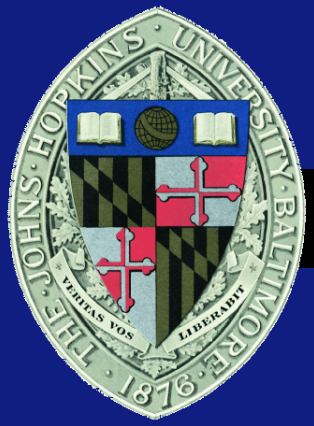
NoSQL

EN 600.420

Instructor: Randal Burns

3 May 2017



Department of Computer Science, *Johns Hopkins University*

Most Cited Computer Science Articles

This list is generated from documents in the CiteSeer^{X} database as of January 17, 2013. This list is automatically generated and may contain err
and citation counts may differ from those currently in the CiteSeer^{X} database, since the database is continuously updated.

All Years | 1990 | 1991 | 1992 | 1993 | 1994 | 1995 | 1996 | 1997 | 1998 | 1999 | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2

1. A P Dempster, N M Laird, D B Rubin.
   Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society, 1977
   6082
2. C A R Hoare.
   Communicating sequential processes. ISSN 0001-0782. URL http://doi.acm.org/10.1145/359576, 1978
   3098
3. L Rabiner.
   A tutorial on hidden Markov models and selected applications in speech recognition. Proc. IEEE, 77:257–286, 1989. 176, 0
   3057
4. I Stoica, R Morris, D Karger, M Kaashoek, H Balakrishnan.
   Chord: A scalable peer-to-peer lookup service for internet applications. In Proceedings of SIGCOMM 2001, 2001
   3011
5. D G Lowe.
   Distinctive Image Features from ScaleInvariants Keypoints. 0
   2992
6. J R Quinlan.
   Induction of Decision Trees. Machine Learning 1:81–106, 1986
   2858
7. R S Sutton, Barto.
   A.G.: Reinforcement Learning: An introduction. 1998
   2766
8. S Kirkpatrick, C D Gelatt, M P Vecchi.
   Optimization by simulation annealing. in Science, 1983
   2754
9. R Bryant.
   Graph-based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers, 1986

Chord

- The seminal paper on Peer-to-Peer Systems
  - Introduced the notion of “Consistent Hashing”
- The fourth most cited CS paper of all time
  - http://citeseerx.ist.psu.edu/stats/articles
  - Now 9th but highest CS systems by far
- Adapted to Dynamo/Cassandra
  - the distribution function in randomized NoSQL systems

Terminology

- Peer-to-peer systems: decentralized and distributed computer networks
- DHT = distributed hash table
  - Data structure that maps key/value pairs to different computers in a distributed system
- Key/value
  - A model for associating a name (key) with an object (value)
  - the name can be interpreted with equality queries and in some systems range (ordered) queries
  - The value is an uninterpreted object of variable size.

Peer-to-Peer Evolution

- Historical ties to SMTP and USENET
  - Content distribution based routing (SMTP) and flooding (USENET)
  - Information traded by computing peers
  - But not true “P2P” systems in that the scalability/reliability is not built into the network protocols
- Popularized by file sharing applications
- Chord defined P2P principles
- P2P underlies many systems:
  - NoSQL storage
  - Content distribution
  - Bitcoin, Spotify, Skype

