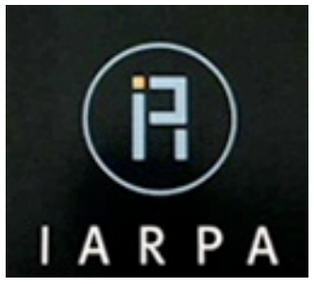
How to Build a Reliable System from Unreliable Components

Jared Saia
Computer Science Department,
University of New Mexico



PROBABILISTIC LOGICS AND THE SYNTHESIS OF RELIABLE ORGANISMS FROM UNRELIABLE COMPONENTS

J. von Neumann

1. INTRODUCTION

The paper that follows is based on notes taken by Dr. R. S. Pierce on five lectures given by the author at the California Institute of Technology in January 1952. They have been revised by the author but they reflect, apart from minor changes, the lectures as they were delivered.

The subject-matter, as the title suggests, is the role of error in logics, or in the physical implementation of logics — in automata-synthesis. Error is viewed, therefore, not as an extraneous and misdirected or misdirecting accident, but as an essential part of the process under consideration — its importance in the synthesis of automata being fully comparable to that of the factor which is normally considered, the intended and correct logical structure.

Our present treatment of error is unsatisfactory and ad hoc. It is the author's conviction, voiced over many years, that error should be treated by thermodynamical methods, and be the subject of a thermodynamical theory, as information has been, by the work of L. Szilard and C. E. Shannon [Cf. 5.2]. The present treatment falls far short of achieving this, but it assembles, it is hoped, some of the building materials, which will have to enter into the final structure.

PROBABILISTIC LOGICS AND THE SYNTHESIS OF RELIABLE ORGANISMS FROM UNRELIABLE COMPONENTS

J. von Neumann

1. INTRODUCTION

The paper that follows is based on notes taken by Dr. R. S. Pierce on five lectures given by the author at the California Institute of Technology in January 1952. They have been revised by the author but they reflect, apart from minor changes, the lectures as they were delivered.

The subject-matter, as the title suggests, is the role of error in logics, or in the physical implementation of logics — in automata-synthesis. Error is viewed, therefore, not as an extraneous and misdirected or misdirecting accident, but as an essential part of the process under consideration — its importance in the synthesis of automata being fully comparable to that of the factor which is normally considered, the intended and correct logical structure.

Our present treatment of error is unsatisfactory and ad hoc. It is the author's conviction, voiced over many years, that error should be treated by thermodynamical methods, and be the subject of a thermodynamical theory, as information has been, by the work of L. Szilard and C. E. Shannon [Cf. 5.2]. The present treatment falls far short of achieving this, but it assembles, it is hoped, some of the building materials, which will have to enter into the final structure.

Networks of Noisy Gates

Networks of Noisy Gates

- We are given a function, f, that can be computed with n gates

Networks of Noisy Gates

- We are given a function, f, that can be computed with n gates
- Must build a network to compute f with *unreliable* gates

Networks of Noisy Gates

- We are given a function, f, that can be computed with n gates
- Must build a network to compute f with *unreliable* gates
- Gates are *unreliable*: with probability ε they *fault*; when they fault, output is incorrect

Networks of Noisy Gates

- We are given a function, f, that can be computed with n gates
- Must build a network to compute f with *unreliable* gates
- Gates are *unreliable*: with probability ε they *fault*; when they fault, output is incorrect
- Q: How many unreliable gates do we need to compute f with probability 1-o(1)

Networks of Noisy Gates

Q: How many unreliable gates do we need to compute f with probability approaching 1?

- O(n log n) gates suffice [Von Neumann '56]

Boosting a Noisy Gate

Boosting a Noisy Gate

- Naive
  - Copy each gate log n times
  - Copy each wire log n times
  - Take majority of all outputs at end

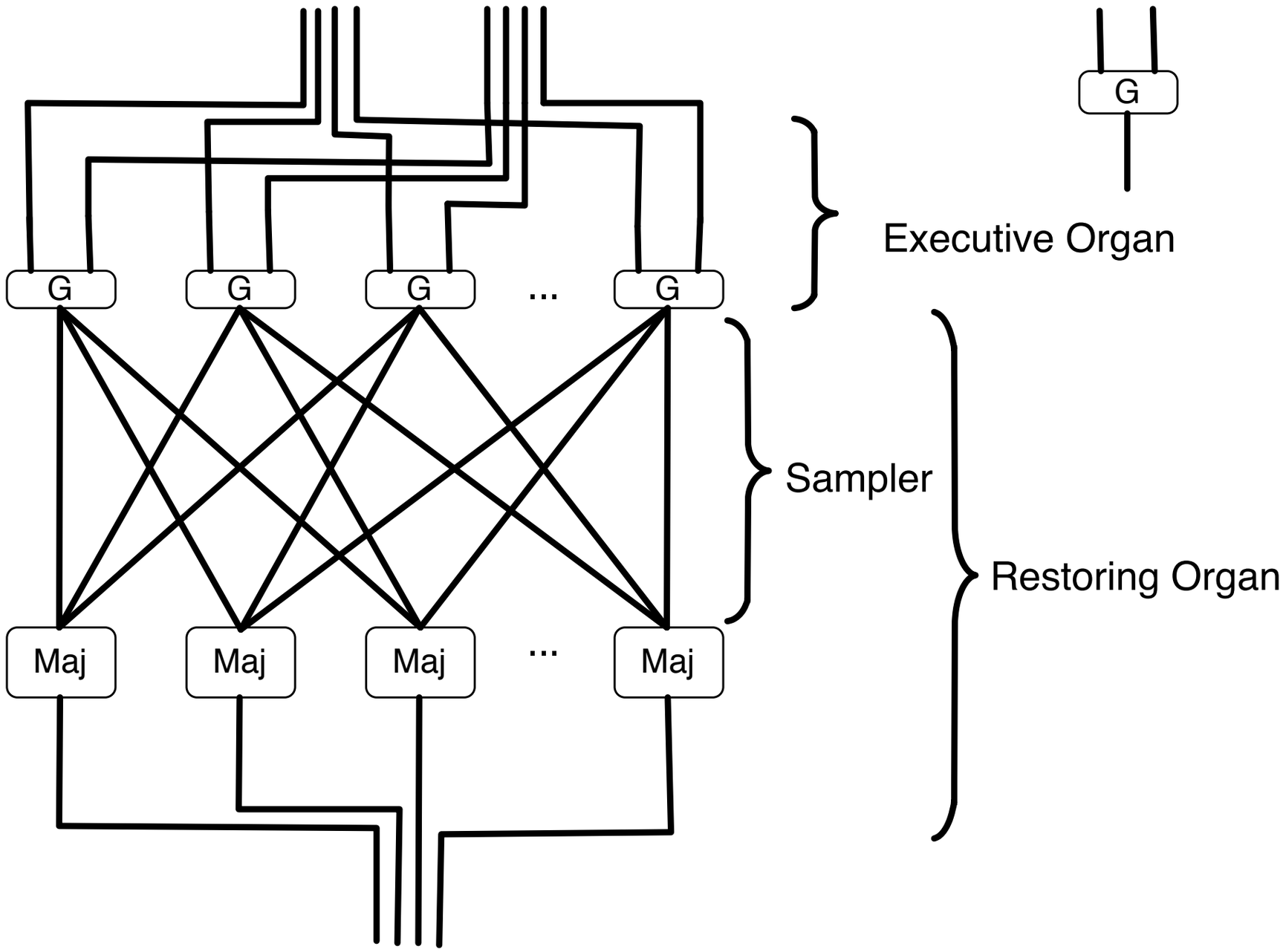
Boosting a Noisy Gate

- Naive
  - Copy each gate log n times
  - Copy each wire log n times
  - Take majority of all outputs at end
- Problem: Error accumulates at each level

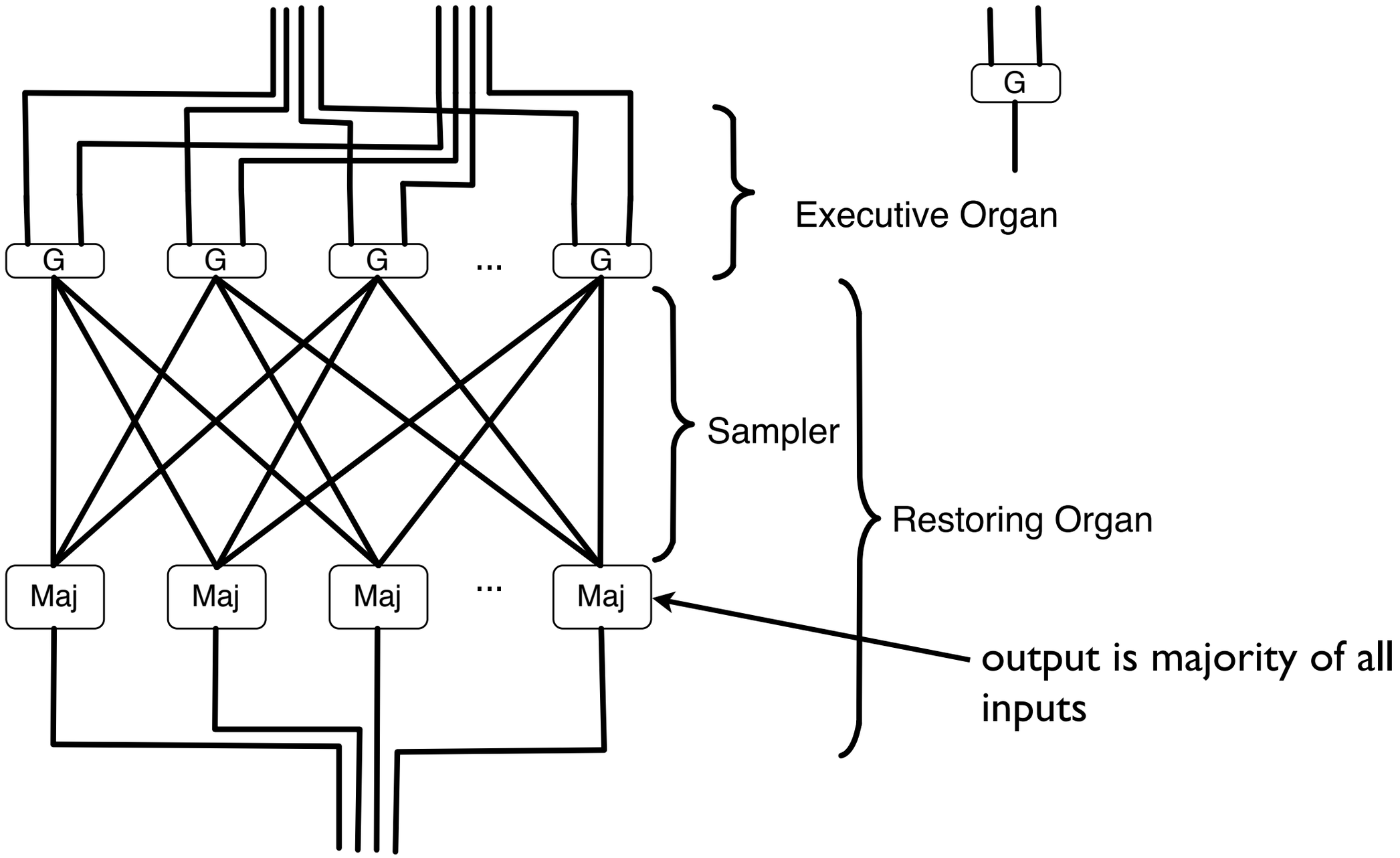
Boosting a Noisy Gate

- Naive
  - Copy each gate log n times
  - Copy each wire log n times
  - Take majority of all outputs at end
- Problem: Error accumulates at each level
- Solution: "Restoring Organ"

