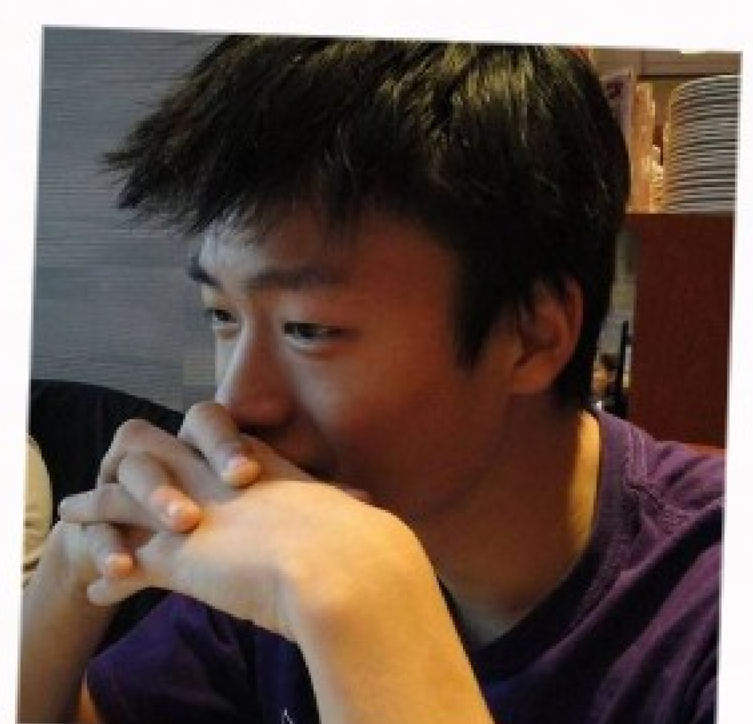
Improved Algorithms for Edge Colouring in the W-Streaming Model

Paul Liu



Moses Charikar

Edge - Colouring

Edge - Colouring

- An assignment of colours to the edges of a graph

Edge - Colouring

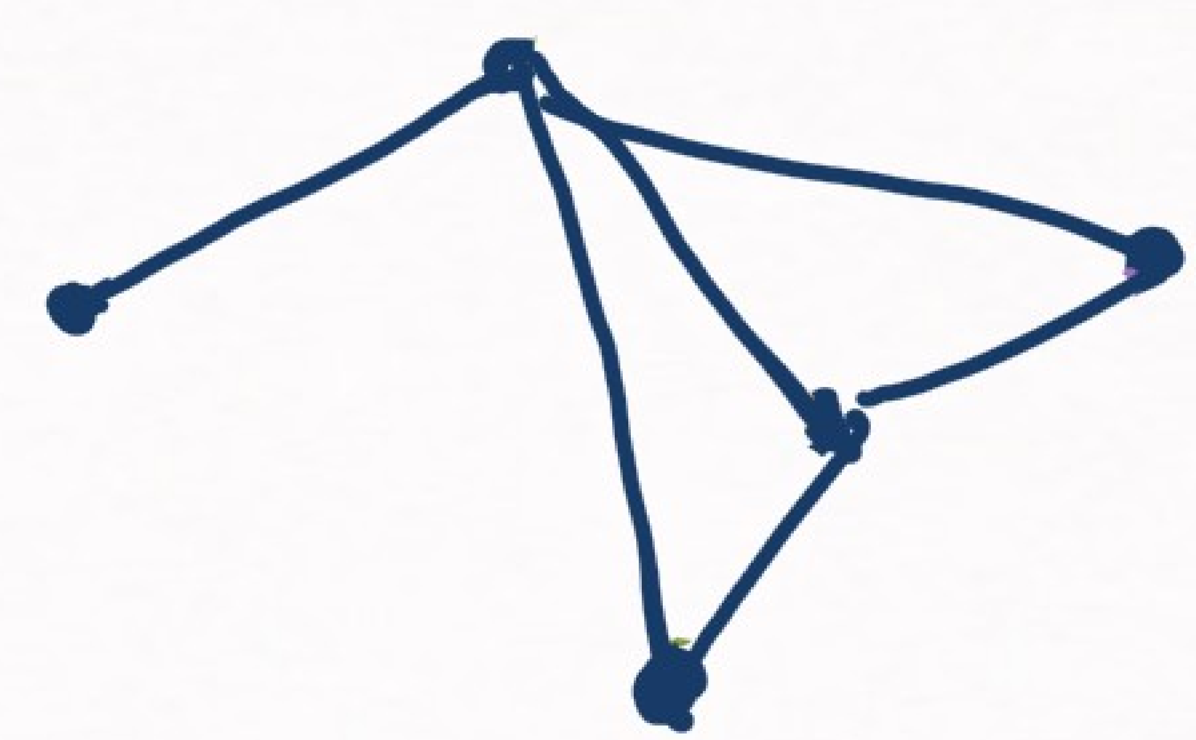
- An assignment of colours to the edges of a graph
- No two adjacent edges can have the same colour