Benefits of P2P

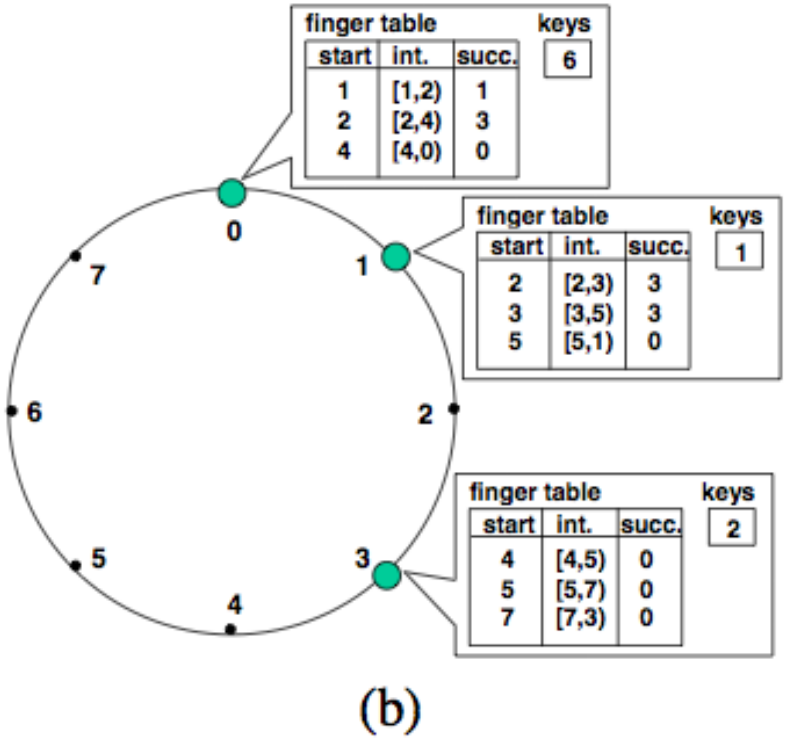
- Deployment:
  - Incremental and low initial cost
  - Contributory networks (a la Torrent) leverage resources at the edge of the Internet
- Decentralized:
  - Survivable
  - Censorship resistant
  - Unattributable (in some cases)

The Good

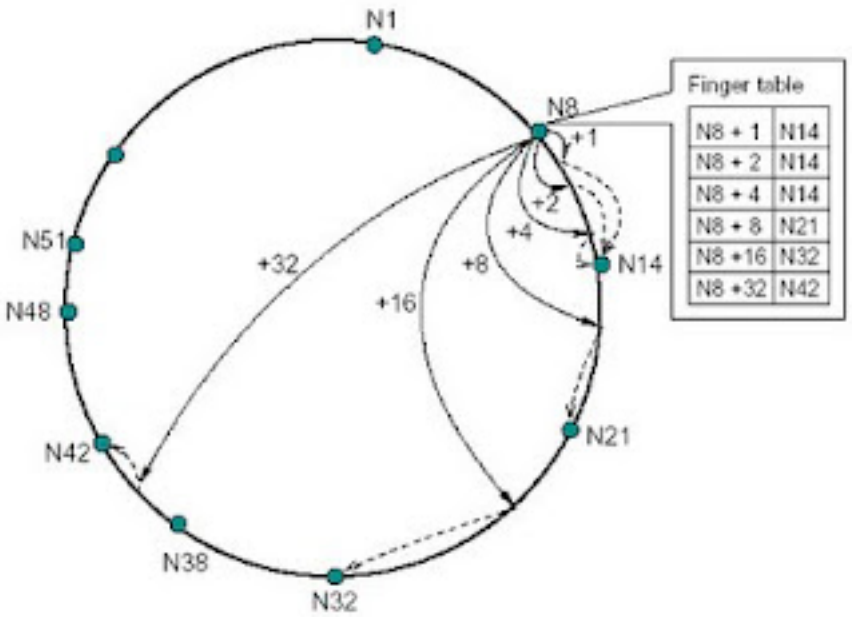
- Scalable location services
  - "A fundamental problem that confronts peer-to-peer applications is to efficiently locate the node that stores a particular data item."
  - "Chord provides support for just one operation: given a key, it maps the key onto a node."
- Communication costs and state scale logarithmically with the number of nodes
  - O(log n): Is this a lower bound?

Routing (finger table) in Chord

- Successor information (must be correct)
- Logarithmic # of pointers to other nodes



From http://wwarodomfr.blogspot.com/2008/09/chord-routing.html



What does Chord provide?

