VLDB 2025 – Research track: Data Stream Mining

Approximate 2-hop neighborhoods on incremental graphs: An efficient lazy approach

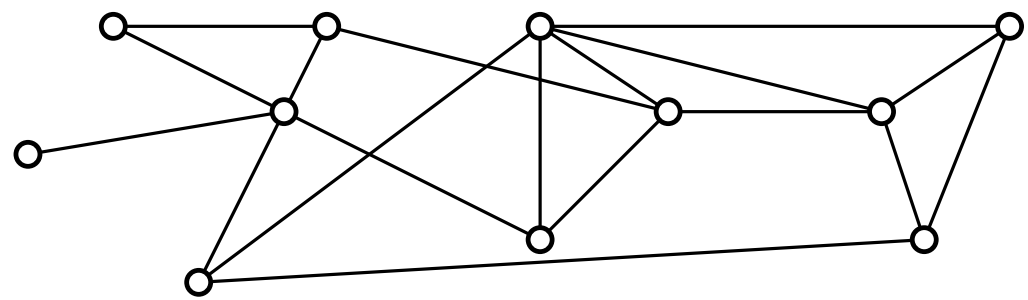
Luca Becchetti[+], Andrea Clementi[*], Luciano Gualà[*],
Luca Pepè Sciarria[*], Alessandro Straziota[*], Matteo Stromieri[*]

(+) *Sapienza*, Univeristy of Rome (*) *Tor Vergata*, Univeristy of Rome

General Framework

Layer 0

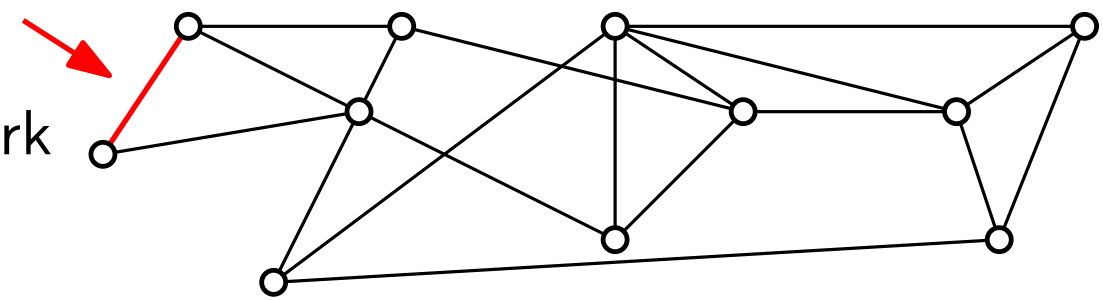
Underlying *dynamic* network



General Framework

Layer 0

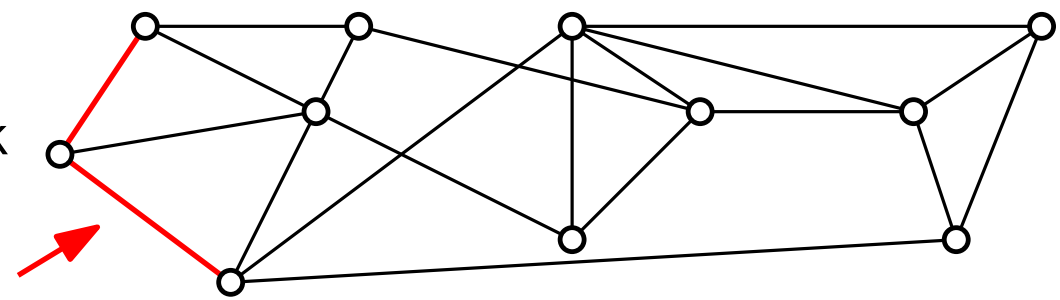
Underlying *dynamic* network



General Framework

Layer 0

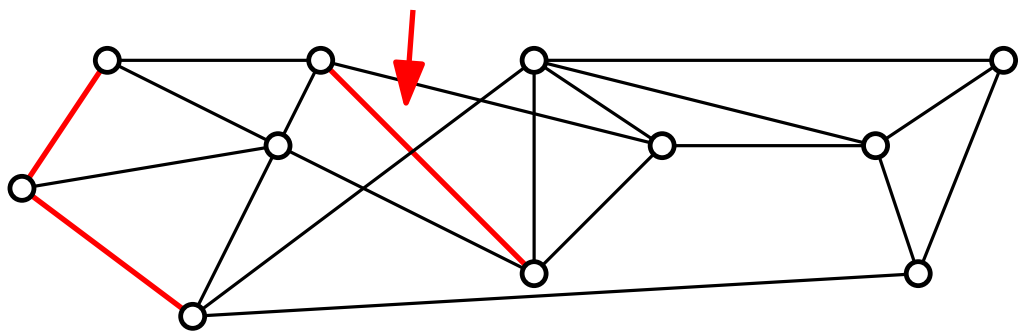
Underlying *dynamic* network



General Framework

Layer 0

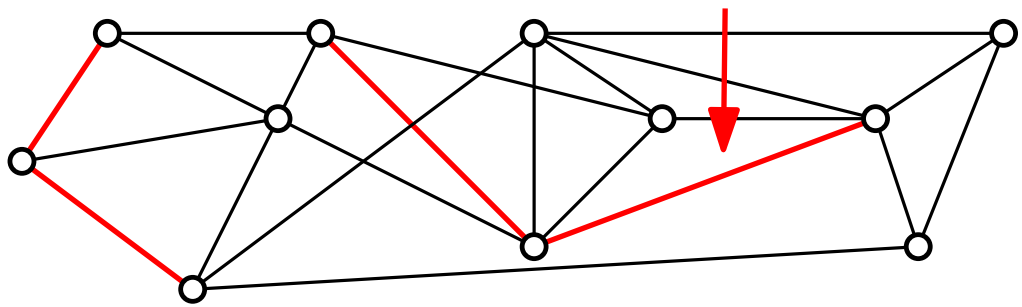
Underlying *dynamic* network



General Framework

Layer 0

Underlying *dynamic* network



General Framework

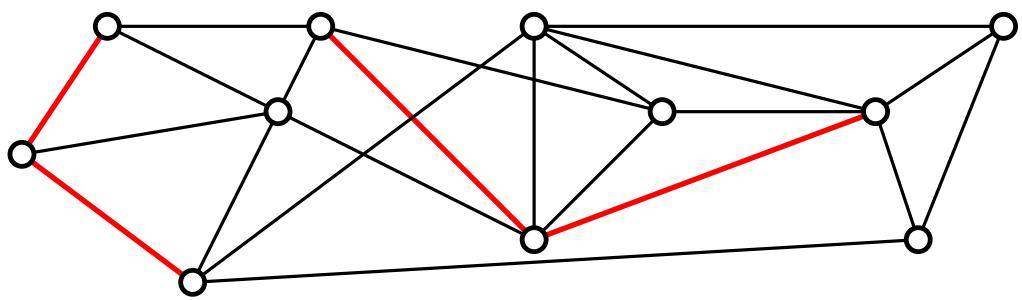
Layer 1

Answer *basic queries* on d-hop neighborhoods (or d-radius balls)

$$B_d(u) = \{v \mid d(u, v) \leq d\}$$

Layer 0

Underlying *dynamic* network



General Framework

Layer 1

Answer *basic queries* on d-hop neighborhoods (or d-radius balls)

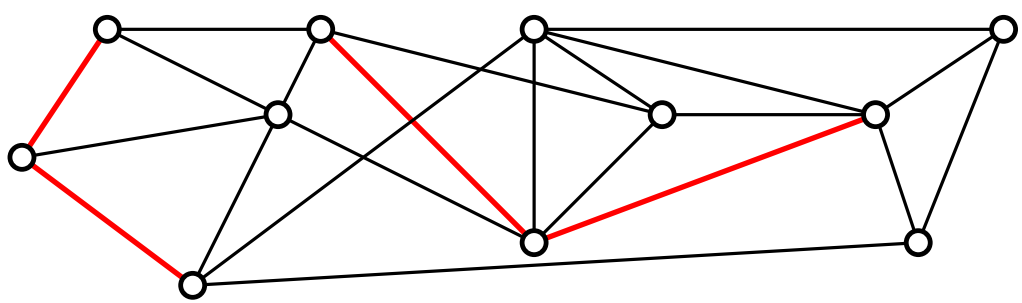
$$B_d(u) = \{v \mid d(u, v) \leq d\}$$

Basic queries:

- *size estimation* of $B_d(u)$

Layer 0

Underlying *dynamic* network



General Framework

Layer 1

Answer *basic queries* on d-hop neighborhoods (or d-radius balls)

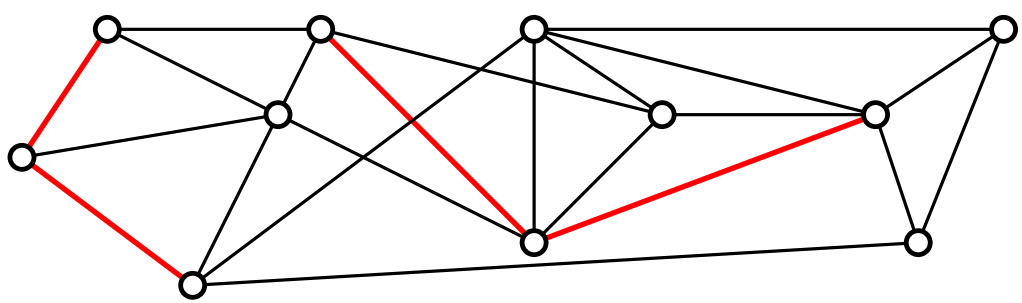
$$B_d(u) = \{v \mid d(u, v) \leq d\}$$

Basic queries:

- *size estimation* of $B_d(u)$
- *jaccard similarity* between $B_d(u), B_d(v)$

Layer 0

Underlying *dynamic* network



General Framework

Layer 1

Answer *basic queries* on d-hop neighborhoods (or d-radius balls)

$$B_d(u) = \{v \mid d(u,v) \leq d\}$$

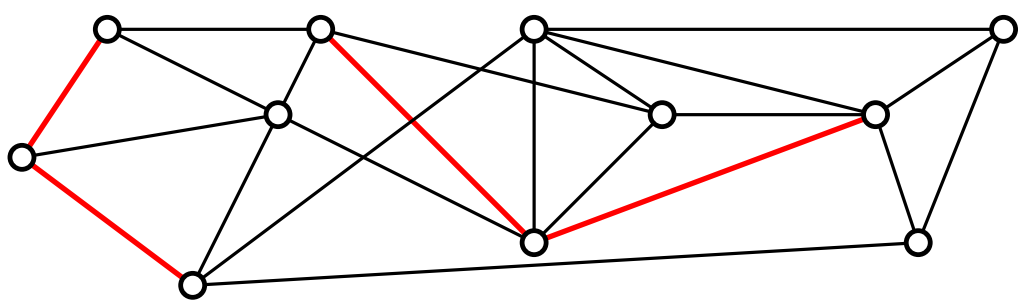
Basic queries:

- *size estimation* of $B_d(u)$
- *jaccard similarity* between $B_d(u), B_d(v)$

$$J(B_d(u), B_d(v)) = \frac{|B_d(u) \cap B_d(v)|}{|B_d(u) \cup B_d(v)|}$$

Layer 0

Underlying *dynamic* network



General Framework

Layer 2 Downstream tasks

Layer 1

Answer *basic queries* on d-hop neighborhoods (or d-radius balls)

$$B_d(u) = \{v \mid d(u,v) \leq d\}$$

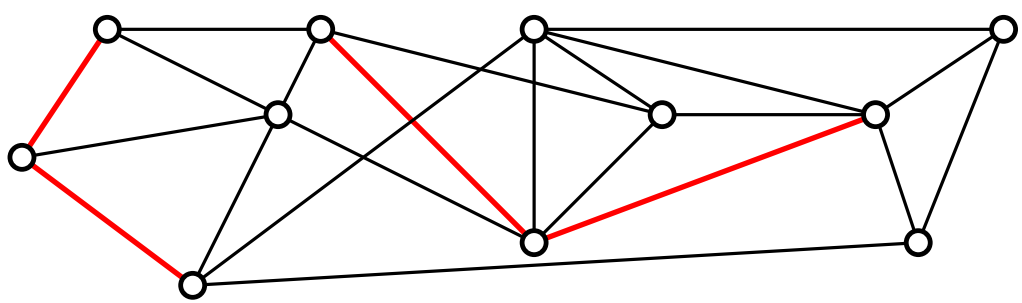
Basic queries:

- *size estimation* of $B_d(u)$
- *jaccard similarity* between $B_d(u), B_d(v)$

$$J(B_d(u), B_d(v)) = \frac{|B_d(u) \cap B_d(v)|}{|B_d(u) \cup B_d(v)|}$$

Layer 0

Underlying *dynamic* network



Example of downstream tasks for 2-hop neighborhoods

Approximate top-k central nodes in a (dynamic) network, according to some distance-based indices, e.g.,

Harmonic Centrality

$$HC(v) = \sum_{u \neq v} \frac{1}{d(u,v)}$$

Closeness Centrality

$$CC(v) = \frac{1}{\sum_u d(u,v)}$$

Efficiency Issues

To answer basic queries (even on *static graphs*) could be inefficient.

Efficiency Issues

To answer basic queries (even on *static graphs*) could be inefficient.

Example: computing $J(B_d(u), B_d(v))$ requires $\Omega(|B_d(u)| + |B_d(v)|)$ time.

Efficiency Issues

To answer basic queries (even on *static graphs*) could be inefficient.

Example: computing $J(B_d(u), B_d(v))$ requires $\Omega(|B_d(u)| + |B_d(v)|)$ time.

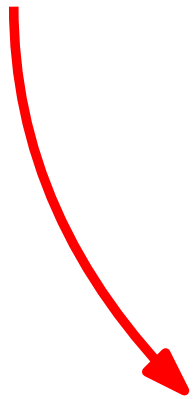
not feasible on large networks

Efficiency Issues

To answer basic queries (even on *static graphs*) could be inefficient.

Example: computing $J(B_d(u), B_d(v))$ requires $\Omega(|B_d(u)| + |B_d(v)|)$ time.

not feasible on large networks



problems overcome with the use of **data sketches**

Data Sketches

Data Sketches

A *data sketch* $\mathcal{S}(A)$ for a (large) set A, is a probabilistic representation, s.t.

Data Sketches

A *data sketch* $\mathcal{S}(A)$ for a (large) set A, is a probabilistic representation, s.t.

i) has **compact size**, usually almost constant

Data Sketches

A *data sketch* $\mathcal{S}(A)$ for a (large) set A, is a probabilistic representation, s.t.

i) has **compact size**, usually almost constant

ii) **efficiently approximate** summaries of the sets

Data Sketches

A *data sketch* $\mathcal{S}(A)$ for a (large) set A, is a probabilistic representation, s.t.

