Fast and Accurate Triangle Counting in Graph Streams Using Predictions

ScalPerf'24

September, 2024

Cristian Boldrin
Dept. of Information Engineering
University of Padova, Italy
cristian.boldrin.2@phd.unipd.it

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITAS STUDII PADUANI
MCCXXII

Problem Definition

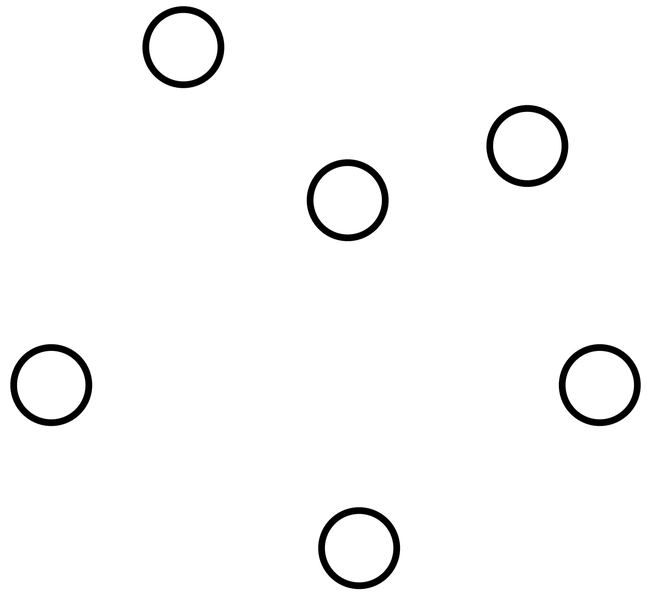
Problem: count the number of triangles in graphs.

Problem Definition

Problem: count the number of triangles in graphs.

Given:

- a set V of nodes, $\left|V\right| = n$

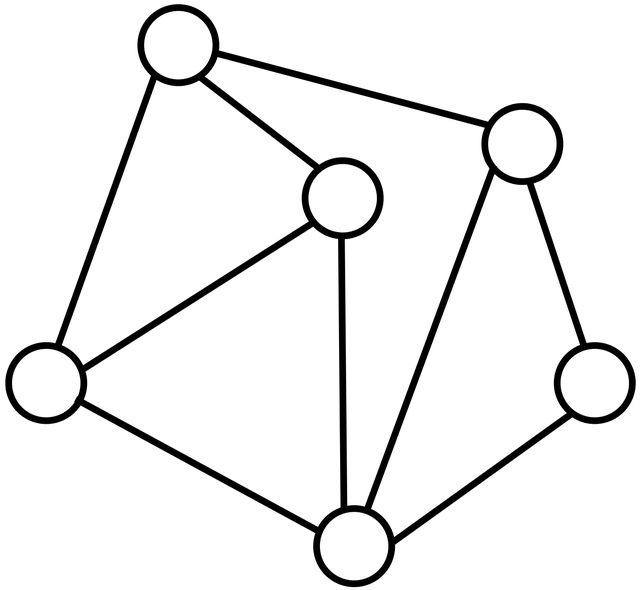


Problem Definition

Problem: count the number of triangles in graphs.

Given:

- a set V of nodes, $\left|V\right| = n$
- A set E of edges, $\left|E\right| = m$



Problem Definition

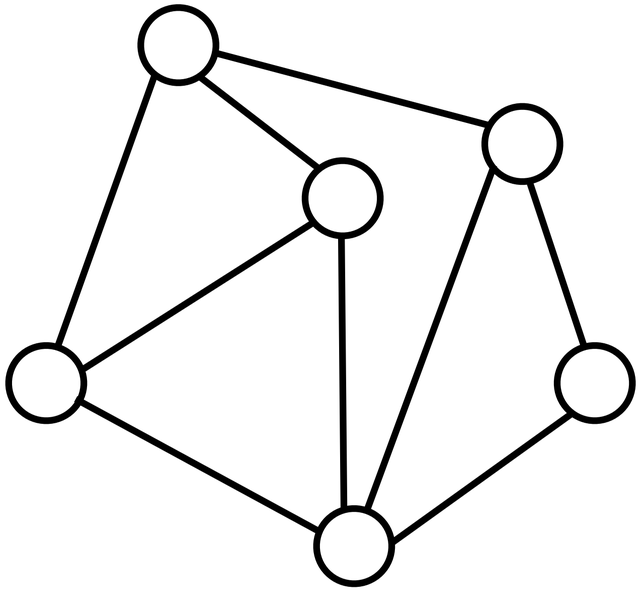
Problem: count the number of triangles in graphs.

Given:

- a set V of nodes, $\left|V\right| = n$
- A set E of edges, $\left|E\right| = m$

Goal:

- Count the **global** number of triangles $\Delta = \{u, v, w\}$, where $\{u, v\}$, $\{w, u\}$, and $\{v, w\}$ are all in the set E of the edges.



Problem Definition

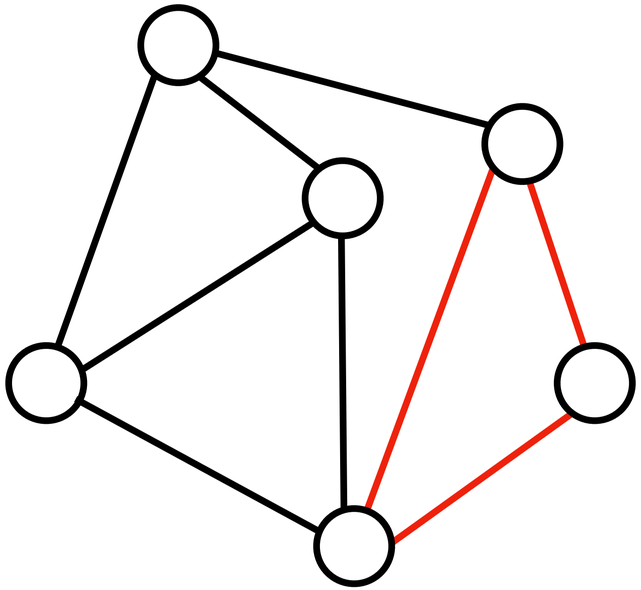
Problem: count the number of triangles in graphs.

Given:

- a set V of nodes, $\left|V\right| = n$
- A set E of edges, $\left|E\right| = m$

Goal:

- Count the **global** number of triangles $\Delta = \{u, v, w\}$, where $\{u, v\}$, $\{w, u\}$, and $\{v, w\}$ are all in the set E of the edges.



Problem Definition

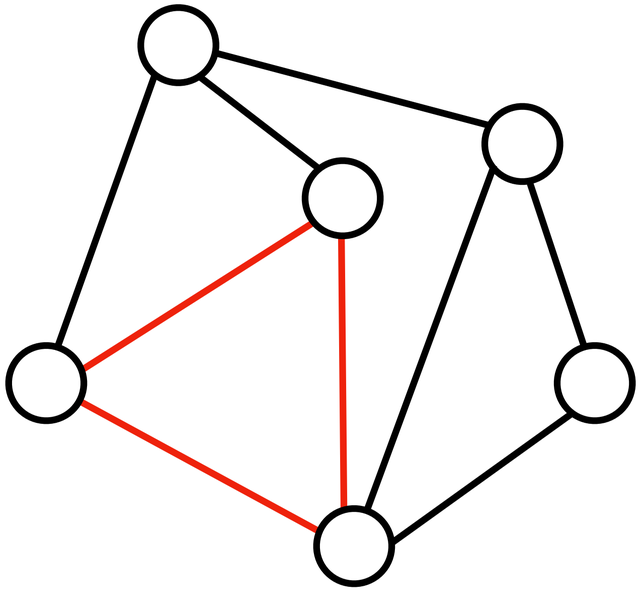
Problem: count the number of triangles in graphs.

Given:

- a set V of nodes, $\left|V\right| = n$
- A set E of edges, $\left|E\right| = m$

Goal:

- Count the **global** number of triangles $\Delta = \{u, v, w\}$, where $\{u, v\}$, $\{w, u\}$, and $\{v, w\}$ are all in the set E of the edges.



Problem Definition

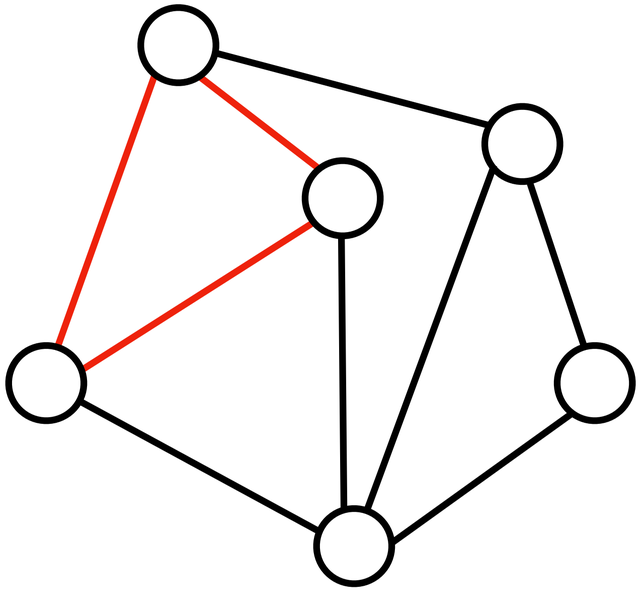
Problem: count the number of triangles in graphs.

Given:

- a set V of nodes, $\left|V\right| = n$
- A set E of edges, $\left|E\right| = m$

Goal:

- Count the **global** number of triangles $\Delta = \{u, v, w\}$, where $\{u, v\}$, $\{w, u\}$, and $\{v, w\}$ are all in the set E of the edges.



Problem Definition

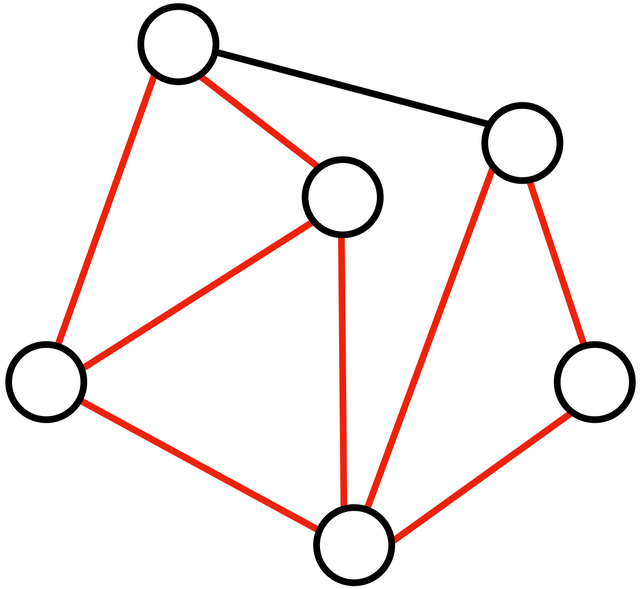
Problem: count the number of triangles in graphs.

Given:

- a set V of nodes, $\left|V\right| = n$
- A set E of edges, $\left|E\right| = m$

Goal:

- Count the **global** number of triangles $\Delta = \{u, v, w\}$, where $\{u, v\}$, $\{w, u\}$, and $\{v, w\}$ are all in the set E of the edges.



Applications:

- Community detection
- Anomaly detection
- Molecular biology

Problem Definition

This talk: Triangle Counting Using Predictions

C. Boldrin and F. Vandin, **“Fast and Accurate Triangle Counting in Graph Streams Using Predictions”**, to appear at ICDM 2024.

Algorithms with Predictions

Use of predictions about the input data has been formalised in the **“Algorithms with Predictions”** framework [Mitzenmacher and Vassilvitskii, 2020]

- Go beyond worst-case analysis
- Oracle empowering effectiveness of classical algorithms

Warm-up: Binary Search

Consider a sorted array A of n elements, we want to check if a given element x is present in the array A.

1	7	9	11	17	23	30	39	42	48	56	61	68	75	80	83	91

Warm-up: Binary Search

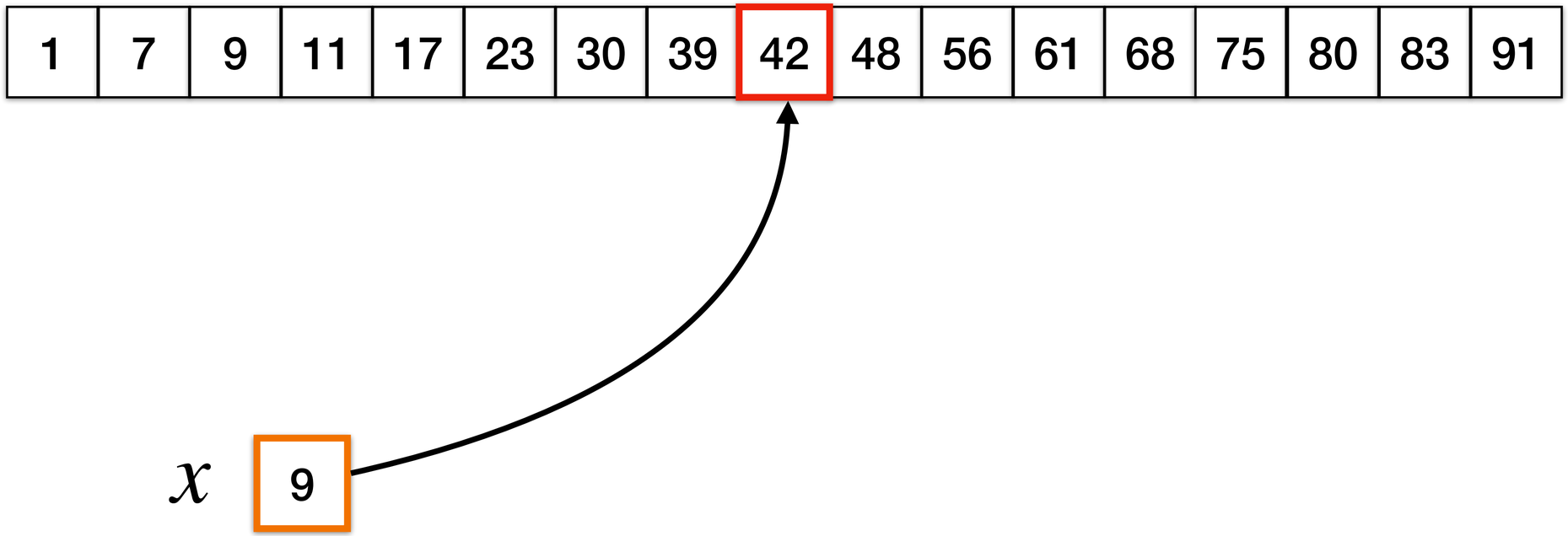
Consider a sorted array A of n elements, we want to check if a given element x is present in the array A.

1	7	9	11	17	23	30	39	42	48	56	61	68	75	80	83	91

x | 9 |

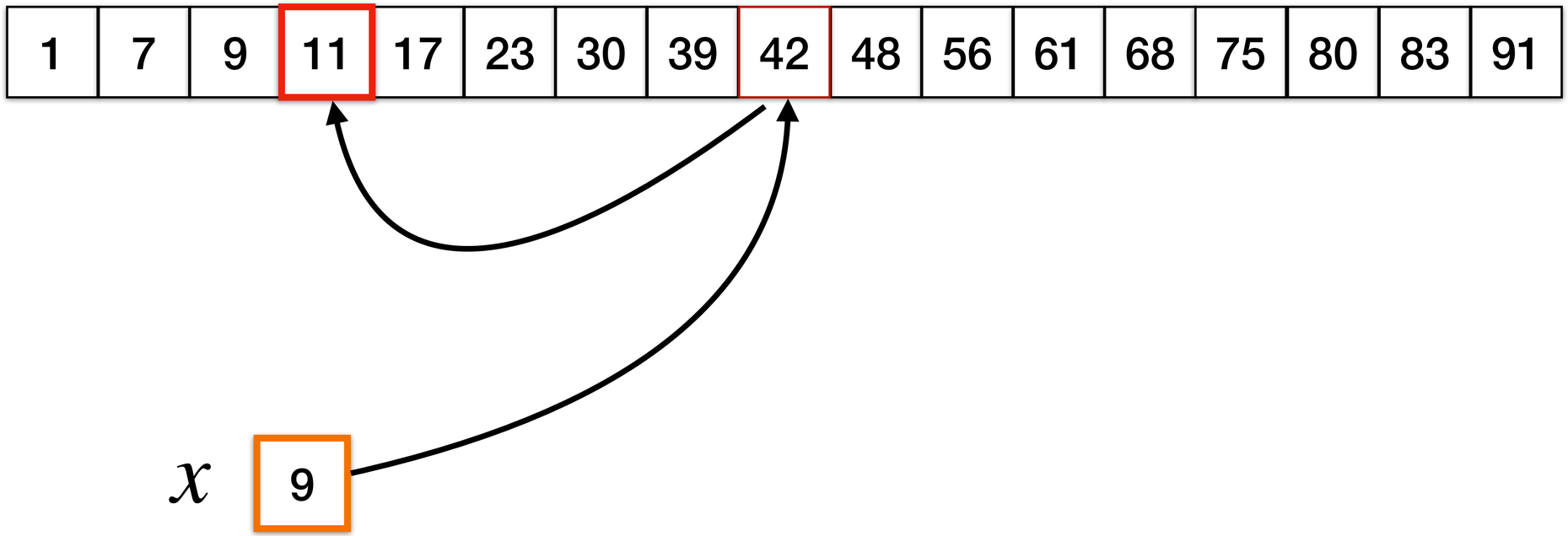
Warm-up: Binary Search

Consider a sorted array A of n elements, we want to check if a given element x is present in the array A.



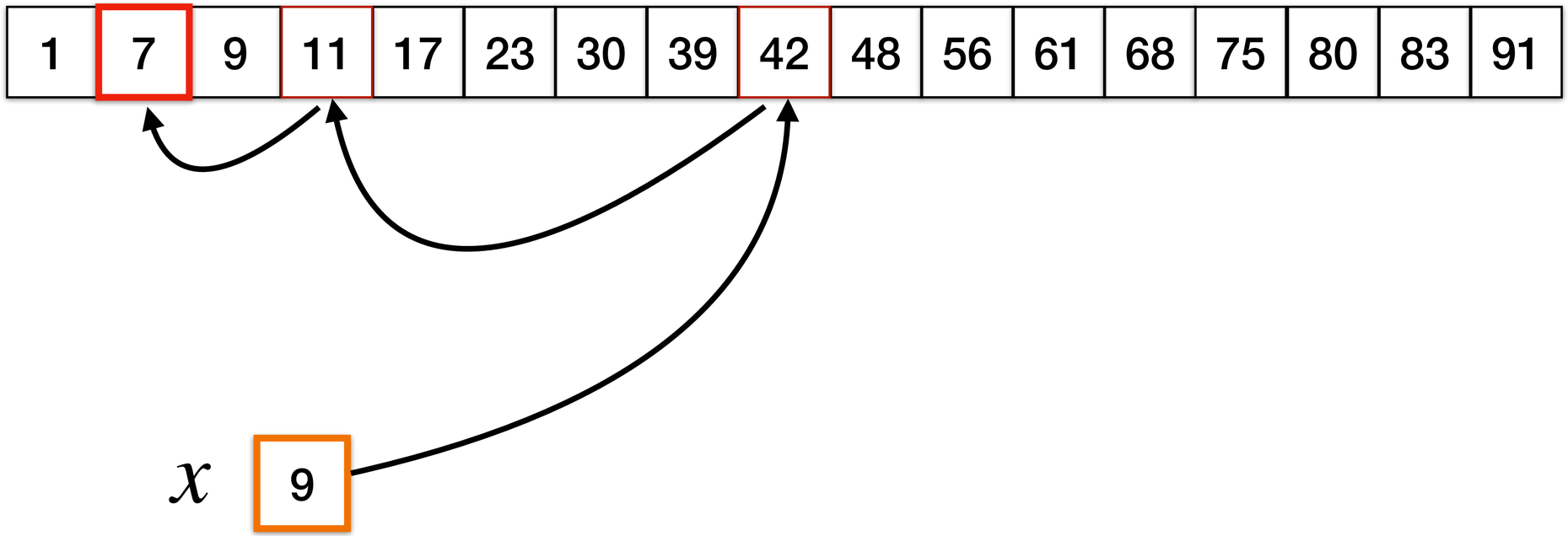
Warm-up: Binary Search

Consider a sorted array A of n elements, we want to check if a given element x is present in the array A.



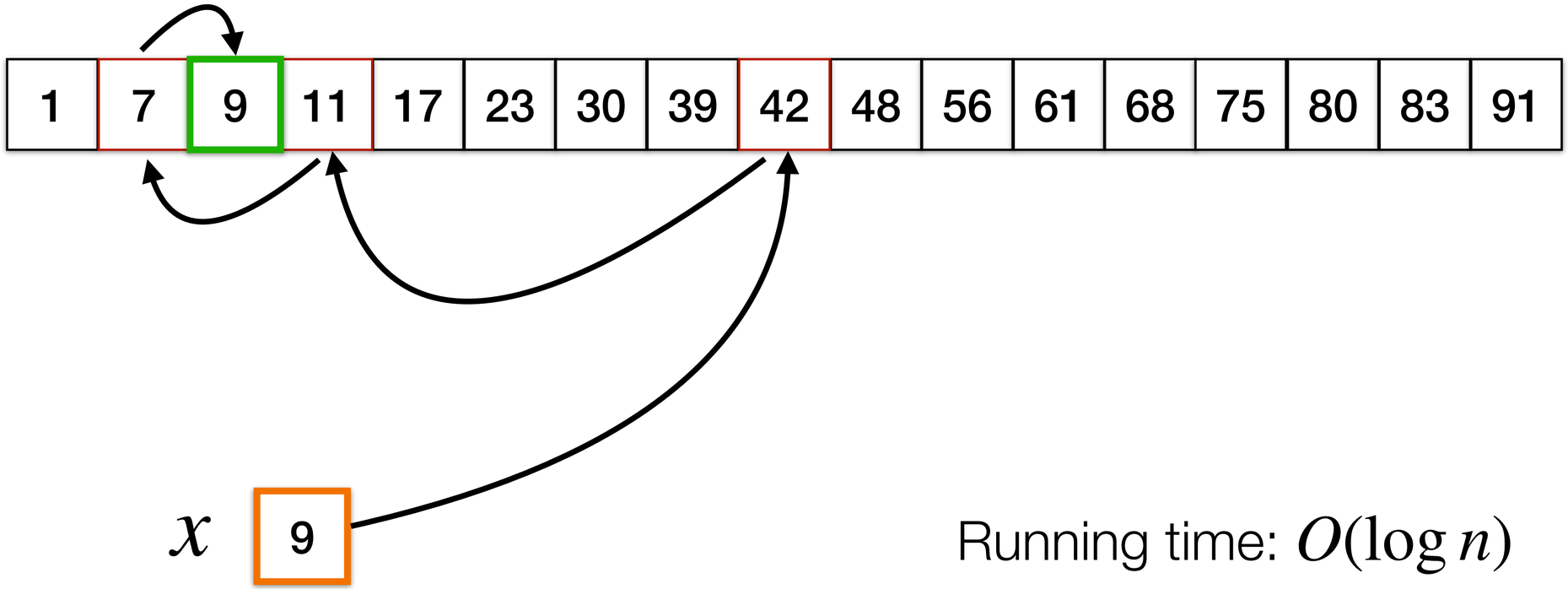
Warm-up: Binary Search

Consider a sorted array A of n elements, we want to check if a given element x is present in the array A.



Warm-up: Binary Search

Consider a sorted array A of n elements, we want to check if a given element x is present in the array A.



Running time: $O(\log n)$

Warm-up: Binary Search

Question: can we go beyond worst case analysis?

Warm-up: Binary Search

Question: can we go beyond worst case analysis?

Example:

- *Array A:* collection of publications from University of Padova, sorted by main author's surname
- *Query element x*: "Boldrin"

Warm-up: Binary Search

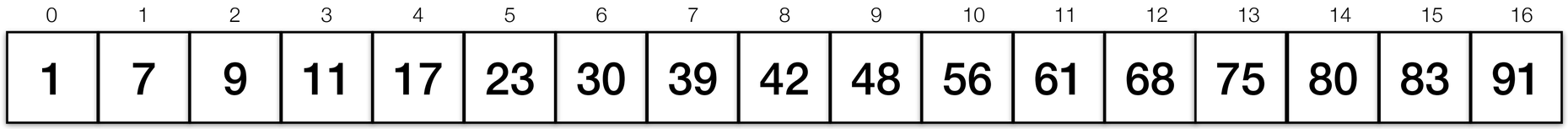
Question: can we go beyond worst case analysis?

Example:

- *Array A:* collection of publications from University of Padova, sorted by main author's surname
- *Query element x*: "Boldrin"
- You would probably start your search near the beginning of the collection
- For example, since 'B' is the second letter out of 26, start search from $2/26 \approx 8\,\%$ of documents
- Proceed by **doubling** binary search

Warm-up: Binary Search

Predictor h telling us approximately where query element x should appear inside A.

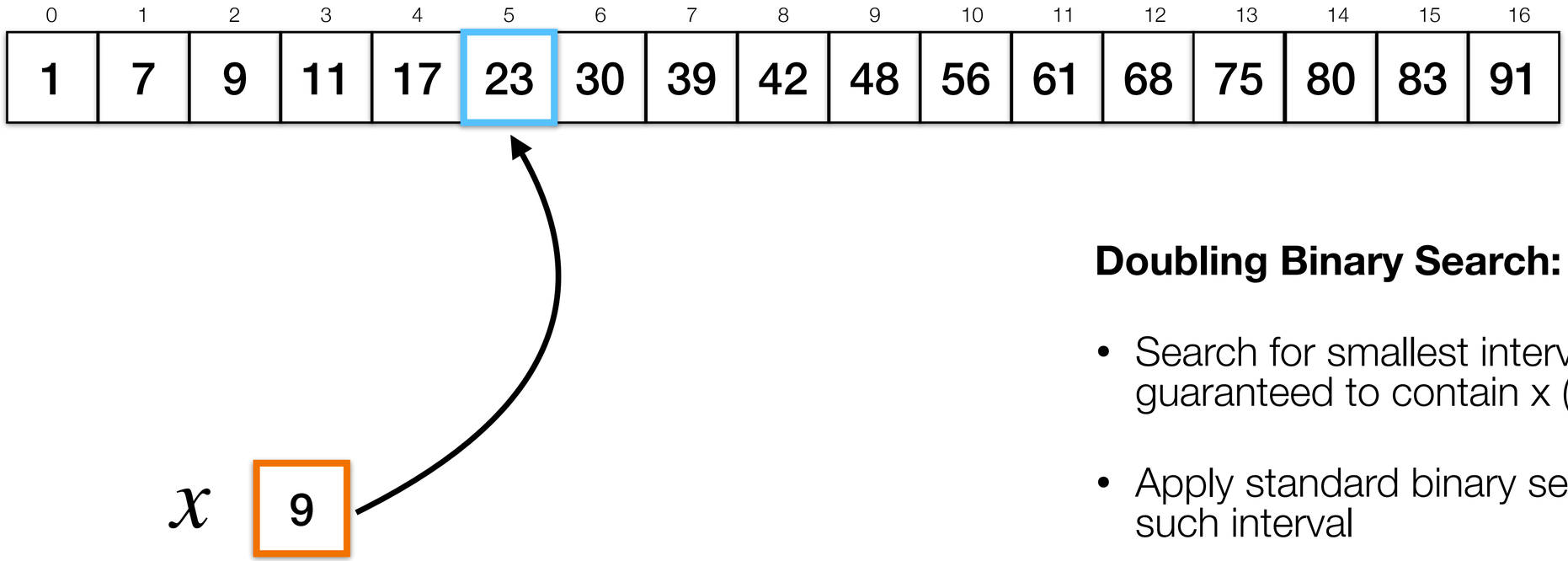


x 9

$$h(x) = 5$$

Warm-up: Binary Search

Predictor h telling us approximately where query element x should appear inside A.



