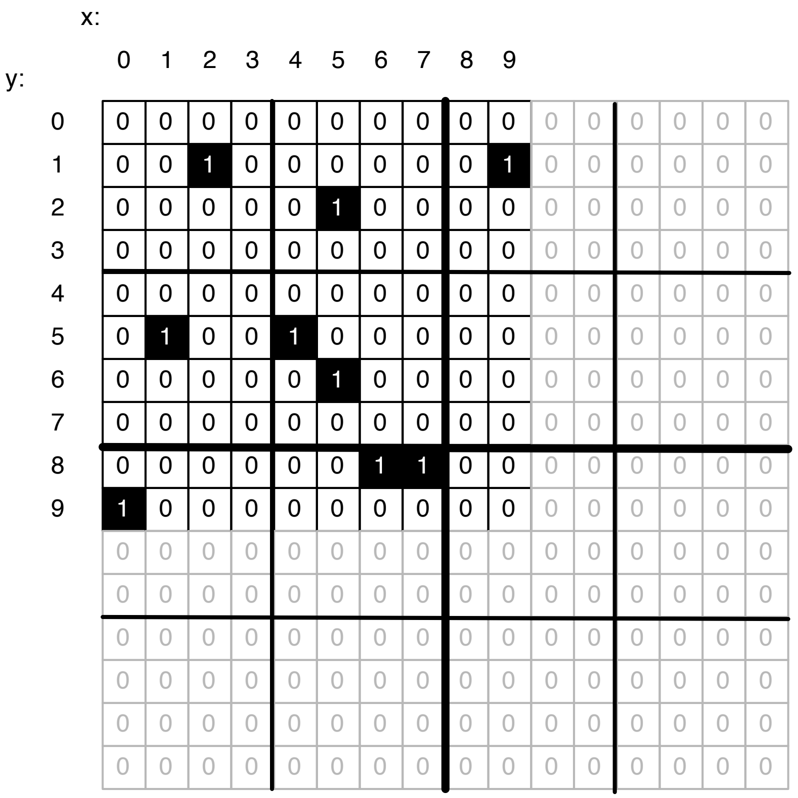
B = 010 1110 10 1 1 0 11

A = [(1',1'), (2',1'), (1', 2'), (1',1'), (0',1'), (0', 1') , (1',0')]

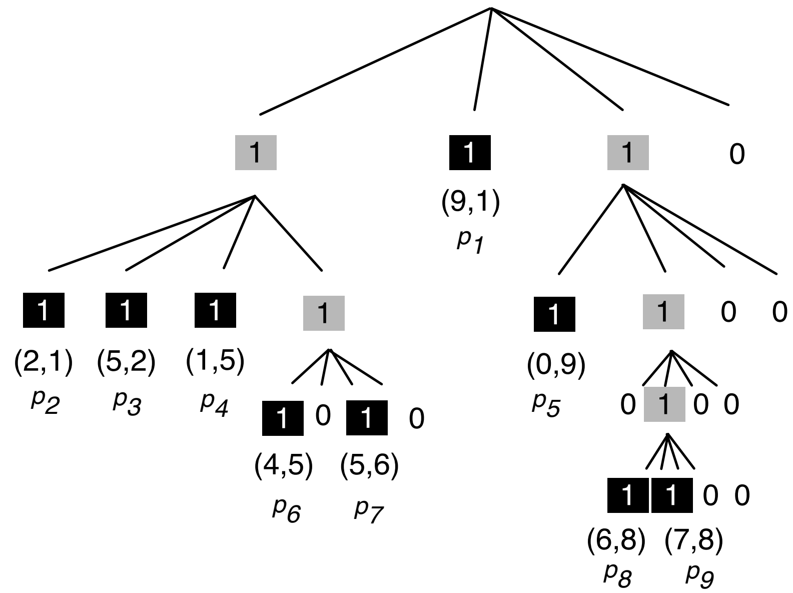
p_1 p_2 p_3 p_4 p_5 p_6 p_7

Compressed k^4-tree (ck^4-tree)

- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
  - Guarantee the information-theoretic lower bound.



y: \ x:	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



T = 1110 1111 1100 1010 0100 1100

B = 010 1110 10 1 1 0 11

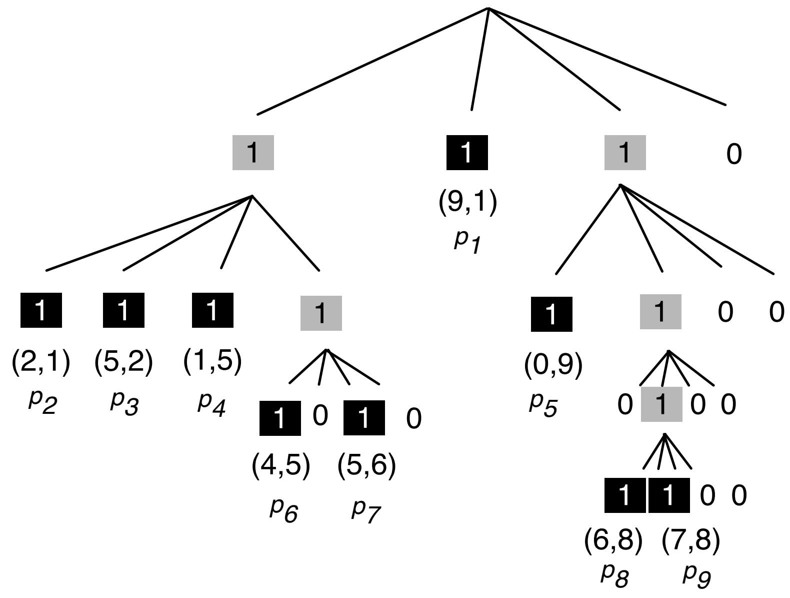
A = [(1',1'), (2',1'), (1', 2'), (1',1'), (0',1'), (0', 1') , (1',0')]

p_1 p_2 p_3 p_4 p_5 p_6 p_7

Compressed k^4-tree (ck^4-tree)

- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
  - Guarantee the information-theoretic lower bound.

y: \ x:	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



T = 1110 1111 1100 1010 0100 1100

B = 010 1110 10   1 1   0   11

A = [(1',1'), (2',1'), (1', 2'), (1',1'), (0',1'), (0', 1') , (1',0')]

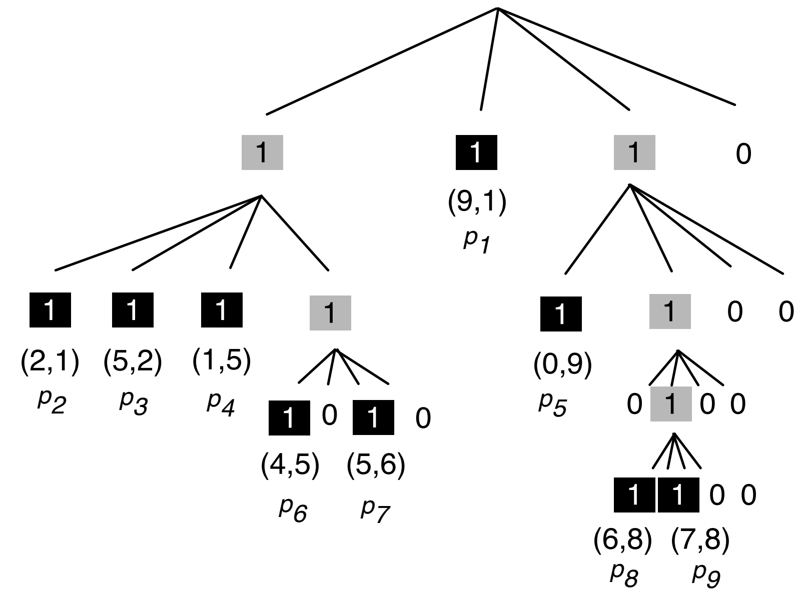
p_1 p_2 p_3 p_4 p_5 p_6 p_7

edge(ab,t=2)?

