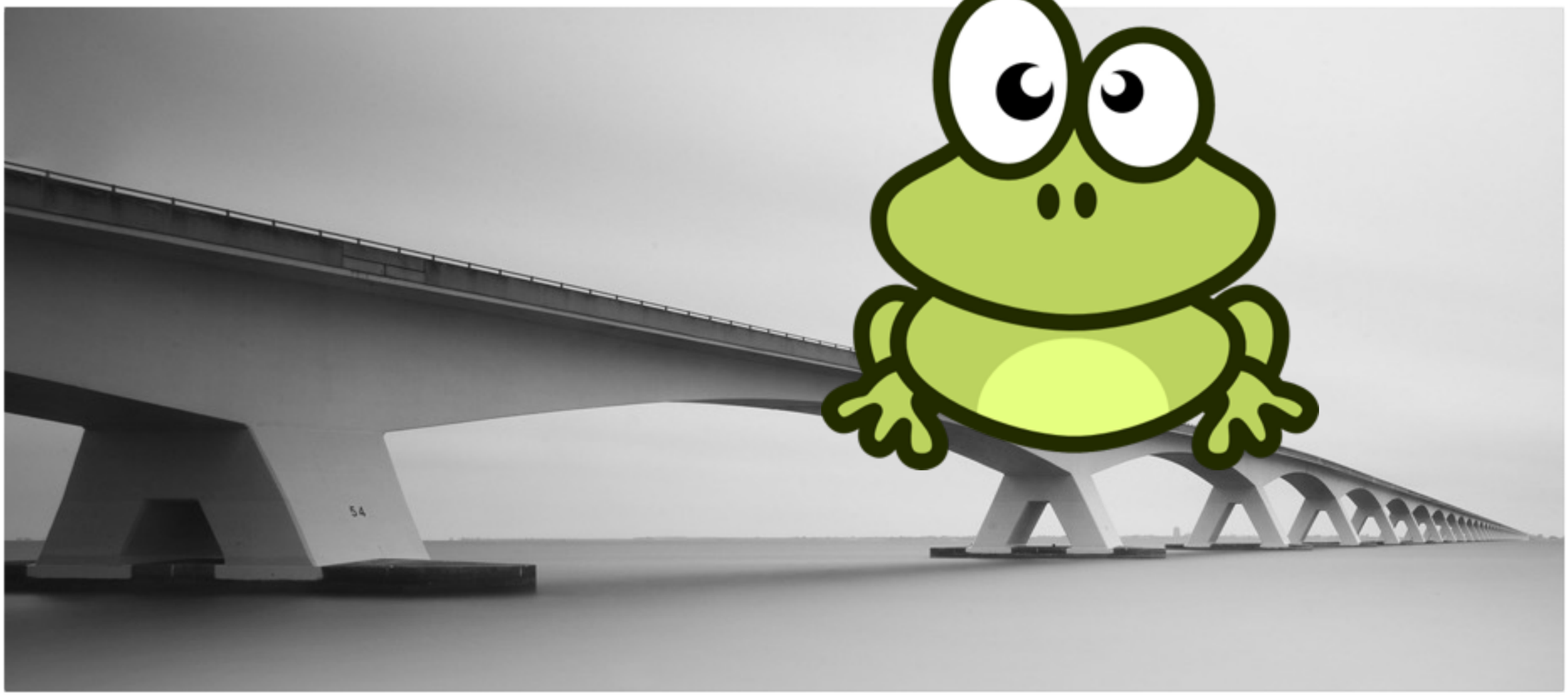


FrogWild!

Fast PageRank Approximations on Graph Engines

Ioannis Mitliagkas
Michael Borokhovich
Alex Dimakis
Constantine Caramanis

VLDB 2015

Contribution

- Fast algorithm for pagerank approximation
  - want quickly identify **top-k pagerank** vertices
- Implemented in GraphLab: 7-10x faster than normal pagerank function
- Provable approximation guarantees for graphs with no bad structure

PageRank [Page et al., 1999]

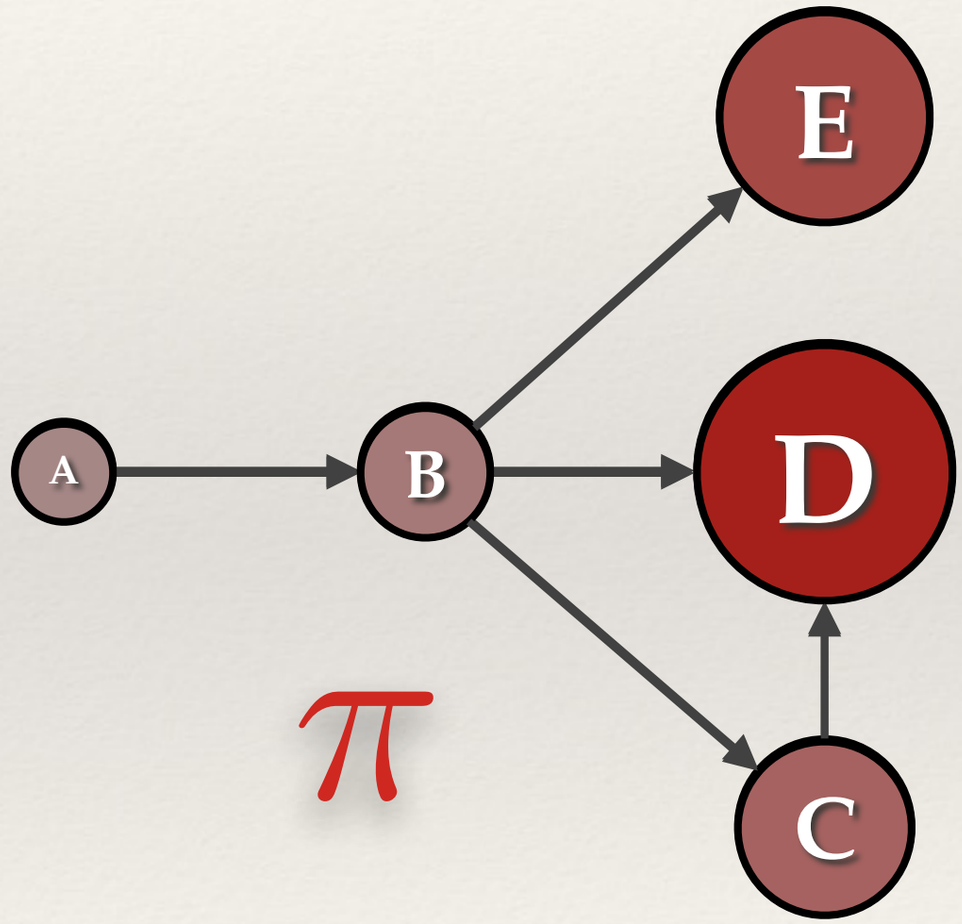
Page Importance

Described by distribution π

Recursive Definition

Important pages are pointed to by

- important pages are pointed to by
  - important pages are pointed to by…



Robust

to manipulation by spammer networks

PageRank Applications

Telco

Find the most influential users in a “calls” graph

Social networks

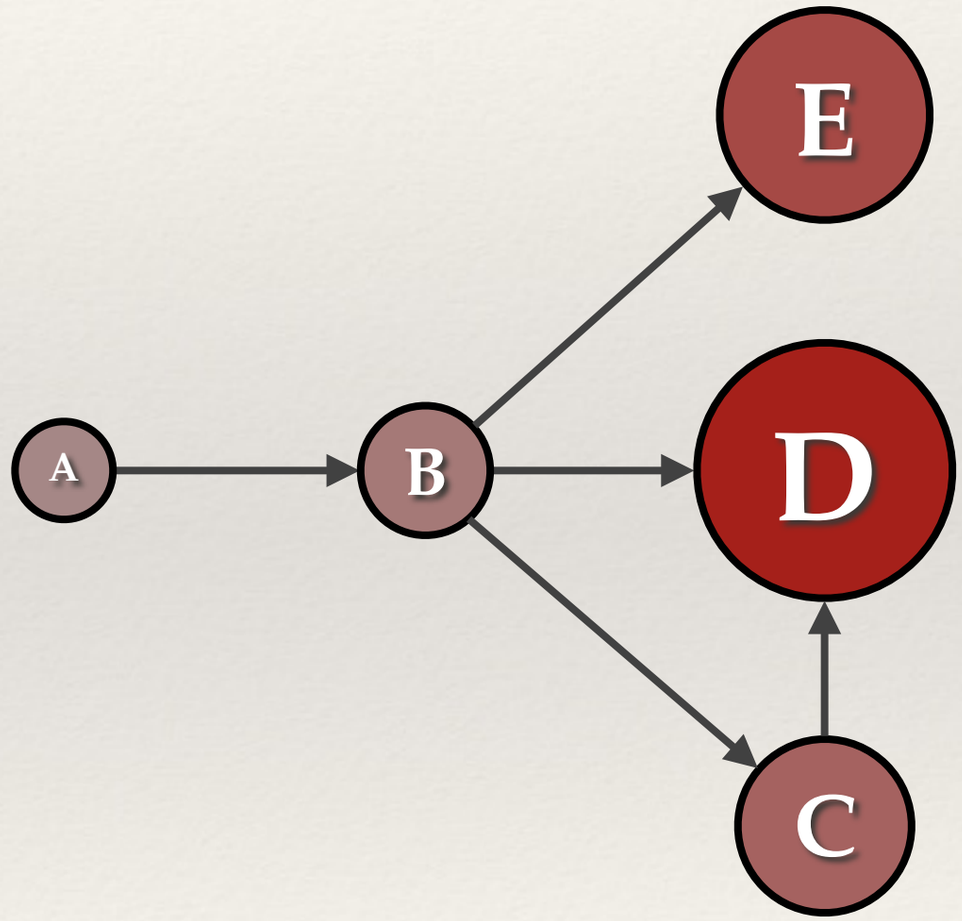
Text analysis

Find keyWords

Recommender systems

booking.com

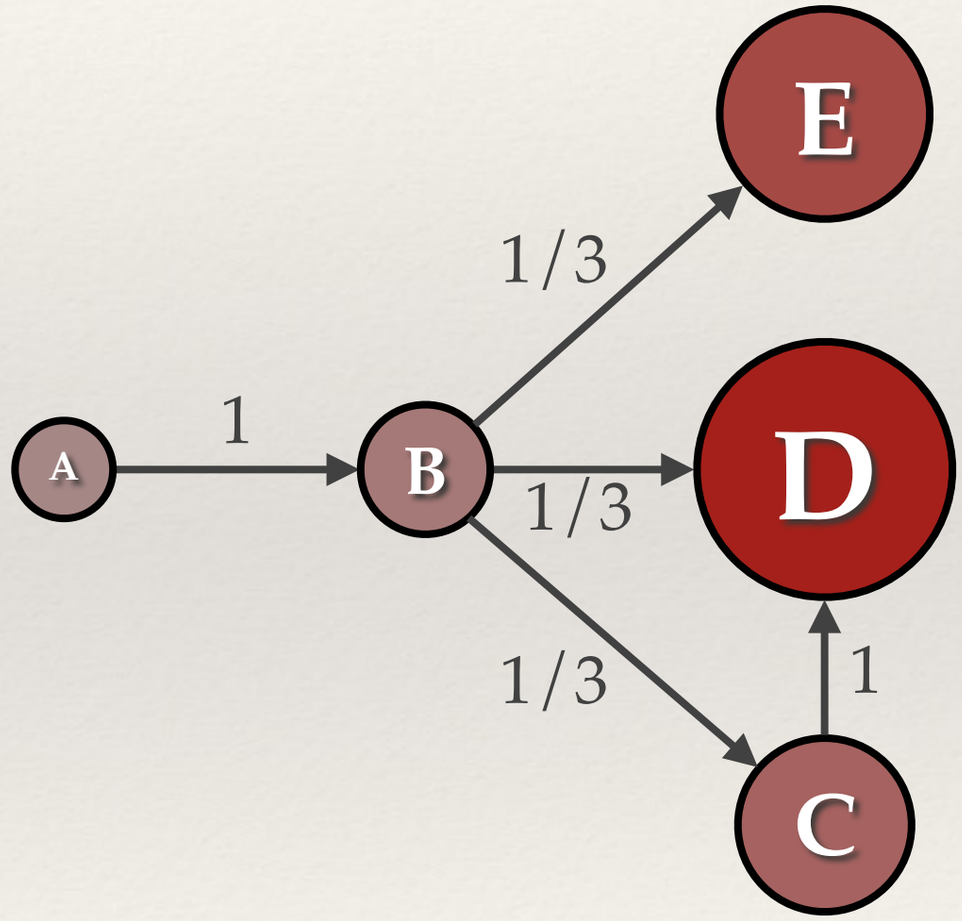
PageRank



PageRank

Normalized Adjacency Matrix

$$P_{ij} = \frac{1}{\mathrm{d}_{\mathrm{out}}(j)}, \quad (j,i) \in G$$



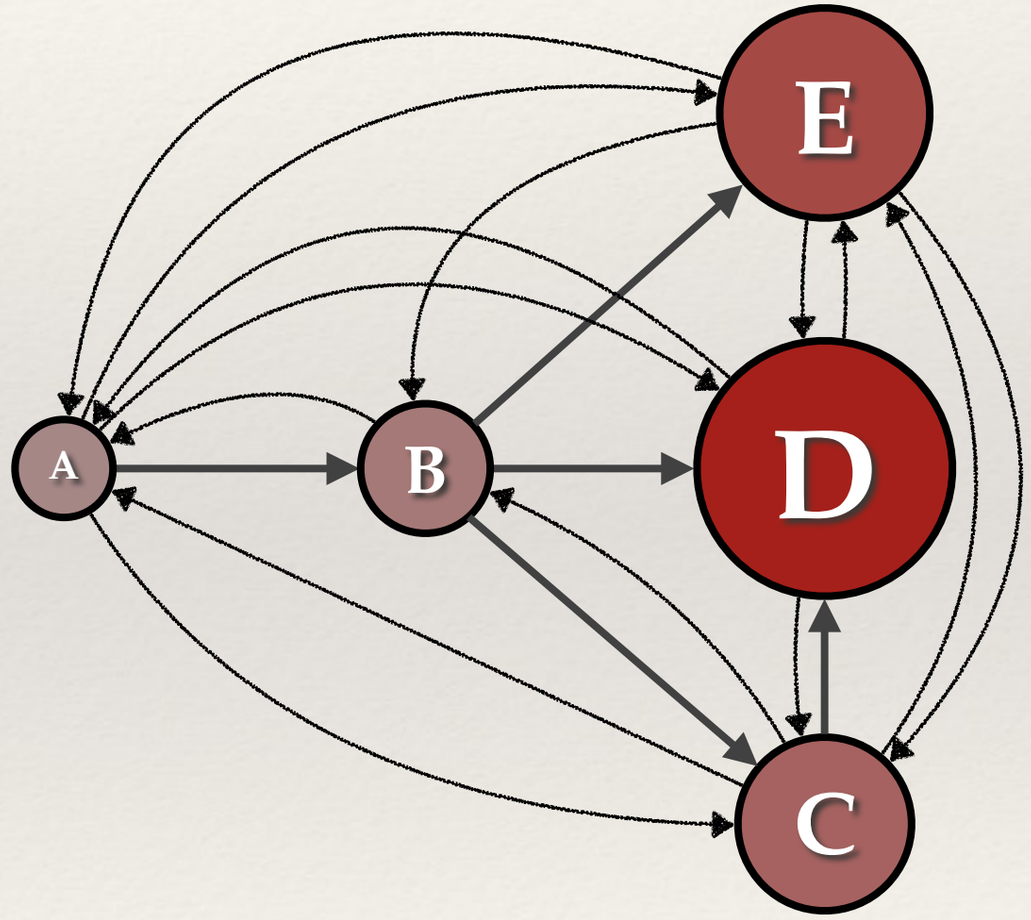
PageRank

Normalized Adjacency Matrix

$$P_{ij} = \frac{1}{\mathrm{d}_{\mathrm{out}}(j)}, \quad (j,i) \in G$$

Augmented Matrix

$$Q_{ij} = (1 - p_T)P_{ij} + \frac{p_T}{n}$$



PageRank

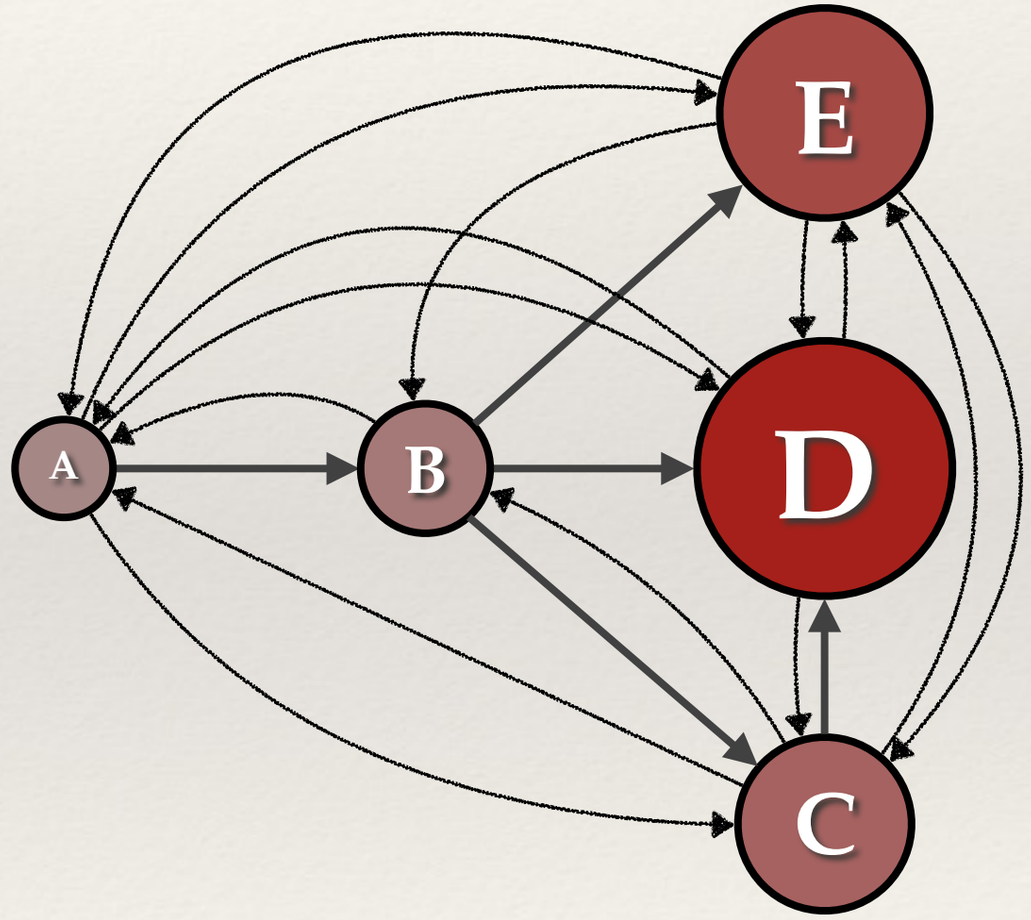
Normalized Adjacency Matrix

$$P_{ij} = \frac{1}{\mathrm{d}_{\mathrm{out}}(j)}, \quad (j,i) \in G$$

Augmented Matrix

$$Q_{ij} = (1 - p_T)P_{ij} + \frac{p_T}{n}$$

PageRank Vector $\pi = Q\pi$



PageRank

Normalized Adjacency Matrix

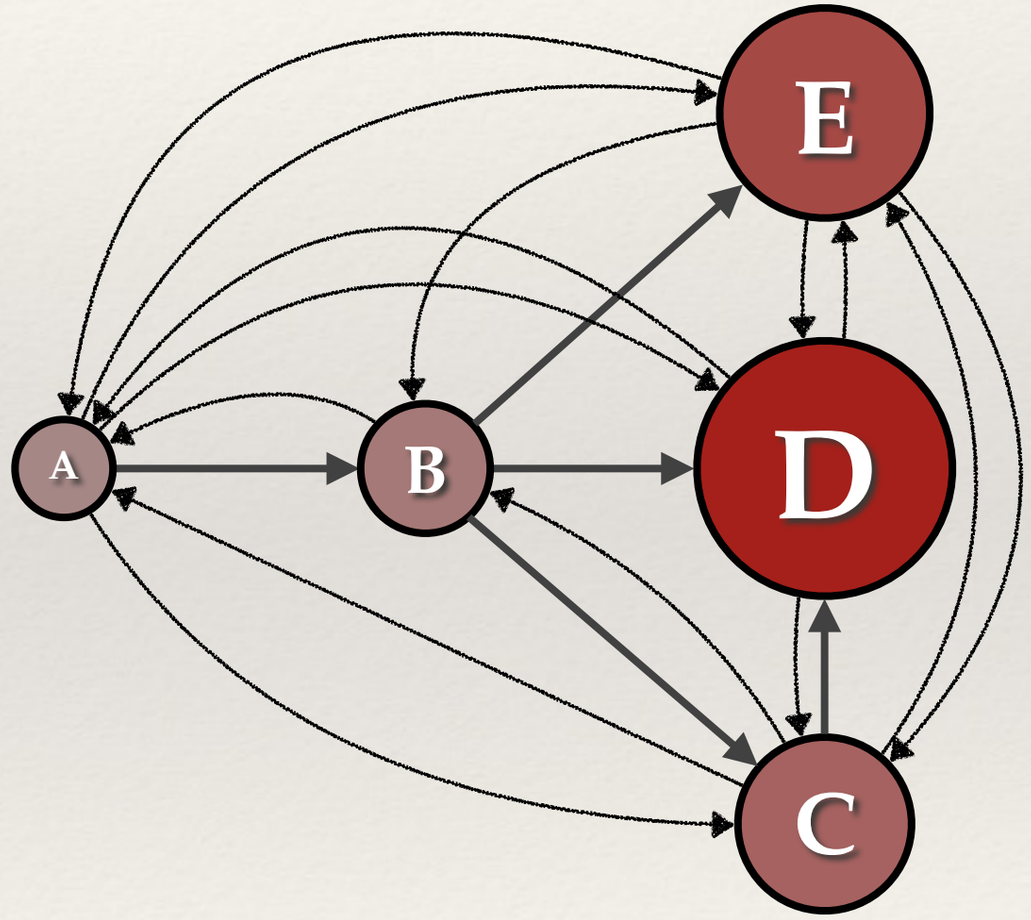
$$P_{ij} = \frac{1}{\mathrm{d}_{\mathrm{out}}(j)}, \quad (j,i) \in G$$

Augmented Matrix

$$Q_{ij} = (1 - p_T)P_{ij} + \frac{p_T}{n}$$

PageRank Vector $\pi = Q\pi$

Power Method $Q^t p^0 \to \pi$



Why Not Precompute?

