

Fully Dynamic Algorithms for Coloring Triangle-Free Graphs

Sepehr Assadi* Helia Yazdanyar*

* University of Waterloo

Problem Definition

- **Vertex Coloring Problem:** Assign colors to vertices of graph $G = (V, E)$ so adjacent vertices have different colors.
- Any **triangle-free** graph G with maximum degree Δ admits $O(\Delta/\ln \Delta)$ coloring. [Johansson '96]
- **Fully Dynamic Setting:** Underlying vertices V are fixed, but edges can be inserted or deleted.
- **Goal:** We want to maintain a $O(\Delta/\ln \Delta)$ coloring over all updates on a triangle-free fully dynamic graph.

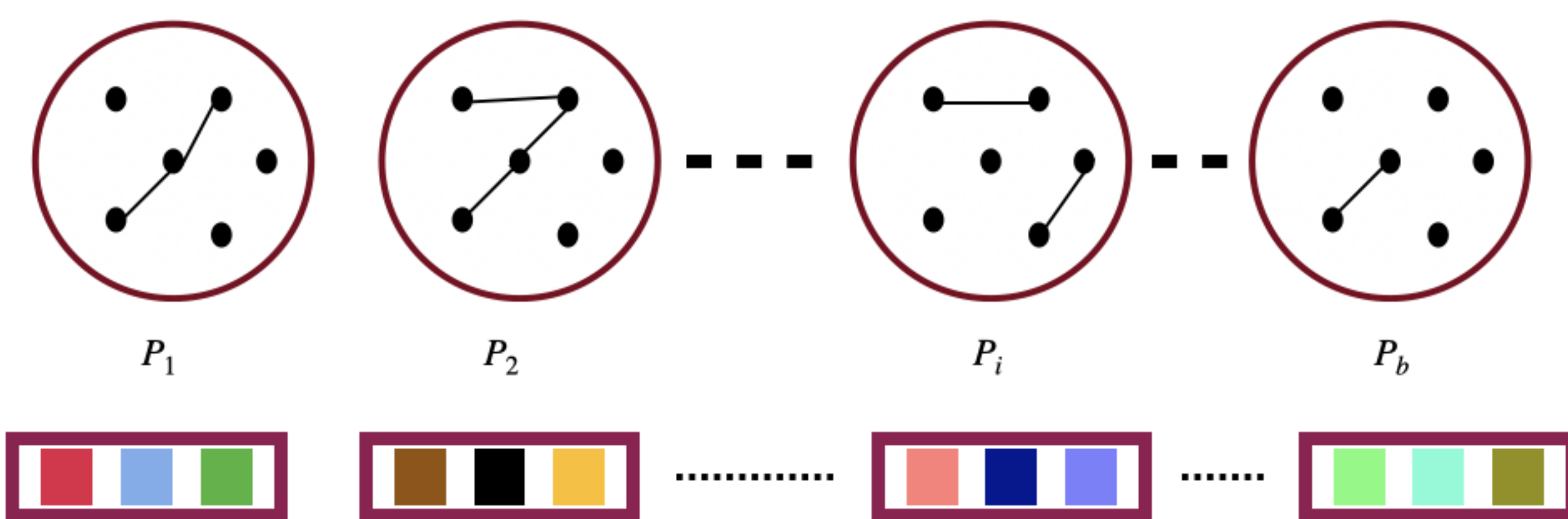
Our Result

There is a randomized algorithm that maintains an $O(\Delta/\ln \Delta)$ coloring of any n -vertex fully dynamic triangle-free graph in $O(\Delta^{o(1)} \log n)$ amortized update time with high probability against adaptive adversary.

A Recursive Algorithm

Degree Splitting: A strategy for $O(\Delta)$ coloring graphs [BCHN'18]

- Partition the vertices and the palette of colors into k parts.
- **Recurse** on each part with its corresponding palette



We want to use the same strategy:

$O(\Delta)$	$O(\Delta/\ln \Delta)$
Degree splitting	Degree splitting requires changes in the techniques
Base Case: Greedy in $O(\Delta)$ time.	Greedy does not work!

- Is there a $\text{poly}(\Delta) \log n$ time base case algorithm?

