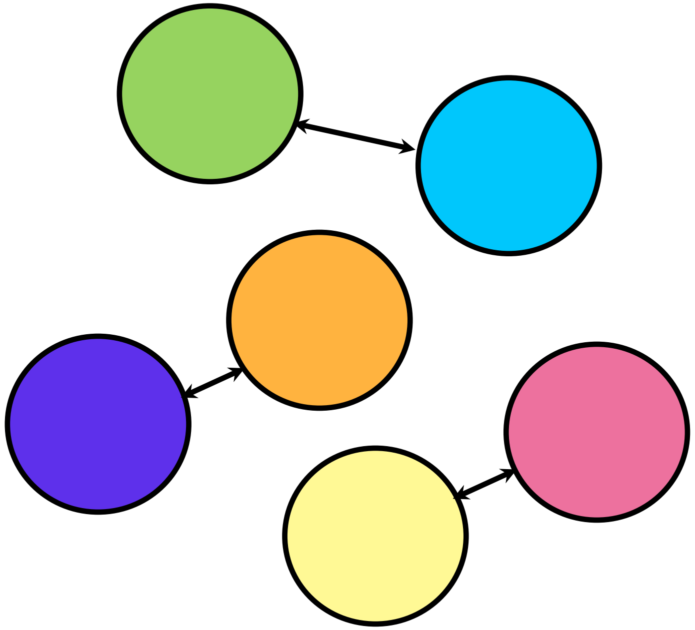
Gossip Techniques

Makoto Bentz
mb434@cs.cornell.edu
Oct. 27, 2010

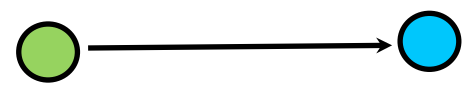
What is Gossip?

- Gossip is the periodic **pairwise** exchange of bounded size messages between **random nodes** in the system in which nodes states may affect each other
- Has O(log n) completion time
- **Benefits:** simplicity, limited resource usage, robustness to failures, and tunable system behavior

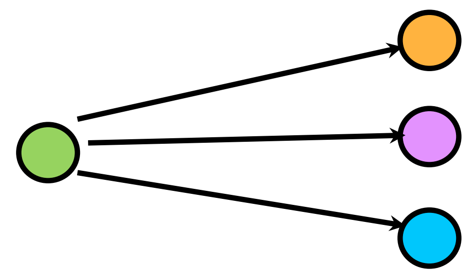


How is Gossip Different?

