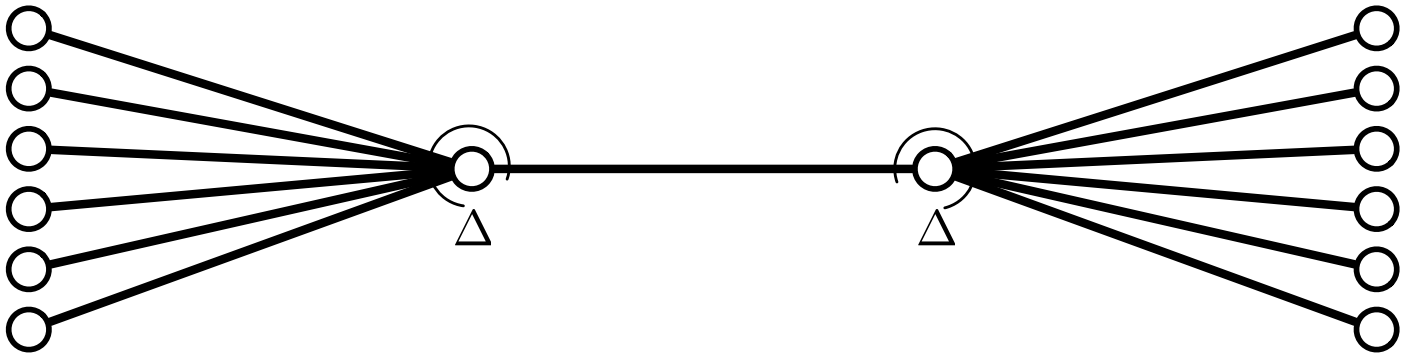
The Distributed Lovász Local Lemma

Seth Pettie
University of Michigan

R. Moser, G. Tardos. *J. ACM* 2010.
K.-M. Chung, S. Pettie, H.-H. Su, *Distributed Computing*, 2017.
S. Brandt, O. Fischer, J. Hirvonen, B. Keller, T. Lampaiänen, J. Rybicki, J. Suomela, J. Uitto, STOC 2016.
Y.-J. Chang, T. Kopelowitz, S. Pettie. FOCS 2016.
Y.-J. Chang, S. Pettie. FOCS 2017.
M. Fischer, M. Ghaffari. DISC 2017.
Y.-J. Chang, Q. He, W. Li, S. Pettie, J. Uitto. SODA 2018.
M. Ghaffari, D. Harris, F. Kuhn. *arxiv 2017.*

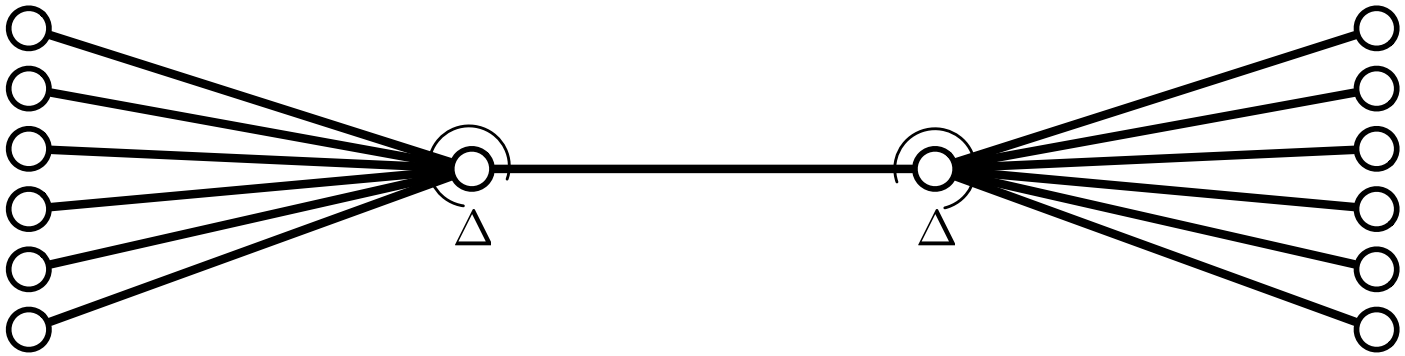
O(1)-time Randomized Experiments



- Max-degree = Δ; Palette size = $(1+\varepsilon)\Delta$.
- Each edge picks a color u.a.r.; permanently colors itself if there are no conflicts with adjacent edges.
  - E(new degree) = $\Delta\left(1-\frac{1}{(1+\epsilon)\Delta}\right)^{2(\Delta-1)} \approx \Delta e^{-2}$.
  - E(new palette size)

$$\approx (\Delta+1)\left(1-\frac{1}{(1+\epsilon)}\left(1-\frac{1}{(1+\epsilon)\Delta}\right)^{2\Delta}\right)^2 \approx (\Delta+1)(1-e^{-2})^2.$$

O(1)-time Randomized Experiments



- Max-degree = Δ; Palette size = $(1+\varepsilon)\Delta$.
- Each edge picks a color u.a.r., permanently colors itself if there are no conflicts with adjacent edges.
  - These estimates hold to within $(1+\delta)$ error with probability $\exp(-\delta^2\Delta)$.
  - Each event only depends on $O(\Delta^3)$ r.v.s
- If $\delta^2\Delta \gg \log n$, we're done. What if $\delta^2\Delta \gg \log\Delta$?

The LLL

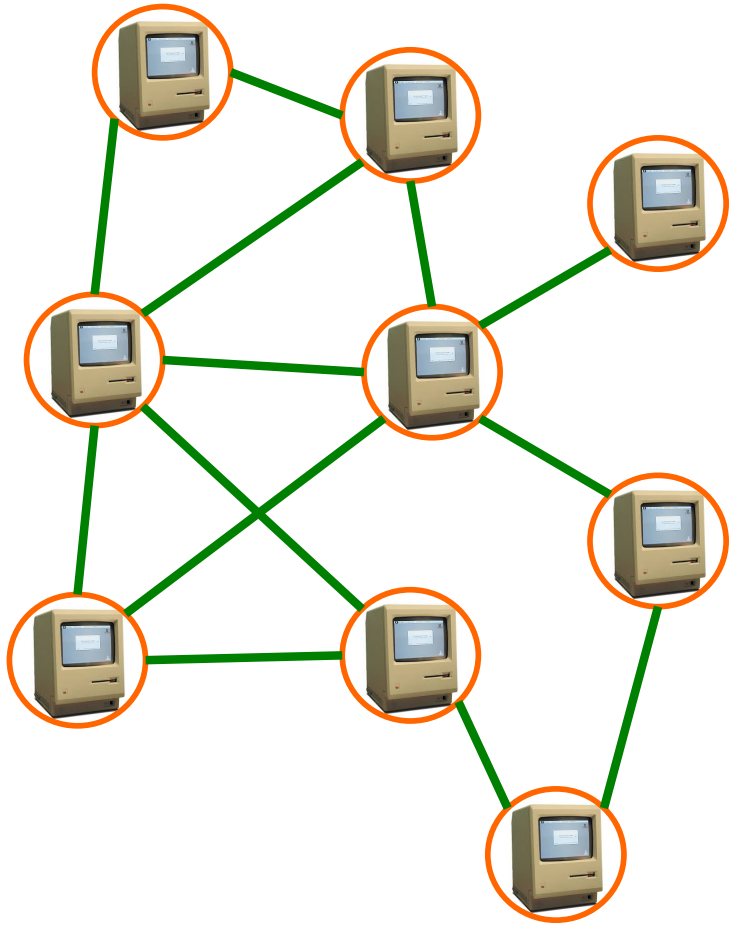
(symmetric, variable version)

- X = a set of *independent discrete random variables.*
- V = a set of "bad events".
  - $v \in V$ depends on $vbl(v) \subset X$.
- The *dependency graph* G=(V,E).
  - $E = \{(u, v) | \, vbl(u) \cap vbl(v) \neq \emptyset\}$.
- Parameters: $p = \max \Pr(u)$, $d =$ max degree in G.
- **Theorem.** If ep(d+1) < 1, there *exists* an assignment to X avoiding all bad events in V.

The LOCAL Model

[Linial'92]

- A graph G=(V,E)
  - Vertex = processor
  - Edge = bidirected communication
  - Time: ***synchronized*** rounds. In each round, each vertex sends a message to each neighbor.
  - ***Computation is free***.
  - ***Message size is unbounded.***
  - "Time" = number of rounds
  - N = number of vertices.
  - Δ = maximum degree.



- ***Randomized*** LOCAL
  - Can generate an ***unbounded number of random bits***

The Distributed LLL

- G = (V,E) is the **dependency graph** of the LLL instance.
- G is also the **communications network** of the LOCAL model.
- Problem: collectively compute an assignment to X that avoids all bad events.