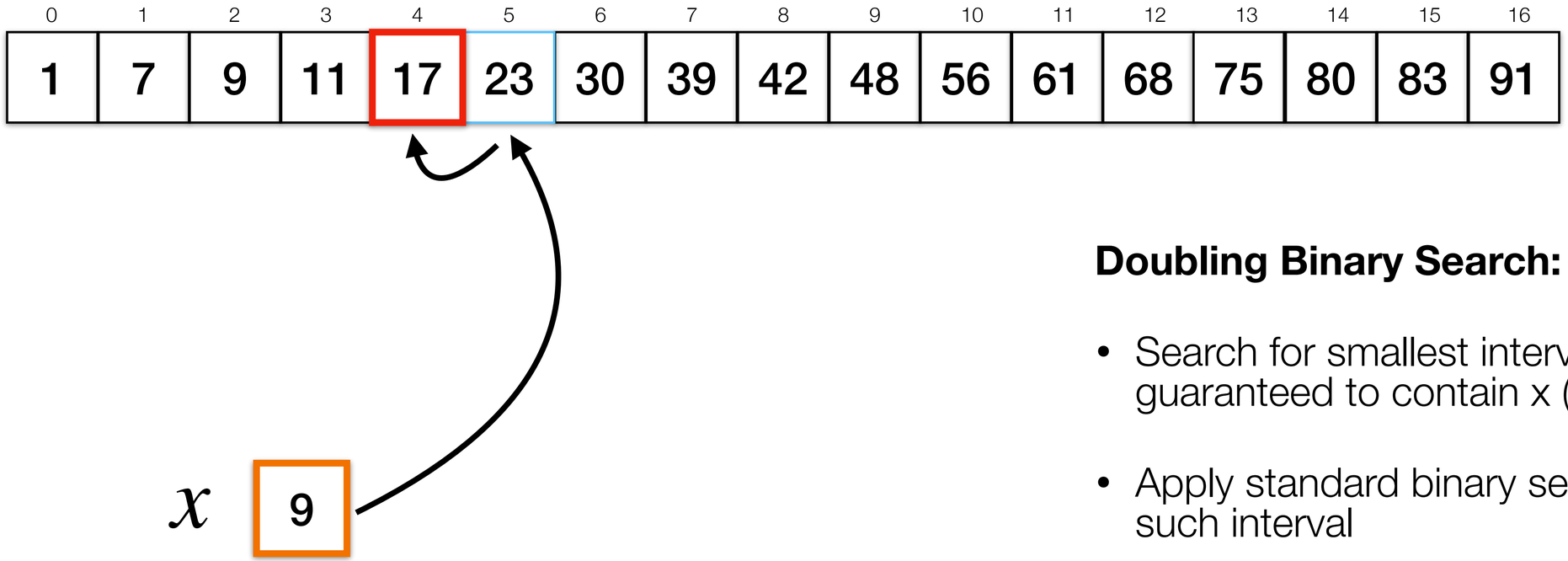
Doubling Binary Search:

- Search for smallest interval guaranteed to contain x (if it exists)
- Apply standard binary search on such interval

$$h(x) = 5$$

Warm-up: Binary Search

Predictor h telling us approximately where query element x should appear inside A.



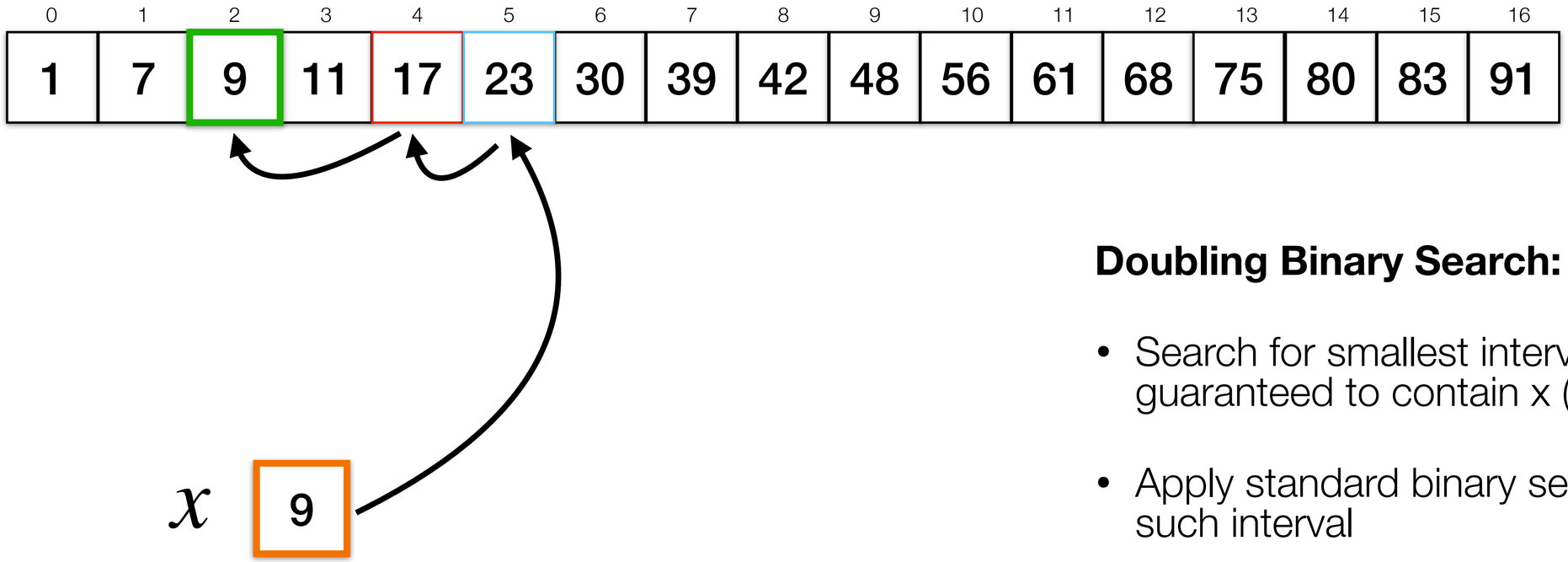
Doubling Binary Search:

- Search for smallest interval guaranteed to contain x (if it exists)
- Apply standard binary search on such interval

$$h(x) = 5$$

Warm-up: Binary Search

Predictor h telling us approximately where query element x should appear inside A.



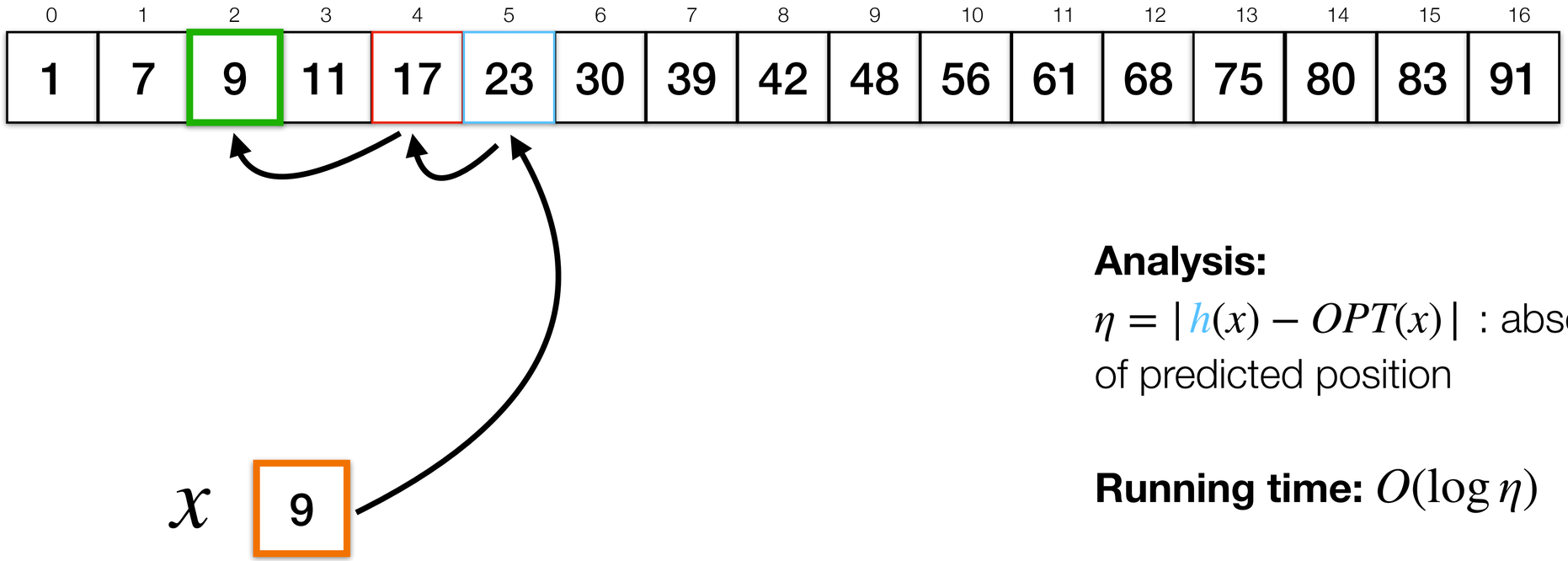
Doubling Binary Search:

- Search for smallest interval guaranteed to contain x (if it exists)
- Apply standard binary search on such interval

$$h(x) = 5$$

Warm-up: Binary Search

Predictor h telling us approximately where query element x should appear inside A.



Analysis:

$\eta = |h(x) - OPT(x)|$: absolute error of predicted position

Running time: $O(\log \eta)$

$$h(x) = 5$$

Algorithms with Predictions

Use of predictions about the input data has been formalised in the **“Algorithms with Predictions”** framework [Mitzenmacher and Vassilvitskii, 2020]

- Go beyond worst-case analysis
- Oracle empowering effectiveness of classical algorithms

Challenges:

- **Consistency:** useful predictions improve performances
- **Robustness:** bad predictions do not worsen too much performances
- **Practicality:** predictions derived by tasks on same data-domain

Algorithms with Predictions

Binary Search Example:

- Classical: $O(\log n)$
- Learning Augmented: $O(\log \eta)$

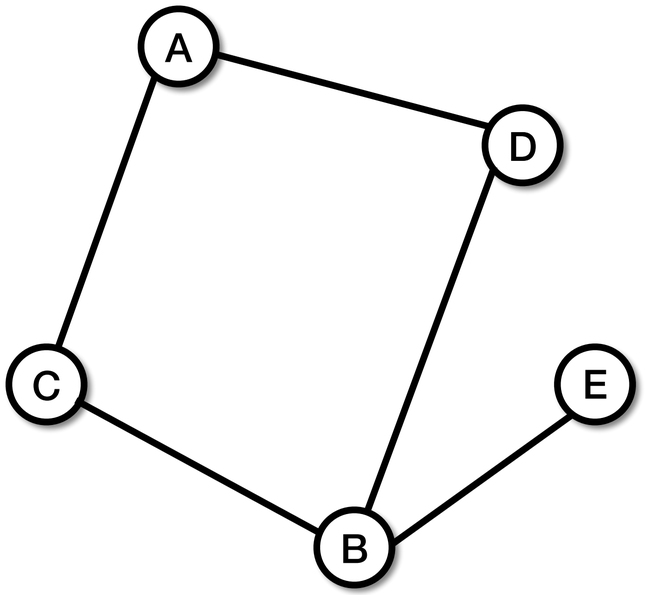
- **Consistency:** useful predictions improve performances
- **Robustness:** bad predictions do not worsen too much performances

Settings of our problem

Streaming model.

Edges are observed as a stream of updates in arbitrary order.

Updates: insertions and deletions.



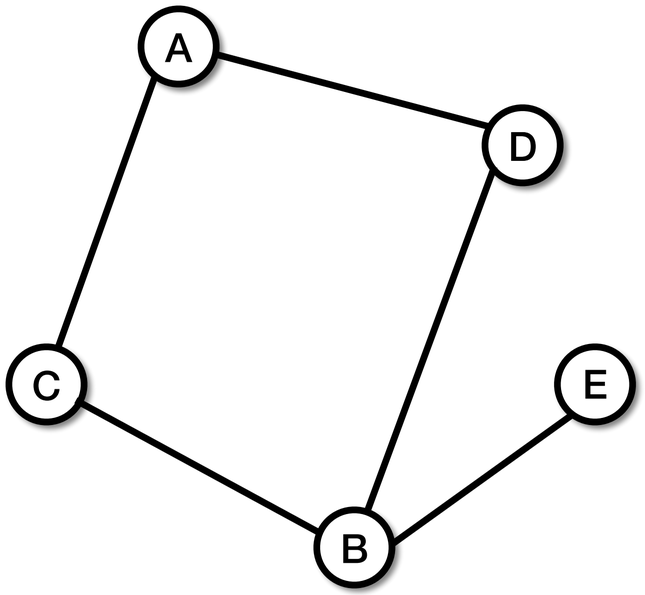
Graph

Settings of our problem

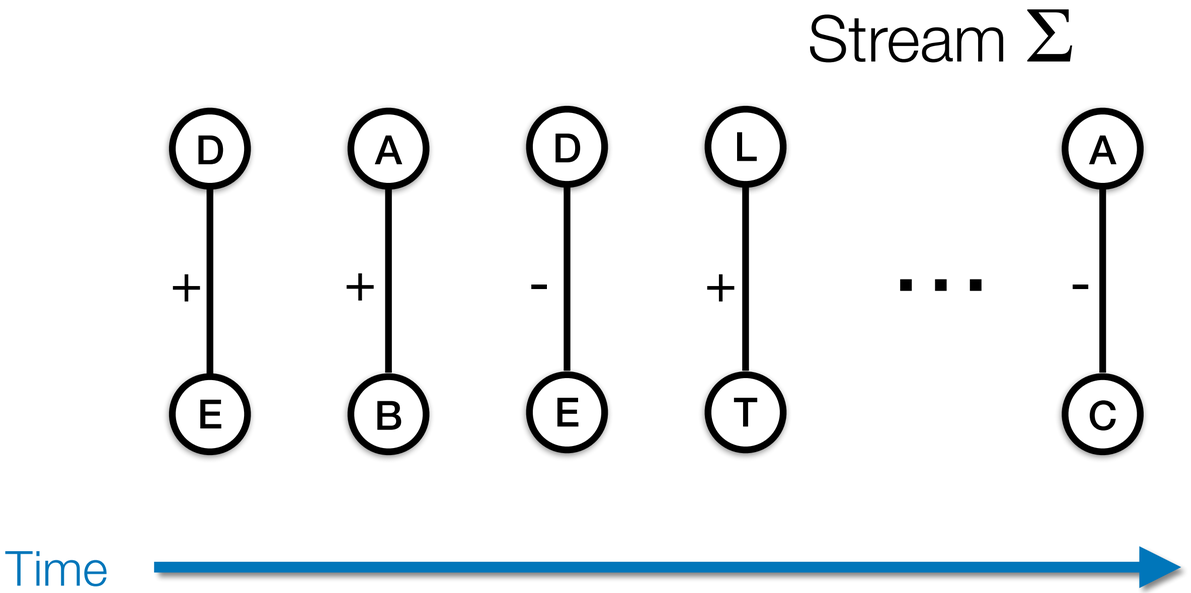
Streaming model.

Edges are observed as a stream of updates in arbitrary order.

Updates: insertions and deletions.



Graph

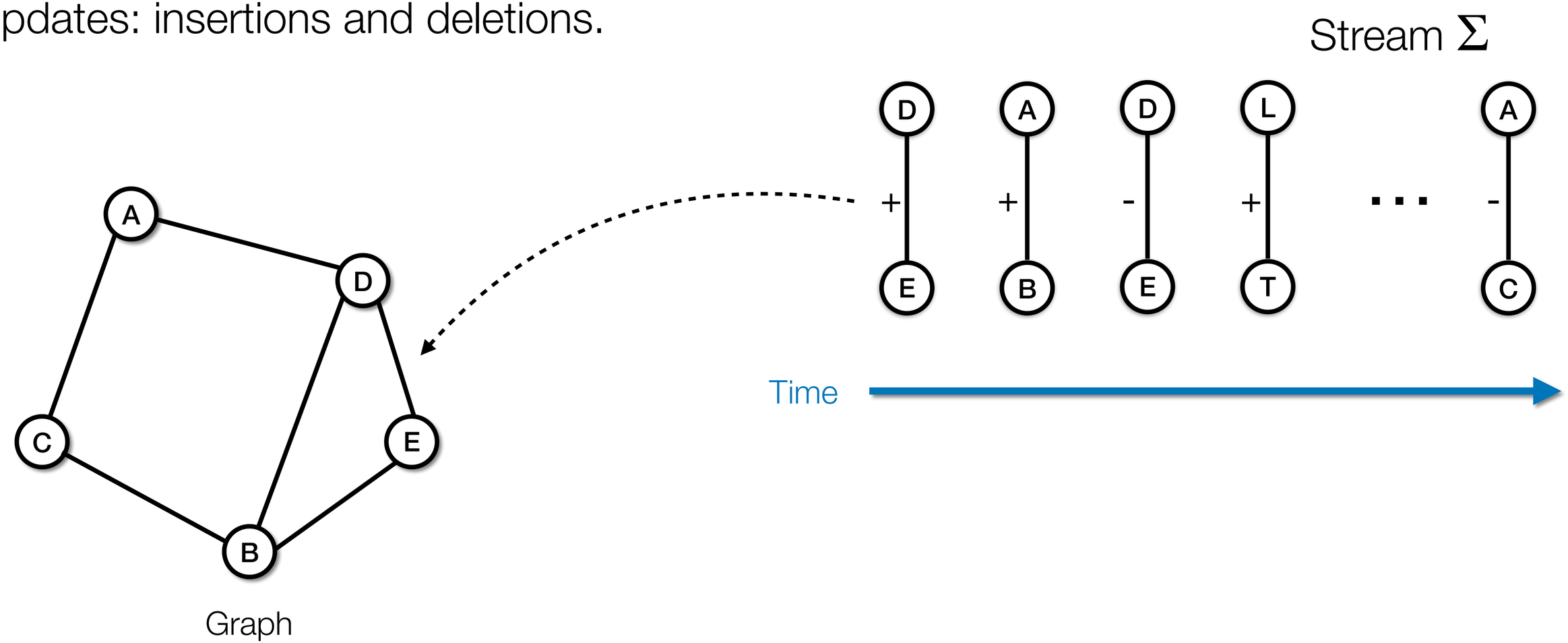


Settings of our problem

Streaming model.

Edges are observed as a stream of updates in arbitrary order.

Updates: insertions and deletions.

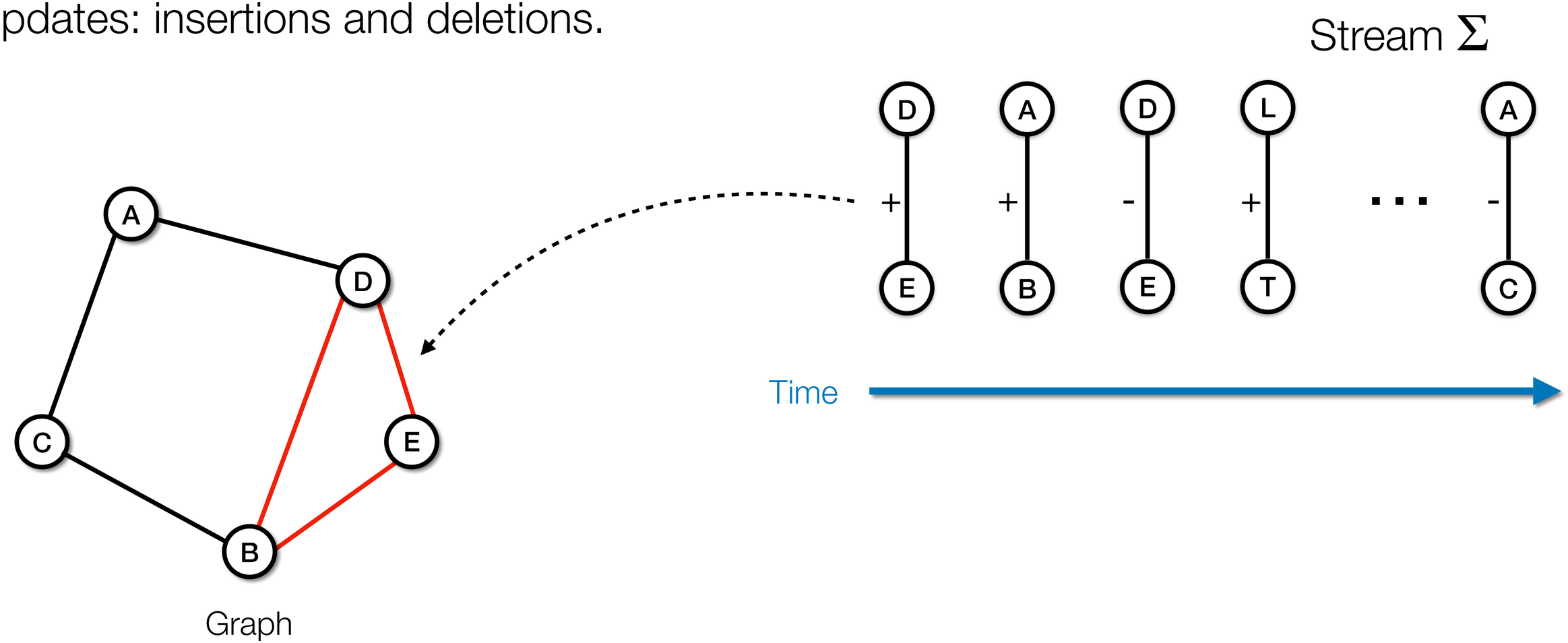


Settings of our problem

Streaming model.

Edges are observed as a stream of updates in arbitrary order.

Updates: insertions and deletions.

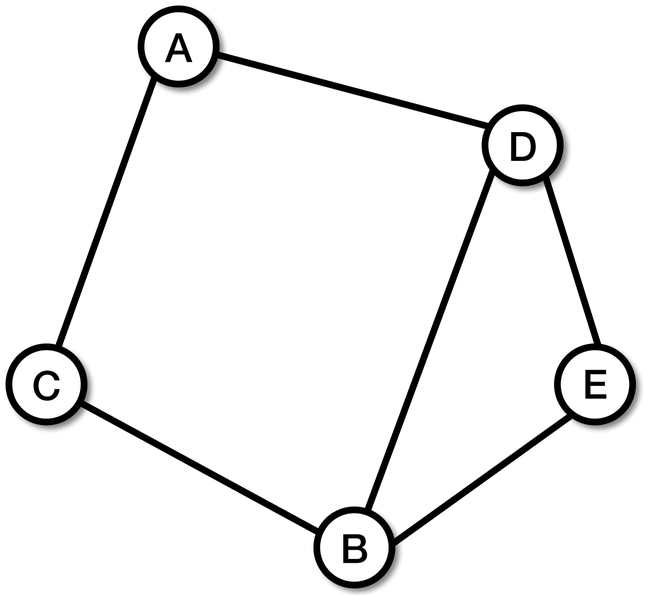


Settings of our problem

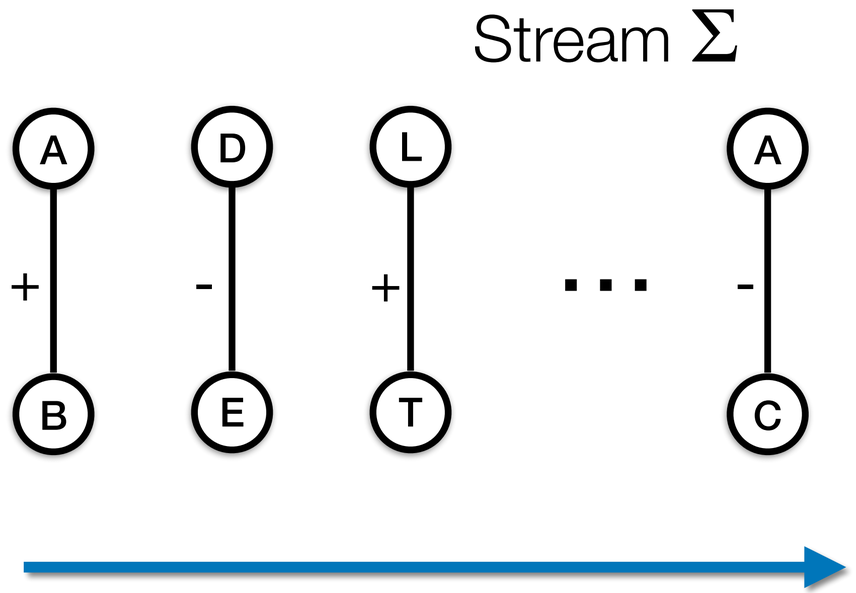
Streaming model.

Edges are observed as a stream of updates in arbitrary order.

Updates: insertions and deletions.



Graph

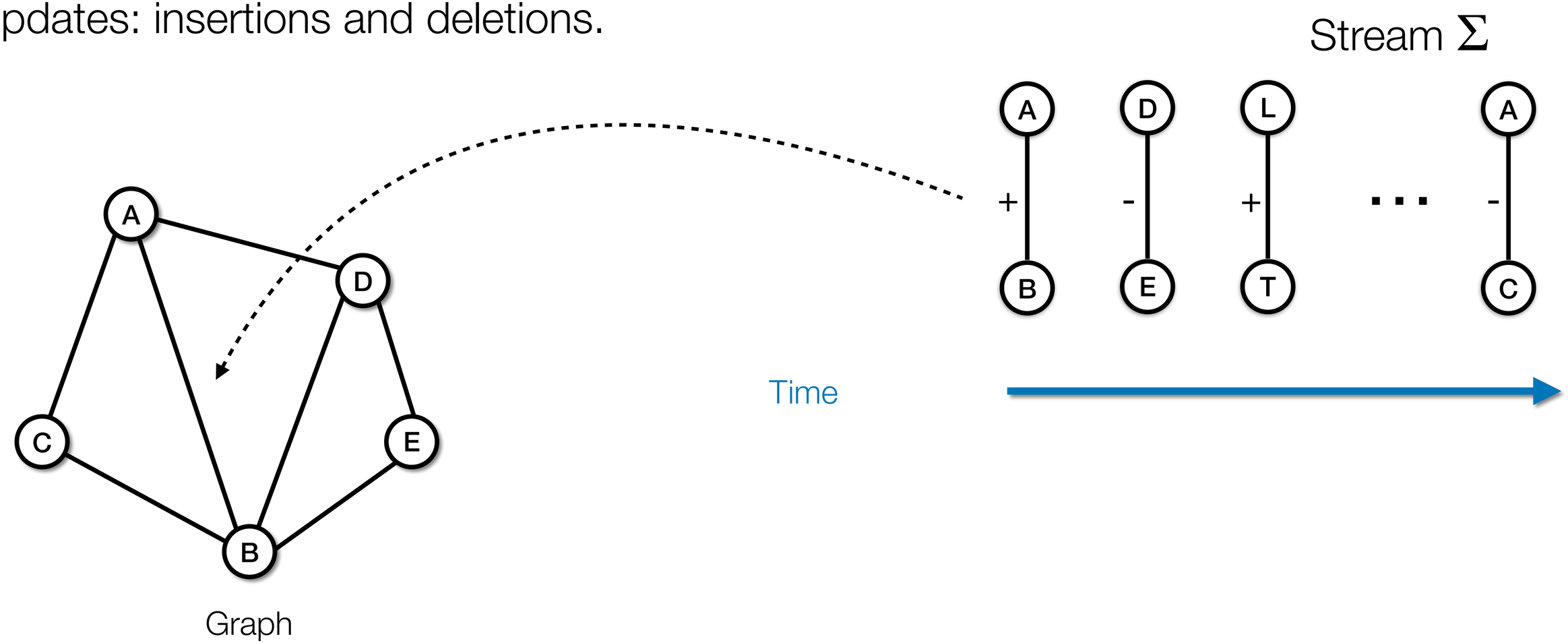


Settings of our problem

Streaming model.