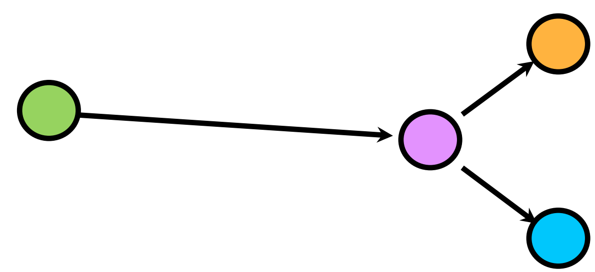
- Unicast: One person tells one person



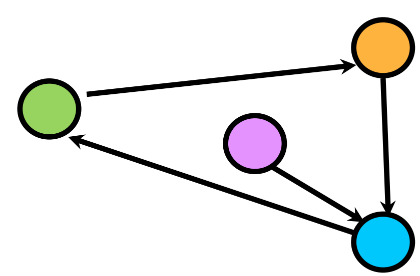
- Broadcast: One node tells everyone



- Multicast: One person tells all via intermediary nodes



- Gossip: Everyone tells someone else what they know

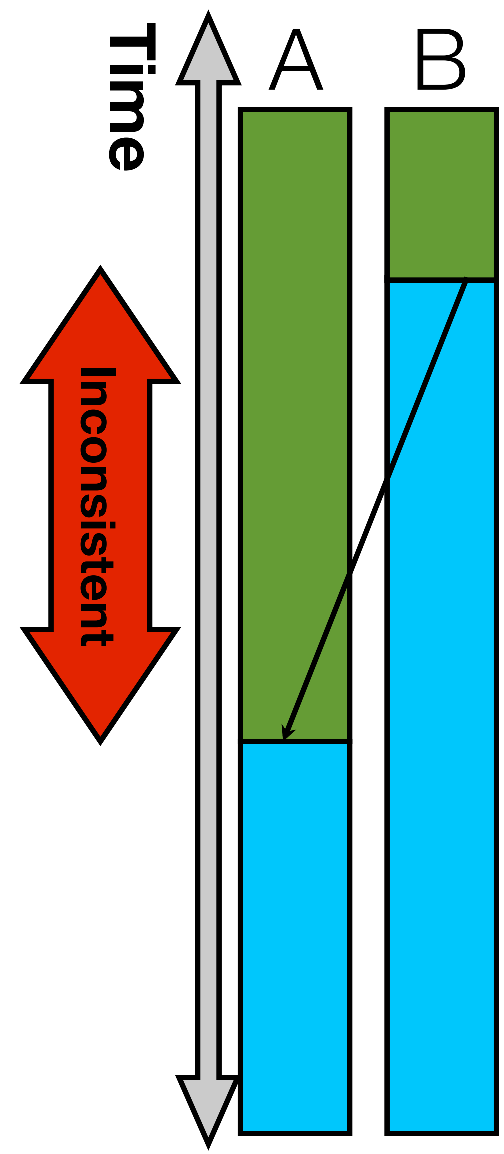


Eventual Consistency

- **Strong Consistency:** After the update completes, any subsequent access will return the updated value.

- **Weak consistency:** System doesn't guarantee subsequent accesses will return the updated value. A number of conditions need to be met before the value will be returned.

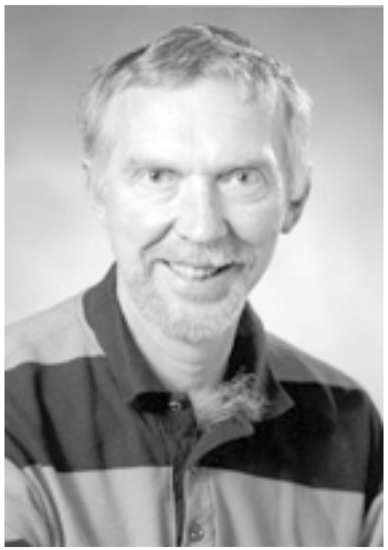
- **Eventual consistency:** Subset of weak consistency; the system guarantees that **if no new updates** are made to the object, **eventually** all accesses will return the **last updated value**.



Gossip Techniques: Papers

- Epidemic algorithms for replicated database maintenance, Demers et al. 6th PODC, 1987.
- Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining, Van Renesse et al. ACM TOCS 2003.
- Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead, Indranil Gupta, Ken Birman, Prakash Linga, Al Demers and Robbert van Renesse. 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03); February 20-21, 2003. Claremont Hotel, Berkeley, CA, USA.