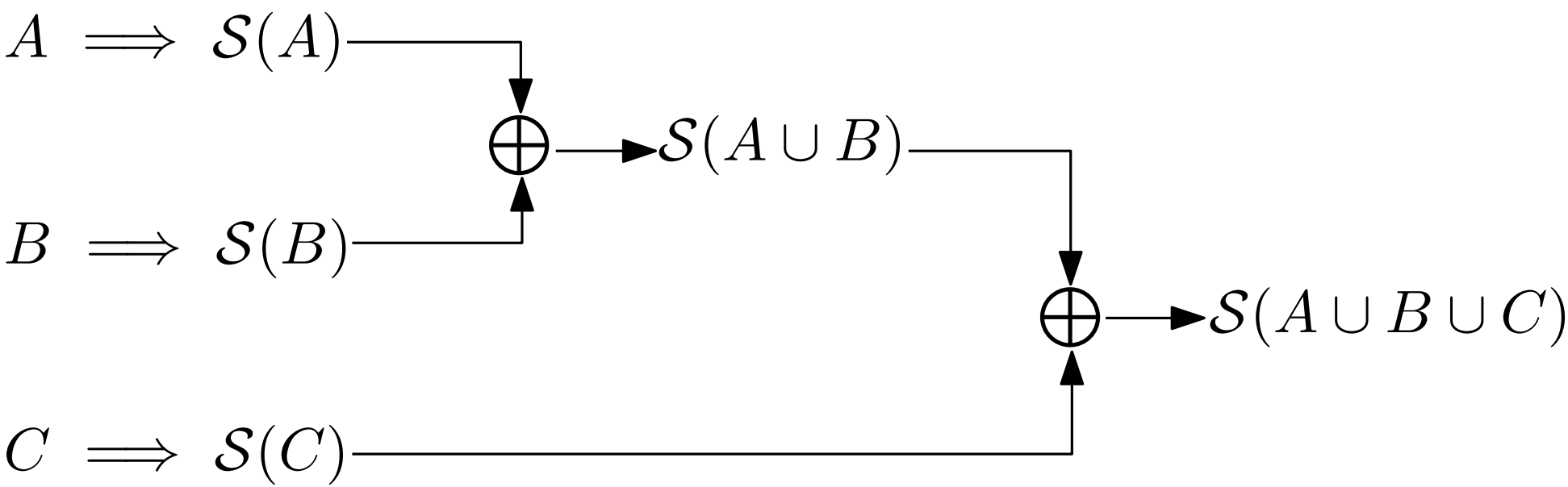
i) has **compact size**, usually almost constant

ii) **efficiently approximate** summaries of the sets

iii) (optional) can be **composable**, i.e., given only $\mathcal{S}(A), \mathcal{S}(B)$ we can compute $\mathcal{S}(A \cup B)$

Data Sketches

A *data sketch* $\mathcal{S}(A)$ for a (large) set A, is a probabilistic representation, s.t.



Example of Data Sketches

Basic Query	Composable Sketches

Example of Data Sketches

Basic Query	Composable Sketches
Size estimation	**Probabilistic Counters**: Morris' counter, LogLog counter, KMV counter

Example of Data Sketches

Basic Query	Composable Sketches
Size estimation	**Probabilistic Counters**: Morris' counter, LogLog counter, KMV counter
Jaccard Similarity	MinHash, Bottom-k, one-permutation hash

Our Contribution

Our Contribution

We propose a **framework** that keeps data sketches updated on the 2-hop neighborhoods of incremental graphs, resulting from sequences of edge insertions.

Our Contribution

We propose a **framework** that keeps data sketches updated on the 2-hop neighborhoods of incremental graphs, resulting from sequences of edge insertions.

i) we provide a strong theoretical analysis on different stream models.

Our Contribution

We propose a **framework** that keeps data sketches updated on the 2-hop neighborhoods of incremental graphs, resulting from sequences of edge insertions.

i) we provide a strong theoretical analysis on different stream models.

ii) we provide an efficient implementation and extensive experimental evaluation, which validates theoretical analysis.

Our Contribution

We propose a **framework** that keeps data sketches updated on the 2-hop neighborhoods of incremental graphs, resulting from sequences of edge insertions.

i) we provide a strong theoretical analysis on different stream models.

ii) we provide an efficient implementation and extensive experimental evaluation, which validates theoretical analysis.

The framework is **general**: any type of sketch can be plugged in, as long as it is composable.

Our Contribution

We propose a **framework** that keeps data sketches updated on the 2-hop neighborhoods of incremental graphs, resulting from sequences of edge insertions.

i) we provide a strong theoretical analysis on different stream models.

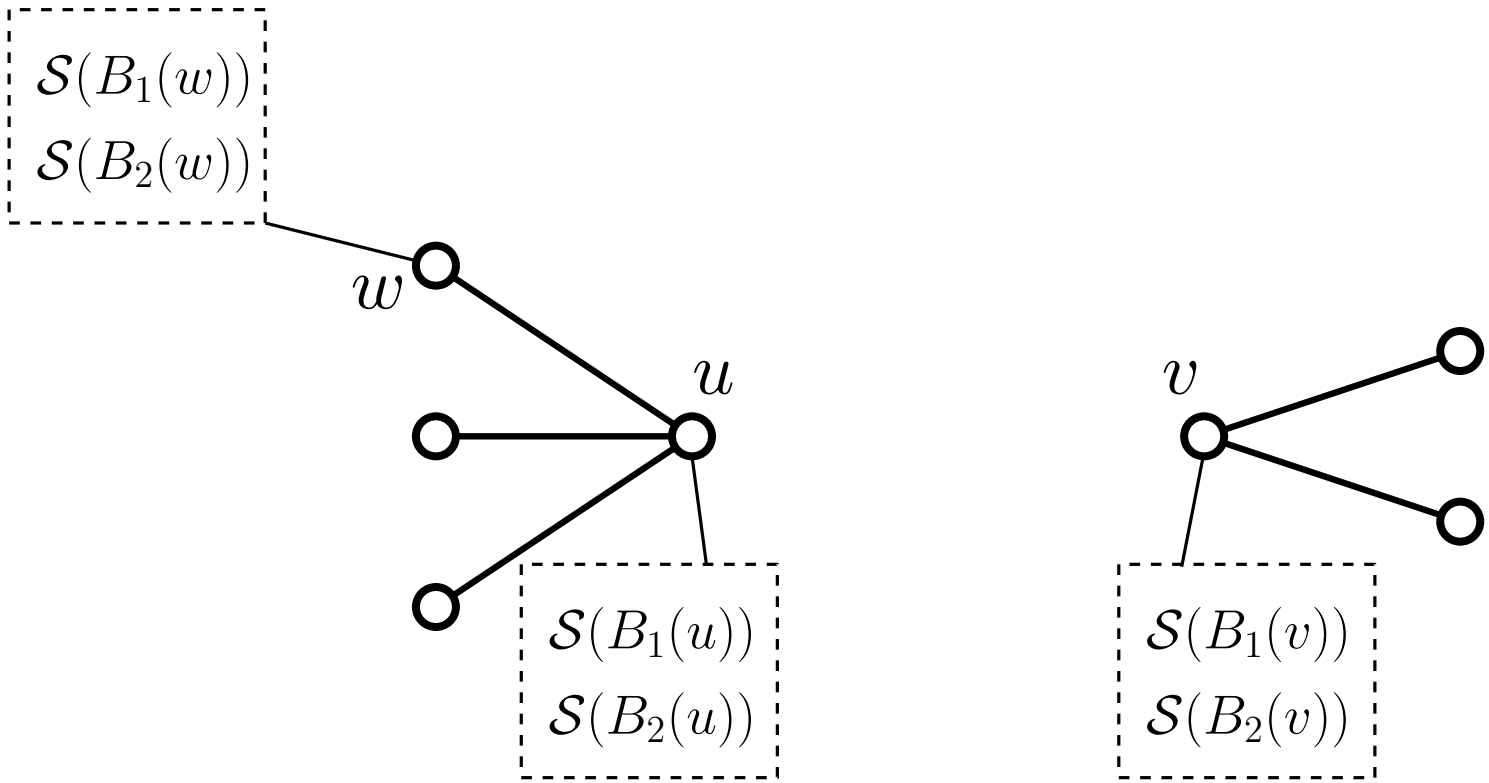
ii) we provide an efficient implementation and extensive experimental evaluation, which validates theoretical analysis.

The framework is **general**: any type of sketch can be plugged in, as long as it is composable.

$\Longrightarrow$ independent of the basic query, and therefore of the downstream task.

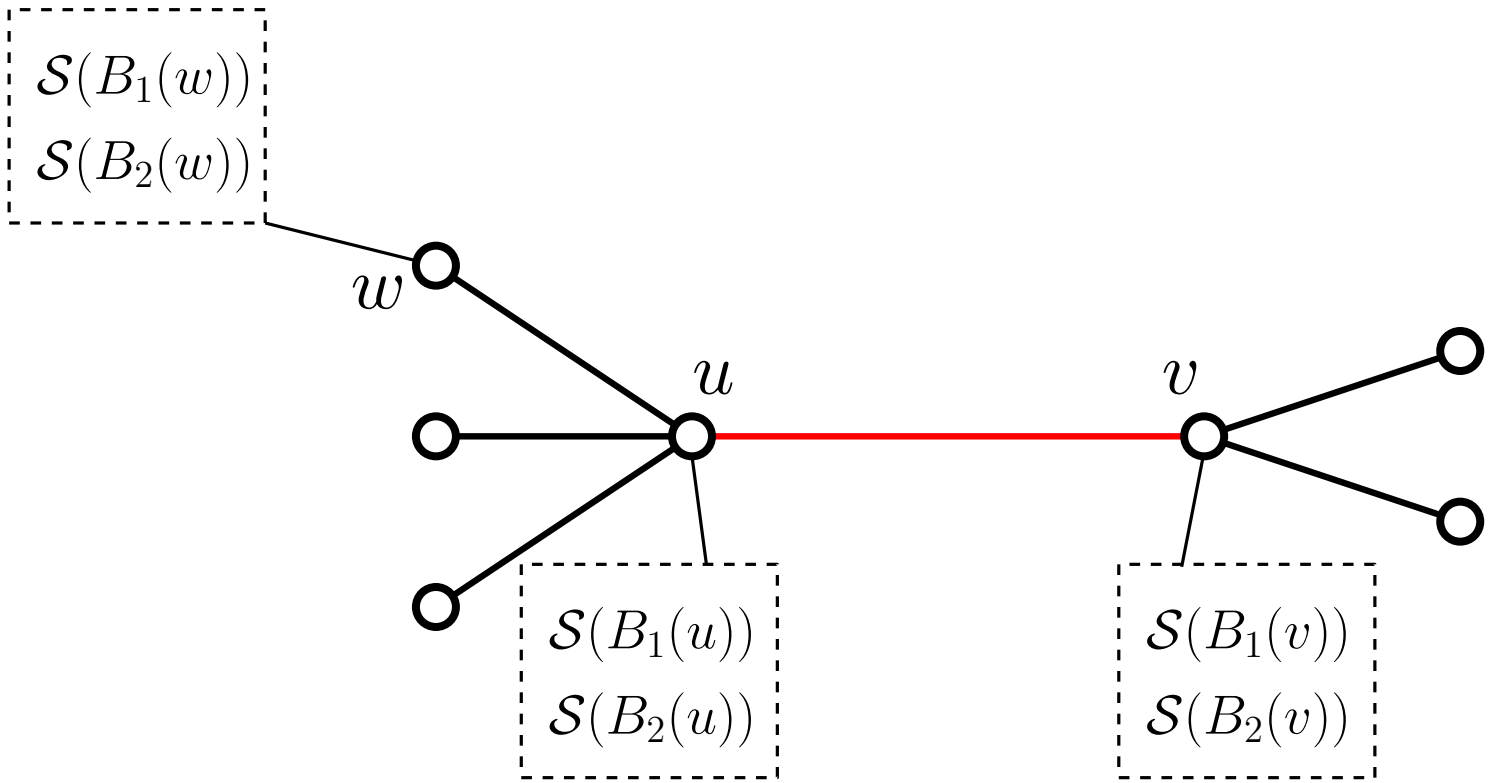
The Idea: Lazy Updates

We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.



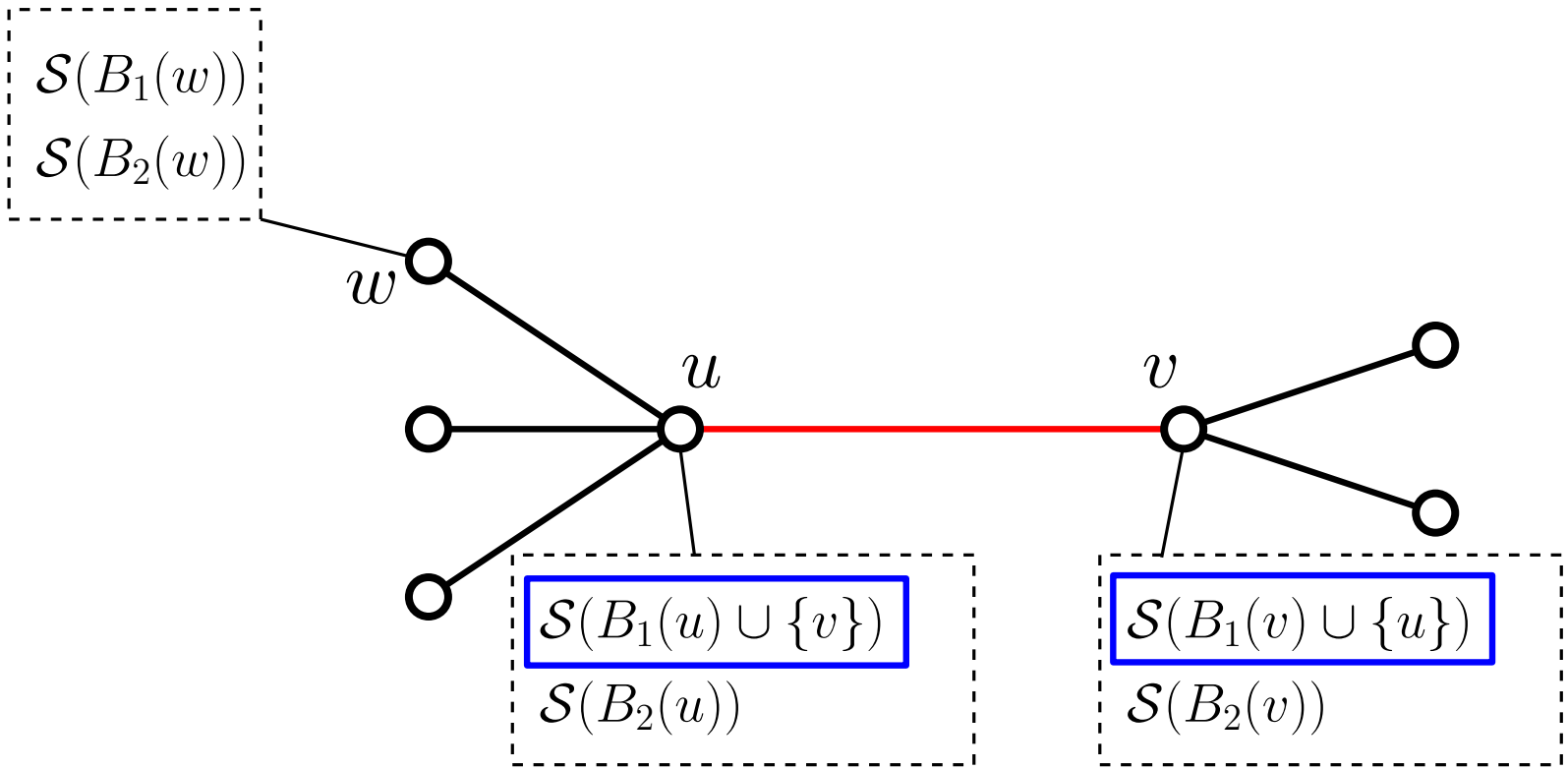
The Idea: Lazy Updates

We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.



