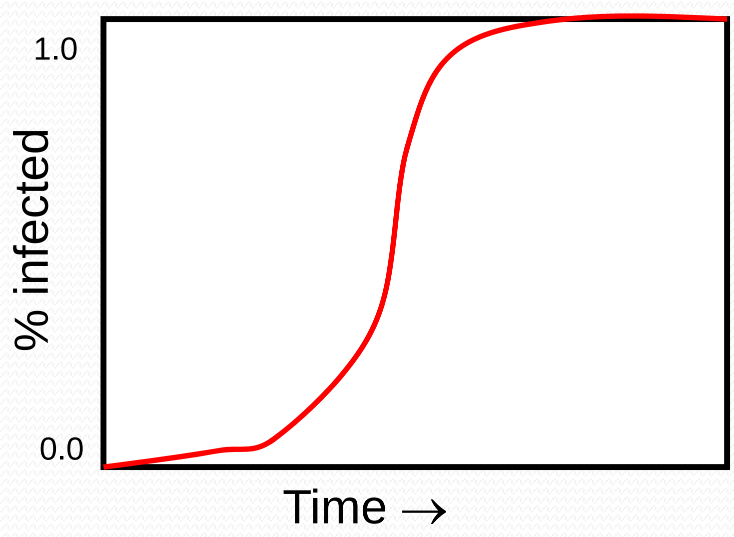
Using Gossip for Aggregation and Monitoring. Astrolabe.

Ken Birman

Cornell University. CS5410 Fall 2008.

Gossip 201

- Last time we saw that gossip spreads in log(system size) time
- But is this actually "fast"?



Gossip in distributed systems

- Log(N) can be a very big number!
  - With N=100,000, log(N) would be 12
  - So with one gossip round per five seconds, information needs *one minute* to spread in a large system!
- Some gossip protocols combine pure gossip with an accelerator
  - For example, *Bimodal Multicast* and *lpbcast* are protocols that use UDP multicast to disseminate data and then gossip to repair if any loss occurs
  - But the repair won't occur until the gossip protocol runs

A thought question

- What's the best way to
  - Count the number of nodes in a system?
  - Compute the average load, or find the most loaded nodes, or least loaded nodes?
- Options to consider
  - Pure gossip solution
  - Construct an overlay tree (via "flooding", like in our consistent snapshot algorithm), then count nodes in the tree, or pull the answer from the leaves to the root…

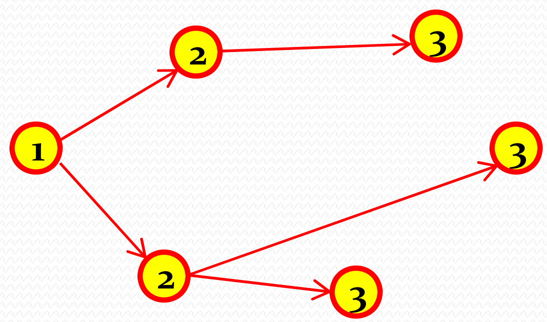
… and the answer is

- Gossip isn't very good for some of these tasks!
  - There are gossip solutions for counting nodes, but they give approximate answers and run slowly
  - Tricky to compute something like an average because of "re-counting" effect, (best algorithm: Kempe *et al)*
- On the other hand, gossip works well for finding the *c* most loaded or least loaded nodes (constant *c*)

- Gossip solutions will usually run in time O(log N) and generally give probabilistic solutions

Yet with flooding… easy!

- Recall how flooding works



Labels: distance of the node from the root

- Basically: we construct a tree by pushing data towards the leaves and linking a node to its parent when that node first learns of the flood
- Can do this with a fixed topology or in a gossip style by picking random next hops

This is a “spanning tree”

- Once we have a spanning tree
  - To count the nodes, just have leaves report 1 to their parents and inner nodes count the values from their children
  - To compute an average, have the leaves report their value and the parent compute the sum, then divide by the count of nodes
  - To find the least or most loaded node, inner nodes compute a min or max…
- Tree should have roughly log(N) depth, but once we build it, we can reuse it for a while

Not all logs are identical!

- When we say that a gossip protocol needs time log(N) to run, we mean log(N) rounds
  - And a gossip protocol usually sends one message every five seconds or so, hence with 100,000 nodes, 60 secs
- But our spanning tree protocol is constructed using a flooding algorithm that runs in a hurry
  - Log(N) depth, but each "hop" takes perhaps a millisecond.
  - So with 100,000 nodes we have our tree in 12 ms and answers in 24ms!