Compressed k^4-tree (ck^4-tree)

- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
  - Guarantee the information-theoretic lower bound.

y: \ x:	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



T = 1110 1111 1100 1010 0100 1100

B = 010 1110 10 1 1 0 11

A = [(1',1'), (2',1'), (1', 2'), (1',1'), (0',1'), (0', 1') , (1',0')]
p_1 p_2 p_3 p_4 p_5 p_6 p_7

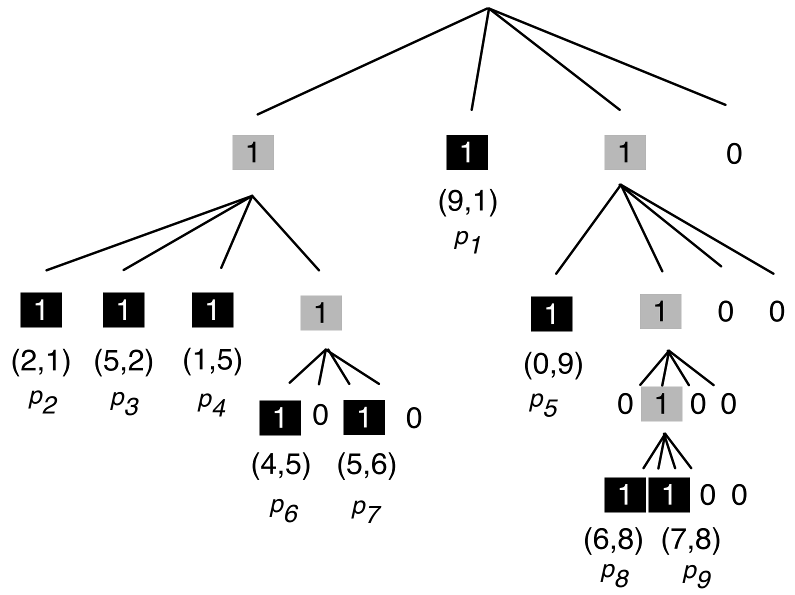
dirnei(a,t=2)?

Compressed k^4-tree (ck^4-tree)

- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
  - Guarantee the information-theoretic lower bound.

x:

y:	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



T = 1110 1111 1100 1010 0100 1100

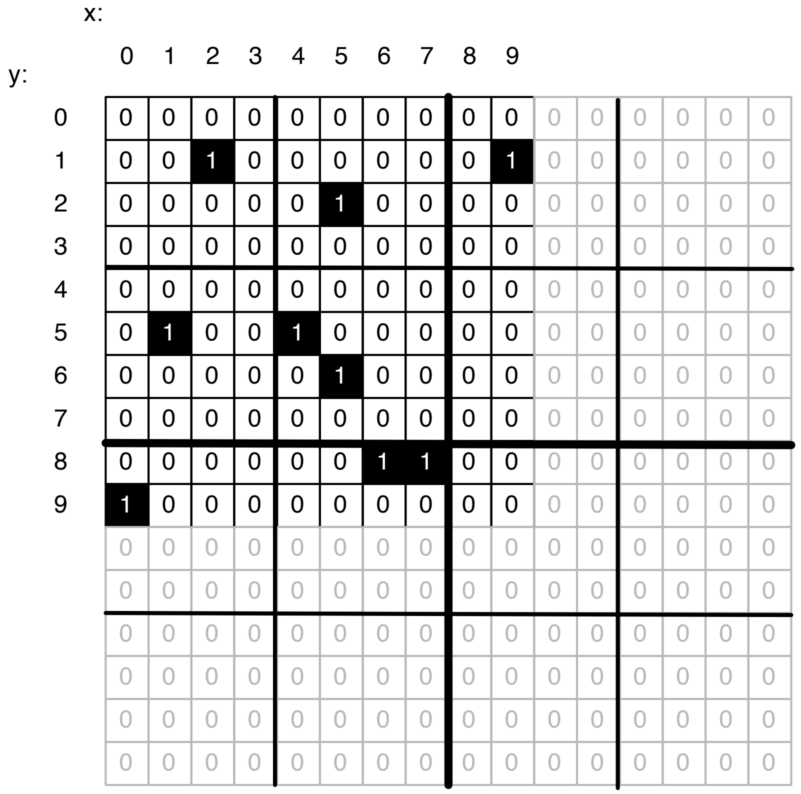
B = 010 1110 10   1 1   0   11

A = [(1',1'), (2',1'), (1', 2'), (1',1'), (0',1'), (0', 1') , (1',0')]
p_1 p_2 p_3 p_4 p_5 p_6 p_7

revnei(a,t=2)?

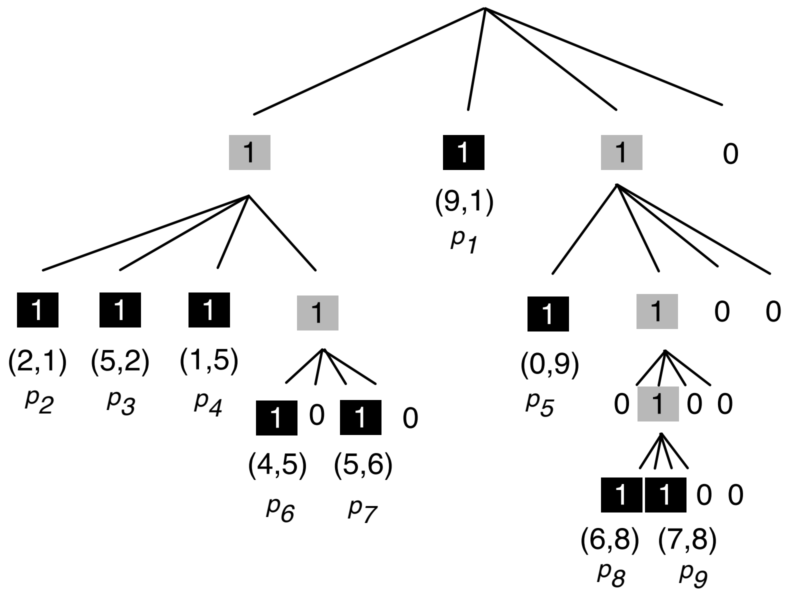
Compressed k^4-tree (ck^4-tree)

- Solution: stop the recursive decomposition when a sub matrix is empty or have one (isolated) cell.
    - Guarantee the information-theoretic lower bound.



x:

y:	0	1	2	3	4	5	6	7	8	9						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



