FORMAL METHODS IN NETWORKING

COMPUTER SCIENCE 598D, SPRING 2010

PRINCETON UNIVERSITY

LIGHTWEIGHT MODELING

IN PROMELA/SPIN AND ALLOY

Pamela Zave

AT&T Laboratories—Research

Florham Park, New Jersey, USA

THE PRESS RELEASE

"Three features that distinguish Chord from many peer-to-peer lookup protocols are its simplicity, provable correctness, and provable performance."

THE (NEWLY DISCOVERED) REALITY

- the only "proof" covers the join-and-stabilize case only, with no failures
- this "proof" is an informal construction of ill-defined terms, unstated assumptions, and unjustified or incomprehensible steps

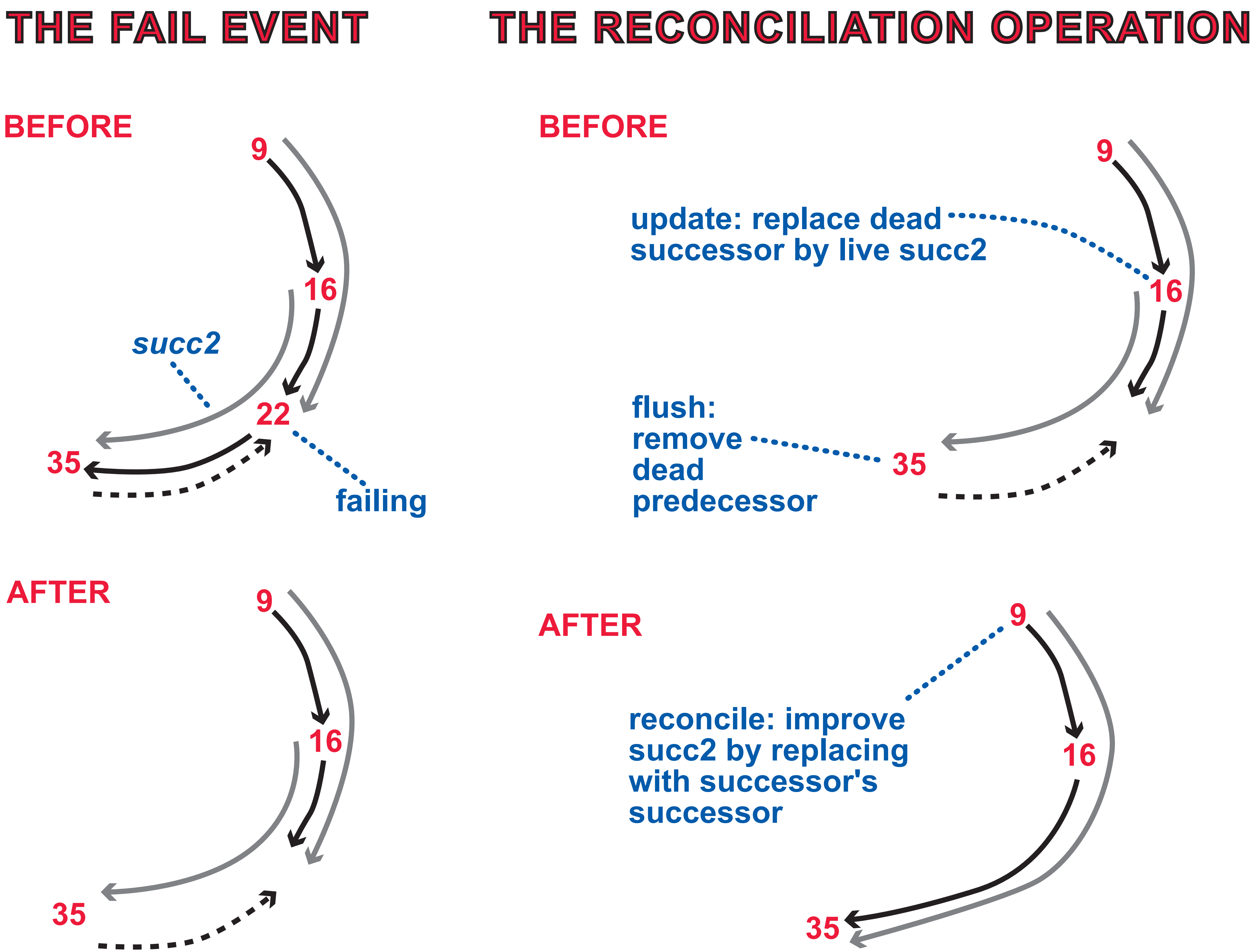
however, the subset can be proven correct, formally

- the full protocol is incorrect, even after bugs with straightforward fixes are eliminated
- *not one* of the six properties claimed invariant for the full protocol is invariantly true
- some of the many papers analyzing Chord performance are based on false assumptions about how the protocol works

USE

LIGHTWEIGHT MODELING

and avoid embarrassment!