- Load balance: inherited from distributed hashing
- Availability: tolerance to failures when nodes appear/ disappear
  - Chord doesn't provide data availability: it's just a lookup service
  - Data availability is provided by mapping storage policy (3-replication) onto service, e.g. Dynamo
- Decentralization: true P2P system, no master role
- Flexible naming: no restriction on keys
  - Central concept in key, value stores

Chord Protocols

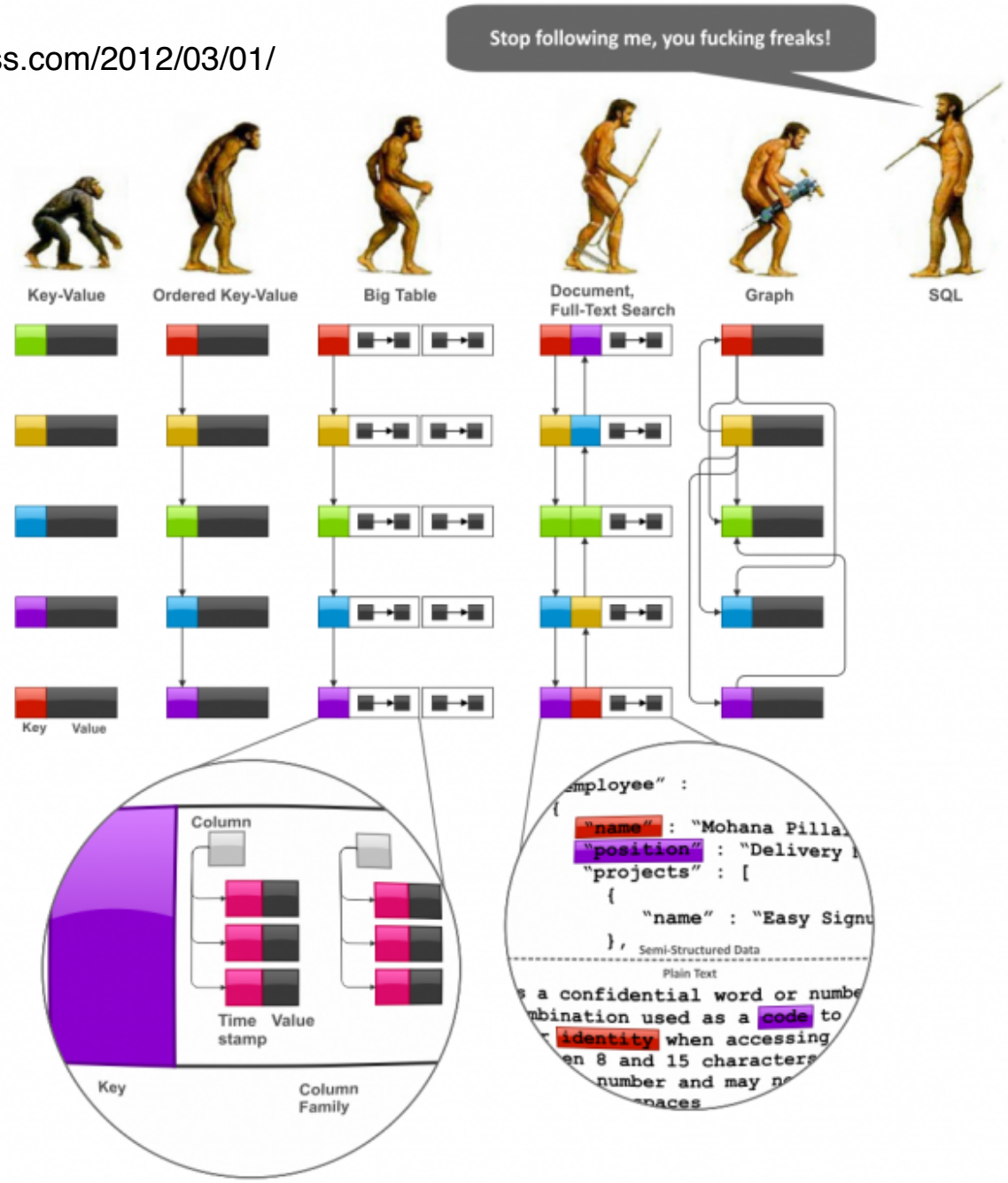
- Join/Depart protocols: orderly modifications
  - State (key/value) transfer among nodes
- Failure/recovery protocols: disorderly modifications
- Stabilization: converging to a new good state

- Issues:
  - Convergence rate (all protocols resolve in O(log2n) steps)
  - Hysteresis: concurrent failures
  - Correctness: by proof