Insight?

- Gossip has time complexity O(log N) but the "constant" can be rather big (5000 times larger in our example)
- Spanning tree had same time complexity but a tiny constant in front

- But network load for spanning tree was much higher
  - In the last step, we may have reached roughly half the nodes in the system
  - So 50,000 messages were sent all at the same time!

Gossip vs "Urgent"?

- With gossip, we have a slow but steady story
  - We know the speed and the cost, and both are low
  - A constant, low-key, background cost
  - And gossip is also very robust
- Urgent protocols (like our flooding protocol, or 2PC, or reliable virtually synchronous multicast)
  - Are way faster
  - But produce load spikes
  - And may be fragile, prone to broadcast storms, etc

Introducing hierarchy

- One issue with gossip is that the messages fill up
  - With constant sized messages…
  - … and constant rate of communication
  - … we'll inevitably reach the limit!

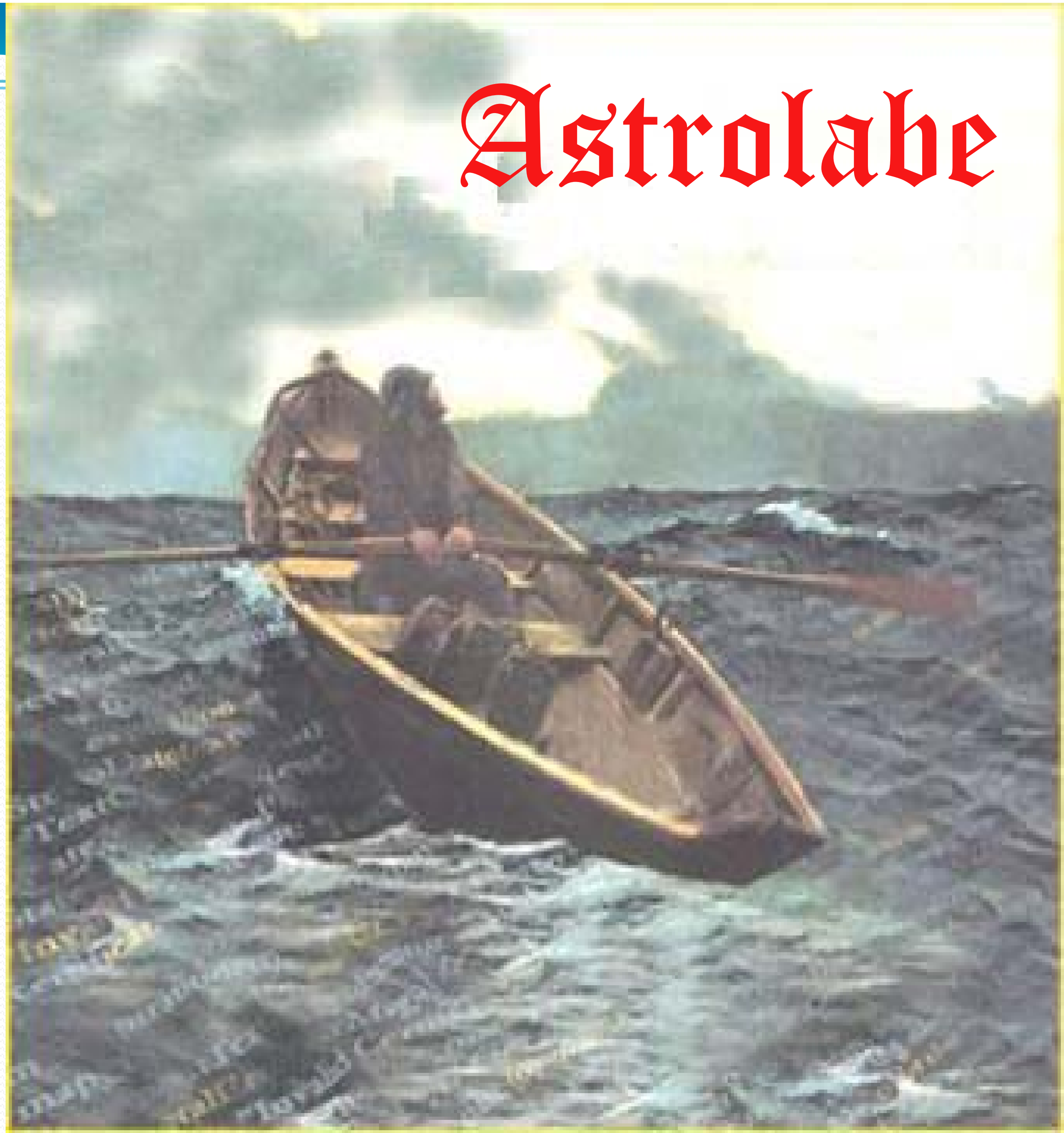
- Can we inroduce hierarchy into gossip systems?

