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

CASE STUDY: CHORD

CHORD IS A WELL-KNOWN DISTRIBUTED HASH TABLE

- has a lookup protocol, which will be ignored
- has a ring-maintenance protocol, which is the subject of study
- although the ring-maintenance protocol does not look much like a normal routing protocol, it has the same purpose

WHY CHOOSE CHORD?

- it's interesting! (actually, it chose me)
- it's easy, because the protocol is presented in compact pseudocode
- it's well-studied already
- "three features that distinguish Chord from many other peer-to-peer lookup protocols are its simplicity, provable correctness, and provable performance"

CONCLUSIONS

- more evidence for the value of lightweight modeling
- although it seems to be an unlikely candidate in many ways, Alloy is actually quite useful for lightweight modeling of network protocols, and is complementary to model checking

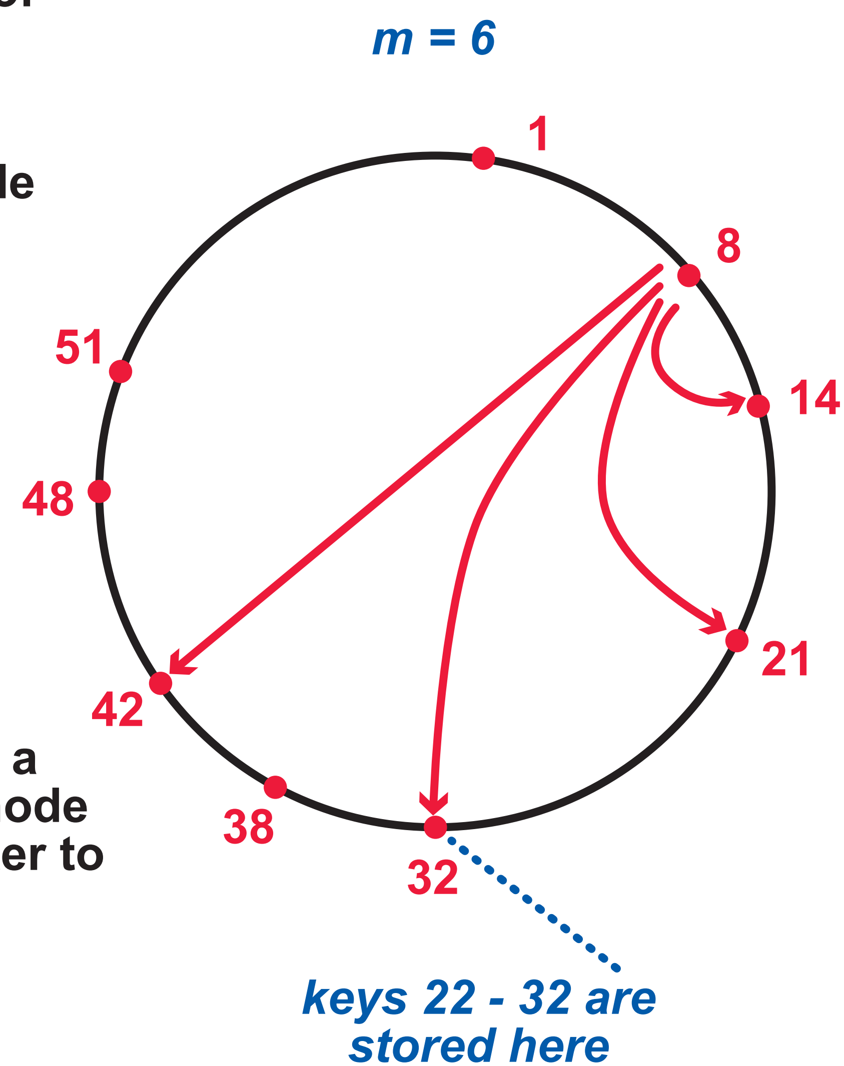
read paper for the full evidence

STORAGE AND LOOKUP OF (KEY, VALUE) PAIRS

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

in this talk I refer to a node only by its identifier

members are arranged in a ring, with each member node having a successor pointer to the next member node



