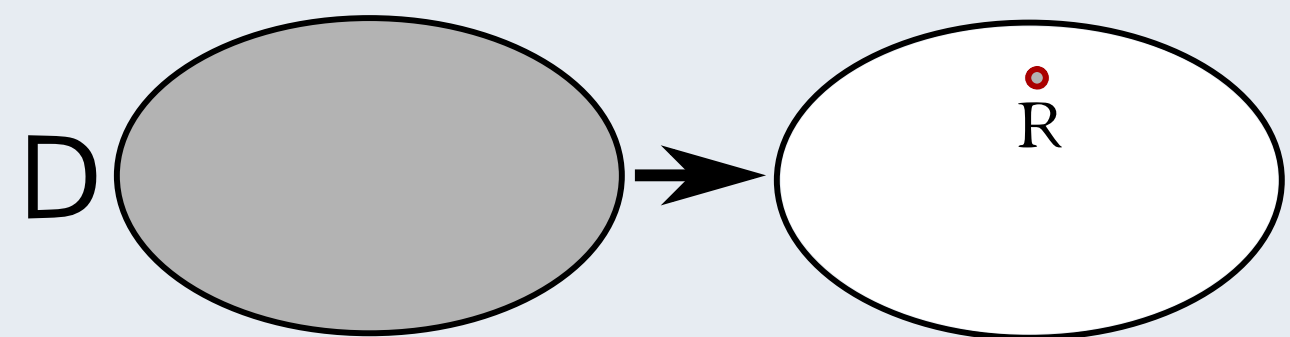


# Local-Access Generators

Amartya Shankha Biswas, Ronitt Rubinfeld, Anak Yodpinyanee  
CSAIL, MIT

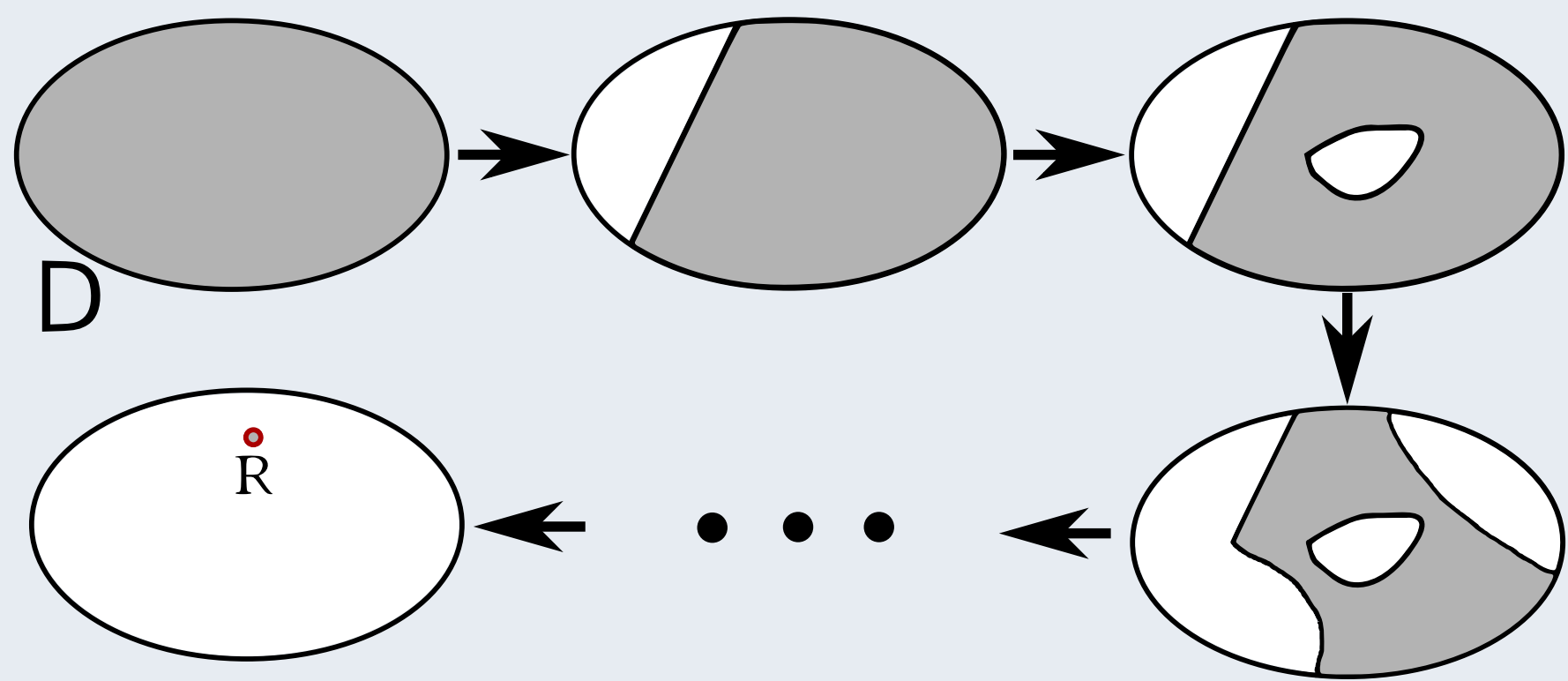
## Partial Sampling from a Distribution

Full Sampling  $R \sim D$  in  $\mathcal{O}(T)$  time



Do we need to spend  $\mathcal{O}(T)$  upfront?

**Partial sampling:** each step should take  $\tilde{\mathcal{O}}(T/N)$  time.



### Problem Statement

A local-access generator of a random object  $R \sim D$ , provides indirect access to  $R'$  with a *query oracle* s.t.