$O(\Delta/\ln \Delta)$ Coloring Algorithm in $\tilde{O}(\Delta^2)$ time

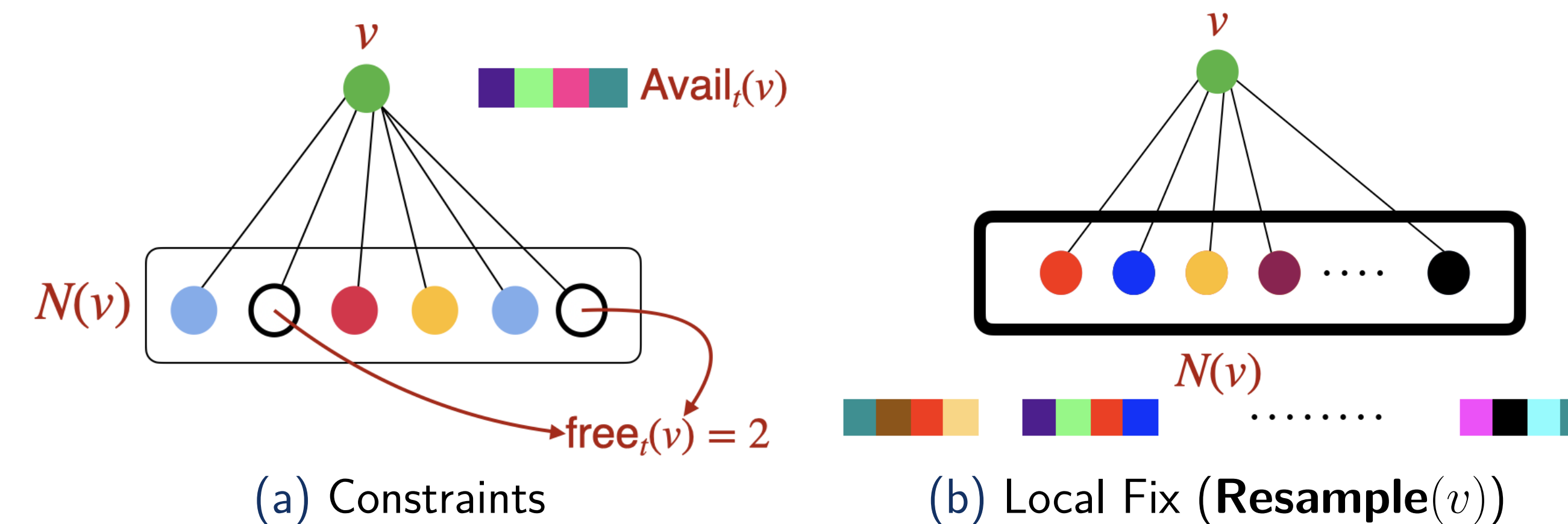
- Split the palette of $q = O(\Delta/\ln \Delta)$ colors into:
 $C_1 = \{1, \dots, q\}$, and $C_2 = \{q+1, \dots, (1+\varepsilon)q\}$
- We want to maintain a **partial coloring** $\varphi: V \rightarrow C_1 \cup \{\perp\}$ such that **most** vertices.
- Then we extend φ using C_2 and a simple **greedy algorithm**.

Maintaining the Partial Coloring φ

- To maintain φ over all updates, we need some constraints.
- If some vertex v does not satisfy a constraint, we say it has a **flaw**.

Constraints: For each vertex v

$$|\text{Avail}_t(v)| \geq \Delta^\varepsilon, \text{ and } |\text{free}_t(v)| \leq \frac{\varepsilon q}{2}$$