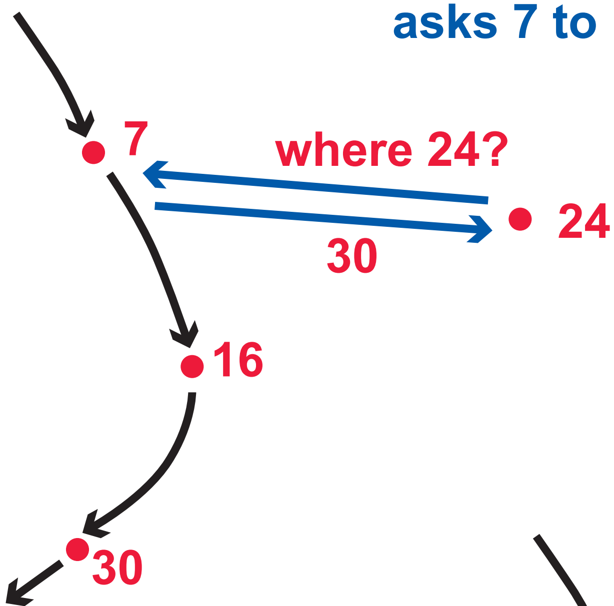
for efficient lookup, each node maintains a finger table, such as this one for node 8:

here + 1	14
here + 2	14
here + 4	14
here + 8	21
here + 16	32
here + 32	42

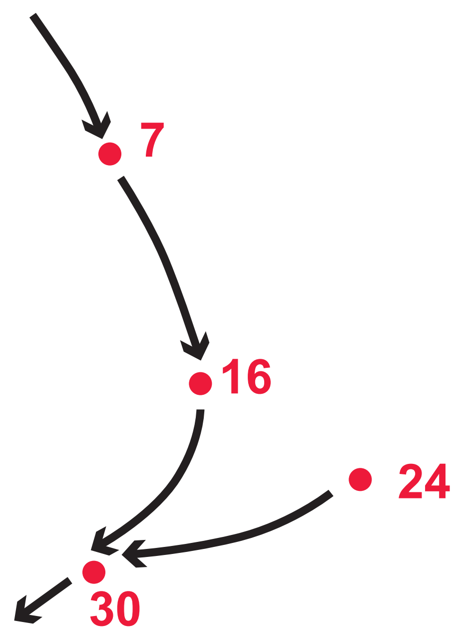
THE JOIN EVENT

BEFORE

joining node 24 must know some member 7, asks 7 to look up 24



AFTER

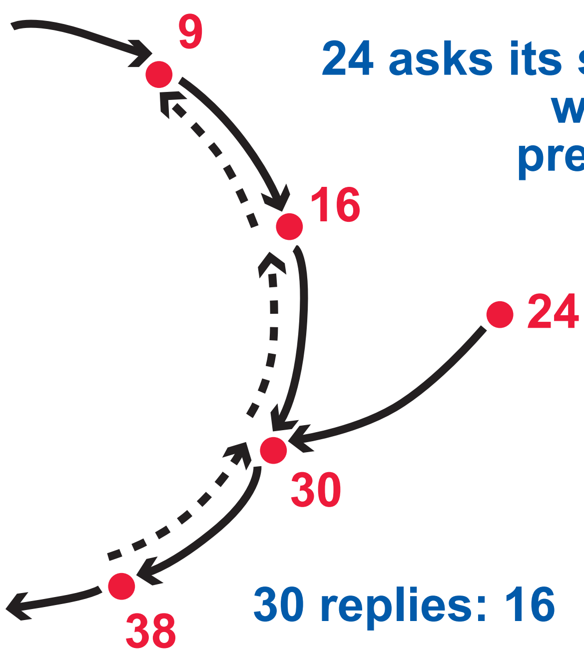


JOINS DISRUPT THE RING STRUCTURE BY ADDING APPENDAGES

THE RING STRUCTURE IS REPAIRED BY STABILIZATION

STABILIZATION OPERATION: THE PREQUEL

1: BEFORE STABILIZE EVENT



24 asks its successor: what is your predecessor?