- All query responses (*partial samples*) are **consistent**
- The **distribution** of  $R'$  is  $\epsilon$ -close to  $D$  in  $L_1$  distance

## Sampling $G(n, p)$ : Vertex-Pair queries

**Vertex-Pair:** Given vertices  $u, v$ , decide whether  $(u, v) \in E$ .  
*Trivial:* just a collection of  $\binom{n}{2}$  Bernoulli RVs with bias  $p$ .

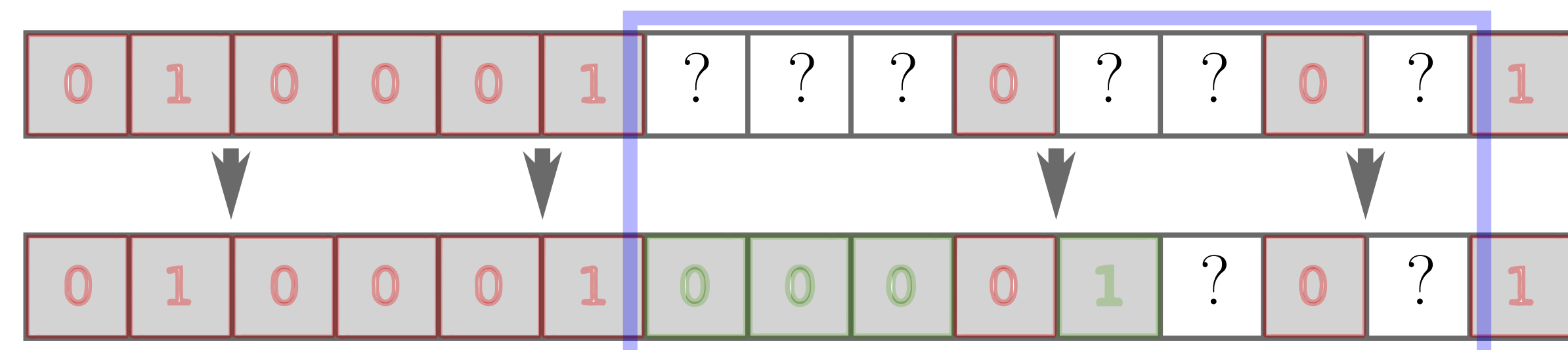
## Next-Neighbor queries (skip-sampling)