Edges are observed as a stream of updates in arbitrary order.

Updates: insertions and deletions.

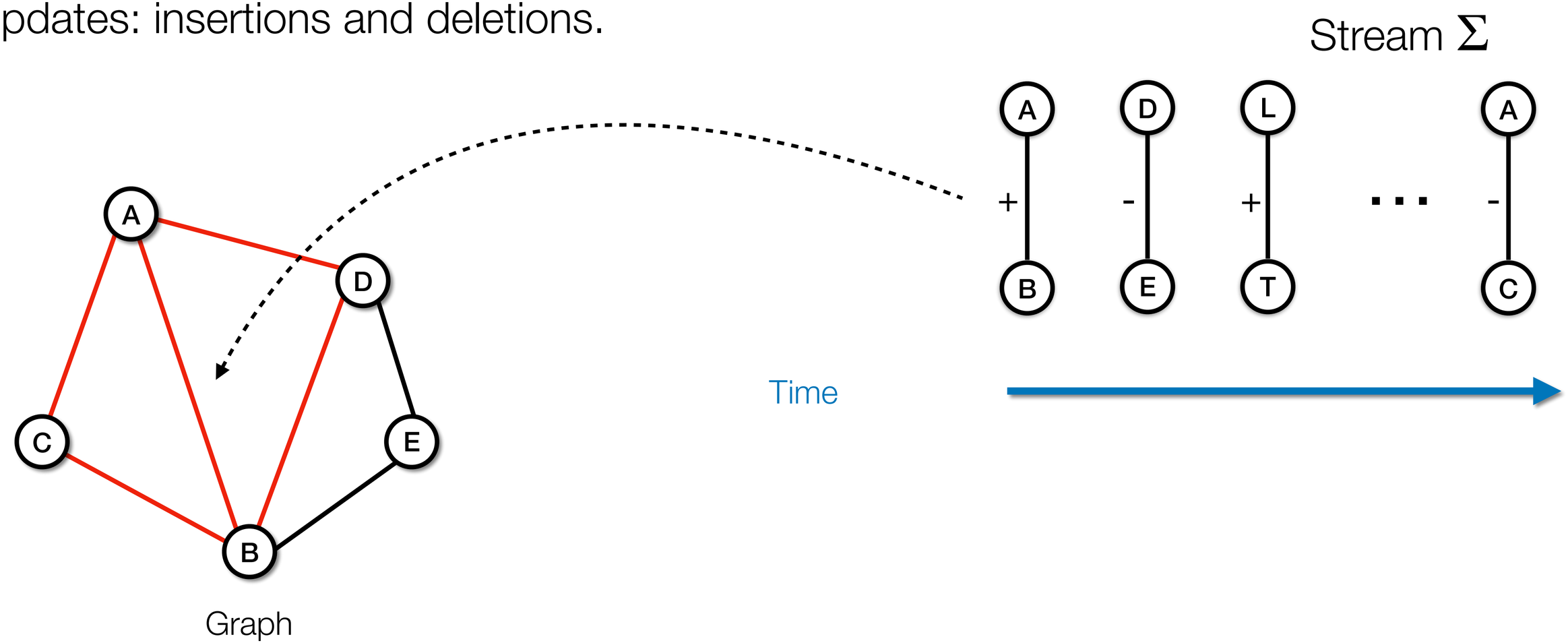


Settings of our problem

Streaming model.

Edges are observed as a stream of updates in arbitrary order.

Updates: insertions and deletions.

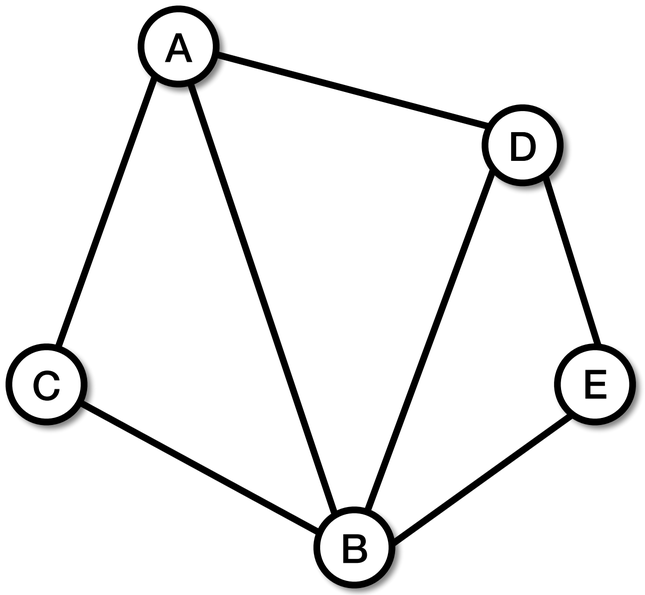


Settings of our problem

Streaming model.

Edges are observed as a stream of updates in arbitrary order.

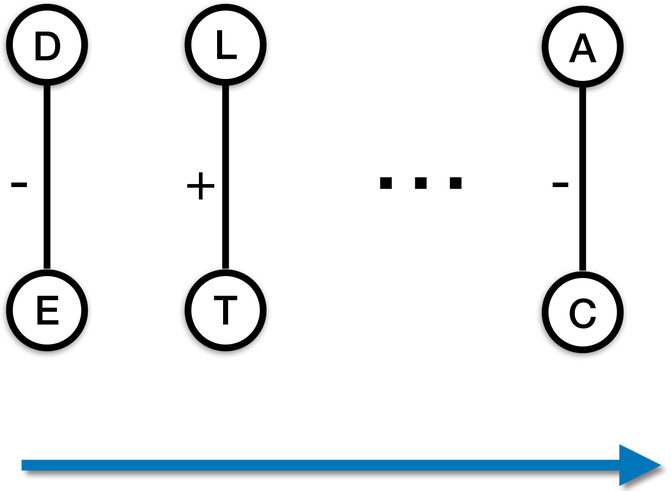
Updates: insertions and deletions.



Graph

Time

Stream Σ

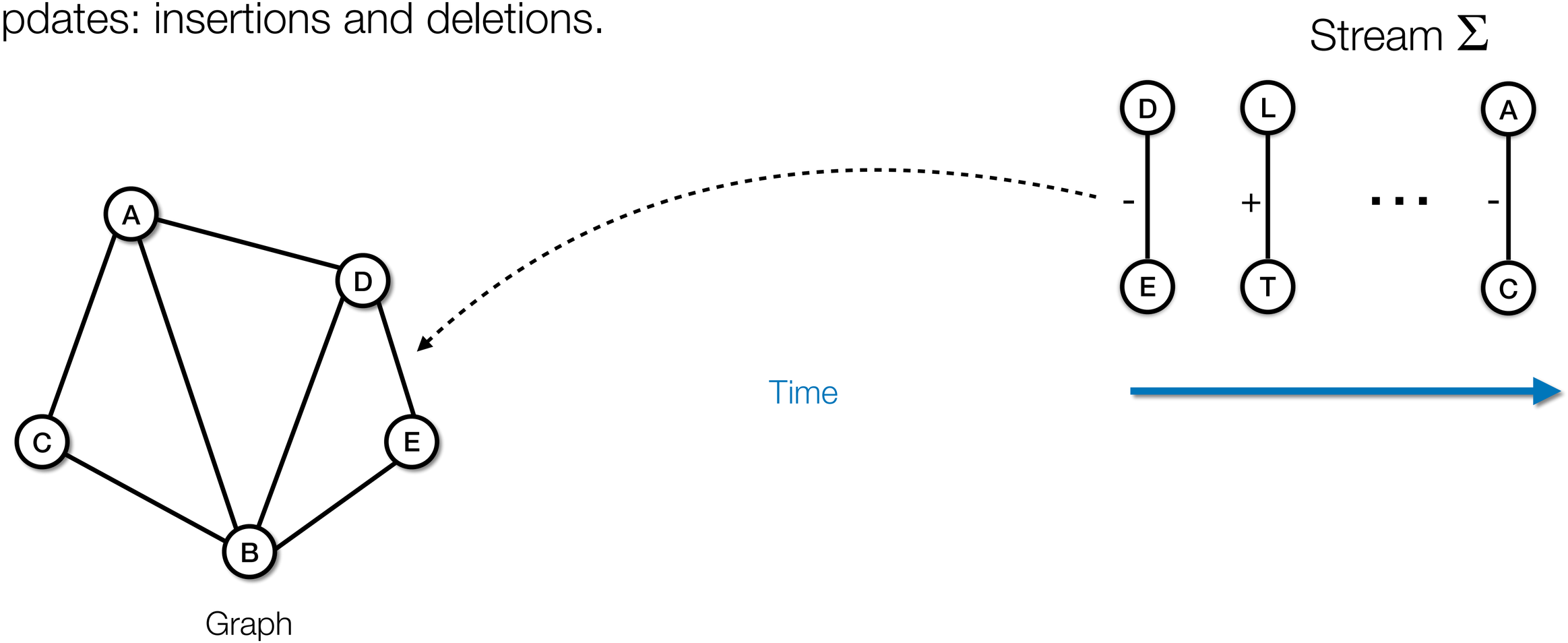


Settings of our problem

Streaming model.

Edges are observed as a stream of updates in arbitrary order.

Updates: insertions and deletions.

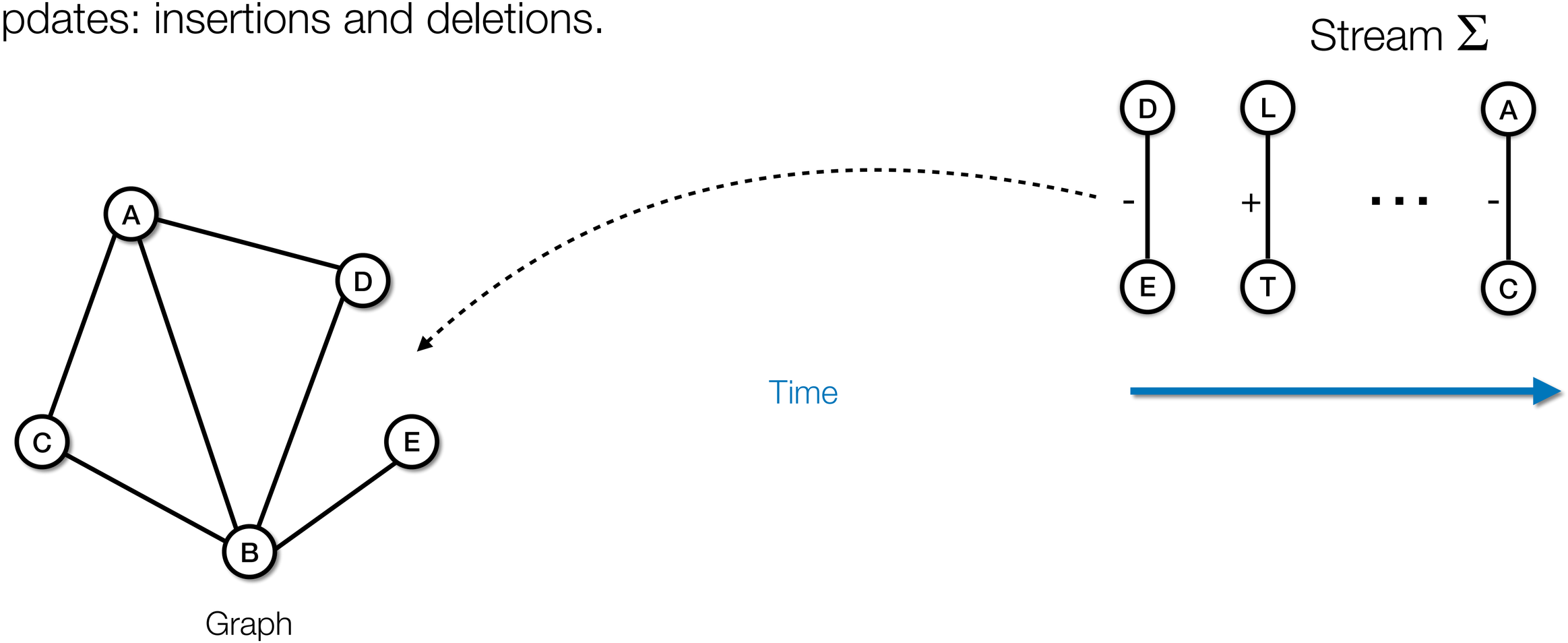


Settings of our problem

Streaming model.

Edges are observed as a stream of updates in arbitrary order.

Updates: insertions and deletions.

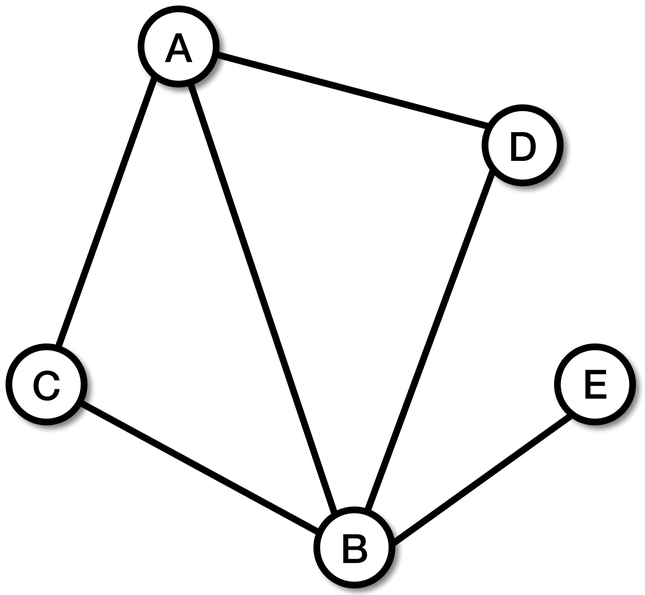


Settings of our problem

Streaming model.

Edges are observed as a stream of updates in arbitrary order.

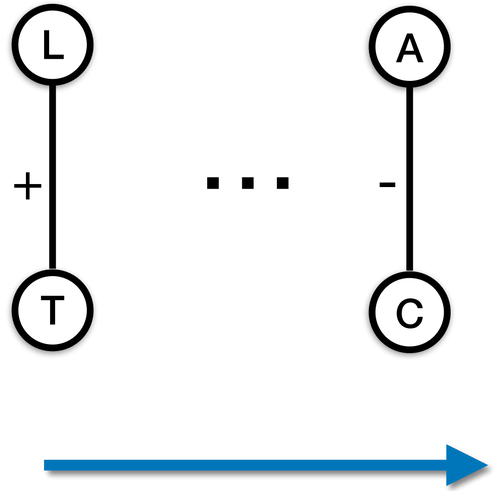
Updates: insertions and deletions.



Graph

Time

Stream Σ

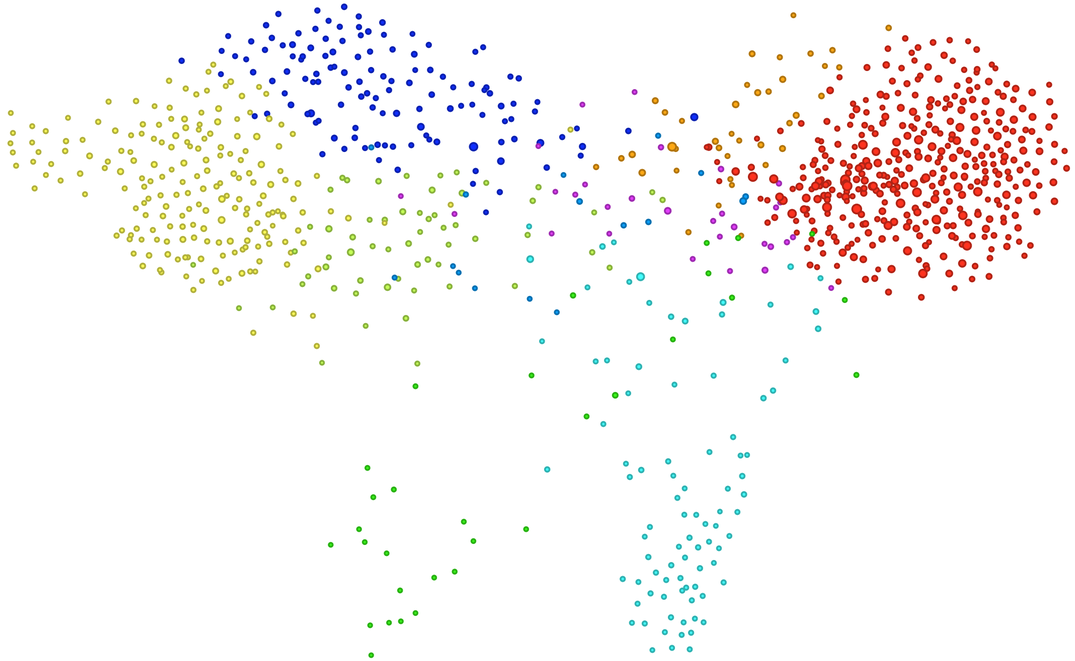


Settings of our problem

In most applications, the **exact** computation of triangles is unfeasible, due to the size of the data.

Settings of our problem

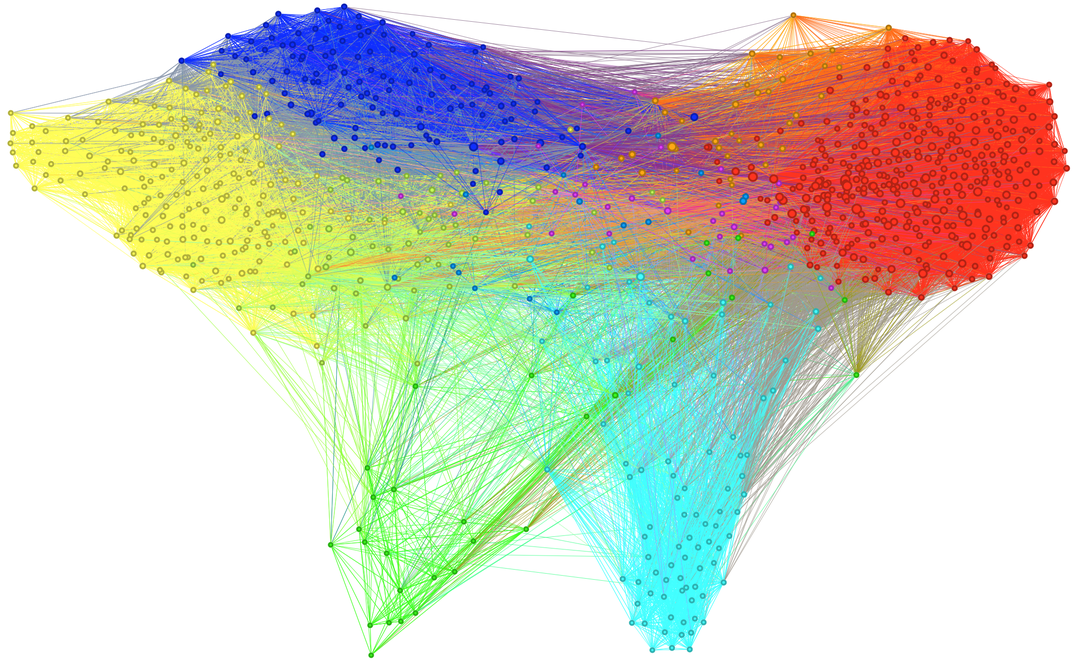
In most applications, the **exact** computation of triangles is unfeasible, due to the size of the data.



Graph of **Twitter** followers

Settings of our problem

In most applications, the **exact** computation of triangles is unfeasible, due to the size of the data.

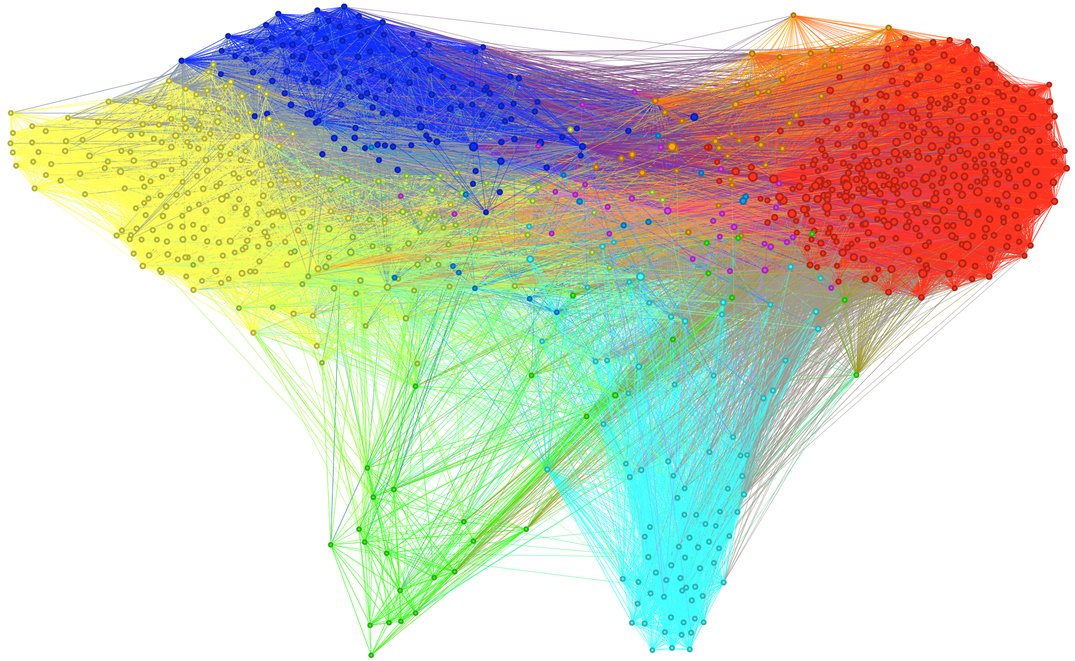


Graph of **Twitter** followers

Settings of our problem

In most applications, the **exact** computation of triangles is unfeasible, due to the size of the data.

- Design **fast** and **efficient** algorithms, that provide **high-quality** approximation

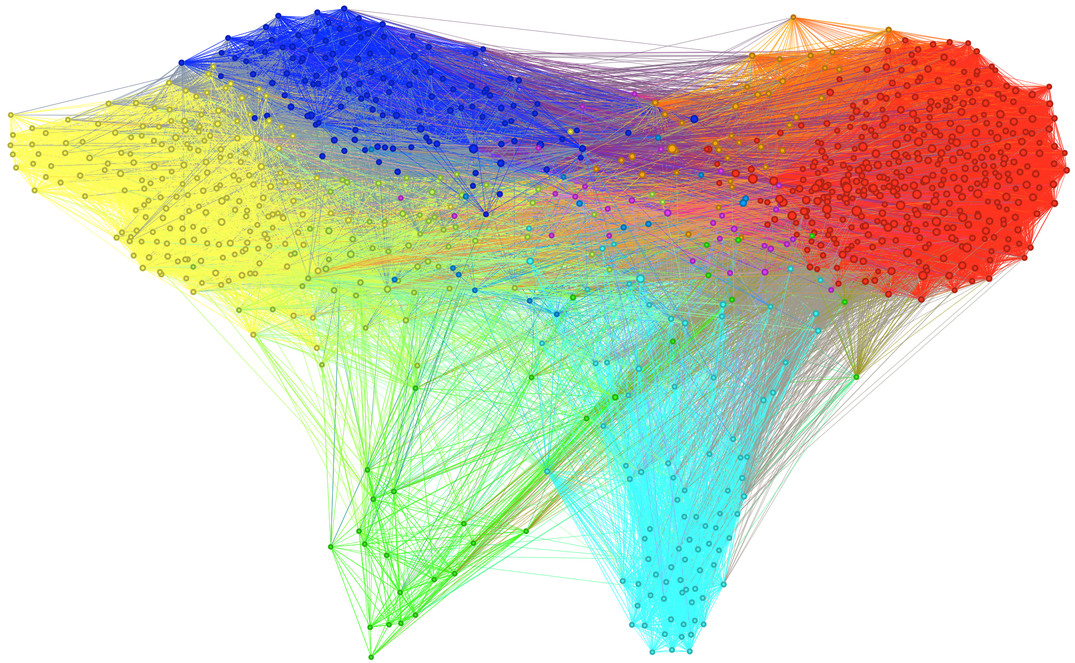


Graph of **Twitter** followers

Settings of our problem

In most applications, the **exact** computation of triangles is unfeasible, due to the size of the data.

- Design **fast** and **efficient** algorithms, that provide **high-quality** approximation
- For example, we can store a small fraction of edges of the graph



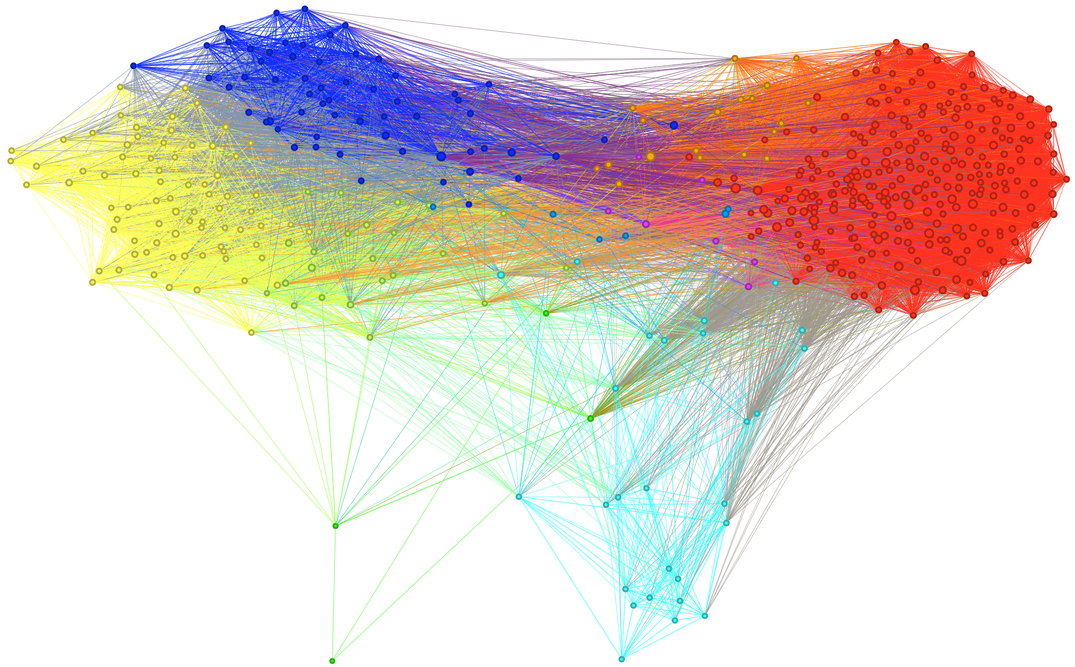
Graph of **Twitter** followers

Settings of our problem

In most applications, the **exact** computation of triangles is unfeasible, due to the size of the data.