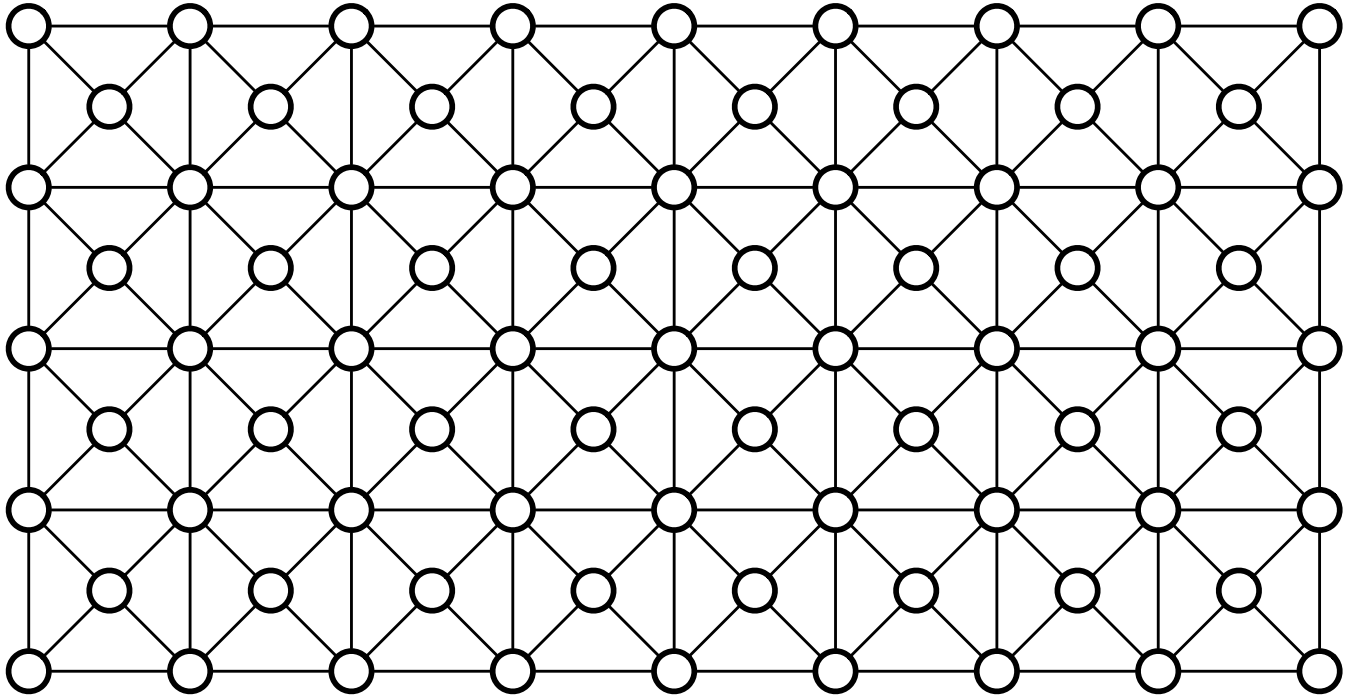
- In reality…
  - H is the LOCAL communications network.
  - We run an r=O(1)-round randomized "experiment" on H that satisfies an LLL criterion (ep(d+1)<1 or something similar.)
  - The dependency graph is H^{2r}. $d = \Delta^{2r} = \mathrm{poly}(\Delta)$
  - Any LLL algorithm executed on H^{2r} can be simulated on H with a factor 2r = O(1) slowdown.

Moser-Tardos [2010] Resampling

- Sample an initial assignment to the variables X.
- $V' = \{v \in V \mid$ v occurs under current assignment$\}$
- While$(V' \neq \emptyset)$
  - M = a maximal independent set of G(V', E).
  - vbl$(M) = \bigcup_{v \in M} vbl(v)$
  - Resample all variables in vbl(M).

- **Theorem.** If $ep(d+1)(1+\epsilon) < 1$, M-T ends after $O(\log_{1+\epsilon} n)$ steps. Time: $O(MIS \cdot \log_{1+\epsilon} n)$.

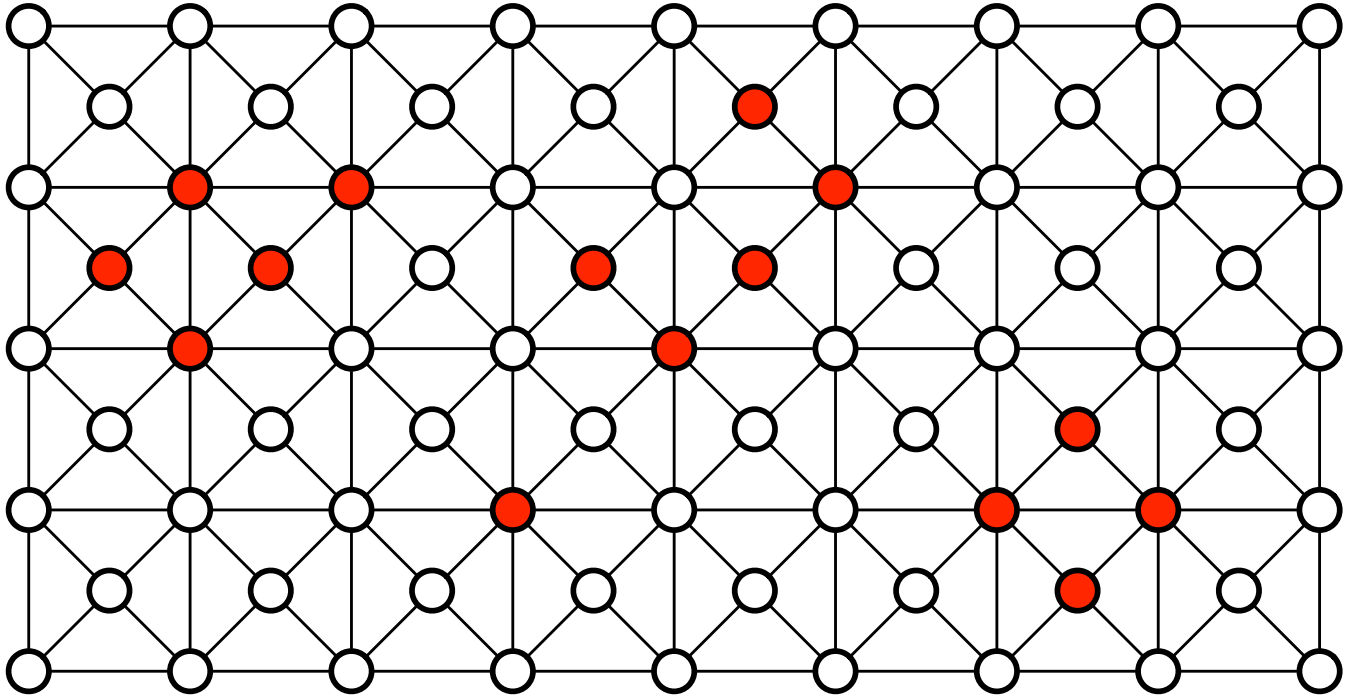
Moser-Tardos in action



Moser-Tardos in action

After the initial assignment to X:

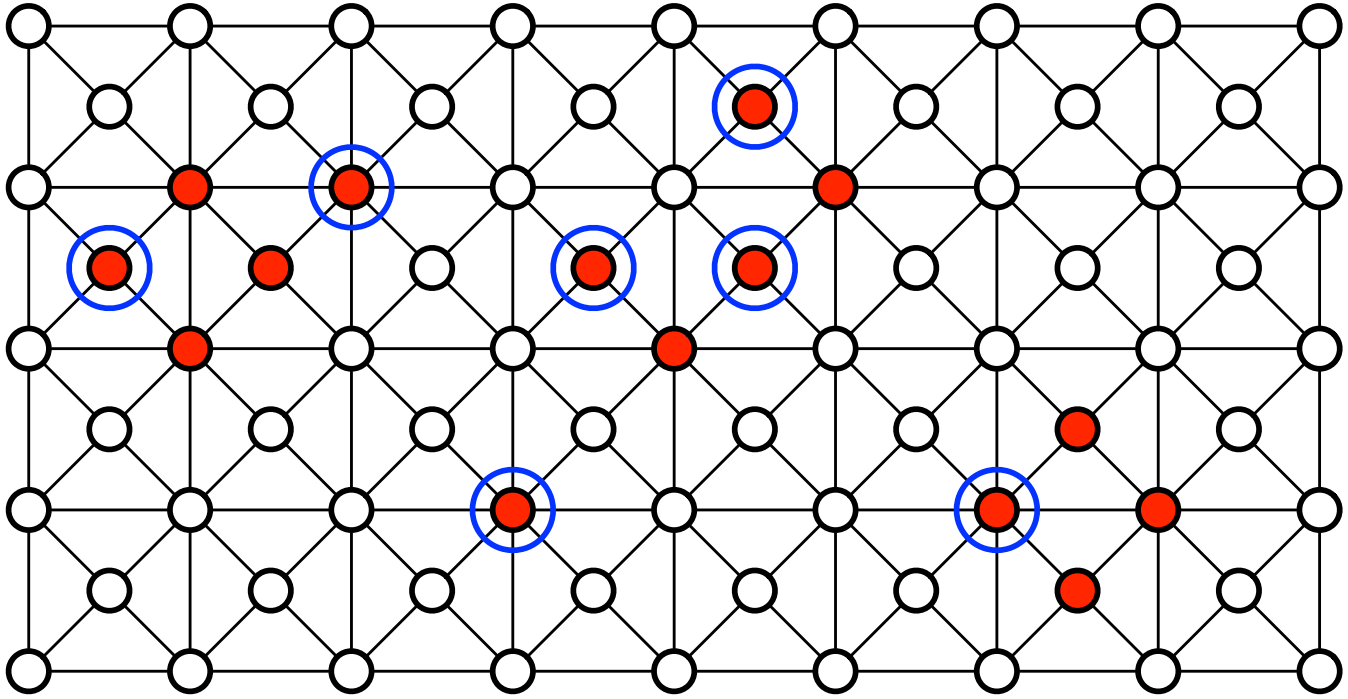
Red = bad event that occurs under current assignment