Edge-Colouring

- An assignment of colours to the edges of a graph
- No two adjacent edges can have the same colour
- Goal: <u>Minimize</u> # distinct colours used

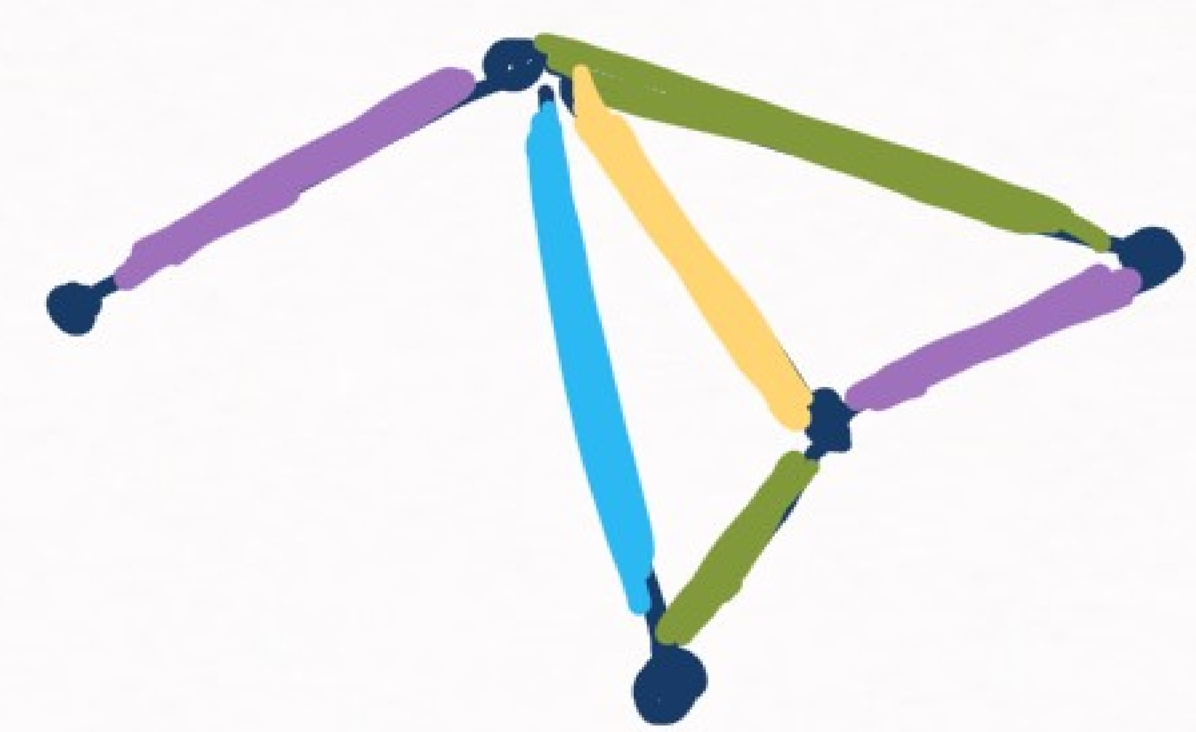
Edge-Colouring

- An assignment of colours to the edges of a graph
- No two adjacent edges can have the same colour
- Goal: <u>Minimize</u> # distinct colours used