Why Not Precompute?

- booking.com user asks: find hotels, within 10 miles from Paris center, with availability on these dates

Why Not Precompute?

- booking.com user asks: find hotels, within 10 miles from Paris center, with availability on these dates

- Engine finds 1000 hotels, must recommend 10-20

Why Not Precompute?

- booking.com user asks: find hotels, within 10 miles from Paris center, with availability on these dates
- Engine finds 1000 hotels, must recommend 10-20
- Engine forms a graph where vertices are hotels

Why Not Precompute?

- booking.com user asks: find hotels, within 10 miles from Paris center, with availability on these dates

- Engine finds 1000 hotels, must recommend 10-20

- Engine forms a graph where vertices are hotels

- Add edge **i** —> **j** if a previous user has seen both hotels **i** and **j** and preferred **j**

Why Not Precompute?

- booking.com user asks: find hotels, within 10 miles from Paris center, with availability on these dates
- Engine finds 1000 hotels, must recommend 10-20
- Engine forms a graph where vertices are hotels
- Add edge **i —> j** if a previous user has seen both hotels **i** and **j** and preferred **j**
- Show 10-20 hotels with highest pagerank to user

Why Not Precompute?

- booking.com user asks: find hotels, within 10 miles from Paris center, with availability on these dates

- Engine finds 1000 hotels, must recommend 10-20

- Engine forms a graph where vertices are hotels

- Add edge **i —> j** if a previous user has seen both hotels **i** and **j** and preferred **j**

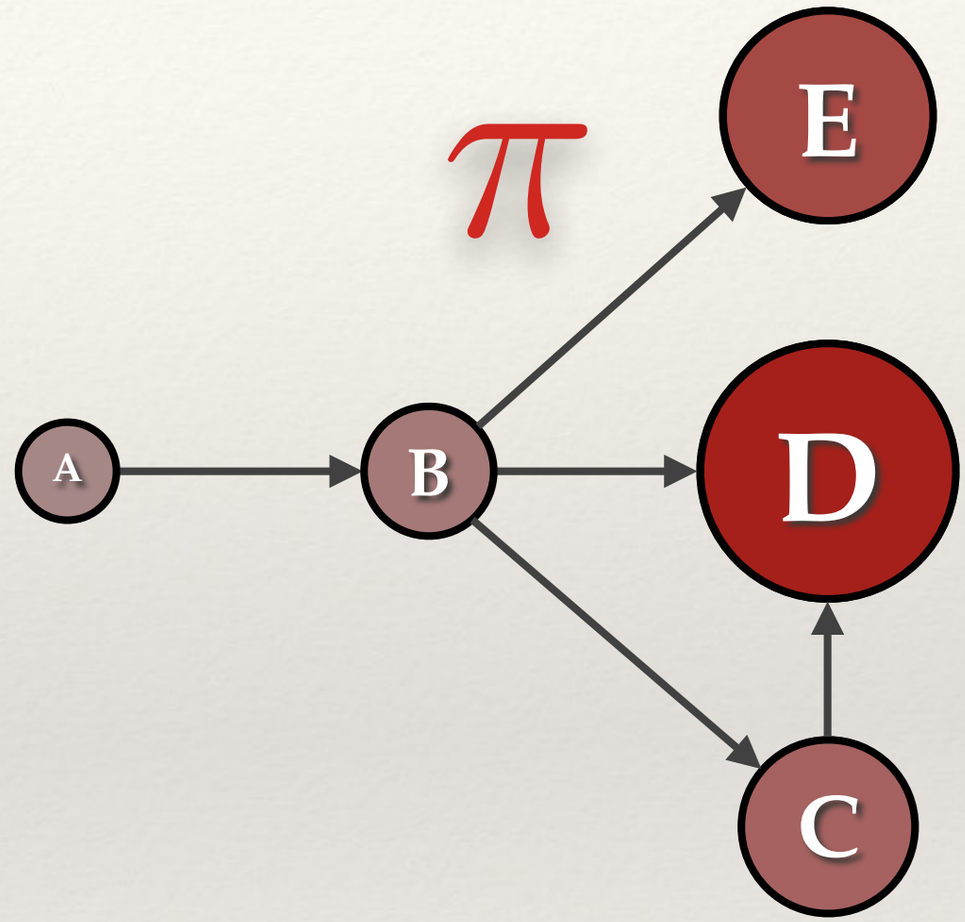
- Show 10-20 hotels with highest pagerank to user

- There are many applications where **graphs are formed at query-time** and high pagerank vertices must be discovered quickly

Top-k PageRank

Looking for k "heavy nodes"

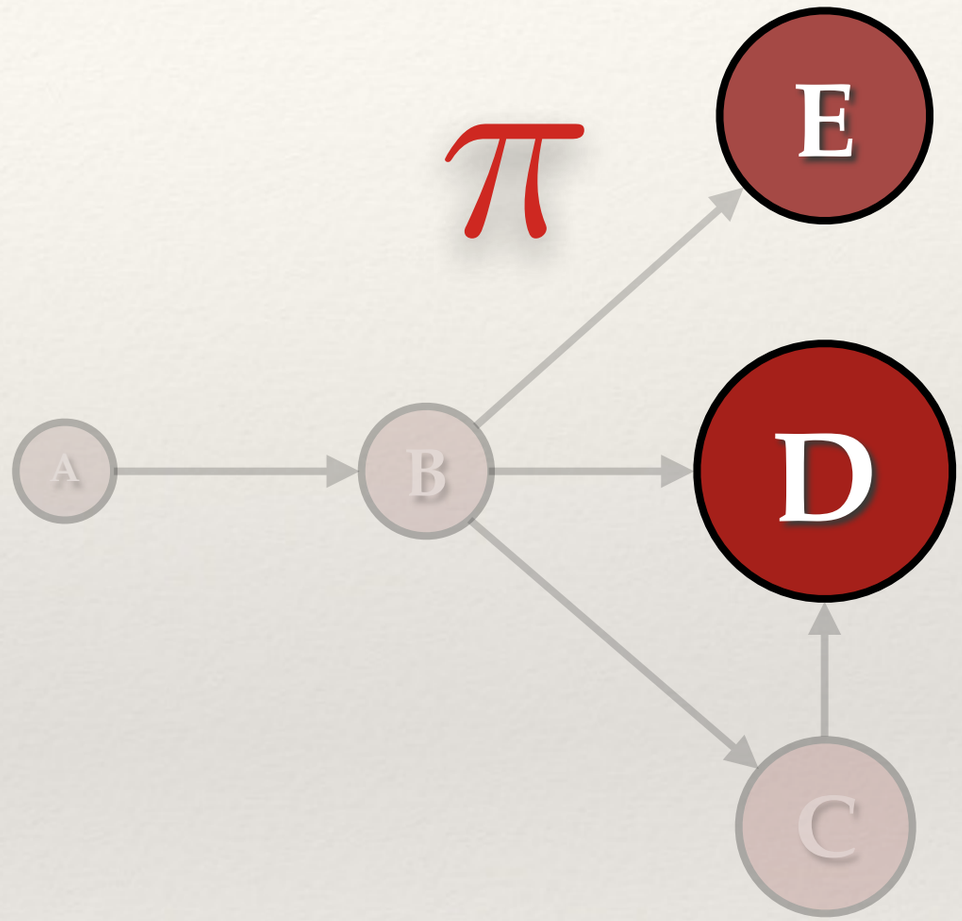
Do not need full PageRank vector



Top-k PageRank

Looking for k "heavy nodes"

Do not need full PageRank vector

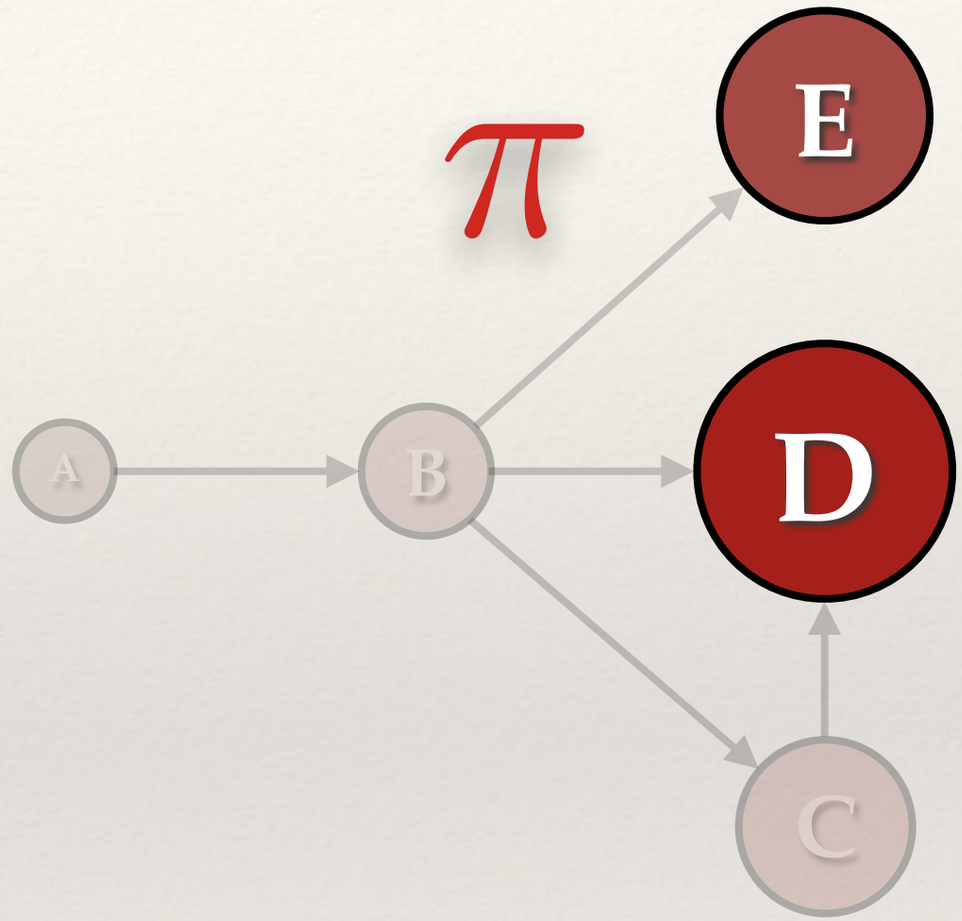


Top-k PageRank

Looking for k "heavy nodes"

Do not need full PageRank vector

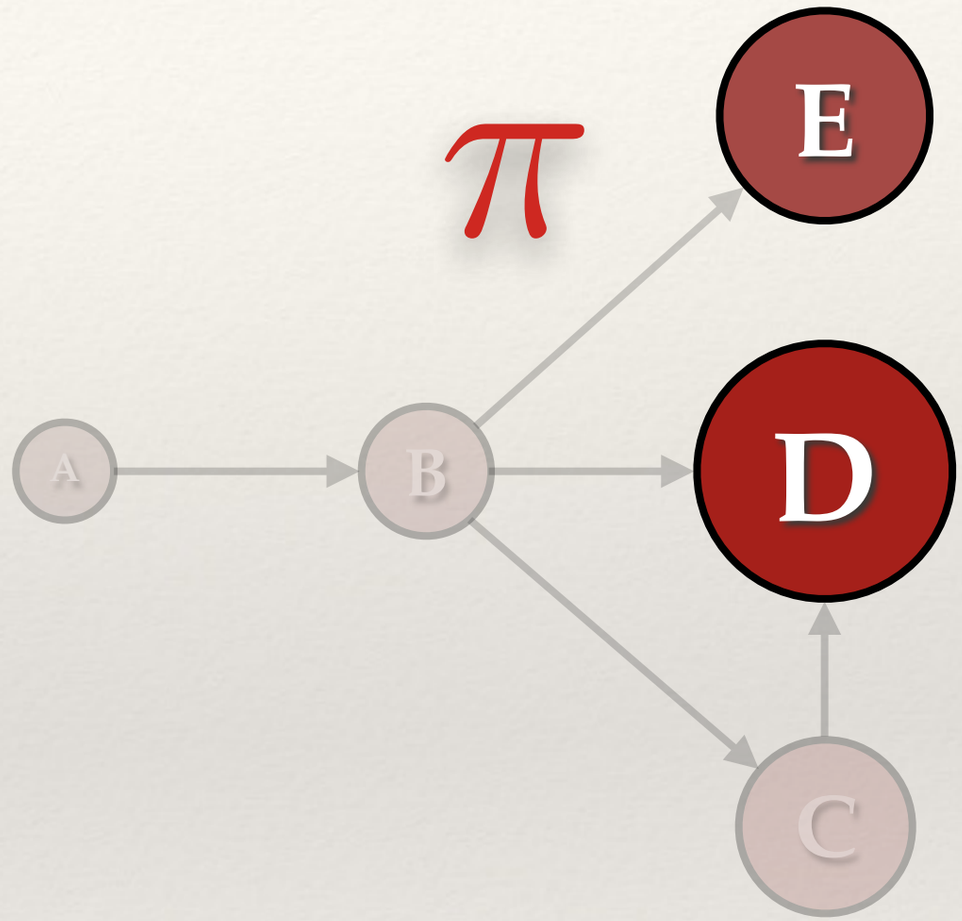
Random Walk Sampling



Top-k PageRank

Looking for k "heavy nodes"

Do not need full PageRank vector



Random Walk Sampling

Captured Mass Metric

Our algorithm ranks k nodes — S_k

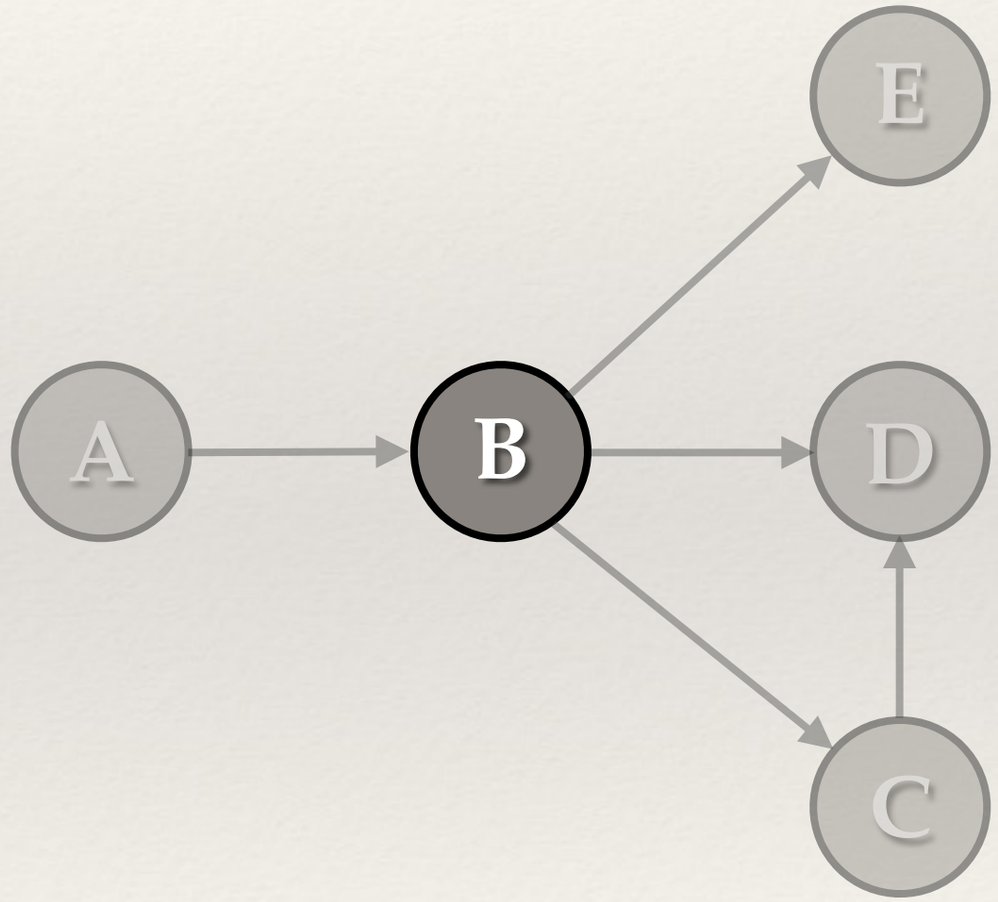
We compare sum of their ranks — $\pi(S_k)$