THE FAIL EVENT
THE RECONCILIATION OPERATION
BEFORE
9
16
succ2
22
35
failing
BEFORE
9
update: replace dead successor by live succ2
16
flush: remove dead predecessor
35
AFTER
9
16
35
AFTER
9
reconcile: improve succ2 by replacing with successor's successor
16
35

ANTECEDENT PREDECESSORS

```
pred AntecedentPredecessors [t: Time] {
   all n: Node | let antes = (succ.t).n |
      n.prdc.t in antes
}
```

at time t, the set of all nodes whose successor is n

WHERE DID IT COME FROM?

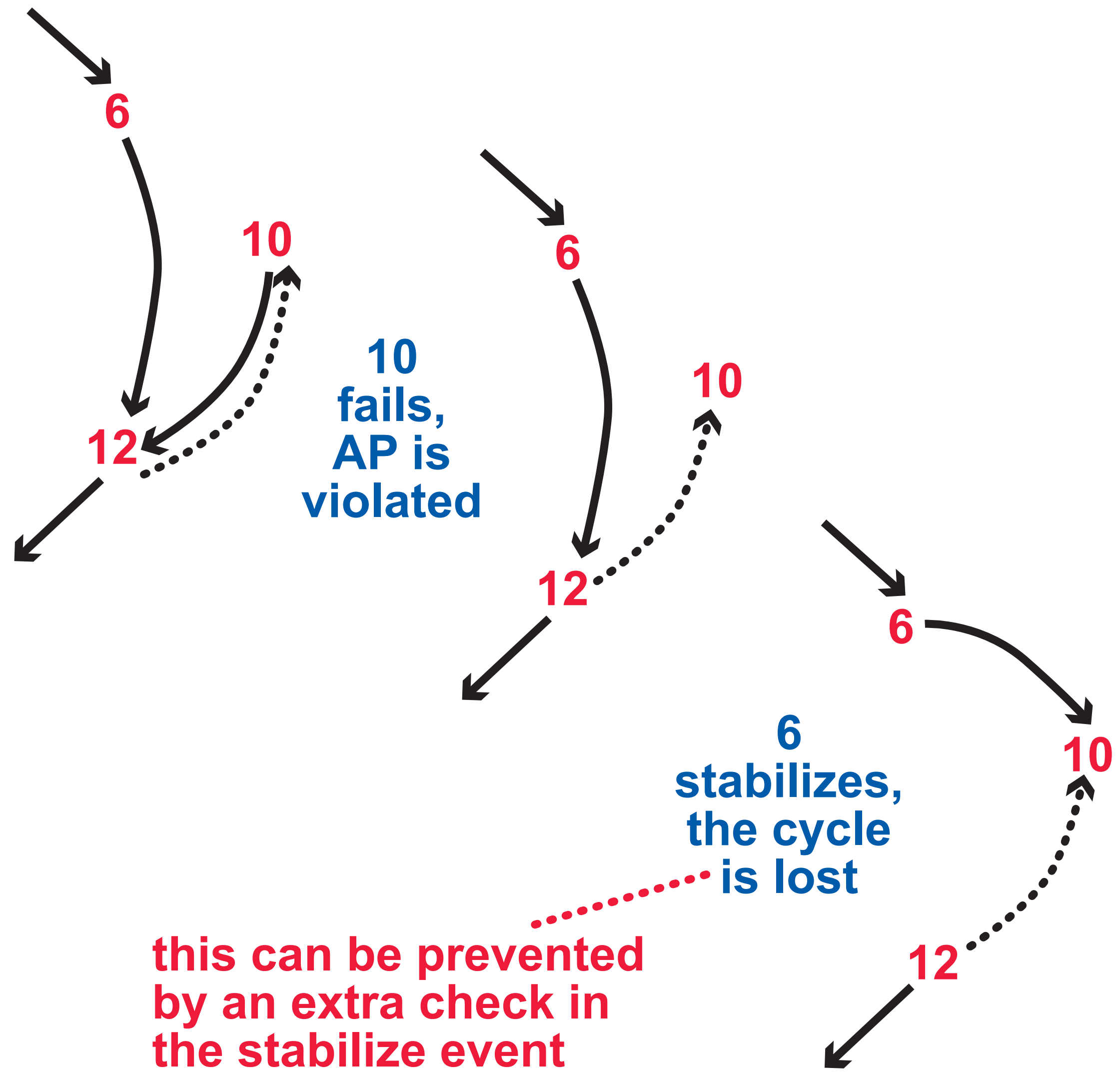
must be an invariant to prove that the pure-join model is correct