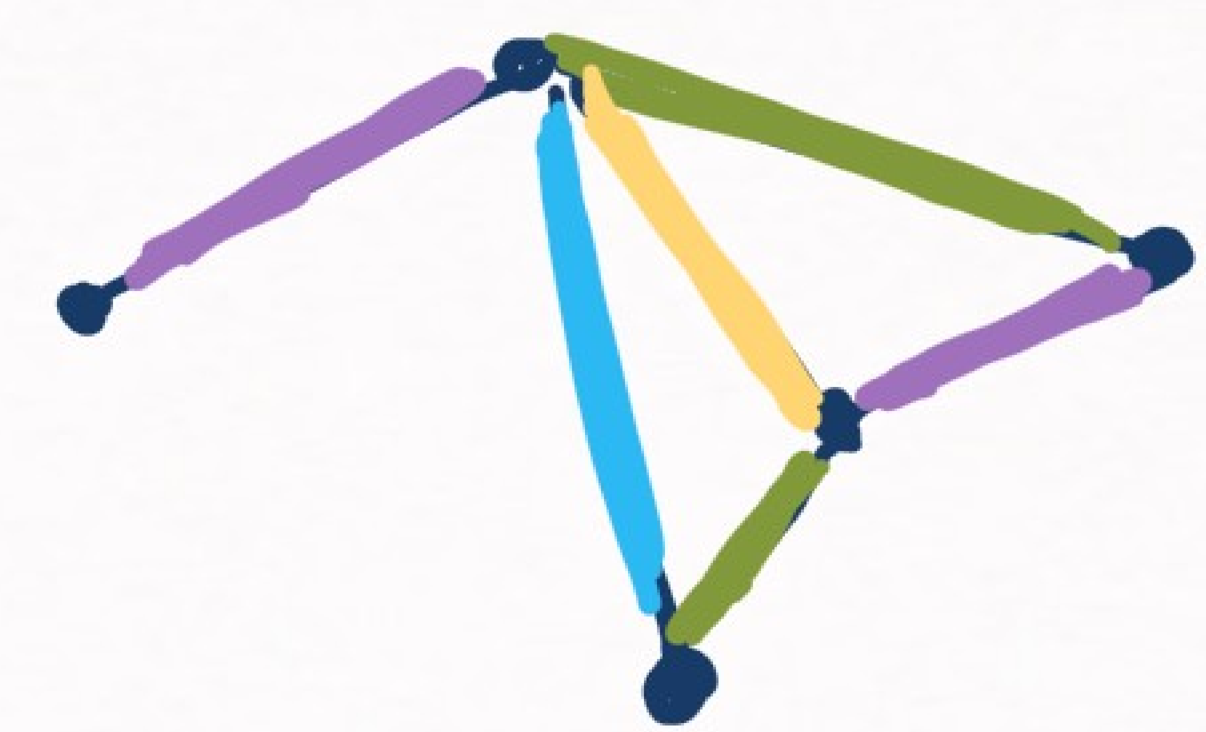
Edge - Colouring

- An assignment of colours to the edges of a graph
- No two adjacent edges can have the same colour
- Goal: **Minimize** # distinct colours used



Edge - Colouring

- An assignment of colours to the edges of a graph
- No two adjacent edges can have the same colour
- Goal: Minimize # distinct colours used
- $\Delta+1$ always possible! (Vizing) (Δ = max deg.)
- Δ required; NP-Complete to determine Δ vs. $\Delta+1$



Edge-colouring in W-streaming

Edge-colouring in W-streaming

- Up to $O(n^2)$ edges streamed in
  #vertices $= n$
- $O(n \operatorname{polylog} n)$ memory for alg.
  (can assume graph is dense)

Edge-colouring in W-Streaming

- Up to $O(n^2)$ edges streamed in
  # vertices $= n$
- $\tilde{O}(n \operatorname{polylog} n)$ memory
  (can assume graph is dense)
- A colour for every edge must be output by the time the stream completes

Edge-colouring in W-Streaming

- Up to $O(n^2)$ edges streamed in
- $\tilde{O}(n \operatorname{polylog} n)$ memory
- A colour for every edge must be output by the time the stream completes

Edge-colouring in W-Streaming

- Up to $O(n^2)$ edges streamed in
- $O(n \operatorname{polylog} n)$ memory
- A colour for every edge must be output by the time the stream completes
- Memory allows for limited "buffering" of output

