UNDERSTANDING AND IMPLEMENTING CHORD

Pamela Zave

Princeton University

Princeton, New Jersey

THE CHORD PROTOCOL MAINTAINS A PEER-TO-PEER NETWORK

identifier of a node (assumed unique) is an m-bit hash of its IP address

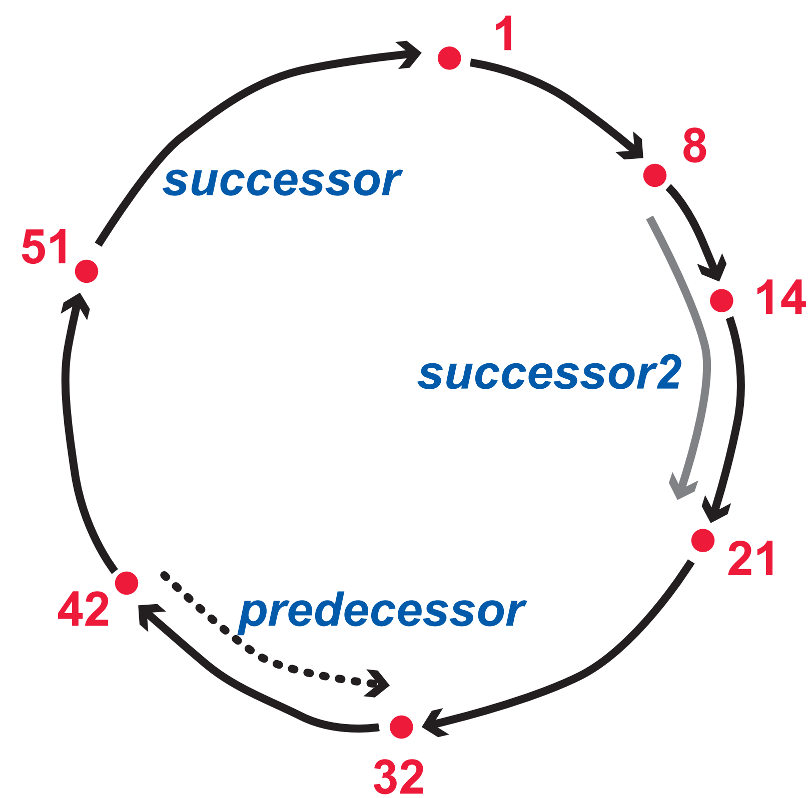
here, the identifier space consist of all 6-bit binary values

“consistent hashing” means that all IP addresses are spread evenly around the identifer space

nodes are arranged in a ring, each node having a successor pointer to the next node (in integer order with wraparound at 0)

redundant pointers support fault-tolerance (extra successors, predecessors)

the protocol preserves the ring structure as nodes join, leave silently, or fail



CHORD AS A DISTRIBUTED HASH TABLE

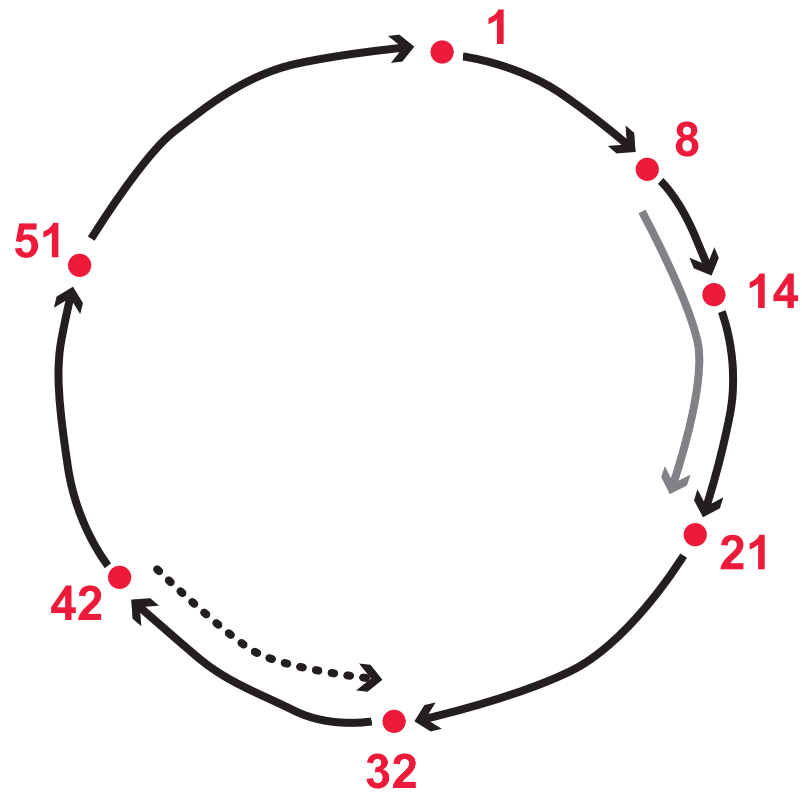
a hash table stores (key, value) pairs

the keys are the same as identifiers

keys 22 through 32 are
stored at 32

keys 33 through 42 are
stored at 42,

etc.



to look up a value, you only need to know the IP address of one Chord member

a list of accessible members is the only central administration needed!

your query to that member is forwarded around the ring until it gets to the member that has the value

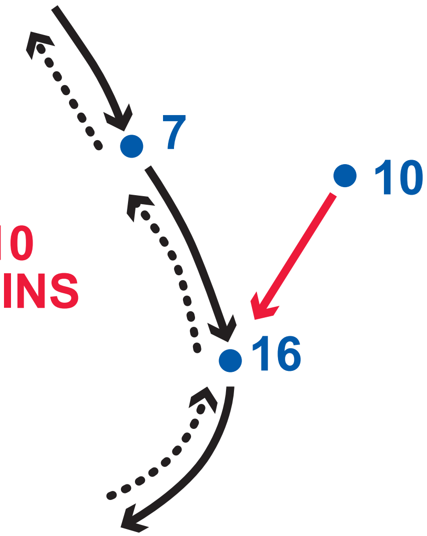
as an optimization, "finger" pointers go across the ring, make lookup faster

OPERATIONS OF THE PROTOCOL

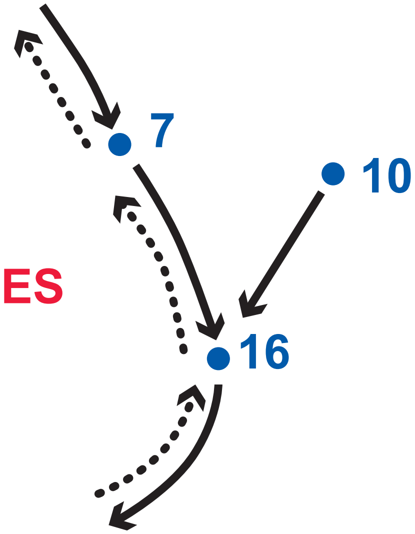
an operation changes
the state of one member

Join and Stabilize are scheduled autonomously,
GetsNotified is caused by another member's Stabilize

