HOW TO MAKE CHORD CORRECT

Pamela Zave

AT&T Laboratories—Research

Bedminster, New Jersey, USA

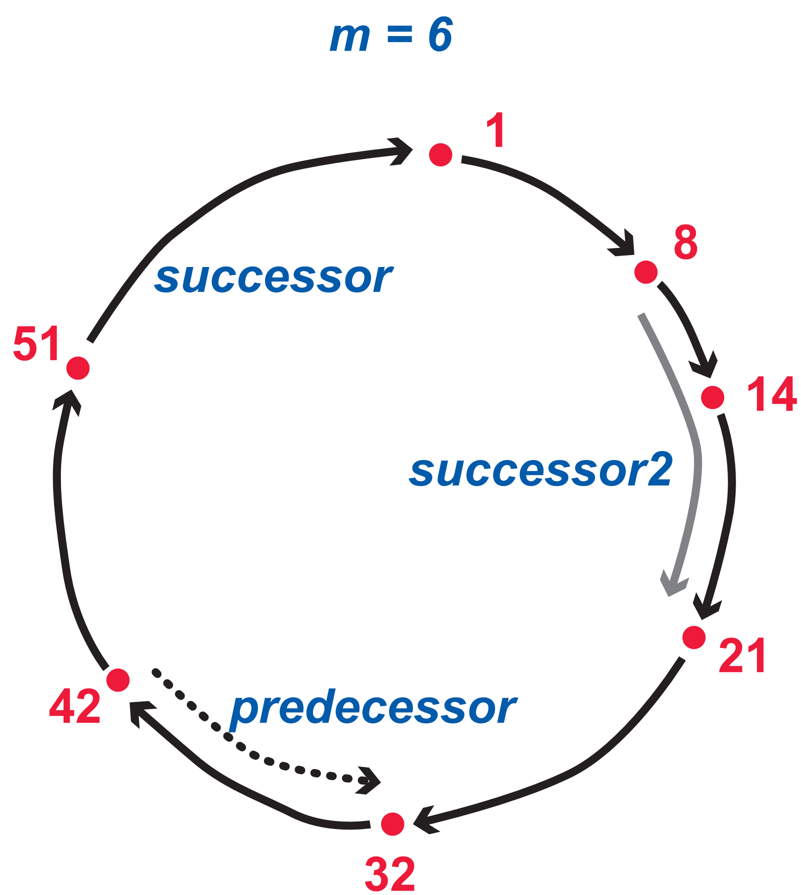
THE CHORD PROTOCOL MAINTAINS A PEER-TO-PEER NETWORK

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

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



THE PROTOCOL IS INTERESTING

- no central administration (almost)
- communication in the network is fast
- protocol operations are simple and fast:

  no timing constraints (almost)

  no multi-node atomic operations

WHY IS CHORD IMPORTANT?

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

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

. . . its simplicity,
. . . provable correctness,
. . . and provable performance."

APPLICATIONS

- allows millions of *ad hoc* peers to cooperate
- used as a building block in fault-tolerant applications
- often used to build distributed key-value stores (where the key space is the same as the Chord identifier space)
- the best-known application is BitTorrent

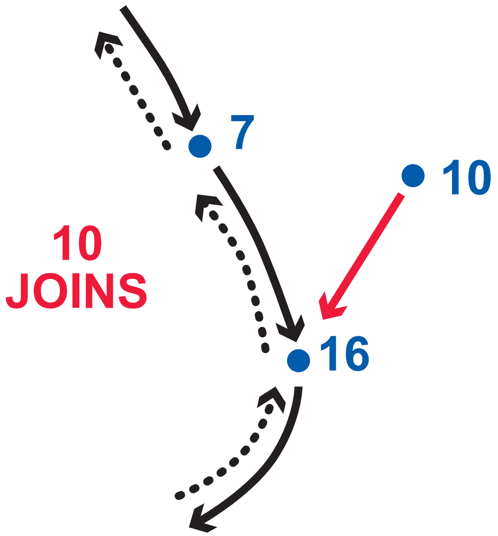
RESEARCH ON PROPERTIES AND EXTENSIONS

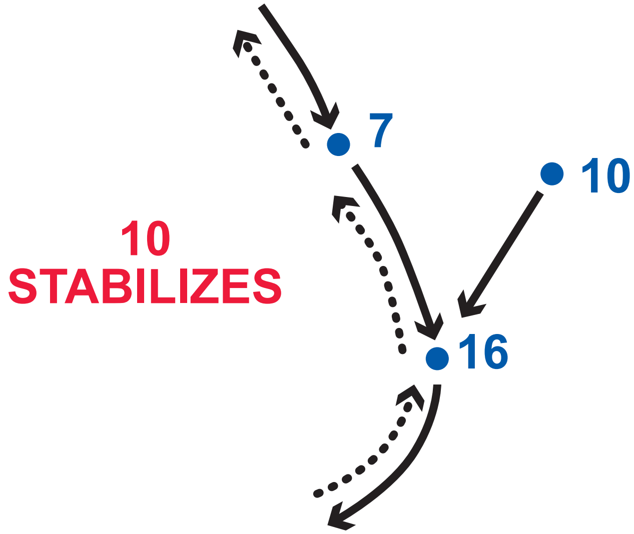
- protection against malicious peers
- key consistency (all nodes agree on which node owns which key)
- data replication and consistency of replicated data
- enhanced queries