with the true Top-k PageRank sum

Platform - Graph Engine

- Engine splits graph across cluster
- **Vertex program** describes logic

GAS abstraction

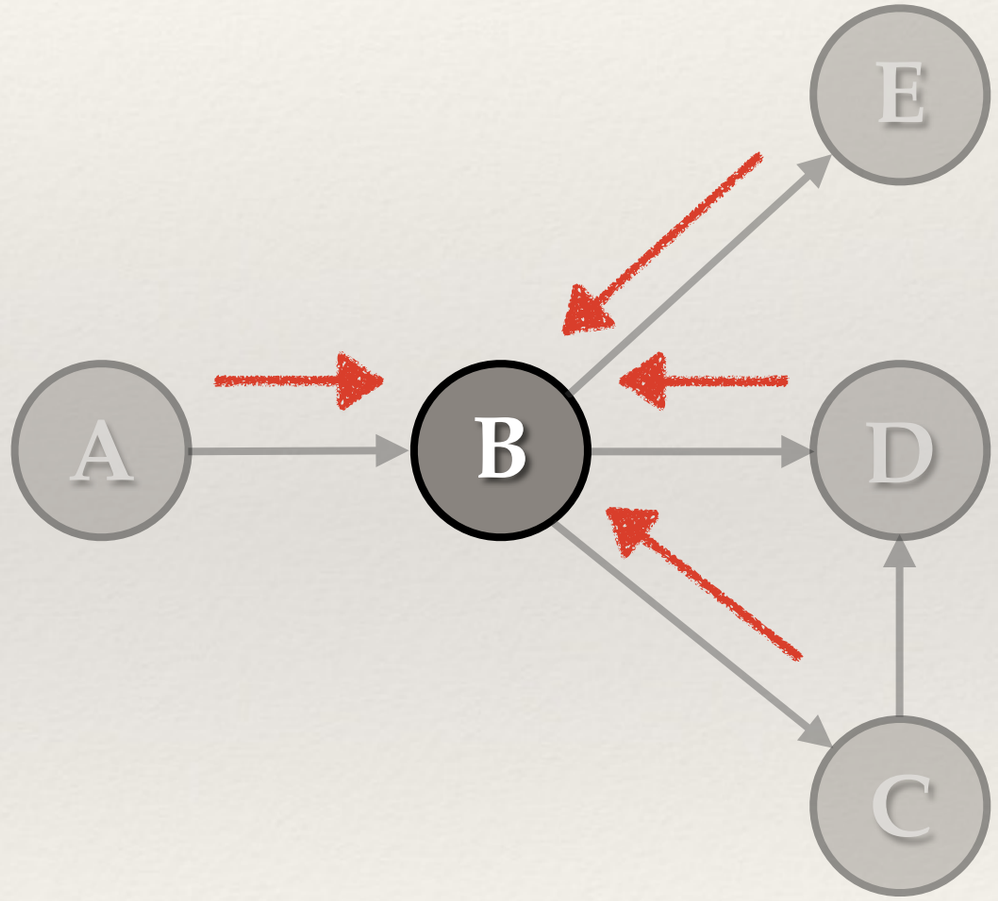


Platform - Graph Engine

- Engine splits graph across cluster
- **Vertex program** describes logic

GAS abstraction

1. Gather

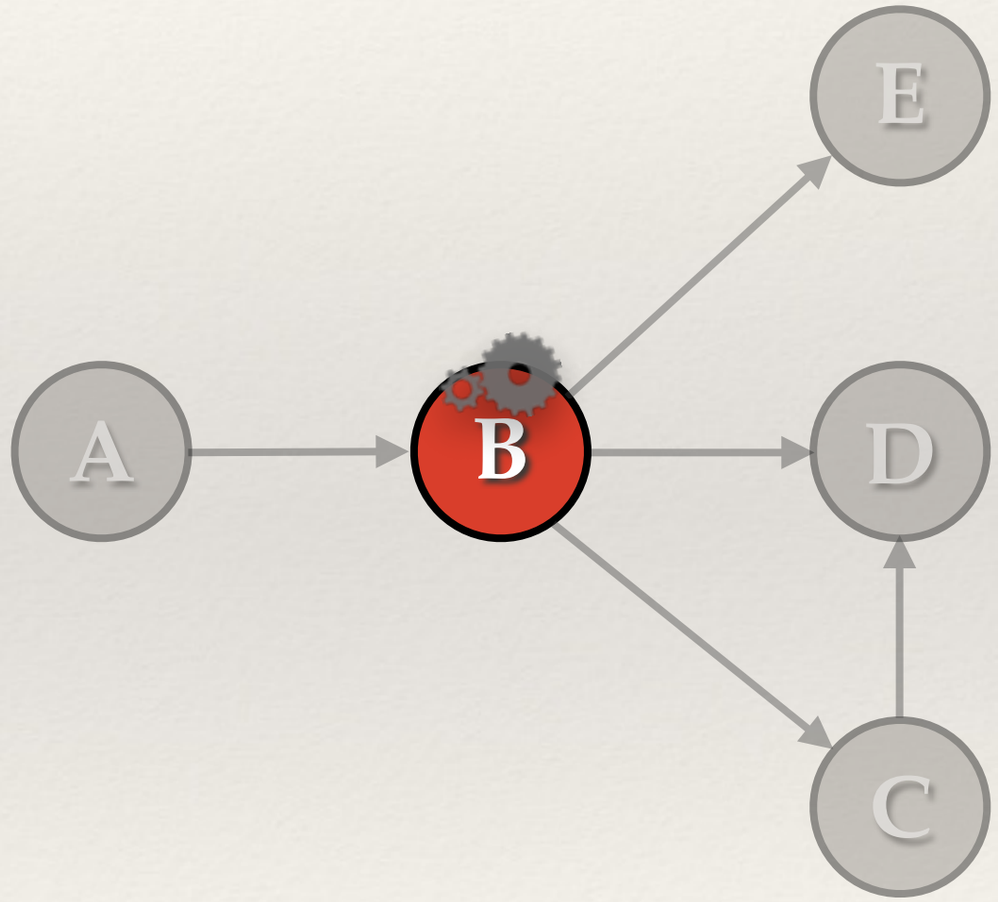


Platform - Graph Engine

- Engine splits graph across cluster
- **Vertex program** describes logic

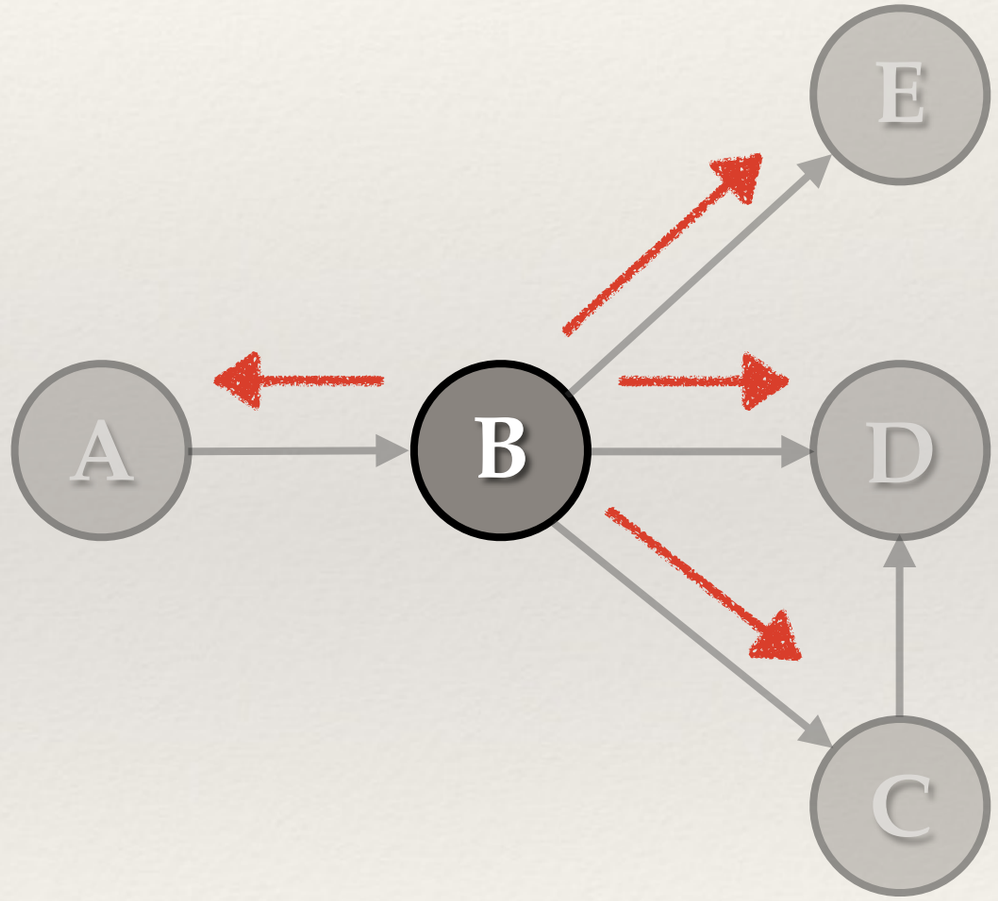
GAS abstraction

1. Gather
2. Apply



Platform - Graph Engine

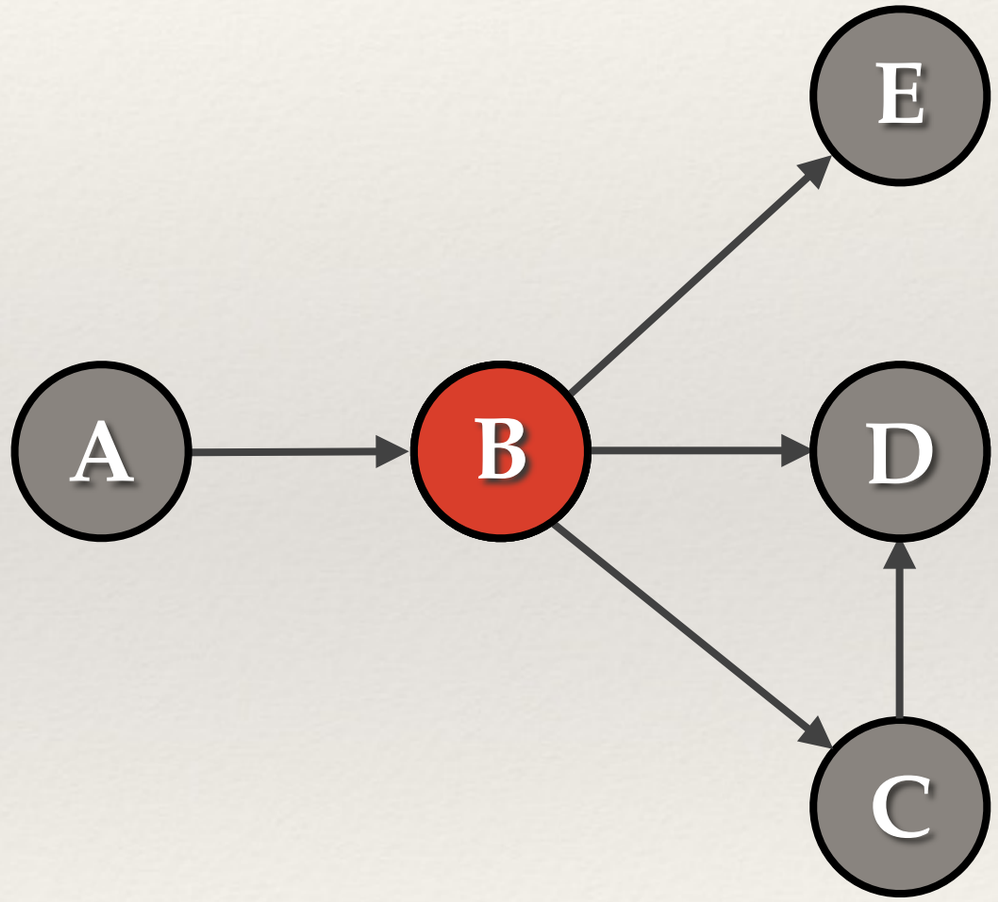
- Engine splits graph across cluster
- **Vertex program** describes logic



GAS abstraction

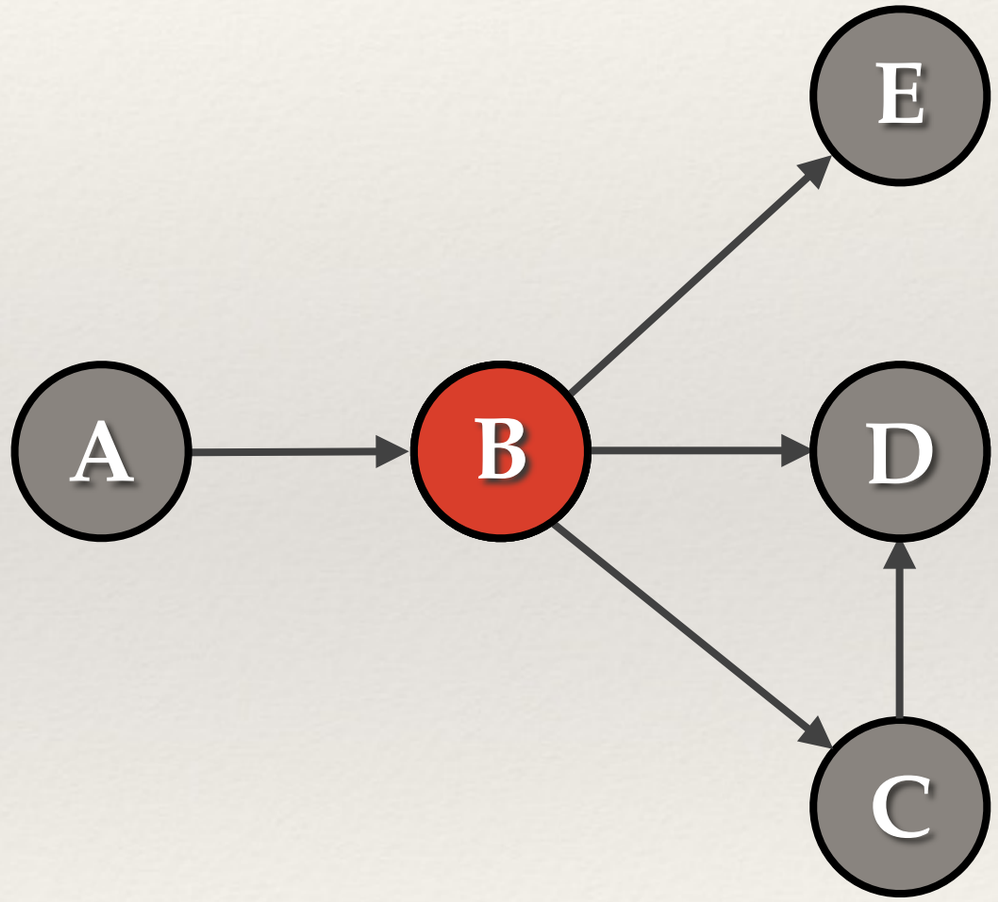
1. Gather
2. Apply
3. Scatter

Vertex Cuts (GraphLab 2.0 - PowerGraph)



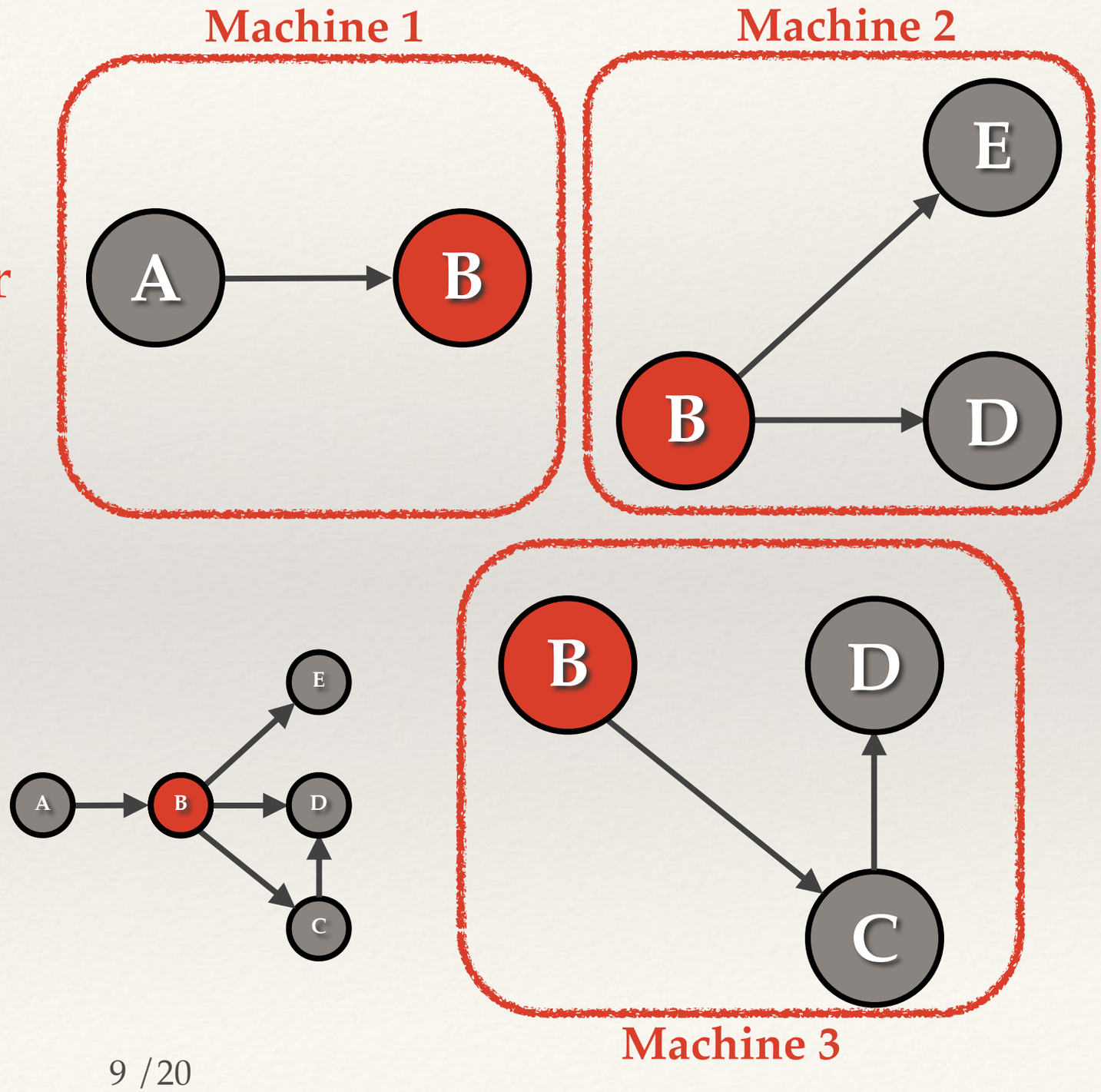
Vertex Cuts (GraphLab 2.0 - PowerGraph)

- Assign **edges** to machines



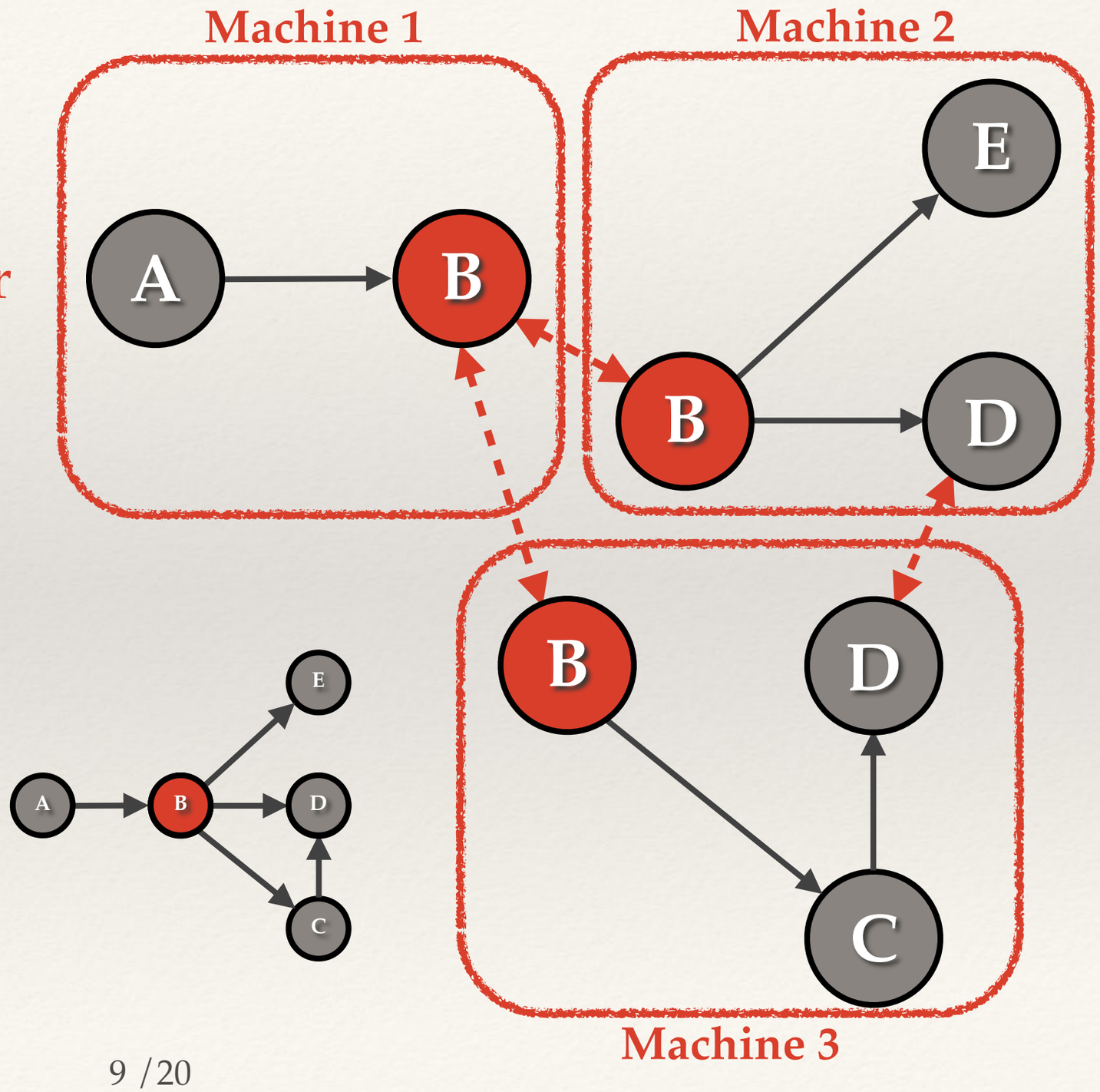
Vertex Cuts (GraphLab 2.0 - PowerGraph)