NoSQL: Non SQL or Not Only SQL

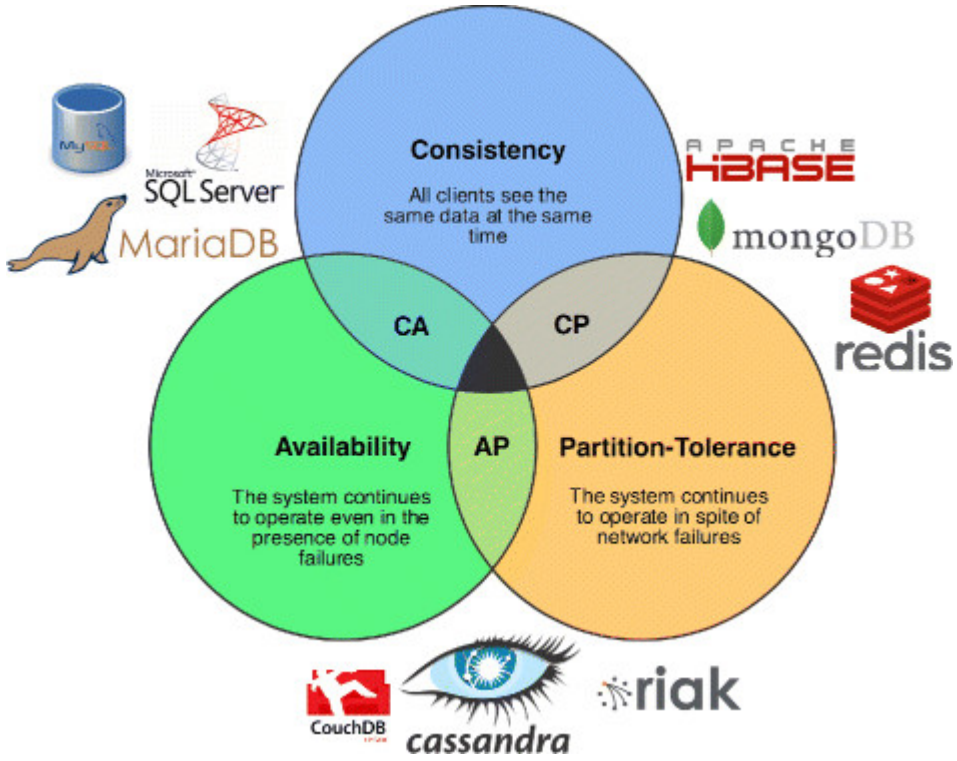
- Storage system with non-relational data models
  - Key value, ordered key value, column, document, graph
- Typically scaleout (distributed systems)
  - Associated with cloud deployments
- Explore tradeoffs in distributed system design space

From: http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/



CAP Theorem

- Conjecture, Eric Brewer, 2000 PODC keynote
  - It is impossible for a distributed computer system to provide more than 2 out of 3 of consistency, availability, partition tolerance
- Proof 2002 by Gilbert and Lynch
- Revisited Brewer 2010
  - Misconceptions about what limitations it places on distributed systems



Data Models

- Key/value: no structure to value data
  - Can support range queries on keys (ordered key index)
  - Can support no range queries (hashed key index) benefit of easier system management
- Tabular data:
  - Columns and sometimes types. Think extensible schemas.
  - Typically columns stored separately so that schemas are totally logical rather than physical
- Document databases:
  - Full text searchability within fields

Big Table: Defining Tabular

A Bigtable is a sparse, distributed, persistent multi-dimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes

- Implications: accessible by row (indexed)
- Accessible by column (schema)
- Values are unintrepreted (no type in BigTable)
- Timestamp: time evolving views of data (snapshots)

Why tabular?