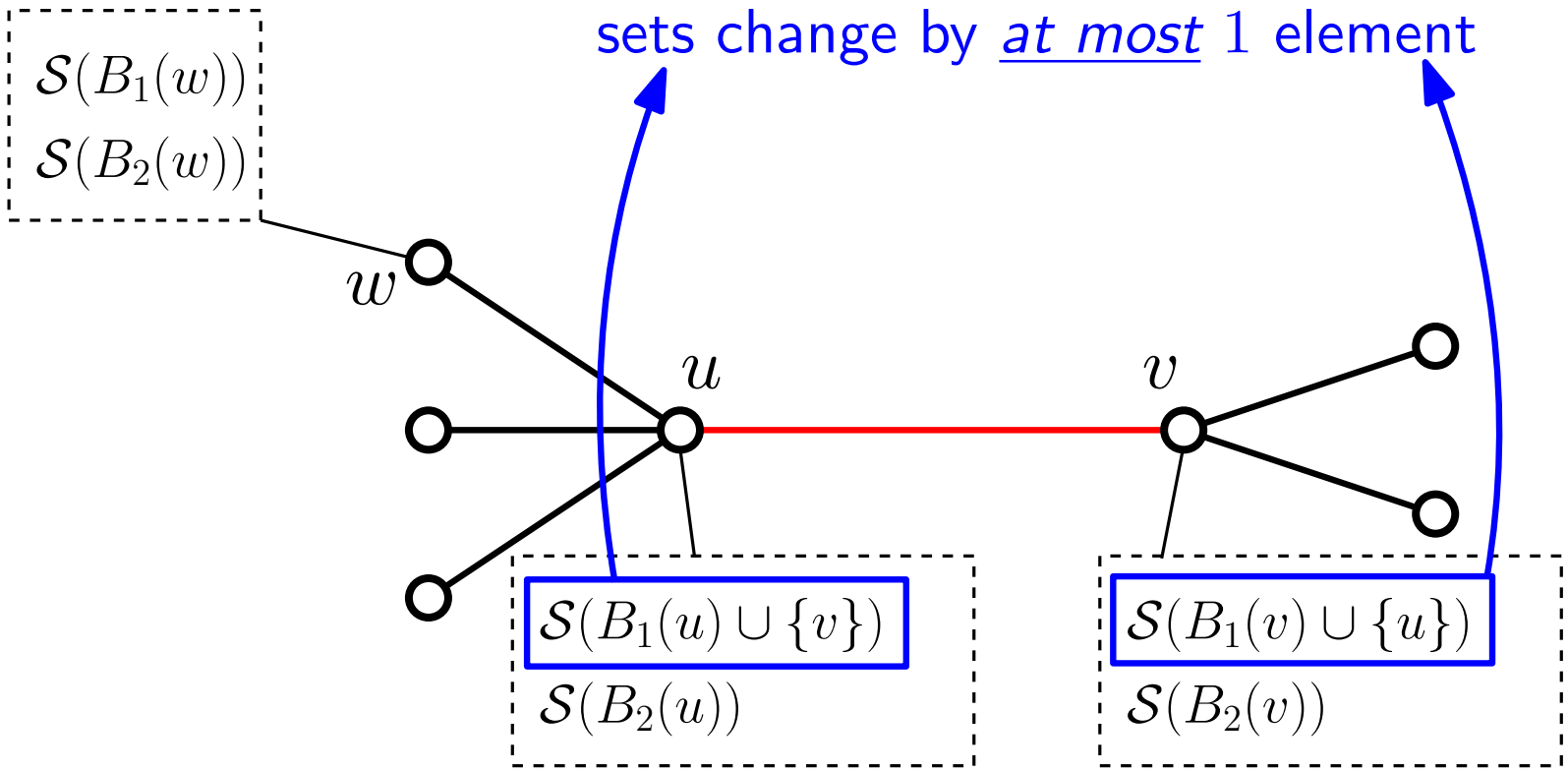
The Idea: Lazy Updates

We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.



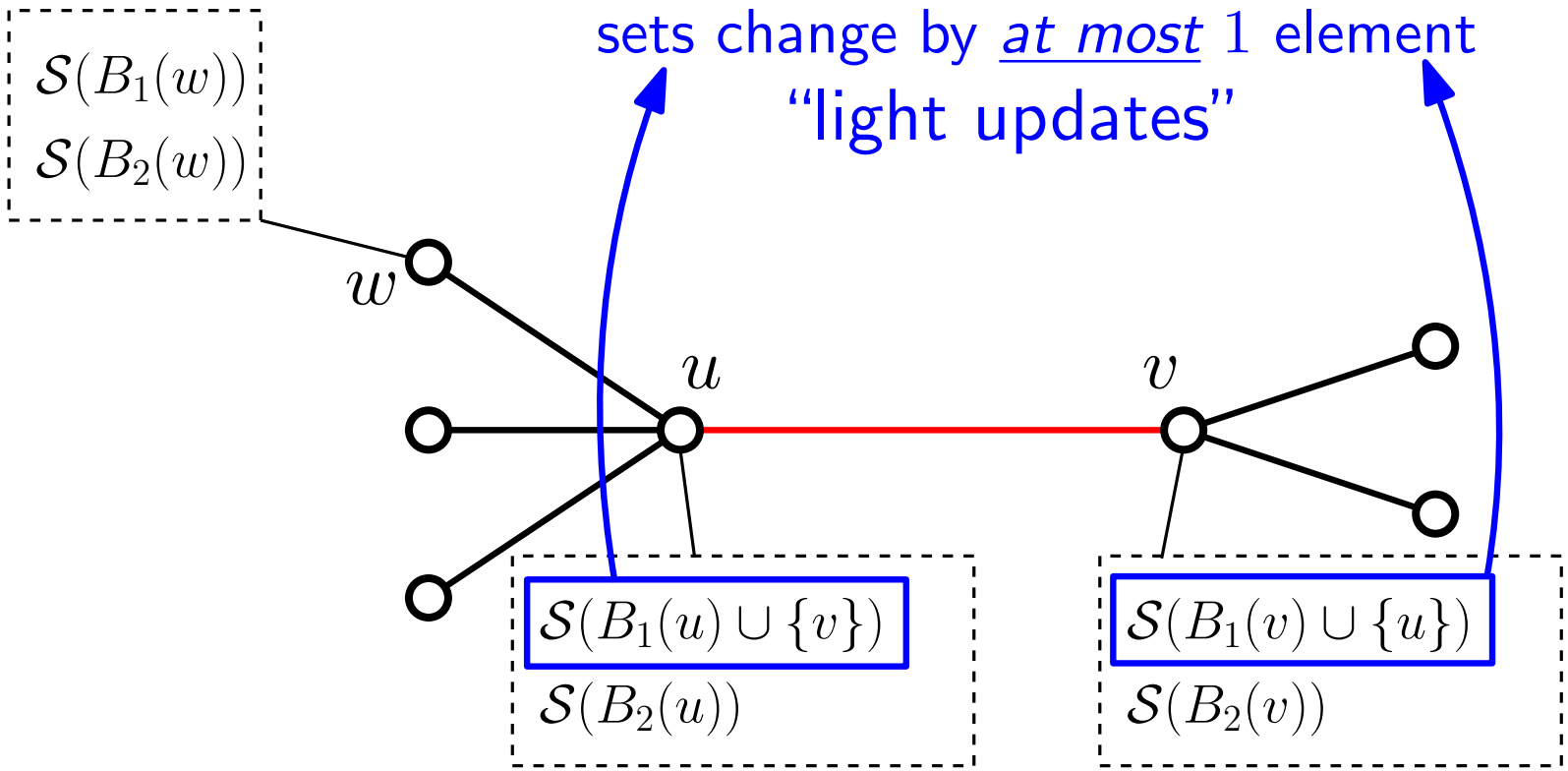
The Idea: Lazy Updates

We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.



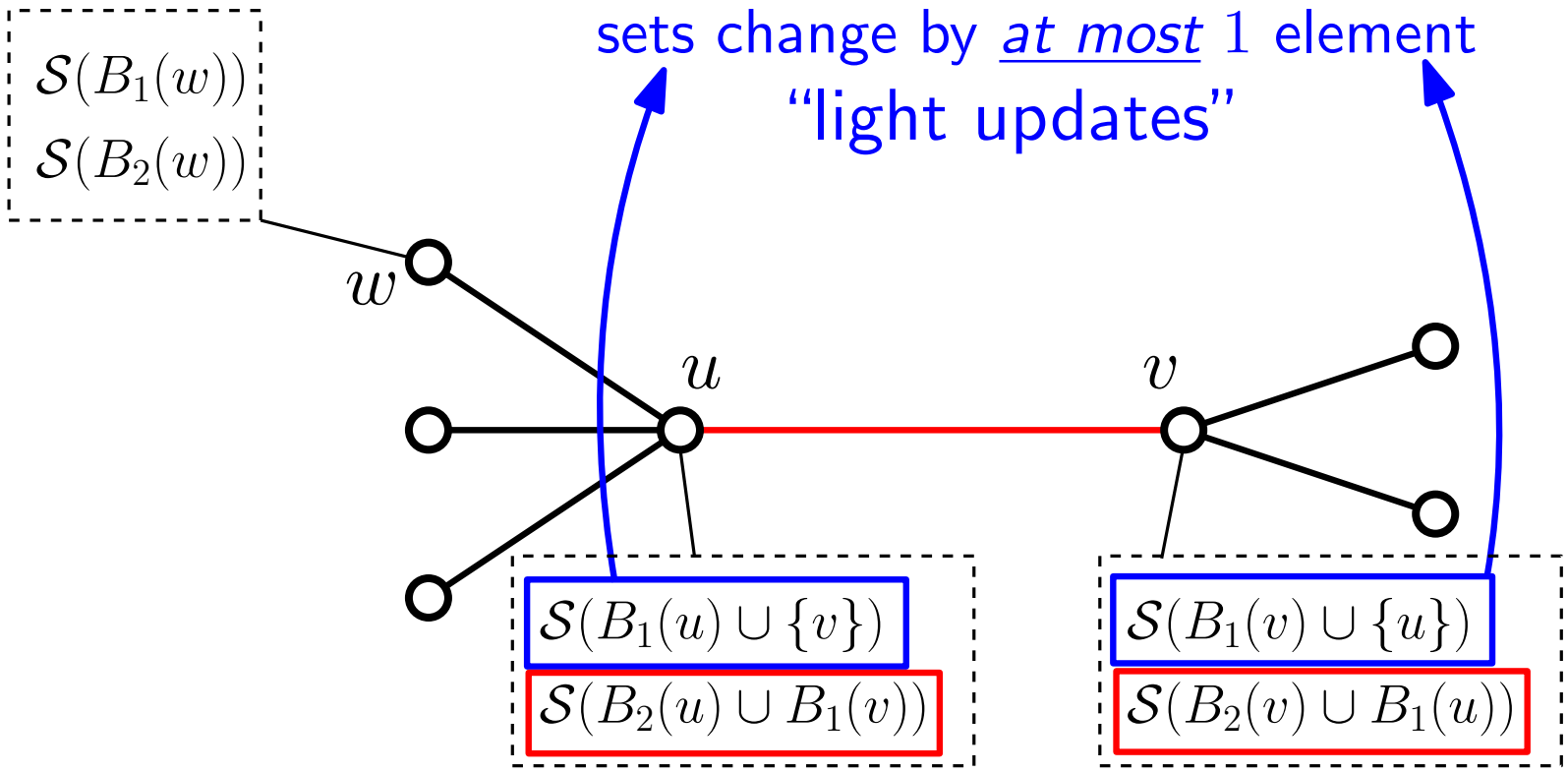
The Idea: Lazy Updates

We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.