- Design **fast** and **efficient** algorithms, that provide **high-quality** approximation
- For example, we can store a small fraction of edges of the graph

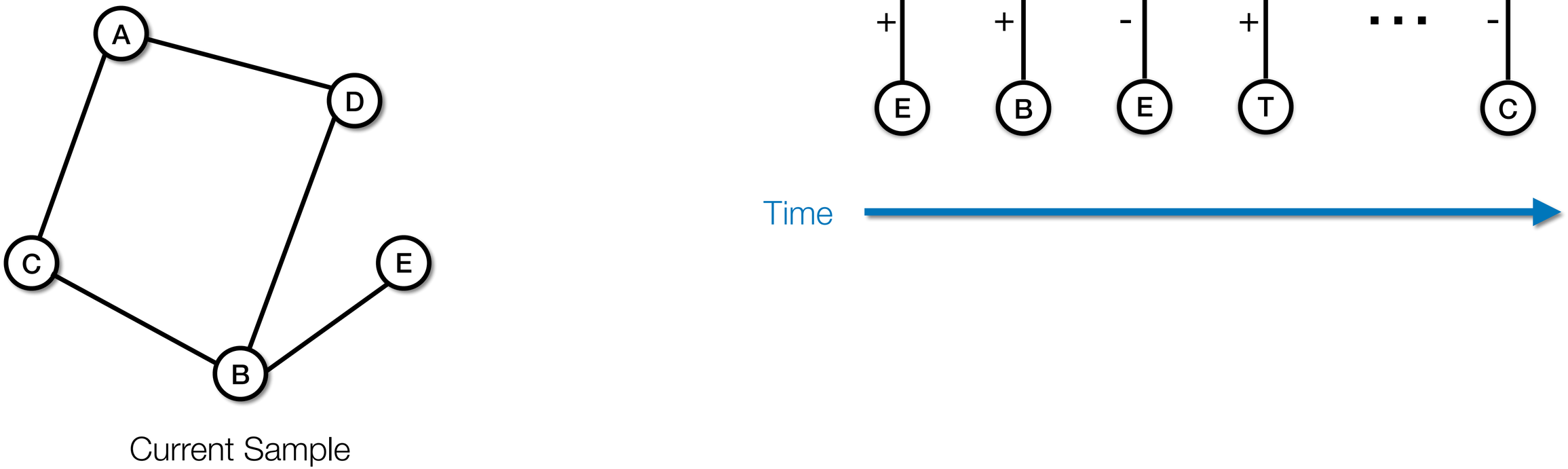
→ Use of **Sampling**



Sample of **Twitter** followers

Edge Sampling in Streaming

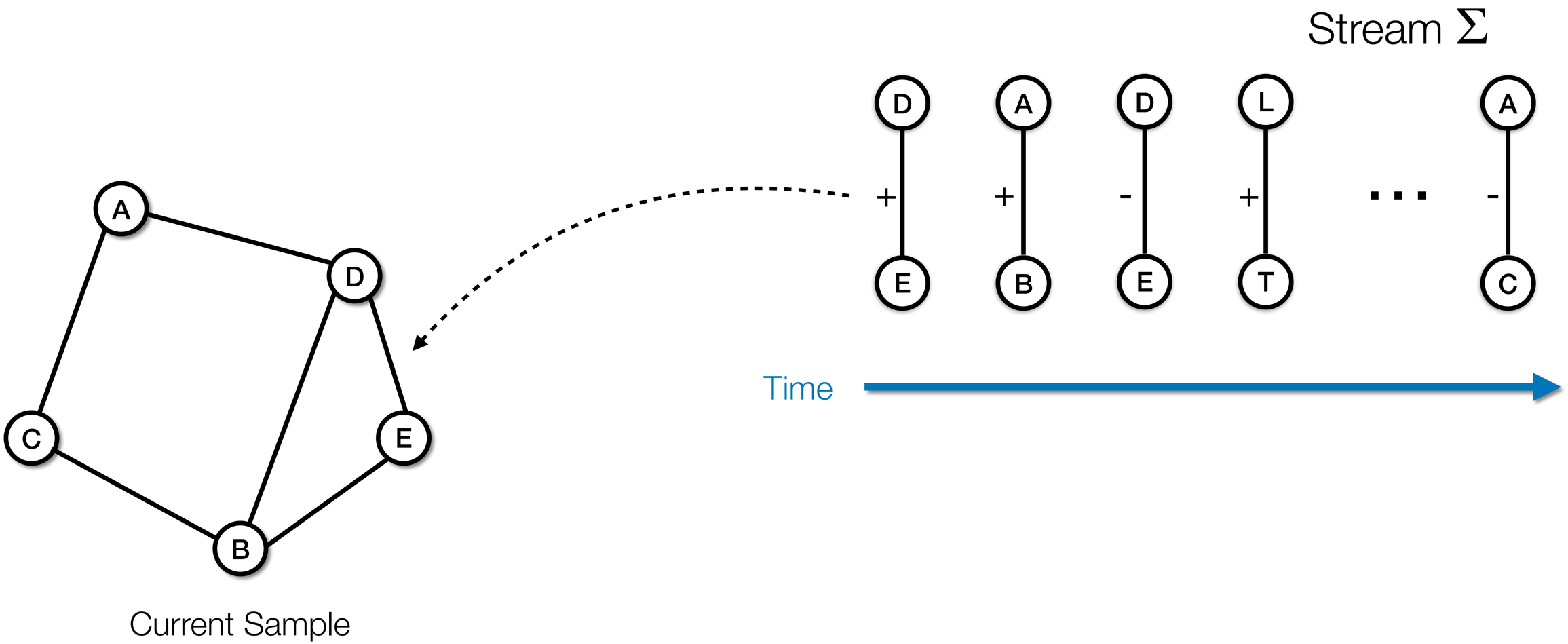
Each incoming edge on the stream is included in the sample with a certain probability.



Current Sample

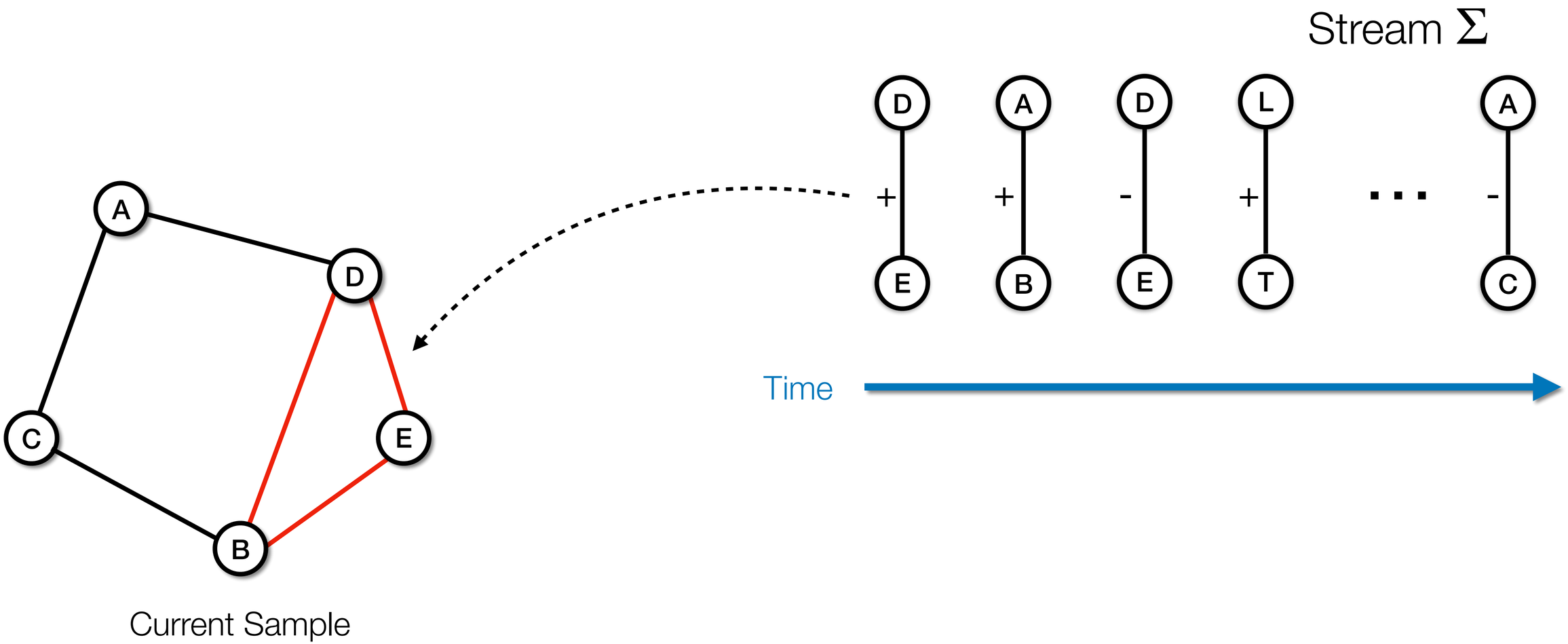
Edge Sampling in Streaming

Each incoming edge on the stream is included in the sample with a certain probability.



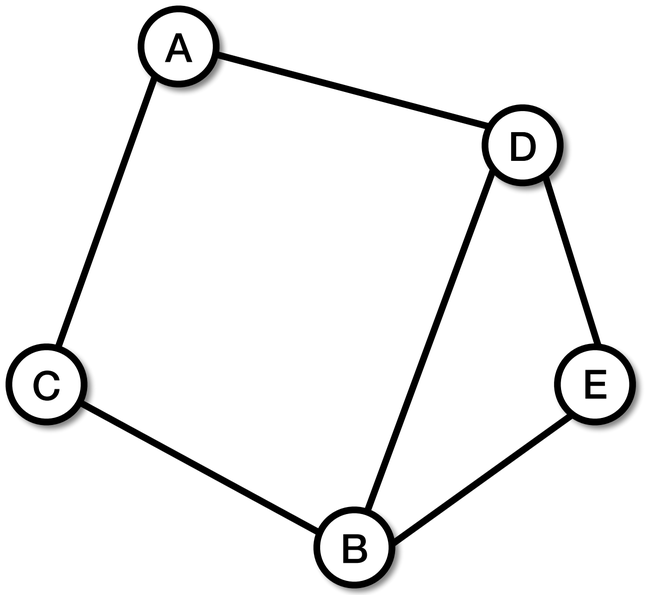
Edge Sampling in Streaming

Each incoming edge on the stream is included in the sample with a certain probability.

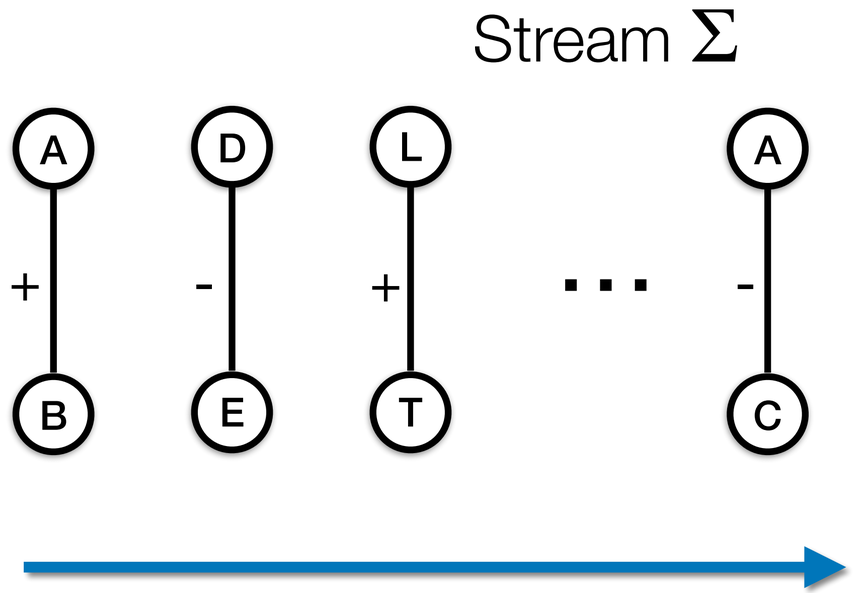


Edge Sampling in Streaming

Each incoming edge on the stream is included in the sample with a certain probability.

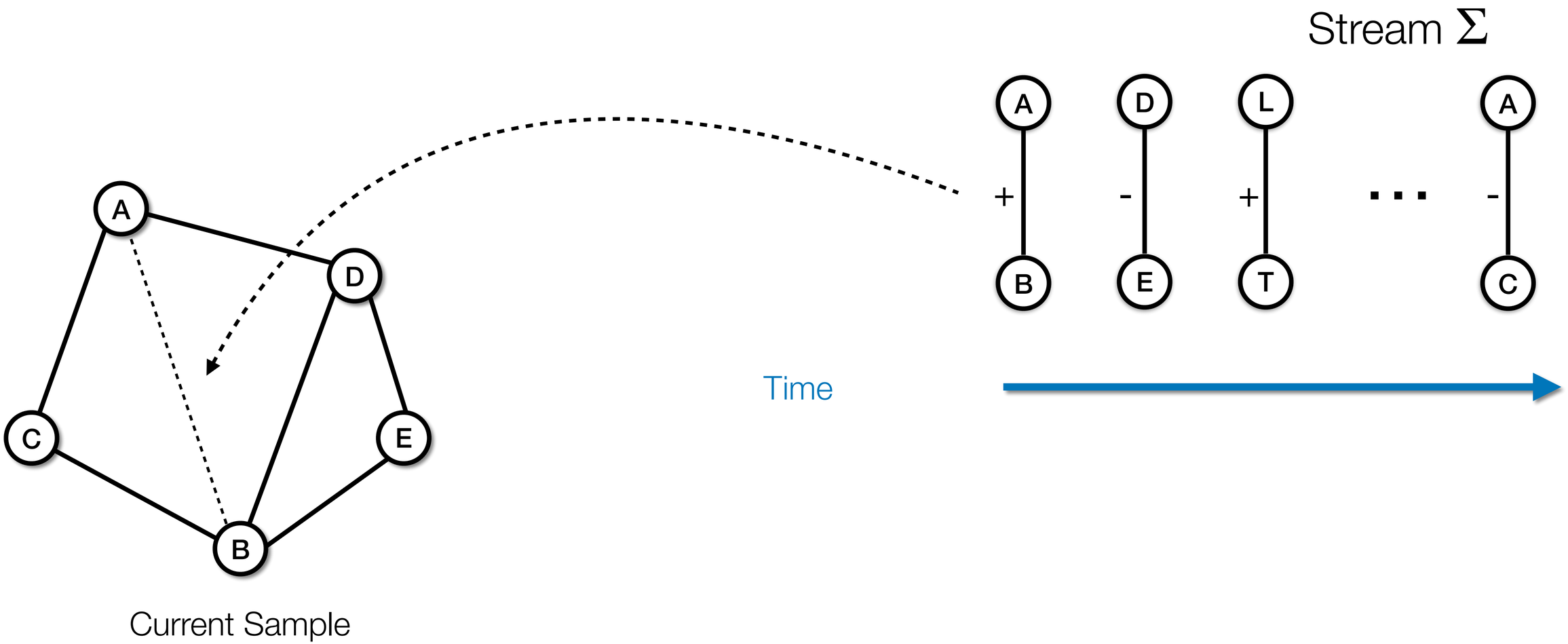


Current Sample



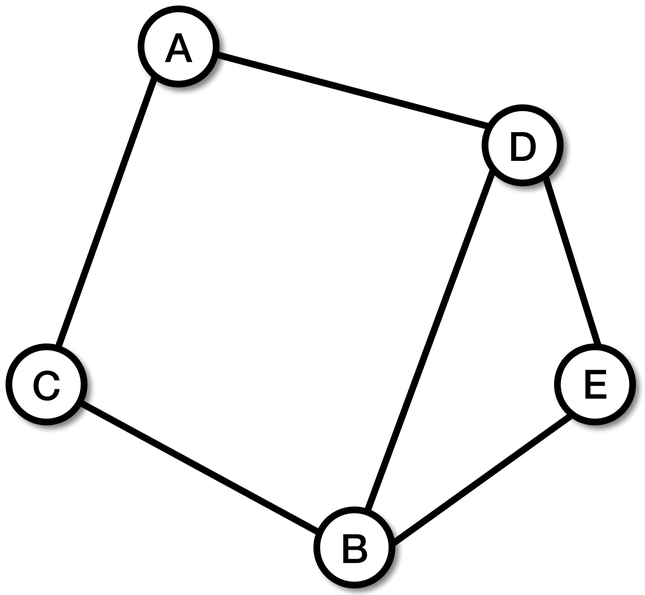
Edge Sampling in Streaming

Each incoming edge on the stream is included in the sample with a certain probability.

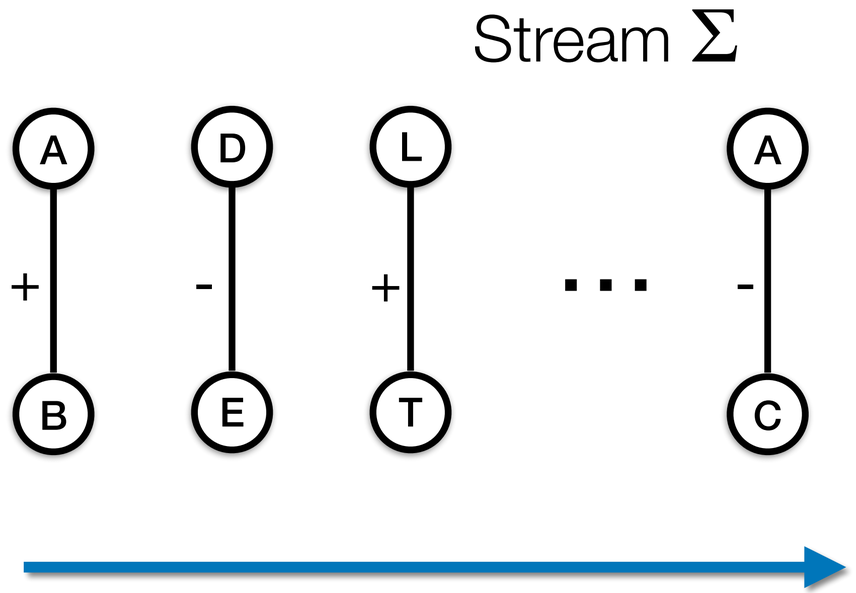


Edge Sampling in Streaming

Each incoming edge on the stream is included in the sample with a certain probability.



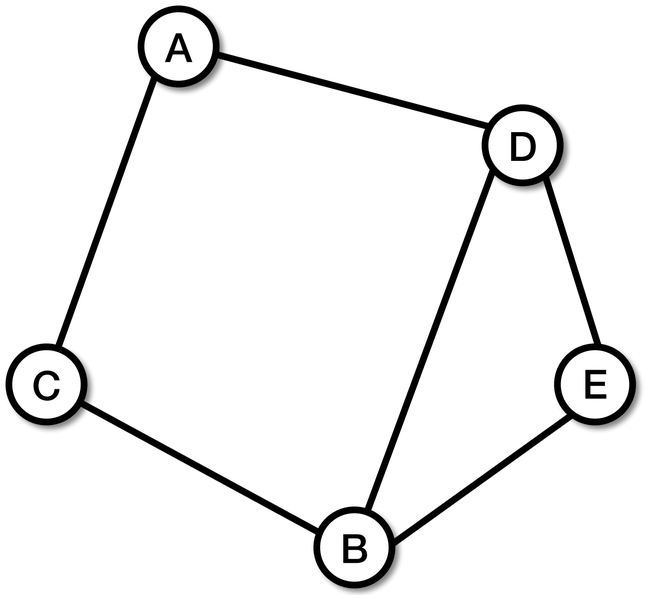
Current Sample



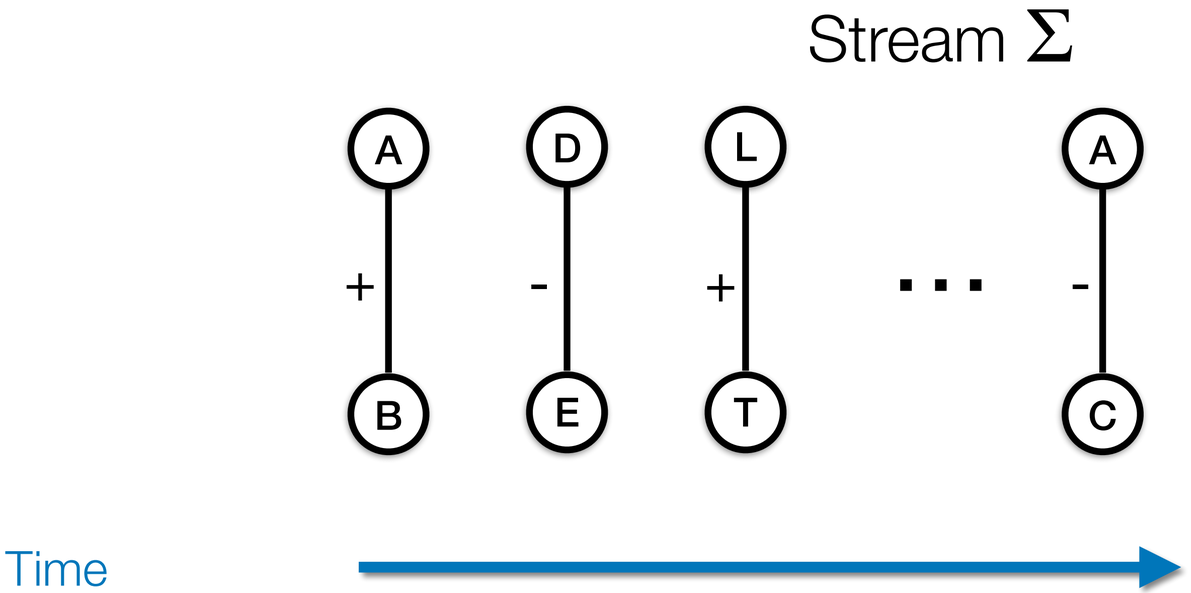
Edge Sampling in Streaming

Each incoming edge on the stream is included in the sample with a certain probability.

Choose carefully the sampling strategy.



Current Sample



Goals of Our Problem

Problem: Approximating the number of global triangles in graph streams using predictions.

Goals:

- Keep **high-quality** approximations **at every time** during the stream
- Updates of edges can only be **accessed once** (one-pass algorithm)
- Design a practical and efficient **predictor**

State of The Art

For **insertion-only** streams, we consider*:*

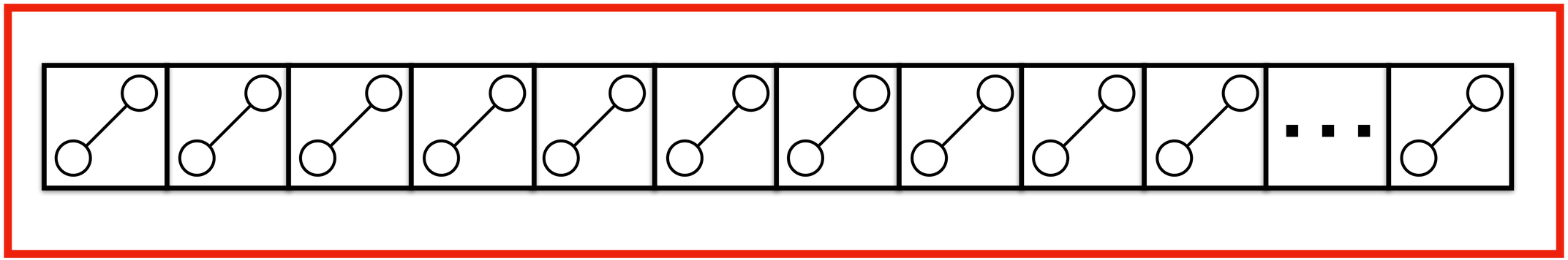
- ***Triest:*** reservoir sampling [De Stefani et al., KDD 2016]

State of The Art

For **insertion-only** streams, we consider*:*

- ***Triest:*** reservoir sampling [De Stefani et al., KDD 2016]

Uniform random sample of k edges



Memory budget k = number of edges to store

Reservoir Sampling

Uniform sampling of edges in the stream [De Stefani et al., KDD 2016].

A sample $S \subseteq E$ is said to be an **uniform sample** if all equal-sized subsets of E are equally likely to be S

$$\mathbb{P}\left[S = A\right] = \mathbb{P}\left[S = B\right], \forall\ A \neq B \subseteq E \text{ such that } \left|A\right| = \left|B\right|.$$

Reservoir Sampling

Uniform sampling of edges in the stream [De Stefani et al., KDD 2016].

A sample $S \subseteq E$ is said to be an **uniform sample** if all equal-sized subsets of E are equally likely to be S

$$\mathbb{P}\left[S = A\right] = \mathbb{P}\left[S = B\right], \forall\ A \neq B \subseteq E \text{ such that } \left|A\right| = \left|B\right|.$$

Let $e^{(t)}$ be the edge at time t. **Reservoir sampling** keeps a **uniform sample** S of k edges as follows:

- If $\left|S\right| < k$, then $e^{(t)}$ is added to sample S
- Otherwise, with probability $\frac{k}{t}$, edge $e^{(t)}$ is added to sample S by replacing an uniformly at random edge from the sample

State of The Art

For **insertion-only** streams, we consider*:*

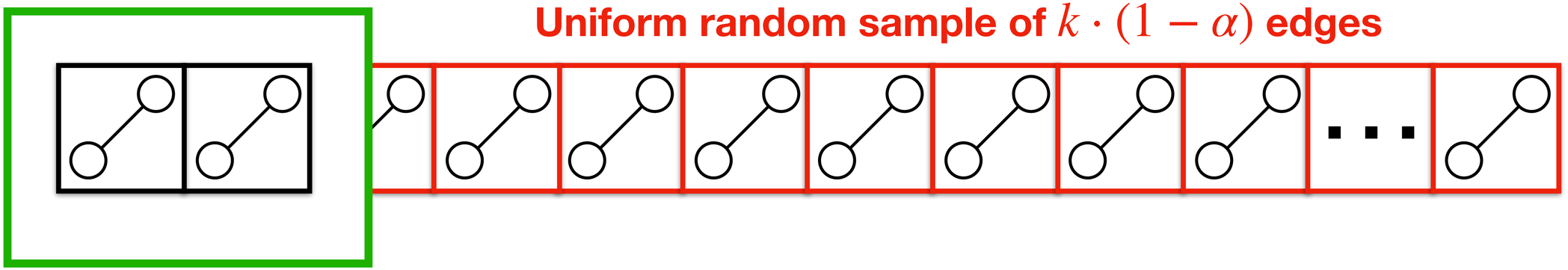
- ***Triest:*** reservoir sampling [De Stefani et al., KDD 2016]
- WRS*:* waiting room + reservoir sampling [Shin K., ICDM 2017]

State of The Art

For **insertion-only** streams, we consider*:*

- ***Triest:*** reservoir sampling [De Stefani et al., KDD 2016]
- WRS*:* waiting room + reservoir sampling [Shin K., ICDM 2017]

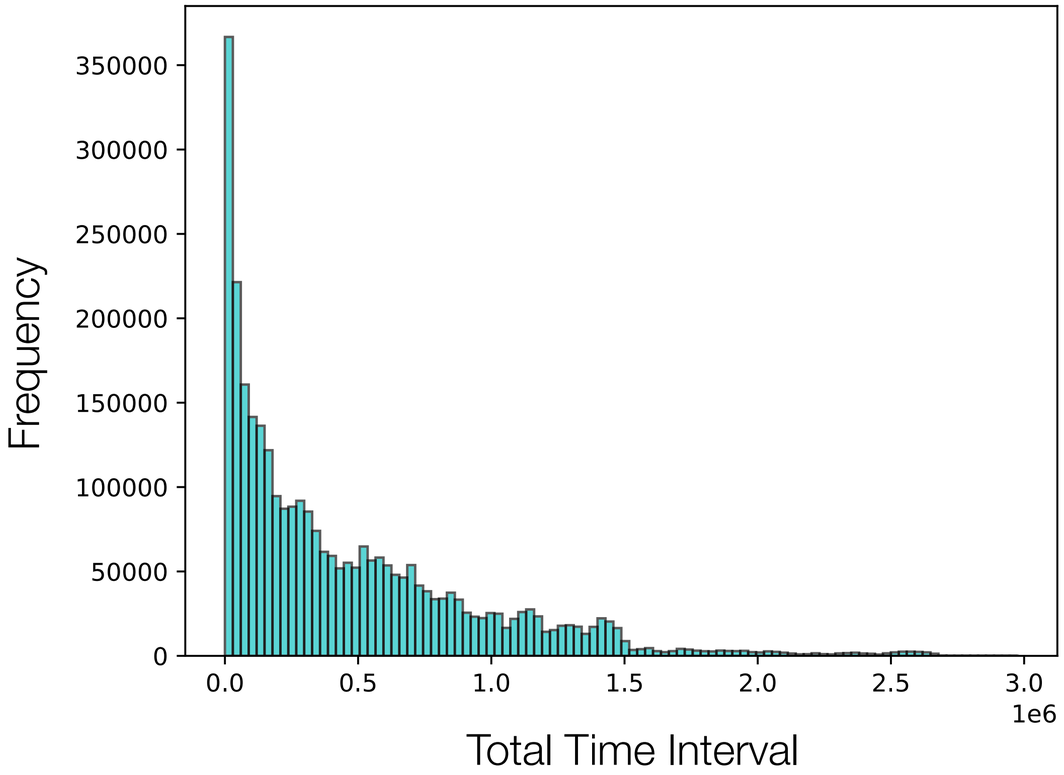
Waiting Room of $k \cdot \alpha$ edges



Memory budget k = number of edges to store

Waiting Room

Most real graph streams observe the tendency that future edges are more likely to form triangles with recent edges rather than with older edges [Shin K., ICDM 2017].