- It adds the notion of columns to the key value store
  - A second dimension of indexing
- This translates into dynamic schemas
  - Create multiple columns that share rows. This can be viewed as a table.
  - Arbitrary views (collections) of columns can be evaluated dynamically
- Implication: one can run ALTER TABLE for free. The schema is not physically encoded in the disk storage so no reorganization costs.
  - This concept is being used in “column databases” too

Interpreting the Data Model

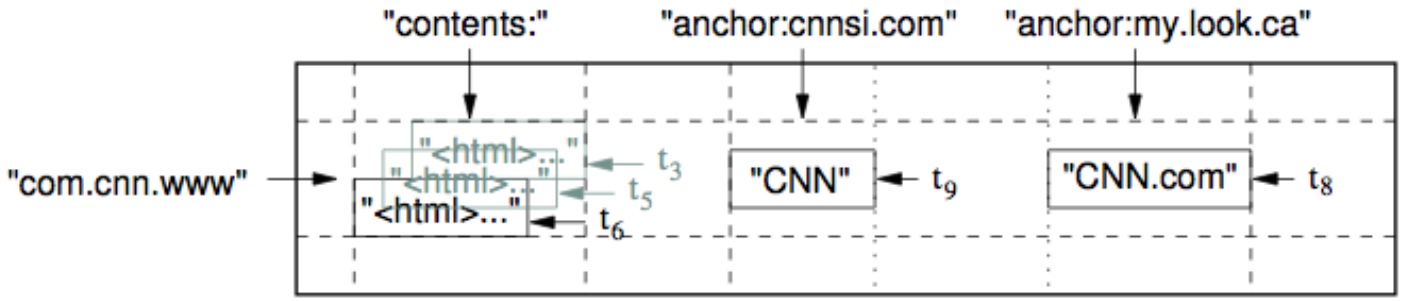


Figure 1: A slice of an example table that stores Web pages. The row name is a reversed URL. The contents column family contains the page contents, and the anchor column family contains the text of any anchors that reference the page. CNN's home page is referenced by both the Sports Illustrated and the MY-look home pages, so the row contains columns named anchor:cnnsi.com and anchor:my.look.ca. Each anchor cell has one version; the contents column has three versions, at timestamps t_3, t_5, and t_6.

- One row, three columns (shown)
- Different views of data (as a function of time)
- Column "families", e.g. anchor
  - Hint that data are of the same type (compressed together)
  - Can be created without any data in them
  - Hundreds of column families. Arbitrary numbers of columns.

Tablets:

"Bigtable maintains data in lexicographic order by row key. The row range for a table is dynamically partitioned. Each row range is called a *tablet,* which is the unit of distribution and load balancing."

- Each column indexed lexicographically by row.
- Hierarchy of tablets form a B+-tree-like data structure
  - Metadata in internal nodes and data only at leaves.

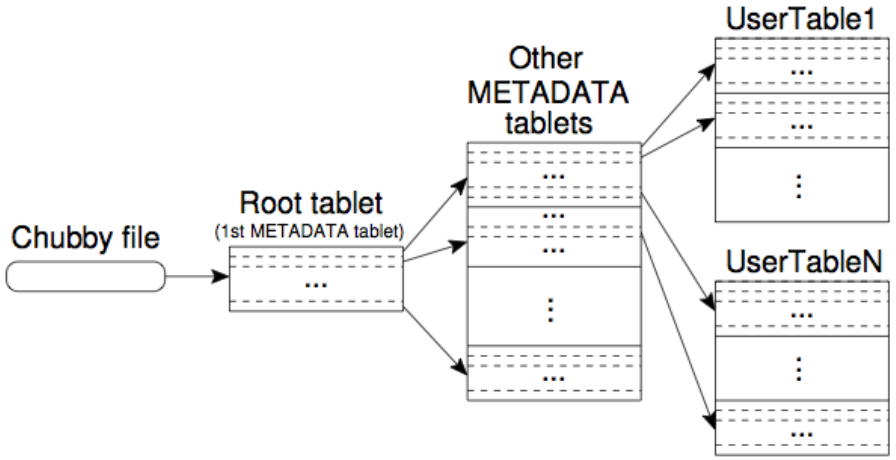


Figure 4: Tablet location hierarchy.

BigTable and Locking

- BigTable is strongly consistent
  - Relies on distributed Paxos
- Only available when lock service (chubby) is available
- Uses locking of ROOT table (and recoverability of locks) to maintain a single system image.

- In terms of the CAP theorem
  - CA (consistency and availability)
  - No P (partition tolerance)

Amazon Dynamo: AP not C

- Allow for divergent replicas and system delusion
- To the benefit of scalability and node autonomy

Table 1: Summary of techniques used in Dynamo and their advantages.

