HOW TO MAKE CHORD CORRECT

(WITH A SURPRISING INVARIANT)

Pamela Zave

AT&T Laboratories

Bedminster, New Jersey

Princeton University

Princeton, New Jersey

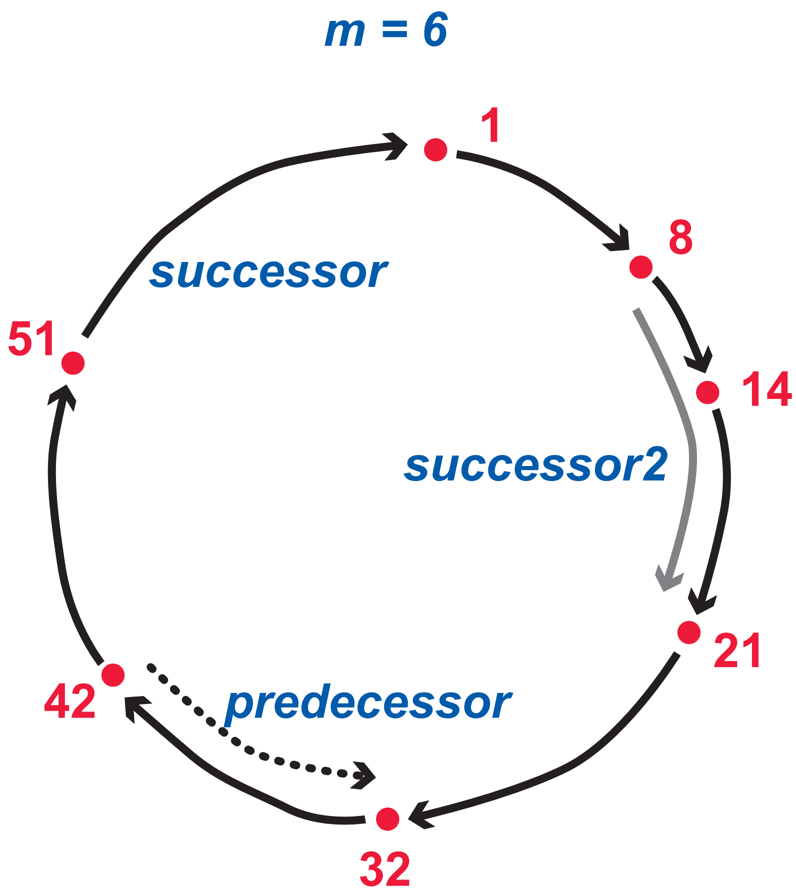
THE CHORD PROTOCOL MAINTAINS A PEER-TO-PEER NETWORK

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

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

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), replicated data consistency

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

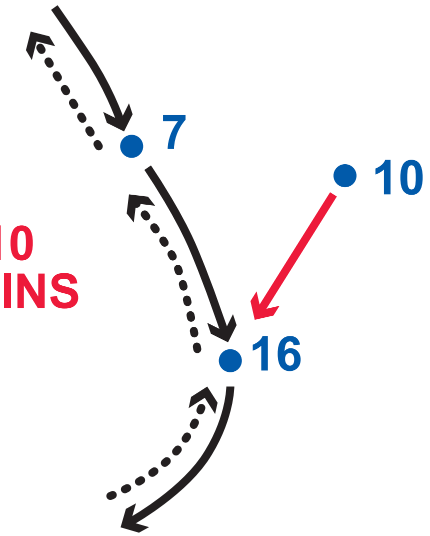
. . . its simplicity,
. . . provable correctness,
. . . and provable performance."

OPERATIONS OF THE PROTOCOL

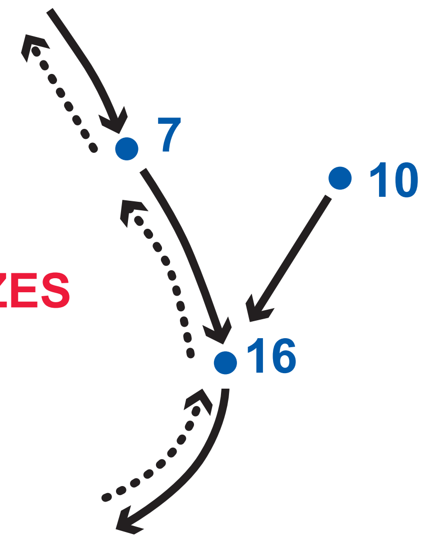
an operation changes the state of one member

