BFT Protocol Forensics

Presented by: Chirag Shetty, Milind Kumar V

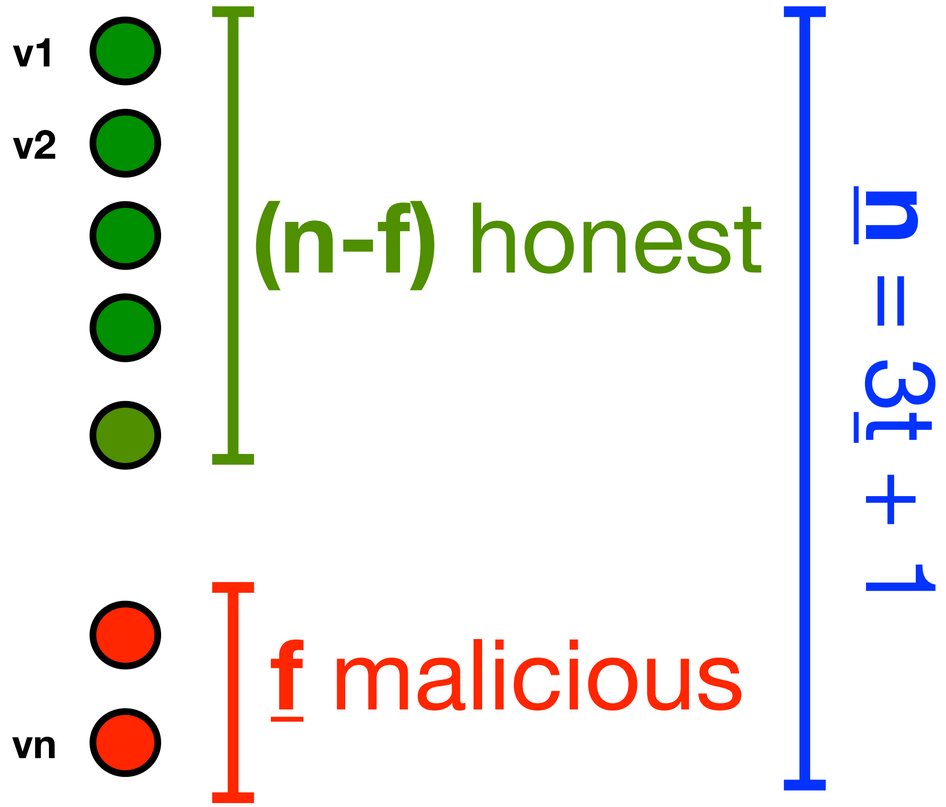
22 April 2022

UIUC, CS 598 FTD (Spring 2022)

The problem we will look at

- SMR
  - with a single value
- Partial synchrony
- Byzantine faults

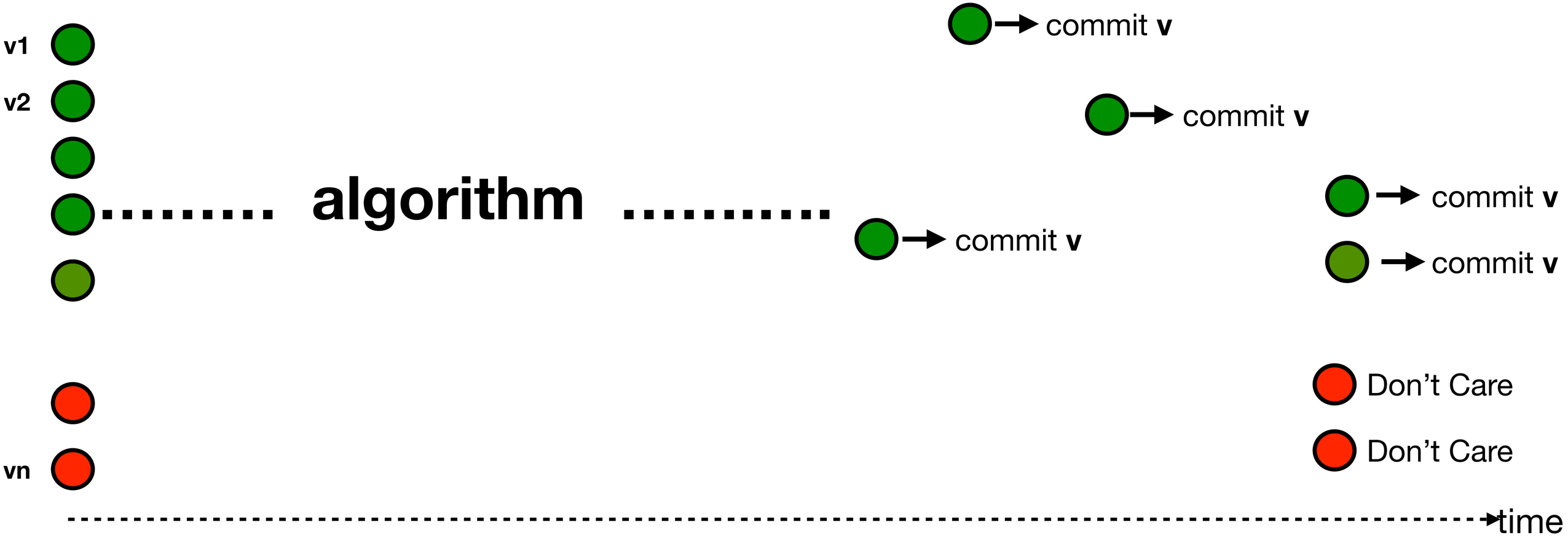
Parameters of the algorithm



Guarantees are provided when f < t + 1

- Safety (Agreement)
- Liveness (Termination)
- Validity

f <= t : All good



n = 3t + 1 **(n-f)** honest **f** malicious **f<=t**

But what if f > t?

Forensics: investigation after a safety violation

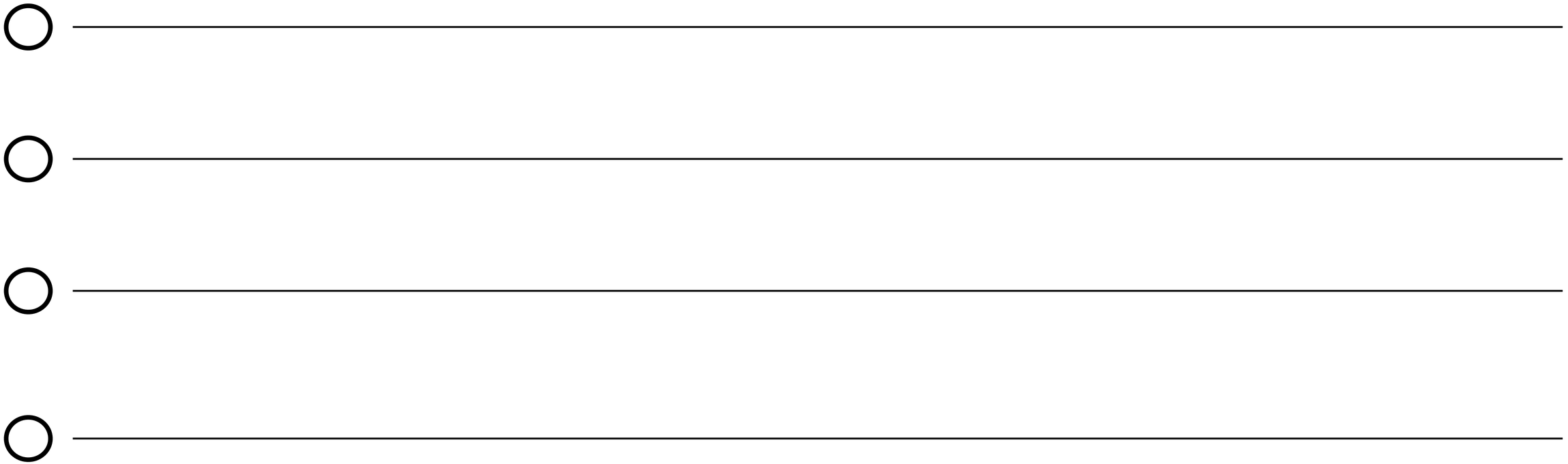
- Identify malicious nodes
  - As many as possible
  - With cryptographic proof
  - In a distributed fashion
- Formalized as forensic support

Parametrizing forensic support as (m, k, d)

- m: maximum number of Byzantine replicas
- k: number of honest replicas needed for proof
- d: number of identifiable Byzantine replicas

PBFT-PK has (2t, 1, t+1) support

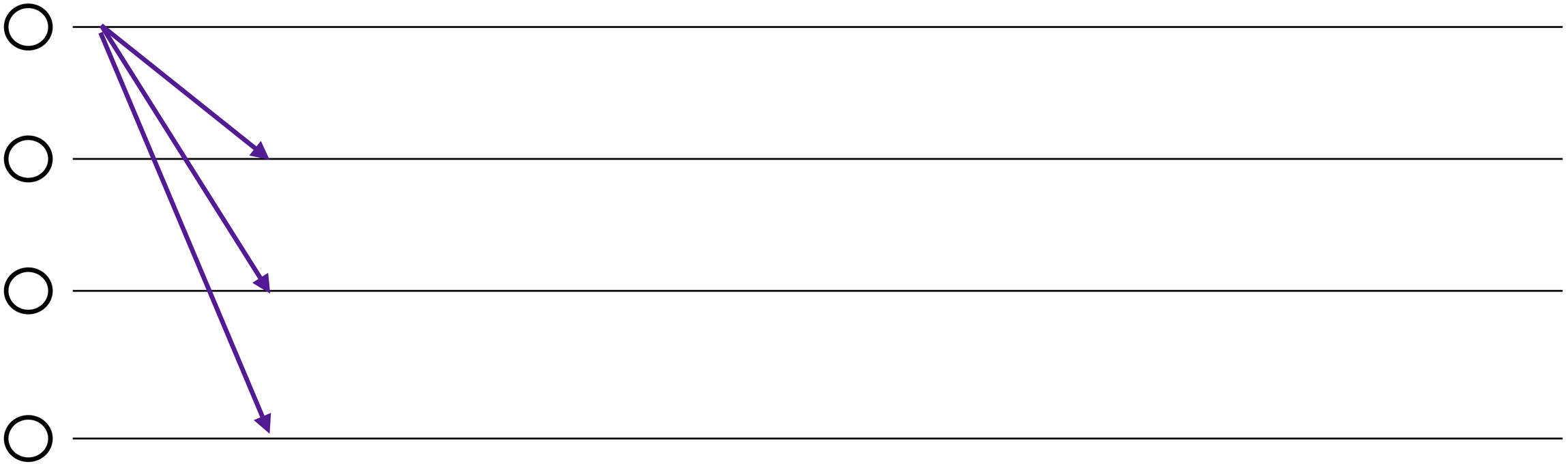
PBFT Steady State



n = 4, t=1

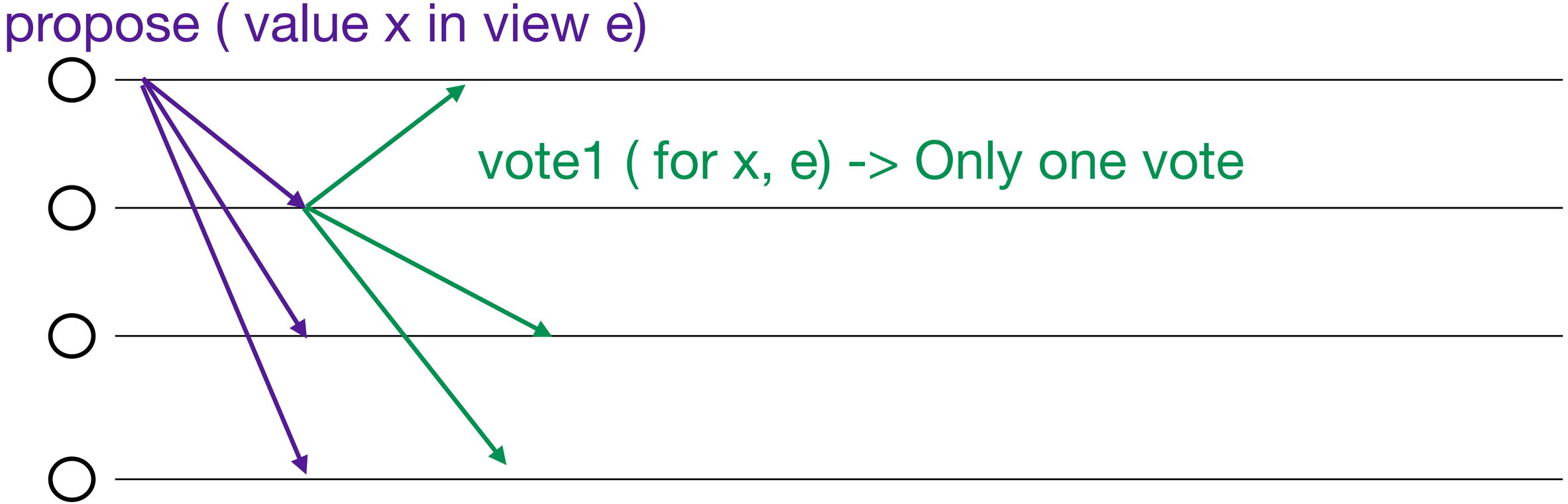
PBFT Steady State

propose (value x in view e)



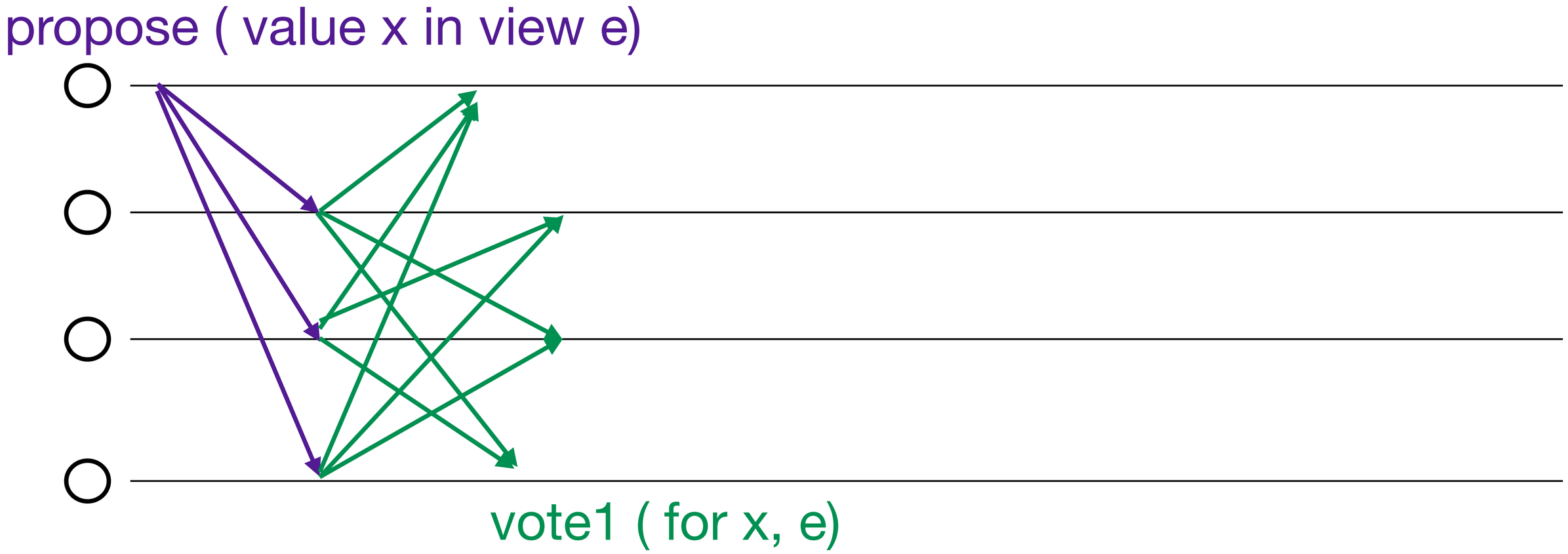
n = 4, t=1

PBFT Steady State



n = 4, t=1

PBFT Steady State

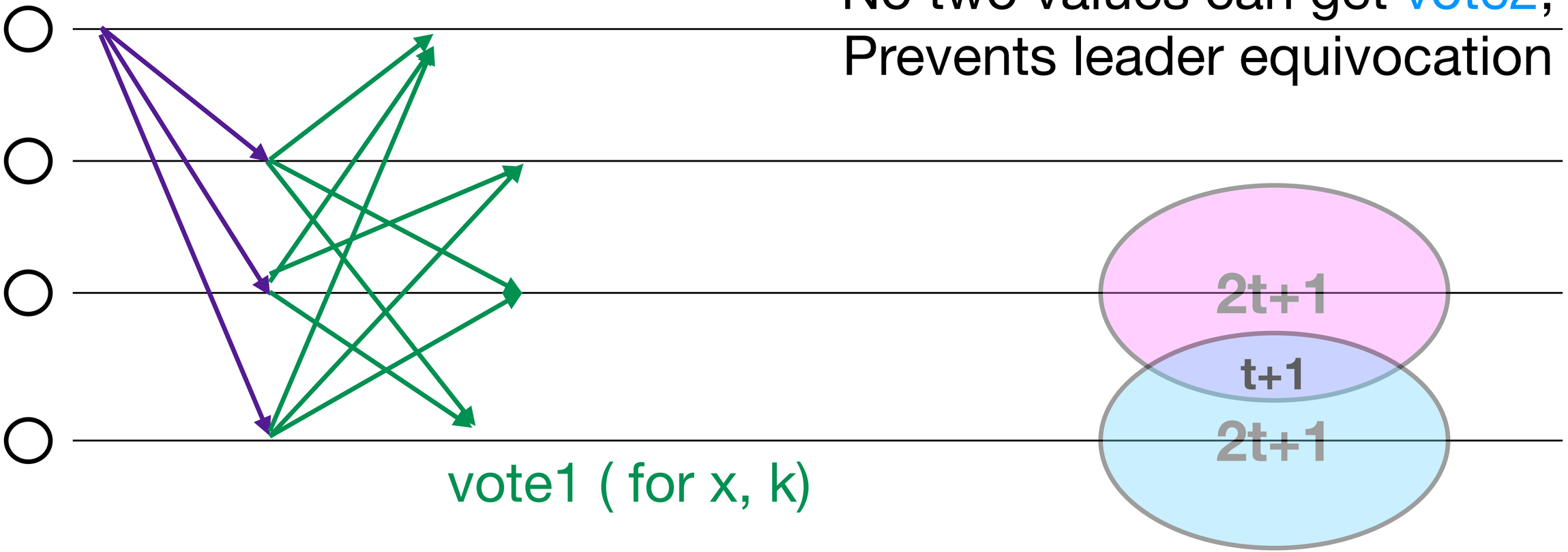


n = 4, t=1

PBFT Steady State

On receiving 2t+1 vote1 for a value, lock value, send vote2

propose (value x in view k)



vote1 (for x, k)