T = 1110 1111 1100 1010 0100 1100

B = 010 1110 10 1 1 0 11

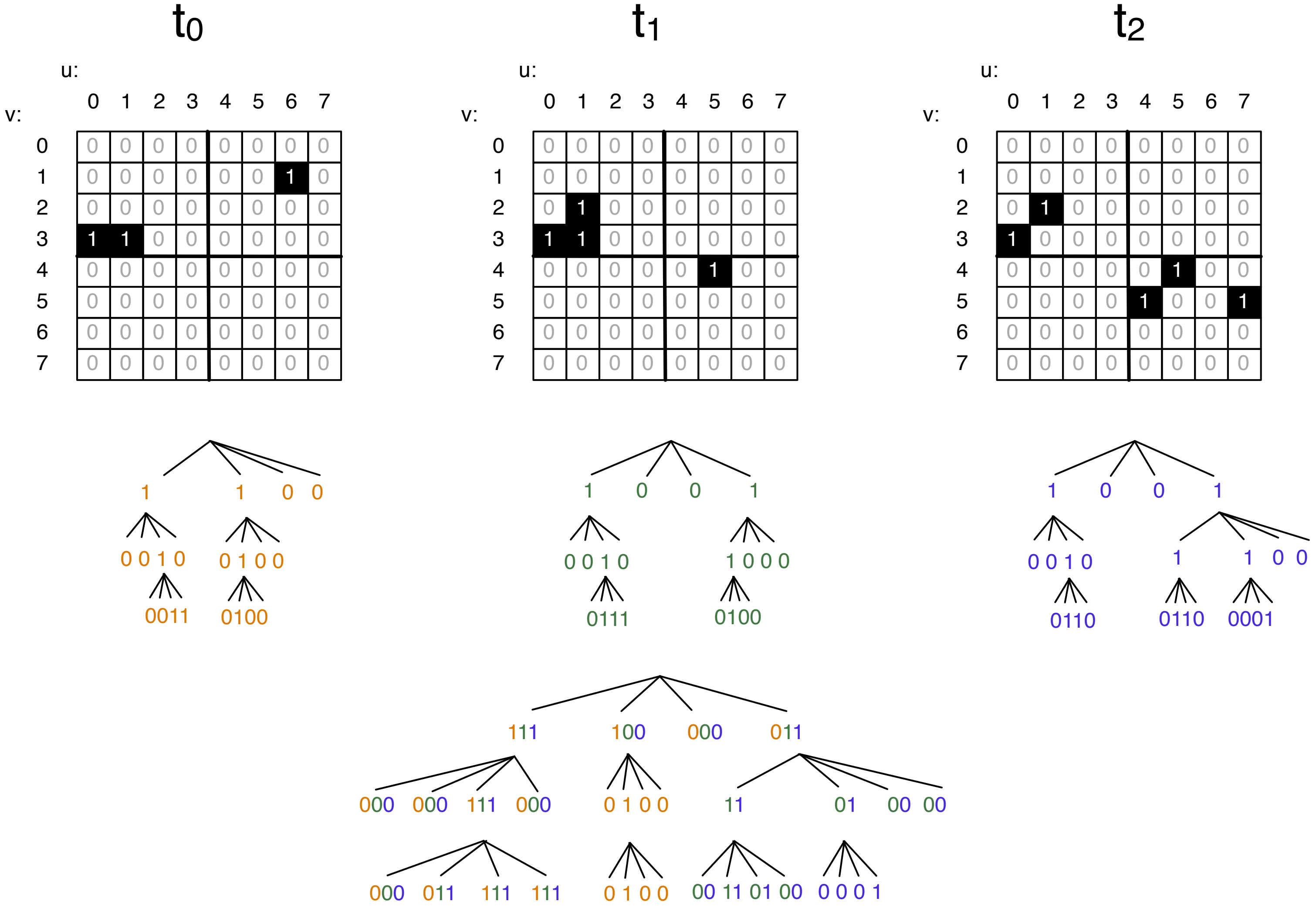
A = [$\underset{p_1}{(1',1')}$, $\underset{p_2}{(2',1')}$, $\underset{p_3}{(1', 2')}$, $\underset{p_4}{(1',1')}$, $\underset{p_5}{(0',1')}$, $\underset{p_6}{(0', 1')}$, $\underset{p_7}{(1',0')}$]

Outline

- ✓ Definition and Motivation.
- ✓ Previous works about temporal graphs.
- ✓ Compression of temporal graphs.
- ✓ Contributions.
- Evaluation.
  - Baselines.
  - Datasets.
  - Space evaluation.
  - Time evaluation.
- Conclusions and future works.

Baselines

- We will compare our structures against two baselines:
  - The k^4-tree representing contacts in 4D with compressed bitmaps.
  - The Interleaved k^2-tree (DCC' 2014):
    - Events stored in binary matrices are stored as k^2-trees.
    - Final data structure is the merge of all k^2-trees.



t_0

u:

v:	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

t_1

u:

v:	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0
4	0	0	0	0	0	1	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

t_2

u:

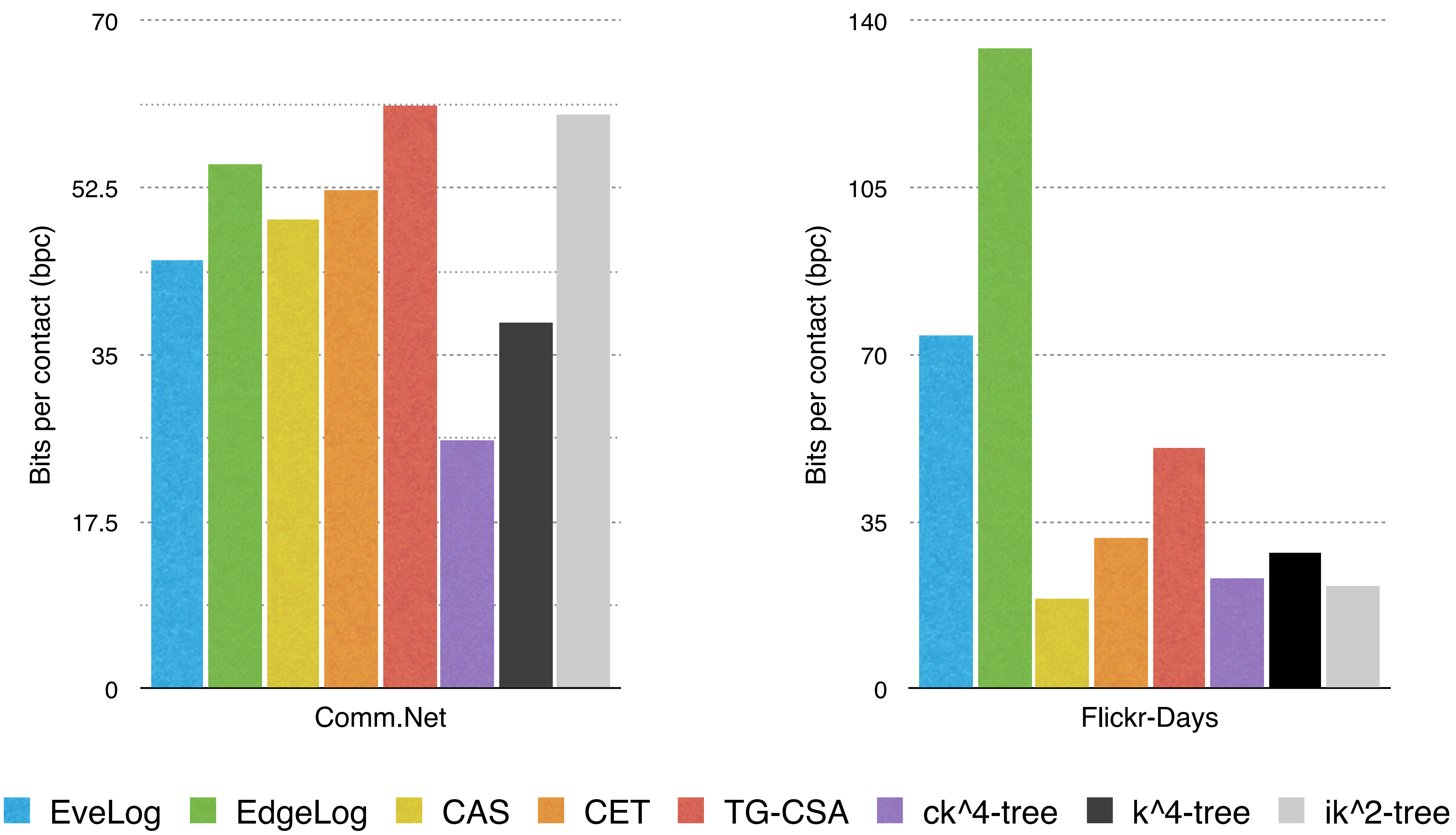
v:	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0
4	0	0	0	0	0	1	0	0
5	0	0	0	0	1	0	0	1
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

