Streaming and Massively Parallel Algorithms for Edge Coloring

Soheil Behnezhad, Mahsa Derakhshan, Marina Knittel,
MohammadTaghi Hajiaghayi, **Hamed Saleh** [ESA'19]

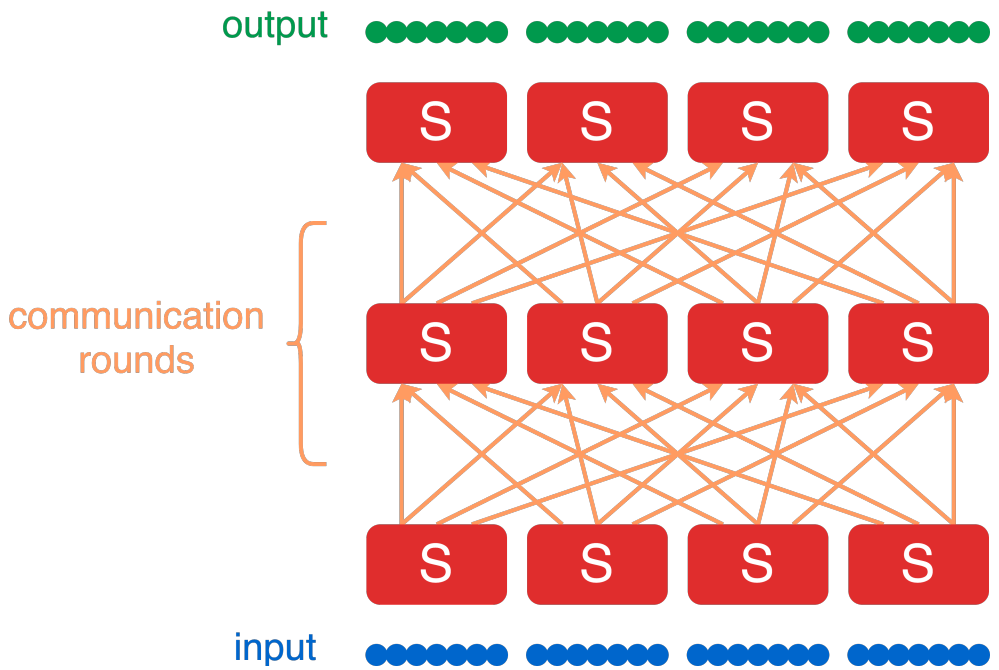
Massively Parallel Computation

- Modern frameworks for Large-scale parallel/distributed data processing: MapReduce, Hadoop, Spark.
- Key Idea: distribute the workload among several machines.
- The **MPC** model: A theoretical model to abstract out the computational power of these frameworks.

[Karloff et. al 2010]
[Goodrich et. al 2011]
[Beame et. al 2013]
[Andoni et. al 2014]

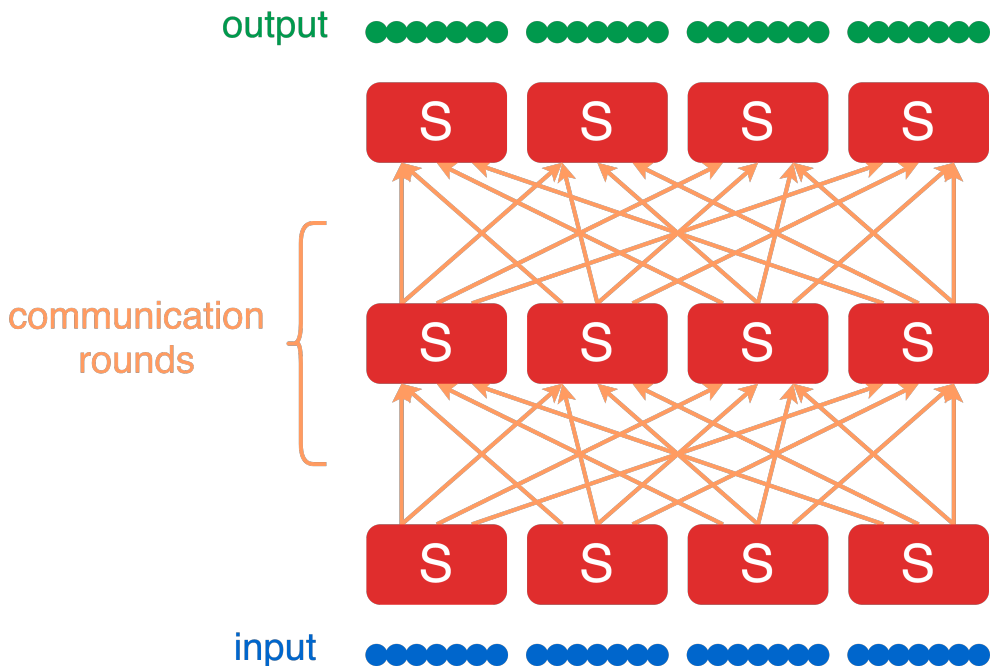
Massively Parallel Computation

- There are M machines each with memory S.
- The input of length N is initially (randomly) distributed among the machines.
  - Sublinear space $S = o(N)$.
  - Usually $N = O(S \cdot M)$.
- The data is processed in several **synchronous rounds**.