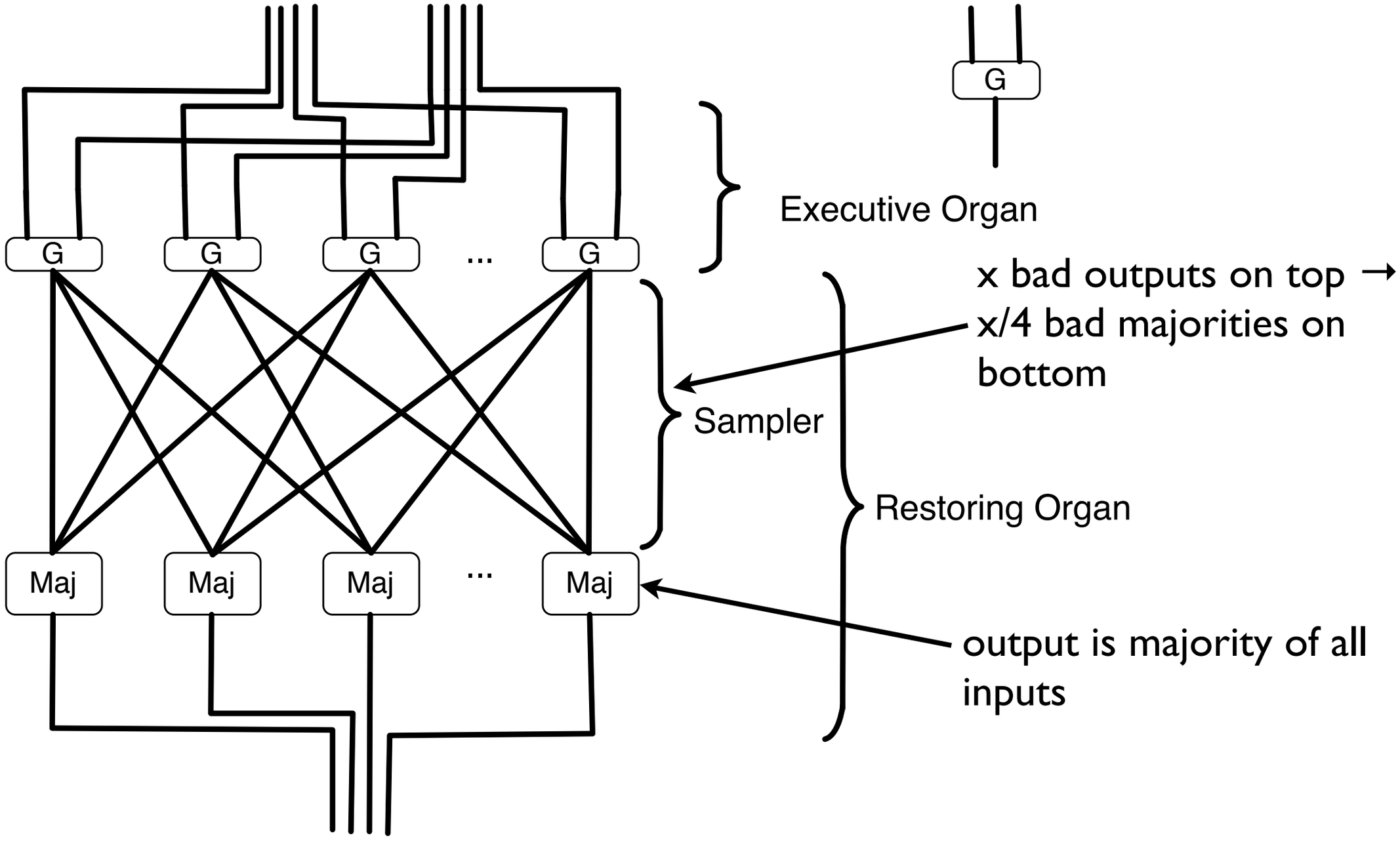
Boosting a Noisy Gate



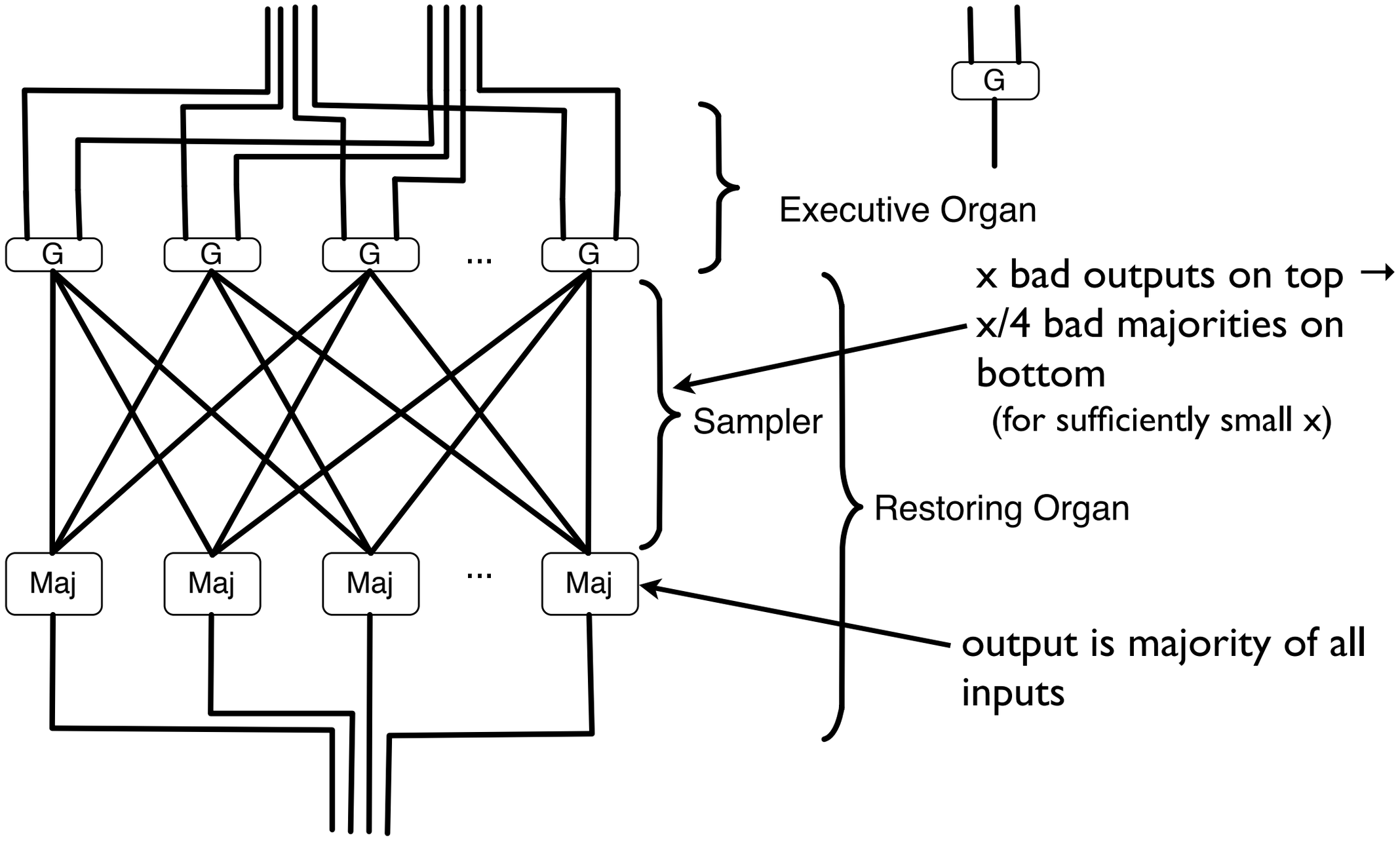
Boosting a Noisy Gate

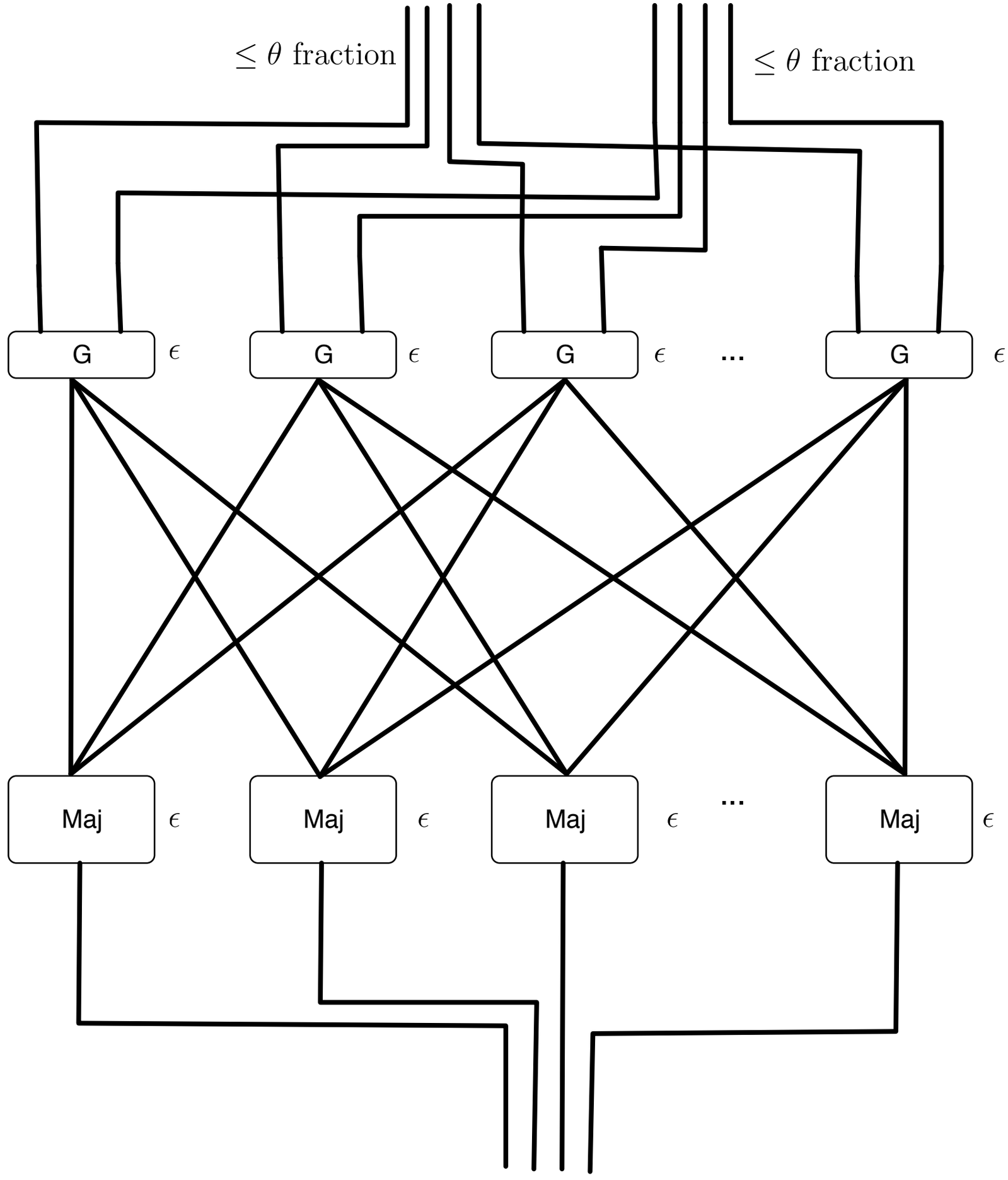


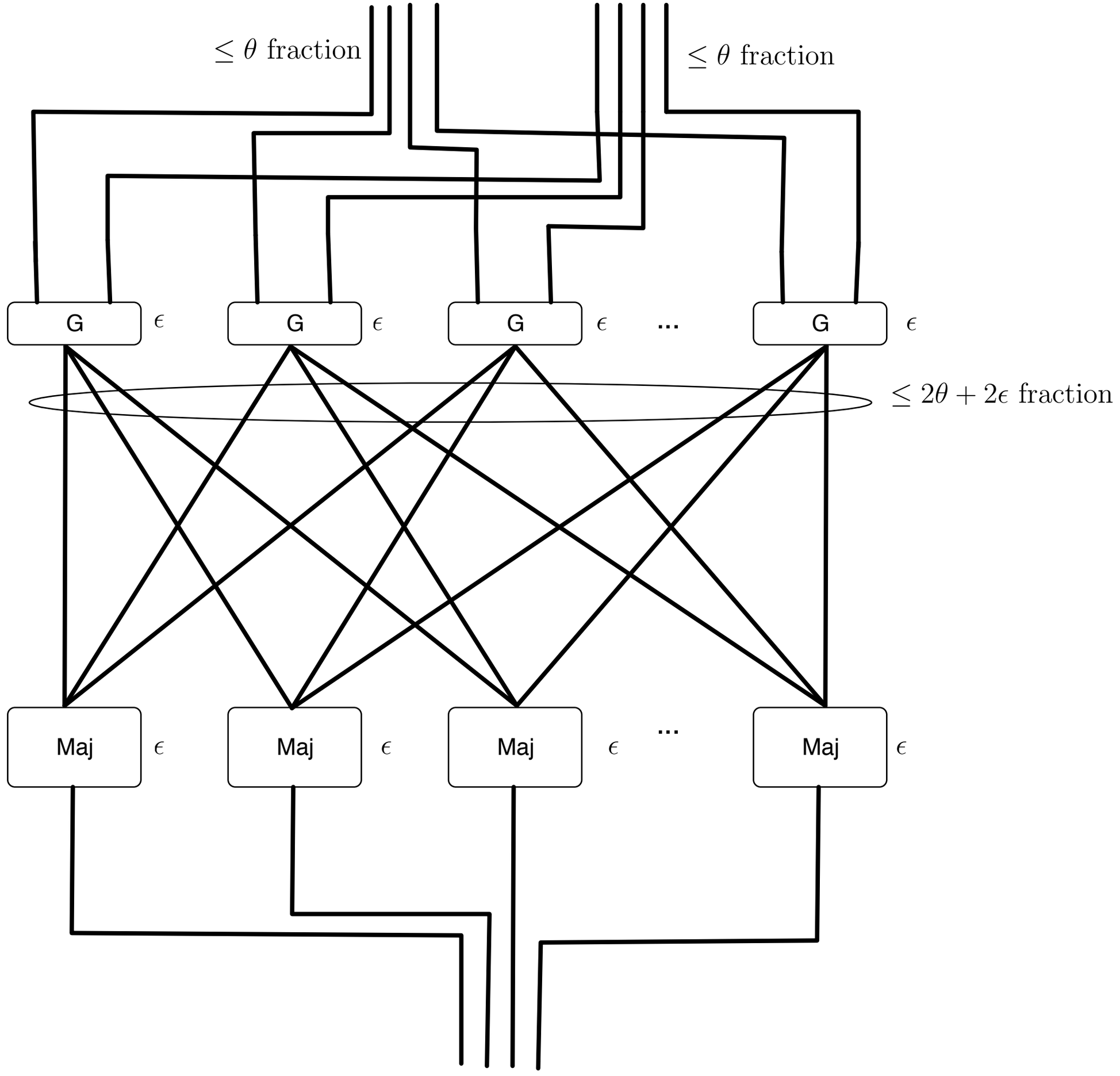
Boosting a Noisy Gate

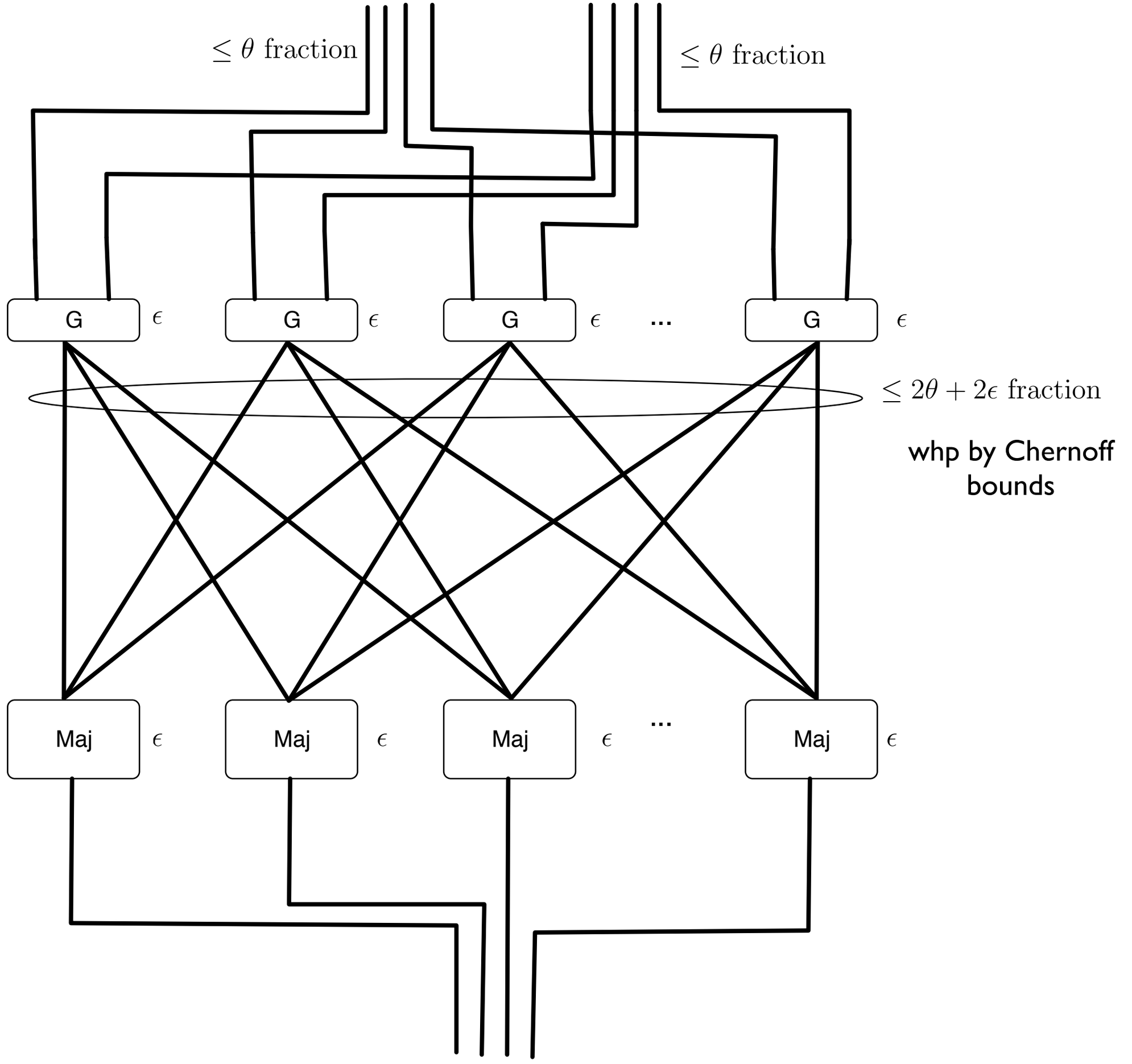


Boosting a Noisy Gate

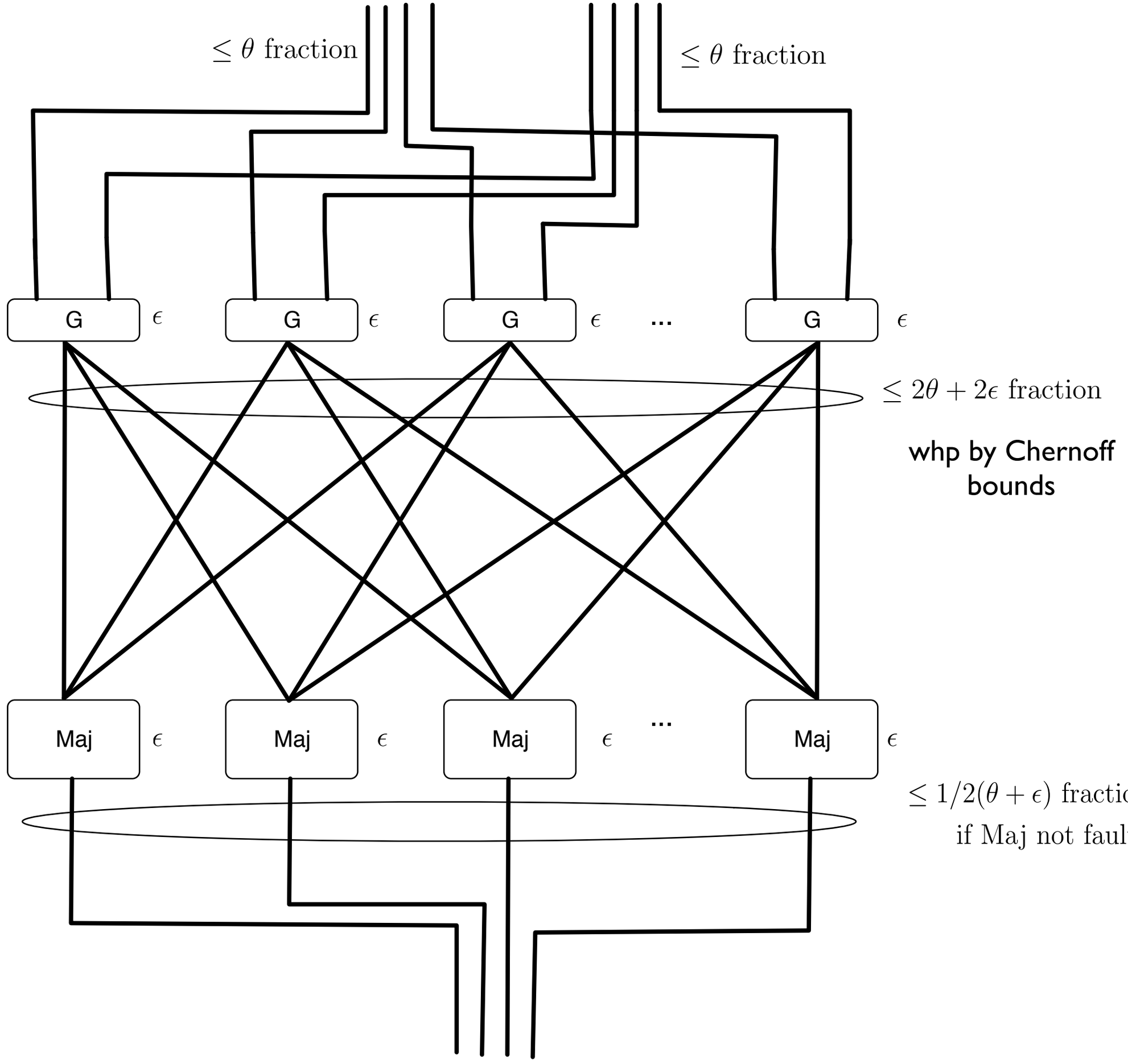


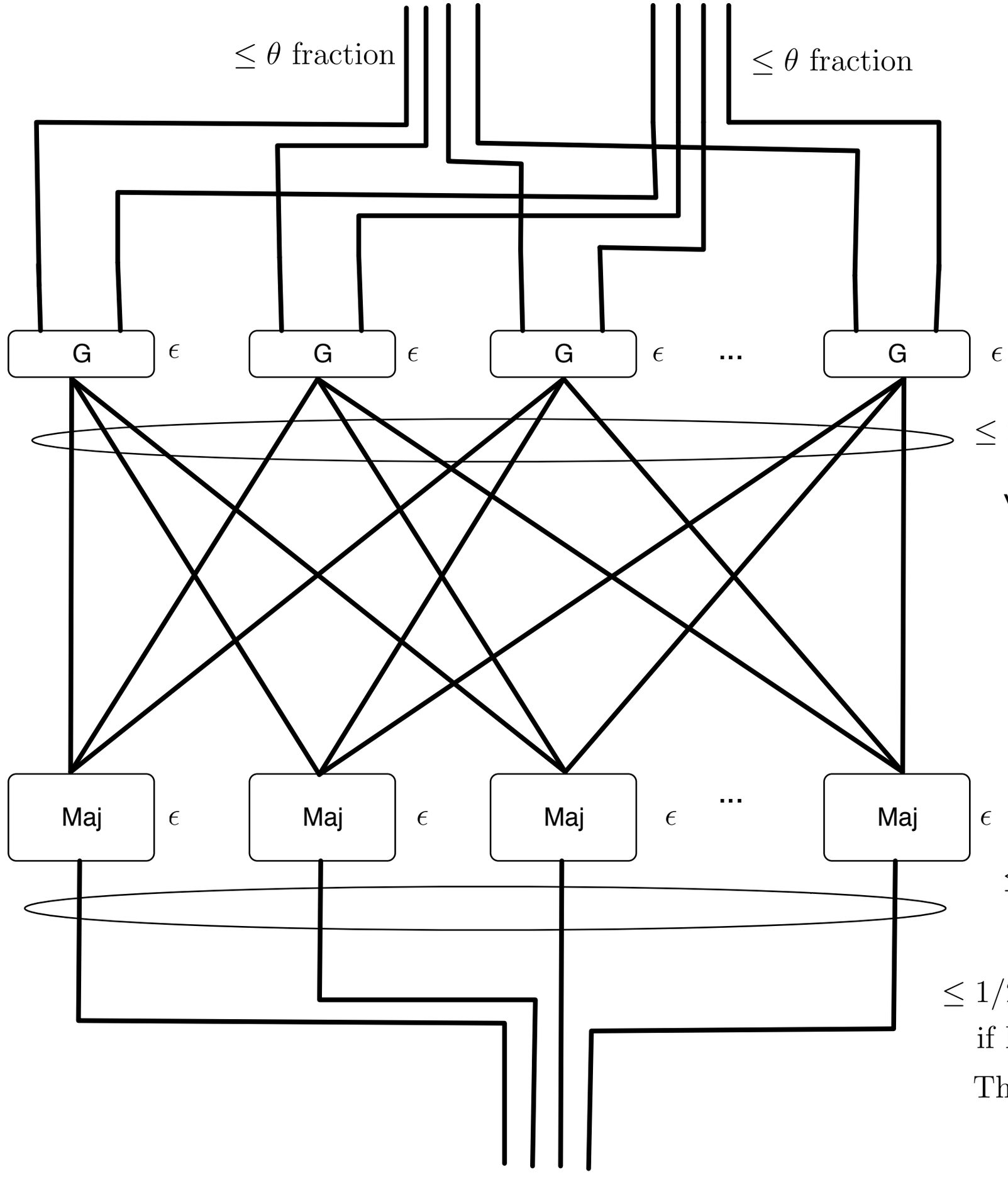

$\leq \theta$ fraction
$\leq \theta$ fraction
G
ϵ
G
ϵ
G
ϵ
...
G
ϵ
Maj
ϵ
Maj
ϵ
Maj
ϵ
...
Maj
ϵ


$\leq \theta$ fraction
$\leq \theta$ fraction
G
ϵ
G
ϵ
G
ϵ
...
G
ϵ
$\leq 2\theta + 2\epsilon$ fraction
Maj
ϵ
Maj
ϵ
Maj
ϵ
...
Maj
ϵ



whp by Chernoff bounds


$\leq \theta$ fraction
$\leq \theta$ fraction
G
ϵ
G
ϵ
G
ϵ
...
G
ϵ
$\leq 2\theta + 2\epsilon$ fraction
whp by Chernoff bounds
Maj
ϵ
Maj
ϵ
Maj
ϵ
...
Maj
ϵ
$\leq 1/2(\theta + \epsilon)$ fraction if Maj not faulty



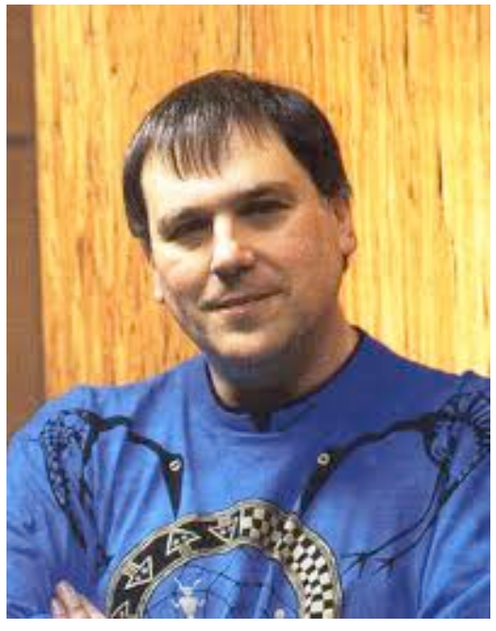
whp by Chernoff bounds

$\leq 1/2(\theta + \epsilon)$ fraction if Maj not faulty

$\leq 1/2(\theta + \epsilon) + 2\epsilon$ fraction if Maj faulty w/ prob ϵ

This last term is $\leq \theta$ for large θ

Networks of Noisy Gates



Q: How many unreliable gates do we need to compute f with probability approaching 1?