**Next-Neighbor:** Return neighbors of  $v$  in order.

*Issue:* Naïve sampling spends  $1/p$  time sampling the 0's  
*Idea:* can compute **Next-Neighbor**'s distribution's CDF from  
 $\mathbb{P}[k \text{ non-neighbors before next-neighbor}] = p(1-p)^k$

- **Skip-sampling** Draw from **Next-Neighbor** distribution
- Can sample from this distribution in  $\tilde{\mathcal{O}}(1)$  time [ELMR17]
- Further analysis required for finite-precision arithmetic

*Issue:* Adjacency matrix is symmetric; we need to **record all generated 0's** in the corresponding column of  $v$



*Issue:* if the sampled neighbor is already 0, must re-sample  
⇒ may hit 0's many times – **too many re-samplings**

## Random-Neighbor queries (Bucketing-Generator)

**Random-Neighbor:** Return a neighbor of  $v$  uniformly at random.

*Issue:* next-neighbor can't jump to a random potential neighbor of  $v$

**Bucketing** Divide each row of the adjacency matrix into contiguous buckets  
⇒ random neighbor of  $v \approx$  random neighbor in a random bucket of  $v$

*Issue:* do **not** know  $\deg(v)$ ; must return each neighbor with probability  $1/\deg(v)$

**Rejection Sampling** Return **any** neighbor with the **same** probability

#neighbors in each bucket  
 $\sim \Theta(1)$  in expectation,  $\mathcal{O}(\log n)$  max w.h.p. ⇒ #buckets  $\sim$  #neighbors

$v$ : 

0	1
---	---

0	0	1
---	---	---

 ... 

1	0
---	---

 ...

**Algorithm**

- Step 1** pick a uniform random bucket  
"fill" this bucket, if needed
- Step 2** pick a uniform random neighbor  $u$
- Step 3** return  $u$  with probability  $\frac{\text{\#neighbors in bucket}}{\mathcal{O}(\log n)}$   
otherwise, try again

return or reject

$$\mathbb{P}[\text{return } u] = \frac{1}{\text{\#buckets}} \times \frac{1}{\text{\#neighbors in bucket}} \times \frac{\text{\#neighbors in bucket}}{\mathcal{O}(\log n)} \approx \frac{\Omega(1/\log n)}{\text{\#neighbors of } v}$$

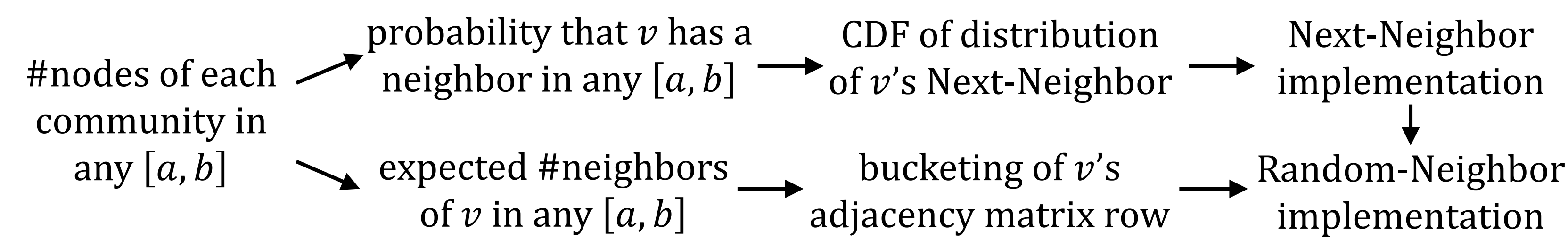
$$\mathbb{P}[\text{return any neighbor}] \approx \Omega(1/\log n) \Rightarrow \mathcal{O}(\log n) \text{ iterations suffice}$$

**Data Structure** bucket maintains its known neighbors and a **filled** marker  
"fill" with expected  $\mathcal{O}(1)$  **Next-Neighbor** queries }  $\mathcal{O}(\log n)$  expected time  
Random-Neighbor succeeds in  $\mathcal{O}(\log n)$  tries }  $\tilde{\mathcal{O}}(n + m)$  total space usage