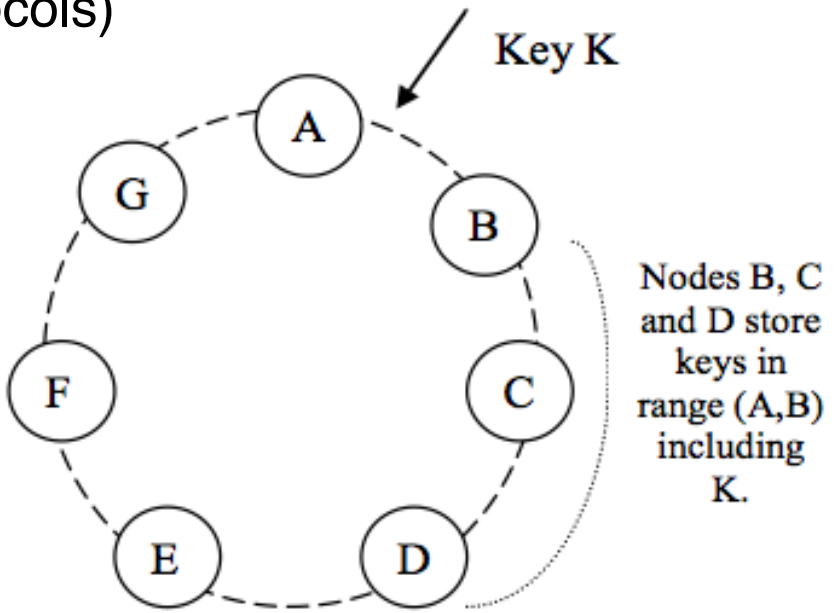
Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

Divergent Replicas

- Concurrent updates to multiple replicas
  - Requires reconciliation
  - Determine conflicting updates and reconciliation
- Clever use of data structures
  - Merkle trees from snapshot/versioning literature
  - Vector clocks from Bayou

Hashing and Replica Placement

- Key/value mapping of data to three nodes
  - Three replicas
- Each write updates to whatever membership set it thinks it sees
  - These may be different, but should be symmetric within groups (a property of gossip protocols)



Data Model and CAP

- Does BigTable's schema require consistency?
- Does consistent hashing require key/value?

"Bigtable is a distributed storage system for managing structured data. It maintains a sparse, multi-dimensional sorted map and allows applications to access their data using multiple attributes [2]. Compared to Bigtable, Dynamo targets applications that require only key/value access with primary focus on high availability where updates are not rejected even in the wake of network partitions or server failures."

Back to Chord

- Actually an arugment for:
  - Software verification
  - Model testing

Chord Protocols

- Join/Depart protocols: orderly modifications
  - State (key/value) transfer among nodes
- Failure/recovery protocols: disorderly modifications
- Stabilization: converging to a new good state

- Issues:
  - Convergence rate (all protocols resolve in O(log2n) steps)
  - Hysteresis: concurrent failures
  - Correctness: by proof

The Bad

- Many theorems and proofs (almost all wrong)
  - http://www2.research.att.com/~pamela/chord-orig.pdf
  - “Not one of the six properties claimed invariant for the full protocol in [9] is invariantly true.”
- Use Alloy to model the Chord protocol
  - Formal modelling language (pseudocode) and automated analysis (invariant proving)
  - Based on model enumeration:
  - Determines several easy “bug” fixes and shows that the protocol is still not correct
  - Provide a corrected protocol with all the same properties