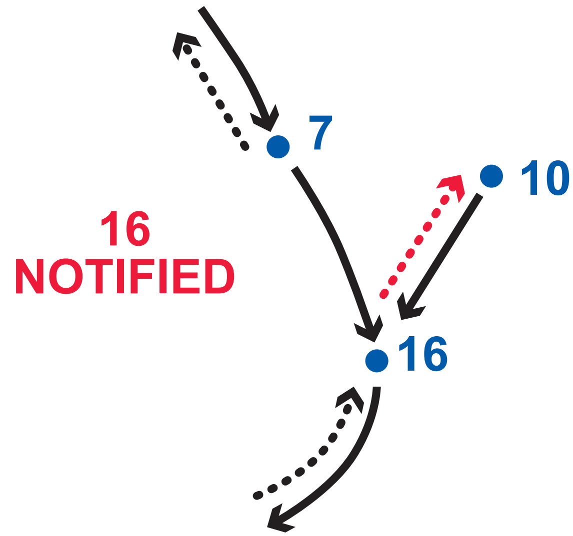
OPERATIONS OF THE PROTOCOL

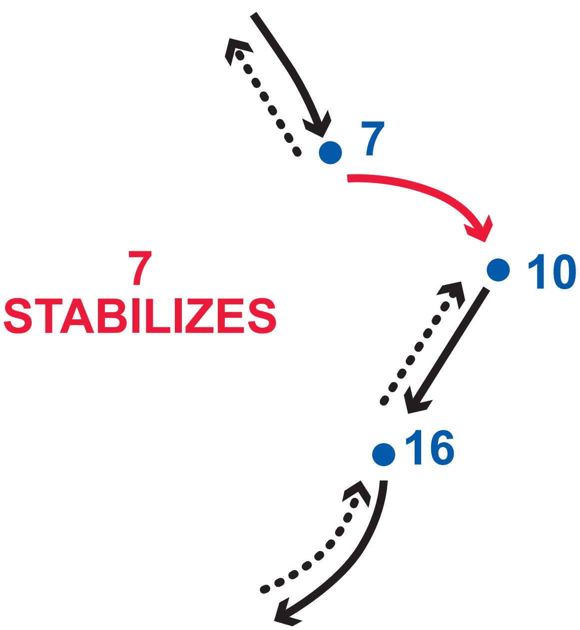
an operation changes
the state of one member

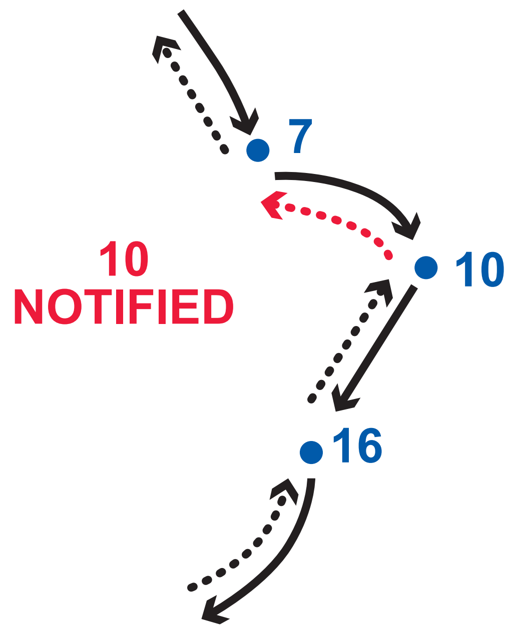
most operations are scheduled, asynchronously
and autonomously, by their own nodes