Epidemic Algorithms: Authors



Alan Demers is a researcher at Cornell University

Dan Greene is at Xerox parc His research now focuses on vehicle networks

Carl Hauser is a Associate Professor at Washington State University

Epidemic Algorithms: Authors

Scott Shenker is an associate professor at U.C. Berkeley

Wes Irish now runs Coyote Hill Consulting LLC

Doug Terry is the Primary Researcher at Microsoft Research Silicon Valley

Epidemic Algorithms: Authors

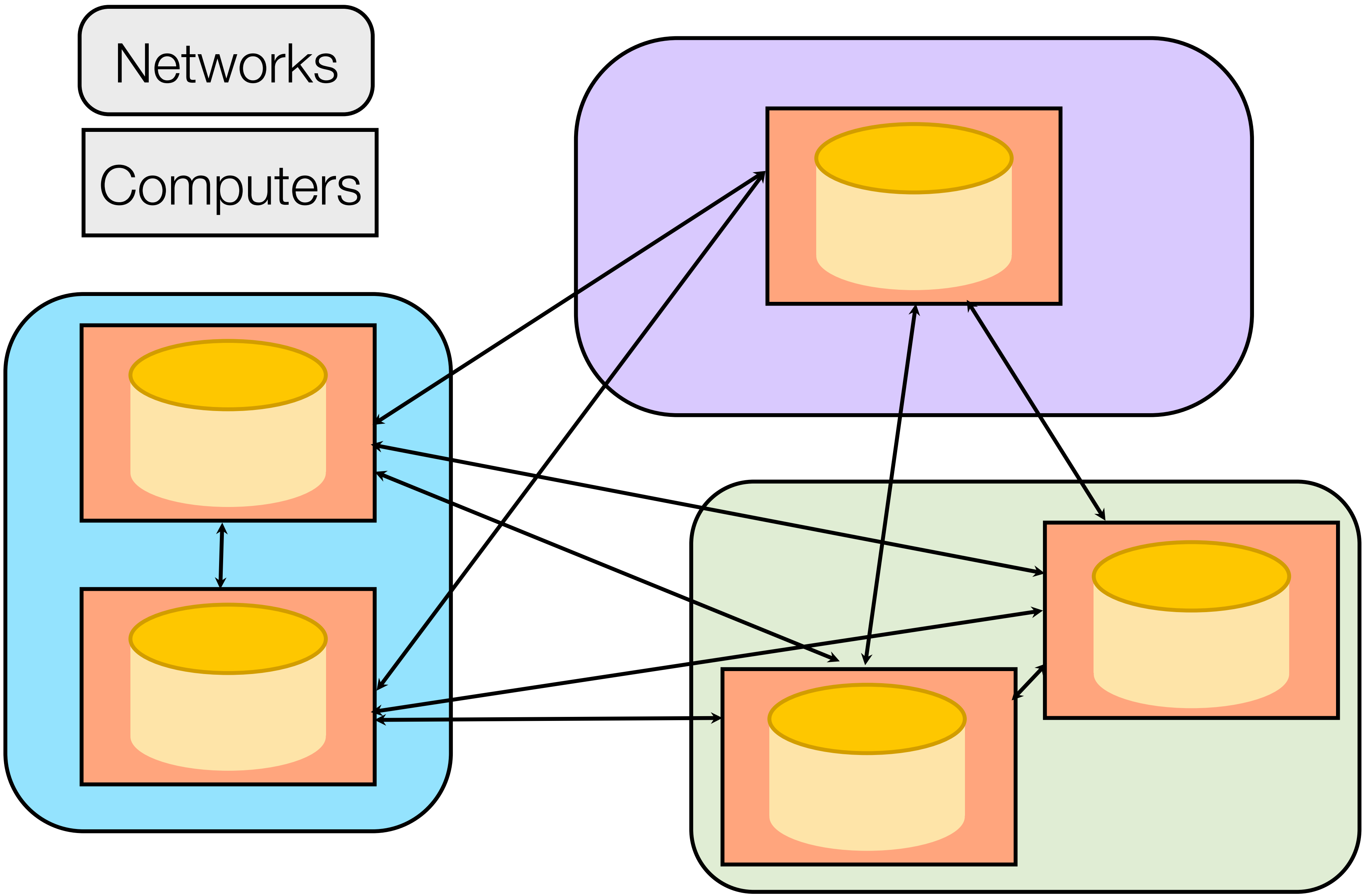
- John Larson worked on Cedar DBMS and LDAP and at Sprint Advanced Technology Labs
- Howard Sturgis discovers 2-phase transaction commit and worked on Cedar DBMS and RPCs
- Dan Swinehart worked on Bayou

parc A Xerox Company

parc Palo Alto Research Center

XEROX. PARC

Epidemic Algorithms: Status Quo



Epidemic Algorithms: Problem Statement

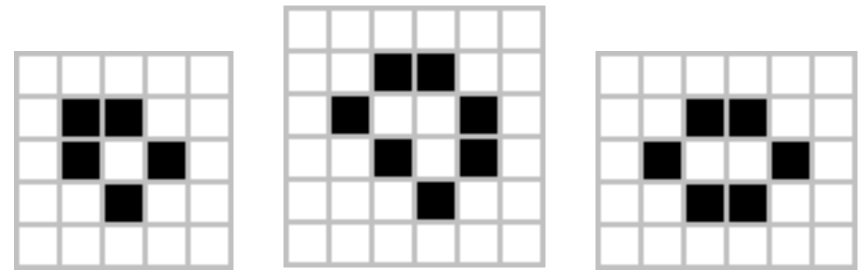
- Clearinghouse Servers on Xerox Corporate Internet
- Several hundred Ethernets connected by gateways and phone lines
    - Several thousand computers
- Three-level hierarchy with top two levels being domains
- ***Need to keep databases on computers between domains (eventually) consistent***