A Counter-example

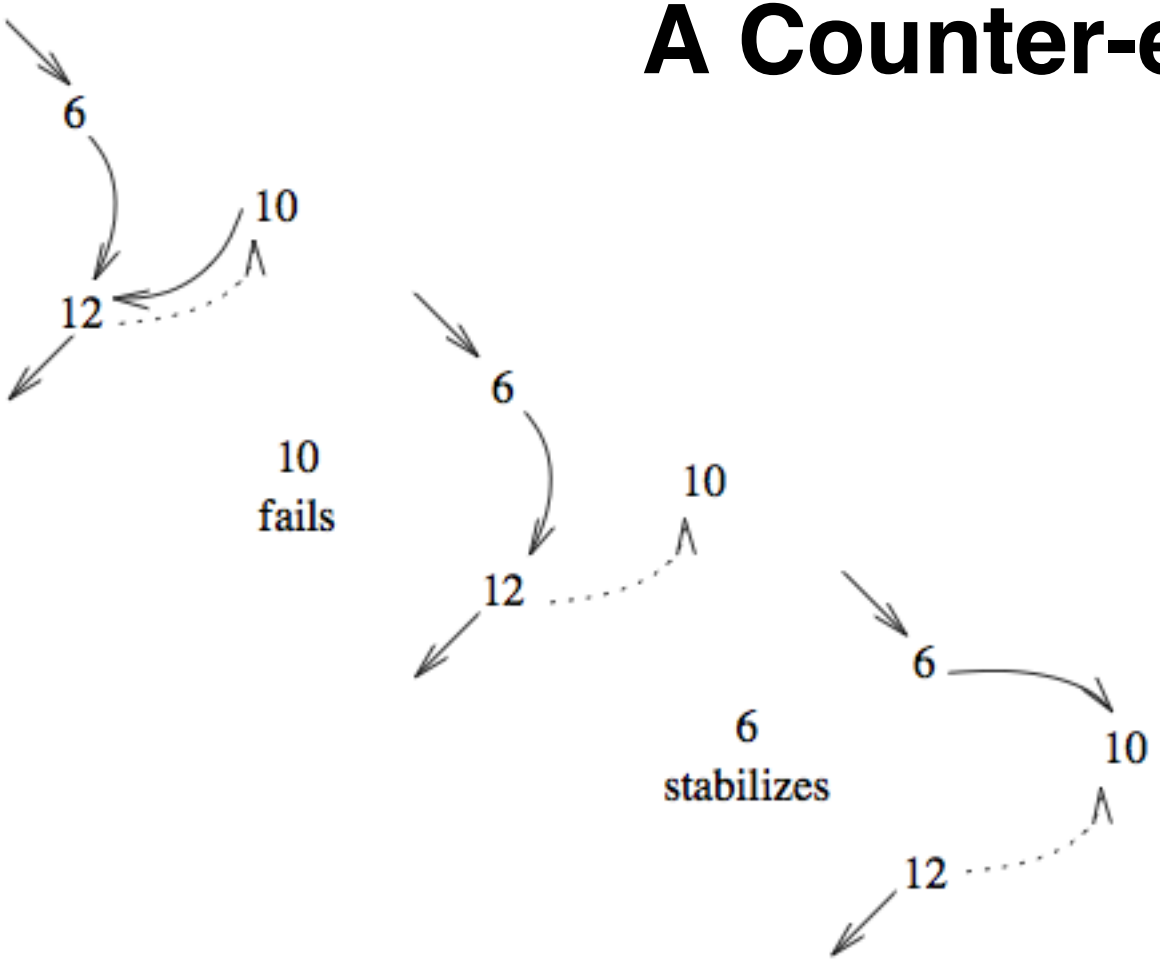


Figure 9: Three stages (upper left to lower right) creating a counterexample to correctness of the Chord protocol.