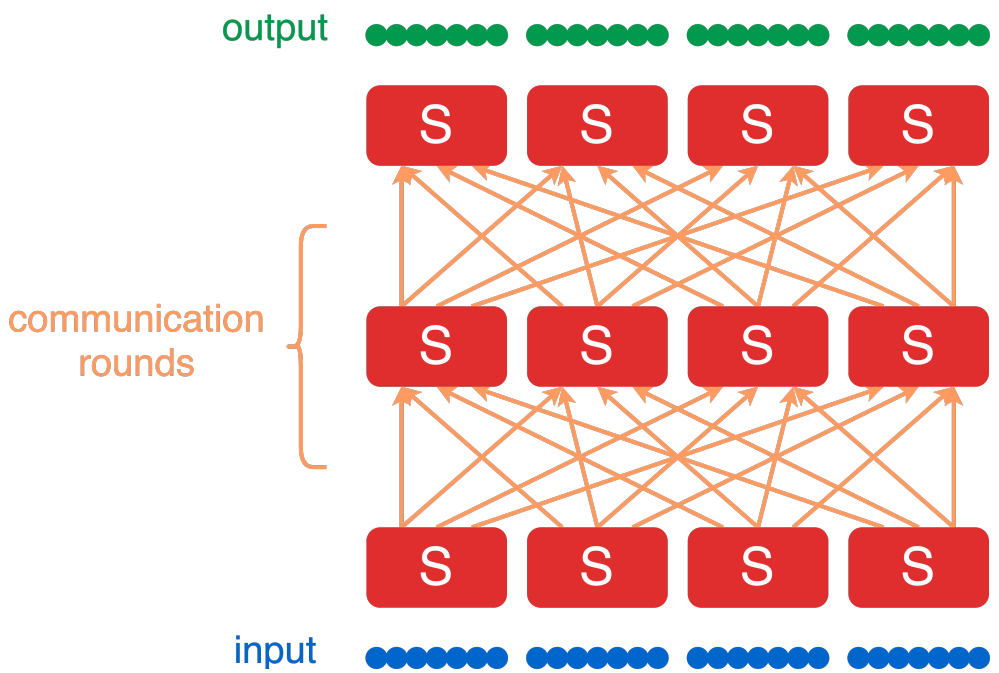
Massively Parallel Computation

- There are M machines each with memory S.
- The data is processed in several **synchronous rounds**. In each round,
  - Machines perform arbitrary computation on their **local** data.
  - Machines **communicate** with each other. Total **incoming/outgoing** messages of each machine is bounded by $O(S)$ words.



Massively Parallel Computation

- There are M machines each with memory S.
- The data is processed in several **synchronous rounds**.
- Main bottleneck: **Communication**. We wish for algorithms with very small number of rounds.
  - Often **sub-logarithmic** rounds.



Related **distributed**/**parallel** models

- The **PRAM** model (shared memory)
  - Any PRAM algorithm running in time $t = t(n)$ can be simulated in $O(t)$ MPC rounds.

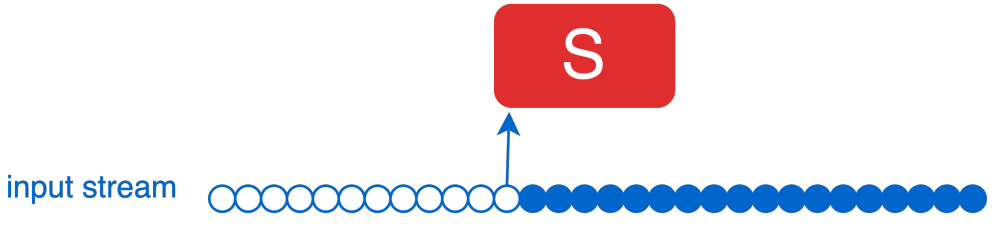
[Karloff et. al 2010]

- The **LOCAL**, **CONGEST**, and **congested-clique** models
  - Similar techniques can be used for both MPC and these distributed models.
  - Congested-clique is almost equivalent to MPC (in terms of the number of rounds).

[Behnezhad et. al 2018]

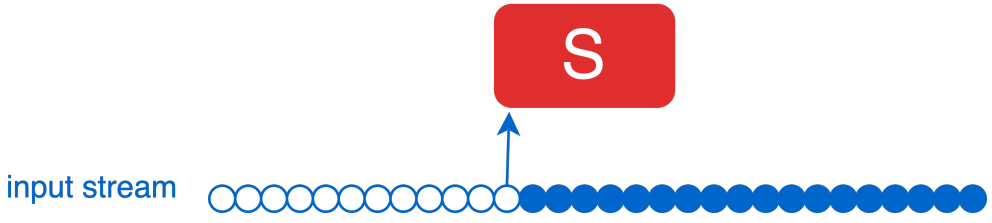
Streaming

- There is a single machines with memory S.
- The input of length N is streamed into the machine.
  - Sublinear space $S = o(N)$.
- The data is processed in several **passes**.



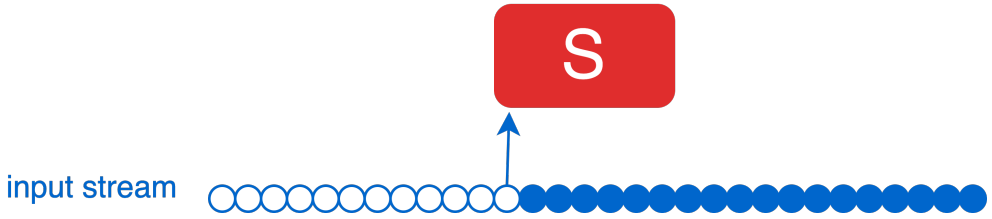
Streaming

- There is a single machines with memory S.
- The data is processed in several **passes**. In each pass:
  - The input entries arrive **sequentially** and **one by one** in a specific order.
  - The order can be either **random** or **adversarial**.



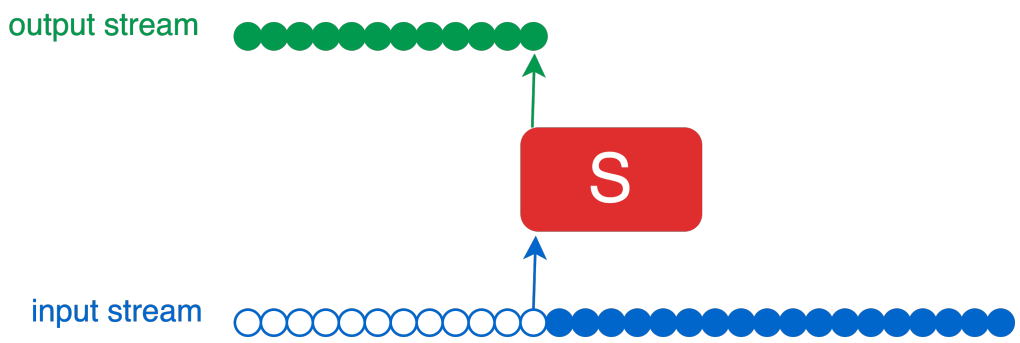
Streaming

- There is a single machines with memory S.
- The data is processed in several **passes**.
- There is a **trade-off** between the number of passes and the space of the machine.