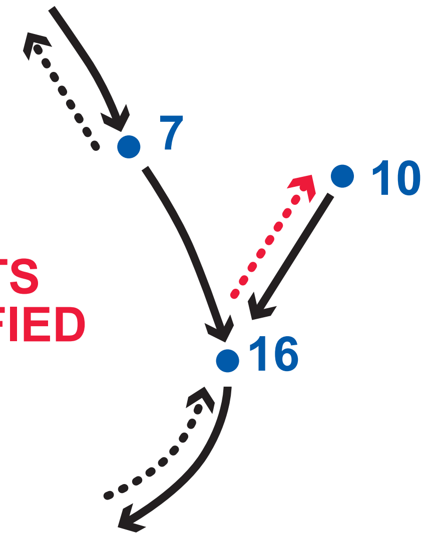
10
JOINS



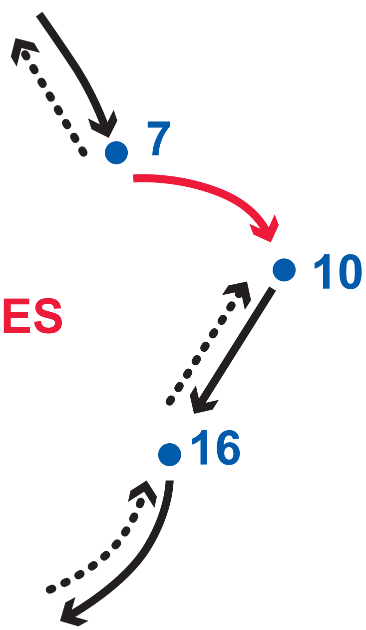
10
STABILIZES



16
GETS
NOTIFIED



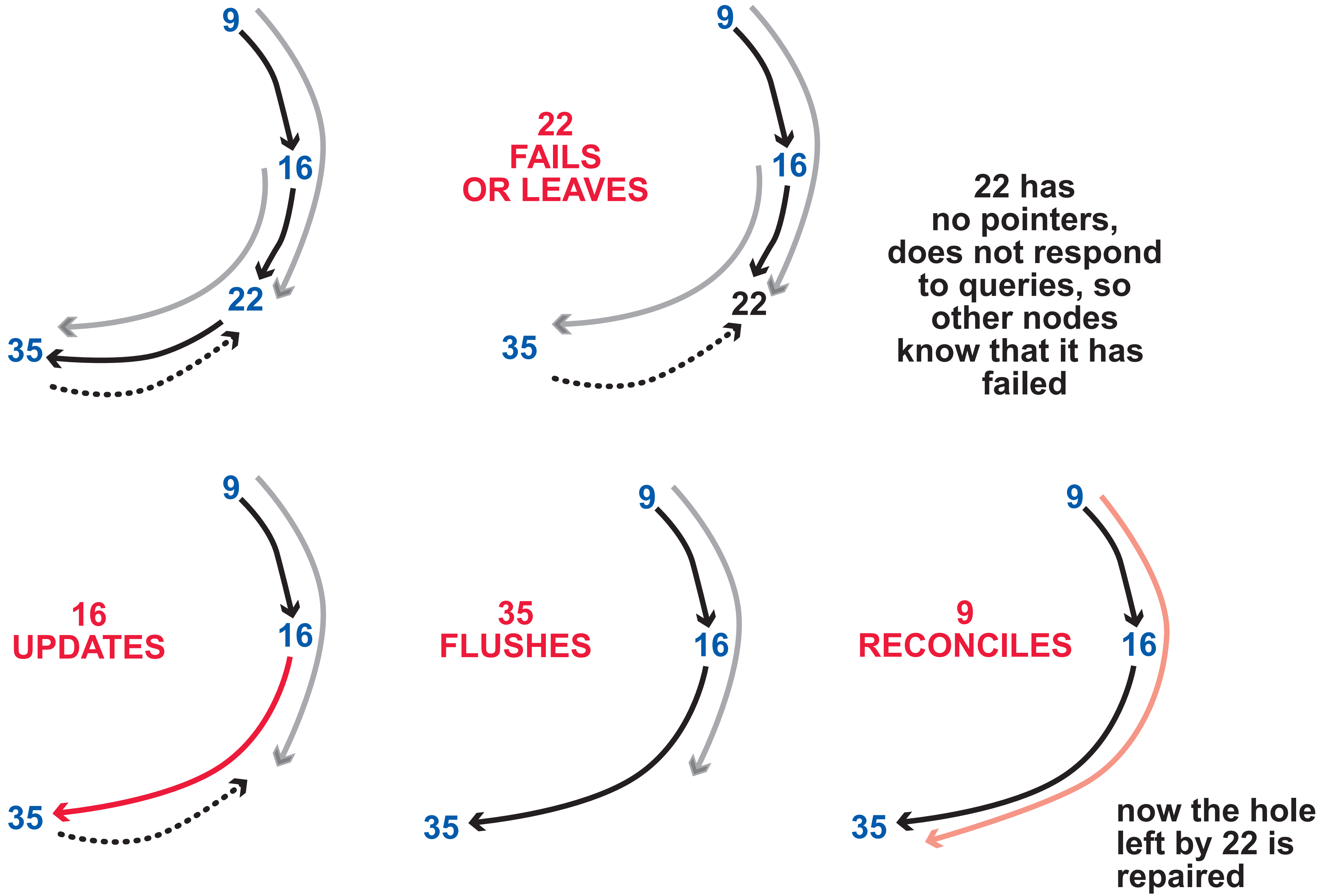
7
STABILIZES



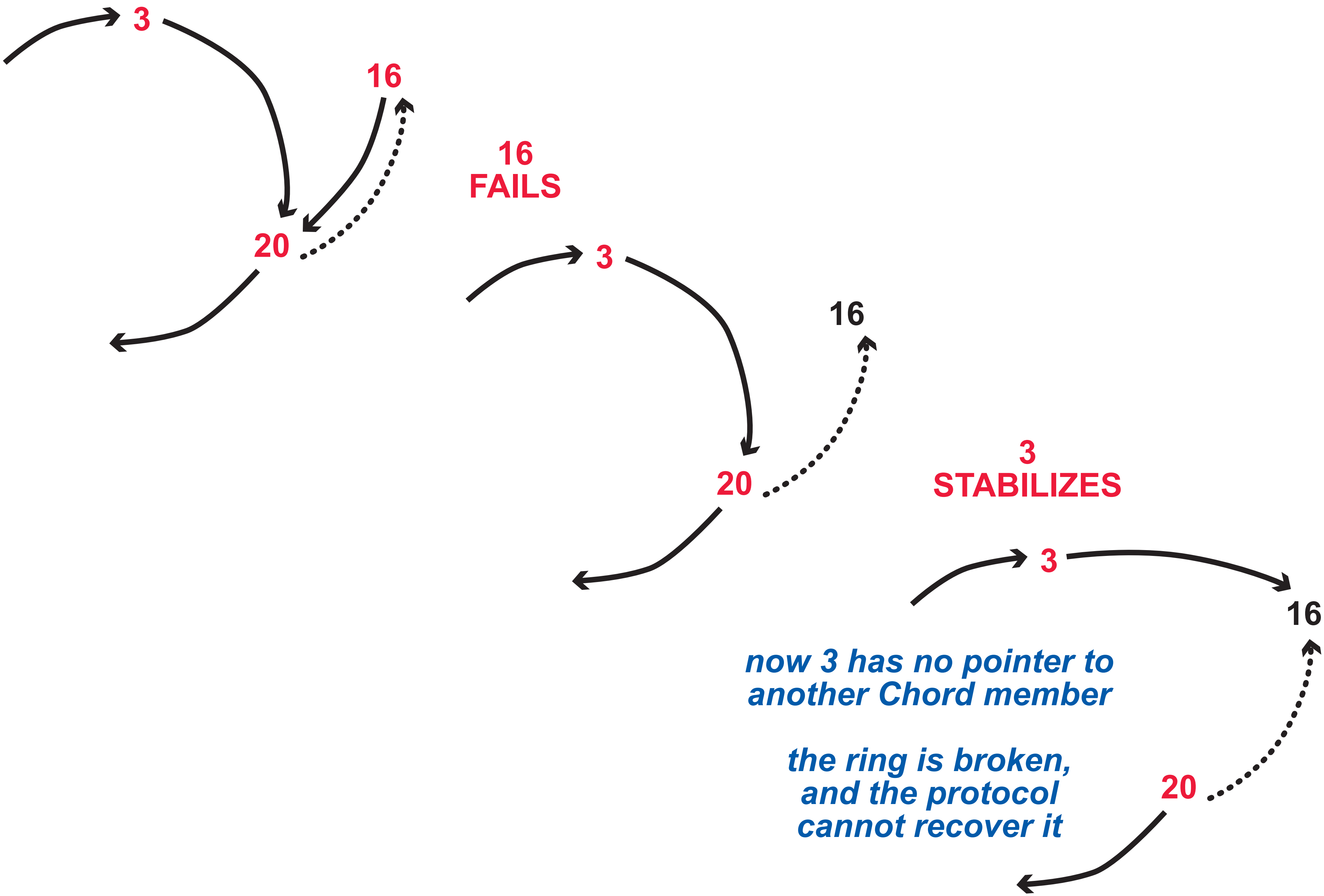
10
GETS
NOTIFIED

now 10 is completely
incorporated into
the ring!

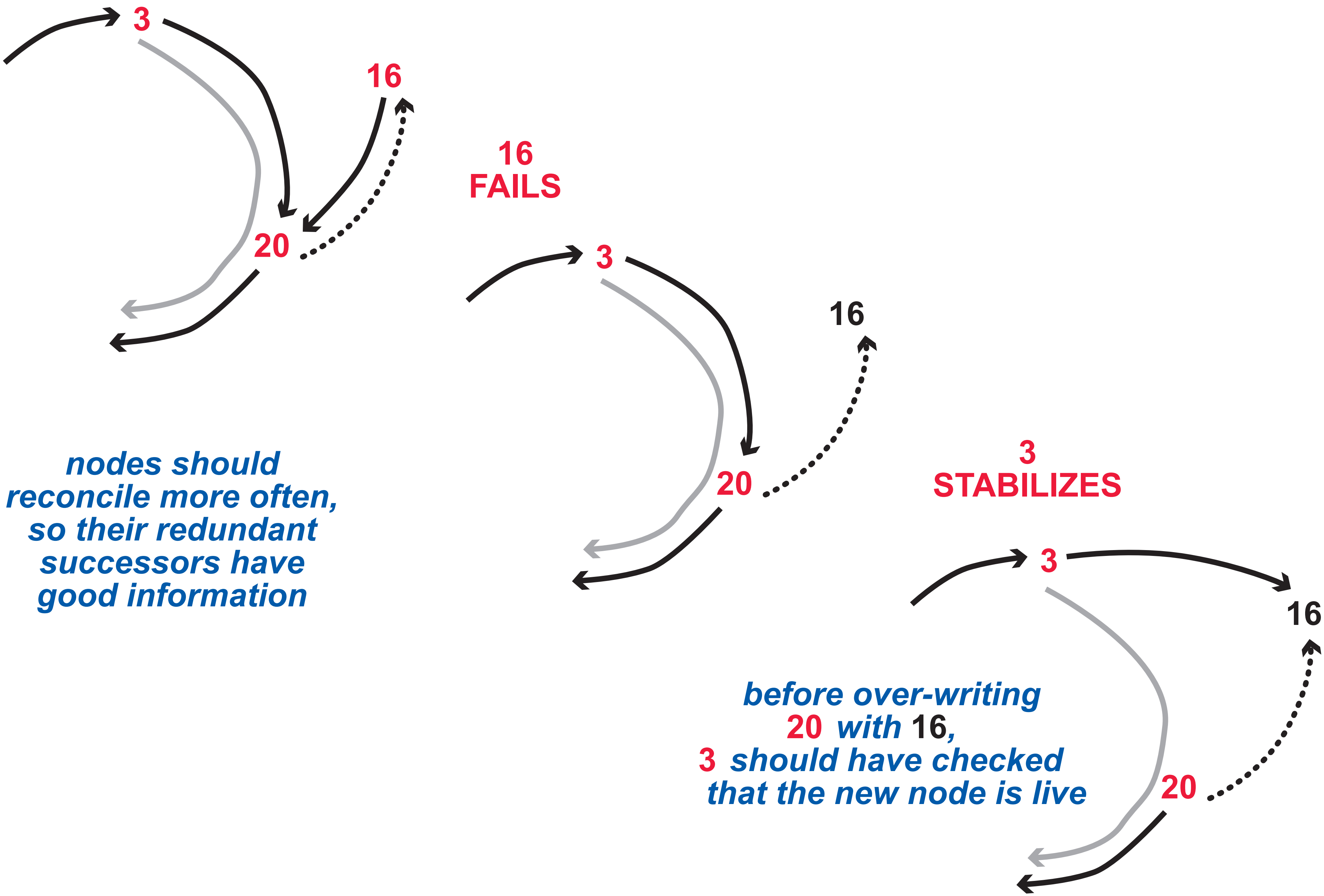
MORE OPERATIONS OF THE PROTOCOL



WHAT YOUR IMPLEMENTATION MIGHT DO



WHAT YOUR IMPLEMENTATION MIGHT DO



DO YOU THINK YOU WOULD FIND AND FIX ALL THESE LITTLE PROBLEMS?

DHTS (there are others besides Chord) have a reputation for being unreliable

when a distributed system fails, it is very difficult to find out why!

Donald Kossmann writes about an experiment he did with M.S. students in computer science at ETH in 2008:

"As part of a lab project on distributed systems, I asked the 8 students to implement the Chord protocol based on the Chord paper *[same one you read]*.

Each student had his/her own implementation. The ETH students are pretty good systems builders so they all did fairly well and got something running: We had a small test suite and all implementations passed that.

Then, I asked the students to do the second part of the project:

- **Run a Chord ring with your own implementation X.**
- **Then add a new node to this ring, but that new node uses Chord implementation Y from another student.**

There was no combination of X and Y (not even if there was only one X node and one Y node) in which two nodes with different Chord implementations could even exchange a message. The interesting thing was that the problem was not serialization or message format or use of different TCP libraries or so. Those would have been easy fixes. There was no way for the students to fix the problem and get anything running even though they would get extra credit if they succeeded.

We allocated 10 weeks for the first part and 4 weeks for the second part. 4 weeks were not enough!"

WHY IS CHORD IMPORTANT TO YOU?

the 2001 SIGCOMM paper introducing Chord is one of the most-referenced papers in computer science, . . .

. . . and won SIGCOMM's 2011 Test of Time Award

APPLICATIONS

- allows millions of *ad hoc* peers to cooperate
- WIDELY used as a building block in distributed fault-tolerant applications

  if you are a Ph.D. student, your dissertation may use Chord!
- the best-known application is BitTorrent

WHY MIGHT YOU CHOOSE CHORD TO IMPLEMENT YOUR DISSERTATION IDEA?