- Assign **edges** to machines
- High-degree nodes replicated
- One replica designated master



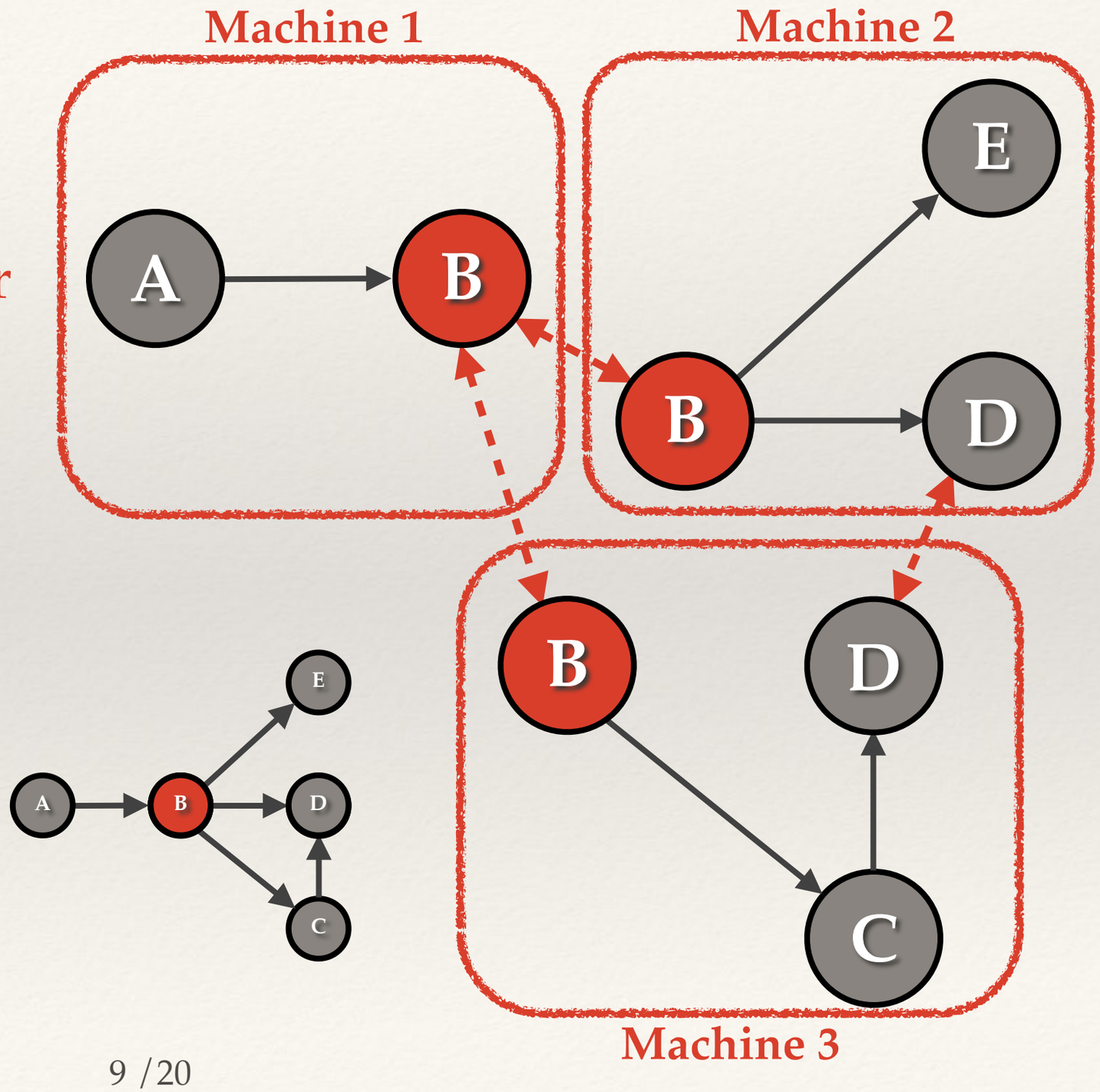
Vertex Cuts (GraphLab 2.0 - PowerGraph)

- Assign **edges** to machines
- High-degree nodes replicated
- One replica designated master
- Need for synchronization
  1. Gather
  2. Apply [on master]
  3. Synchronize mirrors
  4. Scatter



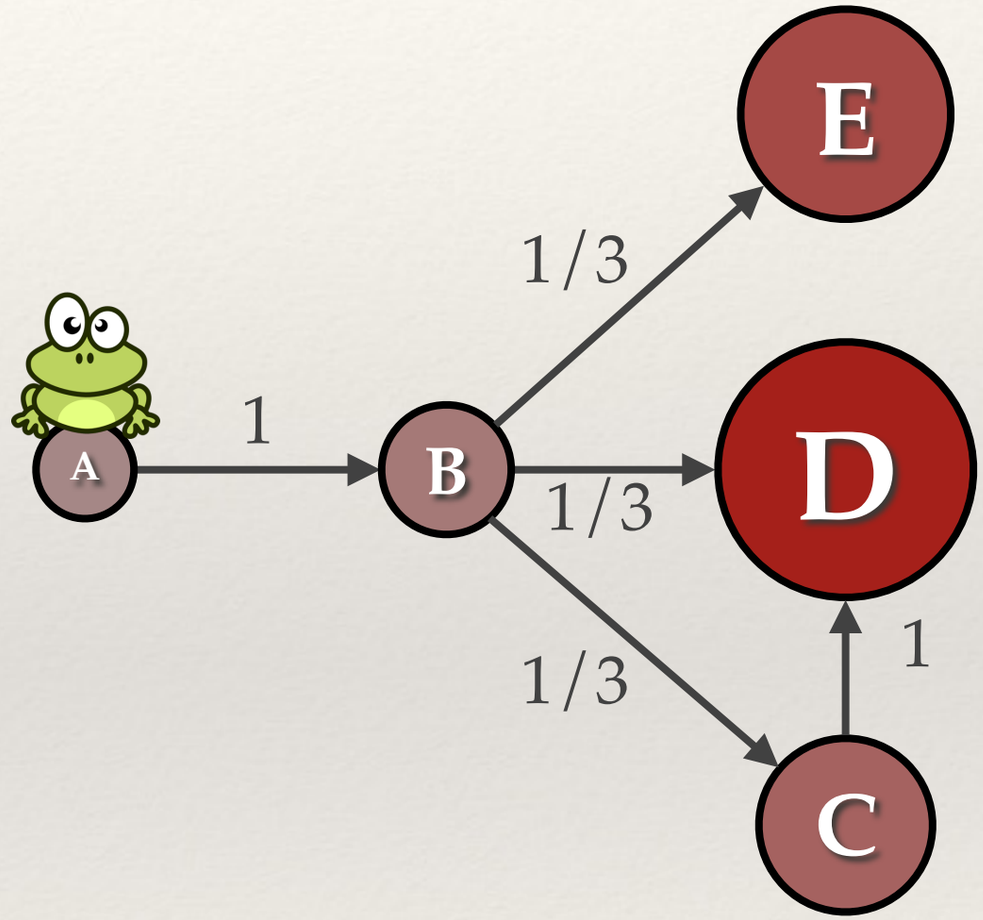
Vertex Cuts (GraphLab 2.0 - PowerGraph)

- Assign **edges** to machines
- High-degree nodes replicated
- One replica designated master
- Need for synchronization
  1. Gather
  2. Apply [on master]
  3. Synchronize mirrors
  4. Scatter
- Network is bottleneck