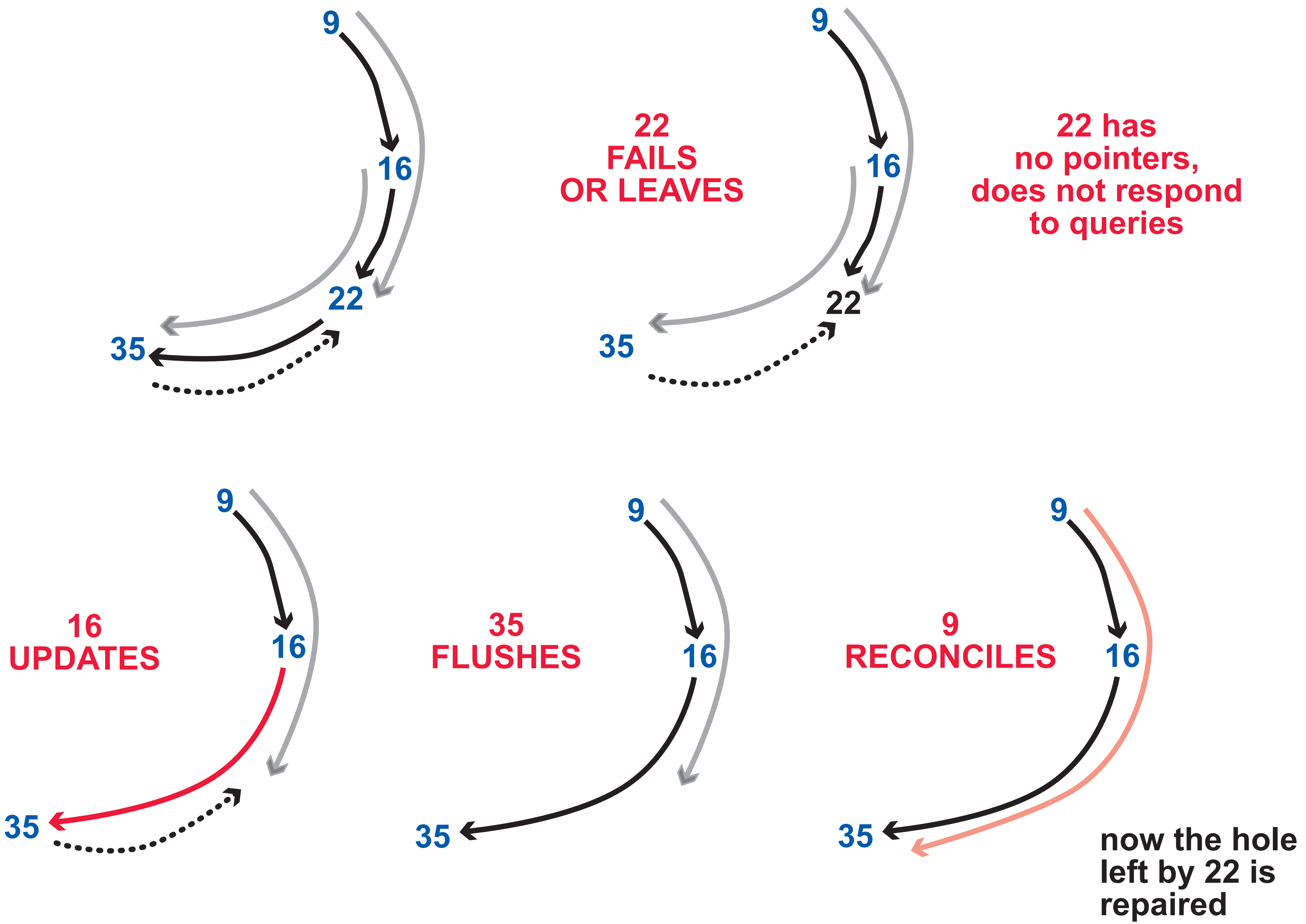






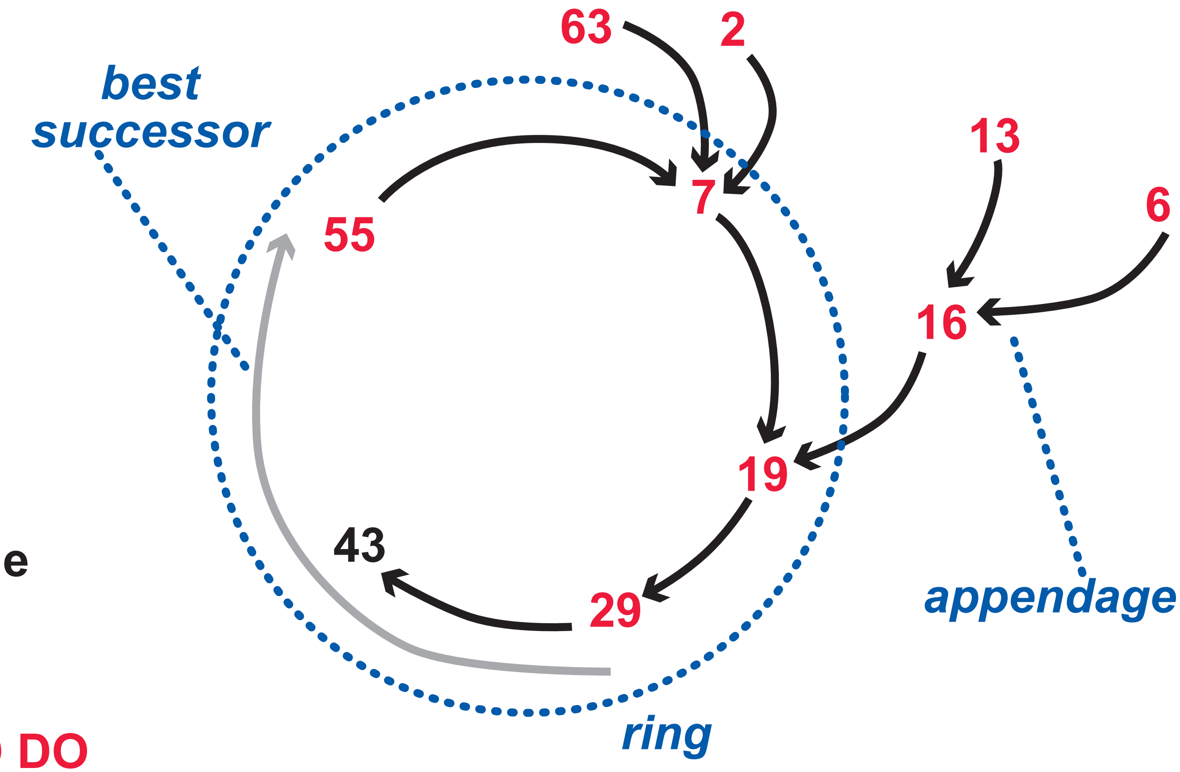
now 10 is fully
integrated into
the ring

MORE OPERATIONS OF THE PROTOCOL



IN A VALID NETWORK . . .

- there is a ring of best successors
- there is no more than one ring
- on the unique ring, the nodes are in identifier order
- from each appendage member, the ring is reachable through best successors



WHAT THE PROTOCOL SHOULD DO

Theorem:
A Chord network is always *Valid*.

Theorem:
In any execution state, if there are no subsequent join or fail events, eventually all pointers will become globally correct and remain globally correct.

"eventual consistency"

WHAT THE PROTOCOL CANNOT DO

Well-Known Fact:

If a Chord network is not *Valid*, then some member nodes are unreachable from some other nodes, . . .

. . . and this cannot be repaired.