Epidemic Algorithms: First Attempt

- Originally using what was a rudimentary form of Direct Mail (Multicast) and Anti-Entropy (Gossip)
- Inefficient/Redundant
  - Anti-Entropy was being redundantly followed by Direct Mail, saturating the network (300 clients -> 90,000 mail messages)
- Not scalable
  - Network capacity saturated -> failure

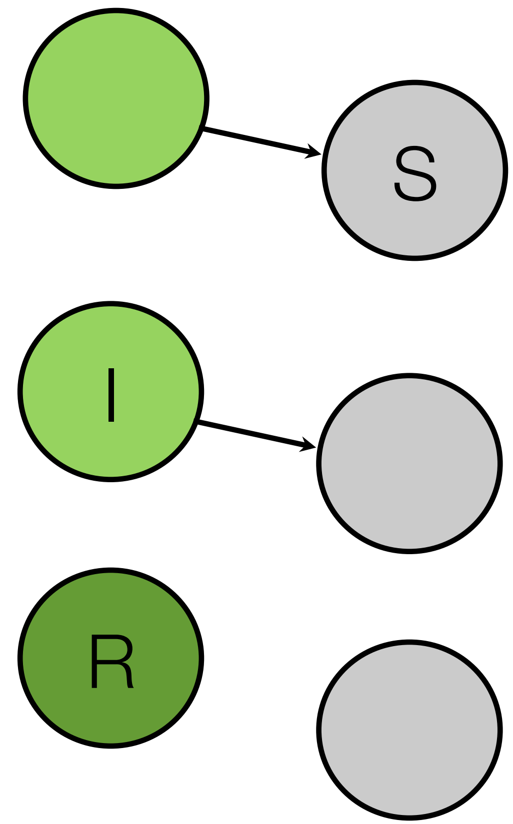
Epidemic Techniques: What are they?

- "Epidemic algorithms follow the paradigm of nature by applying **simple rules to spread information** by just having a **local view** of the environment" Hollerung, Bleckmann
- Conway's **Game of Life** is an epidemic algorithm
- Medical epidemics spread between individuals by contagion



Epidemic Algorithms: Types of Spreading

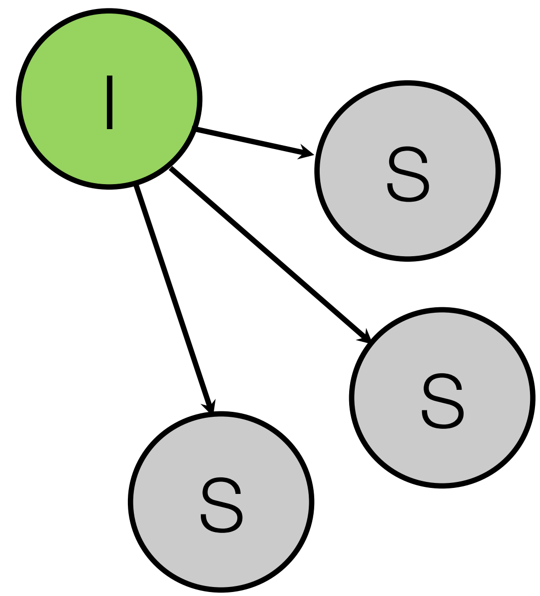
Unit Type	Description
Susceptible	Does not know info, but can get info
Infective	Knows the info and spreads it by the rule
Removed	Knows the info but does not spread it



Can be combinations of the above

Epidemic Algorithms: Direct Mail

- **Direct Mail:** Send to everyone
- Send
    - FOR EACH s' in S
      DO PostMail[to: s', msg : ("Update", s.ValueOf)]
      ENDLOOP
- Receive
    - IF s.Value0f.t < t THEN
      s.ValueOf - (7!,t)
- **Susceptaible to failure**, O(n) bottleneck, Original could have incomplete information
- Xerox system did not use broadcast mailing