"Three features that distinguish Chord from many other peer-to-peer lookup protocols are . . .

. . . its simplicity,

. . . provable correctness,

. . . and provable performance."

WHY IS CHORD IMPORTANT TO YOU?

the 2001 SIGCOMM paper introducing Chord
is one of the most-referenced
papers in computer science, . . .

. . . and won SIGCOMM's 2011 Test of Time Award

APPLICATIONS

- **allows millions of *ad hoc* peers to cooperate**
- **WIDELY used as a building block in distributed fault-tolerant applications**

  if you are a Ph.D. student,
  your dissertation
  may use Chord!

- **the best-known application is BitTorrent**

WHY MIGHT YOU CHOOSE CHORD TO IMPLEMENT YOUR DISSERTATION IDEA?

"Three features that distinguish Chord from many other peer-to-peer lookup protocols are . . .

. . . its simplicity, **TRUE!**

. . . provable correctness, **OOPS!**

. . . and provable performance." **TRUE!**

THE CLAIMS

Correctness Property:

In any execution state, IF there are no subsequent Join or Fail events, . . .

. . . THEN eventually . . .

. . . all pointers in the network will be globally correct, and remain so.

I found these problems by analyzing a small Alloy model

THE REALITY

- even with simple bugs fixed and optimistic assumptions about atomicity, the original protocol is not correct

- of the seven properties claimed invariant of the original version, not one is actually an invariant

not surprisingly, due to sloppy informal specification and proof

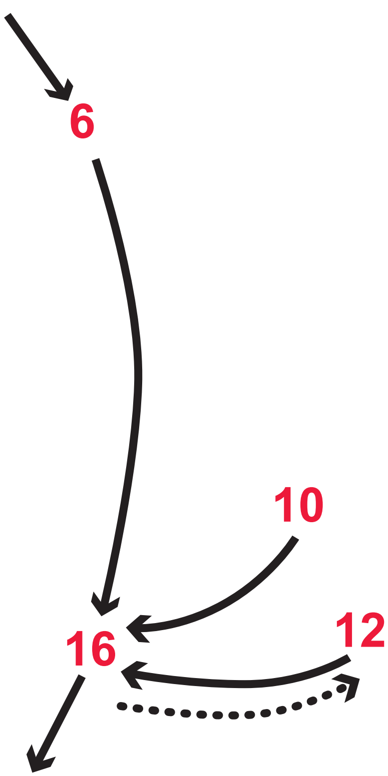
Chris Newcombe and others at AWS credit this work with overcoming their bias against formal methods, which they now use to find bugs.

[CACM, April 2015]

“EVENTUAL REACHABILITY” IS NOT THE ONLY ISSUE

OrderedMerges . . .

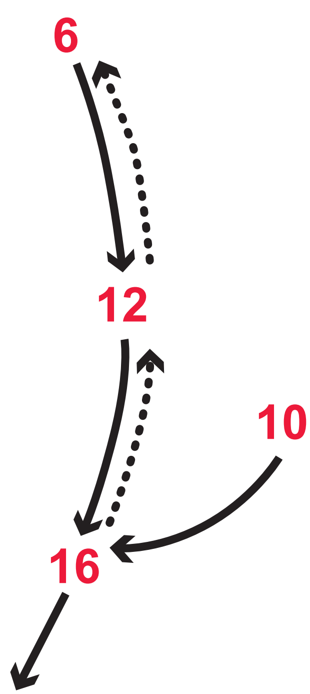
. . . means that appendages merge in the correct places, as they do here



6 STABILIZES,

12 GETS NOTIFIED

***OrderedMerges* is easily violated**



VIOLATIONS OF *OrderedMerges*

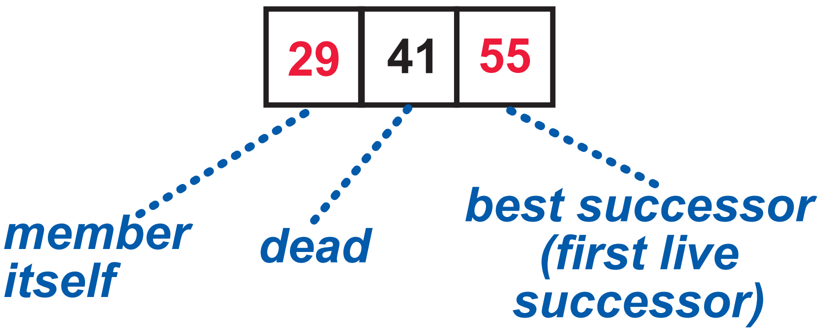
- **are not incorrect**
- **invalidate some assumptions used in performance analysis**
- **can be demonstrated in Chord networks with 3 nodes**

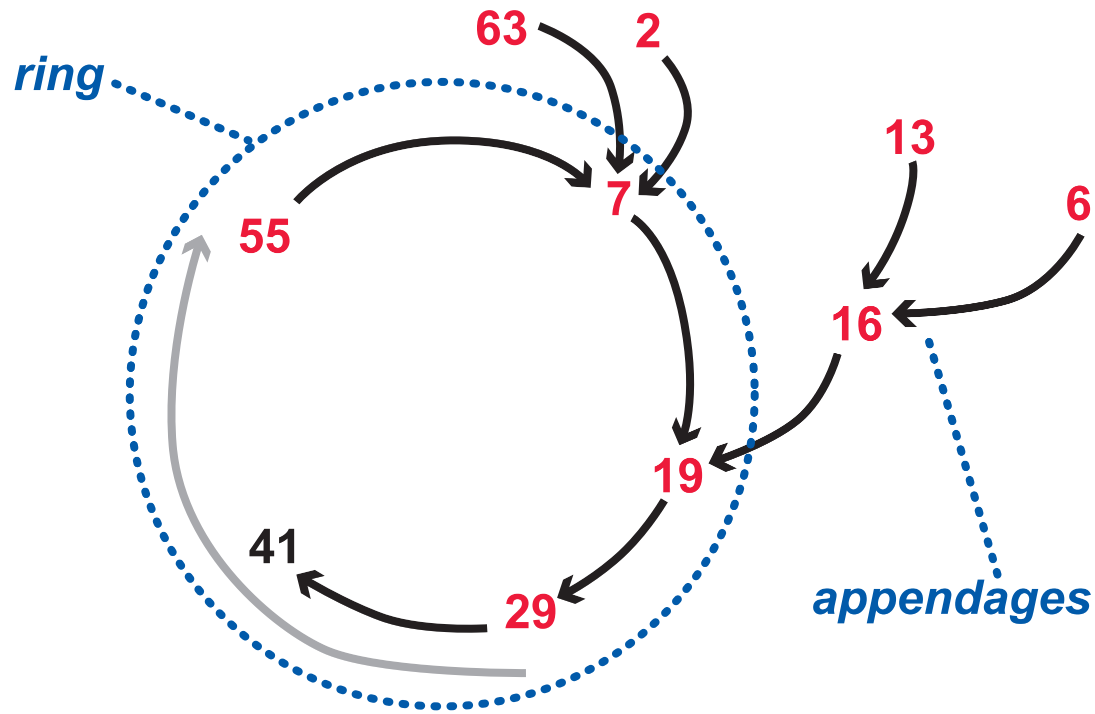
how could they go unknown for ten years?

this is why formal methods are so important

BASIC CORRECTNESS STRATEGY 1

extended successor list (ESL) of 29 (with L = 2):





Original operating assumption:

No failure leaves a member without a live successor.