- O(n log n) gates suffice [Von Neumann '56]
- $\Omega(n \log n)$ gates necessary [PST '91]

Noisy Gates Issues

- Problems
  - O(log n) **resource blowup**
  - Gates more constrained than processors
  - Faults are uncorrelated
  - Faults are fail-stop

Secure Multiparty Computation (MPC)

- Another problem where we want to boost reliability
- Goal: reliable computation of any function f
  - Even when a hidden subset of the processors are *bad* (i.e. Byzantine or controlled by an adversary)

Secure MPC

[Yao ’82]

Secure MPC

[Yao '82]

- n processors want to compute a function f over n inputs. f can be computed with m gates.

Secure MPC

[Yao '82]

- n processors want to compute a function f over n inputs. f can be computed with m gates.
- Each processor has one input

Secure MPC

[Yao '82]

- n processors want to compute a function f over n inputs. f can be computed with m gates.
- Each processor has one input
- Up to t<n/3 processors are *bad*

Secure MPC

[Yao '82]

- n processors want to compute a function f over n inputs. f can be computed with m gates.
- Each processor has one input
- Up to t<n/3 processors are *bad*

Note: The traditional MPC definition has additional privacy requirements that are ignored here

Applications as Functions

- Auctions

$$f = \max(x_1, x_2, \ldots, x_n)$$

- Threshold cryptography

$$f = M^s \mod pq$$

- Information aggregation

$$f = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n} - \left(\frac{\sum_{i=1}^{n} x_i}{n}\right)^2}$$

Applications as Functions

- Auctions

$$f = \max(x_1, x_2, \ldots, x_n)$$

- Threshold cryptography

$$f = M^s \mod pq$$

1) M, p, q are parameters of the function;
2) s is the y intercept of a degree $(d\text{-}1)$ function with points given by the x_i values.

- Information aggregation

$$f = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n} - \left(\frac{\sum_{i=1}^{n} x_i}{n}\right)^2}$$

MPC Results

- Recent renaissance in MPC
  - Since 2008, MPC is used annually in Denmark to hold an auction between 5,000 beet farmers and sugar producers
  - Significant recent theoretical improvements due to homomorphic encryption

MPC for large networks

- We have worked on the problem of designing MPC protocols for large n
- We use **quorums**
  - A quorum is O(log n) processors, most of which are good
  - Can get all processors to agree on n quorums [KS '11]
  - Each gate computed by a quorum

Our Result

- Resource costs
  - Bits sent per processor and computation per processor is $\tilde{O}(\frac{m+n}{n} + \sqrt{n})$
  - Latency is polylog
- We solve MPC with high probability meaning probability of error that goes to 1 as n grows large

Problem

- Resource overhead is still polylogarithmic
- Requires polylog times more computation, communication, etc compared to non-robust algorithms

Can we do better?

- Is logarithmic redundancy necessary for MPC?
- We don’t know
- It’s necessary for a quorum approach, but there may be a smarter way

Noisy Gates and MPC

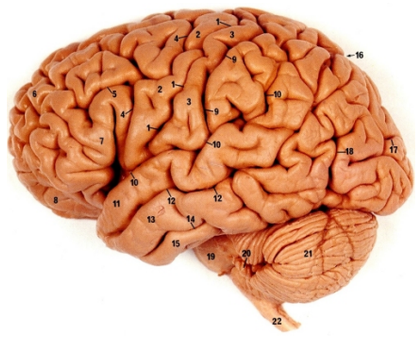
- Good:
  - Simple, well-defined problems
  - Significant theoretical and empirical progress
- Bad
  - Both problems seem to require a redundant approach

Dream Result

- Given parallel algorithm A over n nodes, create parallel algorithm A'
- A' works even if t<n/2 nodes fail
- Resource costs of A' are O(t) more than resource costs of A

Redundancy vs Self-healing

Our algorithms: require many redundant components to tolerate 1 failure



Brain: rewires to have other components help out when a component fails

Towards a Research Agenda

"Make no little plans"

- **Succinct** problems are retained
- **Important** problems span disciplines
- **Hard** problems pull in smart people

Self-healing

- A self-healing system, starting from a correct state, under attack from an adversary, goes only temporarily out of a correct state.
- Our initial work: Under attack from powerful adversary, maintain certain topological properties within acceptable bounds.

Ensuring Robustness

- Want to ensure that our network is robust to node failures
- Idea: Build some redundancy into the network?
- Example: Connectivity
  - Use k-connected graph.
  - Price: degree must be at least k.

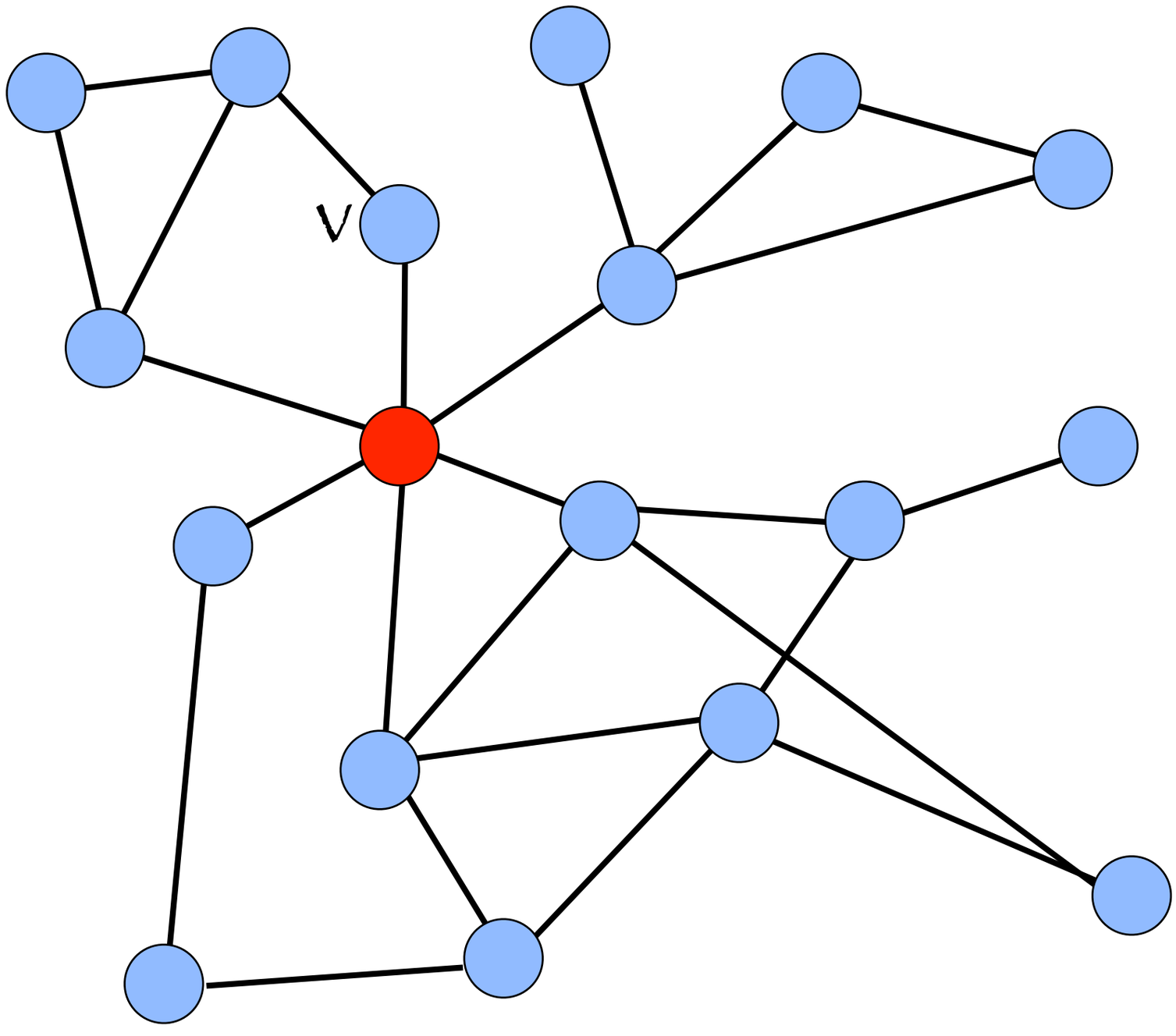
Ensuring Robustness

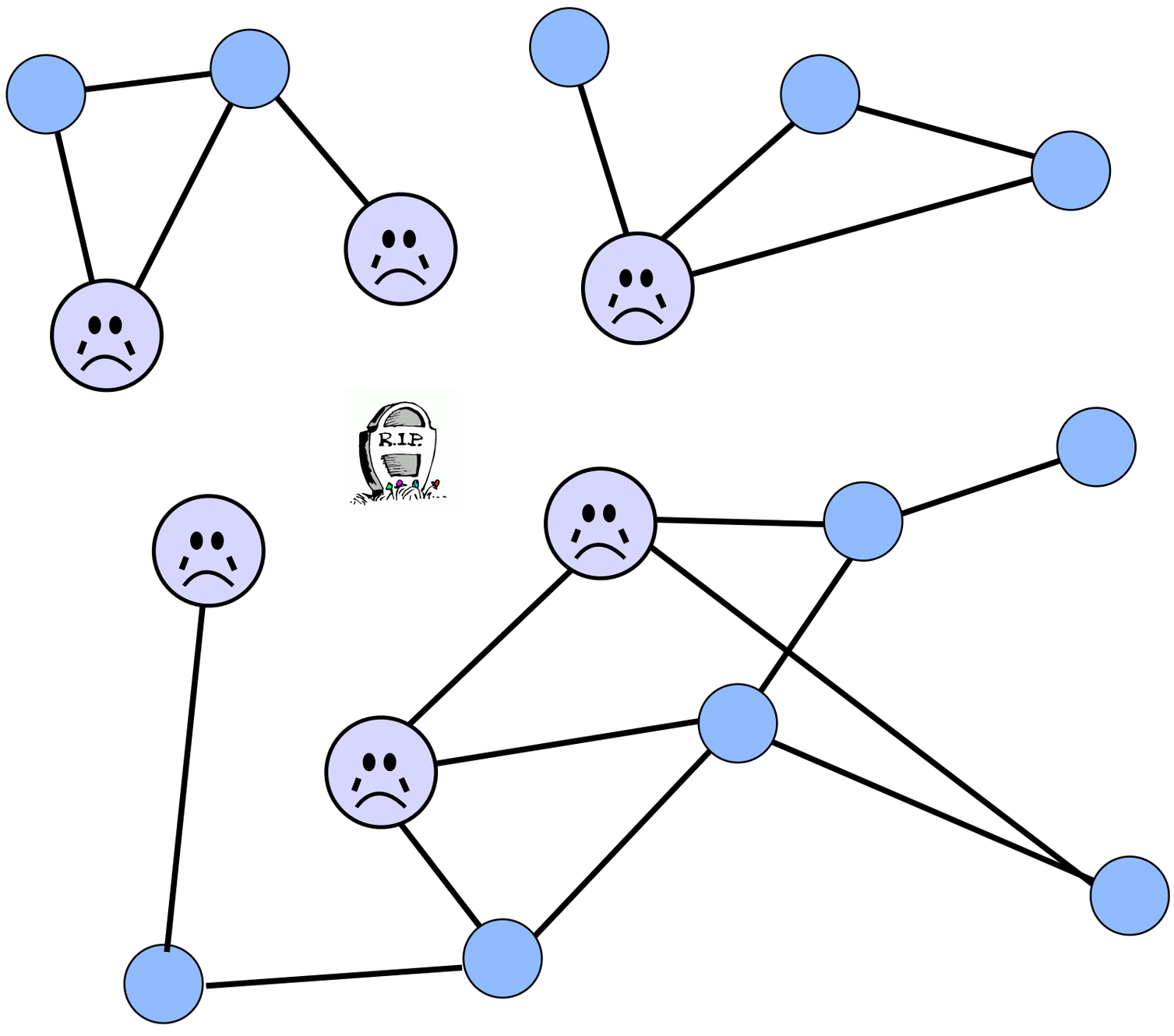
- Want to ensure that our network is robust to node failures.
- Idea: build some redundancy into the network?
- Example: Connectivity
  - Use k-connected graph.
  - Price: degree must be at least k.

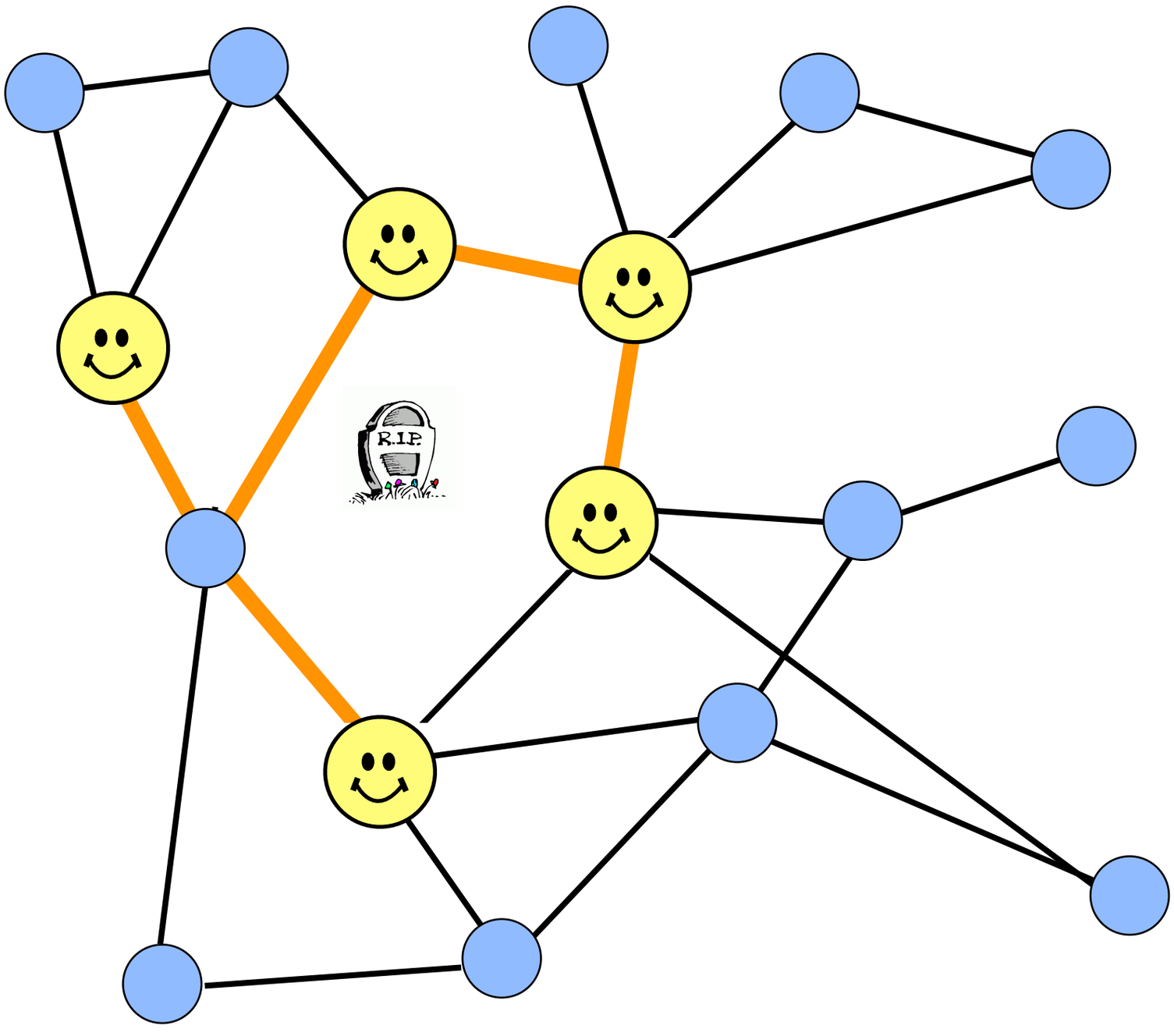
Model

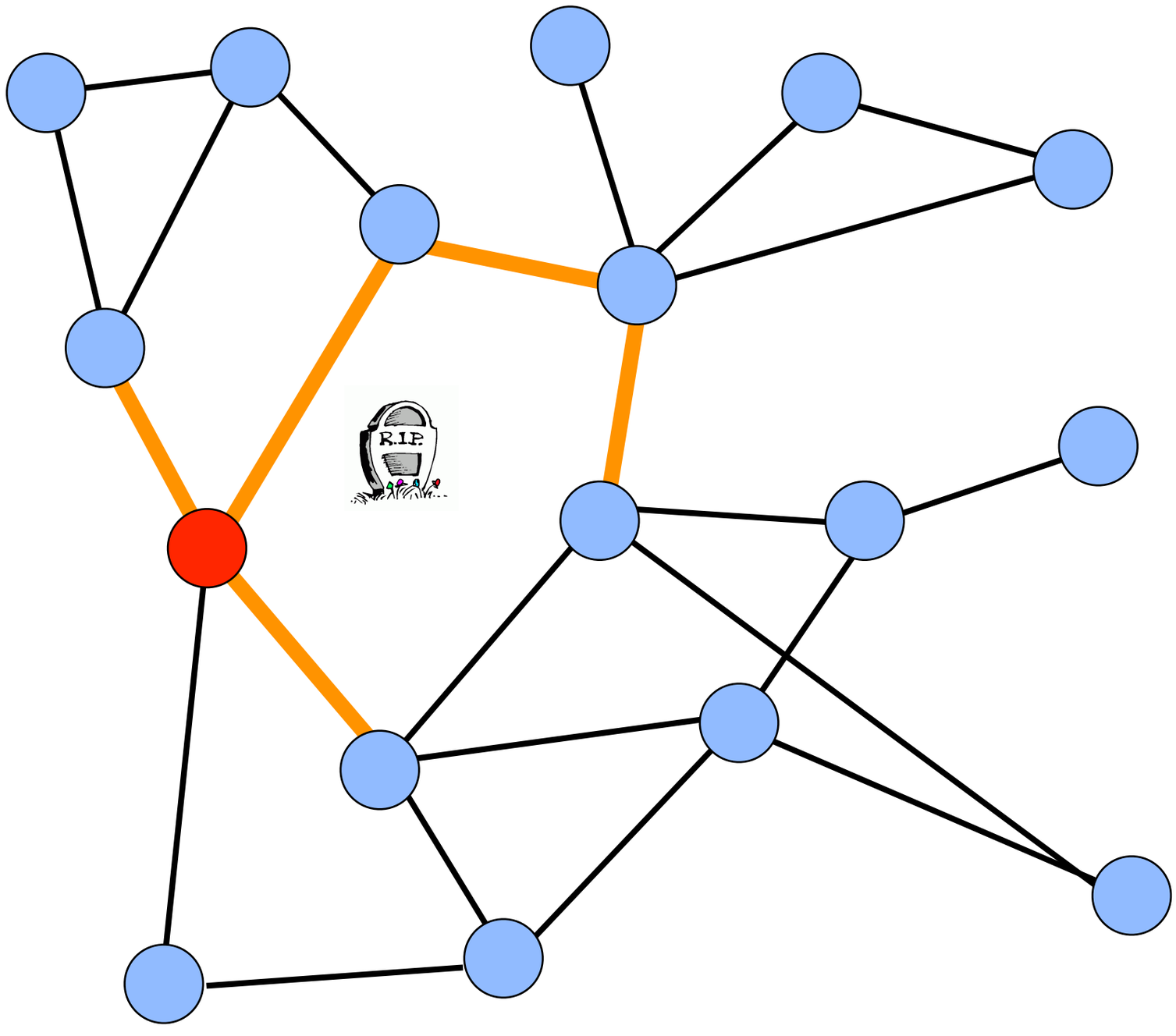
- Start: a network G.
- An adversary inserts or deletes nodes .
- After each node addition/deletion, we can add and/or drop some edges between pairs of nearby nodes, to “heal” the network.