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

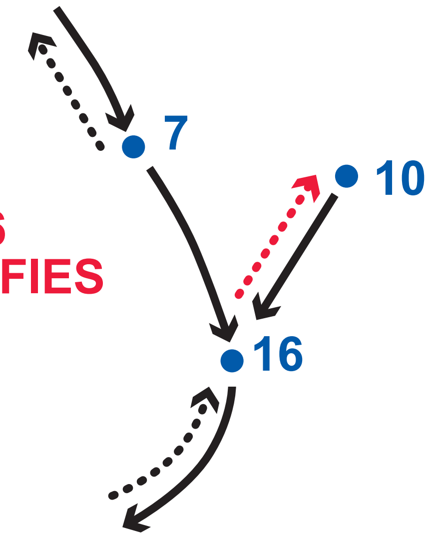
10 JOINS



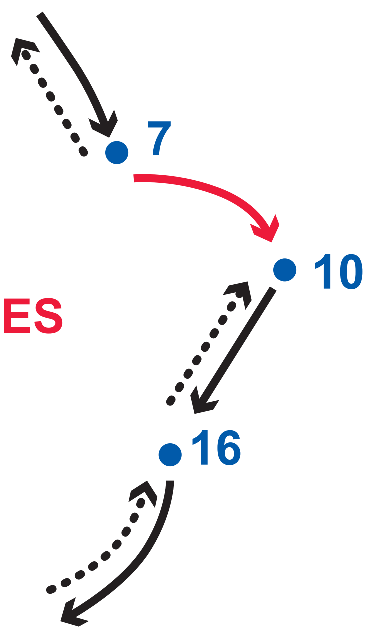
10 STABILIZES



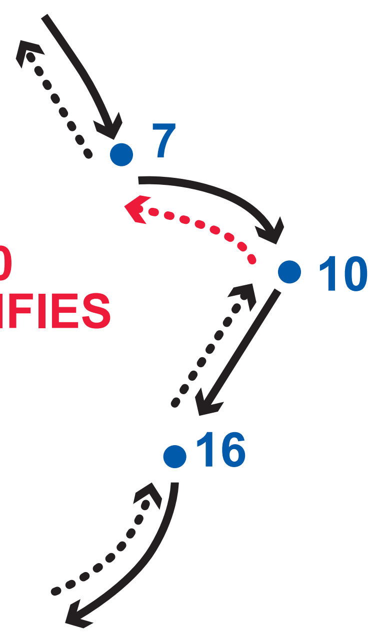
16 RECTIFIES



7 STABILIZES



10 RECTIFIES



in addition, a member can Fail (or leave) silently

there is perfect failure detection

Stabilizing member detects a dead successor and promotes its next successor

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.

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

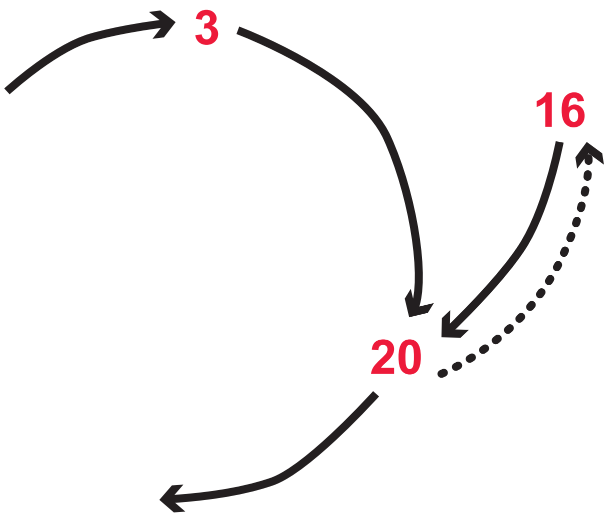
not surprisingly, due to sloppy informal specification and proof