Clearly, with L = 2, Chord is intended to tolerate one failure in a neighborhood, but not two.

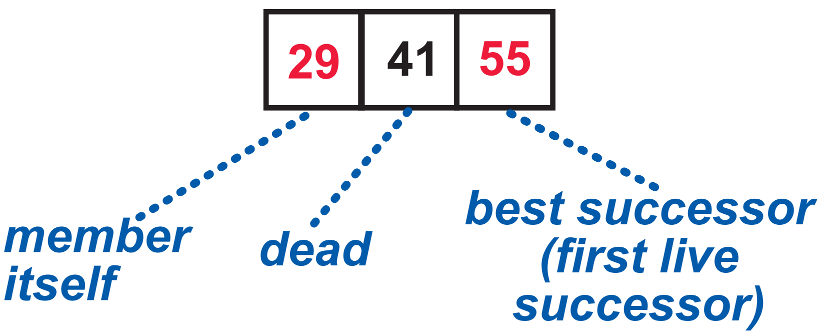
But if an ESL with L = 2 is . . .

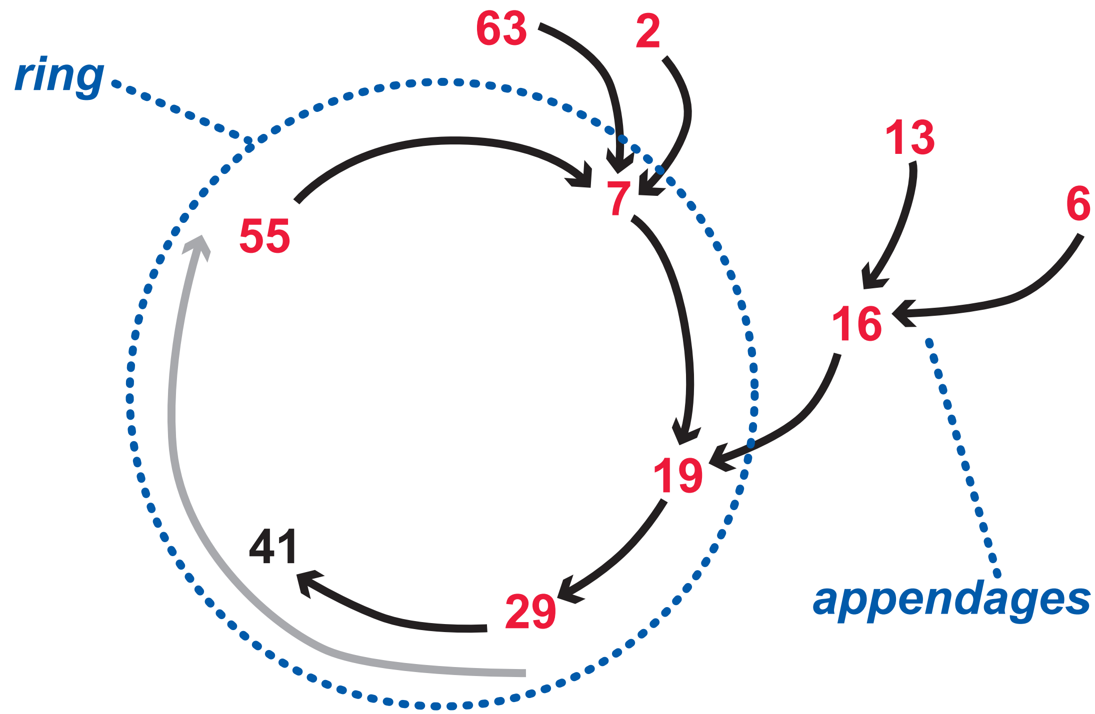
29	32	29

. . . then 32 cannot fail!

BASIC CORRECTNESS STRATEGY 2

extended successor list (ESL)
of 29 (with L = 2):





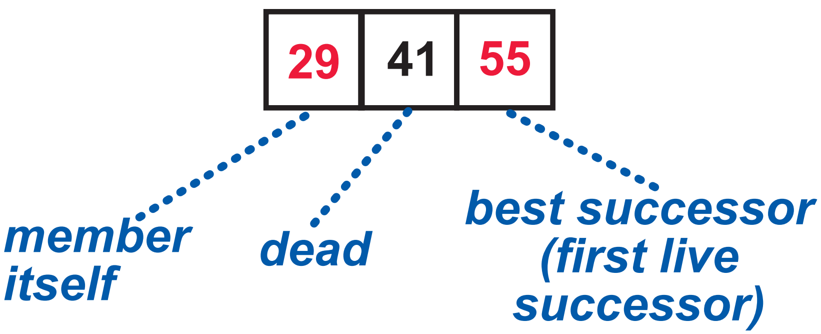
Definition of *FullSuccessorLists:*
The extended successor list of each member has L+1 distinct entries.

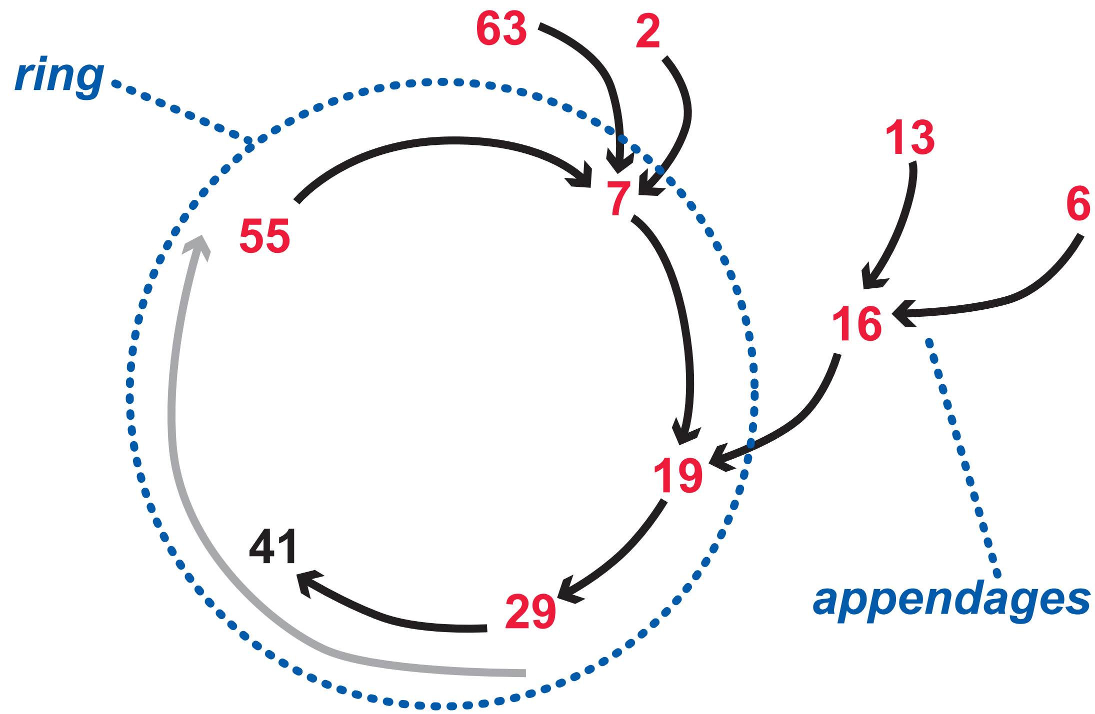
New operating assumption:

If a Chord network has the property *FullSuccessorLists*, then no failure leaves a member without a live successor.