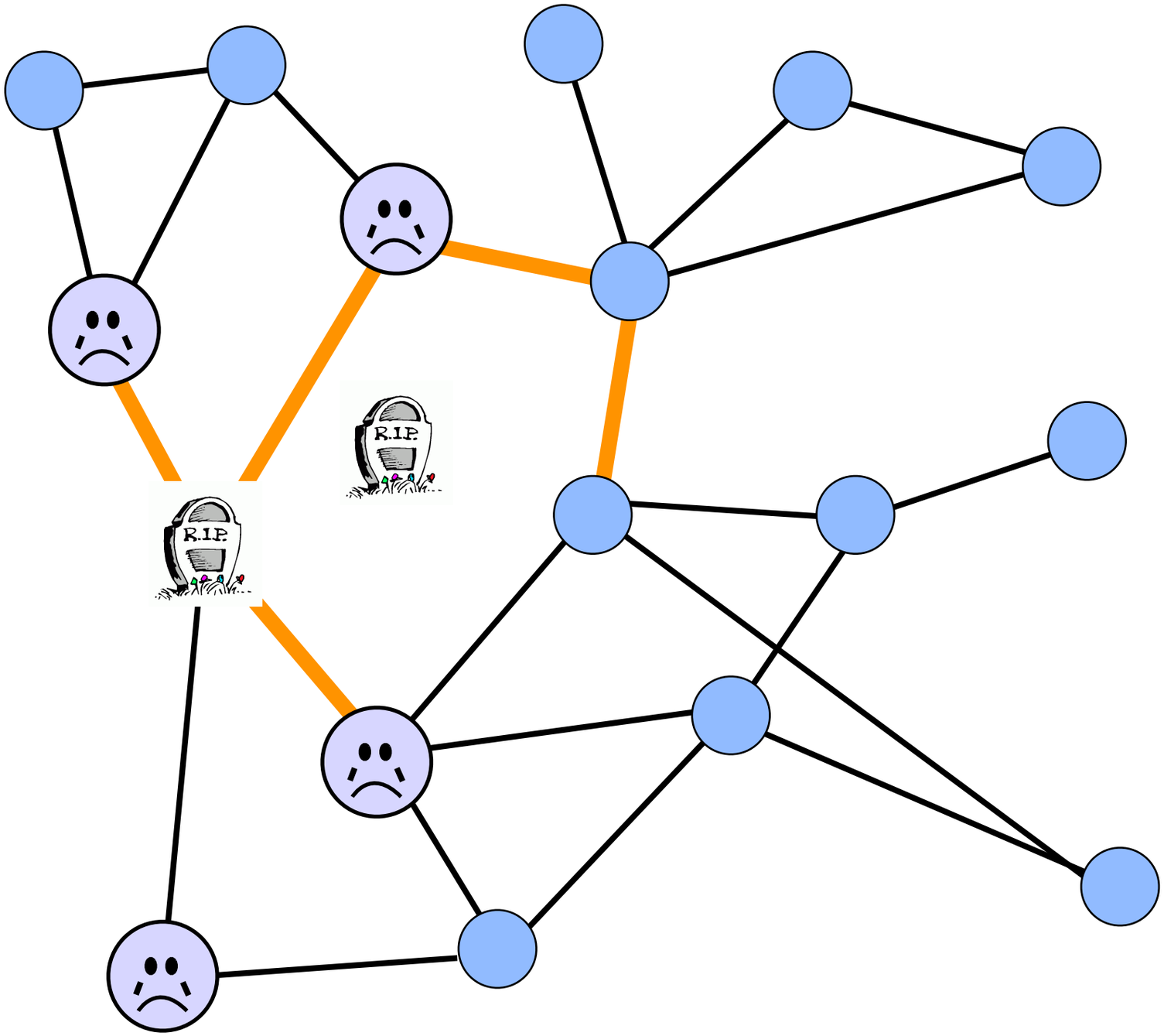
Self-healing illustration

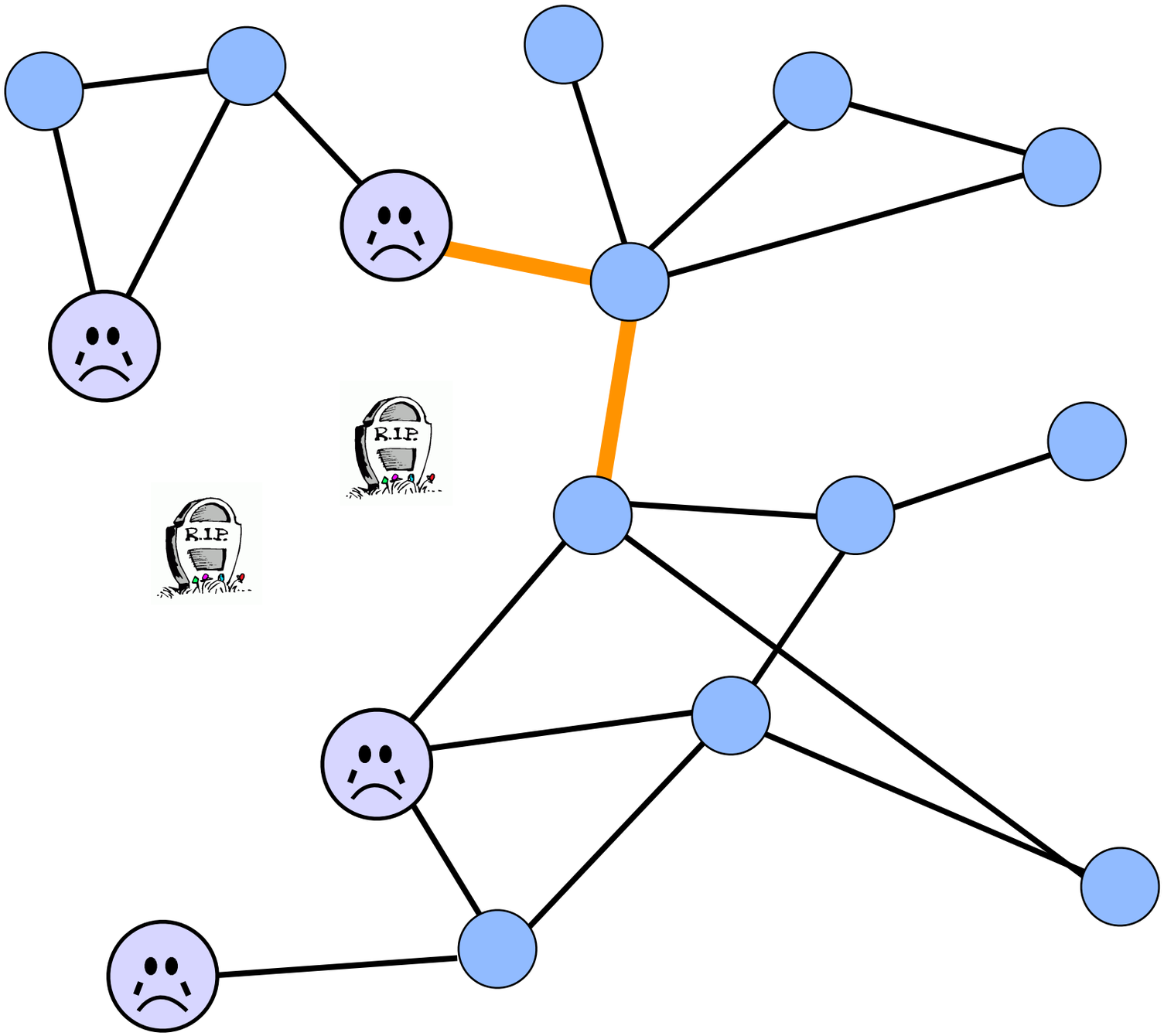



R.I.P.

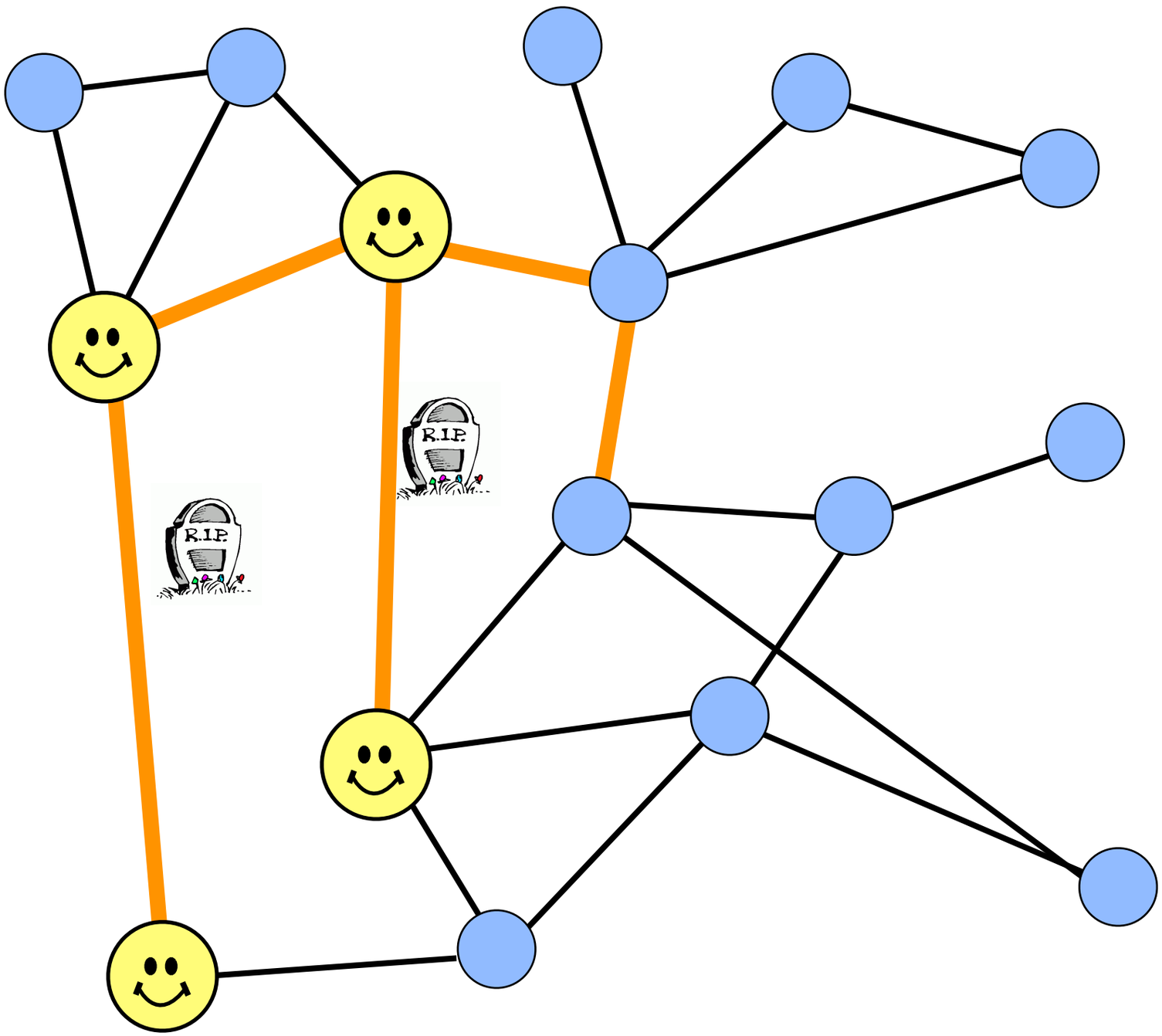

R.I.P.

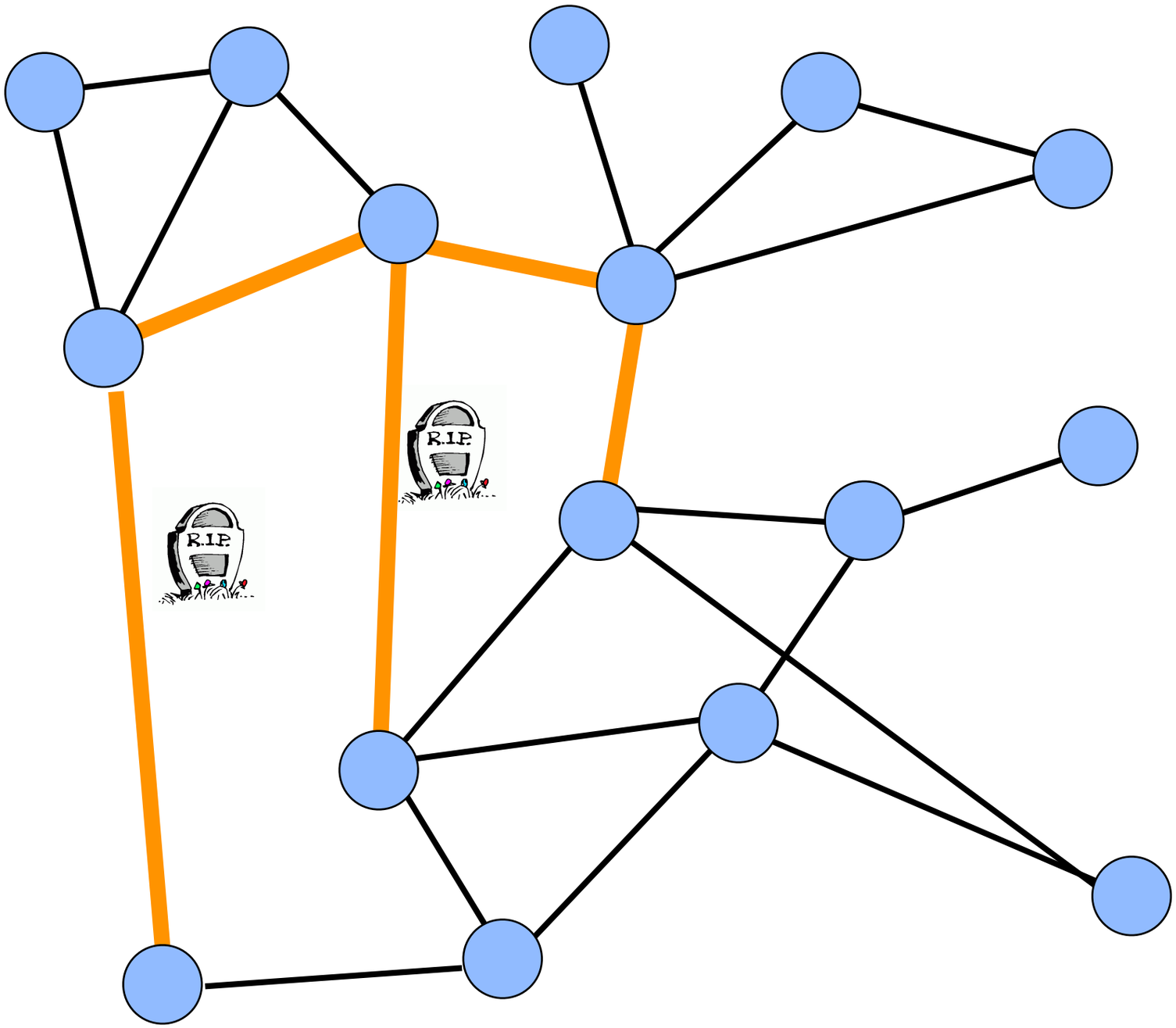

R.I.P.

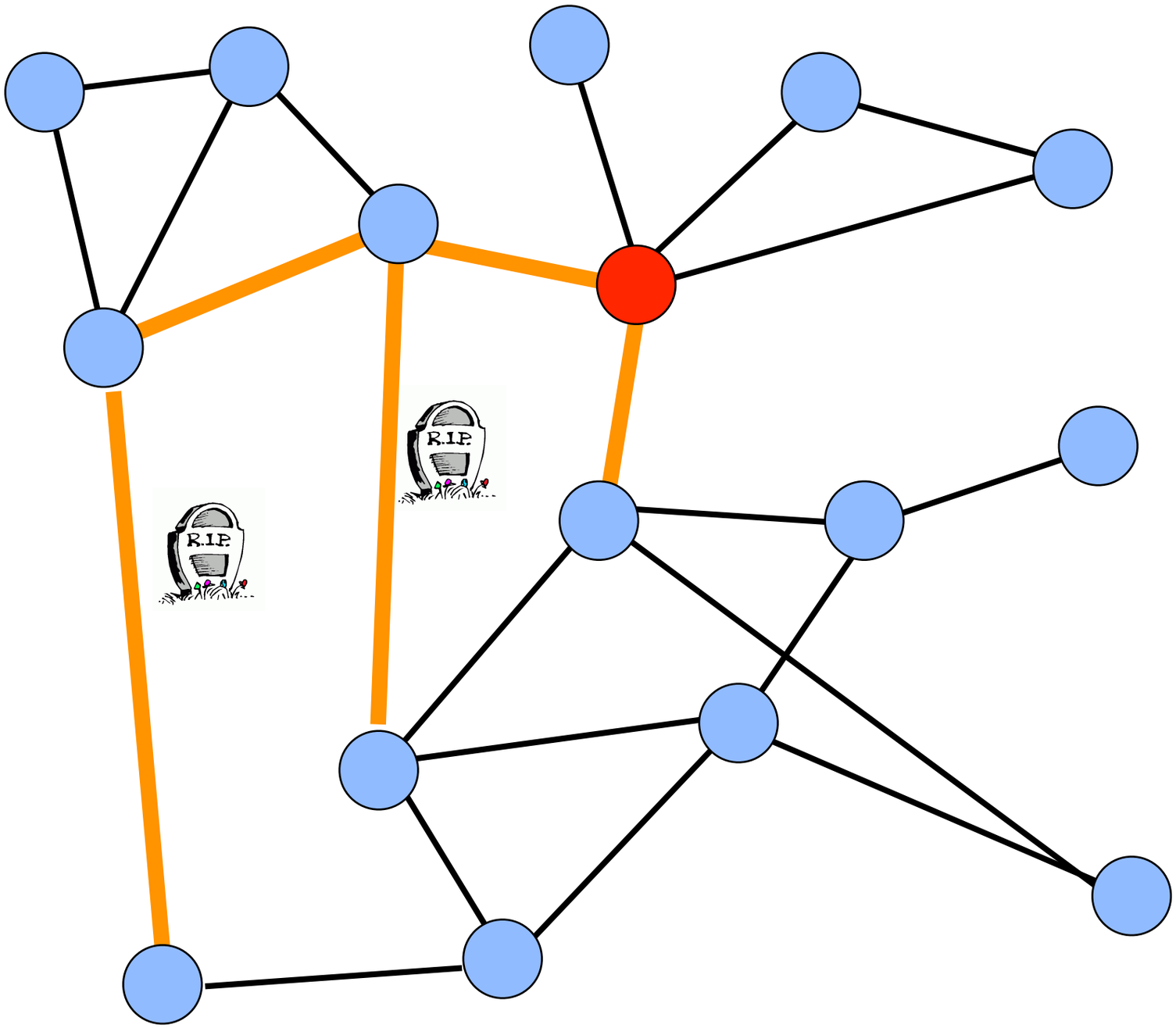

R.I.P.
R.I.P.