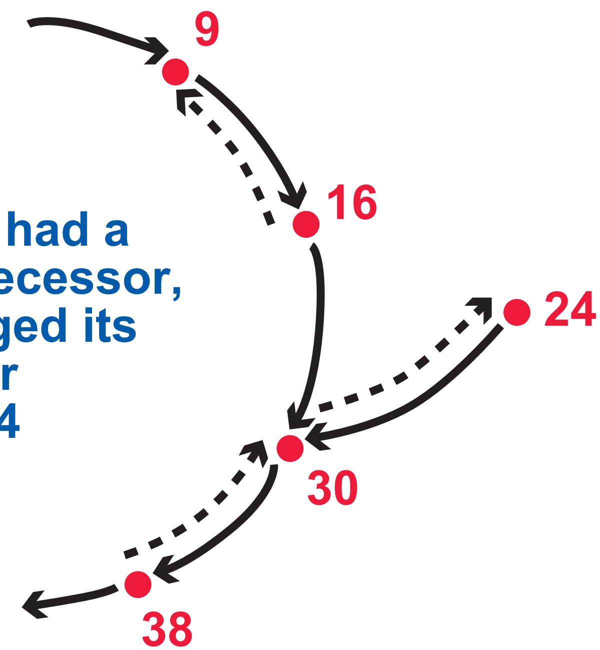
30 replies: 16

2: AFTER STABILIZE EVENT, BEFORE NOTIFY EVENT

because 16 is not a better successor to 24 than 30 is, 24 has not changed its successor pointer