I found these problems by analyzing a small Alloy model

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

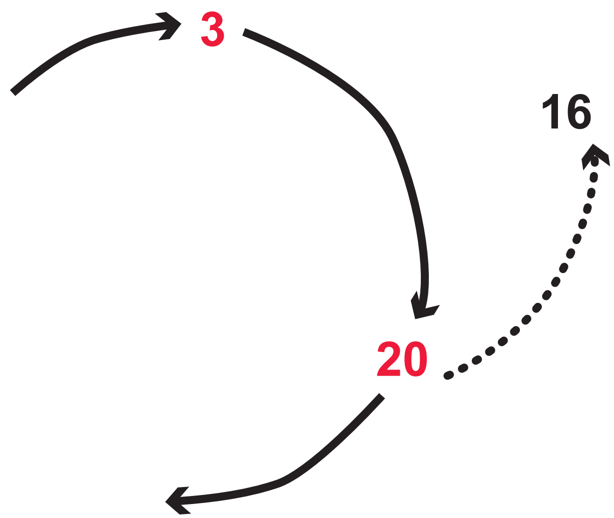
[CACM, April 2015]

A TYPICAL BUG IN ORIGINAL CHORD

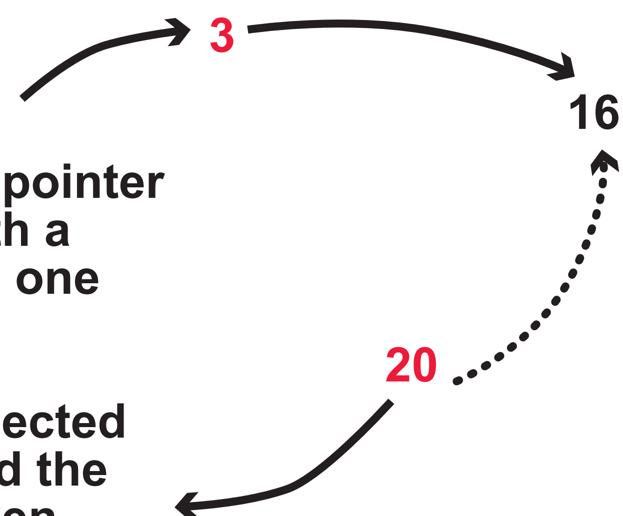


3 has no *successor2* yet (it is not required to have all successors filled in)