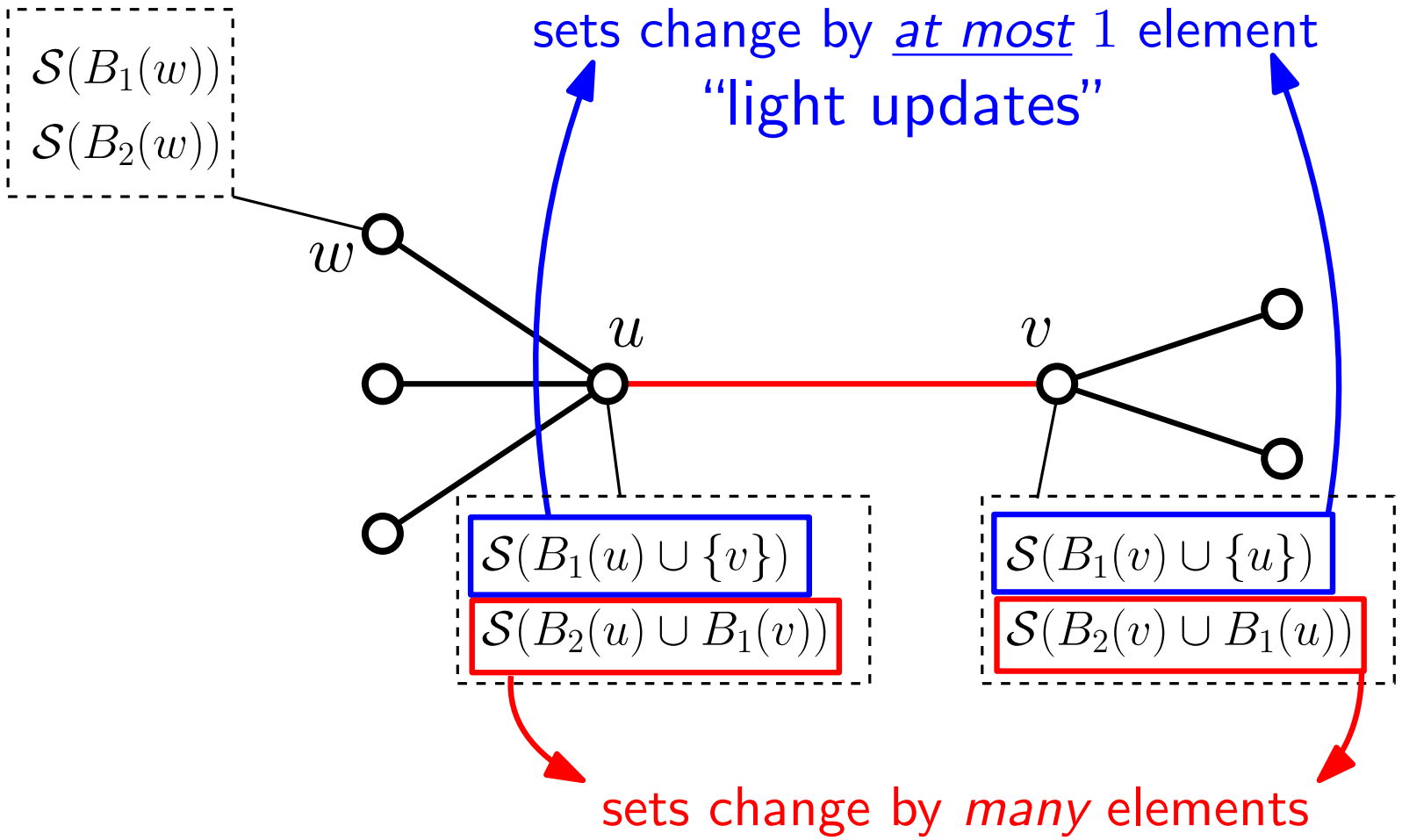
The Idea: Lazy Updates

We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.



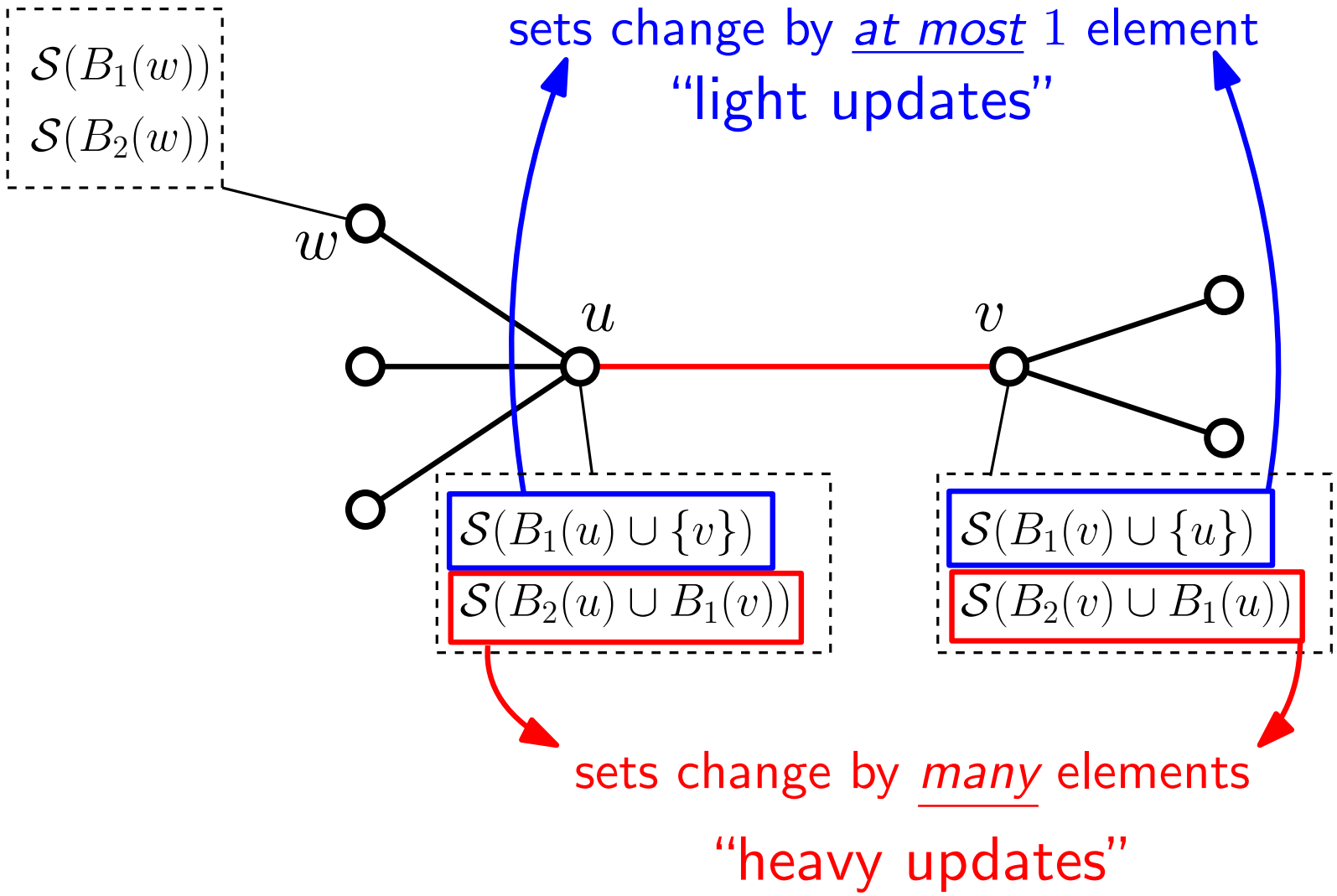
The Idea: Lazy Updates

We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.



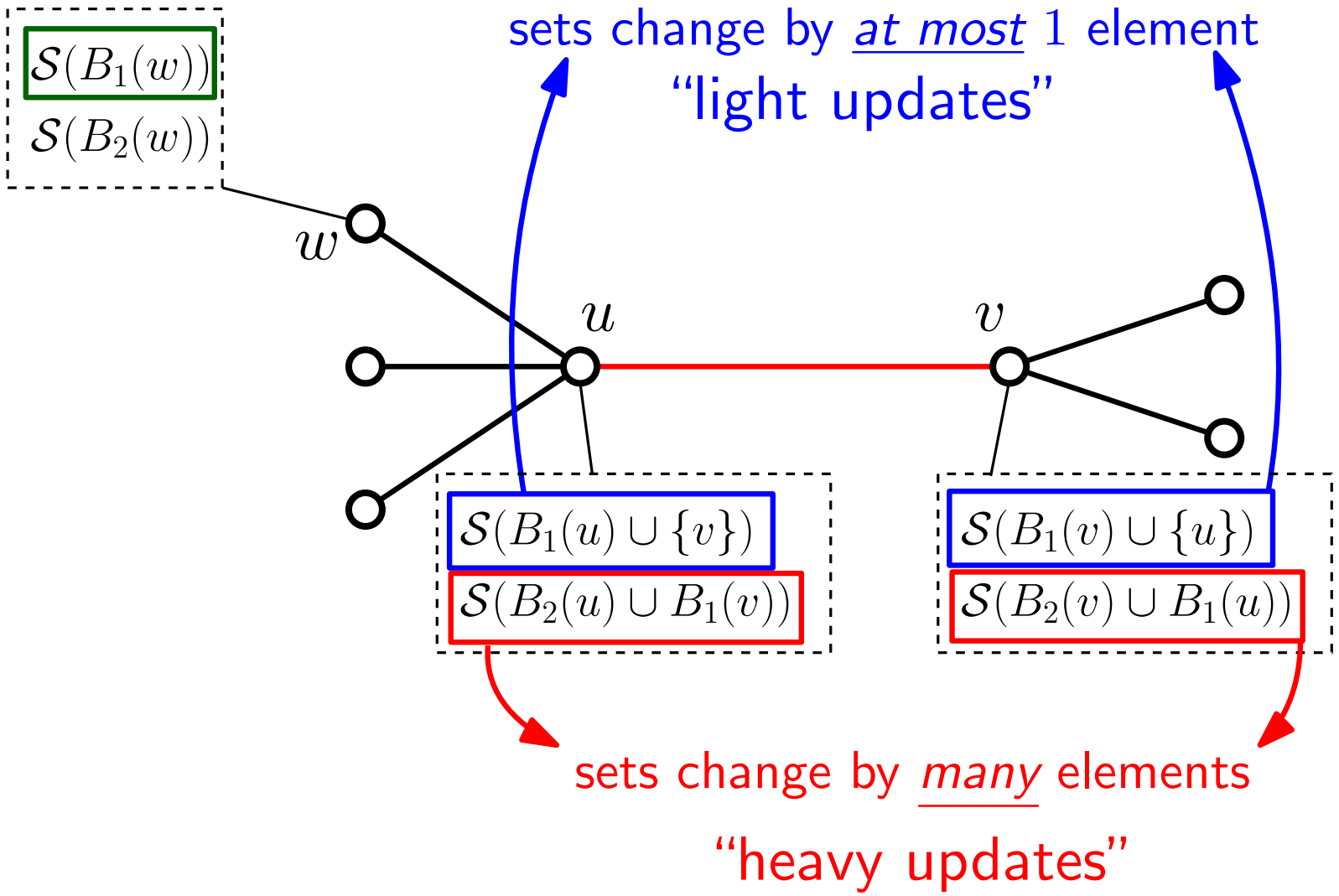
The Idea: Lazy Updates

We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.



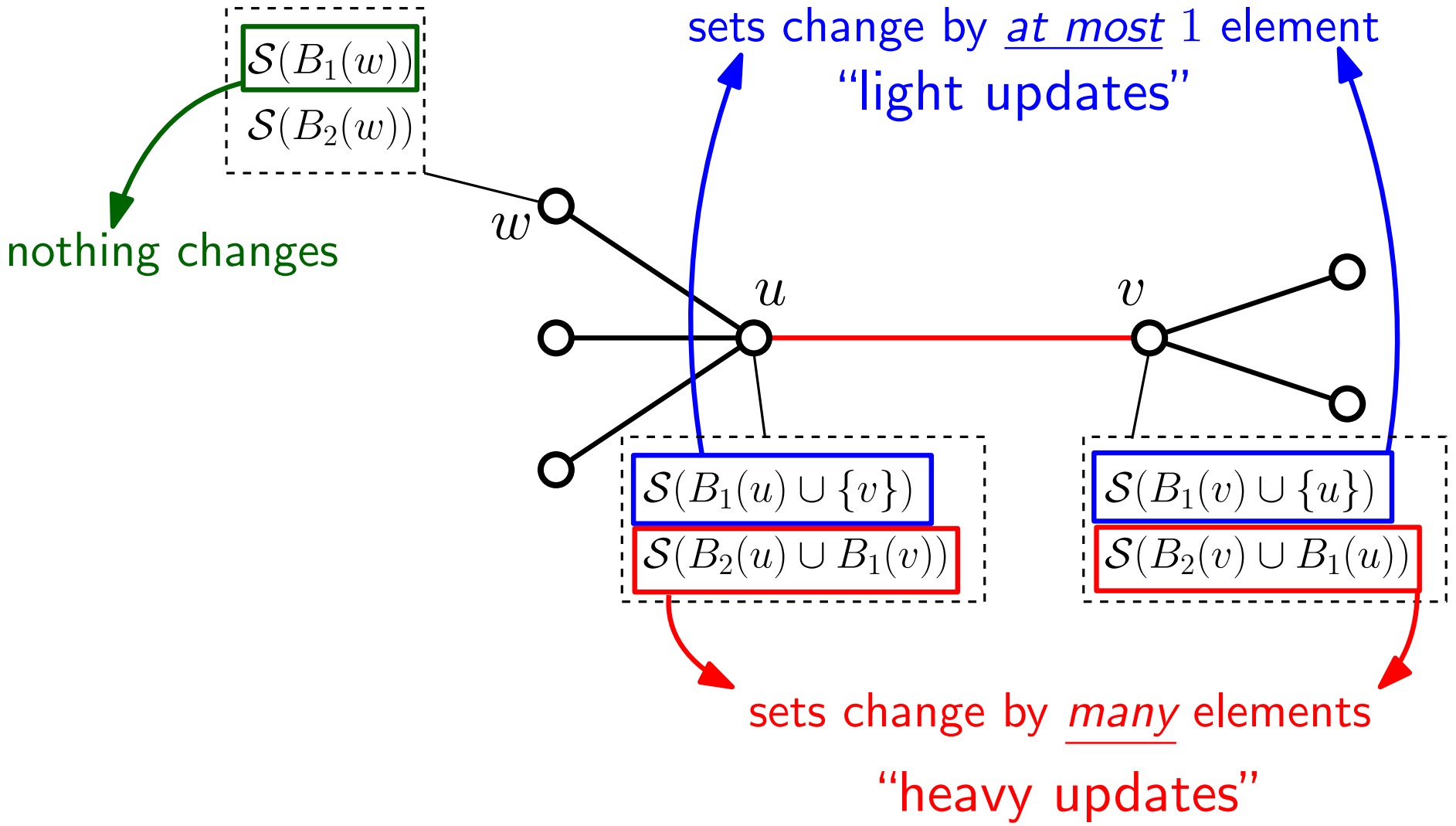
The Idea: Lazy Updates

We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.