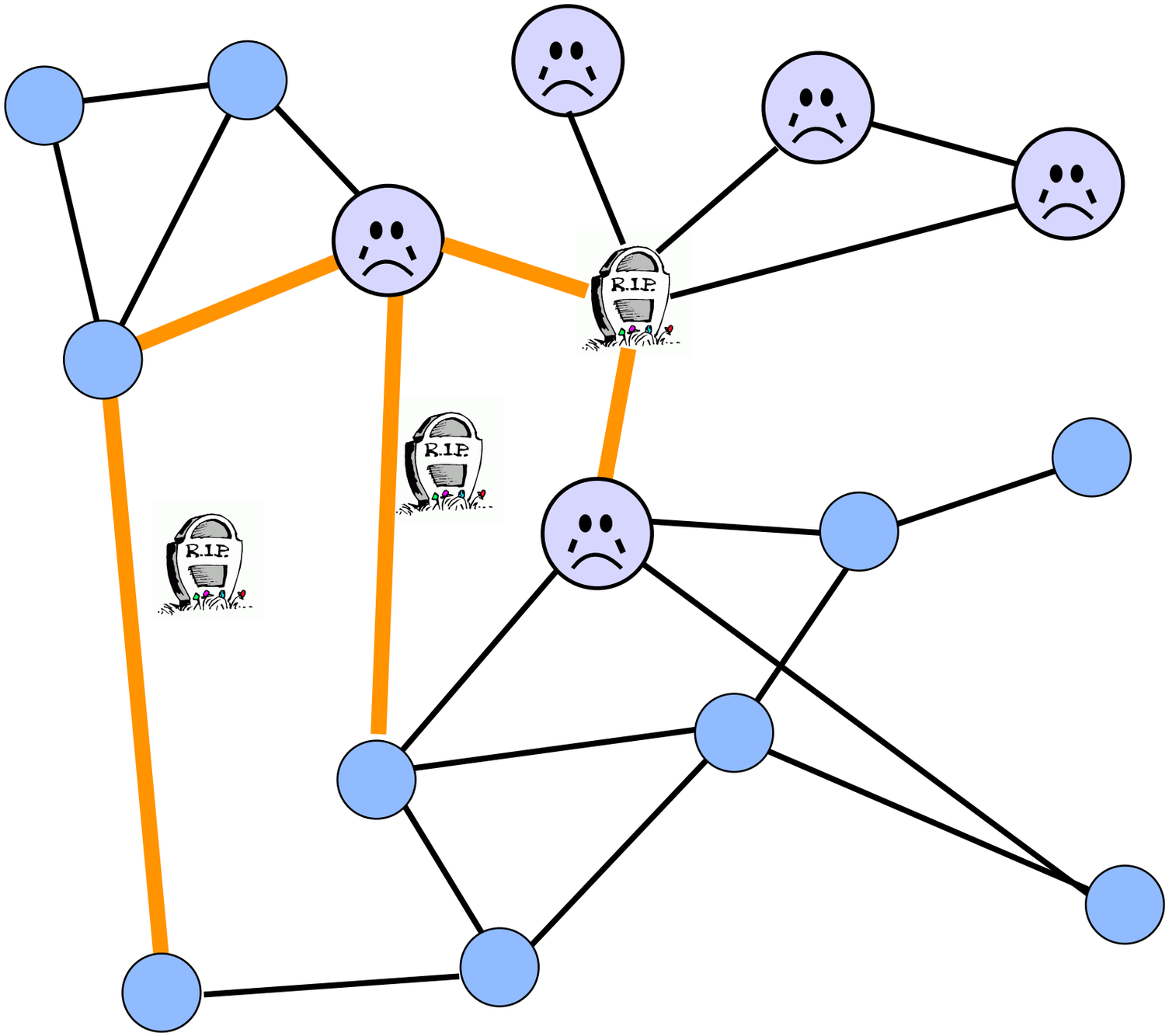

R.I.P.
R.I.P.

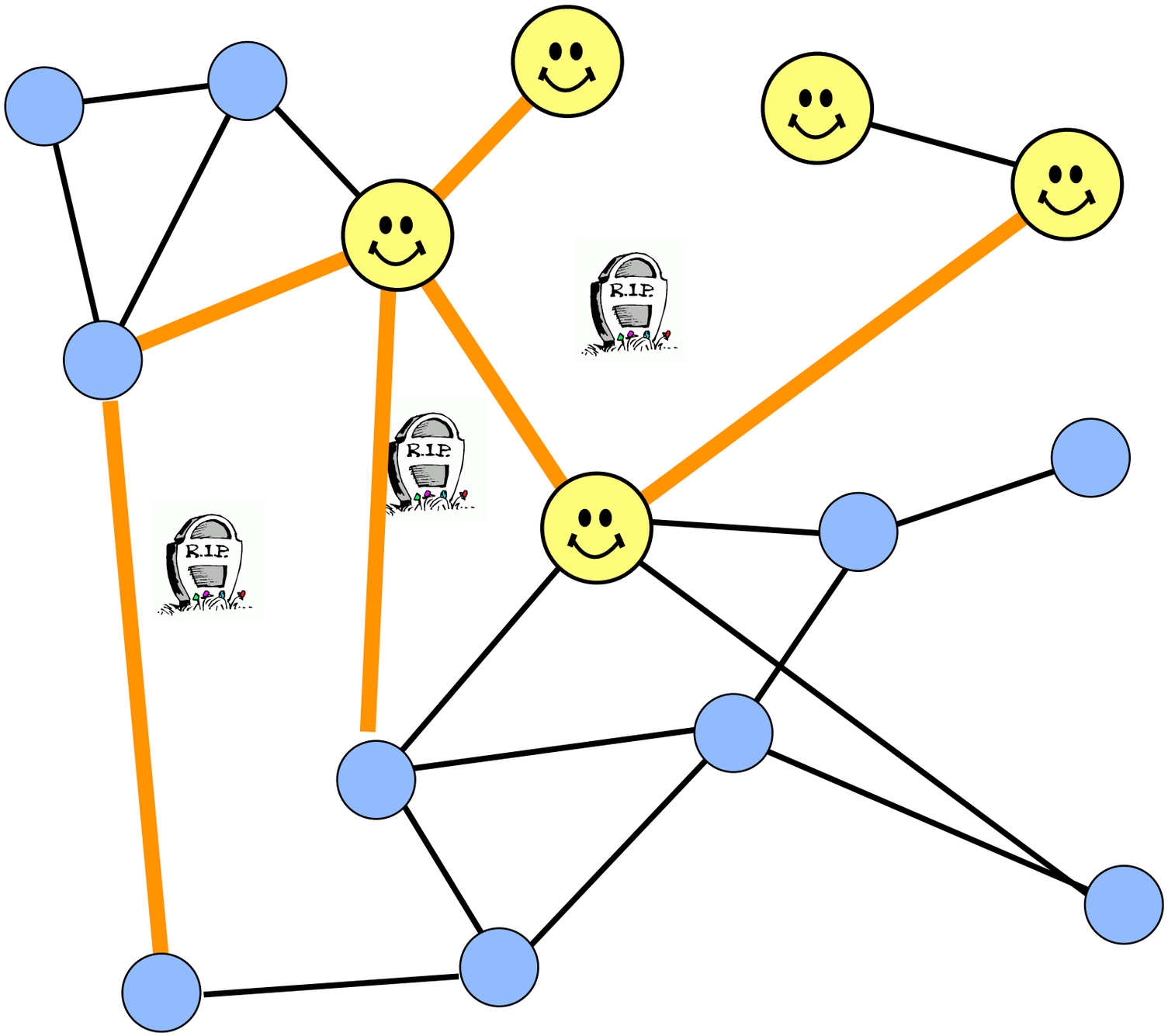
Naive?

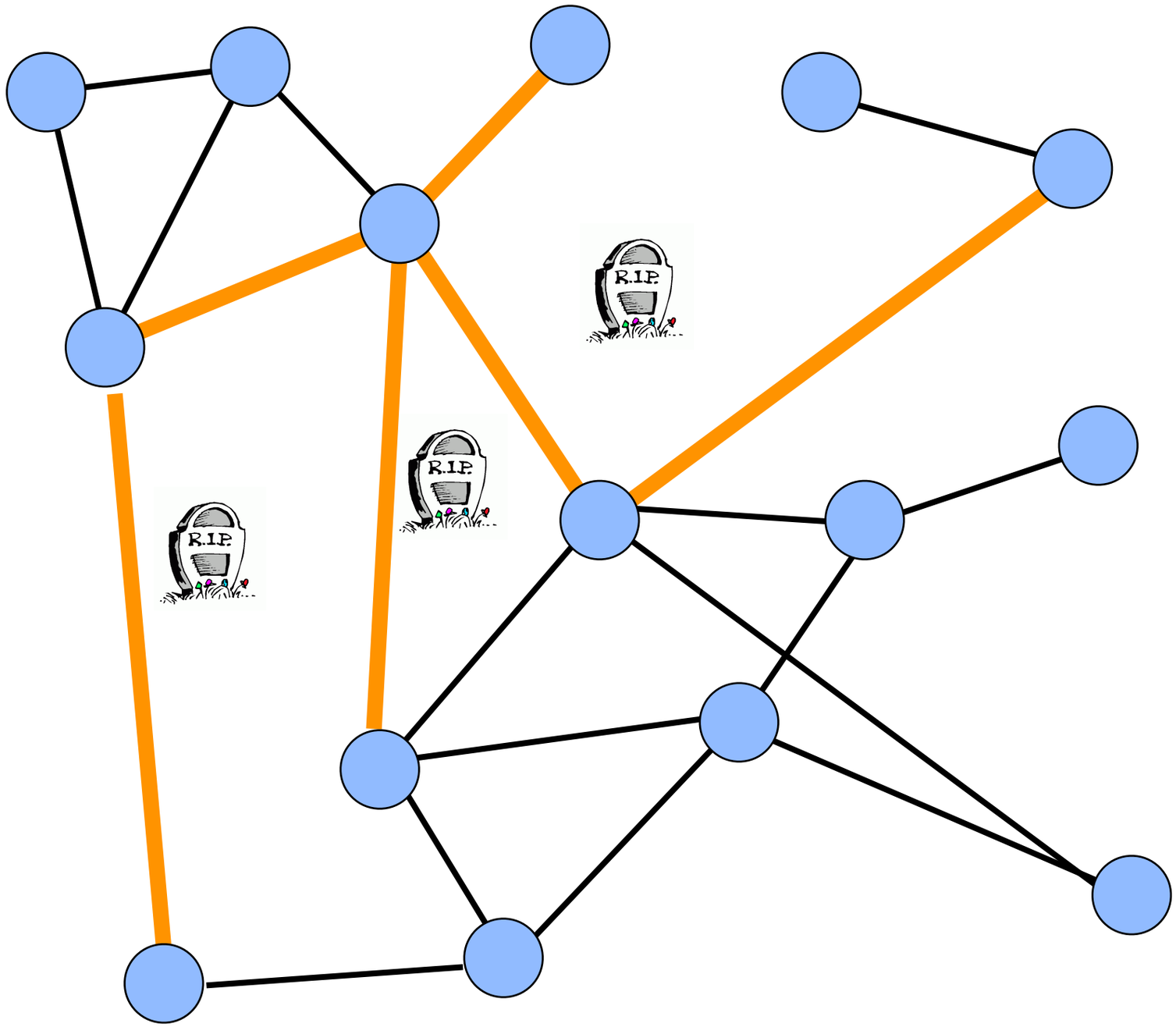


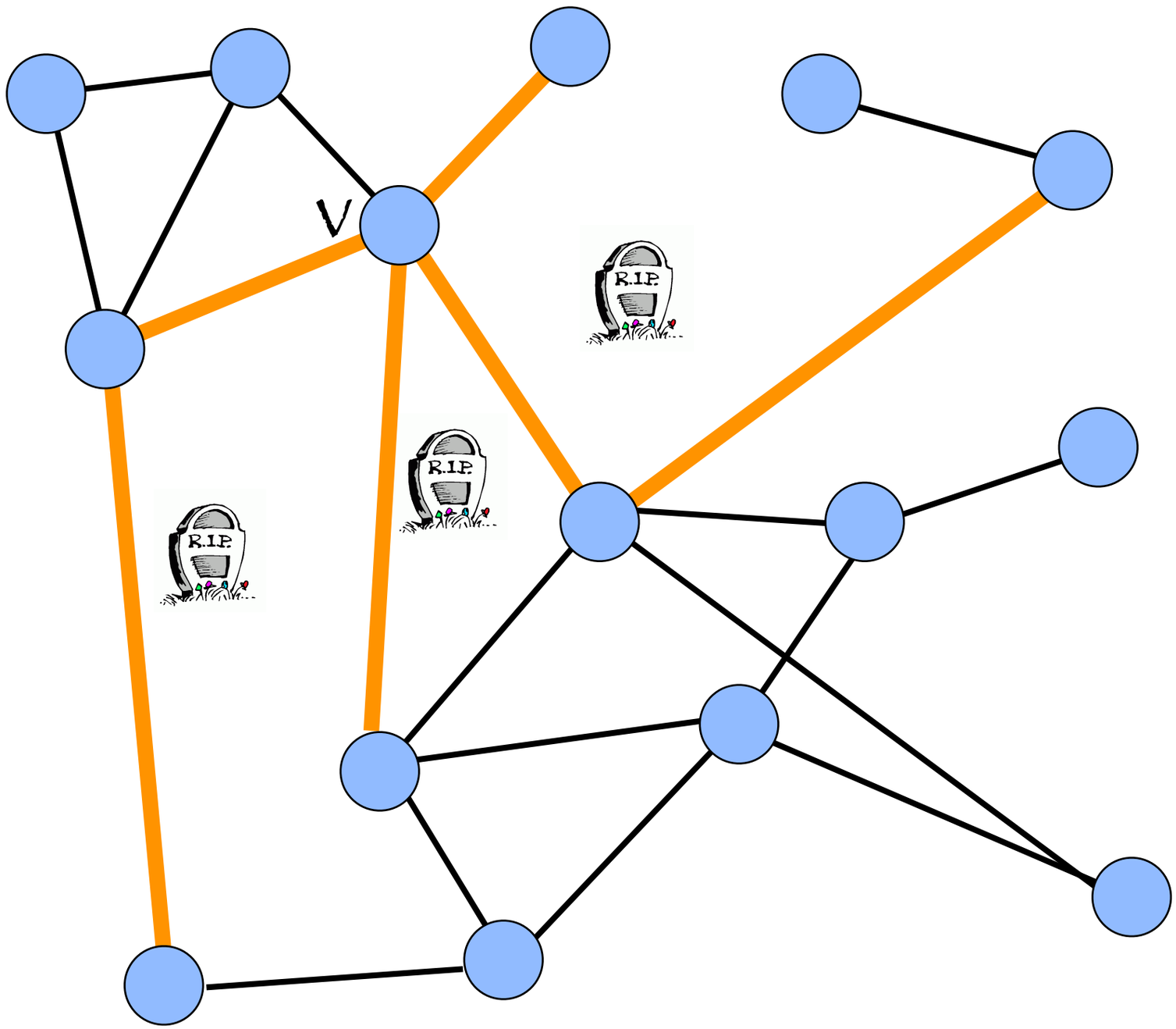

R.I.P.
R.I.P.


R.I.P.
R.I.P.


R.I.P.
R.I.P.
R.I.P.

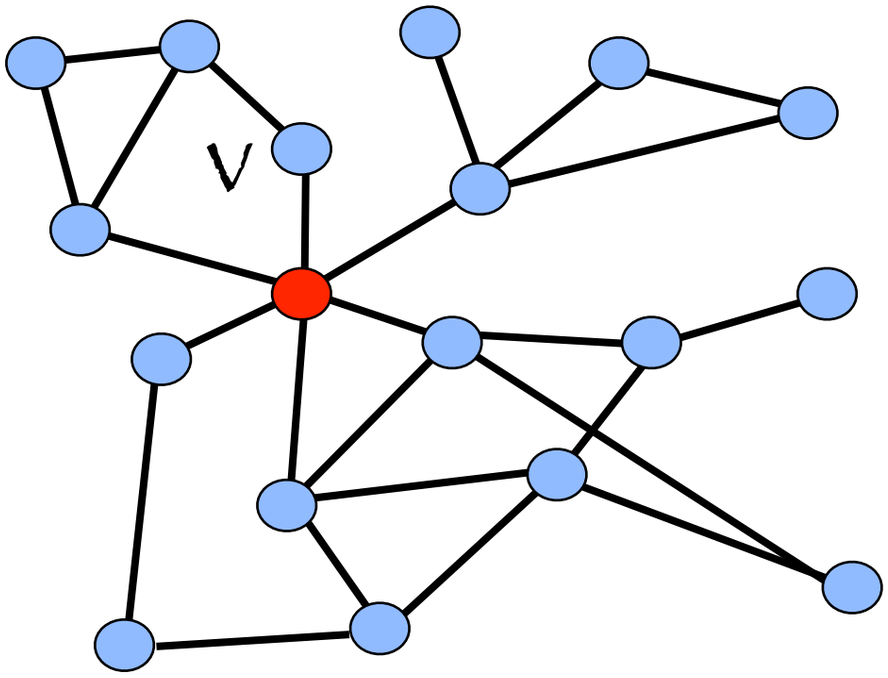

R.I.P.
R.I.P.
R.I.P.


R.I.P.
R.I.P.
R.I.P.

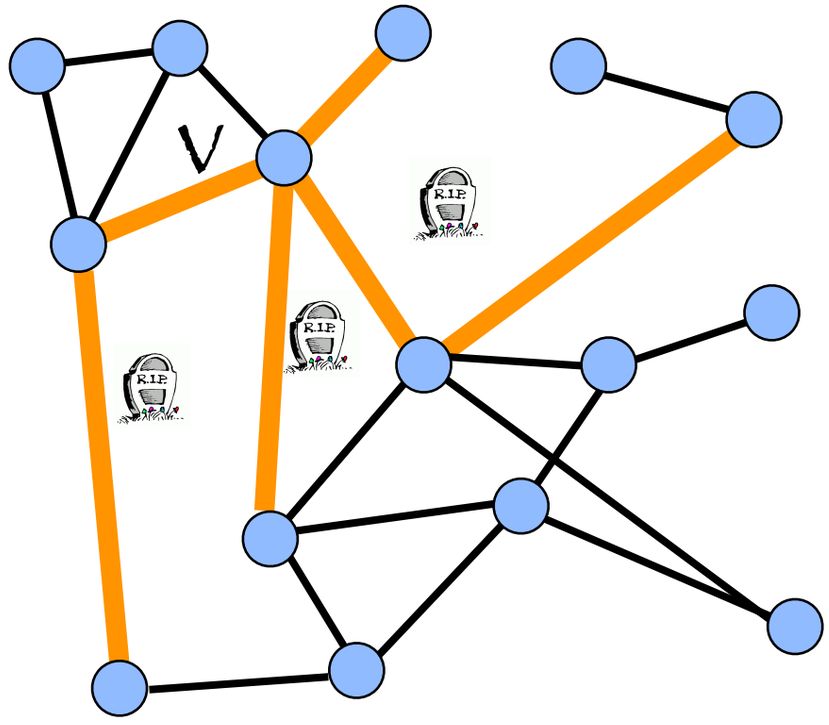


And so on...