BASIC CORRECTNESS STRATEGY 2

extended successor list (ESL)
of 29 (with L = 2):





Definition of *FullSuccessorLists:*
The extended successor list of each
member has L+1 distinct entries.

New operating assumption:

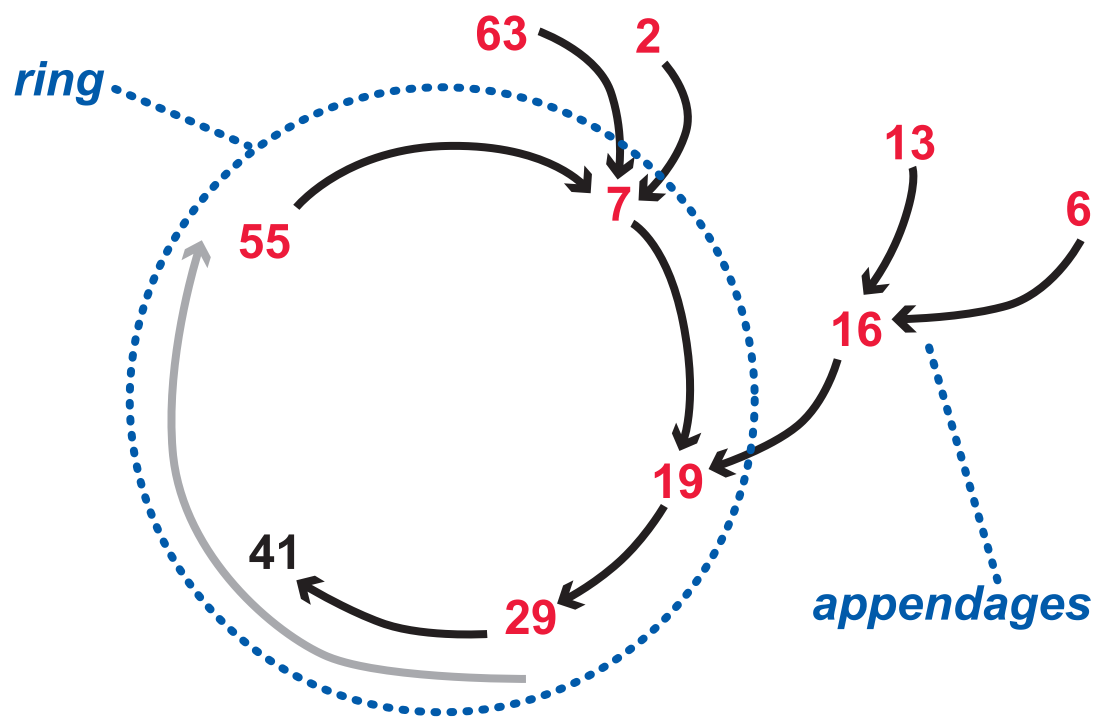
If a Chord network has the property
FullSuccessorLists, then no failure leaves
a member without a live successor.

if not satisfied for the real failure rate,

. . . increase rate of stabilization,

. . . or increase redundancy

BASIC CORRECTNESS STRATEGY 3



TO MAKE ORIGINAL CHORD CORRECT:

- alter the initialization to satisfy *FullSuccessorLists* with all members live

  requires L+1 members

- alter the operations to populate successor lists more eagerly, preserve *FullSuccessorLists* at all times

now it is roughly correct (in hindsight) but how do we prove it without an invariant?

WHY IS FINDING AN INVARIANT SO DIFFICULT?

THE KNOWN, NECESSARY PROPERTIES ARE STATED IN TERMS OF THE RING . . .

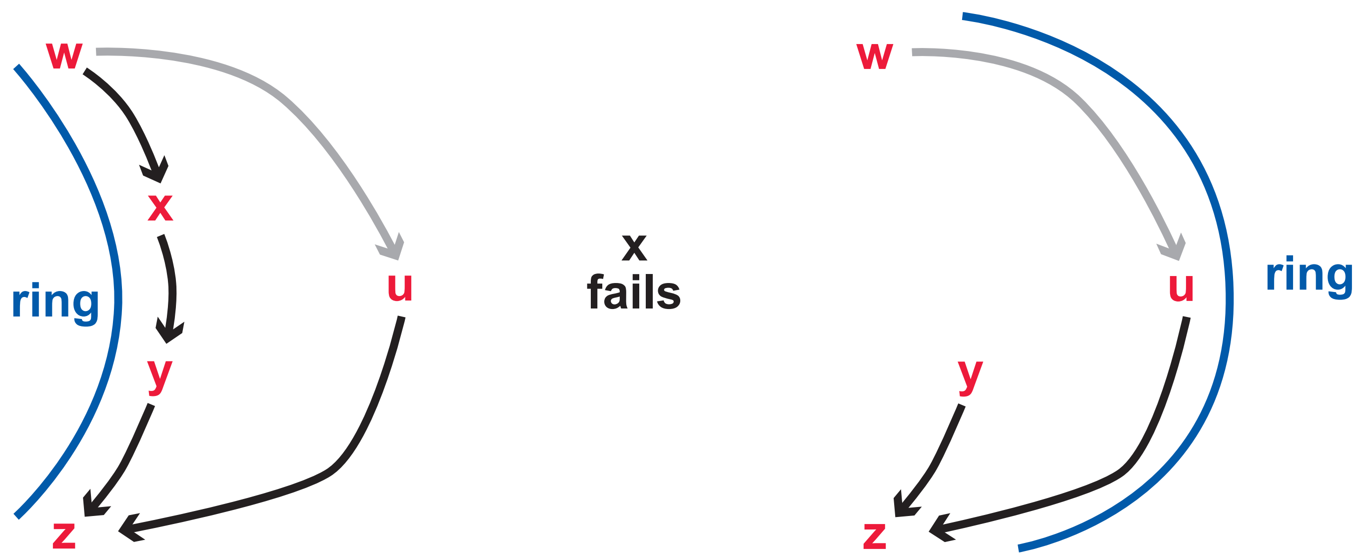
- there is a ring of best successors
- there is no more than one ring
- on the unique ring, the members are in identifier order

about the ring of best successors

- from each appendage member, the ring is reachable through best successors

about the appendages

. . . BUT "RING VERSUS APPENDAGE" IS CONTEXT-DEPENDENT AND FLUID:



AN INTERMEDIATE RESULT

ANOTHER OPERATING ASSUMPTION:

A chord network has a *stable base* of L+1 nodes that are always members.

expensive to implement these high-availability nodes!

a stable base would have 3-6 members, while a Chord network can have millions of members—what is the base doing?

I believe it is just preventing anomalies in small networks, but how can we know for sure?

THE INDUCTIVE INVARIANT:

OneOrderedRing

and ConnectedAppendages

and BaseNotSkipped

no successor list skips over a member of the stable base

THE PROOF OF CORRECTNESS:

by exhaustive enumeration, in Alloy, for all model instances up to N = 9, L = 3

THE FINAL RESULT

ANOTHER OPERATING ASSUMPTION:

None

this is just a formalization of the original operating assumption

THE INDUCTIVE INVARIANT:

OneLiveSuccessor

and SufficientPrincipals

Definition of a *principal member:*
A member that is not skipped by any member's successor list.

Definition of *SufficientPrincipals:*
There are at least L+1 principal nodes.