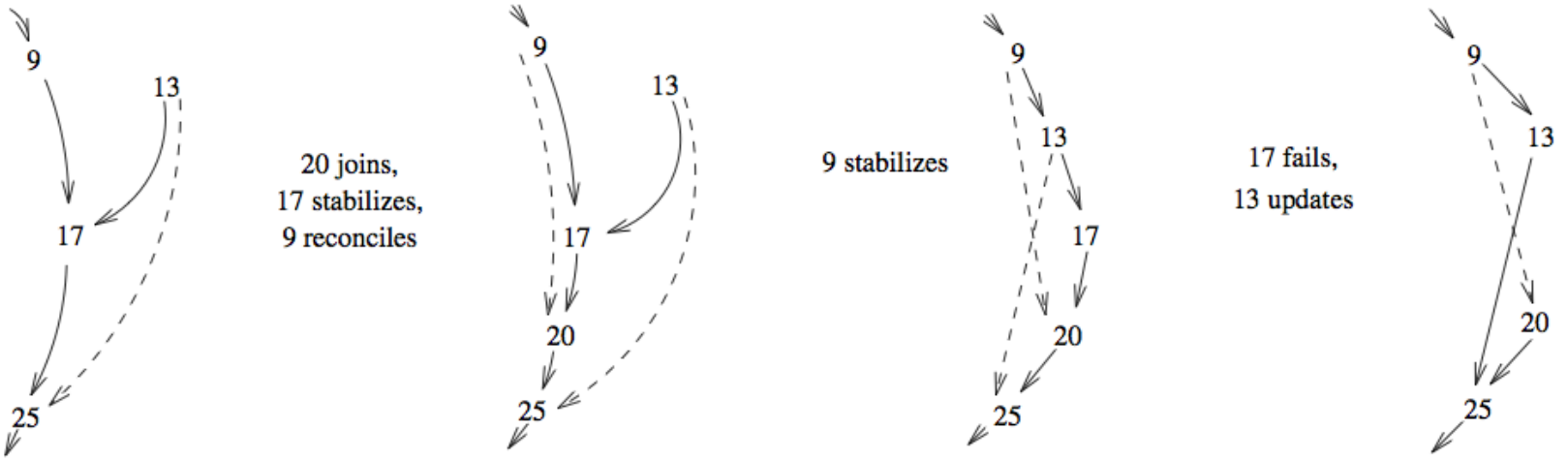


Figure 10: Three stages (left to right) creating a counterexample to `ValidSuccessorList`, and its effect (fourth stage) after a failure. Solid arrows represent primary successors, while dashed arrows represent secondary successors.

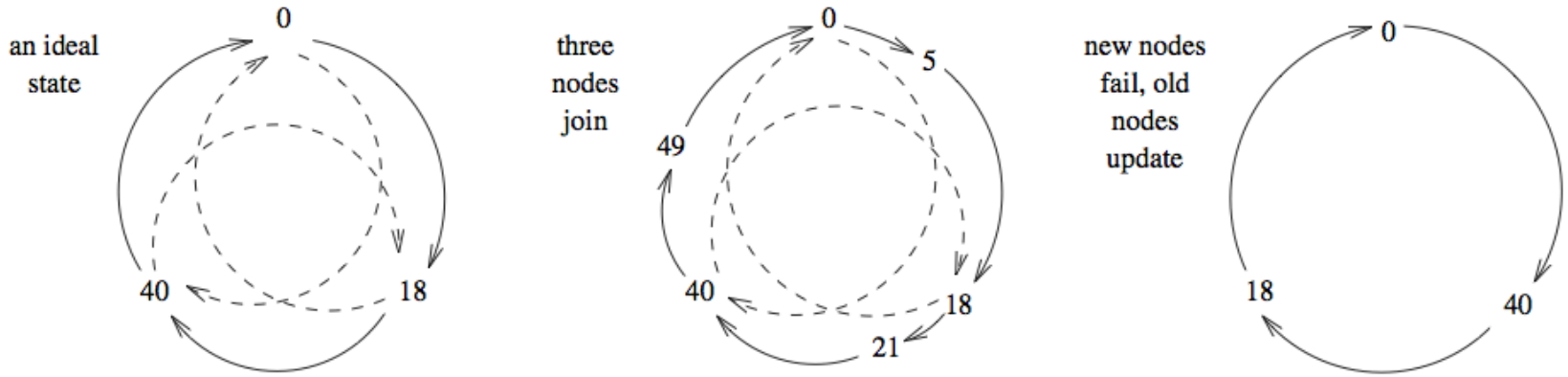


Figure 11: Three stages (left to right) creating a counterexample to correctness of the Chord protocol.

How bad is wrong?

- Doesn't invalidate the system
  - Principles fundamentally correct
  - "The point of these observations is not to disparage the designers of Chord, whose creativity and insight are impressive."
- Several conclusions
  - Requirements and invariants hard to get right
  - Informal "proofs" are suspect
  - Model proving and model enumerate are powerful
- IMO: proving code (verifiable C) is much more powerful than pseudocode (Alloy and TLA)
  - Can't run pseudocode