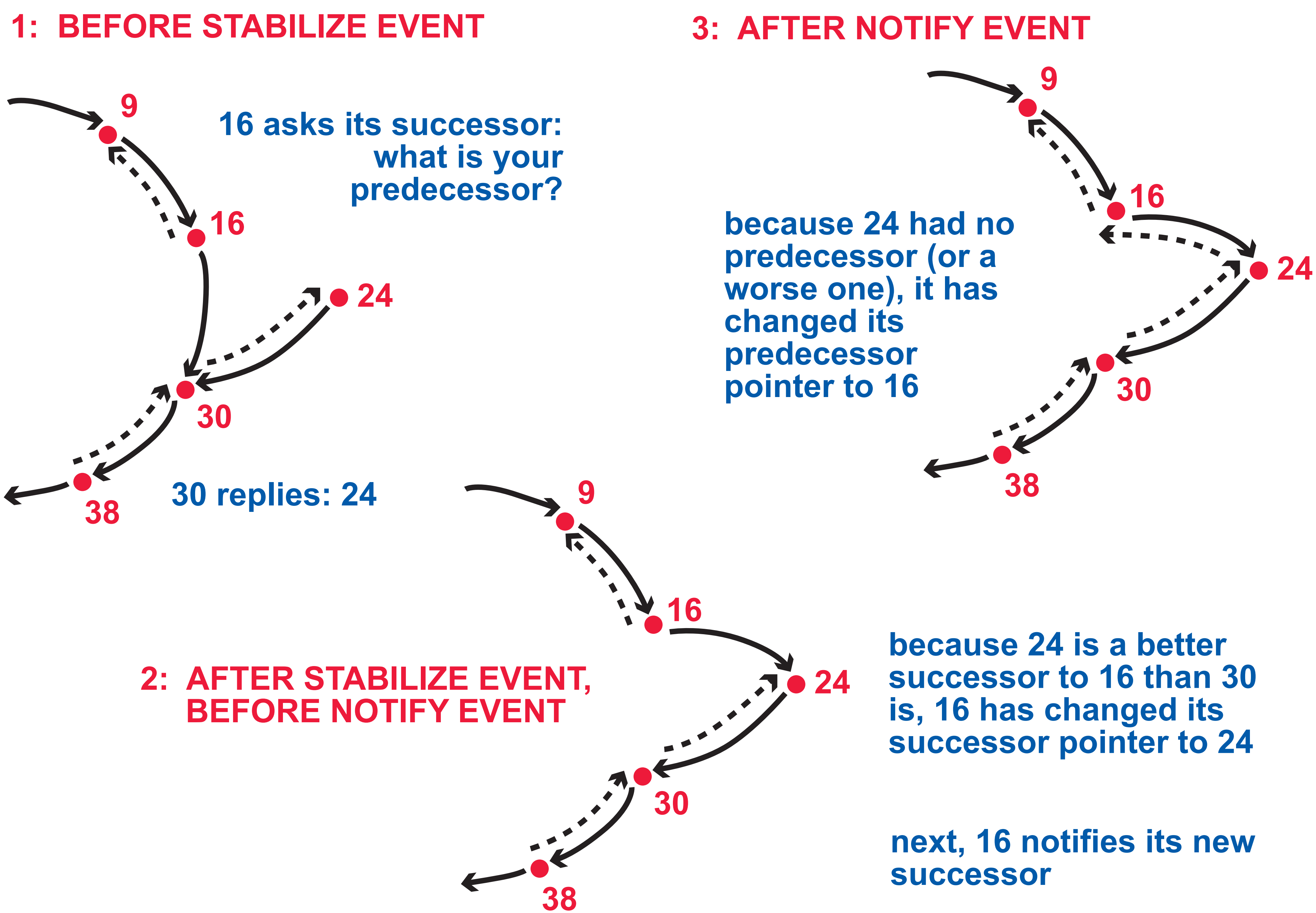
next, 24 notifies its successor

3: AFTER NOTIFY EVENT

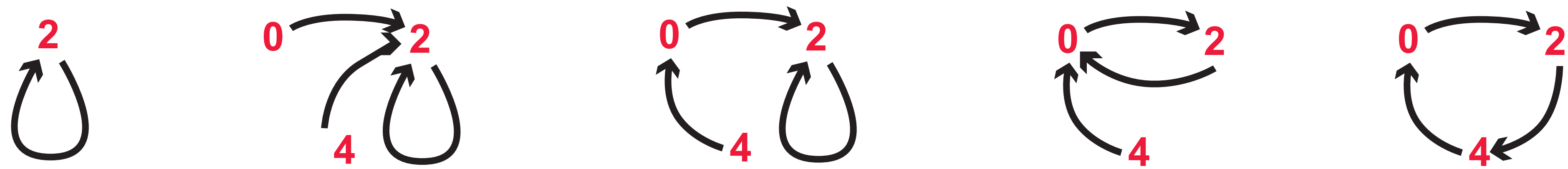


because 30 had a worse predecessor, it has changed its predecessor pointer to 24

THE STABILIZATION OPERATION

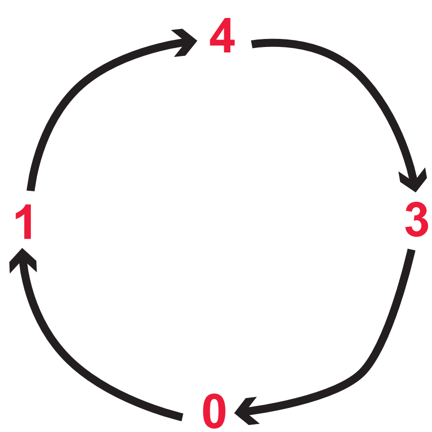


WHAT STABILIZATION CAN DO . . .



. . . AND CANNOT DO

stabilization cannot fix a disordered ring



FULL RING-MAINTENANCE PROTOCOL ALSO INCLUDES NODE FAILURES (OR SILENT LEAVING) . . .

. . . AND RECONCILIATION (WHICH RECOVERS FROM FAILURES)

so there are "join-only" and "full" models—only the join-only model can be proven correct

CHORD STATE (JOIN-ONLY MODEL)

```
sig Node {
   succ: Node lone -> Time,
   prdc: Node lone -> Time  }
```

the type of succ and prdc is Node -> Node -> Time

the successor of node n at time t is n.succ.t

at any time, a node has 0 or 1 successors

```
pred Member [n: Node, t: Time] { some n.succ.t }
```

a node is a member of the ring at time t if it has a successor at time t

```
fact { all n: Node, t: Time | no n.succ.t => no n.prdc.t }
```

if it is not a member, it does not have a predecessor, either

```
pred Between [n1, n2, n3: Node] {
   lt[n1,n3] =>   ( lt[n1,n2] && lt[n2,n3] )
             else ( lt[n1,n2] ||   lt[n2,n3] )  }
```

Between [5, 10, 12]