WAS IT PREVIOUSLY KNOWN?

no, supporting my allegation that the previous "proof" is useless

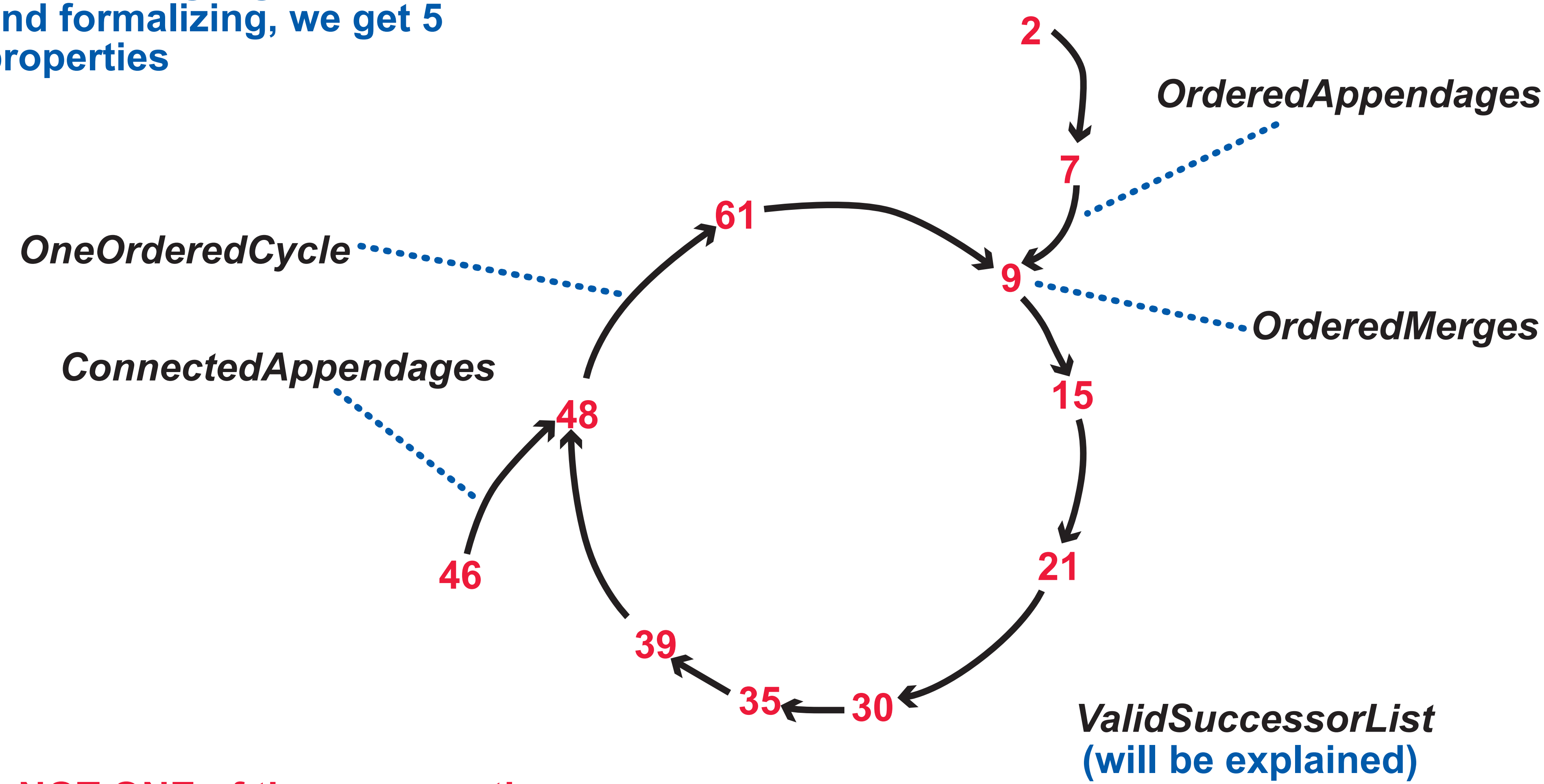
IS IT GOOD FOR ANYTHING ELSE?