Moser-Tardos in action

Red = bad event that occurs under current assignment

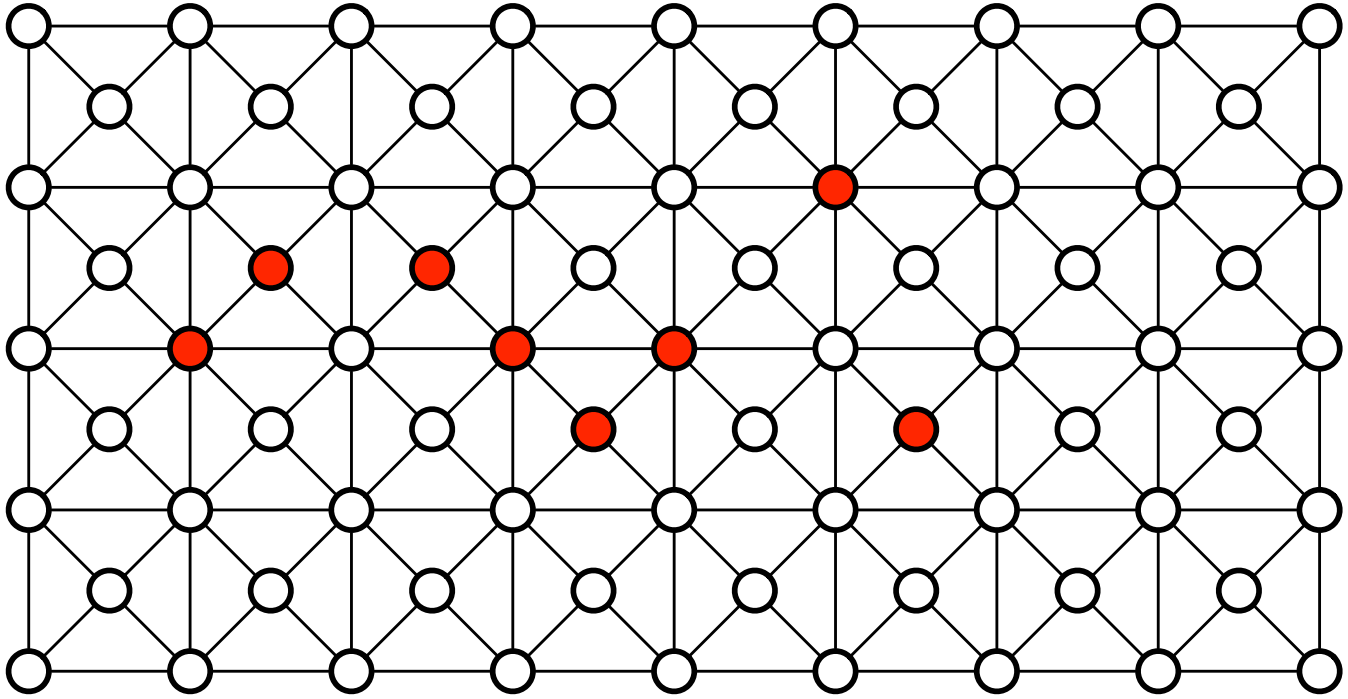
Blue = MIS of **red** nodes.



Moser-Tardos in action

Red = bad event that occurs under current assignment

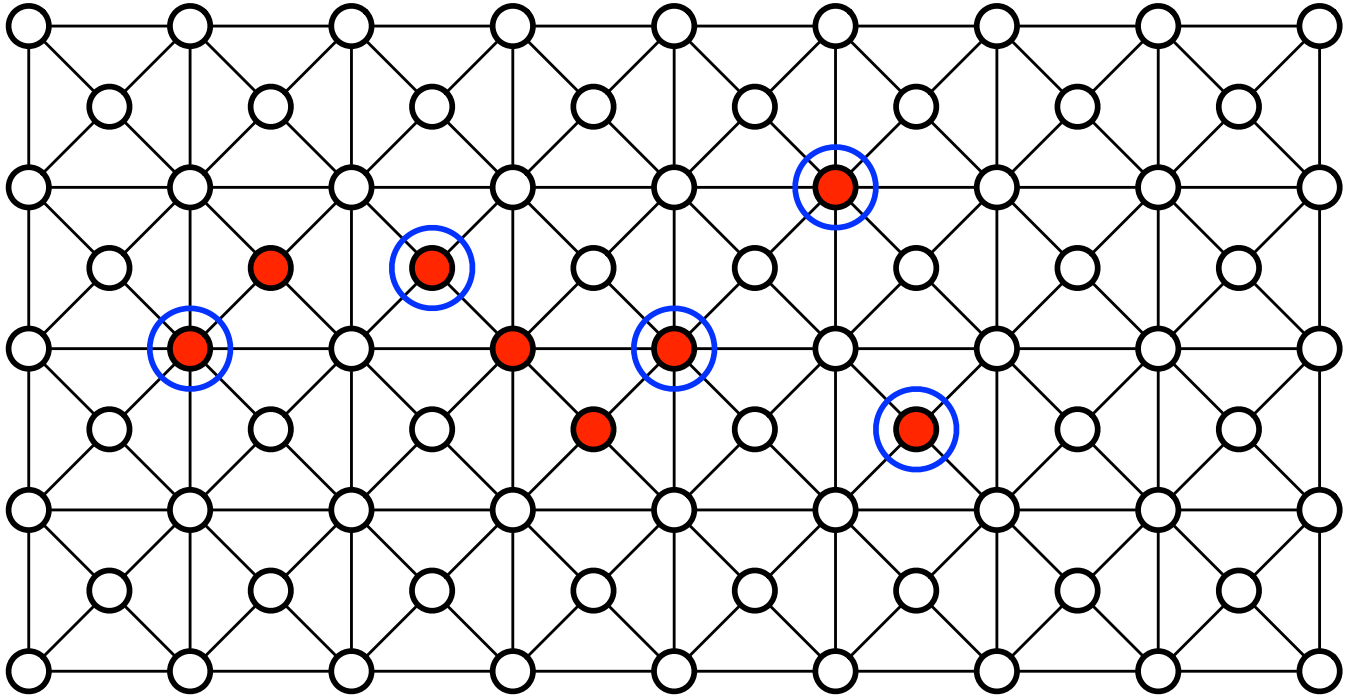
Blue = MIS of **red** nodes.



Moser-Tardos in action

Red = bad event that occurs under current assignment

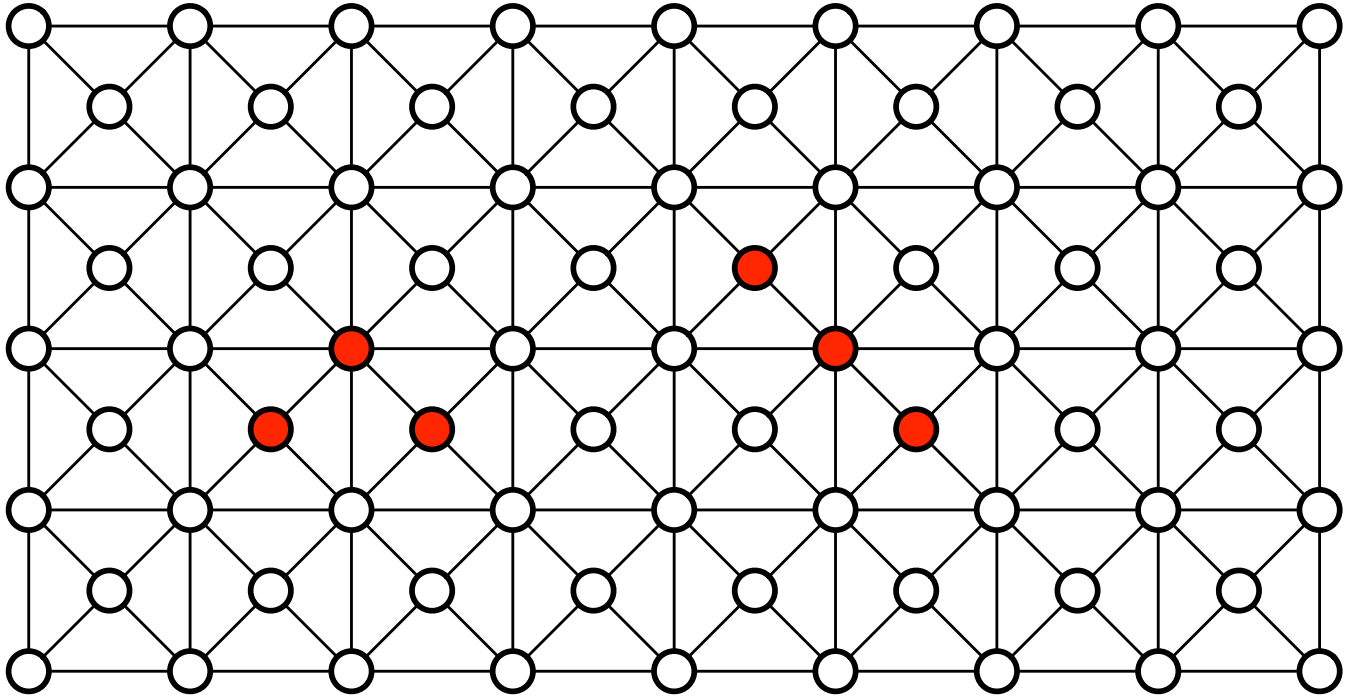
Blue = MIS of **red** nodes.