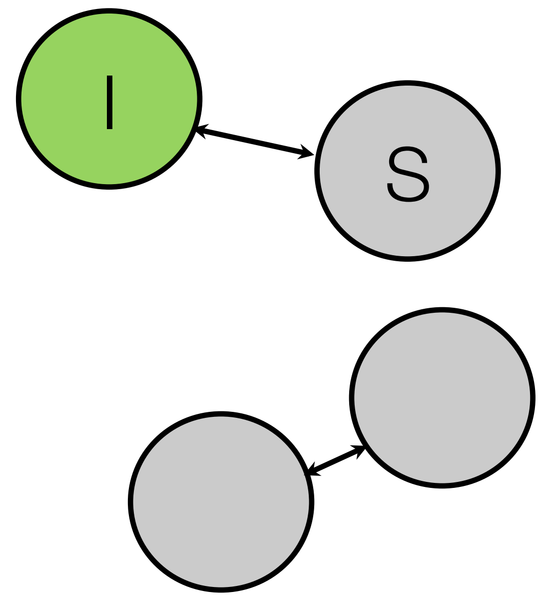
Epidemic Algorithms: Anti-Entropy

- Anti-Entropy: Everyone picks a site at random, and resolves differences between it and its recipient

- FOR SOME s' in S
    DO ResolveDifference[s, s']
  ENDLOOP

- Resolving can be done by push, pull, push-pull

- Slower than direct mail, and **expensive** to compare databases



Epidemic Algorithms: Anti-Entropy: Resolving

- Push

```
ResolveDifference : PROC[.s, s’] = {
    IF s.Value0f.t > s’.ValueOf.t THEN
        s’.ValueOf <- s.ValueOf }
```

- Pull

```
ResolveDifference : PROCis, s’] = {
    IF s.Value0f.t < s’.ValueOf.t THEN
        s.ValueOf + s’.ValueOf }
```

- Push-Pull

```
ResolveDifference  : PR.OC’[s. s’] = {
    SELECT TRUE FROM
        s.Value0f.l > s’.ValueOf.t => s’.ValueOf - s.ValueOf;
        s.ValueOf.t < s’.ValueOf.t => s.ValueOf - s’.ValueOf;
ENDCASE => NULL;
```

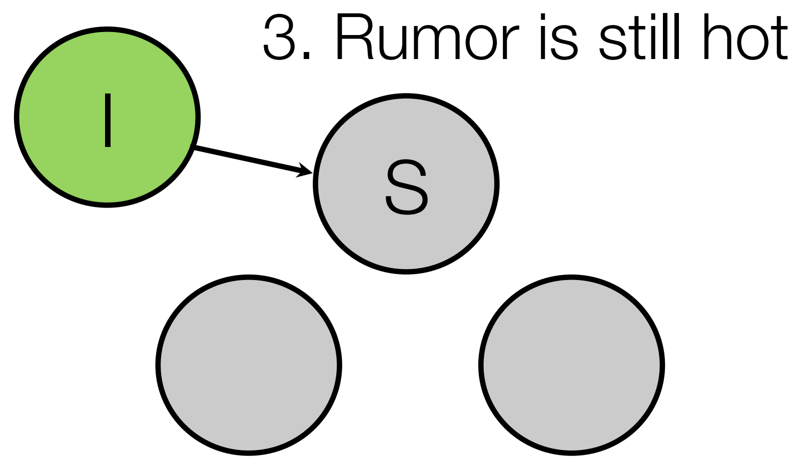
Epidemic Algorithms: Rumor Spreading

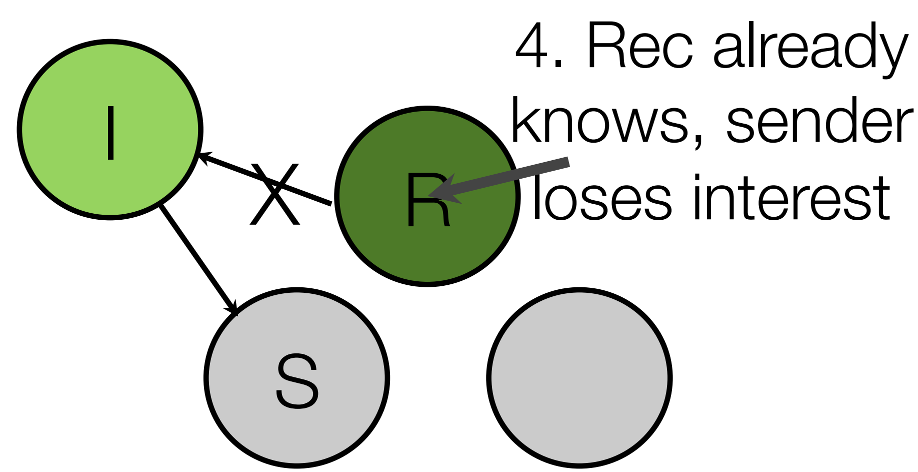
1. There are initially no active people, each person with a rumor is active