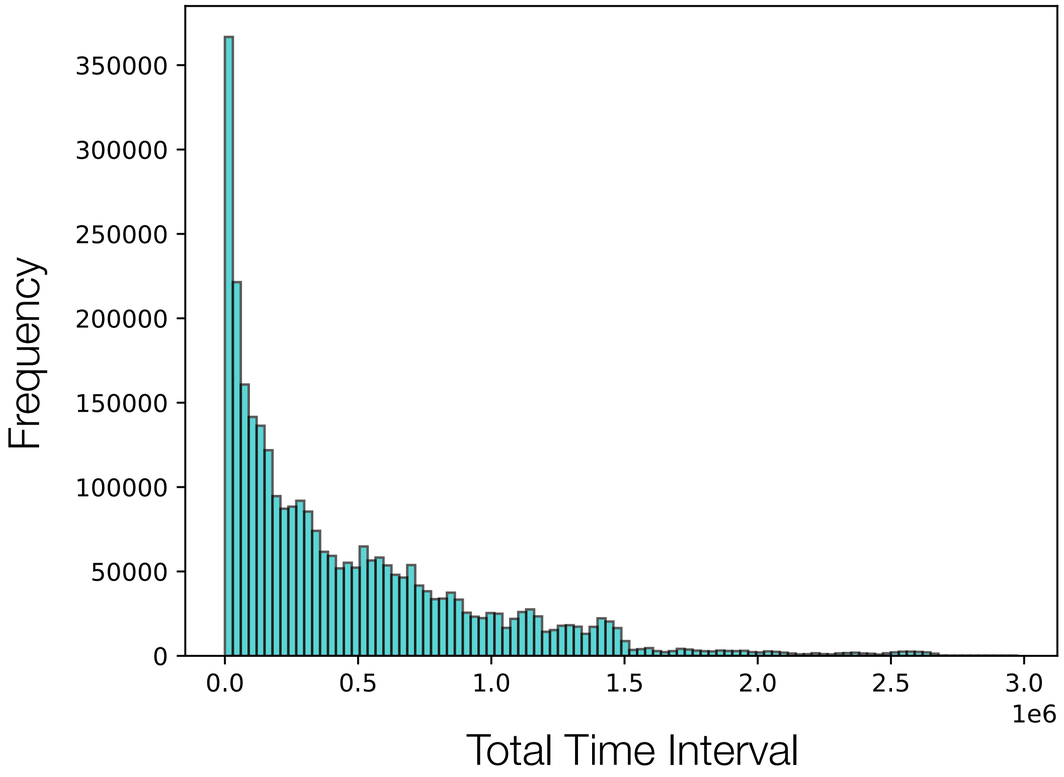


YouTube dataset

Total time interval: time between arrivals of first and last edge, for each triangle.

Waiting Room

Most real graph streams observe the tendency that future edges are more likely to form triangles with recent edges rather than with older edges [Shin K., ICDM 2017].



YouTube dataset

Total time interval: time between arrivals of first and last edge, for each triangle.

Always store the most recent edges in the waiting room W.

State of The Art

For **insertion-only** streams, we consider*:*

- ***Triest:*** reservoir sampling [De Stefani et al., KDD 2016]
- WRS*:* waiting room + reservoir sampling [Shin K., ICDM 2017]
- ***Chen:*** heavy edges set + fixed probability sampling [Chen et al., ICLR 2022] *lack of practical predictor

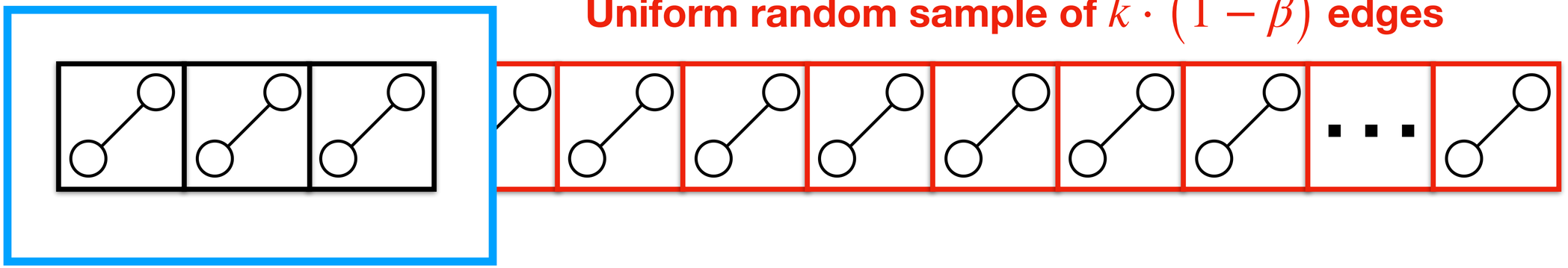
State of The Art

For **insertion-only** streams, we consider*:*

- ***Triest:*** reservoir sampling [De Stefani et al., KDD 2016]
- WRS***:*** waiting room + reservoir sampling [Shin K., ICDM 2017]
- ***Chen:*** heavy edges set + fixed probability sampling [Chen et al., ICLR 2022] *lack of practical predictor

Heavy Edges Set of $k \cdot \beta$ edges

Uniform random sample of $k \cdot \left(1 - \beta\right)$ edges

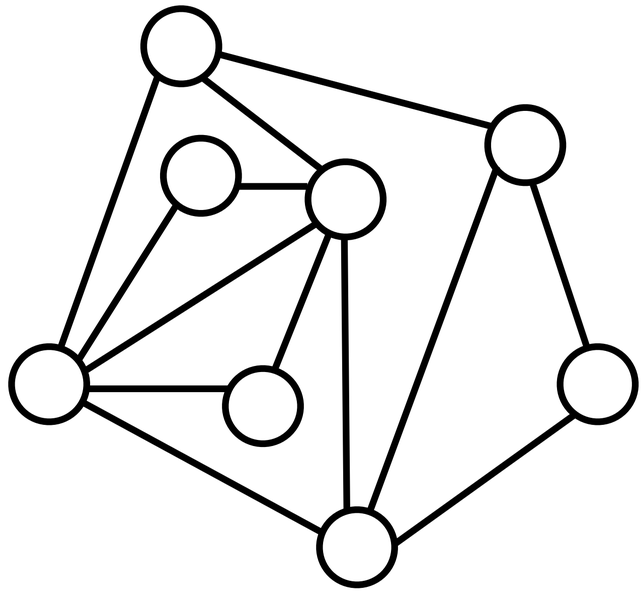


Memory budget k = number of edges to store

Heavy Edges

Heaviness of an edge e: number of triangles incident to e.

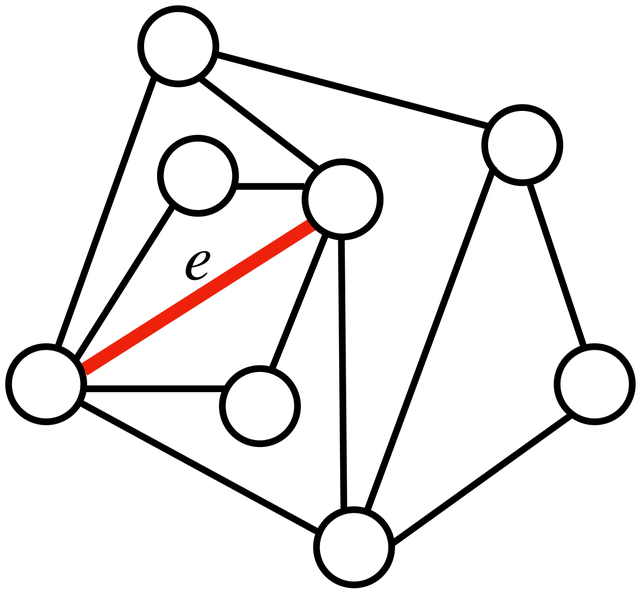
Idea: if an edge is heavy, we want to keep it in our sample.



Heavy Edges

Heaviness of an edge e: number of triangles incident to e.

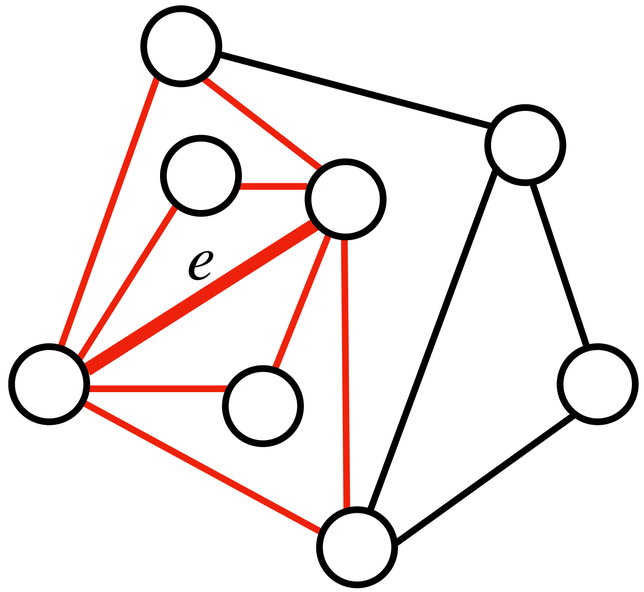
Idea: if an edge is heavy, we want to keep it in our sample.



Heavy Edges

Heaviness of an edge e: number of triangles incident to e.

Idea: if an edge is heavy, we want to keep it in our sample.

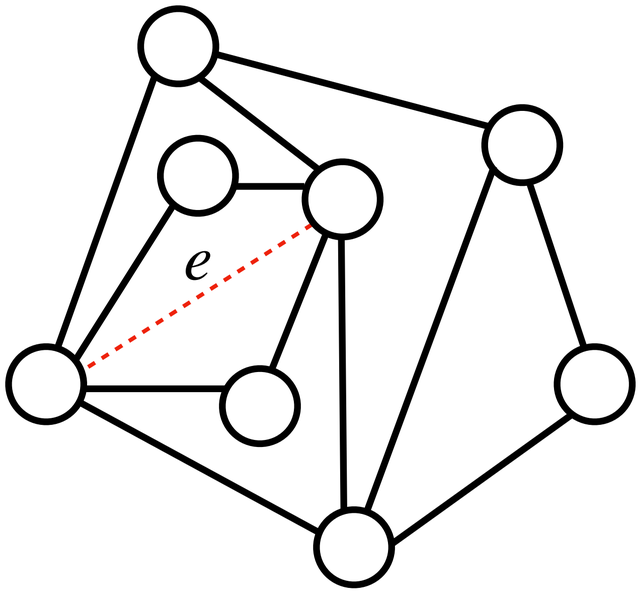


e is **heavy**, incident to
“many” triangles (4 triangles)

Heavy Edges

Heaviness of an edge e: number of triangles incident to e.

Idea: if an edge is heavy, we want to keep it in our sample.

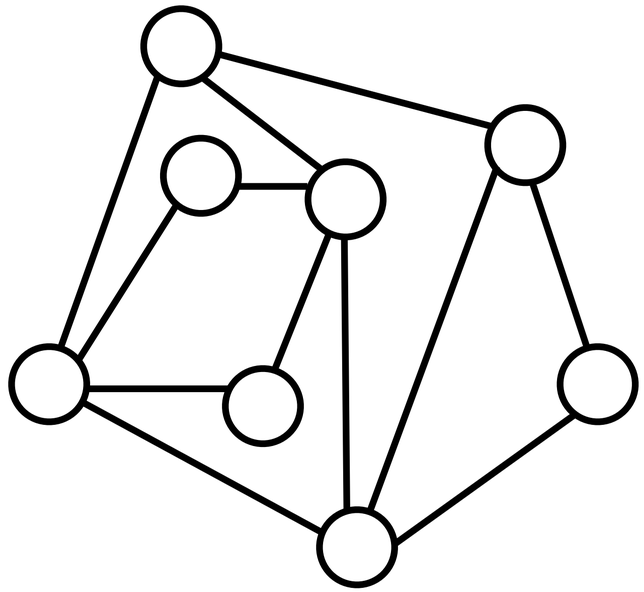


e is **heavy**, incident to
“many” triangles (4 triangles)

Heavy Edges

Heaviness of an edge e: number of triangles incident to e.

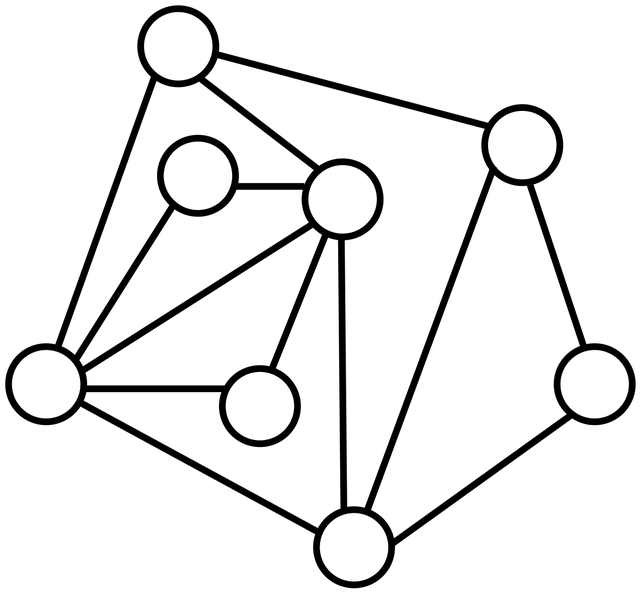
Idea: if an edge is heavy, we want to keep it in our sample.



Heavy Edges

Heaviness of an edge e: number of triangles incident to e.

Idea: if an edge is heavy, we want to keep it in our sample.



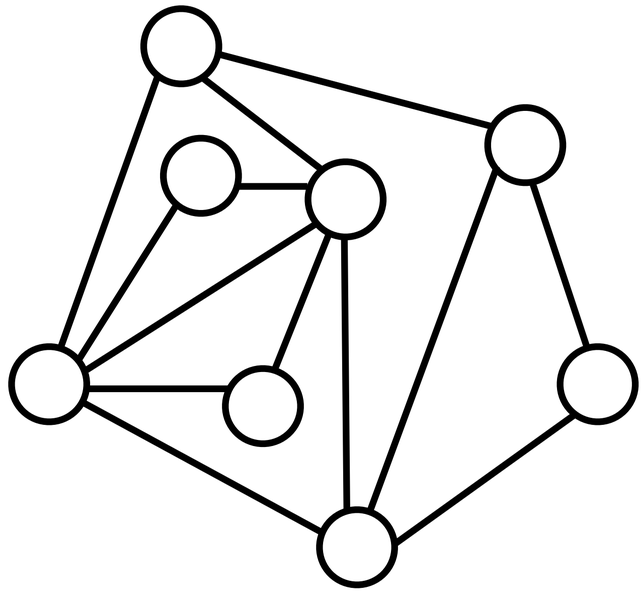
Assume to have a predictor that provides predictions about heaviness of edges.

Let $O_H : E \to \mathbb{R}^+$, where $O_H(e)$ is a measure *related* to the heaviness of each edge e.

Heavy Edges

Heaviness of an edge e: number of triangles incident to e.

Idea: if an edge is heavy, we want to keep it in our sample.



Assume to have a predictor that provides predictions about heaviness of edges.

Let $O_H : E \to \mathbb{R}^+$, where $O_H(e)$ is a measure *related* to the heaviness of each edge e.

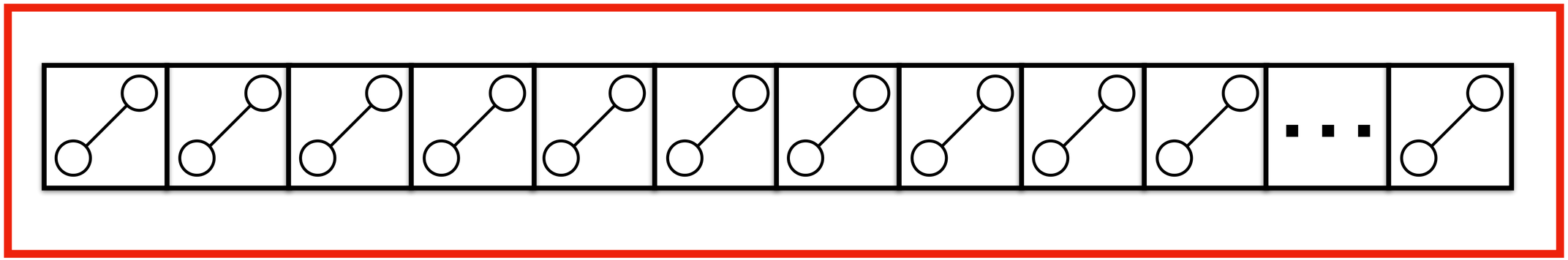
Always store the heaviest edges in the heavy edges set H.

State of The Art

For **fully-dynamic** streams, we consider*:*

- ThinkD$_{acc}$ **:** *random pairing [Shin et al., ECML PKDD 2018]*

Uniform random sample of k edges



Memory budget k = number of edges to store

Random Pairing

Random Pairing: achieve **uniform sample** in fully-dynamic streams.

Goal: **compensate** sample **deletions** using subsequent insertions. Maintain counters d_g and d_b for number of good and number of bad *uncompensated* deletions.

Random Pairing

Random Pairing: achieve **uniform sample** in fully-dynamic streams.

Goal: **compensate** sample **deletions** using subsequent insertions. Maintain counters d_g and d_b for number of good and number of bad *uncompensated* deletions.

When receiving an **edge insertion** $e^{(t)}$:

- If $d_g + d_b = 0$ (deletions compensated), then proceed by reservoir sampling
- Else, add $e^{(t)}$ to sample S with probability $\frac{d_b}{d_g + d_b}$ and decrement counters

Random Pairing

Random Pairing: achieve **uniform sample** in fully-dynamic streams.

Goal: **compensate** sample **deletions** using subsequent insertions. Maintain counters d_g and d_b for number of good and number of bad *uncompensated* deletions.

When receiving an **edge insertion** $e^{(t)}$:

- If $d_g + d_b = 0$ (deletions compensated), then proceed by reservoir sampling
- Else, add $e^{(t)}$ to sample S with probability $\frac{d_b}{d_g + d_b}$ and decrement counters

When receiving an **edge deletion** $e^{(t)}$:

- If $e^{(t)}$ is not in the sample S, then ignore it (**good** sample **deletion**)
- Else, delete $e^{(t)}$ from S (**bad** sample **deletion**)

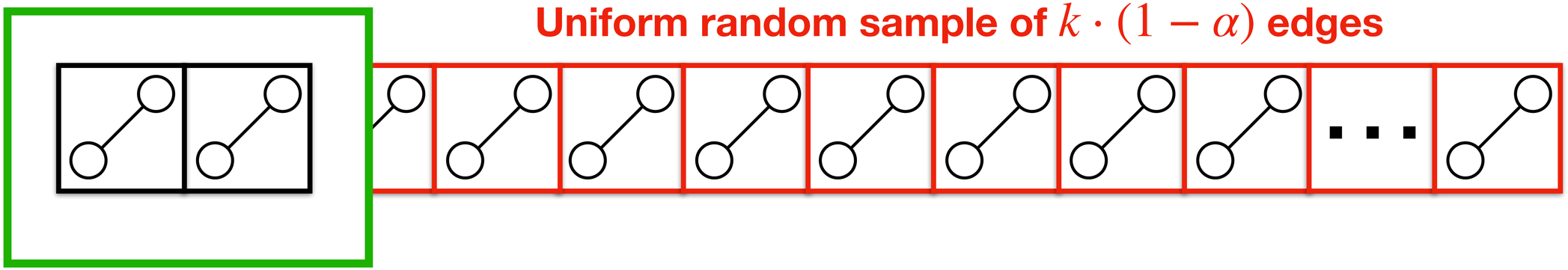
State of The Art

For **fully-dynamic** streams, we consider*:*

- ThinkD$_{acc}$ **:** *random pairing [Shin et al., ECML PKDD 2018]*
- WRS$_{del}$ **:** *waiting room + random pairing sampling [Lee et al., The VLDB Journal 2020]*

Waiting Room of $k \cdot \alpha$ edges

Uniform random sample of $k \cdot (1 - \alpha)$ edges



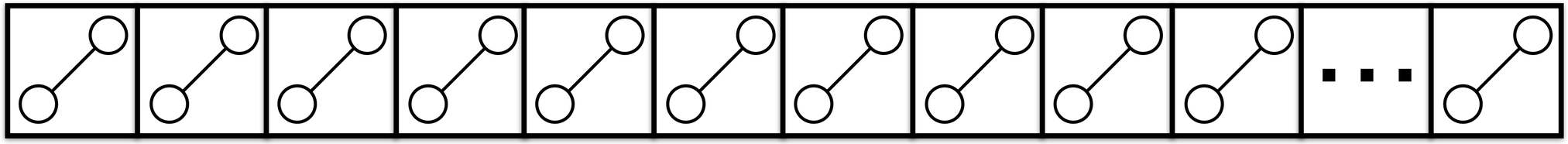
Memory budget k = number of edges to store

Our Contributions

- Our algorithm ***Tonic*** (Triangle cOuNting with predICtions) combines waiting room, heavy edges and uniform sampling for light edges.

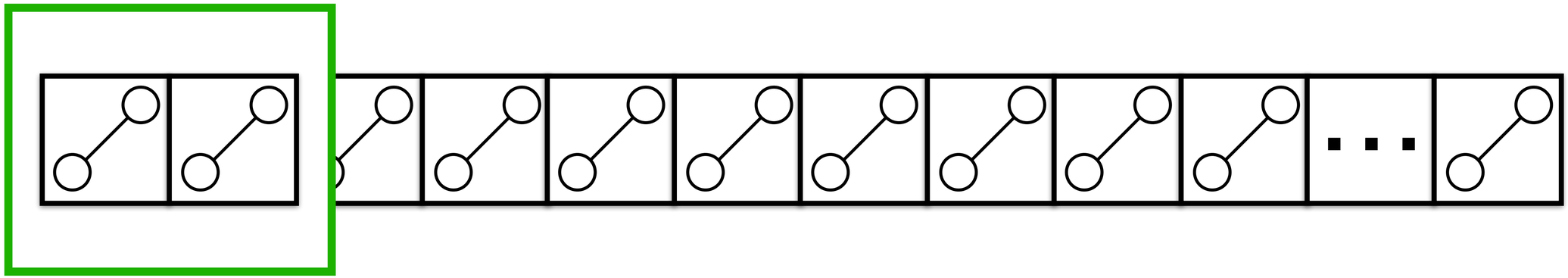
Our Contributions

Memory budget k = number of edges to store



Our Contributions

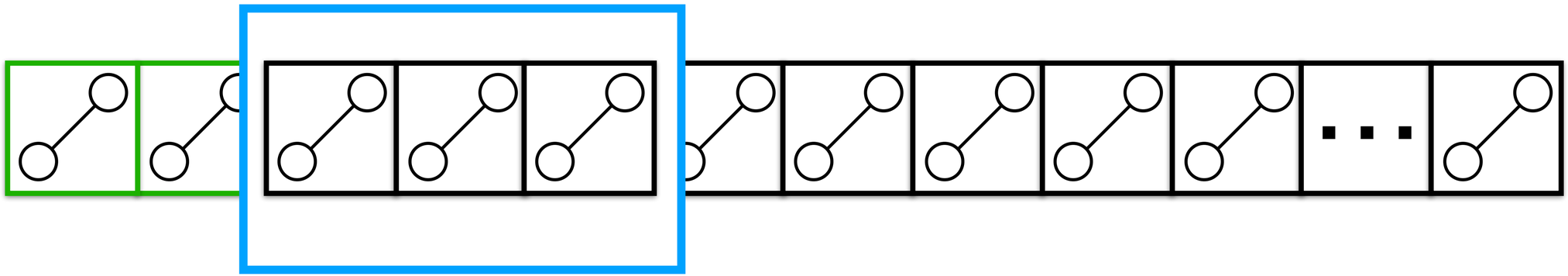
Memory budget k = number of edges to store



Store $k \cdot \alpha$ most recent edges in **waiting room** W

Our Contributions

Memory budget k = number of edges to store

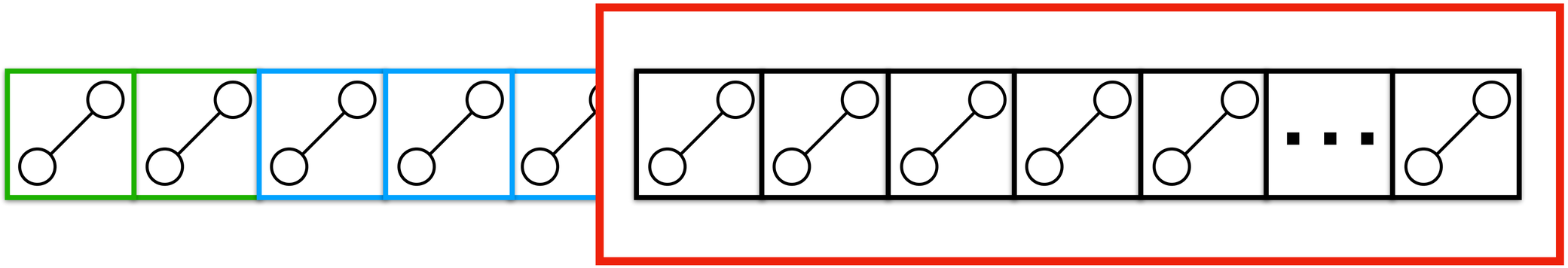


Store $k \cdot \alpha$ most recent edges in **waiting room** W

Store $k \cdot (1 - \alpha) \cdot \beta$ heaviest edges (according to the predictor) in **heavy edge set** H

Our Contributions

Memory budget k = number of edges to store



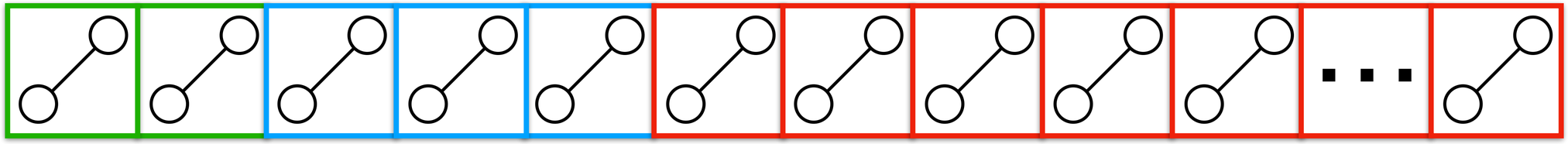
Store $k \cdot \alpha$ most recent edges in **waiting room** W

Store $k \cdot (1-\alpha) \cdot \beta$ heaviest edges (according to the predictor) in **heavy edge set** H

Store a **uniform random sample** S of $k \cdot (1-\alpha) \cdot (1-\beta)$ **light edges**

Our Contributions

Memory budget k = number of edges to store



We empirically fix:
α = **0.05**, and β = **0.2**

Store $k \cdot \alpha$ most recent edges in **waiting room** W

Store $k \cdot (1-\alpha) \cdot \beta$ heaviest edges (according to the predictor) in **heavy edge set** H

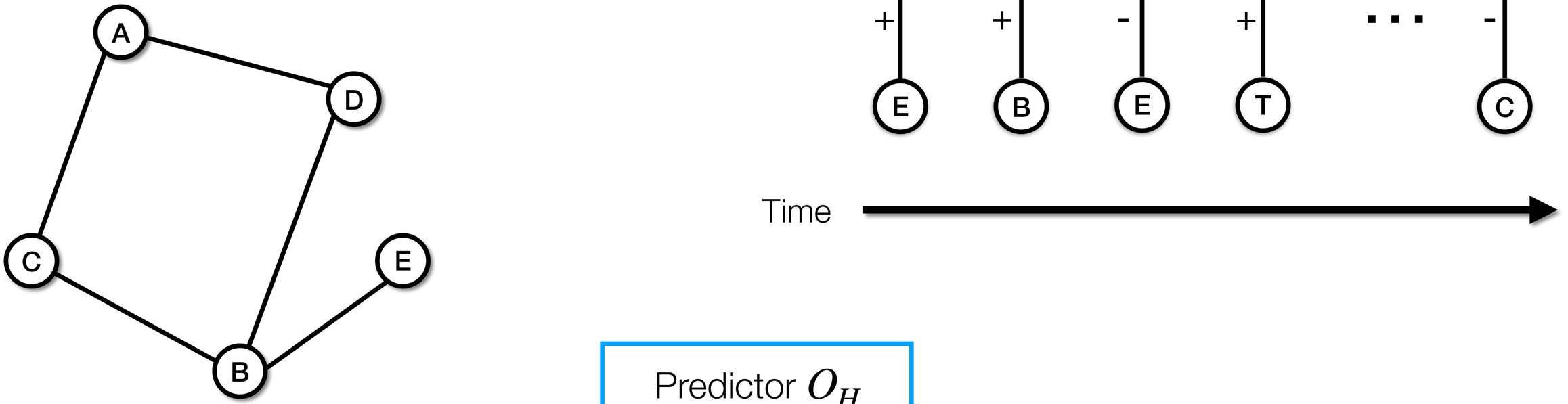
Store a **uniform random sample** S of $k \cdot (1-\alpha) \cdot (1-\beta)$ **light edges**

Our Contributions

- Our algorithm ***Tonic*** (Triangle cOuNting with predICtions) combines waiting room, heavy edges and uniform sampling for light edges.
- *Tonic* provides **fast** and **accurate** approximations of global (and local) triangles in graph streams
- We propose a **simple** and application-independent **predictor**, based on the **degree** of the nodes
- Extensive experimental evaluation shows improvements and scalability of *Tonic*

Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.