Astrolabe

- Intended as help for applications adrift in a sea of information
- Structure emerges from a randomized gossip protocol
- This approach is robust and scalable even under stress that cripples traditional systems

Developed at RNS, Cornell

- By Robbert van Renesse, with many others helping…
- Today used extensively within Amazon.com



Astrolabe is a flexible monitoring overlay

swift.cs.cornell.edu

Name	Time	Load	Weblogic?	SMTP?	Word Version
swift	2271	1.8	0	1	6.2
falcon	1971	1.5	1	0	4.1
cardinal	2004	4.5	1	0	6.0

Periodically, pull data from monitored systems

cardinal.cs.cornell.edu

Name	Time	Load	Weblogic?	SMTP?	Word Version
swift	2003	.67	0	1	6.2
falcon	1976	2.7	1	0	4.1
cardinal	2231	1.7	1	1	6.0

Astrolabe in a single domain

- Each node owns a single tuple, like the management information base (MIB)
- Nodes discover one-another through a simple broadcast scheme (“anyone out there?”) and gossip about membership
  - Nodes also keep replicas of one-another’s rows
  - Periodically (uniformly at random) merge your state with some else…

State Merge: Core of Astrolabe epidemic

swift.cs.cornell.edu

Name	Time	Load	Weblogic?	SMTP?	Word Version
swift	2011	2.0	0	1	6.2
falcon	1971	1.5	1	0	4.1
cardinal	2004	4.5	1	0	6.0

cardinal.cs.cornell.edu

Name	Time	Load	Weblogic ?	SMTP?	Word Version
swift	2003	.67	0	1	6.2
falcon	1976	2.7	1	0	4.1
cardinal	2201	3.5	1	1	6.0

State Merge: Core of Astrolabe epidemic

swift.cs.cornell.edu

Name	Time	Load	Weblogic?	SMTP?	Word Version
swift	2011	2.0	0	1	6.2
falcon	1971	1.5	1	0	4.1
cardinal	2004	4.5	1	0	6.0

swift	2011	2.0

cardinal	2201	3.5

cardinal.cs.cornell.edu

Name	Time	Load	Weblogic ?	SMTP?	Word Version
swift	2003	.67	0	1	6.2
falcon	1976	2.7	1	0	4.1
cardinal	2201	3.5	1	1	6.0

State Merge: Core of Astrolabe epidemic

swift.cs.cornell.edu

Name	Time	Load	Weblogic?	SMTP?	Word Version
swift	2011	2.0	0	1	6.2
falcon	1971	1.5	1	0	4.1
cardinal	2201	3.5	1	0	6.0

cardinal.cs.cornell.edu

Name	Time	Load	Weblogic ?	SMTP?	Word Version
swift	2011	2.0	0	1	6.2
falcon	1976	2.7	1	0	4.1
cardinal	2201	3.5	1	1	6.0

Observations

- Merge protocol has constant cost
  - One message sent, received (on avg) per unit time.
  - The data changes slowly, so no need to run it quickly – we usually run it every five seconds or so
  - Information spreads in O(log N) time
- But this assumes bounded region size
  - In Astrolabe, we limit them to 50-100 rows

Big systems…

- A big system could have *many* regions
  - Looks like a pile of spreadsheets
  - A node only replicates data from its neighbors within its own region

Scaling up… and up…

- With a stack of domains, we don't want every system to "see" every domain
  - Cost would be huge
- So instead, we'll see a summary

cardinal.cs.cornell.edu

Name	Time	Load	Weblogic?	SMTP?	Word Version
swift	2011	2.0	0	1	6.2
falcon	1976	2.7	1	0	4.1
cardinal	2201	3.5	1	1	6.0

Astrolabe builds a hierarchy using a P2P protocol that “assembles the puzzle” without any servers