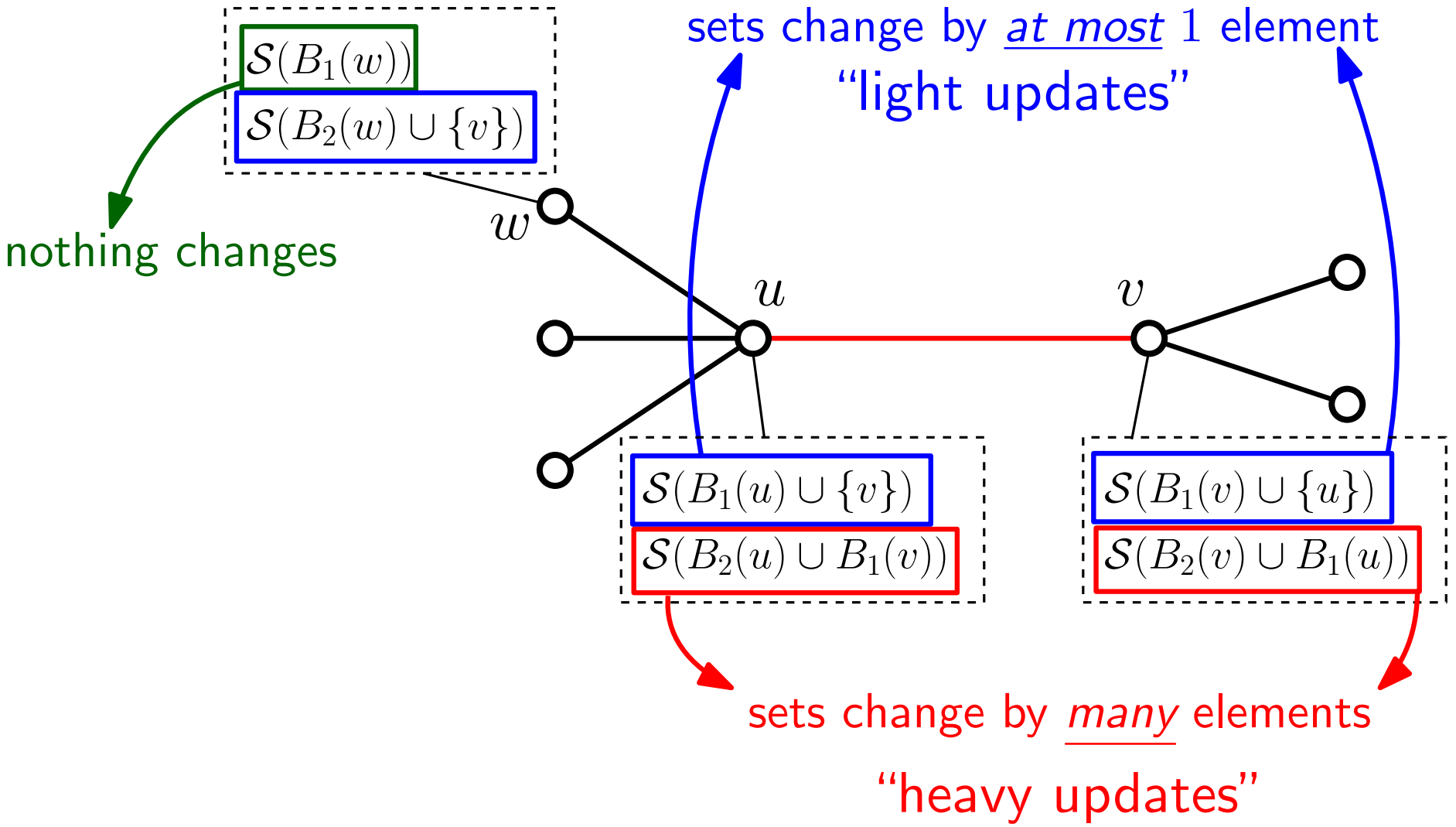
The Idea: Lazy Updates

We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.



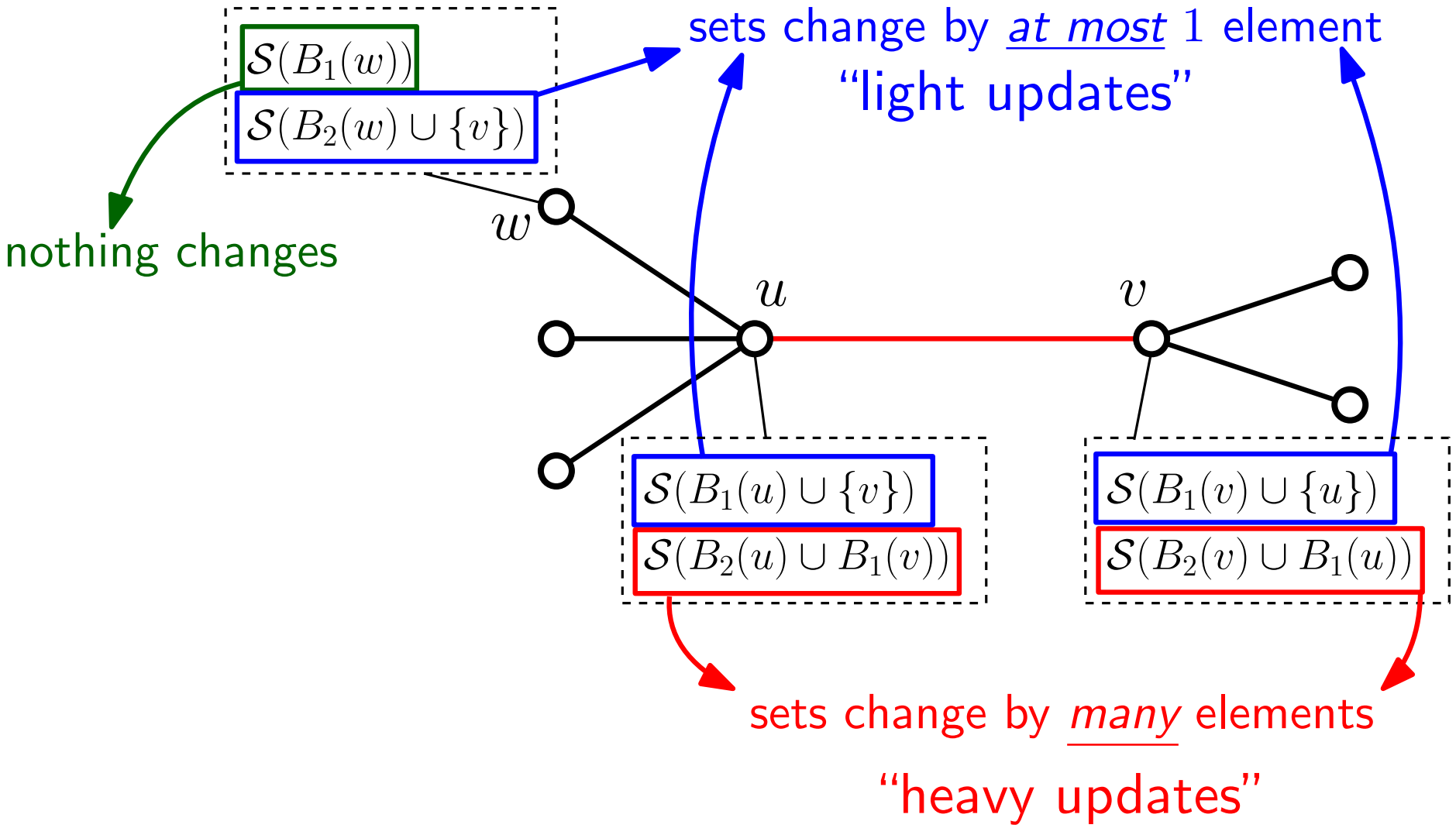
The Idea: Lazy Updates

We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.



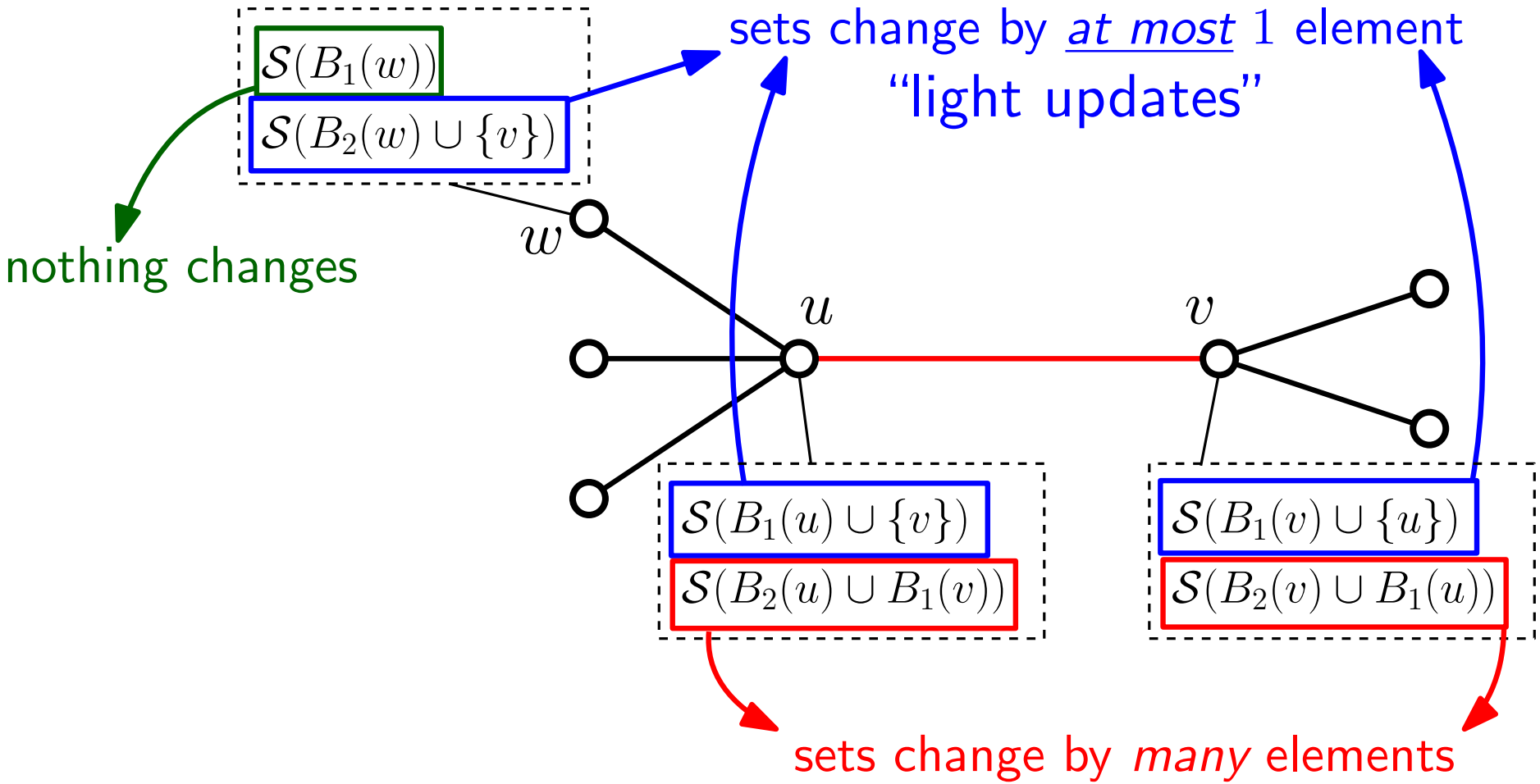
The Idea: Lazy Updates

We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.



The Idea: Lazy Updates

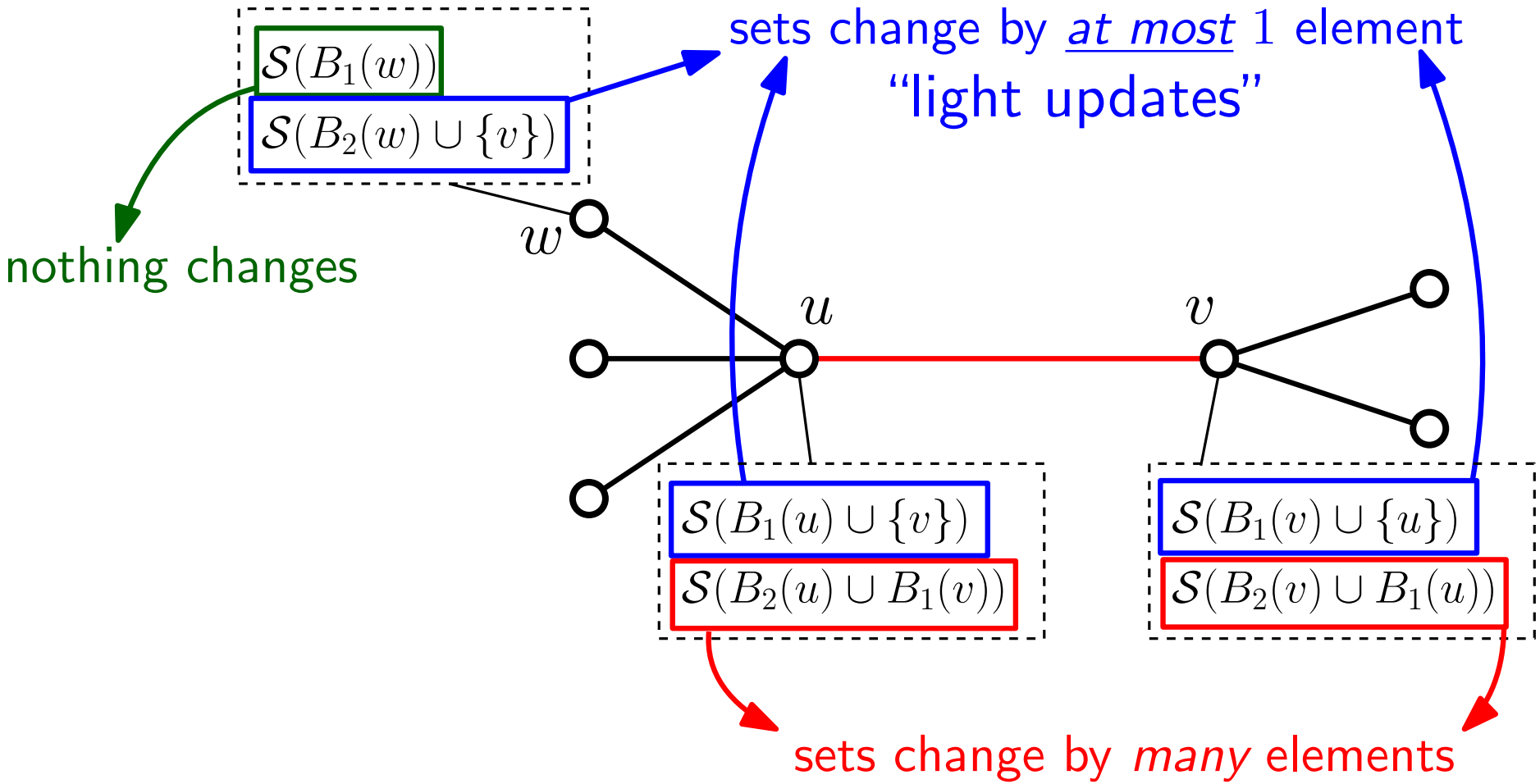
We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.



# of heavy updates: 2

The Idea: Lazy Updates

We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.