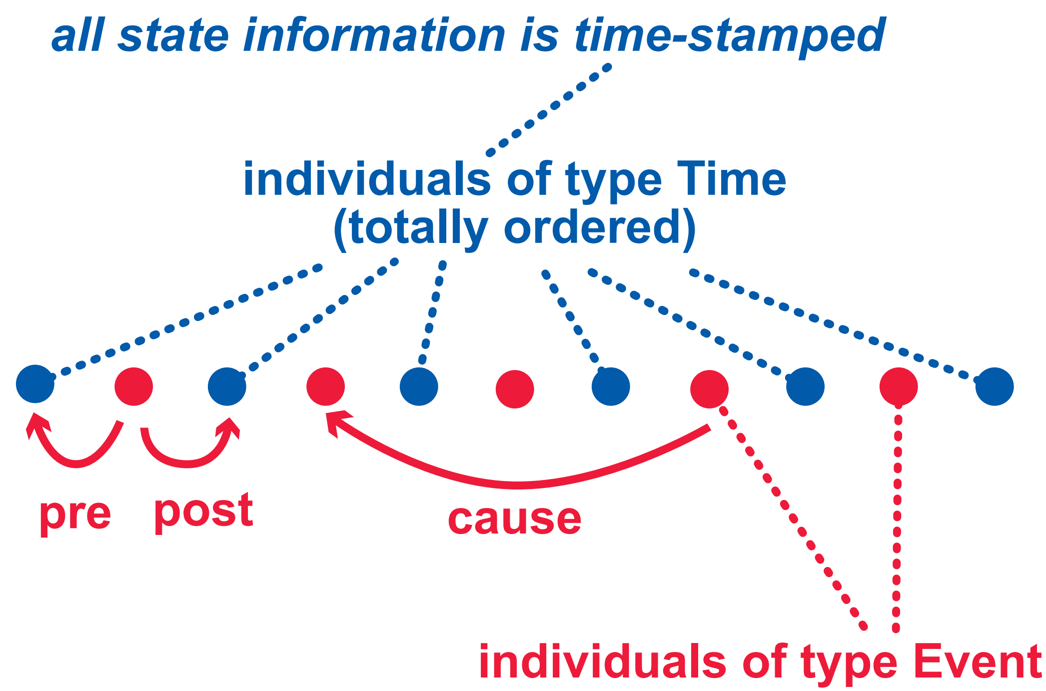
Between [51, 1, 3]

Between [51, 59, 3]

THE MODEL OF TIME (SAME IN EVERY MODEL)

```
sig Time { }
open util/ordering[Time] as trace

abstract sig Event {
   pre:  Time,
   post: Time,
   cause: lone (Event - Null)
}
```




```
fact TemporalStructure {
   all t: Time - trace/last | one e: Event | e.pre = t
   all t: trace/last | no  e: Event | e.pre = t
   all e: Event | e.post = (e.pre).trace/next
}

sig Null extends Event { } { no cause }
```

in a scope with 6 Times, there *must* be 6 times and 5 events—null events take up the slack

```
fact CauseHasSingleEffect { cause in Event lone -> Event }

fact CausePrecedesEffect {
   all e1, e2: Event | e1 = e2.cause => lt[e1.pre,e2.pre]  }
```

these are for convenience in modeling events

MODEL OF JOIN EVENTS

```
abstract sig RingEvent extends Event { node: Node }

sig Join extends RingEvent { } { no cause }

fact NonmemberCanJoin {
  all j: Join, n: j.node, t: j.pre |
      ! Member[n,t]
    && (some m: Node |      Member[m,t]
                        && Between[m,n,m.succ.t]
                        && n.succ.(j.post) = m.succ.t
       )
    && no n.prdc.(j.post)
    && no cause:>j
}
```

every ring event has a node field; event can only change the fields of this node

a join event is a ring event; it has no cause, so it can occur at any time that its preconditions are satisfied

preconditions: n is not a member already; there is a member m such that n is between m and m's successor

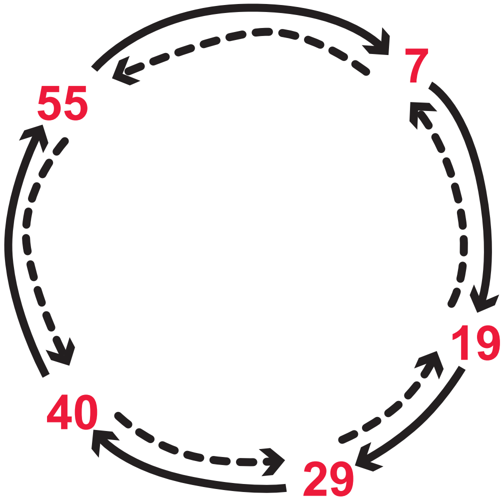
postconditions: the successor of n is the successor of m; n does not have a predecessor (yet)