W-streaming

- There is a single machines with memory S.
- What if the **output** also doesn't fit in the memory?
  - We use the **W-streaming** model.
  - The output is also streamed.



Semi-streaming and **Semi-**MPC

- In many graph problems, we assume $S = O(|V|)$, where the input graph is given as $G = (V, E)$.
- The variant of the streaming model in which $S = O(|V|)$ is called **Semi-**streaming.

[Feigenbaum et. al 2005]
[McGregor 2014]

- This restriction is stricter on **dense** graphs and less strict on **sparse** graphs.
- The standard variants are either too trivial or too hard on sparse graphs.
- Similarly, the **Semi-**MPC model is also defined.

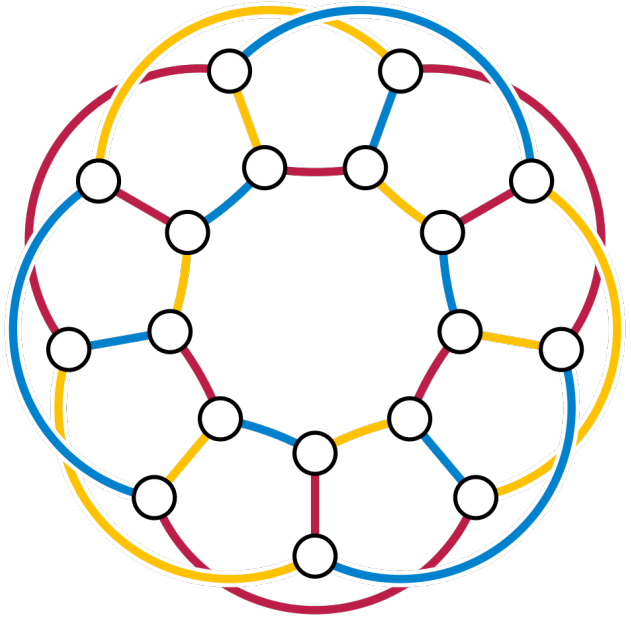
Edge Coloring

Streaming and Massively Parallel Algorithms for Edge Coloring **[ESA'19]**
Behnezhad, Derakhshan, Knittel, Hajiaghayi, **me**

The **Edge Coloring** problem

- Given a graph $G = (V, E)$, a **valid** "edge-coloring" is a function $\mathbf{COL}: E \to [\Psi]$ so that no two **incident edges** have a common color.

- We wish to minimize Ψ, i.e., the number of colors.

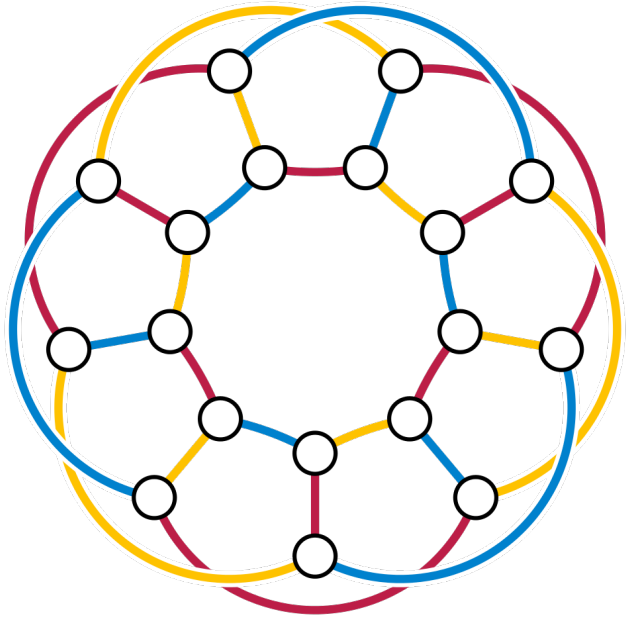


The **Edge Coloring** problem

- Given a graph $G = (V, E)$, a **valid** "edge-coloring" is a function $\mathbf{COL}: E \to [\Psi]$ so that no two **incident edges** have a common color.
- Let Δ be the maximum degree of graph G.
- Then, we know $\Delta \leq \Psi \leq \Delta + 1$.

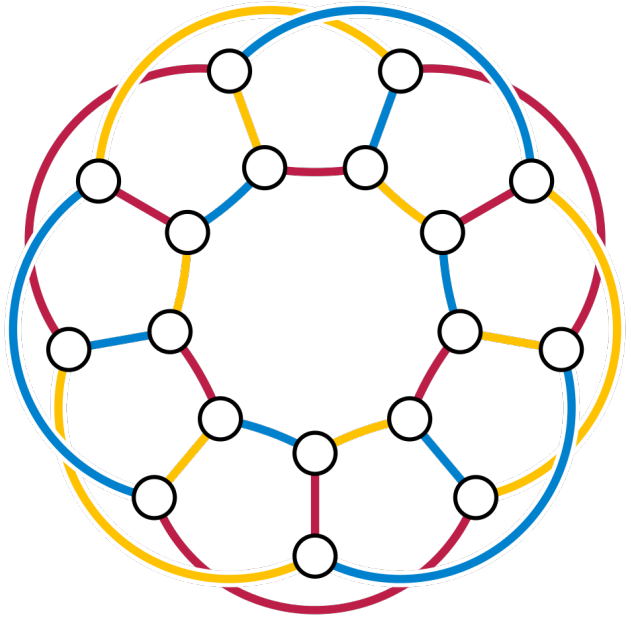
[Vizing 1964]

- An odd cycle ($\Delta = 2$) needs 3 colors.