Moser-Tardos in action

Red = bad event that occurs under current assignment

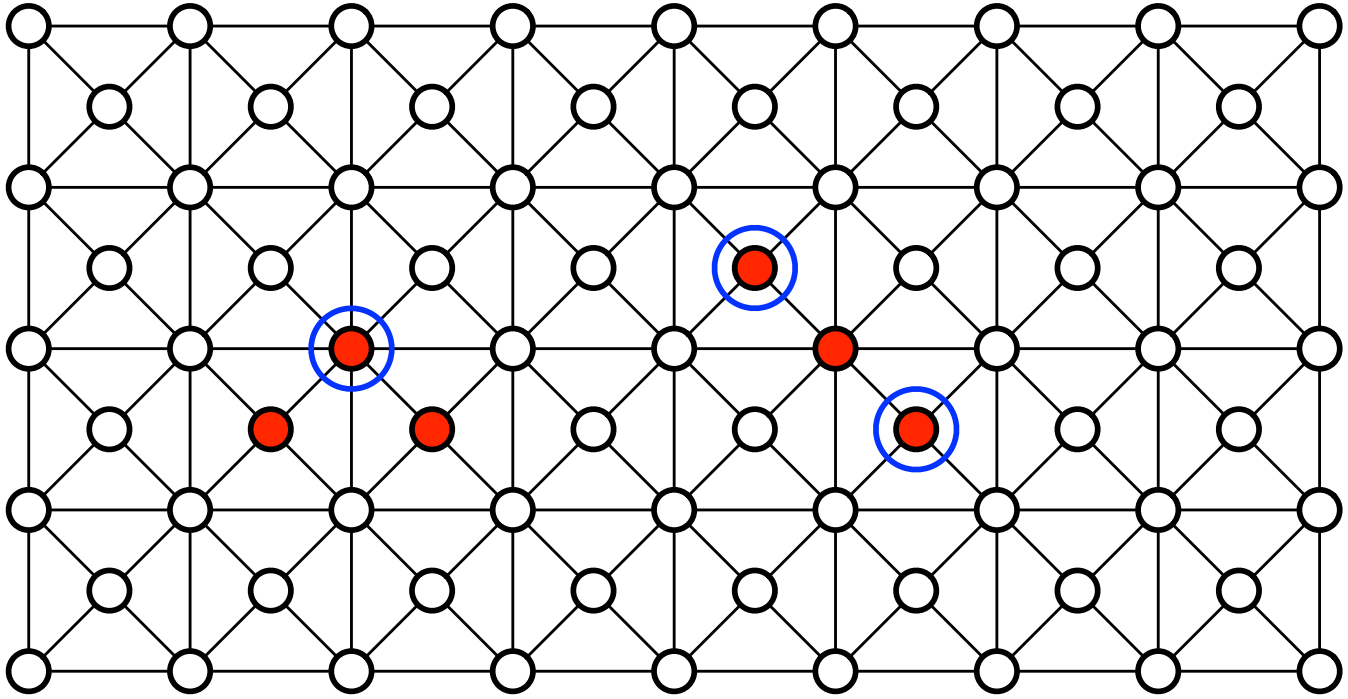
Blue = MIS of **red** nodes.



Moser-Tardos in action

Red = bad event that occurs under current assignment

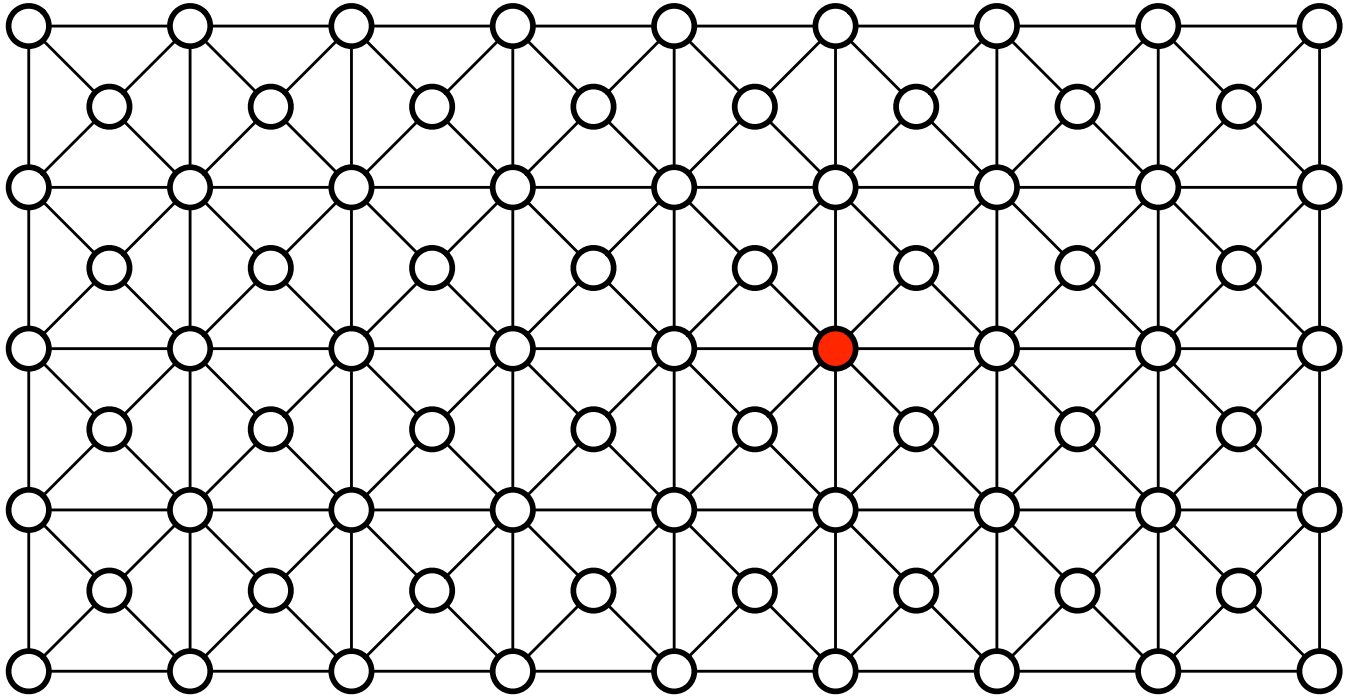
Blue = MIS of **red** nodes.



Moser-Tardos in action

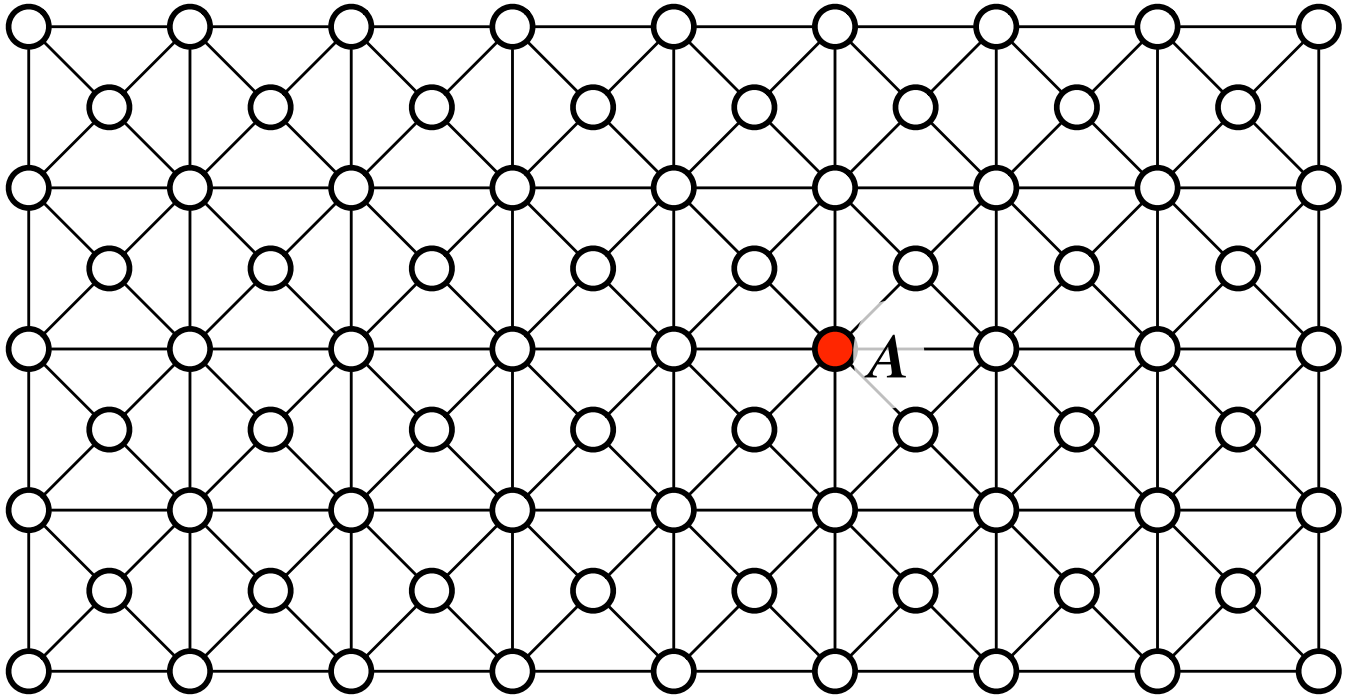
Red = bad event that occurs under current assignment

Blue = MIS of **red** nodes.