2. Someone gets the rumor

3. Each active person then randomly phones other persons to tell them the rumor

4. If the recipient already knows the rumor, then the sender loses interest and becomes inactive

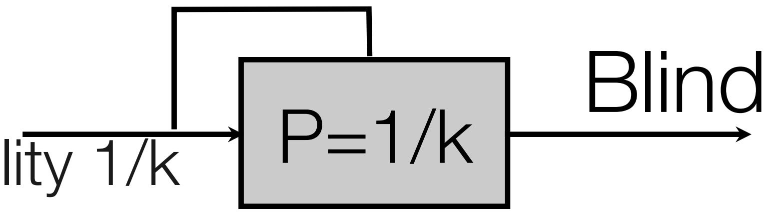


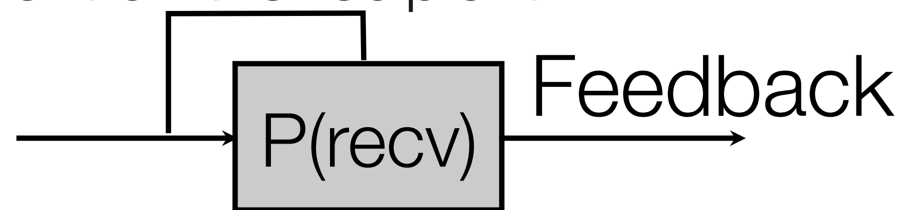


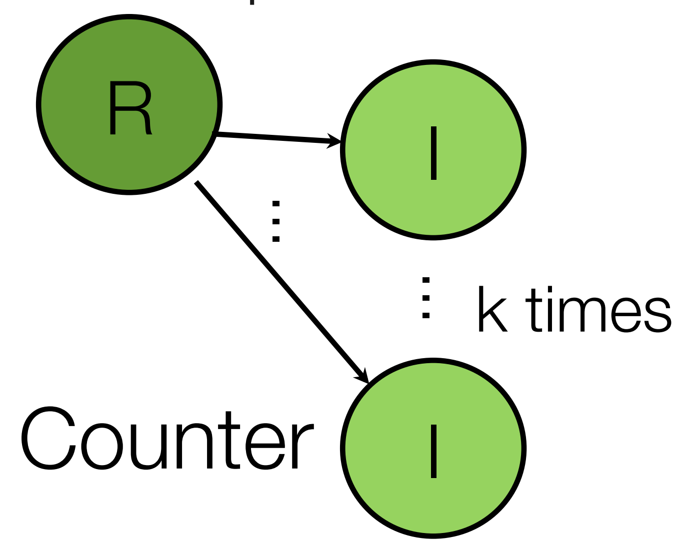
Epidemic Algorithms: Rumor Spreading

- **Blind vs. Feedback**
  Blind senders lose interest with probability 1/k
  Feedback senders lose interest dependent on the recipient

- **Counter vs. Coin**
  Counter loses interest after k unnecessary contacts
  Coin loses interest after a 1/k probability coin toss upon unnecessary contacts







Epidemic Algorithms: Theory

- s + i + r = 1

$$\frac{ds}{dt} = -si$$
$$\frac{di}{dt} = +si - \frac{1}{k}(1 - s)i$$

$$\frac{di}{ds} = -\frac{k+1}{k} + \frac{1}{ks}$$
$$i(s) = -\frac{k+1}{k}s + \frac{1}{k}\log s + c$$

Epidemic Algorithms: Backing up

- A complex epidemic may not converge
- Back up by adding anti-entropy as well as rumor mongering
  - Direct mail is $O(n^2)$ per cycle at worst case
  - Rumor mongering is always $O(n)$ or less
- Death certificates carry timestamps marking deletion
  - Dormant death certificates do not scale well
    (deletion time ~ $O(\log n)$
  - Activation timestamp added to death certificate to prevent rollback of data changed after a death certificate first went out

Epidemic Algorithms: Testing

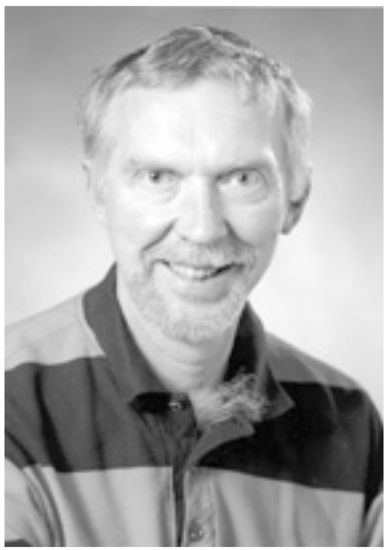
Table 5. Simulation results for anti-entropy, connection limit 1.