Edge-colouring in W-Streaming

- Up to $O(n^2)$ edges streamed in
- $\tilde{O}(n \operatorname{polylog} n)$ memory
- A colour for every edge must be output by the time the stream completes
- Memory allows for limited "buffering" of output

Related Work: Online Model

- Must output a colour for each edge immediately
- Allowed poly(n) memory (usually O(E))

Related Work: Online Model

- Must output a colour for each edge immediately
- Allowed poly(n) memory (usually O(E))
- Two regimes: vertex arrival (all edges adj. to a vertex arrive at once)

  edge-arrival (edges arrive one at a time)

our regime

Related Work: Online Model

- When $\Delta = \omega(\log n)$, $(1+o(1))\Delta$ is possible:
  - Adversarial vertex arrivals [Cohen, Peng, Wajc '19]
  - Random edge arrivals [Bhattacharya, Grandoni, Wajc '20]
    SODA 21

Related Work: Online Model

- When $\Delta = \omega(\log n)$, $(1+o(1))\Delta$ is possible:
  - Adversarial vertex arrivals [Cohen, Peng, Wajc '19]
  - Random edge arrivals [Bhattacharya, Grandoni, Wajc '20] SODA 21
  - All of these algs use too much memory for W-streaming

Progress in W-streaming

Progress in W-streaming

	Random Edge Arrival	Adversarial Edge Arrival
[Behnezhad et al. '19]	- $2e(1+o(1))\Delta$ colours	- $O(\Delta^2)$ colouring

Progress in W-streaming

	Random Edge Arrival	Adversarial Edge Arrival
[Behnezhad et al. '19]	– $2e(1+o(1))\Delta$ colours	– $O(\Delta^2)$ colouring
[Our Results]	– $(1+o(1))\Delta$ colours (uploaded talk; ★animated★ slides!)	– $(1+o(1))\frac{\Delta^2}{s}$ colours * in $\tilde{O}(ns)$ space (all other algs use $\tilde{O}(n)$ space) Today's talk

A simple algorithm for adversarial Arrivals

A simple algorithm for adversarial Arrivals

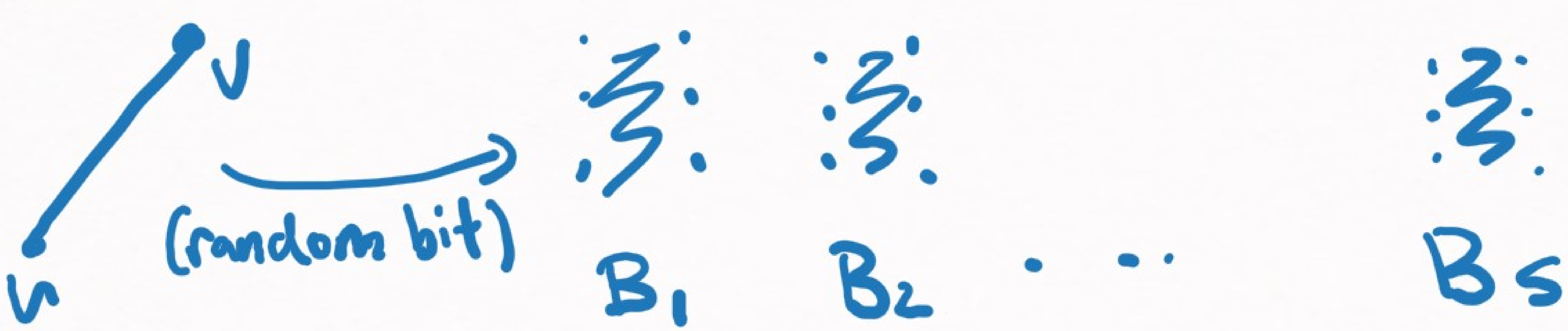
- For each node u, generate $\underline{s := 36 \log n}$ random bits
- Initialize empty graphs $B_1, B_2, \ldots, B_s$

A simple algorithm for adversarial Arrivals

- For each node u, generate $\underline{s := 36 \log n}$ random bits
- Initialize empty graphs $B_1, B_2, \ldots, B_s$
- For each edge (u,v), pick a random bit where the bit strings of u & v differ. Let i be the index of this bit. Add (u,v) to B_i.