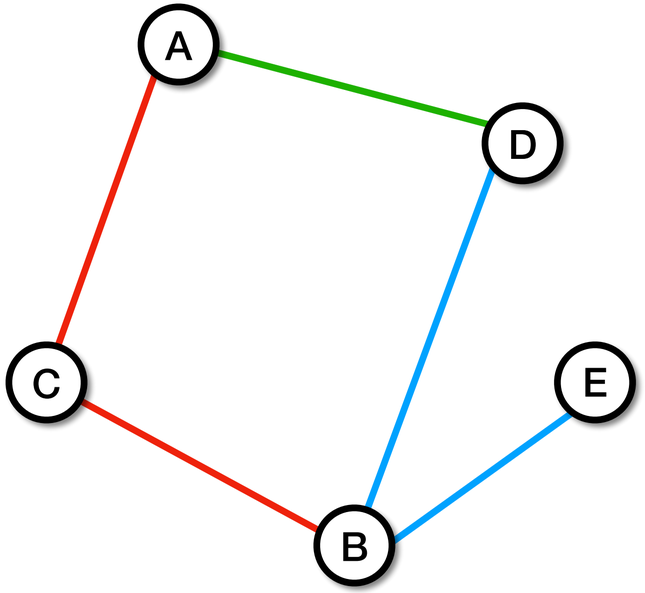


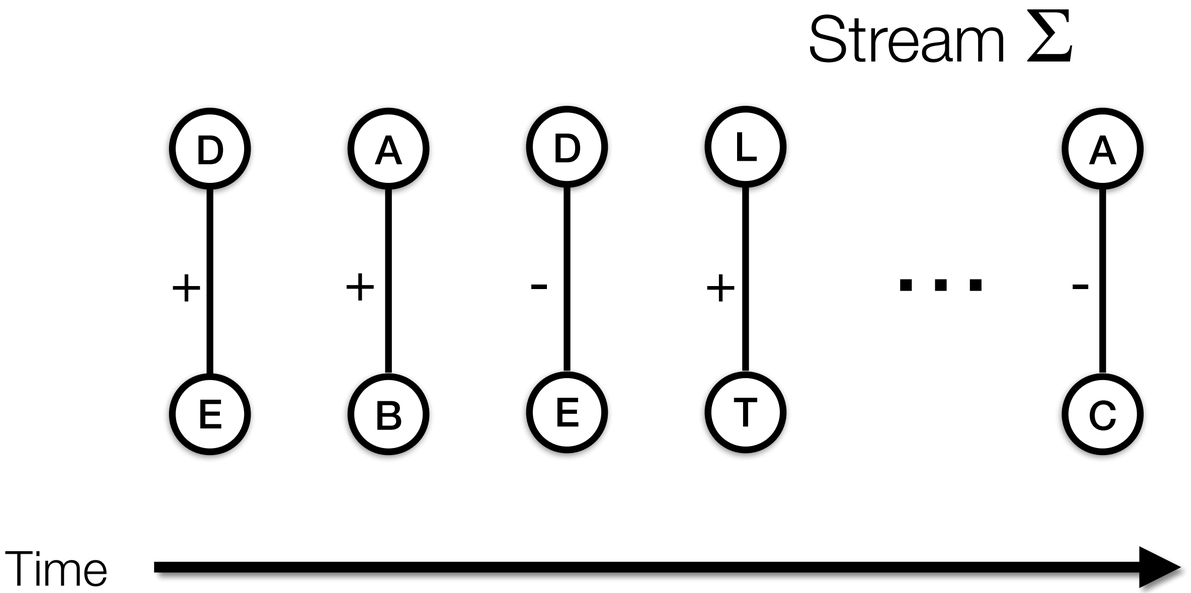
Predictor O_H

Current Sample

Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.





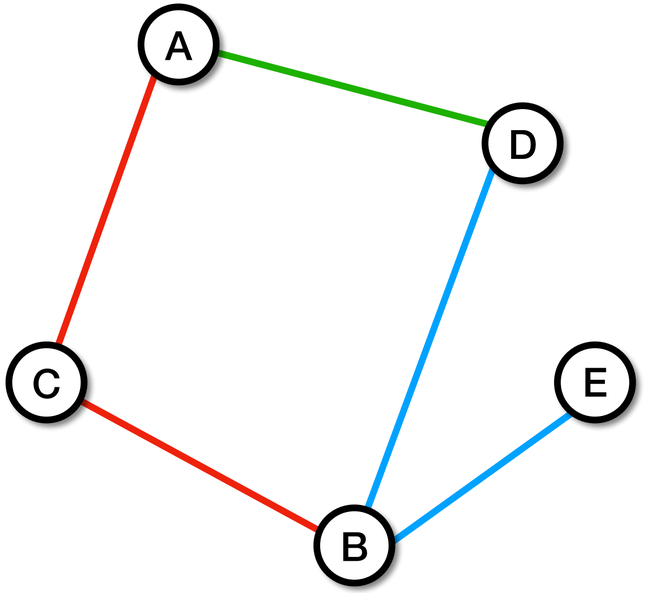
Predictor O_H

Current Sample $W^{(t)} \cup H^{(t)} \cup S^{(t)}$

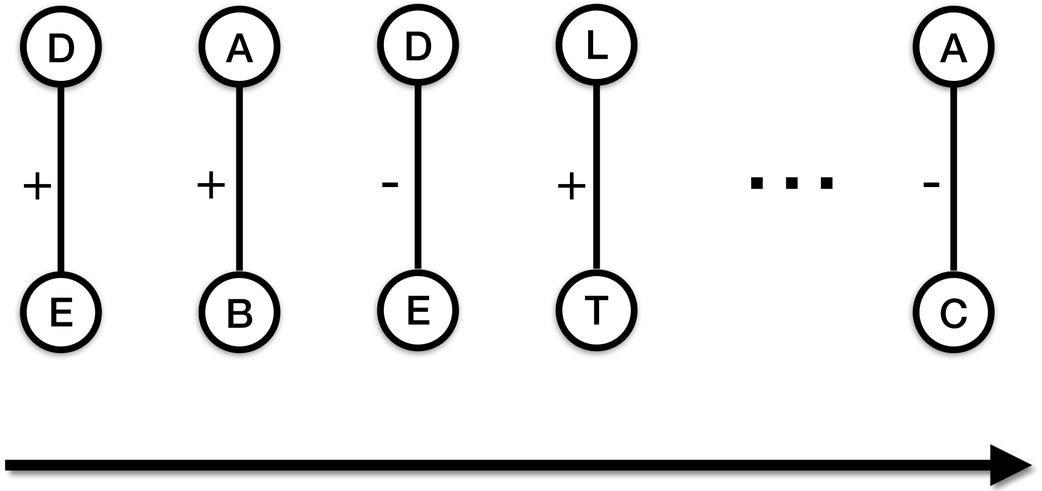
Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.

Count number of triangles closed or deleted by $e^{(t)}$ in our sample.



Stream Σ



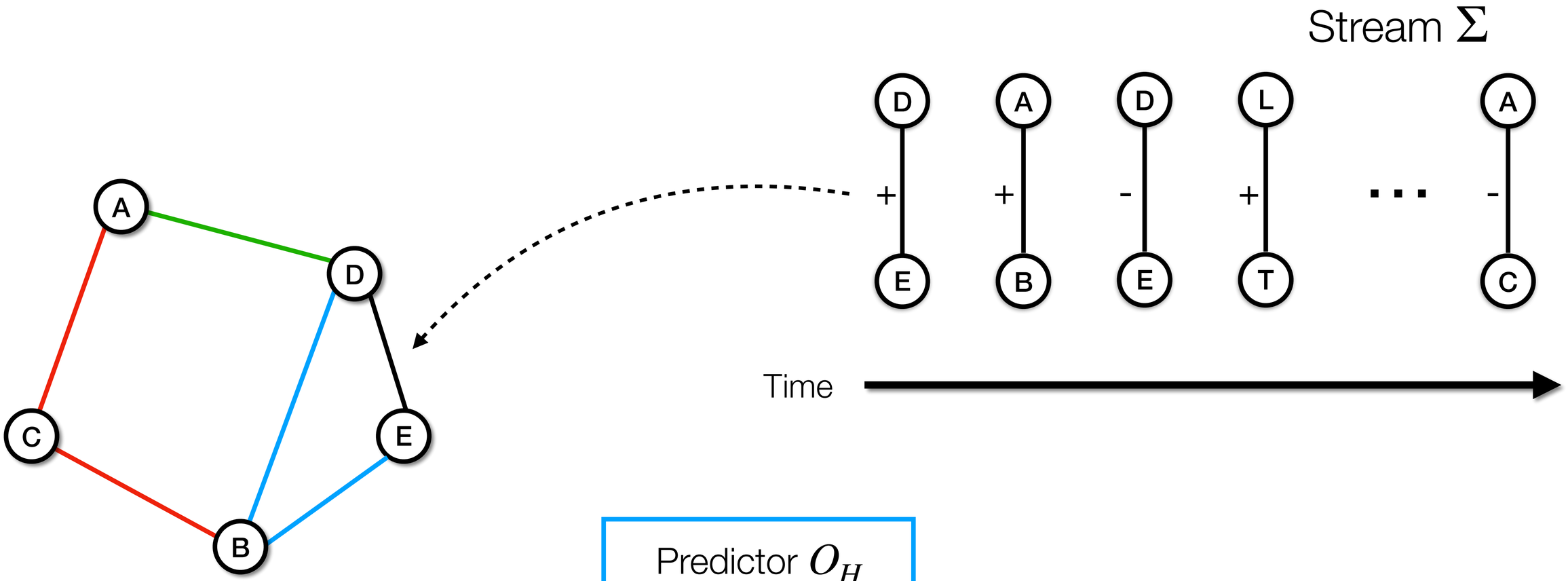
Predictor O_H

Current Sample $W^{(t)} \cup H^{(t)} \cup S^{(t)}$

Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.

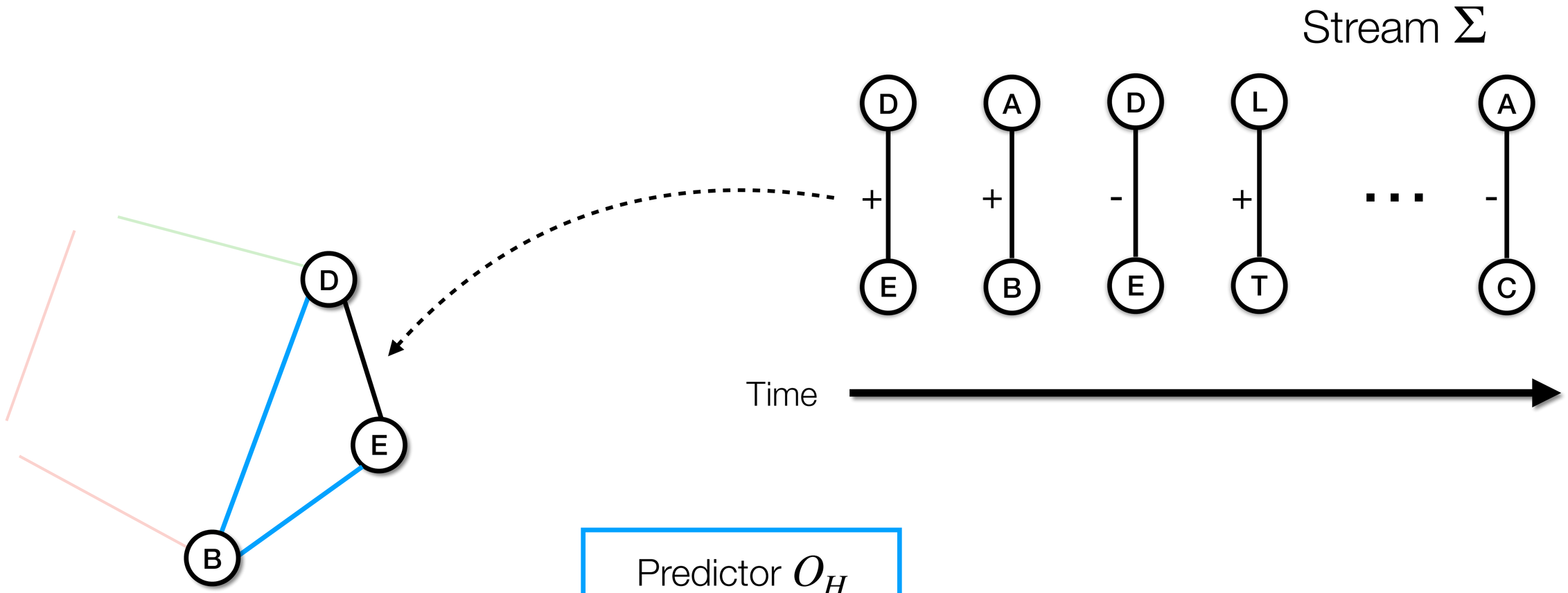
Count number of triangles closed or deleted by $e^{(t)}$ in our sample.



Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.

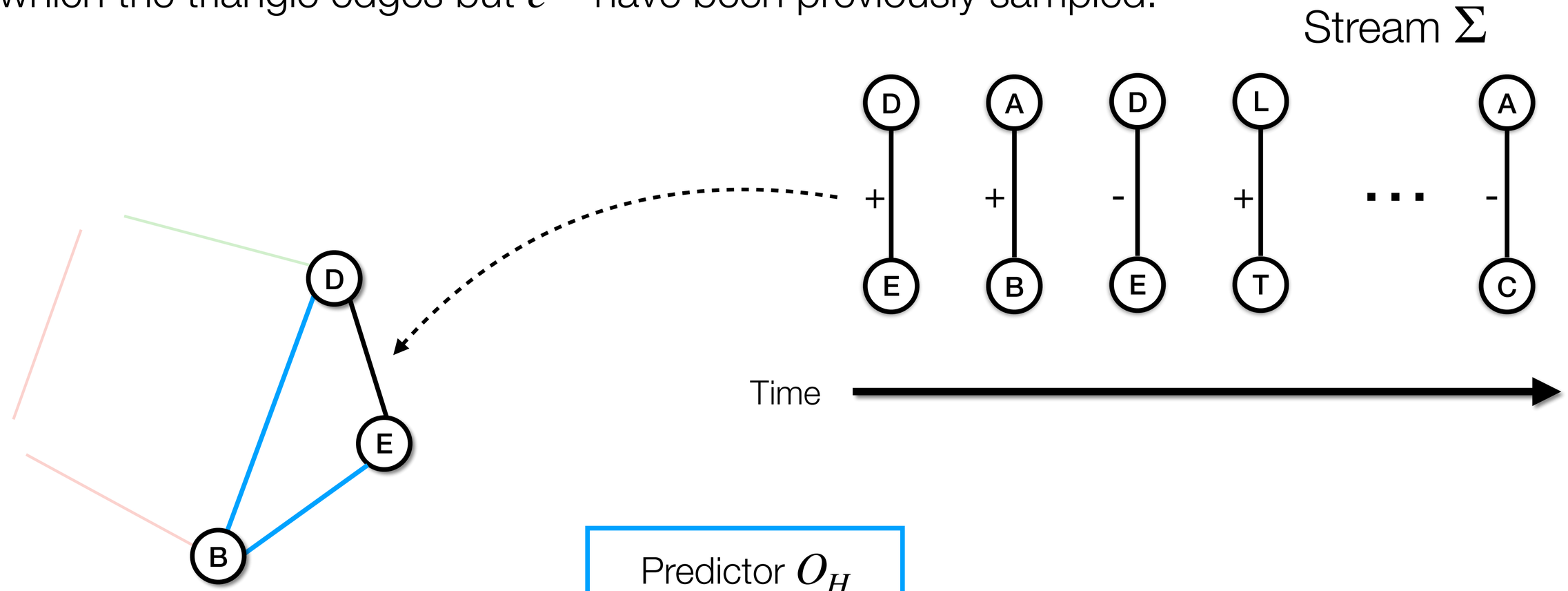
Count number of triangles closed or deleted by $e^{(t)}$ in our sample



Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.

Each triangle counted or deleted in the sample is scaled by the inverse of the probability with which the triangle edges but $e^{(t)}$ have been previously sampled.

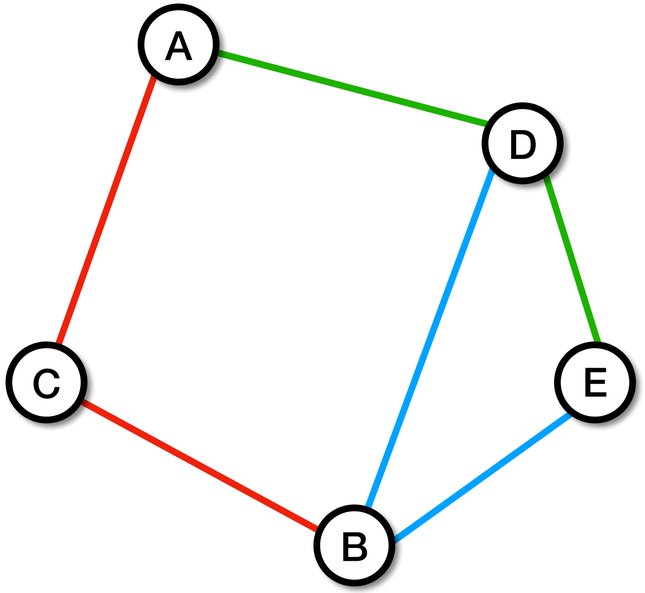


Current Sample $W^{(t)} \cup H^{(t)} \cup S^{(t)}$

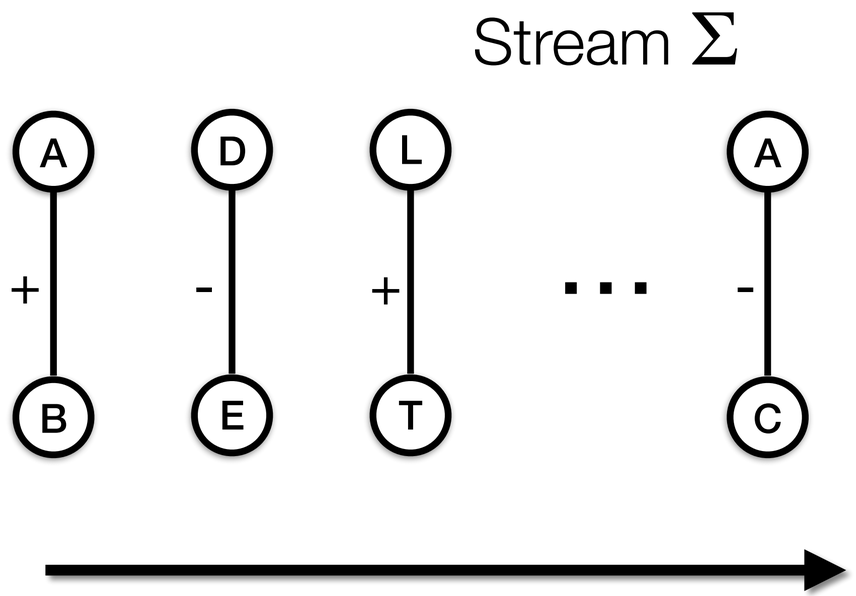
Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.

Finally, $e^{(t)}$ is inserted in the waiting room.



Current Sample $W^{(t)} \cup H^{(t)} \cup S^{(t)}$

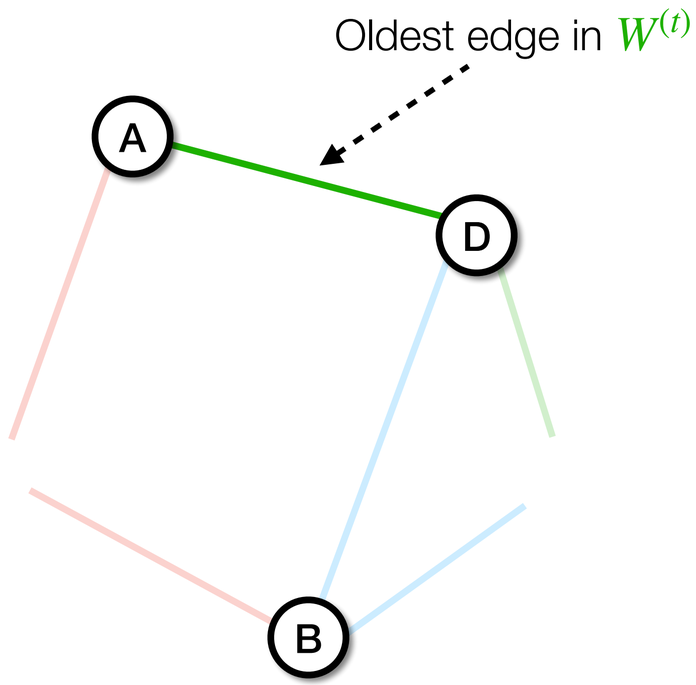


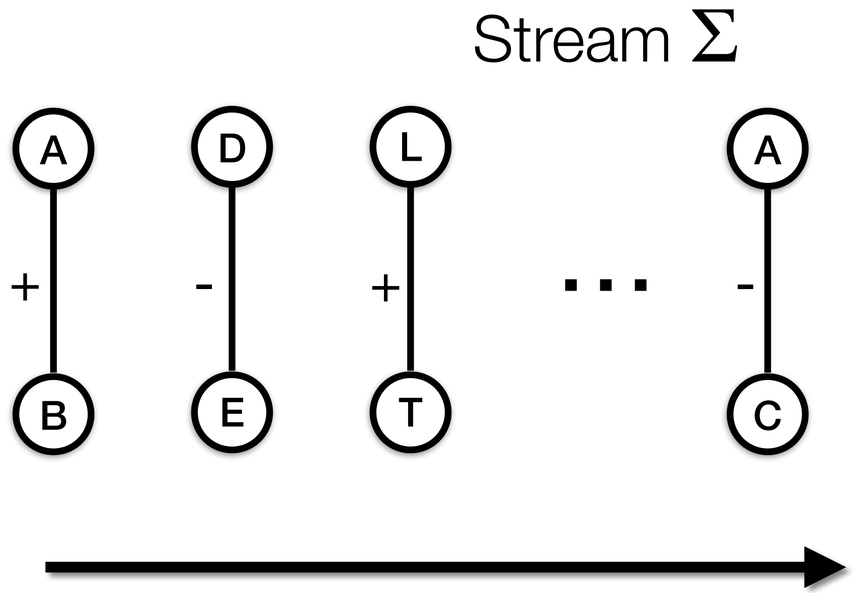
Predictor O_H

Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.

If $W^{(t)}$ is full, pop the oldest edge, and sample the lightest (according to the predictor) between the popped edge and edges in $H^{(t)}$.





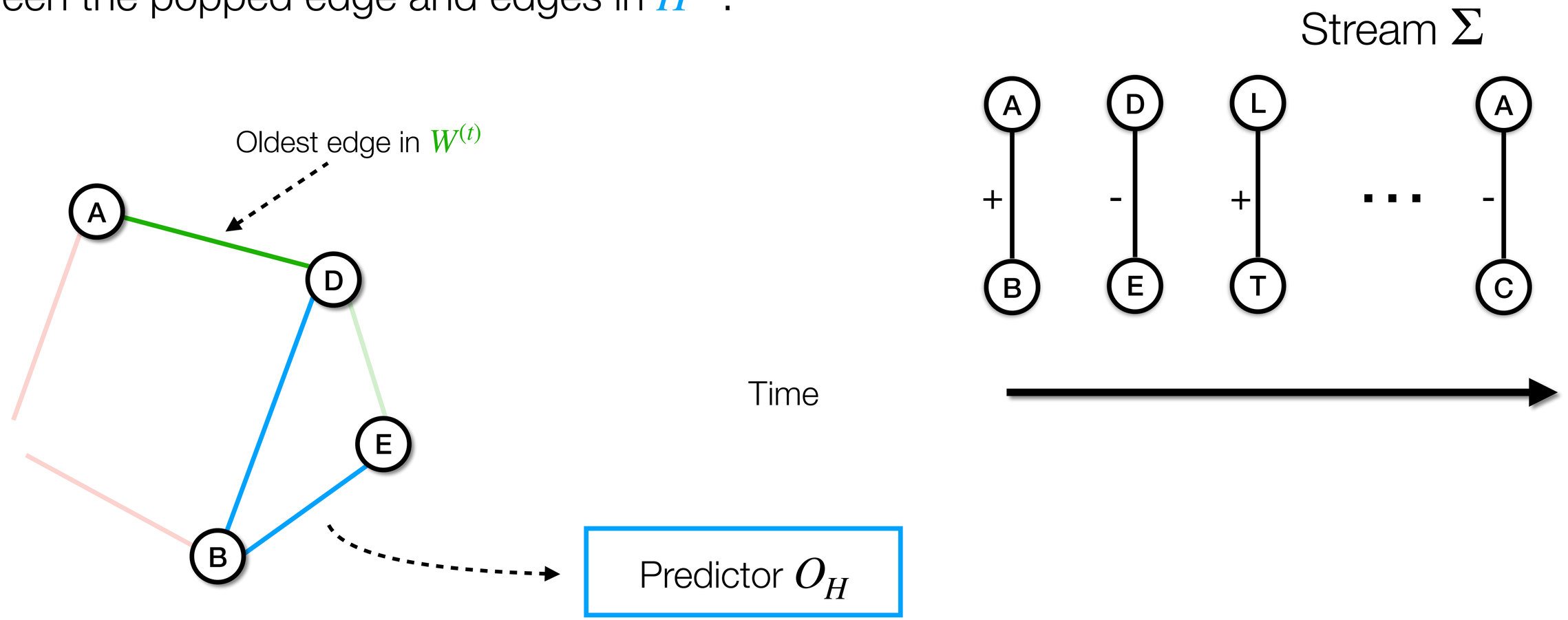
Predictor O_H

Current Sample $W^{(t)} \cup H^{(t)} \cup S^{(t)}$

Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.