Moser-Tardos in action

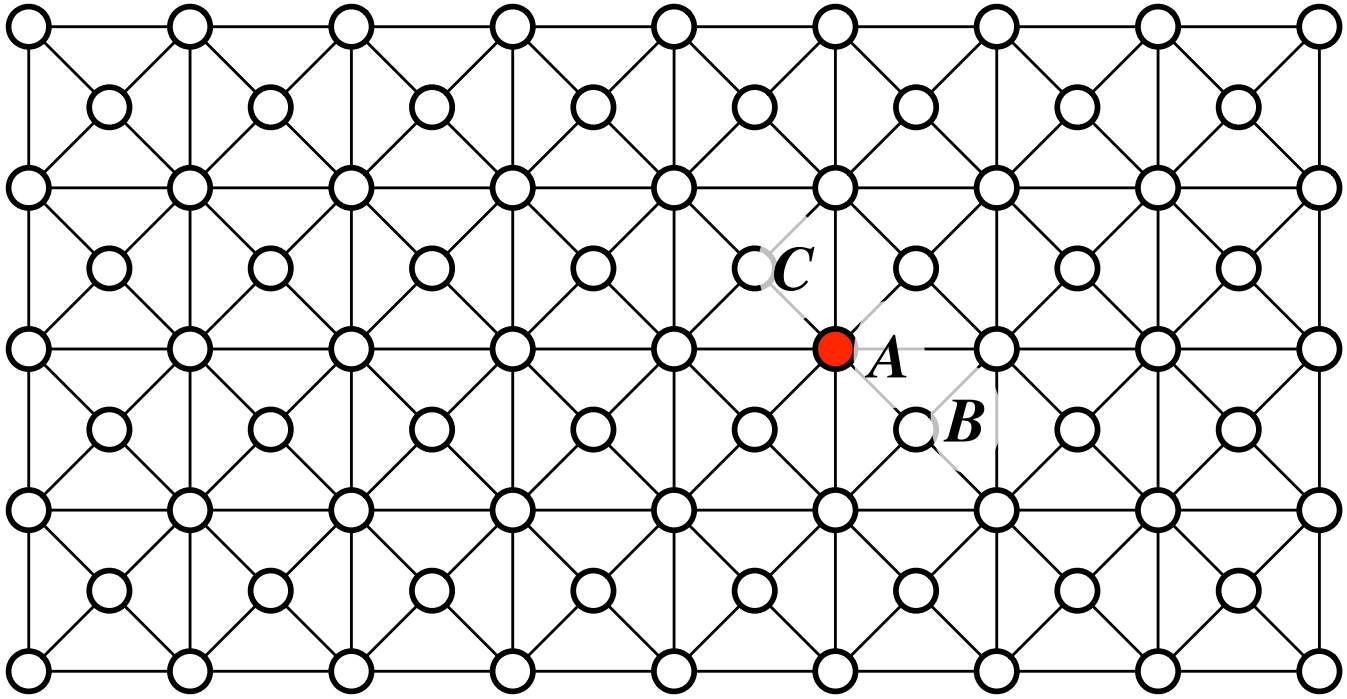
"Witness tree" : rooted at a resampled node; descendants a function of the resampling transcript *in reverse chronological order*.

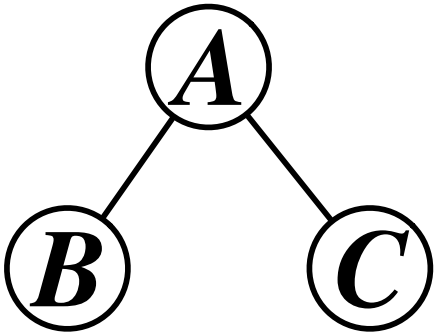


Ⓐ

Moser-Tardos in action

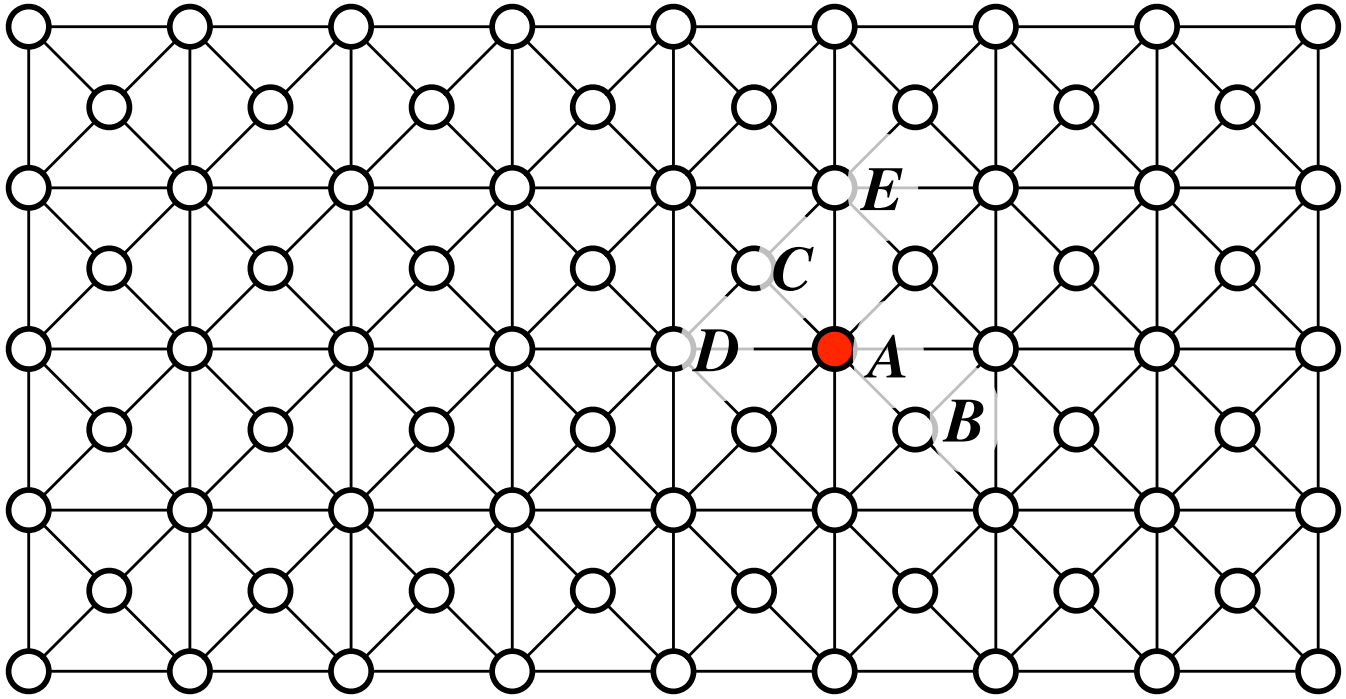
"Witness tree" : rooted at a resampled node; descendants a function of the resampling transcript *in reverse chronological order*.

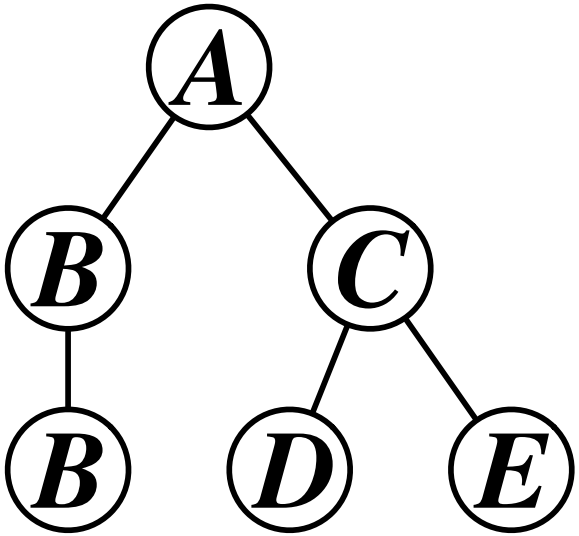




Moser-Tardos in action

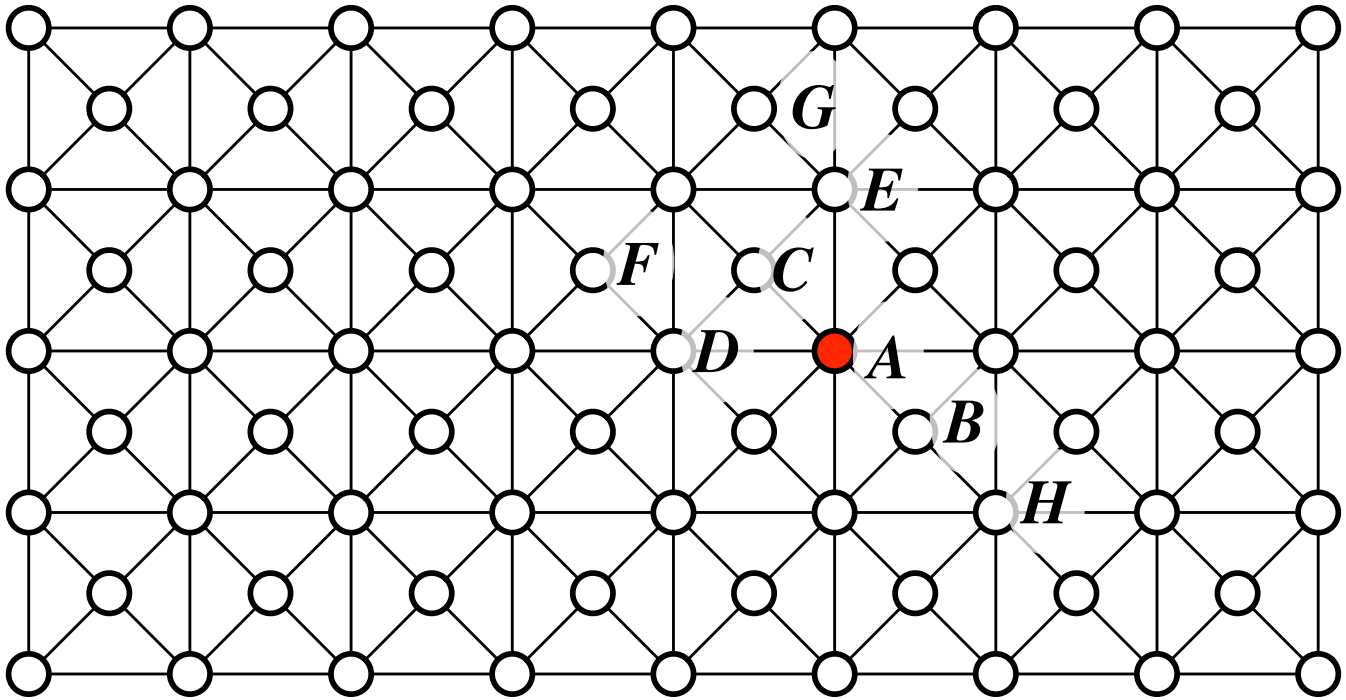
"Witness tree" : rooted at a resampled node; descendants a function of the resampling transcript *in reverse chronological order*.