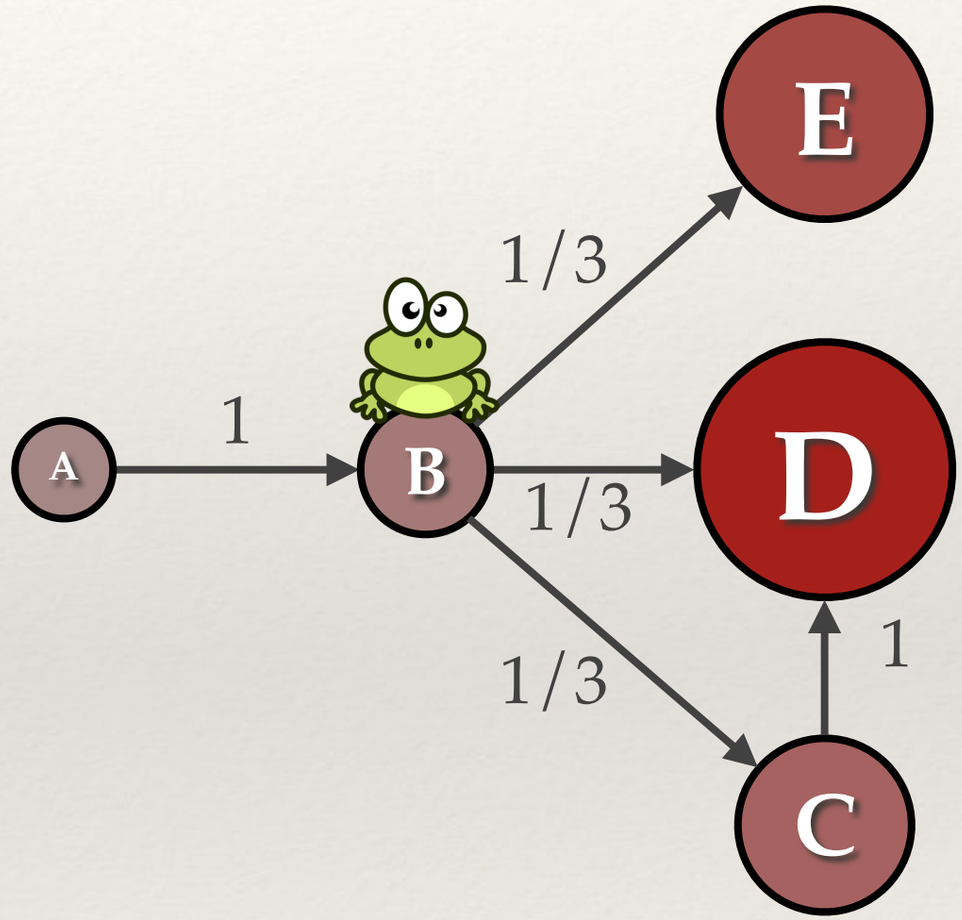
Random Walks

Frog walks randomly on graph



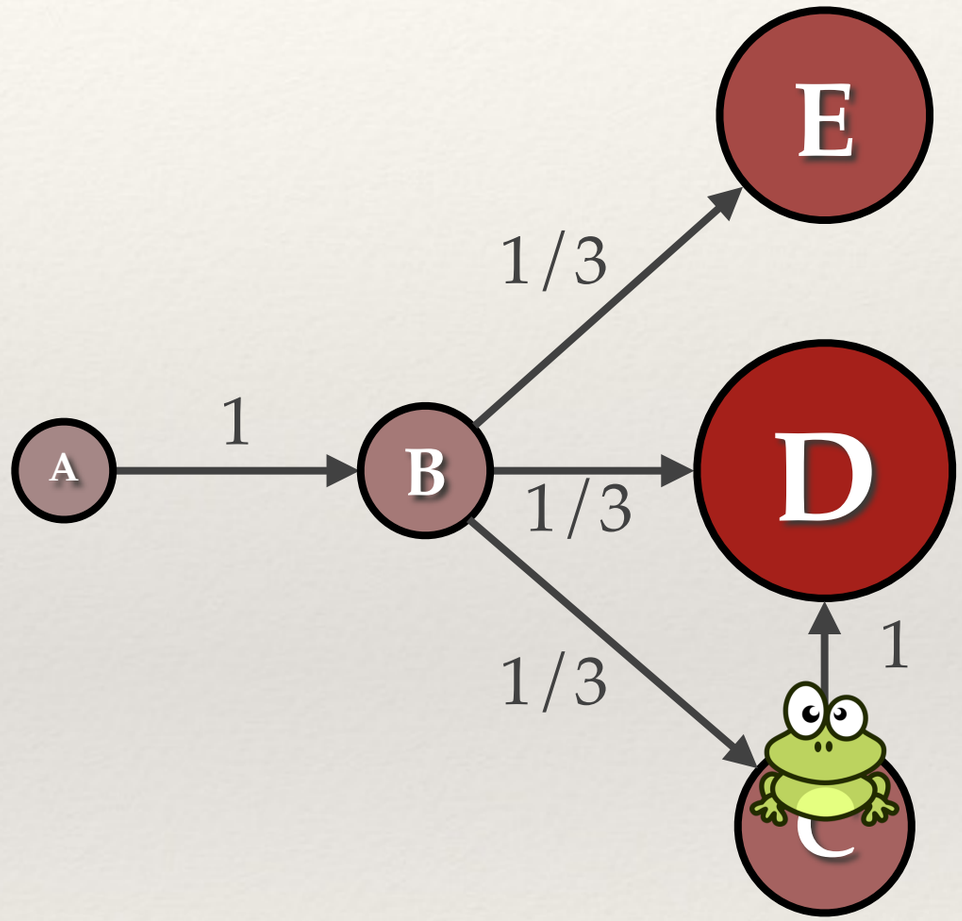
Random Walks

Frog walks randomly on graph



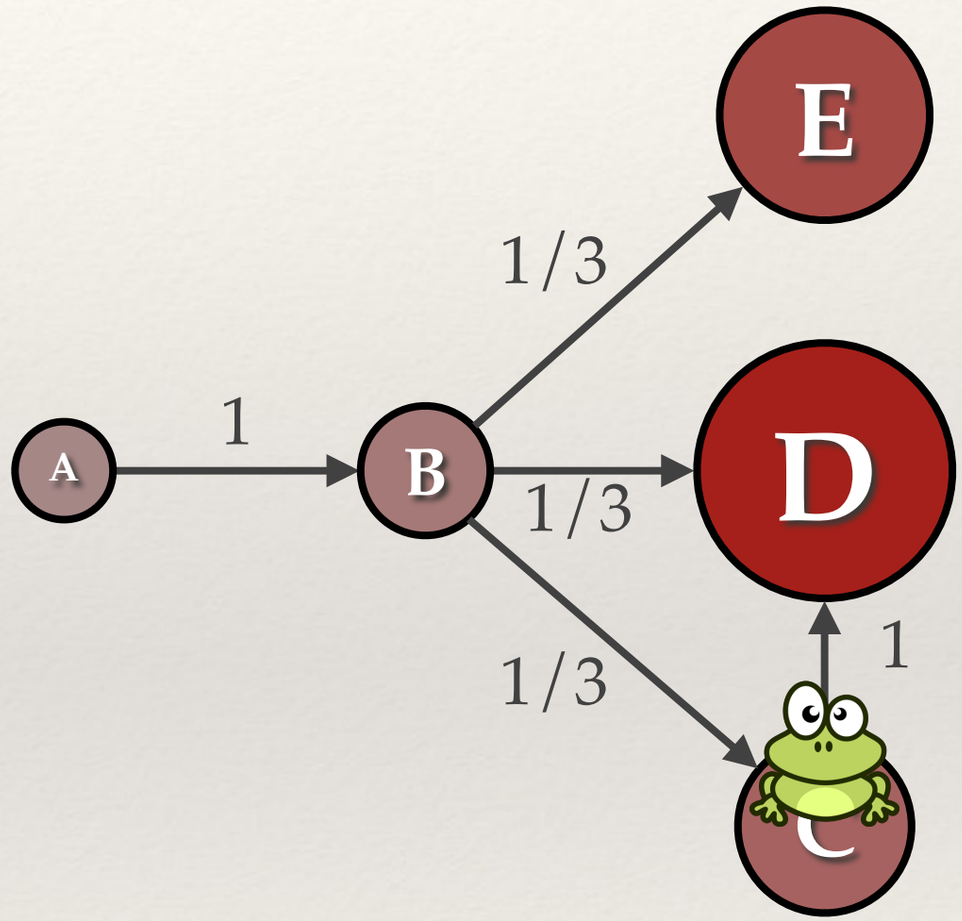
Random Walks

Frog walks randomly on graph



Random Walks

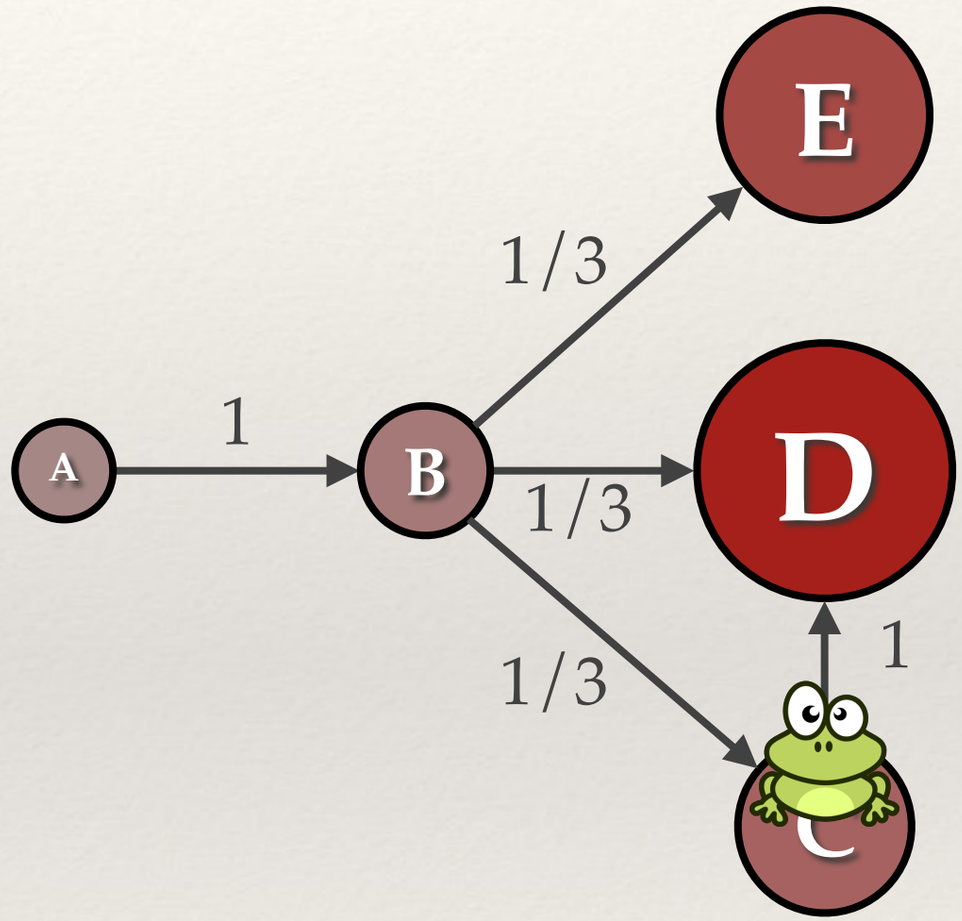
Frog walks randomly on graph



Random Walks

Frog walks randomly on graph

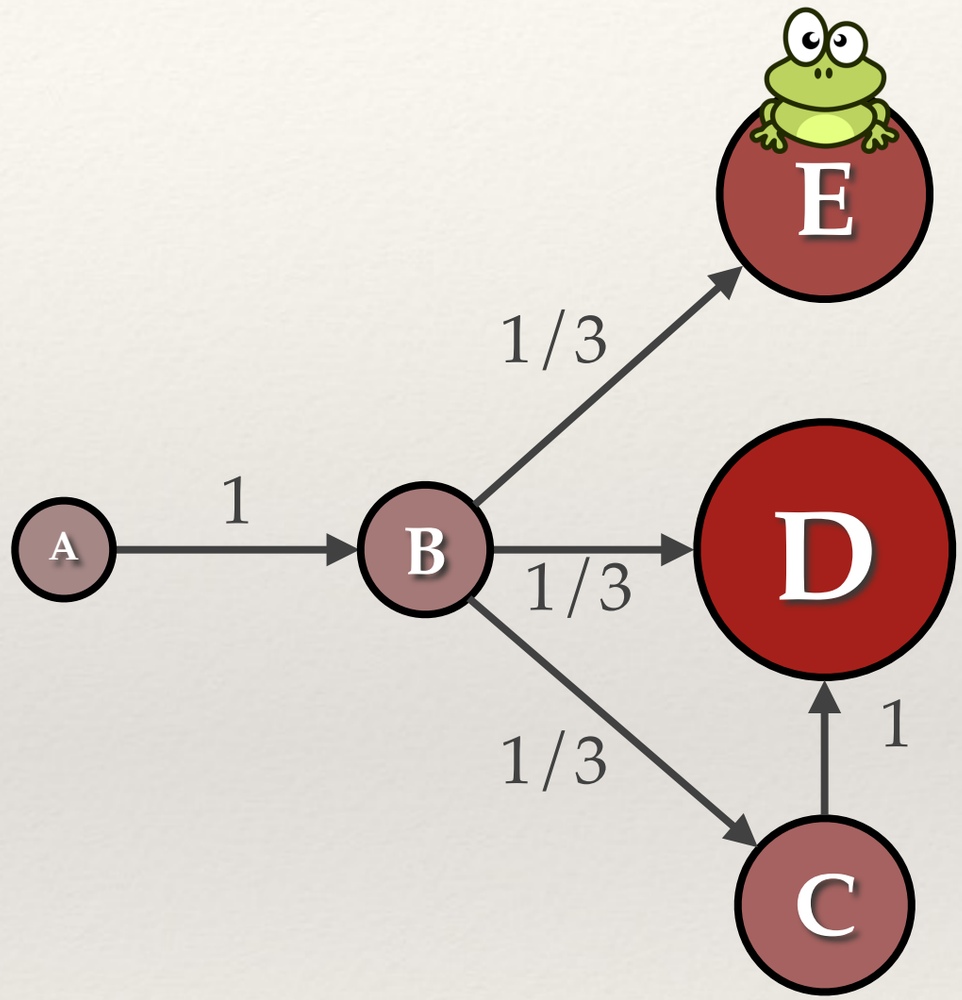
Teleportation w.p. p_T



Random Walks

Frog walks randomly on graph

Teleportation w.p. p_T



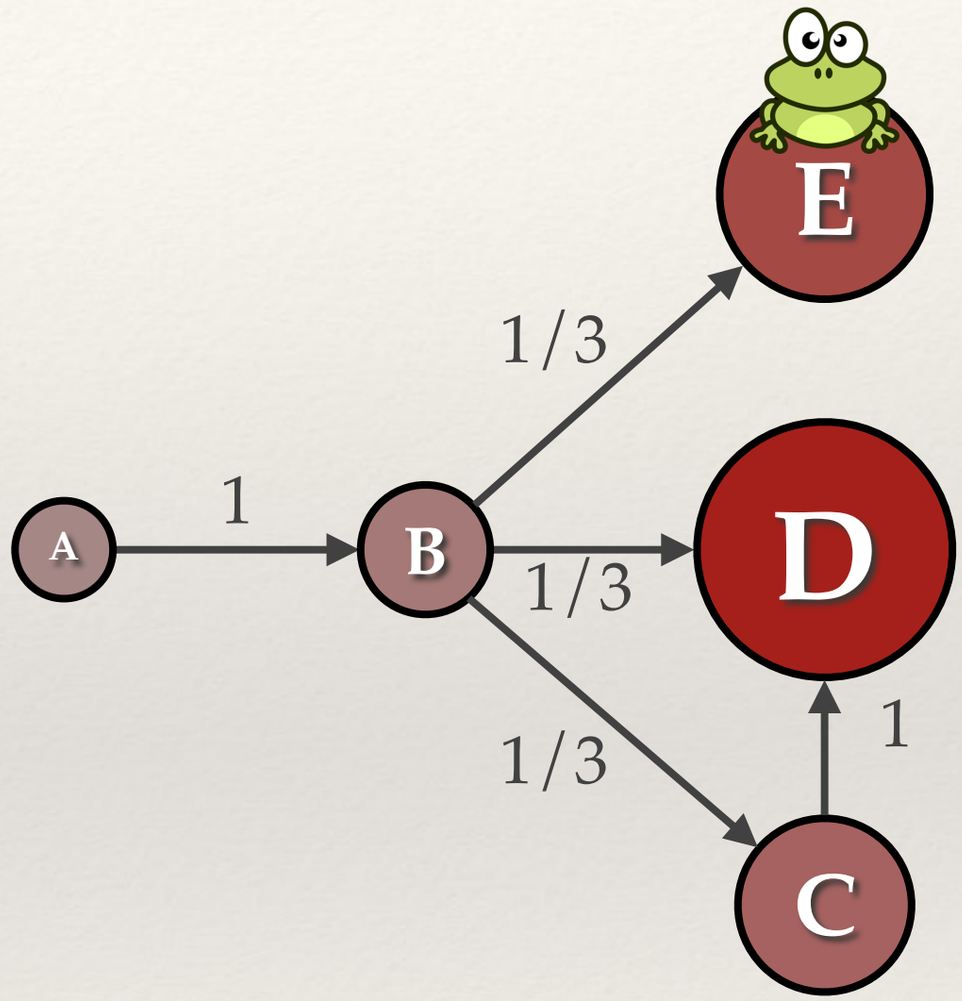
Random Walks

Frog walks randomly on graph

Teleportation w.p. p_T

Sampling after t steps

Frog location gives sample from π



Random Walks

Frog walks randomly on graph

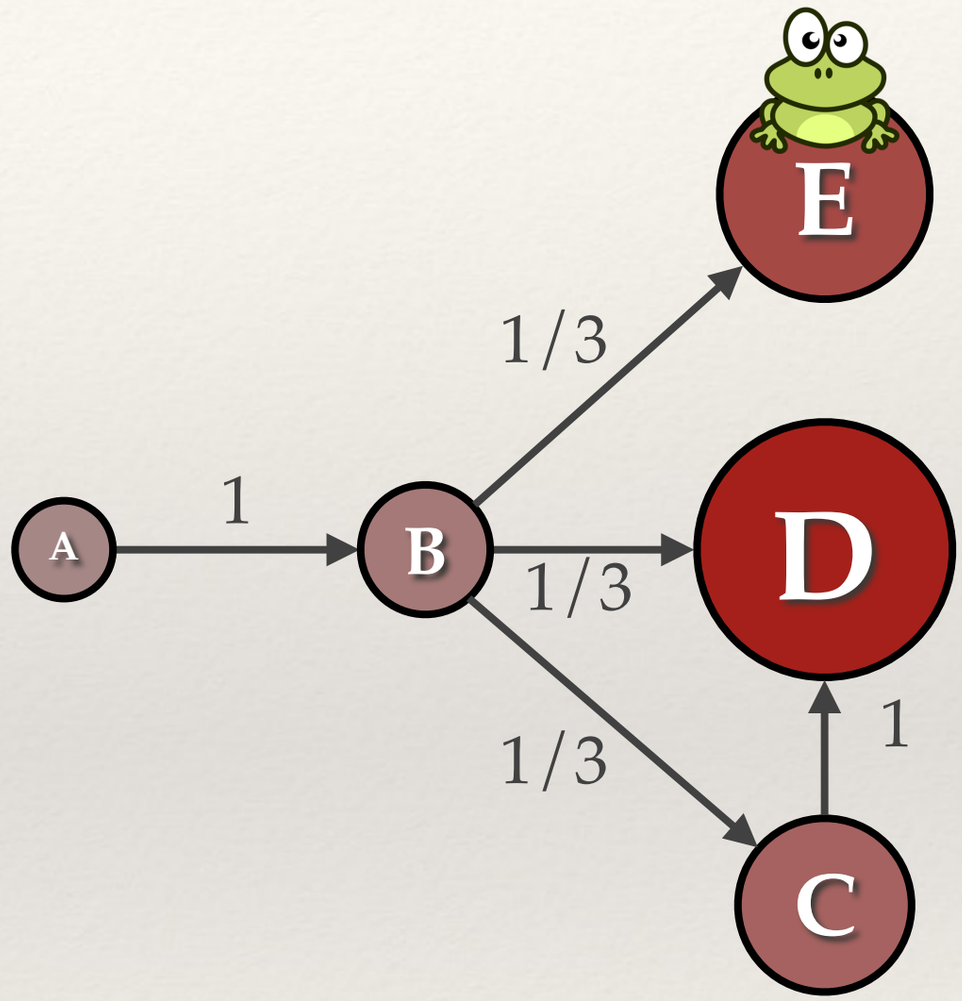
Teleportation w.p. p_T

Sampling after t steps

Frog location gives sample from π

PageRank Vector

Many frogs, estimate vector π



Random Walks

Frog walks randomly on graph

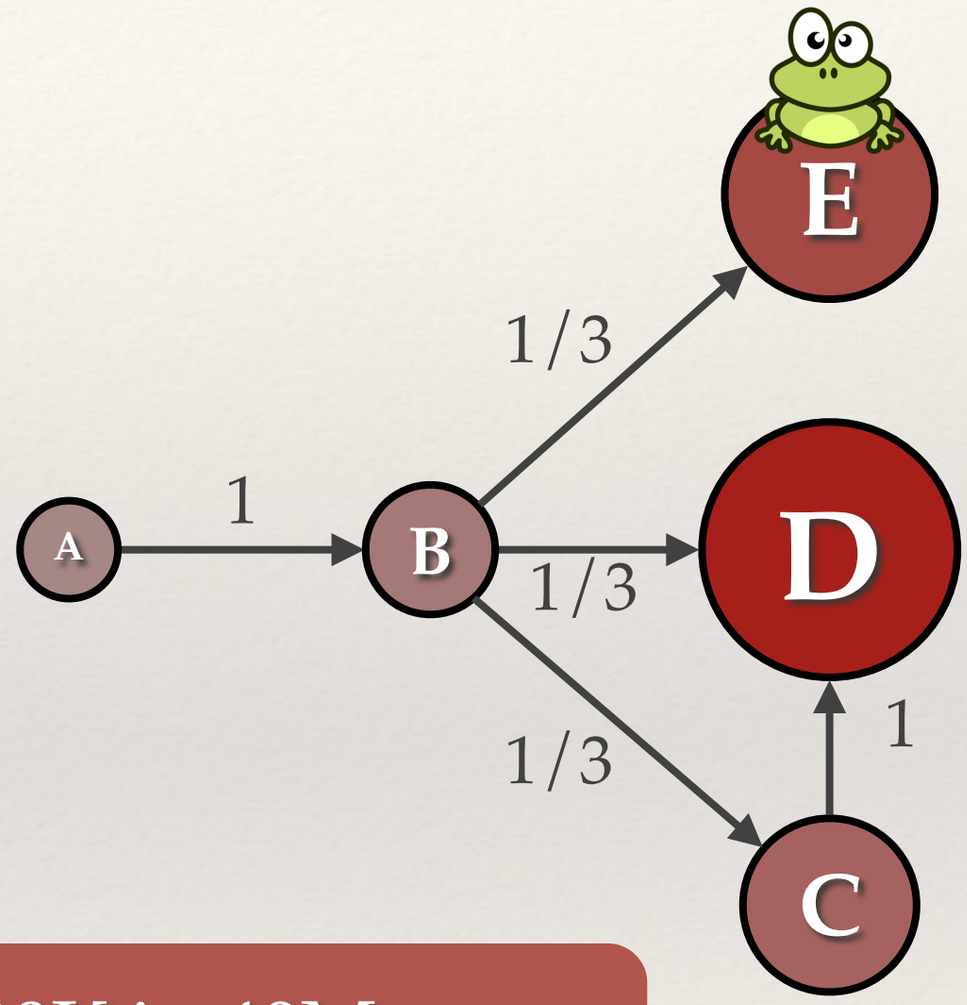
Teleportation w.p. p_T

Sampling after t steps

Frog location gives sample from π

PageRank Vector

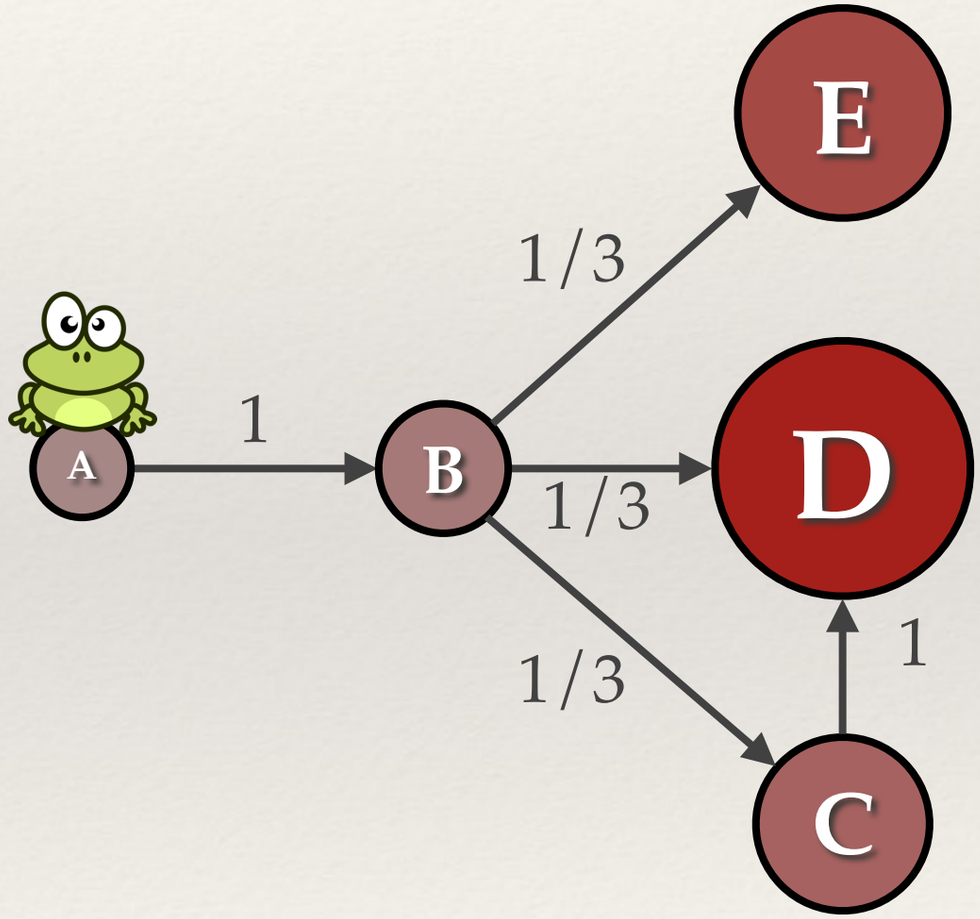
Many frogs, estimate vector π



Lemma: A few random-walkers (e.g. 800K in 40M vertex graph) will suffice to estimate large entries of π vector whp. (mixing time and Chernoff bounds)