No two values can get vote2,
Prevents leader equivocation

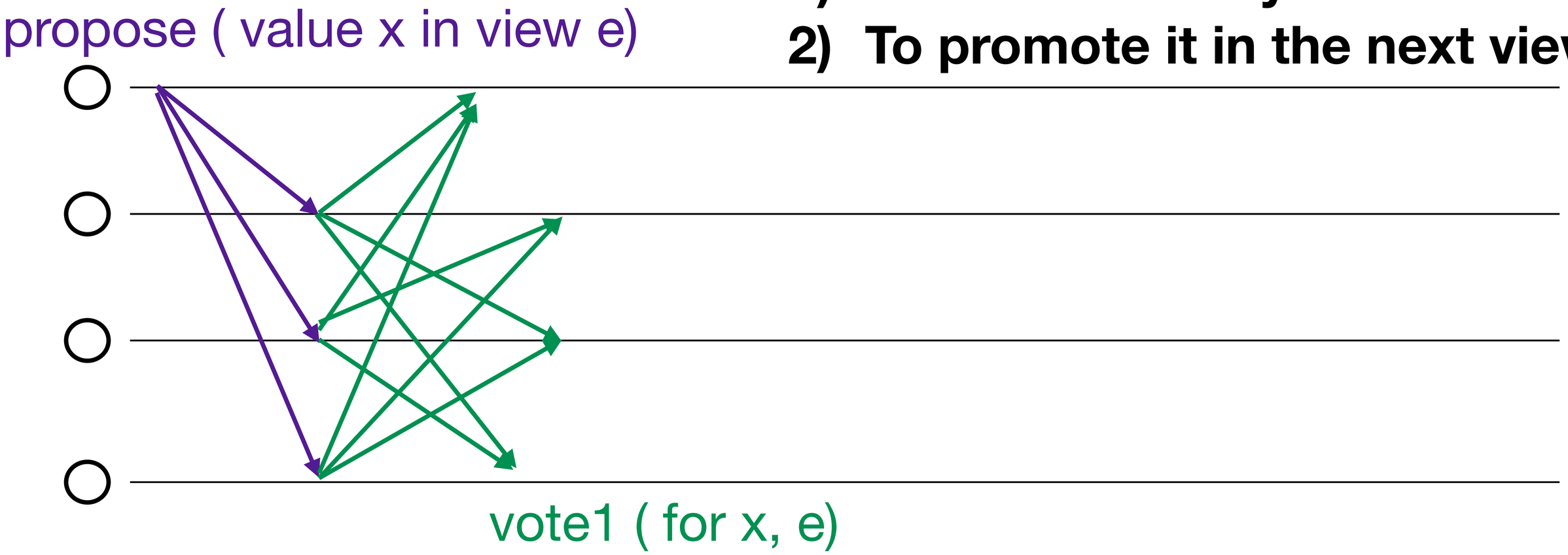
n = 4, t=1

PBFT Steady State

Lock on (x,e):

1) vote2 for only it in view

2) To promote it in the next view

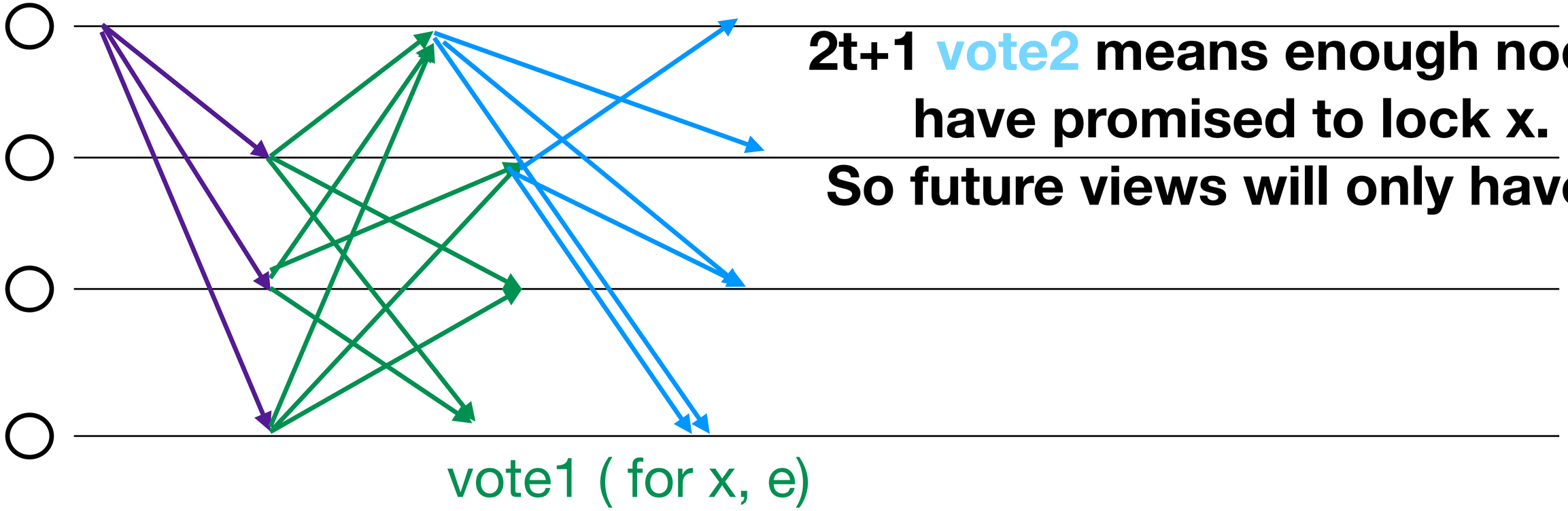


n = 4, t=1

PBFT Steady State

On receiving 2t+1 vote2 for (x,e),
commit value x

propose (value x in view e)



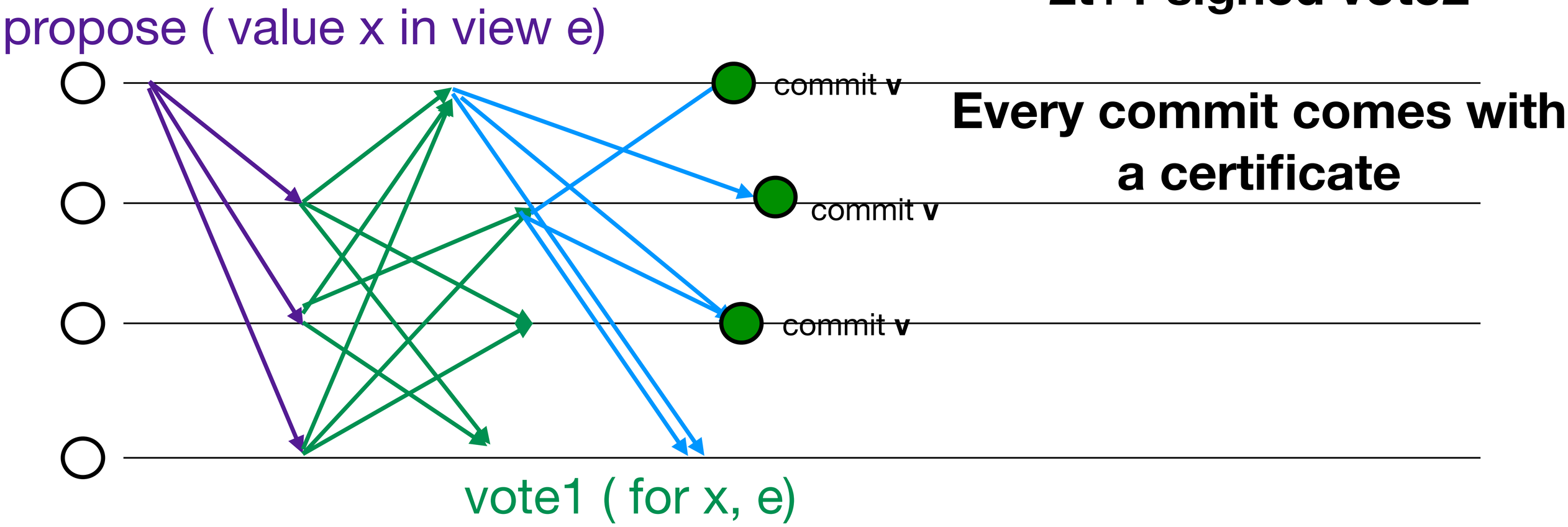
2t+1 vote2 means enough nodes have promised to lock x.
So future views will only have x

vote1 (for x, e)

n = 4, t=1

PBFT Steady State

Remember, commit needs 2t+1 signed vote2

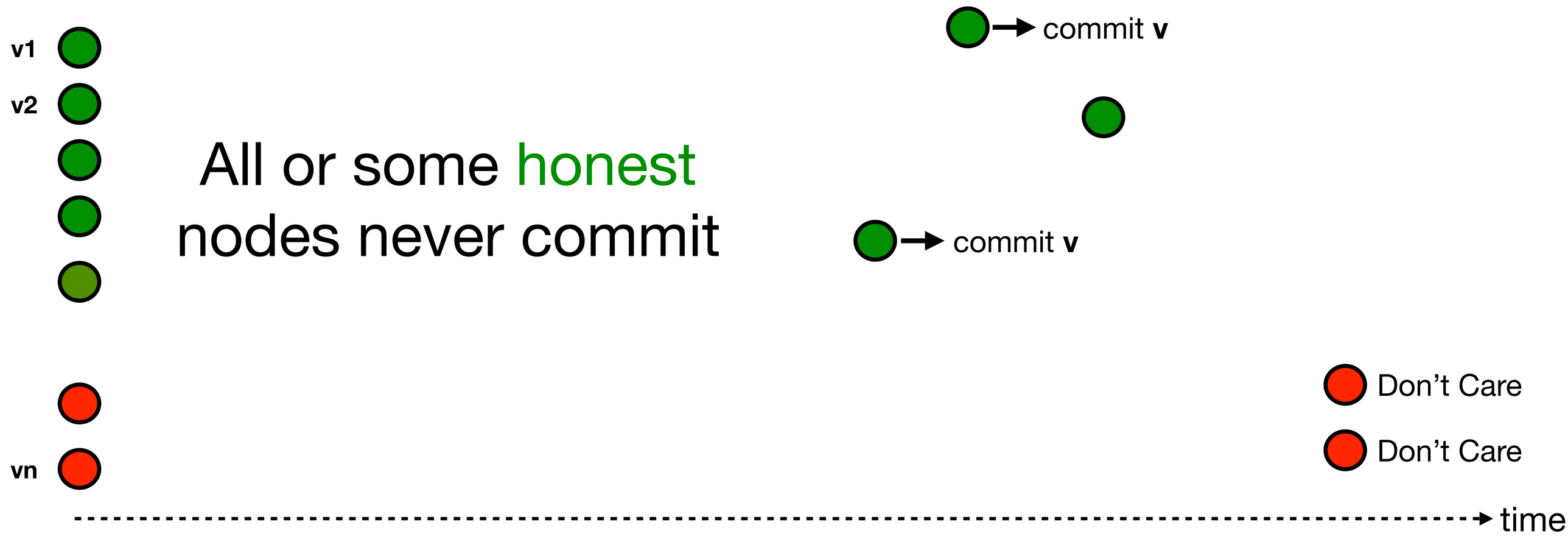


Every commit comes with a certificate

n = 4, t=1

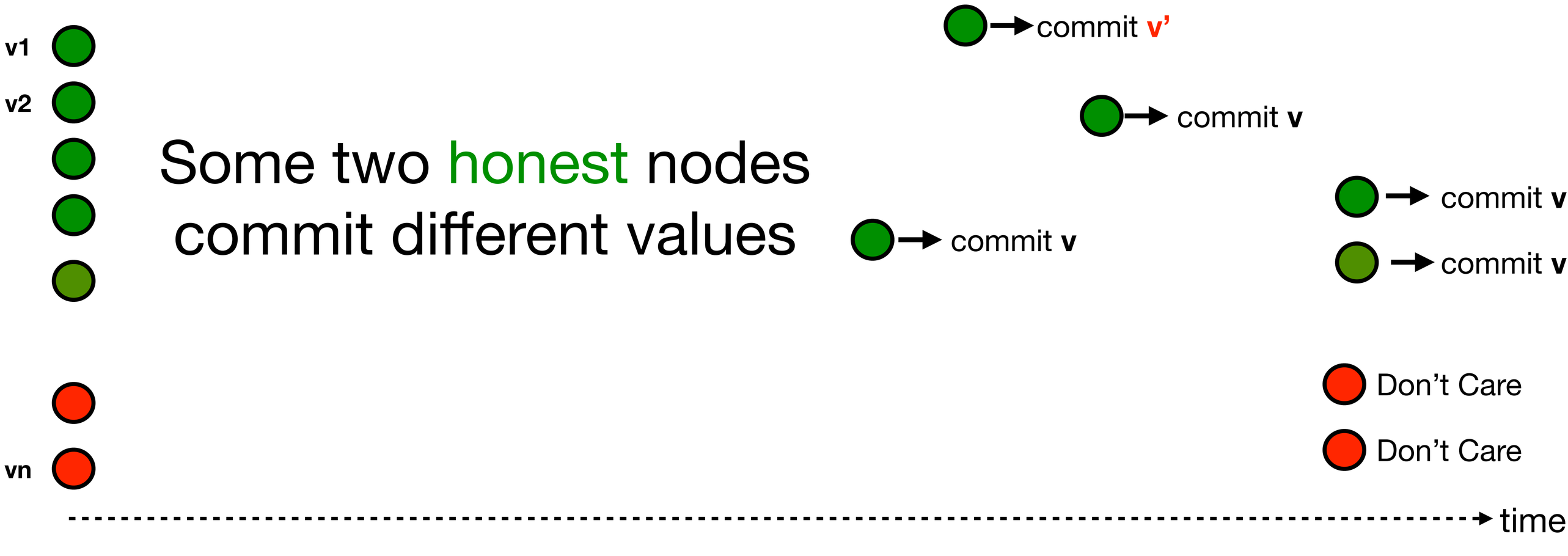
But what if f > t?