Problem



G_0

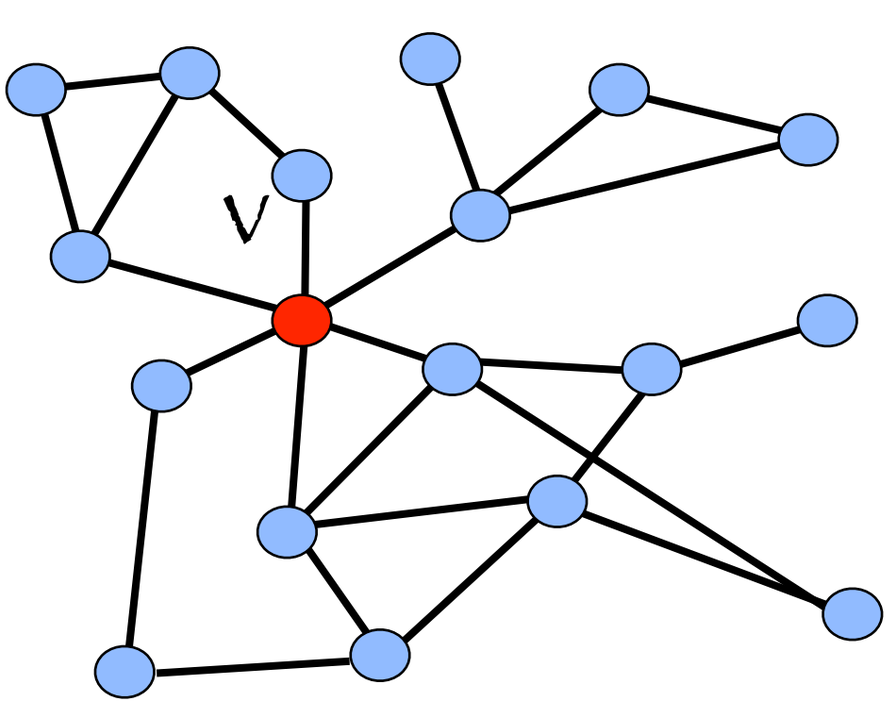


G_3

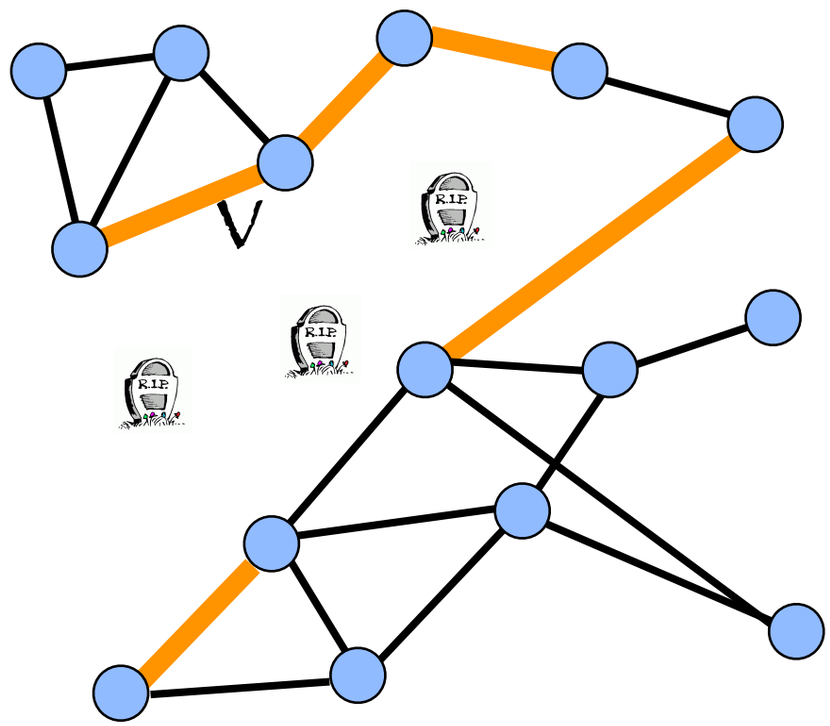
Degree(v,G0) = 2

Degree(v,G3) = 5

Possible healing topologies: Line graph



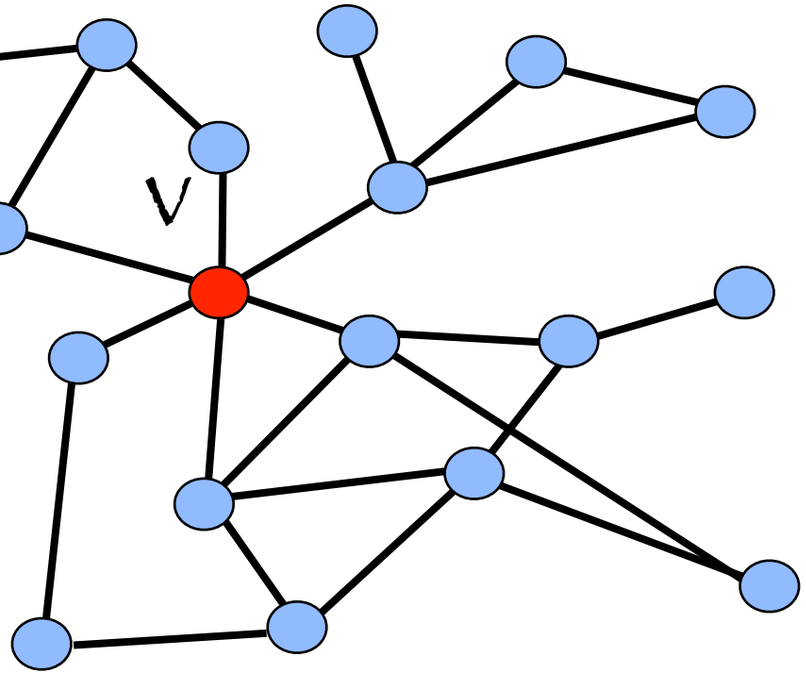
G_0



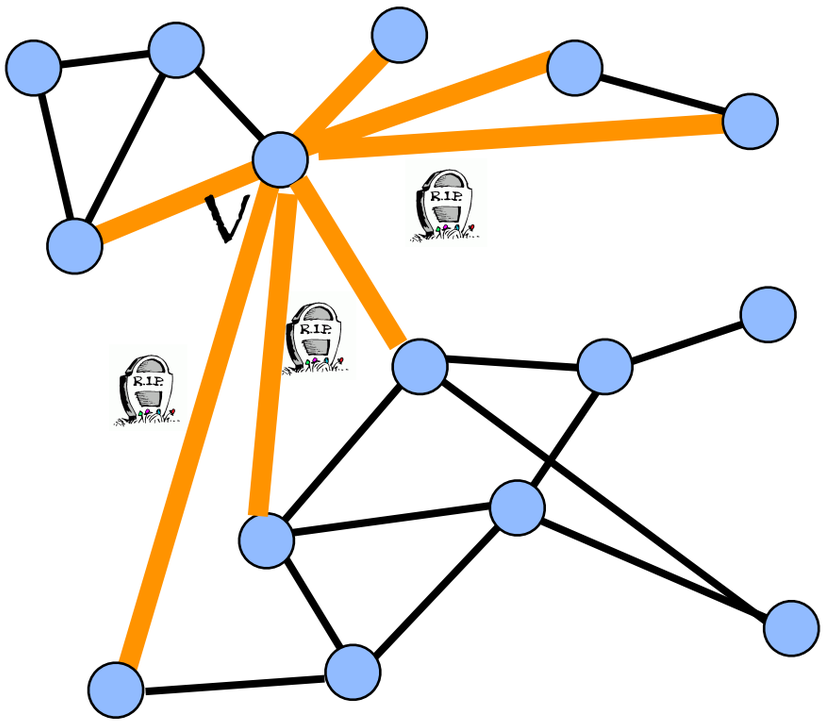
G_3

Low degree increase but diameter/ distances blow up

Possible healing topologies:
Star graph



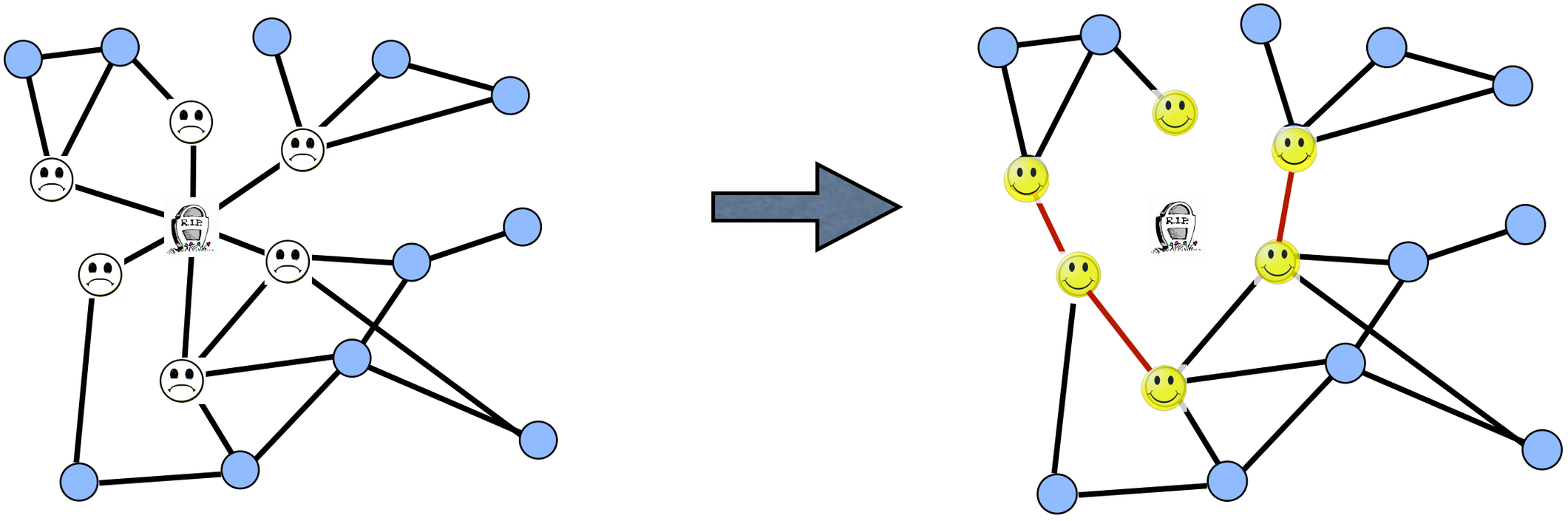
G_0



G_3

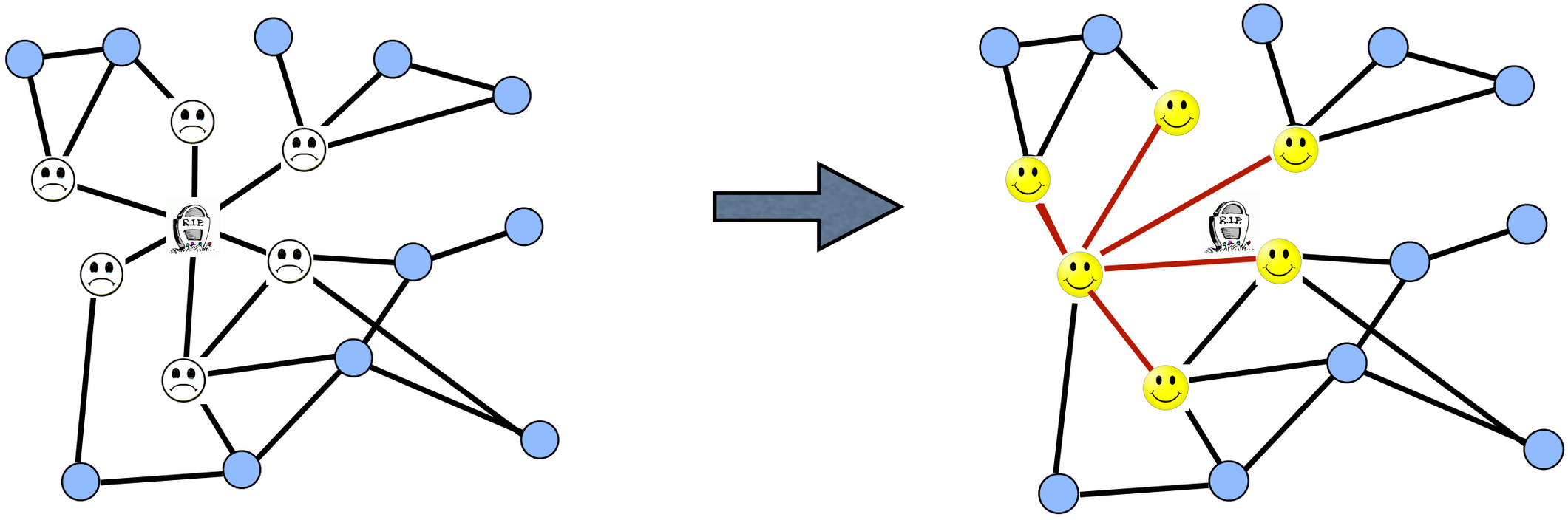
Low distances but degree blows up

Challenge 1: properties conflict



Low degree increase => high diameter/ stretch/ poorer expansion?

Challenge 2: local fixing of global properties



Low diameter => high degree increase?

- Limited global Information with nodes
- Limited resources and time constraints

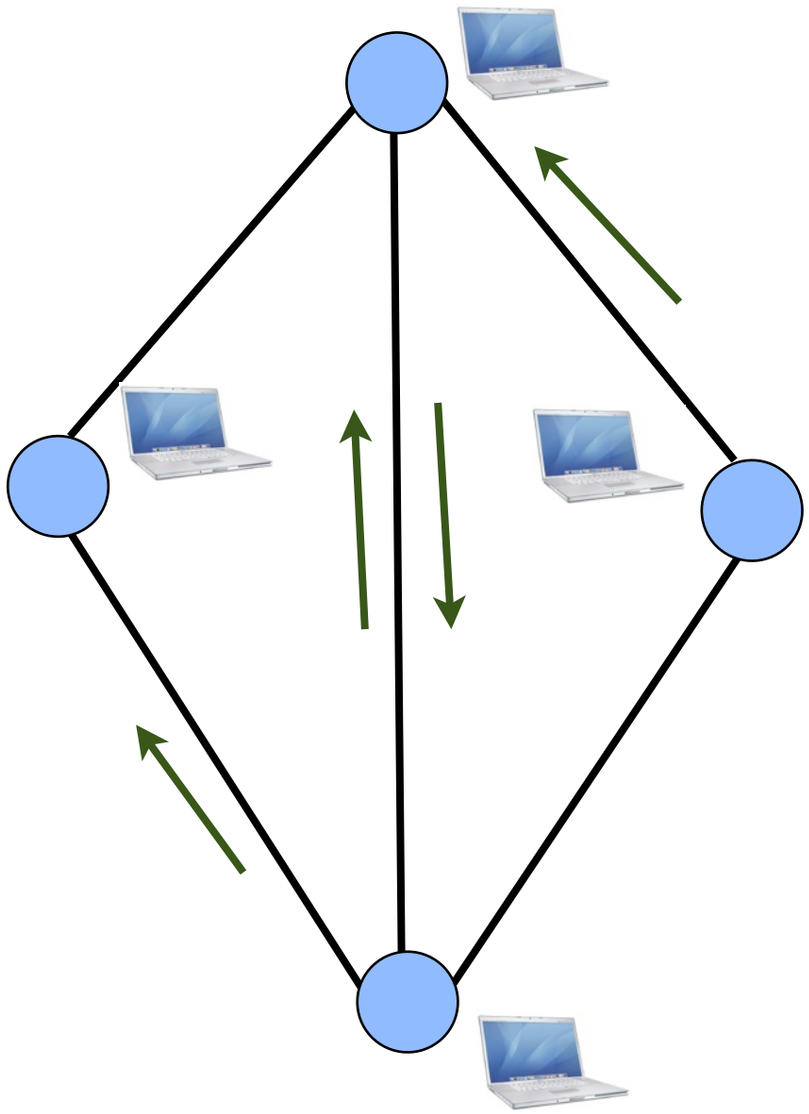
Self-healing Goals

- Healing should be very fast.
- Certain topological properties should be maintained within bounds:
  - Connectivity
  - Degree
  - Diameter/ Stretch

Our SH algorithms

- **DASH** *[IEEE International Parallel & Distributed Processing Symposium 2008]*
- **Forgiving Tree** *[ACM Principles of Distributed Computing, 2008]*
- **Forgiving Graph** *[ACM Principles of Distributed Computing, 2009, Journal of Distributed Computing, 2012]*

Distributed Computing: Message Passing model



- Communication is by sending messages along edges
- Only local knowledge to begin with

Goals

- Ensure connectivity
- Healing should be very fast (constant time)
- If vertex v starts with degree d, then its degree should never be much more than d
- Diameter shouldn't increase by too much

The Forgiving Tree: Model

- Start: a network G.
- Nodes fail in unknown order $v_1, v_2, ..., v_n$
- After each node deletion, we can add and/or drop some edges between pairs of nearby nodes, to "heal" the network

The Forgiving Tree: Main Result

- A distributed algorithm, Forgiving Tree such that, for any network G with max degree D, for an arbitrary sequence of deletions,
- Graph stays connected
- Diameter increases by $\leq$ log(D) factor
- Degrees increase by $\leq$ 3 (additive)
- Each repair takes constant time and involves O(D) nodes.

The Forgiving Tree: Main Result

- A distributed algorithm, Forgiving Tree such that, for any network G with max degree D, for an arbitrary sequence of deletions,
- Graph stays connected
- Diameter increases by $\leq \log(D)$ factor
- Degrees increase by ≤ 3 (additive)
- Each repair takes constant time and involves O(D) nodes.

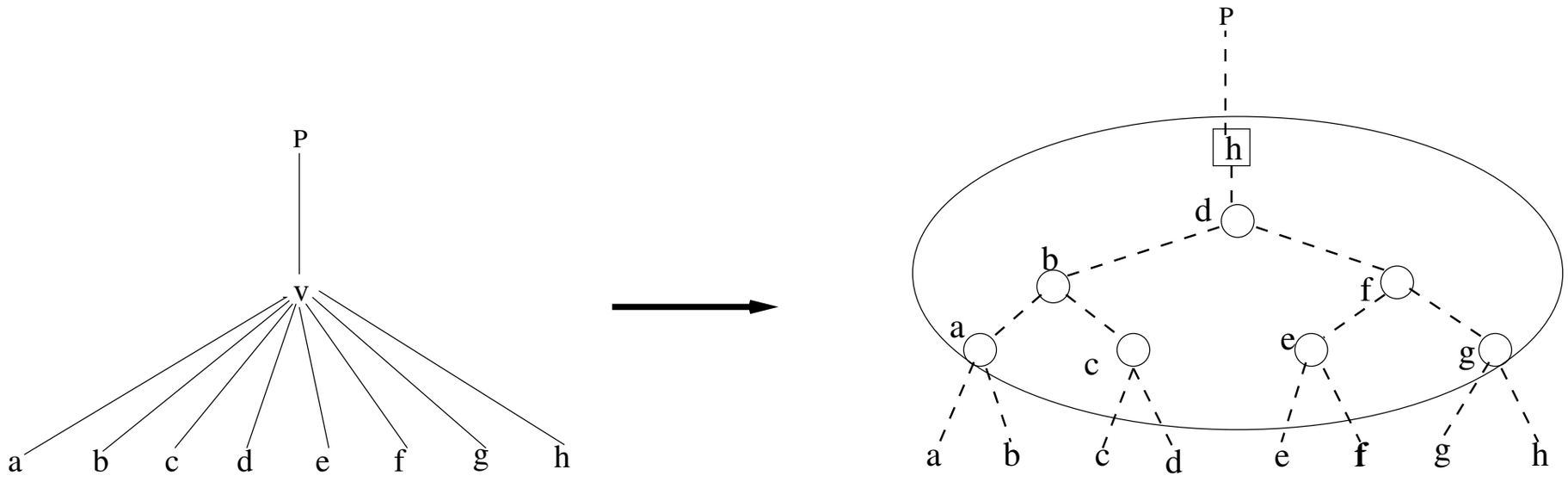
} Matching lower bound

The Forgiving Tree: motivations

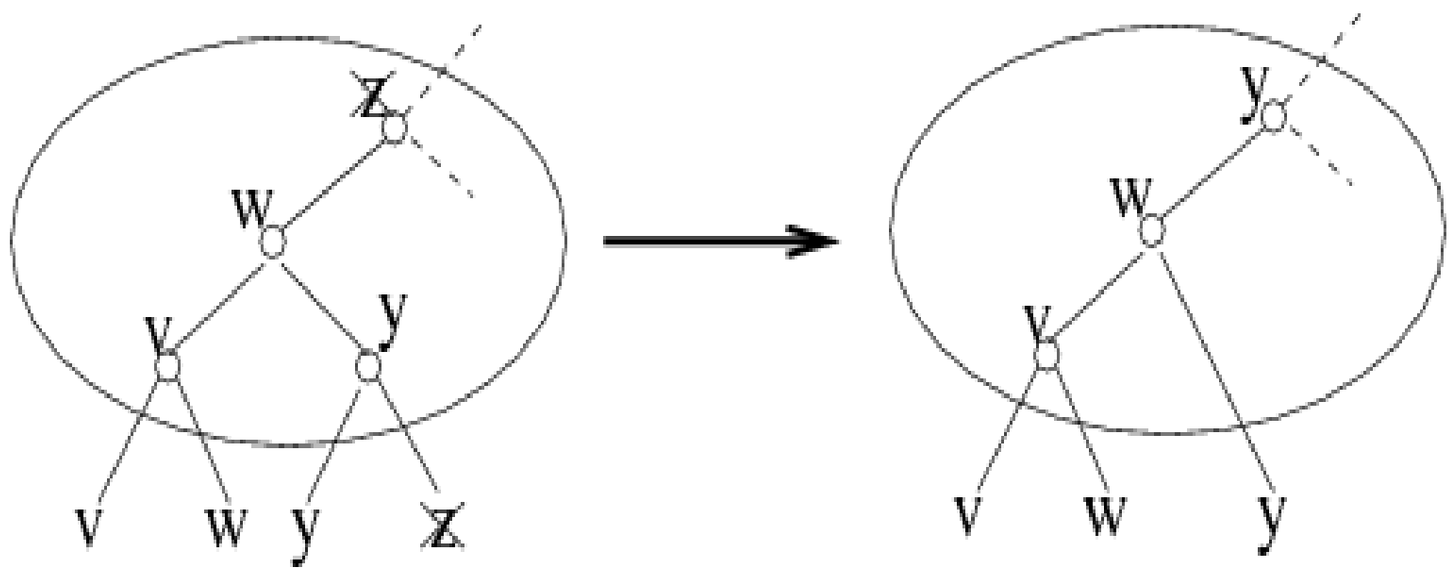
- Trees are the "worst case" for maintaining connectivity. Suppose we are given one.
- *Maintain a virtual tree:* Tracks vertex degrees and also avoid blowing up distances.