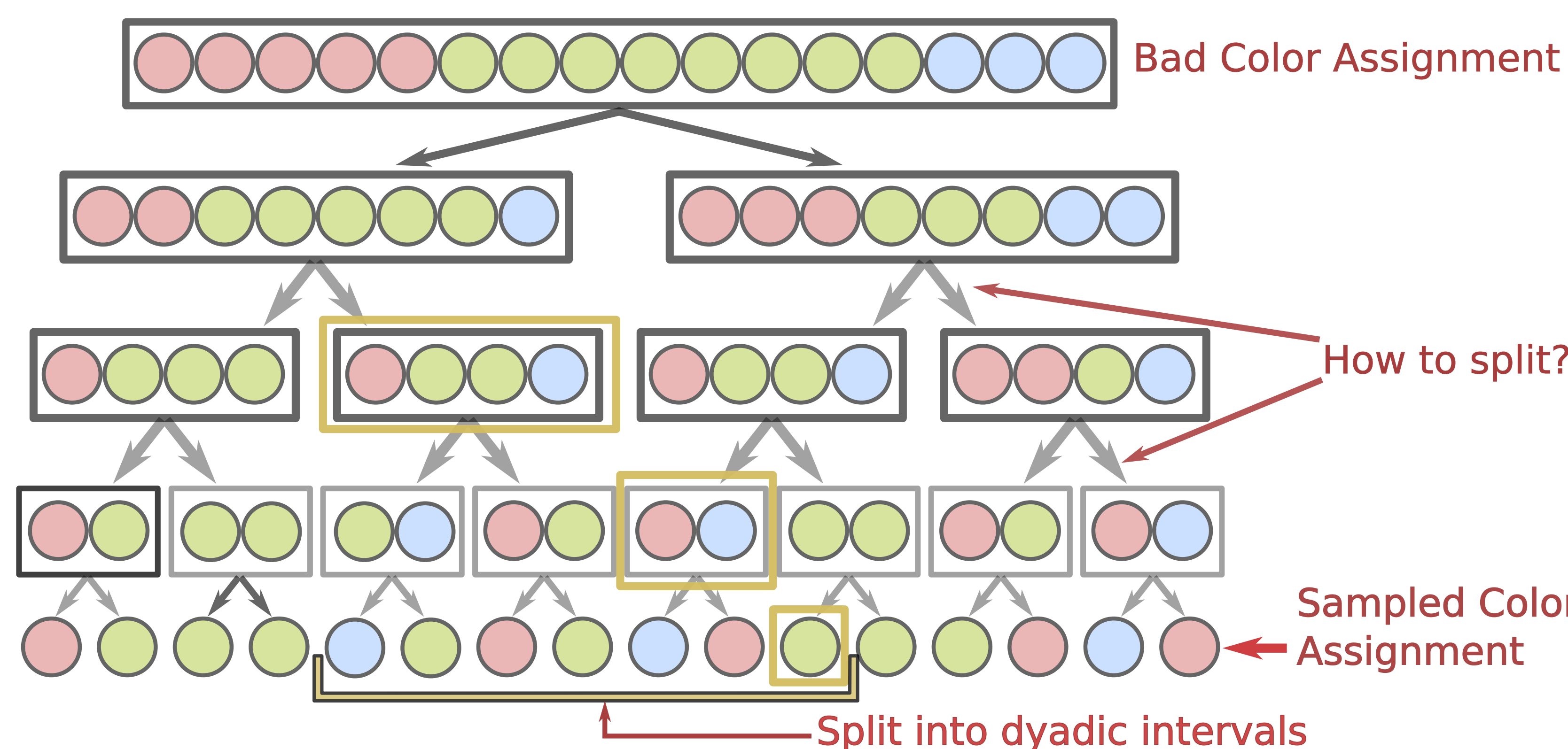
## Stochastic Block Model (Counting-Generator)

**Model:** Each  $v$  is assigned to a *random* community (color) from  $C_1, \dots, C_r$ .  
⇒ If  $u \in C_i, v \in C_j$ , then  $\mathbb{P}[(u, v) \in E] = p_{ij}$ . ( $|C_i|$ 's are given as input)

**Idea #1** use #nodes of each color in *any* contiguous range to can generate SBM



**Idea #2** implement a **Counting-Generator** to answer counting queries  
⇒ BBST, split on-the-fly with *Multivariate Hypergeometric Distribution*



## Multivariate Hypergeometric Distribution

[GGN10] solves the special case of  $r = 2$  and  $B = 2\ell$ .