Dynamically changing query output is visible system-wide

Name	Avg Load	WL contact	SMTP contact
SF	2.2	123.45.61.3	123.45.61.17
NJ	1.6	127.16.77.6	127.16.77.11
Paris	2.7	14.66.71.8	14.66.71.12

SQL query “summarizes” data

Name	Load	Weblogic?	SMTP?	Word Version	...
swift	1.7	0	1	6.2	
falcon	2.1	1	0	4.1	
cardinal	3.9	1	0	6.0	

San Francisco

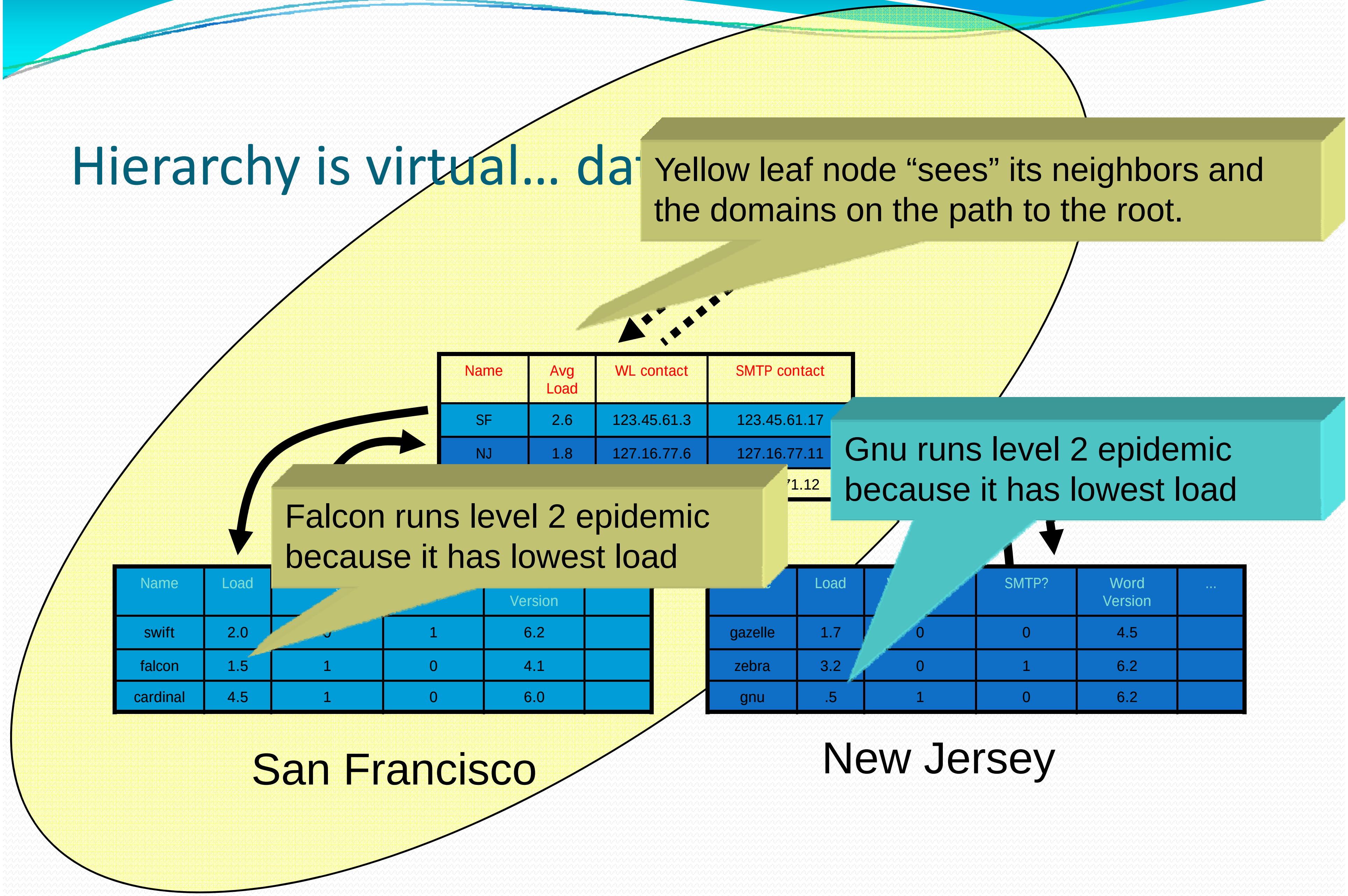
Name	Load	Weblogic?	SMTP?	Word Version	...
gazelle	4.1	0	0	4.5	
zebra	0.9	0	1	6.2	
gnu	2.2	1	0	6.2	