16 FAILS



3 STABILIZES

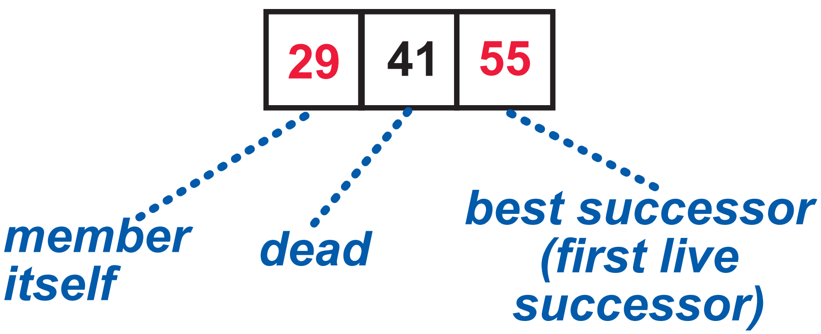


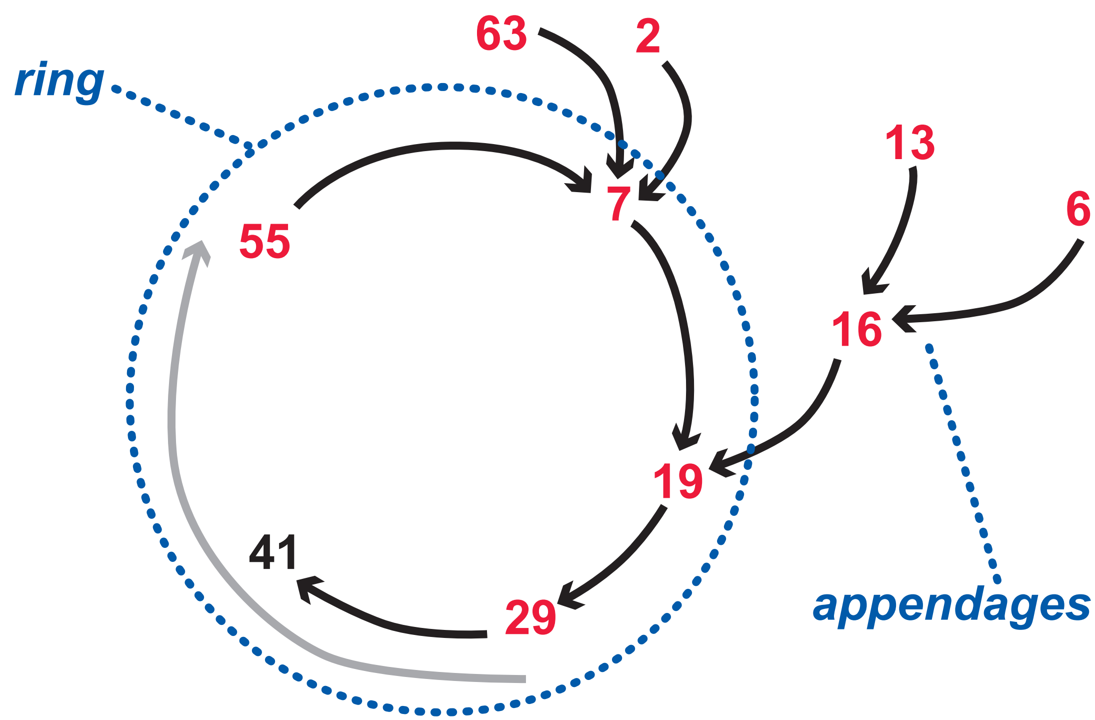
3 has replaced a pointer to a live node with a pointer to a dead one

now 3 is disconnected from the ring, and the ring may be broken

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.

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

29	32	29

. . . then 32 cannot fail!

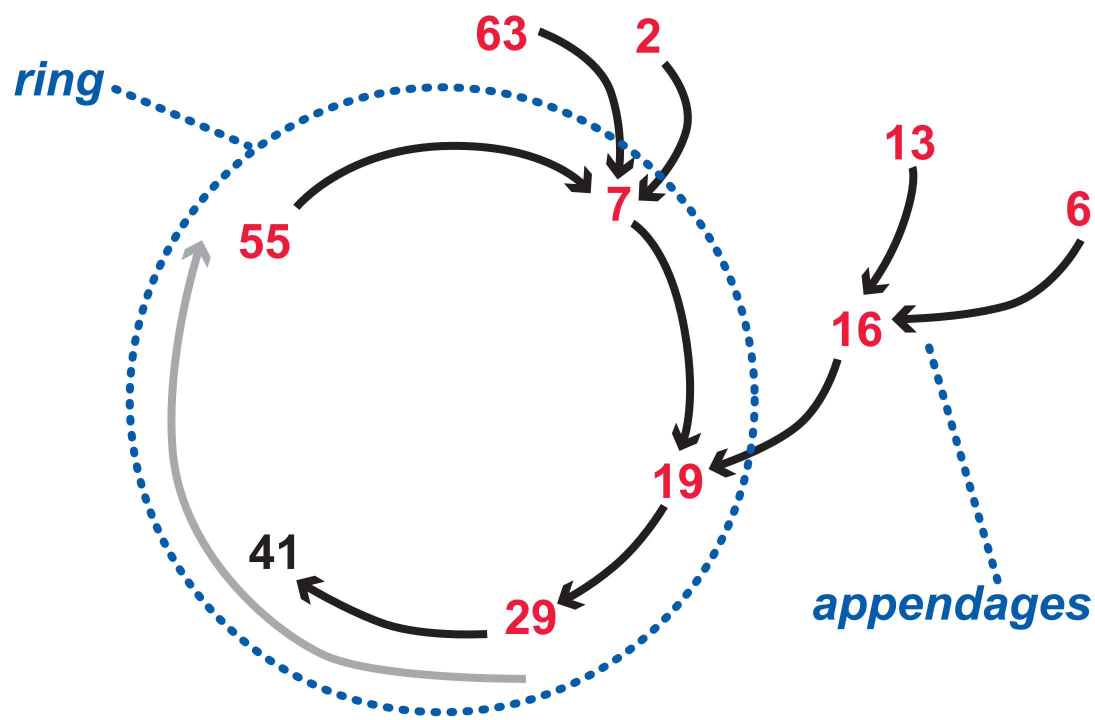
Definition of *FullSuccessorLists:*
The extended successor list of each member has L+1 distinct entries.