FT: first approximation

- Find a spanning tree of G.
- Choose some vertex to be the root, and orient all edges toward the root.
- When a node is deleted, replace it by a balanced binary tree of "virtual nodes"
- Short-circuit any redundant virtual nodes
- Somehow the surviving real nodes simulate the virtual nodes



Replacing v by a balanced binary tree of virtual nodes



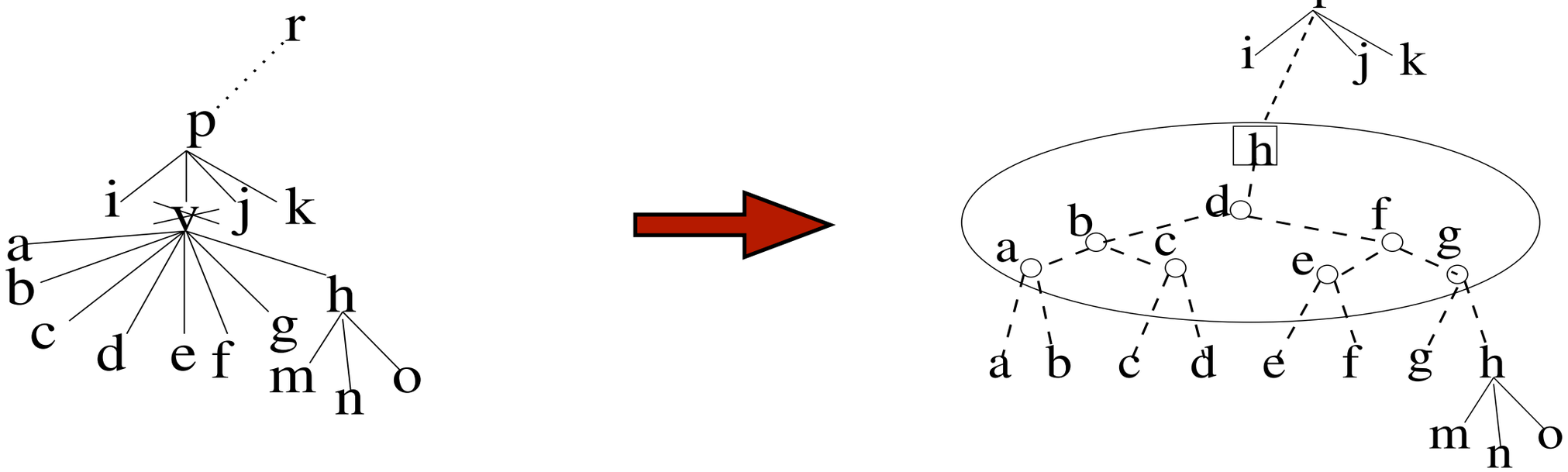
Short-circuiting a redundant virtual node

Virtual Nodes

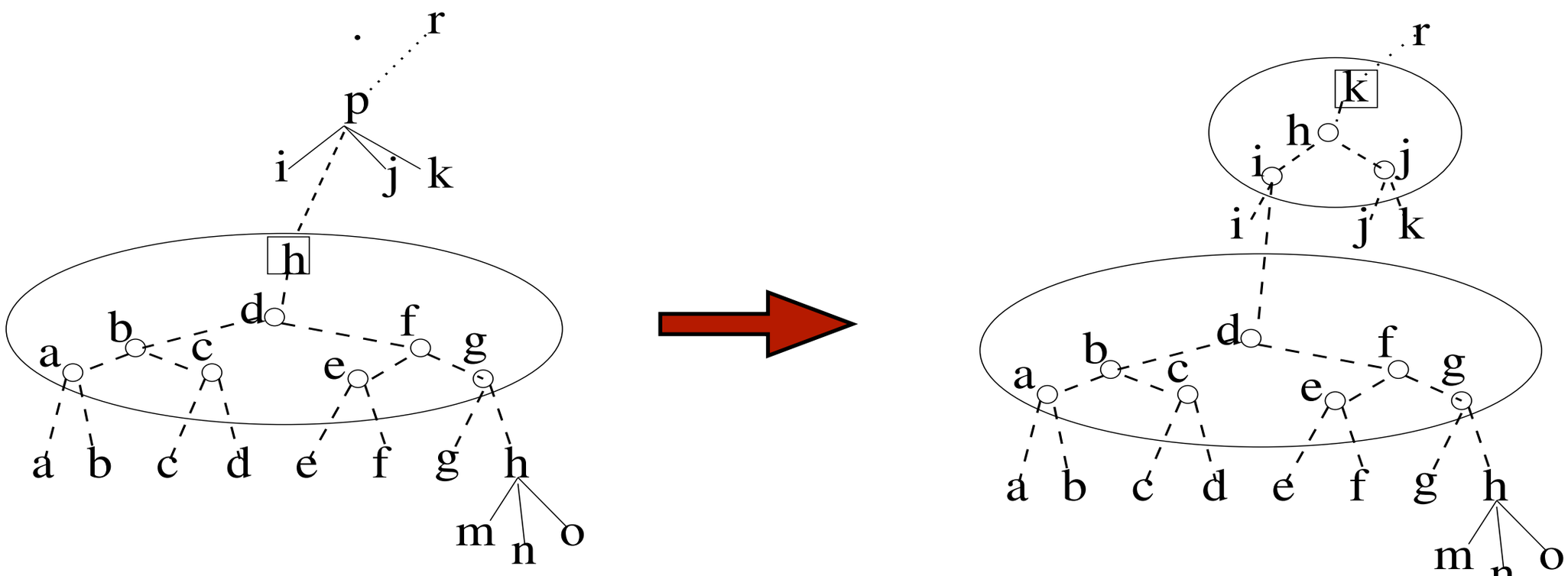
- A virtual node starts with degree 3, since internal node of a complete binary tree.
- If a neighbor is a leaf, and is deleted, the virtual node becomes redundant. Then we "short-circuit" it.
- This ensures that there are always more real nodes than virtual nodes. Each real node needs to simulate at most one virtual.

Algorithm in action

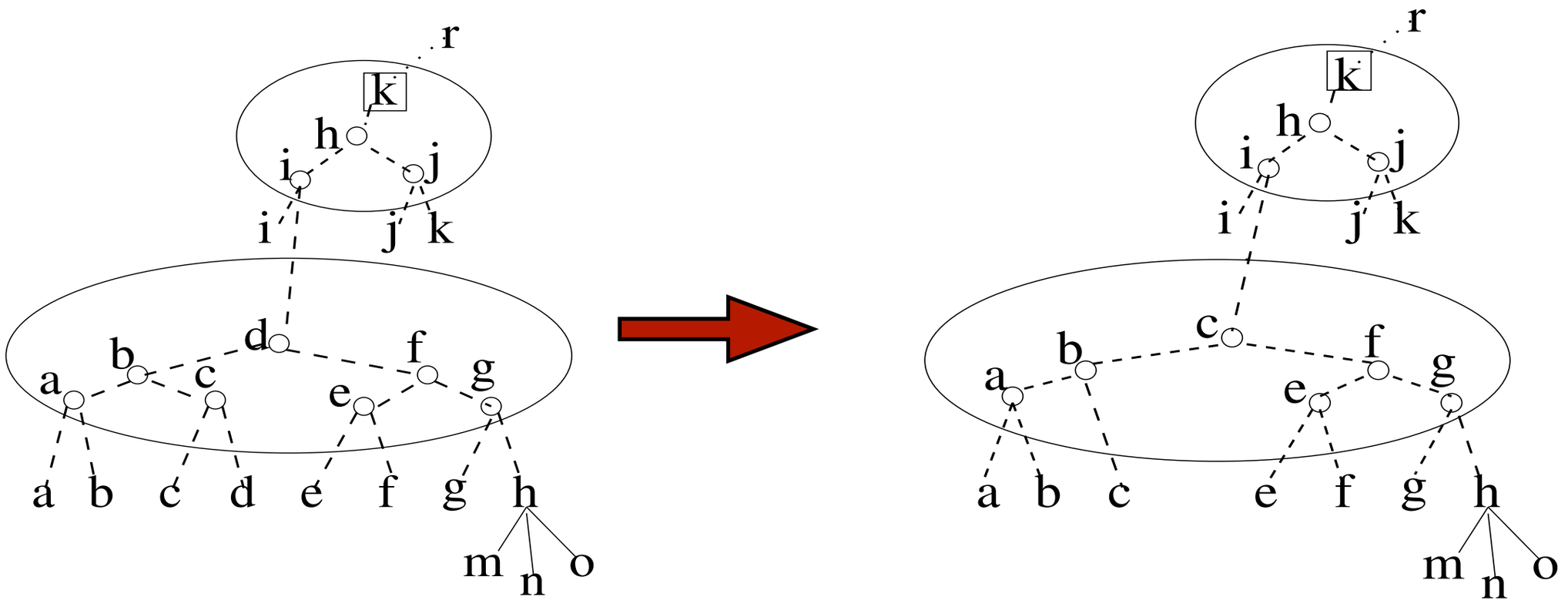
Node v deleted:



Node p deleted:



Node d deleted:



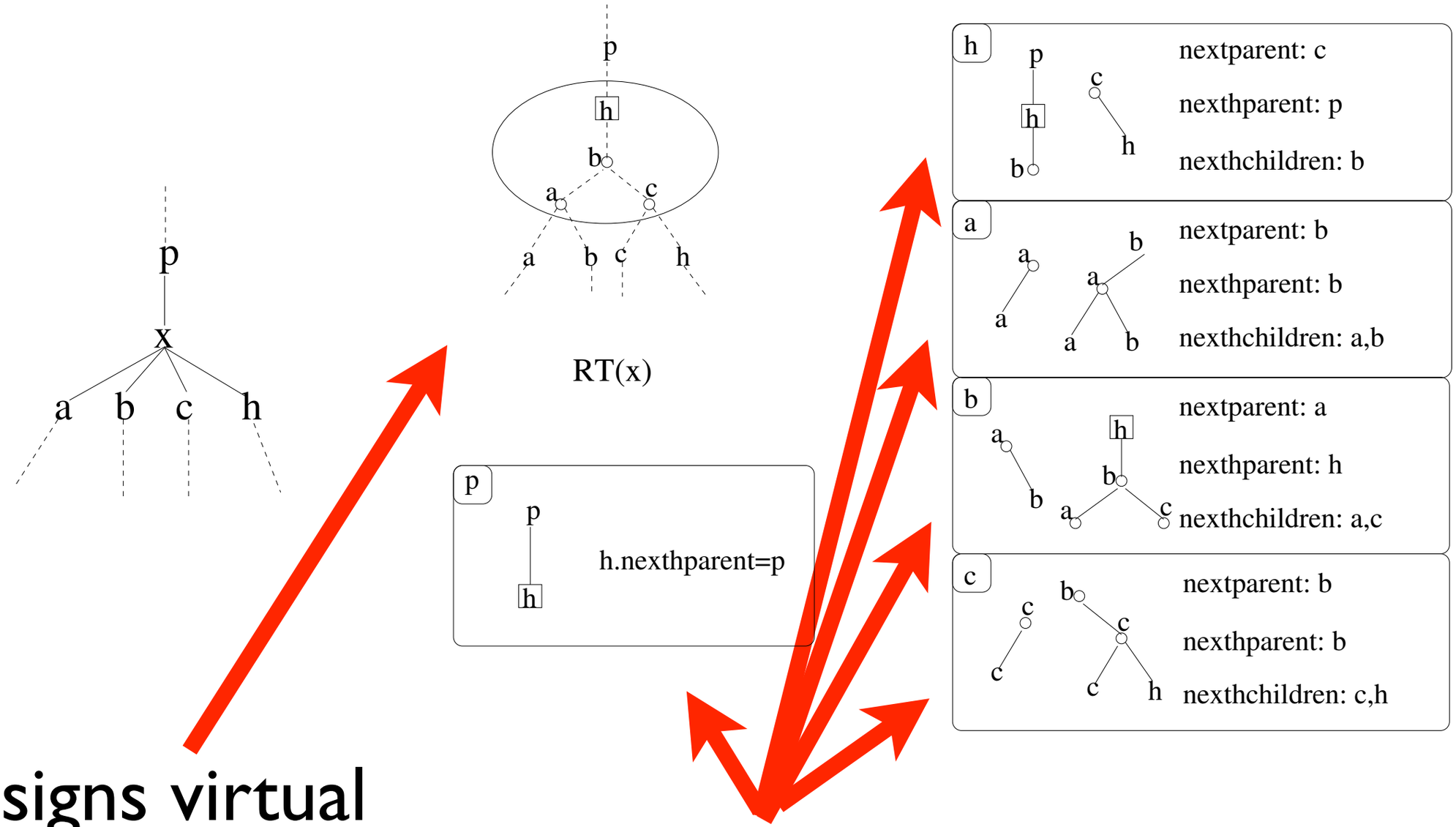
Caveats

- Each virtual node somehow gets assigned to a surviving real node. (next few slides)
- The actual healed network is just the homomorphic image of the restored tree under this map. (later).
- We won't go into full detail about how to implement this in a distributed way.

Assigning virtual duties

- At the start of the algorithm, each non-leaf node v writes a "will" specifying one child to simulate each virtual node which will be created when v is deleted.
- The relevant piece of this will is entrusted to each child
- v may revise the will from time to time
- When v is deleted, the will is implemented

Where there's a will...



x assigns virtual nodes in his will

h is the "heir"

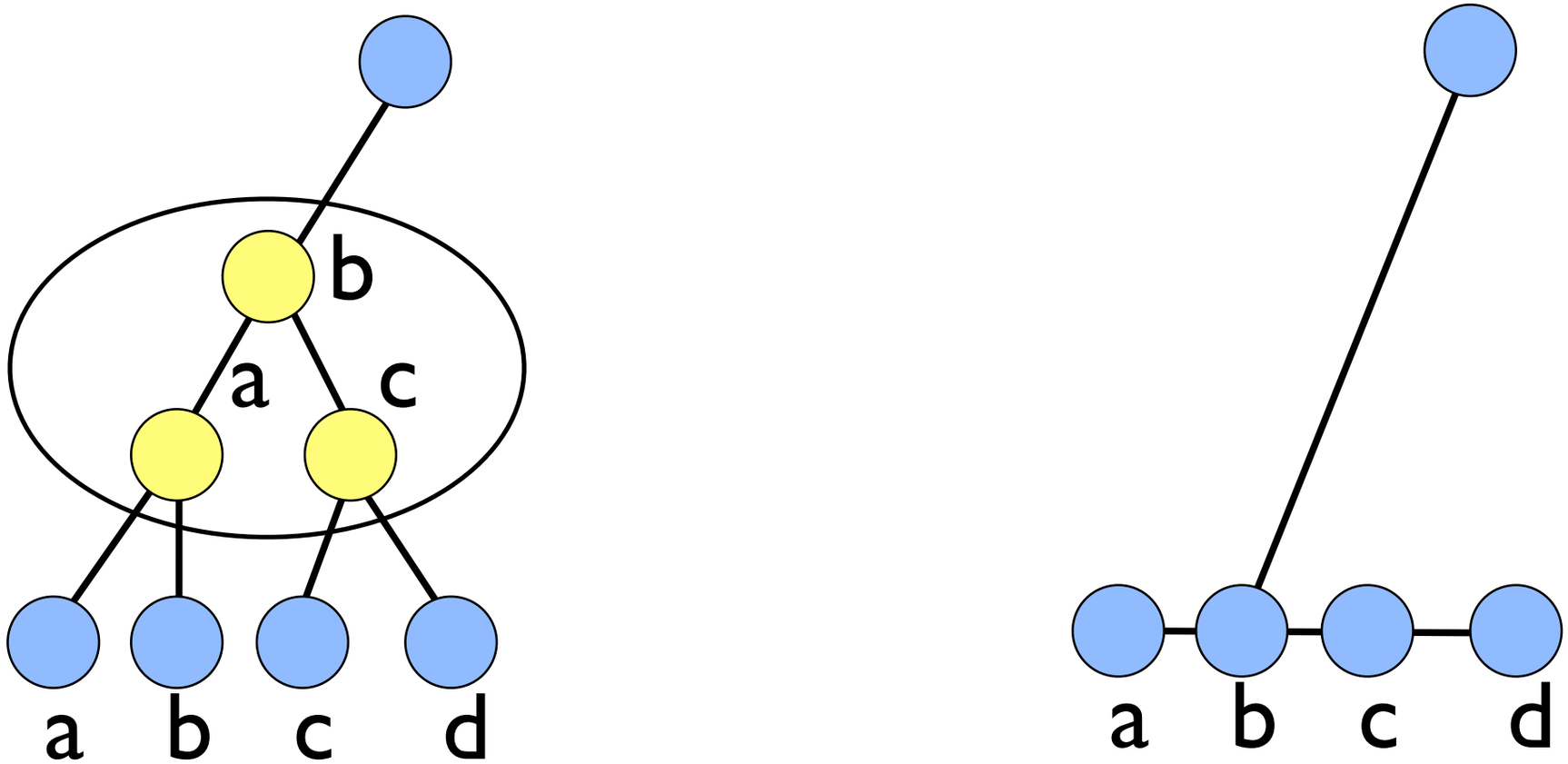
The will gets split into 5 parts--one for each neighbor.