f > t: Case 1 - Liveness Violation



n = 3t + 1 **(n-f)** honest **f** malicious **f>t**

f > t: Case 2 - Safety Violation



Some two honest nodes commit different values

n = 3t + 1 **(n-f)** honest **f** malicious **f>t**

Suppose, there was a safety violation:

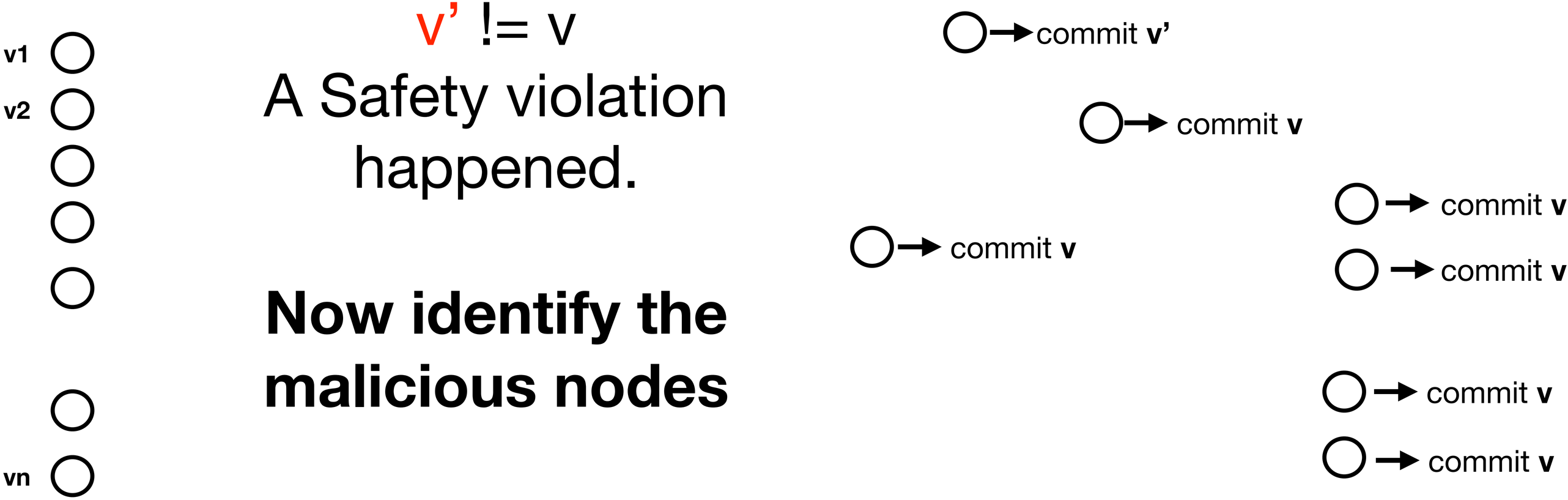
Two nodes committed two different values

Safety Violation: “The day after”

v’ != v

A Safety violation happened.

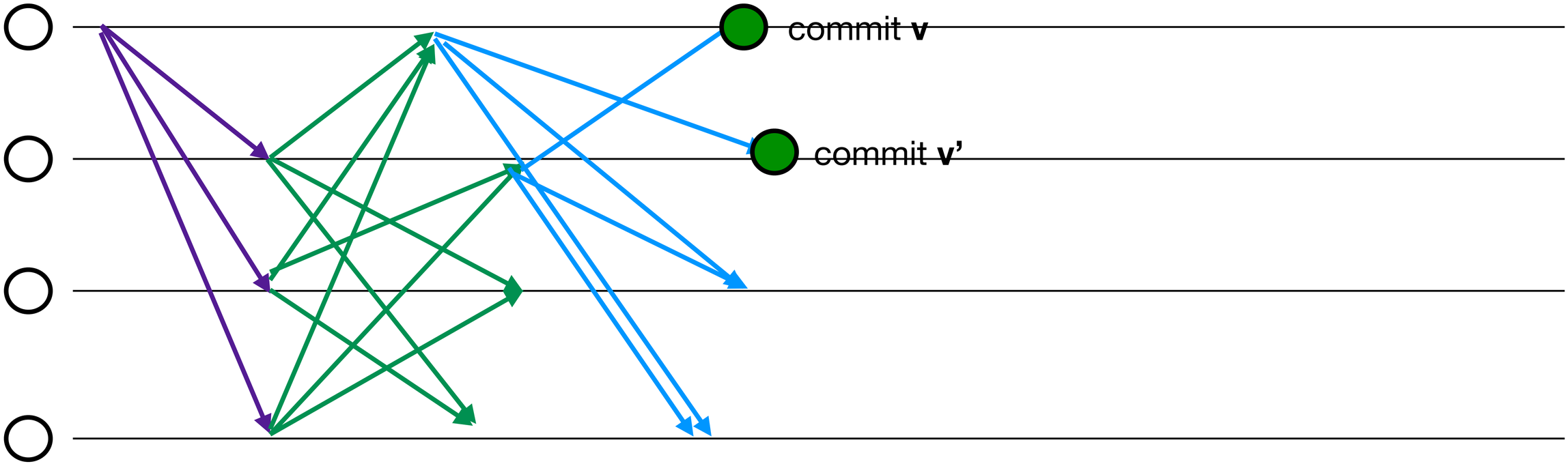
Now identify the malicious nodes



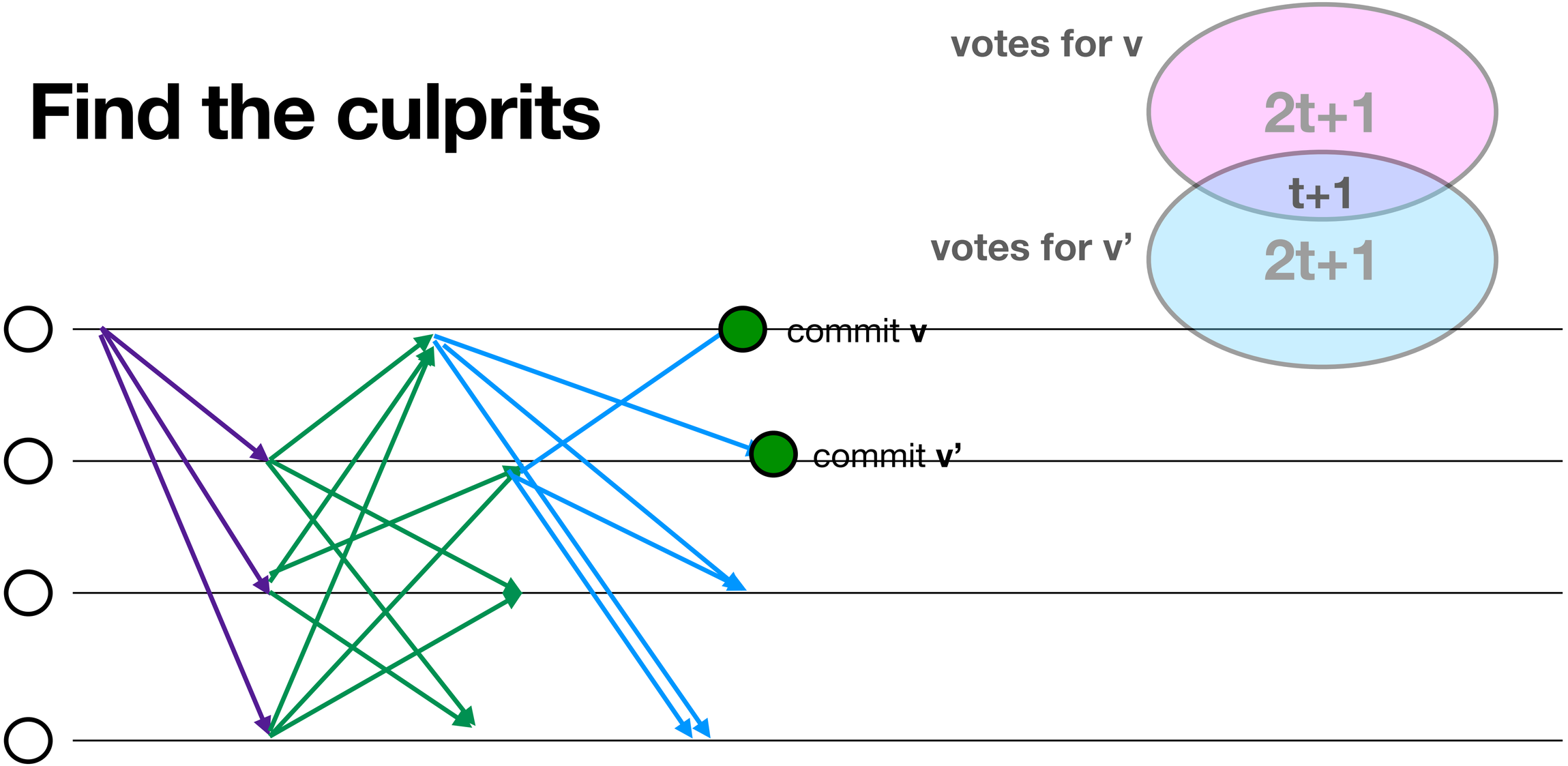
n = 3t + 1 **(n-f)** honest **f** malicious **f>t**

Case 1: In same view

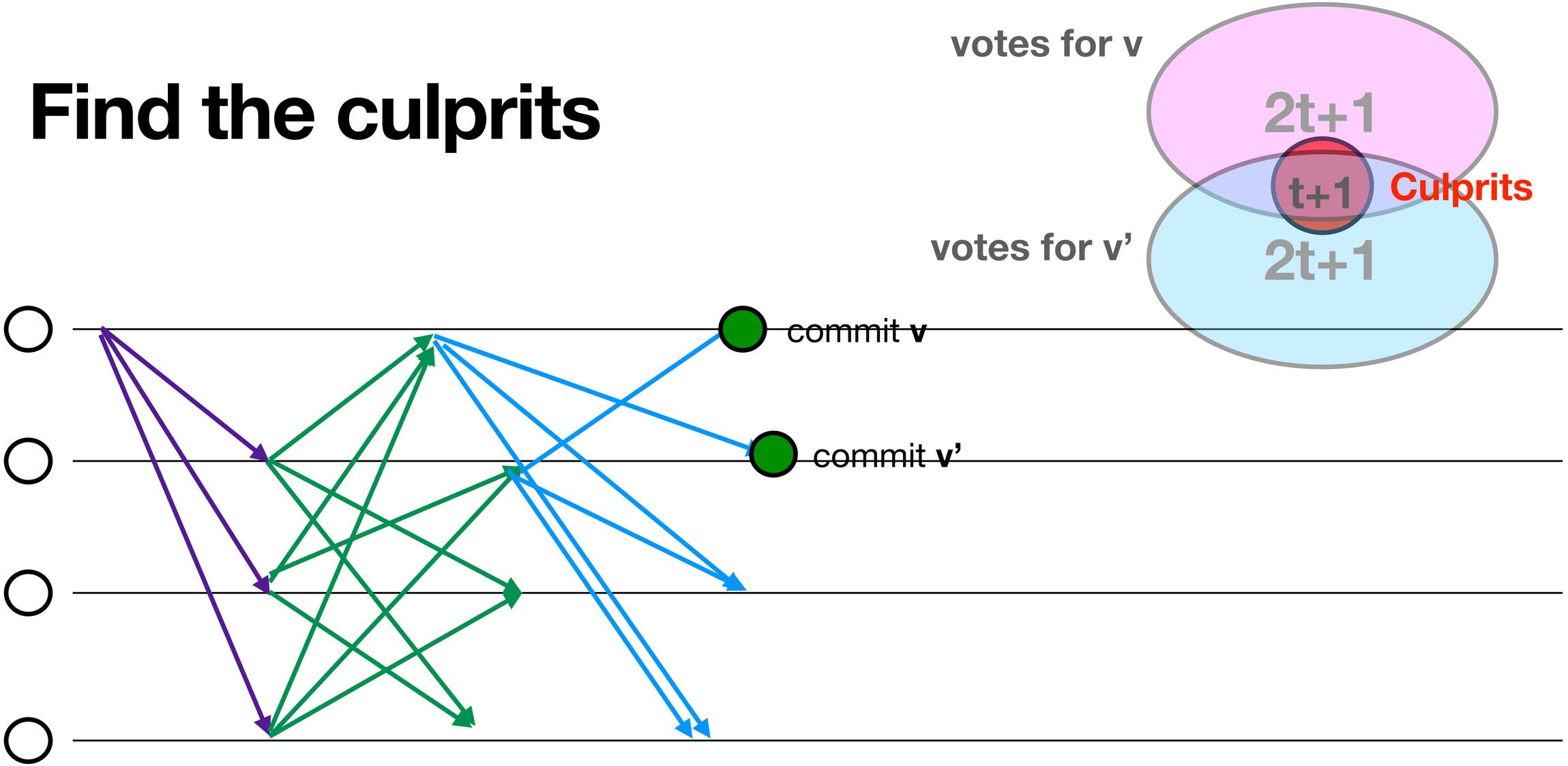
Find the culprits



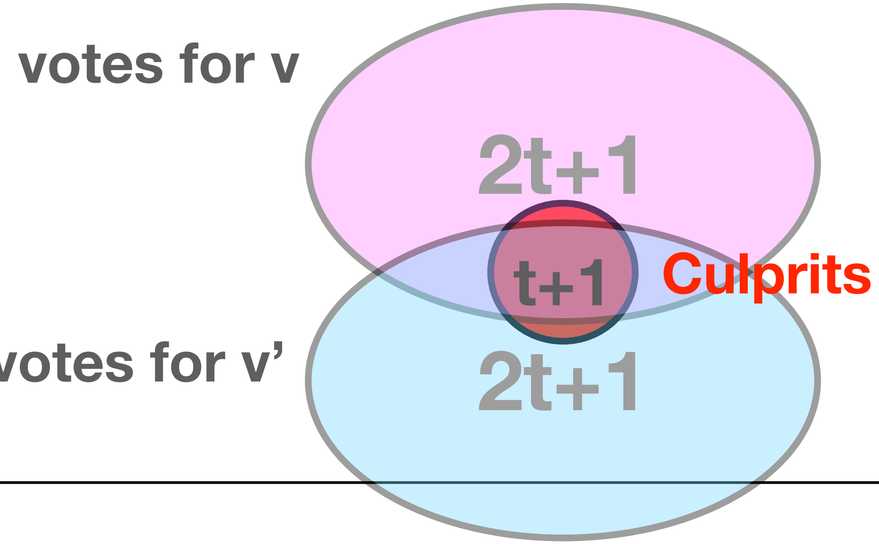
Find the culprits

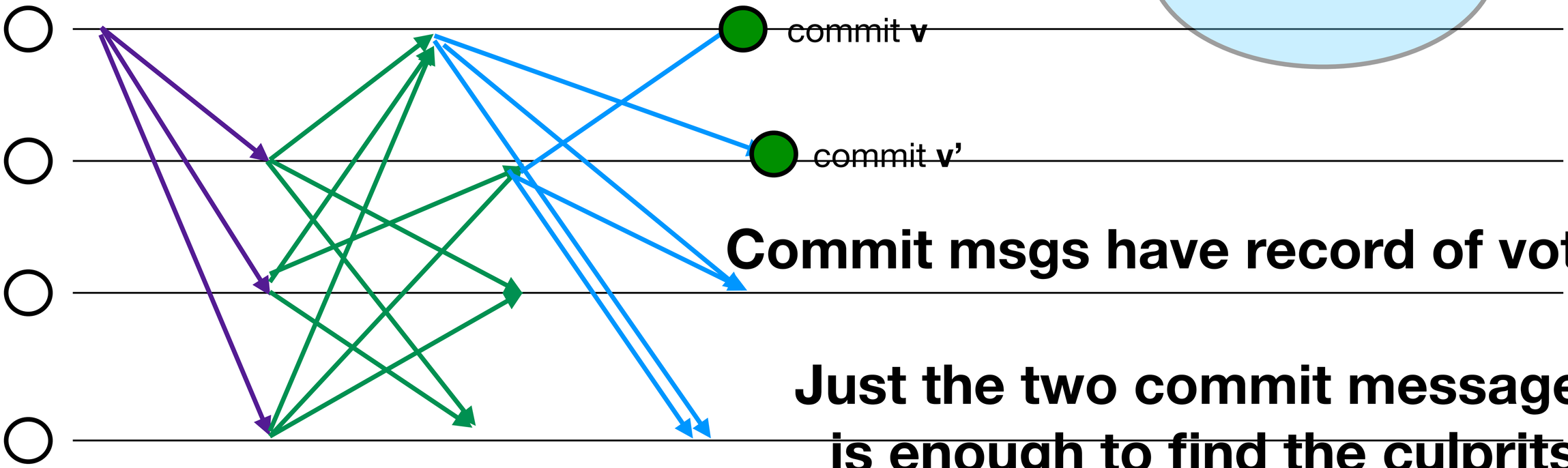


Find the culprits



Find the culprits





Commit msgs have record of vote2's

Just the two commit messages is enough to find the culprits

No transcript needed

Case 2: Across views (x,e) and (x’, e’)