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 failure rate, increase rate of stabilization or increase redundancy

BASIC CORRECTNESS STRATEGY 2



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, so that they always have L entries

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

WHY IS FINDING AN INVARIANT SO DIFFICULT?

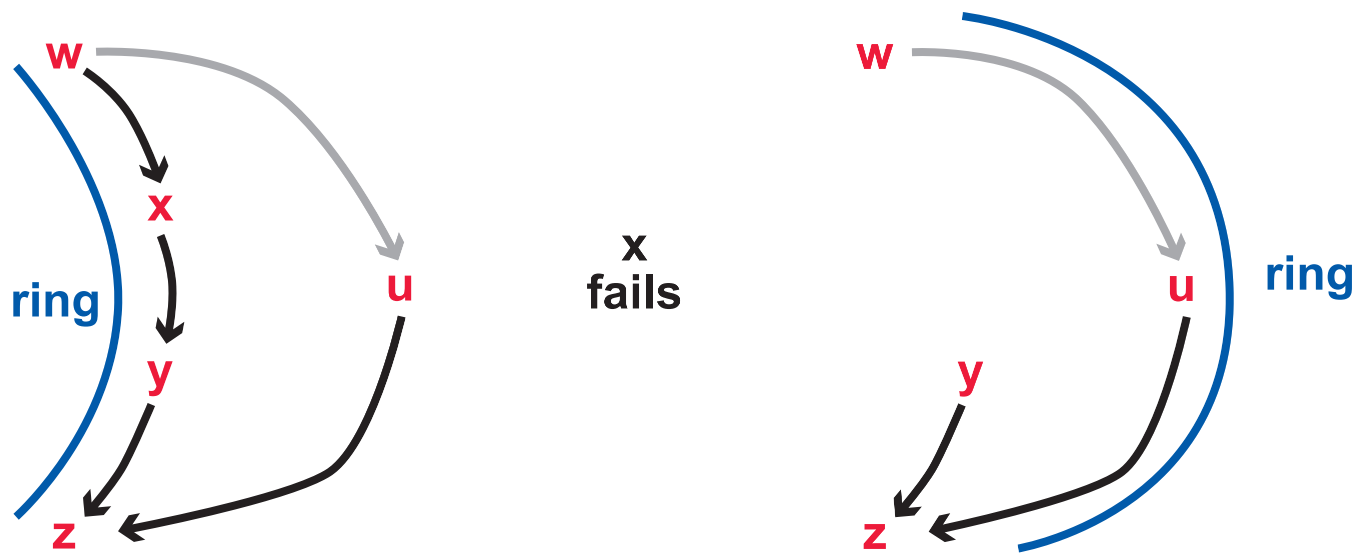
THE KNOWN, NECESSARY PROPERTIES ARE STATED IN TERMS OF THE RING . . .

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

about the ring of best successors (first three properties)

about the appendages (fourth property)

. . . 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

THE INDUCTIVE INVARIANT:

this is just a formalization of the original operating assumption

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)

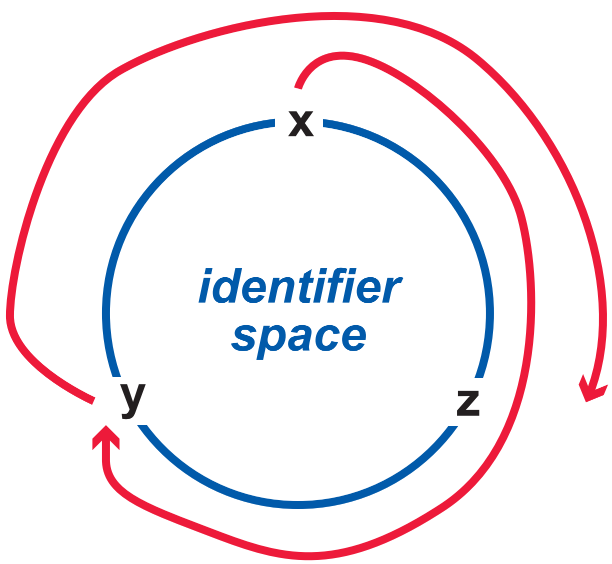
. . . backed up by an Alloy model checked up to N = 9, L = 3 (as a protection against human error)