Datasets

- Experimental settings
  - Synthetic: **Comm.Net**, Powerlaw
  - Wikipedia: WikiLinks, WikiEdit
  - Social Networks: **FlickrDays**, FlickrSecs
  - Communications: YahooNetflow
  - Web Search: YahooSessions

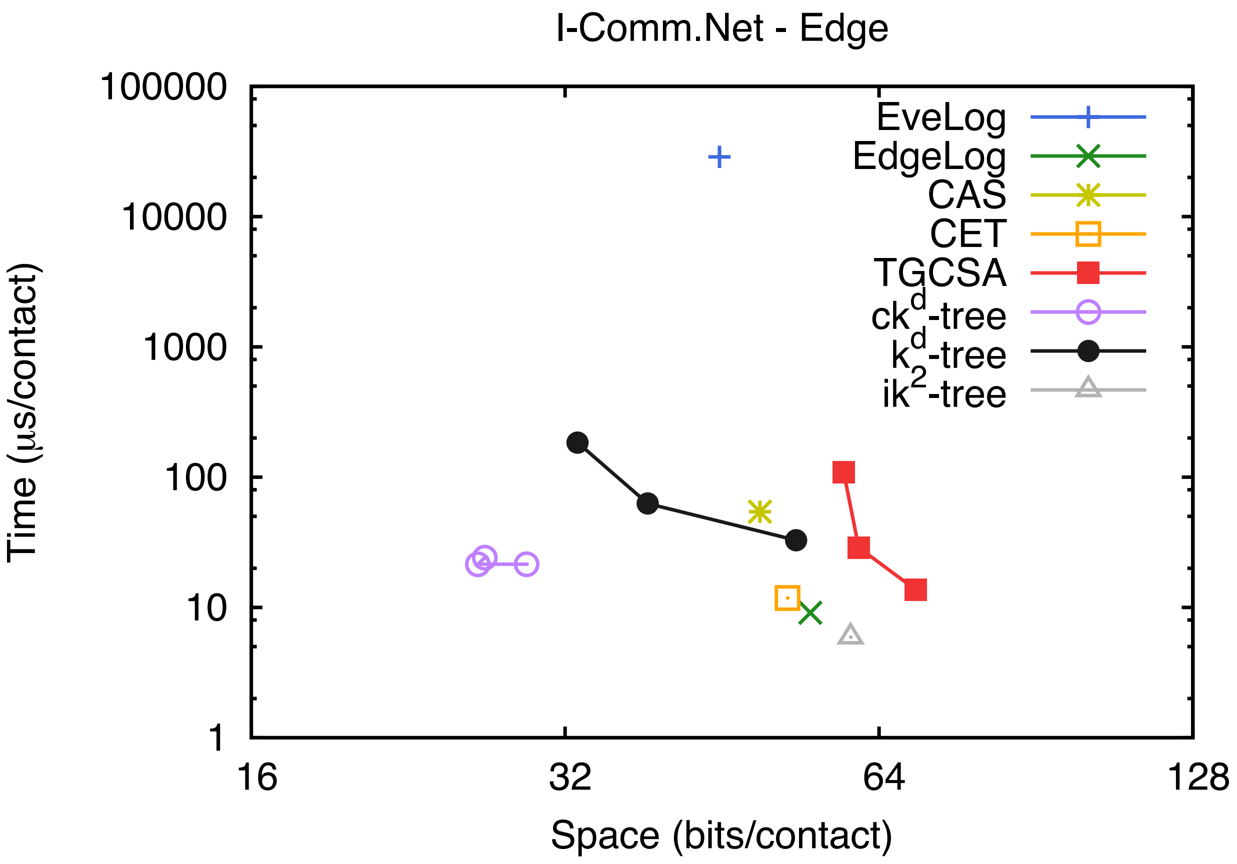
Name	Vertices	Edges	Lifetime	Contacts
Comm.Net	10000	15M	10000	19M
Flickr-Days	2M	33M	135	33M