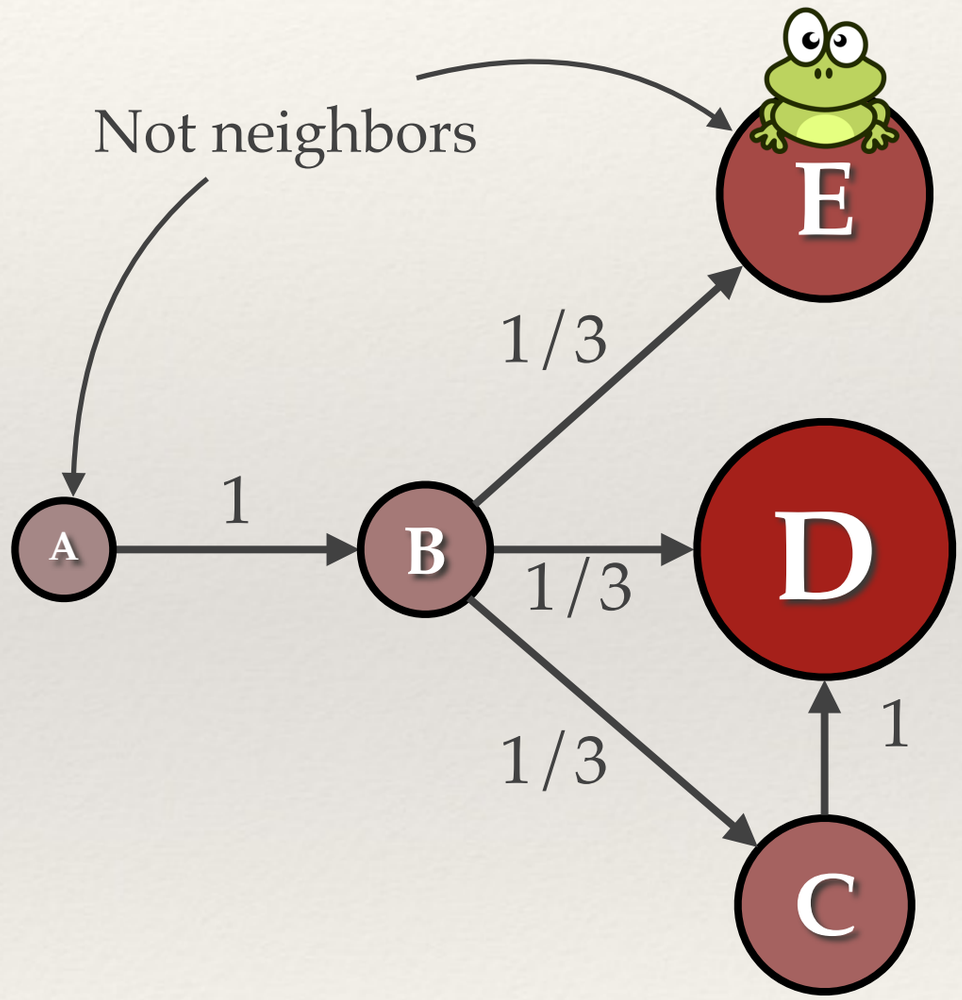
Problem 1: Teleportation

Requires global communication



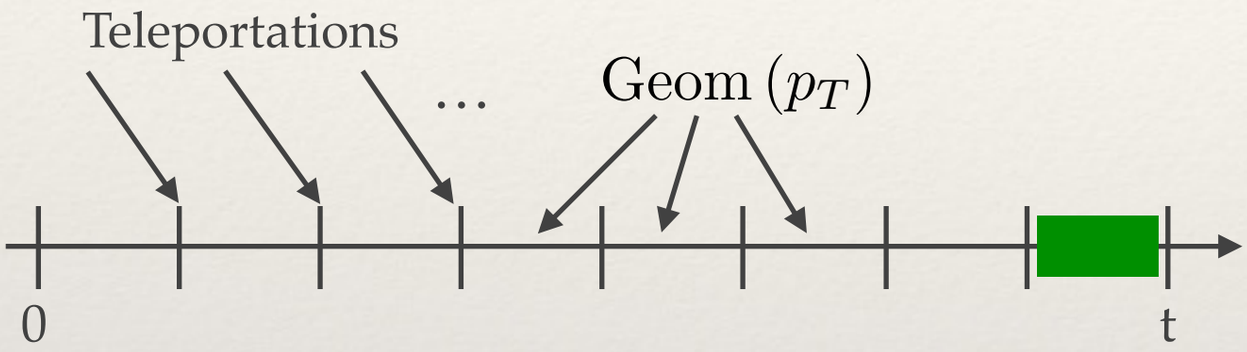
Problem 1: Teleportation

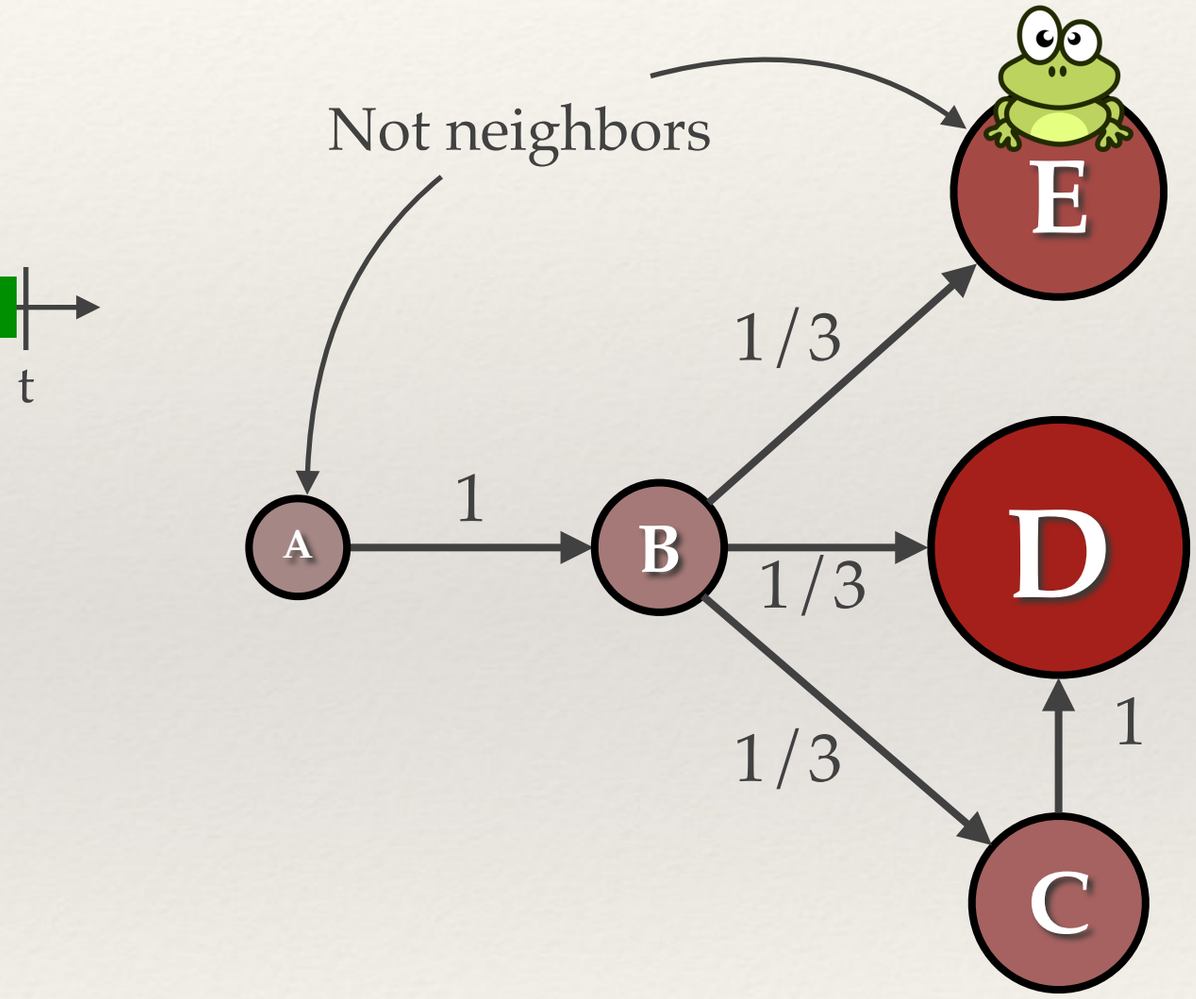
Requires global communication



Problem 1: Teleportation

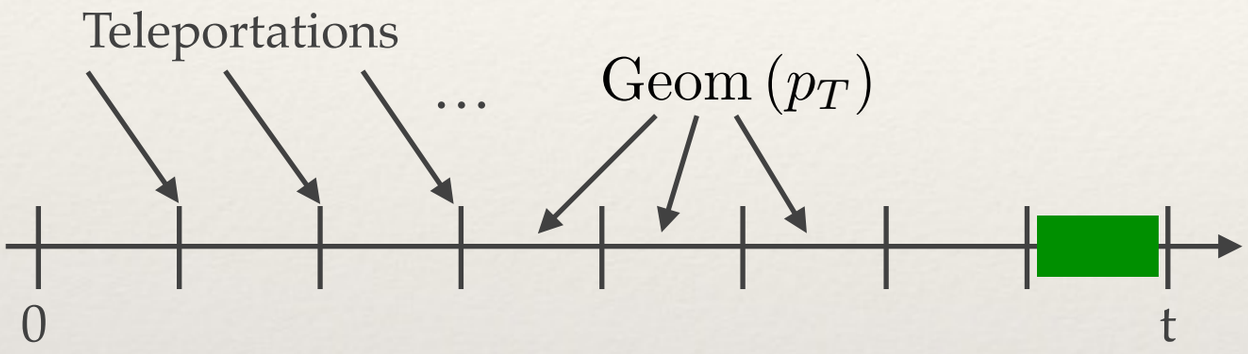
Requires global communication

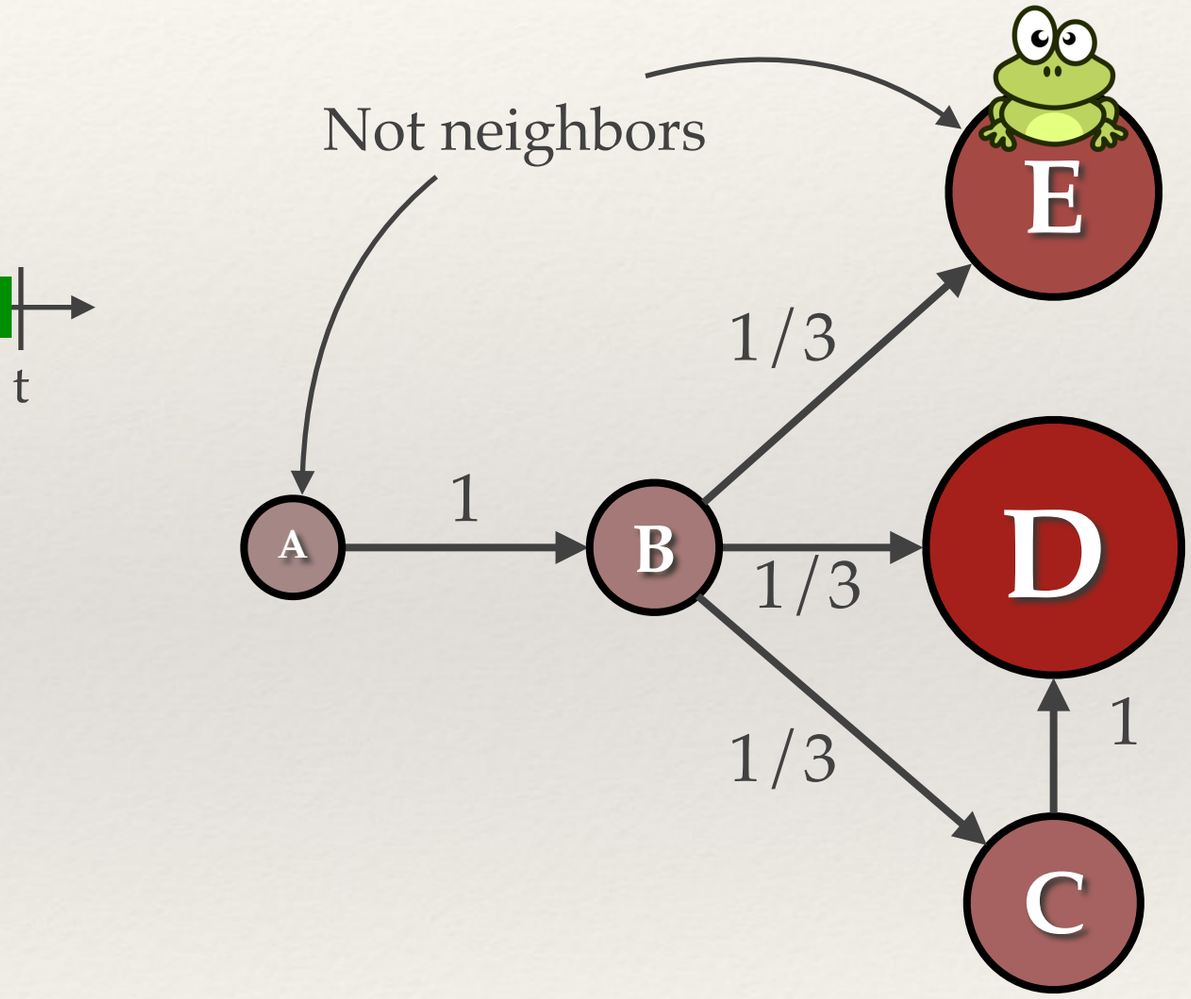




Problem 1: Teleportation

Requires global communication



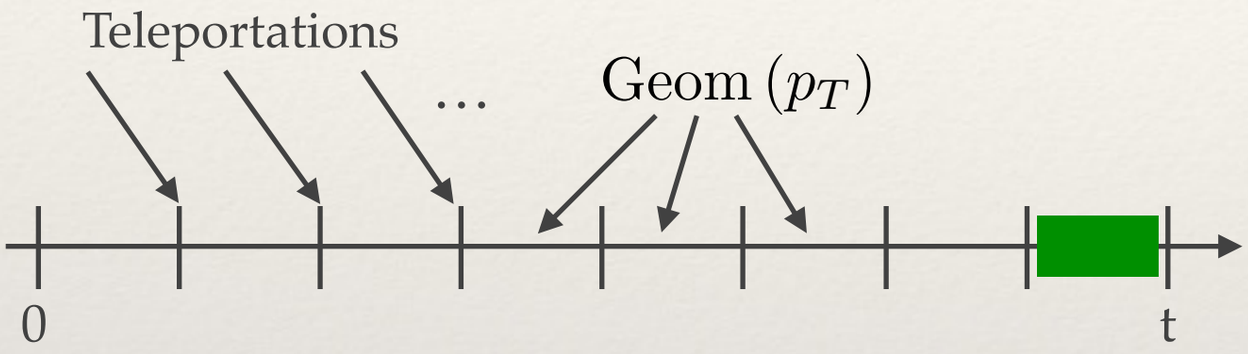


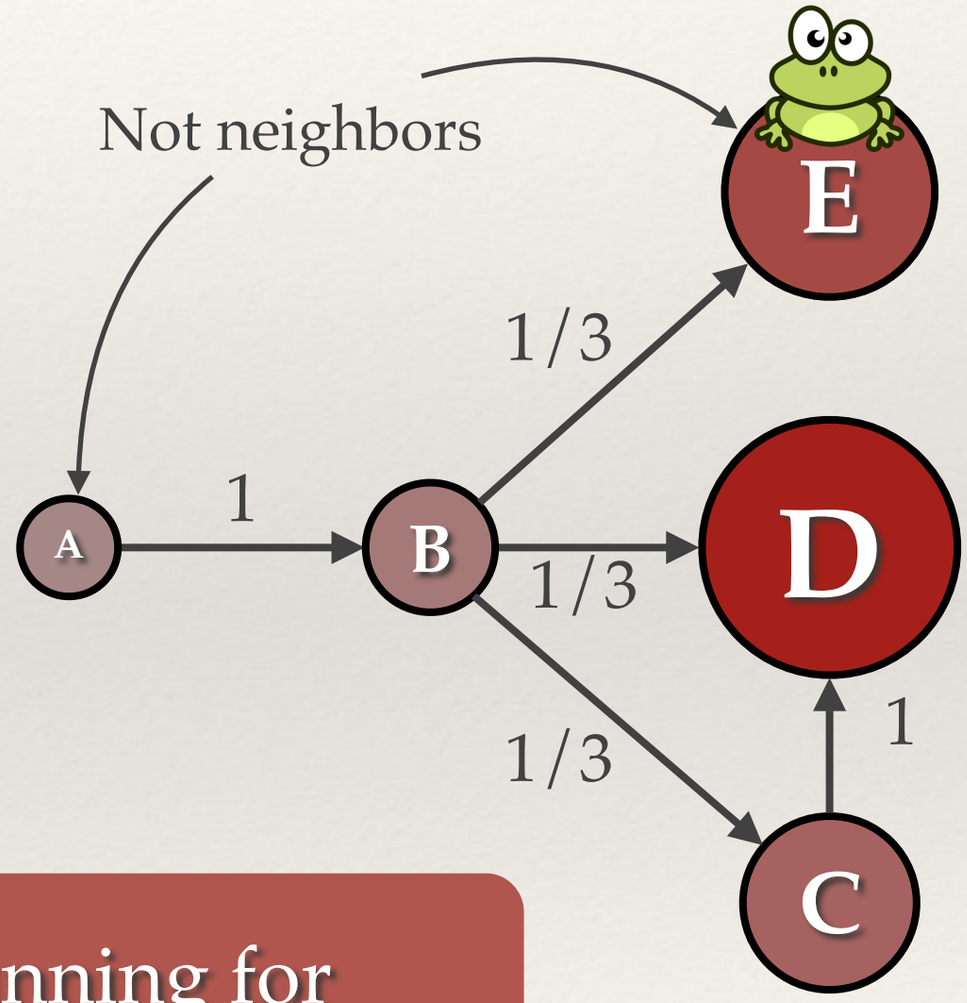
Solution

1. Give them random life
2. Do no teleportation

Problem 1: Teleportation

Requires global communication





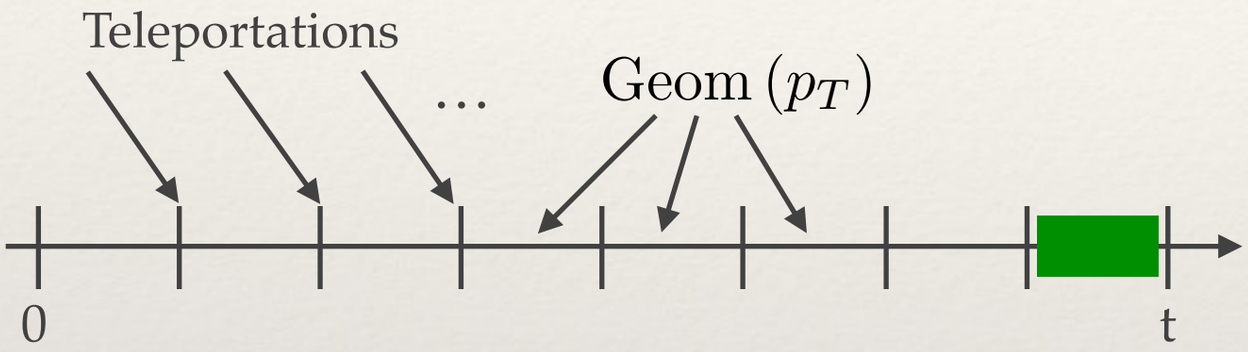
Solution

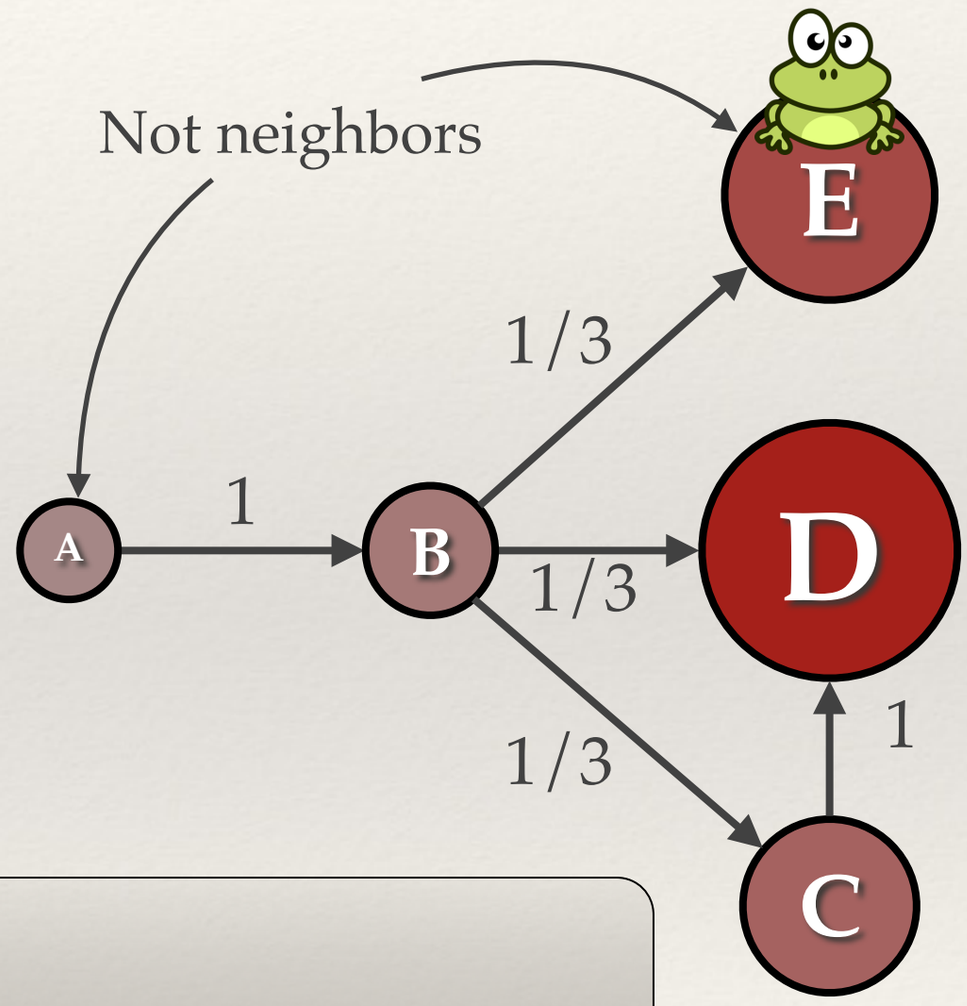
1. Give them random life
2. Do no teleportation

Lemma: This process is the same as running for mixing time + teleportations

Problem 1: Teleportation

Requires global communication





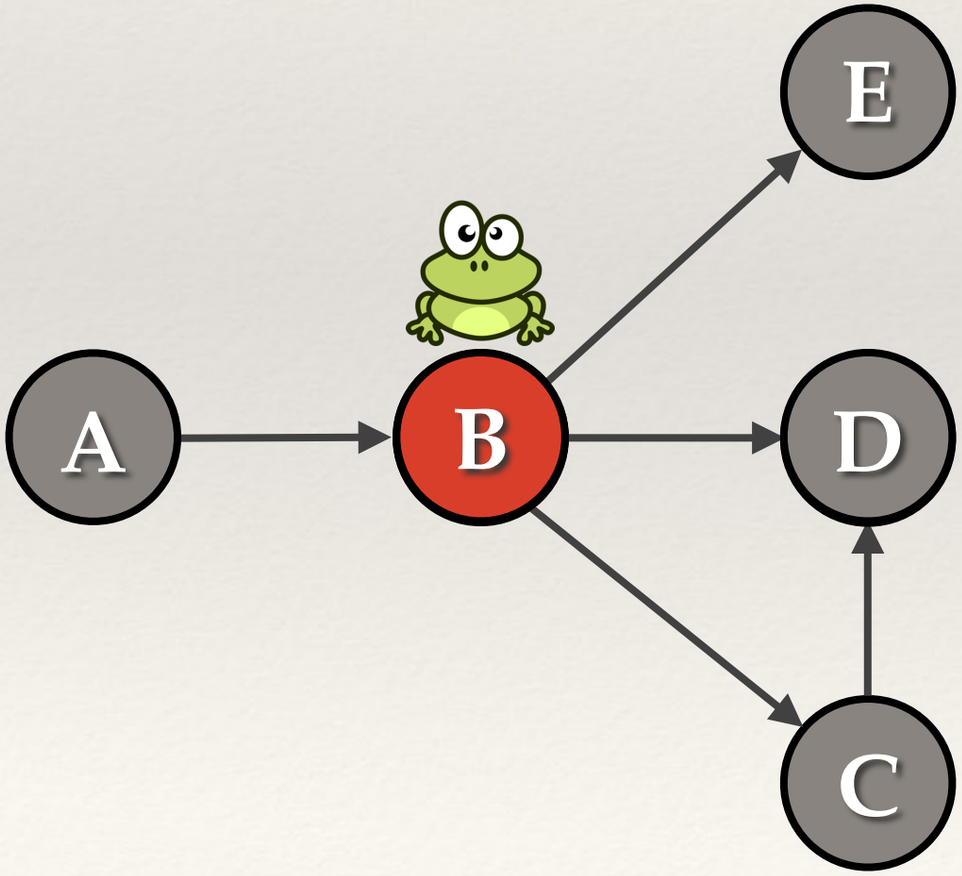
Solution

1. Give them random life
2. Do no teleportation

Used already before:
"Monte Carlo Methods in PageRank Computation: When One Iteration is Sufficient"
K. Avrachenkov, N. Litvak, D. Nemirovsky, N. Osipova (2007)