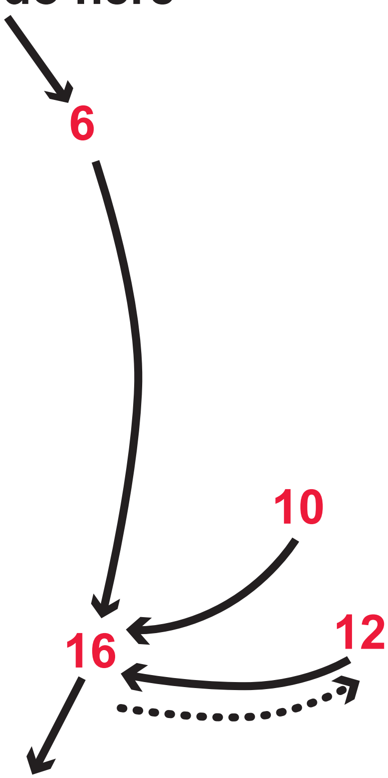
ORIGINAL CHORD IS NOT CORRECT—HOW BAD IS IT?

Because of sloppy, informal specification and proof . . .

. . . of the seven properties claimed to be invariants, *not one* is actually an invariant.

including the four clauses of *Valid*

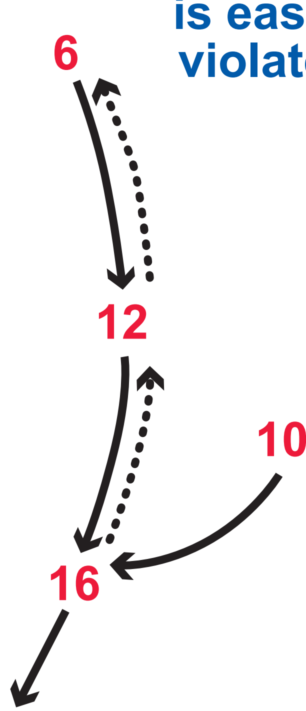
OrderedMerges . . .
. . . means that appendages merge in the correct places, as they do here



6 STABILIZES,

12 NOTIFIED

***OrderedMerges* is easily violated**



VIOLATIONS OF *OrderedMerges*

- are not incorrect
- cause some lookups to fail
- 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

OUTLINE

HOW TO MAKE CHORD CORRECT

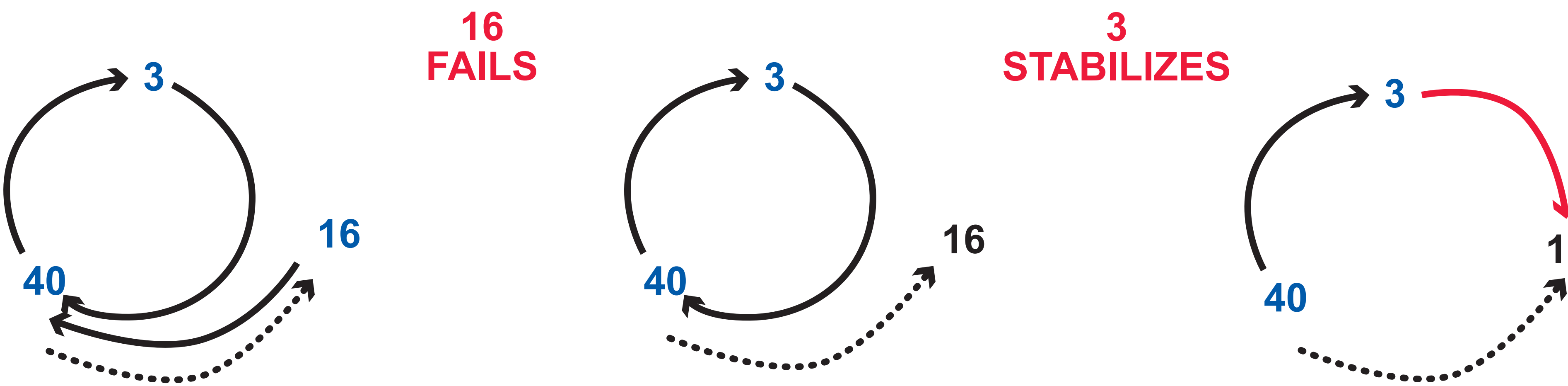
- fixing the protocol
- the elusive inductive invariant
- proof of correctness

THE ROLE OF FORMAL METHODS

COMMUNITY REACTIONS

BUGS IN THE OPERATIONS

A REALLY SIMPLE BUG:



Node 3 had no ***successor2*** yet
(it is not required to have all
successors filled in).

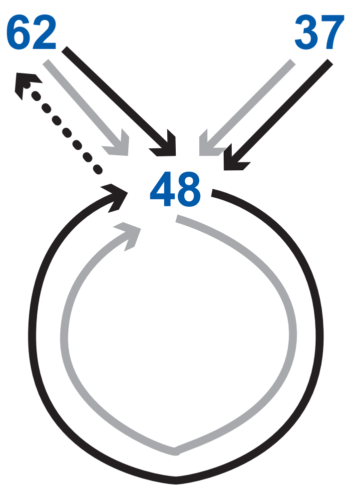
When it stabilized, it replaced a pointer
to a live node (40) with a pointer to a
dead node (16).

Now Node 3 has ***no pointers*** to live nodes,
and there is no ring.

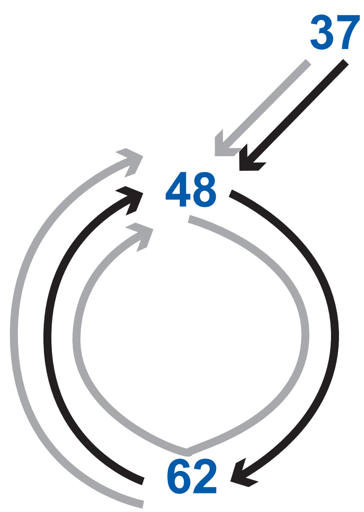
THE BUG IN INITIALIZATION

IN ORIGINAL CHORD, A NETWORK IS
INITIALIZED WITH ONE MEMBER

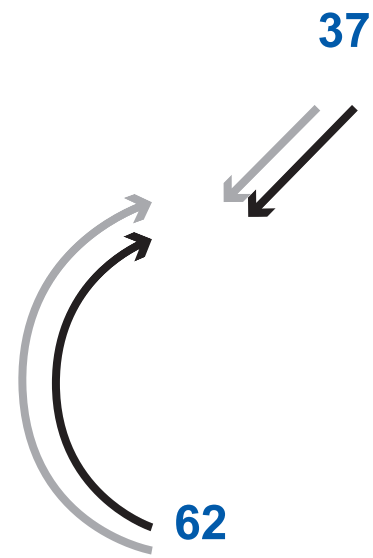
length of successor list
= R = 2



48
STABILIZES,
62
RECTIFIES



48
FAILS



first and second
successors must
duplicate each other

one failure
is allowed
by operating
assumptions . . .