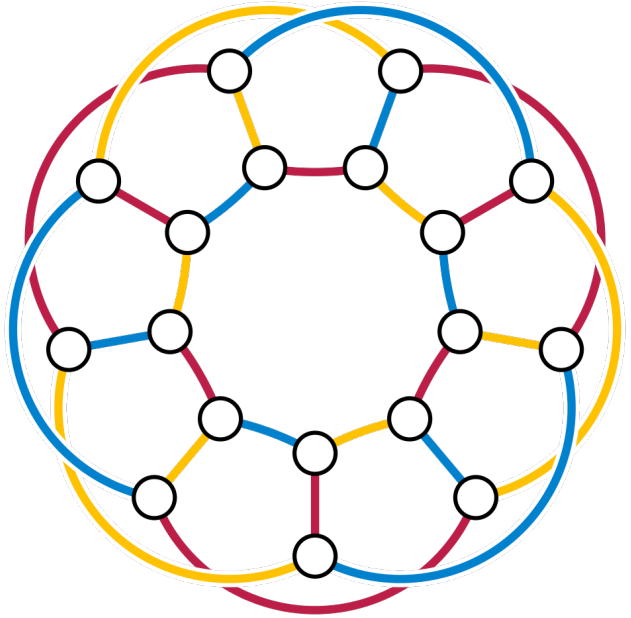
The **Edge Coloring** problem

- Given a graph $G = (V, E)$, a **valid** "edge-coloring" is a function $\mathbf{COL}: E \to [\Psi]$ so that no two **incident edges** have a common color.
- However, $(\Delta + 1)$-coloring algorithms are highly **sequential**.
- There is a greedy $(2\Delta - 1)$-coloring algorithm.



The **Edge Coloring** problem

- Given a graph $G = (V, E)$, a **valid** "edge-coloring" is a function $\mathbf{COL}: E \to [\Psi]$ so that no two **incident edges** have a common color.
- There is a greedy $(2\Delta - 1)$-coloring algorithm.
  - Process edges in an arbitrary order.
  - Color each edge with an available color.



Related work

- The closely related **Vertex** Coloring problem has constant-round MPC and one-pass Streaming algorithms both with $S = \tilde{O}(|V|)$ and $(\Delta + 1)$-colors used.

  [Assadi et. al 2019]

- Unfortunately **not applicable** to Edge Coloring.

- Differences:
  - Similar **greedy** algorithm gives a $(\Delta + 1)$-vertex-coloring.
  - We can store the **output** inside the machines.

Related work

- The only previously known **MPC** algorithm for **Edge** Coloring needs **superlinear** space per machine to achieve a $(\Delta + o(\Delta))$-coloring.

  [Harvey et. al 2018]

- Little to no result known for **Edge** Coloring in the **Streaming** model.

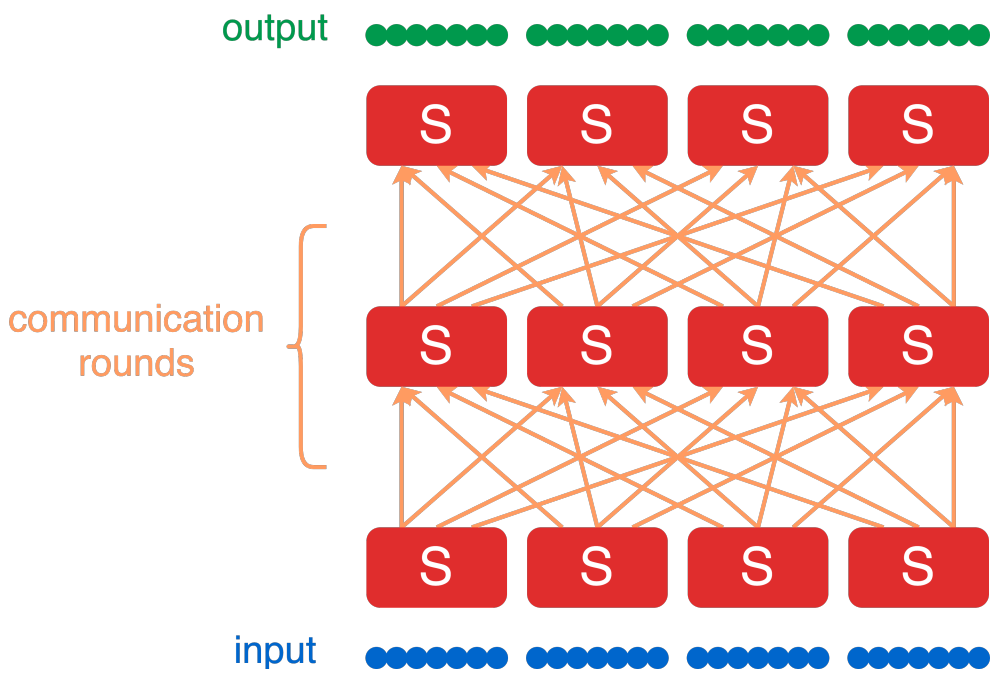
Edge Coloring

Massively **P**arallel **C**omputation

MPC Edge Coloring

- A graph $G = (V, E)$ is given where $|V| = n$ and $|E| = m$, i.e. $N = O(n + m)$.
- There is a **constant**-round MPC algorithm, with $S = O(n)$ and $S \cdot M = O(m)$, which computes an edge-coloring so that $\Psi = \Delta + \tilde{O}(\Delta^{3/4})$.

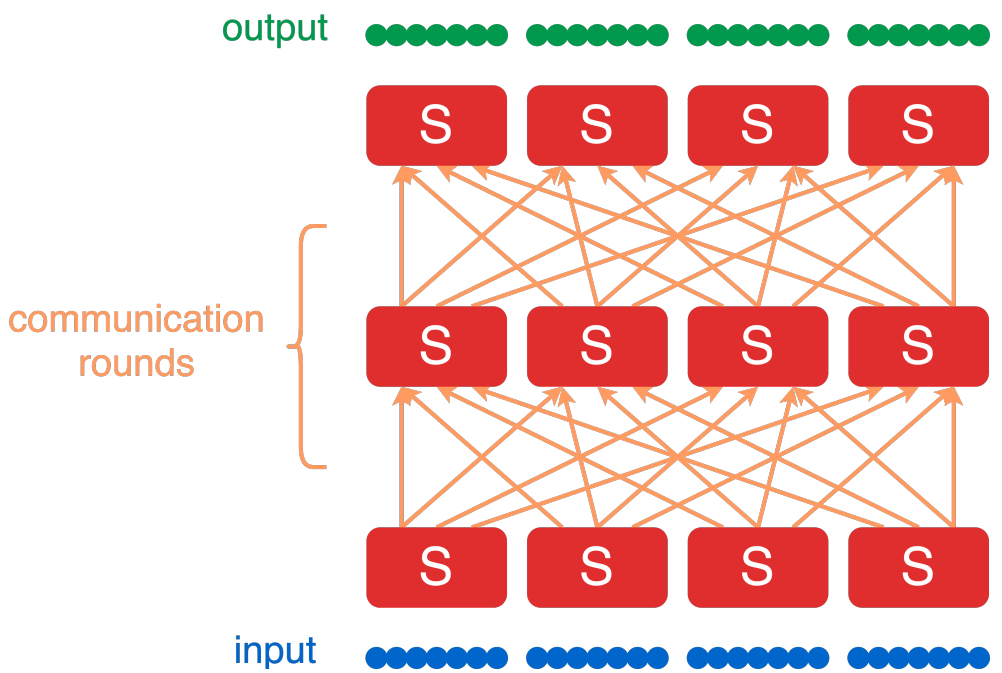
[BDKHS 2019]