Timeout waiting for a commit-> view change

At any node, If leader seems faulty,
send blame(leader e) + status to all

Timeout waiting for a commit-> view change

At any node, If leader seems faulty,
send blame(leader e) + status to all

On receiving 2t+1 blames,
leader e+1 can request view change

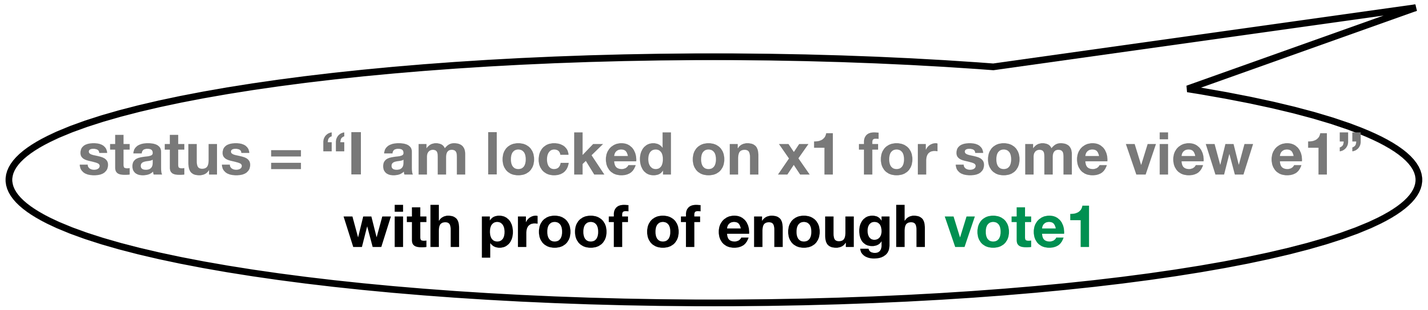
Timeout -> view change

At any node, If leader seems faulty,
send blame(leader k) + status to all

On receiving 2t+1 blames,
leader e+1 can request view change

Timeout -> view change

At any node, If leader seems faulty,
send blame(leader e) + status to all



Promise to ensure future views re-propose potentially committed value

On receiving 2t+1 blames,
leader e+1 can request view change

Timeout -> view change

On receiving 2t+1 blames,
leader e+1 can request view change

Why 2t+1?

Timeout -> view change

On receiving 2t+1 blames,
leader e+1 can request view change

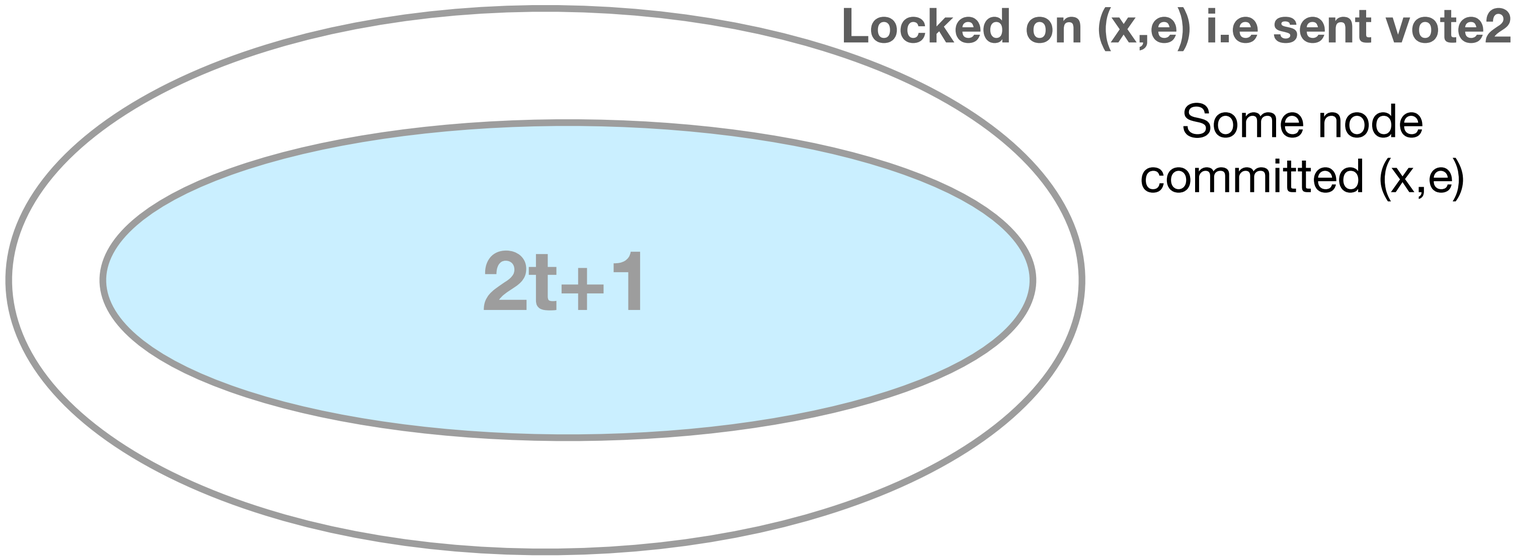
Why 2t+1?

So that the new leader is guaranteed
to see atleast one node that has the latest lock

Timeout -> view change

On receiving 2t+1 blames,
leader e+1 can request view change

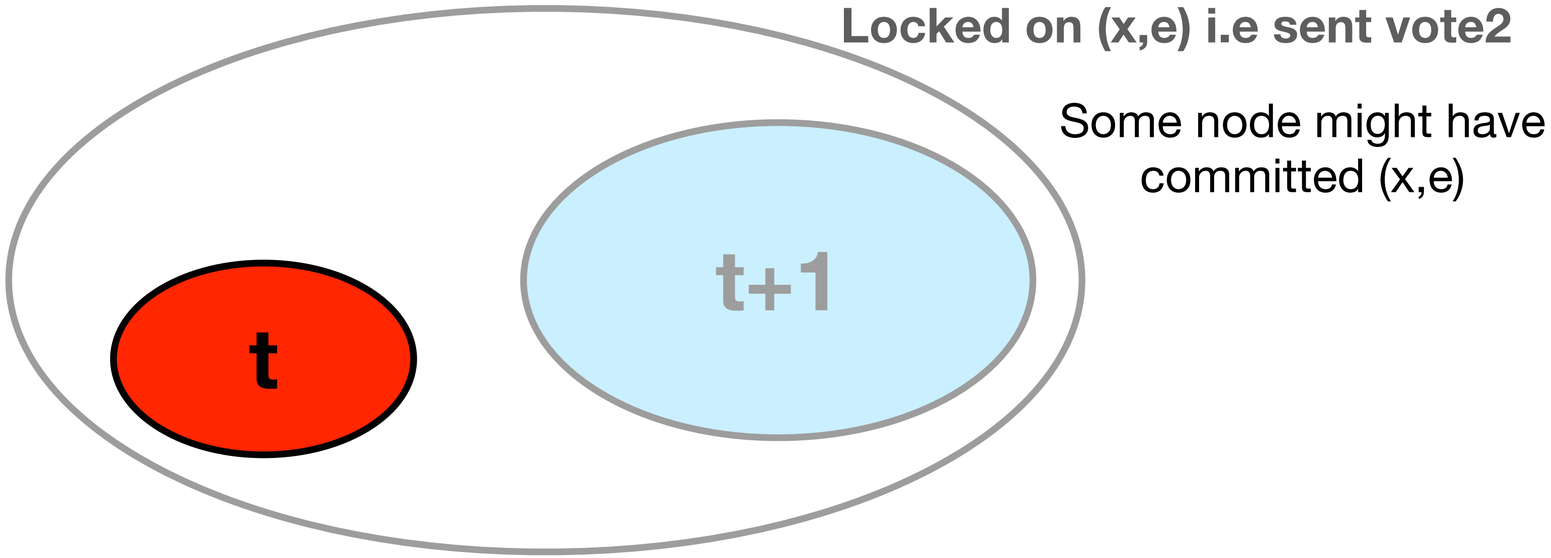
Why 2t+1?



Timeout -> view change

On receiving 2t+1 blames,
leader e+1 can request view change

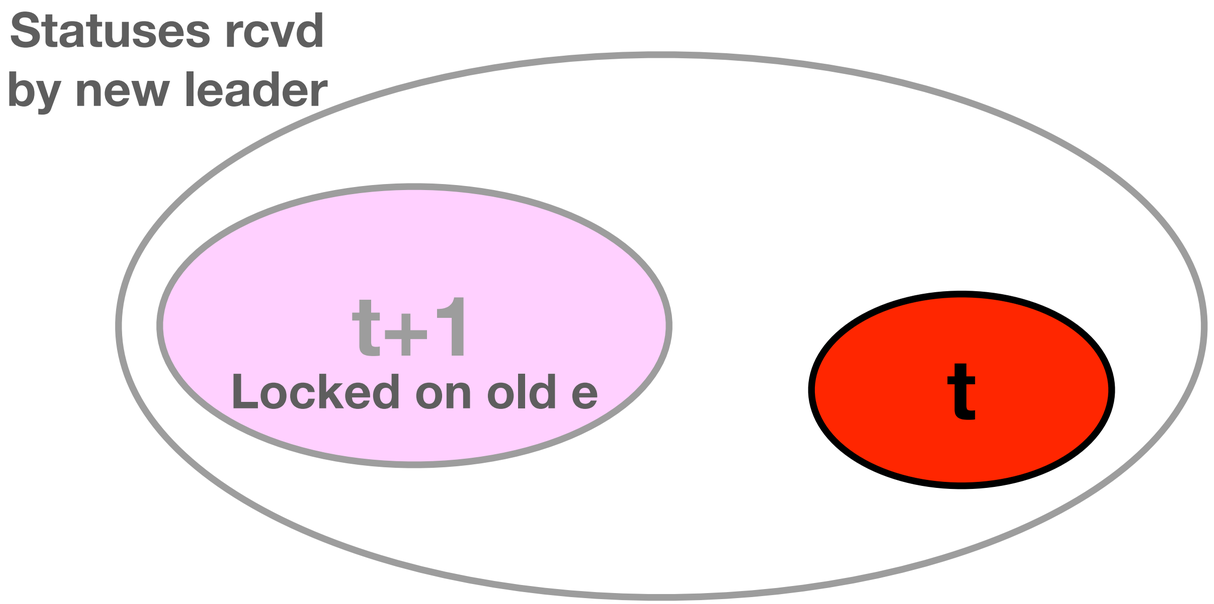
Why 2t+1?



Timeout -> view change

On receiving 2t+1 blames,
leader e+1 can request view change

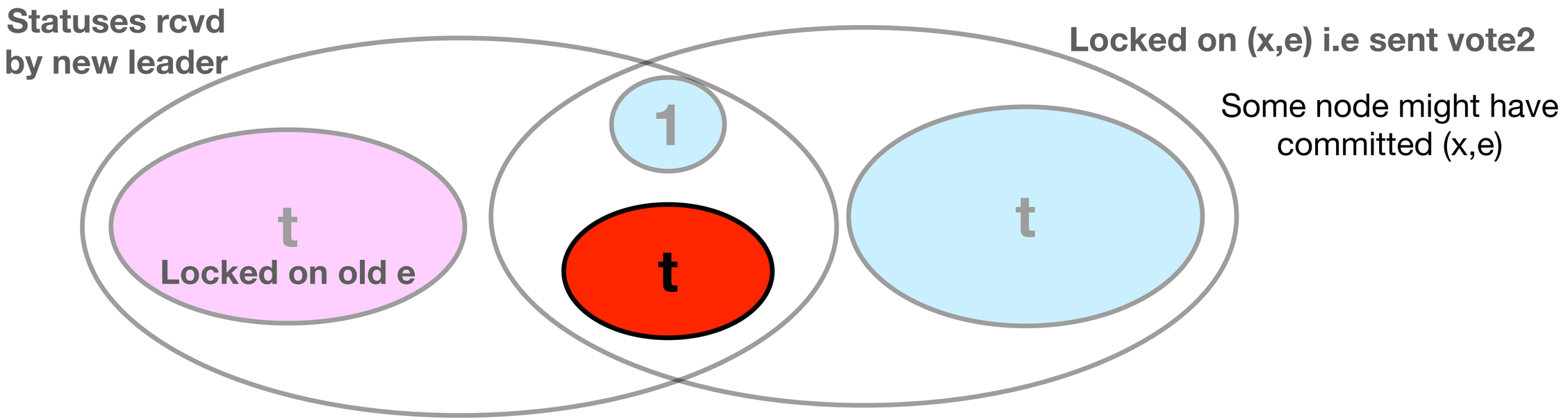
Why 2t+1?



Timeout -> view change

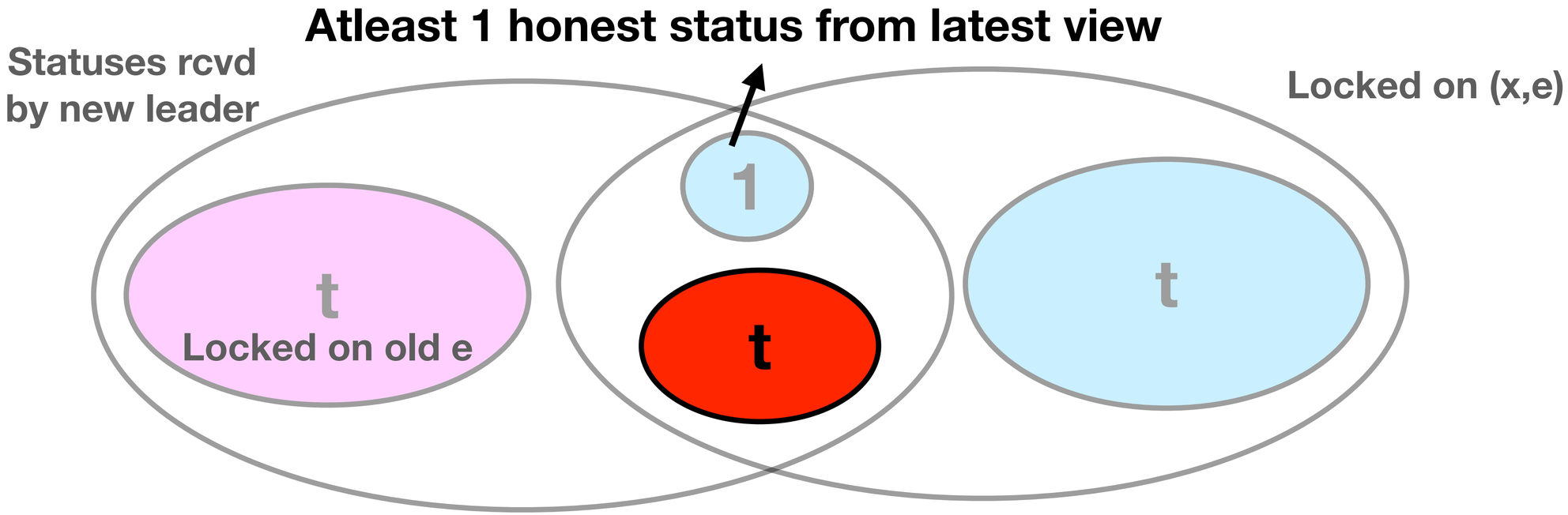
On receiving 2t+1 blames,
leader e+1 can request view change

Why 2t+1?