. . . but the member
nodes, having only
one distinct entry in their
successor lists, cannot
recover from this failure

THE OPERATING ASSUMPTION

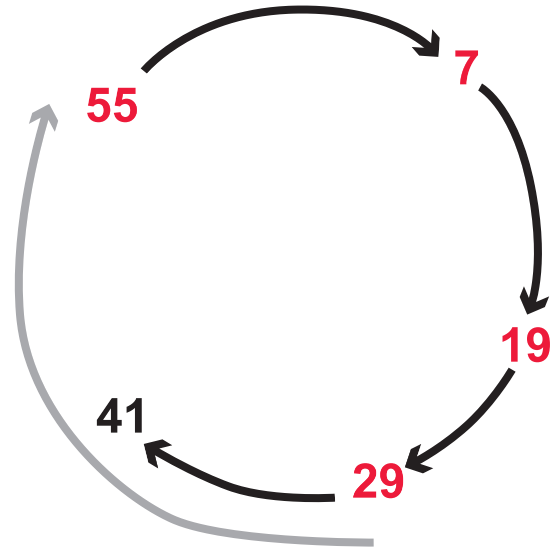
**extended successor list
of 29 (with R = 2):**

29	41	55

member itself

dead

best successor (first live successor)



Definition: **A Chord network has the property *FullSuccessorLists* if the extended successor list of each member has R+1 distinct entries.**

Operating Assumption (a kind of fairness):

> **If a Chord network has the property *FullSuccessorLists*, then every member has a best successor.**
>
> **ultimately, this relates the failure rate to the rate of stabilization**

FIXING ORIGINAL CHORD

FIXING THE OPERATIONS

- merge small, independently scheduled operations into bigger ones

  join + reconcile = JOIN

  stabilize + reconcile + update = STABILIZE

  notified + flush = RECTIFY

  this populates successor lists more eagerly, keeping them fuller

- check that a new pointer is live before replacing another pointer with it

- write complete, precise pseudocode

- be explicit about inter-node communication (queries)

FIX INITIALIZATION

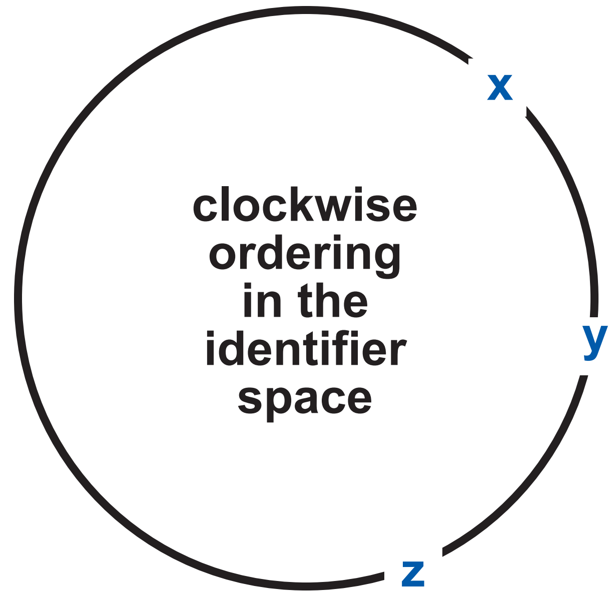
- alter the initialization to satisfy *FullSuccessorLists* with all successors live

  this requires a minimum of $R + 1$ members

WHAT IS THE INDUCTIVE INVARIANT?

NECESSARY BUT NOT SUFFICIENT:

- there is a ring of best successors
- there is no more than one ring
- on the unique ring, the members are in identifier order
- from each appendage member, the ring is reachable through best successors



Between [x, y, z]

! Between [z, y, x]

NOT INVARIANT (EVEN IN CONJUNCTION WITH THE ABOVE):

- *FullSuccessorLists*
- *OrderedSuccessorLists*
- and many, many other well-motivated candidates!

For all contiguous sublists *[x, y, z]* in an extended successor list, *Between [x, y, z]*.

WHY IS IT SO DIFFICULT?