Semi-MPC Edge Coloring

- A graph $G = (V, E)$ is given where $|V| = n$ and $|E| = m$, i.e. $N = O(n + m)$.
- There is a **constant**-round MPC algorithm, with $S = O(n)$ and $S \cdot M = O(m)$, which computes an edge-coloring so that $\Psi = \Delta + \tilde{O}(\Delta^{3/4})$.

  [BDKHS 2019]
- It is technically a **Semi-**MPC algorithm.

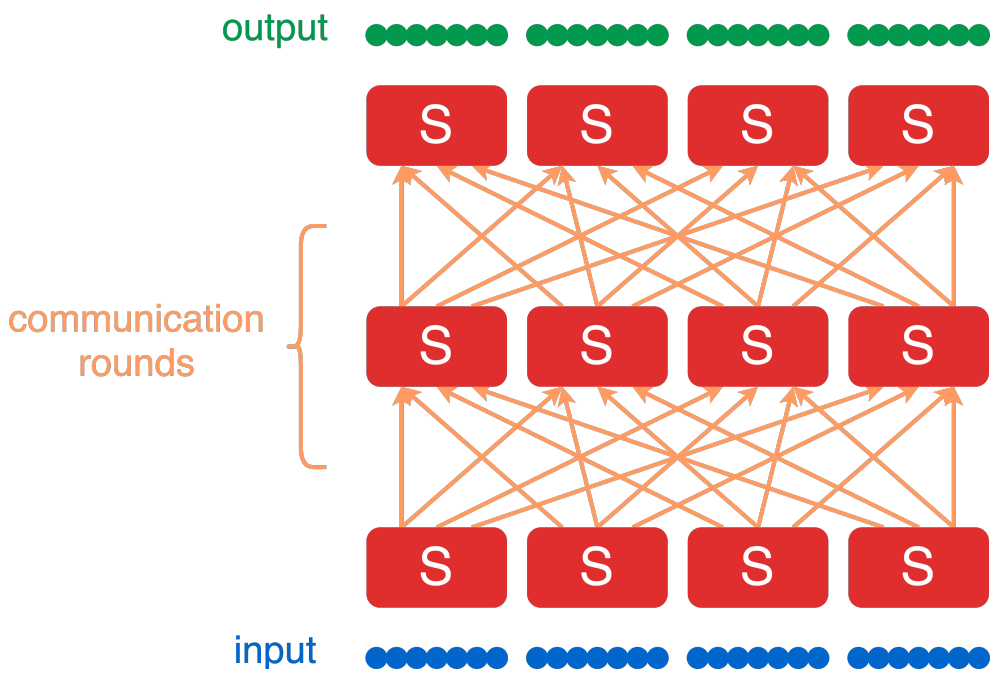


Semi-MPC Edge Coloring

- It is technically a **Semi-**MPC algorithm, but we can achieve $o(n)$ space in dense graphs.
- The exact space-per-machine is equal to

$$S = O\left(\frac{n\Delta}{k^2} + \frac{n}{k}\sqrt{\frac{\Delta}{k}\log n}\right)$$

- Set $k = \sqrt{\Delta} + \log n$.

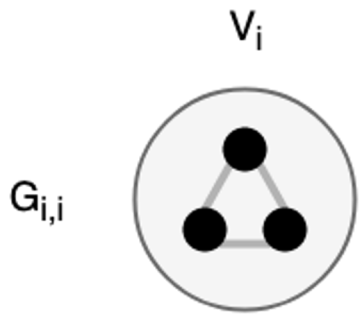


Vertex Partitioning

- Set $\boldsymbol{k} = \sqrt{\Delta} + \log n$.
- Random vertex partitioning: $V = V_1 \cup V_2 \cup \ldots \cup V_{\boldsymbol{k}}$.

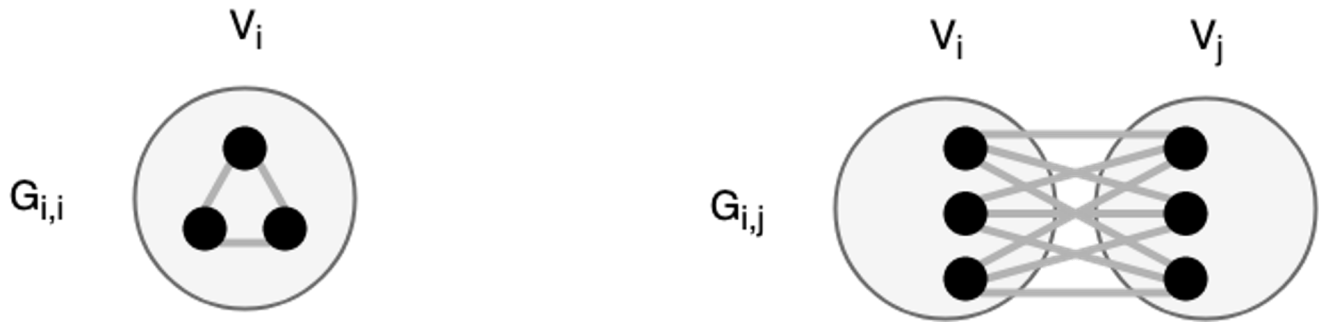
Vertex Partitioning

- Set $\boldsymbol{k} = \sqrt{\Delta} + \log n$.
- Random vertex partitioning : $V = V_1 \cup V_2 \cup \ldots \cup V_{\boldsymbol{k}}$.
- $G_{i,i} = \{ (u, v) \mid u \in V_i \ \wedge v \in V_i \}$: the **induced** subgraph of each partition