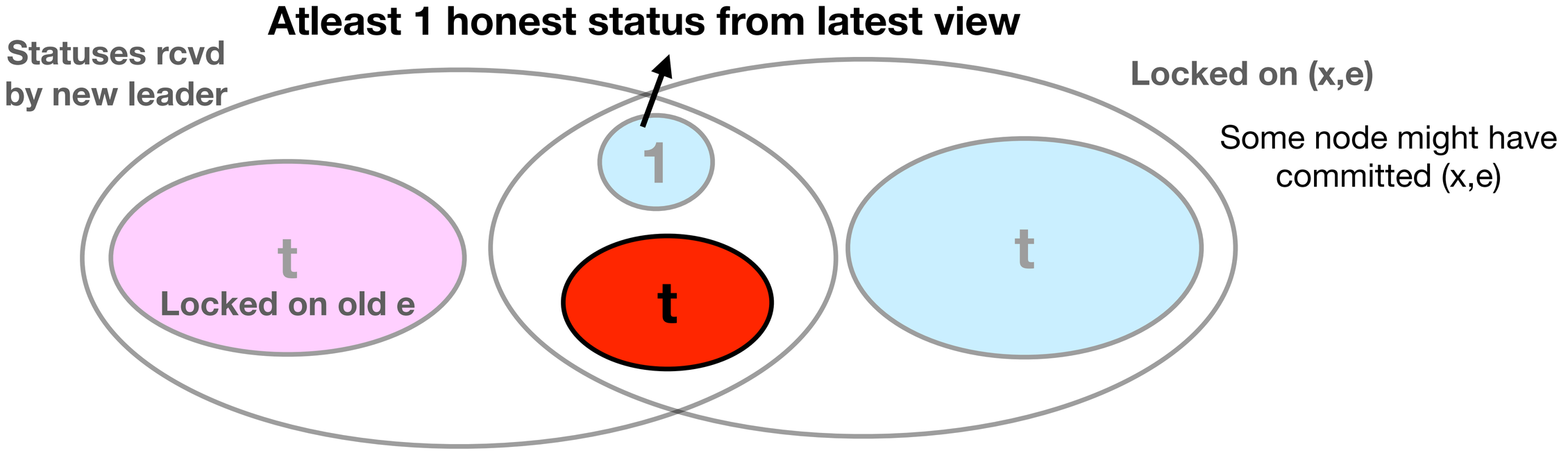
Timeout -> view change

On getting 2t+1 statuses, new leader sends proposes



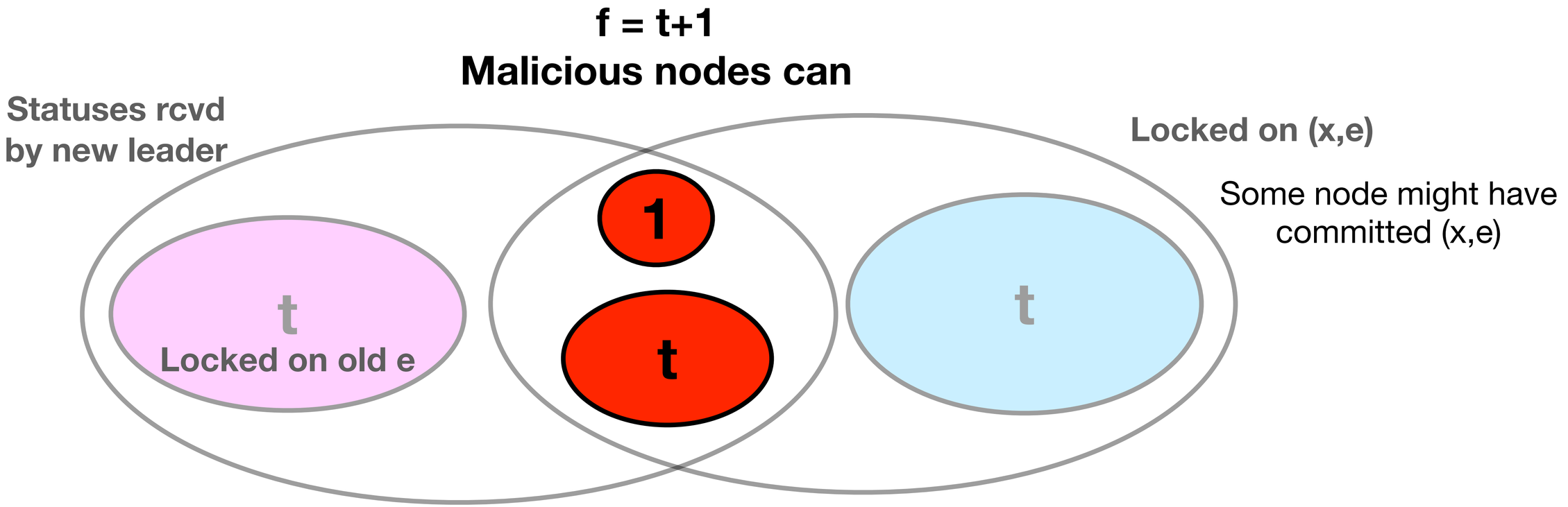
Timeout -> view change

This honest status ensures, new leader proposes same value that is locked in a previous view



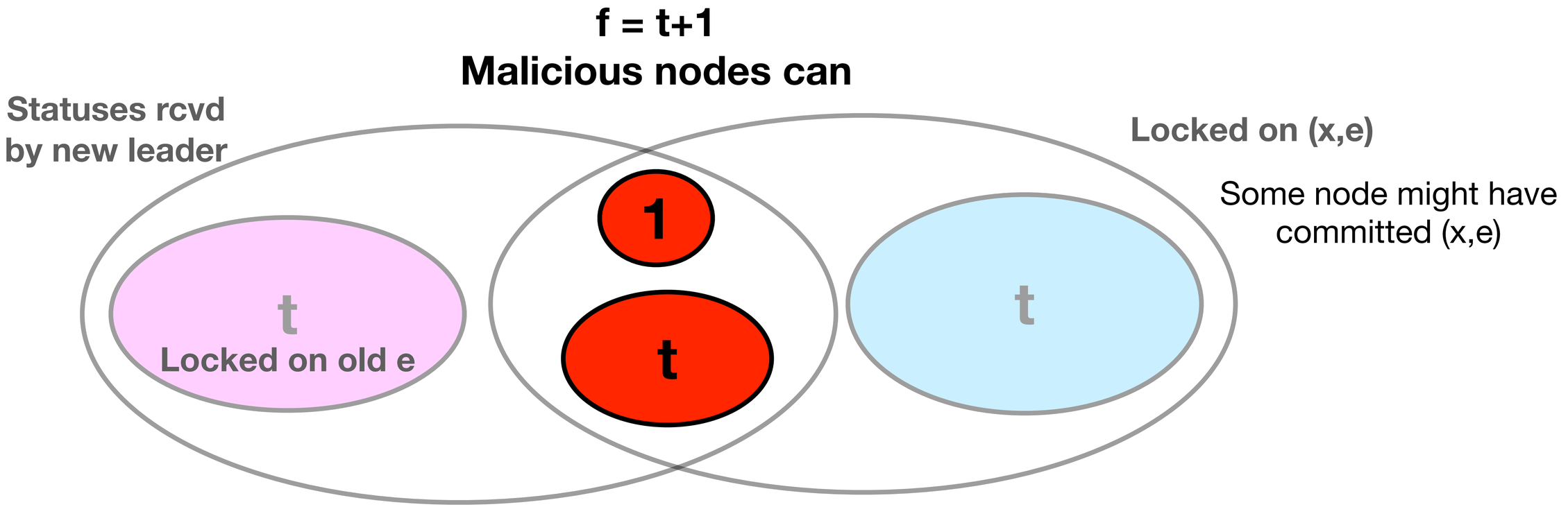
What can happen if f>t?

View change : What can happen if f>t?



View change : What happens if f>t?

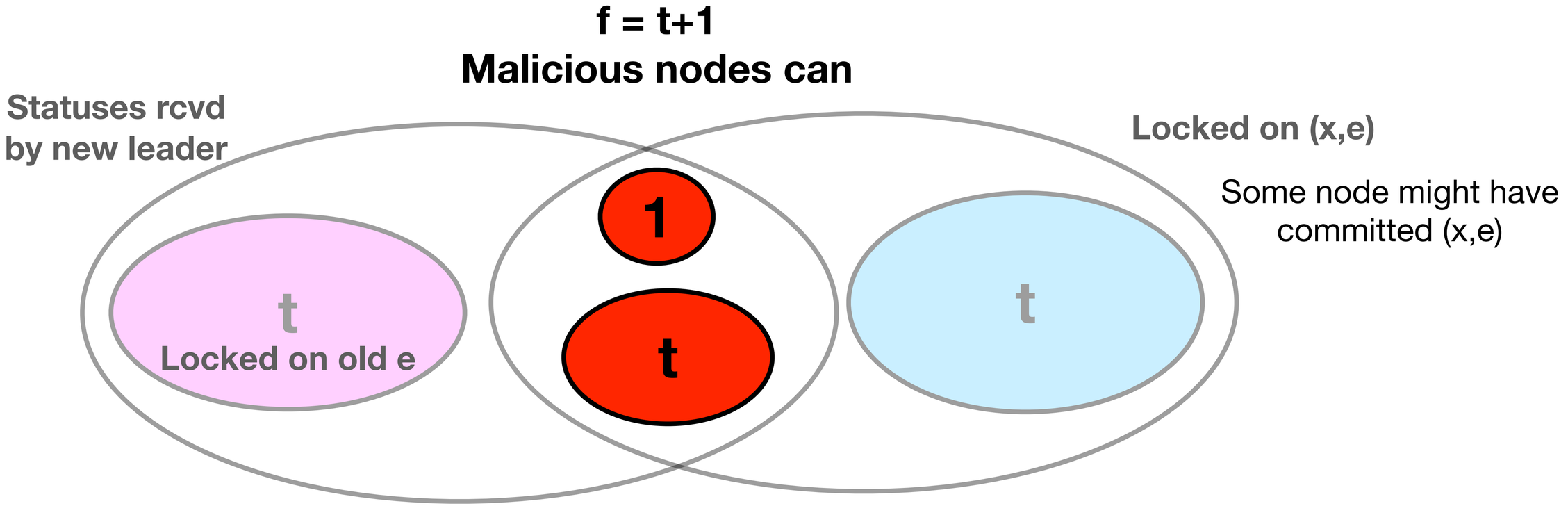
Malicious nodes can influence what a new leader picks



View change : What happens if f>t?

Malicious nodes can influence what a new leader picks

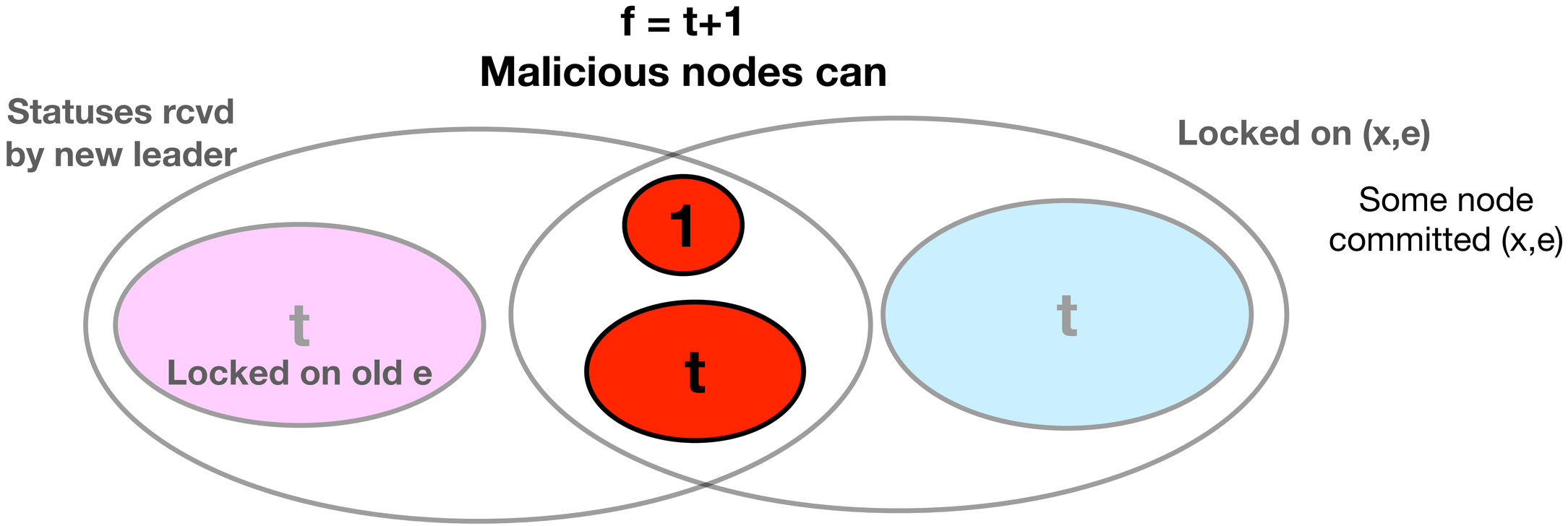
How?



View change : What happens if f>t?

How?

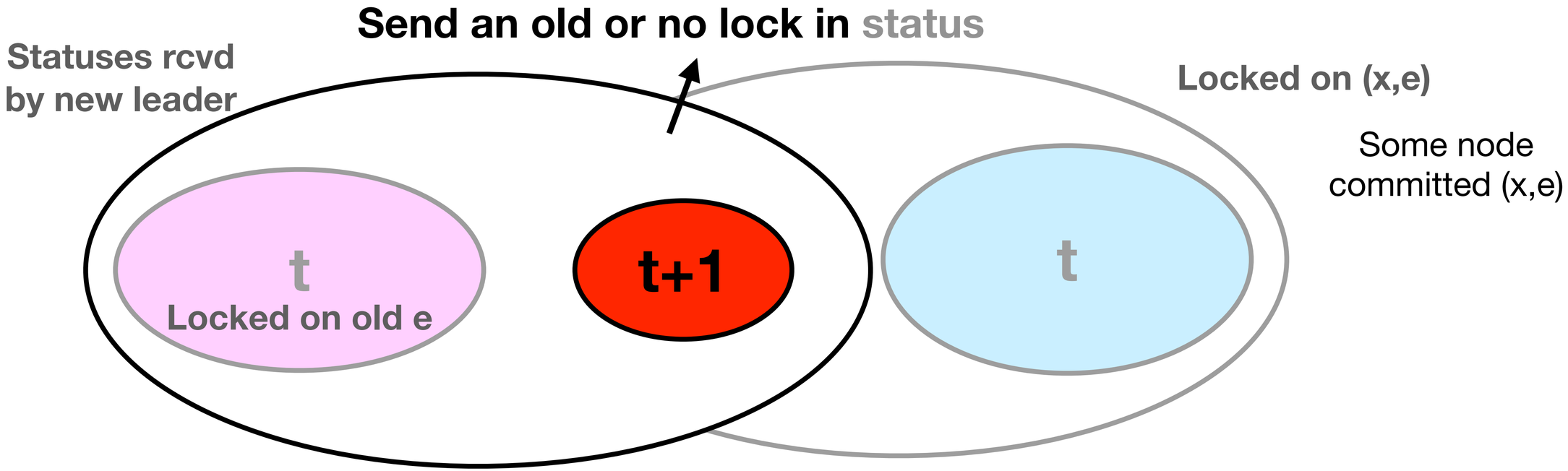
Ensure new leader doesn't see the most recent locked value



View change : What happens if f>t?

How?

Ensure new leader doesn't see the most recent locked value

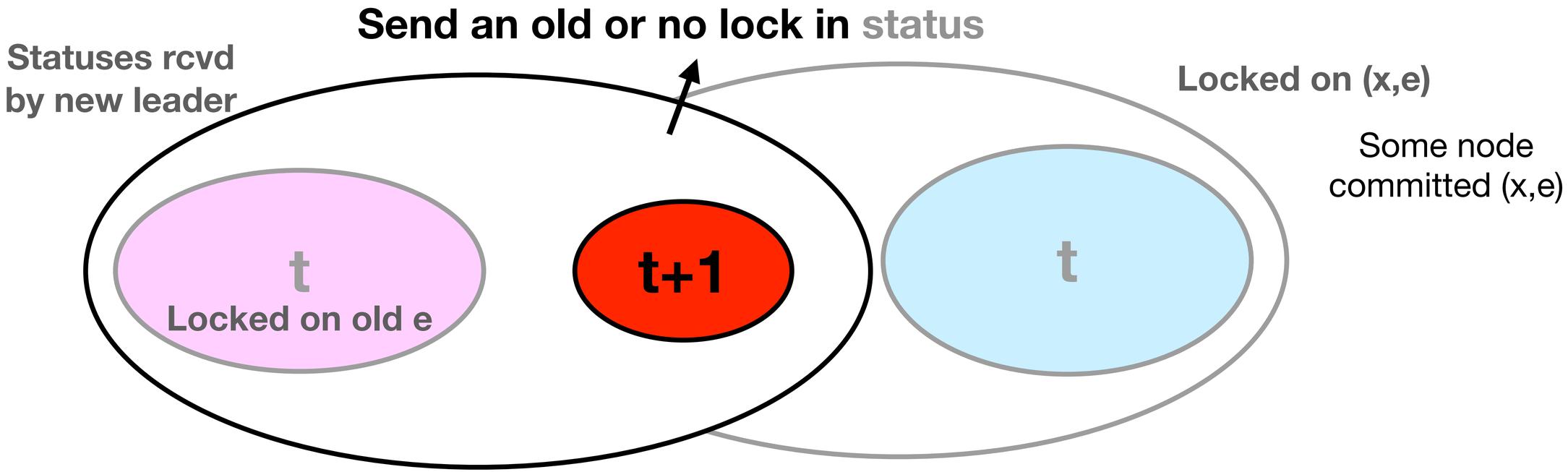


View change : What happens if f>t?