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

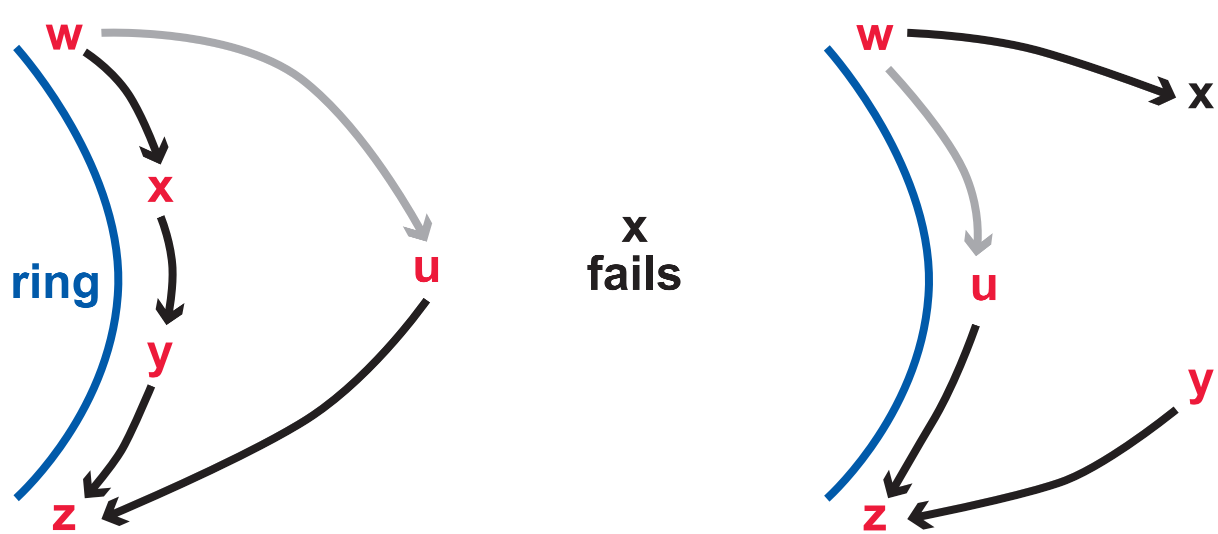
- there is a ring of best successors
- there is no more than one ring
- on the unique ring, the members are in identifier order

about the ring of best successors

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

about the appendages

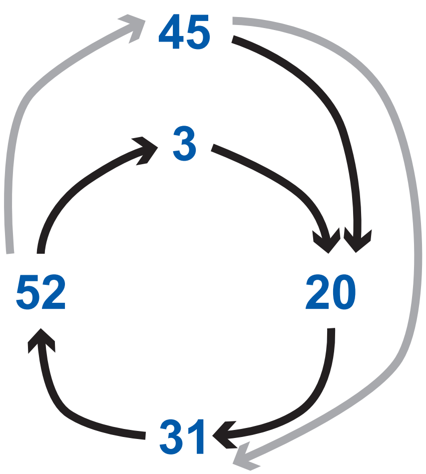
. . . BUT “RING VERSUS APPENDAGE” IS CONTEXT-DEPENDENT AND FLUID:



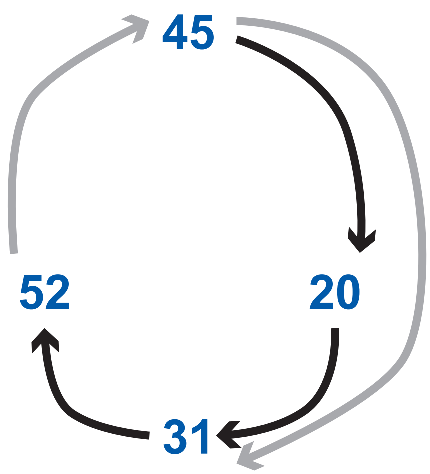
A TRIAL INDUCTIVE INVARIANT

Valid and	**EXCLUDES this extended successor list (R = 2):**
NoDuplicates and	[3, 4, 4] ·········· same first and second successors
OrderedSuccessorLists	[3, 7, 5] ·········· list is not in numerical order

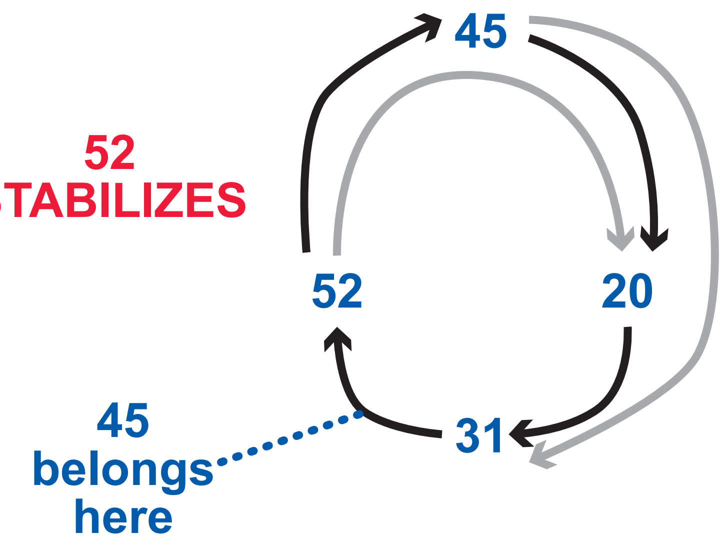
this network satisfies the property



3 FAILS



52 STABILIZES



now the identifiers in the ring are out of order, so the trial invariant is false

THE INDUCTIVE INVARIANT

Definition by example:
If 48 is a member, this extended successor list *skips* it.

45	47	55

ANOTHER OPERATING ASSUMPTION:

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

THE INDUCTIVE INVARIANT:

OneOrderedRing

and ConnectedAppendages

and BaseNotSkipped

no extended successor list skips a member of the stable base

WHY THE INDUCTIVE INVARIANT IS SUFFICIENT

ANOTHER OPERATING ASSUMPTION:

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

THE INDUCTIVE INVARIANT:

OneOrderedRing

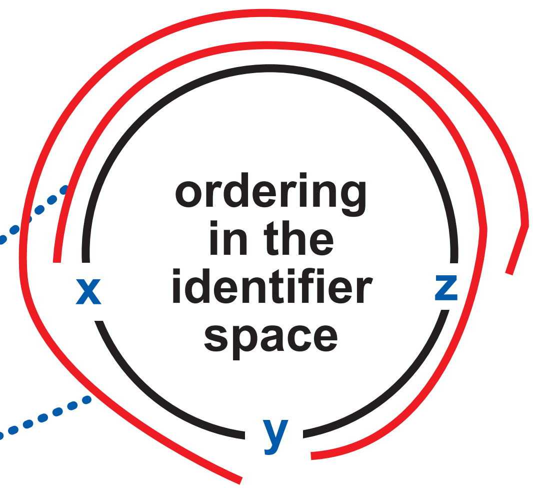
and ConnectedAppendages

and BaseNotSkipped

this applies to ring members and appendages alike

One case of a proof that this invariant implies *OrderedSuccessorLists*:

If *[x,y,z]* appears in a successor list and the values are as pictured, *OSL* is false.