THE PROOF OF CORRECTNESS:

the "stable base" has become something we can prove, rather than an assumption!

informal and intuitive, but . . .

. . . a real proof (no size limits)

. . . backed up by an Alloy model checked up to N = 9, L = 3
(as a protection against human error)

CONCLUSIONS

SPECIFICATION OF A CORRECT VERSION OF CHORD

- initialization is more difficult than original Chord, but a simple protocol will get networks off to a safe start

- otherwise correct Chord is just as efficient as original Chord

  it is an impressive pattern for fault-tolerance

- these peer-to-peer protocols have a (justified) reputation for unreliability

- a correct specification could pave the way for a new generation of reliable, more useful implementations

  it also provides a firm foundation for work on better failure detection and security

FOR YOUR DISSERTATION, ONLY THE BEST WILL DO!

use an implementation based on

"Reasoning about identifier spaces: How to make Chord correct", Pamela Zave, *IEEE Transactions on Software Engineering*

APPENDIX:

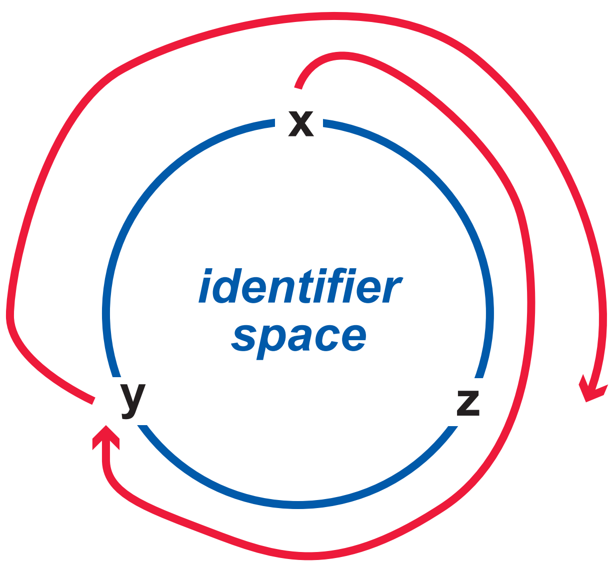
AN OUTLINE OF THE PROOF

ORDERED SUCCESSOR LISTS . . .

. . . ARE IMPLIED BY THE INVARIANT

Definition of *OrderedSuccessorLists*:
For all distinct identifiers *x, y, z*, and sublists *[x, y, z]* of an ESL (whether the sublist is contiguous or not) . . .
between [x, y, z].

hypothesize a disordered extended successor list
[. . . x, . . . y, . . . z, . . .]



Proof of *OrderedSuccessorLists*

[x, . . . y, . . . z] must include L + 1 principal nodes — ***picture, principal nodes not skipped, Sufficient Principals***

x is either not a principal node, or is duplicated in [y, . . . z] — ***between [y, x, z] in identifier space***

same reasoning for z

so the length of [x, . . . y, . . . z] is at least L + 3 — ***(L + 1) plus one x and one z***

but the length of an ESL is always L + 1

CONTRADICTION!

THE PRINCIPAL NODES MAKE THE SHAPE OF THE RING

every member has a best successor (first live successor)

there are sufficient principal nodes

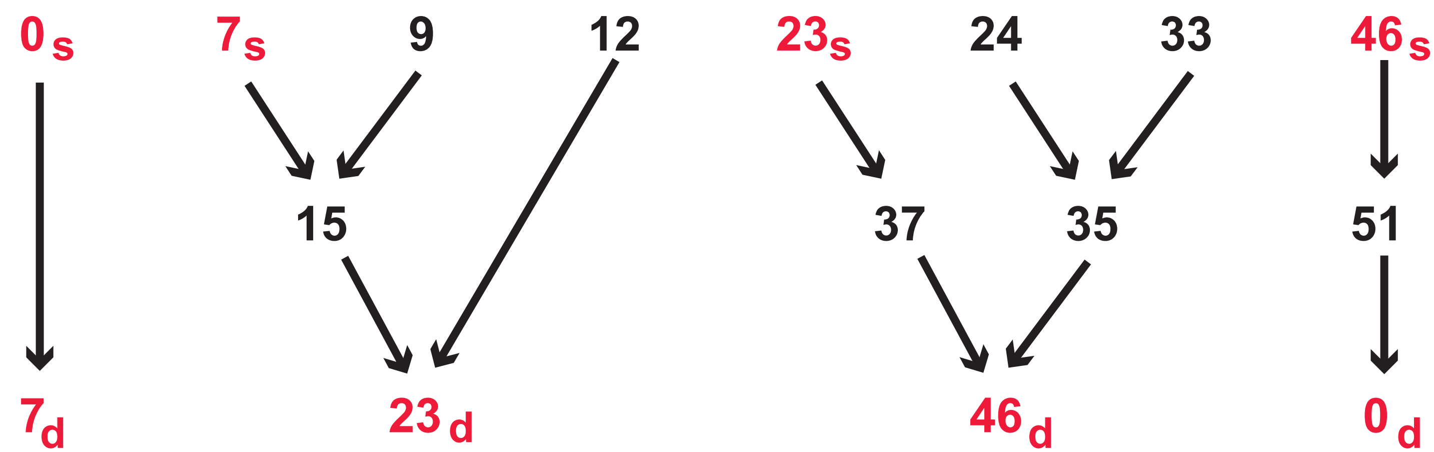
here is a graph of best successors:

these paths . . .