How?

Ensure new leader doesn't see the most recent locked value

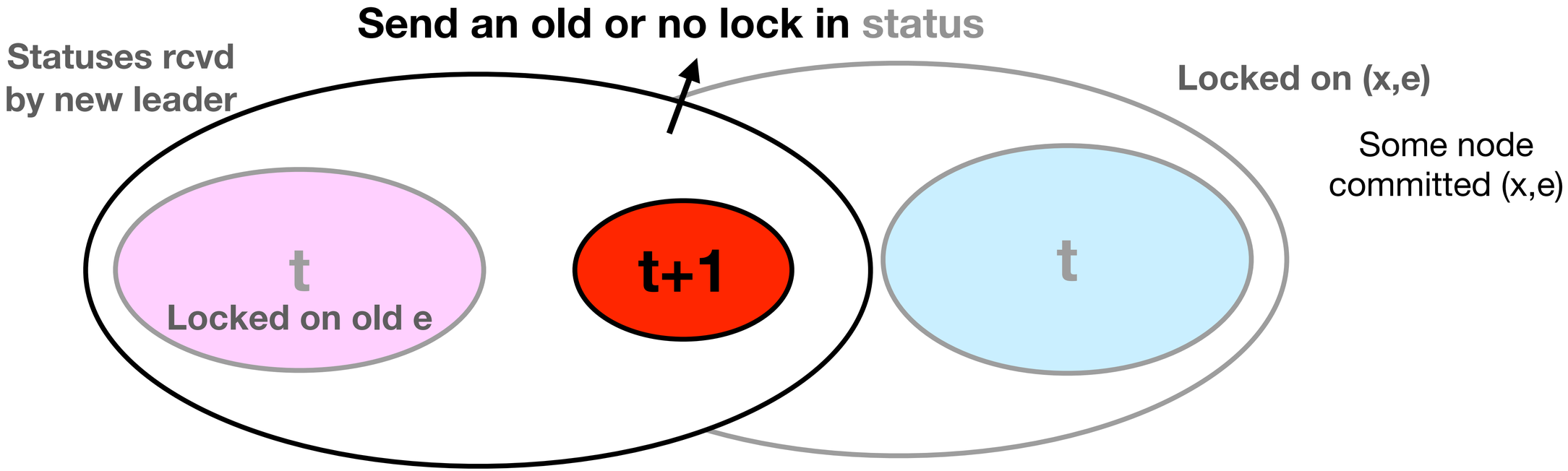
So e+1 leader picks an old lock x' or a new value (!= x)



View change : What happens if f>t?

So e+1 leader picks an old lock x’ or a new value

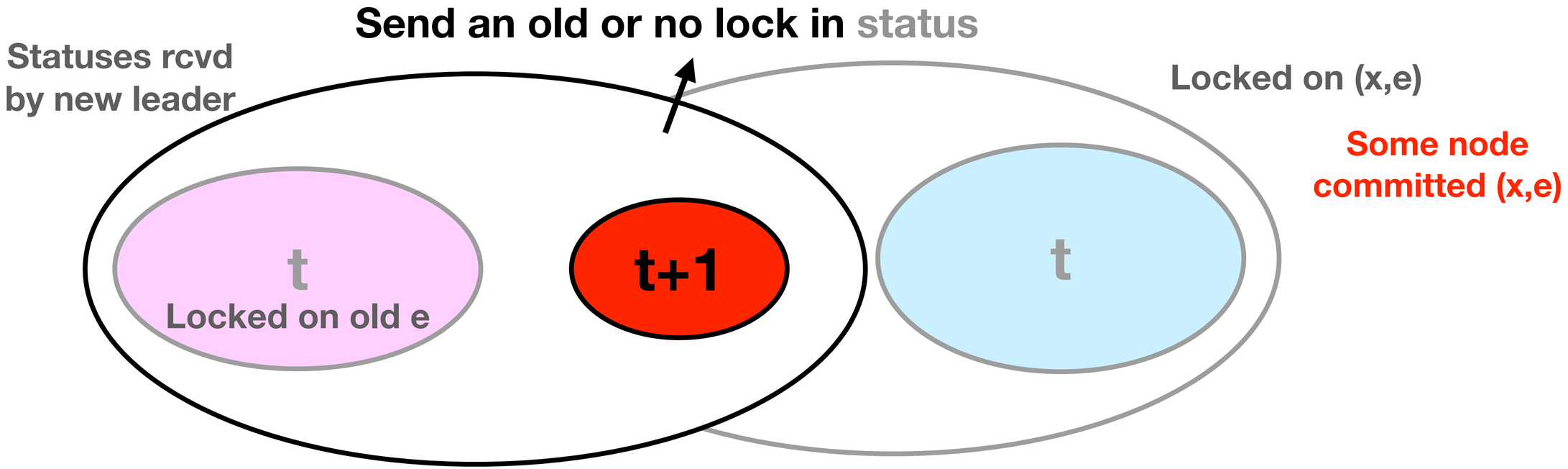
History is forgotten. Then some node can commit x’ in future



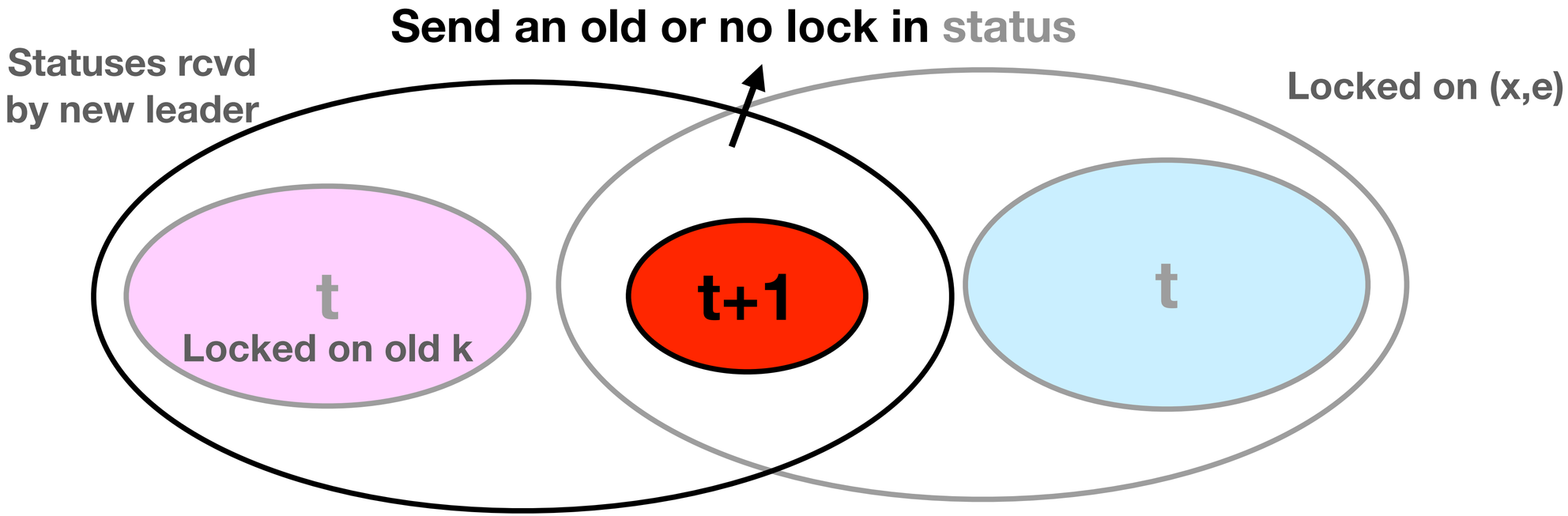
View change : What happens if f>t?

So e+1 leader picks an old lock v' or a new value

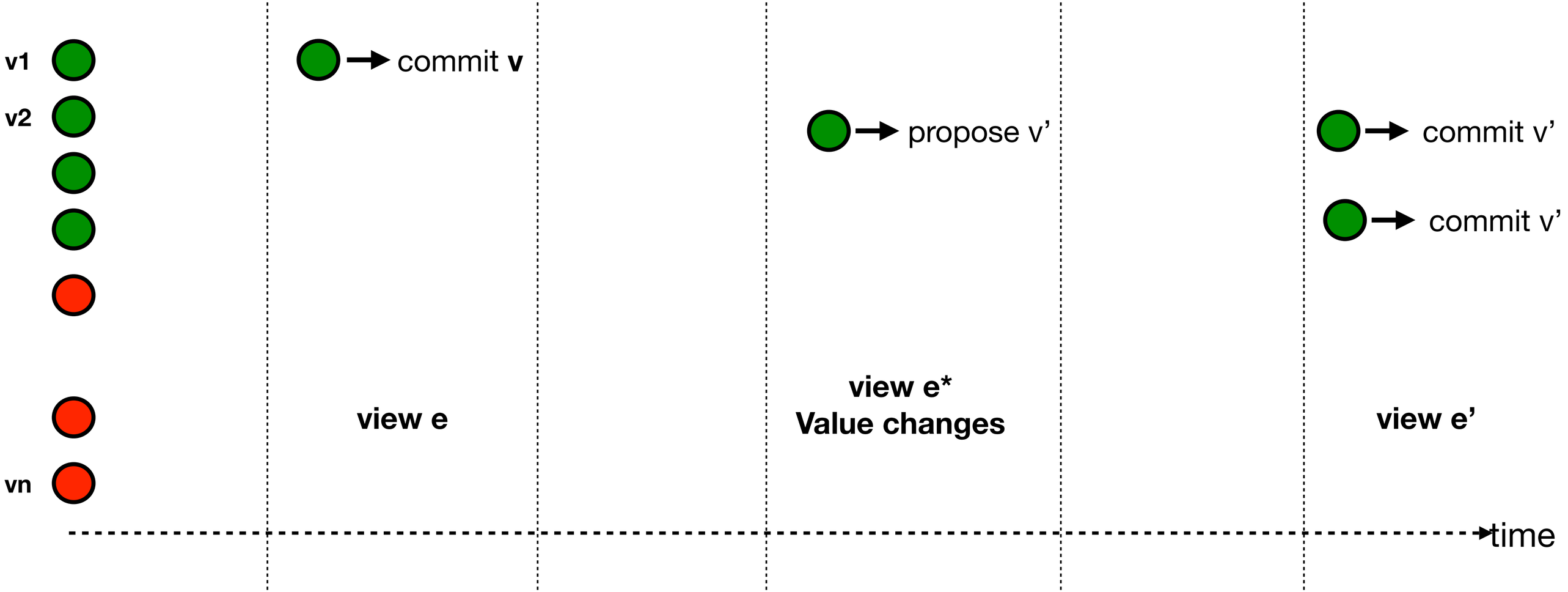
History is forgotten. **Then some node can commit x' in future**



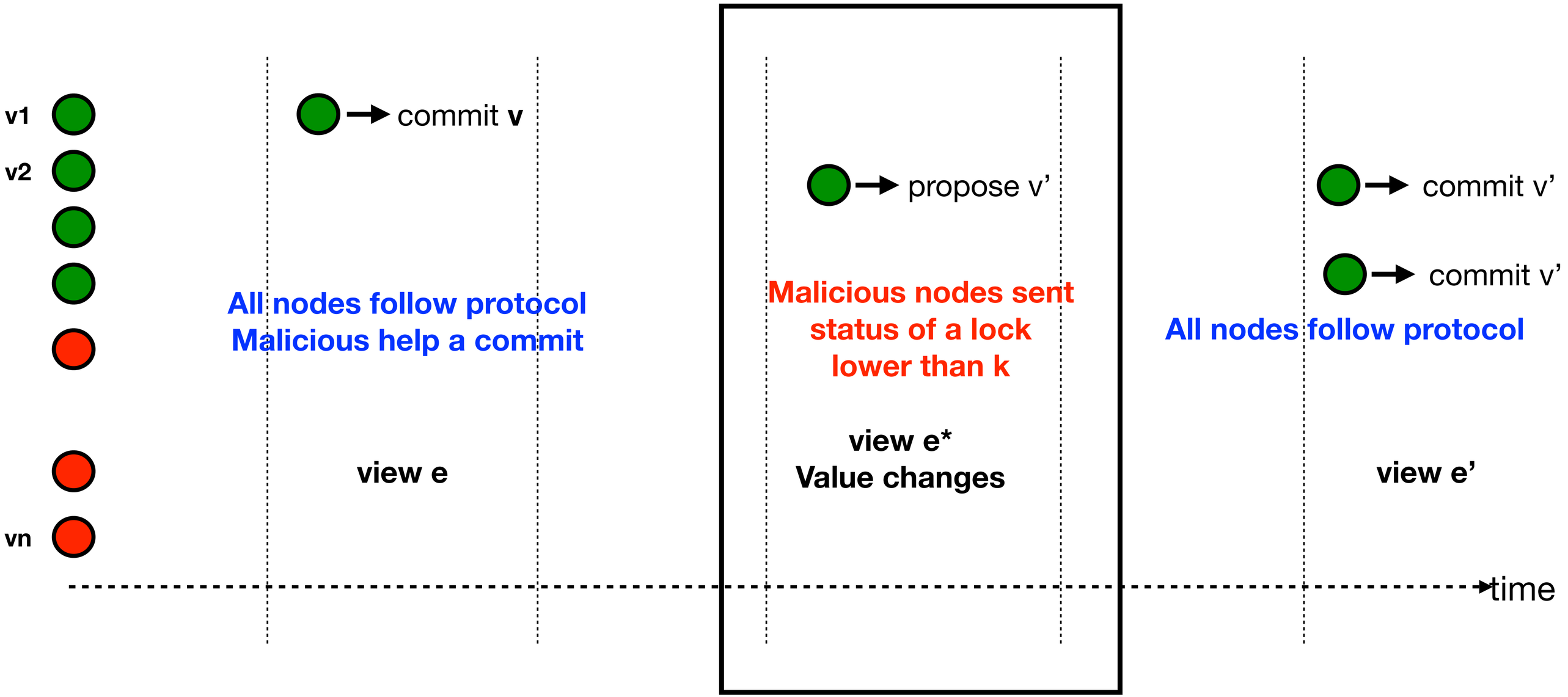
How to identify the culprits?



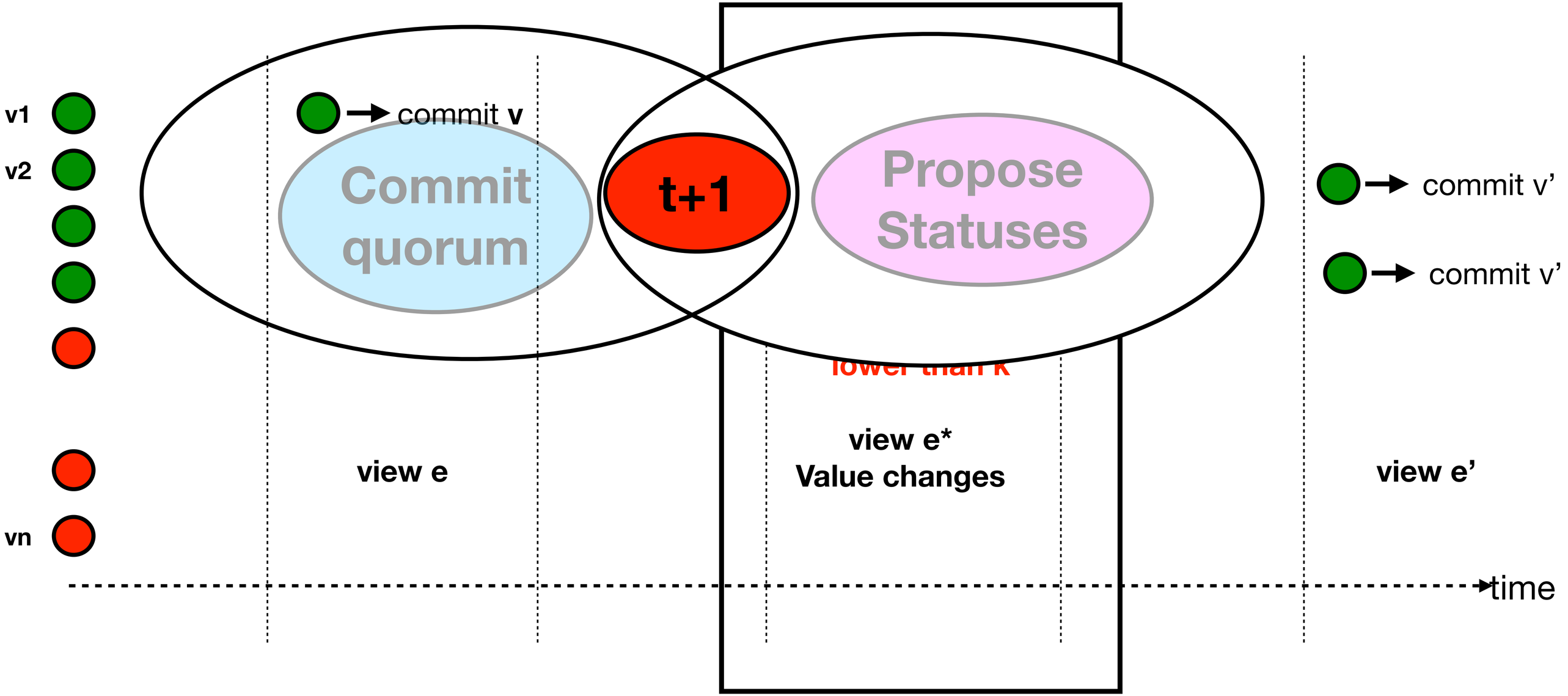
How to identify the culprits?