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

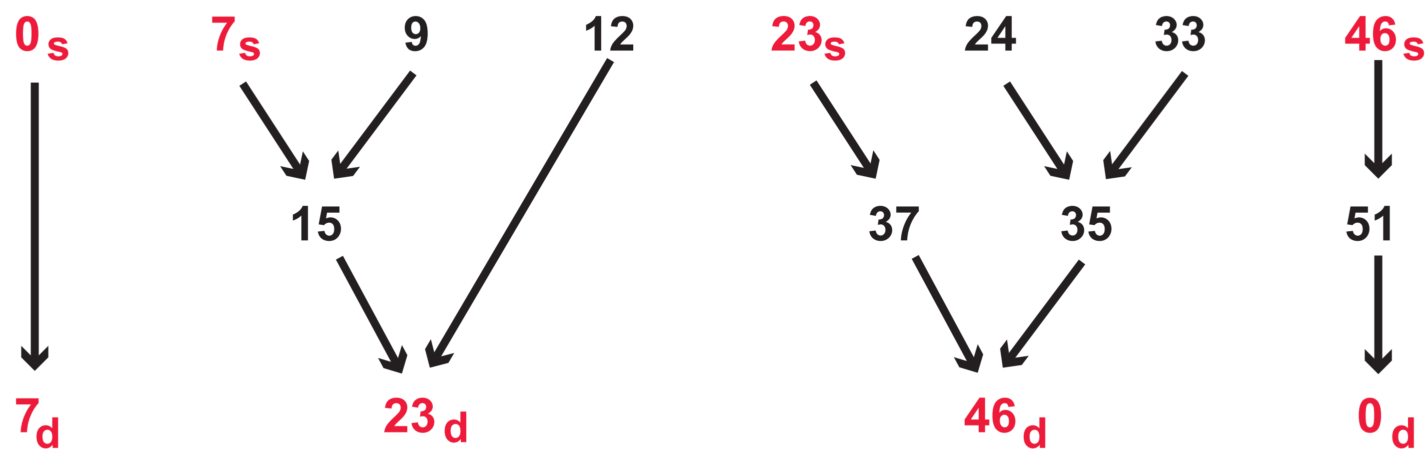
here is a graph of best successors:

these paths . . .