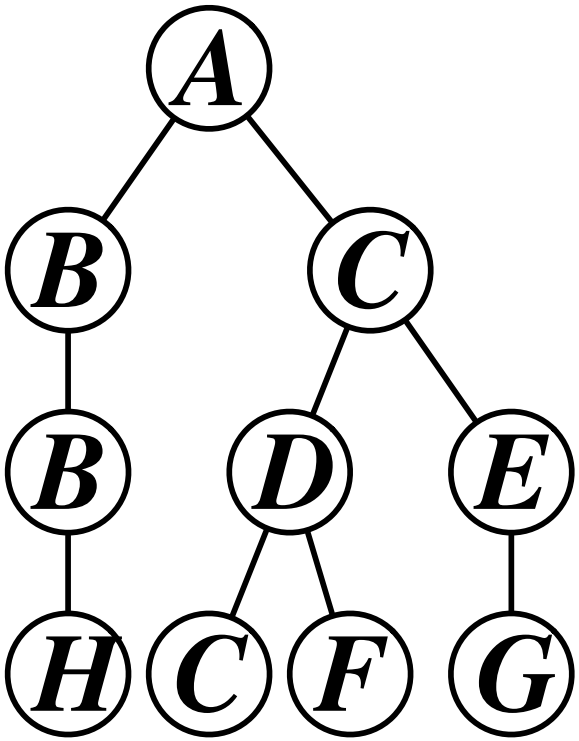




Moser-Tardos in action

"Witness tree" : rooted at a resampled node; descendants a function of the resampling transcript *in reverse chronological order*.





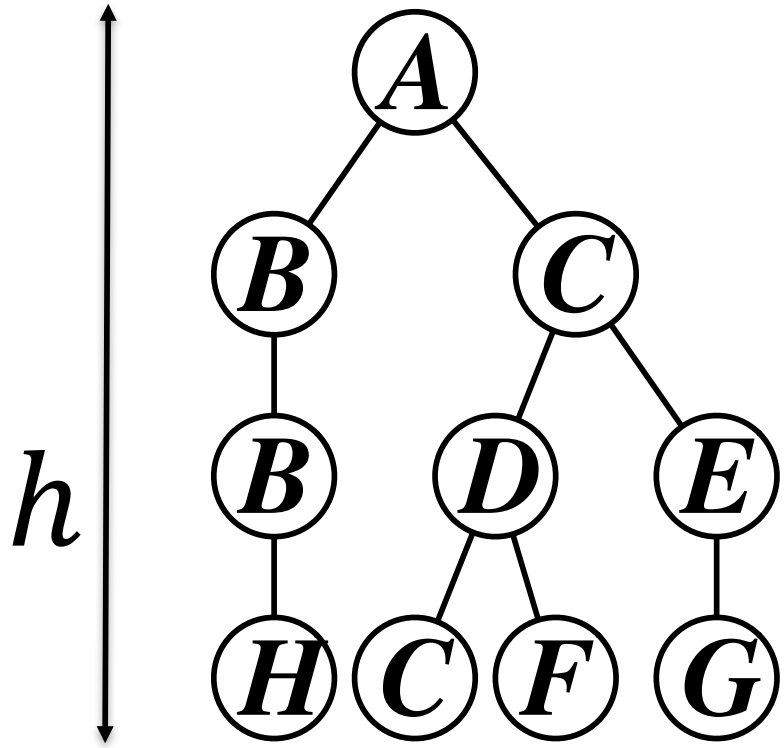
Moser-Tardos in action

Recall the LLL criterion: $ep(d+1)(1+\epsilon) < 1$

Pr(seeing <u>*this*</u> witness tree)
$\leq p^{size}$

Number of labeled witness trees with *size* nodes
$\leq n(e(d+1))^{size}$

W.h.p., all witness trees have size
$O(\log_{1+\epsilon} n)$



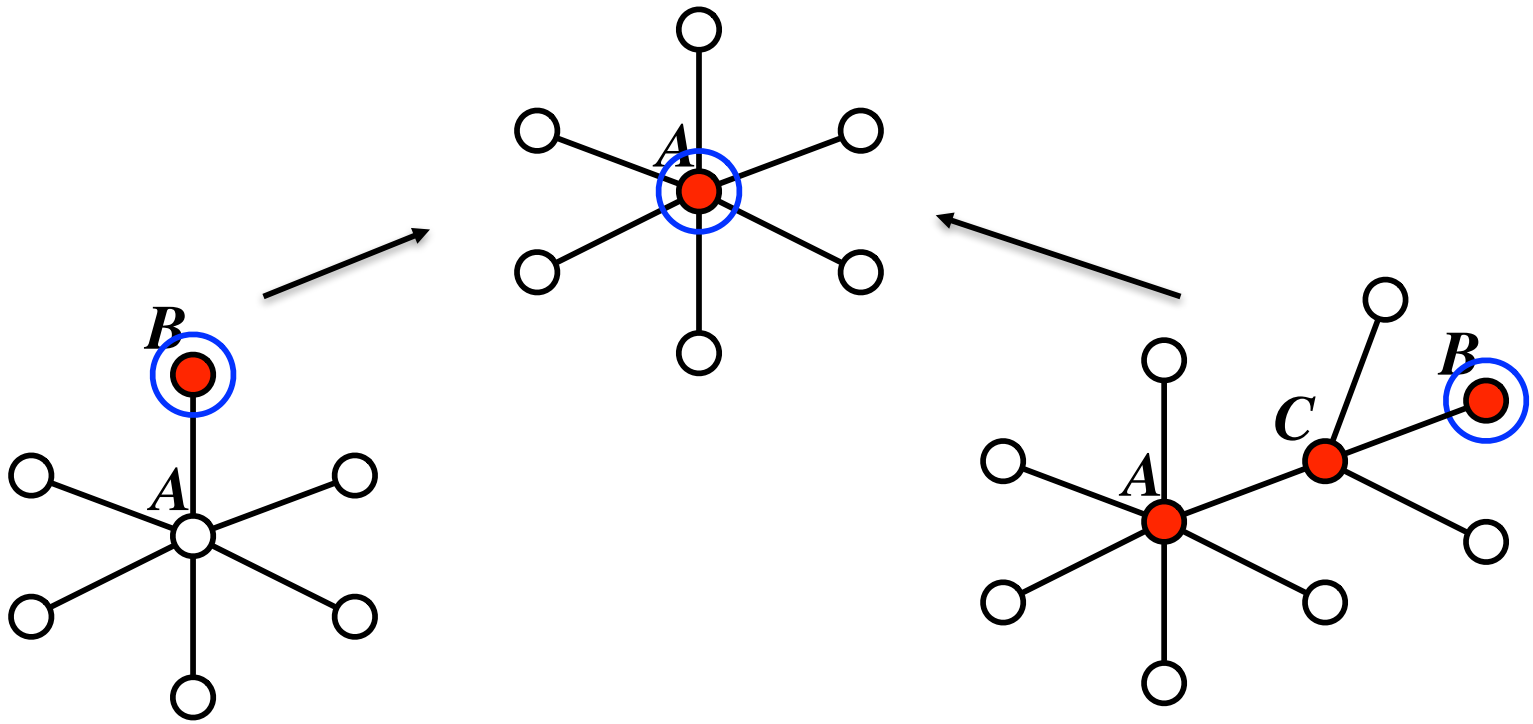
Chung-Pettie-Su [2014] Resampling

- All vertices/bad events given ***unique IDs***.
- Sample an initial assignment to the variables X.
- $V' = \{v \in V \mid$ v occurs under current assignment$\}$
- While$(V' \neq \emptyset)$
  - $U = \{u \in V' \mid ID(u) < ID(v),\ (u,v) \in E, v \in V'\}$
  - Resample all variables in $\bigcup_{u \in U} vbl(u)$.

- Moser-Tardos-type analysis goes through, using 2-neighborhood in lieu of 1-neighborhood.
- **<u>Theorem.</u>** If $epd^2(1+\epsilon) < 1$, $O(\log_{1+\epsilon} n)$ C-P-S resampling steps suffice. Time: $O(\log_{1+\epsilon} n)$.

Time t:

Time t-1:



Lower Bounds

Brandt, Fischer, Hirvonen, Keller, Lempaiänen, Rybicki, Suomela, Uitto 2016
Chang, Kopelowitz, Pettie, 2016
further simplified by Chang, He, Li, Pettie, Uitto 2018

- Randomized LLL algorithms take $\Omega(\log \log n)$ time.
- Deterministic LLL algorithms take $\Omega(\log n)$ time.
- New Problem: ***sinkless orientation***. Given Δ-regular undirected graph G=(V,E), find an orientation of each edge s.t. no vertex has out-degree 0.
  - An LLL instance with: $p = 2^{-\Delta}, d = \Delta$.

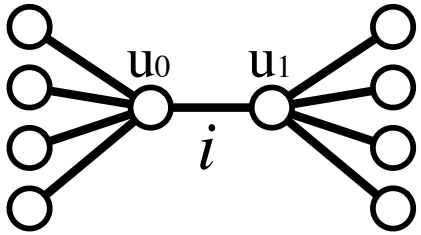
Lower Bounds on Sinkless Orientation/LLL

- Simplifying assumptions:
  - Processors sit on the edges; two processors can communicate if their edges touch.
  - The graph is bipartite and 2-vertex colored.
  - The graph is 2Δ-edge colored.
  - The graph is an infinite Δ-regular tree.
- "Running time" is a vector $(t_1, t_2, \ldots, t_{2\Delta})$
  - Edges colored j terminate in t_j rounds.