yes, it enables us to diagnose and fix a Chord bug



PROPERTIES CLAIMED INVARIANT FOR THE FULL MODEL

after untangling bad definitions and formalizing, we get 5 properties



NOT ONE of these properties is actually an invariant!

ORDERED MERGES

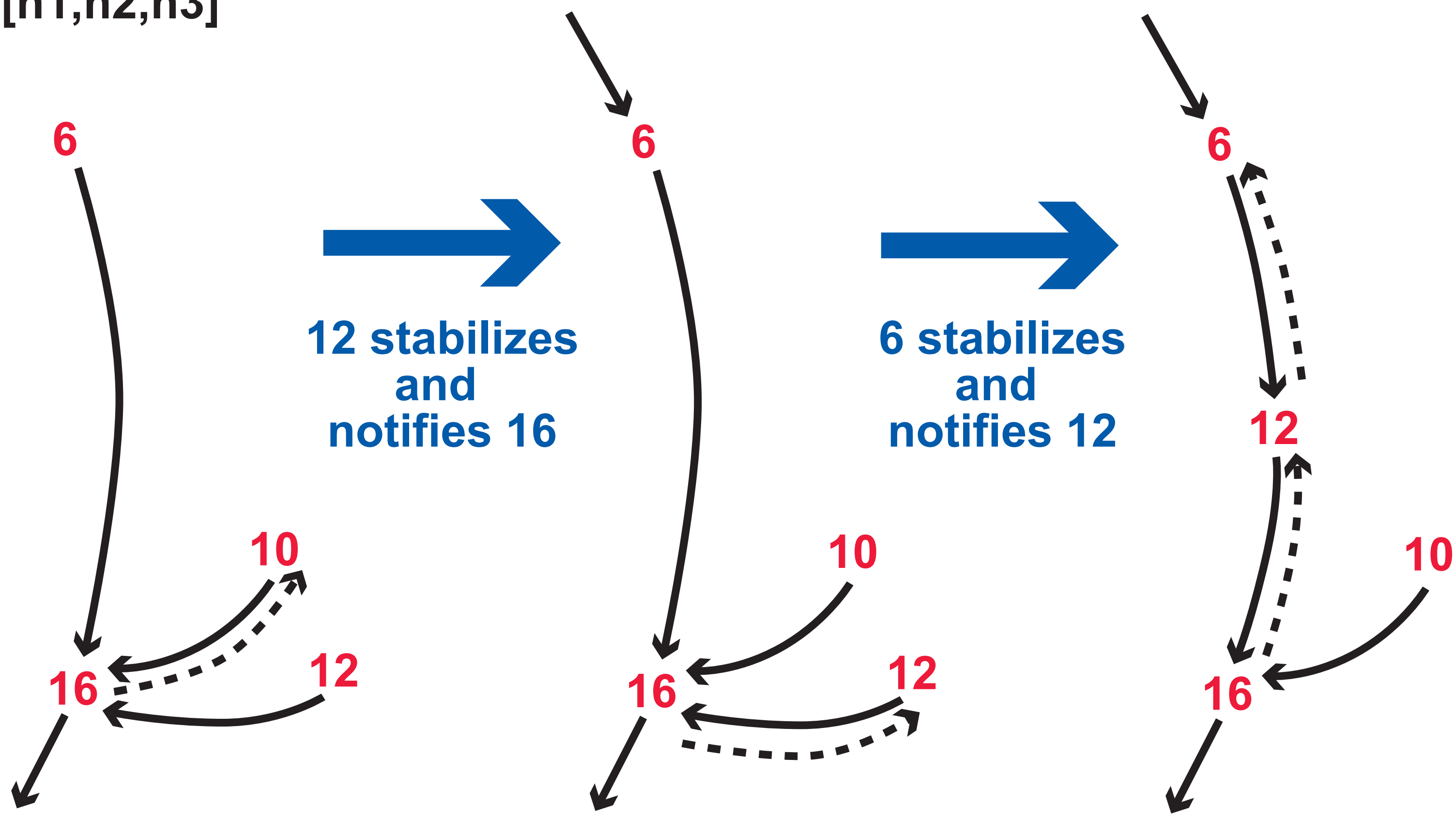
best live successor

```
pred OrderedMerges [t: Time] {
   let cycleMembers =
      { n: Node | n in n.(^(bestSucc.t)) } |
      all disj n1, n2, n3: Node |
         (  n3 in n1.bestSucc.t
         && n3 in n2.bestSucc.t
         && n1 in cycleMembers
         && n2 !in cycleMembers
         && n3 in cycleMembers
         ) => Between[n1,n2,n3]
}
```