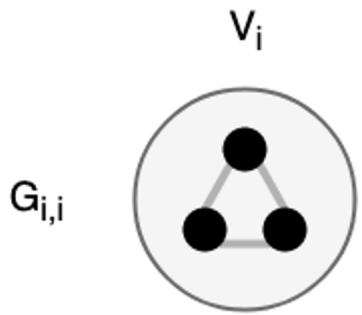
Vertex Partitioning

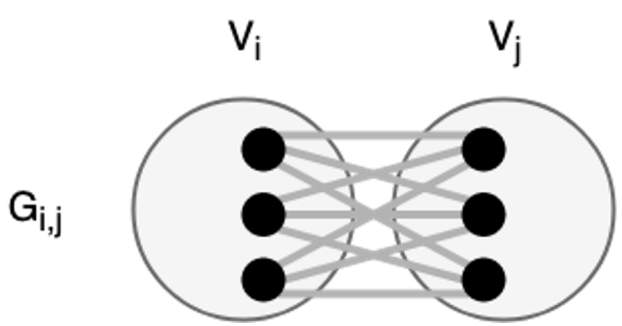
- Random vertex partitioning : $V = V_1 \cup V_2 \cup \ldots \cup V_k$.
- $G_{i,i} = \{ (u, v) \mid u \in V_i \wedge v \in V_i \}$: the **induced** subgraph of each partition
- $G_{i,j} = \{ (u, v) \mid u \in V_i \wedge v \in V_j \}$: the **bipartite induced** subgraph of pairs of partitions



Local Coloring

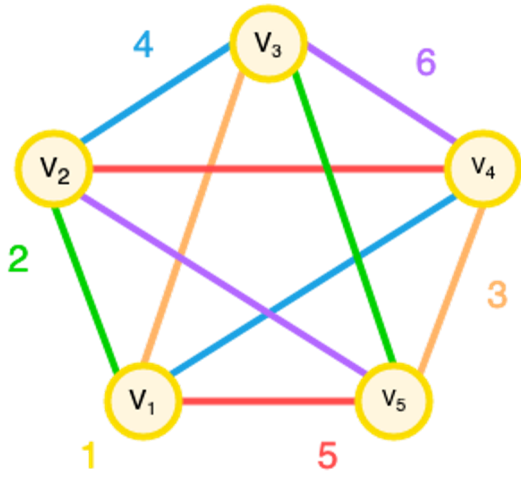
- $G_{i,i} = \{ (u, v) \mid u \in V_i \ \wedge v \in V_i \}$: the **induced** subgraph of each partition
- $G_{i,j} = \{ (u, v) \mid u \in V_i \ \wedge v \in V_j \}$: the **bipartite induced** subgraph of pairs of partitions
- Run Vizing's $(\Delta + 1)$-coloring algorithm on each machine.





Merging Colored Subgraphs

- $k+1$ disjoint color **palettes** are enough.
- The number of colors in each **palette** needs needs to be as large as the maximum degree in subgraphs.
- The maximum degree in each subgraphs is **concentrated**.
- For $\boldsymbol{k} = \sqrt{\Delta} + \log n$, we have $\boldsymbol{\Psi} = \Delta + \tilde{O}(\Delta^{3/4})$.



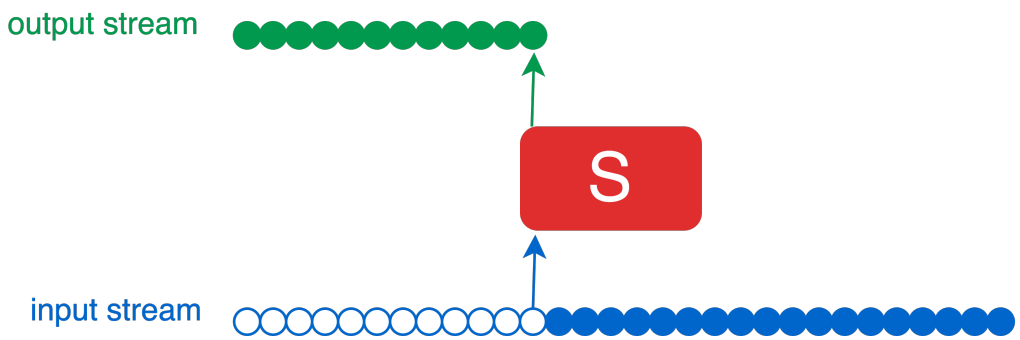
Edge Coloring

Random Streaming

Random Streaming Edge Coloring

- A graph $G = (V, E)$ is given where $|V| = n$ and $|E| = m$, i.e. $N = O(n + m)$.
- There is a **one**-pass **W**-streaming algorithm, with $S = O(n)$, which streams a valid edge-coloring so that $\Psi = (2e + \epsilon)\Delta$.

[BDKHS 2019]



Random Streaming Edge Coloring

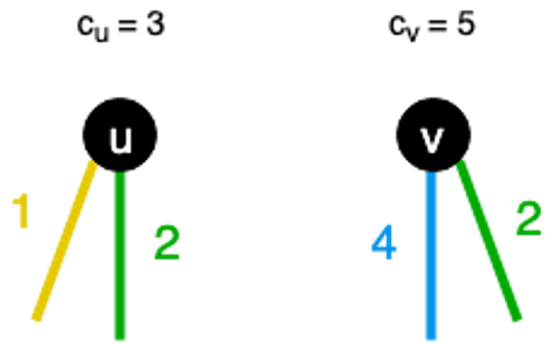
- A graph $G = (V, E)$ is given where $|V| = n$ and $|E| = m$, i.e. $N = O(n + m)$.
- There is a **one**-pass **W**-streaming algorithm, with $S = O(n)$, which streams a valid edge-coloring so that $\Psi = (2e + \epsilon)\Delta$.

[BDKHS 2019]

- The algorithm is straight-forward.
- Maintain a **counter** variable c_v for each vertex, initially set to 1.