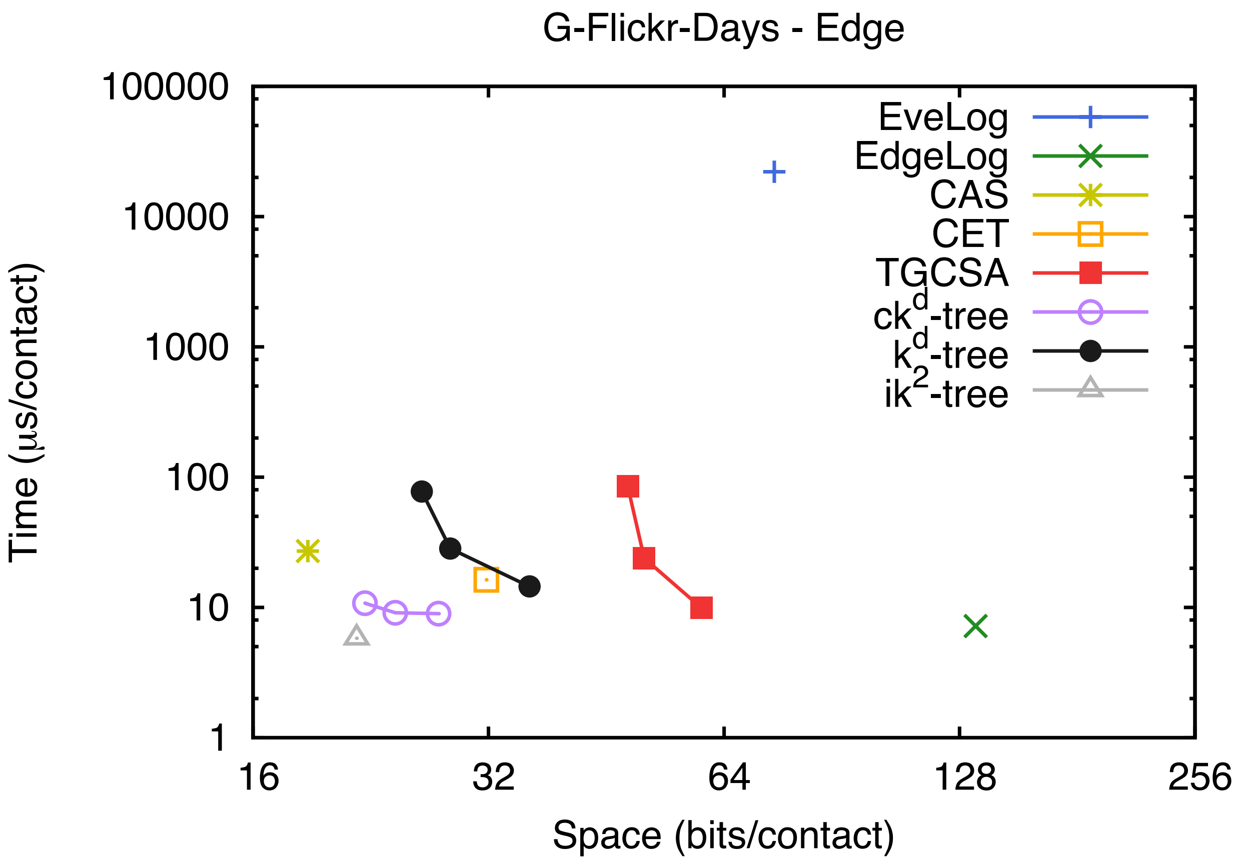
Space Evaluation



Operations about edges

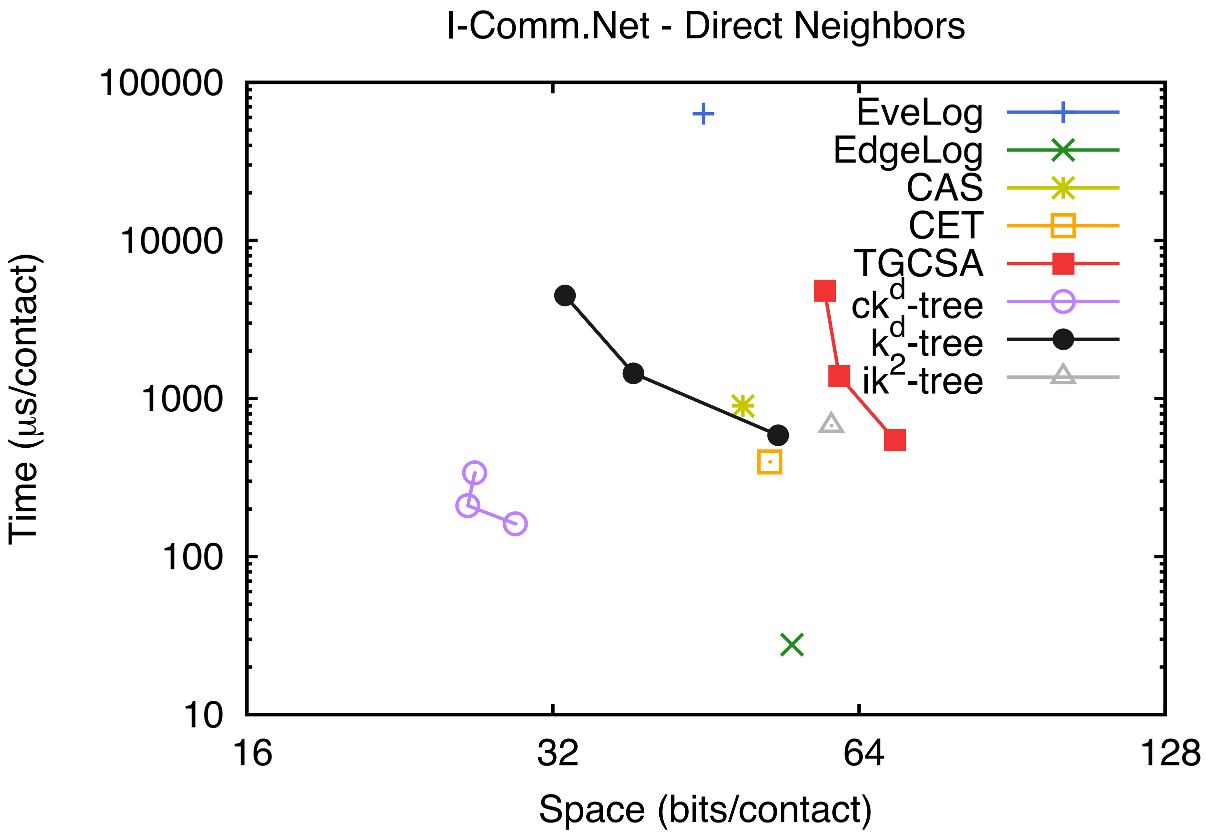
- Return true if the edge uv is active at time t, or false otherwise.
  - Were X and Y friends during the last year?
  - Does Y call X yesterday?