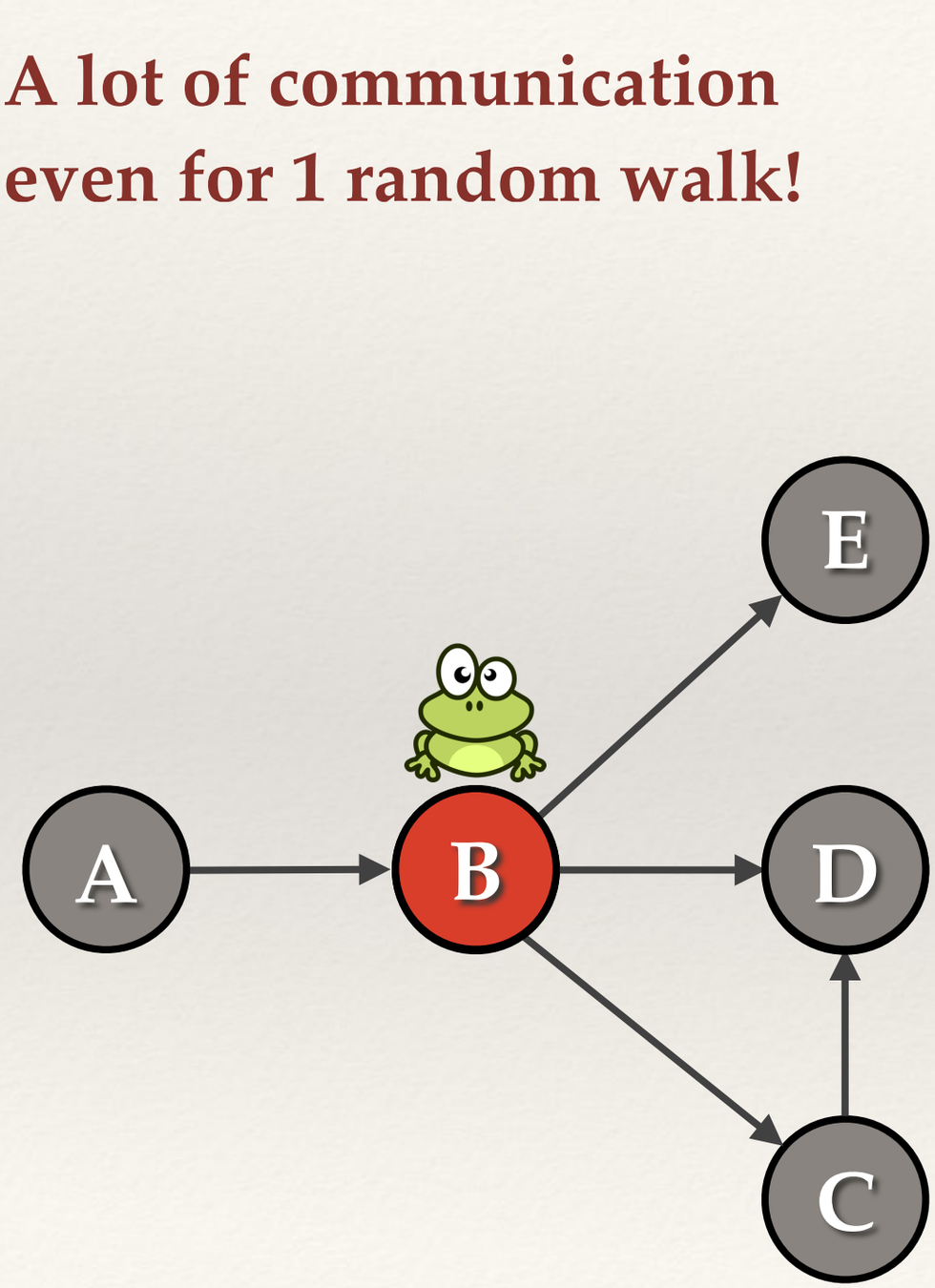
Problem 2: Synchronization traffic

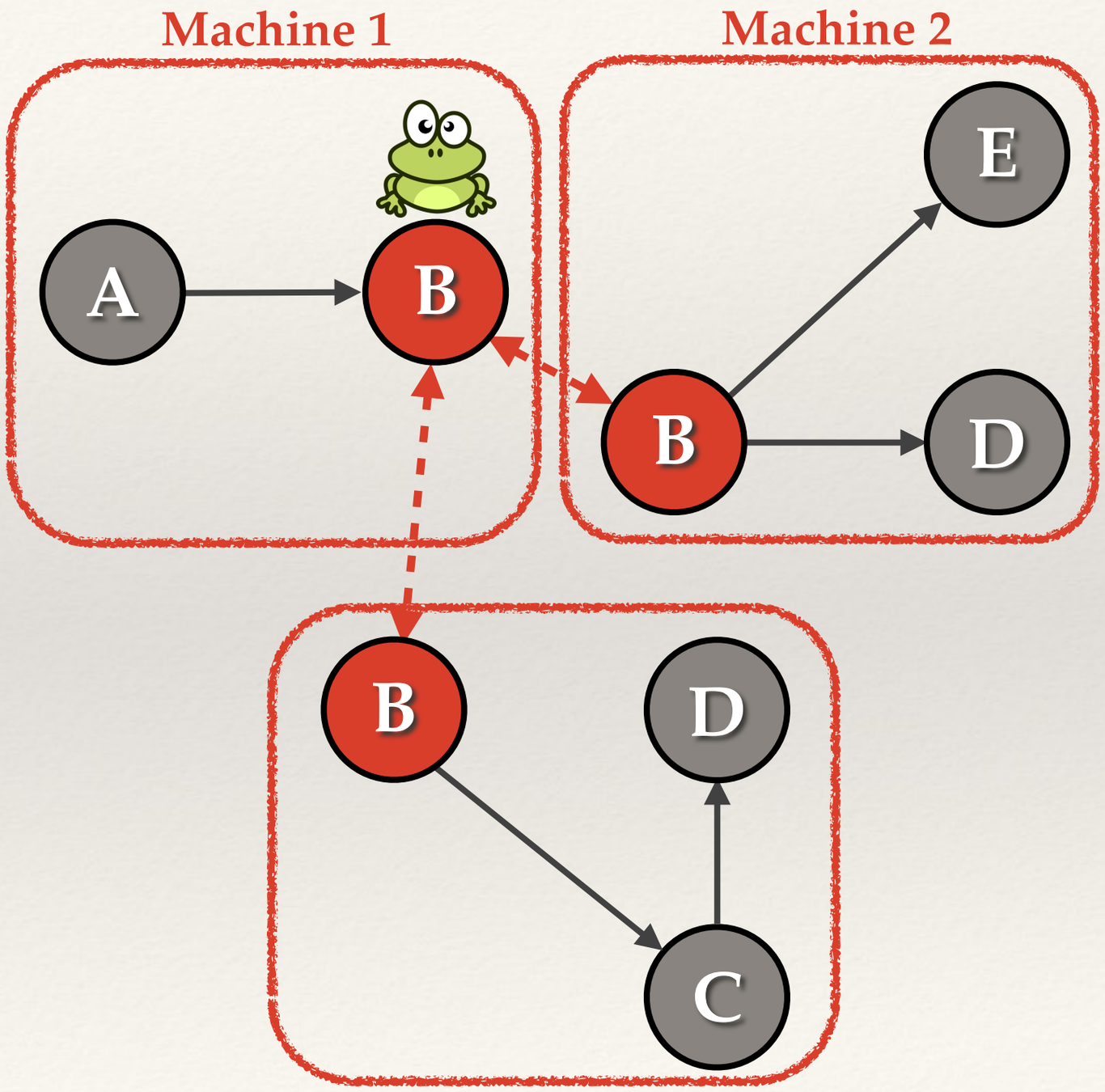
A lot of communication
even for 1 random walk!



Problem 2: Synchronization traffic

A lot of communication even for 1 random walk!

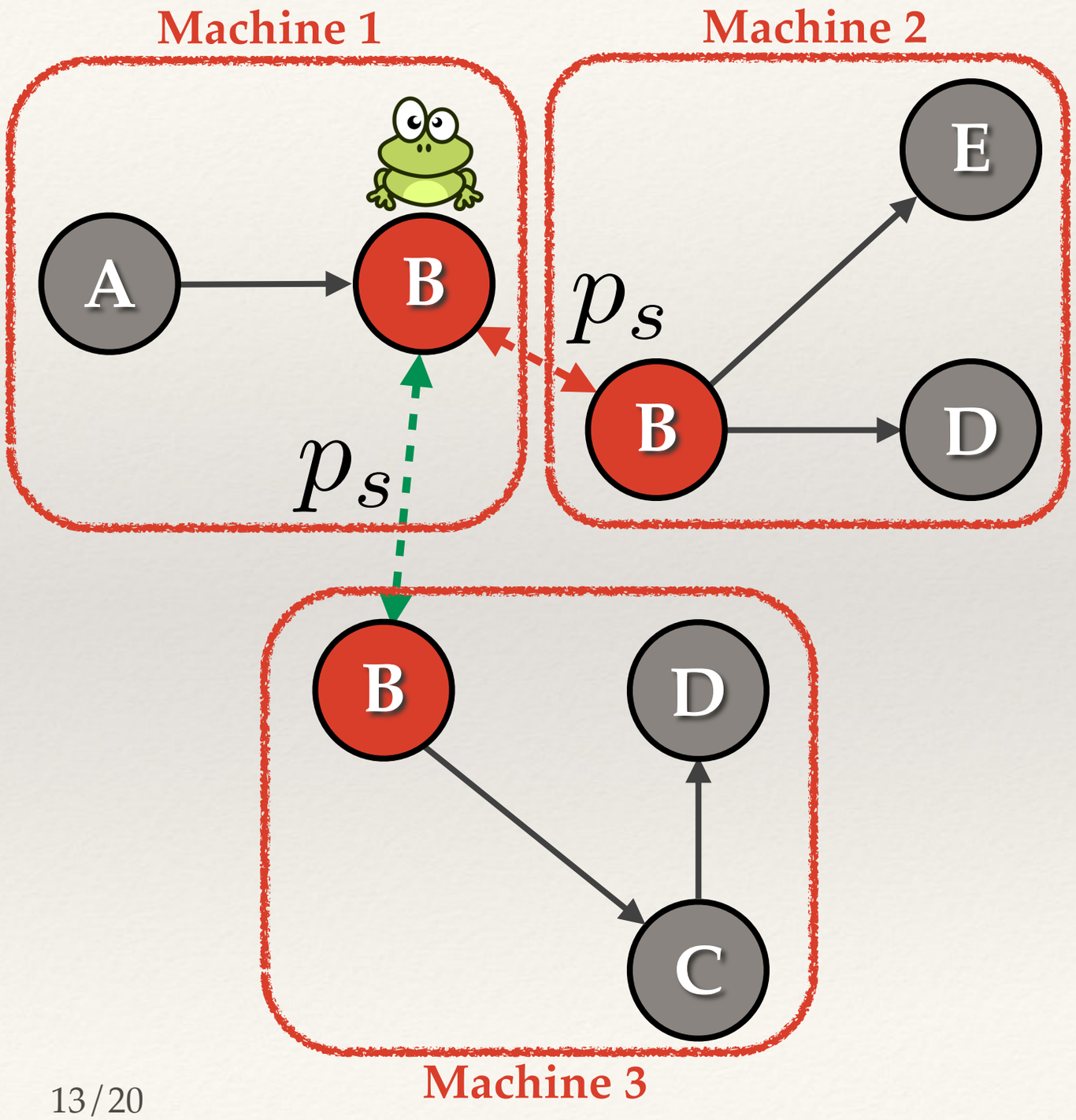




Solution: Randomized Synchronization

Modify GraphLab to expose one parameter p_s to programmer

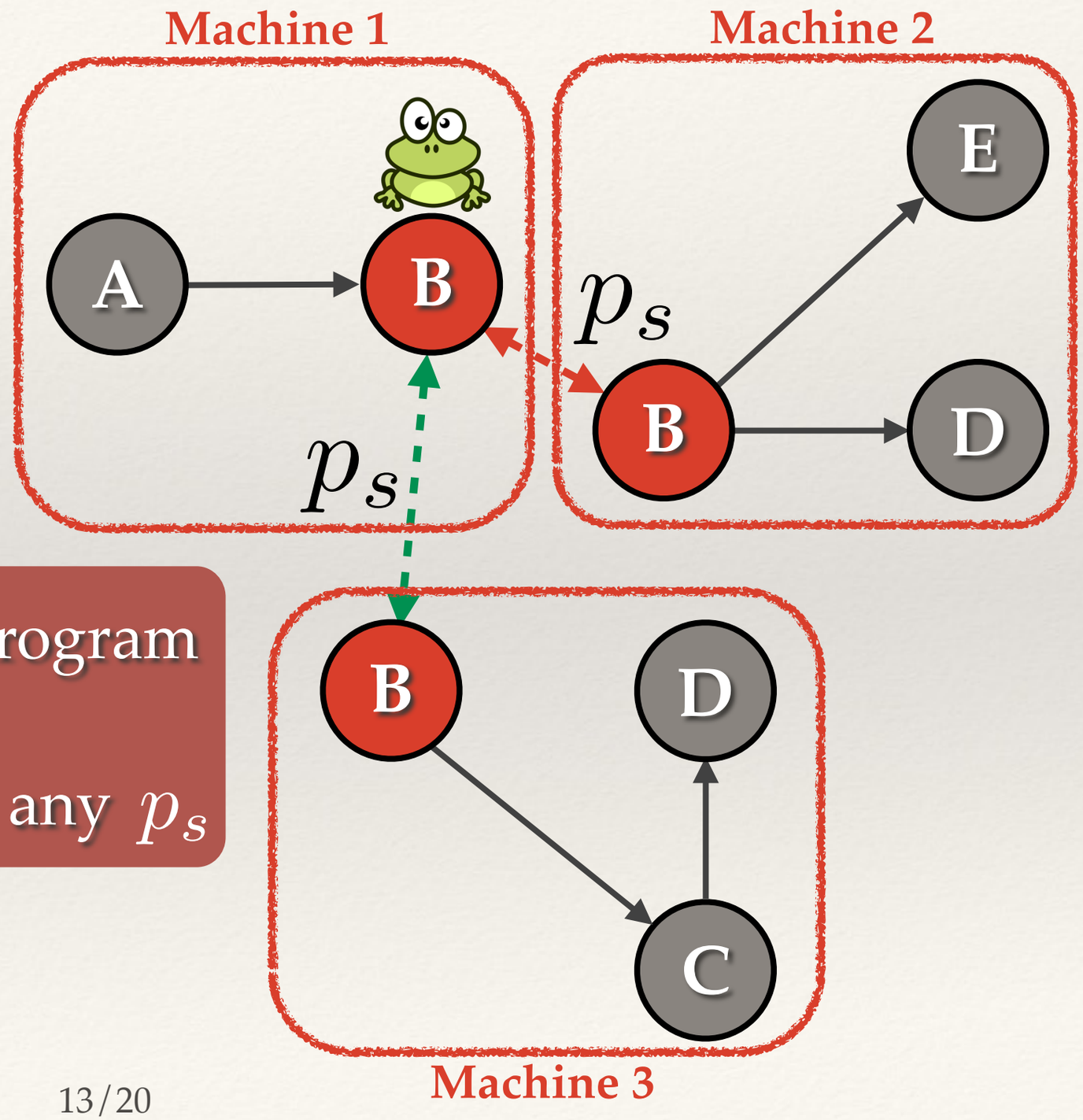
Flip a coin and synchronize each mirror vertex only w.p. p_s



Solution: Randomized Synchronization

Modify GraphLab to expose one parameter p_s to programmer

Flip a coin and synchronize each mirror vertex only w.p. p_s

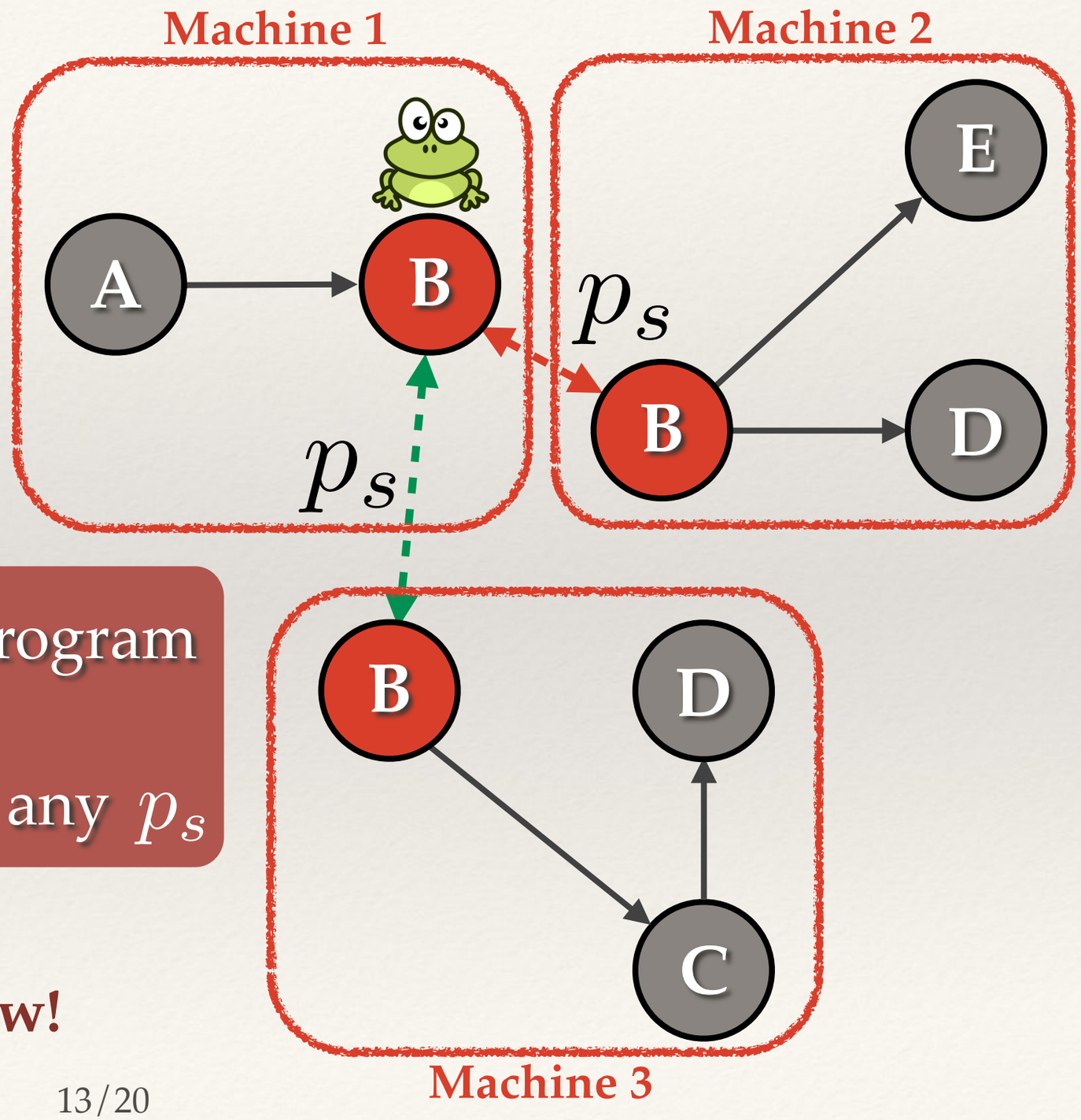


Lemma: we can write a vertex program that matches the marginals of synchronized random walks for any p_s