But there can be no base member between *x* and *y* . . .

. . . and no base member between *y* and *z* . . .

. . . so *y* is the only possible base member, which is a contradiction.

SIGNIFICANCE OF THE BASE

WHAT DOES THE STABLE BASE DO, IN PRACTICE?

this matters because each member must be a high-availability node (cluster) with a fixed IP address

A stable base would have 3-6 members,

. . . while a Chord network can have millions of nodes, most of which are nowhere near a base node,

. . . so how can their operations be affected by the stable base?

CONCLUSIONS

- The purpose of the stable base is to prevent anomalies in small networks.

duplicate entries in successor lists, successor lists that "wrap around" and become disordered, *etc.*

- Once a Chord network *has grown to* a sufficient size, a stable base is not needed!

this does not mean that a large network cannot have flaws!

the large network must have grown from a small network satisfying the invariant

PROOF OUTLINE

THEOREM: **In any reachable state, if there are no subsequent joins or failures, then eventually the network will become ideal and remain ideal.**

PROOF:

1 **Show that the inductive invariant is true of all reachable states.** — **AUTOMATED (exhaustive search over a finite domain)**

2 **An operation that takes 0 or 1 query can be considered atomic. For operations that take 2 queries, show that the first half and the second half can safely be separated by concurrent operations.** — **AUTOMATED**

3 **An *effective* repair operation is one that changes the network state. Define a natural-valued measure of the error in the network, and show that every effective repair operation decreases the error.** — **MANUAL**

4 **Show that whenever the network is not *Ideal*, some effective repair operation is enabled.** — **AUTOMATED**
because the error is finite, after a finite number of repairs, the network will have no error and be Ideal

5 **Show that whenever the network is *Ideal*, no effective repair operation is enabled.** — **AUTOMATED**
once it is ideal it stays ideal, because repair operations will not change it

SMALL SCOPE HYPOTHESIS

NETWORK SIZE

We can only do exhaustive search for networks up to some node size N.

The “small scope hypothesis” makes explicit a folk theorem that most real bugs have small counterexamples.

Well-supported by experience, it is the philosophical basis of lightweight modeling and analysis.

RING STRUCTURES

The hypothesis is especially credible in this study, because ring structures are so symmetrical.

For example, to verify assertions relating pairs of nodes, it is only necessary to check rings of up to size 4.

not directly relevant to Chord

EXPLORATION OF CHORD MODELS CONFIRMS THIS

Original version of Chord was explored with R = 2.

new counterexamples were found at N = 2, 3, 4 (many of each), and 5 (just one)

Nearly-correct versions of Chord were explored with R = 2.

new counterexamples were found at N = 4, 5 (many of each), and 6 (just one)

Correct version of Chord was explored with R = 3.

no new counterexamples

WHAT SCOPE IS BIG ENOUGH?

I feel very safe having analyzed up to R = 3 and N = 9.

OUTLINE

HOW TO MAKE CHORD CORRECT

- fixing the protocol
- the elusive inductive invariant
- proof of correctness

THE ROLE OF FORMAL METHODS

COMMUNITY REACTIONS

LIGHTWEIGHT MODELING

DEFINITION

- constructing a small, abstract logical model of the key concepts of a system
- analyzing the properties of the model with a tool that performs exhaustive enumeration over a bounded domain

WHY IS IT "LIGHTWEIGHT"?

- because the model is very abstract in comparison to a real implementation, it is small and can be constructed quickly
- because the analysis tool is "push-button", it yields results with relatively little effort

in contrast,
theorem proving is not “push-button”

MY FAVORITE TOOLS

ALLOY

- Alloy language combines relational algebra, first-order predicate calculus, and transitive closure
- Alloy Analyzer compiles a bounded model into Boolean constraints, uses SAT solvers to decide whether the constraints are satisfiable

SPIN

- Promela is a simple programming language with concurrent processes, messages, bounded message queues, and fixed-size arrays
- Spin is a model-checker: the program specifies a large finite-state machine that the checker explores exhaustively

WHAT IS EASY

It is easy to model Chord in either Alloy or Promela.

It is easy to . . .

- find bugs
- check assertions
- get counterexamples to assertions

. . . in networks with up to 4 nodes.