New Jersey

Large scale: “fake” regions

- These are
  - Computed by queries that summarize a whole region as a single row
  - Gossiped in a read-only manner within a leaf region
- But who runs the gossip?
  - Each region elects “k” members to run gossip at the next level up.
  - Can play with selection criteria and “k”

Hierarchy is virtual… dat



Hierarchy is virtual… dat

Green node sees different leaf domain but has a consistent view of the inner domain

Name	Avg Load	WL contact	SMTP contact
SF	2.6	123.45.61.3	123.45.61.17
NJ	1.8	127.16.77.6	127.16.77.11
Paris	3.1	14.66.71.8	14.66.71.12

Name	Load	Weblogic?	SMTP?	Word Version	…
swift	2.0	0	1	6.2	
falcon	1.5	1	0	4.1	
cardinal	4.5	1	0	6.0	

San Francisco

Name	Load	Weblogic?	SMTP?	Word Version	…
gazelle	1.7	0	0	4.5	
zebra	3.2	0	1	6.2	
gnu	.5	1	0	6.2	

New Jersey

Worst case load?

- A small number of nodes end up participating in $O(\log_{fanout}N)$ epidemics
  - Here the fanout is something like 50
  - In each epidemic, a message is sent and received roughly every 5 seconds
- We limit message size so even during periods of turbulence, no message can become huge.

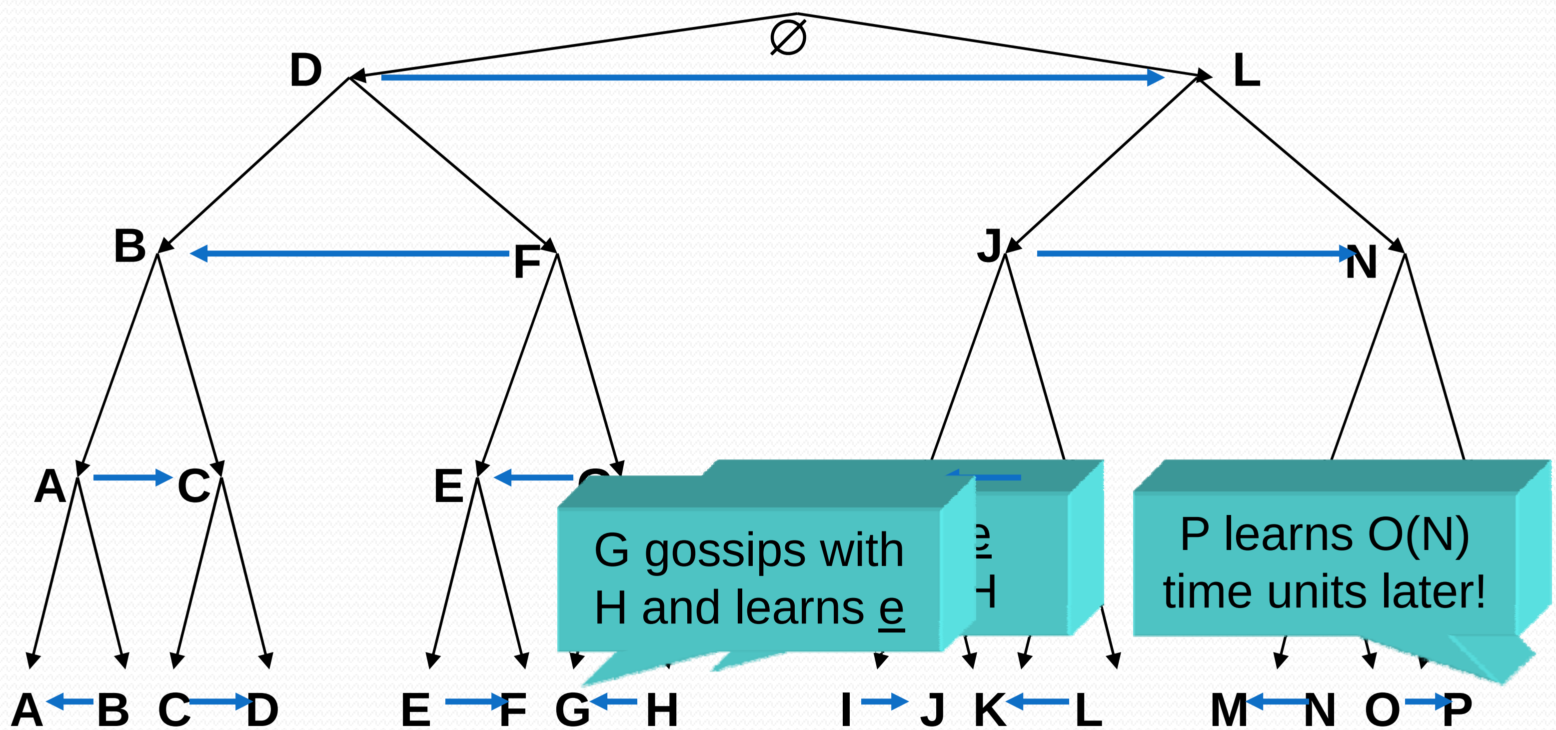
Who uses Astrolabe?