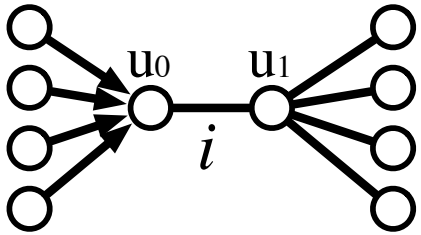
Lower Bounds on Sinkless Orientation/LLL

- Proof idea: take a randomized algorithm running in time $(t)^i \cdot (t-1)^{2\Delta-i}$ with error prob. p, transform it into one with time $(t)^{i-1} \cdot (t-1)^{2\Delta-(i+1)}$, error prob. $O(p^{1/3})$.
  - Only edges colored i will change their algorithm.

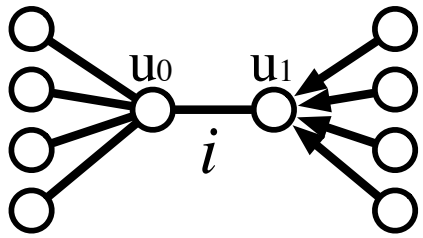
- Fix a specific edge $\{u_0,u_1\}$ colored i.



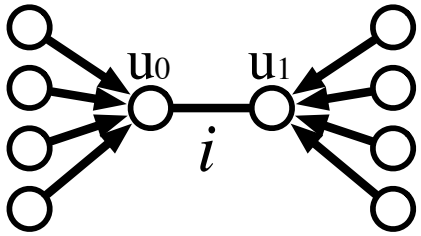
- Fix a specific edge $\{u_0,u_1\}$ colored i.
- E_0 : all neighbors of u_0 oriented towards u_0.



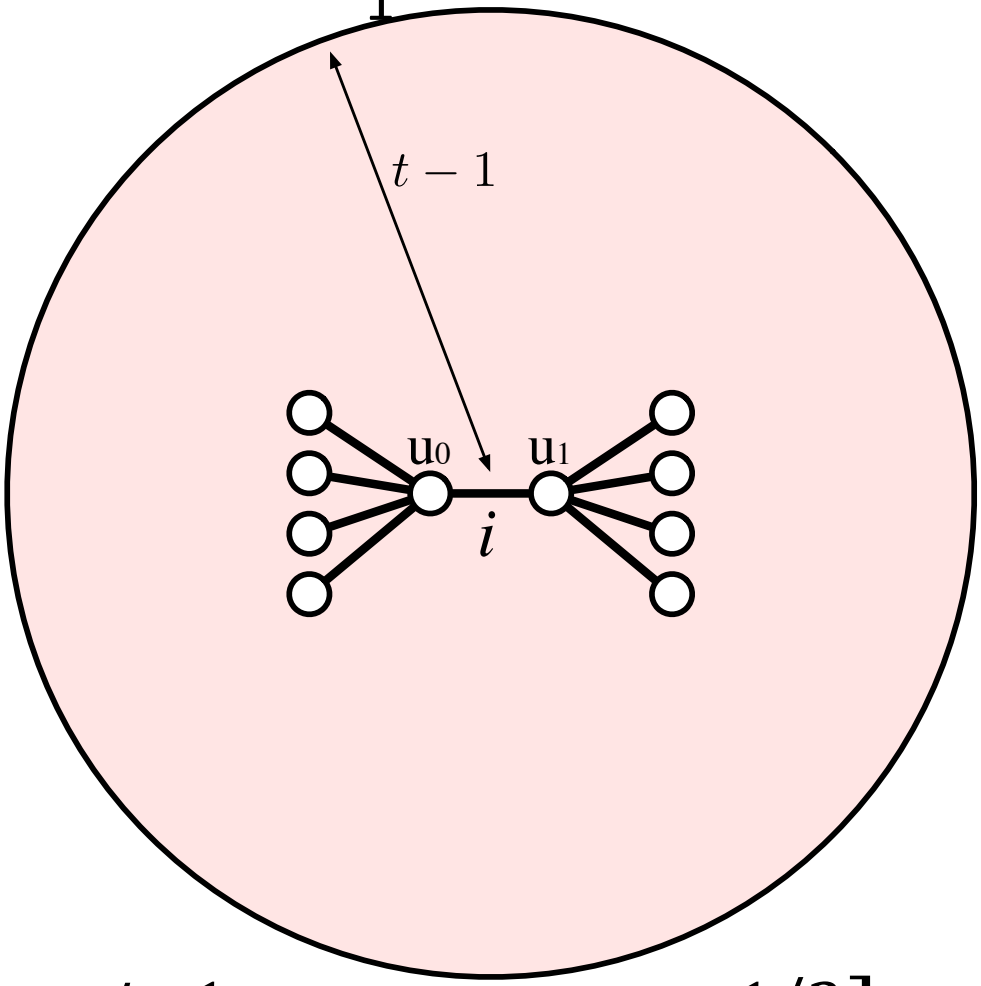
- Fix a specific edge $\{u_0,u_1\}$ colored i.
- E_0 : all neighbors of u_0 oriented towards u_0.
- E_1 : all neighbors of u_1 oriented towards u_1.



- Fix a specific edge $\{u_0,u_1\}$ colored i.
- E_0 : all neighbors of u_0 oriented towards u_0.
- E_1 : all neighbors of u_1 oriented towards u_1.
- $\Pr(E_0 \cap E_1) \leq 2p$.

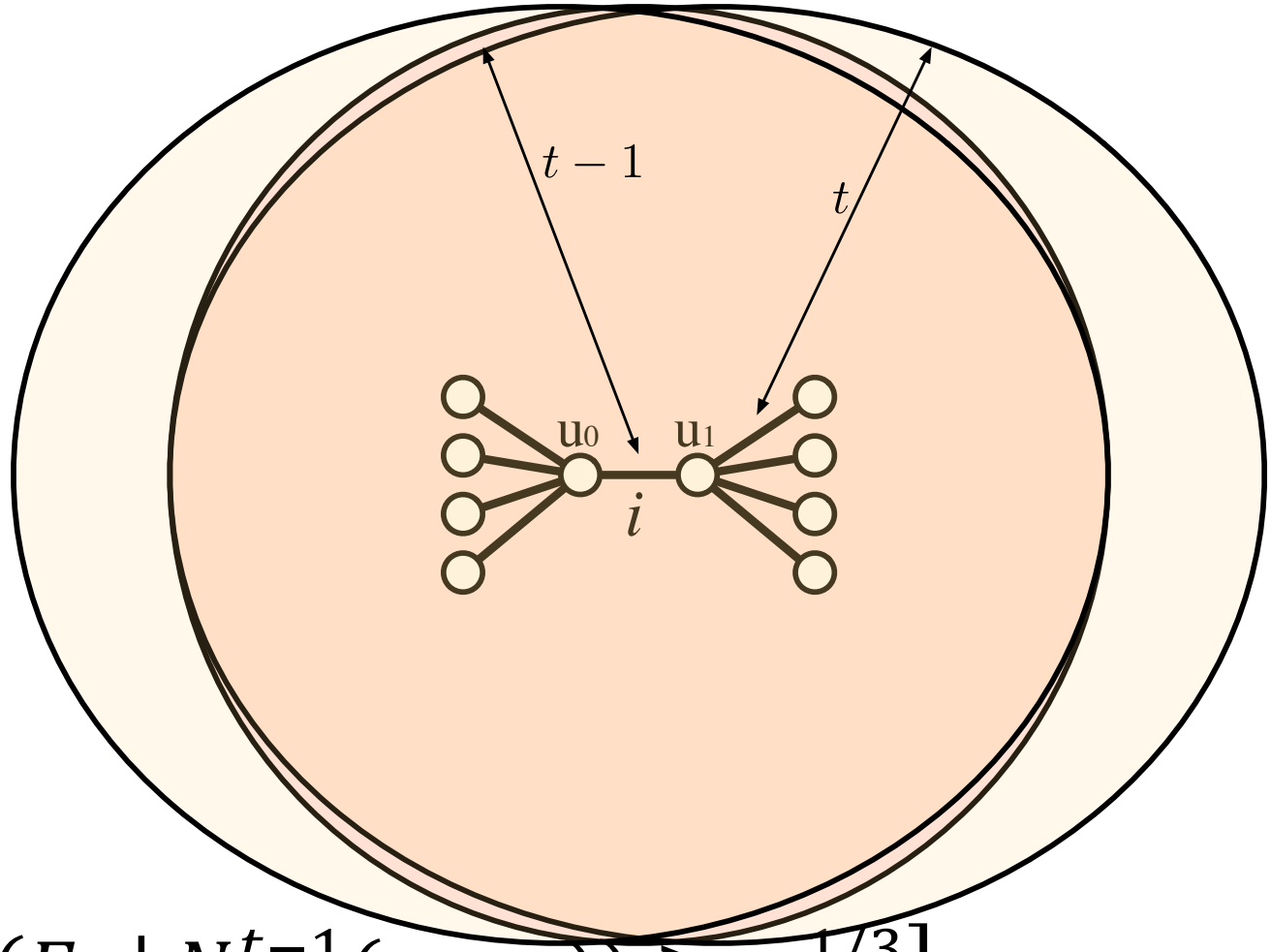


- E_0 : all neighbors of u_0 oriented towards u_0.
- E_1 : all neighbors of u_1 oriented towards u_1.



- E_0^*: $\left[\Pr(E_0 \mid N^{t-1}(u_0, u_1)) \geq p^{1/3}\right]$
- E_1^*: $\left[\Pr(E_1 \mid N^{t-1}(u_0, u_1)) \geq p^{1/3}\right]$

- **Claim.** $\Pr(E_0 \cap E_1 \mid E_0^* \cap E_1^*) \geq p^{\frac{2}{3}}$.