How to identify the culprits?



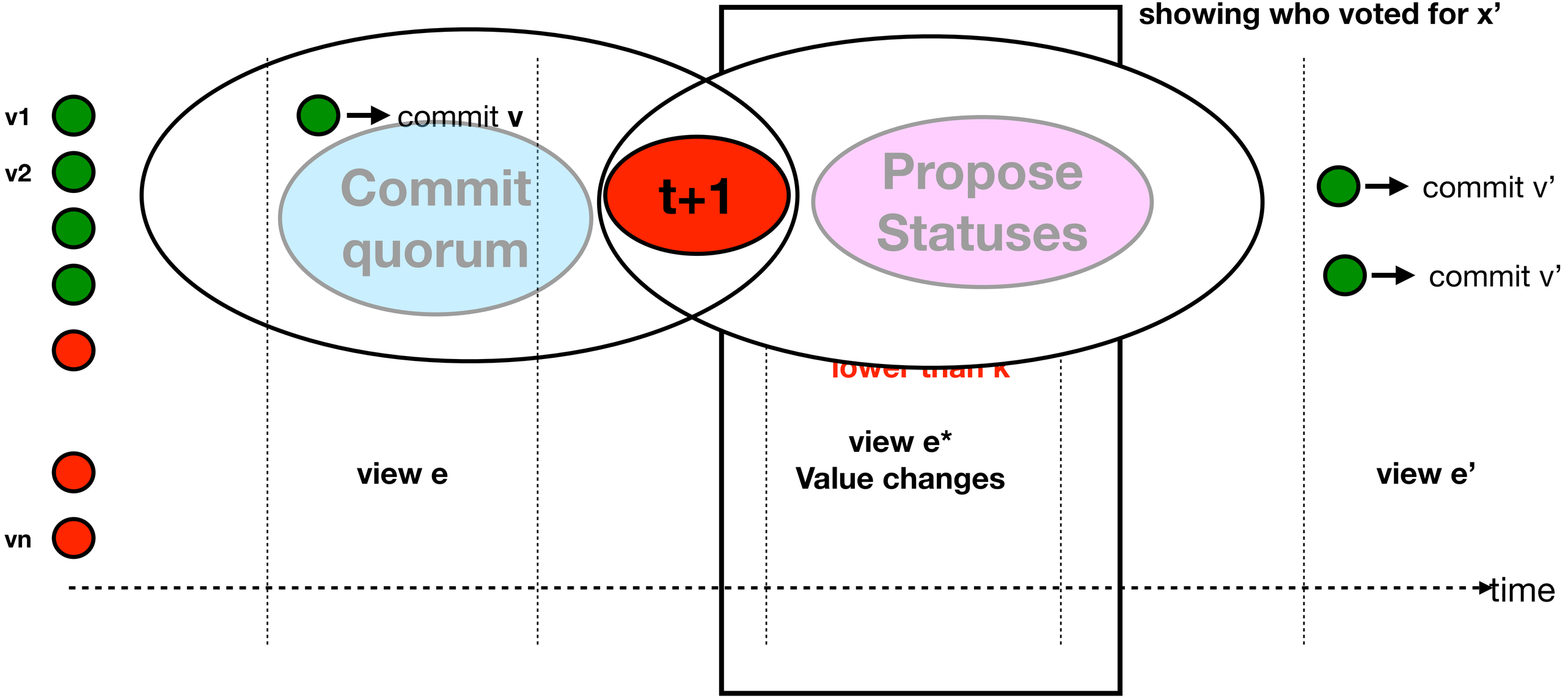
How to identify the culprits?



How to identify the culprits?

k = 1 since only one New propose msg in e* is enough

It contains status message showing who voted for x'



Some Thoughts

of nodes investigated

One commit msg and transcript of one other node proves fault

But how to find e*?

of nodes investigated

One commit msg and transcript of one other node proves fault

But how to find e*?

Although finally only one transcript is required, multiple nodes must be contacted to find e*

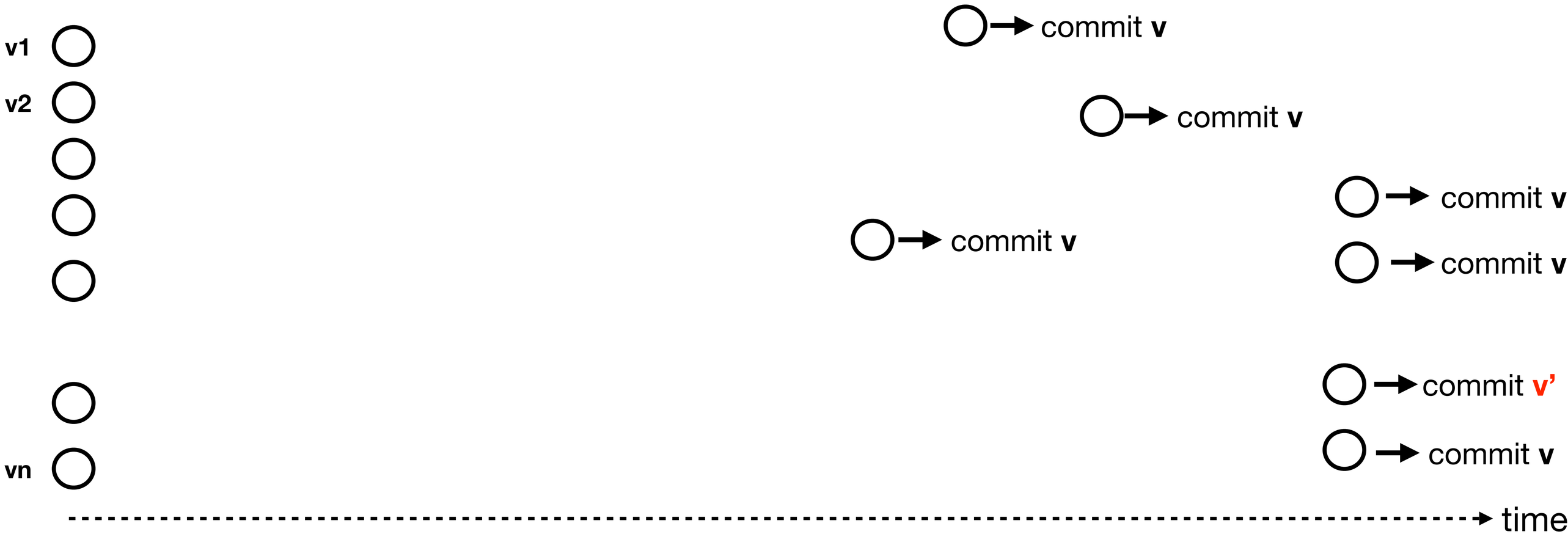
of nodes investigated

One commit msg and transcript of one other node proves fault

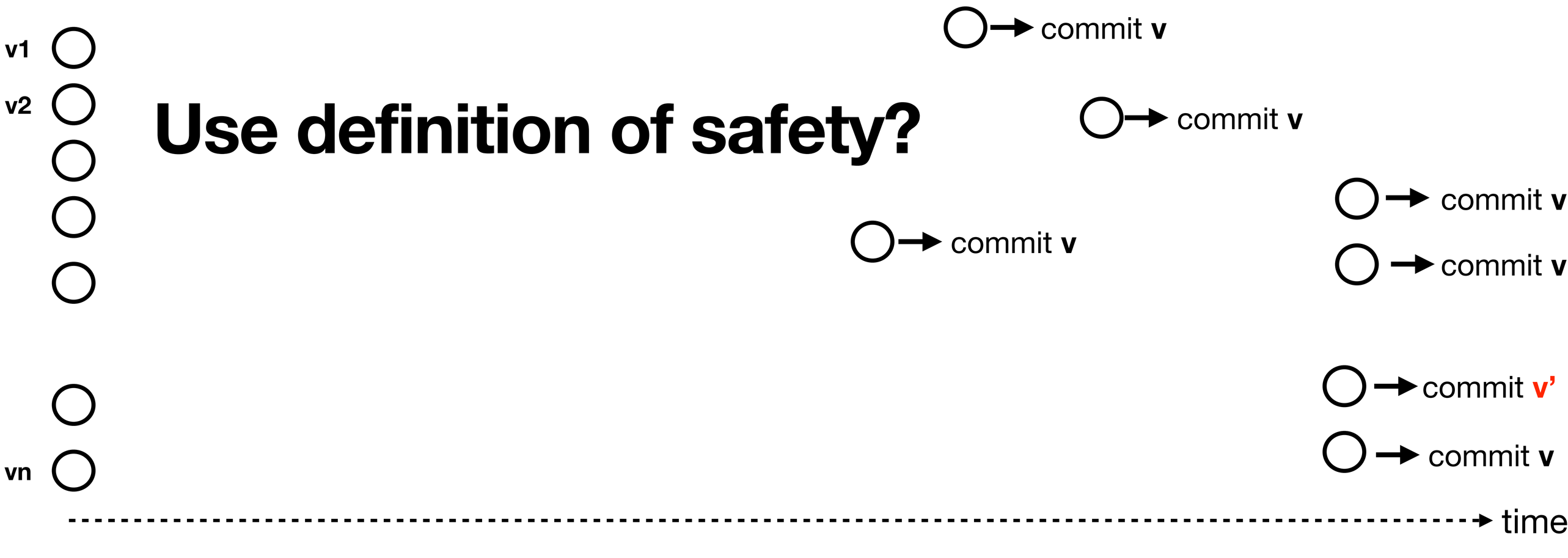
But how to find e*?

Ability to find e* also depends on what information is exactly included with votes

How to detect a Safety Violation?



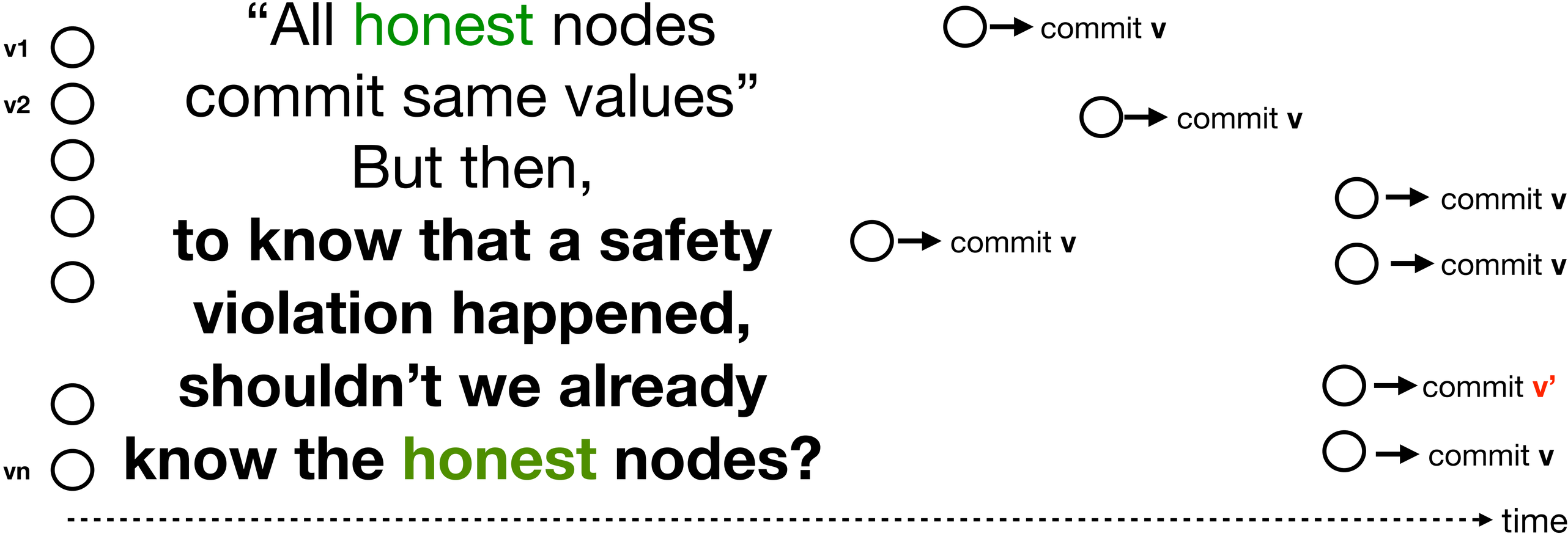
How to detect a Safety Violation?



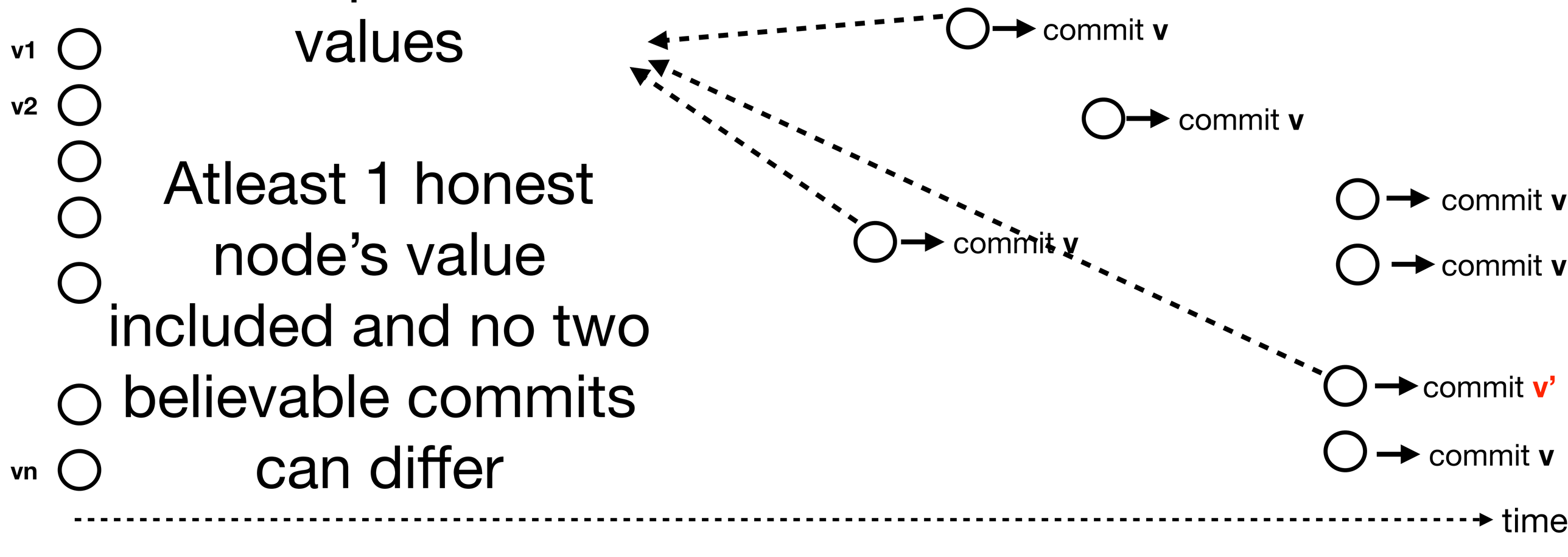
Use definition of safety?

How to detect a Safety Violation?