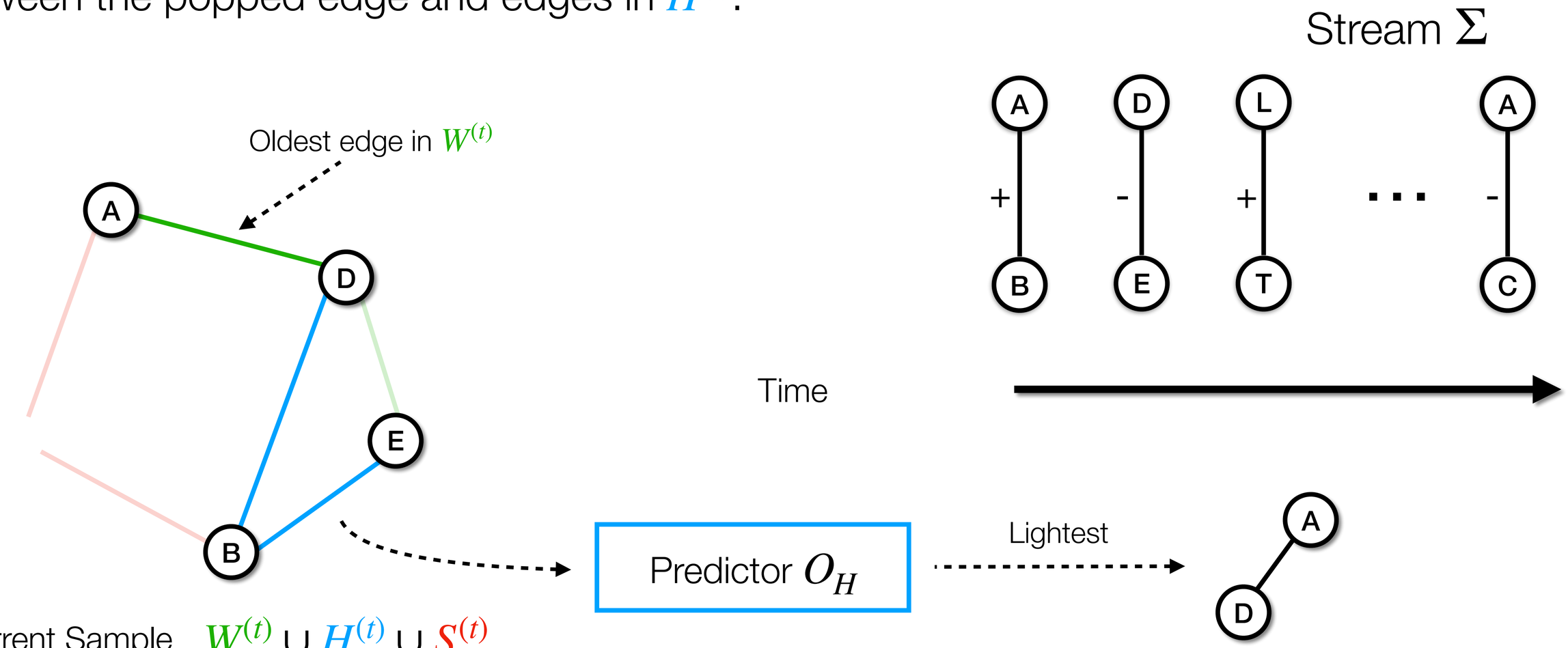
If $W^{(t)}$ is full, pop the oldest edge, and sample the lightest (according to the predictor) between the popped edge and edges in $H^{(t)}$.



Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.

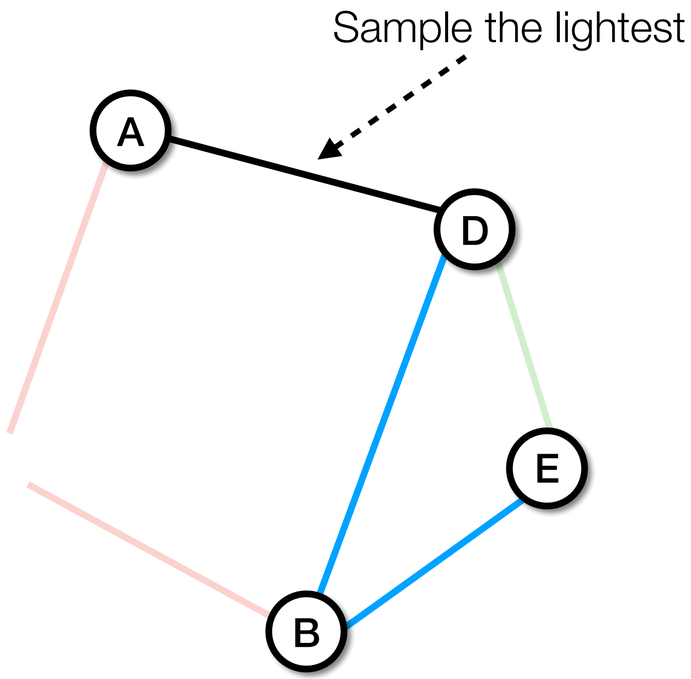
If $W^{(t)}$ is full, pop the oldest edge, and sample the lightest (according to the predictor) between the popped edge and edges in $H^{(t)}$.

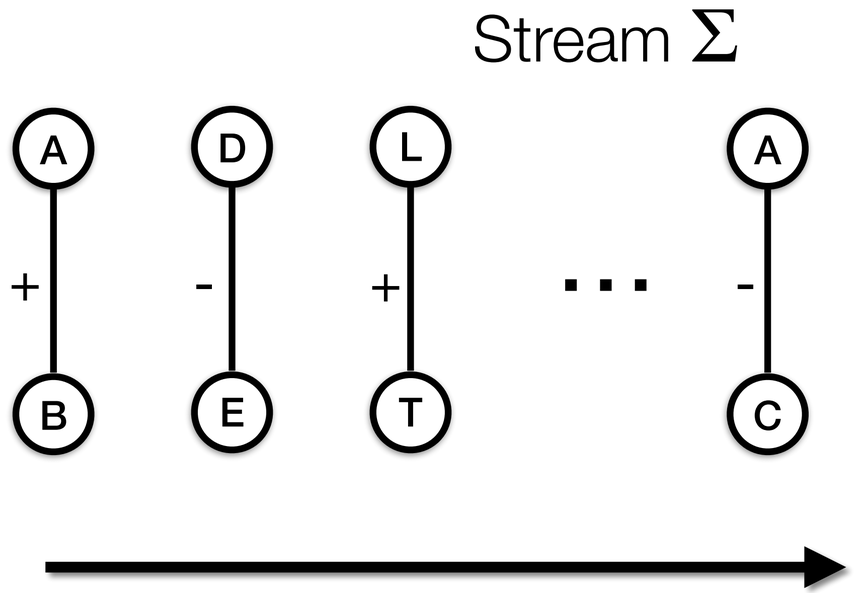


Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.

If $W^{(t)}$ is full, pop the oldest edge, and sample the lightest (according to the predictor) between the popped edge and edges in $H^{(t)}$.





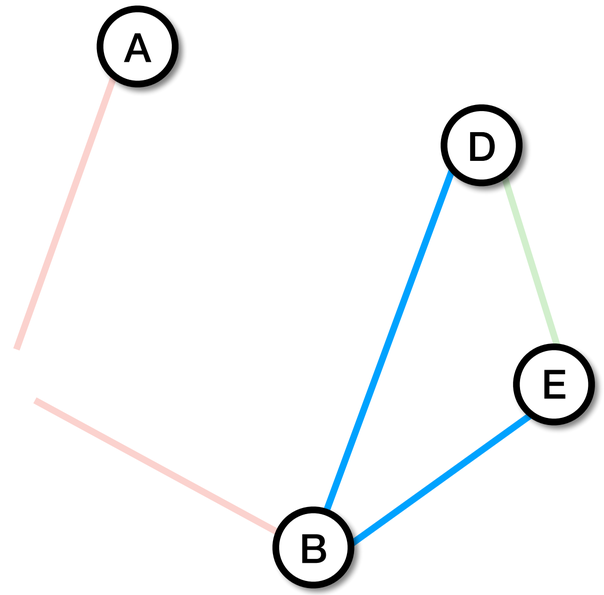
Predictor O_H

Current Sample $W^{(t)} \cup H^{(t)} \cup S^{(t)}$

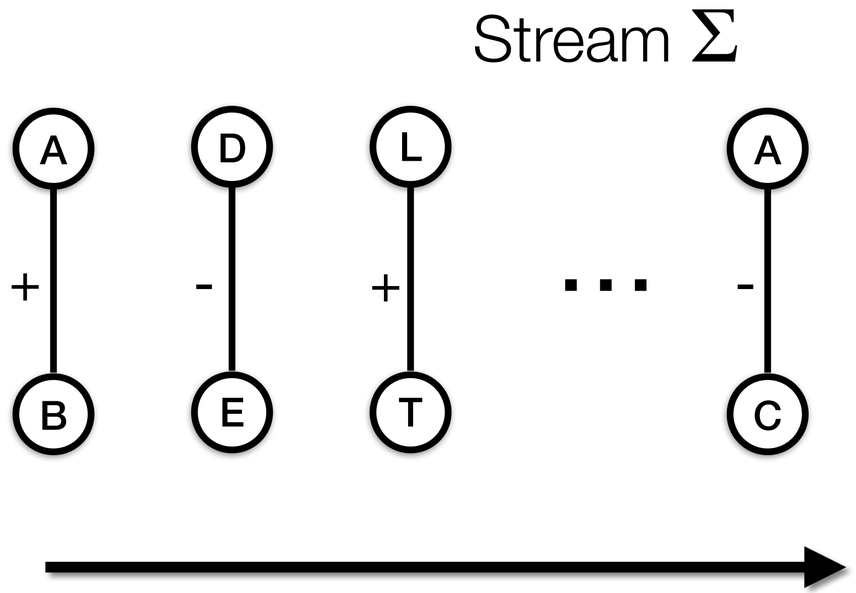
Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.

If $W^{(t)}$ is full, pop the oldest edge, and sample the lightest (according to the predictor) between the popped edge and edges in $H^{(t)}$.



Current Sample $W^{(t)} \cup H^{(t)} \cup S^{(t)}$

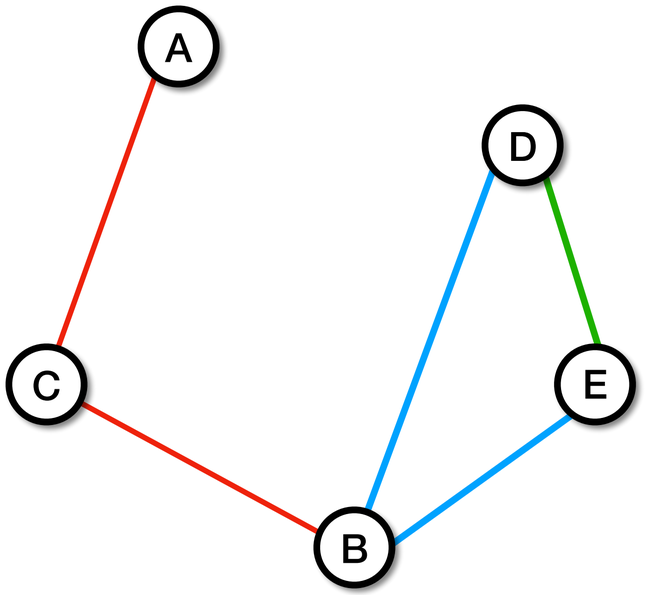


Predictor O_H

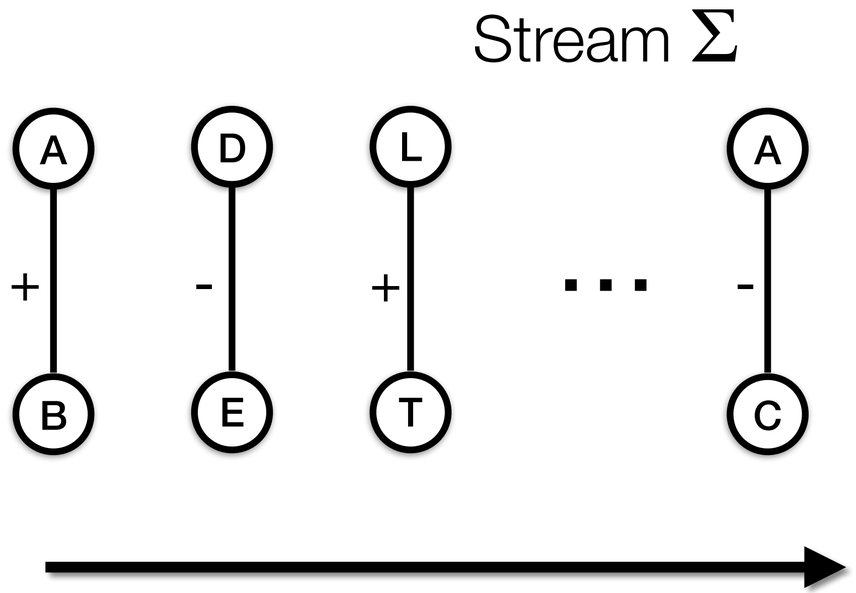
Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.

If $W^{(t)}$ is full, pop the oldest edge, and sample the lightest (according to the predictor) between the popped edge and edges in $H^{(t)}$.



Current Sample $W^{(t)} \cup H^{(t)} \cup S^{(t)}$



Predictor O_H

Tonic: theoretical analysis

Theorem (Unbiasedness of estimates): let $T^{(t)}$ be the true number of global triangles. Then, *Tonic* outputs $\hat{T}^{(t)}$ such that:

$$\mathbb{E}\left[\hat{T}^{(t)}\right] = T^{(t)}, \forall\ t \geq 0$$

Tonic: theoretical analysis

We prove that *useful* predictions in *Tonic* leads to better estimates than using *WRS* alone.

Tonic: theoretical analysis

We prove that *useful* predictions in *Tonic* leads to better estimates than using *WRS* alone.

Consider:

- *WRS* sampling edges leaving the waiting room with probability p
- *Tonic* sampling light edges with probability $p' < p$
- We define an edge e as **heavy** if e appears in $\geq \rho$ triangles (otherwise, **light**)
- **Errors of predictions:** heavy edges involved in $\geq \rho \cdot c$ triangles, light edges involved in $\leq \rho/c$ triangles, for some $c \geq 1$. For edges with heaviness $\in \left[\rho/c \ , \ \rho \cdot c\right]$, the predictor can make arbitrarily wrong choices.

Tonic: theoretical analysis

Let T_H be the total number of triangles in which **heavy** edges appear, and T_L be the total number of triangles in which **light** edges appear.

Tonic: theoretical analysis

Let T_H be the total number of triangles in which **heavy** edges appear, and T_L be the total number of triangles in which **light** edges appear.

Proposition (informal): the variance of the estimates of *Tonic* is less than the variance of the estimates of *WRS* if:

$$T_H > 3 \, \frac{(1/p'^2 - 1/p^2) + c\rho(1/p' - 1/p)}{(1/p - 1)(3 + 4\rho/c)} \cdot T_L$$

Tonic: theoretical analysis

Let T_H be the total number of triangles in which **heavy** edges appear, and T_L be the total number of triangles in which **light** edges appear.

Proposition (informal): the variance of the estimates of *Tonic* is less than the variance of the estimates of *WRS* if:

$$T_H > 3 \, \frac{(1/p'^2 - 1/p^2) + c\rho(1/p' - 1/p)}{(1/p - 1)(3 + 4\rho/c)} \cdot T_L$$

Representative values: if $p' = 0.09 < p = 0.1$, $\rho = 100$ and $c = 1.5$, then the bound above corresponds to T_H being at least one fifth of the overall number of triangles.

Edge Heaviness Predictor

The predictor used by *Tonic* could be implemented by a machine learning model that may consider information other than the graph.

Edge Heaviness Predictor

The predictor used by *Tonic* could be implemented by a machine learning model that may consider information other than the graph.

In our experiments we consider:

- *OracleExact*, storing the value of $\Delta(e)$ for top 10% ($m/10$) **heaviest edges** e, and 0 for the remaining ones

OracleExact

u_1	v_1	$\Delta(\{u_1, v_1\})$
u_2	v_1	$\Delta(\{u_2, v_1\})$
u_3	v_3	$\Delta(\{u_3, v_3\})$
u_2	v_4	$\Delta(\{u_4, v_4\})$
⋮	⋮	⋮
u_1	v_5	$\Delta(\{u_1, v_5\})$

Edge Heaviness Predictor

The predictor used by *Tonic* could be implemented by a machine learning model that may consider information other than the graph.

In our experiments we consider:

- *OracleExact*
- *Oracle-noWR*, storing the value of $\Delta'(e)$ obtained by subtracting to $\Delta(e)$ the number of triangles for which e is in the **waiting room**, for top 10% edges

OracleExact

u_1	v_1	$\Delta(\{u_1, v_1\})$
u_2	v_1	$\Delta(\{u_2, v_1\})$
u_3	v_3	$\Delta(\{u_3, v_3\})$
u_2	v_4	$\Delta(\{u_4, v_4\})$
⋮	⋮	⋮
u_1	v_5	$\Delta(\{u_1, v_5\})$

Oracle-noWR

u_2	v_1	$\Delta'(\{u_2, v_1\})$
u_3	v_3	$\Delta'(\{u_3, v_3\})$
u_2	v_4	$\Delta'(\{u_2, v_4\})$
u_7	v_7	$\Delta'(\{u_7, v_7\})$
⋮	⋮	⋮
u_1	v_4	$\Delta'(\{u_1, v_4\})$

Edge Heaviness Predictor

The predictor used by *Tonic* could be implemented by a machine learning model that may consider information other than the graph.

In our experiments we consider:

- *OracleExact*
- *Oracle-noWR*, storing the value of $\Delta'(e)$ obtained by subtracting to $\Delta(e)$ the number of triangles for which e is in the **waiting room**, for top 10% edges

OracleExact

u_1	v_1	$\Delta(\{u_1, v_1\})$
u_2	v_1	$\Delta(\{u_2, v_1\})$
u_3	v_3	$\Delta(\{u_3, v_3\})$
u_2	v_4	$\Delta(\{u_4, v_4\})$
⋮	⋮	⋮
u_1	v_5	$\Delta(\{u_1, v_5\})$

Oracle-noWR

u_2	v_1	$\Delta'(\{u_2, v_1\})$
u_3	v_3	$\Delta'(\{u_3, v_3\})$
u_2	v_4	$\Delta'(\{u_2, v_4\})$
u_7	v_7	$\Delta'(\{u_7, v_7\})$
⋮	⋮	⋮
u_1	v_4	$\Delta'(\{u_1, v_4\})$

Impractical Predictors

Edge Heaviness Predictor

The predictor used by *Tonic* could be implemented by a machine learning model that may consider information other than the graph.

In our experiments we consider:

- *OracleExact*
- *Oracle-noWR*
- *MinDegreePredictor*, storing the $\bar{n}$ highest-degree **nodes** and **degrees**. Given an edge $e = \{u, v\}$, *MinDegreePredictor* outputs:
  $O_H\left(\{u, v\}\right) = \min\left(deg(u), deg(v)\right)$

OracleExact

u_1	v_1	$\Delta\left(\{u_1, v_1\}\right)$
u_2	v_1	$\Delta\left(\{u_2, v_1\}\right)$
u_3	v_3	$\Delta\left(\{u_3, v_3\}\right)$
u_2	v_4	$\Delta\left(\{u_4, v_4\}\right)$
⋮	⋮	⋮
u_1	v_5	$\Delta\left(\{u_1, v_5\}\right)$

Oracle-noWR

u_2	v_1	$\Delta'\left(\{u_2, v_1\}\right)$
u_3	v_3	$\Delta'\left(\{u_3, v_3\}\right)$
u_2	v_4	$\Delta'\left(\{u_2, v_4\}\right)$
u_7	v_7	$\Delta'\left(\{u_7, v_7\}\right)$
⋮	⋮	⋮
u_1	v_4	$\Delta'\left(\{u_1, v_4\}\right)$

MinDegreePredictor

v_1	$deg(v_1)$
u_1	$deg(u_1)$
v_2	$deg(v_2)$
⋮	⋮
v_3	$deg(v_3)$

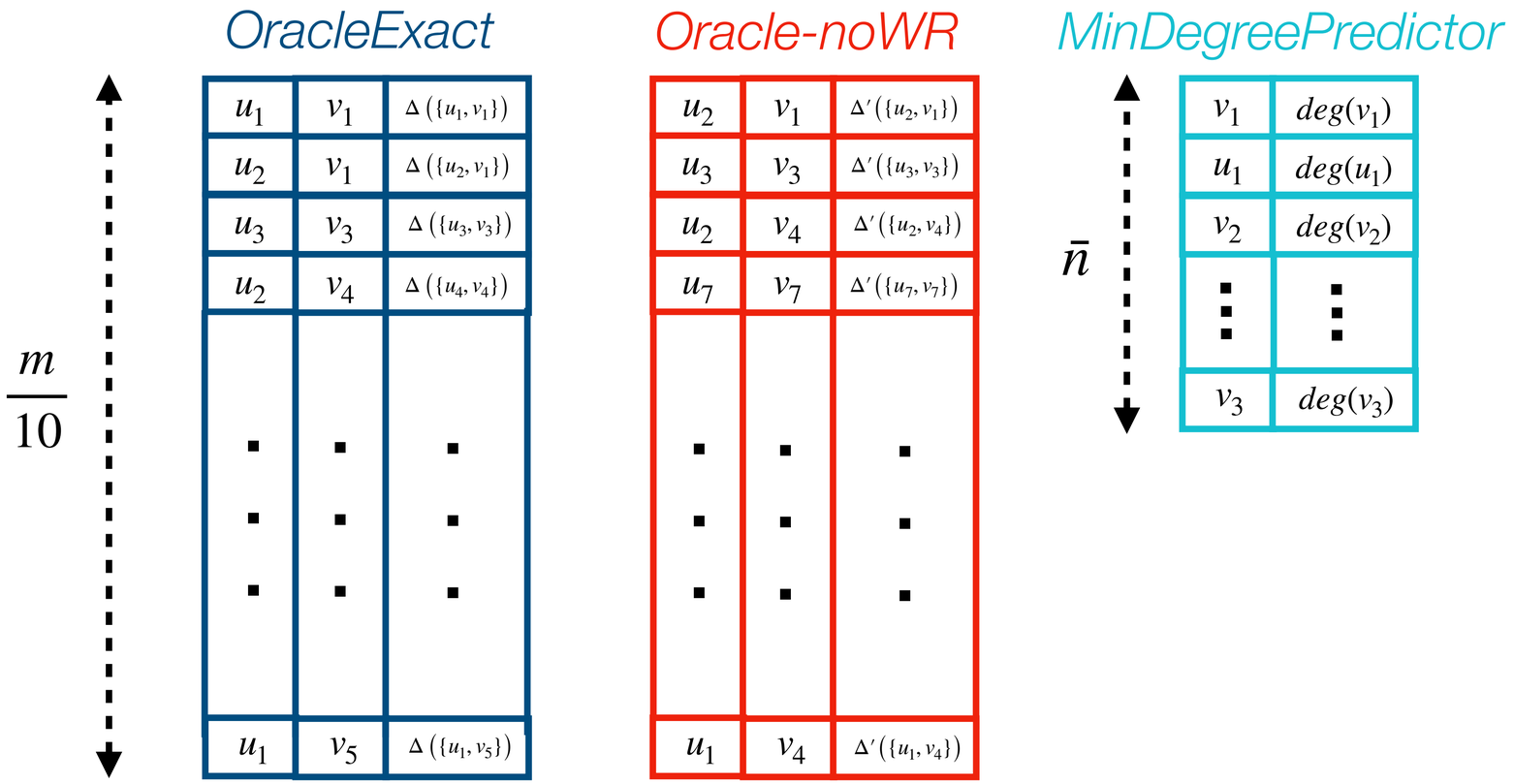
Edge Heaviness Predictor

The predictor used by *Tonic* could be implemented by a machine learning model that may consider information other than the graph.

In our experiments we consider:

- *OracleExact*
- *Oracle-noWR*
- *MinDegreePredictor*

In practice, $\bar{n} \ll m/10$!



Experimental Evaluation

We consider real-world **single graph streams**, from social network, citation network.

Compare *Tonic* with state-of-the-art: algorithms provided with same memory budget k

TABLE I
DATASETS' STATISTICS: NUMBER n OF NODES; NUMBER m OF EDGES; NUMBER T OF TRIANGLES

Dataset	n	m	T
Single Graphs			
Edit EN Wikibooks	$133k$	$386k$	$178k$
SOC YouTube Growth	$3.2M$	$9.3M$	$12.3M$
Cit US Patents	$3.7M$	$16.5M$	$7.5M$
Actors Collaborations	$382k$	$15M$	$346.8M$
Stackoverflow	$2.5M$	$28.1M$	$114.2M$
SOC LiveJournal	$4.8M$	$42.8M$	$285.7M$
Twitter-merged	$41M$	$1.2B$	$34.8B$

Experimental Evaluation

We consider real-world **single graph streams**, from social network, citation network.