A simple algorithm for adversarial Arrivals

- For each node u, generate $\underline{s := 36 \log n}$ random bits
- Initialize empty graphs $B_1, B_2, \ldots, B_s$
- For each edge (u,v), pick a random bit where the bit strings of u & v differ. Let i be the index of this bit. Add (u,v) to B_i.



A simple algorithm for adversarial Arrivals

- Each of the B_i's are <u>bipartite</u>

  (left nodes: i^{th} bit 0, right nodes: i^{th} bit 1)

A simple algorithm for adversarial Arrivals

- Each of the B_i's are <u>bipartite</u>
  (left nodes: i^{th} bit 0, right nodes: i^{th} bit 1)
- Any index equally likely to be chosen, so max deg $B_i \simeq \frac{\Delta}{S}$
  (concentration achieved when $\Delta \simeq \omega(\log n)$)

A simple algorithm for adversarial Arrivals

- Each of the B_i's are <u>bipartite</u>
  (left nodes: i^{th} bit 0, right nodes: i^{th} bit 1)
- Any index equally likely to be chosen, so max deg $B_i \simeq \frac{\Delta}{S}$
  (concentration achieved when $\Delta \simeq \omega(\log n)$)
- Choice of $36 \log n$ ensures at least one mismatched bit

Colouring a bipartite graph

Colouring a bipartite graph

- For each node $u \in B$, initialize a counter $C_u = 0$.

Colouring a bipartite graph

- For each node $u \in B$, initialize a counter $C_u = 0$.
- When an edge (u, v) is streamed in, output colour (C_u, C_v) and increment C_u & C_v by 1.

Colouring a bipartite graph

- For each node $u \in B$, initialize a counter $C_u = 0$.

left node — right node

- When an edge (u, v) is streamed in, output colour (C_u, C_v) and increment C_u & C_v by 1.

Colouring a bipartite graph

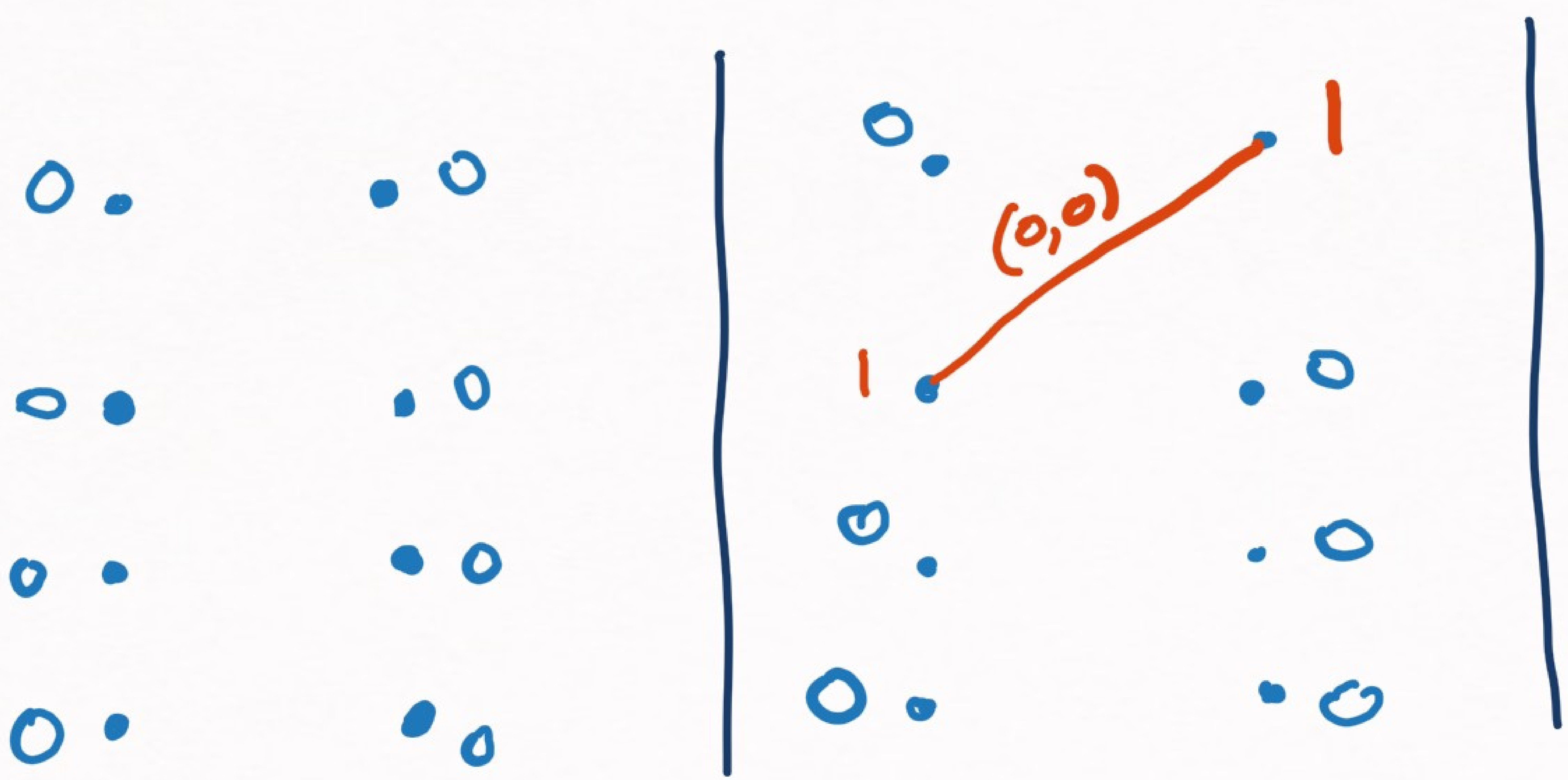
- For each node $u \in B$, initialize a counter $C_u = 0$.
- When an edge $(\overset{\text{left node}}{u}, \overset{\text{right node}}{v})$ is streamed in, output colour (C_u, C_v) and <u>increment</u> C_u & C_v by 1.