The good news:
Violations are repaired by stabilization.

The bad news:
Compromises some lookups.
Invalidates some assumptions used in performance analysis.

easily violated, even in the pure-join model



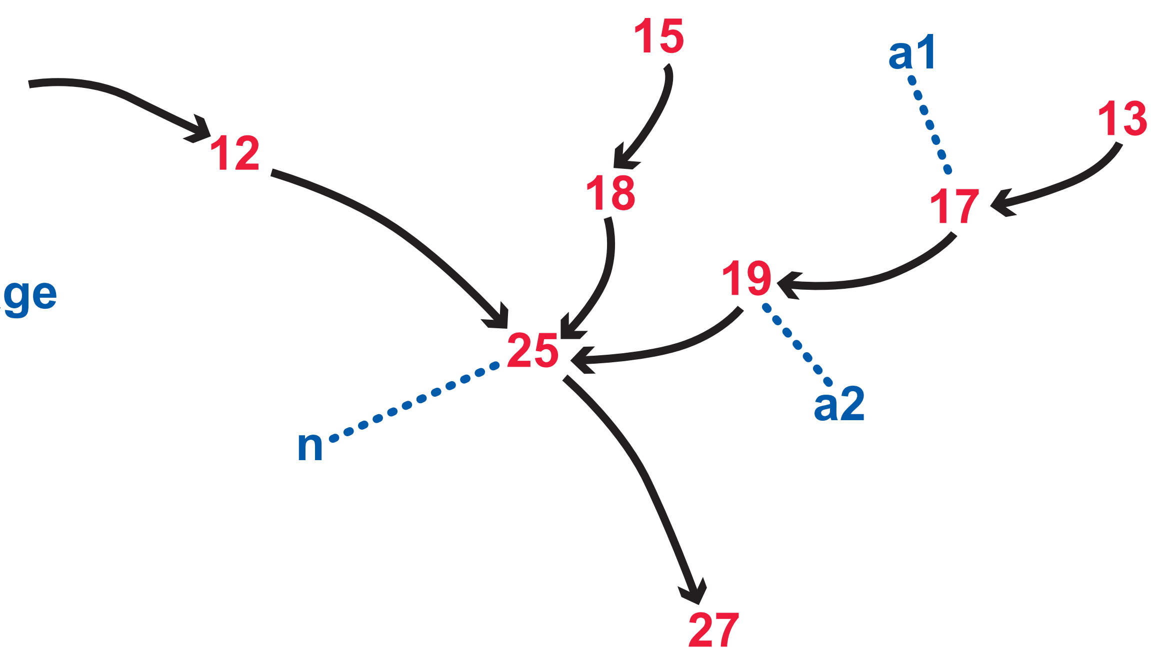
ORDERED APPENDAGES

WHY A POWERFUL ASSERTION LANGUAGE IS NEEDED

```
pred OrderedAppendages [t: Time] {
   let members = { n: Node | Member[n,t] } |
   let cycleMembers = { n: members | n in n.(^(bestSucc.t)) } |
   let appendSucc = bestSucc.t - (cycleMembers -> Node) |
      all n: cycleMembers |
         all disj a1, a2, a3: (members - cycleMembers) + n |
            (  n in a1.(^appendSucc)
            && a2 = a1.appendSucc
            && (a1 in a3.(^appendSucc) || a3 in a2.(^appendSucc) )
            )  => ! Between[a1,a3,a2]
}
```