Spatial Distribution	t_{last}	t_{ave}	Compare Traffic Average	Compare Traffic Bushey	Update Traffic Average	Update Traffic Bushey
uniform	11.00	6.97	3.71	47.54	5.83	75.17
$a = 1.2$	16.89	9.92	1.14	6.39	2.69	18.03
$a = 1.4$	17.34	10.15	1.08	4.68	2.55	13.68
$a = 1.6$	19.06	11.06	0.94	2.90	2.32	10.20
$a = 1.8$	21.46	12.37	0.82	1.68	2.12	7.03
$a = 2.0$	24.64	14.14	0.72	0.94	1.94	4.85

Epidemic Algorithms: Discussion

- I felt like this paper started to rush near the end
  - Great explanation of the theory, weak explanation of the testing and implementation
- This paper goes on to be the foundation of Gossip
  - Cited at least 249+18(PDOC+SIGOPS) times

Bayou: Authors



Alan Demers is a researcher at Cornell University

Carl Hauser is a Associate Professor at Washington State University

Doug Terry is the Primary Researcher at Microsoft Research Silicon Valley

Bayou: Authors

- **Marvin Theimer** is the Senior Principal Engineer at Amazon Web Services

Michael Spreitzer works in Services Management Middleware at Thomas J. Watson Research Center, Hawthorne, NY USA

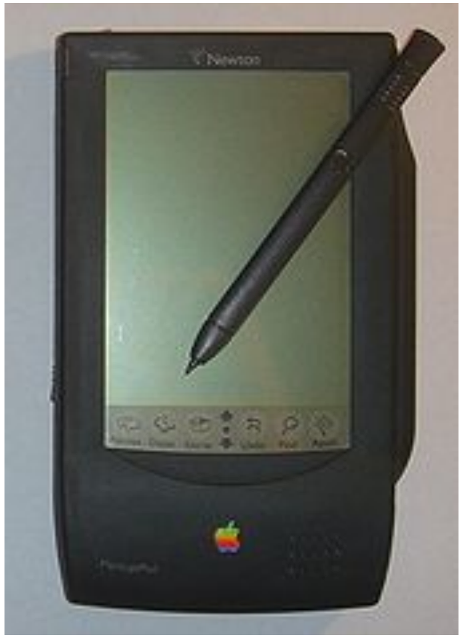
Bayou: The Name

- **TOP 10 Reasons for the name "Bayou":**

- 10. Why not?
- 9. It's better than "UbiData".
- 8. It's a lot better than "DocuData".
- 7. It's not an acronym.
- 6. It's not named after a soft drink (e.g. Tab, Sprite, Coda Cola, ...).
- 5. We're working on replication that's "fluid" like a bayou.
- 4. We're exploring a small part of the "UbiComp Swamp".
- 3. It's the name of a famous tapestry (spelled "Bayeux" however).
- 2. Our system will allow you to access data even when you're "bayou self".
- 1. It's pronounced "Bi-U", which makes it "Ubi" pronounced backwards.

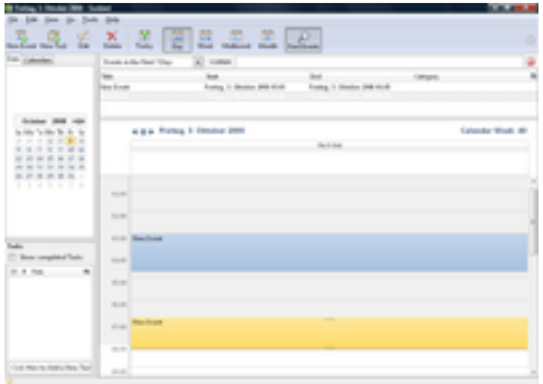
Bayou: The Problem

- Wireless and mobile devices do not permit constant connectivity
- Weak connectivity
- Collaborative applications such as calendars