Colouring a bipartite graph

- For each node $u \in B$, initialize a counter $C_u = 0$.

left node, right node

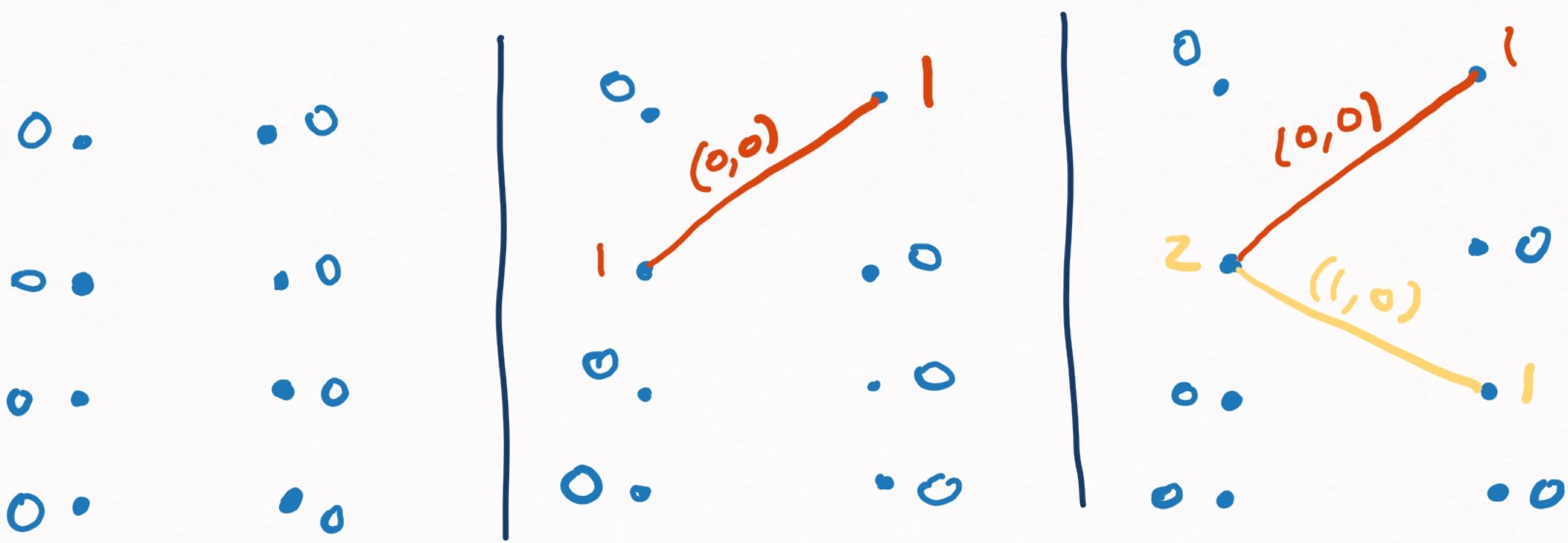
- When an edge (u, v) is streamed in, output colour (C_u, C_v) and increment C_u & C_v by 1.