COUNTING-GENERATOR

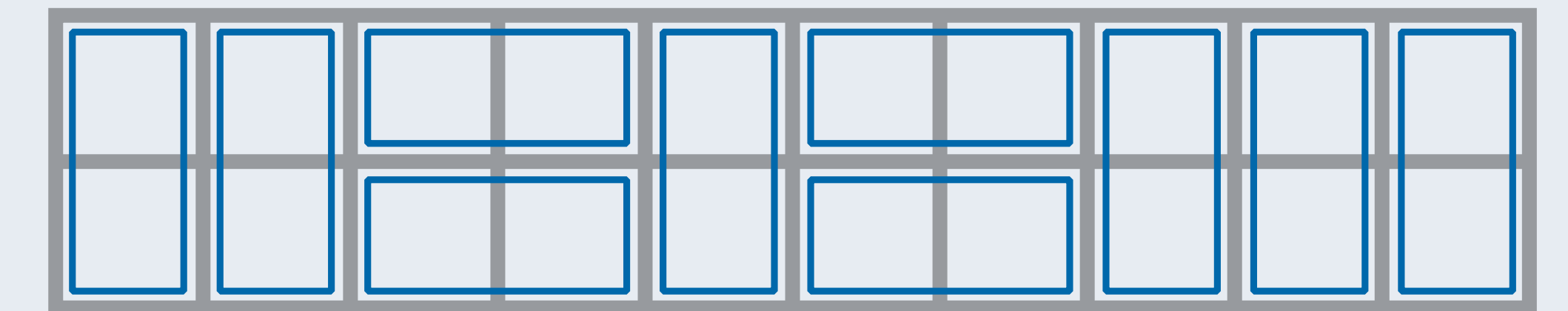
- **Extending to  $B \neq 2\ell$ :** Divide  $\ell$  into dyadic segments.
- **Extending to  $r > 2$ :** Make a tree with a leaf for each  $C_i$ . Every branch in the tree is equivalent to a 2-splitting

- Use COUNTING-GENERATOR to sample community counts
- Run the BUCKETING-GENERATOR as before