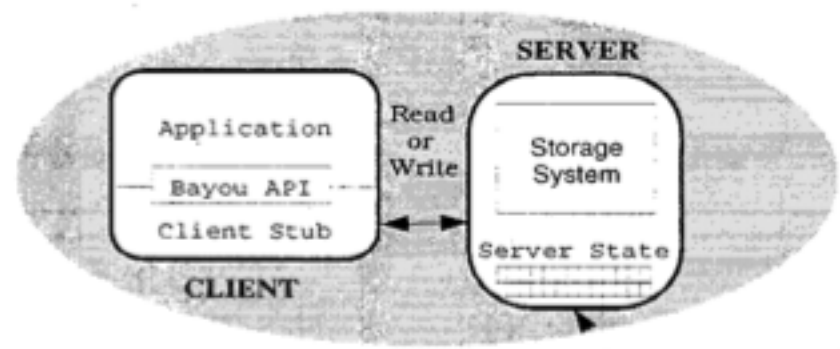
MessagePad 100 (1993)

Powerbook 500 (1994)



Bayou: The Design

- Data collections are replicated at Servers
- Clients run applications that access the servers via an API
  - Read and Write
- Each server stores an ordered log of Writes and the resulting data
- Performs Writes and Conflict Detection
- Anti-Entropy to propagate updates



Bayou: Design: Conflict Detection

- Dependency Checks
  - Application Specific Conflict Checks
  - Write is accompanied with query and expected result required to write (ex. to reserve 2, the set of reserved should not include 2)
- Merge Procedure
  - Conflict Detected -> Merge Procedure
  - High-level, interpreted language code to pick a result in merge
  - Does not lock conflicted data

Bayou: Design: Eventual Consistency

- Bayou replicas all follow Eventual Consistency
- This is ensured by the following two rules
  - Writes are performed in order
  - Conflict Detection and Merge procedure are deterministic, resulting in the same resolve at the server
- Writes are stable after they have been executed for the last time
- Commits will ensure stability

Bayou: Implementation

- Tuple Store, in-memory relational database
- Access Control by public-key cryptography, allows for grants, delegation and revocation

Bayou: Implementation

- Written in ILU (an RPC) and Tcl
- Per-database library mechanism for each write to prevent replicated code

Table 1: Size of Bayou Storage System for the Bibliographic Database with 1550 Entries
(sizes in Kilobytes)

Number of Tentative Writes	0 (none)	50	100	500	1550 (all)
Write Log	9	129	259	1302	4028
Tuple Store Ckpt	396	384	371	269	1
Total	**405**	**513**	**630**	**1571**	**4029**
Factor to 368K bibtex source	1.1	1.39	1.71	4.27	10.95

Bayou: Implementation

Table 2: Performance of the Bayou Storage System for Operations on Tentative Writes in the Write Log
(times in milliseconds with standard deviations in parentheses)

Tentative Writes	0	50		100		500		1550	
	Server running on a Sun SPARC/20 with Sunos								
Undo all	0	31	(6)	70	(20)	330	(155)	866	(195)
(avg. per Write)		.62		.7		.66		.56	
Redo all	0	237	(85)	611	(302)	2796	(830)	7838	(1094)
(avg. per Write)		4.74		6.11		5.59		5.05	
	Server running on a Gateway Liberty Laptop with Linux								
Undo all	0	47	(3)	104	(7)	482	(15)	1288	(62)
(avg. per Write)		.94		1.04		.96		.83	
Redo all	0	302	(91)	705	(134)	3504	(264)	9920	(294)
(avg. per Write)		6.04		7.05		7.01		6.4	

Table 3: Performance of the Bayou Client Operations
(times in milliseconds with standard deviations in parentheses)

Server Client	Sun SPARC/20 same as server		Gateway Liberty same as server		Sun SPARC/20 Gateway Liberty	
Read: 1 tuple	27	(19)	38	(5)	23	(4)
100 tuples	206	(20)	358	(28)	244	(10)
Write: no conflict	159	(32)	212	(29)	177	(22)
with conflict	207	(37)	372	(17)	223	(40)

Bayou: Discussion

- Was a well-written paper
- Industry paper, testing not well explained

Resources

- http://www2.cs.uni-paderborn.de/cs/ag-madh/WWW/Teaching/2004SS/AlgInternet/Submissions/09-Epidemic-Algorithms.pdf