Colouring a bipartite graph

- For each node $u \in B$, initialize a counter $C_u = 0$.
- When an edge (u, v) is streamed in, output colour (C_u, C_v) and <u>increment</u> C_u & C_v by 1.

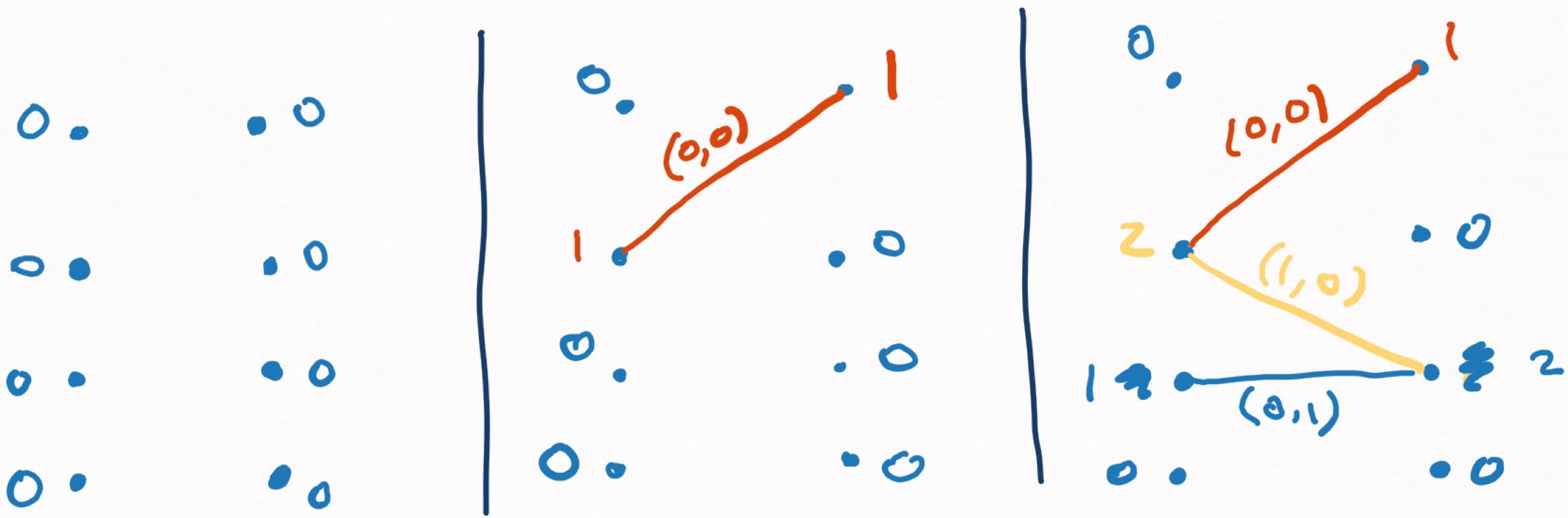
left node → u, right node → v



Colouring a bipartite graph

- For each node $u \in B$, initialize a counter $C_u = 0$.
- When an edge (u, v) is streamed in, output colour (C_u, C_v) and <u>increment</u> C_u & C_v by 1.