- E_0^*: $\left[\Pr(E_0 \mid N^{t-1}(u_0, u_1)) \geq p^{1/3}\right]$
- E_1^*: $\left[\Pr(E_1 \mid N^{t-1}(u_0, u_1)) \geq p^{1/3}\right]$

- **Claim.** $\Pr(E_0 \cap E_1 \mid E_0^* \cap E_1^*) \geq p^{\frac{2}{3}}$.

  $\rightarrow \Pr(E_0^* \cap E_1^*) \leq 2p^{\frac{1}{3}}$.

- The new algorithm:
  - If E_0^* holds, orient as ($u_0 \rightarrow u_1$), otherwise, orient ($u_1 \rightarrow u_0$)
- Two ways to fail:
  - $\overline{E_0^*} \cap E_0$: Probability this happens is $\leq p^{1/3}$.
  - $E_0^* \cap E_1$: Probability this happens is $\leq 3p^{1/3}$.
    - $E_0^* \cap E_1^* \cap E_1$: Probability this happens is $\leq 2p^{1/3}$.

      +
    - $E_0^* \cap \overline{E_1^*} \cap E_1$: Probability this happens is $\leq p^{1/3}$.

Lower Bounds

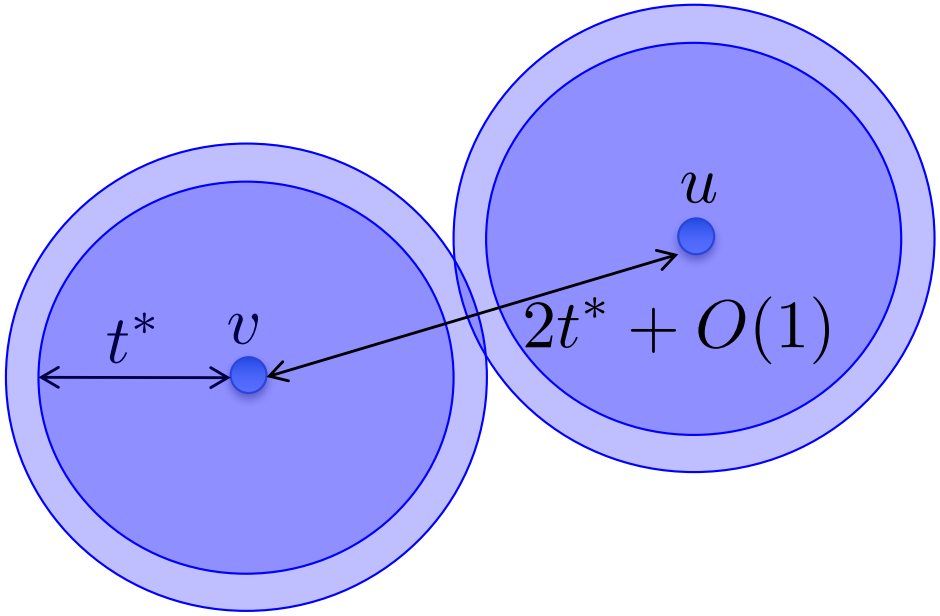
- Transform any time (t,t,…,t) algorithm with error probability p into a time (0,0,…,0) algorithm with error probability $p^{1/3^{2\Delta}}$.
  - 0-round algorithms have high probabilities of failure.
  - If p = 1/poly(n), then $t = \Omega(\Delta^{-1} \log\log n)$.
  - If p = 0, then we need to think about 2 issues.
    - Impossible to solve for *deterministic*, *anonymous* nodes. Need to think about role of unique IDs.
    - The argument breaks down if the algorithm can see a cycle. Proof works up to t < girth/2. Apply to Δ-regular graphs with girth $\Omega(\log_\Delta n)$.

Completeness

The LLL is complete for *sublogarithmic* time

- Suppose we have a randomized distributed LLL algorithm for criterion $p(ed)^c < 1$, for any (possibly big) constant c. The algorithm runs in T_{LLL} time.
- Suppose algorithm A solves some locally checkable labeling problem runs in sublogarithmic time. Then A can be automatically sped-up to run in $O(T_{LLL})$ time.

- Suppose A solves some LCL problem in *sub*logarithmic time with failure probability 1/n.
  - For any ε>0, can write time as $T(n, \Delta) \leq C(\Delta) + \epsilon \log_\Delta n$
- n* = min. value such that: $T(n, \Delta) = t^* \leq \frac{1}{2c} \log_\Delta n^* - O(1)$
  - Follows that t* = O(C(Δ)).



every vertex sees a subgraph that is *consistent with an n*-vertex graph.*

- Build the dependency graph:
  - X_v = the random bits generated locally at v.
  - vbl(v) = $\{X_u \mid u \in N^{t^*+O(1)}(v)\}$
  - E_v = the event that v's neighborhood is incorrectly labeled, when running alg. A with "n" = n^*.
  - H = $(\{E_v\}, \{(E_u,E_v) \mid \text{dist}(u,v) \le 2t^*+O(1)\})$
  - LLL parameters: $p = 1/n^*$, $d = \Delta^{2t^*+O(1)}$

$$pd^c = p \cdot \Delta^{c(2t^*+O(1))} < (1/n^*)n^* = 1$$

- Run a distributed LLL algorithm on "H."
  - 1 step in H simulated with $O(C(\Delta))$ steps in G.
  - Alg. A can be automatically sped up to $O(C(\Delta)\cdot T_{LLL})$ time.

Modern Distributed LLL Algorithms

- The ***graph shattering*** method: [Barenboim, Elkin, Pettie, Schneider 2016]
  - **Step 1** (*Rand.*): Reduce a graph problem on n vertices to many disjoint problems on poly(log n) vertices.
  - **Step 2** (*Det.*): Solve each subproblem using the best available deterministic algorithm.

- [Fischer-Ghaffari'17] + [Molloy-Reed'98]
  - Transform 1 LLL instance with

		parameters	LLL criterion
size	n	(p, d)	$p(ed)^{\lambda} < 1$

  - Into several LLL instances with

size	$poly(d)\log n$	$\left(\sqrt{p}, \mathrm{d}\right)$	$\sqrt{p}(ed)^{\lambda/2} < 1$

  - In time $O(d^2 + \log^* n)$.

Modern Distributed LLL Algorithms

- The ***graph shattering*** method: [Barenboim, Elkin, Pettie, Schneider 2016]
  - **Step 1** (*Rand.*): Reduce a graph problem on n vertices to many disjoint problems on poly(log n) vertices.
  - **Step 2** (*Det.*): Solve each subproblem using the best available deterministic algorithm.
  - **Step 3** *Derandomize* the algorithm from (Step 1)+(Step 2) and plug it back in as the deterministic algorithm in Step 2.

- [Ghaffari, Harris, Kuhn'17] LLL instances satisfying $pd^8 = O(1)$ are solved in time:

$$\exp^{(i)}\left(O\left(\log d + \sqrt{\log^{(i+1)} n}\right)\right)$$

Graph Shattering via Complex Contagions

[Chang, He, Li, Pettie, Uitto 2018]

- Consider a ***partial*** assignment ϕ to the variables X.
- Event v is called "dangerous" if $\Pr(v \mid \phi) \geq \sqrt{p}$.

- Pick a random total assignment ϕ.
- Until there are no dangerous events w.r.t. ϕ:
  - For each dangerous event v, ***unset*** all of vbl(v).

- If $pd^{O(1)} < 1$, this algorithm terminates in $O(\log n)$ rounds and shatters the dependency graph into $poly(d)\log n$-size components.

Graph Shattering via Complex Contagions

- Complex Contagions
  - At time 0, every node is **infected** with prob. $p_0 = d^{-O(1)}$.
  - Any node with $> \mu$ **infected** neighbors becomes **infected**.
  - A state is ***stable*** if it causes no more infection.
- Stable State Problem. Given U_0 infected at time 0, find a *non-trivial* stable state $L \supset U_0$.

Graph Shattering via Complex Contagions

- ***The Hypochondriac Algorithm:***
  - $U_0 =$ sample each vertex with probability p_0.
  - For i from 1 to $\tau = O(\log \log n)$:
    - $U_i = U_{i-1} \cup \left\{ v \mid v \text{ has } > \frac{\mu}{2} \text{ neighbors in } U_{i-1} \right\}$.
  - $L_0 = U_\tau$
  - For i from 1 to τ
    - $L_i = U_0 \cup L_{i-1} \backslash \{ v \mid v \text{ has } \leq \mu \text{ neighbors in } L_{i-1} \}$.
  - Return L_τ.

Sublogarithmic LLL Algorithms

- ***Theorem.*** (informal) If the graph is "tree structured" (T^r for some tree T), then the hypochondriac algorithm finds a non-trivial stable state, w.h.p.

- ***Theorem.*** (informal) [Chang, He, Li, Pettie, Uitto'18] Any "tree structured" LLL dependency graph can be shattered into $poly(d)\log n$-size LLL instances in $O(\log\log n)$ time (via the hypochondriac algorithm.)

The End (is near)

- **Conjecture.** [Chang, Pettie 2017] The randomized LOCAL complexity of the LLL is $O(\log \log n)$ under some polynomial LLL criterion $pd^c = O(1)$.
  - If true, we know what the algorithm must look like:
    - Some $O(\log \log n)$-time algorithm to shatter the dependency graph into poly(log n)-size components.
    - Some $O(\log n)$-time deterministic LLL algorithm.

- Is LLL with criterion $pd = O(1)$ ***strictly harder*** than $pd^c = O(1)$?

The End