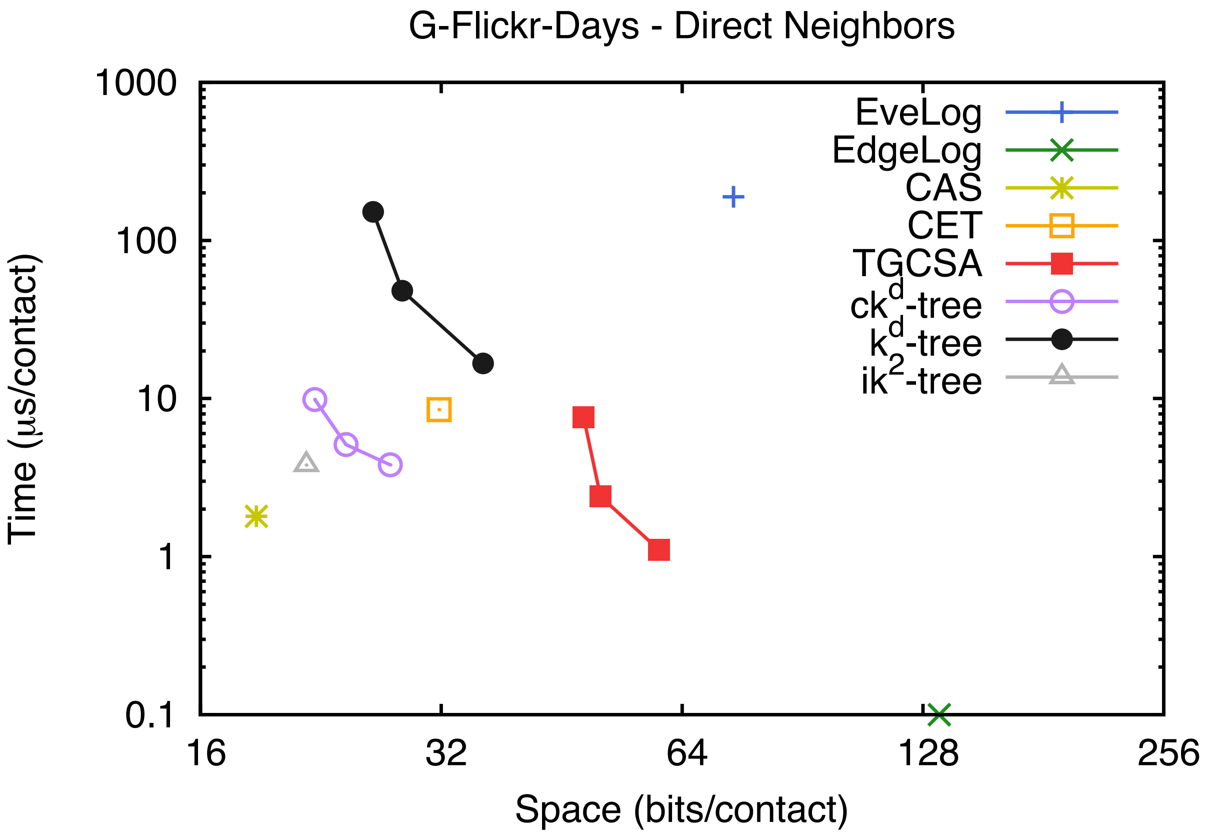




Operations about vertices

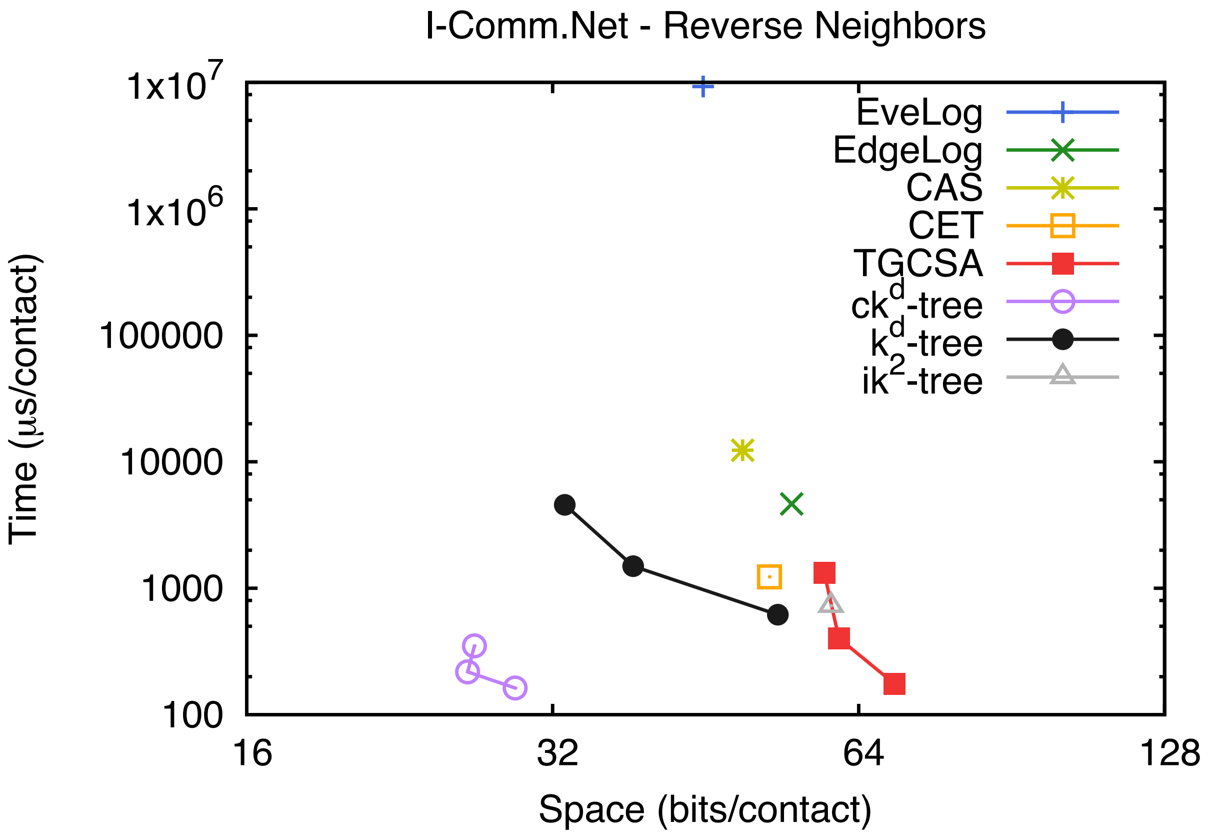
- Retrieve direct and reverse active neighbors of a vertex constrained by a time instant.
    - Who were friends of X during the last year?
    - Who are the telephone numbers called by the number X yesterday?