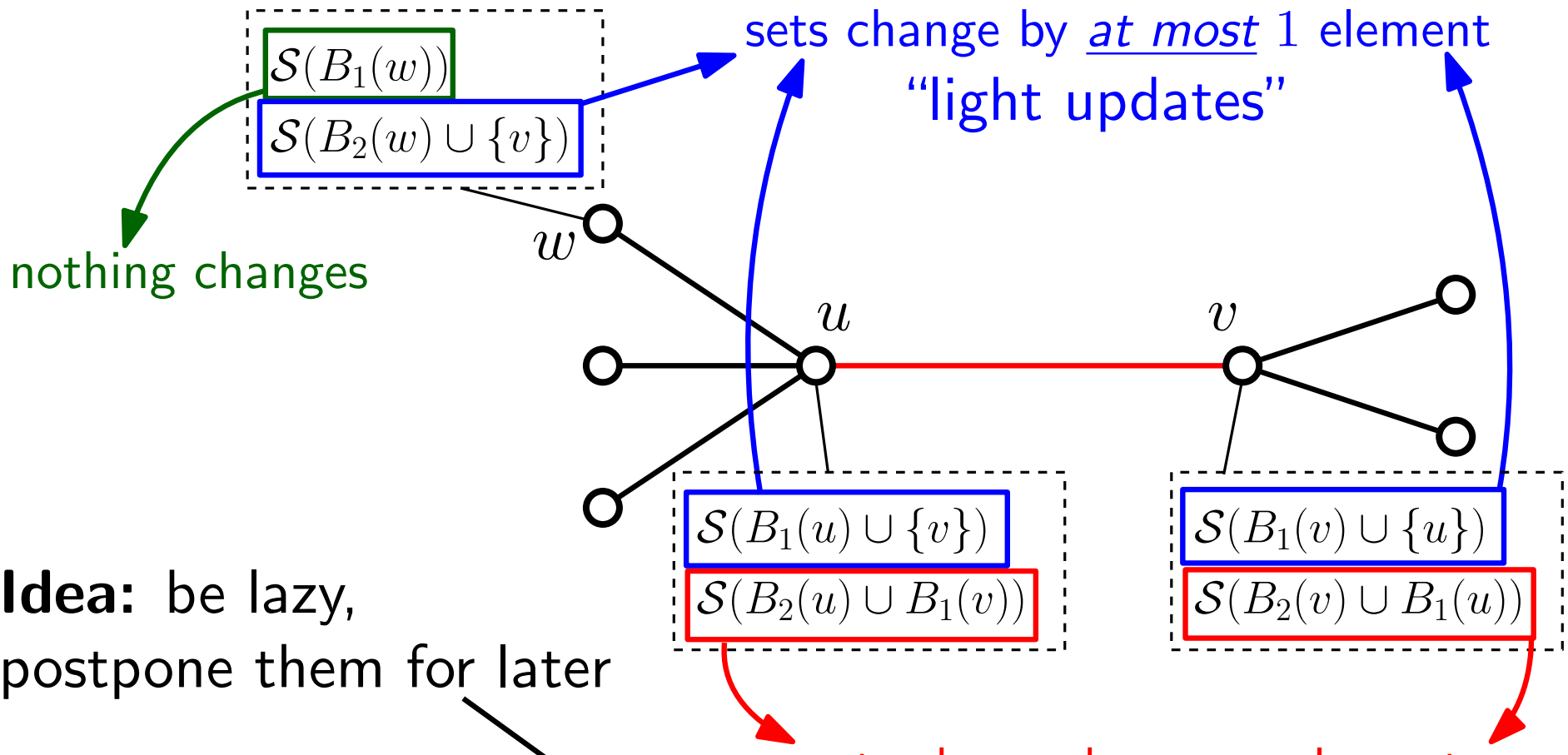


# of heavy updates: 2

# of light updates: $2 + \Delta_u + \Delta_v$

The Idea: Lazy Updates

We store $\mathcal{S}(B_1(u)), \mathcal{S}(B_2(u))$, for every vertex u.



Idea: be lazy,
postpone them for later

\# of heavy updates: 2
\# of light updates: $2 + \boxed{\Delta_u + \Delta_v}$

Lazy-Updates Algorithm

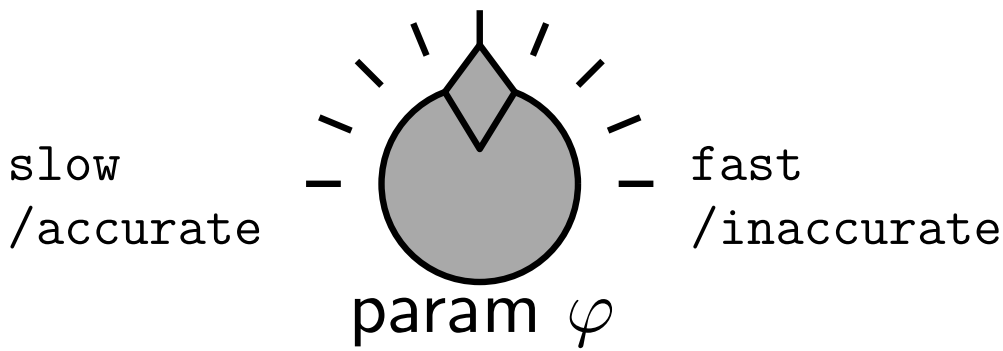
```
1  Function Insert((u, v)):
2      for x ∈ {u, v} do
3          let y ∈ {u, v} \ {x}
4          B̂1(x) ← B̂1(x) ∪ {y}
           // heavy update
5          B̂2(x) ← B̂2(x) ∪ B̂1(y)
6          δx ← δx + 1
7          if δx ≥ φ · Δx then
8              Δx ← Δx + δx
9              δx ← 0
10             foreach z ∈ N(x) do
                   // batch of light updates
11                 B̂2(z) ← B̂2(z) ∪ B̂1(x)
12             end
13         else
14             select k vertices w1, ..., wk ∈ N(x) u.a.r.
15             for i = 1, ..., k do
                   // batch of light updates
16                 B̂2(wi) ← B̂2(wi) ∪ B̂1(x)
17             end
18         end
19     end
20 end
```

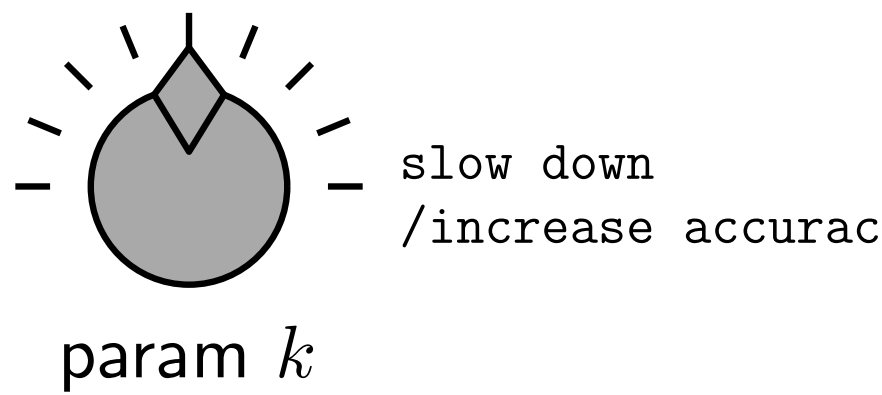
There parameters $\varphi \in (0, 1)$, $k \in \mathbb{N}$ determine the trade-offs between precision and *"laziness"*.

Lazy-Updates Algorithm

```
1  Function Insert((u, v)):
2      for x ∈ {u, v} do
3          let y ∈ {u, v} \ {x}
4          B̂1(x) ← B̂1(x) ∪ {y}
           // heavy update
5          B̂2(x) ← B̂2(x) ∪ B̂1(y)
6          δx ← δx + 1
7          if δx ≥ φ · Δx then
8              Δx ← Δx + δx
9              δx ← 0
10             foreach z ∈ N(x) do
                   // batch of light updates
11                 B̂2(z) ← B̂2(z) ∪ B̂1(x)
12             end
13         else
14             select k vertices w1, ..., wk ∈ N(x) u.a.r.
15             for i = 1, ..., k do
                   // batch of light updates
16                 B̂2(wi) ← B̂2(wi) ∪ B̂1(x)
17             end
18         end
19     end
20 end
```

There parameters $\varphi \in (0, 1)$, $k \in \mathbb{N}$ determine the trade-offs between precision and *"laziness"*.

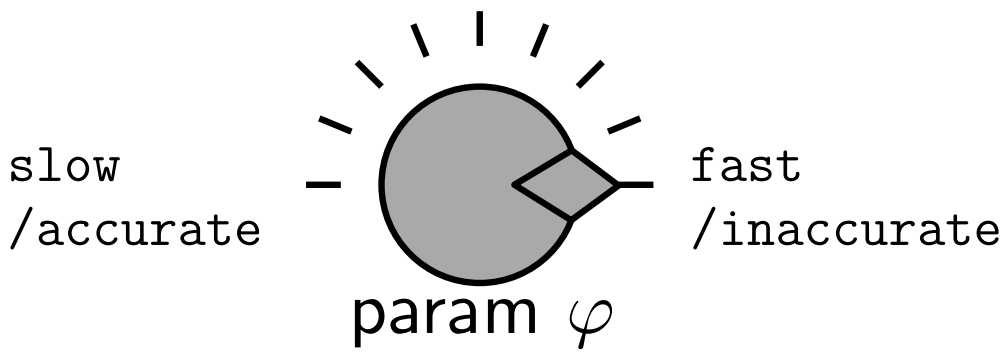


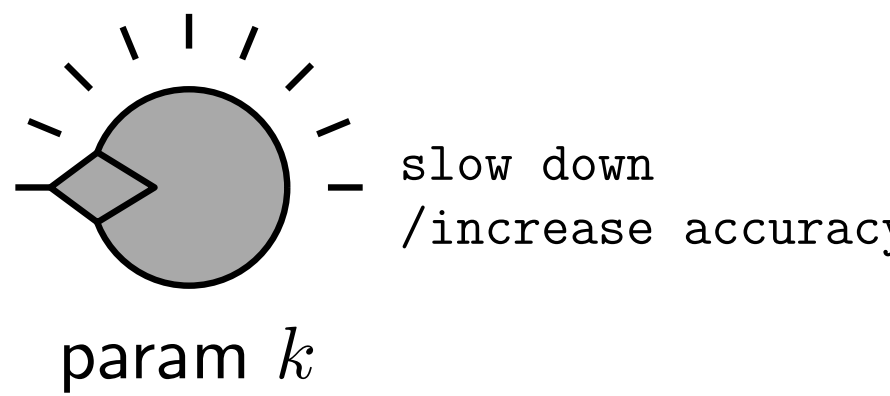


Lazy-Updates Algorithm

```
1  Function Insert((u, v)):
2      for x ∈ {u, v} do
3          let y ∈ {u, v} \ {x}
4          B̂1(x) ← B̂1(x) ∪ {y}
           // heavy update
5          B̂2(x) ← B̂2(x) ∪ B̂1(y)
6          δx ← δx + 1
7          if δx ≥ φ · Δx then
8              Δx ← Δx + δx
9              δx ← 0
10             foreach z ∈ N(x) do
                   // batch of light updates
11                 B̂2(z) ← B̂2(z) ∪ B̂1(x)
12             end
13         else
14             select k vertices w1, ..., wk ∈ N(x) u.a.r.
15             for i = 1, ..., k do
                   // batch of light updates
16                 B̂2(wi) ← B̂2(wi) ∪ B̂1(x)
17             end
18         end
19     end
20 end
```

There parameters $\varphi \in (0, 1)$, $k \in \mathbb{N}$ determine the trade-offs between precision and *"laziness"*.





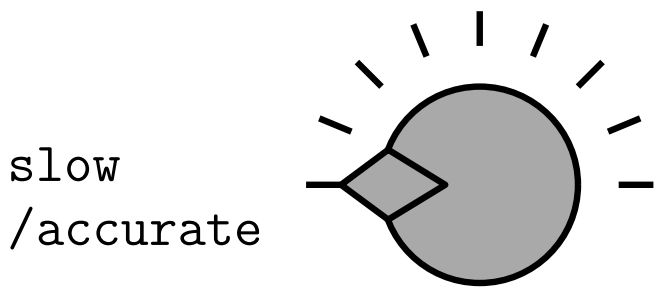
Parameter φ, k are like knobs:

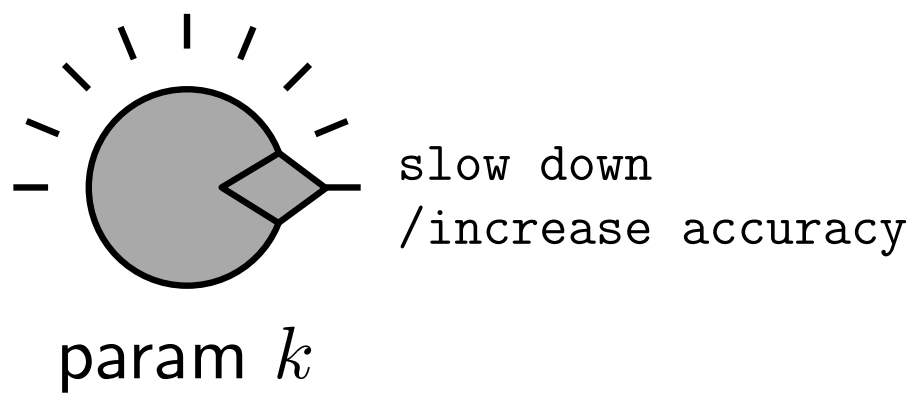
performance regime: $\uparrow \varphi - \downarrow k$

Lazy-Updates Algorithm

```
1  Function Insert((u, v)):
2      for x ∈ {u, v} do
3          let y ∈ {u, v} \ {x}
4          B̂1(x) ← B̂1(x) ∪ {y}
           // heavy update
5          B̂2(x) ← B̂2(x) ∪ B̂1(y)
6          δx ← δx + 1
7          if δx ≥ φ · Δx then
8              Δx ← Δx + δx
9              δx ← 0
10             foreach z ∈ N(x) do
                   // batch of light updates
11                 B̂2(z) ← B̂2(z) ∪ B̂1(x)
12             end
13         else
14             select k vertices w1, ..., wk ∈ N(x) u.a.r.
15             for i = 1, ..., k do
                   // batch of light updates
16                 B̂2(wi) ← B̂2(wi) ∪ B̂1(x)
17             end
18         end
19     end
20 end
```

There parameters $\varphi \in (0, 1)$, $k \in \mathbb{N}$ determine the trade-offs between precision and *"laziness"*.