Safety says
"All honest nodes
commit same values"
But then,
to know that a safety violation happened, shouldn't we already know the honest nodes?



PBFT: Client uses weak certificate (t+1)



Source: PBFT, Castro's Thesis, Chapter 2

PBFT: How to modify?

Suppose, We have implemented PBFT

Now we are told f>t and safety violation is possible. We just need to detect it

What code change required? How many replies should client wait for?

Source: PBFT, Castro's Thesis, Chapter 2

PBFT: How to modify?

Suppose, We have implemented PBFT

Now we are told f>t and safety violation is possible. We just need to detect it

What code change required? How many replies should client wait for?

Wait for all?

Source: PBFT, Castro's Thesis, Chapter 2

What about liveness violation?

Byzantine nodes (f>t) can easily violate liveness by keeping quiet

Progress needs 2t+1

What about liveness violation?

Byzantine nodes (f>t) can easily violate liveness by keeping quiet

But can we identify such liveness violation and nodes that cause it?

What about liveness violation?

Byzantine nodes (f>t) can easily violate liveness by keeping quiet

But can we identify such liveness violation and nodes that cause it?

Large view number without commit indicates possible liveness issue. But can’t prove anything. Psync network = can’t distinguish slow vs dead

Thank you

Some impossibility results: Intuition

No forensic support for PBFT with f>=2t+1

Byzantine nodes can commit any value without involving honest nodes

No forensic support for PBFT with f>=2t+1

Byzantine nodes can commit any value without involving honest nodes

So they can cause safety violation, without leaving enough trace in honest nodes transcript

No forensic support for PBFT with f>=2t+1

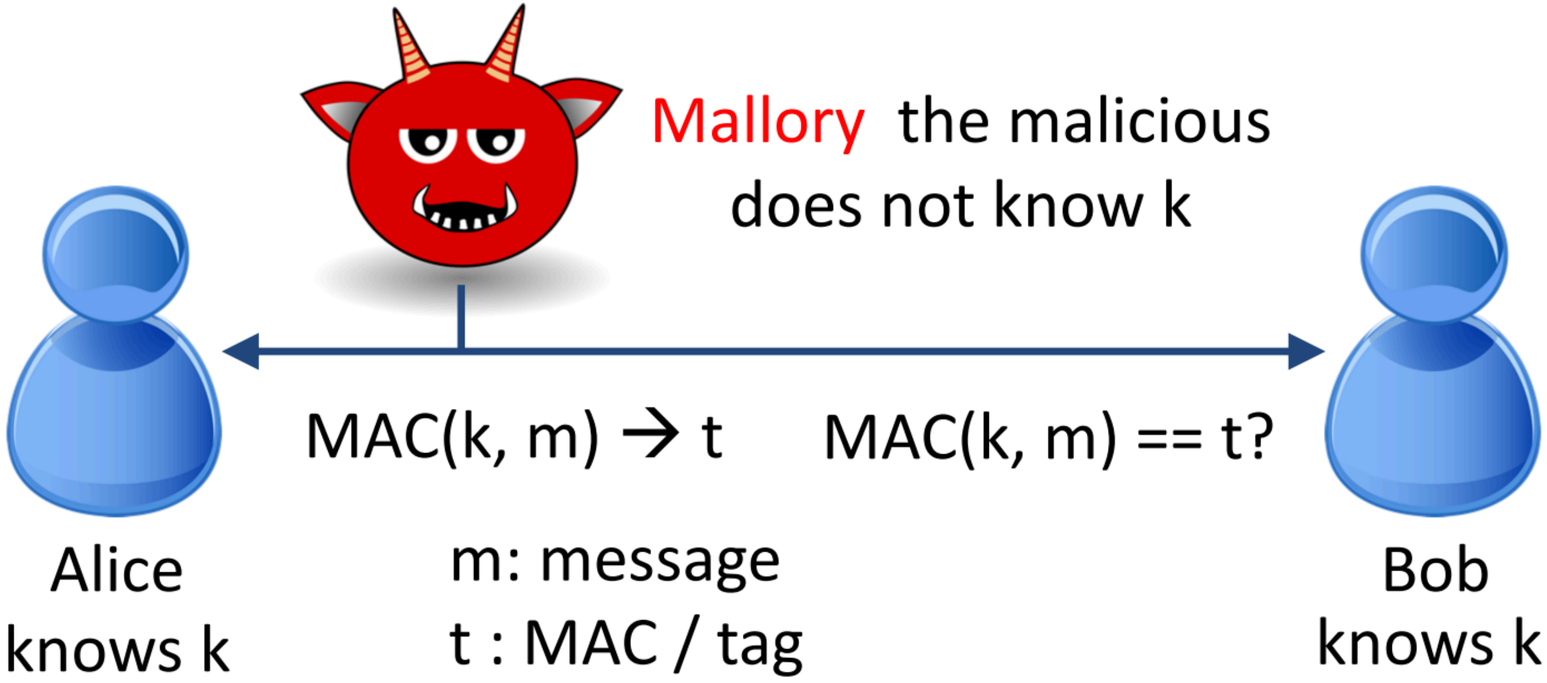
Byzantine nodes can commit any value without involving honest nodes

So they can cause safety violation, without leaving any/enough trace in honest nodes transcript

Proof: Standard way - Construct two worlds
Show that in both cases input to algo is same, but expected outputs are different.

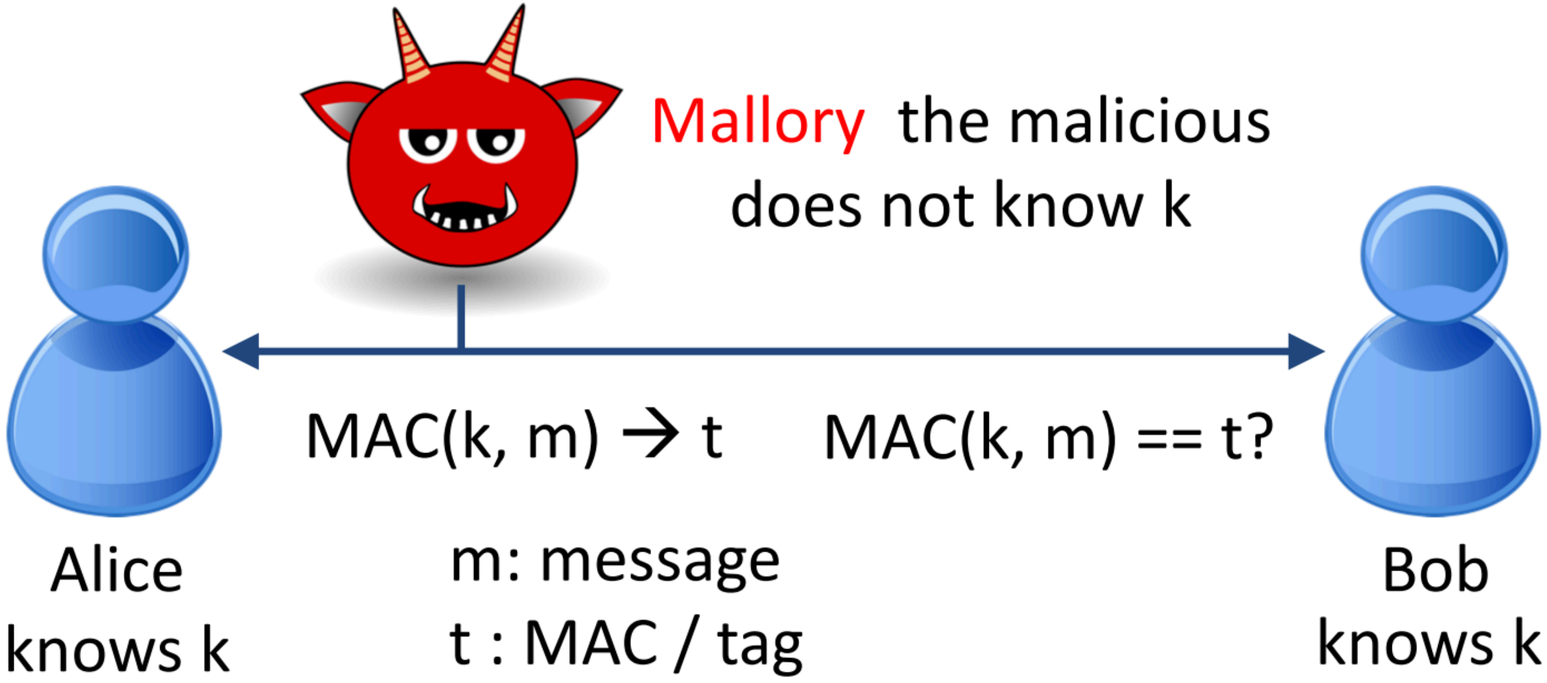
PBFT-MAC: No forensic support

Messages don’t have to be signed. Instead the channels are authenticated.



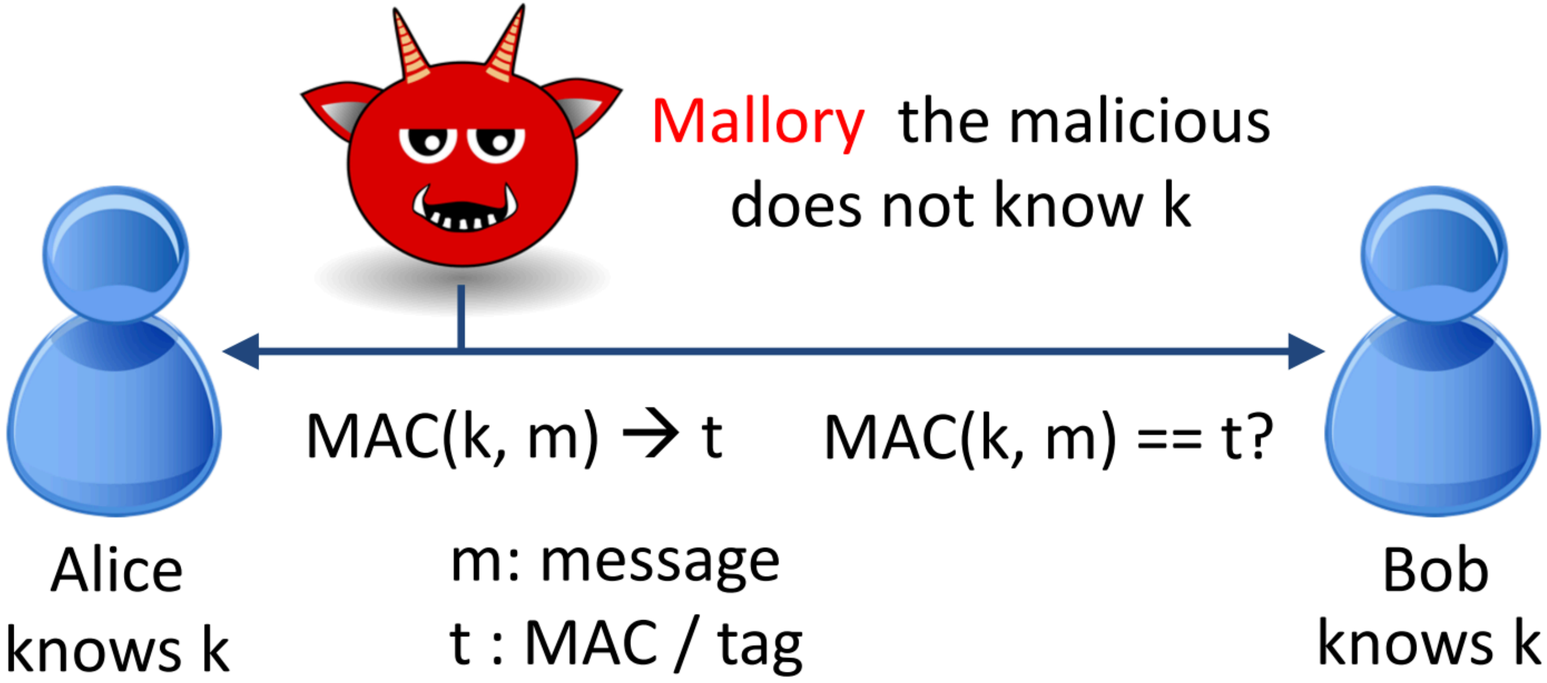
PBFT-MAC

No Forensics possible. Msgs can't be "forwarded"



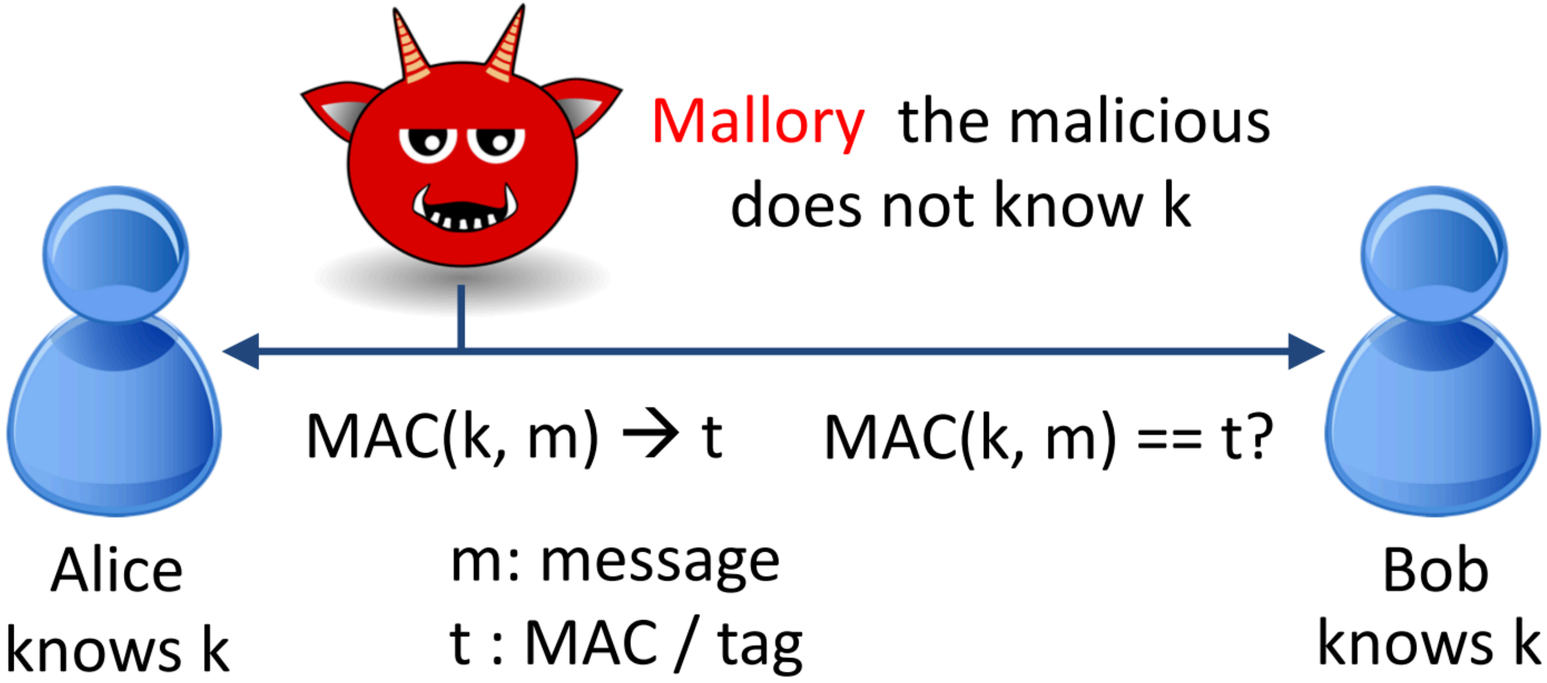
PBFT-MAC

No irrefutable proof possible.
“It’s my word against yours”



PBFT-MAC

No irrefutable proof possible.
“It’s my word against yours”



What have we not told you?

- Hostuff
- Algorand
- VABA
- Diem integration

References

- Sheng, Peiyao, et al. "BFT protocol forensics." Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. 2021.

- Castro, Miguel, and Barbara Liskov. "Practical byzantine fault tolerance." OsDI. Vol. 99. No. 1999. 1999.

- BFT Protocol Forensics, Kartik Nayak, https://www.youtube.com/watch?v=hSRK6PhBSjI

- BFT Protocol Forensics, Peiyao Sheng, https://www.youtube.com/watch?v=HZrCsvOnY2I

So, Safety implies…

Client can verify that
a committed value followed the protocol
(Faulty nodes can’t lie)

To cause a violation, Malicious nodes must cause two believable unequal commits

Source: PBFT, Castro’s Thesis, Chapter 2