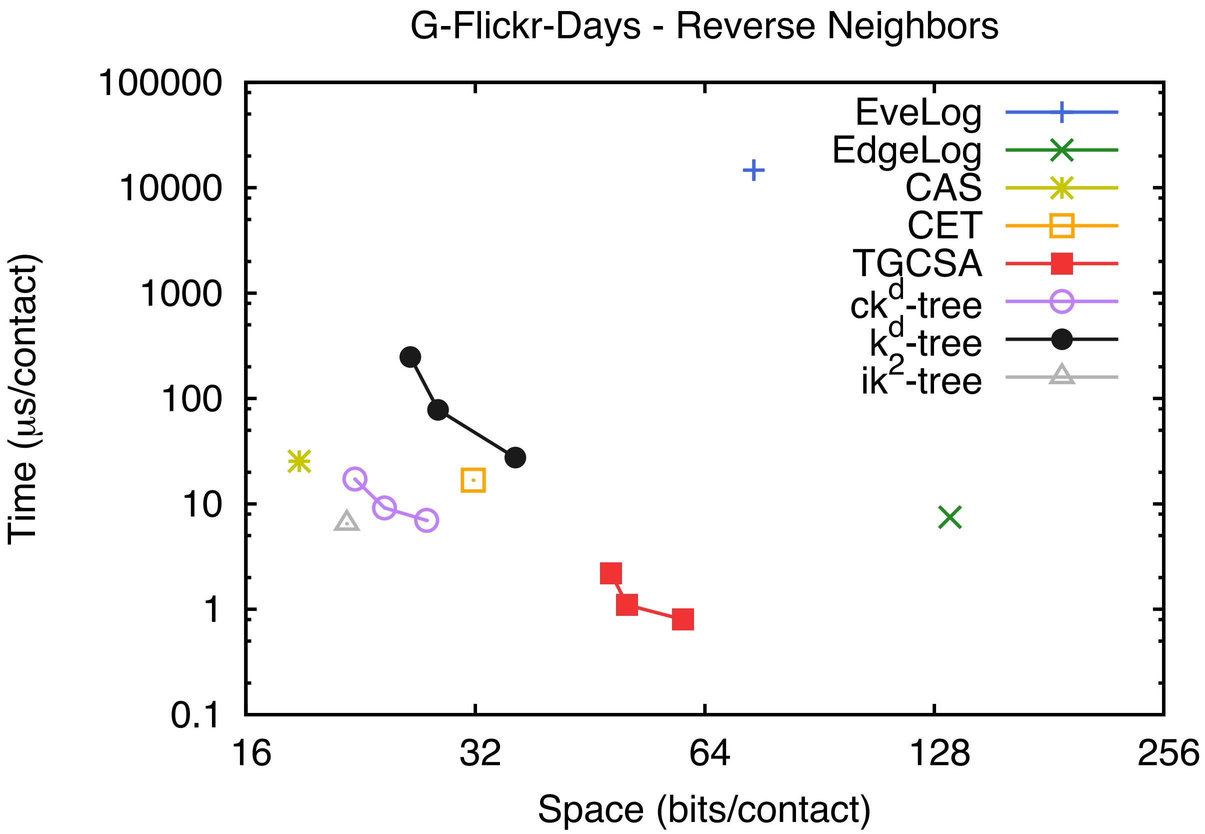

I-Comm.Net - Direct Neighbors


G-Flickr-Days - Direct Neighbors

Operations about vertices

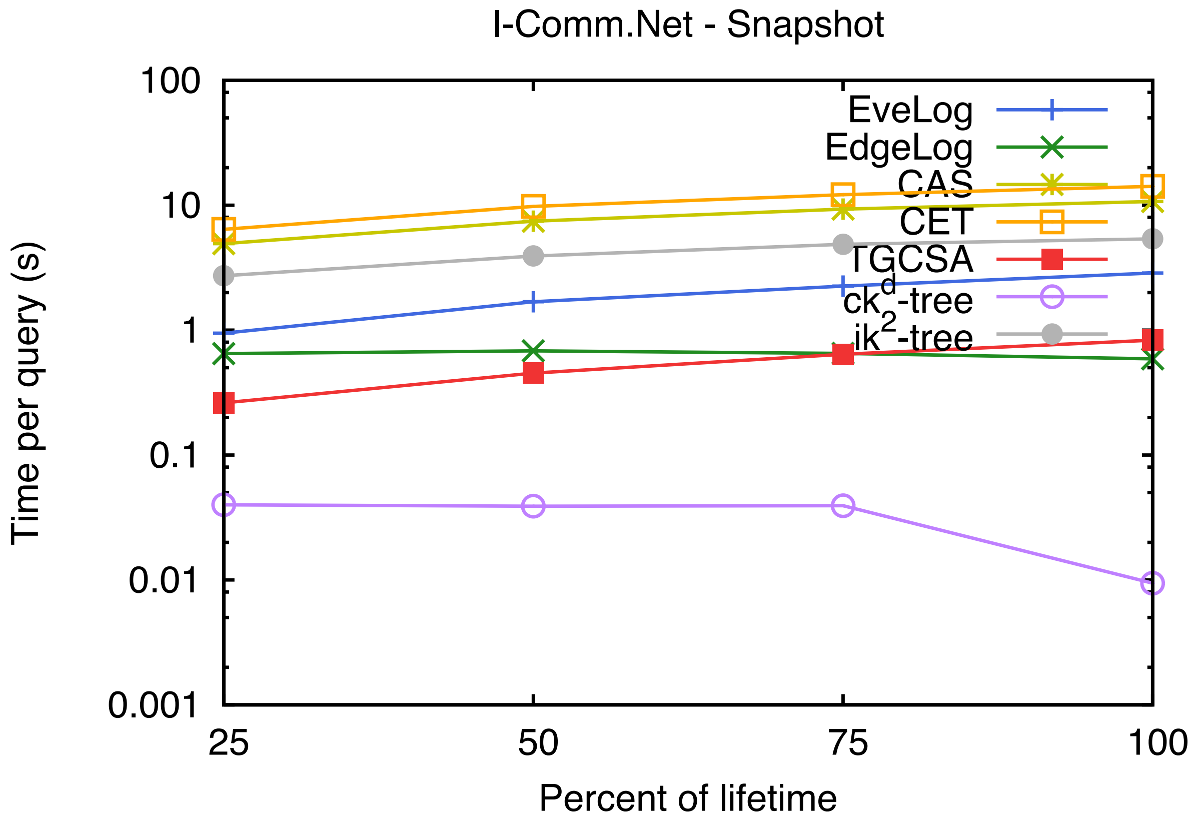
- Retrieve direct and reverse active neighbors of a vertex constrained by a time instant.
  - Who were friends of X during the last year?
  - Who are the telephone numbers called by the number X yesterday?

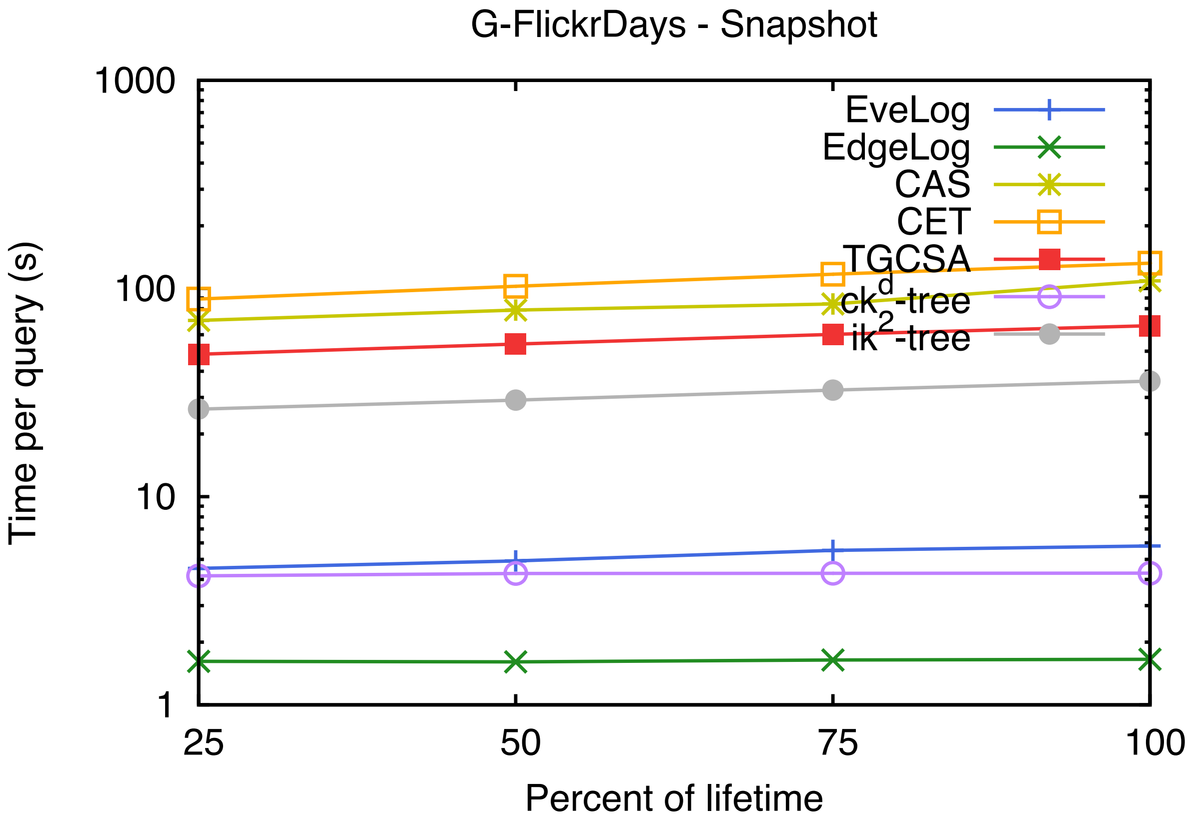




Operations about the state of the graph

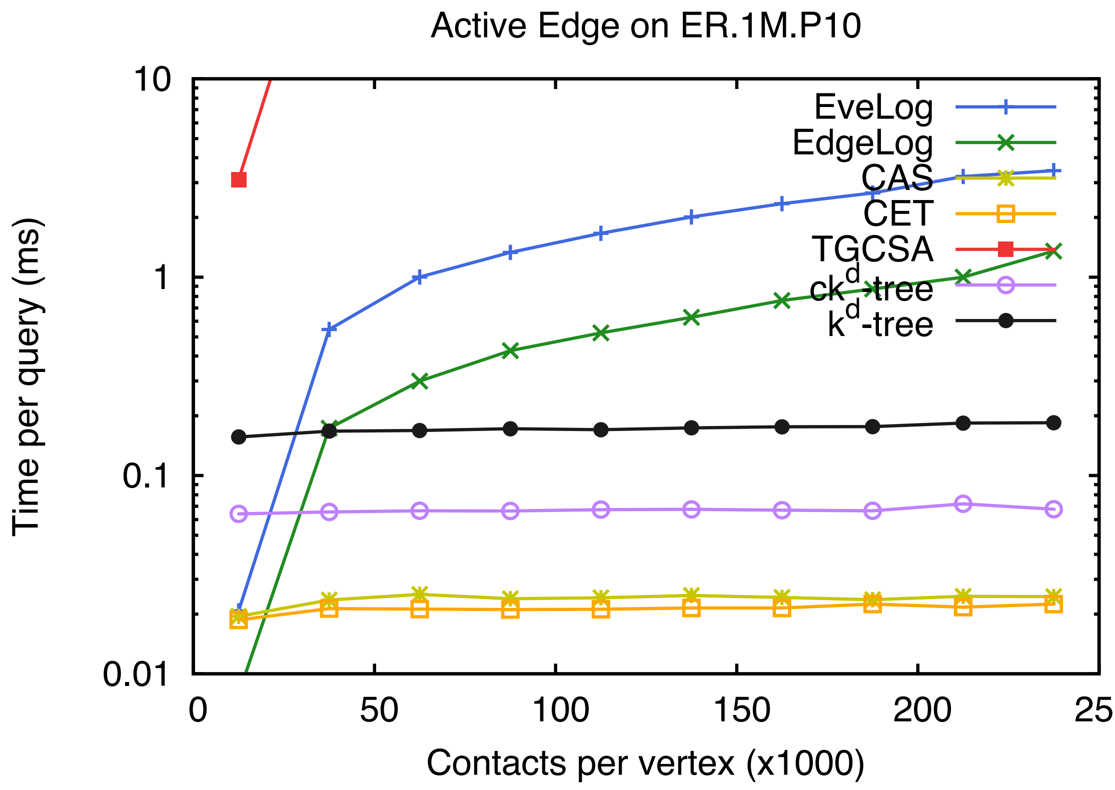
- Retrieve all active edges at time instant t.
  - Which in-air flights (between a pair of cities) were at 4:30 am?
  - Which pairs of numbers were connected by a call at 9 am?
- Time instants: 25%, 50%, 75% and 100% of the lifetime.