with R = 2, this covers a lot of ground

WHAT IS VERY HARD

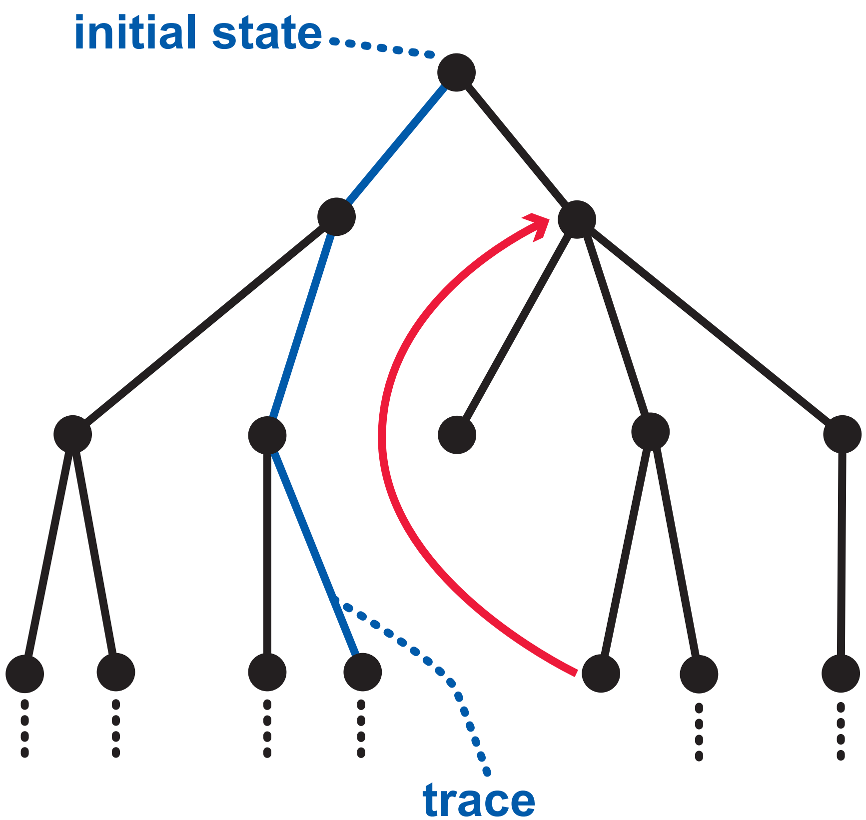
To prove that Chord is correct, we need an inductive invariant (*Valid* is not strong enough).

We don't know if there is an inductive invariant, or if there is one, what it is.

Without an inductive invariant, the value of analysis is limited.

There are many trial invariants, some straightforward and some baroque, and none of them seem to work.

SPIN



all the states explored are true, reachable states

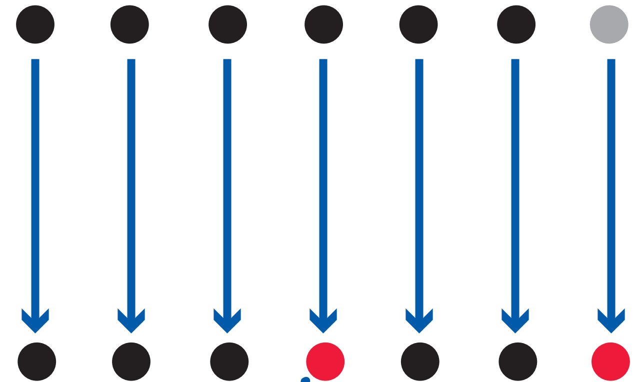
traces can be long, even infinite

HOWEVER, only networks with 4-5 nodes can be analyzed completely, and this is not enough

ALLOY

here are the states that satisfy a trial inductive invariant P:

Alloy computes the results of one operation



this state does not satisfy P

maybe the new state is fine—

if so, make P weaker to accept it and try again

this state does not satisfy P

maybe the old state could never be reached in a real network—

if so, make P stronger to reject the old state and try again

this process might go on for a long time and NEVER CONVERGE

“REAL” VERIFICATION VERSUS BOUNDED ANALYSIS

“Why don’t you do a real proof (for arbitrary R and N)?”

ANSWER # 1

- I don’t think engineers will do “real” proofs with theorem provers.
- I think that engineers can and should do lightweight modeling and analysis, and I am trying to persuade them of its value.

ANSWER # 2

- I don’t know how (but I’m trying).
- A real proof would be very illuminating!

Chord is interesting to
two subfields of computer science

THE REACTION: NETWORKING

. . . most of these problems were easy to find with a small model and the Alloy Analyzer,

. . . so there is an important lesson about lightweight formal modeling as a design tool,

. . . but the networking community rejects papers on fixing Chord

THE REACTION: DISTRIBUTED SYSTEMS

- Amazon Web Services credits this work with overcoming their bias against formal methods, and causing them to start using formal methods to find bugs
- AWS is now using TLA, with great success

REFERENCES

CHORD WORK

"Using lightweight modeling to understand Chord"

ACM SIGCOMM Computer Communications Review, April 2012

"A practical comparison of Alloy and Spin"

Formal Aspects of Computing, 2014

"How to make Chord correct"

arXiv, submitted for publication

AMAZON

"How Amazon Web Services uses formal methods"
Chris Newcombe *et al.*
Communications of the ACM, April 2015

ALL PAPERS AND MODELS

`www.research.att.com/~pamela > Chord`

COMMUNITY REACTIONS

DISTRIBUTED-COMPUTING COMMUNITY IS EMBRACING FORMAL METHODS

Amazon credits this work with giving them the required evidence that formal methods work on real-world systems; Amazon now uses formal methods routinely.

SIGCOMM COMMUNITY IS CREATIVE IN REJECTING FORMAL METHODS

"these flaws are obvious and implementers fix them"

- **this is patently false**
- **even if it were true, why should each implementer have to rediscover them?**

"these flaws are improbable and do not occur in real executions"

how can anyone know that?

- **computing execution probabilities would require implementation-specific information such a timing**
- **there are Chord failures (hearsay), and no one knows the cause of all of them**

note that Chord could use more capabilities and stronger properties, so people build on the basic protocol

- **extensions should not be verified on an unsound foundation**

note that people do not really understand how Chord works (incorrect invariants and performance analyses)

- **the modeling, analysis, invariant, and proof all contribute to insight**
- **dynamically checking the invariant is a good security principle, but cannot be done if the invariant is not known**