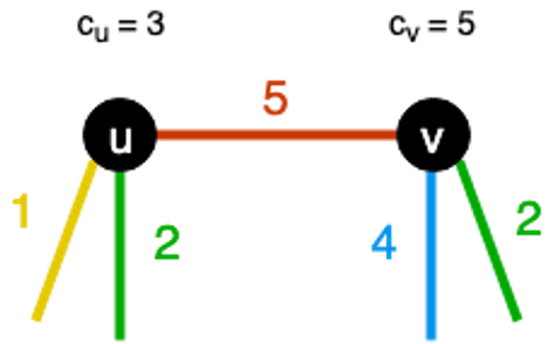
Random Streaming Edge Coloring

- Maintain a **counter** variable c_v for each vertex, initially set to 1.
- Upon arrival of (u, v) color it $\max(c_v, c_u)$.



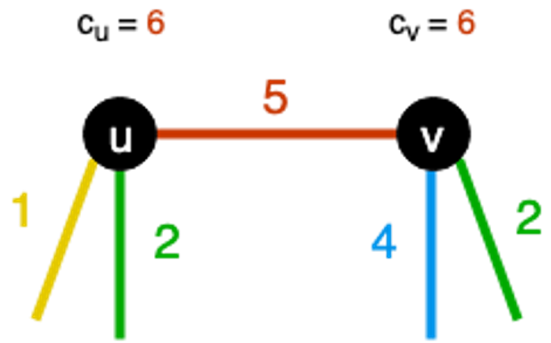
Random Streaming Edge Coloring

- Maintain a **counter** variable c_v for each vertex, initially set to 1.
- Upon arrival of (u, v) color it $\max(c_v, c_u)$.



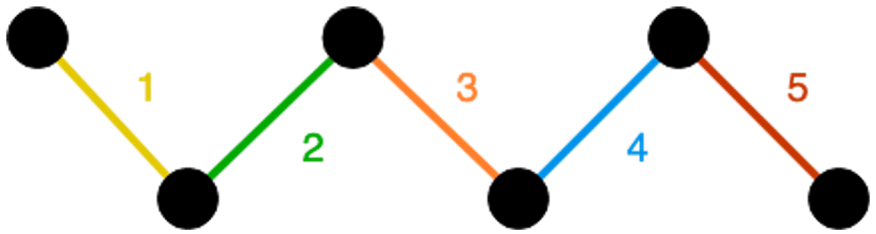
Random Streaming Edge Coloring

- Maintain a **counter** variable c_v for each vertex, initially set to $\mathbf{1}$.
- Upon arrival of (u, v) color it $\max(c_v, c_u)$.
- Set both c_v and c_u to $\max(c_v, c_u) + 1$.



Longest **Monotone** Path

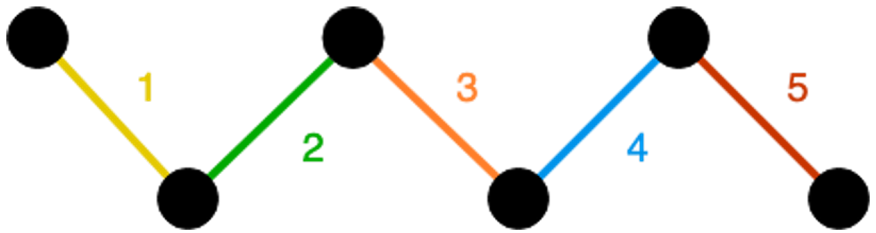
- **Lemma:** Ψ is equal to the length of the longest **monotone** path in the **line graph** after the algorithm finishes.
- We can construct a **monotone** path of length Ψ starting from an edge colored Ψ.



Longest **Monotone** Path

- **Lemma:** Ψ is equal to the length of the longest **monotone** path in the **line graph** after the algorithm finishes.

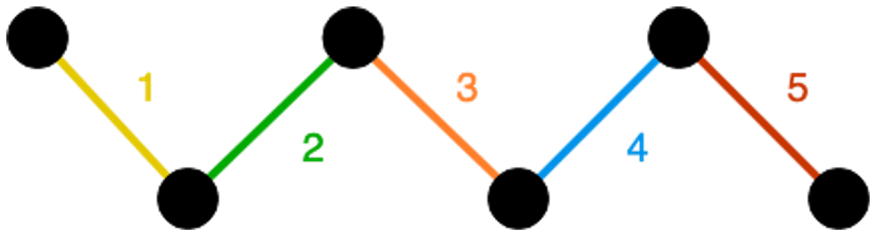
$$\Pr[\Psi \geq \alpha\Delta] \leq \frac{(2\Delta)^{\alpha\Delta}}{(\alpha\Delta)!}$$



Longest **Monotone** Path

- **Lemma:** Ψ is equal to the length of the longest **monotone** path in the **line graph** after the algorithm finishes.

$$\Pr[\Psi \geq (2e + \epsilon)\Delta] \leq n^{-c}$$



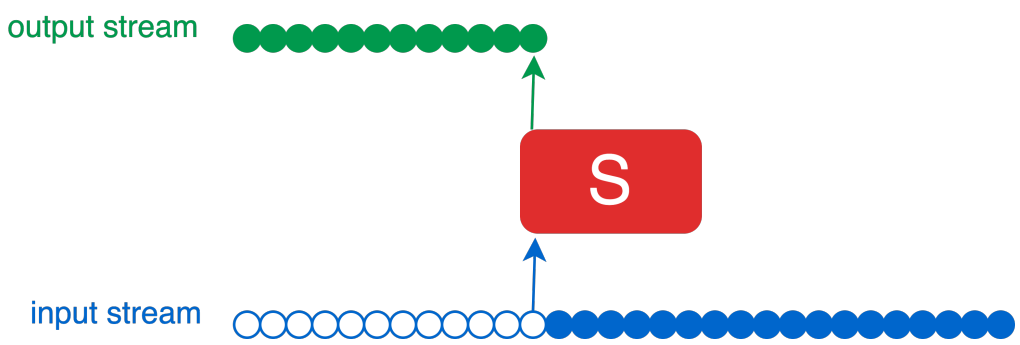
Edge Coloring

Adversarial Streaming

Adversarial Streaming Edge Coloring

- A graph $G = (V, E)$ is given where $|V| = n$ and $|E| = m$, i.e. $N = O(n + m)$.
- There is a **one**-pass **W**-streaming algorithm, with $S = O(n)$, which streams a valid edge-coloring so that $\Psi = O(\Delta^2)$.