. . . do not skip principal nodes

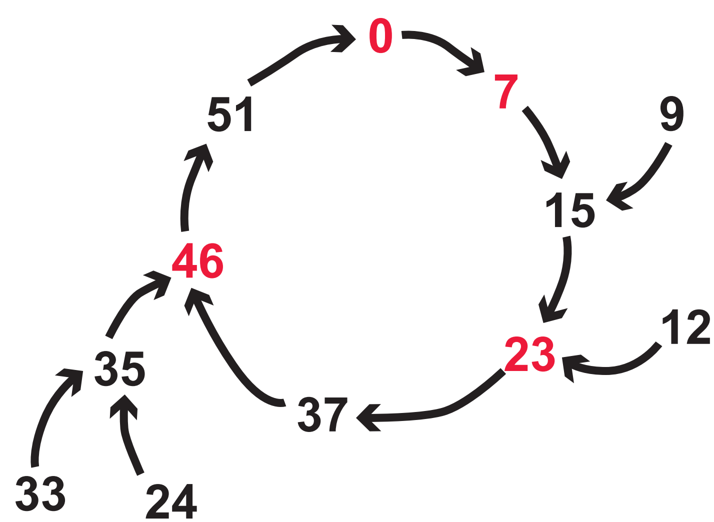
. . . are acyclic

. . . 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

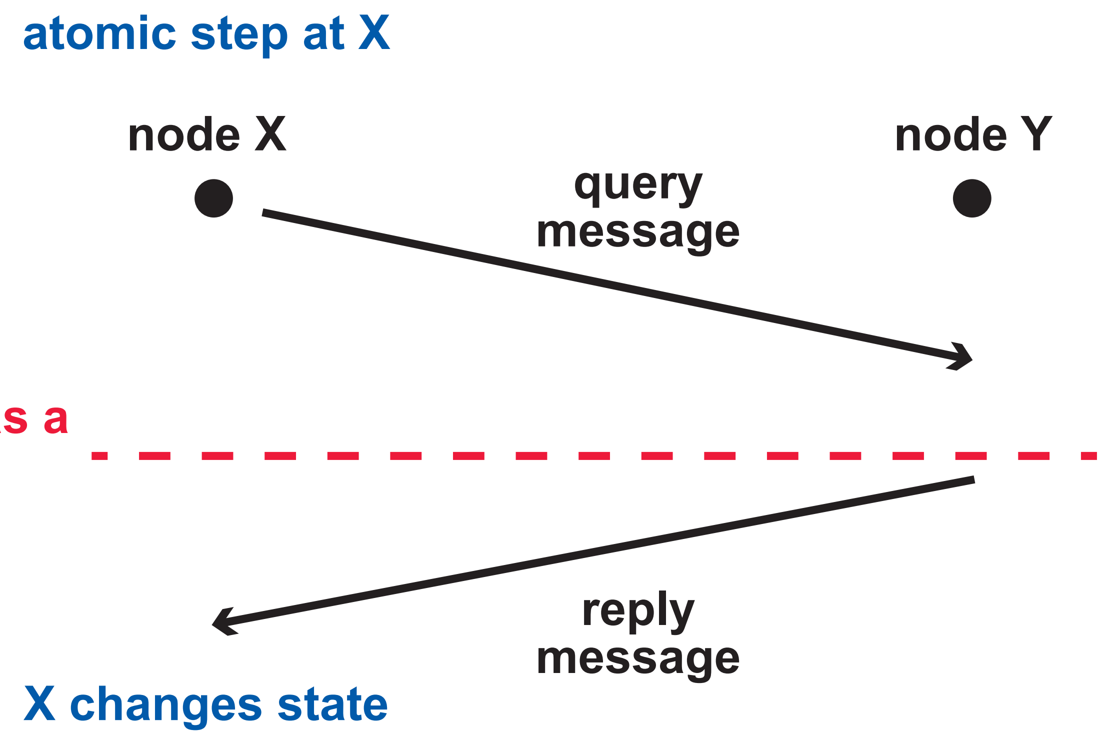


automatically satisfying *OneOrderedRing* and *Connected-Appendages*

CONCURRENCY AND COMMUNICATION

AN OPERATION IS A SEQUENCE OF ATOMIC STEPS

EACH ATOMIC STEP IS AN INTERNAL STATE CHANGE OR THIS:



formal model has a shared-state abstraction

step can be assumed to occur at this instant

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

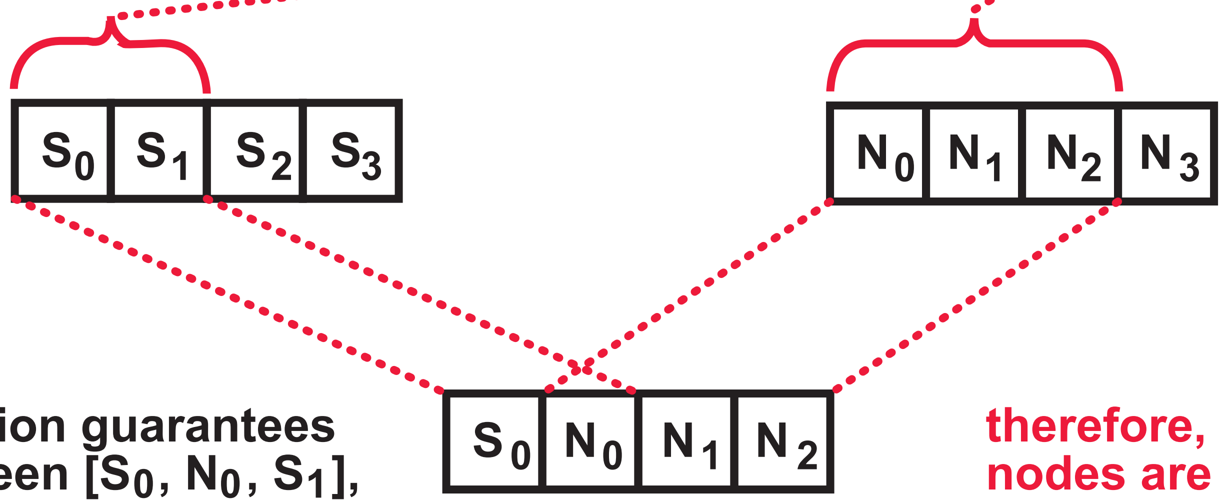
HOW STABILIZE PRESERVES THE INVARIANT

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

`www.research.att.com/~pamela`
`> How to Make Chord Correct`