Parameter φ, k are like knobs:

performance regime: $\uparrow \varphi - \downarrow k$

accurate regime: $\downarrow \varphi - \uparrow k$

Main Results

Performance: $O(\frac{1}{\varphi} + k)$ amortized number of sketch compositions.

Main Results

Performance: $O(\frac{1}{\varphi} + k)$ amortized number of sketch compositions.

Accuracy (under random insertions): By setting $\varphi = \frac{\varepsilon}{1-\varepsilon}$, the sketches will contain **at least** a $(1-\varepsilon)$-fraction of the actual $B_2(u)$, with high probability.

Main Results

Performance: $O(\frac{1}{\varphi} + k)$ amortized number of sketch compositions.

Accuracy (under random insertions): By setting $\varphi = \frac{\varepsilon}{1-\varepsilon}$, the sketches will contain **at least** a $(1-\varepsilon)$-fraction of the actual $B_2(u)$, with high probability.

Accuracy (under adversarial insertions): Lowerbound of $\Omega(\Delta \cdot (1-\varepsilon)^3)$ sketch compositions if we want $(1-\varepsilon)$-fraction.

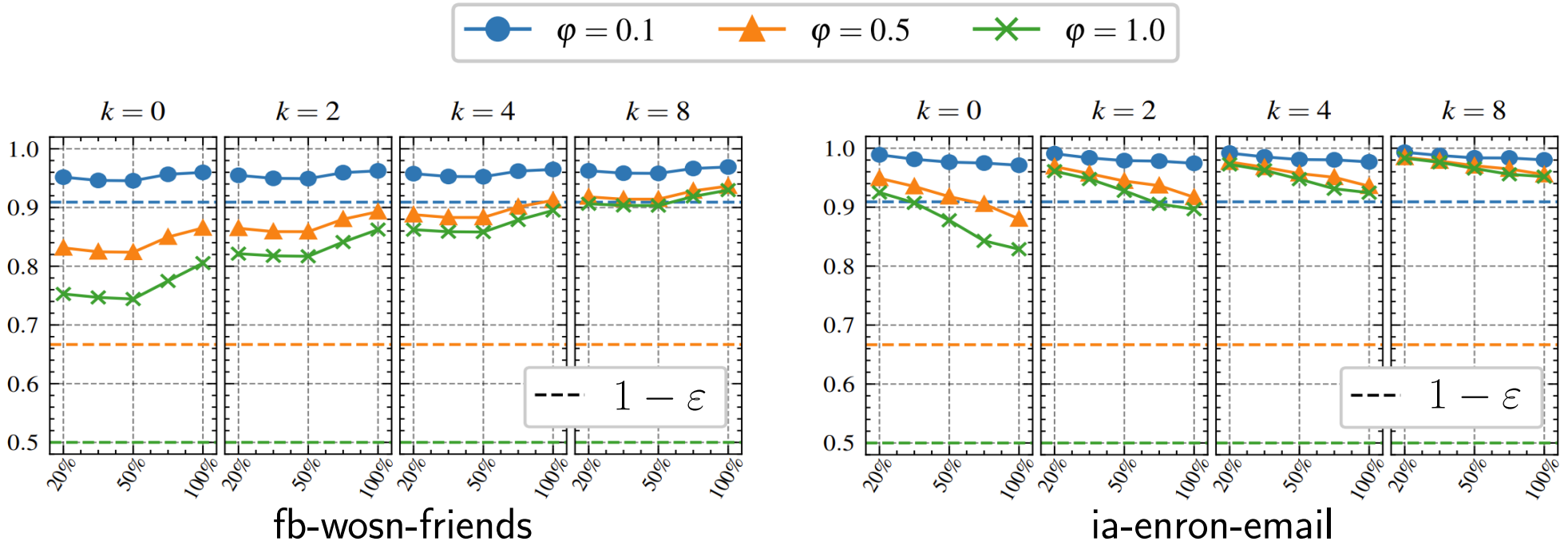
Main Results

Performance: $O(\frac{1}{\varphi} + k)$ amortized number of sketch compositions.

Accuracy (under random insertions): By setting $\varphi = \frac{\varepsilon}{1-\varepsilon}$, the sketches will contain **at least** a $(1-\varepsilon)$-fraction of the actual $B_2(u)$, with high probability.

Accuracy (under adversarial insertions): We characterize a broad family of (dense) graphs for which, by choosing the parameters φ, k appropriately, the Lazy-Update algorithm always guarantees a $(1-\varepsilon)$-fraction with high probability, even under adversarial edge insertion streams.

Lazy-Update in practice (real dynamic streams)



fb-wosn-friends

ia-enron-email

linux-kernel-reply

Performance

Basic Query: size estimation of B_2

Sketch: KMV probabilistic counter

Baseline: Keep everything always updated

Performance

Basic Query: size estimation of B_2

Sketch: KMV probabilistic counter

Baseline: Keep everything always updated

	k	φ 0.1	0.25	0.5	0.75	1
linux	0	14.39x	22.73x	30.27x	32.36x	35.65x
	2	12.00x	18.22x	22.03x	21.49x	23.18x
	4	11.52x	15.34x	17.63x	16.93x	17.54x
	8	9.57x	11.86x	12.80x	12.03x	12.22x
fb-wosn	0	5.48x	9.40x	11.15x	15.23x	16.22x
	2	5.22x	7.35x	7.16x	9.21x	9.67x
	4	4.32x	6.14x	6.05x	7.18x	7.23x
	8	3.79x	4.38x	4.91x	5.31x	5.22x
enron	0	4.17x	5.38x	6.64x	7.12x	7.70x
	2	3.61x	4.11x	4.68x	4.82x	5.27x
	4	3.05x	3.42x	3.78x	3.96x	4.21x
	8	2.51x	2.73x	2.85x	3.01x	3.12x
flickr	0	13.52x	19.97x	26.54x	31.51x	34.20x
	2	12.16x	16.41x	19.52x	21.93x	22.61x
	4	10.76x	13.96x	16.35x	17.17x	17.79x
	8	9.10x	11.19x	12.36x	12.99x	13.28x
youtube	0	41.61x	61.80x	77.08x	86.50x	93.73x
	2	38.16x	51.64x	60.39x	64.93x	65.51x
	4	36.08x	47.43x	52.26x	55.06x	55.90x
	8	33.33x	39.60x	42.73x	43.61x	43.82x

Speed-up wrt baseline

	k	$\varphi = 0.1$	$\varphi = 0.5$	$\varphi = 1$	baseline
linux	0	0.14 ± 0.12	0.19 ± 0.14	0.17 ± 0.11	0.12 ± 0.10
	2	0.13 ± 0.11	0.14 ± 0.10	0.16 ± 0.11	
	4	0.14 ± 0.09	0.17 ± 0.12	0.14 ± 0.10	
	8	0.14 ± 0.11	0.14 ± 0.09	0.13 ± 0.10	
fb-wosn	0	0.16 ± 0.12	0.15 ± 0.11	0.21 ± 0.12	0.14 ± 0.11
	2	0.13 ± 0.09	0.15 ± 0.10	0.17 ± 0.11	
	4	0.14 ± 0.11	0.15 ± 0.10	0.16 ± 0.11	
	8	0.16 ± 0.13	0.14 ± 0.10	0.15 ± 0.11	
enron	0	0.13 ± 0.10	0.16 ± 0.11	0.20 ± 0.14	0.13 ± 0.11
	2	0.14 ± 0.11	0.16 ± 0.12	0.16 ± 0.12	
	4	0.13 ± 0.12	0.15 ± 0.12	0.15 ± 0.12	
	8	0.13 ± 0.11	0.14 ± 0.10	0.16 ± 0.13	
flickr	0	0.17 ± 0.14	0.18 ± 0.12	0.17 ± 0.11	0.17 ± 0.14
	2	0.16 ± 0.12	0.12 ± 0.09	0.14 ± 0.10	
	4	0.14 ± 0.09	0.13 ± 0.10	0.16 ± 0.10	
	8	0.14 ± 0.11	0.13 ± 0.10	0.14 ± 0.09	
youtube	0	0.15 ± 0.11	0.15 ± 0.10	0.24 ± 0.11	0.14 ± 0.11
	2	0.16 ± 0.13	0.14 ± 0.10	0.19 ± 0.11	
	4	0.13 ± 0.11	0.13 ± 0.10	0.15 ± 0.11	
	8	0.13 ± 0.10	0.12 ± 0.09	0.15 ± 0.11	

Average Absolute Percentage Error

Performance

Basic Query: size estimation of B_2

Sketch: KMV probabilistic counter

Baseline: Keep everything always updated

	k	φ 0.1	0.25	0.5	0.75	1
linux	0	14.39x	22.73x	30.27x	32.36x	35.65x
	2	12.00x	18.22x	22.03x	21.49x	23.18x
	4	11.52x	15.34x	17.63x	16.93x	17.54x
	8	9.57x	11.86x	12.80x	12.03x	12.22x
fb-wosn	0	5.48x	9.40x	11.15x	15.23x	16.22x
	2	5.22x	7.35x	7.16x	9.21x	9.67x
	4	4.32x	6.14x	6.05x	7.18x	7.23x
	8	3.79x	4.38x	4.91x	5.31x	5.22x
enron	0	4.17x	5.38x	6.64x	7.12x	7.70x
	2	3.61x	4.11x	4.68x	4.82x	5.27x
	4	3.05x	3.42x	3.78x	3.96x	4.21x
	8	2.51x	2.73x	2.85x	3.01x	3.12x
flickr	0	13.52x	19.97x	26.54x	31.51x	34.20x
	2	12.16x	16.41x	19.52x	21.93x	22.61x
	4	10.76x	13.96x	16.35x	17.17x	17.79x
	8	9.10x	11.19x	12.36x	12.99x	13.28x
youtube	0	41.61x	61.80x	77.08x	86.50x	93.73x
	2	38.16x	51.64x	60.39x	64.93x	65.51x
	4	36.08x	47.43x	52.26x	55.06x	55.90x
	8	33.33x	39.60x	42.73x	43.61x	43.82x

Speed-up wrt baseline

	k	$\varphi = 0.1$	$\varphi = 0.5$	$\varphi = 1$	baseline
linux	0	0.14 ± 0.12	0.19 ± 0.14	0.17 ± 0.11	0.12 ± 0.10
	2	0.13 ± 0.11	0.14 ± 0.10	0.16 ± 0.11	
	4	0.14 ± 0.09	0.17 ± 0.12	0.14 ± 0.10	
	8	0.14 ± 0.11	0.14 ± 0.09	0.13 ± 0.10	
fb-wosn	0	0.16 ± 0.12	0.15 ± 0.11	0.21 ± 0.12	0.14 ± 0.11
	2	0.13 ± 0.09	0.15 ± 0.10	0.17 ± 0.11	
	4	0.14 ± 0.11	0.15 ± 0.10	0.16 ± 0.11	
	8	0.16 ± 0.13	0.14 ± 0.10	0.15 ± 0.11	
enron	0	0.13 ± 0.10	0.16 ± 0.11	0.20 ± 0.14	0.13 ± 0.11
	2	0.14 ± 0.11	0.16 ± 0.12	0.16 ± 0.12	
	4	0.13 ± 0.12	0.15 ± 0.12	0.15 ± 0.12	
	8	0.13 ± 0.11	0.14 ± 0.10	0.16 ± 0.13	
flickr	0	0.17 ± 0.14	0.18 ± 0.12	0.17 ± 0.11	0.17 ± 0.14
	2	0.16 ± 0.12	0.12 ± 0.09	0.14 ± 0.10	
	4	0.14 ± 0.09	0.13 ± 0.10	0.16 ± 0.10	
	8	0.14 ± 0.11	0.13 ± 0.10	0.14 ± 0.09	
youtube	0	0.15 ± 0.11	0.15 ± 0.10	0.24 ± 0.11	0.14 ± 0.11
	2	0.16 ± 0.13	0.14 ± 0.10	0.19 ± 0.11	
	4	0.13 ± 0.11	0.13 ± 0.10	0.15 ± 0.11	
	8	0.13 ± 0.10	0.12 ± 0.09	0.15 ± 0.11	