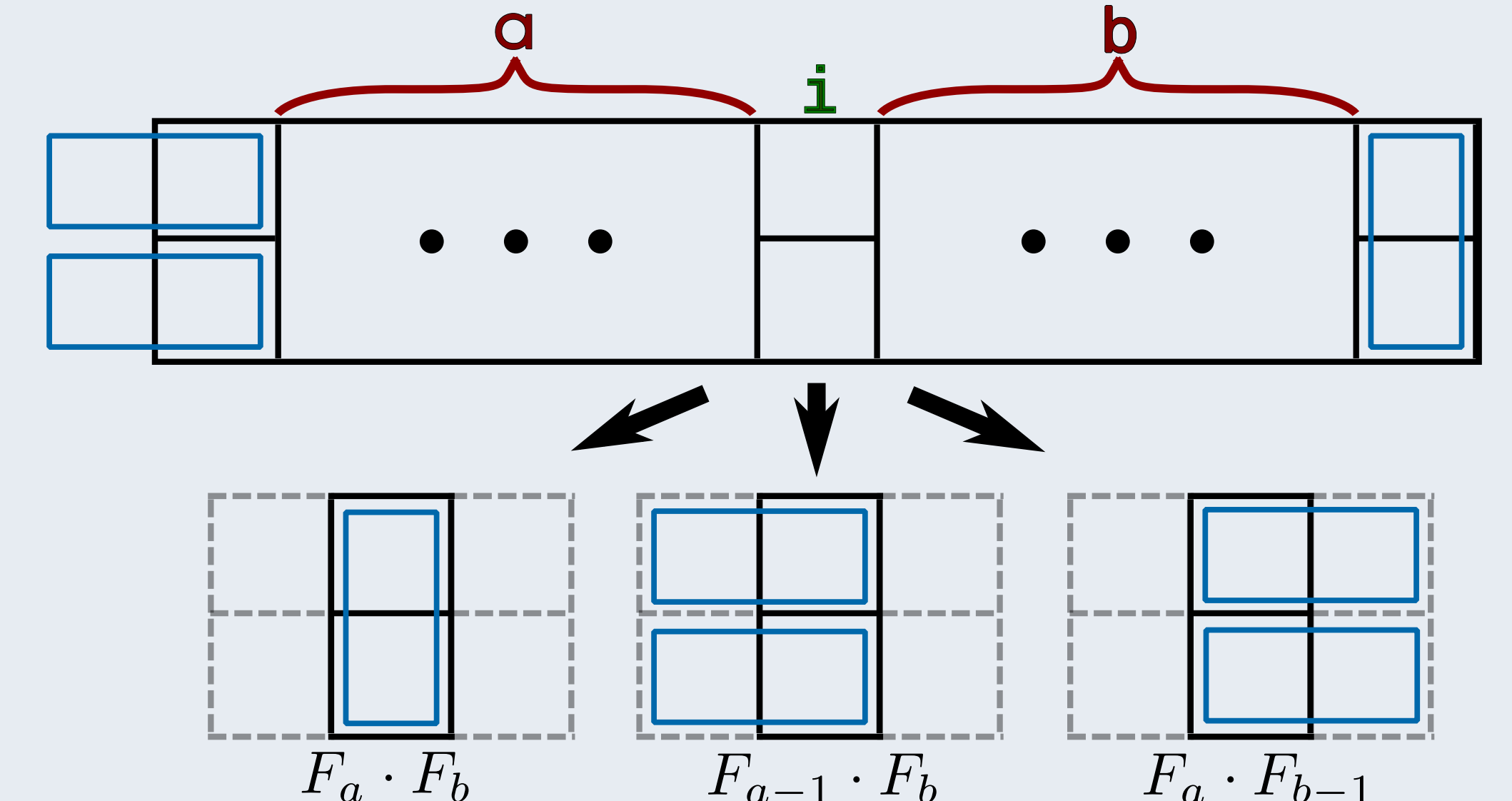
## Work in Progress

### Random Domino Tiling

A  $2 \times n$  grid tiled with dominoes:  $F_n$  tilings possible.



**Query:**  $i^{\text{th}}$  column configuration: *vertical, left, right?*



Sufficient to approximate  $F_c/F_{c-1}$ : Use  $\phi$  if  $c = \Omega(\log(n))$

**Open:**  $k \times n$  grid for  $k = \omega(1)$  and Dimer model

### Random Coloring: Glauber Dynamics

Consider a *uniform* random coloring of the input graph.

**Query:** What is  $v$ 's color in the random coloring?

**Global Algorithm** (Glauber Dynamics) for  $k > 2\Delta$

- Sample  $r = \mathcal{O}(n \log n)$  (vertex, color) pairs  $\langle (v_i, c_i) \rangle_{i \in [r]}$
- For steps  $i = 1, \dots, r$ 
  - If no neighbor of  $v_i$  has color  $c_i$ , set  $v_i$ 's color to  $c_i$
  - Else, do nothing

**Local Algorithm** for  $k = \Omega(\Delta \log n)$

- Locally sample *all* occurrences of  $(v, \star)$ : implemented effciently with the proposed **Counting-Generator**
- Sample  $(w, \star)$  if necessary, where  $w$  is neighbor of  $v$
- Query tree is of size  $\mathcal{O}(1)$  for  $k = \Omega(\Delta \log n)$

**Open:**  $k = o(\Delta \log n)$