Heirs

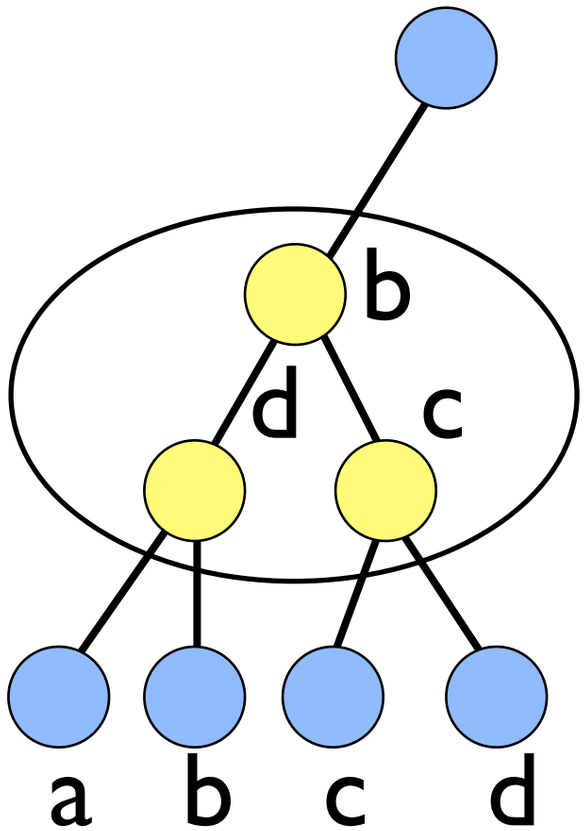
- What happens when the node simulating a virtual node is deleted?
- (a) if the virtual node has become redundant, it is short-circuited as usual.
- (b) if the simulating node w has children, then w's will designates a child to take over the virtual node. This is w's "heir"

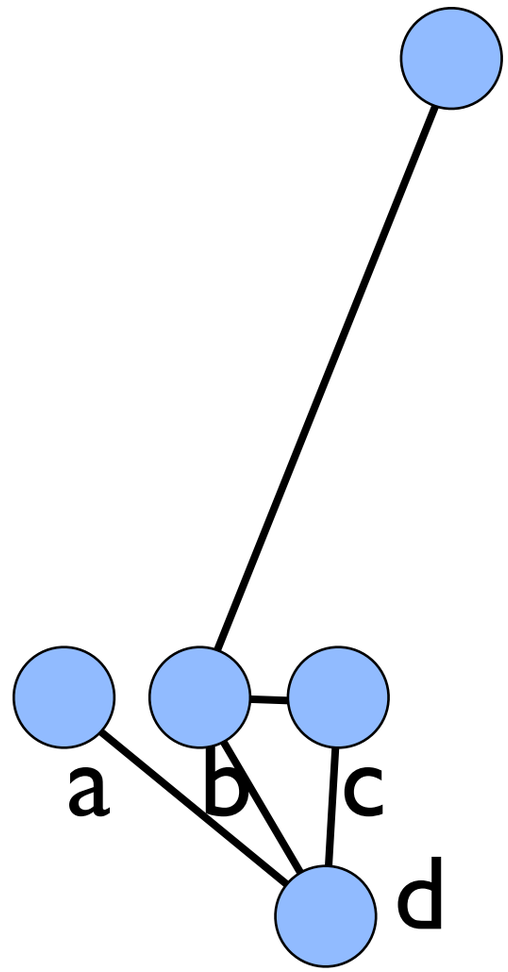
Homomorphism: Given $G_1 = (V_1, E_1), G_2 = V_2, E_2$

a map such that $\{v, w\} \in E_1 \Rightarrow \{f(v), f(w)\} \in E_2$



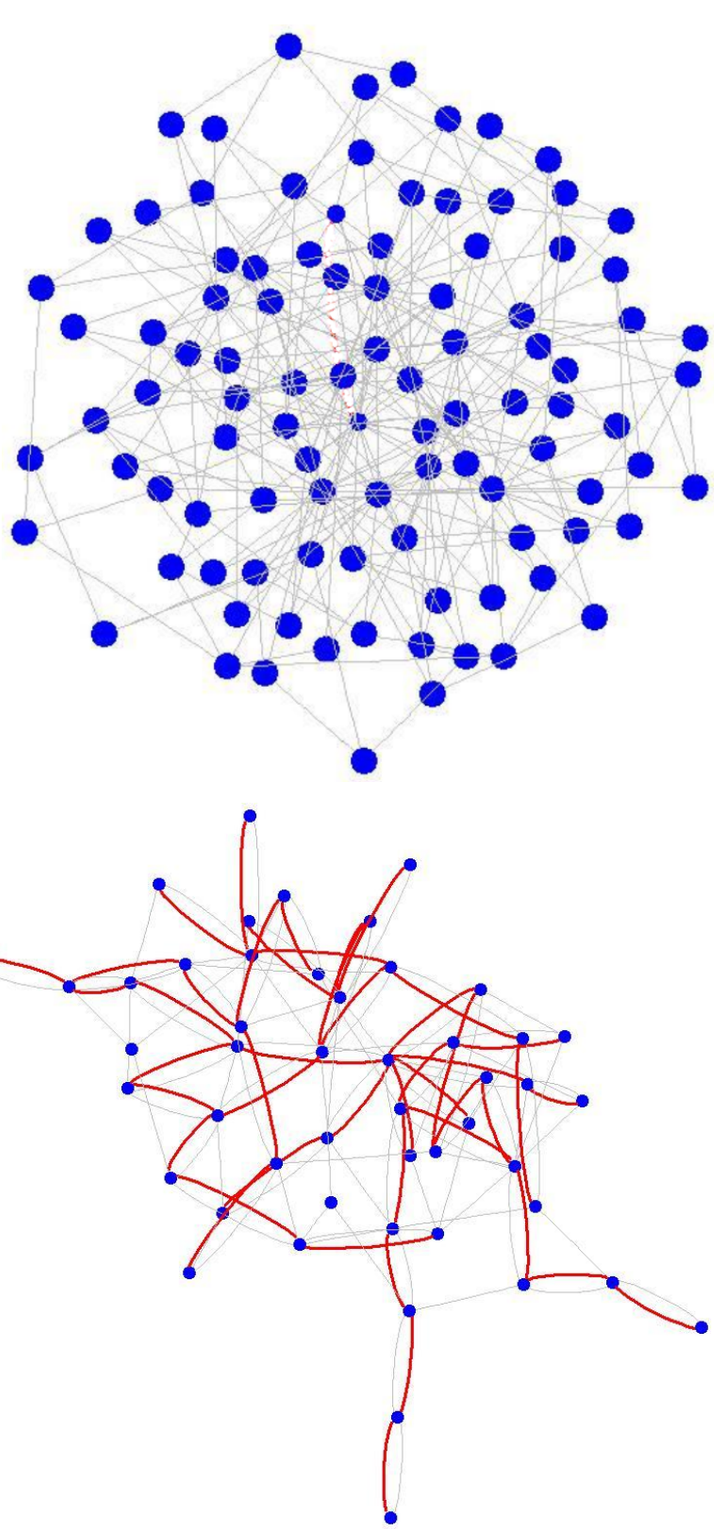
A virtual tree (left) and its homomorphic image (right)

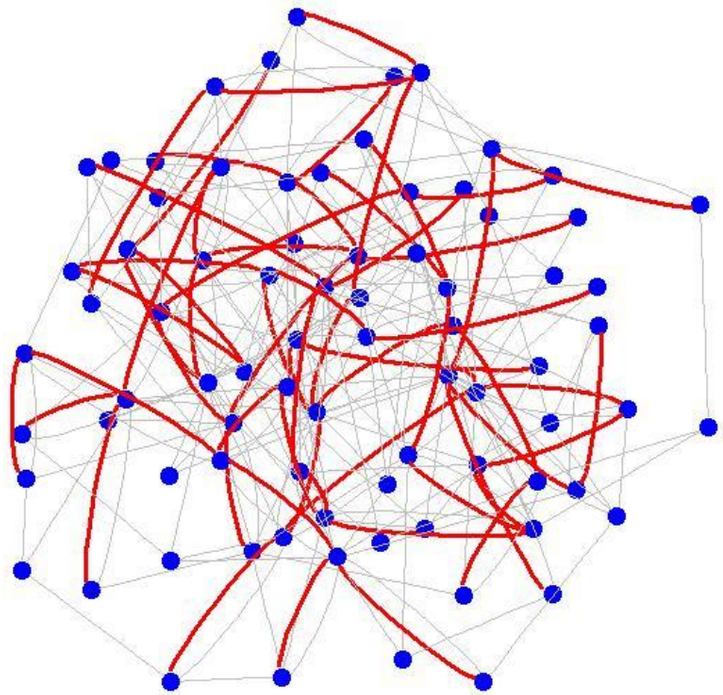


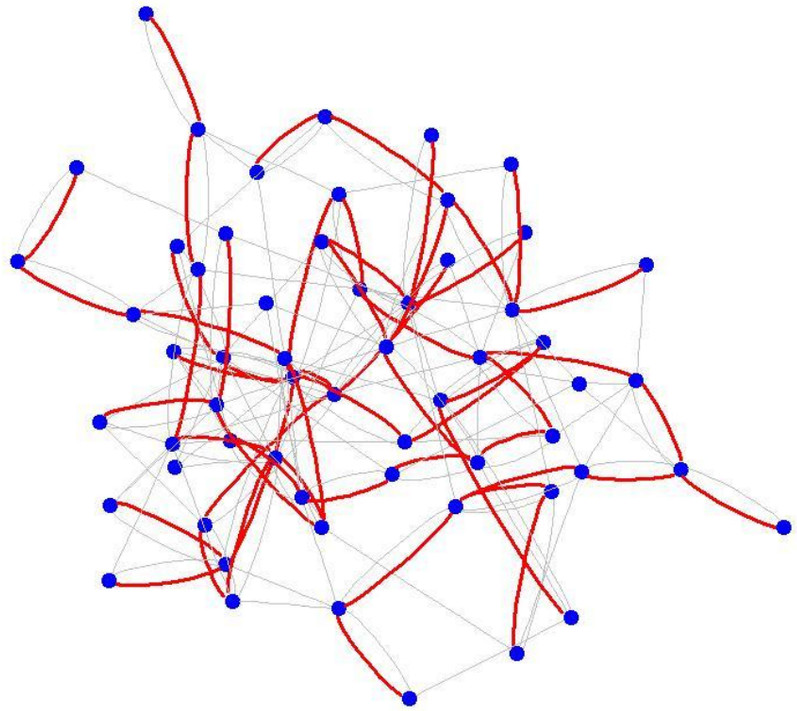


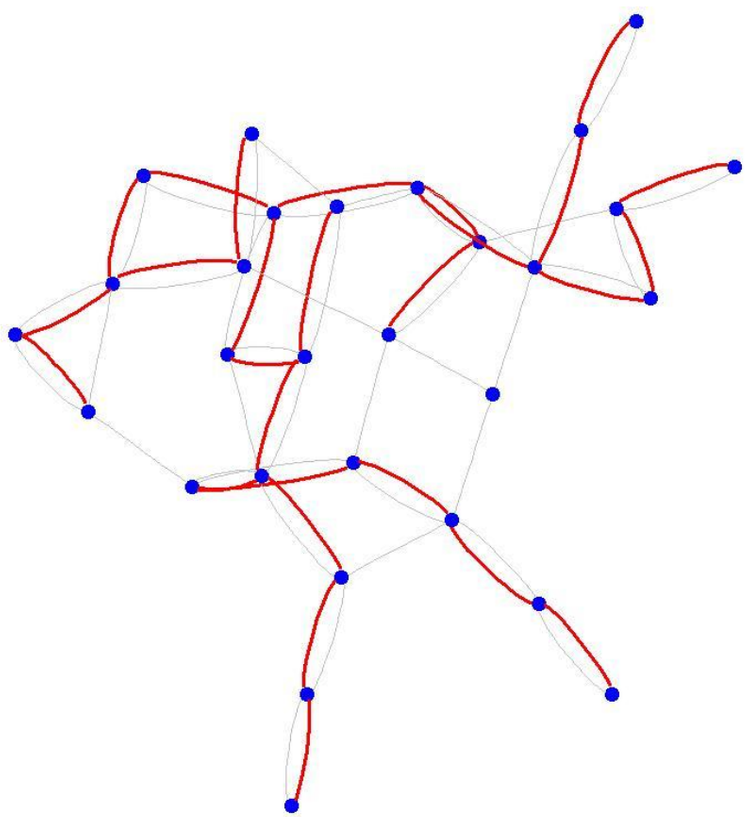
A different labeling of the virtual nodes, and the image
Note that in this case the image is not a tree.

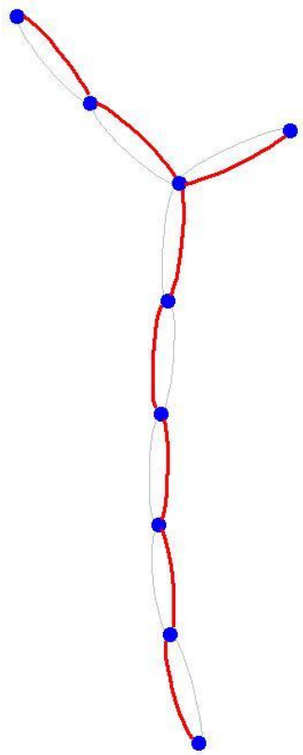
A series of unfortunate events











Forgiving Tree Summary

- Forgiving tree ensures degree increase is additive constant. Diameter increase is log of max degree.
- These parameters are essentially optimal.
- Forgiving tree is fully distributed, has O(1) latency and O(1) messages exchanged per round.

Other Self-Healing Results

Other Self-Healing Results

- Forgiving Graph
  - Handles insertions and deletions
  - Shortest path lengths increase by log n factor; degrees increase by factor of 3

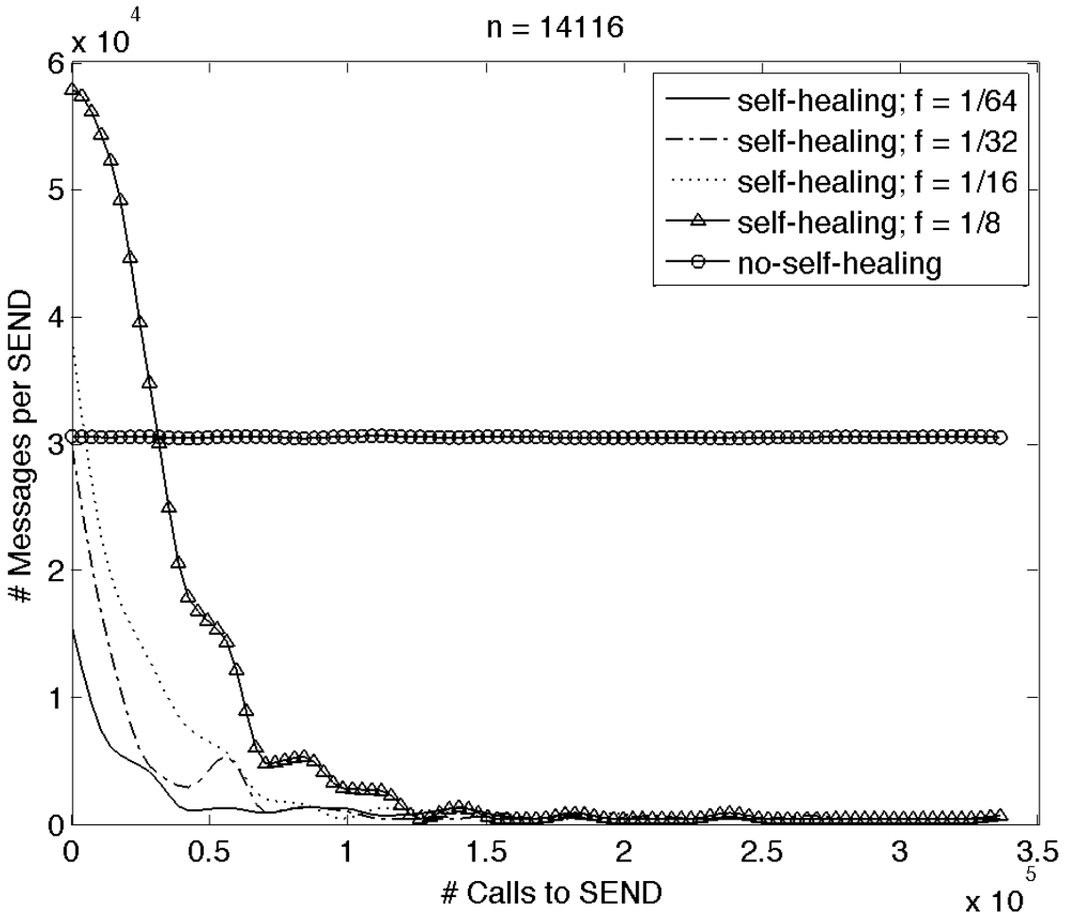
Other Self-Healing Results

- Forgiving Graph
  - Handles insertions and deletions
    - Shortest path lengths increase by log n factor; degrees increase by factor of 3
- BASH (*Byzantine* self-healing)
  - Goal: Reliable message delivery
  - Each *bad* node causes "small" number of message corruptions

BASH Theoretical Result

- If there are n nodes and t<n/8 are bad then we can enable sending of any message with
  - Asymptotically optimal latency and bandwidth cost
  - An expected number of corruptions that is $3t (\log^* n)^2$

BASH Empirical Results



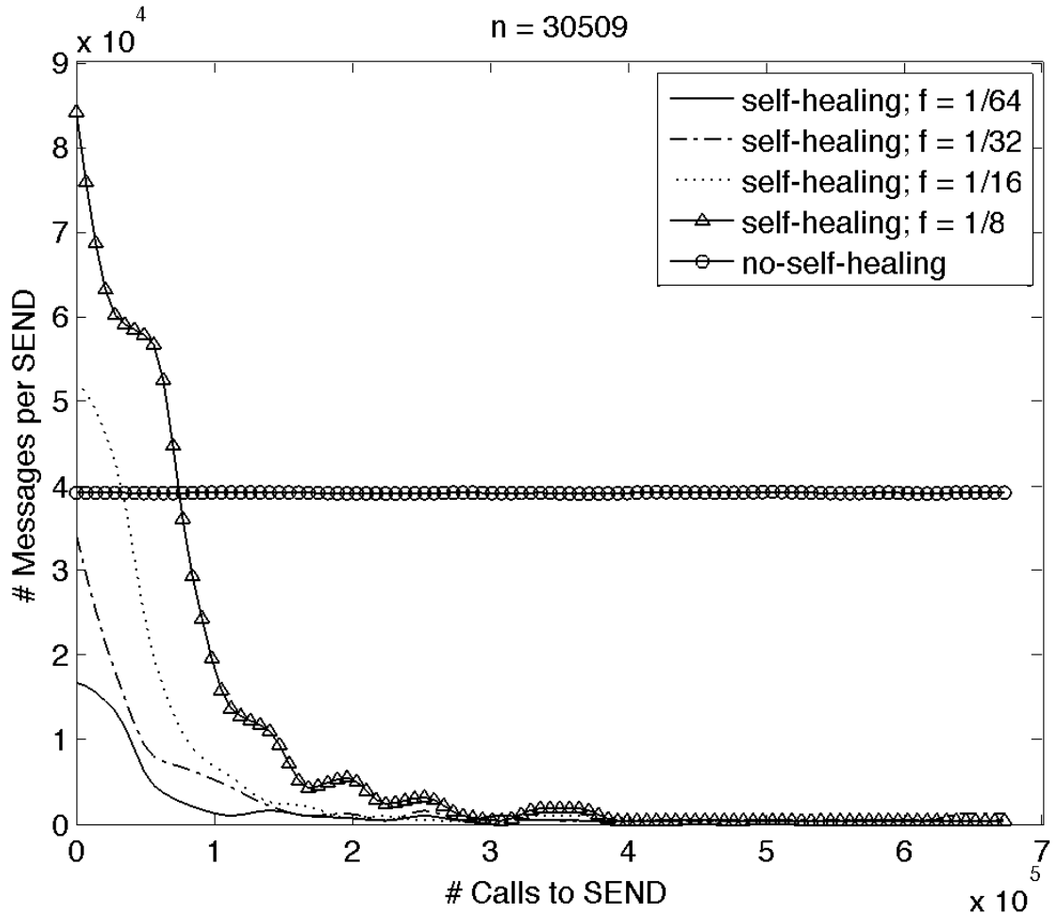


Fig. 2. # Messages per *SEND* versus # calls to *SEND*, for $n = 14{,}116$ and $n = 30{,}509$.

Conclusion

- Decades of work on "reliability boosting"
- Unfortunately, redundancy-based solutions are inherently inefficient
- Claim: we haven't been asking the right questions

Conclusion

- We should demand that resource costs of reliability boosting algorithms increase only when there are faults
- A self-healing approach allows us to achieve these kinds of results
- Q: What is the next step?

Towards a Research Agenda

"Make no little plans"

- **Succinct** problems are retained
- **Important** problems span disciplines
- **Hard** problems pull in smart people

An Agenda

- Step 1: Focus on specific tools and libraries

An Agenda

- Step 1: Focus on specific tools and libraries
- Step 2: Solve problems based on these tools

An Agenda

- Step 1: Focus on specific tools and libraries
- Step 2: Solve problems based on these tools
- Step 3: Proven reliability of these tools attracts research attention

An Agenda

- Step 1: Focus on specific tools and libraries
- Step 2: Solve problems based on these tools
- Step 3: Proven reliability of these tools attracts research attention
- Step 4: Build on algorithmic techniques to full-fledged reliable applications and ultimately a general approach

Relevant Papers

- Varsha Dani, Valerie King, Mahnush Mohavedi and Jared Saia. Quorums Quicken Queries: Efficient Asynchronous Secure Multiparty Computation. *Submitted to Conference on Distributed Computing and Networking (ICDCN),* 2013

- Jeff Knockel, George Saad and Jared Saia. Self-Healing of Byzantine Faults. *Symposium on Security, Safety and Stability (SSS*), 2013.

- Thomas P. Hayes, Jared Saia, and Amitabh Trehan. The forgiving graph: a distributed data structure for low stretch under adversarial attack. In *PODC '09: Proceedings of the 28th ACM symposium on Principles of distributed computing,* .

- Tom Hayes, Navin Rustagi, Jared Saia, and Amitabh Trehan. The forgiving tree: a self-healing distributed data structure. In *PODC '08: Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*.

- Jared Saia and Amitabh Trehan. Picking up the pieces: Self-healing in reconfigurable networks. In IEEE International Parallel & Distributed Processing Symposium. IPDPS 2008,

Questions?