the successor relation on appendages only

a1, a2, a3 have to be confined to the appendage tree rooted at n

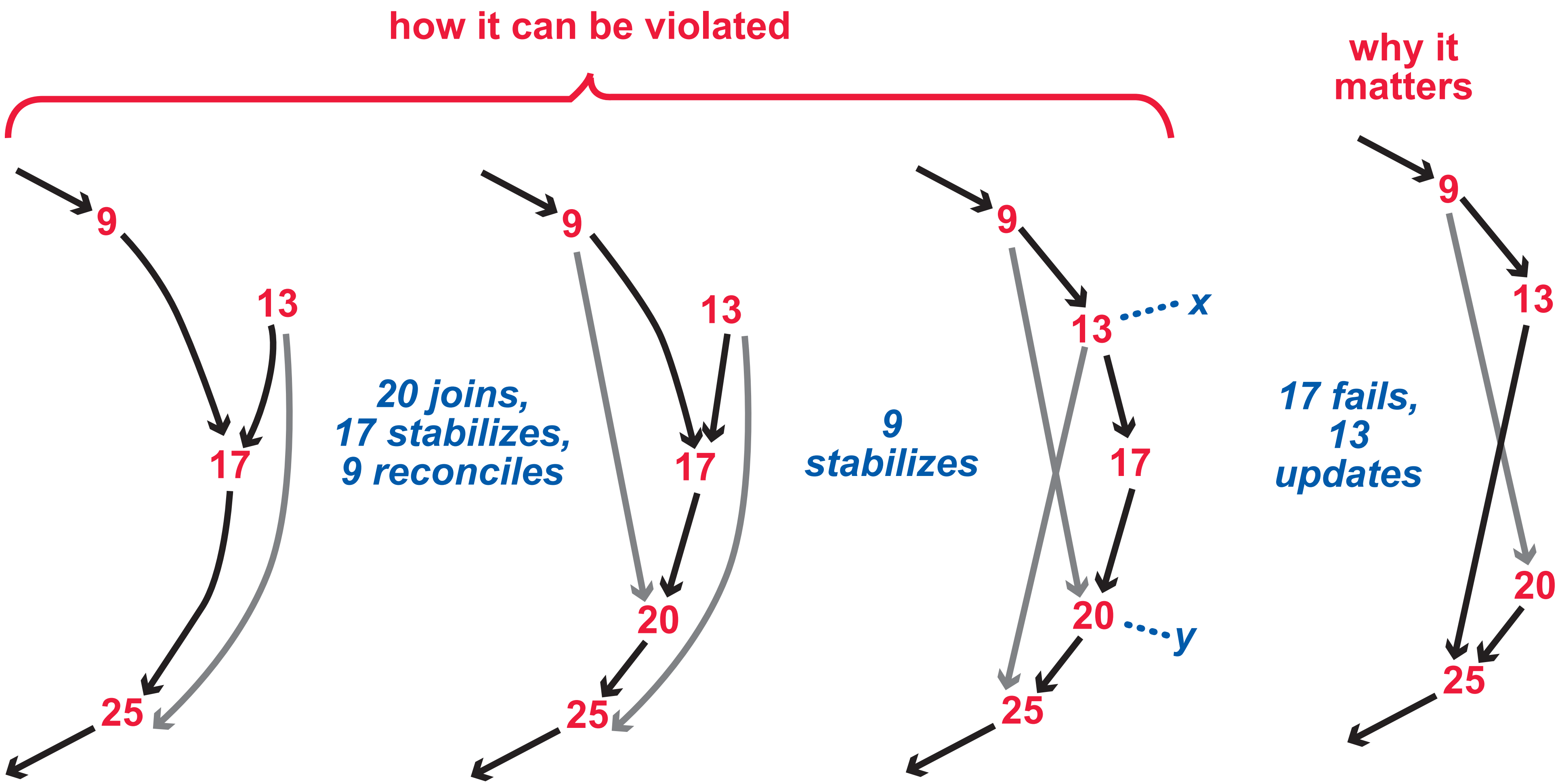


a3 can be 13

a3 cannot be 18

VALID SUCCESSOR LIST

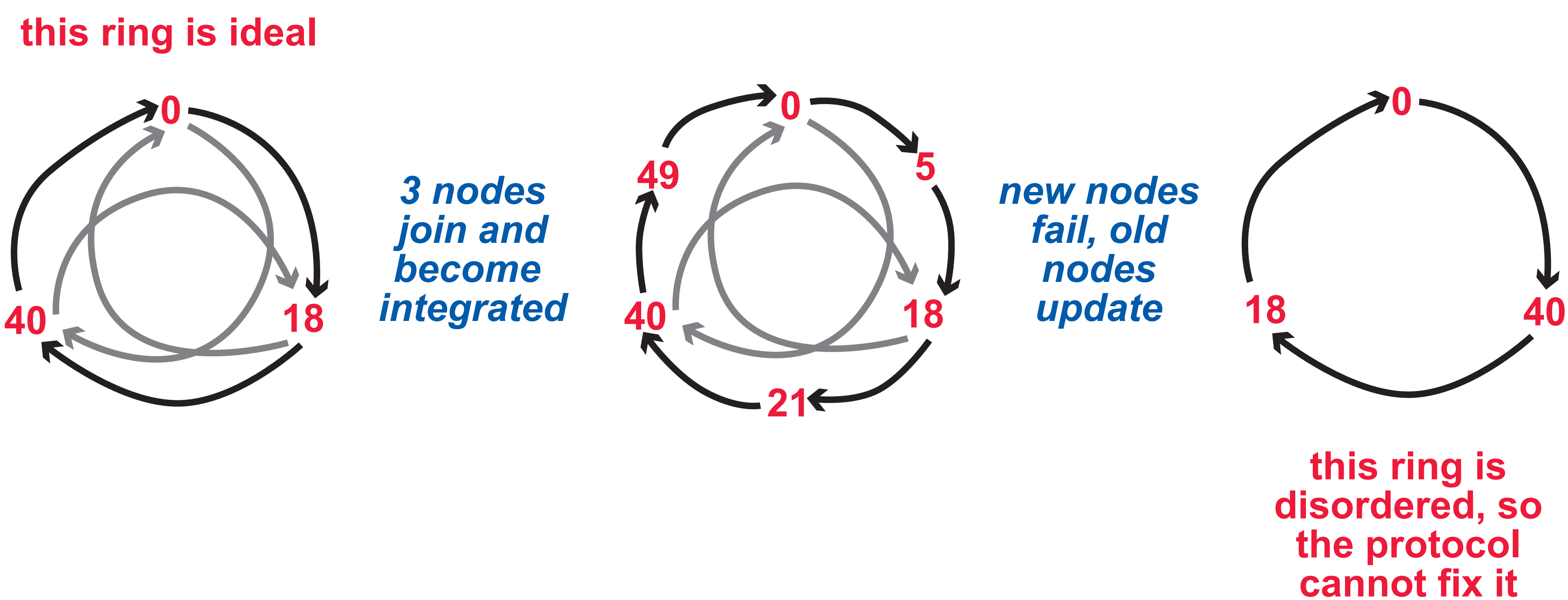
"if a node *x*'s successors skip over a live node *y*, then *y* is not in the successor list of any *x* antecedent"



20 was part of the cycle,
is now an appendage

WHY THE FULL PROTOCOL IS NOT CORRECT

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



this is actually a class of counterexamples:

- any ring of odd size becomes disordered
- any ring of even size splits into two disconnected subnetworks (which the protocol cannot fix)

COMPARISON, REVISITED

	PROMELA/SPIN	ALLOY
state structure	primitive in Promela; displayed poorly by Spin	Alloy language is rich and expressive; many display options
invariants	except for the most basic ones, an invariant must be written as a C program	Alloy language is rich, expressive, and concise
	sometimes searching for the right invariant requires a great deal of trial and error—this is why C programs don't make good invariants	these are not superficial properties—they cannot be slapped on top of Spin like frosting on a cake

at least two studies of Chord have been made using the model checker Mace, and they did not find any of these problems

- very few, very weak invariants, so Mace did not have much to look for
- working on Chord implementations, so Mace could only do heuristic checking, not complete checking