. . . do not skip principal nodes

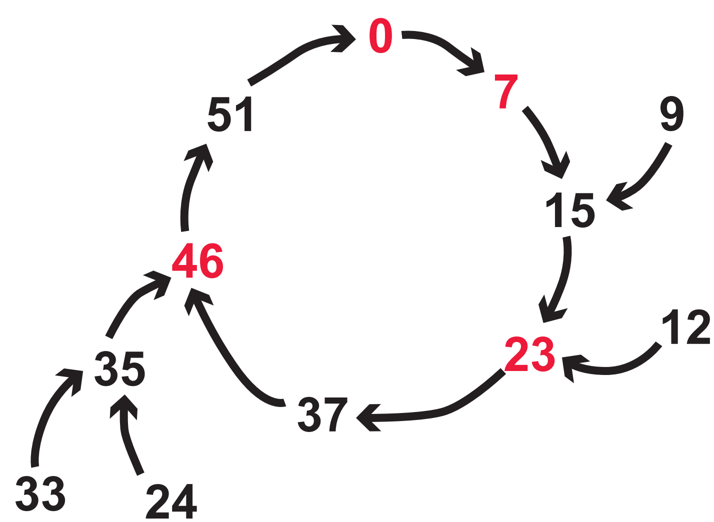
. . . are acyclic

. . . are ordered by identifiers



each tree has exactly one p_s, which is unique to it

so the re-arranged graph must look like this

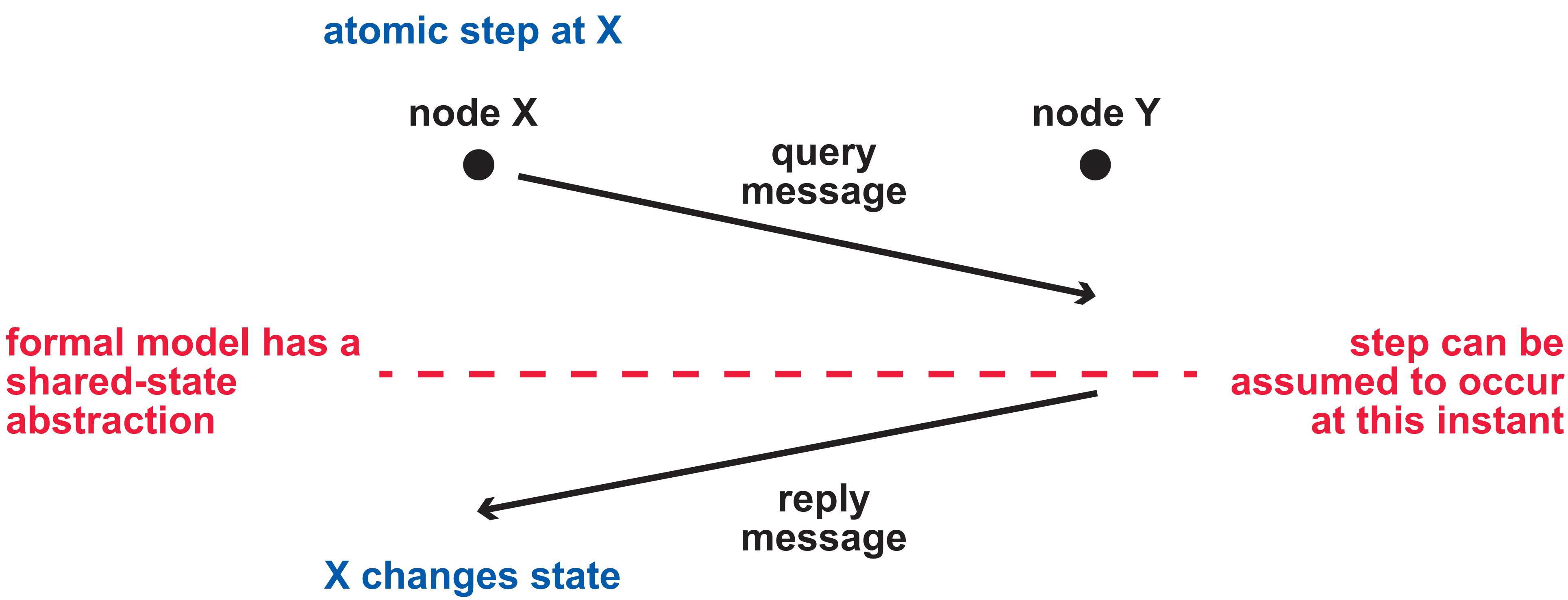


automatically satisfying *OneOrderedRing* and *Connected-Appendages*

CONCURRENCY AND COMMUNICATION

AN OPERATION IS A SEQUENCE OF ATOMIC STEPS

EACH ATOMIC STEP IS AN INTERNAL STATE CHANGE OR THIS:



while waiting for a reply (or timeout), X cannot answer queries about its state

because of the structure of operations, queries cannot form circular waits

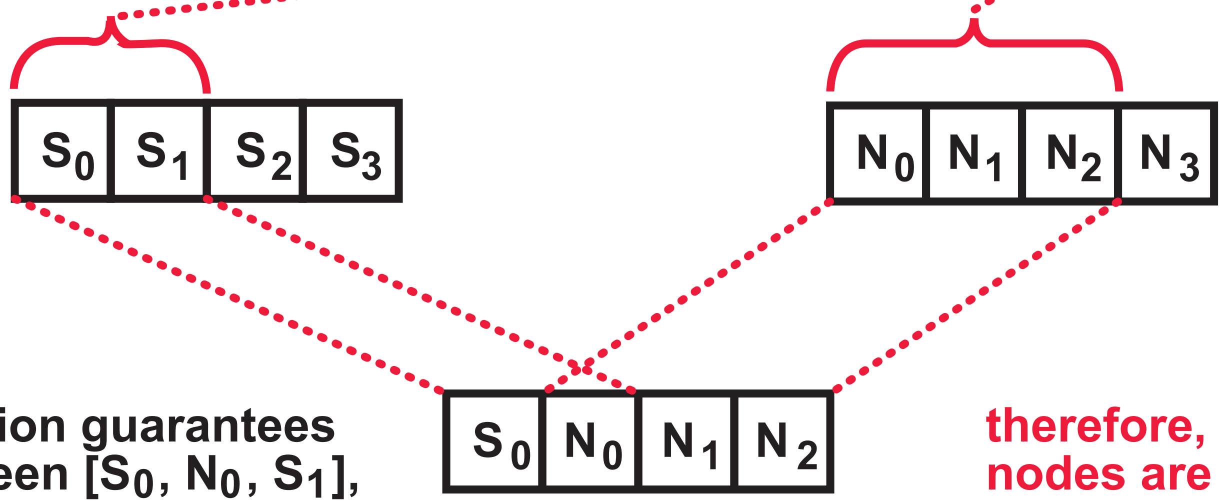
HOW STABILIZE PRESERVES THE INVARIANT

JOIN AND
RECTIFY
ARE SIMILAR

extended successor list of stabilizing node, before Stabilize

extended successor list of its new successor

because invariant holds, no current principal nodes are skipped here or here



precondition guarantees that between [S_0, N_0, S_1], so no current principal nodes are skipped from S_0 to N_0

therefore, no former principal nodes are skipped by this new successor list, and the number of principal nodes has not decreased

some Chord operations need multiple atomic steps

in the new, provably correct, specification, every intermediate operation state is also constructed in this safe way

HOW FAIL PRESERVES THE INVARIANT

PRESERVATION OF *OneLiveSuccessor*

The operating assumption is that no failure leaves a member with no live successor, . . .

. . . so the invariant is assumed to be preserved.

PRESERVATION OF *SufficientPrincipals*

Lemma: The only operation that can cause a node to change from principal to non-principal is its own failure.

Why can't failure of a principal node leave the network with fewer than L+1 principals?

The life history of a long-lived member:

1 Join
2 become principal because all neighbors know you
3 enjoy life as a principal node
4 Fail

Therefore the number of principal nodes is proportional to the number of nodes.

Once the network has grown (especially to millions of members!) it is overwhelmingly improbable that it will have fewer than 3-6 principal nodes.

PROVING PROGRESS

IF THERE ARE NO MORE
JOIN OR FAIL EVENTS . . .

. . . WHILE MEMBERS
CONTINUE TO STABILIZE . . .

1. dead successors are removed, so that every member's first successor is live

2. every member's first successor and predecessor become globally correct

3. tails of all successor lists become correct

as with construction of intermediate successor lists, operations must be specified precisely to ensure correctness

here preconditions must ensure that no operation reverses the progress of a past or current phase

CONCLUSIONS

THE PRODUCT

- initialization is more difficult than original Chord, but a simple protocol will get networks off to a safe start
- otherwise correct Chord is just as efficient as original Chord

it is an impressive pattern for fault-tolerance

- these peer-to-peer protocols have a (justified) reputation for unreliability
- a correct specification could pave the way for a new generation of reliable, more useful implementations

it also provides a firm foundation for work on better failure detection and security

THE PROCESS

- Chord is a very interesting protocol—note that the invariant looks nothing like the properties we care about!
- results would have been impossible to find without model-checking to explore bizarre cases and get ideas from them

that is where the idea of a stable base came from

- the best result was impossible to find without the insights that came from the proof process

```
pamelazave.com
> How to Make Chord Correct
```