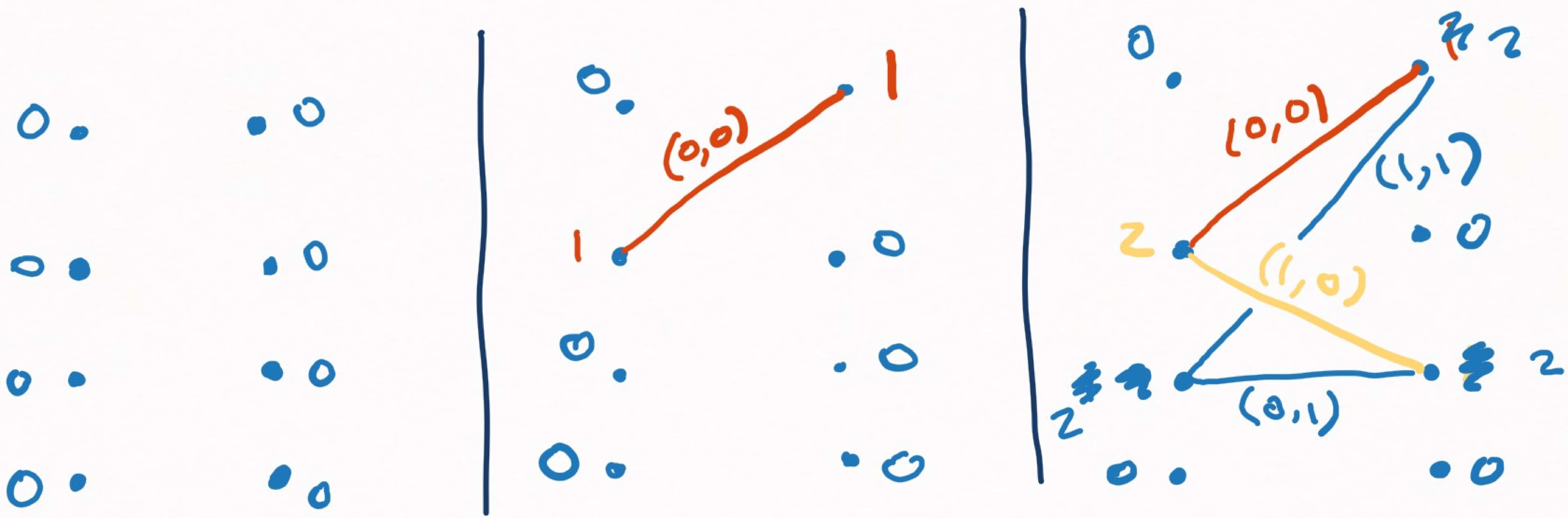
left node / right node



Colouring a bipartite graph

- For each node $u \in B$, initialize a counter $C_u = 0$.
- When an edge (u, v) is streamed in, output colour (C_u, C_v) and increment C_u & C_v by 1.

left node — right node



Colouring a bipartite graph

- Worst case: each counter goes up to max deg B_i ($\simeq \Delta/s$)

  $\hookrightarrow$ (c_u, c_v) has $\left(\frac{\Delta}{s}\right)^2$ possibilities.

- Achievable:

Works for any $p, q \leq \max \deg B_i \simeq \frac{\Delta}{s}$.

Colouring a bipartite graph

- Worst case: each counter goes up to max deg B_i
  ($\simeq \Delta / s$)

  ↳ (c_u, c_v) has $\left(\frac{\Delta}{s}\right)^2$ possibilities.

- Achievable:

Works for any
$p, q \le \max \deg B_i \simeq \frac{\Delta}{s}$.

Wrapping Up

$$\text{Total \# colours} : \quad \underbrace{S}_{\text{\# of } B_i} \cdot \Big(\underbrace{(1+o(1))\frac{\Delta}{S}}_{\text{max deg per } B_i} \Big)^{2}$$

colours used by alg.

$$= \frac{\Delta^2}{S}\,(1+o(1)).$$

Wrapping Up

colours used by alg.

$$\text{Total \# colours} : \underbrace{s}_{\#\text{ of } B_i} \cdot \Big(\underbrace{(1+o(1))\frac{\Delta}{s}}_{\text{max deg per } B_i} \Big)^2$$

$$= \frac{\Delta^2}{s}(1+o(1)).$$

(Worst case can be achieved by streaming poly(Δs) copies of star-shaped graphs


)

Open problem: Is there a $O(\Delta)$ colouring algorithm for **adversarial orders** in W-streaming?

<u>Open problem</u>: Is there a $O(\Delta)$ colouring algorithm for <u>adversarial orders</u> in W-streaming?

THANK YOU!