[BDKHS 2019]



Adversarial Streaming Edge Coloring

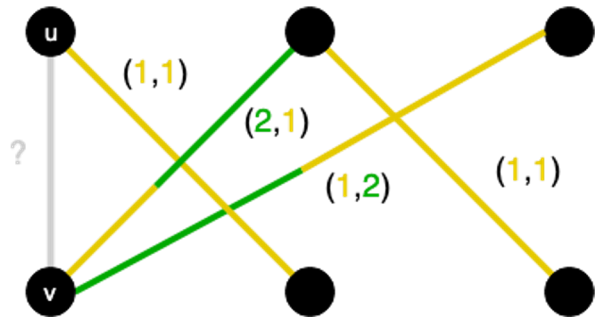
- A graph $G = (V, E)$ is given where $|V| = n$ and $|E| = m$, i.e. $N = O(n + m)$.
- There is a **one**-pass **W**-streaming algorithm, with $S = O(n)$, which streams a valid edge-coloring so that $\Psi = O(\Delta^2)$.

[BDKHS 2019]

- The algorithm is very similar to the random stream algorithm.
- Maintain a **counter** variable c_v for each vertex, initially set to 1.

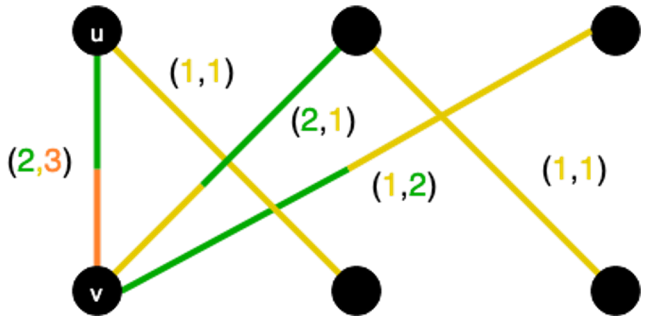
Bipartite Edge Coloring

- Maintain a **counter** variable c_v for each vertex, initially set to $\mathbf{1}$. Also assume the graph is **bipartite**.
- Upon arrival of (u, v) color it (c_u, c_v).



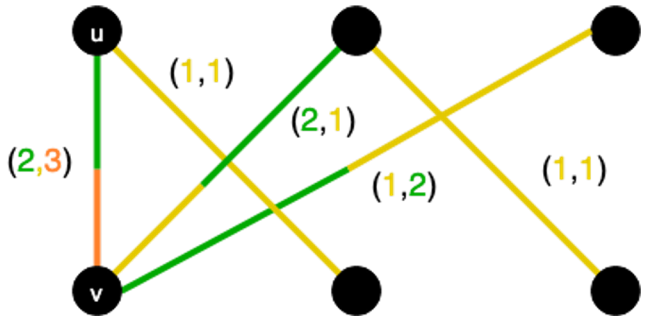
Bipartite Edge Coloring

- Maintain a **counter** variable c_v for each vertex, initially set to 1. Also assume the graph is **bipartite**.
- Upon arrival of (u, v) color it (c_u, c_v).



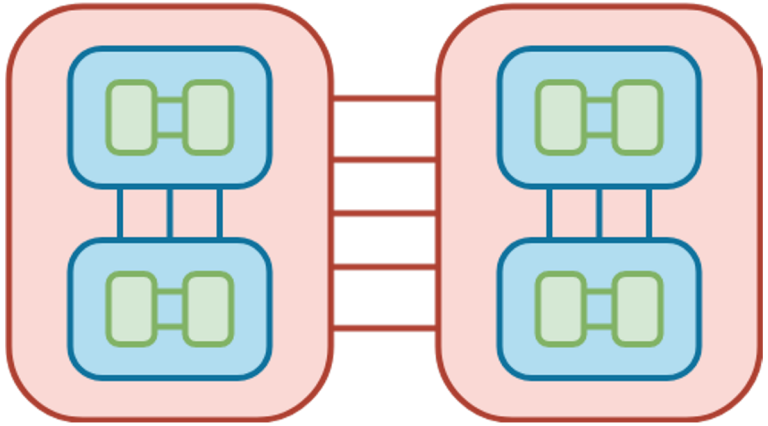
Bipartite Edge Coloring

- Maintain a **counter** variable c_v for each vertex, initially set to 1. Also assume the graph is **bipartite**.
- Upon arrival of (u, v) color it (c_u, c_v).
- Increase both c_u and c_v by 1.



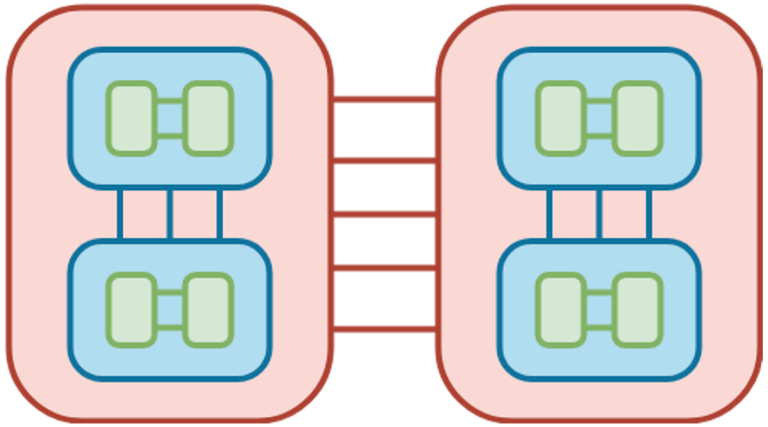
General Edge Coloring

- A **bipartite** graph is colored with Δ^2 colors using this method.
- Decompose the graph into $O(\log n)$ random bipartite graphs, and color each with a different **palette**.



General Edge Coloring

- A **bipartite** graph is colored with Δ^2 colors using this method.
- Decompose the graph into $O(\log n)$ random bipartite graphs, and color each with a different **palette**.
- In total, $\Psi = O(\Delta^2)$.



Thanks for watching!

Any questions?