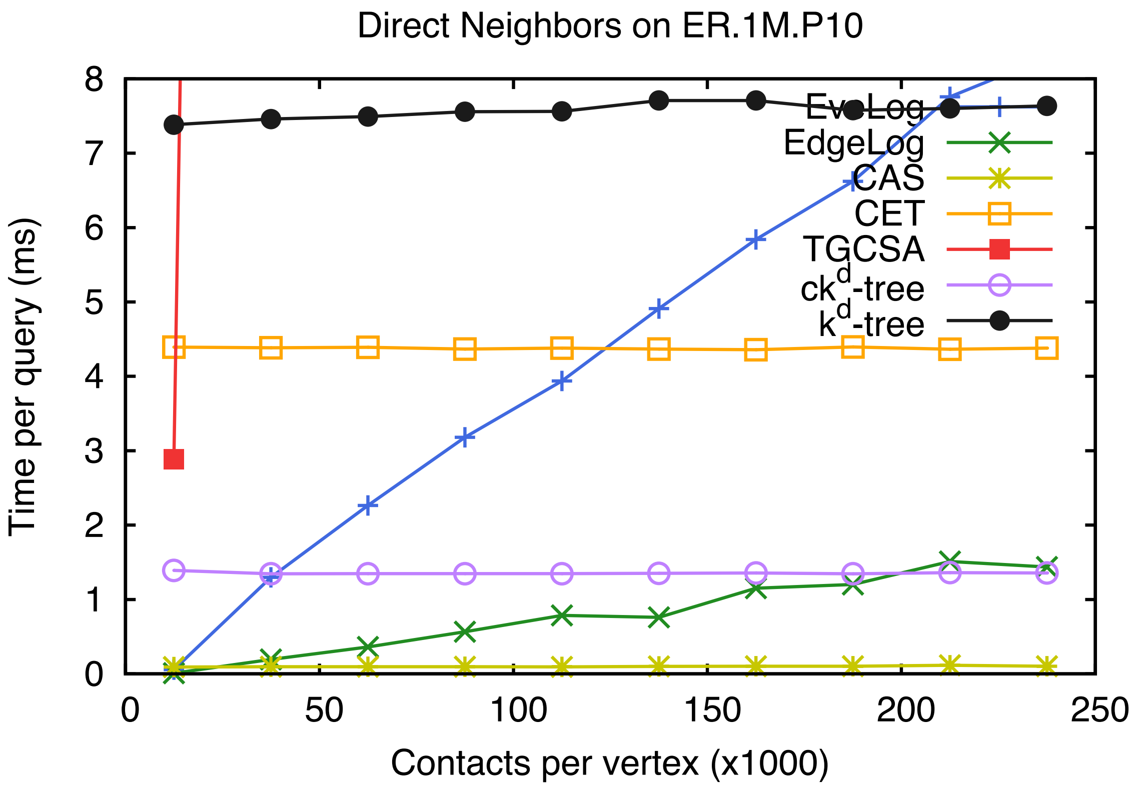




Other evaluations

- Do the number of contacts per vertex affect the time performance?
- Evaluated using a synthetic dataset:
  - Aggregated graph with a uniform degree distribution.
  - A variable distribution in the number of contacts per edge.





Outline

- ✓ Definition and Motivation.
- ✓ Previous works about temporal graphs.
- ✓ Compression of temporal graphs.
- ✓ Contributions.
- ✓ Evaluation.
- Conclusions and future works.

Conclusions

- Our main goal: to design new compact data structures for temporal graphs.
    - All structures reduce space. However, not all are good for all operations.
- Time performance in CAS, CET and ck^d-tree do not depend on the number of contacts per vertex.
- EdgeLog and EveLog:
    - EdgeLog is very fast when there are few contacts per edge, but require space.
    - EveLog uses less space than EdgeLog, but is slow.
- CAS and CET:
    - CAS is faster than CET in operations about edges and direct vertices.
    - CET answer direct and reverse neighbors in the same time, and also answer operations about events.

Conclusions

- TG-CSA
  - Best space when used in incremental graphs.
  - Good performance in all operations, but is highly dependent on the number of contacts per edge (**WIP**).
- ck^d-tree
  - Ensures a space close to the information-theoretic lower bound.
  - Less dependent on the number of contacts per edge.

Future work

- Regarding temporal graphs:
  - To explore more encodings of Ψ in the TG-CSA, it uses 80%-90% of the space.
  - To explore the compression of leaves in the ck^d-tree.
  - To evaluate techniques used to improve space in Web and Social Graphs such as node orderings and representations of bicliques.
  - To explore the usage of the data structures for computing temporal metrics and spatio-temporal paths.
- Regarding data structures:
  - To evaluate the performance of the ck^d-tree in other domains such as the representations of RDF triples and evolving raster data.
  - To explore the use of the Interleaved Wavelet Tree for representing binary relations and point grids.
  - To extend the work of compressed multidimensional data structures to kd-trees and range trees.

Publications and other results

- Journal articles:
  - D. Caro, M. A. Rodríguez, and N. R. Brisaboa, “Data structures for temporal graphs based on compact sequence representations,” Information Systems, vol. 51, pp. 1–26, Jul. 2015.
  - D. Caro, M. A. Rodríguez, and N. R. Brisaboa, “Compressed k^d-tree for temporal graphs,”. Submitted to Knowledge and Information Systems.
  - N. R. Brisaboa, D. Caro, A. Fariña, and M. A. Rodríguez, “A Compressed Suffix- Array Strategy for Temporal-Graph Indexing”. (Work-In-Progress)
- International Conferences:
  - N. R. Brisaboa, D. Caro, A. Fariña, and M. A. Rodríguez, “A Compressed Suffix- Array Strategy for Temporal-Graph Indexing,” presented at the 21st International Symposium on String Processing and Information Retrieval, Ouro Preto, Brazil, 2014, vol. 8799, pp. 77–88.
  - G. D. Bernardo, N. R. Brisaboa, D. Caro, and M. A. Rodriguez, “Compact Data Structures for Temporal Graphs,” presented at the Data Compression Conference (DCC), 2013, p. 477.
- International Workshops:
  - D. Caro, “A compressed hexatree for temporal-graph indexing... or how to compress the k^4-tree,” presented at WCTA 2014, Ouro Preto, Brazil.
- All implementations are available as free software:
  - https://github.com/diegocaro/temporalgraphs

Thank you!

¡Gracias!