Compare *Tonic* with state-of-the-art: algorithms provided with same memory budget k

TABLE I
DATASETS' STATISTICS: NUMBER n OF NODES; NUMBER m OF EDGES; NUMBER T OF TRIANGLES

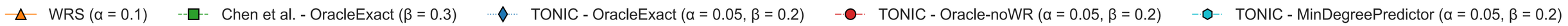
Dataset	n	m	T
Single Graphs			
Edit EN Wikibooks	$133k$	$386k$	$178k$
SOC YouTube Growth	$3.2M$	$9.3M$	$12.3M$
Cit US Patents	$3.7M$	$16.5M$	$7.5M$
Actors Collaborations	$382k$	$15M$	$346.8M$
Stackoverflow	$2.5M$	$28.1M$	$114.2M$
SOC LiveJournal	$4.8M$	$42.8M$	$285.7M$
Twitter-merged	$41M$	$1.2B$	$34.8B$

Global Relative Error:

$$|\hat{T} - T| \; / \; T$$

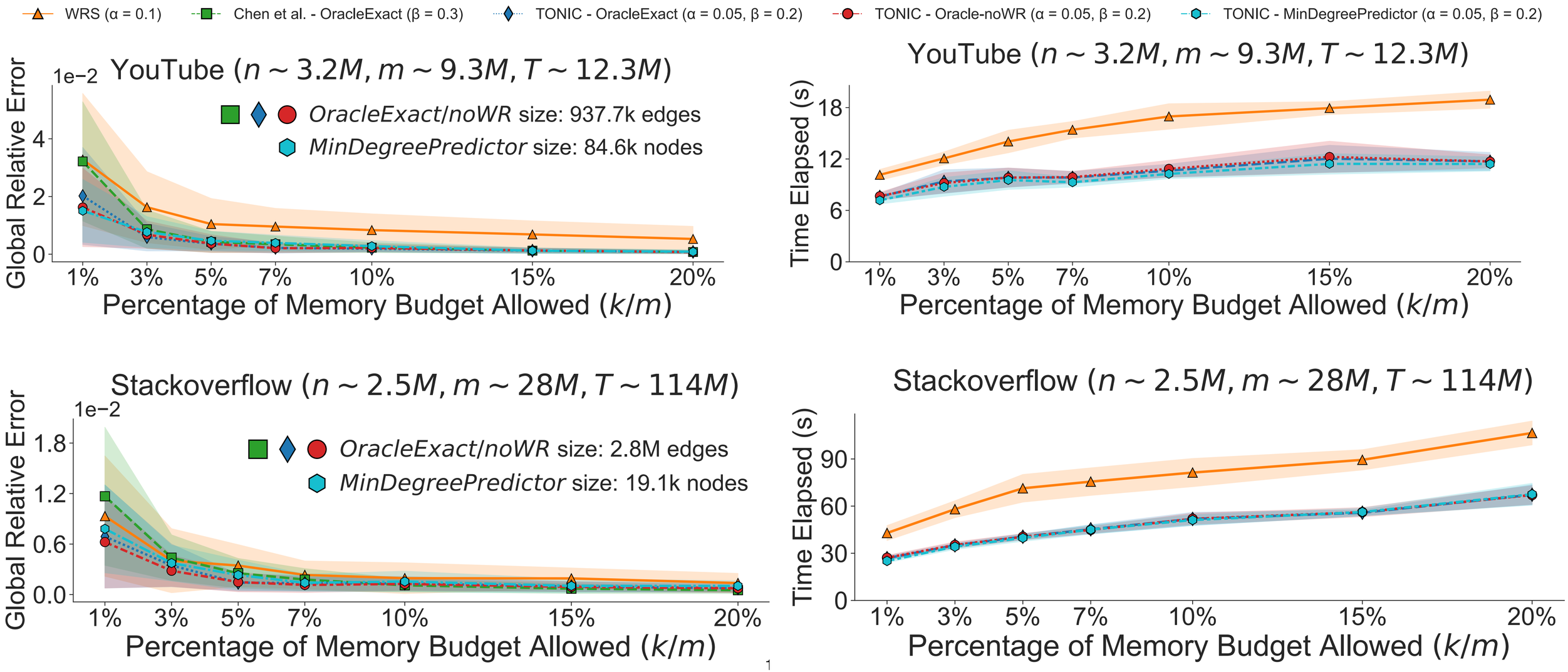
Experimental Evaluation

(1) Global Relative Error and Runtime vs Memory Budget k:



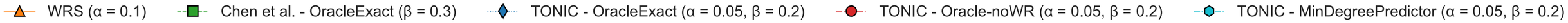
Experimental Evaluation

(1) Global Relative Error and Runtime vs Memory Budget k:



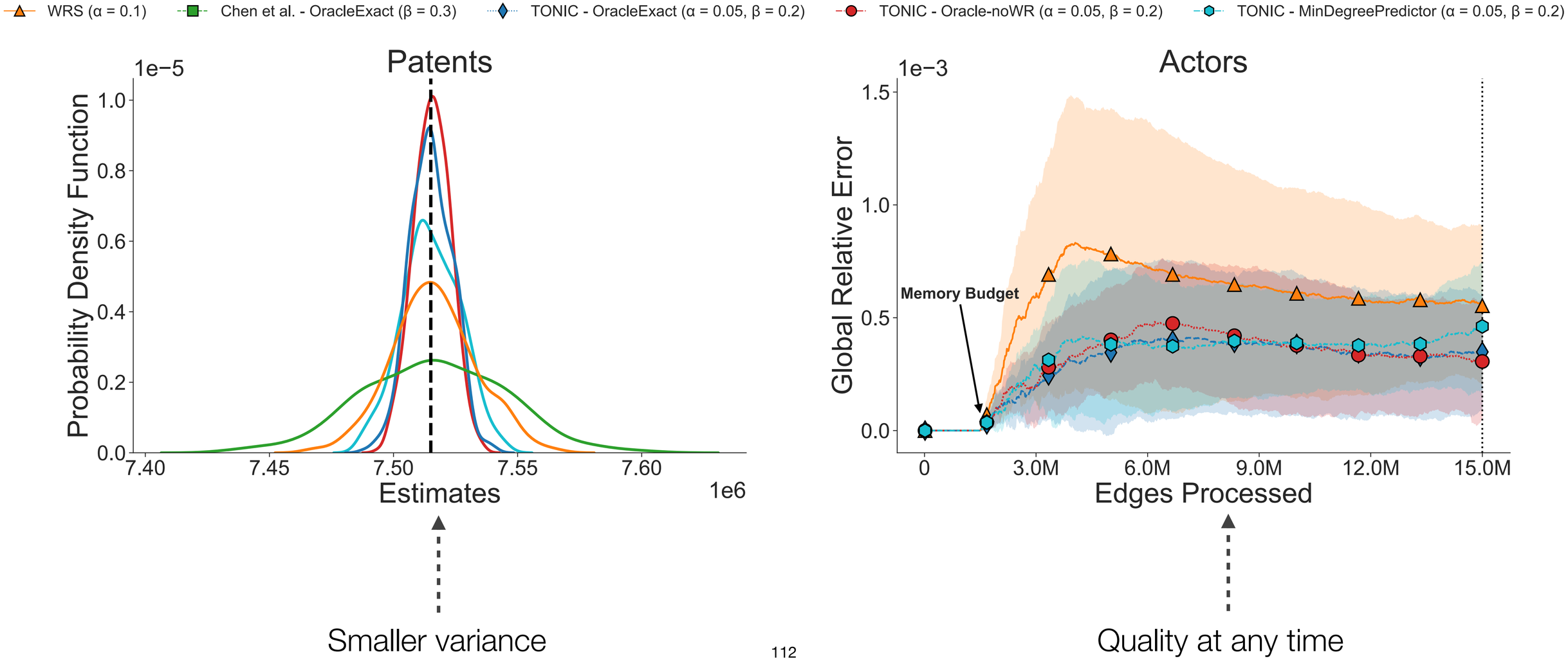
Experimental Evaluation

(2) Quality of approximations in terms of **unbiasedness** and **variance**, and **estimations** at **any time** of the stream.



Experimental Evaluation

(2) Quality of approximations in terms of **unbiasedness** and **variance**, and **estimations** at **any time** of the stream.



Smaller variance

Quality at any time

Experimental Evaluation

We consider **snapshot sequences** from autonomous system networks.

Snapshot Sequences			
Oregon (9 graphs)	$11k$	$23k$	$19.8k$
AS-CAIDA (122 graphs)	$26k$	$53k$	$36.3k$
AS-733 (733 graphs)	$6k$	$13k$	$6.5k$
Twitter (4 graphs)	$29.9M$	$373M$	$4.4B$

Experimental Evaluation

We consider **snapshot sequences** from autonomous system networks.

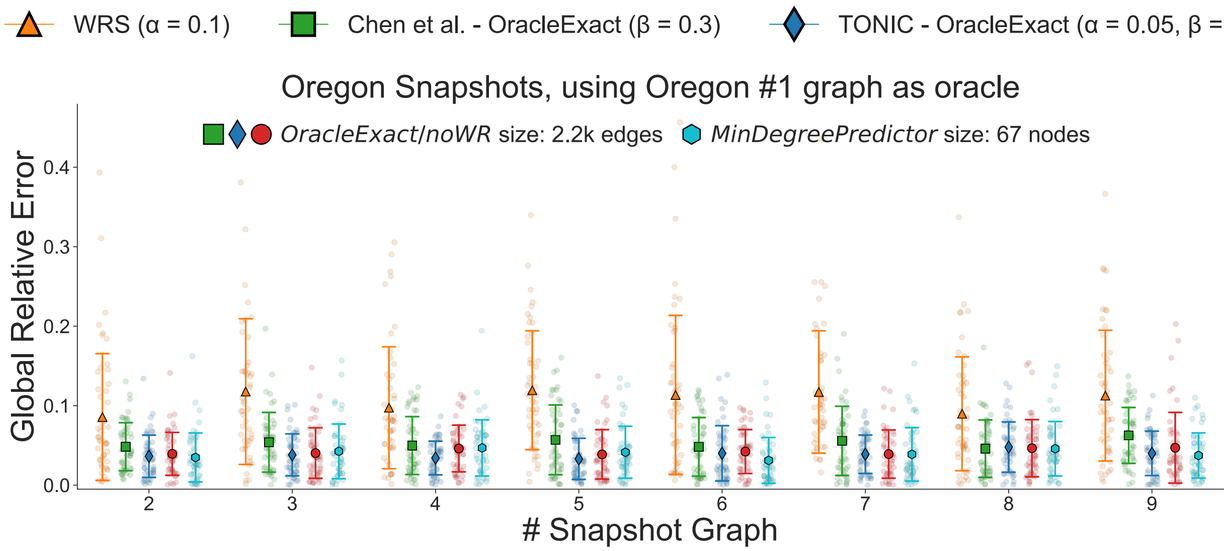
Snapshot Sequences			
Oregon (9 graphs)	$11k$	$23k$	$19.8k$
AS-CAIDA (122 graphs)	$26k$	$53k$	$36.3k$
AS-733 (733 graphs)	$6k$	$13k$	$6.5k$
Twitter (4 graphs)	$29.9M$	$373M$	$4.4B$

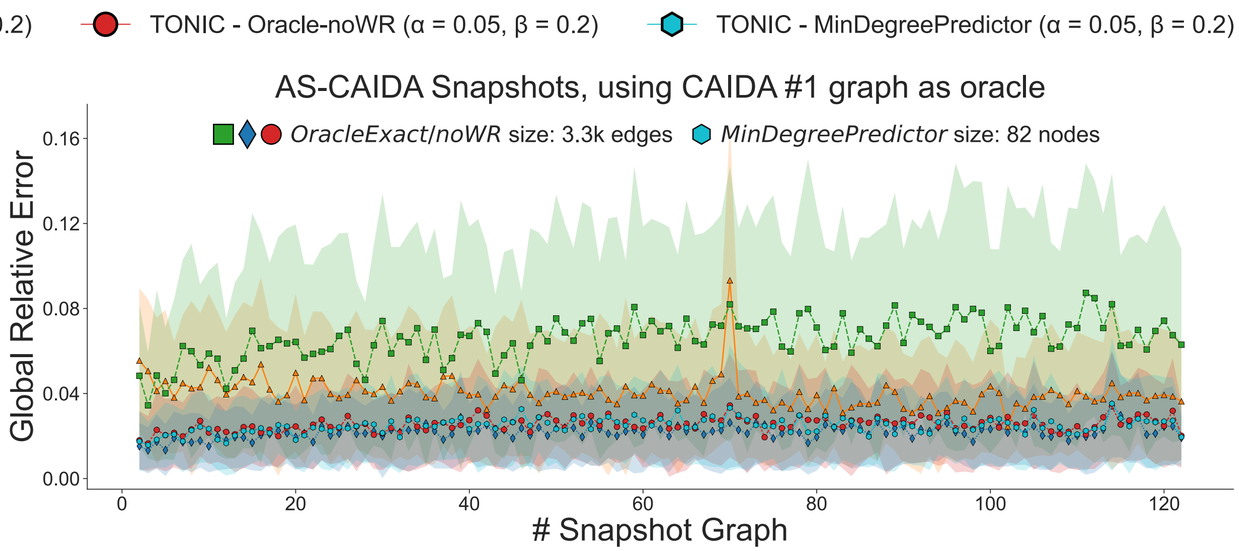
Predictors are trained only on the first graph, and then used for subsequent streams.

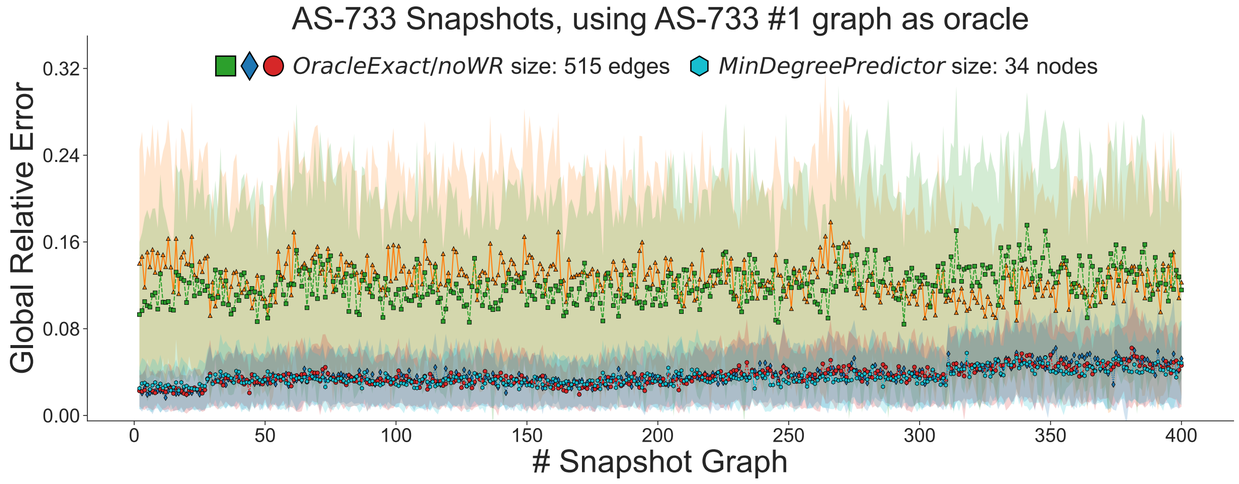
Note that *OracleExact* and *Oracle-noWR* are imperfect predictors.

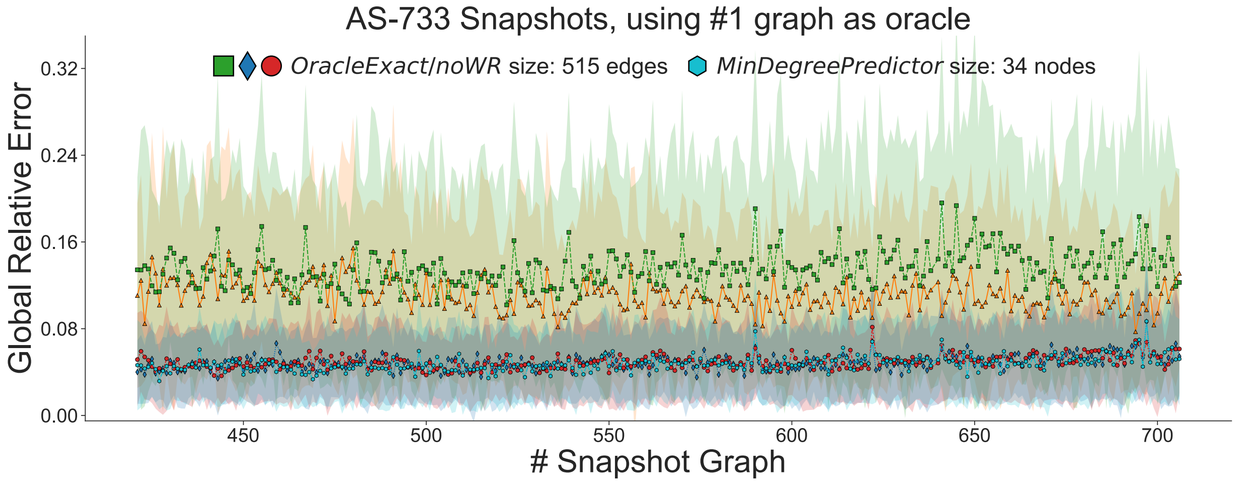
Experimental Evaluation

(3) Evaluation in **snapshot networks.**









Experimental Evaluation

(4) Performances in **fully-dynamic** streams.

Streams are created computing additions and deletions from snapshot networks.

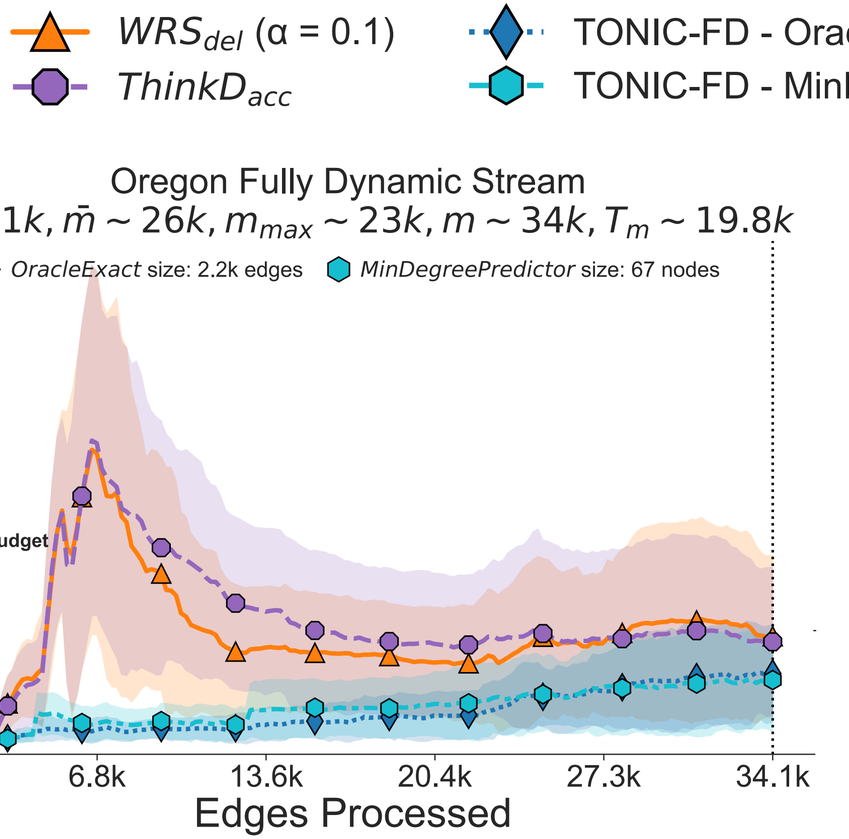
Again, predictors are trained only on the first graph, hence oblivious to removals of edges.

Experimental Evaluation

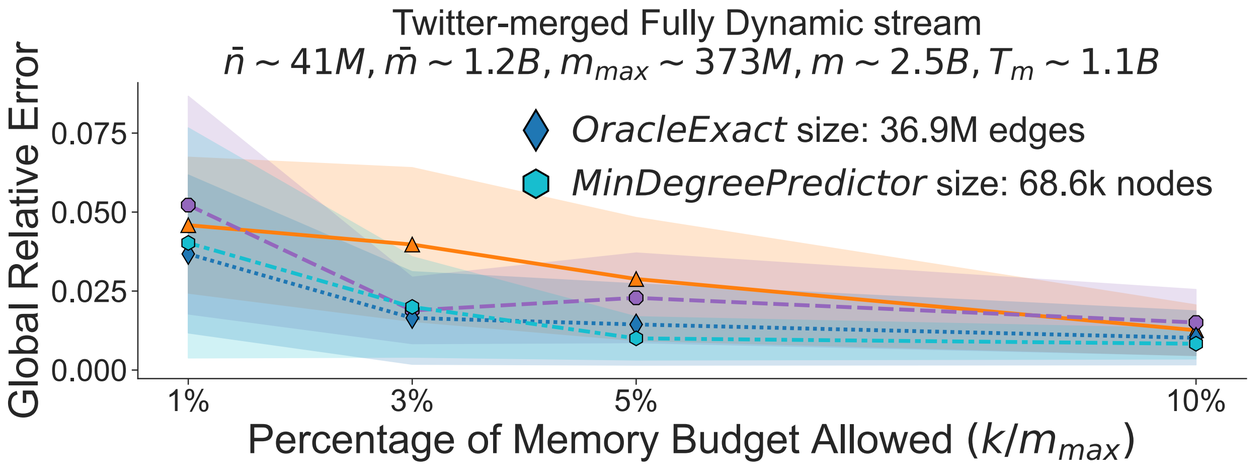
(4) Performances in **fully-dynamic** streams.

Streams are created computing additions and deletions from snapshot networks.

Again, predictors are trained <u>only</u> on the first graph, hence oblivious to removals of edges.



Quality at any time



Better estimates

Conclusion

Our contributions:

- Fast and accurate algorithm for approximating the number of global and local triangles using predictions, both for insertion-only and fully-dynamic streams;

- Proposal of very simple and application-independent predictor, based on the degree of nodes;

- Extensive experimental evaluation, showing significant improvements, especially on networks with sequences of graph streams.

Conclusion

Our contributions:

- Fast and accurate algorithm for approximating the number of global and local triangles using predictions, both for insertion-only and fully-dynamic streams;
- Proposal of very simple and application-independent predictor, based on the degree of nodes;
- Extensive experimental evaluation, showing significant improvements, especially on networks with sequences of graph streams.

Thanks:

Progetto “National Centre for HPC, Big Data and Quantum Computing”, CN00000013 (approvato nell’ambito del Bando M42C – Investimento 1.4 – Avvisto “Centri Nazionali” – D.D. n. 3138 del 16.12.2021, ammesso a finanziamento con Decreto del MUR n. 1031 del 17.06.2022)

DIPARTIMENTO DI INGEGNERIA DELL’INFORMAZIONE

C. Boldrin and F. Vandin, **“Fast and Accurate Triangle Counting in Graph Streams Using Predictions”**, to appear at ICDM 2024.

Code and paper:

arXiv https://arxiv.org/pdf/2409.15205

GitHub https://github.com/VandinLab/Tonic

cristian.boldrin.2@phd.unipd.it

Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.

Each triangle counted or deleted in the sample is scaled by the inverse of the probability with which the triangle edges but $e^{(t)}$ have been previously sampled.

Probability p Computation:

- If **no** edges are light: $p = 1$
- If only **one** edge is light: $p = p^{(t)}$
- If **both** edges are light: $p = p^{(t)} \cdot p^{(t-1)}$

Reservoir Sampling:

$$p^{(t)} = \min\left(1, \frac{k(1-\alpha)(1-\beta)}{\left|\mathscr{L}^{(t)}\right|}\right)$$

Tonic: Overall Algorithm

For each edge $e^{(t)}$ observed on the stream Σ at time t.

Each triangle counted or deleted in the sample is scaled by the inverse of the probability with which the triangle edges but $e^{(t)}$ have been previously sampled.

Probability p Computation:

- If **no** edges are light: $p = 1$
- If only **one** edge is light: $p = p^{(t)}$
- If **both** edges are light: $p = p^{(t)} \cdot p^{(t-1)}$

Random Pairing:

$$p^{(t)} = \min\left(1, \frac{k(1-\alpha)(1-\beta)}{\left|\mathscr{L}^{(t)}\right| + d_g + d_b}\right)$$

Tonic: theoretical analysis

We prove that the predictor helps when it provides fairly reliable information on heavy edges.

Tonic: theoretical analysis

We prove that the predictor helps when it provides fairly reliable information on heavy edges.

If this is not the case, we also prove that our algorithm *Tonic* returns estimates as accurate as *WRS*.

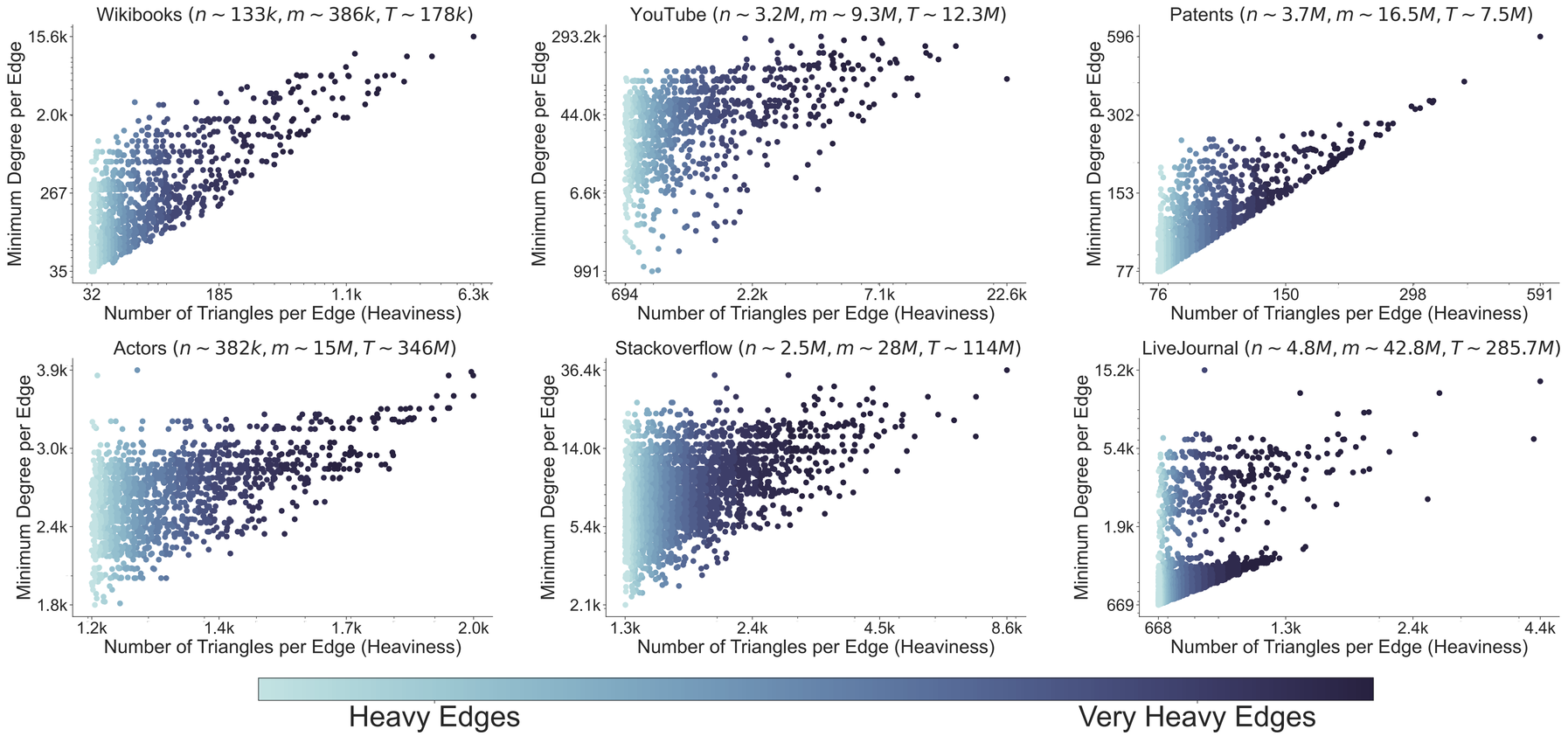
Proposition (informal): the variance of the estimates of *Tonic* is equal than the variance of the estimates of *WRS* if the predictor predicts a randomly chosen set of edges as heavy edges.

Edge Heaviness Predictor

The predictor used by *Tonic* could be implemented by a machine learning model that may consider information other than the graph.

In our experiments we consider:

- *OracleExact*
- *Oracle-noWR*
- *MinDegreePredictor*



Experimental Evaluation

(2) Quality of approximations in terms of unbiasedness and variance, estimations at any time of the stream, and **number** of **counted** and **estimated** triangles.