this event does not cause any event

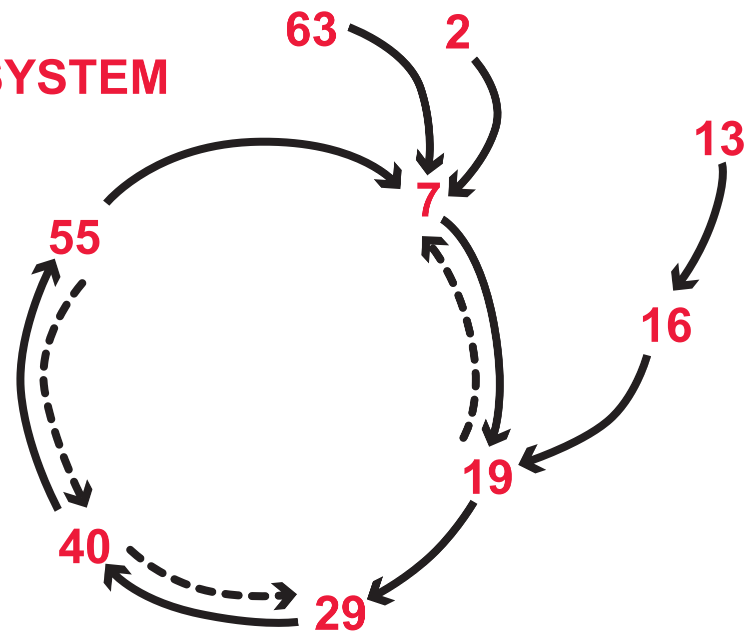
in addition, there are "frame conditions" saying that node fields are not changed except as specified in event specifications

IDEAL VS. VALID STATES

IDEAL



VALID (THE SYSTEM INVARIANT)



VALID is a conjunction of several properties, such as . . .

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

there is at least one cycle

there is no more than one cycle

the cycle is globally ordered by identifiers

VERIFICATION LEMMAS

```
assert InitialIsValid {
  let members = { n: Node | Member[n,trace/first] } |
    (    one members
    && members.succ.trace/first = members
    && no members.prdc.trace/first
    )    => Valid[trace/first]
}
check InitialIsValid for 8 but 0 Event, 1 Time
```

can also analyze traces in which two stabilizations or a join and a stabilization interleave—protocol behavior is different, but the difference is benign

```
assert JoinPreservesValidity {
  some Join && Valid[trace/first] => Valid[trace/last] }
check JoinPreservesValidity for 8 but 1 Event, 2 Time

assert StabilizationPreservesValidity {
  (some Stabilize && Valid[trace/first])
  => (Valid[trace/first.trace/next] && Valid[trace/last]) }
check StabilizationPreservesValidity for 8 but 2 Event, 3 Time

assert ValidRingIsImprovable {
  (Valid[trace/first] && ! Ideal[trace/first]) =>
  (  (some n, newSucc: Node |
        StabilizationWillChangeSuccessor[n,newSucc,trace/first])
  || (some n, nSucc: Node |
        StabilizationShouldChangePredecessor[n,nSucc,trace/first] )
  )
}
check ValidRingIsImprovable for 8 but 0 Event, 1 Time
```

true if stabilization at n will change its successor to newSucc

true if stabilization at n will change the predecessor of nSucc to n

PROOF OUTLINE

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

PROOF:

1 Show that *Valid* is an invariant.

check lemmas:
- "initial ring is *Valid*"
- "join preserves *Valid*"
- "stabilization preserves *Valid*"

2 Show that any time the network is *Valid* and not *Ideal*, some stabilization that will change the network is enabled.

formalize as a lemma and check

Alloy Analyzer checks, exhaustively, all model instances with up to 8 network nodes

this is convincing evidence because no relevant example or counterexample (of *many*) has been bigger than 4

3 Show that the network will become *Ideal*.

- enabled stabilizations and therefore changes will continue to occur
- every change is an improvement
- because the ring is finite, after a finite number of improvements it will be ideal

4 Show that the network will remain *Ideal*.

check lemma "stabilization cannot change an ideal ring"

■

COMPARISON

the price of push-button analysis

	PROMELA/SPIN	ALLOY
model state	small and bounded	small and bounded
reachable state space	automatically generated, exact, not readable	invariant is a user-constructed superset; readable
traces	Spin explores all traces	because temporal sequences are not built in and not optimized (e.g., successive states are both represented even if they are identical), Alloy can only explore short traces
temporal logic	Spin automatically checks any formula in temporal logic	Alloy Analyzer can only check safety properties
temporal sequencing	built into Promela; displayed well by Spin	not built into Alloy language
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