Average Absolute Percentage Error

Performance

Basic Query: size estimation of B_2

Sketch: KMV probabilistic counter

Baseline: Keep everything always updated

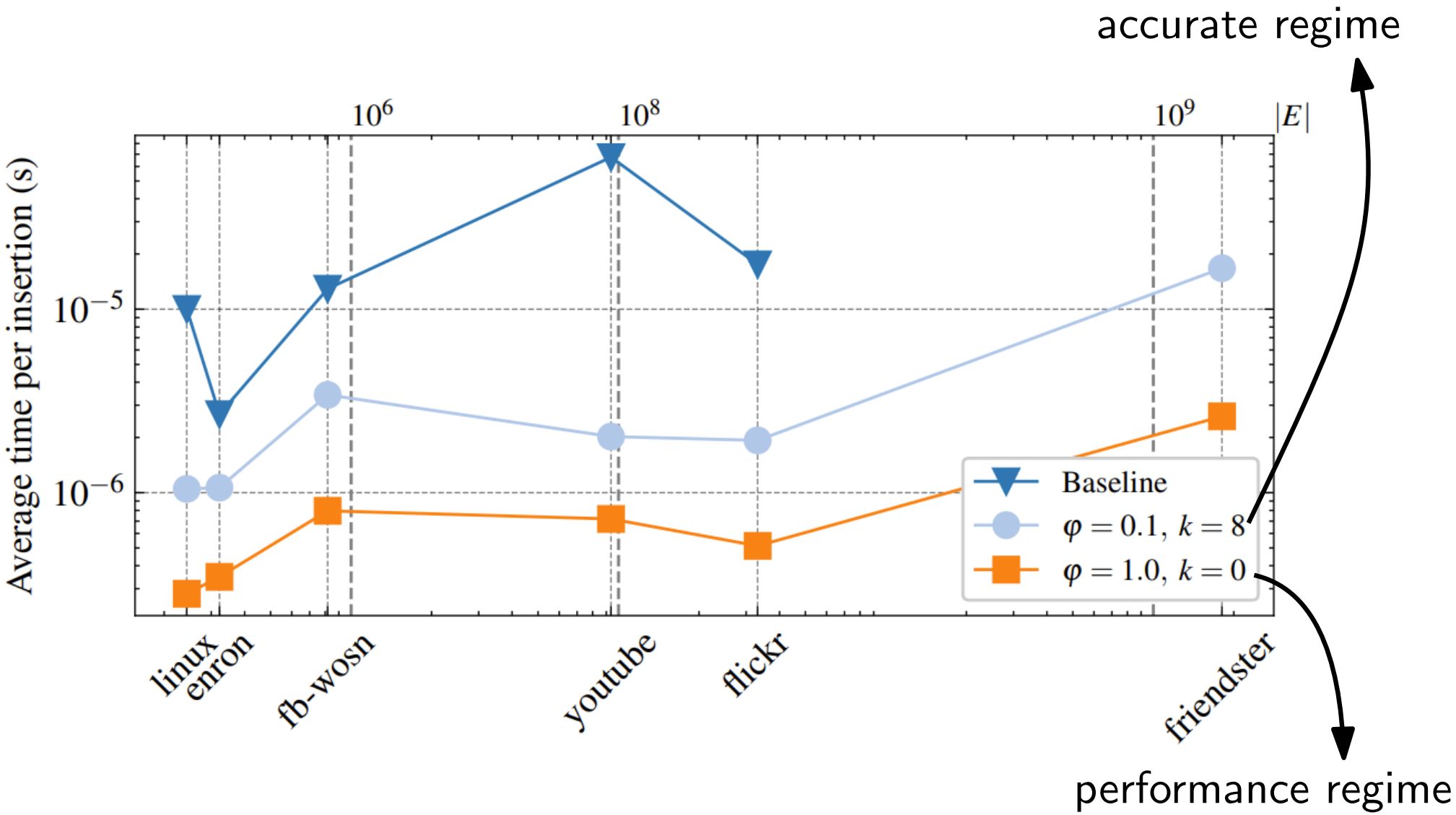
	k	φ 0.1	0.25	0.5	0.75	1
linux	0	14.39x	22.73x	30.27x	32.36x	35.65x
	2	12.00x	18.22x	22.03x	21.49x	23.18x
	4	11.52x	15.34x	17.63x	16.93x	17.54x
	8	9.57x	11.86x	12.80x	12.03x	12.22x
fb-wosn	0	5.48x	9.40x	11.15x	15.23x	16.22x
	2	5.22x	7.35x	7.16x	9.21x	9.67x
	4	4.32x	6.14x	6.05x	7.18x	7.23x
	8	3.79x	4.38x	4.91x	5.31x	5.22x
enron	0	4.17x	5.38x	6.64x	7.12x	7.70x
	2	3.61x	4.11x	4.68x	4.82x	5.27x
	4	3.05x	3.42x	3.78x	3.96x	4.21x
	8	2.51x	2.73x	2.85x	3.01x	3.12x
flickr	0	13.52x	19.97x	26.54x	31.51x	34.20x
	2	12.16x	16.41x	19.52x	21.93x	22.61x
	4	10.76x	13.96x	16.35x	17.17x	17.79x
	8	9.10x	11.19x	12.36x	12.99x	13.28x
youtube	0	41.61x	61.80x	77.08x	86.50x	93.73x
	2	38.16x	51.64x	60.39x	64.93x	65.51x
	4	36.08x	47.43x	52.26x	55.06x	55.90x
	8	33.33x	39.60x	42.73x	43.61x	43.82x

Speed-up wrt baseline

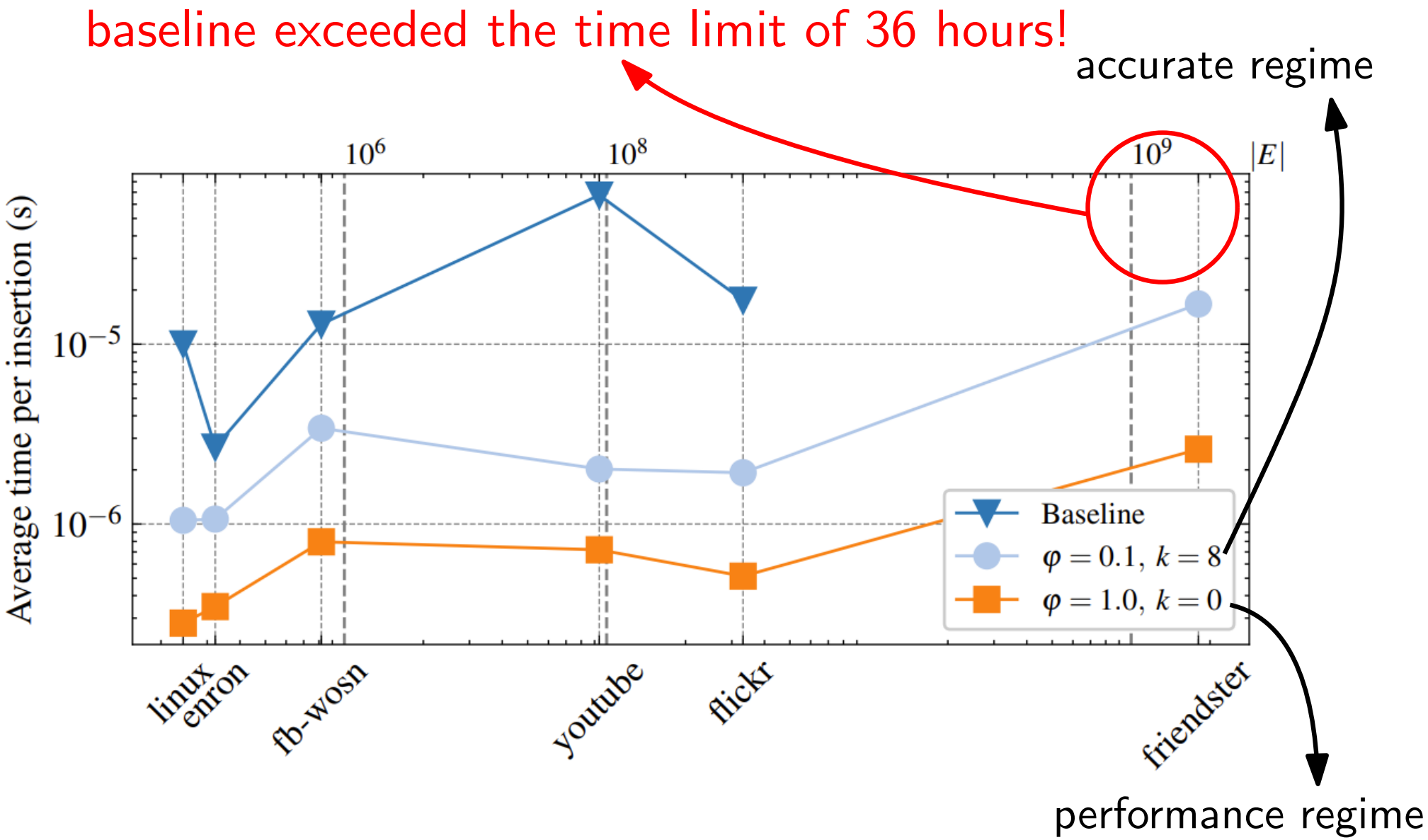
	k	$\varphi = 0.1$	$\varphi = 0.5$	$\varphi = 1$	baseline
linux	0	0.14 ± 0.12	0.19 ± 0.14	0.17 ± 0.11	0.12 ± 0.10
	2	0.13 ± 0.11	0.14 ± 0.10	0.16 ± 0.11	
	4	0.14 ± 0.09	0.17 ± 0.12	0.14 ± 0.10	
	8	0.14 ± 0.11	0.14 ± 0.09	0.13 ± 0.10	
fb-wosn	0	0.16 ± 0.12	0.15 ± 0.11	0.21 ± 0.12	0.14 ± 0.11
	2	0.13 ± 0.09	0.15 ± 0.10	0.17 ± 0.11	
	4	0.14 ± 0.11	0.15 ± 0.10	0.16 ± 0.11	
	8	0.16 ± 0.13	0.14 ± 0.10	0.15 ± 0.11	
enron	0	0.13 ± 0.10	0.16 ± 0.11	0.20 ± 0.14	0.13 ± 0.11
	2	0.14 ± 0.11	0.16 ± 0.12	0.16 ± 0.12	
	4	0.13 ± 0.12	0.15 ± 0.12	0.15 ± 0.12	
	8	0.13 ± 0.11	0.14 ± 0.10	0.16 ± 0.13	
flickr	0	0.17 ± 0.14	0.18 ± 0.12	0.17 ± 0.11	0.17 ± 0.14
	2	0.16 ± 0.12	0.12 ± 0.09	0.14 ± 0.10	
	4	0.14 ± 0.09	0.13 ± 0.10	0.16 ± 0.10	
	8	0.14 ± 0.11	0.13 ± 0.10	0.14 ± 0.09	
youtube	0	0.15 ± 0.11	0.15 ± 0.10	0.24 ± 0.11	0.14 ± 0.11
	2	0.16 ± 0.13	0.14 ± 0.10	0.19 ± 0.11	
	4	0.13 ± 0.11	0.13 ± 0.10	0.15 ± 0.11	
	8	0.13 ± 0.10	0.12 ± 0.09	0.15 ± 0.11	

Average Absolute Percentage Error

Average time per insertion

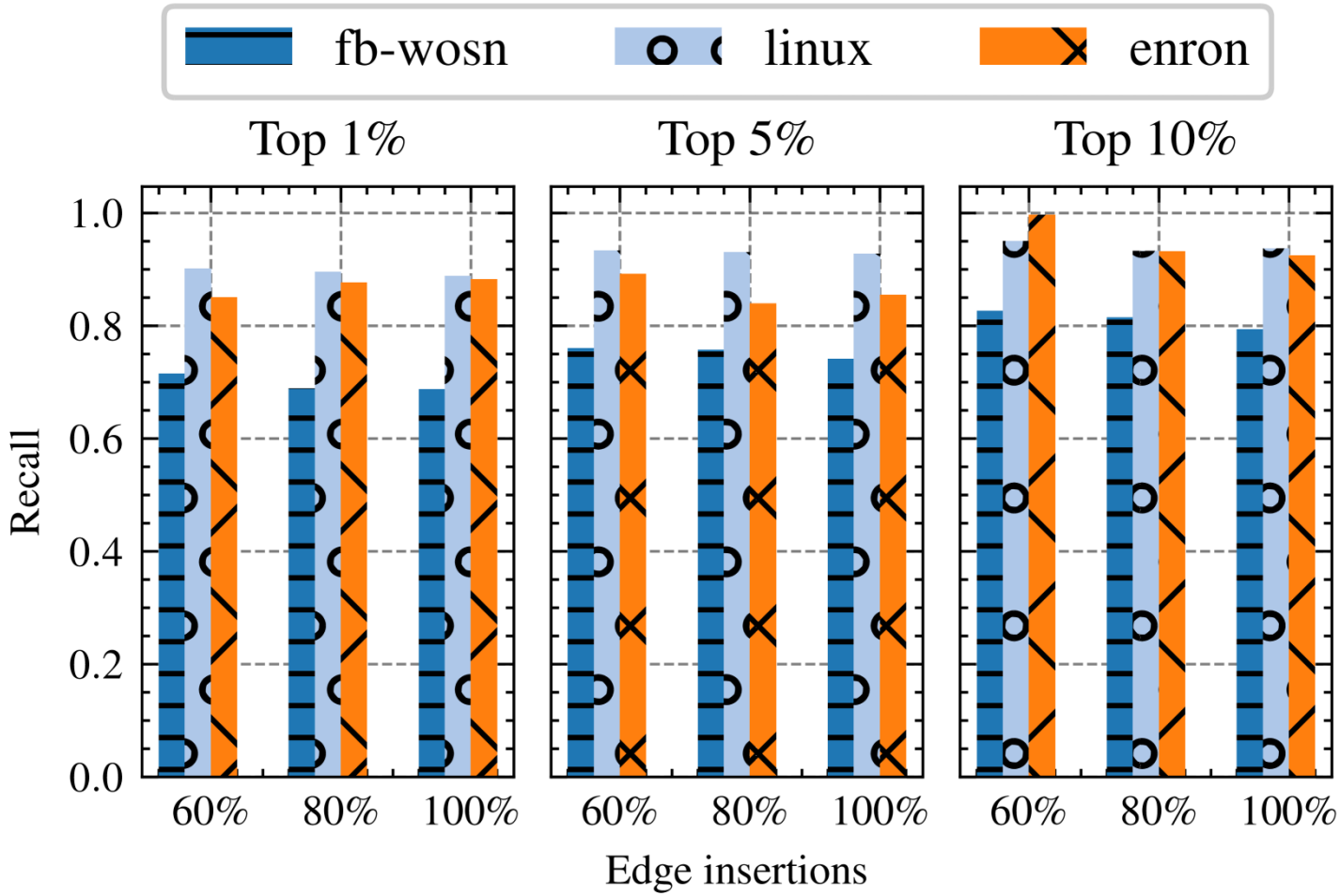


Average time per insertion



Effectiveness of approximation

Downstream task: maintaining top-k nodes with higher *harmonic centrality* (in a dynamic network).



Open problems

Open problems

i) Study the **fully-dynamic** case

Open problems

i) Study the **fully-dynamic** case

- we do not know if it is possible to apply a *lazy approach*

Open problems

i) Study the **fully-dynamic** case

- we do not know if it is possible to apply a *lazy approach*
- data sketches that allow for *removal* are needed

Open problems

i) Study the **fully-dynamic** case

- we do not know if it is possible to apply a *lazy approach*
- data sketches that allow for *removal* are needed

ii) Study the case of d-hop neighborhoods with $d > 2$ (even only incremental).

Open problems

i) Study the **fully-dynamic** case

- we do not know if it is possible to apply a *lazy approach*
- data sketches that allow for *removal* are needed

ii) Study the case of d-hop neighborhoods with $d > 2$ (even only incremental).

iii) Use our framework to solve *other downstream tasks* beyond those discussed.

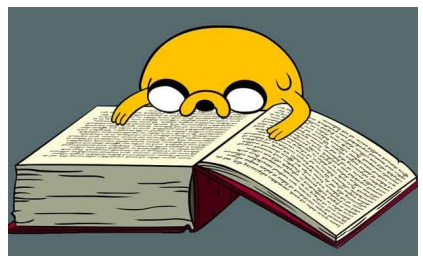
Thanks for your attention

Full version on arXiv

Poster

Presentation

About me



$$L_i(u) = \{v \mid d(u,v) = i\}$$

$$B_i(u) = \{v \mid d(u,v) \leq i\}$$

$$|L_i(u)| = |B_i(u)| - |B_{i-1}(u)|$$

Harmonic Centrality

$$HC(u) = \sum_{v \neq u} \frac{1}{d(u,v)} = \frac{|L_1(u)|}{1} + \frac{|L_2(u)|}{2} + \frac{|L_3(u)|}{3} + \frac{|L_4(u)|}{4} \ldots$$

Truncated Harmonic Centrality

$$HC_2(u) = \frac{|L_1(u)|}{1} + \frac{|L_2(u)|}{2} = |B_1(u)| - 1 + \frac{|B_2(u)| - |B_1(u)|}{2}$$