- Amazon uses Astrolabe throughout their big data centers!
  - For them, Astrolabe helps them track overall state of their system to diagnose performance issues
  - They can also use it to automate reaction to temporary overloads

Example of overload handling

- Some service S is getting slow…
  - Astrolabe triggers a "system wide warning"
- Everyone sees the picture
  - "Oops, S is getting overloaded and slow!"
  - So everyone tries to reduce their frequency of requests against service S
- What about overload in Astrolabe *itself*?
  - Could everyone do a fair share of inner aggregation?

A fair (but dreadful) aggregation tree



What went wrong?

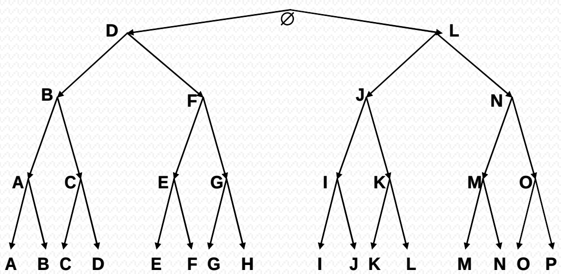
- In this horrendous tree, each node has equal "work to do" but the information-space diameter is larger!
- Astrolabe benefits from "instant" knowledge because the epidemic at each level is run <u>by someone elected from the level below</u>

Insight: Two kinds of shape

- We’ve focused on the aggregation tree
- But in fact should also think about the information flow tree

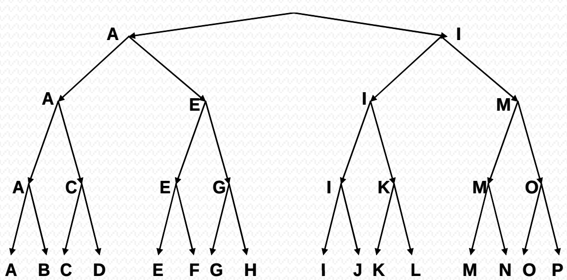
Information space perspective

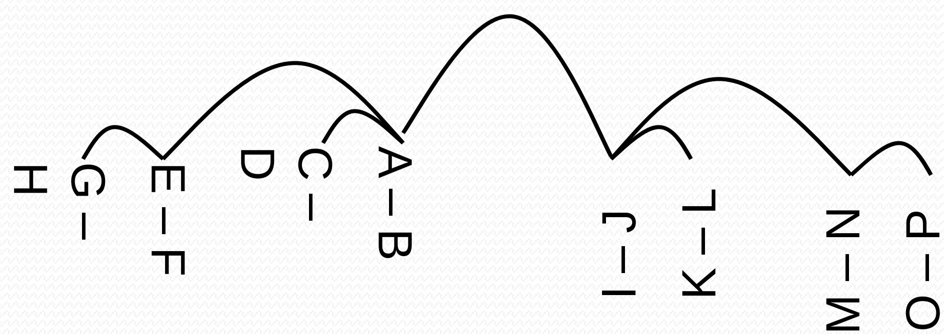
- Bad aggregation graph: diameter O(n)



H – G – E – F – B – A – C – D – L – K – I – J – N – M – O – P

- Astrolabe version: diameter O(log(n))





Summary

- We looked at ways of using Gossip for aggregation
  - Pure gossip isn't ideal for this… and competes poorly with flooding and other urgent protocols
  - But Astrolabe introduces hierarchy and is an interesting option that gets used in at least one real cloud platform
- Power: make a system more robust, self-adaptive, with a technology that won't make things worse
- But performance can still be sluggish