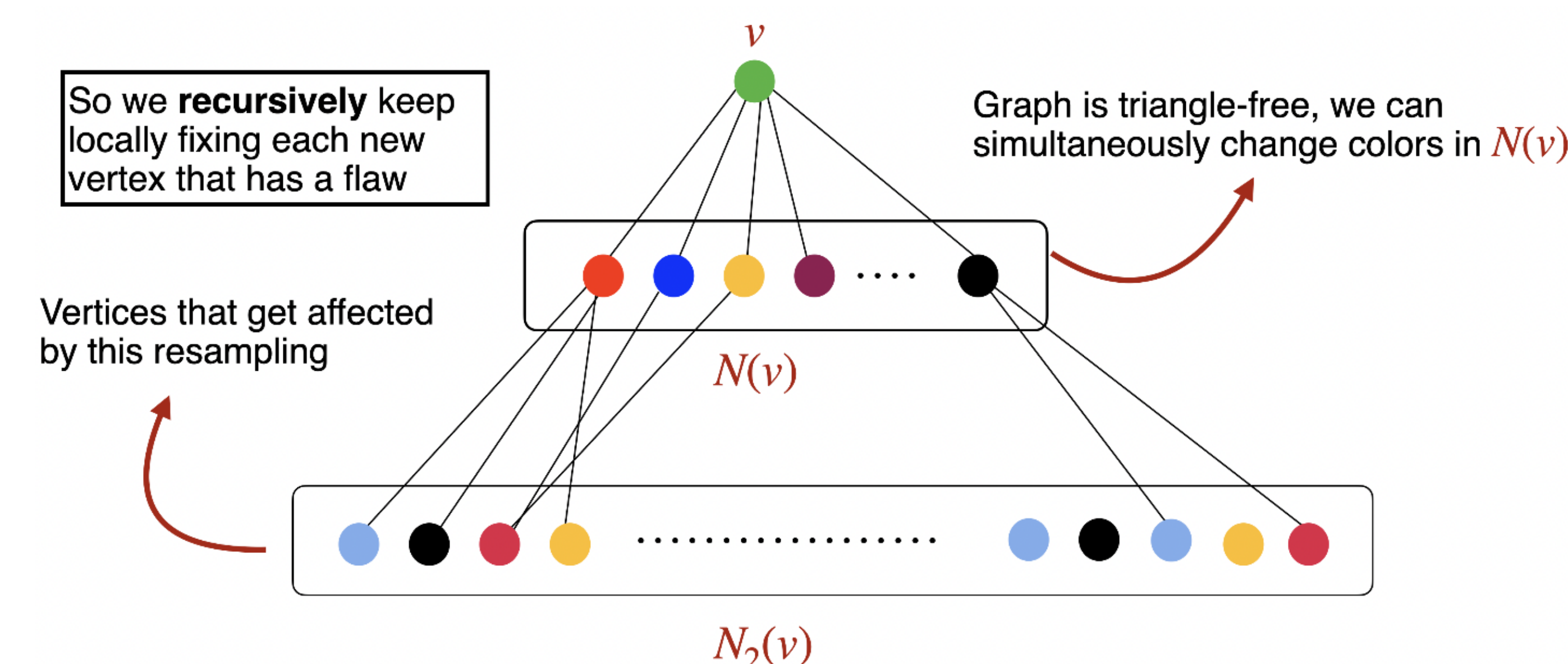


- To maintain these constraints, we **locally fix** any vertex that has a **flaw** by **Resampling** the color of its neighbors (Figure (b)).

Lemma: For each vertex v when calling **Resample**(v)

$$\Pr(v \text{ has a flaw}) \leq 2^{-\Delta^\varepsilon}.$$

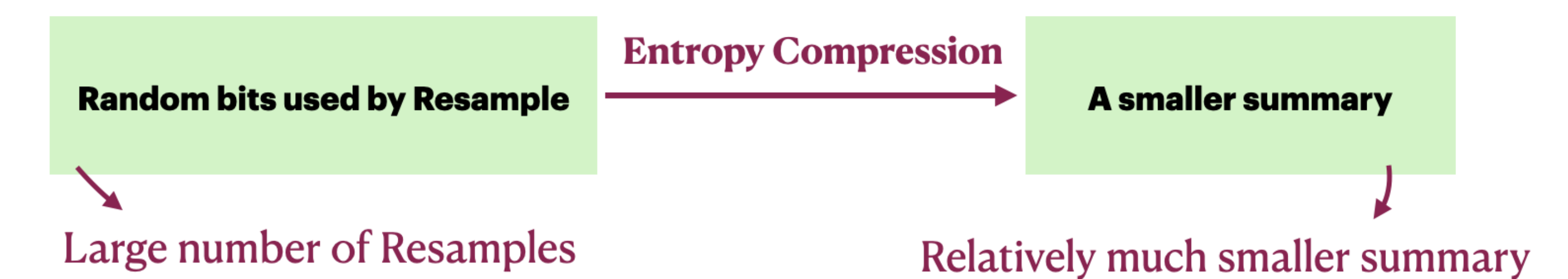
- Thus, with high probability, **Resample** resolves the flaws in $O(\Delta^2)$ time.



- So what is the progress we make by running **Resample**?

Dynamic Entropy Compression

- How many time do we invoke **Resample** over T updates? (let this value be R_T)



- We keep a **journal** of all the **updates** and calls to **Resample**.
- **Key Idea:** instead of writing the exact coloring on $N(v)$ that caused a call to **Resample**(v) in the log, we write the index pointing to this coloring in **Bad**(v) (set of all colorings causing a flaw for v).
- By **Lemma:** $|\text{Bad}(v)| \leq 2^{-\Delta^\varepsilon} \cdot (\# \text{ all possible colorings for } N(v))$.

Algorithm:

Update 1 ($v_1, v_1, \pm$)	Resample(v)	Resample(u)		Update T ($v_s, v_s, \pm$)		Resample(w)	Finish
---------------------------------	-----------------	-----------------	-------	---------------------------------	-------	-----------------	--------

Log:

Update ($i, j, \pm$)	($v, i \in \text{Bad}_1(v)$) causing the flaw	($u, i \in \text{Bad}_1(u)$) causing the flaw	...	Update ($s, r, \pm$)	...	($w, i \in \text{Bad}_T(w)$) causing the flaw	Final φ_T
---------------------------	--	--	-----	---------------------------	-----	--	----------------------

- **Claim:** We can **recover** all the random bits.
- **Random bits** used by algorithm = $R_T \cdot (\Delta \log \Delta^\varepsilon)$
- The size of the compression (log) =

$$\underbrace{T \cdot (2 \log n + 2)}_{\text{Updates}} + \underbrace{R_T \cdot (\Delta \log \Delta^\varepsilon - \Delta^\varepsilon)}_{\text{Calls to Resample}} + \underbrace{n \log \Delta}_{\text{Final coloring}}$$
- Thus, by information theory facts we cannot compress the random bits this much unless $R_T \leq T \log n + n$.

Amortized Update Time

- The amortized update time to maintain φ is w.h.p

$$\frac{O(\Delta^2) \cdot R_T}{T} = \frac{O(\Delta^2) \cdot (T \log n + n)}{T} = O(\Delta^2 \log n)$$
- We extend φ using **simple greedy algorithm** that takes another $O(\Delta \log n)$ time.