Solution: Randomized Synchronization

Modify GraphLab to expose one parameter p_s to programmer

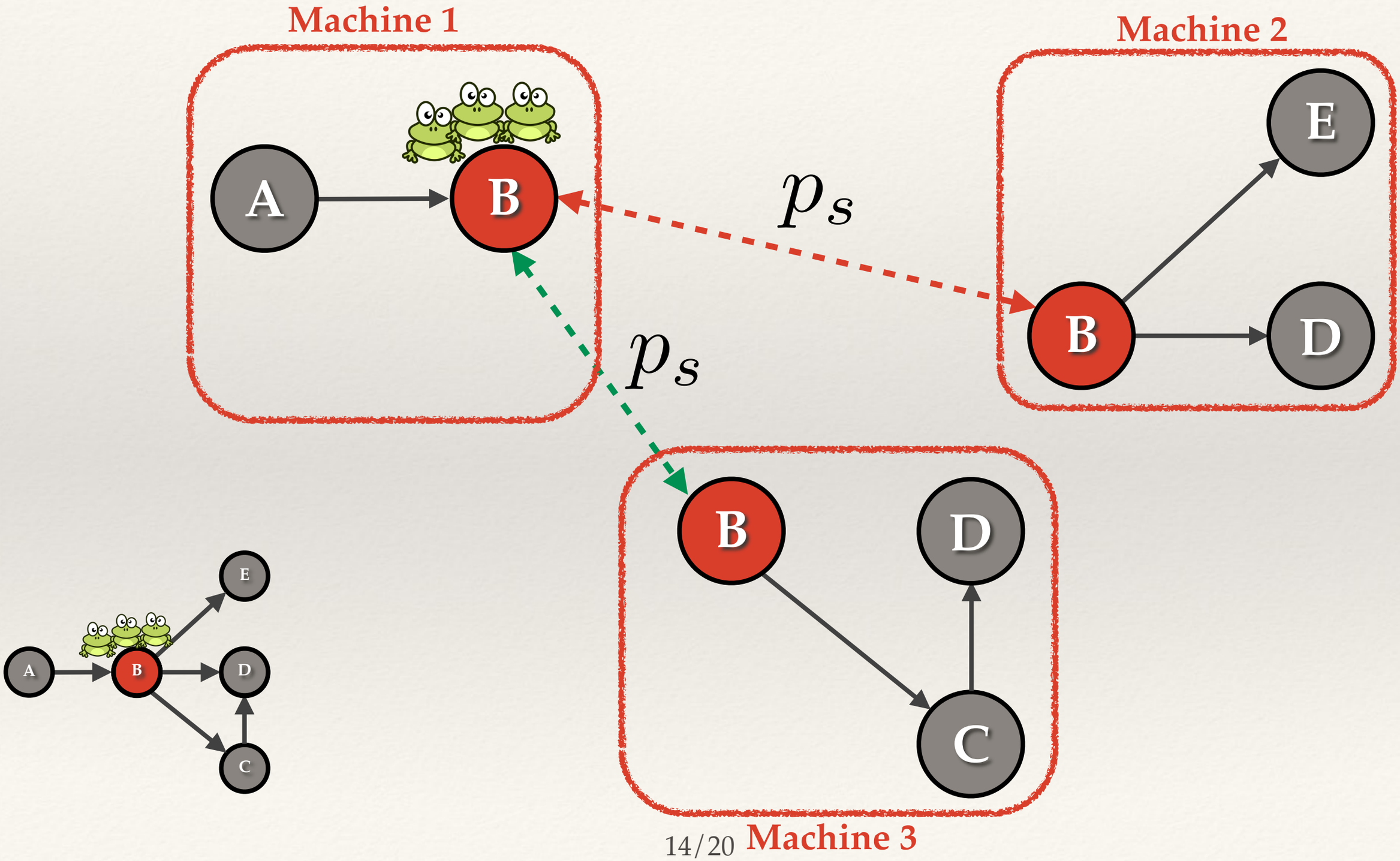
Flip a coin and synchronize each mirror vertex only w.p. p_s



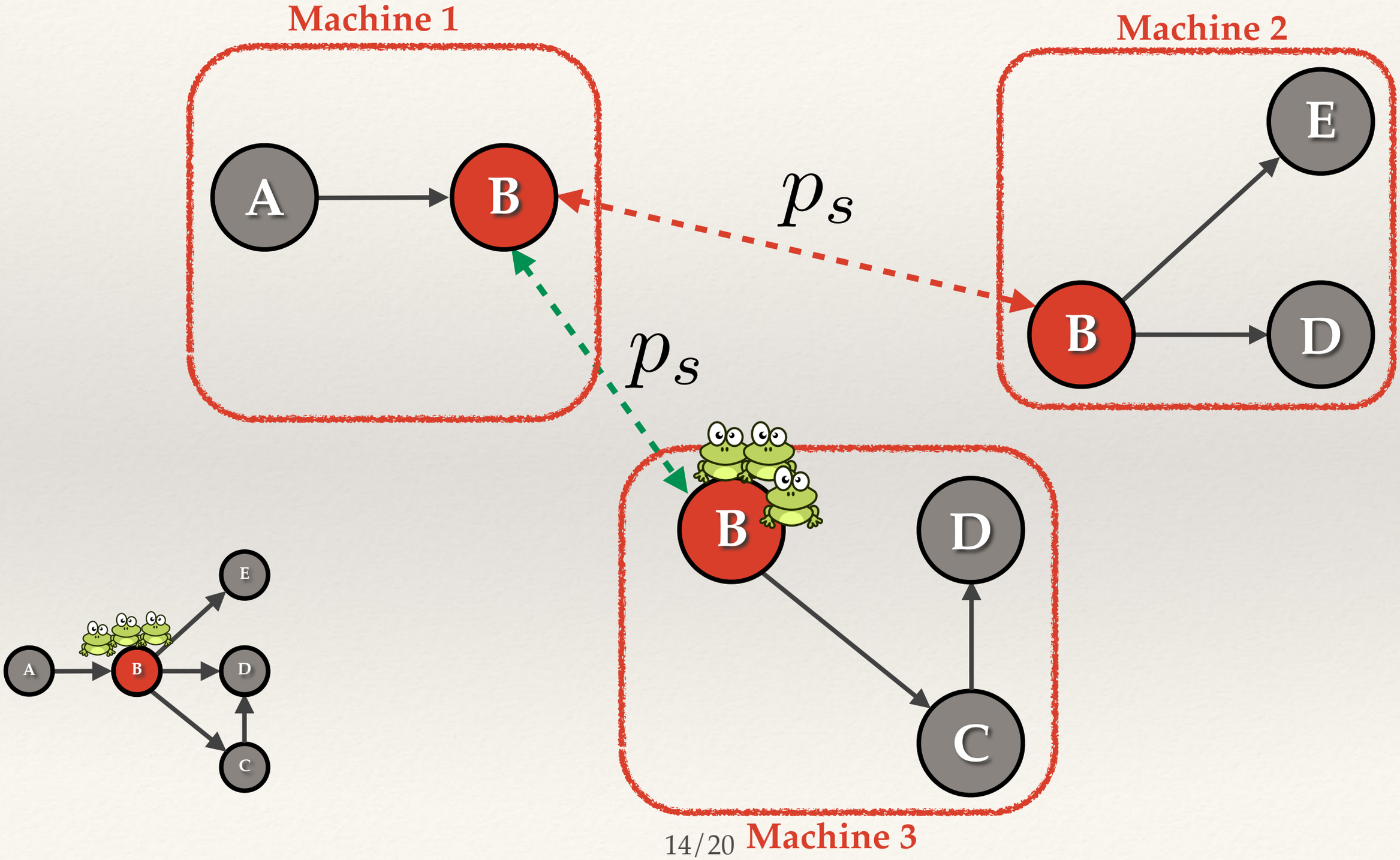
Lemma: we can write a vertex program that matches the marginals of synchronized random walks for any p_s

But walks have dependencies now!

Problem 3: Why dependencies?



Problem 3: Why dependencies?



Solution

We show that dependencies are not too strong if the graph is not bad!

Solution

We show that dependencies are not too strong if the graph is not bad!

Theorem: Mass captured by top-k set, S, of estimate from N frogs after t steps: $\pi(S) \geq \mathrm{OPT} - 2\epsilon \quad \text{w.p.} \quad 1 - \delta$

$$\epsilon < \sqrt{\frac{(1-p_T)^{t+1}}{p_T}} + \sqrt{\frac{k}{\delta}\left[\frac{1}{N} + (1-p_s^2)p_\cap(t)\right]}$$

$p_\cap(t)$ — prob. that two frogs meet after t steps starting from uniform should be small.

Solution

We show that dependencies are not too strong if the graph is not bad!

Theorem: Mass captured by top-k set, S, of estimate from N frogs after t steps: $\pi(S) \geq \mathrm{OPT} - 2\epsilon \quad \text{w.p.} \quad 1 - \delta$

$$\epsilon < \sqrt{\frac{(1-p_T)^{t+1}}{p_T}} + \sqrt{\frac{k}{\delta}\left[\frac{1}{N} + (1-p_s^2)p_\cap(t)\right]}$$

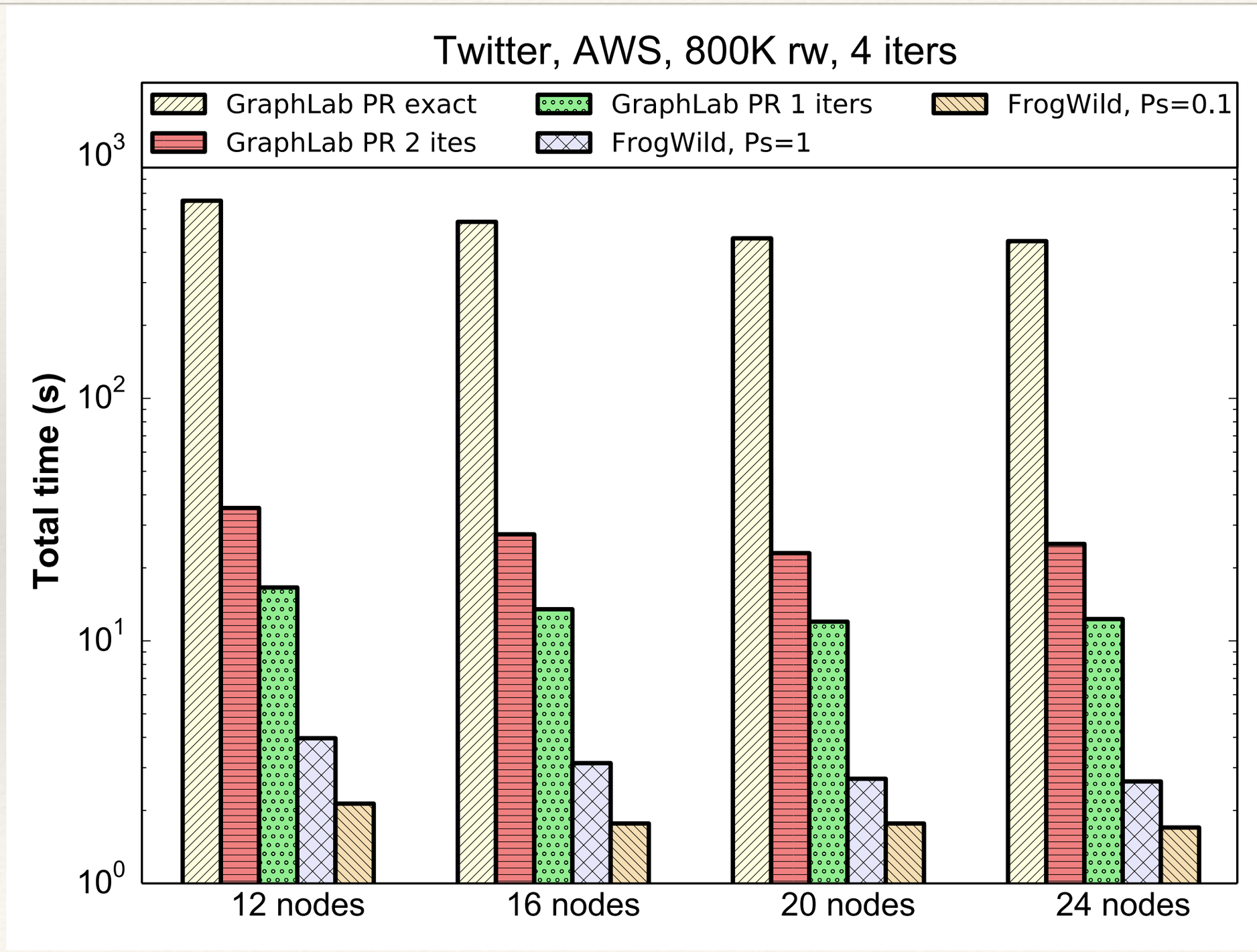
$p_\cap(t)$ — prob. that two frogs meet after t steps starting from uniform should be small.

Bad graph: Star

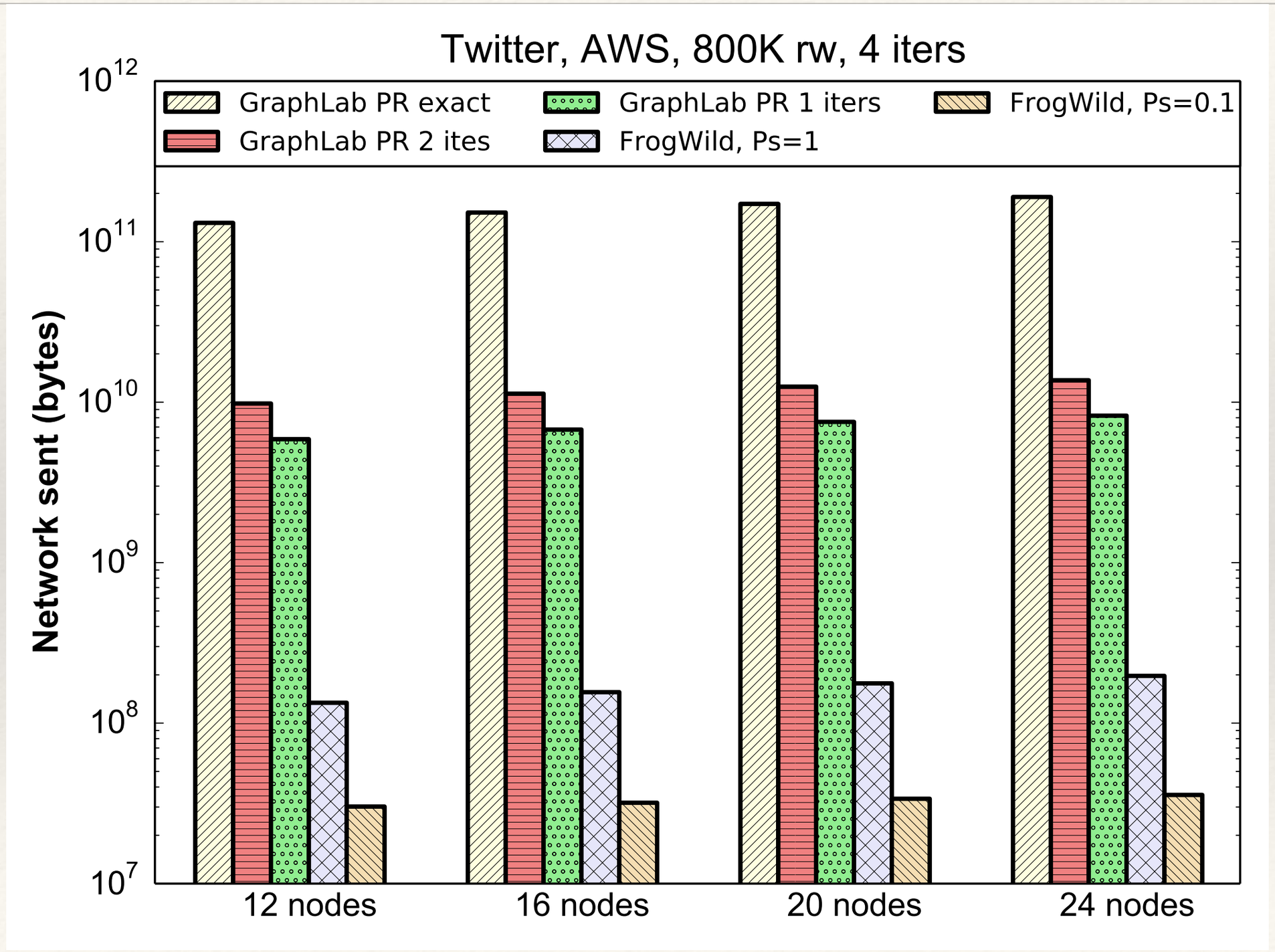
Good graph: Clique, Expander, Power-law

Experiments

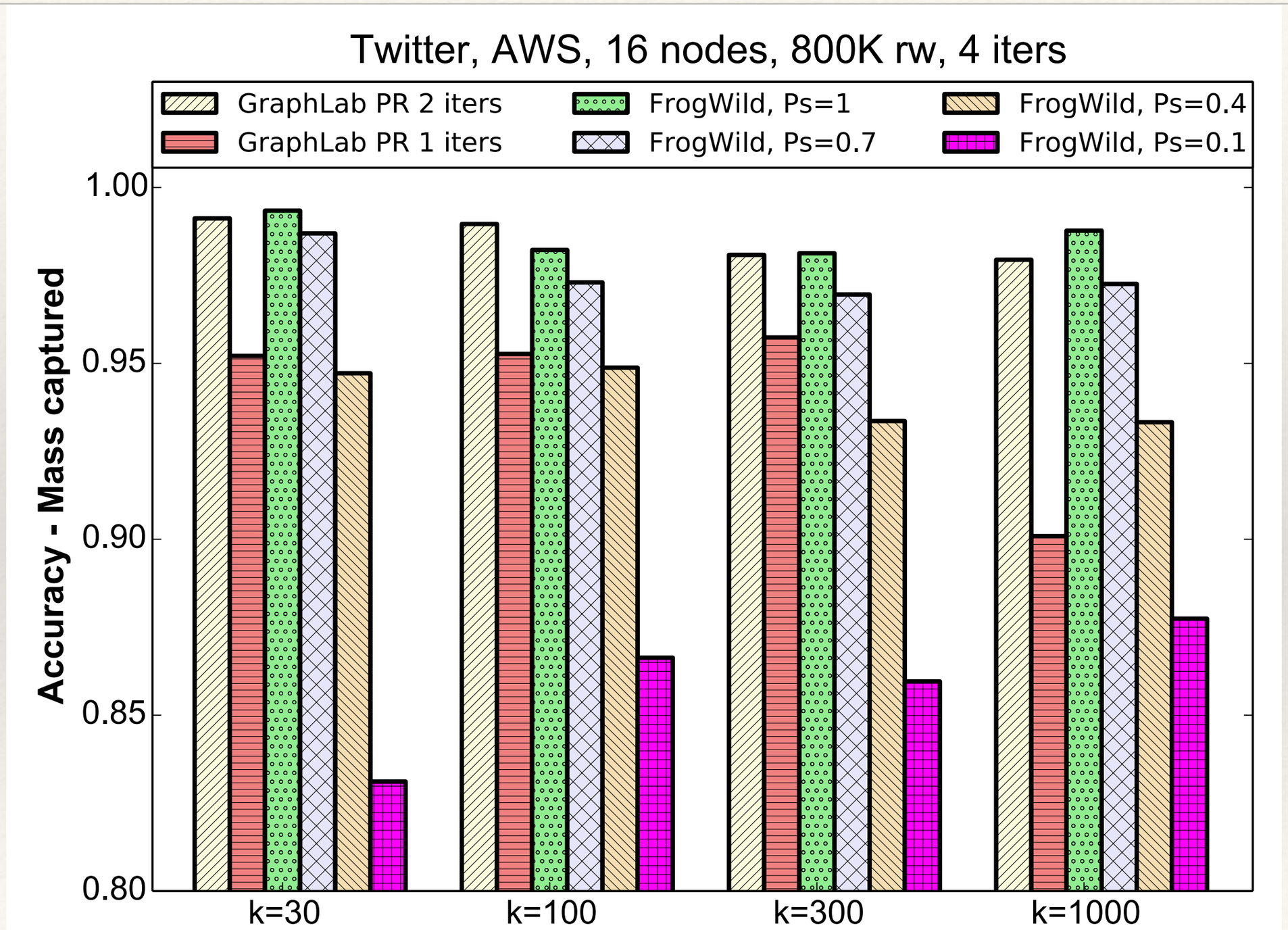
Total Runtime
Twitter Graph - 40M nodes, 1.4B edges



Network Usage
Twitter Graph - 40M nodes, 1.4B edges



Accuracy
Twitter Graph - 40M nodes, 1.4B edges



Summary

Summary

1. Algorithm for top-k PageRank approximation

Summary

1. Algorithm for top-k PageRank approximation
2. Modification of GraphLab
   Exposes very simple API extension (p_S).
   Allows for randomized synchronization.

Summary

1. Algorithm for top-k PageRank approximation
2. Modification of GraphLab
   Exposes very simple API extension (p_S).
   Allows for randomized synchronization.
3. Speed-up of 7-10x

Summary

1. Algorithm for top-k PageRank approximation
2. Modification of GraphLab
   Exposes very simple API extension (p_S).
   Allows for randomized synchronization.
3. Speed-up of 7-10x
4. Theoretical guarantees for solution despite introduced dependencies

Summary

1. Algorithm for top-k PageRank approximation
2. Modification of GraphLab
   Exposes very simple API extension (p_S).
   Allows for randomized synchronization.
3. Speed-up of 7-10x
4. Theoretical guarantees for solution despite introduced dependencies

Thank you!